Implementando um Tradutor de Linguagem Natural para a linguagem LEGAL

Sérgio Muinhos Barroso Lima

Tese de Mestrado

Implementando um Tradutor de Linguagem Natural para a linguagem LEGAL

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Sérgio Muinhos Barroso Lima e aprovada pela Comissão Julgadora.

Campinas, 20 de março de 1997.

Profa. Ariadne Maria Brito R. de Carvalho
Orientadora

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

Instituto de Computação
Universidade Estadual de Campinas

Tese de Mestrado defendida e aprovada em 27 de março de 1997 pela Banca Examinadora composta pelos Professores Doutores

Flávia de Almeida Barros.

Profª. Drª. Flávia de Almeida Barros

Prof°. Dr°. Geovane Cayres Magalhães

Ariadne M. B. R. Carvalho

Profª. Drª. Ariadne Maria Brito Rizzoni Carvalho

Prefácio

Nesta dissertação é apresentada a implementação de um tradutor de Linguagem Natural (LN) para a linguagem LEGAL, que é uma extensão da linguagem SQL que inclui operadores espaciais e facilidades para a manipulação de campos e objetos geográficos. O objetivo deste trabalho é auxiliar os usuários de um Sistema de Informação Geográfica, não especialistas em computação, na formulação de consultas através da utilização de LN. As vantagens e desvantagens da interação em LN são apresentadas, bem como a funcionalidade de cada módulo constituinte do tradutor; além disso, os problemas lingüísticos e as particularidades que as consultas espaciais possuem e que foram tratadas pelo tradutor são apresentadas.

Abstract

This dissertation presents the implementation of a Translator from Natural Language to LEGAL, which is an extension of the SQL language and which includes spatial operators to manipulate geo-fields and geo-objects. The goal of this dissertation is to help Geographical Information System users, who are not computer experts, on the query formulation, through the use of Natural Language. The advantages and disavantages of a Natural Language interaction, the Translator's modules, the linguistic problems encountered and the spatial query's particularities are presented.

Agradecimentos

Eu gostaria de agradecer aos meus pais, por tudo.
Aos meus irmãos, pela amizade e união.
À prof. Ariadne e à prof. Claudia pela orientação e pela oportunidade de trabalhar nessa área tão interessante.
Aos meus amigos João Paulo, Rosana, Jaqueline e Marcus pelo agradável convívio em Campinas.
À minha avó Maria, aos meus tios Pedro, Geni e Bina, e aos primos Marcelo e Aléssia pelo apoio.
Ao grupo de estudo de Bancos de Dados pelas contribuições a esse trabalho.
Ao Juliano pela revisão do texto.
Aos colegas do IC que me apoiaram.
Aos amigos Marcus, Maurício, Beavis, Luciana, Jane, Patrícia, Bino, Raquel, Dedéia, Lagran, Letícia, Liliane, Joyce e Aline pela amizade e bons momentos que passamos nesses dois anos.
Este trabalho foi desenvolvido dentro do projeto CNPq PROTEM/CC-GEOTEC, e financiado parcialmente pelo CNPq e pela CAPES.

Conteúdo

Lista de Figuras

Capítulo 1 - Introdução

1.1 - Motivação e Objetivo

A motivação para esta dissertação surgiu da necessidade de uma interface adequada à formulação de consultas por um usuário de um Sistema de Informação Geográfica (SIG).

A recuperação de dados está entre as mais importantes funções providas por um SIG. A função de recuperação de dados permite aos usuários de SIG buscar dados armazenados em bancos de dados para a sua análise, manipulação e exibição. São utilizadas, geralmente, linguagens de consulta espaciais para a recuperação de dados em SIG, tais como LEGAL [Cam95], Spatial SQL [Egen94], dentre outras, que são em sua maioria extensões do SQL padrão e que incluem operadores para dados espacialmente referenciados.

Sabe-se que a comunidade de usuários de um SIG é muito heterogênea (geógrafos, engenheiros, estatísticos etc.) e alguns deles o utilizam infreqüentemente [Smith89]. Portanto a interface para consultas em um SIG não pode ser projetada de forma a esperar que o usuário tenha conhecimentos de computação para operá-la. Logo, linguagens de consulta a bancos de dados em geral se mostram inadequadas, pois para utilizá-las o usuário necessita saber detalhes sobre a estrutura do banco de dados (entidades, relacionamentos e atributos), além de ter que se confrontar com sintaxes artificiais e às vezes de difícil aprendizado [Wang94].

Segundo [Wall84] as limitações das linguagens de consulta para bancos de dados podem ser eliminadas ou atenuadas da seguinte forma:

- estendendo-se essas linguagens e tornando o interpretador mais inteligente. Um interpretador inteligente deve reconhecer as solicitações do usuário mesmo em uma expressão de consulta incompleta (sentença em linguagem natural elíptica, por exemplo) ou com sutis erros ortográficos, como por exemplo:

"Quais são as cidad com população maior que 1000 habitantes?"

(erro ortográfico na palavra "cidad")

"E as com população menor que 1000?"

(sentença elíptica, isto é, o termo "cidades" foi omitido, mas compreendido através do contexto);

- dando-se mais importância à semântica dos vocábulos e menos à sintaxe. Assim, reduz-se a necessidade do usuário se adaptar a uma sintaxe artificial. Além disso, os vocábulos dessa linguagem devem ter um significado próximo ao seu correspondente em linguagem natural (LN), e um conjunto de vocábulos sinônimos deve ser oferecido; em outras palavras, deve-se oferecer ao usuário várias maneiras de se expressar uma consulta. O exemplo seguinte mostra como um mesmo objetivo pode ser alcançado de diversas formas através da utilização de sinônimos e regras de sintaxe diferentes.

"Recupere a área das cidades cuja população é maior que 1000 habitantes"

"Selecione a área das cidades com mais de 1000 habitantes"

"Qual é a área da cidade com população superior a 1000?";

- reduzindo a necessidade do usuário conhecer os detalhes sobre a estrutura do banco de dados. Palavras sinônimas às classes e atributos do esquema do banco de dados atenuam esse problema, como por exemplo:

"Quais são as **estradas** que cortam o rio Paraopeba?"

"Quais são as **rodovias** que cortam o rio Paraopeba?"

onde "estradas" e "rodovias" são usadas para referenciar uma mesma classe;

- formulando-se consultas incrementalmente, como por exemplo:

"Quais são as cidades cuja população é maior que 1000 habitantes?"

e as consultas complementares, tais como:

"E a altitude dessas cidades?".

Todas essas melhorias podem ser providas por uma interação em linguagem natural; segundo [EN94], a utilização de LN é um passo evolutivo na construção de interfaces para bancos de dados, como mostra a figura 1.1. Entretanto, o interpretador deve cobrir um vocabulário e um conjunto de regras gramaticais densos o suficiente para que um usuário formule as suas consultas com o menor esforço cognitivo possível.

A referência [HC90] detalha uma interface em linguagem natural (ILN) para a recuperação de informações textuais em bibliotecas; os autores comprovaram, através de experimentos práticos, que a maioria dos usuários foram capazes de conseguir resultados satisfatórios em um pequeno período de tempo após a demonstração do sistema, inclusive usuários que nunca haviam utilizado sistemas de consultas "*on-line*". Além disso, grande parte dos usuários acharam extremamente fácil o uso de LN na formulação das consultas. Segundo [Wang94], apesar das vantagens que a utilização de LN em SIG traria, pouco tem sido feito nessa área. Já [Jarke85] diz que embora haja um grande número de sistemas que tratam linguagem natural (exceto em relação à SIG), a viabilidade e vantagens da LN sobre os outros modos de interação continuam não provados, devido aos poucos estudos empíricos sobre os seus resultados práticos.

	Anos 60 a meados dos 70	Anos 70 a meados dos 80	Anos 80 a começo dos 90	Futuro
Interface para usuários	Nenhuma	Linguagens de consulta Formulários	Interfaces Gráficas Menus	Multimídia Linguagem Natural Entrada de dados por voz

Figura 1.1: A evolução da tecnologia de interfaces para bancos de dados.

Apesar das vantagens apresentadas anteriormente, a interação em linguagem natural também possui as suas desvantagens. Dentre elas, destacam-se [Cohen92, EN94 e ART95c]:

- opacidade da cobertura lingüística, isto é, o usuário sabe que o sistema não é capaz de reconhecer quaisquer frases, e não sabe precisamente quais o sistema pode interpretar;

- desconhecimento da cobertura do domínio, isto é, o usuário não sabe ao certo a amplitude do domínio coberto e, sendo assim, ele pode formular consultas sobre objetos que não existem no banco de dados. Algumas ILN para bancos de dados enviam mensagens aos usuários avisando-os de que a sentença contém entidades, ou atributos que não estão contidos no banco de dados. As meta-consultas (consultas sobre a estrutura do banco de dados) podem auxiliar o usuário na diminuição desse tipo de erro;

- a ambigüidade de certas sentenças, como por exemplo: "A menina viu o menino com o binóculo" que tanto poderia ser interpretada como o menino estando com o binóculo, ou a menina vendo o menino através do binóculo. Consultas referentes a dados geográficos podem gerar ainda outros tipos de ambigüidade. Por exemplo, a consulta "Quais as cidades que estão perto do rio Tietê" é ambígua, pois depende da noção de distância de cada usuário, isto é, para um certo usuário, "perto" pode ser algo em torno de um raio de 1 Km, enquanto que para outro usuário esse raio pode ser de 5 Km, e assim por diante. Segundo [Kaiser93 e Regier91] existem várias questões relativas a esse problema, como por exemplo: quais aspectos das relações espaciais estão envolvidos durante o raciocínio espacial? Como as pessoas descrevem essas relações em LN? Como a diferenciação de conceitos espaciais é influenciada pela linguagem nativa, pela cultura e por diferenças individuais de cada pessoa?

- atualmente as ILN representam um "*overhead*" computacional, pois elas requerem um banco de dados léxico específico, e o interpretador é um programa complexo e que pode fazer cair o desempenho na recuperação de dados.

Outro tipo de interface que vem sendo utilizada em bancos de dados são as Interfaces de Manipulação Direta, que apesar das suas inúmeras vantagens, são limitadas sob vários aspectos. Em particular, elas provêem pouco suporte para a identificação de objetos que não estejam na tela, para a especificação de relações temporais, dificuldades para identificar e operar um conjunto de entidades muito grande (que é característica da maioria dos bancos de dados) e dificuldades no uso do contexto da interação com o usuário. Por outro lado essas fraquezas são precisamente os pontos fortes da LN [Cohen92].

A junção desses dois modos de interação (manipulação direta e LN) em uma única interface, chamada Interface Multimodal (IM), resulta em uma interface mais poderosa que ambas isoladamente. As IM foram estimuladas em parte pelo desenvolvimento acelerado da computação multimídia, mas a tônica não é a disponibilidade de duas ou mais modalidades de comunicação, mas sim a integração de dois ou mais modos de interação, cada um suprindo as deficiências do outro e vice-versa [Cohen92].

Entretanto é importante não confundir interfaces multimídia com interfaces multimodais. O termo "mídia" diz respeito ao tipo do sinal usado na comunicação com o usuário (som, imagens etc.), enquanto que o "modo" é relativo às propriedades sintáticas, semânticas e pragmáticas desses sinais [Cohen92].

Considerando todos esses fatores, o objetivo desta dissertação é auxiliar o usuário não especialista em computação, implementando um protótipo experimental de um tradutor de linguagem natural para a linguagem LEGAL, linguagem desenvolvida no INPE e descrita em [Cam95], para que futuramente esse tradutor seja embutido em uma IM para SIG. Além disso, a implementação do tradutor ilustra os problemas e a potencialidade da interação em LN em relação a SIG.

Para alcançar tais objetivos, foi feito um estudo aprofundado sobre técnicas de processamento de linguagem natural, fenômenos lingüísticos, SIG e PROLOG. Além disso, a funcionalidade e as características principais de várias ILN para bancos de dados foram analisadas.

1.2 - Estrutura da Dissertação

O restante desta dissertação está assim organizado:

Capítulo 2 - Processamento de Linguagem Natural, Sistemas de Informação Geográfica e LEGAL: esse capítulo é relativo às principais áreas de conhecimento envolvidas neste trabalho. Conceitos básicos dessas áreas e relevantes a esse trabalho são introduzidos nesse capítulo;

Capítulo 3 - O Tradutor: esse capítulo detalha a implementação do tradutor de LN para LEGAL. Os seus módulos constituintes são analisados, bem como os problemas lingüísticos enfrentados pelo tradutor;

Capítulo 4 - Consultas Espaciais: nesse capítulo são apresentadas as particularidades que as consultas espaciais impõem e que devem ser tratatas pelo tradutor;

Capítulo 5 - Conclusões e Trabalhos Futuros: capítulo destinado às conclusões acerca da pesquisa desenvolvida e detalhada nesta dissertação, e sobre os trabalhos futuros que deverão ser desenvolvidos para aperfeiçoar o tradutor para a construção futura de uma interface multimodal.

Capítulo 2 - Processamento de Linguagem Natural, Sistemas de Informação Geográfica e LEGAL

Este trabalho envolve duas áreas principais: processamento de linguagem natural e sistemas de informação geográfica. Conceitos básicos dessas duas áreas são apresentados neste capítulo, bem como uma introdução à linguagem de consulta LEGAL, linguagem para a qual a expressão de consulta em LN é convertida. Uma aplicação exemplo também é mostrada, sobre a qual algumas consultas possíveis de serem formuladas por um usuário foram estudadas e convertidas para LEGAL.

2.1 - Processamento de Linguagem Natural

2.1.1 - Histórico

Segundo [BF81], as pesquisas em Lingüística Computacional começaram a partir do momento em que os computadores se tornaram disponíveis, na década de 40. Em 1949, Warren Weaver propôs que os computadores poderiam ser utilizados para a tradução automática de textos de uma linguagem natural para outra (japonês/inglês, por exemplo). Nessa época, os sistemas de tradução automática de textos simplesmente convertiam as palavras de uma linguagem para outra (através de um dicionário) e organizavam as palavras convertidas de acordo com a linguagem destino. Apesar da atrativa simplicidade da idéia, tanto na escolha de palavras correspondentes, como na organização dessas palavras para a

formação do texto de saída (que não considerava a formação sintática da sentença de entrada), os resultados não foram satisfatórios, e tal metodologia foi abandonada [BF81].

As teorias lingüísticas introduzidas por Noam Chomsky em 1957 influenciaram radicalmente as pesquisas em lingüística. Chomsky mostrou a importância da gramática na compreensão da LN. Esse avanço teórico, juntamente com o advento das linguagens de programação de alto nível e a melhoria da tecnologia dos computadores (processamento e capacidade de memória), nos anos 60, possibilitaram um grande avanço no processamento de linguagem natural. A linguagem humana começou a ser compreendida como uma complexa habilidade cognitiva, envolvendo conhecimentos de diferentes tipos: a estrutura gramatical das sentenças, o significado das palavras, as regras de conversação, o conhecimento sobre o assunto em foco, dentre outros. Nessa época surgiram os primeiros sistemas práticos que tratavam linguagem natural, sendo porém muito limitados. Tais sistemas procuravam por palavras chaves na sentença de entrada, ignorando a sua estrutura sintática [BF81].

Na década de 70 foram desenvolvidos analisadores sintáticos mais eficientes, como as Redes de Transição Ampliadas [Woods70], que serviram de base para alguns sistemas, como o LUNAR [BF81], por exemplo.

As Gramáticas de Cláusulas Definidas [PW80], que são um formalismo para a descrição de linguagens (com o mesmo poder das Redes de Transição Ampliadas), foram introduzidas por Fernando C. N. Pereira e David H. D. Warren no fim da década de 70 e, juntamente com a linguagem de programação lógica PROLOG, que foi originalmente desenvolvida para o processamento de linguagem natural, deram um novo impulso na implementação de sistemas, como o TUGA [Coelho79]. Nessa época também surgiram sistemas mais sofisticados e flexíveis, como o sistema LIFER [HSS+78], por exemplo. Alguns dos sistemas dessa época passaram a permitir a reconfiguração de alguns dos seus módulos para que tais sistemas pudessem ser utilizados em aplicações diferentes [Tedes90].

Atualmente, em relação ao uso de LN em interfaces para bancos de dados, percebe-se uma tendência à utilização de LN em interfaces multimodais, como o ambiente LINX [García95], por exemplo, que é uma interface multimodal para sistemas de informação baseados em conhecimento, e o sistema Shoptalk [Cohen92], que é um ambiente de suporte a decisão para o monitoramento do controle da qualidade. O motivo dessa atual tendência talvez seja a síntese produtiva que a LN e a manipulação direta fornecem quando unidas.

Segundo [EN94], a tecnologia de reconhecimento e compreensão da fala tem um enorme potencial para que, no futuro, as "portas" dos bancos de dados sejam abertas para pessoas de todas as idades e níveis de sofisticação.

2.1.2 - Conceitos Básicos

Existem duas motivações principais para a pesquisa em lingüística computacional: primeiro, a motivação tecnológica para se construirem sistemas inteligentes usando linguagem natural em **interfaces para bancos de dados**, sistemas de tradução automática (português/inglês, por exemplo), sistemas de análise de textos, sistemas de reconhecimento da fala, dentre outros. A segunda é obter-se um melhor entendimento de como as pessoas se comunicam através da linguagem natural [Allen87].

O objetivo dessa ciência é especificar teorias para a compreensão da linguagem e a produção de formalismos que permitam o desenvolvimento de programas de computador que possam entender e produzir linguagem natural [Allen87]. As sub-áreas típicas desse campo incluem a especificação de algoritmos para análise de linguagem natural, o estudo das suas propriedades computacionais, a construção de formalismos para a representação do conhecimento que possam suportar a análise semântica das sentenças e a modelagem de situações onde o contexto interfere na interpretação das sentenças.

Um programa para compreensão de uma linguagem natural (inglês, francês, português etc.) deve possuir um conhecimento considerável sobre a estrutura da linguagem a ser analisada, incluindo um dicionário de palavras e as regras gramaticais que geram as

combinações de palavras para formar sentenças. Entretanto, é importante frisar que tais programas não tratam uma determinada LN em toda a sua extensão. Devido à complexidade de tais linguagens, esses programas tratam somente subconjuntos de LN relativos ao domínio da aplicação; quanto menor e mais regular esse subconjunto, mais fácil será o seu tratamento.

Geralmente, um sistema de processamento de linguagem natural possui as seguintes partes: o analisador, o vocabulário, o interpretador, a base de conhecimentos e o gerador de respostas [Tedes90]. A sentença de entrada é inicialmente processada por um analisador, que é um programa que, dadas as regras da gramática e um conjunto de palavras (contidas no vocabulário), retorna a estrutura sintática da sentença, desde que a sentença esteja de acordo com as regras estabelecidas. O conhecimento sobre a estrutura gramatical de uma sentença é crucial para o entendimento do seu significado [MG89]. O interpretador consulta a base de conhecimentos sobre informações relativas ao domínio da aplicação e, com base nessas informações, realiza inferências com os ítens semânticos extraídos da estrutura gramatical construída pelo analisador. No caso de sentenças de entrada declarativas o interpretador atualiza, se necessário, a base de conhecimentos. O gerador de respostas, como o próprio nome diz, produz respostas para as sentenças de entrada. Essas respostas variam de acordo com a finalidade do sistema. Em sistemas de tradução automática de textos, a resposta é também um texto, entretanto em uma LN diferente da original (inglês/português, por exemplo). Já em tradutores de LN para consultas a bancos de dados, a resposta é uma consulta em uma linguagem de consulta para bancos de dados (SQL, por exemplo) e, nesses casos, a geração de respostas é mais fácil em comparação à geração de respostas em sistemas de tradução automática de textos, já que a complexidade de um texto em LN é maior quando comparada a uma expressão de consulta em uma linguagem de consulta a bancos de dados.

O desenvolvimento de programas que tratam linguagem natural não é tarefa fácil, principalmente os que envolvem subconjuntos de linguagem natural muito complexos, como os sistemas de tradução automática de textos. Nesses casos, o programa analisador necessita cobrir um extenso vocabulário e um número grande de regras gramaticais. Além disso, a resolução de problemas lingüísticos, como a ambigüidade, a referência pronominal e o tratamento do contexto se tornam problemáticos.

Segundo [Wall84], bancos de dados são os sistemas que melhor se adequam à utilização de LN, pois de acordo com [LS93], embora bancos de dados armazenem um enorme volume de dados, a semântica desse conjunto de dados é regular e bem definida, o que facilita o projeto e a implementação de tradutores de LN para linguagens de consulta a bancos de dados. Talvez por isso, os programas que tratam LN sejam utilizados principalmente para consultar bancos de dados, nos vários tipos de sistemas e aplicações. [And92] e [ART93] descrevem o uso de ILN para aplicações convencionais em bancos de dados relacionais; o uso de ILN para bancos de dados temporais é descrito em [And96], [ART95a] e [ART95b]; [HSS+78] mostra o sistema LIFER (Language Interface Facility with Ellipsis and Recursion) que é aplicado a um banco de dados sobre marinha (navios, submarinos, portos etc.); o sistema TUGA utilizado em consultas bibliográficas é descrito em [Coelho79]; o uso de LN em uma interface multimodal para sistemas de informação baseados em conhecimento é mostrado em [García95]; e [BF81] mostra o sistema LUNAR que permite consultas em LN a dados sobre rochas, composição do solo, e outras informações coletadas pela missão Apollo 11. Em relação a SIG, como foi dito na motivação, pouco tem sido feito em direção ao uso de LN, apesar das vantagens que a LN traria.

As ILN também são utilizadas em sistemas operacionais [LX92], jogos e sistemas especialistas.

2.2 - Sistemas de Informação Geográfica

2.2.1 - Histórico

Os Avanços tecnológicos em Computação, Cartografia e Fotogrametria nos anos 40 e 50 foram a base para o desenvolvimento dos SIG. Os primeiros SIG surgiram na década de 60, tais como o Sistema de Informações Geográficas Canadense (1964) e o Sistema de Informações de Recursos Naturais e Uso do Solo de Nova York (1967) [Smith89]. Os sistemas desse período tinham como principal objetivo a produção de mapas, e como característica a não necessidade de guardar informações topológicas, sendo as informações gráficas e contínuas. Esses sistemas tiveram pouco êxito devido às limitações tecnológicas daquela época [Vasco96].

Nos anos 70 houve um rápido crescimento no número de SIG, possibilitado pelo avanço tecnológico na computação (E. F. Codd propôs o modelo relacional em 1970 e causou um grande salto na tecnologia de bancos de dados) e pelo aumento da disponibilidade de dados espacialmente referenciados em formato digital.

Segundo [PC89], nos anos 80 o uso de SIG foi difundido, pois nessa época houve a popularização e o barateamento da tecnologia de microcomputadores; em conformidade com essa tendência, os SIG passaram a ser desenvolvidos para atender a necessidades mais restritas, ao invés de atender a um conjunto grande de necessidades, simplificando assim os sistemas, tornando-os mais baratos e com melhor desempenho, e adequados à tecnologia dos microcomputadores.

Atualmente as aplicações em SIG cobrem domínios muito variados (desde a cobertura de uma cidade até uma cobertura global) e utilizam tecnologias diversas (desde microcomputadores a supercomputadores) [CCH+96]. Segundo [Cam95], no fim dos anos 90 os SIG deverão ser sistemas orientados à troca de informações entre uma instituição e a sociedade. Essa nova abordagem requererá sistemas distribuídos e interoperabilidade.

2.2.2- Conceitos Básicos

Segundo [RM92], Sistemas de Informação Geográfica são sistemas utilizados para "captura, armazenamento, manipulação, análise e exibição de dados que são espacialmente referenciados à Terra". Os SIG trabalham tanto com dados convencionais (população, por exemplo) quanto com dados espacialmente referenciados (a localização de uma cidade, por exemplo).

Os SIG são utilizados em várias áreas; [Ciferri95] divide essas áreas em dois grupos distintos: aplicações urbanas e rurais (mapeamento urbano básico, transporte, telecomunicações, energia elétrica, saúde, educação, habitação, dentre outros) e aplicações ambientais (fauna, flora, recursos hídricos, dentre outros). Percebe-se então que a comunidade de usuários de SIG é muito heterogênea (a diversidade de usuários de SIG é tão grande quanto a diversidade de pessoas relacionadas ao uso e construção de mapas), composta por geógrafos, engenheiros, estatísticos, dentre outros. Logo, enfatizando o que foi dito na motivação ao trabalho, o projetista de interfaces para tais sistemas não pode esperar que um usuário tenha conhecimento de computação para operá-la.

Os SIG possuem os seguintes grupos básicos de funcionalidade [RM92] :

- captura e edição de dados;

- transferência de dados já capturados para um SIG através de redes de computadores ou de mídias magnéticas, por exemplo;

- armazenamento e manipulação de dados;

- consulta e análise (envolvendo operações métricas, topológicas e de orientação em dados espaciais);

- apresentação dos dados da consulta.

2.2.3 - Objetos e Campos Geográficos

As entidades geo-referenciadas do mundo real (cidades, estados, rios, florestas etc.) são geralmente modeladas segundo campos e objetos, chamados em [Cam95] de geo-campos e geo-objetos, respectivamente.

Um geo-objeto possui características próprias; é um objeto individualizável, como rios, lagos etc. Entidades artificialmente criadas (cidades, estados, rodovias etc.) são geralmente modelados como geo-objetos.

Um geo-campo, por sua vez, não possui identidade própria. São grandezas distribuídas espacialmente, correspondendo a dados contínuos e não individualizáveis (florestas, solos etc.). Por exemplo, existem inúmeras florestas de eucalipto no Brasil, mas existe somente um único Brasil; logo, geralmente, florestas de eucalipto devem ser modeladas como geo-campo e país como geo-objeto.

Um geo-campo ou um geo-objeto podem ser representados de várias formas. A representação desses objetos pode variar de acordo com a escala, projeção cartográfica, dentre outros. A figura 2.1 mostra como a variação da escala afeta a representação da cidade de Belo Horizonte, que é representada por um ponto em um mapa e por um polígono em outro mapa de maior escala. Um geo-campo geralmente é representado em formato matricial. Já, um geo-objeto é geralmente representado em formato vetorial (ponto, linha ou polígono).

O modelo matricial é baseado na subdivisão regular do plano, sob forma de matriz, enquanto que no modelo vetorial, o plano é subdividido em partes irregulares [Good92]. No modelo vetorial as entidades espacialmente referenciadas são representadas como séries de coordenadas X,Y (em dimensão dois) como mostra a figura 2.2 [Mag92]).

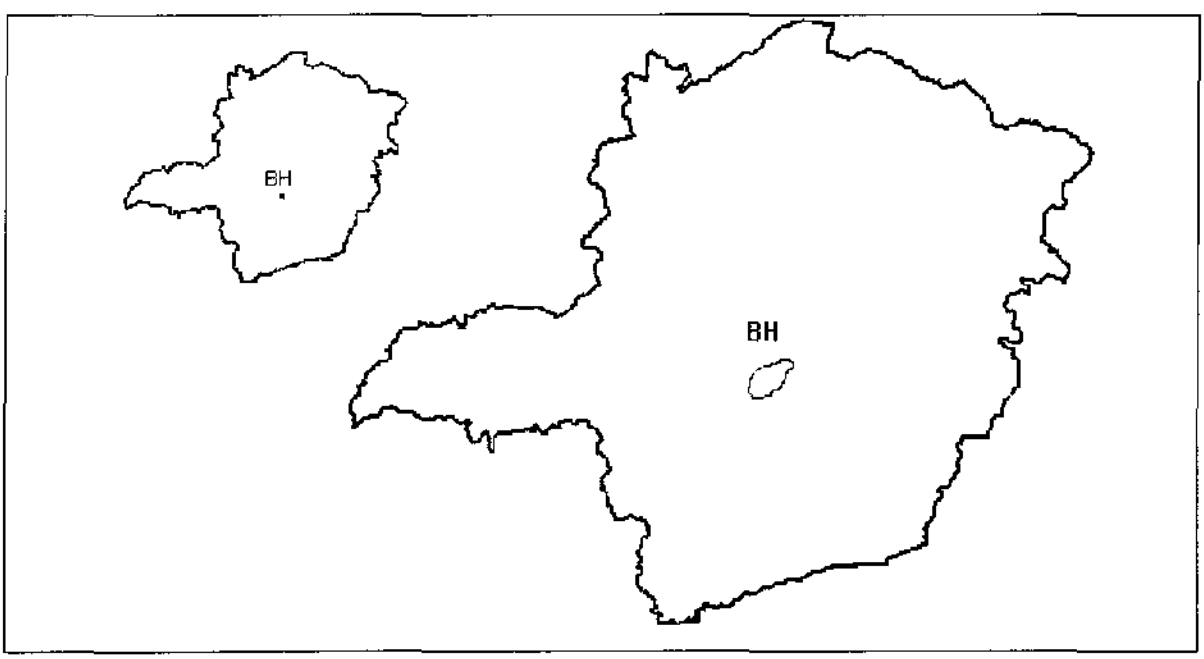


Figura 2.1: Diferentes representações para a cidade de Belo Horizonte.

