NATURAL LANGUAGE ACCESS TO INTERNET SEARCH ENGINES

A Thesis
Submitted to the Faculty of Graduate Studies and Research
In Partial Fulfillment of the Requirements
for the Degree of
Master of Science
in
Computer Science
University of Regina

By
Gayathri Mahalingam
Regina, Saskatchewan
September 1997

Abstract

Natural language access to selected Internet search engines is presented. Searching for relevant documents using the existing search engines poses certain problems that include finding the most appropriate search term and scanning through a large number of potentially relevant documents. We have provided a natural language access which enables the user to present the query in English without any need to transform it to suit the individual search engines. The user's query is semantically analyzed using a HPSG parser to generate appropriate search terms. Through the semantic analysis, we eliminate the contribution of non-keywords to the search and thus help to reduce the number of sites returned.

We have evaluated the performance of our system with regard to the quality of search results. The results demonstrate the ease of expression by which we frame the query and consistency of responses. The total number of sites returned is also less compared to the results returned from the existing search engines.

We have thus presented an application of natural language processing techniques to improve the performance of the existing search engines.

Acknowledgements

I would like to sincerely thank my supervisor, Dr. Nick Cercone, for his valuable guidance and continued financial support.

I would also like to thank my external committee member, Dr. Brent Galloway, Department of Linguistics, SIFC, for his valuable comments and corrections.

I wish to thank my internal committee members, Dr Brien Maguire and Dr.Christine Chan for their attention to my research work and many valuable comments.

I also wish to thank Faculty of Graduate Studies and Research and the Department of Computer Science, University of Regina, for their academic and financial support.

My special thanks to Shinta I. Mayasari and Carlos Rivera for sharing their ideas and helping me in doing this research.

I am grateful to all my friends and family members for their help and support in making this research work possible.

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

The amount of information available on the World Wide Web (WWW) has grown enormously, thus making retrieval of relevant information very difficult. Several tools have been developed for browsing and searching through these collections of highly unstructured and heterogeneous data. These tools organize web pages into listings and allow users to search these listings to find the information needed. Each of these tools has its own listing or catalogue for searching. Based on how the new Uniform Resource Locators [1] (URLs) are added to the catalogue, they can either be classified as a directory [35] or as a spider or robot [2,17,22]. However, we refer to both types as search engines unless otherwise specifically mentioned.

The methods used by these search engines for searching include keyword searching [2,22,35], concept searching [11], and phrasal and natural language (NL) searching [17,25,32].

1.1 Keyword Searching

A "keyword" is simply a descriptor or a word that describes the content of the site to be searched. Sites matching the keywords are categorized by different search engines according to criteria specific to the search engine. Normally all search engines strive to order the sites selected according to some relevance measure based on

[1] Uniform Resource Locators specify the location of the file. URLs include the type of the resource, the address of the server and the location of the file.

keyword matches. Generally the top ranked sites in the search results are the ones

1. which match all the keywords, or
2. which have the keywords in the title [2] and on the page, or
3. in which keywords are repeated several times.

In many cases, keyword searching leads to unsatisfactory results. Usually, the search results contain large numbers of documents, many of which are irrelevant to the subject of the search. The use of boolean operations does not promote fetching of the desired results because either they return very few documents or sometimes no documents. In general, keyword searches face the following problems:

1. Since a keyword can have many synonyms, all synonyms need to be used in the search in order to obtain all relevant sites.
2. Variations in spelling might result in missing some of the relevant sites. A search for "color" might miss a site with the word "colour" and a search for "gray" will not find "grey", and so on.
3. Ambiguity in selection and ordering of keywords might result in different search results when a mere word match is performed. For example, if a search has to be made for sites performing online reservation of flight tickets, the keywords can be any of the following:
   (a) reservation, flight, ticket;
   (b) online reservation, flight, ticket;
   (c) online reservation, air, ticket;
   (d) online reservation, flight OR air, ticket;
   (e) reservation OR booking, flight OR air, ticket.

[2]The "title" refers to the HyperText Markup Language (HTML) tags <title> and </title> primarily used for document identification.

Each of the sets of keywords in combination with options exclusively supported by each search engine like "+" and "-" might be tried by the user, as each of them yields different results both in terms of number of sites returned and in their ordering in the results returned.

1.2 Concept Searching

Search engines which employ a concept searching method look not only for pages containing the exact words entered in the query but also for concepts closely linked to the words in the query. This feature of concept searching broadens the search. For example, a search for "elderly people" might bring up a site that only has "retired people" or "senior citizens" [11]. However, the selection of keywords plays an important role in the search results obtained, as in the case of keyword searching. For example selecting appropriate keyword or keywords may be a difficult task for searches like "airlines flying between Canada and Japan" and "top ranking Computer Science department in Canada".

1.3 Phrasal and Natural Language Searching

Selecting appropriate keywords for a distributed information resource like the World Wide Web may be a difficult task, even for an expert user. Hence many search engines are now providing phrasal and natural language search options. Some queries are better expressed as phrases or in natural language sentences rather than mere keywords. The possibility of finding the most relevant site as a top ranking one is much higher in these search engines when compared to keyword matching and concept searching, as expressing the query in natural language is much easier and more natural than using keywords. However, most of existing natural language search engines allow the users to present the query as phrases only or as questions only. But not all queries can be expressed as phrases or questions. For example, searches for 'air travel in U.S.A.', 'boating in Canada', 'auto rentals in Vancouver', 'visiting Japan', etc. are better expressed as phrases than as questions. Similarly the

following searches are more easy to express as sentences than as questions:
I would like to travel Europe by rail.
I want to stay in Vancouver for two days.
More over, the popularity of these search engines depends on other features like response time, catalogue updates, and relevancy ranking.

1.4 Objectives of this Research

Though there are many search engines, no one of them has been objectively judged to be better than the others. Some search engines are more suitable for searches made on general topics or categories, some may have more features for searching than the others, some may list matches differently, some may return search results more quickly and so on. Hence, the correct choice of the search engine will depend on the specific search requirements of the user.

We provide natural language access to some of the major search engines and directories thus enabling the user to access several catalogues for searching. The user can choose the search engine that may be best suited for a particular search and can also present the query in natural language in the form of word, words, phrases, sentences or questions. The natural language query will be parsed and analyzed resulting in the selection of relevant keywords. Since the Head Driven Phrase Structure Grammar (HPSG) parser contains both syntactic and semantic knowledge [as discussed in Chapter 3], it provides better selection of keywords in the sense that it reduces the listing of irrelevant sites to a great extent. Synonyms of words will be searched automatically wherever applicable. The searching of keywords with their synonyms reduces the possibility of relevant sites being missed. For example the user query, "I want to stay in Vancouver for two days", will be interpreted by the system as a search for hotels or motels or lodges in Vancouver.

This thesis essentially combines some special features of individual search engines and also provides a user-friendly natural language access to their information domain. We focus the domain of discourse on the "travel domain" with approximately 150 words currently built in the lexicon. This concept domain can be certainly extended

to other domains also. In order to facilitate further extensions, the lexicon is designed in such a way that it is modular with respect to domain-dependent and domain-independent portions.

1.5 Organization of the Thesis

This thesis is organized into six chapters. Some features of selected search engines along with a comparison chart of various search engine features are discussed in Chapter 2. A sample comparison is made on the search results of selected search engines and how the rich semantics of natural language could be used to improve the performance of search engines is also discussed in Chapter 2. The overall architecture of the system along with key features of various modules of the system are outlined in Chapter 3. The reasons for choice of HPSG parser for our system is also discussed in Chapter 3. The implementation of various modules of the system is discussed in Chapter 4.

Some experimental results with sample sessions are presented in Chapter 5. A brief conclusion is made in Chapter 6, outlining the summary of contributions. Some possible directions for further research in this area are also outlined in Chapter 6. Additional sample sessions and experimental results are presented in Appendix A. The program code of various modules of the system are listed in Appendix B.

Chapter 2

Literature Review

Despite the many search engines available on the Internet, searching for a relevant site remains a difficult task. One of the main reasons for this difficulty is not analysing the query semantically. In contrast, most of the search engines perform mere keyword matching. In this thesis, we analyse the queries using a natural language parser. In this Chapter we outline features of some of the existing search engines and consider how the semantics of natural language can be used to improve searching.

The basic differences of robots and directories are outlined in Section 2.1. In Sections 2.2 to 2.6, the features of various search engines are discussed. A comparison chart of various search engine features and search results are provided in Section 2.7. The contribution of natural language semantics towards improvement of search results are discussed in Section 2.8.

2.1 Robots versus Directories

A "robot" uses a program to search a catalogue and organize information on the Internet [10]. Robots go by various names and descriptions such as spiders, crawlers, worms and wanderers. They normally visit web sites and other Internet resources automatically and accumulate key information from them which is acted upon when employed for a search. Because of automatic site updating, their search catalogue is very large compared to the directories (ranging up to millions of web sites). There are many "robots" available due to the diverse ways of seeking the sites and the

information that is gathered.

Unlike robots, directories are created by humans. Sites must be submitted to these directories by the owners, which are assessed and assigned to an appropriate category or categories. Due to the human role, directories often provide better results and smaller catalogue size. Each directory has its own method of categorising information.

For searches in a subject of broad and general interest, directories are best suited. For rare or specific searches robots provide better results. Apart from robots and directories, there are hybrid search engines and meta search engines. Hybrid search engines are robots but they have an associated directory with them [17]. Meta search engines [Meta crawler, SavvySearch], get catalogue information from several search engines at the time of the search, and then return the combined results.

2.2 Yahoo

Yahoo is among the premier subject directories utilized on the Web. It is easy to navigate and comprehensive in its listings and additional features. It employs keyword searching. It automatically searches for both categories and web sites and passes searches to AltaVista [2]. Yahoo does not search the full text of each page unless there are no matching categories or sites are found. Yahoo accepts all sites submitted to it and does not rate sites, although it indicates ones which it recommends the most. Often, duplicate sites are listed.

Yahoo's search options include the use of logical connectives such as AND, OR, NOT, and quotes for exact phrase match. Listings added after a particular date can be chosen and a search can be made within titles or URLs. Yahoo finds all keyword matches and then sorts the results according to relevancy within each specific area. The results are ranked in the following manner [34].

- Multiple keyword matches: Documents matching more keywords will have a higher rank than those matching fewer.
- Document section weighting: Documents matching words found in the title are ranked higher than those found in its body or URL.

- Generality of category: Categories matching higher up in the hierarchy (i.e., more general categories) are ranked higher than those deeper in the hierarchy (i.e., more narrowly focussed categories).

2.3 AltaVista

AltaVista uses a "crawler" with scalable indexing software with approximately 31 million pages in the catalogue. As per AltaVista's home page, their spider crawls the web at 3 million pages a day. AltaVista is mostly a full text index. Rather than indexing words, it indexes word patterns. That is, AltaVista treats pages on the web as a sequence of words. A 'word' can be any string of letters and digits. Thus, HAL5000, 602e21 are treated as words. AltaVista does not index punctuation or white space.

AltaVista offers simple and complex query searches with boolean operators. Searches can be made for particular phrases, and in particular fields. In AltaVista's advanced form, the most important keyword or keywords can be specified in the relevance ranking field, so that results are listed in the order that is most relevant.

In AltaVista's simple search, documents are assigned a higher score if:

1. keywords are in the first few words of the document, such as the title.
2. keywords are found close to one another in the document.
3. keywords are used more than once in the document.

Being a "crawler", AltaVista normally returns a large number of results and hence is suited for doing a specific search for obscure sites, documents or information.

2.4 Excite

Excite is a unique search engine using "concept searching" with approximately 55 million pages indexed. Excite is a full text index. It uses Intelligent Concept Extraction (ICE) to find relationships that exist between words and ideas so that

the results of the search will contain words related to the concept being searched. Because of this strategy, the search is broadened, and Excite normally returns a larger number of matching sites. Excite's advanced search allows boolean searching. Excite also provides a "more like this" link and a "sort by site" function. The "more like this" link is provided for each document in the search results returned, and the specified document is taken as an example in a new search for finding similar documents. Excite's list of search results may present several pages from the same site. The "sort by site" function will compress the list to show the names of the sites and relevant documents within them.

The relevance rating for listing the search results is generated by comparing the information in the site against that in the query. The closer the rating is to 100%, the more relevant Excite thinks the document is to the query. Excite seems to favour sites with keywords in the title, and with keywords repeated frequently in relation to the rest of the document.

2.5 Infoseek

Infoseek is a full text indexed search engine with approximately 50 million pages and with two distinct search services "Ultrasmart" and "Ultraseek". Ultrasmart is the default search option. Ultrasmart lists both search engine results and matches in the Infoseek's catalogue of reviewed sites. Infoseek [18] suggests Ultrasmart is good for browsing for a general concept and then searching within the previous results, as any new query will search within the results of the previous search unless specified otherwise. Ultrasmart also gives related areas in the directory relevant to the query. Ultraseek is designed for accurate searching which also includes a number of special searches for searching images, FAQs, news, etc., with boolean operators.

Infoseek supports phrasal and natural language queries like "What are the lyrics to Penny Lane?" [19]. Infoseek claims that its phrasal query handling is different from that of other search engines as Infoseek does not drop common words (like "web") and one letter words. For example the search for "vitamin A" in search engines other than Infoseek will search for "vitamin" only.

Infoseek sorts the results based on how well each document satisfies the query and it also eliminates duplicate pages in the listing. The following factors influence the relevancy scoring:

- query terms (words or phrases) are found in the title or near the start of the document.
- the document contains more of the query terms.
- the document contains query terms that are relatively uncommon in the database.

2.6 Open Text

The "Open Text" search engine also does indexing on full-text. Though it claims to have indexed as many pages as "Lycos" (around 20 - 25 million pages), the Web Master's survey [33] rates it as a small database with 5 million pages. Open Text supports the following searches:

1. search for a single word or group of words;
2. search for a phrase of any length;
3. search for combination of words and phrases;
4. search with boolean operators;
5. search for URLs only and
6. search for titles and headings[1] only.

The results are listed based on the number of times the term searched for occurs on the page as well as where it appears (URL, title, body, etc.,)

[1]Open-Text searches only through the first level of heading or the heading encoded by <H1> tags of HTML in the Web page.

2.7 Informal Comparison

As seen in the Sections 2.2 - 2.6, each search engine differs from others in the way they index the pages, rank the search results, provide search options, etc. Table 2.1 provides a comparison chart of these features of some of the major search engines [33]. The terms used in the table are explained below:

Content: "Full-Text" engines index every word on a web page, although some stop words (common words like "web") may not be included. "Abstract" search engines create a condensed copy of a page. This copy will not include all the words appearing on the original page.

Size: The size shows the number of pages indexed. Actual numbers need not be taken into account because of the following reasons:

- Some search engines count duplicate pages.
- Some search engines count the http addresses linked to the page indexed, even though those links are not indexed.
- Crawlers add pages daily and hence the number keeps changing.

However, this is just to give a comparative idea regarding how big the catalogue is.

Frames Support: The <frame> and <frameset> HTML tags enable segmenting the browser window into frames to display a series of HTML files within a splitscreen. Search engines which do not support frames normally ignore all the information inside the <frame> and <frameset> tags.

Link Popularity: Search engines can determine the popularity of a page by analyzing how many links are there to it from other pages. Some engines use this as a mechanism for determining which pages they will include in the index.

Spam Penalty: All major search engines penalise sites that attempt to "spam" the engines in order to improve their ranking. One common technique used to improve ranking is repeating a word many times in a row. Search engines which penalise "spamming" either exclude these pages from their listings altogether or they downgrade the page's ranking.

Search Engine	Alta Vista	Excite	HotBot	Infoseek	Lycos	Open Text	Web Crawler
Content	Full-text	Full-text	Full-text	Full-text	Abstract	Full-text	Full-text
Size (pages in mills)	Big (30)	Big (55)	Big (54)	Big (20-50)	Medium (20-25)	Small (5)	Small (2)
Frames Support	No	Yes	No	Yes	Yes	No	No
Link Popularity	No	No	Yes	No	Yes	No	Yes
Stop Words	Yes	Yes	Yes	No	Yes	No	No
Spam Penalty	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Meta Tag Support	Yes	No	Yes	Yes	Yes	No	Index tag only

Table 2.1: Comparison Chart on Search Engine Features

Meta Tag Support: Meta Tags are meant for providing descriptions of a page within the <head> container of the web page and are not visible to the average browser. For example, some search engines search for the keywords only within the first 200 characters of the web page. Either 'description' of the web page or 'keywords' of the web page or both can be specified in the meta tags as shown below.

<meta name = "description" content = "description of the web page">

<meta name = "keywords" content = "keywords of the web page">

The index tag is also a <head> section tag of HTML, which declares the current HTML document is a searchable index.

To understand the functioning of search engines, some sample searches were made on three selected engines and the results are presented in Table 2.2.

From the results it is seen that, if the user paraphrases the same query in different ways, the search results not only differ in the number of sites returned but also in their ordering. For searches made on more specific domains with limited number of

Search Terms	Sites returned in Yahoo	Sites returned in AltaVista	Sites returned in Infoseek
reservation, flight, ticket	3	409010	384870
online reservation, flight, ticket	2	412130	3281979
online reservation, air, ticket	4	4205060	3281982
reservation OR booking, flight OR air, ticket	3	6980	1222003
+reservation, +flight, +ticket	3	4250	339

Table 2.2: Comparison Chart on Search Engine Results

relevant sites, the user has to search different options to get the desired sites. On the other hand, if the search is made on a more general topic having many relevant sites, the most appropriate search terms are essential to filter out irrelevant sites.

2.8 Natural Language Semantics

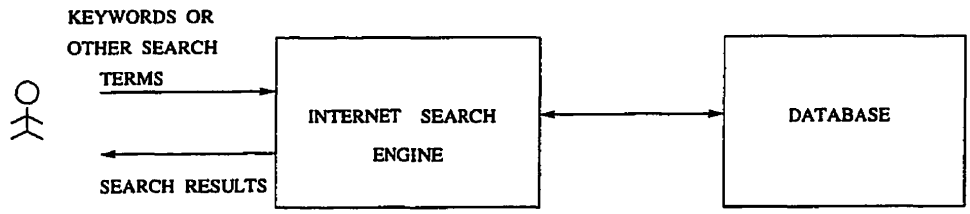
Semantic analysis of the user query could be helpful not only to relieve the user from finding the most appropriate search term, but also for the better performance of search engines by filtering out irrelevant sites to a great extent and providing meaningful listings. The rich semantics of natural language processing makes it a good choice for analysing the user's query content, before feeding it to the search engines.

There have been many applications for computer-based natural language understanding such as speech understanding, information retrieval, question answering systems, machine translation and document or text understanding. One such application is providing a natural language "front-end" to databases, which enables users to access information stored in databases without any need to know the structure of databases

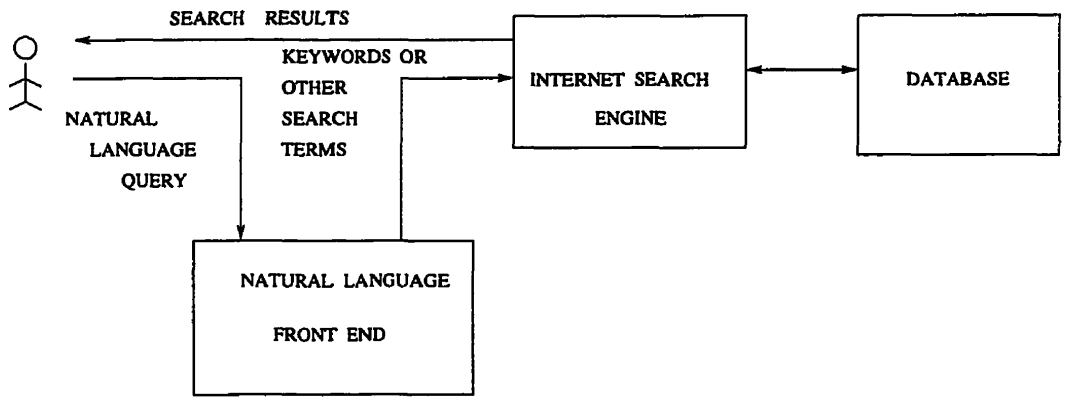
or any query language and without any need for transformation of their query to some other representation [7]. In this thesis, we provide a natural language "front-end" to Internet search engines, which allows users to utilize the search engines, without the need to find appropriate search terms. For a search as illustrated in Table 2.2, the user can present the query in natural language like: "I want to book a flight ticket" or "Show me some sites on online reservation of flight tickets" or even phrases like "online reservation of flight tickets". All these queries would yield the same search results.

Figure 2.1 shows the pictorial representation of existing search engines and the natural language "front-end" provided by us. In the existing search engines, the user has to study the various options provided by each search engine and transform the query into a form suitable for the specific search engine. The natural language "front-end" analyses the natural language query of the user and transforms it into appropriate search terms.

The overview of the natural language front-end is discussed in Chapter 3.



EXISTING SEARCH ENGINES



SEARCH ENGINES WITH NATURAL LANGUAGE "FRONT-END"

Figure 2.1: Pictorial Representation of Search Engines

Chapter 3

System Design Details

We have provided an interface which allows users to choose the search engine best suited for their search and enter the query in the English language. The natural language query is analysed both syntactically and semantically in order to select the most appropriate keywords describing the query. The keywords are interpreted to provide more meaningful search terms by using all the synonyms of the keywords in conjunction with boolean operators supported by individual search engines.

The architecture of the system is outlined in this Chapter. The system layout is discussed in Section 3.1. Features of the front-end of the system are discussed in Section 3.2. The natural language domain that can be supported by the system at present is discussed in Section 3.3. We outline some of the key features of the HPSG parser, along with its configuration in Section 3.4. Some linguistic problems are discussed in Section 3.5. The functions of semantic extractor and semantic interpreter are discussed in Sections 3.6 and 3.7 respectively.

3.1 Overall System Architecture

Figure 3.1 illustrates the architecture of the natural language access provided to the search engines. The natural language query to be searched, along with the choice of the search engine, is pre-processed in order to transform the query into a form suitable for input to the parser as the parser treats capitalized letters as variables and it does not accept special characters and punctuations. The parser, in turn, has

a description of grammar rules for capturing the constraints of the English language with a lexicon or dictionary which contains the words allowed in the input. The Head Driven Phrase Structure (HPSG) parser generates a complex feature structure representing the query. The semantic content of such a complex feature structure is extracted, interpreted and transformed into a form suitable for the search engine that was selected.

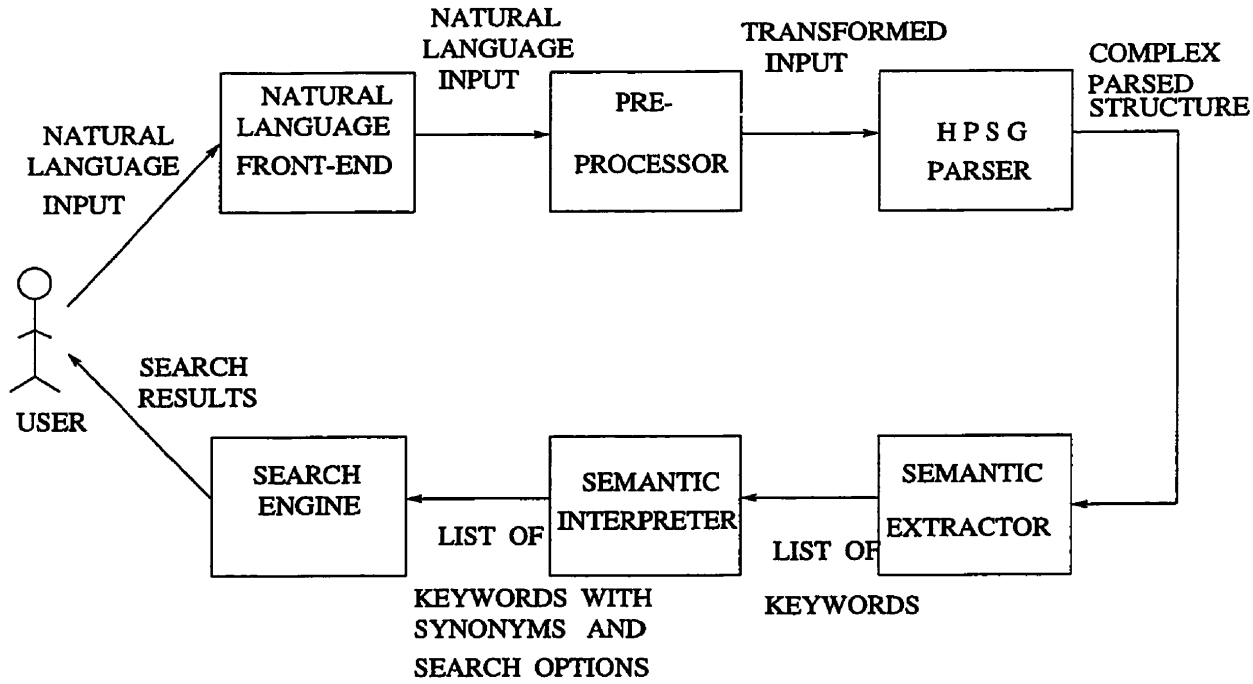


Figure 3.1: Overall System Architecture

3.2 Natural Language Front-end

The natural language front-end provided is a fill-in-form written in JavaScript with a back-end PERL script. The front-end also provides links to an alphabetical listing of words in the lexicon and tips for selecting a suitable search engine. Some of the positive features leading to the choice of JavaScript [20] for the front-end include:

1. Validations of fields can be done before submission of the forms to the back-end PERL script and appropriate error messages can be displayed. For example, if the user leaves any of the fields blank then the JavaScript displays appropriate error messages thus avoiding submission of blank fields to the back-end PERL script.

2. Alert messages can be displayed to warn the user of any possible errors. For example, as the user starts typing the query for search, a JavaScript box alerts the user to check the words in the lexicon.

After initial validations in JavaScript, the user input is submitted to the back-end PERL script. The PERL script verifies that there are no words present in the query which are not in the lexicon and it passes the query to the pre-processor.

