A STEMMING ALGORITHM FOR LATVIAN

by

Karlis Kreslins

A Doctoral Thesis
Submitted in partial fulfilment of the requirement for
the award of Doctor of Philosophy
of
Loughborough University

October 1996

Supervisors: Mr Alan Poulter, BA, MA, Msc, ALA
Mrs Inese A. Smith, BA, MA, FLA
Department of Information and Library Studies

CONTAINS DISKETTE

UNABLE TO COPY

CONTACT UNIVERSITY

IF YOU WISH TO SEE

THIS MATERIAL

To my parents and my godmother
who helped and supported me

ABSTRACT

The thesis covers construction, application and evaluation of a stemming algorithm for advanced information searching and retrieval in Latvian databases. Its aim is to examine the following two questions:

- Is it possible to apply for Latvian a suffix removal algorithm originally designed for English?
- Can stemming in Latvian produce the same or better information retrieval results than manual truncation?

In order to achieve these aims, the role and importance of automatic word conflation both for document indexing and information retrieval are characterised. A review of literature, which analyzes and evaluates different types of stemming techniques and retrospective development of stemming algorithms, justifies the necessity to apply this advanced IR method also for Latvian. Comparative analysis of morphological structure both for English and Latvian language determined the selection of Porter's suffix removal algorithm as a basis for the Latvian stemmer.

An extensive list of Latvian stopwords including conjunctions, particles and adverbs, was designed and added to the initial stemmer in order to eliminate insignificant words from further processing. A number of specific modifications and changes related to the Latvian language were carried out to the structure and rules of the original stemming algorithm.

Analysis of word stemming based on Latvian electronic dictionary and Latvian text fragments confirmed that the suffix removal technique can be successfully applied also to Latvian language. An evaluation study of user search statements revealed that the stemming algorithm to a certain extent can improve effectiveness of information retrieval.

ACKNOWLEDGMENTS

I should like to express the deepest gratitude to my supervisors Mr Alan Poulter and Mrs Inese A. Smith for their invaluable and outstanding guidance, support and help throughout all my study. I should also like to thank my Director of Research Dr Paul Sturges for his advice and administrative work while I was studying at the University.

I am also indebted to Professor Margaret Evans, Head of Department, and Professor John Feather, Pro-Vice Chancellor, for their support during my study. I would like to express my very special thanks to all staff members of the Department for their help and guidance on many occasions.

My deepest gratitude to the British Council, the Soros Foundation Latvia and the Latvian Educational Foundation (UK) for their sponsorship and financial support.

I would like to express my appreciation to Ms Olga Rogova who helped me to solve a number of complex programming problems. I am greatly thankful to Andrejs Spektors, Arts Klints, Aivars Liepa, Ivars Indans, Inguna Greitane and Edgars Gasins for their advice and help not only in programming and information retrieval but also in linguistics. Many thanks to Ainars Bruveris and Lursoft Company as well as SWH Riga for allowing the use of their databases in the evaluation study.

Special thanks are due to Dace Gasina, Baiba Sporane, Inga Grinfelde, Valda Laucina, Baiba Holma, Baiba Muze, Marina Sanuka and Uldis Zarins for their willingness to participate in the evaluation study of my stemming algorithm. I would also like to express my gratitude to Dr Viktorija Drizule, Dr Sarma Klavina and Mrs Anna Maulina for their advice and suggestions regarding the evaluation of Latvian word stemming.

Finally, my deepest thanks to Mr Andris Vilks, Director of the National Library of Latvia, and all the other colleagues in Latvia who provided moral support and encouragement during the period of my research in the UK.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

The development of information technology and electronic media in Latvia is growing more and more rapidly. Access and use of the Internet and WWW search engines are becoming part and parcel of everyday life not only in Latvia but also in the rest of the Baltic countries. Networking of educational institutions and organisations (i.e. universities, colleges) as well as business companies and firms is in progress and several international projects with other European countries related to global area networks (e.g. LIBNET) have been started recently.

Along with networks and access to the Internet, there is evidence of a rapidly growing number of Latvian databases and information systems. To date there are more than a 100 different types of bibliographic, factographic and full-text databases. Moreover, as a candidate state for joining the European Union and NATO, Latvia together with Estonia and Lithuania have been provided with access to a variety of EU and NATO databases.

The above mentioned factors helped determine the intension to introduce new, advanced and efficient information retrieval facilities (including stemming) for end-users in Latvia. This thesis, therefore, examines the design and evaluation of a stemming algorithm for Latvian in order to provide automatic word conflation both for document indexing and document searching in advanced information retrieval systems and databases. Research in information retrieval has confirmed that stemming is a challenging area not only for an English language environment, but also for other languages, including Latvian. This research has been undertaken to help fill the gap relating to stemming in inflective non-English languages using Latvian as a case study.

1.1 Aims and objectives

The aim of this thesis is to investigate the possibility of applying automatic word conflation to Latvian and to design a stemming algorithm for Latvian. In order to achieve this aim, the research covers the following three stages:

- analysis and evaluation of existing automatic word conflation methods for information retrieval in English and their relevance to a Latvian language environment;

- construction of the Latvian stopword list and development of a Latvian stemming algorithm based on one of the English automatic word conflation algorithms;
- performance analysis and evaluation of the designed Latvian stemmer including effectiveness of the stemmer in information retrieval.

The objective of this thesis is to achieve automatic word stemming for Latvian and to examine the performance and effectiveness of the Latvian stemming algorithm in information retrieval.

1.2 Scope of thesis

This thesis defines the role of the stemming procedure in advanced information retrieval system. All basic types of automatic word conflation algorithms are presented and analyzed. Because of the relevance to Latvian language, the retrospective overview of stemming algorithms includes only affix removal stemmers. The structure and complexity of the language determined that the Latvian stemming algorithm will comprise only suffix removal and exclude any kind of prefix removal. The Latvian stemmer was tested and evaluated using three different test collections.

The algorithm was not implemented and evaluated in any existing Latvian information retrieval systems and/or Latvian online databases because, during the evaluation period, no software packages were available in Latvia which can link the stemming algorithm with appropriate database or information system.

The evaluation study did not cover any comparison of the performance between the Latvian stemming algorithm and other English and non-English stemmers because the designed stemming algorithm was the first for Latvian and the grammatical structure of Latvian is distinct from English and other languages for which stemming algorithms have been already applied.

1.3 Hypotheses

This thesis tests the following two hypotheses:

- it is possible to apply for Latvian a suffix removal algorithm originally designed for English;
- stemming in Latvian will produce the same or better information retrieval performance results than manual truncation.

1.4 Methodology

The methodology used in this thesis covers the following areas:

- overview of previous research in construction and development of stemmers for advanced information retrieval systems both in English and non-English languages;
- analysis and comparison of English and Latvian linguistics based on morphology;
- construction of a Latvian stopword list;
- design and development of a Latvian stemming algorithm based on one of the English stemming algorithms;
- analysis and evaluation of this Latvian stemming algorithm.

Detailed description of the general methodology used for testing and evaluation of the Latvian stemmer is presented in Section 7.2.

1.5 Structure of thesis

This thesis consists of three parts. The first part covers a conceptual framework of stemming in information retrieval. Definitions and explanations of basic components in information retrieval systems as well as the role and place of stemming in document indexing and information retrieval are presented in Chapter 2. Analysis and description of existing approaches to automatic stemming of terms are covered by Chapter 3. This chapter also comprises a retrospective review of affix removal stemming algorithms which were applied to English and non-English languages. Finally, Chapter 4 analyzes the structure of Latvian and presents a comparison of English and Latvian morphology.

The second part deals with the construction and development of the Latvian stemming algorithm. Chapter 5 defines the basic principles of stopword selection and describes the design and implementation of a Latvian stopword list. Chapter 6 analyzes the structure of an initial English stemming algorithm and presents the development and modifications of the Latvian stemming algorithm based on this English algorithm.

Finally, the third part of this thesis covers the analysis and evaluation of the Latvian stemming algorithm. Examination and testing of Latvian word stemming using the Latvian electronic dictionary and Latvian text fragments is given in Chapter 7. This chapter also comprises the comparative evaluation of information retrieval effectiveness using manually truncated and automatically stemmed search statements. Chapter 8 presents a summary of conclusions based on previous chapters and outlines areas for further research.

Chapter 2

MAIN CHARACTERISTICS OF INFORMATION RETRIEVAL SYSTEMS

2.1 General concepts of information retrieval

Information retrieval covers a wide scope of disciplines related to non-numerical computing i.e. database management systems, computerised natural language processing, information searching algorithms, multimedia information systems etc.(1) Historically the terms 'information retrieval' and 'document retrieval' have been often used as synonyms because the information being processed usually comprises documents. As noted by Salton (2), information retrieval deals with the representation, storage and access to documents or document surrogates (i.e. abstracts, annotations). In that context information retrieval systems can alternatively been defined as document retrieval systems which cover the following basic functions i.e.

- to process and store documents into the database (i.e. to enter, change and delete information);
- to search for documents and to present them to the user according to his/her query (3).

However, the basic strategy for information retrieval systems is to provide access to document contents which have been stored in the database. It means that the system should determine the most appropriate documents in the database according to users' needs. Moreover, van Rijsbergen mentioned that relevance of retrieved information is important criterion and therefore, the end purpose of document retrieval systems is "to retrieve all the relevant documents at the same time retrieving as few of the non-relevant documents as possible"(4).

Figure 2.1 represents a general model of information retrieval system and its main components. Each document before being stored in the database, usually has assigned special descriptors (subject terms, keywords) which characterise its content. User search queries can also be processed (truncated, conflated) to match relevant documents in the database.

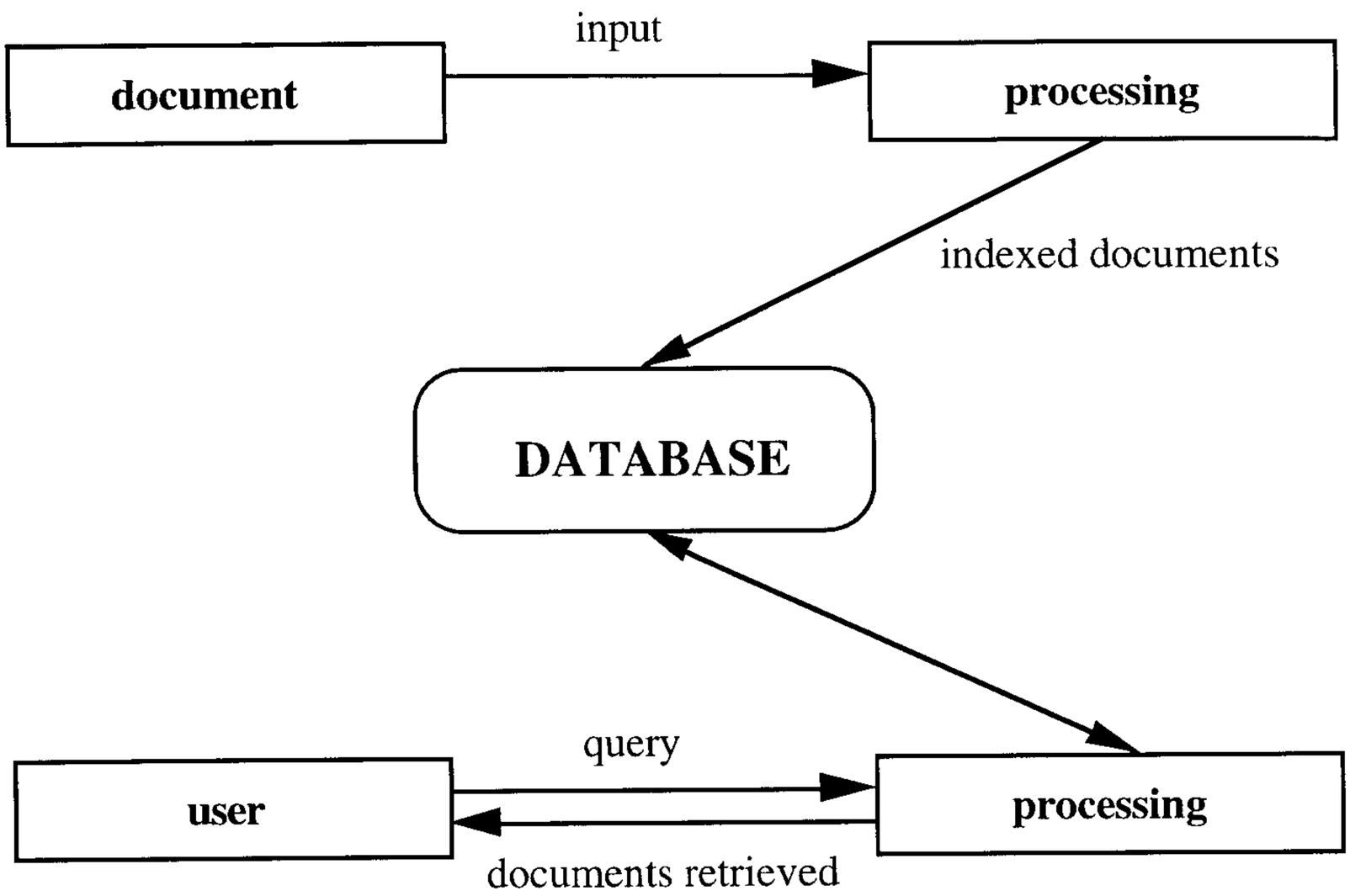

Figure 2.1 General pattern of information retrieval system

Computerised information retrieval (IR) systems have been known since late 50's when Keyword-In-Context indexing (KWIC) and Selective Dissemination of Information (SDI) systems were introduced. These systems used serial files for information retrieval stored mainly on magnetic tapes. The development of computer and telecommunication networks provided access to large, remote databases. The majority of conventional IR systems include two common features:

- inverted file structure;
- use of Boolean operators.

Harman defines *inverted file* as "the sorted list (or index) of keywords (attributes), with each keyword having links to the documents containing that keyword" (5). The inverted file includes keywords, document identification codes and field identification codes as the main entry elements. A dictionary file and a postings file form the structure of the inverted file. The *dictionary file* comprises all indexing terms in alphabetic order including the total number of documents in which those terms appear. The postings file covers all indexing terms which are coupled with the accession number of each document in which the term occurs. Even though the inverted file structure requires additional storage space and maintenance, its essential advantage over serial file organisation is rapid access to and retrieval of documents according to user queries.

Inverted file organisation, which is based on separate individual terms and document reference numbers, allows the use of traditional Boolean operators because the search terms associated with AND, OR, NOT can be correspondingly arranged into intersection set, union set and difference set. However, as mentioned by Ashford (6) the use of Boolean search strategies in conventional IR systems encounter several problems:

1) end-users who are not familiar with Boolean operators have difficulties formulating correct queries, so they require assistance from trained intermediaries;
2) there is no control over the number of documents retrieved by a definite query;
3) Boolean search does not provide any kind of ranking mechanism therefore, all records in the database are simply divided into two subsets- those which are retrieved by the particular query and those which are not.

To overcome the above mentioned problems and to ensure more effective document searching as well as to provide the use of natural language in users' queries, several advanced operational and experimental information retrieval systems (i.e. SMART, INSTRUCT, MASQUERADE) incorporate statistically based approaches. Figure 2.2 illustrates a statistically based information retrieval system and the main IR techniques covered by the system.

Document, database and user as well as the mutual interaction between those elements form the core of information retrieval system. Each document, before being added to the database, has to be appropriately processed to guarantee its successful retrieval from the database. According to Figure 2.2, the whole text is broken into separate words to select definite indexing terms or keywords which describe the contents of particular document.

Document indexing for statistically based IR systems usually involves a stemming procedure. According to Frakes (7) stemming is the automated conflation of related terms, by reducing the words to a common root form. Before the term is stemmed, it has to be matched against the stopword list, to separate meaningful terms from words of zero indexing value i.e. articles, prepositions, very general terms. After that, all remaining nonstoplist terms are stemmed, using an automated word conflation procedure.

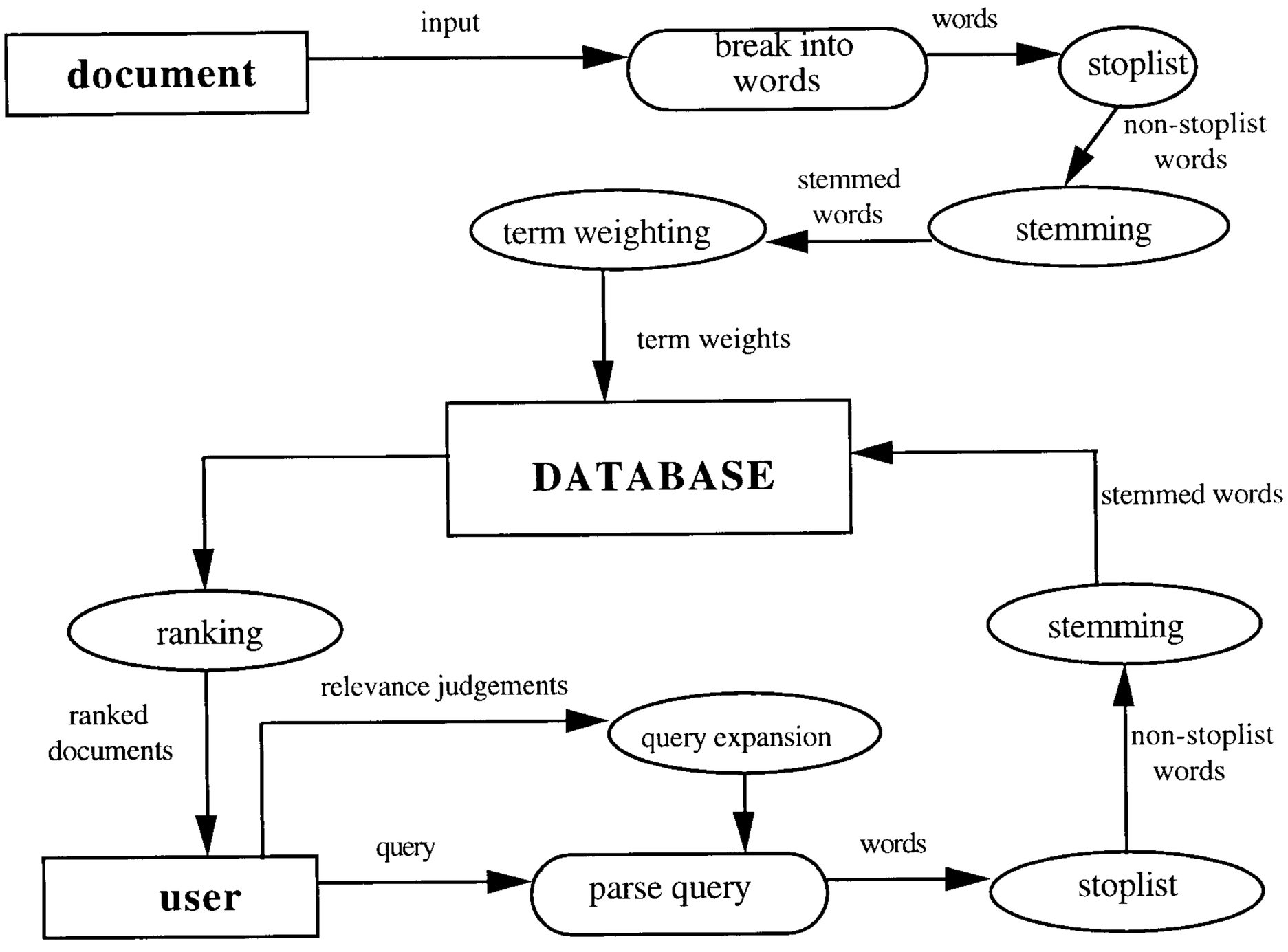

Figure 2.2 Functional view of statistically based IR system

In addition to automatic indexing, each stemmed term can be analysed and evaluated according to its statistical weight. The term weighting mechanism assigns each stem a definite measure of importance which is used in a document ranking procedure as well as to determine the most appropriate identifiers for document indexing. Finally, all relevant indexing terms for a particular document along with the document are added to the database. Therefore, automatic indexing together with word conflation and a term weighting procedure form the core for document/information input.

As described in Figure 2.2, the starting point for document/information output is the user's query which in advanced IR systems can be formulated in natural language. After the query has been entered, it is parsed into separate query terms. Similar to the above mentioned document indexing procedure, each query term is than checked against the stoplist and all remaining terms are conflated via a stemming algorithm. Finally, the stemmed query terms are compared with document content identifiers in the database and the whole matched document set is than presented to the user in a ranked order. Best match search evaluates the set of query stems against the set of stems for corresponding document in the database and along with a statistical term weighting

function, it is one of the most often used methods for document ranking. Advanced information retrieval systems also allow the user to make certain judgments about the relevance of retrieved documents. One of the approaches, which is based on user feedback, is query expansion. Depending on the number and quality of retrieved items, the initial query can be automatically expanded by adding new terms from relevant documents and deleting nonrelevant terms.

Overall, Figure 2.2 reveals, that successful document retrieval in advanced statistically based IR systems to the certain extent depends on following information retrieval principles:

- selection of relevant terms for automatic document/text indexing;
- use of appropriate automatic word stemming techniques both to determine document content descriptors and to process query terms;
- implementation of the best match search including a statistical term weighting method to determine relevance of documents in the database as well as relevance of retrieved information based on users' queries.

The following section will describe the indexing procedure with the emphasis on automatic term selection. A separate section deals with query expansion, best match search and term weighting techniques. Detailed description and analysis of word conflation methods and types as well as a review on affix removal stemming algorithms (algorithms which remove prefixes and/or suffixes) is given in Chapter 3

2.2 Document indexing

As noted by Willett (8), document indexing is used to characterise the contents of documents in the database and to ensure the retrieval of documents, which match a user's query. Indexing can involve either symbols (i.e. classification codes) or terms. Compared with symbols, content descriptors based on keywords and/or subject terms are more commonly used both in manual and automatic document indexing.

Manual indexing which usually is performed by experienced indexers, is still widely used for document description in catalogues and databases. Normally the manual indexing procedure incorporates either uncontrolled vocabulary (i.e. keywords) or controlled vocabulary - thesaurus. It was observed that in many cases where trained indexers were involved, the use of controlled vocabulary for document indexing was preferred rather than keywords. Thesaurus construction requires from indexers a good knowledge in the particular subject area as well as the ability to maintain accuracy and

consistency to identify main, broader, narrower and related terms when building the hierarchical dictionary. It means that indexing personnel should assign the same indexing terms for documents with similar contents to guarantee retrieval of relevant items.

However, practice reveals that manual indexing has certain limitations. As noted by Harter(9), lack of consistency among indexers often creates drawbacks in human prepared indexes and dictionaries. Controlled vocabulary seems to be complicated also for end-users as they have to translate the chosen query words into terms of an artificial language and to find the most appropriate words for relevant document retrieval.

The disadvantages of manual indexing prompted an alternative approach of document content description - automatic indexing. As noted by van Rijsbergen, in automatic indexing "it is anticipated that by processing documents in a computerised manner, the output will be a representation of the content."(10). Research and experiments on automatic indexing have shown that it is a fast and inexpensive method and can produce a *recall* ratio (the proportion of relevant items retrieved out of all relevant items in a system) and *precision* ratio (the proportion of relevant items retrieved out of all items retrieved) at least equivalent to that obtained in manual indexing (11,12). Van Rijsbergen(13) also indicated that even the use of simple automatic indexing procedures for selection of indexing terms in earlier IR systems, e.g. the SMART Project, performed good document retrieval results. Although a wide range of automatic indexing methods have been introduced and used in information retrieval, there is a general agreement that the automatic indexing system should cover following basic modules:

- selection of indexing terms or descriptors;
- term conflation or stemming;
- weighting of terms (14).

As was mentioned before, indexing requires the selection of relevant terms to describe the content of each document stored into the database. Especially, in terms of advanced document retrieval systems with natural language search, it is necessary to identify all the terms which characterise the subject of interest, so that it is possible correctly distinguish the relevant and the nonrelevant documents retrieved for the particular user query (15). Usually appropriate indexing terms can be found in document titles and abstracts or in the document text itself. One of the methods used for automatic term selection is the linguistic approach.

There have been several attempts to introduce linguistic methods for selection of indexing terms. In a majority of cases as noted by Smeaton(16), the complexity of natural language and the complexity of natural language texts were the basic reasons why the integration of automatic language processing into information retrieval failed. However, some of the latest experiments on automatic natural language processing revealed that the linguistic technique can be applied quite successfully for semantic and syntactic construction of term phrases. For example, a natural language parser designed by Spark Jones and Tait(17), produced acceptable noun phrases which can be used for searching in document abstracts. Another example is a Fully Automatic Syntactically Based Indexing System (FASIT) which parses a text into phrases according to certain syntactic categories and than use the most meaningful words and/or phrases as units for indexing (18). Retrieval results indicated that the main concept of FASIT - to group significant terms/phrases using a syntactic approach, is valid and can improve document retrieval.

Nevertheless, certain drawbacks of the linguistic approach for single term selection such as limitations of removing ambiquity from nouns and complexity of linguistic analysis systems, resulted in the development of statistically based techniques. Many useful mathematical models based on the vector space approach and probability theory have been introduced and used for successful document indexing since the seventies. Moreover, as stated by van Rijsbergen(19), Luhn's hypothesis in the late 50s already stated that term power or significance drops off at higher frequency. It means that frequently used words tend to be short function terms which are unable to determine distinction between relevant and nonrelevant documents. Several tests based on the constant rank frequency law of Zipf, have been carried out to confirm the above mentioned theory of frequently used terms. Zipf's law states that, if the distinct terms in the text are arranged in the decreasing order of their frequency of occurrence, then the frequency of occurrence of a word in the frequency order multiplied by its rank in the order is approximately constant i.e.

$$r \times f(r) \approx \text{constant}$$

(where *'r'* is a rank of term by frequency and *'f(r)'* is frequency of term at rank *'r'*). For example, if the term is 'and' with rank $r=3$ and the frequency of term at rank $f(r)=29$ than the constant will be:

$$\text{constant} = \frac{r \times f(r)}{N} = 0.087$$

(where N= 1000 and covers the number of terms in the document sample).

Test results indicated that authors often repeat certain amount of terms instead of substituting them by new words. It was also observed, that medium frequency words can be good indicators for relevant document retrieval. However, Salton emphasised that:

- the elimination of high frequency terms can cause losses in recall;
- elimination of low frequency terms may affect precision;
- there is no certain criterion to distinguish relevant medium frequency terms (20).

Finally, good indexing terms must retrieve relevant documents to which they are assigned as well as distinguish those documents from the whole collection of materials. To disclose the relevant terms more precisely, several additional automatic term extraction and statistical weighting models have been introduced for document indexing i.e.:

- inverse document frequency model;
- signal noise ratio model;
- term discrimination method (21).

The inverse document frequency model measures how frequently a definite term occurs both in the text of individual document and in the whole document collection. It has been proposed that term significance is proportional to the standard occurrence frequency of each word *k* in each document *i*, and inversely proportional to the total number of documents in which a word *k* occurs. In that case the formula for inverse document frequency will be:

$$IDF_k = \log_2 \frac{n}{DOCFREQ_k} + 1$$

where n is the total number of documents in the database and $DOCFREQ_k$ is the document frequency of the definite term k. For example, if within the collection of 1000 documents, the term PHYSICS occurs in 500 documents and the term BIOCHEMISTRY in 100 documents then the inverse document frequency for PHYSICS will be 2.000 but for BIOCHEMISTRY = 4.322. Therefore, the inverse document frequency method revealed that the most relevant indexing terms will be those which occur in a relatively small number of documents.

The *signal noise ratio* model determines term concentration in the document collection. The *noise* of an index term k for collection of n documents can be calculated as follows:

$$NOISE_k = \sum \frac{FREQ_{ik}}{TOTFREQ_k} \times \log \frac{TOTFREQ_k}{FREQ_{ik}}$$

where $FREQ_{ik}$ is the frequency of occurrence for term k in each document and $TOTFREQ_k$ is the frequency of occurrence for term k in the whole document collection. For example, if the term k=PHYSICS occurs once in each document and there are 1000 documents in the collection, then the noise ratio for this term will be

$$NOISE_k = \sum \frac{1}{n} \text{ x } \log_2 \frac{1}{n} = \log_2 1000 = 3$$

(n=number of documents in the collection). If the term occurs evenly in each document of the collection, the noise is growing. Non-specific terms tend to be evenly distributed and usually have a high noise ratio. An inverse function of the noise is signal ratio, which determines the most rarely used terms in the document. The signal for term k can be calculated as follows:

$$SIGNAL_k=\log_2(TOTFREQ_k)-NOISE_k$$

However, tests showed that signal value does not ensure effective performance in an information retrieval process. Overall, use of signal noise ratio method will retrieve only those documents which have indexing terms both with low document and document collection frequencies.

Term discrimination model is another method for selection of terms for automatic indexing. This method measures the degree to which the use of a term can help to distinguish documents from each other. If D_i and D_j are two documents and each of them is identified by a stem of index terms and if the similarity has been calculated for all pairs of documents (D_i, D_j), then the average similarity value, which represents the average document pair similarity in the collection, can be calculated as:

$$AVGSIM = CONSTANT \sum_{i=1}^{n} \sum_{i \neq j} \text{ x } \sum_{j=1}^{n} SIMILAR\ (D_i, D_j)$$

If the term k is removed from all documents in the collection and if $(AVGSIM)_k$ shows the document space density (the degree to which the document is clustered in the "space" of documents) then the discrimination value for each term k can be calculated as follows:

$$DISCVALUE_k = (AVGSIM)_k - AVGSIM$$

The $DISCVALUE_k$ can be calculated for all terms k and the terms will be ranked in decreasing order according to their discrimination values. For example, terms PANEL and FLUTTER from the Cranfield 424 document collection have the lowest DISCVALUE, so they will be placed on the top of the ranked list, whereas terms NUMBER and FLOW from the same collection contained high DISCVALUE, therefore

they will be ranked as poor discriminators (22). Ideal retrieval environment should be multidimensional index term space were all documents are apart as far as possible. The quality of term discrimination can be evaluated taking into account inter-document operations i.e. when the term is and when the term isn't used for indexing. A good indexing term will increase inter-document separation and decrease the space density. Experiments confirmed that the best term discriminators are medium frequency terms in the document collection in which they occur (23). However, Biru (24) suggested that medium frequency indexing terms can include also terms with poor discrimination capabilities. Overall, the perfect indexing term would be one "which minimises the separation of the relevant documents" (25).

As mentioned above, experimental evaluation tests confirmed that statistically based term selection methods can be effective in document indexing and information retrieval processes (26). However, the complexity of statistical term selection determined that instead of implementing one or another mathematical model for relevant term retrieval, it is easier to extract all keywords from the document and the query, and then differentiate them using an automatic word conflation approach coupled with appropriate term weighting module. The exception are terms with extremely high frequency which can be eliminated by the stopword list. It means that non-context bearing words would be automatically refused as document indexing terms as well as being stripped out from the user query. The extraction of stopwords increases the separation of documents in the database and also reduces the size of dictionary file in the structure of inverted file.

Along with the before mentioned mathematical models, word conflation based on automatic stemming algorithms is widely used in document indexing procedures. After elimination of frequently used words (via stoplist), stemmers determine most appropriate word stems (indexing terms) which represent document content. Despite there existing some experimental systems which deal with phrases i.e. the statistically based phrase indexing systems INDEX and INDEXD, the majority of term conflation algorithms process single terms i.e. MARS, MORPHS. Experiments and tests with several information retrieval systems which incorporate word conflation procedure i.e. SMART, also confirmed that automatic indexing based on single terms can considerably improve the document recall ratio. As computerised term conflation is one of the basic elements in the majority of statistically based advanced information retrieval systems, a detailed overview on word conflation procedure and stemming algorithms will be given in Chapter 3.

2.3 Best match searching and relevance feedback

Besides automatic word conflation, which ensures term selection based on words stems, the best match search method and relevance feedback coupled with term weighting functions can also be used to improve information precision and recall ratio. A best match search is often called nearest neighbour method or ranked output search. This approach compares a set of keyword stems from the query against the set of stems of indexing terms which represent document content in the database (27). After that a measure of similarity between the query and each document in the database is calculated and all documents are presented to the end-user in ranked order (28).

The advantages of best match retrieval comparing with the conventional Boolean search, are following:

- best match search does not require users to compose Boolean operations;
- term weighting mechanism helps to determine relevance between the query and documents in the database;
- ranked list of documents gives possibly the most relevant items from the top;
- if the system incorporates a query expansion module, the end-user can make his/her own judgments about the relevance of retrieved documents.

The basic requirements for successful implementation of the best match search are:

- choice of effective nearest match algorithm and
- use of appropriate term weighting mechanism.

An inverted file structure may be useful for the implementation of nearest neighbour searching as such structure allows the scanning all documents in a database for ranked output to be avoided (29).

Query expansion based on relevance feedback is another approach which can improve recall ratio in information retrieval systems. User queries often contain terms which do not match document indexing terms, therefore resulting in no document retrieval (30). Modification of the initial search query by adding and/or deleting search terms can increase the retrieval of relevant documents from the database.

There are two basic components which are used for query expansion and relevance feedback:

- change of terms in the initial query;
- reweighting of query terms (probabilistic theory).

Early tests and experiments on query expansion suggested to calculate and add closely related terms to the original query using vector space model. For example, in the SMART system documents and queries are represented by a vector of terms, where "term" means any form of document and/or query content identifier i.e. a stem of a word (31). Therefore, a document DOC_1 can be identifiedby a collection of terms (stems) e.g. $DOC_1 = STEM_1, STEM_2 \ldots STEM_t$. A document collection may be represented as an array of terms e.g.

	$\underline{STEM_1 \quad STEM_2 \quad \ldots \quad STEM_t}$
DOC_1	$STEM_{11}\ STEM_{12} \ldots STEM_{1t}$
DOC_2	$STEM_{21}\ STEM_{22} \ldots STEM_{2t}$
.	
.	
.	
DOC_n	$STEM_{n1}\ STEM_{n2} \ldots STEM_{nt}$

Similarly, a query can be identified as a vector of query terms (stems) i.e. QUERY = $QSTEM_1, QSTEM_2 \ldots QSTEM_t$.

Figure 2.3 represents an example where three stems identify documents according to the queries. Each axis match a different stem and the position of each document vector in the space is determined by the weight of the stems in that particular vector. The similarity between any two vectors is represented as a function inversely related to the angle between them. The retrieved documents may be arranged to the user in decreasing order of their similarity values according to the search requests.

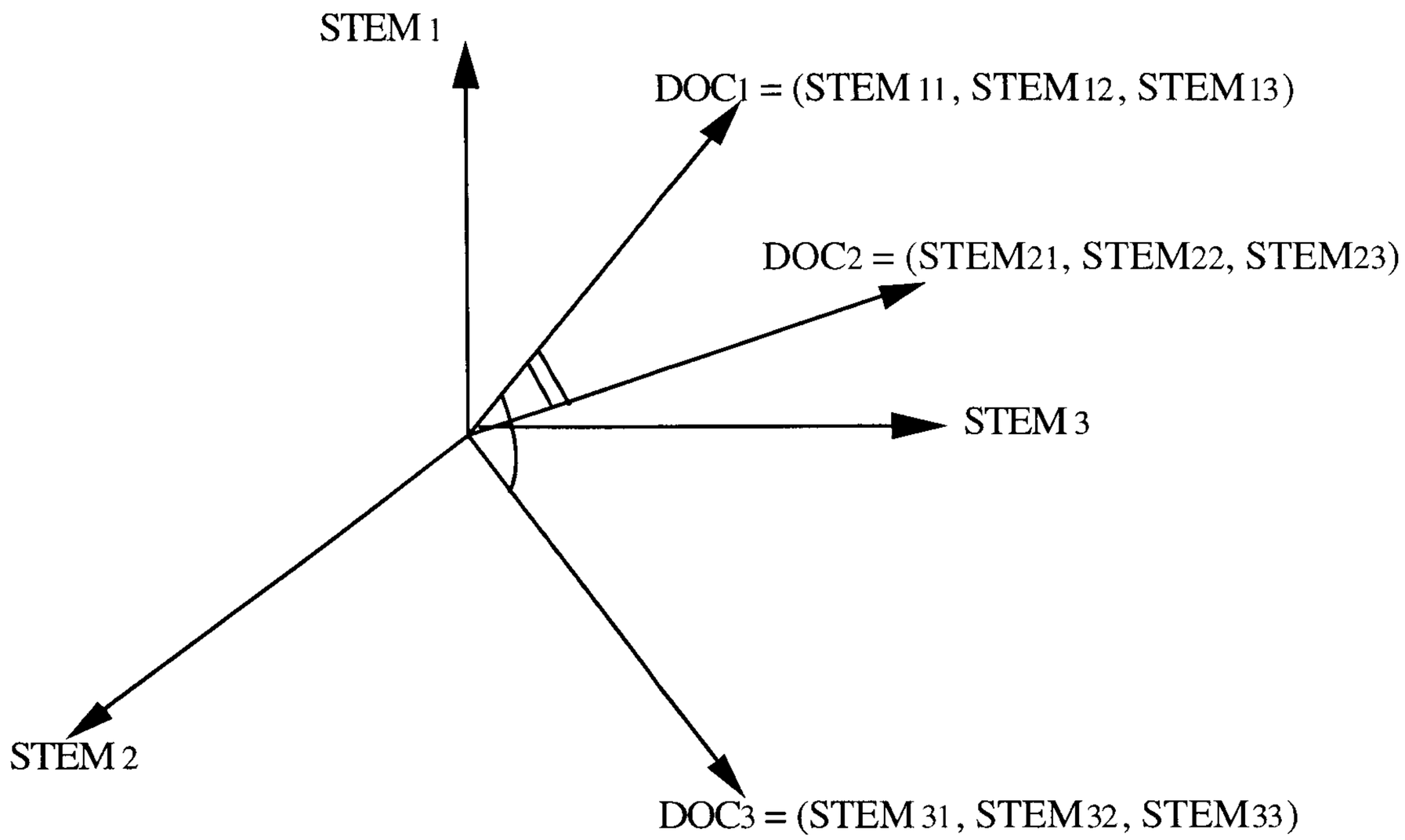

Figure 2.3 Multidimensional vector representation of query space

Research related to term (stem) reweighting without query expansion is mostly based on the probabilistic model which determines distribution of terms in relevant and nonrelevant documents. For example, Robertson and Spark Jones introduced a formula for calculation of term (stem) weights:

$$W_{ij}=\log_2 \frac{r}{(R-r)} \; / \; (\frac{n-r}{N-n-R+r})$$

where W_{ij} is the stem weight for stem i in query j; r= the number of relevant documents for query j with a stem i; R= the total number of relevant documents for a query j; n= the number of documents in the collection having stem i and N= the total number of documents in the collection (32). For example, if the stem ELECTRIC in a query ELECTRICAL ENGINEERING has 23 relevant documents, if the total number of relevant documents for this query is 31, if the total number of documents in the collection having stem ELECTRIC is 56 and the total number of documents in the collection is 164, then the weight of a stem ELECTRIC in a query ELECTRICAL ENGINEERING will be:

$$W = \log_2 \frac{23}{(31-23)} \; / \; \frac{(56-23)}{(164-56-31+23)} = \log_2 \frac{23}{8} \; / \; \frac{33}{100} = 0.94$$

According to this example, the stem ELECTRIC for the query ELECTRICAL ENGINEERING has high weight, so therefore it is a good document content discriminator.

The above described method assumes that the number of relevant documents is known before an initial query is submitted. Nevertheless, tests with manually indexed Cranfield 1400 collection performed good results (33). Salton suggested reweighting query terms by increasing weights of terms from relevant documents and decreasing weights of terms from nonrelevant documents (34). Along with the mentioned term reweighting approaches, the inverse document frequency (IDF) method described in the Section 2.2 is also used for determination of relevant terms for the query. It means that IDF calculates the most important terms within the document collection and within the particular document, and presents those for users for expansion of the initial query.

The initial query can be expanded without term weighting i.e. by using a thesaurus which automatically adds synonyms, broader terms and other relevant words. There have been several attempts to construct a thesaurus which would be based on a term to term association or clustering methods. For example, Harman (35) analysed term-term association (nearest neighbour technique) and suggested that users should have a possibility to !filter the new query terms. It means that users can select relevant terms from the list and add them to a new query. User filtering method with a selection of additional terms from relevant documents performed considerably better results then use of term to term clustering approach.

Query expansion and best match document ranking have been implemented in several experimental systems i.e. OKAPI, CITE. For example, in the CITE system after a user has entered the query in natural language, a list of ranked documents will be offered asking for his/her evaluation and eventual query expansion by adding new terms to the initial query.

2.4. Further developments in IR systems

There are several new areas of research, which might change IR systems in the future:

cluster analysis or automatic classification;

knowledge based approach and expert systems for automatic natural language processing (36).

Automatic classification or cluster analysis is a multi-level statistical method which determines the automatic identification of groups or clusters of similar objects. There are two types of clustering which can radically enhance retrieval of relevant documents:

document clustering on the basis of common terms;

term clustering on the basis of the documents in which they occur.

Use of document clustering can improve performance as the relationships between the file organisation, search mechanism and the documents in a database. Term clustering can enhance document recall ratio as this approach involves term classification to substitute each document term and/or query term by the identifier of the cluster which contains that term. Term classification can also be used to expand the query by adding terms from the clusters which contain one of the initial query terms. Therefore, term classification ensures additional matches between sets of document and query terms.

Implementation of expert systems may also improve access to relevant documents and information in the database. Along with offline query formulation and automatic logon, intermediary systems can also include relevance feedback and advisory techniques which guide the user during the search. Intelligent front-ends are another important part of such expert systems which provide user the opportunity to formulate queries in a natural language. Moreover, intermediary information retrieval system may also help user to select most appropriate search terms and search strategy. Research on knowledge-based information retrieval techniques (i.e. that suggested by Tong and Shapiro) revealed that the implementation of rule-based methods can provide retrieval of factual information from textual databases.

2.5 Conclusion

Automatic document indexing, query expansion and best match searching coupled with computerised word conflation are the basic elements for relevant document search and retrieval in an advanced IR system. Experiments and tests revealed that use of stemming for selection of document content identifiers (during document indexing procedure) as well as for query processing and modification performed far more better results then manual indexing and right hand truncation. Stemming is also the core method for query expansion and document ranking based on user relevance feedback. It allows augmention of the initial query by changing or adding new stems, therefore increasing precision in document retrieval. Stemming together with term (stem) weighting approaches i.e. term discrimination, inverse document frequency model, can also improve the choice of equivalent indexing terms as well as will determine appropriate terms for expansion of the initial query.

Research and experiments also revealed that advanced information retrieval systems are able:

to retrieve larger amount of relevant information than conventional IR systems;

to replace trained and experienced intermediaries by user friendly front-ends for end-users with limited experience, i.e. advanced IR systems which include query expansion and best match search modules will present all retrieved documents in a ranked order as well as allow the user to change the initial query.

As mentioned before, stemming allows successfully carry out automatic indexing, query expansion and best match search. The following chapter will analyse the basic types of stemming algorithms and give an overview both of English and non-English language stemmers, which have been designed and implemented in various information retrieval systems.

References

1. **Willett, Peter.** Introduction. In: P. Willett, ed. *Document retrieval systems*, 1988, p. 1.

2. **Salton, Gerard & Michael J. Mc.Gill.** *Introduction to modern information retrieval*, 1983, p. 7.

3. **Frakes, W.B.** Introduction to information storage and retrieval systems. In: William B. Frakes & Ricardo Baeza-Yates, eds. *Information retrieval: data structures and algorithms*, 1992, p. 2.

4. **Van Rijsbergen C.J.** *Information retrieval*, 1975, p.6.

5. **Harman, Donna et al.** Inverted files. In: William B. Frakes & Ricardo Baeza-Yates eds. *Information retrieval: data structures and algorithms,* 1992, 28-29.

6. **Ashford, John & Peter Willett.** *Text retrieval and document databases*, 1988, p. 64.

7. **Frakes,** ref. 3, p.5

8. **Willett, Peter.** Automatic indexing of documents and queries. In: P. Willett, ed. *Document retrieval systems,* 1988, p. 4.

9. **Harter, Stephen P.** A probabilistic approach to automatic keyword indexing. *Journal of the American Society for Information Science*, 1975, **26**(4), 285.

10. **Van Rijsbergen**, ref. 4, p. 23.

11. **Schamber, Linda.** Relevance and information behavior. In: Martha E. Williams, ed. *Annual review of information science and technology*, 1994, p. 13.

12. **Salton,** ref. 2, p. 59.

13. **Van Rijsbergen,** ref. 4, p. 23.

14. **Willett,** ref. 8, p. 4.

15. **Peat, Helen J. & Peter Willett.** The limitations of term co-occurence data for query expansion in document retrieval systems. *Journal of the American Society for Information Science*, 1991, **42** (5), 378.

16. **Smeaton, Alan F.** Natural language processing and information retrieval. *Information Processing and Management*, 1990, **26**(1), 19.

17. **Spark Jones, K. & J.I.Tait.** Automatic search term variant generation. *Journal of Documentation*, 1984, **40**(1), 58-60.

18. **Dillon, Martin & Ann S. Gray.** FASIT: A fully automatic syntactically based indexing system. *Journal of the American Society for Information Science*, 1983, **34**(2), 99-108.

19. **Van Rijsbergen**, ref. 4, p. 25.

20. **Salton,** ref. 2, p. 62.

21. *Ibid*, p. 63.

22. *Ibid,* 66-68.

23. *Ibid,* p. 63.

24. **Biru, Tesfaye et al.** Inclusion of relevance information in the term discrimination model. *Journal of Documentation*, 1989, **45**(2), 86.

25. *Ibid*, p. 95.

26. **Smeaton**, ref. 16, p. 19.

27. **Willett, Peter**. Best match searching. In: P.Willett, ed. *Document retrieval systems*, 1988, p. 10.

28. **Ashford**, ref. 6, p. 65.

29. **Perry, Shirley A. & Peter Willett**. A review of the use of inverted files for best match searching in information retrieval systems. *Journal of Information Science*, 1983, **6**, 59.

30. **Harman, Donna.** Relevance feedback and other query modification techniques. In: William B. Frakes and Ricardo Baeza-Yates, eds. *Information retrieval: data structures and algorithms,* 1992, 241-263.

31. **Salton**, ref. 2, 118-156.

32. **Robertson, S.E. & K. Spark Jones.** Relevance weighting of search terms. *Journal of the American Society for Information Science,* 1976, **27**, 129-146.

33. *Ibid*, p. 135.

36 **Salton,** ref. 2, 59-75.

35. **Harman,** ref. 30.

36. **Willett, Peter.** Other research areas. In: P. Willett, ed. *Document retrieval systems,* 1988, 19-22.

Chapter 3

WORD CONFLATION AND STEMMING ALGORITHMS

3.1 Introduction and definition

According to Frakes(1), word conflation is a process of matching morphological term variants. As noted in Chapter 2, conflation or reduction of word variants to a single canonical form is used both in document indexing procedures to define most appropriate document content descriptors as well as in information retrieval to determine relevant query terms which match document indexing terms. Therefore, word conflation can be done at indexing time and/or at search time.

Conflation of terms can be achieved either manually or automatically. Right hand truncation is one of the most often used techniques for manual conflation. Many conventional online systems i.e. ERIC, STN allow the searcher to truncate query terms by using wildcard characters i.e.asterisk (*). For example, more records on the subject ELECTRICITY will be retrieved if the initial search term is truncated to ELECTRIC*. However, users often are unfamiliar with the truncation approach. Willett(2) stressed that two major problems are associated with the manual right hand truncation:

- over truncation which means that the remaining stem of a word is too short after truncation;
- under truncation which causes retrieval of too few related relevant words.

For example, in the case of a user over truncating the word ELECTRONICS to ELEC, then both all words related to ELECTRICITY and ELECTRONICS as well as completely unrelated words i.e. ELECTIONS will be retrieved. In the case of a word being under truncated, a user will retrieve very few if any relevant word e.g. if the word COMPUTERS is truncated to COMPUTER, than all relevant documents related to COMPUTING and COMPUTATIONAL will not be retrieved. Walker and Jones (3) also observed that manual truncation is not often used by users as it demands certain experience and skills. Therefore, the use of manual word conflation in information retrieval systems and OPACs requires trained intermediaries who can help users to overcome the above mentioned problems.

Automatic term conflation includes special programs called stemming algorithms or stemmers which reduces morphological variants of a word to a one, single form. Lovins, who designed one of the first automatic word conflation programs, defines a stemming algorithm as a "computational procedure which reduce all words with the same root (or, if prefixes are left untouched, the same stem) to a common form, usually by stripping each word of its derivational and inflectional suffixes" (4).

The following section of this chapter will characterise four types of automatic conflation methods. It will be followed by an analysis and evaluation of existing stemming algorithms used in operational and experimental IR systems. A separate section will deal with the design and development of non-English language stemmers. Finally, the last section will investigate several experimental evaluation methods for stemming algorithms.

3.2 Types of stemming algorithms

Automatic term conflation comprises four different approaches which are summarised in Figure 3.1

- table lookup;
- successor variety;
- n-gram method;
- affix removal (5).