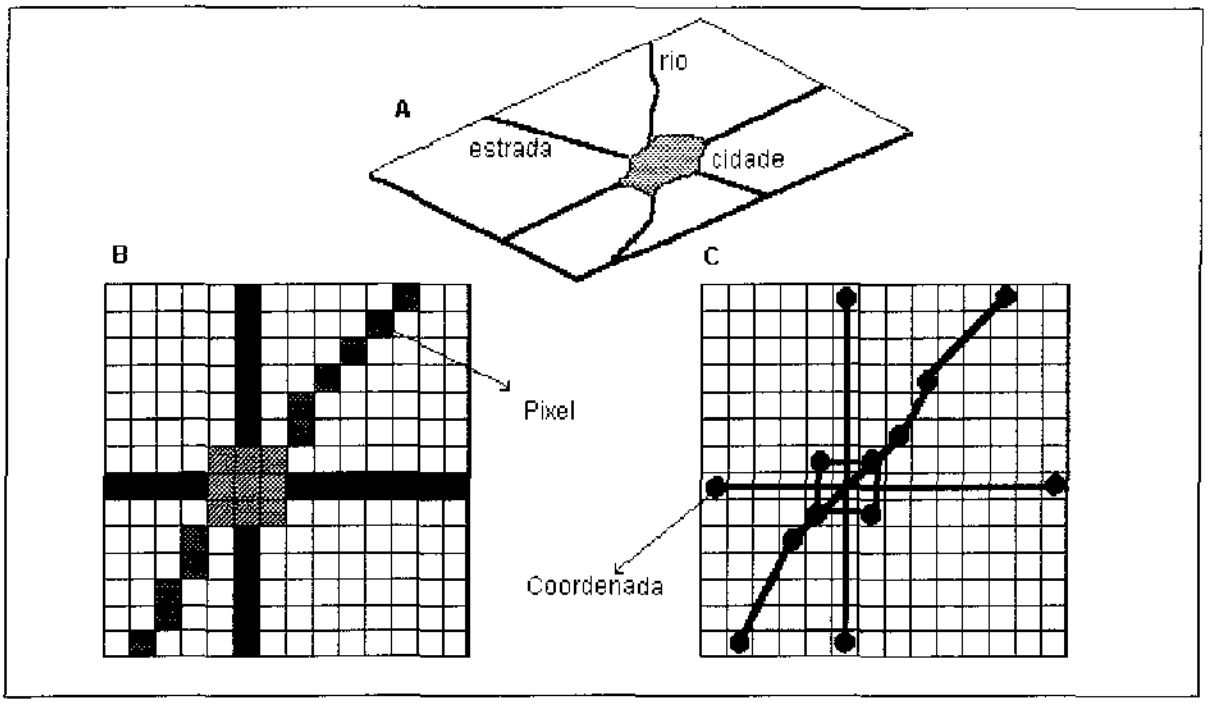


Figura 2.2: Um exemplo de como as entidades espacialmente referenciadas do mundo real (A) são representadas matricialmente (B) e vetorialmente (C). Em C, os geo-objetos "estrada" e "rio" são representados por linhas enquanto "cidade" é representada por um polígono.

Segundo [Mag92], o modelo vetorial é mais complexo que o matricial e a sua adaptação à tecnologia dos microcomputadores é mais difícil. Entretanto, os relacionamentos topológicos são mais facilmente incorporados nesse modelo; em [Frank92] é ressaltada a importância da topologia como um conceito matemático básico para a organização de dados espacialmente referenciados.

A simplicidade para a implementação e o melhor desempenho no processamento em relação ao modelo vetorial são características do modelo matricial; além disso, a aquisição de dados é mais fácil, rápida e barata devido ao uso de digitalizadores de imagens (scanners) [Mag92].

As principais diferenças entre os dois modelos são ilustradas na figura 2.3.

A escolha do modelo de dados é influenciada por vários fatores, incluindo a disponibilidade de programas de computador e a natureza da aplicação [Mag92].

Matricial	Vetorial
Modelo de dados simples	Modelo de dados complexo
Espaço discreto	Espaço contínuo
Tecnologia barata	Tecnologia mais cara
Facilidade na coleta de dados	Coleta de dados difícil e cara
Orientado à área	Orientado a bordas
Aplicações ambientais	Aplicações sócio-econômicas
Processamento topológico dificultado	Processamento topológico facilitado

Figura 2.3: As principais diferenças entre os modelos vetorial e matricial [Mag92].

2.2.4 - O Banco de Dados Exemplo

Todo o trabalho desenvolvido nesta dissertação será exemplificado a partir de um caso básico. A figura 2.4 mostra as classes com os respectivos atributos do banco de dados exemplo.

Os atributos "nome", "prefeito", "governador", "presidente", "sexo" e "partido" são do tipo texto, enquanto "área", "população", "altitude" e "extensão" são atributos numéricos. Já "geometria" descreve a extensão espacial das classes de geo-objetos (todas as classes, exceto a classe "administrador" são classes de geo-objetos).

Os relacionamentos entre a classe não espacial "administrador" e as classes de geo-objetos "cidade", "estado" e "país" são expressos através das chaves estrangeiras "prefeito", "governador" e "presidente", e se relacionam com a chave primária "nome" da classe "administrador" (o relacionamento entre as classes de geo-objetos são relacionamentos espaciais, e não expressos com o uso de chaves). Supõe-se neste trabalho que as classes de geo-objetos estejam representadas vetorialmente. A figura 2.5 mostra exemplos dos geo-objetos "estado", "cidade" e "estrada".

O banco de dados é simples, e serviu de base para o estudo de algumas consultas que o usuário pode realizar. Sobre esse banco de dados foi definido o subconjunto da linguagem natural coberto pelo tradutor (vocabulário e regras gramaticais).

O tradutor está intimamente ligado à aplicação escolhida. Somente consultas relativas ao domínio dessa aplicação podem ser compreendidas pelo tradutor ("Quais as cidades cortadas pela rodovia br040?", por exemplo). Consultas sobre meteorologia, por exemplo, não são cobertas por esse subconjunto da linguagem natural e, portanto, não são entendidas pelo tradutor. O capítulo 5 (Conclusões e Trabalhos Futuros) descreve os passos para se conseguir a portabilidade em relação a várias aplicações.

cidade(nome, área, população, altitude, prefeito, geometria)
município(nome, área, população, geometria)
estado(nome, área, população, governador, geometria)
país(nome, área, população, presidente, geometria)
rio(nome, extensão, geometria)
estrada(nome, extensão, geometria)
lago(nome, área, geometria)
administrador(nome, sexo, partido)

Figura 2.4: O banco de dados utilizado como base para as consultas suportadas pelo tradutor.

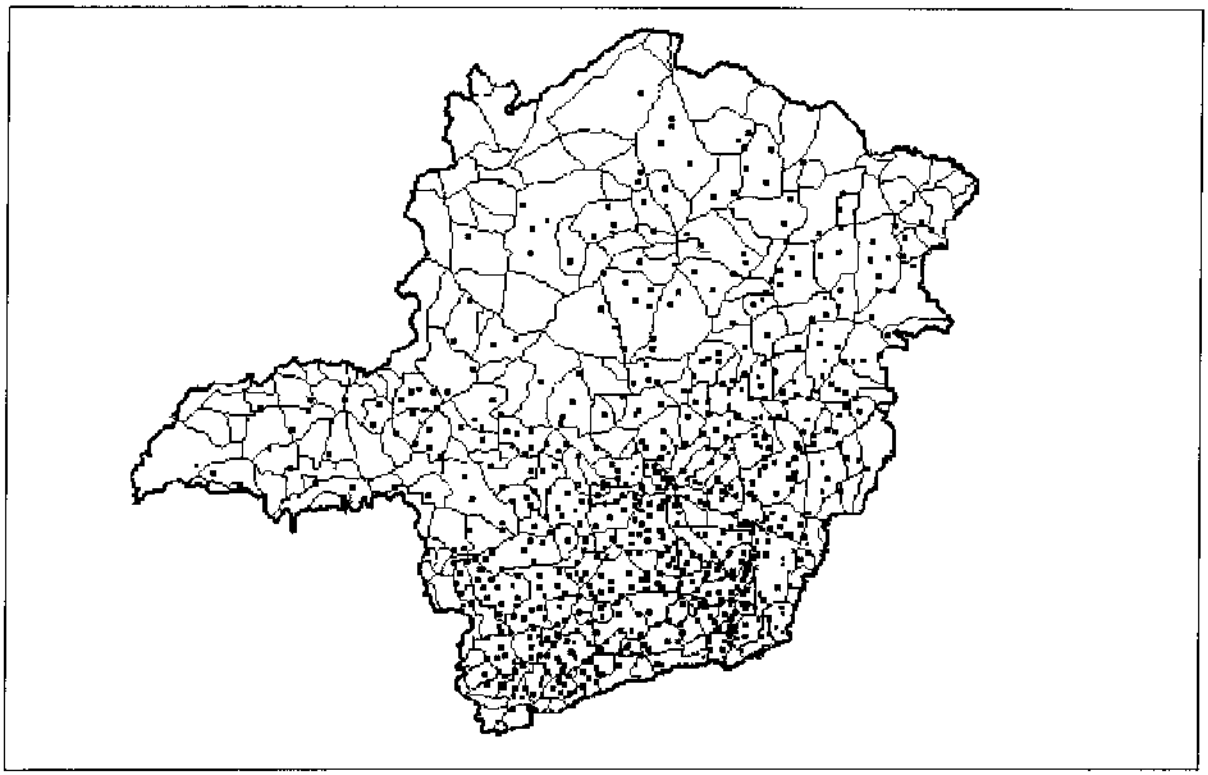


Figura 2.5: Um mapa do estado de Minas Gerais contendo instâncias das classes de geo-objetos "cidade" e "estrada".

2.3 - A Linguagem LEGAL

Como dito anteriormente, a consulta escrita em LN deverá ser convertida para uma linguagem de consulta para SIG. Essa linguagem deverá manipular tanto dados convencionais (números, textos etc.) quanto dados espacialmente relacionados. A linguagem LEGAL (Linguagem Espacial para Geoprocessamento Algébrico) foi escolhida como tal linguagem. Esta dissertação seguiu as especificações da linguagem LEGAL citadas em [Cam95].

A linguagem LEGAL, desenvolvida no INPE, é uma extensão do SQL, e inclui operadores espaciais e facilidades para a manipulação de geo-campos e geo-objetos [CCH+96]. A linguagem LEGAL possui os seguintes grupos de funcionalidade:

- definição do esquema do banco de dados;
- criação de geo-objetos e geo-campos;
- recuperação de dados;
- manipulação de objetos e coleções;
- apresentação dos dados.

Nesta dissertação, somente a parte de recuperação de dados é abordada, isto é, a consulta em LN é convertida para uma expressão de consulta em LEGAL. Consultas em LEGAL envolvem dados convencionais e restrições espaciais em geo-objetos e apenas dados convencionais de geo-campos [Cam95].

Por ser uma extensão do SQL, consultas em LEGAL têm forma análoga ao "SELECT ... FROM ... WHERE ..." dessa linguagem, como mostra a figura 2.6. A cláusula "SELECT" indica os atributos, resultados de operadores e/ou objetos a serem recuperados segundo critérios especificados na cláusula "WHERE"; as classes sobre as quais se realiza a consulta devem constar na cláusula "FROM". O qualificativo "ON MAP" na cláusula "FROM" é usado para indicar sobre qual mapa será computada a restrição espacial. Esse qualificativo pode ser omitido caso haja apenas uma representação para o geo-objeto. A título de simplicidade não utilizamos o qualificativo "ON MAP" nas consultas em LEGAL convertidas pelo tradutor, isto é, admitimos que cada classe de geo-objetos possui uma única representação.

SELECT <lista de valores e/ou objetos>
FROM <objetos> IN <classe> ON MAP <mapa>
WHERE <critérios>

Figura 2.6: Sintaxe de uma consulta em LEGAL.

2.3.1 - Operadores Espaciais

Os relacionamentos espaciais entre geo-objetos podem ser classificados em topológicos, métricos e de orientação [Cília96].

A linguagem LEGAL oferece o operador espacial métrico "distance" que pode ser utilizado tanto na cláusula "SELECT" (caso A, da figura 2.7) quanto na "WHERE" (caso B, da mesma figura) mais cinco operadores topológicos: "cross", "disjoint", "inside", "overlap" e "touch", que devem ser usados somente na cláusula "WHERE" de uma consulta em LEGAL , pois retornam resultados "booleanos". O capítulo 4 traz exemplos de consultas que envolvem esses operadores. A linguagem LEGAL não fornece operadores de orientação; portanto, consultas do tipo "Quais são as cidades localizadas ao norte de Campinas" não são suportadas por essa linguagem.

A: Qual a distância entre as cidades de Campinas e Bauru

```
select  distance(c1,c2)
from    c1 in cidade, c2 in cidade
where   c1.nome = "Campinas" and c2.nome = "Bauru"
```

B: Quais são as cidades a menos de 50 km da cidade de Campinas

```
select  c1
from    c1 in cidade,  c2 in cidade
where   distance(c1,c2) < 50 and c2.nome = "Campinas"
```

Figura 2.7: Exemplos de consultas em LEGAL envolvendo distância.

2.4 - Conclusões

Neste capítulo, fez-se uma breve introdução sobre as duas grandes áreas envolvidas nesta dissertação, processamento de linguagem natural e sistemas de informação geográfica. O banco de dados escolhido para a definição do subconjunto da LN compreendido pelo tradutor, além da linguagem de consulta LEGAL, linguagem para a qual a consulta em LN é convertida, também foram mostrados.

O capítulo seguinte detalha os módulos constituintes do tradutor, isto é, o analisador, o vocabulário, a base de conhecimentos, o interpretador e o gerador de consultas em LEGAL.

Capítulo 3 - O Tradutor

Conforme dito anteriormente, o objetivo deste trabalho é a implementação de um tradutor de linguagem natural para a linguagem LEGAL.

Este capítulo destina-se ao detalhamento dos módulos constituintes do tradutor e dos problemas lingüísticos com os quais o tradutor deverá lidar.

3.1 - Prolog

O tradutor foi implementado em PROLOG (Arity PROLOG [Arity87]). Linguagens declarativas como o PROLOG são ideais para escrever programas em inteligência artificial, porque pode-se trabalhar em alto-nível; ao invés de preocupar-se com os passos que o computador segue na resolução de um problema, pode-se concentrar apenas nos relacionamentos necessários entre os objetos para se chegar à solução desejada. Também em comparação com linguagens procedurais, os programas escritos em PROLOG tendem a ser mais compactos. Entretanto, apesar da aparente simplicidade dos programas em linguagens declarativas, a maioria dos programas em inteligência artificial implementados sob esse paradigma tornam-se complexos [HW88]; além disso, segundo [Schildt89], linguagens declarativas não possuem capacidade de programação procedural, uma deficiência que pode dificultar certas tarefas, tais como adicionar elementos a uma lista.

A programação em PROLOG consiste em [CM84]:

- declarar fatos sobre objetos e os seus relacionamentos, como por exemplo:

```
pai(ismar,apolônio).
pai(apolônio,frederico).
aves([pardal,sabiá,canário]).
```

A estrutura "[pardal,sabiá,canário]" é chamada de lista e é comumente utilizada em programas PROLOG;

- declarar regras sobre os objetos e os seus relacionamentos, como por exemplo:

avô(X, Y) :- pai(X, Z), pai(Z, Y).

Essa regra pode assim ser lida: "X é avô de Y se X é pai de Z e Z é pai de Y";

- consultar sobre os objetos e os seus relacionamentos, como por exemplo:

? - avô(ismar, frederico).

O resultado dessa consulta indica se é verdadeiro ou falso que "ismar" é avô de "frederico".

O PROLOG (Arity PROLOG) ainda tem a grande vantagem de implementar diretamente as gramáticas de cláusulas definidas, que podem ser utilizadas na verificação sintática e na extração da semântica de uma expressão em LN.

Algumas versões do PROLOG, tais como o Arity PROLOG 5.0 e o TURBO PROLOG, permitem a chamada de procedimentos escritos em outras linguagens de programação, como a linguagem C (Lattice C e Microsoft C para o ARITY PROLOG 5.0, e TURBO C para o TURBO PROLOG). Essa característica será muito útil na futura construção da interface multimodal.

O seguinte trecho em Arity PROLOG 5.0 ilustra esse intercâmbio entre as duas linguagens [Arity87]:

```
:- extrn dobro/2:c(dobro).  (declara a função "dobro" como sendo uma
                             função  externa de aridade 2 escrita em linguagem C)
write('Digite um número inteiro'), read(N), dobro (N, D),
write ('O dobro do número é'), write (D).
```

A função " dobro " escrita em C seria:

```
dobro( int x, int *y)
{
        *y = 2*x;
}
```

As referências [HW88], [Le93], [AD89] e [PS87] mostram como alguns problemas em Inteligência Artificial e em Processamento de Linguagem Natural são implementados nessa linguagem.

3.2 - A Arquitetura

A arquitetura do sistema proposto é similar à arquitetura geral para sistemas de processamento de linguagem natural proposta por [Tedes90].

O tradutor é formado por cinco partes: o analisador (em inglês, *parser*), o vocabulário, a base de conhecimentos, o interpretador, e o gerador de consultas em LEGAL, como mostra a figura 3.1. O analisador é incumbido de analisar sintaticamente a consulta em LN e extrair os seus componentes semânticos. Para isso o analisador verifica se os vocábulos da sentença estão contidos no vocabulário do sistema e se eles formam um conjunto gramaticalmente correto. Os componentes semânticos são utilizados pelo interpretador para verificar se a requisição do usuário pode ser satisfeita pelo banco de dados (através de informações contidas na base de conhecimentos), além de outras inferências; em caso afirmativo a consulta em LEGAL é construída pelo gerador de consultas em LEGAL, de acordo com as regras de sintaxe que norteiam essa linguagem. Esses módulos serão detalhados a seguir. O vocabulário e a base de conhecimentos são repositórios de dados, dados esses que são processados por outras partes, por isso a distinção gráfica entre elas.

3.3 - O Analisador

O analisador é um programa cuja função é analisar sintaticamente a consulta em LN e extrair o seu significado. O analisador usa uma gramática capaz de descrever algumas consultas possíveis de serem realizadas por um usuário, isto é, o analisador cobre um subconjunto da língua portuguesa com poder de expressão em relação ao domínio da aplicação.

Se a consulta em LN estiver de acordo com essas regras gramaticais, os seus itens semânticos (classes, atributos e condições) serão extraídos.

Caso contrário assume-se que os seguintes erros podem ter ocorrido:

- erro ortográfico;
- palavra não coberta pelo vocabulário;
- erro gramatical.

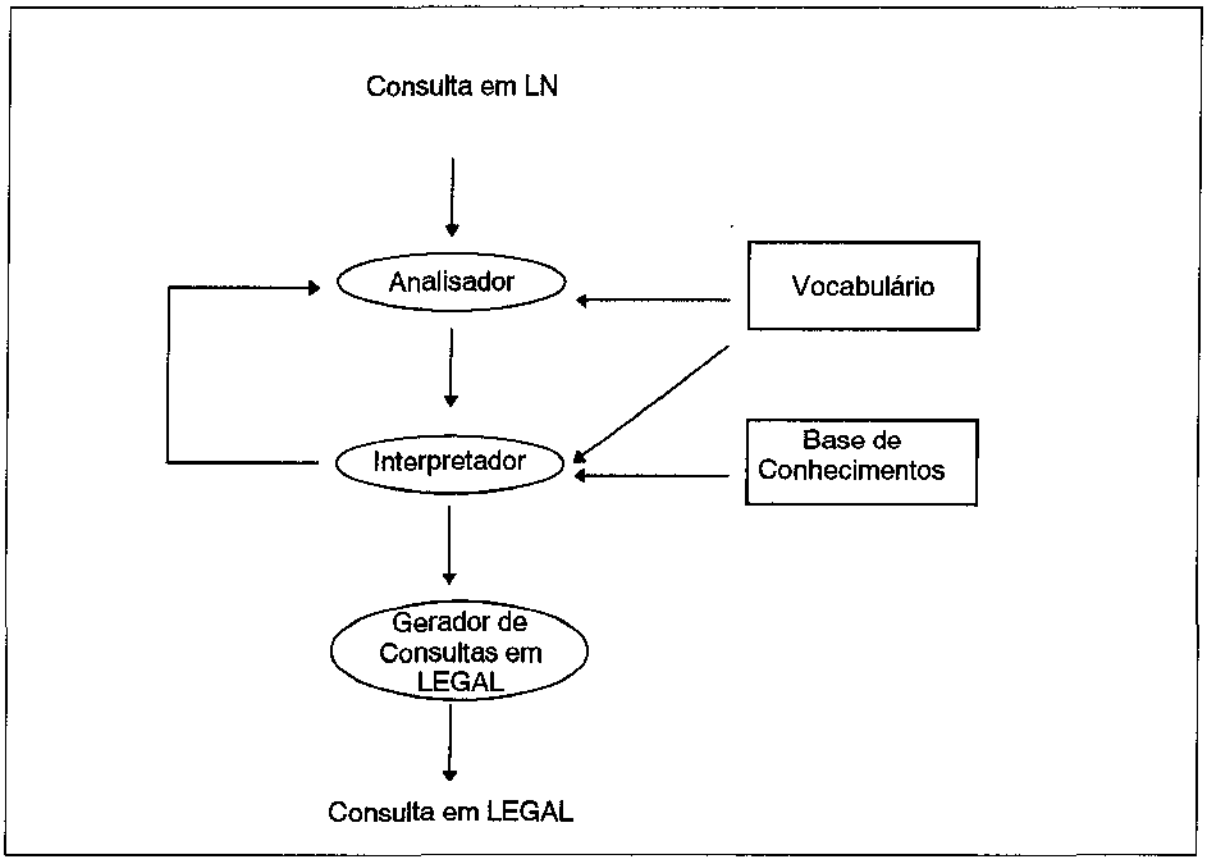


Figura 3.1: A arquitetura do tradutor.

Uma vantagem das consultas em LN sobre as consultas em linguagem formal é que o tradutor pode reconhecer os objetivos do usuário mesmo em uma sentença incompleta

(sentença elíptica, por exemplo) ou com erros ortográficos, como a digitação de "ppulação" ao invés de "população".

Se a primeira análise da sentença falha, o interpretador busca na sentença de entrada (consulta em LN) por erro ortográfico; se houver, a análise é refeita automaticamente com a palavra correta (o que explica a seta retornando do módulo interpretador para o analisador na figura 3.1). Senão, assume-se que a palavra não consta do vocabulário e o fato é notificado ao usuário. Quando o erro for gramatical (em relação ao subconjunto de LN coberto) e, portanto, irrecuperável, o usuário terá que formular a sua consulta de outra maneira. Considere a figura 3.2 onde, no caso A, o sistema soube tratar os erros de digitação das palavras "distância" e "cidad" e traduzir a sentença para LEGAL. No caso B não houve erro ortográfico, mas a palavra "localidade" não faz parte do vocabulário e a tradução falha. Em C, a sentença não obedece a nenhuma regra gramatical prevista pelo analisador e a tradução também falha. Nesse caso o usuário terá que reformular a sua consulta.

A : Qual é a **dstância** entre as **cidad** de Campinas e Araxá ?

```
select  distance(c1,c2)
from    c1 in cidade, c2 in cidade
where   c1.nome = "Campinas" and c2.nome = "Araxá"
```

B : Qual é a área da **localidade** cuja população é 1000 habitantes ?

A palavra "localidade" não foi reconhecida pelo sistema !

C : A área é qual da cidade ?

Sentença não compreendida pelo sistema !

Figura 3.2: Exemplos de comportamento do tradutor.

3.3.1 - As Gramáticas de Cláusulas Definidas

As regras gramaticais do analisador foram escritas utilizando-se uma gramática de cláusulas definidas (GCD). As GCD são um formalismo claro e poderoso o suficiente para a descrição da maioria das construções em linguagem natural. Uma GCD provê não somente a definição de uma linguagem, mas também meios efetivos para analisá-la [PW80].

De acordo com [Carb85], uma GCD define as expressões de uma linguagem em termos de regras, que são equações recursivas sobre tipos de expressões chamadas de não-terminais, e expressões primitivas, os terminais. Por exemplo, os símbolos "o" e "cidade" da figura 3.3 são terminais e representam palavras de diferentes categorias gramaticais (léxico); os símbolos "sentença", "sintagma_nominal" e "predicativo" são não-terminais e representam categorias frasais.

Segundo [PW80], as GCD têm como vantagem a modularidade e a simplicidade na descrição de regras gramaticais, e a possibilidade de representar a recursividade dessas regras, que é característica da maioria das linguagens de interesse. Por exemplo, a recursividade das regras "sintagma_nominal" e "where" da figura 3.3 permitem a seleção de múltiplos atributos segundo múltiplas condições (cite $atributo_1$, $atributo_2$,..., $atributo_n$ da $classe_i$ onde $condição_1$, e/ou $condição_2$,..., e/ou $condição_n$). As GCD são interpretadas pelo Arity PROLOG, interpretador no qual o tradutor foi implementado.

As principais classes de regras gramaticais contidas no analisador são:

- **Sintagma Nominal**: os sintagmas nominais são utilizados para a referência a atributos do banco de dados. Os sintagmas nominais são partes da sentença onde o substantivo é o elemento principal. Os outros constituintes desses sintagmas são os artigos, adjetivos e as conjunções ("e" e "ou") que ligam um sintagma nominal a outro;

- **Sintagma Preposicional**: qualifica partes da sentença. É composto por uma preposição seguida de um substantivo. Esse sintagma relaciona uma noção de origem aos substantivos do sintagma nominal, isto é, ele associa os atributos citados no sintagma nominal com uma determinada classe do banco de dados;

- **Predicativo**: são regras que analisam e extraem as solicitações do usuário relativas à cláusula "SELECT" da linguagem LEGAL, isto é, os atributos e/ou objetos e/ou resultados de operadores (min(altitude), distance(cidade,estrada), por exemplo). O núcleo dessas regras é formado por substantivos ligados pelo sujeito da sentença através de um verbo de ligação;

- **Where**: são regras utilizadas na verificação sintática e na extração dos itens semânticos dos critérios da consulta em LN. Essas regras possibilitam o entendimento de várias formas diferentes de critérios, que podem ser combinados através de conjunções ou vírgulas.

Como dito anteriormente, sobre o banco de dados exemplo foram estudadas algumas consultas que um usuário poderia realizar. O estudo das regras gramaticais que norteiam essas consultas serviu de base para a implementação da gramática do analisador. Por exemplo: a análise da consulta "Qual é o nome e o prefeito das cidades cuja população é superior a 1000, cuja altitude é igual a 500 e cuja área é inferior a 600?" levou à confecção das regras da figura 3.3. Essas regras são similares às regras contidas no analisador, embora tenham sido simplificadas para facilitar o entendimento do leitor (a listagem completa das regras implementadas para o analisador se encontra no apêndice da dissertação). A regra A pode ser entendida como: uma **sentença** pode ser formada por um **pronome** seguido de um **verbo,** seguido de um **predicativo,** que retornará as variáveis "SELECT" e "FROM" (essas variáveis armazenam, respectivamente, os itens semânticos relativos às cláusulas "select" e "from" da linguagem LEGAL) e, finalmente, de **where** que retornará os componentes semânticos relativos à cláusula "where" da linguagem LEGAL ("WHERE", "COMPARADOR", "VALOR" e "CONECTIVO"). A figura 3.4 mostra os itens semânticos relativos a cada uma dessas variáveis. A regra B diz que o **predicativo** pode ser formado por um **sintagma nominal,** de onde os componentes semânticos (atributos) relativos à cláusula "select" da linguagem LEGAL são retirados, seguido de um **sintagma preposicional** que, por sua vez, é formado por uma **preposição** e um **substantivo** correspondente à classe relativa aos atributos do **sintagma nominal**. Leitura semelhante se aplica às outras regras, exceto onde aparece o símbolo ";" que representa disjunção. Por exemplo, a regra C diz que um sintagma nominal pode ser formado por uma conjunção **ou** uma vírgula seguido de um artigo definido **ou** não, mais um substantivo e um outro sintagma nominal.

O processamento das sentenças de acordo com as regras da figura 3.3 produz árvores semelhantes à da figura 3.4 (chamadas árvores de derivação ou análise), de onde os componentes semânticos são extraídos.

```
A:    sentença --> pronome, verbo(_),
      predicativo(SELECT,FROM), where(WHERE,COMPARADOR,VALOR,CONECTIVO).
      /* Qual é a área da cidade cuja população é maior que 1000 */

B:    predicativo(SELECT,FROM)
      --> sintagma_nominal(SELECT), sintagma_preposicional(FROM).
      /* a área da cidade */

      sintagma_nominal ([ATRIBUTO|T]) --> artigo_definido, substantivo_atributo(ATRIBUTO),
      sintagma_nominal(T).
      /* a área */

C:    sintagma_nominal([ATRIBUTO|T]) --> (conjuncao(_); virgula(_)), (artigo_definido; [ ]),
      substantivo_atributo(ATRIBUTO), sintagma_nominal(T).
      /* e/, <a> área */

      sintagma_nominal([ ]) --> [ ].
      /* fim da recursão */

      sintagma_preposicional(FROM) --> preposição, substantivo_classe(FROM).
      /*  da cidade */

      where([ATRIBUTO|A],[COMPARADOR|B],[VALOR|C],CONECTIVO)
      --> pronome, (artigo_definido; [ ]), substantivo_atributo(ATRIBUTO), verbo(_),
      adjetivo(COMPARADOR),  pronome, valor(VALOR), where(A,B,C,CONECTIVO).
      /* cuja <a> população é maior que 1000 */

      where([ATRIBUTO|A],[COMPARADOR|B],[VALOR|C],[CONECTIVO|D])
       --> (conjuncao(CONECTIVO); virgula(CONECTIVO)), pronome, (artigo_definido; [ ]),
      substantivo_atributo(ATRIBUTO), verbo(_), adjetivo(COMPARADOR), pronome,
      valor(VALOR),  where(A,B,C,D).
      /* e/, cuja população <a> população é maior que 1000 */

       where([ ],[ ],[ ],[ ]) --> [ ].
      /* fim da recursão */

      artigo_definido --> [a].
      artigo_definido --> [o].
      substantivo_atributo(area) --> [area].
      substantivo_atributo(populacao) --> [populacao].
      substantivo_classe(cidade) --> [cidade].
      conectivo(and) --> [e].
      conectivo(or) --> [ou].
                                                    ⋮
```

Figura 3.3: Algumas regras gramaticais escritas em GCD.