3.3 Natural Language Domain

Capturing all domains in the lexicon for a vast collection of data that resides on the Internet is not an easy task. Therefore, we have limited our experiment and selected the "TRAVEL" domain as our model domain, with 150 English words representing various sub-domains like FOOD, ACCOMODATION, TRANSPORTATION, PLACES and WEATHER as shown in Figure 3.2. Under each domain or sub-domain there is a finite set of lexical entries selected in such a way to cover most general queries in that domain or sub-domain. The domain independent words include personal pronouns (I, we, etc.), relative pronouns (who, whom), control verbs (like, want, to), auxiliaries (can, may), and prepositions (at, between, in, on, etc.).

The lexicon can be extended over other domains as it has been designed to be modular with respect to the lexical knowledge base [7,24]. The domain independent knowledge base of the lexicon can be used across all domains.

3.4 HPSG parser

Before we describe the functioning of the HPSG parser, we outline some of the distinctive features of the HPSG formalism which led to the choice of HPSG parser

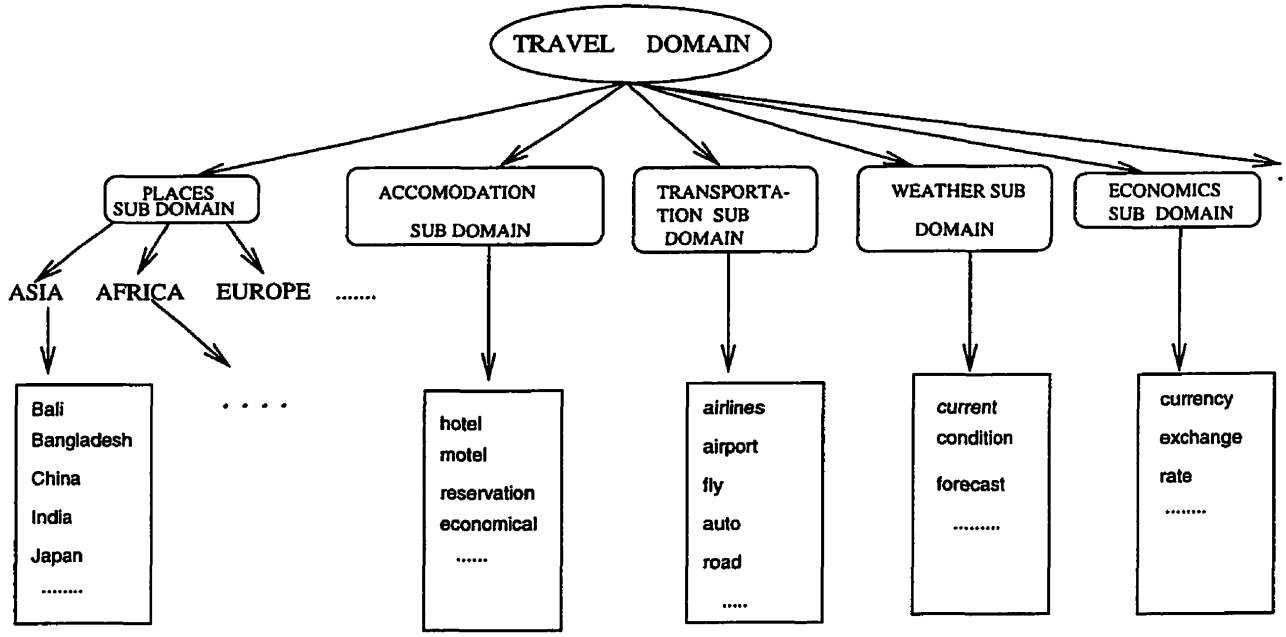


Figure 3.2: 'TRAVEL' Domain

for our system. HPSG is a unification based lexical grammar formalism which has increased emphasis on the lexicon rather than on the grammar rules. As the name indicates, the grammar is head driven which suggests that the central notion of a request is the key to a comprehensive response. This aspect makes it a more efficient parser as indicated in [7]. In HPSG, the knowledge representation is uniform in the form of feature structure or otherwise known as "Attribute Value Matrices" (AVMs), for lexical entries, grammar rules, and principles. Unlike other grammar formalisms, HPSG can parse words, phrases, sentences and questions. Since the feature structures contain information about syntactic and semantic properties of the object modelled, HPSG performs parallel syntactic and semantic analysis.

A feature structure in HPSG must be sorted and well-typed [28]. The sort symbol describes the type of object the structure is modelling. The attributes that can appear in a feature structure are determined by its sort. For example, if a feature structure is of sort *word*, it can have attribute labels PHON and SYNSEM; a feature structure of sort *synsem* can have attribute labels LOC and NONLOC.

3.4.1 Sign

The sign is the basic information-bearing structure in HPSG. The sign can be either lexical or phrasal. The phrasal signs have a daughters feature that gives information about the signs which are the immediate constituents of the sign in question. HPSG uses different classes of syntactic constituents such as head, adjunct, complement and filler and marker [28,7]. In a structure like *western region*, *western* is an adjunct (nominal adjunct) and *region* is the head. The adjuncts can be verbal adjuncts also as in *show me the travel guidebook for Canada.* The head determines all the syntactic properties of the phrase or sentence that contains it. Subcategorization of a lexical or phrasal sign is the specification of the number and kind of other signs that the sign in question has to combine before becoming grammatically saturated. Complements discharge subcategorization requirement on the head. Fillers discharge binding requirements on the head. Binding features provide information about long-distance dependencies for relative pronouns, interrogative expressions, and missing elements called gaps and traces. A marker (*that, for, than, as*) formally marks the constituent in which it occurs and combines with another element that heads the constituent.

As per the present formulation of HPSG [28], all signs possess at least two attributes, the PHON which represents phonology, and the SYNSEM. The SYNSEM attribute includes complex linguistic information on the syntactic and semantic properties of an object. The AVM representation of the sign is shown in Figure 3.3.

Local features correspond to the combinatorial properties of the sign. Non local features are concerned with long distance dependencies. LOC information is, in turn, divided into CAT, CONTENT and CONTEXT. The category attribute (CAT) represents the syntactic properties, such as the grammatical category (noun, verb, etc.) and its subcategorization requirements which are captured in its features HEAD and SUBCAT.

The CONT attribute specifies the sign's contribution to semantic interpretation. The semantic content of *gives* in HPSG will be as shown in Figure 3.4. The meaning of the word *gives* is viewed as a relation with arguments giver, given (to whom it is given), and gift (the object given). Their values are structure shared with that of

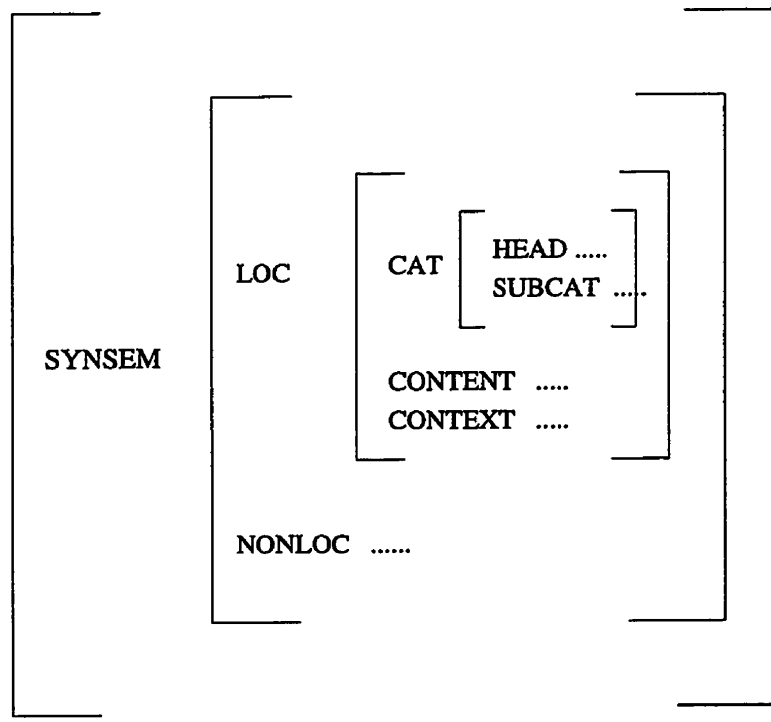


Figure 3.3: AVM representation of sign in HPSG

the SUBCAT feature. The CONTEXT attribute corresponds to presuppositions or conventional implicatures.

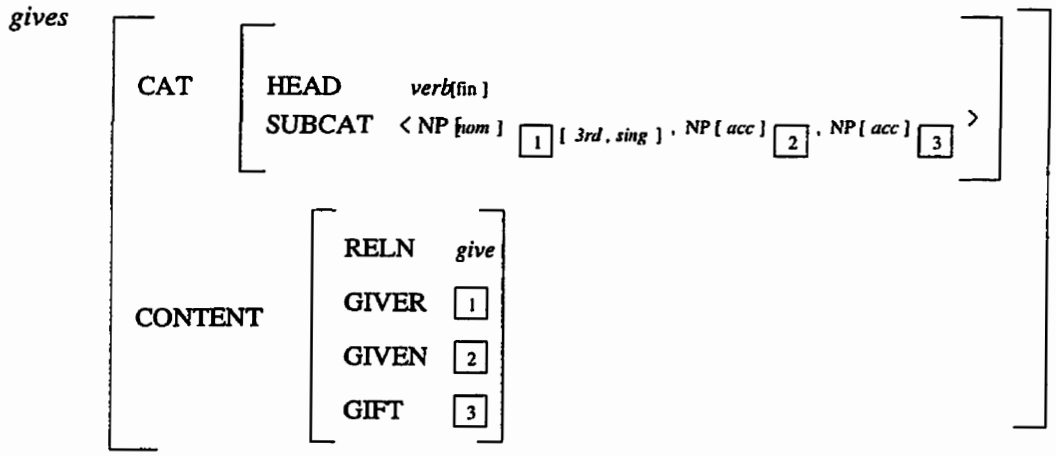


Figure 3.4: Semantic Content of *gives*

Both syntactic and semantic features have been modified in our thesis so as to have appropriate pointers to the keywords. The details of implementation are discussed in Chapter 4.

HPSG uses *unification* as its primary operation. Feature structures A and B unifies to form another feature structure C which has all the information contained in A and that in B but nothing more [27]. If A and B have conflicting information, then the unification fails.

3.4.2 Configuration

Figure 3.5 depicts the configuration of a parser. For parsing a natural language input, a parser requires:

1. a lexicon - which contains a finite set of lexical signs (words), that are permitted in the natural language input;
2. a finite set of grammar rules; and
3. a finite set of principles.

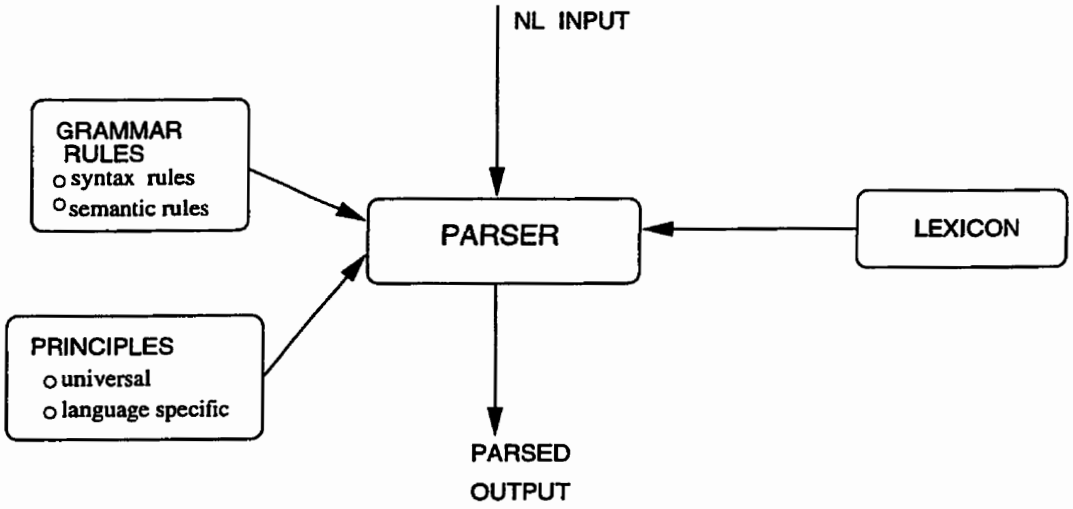


Figure 3.5: Parser Configuration

3.4.3 Grammar Rules and Principles

The grammar rules state how words are grouped into phrases and phrases into large phrases and sentences (or how large phrases are decomposed into their constituents). A grammar rule is a partially specified phrasal sign which constitutes one of the options offered by the language in question for making larger signs from smaller ones [27].

Rather than writing rules of the form "$S \longrightarrow$ NP VP" , "$NP \longrightarrow$ Det Nom " and so on, HPSG states rules in terms of "heads" and "complements". A single rule stating the structural relationship of a head daughter and complement daughters of a phrasal sign can replace 3 string rewriting rules ($S \longrightarrow$ NP VP, $NP \longrightarrow$ Det Nom, $NP \longrightarrow$ NP[GEN] Nom). Hence there are few grammar rules in HPSG. These rules interact with the principles such as the *Head Feature Principle, Subcategorization Principle* and *Semantics Principle*, to characterise a wide range of syntactic phenomena.

The Head Feature Principle states that the HEAD value of any headed phrase is structure shared or token identical with the Head value of the head daughter. As per the Subcategorization Principle, in a headed phrase, the SUBCAT value of the head daughter is the concatenation of the phrase's SUBCAT list with the list of SYNSEM

values of the complement daughter.

3.4.4 Organization of the Lexicon

As a lexical formalism, HPSG requires very descriptive and complex entries for the lexical signs. Two primary methods, namely multiple inheritance and lexical rules, are employed to organize the lexicon and eliminate redundant information. Multiple inheritance helps to eliminate vertical redundancy by classifying words that share common syntactic and semantic properties into a lexical type. Each class can have more than one superclass and each class inherits information contained in all of its superclasses. *Subsumption* (closely related to unification) is employed to define classes and superclasses. If A and B represent two feature structures, then A subsumes B, if B is at least as informative as A.

Lexical rules eliminate horizontal redundancy. Lexical rules are applied to base forms of a word to generate inflected or derived forms. Our lexicon is organized with multiple inheritance and lexical rules along with macros. Sample lexical entries and details of implementation of the lexicon are discussed in the next chapter.

3.5 Linguistic Problems

Ambiguities pervade natural language and are the root of a number of complex problems [12]. Ambiguities can be of several types such as lexical ambiguity, syntactic ambiguity and homographs.

Lexical ambiguity occurs when a word can be represented by more than one grammatical category. Some examples of lexical ambiguities are:

1. Some words can be either a noun or adjunct as in *travelling in Asia* and *I want to see the travel guidebook.*

2. Some words can be either a noun or a verb. For example, *I want to see the report* and *I want to report this to him* .

3. Some words can be either a determiner or a conjunction introducing a subordinate class [12] as in *that hotel* and *that hotel is expensive is a well-known fact.*

Lexical ambiguity can be resolved by syntactic analysis, since a place in a sentence where a noun can occur is different from the place where a verb can occur and so forth. Another type of lexical ambiguity is with homographs, which are words that have the same spelling but have different meanings. In the sentence *He banked his plane in the river bank near the bank where he banks*, the word bank has at least 4 meanings. Sometimes even the same syntactic category of a word can have more than one meaning. *Cheap hotel* can mean 'economical, moderately priced hotel' as well as 'poor quality hotel'.

Syntactic ambiguity is also a common problem which can be due to any one of the following causes:

1. In certain sentences, it is difficult to know if a complement should be attached to the noun which it follows or to the verb. The sentence *The man saw the girl with the telescope* is ambiguous because it is not clear whether it is the man or the girl who has the telescope [12].

2. Minor changes in word order can change the meaning entirely. *They have just built a flying machine* is very much different from *They have built just a flying machine* [5].

3. The same sentence may be interpreted in many ways. In the sentence *Visiting scientists can be interesting*, either scientists can be interesting or the act of visiting.

Pronoun references, negation, conjunction and disjunction can all add to ambiguities as in the cases mentioned below:

- In the sentence *The soldiers fired at the women and I saw several of them fall*, "them" can refer to the soldiers or the women.

- The sentence *John is not going to fly to Toronto* can have at least four interpretations [12]:

  1. It is not John who is flying to Toronto; it is somebody else.
  2. John is going to Toronto but not flying (may be driving).
  3. John is flying but not to Toronto.
  4. John is not going to fly to Toronto (he may just not be going anywhere).

- The word "and" sometimes denotes disjunction rather than conjunction. *List all hotels in Regina and Saskatoon* normally means hotels either in Regina or in Saskatoon rather than hotels in Regina and Saskatoon.

Also the words "and" and "or" can be interpreted in more than one way as indicated below:

1. List all (hotels and motels) in South Africa.
2. List all hotels and (motels in South Africa).
3. List all (hotels and motels in South Africa).

The users can paraphrase their query in different ways. A search for sites on current weather can be expressed as:

- What is the current weather in South Africa?
- I want to see all the sites on current weather in South Africa.
- List all sites on current weather in South Africa.

All the above expressions should yield the same semantic representation or the same keywords for search in this case.

Most of the problems cited above can be resolved by the parallel analysis of syntactic and semantic representations which is one of the very distinctive features of HPSG. In our limited domain of 150 words in the lexicon, the above stated problems can be handled with the extensive semantic knowledge base. However, for future

extensions to other domains, our general approach would be to prompt the user to choose from alternatives to resolve the ambiguity, in the cases where it can not be resolved by the parser.

3.6 Semantic Extractor

The output of the HPSG parser will be a complex feature structure encoding the syntactic and semantic content of the sentence/phrase parsed. The information encoded in the CONT feature of this complex feature structure is to be extracted for further interpretation. In HPSG, since the feature structures are sorted and well-typed, the feature CONT appears more than once in the sign. Figure 3.6 shows the sorted feature structure adopted in the parser [28,24]. In our parser, the type synsem has LOC feature. The LOC feature, in turn, can have CAT and CONT features. Since the CAT feature is of type 'cat' it can have a HEAD feature and SUBCAT feature. The SUBCAT feature is of type synsem-list which in turn could be a synsem or synsemlist depending on the number of signs subcategorized for. Because of the well-typed nature, *synsem-loc-cat-synsemlist-synsem* will also have LOC feature encoding CAT and CONT.

Pointers to the keyword list will be present in the *phrase-synsem-loc-cont* path. More over, not all words in the query will contribute towards the keywords. Hence the function of the semantic extractor module is twofold:

1. To extract the value of the CONT feature appearing in the *phrase-synsem-loc-cont* path, from the complex feature structure output of the parser.

2. To extract the list of keywords alone form the semantic content extracted.

3.7 Semantic Interpreter

Traditional parsers use a set of syntactic grammar rules along with a lexicon to parse the natural language. On the parsed results, semantic rules are applied to convert the parsed result into a logical query [3]. In our thesis, semantic analysis is

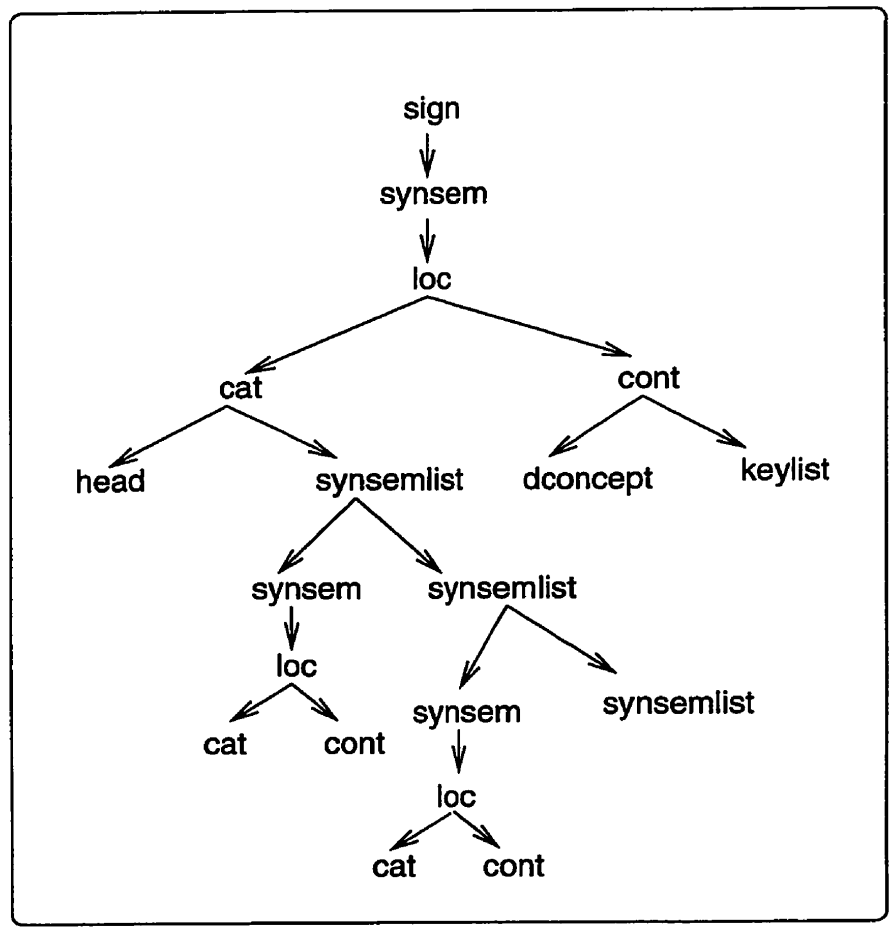


Figure 3.6: Sorted Feature Structure Adopted

carried out in two phases. Since the HPSG parser is used, critical semantic analysis is done in the parser itself. Further detailed analysis is performed in the semantic interpreter as illustrated in Figure 3.7.

The list of keywords from the extractor are converted into appropriate search terms using the semantic rules, synonyms list and search engine information. Semantic rules are primarily applied to group nominal compounds. For example, in *south africa*, the word *south* would be treated as a adjective in the lexicon. However *south africa* has to be present as the name of the country in the search term. In order to ensure that no relevant sites are missed, all the synonyms of keywords where ever applicable are included in the search terms using the boolean option 'OR'. The search engine information provides the details of search options supported by individual search engines so as to map the search terms with the search engine selected. While doing so, the interpreter identifies keywords that must be present in the sites listed and encodes them with special signs like '+'.

The implementation details of each module of the system are discussed in the next Chapter.

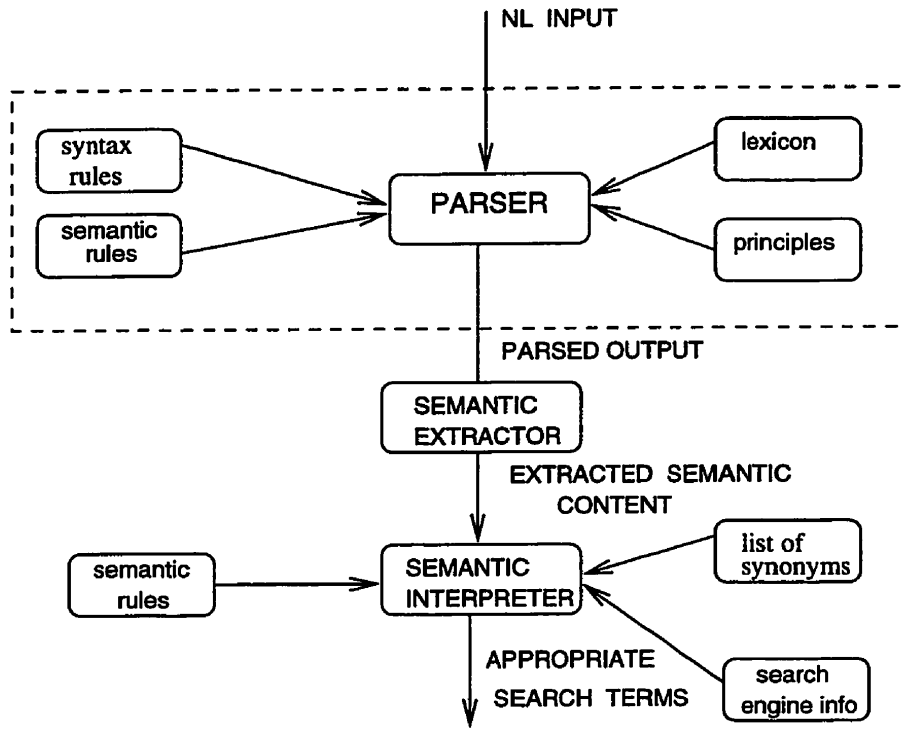


Figure 3.7: Two Phases of Semantic Analysis

Chapter 4

System Implementation

We discuss the implementation details of system modules in this Chapter. The implementation of the natural language front-end is discussed in Section 4.1. In Section 4.2, we outline the implementation of the HPSG lexicon. The implementation of the HPSG parser's semantic extractor and interpreter are discussed in Sections 4.3 and 4.4.

4.1 Natural Language Front-end Implementation

As outlined in Chapter 3, the implementation of the front-end is realized as a JavaScript form that allows users to choose a search engine and enter the text to be searched. Figure 4.1 shows the layout of the front-end developed for this system. Frames have been used for organization. The front-end also provides links to:

- some major search engines;
- alphabetical listing of words in the lexicon and
- tips for selecting an appropriate search engine.