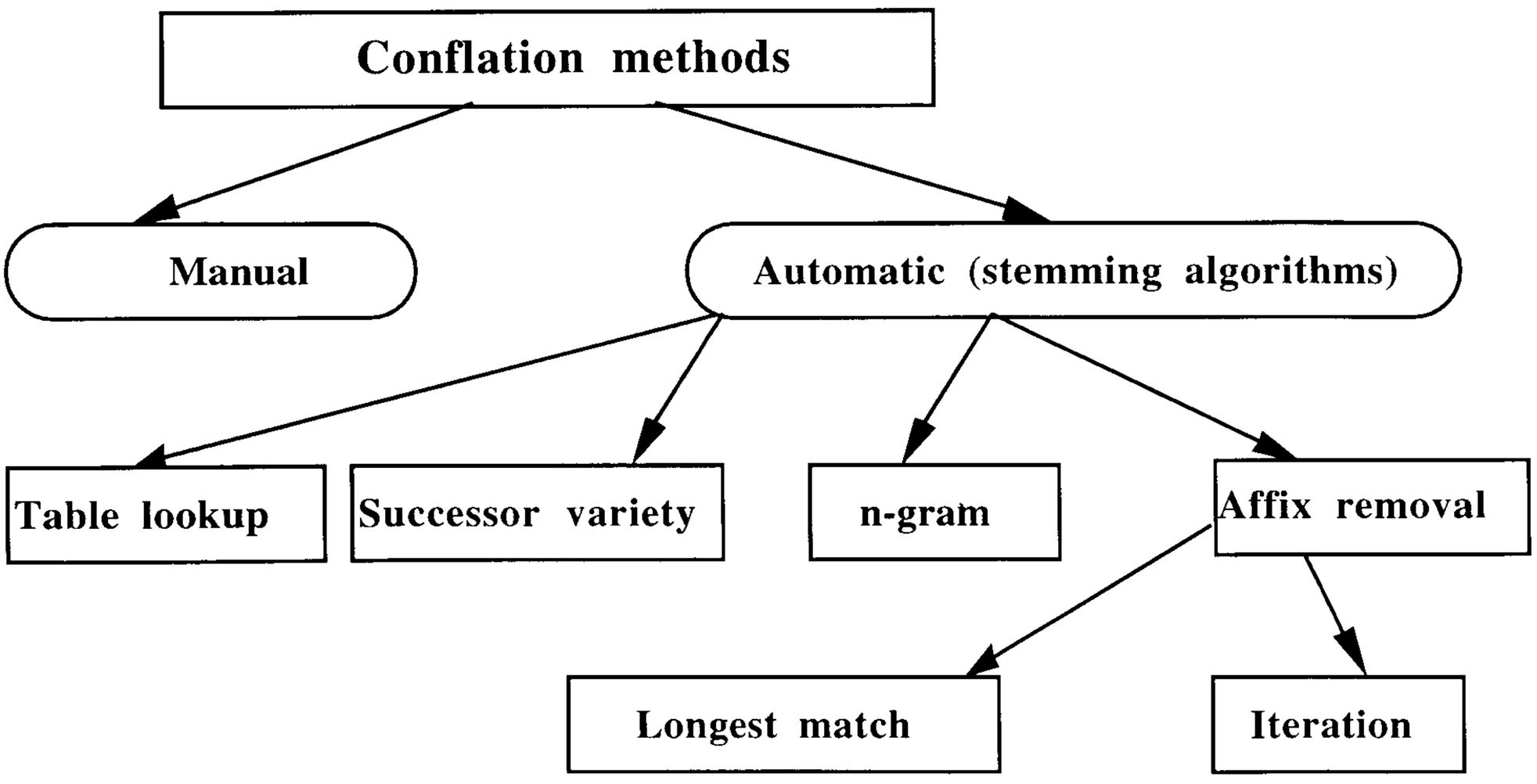

Figure 3.1 Word conflation techniques

The following paragraph outlines general principles of each automatic conflation procedure mentioned above. Detailed description of automatic stemming methods are given below in separate subsections.

In the table lookup method stemming is done via tables which contain index terms and their stems. User query terms can than be stemmed through those tables using a B-tree or hashing. Successor variety is an approach were a stemming algorithm is used to analyse letter sequences in a document corpus. First of all the letter successor varieties for a word are determined and than the word is segmented using cutoff, peak and plateau, complete word or entropy methods. When the word has been segmented, the most relevant stem is selected. The n-gram conflation method is based on term digrams or n-grams. First of all a word is divided into digrams and the unique digrams for each word is determined. After that a similarity matrix is calculated and terms will be grouped together using single link clustering. Finally, affix removal algorithms remove suffixes and/or prefixes from the word leaving a root/stem. Affix removal stemmers can incorporate either iterative (endings and suffixes will be removed step by step) or longest match approach (suffix will be stripped in one step) or both.

3.2.1 Table lookup method

The table lookup method stores all terms and their corresponding stems in a table (See Table 3.1). Each query term is than stemmed via table lookup using B-tree or hashing. *B-tree* is a multi-level tree-structured index, where all associated terms are stored in leaves or buckets. The search for a definite stem can be done by moving down the tree structure and choosing the appropriate branch.

Table 3.1 Example of term storing for table lookup method

Terms	Stems
ELECTRICITY	ELECTRIC
ELECTRICIAN	ELECTRIC
ELECTRIC	ELECTRIC

For example, to find the corresponding stem for words ELECTRICITY and ELECTRICAL, the B-tree stemmer will search for a pointer to the appropriate stem(s) (Figure 3.2). It means, that table lookup method locates both words ELECTRICITY and ELECTRICAL in the B-tree and than follows the pointer to their stem in a separate lookup table.

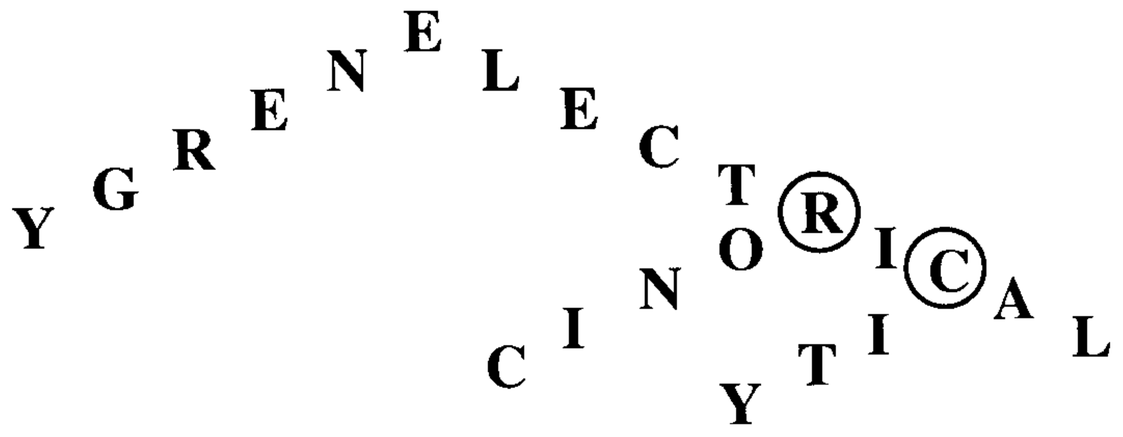

Figure 3.2 Example of B-tree approach for table lookup stemmers

Hashing is another method, which can be used in stemming via table lookup. Hashing maps a key (letters in a word) to a value in a given range. A hashing function produces values which are evenly distributed over a given range. The hashing value gives the table lookup for a stem. For example, the word ELECTRICITY can be hashed as follows. If the letter E has been assigned the integer 5, L=12, C=3, T=20, R=18, I=9 and Y=25, than the signature or address for ELECTRICITY will be the sum of corresponding integers i.e. H= $\sum$ E + L + E + C + T + R + I + C + I + T + Y = 129. At position 129 in the lookup table will be a pointer to the stem of "ELECTRICITY".

The case when two or more different terms have the same address or signature is called *collision*. It means that if the hashing value of terms ELECTRICITY, TECHNOLOGY and COMPUTING is equal, they will be stored in the same slot of hashing table. Collision can be overcome by using either open addressing or overflow addressing techniques. Open addressing approach rehashes the collided term into a table by computing a new index value. It means that double hashing is used there to calculate a new signature for the term. However, some problems may arise if the hashing table is becoming full. In that case the structural reorganisation in the table must be done.

In *overflow addressing* the collided terms are stored in an overflow area and all terms with the same hashing values are linked together. The limitation of this method is that the search may be degenerated to a simple linear search.

Recently several hybrid methods have been devised to improve information retrieval time on B-trees and to range searches in hashing tables, i.e. the *bounded disorder* technique, which organises leaves in B-trees as hashing tables.

The use of table lookup method has several limitations:

1. A table is required for all words and their stems in a particular language. It is unlikely that such a table would contain every word in the language.
2. Storage of this table usually requires a considerable amount of disk space.

3.2.2 Successor variety

The successor variety method deals with letter distribution in the word and word segmentation to determine word and morpheme boundaries in the lexical text. Hafer and Weiss (5) who designed the letter successor variety stemmer, defined a as a test word with the length n, a_i as a prefix of word a with the length i. D was defined as a corpus of words and D_{ai} as a subset of D containing words whose first i letters match a_i exactly. The successor variety of a_i denoted S_{ai} is defined as the number of distinct letters that are in the *i+1st* position of words in the word subset D_{ai}. A test word with the length n has n successor varieties S_{a1}, S_{a2} ... S_{an}. It means that the successor variety of the string (corpus of words) is the number of different letters that follow in words in the corpus.

For example, if the test word is ELECTRIC and the corpus D_{ai} contains following words:

ENERGY
ELLIPSE
ELEMENT
ERROR
ELASTICITY
DATABASE

than to determine the letter successor variety for ELECTRIC, each letter in this test word must be compared with corresponding letters in the word corpus. The first letter of ELECTRIC is "e" and it is matched by five words. "e" is followed by three letters "n", "l" and "r" so, the successor variety of "e" is three. The next successor variety would be also three as "a", "e" and "l" follows "el". The complete word ELECTRIC matches no terms therefore, it has a successor variety of zero. Table 3.2 summarises letter variety counts for ELECTRIC.

Table 3.2 Successor varieties for the test word ELECTRIC

Letters of terms	Successor variety	Letters of successor variety
E	3	N, L, R
EL	3	L, E, A
ELE	1	M
ELECTRIC	0	blank

It was observed, that within a word, the *ith* letter is independent to a certain degree on the *i-1* letter that precede it. For example, the first letter "e" in the word ENERGY is unrestricted as there are no predecessors. The next letter "n" becomes more restricted as

it must be compatible with "e". Within the word units the successor variety tends to decrease from left to right, and especially near the end of a long word the successor variety S_{ai} becomes small.

After the successor varieties for a word has been found, the obtained results are used to segment the word. Hafer and Weiss (6) defined four basic strategies for word segmentation:

- cutoff method;
- peak and plateau technique;
- complete word method;
- entropy method.

The *cutoff method* segments a word by selecting some cutoff value K. The word boundary (stem and affix) is identified if its successor or predecessor variety reaches or exceeds the cutoff value. The method is easy to implement but it requires selecting of the suitable cutoff value, because if the value is too small, many incorrect cutoffs will be done. If the value is too large, than many correct cuts will be left out.

Using the *peak and plateau* strategy, the cut in a word is done after the prefix a_i, if only the successor variety $S_{ai} \geq S_{a-1}$ and $S_{ai} \geq S_{a+1}$. It means that the segmentation is done after a character which letter successor variety exceeds the successor variety value of the letter preceding it and/or following it. This method eliminates the necessity to define a specific cutoff value.

The *complete word* method produces the segmentation after the prefix of a word or before the suffix of the word, if the prefix and/or suffix is a complete word in the corpus. For example, if the test word is ANTIELECTRIC and ANTI appears as a word in the corpus, the break will be made after ANTI. However, use of this

The three above mentioned approaches are based on the variety of successor and predecessor letters, whereas the *entropy method* uses distribution of those letters in a word. For words with unusually high successor or predecessor counts i.e. foreign words or abbreviations, the calculation of letter distribution in those words help to avoid segmentation errors. The entropy approach allows the weighting of the importance of each successor and/or predecessor letter in a word by its probability of occurrence. For example, if $|D_{\alpha i}|$ is the number of words in a text fragment with the i length sequence of letters α and $|D_{\alpha ij}|$ is the number of words in $|D_{\alpha i}|$ with the

successor letter *j*, than the probability that a word of $D_{\alpha i}$ has the successor letter *j* can be calculated as follows:

$$p = \frac{|D\alpha ji|}{|D\alpha i|}$$

The entropy $H\alpha i$ of all words $|D\alpha i|$ in a text fragment can be determined using following formula:

$$H_{\alpha i} = \sum_{p=1}^{26} - \frac{|D_{\alpha ji}|}{|D_{\alpha i}|} \times \log_2 \frac{|D_{\alpha ji}|}{|D_{\alpha i}|}$$

For example, if two words *W1* and *W2* with *i* letter prefix both have successor variety 10, and the first *i* letters of *W1* match 100 words in the text corpus whereas the first *i* letters of *W2* match only 19 words, the probability of *W1* and *W2* hence is 0·1 and 0·53. According to the formula, the entropy *H* for the *i* letter prefix of W1 is 3·3 and for W2 is 2·5.

This equation can also be used to calculate entropy measures for predecessor letters in a word. A cut in the test word can be done if one or both entropies have reached some cutoff value.

Hafer and Weiss carried out 15 various experiments to evaluate correctness and efficiency of the above described word segmentation methods. Analysis revealed that none of the methods performed with satisfactory results, although the use of cutoff and/or peak and plateau approaches may increase the number of relevant word cuts.

After a word has been segmented, the most relevant segment, which can be used as a stem must be considered. Hafer and Weiss (7) introduced the rule, that if the first segment occurs in more than 12 words in the corpus, than it is likely a stem of those words. It means that many segments come from one word.

Overall, the successor variety method comprises three stages:

- determination of letter successor varieties in a word;
- word segmentation based on one of the above mentioned segmentation methods;
- selection of the appropriate stem.

For example, for a test word CHANGEABLE first of all the letter successor variety was determined. After that the most appropriate word segmentation method was selected. In the case, if a word corpus contained CHANGE, the most relevant

techniques for word segmentation was either complete word segmentation or peak and plateau method. Both these approaches segmented CHANGEABLE into CHANGE and ABLE. Finally, the relevant segment of the word, which was used as a stem, has been chosen. Assuming that the first segment occured in more than in one word in a corpus and according to the Hafer and Weiss rule, the stem for the word CHANGEABLE was CHANGE.

The stemming process via successor variety is more mechanical, when compared with other algorithms because it does not require human effort to prepare suffix lists and/or specific affix removal rules. It was observed that the successor variety method is flexible for the determination of segmentation rules and is more adaptable for various kinds of document collections as well as for new languages. However, stemming results also revealed that often words have not been associated with correct stems e.g. a word WIVES was not associated with words which are belonging to the stem WIFE. In several cases a word will be associated with a completely wrong stem i.e. the word ELECTRICAL associated with the stem ELECT.

3.2.3 N-gram stemmers

Automatic word stemming can be also done by using the digram method. A digram is a pair of consecutive letters. The n-gram method allows to calculate association measures between pairs of words based on shared unique digrams.

The example below demonstrates how the words ELECTRIC and ELECTRONICS can be divided into digrams:

electric = el le ec ct tr ri ic
unique digrams = el le ec ct tr ri ic

electronics = el le ec ct tr ro on ni ic cs
unique digrams = el le ec ct tr ro on ni ic cs

The word ELECTRIC has seven digrams, all of which are unique. The are ten unique digrams in the word ELECTRONICS. Both words share six unique digrams i.e. el le ec ct tr ic.

After the unique digrams for a word pair have been identified, a similarity measure can be calculated using Dice's coefficient:

$$S = \frac{2 \times C}{(A + B)}$$

where A is the number of unique digrams in the first term, B is the number of unique

digrams in the second term and C the number of unique digrams shared by A and B. Dice's coefficient for the words ELECTRIC and ELECTRONICS will be:

$$S = \frac{2 \times 6}{(7 + 10)} = \frac{12}{17} = .70$$

The similarity measures of the all pairs of words in the database are forming a similarity matrix. As noted by Frakes (8), Dice's coefficient is symmetric, so a lower triangular similarity matrix can be implemented e.g.

	$word_1$	$word_2$	$word_3$	...	$word_{n-1}$
$word_1$					
$word_2$	S_{21}				
$word_3$	S_{31}	S_{32}			
...					
$word_n$	S_{n1}	S_{n2}	S_{n3}	...	$S_{n(n-1)}$

After the similarity matrix has been defined, all terms can be clustered using *a single link clustering* method. The single link combines together the most similar pairs of words in a data set.

Table 3.3 Similarity coefficients for terms A to D

Step	Pair	Similarity
1	AD	0.8
2	AC	0.6
3	BD	0.6
4	BC	0.6
5	AB	0.7
6	CD	0.6

For example, Table 3.3 presents similarity coeficients for each pair of four words, which are labelled A to D where A = ELECTRONICS, B = ELECTRIC, C = ELECTRICITY and D = ELECTRONICAL. Figure 3.3 shows how all four words according to calculated similarity values can be clustered together using the single link approach.

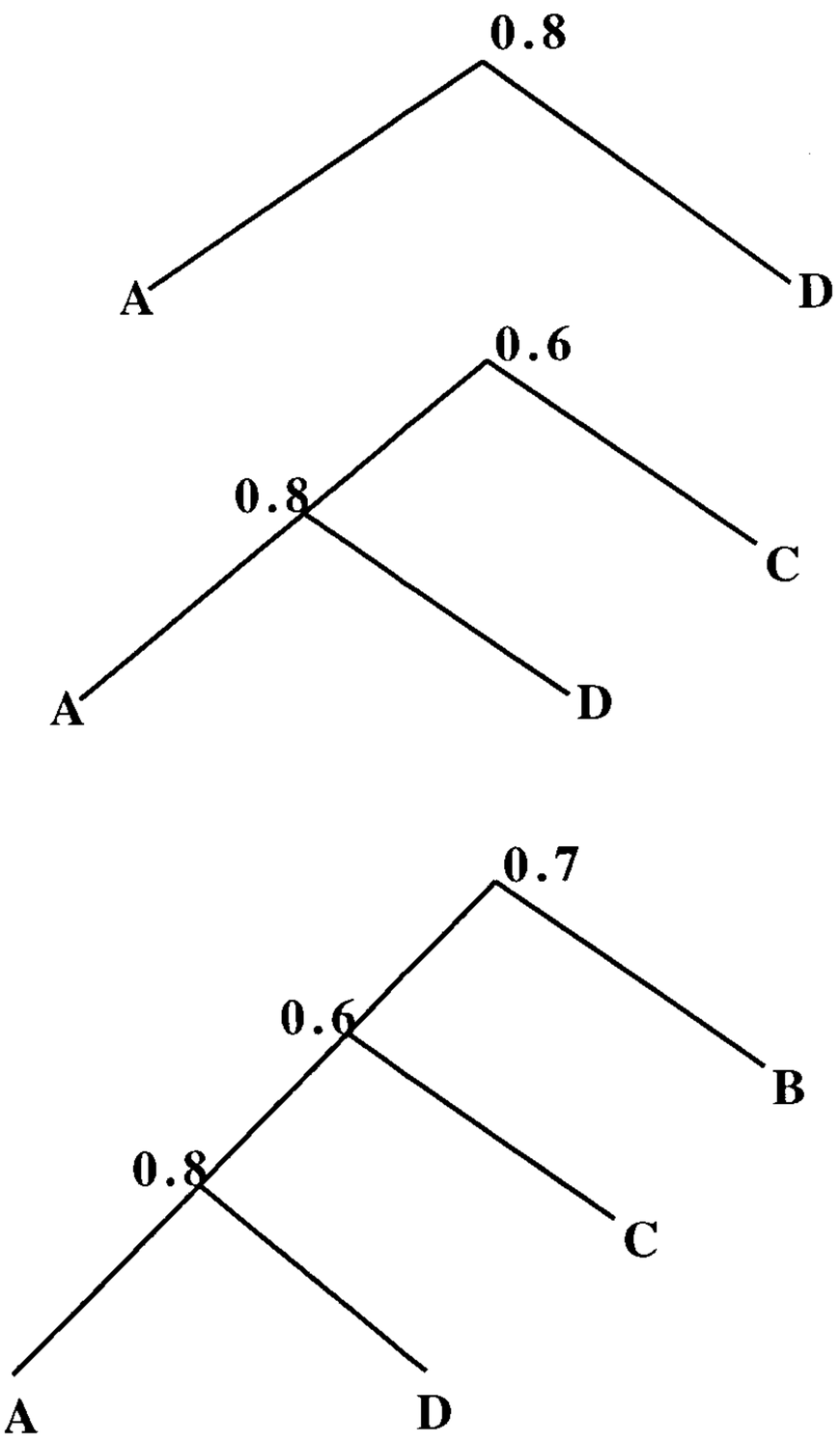

Figure 3.3 Example of single link structure

It has been defined that the distance between two clusters is the distance between the closest pair of words in one of the two clusters. Therefore, there is no need to determine some central cluster and/or to recalculate the similarity matrix. Van Rijsbergen's algorithm and the SLINK algorithm are the most often used methods for presenting single link clusters (9). Van Rijsbergen's algorithm generates the single link hierarchy, which presents the similarity values in any order and does not require the storage of the similarity matrix. The SLINK algorithm is based on a number of operations by which a determination of the single link hierarchy and its updating can be optimally done, therefore the algorithm is efficient for large data collections.

Overall, the n-gram method is based on the clustering principle which means that similar words are grouped together. After determination of unique and shared digrams and computing the similarity coefficient for each pair of words in the set, these coefficients are used for a clustering algorithm. Automatic word conflation is achieved

by considering that all words in a given cluster are equivalent. Stemmed words may comprise different spellings of the same word as well as morphological variants.

The n-gram stemming approach performed good results in particular subject areas i.e. chemistry by testing definite type of words i.e. document titles. For example, Adamson and Boreham (10) observed that the n-gram method correctly calculated similarity measures and successfully clustered document titles from *Chemical Titles.* However, implementation of the n-gram method requires an extremely large amount of computation to cluster any data dictionary (11).

3.2.4 Affix removal stemming algorithms

Affix removal is the most often used method, which removes suffixes and/or prefixes from the word leaving a stem. According to the Figure 3.1 affix removal stemmers incorporate either iteration or the longest match approach, or both.

The *iterative* procedure removes suffixes in several steps starting from the end of the word. For example, using Porter's stemmer, which is based on the iterative method, the word ELECTRICITY will be processed in two steps. First of all, the letter Y will be substituted by letter I and after that the ending -ICITI will be transformed to -IC (ELECTRIC), by removing -ITI from the end of the stem.

The *longest match* approach removes an ending of a word in one step. It means that within each group of endings, if there is more than one relevant match, than the longest ending will be removed from the stem. For example, the Lovins longest match stemmer will remove ending -ICAL from the word ELECTRICAL in one iteration. Comparing with iterative stemmers, the longest match algorithms are often easier to program. However, as longest match stemmers include all compound suffixes, the size of a suffix dictionary is much bigger than it is for iterative stemmers.

Both longest match and iterative stemmers comprise certain rules i.e. conditional and/or recoding rules, which control the removal of relevant affixes from stems. *Conditional rules* often involve minimum length conditions, which prevent overstemming. For example, in Lovin's stemmer code B after definite endings states that suffix stripping will be carried out if the minimum stem length is at least three characters (12). Conditional rules can include also term specific rules, which prevent from stemming terms with certain endings.

Recoding rules usually deal with modifications of a term's resultant stem. For example, character -Y may be replaced by -I to retrieve more relevant stems (ENERGY-ENERGI). In case when the stem ends with a double consonant (e.g. GG) one of them will be removed by the certain recoding rule. The construction and implementation of recoding and conditional rules is one of the most difficult parts in affix removal algorithms, as they require a lot of time and have to be properly designed to produce relevant stems. The following section will deal with development of definite affix removal stemming algorithms as this type of stemmer, according to a number of evaluation studies (13), has achieved good information retrieval performance results.

3.3 Review of automatic stemming algorithms for English

Affix removal algorithms are one of the most often used stemmers for automatic word conflation, which give good recall and precision results in information retrieval. Therefore, the main emphasis in this overview is on stemming algorithms incorporating either longest match approach (word ending is removed in one step) or iteration method (suffixes are removed in several steps), or both.

3.3.1 Early achievements in stemmer construction (1965 - 1970)

One of the first automatic affix removal algorithms, which was based on longest match principle, was constructed in the sixties as the part of **Project Intrex** (1965-1973). The main purpose of this project was to design and develop an experimental integrated information storage and retrieval system, which would provide a user with interactive online access to documents including full texts of documents covering by this database (14). The subject area comprised information in Materials Science and Engineering. Subject indexing was done manually and each document in the database was given a phrase consisting of no more than ten nouns. Subject terms for each document were extracted from the subject index phrases by modifying the phrases into single words and by stemming those words. Subject word stems were arranged in alphabetical order and totally for 20,000 documents in the database, more than 31,000 different word stems has been stored in the alphabetical file.

One of the basic reasons for introducing a stemming procedure in Project Intrex was to improve retrieval effectiveness especially regarding recall and precision. Lovins (15) who designed and developed the stemming algorithm, mentioned three major previous attempts to construct affix removal algorithms. Tukey (16) had built a context sensitive partially iterative stemmer which grouped all endings into four groups. The first group covered only the letter s which can't be removed after i, s or u. The second group was recursive, the third was non-recursive and the fourth group included remaining terminal consonants. All groups had restrictions on stem lengths. Tukey's approach was complex because one group was based on the longest match principle; other on the iterative method.

The second algorithm was designed by Michael Lesk (17) at Harvard University. This stemmer was based on an iterative search for a longest match ending. If no more relevant matches were found, terminals (vowels) i,a and e were removed and then terminal consonants.

Dolby (18) in California developed the third stemmer which involved three stages. The first stage used a set of context dependent rules. The second stage was based on the longest match approach and was context free which meant that the first ending in any group which matched the term was accepted as valid. In this stage the endings were removed in any order and the only restriction was a minimum stem length of two syllables. The final stage was context dependent regarding inflectional forms of terms. Some principles and results from the above mentioned algorithms were used in building the Intrex stemmer. For example, a preliminary list of endings for the stemming algorithm of Project Intrex was partly based on a list of 194 term endings which was transferred from Lesk's work at Harvard University. The preliminary list of Intrex stemmer was analysed and evaluated against the output list of endings from Tukey's iterative stemmer.

Preliminary lists for the stemming algorithm were organised according to the endings from both normal and reverse English words. The structure of preliminary lists allowed to determine whether the removal of an ending will result in:

- two different stems;
- a stem not matching another relevant stem which it should match.

Both those conditions required to add new endings, to dispose of old ones and to apply new context sensitive rules. The final list contained 260 endings which were divided into eleven separate subsets, where endings were grouped according to their length. Within the each subset, endings were arranged in alphabetical order and each ending

was followed by one of 29 condition codes (context sensitive rules). The condition code consists of an alphabetical letter which characterises certain restrictions for the stem preceding the ending. For example, the resultant list of endings covers two different endings -ANCING in subset .06. and -ING in subset .03. for the test term DANCING. The first ending -ANCING cannot be removed as the condition code B requires the minimum stem length consisting of three characters. Code N which follows after the ending -ING, determines that the minimum stem length should be four characters if the next letter is s or three characters in other cases. It means that using this longest match algorithm the ending -ING will be removed from the term DANCING, leaving the stem DANC, which is four characters long.

There is a two step stemming routine in the Intrex algorithm which means that after removing the valid ending, each stem is checked against the list of 34 recoding rules. Recoding rules deal with the removal of double consonants from remaining stems e.g. consonant l from the stem coll (collate-col) as well as transferring one stem into another e.g. -ix to -ic (appendix - appendices).

The Intrex or Lovins longest match stemmer predominated good results in use. Overhage and Reintjies(19) observed that use of the word stemming algorithm, which removes all stopwords from the initial query and than compares the stemmed form of remaining words with stemmed indexing terms, is more superior than other information retrieval techniques. Choice of relevant search terms can also evidently improve information recall and precision ratio. For example, after modifying the initial search query "irradiation embrittlement of metals" to "irradiation embrittlement" the recall ratio changed from zero to 2 per cent with 100 per cent relevance. Further alternations of the same query which included the removal of “embrittlement” leaving only the word “irradiation”, resulted in 90 per cent recall with 40 per cent relevance of retrieved documents (20). A modified version of Lovins stemming algorithm was used for stemming indexing terms in the experimental in-house information retrieval system MASQUERADE, which covered various types of documents in geology and exploration (21).

The experimental fully automatic information system **SMART** was also designed in the middle of the sixties (22). The system involves a longest match stemmer which was based on a modified version of Lovin's algorithm. The SMART stemming algorithm included a list of more than 260 suffixes and several recoding rules. The stemmer operated as follows: for each word the longest possible suffix was determined, leaving valid length of stem which was not less than three characters. After that the word stem was matched against the exception list and in a case of a successful pass, the resultant

stem was formed. This last stage involved the use of recoding rules i.e. to change letter -Y to -I or to remove double consonant from the final stem.

3.3.2 Next generation of stemming algorithms (1970 - 1980)

Two stemming algorithms were analysed and evaluated for an information storage and retrieval system, which was part of the **RADCOL** project and which was designed in 1973 by Informatics for the Rome Air Development Centre (23). The first stemmer covered two stage passes through a list containing 95 suffixes. The other stemming algorithm which was based on a longest match approach, used only a single pass through a more extensive list of endings covering 570 suffixes, so this algorithm was chosen for the RADCOL project.

To create the suffix list, all characters of words which appeared in the index more than ten times were reversed. The reversed terms then were arranged in an alphabetical order. All the characters in neighbouring terms were compared and in a case of a match, strings of characters containing 1,2,3, ... n letters were grouped in a separate list. For example, besides the character string and suffix -ATION (i.e. for the term ORGANISATION), there were also such strings of letters as -N, -ON, -ION, -TION. All strings were systematised and the most frequent endings were grouped together for the final suffix list. This suffix list was also analysed and compared with the list of endings from Lovin's stemmer. Despite the extensive list of suffixes, the RADCOL longest match algorithm included only three recoding and two condition rules.

In 1974 the above described Intrex longest match algorithm was changed and modified by **Dawson** (24) who added more plural and simple suffixes to the list of endings. The final list comprised almost 1,200 suffixes which were arranged in reverse order according to their length to avoid problems related to storage and processing time.

Dawson's stemmer was based on a partial matching approach which means that terms are matched if their stem endings are almost identical e.g. -MIT and -MIS. The stemmer includes nearly fifty of such nearly identical groups of stem endings, and if two stems match on a definite number of characters and the remaining letters of each stem belong to the same group of stem endings, than both stems are stemmed to the same form. For example, if there are two test terms ADMISSION and ADMITTANCE with similar stems ADMISS and ADMITT, than after removing endings -ION and -ANCE as well as one of the double consonants s and t, the remaining stems according to the appropriate conditional rule would be transferred to one single stem ADMIS. The suffix

list and conditional rules for Dawson's stemmer were constructed manually using a Key Letter In Context (KLIC) index.

The KLIC index was also used for creating the word ending lists and corresponding conditional rules for the **INSPEC** stemmer (25). Single index words, which were assigned to each document, formed the basis of KLIC index. Each word was then stored under its constituent character and all terms were grouped in an alphabetical order. The KLIC index covers also a frequency count for each type of term and for each term ending.

The INSPEC algorithm was developed in 1975 and it comprised both longest match and iteration approaches. Along with recoding and conditional rules, the word stemming was carried out by three separate algorithms. The first, partly iterative stemmer (Algorithm 0) removed stopwords and the most common endings i.e. plural forms of terms. Terms which did not match the stopword list, were stemmed by Algorithm 1 which was based on the longest match principle. During this stage the majority of suffixes were removed according to context sensitive rules and a minimum stem length. Algorithm 2 modified the word stem using as a basis definite stem length. The principle of multi stage term stemming was later implemented in the experimental online catalogue OKAPI, which is described in Section 3.3.4

The basic principles of the SMART information retrieval system described before were implemented in the Flexible Information Retrieval System for Text **(FIRST)** which was designed in the middle of seventies. The system incorporated a longest match stemming algorithm, which along with a list of 350 stopwords included a suffix dictionary of approximately 250 suffixes.

At the beginning each term was matched against the stoplist and after a successful pass, the number of characters in the word was checked. If the term was less than three characters long, it was added to the stem dictionary. Terms would not be included in the dictionary if:

- a stem with an added suffix matched the term, e.g. ORGAN + ZATION = ORGANIZATION;
- a stem had a double consonant and an added suffix matched the term, e.g. ADMIT + T + ANCE = ADMITTANCE;
- a vowel e was removed from the stem and a suffix which is beginning with vowel, matched the word;
- a stem had ending y which had been changed to i and an added suffix matched the term e.g. HURR -Y + IED = HURRIED (26).

New words were added to the stem dictionary unless they were suffix variations of the existing stem entries. Therefore, the stem dictionary covered words rather than only their actual stems. A unique stem number was associated with each stem entry in order to find a relevant stem through the lookup algorithm by comparing a word with the corresponding stem number.

Minicomputer Operated Retrieval (Partially Heuristic) System **MORPHS** was designed in the seventies to substitute for an existing manual thesaurus based system (27). Stemming was used for indexing terms and for search terms. MORPHS stemming algorithm included both longest match and iteration procedures. The algorithm also involved standardisation of word forms and special role indicators for affix removal.

The word standardisation usually changed plurals to singular forms i.e. COMPUTERS-COMPUTER. However, in some cases to maintain a consistency, terms were standardised to plural forms i.e. words HALF and HALVES were transferred to HALVE, thus avoiding the term INVOLVES being processed as INVOLF.

Role indicators encompassed specific information about the function of terms (28). After a word passed the standardisation procedure, its suffix in most cases would be substituted by the corresponding role indicator, which would define whether the suffix could be removed or not. Role indicators searches based on term roots or the derived forms of terms. For example, the search could be done either using the root COMPUT, or COMPUT (role A) implying COMPUTING, or COMPUT (role D) implying COMPUTED. The implementation of role indicators ensured:

- the reduction of term lengths;
- the bringing together different word forms e.g. DEFLEXION and DEFLECTING would have the resultant root DEFLECT.

The MORPHS stemmer incorporated a comprehensive suffix list which covered a number of exceptions as well as specific chemical suffixes e.g. OSE for term FRUCTOSE. Along with suffix stripping, the algorithm also provided removal of prefixes. A set of rules regarding the stem length prevented removal of invalid prefixes from terms i.e. PRE from PRESSURE or ANTI from ANTIMONY.

3.3.3 Latest developments (1980 -)

In early and mid-eighties construction of stemming algorithms for English language texts and databases reached its highest level of development. As noted by several authors, after this stage any further changes in stemming rules and/or codes will either decrease performance and efficiency of a stemmer or leave it at the same level (29).

Martin **Porter** at the University of Cambridge in 1980 constructed a stemmer which was based on an iterative suffix removal method (30). The algorithm covered a list of term endings and a set of rules including minimum stem length which determined whether the particular suffix could be removed or not.

The principle that vowels and consonants in English terms were arranged in a certain order was built in Porter's stemmer. It meant that all characters could be divided into two groups containing a set of vowels and a set of consonants. Porter denoted vowels by *v* and consonants by *c*. A list of consonants *ccc...>0* was denoted by *C* and a list of vowels *vvv...*>0 was marked as *V*. In that case every term or a part of term can be described in one of the following four forms:

CVCV ... C
CVCV ... V
VCVC ... C
VCVC ... V

These forms can be summarised by the expression:

[C] VCVC ... [V]

where the square brackets means that the presence of C and/or V is optional. Finally, using *m* as a measure for the word or a part of the word, the above mentioned formula can be expressed as follows *[C] $(VC)^m$ [V]*, where the combination *VC* repeats m times. The measure m facilitates to determine either the suffix has to be removed or not. According to Porter's stemmer, the case m=0 included the null word, m=1 covers the first word i.e.

m=0 TR, EE, TREE
m=1 TROUBLE, TREES
m=2 TROUBLES, PRIVATE.

For example, no suffix would be removed from the term TREE, but ending S would be stripped from the word TROUBLES as m>1.

Porter's algorithm also included several conditional rules which were described in the form S1--S2. It meant that if the term ended with the suffix S1 and the stem before S1 satisfied the given condition, S1 was replaced by S2. For example, if the test word was

REPLACEMENT with S1=EMENT and S2=0, than the algorithm would remove suffix EMENT leaving the stem REPLAC. The algorithm included also other conditions i.e.

*s- meant that the stem ends with letter s (the same principle also for other letters);

v- denoted that the stem contains a vowel;

*d- that stem ended with a double consonant (i.e. -TT, -SS);

*o- that stem ended as a string cvc, where the second c was not W, X or Y (i.e. WIL but not TEX).

The stemming algorithm operated in five steps. The suffix dictionary included about 60 suffixes which were grouped in five different categories. Step 1a and 1b dealt with plurals and past participles. The conditional rules for this step were *m>1* and **S or *T* which meant that the stemmer operated with words which ended S or T. Step 1 also checked for double consonants in the term endings, except consonants L, S, Z (condition **d not *L or *S or *Z*). For example, in step 1 the plural form of TIED would be changed to TI (ending -ED will be removed), but in step 1b the past participle of FIZZED would be transferred to FIZ. Step 1c contained condition *(*v*) Y-I* which changes ending Y to I if m>0, e.g. HAPPY - HAPPI.

Step two, three and four stripped suffixes and modified word stems according to the suffix tables, if m>0 (in step 4 m>1). For example, for the term ORGANIZATION, suffix -ATION would be removed and the remaining stem ORGANIZ would be modified to ORGANIZE. Step 5a included condition *(m>1) E --> but m=1 and not *o) E-->* which dealt with terms ending with vowel E. For example, PROBATE would be transferred to PROBAT but RATE would remain without any changes, because m is less than 1. Step 5b removed double consonants in the remaining stem - condition *m>1 and *d and *l*. Complex suffixes were removed in several steps. For example, for the term OSCILLATORS first of all the ending -S would be removed, leaving OSCILLATOR (step 1a). After that ending -OR would be replaced by -E i.e. OSCILLATE (step 2). Step 4 would remove suffix -ATE leaving OSCILL. Finally, step 5b stripped double consonant -L leaving the resultant stem OSCIL.

Tests revealed that after passing the list of stopwords, the majority of terms were stemmed in step 1. It was also observed that despite the algorithm not stemming prefixes, the presence of prefixes decreased the number of errors in stemming process. Overall, the algorithm was simple, it included only a few context sensitive rules and was economical in response time and storage.

Porter's algorithm was implemented in the experimental information retrieval systems CATALOG and INSTRUCT as well as in an online catalogue OKAPI. As noted by Frakes (31), the CATALOG system, which was introduced in 1984, produced good results in information retrieval and provided user friendly front-ends for inexperienced users. The INteractive System for Teaching Retrieval Using Computational Techniques (INSTRUCT) software package was designed in early eighties for students in Library and Information Studies (32). The system covered documents from LISA database for year 1992. The main purpose of the INSTRUCT system was to help the searcher to select the most relevant items which has been identified by the system. Despite Porter's stemmer, which forms the basis of INSTRUCT, having limitations, it produced good results in information retrieval. As the OKAPI catalogue is one of the very few OPAC's which contains the stemming procedure, it will described below in a separate section in more detail.

Porter's stemmer has been redesigned and implemented by B. Frakes and C. Cox in 1986 and changed by C. Fox in 1991. Frakes (33) mainly changed the structure and renamed functions and variables in the algorithms as well as restricted scopes of functions and variables. Fox added ANSI C declarations and carried out complete testing of the whole stemming algorithm.

Another stemming algorithm **MARS** was also designed in the early eighties to provide access to all searchable terms in the database which were morphologically related to a given search term. The system used linguistic analysis and word decomposition techniques based on morphological lexicon. The MARS stemming algorithm checked each term against the list of stopwords and then split terms into prefix, stem, derivational and inflectional suffixes using a morpheme dictionary and morpheme grammar (34).

All word stems were grouped together in a stem file where special pointers provided links between text terms and stems to enable successful retrieval of those terms. The morpheme dictionary covered affixes, inflectional endings and fillers, where the morpheme is the longest possible string which was obtained from all possible derivations. The list of morphemes was presented as a tree. For example, the term 'TRADITIONALLY' would be a derivation of 'TRADITION' and not 'TRAD(E)'. The morpheme dictionary also included two smaller lists:

- 'irregular' stems such as Latin and Greek plurals and irregular verb forms;
- strings which regularly underwent grammatical change i.e. -Y to -IE (ENTRY - ENTRIES).

A pre-processor evaluated whether the string transformations were necessary or not. It was followed by three lists which processed each word using decomposition grammar. A certain stage in a word had to be reached and certain conditions had to be fulfilled to allow the term to be passed to the next stage. All the conditions were listed in the morpheme grammar for the language.

The **Paice/Husk** stemming algorithm was designed and implemented at Lancaster University in the middle eighties (35). The stemmer was iterative and incorporated one table of rules, where each rule specified either deletion or replacement of an ending. Each line in the rule table included a separate stemming rule. For example, the rule "sei3y> { -ies > -y}" meant that if the word ended in "-ies", then the last three letters would be replaced by -y i.e. LORRIES-LORRY (braces cover comments about the action of each rule), and after that the stemmer would be applied again to the stemmed form of a word. There were three basic and two optional components in each rule:

1) an ending which included one or more characters and which are held in reverse order;
2) an optional intact flag "*";
3) a digit which specified the total remove;
4) an optional appended string of one or more characters;
5) a continuation symbol ">" or ".".

For example, the rule "su*2. {-us > - if intact}" meant that if the word had ending -us and if the word was intact, then the last two letters would be removed and the stemming would be terminated i.e. -us would be removed from the word SURPLUS leaving SURPL, but not from EXHAUS (the stem from a word EXHAUSTION). All rules were arranged into separate sections according to the final letter of the suffix and stored in an array, which ensured a quick access to the rule table by looking up the final letter of the current or stemmed word.

Overall, the stemming algorithm included the following steps:

1) selection of relevant action, which meant that the final letter of a word or part of a word was checked. If no section of the rules corresponded to that letter, the process was terminated.
2) testing applicability of the rule. Before applying any of matching rules, a simple acceptability test for each word was carried out. If the final letters of the word did not match the reversed ending in the rule or if the ending matched and the intact flag is set but a word is not intact than go to step 4.

3) application of the rule;
4) look for another rule.

The Paice/Husk stemmer has not been formally evaluated, however it works efficiently and is easy to implement.

3.3.4 Automatic stemming and OPAC's

There are very few experimental online catalogues which incorporate stemming algorithms. One of the main reasons is the extensive amount of information in different subject areas covered by OPACs, which means that the stemmer has to process various types of indexing and search terms. OKAPI is a computerised catalogue which incorporates a stemming operation based on Porter's algorithm (36). OKAPI was developed at the Polytechnic of Central London in 1984-1986. In OKAPI Porter's stemmer is split into two separate algorithms which cover weak and strong stemming. Weak stemming (step 1) removes regular English plurals as well as -ED and -ING endings. After that the double consonant endings are reduced to single. The algorithm involves also a minimum stem length which is at least four characters. Strong stemming (steps 2-5 in Porter) removes suffixes according to the specific conditional rules and suffix tables. In order to achieve better information performance, records retrieved with weak stems are displayed before the records which has been found using strong stems. Overall, it was also observed that weak stemming in online catalogues is more successful and efficient in information retrieval than use of strong stemming.

Computerised Information Transfer in English (CITE) is another online catalogue which was designed in eighties and uses a stemming algorithm to improve access to a collection of monographs at the National Library of Medicine in Maryland (37). The stemmer includes a suffix dictionary which is organised as a pseudo-tree structure and contains terms from Medical Subject Headings (MeSH). There are eight levels in the tree and each level corresponds to the character position of the suffixes. The maximum suffix length is up to eight characters. CITE catalogue uses iterative stemming algorithm which strips endings according to specific combinations of conditions and actions associated with those combinations. For example, combination *A 1&3* means that suffix is detected, but it is a part of a larger suffix (code 1 means- node letter begins a suffix, code 3- letter preceding node letter is part of a suffix). Action *A* determines that the searching should be continued in order to remove the largest possible suffix, but if the remaining root is less than five characters long, than the shorter suffix should be stripped. Condition *D 2&4* states that the character is not a suffix and the stemming process have to be terminated (code 2- node letter does not begin a suffix, code 4- letter

preceding node letter is not a part of a suffix). Action D terminates the stemming process. Design and implementation of the suffix list for CITE catalogue confirmed, that many words in medical English morphologically and syntactically do not differ from general English words, therefore traditional affix removal techniques can be applied also for those terms.

3.4 Construction of non-English language stemming algorithms

All stemmers described in the previous section have been designed for an English language environment. Design and use of various conditional and recoding rules form the most crucial part in the process of construction stemming algorithms. However, those rules can be applied also to other languages, if the semantic importance of the particular language is based on stems rather than on suffixes. Moreover, as noted before, to date any further developments of English stemming algorithms will not significantly increase effectiveness of information retrieval, whereas improvements can be successfully carried out for a number of more complex non-English languages.

Grammatical characteristics and especially morphological complexity determine the adoption of conflation techniques in other languages. For example, it is difficult to apply any English stemming algorithms for German language, as the later consists of many compound terms (38). Descriptions of stemming algorithms which have been designed and implemented for non-English languages e.g. French, Turkish, Slovene, Latin etc. are given below.

3.4.1 Stemmer for French terms

French is an inflective language and has a number of irregularities in morphology and orthography. Even the application of the weakest English stemmer for French language will require a comprehensive suffix dictionary of about 3,000 inflectional suffixes. French terms also have differences between linguistic and semantic meanings. However, the French dictionary is more constant and stable than English or German.

According to Savoy(39), the stemming procedure for French texts consists of two stages:

1) morphological analysis of terms;
2) removal of derivational suffixes according to the grammatical categories.

The morphological analysis requires a dictionary file and a declension file. In dictionary file each term is associated with a certain declension number, gender and grammatical category. For example, the term ROBUSTE (robust) is characterised as adjective, which uses declension number five and the term is masculine in singular form. Declension number five can be found in the declension file which states that ending -s will be removed if the term is in masculine or feminine and in plural form. All declension forms are organised in a truncated digital search tree which determines that the morphological analysis starts from the end of a word. Apart from removing inflectional suffixes, the morphological analysis evaluates the past participle and returns the infinitive form of the verb. For example, the term NEUVES (new) will be processed in following way: first of all three last characters in reverse order i.e. -SEV will be removed from the term (one character at time) and after that character F will be added to the remaining stem NEU forming the stem NEUF.

The derivational process is similar to Porter's iterative affix removal approach. Derivational suffixes can be determined by using a suffix list based on four tables which correspond to four grammatical categories- nouns, adjectives, verbs and adverbs. Each grammatical category covers special rules and several restrictions regarding gender and/or the remaining stem length. When the grammatical category and suffix of the term are determined, it is possible to find a term's stem and the grammatical category of the corresponding stem. For example, for the adjective VOLCANIQUE (volcanic) suffix -IQUE will be removed leaving the stem VOLCAN (volcano) which is a noun.

The French stemmer has been evaluated using three basic tests. The first experiment covered weak stemming which removed inflectional suffixes (plurals, past participle) from 50 test terms. According to results all 50 terms were stemmed correctly. The second test was dealing with prefix removal and the success rate was also high. Finally, the third test which evaluated suffix removal procedure from terms containing only derivational suffixes, also revealed high ratio of correct results. It was also observed that the use of grammatical categories can decrease number of overstemmed terms.

3.4.2 Slovene stemming algorithm

The Slovene language is similar to English in the sense of creating words by adding suffixes to a basic stem. However, Slovene is an inflective language and covers six different cases where nouns, verbal nouns, adjectives, numerals and pronouns can be not only in singular and plural forms, but also in dual form. A stemming algorithm for Slovene language was developed in 1990/91 by Popovic(40). The stemmer is based on

Porter's algorithm and includes a comprehensive list of 5276 Slovene suffixes together with a set of context sensitive rules. Each suffix is associated with a minimum stem length and one of eight codes which determine the definite context sensitive rule that can be applied for the particular term. After the suffix has been removed, three sets of recording rules check the remaining stem to determine whether it should be modified or not. The stemmer is accompanied by an extensive list of stopwords.

Tests and experimental evaluation of the Slovene stemming algorithm were based on the system INSTRUCT which has been mentioned before in section 2.3.9. First of all, the Sign Test which calculates the probabilistic number of relevant documents, was used to determine the difference between the manual word truncation and automatic stemming. Results revealed that the number of relevant documents retrieved by stemmed and truncated searches were almost equal. However, a further statistical analysis and testing based on the two-tailed Sign Test and Kendall's Coefficient of Concordance showed that the performance between conflated and nonconflated text is far greater in favour of stemming. Moreover, the modified version of Porter's algorithm for Slovene language performed considerably better then the original one.

3.4.3 Algorithm for stemming in Latin

Another example of automatic word conflation for non-English language is a stemming algorithm for searching databases of Latin words and texts, which was developed at the University of Sheffield (41) Along with the above described French and Slovene, Latin is inflective language which includes five declensions for nouns and adjectives as well as four conjugations for verbs.

Because of the complexity of Latin language e.g. many words have more than one distinct stem, manual right hand truncation usually produces poor results. Moreover, users have to have sufficient knowledge of Latin morphological structure not to undertruncate or overtruncate a search word(s). To overcome this problem, all Latin words were grouped into two separate classes:

- nouns and adjectives
- verbs

Two distinct sets of rules based on the above mentioned classes of words were implemented into the stemming algorithm. The first set of rules removed suffixes from nouns and adjectives in all five declension forms, whereas the second set was stripping suffixes associated with four conjunctions of verbs. The structured form of the stemming algorithm allowed to avoid of proceeding all classes of words through the same stemming routine which means that words with suffixes relevant to nouns would

not be processed by the stemming rules for verbs. After the stemming procedure was completed, the algorithm generated two stem dictionaries which included all the resultant word stems.

Similarly to the before described stemming algorithms for French and Slovene, the Latin stemmer was written in C programming language and used a number of Porter's data structures and stemming rules. The algorithm incorporated the longest match suffix removal approach leaving the minimum stem length at least three characters. Because of the complexity of language and to distinguish different words with similar roots, the stemmer in many cases intentionally used understemming rather than overstemming.

Analysis and evaluation of the stemming algorithm based on several test collections revealed that automatic word conflation for Latin is more efficient than manual right hand truncation. For example, evaluation of the sample test collection C consisting of 49 complete selected documents, in average reached the success rate of 99%.

3.4.4 Word stemming in other languages

Besides the above mentioned stemming algorithms for French and Slovene languages, attempts to use automatic term conflation approach have been done also for Turkish, Finnish, Russian and Arabic. Because of the complexity of morphological and grammatical structure, none of those languages have tried to test and/or implement any of the before described stemming algorithms for English language.

Some experimental parsing algorithms have been constructed for **Turkish** language. Turkish can be characterised as an agglutinative language where words are formed by combining together root terms and morphemes. Morphemes in agglutinative languages often have no strict boundaries from the root terms. One of the first parsing algorithms for automatic word analysis was designed by Koksal. The algorithm involves the minimum stem length which is presented in a root dictionary. Each term is processed from the left to right and after a root is determined, the remaining part of the term is searched in a suffix morpheme dictionary to identify morphemes. However, this parsing algorithm did not cover any semantic analysis of terms, which is essential for Turkish and other agglutinative languages, as the most suffixes can be linked only to the limited number of roots. Another drawback of the previous parser was explicit use of iterative procedure for suffix derivation, which is not effective for Turkish, as the number of iterations is not high in this language.