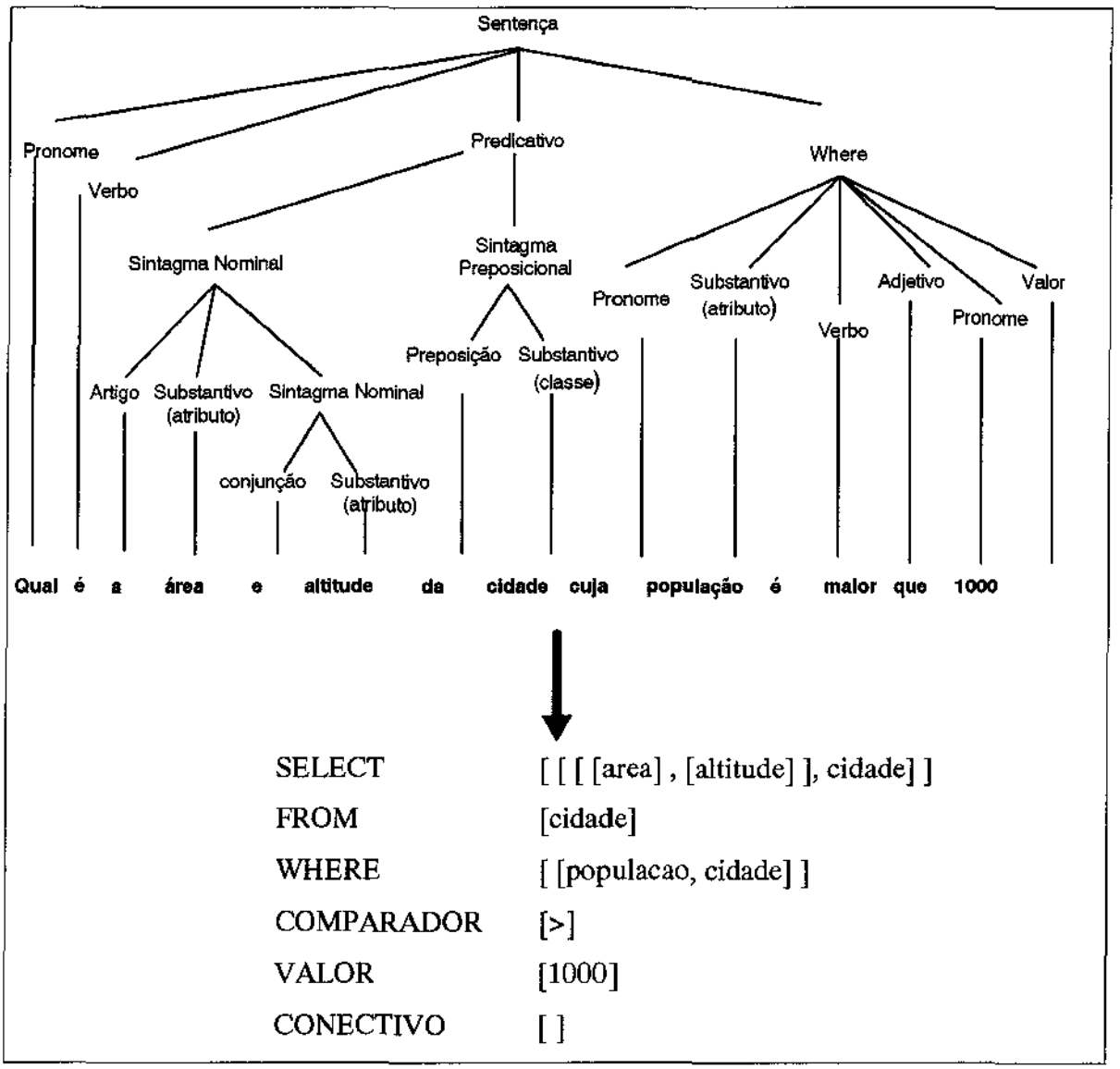


Figura 3.4: A árvore de derivação da sentença: "Qual é a área e altitude da cidade cuja população é maior que 1000" e os itens semânticos dela extraídos.

3.4 - O Vocabulário

O vocabulário contém as palavras com as quais o usuário pode se expressar, ou seja, com palavras comumente utilizadas na língua portuguesa (artigos, preposições etc.), mais palavras relacionadas ao domínio da aplicação, isto é, palavras que descrevem as classes, atributos e domínios do banco de dados, tais como: "cidade", "altitude" e "metros", respectivamente, além de alguns sinônimos dessas palavras. A figura 3.5 mostra exemplos de diferentes categorias de palavras contidas no vocabulário.

```
eh_subst_class(estrada).
eh_subst_atr(extensao).
eh_pronome(qual).
eh_verbo(atravessada,cruzamento).
eh_artigo_def(a).
eh_artigo_indef(um).
eh_preposicao(de).
eh_adjetivo(menor,<).
eh_adjetivo_fSQL(minima,min).
eh_conectivo(e,and).
eh_unidade(metros,[altitude,extensão]).
eh_sinonimo(rodovia,estrada).
```

Figura 3.5: Exemplos de classes de palavras contidas no vocabulário.

O vocabulário é usado também na conversão de palavras em LN para os seus termos correspondentes em LEGAL, como mostra a figura 3.6, onde a palavra "média" é convertida para "avg" e a palavra "maior" para o símbolo ">". Além disso as palavras que expressam relacionamentos topológicos são relacionadas às respectivas noções topológicas que elas expressam (eh_verbo(cortada,cruzamento)).

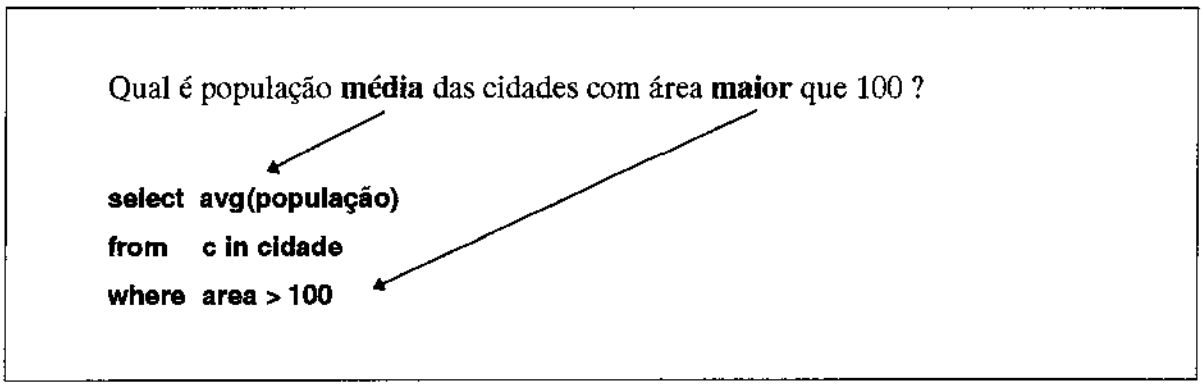


Figura 3.6: Exemplos de conversão de palavras em LN para os correspondentes termos em LEGAL.

3.5 - A Base de Conhecimentos

A base de conhecimentos (figura 3.7) representa a estrutura do banco de dados. Nela estão contidos os nomes das classes, os seus respectivos atributos, além dos relacionamentos entre a classe não espacial "administrador" e as classes de geo-objetos "cidade", "estado" e "país" (através de chaves primária e estrangeira) e informações sobre a representação de cada classe de geo-objetos.

A base de conhecimentos é utilizada para :

- verificar se as classes e atributos citados na consulta em LN fazem parte do banco de dados;

- servir de fonte para a construção de relacionamentos que envolvam classes não espaciais, e de operadores topológicos com base em informações sobre a representação de cada classe de geo-objetos;

- servir de fonte de informações para as meta-consultas (consultas sobre a estrutura do banco de dados).

```
eh_atributo(nome,cidade).
eh_atributo(populacao,cidade).
eh_atributo(area,cidade).
eh_atributo(altitude,cidade).
eh_atributo(prefeito,cidade).
eh_atributo(geometria,cidade).


eh_atributo(nome,estado).
eh_atributo(populacao,estado).
eh_atributo(area,estado).
eh_atributo(governador,estado).
eh_atributo(geometria,estado).


eh_atributo(nome,administrador).
eh_atributo(sexo,administrador).
eh_atributo(partido,administrador).

eh_classe(cidade).
eh_classe(estado).
eh_classe(administrador).

representacao(cidade,d0)          /* d0 = dimensão zero (ponto) */
representacao(estado,d2)          /* d2 = dimensão dois (polígono) */

/* relacionamento([classe1,classe2], chave-estrangeira, chave-primaria]). */
   relacionamento([cidade,administrador],[prefeito,cidade],[nome,administrador]).
   relacionamento([estado,administrador],[governador,cidade],[nome,administrador]).
```

Figura 3.7: A base de conhecimentos acerca das entidades "cidade", "estado" e "administrador".

3.6 - O Interpretador

O interpretador verifica se a consulta em LN é válida, isto é, embora a consulta esteja gramaticalmente correta, alguma classe e/ou atributo pode não estar contido no banco de dados. Se a consulta for válida o gerador de consultas em LEGAL construirá a consulta em LEGAL com os componentes semânticos extraídos pelo analisador.

O desconhecimento do domínio coberto pelo banco de dados pode levar o usuário a formular consultas sobre objetos que não estejam no banco de dados. Se isto ocorrer, o tradutor avisará o usuário de que tais objetos não fazem parte do banco de dados (figura 3.8).

Qual é a população da zona da mata ?

A classe "zona" não faz parte do banco de dados !

Qual é a latitude da cidade de campinas ?

O atributo "latitude" não está contido na classe "cidade" !

Figura 3.8: Consultas a classes e atributos não contidos no banco de dados.

Para atenuar esses erros, o usuário pode utilizar as chamadas meta-consultas (consultas sobre a estrutura do banco de dados), buscando por informações contidas na base de conhecimentos. As meta-consultas, assim como as consultas comuns, são processadas pelo analisador. O interpretador é incubido de recuperar as meta-informações na base de conhecimentos, de acordo com as necessidades do usuário (extraídas pelo analisador). Alguns exemplos de meta-consultas suportadas pelo tradutor são mostradas na figura 3.9.

O que tem em cidade ?

Atributos: nome, area, populacao, altitude, prefeito e geometria.

Quais são os atributos de administrador ?

Atributos: nome, sexo e partido.

Quais são as classes do banco de dados ?

Classes: cidade, municipio, estado, pais, rio, lago, administrador e estrada.

Figura 3.9: Alguns exemplos de meta-consultas.

O interpretador é encarregado de outras funções, tais como:

- resolução de ambigüidades (a seção 3.8.2 mostra a metodologia para a resolução de ambigüidades);

- construção do relacionamento entre uma classe espacial e outra não espacial, ou entre duas destas, quando necessário. Essa construção é feita com base em informações contidas na base de conhecimentos (através da utilização das chaves primária e estrangeira). A primeira consulta da figura 3.10 mostra em sublinhado a construção do relacionamento entre "cidade" e "administrador";

- conversão de palavras que expressam relacionamentos topológicos para os operadores topológicos da linguagem LEGAL adequados (o capítulo 4 detalha essa conversão);

- gerenciamento de informações relativas ao contexto (a seção 3.8.1 mostra como o contexto da interação é utilizado);

- correção ortográfica. Foi implementado um corretor ortográfico simples, utilizado para ilustrar uma vantagem de um interpretador inteligente. Quando a análise de uma sentença falha por um possível erro ortográfico, o corretor ortográfico compara o termo da sentença que causou o erro com as palavras do vocabulário e tenta retornar o vocábulo mais parecido com tal termo. Se ele não conseguir, admite-se que tal termo não é relativo a nenhuma palavra do vocabulário (como o caso B da figura 3.2).

3.7 - O Gerador de Consultas em LEGAL

O gerador de consultas em LEGAL é utilizado na parte final da tradução. Ele é responsável pela construção da consulta em LEGAL correspondente à consulta em LN.

Alguns problemas surgem na construção da consulta em LEGAL, como a qualificação de atributos para eliminar ambigüidades e a utilização de "apelidos" (aliases). Quando um atributo relativo à cláusula "Select" ou "Where" pertence a mais de uma classe citada em "From" o gerador de consultas em LEGAL qualifica (através de prefixação) esse atributo; caso contrário a prefixação não é necessária e, portanto, não realizada pelo gerador de consultas em LEGAL. Na figura 3.10, em ambas as consultas, o atributo "nome" é prefixado. Essa prefixação é necessária para eliminar ambigüidades, pois esse atributo pertence a todas as classes envolvidas em cada consulta. A outra consulta exemplifica a utilização dos "aliases" "e1", "e2" e "e3".

Qual é a população média das cidades cujo prefeito é do partido PT?

```
select avg(população)
from   c in cidade, a in administrador
where partido = "PT" and prefeito = a.nome
```

Quais são as rodovias que cruzam a via Dutra e que cruzam a estrada br265 ?

```
select  e1
from    e1 in estrada, e2 in estrada, e3 in estrada
where   cross(e1,e2), and e2.nome = "Dutra" and
        cross(e1,e3) and e3.nome = "br265"
```

Figura 3.10: Construção do relacionamento (primeira consulta em sublinhado) e o uso de "aliases" (segunda consulta).

3.8 - Problemas Lingüísticos

Esta seção apresenta alguns problemas lingüísticos que o tradutor enfrenta em algumas consultas em LN. A maioria desses problemas não ocorrem somente em consultas a bancos de dados em LN, mas também em grande parte dos outros tipos de sistemas que tratam LN [ART95c].

3.8.1 - O Contexto

Um tradutor inteligente não deve somente processar sentenças isoladas. Existem construções na língua portuguesa, assim como em outras línguas, que permitem referências a objetos mencionados em sentenças anteriores. A elipse e a referência pronominal são exemplos dessas construções [Wall84]. Assim, o sistema permite ao usuário formular consultas incrementalmente, isto é, o usuário pode formular as suas requisições de forma abreviada e inteligente.

Como a implementação de questões relativas ao contexto envolvendo várias sentenças anteriores é extremamente difícil [Wall84], decidiu-se neste trabalho que o contexto será estabelecido somente pela consulta imediatamente anterior mesmo porque, segundo [Egen92], os usuários de SIG tipicamente trabalham diretamente com os resultados das suas consultas.

O contexto armazena os atributos relativos à cláusula "Select", classes da cláusula "From"e os critérios da cláusula "Where", ou seja, as variáveis "SELECT", "FROM", "WHERE", "COMPARADOR", "VALOR" e "CONECTIVO" vistas anteriormente. Após cada consulta traduzida com sucesso, o contexto é atualizado com os itens semânticos dessa consulta.

O contexto serve de base para a implementação da referência pronominal e de alguns casos de elipse suportados pelo tradutor, como mostrado a seguir.

3.8.1.1 - Elipse

Segundo [Cegalla91], elipse é a omissão de um termo da oração facilmente identificável, quer por elementos presentes na própria sentença, quer pelo contexto. A resolução da elipse depende da estrutura gramatical da sentença. Quando o termo omitido é um atributo relativo à cláusula "Where", procura-se na própria sentença por palavras que indiquem esse termo. Já quando o termo faltante é relativo a uma classe ou a atributos relativos à cláusula "Select", recorre-se ao contexto em busca do termo omitido. Na figura 3.11, a consulta A ilustra um caso onde a elipse é resolvida com informações contidas na própria sentença; nesse caso a consulta completa correspondente seria: "Quais são as cidades com população superior a 1000 habitantes", onde o termo "população" foi omitido. O tradutor então recuperará o termo elidido através da unidade "habitantes" mencionada, pois o vocabulário informa a que atributos tal unidade pertence (eh_unidade(habitantes,[população])).

A: Quais são as cidades com mais de 1000 habitantes ?

```
select c
from   c in cidade
where  população > 1000
```

B: Qual é a rodovia com mais de 100000 m ?

```
select e
from   e in estrada
where  extensão > 100000
```

C: E a com extensão inferior a 5000 m ?

```
select e
from   e in estrada
where  extensão < 5000
```

D: Qual é a área e a altitude da cidade cuja população é maior que 45000 ?

```
select área, altitude
from   c in cidade
where  população > 45000
```

E: E da cidade com população menor que 10000 ?

```
select área, altitude
from   c in cidade
```

```
where população < 10000
```

Figura 3.11: Exemplos de algumas consultas com elipse.

No caso B, "metros" é domínio tanto de "extensão" quanto de "altitude" (eh_unidade(metros,[altitude,extensão])), mas como a classe "rodovia" não tem o atributo "altitude", o tradutor entende o termo faltante como "extensão". Entretanto, se no conjunto de atributos possíveis mais de um atributo puder pertencer a uma classe, a sentença será considerada ambígua e o sistema perguntará ao usuário a qual dos possíveis atributos ele quis se referir.

No caso C a elipse só pode ser tratada através de informações do contexto. Nessa situação o interpretador sabe, pela estrutura gramatical da sentença, que o termo omisso é relativo a uma classe; então busca-se no contexto pela última classe mencionada.

O caso E mostra a omissão dos atributos a serem selecionados. Também devido à estrutura gramatical da sentença, o interpretador busca os atributos no contexto gerado pela consulta anterior (consulta D).

3.8.1.2 - Referência Pronominal

A referência pronominal também pode ser usada na formulação de consultas incrementais. Assim como a elipse, a referência pronominal abrevia as sentenças, tornando-as mais adequadas a uma interação inteligente. Somente um tipo de referência pronominal é compreendida pelo tradutor: a referência a critérios da consulta anterior. Quando detectado esse tipo de referência pronominal (através da estrutura gramatical da sentença), busca-se, no contexto gerado pela consulta anterior, os critérios dessa consulta.

A figura 3.12 mostra como a combinação de elipse e referência pronominal torna a interação do usuário com o sistema mais inteligente e menos cansativa. Na terceira consulta dessa figura, a referência pronominal ("desses estados") é resolvida através do contexto, isto é, o interpretador recupera os critérios da consulta anterior.

Qual é o nome e a área das cidades com população inferior a 55000 habitantes ?

```
select nome, area
from   c in cidade
where população < 55000
```

E das cidades a menos de 50 km do rio Tietê ?

```
select c.nome, area
from   c in cidade, r in rio
where distance(c,r) < 50 and r.nome = "Tietê"
```

E qual é a altitude **dessas cidades** ?

```
select altitude
from   c in cidade, r in rio
where distance(c,r) < 50 and r.nome = "Tietê"
```

Figura 3.12: Exemplos de consultas com elipse (segunda consulta) e referência pronominal (terceira consulta).

3.8.2 Ambigüidade

A ambigüidade é um dos maiores problemas em processamento de linguagem natural. Algumas palavras da língua portuguesa podem ter mais de um significado, como por exemplo:

"A manga é amarela"

O substantivo "manga" tanto pode significar uma fruta como a parte de uma camisa.

Esse tipo de ambigüidade pode ser eliminado em situações de domínio restrito [Wall84]. Por exemplo, em um banco de dados sobre flora, jamais "manga" se referiria à parte de uma camisa e sim à fruta.

Mesmo restringindo-se a um domínio específico, ainda podem ocorrer ambigüidades, mas de outra natureza: as ambigüidades estruturais. O caso C da figura 3.13 mostra uma sentença ambígua, mas que não tem palavras com dúbio significado (em relação ao domínio da aplicação).

O tradutor é capaz de resolver alguns tipos de ambigüidade, como demonstra a figura 3.13. No caso A, a sentença é completa, isto é, o usuário detalhou completamente a sua requisição. A consulta B é incompleta, isto é, o usuário não especificou a que classe pertence o atributo altitude; entretanto a sentença não é ambígua, pois "altitude" só é atributo de "cidade" e o tradutor assim consegue relacioná-lo corretamente. Já no caso C a sentença é incompleta e ambígua pois o atributo "área" pode pertencer a quaisquer das duas classes mencionadas anteriormente.

A: Qual é a altitude da **cidade** cuja população do **estado** é maior que 100000 e cuja área da **cidade** é inferior a 100000 m_2 ?

B: Qual é a **cidade** cujo **estado** tem área superior a 500000 m_2 e com **altitude** inferior a 1000 m ?

C: A: Qual é a altitude da **cidade** cuja população do **estado** é maior que 100000 e cuja **área** é inferior a 100000 m_2 ? (usuário)

O atributo área pertence a cidade ou estado ? (sistema)

cidade (usuário)

Figura 3.13: Exemplos de sentenças ambíguas (última) e não ambíguas.

O algoritmo para a verificação e resolução de ambigüidades adotado é mostrado na figura 3.14.

Se a sentença é incompleta:
    Se há apenas uma classe mencionada anteriormente ao atributo:
        Não há ambigüidade e o atributo pertence a essa classe.
    Senão
        Verifica-se a quais classes anteriormente mencionadas o atributo pode pertencer
        Se tal atributo pode pertencer a uma única classe
            Não há ambigüidade e o atributo pertence a essa classe.
        Senão
            A sentença é ambígua e pergunta-se ao usuário a respeito da sua intenção, isto é, a qual classe pertence o atributo.

Figura 3.14: Algoritmo para verificação e resolução de ambigüidades utilizado.

3.8.3 - Conjunção e Disjunção

Nem sempre a conjunção "e" deve ser convertida diretamente para "and". A correta conversão dependerá da estrutura gramatical da sentença. Observe o caso A da figura 3.15. No resultado A.1, o "e" e a vírgula foram traduzidos para "and", resultando em uma consulta errônea, pois é impossível que uma cidade tenha dois nomes distintos. O tradutor deve então, neste caso, converter o "e" e a vírgula para "or" e formar a consulta A.2 corretamente.

Já o caso B, de estrutura gramatical diferente do caso A, o "e" e o "ou" devem ser convertidos diretamente para "and" e "or" respectivamente, pois trata-se claramente de conjunções e disjunção de critérios.

A: Qual é a área das cidades de Campinas, Santos e Bauru ?

A.1:
```
select área
from   c in cidade
where  nome = "Campinas" and nome = "Santos" and nome = "Bauru"
```

A.2:
```
select área
from   c in cidade
where  nome = "Campinas" or nome = "Santos" or nome = "Bauru"
```

B: Qual é a cidade cuja área é maior que 10000 **e** com população superior a 85000 **ou** cuja área é menor que 5000 **e** com menos de 8000 habitantes ?

B.1:
```
select c
from   c in cidade
where  área > 10000 and população > 85000 or área < 5000 and
       população < 8000
```

Figura 3.15: Como o tradutor trata conjunções e disjunções.

3.9 - Conclusões

Este capítulo destinou-se ao detalhamento da implementação do tradutor, porém sem se preocupar com as características particulares que as consultas espaciais possuem. Essas características serão analisadas no capítulo seguinte.

A funcionalidade de cada módulo foi analisada, bem como os fenômenos lingüísticos tratados pelo tradutor. A referência pronominal e a elipse, embora de implementação problemática, proporcionam uma interação inteligente e menos cansativa; já a ambigüidade, também de difícil resolução, faz cair o desempenho na recuperação dos dados, pois o usuário é perguntando sobre a sua intenção, já que é impossível a inferência dessa intenção por parte do tradutor sem a intervenção do usuário.

A figura 3.16 sumariza os passos principais na tradução da consulta "Selecione a área média, o nome da cidade e o partido do prefeito cujo sexo é masculino". Isto é, o analisador constrói a árvore de derivação da sentença e dela extrai os itens semânticos (representados pelas variáveis SELECT, FROM, WHERE, COMPARADOR, VALOR e CONECTIVO, que estão estruturados em forma de listas do PROLOG). O interpretador valida esses itens e constrói o relacionamento entre as classes. O gerador de consultas em LEGAL, com base nos itens semânticos processados pelo interpretador, monta a consulta em LEGAL correspondente. Como o atributo "nome" pertence a ambas as classes ele é prefixado para contornar a ambigüidade.

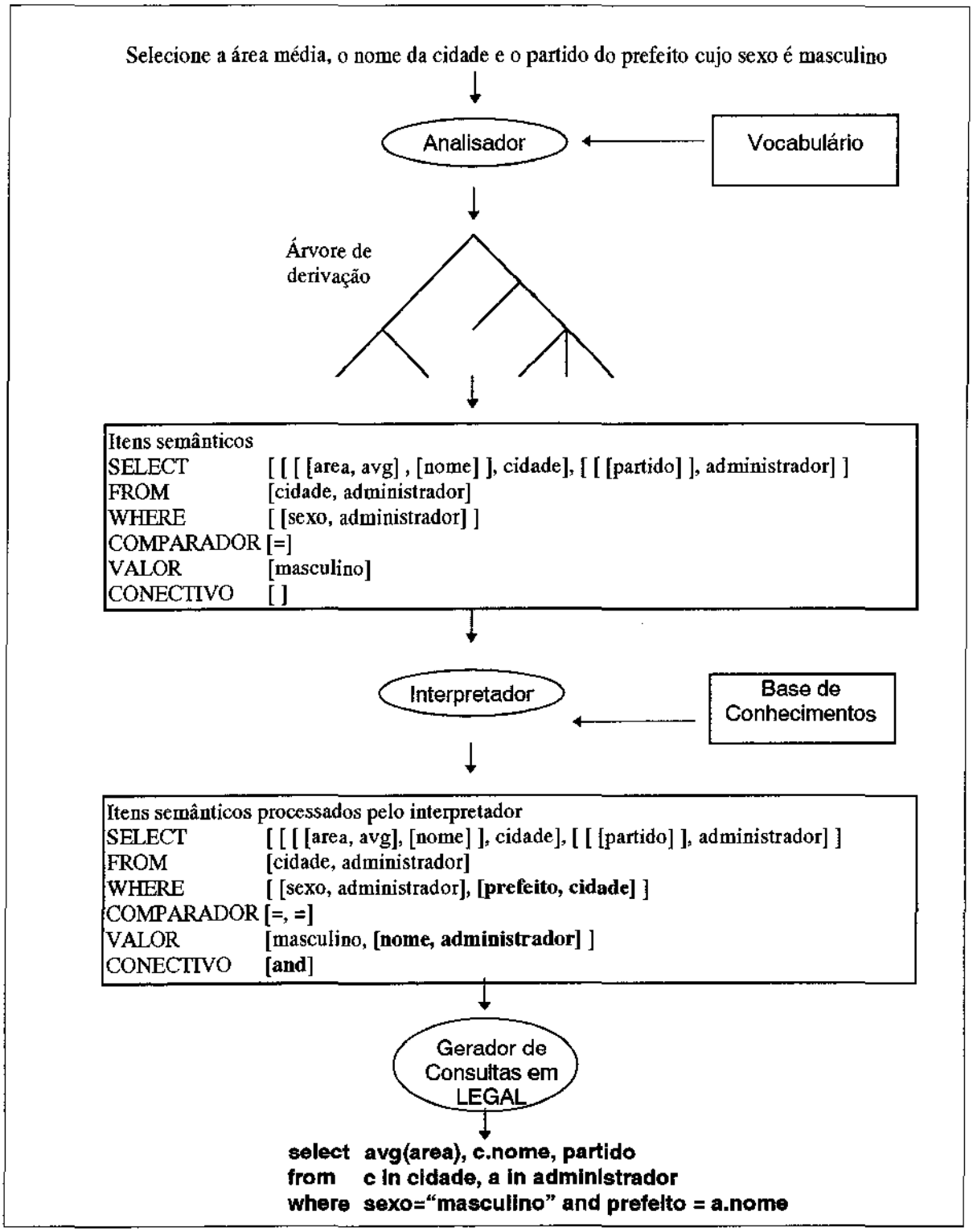


Figura 3.16: Os principais passos na tradução de uma consulta.

Capítulo 4 - Consultas Espaciais

Este capítulo descreve as particularidades impostas pelas consultas espaciais e que devem ser implementadas pelo tradutor.

[BM92] apresenta um conjunto de tipos de consultas que um SIG deve suportar. Devido às limitações da linguagem LEGAL, somente alguns tipos são compreendidos pelo tradutor. Por exemplo as consultas: "Qual é o menor caminho entre as cidades de Campinas e Londrina?" e consultas de orientação, como "Quais são as cidades ao norte da cidade de Campinas?" não são suportadas pela linguagem LEGAL e, portanto, não são compreendidas pelo tradutor.

Desse conjunto, os seguintes tipos de consulta são suportados pelo tradutor:

- consultas não espaciais;
- consultas que envolvem relacionamentos espaciais;
- consultas que envolvem negação de termos que expressam relacionamentos espaciais; e
- disjunção de expressões que envolvem relacionamentos espaciais.

Consultas não espaciais envolvem apenas atributos convencionais, como por exemplo "Qual é a altitude da cidade com mais de 1000 habitantes?".

As consultas que envolvem relacionamentos espaciais podem conter restrições métricas ("Selecione as cidades a menos de 50 km do estado de Minas Gerais?") ou topológicas ("Quais são as cidades cortadas pelo rio Paraopeba?") ou ambas ("Recupere as cidades cortadas pelo rio Paraopeba e a menos de 100 km da cidade de Belo Horizonte"). Além disso, o tradutor também suporta critérios que envolvam relacionamentos espaciais e restrições não espaciais ("Quais as cidades cortadas pelo rio Paraopeba com mais de 10000 habitantes?").

O tradutor compreende certas palavras em LN que podem freqüentemente ser utilizadas na aplicação para expressar noções topológicas e que servem para exemplificar a problemática na conversão dessas palavras em operadores topológicos da linguagem LEGAL. Essas palavras foram agrupadas de acordo com a noção topológica por elas expressas, como demonstra a figura 4.1.

A conversão desses grupos de palavras em LN que expressam relacionamentos topológicos para os seus respectivos operadores topológicos na linguagem LEGAL é analisada na seção 4.2.

Relacionamento topológico	Cruzamento	Adjacência	Intersecção	Inclusão
Palavra ou expressão em LN	cruzada atravessada cortada que cruza que atravessa que corta	vizinha adjacente colada	intercepta que interseciona	contidas inclusas sintagmas preposicionais (do estado)
Exemplo de consulta	Quais as cidades **cortadas** pela via Dutra?	Quais as cidades **vizinhas** à cidade de Campinas?	Quais os lagos que **intersecionam** o município de Foz do Iguaçu?	Recupere os municípios **do estado** de São Paulo.

Figura 4.1: Exemplos de palavras contidas no vocabulário, os respectivos relacionamentos topológicos por elas expressos e consultas associadas.