4.2 HPSG Lexicon Implementation

The HPSG lexicon is developed using the Attribute Logic Engine (ALE) version 2.0.1, as the lexical representation language. ALE integrates phrase structure parsing

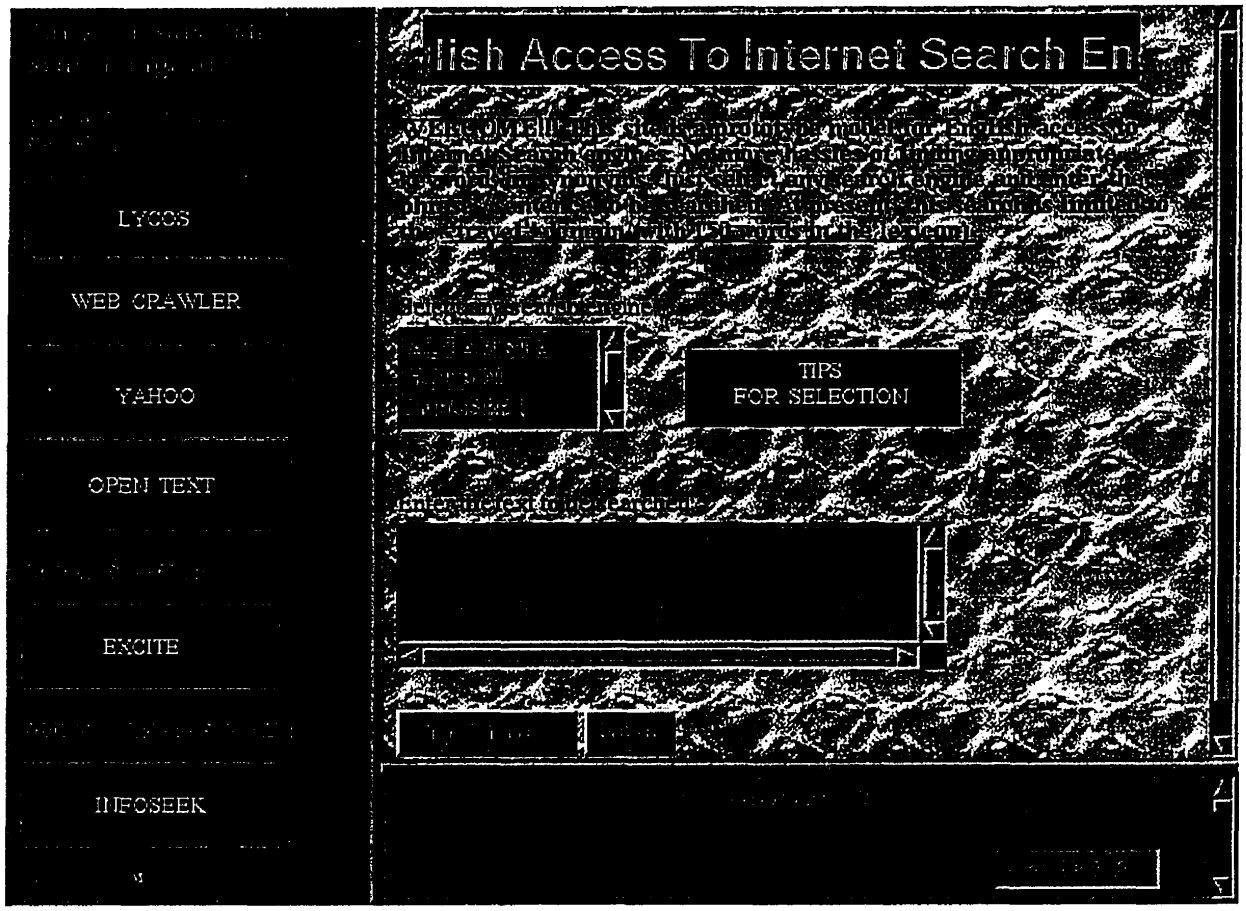


Figure 4.1: Layout of the Front-end

and constraint logic programming with typed feature structures as terms of representation [6]. The details of the typed feature structures adopted are discussed in the next subsection. ALE employs a bottom-up chart parser and a compiler which compiles grammars into Prolog parses [6,21]. In addition to type hierarchies, ALE can also handle macros and lexical rules which are used for organizing the lexicon. The EMACS user interface provided for ALE facilitates easy compilation and modifications [6].

4.2.1 Typed Feature Structures

In ALE, every feature structure used must be typed. Types specify the features that can appear and the values these features can take. In ALE, types are arranged in an *inheritance hierarchy*, whereby constraints on more general types are inherited by their more specific subtypes. If a feature is appropriate for a type, it will also be appropriate for all its subtypes. Some general properties and points to be noted regarding typed feature structures in ALE are listed below [6]:

1. The relation of sub-typing is taken to be transitive. That is, if a is a subtype of b, and b is a subtype of c, then a is also a subtype of c. It is enough if the user specifies the direct subtyping relationship. The transitive relation will be automatically computed by the compiler.

2. The derived transitive subtyping must be anti-symmetric. That is, there should not be two distinct types each of which is a subtype of the other.

3. There must be a unique type (referred as bot), which is the most general type. All other types must be a subtype of this general type (bot).

4. Every pair of types which have a common subtype must have a unique, more general subtype.

In our lexicon, **bot** is the most general type which has three subtypes: system, syntax and semantics. Figure 4.2 illustrates the hierarchy of bot and system. Boolean has two subtypes: minus and plus. The list can be either an empty list (elist) or a non-empty list (nelist). A non-empty list will have two features namely the head (hd) and the tail (tl). The head can be of type bot whereas the tail must be a list. Apart from empty and non-empty lists we also have synsemlist, signlist and keylist. These lists can also be either a non-empty list or an empty list. The non-empty synsemlist, signlist and keylist (nesysnsemlist, nesignlist and nekeylist) have two features, head (hd) and tail (tl). Unlike a non-empty list which can have its head of type bot, non-empty synsemlist, signlist and keylist can have objects of type synsem, sign or keyword only as their heads. The synsemlist, signlist and keylists are used in turn for defining the SUBCAT feature, COMP-DTRS feature and CONT feature respectively.

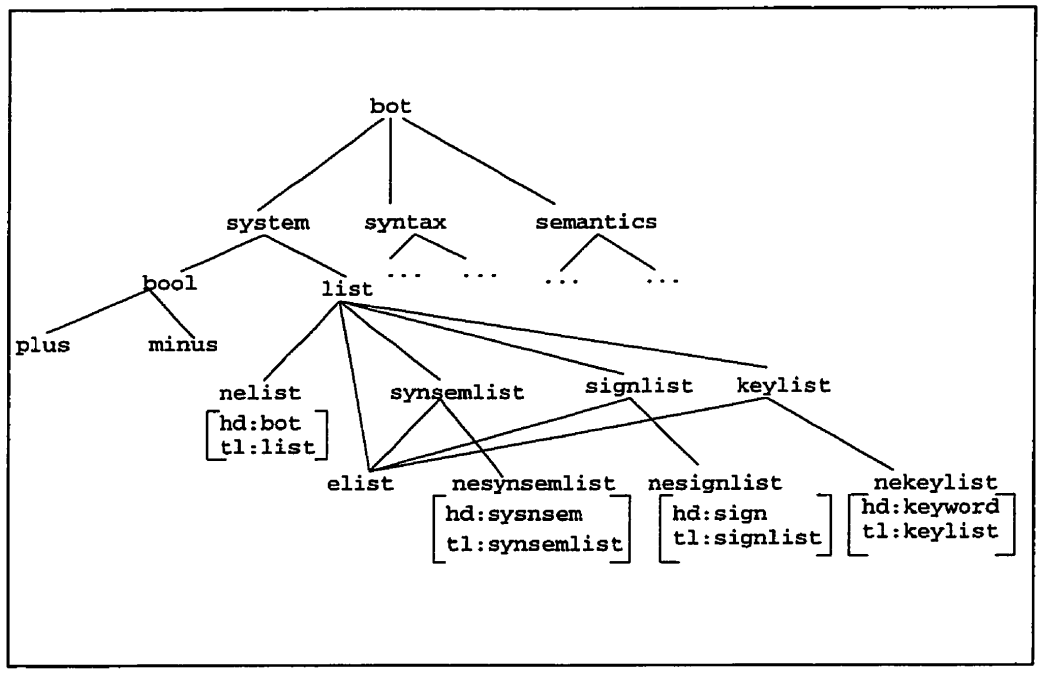


Figure 4.2: Hierarchy of **bot** and **system**

The syntax and semantics have features representing the syntactic and semantic information of the object. Sign is a subsort of syntax which in turn has two subsorts: word and phrase. Both word and phrase have an attribute synsem. Synsem has an attribute local (loc). Local information is in turn divided into category (cat) and content (cont) attributes. The category value includes information about the syntactic category of the word and the grammatical arguments required. This information will be captured in four attributes: head, subcat, specifier (spr) and marking. The hierarchy of head is shown in Figure 4.3. The representation of the head feature in ALE is shown in Figure 4.4.

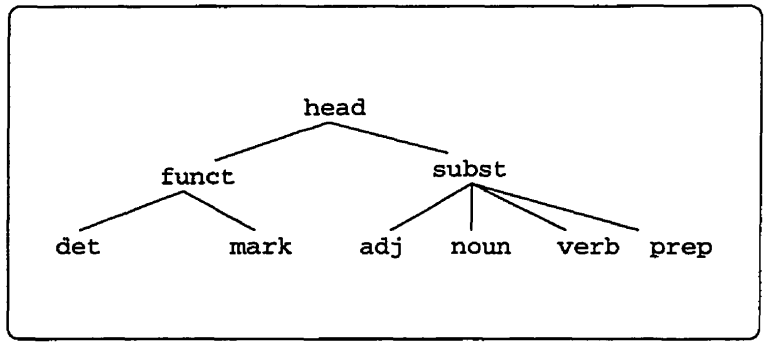


Figure 4.3: Hierarchy of **head**

The content (cont) feature constitutes the word's contribution to the semantic interpretation of the phrase that contains it. The value of the content feature, as illustrated in Chapter 3, is not exactly suitable for our system, since the content feature must encode the word's contribution to the keywords. Hence the content feature is modified to include the feature keyword. The keyword feature is a keylist whose head is a keyword and its tail is a keylist. The keywords have two subsorts namely general concept (gencon) and the specific concepts (specon). Under the general concepts, words which do not contribute to the keywords like *i, web,* and *online* will be listed. The keywords are listed under specific concepts.

When extending to other domains, only the content feature needs to be modified to include keywords from that domain. The syntax and system can be retained across domains.

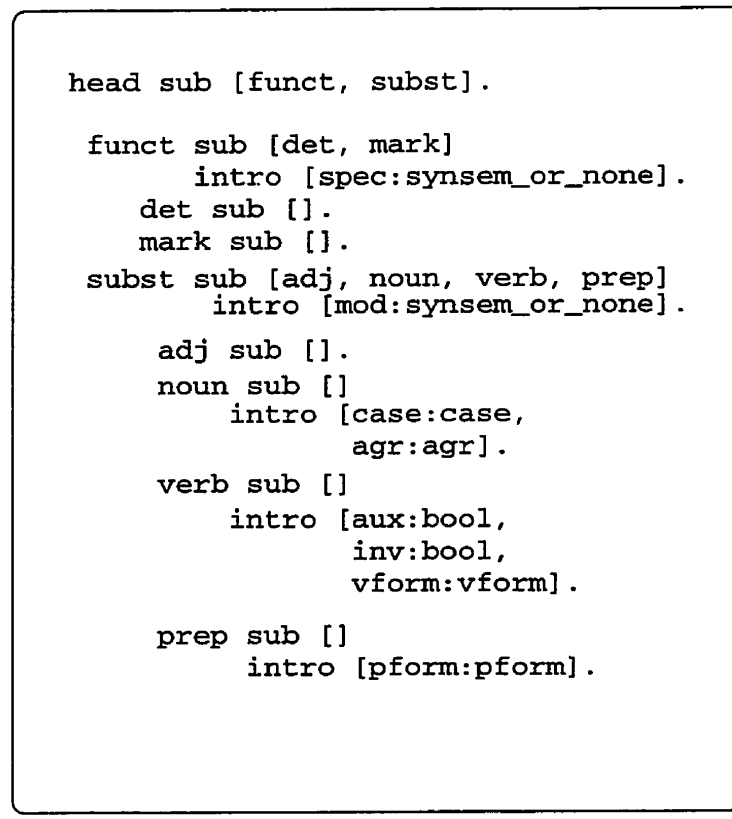


Figure 4.4: Hierarchy of **head** in ALE

4.2.2 Macros and Lexical Rules

Specifying full paths everywhere in the lexicon will make it difficult to read and to maintain consistency in representations. Macros allow users to identify a description with a name and later refer to the description by its name. For example, **head_s(noun) macro(loc:cat:head:X)** refers to (loc:cat:head:X). Later, this macro can be referred to as: **@head_s(noun)** to denote (loc:cat:head:noun). A macros can use other macros in its definition. For example, **vform_s(X) macro @head_s(vform:X)** refers to (loc:cat:head:vform:X). Macros reduce redundant information in the lexicon to a great extent. Figure 4.5 shows some sample entries from the lexicon after defining the corresponding macros.

```
south    ---> @adjective_lex,
              @keyword( south ).

africa   ---> @npNoCompNoMod_lex,
              @keyword( africa ).

i        ---> @ppronSubj_lex.

give     ---> @ditrans_lex,
              @keyword( gencon ).

on       ---> @preposition_lex(on).
```

Figure 4.5: Sample Lexical Entries

Lexical rules are used in our lexicon to reduce the number of entries. Instead of having separate lexical entries for each inflected form of a verb, only the base form is entered. Lexical rules are applied to the base form to derive its inflected forms. One such lexical rule used in our lexicon for generating the gerundive form of a verb from its base form is illustrated in Figure 4.6. This rule is used to parse phrases like "visiting South Africa". The input feature structure represents the base form. The output feature structure is exactly the same as the input but in the gerundive form. Morphs statements ensure that the gerundive form of *stay* will be *staying* whereas

that of *give* will be *giving* and *travel* will be *travelling.*

```
gerform lex_rule
   (word, @head(verb),
          @vform(bse),
          @aux(minus),
          @mod(Mod),
          @inv(Inv),
          @subcat([Subcat|Moresubcats]),
          @cont(Cont))
**>
    (word, @head(verb),
           @vform(bse),
           @aux(minus),
           @mod(Mod),
           @inv(Inv),
           @subcat([Subcat|moresubcats]),
           @cont(Cont))
morphs
     (X,e) becomes (X, ing),
      travel becomes travelling,
      X becomes (X, ing).
```

Figure 4.6: Sample Lexical Rule

4.2.3 Grammar Rules

Grammar rules are used in HPSG to govern the combination of constituents like head, complement, adjunct and filler, to form large phrases/sentences. As stated in Chapter 3 very few rules are sufficient in HPSG to capture a wide variety of string rewrite rules. In HPSG, four rules are sufficient to govern the head/complement structures and head/adjunct structures. In this section, we discuss two such rules: Schema 1 and Schema 2 and the implementation of Schema 2 in our lexicon.

Schema 1, shown in Figure 4.7, governs the combination of a non-lexical head with its final complement. This rule states that one of the possibilities of a phrasal sign in English is to be a saturated sign (SUBCAT <>) which has as its constituents a complement daughter (COMP-DTR < [] >) and a non-lexical head daughter [27]. Thus this rule can replace the traditional rules: $S \longrightarrow$ NP, VP and $NP \longrightarrow$ Det, N'.

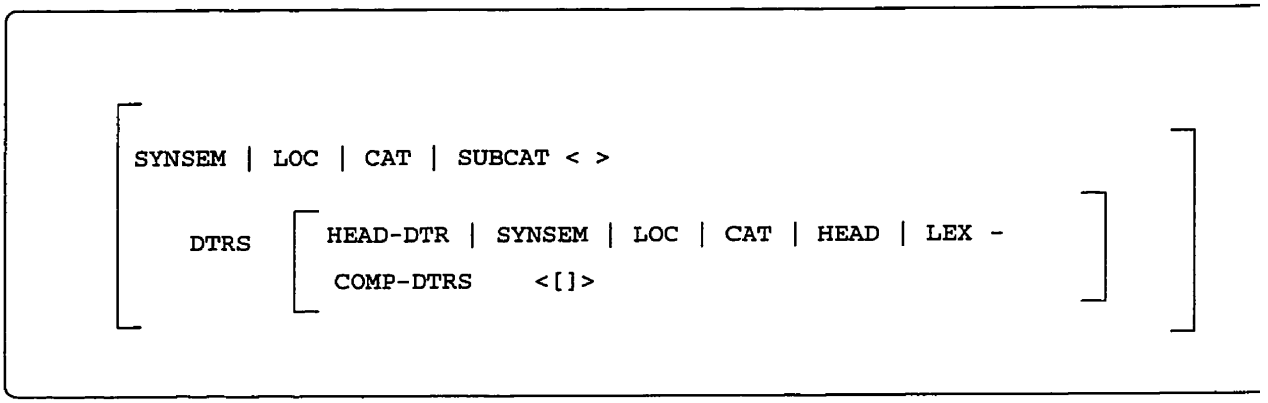


Figure 4.7: Schema 1

Schema 2, shown in Figure 4.8, on the other hand, governs the combination of a lexical head with zero or more complement daughters except its final complement daughter. This rule states the possibility of an unsaturated phrasal sign (SUBCAT < [] >) in English whose head daughter is an uninverted lexical sign (INV -). This rule can also convert a lexical head requiring a single complement to a non-lexical constituent requiring a single complement. Thus, this rule captures the generalizations of the following string rewrite rules:

$VP \longrightarrow$ V, NP; $VP \longrightarrow$ V, NP, NP;

$VP \longrightarrow$ V, PP; $PP \longrightarrow$ P, NP;

$VP \longrightarrow$ V ; $N' \longrightarrow$ N;

The implementation of Schema 2 in ALE is illustrated in Figure 4.9. Mothers and daughters are separated by the symbol "===>". Daughters are prefixed with "cat>". To specify a list of complement daughters, the prefix "cats>" is used. The constraints on the rule can be specified on the right hand side of the rule by using definite clauses in the "goal>' statement. Principles are also used in "goal" statements

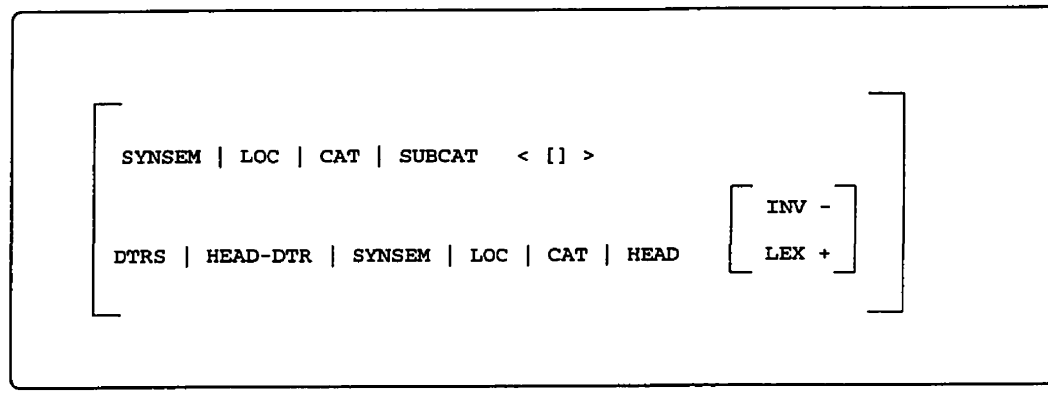


Figure 4.8: Schema 2

to enforce further restrictions on categories in rules. For example, in this schema, if we want to say that the mother category inherits the same head features as the head daughter, then we include the head_feature_principle in the "goal" statement.

```
schema2 rule
(Mother, phrase, @subcat([SubjSynsem]))
===>
cat> (HeadDtr, word, (@subcat([SubjSynsem|CompSynsem]))),
goal> (synsem_to_phrase(CompSynsem, Comp),
       is_not_subject(Comp, SubjSynsem)
       ),
cats> Comp,
goal> (principles(Mother, HeadDtr, HeadDtr, Comp, CompSynsem)).
```

Figure 4.9: Sample Grammar Rule in ALE

As illustrated in the figure, the mother category is a phrase which is missing its subject. The head daughter is a word which subcategorizes for subject and other complements. Since the complement daughters can be zero or more as per the schema, we use "cats>". Definite clauses are used here to ensure that necessary matches are

effected. The first constraint is for converting the type *synsem* to *phrase* as the head's SUBCAT are of type *synsem*, whereas the complement daughters will be of the type *phrase*. To ensure the complement daughter is not the subject, we use the second constraint. Since more than one principle is to be used for constraining the applicability of the rule, we use the definite clause "principles". The definite clause "principles" is defined as illustrated in Figure 4.10.

```
principles(Mother, Head, SemHead, Comps, CompDtrsSynsem) if
          (head_feature_principle(Mother, Head),
           subcat_principle(Mother, Head, CompDtrsSynsem),
           marking_principle(Mother, Head),
           semantic_principle(Mother, SemHead, Comps)
           ).
```

Figure 4.10: Definition of Principles

4.2.4 Principles

In HPSG, principles are applied in conjunction with the grammar rules. In this section we discuss the implementation of one of the principles in our lexicon, namely the Semantic principle. As per HPSG formalism [28], the semantic principle states that the semantic content of a phrase is "token identical" with that of the adjunct daughter in a head-adjunct structure, and with that of the head daughter otherwise. However, in our lexicon, this principle needs to be modified since, daughters other than the head daughter can also contribute to the keywords. Hence, the semantic principle in our lexicon, as illustrated in Figure 4.11, ensures that the semantic content of the mother is the unification of the semantic contents of semantic head with all other complement daughters.

While unifying the content of all the daughters with the head daugther, we require two separate functions, as shown in Figure 4.12, because the content feature is of type *keylist* which has a *keyword* as its head and a *keylist* as its tail, and the tail can be either an empty-list or a non-empty list.

```
semantic_principle( Mother, SemHd, [Comp|Comps] ) if
                unify_cont_features( SemHd, Comp, NewSemHd ),
                semantic_principle( Mother, NewSemHd, Comps ).
```

Figure 4.11: Semantic Principle in our Lexicon

```
unify_cont_features((@cont(keyword:(ne_key_list, (hd:KeyHd,
                                                  tl:e_list)))),
                    (@cont(keyword:(ne_key_list, (hd:KeyComp,
                                                  tl:e_list)))),
                    (@cont(keyword:(ne_key_list, (hd:KeyHd,
                                        tl:(ne_key_list, (hd:KeyComp,
                                                          tl:e_list))))))
                    ) if
                    !, true.


unify_cont_features((@cont(keyword:(ne_key_list, (hd:KeyHd,
                                                  tl:KeyHdTl)))),
                    (@cont(keyword:(ne_key_list, (hd:KeyComp,
                                                  tl:CompTl)))),
                    (@cont(keyword:(ne_key_list, (hd:KeyMother,
                                                  tl:KeyMotherTl))))
                    ) if
      !, join_finding((ne_key_list, (hd:KeyHd,
                                     tl:KeyHdTl)),
                      (ne_key_list, (hd:KeyComp,
                                     tl:CompTl)),
                      (ne_key_list, (hd:KeyMother,
                                     tl:KeyMotherTl))).
```

Figure 4.12: Unification of CONTENT Feature

As a sample output of this unification, the CONT feature from the parse result of a sentence "I want to visit South Africa" is shown in Figure 4.13. The heads "gencon" (general concept) are the contribution of words like *i*, *want*, and *to*. The word *visit* contributes the keyword "travel".

```
CONT cont
     KEYWORD ne_key_list
             HD gencon
             TL ne_key_list
                HD gencon
                TL ne_key_list
                   HD travel
                   TL ne_key_list
                      HD south
                      TL ne_key_list
                         HD africa
                         TL ne_key_list
                            HD gencon
                            TL e_list
```

Figure 4.13: Sample Output of Unification

4.3 Implementation of Semantic Extractor

The primary predicate for parsing in ALE is: **rec(String)**, where **rec** stands for recogoniser and **String** is the input string entered as a Prolog list of atoms. The output will be a category (in the form of typed feature structure) derived for the input [6]. However, this predicate does not provide parameters for retrieving the output [24]. Instead, the predicate **rec(String, Tag, Struct, Ineq)** is used. Here also, the parameter **String** stands for the input string. Since we do not use inequalities, the parameter **Ineq** is not specified. The output of this predicate will be in the form of **Tag** and **Struct**, the form in which ALE encodes feature structures internally. A

sample output of this predicate for the input string [i, want, to, visit, south, africa], will be as shown below:

```
Ineq = [],
Struct = phrase(_A1-synsem(_Z-loc(_Y-cat(_X-verb(_W-bool,_V-bool,_U-none,
_T-bse),_S-unmarked,_R-synsem_list,_Q-e_list),_P-cont(_O-db_action,
_N-ne_key_list(_M-gencon,_L-ne_key_list(_K-gencon,_J-ne_key_list(_I-travel,
_H-ne_key_list(_G-south,_F-ne_key_list(_E-africa,_D-ne_key_list(_C-gencon,
_B-e_list)))))),_A-dconcept)))) ?
```

The semantic extractor program, written in Prolog, extracts the list of keywords from such a complex structure. The CONT feature for the input string [i, want, to, visit, south, africa], is a non-empty keylist whose heads are [gencon, gencon, travel, south, africa]. The "gencon" denotes the general concept corresponding to the words "i", "want" and "to". The ordering of the keywords in the list depicts the unification of semantic head with the complement daughters. The semantic extractor removes the "gencon" (general concept) heads from the list. Hence, output of the semantic extractor for the above parse will be a list with only three keywords: [travel, south, africa].

4.4 Implementation of Semantic Intrepretor

The semantic intrepretor written in PERL script handles the following tasks:

1. The output of the semantic extractor will be a parsed list of keywords like [travel, south africa], for successful parses. Otherwise, it would be a string 'No parse' denoting the syntactic error. If the input text gets sucessfully parsed but without having any keywords, the extractor will be returning a empty list. For both the syntactic error and no keywords error, the intrepretor displays appropriate error messages.

2. In nominal compunds like "South Africa" and "New York", the words "south " and "new" are treated as adjectives by the parser and hence will be contributing

for two separate keywords. These keywords are to be unified to represent a nominal compund, in order to ensure that synonyms are not added to them. For example, the word "new" will be replaced with "new OR recent" if treated as a separate keyword.

3. After grouping the nominal compounds, the synonyms for keywords if any are logically "OR"ed.

4. The list of keywords is formatted to suit the individual search engines. For example, the search engine HOTBOT, can not handle "+" sign, denoting the words that must be present in the sites returned, with boolean options.