Therefore, Solak(42) designed a parser which is mostly based on morphological analysis of the structure of Turkish words. The purpose of this algorithm was to use it as a spelling checker with a further possibility to develop it as a stemming algorithm for information retrieval system. The morphological analysis of Turkish terms comprises three stages:

1) root determination;
2) morphophonemic checks;
3) morphological parsing.

The first stage deals with root determination and it is based on the dictionary of 23,000 words containing a root term and several flags which help to detect certain features of the particular term. Difficulties can arise if the root of the term is deformed, which means that the last consonant in some roots may change to another one. After the root has been found, the rest of the word is considered as suffix(es). At this stage the vowel harmony and usage of passing vowels and consonants are checked according to the specific morphological rules. Finally, the morphological parsing includes two sets of rules which are used for two main root classes. When the root is determined, the class of roots involves the appropriate set of rules. The whole process of morphological analysis is carried out by the lexical analyser (LEX) and Yet Another Compiler-Compiler (YACC) which generates a parser for examining input terms and grouping them into syntactical clusters.

Finnish is another example of an agglutinative language, where suffixes often are added to the root to modify and/or extend the meaning of a term. One of the experimental algorithms for removing suffixes from the end of the term was produced by Brodda and Karlsson. No stem dictionary is involved in this approach and after the suffix removal from a term, the remaining part is assumed to be the root.

Suffix removal algorithm for **Greek** is one of the first attempts to construct stemming algorithm for language which is based on non-Latin character set (43) The grammatical structure of Greek language covers a rich inflectional system which includes 41 forms of suffixes. Similarly to the above mentioned Slovene, Latin languages, nouns in Greek have four different cases and the declension is carried out according to 41 categories of nouns e.g. 14 for the masculine, 14 for the feminine and 13 for neutrer.

The iterative algorithm was based on two stage suffix removal procedure:

- analysis and removal of inflectional suffixes;
- removal of derivational suffixes which correspond to their grammatical categories.

All suffixes were grouped into three different tables according to three classes of words:

- nouns;
- adjectives;
- verbs.

The total 65 types of different suffixes were included into the final version of the algorithm. Conditional rules for suffix removal covered several restrictions on the suffix length depending on a resulting stem and the minimum length of a stem was no less than three characters. The algorithm was supplied by a stoplist which removes stopwords e.g. definite and indefinite articles, conjunctions, prepositions, pronouns, before the actual suffix removal procedure was started.

Preliminary evaluation based on two small test collections covering documents in medicine and computing as well as analysis of user enquiries revealed that the majority of errors were caused by understemming. Although the algorithm have not been implemented into any of Greek databases and/or tested using large document collections, the initial evaluation showed that in 90% of all cases the Greek stemmer produced correct stems.

Sagvall designed a morphological analyser for **Russian** language which checks an initial substring of the word in a root dictionary. After that possible suffixes, which can follow the root of a term, are determined according to grammatical rules and categories.

An experimental information retrieval system for a collection of 23,000 documents in **Arabic** language was introduced by Al-Kharashi(44). The main purpose of this system was to compare which of the following three choices is the best for automatic indexing and information retrieval:

- use of the complete term;
- the stem of a term;
- the root of a term.

The Arabic language is based on a root and pattern structure which means that majority of terms are derived from a short list of roots. Root is the part of a word without prefixes, affixes and endings. Stem is the part of a word without the ending. The experiment involved manually developed word-root-stem dictionary which ensured to identify the stem or the root for each term as well as exclude stopwords. For 355 records, the dictionary included 1,126 terms, 725 stems and 526 roots. A corresponding stem and root structure was determined for each keyword. Similarity

measurements based on cosine, Dice and Jaccard binary coefficients were involved, to evaluate recall and precision ratio for information retrieval using a complete term or term's root and/or stem. Results confirmed that the stem and root retrieval methods are efficient and can retrieve more relevant documents than the one based on full word. Statistical tests i.e. the Sign test also revealed good results in favour of stem and root retrieval approaches. It was also mentioned that in future the word-stem-root dictionary should be replaced by a morphological algorithm for more efficient word conflation in IR system.

Overall, the complexity of design and implementation of non-English stemming algorithms to a certain extend depends on the morphological structure of a particular language. English affix removal stemmers can be successfully applied for languages which use single, separate terms to form the text and where each term has strictly determined root and suffix boundaries i.e. French, Slovene, Latin. Moreover, the Slovene and Latin stemming algorithm achieved even better information retrieval results than the similar algorithm for English language environment. It is more difficult is to design and implement stemmers for agglutinative languages i.e. Turkish, Finnish as well for languages which use a non-Latin alphabet i.e. Arabic, Chinese, Russian so therefore, at present very few automatic word conflation approaches have been introduced for those languages.

3.5 Evaluation of stemming algorithms

Sections 3.3 and 3.4 revealed that affix removal is one of the most often used stemming approaches in information retrieval systems. The majority of stemmers described in the section 3.3 incorporate longest match approach which in some cases is coupled by the iteration method. However, many of those stemmers are actually modified versions of previously designed algorithms i.e. Dawson's suffix removal algorithm is based on the Lovin's longest match stemmer. The iterative approach has been successfully applied for non-English language stemming algorithms. All the affix removal stemming algorithms mentioned before in section 3.3 are summarised in table 3.4.

Table 3.4 Variety of affix removal stemmers

Stemming algorithm	Longest match	Iteration	Other
LOVINS (INTREX)	x		
DAWSON	x		
INSPEC	x	x	
RADCOL	x		
SMART	x		
FIRST	x		
MORPHS	x	x	
MARS			x
PORTER		x	
INSTRUCT		x	
PAICE/HUSK		x	

To determine the difference and efficiency between the variety of term conflation algorithms, several evaluation studies and tests have been carried out for stemmers in information retrieval systems. One of the first experimental studies was done by Salton(45) who compared retrieval results based on iterative longest match method using fully stemmed words and terms with suffix 's' removed. Three document collections i.e. IRE-3 covering 780 computer science abstracts and 34 user queries, ADI consisting of 82 documents and 35 queries and Cranfield-1 covering 200 aerodynamics abstracts and 42 queries, have been used for the evaluation study. Calculations based on 14 dependent variables for each query i.e. rank recall, log precision, normalised recall, normalised precision and precision for ten recall levels were used to compare both the above mentioned stemming methods. Related group t tests and sign tests were used to analyse the calculated data.

For the IRE-3 collection, there were 272 cases, which favoured full stemming, in 132 cases the preferrence was given to suffix 's' stemming and in 72 cases neither one nor another method was preferred. The effect size for the IRE-3 collection was .175. For the ADI collection, in 254 cases the preferrence was given to full stemming, 107 cases favoured suffix 's' stemming and 129 cases did not favoured to any of both stemming approaches. The effect size for ADI collection was .20. For the Cranfield-1 collection, full stemming method was chosen in majority cases and the effect size for this collection was .235. Results revealed that stemming may significantly affect retrieval performance depending on the type of vocabulary i.e. the Cranfield collection is more technical and homogenous than ADI and IRE-3, therefore the results were better for that collection.

Van Rijsbergen (46) evaluated Porter's stemmer against the Dawson's longest match algorithm using the Cranfield-1 collection. The results based on ten paired recall - precision levels revealed that Porter's stemmer was slightly better than the Dawson's stemmer. No statistical results have been reported on this evaluation study.

Another evaluation was carried out by Lennon(47) who analysed the retrieval effectiveness and inverted file compression of several stemming algorithms i.e. RADCOL, Hafer & Weiss, Lovins, INSPEC, Porter. Tests were based on Cranfield-1400 document collection which covered 1,396 documents and 225 user queries. For each stemmer, words from document titles and user queries were stemmed and stems were replaced by stem numbers for easier processing. The effectiveness of document search was defined by measure E, which can be calculated using the following formula:

$$E = 1 - \frac{(1 + b^2)\,PR}{b^2P + R}$$

where P=precision, R=recall and b measures the relative importance added to recall and precision by the user in a case if relevant documents are retrieved.

The evaluation study covered analysis of the relative effectiveness of stemming vs nonstemming. All stemmers except Hafer and Weiss successor variety algorithm performed better information retrieval results than unstemmed terms. The relative performance of various stemmers was also evaluated and experiments revealed that there is a little difference using one or another stemming algorithm in terms of retrieval effectiveness.

Walker and Jones(48) analysed Porter's stemmer using an online book catalogue RCL. It was observed that stemming can considerably increase recall. Experiments also revealed that weak stemming does not decrease precision, which strong stemming does. Weak stemming also performed better document retrieval results for OPACs e.g. OKAPI than strong stemming. Therefore, it was recommended to use weak stemming first and to reserve strong stemming for those cases, when no documents have been retrieved by the weak stemming approach.

Three different stemmers i.e SMART based on Lovins' longest match algorithm, Porter's iterative stemmer and 'S' algorithm, which conflates singular and plural term forms, have been evaluated by Harman(49). Experiments were based on IRX system which covered Cranfield-1400, Medlars and CACM document collections. Recall and precision ratio were analysed for each group of documents. Tests with the Cranfield collection revealed that stemming did not increase significantly performance. The best

results in terms of relevance has been retrieved from CACM collection. The experimental study for evaluation of stemmers involved such methods as:

- reweighting of term expansions;
- selective stemming based on query length;
- selective stemming based on term importance.

Tests revealed that Porter's stemmer produced more term variants for a given word therefore, expanding the initial query. Lovins algorithm retrieved a larger number of term variants after matching the given root of a term. It was observed that after the stemming process, nonrelevant documents often received higher ranking scores than relevant items. Overall, the study confirmed that all three stemmers did not evidently improved information performance.

Frakes(50) carried on experimental study covering the evaluation of right hand truncation vs automatic stemming. Results showed that there is no evidential difference between right hand truncation and involvement of stemming procedures. However, as it was mentioned by Harman, stemmed query terms are more convenient for end users than use of truncation and wildcard characters.

Summarising the above mentioned experimental studies, the evaluation of stemming procedure in information retrieval systems can be based on:

- comparison of different stemming algorithms i.e. Harman's study;
- analysis of use of full word vs stemmed term i.e. Lennon's test;
- comparison of truncation vs stemming i.e. Frakes study.

3.6 Conclusion

Several of recent information retrieval systems i.e. CITE, MARS, INSTRUCT incorporate one or another of stemming methods e.g. stemming based on morphological analysis of terms, stemming covering suffix dictionaries etc. According to Willett(51), automatic term conflation in information retrieval systems:

- may reduce the number of distinct terms and therefore the size of dictionary;

- may increase information retrieval effectiveness and particularly the recall ratio as the conflation procedure can easily determine semantically similar terms.

Evaluation studies analysing stemming techniques and stemmers confirmed that automatic word conflation can improve information retrieval performance. There is also no evidence that stemming can degrade retrieval effectiveness. Tests also determined that stemming results often depends on the type of vocabulary, involved in a information retrieval process. It was also observed that stemming increases recall ratio but at the cost of decreasing precision.

Overall, at present most research and experiments regarding stemming procedure and stemming algorithms has been carried out in English language environment and based on English materials. Analysis of computerised term conflation in non-English languages (section 3.4) reflected, that very few stemming algorithms have been developed or adopted for information retrieval purposes in other languages than English. The complexity of language structure is often the main reason for such restrictions. To date no stemming algorithms exist also for Latvian. In order to determine the appropriateness of implementation of automatic term conflation for information retrieval in Latvian, it is necessary to characterise the morphological structure of Latvian language which is described in Chapter 4.

References

1. **Frakes W.B.** Stemming algorithms. In: William B. Frakes & Ricardo Baeza-Yates, eds. *Information retrieval: data structures and algorithms*, 1992, p. 131.

2. **Willett, Peter.** Autoamtic indexing of documents and queries. In: P. Willett ed. *Document retrieval systems*, 1988, p.8.

3. **Walker, Stephen & Richard M. Jones.** *Improving subject retrieval in online catalogues. 1.: Stemming, automatic spelling correction and cross-reference tables.* London: The Polytechnic of Central London, 1987, p.21.

4. **Lovins, Juliet.** Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 1968, **11**(1), 22.

5. **Hafer, Margaret A. & Stephen F. Weiss.** Word segmentation by letter successor varieties. *Information Storage and Retrieval*, 1974, **10**, 372.

6. *Ibid.*, pp. 373-375.

7. *Ibid.*, p. 375.

8. **Frakes**, ref. 1, p. 137.

9. **Rasmussen, Edie**. Clustering algorithms. In: William B. Frakes & Ricardo Baeza-Yates, eds. *Information retrieval: data structures and algorithms,* 1992, pp. 430-431.

10. **Adamson, G. &J. Boreham.** The use of an association measure based on character structure to identify semantically related pairs of words and document titles. *Information Storage and Retrieval*, 1974, **10**, 253-260.

11. **Lennon, Martin, David S. Pierce, Brian D. Tarry & Peter Willett.** An evaluation of some conflation algorithms for information retrieval. *Journal of Information Science*, 1981, **3**, 179-180.

12. **Lovins**, ref. 4, pp. 29-30.

13. **Frakes,** ref. 1, pp. 143-147.

14. **Overhage, Carl F.J. & J. Francis Reintjes**. Project Intrex: a general review. *Information Storage and Retrieval*, 1974, **10**, 157-188.

15. **Lovins**, ref. 4, pp. 24-25.

16. *Ibid.*

17. **Salton, Gerard & M. E. Lesk.** The SMART automatic document retrieval system. *Communications of the ACM,* 1965, **8** (6), 391-398.

18. **Lovins**, ref. 4, p. 24.

19. **Overhage & Reintjes**, ref. 14, p. 174.

20. *Ibid.*

21. **Brzozowski, J.P.** MASQUERADE: searching the full text of abstracts using automatic indexing. *Journal of Information Science*, 1983, **6**, 69.

22. **Salton, G. & M.J. McGill.** The SMART and SIRE experimental retrieval systems. In: Peter Willett, ed. *Document retrieval systems,* 1988, pp. 192-229.

23. **Lennon** et al., ref. 11, p. 178.

24. **Walker & Jones,** ref. 3, p. 28.

25. **Popovic, Mirko**. *Implementation of a Slovene language based free-text retrieval system*, 1991, pp. 42-43.

26. **Dattola, Robert T.** FIRST: Flexible information retrieval system for text. *Journal of the American Society for Information Science,* 1979, **30**(1), 9-14.

27. **Jones, Kevin P. & Colin L.M. Bell.** The automatic extraction of words from texts especially for input into information retrieval systems based on inverted files. In: C.J. van Rijsbergen, ed. *Research and development in information retrieval,* 1984, p. 409.

28. **Bell, Colin L.M. & Kevin P. Jones.** A minicomputer retrieval system with automatic root finding and roling facilities. *Program,* 1976, **10**(1), 16-21.

29. **Harman, Donna.** How effective is suffixing? *Journal of the American Society for Information Science,* 1991, **42**(1), 7-15.

30. **Porter, Martin.** An algorithm for suffix stripping. *Program,* 1980, **14** (3), 130-137

31. **Frakes, W.B.** Term conflation for information retrieval. In: C.J. van Rijsbergen, ed. *Research and development in information retrieval*, 1984, pp. 383-389.

32. **Hendry, Ian G., Peter Willett & Frances E. Wood.** INSTRUCT: a teaching package for experimental methods in information retrieval. Part I. The user's view. *Program,* 1986, **20**(3), 245-263.

33. **Frakes,** ref. 1, pp. 151-160.

34. **Niedermair, G.Th. G. Thurmair & I. Büttel.** MARS: a retrieval tool on the basis of morphological analysis. In: C.J. van Rijsbergen, ed. *Research and development in information retrieval,* 1984, pp. 375-378.

35. **Paice, Chris D.** Another stemmer. *ACM SIGIR Forum,* 1990, 24(3), 56-61.

36. **Walker, Stephen.** OKAPI: evaluating and enhancing an experimental online catalog. *Library Trends,* 1987, **35**(4), 631-645.

37. **Ulmschneider, John E. & Tamas Doszkocs.** A practical stemming algorithm for online search assistance. *Online Review,* 1983, **7**(4), 305-314.

38. **Fuhr, Norbert von.** Zur Überwindung der Diskrepanz zwischen Retrievalforschung und -praxis [Bridging the gap between retrieval research and practice]. *Nachrichten für Dokumentation,* 1990, **41**(1), 3-7.

39. **Savoy, Jacques.** Stemming of French words based on grammatical categories. *Journal of the American Society for Information Science,* 1993, **44**(1), 2-7.

40. **Popovic, Mirko & Peter Willett.** The effectiveness of stemming for natural-language access to Slovene textual data. *Journal of the American Society for Information Science,* 1992, **43**(5), 384-390.

41. **Schinke, Robyn et al.** A stemming algorithm for Latin text databases. *Journal of Documentation,* 1996, **52**(2), 172-187.

42. **Solak, Aysin & Kemal Oflazer.** Design and implementation of a spelling checker for Turkish. *Literary and Linguistic Computing,* 1993, **8**(3), 113-124.

43. **Kalamboukis, T. Z.** Suffix stripping with modern Greek. *Program,* 1995, **29**(3), 313-321.

44. **Al-Kharashi, Ibrahim A. & Martha W. Evens.** Comparing words, stems and roots as index terms in an Arabic information retrieval system. *Journal of the American Society for Information Information Science,* 1994, **45**(8), 548-560.

45. **Salton, Gerard.** Automatic information organization and retrieval, 1968, pp. 280-349.

46. **Frakes,** ref. 1, p. 144.

47. **Lennon** et al., ref. 11, pp. 180-183.

48. **Walker & Jones**, ref. 3, 25.

49. **Harman,** ref. 29.

50. **Frakes**, ref. 1, 146-147.

51. **Willett,** ref. 2, 7-9.

Chapter 4
STRUCTURE OF LATVIAN LANGUAGE

4.1 Introduction

Latvian along with Lithuanian and the extinct Old Prussian and Curonian languages forms a separate Baltic branch of the Indo-European language family (1). Because of a few shared features, some linguists have placed Baltic languages in the same group as Slavic languages (2). Nevertheless, over the centuries Latvian has been much influenced by German, Russian and Scandinavian languages.

The structure of Latvian language is similar to Lithuanian; however, Latvian in several aspects is more modern than Lithuanian. For example, the reduction of vowels in final syllables has progressed further in Latvian than in Lithuanian. Also by the influence from Scandinavian languages, word accent in Latvian is fixed on the first syllable (3).

This chapter comprises a brief introduction to Latvian character set, which is followed by the detailed description of Latvian morphology as the concept of word formation is the basis for automatic word stemming. Finally, a comparative analysis of Latvian and English language structure, including common and distinguished features of both languages, is presented.

4.2 Latvian alphabet and pronunciation

Latvian, similar to English and other European languages uses the Latin alphabet, but there are several diacritic marks for some letters. Contemporary Latvian language consists of 33 characters: A, Ā, B, C, Č, D, E, Ē, F, G, Ģ, H, I, Ī, J, K, Ķ, L, Ļ, M, N, Ņ, O, P, R, S, Š, T, U, Ū, V, Z, Ž. Character Y is optional and it is used in Latgalian (Latvian dialect) and Livs languages (predecessor of the Latvian language) i.e. puyrs - puhr. Latvian texts can also contain the so called softened r (ŗ) i.e. kaŗš - war (4).

All Latvian characters can be divided into two basic groups - vowels and consonants. There are four short vowels i.e. A, E, I, U and four long vowels i.e. Ā, Ē, Ī, Ū, in Latvian language.

- vowel ā (e.g. māja - a house) is pronounced similar to the *a* of English word father;

- vowel ē represents two different phonemes: "closed ē" (e.g. vējš - wind), which is pronounced similar to the first "e" in the English word there
  and
  "open ē" (e.g. lēmums - a decision), which is pronounced as "a" in the English bad;
- vowel ī (e.g. pīle - a duck) is similar to the English "ea" i.e. sea, but a little more closed;
- vowel ū (e.g. lūsis - a lynx) is pronounced as the English "oo" i.e. room, but more closed.

Vowel O can be either short or long, depending on the specific word. For example, in the word ozols (an oak) both o are "closed" and their pronounciation is similar to English oa in the word cloakroom. Word foto (a photo) contains "long or open o" which is pronounced as o in the English word got.

Latvian language contains the following consonants: B, C, Č, D, F, G, Ģ, H, J, K, Ķ, L, Ļ, M, N, Ņ, P, R, S, Š, T, V, Z, Ž. Several consonants are pronounced as they are spelled. Pronounciation of those consonants, which differ from English, is given below:

- consonant c (e.g. cilvēks - a person) is pronounced like the ts of the English word slots;
- č (e.g. čūska - a snake) is pronounced like the ch of English child;
- g (e.g. gabals - a piece) is pronounced like the g of the English word goose;
- there is no definite English equivalent for consonant ģ (e.g. ģitāra - a guitar). However, English speakers can perceive it as an odd kind of d (e.g. similar to de in the word Dewey);
- j (e.g. jērs - a lamb) is pronounced like the y of the English word yellow;
- consonant ķ (e.g. ķengurs - a kangaroo) has no equivalent letter(s) in English, but it can be likely perceived as an odd kind of t (e.g. similar to tu in the word tulip);
- consonant ļ (e.g. ļaunums - evil) is pronounced as the combination of letters l+i in the English word million;
- ņ (e.g. riņķis - a circle) is pronounced as the combination of letters n+i in the English onion;
- consonant š (e.g. šautene - a gun) is pronounced like the sh of the English shelf;
- consonant ž (e.g. žurnāls - a magazine) is pronounced like the s of the English word measure.

For better description of morphemic alternations caused by the Latvian inflectional morphology, all consonants can be further classified into the following groups:

- sonorants i.e. j, l, ļ, m ,n, ņ, r;
- obstruents, which can be divided into
  - voiced consonants i.e. b, d, g, ģ, v, z, ž and corresponding
  - voiceless consonants i.e. p, t, k, ķ, f, s, š, c, č, h (5).

The following sections will highlight that the position of sonorants and obstruents at the end of the stem is often important in the change of stems and suffixes during the inflection of words. For example, according to the grammatical rule, voiced consonants before voiceless consonants in pronounciation become voiceless i.e. logs (a window) sounds like loks (an arc) (6).

4.3 Morphological system in Latvian language

4.3.1 The concept of morphology and morphemes

As noted by Anderson "Morphology is the study of the structure of words, and of the ways in which their structure reflects their relation to other words - both within some larger construction such as sentence and across the total vocabulary of the language" (7). Each word consists of morphemes. The morpheme is the smallest unit of a language that cannot be segmented further and which contains a constant meaning (8). Root, suffix, prefix and ending or flection are the basic types of morphemes.

- root is a morpheme, which contains the meaning of a word;
- ending or flection is a morpheme, which is located at the end of a word and which is flective according to the case and declension of a word;
- suffix is a morpheme, which is located between the root and the flection;
- prefix is a morpheme, which is located before the root.

Grammatical concepts, i.e. word-final and stem, are also used for the description of a word. The word-final is a part of a word which contains the last suffix together with the flection (9). According to different schools of linguists the stem can be definied either as

- a part of a word without the flection (10)

or

- a part of a word without the flection and the last suffix (11).

The latter definition of the stem will be used for this dissertation as it has been approved and used by the majority of computing linguistics experts in Latvia (12).

Figure 4.1 illustrates the basic order of morphemes i.e. roots, suffixes, prefixes and flections in the words of the Latvian language (13).

R	(DZĪV - root for the word *life*)
R + F	(DZĪV + E - life)
R + S + F	(DZĪV + NIEK + S - animal)
R + S_1 + S_2 + F	(DZĪV + ĪG + UM + S - vitality)
P + R + S + F	(AT + DZĪV + INĀ + T - to revive)
P + R + S_1 + S_2 + F	(PĀR + DZĪV + OJ + UM + S - experience)
P_1 + P_2 + R + S + F	(NE + AP + DZÈV + OT + S - uninhabited)
R_1 + R_2 + F	(DZĪV + SUDRAB + S - quick silver)
R_2 + R_1 + F	(PUS + DZĪV + S - half dead)

R = root, F = flection, S = suffix, P = prefix.

Figure 4.1 Location of morphemes in Latvian words.

In addition to the above mentioned, morphemes can be also classified into the following groups:

- lexical and grammatical morphemes,
- free and bound morphemes.

Lexical morphemes, i.e. Nouns and verbs, have a meaning in and of themselves, while grammatical morphemes (e.g. articles, conjunctions) have no such meaning and they express relationship between lexical morphemes. Free morphemes can stand alone as words and they may be either lexical or grammatical, while bound morphemes cannot form a separate word, e.g. the plural ending -s in a word dogs (14).

In Latvian language there are either bound morphemes (e.g. in the word suns- a dog sun is the root and s is the ending) or grammatical morphemes (e.g. the conjunction bet- but). Practically, there are no lexical and/or free morphemes in Latvian grammar. In order to characterise the morphological complexity of the Latvian language, the following subsection will introduce the basic categories of words and inflections associated with those categories.

4.3.2 The inflectional system of Latvian words

Latvian is an inflective language and has ten general categories or classes of words:

1. nouns (e.g. koks - a tree, māja - a house);

2. pronouns (e.g. tu - you, mans - my);
3. verbs (e.g. skriet - to run, lasīt - to read, to pick);
4. adjectives (e.g. saulains - sunny, dzeltens - yellow);
5. numerals (e.g. pieci - five, simts - a hundred);
6. adverbs (e.g. pērn - last year, gandrīz - nearly);
7. prepositions (e.g. ar - with, pār - across);
8. conjunctions (e.g. tomēr - however, ja - if);
9. particles (e.g. vienīgi - only, vis - most);
10. interjections (e.g. skat! - look, sveiks! - cheers).

All the above listed word classes can be divided into inflectional and non-inflectional categories. Nouns, pronouns, adjectives, verbs and numerals form the inflectional category of words, while adverbs, prepositions, conjunctions, particles and interjections belong to the non-inflectional category (15). Non-inflectional categories or words containing only grammatical morphemes are also called non-content bearing words, which can be considered to be included in a stopword list for an information retrieval system. Structure and characteristics of nouns, adjectives, pronouns and verbs are described in separate subsections, as these are the core elements of the Latvian inflectional system.

Nouns

There are six declensions and six basic cases, i.e. nominative, genitive, dative, accusative, instrumental and locative for nouns in the Latvian grammar. Vocative case is optional and usually it has the same ending as the nominative or some special ending, or no ending e.g. akmen! (stone). Nouns in the I, II and III declension are in the masculine gender, but the majority of nouns in the IV, V and VI declension belong to the feminine gender (one of the exceptions is puika - a boy, which belongs to the IV declension, but is in masculine gender). International words with vocalic endings i.e. ā, ē, e, o, u, ū form a group of nondeclinable nouns, e.g. radio, kanoe (canoe). Table 4.1 reflects inflectional endings of nouns. If the declension and the case contain more than one ending, variants are respectively indicated in brackets. The asterisk (*) specifies palatalisation of the final root (stem) consonant.

Table 4.1 Inflective endings of Latvian nouns

Declensions		I	II	III	IV	V	VI
Number	Case/Gender	Masculine			Feminine		
S	Nominative	-s (-š)	-is (s)	-us	-a	-e	-s
I	Genitive	-a	-a* (-s)	-us	-as	-es	-s
N	Dative	-am	-im	-um	-ai	-ei	-ij
G	Accusative	-u	-i	-u	-u	-i	-i
U	Instrumental	-u	-i	-u	-u	-i	-i
L	Locative	-ā	-ī	-ū	-ā	-ē	-ī
A							
R							
P	Nominative	-i	-i*	-i	-as	-es	-is
L	Genitive	-u	-u*	-u	-u	-u*(-u)	-u*(-u)
U	Dative	-iem	-iem*	-iem	-ām	-ēm	-īm
R	Accusative	-us	-us*	-us	-as	-es	is
A	Instrumental	-iem	-iem*	-iem	-ām	-ēm	-īm
L	Locative	-os	-os*	-os	-ās	-ēs	-īs

Palatalisation of the final stem consonant changes the root of a noun and it is one of the basic criteria, which determine the complexity of inflective Latvian grammar. As seem in Table 4.1 nouns in the II, V and VI declension have the palatalisation of consonants, which is carried out in the following way:

- consonant b is changed to bj (b+j) e.g. gulbis (a swan) - gulbji (swans);
- c to č e.g. līcis (a bay) - līči (bays);
- d to ž e.g. lode (a bullet) - ložu (bullets');
- l to ļ e.g. brālis (a brother) - brāļi (brothers);
- m to mj e.g. kurmis (a mole) - kurmji (moles);
- n to ņ e.g. suns (a dog) - suņi (dogs);
- p to pj e.g. upe (a river) - upju (rivers);
- s to š e.g. lasis (a salmon) - laši (salmons);
- t to š e.g. nakts (a night) - nakšu (nights);
- v to vj e.g. cirvis (an axe) - cirvji (axes);
- z to ž e.g. birzs (a birch grove) - biržu (birch groves').

In addition, the combination of consonants s+t (st) changes to š e.g. pāksts (a pod) - pākšu (pods'), sn changes to šņ e.g. krāsns (a stove) - krāšņu (stoves') and zn to

žņ e.g. zvaigzne (a star) - zvaigžņu (stars). Regressive assimilation occurs when consonant l changes to ļ, n to ņ and z to ž, and these are preceded by l, s or z, which become respectively ļ, š and ž, i.e. ll changes to ļļ, sl to šļ, zl to žļ. For example, the noun lelle (a doll) in singular nominative case changes to leļļu (dolls) in plural genitive, or the singular nominative case of the word zizlis (a staff) changes to zižļi (staves) in plural nominative case. Palatalisation of the final stem consonants g and k to dz and c, respectively, applies for nouns in a diminutive form, i.e. logs (a window) lodziņš (a little window), or vāks (a lid) vāciņš (a little lid). In modern Latvian grammar the former r to ŗ palatalisation no longer applies (16).

There is a special group of nouns so called 'deverbal' reflexive nouns, which are formed with the suffix -šan e.g. vēlē the verb (wishes) + šan + ās = vēlēšanās (a wish). These nouns are always feminine and have only nominative, genitive, accusative and instrumental case. Table 4.2 shows the inflectional endings of deverbal reflexive nouns.

Table 4.2 Example of deverbal reflexive nouns

Case / Number	SINGULAR	PLURAL
Nominative	-ās (vēlēšanās a wish)	-ās (vēlēšanās)
Genitive	-ās (vēlēšanās)	-os (vēlēšanos)
Accusative	-os (vēlēšanos)	-ās (vēlēšanās)
Instrumental	-os (ar vēlēšanos)	-

Nouns are usually derived with word finals (suffixes + endings) from:

1) other nouns e.g. skola (a school) + otājs = skolotājs (a teacher);
2) adjectives e.g. salds (sweet) - saldumi (sweets);
3) verbs e.g. klausīties (to listen) - klausīšanās (listening);
4) numerals e.g. viens (one) - vienatne (solitude);
5) adverbs e.g. tagad (now) - tagadne (the present).

Nouns can also be derived using various prefixes i.e. ār- (ārzemes - abroad), zem- (zemūdene - submarine), pie- (piepilsēta - suburbs). Because a nominal stem may have a palatalisation in the final consonant, both the nominative and genitive stems are used for derivation.

Pronouns

Pronouns, similar to nouns, are a category of words, which is characterised by a declension, case, number and gender. Pronouns can be divided into the following groups:

- personal pronouns i.e. tu (you), viņš (he);
- possesive pronouns i.e. mans (my), tavējs (yours);
- the reflexive pronoun sevis (myself);
- demonstrative pronouns i.e. šis (this), tas (that);
- relative pronouns i.e. kāds (somebody);
- indefinite pronouns i.e. daži (some), cits (other).

There are also interrogative (e.g. kurš - who), general (e.g. ikviens - eveybody), definite (e.g. pats - myself) and negative (neviens - nobody) pronouns. Personal pronouns in the first and second persons both in singular and plural have irregular flectional form, which are presented in Table 4.3. Third person pronouns use the same flectional pattern as nouns of the I and IV declension.

Table 4.3 Example of flection for irregular personal pronouns

SINGULAR			PLURAL	
Case	1st person	2nd person	1st person	2 nd person
Nom.	es (I)	tu (you)	mēs	jūs
Gen.	manis	tevis	mūsu	jūsu
Dat.	man	tev	mums	jums
Acc.	mani	tevi	mūs	jūs
Instr.	mani	tevi	mums	jums
Loc.	manī	tevī	mūsos	jūsos

All other types of pronouns follow the same flectional patterns as I and IV declension nouns. A specific morphological case covers demonstrative and relative pronouns, which synchronically have the roots consisting of one letter, i.e. š, t and k, and from which all other forms are derived by suffixation, e.g. adjectives with the suffix -ād (šāds - such).

Adjectives

Latvian language uses a definite and an indefinite form of adjectives. Indefinite adjectives correspond to the same form of flection as masculine nouns of the I

declension and feminine nouns of the IV declension. Definite adjectives are derived from indefinite endings using specific rules, which comprise lengthening the indefinite endings, i.e. a becomes ā (respectively, in the plural -as changes to -ās), u changes to o, and i to ie. For example, the definite form for an indefinite adjective and a noun balta roze (a white rose) will be baltā roze (the white rose). In definite adjectives syllabic and consonantal endings are preceded by -aj e.g. skaistajā parkā (in the beautiful park). The possible endings for definite adjectives are shown in Table 4.4.

Table 4.4 Endings and flectional pattern for definite adjectives

	Masculine		Feminine	
Case	Singular	Plural	Singular	Plural
Nom.	-ais	-ie	-ā	ās
Gen.	-ā	-o	-ās	-o
Dat.	-ajam	-ajiem	-ajai	ajām
Acc.	-o	-os	-o	-ās
Instr.	-o	-ajiem	-o	-ajām
Loc.	-ajā	-ajos	-ajā	-ajās

Some compound nouns in the genitive case may adopt an adjectival function, i.e. šaursliežu (narrow-track), which consists from šaurs (narrow) and sliede (a track). Morphologically they are not declinable and they do not contain comparative degrees.

The comparative degree of adjectives is formed by the suffix -āk, i.e. bāls (pale) - bālāks (paler). The superlative degree is composed by the suffix -āk, the definite ending and the preposition vis-, e.g. visskaistākais (the most beautiful). Similar to nouns, adjectives may be derived from nouns, adjectives, numerals, pronouns, verbs and adverbs using suffixes (i.e. -ād, -ain, -ēj, -en, -gan, -īg, -iņ, -isk) and appropriate endings, e.g. vējš (wind) - vējains (windy), liels (big) - lielisks (splendid), runāt (to talk) - runātīgs (talkative).

Verbs

Latvian verbs are divided into two broad groups: primary and secondary verbs. Conjugation is another criterion, which characterises a verb. All verbs can be grouped into three conjugations. Primary verbs belong to the I conjugation and their infinitive form consists of a root and a flection (R+F) e.g. rakt (to dig), celt (to build). Secondary verbs belong either to the II or to the III conjugation. The II conjugation verbs have the infinitive form R+Td+F, where Td is type designator. The type

designator is always a long vowel, i.e. ā, ē, ī, o, ū. The following verbs are examples of secondary verbs in the II conjugation: domāt (to think), medīt (to hunt), balsot (to vote), dabūt (to get). The III conjugation contains secondary verbs, which in the present tense have the form R + F e.g. zināt - to know (in the present tense: es zinu - I know).

Verbs can also have a reflexive form, which is indicated by the lengthening of the vowel + s. Vowel lengthening is carried out in the following way: a becomes ā, i changes to ie, and u to o, e.g. smēja - smējās (was laughing). The infinitive form of reflexive verbs always includes the morphemes ie and s, e.g. sacensties (to compete).

Derivation by suffixation is based on the following stems:

- the past tense stem;
- the infinitive stem;
- the participial stem.

There are several rules both for primary and secondary verbs for deriving the infinitive form from the past tense stem and vice versa. For primary verbs, past tense roots with the stem final consonants b, g, k, l, m, p, r, s will have the same stem as in infinitive form, e.g. lika - likt (to put). Another rule for primary verbs determines that morphemes -āj, -ēj, -ij in the past tense change to -ā, -ē, -ī + t in the infinitive form e.g. lija - līt (to rain). Respectively, -uv changes to -ū+t, e.g. kļuvu - kļūt (to become) and -āv to -au+t e.g. šāva - šaut (to shoot). Primary verbs also have palatalisation of final stem consonants, i.e. c, dz change to k, g + t and t, d to s + t, e.g. sauca - saukt (to call), lūdza - lūgt (to ask). For secondary verbs, the past stem is derived from the infinitive stem by deleting the infinitive ending -t and adding the concatenator -j + F (corresponding flection) e.g. sekot - sekoja (to follow - followed).

Participles are usually derived from:

- the present tense stem by adding suffixes -am, -ām, -oš, -ot e.g. zinu - zināms (I know - known);
- the infinitive stem by the suffix -dam e.g. skriet - skriedams (to run - running);
- the past stem with the suffix -us e.g. stāstīja - stāstījusi (told - has been told).

Numerals, similar to nouns, are defined by case, gender and number, and they follow the same flectional patterns as nouns. **Adverbs** belong to the non-inflectional group of words. Adverbs can be derived from other classes of words by deleting the flection or by adding suffixes, i.e. -am, -iem, ām, -im, -u, -i etc., e.g. vakars (an evening) - vakar (yesterday), ātrs (quick) - ātri (quickly). **Prepositions, conjunctions, interjections** and **particles** consist only of the root element or grammatical morpheme, e.g. zem - under, pēc - after. Occasionally, they may have derivatives, i.e. pēc (after) - pēcāk (later). As non-content bearing words and because of the grammatical structure, prepositions, conjunctions, interjections and particles are the most appropriate and relevant candidates for a Latvian stopword list.

In order to compare the morphological system of English and Latvian, the following section will describe the basic structure of English words. The main emphasis will be on affixation in English grammar, as this concept covers the inflectional aspect, which is one of the core elements in Latvian language.

4.4 Comparative analysis of English and Latvian morphology

The English language, as does Latvian, belongs to the Indo-European language group and its character set is based on the Latin alphabet. Both in English and Latvian words are created by adding suffixes and/or prefixes to a basic stem (root) i.e. nation + al = national. However, contrary to Latvian grammar, English morphology contains also lexical free morphemes i.e. nouns, verbs (e.g. land, buy). Figure 4.2 describes types of morphemes used in English (17).

Modern English grammar recognises the following categories of words: nouns, verbs, adjectives, adverbs, prepositions, conjunctions and determiners or articles. Nouns can be divided into pronouns, proper nouns and common nouns, while verbs (auxiliary verbs) can be grouped into modal and non-modal verbs (18). English nouns, similar to those in Latvian, have a number (singular and plural), e.g. a sister - sisters, and a case (possessive case), e.g. sister's. Finiteness is one of the elements which characterises verbs. Finite verbs have either imperative or tensed mood, while non-finite verbs contain infinitive or perfect aspect. Adjectives in English, as in Latvian, contain the comparative degrees, i.e. normal, comparative and superlative. Nouns, verbs and adjectives are the three classes of words which, as shown in Figure 4.2, contain inflectional bound grammatical morphemes or inflectional affixes (suffixes).

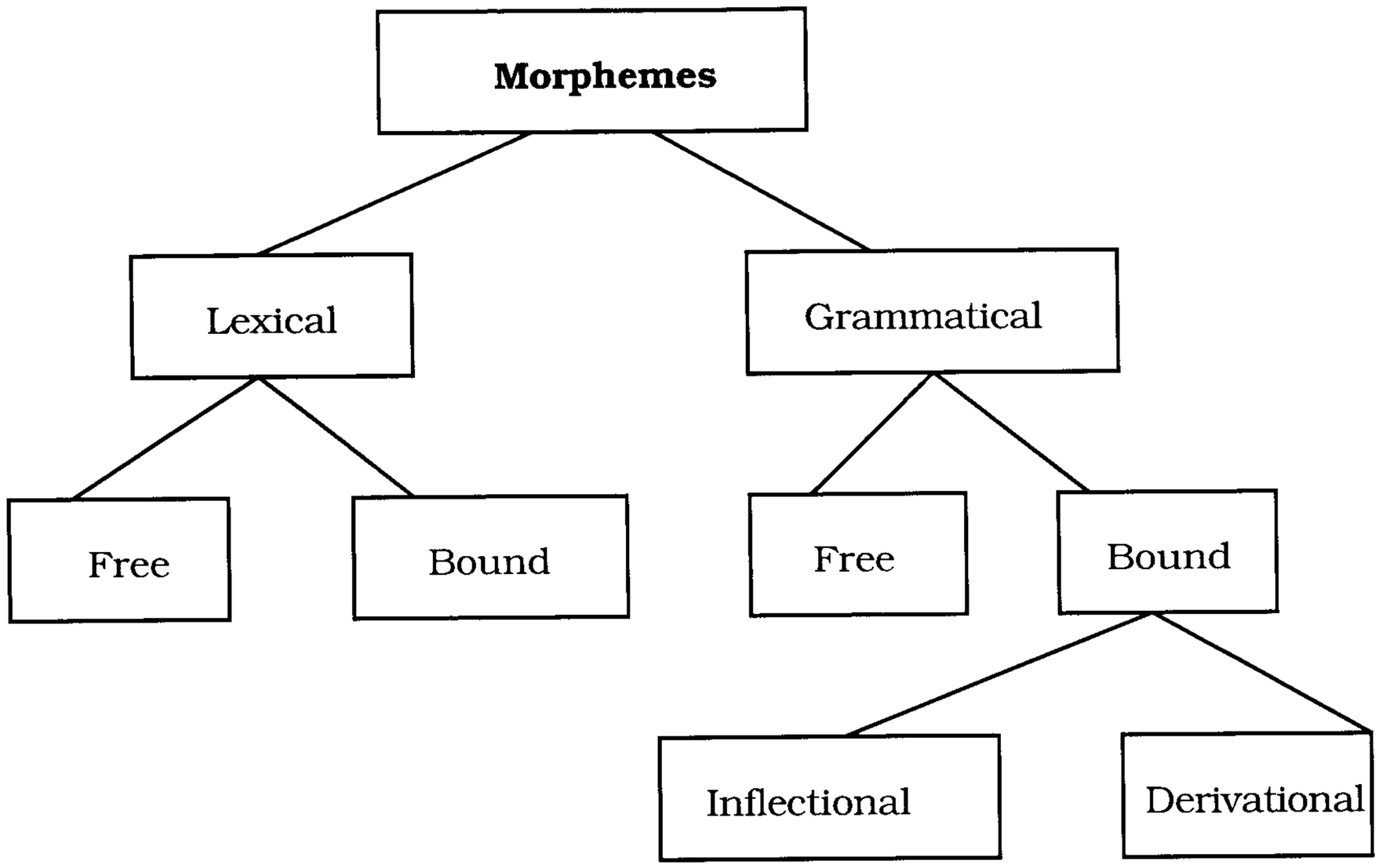

Figure 4.2 Types of English morphemes

The inflective nature of the above mentioned classes of English words is determined by eight inflectional suffixes. The hierachical structure of those suffixes is presented in Figure 4.3 (letter m denotes morpheme).

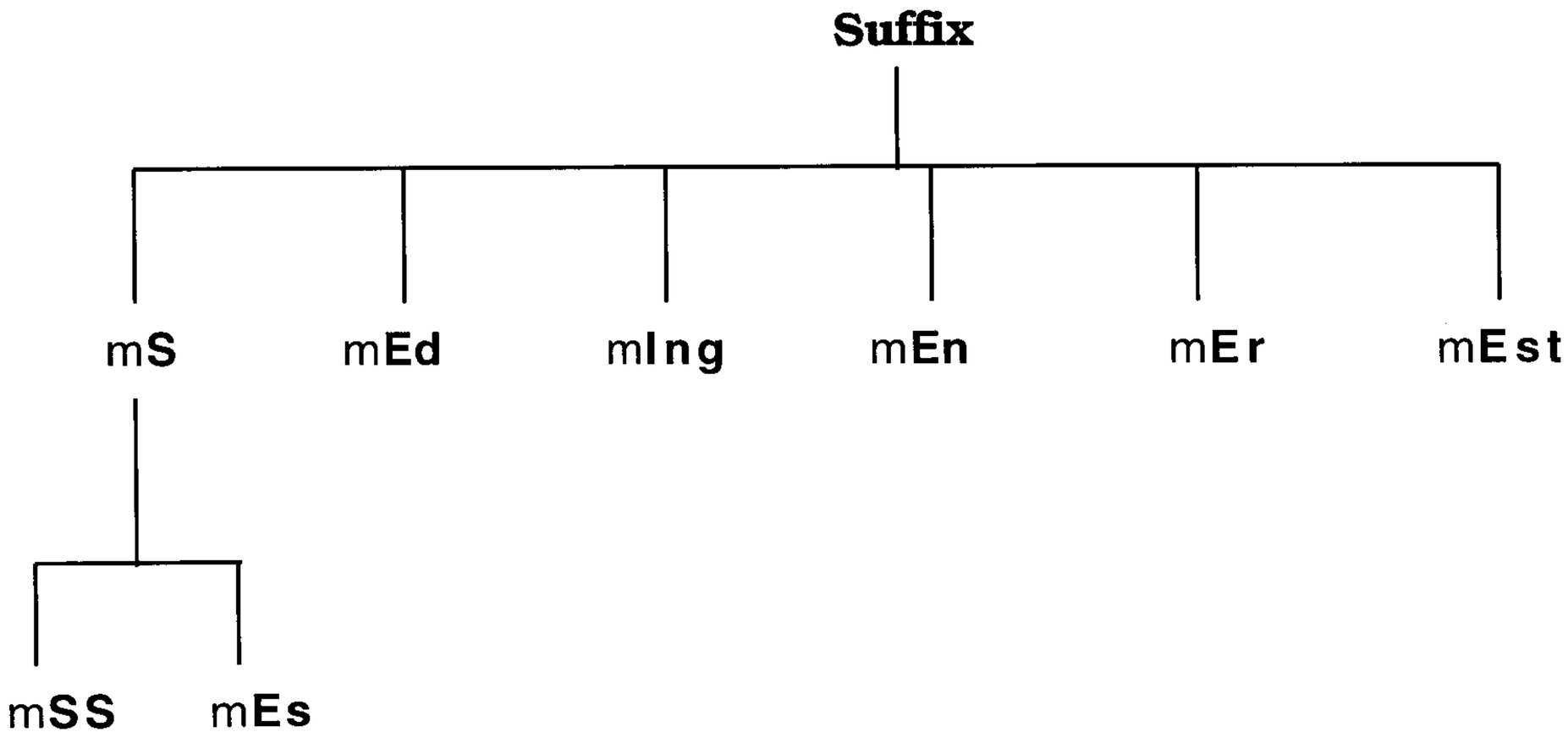

Figure 4.3 Structure of English inflectional suffixes

Suffix or ending -s and, respectively, suffixes -ss and -es are used in plural nouns, i.e. a wing - wings, a church - churches. Possessive nouns also contain suffix -s, e.g. vehicle's. Verbs may contain suffixes -ed, -ing, -en and -s, i.e. all present tense verbs have the ending -s (e.g. to like - she likes), past tense verbs include the suffix -ed (e.g. talked). Suffix -ing is usually found in verbs in the present participle, e.g. walking, while past particles contain the suffix -en, e.g. chosen. Comparative adjectives contain the suffix -er, e.g. smaller, but suffix -est is found in superlative adjectives i.e. nearest. Along with the above mentioned inflectional suffixes, some linguistic theories additionally recognise suffix -n't, which is common for the negative form of verbs e.g. didn't (19). Besides the inflectional suffixes, English grammar includes a certain number of derivational affixes (both suffixes and prefixes) i.e.:

- the suffix -ise is used to change a noun or an adjective into the corresponding verb, e.g. a critic - to criticise, real - to realise;
- the suffix -ful changes a noun into the corresponding adjective, e.g. care - careful;
- the suffix -ly may be attached to an adjective and change it to the adverb or -ly is also used to change a noun into the corresponding adjective, e.g. quick - quickly, a neighbour - neighbourly;
- -un, -dis, -a, -anti are the most frequently used derivational prefixes, e.g. unassuming, atypical, disproportionate(20).

Some English nouns similar to Latvian nouns may have consonant palatalisation, i.e. consonant f in the plural changes to v, e.g. a life - lives. English morphology also contains indefinite and definite forms of words, but contrary to Latvian grammar, these forms are determined by the indefinite article a (an) and the definite article the, e.g. a nice house - the nice house. Table 4.5 summarises the basic common and distinct features of both English and Latvian languages.

Table 4.5 Basic characteristics of Latvian and English morphology

Characteristics/Language	Latvian	English
Language family	Indo - European	Indo - European
Alphabet	Latin (with diacritics)	Standard Latin
Types of morphemes		
Lexical	-	x
Grammatical	x	x
Free	-	x
Bound	x	x
Consonant palatalisation	Nouns in the II, V and VI declensions both in the singular and plural	Some nouns and only in the plural
Inflectional system	For nouns, pronouns, adjectives, verbs and numerals according to the case, declension, gender, conjugation and number	For nouns, verbs and adjectives determined by eight/nine inflectional suffixes
Indefinite and definite form	Determined by the endings of adjectives	Determined by indefinite and definite articles

4.5 Conclusion

From the discussion above about the Latvian morphological system, the following points can be stressed:

- The Latvian language contains a rich inflectional morphology both in verbal and in nominal systems;
- Latvian morphology covers various morphemic alternations, which appear in stems (roots) and in suffixes during inflection. Palatalisation of the final root consonant is one of such alternations, which determines the complexity of the language, especially in terms of computerised word and text processing, i.e. truncation.

Comparison of Latvian and English grammars reveals that the basic principles and methods used for English language with certain modifications might also be applied in Latvian language. However, the complexity of the Latvian inflectional morphology determines that certain types of automatic word conflation methods described in Chapter 3, i.e. successor variety and n-gram stemmers, are basically impossible to apply for Latvian. For example, the successor variety method requires determination of the successor varieties for a word which in many cases is extremely complicated for Latvian words.