Sentenças negativas são sentenças em que a descrição do relacionamento topológico vem precedida pelo advérbio de negação "não", como por exemplo a sentença "Quais são as estradas que não atravessam cidades com mais de 10000 habitantes?".

O tradutor também suporta a disjunção (e conjunção) de expressões que envolvem relacionamentos espaciais, como "Qual é a cidade atravessada pela rodovia Transamazônica **ou** atravessada pelo rio Purus?".

4.1 - Relacionamentos Topológicos

Esta seção é uma breve introdução aos operadores topológicos suportados pela linguagem LEGAL: "cross", "disjoint", "inside", "overlap" e "touch", para os quais as palavras da figura 4.1 deverão ser convertidas. A análise desses operadores será feita considerando-se um espaço topológico de dimensão dois. Os elementos topológicos são de três tipos: ponto, linha e polígono, onde cada linha tem **dois pontos terminais** ou então é circular, mas que não pode auto-interceptar-se; um polígono é uma superfície, uma região simples, sem buracos e que não é uma união de conjuntos de pontos disjuntos entre si [CCH+96].

Considere as seguintes definições [Cam95]:

Seja dim(Ω) a dimensão de um conjunto de pontos (Ω), temos:

$\dim(\Omega) = \lambda \quad \leftrightarrow \quad \Omega = \varnothing$.

$\dim(\Omega) = 0 \quad \leftrightarrow$ Ω contém pelo menos um ponto e nenhuma linha ou polígono.

$\dim(\Omega) = 1 \quad \leftrightarrow$ Ω contém pelo menos uma linha e nenhum polígono.

$\dim(\Omega) = 2 \quad \leftrightarrow$ Ω contém pelo menos um polígono.

Seja δω a fronteira de um elemento topológico simples ω, temos:

$\delta\omega = \varnothing \quad \leftrightarrow$ ω é um ponto ou uma linha circular.

$\delta\omega = \{Pi,Pf\} \quad \leftrightarrow$ ω é linha não circular e Pi e Pf são os seus pontos terminais.

$\delta\omega = L \quad \leftrightarrow$ ω é uma região simples e L é uma linha circular.

Seja ω^0 o interior de um elemento topológico simples ω, temos:

$$\omega^0 = \omega - \delta\omega$$

Tendo em vista essas definições e notações, define-se o relacionamento topológico "cross" da seguinte maneira:

$$\omega 1 \text{ cross } \omega 2 \leftrightarrow (\omega_1 \cap \omega_2 \neq \omega_1) \wedge (\omega_1 \cap \omega_2 \neq \omega_2) \wedge$$
$$\dim(\omega_1^{\,0} \cap \omega_2^{\,0}) = (\max(\dim(\omega_1^{\,0}),\dim(\omega_2^{\,0})) - 1)$$

A figura 4.2 mostra exemplos do relacionamento "cross".

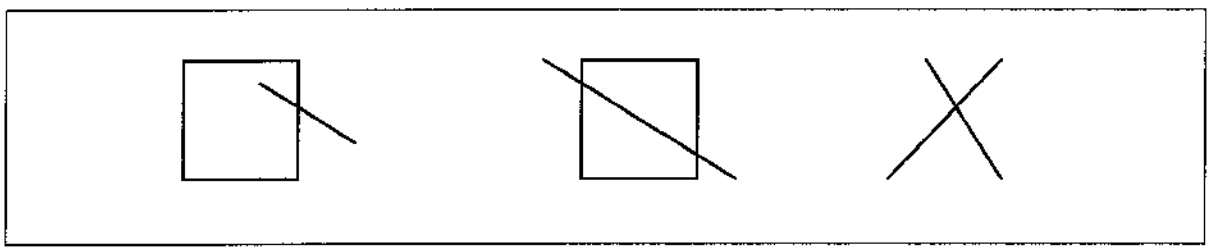


Figura 4.2: Exemplos do relacionamento "cross".

O relacionamento "disjoint" é definido da seguinte maneira:

$$\omega_1 \text{ disjoint } \omega_2 \leftrightarrow (\omega_1 \cap \omega_2) = \varnothing$$

A figura 4.3 mostra exemplos de "disjoint".

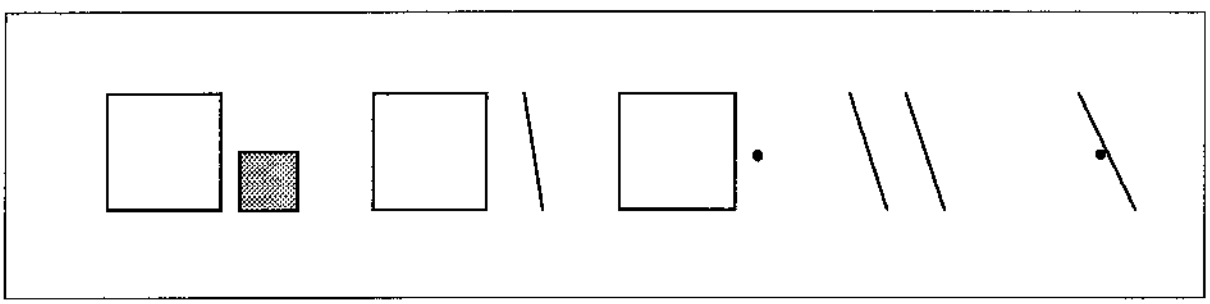


Figura 4.3: Alguns exemplos do relacionamento "disjoint".

Já "inside" é assim definido :

$$\omega_1 \text{ inside } \omega_2 \leftrightarrow (\omega_1^0 \cap \omega_2^0 \neq \varnothing) \wedge (\omega_1 \cap \omega_2 = \omega_1)$$

A figura 4.4 mostra exemplos de "inside".

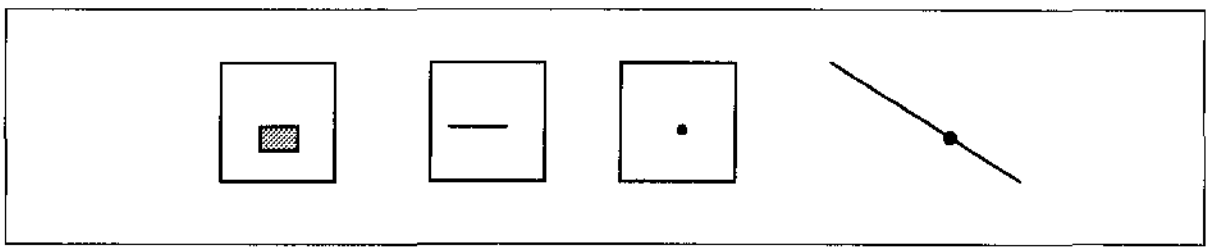


Figura 4.4: Exemplos do relacionamento "inside".

O relacionamento topológico "overlap" é definido como :

$$\omega 1 \text{ overlap } \omega 2 \leftrightarrow (\omega_1 \cap \omega_2 \neq \omega_1) \wedge (\omega_1 \cap \omega_2 \neq \omega_2) \wedge$$
$$\dim(\omega_1^0 \cap \omega_2^0) = \dim(\omega_1^0) = \dim(\omega_2^0)$$

A figura 4.5 mostra exemplos de tal relacionamento.

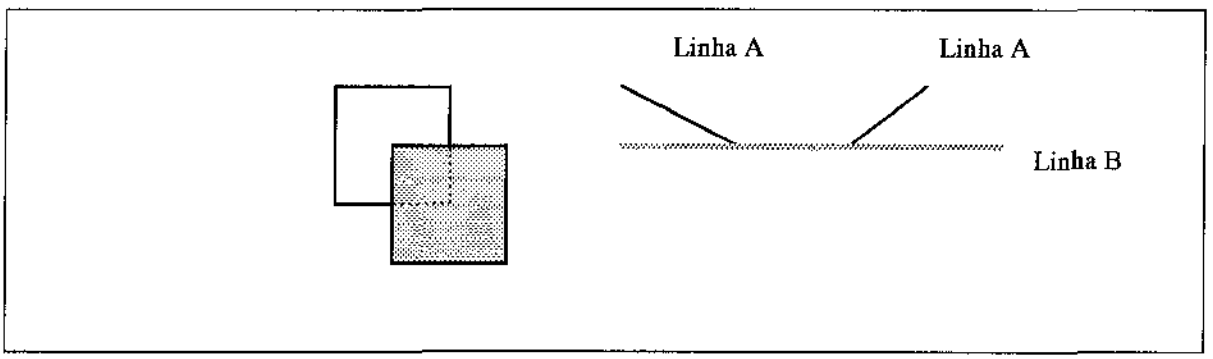


Figura 4.5: Exemplos do relacionamento "overlap".

Por fim, define-se o relacionamento topológico "touch" da seguinte maneira:

$$\omega_1 \text{ touch } \omega_2 \leftrightarrow (\omega_1 \cap \omega_2 \neq \varnothing) \wedge (\omega_1^{\,0} \cap \omega_2^{\,0} = \varnothing)$$

A figura 4.6 mostra exemplos desse relacionamento.

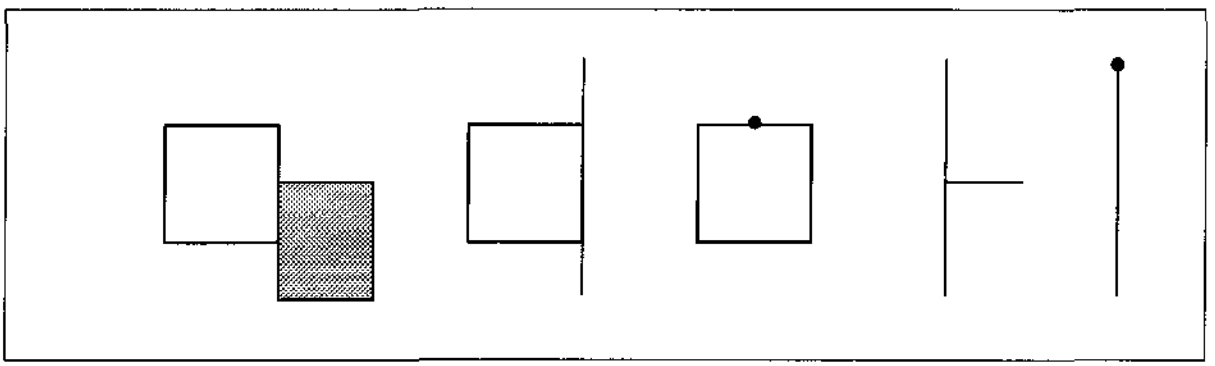


Figura 4.6: Exemplos de relacionamentos "touch".

4.2 - O Problema da Representação

Os relacionamentos topológicos expressos em LN não podem ser mapeados diretamente para os operadores topológicos da linguagem LEGAL. Por exemplo, nem sempre a palavra "cruzadas" na consulta "Que cidades são cruzadas pela via Dutra?" deve ser transformada em "cross". O objetivo desta seção é mostrar como essa conversão é problemática e propor idéias de como ela pode ser realizada com base na representação das classes de geo-objetos (ponto, linha ou polígono).

As próximas seções detalharão como cada grupo de palavras da figura 4.1 pode ser convertido para LEGAL. A conversão dessas palavras será baseada nos arranjos espaciais entre cada par de representações de geo-objetos que podem envolver a noção topológica por elas expressa. Esses arranjos serão separados em grupos de casos dependendo da representação dos geo-objetos envolvidos.

4.2.1 - Situações que Envolvem Cruzamentos

A palavras e as expressões em LN que expressam cruzamentos (cruzadas, atravessadas, cortadas, que cruzam etc.) não devem ser sempre convertidas para "cross". A figura 4.7 mostra alguns casos onde a noção de cruzamento está envolvida e que podem ser expressos em LN coerentemente de acordo com a aplicação. Por exemplo, a consulta "Quais são as cidades cortadas pelo estado de São Paulo" não faz sentido, pois não é coerente a relação de cruzamento entre um ponto e um polígono (supondo a representação de cidade como ponto e de estado como polígono). Nesses casos a conversão falha e o usuário é notificado de que não é possível tal relacionamento espacial com as classes envolvidas.

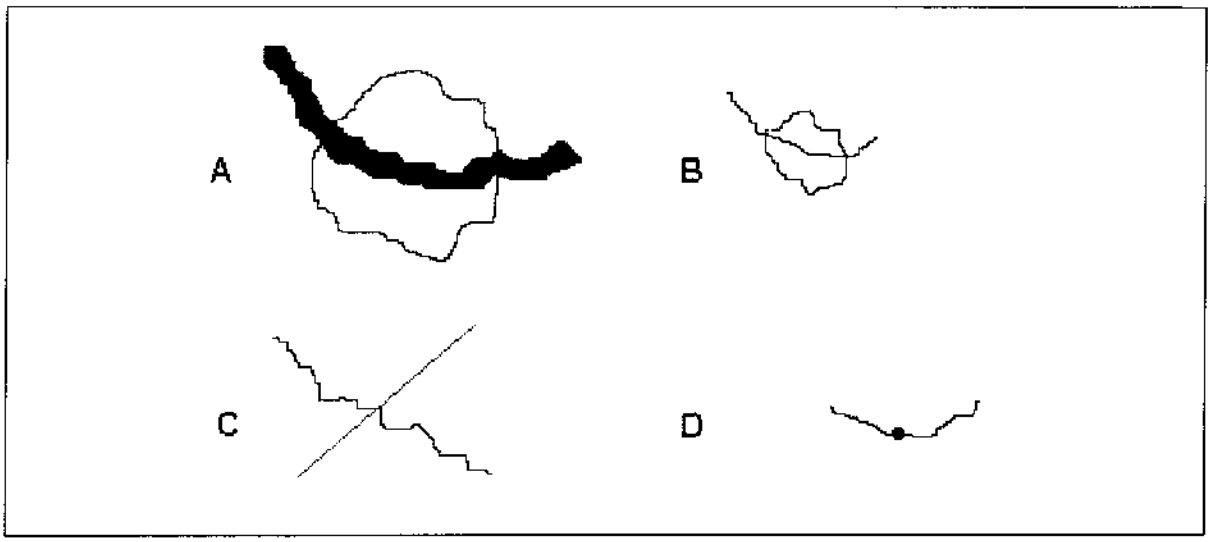


Figura 4.7: Exemplos de situações que envolvem a noção de cruzamento.

A consulta "Quais cidades são atravessadas pelo rio Tietê?" pode envolver, dependendo da representação dos geo-objetos "cidade" e "rio", os casos A, B e D da figura 4.7, porém cada caso requer conversões adequadas.

Quando os geo-objetos são representados por polígonos (caso A), as palavras e expressões em LN que representam cruzamentos devem ser mapeadas para "overlap". Logo a consulta anterior seria assim convertida:

```
select c
from   c in cidade, r in rio
where c overlap r and r.nome = "Tietê"
```

Já o caso B, onde um geo-objeto é representado por um polígono e o outro por uma linha, mostra que o termo em LEGAL que deve ser usado é "cross".

Em C, ambos geo-objetos são representados por linhas e nessa situação também "cross" deve ser utilizado. Consultas do tipo "Selecione as estradas cortadas pelo rio Tietê" podem se associar a esse caso.

No caso D um geo-objeto é representado por uma linha e o outro por um ponto, e o operador topológico apropriado é "inside".

4.2.2 - Situações que Envolvem Adjacência

As palavras em LN que exprimem adjacência (vizinhas, adjacentes, coladas etc.) também requerem tratamento especial em relação à representação dos geo-objetos, para que a correta conversão seja realizada.

A figura 4.8 mostra situações onde a noção de adjacência pode ser aplicada.

No caso A ambos geo-objetos são representados por polígonos e "touch" deve ser utilizado. A consulta "Quais são os estados vizinhos ao estado do Paraná" deve então ser convertida nesse caso para:

```
select  e1
from    e1 in estado, e2 in estado
where   e1 touch e2 and e2.nome = "Paraná"
```

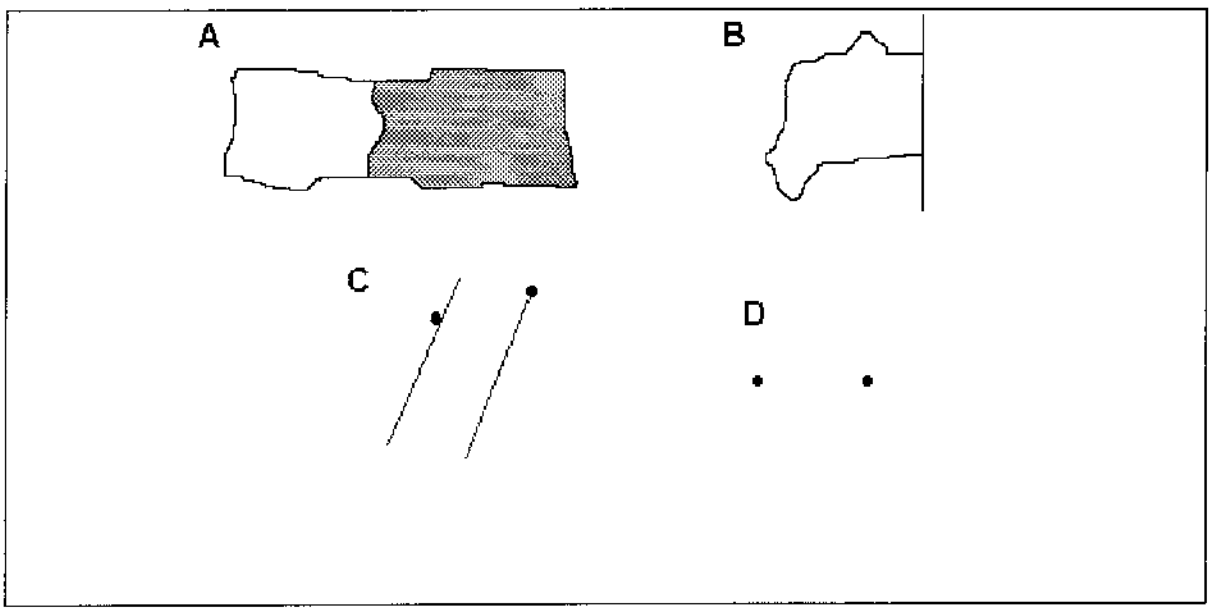


Figura 4.8: Situações onde a noção de adjacência está envolvida.

A consulta "Quais são as cidades adjacentes à rodovia BR040?" pode envolver os casos B e C. Em B, um geo-objeto é representado por uma linha e o outro por um polígono; logo "touch" deve ser usado. Entretanto, o primeiro arranjo entre um ponto e uma linha do caso C requer uma aproximação por distância (distance < 1, por exemplo), e o segundo a utilização de "touch", logo a consulta anterior deve ser convertida para:

```
select  c
from    c in cidade, e in estrada
where   (distance(c,e) < 1 or c touch e) and e.nome = "BR040"
```

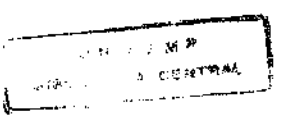
O caso D dessa figura mostra um caso semelhante ao que acontece com as cidades de Belo Horizonte e Contagem em Minas Gerais. Neste caso, as duas cidades são representadas por pontos (como em alguns mapas de Minas Gerais); apesar das duas cidades não estarem "fisicamente" coladas no mapa, existe a relação de adjacência entre elas, que pode ser mostrada em mapas de maior escala. Nesse caso uma aproximação por distância deve ser utilizada, como por exemplo: distance(c1,c2) < 1 (onde c1 e c2 são "aliases" da classe cidade).

4.2.3 - Situações que Envolvem Intersecção

As consultas em LN que expressam intersecção podem envolver os arranjos espaciais mostrados na figura 4.9.

No caso A ambos geo-objetos são representados por polígonos, e deve-se utilizar uma disjunção dos operadores topológicos "inside" e "overlap", como na consulta "Mostre os lagos que intersecionam o município de Campinas".

```
select  l
from    l in lago, c in cidade
where   (l inside c or l overlap c) and c.nome = 'Campinas'
```

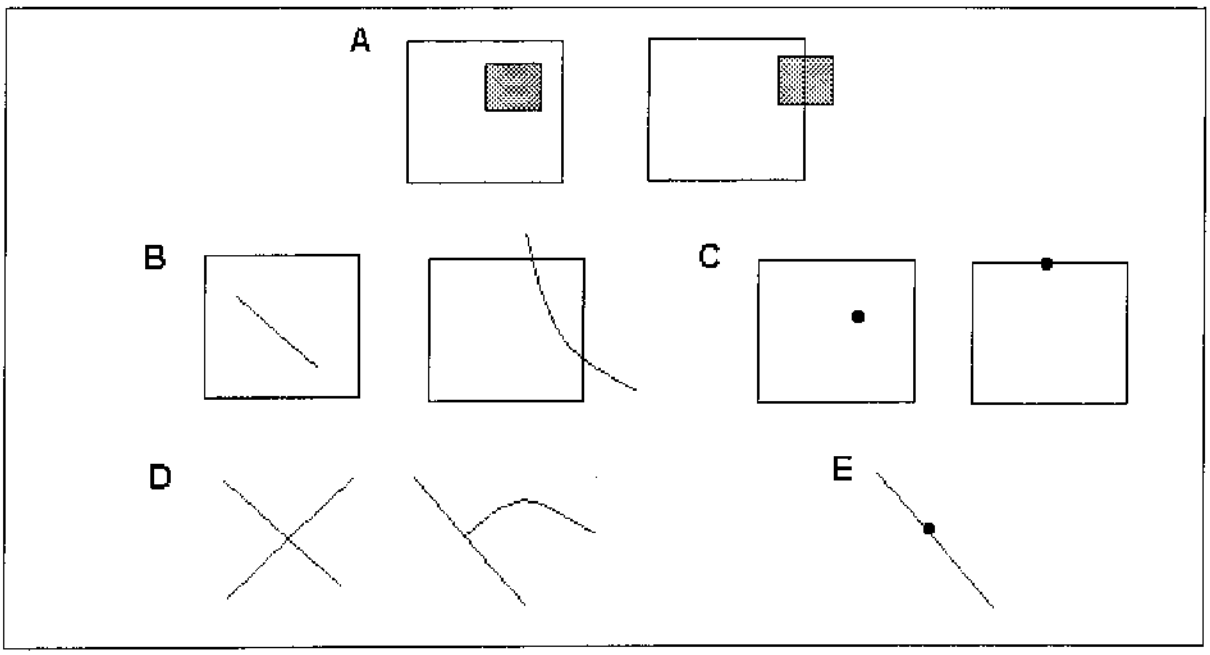


Figura 4.9: Arranjos espaciais que podem ser envolvidos por consultas que expressam intersecção.

Em B temos intersecção entre linhas e polígonos e uma disjunção entre "inside" e "cross" deve ser usada. A consulta "Quais estradas intersecionam o estado de São Paulo?" se relaciona a esse caso.

No caso C, há intersecção entre um polígono e um ponto e "inside" e "touch" devem ser utilizados. A consulta "Quais lagos intersecionam o estado de São Paulo?" envolve esse caso quando "lago" é representado por um ponto e "cidade" por um polígono.

A relação de intersecção entre duas linhas é mostrada em D, e uma disjunção entre "cross" e "touch" deve ser utilizada. Um exemplo de consulta relacionada a esse caso é "Quais as estradas que intersecionam a rodovia BR040?".

O caso E, ilustra intersecção entre linha e ponto e o operador topológico "inside" deve ser usado. A consulta "Quais cidades são interceptadas pela rodovia BR050?" pode envolver esse caso.

4.2.4 - Situações que Envolvem Inclusão

A figura 4.10 mostra alguns casos onde a noção de inclusão pode ser expressa em uma consulta em LN, como por exemplo: "Selecione os municípios do estado do Pará" (caso A), "Quais são as estradas do estado do Pará?" (caso B) e "Quais as cidades do estado do Pará?" (caso C). Em todos esses casos somente "inside" deve ser utilizado. D é um exemplo de caso onde, apesar de existir a relação de inclusão entre os dois geo-objetos, ela não pode ser expressa com sentido em LN sobre a aplicação, como a consulta "Quais as cidades contidas na estrada BR040?"; nesse caso o tradutor responde "Não é possível a relação de inclusão entre a classe cidade de dimensão zero e a classe estrada de dimensão um !".

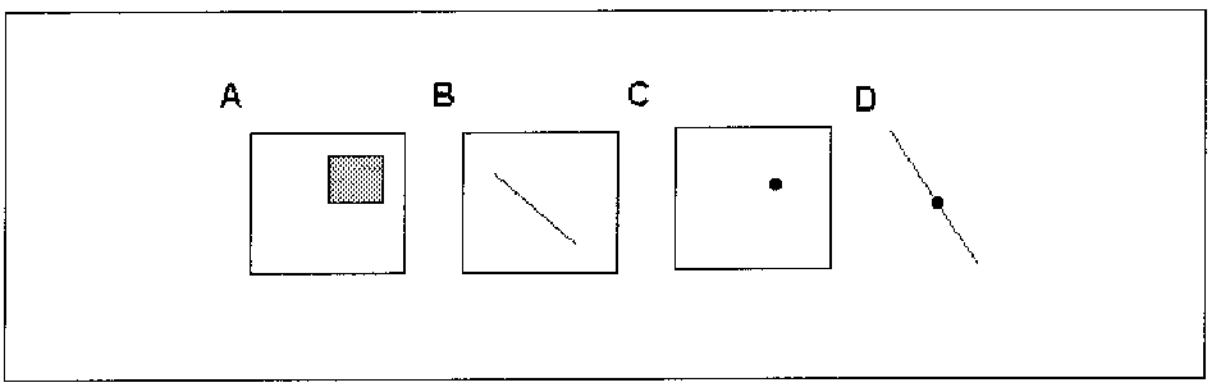


Figura 4.10: Situações que envolvem a noção de inclusão.

4.2.5 - Expressões Negativas

O advérbio de negação "não", embora pouco freqüentemente, pode ser utilizado pelo usuário na formulação de consultas, como por exemplo "Quais são as estradas do estado do Rio de Janeiro que não cruzam a via Dutra?". As situações que envolvem expressões negativas e o seu devido tratamento são analisados a seguir.

4.2.5.1 - Situações que Envolvem Negação de Cruzamentos

Na figura 4.11 estão ilustrados alguns casos que envolvem a negação da noção de cruzamentos. No caso A, onde ambos geo-objetos são representados por polígonos, deve-se usar uma disjunção entre "touch" e "disjoint", como na consulta: "Quais são as cidades não atravessadas pelo rio Pará?".

A consulta anterior também pode ser relacionada aos casos B e D. Em B, uma disjunção entre "touch" e "disjoint" deve ser aplicada, enquanto que em D somente "disjoint" deve ser utilizado.

A relação "não cruzadas" entre linhas, mostrada no caso C, também requer o uso de "disjoint".

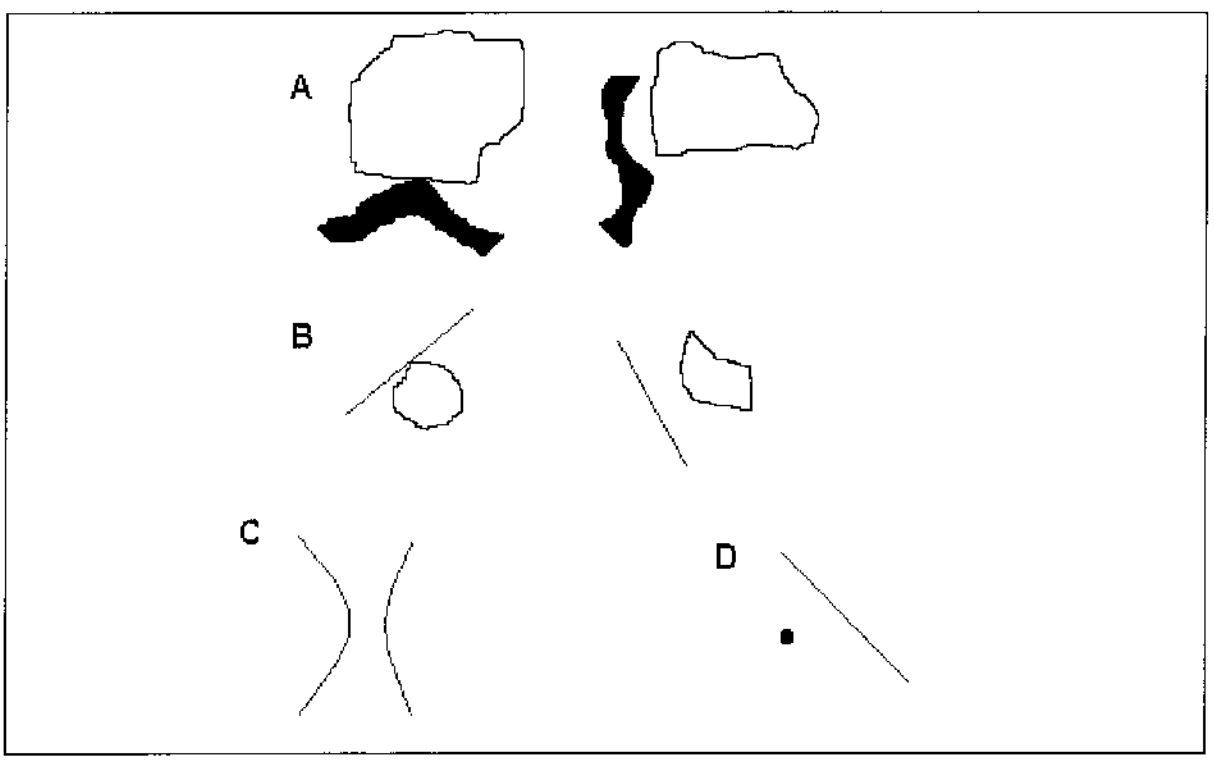


Figura 4.11: Situações que envolvem a negação da noção de cruzamentos, que podem ser expressas em LN.

4.2.5.2 - Situações que Envolvem Negação de Adjacência

Exemplos de situações que envolvem negação de adjacência e que podem ser expressas em LN estão ilustradas na figura 4.12.

Uma disjunção entre os operadores topológicos "inside", "overlap" e "disjoint" deve ser aplicada em situações correlatas ao caso A , com consultas do tipo "Quais são os lagos não adjacentes à cidade de Campinas?".