5. Finally, the sematic intrepretor passes the formatted list of keywords to the selected search engine using the form method "GET". Options provided exclusively by certain search engines are chosen as default values using hidden variables. Figure 4.14 shows the script for passing the formatted list of keywords to the search engine HOTBOT. The value of the variabe $hot represents the default and boolean option selected for searching.

```
if ($sename eq 'HOTBOT') {
print STDOUT "<FONT SIZE = 5>";
print STDOUT "FORM ACTION=\"http://www.hotbot.com/\" METHOD=\"GET\">";
print STDOUT "<INPUT TYPE=\"submit\" VALUE=\"CONTINUE\">";
print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"sw\" VALUE=\"web\">";
print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"SM\" VALUE=\"$hot\">";
print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"MT\" VALUE=\"$str3\">";
print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"DE\" VALUE=\"1\">";
print STDOUT "</FORM>";
print STDOUT "</FORM>";
}
```

Figure 4.14: Sample Script for Passing Keywords to Search Engine

The experimental results are detailed in Chapter 5.

Chapter 5

Experimental Results

In this Chapter, we present some experimental results of our natural language interface to access some major contemporary search engines. Some factors considered for assessing the quality of search results obtained are outlined in Section 5.1. The error handling features of our system are discussed in Section 5.2. Some sample inputs and their corresponding outputs are presented in Section 5.3.

An additional set of sample inputs and outputs is presented in Appendix A.

5.1 Factors for Comparison

Various factors are to be considered for assessing the quality of the search result, which is one of the prime reasons that, thus far, none of the search engines have been adjudged as the best. We restrict ourselves to consider the three key factors for assessing the quality of the search results:

1. the ease of expressing the query by the user to get the desired output;
2. number of sites returned and
3. the relevancy of the sites returned to the query.

The ease of expressing the query is a significant factor, because each search engine provides its own set of search options ("boolean", "+", "-", etc.). A query can become quickly complicated, if there are more than one or two keywords with some of the

keywords having few synonyms. In some of the existing natural language search engines, the user has to pick appropriate words for getting the desired results, as words other than keywords also contribute to the search.

Quite often, a very large number of sites are returned while searching. The search engines normally return all sites that matches any or all of the keywords in the query. Hence for a query, "visiting Japan", sites on visiting Professor from Japan may also be listed, even though this site may not be of interest to the user. The number of sites returned can be considered as an important factor for comparison if the relevancy is maintained and appropriate sites are not missed.

There are two significant reasons for getting large number of sites for a query:

1. the keywords selected may not be precise enough and

2. in some natural language search engines, words other than keywords contribute to the search.

For example, if we consider the following two queries:

"Which airlines fly between Canada and Japan?" and

"What are all the airlines flying between Canada and Japan?".

Both of these queries should yield the same search results in a natural language search engine, not only in the number of sites listed but also in their ranking.

Relevancy of the sites returned also depend on the keywords matched. Hence in keyword search engines, a search for "cheap hotels in Toronto" might also list a site related to "cheap air tickets to Toronto" as the most matching site since the two keywords "cheap" and "Toronto" are present. In NLAISE we refer to the term "relevancy" as interpretation of natural language text to the appropriate search term. For example, if the user presents the query as "I want to stay in Vancouver for two days", then NLAISE will semantically interpret this query for a search of sites on hotels or motels or lodges in Vancouver. Instead if keywords such as "staying, Vancouver" are searched without NLAISE, the sites on hotels or motels in Vancouver may not be listed as top ranking ones.

The sample input and output are chosen in such a way to facilitate analysis of the above factors.

5.2 Sample Output

Before we analyse the quality of the search results, one complete set of search using our system Natural Language Access to Internet Search Engines(NLAISE) is presented. This search was done using AltaVista for the text, "What is the current weather in South Africa?". Figure 5.2 illustrates the system response to the query, and Figure 5.3 provides the output in AltaVista (a keyword search engine). For better view, Figure 5.4 displays the front page of the search result.

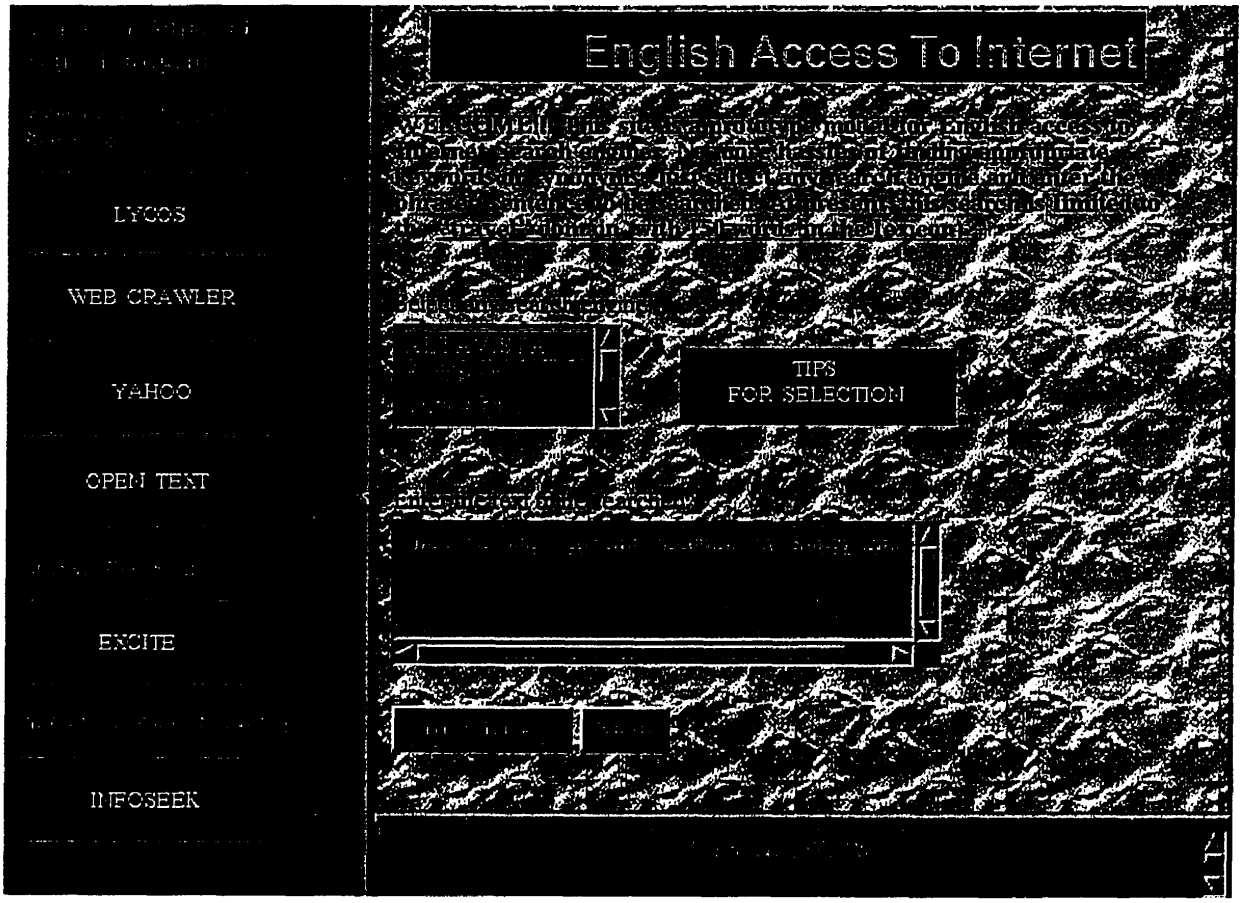


Figure 5.1: Search for "What is the current weather in South Africa?" in AltaVista

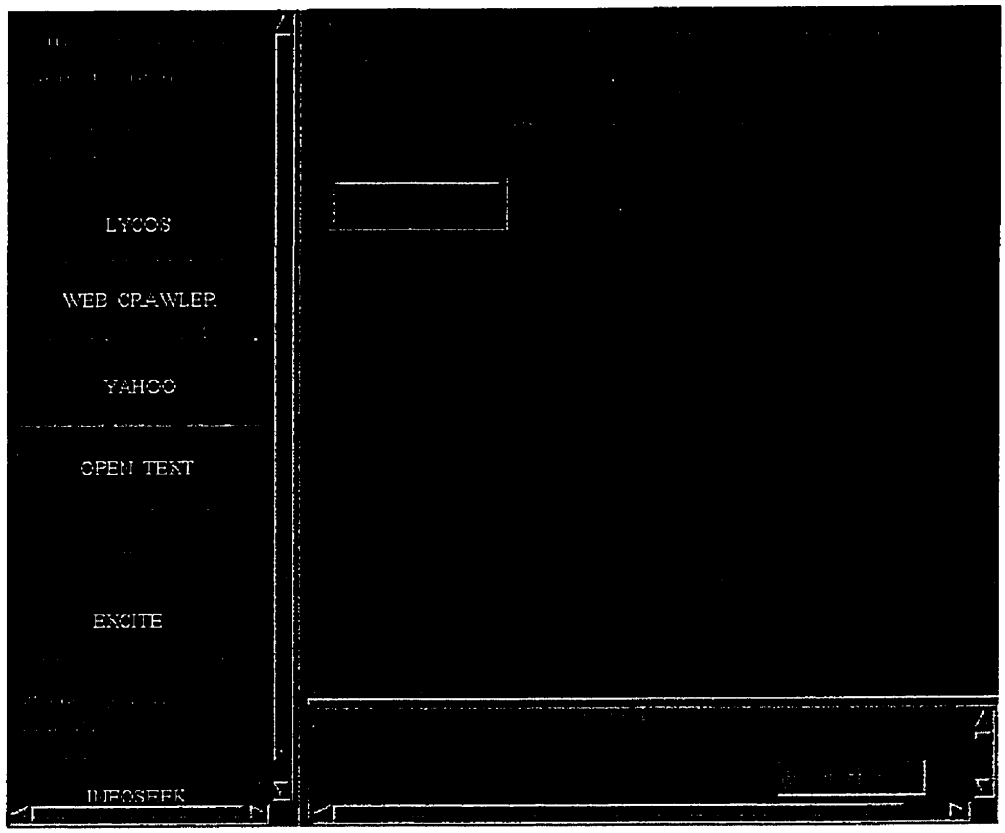


Figure 5.2: NLAISE System Response

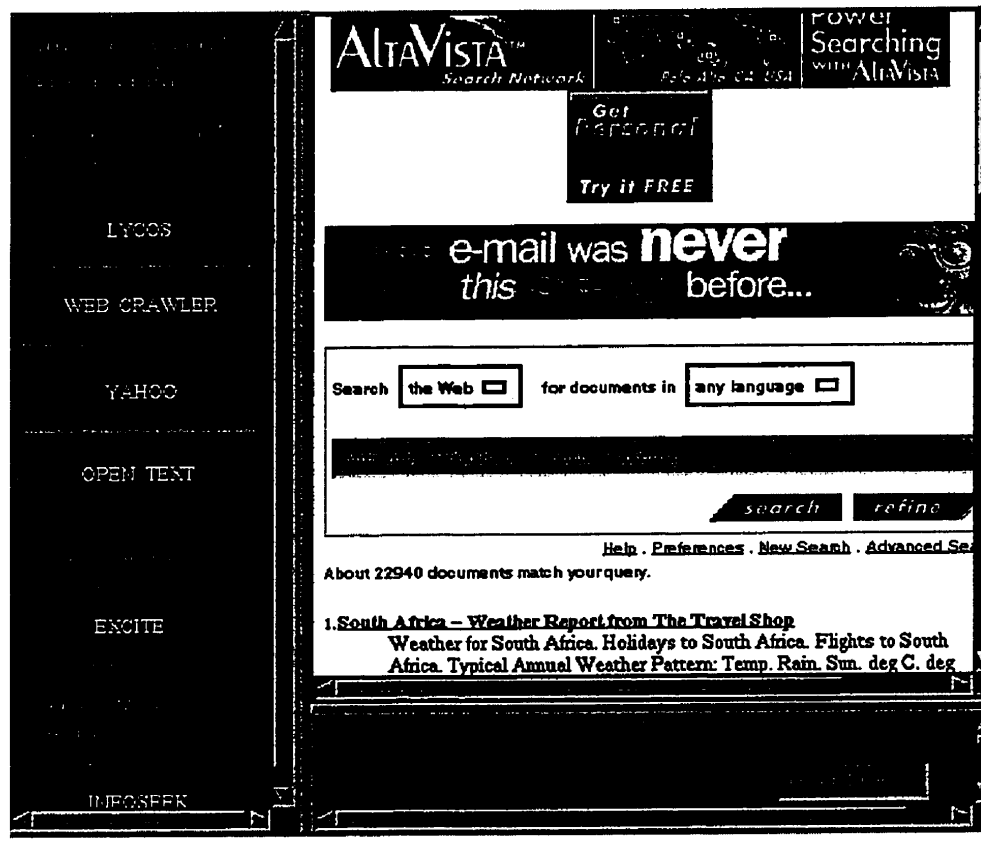


Figure 5.3: Output for the Search

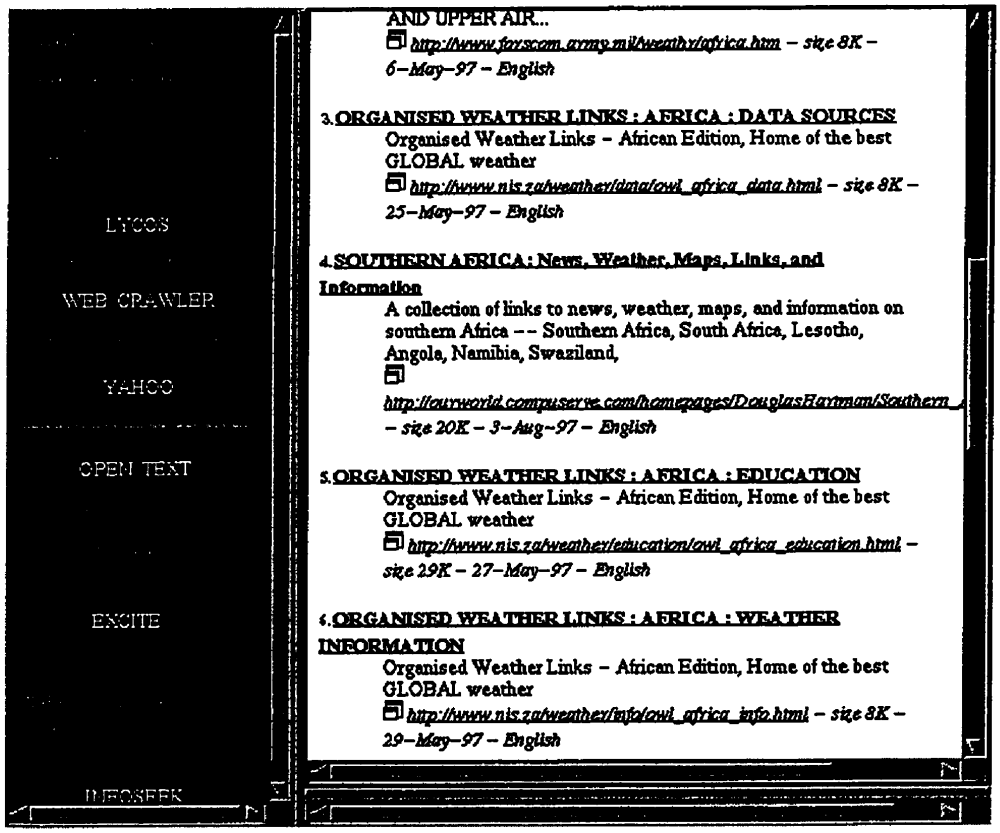


Figure 5.4: Listing of the Front Page of the Results

Around 50 sample searches were made with and without NLAISE in each of the 5 selected search engines. In this section we have tabulated some of the search results to facilitate comparisons based on the factors mentioned in section 5.1. We have presented results for comparison from two search engines, namely AltaVista and Infoseek in Tables 5.2 - 5.5. Comparison of results from other search engines are presented in Appendix A.

Table 5.1 illustrates the ease of expression in our system. NLAISE can handle a word, words, phrase, sentence and question, which is one of the special feature of a HPSG parser. On the other hand, the existing natural language search engines can handle either word, words, phrase or questions only. The results tabulated denote the number of sites returned for the query. Since Yahoo is a directory, its database is

hierarchically organized into subject categories and Web sites are indexed under each category. Yahoo's search results return both the categories and the sites matched.

Sample searches in Tables 5.2 - 5.5, illustrate the following:

1. The total number of sites returned in Tables 5.2 and 5.3 (with NLAISE) is less than the number of sites returned in Tables 5.4 and 5.5.

2. In Tables 5.2 and 5.3, the search terms generated for different queries like "What is the currency of Canada", "What is the Canadian currency?", and "Canadian currency", remain consistent and hence the results are also consistent; whereas in tables 5.4 and 5.5 these queries return different results.

Summarizing the Tables 5.1-5.5 and other results tabulated in Appendix A, we can find that:

1. Both in keyword search engines and existing natural language search engines, different inputs conveying the same keywords result in different search results; whereas, in NLAISE results appear consistent.

2. The total number of sites listed in NLAISE is much lower in comparison to results from the existing search engines in 92% of the sample inputs .

3. Queries in NLAISE can be expressed in much simpler and easier form when compared to the existing search engines.

4. The search terms generated by NLAISE is relevant to the user query in all cases whether the query is a word, words, phrase, sentence or question.

User Input	Results in Yahoo	Results in Alta Vista	Results in Info-seek	Results in Web Crawler	Results in HotBot
Canada	115 category 8046 sites	7583480	68668	102947	126887
South Africa	26 category 1427 sites	596540	9950	122425	251576
Air Canada	8 category 316 sites	188100	1060	178350	149974
travelling in Japan	8 category 170 sites	85500	1994	128101	70111
shopping in Vancouver	12 sites	20900	757	71196	10750
air travel in U.S.A.	30 sites	4100	104	264476	38370
Give me some places of tourist attractions in Singapore	5 sites	4260	813	47535	1405
What is the exchange of Canadian currency?	2 category	7370	713	176562	5662
What is the weather forecast for Regina?	198 sites	198	24	68795	316
I want to travel Europe by rail	14 sites	21630	169	156407	16902

Table 5.1: Ease of Expression in NLAISE

User Input	Keywords generated by the system	Search results
I want to stay in Vancouver for two days.	+(hotel OR motel OR lodge), +vancouver	20820 sites returned.
visiting Tokyo	+travel, +tokyo	26660 sites returned.
I want to schedule a visit to Tokyo	+travel, +tokyo	26660 sites returned.
I want to make an online reservation for hotels in Toronto	+reservation, +hotel, +toronto	3100 sites returned.
What is the currency of Canada?	+currency, +canada	24660 sites returned.
What is the Canadian currency?	+currency, +canada	24660 sites returned.
Canadian currency	+currency, +canada	24660 sites returned.
Show me web sites on current weather condition in Alaska	current, +weather +alaska	37490 sites returned.
I want to see the road maps of Canadian cities.	+maps, +canadian +city	23830 sites returned.
Give me details about Alaska cruise	+alaska, + cruise	33400 sites returned.
I would like to know some details about shopping malls in New York.	+shopping, + mall +new, +york	9580 sites returned.

Table 5.2: Search Results in AltaVista with NLAISE

User Input	Keywords generated by the system	Search results
I want to stay in Vancouver for two days.	+(hotel OR motel OR lodge), +vancouver	655 sites returned.
visiting Tokyo	+travel, +tokyo	237 sites returned.
I want to schedule a visit to Tokyo	+travel, +tokyo	237 sites returned.
I want to make an online reservation for hotels in Toronto	+reservation, +hotel, +toronto	49 sites returned.
What is the currency of Canada?	+currency, +canada	910 sites returned.
What is the Canadian currency?	+currency, +canada	910 sites returned.
Canadian currency	+currency, +canada	910 sites returned.
Show me web sites on current weather condition in Alaska	current, +weather +alaska	641 sites returned.
I want to see the road maps of Canadian cities.	+maps, +canadian +city	95 sites returned.
Give me details about Alaska cruise	+alaska, + cruise	1285 sites returned.
I would like to know some details about shopping malls in New York.	+shopping, + mall +new, +york	45 sites returned.

Table 5.3: Search Results in Infoseek with NLAISE

User Input	Search results
accommodation, vancouver	115620 sites returned.
hotels, toronto	736230 sites returned.
visiting, tokyo	852950 sites returned.
online, reservation, hotel, toronto	424350 sites returned.
currency, canada	385200 sites returned.
canadian currency	383670 sites returned.
current weather condition, Alaska	718660 sites returned.
road maps, candian cities	1034330 sites returned.
Alaska cruise	381820 sites returned.
shopping malls, New York	954520 sites returned.

Table 5.4: Search Results in AltaVista without NLAISE

5.3 Error Handling

When a natural language system cannot understand a query, it is important that the system respond to the user by providing diagnostic messages, showing the reason for its inability to handle the query. This becomes very significant when the domain is limited. Sometimes the user may try to rephrase the query in a different syntactic manner, although the failure is due to limitation of the domain.

In our system error handling is performed at two levels. The Figures 5.5 and 5.6 are handled by the front-end JavaScript. The validation of errors in JavaScript reduces delays, as otherwise validations are to be performed by the server. Hence, data must travel from client to server, be processed, and then return to the client. Also the handling of errors in the front-end simplifies the server program.

The back-end PERL script validates the input for the following and prompts the user accordingly.

1. Lexical Error: for words that are not present in the lexicon,

User Input	Search results
" accommodation in Vancouver"	4 sites returned.
accommodation in Vancouver	4834645 sites returned.
Where to stay in Vancouver	20060160 sites returned.
Visiting Tokyo	315235 sites returned.
"hotels in toronto"	132 sites returned.
What are the hotels in Toronto that provide provide online reservation?	22461233 sites returned.
Online reservation for hotels in Toronto	11952387 sites returned.
What is the currency of Canada?	22461318 sites returned.
What is the Canadian currency?	22461308 sites returned.
Canadian currency	113517 sites returned.
What is the current weather condition in Alaska?	26435568 sites returned.
road maps, Canadian cities	3430194 sites returned.
Alaska cruise details	1922574 sites returned.
shopping malls, New York	23462205 sites returned.

Table 5.5: Search Results in Infoseek without NLAISE

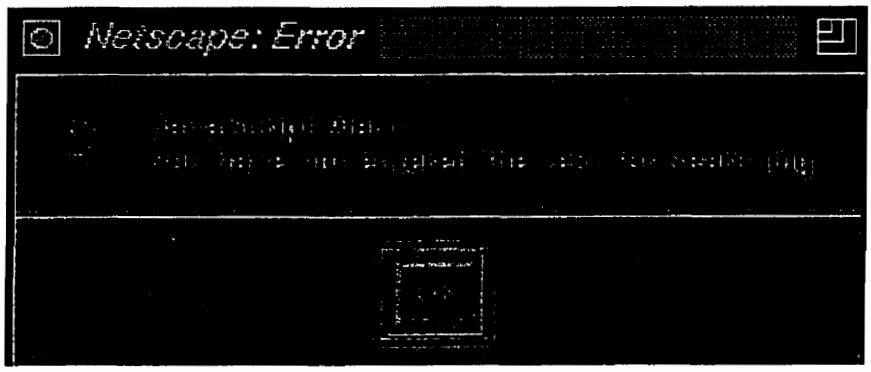


Figure 5.5: Javascript Error for "text" Field

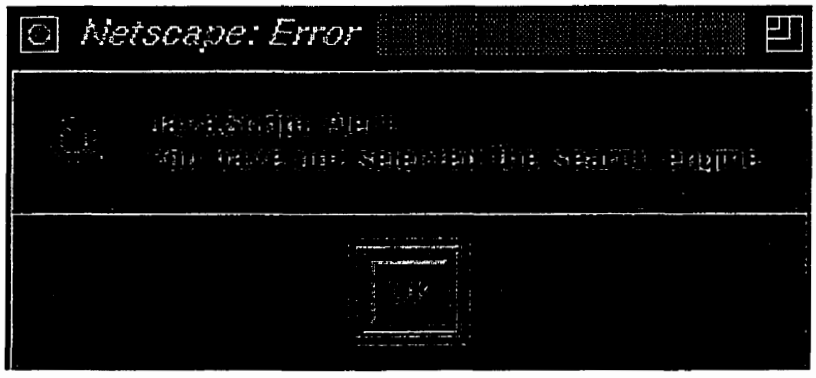


Figure 5.6: Javascript Error for "search engine" Field

2. No Keywords: for a text which does not contribute any keyword, and

3. Syntactic Error: for text that does not get parsed by the lexicon.

For syntactic errors the system will proceed using the keywords found in the text, after displaying appropriate error message to the user. This is necessary to account for grammatical errors made by the user. For example, if the user enters the query "I want visit Japan", the system will display an error message as shown in Figure 5.7 and proceed further with the keywords "travel, japan". Also our system, at the present stage, can handle only a portion of the syntax of the English language.

Conclusions drawn and future extensions of the system are discussed in Chapter 6.

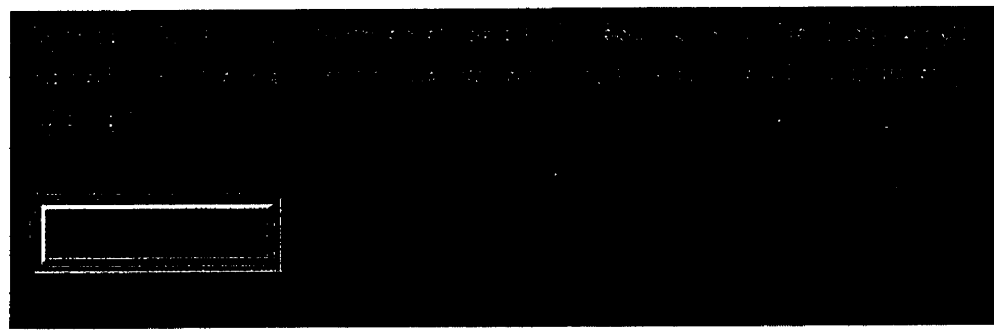


Figure 5.7: Sample Display for Syntax Error

Chapter 6

Conclusions

With the constantly increasing number of documents in the World Wide Web, and many of these documents containing natural language texts, there is wide scope for applying NL techniques in the WWW. We have presented one such application of natural language processing for information retrieval in the WWW. Some common problems encountered by the users employing the existing search tools include finding the most appropriate search term and scanning through a large number of potentially relevant documents to find the exact requirement.

