Logical-Linguistic Model and Experiments in Document Retrieval

by

Tengku Mohd Tengku Sembok

A thesis submitted to the
Faculty of Science
University of Glasgow
for the degree of
Doctor of Philosophy

Department of Computing Science
University of Glasgow
August 1989

Dedicated to my parents,
Yak and Wak.

Declaration

This thesis is entirely on my original work and no part is done in collaboration. Where the work of others is used, explicit reference is made in the text. No part of this thesis has been, or is being submitted for a degree at any other university.

CONTENT

Abbreviations

AI	-	Artificial Intelligence
DRS	-	Document Retrieval System
IR	-	Information Retrieval
NLP	-	Natural Language Processing
NLU	-	Natural Language Understanding
PTQ	-	Proper Treatment of Quantification in English Fragment
SILOL	-	Simple Logical-Linguistic Document Retrieval System
SMG	-	Simplified Montague Grammar
SUNG	-	Simple Unification Noun-Phrase Grammar
TMS	-	Theory of Montague Semantics
UNIL	-	Uncertain Implication Language

Financial support and study leave were provided by the
Service Department of Malaysia and the National Univers
Malaysia, which are highly appreciated and gratefully acknowledged.

Acknowledgement

There are many people that I would like to thank and convey my gratefulness. Foremost is my first supervisor, Prof. C.J. van Rijsbergen, who provided me, over the years, with the ideas and supply of the relevant material to this work. I am grateful to Dr. Fairouz Kamareddine, my second supervisor, especially for her supervision and comments in the writing phase of this thesis. My thanks also go to Dr. Ruben Leon and Ian Campbell, for their help in the system related problems and C. I also like to thank my colleagues Roslan, Riaz, Saeed, Hakim, Djamal, Khaled, Francis and others for many happy and unforgettable moments that we all shared together as students in this department. Special thanks go to my colleague Shona Douglas for the reading and comments on parts of the thesis.

Last but not the least, I would like to thank my wife Nora and my daughters, Madeehah and Qanitah, for their patience and support over the years we spent in Glasgow.

Financial support and study leave were provided by the Public Service Department of Malaysia and the National University of Malaysia, which are highly appreciated and gratefully acknowledged.

Summary

Conventional document retrieval systems have relied on the extensive use of the keyword approach with statistical parameters in their implementations. Now, it seems that such an approach has reached its upper limit of retrieval effectiveness, and therefore, new approaches should be investigated for the development of future systems. With current advances in hardware, programming languages and techniques, natural language processing and understanding, and generally, in the field of artificial intelligence, there are now attempts being made to include linguistic processing into document retrieval systems. Few attempts have been made to include parsing or syntactic analysis into document retrieval systems, and the results reported show some improvements in the level of retrieval effectiveness.

The first part of this thesis sets out to investigate further the use of linguistic processing by including *translation*, instead of only parsing, into a document retrieval system. The translation process implemented is based on unification categorial grammar and uses C-Prolog as the building tool. It is used as the main part of the indexing process of documents and queries into a knowledge base *predicate representation*. Instead of using the vector space model to represent documents and queries, we have used a kind of knowledge base model which we call *logical-linguistic model*. A development of a robust parser-translator to perform the translation is discussed in detail in the thesis. A method of dealing with ambiguity is also incorporated in the parser-translator implementation.

The retrieval process of this model is based on a logical implication process implemented in C-Prolog. In order to handle uncertainty in

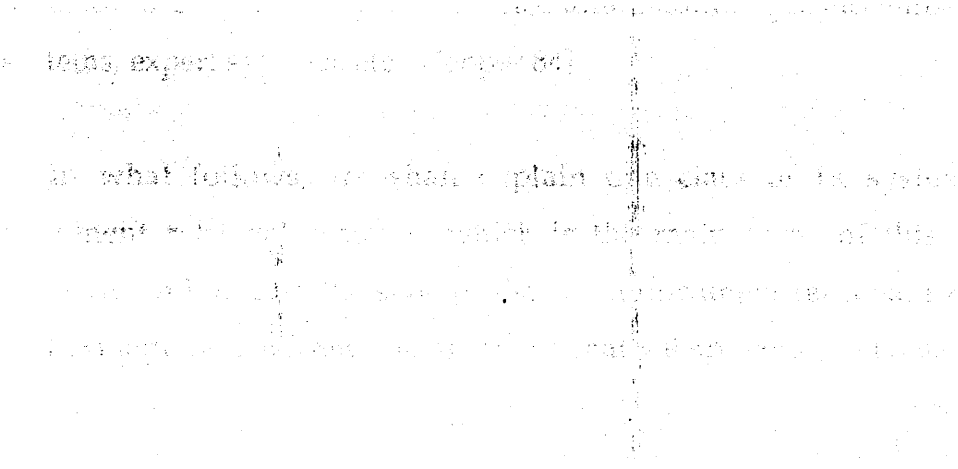
evaluating similarity values between documents and queries, meta level constructs are built upon the C-Prolog system. A logical meta language, called UNIL (UNcertain Implication Language), is proposed for controlling the implication process. Using UNIL, one can write a set of implication rules and thesaurus to define the matching function of a particular retrieval strategy. Thus, we have demonstrated and implemented the *matching* operation between a document and a query as an *inference* using *unification*. An inference from a document to a query is done in the context of global information represented by the implication rules and the thesaurus.

A set of well structured experiments is performed with various retrieval strategies on a test collection of documents and queries in order to evaluate the performance of the system. The results obtained are analysed and discussed.

The second part of the thesis sets out to implement and evaluate the *imaging retrieval* strategy as originally defined by van Rijsbergen. The imaging retrieval is implemented as a relevance feedback retrieval with nearest neighbour information which is defined as follows. One of the best retrieval strategies from the earlier experiments is chosen to perform the initial ranking of the documents, and a few top ranked documents will be retrieved and identified as relevant or not by the user. From this set of retrieved and relevant documents, we can obtain all other unretrieved documents which have any of the retrieved and relevant documents as their nearest neighbour. These unretrieved documents have the potential of also being relevant since they are 'close' to the retrieved and relevant ones, and thus their initial similarity values to the query will be updated according to their distances from their nearest neighbours. From the updated similarity values, a new ranking of documents can be obtained and evaluated.

A few sets of experiments using imaging retrieval strategy are performed for the following objectives: to search for an appropriate updating function in order to produce a new ranking of documents, to determine an appropriate nearest neighbour set, to find the relationship of the retrieval effectiveness to the size of the documents shown to the user for relevance judgement, and lastly, to find the effectiveness of a multi-stage imaging retrieval. The results obtained are analysed and discussed.

Generally, the thesis sets out to define the logical-linguistic model in document retrieval and demonstrates it by building an experimental system which will be referred to as SILOL (a Simple Logical-linguistic document retrieval system). A set of retrieval strategies will be experimented with and the results obtained will be analysed and discussed.



Chapter 1: Introduction

1.1 Information Retrieval

Information Retrieval (IR) can be defined broadly as *the study of how to determine and retrieve from a corpus of stored information the portions which are responsive to particular information needs.* Let us assume that there is a store consisting of a large collection of information on some particular topic, or a combination of various topics. The information may be stored in a highly structured form or in an unstructured form, depending upon its application. A user of the store, at times, seeks certain information which he may not know. He therefore has to express his information need as a request for information in one form or another. Thus IR is concerned with the determining and retrieving of information that is relevant or likely to be relevant to his information need as expressed by his request. Some recent research in IR has demonstrated a wide range of topics encompassed by this definition, e.g. document retrieval systems, database management systems, office automation, question-answering systems, expert systems, etc. [Cooper 84].

In what follows, we shall explain one class of IR systems, i.e document retrieval systems, which is the main focus of this thesis. Then we will discuss the state-of-the-art in document retrieval systems, and explain the aims and the approach that we are going to take.

1.2 Document Retrieval Systems

Document retrieval systems constitute one class of IR systems and

are considered by some researchers as the main focus of interest in IR [BCS 87]. Research in this area is concerned with the investigation of techniques to effectively retrieve more relevant material, and the use of software or hardware techniques to efficiently implement the retrieval operations.

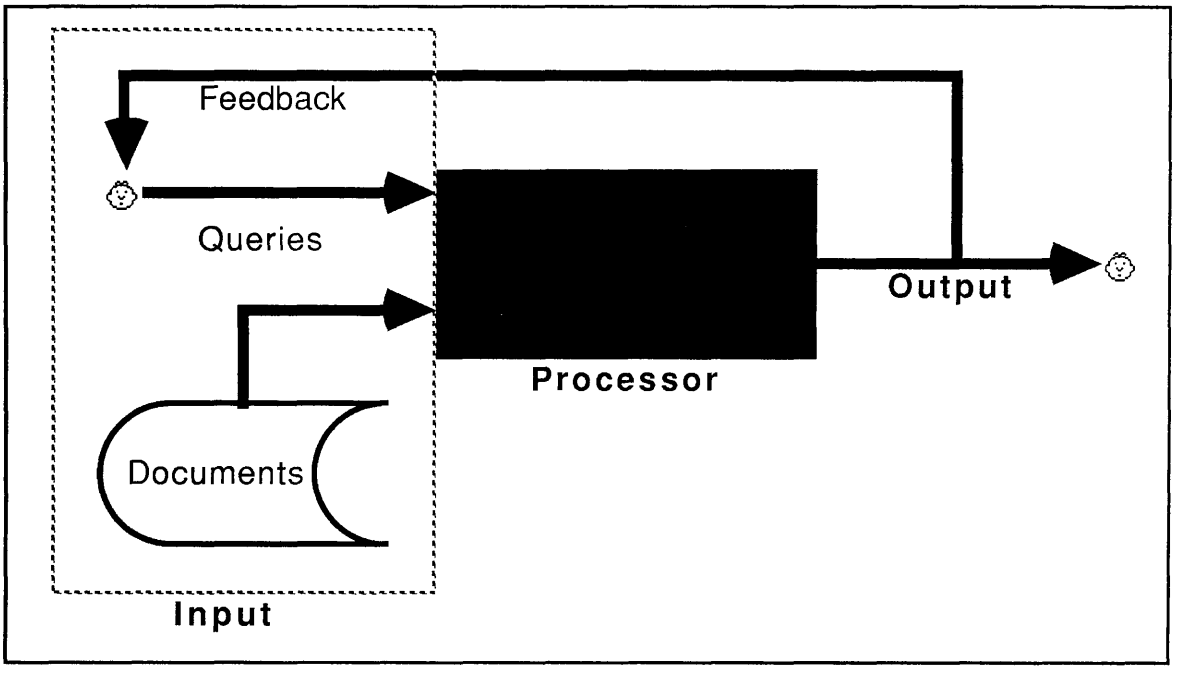


Figure_1.1: A Typical Document Retrieval System

A typical document retrieval system can be illustrated by the diagram in Figure_1.1 as a system constituting of **input, processor** and **output** [van Rijsbergen 79]. The main input to the system are queries and documents which are originally in the form of natural language which are not suitable for computers. Therefore, the main problem here is to find a suitable representation of each document and query for computer use. The representation could be based upon a list of **keywords**, or extracted words which are considered to be important, and an artificial language in which the queries and documents can be formulated. During a search session, it is possible for a user to input a **feedback** to the system after a sample or initial retrieval in order to

improve the subsequent retrieval. The feedback can be in the form of updating or revising the request or identification of relevant documents from the output of the initial retrieval.

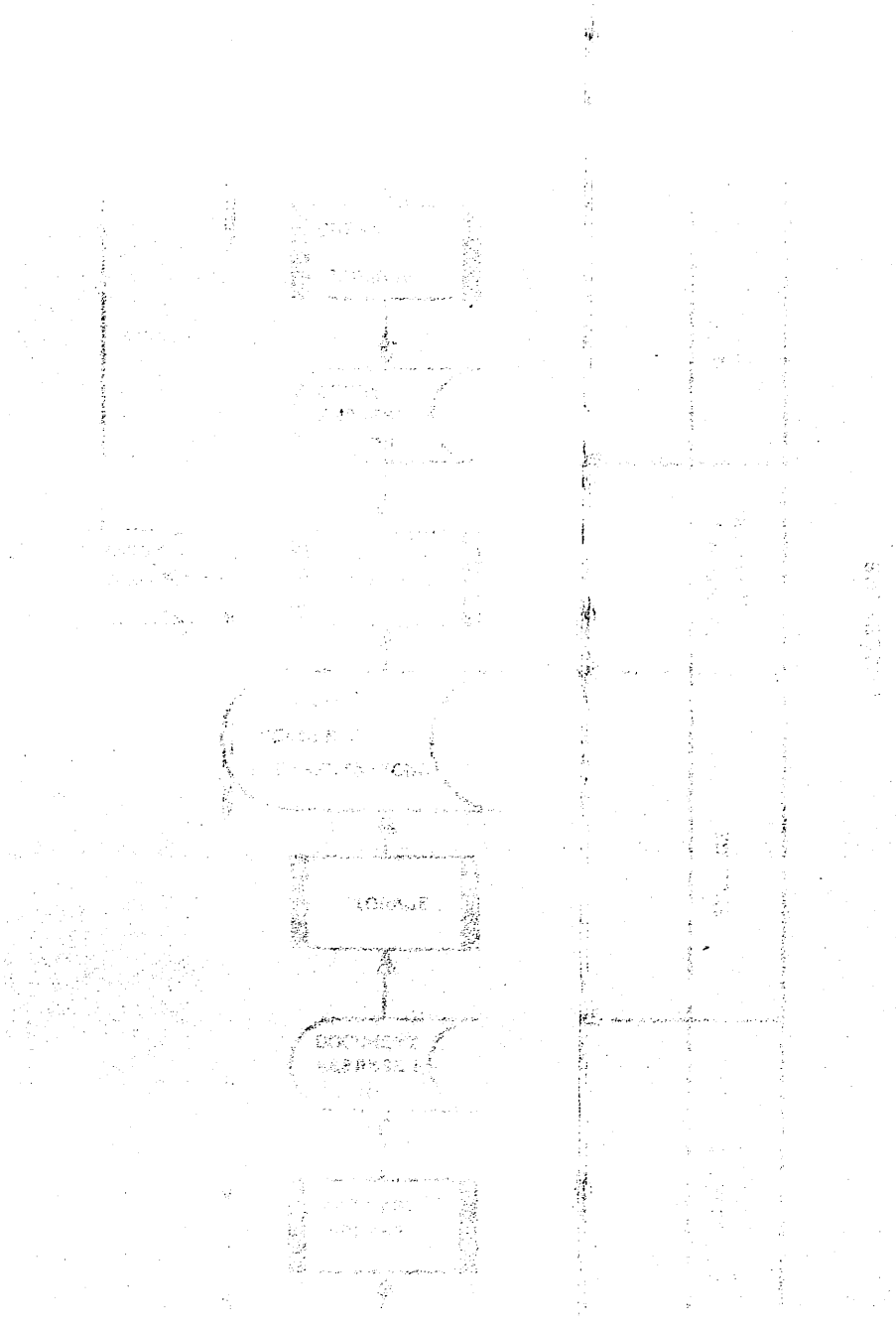
The processor is the component which is concerned with the retrieval process which performs the actual retrieval function in response to a query. The retrieval process may also involve structuring of the documents in some appropriate way, such as classifying and clustering them in order to improve the retrieval effectiveness.

The output of the system is usually in the form of a set of citations or document numbers. As far as an operational system is concerned, this is the end product of the system. But in an experimental system, the output is evaluated in order to assess the performance of the system.

Figure_1.2 illustrates a general structure of a document retrieval system in more details [Harper 80]. The necessary processes of the system are query formulation, query indexing, document indexing, storage, and retrieval. Associated with each process is a processor and the object generated from it. For example, the query formulation process is associated with a processor, the user, which generates a query out of an information need. The query indexer and the document indexer are two important processors of the system, in which the content analyses of the query and the documents are performed to generate their reduced representations. The storage processor organises the whole document representations and stores them in the system. The retrieval processor performs similarity calculations between the query representation and the stored document representations in order to retrieve the relevant documents which fulfill the user's information

need.

In the following section, we will describe the state-of-the-art of document representation and retrieval strategy.



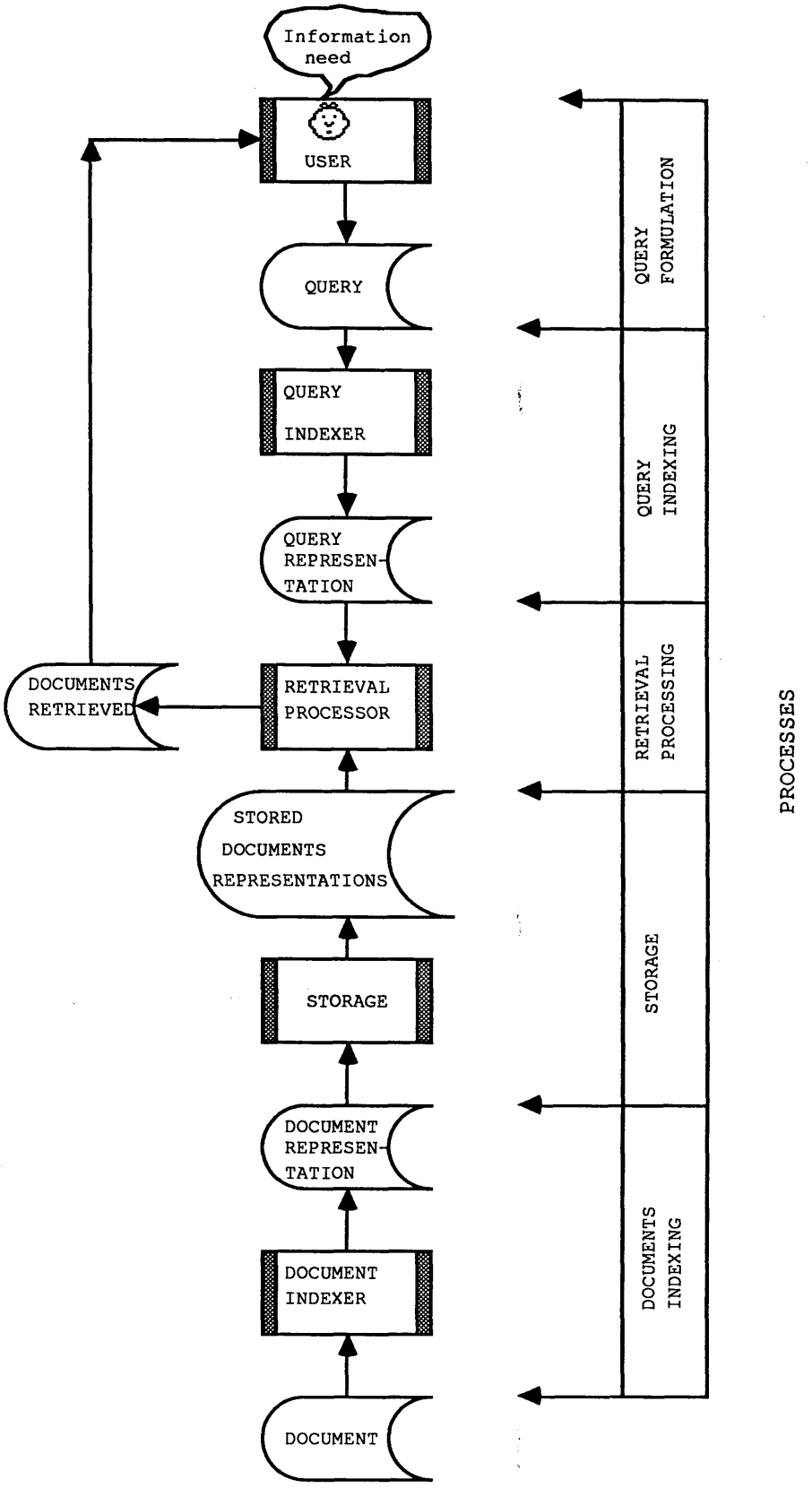


Figure_1.2: A General Structure of a Document Retrieval System

1.3 State-of-the-art

Since natural languages are not suitable for computer use, models have been constructed within which documents and queries are represented and retrieval strategies formulated to retrieve documents which are relevant with respect to a query. In this section we will describe some models which are commonly used in conventional document retrieval systems.

1.3.1 Conventional Models

In conventional document retrieval systems, documents are represented by sets of keywords, or index terms of the form

$$D_i = (t_{i_1}, w_{i_1}; t_{i_2}, w_{i_2}; \ldots; t_{i_n}, w_{i_n})$$

where w_{i_j} represents the value or weight of term t_{i_j} which is assigned to document D_i. In ordinary retrieval environments which are based on the *set theoretic* model, the terms are unweighted. Thus, the values of the w_{i_j} are restricted to either 0 or 1 for terms that are respectively absent from, or present in, a given document. The requests are expressed as Boolean combinations of index terms using logical operators *and*, *or*, and *not*. For example, a query Q_j might be expressed as

$$Q_j = ((t_{j_i} \text{ and } t_{j_2}) \text{ or } t_{j_3}).$$

In response to the query given above, all documents indexed either

by the combination of t_{j1} and t_{j2}, or by t_{j3} would be retrieved.

The systems which are based on this model exhibit several well known disadvantages, for examples [Salton&Buckley 88]:

1) the generation of Boolean queries is a complex task and, therefore, trained search intermediaries are needed;

2) the ranking of the documents retrieved, in decreasing order of relevance with respect to a particular query, cannot be provided;

3) there is no obvious way of limiting the number of documents retrieved;

4) term importance cannot be assigned to search terms since there is no weighting scheme.

The *vector space* model offers a solution to the shortcomings mentioned above. In this model, both documents and queries are expressed as a set of weighted index terms, and Boolean operators are not used in formulating the queries. In a vector space system using n different index terms $\{t_1,t_2,\ldots,t_n\}$, a document D_i and a query Q can be expressed as n-dimensional vectors of the form

$$D_i = (d_{i1},d_{i2},\ldots,d_{in}) \text{ and } Q = (q_1,q_2,\ldots,q_n)$$

where d_{ij} and q_j represent the weights of term j in document D_i and query Q, respectively. The basic weights for the terms throughout the document collection are normally calculated using statistical techniques. For example, using the *inverse document frequency*(idf)

weighting scheme, the basic weight for term t_j is calculated as

$$w_j = - \log (Freq(t_j) / N)$$

where $Freq(t_j)$ is the number of documents in which the term t_j appears at least once and N is the total number of documents in the system. Then, the weights of term t_j assigned to the vectors D_i or Q are the values of some functions of w_i which may include the frequency of the term t_j occuring in D_i or Q. The terms that are absent from the respective vectors are given zero weights.

There are many ways of computing similarity coefficients between a given query and each stored document. For example, one can use the well-known inner-product function or the cosine similarity function as follows:

Inner Product Function:

$$sim(D_i,Q) = \Sigma_{j=1,n} (q_j \cdot d_{ij})$$

Cosine Function:

$$sim(Di,Q) = (\Sigma_{j=1,n} d_{ij} \cdot q_j) / (\sqrt{\Sigma_{j=1,n} d_{ij}^2} \cdot \sqrt{\Sigma_{j=1,n} q_j^2})$$

With the availability of similarity coefficients, the documents retrieved can be ranked in decreasing order of the similarity values. The number of documents to be retrieved can also be limited by imposing a threshold value on the similarity coefficients that must be achieved by documents which are to be retrieved. Nevertheless, vector space systems suffer from one main disadvantage. The *synonym* specifications, reflected by or-clauses, and *phrases*, represented by and-clauses, cannot be expressed in vector space representation. These two concepts are considered very useful in enhancing the effectiveness of retrieving relevant documents. Because, some words are highly specific

and thus identify a narrow range of concepts, whereas some are very general and thus are associated with a broad range of concepts. A highly specific word will retrieve too few documents, and on the other hand, a very general word will retrieve too many documents. In order to avoid the problems posed by words having too high or too low specificity, it is often suggested that thesaurus and phrases be used in indexing. Thus, to tackle part of the problem, the simple vector space model have been refined to have more than one vector to represent a query or document [Fox 83]. Each of the vectors may represent different kind of information. The use of extended vectors for phrase indexing has been implemented by Fagan in [Fagan 87].

There are other models besides the two main ones discussed above, such as the fuzzy set theoretic [Radecki 77], the extended boolean [Salton 88] and the probabilistic [Harper 80] models. They are either variants of the two main models or closely related to them, at least, as far as document and query representations are concerned.

1.3.2 Keywords vs Meaning

Until now, almost all of the work in information retrieval (IR) has been based on the assumption that a formal notion of *meaning* is not required to solve IR problems. The keywords approach as demonstrated by the set theoretic model and the vector space model, where absence or presence of keywords and their distributions are the only information being considered, has been typically assumed by many researchers to be sufficient. However, some have concluded that this assumption is *wrong* [van Rijsbergen 86]. The Keywords approach with its statistical techniques is judged to have reached its theoretical limit and further attempts for improvement are considered

a waste of time.

On the other hand, progress towards new models which incorporate the notion of meaning has been very slow. It has been suggested that some attempt should be made to develop a naive model which uses more than just keywords as the content indicator of each document in the system. Thus, one of the objectives set out in this thesis is to investigate, to implement, and to evaluate a document retrieval system using a new model which is based on a *logical-linguistic* framework. In this model the indexing of documents and queries is achieved through semantic translation of natural language into a predicate representation. Its retrieval process is performed through logical implication using Prolog matching and unification primitives coupled with meta level constructs to handle uncertainty in evaluating similarity values between documents and queries.

In what follows, we will introduce the notion of logical-linguistic model and then we shall discuss our approach.

1.4 Logical-linguistic Model

A logical-linguistic model of document retrieval systems, in our view, must have both logical and linguistic capabilities as defined by Cooper [Cooper 78]. The term logical is used generally to include deductive and inductive inference, probability theory and statistical decision theory, whilst, linguistic is used to include syntax, semantics and pragmatics of language.

The need for linguistic capability is obvious in document retrieval systems where documents are in the form of natural language texts and

queries are formulated using natural language. Through linguistic processing a limited representation of the meaning of documents and queries can be obtained.

The logical capability is necessary because the facts expressed directly by stored documents are too particularised in expression to be responsive to a wide variety of requests, whereas, their logical consequences may be sufficient to satisfy a range of related requests.

Ideally, the logical and linguistic components of a logical-linguistic document retrieval system should be based upon a common unified theory of logic and language. A Montague-style semantics [Dowty et al 81] could be an appropriate choice in this type of work. Another possible choice can be the logical representation used in Metamorphosis Grammar as proposed by Colmerauer [Colmerauer 78] and extended by Warren [Warren 82] and Saint-Dizier [Saint-Dizier 86]. Both representations have not yet been chosen as a basis for an IR system. The current state-of-the-art in IR is yet far away from this ideal. Even if we had an appropriate semantics which could be computed efficiently, we still would not know how to use it to retrieve documents in response to requests [van Rijsbergen 86].

Thus, our aim is to investigate, implement and evaluate a simple system which is based on this model. The proposed system will be referred to as **SILOL** (**SI**mple **LO**gical-Linguistic document retrieval system). In what follows, we will describe the approach taken in implementing SILOL.

1.5 An Approach

The approach generally adopted in implementing SILOL is that proposed in [van Rijsbergen 86], where a document is viewed as a set of sentences which is partially translated into a semantic representation, and like-wise, a query or request is viewed as sentences or noun phrases and similarly translated.

The retrieval strategy is to find the relationship or the similarity between the semantics of the stored documents and the semantics of a request. This process is carried out by a sort of *uncertain logical implication,* through a matching and unification process implemented in a Prolog system. There should always be a measure of uncertainty associated with such an implication since documents rarely strictly imply request. The relationships between the documents and the request are then ranked according to their similarity coefficients, and the ones with top scores are retrieved and presented to the users.

This approach is similar to the one adopted in DBMS's and question-answering systems in the sense that the answer is obtained through a process of logical satisfaction. But it is different in the sense that a request in IR systems is a *closed* sentence containing no variables, and the answer is derived from the relationships between the documents and the request. Whereas in DBMS's, a request is an *open* sentence which contains variables and the answer is an instantiation of those variables based on the semantics of the data stored.

Chapter 2: Linguistic Processing in Document Retrieval Systems

2.1 Introduction

With the advancement made in the field of Artificial Intelligence (AI), attempts have been made to use some of the AI techniques in IR work [Salton 86]. Natural language processing (NLP), or linguistic processing, is one of the areas which are under active investigation currently for the purposes of content analysis. The idea of using linguistic methods in IR is not new. In 1959, Zellig Harris had suggested the application of syntactic analysis to content analysis in information retrieval. However, up to the present date, very little research has been done in this area to be able to give a *strong* conclusion with respect to the value of using linguistic methods in IR [Fagan 87]. Furthermore, most of the work that had been carried out is on the syntactic analysis rather than on the semantic side. Some researchers have pointed out that syntactic analysis without correspondingly sophisticated semantic information may not be sufficient to provide significant improvement in content analysis [Walker 81]. But, the progress towards achieving this goal seems to be very slow. It may be due to the complexity of natural language understanding coupled with the problem of not knowing how to apply it to document retrieval systems.

In this chapter, we will discuss the problem faced with content analysis and representation, applications of NLP in document retrieval systems, and finally, introduce the application of *translation* in a document retrieval system.

2.2 Content Analysis and Representation

In conventional document retrieval systems, documents and queries are represented by an unstructured collection of simple descriptors, i.e. the keywords. This representation is not an ideal document or query content indicator for use in IR systems. Given the following titles of documents:

(1) New *curriculum* and *computer* facility for management *science* students,
(2) The undergraduate *curriculum* in *computer science,*
(3) 1989 undergraduate *computer science curriculum.*

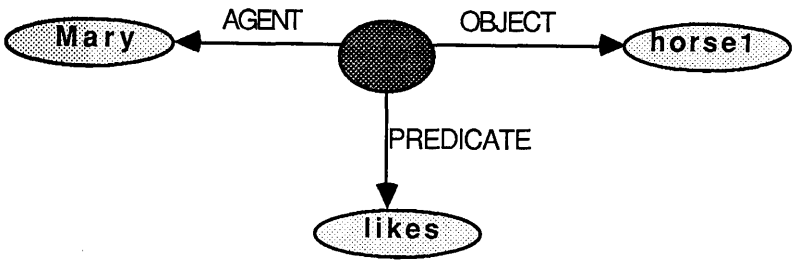
It is easy to see that the three independent terms, *curriculum, computer* and *science,* characterise all the three titles equally well. While, the phrase *computer science* is only applicable to titles (2) and (3) only. The representation of a document containing the phrase *computer science* would be more accurate if the phrase can be derived or established from the document's representation itself. This would allow a query containing the same phrase to fully match with documents like (2) and (3), but not with documents like (1). Going a step further, a good content indicator representation would allow a query with a phrase *computer science curriculum* to match documents (2) and (3) equally, but not document (1); eventhough, only document (3) has exactly the same phrase *computer science curriculum.* In order to do this the retrieval processor, in one way or another, must be provided with enough information to recognise phrases. In this particular example, a conventional document retrieval system would wrongly match the query containing the phrase *computer science curriculum* with all the three documents equally

well since the information provided by the keyword representation is not informative enough.

The example given above illustrates an obvious shortcoming of the conventional document representation models, such as the vector space model, used in most automatic document retrieval systems. In these systems, a document is represented by an unstructured collection of keywords or terms which are generally assumed to be statistically independent. The representation does not include any information on syntactic or semantic relationships among those terms. We feel that this kind of representations is too simplified to be highly effective. Thus, the major work carried out in this thesis is generally addressed towards an improvement in the method of automatic content analysis and document representation. We hold the view that a more accurate representation can be constructed if the method of content analysis takes into account information about the structure of document and query texts, i.e. the information concerning the syntactic and the semantic structure of the texts.