A noção de não adjacência entre um polígono e uma linha é apresentada no caso B, onde o uso da disjunção entre "inside", "cross" e "disjoint" levará aos resultados esperados. A consulta "Quais as estradas não adjacentes à cidade de Campinas?" é um exemplo de expressão em LN associada a esse caso.

A expressão de consulta em LN anterior também pode ser utilizada no caso C, porém a consulta em LEGAL deve conter a disjunção entre "inside" e "distance > 1".

E quando ambos geo-objetos são representados por pontos, como no caso D, deve-se utilizar uma aproximação por distância, como por exemplo:

"Quais são as cidades não vizinhas à cidade de Curvelo?"

```
select e1
from   e1 in cidade, e2 in cidade
where  distance(e1,e2) > 1 and e2.name = 'Curvelo'
```

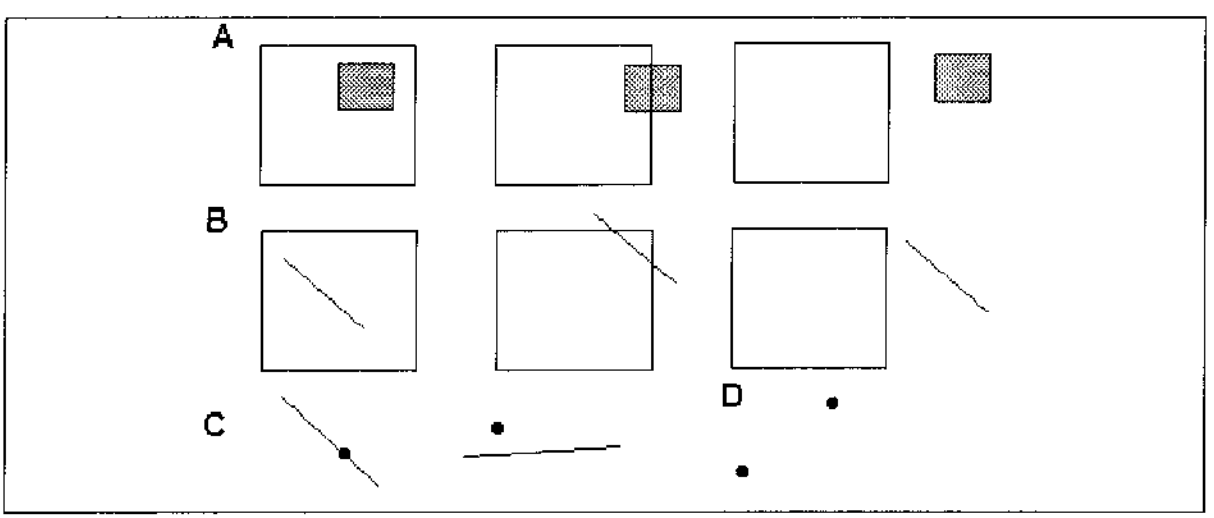


Figura 4.12: Situações que envolvem a negação da noção de adjacência e que fazem sentido quando expressas em LN.

4.2.5.3 - Situações que Envolvem Negação de Intersecção

Exemplos de relações de não intersecção entre dois geo-objetos que podem ser expressas em LN estão ilustradas na figura 4.13. O caso A mostra a relação de não intersecção entre dois polígonos, onde o uso da disjunção entre "touch" e "disjoint" deve ser usada. Esse caso (juntamente com o caso C) se aplica a consultas do tipo: "Quais os lagos que não intersecionam o município de Campinas?". Porém o caso C (polígono/ponto) requer o uso somente de "disjoint".

O caso B mostra essa mesma relação entre uma linha e um polígono e, como anteriormente, uma disjunção entre "touch" e "disjoint" deve ser utilizada. A esse caso se relacionam consultas do tipo "Quais as estradas que não têm intersecção com a cidade de Betim?". O caso E é análogo ao caso B em relação às consultas a ele relacionadas, porém "disjoint" deve ser utilizado.

O caso D mostra duas linhas nessa relação, onde "disjoint" se aplica. A consulta "Quais as rodovias que não intersecionam a via Dutra?" é um exemplo desse caso.

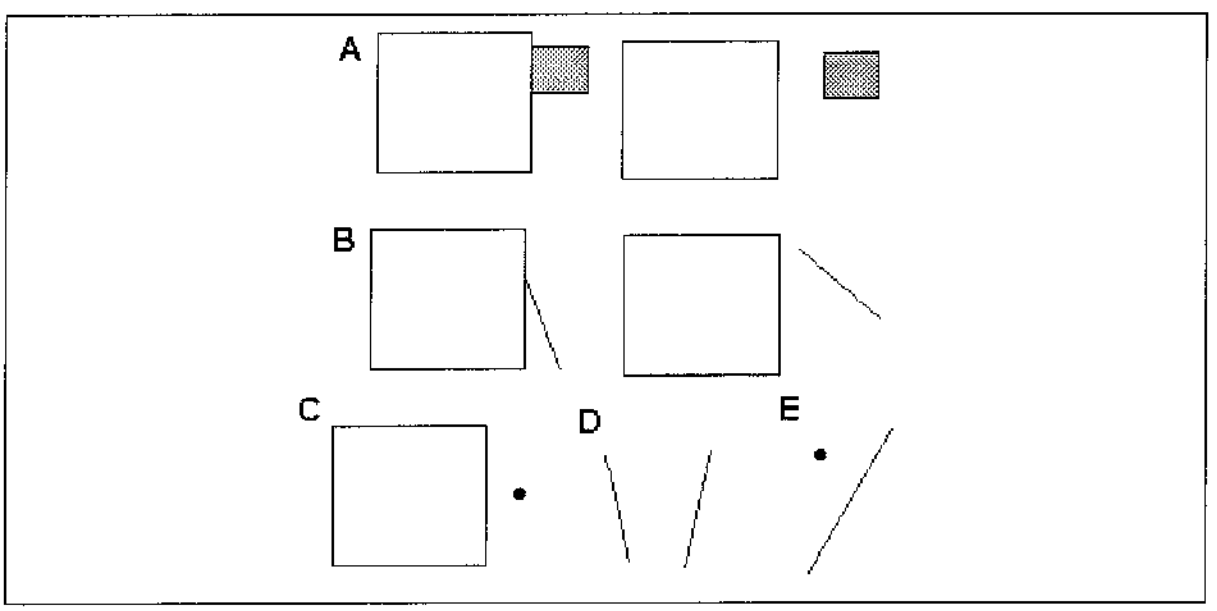


Figura 4.13: Exemplos da relação de não intersecção entre dois geo-objetos.

4.2.5.4 - Situações que Envolvem Negação de Inclusão

Consultas do tipo "Selecione os lagos não contidos no município de Itabirito" envolvem o caso A da figura 4.14, onde a disjunção entre "touch", "overlap" e "disjoint" é a correta tradução para a expressão "não contidas" (quando os geo-objetos envolvidos são polígonos), como mostra a consulta em LEGAL abaixo:

```
select  l.area
from    l in lago, m in municipio
where   (l touch m  or l overlap m or l disjoint m)
        and m.nome = "Itabirito"
```

Ao caso B se relacionam consultas do tipo "Quais as estradas não inclusas no estado de Minas Gerais?", e a disjunção dos operadores topológicos "touch", "cross" e "disjoint" pode ser aplicada nesse caso.

O caso C ilustra a relação de não inclusão entre ponto e polígono. Esses casos estão envolvidos em consultas do tipo "Quais as cidades não contidas no estado do Amazonas?". Uma disjunção entre "touch" e "disjoint" pode ser utilizada nesse caso.

O caso D mostra essa mesma relação entre um ponto e linha, porém não existe consulta em LN coerente que expresse esta situação (pelo menos na aplicação escolhida). Por exemplo a consulta "Quais as cidades não contidas na via Dutra?" não faz sentido.

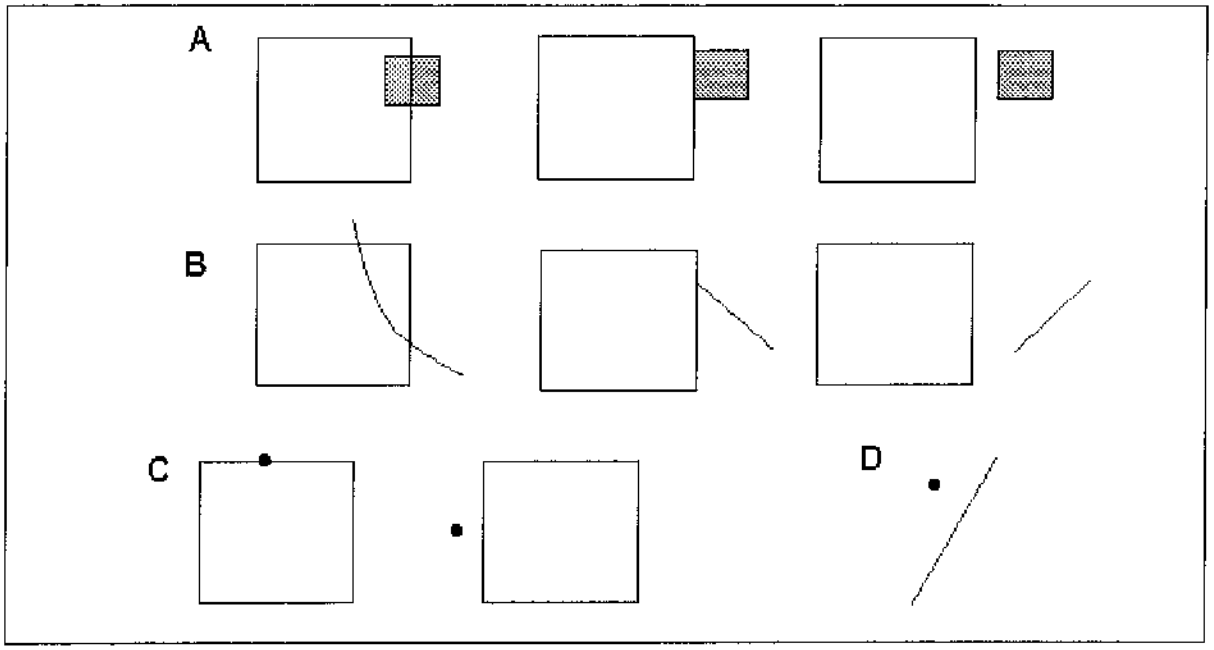


Figura 4.14: Situações que envolvem a noção de não inclusão.

4.2.6 - A Implementação

As particularidades que a representação das classes de geo-objetos impõem sobre a conversão das consultas em LN para consultas correspondentes em LEGAL, foram implementadas, de acordo com os casos vistos anteriormente. Para tanto as seguintes informações foram necessárias:

- a representação de cada classe de geo-objetos envolvida no relacionamento topológico. Essa representação consta na base de conhecimentos com a seguinte notação: "d0" (ponto), "d1" (linha) e "d2" (polígono);

- sentença positiva ou negativa; a sentença será considerada negativa quando da presença do advérbio de negação "não" precedendo a palavra que expressa o relacionamento topológico;

- a noção topológica expressa pela palavra em LN.

Por exemplo na consulta: "Quais são as cidades cortadas pelo rio Tietê?" o analisador extrai as seguintes informações: sentença positiva e palavra que expressa o relacionamento topológico "cortadas", o que leva à noção de cruzamento. A base de conhecimentos informa que a representação para cidades é "d0" e para rio "d1". O interpretador toma essas informações e retorna a correta conversão (inside(cidade,rio) and rio.nome="Tietê"). O código em PROLOG relativo a essa implementação está no apêndice da dissertação.

Conforme dito anteriormente, a título de simplificação, admitiu-se neste trabalho a existência de apenas uma única representação para cada classe de geo-objetos, mas uma classe de geo-objetos pode ter mais de uma representação; por exemplo, uma cidade pode ser representada por um polígono, e em um mapa de menor escala por um ponto. Essa multiplicidade de representações gera problemas na conversão, isto é, qual representação deverá servir de base para a correta conversão de LN para LEGAL? As sugestões para a solução desse problema estão descritos no capítulo 5 (Conclusões e Trabalhos Futuros).

4.2 - Conclusões

Este capítulo mostrou o potencial do tradutor em relação ao tratamento das consultas espaciais, as particularidades que a recuperação de dados em SIG possui e que a diferencia das consultas convencionais. Como visto, o tradutor trata consultas que envolvem restrições métricas e topológicas. Um aspecto importante é a conversão de palavras em LN que expressam relacionamentos topológicos para os respectivos termos em LEGAL, onde a representação de cada classe de geo-objetos é informação crucial para tal conversão.

Capítulo 5 - Conclusões e Trabalhos Futuros

5.1 - Conclusões

Esta dissertação descreveu a implementação de um tradutor de LN para a linguagem de consulta LEGAL. Para tanto, mostrou-se, no capítulo 1 (Introdução) as vantagens que uma interação em LN proporciona a usuários leigos em computação e/ou que às vezes utilizam infreqüentemente um computador, isto é, usuários típicos de um SIG. No capítulo 2 (Processamento de Linguagem Natural, Sistemas de Informação Geográfica e LEGAL) os conceitos básicos relativos às principais áreas envolvidas no trabalho foram apresentados, conceitos esses que servem de base para o entendimento dos capítulos posteriores. Os módulos constituintes do tradutor foram detalhados no capítulo 3 (O Tradutor), bem como os problemas lingüísticos tratados pelo tradutor. As particularidades que as consultas espaciais possuem foram abordadas no capítulo 4 (Consultas Espaciais) e, por fim, este capítulo (Conclusões e Trabalhos Futuros) mostra as conclusões sobre a pesquisa desenvolvida nessa dissertação e os trabalhos futuros que aperfeiçoarão o tradutor e que construirão a interface multimodal, objetivo principal dessa linha de pesquisa.

Apesar do sistema desenvolvido ser um protótipo experimental, percebe-se a grande potencialidade que o tradutor possui, principalmente em se tratando de usuários de SIG. Os pontos fortes do uso de LN em SIG são a conversão automática de palavras em LN que expressam noções topológicas para critérios de consulta em LEGAL, critérios estes que podem ser complicados dependendo da situação, como a disjunção de até três operadores topológicos em certos casos. Outra grande facilidade é a construção do relacionamento entre classes automaticamente, bem como a criação de "aliases" para as classes, quando necessário. Para um usuário leigo em computação seria difícil a criação de tais

relacionamentos, pois esses usuários provavelmente não saberiam o que é chave primária ou estrangeira. Além disso, a correção ortográfica melhora o desempenho dos usuários, tendo em vista que usuários leigos cometem muitos erros de digitação. O suporte a sentenças elípticas e com referência pronominal possibilita uma interação com o usuário mais inteligente e menos cansativa, já que ele pode abreviar as suas requisições omitindo termos compreensíveis pelo contexto ou por informações contidas na própria sentença. Entretanto, essas vantagens só serão comprovadas quando da implementação final da interface multimodal, seguido de experimentos práticos com usuários de SIG. Apesar das vantagens que uma interação em LN proporciona, temos que levar em conta também as suas desvantagens, como a opacidade da cobertura lingüística, o desconhecimento da cobertura do domínio, a ambigüidade inerente à LN, a dificuldade em se alcançar a portabilidade em relação à diferentes aplicações (questão abordada a seguir). Além disso, a implementação do tradutor foi bastante trabalhosa, especialmente na definição e implementação das regras gramaticais; ademais, a resolução de fenômenos lingüísticos e a conversão de palavras em LN para operadores topológicos da linguagem LEGAL não foram triviais.

Podemos sumarizar as contribuições deste estudo como:

- a implementação de um tradutor de LN para a linguagem LEGAL: este é um passo em direção à implementação de uma interface multimodal para SIG, propiciando um ambiente de interação eficiente e inteligente para os usuários de SIG; além disso, a implementação ilustrou os problemas e a potencialidade que uma interação em LN para SIG possui;

- a aplicação de técnicas de processamento de linguagem natural em SIG: apesar das vantagens, tem sido pouco utilizada.

5.2 - Trabalhos Futuros

Os trabalhos futuros estão divididos em duas partes: extensões ao tradutor e extensões gerais. As extensões ao tradutor envolvem a extensão gramatical e léxica, um estudo detalhado do contexto da interação, o gerenciamento de múltiplas representações para as classes de geo-objetos e o estudo e a implementação de métodos para propiciar portabilidade ao tradutor em relação à aplicações diferentes. A outra parte envolve a própria construção da interface multimodal e testes empíricos sobre esta.

5.2.1 - Extensões ao tradutor

As principais extensões que devem ser feitas ao tradutor são:

- **Extensão gramatical e léxica:** a extensão da gramática e do vocabulário, isto é, ampliar o subconjunto da LN coberto pelo tradutor, para que o usuário tenha mais facilidade para se expressar. Essa extensão deve permitir ao usuário a formulação de consultas mais complexas, como as consultas "aninhadas" do SQL (a cláusula "WHERE" contendo outras consultas), além de outras formas de elipse e referência pronominal. Tal extensão não é tarefa difícil; porém, no caso das consultas "aninhadas", a construção da consulta em LEGAL, realizada pelo gerador de consultas em LEGAL, será mais complicada. A adição de novas palavras que expressem relacionamentos topológicos requererá o estudo da noção topológica que elas expressam e como elas devem ser convertidas para LEGAL;

- **Contexto Envolvendo Várias Sentenças Anteriores:** será que é vantajosa a implementação de um contexto envolvendo várias sentenças anteriores, tendo em vista que tal tarefa é extremamente difícil [Wall84] e que os usuários de SIG geralmente trabalham com os resultados das suas consultas [Egen92]? Um estudo mais detalhado sobre essa questão deve ser realizado para se definir a relação custo/benefício de tal implementação. [GS86] aborda questões relativas a esse assunto;

- **Múltiplas Representações:** como dito anteriormente, admitiu-se a existência de apenas uma única representação para cada classe de geo-objetos. O tratamento de múltiplas representações, por parte do tradutor, requererá:

- que o usuário indique sobre qual mapa a consulta será realizada ("Quais as cidades cortadas pelo rio Pará, no mapa do estado de Minas Gerais", por exemplo); se o usuário não especificar o mapa, procura-se no contexto o mapa atual;

- a base de conhecimentos deverá conter informações sobre os mapas utilizados na aplicação, isto é, informações sobre a representação de cada classe de geo-objetos em cada mapa;

- a utilização do qualitativo "ON MAP" na construção da consulta correspondente em LEGAL, se necessário.

- **Portabilidade:** atualmente, grande parte das pesquisas sobre consultas em LN para bancos de dados são a respeito de portabilidade [ART95c].

Das cinco partes do tradutor, três são dependentes da aplicação: a base de conhecimentos, o vocabulário e o analisador; somente o interpretador e o gerador de consultas em LEGAL são independentes, pois a verificação dos itens semânticos extraídos pelo analisador, de acordo com informações da base de conhecimentos (qualquer que seja a base de conhecimentos) não depende da aplicação; as demais atribuições do interpretador também são independentes. O gerador de consultas em LEGAL também é independente, pois as regras de sintaxe da linguagem LEGAL são independentes da aplicação. A base de conhecimentos, por representar a estrutura da aplicação escolhida, é totalmente dependente da aplicação. Já o analisador é parcialmente dependente, pois certas regras gramaticais podem ser utilizadas em várias aplicações diferentes; por exemplo, a regra: "Recupere o $atributo_1$,...,$atributo_n$ da $classe_i$ onde $condição_1$ e/ou ... e/ou $condição_m$" pode ser utilizada praticamente em qualquer aplicação. O vocabulário pode ser dividido em duas partes: uma relativa ao domínio da aplicação e outra invariável. A parte invariável corresponde às palavras comumente utilizadas na língua portuguesa (artigos, preposições, pronomes etc.); a outra parte é relativa aos nomes de classes e atributos pertencentes à aplicação, mais os seus sinônimos. Tendo em vista essas características, algumas soluções podem ser aplicadas para a reconfiguração desses módulos de acordo com uma nova aplicação, como mostrado a seguir.

Possíveis Soluções: uma solução inicial seria a alteração do código dependente da aplicação por um programador. Tal solução não é viável tendo em vista que tal programador tem que ser especialista em processamento de linguagem natural, além de ter que conhecer profundamente o sistema.

Soluções mais inteligentes podem ser encontradas tomando como base as características de cada módulo do tradutor.

Inicialmente, dois métodos podem ser aplicados para a reconfiguração da base de conhecimentos e do vocabulário. No primeiro, o tradutor, quando da migração para outra aplicação, pergunta ao administrador do banco de dados (ABD), sobre detalhes da estrutura do novo banco de dados, bem como sobre os sinônimos dos componentes dessa estrutura, como mostra o caso A da figura 5.1. Com base nessas informações, tanto a base de conhecimentos quanto a parte dependente do vocabulário são reconfigurados. Esse método é utilizado nos sistemas TEAM [ART95c] e TQA [Damerau85].

Alguns Sistemas Gerenciadores de Bancos de Dados permitem consultas à estrutura dos bancos de dados por eles gerenciados. O tradutor pode utilizar tais consultas para uma reconfiguração automática da base de conhecimentos e do vocabulário. O ABD deve somente associar os componentes da estrutura do banco de dados aos seus respectivos sinônimos através de perguntas como no método anterior, ou que o usuário ensine o tradutor tais sinônimos ao longo do tempo, através de palavras não reconhecidas pelo tradutor, como mostra o caso B da figura 5.1, ou ainda explicitamente, como no sistema LIFER [HSS+78] (caso C da mesma figura). Talvez seja melhor que o próprio usuário alimente o sistema com sinônimos, pois o ABD pode associar sinônimos que não sejam utilizados pelos usuários. Esse método parece melhor que o anterior, pois automatiza a reconfiguração da base de conhecimentos e parte do vocabulário (exceto sinônimos).

O analisador também pode ser reconfigurado ao longo do tempo pelo usuário, como mostra o caso D da figura 5.1. Para tanto, o analisador deve oferecer um conjunto de regras gramaticais inicial que possa ser utilizado em várias aplicações e permita que o usuário ensine individualmente (pois cada usuário pode ter uma maneira pessoal de se expressar) novas regras. Tais analisadores são chamados analisadores adaptáveis [Lehman92]. Além de facilitar na reconfiguração, os analisadores adaptáveis são mais adequados a usuários freqüentes, pois pode ser desagradável, para tais usuários, ter que se expressar constantemente de um modo limitado; além disso, os analisadores adaptáveis podem ser utilizados para a definição de conceitos nebulosos, como "perto", "longe", "quente", dentre outros, como mostra o caso D da figura 5.1.

A:	file name: cabo
	fields: id, nome, diâmetro, material
	synonyms: fio
	primary key: id
	⋮
B:	Qual é a area das localidades com mais de 1000 habitantes? (usuário)
	A palavra "localidades" não é reconhecida pelo sistema! (sistema)
	Qual é o sinônimo dessa palavra? (sistema)
	cidade (usuário)
C:	Defina fio como sinônimo de cabo
D:	Defina "Qual é a área da cidade de Campinas" como "Área cidade Campinas"
	Defina "a menos de 10 km" como "perto"
	Defina "a mais de 50 km" como "longe"
	Defina "cuja temperatura é maior que 30" como "temperatura quente"

Figura 5.1: Alguns métodos para a reconfiguração do tradutor.

5.2.2 - Extensões Gerais

As principais extensões gerais são as seguintes:

- **Construção da Interface Multimodal:** a construção de uma interface multimodal não envolve apenas a junção entre manipulação direta e linguagem natural. Outros problemas surgirão nessa tarefa, como a comunicação entre esses modos de interação, como por exemplo, se um usuário selecionar uma cidade em um mapa através de manipulação direta e formular a seguinte consulta em LN "Qual é a população dessa cidade?". Logo ações feitas através de manipulação direta poderão afetar o contexto utilizado na LN; portanto, mecanismos de comunicação entre os modos de interação terão que ser implementados. As referências [García95] e [Cohen92] detalham a implementação de interfaces multimodais;

- **Testes Práticos:** embora haja um grande número de sistemas que tratam linguagem natural, a viabilidade e vantagens da LN sobre os outros modos de interação continuam não provados, devido aos poucos estudos empíricos sobre os seus resultados práticos [Jarke85]. Seria interessante, então, a execução de experimentos práticos, após a construção da interface multimodal, e sobre esta, que comparassem as vantagens e desvantagens teóricas com os resultados dos experimentos, principalmente em se tratando de SIG, onde o processamento de linguagem natural tem sido pouco aplicado.

Apêndice - Código Fonte do Tradutor

O código fonte do tradutor está escrito em Arity Prolog (aproximadamente 1500 linhas de código). O código fonte relativo a cada módulo do tradutor será mostrado a seguir.