In order to relieve users from finding the appropriate search terms, natural language search engines are available in WWW. However, even in these natural language search engines, different queries yielding to the same keywords, yield different results. Hence the user has to paraphrase the query in different ways to obtain the desired results.

6.1 Contributions

We have provided an application of natural language processing to improve the handling of user queries. Our system, NLAISE, enables users to present their queries to major search engines in natural language (English), without the need to study the various options provided by each search engine (such as phrase within quotes, questions only, and keywords only). Web searchers can describe the object of their search easily and meaningfully in English, knowing that the appropriate keywords

will be extracted from their description.

We have used a HPSG parser to analyze the semantics of the user query to generate appropriate keywords for searching. Since the semantic content of the words is analysed in parallel while performing the syntactic analysis, the influence of non-keywords in a search is completely eliminated, thus maintaining consistency, reduction in number of sites returned while retaining relevancy.

Various tests performed on our system demonstrate the following:

1. NLAISE provides easy and friendly access to some of the existing search engines.
2. NLAISE improves the search results by generating appropriate keywords after performing semantic analysis of the query.
3. NLAISE also demonstrates that natural language processing concepts can be positively applied for better performance of information retrieval tools in WWW.

Of course NLAISE is only an initial experiment providing positive feasibility for this type of web searching. Several enhancements should prove instructive for the future.

6.2 Future Extensions

Several extensions can be made to NLAISE. One desirable extension is to enhance its domain knowledge. We have tested this system for a limited "travel" domain. However, simple extensions to other domains are made possible because of the modularity of domain dependent and domain independent knowledge. While extending coverage over other domains, we anticipate an increase in the complexity of the intrinsic problem of natural language understanding, in particular, "handling the ambiguities". We propose to allow users to choose the appropriate keywords from alternatives to resolve ambiguity, if the parser cannot choose amongst alternatives.

Another possible extension is improving the syntactic knowledge of the parser to handle more complex and powerful queries in natural language such as handling long distance dependencies and ellipses.

As a further step to filter out irrelevant information, we also propose to have an interactive interface as a future extension. The information gathered from the user's response is to be used for further analysis of the sites returned to help choose the relevant sites from them. For example, if the user types in a text "I want to schedule a trip to Japan", the system might respond by asking further details as to whether it is a business trip or a vacation trip. This information can be used to filter out irrelevant sites from the sites listed for the search "I want to schedule a trip to Japan".

At present we have not explored the options exclusively provided by each search engine such as, searches within top 5% sites, maximum hits and searches on documents added after a specific period. Also the users cannot specify words that must be present in the search term. For example, for a query "List all web sites on current weather in Alaska", the system selects keywords as "current, weather, Alaska". Some times the user may wish to include specific words like "web, site" in the search term. In future extensions we propose to retain all the features offered by a search engine while providing a natural language access and allow the users to specify words that must be present in the search term, if any.

Appendix A

In this section, some sample search results with and without NLAISE are furnished. Figure A.1 and A.2 show the search results in Infoseek with NLAISE for the text "What is the weather forecast for Regina?". Figure A.3 and A.4 illustrate the search results in Infoseek for the text "What is the weather forecast for Regina?" without NLAISE. Figure A.5 and A.6 present the search results in AltaVista for the query "I want to see the road maps of Canadian cities" with NLAISE. The search results for "road maps, Canadian cities" in AltaVista without NLAISE are presented in Figures A.7 and A.8.

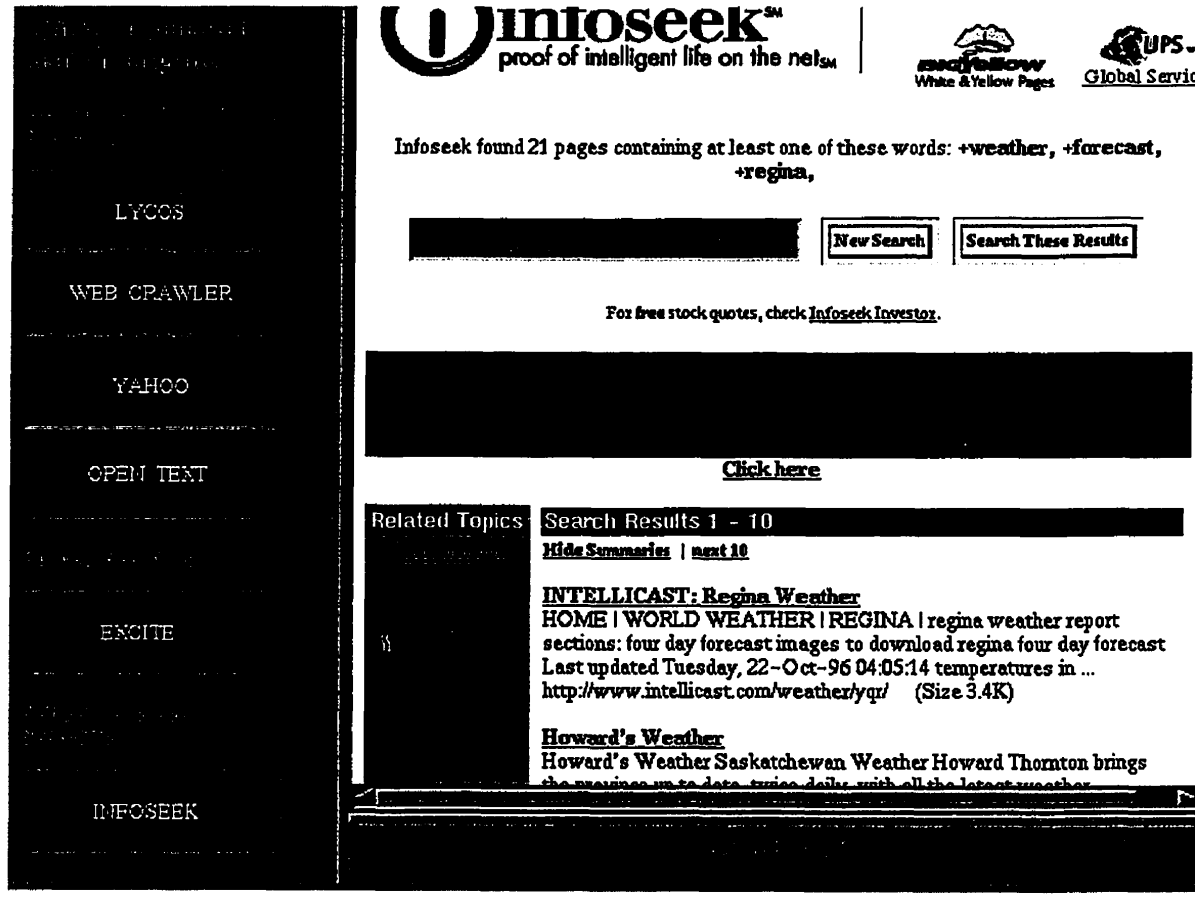


Figure A.1: Sample Result in Infoseek with NLAISE

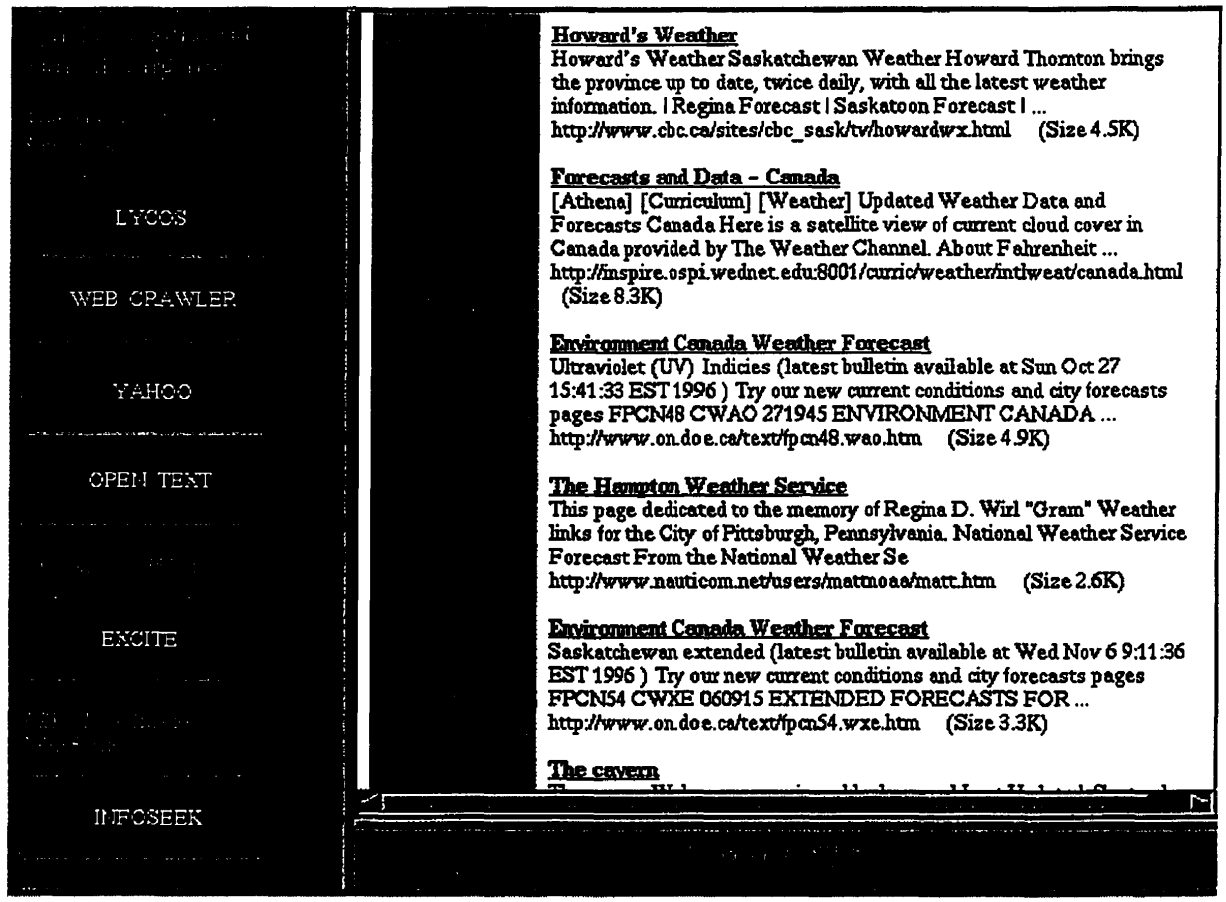


Figure A.2: Sample Result in Infoseek with NLAISE

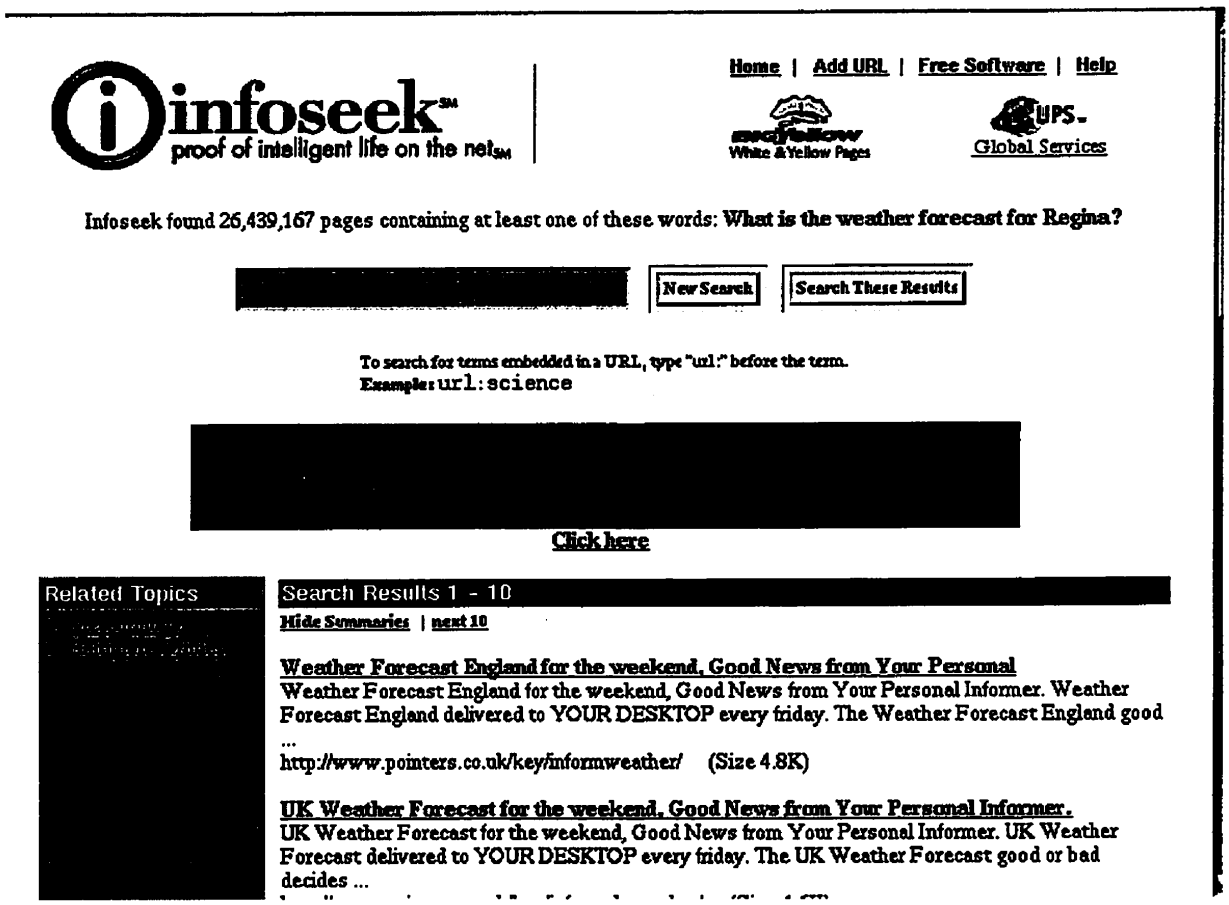


Figure A.3: Sample Result in Infoseek without NLAISE

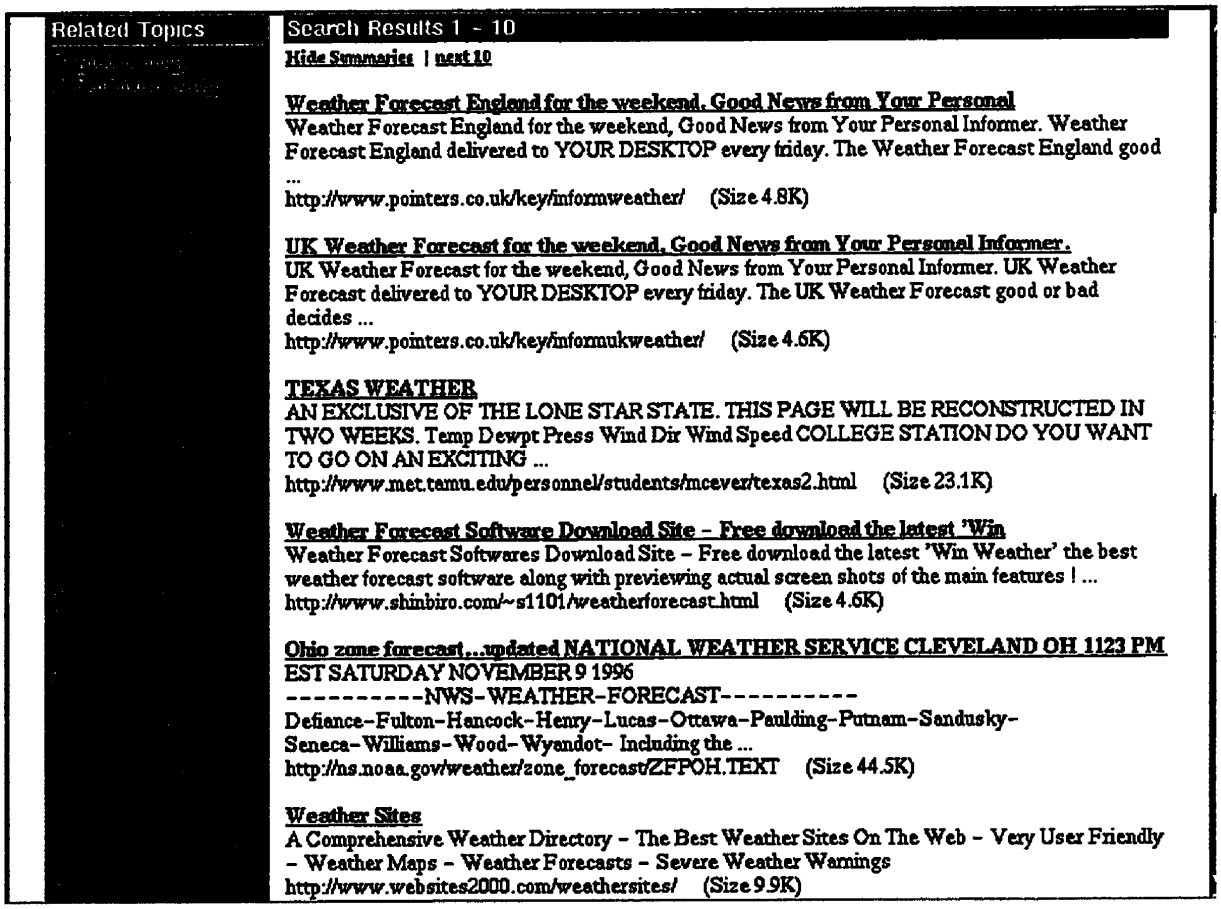


Figure A.4: Sample Result in Infoseek without NLAISE

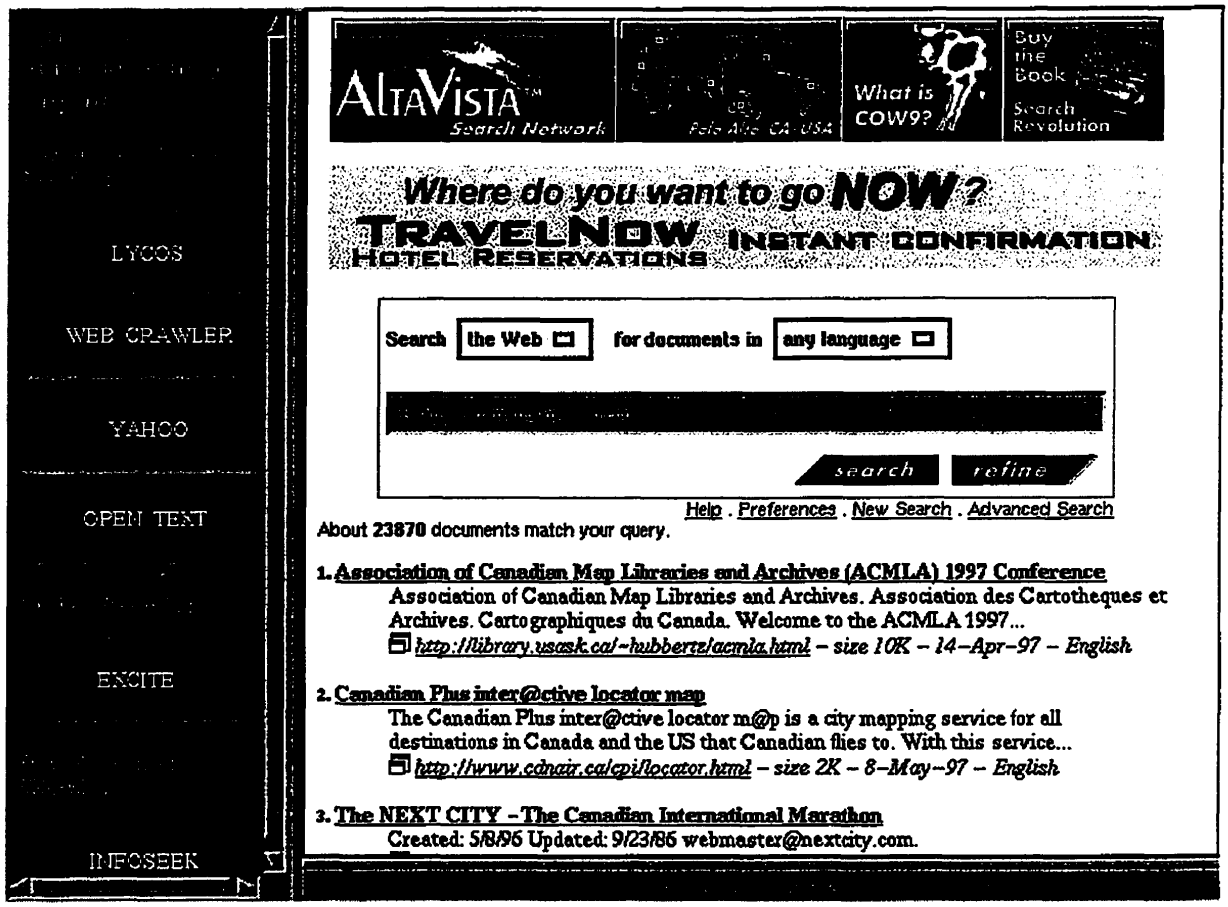


Figure A.5: Sample Result in AltaVista with NLAISE

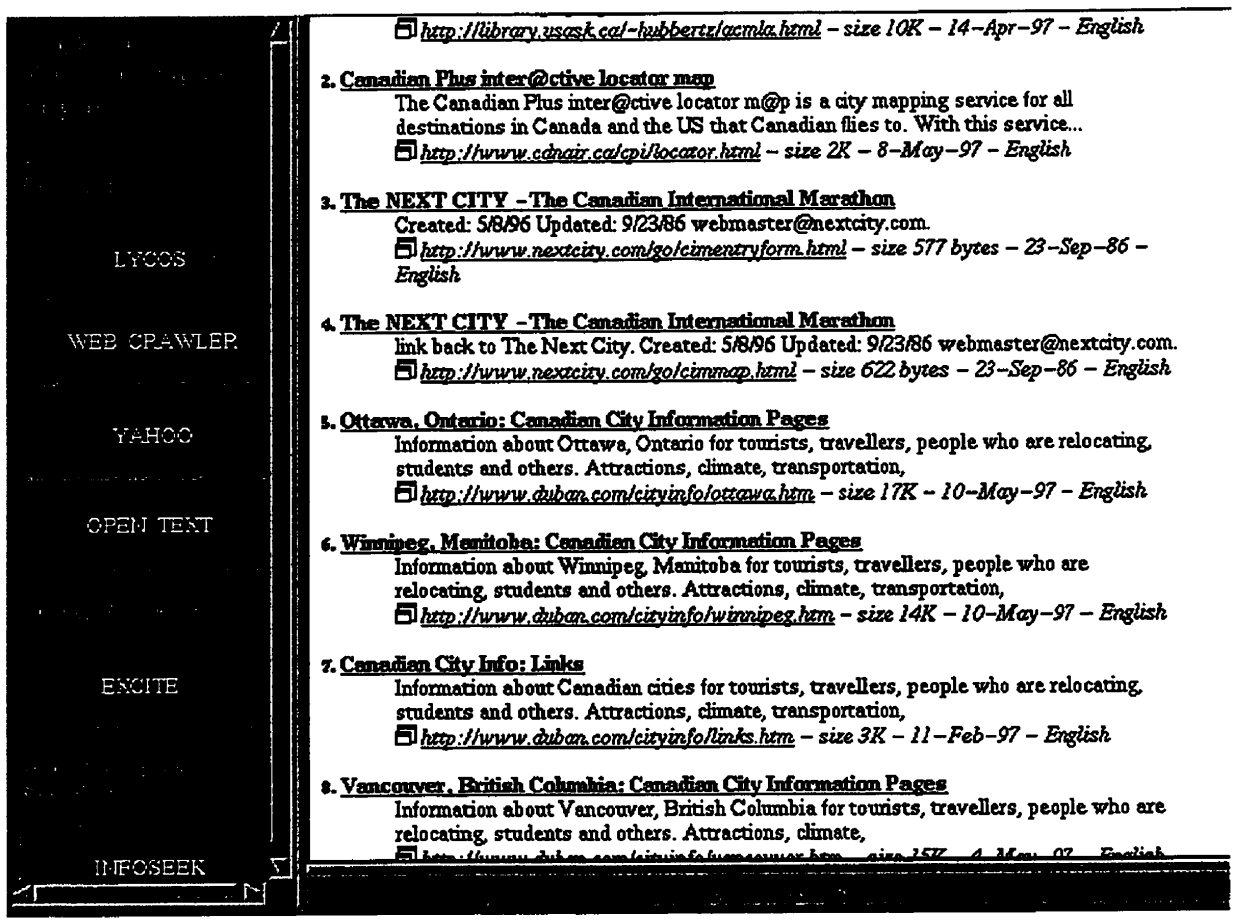
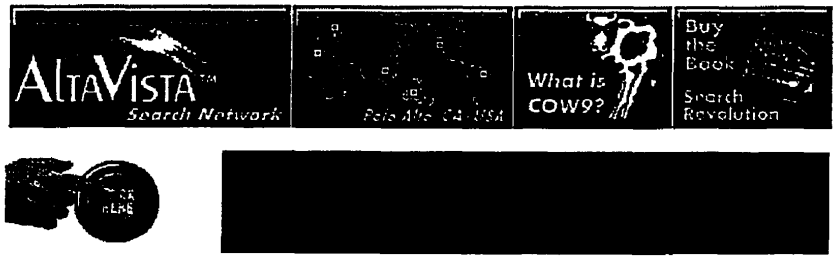
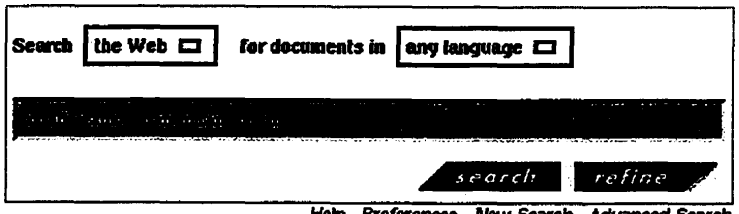


Figure A.6: Sample Result in AltaVista with NLAISE





Help . Preferences . New Search . Advanced Search

About **1034940** documents match your query.