In order to achieve a more accurate representation of documents and queries, the simple keyword representation ought to be replaced by a *knowledge representation*. Four of the most important knowledge representation formalisms can be summarised as follows [McCalla&Cercone 83][Allen 87]:

1. Semantic Networks: A data structure consisting of nodes and links between nodes, representing concepts and their relationships. There is a set of specialised inference procedures that operate on the data structure. For example, the semantic network for "Mary likes the horse" is represented as follows:



A semantic network representation of 'Mary likes the horse'.

2. Logic: Many of the logical representations used in AI are variants or subsets of the first-order predicate calculus (FOPC). Logic was developed as a formal notation to capture the essential properties of natural language and reasoning. In FOPC, there are terms which identify objects and propositions which assert properties of objects and identify relationships between objects. Most logic-based systems used the resolution method as inference process. A sentence *Mary likes her mother* is expressed in FOPC as the proposition :

   LIKES(MARY,mother(MARY)).

3. Frame: A frame is simply a cluster of facts and objects that describe some situation, together with specific inference rules for reasoning about the situation. The situations represented could be anything such as visual scenes, structures of physical objects or methods by which some actions are performed. For example, a frame for FLIGHT in an airline system can be represented as follows:

FLIGHT:

ID: *flight_number*

Destination: *city*

Source: *city (default Glasgow)*

DepartTime: *time*

ArrTime: *time*

4. Production Systems: In these systems knowledge is represented as a set of production rules of the form :

pattern -> action.

A control loop tries each rule in turn, executing the *action* if the *pattern* matches.

The knowledge representation formalism we chose to represent documents and queries in our work is logic base, which is a subset of FOPC and will be referred to as *predicate representation*. Since a Prolog system has been used as the basis of implementing SILOL, we also use production rule formalisms in our system for defining retrieval strategies and the thesaurus.

2.3 Natural Language Processing in Document Retrieval Systems

Natural language processing (NLP) has been defined as *the formulation and investigation of computationally effective mechanisms for communication through natural language* and it is divided into two major areas [Smeaton 87] :

1) General NLP : The goal is to make models of human language, and to test these models in story understanding or dialogue

modelling systems. A general NLP system requires a large amount of real-world knowledge and, therefore, can only cover a narrow domain area.

2) Applied NLP : It is simply concerned with providing people with the facilities to communicate with machines in natural language. It doesn't matter how it is done so long as it serves the purpose.

It seems that the using of linguistic processing in document retrieval work can be classified as applied natural language processing. It doesn't matter whether the system understands its natural language input in a cognitively plausible way or not, as long as it improves the level of retrieval effectiveness !

It has been generally assumed that the application of linguistic processing could improve the level of retrieval effectiveness. Experiments to test this assumption have been carried out recently by several people, e.g. [DeFude 84], [Dillon&Gray 83], [Berrut&Palmer 86], [Croft&Lewis 87], [Fagan 87] and [Smeaton 87]. Our interest lies in the last two, the work of Fagan and Smeaton, in which the basis of our retrieval strategies can be found.

Smeaton has incorporated parsing of queries only, into retrieval strategies using the CACM collection [Fox 83] for his experiments. In summary, a user's natural language query is parsed and from the resulting parse tree *dependencies* between pairs or triples of words from the user's input are identified based on the syntactic classes of word occurrence. The words from the user's query are used to initially assign a score to each document. These document scores are then incremented if any of the identified dependencies between words in the user's query co-occur within the same sentence in the text of the

stored documents. The documents are then ranked based on this incremented score and the top documents are presented to the user. This retrieval strategy is capable of providing retrieval for the user requiring documents *about* some phrase. The grammatical coverage of the query language is restricted only to noun-phrase constructs. After a query is parsed, the parser returns a fixed-format predefined tree structure where the leaf nodes are lists of terminal symbols. A set of heuristic rules are used to derive word dependencies from these leaf nodes. The dependencies are restricted to only pairs or triples of word stem dependencies. For example, the noun-phrase **big green apple** will produce the following dependencies: **(big green), (big apple), (green apple)** and **(big green apple)**. The results obtained by Smeaton indicate that significant improvements in retrieval effectiveness can be obtained by incorporating syntactic information into document retrieval strategy.

Smeaton has only applied the automatic syntactic analysis to queries and not to document texts. The syntactic analysis should be applied both to the queries and the documents so that more informative matches between them could be achieved. Fagan in his work as reported in [Fagan 87] has applied automatic syntactic analysis to both queries and documents. He has used the PLNLP parser [Jensen 86] to parse both the document texts and the user queries of the CACM collection. The PLNLP system provides a broad-coverage syntactic grammar, a large general purpose dictionary, and facilities for manipulating the output of the syntactic analyser. Using these facilities, the texts are processed into parse trees of noun phrases where the modifiers and headnouns are combined according to certain rules to form pairs of words to be used as phrase indexing terms. These terms are incorporated with single terms to form a vector representation of documents or queries, and the retrieval strategy based on the vector

space model is used. The results obtained by Fagan have indicated a significant improvement in retrieval effectiveness.

2.4 Parsing in Document Retrieval Systems

Most of the work done in applying NLP in document retrieval systems is in the application of syntactic processing in content analysis, more specifically the use of syntactic parsing of document and query texts. Smeaton has classified this application into two principle areas. Firstly, using parsing as part of document indexing process, and secondly, using parsing as part of retrieval process. He has summarised in [Smeaton 87] some important work in these areas.

2.4.1 Parsing as Part of Document Indexing Process

Dillon and Gray [Dillon&Gray 83] and Fagan [Fagan 87] have used syntactic parsing in performing fully automatic document indexing process. In the work of Dillon and Gray, the indexing process is carried out by assigning syntactic categories to all words in an input text, extracting content-bearing terms and phrases by searching for predefined syntactic patterns, and finally grouping synonymous concepts together.

Fagan has used an existing natural language processing system PLNLP to parse document texts into parse trees of noun phrases. From these parse trees, words are combined based on the idea of constructing phrase descriptors that consist of the head of a noun phrase construct together with the head of its modifier. For examples, the phrase *curriculum in computer science* produces the following phrase

descriptors: **science curriculum** and **computer science.**

2.4.2 Parsing as Part of Retrieval Process

There are quite a number of systems that used parsing as a part of the retrieval process [Smeaton 87]. In IRES Document Retrieval System [DeFude 85], user is provided with natural language interface, and his query is parsed to recognise the important concepts and logical relationship between those concepts. In order to turn those concepts and relationships into a query representation, they are checked against the thesaurus which has been generated automatically using statistical and syntactic methods as described in [DeFude 84].

In the work by Smeaton, a user's query is parsed into parse trees which are used to generate dependencies between terms of the query using a set of heuristic rules. These syntactically identified dependencies are then used as an aid to statistically-based retrieval by searching for the co-occurrences of dependent terms within the document texts. The statistically-based retrieval performs a tf x idf (i.e. term frequency x inverse document frequency) ranking on the documents, and the top 50 documents from this ranking are reranked based on the identified dependencies.

In the work of Fagan [Fagan 87], a user's query is parsed, as documents are parsed, into parsed trees. From these parsed trees single term descriptors and phrase descriptors are derived to form the query representation or index. The phrase descriptors are formed by combining words based on head-modifier relationships. Since the system is based on the extended vector space model, thus a document or a query index is constructed of two subvectors. One contains single

term descriptors, and another contains phrase descriptors. In order to calculate the similarity between a query vector and a document vector, a partial similarity is calculated for each subvector, and the overall similarity is then calculated as a weighted sum of the two partial similarities. The similarities are calculated using a normalised tf x idf weighting. The weight of a phrase descriptors is calculated by taking the average weight of the terms involved in forming the phrase.

2.5 Translation in Document Retrieval System

Several other possibilities might be considered in improving further automatic indexing methods using linguistic processing. These can range from complex semantic analysis based on the knowledge gathered on particular domain, to simple syntactic analysis as discussed above. Both Smeaton and Fagan have used only syntactic analysis coupled with some heuristic rules to enhance representation of documents and/or queries with phrases as part of the indexes. It has been argued that in order to implement semantic analysis of texts, as in Question-Answering Systems, we need detailed semantic knowledge on a particular domain. We agree with that, but the lack of detailed knowledge about the domain must not stop us from using some of the techniques used in semantic analysis for document retrieval problems. Some semantic techniques may not require the knowledge of domain at all. An example is the technique of translating natural language to semantic representation as used in Montague's PTQ[1] or Jowsey's Simplified Montague Grammar(SMG) which can be used without detailed knowledge of the domain. These two grammars have their

[1] PTQ is an abbreviation for "The Proper Treatment of Quantification in Ordinary English", which represents Montague's efforts to apply the techniques developed within mathematical logic to the semantics of natural languages.

own semantics of representing the meanings of natural language expressions. The first is based on a high order intensional logic and the second is based on a sorted first order logic. These semantics can only be applied to document retrieval system if we know how to use them to retrieve documents in response to requests. That is the main problem we are facing today in trying to use an established grammar and its semantics in a document retrieval system.

Hence, in this thesis, we will apply a semantic translation process in SILOL by adopting an established grammar as a basis of translation technique and altering or simplifying its semantic representation to suit our purpose. The grammar chosen is Jowsey's SMG. SMG is a type of categorial grammar which uses the unification process in the parsing and translation. We have chosen this grammar because of three main reasons, namely:

1) In SMG, the semantic translation is based on the principle of compositionality, i.e. the translation of a constituent is built up by using the translations of its subconstituents [Allen 87]. Thus the semantic representation produced from the interpretation is structurally similar to the syntactic representation. The concept of compositionality is suitable for implementing partial translation where there are many unknown words and uncovered grammar constructs.

2) Since SMG parsing and translation is based on unification process, therefore, it can be implemented nicely using Prolog. Hence, we can built the whole SILOL system using Prolog, since the information representation and retrieval process are all based on logic.

3) More importantly, we have chosen this grammar in our work because it will be a first step towards the understanding of how to use a Montague-style grammar and semantics in the work of document retrieval. Montague-style semantics is a rich and powerful semantics which includes the notions of intensionality, modality, tenses and quantification. Thus, this type of grammars has lately become quite appealing to some researchers in the field of IR [van Rijsbergen 86][Frost 88].

With regards to the semantic representation, the document or query is translated into a set of first order predicates , in the form of one-place and multi-place predicates, which is used as its content indicator or index. One-place predicates are equivalent to single term descriptors in conventional system, and syntactic term dependencies can be derived from multi-place predicates. Thus, the simplest form of retrieval strategy can be based on the approach taken by Fagan and Smeaton.

Chapter 3: Montague Semantics

3.1 Introduction

Since we have chosen Jowsey's SMG as the basis of semantic translation technique in our work, in this chapter we will give an overview of it in order to illustrate how the translation process is carried out.

SMG is a simplified version of Montague's PTQ in a first order form. The translation technique adopted in these grammars is based on the principle of compositionality. Eventhough, it is a controversial subject in computational linguistic, it has the advantage that it simplifies the semantic translation process [Lewis et al 89]. The principle of compositionality has made partial translation of texts easier, and helps in building a robust parser.

We have pointed out that, ideally, the logical and linguistic components of a logical-linguistic document retrieval system should be based upon a common unified theory of logic and language. Thus, any Montague-style semantics can be considered as one of the candidates. But, for a preliminary investigation into this area of applied NLP, we will be contented with a very simple semantic theory which can be implemented quite easily and shows the potential for improving the level of retrieval effectiveness. Since we do not yet know how to use the notions of intensionality, modality, tenses and quantification in retrieving relevant documents with respect to a query, we have made some alterations and simplifications in the semantic representation of documents and queries to exclude the above notions. Nevertheless, we believe that these notions are important and should be exploited and

used in the document retrieval systems of the future. Especially in legal document retrieval systems, where the law regarding crimes already committed and the intentions of committing crimes are different and also when the 'world' or circumstances in which crimes are committed is important. But it will take sometimes for this ideal to materialise.

3.2 Montague's PTQ

PTQ is an abbreviation for "The Proper Treatment of Quantification in Ordinary English", which represents Montague's efforts to apply the techniques developed within mathematical logic to the semantics of natural languages. PTQ is used to derive the truth-conditional model-theoretic semantic interpretation for a fragment of the English language. The grammar does not give a direct model-theoretic interpretation of English. But instead, each English expression is first translated into an expression of Montague's intensional logic IL. The interpretation in IL serves indirectly as the interpretation of the English fragment.

3.2.1 Introduction to Montague's Intensional Logic (IL)

This section includes a brief discussion of some basic notions underlying IL and how an English expression is translated into an IL expression. More elaborate and comprehensive accounts of it can be found in [Dowty et al 81] and [Frost 86].

Montague's Intensional Logic (IL) is a formal system that employs a type hierarchy, higher-order quantification (i.e. variables and quantifiers for each type), lambda-abstraction for all types, tenses,

modal operators, intension and extension operators, and a model theory which is based on co-ordinate semantics. The view of the universe as regarded by IL consists of two truth values (True and False), a set of entities E, a set of possible worlds W, a set of time points T, and a function space constructed from the basic types.

Montague follows a rigorous formalised procedure in translating expressions from English into IL. The basic components of the translation procedure are as follows:

1) The natural language expression is analysed using a set of syntactic rules. A syntactic rule contains information specifying how a complex expression of a given syntactic category can be constructed from simpler components of given syntactic categories. The output from this analysis is one or more parse trees.

2) Every syntactic rule has a translation rule associated with it. A translation rule specifies the translation of the output from a syntactic rule in terms of its inputs. Thus, the translation rules specify how to translate the syntax trees, produced in (1), to expressions in IL. In order to avoid the problem of ambiguity, Montague has imposed that each English word or basic expression is translated into only one IL expression.

The above translation procedure has ensured that the translation obeys the principle of compositionality, i.e. the translation of the whole English expression is determined by the translation of its parts and the syntactic rules used in forming it. In the following section, we will discuss in more details the translation process based on the PTQ approach carried out in Jowsey's SMG.

3.3 Jowsey's SMG

Jowsey in [Jowsey 87] has used a simplified version of Montague Grammar (SMG) as a front end to a general reasoning system. Instead of using a higher-order logic in translation, a five-sorted extensional first order logic is chosen and a standard theorem proving technique such as resolution is used.

3.3.1 The Theory of Montague Semantics: A five sorted extensional first order logic

In place of Montague's higher-order Intensional Logic IL, used in Montague's PTQ, Jowsey has used his own Theory of Montague Semantics(TMS) in defining the semantics for SMG. TMS is a five sorted extensional first-order logic which is capable of translating every sentence of the English fragment in PTQ into a first-order expression. The *five sorts* used in TMS are as follows:

1. e - the sort *entity*.
2. s - the sort *time-index* which refers to a point in time; the relation '<' denotes temporal anteriority, e.g. I<J, where I and J are of sort s, denotes that I is earlier than J; the symbol ! is used to denotes the point in time *now*.
3. o - the sort *proposition;* for this sort, a truth predicate **true** is defined to relate objects of type o to indices, e.g. true(U,I) expresses the fact that the proposition U is true at the index I.
4. p - the sort *property;* for this sort, a function ' is defined to express propositions relating the relationships between properties and entities, e.g. V'X expresses the proposition

that the entity X has the property V.

5. q - the sort *quantity* which is defined as a property of properties; for this sort, a function " is defined to express propositions relating the relationships between quantities and properties, e.g. W"V expresses the proposition that the property V has the quantity W.

The logical connectives, quantifiers and predicates used in TMS are as defined below, where a and b are well-formed expressions of TMS:

~ α	- it is not the case that α
α & β	- both α and β
α v β	- either α or β or both
α => β	- if α then β
α <=> β	- α if and only if β
α = β	- α is equivalent to β
all(X:x, α)	- for all X of sort x such that α
exists(X:x, α)	- there exists at least one X of sort x such that α
anypredicate(X,Y,...,Z,I)	- the predicate anypredicate with arguments X,Y,...,Z of some sorts and index I.

3.3.2 Syntactic Categories and Types

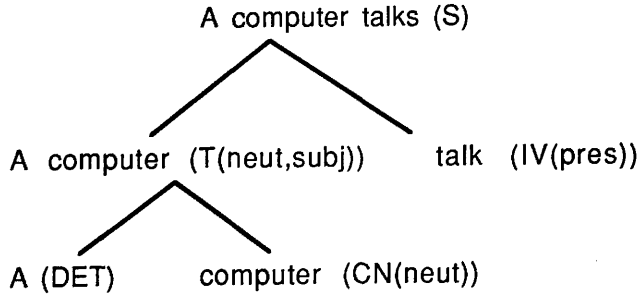
Following Montague's PTQ, the categories of the English fragment covered by SMG are given by the set Cat:

Cat = {S,AS,AUX,T,DET,ACN,CN,IV,TV,SV,IVV,IAV,TAV}

The elements of Cat are symbols representing the following English categories: sentences(S), sentence adverbs(AS), auxiliary verbs(AUX), noun phrases(T), determiners(DET), adjectives(ACN), common nouns(CN), verb phrases(IV), transitive verbs(TV), sentence complement verbs(SV), infinitive complement verbs(IVV), adverb phrases(IAV) and prepositions(TAV).

In order to handle *tenses, genders* and *cases* embedded in English grammar, three of the categories defined in Cat are given additional feature specifications. These three categories are noun phrases(T), common nouns(CN) and verb phrases(IV). The category T is given two types of feature specifications, Gender and Case. The category CN is given only Gender feature specification. Gender can take the value of masculine(masc), feminine(fem) or neuter(neut), and Case can take the value of subject(subj) or object(obj). These feature specifications can be used in resolving anaphora references of pronouns such as he, him, she, her, it, etc. The category IV is given one type of feature specification, Tense, which may take the value of base, present(pres), or past. Thus, in a sentence 'A computer talks', the categories assigned to the components of this sentence is as shown in derivation tree Tree_1.

(Tree_1): Note: the symbols in the parentheses are syntactic categories of the expressions. Feature specifications may not be shown in later trees.



The syntactic categories S, CN, and IV are defined as basic syntactic categories. All other complex categories can be defined in terms of these basic categories based on the following definition:

1. S, CN and IV are syntactic categories.
2. If A and B are any syntactic categories, then A/B is a syntactic category.

The '/' notation is based on the categorial grammar developed by Ajdukiewicz [Ajdukiewicz 35]. An expression of category A/B when combined with an expression of category B gives an expression of category A. Thus, the notation suggests a kind of algebraic cancellation as in the division-multiplication operation of (A/B . B = A). This operation is known as the rule of functional application. Following are some examples of complex categories defined in terms of the basic categories: T = S/IV, DET = T/CN, and TV = IV/T.

Every English expression of a particular syntactic category is translated into a semantic expression of a corresponding type. Types are defined as follows:

1) e is a type, representing objects of sort entity.
2) t is a type, representing truth values.
3) If a and b are types then <a,b> is a type, representing a class of functions from objects of type a to objects of type b.
4) If a is a type then <s,a> is a type representing functions from time indexes to objects of type a.

The correspondence between syntactic categories and semantic types can be defined using a function *type* which maps syntactic categories into semantic types as follows:

1. *type*(S) = <s,t>
2. *type*(CN) = *type*(IV) = <e,<s,t>>
3. For all categories A and B, *type*(A/B) = <*type*(B),*type*(A)>.

For example, the corresponding type for category T is derived as follows:

$$\begin{aligned} type(T) &= type(S/IV) \\ &= < type(IV), type(S) > \\ &= <<e,<s,t>>,<s,t>> \end{aligned}$$

3.3.3 Translating Basic English Expressions

Every word, or basic expression, used in the fragment has to be defined and given a semantic representation. A collection of these basic expressions will serve as a semantic dictionary or lexicon which represents a collection of semantic representations of the words used in the fragment. Each basic expression is translated into a semantic representation in Prolog list notation, called a template, of an appropriate type depending on its syntactic category. The template is written in the form [input | output]. The output part is the actual first order translation of the expression, whilst, the input part defines the input variables to the expression. For example, the basic expression 'horse' of a basic category CN is translated into [X,I | horse(X,I)] of type <e,<s,t>>, which is equivalent to the *intension* of a 'horse' in PTQ. If the expression is not a basic category then the input part also defines the pattern of the template that the expression can be combined with. For example, the adjective 'red' of category ACN, which is defined in terms of the basic categories as CN/CN, is translated into

[[Y,J | A],Y,J | red(Y,J) & A]

where the head of its input part, i.e. [Y,J | A], conforms to the pattern of the CN template that it can be combined with. Thus in the '/' notation the template for 'red' is written as

[Y,J | red(Y,J) & A]/[Y,J | A]

where [Y,J | A] is known as its active part. Basic expressions are combined to form complex expressions through unification processes. There are several ways of combining expressions, the main one used in categorial grammar is the rule of functional application which consists of two steps as follows [Zeevat et al 87]:

1) Instantiating of the active part with the template of the expression to be combined with. For example, in the combination of the basic expressions 'red' with 'horse', the instantiating process of the two templates can be illustrated as follows (the components to be instantiated are written in italics):

[*[Y,J | A]*,Y,J | red(Y,J) & A]] instantiated with *[X,I | horse(X,I)]*

becomes

[*[X,I | horse(X,I)]*,X,I | red(X,I) & horse(X,I)]

where Y is unified with X, J with I and A with horse(X,I).

2) Stripping of the active part. In the example above, the template obtained after the instantiating process is stripped of its instantiated active part *[X,I | horse(X,I)]* becomes:

[X,I | red(X,I) & horse(X,I)].

This unification process will be discussed further in the following section.

The SMG translations of some basic expressions into template representations are given in Table_1.

Word	Category (its equivalent)	Semantic
computer	CN	[X,I|computer(X,I)]
red	ACN(CN/CN)	[[X,I|A],X,I|red(X,I) & A]
a	DET(T/CN)	[[X,I|A],[X,I|B],I|exists(X:e,A & B)]
John	T(S/IV)	[[j|A]|A]
talk	IV	[X,I|talk(X,I)]
catch	TV(IV/T)	[[[Y,I|catch(X,Y,I),I,A],X,I|A]
believe	SV(IV/S)	[[J|A],X,I|exists(U:o, believe(X,U,I) & all(J:s, true(U,J) <=> A))]
rapidly	IAV(IV/IV)	[[Y,J|A],X,I|exists(V:p, rapid(X,V,I) & all(Y:e, all(J:s, true(V'Y,J) <=> A))]
conceive	TV(IV/T)	[[[Y,J|true(V'Y,J)],J|A],X,I|exists(W:q, conceive(X,W,I) & all(V:p, all(J:s, true(W"V,J) <=> A)))]

Table_3.1: Translation of Some Basic English expressions.

3.3.4 Syntactic and Translation Rules

The syntactic and translation rules of SMG are equivalent to the rules defined in PTQ. Each rule is given its implementation rule in

Prolog which is written in the following form:

fn >> $syntax_1$:$semantics_1$ + . . . + $syntax_n$:$semantics_n$ => $syntax_r$:$semantics_r$

[<- predicate(arg_1,arg_2, . . . ,arg_n)]

where:

fn - is the label given to the rule.

>> - is the delimeter between the label and the rule.

$syntax_i$:$semantics_i$ - is the syntactic category and the semantic form of the i^{th} component to be combined, to form the resultant expression $syntax_r$:$semantics_r$

+ - is the combination symbol

[. . .] - is a meta symbol containing optional predicate to be invoked when applying the rule.

Through these implementation rules basic English expressions are combined to form complex expressions, and at the same time translated into TMS expressions using the Prolog unification process. The implementation rule for the determiner-noun construct, which states that a determiner(DET) can be combined with a common noun(CN) to form a noun phrase(T), is written as follows[2]:

f2 >> det:[P | Q] + cn(Gend):P => t(Gend,Case):Q

[2]Lower case letters are used for syntactic categories in writing implementation rules, because upper case letters are used for variables in Prolog.

Instead of using higher-order beta-reduction with lambda-expressions as in Montague's PTQ, the translation is implemented by using first-order template unification process. Unifications are activated by the implementation rules which specify the correct combinations of two or more categories to form a resultant category.

Through the semantic forms in the rule, $semantics_i$, one can specify how unification has to be carried out to produce the resultant category. There follows an example of how the templates of a determiner 'a' and a common noun 'computer' are unified using the rule of functional application.

The determiner-noun rule is :

f2 >> det:[P | Q] + cn(Gend):P => t(Gend,Case):Q

This rule specifies that the semantic template [P | Q] of the determiner is to be instantiated with respect to the semantic template P of the common noun, thus, all the variables with the same names will be unified. Then the instantiated template [P | Q] is to be stripped of its active part P to produce the semantic template Q for the resultant expression.

Given the templates for determiner 'a' and common noun 'computer' respectively as follows:

[[X,I | A],[X,I | B],I | exists(X:e,A&B)]
[Z,J | computer(Z,J)]

and based on the determiner-noun rule, the above two templates are

then combined, i.e. instantiated and stripped, as shown below to produce the resultant template:

The active part of the determiner 'a' template is to be instantiated with the template of the common noun 'computer', both are written in italics:

[X,I | A],[X,I | B],I | exists(X:e,A & B)]
[Z,J | computer(Z,J)]

The variable X is unified with Z, I with J and A with computer(Z,J) to produce the following instantiated template of the determiner:

[[Z,J | computer(Z,J)],[Z,J | B],J | exists(Z:e,computer(Z,J) & B)]

The instantiated template is then stripped of its active part to become:

[[Z,J | B],J | exists(Z:e,computer(Z,J)) & B]

which is the template for the noun phrase 'a computer'.

Templates can also be combined and unified by relativisation processes [Jowsey 87]. For example, a common noun 'computer' template [X,I | computer(X,I)] can be relativised with a sentence 'it runs' template [J | run(Y,J)] to produce a resultant template:

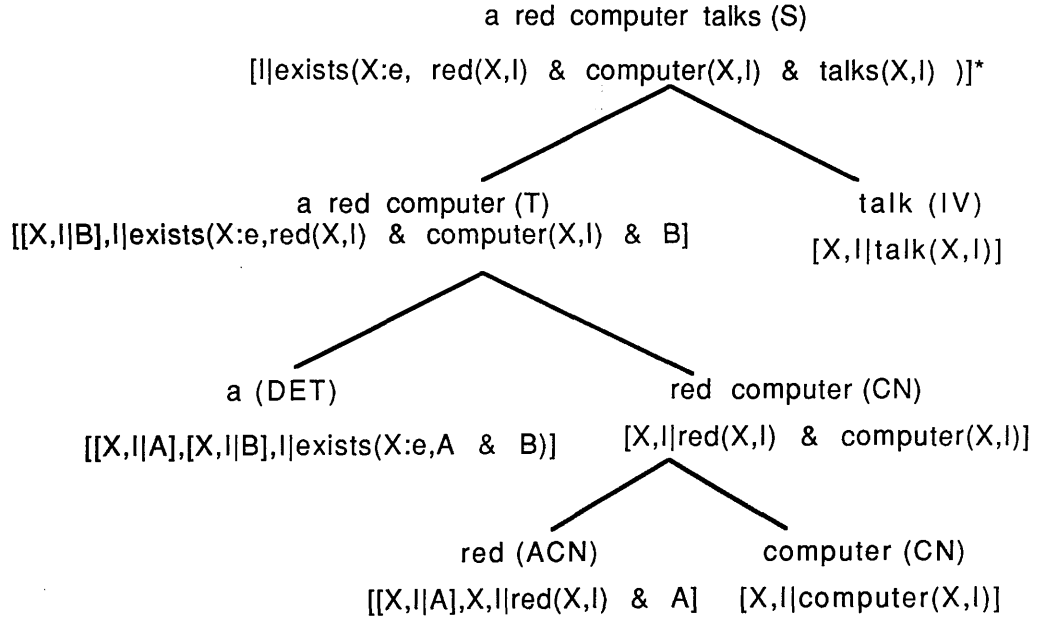
[Y,J | computer(Y,J) & run(Y,J)]

where X is unified with Y and I with J.

Below are examples, illustrated by derivation trees, of how English phrases or sentences are translated into TMS expressions:

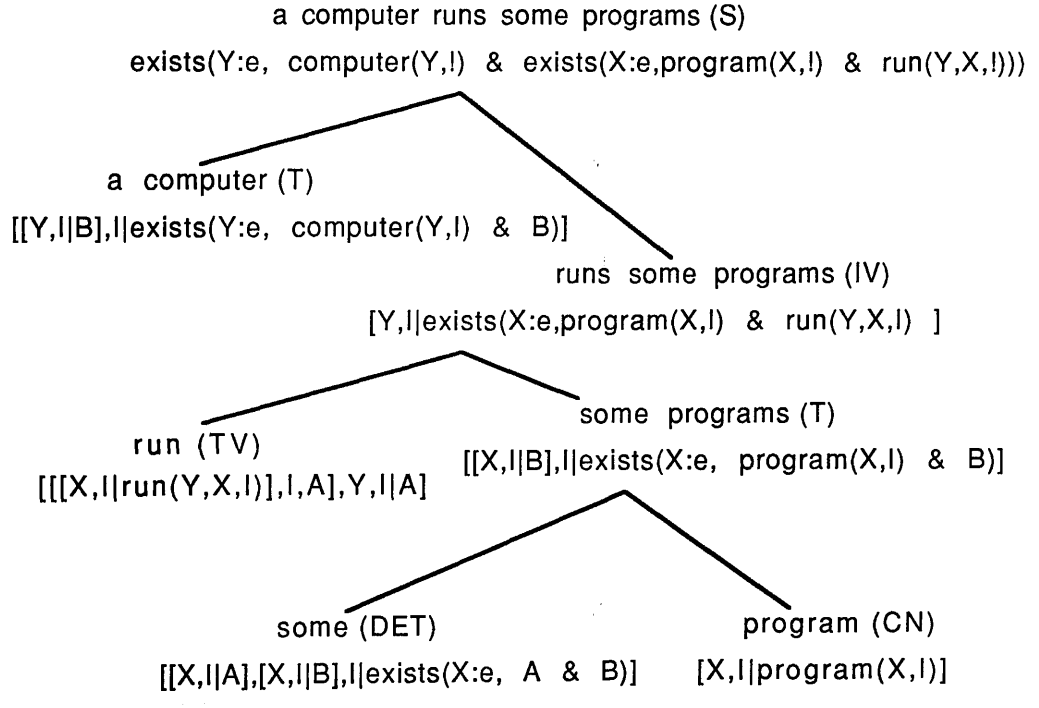
Example_1: 'A red computer talks' is translated into

exists(X:e, red(X,!) & computer(X,!) & talk(X,!)).



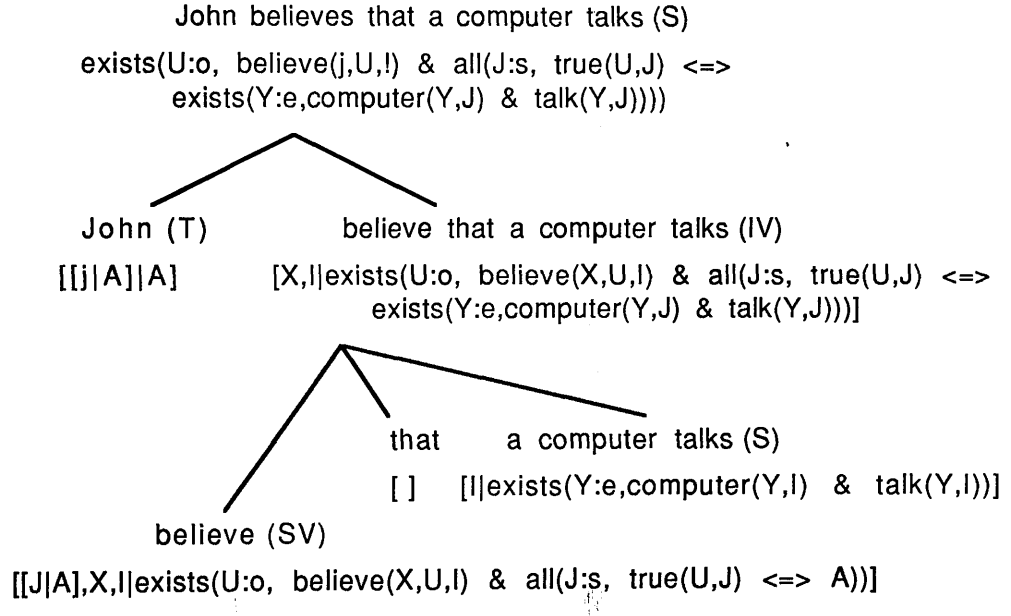
**The variables I's in the final template are then unified with '!', and then it is stripped to produce the expression* exists(X:e, computer(X,!) & red(X,!) & talk(X,!)).

