Text categorization using lexical chains

Tue Haste Andersen
Supervisor: László Béla Kovács

Department of Computer Science,
Copenhagen University

February 28, 2000

Abstract

In this report I present a prototype system for use in dynamic text categorization research. The system implements lexical chaining, as described in recent literature. On top of this is built a simple extension to use for automatically identifying one or several categories to place a given text in. The initial tests presented in this report does not give any useful results, however, it give rise to new questions and possible directions for future research of lexical chaining and its uses in text categorization. Along with the implementation, previous research and the lexicographic database WordNet are discussed.

Keywords: Text categorization, dynamic, lexical chaining, WordNet, text retrieval.

Contents

Introduction

Categorization is of importance when large amount of information needs to be stored in some way, for later retrieval. Examples of uses are categorization of books and articles, web pages, email messages and message routing. Today it is of growing importance to find good ways to do this automatically, as the amount of information available electronically is growing fast.

Automatic categorization of text can be done in many ways, depending on the use of the categorized texts. Below is listed three approaches, grouped by what kind of information is available prior to categorization:

Pre-categorized texts. If pre-categorized material is available, it is possible to deduce simple, yet fast and effective rules, to use for categorizing new material of same type. Studies in rule induction have been done by Quinlan [Qui96] and Cohen [Coh95a] primary focusing on efficiency, in terms of speed.

Semantic networks. When a semantic network relating word to actions, concepts, etc., is available, material can be categorized in a number of ways, relying on the network.

User model. In a system based on a user model, it is possible to use the information collected about the users behavior, preferences etc., for categorizing data. This approach could be made far more dynamic than that of the rule induction.

In this report, I will focus on a browsable hypertext system, similar to those present on Internet portals like the "WWW virtual library"[1]. Here the presented links are selected and categorized by humans. Therefore it will be necessary to employ more people to find, select and categorize the information, as the Internet grows. If part of this process could be automated, enormous amounts of resources could be saved. It might be difficult to make a system that ensures high quality in the selected documents, but to categorize documents automatically seems more reasonable, because a complete knowledge of the documents is not required. Instead a rather coarse grained view of the topics present in the documents will do.

[1] `http://vlib.org/`

The categories on a typical Internet portal are stored in a hierarchy or network structure of topic labels. Today documents are manually placed in these categories, and the whole structure of category labels is also extended manually. The problem is therefore not only to assign a predefined label to a given document, but also to place a document into a hierarchy or network of topic labels, *and* to extend this structure whenever needed. For this purpose, the semantic network approach mentioned above, can provide the necessary background information. The fact that such background information bases are public available, makes it possible to rapidly develop systems that can be used for experimentation on real life data. Furthermore, the learning phase is not limited to symbol manipulation of known data, as is the case of the two other examples mentioned above (in the first case pre-categorized texts; in the user model, recorded user actions.)

In the following chapters I will proceed by introducing background information on the lexicographic database WordNet[2], along with an overview of research in text retrieval problems. This is followed by a description of the theory of lexical chaining, which is used to disambiguate meanings of words, as represented in WordNet. Finally a method for categorizing texts is presented, along the implementation of the prototype system[3] and experiments, showing initial results. The main goal is to examine if a good method for text categorization using lexical chaining can be found, and to test the usability of the lexical chaining technique in practice.

[2] WordNet can be downloaded from `http://www.cogsci.princeton.edu/~wn/`

[3] The system will soon be made available for download from `http://www.diku.dk/students/haste/textcat/`

Chapter 1

Background

The system described in this project heavily relies on a lexicographic database, called WordNet [Fel98]. Other databases are available which contains information about language and semantics, but WordNet is unique in that it covers a large part of the target language, English, and has well defined concepts and relations.