1. **Four Canadian Cities**
   Four Canadian cities rated in top ten in survey of locational advantages. Corporate Resources Group (CRG) of Geneva regularly surveys major world cities...
   http://www.dfait-maeci.gc.ca/english/news/newsletr/invest/eSurv.htm – size 3K – 29-Jan-97 – English

2. **Travel - Canadian Cities**
   CANADIAN CITIES. HOME Agents Airlines Canadian Information Destinations Hotels ...
   http://www.front.net/home/websites/travel/cdn_cities.html – size 9K – 19-Aug-96 – English

3. **Canadian Cities - Sarnia**
   nbsp; A few clouds. 9 °C. 66% 2 °C. 102.7 kPa. 24 km/h E. 3 °C. Mainly clear skies. 12 °C. ...
   http://www.theweathernetwork.com/city/can/data/sarnia.html – size 9K – 22-May-97 – English

Figure A.7: Sample Result in Alta Vista without NLAISE

regularly surveys major world cities...
http://www.dfait-maeci.gc.ca/english/news/newsletr/invest/eSurv.htm – size 3K – 29-Jan-97 – English

2. **Travel - Canadian Cities**
CANADIAN CITIES. HOME Agents Airlines Canadian Information Destinations Hotels ...
http://www.fronz.net/home/websites/travel/cdn_cities.html – size 9K – 19-Aug-96 – English

3. **Canadian Cities - Sarnia**
nbsp; A few clouds. 9 °C. 66% 2 °C. 102.7 kPa. 24 km/h E. 3 °C. Mainly clear skies. 12 °C. ...
http://www.theweathernetwork.com/city/can/data/sarnia.html – size 9K – 22-May-97 – English

4. **Canadian Cities - Gimli**
nbsp; 2 °C. 75% -9 °C. N/A. 33 km/h N. -2 °C. Mainly clear skies. 15 °C. 2 °C....
http://www.theweathernetwork.com/city/can/data/gimli.html – size 9K – 23-May-97 – English

5. **Canadian Cities - Montreal**
nbsp; A few clouds. 14 °C. 44% 10 °C. 102.0 kPa. 17 km/h NW. 2 °C. Clear. 16 °C. ...
http://www.theweathernetwork.com/city/can/data/montreal.html – size 9K – 23-May-97 – English

6. **WHO Collaborating Centre...Healthy Cities: Canadian Groups**
WHO Collaborating Centre for Research on Healthy Cities. Healthy Cities Related References in Canada: Healthy Cities Homepages. Toronto Healthy City...
http://www.rulimburg.nl/~who-city/canref.html – size 729 bytes – 28-Oct-96 – English

7. **Canadian Cities - Charlottetown**
nbsp; Partly cloudy. 5 °C. 70% -2 °C. 101.2 kPa. 22 km/h N. 0 °C. Cloudy with clear breaks. 12 °C....
http://www.theweathernetwork.com/city/can/data/charlottetown.html – size 9K – 23-May-97 – English

8. **Canadian Cities - Prince George**
nbsp; Partly cloudy. 16 °C. 42% 16 °C. 101.2 kPa. 0 km/h. 3 °C. Mainly sunny. 19 °C. ...
http://www.theweathernetwork.com/city/can/data/prince_george.html – size 9K – 24-May-97 – English

9. **Canadian Cities - St. Anthony**
nbsp; Fog. 3 °C. 100% 1 °C. 100.9 kPa. 11 km/h N. 3 °C. Mainly cloudy. 5 °C. 0 °C...
http://www.theweathernetwork.com/city/can/data/st_anthony.html – size 9K – 24-May-97 – English

10. **Weather Data - Canadian Cities**

Figure A.8: Sample Result in Alta Vista without NLAISE

User Input	Results in Yahoo	Results in Alta Vista	Results in Info-seek	Results in Web Crawler	Results in HotBot
U.S.A	37 categories 1170 sites	241766	67451	123259	1487128
New York	154 categories 4169 sites	4735220	9806	88881	1309440
Canadian Airlines	2 categories 25 sites	26710	1483	52374	548446
auto rentals in Canada	4 categories 53 sites	3560	35	150215	2298
booking flight tickets	5100 sites (AV)	5100	170	128090	7116
Give me a list of restaurants in New York	35 categories 283 sites	31640	161	115169	47174
I like to see the travel guidebook for China	18 categories 337 sites	8410	1995	117734	787
List all hotels and motels in Tokyo.	6 sites	293	562	60373	56374
I want to schedule a business trip to New York	20 categories 268 sites	20020	164	435586	68606
List all travel agents in Vancouver	5 sites	3020	98	139301	3417
What is the airfare from Toronto to Vancouver	8570 sites (AV)	8570	198	12246	638
Which hotels in U.S.A have online resevation?	6 sites	2910	324	172685	8363
I would like to see the sites on boating in U.S.A.	24 sites	3160	630	130182	9275
What is the currency of China	1 site	10530	1279	32681	16483

Table A.1: More Samples with NLAISE

User Input	Results in Yahoo	Results in Web crawler	Results in Hot Bot
I want to stay in Toronto for two days.	12 sites	79786	564334
visiting Tokyo	1category 22 sites	106421	23133
I want to schedule a visit to Tokyo	1 category 22 sites	106421	23133
I want to make an online reservation for hotels in Toronto	3100(AV)	77331	2067
What is the currency of Canada?	2 categories 14 sites	111580	31597
What is the Canadian currency?	2 categories 14 sites	111580	31597
Show me web sites on current weather condition in Alaska	1 category 17 sites	213925	11971
I want to see the road maps of canadian cities.	1 category 46 sites	264325	17823
Give me details about Alaska cruise	1 category 71 sites	28912	9325
I would like to know some details about shopping malls in New York.	1 category 9 sites	780333	13699

Table A.2: Search Results in Yahoo, WebCrawler and HotBot with NLAISE

User Input	Results in in Yahoo	Results in HotBot	Results in Web Crawler
U.S.A	37categories 1181 sites	No results 1487128 for USA	123259
Air Canada	8 categories 316 sites	150037	178350
New York	154 categories 4185 sites	1309942	88881
accomodation, Toronto	16 sites	1151	25569
shopping, New York	1 category	57778	139998
air travel, U.S.A.	30 sites	38370	264476
weather forecast, Regina	150590(AV)	316	68795
staying, toronto	99660(AV)	5623	29590
visiting, Tokyo	857520(AV)	10324	61722
online reservation, hotels, Toronto	8 sites	635	264617
currency, Canada	2 categories 14 sites	31597	111580
Canadian, currency	2 categories 18 sites	17524	52169
current weather condition, Alaska	6 sites	2078	231440
road maps, canadian cities	1 site	2631	199802
Alaska cruise	1 category	9325	108215
shopping mall, New York	1 category 9 sites	45	141599