1 - Código Fonte do Analisador

```
/*_____________________________________________________________________

                    Tradutor LN -> LEGAL
                    Modulo Parser
                    Regras Gramaticais
_____________________________________________________________________*/

parser(S) :- sentenca(S,[]), start, !.
/* sentenca traduzida com sucesso */

parser(S) :- eh_erro(ERR), (( verifica_erro_orto(S,ERR,S2), parser(S2)); ( nl, write('A palavra '),
write(ERR),
 write(' nao foi reconhecida pelo sistema !'), nl)), remove(eh_erro), start,!.
/* erro ortografico ou palavra nao reconhecida */

parser(S) :- nl, write('Sentensa incompreendida pelo sistema !'), nl, start, !.
/* erro gramatical */

sentenca --> ((pronome, ( verbo(_); [])); verbo(_)), predicativo(SELECT,FROM_S,_,_,_,_),
where(WHERE,COMP,VAL,CONECT,FROM_W,FROM_S,FROM_U),
{ WHERE \= [], verifica_where(WHERE,SIN_W), verifica_from(FROM_W,SIN_F),
uniao(FROM_S,SIN_F,FROM),
join(FROM,FROM_J,SIN_W,COMP,VAL,CONECT,WHERE_J,COMP_J,VAL_J,CONECT_J),
verify_alias(SELECT,FROM_J,WHERE_J,VAL_J,CONECT_J,S_A,F_A,W_A,V_A),
monta_consulta(S_A,F_A,W_A,COMP_J,V_A,CONECT_J),
set_context(SELECT,FROM_J,WHERE_J,COMP_J,VAL_J,CONECT_J) }.
/* (Qual <e>)/Cite a populacao e o mapa da cidade cuja area e maior que 10 */

sentenca --> ((pronome, ( verbo(_); [])));
verbo(_)),pedicativo(SELECT,FROM,WHERE,COMP,VAL,CONECT),
{ join(FROM,FROM_J,WHERE,COMP,VAL,CONECT,WHERE_J,COMP_J,VAL_J,CONECT_J),
verify_alias(SELECT,FROM_J,WHERE_J,VAL_J,CONECT_J,S_A,F_A,W_A,V_A),
monta_consulta(S_A,F_A,W_A,COMP_J,V_A,CONECT_J),
set_context(SELECT,FROM_J,WHERE_J,COMP_J,VAL_J,CONECT_J) }.
/* (Qual <e>)/Cite a populacao e o mapa da cidade de campinas e jundiai */
```

```
sentenca --> conjuncao(_), sp_unit_from(FROM_SEL),
where(WHERE,COMP,VAL,CONECT,FROM_W,FROM_SEL,F_ULT), { WHERE \= [] ,
contexto(SELECT_ANT,_,_,_,_,_), verifica_from(FROM_SEL,FROM_S),
verifica_from(FROM_W,SIN_FW), uniao(FROM_S,SIN_FW,SIN_F), verifica_where(WHERE,SIN_W),
extrai_atrs(SELECT_ANT,FROM_S,SELECT), SELECT \= [],
join(SIN_F,FROM_J,SIN_W,COMP,VAL,CONECT,WHERE_J,COMP_J,VAL_J,CONECT_J),
verify_alias(SELECT,FROM_J,WHERE_J,VAL_J,CONECT_J,S_A,F_A,W_A,V_A),
monta_consulta(S_A,F_A,W_A,COMP_J,V_A,CONECT_J),
set_context(SELECT,FROM_J,WHERE_J,COMP_J,VAL_J,CONECT_J) }.
/* Elipse: E da cidade cuja .... OBS : referencia a atributos passados */

sentenca --> conjuncao(_), ( artigo_def; [] ), { contexto(SELECT,_,_,_,_,_),
extrai_from_sel(SELECT,FROM_SEL) },
where(WHERE,COMP,VAL,CONECT,FROM_W,FROM_SEL,F_ULT),
{ verifica_from(FROM_W,SIN_FW), uniao(FROM_SEL,SIN_FW,SIN_F),
verifica_select(SELECT,SIN_S), verifica_where(WHERE,SIN_W),
join(SIN_F,FROM_J,SIN_W,COMP,VAL,CONECT,WHERE_J,COMP_J,VAL_J,CONECT_J),
verify_alias(SIN_S,FROM_J,WHERE_J,VAL_J,CONECT_J,S_A,F_A,W_A,V_A),
monta_consulta(S_A,F_A,W_A,COMP,V_A,CONECT),
set_context(SIN_S,FROM_J,WHERE_J,COMP,VAL,CONECT) }.
/* Elipse: E onde ...; e <a> que tem ... OBS: referencia a atributos passados */

sentenca -->((conjuncao(_),((pronome,(verbo(_); [])); [])); ((pronome,(verbo(_); []));[])),
sn(ATRs), pronome, subst_class(FROM_SEL),
{ verifica_from([FROM_SEL],[FROM_S]),
SELECT = [[ATRs,FROM_S]], contexto(SELECT_ANT,FROM,WHERE,COMP,VAL,CONECT),
extrai_from_sel(SELECT_ANT,FROM_SEL_ANT), [FROM_S] = FROM_SEL_ANT,
verifica_select(SELECT,SIN_S), verify_alias(SIN_S,FROM,WHERE,VAL,CONECT,S_A,F_A,W_A,V_A),
monta_consulta(S_A,F_A,W_A,COMP,V_A,CONECT),
set_context(SIN_S,FROM,WHERE,COMP,VAL,CONECT) }.
/* Referencia pronominal: (E <qual <e>> / qual <e>) a area e populacao dessas cidades
OBS: referencia`a condicoes anteriores... */

sentenca --> conjuncao(_), sp_unit_from(FROM_SEL),
sp(WHERE,COMP,VAL,CONECT,FROM_SEL), { WHERE \= [],
contexto(SELECT_ANT,_,_,_,_,_), verifica_from(FROM_SEL,SIN_F),
extrai_atrs(SELECT_ANT,SIN_F,SELECT), SELECT \= [],
verifica_where(WHERE,SIN_W),
join(SIN_F,FROM_J,SIN_W,COMP,VAL,CONECT,WHERE_J,COMP_J,VAL_J,CONECT_J),
verify_alias(SELECT,FROM_J,WHERE_J,VAL_J,CONECT,S_A,F_A,W_A,V_A),
monta_consulta(SELECT,F_A,W_A,COMP,V_A,CONECT),
set_context(SELECT,FROM_J,WHERE_J,COMP,VAL,CONECT) }.
/*Elipse: E das cidades de Jundiai e Caxias ... OBS: referencia a atributos passados*/

predicativo(SIN_S,SIN_F,_,_,_,_) --> sn_p(SELECT,FROM_SEL), { FROM_SEL \= [],
verifica_from(FROM_SEL,SIN_F), verifica_select(SELECT,SIN_S) }.
/* ... a populacao e a area da cidade ... */
```

```
predicativo(SIN_S,FROM_J,WHERE_J,COMP_J,VAL_J,CONECT_J) -->
sn_p(SELECT,FROM), sp(WHERE,COMP,VAL,CONECT,FROM), { verifica_val(VAL),
verifica_from(FROM,SIN_F), verifica_select(SELECT,SIN_S), verifica_where(WHERE,SIN_W),
join(SIN_F,FROM_J,SIN_W,COMP,VAL,CONECT,WHERE_J,COMP_J,VAL_J,CONECT_J) }.
/*... a populacao e a area da cidade de Campinas, de Santos e Jundiai...*/
predicativo([[[[nome]],FROM_S]],[FROM_S],_,_,_,_) -->
(artigo_def; []), subst_class(FROM_SEL), { verifica_from([FROM_SEL],[FROM_S]) }.
 /* ... <a> cidade ... */

predicativo(SIN_S,FROM,_,_,_,_) --> sn(ATRs),
sp_from(FROM_SEL),{ verifica_from(FROM_SEL,FROM), fill_select(ATRs,FROM,SELECT),
verifica_select(SELECT,SIN_S) }.
/* ... a area e a populacao da cidade e do estado ... */

predicativo(SELECT,FROM,WHERE,COMP,VAL,CONECT) --> sn([[OP]]), preposicao,
artigo_def, subst_class(F1), sp_unit_val([VAL1]), conjuncao(_), artigo_def, subst_class(F2),
sp_unit_val([VAL2]), { alias([F1,F2],[FAL1,FAL2]), FROM = [FAL1,FAL2],
(FAL1 = [F1,AL1], FAL2 = [F2,AL2], FR1 = AL1, FR2 = AL2; FR1 = F1, FR2 = F2),
operador(OP,OPER), SELECT = [OPER,[[[geometria]],FR1],[[[geometria]],FR2]],
WHERE = [[nome,FR1],[nome,FR2]], COMP = ['=','='], VAL = [VAL1,VAL2], CONECT = [and] }.
/*  a distancia entre a cidade de campinas e o estado do para */

predicativo([[[[nome]],FROM_S]],SIN_F,
[[inside,[geometria,FROM_S],[geometria,FROM_W]],[nome,FROM_W]],[=],VAL,[and])
--> artigo_def, subst_class(FROM_SEL), sp_unit_from([FROM_W]), sp_unit_val(VAL),
{ uniao([FROM_SEL],[FROM_W],FROM), verifica_from([FROM_SEL],[FROM_S]),
verifica_from(FROM,SIN_F) }.
 /* ... as estradas do estado do para */

predicativo(_,_,_,_,_,_) --> artigo_def, subst_estr(ESTR), preposicao,
(subst_class(OBJ); subst_estr(OBJ)), {answer(ESTR,OBJ)}.
/* Meta Consulta: Quais <sao> os atributos de cidade
  Quais <sao> as classes do banco de dados       */

predicativo(_,_,_,_,_,_) --> artigo_def, pronome, verbo(_), preposicao,
(subst_class(VAR), {ESTR = atributos}; subst_estr(VAR), {ESTR = classes}),
{answer(ESTR,VAR)}.
/* Meta Consulta: O que tem em cidade / O que tem no banco de dados */

sn_p([[SELECT,FROM]|A],[FROM|B]) --> sn(SELECT), { SELECT \= [] },
sp_unit_from([FROM]), sn_p(A,B).
 /* a area da cidade da caxias e a populacao da cidade de bauru */

sn_p([],[]) --> [].

sn([[H|A]|T]) --> artigo_def, subst_atr(H), (adjetivo_fSQL(A); ([], { A = [] })), sn(T).
/* a area <minima> */

sn([[H|A]|T]) --> (conjuncao(_); virgula(_)), (artigo_def; []), subst_atr(H),
(adjetivo_fSQL(A); ([], { A = [] })), sn(T).
/* e/, <a> area <minima> */

sn([]) --> [].
```

```
sp_from([FROM|A]) --> preposicao, subst_class(FROM), sp_from(A).
/* da cidade */


sp_from([FROM|A]) --> (conjuncao(_); virgula(_)), preposicao,
subst_class(FROM), sp_from(A).
/* e do estado */

sp_from([FROM|A]) --> (conjuncao(_); virgula(_)), subst_class(FROM), sp_from(A).
/* e estado */

sp_from([]) -->[].

sp_unit_from([UNIT]) --> preposicao, (pronome; []), subst_class(UNIT).
/* da cidade / de cada cidade */

sp_unit_val([UNIT]) --> preposicao, valor(UNIT).
/* de Campinas */

sp([[nome,FROM]|A],['='|B],[VAL|C],CONECT,[FROM]) --> preposicao, valor(VAL),
sp(A,B,C,CONECT,[FROM]).
 /* de Campinas */

sp([[nome,FROM]|A],['='|B],[VAL|C],[or|D],[FROM]) -->
( conjuncao(_); virgula(_) ), preposicao, valor(VAL),sp(A,B,C,D,[FROM]).
 /* e/, de Campinas */

sp([[nome,FROM]|A],['='|B],[VAL|C],[or|D],[FROM]) -->
( conjuncao(_); virgula(_) ), valor(VAL), sp(A,B,C,D,[FROM]).
/* e/,  Campinas */

sp([],[],[],[],_) --> [].

where([[WHERE,FROM_W]|A],[COMP|B],[VAL|C],CONECT,[FROM_W|D],FROM_S,F_ULT)
--> pronome, (artigo_def; []), subst_atr(WHERE), sp_unit_from([FROM_W]),
verbo(_), adjetivo(COMP), (pronome; preposicao), valor(VAL),
unidade(_), {append([FROM_W],F_ULT,F_ULT1)},
where(A,B,C,CONECT,D,FROM_S,F_ULT1).
/*... cuja <a> area da cidade eh maior/igual que/a 100 <m2>...  */

where([[WHERE,FROM]|A],[COMP|B],[VAL|C],CONECT,FROM_W,FROM_S,F_ULT)
--> pronome, (artigo_def; []), subst_atr(WHERE), verbo(_), adjetivo(COMP),
(pronome; preposicao), valor(VAL), unidade(_),
{((get_first(F_ULT,PELE), uniao(FROM_S,PELE,FROM_DES),
desambig(WHERE,FROM_DES,FROM));
(((FROM_S = [UNIT], FROM = UNIT); desambig(WHERE,FROM_S,FROM)))) },
where(A,B,C,CONECT,FROM_W,FROM_S,F_ULT).
 /*... cuja <a> populacao eh maior/inferior que/a 100 <habitantes> ... */
```

```
where([[WHERE,FROM_W]|A],[COMP|B],[VAL|C],[CONECT|D],[FROM_W|E],FROM_S,F_ULT)
--> (conjuncao(CONECT); virgula(CONECT)), (pronome; []), (artigo_def; []),
subst_atr(WHERE), sp_unit_from([FROM_W]), verbo(_), adjetivo(COMP),
(pronome; preposicao), valor(VAL), unidade(_),
{ append([FROM_W],F_ULT,F_ULT1) }, where(A,B,C,D,E,FROM_S,F_ULT1).
/*...e/, <cuja> <a> populacao do estado eh maior/igual que/a 100 <habitantes> ... */

where([[WHERE,FROM]|A],[COMP|B],[VAL|C],[CONECT|D],FROM_W,FROM_S,F_ULT)
--> (conjuncao(CONECT); virgula(CONECT)), (pronome; []), (artigo_def; []),
subst_atr(WHERE), verbo(_), adjetivo(COMP), (pronome; preposicao), valor(VAL),
unidade(_), {((get_first(F_ULT,PELE), uniao(FROM_S,PELE,FROM_DES),
desambig(WHERE,FROM_DES,FROM));
(((FROM_S = [UNIT], FROM = UNIT); desambig(WHERE,FROM_S,FROM)))) },
where(A,B,C,D,FROM_W,FROM_S,F_ULT).
/* ... e/, <cuja> <a> populacao eh maior/inferior que/a 100 <habitantes> ...*/

where([[WHERE,FROM_W]|A],[COMP|B],[VAL|C],CONECT,[FROM_W|D],FROM_S,F_ULT)
--> preposicao, (artigo_def; []), subst_atr(WHERE), sp_unit_from([FROM_W]),
adjetivo(COMP), (pronome; preposicao), valor(VAL), unidade(_),
{ append([FROM_W],F_ULT,F_ULT1) }, where(A,B,C,CONECT,D,FROM_S,F_ULT1).
/* ... com <a> populacao da cidade igual/menor a/que 100 <habitantes> ... */

where([[WHERE,FROM]|A],[COMP|B],[VAL|C],CONECT,FROM_W,FROM_S,F_ULT)
--> preposicao, (artigo_def; []), subst_atr(WHERE), adjetivo(COMP),
(pronome; preposicao), valor(VAL), unidade(_),
{((get_first(F_ULT,PELE), uniao(FROM_S,PELE,FROM_DES),
desambig(WHERE,FROM_DES,FROM));
(((FROM_S = [UNIT], FROM = UNIT); desambig(WHERE,FROM_S,FROM)))) },
where(A,B,C,CONECT,FROM_W,FROM_S,F_ULT).
/* ... com <a> populacao igual/maior a/que 100 <habitantes> ... */

where([[WHERE,FROM_W]|A],[COMP|B],[VAL|C],[CONECT|D],[FROM_W|E],FROM_S,F_ULT)
--> (conjuncao(CONECT); virgula(CONECT)), (preposicao; []), (artigo_def; []),
subst_atr(WHERE), sp_unit_from([FROM_W]), adjetivo(COMP),
(pronome;preposicao), valor(VAL), unidade(_), {append(FROM_W,F_ULT,F_ULT1)},
where(A,B,C,D,E,FROM_S,F_ULT1).
/*...e/, <com> <a> populacao da cidade igual/maior a/que 100 <habitantes>...*/

where([[WHERE,FROM]|A],[COMP|B],[VAL|C],[CONECT|D],FROM_W,FROM_S,F_ULT)
--> (conjuncao(CONECT); virgula(CONECT)), (preposicao; []), (artigo_def; []),
subst_atr(WHERE), adjetivo(COMP), (pronome;preposicao),valor(VAL),unidade(_),
{((get_first(F_ULT,PELE), uniao(FROM_S,PELE,FROM_DES),
desambig(WHERE,FROM_DES,FROM));
(((FROM_S = [UNIT], FROM = UNIT); desambig(WHERE,FROM_S,FROM)))) },
where(A,B,C,D,FROM_W,FROM_S,F_ULT).
/* ... e/, <com> <a> populacao igual/menor a/que 100 <habitantes> ... */

where([[WHERE,FROM_W]|A],['='|B],[VAL|C],CONECT,[FROM_W|D],FROM_S,F_ULT)
--> pronome, (artigo_def; []), subst_atr(WHERE), sp_unit_from([FROM_W]), verbo(_),
valor(VAL), unidade(_), { append([FROM_W],F_ULT,F_ULT1) },
where(A,B,C,CONECT,D,FROM_S,F_ULT1).
/* ... cuja <a> populacao da cidade eh 100 <habitantes> ... */
```

```
where([[WHERE,FROM]|A],['='|B],[VAL|C],CONECT,FROM_W,FROM_S,F_ULT)
--> pronome, (artigo_def; []), subst_atr(WHERE), verbo(_), valor(VAL),
unidade(_), {((get_first(F_ULT,PELE), uniao(FROM_S,PELE,FROM_DES),
desambig(WHERE,FROM_DES,FROM));
(((FROM_S = [UNIT], FROM = UNIT); desambig(WHERE,FROM_S,FROM)))) },
where(A,B,C,CONECT,FROM_W,FROM_S,F_ULT).
 /* ... cuja <a> populacao eh 100 <habitantes> ... */
where([[WHERE,FROM_W]|A],['='|B],[VAL|C],[CONECT|D],[FROM_W|E],FROM_S,F_ULT)
--> (conjuncao(CONECT); virgula(CONECT)), (pronome; []), (artigo_def; []),
subst_atr(WHERE), sp_unit_from([FROM_W]), verbo(_), valor(VAL),
unidade(_), { append([FROM_W],F_ULT,F_ULT1) }, where(A,B,C,D,E,FROM_S,F_ULT1).
 /* ... e/, <cuja> <a> populacao da cidade eh 100 <habitantes> ... */

where([[WHERE,FROM]|A],['='|B],[VAL|C],[CONECT|D],FROM_W,FROM_S,F_ULT)
--> (conjuncao(CONECT); virgula(CONECT)), (pronome; []), (artigo_def; []),
subst_atr(WHERE), verbo(_), valor(VAL), unidade(_),{((get_first(F_ULT,PELE),
uniao(FROM_S,PELE,FROM_DES), desambig(WHERE,FROM_DES,FROM));
(((FROM_S = [UNIT], FROM = UNIT); desambig(WHERE,FROM_S,FROM)))) },
where(A,B,C,D,FROM_W,FROM_S,F_ULT).
 /* ... e/, <cuja> <a> populacao eh 100 <habitantes> ... */

where([[WHERE,FROM_W]|A],['='|B],[VAL|C],[CONECT|D],[FROM_W|E],FROM_S,F_ULT)
--> (conjuncao(CONECT); virgula(CONECT)), (preposicao; []), (artigo_def; []),
subst_atr(WHERE), sp_unit_from([FROM_W]), valor(VAL), unidade(_),
{ append([FROM_W],F_ULT,F_ULT1) }, where(A,B,C,D,E,FROM_S,F_ULT1).
 /* ... e/, <com> <a>  area da cidade 100 <m2> ... */

where([[WHERE,FROM]|A],['='|B],[VAL|C],[CONECT|D],FROM_W,FROM_S,F_ULT)
--> (conjuncao(CONECT); virgula(CONECT)), (preposicao; []), (artigo_def; []),
subst_atr(WHERE), valor(VAL), unidade(_),
{((get_first(F_ULT,PELE), uniao(FROM_S,PELE,FROM_DES),
desambig(WHERE,FROM_DES,FROM));
(((FROM_S = [UNIT], FROM = UNIT); desambig(WHERE,FROM_S,FROM)))) },
where(A,B,C,D,FROM_W,FROM_S,F_ULT).
 /* ... e/, <com> <a>  area 100 <m2> ... */

where([[WHERE,FROM_W]|A],[COMP|B],[VAL|C],CONECT,[FROM_W|D],FROM_S,F_ULT)
--> pronome, (artigo_def; []), subst_class(FROM_W), verbo(_), subst_atr(WHERE),
adjetivo(COMP), (pronome; preposicao), valor(VAL), unidade(_),
{ append([FROM_W],F_ULT,F_ULT1) }, where(A,B,C,CONECT,D,FROM_S,F_ULT1).
/* ... cujo <o> estado tem area maior/superior que/a 100 <m2> ... */

where([[WHERE,FROM_W]|A],[COMP|B],[VAL|C],[CONECT|D],[FROM_W|E],FROM_S,F_ULT)
--> (conjuncao(CONECT); virgula(CONECT)) , (pronome; []), (artigo_def; []),
subst_class(FROM_W), verbo(_), subst_atr(WHERE), adjetivo(COMP), (pronome; preposicao),
valor(VAL), unidade(_), { append([FROM_W],F_ULT,F_ULT1) },
where(A,B,C,D,E,FROM_S,F_ULT1).
/* ... e/, <cujo> <o> estado tem area maior/superior que/a 100 <m2> ... */
```

```
where([[WHERE,FROM_W]|A],['='|B],[VAL|C],CONECT,[FROM_W|E],FROM_S,F_ULT)
--> pronome, (artigo_def; []), subst_class(FROM_W), verbo(_), subst_atr(WHERE), valor(VAL),
unidade(_), { append([FROM_W],F_ULT,F_ULT1) },
where(A,B,C,CONECT,E,FROM_S,F_ULT1).
 /*  ... <cuja> <o> estado tem area 100 <m2> ... */

where([[WHERE,FROM_W]|A],['='|B],[VAL|C],[CONECT|D],[FROM_W|E],FROM_S,F_ULT)
--> (conjuncao(CONECT); virgula(CONECT)), (pronome; []), (artigo_def; []),
subst_class(FROM_W), verbo(_), subst_atr(WHERE), valor(VAL), unidade(_),
{ append([FROM_W],F_ULT,F_ULT1) }, where(A,B,C,D,E,FROM_S,F_ULT1).
 /*  ... e/, <cuja> <o> estado tem area 100 <m2> ... */

where([[WHERE,FROM]|A],[COMP|B],[VAL|C],CONECT,FROM_W,FROM_S,F_ULT)
--> (pronome; []), verbo(_), subst_atr(WHERE), adjetivo(COMP), (pronome; preposicao),
valor(VAL), unidade(_),
{((get_first(F_ULT,PELE), uniao(FROM_S,PELE,FROM_DES),
desambig(WHERE,FROM_DES,FROM));
(((FROM_S = [UNIT], FROM = UNIT); desambig(WHERE,FROM_S,FROM)))) },
where(A,B,C,CONECT,FROM_W,FROM_S,F_ULT).
/*  ... <que> tem area maior/superior que/a 100 <m2> ... */

where([[WHERE,FROM]|A],[COMP|B],[VAL|C],[CONECT|D],FROM_W,FROM_S,F_ULT)
--> (conjuncao(CONECT); virgula(CONECT)), (pronome; []), verbo(_),subst_atr(WHERE),
adjetivo(COMP), (pronome; preposicao), valor(VAL), unidade(_), {((get_first(F_ULT,PELE),
uniao(FROM_S,PELE,FROM_DES), desambig(WHERE,FROM_DES,FROM));
(((FROM_S = [UNIT], FROM = UNIT); desambig(WHERE,FROM_S,FROM)))) },
where(A,B,C,D,FROM_W,FROM_S,F_ULT).
/*  ... e/, <que> tem area maior/superior que/a 100 <m2> ... */

where([[WHERE,FROM]|A],['='|B],[VAL|C],CONECT,FROM_W,FROM_S,F_ULT)
--> (pronome; []), verbo(_), subst_atr(WHERE), valor(VAL), unidade(_),
{((get_first(F_ULT,PELE), uniao(FROM_S,PELE,FROM_DES),
desambig(WHERE,FROM_DES,FROM));
(((FROM_S = [UNIT], FROM = UNIT); desambig(WHERE,FROM_S,FROM)))) },
where(A,B,C,CONECT,FROM_W,FROM_S,F_ULT).
 /*  ... <que> tem area 100 <m2> ... */

where([[WHERE,FROM]|A],['='|B],[VAL|C],[CONECT|D],FROM_W,FROM_S,F_ULT)
--> (conjuncao(CONECT); virgula(CONECT)), (pronome; []), verbo(_), subst_atr(WHERE), valor(VAL),
unidade(_), {((get_first(F_ULT,PELE), uniao(FROM_S,PELE,FROM_DES),
desambig(WHERE,FROM_DES,FROM));
(((FROM_S = [UNIT], FROM = UNIT); desambig(WHERE,FROM_S,FROM)))) },
where(A,B,C,D,FROM_W,FROM_S,F_ULT).
 /*  ... e/, <que> tem area 100 <m2> ... */

where([[distance,[geometria,FROM_W],[geometria,FROM_S]],[nome,FROM_W]|A],[COMP1,'='|B],
[VAL1,VAL2|C],[and|CONECT],[FROM_W|E],[FROM_S],F_ULT)
--> preposicao, adjetivo(COMP1), preposicao, valor(VAL1), unidade(_), preposicao, subst_class(FROM_W),
(preposicao; []), valor(VAL2), where(A,B,C,CONECT,E,[FROM_S],F_ULT).
/* a menos de 50 Km do rio paraopeba */
```

```
where([[distance,[geometria,FROM_W],[geometria,FROM_S]],
[nome,FROM_W]|A],[COMP1,'='|B],[VAL1,VAL2|C],[CONECT,and|D],[FROM_W|E],[FROM_S],F_ULT)
--> (conjuncao(CONECT); virgula(CONECT)), preposicao, adjetivo(COMP1),
preposicao, valor(VAL1), unidade(_), preposicao, subst_class(FROM_W),
(preposicao; []), valor(VAL2),where(A,B,C,D,E,[FROM_S],F_ULT).
/* <e/,> a menos de 50 Km do rio paraopeba */

where(WHERE,COMP,VAL,CONECT,[FROM_W|E],[FROM_S],F_ULT)
--> (pronome; []), ((adverbio_neg, {ADV=nao}); ([], {ADV=sim} )),
(verbo(NOCAO); adjetivo_topol(NOCAO)), (preposicao; []), (artigo_indef; []), subst_class(FROM_W),
{ oper_topologicos(ADV,NOCAO,FROM_S,FROM_W,OP_TOPOLS,COMP1,VAL1,CONECT1),
append(OP_TOPOLS,A,WHERE), append(COMP1,B,COMP),
(VAL1 \= [], append([VAL1],C,VAL); VAL=C), append(CONECT1,D,CONECT),
append([FROM_W],F_ULT,F_ULT1) }, where(A,B,C,D,E,[FROM_S],F_ULT1).
/* atravessadas por um rio; que passam por um rio; que atravesssam um rio;
que nao passam por um rio; que passam por cidades */

where([[OPER,[geometria,FROM_W],[geometria,FROM_S]]|A],COMP,VAL,
[CONECT|B],[FROM_W|C],[FROM_S],F_ULT)
--> (conjuncao(CON); virgula(CON)), (pronome; []), ((adverbio_neg, {ADV=nao}); ([], {ADV=sim} )),
verbo(ADV,OPER), (preposicao; []), (artigo_indef; []), subst_class(FROM_W),
{ oper_topologicos(ADV,NOCAO,FROM_S,FROM_W,OP_TOPOLS,COMP1,VAL1,CONECT1),
append(OP_TOPOLS,A,WHERE), append(COMP1,B,COMP), append(VAL1,C,VAL),
append([CON],CONECT1,CONECT2), append(CONECT2,D,CONECT),
append([FROM_W],F_ULT,F_ULT1) }, where(A,B,C,D,E,[FROM_S],F_ULT1).
{append([FROM_W],F_ULT,F_ULT1)}, where(A,COMP,VAL,B,C,[FROM_S],F_ULT1).
/*e/, cortadas por um rio; e/, que passam por um rio; e/, que atravesssam um rio; e/, que nao passam por
cidades*/

where(WHERE,COMP,VAL,CONECT,[FROM_W|E],[FROM_S],F_ULT)
--> (pronome; []), ((adverbio_neg, {ADV=nao}); ([], {ADV=sim} )),
(verbo(NOCAO); adjetivo_topol(NOCAO)), (preposicao; pronome; artigo_def), subst_class(FROM_W),
(preposicao; []), valor(VALOR),
{ oper_topologicos(ADV,NOCAO,FROM_S,FROM_W,OP_TOPOLS,COMP1,VAL1,CONECT1),
append(OP_TOPOLS,[[nome,FROM_W]],WHERE2), append(COMP1,[=],COMP2),
((VAL1 \= [], append([VAL1],[VALOR],VAL2)); (VAL2 = [VALOR])),
append(CONECT1,[and],CONECT2), append(WHERE2,A,WHERE), append(COMP2,B,COMP),
append(VAL2,C,VAL), append(CONECT2,D,CONECT), append([FROM_W],F_ULT,F_ULT1)},
where(A,B,C,D,E,[FROM_S],F_ULT1).
/* cortadas pelo rio para; contidas no estado do para; que cruzam o rio para; vizinhas ao estado do para */

where(WHERE,COMP,VAL,CONECT,[FROM_W|E],[FROM_S],F_ULT)
--> (conjuncao(CON); virgula(CON)), (pronome; []), ((adverbio_neg, {ADV=nao}); ([], {ADV=sim} )),
(verbo(NOCAO); adjetivo_topol(NOCAO)), (preposicao; pronome; artigo_def),
subst_class(FROM_W), (preposicao; []), valor(VALOR),
{ oper_topologicos(ADV,NOCAO,FROM_S,FROM_W,OP_TOPOLS,COMP1,VAL1,CONECT1),
append(OP_TOPOLS,[[nome,FROM_W]],WHERE2), append(COMP1,[=],COMP2),
((VAL1 \= [], append([VAL1],[VALOR],VAL2)); (VAL2 = [VALOR])),
append([CON],CONECT1,CONECT2), append(CONECT2,[and],CONECT3),
append(WHERE2,A,WHERE), append(COMP2,B,COMP), append(VAL2,C,VAL),
append(CONECT3,D,CONECT),append([FROM_W],F_ULT,F_ULT1) },
where(A,B,C,D,E,[FROM_S],F_ULT1).
/* e/, cortadas pelo rio paraopeba; e/, contidas no estado do para e/, vizinhas ao estado do para */
```

```
where([[WHERE,FROM]|A],[COMP|B],[VAL|C],CONECT,FROM_W,FROM_S,F_ULT)
--> preposicao, adjetivo(COMP), preposicao, valor(VAL), unidade(ATRs),
(( { get_first(F_ULT,PELE), desambig_atr(ATRs,PELE,WHERE), [FROM] = PELE,
CONECT =[and|D] }, where(A,B,C,D,FROM_W,FROM_S,F_ULT));
( { desambig_atr(ATRs,FROM_S,WHERE),
((FROM_S = [UNIT], FROM = UNIT);  desambig(WHERE,FROM_S,FROM)) },
where(A,B,C,CONECT,FROM_W,FROM_S,F_ULT))).
/*  com mais de 1000 habitantes */

where([[WHERE,FROM]|A],[COMP|B],[VAL|C],[CONECT|D],FROM_W,FROM_S,F_ULT)
--> (conjuncao(CONECT); virgula(CONECT)), preposicao, adjetivo(COMP),
preposicao, valor(VAL), unidade(ATRs),
{ ((get_first(F_ULT,PELE), desambig_atr(ATRs,PELE,WHERE), [FROM] = PELE);
(desambig_atr(ATRs,FROM_S,WHERE),
((FROM_S = [UNIT], FROM = UNIT);  desambig(WHERE,FROM_S,FROM)))) },
where(A,B,C,D,FROM_W,FROM_S,F_ULT).
/* e/,  com mais de 1000 habitantes */

where([],[],[],[],[],_,_) --> [].

subst_atr(X) --> [X], { eh_subst_atr(X), ! }.
subst_atr(_) --> [PERR], { erro(PERR,eh_subst_atr), !, fail }.

subst_class(X) --> [X], { eh_subst_class(X), ! }.
subst_class(_) --> [PERR], { erro(PERR,eh_subst_class), !, fail }.

subst_estr(X) --> [X], { eh_subst_estr(X), ! }.
subst_estr([banco,de,dados]) --> [banco,de,dados], { ! }.
subst_estr(_) --> [PERR], { erro(PERR,eh_subst_estr), !, fail }.

adjetivo(COMP) --> [X], { eh_adjetivo(X,COMP), ! }.
adjetivo(_) --> [PERR], { erro(PERR,eh_adjetivo), !, fail }.

adjetivo_fSQL(FUNCTION) --> [X], { eh_adjetivo_fSQL(X,FUNCTION), ! }.
adjetivo_fSQL(_) --> [PERR], { erro(PERR,eh_adjetivo_fSQL), !, fail }.

adjetivo_topol(TOPER) --> [X], { eh_adjetivo_topol(X,TOPER), ! }.
adjetivo_topol(_) --> [PERR], { erro(PERR,eh_adjetivo_topol), !, fail }.

pronome --> [X], { eh_pronome(X),! }.
pronome --> [PERR], { erro(PERR,eh_pronome), !, fail }.

verbo(NOCAO) --> [X], { eh_verbo(X,NOCAO), ! }.
verbo(_) --> [PERR], { erro(PERR,eh_verbo), !, fail }.

conjuncao(CONECT) --> [X], { eh_conectivo(X,CONECT), ! }.
conjuncao(_) --> [PERR], { erro(PERR,conectivo), !, fail }.

artigo_def --> [X], { eh_artigo_def(X), ! }.
artigo_def --> [PERR], { erro(PERR,eh_artigo_def), !, fail }.

artigo_indef --> [X], { eh_artigo_indef(X), ! }.
artigo_indef --> [PERR], { erro(PERR,eh_artigo_indef), !, fail }.
```

```
preposicao --> [X], { eh_preposicao(X), ! }.
preposicao --> [PERR], { erro(PERR,eh_preposicao), !, fail }.

adverbio_neg --> [nao], { ! }.

valor([H|T]) --> [H], valor(T), { (number(H); not_class(H)) }.
valor([]) --> [].

unidade(ATRs) --> [X], { eh_unidade(X,ATRs) }.
unidade(_) --> [].
unidade(ATRs) --> unidade(_), unidade(ATRs), !.

virgula(and) --> [','].
```