Another database that probably contains more information than WordNet, is Cyc available from Cycorp[1]. However, this database has been criticized for not being well-structured, and therefore difficult to use in text retrieval problems [KMF96]. In [KMF96] it is demonstrated that a number of problems exists when Cyc is used in the solving of classic text retrieval problems. These includes:

- Incomplete and non-uniform coverage of knowledge concepts. WordNet has also been criticized for incomplete coverage, but has a uniform structure, where Cyc lacks selectional constraints on the knowledge.
- Inefficient accessibility of knowledge, because the whole database has to be searched to find everything about a given concept.

There are other electronically available ontologies including Pangloss, Mikrokosmos and EDR [KMF96]. However, none of these have as wide coverage of a language as WordNet or Cyc.

Among other, these are the reasons why I have chosen to base this project on WordNet. In this chapter I will give an overview of what WordNet contains, and how it is structured. Secondly I will present an overview of the field of text retrieval in the context of text categorization.

[1] See `http://www.cycorp.com/`

Word	Synset id	Synset gloss
Orange	103880945	Any of a range of colors between red and yellow.
	105783752	Round yellow to orange fruit of any of several citrus trees.
	109007985	Any citrus tree bearing oranges.
	110758147	Any pigment producing the orange color.

Table 1.1: Example of the synsets representing the different senses of the word "Orange."

1.1 WordNet

In WordNet every word is represented by a set of synonyms, called synsets (see table 1.1.) Each synonym represents a meaning of the word. Each synset have a unique id, most of them have a gloss phrase assigned to them, which is some informal English text, describing the meaning of the synonym. But apart from this phrase, the synonyms are defined in terms of links to other synsets. This means that the information in WordNet is build around sets of synonyms, as they are the primary keys to represent the semantics of words.

Figure 1.1 shows a search on the word "orange" in WordNet. In the left window we see that the word both can be used as an adjective, and as a noun. Furthermore it shows different types of links to the word – E.g. following "... is a kind of orange (hyponyms)", will show the synsets which are specialization's of orange. The right window shows the four different senses found of the word, and displays the gloss phrase associated with each.

A number of different relations exist between synsets in WordNet. They are listed in table 1.2, along with their type. A semantic relation means that the relation holds between meanings of words (more specifically, synsets), and a lexical relation means that its between word forms, not meanings of words.

The most important relation of table 1.2 is synonymy. Of other important relations represented in WordNet is antonymy, which is a lexical relation describing relations between words of opposite meaning, e.g. *ascend* and *descend* are antonyms. Hyponymy and hypernymy is a semantic relation describing a hierarchy between word meanings. Here *tool* is a hypernym of *fork*, and *fork* is a hyponym of *tool*. Meronymy and holonymy can be described by the *has-a* or *part-of* relation: E.g. a *motor* is part of a *car*, therefore *motor* is a meronym of *car* and *car* is a hypernym of *motor*. In WordNet the actual representation of the *car-motor* relation, consists of several intermediate levels: *motor* is a hypernym of *engine*, which is a hypernym of *automobile engine*, which is a part meronym of *car*. Furthermore meronymy is divided into three types of relations in WordNet, namely member,