Example_2: 'A computer runs some programs' is translated into
exists(Y:e, computer(Y,!) & exists(X:e, program(X,!) & run(Y,X,!))).



Example_3: 'John believes that a computer talks' is translated into

exists(U:o, believe(j,U,!) & all(J:s, true(U,J) <=>
exists(Y:e, computer(Y,J) & talk(Y,J)))).



Chapter 4: A Semantic Translation for SILOL

4.1 Introduction

A complete translation of unrestricted queries and the whole texts of documents need a wide coverage of English grammar and lexicon. This seems to be impractical as far as our work is concerned. Thus we will adopt a partial translation strategy which is based on a limited noun phrase grammar and lexicon which are built solely for experimental purposes. The translation of stored documents will only be done partially based on the limited grammar and lexicon, and the queries are restricted to noun phrase form. This restriction on the query should not be very constraining to the user, since the objective of a natural language query is to describe some *thing* in which the user is interested and this can be done using noun phrases. Some evidence has been gathered to support this view [Waldstein 1981]. However, the queries are allowed to be more than noun phrases, but it will only be partially translated based on the limited grammar and lexicon.

Another point to consider, as far as application to document retrieval system is concerned, is how accurate or how detailed the translation should be. We are not dealing with question-answering system where the translation should be as close as possible to the *real* meaning of the natural language phrases, in order to give an *exact* answer to a question. In the case of document retrieval systems we are only trying to retrieve a set of *most* relevant documents for a query. Hence, in one way or another, we are performing a plausible matching between a given query and the stored documents. Thus, the aim of the translation in this case is to produce good content indicators or indexes for documents and queries, and not their exact meaning. This means

that a simplified version of any semantic theory chosen may be adequate for our application, and the extent of simplification may depend upon the basis of retrieval strategies that are going to be used. However, as the semantic theory is refined and improved the approach to retrieval may remain the same.

In this chapter we will discuss the *generalised-relationship* concept on which we have based our translation. Based on this concept we have built a limited categorial unification grammar a la Jowsey's SMG. The foundation of the grammar and the natural language constructs implemented will be discussed in the following sections.

4.2 Generalised-relationship concept

Jowsey's TMS is able to translate the whole fragment of English given in Montague's PTQ into its first order form, which shows the richness and power of its features and capability. However, even if we are able to adopt all its features, we still do not know how to use it for the purpose of retrieving relevant documents in response to a query. Hence, to make things simpler to meet our purpose, we have left out the notion of *sorts* and *tenses*. Also we have based our semantic representation on, what we call, the *generalised-relationship* concept. Which means, the dependencies or relationships between words are represented explicitly by predicates **a**, **r**, **vso**, **sv**, **ov** and **p**, which respectively stand for the following relationships: **adjective-noun**, **noun-noun**, **verb-subject-object**, **subject-verb**, **object-verb** and **preposition-nouns**. In the case of the preposition-nouns predicate **p**, we have broken it down into its individual prepositional predicates, i.e. **in**, **inside**, **of**, **on**, **for**, **by**, **with**, **within**, **without**, and **using**. The approach adopted here is related to the approach taken by case grammar which

outlines the range of semantic roles, called cases, that a noun phrase may play when used along with a verb or adjective [Allen 87]. This representation is adopted in order to suit the basis of the retrieval strategies which is based on the dependency approach that will be used in our system. In this way the retrieval process can be implemented easily and efficiently.

4.3 Simple Unification Noun-Phrase Grammar (SUNG)

Based on our generalised-relationship concept, we have built a restricted grammar which will be referred to as a **S**imple **U**nification **N**oun-Phrase **G**rammar (SUNG). The parser-translator of SUNG is written in C-Prolog. Although SUNG is defined as a restricted noun phrase grammar, it includes a simple sentence construct and is capable of accepting any unrestricted natural language input and performing a partial translation on it. The syntax of SUNG can be summarised by the following production rules expressed in BNF notation where the symbols in [...] are optional:

nounphrase	**::= [determiner] [modifiers] heads [postmodifier]**
heads	**::= head | conjunction heads**
head	**::= commonnouns**
modifiers	**::= modifier modifiers | conjunction modifiers**
modifier	**::=adjectives | verbs(past participles) | verbs(present participles) | commonnouns**
postmodifier	**::= relativeclauses | verbbyphrases**
sentence	**::= subject verb object**
subject	**::= nounphrase**
object	**::= nounphrase**

The actual syntax of SUNG is defined by the grammar given in Appendix_A.

4.3.1 Syntactic Categories and Types

The syntactic categories defined in SUNG are given in the set CATs:

CATs = { S, T, DET, ACN, ACN2, CN, IV, TV, TAV}

The elements of CATs are symbols representing the following English categories: sentences(S), noun phrases(T), determiners(DET), adjectives(ACN), postmodifiers(ACN2), common nouns(CN), verb phrases(IV), transitive verbs(TV), and prepositions(TAV).

4.3.2 Translation of Basic English Expressions

Generally, the translation of basic expressions or English words into semantic templates is based on their syntactic categories as follows:

Categories	Template Forms
CN	[X | predicate(X)]
IV	[S | predicate(V) & sv(V,S)]
ACN	[[Y | A],Y | predicate(X) & A & a(X,Y)]
TV	[[[O | predicate(V) & vso(V,S,O)] | A],S | A]
DET	[[X | A],[X | B] | A appropriate-operator B]
TAV	[[[X | predicate(Y,X)] | A],[Y | C],Y | A & C]

where :

'predicate' is the English word of the basic expression,

'a' stands for the adjective-noun relationship,

'vso' stands for the verb-subject-object relationship,

'sv' stands for the subject-verb relationship,

'appropriate-operator' may be & or =>.

Following are some examples of the translations:

Words	Template Forms
computer	[X | computer(X)]
run	[X | run(Y) & sv(Y,X)]
fast	[[Y | A],Y | fast(X) & A & a(X,Y)]
punch	[[[O | punch(V) & vso(V,S,O)] | A],S | A]
a	[[X | A],[X | B] | A & B]
in	[[[X | in(Y,X)] | A],[Y | C],Y | A & C]

Our experimental lexicon of basic expressions is built based on the dictionary created and used by Smeaton in his experiments [Smeaton 87] which roughly consists of 600 different words.

4.3.3 Translation of Complex Expressions

The grammar rules which carry out the translation of complex expressions are given in Appendix_A. In this section we will discuss how the translation of the complex expressions is being carried out in SILOL based on the SUNG.

4.3.3.1 Adjective-Noun Compounds

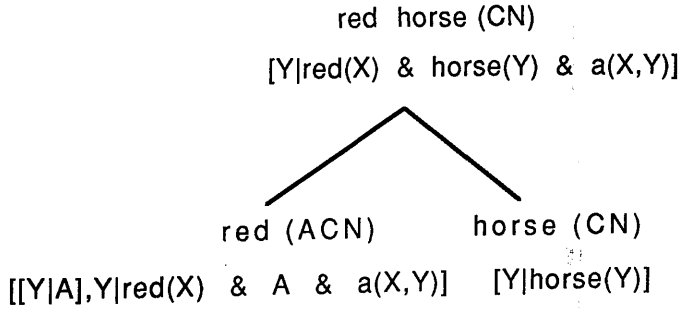
Keenan and Faltz [Keenan&Faltz 85] have made a distinction between adjectives that introduce a restriction on the noun and those that do not. The restrictive and non-restrictive adjectives may be subcategorised further. For example, restrictive adjectives are

subcategorised into intersective and subsectives adjectives. Semantically, different classes of adjectives should have different representations. For example, Saint-Dizier [Saint-Dizier 86] translates intersective and subsective adjectives differently, and so does Jowsey [Jowsey 87]. Jowsey has translated the phrase *red horse*, where red is an intersective adjective, into *red(X,I) & horse(X,I)*. The phrase *tall horse*, where tall is a subsective adjective, is translated into the following complex expression:

exists(X1:e,exists(V2:p,tall(X1,V2,!)&all(I4:s,true(V2'X3,I4)<=>horse(X3,I4)))).

But for our application, we generalise the translation of any adjective into just one form, that is, a *generalised-relationship* form represented by a two-place predicate **a**. For example given a phrase *red horse*, its translation will read as *X is red and Y is horse and there is a relationship **a** between X and Y*, and in terms of first order predicate formulae it is written as: *red(X) & horse(Y) & a(X,Y)*. This translation is illustrated in derivation tree Tree_4.1.

(Tree_4.1):



Likewise, those past participles which act as adjectives are treated similarly. For example *poisoned food* is translated into: poison(X) & food(Y) & a(X,Y). In fact, we have based all our translation on this

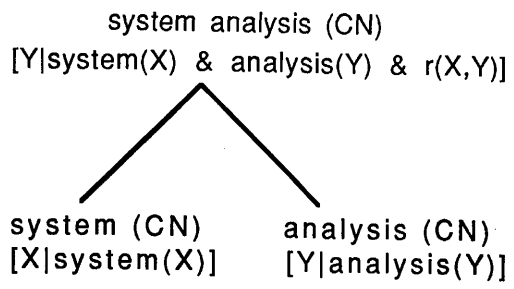
generalised-relationship form throughout our work. In this form, the relationships between words or one-placed predicates are explicitly expressed by multi-place predicates. For example, the relationship between the above predicates poison(X) and food(Y) is explicitly expressed by the predicates a(X,Y). Thus, one-placed predicates are equivalent to single term descriptors, and multi-place predicates are equivalent to term-dependency pairs or triples. With this form, we can devise retrieval strategies based on the dependency approach as performed by Smeaton and Fagan more easily and efficiently than Jowsey's form of 'poison(X) & food(X)'.

4.3.3.2 Noun-noun Compounds

Noun-noun Compound construct has not been discussed in Jowsey's work [Jowsey 87], but Dowty [Dowty 79] has given several translations of them based on the writings of several authors such as Bradley [Bradley 06], Levi [Levi 75] and Downing [Downing 77].

Using our generalised-relationship form, the noun-noun compound phrase *system analysis* is translated into: *system(X) & analysis(Y) & r(X,Y)*, as illustrated in the derivation tree Tree_4.2. This translation is almost similar to that of Bradley.

(Tree_4.2):



4.3.3.3 Postmodifiers

There are many types of postmodifiers commonly used to describe the head noun in noun phrases. We have considered only a few of them, and below we describe how they are treated with respect to the framework of the SMG implementation rules and our application.

4.3.3.3.1 Relative Clauses

There are two types of relative clauses that interest us, they are such-that and wh relative clauses.

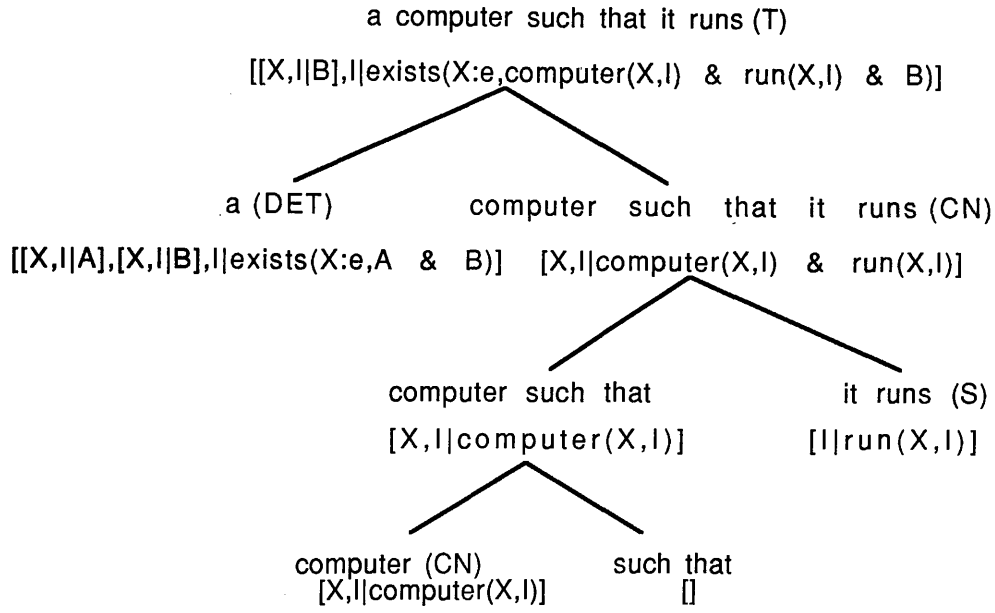
4.3.3.3.1.1 Such-that relative clauses

Rules for handling such-that relative clauses are given in Montague's PTQ [Montague 73]. A such-that relative clause is a restrictive postmodifier of a common noun. It derives a predicate from the sentence modifier in the given phrase in order to intersect the noun which it modifies. For example, the phrase *a computer such that it runs* is translated by Jowsey, following Montague, into:

exists(X:e, computer(X,!) & run(X,!))

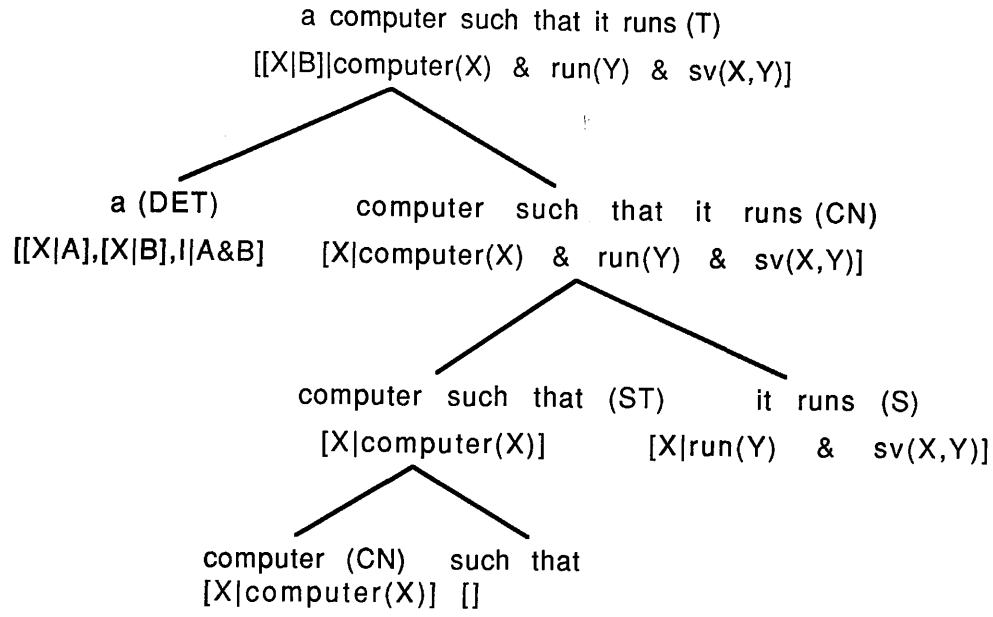
as illustrated in derivation tree Tree_4.3.

(Tree_4.3):



Using our generalised-relationship approach, the translation of the phrase *a computer such that it runs* is *computer(X) & run(Y) & sv(X,Y)* as illustrated by derivation tree Tree_4.4.

(Tree_4.4):

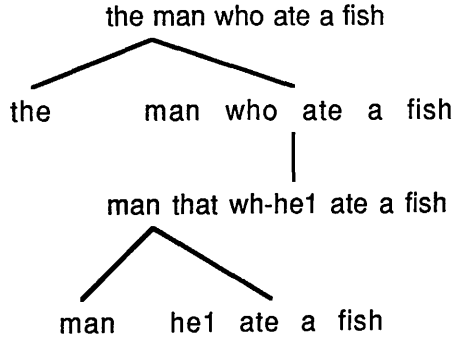


Instead of having only one predicate *run(X)* to translate the sentence *it runs*, we have used a conjunction of two predicates *run(Y) & sv(X,Y)*. This approach is taken in order to explicitly express the relationships between subjects and verbs. In this way, the implementation of retrieval strategies which are based on the term-dependency approach will be implemented more efficiently. This approach is also adopted in translating simple sentences having subjects, verbs and objects. For example, the sentence *computer needs a brain* is translated into *computer(X) & need(Y) & brain(Z) & vso(Y,X,Z)*, where the predicate *vso* denotes the relationship between the verb Y, the subject X and the object Z. Likewise, the relationship between a verb and its object is represented by a predicate *ov*. The predicate ov is only used when the parser-translator fails to find the subject for a transitive verb.

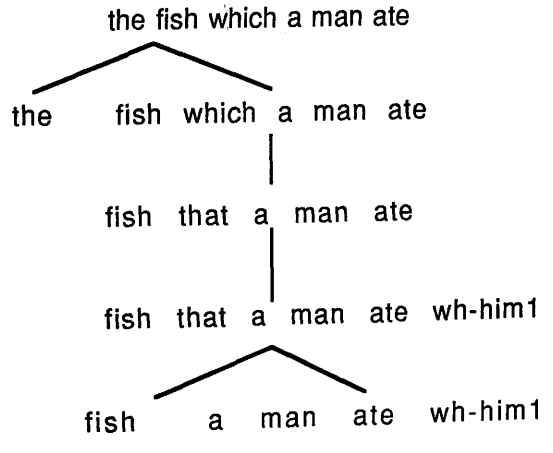
4.3.3.3.1.2 Wh relative clauses

Although Montague's semantic treatment of such-that relative clauses is accepted as adequate, its syntax is not considered to be so [Rodman 76]. The such-that form of relative clauses is only popularly used in Mathematical notation and it is not ordinary English usage. Rodman has given syntactic and semantic rules which are necessary for the production of non-restrictive relative clauses, known as wh-clauses. Derivation trees Tree_4.5 and Tree_4.6 illustrate Rodman's way of parsing the wh-clauses:

(Tree_4.5):



(Tree_4.6):

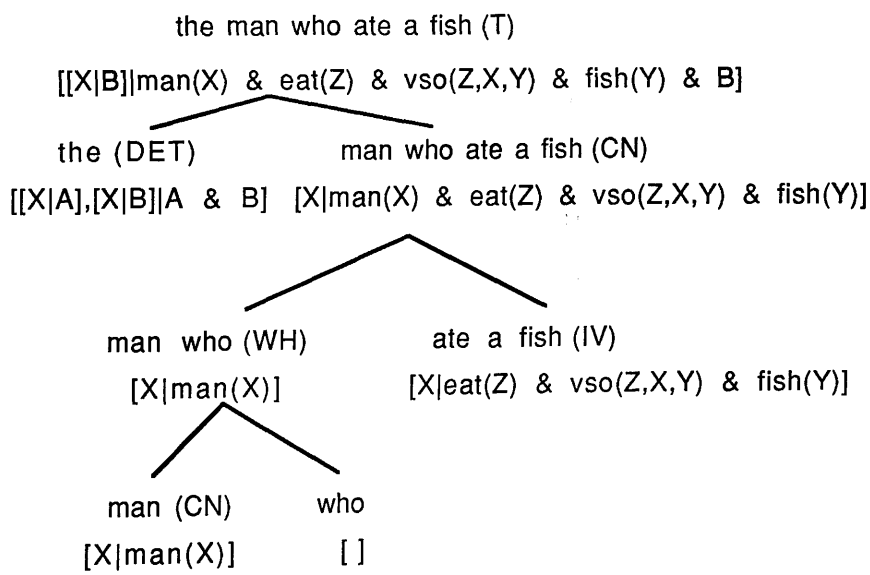


Looking at derivation trees Tree_4.5 and Tree_4.6, it is easy to see

how it works from the bottom up, that is, how to produce a wh-clause from its components. But to parse a wh-relative clause into its components is not trivial, because of the introduction of a dummy pronoun, wh-him1 in the above example, at the appropriate place somewhere in the middle of the clause. So, his treatment is difficult to implement using the SMG implementation rule.

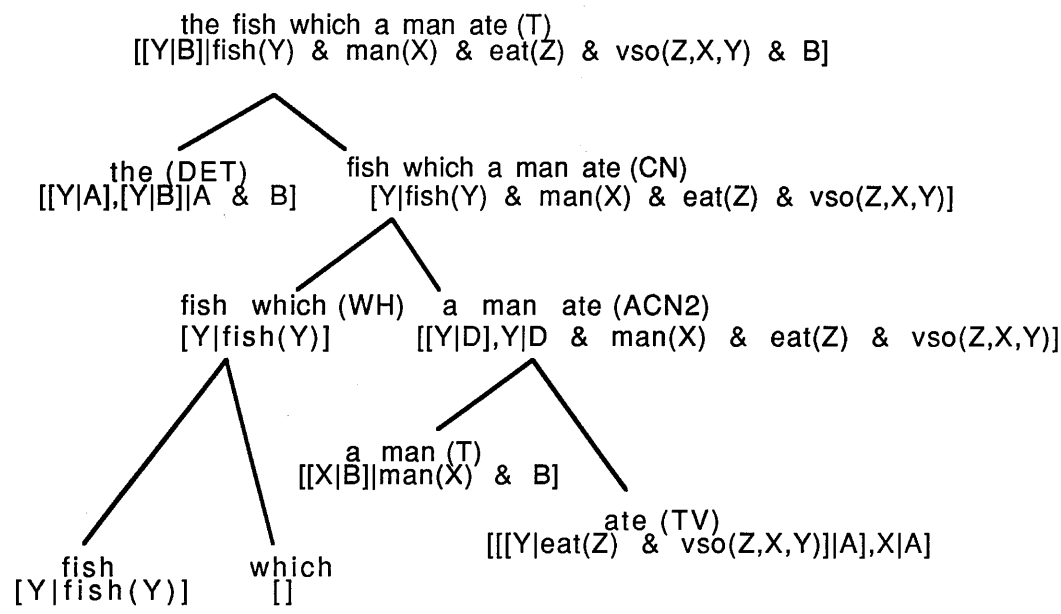
We introduce our own treatment of wh-relative clauses which can be implemented simply by defining a set of new rules. To illustrate our treatment, consider the same sentence as in Rodman's examples *the man who ate a fish,* this same sentence can be equally derived as shown in derivation tree Tree_4.7 below:

(Tree_4.7):



Now, consider the following sentence *the fish which the man ate.* This sentence can be derived as shown in the following derivation tree Tree_4.8:

(Tree_4.8):

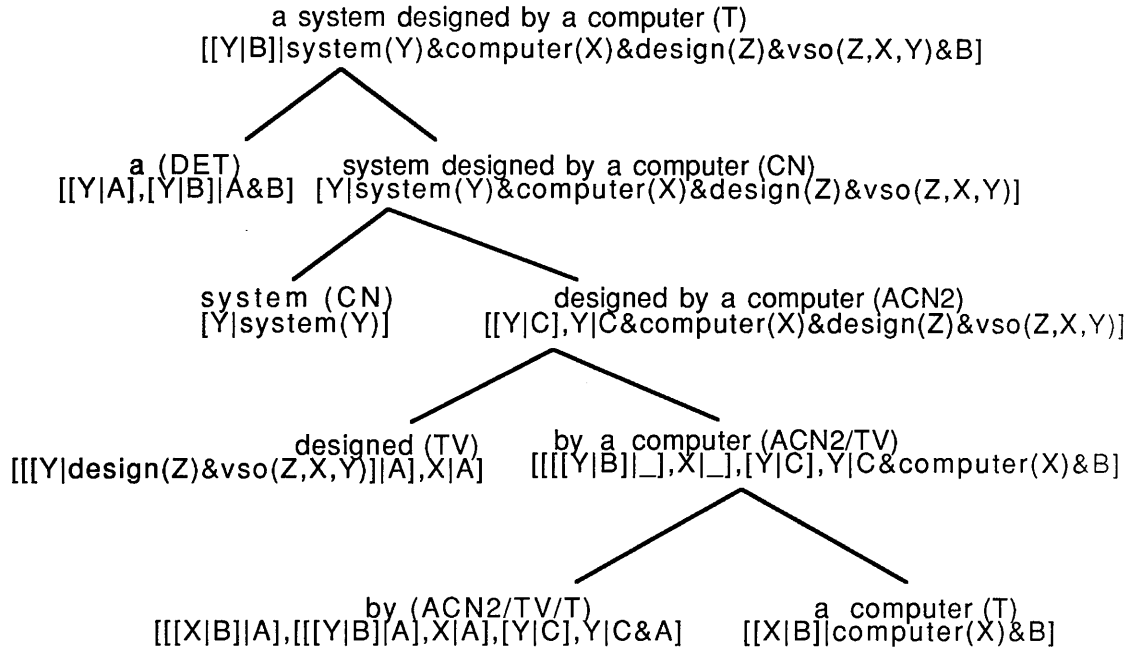


In both cases, the derivations do not involve introduction of a dummy pronoun wh-he. Thus, this treatment of wh-clauses is more suitable to the framework of the SMG implementation rule. One of the new categories introduced is ACN2 which represents postmodifiers. A similar method is used in [Thomason 76] in handling the preposition *of*. In handling the preposition *by*, which will be discussed in the following section, the same category ACN2 will be used.

4.3.3.3.2 Verb-by Phrase Postmodifier

A verb combined with a preposition *by* can form a verb phrase postmodifier which can be categorised under ACN2. Consider the sentence *a system designed by a computer*. This sentence can be derived as shown in derivation tree Tree_4.9:

(Tree_4.9):



4.3.3.4 Conjunctions

Even in a restricted noun phrase grammar, conjunctions are found to be very complicated. Grammatically, conjunctions can occur almost anywhere, either within an individual noun phrase or between noun phrases. The way we handle conjunctions is by using the equivalence predicate '=' as illustrated by the examples below.

1. Common noun conjunctions.

Phrases	Translations
'computer and man'	[Z | (computer(X) & =(Z,X)) & (man(Y) & =(Z,Y))]
'computer or man'	[Z | (computer(X) & =(Z,X)) v (man(Y) & =(Z,Y))]

2. Term conjunctions.

Phrases	Translations
'a computer or a man'	[[Z | C] | (computer(X) & =(Z,X)) v (man(Y) & =(Z,Y)) & C]
'a computer and a man'	[[Z | C] | (computer(X) & =(Z,X)) & (man(Y) & =(Z,Y)) & C]

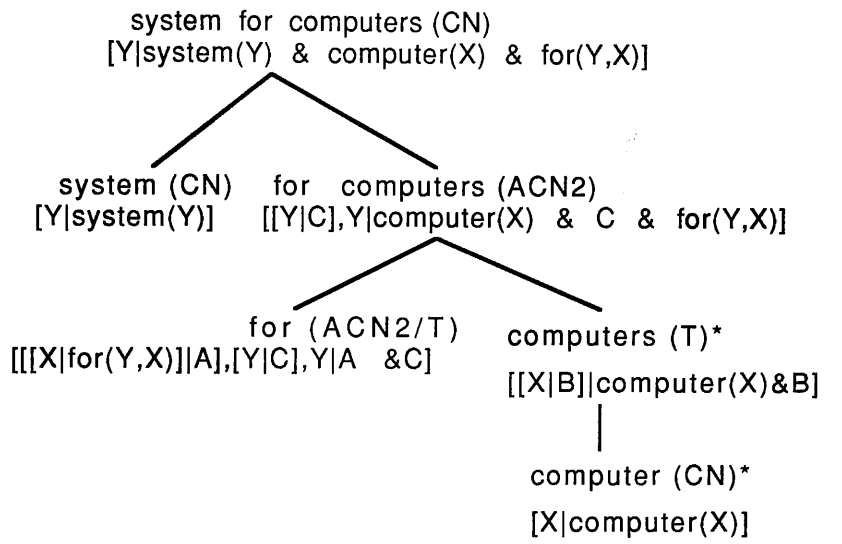
3. Modifier conjunctions.

Phrases	Translations
'sequential and fast'	[[W | C],W | ((sequential(X) & =(Z,X)) & (fast(Y) & =(Z,Y))) & a(Z,W) & C]
'sequential or fast'	[[W | C],W | ((sequential(X) & =(Z,X)) v (fast(Y) & =(Z,Y))) & a(Z,W) & C]

4.3.3.5 Prepositions

Generally, prepositions used between noun phrases are classified under category ACN2/T. For example, the phrase *system for computers* is translated as shown in the following derivation tree Tree_4.10:

(Tree_4.10):



**This is a case of an invisible determiner where a CN is raised automatically to T in order to be able to combine with its adjacent component of category ACN2/T.*

The introduction of a predicate **for(X,Y)** is to establish the relationship between **X** and **Y**. This is the general treatment applied to all other prepositions implemented in SILOL.

Chapter 5: Parsing and Translation Strategy

5.1 Introduction

In this chapter, the parsing algorithm used in implementing the SUNG will be discussed. SUNG is a type of categorial unification grammar based on Jowsey's SMG where the parsing algorithm is implemented using the Prolog unification mechanism. But SMG is non-robust in nature, in the sense that it cannot process ill-formed input texts. There are many reasons why an input may be considered ill-formed [Salton 89], for example, it is ill-formed if it contains words which are not in the lexicon or its structure is beyond the scope covered by the grammar implemented. SUNG adopts a strategy of robust partial translation to handle all forms of input texts by first locating sequences of known words and then trying to parse each of them into a shortest possible string. We will call this approach *island focusing*.

Dealing with natural language processing, one cannot run away from ambiguity. Ambiguity may be due to a word having many meanings or a phrase having many parse trees. In this chapter we will also discuss how SUNG tries to minimise ambiguities by adopting a multi-pass parsing strategy.

5.2 Unification Grammar

Unification has been used extensively in the parsing techniques of categorial grammar due to the combination of theoretical and practical considerations. The theoretical consideration is to integrate semantics as tightly as possible with syntax, and the practical consideration is to

develop a theory which could be implemented as an efficient parser [Zeevat et al 87].

Classical categorial grammar consists of a set of rules for the grammar, a set of categories with a list of basic expressions under each category, and the functional application rule. For example, given that *noun(N)* and *predicate(P)* are basic categories in a categorial grammar, then the definition of categories in this grammar is as follows :

a) N and P are categories,

b) If A and B are categories, then A | B is a category. B is known as the active part of A | B.

The rule of functional application is defined as follows:

If E_1 is an expression of category A | B and E_2 is an expression of category B, then E_1E_2 is an expression of category A.

The application of the grammar rules and the basic expressions under functional application rule will generate a set of complex expressions.

Various extensions to the classical categorial grammar have been made in natural language processing applications. Montague PTQ, Unification Categorial Grammar (UCG) [Zeevat et al 87] and Simplified Montague Grammar (SMG) are three examples where the notion of category is expanded, other combination rules are used in generating complex expressions, and each expression is given a semantic representation. Each expression in a unification categorial grammar, i.e. a grammar that uses the unification process in parsing (e.g. UCG and SMG), is given a semantic representation in the form of a template which consists of at least:

a) a category specification - the category of the expression, and
b) a semantic representation - the semantic translation of the expression.

In the UCG of [Zeevat et al 87], which consists of three primitive categories (noun, sentence and noun phrase), each expression is given a representation called sign. A sign consists of:

a) a description of how the expression is phonologically derived,
b) a category specification,
c) a semantic representation, and
d) an order specification telling where the expression should be placed when combined with other expression.