2 - Código Fonte do Vocabulário

```
/*_____________________________________________________________________

                    Tradutor LN -> LEGAL
                    Modulo Vocabulario
                    Conjunto de Palavras
_____________________________________________________________________*/

eh_subst_class(municipio).
eh_subst_class(municipios).
eh_subst_class(rio).
eh_subst_class(lago).
eh_subst_class(lagos).
eh_subst_class(estrada).
eh_subst_class(estradas).
eh_subst_class(rodovia).
eh_subst_class(rodovias).
eh_subst_class(via).
eh_subst_class(cidade).
eh_subst_class(cidades).
eh_subst_class(estado).
eh_subst_class(estados).
eh_subst_class(administrador).
eh_subst_class(prefeito).
eh_subst_class(governador).
eh_subst_class(presidente).
eh_subst_class(governante).
eh_subst_class(pais).
eh_subst_atr(area).
eh_subst_atr(populacao).
eh_subst_atr(nome).
eh_subst_atr(altitude).
eh_subst_atr(latitude).
eh_subst_atr(distancia).
eh_subst_atr(extensao).
```

```
eh_subst_atr(prefeito).
eh_subst_atr(governador).
eh_subst_atr(presidente).
eh_subst_atr(sexo).
eh_subst_atr(partido).
eh_subst_atr(geometria).
eh_subst_estr(atributos).
eh_subst_estr(classes).
eh_subst_estr([banco,de,dados]).
eh_pronome(qual).
eh_pronome(quais).
eh_pronome(que).
eh_pronome(cuja).
eh_pronome(cujo).
eh_pronome(onde).
eh_pronome(dessa).
eh_pronome(dessas).
eh_pronome(desses).
eh_pronome(nessa).
eh_pronome(nessas).
eh_pronome(nesse).
eh_pronome(nesses).
eh_pronome(cada).
eh_verbo(eh,_).
eh_verbo(sao,_).
eh_verbo(cite,_).
eh_verbo(mostre,_).
eh_verbo(selecione,_).
eh_verbo(recupere,_).
eh_verbo(tem,_).
eh_verbo(seja,_).
eh_verbo(corta,cruzamento).
eh_verbo(cortam,cruzamento).
eh_verbo(cortada,cruzamento).
eh_verbo(cortado,cruzamento).
eh_verbo(cortadas,cruzamento).
eh_verbo(cortados,cruzamento).
eh_verbo(cruza,cruzamento).
eh_verbo(cruzam,cruzamento).
eh_verbo(cruzada,cruzamento).
eh_verbo(cruzadas,cruzamento).
eh_verbo(cruzado,cruzamento).
eh_verbo(cruzados,cruzamento).
eh_verbo(atravessa,cruzamento).
eh_verbo(atravessam,cruzamento).
eh_verbo(atravessada,cruzamento).
eh_verbo(atravessadas,cruzamento).
eh_verbo(atravessado,cruzamento).
eh_verbo(atravessados,cruzamento).
eh_verbo(intercepta,intersecao).
eh_verbo(interceptam,intersecao).
eh_verbo(interceptadas,intersecao).
eh_verbo(interceptados,intersecao).
```

```
eh_verbo(interseciona,intersecao).
eh_verbo(intersecionam,intersecao).
eh_verbo(contida,inclusao).
eh_verbo(contidas,inclusao).
eh_verbo(contido,inclusao).
eh_verbo(contidos,inclusao).
eh_verbo(inclusa,inclusao).
eh_verbo(inclusas,inclusao).
eh_verbo(incluso,inclusao).
eh_verbo(inclusos,inclusao).
eh_artigo_def(a).
eh_artigo_def(as).
eh_artigo_def(o).
eh_artigo_def(os).
eh_artigo_indef(um).
eh_artigo_indef(uma).
eh_preposicao(a).
eh_preposicao(ao).
eh_preposicao(da).
eh_preposicao(das).
eh_preposicao(de).
eh_preposicao(do).
eh_preposicao(dos).
eh_preposicao(com).
eh_preposicao(entre).
eh_preposicao(por).
eh_preposicao(pela).
eh_preposicao(pelo).
eh_preposicao(no).
eh_preposicao(em).
eh_adjetivo(maior,>).
eh_adjetivo(mais,>).
eh_adjetivo(superior,>).
eh_adjetivo(menor,<).
eh_adjetivo(menos,<).
eh_adjetivo(inferior,<).
eh_adjetivo(igual,=).
eh_adjetivo_fSQL(maxima,max).
eh_adjetivo_fSQL(maximo,max).
eh_adjetivo_fSQL(minima,min).
eh_adjetivo_fSQL(minimo,min).
eh_adjetivo_fSQL(media,avg).
eh_adjetivo_fSQL(medio,avg).
eh_adjetivo_topol(vizinha,adjacencia).
eh_adjetivo_topol(vizinhas,adjacencia).
eh_adjetivo_topol(vizinho,adjacencia).
eh_adjetivo_topol(vizinhos,adjacencia).
eh_adjetivo_topol(adjacente,adjacencia).
eh_adjetivo_topol(adjacentes,adjacencia).
eh_conectivo(e,and).
eh_conectivo(ou,or).
eh_unidade(m,[altitude,extensao]).
eh_unidade(metros,[altitude,extensao]).
```

```
eh_unidade(m2,[area]).
eh_unidade(quadrados,[area]).
eh_unidade(km,[altitude,extensao]).
eh_unidade(kilometros,[altitude,extensao]).
eh_unidade(habitantes,[populacao]).

/* palavras sinonimas a nomes de atributos e classes    */

eh_sinonimo(municipios,municipio).
eh_sinonimo(cidades,cidade).
eh_sinonimo(estados,estado).
eh_sinonimo(estradas,estrada).
eh_sinonimo(rodovia,estrada).
eh_sinonimo(rodovias,estrada).
eh_sinonimo(via,estrada).
eh_sinonimo(rios,rio).
eh_sinonimo(lagos,lago).
eh_sinonimo(governante,administrador).
eh_sinonimo(prefeito,administrador).
eh_sinonimo(governador,administrador).
eh_sinonimo(presidente,administrador).
```

3 - Código Fonte da Base de Conhecimentos

```
/*_____________________________________________________________________

                    Tradutor LN -> LEGAL
                    Base de Conhecimentos
                    Estrutura do Banco de Dados
_____________________________________________________________________*/

eh_atributo(nome,cidade).
eh_atributo(populacao,cidade).
eh_atributo(area,cidade).
eh_atributo(altitude,cidade).
eh_atributo(prefeito,cidade).
eh_atributo(geometria,cidade).

eh_atributo(nome,estado).
eh_atributo(populacao,estado).
eh_atributo(area,estado).
eh_atributo(governador,estado).
eh_atributo(geometria,estado).

eh_atributo(nome,pais).
eh_atributo(populacao,pais).
eh_atributo(area,pais).
eh_atributo(presidente,pais).
eh_atributo(geometria,pais).
```

```
eh_atributo(nome,municipio).
eh_atributo(populacao,municipio).
eh_atributo(area,municipio).
eh_atributo(geometria,municipio).

eh_atributo(nome,rio).
eh_atributo(extensao,rio).
eh_atributo(geometria,rio).

eh_atributo(nome,lago).
eh_atributo(area,lago).
eh_atributo(geometria,lago).

eh_atributo(nome,estrada).
eh_atributo(extensao,estrada).
eh_atributo(geometria,estrada).

eh_atributo(nome,administrador).
eh_atributo(sexo,administrador).
eh_atributo(partido,administrador).

eh_classe(cidade).
eh_classe(estado).
eh_classe(pais).
eh_classe(municipio).
eh_classe(rio).
eh_classe(lago).
eh_classe(estrada).
eh_classe(administrador).

representacao(cidade,d0).
representacao(minicipio,d2).
representacao(estado,d2).
representacao(pais,d2).
representacao(estrada,d1).
representacao(rio,d1).
representacao(lago,d2).

relacionamento([cidade,administrador],[prefeito,cidade],[nome,administrador]).
relacionamento([estado,administrador],[governador,estado],[nome,administrador]).
relacionamento([pais,administrador],[presidente,pais],[nome,administrador]).
```

4 - Código Fonte do Interpretador

```
/*_____________________________________________________________________

                    Tradutor LN -> LEGAL
                    Modulo Interpretador
                    Conjunto de predicados Auxiliares
_____________________________________________________________________*/

/*_____________________________________________________________________

                    Predicados de Uso Geral
_____________________________________________________________________*/

/* uniao(L1,lL2,LR) : une os elementos de duas listas verificando sinonimos
e eliminando os repetidos */

uniao(L1,L2,LR) :- append(L1,L2,L3), elimina_repet(L3,LR), !.

/* elimina_repet(LE,LS) : elimina elementos repetidos e sinonimos de LE
e retorna LS */

elimina_repet([],[]).
elimina_repet([H|T],L) :- membro(H,T), elimina_repet(T,L), !.
elimina_repet([H|T],L) :- (eh_sinonimo(H,S); eh_sinonimo(S,H)), membro(S,T),
elimina_repet(T,L), !.
elimina_repet([H|T],[H|L]) :- elimina_repet(T,L), !.

append([],L,L).
append([H|T],L,[H|L1]) :- append(T,L,L1).

membro(X,[X|L]).
membro(X,[H|L]) :- membro(X,L).

/* get_first(LISTA,PELE) : pega o 1§ elemento de uma lista se ele existir e o
retorna em forma de lista, senao o procedimento falha */

get_first([H|T],[H]) :- nonvar(H).

/* extract_val(WHERE,LVAL,FROM,VAL) : extrai o valor VAL correspondente
ao nome da classe FROM, essa busca e feita em WHERE [nome,FROM] e o seu
valor correspondente */

extract_val([],[],_,_) :- !.
extract_val([[A1,F1]|R],[V1|R1],FROM,VAL) :- (A1 = nome, F1 = FROM, VAL = V1);
extract_val(R,R1,FROM,VAL), !.
extract_val([[OPER,A,B]|R],LVAL,FROM,VAL) :- OPER \= distance,
extract_val(R,LVAL,FROM,VAL), !.
extract_val([[OPER,A,B]|R],[V1|R1],FROM,VAL) :- OPER = distance, extract_val(R,R1,FROM,VAL), !.
```

```
/* exist(predicado,constante): verifica se existe o predicado 'predicado' com argumento 'constante' */

exist(PRED_NAME,CONST) :- PRED =.. [PRED_NAME,CONST], call(PRED).

/* remove(predicado) : remove todas as ocorrencias do predicado 'predicado'
de aridade 1 OBS: sempre retorna TRUE*/

remove(PRED_NAME) :- PRED =.. [PRED_NAME,X], retract(PRED), fail.
remove(_).

/* Answer(ARG1,ARG2): Resposta as meta-consultas */

answer(atributos,CLASS) :- verifica_from([CLASS],[SIN]), nl, write('A classe '), write(SIN),
write(' possui os seguintes atributos:'), nl, eh_atributo(ATR,SIN), write(ATR), tab(1), fail.
answer(classes,[banco,de,dados]) :- nl, write(' O banco de dados possui as seguintes classes:'), nl,
eh_classe(CLASS), write(CLASS), tab(1), fail.
answer(_,_) :- break.

/* Atualiza informacoes sobre o contexto */

set_context(SELECT,FROM,WHERE,COMP,VAL,CONECT) :-
retract(contexto(_,_,_,_,_,_)),
assert(contexto(SELECT,FROM,WHERE,COMP,VAL,CONECT)).

set_context(SELECT,FROM,WHERE,COMP,VAL,CONECT) :-
assert(contexto(SELECT,FROM,WHERE,COMP,VAL,CONECT)).

/* extrai_from_sel(SELECT,FROM_SEL): Extrai as classes associadas aos atributos contidos em SELECT*/

extrai_from_sel([],[]).
extrai_from_sel([[L,R]|A],[R|B]) :- extrai_from_sel(A,B).

/* extrai_from_where(WHERE,FROM_WHERE): Extrai as relacoes associadas
aos atributos contidos em WHERE */

extrai_from_where([],[]).
extrai_from_where([[_,F1]|L],[F1|L1]) :- extrai_from_where(L,L1).

/* extrai_atrs(SELECT_ANT,CLASSE,SELECT_POS): extrai os atributos de SELECT_ANT que sao
associados `as relacoes contidas em CLASSE */

extrai_atrs([],_,[]).
extrai_atrs([[L,R]|A],CLASSE,[[L,R]|B]) :- membro(R,CLASSE),
extrai_atrs(A,CLASSE,B).
extrai_atrs([[L,R]|A],CLASSE,B) :- not(membro(R,CLASSE)),
extrai_atrs(A,CLASSE,B).

/* fill_select(ATR,FROM,SELECT) : constroi a variavel lista SELECT, e' usadaquando se deseja selecionar
atributos que tem o mesmo nome em varias classes */

fill_select(_,[],[]).
fill_select(ATR,[FROM|L],[[ATR,FROM]|L1]) :- fill_select(ATR,L,L1).
```

```
/*_____________________________________________________________________

                    Formatacao da Entrada de Dados
_____________________________________________________________________*/

start :- nl, nl, read_line(0,STR),
((STR = $q$, break); string_sentenca(STR,S), parser(S)).

string_sentenca(STR,[H|T]) :- string_search(' ',STR,POS),
substring(STR,0,POS,SH), (int_text(H,SH); atom_string(H,SH)),
concat(SH,' ',HSPACE), list_text(LHSPACE,HSPACE),
list_text(LSTR,STR), append(LHSPACE,LSTR1,LSTR), list_text(LSTR1,STR1),
string_sentenca(STR1,T).
string_sentenca(ULT_STR,[ULT_ELE]) :- int_text(ULT_ELE,ULT_STR);
                                      atom_string(ULT_ELE,ULT_STR).

/*_____________________________________________________________________

                Conversao de Palavras Para Operadores Topologicos
_____________________________________________________________________*/

oper_topologicos(ADV,NOCAO,CLASS1,CLASS2,WHERE,COMP,VAL,CONECT) :-
verifica_from([CLASS1],[SIN1]), verifica_from([CLASS2],[SIN2]),
representacao(SIN1,R1), representacao(SIN2,R2),
get_oper_topols(ADV,NOCAO,R1,R2,SIN1,SIN2,WHERE,COMP,VAL,CONECT), !.

get_oper_topols(sim,cruzamento,X,Y,SIN1,SIN2,
[[inside,[geometria,SIN1],[geometria,SIN2]]],[],[],[]) :-
(X=d1, Y=d0); (X=d0, Y=d1), !.

get_oper_topols(sim,cruzamento,X,Y,SIN1,SIN2,
[[cross,[geometria,SIN1],[geometria,SIN2]]],[],[],[]) :-
(X=d1, Y=d1); (X=d1, Y=d2); (X=d2, Y=d1), !.

get_oper_topols(sim,cruzamento,d2,d2,SIN1,SIN2,
[[overlap,[geometria,SIN1],[geometria,SIN2]]],[],[],[]) :- !.

get_oper_topols(sim,adjacencia,d2,d2,SIN1,SIN2,
['(',[touch,[geometria,SIN1],[geometria,SIN2]],
[inside,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]) :- !.

get_oper_topols(sim,NOCAO,d0,d0,SIN1,SIN2,
[[distance,[geometria,SIN1],[geometria,SIN2]]],[<],[1],[]) :-
(NOCAO=adjacencia; NOCAO=conexao), !.

get_oper_topols(sim,adjacencia,X,Y,SIN1,SIN2,
[[touch,[geometria,SIN1],[geometria,SIN2]]],[],[],[]) :-
(X=d1, Y=d2); (X=d2, Y=d1); (X=d0, Y=d1); (X=d1, Y=d0), !.

get_oper_topols(sim,conexao,d2,d2,SIN1,SIN2,
['(',[touch,[geometria,SIN1],[geometria,SIN2]],
[overlap,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]) :- !.
```

```
get_oper_topols(sim,conexao,X,Y,SIN1,SIN2,
['(',[touch,[geometria,SIN1],[geometria,SIN2]],
[cross,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]) :-
(X=d2, Y=d1); (X=d1, Y=d2); (X=d1, Y=d1), !.

get_oper_topols(sim,conexao,X,Y,SIN1,SIN2,
['(',[touch,[geometria,SIN1],[geometria,SIN2]],
[inside,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]) :-
(X=d0, Y=d1); (X=d1, Y=d0), !.

get_oper_topols(sim,intersecao,d2,d2,SIN1,SIN2,
['(',[overlap,[geometria,SIN1],[geometria,SIN2]],
[inside,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]) :- !.

get_oper_topols(sim,intersecao,X,Y,SIN1,SIN2,
['(',[cross,[geometria,SIN1],[geometria,SIN2]],
[inside,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]) :-
(X=d2, Y=d1); (X=d1, Y=d2), !.

get_oper_topols(sim,intersecao,d1,d1,SIN1,SIN2,
['(',[cross,[geometria,SIN1],[geometria,SIN2]],
[touch,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]) :- !.

get_oper_topols(sim,intersecao,X,Y,SIN1,SIN2,
[[inside,[geometria,SIN1],[geometria,SIN2]]],[],[],[]) :-
(X=d0, Y=d1); (X=d1, Y=d0); (X=d0, Y=d2); (X=d2, Y=d0), !.

get_oper_topols(sim,conexao,X,Y,SIN1,SIN2,
['(',[cross,[geometria,SIN1],[geometria,SIN2]],
[inside,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]) :-
(X=d2, Y=d1); (X=d1, Y=d2), !.

get_oper_topols(sim,conexao,X,Y,SIN1,SIN2,
[[inside,[geometria,SIN1],[geometria,SIN2]]],[],[],[]) :-
(X=d2, Y=d0); (X=d0, Y=d2), !.
get_oper_topols(sim,conexao,d1,d1,SIN1,SIN2,
['(',[touch,[geometria,SIN1],[geometria,SIN2]],
[cross,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]), !.

get_oper_topols(sim,conexao,X,Y,SIN1,SIN2,
[[inside,[geometria,SIN1],[geometria,SIN2]]],[],[],[]) :-
(X=d0, Y=d1); (X=d1, Y=d0), !.


get_oper_topols(sim,inclusao,X,Y,SIN1,SIN2,
[[inside,[geometria,SIN1],[geometria,SIN2]]],[],[],[]) :- X=d2; Y=d2, !.

get_oper_topols(nao,cruzamento,X,Y,SIN1,SIN2,
['(',[touch,[geometria,SIN1],[geometria,SIN2]],
[disjoint,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]) :-
(X=d2, Y=d2); (X=d2, Y=d1); (X=d1, Y=d2); (X=d1, Y=d0); (X=d0, Y=d1), !.
```

```
get_oper_topols(nao,cruzamento,d1,d1,SIN1,SIN2,
[[disjoint,[geometria,SIN1],[geometria,SIN2]]],[],[],[]), !.

get_oper_topols(nao,adjacencia,d2,d2,SIN1,SIN2,
['(',[overlap,[geometria,SIN1],[geometria,SIN2]],
[disjoint,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]), !.

get_oper_topols(nao,adjacencia,X,Y,SIN1,SIN2,
['(',[cross,[geometria,SIN1],[geometria,SIN2]],
[disjoint,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]) :-
(X=d2, Y=d1); (X=d1, Y=d2), !.

get_oper_topols(nao,adjacencia,X,Y,SIN1,SIN2,
['(',[inside,[geometria,SIN1],[geometria,SIN2]],
[disjoint,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]) :-
(X=d1, Y=d0); (X=d0, Y=d1), !.

get_oper_topols(nao,NOCAO,d0,d0,SIN1,SIN2,
[[distance,[geometria,SIN1],[geometria,SIN2]]],[>],[1],[]) :-
(NOCAO=adjacencia; NOCAO=conexao), !.

get_oper_topols(nao,conexao,X,Y,SIN1,SIN2,
[[disjoint,[geometria,SIN1],[geometria,SIN2]]],[],[],[]) :-
(X=d2, Y=d2); (X=d2, Y=d1); (X=d1, Y=d2); (X=d1, Y=d1), !.

get_oper_topols(nao,intersecao,X,Y,SIN1,SIN2,
['(',[touch,[geometria,SIN1],[geometria,SIN2]],
[disjoint,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]) :-
(X=d2, Y=d2); (X=d2, Y=d1); (X=d1, Y=d2); (X=d1, Y=d0); (X=d0, Y=d1), !.

get_oper_topols(nao,intersecao,d1,d1,SIN1,SIN2,
[[disjoint,[geometria,SIN1],[geometria,SIN2]]],[],[],[]), !.

get_oper_topols(nao,inclusao,d2,d2,SIN1,SIN2,
['(',[touch,[geometria,SIN1],[geometria,SIN2]],
[disjoint,[geometria,SIN1],[geometria,SIN2]],
[overlap,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or,or]), !.

get_oper_topols(nao,inclusao,X,Y,SIN1,SIN2,
['(',[touch,[geometria,SIN1],[geometria,SIN2]],
[cross,[geometria,SIN1],[geometria,SIN2]],
[overlap,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or,or]) :-
(X=d2, Y=d1); (X=d1, Y=d2), !.

get_oper_topols(nao,inclusao,X,Y,SIN1,SIN2,
['(',[touch,[geometria,SIN1],[geometria,SIN2]],
[disjoint,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]) :-
(X=d2, Y=d0); (X=d0, Y=d2); (X=d1, Y=d0); (X=d0, Y=d1), !.

get_oper_topols(nao,adjacencia,X,Y,SIN1,SIN2,
['(',[touch,[geometria,SIN1],[geometria,SIN2]],
[disjoint,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]) :-
(X=d2, Y=d0); (X=d0, Y=d2), !.
```

```
get_oper_topols(nao,adjacencia,X,Y,SIN1,SIN2,
['(',[inside,[geometria,SIN1],[geometria,SIN2]],
[disjoint,[geometria,SIN1],[geometria,SIN2]],')'],[],[],[or]) :-
(X=d1, Y=d0); (X=d0, Y=d1), !.

get_oper_topols(ADV,NOCAO,X,Y,SIN1,SIN2,_,_,_,_) :-
((X=d0, STR1 = 'zero'); (X=d1, STR1= 'um');
(X=d2, STR1 = 'dois')), ((Y=d0, STR2 = 'zero');
(Y=d1, STR2 = 'um');   (Y=d2, STR2 = 'dois')),
nl, write(' Nao e possivel a relacao de '), (ADV=nao, write(ADV); true),
tab(1), write(NOCAO), write(' entre a classe '), write(SIN1),
write(' de dimensao '), write(STR1), write(' e a classe '), write(SIN2),
write(' de dimensao '), write(STR2), nl, break.

/*_____

                    Resolucao de Ambiguidades

_____*/

/* Desambiguicao de sentensas em relacao a classes */

desambig(ATR,LFROM,FROM) :- find_atr(ATR,LFROM,PFROM), PFROM \= [],
(( PFROM = [UNIT], FROM = UNIT) ; ( write('O atributo '), write(ATR),
write(' se refere a '), print_list_extenso(PFROM), write(' ?'), nl,
read(FROM))), !.

/* Desambiguicao em relacao a atributos */

desambig_atr(ATR,FROM,WHERE) :- get_atr(ATR,FROM,PWHERE), PWHERE \= [],
((PWHERE = [UNIT], WHERE = UNIT); (write(' Voce esta se referindo a '),
print_list_extenso(PWHERE), write(' ?'), nl, read(WHERE))), !.

/* get_atr(ATR,FROM,ATR&FROM) : Da lista de atributos ATR, verifica-se quais
deles ocorrem nas classes FROM e os coloca em ATR&FROM */

get_atr([],_,[]) :- !.
get_atr([ATR|L],FROM,[WHERE|L1]) :- find_atr(ATR,FROM,PFROM), PFROM \= [],
WHERE = ATR, get_atr(L,FROM,L1), !.
get_atr([ATR|L],FROM,L1) :- get_atr(L,FROM,L1), !.

/* find_atr(ATR,FROM,LFROM) : Retorna as classes LFROM que contem o atributo
ATR dentro do dominio FROM */

find_atr(_,[],[]) :- !.
find_atr(ATR,[F|L],[F|L1]) :- (eh_atributo(ATR,F);
(eh_sinonimo(ATR,SIN), eh_atributo(SIN,F)); (eh_sinonimo(F,SINF),
eh_atributo(ATR,SINF)); (eh_sinonimo(ATR,SIN), eh_sinonimo(F,SINF),
eh_atributo(SIN,SINF))), find_atr(ATR,L,L1),!.
find_atr(ATR,[F|L],L1) :- find_atr(ATR,L,L1),!.
```

```
/*_____________________________________________________________________

                    Predicados para a Validacao das Consultas
_____________________________________________________________________*/

verifica_select([],[]) :- !.
verifica_select([[L,F]|A],[[SIN,F1]|B]) :- eh_valido(L,F,SIN,F1),
verifica_select(A,B), !.
verifica_select([OP,[L,F]|A],[OP,[SIN,F1]|B]) :- eh_valido(L,F,SIN,F1),
verifica_select(A,B), !.

verifica_where([],[]) :- !.
verifica_where([SIMBOL|T],[SIMBOL|L]) :- (SIMBOL = '('; SIMBOL = ')'),
                                          verifica_where(T,L).
verifica_where([[W,F]|A],[[SIN,F1]|B]) :- verifica_from([F],[SINF]),
eh_valido([[W]],SINF,[[SIN]],F1), verifica_where(A,B), !.
verifica_where([[OP_BIN,[W1,F1],[W2,F2]]|A],
[[OP_BIN,[SIN1,SINF1],[SIN2,SINF2]]|B])
:- verifica_from([F1],[SINF1]), eh_valido([[W1]],SINF1,[[SIN1]],SINF1),
verifica_from([F2],[SINF2]), eh_valido([[W2]],SINF2,[[SIN2]],SINF2),
verifica_where(A,B), !.

verifica_from([],[]) :- !.
verifica_from([F|L],[F|L1]) :- eh_classe(F), verifica_from(L,L1),!.
verifica_from([F|L],[SIN|L1]) :- eh_sinonimo(F,SIN), verifica_from(L,L1),!.
verifica_from([F|_],_) :- nl, write('A classe '), write(F), tab(1),
write('nao esta contida no banco de dados !'), nl, break,!.

verifica_val([]).
verifica_val([VAL|L]) :- not(eh_classe(VAL)), verifica_val(L),!.
verifica_val([VAL|L]) :- eh_sinonimo(VAL,SIN), not(eh_classe(SIN)),
verifica(L),!.

eh_valido([],_,[],_).
eh_valido([[A|Adj]|T],F,[[A|Adj]|L],F) :- eh_atributo(A,F),
eh_valido(T,F,L,F).
eh_valido([[A|Adj]|T],F,[[SIN|Adj]|L],F) :- eh_sinonimo(A,SIN),
eh_atributo(SIN,F), eh_valido(T,F,L,F).
eh_valido([[A|Adj]|T],FROM,[[A|Adj]|L],SIN) :- eh_sinonimo(FROM,SIN),
eh_atributo(A,SIN), eh_valido(T,SIN,L,SIN).
eh_valido([[A|Adj]|T],FROM,[[SIN1|Adj]|L],SIN2) :- eh_sinonimo(FROM,SIN2),
eh_sinonimo(A,SIN1), eh_atributo(SIN1,SIN2), eh_valido(T,SIN2,L,SIN2).
eh_valido([[A|Adj]|T],FROM,_,_) :- nl, write(' O atributo '), write(A),
write(' nao esta contido na classe '), write(FROM), write(' !'), nl, break.
```

```
/*_____________________________________________________________

        Predicados para a Construcao do Relacionamento Nao Espacial
_____________________________________________________________*/

/* join : Verifica por relacionamentos entre classes FROM e retorna
as condicoes para que esse relacionamento seja satisfeito */

join([H|T],FROM,WHERE_ANT,COMP_ANT,VAL_ANT,CON_ANT,WHERE,COMP,VAL,CONECT)
:-T=[], FROM=[H|T], WHERE=WHERE_ANT, COMP=COMP_ANT ,VAL=VAL_ANT ,CONECT =
CON_ANT.

join([H|T],FROM,WHERE_ANT,COMP_ANT,VAL_ANT,CON_ANT,WHERE,COMP,VAL,CONECT)
:- comb_2_2([H|T],L), verifica_relacionamento(L,W_J,C_J,V_J,CON_J),
elimina_repet(W_J,C_J,V_J,CON_J,W,C,V,CON),extrai_from_where(W,F_J), uniao([H|T],F_J,FROM),
append(WHERE_ANT,W,WHERE), append(COMP_ANT,C,COMP),
append(VAL_ANT,V,VAL), append(CON_ANT,CON,CONECT).

/* verifica_relacionamento: verifica os relacinamentos entre as classes */

verifica_relacionamento([],[],[],[],[]).

verifica_relacionamento([[R1,R2]|L],L1,L2,L3,L4) :- relacionamento([R1,R2],A1,A2,C1,CON1,[]),
verifica_relacionamento(L,LA,C,LB,CON), append(A1,LA,L1), append(C1,C,L2),
append(A2,LB,L3), append(CON1,CON,L4).

verifica_relacionamento([[F1,F2]|L],L1,L2,L3,L4) :- verifica_relacionamento(L,L1,L2,L3,L4).

/* relacionamentos : verifica por relacionamentos */

relacionamentos([R1,R2],[A1],[A2],['='],[and],XANT) :- (relacionamento([R1,R2],A1,A2);
relacionamento([R2,R1],A2,A1)), !.

relacionamentos([R1,R2],[A1|L1],[AX|L2],['='|L3],[and|L4],XANT) :- ((relacionamento([R1,X],A1,AX),
not(membro(X,XANT))); (relacionamento([X,R1],AX,A1)), not(membro(X,XANT))),
append(XANT,[X],LX), relacionamentos([X,R2],L1,L2,L3,L4,LX),!.

/* elimina_repet: elimina elementos repetidos */

elimina_repet([],[],[],[],[],[],[],[]).

elimina_repet([A|LW],[B|LC],[C|LV],[D|LCON],[A|W],[B|COMP],[C|V],[D|CON]) :-
not(membro(A,LW)), elimina_repet(LW,LC,LV,LCON,W,COMP,V,CON).
elimina_repet([A|LW],[B|LC],[C|LV],[D|LCON],W,COMP,V,CON) :- membro(A,LW),
elimina_repet(LW,LC,LV,LCON,W,COMP,V,CON).

/* comb_2_2 : gera combinacoes dois a dois dos elementos da lista,
ex: [a,b,c] -> [[a,b],[a,c],[b,c]]          OBS: [a] -> []
   [[a,b],[a,c],[b,c]] -> [a,b,c]                    */

comb_2_2([],[]).
comb_2_2([H|T],C2_2) :- comb(H,T,L1), comb_2_2(T,L2), append(L1,L2,C2_2), !.
```

```
/* comb : gera combinacoes 2 a 2 de um elemento com elementos de uma lista
ex : a, [b,c] -> [[a,b],[a,c]] */

comb(_,[],[]).
comb(H,[H1|T],[[H,H1]|L]) :- comb(H,T,L).

/*_____________________________________________________________________

                    Predicados para Verificacao Ortografica
_____________________________________________________________________*/

/* erro: atualiza informacoes sobre erros ortograficos */

erro(PERR,CLASS) :- not_class(PERR), remove(eh_erro),
assert(eh_erro(PERR)), !,
(not(exist(class_erro,CLASS)), assert(class_erro(CLASS))); true.

/* not_class: verifica se a palavra pertence a alguma classe de palavras do vocabulario */

not_class(X) :- not(eh_subst_atr(X)), not(eh_subst_class(X)),
not(eh_pronome(X)), not(eh_adjetivo(X,_)), not(eh_adjetivo_topol(X,_)),
not(eh_preposicao(X)),
not(eh_artigo_def(X)), not(eh_artigo_indef(X)), not(eh_verbo(X,_)),
not(conectivo(X,_)), not(eh_unidade(X,_)), not(X = ',').

/* change: troca um elemento em uma lista */

change([ERR|T],ERR,SOLUTION,[SOLUTION|T]).
change([QQ|T],ERR,SOLUTION,[QQ|T1]) :- change(T,ERR,SOLUTION,T1).

/* verifica_erro_orto: recupera o erro e coloca a nova palavra na sentenca */

verifica_erro_orto(SENTENSE,ERRO,S2) :- (retract(match_word(_,_)); true),
recup_erro_orto(ERRO),
match_word(SOLUTION,_), change(SENTENSE,ERRO,SOLUTION,S2).

/* recupera_erro_orto: tenta recuperar o erro procurando a palavra mais parecida no vocabulario */
 ( disjuncao de PREDs eh porque tem vocabulos com 1, 2 ou 3 argumentos) */

recup_erro_orto(ERRO) :- class_erro(CLASS), (PRED =.. [CLASS,PRECUP];
PRED =.. [CLASS,PRECUP,_]; PRED =.. [CLASS,_,PRECUP,_]), call(PRED),
list_text(LERRO,ERRO), list_text(LPRECUP,PRECUP), string_length(PRECUP,T1),
string_length(ERRO,T2), DIF_TAM = T1 - T2,(DIF_TAM =< 2, DIF_TAM >= -2),
compara(LERRO,LPRECUP,Point), update_match(PRECUP,Point), fail.

recup_erro_orto(_).

/* update_match: compara as possiveis palavras e armazena a mais parecida com o erro */

update_match(Word,Point) :- match_word(_,P), ((Point > P,
retract(match_word(_,_)), assert(match_word(Word,Point))); true), !.
update_match(Word,Point) :- not(match_word(_,_)),assert(match_word(Word,Point)), !.
```

```
/* compara: compara o erro com as palavras do vocabulario e retorna a pontuacao da semelhanca */

compara([],_,0) :- !.
compara(_,[],0) :- !.
compara([H|T],[H|T1],Points) :-  Points = P + 1, compara(T,T1,P), !.
compara([H|T],[A|T1],Points) :- compara([H|T],T1,Points), !.
```