As described in Chapter 3, many non-English stemmers are based on the affix removal approach, e.g. a stemming algorithm for Slovene language which incorporates Porter's stemmer. Affix removal algorithms involve human preparation of stoplists, suffix lists and recoding/condition rules, which therefore allow definition of specific suffix removal rules for each particular case.

The following chapter (Chapter 5) will describe methods and principles used for creating a list of stopwords for Latvian. A brief description on research and development related to stoplist design in Latvia is also included.

References

1. **Shabad, Theodore.** Latvia. In: *The Encyclopedia Americana*, vol. 17, 1978, pp. 55-57.

2. **Bērziņa-Baltiņa, V.** *Latviešu valodas gramatika* [The grammar of Latvian language], 1965, pp. 8-9.

3. Latvian language. In: *The New Encyclopaedia Britannica*. Micropaedia, 15th ed., vol. 7, 1988, p. 186.

4. **Fennell, Trevor G. & Henry Gelsen.** *A grammar of modern Latvian,* vol. 1, pp. XXIII-XXVIII.

5. **Bērziņa-Baltiņa,** ref. 2, pp. 19-20.

6. **Ziemele, Lidija.** *Praktiska latviešu valodas un pareizrakstības mācība* {Practical Latvian grammar and spelling], 1979, pp. 47-50.

7. **Anderson, Stephen A.** Morphological theory. In: Frederick J. Neumeyer,ed. *Linguistic theory. Part of Linguistics*: the Cambridge survey, vol. 1, 1988, p. 146.

8. **Parker, Frank.** *Linguistics for non-linguists,* 1986, p. 66.

9. **Ceplīte, B. & L. Ceplītis.** *Latviešu valodas praktiskā gramatika* [The practical grammar of Latvian language], 1991, pp. 6-7.

10. *Ibid.*

11. Interview with Dr. Sarma Kļaviņa, Department of Baltic Languages, University of Latvia, Riga, Latvia, 4 April 1996.

12. *Ibid.*

13. **Metuzāle-Kangere, Baiba.** *A derivational dictionary of Latvian,* 1985, p. XXXII.

14. **Parker,** ref. 8, p. 68.

15. **Ceplīte,** ref. 9, pp. 8-9.

16. **Metuzāle-Kangere,** ref. 10, p. XII.

17. **Parker,** ref. 8, pp. 65-82.

18. **Hudson, Richard**. *English word grammar*, 1991, pp. 167-188.

19. *Ibid.*

20. **Parker,** ref. 8, pp. 73-74.

Chapter 5

DESIGN OF A LATVIAN STOPWORD LIST

5.1 Introduction

As noted in Chapter 3, almost all automatic word conflation algorithms (i.e. FIRST, MARS, MORPHS) incorporate so called stoplists or lists of stopwords. It was observed already in the early stage of automatic information retrieval that non-content bearing words and frequently occurring words are poor indexing and search terms, so they should be eliminated from further consideration. For example, Luhn (1) stated that the frequency of words in a text is related to the importance of those words for content representation and that many frequently appearing words in English are short function words which therefore are worthless as indexing terms. Salton (2) emphasised that a search using words with low discrimination value will retrieve almost every document in a database, therefore such words must be included in a stoplist. Fox (3) defines a stoplist as "a list of words filtered out during automatic indexing because they make poor index terms". The same author presents two approaches on how to filter stoplist words from an input stream:

- to examine the output of lexical analyser and eliminate any stopwords;
- to remove stopwords as a part of lexical analysis (4).

Lexical analysis is the first stage of automatic indexing and query processing which converts an input string of characters into a stream of words. It means that lexical analysis allows the removal of all non-alphabetic characters i.e. digits, hyphens and other punctuation from the input word before it is further processed i.e. stemmed. Besides that, lexical analysers usually convert all characters either to lower or upper case as the case of letters is not significant neither for indexing terms nor for query search terms.

Both above mentioned methods are similar and usually, depending on the type of stemmer, they are built into the initial stemming program. The first method mentioned above removes stopwords from the output of lexical analyser. It means that each word after lexical analysis is checked against the stoplist and eliminated if found. The second approach allows the exclusion of stopwords as a part of lexical analysis. Removal of Latvian stopwords is based on the first technique as the lexical analyser is incorporated

within the stemming algorithm before the stoplist. After examination of input characters, the stemmer matches all words against the list of stopwords and removes if found. Overall, elimination of stopwords increases the speed of automatic indexing, saves disk space and improves effectiveness of information retrieval.

This chapter will cover a brief overview on research and development in computing linguistics and information retrieval with particular emphasis on design and construction of Latvian stopword lists. It will be followed by a section on the basic principles and criteria of creating a list of stopwords for Latvian language.

5.2 Research on computing linguistics and information retrieval in Latvia

Research on computerised analysis and processing of Latvian language started in early 1970's when V. Drizule at the Institute of Electronics and Computing Technology designed an algorithm for the morphological analysis of Latvian words (5). The main purpose of this experiment was to examine and evaluate correlation between the grammatical nature of a word and the final letters which form the ending of this word. The algorithm contained one table-matrix of endings which consisted of two letters and 36 matrixes which contained endings with three or four letters. Basically, the program analysed Latvian words extracted from scientific and technical documents and defined grammatical forms of those words i.e. nouns, adjectives etc. based on their final letters. It was recommended that this computerised morphological analyser of Latvian words be implemented into an information and/or document retrieval system. However, due to financial problems and several technical reasons i.e. the algorithm was created in BASIC programming language and based on mini personal computers "WANG", it has never been developed to such a level.

In 1970's the Laboratory of Mathematical Linguistics of the Institute of Linguistics and Literature together with the Institute of Electronics and Computing Technology created a frequency dictionary of Latvian language (6). The dictionary contained words which were taken from more than 300 Latvian texts in science and humanities and which were arranged according to their computed frequency value. Table 5.1 shows an example of the most frequently used Latvian words which are taken from the frequency dictionary.

The Artificial Intelligence Laboratory of the Institute of Mathematics and Computing Science in 1990 created an electronic dictionary which covers more than 35,000 Latvian words i.e. nouns, adjectives, verbs (7). This dictionary was based on the Latvian inverse dictionary (8) which was compiled using a Latvian-Russian bilingual dictionary

published in 1964. The main purpose of developing such an electronic dictionary was to have the majority of Latvian words in a computerised form which could eventually be used in morphological and grammatical evaluations. The computerised analysis of sentences and automatic declination of Latvian words are also research areas of this laboratory.

Table 5.1 Example of word frequency in Latvian texts

Words		Frequency*
un	(and)	9899 / 300
būt	(to be)	7868 / 300
tas	(this)	5607 / 300
ar	(with)	5027 / 300
no	(from)	3651 / 299
kas	(what)	2692 / 295
šis	(that)	2653 / 298
arī	(also)	2605 / 296
par	(about)	2585 / 294
ka	(because)	2429 / 285
kā	(such as)	1921 / 297
varēt	(to be able)	1874 / 277

* the number shows how many times totally and in how many texts a word has been used e.g. 9899/300 means that the word un was used 9899 times and it appears in 300 documents. There were 300 documents in the sample.

In early 1990's the information and translation company TILDE designed the first spell checker for Latvian language. The spell checker breaks up each word into syllables and than compares the divided word with the original one from the built-in dictionary.

At present there are two integrated information and library systems in Latvia which offer advanced information retrieval capabilities. The majority of university (academic) libraries in Latvia use the integrated library and information system ALISE (Advanced Library Information Service) originally designed and developed at the University of Latvia (9). Along with the traditional information access points i.e. author, title, keyword(s), subject etc., the system also offers right hand truncation of search terms. For example, in order to retrieve all documents on economics, economy, economical etc. user can enter a truncated form of those words i.e. ns=econom or ns=economic (ns is Latvian abbreviation for title). Both Boolean and comparative operators are used in ALISE, too.

LIBER is the integrated library and information system installed at the Latvian Academic Library (before known as the Main Library of the Latvian Academy of Sciences). Similarly to ALISE, the system offers traditional information retrieval criteria

as well as Boolean operators and word truncation (10). The complexity of Latvian grammar is one of the main reasons why a stemming algorithm for Latvian language has not been developed yet. However, the morphological comparison of English and Latvian languages discussed in Chapter 4 revealed, that advanced information retrieval methods i.e. stemming used for English, with certain modifications might be also applied for Latvian language. Moreover, both of the present information systems in Latvia mentioned above incorporate lists of some Latvian stopwords mostly prepositions and conjunctions. At present the ALISE information system incorporates a list of 19 prepositions and conjunctions created by librarians of the University of Latvia Library i.e. AIZ (beyond), AR (with), BET (but), JA (if), JĀ (yes), KA (that), KĀ (how), KO (what), NE (not), NĒ (no), PA (along, on), PAR (about), PĀR (over), PĒC (after), PIE (at), UN (and), UZ (on) VAI (whether), ZEM (under) (11).

LIBER integrated information system also includes a short list of 8 English, German and Latvian articles and conjunctions i.e. A, THE, DER, DIE, DAS, UN (and), VAI (whether) and BET (but) (12). Both integrated information systems offer the opportunity to develop an individual stoplist within each organisation/ institution which uses the particular system.

A number of information and computing specialists in Latvia i.e. I. Greitane from the Institute of Mathematics and Computing Science (13) and A. Klints from the ALISE Information Systems Ltd company (14) suggested that a longer list of Latvian stopwords will increase the efficiency of information retrieval and that such stoplist at least should include all conjunctions, prepositions and particles. Moreover, Dr. V. Drizule an expert of computing linguistics at the University of Latvia, recommended adding the majority of Latvian adverbs which are not containing meaningful information into the stoplist, too (15).

5.3 Construction of the Latvian stoplist

As it was pointed out before, non-content bearing words or function words as well as high frequency words are the most commonly used candidates for a negative dictionary (stoplist). Traditionally, a list of stopwords includes following word classes:

- prepositions
- conjunctions
- particles
- auxiliary verbs

For example, Van Rijsbergen's stoplist covers 250 English prepositions, conjunctions, articles and pronouns (16). Along with the above mentioned word groups, a list of Slovene stopwords also comprises substantive and adjectival pronouns, numerals, adverbs and predicates (17). The total amount of words covered by the stoplist can vary, e.g. the commercial information system ORBIT has only eight English stopwords but the Slovene stemming algorithm includes a stoplist with more than 1.500 words (18). However, the design and implementation of relevant stopword list can evidently increase the speed and efficiency of information retrieval.

According to the analysis of general categories of words discussed in Chapter 4 and taking into account the existing stoplists, the first candidates for Latvian stoplist were:

- prepositions
- conjunctions
- particles

Overall, 33 prepositions, 48 conjunctions and 15 particles has been included into the stoplist (See Appendix 1). A number of conjunctions consist of more than one word where often both words are the same e.g. ne - ne (neither nor), gan - gan (as as) etc. In that case only the first word was included into the stoplist. The same principle was also used for compound conjunctions and particles which means that those compound auxiliary words consisting of the same words as single conjunctions and particles, were not repeated in the stopword list e.g. lai (may), arī (also), lai arī (may also), it (in particular), kā (how), it kā (whatsoever).

Along with prepositions and conjunctions, the Latvian stoplist also contains 461 pronouns. The number of pronouns in the stoplist is so high because in Latvian both indefinite and definite pronouns in masculine and feminine cases both in singular and plural forms have six declensions, and the stopword list covers all types and forms of pronouns in all declensions. Pronouns which have the same form in more than one declension, were included in the stoplist only once e.g. mana (mine - masculine, genitive), mana (my - feminine, nominative).

As mentioned before, a number of stoplists contain auxiliary verbs therefore, it was decided to include all four Latvian auxiliary verbs būt (to be), tikt (to arrive), tapt (to become), kļūt (to become) into the list of stopwords. Because of the different modes and tenses, totally there are 55 forms of the above mentioned infinitive auxiliary verbs which are covered by the stoplist. Three forms of the auxiliary verbs were excluded from this stoplist as they overlap with corresponding nouns which are not stopwords

i.e. tapa (past tense from tapt and a noun - spigot), topi (present tense from tapt and a noun - tops) and tiksi (future tense from tikt and a name of the Far North city).

No Latvian verbs apart from varēt (to be able) were included in the stoplist as this verb has very high frequency coefficient (19). All forms of this verb except varu (am able) and varam (are able) which overlap with a noun *copper* in genitive and dative declensions, were chosen for the list. The stoplist also covers 25 most often used interjections which were extracted from the electronic dictionary compiled at the Institute of Mathematics and Computing Science. Finally, according to suggestions 189 adverbs e.g. kāpēc, kādēļ (why), tālab, tālabad (because), dikti (very) etc. were chosen for the stoplist (20). Because of complexity and variety i.e. number of declensions, indefinite and definite case etc., no numerals were included into the stoplist.

The total amount of words chosen for the list of Latvian stopwords was 839. There exists two basic methods for adding a stoplist to the stemming algorithm:

- to create a separate list of stopwords with a link to the main stemming algorithm;
- to place stopwords in the original stemmer by adding a separate set of rules for the stoplist.

The number of stopwords and the flexibility of an algorithm used for Latvian word stemming determined to choose a second approach for the Latvian stoplist. To filter stopwords in the stemmer, a separate set of rules i.e. static RuleList step0a to step 0n were created (See Appendix 2.3). The stopword removal rules along with the stoplist were placed before stemming rules to ensure that stopword elimination is carried out before the actual word stemming starts.

5.4 Conclusion

The design and construction of a Latvian list of stopwords involves the same principles and methods as the majority of existing stoplists i.e.only non-content bearing words and frequently used words have been chosen for the stoplist. Stopwords from both ALISE and LIBER integrated systems were transferred into the final list of Latvian stopwords. Because of morphological complexity, no particular frequency measurements of Latvian words have been carried out. However, all candidates for the Latvian stoplist were compared and analysed against the automatically compiled Frequency Dictionary of Latvian Words. Not all frequently used words listed in the

Dictionary were relevant for the stoplist because the Dictionary covers words in specific subject areas, whereas the Latvian stemmer is designed for general texts.

Because the list of Latvian stopwords is placed within the structure of Latvian stemming algorithm, the evaluation of Latvian stoplist will be carried out along with analysis and evaluation of the whole Latvian stemmer (Chapter 7). The next chapter (Chapter 6) will justify the choice of algorithm for Latvian stemmer. It will be followed by the description of methods and principles used for design and construction of the Latvian stemming algorithm.

References

1. **Luhn, H.P.** A statistical approach to mechanized encoding and searching of library information. *IBM Journal of Research and Development*, 1957, **1**(4), 309-317.

2. **Salton, Gerard & Michael J. McGill.** *Introduction to modern information retrieval*, 1983, 131-132.

3. **Fox, Christopher.** Lexical analysis and stoplists. In: William B. Frakes & Ricardo Baeza-Yates, eds. *Information retrieval: data structures and algorithms*, 1992, p. 113.

4. *Ibid.*, p.116.

5. **Drizule, Viktorija.** *Razrabotka priblizhennih metodov avtomaticheskovo morfologicheskovo analiza tekstov latysskovo jazyka* [Design of proximate methods for morphological analysis of Latvian language]: A summary of dissertation, 1975, pp. 3-18.

6. **Jakubaite, T. et al.** *Latviešu valodas biežuma vārdnīca* [Frequency dictionary of Latvian language], vol. 4: Science, 1976, 644 p.

7. Interview with Andrejs Spektors, Institute of Mathematics and Computing Science, Riga, Latvia, 26 March 1996.

8. **Soida S. & S. Kļaviņa.** *Latviešu valodas inversā vārdnīca* [Inverse dictionary of Latvian language], 1970, 256 p.

9. Arts Klints to Karlis Kreslins, 25 January 1996.

10. Interview with Aivars Liepa, Latvian Academic Library, Riga, Latvia, 3 January 1996.

11. Arts Klints to Karlis Kreslins, 27 January 1996.

12. Aivars Liepa to Karlis Kreslins, 14 September 1996.

13. Inguna Greitane to Karlis Kreslins, 14 September 1995.

14. **Klints**, Ref. 9.

15. Interview with Viktorija Drizule, Riga, 4 April 1996.

16. **Van Rijsbergen, C.J**. *Information retrieval,* 1975, pp. 17-19.

17. **Popovic, Mirko.** *Implementation of a Slovene language based free-text retrieval system,* 1991, p. 98.

18. **Fox,** ref. 3

19. **Jakubaite,** ref. 6, p.25.

20. **Drizule,** ref. 15.

Chapter 6

CONSTRUCTION OF A LATVIAN STEMMING ALGORITHM

6.1 Introduction

The grammatical structure of language is one of the basic factors which determine the choice of appropriate stemming algorithm for automatic word conflation in this language. According to Chapter 3, the majority of non-English stemmers i.e. French (1) or Slovene (2) algorithms are based on either longest match or iterative stemming principle. Conclusions from Chapter 4 confirm that suffix removal algorithm will be the most relevant stemming method also for the morphological structure of Latvian language.

The above mentioned conclusion was approved by linguistic experts in the Latvian language. V. Drizule said that one of the first ending and suffix removal algorithm for Latvian language which was designed and tested in 1970's and which achieved good test results, was based on iterative affix removal principle (3). It was emphasised also by S. Klavina that the complexity of Latvian grammar i.e. different declensions, conjunctions, consonant palatalisation required the use of specific type of stemming algorithm which allowed the definition of the length of each ending and/or suffix individually according to the size of a word (4). Taking into account the requirements for automatic word conflation in the Latvian language discussed previously, Porter's stemming algorithm modified by B. Frakes. C. Cox and S. Fox was chosen as the basis for a Latvian stemmer. Moreover, the rules and conditions of Porter's stemmer were successfully used for word stemming in Slovene language. As both Latvian and Slovene are inflective languages, the chosen algorithm should be also relevant for stemming Latvian words.

This chapter will cover basic characteristics of the original stemming algorithm designed by Porter and implemented by Frakes, Cox and Fox. It will be followed by a detailed description of modifications and changes which were carried out to the initial structure of the stemmer in order to construct the stemming algorithm for Latvian language.

6.2 Description of the initial stemming program

Porter's stemmer is one of the suffix removal algorithms which processes the word using an iterative stemming approach. Detailed description of the original Porter's algorithm is given in Chapter 3, therefore this section will cover main modifications and changes which were carried out by B. Frakes, C. Cox and C. Fox.

B Frakes and C. Cox in 1986 and C. Fox in 1990 implemented the Porter algorithm and added following new features to the initial program:

- designed a deterministic finite automata for computing the word size
- restructured existing structures of the algorithm
- renamed functions and variables
- restricted function and variable scopes (5).

The Deterministic Finite Automation (DFA) is used to control the validity of a word size based on vowel-consonant pairs. DFA calculates the number of vowel-consonant pairs in a word ignoring initial consonants and final vowels. The finite automata includes three different arcs for measuring the word length. Arc 0 checks the first letter of a word. If the initial letter is a vowel, then the DFA switches to arc 1 which is the vowel arc . If the initial letter is a consonant or y, then the machine changes to arc 2 which is a consonant arc. The result counter is incremented on the transition from arc 1 to arc 2 as this transition occurs after calculation of a vowel-consonant pair. For example, the word size of *"town"* is 1 as there is only one vowel-consonant pair *'ow'* , but the word length of "level" is 2 because the word has two consonant-vowel pairs *'le'* and *'ve'*.

The structure of the initial Porter's algorithm was also changed by the above mentioned authors. For example, all five basic Porter's rules (6) were arranged into separate rule lists and within the rule list each rule was numbered. Besides that a definite condition was added to each rule which determined the length of ending and/or suffix and the minimal valid length of the resulting word stem. The size of a word before stemming was calculated by the DFA using condition rules mentioned before.

Several rules were rewritten and a number of functions and variables were renamed by Frakes, Fox and Cox. For example, rules 106 and 107 in the rule list step1b_rules were applied only if the word parameter contained a vowel. It meant that a word had to contain a vowel either as its first letter or in any other letters in a remaining stem. Rule 502 in the static rule list step5a_rules was applied only if the current word met a special condition for removing letter "e" i.e. the state of a word root was no less than 1 and

there was no consonant-vowel-consonant combination at the end of a word.

Apart from the rewritten rules, several variables and functions have been renamed and/or added e.g. 'integer word size' which counts the length of a word, integer replace end which replaces the end of a word if the corresponding rule and condition is valid, and others. The modified program also contains restricted functions and scopes of variables i.e. integer for minimum root size, integer for replacing the end of a word.

In July 1991 the modified program was changed by C. Fox who:

- added ANSI C declarations
- designed a program STEMMER.C for testing the validity of input character stream.

It means that the stemmer was fully implemented in the C programming language. An additional program STEMMER.C was created which checks the validity of each input word (See Figure 6.1). According to the flowchart in Figure 6.1, the basic stemming program STEM.C is embedded into the STEMMER.C which means that after the input word has been tested, it will processed by the stemming procedure and then the end result will be output on the screen using the main program STEMMER.C.

Overall, the English version of the stemming algorithm processes a word as follows (Figure 6.1 and 6.2):

1) each word is scanned to separate and remove non-alphabetic characters i.e. digits, punctuation, other symbols (STEMMER.C);
2) after the scanning all upper case letters are changed to the lower case (STEMMER.C);
3) additional testing is carried out to determine whether a word contains any vowels. No stemming action will be taken if the test word contains only consonants (STEMMER.C)
4) the valid word is matched against the first set of rules (step 1a and step 1b). If it meets conditions and the root length of a word is valid, the ending is replaced or removed (STEM.C)
5) after that the modified word is matched against the list of suffixes and if the test word contains a valid suffix it will be replaced (step 1b1, 1c, 2 and 3) and then removed (step 4) (STEM.C);
6) if the word ends with -e and root length is valid, it will be removed (step 5a) and if the word contains the ending -ll, the second -l will be removed (step 5b); (STEM.C)
7) finally, the stemmed word is output to the screen (STEMMER.C).

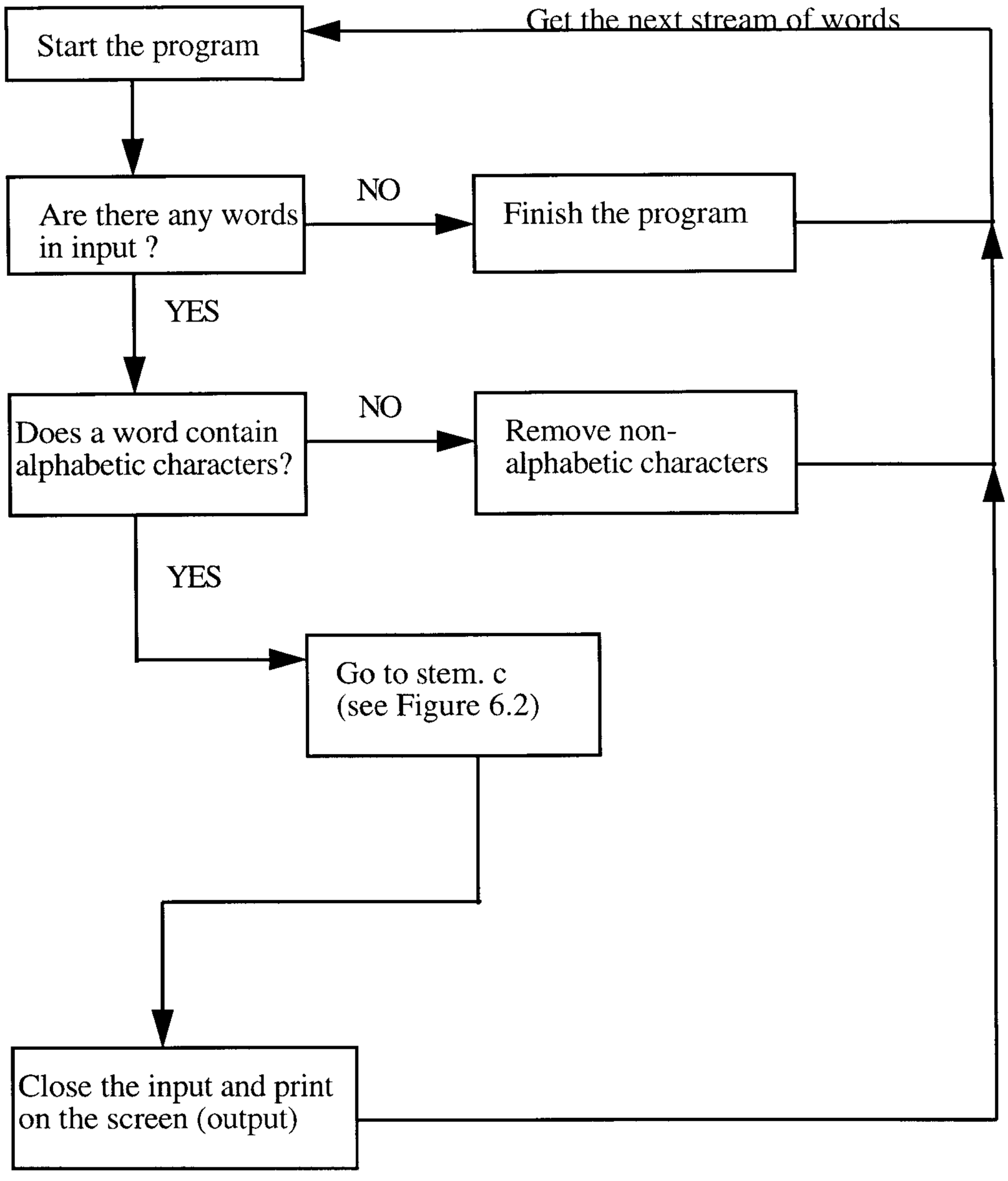

Figure 6.1 Flowchart of the main stemmer (STEMMER.C)

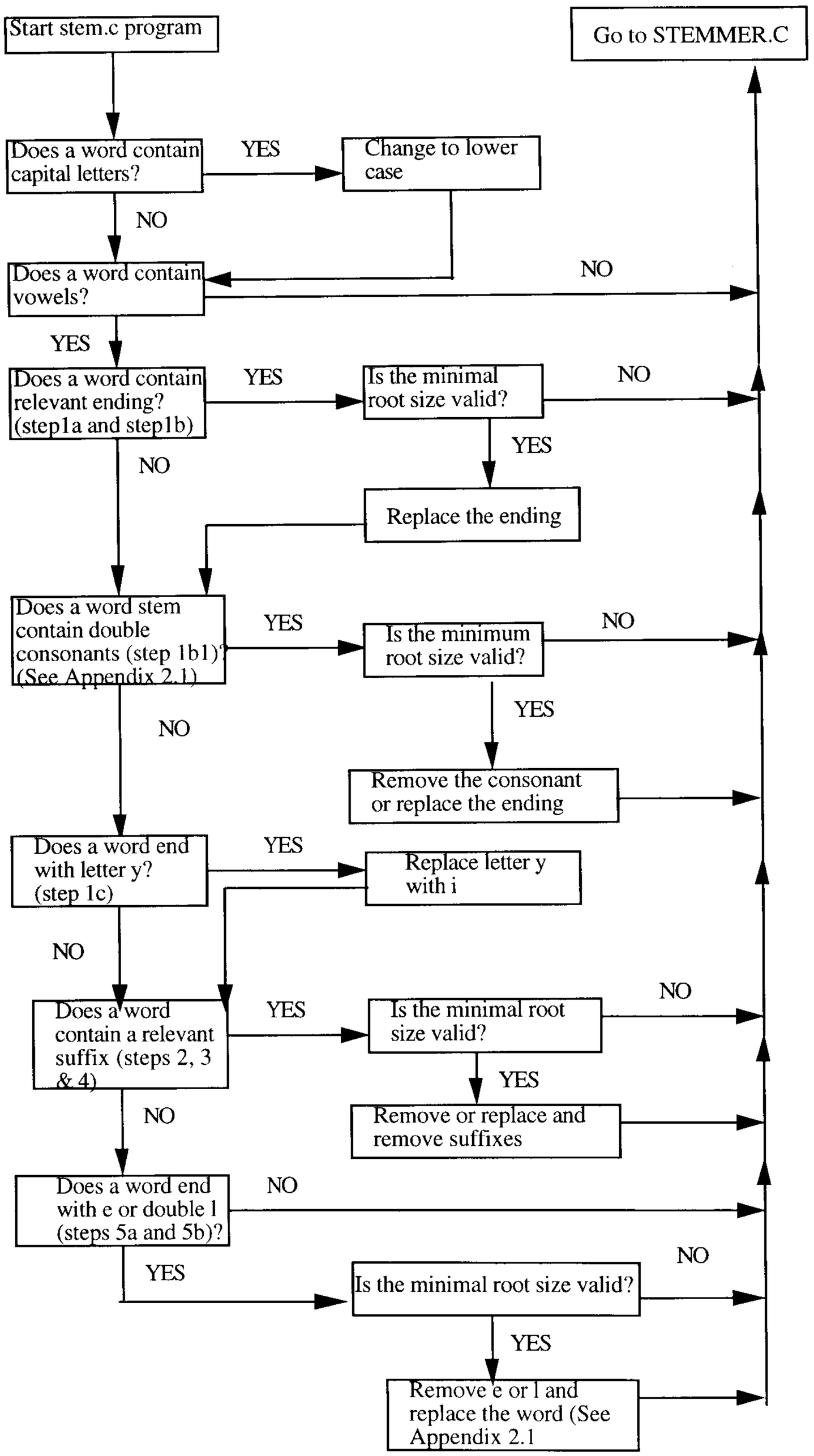

Figure 6.2 Flowchart of the stemming procedure for English (STEM.C)

6.3 Development of the Latvian stemmer

The construction of a stemming algorithm for Latvian includes the following steps:

- design of Latvian stoplist
- general modifications of the original algorithm
- creation of list of endings
- modification of rules for Latvian consonant palatalisation
- design of Latvian suffix list
- creation of additional rules
- testing and modification of the designed stemmer

All the mentioned components will be described below in separate sections except of design and implementation of the Latvian stoplist, which was discussed before in Chapter 5.

6.3.1 General modifications

Development and modifications of the Latvian stemming algorithm which initially were carried out using Borland C++2.0 DOS version have been converted to Object Windows version 4.0 for Borland C++ programming language. Figure 6.3 presents the flowchart of the Latvian stemming algorithm. Initial modifications were related to the Latvian character set (See Appendix 2.3). A new integer isLatv was introduced to recognise Latvian letters. Additional static integers *SLatv and *BLatv were defined to switch also upper case letters with diacritics to lower case.

Apart from the Latvian character set, all Latvian vowels were defined separately in 'Private Defines and Data Structure', which determines all integers i.e. minimal root size, suffix size etc. in order to comput the word size using DFA. Because of the similar grammatical structure of both Latvian and English words, no changes except for the addition of Latvian vowels, were applied to the deterministic finite automata (DFA). All conditional and recoding rules for English in the 'Initialised Private Data Structures' were replaced with Latvian ones.

Modifications which were added to the program STEMMER.C i.e. integer for checking the validity of Latvian letters, are placed in Appendix 2.4 and highlighted in bold.

6.3.2 List of Latvian endings

Endings of Latvian nouns, verbs and adjectives were placed in a separate set of rules (static rule list step 1a to step 1a6) which follows the lists of Latvian stopwords (step 0a

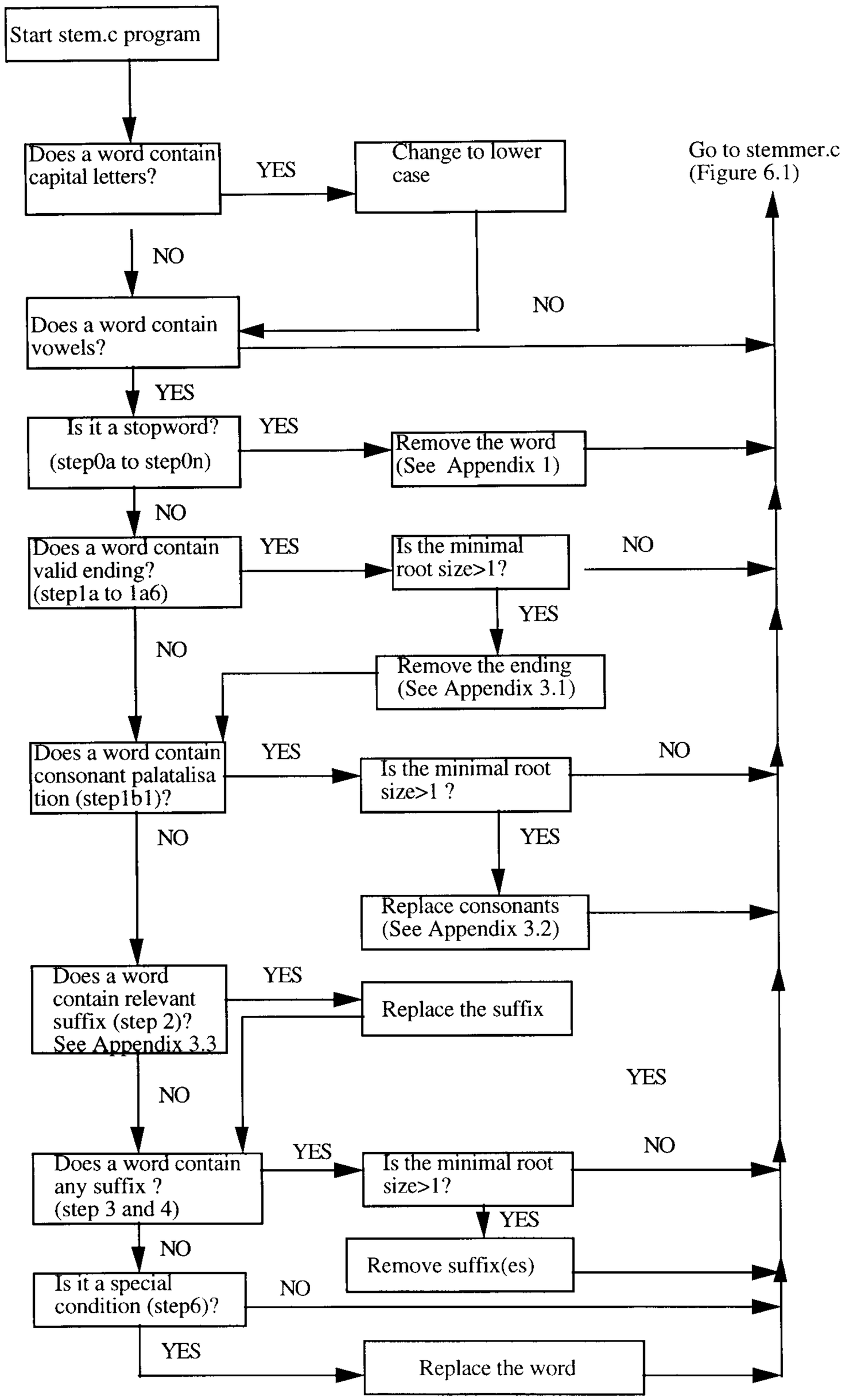

Figure 6.3 Flowchart of the Latvian stemming algorithm

to step 0n). Latvian morphology produces a large number of word variants which include a number of distinct endings. An ending is the last letter or letters in a word which follow after the root or the stem. Latvian words especially nouns and adjectives are usually used not only in a standard Nominative case but also in other declensions i.e. Genitive, Dative etc. Therefore, all forms of declension both in singular and plural are covered by the list of Latvian endings (See Appendix 3.1).

In terms of the word size, the majority of endings will be removed only if the word has two or more syllables (vowel and consonant pairs). These restrictions regarding the word size were introduced, because for short Latvian words consisting of three or four characters the final root after removing endings can be common for multiple different words which in information searching will cause retrieval of many nonrelevant documents. For example, for the word aka (a well) following additional words will have the same root ak, if the ending -a will be removed:

akacis - a bog
akācija - an acacia
akadēmija - an academy
akcija - a share
akcents - an accent
akcīze - an excise tax

6.3.3 Consonant palatalisation

Rules 108 to 117 in the static RuleList step1b1 deal with the consonant palatalisation which is characteristic for certain II declension Latvian nouns in plural Genitive and/or Nominative cases i.e. dzērve (a crane) - dzērvju (cranes'). According to the above mentioned rules, all words containing the consonant palatalisation will be converted from the plural Nominative and/or Genitive case to the singular Nominative (See Appendix 3.2).

However, nine out of twenty cases of consonant palatalisation described in the textbook of Latvian grammar (7), are covered by the Latvian stemming algorithm because there are several exceptional words i.e. homographs for which one or another type of consonant palatalisation can be determined only from the context e.g. a word ložu can contain either consonant ž-d palatalisation if it means bullets' (in possessive case), or z-ž palatalisation if it is used as the term lottery tickets' or the same word can contain no consonant palatalisation if it means boxes'. Because of the morphological complexity, the following Latvian consonant palatalisations were not possible to include into conditional rules of the stemming algorithm:

- consonant ņ - n palatalisation;

- consonant ļ - l palatalisation;
- palatalisation of consonants t - š;
- palatalisation of consonants d - ž;
- consonant c - č palatalisation;
- consonant s - š palatalisation;
- consonant z - ž palatalisation;
- palatalisation of consonants kst -kš;
- palatalisation of consonants st - š.

6.3.4 Design of the Latvian suffix list

Since Latvian compared to English language is characterised by a wider range of suffixes which have three or more letters, it is difficult to define a relevant threshold for automatic choice of suffixes. A suffix is a part of the word between the root and the ending. Therefore, the selection of suffixes for the Latvian stemming algorithm was carried out manually, using textbooks of Latvian grammar (8, 9) as well as recommendations from linguists (10, 11).

All selected Latvian suffixes are grouped into three separate sets of rules: static RuleList step_2 rules, static RuleList step3_rules and static RuleList step4_rules (See Appendix 2.3). Rules 203 to 214 are modifying resulting word stems by removing suffixes which are proceeding word endings. For example, suffix -iecīb in the word lauksaimniecība (agriculture) will be changed to -iec by removing -īb. Rules 301 to 322 and 401 to 437 are listing suffixes which are grouped according to their type and length and which will be removed from a word if the remaining stem satisfies a certain condition of the minimal stem length. For example, in the above mentioned word lauksaimniecība suffix -iec will be removed after stripping the ending -a, if the minimal length of a remaining word is more than two syllables.

6.3.5 Special conditions

Apart from the sets of rules covering endings, suffixes and cases of consonant palatalisation in Latvian words, a separate rule dealing with the stem -šun was added to the stemmer. The case was classified as a special condition because the word šunelis is the only one where consonant š-s palatalisation is at the beginning of a word. After the ending -is and the suffix -el is removed from the initial word šunelis (a doggie), the rule 601 converts remaining stem šun to sun which is a stem of the word suns (a dog).

Because the diminutive form of this word is completely different from the normal form, it was not possible to process the word and its remaining stem using any of the above mentioned sets of rules.

6.3.6 Testing and analysis of the stemmer

Detailed analysis of the Latvian stemming algorithm using several evaluation approaches i.e. electronic dictionary, user enquiries etc. as well as final corrections and modifications which were carried out to the stemmer according to the obtained results of evaluation, is discussed in the following chapter (Chapter 7). However, in order to observe and to assess generally the relevance and success of automatic Latvian word conflation procedure performed by the stemmer, simple preliminary tests based on single Latvian words were carried out.

For example, several test words e.g. FIZIKA, INFORMĀCIJA with a maximum length three syllables and containing no suffixes were entered in the test file to check the accuracy and consistency of Latvian ending removal. Test revealed that the algorithm correctly removed standard endings and with a few exceptions e.g. endings in plural Dative, also endings in different declensions.

To check the correctness of suffix stripping and relevance of the remaining stem, a small number of Latvian test words i.e. nouns, adjectives, verbs containing one or more suffixes e.g. MĀJA, BIBLIOTĒKA, SKOLA etc. were processed by the stemmer. Test results showed that overall the majority of words were stemmed correctly.

6.4 Conclusion

Porter's iterative stemming algorithm which have achieved good results in automatic word conflation both for English and non-English languages i.e. Slovene, has been chosen also for the construction of the Latvian stemmer. Basic principles and the structure of the initial Porter's algorithm was maintained in the stemming algorithm for Latvian. However, the complexity of Latvian language and its morphological structure required the additional removal or change of several rules in the initial stemmer e.g. rules dealing with consonant palatalisation, as well as to carry out a number of general modifications concerning Latvian diacritics, word length etc.

Because the detailed analysis of the Latvian stemmer will be discussed in Chapter 7, only preliminary tests were carried out to assess the general operation and functions of

the stemming algorithm. Along with evaluation methods which were used to analyse the effectiveness of the designed Latvian stemming algorithm including recall and precision criteria, the following chapter will also describe general methodology applied for detailed evaluation of the stemmer. All additional modifications to the stemmer, which were carried out according to the results obtained, are covered by Chapter 7, too.

References

1. **Savoy, Jaques.** Stemming of French words based on grammatical categories. *Journal of the American Society for Information Science,* 1992, **44** (1), 2-7.

2. **Popovic, Mirko & Peter Willett.** The effectiveness of stemming for natural - language access to Slovene textual data. *Journal of the American Society for Information Science*, 1992, **43** (5), 384-390.

3. Interview with Viktorija Drizule, Riga, 4 April 1996.

4. Interview with Dr. Sarma Klavina, Department of Baltic languages, University of Latvia, Riga, Latvia, 4 April 1996.

5. **Frakes W.B.** Stemming algorithms. In: William B. Frakes & Ricardo Baeza-Yates, eds. *Information retrieval: data structures and algorithms*, 1992, pp. 151-160.

6. **Porter, Martin.** An algorithm for suffix stripping. *Program,* 1980, **14** (3), 130-137.

7. **Ceplīte, B & L. Ceplītis.** *Latviešu valodas praktiskā gramatika* [A practical grammar of Latvian language], 1991, pp. 25-50.

8. **Metuzāle-Kangere, Baiba.** *A derivational dictionary of Latvian,* 1985, pp. XXV- XXVIII.

9. **Ceplīte**, ref. 7.

10. **Drizule**, ref. 3.

11. Interview with Andrejs Spektors, Institute of Mathematics and Computing Science, Riga, Latvia, 10 April 1996.

Chapter 7

EVALUATION OF A LATVIAN STEMMING ALGORITHM

7.1 Introduction

As noted by Hull (1), the most often used methodology for the evaluation of a new information retrieval strategy i.e. stemming incorporates the following steps:

- choice of relevant test collection(s) with user search statements;
- testing of the new retrieval method by carrying out an information retrieval experiment and comparing results, using a baseline, which is obtained from a standard approach e.g. manual truncation;
- calculation of traditional evaluation measures i.e. precision and recall.

The superiority of the new information retrieval technique is justified if it achieves better results than the standard baseline.

The above mentioned approach to information retrieval evaluation is common, because it provides an objective means of evaluation and requires a minimum of experimental work, if relevant test collections are available. The summary of Chapter 3 also revealed that the comparison of traditional manual word truncation versus automatic word stemming is one of the generally used evaluation procedures for stemming algorithms (2). As the described methodology has been successfully applied for the analysis and evaluation of both English and non-English language e.g. Slovene (3) stemmers, it was also chosen as the basis for evaluation of the Latvian stemming algorithm.

Evaluation of the Latvian stemming algorithm is based on the following test collections:

- an electronic dictionary of Latvian nouns, adjectives, verbs and adverbs in their standard forms;
- text fragments from a Latvian full-text online database of legal documents, protocols, laws etc.;
- a set of user search enquiries to a Latvian online bibliographic database of articles from periodicals.

The first two collections were used to analyze morphological correctness of stemmed words both in standard forms and in declensions, whereas the latter was used to test the effectiveness of information retrieval using non-stemmed and stemmed forms of search terms. A detailed description of test collections is given in Section 7.3.

The next section (7.2) of this chapter outlines the general methodological framework which was applied to the analysis of the Latvian stemmer. It is followed (7.3) by justification of criteria used for the selection of relevant collections as well as characteristics of the chosen Latvian test collections. A separate section (7.4) deals with the results obtained from testing the Latvian stemmer using all three types of test collections. Finally, Section 7.5 evaluates the effectiveness of the Latvian stemmer in information retrieval based on recall and precision measurements.

7.2 General methodology

As outlined in Chapter 1, this thesis examines the following two hypotheses:

1. a suffix removal stemming algorithm based on the design of an English language environment can be applied for Latvian;
2. stemming in Latvian will produce the same or better information retrieval results as manual right hand truncation.

The general methodology applied for the evaluation of the designed stemming algorithm and for testing of the above mentioned hypotheses, consisted of the following components:

- selection of relevant test collections,
- testing and examination of the stemmer based on the chosen test collections,
- analysis and evaluation of the results obtained.

In order to test both the morphological accuracy of stemmed Latvian words and the effectiveness of information retrieval, three different test collections were selected:

- electronic dictionary of all basic classes of Latvian words ie. nouns, adjectives, verbs and adverbs,
- text fragments extracted from a full-text Latvian database of legal acts and documents,
- user search statements of a test collection consisting of 60 different queries and applied to the Analytical information system of periodicals database.

The grammatical correctness of stemmed Latvian words both in standard forms and in different declensions and conjugations was tested by the electronic dictionary and the text fragments. Fragments of Latvian texts were also used to examine the accuracy of stopword removal. Testing and analysis of information retrieval performance, including calculation of recall and precision ratio, were based on non-stemmed and stemmed words from a set of user search enquiries.

As noted before, performance of the Latvian stemming algorithm was examined in three different ways The first stage of evaluation, which assessed validity of remaining word stems and which was based on electronic dictionary of Latvian words in standard forms, was carried out as follows:

- four word classes e.g. nouns, verbs, adjectives and adverbs covered by the Latvian electronic dictionary were fed into and processed by the stemming algorithm;
- the obtained stems were browsed and, in case of incorrectness, appropriate modifications to the set of stemming rules were done;
- incorrectly stemmed words were repeatedly tested by the algorithm;
- remaining stems were assessed by Latvian experts in morphology and computing linguistics, and results of their observations as well as recommendations were filled in special evaluation forms;
- final modifications to the stemmer based on suggestions from Latvian linguists were carried out and incorrectly stemmed words were fed into and tested by the algorithm.

The next level of examination tested the correctness of stopword removal as well as the correctness of stems obtained from words in different declensions and conjugations. This part of the evaluation was based on Latvian text fragments and incorporated the following procedures:

- five fragments of Latvian texts were fed into and processed by the stemmer;
- the obtained stems were checked against the original texts both to examine the correctness of suffix and ending removal from Latvian words in different declensions and conjugations and to test the accuracy of stopword stripping;
- in the case of incorrect stopword removal and/or stemming, appropriate modifications to the stopword list and to the set of stemming rules were carried out;
- incorrectly stemmed words and/or unremoved stopwords were repeatedly tested by the modified stemming algorithm.

Finally, the impact of the Latvian stemmer on the information retrieval process was examined using a set of user search statements based on a Latvian online bibliographic database. The last stage of evaluation included the following methods:

- a relevant Latvian online database with comprehensive annotations and with a sufficient number of records was chosen;

- two independent users were selected to carry out a pilot test which included writing down in specifically designed forms five full search statements and then truncating them using right hand truncation. Both sets of full search enquiries were swapped between the users and manually truncated for comparision;
- full search statements were fed into and processed by the stemmer to produce stemmed search statements;
- nonstemmed, manually truncated and stemmed words from each search statement were listed down and compared with each other;
- document retrieval using the online Latvian database was carried out using nonstemmed, manually truncated and stemmed search statements gained from the pilot test;
- retrieved documents were browsed and evaluated in terms of their relevance;
- in the full test an additional ten independent users were selected and each of them created five full search statements which they manually truncated afterwards. All sets of full search queries were swapped between the users and manually truncated for comparison;
- nonstemmed user search queries were processed by the stemming algorithm;
- correlation between nonstemmed, truncated and stemmed words within the each search statement were listed down and compared;
- document retrieval using the online Latvian bibliographic database was carried out using full, manually truncated and stemmed user search queries;
- all retrieved documents were browsed and their relevance was evaluated;
- recall and precision measures based on nonstemmed, right hand truncated and stemmed user search statements were calculated;
- obtained results were analyzed and evaluated.

More detailed description of the selected Latvian test collections will be presented in the next section, which will be followed by the analysis and evaluation of results obtained from testing those collections by the developed stemming algorithm.

7.3 Test collections

As noted before in Section 7.2, the following Latvian test collections were selected in order to evaluate performance of the Latvian stemming algorithm:

- a electronic dictionary of all main Latvian classes of words;
- fragments of texts from a Latvian full-text database;
- Latvian online database for testing user search queries.

This section will justify the choice of test collections stated above as well as examine the Latvian databases which were selected for extracting words and text fragments, and for testing user search statements.

7.3.1 Electronic dictionary

The electronic dictionary of Latvian words was selected in order to test the performance of the stemming procedure for words in standard forms. The dictionary was created in 1990 by the Artificial Intelligence Laboratory of the Institute of Mathematics and Computing Science. A Latvian inverse dictionary (4) which was compiled using a Latvian-Russian bilingual dictionary, formed the basis of the electronic dictionary. Additional 6000 words were entered into the dictionary by linguistic computing specialists of the Institute.

The electronic dictionary covered more than 40,000 Latvian words which were stored into 15 separate files. All words in the dictionary were grouped into four classes i.e. Nouns, Adjectives, Verbs, Adverbs, and within the each class, words were arranged according to their endings and final suffixes. For example 25,500 nouns were classified into eight different files, where the first file included all nouns ending with iba, -ica,-nīca, -āda, -ēda, -āja, -ēja, -īja, etc. Similarly 10,000 verbs, 4000 adjectives and 800 adverbs were grouped correspondingly into four, two and one files.

Overall, the following criteria favoured selection of the electronic dictionary:

- it included all main classes of words;
- words covered broad range of subject areas;
- all words were in standard forms i.e. nouns, adjectives were listed in Nominative declension etc.;
- it was the only available dictionary of Latvian words in computerised form.

7.3.2 Full-text database

To examine the removal of stopwords and to test the stemming of words in different declension forms and conjugations, several fragments of Latvian texts were extracted from an online full-text database. The Latvian Legal Database (NAIS - Normatīvo Aktu Informācijas Sistēma) which is the largest online full-text database, was selected as a source for text fragments mentioned above.

The database was created in 1991 by Software House Riga, the biggest computer company in Latvia, and to date it covers more than 23,000 legal acts and documents

adopted by the Parliament and the Cabinet of Ministers of Latvia as well as international documents, protocols and agreements translated in Latvian. The Latvian Legal Database is a commercial database and updates are carried out on a weekly basis.