Table A.3: Search Results in Yahoo and HotBot and Web Crawler without NLAISE

Appendix B

Program Listings

```
% ******************************************************
%       This file contains the program code for the
%       HPSG parser developed for the travel domain.
%  Last modified on August 14th, 1997
% System:   ALE 2.0.1 under Sicstus 2.1 #9
% ******************************************************

% ******************************************************
% Type Declaration
% ******************************************************

% The hierarchy under bot
% Bot has three top-level subtypes
bot sub [system, syntax, semantics].

% basic data types
system sub [bool, list].

% syntax: subset of english grammar
syntax sub [sign, synsem_or_none, loc, cat, head, case, marking,
agr, pers, num, pform, vform].

% semantics: domain specific
semantics sub [cont, keyword, db_action, dconcept ].


%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Basic data types:
%       boolean and list.
%%%%%%%%%%%%%%%%%%%%%%%%%%%

% boolean
```

```
bool sub [minus, plus].
  minus sub □.
  plus  sub □.

% list
list sub [e_list, ne_list, synsem_list, sign_list, key_list].
  e_list  sub □.
  ne_list sub [ne_synsem_list, ne_sign_list, ne_key_list]
intro [hd:bot,
       tl:list].

  synsem_list sub [e_list, ne_synsem_list].
       ne_synsem_list sub □
intro [ hd: synsem,
        tl: synsem_list].
  sign_list sub [e_list, ne_sign_list].
        ne_sign_list sub □
     intro [ hd: sign,
tl: sign_list].

   key_list sub [e_list, ne_key_list].
  ne_key_list sub □
intro [ hd: keyword,
        tl: key_list].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Syntactic Hierarchy:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sign sub [word, phrase]
intro [synsem:synsem].
  word   sub □.
  phrase sub □.

synsem_or_none sub [synsem, none].
  synsem sub □
     intro [loc:loc].
  none sub  □.

loc sub □
        intro [cat: cat,
               cont: cont].


% The CAT feature
cat sub □
        intro [head:   head,
               subcat: synsem_list,
```

```
        spr: synsem_list,
         % used for specifier selection
        marking: marking].


head sub [funct, subst, dummy, how].
  funct sub [det, mark]
        intro [spec: synsem_or_none].
    det  sub [].
    mark sub [].
  subst sub [adj, noun, verb, prep]
        intro [mod: synsem_or_none].
    adj  sub [].
    noun sub []
        intro [case: case,
               agr: agr].
    verb sub []
        intro [aux:   bool,
               inv:   bool,
               vform: vform].
    prep sub []
        intro [pform: pform].
  dummy sub [].
  how sub [].

marking sub [marked, unmarked].
  marked sub [comp, conj].
comp sub [for, that].
  %%% for sub [].
  that sub [].
conj sub [and, or].
  and sub []. or sub [].
  unmarked sub [].

case sub [nom, acc, dat].
  nom   sub [].
  acc sub [].
  dat sub [].

agr sub []
        intro [person: pers,
               num: num].

pers sub [first, second, third].
  first sub [].          second sub [].  third sub [].

num sub [singular, plural].
  singular sub [].       plural sub [].
```

```
pform sub [after, at, between, by, during, for, from,
                in, of, on, over, to, with, under, about, near].
  after sub [].          from sub [].     with sub [].
  at sub [].             in sub [].       between sub [].
  of sub [].             by sub [].       on sub [].
  during sub [].         over sub [].     for sub [].
  to sub [].             under sub [].    about sub [].
  near sub [].

vform sub [fin, bse, ger, inf, pas, prt, psp].
  fin sub [].            % finite, e.g., speaks, spoke
  bse sub [].            % base, e.g., speak, be
  ger sub [].            % gerundive, e.g., speaking
  inf sub [].            % infinitive, e.g., to see
  pas sub [].            % passive, e.g., spoken
  prt sub [].            % present participle, e.g., speaking
  psp sub [].            % past participle, e.g., spoken

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Semantic Hierarchy:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% The CONT feature
cont sub []
        intro [
                sem_mark: dconcept,
keyword: key_list].


keyword sub [ykey, nkey, dconcept].
ykey sub [specific_con].
nkey sub [gencon].

% used for selectional restrictions
% especially in nominal compounds

dconcept sub [gencon, specific_con].
  gencon sub [].
  specific_con sub [places, directions, transportation, weather,
  food, accomodation, economics, business].

  places sub [asia, africa, america, canada, europe, midwest,
  toronto, vancouver, tokyo, usa, new, york, alaska, regina,
  publications, city, country, region, asian, african, canadian,
  british, attractions].
  asia sub [india, bhutan, china, japan, bali, singapore, korea,
  nepal, thailand, taiwan, tibet].
```

```
india sub [].       china sub [].
bhutan sub [].      japan sub [].
bali sub [].        singapore sub [].
korea sub [].       nepal sub [].
thailand sub [].    taiwan sub [].
tibet sub [].       usa sub [].
tokyo sub [].       toronto sub [].
vancouver sub [].   alaska sub [].
regina sub [].

      africa sub [].    europe sub [].
      america sub [].   midwest sub [].
      canada sub [].    new sub [].
      york sub [].

      publications sub [guide, guidebook, magazines].

      transportation sub [air, flight, train, rail, fare, rate,
      ticket, auto, rentals, boat, boating, cruise, booking, airport,
      airlines, airways, road, map, fly, information, reservation,
      agency, agent, international, tourist, travel].

      directions sub [north, south, east, west, central].
      food sub [ restaurants].
      accomodation sub [hotel, motel, lodging, cheap, economical].
      weather sub [current, forecast].
      economics sub [currency, exchange].
      business sub [shopping, malls, gift].


      flight sub [].          air sub [].
      train sub [].           fare sub [].
      auto sub [].              information sub [].
      restaurants sub [].     guidebook sub [].
      rate sub [].            rail sub [].
      region sub [].          magazines sub [].
      guide sub [].           ticket sub [].
      boat sub [].            boating sub [].
      shopping sub [].        gift sub [].
      malls sub [].           booking sub [].
      asian sub [].           african sub [].
      canadian sub [].        british sub [].
      north sub [].  east sub [].
      south sub [].             west sub [].
      current sub [].         currency sub [].
      agent sub [].           hotel sub [].
      motel sub [].           travel sub [].
      lodging sub [].         city sub [].
```

```
      country sub [].          airport sub [].
      airways sub [].          airlines sub [].
      road sub [].             map sub [].
      rentals sub [].          reservation sub [].
      international sub [].    exchange sub [].
      agency sub [].           fly sub [].
      economical sub [].  central sub [].
      cheap sub [].               cruise sub [].
      tourist sub [].             attractions sub [].
      forecast sub [].

% ******************************************************
% Basic syntactic macros: for accessing feature values.
%               type synsem
% ******************************************************

%%% head
head_s(X)       macro (loc:cat:head:X).

%%% subcat
subcat_s(X)     macro (loc:cat:subcat:X).

%%% spr
spr_s(X) macro (loc:cat:spr:X).

%%% specifier
spec_s(X) macro @head_s(spec: X).

%%% verb
vform_s(X)      macro @head_s(vform:X).

%%% aux
aux_s(X)        macro @head_s(aux:X).

%%% inv
inv_s(X)        macro @head_s(inv:X).

%%% noun
case_s(X)       macro @head_s(case:X).
agr_num_s(X)    macro @head_s(agr:num:X).
agr_pers_s(X)   macro @head_s(agr:num:X).


%%% preposition
pform_s(X)      macro @head_s(pform:X).

%%% mod
mod_s(X)        macro @head_s(mod: X).
```

```
%%% marking
marking_s(X) macro (loc:cat:marking:X).

% ************************************************
% Basic syntactic macros: access feature values.
%                type sign
% ************************************************

%%% head
head(X)          macro (synsem: @head_s(X)).

%%% subcat
subcat(X)        macro (synsem: @subcat_s(X)).

%%% spr
spr(X) macro (synsem: @spr_s(X)).

%%% specifier
spec(X) macro (synsem: @spec_s(X)).

%%% verb
vform(X)         macro (synsem: @vform_s(X)).

%%% aux
aux(X)           macro (synsem: @aux_s(X)).

%%% inv
inv(X)           macro (synsem: @inv_s(X)).

%%% noun
case(X)          macro (synsem: @case_s(X)).
agr_num(X)       macro (synsem: @agr_num_s(X)).
agr_pers(X)      macro (synsem: @agr_pers_s(X)).

%%% preposition
pform(X)         macro (synsem: @pform_s(X)).

%%% mod
mod(X)           macro (synsem: @mod_s(X)).

%%% marking
marking(X) macro (synsem: @marking_s(X)).


% ************************************************************
% Macros use in subcategorization lists and modifier features.
%                type synsem
```

```
% ************************************************************

%%% adjective
adjective_s             macro ( @head_s(adj),
                                @subcat_s([]),
@spr_s(e_list),
                                @mod_s(@head_s(noun)) ).

%%% determiner
determiner_s            macro ( @head_s(det),
       /*
@spec_s((@head_s(noun),
 @subcat_s(ne_synsem_list))),
                                              */
@spec_s( @head_s(noun) ),
        @spr_s( e_list ),
                                @subcat_s([]) ).
possesiveDet_s          macro ( @head_s(det),
                                @subcat_s([@np_s]) ).

%%% noun
common_s                macro ( @head_s(noun),
@spr_s([@determiner_s]) ).
                                %%% @subcat_s([@determiner_s]) ).
np_s                    macro ( @head_s(noun),
                                @subcat_s([]) ).
npObj_s                 macro ( @np_s,
                                @case_s(acc) ).
npSubj_s                macro ( @np_s,
                                @case_s(nom) ).

%%% preposition
prep_s(PForm)           macro ( @head_s(prep),
@spr_s( e_list ),
                                @pform_s(PForm) ).
/*
preposition_s(PForm)    macro ( @prep_s(PForm),
% @mod( none ),
                                @mod_s((@head_s(noun), @subcat_s([]))),
                                @subcat_s([@head_s(noun), @head_s(noun)]) ).
*/

% Preposition subcats for one NP and a 'dummy' head
% and it modifies a noun or a VP
preposition_s(PForm) macro ( @prep_s(PForm),
( @mod_s(@head_s(noun));
          @mod_s(@head_s(verb))),
@subcat_s([@head_s(dummy), @head_s(noun)]) ).
```

```
%%% verb
verb_s(VForm)           macro ( @head_s(verb),
                                @vform_s(VForm) ).
xcomp_s(VForm)          macro ( @verb_s(VForm),
                                @aux_s(minus),
                                @inv_s(minus),
                                @subcat_s([@np_s]) ).

%%% conjunct
% temporary fix for conjunction and general coordination
%
% see note and lex re: coordination in coord.txt
% and coord.ale

conj_s(ConjForm) macro ( @head_s(mark),
% @spec_s(none),
@spr_s(e_list),
@subcat_s(e_list),
@marking_s(ConjForm) ).


%***********************************************
% type: sign

%%% adjective
adjective               macro ( @head(adj),
                                @subcat(□),
@spr(e_list),
@marking( unmarked ),
                                @mod(@common_s) ).

%%% determiner
determiner              macro ( @head(det),
@spec( @head_s(noun) ),
@spr(e_list),
@marking( unmarked ),
                                @subcat(□) ).

possesiveDet            macro ( @head(det),
% @spec( none ),
@spec( @head_s(noun) ),
@spr(e_list),
@marking( unmarked ),
                                @subcat([@head_s(noun)]) ).

%%% noun
```

```
common                  macro ( @head(noun),
% @spr([@determiner_s]),
@spr( [@head_s(det)] ),
@mod(@head_s(noun)),
@marking( unmarked ),
@subcat(e_list) ).
                                % @subcat([@determiner_s]) ).
np                      macro ( @head(noun),
@spr(e_list),
@marking( unmarked ),
                                @subcat([]) ).
npNonMod                macro ( @np,
                                @mod(none) ).
npSubj                  macro ( @npNonMod,
                                @case(nom) ).
npObj                   macro ( @npNonMod,
                                @case(acc) ).


%%% preposition
prep(PForm)             macro ( @head(prep),
@spr(e_list),
                                @pform(PForm) ).
/*
preposition(PForm)      macro ( @prep(PForm),
% @mod( none ),
                                @mod((@head_s(noun), @subcat_s([]))),
                                @subcat([@head_s(noun), @head_s(noun)]) ).
*/

% Preposition subcats for one NP and one 'dummy' head
% This avoid having to add an additional phrase rule
% to capture PP -> P, NP.
% Preposition modifies a noun or a VP


preposition(PForm) macro ( @prep(PForm),
@marking( Marking ),
( @mod(@head_s(noun));
          @mod(@head_s(verb))),
@subcat([@head_s(dummy),
 (@head_s(noun), @marking_s(Marking))
]) ).


preposition( PForm, Mark )
macro ( @prep( PForm ),
( @mod( (@head_s(noun),
```

```
 @smarker_s( Mark ) )
)
),
@subcat([@head_s(dummy), @head_s(noun)]) ).

%%% verb

verb(VForm)              macro ( @head(verb),
@marking( unmarked ),
                                 @vform(VForm) ).
vp                       macro ( @head(verb),
                                 @vform(bse),
                                 @aux(minus),
@marking( unmarked ),
                                 @mod(none) ).


xcomp(VForm)             macro ( @verb(VForm),
                                 @aux(minus),
                                 @inv(minus) ).


% auxiliary
auxil(CompForm)          macro ( @head(verb),
                                 @vform(fin),
                                 @aux(plus),
@marking( unmarked ),
                                 @subcat( [ @npSubj_s,
                                           (@head_s(verb), @vform_s(CompForm)
                                              @subcat_s(ne_synsem_list)
                                            )
                                           ] ) ).


%%% Verbs subcat for (saturated) noun phrase
%%%
% ditransitive
% e.g., "John gives Sandy the book"
ditrans                  macro ( @vp,
                                 @subcat( [ @np_s,
                                            @npObj_s,
                                           (@head_s(noun), @case_s(acc))
                                           ] ) ).

% control verbs
icontrolv( VForm, CForm)
                         macro ( @verb(VForm),
                                 @mod(none),
```

```
                                @subcat( [ @np_s, @xcomp_s(CForm) ] ) ).

tcontrolv( VForm, CForm)
                        macro ( @verb(VForm),
                                @mod(none),
                                @subcat( [ @npSubj_s,
                                           @npObj_s,
                                           @xcomp_s(CForm)
                                         ] ) ).

% intransitive
intrans                 macro ( @vp,
                                @subcat([@np_s]) ).


intransPP(PForm)        macro ( @vp,
                                @subcat( [ @npSubj_s, @prep_s(PForm) ] ) ).

% transitive
trans                   macro ( @vp,
                                @subcat( [ @np_s, @npObj_s ] ) ).


%%% conjunct

conj(ConjForm) macro ( @head(mark),
% @spec(none),
@spr(e_list),
@subcat(e_list),
@marking(ConjForm) ).


% ****************************************************************
%  Macros for use in lexical entries
% ****************************************************************

%%%%%%%%%%%%%%%%%%
%%% adjective

adjective_lex           macro (word, @adjective).


%%%%%%%%%%%%%%%%%%
%%% determiner
```

```
determiner_lex          macro (word, @determiner).
possesiveDet_lex        macro (word, @possesiveDet).

%%%%%%%%%%%%%%%%%%
%%% noun

cn_lex                  macro (word, @common).
proper_lex              macro (word, @npNonMod),
      @keyword( gencon ).
ppronSubj_lex           macro (word, @npSubj),
      @keyword( gencon ).
ppronObj_lex            macro (word, @npObj),
      @keyword( gencon ).

%%%%%%%%%%%%%%%%%%
%%% preposition

preposition_lex(PForm)  macro (word, @preposition(PForm)),
@keyword( gencon ).
preposition_lex(PForm, Mark)
macro (word, @preposition( PForm, Mark )),
@keyword( gencon ).

for_lex macro (word, @for_prep).

%%%%%%%%%%%%%%%%%%
%%% verb

aux_lex(CompForm)       macro (word, @auxil(CompForm)).
ditrans_lex             macro (word, @ditrans).
icontrolv_lex(VForm, Form)
                        macro (word, @icontrolv(VForm, Form)).
intrans_lex             macro (word, @intrans).
intransg_lex            macro (word, @intransger).
intransPP_lex(PForm)    macro (word, @intransPP(PForm)).
tcontrolv_lex(VForm, Form)
                        macro (word, @tcontrolv(VForm, Form)).
trans_lex               macro (word, @trans).
ditrans_lex             macro (word, @ditrans).

%%%%%%%%%%%%%%%%%%
%%% conjunct
% temporary

conj_lex( ConjForm ) macro (word, @conj( ConjForm )).


and_coord_lex macro (@conj_lex( and )).
```

```
or_coord_lex macro (@conj_lex( or )).


% ************************************************************
%  Semantic Macros: Domain Independent
% ************************************************************

% *************
% type synsem
% *************
cont_s( Cont )          macro (loc:cont:Cont).
action_s( Act )         macro @cont_s(action: Act).
smarker_s(Marker)       macro @cont_s(sem_mark:Marker).
keyword_s( Keys )       macro @cont_s(keyword:(hd:Keys)).

% *************
% type sign
% *************
cont(Cont)              macro (synsem:loc:cont:Cont).
action(Act)             macro @cont(action:Act).
smarker(Marker)         macro @cont(sem_mark:Marker).
keyword( Keys )         macro @cont(keyword:(ne_key_list, (hd:Keys,
   tl:e_list))).
subkeyword( Keys )      macro @cont(keyword:(tl:(ne_key_list, hd:Keys,
      tl:key_list))).


% The following macro definitions for nouns and adjectives
% are used when semantics is taken into consideration.

%%% We also classify nouns into  classes
%%% 1. np takes complement and acts as a modifier of other nouns.
%%% 2. np takes complement and not a modifier.
%%% 3. np takes no complement and acts as a modifier.
%%% 4. np takes no complement and not a modifier.

npCompMod( Comp, Mod ) macro ( @head( noun ),
% @spr([@determiner_s]),
@spr( [@head_s(det)] ),
@mod( ( @head_s(noun),
@smarker_s(Mod) )
    ),
@subcat( [ ( @head_s(noun),
     @smarker_s( Comp ) ) ]
)
).

npCompNoMod( Comp ) macro ( @head( noun ),
```

```
% @spr([@determiner_s]),
@spr( [@head_s(det)] ),
@mod( none ),
@subcat( [ ( @head_s(noun),
     @smarker_s( Comp ) ) ]
)
).

npNoCompMod( Mod ) macro ( @head( noun ),
% @spr([@determiner_s]),
           @spr( [@head_s(det)] ),
@mod( ( @head_s(noun),
@smarker_s(Mod) )
    ),
@subcat( e_list ) ).

npNoCompNoMod macro ( @head( noun ),
% @spr([@determiner_s]),
@spr( [@head_s(det)] ),
@mod( none ),
@subcat( e_list ) ).


npMod( Mod ) macro ( @np,
@mod( ( @head_s(noun),
@smarker_s(Mod) ) )
).

% adjective that modifies an object having a specific concept

adj(Mark)              macro ( @adjective,
                               @mod( @smarker_s(Mark) ) ).

% Macros to be used in lexical entries.

%%%% nouns
npCompMod_lex( Comp, Mod )
macro (word, @npCompMod( Comp, Mod )).
npCompNoMod_lex( Comp ) macro (word, @npCompNoMod( Comp )).
npNoCompMod_lex( Mod ) macro (word, @npNoCompMod( Mod )).
npNoCompNoMod_lex macro (word, @npNoCompNoMod).
npMod_lex( Mod ) macro (word, @npMod( Mod )).

% adjective
adj_lex( Marker ) macro (word, @adj( Marker )).


% ******************************************************
```

```
%       This file contains lexical entries
% ******************************************************


%%%%%%%%%%%%%%%%
%%% determiner

a                           ---> @determiner_lex.
an                          ---> @determiner_lex.
all                         ---> @determiner_lex.
any                         ---> @determiner_lex.
each                        ---> @determiner_lex.
every                       ---> @determiner_lex.
some                        ---> @determiner_lex.
the                         ---> @determiner_lex.
this                        ---> @determiner_lex.
that                        ---> @determiner_lex.
is                          ---> @determiner_lex.
are                         ---> @determiner_lex.
am                          ---> @determiner_lex.

% possessive determiner
s                           ---> @possesiveDet_lex.

%%%%%%%%%%%%%%%%
%%% noun
asia                        --->    @npNoCompNoMod_lex,
                                    @keyword( asia ).

africa                      --->    @npNoCompNoMod_lex,
                                    @keyword( africa ).

america                     --->    @npNoCompNoMod_lex,
                                    @keyword( america ).

europe                      --->    @npNoCompNoMod_lex,
                                    @keyword( europe ).

midwest                     --->    @npNoCompNoMod_lex,
                                    @keyword( midwest ).

india                       --->    @npNoCompNoMod_lex,
                                    @keyword( india ).

bhutan                      --->    @npNoCompNoMod_lex,
                                    @keyword( bhutan ).

bali                        --->    @npNoCompNoMod_lex,
```

```
                                    @keyword( bali ).

china                    --->       @npNoCompNoMod_lex,
                                    @keyword( china ).

japan                    --->       @npNoCompNoMod_lex,
                                    @keyword( japan ).

singapore                --->       @npNoCompNoMod_lex,
                                    @keyword( singapore ).

korea                    --->       @npNoCompNoMod_lex,
                                    @keyword( korea ).

nepal                    --->       @npNoCompNoMod_lex,
                                    @keyword( nepal ).

air                      --->       @npNoCompNoMod_lex,
                                    @keyword( air ).

canada                   --->       @npNoCompNoMod_lex,
                                    @keyword( canada ).

toronto                  --->       @npNoCompNoMod_lex,
                                    @keyword( toronto ).

vancouver                --->       @npNoCompNoMod_lex,
                                    @keyword( toronto ).

tokyo                    --->       @npNoCompNoMod_lex,
                                    @keyword( tokyo ).

usa                      --->       @npNoCompNoMod_lex,
                                    @keyword( usa ).

fare                     --->       @npNoCompNoMod_lex,
                                    @keyword( fare ).

rate                     --->       @npNoCompNoMod_lex,
                                    @keyword( rate ).

region                   --->       @npNoCompNoMod_lex,
                                    @keyword( region ).

travel                   --->       @npNoCompNoMod_lex,
                                    @keyword( travel ).

trip                     --->       @npNoCompNoMod_lex,
```

```
                                    @keyword( travel ).

weather                 --->        @npNoCompNoMod_lex,
                                    @keyword( weather ).

new                     --->        @npNoCompNoMod_lex,
                                    @keyword( new ).

currency                --->        @npNoCompNoMod_lex,
                                    @keyword( currency ).

condition               --->        @npCompNoMod_lex( weather ),
                                    @keyword( gencon ).

york                    --->        @npCompNoMod_lex( new ),
                                    @keyword( york ).

forecast                --->        @npCompNoMod_lex( weather ),
                                    @keyword( forecast ).

site                    --->        @npNoCompNoMod_lex,
                                    @keyword( gencon ).

hotel                   --->        @npNoCompNoMod_lex,
                                    @keyword( hotel ).

motel                   --->        @npNoCompNoMod_lex,
                                    @keyword( motel ).

lodge                   --->        @npNoCompNoMod_lex,
                                    @keyword( lodging ).

lodging                 --->        @npNoCompNoMod_lex,
                                    @keyword( lodging ).

accomodation            --->        @npNoCompNoMod_lex,
                                    @keyword( accomodation ).

city                    --->        @npNoCompNoMod_lex,
                                    @keyword( city ).

country                 --->        @npNoCompNoMod_lex,
                                    @keyword( country ).

magazine                --->        @npNoCompNoMod_lex,
                                    @keyword( magazines ).

map                     --->        @npNoCompNoMod_lex,
```

```
                                    @keyword( map ).

information             --->        @npNoCompNoMod_lex,
                                    @keyword( information ).

reservation             --->        @npNoCompNoMod_lex,
                                    @keyword( reservation ).

train                   --->        @npNoCompNoMod_lex,
                                    @keyword( train ).

rail                    --->        @npNoCompNoMod_lex,
                                    @keyword( rail ).

detail                  --->        @npNoCompNoMod_lex,
                                    @keyword( gencon ).

airport                 --->        @npNoCompNoMod_lex,
                                    @keyword( airport ).

restaurant              --->        @npNoCompNoMod_lex,
                                    @keyword( restaurants ).

guidebook               --->        @npNoCompNoMod_lex,
                                    @keyword( guidebook ).

guide                   --->        @npNoCompNoMod_lex,
                                    @keyword( guide ).

rental                  --->        @npNoCompNoMod_lex,
                                    @keyword( rentals ).

agent                   --->        @npNoCompNoMod_lex,
                                    @keyword( agent ).

agency                  --->        @npNoCompNoMod_lex,
                                    @keyword( agency ).

airline                 --->        @npNoCompNoMod_lex,
                                    @keyword( airlines ).

airway                  --->        @npNoCompNoMod_lex,
                                    @keyword( airways ).

day                     --->        @npNoCompNoMod_lex,
                                    @keyword( gencon ).

today                   --->        @npNoCompNoMod_lex,
```

```
                                    @keyword( gencon ).

tommorrow               --->        @npNoCompNoMod_lex,
                                    @keyword( gencon ).

regina                  --->        @npNoCompNoMod_lex,
                                    @keyword( regina ).

week                    --->        @npNoCompNoMod_lex,
                                    @keyword( gencon ).

list                    --->        @npNoCompNoMod_lex,
                                    @keyword( gencon ).

ticket                  --->        @npNoCompNoMod_lex,
                                    @keyword( ticket ).

cruise                  --->        @npNoCompNoMod_lex,
                                    @keyword( cruise ).

alaska                  --->        @npNoCompNoMod_lex,
                                    @keyword( alaska ).

boat                    --->        @npNoCompNoMod_lex,
                                    @keyword( boat ).

attraction              --->        @npNoCompNoMod_lex,
                                    @keyword( attractions ).

place                   --->        @npNoCompNoMod_lex,
                                    @keyword( gencon ).


shopping                --->        @npNoCompNoMod_lex,
                                    @keyword( shopping ).

mall                    --->        @npNoCompNoMod_lex,
                                    @keyword( malls ).

%%% personal pronoun

you                      ---> @proper_lex.

i                        ---> @ppronSubj_lex.
he                       ---> @ppronSubj_lex.
she                      ---> @ppronSubj_lex.
they                     ---> @ppronSubj_lex.
we                       ---> @ppronSubj_lex.
```

```
her                     ---> @ppronObj_lex.
him                     ---> @ppronObj_lex.
me                      ---> @ppronObj_lex.
us                      ---> @ppronObj_lex.
them                    ---> @ppronObj_lex.


%%%%%%%%  Adjectives %%%%%%%%

air                     --->   @adjective_lex,
                               @keyword( air ).

east                    --->   @adjective_lex,
                               @keyword( east ).

eastern                 --->   @adjective_lex,
                               @keyword( east ).

west                    --->   @adjective_lex,
                               @keyword( west ).

western                 --->   @adjective_lex,
                               @keyword( west ).

north                   --->   @adjective_lex,
                               @keyword( north ).

northern                --->   @adjective_lex,
                               @keyword( north ).

south                   --->   @adjective_lex,
                               @keyword( south ).

southern                --->   @adjective_lex,
                               @keyword( south ).

central                 --->   @adjective_lex,
                               @keyword( central ).

current                 --->   @adjective_lex,
                               @keyword( current ).

web                     --->   @adjective_lex,
                               @keyword( gencon ).

african                 --->   @adjective_lex,
```

```
                                    @keyword( african ).

asian                    --->       @adjective_lex,
                                    @keyword( asian ).

canadian                 --->       @adjective_lex,
                                    @keyword( canadian ).

road                     --->       @adjective_lex,
                                    @keyword( road ).

travel                   --->       @adjective_lex,
                                    @keyword( travel ).

online                   --->       @adjective_lex,
                                    @keyword( gencon ).

economical               --->       @adjective_lex,
                                    @keyword( economical ).

cheap                    --->       @adjective_lex,
                                    @keyword( cheap ).

lodging                  --->       @adjective_lex,
                                    @keyword( lodging ).

international            --->       @adjective_lex,
                                    @keyword( international ).

auto                     --->       @adjective_lex,
                                    @keyword( auto ).

exchange                 --->       @adjective_lex,
                                    @keyword( exchange ).

one                      --->       @adjective_lex,
                                    @keyword( gencon ).

two                      --->       @adjective_lex,
                                    @keyword( gencon ).

three                    --->       @adjective_lex,
                                    @keyword( gencon ).

alaska                   --->       @adjective_lex,
                                    @keyword( alaska ).

flight                   --->       @adjective_lex,
```

```
                                @keyword( flight ).

tourist                 --->    @adjective_lex,
                                @keyword( tourist ).

gift                    --->    @adjective_lex,
                                @keyword( gift ).

shopping                --->    @adjective_lex,
                                @keyword( shopping ).

british                 --->    @adjective_lex,
                                @keyword( british ).

business                --->    @adjective_lex,
                                @keyword( business ).

boating                 --->    @adjective_lex,
                                @keyword( boating ).

weather                 --->    @adjective_lex,
                                @keyword( weather ).


%%%%%%%%%%%%%%%%%%%%%%%%
%%% verbs

%%% intransitive
stay                    --->    @intrans_lex,
                                @keyword( accomodation ).

boat                    --->    @intrans_lex,
                                @keyword( boating ).

fly                     --->    @intrans_lex,
                                @keyword( gencon ).
%%% transitive
have                    ---> @trans_lex.

visit                   ---> @trans_lex,
                             @keyword( travel ).
travel                  ---> @trans_lex,
                             @keyword( travel ).
want                    ---> @trans_lex,
                             @keyword( gencon ).
schedule                ---> @trans_lex,
                             @keyword( gencon ).
list                    ---> @trans_lex,
```

```
                                @keyword( gencon ).

know                        ---> @trans_lex,
                                 @keyword( gencon ).

make                        ---> @trans_lex,
                                 @keyword( gencon ).

book                        ---> @trans_lex,
                                 @keyword( booking ).

%%% ditransitive
reserve                 ---> @ditrans_lex,
                             @keyword( gencon ).
see                     ---> @ditrans_lex,
                             @keyword( gencon ).
give                    ---> @ditrans_lex,
                             @keyword( gencon ).
show                    ---> @ditrans_lex,
                             @keyword( gencon ).

%%% control verbs
% e.g., I want to see the report
like                        ---> @icontrolv_lex(bse, inf),
                                 @keyword( gencon ).
to                          ---> @icontrolv_lex(inf, bse),
                                 @keyword( gencon ).
want                        ---> @icontrolv_lex(bse, inf),
                                 @keyword( gencon ).

%%% auxilliaries

can                         ---> @aux_lex(bse).
may                         ---> @aux_lex(bse).
would ---> @aux_lex(bse).
ought                       ---> @aux_lex(inf).
do                          ---> @aux_lex(bse).

%%% preposition
after                       ---> @preposition_lex(after).
at                          ---> @preposition_lex(at).
between                     ---> @preposition_lex(between).
by                          ---> @preposition_lex(by).
during                      ---> @preposition_lex(during).
from                        ---> @preposition_lex(from).
in                          ---> @preposition_lex(in).
on                          ---> @preposition_lex(on).
over                        ---> @preposition_lex(over).
```

```
to                          ---> @preposition_lex(to).
under                       ---> @preposition_lex(under).
with                        ---> @preposition_lex(with).
for                         ---> @preposition_lex(for).
of                          ---> @preposition_lex(of).
about                       ---> @preposition_lex(about).
near                        ---> @preposition_lex(near).


and ---> @and_coord_lex.
or ---> @or_coord_lex.

% ******************
% Lexical rules
% ******************


% ************************************
% singular nouns-> plural nouns
% ************************************

sing_to_pl_noun lex_rule
 (word, @head(noun),
@agr_num(singular),
@spr((ne_synsem_list, Spr)),
@mod(Mod),
@subcat(Sub),
@marking(Mark),
@cont(Cont))
    **>
 (word, @head(noun),
@agr_num(plural),
@spr((ne_synsem_list, Spr)),
@mod(Mod),
@subcat(Sub),
@marking(Mark),
@cont(Cont))
  morphs
     goose becomes geese,
     child becomes children,
     airway becomes airways,
     [k,e,y] becomes [k,e,y,s],
     (X,day) becomes (X,days),
     (X,man) becomes (X,men),
     (X,F) becomes (X,F,es) when fricative(F),
     (X,y) becomes (X,[i,e,s]),
     X becomes (X,s).
```

```
fricative([s]).
fricative([c,h]).
fricative([s,h]).
fricative([x]).

% ************************************
%  bse verbs -> fin verbs
% ************************************

bse_to_3_sing lex_rule
   (word, @head(verb),
  @vform(bse),
  @aux(minus),
  @mod(Mod),
  @inv(Inv),
  @subcat([Subcat|MoreSubcats]),
  @cont(Cont))
 **>
   (word, @head(verb),
  @vform(fin),
  @aux(minus),
  @mod(Mod),
  @inv(Inv),
        @subcat([Subcat|MoreSubcats]),
  @cont(Cont))

%   if bse_to_3_sing_check(Subcat, NewSubcat)
   morphs
(X,y) becomes (X, i, e, s), % try -> tries
have becomes has,
 X    becomes (X, s). % walk -> walks

% For third singular rule, we need to ensure that NPs that the
% verb subcategorize for have the same agreement as the verb itself.

bse_to_3_sing_check(X, X)  % not implemented
if true.


% ************************************
% passive lex rule
% ************************************

bse_to_pass lex_rule
   (word, @head(verb),
          @vform(bse),
  @aux(minus),
  @mod(Mod),
```

```
          @inv(Inv),
          @subcat([Subcat|MoreSubcats]),
          @cont(Cont))
 **>
   (word, @head(verb),
      @vform(pas),
      @aux(minus),
      @mod(Mod),
      @inv(Inv),
      @subcat([Subcat|MoreSubcats]),
      @cont(Cont))

     morphs
(X,y) becomes (X, ied),
give becomes given,
see becomes seen,
have becomes had,
X becomes (X,ed),
        (X,e) becomes (X, ed).

% ************************************
% gerform lex rule
% ************************************

gerform lex_rule
   (word, @head(verb),
          @vform(bse),
  @aux(minus),
  @mod(Mod),
  @inv(Inv),
  @subcat([Subcat|MoreSubcats]),
  @cont(Cont))
 **>
   (word, @head(verb),
          @vform(ger),
          @aux(minus),
  @mod(Mod),
          @inv(Inv),
          @subcat([Subcat|MoreSubcats]),
          @cont(Cont))

      morphs
        (X,e) becomes (X, ing),
travel becomes travelling,
X becomes (X, ing).

% ****************************************************************
%  Principles
```

```
% ************************************************************

% ************************************************************
% head_feature_principle(Mother, Head_Dtr)
% The HEAD value of any headed phrase is structure-shared
% with the HEAD value of the head daughter.
% ************************************************************

head_feature_principle(@head(X), @head(X))
if true.

% ************************************************************
% subcat_principle(Mother, Head_Dtr, Comp_Dtr_Synsems)
% ************************************************************

subcat_principle( @subcat(MotherSubcat),
  @subcat(HeadDtrSubcat),
  CompDtrSynsems)
if append(MotherSubcat, CompDtrSynsems, HeadDtrSubcat).

% ************************************************************
% semantic_principle(Mother, SemHead_Dtr, Comp_DtrsList)
% ************************************************************
semantic_principle( SemHd, SemHd, [] ) if
 true.
semantic_principle( Comp, [], Comp ) if
true.
semantic_principle( Mother, SemHd, [Comp|Comps] ) if
              unify_cont_features( SemHd, Comp, NewSemHd ),
      semantic_principle( Mother, NewSemHd, Comps ).

% ************************************************************
% specifier_principle(Spec_Dtr, Head_Dtr)
% ************************************************************

/* original
specifier_principle( @head(OtherDtr), synsem: HeadDtrSynsem) if
specifier(OtherDtr, HeadDtrSynsem).
*/

% adding constrain that SpecDtr must have an empty subcat
specifier_principle( (@head(OtherDtr), @subcat(e_list)),
synsem: HeadDtrSynsem) if
specifier(OtherDtr, HeadDtrSynsem).

%specifier(subst, _) if true.
%specifier((funct, spec: Spec), Spec) if true.
```

```
% ***************************************************************
% marking_principle(Mother, Dtr)
% ***************************************************************

marking_principle( @marking( Mark ), @marking( Mark ) ) if
true.

% ***************************************************************
%  principles(Mother, HeadDtr, SemHead,
% OtherDtrs, CompDtrsSynsem)
% ***************************************************************

principles(Mother, Head, SemHead, Comps, CompDtrsSynsem) if
(head_feature_principle(Mother, Head),
 subcat_principle(Mother, Head, CompDtrsSynsem),
 marking_principle( Mother, Head ),
 semantic_principle(Mother, SemHead, Comps)
 ).

% ***************************************************************
% unify_cont_features( SemHd, Comp, NewSemHd )
% ***************************************************************
unify_cont_features( SemHd, Comp, NewSemHd ) if
unify_action_features( SemHd, Comp, NewSemHd ),
        unify_sem_list_features( SemHd, Comp, NewSemHd ),
        unify_sem_mark_features( SemHd, Comp, NewSemHd ).



unify_action_features( @action(Hd), @action(Comp),
( @action(Hd), @action(Comp) )
      ) if
true.



% Semantic marker of mother is inherited from the semantic marker
% of the semantic head daughter,
% except in the case of prepositional phrases
% For this later case, semantic marker of mother is obtained
% by unifying the semantic marker of semantic head daughter
% and the complement dtr.
unify_sem_mark_features( (@head(prep), @smarker(Hd)),
 @smarker( Comp ),
 (@smarker(Hd), @smarker(Comp))
) if
!, true.
```

```
unify_sem_mark_features( @smarker(Hd), _, @smarker(Hd) ) if
true.

unify_sem_list_features((@cont(keyword:(ne_key_list, (hd:KeyHd,
     tl:e_list)))),
                          (@cont(keyword:(ne_key_list, (hd:KeyComp,
      tl:e_list)))),
           (@cont(keyword:(ne_key_list, (hd:KeyHd,
        tl:(ne_key_list, (hd:KeyComp,
                  tl:e_list))))))
       ) if
                    !, true.


unify_sem_list_features( (@cont(keyword:(ne_key_list, (hd: KeyHd,
      tl: KeyHdTl)))),

  (@cont(keyword:(ne_key_list, (hd: KeyComp,
tl: CompTl)))),

 (@cont(keyword:(ne_key_list, (hd: KeyMother,
       tl: KeyMotherTl))))
     ) if
!, join_finding((ne_key_list, (hd: KeyHd,
                               tl: KeyHdTl)),
    (ne_key_list, (hd: KeyComp,
   tl: CompTl)),
           (ne_key_list, (hd: KeyMother,
  tl: KeyMotherTl))).
/*
unify_key_conjunct( (ne_key_list, (hd:HDF, tl:HDL)),
    CompDtr,
    (ne_key_list, (hd:HDF, tl:(HDL, CompDtr)))
    ) if
    !, true.
*/

join_finding( (ne_key_list, (hd: HdDtrHd,
              tl: HdDtrTl)),
               (ne_key_list, (hd: CompDtrHd,
                              tl: CompDtrTl)),
               (ne_key_list, (hd: HdDtrHd,
     tl: MotherTl))
                ) if
       !, create_join_structure( HdDtrTl, CompDtrHd, CompDtrTl, MotherTl ).
```

```
% ************************************************
% create_join_structure
% ************************************************
create_join_structure( e_list,
       CompDtrHd,
       CompDtrTl,
                      (ne_key_list, (hd: CompDtrHd,
     tl: CompDtrTl))
                      ) if
      !, true.

create_join_structure((ne_key_list, (hd: HdDtrHd,
      tl: HdDtrTl)),
                       CompDtrHd,
CompDtrTl,
(ne_key_list, (hd: HdDtrHd,
       tl: Mother ))
                        ) if
                       !, create_join_structure( HdDtrTl, CompDtrHd,
       CompDtrTl, Mother ).

% ************************************************************
%  Utilities
% ************************************************************

% append(List1, List2, AppendedList)

append([], L, L)
if true.
append([Hd|Tl], L2, [Hd|NewTl])
if append(Tl, L2, NewTl).

% synsem_to_phrase(Synsem, Phrase)

synsem_to_phrase( [], [] )
if !, true.

synsem_to_phrase( [Syn|Synsems], [(phrase, synsem:Syn) | Signs])
if synsem_to_phrase( Synsems, Signs).

% sign_to_synsem( Sign, Synsem )
sign_to_synsem(synsem:Synsem, Synsem) if
true.

% The following predicates are intended for use
% within schema 2 to enforce the requirement
% that complement dtr is not a subject dtr.
```

```
is_not_subject([_Comp|_Comps], _Subj)
if true.

% ******************************************************
%       Grammar rules
% ******************************************************

np_det rule
(@npNoCompNoMod_lex, @keyword( Xyz ))
===>
cat> @determiner_lex,
cat> (@npNoCompNoMod_lex, @keyword( Xyz )).

adj_det rule
(@adjective_lex, @keyword( Xyz ))
===>
cat> @determiner_lex,
cat> (@adjective_lex, @keyword( Xyz )).

schema1 rule
(Mother, phrase, ( @spr(Spr), @subcat(e_list) ) )
===>
cat> (SubjDtr, phrase, synsem:SubjSynsem),
cat> (HeadDtr, phrase, @spr(Spr)),
goal>
     ( principles(Mother, HeadDtr, HeadDtr,  [SubjDtr], [SubjSynsem]) ).

% Here, we should enforce that Comp cannot be the subject daughter
% Otherwise, sentence like 'kim walk' would be
% admissible by both schemas 1 and 2.

schema2 rule
(Mother, phrase, @subcat([SubjSynsem]))
===>
cat> (HeadDtr, word, (@subcat([SubjSynsem|CompSynsem]))),
goal> (synsem_to_phrase(CompSynsem, Comp),
       is_not_subject(Comp, SubjSynsem)
       ),
cats> Comp,
goal> ( principles(Mother, HeadDtr, HeadDtr, Comp, CompSynsem) ).

% Admission of auxiliary and inverted structure.

% To avoid the admissibility of "can who walk",
% we should enforce more constraint on the head daughter
% should enforce that nonloc is empty

schema3 rule
```

```
(Mother, phrase, @subcat(e_list))
===>
cat> (HeadDtr, word, (@subcat([SubjSynsem|CompSynsem]),
                      @aux(plus),
@inv(plus))),
goal> synsem_to_phrase([SubjSynsem|CompSynsem], SComps),
cats> SComps,
goal> ( principles(Mother, HeadDtr, HeadDtr, SComps,
[SubjSynsem|CompSynsem]) ).

%%% specifier head

specifier_head rule
(Mother, phrase, ( @spr(e_list), @subcat(e_list) ) )
===>
cat> (SpecDtr, phrase, ( @head(det), @spr(e_list) ) ),
goal> (sign_to_synsem(SpecDtr, SpecDtrSynsem)),
cat> (HeadDtr, phrase, @spr([SpecDtrSynsem]) ),
% cat> (HeadDtr, phrase, (@spr([SpecDtrSynsem]), @subcat(e_list))),
goal> (
%  principles(Mother, HeadDtr, HeadDtr,  [SpecDtr], [SpecDtrSynsem]),
       head_feature_principle( Mother, HeadDtr),
semantic_principle(Mother, HeadDtr, [SpecDtr]),
specifier_principle( SpecDtr, HeadDtr )
).


% enforcing the req. that SpecDtr must be a determiner
/*
specifier_head rule
(Mother, phrase,  ( @spr(e_list), @subcat(Sub)) )
===>
cat> (SpecDtr, word, (@head(det), @spr(e_list), @subcat(Sub)) ),
goal> (sign_to_synsem(SpecDtr, SpecDtrSynsem)),
cat> (HeadDtr, phrase, @spr([SpecDtrSynsem]) ),
goal> (
%  principles(Mother, HeadDtr, HeadDtr,  [SpecDtr], [SpecDtrSynsem]),
       head_feature_principle( Mother, HeadDtr),
semantic_principle(Mother, HeadDtr, [SpecDtr]),
specifier_principle( SpecDtr, HeadDtr )
).
*/


%%% head/adjunct structures

schema5a rule
(Mother, phrase)
```

```
===>
cat> (AdjnDtr, phrase, ( @mod(Mod), ((@head(noun), @spr(ne_synsem_list));
      @head(adj);
     (@head(prep),@subcat([@head_s(dummy)]) )
     )
) ),
cat> (HeadDtr, phrase, synsem: Mod),
goal> ( principles(Mother, HeadDtr, AdjnDtr, [HeadDtr], □) ).

schema5b rule
(Mother, phrase)
===>
cat> (HeadDtr, phrase, synsem: Mod),
cat> (AdjnDtr, phrase, (@mod(Mod), @head(prep), @subcat([@head_s(dummy)]) ))
goal> ( principles(Mother, HeadDtr, AdjnDtr, [HeadDtr], □) ).

conjunct rule
(Mother, phrase, (@head(Head), @subcat(e_list)) )
===>
cat> (FirstDtr, phrase, (@head( Head ), @subcat(Subcat), @spr(Spr))),
cat> (ConjDtr, phrase, (@head(mark), @marking(conj))),
cat> (SecDtr, phrase, (@head(Head), @subcat(Subcat), @spr(Spr))),
goal> ( marking_principle( Mother, ConjDtr ),
semantic_principle( Mother, FirstDtr, [SecDtr] ) ).



word_to_phrase_0 rule
(phrase,synsem:Synsem)
===>
cat> (word, synsem:(Synsem, @subcat_s(e_list))).

word_to_phrase_1 rule
(phrase, synsem:Synsem)
===>
cat> (word, synsem:(Synsem,@subcat_s(ne_list))).

*************************************************************************
```

```
*************************************************************************

                        Front-end HTML files

*************************************************************************

This file splits the screen into three frames.

*************************************************************************
<html>
<head>
<title> Natural Language Interface To Internet Search Engines </title>
</head>
<frameset cols="30%, 70%">
<frame src="links.html" name = "frame1">
<frameset rows="75%, 25%">
<frame src="main.html" name = "frame2">
<frame src="comments.html" name = "frame3">
</frameset>
</frameset>
</html>

*************************************************************************
```

```
*************************************************************************

      This file provides links to the exisiting search engines.

*************************************************************************


<html>
<head>
</head>
<BODY BGCOLOR = "#BB99AA">
</BODY>
<h2>
Links to selected search engines
</h2>
<h4>
Keyword and Phrasal Searching:
</h4>
<a href="http://www.lycos.com/" TARGET = "parent"><img src="lycos1.gif"></a>
<a href="http://www.webcrawler.com/" TARGET = "parent"><img src="web2.gif"><
<a href="http://www.yahoo.com/" TARGET = "parent"><img src="yah1.gif"></a>
<a href="http://www.opentext.com/" TARGET = "parent"><img src="open1.gif"></
<h4>
Concept Searching:
</h4>
<a href="http://www.excite.com/" TARGET = "parent"><img src="excite1.gif"></


<h4>
Natural Language Searching:
</h4>
<a href="http://www.infoseek.com/" TARGET = "parent"><img src="info1.gif"></
</html>

*************************************************************************
```

```
*****************************************************************************

                    Code for the comments form.

*****************************************************************************

<html>
<head>
<SCRIPT LANGUAGE="JavaScript">
<!--  HIDE ME FROM THAT BROWSER
var timerID = null
var timerRunning = false

function stopclock(){
    // cannot directly test timerID on DEC OSF/1 in beta 4.
    if(timerRunning)
    clearTimeout(timerID)
    timerRunning = false
    }

function startclock(){
     // Make sure the clock is stopped
      stopclock()
      showtime()
     }
function showtime(){
    var now = new Date()
    var hours = now.getHours()
    var minutes = now.getMinutes()
    var seconds = now.getSeconds()
    var timeValue = "" + ((hours > 12) ? hours - 12 : hours)
    timeValue  += ((minutes < 10) ? ":0" : ":") + minutes
    timeValue  += ((seconds < 10) ? ":0" : ":") + seconds
    timeValue  += (hours >= 12) ? " P.M." : " A.M."
    document.clock.face.value = timeValue
    timerID = setTimeout("showtime()",1000)
    timerRunning = true
    }
function testfun(){
document.test.submit();
}

    //-->
</SCRIPT>

</head>
```

```
<BODY BGCOLOR = "#BB99AA"  onLoad="startclock()">
<h2>
<CENTER>
COMMENTS </CENTER></h2>
<div align = right>
<FORM NAME="clock" onSubmit="0">
     <INPUT TYPE="text" NAME="face" SIZE=13 VALUE ="">
</div>
</FORM>
<align = left>If you wish to enter your comments/suggestions please
              fill in the form. </align>
<FORM NAME = "comments" METHOD="post" ACTION="http://www.cs.uregina.ca/
                    cgi-bin/cgiwrap?user=mahaling&script=comments.pl">
<dl>
<dt> Name :
<dd><input type="text" name= "name" size=30>
<dt> EMail Address:
<dd><input type="text" name="email" size=30>
</dl>
<p>
Your comments / suggestions :
<textarea name="suggs" rows=4 cols=65></textarea>
<p>
<input type= "submit" value="submit the form">
<input type= "reset" value="clear the form" >
</body>
</form>
<hr>
<BR>

</BODY>

</html>
```