In SUNG, based on SMG, each expression is given a semantic representation called *template* which is equivalent to *sign* in UCG and consists of:

a) Word or String - which is the description of how the expression is phonologically derived by specifying the rule and the constituent expressions
b) Syntax - a category specification
c) Semantics - a semantic representation
d) Store - a working storage for resolving anaphora pronouns

which is represented in a following list notation:

[Syntax#String#Semantics#Store]

For example, the template for a noun phrase *a computer* is represented as follows:

[t(neut,Case)#[f2,a,computer]#[[X,I | A] | computer(X)&A]#[]]

The Store is not used currently, since we have eliminated anaphora pronouns resolution routine from the parser. It is difficult to resolve the referent of an anaphora pronoun (e.g. he, she, it) in a situation where there are so many unknown words in the texts. This routine can possibly be included in the system if a commercial online dictionary is used to solve the unknown word problem.

In unification grammars, the syntactic parsing and semantic translation process are carried out in parallel. The semantic translation process is performed through the instantiation and stripping processes as already described and illustrated in section 3.3.4.

5.3 Partial Translation in SUNG

The parser implemented for SMG by Jowsey is non-robust type which can only accept syntactically well-formed sentences with syntactically and semantically predefined words. It returns a null value on receiving ill-formed input. An input is considered an ill-formed if it contains one of the following:

1. An unknown word - a word that is not predefined in the lexicon, this includes miss-spelled words.
2. A grammatically illegal syntactic structure - the structure of the input is grammatically wrong.
3. A not-covered syntactic structure - the structure of the input is not covered by the rules implemented, eventhough it is grammatically correct.

The lexicon implemented in SILOL is an experimental one and the set of grammatical rules implemented is very limited, thus, the SUNG

parser-translator ought to be robust enough to perform partial translations on the document collection and the queries. For example, given a document title as follows (the title of the first CACM document):

Preliminary Report - International Algebraic Language

and only the words *Algebraic* and *Language* are predefined in the lexicon from among the words of the title. The SUNG parser-translator must be able to produce the following partial translation of it:

preliminary(X1) & report(X2) & international(X3) &
algebra(X4) & language(X5) & a(X4,X5).

Notice that the consecutive known words are parsed to form a phrase as reflected by the predicate a(X4,X5), and the unknown words are translated into one-place predicates. Figure_5.1 illustrates the general strategy of partial translation adopted by the SUNG parser-translator using the island focusing approach. The strategy is first to locate the sequences, or the *islands*, of the known words in between the unknown words. Then, the islands are parsed based on the grammar rules available. Some islands might be reduced to a single syntactic category, or a *token*, and some might not be. That depends upon the scope of the grammar implemented.

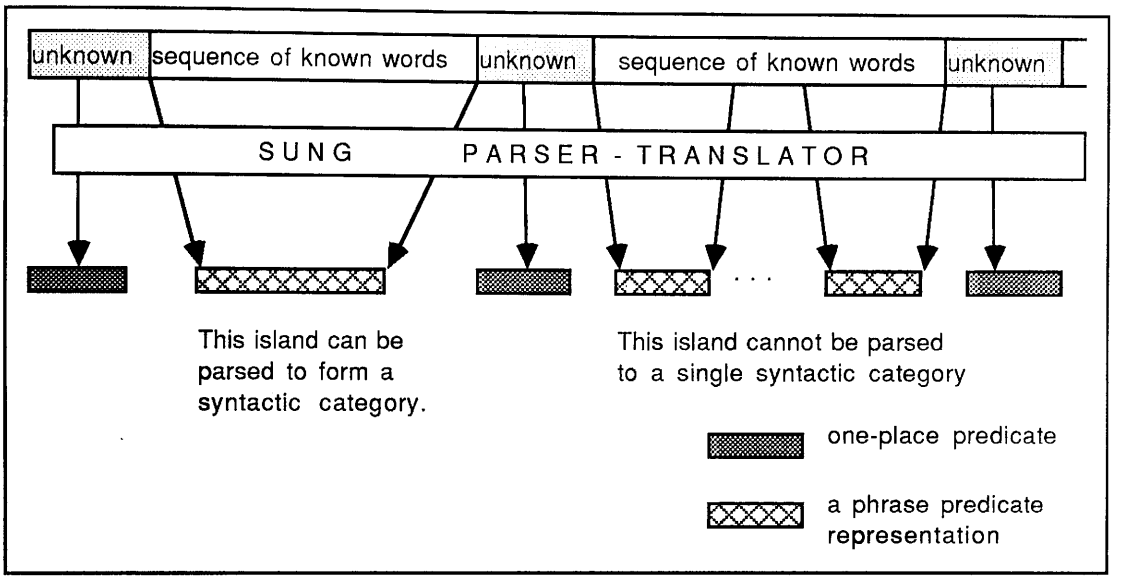


Figure_5.1: Partial Translation Strategy based on Island Focusing

5.4 Bottom-up Shift-Reduce Parser

The parsing algorithm used for SUNG is bottom-up, right to left, shift and reduce algorithm [Aho&Ullman 77]. The bottom-up parser attempts to construct a parse tree for an island beginning at the leaves (the bottom) and working up towards the root (the top). We can think of this process as one of trying to reduce a sequence of words in an island to a single syntactic category. At each step, the right-most part of the sequence which matches the left side of a grammar rule is reduced to the category on the right of the rule. The bottom-up parser avoids the problem of infinite loops on parsing left-recursive structures suffered by top-down approaches. Examples of the left-recursive structures in SUNG are the postmodifying and noun-noun phrases as reflected by the following rules:

facn2 >> cn(Gend):PP + acn2:[PP | P] => cn(Gend):P.

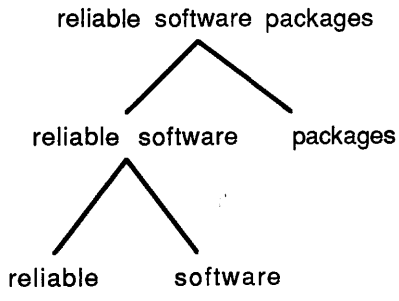
fcn >> cn(Gendx):[X | A] + cn(Gendy):[Y | B] => cn(Gendy):[Y | A&B&r(Y,X)].

The shift and reduce algorithm is chosen because it is the simplest method of implementing bottom-up parsing, although it is not an efficient one. A more efficient algorithm may be investigated as a further research in this area.

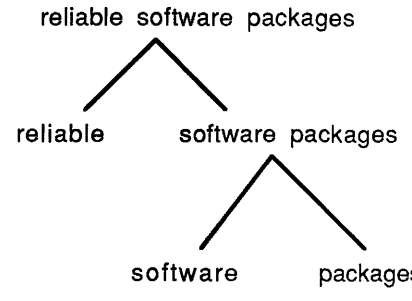
5.5 Handling of Ambiguities

Ambiguity arises when an island can have more than one parse tree. For example, an island which constitutes the phrase *reliable software packages* will have two parse trees as follows:

(Tree_5.1):



(Tree_5.2):



The parsing algorithm implemented for SUNG is a non-deterministic type, built upon the Prolog built-in non-deterministic backtracking . Thus, it is capable of deriving all the possible parses for a given island. The question now arises as to which of these parses to chose. In handling this problem, our strategy is, first, to reduce the number of possible parses. For example, the number of possible parses for the island *reliable software packages* can be reduced to only one, i.e.

Tree_5.2. Our way of doing this is by adopting multiple-pass parsing, i.e. by setting up groups of grammar rules based on priorities over which phrase constructs should be formed first, and each parsing pass is based on one of these grammar groups. In the *reliable software packages* example, if the noun-noun compound rule is given higher priority and grouped for the first pass and the adjective-noun compound rule is grouped for the second pass, then the island will only have one possible parse tree (Tree_5.2) as illustrated below:

reliable software packages
$=>_{\text{pass-1:cn+cn=>cn}}$ reliable (software packages)
$=>_{\text{pass-2:adj+cn=>cn}}$ (reliable (software packages))

where the subscripted arrows ($=>_{\text{subscript}}$) denote parsing steps with the pass numbers and the rules used.

The setting up of grammar groups for each pass is done heuristically based on the following priority criterion:

priority	type of construct
1	conjunction of adjectives
2	noun-noun compounds
3	adjective-noun compounds
4	preposition-noun compounds
5	noun-postmodifier compounds
6	determiner-noun compounds
7	others

We end up having 6 grammar groups, namely: 0, 1, 2, 3, 4 and l(stands for last). The grammar groups are given in Appendix_B, the labels prefixing each rule signify the group names. Our multiple-pass

parser is implemented in Prolog and is based on a simple blackboard architecture, where each grammar group acts as an expert operating on a common blackboard which is initially loaded with the problem to be solved, i.e. the texts to be parsed and translated. The experts then will be scheduled one after another to parse the texts on the blackboard based on their respective knowledge, i.e. their grammar rules. This architecture is depicted in Figure_5.2. The experts scheduler that we implemented for the parser is a simple one-cycle linear type. The performance of the parser-translator can possibly be enhanced by a better grouping of rules and an intelligent scheduler which can direct the parsing operation, especially in facing ambiguity, to choose a semantic translation which is highly preferred to others.

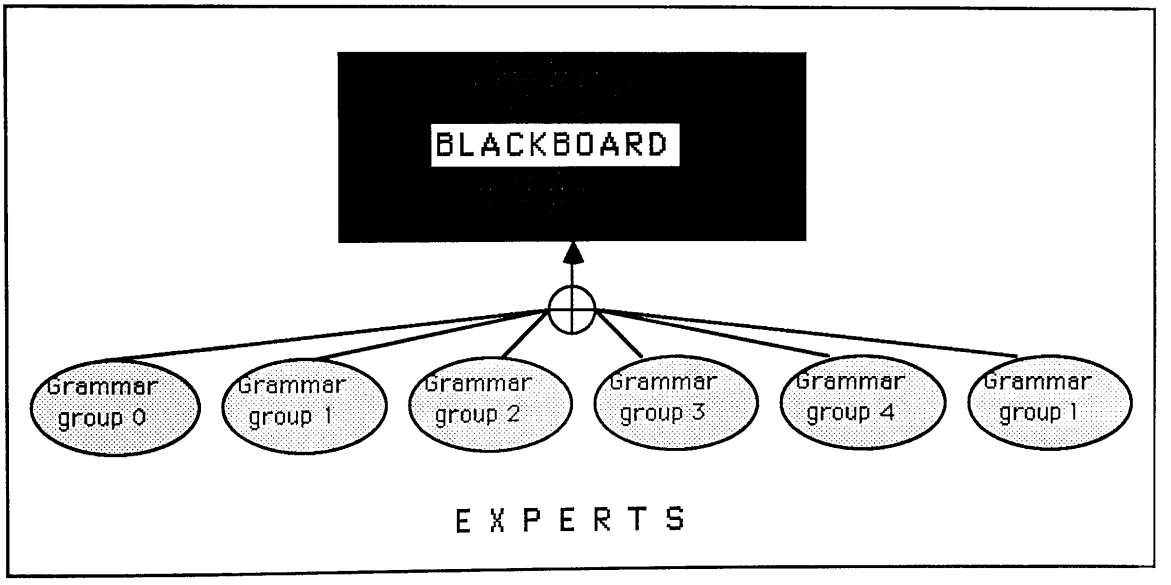
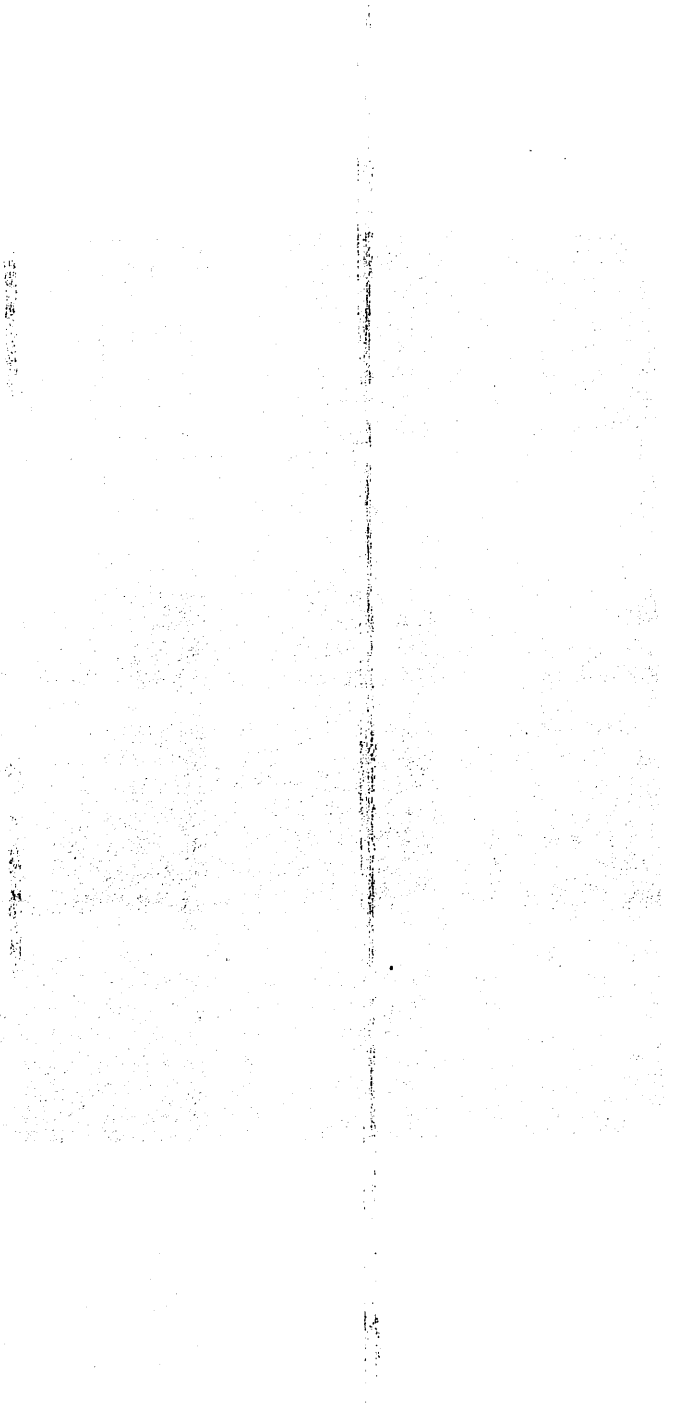


Figure_5.2: A Simple Blackboard Architecture for a Parser

Although the number of possible parses are reduced by adopting multiple-pass strategy, ambiguity still persists as long as more than one parse tree can be derived from a single island. As discussed in section 5.5, an island may be reduced to a single token or more. So, in the case of ambiguity, our strategy is to choose the parse with the smallest number of tokens. But there may be more than one parse having the

smallest number of tokens. In this case, we simply select the first parse encountered. This is not a bad strategy as it may seem, because we have constructed the parser to produce preferred translations through the multi-pass technique. Alternatively, we could collect all the parses with the smallest number of tokens. But this will increase the parsing time. Working in a Prolog environment where execution time is known to be slow, time saving does matter.



Chapter 6: Translation as Part of Document Retrieval System

6.1 Introduction

Instead of using only parsing as part of document retrieval systems, SILOL has used translation, which of course includes parsing, as part of its system. Natural language texts are directly translated into logical form which can be used as a complete contents indicator of a document or query. The translation technique used is based on the semantic translation technique as described in the earlier sections. This approach has contrasted SILOL from those systems described by Smeaton [Smeaton 87], which parse natural language texts into parse trees and where words are combined to form phrase descriptors using heuristic rules. Also, in these systems, single term descriptors are formed separately using conventional methods and combined with the phrase descriptors to form a complete content indicator.

In this chapter we will describe how documents and queries are processed to form their respective indexes through the translation and the normalisation process which is composed of simplification and reduction processes. The similarity values between document and query indexes are computed using *uncertain implication processes* which will also be discussed in this chapter. An uncertain meta Prolog language, which will be referred to as **UN**certain **I**mplication **L**anguage (UNIL), is provided to assist the implication process. This language is used to define implication rules for any particular retrieval strategy and for defining synonyms or thesaurus.

6.2 Translation as Part of Document Indexing Process

In SILOL, the semantic translation process is being used as a major part of the document indexing process. All words in the texts should be identified and given their respective syntactic categories and semantic templates. But for our experimental purpose, only a set of words are chosen to be given their syntactic categories and semantic templates to form an experimental lexicon. Most of these words are those which are used in the queries that will be used to evaluate the performance of SILOL. Based on the given SUNG grammar, a document text is partially translated into its logical representation which is composed of a set of predicates and logical connectives. Those consecutive words having syntactic categories and semantic templates are parsed to form phrases and are translated into a set of predicates joined by logical connectives. Words which are not defined are translated into one-place predicates, and thus, they still serve as parts of document representations. Thus, in our approach, single term and phrase descriptors are formed within the same process.

The predicate names, which are actually English words of the texts, are stemmed before undergoing further indexing processing. The stemming algorithm used is Porter's algorithm [Porter 80] written in Prolog. The stemmed logical representations are then normalised, i.e. *simplified* and *reduced*, to form document indexes. Figure_6.1 shows the steps of the document indexing process, and Figure_6.2 illustrates an example of the indexing process.

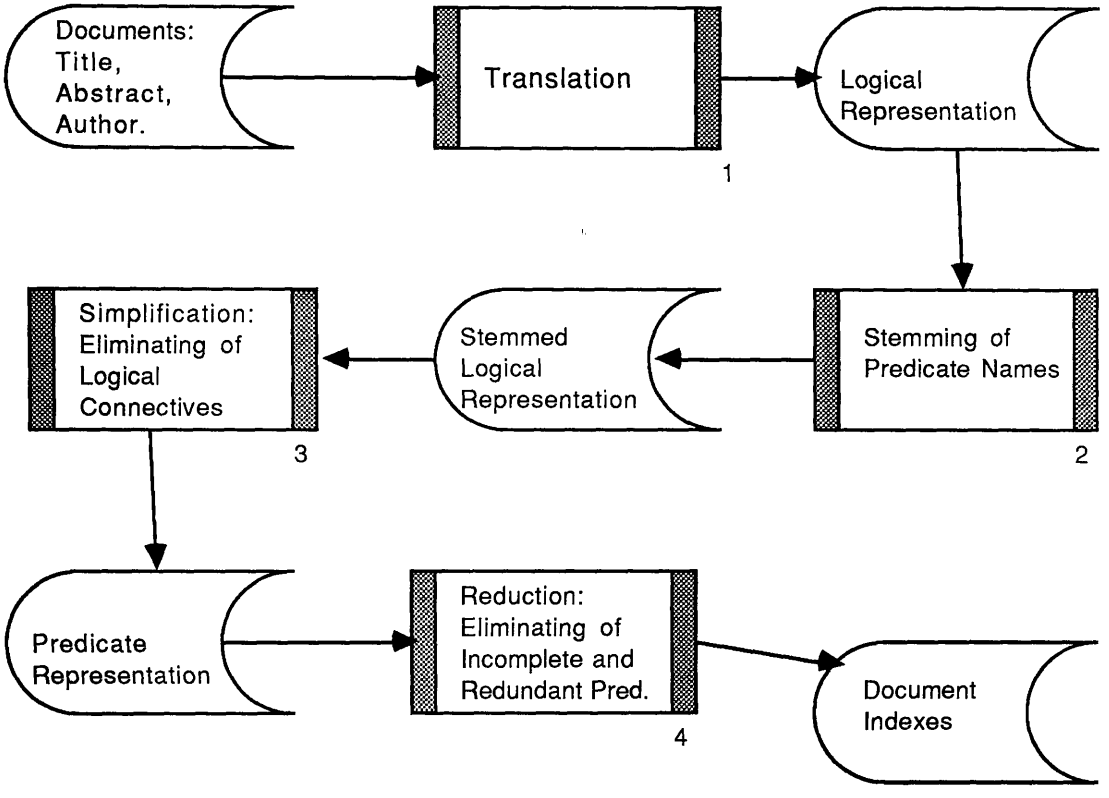


Figure_6.1: Document Indexing Process

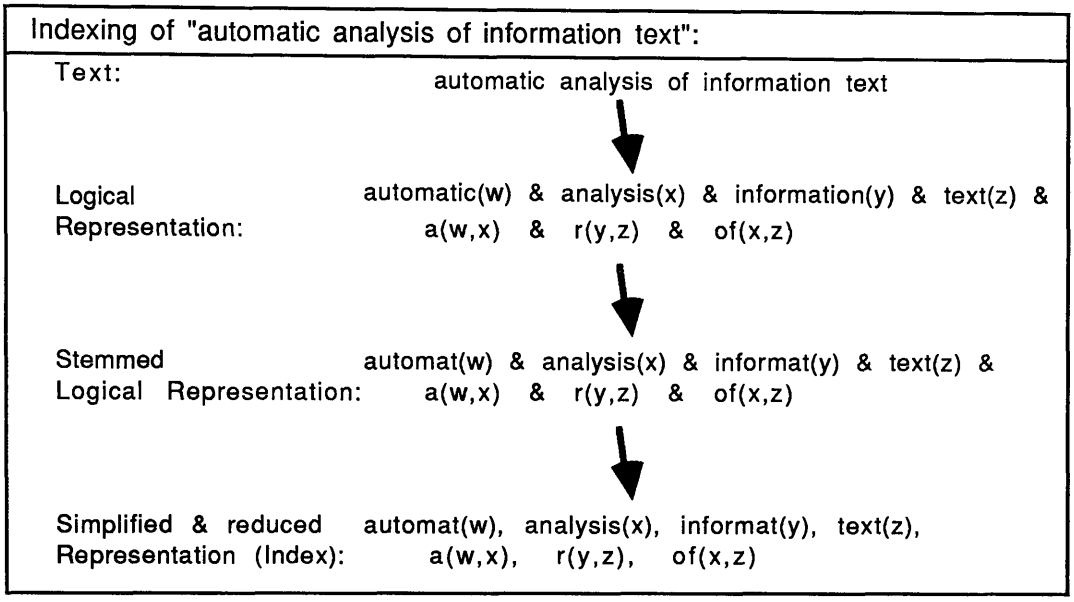


Figure_6.2: An example of indexing process

6.2.1 Simplification Process

The first step in the simplification process is to get rid of the logical connectives. This is done for the simple reason that at present our retrieval strategies do not differentiate them. But instead, the predicates are all assumed to be joined by logical AND. For example, initially, the phrase 'slow or fast processor' is translated into the following representation:

((slow(X) & =(Z,X)) v (fast(Y) & =(Z,Y))) & process(W) & a(Z,W)

but it is simplified by replacing all connectives with logical AND's which are represented by comas (,):

slow(X), =(Z,X), fast(Y), =(Z,Y), process(W), a(Z,W).

The second step of the simplification process is to get rid of the '=' predicates by replacing the representation with an equivalent representation without the '=' predicates. This is done in order to reduce the complexity and to increase the speed of the implication process. For example, the above representation with '=' predicates is simplified into:

slow(X), fast(Y), process(W), a(X,W), a(Y,W).

6.2.2 Reduction Process

The reduction process is to get rid of redundancy which exists in two forms:

1) duplicates of one-place predicates: i.e. one-place predicates having the same name,
2) incomplete predicates: i.e multi-place predicates with dangling arguments.

This redundancy must be eliminated in order to reduce the logical representations to its minimal form. For example, after the simplification process, we may have the following representation:

comput(X),
comput(Y),
comput(Z),
scienc(A),
scienc(B),
program(C),
r(X,A),
r(Y,B),
of(B,C),
in(A,K).

The above representation is not in a minimal form, since it contains more than one one-place predicates with the same name (i.e. computer(X),computer(Y),...) and an incomplete predicate "in(A,K)" with dangling argument K. The task of the reduction process is to reduce it to the following minimal representation:

comput(X),
scienc(A),
program(C),
r(X,A),
of(A,C).

Thus, the reduction algorithm constitutes the following three steps:

1. Deleting incomplete predicates, i.e. predicates with dangling argument(s) are deleted.
2. Unifying the variables of one-place predicates with the same name - this process will automatically affect the variables of the multi-place predicates.
3. Deleting of duplicate predicates - those predicates with the same names and arguments are deleted.

6.3 Translation as Part of Retrieval Process

A user's query is translated into its logical representation as documents are translated. This representation is then simplified and partially reduced. The resulted representation is then ready to be matched with the document representations and their similarity coefficients or values calculated. The matching is performed through an uncertain implication process where values are combined and propagated which finally gives similarity values between the query and the documents. In the following subsections we will describe how the

user's query is processed and how its similarity values to the documents are calculated.

6.3.1 Simplification and Partial Reduction Process

The simplification process here is similar to the one used to process the documents. The partial reduction process is to remove the incomplete predicates from the simplified representation, but duplicates of one-place predicates are not removed and duplicates of multi-place predicates may or may not be removed depending upon the retrieval strategy adopted. The duplicates of one-place predicates are not removed because our retrieval strategy is based on the tf x idf weighting scheme which takes into account the term frequencies within-query and within-document. Concerning the duplicates of multi-place predicates, it depends on the retrieval strategy adopted whether to include them or not in calculating similarity values. For example, after the simplification process is done on the query representation, we may have the following simplified representation:

```
comput(X),
comput(Y),
comput(Z),
scienc(A),
scienc(B),
program(C),
r(X,A),
r(Y,B),
of(B,C),
in(A,K).
```

If the partial reduction process is applied on this query representation, i.e. by eliminating incomplete predicates only, then we will have the following partially reduced representation with duplicates in both one-place and multi-place predicates.

```
comput(X),
comput(Y),
comput(Z),
scienc(A),
scienc(B),
program(C),
r(X,A),
r(Y,B),
of(B,C).
```

After undergoing the simplification and reduction processes, the query and the documents are represented by a common representation scheme, i.e. the predicate representation form. Now, we need an algorithm to compute the similarity values between the query and the documents. The following section will discuss an uncertain implication process that is used to perform this task.

6.3.2 Uncertain Implication Process

An *uncertain implication process* is used to combine and propagate values that will give a measure of similarity between a document and a query through a process of deduction under uncertainty. In this process each successfully instantiated predicate will be given a value to be combined with other values or propagated to other predicates. Unsuccessfully instantiated predicates are given a zero value. In a logically strict implication process, such as in Prolog, a successfully instantiated predicate is given a TRUE value and an unsuccessfully instantiated one is given a FALSE value. In our case these values are not boolean, but real figures based on some statistical calculation. A basis of calculating these values in our uncertain implication process will be explained in section 6.3.2.1. An uncertain meta Prolog language, which will be referred to as UNIL, is provided to assist and control the

implication process. The language UNIL will be discussed in section 6.3.2.2.

6.3.2.1 Calculation of Propagated Value

An Implication process is used to combine and propagate values that will give a measure of similarity between a query index and a document index. Where should these values come from? It seems that the statistically-based weighting schemes are the best we have so far for this purpose. Thus, in our experiments, the values used are the weights of the stemmed predicate names based on the tf x idf weighting scheme. This weighting scheme is chosen because it is generally considered as being the most effective [Salton 86]. Thus, the values that will be assigned to the successfully instantiated predicates during this implication process are as follows:

1. For a one-place predicate P(arg), where P is a stemmed predicate name and arg is its argument: Its weight in a particular document is the tf x idf weight of the word P in the document, i.e. the frequency of P in the document multiplied by its inverse document frequency (idf) value. The idf value of a word of term t is computed using the following formula:

$$idf(t) = - \log(Freq(t)/N)$$

   where Freq(t) is the number of documents in which the term t appears at least once, and N is the total number of documents in the system.

2. For a multi-place predicate Relation(arg1,...,argn), where

Relation can be any one of the generalised-relations: There are many ways of assigning weight to a multi-place predicate. One way is to give it a constant value as done by Smeaton. Another is to take the average weight of the predicates involved in the relationship as performed by Fagan. For example given the following predicates and weights:

comput(X)	weighs 5.6
scienc(Y)	weighs 3.7

then the weight for 'computer science' noun-noun relationship predicate r(X,Y) is 4.65.
Instead of using the average, we may also try to use the sum of the weights of the predicates involved.

The final similarity value between a query and a document is obtained by summing up the values of all predicates in the query index which are successfully instantiated during the implication process. This represents a basis of retrieval strategies that can be applied to our logical-linguistic model. The results reported in the later chapters are based on this basis of retrieval strategies.

The architecture of the retrieval process of SILOL is depicted in Figure_6.3. The *Retrieval Processor* receives a query from a user and passes it to the *Translator and Indexing Unit* which transforms it into its query index. The Retrieval Processor then invokes the Implication Unit to compute similarity values between the query index and the document indexes by consulting the *Implication Rules* and the *Thesaurus,* if there are any. Documents with top similarity values are then presented to the user.

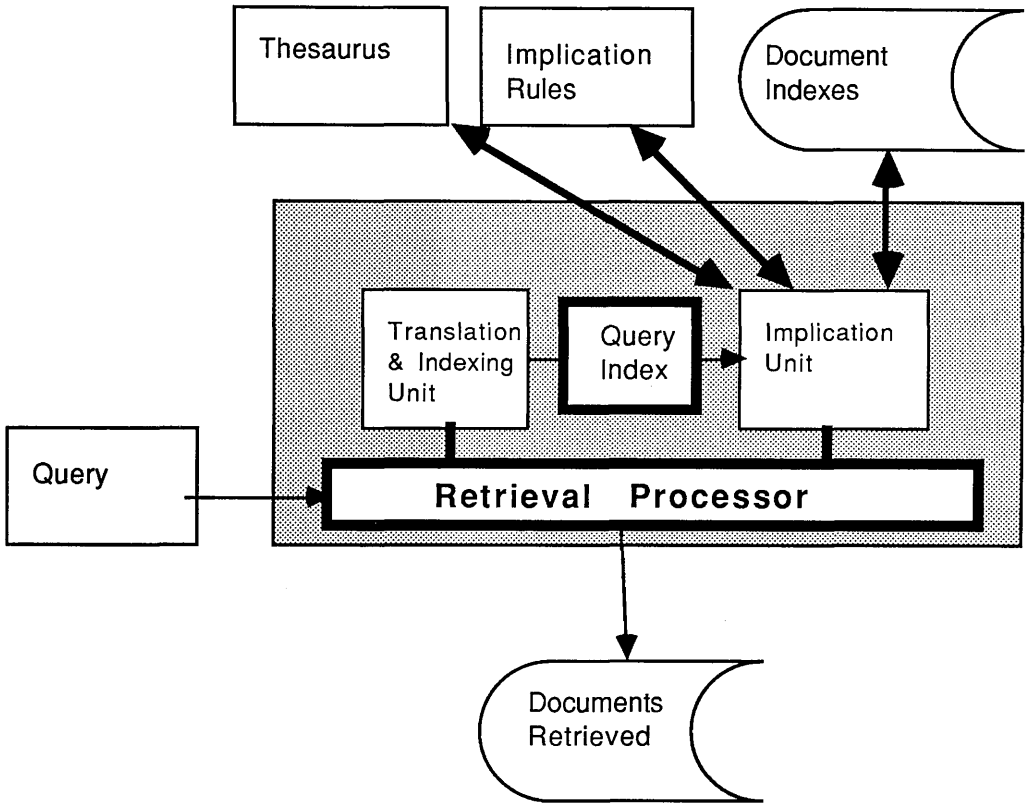


Figure_6.3: An Architecture of a Retrieval Process

6.3.2.2 Uncertain Implication Language (UNIL)

UNIL is used to define a set of implication rules in the handling of multi-place predicates for a particular retrieval strategy, and it can also be used to define a thesaurus. Thus we divide the rules written in UNIL into two types:

1) *implication rules* for multi-place predicates, and
2) *synonym rules* for defining thesaurus.