All legal acts and documents in the database can be accessed using following search parameters:

- topic/subject area,
- department which issued the document,
- type of document,
- date of issue,
- title of document,
- document contents.

The selected text fragments were mostly extracted from legal acts related to the Ministry of Foreign Affairs as well as from regulations dealing with transport registration, bank activities and the process of privatisation.

7.3.3 Online bibliographic databases

Information retrieval performance based on the Latvian stemming algorithm was evaluated using sets of search enquiries related to a Latvian online bibliographic database. In order to create relevant search statements and to carry out the evaluation study, the potential database had to meet following requirements:

- it should be online and accessible via Internet to retrieve documents and to evaluate them using unstemmed and stemmed search queries;
- the database should contain not only bibliographic information on documents, but also detailed annotations;
- contents should only be in Latvian;
- along with traditional information retrieval techniques, the database should offer a full and truncated keyword search in annotation (contents) field;
- there should be reasonable number of documents i.e. no less than 1000 records in the database.

Four bibliographic databases, which most closely matched the above mentioned features, were considered as the test database:

- Culture and Arts Information Database;
- Database on articles from periodicals published in Latvia;
- Database on books published in Latvia;
- Analytical Information System of Periodicals.

The **database on culture and arts information** was developed in 1988/1989 by the National Library of Latvia. It covers around 600 journal articles in Latvian, English, German and other languages. Documents in Latvian formed less than one sixth of the total amount of records. Along with document bibliographic description, each record was supplied with content descriptors (keywords) in Latvian and in a number of cases with Latvian annotations. Retrieval of records was based on traditional information search parameters e.g. author, title, keywords in contents field, journal title, year of publication. The database is off-line and available only on the site. The last update was carried out early in 1994.

A database on books published in Latvia and a database on articles from periodicals published in Latvia both were developed in 1992/1993 by the Institute of Bibliography at the National Library of Latvia. More than 3000 records are covered by the **database on Latvian books** and each record contains standard bibliographic information e.g. author, title, subject etc. as well as in several cases a brief description of contents. Although the majority of materials in the database are in Latvian, about 10% of documents are in other languages e.g. Russian, English, German. All records are arranged according to subjects and within the subject they are listed in alphabetical order.

The **database on articles from periodicals** contains more than 10,000 records which are arranged similarly to the database of books. Each article covers bibliographic information on author, title and source as well as a short annotation. The following example shows the structure of a record:

> **Krūmiņš, Māris.** Futbola sezona Eiropā tuvojas beigām // Nedēļa Tev.- 1995.- 12./18. jūn.- A18.

Both databases are updated daily however, to date there are no search facilities available in any of those databases and the access to records is provided only on the site.

More than 6000 articles from the biggest Latvian newspapers are covered by the **Analytical information system of periodicals** developed by the Lursoft company. The commercial database was created in 1994/1995 and it mostly contains materials related to politics, law and legislation, economics, financial and banking systems. Information of each record is presented in following fields:

- date of publication;
- title of periodical;
- title of article;
- author of article;
- number of page;

- text (annotation);
- main subject areas (maximum three).

Record retrieval can be carried out using one or combination of following search parameters:

- date of publication (range from ... to);
- title of periodical;
- title of article;
- author of article;
- full or truncated keywords from annotation;
- subject (s) from pop-up menu;
- Boolean AND, OR operators.

Retrieved documents can be viewed either as a list of titles or as full records. The searcher can also specify the maximum number of records to be retrieved. The following example shows the structure of a record in the database:

Publikācijas datums:	31.01.96
Izdevums:	DIENA
Virsraksts:	Kauls nevarot viens tikt galā ar lauksaimniecību
Autors:	Melnace Baiba
Lappuses nr.:	1
Teksts:	Pašreizējais zemkopības ministrs, Ministru prezidenta biedrs Alberts Kauls (Vienības partija) uzskata, ka Latvijas lauksaimniecības sfērā ir tik daudz darāmā, ka viņam vajadzētu kļūt par atbrīvoto Ministra prezidenta biedru, nodarboties ar lauksaimniecības jautājumiem, bet zemkopības ministra amatu ieņemt Kaula partijas biedram (Roberts Dilba). Ministru prezidents Andris Šķēle TV ziņu raidījumā "Panorāma" 30. Janvārī teica, ka valdības veidošanas laikā "tāds jautājums tika diskutēts, bet noraidīts".
Tēma nr. 1	VALDĪBA
Tēma nr.2	PERSONAS
Tēma nr. 3	LAUKI

The analytical information system of periodicals is available online and accessable on the Internet using following URL address:

<URL: http://www.lursoft.bkc.lv>

Although the Analytical information system described above contained certain drawbacks in its information retrieval mechanism e.g. incomplete implementation of standard Boolean AND and OR operators for truncated keywords in document annotation field, it was the database which most closely matched the majority of requirements for a test database listed before. Moreover, the analytical information system of periodicals was also recommended by information and computing specialists

in Latvia (5) who confirmed, that it was the only database which covered both basic specifications:

- coverage of documents in Latvian with detailed annotations;
- accessibility and information retrieval via Internet.

A description of query development based on the selected online bibliographic database of periodicals is presented in section 7.4.3 before the analysis of information retrieval performance.

7.4 Analysis of results

According to the general methodology outlined in Section 7.2, the evaluation of Latvian stemming algorithm involved three stages:

1) testing of standard Latvian word stemming based on the electronic dictionary;
2) testing of stopword removal and word stemming in different declensions using fragments of Latvian texts;
3) examination of the stemming procedure in information retrieval based on user search statements.

This section will discuss and analyse results obtained from the above mentioned evaluation studies.

7.4.1 Standard word stemming

The initial testing of the developed Latvian stemming algorithm was based on the electronic dictionary which covered all basic classes of Latvian words. For this purpose 15 files of Latvian nouns, adjectives, verbs and adverbs described in section 7.3.1 were fed into and processed by the stemmer. According to the first test results, on average words were stemmed correctly however, there were several obvious stemming errors which can be clustered into the following groups:

- overstemming in short words leaving only one or two letters e.g. gars (a spirit) and g after removing ending -s and suffix -ar or upe (a river) and up after removing -e;
- incorrect removal of endings, especially in verbs e.g. liet instead of lieto in a verb lietot (to use);
- incorrect suffix removal leaving only a root instead of a stem e.g. kanād instead of kanādiet in a word kanādiete (Canadian);

- understemming in long nouns especially in compound words e.g. elektroenerģij instead of elektroenerģ in a word elektroenerģija (electrical energy);

In order to correct the above listed stemming errors, a number of modifications in suffix and ending removal rules related to the length of resulting stem were carried out. For example, the length of a word containing ending -s (rule 104 in a static RuleList step 1a4) was changed from -1 to 0 which means that the ending -s will be removed only if the word contains at least three characters. Similar modifications were done in rule 427 (static RuleList step4) where to avoid understemming, the length of a word with the suffix -ij was changed from 2 to 1.

All Latvian words were repeatedly processed by the modified stemming algorithm and resulted stems were printed out. Three independent Latvian experts in computing and linguistics were selected in order to evaluate the printed version of all stemmed Latvian words:

- Anna Auliņa, linguist, Head of the Research Department at the National Library of Latvia;
- Dr. Sarma Kļaviņa, expert in computing linguistics and senior lecturer at the Department of Baltic languages, University of Latvia;
- Dr. Viktorija Drīzule, expert in computing linguistics and senior lecturer at the Language Centre, University of Latvia.

The evaluation results in ranked order for each separate class of Latvian words together with suggestions and comments were presented in special evaluation form (See Appendix 4.1). Table 7.1 displays the results of Latvian word stemming ranked by each respondent including average evaluation mark for all Latvian nouns, adjectives, verbs and adverbs.

Table 7.1 Ranked evaluation results of stemmed Latvian words

	Evaluator #1	Evaluator #2	Evaluator #3	Average
Nouns	4	3	4	3.6
Adjectives	4	4	4	4
Verbs	4	3	4	3.6
Adverbs	4	4	3	3.6

where 5 = excellent, 4 = good, 3 = average, 2 = poor, 1 = very poor

According to the comments pointed out in evaluation forms, the majority of Latvian nouns were stemmed correctly and the resulting stems were acceptable. However, the following three problems were revealed by the Latvian linguistic specialists:

1) inconsistency in suffix removal from nouns which contain suffix-īca and/or -nīca e.g. a word kafejnīca (a cafe)was stemmed as kafej but tējnīca (a tea house) as tējn;
2) misinterpretation of the meaning a root and a stem, which for some nouns caused word overstemming, leaving only the root of a word instead of its stem e.g. a word ieskrambājums (a scratch) was stemmed as ieskramb instead of ieskrambāj;
3) removal of ending o from short words leaving a stem which is common for a number of other different nouns e.g. the stem from a word auto (a car) is aut which is also common for autors (an author), autentisks (authentic), autorizēts (authorised) etc.

All three respondents evaluated the stemming of Latvian adjectives as adequate and the only recommendation regarding suffix removal from this class of words was as follows:

- not to remove suffixes -āb and -īb as they are not typical for Latvian language and very few words contain this type of suffixes.

The following comments were outlined in terms of suffix removal from Latvian verbs:

- inaccurate stemming of reflexive verbs with morphemes -ie and -s e.g. the verb karāties (to hang) was stemmed as karāt whereas the correct form would be karā;
- few inconsistences in stemming verbs which contain vowel and/or consonant palatalisation in the stem e.g. a verb sapņot (to dream) have to be stemmed as sapņo instead of sapņ;
- it was recommended to include some secondary forms of modal verbs into the stoplist e.g. pabūt (to be), nokļūt (to reach).

Majority of Latvian adverbs were stemmed correctly and both following suggestions related only to the extension Latvian stoplist:

1) several meaningless adverbs should be added to the list of Latvian stopwords e.g. nākamreiz (next time), beidzot (finally);

2) only those adverbs which were generated from adjectives and which have comparative degrees, should be stemmed e.g. klusītiņām (silently), pakāpeniski (gradually).

In general, all three experts evaluated the stemming of standard Latvian words as adequate, correct and essential both for Latvian morphology and information retrieval purposes. In order to explain problems and suggestions more in detail, each respondent separately agreed to participate in a half an hour interview, based on questions from evaluation forms.

Results from evaluation forms and interviews were analysed and following modifications to the Latvian stemming algorithm were carried out:

- all suffixes from the static RuleList step2, step3 and step4 were repeatedly checked and appropriate corrections regarding the length of suffixes were made. For example, in order to avoid inconsistency in stemming words with suffix -īc and -nīc, the length of suffix -īc (Rule 420) was changed from 2 to 1, but the length of suffix -nīc (Rule 305) from 1 to 2. In this case both incorrectly stemmed words kafejnīca and tējnīca described before will be stemmed as kafej and tēj;
- several modifications concerning the length of endings were done in the static RuleList step1a1 to step1a6. For example, to ensure that only those words with the ending -o will be stemmed, which contain four or more characters, the length of -o in the static RuleList step1a6 Rule103 was changed from -1 to 0. Similarly, in order to stem correctly reflexive verbs, the length of -ie in the static RuleList step1a4 was changed from 0 to 1;
- according to suggestions, more than 180 meaningless adverbs were added to the list of Latvian stopwords;
- no additional verbs were shortlisted for the stoplist because secondary forms of modal verbs are different and contain certain meaning which can be useful for information retrieval.

Finally, both incorrectly and correctly stemmed words were repeatedly processed by the modified stemming algorithm and the relevance of resulting stems was evaluated. Examples of unstemmed and stemmed forms of Latvian nouns, adjectives, verbs and adverbs are presented in Appendix 4.2.

7.4.2 Word stemming in a text corpus

As described before, the first stage of testing the performance of the Latvian stemming algorithm was based on Latvian words only in their standard forms. However, along with standard forms, words in Latvian texts normally are used in various declensions including consonant palatalisations etc. During the initial stage of evaluation, it was mentioned by Latvian linguistic experts that the stemmer testing should involve not only words from the electronic dictionary, but also Latvian words in a text corpus (6).

Therefore, in order to evaluate stemming of Latvian words in different declensions and conjugations as well as to test removal of stopwords, five short fragments of Latvian texts were selected. As noted in Section 7.3.2, all text fragments were extracted from the full-text online Latvian database of legal documents. The selected test sample contained 2071 words and covered fragments from legislative documents and regulations on the following topics:

- property privatisation;
- handling and transportation of hazardous goods;
- bank exchange rates;
- customs taxes on motor vehicles;
- Ministry of Foreign Affairs.

All text fragments were stored in a separate file which was processed by the stemming algorithm. In total 1574 words were stemmed and 497 were removed during the stemming procedure. Overall, all stopwords including adverbs and pronouns e.g. kurā, ārpus, to were stripped correctly and no unremoved stopwords were identified in the remaining text corpus. Because of the lexical analyser which was built into the Latvian stemmer and which was described in Section 5.1, all non-alphabetic characters e.g. numbers and punctuation marks (hyphens, semicolons etc.) were automatically removed from text fragments.

The majority of Latvian words which were in different declensions and conjugations, were stemmed correctly and contained a valid resulting stem. Understemming of some words was the only drawback which was revealed during the analysis of the stemmed test sample. For example, a word starptautiskajā (in international) was stemmed as starptautiskaj, by removing only the ending -ā, whereas the relevant stem would be starptautisk, by removing also the suffix -aj. After completing modifications to the stemming algorithm e.g. adding rule 438 to the static RuleList step4 which determines

removal of suffix **-aj**, the whole file of test sample was repeatedly tested and analysed. The following example comparatively characterises an unstemmed and stemmed fragment of Latvian text:

Ārlietu ministrija ir valsts pārvaldes iestāde, kura izstrādā un realizē Latvijas Republikas ārpolitiku, piedalās Latvijas ārējās ekonomiskās politikas veidošanā un kuras uzdevums ir ar politiskiem un diplomātiskiem līdzekļiem nodrošināt Latvijas Republikas ārpolitikas koncepcijas realizēšanu, lai radītu maksimāli labvēlīgus apstākļus Latvijas Republikas starptautiskajai drošībai un iekšpolitiskajai stabilitātei, kā arī sekmētu Latvijas Republikas iekļaušanos starptautiskajā apritē. Ārlietu ministrija darbojas saskaņā ar Latvijas Republikas Satversmi, spēkā esošajiem likumdošanas aktiem un šo nolikumu. Savā darbībā Ārlietu ministrija ir tiešl pakļauta Ministru kabinetam.

ārliet* ministr* valst* pārvald* iestād* izstrād* realiz* latv* republik* ārpolit* piedal* latv* ārēj* ekonom* politik* veid* uzdevum* polit* diplom* līdzek* nodrošin* latv* republik* ārpolit* koncepc* realizē* radīt* maksimāl* labvēl* apstāk* latv* republik* starptautisk* drošīb* iekšpolitisk* stabilit* sekmēt* latv* republik* iekļaušan* starptautisk* aprit* ārliet* ministr* darboj* saskaņ* latv* republik* satversm* spēk* esošaj* likumd* akt* nolikum* darbīb* ārliet* ministr* tieš* pakļaut* ministr* kabin*

7.4.3 Stemmer performance in information retrieval

The final stage of evaluation tested effectiveness of the designed stemming algorithm in information retrieval. For this purpose, 60 search queries based on the Latvian online bibliographic database described in Section 7.3.3, were created by 12 independent users. As noted by Tague (7), a query is the statement of user's requirement which is usually expressed in a form of short question or statement. It was also mentioned that a user must be aware about the content and scope of a particular database as well as the search statement must be correctly definied and formulated (8).

The selection of respondents for query development was based on the following criteria:

- experience in information retrieval;
- availability and willingless;
- job title;
- type of organisation / institution.

The last two factors were chosen to ensure the variety of participants in terms of their professional background and job title as well as to show the possible impact of those factors in creating search queries. Following table (Table 7.2) summarises the basic background data of all 12 respondents.

Table 7.2 Basic information on participants

Code of respondent	Job title	Organisation	Years of experience
1	Engineer - programmer	Latvian Academic Library	7
2	Lecturer	University of Latvia	5
3	Programmer & information specialist	Company ARCIS	22
4	Head of Laboratory	Institute of Maths and Computing	34
5	Head of Library	Bank of Latvia	23
6	Information specialist	Centre of the US Information Services	12
7	Head of Department	Institute of Bibliography	18
8	Internet administrator	Parliament of the Republic of Latvia	10
9	System analyst & project manager	Company ALISE IS Ltd.	4
10	Head of Department	University of Latvia	20
11	Undergraduate student	University of Latvia	3
12	Senior bibliographer	University of Latvia Library	18.5

According to Table 7.2 the majority of respondents (67%) had 10 and more years of experience in information retrieval and only two participants were less than five years involved in information searching. The present work and or job position of all participants to a certain extent was related to information management and information technology however, only a half of them had relevant educational background i.e. in information and library science.

All respondents before they generated search statements, were familiarised with the main characteristics of the test database including scope of materials, subject areas, sources of information. When it was possible to access the database, users were able to browse a brief demo version. As outlined in the general methodology (Section 7.2) each user had to create five different search statements and then manually conflate them using right hand truncation. Five full search queries where photocopied and swapped with the next respondent for another manual truncation. It meant that the second participant truncated first five search statements, the third processed the second set of enquiries etc., but the first user manually truncated the last five full search queries. The background information of each respondent e.g. name, organisation, job position etc. was coded and separated from the search statements as well as it was confidential for the rest of users. All search enquiries were filled in a structured form which is presented in Appendix 4.3.

Before presenting detailed analysis of each separate set of search statements, Table 7.3 lists the main quantitative parameters of all search queries.

Table 7.3 Quantitative characteristics of unstemmed and stemmed search queries

Quantitative parameters	Before stemming	After stemming
Total number of search queries	60	60
Total number of words in queries	300	245
Median number of words per query	5	4
Maximum number of words in a query	10	7
Minimum number of words in a query	3	2

According to Table 7.3, the longest search statement before stemming contained 10 words whereas the shortest included only three words. After the stopword removal the maximum and minimum number of words in a query was respectively seven and two. Average number of words per unstemmed search statement was five, but the median number of words per stemmed query was four.

The first two sets of ten search statements generated by respondent 1 and 2 were used as the pilot test. The following five search statements covered the first set of enquiries:

1) Dānijas - Latvijas universitāšu sakari (Relations between universities in Denmark and Latvia);
2) Baltijas valstu robežu strīdi (Sea border disputes between the Baltic States);
3) Jaunās 100 dolāru naudas zīmes (New 100 dollar notes);
4) Eiropas Kopienas materiāli par Latviju (European Community materials on Latvia);
5) Jaunumi Disneja filmu studijā (New releases from Disney Film Studio).

The manual truncation and stemming which was applied to the above listed search statements, modified those enquiries as follows:

1) Dānij* Latvij* univers* sakar* (T1)
   Dān* Latvij* universit* sakar* (T2)
   dān* latv* universitā* sakar* (S)

2) Balt* robež* strīd* (T1)
   Balt* valst* robež* strīd* (T2)
   balt* valst* robež* strīd* (S)

3) Jaun* dolār* naud* (T1)
   Jaun* 100 dolār naudaszīm* (T2)
   jaun* dolār naudaszīm* (S)

4) Eirop* Kopien* materiāl* latv* (T1)
   Eirop* Kopien* materiāl* Latvij* (T2)
   eirop* kopien* materiāl* latvij* (S)

Key:	
T1	Truncated by the first user
T2	Truncated by the second user
S	Stemmed by the algorithm

5) Jaun* Disnej* film* (T1)
Disnej* film* studij* (T2)
jaunum* disnej* film* studij* (S)

Both manual truncation and the stemming algorithm removed stopword **par** from the forth search statement. Because of the built-in lexical analyser, the stemmer also stripped number 100 from the third search enquiry, which can be a reason for retrieval of non relevant documents.

The second set of search statements also included five different queries:

1) Par Latvijas banku stāvokli 1996.g. martā (On the situation of Latvian banks in March 1996);
2) Par Latvijas un Igaunijas līgumiem zvejniecībā (On Latvian and Estonian fishing agreements);
3) Par informācijas aizsardzības likumu Latvijā (On the Law of information protection in Latvia);
4) Par deputātu ienākumu deklarācijām (On the declarations of MP's salaries);
5) Par pašvaldību finansiālo situāciju (On the financial situation of local governments).

As for the first set of queries all five full search statements were manually truncated and stemmed by the algorithm as follows:

1) Latv* bank* 1996. g. mart* (T1)
Latvij* bank* 1996. mart* (T2)
latv* bank* stāavokl* g mart* (S)

2) Latv* Igaun* līgum* zvej* (T1)
Latvij* Igaunij* līgum* zvejniec* (T2)
latv* igaun* līgum* zvejniec* (S)

3) inform* aizsardz* likum* Latvij* (T1)
Latvij* informāc* aizsardzīb (T2)
informāc* aizsardz* likum* latvij* (S)

4) deputāt* ienākum* deklar* (T1)
deputāt ienākum* deklarāc* (T2)
deput* ienākum* deklarāc* (S)

5) pašvald* finans* (T1)
pašvaldīb* finans* situācij* (T2)
pašvald* finansiāl* situāc* (S)

All five search queries contained only one stopword **par** which was removed both by manual truncation and by stemming. In two search statements respondents decided to exclude two meaningless nouns e.g. in query #3 the word **likumu** (law) was removed but the word **situāciju** (situation) was deleted from the query #5. The lexical analyser of the stemmer automatically removed a year 1996 from the first query.

After the truncation and stemming of all search statements was completed, both full and conflated versions of queries were used to search for relevant documents in the Latvian bibliographic database of periodicals. Tables 7.4A and 7.4B comparatively present the total number of documents retrieved by each search query as well as the number of relevant documents per each type of search enquiry

Table 7.4A Number of documents retrieved by the first set of queries

Query	Full		Truncated (1st user)		Truncated(2nd user)		Stemmed	
	Total	Relevant	Total	Relevant	Total	Relevant	Total	Relevant
# 1	0	0	0	0	0	0	0	0
# 2	50	1	50	1	50	2	50	2
# 3	0	0	2	0	2	0	2	0
# 4	3	1	3	1	3	1	3	1
# 5	0	0	0	0	0	0	0	0

Table 7.4B Number of documents retrieved by the second set of queries

Query	Full		Truncated (1st user)		Truncated(2nd user)		Stemmed	
	Total	Relevant	Total	Relevant	Total	Relevant	Total	Relevant
# 1	11	1	50	9	50	9	50	9
# 2	9	0	44	2	11	1	11	1
# 3	0	0	2	2	2	2	2	2
# 4	0	0	1	1	1	1	1	1
# 5	1	1	3	2	3	2	2	2

According to Table 7.4A full, truncated and stemmed version of two queries (#1 and #5) in the first set of search statements matched zero hits whereas manually truncated query #3 retrieved two documents which were both irrelevant. Unsuccessful retrieval of information based on the above mentioned search statements can be justified because of the irrelevance of requests to the content of a database e.g. the Analytical system of Latvian periodicals do not cover any materials on films and videos (Query #5) as well as there being no documents on international links related to higher educational institutions (Query #1). Limited access to the database content was the basic reason of irrelevant search statements. Because of the very specific requirement e.g. information on 100 dollar notes, no relevant documents were matched by Query #3. The broadness

of the topic covered by search statement #2 was one of the main reasons why so many records dealing with Baltic sea borders in general were retrieved. Moreover, as the second user removed a word valstu (countries' in possessive case) from the above mentioned truncated query, the total amount of documents retrieved reached 44 which included all records not only on the Baltic countries, but also on the Baltic sea.

Analysis of results obtained from the second set of queries (Table 7.4B) revealed that the majority of documents retrieved by full, truncated and stemmed versions were relevant and matched initial search requests. A large number of non relevant documents were retrieved by manually truncated and stemmed query #1 because of the ambiguous formulation of the search request e.g. The truncated version of Latvijas banku covered not only all documents related to Latvian banks in general, but also records containing information on Bank of Latvia. Overall, the pilot test confirmed that full, truncated and stemmed search statements could be used with the selected Latvian online bibliographic database in order to retrieve relevant documents. The initial test also revealed that there is no significant difference between documents retrieved by the different manual truncations of the source query as well as between the truncated and stemmed search statements. Table 7.5 shows the median number of retrieved relevant and irrelevant documents based on all queries used for the pilot test, except of those which matched zero records. Statistical measures characterising the effectiveness of the stemming algorithm i.e recall and precision were calculated for the whole set of queries including those used in the pilot study and will be presented in Section 7.5.

Table 7.5 Median of documents retrieved in the pilot test

Type of query	Median of retrieved documents in total	Median of relevant documents
Full	10	1
Truncated by the 1st user	22	3
Truncated by the 2nd user	17	3
Stemmed	17	3

Analysis of the remaining set of search queries generated by ten different respondents followed the same principle which was applied in the pilot study. The basic subject areas covered by 50 search statements are summarised in Table 7.6.

Table 7.6 Main subject areas covered by queries

Subject area	Number of queries
Finances / banks /exchange rates	10
Fishing and disputs on sea borders	6
Foreign investments	3
Latvian army	4
Citizenship	3
International economical cooperation	4
Criminal situation in Latvia	3
Information systems, libraries, culture	5
Education	2
European Union / politics in general	2
Elections of Parliament	3
Small enterprises	1
Local governments	1
Miscellaneous	3

As shown in Table 7.6, the majority of search statements were dealing with banks, finances, economicy, politics, defence and law which were relevant topics and corresponded to materials covered by the database. The number of queries related to fishing and sea borders was high, because at the time of generating search statements i.e. March, April 1996 the topic was covered by the mass media e.g. disputs on sea borders between Latvian and Estonia were discussed on TV news and presented in periodicals. All unstemmed queries as well as manually truncated and automatically stemmed search statements are presented in Appendix 4.4.

In order to discuss and analyze information retrieval results, Table 7.7 comparatively presents the total number of documents retrieved and the number of relevant documents retrieved per query. All queries are grouped in the same order as they were listed in Appendix 4.4. Search statements which resulted into zero hits or which retrieved no relevant records, were not included in the table.

Table 7.7 Number of documents retrieved per each query

# of Query	Unstemmed		Truncated (1)		Truncated (2)		Stemmed	
	Total	Relevant	Total	Relevant	Total	Relevant	Total	Relevant
1	0	0	1	1	1	1	1	1
2	0	0	1	1	1	1	1	1
3	9	0	36	0	36	1	11	2
4	0	0	1	1	1	1	1	1
5	5	5	6	6	6	6	6	6
6	0	0	3	1	3	1	3	1
7	4	3	6	4	5	4	6	4
11	0	0	4	3	4	3	4	3
12	7	1	10	1	10	1	10	1
13	0	0	3	3	3	3	3	3
16	24	5	24	5	27	5	27	5
17	21	12	50	14	50	14	25	14
18	1	1	7	5	7	5	7	5
19	9	1	9	1	9	1	9	1
21	4	4	6	6	6	6	6	6
22	6	6	50	43	50	43	45	43
23	7	1	14	1	9	1	13	1
24	4	3	29	11	27	10	27	10
25	2	2	2	2	2	2	10	2
26	0	0	1	1	1	1	1	1
27	1	1	1	1	45	1	1	1
31	6	2	19	6	19	6	19	6
32	2	2	2	2	4	4	4	4
33	23	4	28	6	50	0	30	0
34	3	2	28	0	2	1	3	2
35	0	0	25	1	1	1	1	1
37	0	0	3	2	3	2	3	2
38	5	2	13	3	13	3	13	3
40	2	2	5	3	4	3	4	3
42	5	2	0	0	13	4	13	4
43	1	1	1	1	1	1	1	1
47	2	1	2	1	2	1	2	1
49	2	2	6	4	6	4	4	4
50	1	1	1	1	1	1	1	1

Assessment of document relevance was judged by the researcher on the basis of information obtained from the title and annotation of each retrieved document. The reliability of this approach for determing the total amount of relevant records, which is one of the values for calculating recall ratio, was reasonably high (about 90%) because of the managable amount of records at the time (i.e. all 800-1000 records was possible to browse). All retrieved documents to be judged for relevance consisted of the pooled output of four different types of search e.g. unstemmed, truncated by the first user, truncated by the second user and stemmed by the algorithm. The following two reasons determined the use of this particular principle for document relevance assessment:

- geographical location e.g. all respondents who generated search queries were residing in Latvia and for document evaluation purposes only some of them could be remotely accessable via e-mail or Internet;
- security system and restrictions of the database e.g. the Analytical information system of periodicals is a commercial database and only authorised users who have paid subscription fees are provided with full access to the database. The database security system allows the printing of a certain number of retrieved documents therefore, it would be impossible to provide respondents with a complete set of retrieved documents in a printed form.

According to Table 7.7 the majority of full, truncated and stemmed search queries matched certain amount of records in the database. In 22 cases there were either minor or no differences between number of documents retrieved in total and number of relevant documents. For example, six relevant documents out of six were retrieved using both manually truncated and stemmed versions of Query #5:

Par Baltijas banku (On the Bank Baltija)

Similar results were achieved with full, truncated and stemmed Query #43:

Zivju pārstrādes uzņēmumi Latvijā (Fish processing enterprises in Latvia).

Several search statements were not clearly defined and contained some ambiguous words which resulted in retrieving a high number of documents in total, but a low number of relevant documents. For example, one relevant record out of nine was retrieved by the Query #19:

Skandāls ar "Parex" bankas garantijām (Scandal on the guaranties of Parex bank).

because of the multiple meaning of a word garantijām.

Retrieval of irrelevant documents was also caused by the overtruncation and/or overstemming of words in a search statement as well as because of removing meaningful words or numbers from the query.

For example, 24 irrelevant records and only one relevant record matched truncated Query #35:

> Valsts un privātie monopoluzņēmumi Latvijā (State and private monopoly enterprises in Latvia)

because the word monopoluzņēmumi was manually overconflated as monopol* which resulted into retrieval of documents related to the state policy on alcohol monopoly.

Another example shows the automatic removal of the meaningful number 21 from the stemmed Query #25:

> Par "Klubs - 21" dalībnieku iesaistīšanos politiskajā darbībā (On the involvment of "Club - 21" into political activities)

resulted into retrieval of eight irrelevant and two relevant records whereas manually truncated and unstemmed form of this query matched two relevant documents out of two in the total.

Several unstemmed queries hit zero records because they contained words in different declensions which did not match appropriate words in the documents' contents (annotation) field. For example, Query #13:

> Klīrings un citas norēķinu sistēmas (Clearing and other systems of payment)

matched zero records because there were no documents containing a word norēķinu in Accusative declension.

Sixteen unstemmed, manually truncated and stemmed search statements retrieved zero documents as they were defined incorrectly and/or because they covered a subject area not relevant to the database content. The main reason for incorrect search queries was lack of detailed information on topics covered by the information system. For example, completely irrelevant requirement was formulated in Query #30:

> Rītdienas laika prognoze (Weather broadcast for tomorrow).

Similar inadequate search statement was definied in Query #41:

> Izstāde "Datortehnika '96" (Exhibition "Computer technology '96")

The median number of relevant documents and the median number of documents in total retrieved by unstemmed, truncated and stemmed queries are presented in Table 7.8. Queries which retrieved zero documents or which retrieved no relevant documents where not included in calculations.

Table 7.8 Median number of documents retrieved in the main test

Type of query	Median of retrieved documents in total	Median of relevant documents
Full	5	2
Truncated by the 1st user	12	4
Truncated by the 2nd user	12	4
Stemmed	9	4

According to Table 7.8 there is no difference between the total number of documents retrieved by manually truncated queries carried out by the first and by the second respondent. The median number of relevant documents both using truncated and stemmed queries was the same. Stemmed search statements produced slightly less irrelevant records than manually truncated queries. Because of the search words which were used in different declensions, the median number of records retrieved by full queries was relatively small. The following section (Section 7.5) will evaluate the effectiveness of the Latvian stemming algorithm in information retrieval using recall and precision measurements.

7.5. Effectiveness of the stemming algorithm

The effectiveness of a stemming algorithm in information retrieval is usually calculated using two standard measures:

- recall,
- precision (10).

Recall value shows the ratio between the number of relevant documents retrieved per query and the total number of relevant records in the collection e.g.

$$\text{Recall} = \frac{\text{Number of relevant records}}{\text{Total number of relevant records}}$$

Precision measure can be calculated as the proportion between relevant records retrieved per query and total number of records retrieved per query e.g.

$$\text{Precision} = \frac{\text{Number of relevant records}}{\text{Total number of records}}$$

As mentioned in the general methodology (Section 7.2) the effectiveness of the Latvian stemming algorithm in information retrieval based on full, manually truncated and stemmed search queries will be also evaluated using standard recall and precision values. Table 7.9 summarises recall and precision measures for all queries used in the main test except those which retrieved zero records.

Table 7.9 Recall and precision values based on queries from the main test

# of Query	Unstemmed		Truncated (1)		Truncated (2)		Stemmed	
	R	P	R	P	R	P	R	P
1	0	0	1	1	1	1	1	1
2	0	0	1	1	1	1	1	1
3	0	0	0	0	0.5	0.03	1	0.18
4	0	0	1	1	1	1	1	1
5	0.83	1	1	1	1	1	1	1
6	0	0	1	0.33	1	0.33	1	0.33
7	0.75	0.75	1	0.66	1	0.8	1	0.66
11	0	0	1	0.75	1	0.75	1	0.75
12	1	0.14	1	0.1	1	0.1	1	0.1
13	0	0	1	1	1	1	1	1
16	1	0.20	1	0.20	1	0.19	1	0.19
17	0.86	0.57	1	0.28	1	0.28	1	0.56
18	0.2	1	1	0.71	1	0.71	1	0.71
19	1	0.11	1	0.11	1	0.11	1	0.11
21	0.66	1	1	1	1	1	1	1
22	0.14	1	1	0.86	1	0.86	1	0.96
23	1	0.14	1	0.07	1	0.11	1	0.08
24	0.27	0.75	1	0.38	0.90	0.37	0.90	0.37
25	1	1	1	1	1	1	1	0.2
26	0	0	1	1	1	1	1	1
27	1	1	1	1	1	0.2	1	1
31	0.33	0.33	1	0.32	1	0.32	1	0.32
32	0.5	1	0.5	1	1	1	1	1
33	0.66	0.17	1	0.21	0	0	0	0
34	1	0.66	0	0	0.5	0.5	1	0.66
35	0	0	1	0.04	1	1	1	1
37	0	0	1	0.66	1	0.66	1	0.66
38	0.66	0.4	1	0.23	1	0.23	1	0.23
40	0.66	1	1	0.6	1	0.75	1	0.75
42	0.5	0.4	0	0	1	0.31	1	0.31
43	1	1	1	1	1	1	1	1
47	1	0.5	1	0.5	1	0.5	1	0.5
49	0.5	1	1	0.66	1	0.66	1	1
50	1	1	1	1	1	1	1	1

R = recall P = precision

According to Table 7.9, recall and precision values obtained from both manually truncated search queries are almost equal. There is a relatively small difference between recall and precision measures using truncated and stemmed queries. However, comparing both information retrieval performance ratios obtained from unstemmed queries and truncated/stemmed search statements is significant. The main reason of such difference is the use of search words in different declensions which matched a very small number (if any) of records.

In order to show and to evaluate the overall information retrieval performance results, Table 7.10 presents the median number of recall and precision values based on full, truncated and stemmed search queries which were used in the main test.

Table 7.10 Median of recall and precision ratios for all types of queries

Query	Recall	Precision
Full	0.53	0.47
Truncated by the 1st user	0.90	0.58
Truncated by the 2nd user	0.94	0.61
Stemmed	0.97	0.64

Table 7.10 confirms that truncated/stemmed queries in Latvian language produce significantly better recall and precision results in information retrieval than use of unstemmed search statements e.g. full queries retrieved only 53% (0.53) relevant records whereas stemmed search statements matched more than 90% of all relevant records. The table also reveals that there is no major difference in information retrieval performance using stemmed or truncated queries. However, stemmed search statements were more efficient and produced slightly better recall and precision values than manually truncated queries. Figure 7.1 graphically shows the performance of full, truncated and stemmed queries in information retrieval.

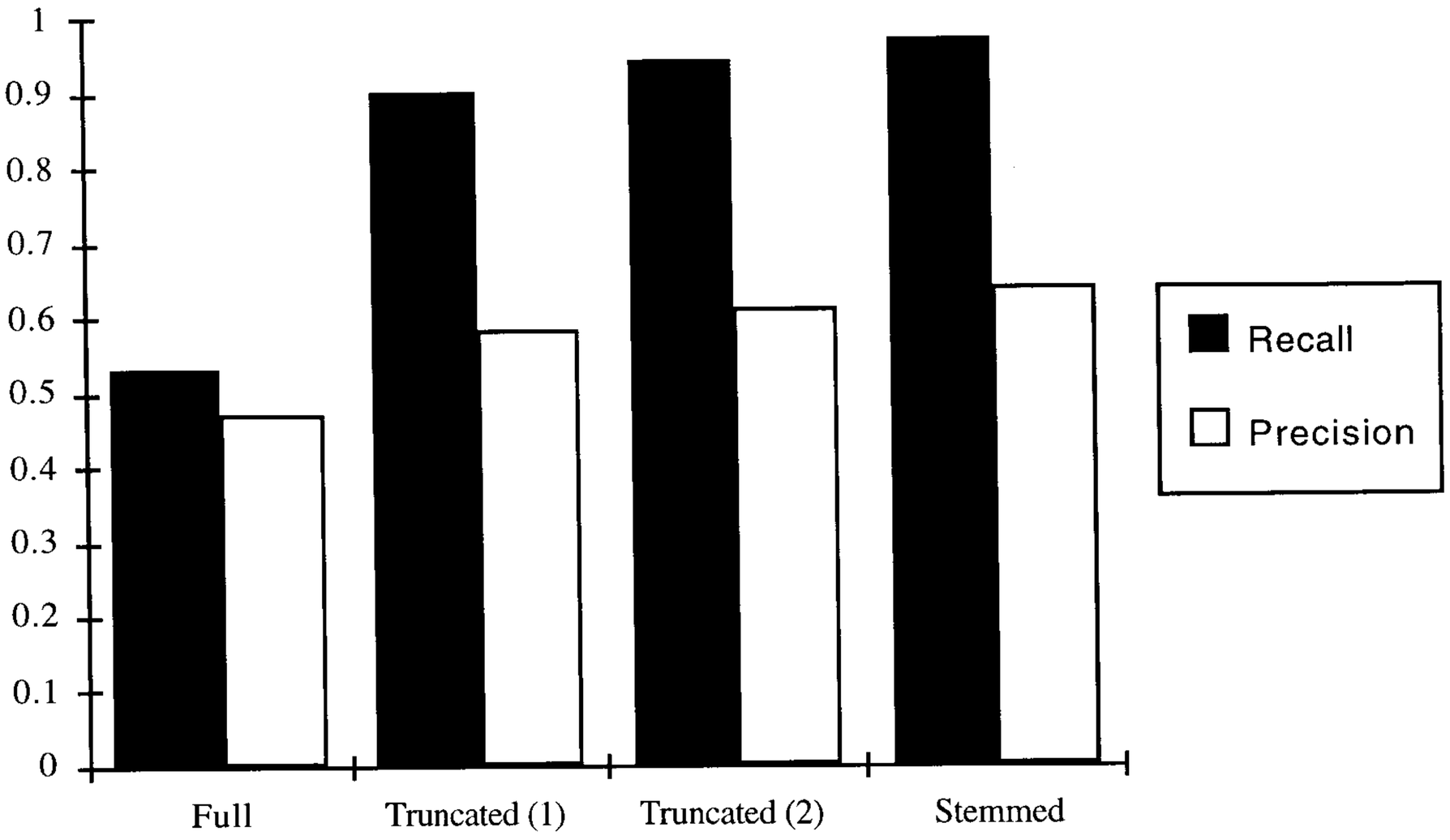

Figure 7.1 Recall and precision of full, truncated and stemmed queries

Recall and precision measures in both Table 7.10 and Figure 7.1 exclude queries which matched zero records in the database. Table 7.10 presents the median recall and precision measures of document retrieval using all 50 full, truncated and stemmed queries.

Table 7.11 Median recall and precision ratios of all queries used in the main test

Query	Recall	Precision
Full	0.30	0.27
Truncated by the 1st user	0.51	0.33
Truncated by the 2nd user	0.53	0.34
Stemmed	0.55	0.36

7.6 Conclusion

The main objective of this evaluation study was to test general performance of the Latvian stemming algorithm using both Latvian words in their standard forms and words in declensions as well as to analyse and evaluate the effectiveness of the Latvian stemmer in information retrieval based on user search statements.

The evaluation results confirmed both hypothesis outlined in the general methodology:

- a stemming algorithm designed for English language can be applied for Latvian;
- use of stemming in information retrieval performs the same or better results than manual right hand truncation.

Initial testing of Latvian words from the electronic dictionary revealed that the stemming algorithm correctly and in an appropriate way removes suffixes from standard Latvian words. Because of the language complexity some rules in the stemmer were changed allowing to remove all suffixes and endings from a word leaving instead of the stem only the root of a word.

Results obtained from the testing of Latvian text fragments showed that the algorithm produces relevant stems of Latvian words used in different declensions and conjugations. Information retrieval performance results based on recall and precision measures confirmed that truncated and stemmed queries can be more efficient and can retrieve significantly more relevant documents than unstemmed search statements.

Because of the time limits and due to some technical problems e.g. the lack of appropriate software package for linking and hardware, the Latvian stemming algorithm over the period of evaluation was not been implemented in any of existing Latvian databases. However, evaluation results of information retrieval effectiveness revealed that queries stemmed by the Latvian stemming algorithm produced a better ratio of relevant documents than manually truncated search statements.

References

1. **Hull, David A.** Stemming algorithms: a case study for detailed evaluation. *Journal of the American Society for Information Science,* 1996, **47** (1), 70-72.

2. **Frakes W.B.** Stemming algorithms. In: William B. Frakes & Ricardo Baeza-Yates, eds. *Information retrieval: data structures and algorithms*, 1992, pp. 143-147.

3. **Popovic, Mirko.** *Implementation of a Slovene language based free-text retrieval system,* 1991.

4. **Soida S. & S. Kļaviņa.** *Latviešu valodas inversā vārdnīca* [Inverse dictionary of Latvian language], 1970.

5. Ivars Indans to Karlis Kreslins, 5 February 1996.

6. Interview with Sarma Klavina, Department of Baltic languages, University of Latvia, Riga, Latvia, 4 April 1996.

7. **Tague J.M.** The pragmatics of information retrieval experimentation. In: K. Spark-Jones, ed. *Information retrieval experiment,* 1981, pp. 59-102.

8. *Ibid.*

9. **Hull,** ref. 1.

Chapter 8

CONCLUSION

8.1 Summary of results

As indicated in Chapter 1, the aim of this thesis was to introduce automatic word conflation for Latvian in order to improve the effectiveness of access to Latvian databases and information systems. The requirement for design and implementation of advanced information retrieval techniques (i.e. stemming) is determined by two factors:

- the anticipated growth in the number of Latvian online bibliographic, full-text and other types of databases;
- increasing user demands for relevant and easy retrievable information.

The Latvian stemming algorithm which is based on the English stemmer can be also used to provide access to international databases (e.g. EU and/or NATO databases), therefore allowing users to formulate their search queries in a native language and providing document retrieval in English.

To date the retrieval software for Latvian databases and information systems is based on traditional Boolean search operators, which pose problems for unexperienced users e.g. in formulating correct search queries. Implementation and use of the Latvian stemmer as the front-end of different databases to a great extent can help end-users in the process of information searching and retrieval e.g. it will allow them to formulate a search query in the natural language without any assistance from trained intermediaries.

Comparative analysis of the English and Latvian languages and their grammatical structures revealed that words in both languages are created by adding suffixes to a basic stem. This statement justified the selection of an English algorithm as the basis for the Latvian stemmer. However, the morphological complexity of the Latvian language determined that extensive modifications to the structure of the initial stemming algorithm were necessary. Overall, the initial rules for English language were replaced with the new knowledge base comprising conditional and recoding rules for Latvian e.g.:

- extensive Latvian stopword list;
- separate list of Latvian word endings;
- rule lists of Latvian suffixes according to their length;
- special rules of consonant palatalisation;
- special conditions.

Conditional and recoding rules for Latvian language were placed into the structure of Porter- Frakes' suffix removal algorithm because to date it is one of the most flexible stemmers and gives good information retrieval results both for English and non-English languages. If there is a more advanced stemmer, than the same knowledge base could be used for this stemming algorithm. The knowledge base of the Latvian stemmer (e.g. the stopword list) can be also modified and developed according to the area in which the stemmer is going to be implemented.

The initial testing and examination of the Latvian stemming algorithm using an electronic dictionary of Latvian nouns, adjectives, verbs and adverbs, as well as fragments of Latvian texts, confirmed that the Latvian stemmer stems both Latvian words in standard forms and in declensions correctly and leaves relevant resulting stems of words as well as removes appropriate stopwords.

Evaluation results of information retrieval effectiveness proved that search statements processed by the Latvian stemming algorithm produced more relevant documents than traditional manual right hand truncation.

Latvian language was used as an example of an inflective language. The knowledge base of this stemming algorithm can be adapted to other inflective languages which belong to the same language family as Latvian (i.e. Italian, Celtic, Germanic, Slavic) and which use both Roman and non-Roman character sets. Moreover, even if the language does not belong to the same language group, but its morphological structure is similar to Indo-European languages, the algorithm might be applicable for this language. For example, the knowledge base could not be implemented for agglutinative languages (e.g. Finnish or Turkish) or Chinese but possibly it could be applied for the Semitic language group (e.g. Arabic, Hebrew).

8.2 Further research

- A more extensive evaluation study of the Latvian stemming algorithm covering additional statistical methods for examination e.g. Anova test and/or T-test could be carried out, if the algorithm was implemented in some of the existing Latvian databases and tested by users of these databases. To date several information and computer software companies in Latvia e.g. Software House Riga, Lursoft Ltd. etc. expressed their interest in applying the stemming algorithm for information retrieval from their databases.

- Additional modifications and improvements to the structure of stemming algorithm regarding the complexity of Latvian language e.g. to cover all cases of consonant palatalisation, can be done.

- Comparative analysis of stemming both English and Latvian words using different types of stemming algorithms e.g. n-gram, successor variety stemmers can be carried out.

- Analysis and evaluation of the Latvian stemming algorithm used in a multilingual database or information system can be completed.

BIBLIOGRAPHY

Adamson, G. & J. Boreham. The use of an association measure based on character structure to identify semantically related pairs of words and document titles. *Information Storage and Retrieval,* 1974, **10,** 253-260.

Al-Kharashi, Ibrahim A. & Martha W. Evans. Comparing words, stems and roots as index terms in an Arabic information retrieval system. *Journal of the American Society for Information Science,* 1994, **45**(8), 548-560.

Ashford, John & Peter Willett. *Text retrieval and document databases.* Bromley: Chartwell-Bratt, 1988.

Bell, Colin L. M. & Kevin P. Jones. A minicomputer retrieval system with automatic root finding and roling facilities. *Program,* 1976, **10**(1), 14-27.

Biru, Tesfaye et al. Inclusion of relevance information in the term discrimination model. *Journal of Documentation,* 1989, **45**(2), 85-109.

Brzozowski, J. P. MASQUERADE: searching the full text of abstracts using automatic indexing. *Journal of Information Science,* 1983, **6,** 67-73.

Ceplīte, B. & L. Ceplītis. *Latviešu valodas praktiskā gramatika* [The practical grammar of Latvian language]. Riga: Zvaigzne, 1991.

Dattola, Robert T. FIRST: Flexible information retrieval system for text. *Journal of the American Society for Information Science,* 1979, **30**(1), 9-14.

Dillon, Martin & Ann S. Gray. FASIT: A fully automatic syntactically based indexing system. *Journal of the American Society for Information Science,* 1983, **34**(2), 99-108.

Drizule, Viktorija. *Razrabotka priblizhennih metodov avtomaticheskovo morfologicheskovo analiza tekstov latyshskovo jazyka* [Design of proximate methods for morphological analysis of Latvian language]: A summary of dissertation. Minsk, 1975.

Dubois, C. P. R. Multilingual information systems: some criteria for the choice of specific techniques. *Journal of Information Science,* 1979, **1**, 5-12.

Ekmekcioglu, Cuna F., Alexander M. Robertson & Peter Willett. Effectiveness of query expansion in ranked-output document retrieval systems. *Journal of Information Science,* 1992, **18**(2), 139-147.

Fagan, Joel L. The effectiveness of a nonsyntactic approach to automatic phrase indexing for document retrieval. *Journal of the American Society for Information Science,* 1989, **40**(2), 115-132.

Fennell, Trevor G. & Henry Gelsen. *A grammar of modern Latvian.* 3 vols. The Hague: Mouton, 1980.

Frakes, William B. & Ricardo Baeza-Yates. *Information retrieval: data structures and algorithms.* Englewood Cliffs, N. J.: Prentice Hall, 1992.

Fuhr, Norbert von. Zur Überwindung der Diskrepanz zwischen Retrievalforschung und- praxis [Bridging the gap between retrieval research and practice]. *Nachrichten für Dokumentation,* 1990, **41**(1), 3-7.

Hafer, Margaret A. & Stephen F. Weiss. Word segmentation by letter successor varieties. *Information Storage & Retrieval,* 1974, **10,** 371-385.

Hancock-Beaulieu, Micheline & Stephen Walker. An evaluation of automatic query expansion in an online library catalogue. *Journal of Documentation,* 1992, **48**(4), 406-421.

Hancock-Beaulieu, Micheline & Stephen Walker. Query expansion: advances in research in online catalogues. *Journal of Information Science,* 1992, **18**(2), 99-103.

Harman, Donna. How effective is suffixing? *Journal of the American Society for Information Science,* 1991, **42**(1), 7-15.

Harter, Stephen P. A probabilistic approach to automatic keyword indexing. *Journal of the American Society for Information Science,* 1975, **26,** 280-289.

Hendry, Ian G., Peter Willett & Frances E. Wood. INSTRUCT: a teaching package for experimental methods in information retrieval. Part 1. The users' view. *Program,* 1986, **20**(3), 245-263.

Hendry, Ian G., Peter Willett & Frances E. Wood. INSTRUCT: a teaching package for experimental methods in information retrieval. Part 2. Computational aspects. *Program,* 1986, **20**(4), 382-393.