****

```
*************************************************************************
  This is the main HTML file which allows the users to select the search
  engine and enter the text for searching.

*************************************************************************

<html>
<head>
<SCRIPT LANGUAGE="JAVASCRIPT">
<!--
function testfun(form){
var cSname = form.sname.value
var cText = form.text.value
var nStringLength = cText.length
   if((!form.sname[0].selected) && (!form.sname[1].selected)
   && (!form.sname[2].selected) && (!form.sname[3].selected)
   && (!form.sname[4].selected)) {
   alert("You have not selected the search engine.");
    }
   else if(cText == "")
       alert("You have not entered the text for searching.")
    else {
       document.test.submit();
       }
}
function tessel(){
alert( "Please check the list of words in the lexicon");

}
//-->
</SCRIPT>
</head>
<body bgcolor = "#BB99AA" background = "fossil4.gif" LINK="#994455">

<h2>
<center>
<applet code=welcome1.class WIDTH=400 Height = 40> <PARAM NAME="text"
VALUE="English Access To Internet Search Engines"></applet>
</center>
</h2>
<h3>
   WELCOME!!! This site is a prototype model for English
   access to Internet Search engines. No more hassles of
   finding appropriate keywords or synonyms. Just select
   any search engine and enter the  phrase / sentence to
   be searched. At present, this search is limited to the
   <a href = "domain.html"><strong>''travel'' </strong>
```

```
   domain (with 150 words in the lexicon) </a>.
<br>
</h3>

<FORM NAME="test" METHOD="post" ACTION="http://www.cs.uregina.ca
                    /cgi-bin/cgiwrap?user=mahaling&script=nla.pl">

<FONT SIZE = 4>
<br>
Select any search engine:
<br>
<SELECT NAME="sname" size = 3 onchange = "tessel()">
<OPTION>ALTAVISTA</OPTION>
<OPTION>HOTBOT</OPTION>
<OPTION>INFOSEEK</OPTION>
<OPTION>YAHOO</OPTION>
<OPTION>WEB CRAWLER</OPTION>
</SELECT>
<a href="http://www.cs.uregina.ca/~mahaling/nlaise/tips.html"
       TARGET = "frame2"><img hspace = 30 src="tfs1.gif"></a>
</br>
<br>
<br>
Enter the text to be searched:
<br>
<textarea name = "text" rows="4" cols="40"></textarea>
<br>
<br>
</FONT>
<FONT SIZE = 5>
<B>
<input type="button" name="button1" value="Continue... "
                          onClick = "testfun(this.form)">
<input type="reset" name="Clear All">
</B>
</FONT>
</FORM>
</BODY>

</html>

**************************************************************************
```

```
*************************************************************************

   This file contains the code for back-end PERL script, and semantic
   intrepreter.

*************************************************************************

#!/usr/local/bin/perl.5

# Define constants.
$recipient = 'mahaling@cs.uregina.ca';

# Print out a content-type for HTTP/1.0 compatibility
print "Content-type: text/html\n\n";

# Print a title and initial heading.

print "<Head><Title>NLA SCRIPT.</Title></Head>";

# Get the input.
read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});

# Split the name-value pairs
@pairs = split(/&/, $buffer);

foreach $pair (@pairs)
{
    ($name, $value) = split(/=/, $pair);

    # Un-Webify plus signs and %-encoding
    $value =~ tr/+/ /;
    $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;


    #  For debugging purposes
    # print "Setting $name to $value<P>";

    $FORM{$name} = $value;
}



$sename = $FORM{'sname'};
$ttext  = $FORM{'text'};

# If there is no name entered, then produce an error message.
&invalid_response unless $FORM{'sname'};
```

```
# subroutine blank_response
sub invalid_response
{
  print "<FONT SIZE = 5><b>  The required field \"Search Engine Name\"
  was blank.  Please enter this.</b></FONT>";
  print "<A HREF=\"http://www.cs.uregina.ca/~mahaling/nlaise/main.html\">
  <br><br><font size = 5>Go Back</A></font>";
  exit;
}

&text_response unless $FORM{'text'};
# subroutine blank_response
sub text_response
{
print "<FONT SIZE = 5> <b>  The required field \"text to be searched\"
       was blank. Please enter this.</b></FONT><br><br>";
print "<A HREF=\"http://www.cs.uregina.ca/~mahaling/nlaise/main.html\">
       <FONT SIZE =5> Go Back</FONT></A>";
exit;
}

# Pre-processor transforms capitalized letters and punctuations.

$ttext =~  tr/A-Z/a-z/;
$ttext =~ s/[^a-zA-Z0-9\s]//g;

# Verification for words not in the lexicon and printing
# apprpriate error message.

$lexfile = "/u/mahaling/public_html/cgi-bin/lexifile";
@words = split(' ', $ttext);
foreach $word (@words) {
$lex_lines = `grep $word $lexfile`;
chop ($lex_lines);
if ($lex_lines eq "") {
print "<BODY BGCOLOR = \"#BB99AA\">";
print "<FONT SIZE = 5> <b> Sorry! No matching lexical entry found.
       Please check the list of words in the lexicon and try again.
       </b></FONT><br><br>";
print "<A HREF=\"http://www.cs.uregina.ca/~mahaling/nlaise/main.html\">
       <FONT SIZE = 5> Go Back</FONT></A>";
exit;
}
}


# split the text into words and format it to suit the parser.
```

```
$spa = ' ';
$var1 .= '[';
@words = split(' ', $ttext);
$j = ($#words);
if ($#words == 0) {
$var1 .= $words[0];
splice(@abz, $#abz+1, 0, $var1);
splice(@abz, $#abz+1, 0, $spa);
$var2 .= '].';
splice(@abz, $#abz+1, 0, $var2);
}
else {
$var1 .= $words[0];
$var1 .= $sep;
splice(@abz, $#abz+1, 0, $var1);
for($i=1; $i<$j; $i++) {
splice(@abz, $#abz+1, 0, $words[$i]);
splice(@abz, $#abz+1, 0, $sep);
splice(@abz, $#abz+1, 0, $spa);
}
$var2 .= $words[$#words];
$var2 .= '].';
splice(@abz, $#abz+1, 0, $var2);
}

#print "The array is:@abz<br>";

# Since at present the parser cannot handle the entire syntax of the
# English language, the system has to proceed with the keywords found
# even if there is a syntactic error.

$sep = ',';
$keyfile = "/u/mahaling/public_html/cgi-bin/keyfile";
@words = split(' ', $ttext);
FILE:
for ($i = 0; $i <= $#words; $i++) {
if ($words[$i] eq ' ') {
next FILE;
}
else {
@key_lines = `grep $words[$i] $keyfile`;
$z = 0;
foreach $z (0 .. $#key_lines) {
($field1 , $field2) = split(":", $key_lines[$z]);
if ($field1 eq $words[$i]) {
$comp1 .= $field2;
$comp1 .= $sep;
```

```
}
}
}
#*****************************************************************

#Send the formatted array to the parser.

system("/u/mahaling/search/Internet/research/C_code/client @abz");


# The different parses  of the parser will be analysed by the
# semantic extractor and extracts the keywords alone from the correct parse.
# Get the output of the semantic extractor.

$out = "/u/mahaling/public_html/cgi-bin/newfile";
open(outfile, $out);
$_ = <outfile>;

# Verify for all types of error and print error messages.

if ($_ eq "") {
print "<BODY BGCOLOR = \"#BB99AA\">";
print "<FONT SIZE = 5> Sorry! Background process needed to proceed further";
exit;
}

if (($_ =~ /No parses/)&&($comp1 eq "")) {
print "<BODY BGCOLOR = \"#BB99AA\">";
print "<FONT SIZE = 5> Syntactic Error! No Keywords found!!
       Please retry with some other text. <BR> <a href = \"http://
       www.cs.uregina.ca/~mahaling/nlaise/main.html\">
       Go Back </a><BR></FONT>";
exit;
}

if ((!($_ =~ /No parses/))&&($comp1 eq "")) {
print "<BODY BGCOLOR = \"#BB99AA\">";
print "<FONT SIZE = 5> No Keywords found!! Please retry again.
       <BR> <a href = \"http://www.cs.uregina.ca/~mahaling/
       nlaise/main.html\">Go Back </a><BR></FONT>";
exit;
}

if (($comp1 ne "")) {
  if (($_ =~ /No parses/)) {
  print "<BODY BGCOLOR = \"#BB99AA\">";
  print "<FONT SIZE = 5> Syntactic Error! However search results
  will be displayed based on the keywords selected from your text :
```

```
  @xyz <BR> </FONT>";
}

if (!($_ =~ /No parses/)) {
  print "<body bgcolor = \"#BB99AA\">";
print "<FONT SIZE = 5> Thank You! The keywords selected for
       searching in $sename
       are:<br><br> @xyz <BR> </FONT>";
}

#-------------------------------------------------------------------------
# Findout if there are any synonymes to the keywords.
#-------------------------------------------------------------------------
$synofile = '/u/mahaling/public_html/cgi-bin/synofile';

# Group nominal Compounds before finding synonymes to avoid
# substitution of new or recent in New York

if($comp1 ne "") {
if(($comp1 =~ /south,/)&&($comp1 =~ /africa/)) {
($comp1 =~ s/south,africa/southafrica,/);
}
if(($comp1 =~ /new,/)&&($comp1 =~ /york/)) {
($comp1 =~ s/new,york/newyork,/);
}
@words = split(',', $comp1);
$i = 0;
$j = ($#words);
for($i=0; $i<=$j; $i++) {
@synolines = `grep $words[$i] $synofile`;
foreach $syn (0 .. $#synolines) {
($field1 , $field2) = split(":", $synolines[$syn]);
if ($field1 eq $words[$i]) {
splice(@xyz, $#xyz+1, 0, $field2);
splice(@xyz, $#xyz+1, 0, $sep);
splice(@xyz, $#xyz+1, 0, $spa);
}
}
}
#@syno = `grep $words[$#words] $synofile`;
#foreach $syn1 (0 .. $#syno) {
#($field1 , $field2) = split(":", $syno[$syn1]);
#if ($field1 eq $words[$#words]) {
#splice(@xyz, $#xyz+1, 0, $field2);
#}
#}
}
#-------------------------------------------------------
```

```
#Format the list of keywords with synonymes to suit the individual
#search engine.
$k=0;
for($k=0; $k<=$#xyz; $k++) {
$str1 .= $xyz[$k];
}
$str1alta = $str1;
$str1alta =~ s/\, AND\, / AND /g;
$str1alta =~ s/\, OR\, / OR /g;

$str1 =~ s/\, AND\, \+/ OR  /g;
$str1 =~ s/\, OR\, \+/ OR /g;

$str2 = $str1;
$str2 =~ s/AND/ /g;
$str2 =~ s/OR/ /g;

$str3 = $str1;
$str4 = reverse($str3);
$str4 =~ s/\,//;
$str3 = reverse($str4);
if (($str3 =~/AND/)||($str3 =~ /OR/)) {
$str3 =~ s/\+//g;
$str3 =~ s/\,/ AND /g;
$hot = 'B';
}
else {
$str3 =~ s/\,/ /g;
#$str3 =~ s/\+//g;
chop $str3;
$hot = 'MC';
}

#-------------------------------------------------------------------
#Send the final strings to the search engine selected by the user.
#-------------------------------------------------------------------
   if ($sename eq 'ALTAVISTA') {
    print STDOUT "<FONT SIZE = 5>";
    print STDOUT "<FORM ACTION=\"http://www.altavista.digital.com/
  cgi-bin/query\" METHOD=\"GET\">";
    print STDOUT "<INPUT TYPE=\"submit\" VALUE=\"CONTINUE\">";
    print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"q\" VALUE=\"$str1alta\">";
    print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"what\" VALUE=\"web\" CHECKE
    print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"fmt\" VALUE=\"d\">";
    print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"pg\" VALUE=\"q\">";
    print STDOUT "</FONT>";
    print STDOUT "</FORM>";
```

```
 }

 if ($sename eq 'YAHOO') {
 print STDOUT "<FONT SIZE = 5>";
 print STDOUT "<FORM ACTION=\"http://search.yahoo.com/bin/search\"
METHOD=\"GET\">";
 print STDOUT "<INPUT TYPE=\"submit\" VALUE=\"CONTINUE\">";
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"p\" VALUE=\"$str1\">";
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"n\" VALUE=\"50\" SELECTED>";
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"t\" CHECKED>";
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"u\" CHECKED>";
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"c\" CHECKED>";
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"s\" VALUE=\"a\" CHECKED>";
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"w\" VALUE=\"s\">";
 print STDOUT "</FONT>";
 print STDOUT "</FORM>";
 }

 if ($sename eq 'INFOSEEK') {
 print STDOUT "<FONT SIZE = 5>";
 print STDOUT "<FORM ACTION=\"http://www2.infoseek.com/Titles\"
METHOD=\"GET\">";
 print STDOUT "<INPUT TYPE=\"submit\" VALUE=\"CONTINUE\">";
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"qt\" VALUE=\"$str1\">";
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"cat\" VALUE=\"\">";
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"col\" VALUE=\"WW\">";
 print STDOUT "</FONT>";
 print STDOUT "</FORM>";
 }

 if ($sename eq 'LYCOS') {
 print STDOUT "<FONT SIZE = 5>";
 print STDOUT "<FORM ACTION=\"http://lycospro.lycos.com/cgi-bin/
pursuit\" METHOD=\"GET\">";
 print STDOUT "<INPUT TYPE=\"submit\" VALUE=\"CONTINUE\">";
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"query\" VALUE=\"$str2\">";
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"cat\" VALUE=\"lycos\" SELECTI
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"matchmode\" VALUE=\"and\">";
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"mtemp\" VALUE=\"nojava\">";
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"etemp\" VALUE=\"error_nojava'

 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"maxhits\" VALUE=\"50\">";
 print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"adv\" VALUE=\"1\">";

 print STDOUT "</FONT>";
 print STDOUT "</FORM>";
 }
```

```
    if ($sename eq 'WEB CRAWLER') {
    print STDOUT "<FONT SIZE = 5>";
    print STDOUT "<FORM ACTION=\"http://webcrawler.com/cgi-bin/WebQuery\"
   METHOD=\"GET\">";
    print STDOUT "<INPUT TYPE=\"submit\" VALUE=\"CONTINUE\">";
    print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"searchText\" VALUE=\"$str1\"
    print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"andOr\" VALUE=\"\">";
    print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"maxHits\" VALUE=\"50\">";
    print STDOUT "</FONT>";
    print STDOUT "</FORM>";
    }

    if ($sename eq 'HOTBOT') {
    print STDOUT "<FONT SIZE = 5>";
    print STDOUT "<FORM ACTION=\"http://www.hotbot.com/\" METHOD=\"GET\">";
    print STDOUT "<INPUT TYPE=\"submit\" VALUE=\"CONTINUE\">";
    print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"sw\" VALUE=\"web\">";
    print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"SM\" VALUE=\"$hot\">";
    print STDOUT "       ";
    print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"MT\" VALUE=\"$str3\">";
    print STDOUT "<INPUT TYPE=\"hidden\" NAME=\"DE\" VALUE=\"1\">";
    print STDOUT "</FONT>";
    print STDOUT "</FORM>";
    }

# ----------------------------------------------------
```

Bibliography

[1] AAAI Symposium on Natural Language Processing in Web. 1997. http://crl.nmsu.edu/users/mahesh/aaai-web-nlp-symposium.html.

[2] Alta Vista Home Page. June 1997. http://altavista.digital.com/.

[3] ANDROUTSOPOULUS, I., RITCHIE, G. D. and THANISCH, P. March 1995. "Natural Language Interfaces to Databases - An Introduction". *Computation and Language E-Print Archive.* http://xxx.lanl.gov/list/cmp-lg/9503, no. cmp-lg/9503016.

[4] ANIKINA, N., GOLENDER, V., KOZHUKHINA, S., VAINER, L., ZAGATSKY, B. March 1997. "REASON: NLP-based Search System for the WWW". *AAAI Spring Symposium Paper on Natural Language Processing for the World Wide Web.* Stanford University, USA.

[5] CAPINDALE, R. A. and CRAWFORD, R. G. 1990. "Using a Natural Language Interface with Casual Users". *International Journal of Man-Machine Studies.* 32: 341-362.

[6] CARPENTER, B. and PENN, G. December 1994. "The Attribute Login Engine User's Guide". *Computational Linguistics Program.* Carnegie Mellon University, PA.

[7] CERCONE, N., McFETRIDGE, P., POPOWICH, F., et.al. 1993. "The SystemX Natural Language Interface: Design, Implementation and Evaluation". *Technical Report 93-03.* Simon Fraser University, BC, Canada.

[8] CERCONE, N. and McCALLA, G. 1986. "Accessing Knowledge through Natural

[9] CHANDRASEKAR, R. and RAMANI, S. 1989. "Interactive communication of sentential structure and content: an alternative approach to man-machine communication". *International Journal of Man-Machine Studies.* volume: 30. pp 121-148.

[10] Eureka! Secrets of Searching the Web and Promoting your Web site. June 1997. http://www.best.com/mentorms/eureka—i.htm.

[11] Excite Home Page. June 1997. http://www.excite.com.

[12] GAL, A., LAPALME, G., et.al., 1991. *Prolog for Natural Language Processing.* John Wiley & Sons.

[13] GAZDAR, G. and MELLISH, C. 1989. *Natural Language Processing in Prolog: An Introduction to Computational Linguistics.* Addison-Wesley Publishing Co, Menlo Park, CA.

[14] GROSZ, B. J., JONES, K.S. and WEBBER, B. L. 1986. *Readings in Natural Language Processing.* Morgan Kaufmann Publishers Inc.

[15] GUNDAVARAM, S. March 1996. *CGI Programming on the World Wide Web.* O'Reilly & Associates, Inc. CA.

[16] HELBIG, H., GNORLICH, C. and MENKE, D. Realization of a User-friendly Access to Networked Information Retrieval Systems. http://voss.fernuni-hagen.de/gebiete/pi7/papers/dag.ps(.gz).

[17] Infoseek Home Page. June 1997. http://www.infoseek.com.

[18] Infoseek Help Page. June 1997. http://www.infoseek.com/Help?pg=DChelp.html&sv=N4.

[19] Infoseek Search Page. June 1997. http://home.netscape.com/escapes/search/netsearch2.html.

[20] KENT, P. and KENT, J. 1996. *Official Netscape JavaScript Book.* Netscape Press.

aby, BC.

[22] Lycos Home Page. June 1997. http://www.lycos.com.

[23] MATHESON, C. May 1996. HPSG Grammars in ALE. http://www.ltg.hcrc.ed.ac.uk/projects/ledtools/ale-hpsg.

[24] MAYASARI, S. I. September 1995. "An HPSG Lexicon for a Physical Activity Database". *Technical Report CS-95-09.* Department of Computer Science. University of Regina, Regina, SK.

[25] Opentext Home Page. June 1997. http://www.opentext.com.

[26] PENN, G. "Compiling Typed Attribute-Value Logic Grammars". Computational Linguistics Program. Philosophy Department, Carnegie Mellon University.

[27] POLLARD, C. and SAG, I. 1987. *Information-based Syntax and Semantics: Fundamentals.* CSLI. Stanford University, CA. Volume 1.

[28] POLLARD, C. and SAG, I. 1992. *Head-driven Phrase Structure Grammar.* CSLI. Stanford University, CA.

[29] SAMAD, T. 1986. *A Natural Language Interface for Computer-Aided Design.* Kluwer Academic Publishers, USA.

[30] STERLING, L. and SHAPIRO, E. 1986. *The Art of Prolog Advanced Programming Techniques.* MIT Press, USA.

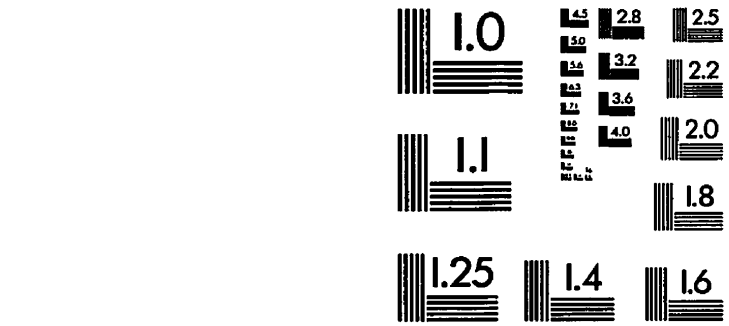
[31] WALL, L. and SCHWARTZ, R. L. 1992. *Programming PERL.* O'Reilly & associates, Inc. USA.

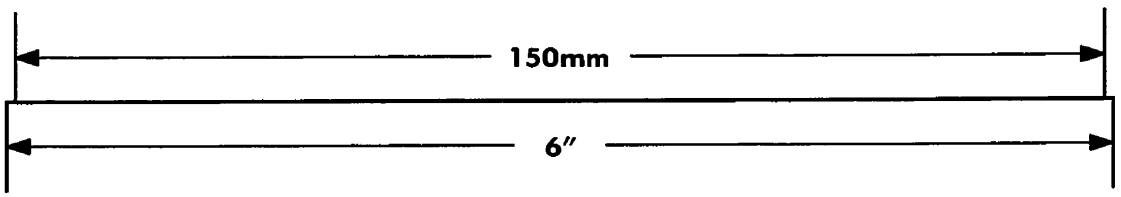
[32] Webcrawler Home Page. June 1997. http://www.webcrawler.com.

[33] Web Master's Guide to Search Engines. June 1997. http://calafia.com/webmasters.

[34] Yahoo Help Page. June 1997. http://search.yahoo.com/search/help?p=&a=n.

[35] Yahoo Home Page. June 1997. http://www.yahoo.com.





APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989