UNIL is Prolog-like except for additional syntactic constructs that

facilitate *uncertainty* propagation [Abdullah 86] and *no-chaining* of rules. A rule is written in the following Prolog-like forms:

$$r: B :- A_1 \text{ op } A_2 \text{ op } \ldots A_m.$$

where r is the label to differentiate UNIL rules from ordinary Prolog rules, B and A_i are predicates and op's are combination operators. An operator can be a conjunction(,), a disjunction(;), a summation(@) or a multiplication(&). For example, the following rule defines the implication between the noun-noun phrases with the adjective-noun phrases and some prepositional phrases :

r: r(X,Y) :- a(X,Y) ; of(X,Y) ; on(X,Y).

i.e. a(X,Y), of(X,Y) or on(X,Y) implies r(X,Y).

6.3.2.2.1 Combination operators

There are four combination operators defined in UNIL, namely:

1. the conjunction operator (,)
2. the disjunction operator (;)
3. the summation operator (@)
4. the multiplication(&).

The combination operators will determine how the values of the predicates on the right-hand-side(RHS) of the rule are combined and propagated to the predicate on the left-hand-side(LHS). The conjunction(,) and disjunction(;) operators, based on Fuzzy Logics [Zadeh 83], are the minimum and the maximum binary operators

respectively. The summation(@) and multiplication(&) operators are the binary sum and multiplication operators, respectively. Following are few examples of how the values are computed using these operators :

20.0 ; 10.0 = 20.0

1.2 , 100.3 = 1.2

10.0 @ 20.4 = 30.4

5 @ 2 ; 4 = 7

5 @ (2 ; 4) = 9

(1 ; 2 ; 3) @ (1 , 2 , 3) = 4

2 & 3 = 6.

The above expressions are evaluated from left to right and the order of operator precedence is as follows:

,

&

@, ;

The use of brackets may specify the precedence explicitly.

6.3.2.2.2 Uncertainty Construct - cf(Const)`

The uncertainty construct in UNIL allows the predicates A_i on the RHS of the rules to be qualified with uncertainty factor values (cf). This construct allows us to define partial matching. Any predicate A_i can be qualified with a cf value Const as follows :

cf(Const)`A_i

The value given to cf(Const)`A_i is the product of Const and the value of the predicate A_i. Following are a few examples of the implication

and synonyms rules using the uncertain construct (all implication rules have to undergo stemming process) :

Implication rules:

r: of(X,Y) :- cf(0.9)`for(X,Y).

r: a(X,Y) :- cf(0.5)`(r(X,Y);r(Y,X)).

Synonym rules:

r: computer(X) :- cf(0.6)`minicomputer(X).

r: computer(X) :- cf(0.5)`(machine(X);processor(X)).

6.3.2.2.3 No-chaining Construct - d:

The *no-chaining construct* is introduced in order to control the chaining of rules. This construct is used to avoid infinite chaining of rules. For example, if we have the following implication rules defined to equate the two multi-place predicates r and a as equivalent:

r: r(X,Y) :- a(X,Y).

r: a(X,Y) :- r(X,Y).

then we will have the problem of infinite loops. Imagine that a query index contains a predicate r(A,B) and there is no occurrences of predicates r and a in the document under investigation. This will cause the retrieval processor to invoke the implication rules to perform the inference process. The first rule above will be triggered first, which subsequently will trigger the second rule, which subsequently will trigger the first rule, which subsequently will trigger the second rule, ... until infinitum.

In order to prevent this, we introduce the no-chaining construct

by introducing a label 'd' to be written before any predicates in the body parts, or the RHS, of the rules. The label 'd' before a predicate P means that the presence of P in a document is check without consulting any rule. If only the predicate P can be satisfied directly from the document index then the value or weight of P in the document is calculated, otherwise, the predicate is considered unsuccessfully instantiated and is given a zero value. Thus the above two implication rules are rewritten as follows:

r: r(X,Y) :- d:a(X,Y).
r: a(X,Y) :- d:r(X,Y).

6.3.2.2.4 Checking Query Index - q:

Sometimes it is necessary for an implication rule to check the presence or absence of any predicate in a query index. This is useful in defining synonyms. Thus, a label 'q' is introduced to implement this capability. If a predicate with label 'q' cannot be satisfied directly from the query index then the rule is considered as failed and no values are propagated. No values are calculated or propagated from predicates with label 'q'. Below is an example of synonym rule which tries to match a phrase 'programming language' in a query with a word 'Fortran' in documents with uncertainty factor of 0.9 :

r: language(X1) :- q:programming(X2), q:r(X1,X2), cf(0.9)`d:fortran(Y).

6.3.2.2.5 Cut - !

A cut (!) facility is introduced in UNIL in order to speed up the implication. A cut will terminate further processing of the RHS of a

rule if any previous predicate is successfully instantiated. For example, given the following implication rule :

r: r(X,Y) :- a(X,Y) ; of(X,Y) ; on(X,Y) ; with(X,Y) ; using(X,Y).

this rule will cause all the predicates on the RHS to be evaluated and their maximum value propagated to the predicate on the LHS. But all of them might be evaluated to the same value, as what happen in our evaluation scheme where the weights of a(X,Y), of(X,Y), ..., using(X,Y) are calculated using the same function with X and Y as input, so it is time saving to stop evaluating the rest of the predicates once a predicate is successfully instantiated. Thus, the above rule can be written with cut (!) as follows:

r: r(X,Y) :- a(X,Y) ;!; of(X,Y) ;!; on(X,Y) ;!; with(X,Y) ;!; using(X,Y).

6.3.2.2.6 Macro Predicates

Some macro predicates are provided in order to make the writing of the rules more simpler. They are as follows:

exists_pair(X,Y) : Evaluated to zero if there is no two-place predicates between X and Y, in that order. Otherwise, it is evaluated to the value of the multi-place predicate between X and Y. This macro is to help in specifying a retrieval strategy which does not differentiate dependency types.

sum_pair(X,Y) : Evaluated to zero if there is no two-place predicates between X and Y, in that order. Otherwise, it is evaluated to the sum of values of all predicates with arguments X and Y in that order.

average(X,Y,..,Z) : Evaluated to the average weight between X,Y,...,Z. This macro is used when the values of the multi-place predicates are calculated as the average weight of their arguments.

sum(X,Y,...,Z) : Evaluated to the sum of the weights of X,Y,...,Z. This macro is used when the values of the multi-place predicates are calculated as the sum of the weights of their arguments.

pair_constant : Gives the Smeaton's constant for pair dependencies. This macro is used when the values of the multi-place predicates are calculated as Smeaton's constant.

triple_constant : Gives the Smeaton's constant for triple dependencies. This macro is used when the values of the multi-place predicates are calculated as Smeaton's constant.

pair(X,Y) : Evaluated to TRUE or FALSE depending upon whether there is an occurrence of dependency between X and Y, in that order. This is a strict boolean predicate which returns a boolean value.

6.3.2.2.7 Strict Construct : s`

Sometimes it is useful to be able to imbed in the UNIL implication rules some strict implications which handle only the boolean values 0 and 1. A strict construct is provided in UNIL to implement this by

marking the predicates which need to be instantiated or processed strictly with **s`**, as illustrated in the following examples :

```
r: one_or_zero(X,Y) :- s`r:(X,Y).
r: a(X,Y) :- exists_pair(X,Y) @
                average(X,Y) & s`(vso(X,Y,_);vso(X,_,Y);vso(_,X,Y)).
```

Those predicates which are marked with strict tag s` will only be evaluated strictly using the Prolog resolution mechanism to a value of 1 or 0 depending on whether they are successfully instantiated or not.

It is possible to invoke an ordinary Prolog rule from a UNIL rule by using the strict construct. The following example will illustrate this invocation in looking for transitive dependencies:

```
r: weight_pair(X,Y) :- exists_pair(X,Y) ; average(X,Y) &
                                s`transitive_pair(X,Y).
transitive_pair(X,Y) :- pair(X,Z) , (pair(Z,Y) ; pair(Y,Z)).
```

6.3.2.2.8 Examples of Implication Rules and Synonyms

In the previous section we have described the language UNIL designed solely to provide the facilities for controlling and defining the implication process. In this section we will give some examples of how this language is used.

Following is an example of implication rules in UNIL for a retrieval strategy which does not distinguish multi-place predicates but take into consideration the order of their arguments:

```
r: a(X,Y):- anytype(X,Y).
r: r(X,Y):- anytype(X,Y).
r: in(X,Y):- anytype(X,Y).
r: inside(X,Y):- anytype(X,Y).
r: of (X,Y):- anytype(X,Y).
r: on(X,Y):- anytype(X,Y).
r: for(X,Y):- anytype(X,Y).
r: by(X,Y):- anytype(X,Y).
r: with(X,Y):- anytype(X,Y).
r: within(X,Y):- anytype(X,Y)..
r: without(X,Y):- anytype(X,Y).
r: using(X,Y):- anytype(X,Y).
r: sv(X,Y):- anytype(X,Y).
r: ov(X,Y):- anytype(X,Y).
r: anytype(X,Y):- exists_pair(X,Y);
  pair_constant & s`(d:vso(X,Y,_);d:vso(X,_,Y);
                                    d:vso(_,X,Y)).
r: vso(X,Y,Z):- d:vso(X,Y,Z);
  (exists_pair(X,Y);exists_pair(X,Z);
        exists_pair(Y,Z)).
```

Synonyms can be defined by experts in the domain knowledge concerned. Study has to be made of how to combine and propagate the weights in defining the synonym rules. Following are few examples of synonym rules in the area of computer science :

1. *tss* is an abbreviation of *time sharing system*:

```
r: system(X):- q:time(Y), q:sharing(Z), q:r(Y,X), q:(Z,X), d:tss(_).
```

2. *intermediate language* is synonymous to *intermediate code*:

```
language(X):- q:intermediate(Y), q:a(Y,X), d:intermediate(A),
 d:code(B), d:(A,B).
```

3. *lexical analysis, syntax analysis, code generation and optimization* are components of a *compiler*:

```
compiler(_):-
    (d:lexical(A),d:analysis(B),d:a(A,B))@
    (d:syntax(C),analysis(D),r(C,D)) @
    (d:code(E),d:generation(F),r(E,F)) @
    (d:code(G),d:optimization(H),r(G,H)).
```

4. *monitor, semaphore, guard, and synchronization* are relevant to *communicating processes*:

```
communicating(X):- q:processes(Y), q:r(X,Y),(d:monitor(_) @
d:semaphore(_) @ d:guard(_) @ d:synchronization(_)).
```

5. *security* is synonymous to *privacy, cryptography, protection* and *encryption*:

```
security(_):- d:privacy(_)@d:cryptography(_) @ d:protection(_) @
d:encryption().
```

Chapter 7: Experimental Setup

7.1 The Test Collection

In order to perform experiments with our system, a familiar CACM test collection of documents and queries gathered by Fox at Cornell [Fox 83] has been used. This collection contains 3204 "Communications of ACM" articles and 64 natural language queries together with their relevance assessments. From the collection of articles we use only the title, author and abstract fields to represent our corpus of stored documents in performing our experiments. Regarding the queries for the experiments, we have chosen the same 48 queries selected by Smeaton from the above collection. These 48 queries have been transformed to noun phrase queries by Smeaton due to the limitation of his grammar which only covers noun phrases.

7.2 Translation Statistics on the Test Collection Documents

The corpus of stored documents was translated by the SUNG parser-translator, and the statistics obtained after simplification and reduction process are shown in Table_7.1.

No. of documents:		3204
No. of occurrences of each multi-place predicate:-		
	r:	3034
	for:	299
	a:	1826
	with	39
	within	4
	without	4
	sv	335
	ov	209
	of	932
	vso	66
	in	147
	by	30
	using	13
	on	45
Total number of dependencies (or multi-place predicates):		6983
Average no. of dependencies per document:		2.18

Table_7.1: Translation Statistics on the Documents

In spite of having a very limited grammar and lexicon, the number of multi-place predicates obtained after simplification and reduction process is very encouraging: an average of 2.18 per document. We envisage that by using a commercial on-line dictionary, the number of unknown words will be reduced and thus the translation statistics will be much improved.

7.3 Organisation of Retrieval Strategies

There are two sets of retrieval experiments to be performed. One is, a set of experiments based on retrieval strategies without relevance feedback which will be referred to as non-feedback retrieval

experiments. Another is a set of experiments with relevance feedback based on the imaging approach as proposed by van Rijsbergen [van Rijsbergen 89].

7.3.1 Non-feedback Retrieval

Ideally, what should be done is to evaluate the retrieval strategies exhaustively with all values of all the control parameters. But, this would lead to an unmanageable large set of experiments, since there are too many parameters and variants to control. It is important to stress that it is not our intention to produce the best retrieval strategy for our system, but the aim is merely to demonstrate the viability of this model and to show that it is more effective than the conventional approach.

In order to make the number of experiments manageable, we have organised our retrieval strategies based on several control parameters. There are four main parameters used in defining retrieval strategies, and they are as follows:

1. The weighting scheme for multi-place predicates:
   a) constants - the weighting is based on Smeaton's constants.
   b) average - the average scheme as used by Fagan.
   c) sum - the weight of a multi-place predicate is calculated as the sum of the weights of the predicates involved in the relationship.

2. Types of Relationships:
   a) typed - names of multi-place predicates are being distinguished in the implication process. In this case, the predicate r(x,y) does

not imply a(x,y).

b) non-typed - names of multi-place predicates are regarded as equivalent in the implication process. In this case, predicate r(x,y) implies a(x,y).

3. Order of arguments:

a) ordered - the order of arguments of multi-place predicates is taken into consideration in the implication process. Thus, r(x,y) does not imply r(y,x).

b) non-ordered - the order of arguments of multi-place predicates is immaterial. Thus, r(x,y) implies r(y,x).

4. Duplication of multi-place predicates:

a) duplicates allowed - duplication of multi-place predicates having the same name and arguments are not eliminated from the query indexes.

b) duplicates eliminated - the above duplicates are eliminated from the query indexes.

7.3.2 Retrieval with Nearest Neighbour Relevance Feedback

After performing experiments on various retrieval strategies using the parameters as described above, we would then like to perform experiments using relevance feedback with nearest neighbours information. One of the best retrieval strategies from the non-feedback retrieval experiments will be chosen to perform the initial ranking of documents with respect to a query and a few top ranked documents will be retrieved and identified as relevant or not. From this set of retrieved and relevant documents, we can obtain all other unretrieved documents which have any of the retrieved and relevant documents as

their nearest neighbour. This set of unretrieved documents have the potential of also being relevant since they are 'close' to the retrieved and relevant ones, and thus their initial similarity values to the query will be updated based on their distances to their nearest neighbours. From the updated similarity values, a new ranking of documents can be obtained and evaluated.

The detailled implementation of this retrieval and its results shall be discussed in Chapter 9.

7.4 Translation Statistics on the Experimental Queries

Likewise, the translation statistics on the 48 experimental queries are obtained. The figures obtained are shown in Table_7.2. The average number of dependencies per query obtained is 5.9 when duplicates are allowed. The range of average number of dependencies per query obtained by Smeaton is between 3.75 to 9.98, depending upon which heuristic rules he used. The results obtained by Smeaton, show that the quality of dependencies rather than the number of dependencies improves the retrieval effectiveness. The average number of dependencies obtained by our method seems to lie in the middle of his range. This may be an indication of its quality.

No. of queries:		48
Total no. of occurrences of each multi-place predicate:-		
	r:	147
	for:	21
	a:	50
	with	5
	sv	6
	ov	10
	of	26
	vso	1
	in	9
	by	3
	using	5
	on	2
Total number of dependencies (or multi-place predicates):	285	
Average no. of dependencies per query:	5.9	

Table_7.2: Translation Statistics on the Queries

7.5 Methods of Evaluating Retrieval Effectiveness

To evaluate and compare retrieval effectiveness we have used mainly the Recall Cutoff evaluation technique [van Rijsbergen 79], i.e. by taking the average precision values at standard recall points of 10, 20, . . . , 100, and using macro-evaluation approach to get the overall average precision values.

An additional evaluation method, Document Cutoff across rank, which is less opaque than the precision-recall evaluation is used to assist the evaluation process further. Below we explain the methods in details.

7.5.1 Recall Cutoff Evaluation

Given the following table :

	RELEVANT	NON-RELEVANT	
RETRIEVED	$A \cap B$	$\bar{A} \cap B$	B
NOT RETRIEVED	$A \cap \bar{B}$	$\bar{A} \cap \bar{B}$	$\bar{B}$
	A	$\bar{A}$	N

where A is the set of relevant documents, B is the set of retrieved documents and N is the set of all documents in the collection. The measures of effectiveness Precision and Recall are then defined as follows:

$$\text{Precision} = |A \cap B| \; / \; |B|$$

$$\text{Recall} = |A \cap B| \; / \; |A|$$

For each query submitted to a retrieval system, a number of tables as shown above can be constructed by varying some parameter. Possible parameters are the rank position of the document or the document score. Based on each of these tables a precision-recall value can be calculated. If λ is the parameter, then P_λ denotes precision, R_λ denotes recall, and a precision-recall value will be denoted by the ordered pair (R_λ, P_λ). A precision-recall curve can be obtained by plotting and joining these pairs. The curves for all the queries are subsequently averaged to obtain the overall performance curve.

We have used rank position for the parameter λ, and tied ranks due

to documents having the same score are resolved by ordering them using their serial numbers. To obtain the set of observed points we specify a subset of the parameters λ. Thus (R_θ, P_θ) is an observed point if θ corresponds to a value of 1 at which an increase in recall is produced. We now have a set of observed points:

$$G_S = \{ (R_{\theta S}, P_{\theta S}) \}$$

which is then used in interpolating precision at the standard recall values R = {10, 20, ..., 100}. To interpolate between any two points we define:

$$P_S(R) = \{\sup P : R' >= R \text{ s.t. } (R', P) \in G_S\}$$

where R is a standard recall value. The average precision value at the standard recall value R is calculated as :

$$\tilde{P}(R) = \Sigma_{s \varepsilon S} P_S(R) / |S|.$$

This calculation of the average precision is known as the macro-evaluation approach.

7.5.2 Document Cutoff Evaluation Across Rank

As addition to recall cutoff evaluation, we will also use document cutoff evaluation across rank. It simply gives the number of relevant documents retrieved at specified rank positions summed over all the queries. It is also possible to include the number of queries which fail to retrieve any relevant document by these same rank positions in this evaluation.

7.5.3 Significance Test

The reason for evaluating retrieval strategies is to compare their retrieval effectiveness. Suppose that we have evaluated two retrieval strategies using precision-recall evaluation. To decide whether there is any significant difference in their performance, we can use the significance test proposed by Spark-Jones and Bates which uses a crude rule of thumb based on the areas under the precision-recall curves [Harper 80]. The difference is considered to be significant if the difference between the areas is a least 5% of the smaller area. Or in other words, at each standard recall value, the precision of one strategy should be more than 5% of the other strategy. An increase of 10% is considered as material. It has become a common practice to estimate the differences in area by eyes rather than by program.

Chapter 8: Experimental Results of the Non-Feedback Retrieval

8.1 Benchmark

We have chosen to use tf x idf weighting scheme as part of our retrieval strategies, thus, the tf x idf ranking using only one-place predicates in the query index without any implication rule will be the basis of our benchmarks retrieval strategy. This retrieval strategy implemented on the CACM test collection uses within-query, within-document and collection-wide term frequencies. The score for each document is computed as the sum of the individual term (predicate in our case) weights of the index terms which occur in the document texts. If there is more than one occurrence of an index term in either the document or query then all of them are counted. The tf x idf score for the j^{th} document (d_j) is calculated as given in the following formula:

$$S_{d_j} = \Sigma_{i \varepsilon Q} \; w_i \cdot freq(i,j)$$

where Q is the set of terms in the query index, w_i is the individual idf term weight of the i^{th} term and freq(i,j) is the number of times the i^{th} term occurs in the document d_j.

The set of precision figures at standard recall points for the benchmark retrieval is given in Table_8.0.

Recall Levels	BENCHMARK
10	52.22
20	38.52
30	31.90
40	24.49
50	21.01
60	17.59
70	12.13
80	10.23
90	7.04
100	6.09
Average	22.12

Table_8.0: Benchmark

8.2 Presentation and Analysis of Results

Several retrieval strategies have been performed and the results obtained are given and analysed individually in the following sections.

8.2.1 Experiment_1: Most rigid set of implication rules.

In the first few experiments, we have adopted the constant weighting scheme and allowed duplicates of multi-place predicates in the query index. This strategy is generally assumed unless otherwise stated.

The first retrieval strategy experimented with is the one with the most rigid implication rules, i.e. the retrieval strategy with *typed* and *ordered* parameters which distinguishes the names of multi-place predicates and takes the order of their arguments into consideration in performing matching operation. The results obtained for this strategy are given in Table_8.1.

Strategies:			Exp_1
Parameters	weight	BENCHMARK	constant
	type		yes
	order		yes
	duplicate		yes
Recall Levels	10	52.22	54.32
	20	38.52	40.96
	30	31.90	33.05
	40	24.49	24.61
	50	21.01	21.67
	60	17.59	18.32
	70	12.13	12.46
	80	10.23	10.69
	90	7.04	7.24
	100	6.09	6.17
	Average	22.12	22.95
	% Increase		3.7

Table_8.1: The Most Rigid Retrieval Results

The average precision obtained shows a marginal improvement of 3.7% over the benchmarks. This is quite predictable, since the implication rules are too strict to strike a good number of matches. In this strategy the phrase *information technology* is not considered to be fully matched with the phrase *technology of information.*

8.2.2 Experiment_2: Non-typed Retrieval Strategy

We have seen that a set of most rigid implication rules has given just a marginal improvement over the benchmark. The next retrieval strategy to try is the one with less rigid implication rules. In this strategy, the implication rules defined do not distinguish the types of dependencies, i.e. the multi-place predicate names are indistinguishable. Thus, r(x,y) implies a(x,y), sv(x,y), etc. But the ordered parameter, which differentiates between r(x,y) and r(y,x), still holds. The results obtained are given in Table_8.2.

Strategies:			Exp_1	Exp_2
Parameters	weight	BENCHMARK	constant	constant
	type		yes	no
	order		yes	yes
	duplicate		yes	yes
Recall Levels	10	52.22	54.32	54.40
	20	38.52	40.96	41.42
	30	31.90	33.05	33.10
	40	24.49	24.61	24.87
	50	21.01	21.67	21.70
	60	17.59	18.32	19.32
	70	12.13	12.46	13.49
	80	10.23	10.69	11.73
	90	7.04	7.24	8.28
	100	6.09	6.17	7.21
Average		22.12	22.95	23.55
% Increase			3.7	6.4

Table_8.2: Exp_2 non-typed Retrieval Results

The average precision value obtained shows a substantial improvement over the previous results (from 3.7% to 6.4%). This means that a less rigid implication rule has increased the number of matches which brings improvement to the effectiveness of retrieval.

8.2.3 Experiment_3: Non-typed and Non-ordered Retrieval Strategy

The strictness of the last implication rules can be lessened further by adopting non-ordered parameter. This strategy will increase the number of matches further, since r(x,y) will now implies r(y,x), a(x,y), a(y,x), etc. The results obtained are shown in Table_8.3.

Strategies:			Exp_2	Exp_3
Parameters	weight	BENCHMARK	constant	constant
	type		no	no
	order		yes	no
	duplicate		yes	yes
Recall Levels	10	52.22	54.40	54.74
	20	38.52	41.42	41.23
	30	31.90	33.10	33.10
	40	24.49	24.87	24.85
	50	21.01	21.70	21.76
	60	17.59	19.32	19.37
	70	12.13	13.49	13.52
	80	10.23	11.73	11.81
	90	7.04	8.28	8.28
	100	6.09	7.21	7.21
Average		22.12	23.55	23.59
% Increase			6.4	6.6

Table_8.3: Exp 3 Non-typed and non-ordered retrieval

The results obtained for this strategy are better than the previous one, eventhough the level of improvement is not substantial. Now we can conclude that a less strict set of implication rules performs better than a strict one in terms of retrieval effectiveness. The experiments have shown that the percentage increase in the average precision value obtained by the current least strict set of implication rules is almost double the value obtained by the most strict set (from 3.7% to 6.6%).

8.2.4 Experiment_4: No-duplicates allowed Retrieval Strategy

The retrieval strategies in all previous experiments allow duplicates of multi-place predicates which have the same name and arguments in the query index. Our next experiment is to test whether the absence of these duplicates affects the retrieval effectiveness. So the next retrieval strategy is similar to the one in Experiment_3 (Exp_3) except for the

no-duplicate parameter. The results obtained are given in Table_8.4 which shows a slight decrease in the average precision value obtained by Exp_3. But when we observe closely the values obtained for all the recall levels, we find that the values of the first three recall levels (10,20,30) are less than their counter-parts in Exp_3. Since the first few recall levels are more important than the rest in evaluating retrieval effectiveness, we can conclude that the retrieval strategy which allows duplicates in the query representation is better than the one which doesn't. From the results obtained by Smeaton, it can be concluded that the quality of word dependency improves retrieval effectiveness. This suggests that the quality of the word dependencies obtained from the multi-place predicates is high, and therefore we surmise that the more we include them in the calculation the better.

Strategies:			Exp_3	Exp_4
Parameters	weight	BENCHMARK	constant	constant
	type		no	no
	order		no	no
	duplicate		yes	no
Recall Levels	10	52.22	54.74	54.14
	20	38.52	41.23	40.73
	30	31.90	33.10	32.48
	40	24.49	24.85	25.54
	50	21.01	21.76	22.05
	60	17.59	19.37	19.66
	70	12.13	13.52	13.54
	80	10.23	11.81	11.83
	90	7.04	8.28	8.34
	100	6.09	7.21	7.26
Average		22.12	23.59	23.56
% Increase			6.6	6.5

Table_8.4: Exp_4 The no-duplicate Retrieval Results

8.2.5 Experiment_5: Average Weight Strategy

In all the experiments so far, the weights given to multi-place

predicates are based on Smeaton's constants. Fagan used a different method of calculating these weights, i.e. the average method as described in section 6.3.2.1, where the weight for a particular multi-place predicate is the average weight of all the predicates involved. Our next experiment is to compare which of these two weighting schemes performs better in our system. The retrieval strategy adopted for this experiment is similar to Exp_4, except in this experiment the average weighting scheme of Fagan is used for multi-place predicates. The results obtained are given in Table_8.5. There is a substantial increase in the average precision value obtained, 8.2%, as compared to the results obtained using Smeaton's constants (6.6%). The precision values obtained at all recall levels also give better results. This strongly concludes that Fagan averaging scheme performs better than Smeaton's constants in our system.

Strategies:			Exp_3	Exp_5
Parameters	weight	BENCHMARK	constant	average
	type		no	no
	order		no	no
	duplicate		yes	yes
Recall Levels	10	52.22	54.74	55.51
	20	38.52	41.23	42.36
	30	31.90	33.10	34.35
	40	24.49	24.85	24.86
	50	21.01	21.76	21.94
	60	17.59	19.37	19.46
	70	12.13	13.52	13.54
	80	10.23	11.81	11.85
	90	7.04	8.28	8.29
	100	6.09	7.21	7.23
	Average	22.12	23.59	23.94
	% Increase		6.6	8.2

Table_8.5: Exp_5 The Average Weight Retrieval Results

8.2.6 Experiment_6: Sum Weight Strategy

The next weighting scheme for the multi-place predicates to try is the sum weighting scheme which calculates the weight of a multi-place predicate as the sum of the predicates involved in the relationship. This weighting scheme gives more weighting or importance to the occurrences of multi-place predicates. The retrieval strategy of Exp_6 is performed with this weighting scheme and the results obtained are shown in Table_8.6. The results obtained show an increase of the average precision by 4.3% over the benchmark. Thus, we can conclude that this weighting scheme is not as good as Fagan's average or Smeaton's constants scheme.

Strategies:			Exp_3	Exp_5	Exp_6
Parameters	weight	BENCHMARK	constant	average	sum
	type		no	no	no
	order		no	no	no
	duplicate		yes	yes	yes
Recall Levels	10	52.22	54.74	55.51	54.29
	20	38.52	41.23	42.36	40.67
	30	31.90	33.10	34.35	34.01
	40	24.49	24.85	24.86	24.75
	50	21.01	21.76	21.94	22.02
	60	17.59	19.37	19.46	18.47
	70	12.13	13.52	13.54	12.45
	80	10.23	11.81	11.85	10.73
	90	7.04	8.28	8.29	7.25
	100	6.09	7.21	7.23	6.96
Average		22.12	23.59	23.94	23.08
% Increase			6.6	8.2	4.3

Table_8.6: Exp 6 Sum Weight Retrieval Results

8.2.7 Experiment_7: Counting the Types and Orders of Dependencies

In experiment Exp_5, each dependency in a query is only checked for its absence or presence in the documents regardless of the existence

of different types of its dependencies and different orders of its arguments. For example, in matching a dependency r(x,y) in a query to a document which contains r(x,y), r(y,x) and a(x,y), only one match is considered by the retrieval strategy such as the one in Exp_5.

However, in the above example, the number of matches can be considered as three if we take into consideration the type of dependencies and the order of arguments, i.e. r(x,y), r(y,x) and a(x,y). Thus, the next retrieval strategy to try is similar to the strategy used in Exp_5, but with the counting of the types and orders included. The results obtained for this retrieval is shown in Table_8.7. The average precision obtained is 25.86 which is an increase of 16.8% over the benchmark. This is the highest precision that we got so far. The improvement in the results suggests that the existence of different kinds of relationships between a set of words in a document increases the importance of its dependencies in that document.

Strategies:			Exp_5	Exp_7
Parameters	weight	BENCHMARK	average	average
	type		no	no(+count)
	order		no	no(+count)
	duplicate		yes	yes
Recall Levels	10	52.22	55.51	57.44
	20	38.52	42.36	44.62
	30	31.90	34.35	37.43
	40	24.49	24.86	27.47
	50	21.01	21.94	24.76
	60	17.59	19.46	20.98
	70	12.13	13.54	15.06
	80	10.23	11.85	13.20
	90	7.04	8.29	9.56
	100	6.09	7.23	8.04
Average		22.12	23.94	25.86
% Increase			8.2	16.8

Table_8.7: Exp_7 Counting of Types and Orders Results

Figure_8.1 shows the precision recall curves obtained from the results of Exp_5 and Exp_7 which show improvements occurring at all levels of recall over the benchmark.