Hudson, Richard. *English word grammar.* Oxford: Basil Blackwell, 1991.

Hull, David A. Stemming algorithms: a case study for detailed evaluation. *Journal of the American Society for Information Science,* 1996, **47**(1), 70-84.

Jakubaite, T. et al. *Latviešu valodas biežuma vārdnīca* [Frequency dictionary of Latvian language], vol 4: Science. Riga: Zinātne, 1976.

Johnson, Bonnie & Elaine Peterson. Reviewing initial stopword selection. *Information Technology and Libraries,* 1992, **11**(2), 136-139.

Jones, Leslie P., Edward W. Gassie & Sridhar Radhakrishnan. INDEX: the statistical basis for an automatic conceptual phrase-indexing system. *Journal of the American Society for Information Science,* 1990, **41**(2), 87-97.

Jung Soon, Ro. An evaluation of the applicability of ranking algorithms to improve the effectiveness of full-text retrieval. II. On the effectiveness of ranking algorithms on full-text retrieval. *Journal of the American Society for Information Science,* 1988, **39**(3), 147-160.

Kalamboukis, T. Z. Suffix stripping with modern Greek. *Program,* 1995, **29**(3), 313-321.

Keen, Michael E. The use of term position devices in ranked output experiments. *Journal of Documentation,* 1991, **47**(1), 1-22.

Lennon, Martin et al. An evaluation of some conflation algorithms for information retrieval. *Journal of Information Science,* 1981, **3,** 177-183.

Lochbaum, Karen E. & Lynn A. Streeter. Comparing and combining the effectiveness of latent semantic indexing and the ordinary vector space model for information retrieval. *Information Processing and Management,* 1989, **25**(6), 665-676.

Lovins, J. B. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics,* 1968, **11**(1), 22-31.

Lovins, Julie B. Error evaluation for stemming algorithms as clustering algorithms. *Journal of the American Society for Information Science,* 1971, **22**(1), 28-40.

Luhn, H. P. A statistical approach to mechanical encoding and searching of library information. *IBM Journal of Research and Development,* 1957, **1**(4) 309-317.

Metuzāle-Kangare, Baiba. *A derivational dictionary of Latvian.* Hamburg: Buske, 1985.

Neumeyer, Frederick J., ed. *Linguistics, the Cambridge survey.* Vol. 1. Linguistic theory, foundations. Cambridge: Cambridge University Press, 1988.

Overhage, Carl F. J. & Francis J. Reintjes. Project INTREX: a general review. *Information Storage and Retrieval,* 1974, **10**, 157-188.

Paice, C. Another stemmer. *ACM SIGIR Forum,* 1990, **24**(3), 56-61.

Parker, Frank. *Linguistics for non-linguists.* London: Taylor and Francis, 1986.

Peat, Helen J. & Peter Willett. The limitations of term co-occurrence data for query expansion in document retrieval systems. *Journal of the American Society for Information Science,* 1991, **42**(5), 378-383.

Perry, Shirley A. & Peter Willett. A review of the use of inverted files for best match searching in information retrieval systems. *Journal of Information Science,* 1983, **6**, 59-66.

Pietilainen, Pirkko. Local feedback and intelligent automatic query expansion. *Information Processing and Management,* 1983, **19**(1), 51-58.

Popovič, Mirko. *Implementation of a Slovene language-based free text retrieval system:* PhD Thesis. Sheffield: University of Sheffield, 1991.

Popovič, Mirko & Peter Willett. The effectiveness of stemming for natural language access to Slovene textual data. *Journal of the American Society for Information Science,* 1992, **43**(5), 384-390.

Porter, Martin. An algorithm for suffix stripping. *Program,* 1980, **14**(3), 130-137.

Robertson, S. E. On relevance weight estimation and query expansion. *Journal of Documentation,* 1986, **42**(3), 182-188.

Robertson, S. E. On term selection for query expansion. *Journal of Documentation,* 1990, **46**(4), 359-364.

Robertson, S. E. & M. M. Hancock-Beaulieu. On the evaluation of IR systems. *Information Processing and Management,* 1992, **28**(4), 457-466.

Robertson, S. E. & K. Spark Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science,* 1976, **27**, 129-146.

Salton, Gerard. *Automatic information organization and retrieval.* London: McGraw-Hill Book Company, 1968.

Salton, Gerard. *Automatic text processing: the transformation, analysis and retrieval of information by computer.* Wokingham: Addison-Wesley, 1989.

Salton, Gerard, ed. *The SMART retrieval system: experiments in automatic document processing.* Englewood Cliffs, N. J.: Prentice Hall, 1971.

Salton, Gerard & Christopher Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management,* 1988, **24**(5), 513-523.

Salton, Gerard & M. E. Lesk. The SMART automatic document retrieval system. *Communications of the ACM,* 1965, **8**(6), 391-398.

Salton, Gerard & Michael J. McGill. *Introduction to modern information retrieval.* London: McGraw-Hill, 1983.

Savoy, Jacques. Stemming of French words based on grammatical categories. *Journal of the American Society for Information Science,* 1993, **44**(1), 1-9.

Schinke, Robyn et al. A stemming algorithm for Latin text databases. *Journal of Documentation,* 1996, **52**(2), 172-187.

Selkirk, Elisabeth O. *The syntax of words.* London: MIT Press, 1982.

Smeaton, Alan F. Natural language processing and information retrieval. *Information Processing and Management,* 1990, **26**(1), 19-20.

Soida, S. & S. Kļaviņa. *Latviešu valodas inversā vārdnīca* [Inverse dictionary of Latvian language]. Riga: Zinātne, 1970.

Solak, Aysin & Kemal Oflazer. Design and implementation of a spelling checker for Turkish. *Literary and Linguistic Computing,* 1993, **8**(3), 118-130.

Spark Jones, K. & J. I. Tait. Automatic search term variant generation. *Journal of Documentation,* 1984, **40**(1), 50-66.

Spark Jones, Karen. *Information retrieval experiment.* London: Butterworths, 1981.

Ulmschneider, John & Tamas Doszkocs. A practical stemming algorithm for online search assistance. *Online Review,* 1983, **7**(4), 301-318.

Van Rijsbergen, C. J., ed. *Information retrieval.* 2Nd ed. London: Butterworths, 1979.

Van Rijsbergen C. J., ed. *Research and development in information retrieval.* Cambridge: Cambridge University Press, 1984.

Vickery, Brian & Alina Vickery. An application of language processing for a search interface. *Journal of Documentation,* 1992, **48**(3), 255-275.

Vickery, Brian C. & Alina Vickery. *Information science in theory and practice.* London: Butterworths, 1987.

Wade, Stephen J. & Peter Willett. INSTRUCT: a teaching package for experimental methods in information retrieval. Part 3. Browsing, clustering and query expansion. *Program,* 1988, **22**(1), 44-61.

Walker, Stephen. OKAPI: evaluating and enhancing an experimental online catalog. *Library Trends,* 1987, **35**(4), 631-645.

Walker, Stephen & Rachel de Vere. *Improving subject retrieval in online catalogues.* 2. Relevance feedback and query expansion. London: The British Library Research and Development Department, 1990.

Walker, Stephen & Richard M. Jones. *Improving subject retrieval in online catalogues.* 1. Stemming, automatic spelling correction and cross-reference tables. London: The Polytechnic of Central London, 1987.

Willett, Peter, ed. *Document retrieval systems.* London: Taylor Graham, 1988.

Wong, S. K. M. & Y. Y. Yao. Query formulation in linear retrieval models. *Journal of the American Society for Information Science,* 1990, **41**(5), 334-341.

Wu, Zimin. *A partial syntactic analysis-based pre-processor for automatic indexing and retrieval of Chinese texts:* PhD Thesis. Loughborough: Loughborough University of Technology, 1992.

APPENDICES

APPENDIX 1
LATVIAN STOPLIST

Prepositions

aiz	ap	ar	apakš	ārpus
augšpus	bez	caur	dēļ	gar
iekš	iz	kopš	labad	lejpus
līdz	no	otrpus	pa	par
pār	p´c	pie	pirms	pret
priekš	starp	šaipus	uz	viņpus
virs	virspus	zem	apakšpus	

Conjunctions

un	bet	jo	ja	ka
lai	tomēr	tikko	turpretī	arī
kaut	gan	tādēļ	tā	ne
tikvien	vien	kā	ir	te
vai	kamēr			

Particles

ar	diezin	droši	diemžēl	nebūt
ik	it	taču	nu	pat
tiklab	iekšpus	nedz	tik	nevis
turpretim	jeb	iekam	iekām	iekāms
kolīdz	līdzko	tiklīdz	jebšu	tālab
tāpēc	nekā	itin	jā	jau
jel	nē	nezin	tad	tikai
vis	tak	iekams	vien	

Pronouns

es	tu	viņš	kurš	viss
manis	tevis	viņa	kura	visa
man	tev	viņam	kuram	visam
mani	tevi	viņu	kuru	visu
manī	tevī	viņā	kurā	visā
mēs	jūs	viņi	kuri	visi
mūsu	jūsu	viņiem	kuriem	visiem
mums	jums	viņus	kurus	visus
mūs	jūsos	viņos	kuros	visos
mūsos		viņas	kuras	visas
		viņai	kurai	visai
		viņām	kurām	visām
		viņās	kurās	visās
sevis	tas	tiem	šis	šīs
sev	tā	tos	šī	šās
sevi	tam	tais	šā	šie
sevī	to	tajos	šim	šiem
kas	tajā	tanīs	šo	šām
kā	tai	tām	šai	šos
kam	tanī	tajās	šajā	šais
ko	tās		šinī	šajos
kur	tie			šajās
				šinīs
mans	manas	tavi	savs	savos
mana	manai	taviem	sava	savas
manam	manām	tavus	savam	savai
manu	manās	tavos	savu	savām
manā	tavs	tavas	savā	savās
mani	tava	tavai	savi	
maniem	tavam	tavām	saviem	
manus	tavu	tavās	savus	
manos	tavā			

cits	dažs	kāds	kurš	tāds
cita	daža	kāda	kura	tāda
citam	dažam	kādam	kuram	tādam
citu	dažu	kādu	kuru	tādu
citā	dažā	kādā	kurā	tādā
citi	daži	kādi	kuri	tādi
citiem	dažiem	kādiem	kuriem	tādiem
citus	dažus	kādus	kurus	tādus
citos	dažos	kādos	kuros	tādos
citas	dažas	kādas	kuras	tādas
citai	dažai	kādai	kurai	tādai
citām	dažām	kādām	kurām	tādām
citās	dažās	kādās	kurās	tādās

šāds	katrs	manējs	manējais	tavēji
šāda	katra	manēja	manējo	tavējiem
šādam	katram	manējam	manējie	tavējus
šādu	katru	manēju	manējās	tavējos
šādā	katrā	manējā	manējai	tavējais
šādi	katri	manēji	manējām	tavējo
šādiem	katriem	manējiem	manējas	tavējie
šādus	katrus	manējus	tavējs	tavējās
šādos	katros	manējos	tavēja	tavējai
šādas	katras		tavējam	tavējām
šādām	katrai		tavēju	tavējas
šādās	katrām		tavējā	
	katrās			

savējs	viņējs	jūsējs	mūsējs	šitas
savēja	viņēja	jūsēja	mūsēja	šitā
savējam	viņējam	jūsējam	mūsējam	šitam
savēju	viņēju	jūsēju	mūsēju	šito
savējā	viņējā	jūsējā	mūsējā	šitai
savēji	viņēji	jūsēji	mūsēji	šitie
savējiem	viņējiem	jūsējiem	mūsējiem	šitiem
savējus	viņējus	jūsējus	mūsējus	šitos
savējos	viņējos	jūsējos	mūsējos	šitās
savējais	viņējais	jūsējais	mūsējais	šitām

savējo	viņējo	jūsējie	mūsējo	
savējie	viņējie	jūsējie	mūsējie	
savējās	viņējās	jūsējās	mūsējās	
savējai	viņējai	jūsējai	mūsējai	
savējām	viņējām	jūsējām	mūsējām	
savējas	viņējas	jūsējas	mūsējas	

šitāds	ikkatrs	jebkāds	jebkas	ikkurš
šitāda	ikkatra	jebkāda	jebkā	ikkura
šitādas	ikkatram	jebkādam	jebkam	ikkuram
šitādu	ikkatru	jebkādu	jebko	ikkuru
šitādā	ikkatrā	jebkādā	jebkurš	ikkurā
šitādas	ikkatras	jebkādas	jebkura	ikkuras
šitādai	ikkatrai	jebkādai	jebkuram	ikkurai
šitādi	ikkatri	jebkādi	jebkuru	ikkuri
šitādiem	ikkatriem	jebkādiem	jebkurā	ikkuriem
šitādus	ikkatrus	jebkādus	jebkuras	ikkurus
šitādos	ikkatros	jebkādos	jebkurai	ikkuros
šitādām	ikkatrām	jebkādām	jebkuri	ikkurām
šitādās	ikkatrās	jebkādās	jebkuriem	ikkurās
			jebkurus	
			jebkuros	
			jebkurām	
			jebkurās	

ikviens	nekas	nekādi	nevieni	pašai
ikviena	nekā	nekādiem	nevieniem	paši
ikvienam	nekam	nekādus	nevienus	pašiem
ikvienu	neko	nekādos	nevienos	pašus
ikvienā	nekāds	nekādām	nevienām	pašos
ikvienas	nekāda	nekādās	nevienās	pašām
ikvienai	nekādam	neviens	pats	pašās
ikvieni	nekādu	neviena	paša	
ikvieniem	nekādā	nevienam	pašam	
ikvienus	nekādas	nevienu	pašu	
ikvienos	nekādai	nevienā	pašā	
ikvienām		nevienas	pati	
ikvienās		nevienai	pašas	

Modal verbs

būt	tikt	tapt	kļūt	kļūšu
biju	tiku	tapi	kļuvu	kļūsi
biji	tiki	tapāt	kļuvi	kļūs
bija	tika	topat	kļuva	kļūsim
bijām	tikām	tapšu	kļuvām	kļūsiet
bijāt	tikāt	tapsi	kļuvāt	
esmu	tieku	taps	kļūstu	
esi	tiec	tapsim	kļūsti	
esam	tiek	tapsiet	kļūst	
esat	tiekam		kļūstam	
būšu	tiekat		kļūstat	
būsi	tikšu			
būs	tiks			
būsim	tiksim			
būsiet	tiksiet			

Verbs

varēt	varēju	varējām	varēšu	varēsim
var	varēji	varējāt	varēsi	varēsiet
varat	varēja		varēs	

Interjections

klau	re	ak	skat	parau
lūk	pag	palūk	paskat	tpū
ūja	raug	nudien	paklau	vau
urrā	paraug	ekur	ņau	redz
urā	ai	kuš	rau	

Adverbs

kāpēc	še	pārāk	aplam	iepretim
kādēļ	tādējādi	agrāk	piemēram	prom
kālab	visādi	vairāk	apmēram	patlaban
tālab	visvisādi	visvairāk	nepagalam	diezgan
kālabad	citādi	mazāk	pavisam	varbūt
tālabad	parasti	drīzāk	nepavisam	šeitan
kamdēļ	dikti	visbiežāk	paretam	secen
tamdēļ	ļoti	cik	šimbrīžam	šobaltdien
bezgala	velti	necik	joprojām	kādudien
nenieka	pēkšņi	šitik	aumaļām	citudien
samērā	respektīvi	atkal	lēnām	daždien
vērā	līdzi	tūdaļ	pamazām	mūždien
visupēc	pretī	pakaļ	gaužām	arvien
tagad	labāk	iepakaļ	aizgūtnēm	aizvien
kad	pēcāk	nopakaļ	pārpārēm	varen
jebkad	citādāk	visnotaļ	caurcaurēm	sen
nekad	savādāk	atpakaļ	pamazītēm	pasen
šad	turpmāk	pa laikam	pretim	nesen
bāztin	sensenis	nelabprāt	dažkārt	kādreiz
druscin	vairs	manuprāt	nost	vairākreiz
tūliņ	papildus	mūsuprāt	pārlieku	cikreiz
mazlietiņ	pārmijus	tīšuprāt	šeit	tikreiz
neparko	blakus	tavuprāt	tūlīt	vēlreiz
vienkop	ieblakus	ciet	pirmīt	nākamreiz
kurp	līdztekus	mazliet	maķenīt	viņreiz
šurp	aplinkus	vieviet	atstatu	šoreiz
turp	izklaidus	vienuviet	maz	toreiz
vispār	vienlaidus	dažviet	pamaz	pašreiz
viscaur	neviļus	beidzot	nemaz	nākošreiz
jebkur	abpus	visbeidzot	vismaz	citreiz
nekur	vienpus	vairākkārt	daudzmaz	citureiz
visur	katrpus	pirmkārt	bezmaz	daudzreiz
šur	otrpus	vienkārt	vienlīdz	uzreiz
tur	virspus	galvenokārt	puslīdz	dažreiz
citur	papriekš	apkārt	daudz	drīz
vietumis	iepriekš	visapkārt	nedaudz	gandrīz

retumis	klāt	citkārt	reiz	allaž
reizumis	labprāt	daudzkārt	ikreiz	

APPENDIX 2.1

Main stemming program STEMMER.C

```
/*************************************  stemmer.c  ************************

*  Program to demonstrate and test the Porter stemming function.  This
*  program takes a single filename on the command line and lists stemmed
*  terms on stdout.
*
**/

#include <stdio.h>
#include <ctype.h>

#include "stem.h"

/*****************************************************************************/
/**********************  Private Defines and Data Structures  ****************/

#define EOS                     '\0'

/*****************************************************************************/
/*************************  Private Function Definitions  ********************/

#ifdef __STDC__

static char * GetNextTerm( FILE *stream, int size, char *term );

#else

static char * GetNextTerm();

#endif

/*****************************************************************************

    GetNextTerm( stream, size, term )

  Returns:      char * -- buffer with the next input term, NULL at EOF

  Purpose:      Grab the next token from an input stream

  Plan:         Part 1: Return NULL immediately if there is no input
                Part 2: Initialize the local variables
                Part 3: Main Loop: Put the next word into the term buffer
                Part 4: Return the output buffer

  Notes:        None.
**/

static char *
GetNextTerm( stream, size, term )
  FILE *stream;  /* in: source of input characters */
  int size;     /* in: bytes in the output buffer */
  char *term;   /* in/out: where the next term in placed */
```

```
{
char *ptr;  /* for scanning through the term buffer */
int ch;     /* current character during input scan */

        /* Part 1: Return NULL immediately if there is no input */
if ( EOF == (ch = getc(stream)) ) return( NULL );

        /* Part 2: Initialize the local variables */
*term = EOS;
ptr = term;

        /* Part 3: Main Loop: Put the next word into the term buffer */
do
 {
   /* scan past any leading non-alphabetic characters */
 while ( (EOF != ch ) && !isalpha(ch) ) ch = getc( stream );

    /* copy input to output while reading alphabetic characters */
 while ( (EOF != ch ) && isalpha(ch) )
    {
    if ( ptr == (term+size-1) ) ptr = term;
    *ptr++ = ch;
    ch = getc( stream );
     }

     /* terminate the output buffer */
  *ptr = EOS;
  }
while ( (EOF != ch) && !*term );

            /* Part 4: Return the output buffer */
 return( term );

  } /* GetNextTerm */


/*****************************************************************************/
/*****************************************************************************

     main( argc, argv )

  Returns: int -- 0 on success, 1 on failure

  Purpose:      Program main function

  Plan:         Part 1: Open the input file
                Part 2: Process each word in the file
                Part 3: Close the input file and return

  Notes:        None
**/

int
main( argc, argv )
  int argc;     /* in: how many arguments */
  char *argv[]; /* in: text of the arguments */
  {
  char term[64];   /* for the next term from the input line */
```

```
FILE *stream;    /* where to read characters from */

                /* Part 1: Open the input file */
if ( !(stream = fopen(argv[1],"r")) ) exit(1);

              /* Part 2: Process each word in the file */
while( GetNextTerm(stream,64,term) )
  if ( Stem(term) ) (void)printf( "%s\n", term );

              /* Part 3: Close the input file and return */
(void)fclose( stream );
return(0);

} /* main */
```

APPENDIX 2.2

Initial stemming program STEM.C

```
/*********************************  stem.c  ***********************************

    Purpose:    Implementation of the Porter stemming algorithm documented
                in: Porter, M.F., "An Algorithm For Suffix Stripping,"
                Program 14 (3), July 1980, pp. 130-137.

    Provenance: Written by B. Frakes and C. Cox, 1986.
                Changed by C. Fox, 1990.
                - made measure function a DFA
                - restructured structs
                - renamed functions and variables
                - restricted function and variable scopes
                Changed by C. Fox, July, 1991.
                - added ANSI C declarations
                 - branch tested to 90% coverage

    Notes:      This code will make little sense without the the Porter
                 article.  The stemming function converts its input to
                 lower case.
**/

/**************************  Standard Include Files
***************************/

#include <stdio.h>
#include <string.h>
#include <ctype.h>

/*****************************************************************************/
/******************  Private Defines and Data Structures  ********************/

#define FALSE                   0
#define TRUE                    1
#define EOS                     '\0'

#define IsVowel(c)      ('a'==(c)||'e'==(c)||'i'==(c)||'o'==(c)||'u'==(c))

typedef struct {
        int id;                 /* returned if rule fired */
        char *old_end;          /* suffix replaced */
        char *new_end;          /* suffix replacement */
        int old_offset;         /* from end of word to start of suffix */
        int new_offset;         /* from beginning to end of new suffix */
        int min_root_size;      /* min root word size for replacement */
        int (*condition)();     /* the replacement test function */
        } RuleList;

static char LAMBDA[1] = "";  /* the constant empty string */
static char *end;            /* pointer to the end of the word */
```

```
/*************************************************************************/
/*********************  Private Function Declarations  *******************/

#ifdef __STDC__

static int WordSize( char *word );
static int ContainsVowel( char *word );
static int EndsWithCVC( char *word );
static int AddAnE( char *word );
static int RemoveAnE( char *word );
static int ReplaceEnd( char *word, RuleList rule );

#else

static int WordSize( /* word */ );
static int ContainsVowel( /* word */ );
static int EndsWithCVC( /* word */ );
static int AddAnE( /* word */ );
static int RemoveAnE( /* word */ );
static int ReplaceEnd( /* word, rule */ );

#endif

/*************************************************************************/
/*****************  Initialized Private Data Structures  *****************/

static RuleList step1a_rules[] =
        {
         101,  "sses",          "ss",         3,  1, -1,      NULL,
         102,  "ies",           "i",          2,  0, -1,      NULL,
         103,  "ss",            "ss",         1,  1, -1,      NULL,
         104,  "s",             LAMBDA,       0, -1, -1,      NULL,
         000,  NULL,            NULL,         0,  0,  0,      NULL,
        };

static RuleList step1b_rules[] =
        {
         105,  "eed",           "ee",         2,  1,  0,      NULL,
         106,  "ed",            LAMBDA,       1, -1, -1,      ContainsVowel,
         107,  "ing",           LAMBDA,       2, -1, -1,      ContainsVowel,
         000,  NULL,            NULL,         0,  0,  0,      NULL,
        };

static RuleList step1b1_rules[] =
        {
         108,  "at",            "ate",        1,  2, -1,      NULL,
         109,  "bl",            "ble",        1,  2, -1,      NULL,
         110,  "iz",            "ize",        1,  2, -1,      NULL,
         111,  "bb",            "b",          1,  0, -1,      NULL,
         112,  "dd",            "d",          1,  0, -1,      NULL,
         113,  "ff",            "f",          1,  0, -1,      NULL,
         114,  "gg",            "g",          1,  0, -1,      NULL,
         115,  "mm",            "m",          1,  0, -1,      NULL,
         116,  "nn",            "n",          1,  0, -1,      NULL,
         117,  "pp",            "p",          1,  0, -1,      NULL,
         118,  "rr",            "r",          1,  0, -1,      NULL,
         119,  "tt",            "t",          1,  0, -1,      NULL,
```

```
    120,  "ww",          "w",          1,  0, -1,      NULL,
    121,  "xx",          "x",          1,  0, -1,      NULL,
    122,  LAMBDA,        "e",          -1,  0, -1,     AddAnE,
    000,  NULL,          NULL,         0,  0,  0,      NULL,
    };

static RuleList step1c_rules[] =
    {
    123,  "y",           "i",          0,  0, -1,      ContainsVowel,
    000,  NULL,          NULL,         0,  0,  0,      NULL,
    };

static RuleList step2_rules[] =
    {
    203,  "ational",     "ate",        6,  2,  0,      NULL,
    204,  "tional",      "tion",       5,  3,  0,      NULL,
    205,  "enci",        "ence",       3,  3,  0,      NULL,
    206,  "anci",        "ance",       3,  3,  0,      NULL,
    207,  "izer",        "ize",        3,  2,  0,      NULL,
    208,  "abli",        "able",       3,  3,  0,      NULL,
    209,  "alli",        "al",         3,  1,  0,      NULL,
    210,  "entli",       "ent",        4,  2,  0,      NULL,
    211,  "eli",         "e",          2,  0,  0,      NULL,
    213,  "ousli",       "ous",        4,  2,  0,      NULL,
    214,  "ization",     "ize",        6,  2,  0,      NULL,
    215,  "ation",       "ate",        4,  2,  0,      NULL,
    216,  "ator",        "ate",        3,  2,  0,      NULL,
    217,  "alism",       "al",         4,  1,  0,      NULL,
    218,  "iveness",     "ive",        6,  2,  0,      NULL,
    219,  "fulnes",      "ful",        5,  2,  0,      NULL,
    220,  "ousness",     "ous",        6,  2,  0,      NULL,
    221,  "aliti",       "al",         4,  1,  0,      NULL,
    222,  "iviti",       "ive",        4,  2,  0,      NULL,
    223,  "biliti",      "ble",        5,  2,  0,      NULL,
    000,  NULL,          NULL,         0,  0,  0,      NULL,
    };

static RuleList step3_rules[] =
    {
    301,  "icate",       "ic",         4,  1,  0,      NULL,
    302,  "ative",       LAMBDA,       4, -1,  0,      NULL,
    303,  "alize",       "al",         4,  1,  0,      NULL,
    304,  "iciti",       "ic",         4,  1,  0,      NULL,
    305,  "ical",        "ic",         3,  1,  0,      NULL,
    308,  "ful",         LAMBDA,       2, -1,  0,      NULL,
    309,  "ness",        LAMBDA,       3, -1,  0,      NULL,
    000,  NULL,          NULL,         0,  0,  0,      NULL,
    };

static RuleList step4_rules[] =
    {
    401,  "al",          LAMBDA,       1, -1,  1,      NULL,
    402,  "ance",        LAMBDA,       3, -1,  1,      NULL,
    403,  "ence",        LAMBDA,       3, -1,  1,      NULL,
    405,  "er",          LAMBDA,       1, -1,  1,      NULL,
    406,  "ic",          LAMBDA,       1, -1,  1,      NULL,
    407,  "able",        LAMBDA,       3, -1,  1,      NULL,
    408,  "ible",        LAMBDA,       3, -1,  1,      NULL,
```

```
    409,  "ant",          LAMBDA,    2, -1,  1,      NULL,
    410,  "ement",        LAMBDA,    4, -1,  1,      NULL,
    411,  "ment",         LAMBDA,    3, -1,  1,      NULL,
    412,  "ent",          LAMBDA,    2, -1,  1,      NULL,
    423,  "sion",         "s",       3,  0,  1,      NULL,
    424,  "tion",         "t",       3,  0,  1,      NULL,
    415,  "ou",           LAMBDA,    1, -1,  1,      NULL,
    416,  "ism",          LAMBDA,    2, -1,  1,      NULL,
    417,  "ate",          LAMBDA,    2, -1,  1,      NULL,
    418,  "iti",          LAMBDA,    2, -1,  1,      NULL,
    419,  "ous",          LAMBDA,    2, -1,  1,      NULL,
    420,  "ive",          LAMBDA,    2, -1,  1,      NULL,
    421,  "ize",          LAMBDA,    2, -1,  1,      NULL,
    000,  NULL,           NULL,      0,  0,  0,      NULL,
    };

static RuleList step5a_rules[] =
    {
    501,  "e",            LAMBDA,    0, -1,  1,      NULL,
    502,  "e",            LAMBDA,    0, -1, -1,      RemoveAnE,
    000,  NULL,           NULL,      0,  0,  0,      NULL,
    };

static RuleList step5b_rules[] =
    {
    503,  "ll",           "l",       1,  0,  1,      NULL,
    000,  NULL,           NULL,      0,  0,  0,      NULL,
    };

/*****************************************************************/
/*********************  Private Function Declarations  ***********************/

/*****************************************************************
```

WordSize(word)

Returns: int -- a weird count of word size in adjusted syllables

Purpose: Count syllables in a special way: count the number vowel-consonant pairs in a word, disregarding initial consonants and final vowels. The letter "y" counts as a consonant at the beginning of a word and when it has a vowel in front of it; otherwise (when it follows a consonant) it is treated as a vowel. For example, the WordSize of "cat" is 1, of "any" is 1, of "amount" is 2, of "anything" is 3.

Plan: Run a DFA to compute the word size

Notes: The easiest and fastest way to compute this funny measure is with a finite state machine. The initial state 0 checks the first letter. If it is a vowel, then the machine changes to state 1, which is the "last letter was a vowel" state. If the first letter is a consonant or y, then it changes to state 2, the "last letter was a consonant state". In state 1, a y is treated as a consonant (since it follows a vowel), but in state 2, y is treated as a vowel (since it follows a consonant. The result counter is incremented on the transition from state 1 to state 2, since this

```
                transition only occurs after a vowel-consonant pair, which
                is what we are counting.
**/

static int
WordSize( word )
  char *word;   /* in: word having its WordSize taken */
  {
  register int result;   /* WordSize of the word */
  register int state;    /* current state in machine */

result = 0;
state = 0;

        /* Run a DFA to compute the word size */
while ( EOS != *word )
  {
  switch ( state )
    {
    case 0: state = (IsVowel(*word)) ? 1 : 2;
            break;
    case 1: state = (IsVowel(*word)) ? 1 : 2;
            if ( 2 == state ) result++;
            break;
    case 2: state = (IsVowel(*word) || ('y' == *word)) ? 1 : 2;
            break;
    }
  word++;
  }

  return( result );

  } /* WordSize */

/*****************************************************************************

    ContainsVowel( word )

  Returns:      int -- TRUE (1) if the word parameter contains a vowel,
                FALSE (0) otherwise.

  Purpose:      Some of the rewrite rules apply only to a root containing
                a vowel, where a vowel is one of "aeiou" or y with a
                consonant in front of it.

  Plan:         Obviously, under the definition of a vowel, a word contains
                a vowel iff either its first letter is one of "aeiou", or
                any of its other letters are "aeiouy".  The plan is to
                test this condition.

  Notes:        None
**/

static int
ContainsVowel( word )
  char *word;   /* in: buffer with word checked */
  {
```

```
  if ( EOS == *word )
    return( FALSE );
  else
    return( IsVowel(*word) || (NULL != strpbrk(word+1,"aeiouy")) );


  } /* ContainsVowel */

/*************************************************************************

      EndsWithCVC( word )

    Returns:    int -- TRUE (1) if the current word ends with a
                consonant-vowel-consonant combination, and the second
                consonant is not w, x, or y, FALSE (0) otherwise.

    Purpose:    Some of the rewrite rules apply only to a root with
                this characteristic.

    Plan:       Look at the last three characters.

    Notes:      None
**/

static int
EndsWithCVC( word )
  char *word;   /* in: buffer with the word checked */
  {
  int length;     /* for finding the last three characters */

  if ( (length = strlen(word)) < 2 )
    return( FALSE );
  else
    {
    end = word + length - 1;
    return(    (NULL == strchr("aeiouwxy",*end--))      /* consonant */
          && (NULL != strchr("aeiouy",  *end--))       /* vowel */
          && (NULL == strchr("aeiou",   *end  )) );   /* consonant */
    }

  } /* EndsWithCVC */

/*************************************************************************

      AddAnE( word )

    Returns:    int -- TRUE (1) if the current word meets special conditions
                for adding an e.

    Purpose:    Rule 122 applies only to a root with this characteristic.

    Plan:       Check for size of 1 and a consonant-vowel-consonant ending.

    Notes:      None
**/

static int
```

```
AddAnE( word )
  char *word;
  {

  return( (1 == WordSize(word)) && EndsWithCVC(word) );

  } /* AddAnE */

/*************************************************************************

        RemoveAnE( word )

  Returns:      int -- TRUE (1) if the current word meets special conditions
                for removing an e.

  Purpose:      Rule 502 applies only to a root with this characteristic.

  Plan:         Check for size of 1 and no consonant-vowel-consonant ending.

  Notes:        None
**/

static int
RemoveAnE( word )
  char *word;
  {

  return( (1 == WordSize(word)) && !EndsWithCVC(word) );

  } /* RemoveAnE */

/*************************************************************************

    ReplaceEnd( word, rule )

  Returns:      int -- the id for the rule fired, 0 is none is fired

  Purpose:      Apply a set of rules to replace the suffix of a word

  Plan:         Loop through the rule set until a match meeting all conditions
                is found.  If a rule fires, return its id, otherwise return 0.
                Connditions on the length of the root are checked as part of this
                function's processing because this check is so often made.

  Notes:        This is the main routine driving the stemmer.  It goes through
                a set of suffix replacement rules looking for a match on the
                current suffix.  When it finds one, if the root of the word
                is long enough, and it meets whatever other conditions are
                required, then the suffix is replaced, and the function returns.
**/

static int
ReplaceEnd( word, rule )
  char *word;       /* in/out: buffer with the stemmed word */
  RuleList *rule;   /* in: data structure with replacement rules */
  {
  register char *ending;  /* set to start of possible stemmed suffix */
  char tmp_ch;            /* save replaced character when testing */
```

```
while ( 0 != rule->id )
   {
   ending = end - rule->old_offset;
   if ( word <= ending )
      if ( 0 == strcmp(ending,rule->old_end) )
         {
         tmp_ch = *ending;
         *ending = EOS;
         if ( rule->min_root_size < WordSize(word) )
            if ( !rule->condition || (*rule->condition)(word) )
               {
               (void)strcat( word, rule->new_end );
               end = ending + rule->new_offset;
               break;
               }
            *ending = tmp_ch;
            }
      rule++;
      }
return( rule->id );

   } /* ReplaceEnd */

/*****************************************************************************/
/*********************  Public Function Declarations  ************************/

/*****************************************************************************

      Stem( word )

Returns:    int -- FALSE (0) if the word contains non-alphabetic characters
                    and hence is not stemmed, TRUE (1) otherwise

Purpose:    Stem a word

Plan:       Part 1: Check to ensure the word is all alphabetic
            Part 2: Run through the Porter algorithm
            Part 3: Return an indication of successful stemming

Notes:      This function implements the Porter stemming algorithm, with
            a few additions here and there.  See:

            Porter, M.F., "An Algorithm For Suffix Stripping,"
            Program 14 (3), July 1980, pp. 130-137.

            Porter's algorithm is an ad hoc set of rewrite rules with
            various conditions on rule firing.  The terminology of
            "step 1a" and so on, is taken directly from Porter's
            article, which unfortunately gives almost no justification
            for the various steps.  Thus this function more or less
            faithfully refects the opaque presentation in the article.
            Changes from the article amount to a few additions to the
            rewrite rules;  these are marked in the RuleList data
            structures with comments.
**/
int
Stem( word )
   char *word;  /* in/out: the word stemmed */
```

```
{
int rule;    /* which rule is fired in replacing an end */

        /* Part 1: Check to ensure the word is all alphabetic */
for ( end = word; *end != EOS; end++ )
  if ( !isalpha(*end) ) return( FALSE );
  else *end = tolower( *end );
end--;

          /*  Part 2: Run through the Porter algorithm */
(void)ReplaceEnd( word, step1a_rules );
  rule = ReplaceEnd( word, step1b_rules );
  if ( (106 == rule) || (107 == rule) )
    (void)ReplaceEnd( word, step1b1_rules );
  (void)ReplaceEnd( word, step1c_rules );

  (void)ReplaceEnd( word, step2_rules );

  (void)ReplaceEnd( word, step3_rules );

  (void)ReplaceEnd( word, step4_rules );

  (void)ReplaceEnd( word, step5a_rules );
  (void)ReplaceEnd( word, step5b_rules );
/* Part 3: Return an indication of successful stemming */
  return( TRUE );

  } /* Stem */
```