5 - Código Fonte do Gerador de Consultas em LEGAL

```
/*_____________________________________________________________________

        Construcao da Consulta em LEGAL Com os Elementos Extraidos Pelo Analisador
_____________________________________________________________________*/

/*monta_consulta:  monta a consulta inteira */

monta_consulta(SELECT,FROM,WHERE,COMP,VAL,CONECT) :- nl,
                                write(' select '), print_select(SELECT,FROM),
                                nl, write(' from   '), print_from(FROM),
                                nl, ((WHERE \= [], write(' where  '),
                                print_where(WHERE,COMP,VAL,CONECT,FROM));true).

/* print_list_extenso: imprime uma lista em extenso*/

print_list_extenso([H]) :- write(H).
print_list_extenso([H|T]) :- write(H), write(' ou '), print_list_extenso(T).

print_from([F]) :- F \= [H|L], write(F).
print_from([F|T]) :- F \= [H|L], write(F), write(','), tab(1), print_from(T).
print_from([[F,ALIAS]]) :- write(ALIAS), write(' in '), write(F).
print_from([[F,ALIAS]|T]) :- write(ALIAS), write(' in '), write(F),
write(','), tab(1), print_from(T).

/* print_atr: imprime um atributo (usada em print-atributo) */

print_atr([H],[]) :- write(H).
print_atr([H],FROMi) :- write(FROMi), write('.'), write(H).
print_atr([H|Adj],[]) :- write(Adj), write('('), write(H), write(')').
print_atr([H|Adj],FROMi) :- write(Adj), write('('), write(FROMi), write('.'),
write(H), write(')').

/* print_atributo: imprime uma lista de atributos */

print_atributo(nfim,[],_,_).
print_atributo(fim,[[H|Adj]],_,FROM) :- conflito(H,FROM,N),  1 is N,
print_atr([H|Adj],[]).
print_atributo(fim,[[H|Adj]],FROMi,_) :- print_atr([H|Adj],FROMi).
print_atributo(F_NF,[[H|Adj]|T],FROMi,FROM) :- conflito(H,FROM,N),  1 is N,
print_atr([H|Adj],[]), write(','), tab(1), print_atributo(F_NF,T,FROMi,FROM).
```

```
print_atributo(F_NF,[[H|Adj]|T],FROMi,FROM) :- print_atr([H|Adj],FROMi),
write(','), tab(1), print_atributo(F_NF,T,FROMi,FROM).

/* print_select: imprime a lista SELECT */

print_select([[L,FROMi]|[]],FROM) :- print_atributo(fim,L,FROMi,FROM).
print_select([[L,FROMi]|A],FROM) :- print_atributo(nfim,L,FROMi,FROM),print_select(A,FROM).


print_select([OP_BIN,[A1,FROMi],[A2,FROMj]],FROM) :- write(OP_BIN), write('('),
print_atributo(fim,A1,FROMi,FROM), write(','), print_atributo(fim,A2,FROMj,FROM), write(')').

print_where([],[],[],[],_).

/* print_where: imprime a lista WHERE */

print_where([SIMBOL|A],COMP,VAL,CONECT,FROM) :- SIMBOL='(',
write(SIMBOL), print_where(A,COMP,VAL,CONECT,FROM).
print_where([WHERE|A],[COMP|B],[VAL|C],[],FROM) :- print_atr_val(WHERE,FROM), tab(1),
write(COMP), tab(1), print_atr_val(VAL,FROM), tab(1), print_where(A,B,C,[],FROM), !.

print_where([WHERE|A],[COMP|B],[VAL|C],[CONECT|D],FROM) :-
print_atr_val(WHERE,FROM), tab(1), write(COMP), tab(1),
print_atr_val(VAL,FROM), tab(1), write(CONECT), tab(1),
print_where(A,B,C,D,FROM), !.

/* parenteses */
print_where([[OP_BIN,W1,W2],')'|A],[COMP|B],[VAL|C],[CONECT|D],FROM) :-
OP_BIN = distance, write(OP_BIN), write('('), print_atr_val(W1,FROM),
write(','), print_atr_val(W2,FROM), write(')'), tab(1), write(COMP), tab(1),
print_atr_val(VAL,FROM), write(')'), tab(1), write(CONECT), tab(1),
print_where(A,B,C,D,FROM), !.

print_where([[OP_BIN,W1,W2]|A],[COMP|B],[VAL|C],[CONECT|D],FROM) :-
OP_BIN = distance, write(OP_BIN), write('('), print_atr_val(W1,FROM),
write(','), print_atr_val(W2,FROM), write(')'), tab(1), write(COMP), tab(1),
print_atr_val(VAL,FROM), tab(1), write(CONECT), tab(1),
print_where(A,B,C,D,FROM), !.

print_where([[OP_BIN,W1,W2]|A],[COMP|B],[VAL|C],[],FROM) :-
OP_BIN = distance, write(OP_BIN), write('('), print_atr_val(W1,FROM),
write(','), print_atr_val(W2,FROM), write(')'), tab(1), write(COMP), tab(1),
print_atr_val(VAL,FROM), tab(1), print_where(A,B,C,[],FROM), !.

/* parenteses */
print_where([[OP_BIN,W1,W2],')'|A],COMP,VAL,[CONECT|D],FROM) :-
print_atr_val(W1,FROM), tab(1), write(OP_BIN), tab(1),
print_atr_val(W2,FROM), write(')'), tab(1),
write(CONECT), tab(1), print_where(A,COMP,VAL,D,FROM), !.

print_where([[OP_BIN,W1,W2]|A],COMP,VAL,[CONECT|D],FROM) :-
print_atr_val(W1,FROM), tab(1), write(OP_BIN), tab(1), print_atr_val(W2,FROM), tab(1),
write(CONECT), tab(1), print_where(A,COMP,VAL,D,FROM), !.
```

```
/* parenteses */
print_where([[OP_BIN,W1,W2],')'|A],COMP,VAL,[],FROM) :-
(OP_BIN \= distance), print_atr_val(W1,FROM), tab(1), write(OP_BIN),
tab(1), print_atr_val(W2,FROM), write(')'), tab(1),
print_where(A,COMP,VAL,[],FROM), !.

print_where([[OP_BIN,W1,W2]|A],COMP,VAL,[],FROM) :-
(OP_BIN \= distance), print_atr_val(W1,FROM), tab(1),
write(OP_BIN), tab(1), print_atr_val(W2,FROM), tab(1),
print_where(A,COMP,VAL,[],FROM), !.

/* print_atr_val: imprime um atributo ou valor relativo a lista WHERE */

print_atr_val([A,FROMi],FROM) :- conflito(A,FROM,N), 1 is N, write(A), !.
print_atr_val([A,FROMi],FROM) :- conflito(A,FROM,N), N > 1,
                                 write(FROMi), write('.'), write(A), !.
print_atr_val([VAL],_) :- number(VAL), write(VAL), !.
print_atr_val([VAL],_) :- atom(VAL), write(""), write(VAL), write(""), !.
print_atr_val(VAL,_) :- VAL = [H|T], H \= distance, H \= touch, H \= inside,
H \= disjoint, H \= cross, H \= overlap, write(""), print_val(VAL),
write(""), !.

print_val([H]) :- atom(H), write(H), !.
print_val([H|T]) :- atomic(H), write(H), tab(1), print_val(T), !.

/* conflito : conta quantas vezes um atributo (H) aparece em uma lista de classes */

conflito(H,[],0) :- !.
conflito(H,[FROMi|L],N) :- eh_atributo(H,FROMi), N = N1 + 1,
conflito(H,L,N1), !.
conflito(H,[[FROMi,ALIAS]|L],N) :- eh_atributo(H,FROMi), N = N1 + 1,
conflito(H,L,N1), !.
conflito(H,[FROMi|L],N) :- not(eh_atributo(H,FROMi)), conflito(H,L,N), !.
conflito(H,[[FROMi,ALIAS]|L],N):-not(eh_atributo(H,FROMi)),conflito(H,L,N),!.

/*_____________________________________________________________________

                        Formacao de Aliases
_____________________________________________________________________*/

/* verify_alias(SELECT,FROM,WHERE,CONECT,S_A,F_A,W_A): verifica a necessidade do uso de alias
(exist_GisOper) e constroi os alias em SELECT, FROM E WHERE */

verify_alias(SELECT,FROM,WHERE,VAL,CONECT,S_A,F_INI,W_A,V_A) :-
((exist_GisOper(WHERE,CLASS,CONECT,N), N > 0, make_alias(FROM,CLASS,F_A,N+1,1)); F_A =
FROM),make_initials(F_A,F_INI), alias_select(SELECT,S_A,F_INI),alias_where(WHERE,F_INI,W_A),
alias_val(VAL,F_INI,V_A), !.
```

```
/* exist_GisOper(LIST,CLASS,CONECT,NUM): verifica se na lista LIST existem
operadores metricos ou topologicos que agem sobre classes de mesmo nome
CLASS e o numero de vezes que isso ocorre */

exist_GisOper([],_,[],0).
exist_GisOper([[OP,[A1,W],[A2,W]]|T],W,CONECT,N) :-
((CONECT=[H|L], CON=L, ((H = and, N = N1+1);(N = N1))); (N=N1+1, CON=CONECT)),
exist_GisOper(T,W,CON,N1), !.
exist_GisOper([H|T],W,[CONECT|L],N) :- (H \= '('; H \= ')'), exist_GisOper(T,W,L,N), !.
exist_GisOper([H|T],W,CONECT,N) :- exist_GisOper(T,W,CONECT,N), !.

/* make_alias(FROM_UNIT,CLASS,FROM_ALIAS,N_VEZES,CONTADOR) : uma vez
detectada a necessidade de alias para a classe CLASS faz-se N_VEZES
alias para essa classe */

make_alias([CLASS|T],CLASS,[FAL1|L],REPET,Count) :- Count =< REPET,
substring(CLASS,0,1,CHAR), int_text(Count,StrCount), concat(CHAR,StrCount,ALIAS),
FAL1 = [CLASS,ALIAS], Count1 is Count + 1, make_alias([CLASS|T],CLASS,L,REPET,Count1), !.

make_alias([CLASS,F|T],CLASS,[F|L],R,C):-make_alias([CLASS|T],CLASS,L,R,C),!.
make_alias([F|T],CLASS,[F|L],R,C):- F \= CLASS,make_alias(T,CLASS,L,R,C), !.

make_alias(_,_,[],_,_) :- !.

/* make_initials(FROM,INITIALS): cria iniciais ( e in estado ) para as classes de FROM */

make_initials(F,F_INITIALS) :- make_init(F,F_INIT),
verify_repet(F_INIT,F_INITIALS,[]).

make_init([],[]).
make_init([F|T],[[F,I]|L]) :- substring(F,0,1,I), make_init(T,L), !.
make_init([[F,I]|T],[[F,I]|L]) :- make_init(T,L), ! .

/* verify_repet: verifica se a lista de iniciais contem elementos repetidos */

verify_repet([],[],_).

verify_repet([[F,I]|T],[[F,I1]|L],CONTROL) :- membro([_,I],T), (membro([I,CONT],CONTROL); CONT =
0), CONT1 is CONT + 1, troca([I,CONT],[I,CONT1],CONTROL,CONTROL1),
int_text(CONT1,StrCONT1), concat(I,StrCONT1,I1),verify_repet(T,L,CONTROL1),!.

verify_repet([[F,I]|T],[[F,I1]|L],CONTROL) :- membro([I,CONT],CONTROL), CONT1 is CONT + 1,
int_text(CONT1,StrCONT1), concat(I,StrCONT1,I1), verify_repet(T,L,CONTROL), !.

verify_repet([H|T],[H|L],CONTROL) :- verify_repet(T,L,CONTROL), !.

/* troca(A,B,L1,L2): troca o elemento A da lista L1 por B em L2 */

troca(A,B,[A|L],[B|L]).
troca(A,B,[H|T],[H|L]) :- troca(A,B,T,L).
troca(A,B,[],L) :-  L = [B].
```

```
/* alias_select(SEL,SEL_A) : constroi aliases em select */

alias_select([],[],_).
alias_select([[ATRs,FROM]|T],[[ATRs,ALIAS]|L],LALIAS) :- membro([FROM,ALIAS],LALIAS),
alias_select(T,L,LALIAS), !.
alias_select(SEL,SEL_A,[F|T]) :- alias_select(SEL,SEL_A,T), !.

/* alias_where(WHERE_ANTERIOR,FROM,WHERE_) */

alias_where([],_,[]) :- !.

/* [[[cross,[geo,c1],[geo,c2]],[nome,c2]] */

alias_where([[OP,[A1,W],[A2,W]],[nome,W]|T],[[W,AL1],[W,AL2]|F],
[[OP,[A1,AL1],[A2,AL2]],[nome,AL2]|L]) :- alias_where(T,[[W,AL1]|F],L), !.

/* [(,[cross,[geo,c1],[geo,c2]],[cross,[geo,c1],[geo,c2]],),[nome,c2]] --> 2 operadores*/

alias_where(['(',[OP1,[A1,W],[A2,W]],[OP2,[A1,W],[A2,W]],')',[nome,W]|T],[[W,AL1],[W,AL2]|F],
['(',[OP1,[A1,AL1],[A2,AL2]],[OP2,[A1,AL1],[A2,AL2]],')',[nome,AL2]|L])
:- alias_where(T,[[W,AL1]|F],L), !.


/* [(,[cross,[geo,c],[geo,e]],[cross,[geo,c],[geo,e]],),[nome,e]] --> 2 operadores, so iniciais*/

alias_where(['(',[OP1,[A1,W1],[A2,W2]],[OP2,[A1,W1],[A2,W2]],')',[nome,W]|T],ALIAS,
['(',[OP1,[A1,AL1],[A2,AL2]],[OP2,[A1,AL1],[A2,AL2]],')',[nome,AL2]|L])
:- membro([W1,AL1],ALIAS), membro([W2,AL2],ALIAS), alias_where(T,[[W,AL1]|F],L), !.

/* [(,[cross,[geo,c1],[geo,c2]],[cross,[geo,c1],[geo,c2]],[cross,[geo,c1],[geo,c2]],),[nome,c2]] --> 3
operadores*/

alias_where(['(',[OP1,[A1,W],[A2,W]],[OP2,[A1,W],[A2,W]],[OP3,[A1,W],[A2,W]],')',[nome,W]|T],[[W,A
L1],[W,AL2]|F],['(',[OP1,[A1,AL1],[A2,AL2]],[OP2,[A1,AL1],[A2,AL2]],[OP3,[A1,AL1],[A2,AL2]],')',
[nome,AL2]|L]) :- alias_where(T,[[W,AL1]|F],L), !.

/* [(,[cross,[geo,c],[geo,e]],[cross,[geo,c],[geo,e]],[cross,[geo,c],[geo,e]],),[nome,e]] --> 3 operadores, so
iniciais*/

alias_where(['(',[OP1,[A1,W1],[A2,W2]],[OP2,[A1,W1],[A2,W2]],[OP3,[A1,W1],[A2,W2]],')',[nome,W]|T]
,ALIAS,['(',[OP1,[A1,AL1],[A2,AL2]],[OP2,[A1,AL1],[A2,AL2]],[OP3,[A1,AL1],[A2,AL2]],')',
[nome,AL2]|L]) :- membro([W1,AL1],ALIAS), membro([W2,AL2],ALIAS),
alias_where(T,[[W,AL1]|F],L), !.

/* [[cross,[geo,c1],[geo,c1]] */

alias_where([[OP,[A1,W],[A2,W2]]|T],[[W,AL1],[W,AL2]|F],[[OP,[A1,AL1],[A2,AL2]]|L]) :- W = W2,
alias_where(T,[[W,AL1]|F],L), !.

/* [[cross,[geo,c1],[geo,estrada]], ALIAS em cidade */

alias_where([[OP,[A1,W],[A2,W2]]|T],[[W,AL1]|F],[[OP,[A1,AL1],[A2,INITIALS]]|L]) :- W \= W2,
membro([W2,INITIALS],F), alias_where(T,[[W,AL1]|F],L), !.
```

```
/* [[cross,[geo,cidade],[geo,e1]], ALIAS em estrada */

alias_where([[OP,[A1,W1],[A2,W]]|T],[[W,AL1]|F],[[OP,[A1,INITIALS],[A2,AL1]]|L]) :- W \= W1,
membro([W1,INITIALS],F),alias_where(T,[[W,AL1]|F],L), !.

/* [[area,c1]] ALIAS em cidade */

alias_where([[ATR,W]|T],[[W,AL1]|F],[[ATR,AL1]|L]) :- alias_where(T,[[W,AL1]|F],L), !.

/* [[area,estrada]] ALIAS generico */

alias_where([[ATR,FROM]|T],LALIAS,[[ATR,ALIAS]|L]) :-
membro([FROM,ALIAS],LALIAS), alias_where(T,LALIAS,L), !.

/* alias_val(VAL,FROM,VAL_ALIAS) */

alias_val([],_,[]).
alias_val([[VAL,FROM]|T],LALIAS,[[VAL,ALIAS]|L]) :-
membro([FROM,ALIAS],LALIAS), alias_val(T,LALIAS,L), !.
alias_val([H|T],LALIAS,[H|L]) :- alias_val(T,LALIAS,L), !.
```

Bibliografia

[AD89] H. Abramson, V. Dahl. Logic Grammars. Spring Verlag, 1989.

[Allen87] J. Allen. Natural Language Understanding. The Benjamim/Cummings Publishing Inc., 1987.

[And92] I.Androutsopoulos. Interfacing a Natural Language Front-End to a Relational Database. Msc Dissertation, Departament of Artificial Intelligence, University of Edinburg, 1992.

[And96] I. Androutsopoulos. A Principled Framework for Constructing Natural Language Interfaces to Temporal Databases. PhD Thesis, Departament of Artificial Intelligence, University of Edinburg, 1996.

[Arity87] Using the Arity/PROLOG Interpreter and Compiler. Arity Corporation, 1987.

[ART93] I. Androutsopoulos, G. D. Ritchie, P. Thanisch. MASQUE/SQL - An Efficient and Portable Natural Language Query for Relational Databases. In Proceedings of the 6th International Conference on Industrial and Engeneering Applications of Artificial Intelligence and Expert Systems. Gordon and Breach Publishers Inc., 1993.

[ART95a] I. Androutsopoulos, G. D. Ritchie, P. Thanisch. A Framework for Natural Language Interfaces to Temporal Databases. Research Paper nº 734, Departament of Artificial Intelligence, University of Edinburg, 1995.

[ART95b] I. Androutsopoulos, G. D. Ritchie, P. Thanisch. Experience Using TSQL2 in a Natural Language Interface. In Recent Advances in Temporal Databases (Proceedings of the International Workshop on Temporal Databases, Zurich, september, 1995).

[ART95c] I. Androutsopoulos, G. D. Ritchie, P. Thanisch. Natural Language Interfaces to Databases - An Introduction. Journal of Natural Language Engineering, vol. 1, nº 1, pages 29-81, Cambridge University Press, 1995.

[BF81] A. Barr, E. A. Feigenbaum. The Handbook of Artificial Intelligence. Addison-Wesley Publishing Company, 1981.

[BM92] P. Boursier, M. Mainguenaud. Spatial Query Languages: Extended SQL vs. Visual Languages vs. Hypermaps. In Proc 5th International Symposium on Spatial Data Handling, pages 249-259, vol. 1, 1992.

[Cam95] G. Câmara. Modelos, Linguagens e Arquiteturas para Bancos de Dados Geográficos. Tese de Doutorado, INPE, 1995.

[Carb85] J. G. Carbonell et al. Natural Language Parsing Systems. Spring Verlag, 1985.

[CCH+96] G. Câmara, M. A. Casanova, A. S. Hermerly, G. C. Magalhães e C. B. Medeiros. Anatomia de Sistemas de Informação Geográfica. 10ª Escola de Computação, Instituto de Computação, UNICAMP, Campinas, 1996

[Cegalla91] D. P. Cegalla. Novíssima Gramática da Língua Portuguesa. Companhia Editora Nacional, 1991.

[Ciferri95] R. R. Ciferri. Um Benchmark Voltado à Análise de Desempenho de Sistemas de Informações Geográficas. Dissertação de Mestrado, Departamento de Ciência da Computação, UNICAMP, 1995.

[Cília96] M. A. Cília. Bancos de Dados Ativos como Suporte a Restrições Topológicas em Sistemas de Informações Geográficas. Dissertação de Mestrado, Departamento de Ciência da Computação, UNICAMP, 1996.

[CM84] W. F. Cloksin, C. S. Mellish. Programing in PROLOG. Springer-Verlag, 1984.

[Coelho79] H. M. F. Coelho. A Program Conversing in Portuguese Providing a Library Service. PhD Thesis, University of Edinburg, 1979.

[Cohen92] P. R. Cohen. The Role of Natural Language in a Multimodal Interface. Proceedings of the ACM Symposium on User Interface Software and Technology, november, 1992.

[Damerau85] F. J. Damerau. Problems and Some Solutions in Customuzation of Narural Language Database Front Ends. ACM Transactions on Office Information Systems, vol.3, nº 2, pages 165-184, April 1985.

[Egen92] M. J. Egenhofer. Why not SQL! Int. J. Geographical Information Systems, vol. 6, nº 2, pages 71-85, 1992.

[Egen94] M. J. Egenhofer. Spatial SQL: A Query and Presentation Language. IEEE Transactions on Knowledge and Data Engineering, vol. 6, nº 1, february, 1994.

[EN94] R. Elmarsi, S. B. Navathe. Fundamentals of Database Systems. The Benjamin/Cummings Publishing Company Inc., 1994.

[Frank92] A. U. Frank. Spatial Concepts, Geometric Data Models, and Geometric Data Structures. Computers & Geosciences, vol. 18, nº 4, pages 409-417, 1992.

[García95] L. S. García. LINX: Um Ambiente Integrado de Interface para Sistemas de Informação Baseados em Conhecimento. Tese de Doutorado, Departamento de Informática, PUC-Rio, 1995.

[Good92] M. F. Goodchild. Geographical Data Modeling. Computers & Geosciences, vol. 18, nº 4, pages 401-408, 1992.

[GS86] B. J. Grosz, C. L. Sidner. Attention, Intentions, and the Structure of Discourse. Computational Linguistics, vol. 12, nº 3, july-september, 1986.

[HC90] D. Harman, G. Candela. Bringing Natural Language Information Retrieval out of the Closet. SIGCHI Bulletim, vol 22, nº 1, july 1990.

[HSS+78] G. G. Hendrix, E. D. Sacerdoti, D. Sagalowiscz and J. Slocum. Developing a Natural Language Interface to Complex Data. ACM Transactions on Database Systems, vol 3, nº 2, pages 105-147, june 1978.

[HW88] T. Hengl, K. Weiskamp. Artificial Intelligence with TURBO PROLOG. John Wiley & Sons, Inc., 1988.

[Jarke85] M. Jarke et al. A Field Evaluation of Natural Language for Data Retrieval. IEEE Transactions on Software Engineering, vol. 11, nº 1, January 1985.

[Kaiser93] M. Kaiser. Towards a Cognitively Based Approach os description of Spatial Deixis. Technical Report, International Computer Science Institute, 1993.

[Le93] T. V. Le. Techniques of PROLOG Programing With Implementation of Logical Negation and Quantified Goals. John Wiley & Sons, Inc, 1993.

[Lehman92] J. F. Lehman. Adaptative Parsing: Self-Extending Natural Language Interfaces. Kluwer Academic Publishers, 1992.

[LS93] G. Luger, W. A. Stubblefield. Artificial Intelligence Structures and Strategies for Complex Problem Solving. The Benjamin/Cummings Publishing Company, Inc, 1993.

[LX92] X. Li, D. Xing. General Natural Language for Operating Systems. SIGART Bulletin, vol. 3, nº 4, october, 1992.

[Mag92] D. J. Maguire. The Raster GIS Design Model - A Profile of ERDAS. Computers & Geosciences, vol. 18, nº 4, pages 463-470, 1992.

[MG89] C. Mellish, G. Gazdar. Natural Language Processing in PROLOG. The Riverside Printing co., 1989.

[PC89] P. Parent and R. Church. Evolution of Geographic Information Systems as Decision Making Tools. Fundamentals of Geographic Informations Systems: A Compendium, 1989.

[PS87] F. C. N. Pereira, S. M. Shieber. PROLOG and Natural Language Analysis. Center for the Study of Language and Information, Leland Stanford Junior University, 1987.

[PW80] F. C. N. Pereira, D. H. D. Warren. Definite Clause Grammars for Language Analysis - a Survey of the Formalism and a Comparsion with Augmented Transition Networks. Artificial Intelligence, vol. 13, pages 231-278. North-Holland Publishing Company, 1980.

[Regier91] T. Regier. Learning Spatial Concepts Using a Partially-Structured Connectionist Architeture. Technical Report, International Computer Science Institute, 1993.

[RM92] J. F. Raper and D. J. Maguire. Design Models and Funcionality in GIS. Computers & Geosciences, vol. 18, nº 4, pages 387-394, 1992.

[Schildt89] H. Schildt. Inteligência Artificial Utilizando Linguagem C. Mc Graw Hill, 1989.

[Smith89] T. Smith et al. Requirements and Principles for the Implementation and Construction of Large-Scale Geographic Informations Systems. Fundamentals of Geographic Informations Systems : A Compendium, 1989.

[Tedes90] G. O. Tedesco. Um Estudo Sobre Interfaces em Linguagem Natural, com Vistas à Interação entre Usuários e Bases de Conhecimento. Dissertação de Mestrado, Faculdade de Engenharia Elétrica, UNICAMP, 1990.

[Vasco96] R. C. S. Vasconcelos. Análise Comparativa do uso dos Modelos Relacional e Orientado a Objetos em Sistemas de Informações Geográficas. Dissertação de Mestrado, Departamento de Ciência da Computação, UNICAMP, 1996.

[Wall84] M. Wallace. Communicating with Databases in Natural Language. Ellis Horwood Limited, 1984.

[Wang94] F. Wang. Towards a Natural Language User Interface: an Approach of Fuzzy Query. International Journal of Geographic Information Systems, vol. 8, nº 2, pages 143-162, 1994.

[Woods70] W. A. Woods. Transition Networks Grammars for Natural Language Analysis. Communications of the ACM, vol. 13, nº 10, pages 591-606, 1970.