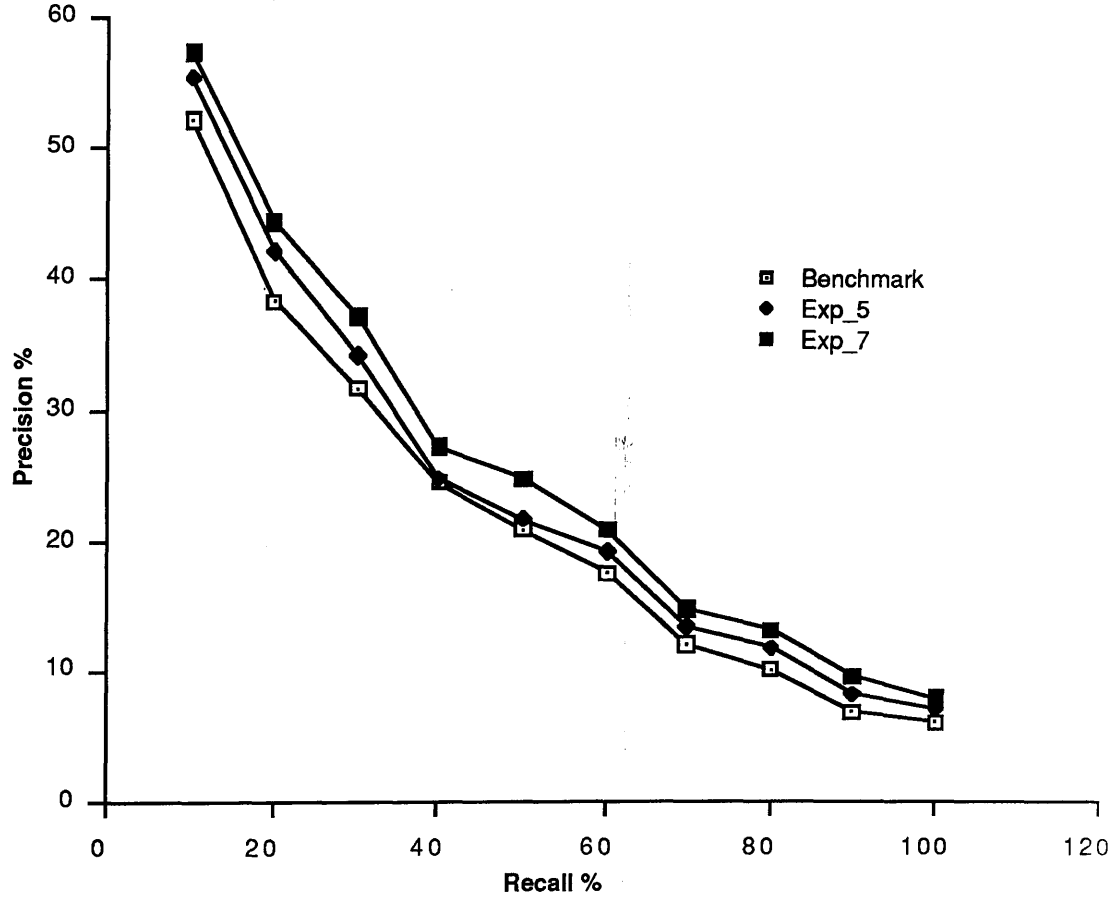


Figure_8.1: Precision recall curves of Exp-5 and Exp-7

8.2.8 Experiment_8: Transitive Dependency

If there are dependencies between x and y and between y and z in a document, such as r(x,y) and a(y,z), then x and z are said to be transitively dependent. The next retrieval strategy to try is the one that

takes into consideration this kind of one level transitive dependencies. The implication rules used in this experiment are the same as Exp_7 but with the addition of an implication rule for the transitive dependency. The results obtained by this strategy are given in Table_8.8 which show an improvement of only 4.6% over the benchmark. The strategy in Exp_7 which does not consider transitive dependency performs far better than the current strategy. This means that the transitive dependency does not portray the true dependency.

Strategies:			Exp_5	Exp_7	Exp_8
Parameters	weight	BENCHMARK	average	average	average
	type		no	no(+count)	no(+count)
	order		no	no(+count)	no(+count)
	duplicate		yes	yes	yes
	others		-	counting	counting & transitive
Recall Levels	10	52.22	55.51	57.44	56.39
	20	38.52	42.36	44.62	41.62
	30	31.90	34.35	37.43	35.42
	40	24.49	24.86	27.47	26.01
	50	21.01	21.94	24.76	22.58
	60	17.59	19.46	20.98	17.17
	70	12.13	13.54	15.06	11.47
	80	10.23	11.85	13.20	9.78
	90	7.04	8.29	9.56	6.04
	100	6.09	7.23	8.04	4.9
Average		22.12	23.94	25.86	23.14
% Increase			8.2	16.8	4.6

Table_8.8:Exp 8 Transitive Dependency Results

8.9 Experiment_9: Retrieval with a set of synonyms

The uncertain implication language, UNIL, provides the facility to define synonyms to be used with any retrieval strategy. Based on the experimental queries, we have defined a set of synonyms to be used with the retrieval strategy of Exp_7. We know that the formulation of these rules is ad hoc and biased, and therefore, the results obtained may

not reflect the true situation when an operational thesaurus is used. But this experiment is performed just to demonstrate the usage of synonyms in our system. The synonym rules are defined as follows:

1. *tss* is an abbreviation of *time sharing system*:
   r: system(X):- q:time(Y), q:sharing(Z), q:r(Y,X), q:(Z,X), d:tss(_).

2. *intermediate language* is synonymous to *intermediate code*:

   language(X):- q:intermediate(Y), q:a(Y,X), d:intermediate(A),
   d:code(B), d:(A,B).

3. *lexical analysis, syntax analysis, code generation* and *optimisation* are components of a *compiler*:
   compiler(_):- (d:lexical(A),d:analysis(B),d:a(A,B))
   @(d:syntax(C),analysis(D),r(C,D)) @
   (d:code(E),d:generation(F),r(E,F)) @
   (d:code(G),d:optimization(H),r(G,H)).

4. *monitor, semaphore, guard,* and *synchronization* are relevant to *communicating processes*:
   communicating(X):- q:processes(Y), q:r(X,Y),(d:monitor(_) @
   d:semaphore(_) @ d:guard(_) @ d:synchronization(_)).

5. *security* is synonymous to *privacy, cryptography, protection* and *encryption*:
   security(_):- d:privacy(_)@d:cryptography(_) @ d:protection(_) @
   d:encryption.

6. *parallel* is synonymous to *concurrence* and *synchronization.*
   parallel(_):- d:concurrence(_) @ d:synchronization(_).

7. *microcode* is synonymous to *low level code*:

```
microcode(_):- d:low(X), d:level(Y), d:code(Z), d:r(X,Z), d:r(Y,Z).
```

8. *modelling* is synonymous to *simulation*:

```
modelling(_):- d:simulation(_).
```

9. *paging* and *fragmentation* are components of *memory management*:

```
memory(X):- q:management(Y), q:r(X,Y), (d:paging(_) @
 d:fragmentation(_)).
```

10. *ELI* is synonymous to *EL1*, and *ELII* to *EL2*:

```
eli(_):- el1(_).
elii(_):- el2(_).
```

The results obtained for this experiment with synonyms are given in Table_8.9 which shows an improvement of 24.3% when compared to the 16.8% achieved by the retrieval without synonyms. This experiment shows that with a set of synonyms properly defined by an expert in the domain knowledge can improve the effectiveness of retrieval tremendously.

Strategies:			Exp_5	Exp_7	Exp_9
Parameters	weight	BENCHMARK	average	average	average
	type		no	no(+count)	no(+count)
	order		no	no(+count)	no(+count)
	duplicate		yes	yes	yes
	others		-	counting	counting & synonyms
Recall Levels	10	52.22	55.51	57.44	58.74
	20	38.52	42.36	44.62	45.64
	30	31.90	34.35	37.43	38.06
	40	24.49	24.86	27.47	28.64
	50	21.01	21.94	24.76	26.00
	60	17.59	19.46	20.98	22.99
	70	12.13	13.54	15.06	17.68
	80	10.23	11.85	13.20	15.62
	90	7.04	8.29	9.56	11.55
	100	6.09	7.23	8.04	10.14
Average		22.12	23.94	25.86	27.51
% Increase			8.2	16.8	24.3

Table_8.9:Exp_9 Retrieval with Synonyms Results

8.3 Conclusion

All strategies experimented above have shown better performances than the tf x idf benchmark. Our best retrieval strategy from Exp_7 has produced an increase of 16.8% in average precision over the benchmark. This figure is very encouraging when compared with the figures obtained by Smeaton and Fagan which are 5.07% and 8.7% respectively. The level of improvement we obtained can be considered as significant, see the precision recall curves in Figure_8.2. When the same retrieval strategy is used with a small set of synonyms in Exp_9, the increase in average precision obtained is 24.3%. Although the set of synonyms used is biased to the queries experimented with, the results obtained suggest that there is scope for further improvement. Figure_8.2 also shows the precision recall curve obtained from Exp_9.

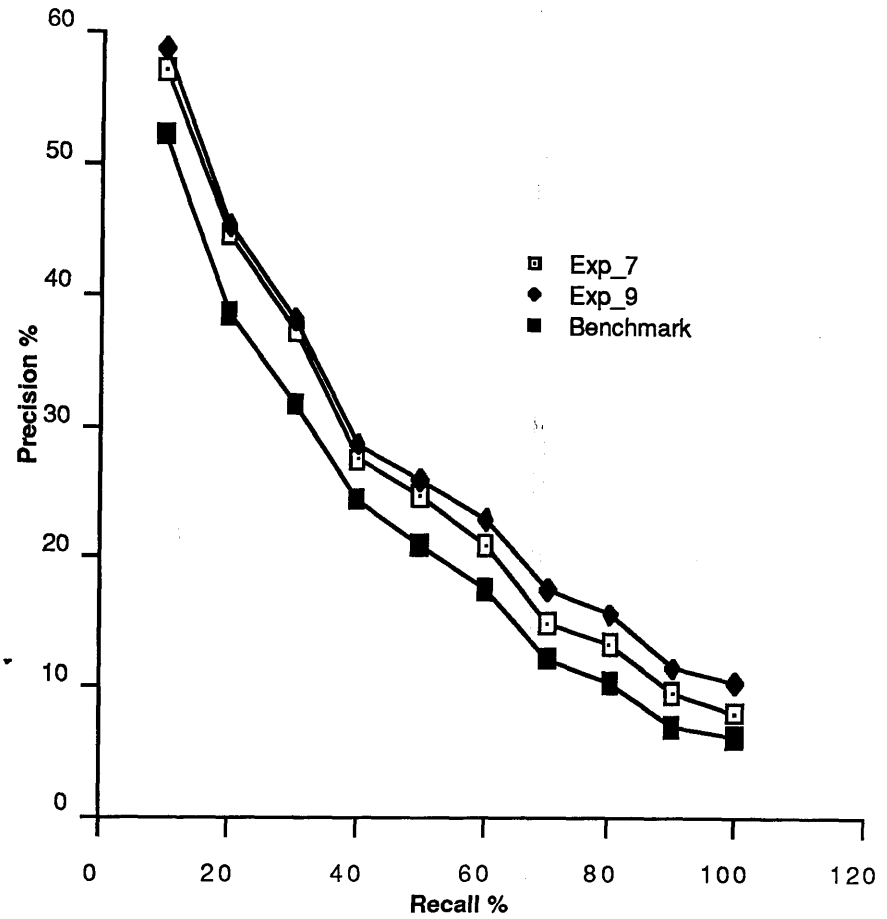


Figure_8.2: Precision Recall Curves

Table_8.10 shows the results of the Document Cutoff evaluation performed on the best four retrieval strategies carried out in the above experiments, i.e. Exp_3, Exp_5, Exp_7, and Exp_9. The results of this evaluation confirm and strengthen the results obtained from the recall cutoff evaluation regarding the grading of the effectiveness of each strategy. One obvious improvement of the retrieval with synonyms over the others is in the Qfail figures, i.e. the number of queries which fail to retrieve any relevant document at each cutoff rank position. At cutoff 30, the value of Qfail is 0 for this strategy. This is probably what caused the large increase in the average precision obtained by this

strategy over the others.

		Number of Documents Retrieved & Relevant (Qfail*)				
		Benchmark	Exp_3	Exp_5	Exp_7	Exp_9
Parameter	weigth		constant	average	average	average
	type		no	no	no	no
	order		no	no	no	no
	dup		yes	yes	yes	yes
	other		-	-	count	count & synonyms
CUTOFF	5	74(12)	76(11)	75(11)	78(9)	76(7)
	10	111(4)	119(3)	120(3)	129(2)	131(1)
	15	155(2)	156(2)	156(2)	177(2)	176(1)
	20	188(2)	188(2)	189(2)	207(2)	208(1)
	25	203(2)	207(2)	210(2)	231(2)	233(1)
	30	228(2)	229(2)	232(2)	255(1)	255(0)
	40	255(2)	261(2)	261(2)	288(1)	284(0)
	50	281(2)	284(2)	285(2)	315(1)	315(0)
	100	360(1)	365(1)	364(1)	397(1)	399(0)
	200	458(1)	460(1)	460(1)	493(1)	504(0)
	400	558(1)	558(1)	558(1)	553(1)	563(0)
	600	598(1)	599(1)	599(1)	592(1)	603(0)
	800	621(1)	621(1)	621(1)	617(1)	630(0)
	1000	628(1)	628(1)	628(1)	624(1)	637(0)
% increase in average precision			6.6	8.2	16.8	24.3

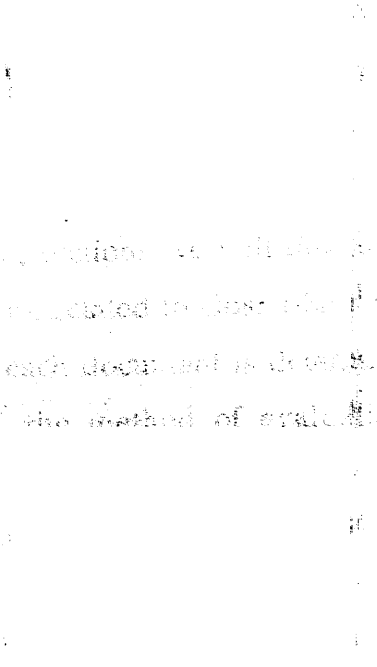
*Qfail = Number of queries which fail to retrieve any relevant document.

Table_8.10: Document Cutoff evaluation on the best four strategies

Chapter 9: Nearest Neighbour Relevance Feedback Experiments

9.1 Introduction

After performing non-feedback experiments on various retrieval strategies as described in Chapter 8, the second part of the thesis is to perform experiments using relevance feedback with nearest neighbour information. The retrieval strategy of Exp_7 is chosen to perform the initial ranking of documents with respect to each query, and from this ranking a few, say C, top ranked documents will be retrieved and identified as relevant or not by the user. C is the cutoff point which can be varied and experimented upon to obtain the best possible value. The C top ranked documents which are judged as relevant will be referred to as 'retrieved and relevant' documents or simply as RR. From this set of retrieved and relevant documents, we can obtain all the unretrieved documents which have any of the RR as their nearest neighbour as illustrated in Figure_9.1. This set of unretrieved documents have the potential of also being relevant since they are 'close' to the RR, and they will be referred to as potentially relevant documents or simply as PR.



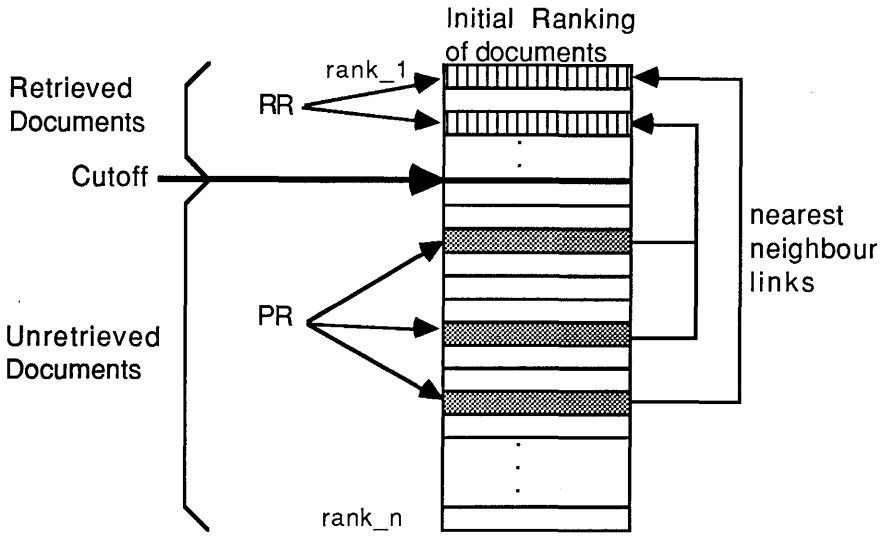


Figure_9.1: Determining the potentially relevant documents (PR)

The nearest neighbours for each document are determined using dissimilarity coefficients or 'distance' measurements between each document. The initial similarity values or scores of the PR to the query will be updated according to their distances from the RR. With these updated scores a new ranking of documents is obtained and evaluated. This relevance feedback retrieval strategy is an implementation of the *imaging* retrieval as defined and given theoretical foundation by van Rijsbergen in [van Rijsbergen 1989]. The strategy in our implementation focuses on looking for 'images' of the retrieved and relevant documents from among the unretrieved documents.

In the following sections we will discuss how our experiments on imaging retrieval are related to cluster-based retrieval, how the nearest neighbour set for each document is determined, how the scores of PR are updated, and the method of evaluation which determines the effectiveness of the imaging retrieval. Finally, we will discuss various experiments carried out within this strategy and the results obtained.

9.2 Cluster-based retrieval

Van Rijsbergen proposed the cluster hypothesis which simply states that *closely associated documents tend to be relevant to the same requests* [van Rijsbergen 79]. Clustering is a process of picking out closely associated documents and grouping them together into one cluster. It is assumed that clustering of documents will increase the level of retrieval effectiveness. Croft has given the following simple example to support this assumption [Croft 1978]. A relevant document, because of an error in the indexing process, may lack a term which would cause it to be retrieved. However, if the documents are clustered, this document may be clustered with other relevant documents that do have the required term. Therefore, it could be retrieved by a cluster search.

One example of clustering algorithms is the single-link as experimented with by [van Rijsbergen 1972] and [Croft 1978]. The basic input to the single-link clustering algorithm is the dissimilarity coefficient and the output is a hierarchy with associated numerical levels called a dendrogram. The hierarchy can be represented by a tree structure such that each node represents a cluster, as illustrated in Figure_9.2. The numerical level of a cluster can be defined as follows: if a cluster has a numerical level of L, this implies that every document in that cluster must have a dissimilarity value less than L for at least one other document in the cluster.

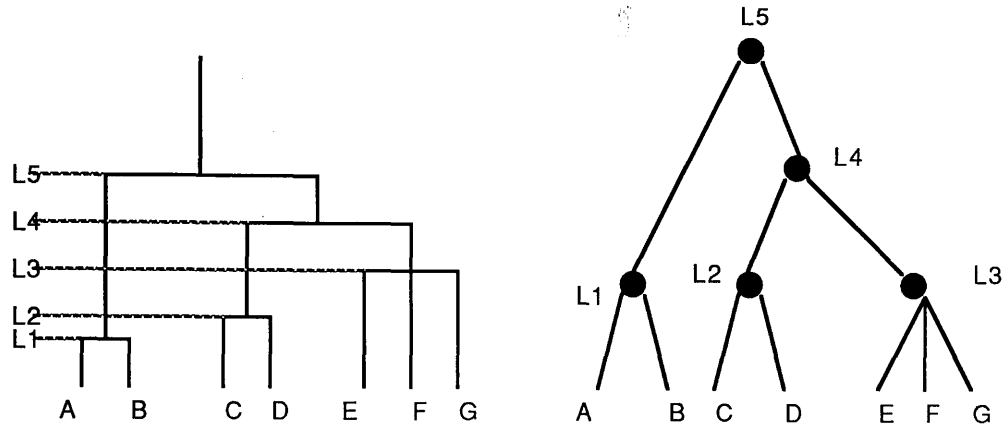


Figure_9.2: A dendrogram with corresponding tree

There is a variety of search strategies that can be performed on the hierarchy. The search strategies used for cluster-based retrieval can be grouped into two types: top-down and bottom-up. In a top-down search strategy, the search starts at the root of the tree, i.e. node_0 as illustrated in Figure_9.3. The search proceeds by evaluating a matching function at the node immediately descendent from node_0, i.e. node_1 and node_2 in Figure_9.3. This process repeats itself down the tree. The search is directed by a decision rule and a stopping rule. The decision rule decides at each stage which node is to be expanded further according to the values of the matching function. The stopping rule decides when the search should be terminated and retrieval should be performed. The decision rule and the stopping rule used in the example as illustrated in Figure_9.3 are as follows:

decision rule: chose the node with the maximum value of the matching function obtained within a filial set.

stopping rule: stop when the current maximum is less than the previous maximum.

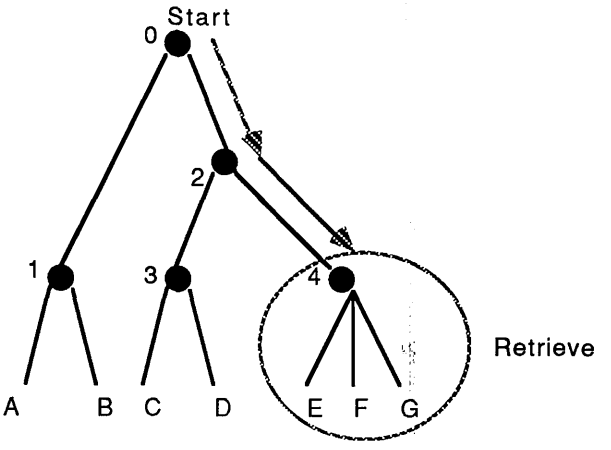


Figure_9.3: A Top-down search strategy

A bottom-up search is one which starts the search at one of the terminal nodes and proceeds upwards towards the root. A decision rule is not required in this search strategy. But a stopping rule is needed and it can be simply a cut-off. A typical search is to find and retrieve the largest cluster containing the document represented by the starting node and not exceeding the cut-off in size. To start the bottom-up search, it is necessary to know an appropriate document with respect to the query. This may be an identification of a relevant document given by the user. Figure_9.4 illustrates a bottom-up search strategy. Croft has performed an evaluation of bottom-up searches in terms of efficiency and effectiveness in [Croft 1978] and he has concluded that the bottom-up search is better than the conventional serial/inverted search.

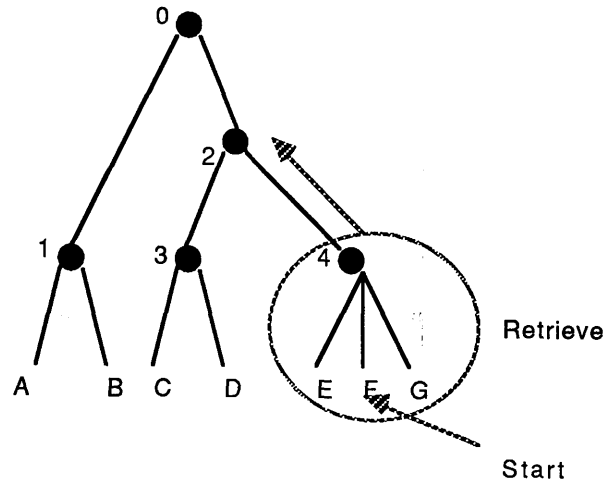


Figure_9.4: A bottom-up search strategy

In our experiments on relevance feedback using nearest neighbour information, for each document D there is a cluster or set of documents which are identified as documents having D as one of their nearest neighbours, as illustrated in Figure_9.5. The distance information between the neighbours is also included in these clusters which will be used for further processing. The relevance feedback information, in terms of identifications of some relevant documents, obtained from the user, will select the clusters which are identified with those relevant documents to be looked into for further processing. The selected clusters are actually the potentially relevant documents, i.e. the PR set.

The data structure given in Figure_9.5 which is used to represent the nearest neighbour information has made the process of determining the PR set much easier. By knowing the RR set, through the relevance feedback, we can directly determine the PR set. This data structure will be discussed further in the next section.

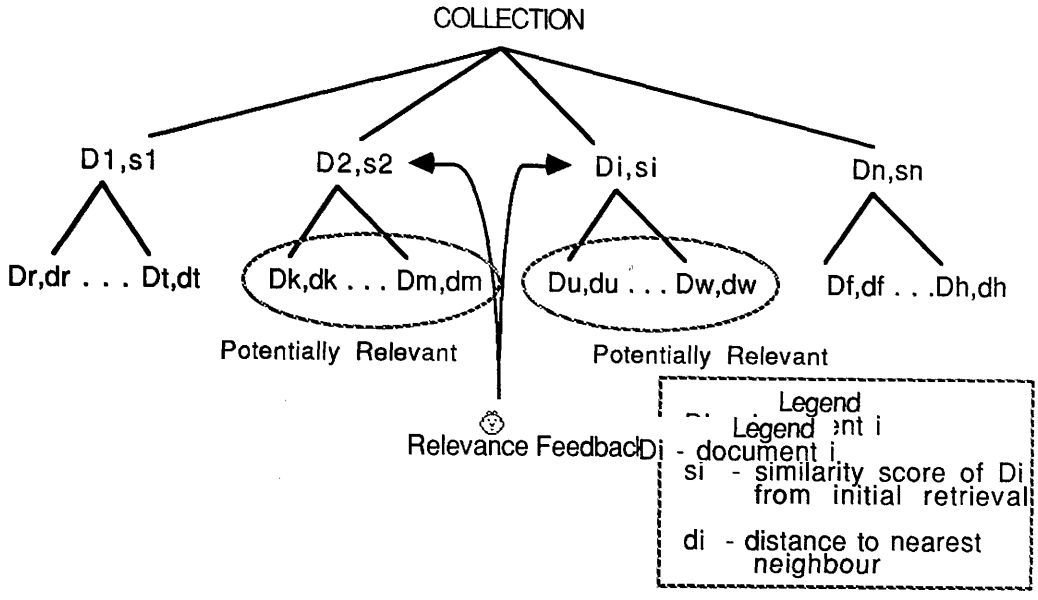


Figure_9.5: Relevance Feedback with Nearest Neighbours

9.3 Definition and Determination of Nearest Neighbours

We define nearest neighbours of a document as a set of documents whose members satisfy certain conditions based on their 'distances' to the document concerned. The distances between documents are measured using dissimilarity coefficients calculated using Dice's coefficient [van Rijsbergen 79]. The dissimilarity coefficient between two documents represented by X and Y are calculated as below:

$$\begin{aligned} \text{dissimilarity coefficient} &= 1 - \text{Dice's coefficient} \\ &= 1 - 2|X \cap Y| / (|X| + |Y|) \end{aligned}$$

where $|A|$ gives the size of the set A. A dissimilarity coefficient denotes a measure of distance between two documents. Our determination of nearest neighbour sets is based on this distance measurement. The following example illustrate how dissimilarity coefficients are calculated in our system. Given two documents X and Y

in our predicate representation form:

X :

[1988(V),undergraduate(W),curriculum(X),computer(Y),science(Z),r(W,X),in(X,Z)]

Y :

[1987(A),undergraduate(B),computer(C),science(D),curriculum(E),r(D,E),r(C,E),r(B,E)]

the results of the intersection between X and Y, which is computed using unification process as illustrated in Figure_9.6, is :

$X \cap Y$:

[undergraduate(W),curriculum(X),computer(Y),science(Z),r(W,X)]

Thus, the dissimilarity coefficient or distance between X and Y is :

$$1 - 2|X \cap Y| / (|X| + |Y|)$$
$$= 1 - 2(5/(7+8))$$
$$= 1 - 0.67$$
$$= 0.33$$

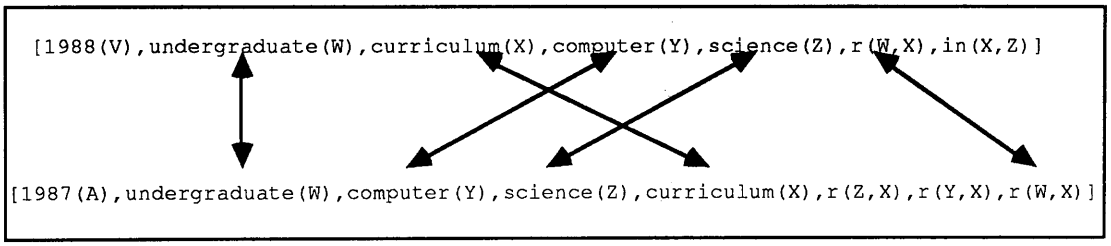


Figure_9.6: $X \cap Y$

In performing the intersection operation between documents, implication rules for matching the multi-place predicates have to be defined. Due to execution time factor, the calculation of dissimilarity

coefficients between documents is implemented in C and the implication rules adopted are equivalent to the *non-typed and ordered* parameters as defined for retrieval strategies. In fact, all programs that implement imaging retrieval are written in C.

There are many ways by which the nearest neighbour set of each document can be determined, for example:

1) by selecting only the closest documents to the document concerned,
2) by setting a threshold value on the distance and limiting the number of nearest neighbours for each document.

In our experiments, we have used two methods for determining nearest neighbour sets. They are:

1. The Closest Nearest Neighbours.
The nearest neighbour set of a document is formed by selecting only the documents with the shortest distance to it. There may be more than one documents with the shortest distance.
To illustrate how the nearest neighbour information is represented in our implementation, assume that the collection contains n documents: D1, D2, . . . , Dn. If the closest nearest neighbour sets of some documents are given as follows :

document	closest nearest neighbour set with distance
D1	{D3,0.1}
D2	{D7,0.2; D9,0.2}
:	:
D10	{D3,0.4; D7,0.4; D1002,0.4}
:	:
Dn	{D1,0.3}

then, based on the structure in Figure_9.5, the nearest neighbour information represented in our system for the above collection is depicted in Figure_9.7.

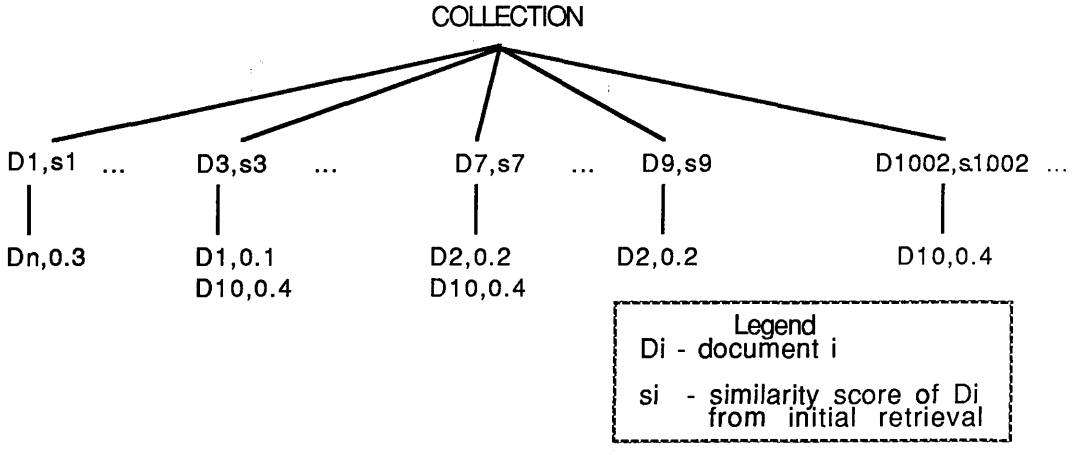


Figure_9.7: The representation of the closest nearest neighbour information

2. The Ten Nearest Neighbours.

The sizes of the closest nearest neighbour sets determined by the above method may be too small and may not alter the initial

ranking of the documents much. Thus, we decided to increase their sizes by selecting at least the first ten closest documents instead of selecting only the closest documents into the nearest neighbour sets. The size can be greater than ten if the number of the closest nearest neighbours is more than ten, and the size can be less than ten if the number of the neighbours with distances less than 1 is less than ten.

The representation of the nearest neighbour information in the case of using the ten nearest neighbours is similar to the case when the closest nearest neighbours are used. Assume the collection contains the same n documents as in the example above, and the ten nearest neighbour sets of some documents are given as follows :

document	closest nearest neighbour set with distance
D1	{D3,0.1; D7,0.2; D5,0.9}
D2	{D7,0.2; D9,0.2; D3,0.3; D21,0.4; D1,0.4; D4,0.5; D5,0.5; D6,0.6; D8,0.7; D1002,0.9}
:	:
D10	{D3,0.4; D7,0.4; D1002,0.4; D5,0.9}
:	:
Dn	{D1,0.3}

Then, the representation of the nearest neighbour information for this example is given in Figure_9.8.