APPENDIX 2.3
Latvian stemming algorithm

```
/*********************************  stem.c  *********************************

    Purpose:    Implementation of the Porter stemming algorithm documented
                in: Porter, M.F., "An Algorithm For Suffix Stripping,"
                Program 14 (3), July 1980, pp. 130-137.

    Provenance: Written by B. Frakes and C. Cox, 1986.
                Changed by C. Fox, 1990.
                  - made measure function a DFA
                  - restructured structs
                  - renamed functions and variables
                  - restricted function and variable scopes
                Changed by C. Fox, July, 1991.
                  - added ANSI C declarations
                  - branch tested to 90% coverage
                Changed by K. Kreslins, 1995/1996.
                  - restructured structurs according to the Latvian language
                  - added Latvian stoplist
                  - changed rules
                  - modified suffix lengths

    Notes:      This code will make little sense without the Porter
                article.  The stemming function converts its input to
                lower case.
**/

/************************  Standard Include Files  ************************/

#include <c:\bc4\include\stdio.h>
#include <c:\bc4\include\string.h>
#include <c:\bc4\include\ctype.h>

/*****************************************************************************/
/*******************  Private Defines and Data Structures  *******************/

#define FALSE                   0
#define TRUE                    1
#define EOS                    '\0'

#define IsVowel(c)
('a'==(c)||'ā'==(c)||'e'==(c)||'ē'==(c)||'i'==(c)||'ī'==(c)||'o'==(c)||'u'==(c)||ū'==(c))

typedef struct {
        int id;                 /* returned if rule fired */
        char *old_end;          /* suffix replaced */
        char *new_end;          /* suffix replacement */
        int old_offset;         /* from end of word to start of suffix */
        int new_offset;         /* from beginning to end of new suffix */
        int min_root_size;      /* min root word size for replacement */
        int (*condition)();     /* the replacement test function */
        } RuleList;

static char LAMBDA[1] = "";   /* the constant empty string */
static char *end;             /* pointer to the end of the word */
```

```
/*****************************************************************/
/**********************  Private Function Declarations  **********************/

#ifdef __STDC__

static int WordSize( char *word );
static int ContainsVowel( char *word );
static int EndsWithCVC( char *word );
static int AddAnE( char *word );
static int RemoveAnE( char *word );
static int ReplaceEnd( char *word, RuleList rule );

#else

static int WordSize( /* word */ );
static int ContainsVowel( /* word */ );
static int EndsWithCVC( /* word */ );
static int AddAnE( /* word */ );
static int RemoveAnE( /* word */ );
static int ReplaceEnd( /* word, rule */ );
static int CompStopW( /* word, rule */ );
static int ReplaceW( /* word, rule */ );

#endif

int islatv(int ch);
int SmallLatv(int ch);
static int IsLatVowel( /* word */ );

/*****************************************************************/
/*******************  Initialized Private Data Structures  ***********************/

static RuleList step0a_rules[] =
        {
         001,  "aiz",        LAMBDA,     2, -1, -1,      NULL,
         002,  "ap",         LAMBDA,     1, -1, -1,      NULL,
         003,  "ar",         LAMBDA,     1, -1, -1,      NULL,
         004,  "apakš",      LAMBDA,     4, -1, -1,      NULL,
         005,  "ārpus",      LAMBDA,     4, -1, -1,      NULL,
         006,  "augšpus",    LAMBDA,     6, -1, -1,      NULL,
         007,  "bez",        LAMBDA,     2, -1, -1,      NULL,
         010,  "caur",       LAMBDA,     3, -1, -1,      NULL,
         011,  "dēļ",        LAMBDA,     2, -1, -1,      NULL,
         012,  "gar",        LAMBDA,     2, -1, -1,      NULL,
         013,  "iekš",       LAMBDA,     3, -1, -1,      NULL,
         014,  "iz",         LAMBDA,     1, -1, -1,      NULL,
         015,  "kopš",       LAMBDA,     3, -1, -1,      NULL,
         016,  "labad",      LAMBDA,     4, -1, -1,      NULL,
         017,  "lejpus",     LAMBDA,     5, -1, -1,      NULL,
         020,  "līdz",       LAMBDA,     3, -1, -1,      NULL,
         021,  "no",         LAMBDA,     1, -1, -1,      NULL,
         022,  "otrpus",     LAMBDA,     5, -1, -1,      NULL,
         023,  "pa",         LAMBDA,     1, -1, -1,      NULL,
         024,  "par",        LAMBDA,     2, -1, -1,      NULL,
         025,  "pār",        LAMBDA,     2, -1, -1,      NULL,
         026,  "pēc",        LAMBDA,     2, -1, -1,      NULL,
         027,  "pie",        LAMBDA,     2, -1, -1,      NULL,
         030,  "pirms",      LAMBDA,     4, -1, -1,      NULL,
```

```
    031,  "pret",        LAMBDA,    3, -1, -1,      NULL,
    032,  "priekš",      LAMBDA,    5, -1, -1,      NULL,
    033,  "starp",       LAMBDA,    4, -1, -1,      NULL,
    034,  "šaipus",      LAMBDA,    5, -1, -1,      NULL,
    035,  "uz",          LAMBDA,    1, -1, -1,      NULL,
    036,  "viņpus",      LAMBDA,    5, -1, -1,      NULL,
    037,  "virs",        LAMBDA,    3, -1, -1,      NULL,
    040,  "virspus",     LAMBDA,    6, -1, -1,      NULL,
    041,  "zem",         LAMBDA,    2, -1, -1,      NULL,
    042,  "un",          LAMBDA,    1, -1, -1,      NULL,
    043,  "bet",         LAMBDA,    2, -1, -1,      NULL,
    044,  "jo",          LAMBDA,    1, -1, -1,      NULL,
    045,  "ja",          LAMBDA,    1, -1, -1,      NULL,
    046,  "ka",          LAMBDA,    1, -1, -1,      NULL,
    047,  "lai",         LAMBDA,    2, -1, -1,      NULL,
    050,  "tomēr",       LAMBDA,    4, -1, -1,      NULL,
    051,  "tikko",       LAMBDA,    4, -1, -1,      NULL,
    052,  "turpretī",    LAMBDA,    7, -1, -1,      NULL,
    053,  "arī",         LAMBDA,    2, -1, -1,      NULL,
    054,  "kaut",        LAMBDA,    3, -1, -1,      NULL,
    055,  "gan",         LAMBDA,    2, -1, -1,      NULL,
    056,  "tādēļ",       LAMBDA,    4, -1, -1,      NULL,
    057,  "tā",          LAMBDA,    1, -1, -1,      NULL,
    060,  "ne",          LAMBDA,    1, -1, -1,      NULL,
    061,  "tikvien",     LAMBDA,    6, -1, -1,      NULL,
    062,  "vien",        LAMBDA,    3, -1, -1,      NULL,
    063,  "kā",          LAMBDA,    1, -1, -1,      NULL,
    064,  "ir",          LAMBDA,    1, -1, -1,      NULL,
    065,  "te",          LAMBDA,    1, -1, -1,      NULL,
    066,  "vai",         LAMBDA,    2, -1, -1,      NULL,
    067,  "kamēr",       LAMBDA,    4, -1, -1,      NULL,
    070,  "apakšpus",    LAMBDA,    7, -1, -1,      NULL,
    071,  "ar",          LAMBDA,    1, -1, -1,      NULL,
    072,  "diezin",      LAMBDA,    5, -1, -1,      NULL,
    073,  "ik",          LAMBDA,    1, -1, -1,      NULL,
    074,  "it",          LAMBDA,    1, -1, -1,      NULL,
    075,  "taču",        LAMBDA,    3, -1, -1,      NULL,
    076,  "nu",          LAMBDA,    1, -1, -1,      NULL,
    077,  "pat",         LAMBDA,    2, -1, -1,      NULL,
    000,  NULL,          NULL,      0,  0,  0,      NULL,
  };

static RuleList step0b_rules[] =
  {
    001,  "tiklab",      LAMBDA,    5, -1, -1,      NULL,
    002,  "iekšpus",     LAMBDA,    6, -1, -1,      NULL,
    003,  "nedz",        LAMBDA,    3, -1, -1,      NULL,
    004,  "tik",         LAMBDA,    2, -1, -1,      NULL,
    005,  "nevis",       LAMBDA,    4, -1, -1,      NULL,
    006,  "turpretim",   LAMBDA,    8, -1, -1,      NULL,
    007,  "jeb",         LAMBDA,    2, -1, -1,      NULL,
    010,  "iekam",       LAMBDA,    4, -1, -1,      NULL,
    011,  "iekām",       LAMBDA,    4, -1, -1,      NULL,
    012,  "iekāms",      LAMBDA,    5, -1, -1,      NULL,
    013,  "kolīdz",      LAMBDA,    5, -1, -1,      NULL,
    014,  "līdzko",      LAMBDA,    5, -1, -1,      NULL,
    015,  "tiklīdz",     LAMBDA,    6, -1, -1,      NULL,
    016,  "jebšu",       LAMBDA,    4, -1, -1,      NULL,
```

```
    017,  "tālab",        LAMBDA,     4, -1, -1,        NULL,
    020,  "tāpēc",        LAMBDA,     4, -1, -1,        NULL,
    021,  "nekā",         LAMBDA,     3, -1, -1,        NULL,
    022,  "itin",         LAMBDA,     3, -1, -1,        NULL,
    023,  "jā",           LAMBDA,     1, -1, -1,        NULL,
    024,  "jau",          LAMBDA,     2, -1, -1,        NULL,
    025,  "jel",          LAMBDA,     2, -1, -1,        NULL,
    026,  "nē",           LAMBDA,     1, -1, -1,        NULL,
    027,  "nezin",        LAMBDA,     4, -1, -1,        NULL,
    030,  "tad",          LAMBDA,     2, -1, -1,        NULL,
    031,  "tikai",        LAMBDA,     4, -1, -1,        NULL,
    032,  "vis",          LAMBDA,     2, -1, -1,        NULL,
    033,  "droši",        LAMBDA,     4, -1, -1,        NULL,
    034,  "diemžēl",      LAMBDA,     5, -1, -1,        NULL,
    035,  "tak",          LAMBDA,     2, -1, -1,        NULL,
    036,  "nebūt",        LAMBDA,     4, -1, -1,        NULL,
    037,  "varbūt",       LAMBDA,     5, -1, -1,        NULL,
    040,  "klau",         LAMBDA,     3, -1, -1,        NULL,
    041,  "lūk",          LAMBDA,     2, -1, -1,        NULL,
    042,  "iekams",       LAMBDA,     5, -1, -1,        NULL,
    043,  "vien",         LAMBDA,     3, -1, -1,        NULL,
    044,  "es",           LAMBDA,     1, -1, -1,        NULL,
    045,  "manis",        LAMBDA,     4, -1, -1,        NULL,
    046,  "man",          LAMBDA,     2, -1, -1,        NULL,
    047,  "mani",         LAMBDA,     3, -1, -1,        NULL,
    050,  "manī",         LAMBDA,     3, -1, -1,        NULL,
    051,  "mēs",          LAMBDA,     2, -1, -1,        NULL,
    052,  "mūsu",         LAMBDA,     3, -1, -1,        NULL,
    053,  "mums",         LAMBDA,     3, -1, -1,        NULL,
    054,  "mūs",          LAMBDA,     2, -1, -1,        NULL,
    055,  "mūsos",        LAMBDA,     4, -1, -1,        NULL,
    056,  "tu",           LAMBDA,     1, -1, -1,        NULL,
    057,  "tevis",        LAMBDA,     4, -1, -1,        NULL,
    060,  "tev",          LAMBDA,     2, -1, -1,        NULL,
    061,  "tevi",         LAMBDA,     3, -1, -1,        NULL,
    062,  "tevī",         LAMBDA,     3, -1, -1,        NULL,
    063,  "jūs",          LAMBDA,     2, -1, -1,        NULL,
    064,  "jūsu",         LAMBDA,     3, -1, -1,        NULL,
    065,  "jums",         LAMBDA,     3, -1, -1,        NULL,
    066,  "jūsos",        LAMBDA,     4, -1, -1,        NULL,
    067,  "viņš",         LAMBDA,     3, -1, -1,        NULL,
    070,  "viņa",         LAMBDA,     3, -1, -1,        NULL,
    071,  "viņam",        LAMBDA,     4, -1, -1,        NULL,
    072,  "viņu",         LAMBDA,     3, -1, -1,        NULL,
    073,  "viņā",         LAMBDA,     3, -1, -1,        NULL,
    074,  "viņi",         LAMBDA,     3, -1, -1,        NULL,
    075,  "viņiem",       LAMBDA,     5, -1, -1,        NULL,
    076,  "viņus",        LAMBDA,     4, -1, -1,        NULL,
    077,  "viņos",        LAMBDA,     4, -1, -1,        NULL,
    000,  NULL,           NULL,       0,  0,  0,        NULL,
    };

static RuleList step0c_rules[] =
    {
    001,  "viņas",        LAMBDA,     4, -1, -1,        NULL,
    002,  "viņai",        LAMBDA,     4, -1, -1,        NULL,
    003,  "viņām",        LAMBDA,     4, -1, -1,        NULL,
    004,  "viņās",        LAMBDA,     4, -1, -1,        NULL,
```

```
005,  "kurš",      LAMBDA,    3, -1, -1,    NULL,
006,  "kura",      LAMBDA,    3, -1, -1,    NULL,
007,  "kuram",     LAMBDA,    4, -1, -1,    NULL,
010,  "kuru",      LAMBDA,    3, -1, -1,    NULL,
011,  "kurā",      LAMBDA,    3, -1, -1,    NULL,
012,  "kuri",      LAMBDA,    3, -1, -1,    NULL,
013,  "kuriem",    LAMBDA,    5, -1, -1,    NULL,
014,  "kurus",     LAMBDA,    4, -1, -1,    NULL,
015,  "kuros",     LAMBDA,    4, -1, -1,    NULL,
016,  "kuras",     LAMBDA,    4, -1, -1,    NULL,
017,  "kurai",     LAMBDA,    4, -1, -1,    NULL,
020,  "kurām",     LAMBDA,    4, -1, -1,    NULL,
021,  "kurās",     LAMBDA,    4, -1, -1,    NULL,
022,  "viss",      LAMBDA,    3, -1, -1,    NULL,
023,  "visa",      LAMBDA,    3, -1, -1,    NULL,
024,  "visam",     LAMBDA,    4, -1, -1,    NULL,
025,  "visu",      LAMBDA,    3, -1, -1,    NULL,
026,  "visā",      LAMBDA,    3, -1, -1,    NULL,
027,  "visi",      LAMBDA,    3, -1, -1,    NULL,
030,  "visiem",    LAMBDA,    5, -1, -1,    NULL,
031,  "visus",     LAMBDA,    4, -1, -1,    NULL,
032,  "visos",     LAMBDA,    4, -1, -1,    NULL,
033,  "visas",     LAMBDA,    4, -1, -1,    NULL,
034,  "visai",     LAMBDA,    4, -1, -1,    NULL,
035,  "visām",     LAMBDA,    4, -1, -1,    NULL,
036,  "visās",     LAMBDA,    4, -1, -1,    NULL,
037,  "sevis",     LAMBDA,    4, -1, -1,    NULL,
040,  "sev",       LAMBDA,    2, -1, -1,    NULL,
041,  "sevi",      LAMBDA,    3, -1, -1,    NULL,
042,  "sevī",      LAMBDA,    3, -1, -1,    NULL,
043,  "kas",       LAMBDA,    2, -1, -1,    NULL,
044,  "kā",        LAMBDA,    1, -1, -1,    NULL,
045,  "kam",       LAMBDA,    2, -1, -1,    NULL,
046,  "ko",        LAMBDA,    1, -1, -1,    NULL,
047,  "kur",       LAMBDA,    2, -1, -1,    NULL,
050,  "tas",       LAMBDA,    2, -1, -1,    NULL,
051,  "tā",        LAMBDA,    1, -1, -1,    NULL,
052,  "tam",       LAMBDA,    2, -1, -1,    NULL,
053,  "to",        LAMBDA,    1, -1, -1,    NULL,
054,  "tajā",      LAMBDA,    3, -1, -1,    NULL,
055,  "tai",       LAMBDA,    2, -1, -1,    NULL,
056,  "tanī",      LAMBDA,    3, -1, -1,    NULL,
057,  "tās",       LAMBDA,    3, -1, -1,    NULL,
060,  "tie",       LAMBDA,    2, -1, -1,    NULL,
061,  "tiem",      LAMBDA,    3, -1, -1,    NULL,
062,  "tos",       LAMBDA,    2, -1, -1,    NULL,
063,  "tais",      LAMBDA,    3, -1, -1,    NULL,
064,  "tajos",     LAMBDA,    4, -1, -1,    NULL,
065,  "tanīs",     LAMBDA,    4, -1, -1,    NULL,
066,  "tām",       LAMBDA,    2, -1, -1,    NULL,
067,  "tajās",     LAMBDA,    4, -1, -1,    NULL,
070,  "šis",       LAMBDA,    2, -1, -1,    NULL,
071,  "šī",        LAMBDA,    1, -1, -1,    NULL,
072,  "šā",        LAMBDA,    1, -1, -1,    NULL,
073,  "šim",       LAMBDA,    2, -1, -1,    NULL,
074,  "šo",        LAMBDA,    1, -1, -1,    NULL,
075,  "šai",       LAMBDA,    2, -1, -1,    NULL,
076,  "šajā",      LAMBDA,    3, -1, -1,    NULL,
```

```
    077,  "šinī",        LAMBDA,     3, -1, -1,      NULL,
    000,  NULL,          NULL,       0,  0,  0,      NULL,
  };

static RuleList step0d_rules[] =
  {
    001,  "šīs",         LAMBDA,     2, -1, -1,      NULL,
    002,  "šās",         LAMBDA,     2, -1, -1,      NULL,
    003,  "šie",         LAMBDA,     2, -1, -1,      NULL,
    004,  "šiem",        LAMBDA,     3, -1, -1,      NULL,
    005,  "šām",         LAMBDA,     2, -1, -1,      NULL,
    006,  "šos",         LAMBDA,     2, -1, -1,      NULL,
    007,  "šais",        LAMBDA,     3, -1, -1,      NULL,
    010,  "šajos",       LAMBDA,     4, -1, -1,      NULL,
    011,  "šajās",       LAMBDA,     4, -1, -1,      NULL,
    012,  "šinīs",       LAMBDA,     4, -1, -1,      NULL,
    013,  "mans",        LAMBDA,     3, -1, -1,      NULL,
    014,  "mana",        LAMBDA,     3, -1, -1,      NULL,
    015,  "manam",       LAMBDA,     4, -1, -1,      NULL,
    016,  "manu",        LAMBDA,     3, -1, -1,      NULL,
    017,  "manī",        LAMBDA,     3, -1, -1,      NULL,
    021,  "maniem",      LAMBDA,     5, -1, -1,      NULL,
    022,  "manus",       LAMBDA,     4, -1, -1,      NULL,
    023,  "manos",       LAMBDA,     4, -1, -1,      NULL,
    024,  "manas",       LAMBDA,     4, -1, -1,      NULL,
    025,  "manai",       LAMBDA,     4, -1, -1,      NULL,
    026,  "manām",       LAMBDA,     4, -1, -1,      NULL,
    027,  "manās",       LAMBDA,     4, -1, -1,      NULL,
    030,  "tavs",        LAMBDA,     3, -1, -1,      NULL,
    031,  "tava",        LAMBDA,     3, -1, -1,      NULL,
    032,  "tavam",       LAMBDA,     4, -1, -1,      NULL,
    033,  "tavu",        LAMBDA,     3, -1, -1,      NULL,
    034,  "tavā",        LAMBDA,     3, -1, -1,      NULL,
    035,  "tavi",        LAMBDA,     3, -1, -1,      NULL,
    036,  "taviem",      LAMBDA,     5, -1, -1,      NULL,
    037,  "tavus",       LAMBDA,     4, -1, -1,      NULL,
    040,  "tavos",       LAMBDA,     4, -1, -1,      NULL,
    041,  "tavas",       LAMBDA,     4, -1, -1,      NULL,
    042,  "tavai",       LAMBDA,     4, -1, -1,      NULL,
    043,  "tavām",       LAMBDA,     4, -1, -1,      NULL,
    044,  "tavās",       LAMBDA,     4, -1, -1,      NULL,
    045,  "savs",        LAMBDA,     3, -1, -1,      NULL,
    046,  "sava",        LAMBDA,     3, -1, -1,      NULL,
    047,  "savam",       LAMBDA,     4, -1, -1,      NULL,
    050,  "savu",        LAMBDA,     3, -1, -1,      NULL,
    051,  "savā",        LAMBDA,     3, -1, -1,      NULL,
    052,  "savi",        LAMBDA,     3, -1, -1,      NULL,
    053,  "saviem",      LAMBDA,     5, -1, -1,      NULL,
    054,  "savus",       LAMBDA,     4, -1, -1,      NULL,
    055,  "savos",       LAMBDA,     4, -1, -1,      NULL,
    056,  "savas",       LAMBDA,     4, -1, -1,      NULL,
    057,  "savai",       LAMBDA,     4, -1, -1,      NULL,
    060,  "savām",       LAMBDA,     4, -1, -1,      NULL,
    061,  "savās",       LAMBDA,     4, -1, -1,      NULL,
    062,  "cits",        LAMBDA,     3, -1, -1,      NULL,
    063,  "cita",        LAMBDA,     3, -1, -1,      NULL,
    064,  "citam",       LAMBDA,     4, -1, -1,      NULL,
    065,  "citu",        LAMBDA,     3, -1, -1,      NULL,
```

```
066,  "citā",       LAMBDA,     3, -1, -1,      NULL,
067,  "citi",       LAMBDA,     3, -1, -1,      NULL,
070,  "citiem",     LAMBDA,     5, -1, -1,      NULL,
071,  "citus",      LAMBDA,     4, -1, -1,      NULL,
072,  "citos",      LAMBDA,     4, -1, -1,      NULL,
073,  "citas",      LAMBDA,     4, -1, -1,      NULL,
074,  "citai",      LAMBDA,     4, -1, -1,      NULL,
075,  "citām",      LAMBDA,     4, -1, -1,      NULL,
076,  "citās",      LAMBDA,     4, -1, -1,      NULL,
077,  "dažs",       LAMBDA,     3, -1, -1,      NULL,
000,  NULL,         NULL,       0,  0,  0,      NULL,
        };

static RuleList step0e_rules[] =
        {
         001,  "daža",      LAMBDA,     3, -1, -1,      NULL,
         002,  "dažam",     LAMBDA,     4, -1, -1,      NULL,
         003,  "dažu",      LAMBDA,     3, -1, -1,      NULL,
         004,  "dažā",      LAMBDA,     3, -1, -1,      NULL,
         005,  "daži",      LAMBDA,     3, -1, -1,      NULL,
         006,  "dažiem",    LAMBDA,     5, -1, -1,      NULL,
         007,  "dažus",     LAMBDA,     4, -1, -1,      NULL,
         010,  "dažos",     LAMBDA,     4, -1, -1,      NULL,
         011,  "dažas",     LAMBDA,     4, -1, -1,      NULL,
         012,  "dažai",     LAMBDA,     4, -1, -1,      NULL,
         013,  "dažām",     LAMBDA,     4, -1, -1,      NULL,
         014,  "dažās",     LAMBDA,     4, -1, -1,      NULL,
         015,  "kāds",      LAMBDA,     3, -1, -1,      NULL,
         016,  "kāda",      LAMBDA,     3, -1, -1,      NULL,
         017,  "kādam",     LAMBDA,     4, -1, -1,      NULL,
         020,  "kādu",      LAMBDA,     3, -1, -1,      NULL,
         021,  "kādā",      LAMBDA,     3, -1, -1,      NULL,
         022,  "kādi",      LAMBDA,     3, -1, -1,      NULL,
         023,  "kādiem",    LAMBDA,     5, -1, -1,      NULL,
         024,  "kādus",     LAMBDA,     4, -1, -1,      NULL,
         025,  "kādos",     LAMBDA,     4, -1, -1,      NULL,
         026,  "kādas",     LAMBDA,     4, -1, -1,      NULL,
         027,  "kādai",     LAMBDA,     4, -1, -1,      NULL,
         030,  "kādām",     LAMBDA,     4, -1, -1,      NULL,
         031,  "kādās",     LAMBDA,     4, -1, -1,      NULL,
         032,  "kurš",      LAMBDA,     3, -1, -1,      NULL,
         033,  "kura",      LAMBDA,     3, -1, -1,      NULL,
         034,  "kuram",     LAMBDA,     4, -1, -1,      NULL,
         035,  "kuru",      LAMBDA,     3, -1, -1,      NULL,
         036,  "kurā",      LAMBDA,     3, -1, -1,      NULL,
         037,  "kuri",      LAMBDA,     3, -1, -1,      NULL,
         040,  "kuriem",    LAMBDA,     5, -1, -1,      NULL,
         041,  "kurus",     LAMBDA,     4, -1, -1,      NULL,
         042,  "kuros",     LAMBDA,     4, -1, -1,      NULL,
         043,  "kuras",     LAMBDA,     4, -1, -1,      NULL,
         044,  "kurai",     LAMBDA,     4, -1, -1,      NULL,
         045,  "kurām",     LAMBDA,     4, -1, -1,      NULL,
         046,  "kurās",     LAMBDA,     4, -1, -1,      NULL,
         047,  "tāds",      LAMBDA,     3, -1, -1,      NULL,
         050,  "tāda",      LAMBDA,     3, -1, -1,      NULL,
         051,  "tādam",     LAMBDA,     4, -1, -1,      NULL,
         052,  "tādu",      LAMBDA,     3, -1, -1,      NULL,
         053,  "tādā",      LAMBDA,     3, -1, -1,      NULL,
```

```
    054,  "tādi",        LAMBDA,    3, -1, -1,      NULL,
    055,  "tādiem",      LAMBDA,    5, -1, -1,      NULL,
    056,  "tādus",       LAMBDA,    4, -1, -1,      NULL,
    057,  "tādos",       LAMBDA,    4, -1, -1,      NULL,
    060,  "tādas",       LAMBDA,    4, -1, -1,      NULL,
    061,  "tādai",       LAMBDA,    4, -1, -1,      NULL,
    062,  "tādām",       LAMBDA,    4, -1, -1,      NULL,
    063,  "tādās",       LAMBDA,    4, -1, -1,      NULL,
    064,  "šāds",        LAMBDA,    3, -1, -1,      NULL,
    065,  "šāda",        LAMBDA,    3, -1, -1,      NULL,
    066,  "šādam",       LAMBDA,    4, -1, -1,      NULL,
    067,  "šādu",        LAMBDA,    3, -1, -1,      NULL,
    070,  "šādā",        LAMBDA,    3, -1, -1,      NULL,
    071,  "šādi",        LAMBDA,    3, -1, -1,      NULL,
    072,  "šādiem",      LAMBDA,    5, -1, -1,      NULL,
    073,  "šādus",       LAMBDA,    4, -1, -1,      NULL,
    074,  "šādos",       LAMBDA,    4, -1, -1,      NULL,
    075,  "šādas",       LAMBDA,    4, -1, -1,      NULL,
    076,  "šādai",       LAMBDA,    4, -1, -1,      NULL,
    077,  "šādām",       LAMBDA,    4, -1, -1,      NULL,
    000,  NULL,          NULL,      0,  0,  0,      NULL,
  };

static RuleList step0f_rules[] =
  {
    001,  "šādās",       LAMBDA,    4, -1, -1,      NULL,
    002,  "katrs",       LAMBDA,    4, -1, -1,      NULL,
    003,  "katra",       LAMBDA,    4, -1, -1,      NULL,
    004,  "katram",      LAMBDA,    5, -1, -1,      NULL,
    005,  "katru",       LAMBDA,    4, -1, -1,      NULL,
    006,  "katrā",       LAMBDA,    4, -1, -1,      NULL,
    007,  "katri",       LAMBDA,    4, -1, -1,      NULL,
    010,  "katriem",     LAMBDA,    6, -1, -1,      NULL,
    011,  "katrus",      LAMBDA,    5, -1, -1,      NULL,
    012,  "katros",      LAMBDA,    5, -1, -1,      NULL,
    013,  "katras",      LAMBDA,    5, -1, -1,      NULL,
    014,  "katrai",      LAMBDA,    5, -1, -1,      NULL,
    015,  "katrām",      LAMBDA,    5, -1, -1,      NULL,
    016,  "katrās",      LAMBDA,    5, -1, -1,      NULL,
    017,  "manējs",      LAMBDA,    5, -1, -1,      NULL,
    020,  "manēja",      LAMBDA,    5, -1, -1,      NULL,
    021,  "manējam",     LAMBDA,    6, -1, -1,      NULL,
    022,  "manēju",      LAMBDA,    5, -1, -1,      NULL,
    023,  "manējā",      LAMBDA,    5, -1, -1,      NULL,
    024,  "manēji",      LAMBDA,    5, -1, -1,      NULL,
    025,  "manējiem",    LAMBDA,    7, -1, -1,      NULL,
    026,  "manējus",     LAMBDA,    6, -1, -1,      NULL,
    027,  "manējos",     LAMBDA,    6, -1, -1,      NULL,
    030,  "manējais",    LAMBDA,    7, -1, -1,      NULL,
    032,  "manējo",      LAMBDA,    5, -1, -1,      NULL,
    033,  "manējie",     LAMBDA,    6, -1, -1,      NULL,
    034,  "manējās",     LAMBDA,    6, -1, -1,      NULL,
    035,  "manējai",     LAMBDA,    6, -1, -1,      NULL,
    036,  "manējām",     LAMBDA,    6, -1, -1,      NULL,
    037,  "manējas",     LAMBDA,    6, -1, -1,      NULL,
    040,  "tavējs",      LAMBDA,    5, -1, -1,      NULL,
    041,  "tavēja",      LAMBDA,    5, -1, -1,      NULL,
    042,  "tavējam",     LAMBDA,    6, -1, -1,      NULL,
```

```
  043,  "tavēju",      LAMBDA,     5, -1, -1,      NULL,
  044,  "tavējā",      LAMBDA,     5, -1, -1,      NULL,
  045,  "tavēji",      LAMBDA,     5, -1, -1,      NULL,
  046,  "tavējiem",    LAMBDA,     7, -1, -1,      NULL,
  047,  "tavējus",     LAMBDA,     6, -1, -1,      NULL,
  050,  "tavējos",     LAMBDA,     6, -1, -1,      NULL,
  051,  "tavējais",    LAMBDA,     7, -1, -1,      NULL,
  053,  "tavējo",      LAMBDA,     5, -1, -1,      NULL,
  054,  "tavējie",     LAMBDA,     6, -1, -1,      NULL,
  055,  "tavējās",     LAMBDA,     6, -1, -1,      NULL,
  056,  "tavējai",     LAMBDA,     6, -1, -1,      NULL,
  057,  "tavējām",     LAMBDA,     6, -1, -1,      NULL,
  060,  "tavējas",     LAMBDA,     6, -1, -1,      NULL,
  061,  "savējs",      LAMBDA,     5, -1, -1,      NULL,
  062,  "savēja",      LAMBDA,     5, -1, -1,      NULL,
  063,  "savējam",     LAMBDA,     6, -1, -1,      NULL,
  064,  "savēju",      LAMBDA,     5, -1, -1,      NULL,
  065,  "savējā",      LAMBDA,     5, -1, -1,      NULL,
  066,  "savēji",      LAMBDA,     5, -1, -1,      NULL,
  067,  "savējiem",    LAMBDA,     7, -1, -1,      NULL,
  070,  "savējus",     LAMBDA,     6, -1, -1,      NULL,
  071,  "savējos",     LAMBDA,     6, -1, -1,      NULL,
  072,  "savējais",    LAMBDA,     7, -1, -1,      NULL,
  074,  "savējo",      LAMBDA,     5, -1, -1,      NULL,
  075,  "savējie",     LAMBDA,     6, -1, -1,      NULL,
  076,  "savējās",     LAMBDA,     6, -1, -1,      NULL,
  077,  "savējai",     LAMBDA,     6, -1, -1,      NULL,
  000,  NULL,          NULL,       0,  0,  0,      NULL,
};

static RuleList step0g_rules[] =
        {
  001,  "savējām",     LAMBDA,     6, -1, -1,      NULL,
  002,  "savējas",     LAMBDA,     6, -1, -1,      NULL,
  003,  "viņējs",      LAMBDA,     5, -1, -1,      NULL,
  004,  "viņēja",      LAMBDA,     5, -1, -1,      NULL,
  005,  "viņējam",     LAMBDA,     6, -1, -1,      NULL,
  006,  "viņēju",      LAMBDA,     5, -1, -1,      NULL,
  007,  "viņējā",      LAMBDA,     5, -1, -1,      NULL,
  010,  "viņēji",      LAMBDA,     5, -1, -1,      NULL,
  011,  "viņējiem",    LAMBDA,     7, -1, -1,      NULL,
  012,  "viņējus",     LAMBDA,     6, -1, -1,      NULL,
  013,  "viņējos",     LAMBDA,     6, -1, -1,      NULL,
  014,  "viņējais",    LAMBDA,     7, -1, -1,      NULL,
  016,  "viņējo",      LAMBDA,     5, -1, -1,      NULL,
  017,  "viņējie",     LAMBDA,     6, -1, -1,      NULL,
  020,  "viņējās",     LAMBDA,     6, -1, -1,      NULL,
  021,  "viņējai",     LAMBDA,     6, -1, -1,      NULL,
  022,  "viņējām",     LAMBDA,     6, -1, -1,      NULL,
  023,  "viņējas",     LAMBDA,     6, -1, -1,      NULL,
  024,  "jūsējs",      LAMBDA,     5, -1, -1,      NULL,
  025,  "jūsēja",      LAMBDA,     5, -1, -1,      NULL,
  026,  "jūsējam",     LAMBDA,     6, -1, -1,      NULL,
  027,  "jūsēju",      LAMBDA,     5, -1, -1,      NULL,
  030,  "jūsējā",      LAMBDA,     5, -1, -1,      NULL,
  031,  "jūsēji",      LAMBDA,     5, -1, -1,      NULL,
  032,  "jūsējiem",    LAMBDA,     7, -1, -1,      NULL,
  033,  "jūsējus",     LAMBDA,     6, -1, -1,      NULL,
```

```
  034,  "jūsējos",      LAMBDA,     6, -1, -1,      NULL,
  035,  "jūsējais",     LAMBDA,     7, -1, -1,      NULL,
  037,  "jūsējo",       LAMBDA,     5, -1, -1,      NULL,
  040,  "jūsējie",      LAMBDA,     6, -1, -1,      NULL,
  041,  "jūsējās",      LAMBDA,     6, -1, -1,      NULL,
  042,  "jūsējai",      LAMBDA,     6, -1, -1,      NULL,
  043,  "jūsējām",      LAMBDA,     6, -1, -1,      NULL,
  044,  "jūsējas",      LAMBDA,     6, -1, -1,      NULL,
  045,  "mūsējs",       LAMBDA,     5, -1, -1,      NULL,
  046,  "mūsēja",       LAMBDA,     5, -1, -1,      NULL,
  047,  "mūsējam",      LAMBDA,     6, -1, -1,      NULL,
  050,  "mūsēju",       LAMBDA,     5, -1, -1,      NULL,
  051,  "mūsējā",       LAMBDA,     5, -1, -1,      NULL,
  052,  "mūsēji",       LAMBDA,     5, -1, -1,      NULL,
  053,  "mūsējiem",     LAMBDA,     7, -1, -1,      NULL,
  054,  "mūsējus",      LAMBDA,     6, -1, -1,      NULL,
  055,  "mūsējos",      LAMBDA,     6, -1, -1,      NULL,
  056,  "mūsējais",     LAMBDA,     7, -1, -1,      NULL,
  060,  "mūsējo",       LAMBDA,     5, -1, -1,      NULL,
  061,  "mūsējie",      LAMBDA,     6, -1, -1,      NULL,
  062,  "mūsējās",      LAMBDA,     6, -1, -1,      NULL,
  063,  "mūsējai",      LAMBDA,     6, -1, -1,      NULL,
  064,  "mūsējām",      LAMBDA,     6, -1, -1,      NULL,
  065,  "mūsējas",      LAMBDA,     6, -1, -1,      NULL,
  066,  "šitas",        LAMBDA,     4, -1, -1,      NULL,
  067,  "šitā",         LAMBDA,     3, -1, -1,      NULL,
  070,  "šitam",        LAMBDA,     4, -1, -1,      NULL,
  071,  "šito",         LAMBDA,     3, -1, -1,      NULL,
  072,  "šitai",        LAMBDA,     4, -1, -1,      NULL,
  073,  "šitie",        LAMBDA,     4, -1, -1,      NULL,
  074,  "šitiem",       LAMBDA,     5, -1, -1,      NULL,
  075,  "šitos",        LAMBDA,     4, -1, -1,      NULL,
  076,  "šitās",        LAMBDA,     4, -1, -1,      NULL,
  077,  "šitām",        LAMBDA,     4, -1, -1,      NULL,
  000,  NULL,           NULL,       0,  0,  0,      NULL,
 };

static RuleList step0h_rules[] =
 {
  001,  "šitāds",       LAMBDA,     5, -1, -1,      NULL,
  002,  "šitāda",       LAMBDA,     5, -1, -1,      NULL,
  003,  "šitādam",      LAMBDA,     6, -1, -1,      NULL,
  004,  "šitādu",       LAMBDA,     5, -1, -1,      NULL,
  005,  "šitādā",       LAMBDA,     5, -1, -1,      NULL,
  006,  "šitādas",      LAMBDA,     6, -1, -1,      NULL,
  007,  "šitādai",      LAMBDA,     6, -1, -1,      NULL,
  010,  "šitādi",       LAMBDA,     5, -1, -1,      NULL,
  011,  "šitādiem",     LAMBDA,     7, -1, -1,      NULL,
  012,  "šitādus",      LAMBDA,     6, -1, -1,      NULL,
  013,  "šitādos",      LAMBDA,     6, -1, -1,      NULL,
  014,  "šitādām",      LAMBDA,     6, -1, -1,      NULL,
  015,  "šitādās",      LAMBDA,     6, -1, -1,      NULL,
  016,  "ikkatrs",      LAMBDA,     6, -1, -1,      NULL,
  017,  "ikkatra",      LAMBDA,     6, -1, -1,      NULL,
  020,  "ikkatram",     LAMBDA,     7, -1, -1,      NULL,
  021,  "ikkatru",      LAMBDA,     6, -1, -1,      NULL,
  022,  "ikkatrā",      LAMBDA,     6, -1, -1,      NULL,
  023,  "ikkatras",     LAMBDA,     7, -1, -1,      NULL,
```

```
    024,  "ikkatrai",     LAMBDA,     7, -1, -1,     NULL,
    025,  "ikkatri",      LAMBDA,     6, -1, -1,     NULL,
    026,  "ikkatriem",    LAMBDA,     8, -1, -1,     NULL,
    027,  "ikkatrus",     LAMBDA,     7, -1, -1,     NULL,
    030,  "ikkatros",     LAMBDA,     7, -1, -1,     NULL,
    032,  "ikkatrām",     LAMBDA,     7, -1, -1,     NULL,
    033,  "ikkatrās",     LAMBDA,     7, -1, -1,     NULL,
    034,  "jebkāds",      LAMBDA,     6, -1, -1,     NULL,
    035,  "jebkāda",      LAMBDA,     6, -1, -1,     NULL,
    036,  "jebkādam",     LAMBDA,     7, -1, -1,     NULL,
    037,  "jebkādu",      LAMBDA,     6, -1, -1,     NULL,
    040,  "jebkādā",      LAMBDA,     6, -1, -1,     NULL,
    041,  "jebkādas",     LAMBDA,     7, -1, -1,     NULL,
    042,  "jebkādai",     LAMBDA,     7, -1, -1,     NULL,
    043,  "jebkādi",      LAMBDA,     6, -1, -1,     NULL,
    044,  "jebkādiem",    LAMBDA,     8, -1, -1,     NULL,
    045,  "jebkādus",     LAMBDA,     7, -1, -1,     NULL,
    046,  "jebkādos",     LAMBDA,     7, -1, -1,     NULL,
    047,  "jebkādām",     LAMBDA,     7, -1, -1,     NULL,
    050,  "jebkādās",     LAMBDA,     7, -1, -1,     NULL,
    051,  "jebkas",       LAMBDA,     5, -1, -1,     NULL,
    053,  "jebkā",        LAMBDA,     4, -1, -1,     NULL,
    054,  "jebkam",       LAMBDA,     5, -1, -1,     NULL,
    055,  "jebko",        LAMBDA,     4, -1, -1,     NULL,
    056,  "jebkurš",      LAMBDA,     6, -1, -1,     NULL,
    057,  "jebkura",      LAMBDA,     6, -1, -1,     NULL,
    060,  "jebkuram",     LAMBDA,     7, -1, -1,     NULL,
    061,  "jebkuru",      LAMBDA,     6, -1, -1,     NULL,
    062,  "jebkurā",      LAMBDA,     6, -1, -1,     NULL,
    063,  "jebkuras",     LAMBDA,     7, -1, -1,     NULL,
    064,  "jebkurai",     LAMBDA,     7, -1, -1,     NULL,
    065,  "jebkuri",      LAMBDA,     6, -1, -1,     NULL,
    066,  "jebkuriem",    LAMBDA,     8, -1, -1,     NULL,
    067,  "jebkurus",     LAMBDA,     7, -1, -1,     NULL,
    070,  "jebkuros",     LAMBDA,     7, -1, -1,     NULL,
    071,  "jebkurām",     LAMBDA,     7, -1, -1,     NULL,
    072,  "jebkurās",     LAMBDA,     7, -1, -1,     NULL,
    074,  "ikkurš",       LAMBDA,     5, -1, -1,     NULL,
    075,  "ikkura",       LAMBDA,     5, -1, -1,     NULL,
    076,  "ikkuram",      LAMBDA,     6, -1, -1,     NULL,
    077,  "ikkuru",       LAMBDA,     5, -1, -1,     NULL,
    000,  NULL,           NULL,       0,  0,  0,     NULL,
   };

static RuleList step0i_rules[] =
   {
    001,  "ikkurā",       LAMBDA,     5, -1, -1,     NULL,
    002,  "ikkuras",      LAMBDA,     6, -1, -1,     NULL,
    003,  "ikkurai",      LAMBDA,     6, -1, -1,     NULL,
    004,  "ikkuri",       LAMBDA,     5, -1, -1,     NULL,
    005,  "ikkuriem",     LAMBDA,     7, -1, -1,     NULL,
    006,  "ikkurus",      LAMBDA,     6, -1, -1,     NULL,
    007,  "ikkuros",      LAMBDA,     6, -1, -1,     NULL,
    010,  "ikkurām",      LAMBDA,     6, -1, -1,     NULL,
    011,  "ikkurās",      LAMBDA,     6, -1, -1,     NULL,
    012,  "ikviens",      LAMBDA,     6, -1, -1,     NULL,
    013,  "ikviena",      LAMBDA,     6, -1, -1,     NULL,
    014,  "ikvienam",     LAMBDA,     7, -1, -1,     NULL,
```

```
  016,  "ikvienu",      LAMBDA,     6, -1, -1,      NULL,
  017,  "ikvienā",      LAMBDA,     6, -1, -1,      NULL,
  020,  "ikvienas",     LAMBDA,     7, -1, -1,      NULL,
  021,  "ikvienai",     LAMBDA,     7, -1, -1,      NULL,
  022,  "ikvieni",      LAMBDA,     6, -1, -1,      NULL,
  023,  "ikvieniem",    LAMBDA,     8, -1, -1,      NULL,
  024,  "ikvienus",     LAMBDA,     7, -1, -1,      NULL,
  025,  "ikvienos",     LAMBDA,     7, -1, -1,      NULL,
  026,  "ikvienām",     LAMBDA,     7, -1, -1,      NULL,
  027,  "ikvienās",     LAMBDA,     7, -1, -1,      NULL,
  030,  "nekas",        LAMBDA,     4, -1, -1,      NULL,
  031,  "nekā",         LAMBDA,     3, -1, -1,      NULL,
  032,  "nekam",        LAMBDA,     4, -1, -1,      NULL,
  033,  "neko",         LAMBDA,     3, -1, -1,      NULL,
  034,  "nekāds",       LAMBDA,     5, -1, -1,      NULL,
  035,  "nekāda",       LAMBDA,     5, -1, -1,      NULL,
  037,  "nekādam",      LAMBDA,     6, -1, -1,      NULL,
  040,  "nekādu",       LAMBDA,     5, -1, -1,      NULL,
  041,  "nekādā",       LAMBDA,     5, -1, -1,      NULL,
  042,  "nekādas",      LAMBDA,     6, -1, -1,      NULL,
  043,  "nekādai",      LAMBDA,     6, -1, -1,      NULL,
  044,  "nekādi",       LAMBDA,     5, -1, -1,      NULL,
  045,  "nekādiem",     LAMBDA,     7, -1, -1,      NULL,
  046,  "nekādus",      LAMBDA,     6, -1, -1,      NULL,
  047,  "nekādos",      LAMBDA,     6, -1, -1,      NULL,
  050,  "nekādām",      LAMBDA,     6, -1, -1,      NULL,
  051,  "nekādās",      LAMBDA,     6, -1, -1,      NULL,
  052,  "neviens",      LAMBDA,     6, -1, -1,      NULL,
  053,  "neviena",      LAMBDA,     6, -1, -1,      NULL,
  054,  "nevienam",     LAMBDA,     7, -1, -1,      NULL,
  055,  "nevienu",      LAMBDA,     6, -1, -1,      NULL,
  056,  "nevienā",      LAMBDA,     6, -1, -1,      NULL,
  060,  "nevienas",     LAMBDA,     6, -1, -1,      NULL,
  061,  "nevienai",     LAMBDA,     7, -1, -1,      NULL,
  062,  "nevieni",      LAMBDA,     6, -1, -1,      NULL,
  063,  "nevieniem",    LAMBDA,     8, -1, -1,      NULL,
  064,  "nevienus",     LAMBDA,     7, -1, -1,      NULL,
  065,  "nevienos",     LAMBDA,     7, -1, -1,      NULL,
  066,  "nevienām",     LAMBDA,     7, -1, -1,      NULL,
  067,  "nevienās",     LAMBDA,     7, -1, -1,      NULL,
  070,  "pats",         LAMBDA,     3, -1, -1,      NULL,
  071,  "paša",         LAMBDA,     3, -1, -1,      NULL,
  072,  "pašam",        LAMBDA,     4, -1, -1,      NULL,
  073,  "pašu",         LAMBDA,     3, -1, -1,      NULL,
  074,  "pašā",         LAMBDA,     3, -1, -1,      NULL,
  075,  "pati",         LAMBDA,     3, -1, -1,      NULL,
  076,  "pašas",        LAMBDA,     4, -1, -1,      NULL,
  077,  "pašai",        LAMBDA,     4, -1, -1,      NULL,
  000,  NULL,           NULL,       0,  0,  0,      NULL,
 };

static RuleList step0j_rules[] =
 {
  001,  "paši",         LAMBDA,     3, -1, -1,      NULL,
  002,  "pašiem",       LAMBDA,     5, -1, -1,      NULL,
  003,  "pašus",        LAMBDA,     4, -1, -1,      NULL,
  004,  "pašos",        LAMBDA,     4, -1, -1,      NULL,
  005,  "pašām",        LAMBDA,     4, -1, -1,      NULL,
```

```
006,  "pašās",      LAMBDA,     4, -1, -1,      NULL,
007,  "būt",        LAMBDA,     2, -1, -1,      NULL,
010,  "biju",       LAMBDA,     3, -1, -1,      NULL,
011,  "biji",       LAMBDA,     3, -1, -1,      NULL,
012,  "bija",       LAMBDA,     3, -1, -1,      NULL,
013,  "bijām",      LAMBDA,     4, -1, -1,      NULL,
014,  "bijāt",      LAMBDA,     4, -1, -1,      NULL,
015,  "esmu",       LAMBDA,     3, -1, -1,      NULL,
016,  "esi",        LAMBDA,     2, -1, -1,      NULL,
017,  "esam",       LAMBDA,     3, -1, -1,      NULL,
020,  "esat",       LAMBDA,     3, -1, -1,      NULL,
021,  "būšu",       LAMBDA,     3, -1, -1,      NULL,
022,  "būsi",       LAMBDA,     3, -1, -1,      NULL,
023,  "būs",        LAMBDA,     2, -1, -1,      NULL,
024,  "būsim",      LAMBDA,     4, -1, -1,      NULL,
025,  "būsiet",     LAMBDA,     5, -1, -1,      NULL,
026,  "tikt",       LAMBDA,     3, -1, -1,      NULL,
027,  "tiku",       LAMBDA,     3, -1, -1,      NULL,
030,  "tiki",       LAMBDA,     3, -1, -1,      NULL,
031,  "tika",       LAMBDA,     3, -1, -1,      NULL,
032,  "tikām",      LAMBDA,     4, -1, -1,      NULL,
033,  "tikāt",      LAMBDA,     4, -1, -1,      NULL,
034,  "tieku",      LAMBDA,     4, -1, -1,      NULL,
035,  "tiec",       LAMBDA,     3, -1, -1,      NULL,
037,  "tiek",       LAMBDA,     3, -1, -1,      NULL,
040,  "tiekam",     LAMBDA,     5, -1, -1,      NULL,
041,  "tiekat",     LAMBDA,     5, -1, -1,      NULL,
042,  "tikšu",      LAMBDA,     4, -1, -1,      NULL,
043,  "tiks",       LAMBDA,     3, -1, -1,      NULL,
044,  "tiksim",     LAMBDA,     5, -1, -1,      NULL,
045,  "tiksiet",    LAMBDA,     6, -1, -1,      NULL,
046,  "tapt",       LAMBDA,     3, -1, -1,      NULL,
047,  "tapi",       LAMBDA,     3, -1, -1,      NULL,
050,  "tapāt",      LAMBDA,     4, -1, -1,      NULL,
051,  "topat",      LAMBDA,     4, -1, -1,      NULL,
052,  "tapšu",      LAMBDA,     4, -1, -1,      NULL,
053,  "tapsi",      LAMBDA,     4, -1, -1,      NULL,
054,  "taps",       LAMBDA,     3, -1, -1,      NULL,
055,  "tapsim",     LAMBDA,     5, -1, -1,      NULL,
056,  "tapsiet",    LAMBDA,     6, -1, -1,      NULL,
057,  "kļūt",       LAMBDA,     3, -1, -1,      NULL,
060,  "kļuvu",      LAMBDA,     4, -1, -1,      NULL,
061,  "kļuvi",      LAMBDA,     4, -1, -1,      NULL,
062,  "kļuva",      LAMBDA,     4, -1, -1,      NULL,
063,  "kļuvām",     LAMBDA,     5, -1, -1,      NULL,
064,  "kļuvāt",     LAMBDA,     5, -1, -1,      NULL,
065,  "kļūstu",     LAMBDA,     5, -1, -1,      NULL,
066,  "kļūsti",     LAMBDA,     5, -1, -1,      NULL,
067,  "kļūst",      LAMBDA,     4, -1, -1,      NULL,
070,  "kļūstam",    LAMBDA,     6, -1, -1,      NULL,
071,  "kļūstat",    LAMBDA,     6, -1, -1,      NULL,
072,  "kļūšu",      LAMBDA,     4, -1, -1,      NULL,
073,  "kļūsi",      LAMBDA,     4, -1, -1,      NULL,
074,  "kļūs",       LAMBDA,     3, -1, -1,      NULL,
075,  "kļūsim",     LAMBDA,     5, -1, -1,      NULL,
076,  "kļūsiet",    LAMBDA,     6, -1, -1,      NULL,
000,  NULL,         NULL,       0,  0,  0,      NULL,
};
```

```
static RuleList step0k_rules[] =
        {
          001,  "ūja",          LAMBDA,     2, -1, -1,      NULL,
          002,  "urrā",         LAMBDA,     3, -1, -1,      NULL,
          003,  "urā",          LAMBDA,     2, -1, -1,      NULL,
          004,  "re",           LAMBDA,     1, -1, -1,      NULL,
          005,  "pag",          LAMBDA,     2, -1, -1,      NULL,
          006,  "raug",         LAMBDA,     3, -1, -1,      NULL,
          007,  "paraug",       LAMBDA,     5, -1, -1,      NULL,
          010,  "ai",           LAMBDA,     1, -1, -1,      NULL,
          011,  "ak",           LAMBDA,     1, -1, -1,      NULL,
          012,  "palūk",        LAMBDA,     4, -1, -1,      NULL,
          013,  "nudien",       LAMBDA,     5, -1, -1,      NULL,
          014,  "ekur",         LAMBDA,     3, -1, -1,      NULL,
          015,  "kuš",          LAMBDA,     2, -1, -1,      NULL,
          016,  "skat",         LAMBDA,     3, -1, -1,      NULL,
          017,  "paskat",       LAMBDA,     5, -1, -1,      NULL,
          020,  "paklau",       LAMBDA,     5, -1, -1,      NULL,
          021,  "ņau",          LAMBDA,     2, -1, -1,      NULL,
          022,  "rau",          LAMBDA,     2, -1, -1,      NULL,
          023,  "parau",        LAMBDA,     4, -1, -1,      NULL,
          024,  "nu",           LAMBDA,     1, -1, -1,      NULL,
          025,  "tpū",          LAMBDA,     2, -1, -1,      NULL,
          026,  "vau",          LAMBDA,     1, -1, -1,      NULL,
          027,  "redz",         LAMBDA,     3, -1, -1,      NULL,
          030,  "varēt",        LAMBDA,     4, -1, -1,      NULL,
          031,  "var",          LAMBDA,     2, -1, -1,      NULL,
          032,  "varat",        LAMBDA,     4, -1, -1,      NULL,
          033,  "varēju",       LAMBDA,     5, -1, -1,      NULL,
          034,  "varēji",       LAMBDA,     5, -1, -1,      NULL,
          035,  "varēja",       LAMBDA,     5, -1, -1,      NULL,
          036,  "varējām",      LAMBDA,     6, -1, -1,      NULL,
          037,  "varējāt",      LAMBDA,     6, -1, -1,      NULL,
          040,  "varēšu",       LAMBDA,     5, -1, -1,      NULL,
          041,  "varēsi",       LAMBDA,     5, -1, -1,      NULL,
          042,  "varēs",        LAMBDA,     4, -1, -1,      NULL,
          043,  "varēsim",      LAMBDA,     6, -1, -1,      NULL,
          044,  "varēsiet",     LAMBDA,     7, -1, -1,      NULL,
          045,  "kāpēc",        LAMBDA,     4, -1, -1,      NULL,
          046,  "kādēļ",        LAMBDA,     4, -1, -1,      NULL,
          047,  "kālab",        LAMBDA,     4, -1, -1,      NULL,
          050,  "tālab",        LAMBDA,     4, -1, -1,      NULL,
          051,  "kālabad",      LAMBDA,     6, -1, -1,      NULL,
          052,  "tālabad",      LAMBDA,     6, -1, -1,      NULL,
          053,  "kamdēļ",       LAMBDA,     5, -1, -1,      NULL,
          054,  "tamdēļ",       LAMBDA,     5, -1, -1,      NULL,
          055,  "bezgala",      LAMBDA,     6, -1, -1,      NULL,
          056,  "nenieka",      LAMBDA,     6, -1, -1,      NULL,
          057,  "samērā",       LAMBDA,     5, -1, -1,      NULL,
          060,  "vērā",         LAMBDA,     3, -1, -1,      NULL,
          061,  "visupēc",      LAMBDA,     6, -1, -1,      NULL,
          062,  "tagad",        LAMBDA,     4, -1, -1,      NULL,
          063,  "kad",          LAMBDA,     2, -1, -1,      NULL,
          064,  "jebkad",       LAMBDA,     5, -1, -1,      NULL,
          065,  "nekad",        LAMBDA,     4, -1, -1,      NULL,
          066,  "šad",          LAMBDA,     2, -1, -1,      NULL,
          067,  "še",           LAMBDA,     1, -1, -1,      NULL,
          070,  "tādējādi",     LAMBDA,     7, -1, -1,      NULL,
```

```
  071,  "visādi",       LAMBDA,     5, -1, -1,      NULL,
  072,  "visvisādi",    LAMBDA,     8, -1, -1,      NULL,
  073,  "citādi",       LAMBDA,     5, -1, -1,      NULL,
  074,  "parasti",      LAMBDA,     6, -1, -1,      NULL,
  075,  "dikti",        LAMBDA,     4, -1, -1,      NULL,
  076,  "ļoti",         LAMBDA,     3, -1, -1,      NULL,
  077,  "velti",        LAMBDA,     4, -1, -1,      NULL,
  000,  NULL,           NULL,       0,  0,  0,      NULL,
};

static RuleList step0l_rules[] =
{
  001,  "pēkšņi",       LAMBDA,     5, -1, -1,      NULL,
  002,  "respektīvi",   LAMBDA,     9, -1, -1,      NULL,
  003,  "līdzi",        LAMBDA,     4, -1, -1,      NULL,
  004,  "pretī",        LAMBDA,     4, -1, -1,      NULL,
  005,  "labāk",        LAMBDA,     4, -1, -1,      NULL,
  006,  "pēcāk",        LAMBDA,     4, -1, -1,      NULL,
  007,  "citādāk",      LAMBDA,     6, -1, -1,      NULL,
  010,  "savādāk",      LAMBDA,     6, -1, -1,      NULL,
  011,  "turpmāk",      LAMBDA,     6, -1, -1,      NULL,
  012,  "pārāk",        LAMBDA,     4, -1, -1,      NULL,
  013,  "agrāk",        LAMBDA,     4, -1, -1,      NULL,
  014,  "vairāk",       LAMBDA,     5, -1, -1,      NULL,
  015,  "visvairāk",    LAMBDA,     8, -1, -1,      NULL,
  016,  "mazāk",        LAMBDA,     4, -1, -1,      NULL,
  017,  "drīzāk",       LAMBDA,     5, -1, -1,      NULL,
  020,  "visbiežāk",    LAMBDA,     8, -1, -1,      NULL,
  021,  "cik",          LAMBDA,     2, -1, -1,      NULL,
  022,  "necik",        LAMBDA,     4, -1, -1,      NULL,
  023,  "šitik",        LAMBDA,     4, -1, -1,      NULL,
  024,  "atkal",        LAMBDA,     4, -1, -1,      NULL,
  025,  "tūdaļ",        LAMBDA,     4, -1, -1,      NULL,
  026,  "pakaļ",        LAMBDA,     4, -1, -1,      NULL,
  027,  "iepakaļ",      LAMBDA,     6, -1, -1,      NULL,
  030,  "nopakaļ",      LAMBDA,     6, -1, -1,      NULL,
  031,  "visnotaļ",     LAMBDA,     7, -1, -1,      NULL,
  032,  "atpakaļ",      LAMBDA,     6, -1, -1,      NULL,
  033,  "palaikam",     LAMBDA,     7, -1, -1,      NULL,
  034,  "aplam",        LAMBDA,     4, -1, -1,      NULL,
  035,  "piemēram",     LAMBDA,     7, -1, -1,      NULL,
  036,  "apmēram",      LAMBDA,     6, -1, -1,      NULL,
  037,  "nepagalam",    LAMBDA,     8, -1, -1,      NULL,
  040,  "pavisam",      LAMBDA,     6, -1, -1,      NULL,
  041,  "nepavisam",    LAMBDA,     8, -1, -1,      NULL,
  042,  "paretam",      LAMBDA,     6, -1, -1,      NULL,
  043,  "šimbrīžam",    LAMBDA,     8, -1, -1,      NULL,
  044,  "joprojām",     LAMBDA,     7, -1, -1,      NULL,
  045,  "aumaļām",      LAMBDA,     6, -1, -1,      NULL,
  046,  "lēnām",        LAMBDA,     4, -1, -1,      NULL,
  047,  "pamazām",      LAMBDA,     6, -1, -1,      NULL,
  050,  "gaužām",       LAMBDA,     5, -1, -1,      NULL,
  051,  "aizgūtnēm",    LAMBDA,     8, -1, -1,      NULL,
  052,  "pārpārēm",     LAMBDA,     7, -1, -1,      NULL,
  053,  "caurcaurēm",   LAMBDA,     9, -1, -1,      NULL,
  054,  "pamazītēm",    LAMBDA,     8, -1, -1,      NULL,
  055,  "pretim",       LAMBDA,     5, -1, -1,      NULL,
  056,  "iepretim",     LAMBDA,     7, -1, -1,      NULL,
```

```
  057,  "prom",          LAMBDA,    3, -1, -1,    NULL,
  060,  "patlaban",      LAMBDA,    7, -1, -1,    NULL,
  061,  "diezgan",       LAMBDA,    6, -1, -1,    NULL,
  062,  "šeitan",        LAMBDA,    5, -1, -1,    NULL,
  063,  "secen",         LAMBDA,    4, -1, -1,    NULL,
  064,  "šobaltdien",    LAMBDA,    9, -1, -1,    NULL,
  065,  "kādudien",      LAMBDA,    7, -1, -1,    NULL,
  066,  "citudien",      LAMBDA,    7, -1, -1,    NULL,
  067,  "daždien",       LAMBDA,    6, -1, -1,    NULL,
  070,  "mūždien",       LAMBDA,    6, -1, -1,    NULL,
  071,  "arvien",        LAMBDA,    5, -1, -1,    NULL,
  072,  "aizvien",       LAMBDA,    6, -1, -1,    NULL,
  073,  "varen",         LAMBDA,    4, -1, -1,    NULL,
  074,  "sen",           LAMBDA,    2, -1, -1,    NULL,
  075,  "pasen",         LAMBDA,    4, -1, -1,    NULL,
  076,  "nesen",         LAMBDA,    4, -1, -1,    NULL,
  077,  "bāztin",        LAMBDA,    5, -1, -1,    NULL,
  000,  NULL,            NULL,      0,  0,  0,    NULL,
 };

static RuleList step0m_rules[] =
 {
  001,  "druscin̦",       LAMBDA,    6, -1, -1,    NULL,
  002,  "mazdruscin̦",    LAMBDA,    9, -1, -1,    NULL,
  003,  "tūlin̦",         LAMBDA,    4, -1, -1,    NULL,
  004,  "mazlietin̦",     LAMBDA,    8, -1, -1,    NULL,
  005,  "neparko",       LAMBDA,    6, -1, -1,    NULL,
  006,  "vienkop",       LAMBDA,    6, -1, -1,    NULL,
  007,  "kurp",          LAMBDA,    3, -1, -1,    NULL,
  010,  "šurp",          LAMBDA,    3, -1, -1,    NULL,
  011,  "turp",          LAMBDA,    3, -1, -1,    NULL,
  012,  "vispār",        LAMBDA,    5, -1, -1,    NULL,
  013,  "viscaur",       LAMBDA,    6, -1, -1,    NULL,
  014,  "jebkur",        LAMBDA,    5, -1, -1,    NULL,
  015,  "nekur",         LAMBDA,    4, -1, -1,    NULL,
  016,  "visur",         LAMBDA,    4, -1, -1,    NULL,
  017,  "šur",           LAMBDA,    2, -1, -1,    NULL,
  020,  "tur",           LAMBDA,    2, -1, -1,    NULL,
  021,  "citur",         LAMBDA,    4, -1, -1,    NULL,
  022,  "vietumis",      LAMBDA,    7, -1, -1,    NULL,
  023,  "retumis",       LAMBDA,    6, -1, -1,    NULL,
  024,  "reizumis",      LAMBDA,    7, -1, -1,    NULL,
  025,  "sensenis",      LAMBDA,    7, -1, -1,    NULL,
  026,  "vairs",         LAMBDA,    4, -1, -1,    NULL,
  027,  "papildus",      LAMBDA,    7, -1, -1,    NULL,
  030,  "pārmijus",      LAMBDA,    7, -1, -1,    NULL,
  031,  "blakus",        LAMBDA,    5, -1, -1,    NULL,
  032,  "ieblakus",      LAMBDA,    7, -1, -1,    NULL,
  033,  "līdztekus",     LAMBDA,    8, -1, -1,    NULL,
  034,  "aplinkus",      LAMBDA,    7, -1, -1,    NULL,
  035,  "izklaidus",     LAMBDA,    8, -1, -1,    NULL,
  036,  "vienlaidus",    LAMBDA,    9, -1, -1,    NULL,
  037,  "neviļus",       LAMBDA,    6, -1, -1,    NULL,
  040,  "abpus",         LAMBDA,    4, -1, -1,    NULL,
  041,  "vienpus",       LAMBDA,    5, -1, -1,    NULL,
  042,  "katrpus",       LAMBDA,    6, -1, -1,    NULL,
  043,  "otrpus",        LAMBDA,    5, -1, -1,    NULL,
  044,  "virspus",       LAMBDA,    6, -1, -1,    NULL,
```

```
  045,  "papriekš",          LAMBDA,     7, -1, -1,      NULL,
  046,  "iepriekš",          LAMBDA,     7, -1, -1,      NULL,
  047,  "klāt",              LAMBDA,     3, -1, -1,      NULL,
  050,  "labprāt",           LAMBDA,     6, -1, -1,      NULL,
  051,  "nelabprāt",         LAMBDA,     8, -1, -1,      NULL,
  052,  "manuprāt",          LAMBDA,     7, -1, -1,      NULL,
  053,  "mūsuprāt",          LAMBDA,     7, -1, -1,      NULL,
  054,  "tīšuprāt",          LAMBDA,     7, -1, -1,      NULL,
  055,  "tavuprāt",          LAMBDA,     7, -1, -1,      NULL,
  056,  "ciet",              LAMBDA,     3, -1, -1,      NULL,
  057,  "mazliet",           LAMBDA,     5, -1, -1,      NULL,
  060,  "vienviet",          LAMBDA,     7, -1, -1,      NULL,
  061,  "vienuviet",         LAMBDA,     8, -1, -1,      NULL,
  062,  "dažviet",           LAMBDA,     6, -1, -1,      NULL,
  063,  "beidzot",           LAMBDA,     6, -1, -1,      NULL,
  064,  "visbeidzot",        LAMBDA,     9, -1, -1,      NULL,
  065,  "vairākkārt",        LAMBDA,     9, -1, -1,      NULL,
  066,  "pirmkārt",          LAMBDA,     7, -1, -1,      NULL,
  067,  "vienkārt",          LAMBDA,     7, -1, -1,      NULL,
  070,  "galvenokārt",       LAMBDA,    10, -1, -1,      NULL,
  071,  "apkārt",            LAMBDA,     5, -1, -1,      NULL,
  072,  "visapkārt",         LAMBDA,     8, -1, -1,      NULL,
  073,  "citkārt",           LAMBDA,     6, -1, -1,      NULL,
  074,  "daudzkārt",         LAMBDA,     8, -1, -1,      NULL,
  075,  "dažkārt",           LAMBDA,     6, -1, -1,      NULL,
  076,  "nost",              LAMBDA,     3, -1, -1,      NULL,
  077,  "pārlieku",          LAMBDA,     7, -1, -1,      NULL,
  000,  NULL,                NULL,       0,  0,  0,      NULL,
};

static RuleList step0n_rules[] =
        {
  001,  "šeit",              LAMBDA,     3, -1, -1,      NULL,
  002,  "tūlīt",             LAMBDA,     4, -1, -1,      NULL,
  003,  "pirmīt",            LAMBDA,     5, -1, -1,      NULL,
  004,  "maķenīt",           LAMBDA,     6, -1, -1,      NULL,
  005,  "atstatu",           LAMBDA,     6, -1, -1,      NULL,
  006,  "maz",               LAMBDA,     2, -1, -1,      NULL,
  007,  "pamaz",             LAMBDA,     4, -1, -1,      NULL,
  010,  "nemaz",             LAMBDA,     4, -1, -1,      NULL,
  011,  "vismaz",            LAMBDA,     5, -1, -1,      NULL,
  012,  "daudzmaz",          LAMBDA,     7, -1, -1,      NULL,
  013,  "bezmaz",            LAMBDA,     5, -1, -1,      NULL,
  014,  "vienlīdz",          LAMBDA,     7, -1, -1,      NULL,
  015,  "puslīdz",           LAMBDA,     6, -1, -1,      NULL,
  016,  "daudz",             LAMBDA,     4, -1, -1,      NULL,
  017,  "nedaudz",           LAMBDA,     6, -1, -1,      NULL,
  020,  "reiz",              LAMBDA,     3, -1, -1,      NULL,
  021,  "ikreiz",            LAMBDA,     5, -1, -1,      NULL,
  022,  "kādreiz",           LAMBDA,     6, -1, -1,      NULL,
  023,  "vairākreiz",        LAMBDA,     9, -1, -1,      NULL,
  024,  "cikreiz",           LAMBDA,     6, -1, -1,      NULL,
  025,  "tikreiz",           LAMBDA,     6, -1, -1,      NULL,
  026,  "vēlreiz",           LAMBDA,     6, -1, -1,      NULL,
  027,  "nākamreiz",         LAMBDA,     8, -1, -1,      NULL,
  030,  "viņreiz",           LAMBDA,     6, -1, -1,      NULL,
  031,  "šoreiz",            LAMBDA,     5, -1, -1,      NULL,
  032,  "toreiz",            LAMBDA,     5, -1, -1,      NULL,
```

```
    033,  "pašreiz",          LAMBDA,     6, -1, -1,      NULL,
    034,  "nākošreiz",        LAMBDA,     8, -1, -1,      NULL,
    035,  "citreiz",          LAMBDA,     6, -1, -1,      NULL,
    036,  "citureiz",         LAMBDA,     7, -1, -1,      NULL,
    037,  "daudzreiz",        LAMBDA,     8, -1, -1,      NULL,
    040,  "uzreiz",           LAMBDA,     5, -1, -1,      NULL,
    041,  "dažreiz",          LAMBDA,     5, -1, -1,      NULL,
    042,  "drīz",             LAMBDA,     3, -1, -1,      NULL,
    043,  "gandrīz",          LAMBDA,     6, -1, -1,      NULL,
    044,  "allaž",            LAMBDA,     4, -1, -1,      NULL,
    000,  NULL,               NULL,       0,  0,  0,      NULL,
    };

static RuleList step1a_rules[] =
    {
     100,  "ies",             LAMBDA,     2, -1,  0,      NULL,
     101,  "iem",             LAMBDA,     2, -1, -1,      NULL,
     102,  "ām",              LAMBDA,     1, -1, -1,      NULL,
     103,  "am",              LAMBDA,     1, -1, -1,      NULL,
     104,  "ēm",              LAMBDA,     1, -1, -1,      NULL,
     000,  NULL,              NULL,       0,  0,  0,      NULL,
    };

static RuleList step1a1_rules[] =
    {
     100,  "em",              LAMBDA,     1, -1, -1,      NULL,
     101,  "īm",              LAMBDA,     1, -1, -1,      NULL,
     102,  "im",              LAMBDA,     1, -1, -1,      NULL,
     103,  "um",              LAMBDA,     1, -1, -1,      NULL,
     104,  "us",              LAMBDA,     1, -1, -1,      NULL,
     000,  NULL,              NULL,       0,  0,  0,      NULL,
    };

static RuleList step1a2_rules[] =
    {
     100,  "as",              LAMBDA,     1, -1, -1,      NULL,
     101,  "es",              LAMBDA,     1, -1, -1,      NULL,
     102,  "u",               LAMBDA,     0, -1, -1,      NULL,
     103,  "os",              LAMBDA,     1, -1, -1,      NULL,
     104,  "ai",              LAMBDA,     1, -1, -1,      NULL,
     000,  NULL,              NULL,       0,  0,  0,      NULL,
    };

static RuleList step1a3_rules[] =
    {
     100,  "t",               LAMBDA,     0, -1, -0,      NULL,
     101,  "u",               LAMBDA,     0, -1, -0,      NULL,
     102,  "ei",              LAMBDA,     1, -1, -1,      NULL,
     103,  "ij",              LAMBDA,     1, -1, -1,      NULL,
     104,  "īs",              LAMBDA,     1, -1, -1,      NULL,
     000,  NULL,              NULL,       0,  0,  0,      NULL,
    };

static RuleList step1a4_rules[] =
    {
     100,  "ēs",              LAMBDA,     1, -1, -1,      NULL,
     101,  "is",              LAMBDA,     1, -1, -1,      NULL,
     102,  "ais",             LAMBDA,     2, -1, -1,      NULL,
```

```
    103,  "ie",          LAMBDA,    1, -1, -1,    NULL,
    104,  "s",           LAMBDA,    0, -1,  0,    NULL,
    000,  NULL,          NULL,      0,  0,  0,    NULL,
  };

static RuleList step1a5_rules[] =
  {
    100,  "š",           LAMBDA,    0, -1, -1,    NULL,
    101,  "a",           LAMBDA,    0, -1,  0,    NULL,
    102,  "i",           LAMBDA,    0, -1, -1,    NULL,
    103,  "e",           LAMBDA,    0, -1, -1,    NULL,
    104   "ā",           LAMBDA,    0, -1,  0,    NULL,
    000,  NULL,          NULL,      0,  0,  0,    NULL,
  };

static RuleList step1a6_rules[] =
  {
    100,  "ē",           LAMBDA,    0, -1,  0,    NULL,
    101,  "ī",           LAMBDA,    0, -1,  0,    NULL,
    102,  "ū",           LAMBDA,    0, -1,  0,    NULL,
    103,  "o",           LAMBDA,    0, -1,  0,    NULL,
    000,  NULL,          NULL,      0,  0,  0,    NULL,
  };

static RuleList step1b1_rules[] =
  {
    108,  "pj",          "p",       1,  0, -1,    NULL,
    109,  "bj",          "b",       1,  0, -1,    NULL,
    110,  "mj",          "m",       1,  0, -1,    NULL,
    111,  "vj",          "v",       1,  0, -1,    NULL,
    112,  "šļ",          "sl",      1,  0, -1,    NULL,
    113,  "žņ",          "zn",      1,  0, -1,    NULL,
    114,  "dž",          "dz",      1,  0,  1,    NULL,
    115,  "šņ",          "sn",      1,  0,  0,    NULL,
    116,  "žļ",          "zl",      1,  0, -1,    NULL,
    117,  "ļņ",          "ln",      1,  0, -1,    NULL,
    000,  NULL,          NULL,      0,  0,  0,    NULL,
  };

static RuleList step2_rules[] =
  {
    203,  "acionāl",     "acion",   6,  3,  0,    NULL,
    204,  "ācij",        "āc",      3,  1,  0,    NULL,
    205,  "ārij",        "ār",      3,  1,  0,    NULL,
    206,  "iecīb",       "iec",     4,  2,  0,    NULL,
    207,  "ainīb",       "ain",     4,  2,  0,    NULL,
    208,  "ādīb",        "ād",      3,  1,  0,    NULL,
    209,  "ātīb",        "āt",      3,  1,  0,    NULL,
    210,  "dzīb",        "dz",      3,  1,  0,    NULL,
    211,  "āfij",        "āf",      3,  1,  0,    NULL,
    212,  "omij",        "om",      3,  1,  0,    NULL,
    213,  "oģij",        "oģ",      3,  1,  0,    NULL,
    214,  "orij",        "or",      3,  1,  0,    NULL,
    000,  NULL,          NULL,      0,  0,  0,    NULL,
  };
```

```
static RuleList step3_rules[] =
        {
          301,  “iek”,          LAMBDA,     2, -1,  1,      NULL,
          302,   “iec”,         LAMBDA,     2, -1,  1,      NULL,
          303,  "niek",         LAMBDA,     3, -1,  0,      NULL,
          304,  "niec",         LAMBDA,     3, -1,  0,      NULL,
          305,  "nīc",          LAMBDA,     2, -1,  1,      NULL,
          306,  "ain",          LAMBDA,     2, -1,  0,      NULL,
          307,  "ant",          LAMBDA,     2, -1,  2,      NULL,
          308,  "ier",          LAMBDA,     2, -1,  1,      NULL,
          309,  "iet",          LAMBDA,     2, -1,  1,      NULL,
          310,  "inā",          LAMBDA,     2, -1,  1,      NULL,
          311,  "ing",          LAMBDA,     2, -1,  1,      NULL,
          312,  "ism",          LAMBDA,     2, -1,  1,      NULL,
          313,  "isk",          LAMBDA,     2, -1,  0,      NULL,
          314,  "ist",          LAMBDA,     2, -1,  1,      NULL,
          315,  "šan",          LAMBDA,     2, -1,  0,      NULL,
          316,  "iem",          LAMBDA,     2, -1,  1,      NULL,
          317,  "ām",           LAMBDA,     1, -1,  1,      NULL,
          318,  "am",           LAMBDA,     1, -1,  2,      NULL,
          319,  "ēm",           LAMBDA,     1, -1,  1,      NULL,
          320,  "em",           LAMBDA,     1, -1,  2,      NULL,
          321,  "īm",           LAMBDA,     1, -1,  2,      NULL,
          322,  "im",           LAMBDA,     1, -1,  2,      NULL,
          323,  "um",           LAMBDA,     1, -1,  2,      NULL,
          000,  NULL,           NULL,       0,  0,  0,      NULL,
        };

static RuleList step4_rules[] =
{
  401,  "āb",           LAMBDA,     1, -1,  2,      NULL,
  402,  "ād",           LAMBDA,     1, -1,  1,      NULL,
  403,  "āj",           LAMBDA,     1, -1,  1,      NULL,
  404,  "āl",           LAMBDA,     1, -1,  3,      NULL,
  405,  "ān",           LAMBDA,     1, -1,  1,      NULL,
  406,  "ār",           LAMBDA,     1, -1,  1,      NULL,
  407,  "āt",           LAMBDA,     1, -1,  1,      NULL,
  408,  "āz",           LAMBDA,     1, -1,  1,      NULL,
  409,  "āž",           LAMBDA,     1, -1,  1,      NULL,
  410,  "al",           LAMBDA,     1, -1,  2,      NULL,
  411,  "av",           LAMBDA,     1, -1,  1,      NULL,
  412,  "ēj",           LAMBDA,     1, -1,  2,      NULL,
  413,  "ēk",           LAMBDA,     1, -1,  1,      NULL,
  414,  "ēt",           LAMBDA,     1, -1,  1,      NULL,
  415,  "ēz",           LAMBDA,     1, -1,  1,      NULL,
  416,  "ej",           LAMBDA,     1, -1,  1,      NULL,
  417,  "el",           LAMBDA,     1, -1,  3,      NULL,
  418,  "er",           LAMBDA,     1, -1,  2,      NULL,
  419,  "īb",           LAMBDA,     1, -1,  1,      NULL,
  420,  "īc",           LAMBDA,     1, -1,  1,      NULL,
  421,  "īd",           LAMBDA,     1, -1,  3,      NULL,
  422,  "īg",           LAMBDA,     1, -1,  1,      NULL,
  423,  "īj",           LAMBDA,     1, -1,  1,      NULL,
  424,  "īt",           LAMBDA,     1, -1,  2,      NULL,
  425,  "īv",           LAMBDA,     1, -1,  2,      NULL,
  426,  "īz",           LAMBDA,     1, -1,  1,      NULL,
  427,  "ij",           LAMBDA,     1, -1,  1,      NULL,
  428,  "il",           LAMBDA,     1, -1,  2,      NULL,
```

```
        429,  "ik",           LAMBDA,     1, -1,  2,      NULL,
        430,  "iņ",           LAMBDA,     1, -1,  1,      NULL,
        431,  "ol",           LAMBDA,     1, -1,  2,      NULL,
        432,  "oņ",           LAMBDA,     1, -1,  0,      NULL,
        433,  "on",           LAMBDA,     1, -1,  1,      NULL,
        434,  "or",           LAMBDA,     1, -1,  1,      NULL,
        435,  "ot",           LAMBDA,     1, -1,  1,      NULL,
        436,  "ul",           LAMBDA,     1, -1,  2,      NULL,
        437,  "īn",           LAMBDA,     1, -1,  0,      NULL,
        438,  “aj”,           LAMBDA,     1, -1,  0,      NULL,
        000,  NULL,           NULL,       0,  0,  0,      NULL,
        };

static RuleList step6_rules[] =
        {
         601,  "šun",                 "sun",          0,  0,  0,      NULL,
         000,  NULL,                  NULL,           0,  0,  0,      NULL,
        };

static char *iflatv[]={ "ĀāČčĒēĢģĪīĶķĻļŅņŠšŪūŽž" } ;
static char *Slatv[]={ "āčēģīķļņšūž" } ;
static char *Blatv[]={ "ĀČĒĢĪĶĻŅŠŪŽ" } ;
static char *Vlatv[]={ "āīēū" } ;
/*****************************************************************/
/*********************  Private Function Declarations  ***********************/
static int IsLatVowel( word )
  char *word;   /* in: word */
  {
   char *ist;
  if (IsVowel(*word)) return( TRUE );
  else
        {
          ist = strchr(*Vlatv,*word);
          if ( NULL != ist ) return( TRUE ); else return( FALSE );
        };
  } /*IsLatVowel*/

int islatv(ch)
int ch;       /* current character during input scan */
{
  char *ist;
  ist = strchr(*iflatv,ch);
  if ( NULL != ist ) return( TRUE ); else return( FALSE );
} /* islatv */

int SmallLatv(ch)
int ch;       /* current character during input scan */
{
  union tet
  {
  char cha;
  int chi;
  } chch;
  char *ist;
  chch.cha=ch;
  ist = strchr(*Blatv,ch);
  if ( NULL != ist )
  {
```

```
  ist += *Slatv - *Blatv;
  chch.cha = *ist;
  return( chch.chi );
  };
  return( chch.chi );
} /* SmallLatv */
/*FN*************************************************************
```

WordSize(word)

Returns: int -- a count of word size in adjusted syllables

Purpose: Count syllables in a special way: count the number vowel-consonant pairs in a word, disregarding initial consonants and final vowels. For example, the WordSize of "sols" (a desk) is 1, of “upe” (a river) is 1, of "klase" (a class) is 2, of "pasaule" (the world) is 3.

Plan: Run a DFA to compute the word size

Notes: The easiest and fastest way to compute this measure is with a finite state machine. The initial state 0 checks the first letter. If it is a vowel, then the machine changes to state 1, which is the "last letter was a vowel" state. If the first letter is a consonant, then it changes to state 2, the "last letter was a consonant state". The result counter is incremented on the transition from state 1 to state 2, since this transition only occurs after a vowel-consonant pair, which is what we are counting.

```
**/

static int
WordSize( word )
  char *word;  /* in: word having its WordSize taken */
  {
  register int result;  /* WordSize of the word */
  register int state;   /* current state in machine */

  result = 0;
  state = 0;

                  /* Run a DFA to compute the word size */
  while ( EOS != *word )
    {
    switch ( state )
          {
          case 0: state = (IsLatVowel(word)) ? 1 : 2;
                  break;
          case 1: state = (IsLatVowel(word)) ? 1 : 2;
                  if ( 2 == state ) result++;
                  break;
          case 2: state = (IsLatVowel(word) || ('y' == *word)) ? 1 : 2;
                  break;
          }
  word++;
  }

return( result );
```

```
  } /* WordSize */
/*FN*****************************************************************

    ContainsVowel( word )

  Returns:      int -- TRUE (1) if the word parameter contains a vowel,
                FALSE (0) otherwise.

  Purpose:      Some of the rewrite rules apply only to a root containing
                a vowel, where a vowel is one of "aāeēiīouū" or y with a
                consonant in front of it.

  Plan:         Under the definition of a vowel, a word contains
                a vowel if either its first letter is one of "aāeēiīouū", or
                any of its other letters are "aāeēiīouūy".  The plan is to
                test this condition.

  Notes:        None
**/

static int
ContainsVowel( word )
  char *word;   /* in: buffer with word checked */
  {

  if ( EOS == *word )
    return( FALSE );
  else
    return( IsLatVowel(word) || (NULL != strpbrk(word+1,"aeiouy")) );

  } /* ContainsVowel */

/*FN*****************************************************************

    EndsWithCVC( word )

  Returns:      int -- TRUE (1) if the current word ends with a
                consonant-vowel-consonant combination,
                FALSE (0) otherwise.

  Purpose:      Some of the rewrite rules apply only to a root with
                this characteristic.

  Plan:         Look at the last three characters.

  Notes:        None
**/

static int
EndsWithCVC( word )
  char *word;   /* in: buffer with the word checked */
  {
  int length;       /* for finding the last three characters */

  if ( (length = strlen(word)) < 2 )
    return( FALSE );
  else
```

```
    {
    end = word + length - 1;
    return(    (NULL == strchr("aeiou",*end--))      /* consonant */
            && (NULL != strchr("aeiou",  *end--))       /* vowel */
            && (NULL == strchr("aeiou",   *end  )) );   /* consonant */
    }

  } /* EndsWithCVC */


/*FN**************************************************************
    ReplaceEnd( word, rule )

  Returns:      int -- the id for the rule fired, 0 is none is fired

  Purpose:      Apply a set of rules to replace the suffix of a word

  Plan:         Loop through the rule set until a match meeting all conditions
                is found.  If a rule fires, return its id, otherwise return 0.
                Connditions on the length of the root are checked as part of this
                function's processing because this check is so often made.

  Notes:        This is the main routine driving the stemmer.  It goes through
                a set of suffix replacement rules looking for a match on the
                current suffix.  When it finds one, if the root of the word
                is long enough, and it meets whatever other conditions are
                required, then the suffix is replaced, and the function returns.
**/

static int
ReplaceEnd( word, rule )
  char *word;       /* in/out: buffer with the stemmed word */
  RuleList *rule;    /* in: data structure with replacement rules */
  {
  register char *ending;   /* set to start of possible stemmed suffix */
  char tmp_ch;             /* save replaced character when testing */

  while ( 0 != rule->id )
    {
    ending = end - rule->old_offset;
    if ( word <= ending )
          if ( 0 == strcmp(ending,rule->old_end) )
             {
             tmp_ch = *ending;
             *ending = EOS;
             if ( rule->min_root_size < WordSize(word) )
               if ( !rule->condition || (*rule->condition)(word) )
                    {
                    (void)strcat( word, rule->new_end );
                    end = ending + rule->new_offset;
                    break;
                    }
          *ending = tmp_ch;
          }
rule++;
}
      return( rule->id );
```

```
    } /* ReplaceEnd */

/*FN*************************************************************

     CompStopW( word, rule )

Returns:        int -- the id for the rule fired, 0 is none is fired

Purpose:        Apply a set of rules to remove the stopword

Plan:           Loop through the rule set until a match meeting all conditions
                is found.  If a rule fires, return its id, otherwise remove the
                stopword.

Notes:          None
**/

static int
CompStopW( word, rule )
  char *word;       /* in/out: buffer with the stemmed word */
  RuleList *rule;   /* in: data structure with replacement rules */
  {
  register char *ending;  /* set to start of possible stemmed suffix */
  char tmp_ch;            /* save replaced character when testing */

  while ( 0 != rule->id )
    {
        if ( 0 == strcmp(word,rule->old_end) )
          {
          *word = EOS;
          }
    rule++;
    }

  return( rule->id );

  } /* CompStopW */

/*FN*************************************************************

     ReplaceW( word, rule )

Returns:        int -- the id for the rule fired, 0 is none is fired

Purpose:        Apply a set of rules to replace the word

**/

static int
ReplaceW( word, rule )
  char *word;       /* in/out: buffer with the stemmed word */
  RuleList *rule;   /* in: data structure with replacement rules */
  {
  register char *ending;  /* set to start of possible stemmed suffix */
  char tmp_ch;            /* save replaced character when testing */

  while ( 0 != rule->id )
    {
```

```
        if ( 0 == strcmp(word,rule->old_end) )
           {
           *word = EOS;
           (void)strcat( word,rule->new_end );
           }
  rule++;
   }

return( rule->id );

} /* CompStopW */
```

```
/************************************************************************/
/*********************  Public Function Declarations  *********************/
```

```
/*FN*************************************************************
```

Stem(word)

Returns: int -- FALSE (0) if the word contains non-alphabetic characters and hence is not stemmed, TRUE (1) otherwise

Purpose: Stem a word

Plan: Part 1: Check to ensure the word is all alphabetic
Part 2: Run through the Porter algorithm
Part 3: Return an indication of successful stemming

Notes: This function implements the Porter stemming algorithm, with a few additions here and there. See:

Porter, M.F., "An Algorithm For Suffix Stripping," Program 14 (3), July 1980, pp. 130-137.

Porter's algorithm is an ad hoc set of rewrite rules with various conditions on rule firing. The terminology of "step 1a" and so on, is taken directly from Porter's article, which unfortunately gives almost no justification for the various steps. Thus this function more or less faithfully refects the opaque presentation in the article. Changes from the article amount to a few additions to the rewrite rules; these are marked in the RuleList data structures with comments.

```
**/

int
Stem( word )
char *word;  /* in/out: the word stemmed */
{
int rule;    /* which rule is fired in replacing an end */

          /* Part 1: Check to ensure the word is all alphabetic */
for ( end = word; *end != EOS; end++ )
  if ( !islatv(*end) )
  {
        if ( !isalpha(*end) ) return( FALSE );
        else *end = tolower( *end );
  }
```

```
  else *end = SmallLatv( *end );
end--;

                /*  Part 2: Run through the Porter algorithm */
(void)CompStopW( word, step0a_rules );
(void)CompStopW( word, step0b_rules );
(void)CompStopW( word, step0c_rules );
(void)CompStopW( word, step0d_rules );
(void)CompStopW( word, step0e_rules );
(void)CompStopW( word, step0f_rules );
(void)CompStopW( word, step0g_rules );
(void)CompStopW( word, step0h_rules );
(void)CompStopW( word, step0i_rules );
(void)CompStopW( word, step0j_rules );
(void)CompStopW( word, step0k_rules );
(void)CompStopW( word, step0l_rules );
(void)CompStopW( word, step0m_rules );
(void)CompStopW( word, step0n_rules );

(void)ReplaceEnd( word, step1a_rules );
(void)ReplaceEnd( word, step1a1_rules );
(void)ReplaceEnd( word, step1a2_rules );
(void)ReplaceEnd( word, step1a3_rules );
(void)ReplaceEnd( word, step1a4_rules );
(void)ReplaceEnd( word, step1a5_rules );

(void)ReplaceEnd( word, step1b1_rules );

(void)ReplaceEnd( word, step2_rules );

(void)ReplaceEnd( word, step3_rules );

(void)ReplaceEnd( word, step4_rules );

(void)ReplaceW( word, step6_rules );

        /* Part 3: Return an indication of successful stemming */
return( TRUE );

} /* Stem */
```

APPENDIX 2.4

Main program for Latvian stemmer (STEMMER.C)

```
/*********************************  stemmer.c  ************************

*  Program to demonstrate and test the Porter stemming function.  This
*  program takes a single filename on the command line and lists stemmed
*  terms on stdout.
*
**/

#include <c:\bc4\include\stdio.h>
#include <c:\bc4\include\ctype.h>

#include "stem.h"

/*****************************************************************/
/*********************  Private Defines and Data Structures  *****************/

#define EOS                     '\0'

/*****************************************************************/
/*************************  Private Function Definitions  ********************/

#ifdef __STDC__

static char * GetNextTerm( FILE *stream, int size, char *term );

#else

static char * GetNextTerm();

#endif

/*****************************************************************

    GetNextTerm( stream, size, term )

  Returns:      char * -- buffer with the next input term, NULL at EOF

  Purpose:      Grab the next token from an input stream

  Plan:         Part 1: Return NULL immediately if there is no input
                Part 2: Initialize the local variables
                Part 3: Main Loop: Put the next word into the term buffer
                Part 4: Return the output buffer

  Notes:        None.
**/

static char *
GetNextTerm( stream, size, term )
  FILE *stream;  /* in: source of input characters */
  int size;     /* in: bytes in the output buffer */
  char *term;   /* in/out: where the next term in placed */
```

```
{
char *ptr;  /* for scanning through the term buffer */
int ch;     /* current character during input scan */

        /* Part 1: Return NULL immediately if there is no input */
if ( EOF == (ch = getc(stream)) ) return( NULL );

        /* Part 2: Initialize the local variables */
*term = EOS;
ptr = term;

      /* Part 3: Main Loop: Put the next word into the term buffer */
do
  {
      /* scan past any leading non-alphabetic characters */
      fil = islatv(ch);
      while ( (EOF != ch ) && !isalpha(ch) ) || fil) )
      . .
      ch = getc( stream );
      fil = islatv(ch);
      };

    /* copy input to output while reading alphabetic characters */
      while ( (EOF != ch ) && isalpha(ch) || islatv(ch)) )
      . .

      if ( ptr == (term+size-1) ) ptr = term;
      *ptr++ = ch;
      ch = getc( stream );
          }

       /* terminate the output buffer */
     *ptr = EOS;
     }
  while ( (EOF != ch) && !*term );

                /* Part 4: Return the output buffer */
  return( term );

  } /* GetNextTerm */

/*************************************************************************/
/*************************************************************************

      main( argc, argv )

  Returns: int -- 0 on success, 1 on failure

    Purpose:    Program main function

    Plan:       Part 1: Open the input file
                Part 2: Process each word in the file
                Part 3: Close the input file and return

    Notes:      None
**/

int
```

```
main( argc, argv )
  int argc;     /* in: how many arguments */
  char *argv[]; /* in: text of the arguments */
  {
  char term[64];   /* for the next term from the input line */
  FILE *stream;    /* where to read characters from */

                    /* Part 1: Open the input file */
  if ( !(stream = fopen(argv[1],"r")) ) exit(1);

                /* Part 2: Process each word in the file */
while( GetNextTerm(stream,64,term) )
  if ( Stem(term) ) (void)printf( "%s\n", term );

            /* Part 3: Close the input file and return */
(void)fclose( stream );
return(0);

} /* main */
```

APPENDIX 3.1

List of Latvian endings

-ies	-um	-ām	-am	-ēm
-em	-īm	-im	-iem	-us
-as	-es	-ās	-os	-ai
-t	-u	-ei	-ij	-īs
-ēs	-is	-ais	-ie	-s
-š	-a	-i	-e	-ā
-ē	-ī	-ū	-o	

APPENDIX 3.2

List of consonant palatalisation

pj - p	bj - b	mj - m	vj - v	šļ - sl
žņ - zn	dž - dz	šņ - sn	žļ - zl	ļņ - ln

APPENDIX 3.3

List of Latvian suffixes

-acion	-āc	-ār	-iec	-ain
-ād	-āt	-dz	-āf	-om
-oģ	-or	-niek	-niec	-nīc
-iek	-ant	-ier	-iet	-inā
-ing	-ism	-isk	-ist	-šan
-iem	-ām	-am	-ēm	-em
-īm	-im	-um	-āb	-āj
-āl	-ān	-āz	-āž	-al
-av	-ēj	-ēk	-ēt	-ēz
-ej	-el	-er	-īb	-īc
-īd	-īg	-īj	-īt	-īv
-īz	-ij	-il	-ik	-iņ
-ol	-oņ	-on	-or	-ot
-ul	-īn			

APPENDIX 4.1

EVALUATION FORM FOR STEMMED LATVIAN WORDS

A. Name of the linguistic specialist _____

B. Position _____

C. Institution/organisation _____

D. Please rate (tick) the quality of stemmed nouns (5 - very good ... 1 - very poor):

1. ☐ 2. ☐ 3. ☐ 4. ☐ 5. ☐

Problems _____

_____

Suggestions and comments _____

_____

E. Please rate (tick) the accurancy of stemmed adjectives:

1. ☐ 2. ☐ 3 ☐ 4. ☐ 5. ☐

Problems _____

_____

Suggestions & comments _____

_____

F. Please rate (tick) the accuracy of stemmed verbs:

1. ☐ 2. ☐ 3. ☐ 4. ☐ 5. ☐

Problems _____

_____

Suggestions & comments _____

_____

G. Please rate (tick) the accuracy of stemmed adverbs:

1. ☐ 2. ☐ 3. ☐ 4. ☐ 5. ☐

Problems _____

_____

Suggestions & comments _____

_____

H. Suggestions for further improvements _____

_____

_____

_____

Thank you

APPENDIX 4.2

Examples of Latvian word stemming in standard forms

NOUNS

Unstemmed	Stemmed
evolūcija	evolūc
pēcrevolūcija	pēcrevolūc
rezolūcija	rezolūc
stadija	stad
traģēdija	traģēd
telestudija	telestud
reliģija	reliģ
revīzija	revīz
žalūzija	žalūz
korozija	koroz
boja	boj
pļauja	pļauj
salaka	salak
kabelis	kabel
trifelis	trifel
panelis	panel
lietusmētelis	lietusmēt
svētelis	svētel
bēglis	bēgl
slēpnis	slēpn
cietoksnis	cietoksn
zveltnis	zveltn
gultnis	gultn
sūtnis	sūtn
lasītava	lasīt
klausītava	klausīt
tītava	tīt
slidotava	slidot
tirgotava	tirgot
prāva	prāv
alternatīva	alternat

prerogatīva	prerogat
olīva	olīv
brigāde	brigād
magone	magon
grāfiene	grāf
deputāte	deput
tonalitāte	tonalit
berete	beret
pūce	pūc
tēze	tēz
ateljē	atelj
jaunstrāvnieks	jaunstrāv
pesimisms	pesim
reformisms	reform
organisms	organ
mehānisms	mehān
alpīnisms	alpīn
grebums	greb
fotoalbums	fotoalb
krūšdobums	krūšdob
iedegums	iedeg
aizliegums	aizlieg
sniegums	snieg
noziegums	nozieg
augstspriegums	augstsprieg
maigums	maig
pļāpīgums	pļāpīg
ierosinājums	ierosin
nocietinājumns	nocietin
apdrošinājums	apdrošin
dāvinājums	dāvin
integrālvienādojums	integrālvienādoj
tēlojums	tēloj
tīklojums	tīkloj
operators	operat
radiators	radiat
inspektors	inspekt
gravētājs	gravēt
organizētājs	organizēt

ADJECTIVES

Unstemmed	Stemmed
pozitīvs	pozit
reaktīvs	reakt
efektīvs	efekt
perspektīvs	perspekt
subjektīvs	subjekt
selektīvs	selekt
instinktīvs	instinkt
stīvs	stīv
abrazīvs	abraz
dzīvs	dzīv
precīzs	precīz
dižs	diž
klajš	klaj
apaļš	apaļ
pusapaļš	pusap
mierīgs	mier
zvērīgs	zvēr
atturīgs	attur
pirmklasīgs	pirmklas
bravūrīgs	bravūr
nebalsīgs	nebals
prātīgs	prāt
sātīgs	sāt
simtprocentīgs	simtprocent
pelēks	pelēk
daudzgadīgs	daudzgad
loģisks	loģisk
elektrotehnisks	elektrotehn
ekonomisks	ekonom
personisks	person
metrisks	metr
satīrisks	satīr
aizvēsturisks	aizvēstur
latvisks	latv
maksimāls	maksim

sentimentāls	sentiment
tāls	tāl
kvēls	kvēl
ceremoniāls	ceremoni
jauks	jauk
lojāls	lojāl
dokumentāls	dokument
monumentāls	monument
tīkams	tīkam
mēms	mēm
zilacains	zilac
lēns	lēn
grumbuļains	grumbuļ
caurumains	caurum
čemurains	čemur
trīskrāsains	trīskrās
vārpatains	vārpat
baltmatains	baltmat
moderns	modern
kluss	klus
ass	ass
mundrs	mundr
ātrs	ātr
populārs	popul
elementārs	element
fragmentārs	fragment
šaurs	šaur
elegants	eleg
sarkanbalts	sarkanbalt
erudīts	erud
sniegots	snieg
skujots	skuj
krunkains	krunk
komplicēts	komplic
apdāvināts	apdāvin
ciets	ciet
riskants	risk
jauns	jaun
rātns	rātn

VERBS

Unstemmed	Stemmed
apsveicināties	apsveicin
pazemināties	pazemin
cepināties	cepin
nocietināties	nocietin
mētāties	mētā
žāvāties	žāvā
apieties	apiet
sabojāties	sabojā
samērcēties	samērcē
laimēties	laimē
izārstēties	izārstē
balotēties	balotē
klauvēties	klauvē
zagties	zagt
izkliegties	izkliegt
blēdīties	blēdī
žuburoties	žuburot
vairoties	vairot
aizņemties	aizņemt
iepatikties	iepatikt
pumpuroties	pumpuro
plēsties	plēst
berzties	berzt
grauzties	grauzt
atgriezties	atgriezt
kompromitēt	kompromitē
boikotēt	boikotē
protestēt	protestē
startēt	startē
rīvēt	rīvē
dzirksteļot	dzirksteļo
izstarot	izstaro
stiebrot	stiebro
pilvarot	pilnvaro
sapņot	sapņo

knābāt	knābā
pasargāt	pasargā
pogāt	pogā
nejaudāt	nejaudā
bradāt	bradā
taujāt	taujā
drukāt	drukā
plūkāt	plūkā
pacilāt	pacilāt
māt	māt
lamāt	lamā
sasaldināt	sasaldin
apdarināt	apdarin
sašķidrināt	sašķidrin
izvingrināt	izvingrin
apmulsināt	apmulsin
burāt	burā
dāvāt	dāvā
grābt	grāb
iet	iet
riet	riet
ratificēt	ratificē
lādēt	lādē
lodēt	lodēt
spolēt	spolē
kontrolēt	kontrolē
sakarsēt	sakarsē
izolēt	izolē
smēķēt	smēķēt
reaģēt	reaģē
spēlēt	spēlē
apģērbt	apģērb
sūnot	sūno
atzarot	atzaro
stīpot	stīpo
uzskaņot	uzskaņo
vēsmot	vēsmo
burbuļot	burbuļo
simbolizēt	simbolizē

ADVERBS

Unstemmed	Stemmed
avansveidā	avansveid
rokrokā	rokrok
pusmastā	pusmast
personīgi	person
ziņkārīgi	ziņkār
saderīgi	sader
pamatīgi	pamat
draudzīgi	draudz
krusteniski	krusten
zviedriski	zviedr
tenteriski	tenter
stūriski	stūr
gareniski	garen
slepeni	slepen
šovasar	šovasar
kalnup	kalnup
rītvakar	rītvakar
pusrikšus	pusrik
pazagšus	pazag
braukšus	brauk
piecatā	piecat
portugāliski	portug
pastāvīgi	pastāv
izcili	izcil
pirmdien	pirmdien
pērnruden	pērnruden
vakar	vakar
mājup	mājup
peldus	peld
sēdus	sēd
puszviļus	puszviļ
dīkā	dīk
lejā	lej
kūleniski	kūlen
viszēlīgi	viszēl

APPENDIX 4.3

EVALUATION FORM FOR THE ANALYTICAL INFORMATION SYSTEM OF PERIODICALS

Code _____

Please write five (5) complete search statements based on this database:

(Example: On trade agreements between Latvia and Great Britain)

1. _____

2. _____

3. _____

4. _____

5. _____

Please truncate your five search statements excluding stopwords as appropriate:

1. _____

2. _____

3. _____

4. _____

5. _____

APPENDIX 4.4

A set of full, truncated and stemmed queries used in the main test

F = full; T1 = Truncated by the 1st user; T2 = Truncated by the 2nd user; S = Stemmed

1. **Par Latvijas eksportu (F)**
   Eksport* Latv* (T1)
   Latv* eksport* (T2)
   latv* eksport* (S)

2. **Par jūras kuģniecības darbību (F)**
   kuģniec* jūr* (T1)
   jūr* kuģn* darb* (T2)
   jūr* kuģniec* darbīb* (S)

3. **Par zvejniecību jūrā (F)**
   ziv* zvejn* (T1)
   zvejn* jūr (T2)
   zvejniec* jūr* (S)

4. **Par ledusskapju rūpniecību (F)**
   ledusskap* saldēt* (T1)
   ledusskap* rūpn* (T2)
   ledusskap* rūpniec* (S)

5. **Par Baltijas banku (F)**
   Baltij* bank* (T1)
   Balt* bank* (T2)
   balt* bank* (S)

6. **Par Latvijas bibliotēku materiālo apgādi (F)**
   latv* biblio* mater* apgād* (T1)
   latv* biblio* materiāl* apgād* (T2)
   latv* bibliotēk* materiāl* apgād* (S)

7. **Labdarības fondi Latvijā (F)**

labdar* fond* Latv* (T1)

labdarīb* fond* Latv* (T2)

labdar* fond* latvij* (S)

8. **Likumi, kas regulē kultūras darbu Latvijā (F)**

Likum* regul* kultū* darb* Latv* (T1)

Likum* kult* darb* Latv* (T2)

likum* regul* kultūr* darb* latvij* (S)

9. **Informācijas sistēma par Latvijas kultūru (F)**

Inform* sistēm* Latv* kultū* (T1)

Informāc* sist* par Latv* kult* (T2)

informāc* sistēm* latv* kultūr* (S)

10. **Kultūras struktūras Saeimā un Ministru Kabinetā un to sastāvs (F)**

Kultū* struktūr* Saeim* Ministr* Kabin* sastāv* (T1)

Kult* strukt* Saeim* Ministr* kab* sastāv* (T2)

kultūr* struktūr* saeim* ministr* kabinet* sastāv* (S)

11. **Valsts ilgtermiņa kredīti (F)**

Valsts ilgterm* kred* (T1)

Ilgtermiņa kredīt* valst* (T2)

valst* ilgterm* kredīt* (S)

12. **Likumdošana par pilsētu pašvaldībām (F)**

Likumd* pils* pašvald* (T1)

Likum* pašvaldīb* pilsēt* (T2)

likumd* pilsēt* pašvald* (S)

13. **Klīrings un citas norēķinu sistēmas (F)**

Klīring* norēķ* sist* (T1)

Klīring* norēķ* sistēm* (T2)

klīring* norēķin* sistēm* (S)

14. **Inflācijas koeficients un pārtikas grozs ASV (F)**

Inflāc* koefic* pārtika* groz* ASV (T1)

inflācij* koeficient* pārtika* groz* ASV (T2)

inflāc* koefic* pārtik* groz* asv (S)

15. **Rietumeiropas centrālo banku struktūra (F)**
Rietumeirop* centr* bank* strukt* (T1)
Rietumeirop* centrāl* bank* struktūra (T2)
rietumeirop* centrāl* bank struktūr* (S)

16. **Jaunkareivju (jauniesaukto) pazemošana Latvijas armijā (F)**
armija• ārpusreglament* jauniesaukt* Latvi* (T1)
Latv* armij* jaunkareiv* (T2)
jaunkareiv* jauniesaukt* pazem* latv* armij* (S)

17. **Ārvalstu investīcijas Latvijā (F)**
Latvija* investīcija* (T1)
Invest* Latv* (T2)
ārvalst* invest* latvij* (S)

18. **Krievijas armijas virsnieku uzturēšanās atļaujas Latvijā (F)**
Krievija* (kriev*) armija* virsniek* Latvij* (T1)
Kriev* arm* virsn* uzturēš* Latv* (T2)
kriev* arm* virsniek* uzturē* atļauj* latvij* (S)

19. **Skandāls ar "Parex" bankas garantijām (F)**
Parex bank* garant* Dānij* (dāņu) (T1)
Parex banka (T2)
skandāl* parex bank* garant* (S)

20. **ASV Aizsardzības ministra V. Perija vizīte Latvijā (F)**
Viljam* Perijs Latvij* ASV vizīt* (apmeklē*) (T1)
ASV aizsardz* min* Perij* vizīt* Latv* (T2)
asv aizsardz* ministr* v perij* vizīt* latvij* (S)

21. **Par "Bankas Baltija" finansiālo krīzi (F)**
Banka* Baltija* finans* krīze (T1)
Banka* Baltija* finans* krīze (T2)
bank* baltij* finansiāl* krīz (S)

22. **Par Pilsonības likumu Latvijā (F)**
Pilson* lik* (T1)
Pilson* lik* Latv* (T2)
pilson* likum* latvij* (S)

23. **Par ārvalstu attieksmi pret militāro konfliktu Čečenijā (F)**
Ārvalstu attieksme milit* konf* Čeč* (T1)
ārvalstu attieksme* mil* konfl* Čečenijā (T2)
ārvalst* attieksm* milit* konflikt* čečen* (S)

24. **Latvijas Republikas 6. Saeimas kandidātu izvirzīšana (F)**
6. Saeim* kand* (T1)
Latv* Republ* 6. Saeimas kandid* izvirzīšana (T2)
latv* republik* saeim* kandid* izvirzī* (S)

25. **Par "Klubs-21" dalībnieku iesaistīšanos politiskajā darbībā (F)**
Klub* 21 dalībn* (T1)
Klubs-21 dalībn* iesaist* polit* darbībā (T2)
klub* dalībn* iesaistī* politiskaj* darbīb* (S)

26. **Lata kurss Latvijas un ārvalstu bankās (F)**
Lata kurss Latv* ārv* bankās (T1)
Lat* kurs* (T2)
lat* kurs* latv* ārvalst* bank* (S)

27. **Starptautisko organizāciju darbība Latvijā (F)**
Starpt* org* darb* Latv* (T1)
Starptautisk* organizācij* Latvij* (T2)
starptaut* organizāc* darbīb* latvij* (S)

28. **Pēdējās nedēļas kriminālhronika (F)**
krim* hron* pēdējā* ned* (T1)
kriminālhronik* (T2)
pēdēj* nedēļ* kriminālhron* (S)

29. **Latvijas Prezidenta vizīte Anglijā (F)**
Latv* prezidenta vizīte Anglijā (T1)
prezident* Anglij* Latvij* (T2)
latv* prezident* vizīt* anglij* (S)

30. **Rītdienas laika prognoze (F)**
laika prog* rīt* (T1)
laik* prognoz* (T2)
rītdien* laik* prognoz* (S)

31. **Par 6. Saeimas vēlēšanām Latvijā (F)**
6. Saeim* vēl* (T1)
Par 6. Saeimas vēl* Latv* (T2)
saeim* vēlē* latvij* (S)

32. **Ārpusreglamenta attiecības Latvijas armijā (F)**
armij* ārpusreglament* latvij* (T1)
ārpusregl* attiec* Latv* arm* (T2)
ārpusreglament* attiec* latv* armij* (S)

33. **LR lēmumi un likumi, kas vērsti uz LR iestāšanos ES (F)**
Eirop* Savienība (lēmum* likum*) (T1)
LR lēm* lik* vērsti uz LR iestāš* ES (T2)
lr lēmum* likum* vērst* lr iestā* (S)

34. **Latvijas banku sistēmas attīstība 1992.-1996. (F)**
bank* Latvij* (T1)
Latv* banku sist* attīst* 92.-96. (T2)
latv* bank* sistēm* attīst* (S)

35. **Valsts un privātie monopoluzņēmumi Latvijā (F)**
monopol* Latvij* valst* privāt* (T1)
valsts priv* monopoluzņ* Latv* (T2)
valst* privāt* monopoluzņēm* latvij* (S)

36. **Par Latvijas sadarbības līgumiem jūras lietās ar Eiropas valstīm(F)**
Latv* sadarb* līg* jūras lietās ar Eir* valstīm (T1)
Latvij* sadarb* līgum* jūr* liet* Eirop* valst* (T2)
latv* sadarb* līgum* jūr* liet* eirop* val* (S)

37. **Informācijas centru darbība biznesa struktūrās (F)**
Inf* c* darb* bizn* strukt* (T1)
Informāc* centr* biznes* struktūr (T2)
informāc* centr* darbīb* biznes* struktūr* (S)

38. **Latvijas un Igaunijas jūras robežas līgumu vēsture (F)**
Latv* un Ig* jūras rob* līg* vēst* (T1)
Latvij* Igaynij* jūra* robež* līgum* vēstur* (T2)
latv* igaun* jūr* robež* līgum* vēstur* (S)

39. **Latvijas un Islandes ekonomiskie un politiskie sakari (F)**
Latv* un Isl* ek* un pol* sak* (T1)
Latvij* Island* ekonom* polit* sakar* (T2)
latv* island* ekonom* polit* sakar* (S)

40. **Latvijas un Lielbritānijas tirdzniecības līgumi (F)**
Latv* un Lielbr* tirdzn* līg* (T1)
Latvij* Lielbrit* tirdzn* līgum* (T2)
latv* liebrit* tirdzniec* līgum* (S)

41. **Izstāde "Datortehnika '96" (F)**
Izstād* Datortehnik* '96 (T1)
Izstā* Datortehnik* (T2)
izstād* datortehn* (S)

42. **Latvijas un Igaunijas jūras robeža (F)**
Latvij* un Igaunij* jūr* robež* (T1)
Latv* Igaunij* jūr* robež* (T2)
latv* igaun* jūr* robež* (S)

43. **Zivju pārstrādes uzņēmumi Latvijā (F)**
Zivj* pārstrād* Latvij* (T1)
Ziv* pārstrād* uzņēm* Latv* (T2)
ziv* pārstrād* uzņēmum* latvij* (S)

44. **Latvijas karavīri Bosnijā (F)**
Latvij* kar* Bosnij* (T1)
Latv* karavīr* Bosni* (T2)
latv* karavīr* bosnij* (S)

45. **Lata un dolāra kursu attiecības (F)**
Lat* un dolār* kurs* (T1)
Lat* dolār* kurs* attiec* (T2)
lat* dolār* kurs* attiec* (S)

46. **Spēļu metodes izmantošana pirmsskolas vecuma bērnu apmācībā (F)**
Spēl* metod* izmant* pirmsskol* vecum* bērn* apmāc* (T1)
Mācīb* pirmsskol* spēl* (spēļ*) (T2)
spēļ* metod* izmant* pirmsskol* vecum* bērn* apmāc* (S)

47. **Angļu valodas pasniegšanas metodika (F)**
Angļu valod* pasnieg* metod* (T1)
metod* angļ* valod* (T2)
angļ* valod* pasnieg* metodik* (S)

48. **Latvijas un Krievijas krimināl kodeksu salīdzinošā analīze (F)**
Latv* Kriev* krimināl kodeks* salīdz* analī* (T1)
krimināl kodeks* Latvij* Krievij* analīz* (T2)
latv* kriev* krimināl kodek* salīdzinoš* anal* (S)

49. **Pusaudžu kriminālās uzvedības psiholoģija (F)**
Pusaud* krimināl* uzve* psiholoģi* (T1)
psiholo* pusaud* uzvedīb* krimināl* (T2)
pusaudž* krimināl* uzved* psiholoģ* (S)

50. **Brīvās tirdzniecības zonas un off-shore kompānijas (F)**
Brīv* tir* zon* off-shore kompāni* (T1)
Brīv* tirdzniec* zon* off-shore kompān* (T2)
brīv* tirdzniec* zon* off shor* kompān* (S)