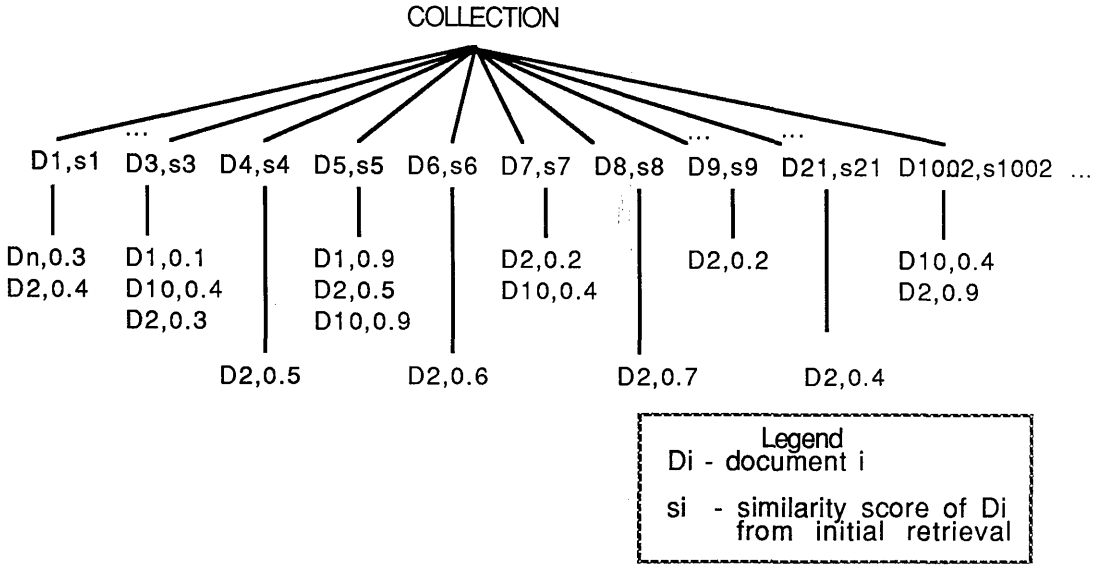


Figure_9.8: The representation of the ten nearest neighbours information

9.4 Updating the Scores

After determining the set of the potentially relevant documents PR from the nearest neighbour information by knowing the RR documents, their scores have to be updated in order to produce the new ranking of the documents. The question now is "by how much should their scores be increased?". In what follows, we will describe one way of calculating this increment.

The initial ranking of the documents produces the following set of document identifications and their scores :

$$\text{Initial Ranking} = \{ D_1, \text{score}_1; D_2, \text{score}_2; \ldots ; D_n, \text{score}_n \}$$

where score_i is the score of the document D_i. From the initial ranking, we can obtain the C top ranked documents and thus produce the

relevant and retrieved set RR, in descending order of scores, as follows:

$$RR = \{ D_{r_1}, score_{r_1}; D_{r_2}, score_{r_2}; \ldots ; D_{r_l}, score_{r_l} \}$$

where $score_{r_i}$ is the score of document D_{r_i}. The highest score in this set is $score_{r_1}$, and for the sake of clarity, we rename this score as $score_{r_{max}}$. From this RR set we can obtain a set of potentially relevant documents PR whose members having at least a nearest neighbour which is a member of RR:

$$PR = \{ D_{p_1}, d_{p_1 i}; D_{p_2}, d_{p_2 j}; \ldots D_{p_m}, d_{p_m k} \}$$

where D_{p_i} is a potentially relevant document with a distance of $d_{p_i j}$ to its nearest neighbour D_j which is a member of RR. D_{p_i} and D_{p_k}, where $i \neq k$, may refer to the same document. If this happens, the increment value to the score will take the highest increment value of the two.

If D_r is a member of RR and D_p is a member of PR whose nearest neighbour is D_r, then a similarity diagram of D_r and D_p with respect to a query Q can be depicted as in Figure_9.9, where $(1\text{-}d_{pr})$ is the similarity value between D_p and D_r.

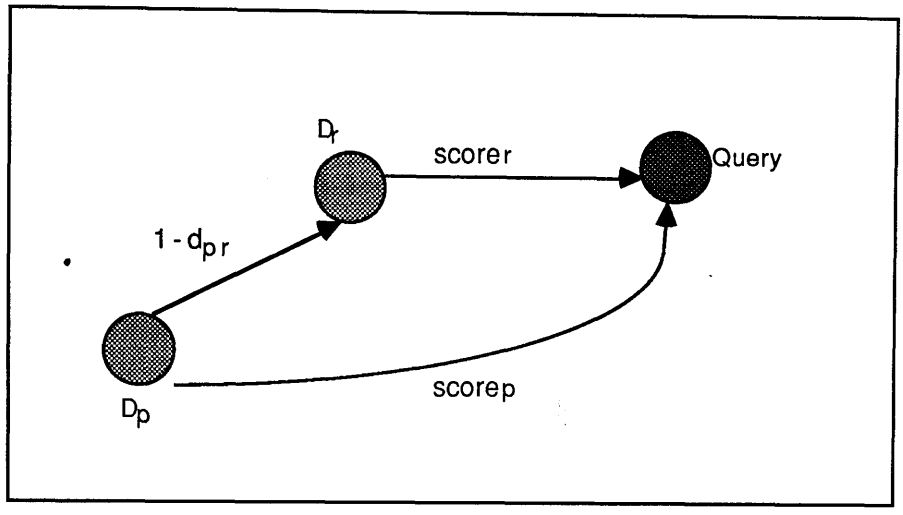


Figure_9.9: Similarity Diagram

One of the possible ways of updating the $score_p$ is by increasing it with a value calculated using the increment function as follows:

$$increment = K * [(1-d_{pr}) * score_r * (score_r / score_{r_{max}})]$$

where $(1-d_{pr})$ is the similarity coefficient between the documents D_p and D_r, $(score_r / score_{r_{max}})$ is the relative similarity coefficient of D_r to Q with respect to the highest score obtained by RR, and K is some constant greater than 0 and less than or equal to 1. Which means, the increment is some percentage of the score $score_r$ based on:

1) the distance between the potentially relevant document and the relevant document concerned, and
2) the importance of the relevant document concerned with respect to the other documents which are judged to be relevant.

9.5 Evaluating Imaging Retrieval

The method used to evaluate the effectiveness of our imaging retrieval is the residual ranking evaluation technique as described in [Harper 80] and [Salton&Buckley 88]. When comparing the initial ranking with the new ranking obtained after the feedback, the feedback documents which have already been seen by the user should be removed from both rankings. A ranking with the feedback documents removed is known as a residual ranking. The initial residual ranking is obtained by removing the C top ranked documents which have been judged by the user from the initial ranking. The new residual ranking is obtained by removing the same C documents as above from the new ranking generated after the feedback. Then, the precision-recall cutoff evaluation as described in Chapter 7 is performed on both residual rankings. In this residual evaluation technique, the queries that retrieve all the relevant documents within the C top ranked documents of the initial ranking are excluded from the evaluation, since there are no more relevant documents remaining to be retrieved.

The document cutoff evaluation as described earlier in Chapter_7 can also be performed on both the initial and the new rankings. This evaluation is less opaque than the precision-recall cutoff evaluation, since it gives the actual number of the relevant documents retrieved at specified rank positions summed over all the queries.

9.6 Experimental Results

In this section we will discuss the experimental results obtained in performing the imaging retrieval as described above.

9.6.1 Benchmark

The precision-recall cutoff evaluation is done on the initial residual ranking with the cutoff point C equal to 10, and the results obtained are given in Table_9.1. The average precision obtained is 15.81.

		Benchmark Precision
Recall Levels	10	37.39
	20	28.70
	30	22.36
	40	16.75
	50	14.41
	60	11.67
	70	9.69
	80	7.01
	90	5.36
	100	4.69
Average		15.81

Table_9.1: Benchmark with cutoff C=10

9.6.2 Experiment_A: Using Closest Nearest Neighbours

A few experiments are performed using the closest nearest neighbour sets with different increment functions as follows:

A1: $\text{increment} = 0.5 * (1\text{-}d_{pr}) * \text{score}_r * (\text{score}_r/\text{score}_{r_{max}})$

A2: $\text{increment} = (1\text{-}d_{pr}) * \text{score}_r * (\text{score}_r/\text{score}_{r_{max}})$

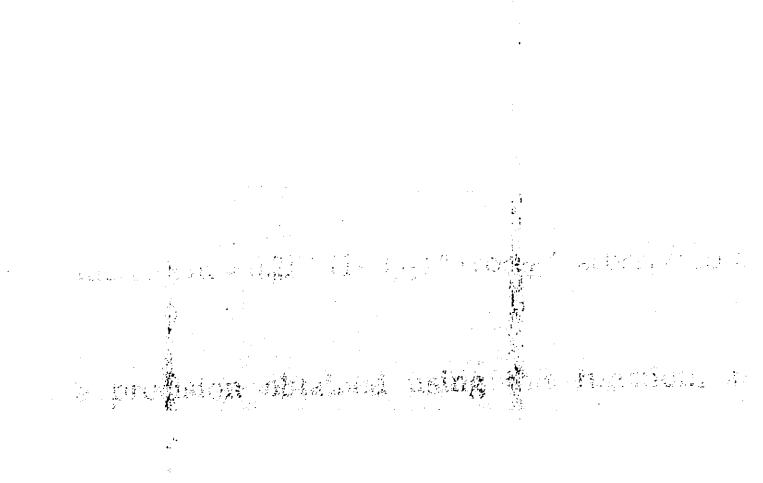
A3: $\text{increment} = (1\text{-}\ d_{pr}) * \text{score}_r$

The cutoff point C used in these experiments is 10. The purpose of this set of experiments is to find a suitable increment function. The results obtained from these experiments are shown in Table_9.2 with the average precisions for A1, A2 and A3 respectively as follows: 16.0,

15.54 and 15.33. The precision recall bar-chart of these results is shown in Figure_9.10. The results show that the function which gives the greater increment values produces a lower average precision. In fact, the increment function A2 and A3 give the average precision values which are worse than the benchmark. This is probably because the increment values added to the PR document scores are too big and many of the PR documents are not really relevant at all.

	Precisions			
		Increment Functions		
	B'chmark	A1	A2	A3
Recall Levels 10	37.39	38.25	40.01	42.34
20	28.70	28.91	28.53	27.98
30	22.36	22.41	21.32	21.53
40	16.75	17.07	15.96	16.00
50	14.41	14.54	13.78	14.29
60	11.67	11.99	10.88	10.62
70	9.69	9.67	8.61	8.54
80	7.01	6.96	5.89	5.87
90	5.36	5.32	4.25	4.23
100	4.69	4.82	3.98	3.96
Average	15.81	16.00	15.32	15.54
% Increase		1.2	-3.1	-1.7

Table_9.2: Results of Experiments using A1,A2 and A3



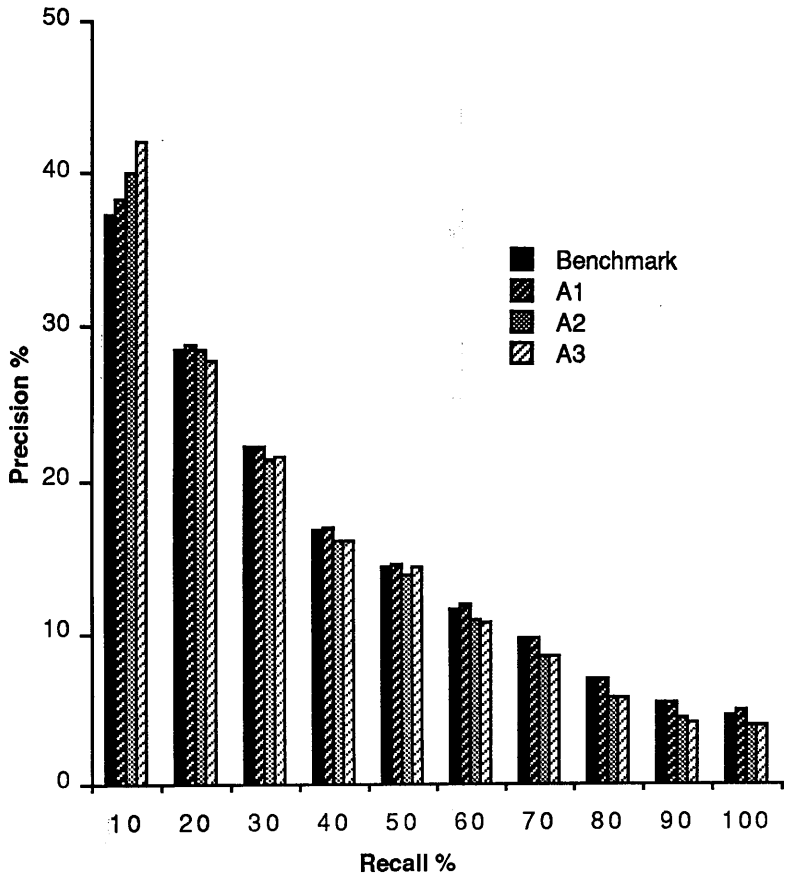


Figure_9.10: Precision recall bar-chart for the increment function experiments

The next experiment to carry out is with an increment function which gives lesser increment values than A1. The function is :

A4: $\text{increment} = 0.25 * (1 - d_{pr}) * \text{score}_r * (\text{score}_r / \text{score}_{r_{max}})$

The average precision obtained using this function, as shown in Table_9.3, is 15.89 which is inferior to the value obtained when using A1.

		Precisions		
		B'chmark	Incerement Functions	
			A1	A4
Recall Levels	10	37.39	38.25	37.50
	20	28.70	28.91	28.86
	30	22.36	22.41	22.36
	40	16.75	17.07	16.88
	50	14.41	14.54	14.52
	60	11.67	11.99	11.99
	70	9.69	9.67	9.69
	80	7.01	6.96	7.02
	90	5.36	5.32	5.36
	100	4.69	4.82	4.69
Average		15.81	16.00	15.89
% Increase			1.2	0.3

Table_9.3: The Results using A4

Looking at the results in Table_9.2, eventhough the average precisions obtained using the functions A2 and A3 are lower than the benchmark, but the precisions obtained by these functions at recall level 10% are higher than the precisions obtained by the benchmark and A1. This is probably the result of giving higher increment values to the PR, which has pushed the relevant documents among the PR further up the rank. A similar argument can be given to explain the drop in the average precision obtained by A2 and A3, that is, the non-relevant documents among the PR have been pushed too high up the ladder and caused the drop in the overall precision.

Document Cutoff Points:	Number of Documents Retrieved & Relevant			
	Benchmark	Increment Function:		
		A1	A2	A3
10	129	-	-	-
15	177	179	181	186
20	207	213	217	220
25	231	238	235	237
30	255	262	263	263
40	288	290	291	294
50	315	317	319	320
100	397	397	398	398
200	493	494	495	498
400	553	555	554	555
600	592	592	593	593
800	617	618	618	618
1000	624	625	625	625

Table_9.4: Document Cutoff Evaluation for the increment function experiments

The document cutoff evaluation on the results obtained in the above experiments is given in Table_9.4. This evaluation has shown that for document cutoff points at 50 and below, A3 has outperformed A2 and A2 has outperformed A1. But the performance seems to level up at higher document cutoff points, and based on the precision-recall figures, A1 will finally outperform A2 and A3. Thus the increment function A3 seems to be the best choice for an operational system, but for our experimental purpose we will chose A1 since it has given a constant improvement over the benchmark at almost all the levels of recall and in the overall average precision.

9.5.3 Experiment_B: Using Ten Nearest Neighbours

It is felt that the number of documents in the PR set may be too small to be able to have many really relevant documents which could significantly alter the initial ranking and produce better results. If the size of the PR set is increased then there is a better chance that it will contain more relevant documents, but the chance of having more non-relevant documents also increases. In order to increase the size of the PR set, we have to increase the number of documents in the nearest neighbour sets. Thus, in the following experiments we have used the ten nearest neighbour sets with the following increment functions:

B1: $\text{increment} = 0.5 * (1-d_{pr}) * score_r * (score_r / score_{r_{max}})$

B2: $\text{increment} = (1-d_{pr}) * score_r * (score_r / score_{r_{max}})$

		Precisions			
			Increment Functions		
		Benchmark	A1	B1	B2
Recall Levels	10	37.39	38.25	38.46	36.93
	20	28.70	28.91	29.79	28.01
	30	22.36	22.41	22.67	21.51
	40	16.75	17.07	15.84	16.10
	50	14.41	14.54	13.73	13.41
	60	11.67	11.99	11.33	10.87
	70	9.69	9.67	8.53	8.58
	80	7.01	6.96	6.13	5.90
	90	5.36	5.32	4.48	4.25
	100	4.69	4.82	3.9	3.58
Average		15.81	16.00	15.49	14.92
% Increase			1.2	-2.0	-5.6

Table_9.5: Results of Experiments using B1 and B2.

The results of the experiments using the above two increment functions are given in Table_9.5 which show that the average precision

obtained for the functions B1 and B2 are 15.49 and 14.92, respectively. These results are worse than the benchmark. The reason is, probably, that there are too many non-relevant documents in the PR set. In order to filter some of them out, we can impose a threshold value on their distance measurements. The following two increment functions are introduced to impose this threshold value:

B3: If $d_{pr} < 0.7$ then

increment = $0.5 * (1-d_{pr}) * score_r * (score_r / score_{r_{max}})$

else increment = 0.

B4: If $d_{pr} < 0.6$ then

increment = $0.5 * (1-d_{pr}) * score_r * (score_r / score_{r_{max}})$

else increment = 0.

	Precisions			
		Increment Functions		
	B'chmark	A1	B3	B4
Recall Levels 10	37.39	38.25	37.71	37.56
20	28.70	28.91	28.97	28.87
30	22.36	22.41	22.47	22.53
40	16.75	17.07	17.02	16.92
50	14.41	14.54	14.52	14.41
60	11.67	11.99	11.99	11.67
70	9.69	9.67	9.69	9.69
80	7.01	6.96	7.01	7.01
90	5.36	5.32	5.36	5.36
100	4.69	4.82	4.69	4.69
Average	15.81	16.00	15.94	15.87
% Increase		1.2	0.8	0.4

Table_9.6: Results of Experiments using B3 and B4

The average precisions obtained for B3 and B4 are 15.94 and 15.87, respectively, as shown in Table_9.6. Although there is some improvement over the results of B1 and B2, the results obtained are no

better than when using the closest nearest neighbour sets with the increment function A1. Therefore, we can conclude that it is better to use the closest nearest neighbour sets because the results obtained are slightly better and the amount of calculations involved is less.

The best results obtained so far in our relevance feedback retrieval is when using the closest nearest neighbour sets and the increment function A1. The average precision obtained is 16.0 which is an increase of 1.2% over the benchmark. Although the improvement is not very significant, it shows that this sort of relevance feedback retrieval strategy is viable and valid. The level of improvement might be increased by doing further experiments in order to find the more effective increment function and the cutoff point.

9.6.4 Experiment_C: Cutoff Point Experiments

In order to investigate the performance of various cutoff points in imaging retrieval, experiments with cutoff points at 5, 10, 15, and 20 are performed and their results are evaluated. The results of the residual ranking evaluation are given in Table_9.7. The results have shown that the cutoff point with higher value gives greater improvement over its respective benchmark. The cutoff point at 5 shows a negligible difference from its benchmark, in fact, it shows a decrease of 0.2% in average precision over the benchmark. As the value of the cutoff point is increased to 10, 15 and 20 the respective improvements in average precision obtained are 1.2, 1.14 and 9.6. The results obtained suggest that there is a relationship between the performance and the cutoff points, i.e. higher cutoff points tend to give better results.

	Precisions							
	cutoff = 5		cutoff = 10		cutoff = 15		cutoff = 20	
Recall Levels	Benchmark	new ranking	Benchmark	new ranking	Benchmark	new ranking	Benchmark	new ranking
10	38.33	37.35	37.39	38.25	24.46	27.03	23.95	28.19
20	29.86	29.82	28.70	28.91	20.55	20.82	14.56	16.97
30	22.87	23.09	22.36	22.41	17.19	17.60	11.77	12.63
40	19.00	19.20	16.75	17.07	14.28	14.41	9.13	9.26
50	16.20	16.47	14.41	14.54	11.63	11.22	7.37	7.40
60	11.16	11.29	11.67	11.99	9.12	8.77	4.89	4.97
70	8.97	8.95	9.69	9.67	5.73	5.35	3.78	3.79
80	6.10	6.04	7.01	6.96	4.69	4.30	2.91	2.91
90	4.31	4.28	5.36	5.32	3.35	2.96	2.03	2.03
100	3.47	3.63	4.69	4.82	2.71	2.49	1.28	1.48
Average	16.03	16.02	15.81	16.00	11.37	11.50	8.17	8.96
% Increase		-0.06		1.2		1.14		9.6

Table_9.7: Residual Ranking Evaluation Results

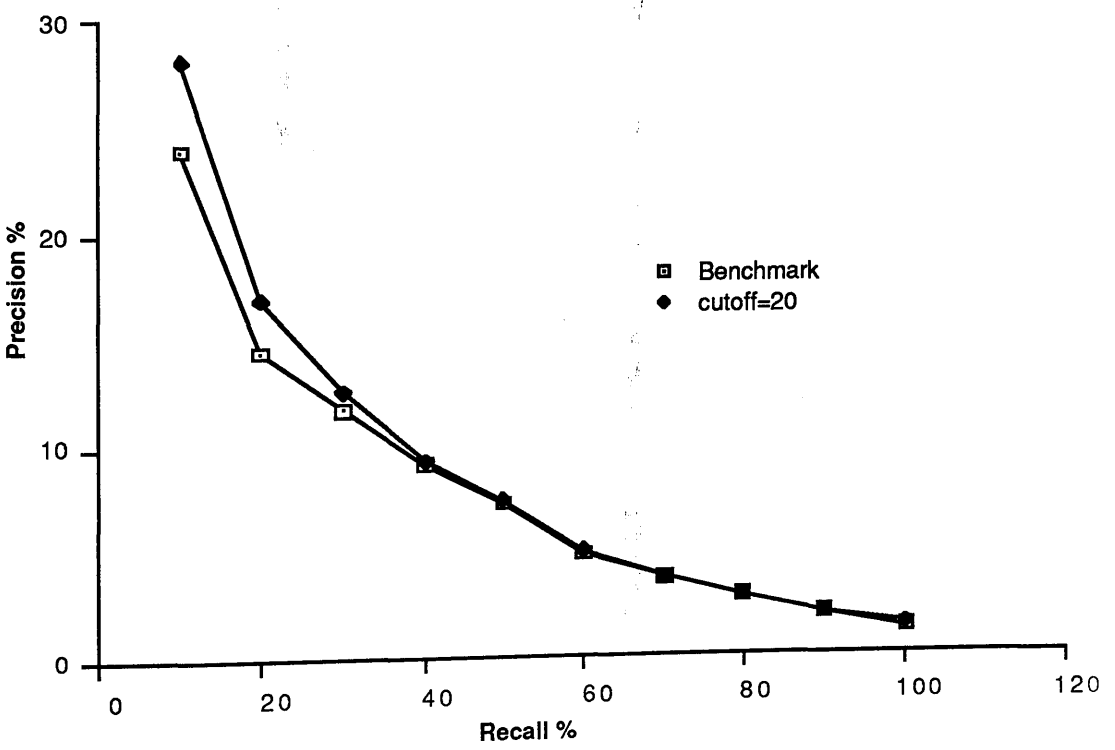


Figure_9.11: Precision recall curve of experiment with cutoff=20

Figure_9.11 compares the precision recall curve obtained from the experiment with cutoff point at 20 to the benchmark's curve. There are large differences in precisions at the lower recall levels, but the differences are negligible at the level of recall 40% onwards.

Table_9.8 shows the results of the document cutoff evaluation performed on the results of the above experiments. As suggested in the above analysis, the number of retrieved and relevant documents obtained are slightly higher in the experiment with higher value of cutoff point.

	Number of Documents Retrieved & Relevant				
Document Cutoff Points:	B'chmark	Imaging Retrieval with cutoff C at:			
		5	10	15	20
5	78	-	-	-	-
10	129	130	-	-	-
15	177	179	179	-	-
20	207	212	213	214	-
25	231	236	238	238	239
30	255	261	262	262	262
40	288	290	290	290	290
50	315	316	317	318	319
100	397	397	397	396	396
200	493	494	494	494	494
400	553	554	555	555	555
600	592	592	592	592	592
800	617	617	618	618	618
1000	624	625	625	626	626

Table_9.8: Document Cutoff Evaluation for cutoff experiments

9.6.5 Experiment_D: Multi-stage Imaging Retrieval

In the above experiments we have performed only a one-stage relevance feedback retrieval as depicted in Figure_9.12, i.e. the user only gives the feedback on the relevant documents once. The number of stages could be increased to n, where the user gives feedback on the relevant documents n times and there are n times of reranking of the documents as depicted in Figure_9.13. At each stage the user will be given the C top ranked documents to be judged. The user will decide which documents are relevant and which are not. With this feedback information, a new residual ranking is obtained and again the C top ranked documents from the new residual ranking will be given to the user for relevance judgement. The relevance judgement process will be repeated n times for n-stage relevance feedback retrieval. This n-stage relevance feedback retrieval will be referred to as multi-stage imaging retrieval.

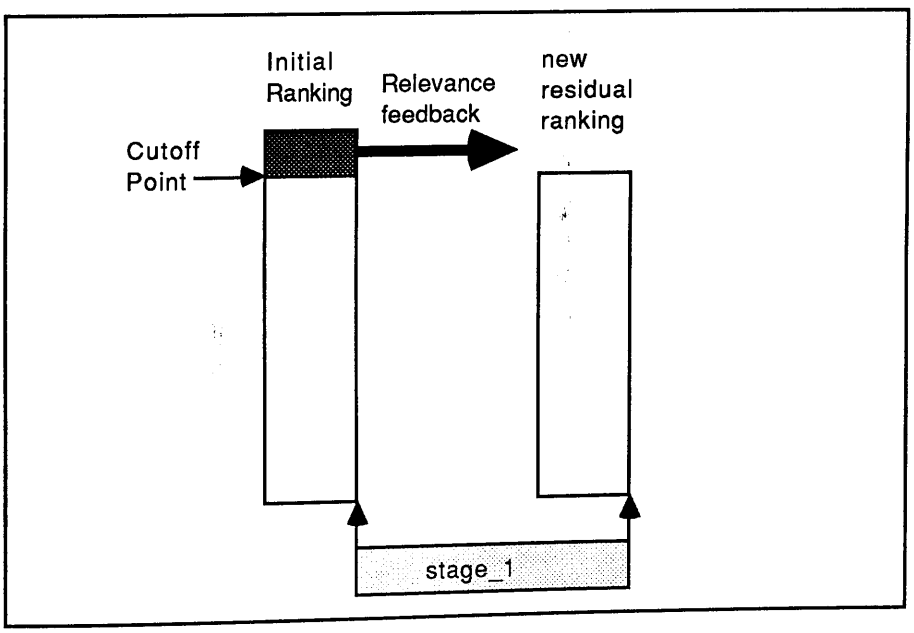


Figure_9.12: One-stage Imaging Retrieval

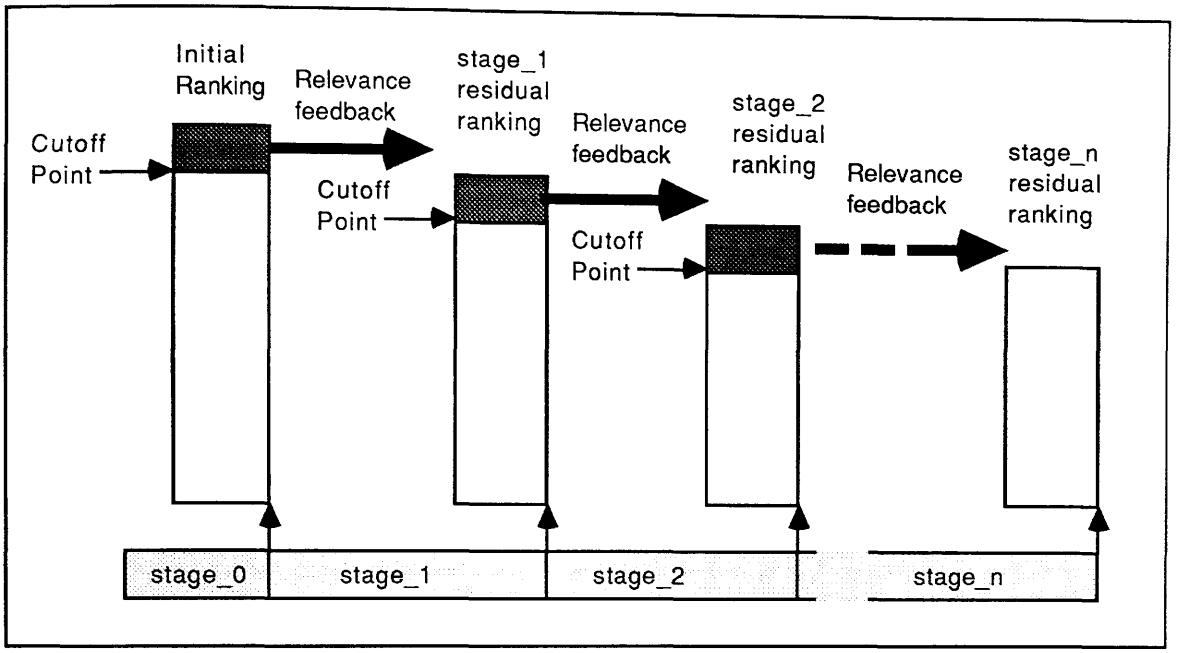


Figure_9.13: Multi-stage Imaging Retrieval

9.6.5.1 Method of Evaluating Multi-stage Imaging Retrieval

The method used in evaluating the effectiveness of the multi-stage imaging retrieval is based on the residual ranking evaluation. The effectiveness of the retrieval at stage_n is determined by comparing the average precision value obtained by the stage-n residual ranking to the value obtained by the residual of the stage_(n-1) residual ranking, as illustrated in Figure_9.14. The initial ranking is labeled as stage_0.

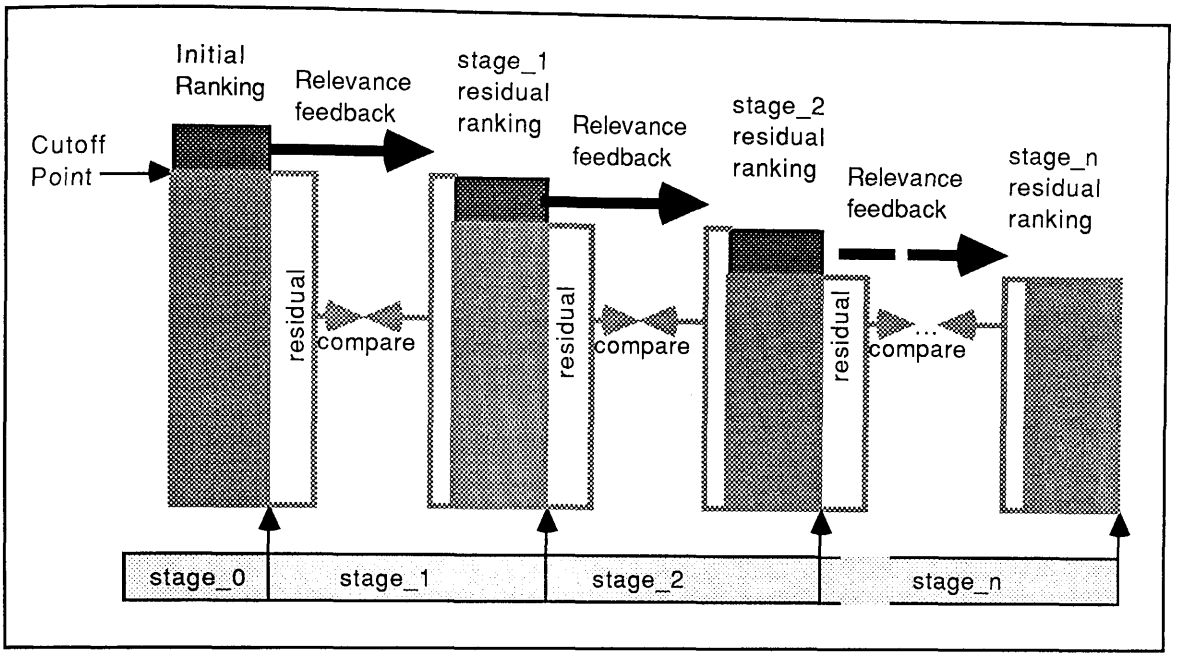


Figure_9.14: Evaluating the Effectiveness of Imaging Retrieval

9.6.5.2 Results of Multi-stage Imaging Retrieval Experiment

A 3-stage imaging retrieval is performed with a value of C being equal to 10. The increment function used in this retrieval is the A1 function as used in the earlier experiments. The results obtained from this experiment will be discussed stage by stage as follows:

Stage_1:

The precision-recall values obtained by the initial residual ranking and the stage_1 residual ranking are given in Table_9.9 where the average precisions obtained are 15.81 and 16.00, respectively. This shows that the stage_1 ranking is better than the initial residual ranking by 1.2%. These results are equivalent to the one-stage retrieval with cutoff point at 10.

		Precisions	
		B'chmark	stage_1
Recall Levels	10	37.39	38.25
	20	28.70	28.91
	30	22.36	22.41
	40	16.75	17.07
	50	14.41	14.54
	60	11.67	11.99
	70	9.69	9.67
	80	7.01	6.96
	90	5.36	5.32
	100	4.69	4.82
Average		15.81	16.00
% Increase			1.2

Table_9.9: Recall Cutoff Evaluation at Stage-1

Stage_2:

The precision-recall values obtained by the stage_2 residual ranking are given in Table_9.10 with an average precision of 7.82. In order to determine the effectiveness of the retrieval at this point, the values obtained should be compared to the values obtained by the residual of the stage_1 residual ranking which is also given in Table_9.10 with an average precision of 7.55. This means that the residual ranking at this stage is better than the previous residual ranking by 3.5%.

		Precisions	
		residual of stage_1	stage_2
Recall Levels	10	21.32	23.23
	20	13.77	14.31
	30	11.00	11.10
	40	7.90	7.94
	50	6.69	6.75
	60	4.86	4.86
	70	3.63	3.63
	80	2.88	2.88
	90	2.01	2.01
	100	1.47	1.47
Average		7.55	7.82
% Increase			3.5

Table_9.10: Recall Cutoff Evaluation at Stage-2

The results obtained at this stage can be compared to the results of the one-stage retrieval with cutoff point at 20. In order to do this, we performed document cutoff evaluation on these two sets of results. Table_9.11 gives the results of the document cutoff evaluation, where it appears that at document cutoff points 200 and below the performance of the multi-stage retrieval is better than the one-stage retrieval and the performance seems to be the same at the higher document cutoff points. The improvement obtained at this stage may also be due to the improvement obtained by the previous stage.

	Number of Documents Retrieved & Relevant	
Document Cutoff Points:	one-stage with cutoff =20	multi-stage at stage-2 with cutoff = 10
25	239	240
30	262	263
40	290	292
50	319	321
100	396	398
200	494	495
400	555	555
600	592	592
800	618	618
1000	626	625

Table_9.11: Document Cutoff Evaluation at Stage-2

Stage_3:

The precision-recall values obtained by stage_3 residual ranking are given in Table_9.12 with an average precision of 5.83. For comparison, the values obtained by the residual of the stage_2

residual ranking are also given in Table_9.12 with the average precision of 6.19. At this stage the average precision obtained is worse than the previous residual ranking by 5.8%. This tell us that the performance at each stage is not necessarily better than the previous stage.

		Precisions	
		residual of stage_2	stage_3
Recall Levels	10	14.07	13.26
	20	11.65	11.23
	30	7.79	7.28
	40	6.90	6.44
	50	6.01	5.61
	60	4.49	4.19
	70	3.71	3.46
	80	3.15	2.94
	90	2.39	2.24
	100	1.73	1.6
Average		6.19	5.83
% Increase			-5.8

Table_9.12: Recall Cutoff Evaluation at Stage-3

As in the previous stage, the results obtained at this stage can be compared to the results of one-stage retrieval with cutoff point at 30 using the document cutoff evaluation. Table_9.13 gives the results of the document cutoff evaluation, where it appears that the performance at this stage is worse than the one-stage retrieval.

Document Cutoff Points:	Number of Documents Retrieved & Relevant	
	one-stage with cutoff = 30	multi-stage at stage-3 with cutoff = 10
35	274	270
40	292	286
45	301	302
50	319	330
100	397	401
200	494	488
400	555	547
600	592	584
800	618	610
1000	626	617
2000	665	656
3000	716	708

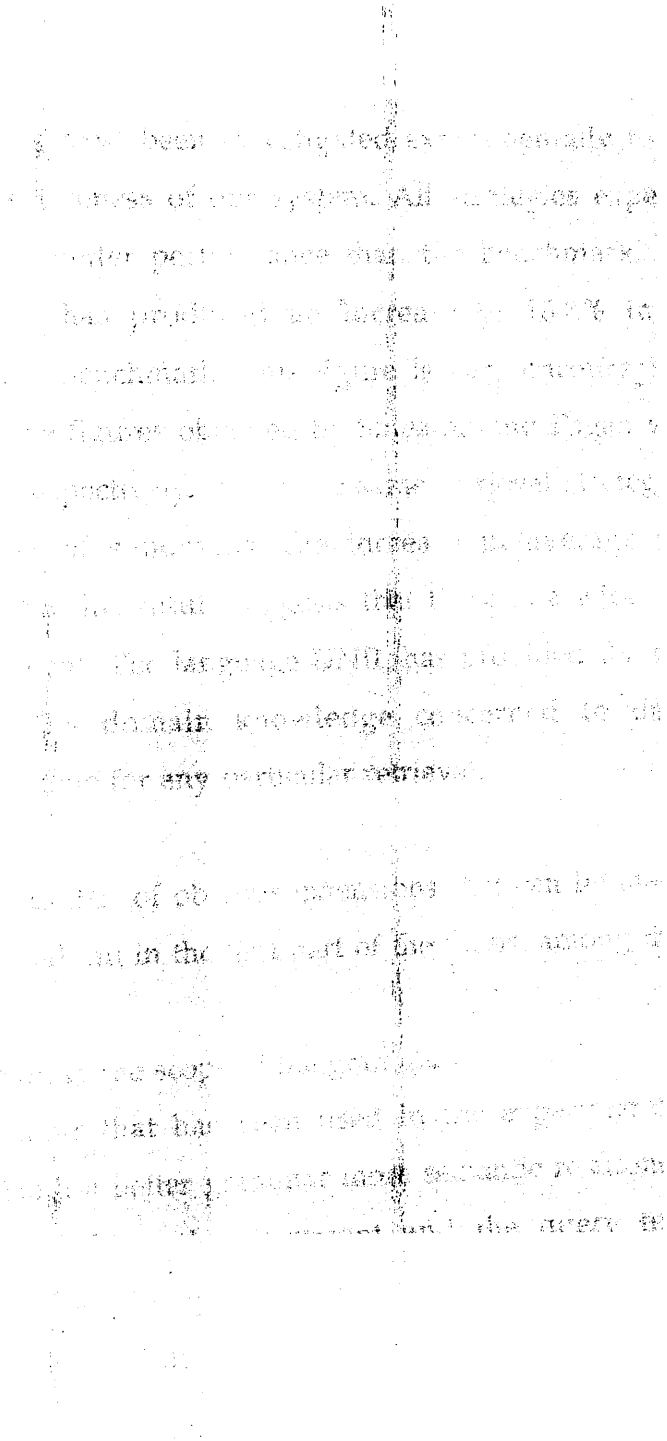
Table_9.13: Document Cutoff Evaluation at Stage-3

9.7 Conclusion

From the one-stage experiments we have found an increment function which gives consistence improvements in precision at all levels of recall. We also discovered that the size of cutoff for relevance feedback influences the level of retrieval effectiveness. A larger cutoff increases the number of relevant documents which are fed back to the system and therefore increases the effectiveness of the subsequent retrieval.

The results from the multi-stage experiments show that a two-stage imaging retrieval performed with cutoff point at 10 is better than the one-stage imaging retrieval with cutoff point at 20. But when the

number of stages is increased to three, there is a reduction in performance. Thus, it suggests that there is an optimal number of stages. In our limited experiments, it suggests that two is one of the possibilities.



Chapter 10: Conclusions

The first part of the thesis sets out to investigate further the use of linguistic processing in the work of IR. Instead of using parsing and a set of heuristic rules in the indexing process, we have used semantic translation of natural language into a predicate representation which has been adopted as the content indicator representation. Several retrieval strategies have been investigated experimentally to evaluate the retrieval effectiveness of our system. All strategies experimented with have shown better performance than the benchmark. Our best retrieval strategy has produced an increase of 16.8% in average precision over the benchmark. This figure is very encouraging when compared with the figures obtained by Smeaton and Fagan which are 5.07% and 8.7% respectively. When the same retrieval strategy is used with a small set of synonyms, the increase in average precision obtained is 24.3%. This result suggests that there is a wide scope for further improvement. The language UNIL has provided the facility to the experts in the domain knowledge concerned to define the synonyms at any time for any particular retrieval.

There are a number of obvious extensions that can be made to the experiments carried out in the first part of the thesis, among them are:

1) Extension to the scope of the grammar.
The grammar that has been used in the experiments is very limited. With a better grammar more semantic relationships can be captured from the document and the query texts. The document and the query representations can then be more accurate, and thus more informative matching can be performed.

2) Extension to the lexicon.

It would be interesting to see how the system performs with a large lexicon. In the experiments performed, many words in the document texts are unknown to the system and are just translated into one-place predicates. With a larger lexicon there will be more multi-place predicates generated and in this situation the experiment of retrieval strategy with transitive dependency might perform better.

3) More complex retrieval strategies.

More complex retrieval strategies can be investigated to improve the performance. For example, by having a partial matching on the types of dependencies with implication rules such as :

r: on(X,Y) :- cf(0.5)`of(X,Y).

To do this we need to study the semantics of each multi-place predicate carefully in order to assign the uncertainty factors (cf).

In translating natural language texts into the predicate representation we have followed the technique adopted by Jowsey's SMG. Ideally, we should have used his semantic representation in its entirety to represent the documents and the queries. But, it has been mentioned at the beginning that even if we had an appropriate semantics which could be computed efficiently, we still would not know how to use it to retrieve documents in response to requests. We think that this is the main problem that has to be tackled when trying to use an established semantic theory in document retrieval systems. Due to this problem, we have decided to take easier solution by using our own semantic representation which is based on the retrieval

strategies used. Based on the basis of retrieval strategies adopted in our experiments, i.e. retrieval based on dependencies, we have adopted the generalised-relationship concept to represent the semantic of documents and queries. Further study has to be made of this representation if the scope of the grammar used is to be expanded.

In the first part of the thesis we also have demonstrated and implemented matching as a simple inference using the unification process of a Prolog system. The inference from a document to a query is performed in a context of global information represented in the form of implication rules and the thesaurus. In doing this one of the problems we faced is the speed of the Prolog system used. Nevertheless, with the availability of parallel machines and parallel logic programming languages, there is scope for research to improve the efficiency in execution time. Moreover, there is parallelism inherent in the document retrieval operation that can be exploited to increase the speed. The simplest is that the inference between each document and the query could be done in parallel.

The second part of the thesis is concerned with an implementation of the imaging retrieval strategy and the evaluation on its performance. The results obtained by the experiments performed are very encouraging. In the one-stage imaging retrieval experiments, we have obtained an increase in average precision by 9.6% over the benchmark when using a cutoff point at 20 (the number of documents shown to the user for relevance feedback). In the multi-stage experiments, we have found that a two-stage imaging is better than its benchmark, and a three-stage imaging is worse than its benchmark. We envisage that imaging retrieval with number of stages higher than two will give worse result than its benchmark due to the reason that the scores of the non-relevant documents in the potentially relevant set

are updated too often. The results obtained for the two-stage imaging retrieval with cutoff point at 10 is in fact better than a one-stage retrieval with a cutoff point at 20. These results are enough to show the viability and validity of the imaging retrieval strategy and to support it as something worth looking into further.

The method used in determining the increment function to produce a better new ranking of documents is based on intuition rather than on a well founded formulation. Therefore, this is another area where further research can be undertaken in order to produce a better increment function.

Generally, the thesis has defined the logical-linguistic model of document retrieval systems, demonstrated how a system called SILOL is implemented based on this model, and evaluated the retrieval effectiveness of this system.

References

[Abdullah 86]

Abdullah W.A.T.W., "Fuzziness and Clausal Logic", *Malaysian Journal of Computer Science*, Vol.2, pp. 17-21, 1986.

[Aho&Ullman 77]

Aho,V.A., Ullman, J.D., *Principles of Compiler Design*, Addison-Wesley, 1977.

[Ajdukiewicz 35]

Ajdukiewicz K., "Die syntakische konnexitat", Studia Philosophica 1 (1935):1-27; translated as "Syntactic connnexion", Polish Logic, pp. 207-231, S.McCall(Ed.), Oxford:Clarendon Press, 1967.

[Allen 87]

Allen J., *Natural Language Understanding*, The Benjamin/Cummings Publishing Company, Inc., 1987.

[Berrut&Palmer 86]

Berrut, C., Palmer, P., "Solving grammatically ambiguities within a surface syntactical parser for automatic indexing", in F. Rabitti, editor, *Proceeding of the ACM Conference on Research and Development in Information Retrieval*, pp.123-130, 1986.

[BCS 87]

British Computer Society Information Retrieval Specialist Group, *Report of Joint Meeting on Information Retrieval: Current Trends and Future Prospects*, Newsletter of the BCS IR Specialist Group, Issue 32, May 1987.

[Bradley 06]

Bradley H., *The Making of English*, Macmillan, 1906.

[Colmerauer 78]

Colmerauer, A., "Metamorphosis grammars", in Bolc L. (Ed.), *Natural Language Communication with Computers*, Springer-Verlag, 1978.

[Cooper 84]

Cooper W.S., "Bridging the Gap between AI and IR", In van Rijsbergen(Ed.) *Research and Development in Information Retrieval*, pp.259-265, 1984.

[Cooper 78]

Cooper W.S., *Foundations of Logico-Linguistics*, D.Reidel Publishing Co., 1978.

[Croft 78]

Croft, W.B., "Organizing and Searching Large Files of Document Descriptions", Ph.D. Thesis, Churchill College, University of Cambridge, 1978.

[Croft&Lewis 87]

Croft, W.B., Lewis, D.D., "An Approach to Natural Language Processing for document retrieval", in C.T. Yu and van Rijsbergen, editors, *Proceedings of the 10th ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.26-32, June 1987.

[DeFude 84]

DeFude B., "Knowledge based systems vs thesaurus: an architecture problem about expert system design". In C.J. van Rijsbergen(Ed.), *Research and Development in Information Retrieval, Proceedings of the 3rd BCS ACM Symposium*, pp.267-280, 1984.

[DeFude 85]

DeFude B., "Different Levels of Expertise for an Expert System in Information Retrieval", *Proceedings of the 8th ACM SIGIR Conference*, Montreal, pp147-153, 1985.

[Dillon&Gray 83]

Dillon M., Gray A.S., "FASIT: A Fully Syntactically Based Indexing System", *Journal of the American Society for Information Science* 34(2), pp.99-108, 1983.

[Downing 77]

Downing Pamela, "On the Creation and Use of English Compound Nouns", *Language* 53, 4, pp.810-842, 1977.

[Dowty 79]

Dowty D.R., *Word Meaning and Montague Grammar: The Semantics of Verbs and Times in Generative Semantics and in Montague's PTQ*, D.Reidel Publishing Company, 1979.

[Dowty et al 81]

Dowty D.R., Wall R.E., Peters S., *Introduction to Montague Semantics*, D. Reidel Publishing Co., 1981.

[Fagan 87]

Fagan J.L.,"Experiments in Automatic Phrase Indexing For Document Retrieval: A Comparison of Syntactic and Non-Syntactic Methods", Ph.D. Thesis, Department of Computer Science, Cornell University, Ithaca, New York, 1987.

[Fox 83]

Fox E.A., *Characterization of Two New Experimental Collections in Computer and Information Science Containing Textual and Bibliographic Concepts*, Technical Report 83-561, Department of Computer Science, Cornell University, September 1983.

[Frost 88]

Frost, R., "A Knowledge Base System with a Natural Language Front-End", Technical Report, School of Computer Science, University of Windsor, Canada, Jan. 1988.

[Harper 80]

Harper D.J., "Relevance Feedback in Document Retrieval Systems: An Evaluation of Probabilistic Strategies", Ph.D Thesis, Jesus College, Cambridge University, 1980.

[Jensen 86]

Jensen K., "Parsing Strategies in a Broad-Coverage Grammar of English", Technical Report RC12147, *IBM Research Report*, 1986.

[Jowsey 87]

Jowsey E., "Montague Grammar and First Order Logic", Edinburgh Working Papers in Cognitive Science, Vol.1, University of Edinburgh, 1987.

[Keenan& Faltz 85]

Keenan E.L., Faltz L.M., *Boolean Semantics for Natural Language*, Vol. 23, Reidel, 1985

[Levi 75]

Levi Judith N., "The Syntax and Semantics of Non-predicating Adjectives in English", University of Chicago dissertation, 1975.

[Lewis et al 89]

Lewis, D.D., Croft, W.B., Bhandaru, "Language-Oriented Information Retrieval", to appear in *International Journal of Intelligent Systems*, 1989.

[McCalla&Cercone 83]

Approaches to Knowledge Representation", *IEEE Computer*, 16(10), pp.12-18, 1983.

[Montague 73]

Montague R., "The proper treatment of quantification in ordinary English", Reprinted in R.H.Thomason (ed.)(1974), *Formal Philosophy: Selected Papers of Richard Montague*, Yale University Press, New Haven, Conn., 1973.

[Porter 80]

Porter M.F., "An Algorithm for suffix stripping", Program, 14(3), pp.130-137, 1980.

[Radecki 77]

Radecki, T., "Mathematical model of time-effective information system based on the theory of fuzzy sets", *I.P.&M.*, Vol.13, pp.313-318, 1977.

[Rodman 76]

Rodman R., "Scope phenomena, 'Movement Transformations', and Relative clauses", In Partee B.H.(ed.), *Montague Grammar*, Academic Press, 1976.

[Salton 86]

Salton G., "Recent trends in Automatic Information Retrieval", *Proc. of 1986 ACM Conference on Research and Development in Information Retrieval*, Rabitti F.(Ed.), pp.1-10, 1986.

[Salton&Buckley 88]

Salton, G., Buckley, C., "Parallel Text Search Methods", Communication of the ACM, Vol.31, pp.202-215, 1988.

[Salton 88]

Salton,G., "A Simple Blueprint for Automatic Boolean Query Processing", Information Processing & Management, Vol.24, No. 3, pp.269-280, 1988.

[Salton 89]

Salton, G., *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley, 1989.

[Saint-Dizier 86]

Saint-Dizier P., "An approach to Natural-Language Semantic in Logic Programming", J. Logic Programming 4, pp.329-356, 1986.

[Smeaton 87]

"Using Parsing of Natural Language as part of Document Retrieval", Ph.D. Thesis, Department of Computer Science, University College Dublin, Ireland, 1987.

[Thomason 76]

Thomason R.H., "Some Extensions of Montague Grammar", In Partee B.H.(ed.), *Montague Grammar*, Academic Press, 1976.

[Thurmair 86]

Thurmair, G., "A common architecture for different text processing techniques in an information retrieval environment". In F. Rabitti(Ed.), *Proceedings of the ACM Conference on Research and Development in Information Retrieval*, pp.138-143, 1986.

[van Rijsbergen 72]

van Rijsbergen, C.J., "Automatic Information Structuring and Retrieval", Ph.D. Thesis, University of Cambridge, 1972.

[van Rijsbergen 79]

van Rijsbergen, C.J., *Information Retrieval*, 2nd edition, 1979.

[van Rijsbergen 86]

van Rijsbergen C.J., "A non-classical logic for information retrieval", *The Computer Journal* 26(6), pp.481-485, 1986.

[van Rijsbergen 89]

van Rijsbergen C.J., "Towards an Information Logic", Research Report CSC/89/R8, Dept. of Computing Science, University of Glasgow, 1989.

[Waldstein 81]

Waldstein R., "The role of Noun Phrases as Content Indicators", Ph.D. Thesis, School of Information Studies, Syracuse University, Syracuse, New York, 1981.

[Walker 81]

Walker, D.E., "The Organisation and use of Information: Contributions of Information Science, Computational Linguistics, and Artificial Intelligence", *Journal of the American Society for Information Science*, 32, pp.347-363, 1981.

[Warren 82]

Warren D.H.D., Pereira F.C.N., "An Efficient Easily Adaptable System for Interpreting Natural Language Queries", *American Journal of Computational Linguistics*, Vol.8, No.3-4, July-December 1982.

[Zadeh 83]

Zadeh L.A., "A Theory of Approximate Reasoning", in *Machine Intelligence 9*, J. E. Hayes, D. Michie, and L.I. Kulich, eds., Wiley, pp.407-428, 1979.

[Zeevat et al 87]

Zeevat H., Klein E., Calder J., "Unification Categorial Grammar", Edinburgh Working Papers in Cognitive Science, Vol.1, University of Edinburgh, 1987.

APPENDIX_A: The SUNG Implementation Rules

```
% noun-noun phrase
fn >> cn(Gendx):[Y | B] + cn(Gend):[X | A]
      => cn(Gend):[X | A & B & r(Y,X)].

% sentence - past and present tense being treated equally.
f4 >> t(Gend,subj):[P | F] + iv(Tense):P => s:F.

% Determiner-noun
f2 >> det:[P | PP] + cn(Gend):P => t(Gend,Case):PP.

% Adjective
fa >> acn:[P | Q] + cn(Gend):P => cn(Gend):Q.

% Transitive verb
f5 >> tv(Tense):[PP | P] + t(Gend,obj):PP => iv(Tense):P.

%WH Relative Clauses
fr >> cn(Gend):P + who(Case):[] => wh(Gend,Case):P.
fr >>  wh(Gend,subj):Sem1 + iv(Tense):Sem2 => cn(Gend):Sem
       <- relativisewho(X,Sem1,Sem2,Sem).
relativisewho( X, [X | A], [X | B], [X | A&B] ).
fr >> t(Gend,Case):[[X | B] | A] + tv(Tense):[ [ [Y | B] | C],X | C]
     => acn2:[ [Y | D],Y | D&A].
fr >> wh(Gend,Case):PP + acn2:[PP | P] => cn(Gend):P.

% preposition  %
fp >> prep:[PP | P] + t(Gend,Case):PP => acn2:P.
```

```
%preposition by
fby >> by:[PP | P] + t(Gend,Case):PP => tby:P.
fby >> tv(past):PP + tby:[PP | P] => acn2:P.
facn2 >> cn(Gend):PP + acn2:[PP | P] => cn(Gend):P.

% type raising from cn to t
fby >> by:[PP | P] + cn(Gend):CN => tby:P
      <- raise(CN,PP).
fp >> prep:[PP | P] + cn(Gend):CN => acn2:P
      <- raise(CN,PP).
f4 >> cn(Gend):CN + iv(Tense):P => s:F
      <- raise(CN,[P | F]).
f5 >> tv(Tense):[PP | P] + cn(Gend):CN => iv(Tense):P
      <- raise(CN,PP).
raise([X | A],[[X | B] | A&B]).

% Coordination or Conjunction
% Sentential Coordinators
Fn >> s:[A] + c(Co):[] + s:[B] => s:[C]
  <- co_ordinate( Co, Fn, A, B, C ).
co_ordinate( and, f8, A, B, A & B ).
co_ordinate( or, f9, A, B, A v B ).

% Predicate Coordinators
Fn >> iv(Tense):[X | A] + c(Co):[] + iv(Tense):[X | B] =>
      iv(Tense):[X | C]
       <- co_ordinate( Co, Fn, A, B, C ).
% Common noun Coordinators
f9 >> cn(_):[X | A] + c(or):[] + cn(_):[Y | B] =>
      cn(neut):[Z | (A & B)&('='(X,Z) v '='(Y,Z))].
```

```
f8 >> cn(_):[X | A] + c(and):[] + cn(_):[Y | B] =>
      cn(neut):[Z | (A & B)&('='(X,Z) & '='(Y,Z))].
% Term Coordinators
f9 >> t(Gend1,Case):[[Y | '='(Y,X)] | A] + c(or):[] +
      t(Gend2,Case):[[Z | '='(Z,X)] | B]
      => t(Gend,Case):[[X | C] | (A v B)&C].
f8 >> t(_,Case):[[Y | '='(Y,X)] | A] + c(and):[] +
      t(_,Case):[[Z | '='(Z,X)] | B]
      => t(_,Case):[[X | C] | (A v B)=>C].
% Modifier Coordinators
Fn >> acn:[[X | '='(X,Z)],X | P] + c(Co):[] + acn:[[Y | '='(Y,Z)],Y | Q] =>
     acn:[[Z | C],Z | R&C]
      <- co_ordinate( Co, Fn, P, Q, R).
```

APPENDIX_B: Grouping of the Grammar rules

```
0:Fn >> acn:[[X | '='(X,Z)],X | P] + c(Co):[] + acn:[[Y | '='(Y,Z)],Y | Q] =>
    acn:[[Z | C],Z | R&C]
     <- co_ordinate( Co, Fn, P, Q, R).

1:fn >> cn(Gendx):[Y | B] + cn(Gend):[X | A]
      => cn(Gend):[X | A & B & r(Y,X)].
1:fa >> acn:[P | Q] + cn(Gend):P => cn(Gend):Q.
1:f9 >> cn(_):[X | A] + c(or):[] + cn(_):[Y | B] =>
      cn(neut):[Z | (A & B)&('='(X,Z) v '='(Y,Z))].
1:f8 >> cn(_):[X | A] + c(and):[] + cn(_):[Y | B] =>
      cn(neut):[Z | (A & B)&('='(X,Z) & '='(Y,Z))].

2:facn2 >> cn(Gend):PP + acn2:[PP | P] => cn(Gend):P.
2:fby >> by:[PP | P] + cn(Gend):CN => tby:P
    <- raise(CN,PP).
2:fp >> prep:[PP | P] + cn(Gend):CN => acn2:P
    <- raise(CN,PP).
2:f5 >> tv(Tense):[PP | P] + cn(Gend):CN => iv(Tense):P
    <- raise(CN,PP).
2:fr >> cn(Gend):P + who(Case):[] => wh(Gend,Case):P.

3:fr >>  wh(Gend,subj):Sem1 + iv(Tense):Sem2 => cn(Gend):Sem
    <-  relativisewho(X,Sem1,Sem2,Sem).

4:facn2 >> cn(Gend):PP + acn2:[PP | P] => cn(Gend):P.
4:f2 >> det:[P | PP] + cn(Gend):P => t(Gend,Case):PP.
4:f9 >> t(Gend1,Case):[[Y | '='(Y,X)] | A] + c(or):[] +
    t(Gend2,Case):[[Z | '='(Z,X)] | B]
    => t(Gend,Case):[[X | C] | (A v B)&C].
4:f8 >> t(_,Case):[[Y | '='(Y,X)] | A] + c(and):[] +
    t(_,Case):[[Z | '='(Z,X)] | B]
    => t(_,Case):[[X | C] | (A v B)=>C].
4:fp >> prep:[PP | P] + t(Gend,Case):PP => acn2:P.

%preposition by
1:fby >> by:[PP | P] + t(Gend,Case):PP => tby:P.
1:fby >> tv(past):PP + tby:[PP | P] => acn2:P.
1:facn2 >> cn(Gend):PP + acn2:[PP | P] => cn(Gend):P.
```

```
% preposition %
l:fp >> prep:[PP | P] + t(Gend,Case):PP => acn2:P.

% noun-noun phrase %
l:fn >> cn(Gendx):[Y | B] + cn(Gend):[X | A]
      => cn(Gend):[X | A & B & r(Y,X)].

% sentence - past and present being treated equally (notice Tense)
l:f4 >> t(Gend,subj):[P | F] + iv(Tense):P => s:F.

l:f2 >> det:[P | PP] + cn(Gend):P => t(Gend,Case):PP.

l:fa >> acn:[P | Q] + cn(Gend):P => cn(Gend):Q.

l:f5 >> tv(Tense):[PP | P] + t(Gend,obj):PP => iv(Tense):P.

% raising:-
l:fby >> by:[PP | P] + cn(Gend):CN => tby:P
    <- raise(CN,PP).
l:fp >> prep:[PP | P] + cn(Gend):CN => acn2:P
    <- raise(CN,PP).
l:f4 >> cn(Gend):CN + iv(Tense):P => s:F
    <- raise(CN,[P | F]).
l:f5 >> tv(Tense):[PP | P] + cn(Gend):CN => iv(Tense):P
    <- raise(CN,PP).

raise([X | A],[[X | B] | A&B]).

% Change Tense to pres to cater for tenses
l:fr >> cn(Gend):P + who(Case):[] => wh(Gend,Case):P.
l:fr >>  wh(Gend,subj):Sem1 + iv(Tense):Sem2 => cn(Gend):Sem
    <- relativisewho(X,Sem1,Sem2,Sem).
relativisewho( X, [X | A], [X | B], [X | A&B] ).

l:fr >> t(Gend,Case):[[X | B] | A] + tv(Tense):[ [ [Y | B] | C],X | C]
    => acn2:[ [Y | D],Y | D&A].
l:fr >> wh(Gend,Case):PP + acn2:[PP | P] => cn(Gend):P.
```

```
% CONJUNCTION
% Sentential Coordinatorsl:
Fn >> s:[A] + c(Co):[] + s:[B] => s:[C]
  <- co_ordinate( Co, Fn, A, B, C ).

co_ordinate( and, f8, A, B, A & B ).
co_ordinate( or, f9, A, B, A v B ).

% Predicate Coordinators
l:Fn >> iv(Tense):[X | A] + c(Co):[] + iv(Tense):[X | B] =>
    iv(Tense):[X | C]
    <- co_ordinate( Co, Fn, A, B, C ).

% Common noun Coordinators
l:f9 >> cn(_):[X | A] + c(or):[] + cn(_):[Y | B] =>
      cn(neut):[Z | (A & B)&('='(X,Z) v '='(Y,Z))].

l:f8 >> cn(_):[X | A] + c(and):[] + cn(_):[Y | B] =>
      cn(neut):[Z | (A & B)&('='(X,Z) & '='(Y,Z))].

% Term Coordinators
l:f9 >> t(Gend1,Case):[[Y | '='(Y,X)] | A] + c(or):[] +
    t(Gend2,Case):[[Z | '='(Z,X)] | B]
    => t(Gend,Case):[[X | C] | (A v B)&C].

l:f8 >> t(_,Case):[[Y | '='(Y,X)] | A] + c(and):[] +
    t(_,Case):[[Z | '='(Z,X)] | B]
    => t(_,Case):[[X | C] | (A v B)=>C].

% Modifier Coordinators
l:Fn >> acn:[[X | '='(X,Z)],X | P] + c(Co):[] + acn:[[Y | '='(Y,Z)],Y | Q] =>
     acn:[[Z | C],Z | R&C]
      <- co_ordinate( Co, Fn, P, Q, R).
```