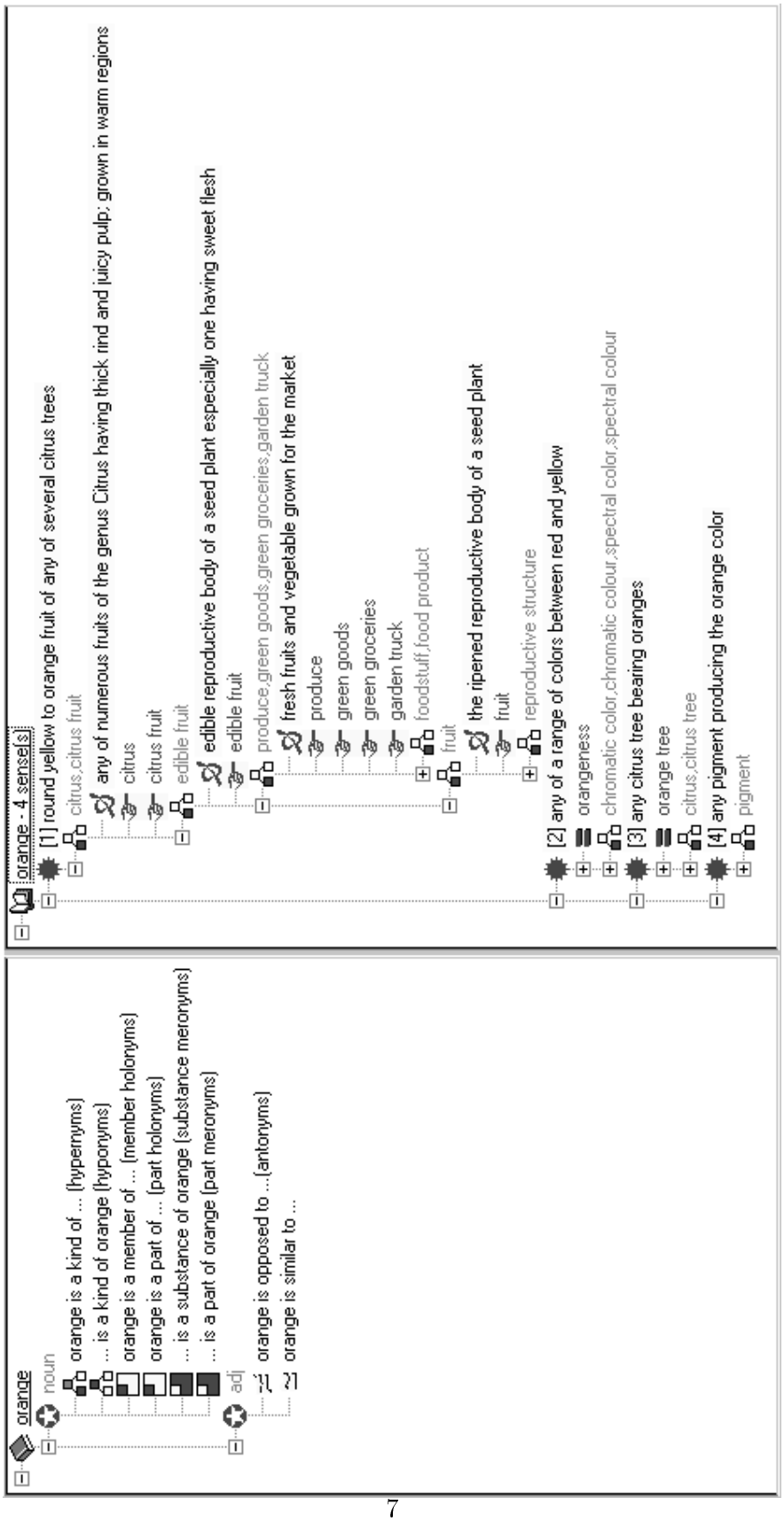

Figure 1.1: Screen shot from WordNet TreeWalk showing a search on "Orange." WordNet TreeWalk is available for download from `http://www.ac-toulouse.fr/wordnet/`

Name	Abbreviation	Type	Word forms
Attribute	at	Semantic	Adjectives-nouns
Synonymy	sim	Semantic	Adjectives
Antonymy	ant	Lexical	
Hyponymy/Hypernymy	hyp	Semantic	Nouns and verbs
Meronymy/Holonymy	mm,ms,mp	Semantic	Nouns
Entailment	ent	Semantic	Verbs
See also	sa	Lexical	Adjectives and verbs
Cause	cs	Semantic	Verbs
Participle	ppl	Lexical	Verbs-adjectives
Pertain	per	Lexical	Adjectives-adverbs, adjectives-nouns
Group	vgp	Semantic	Verbs

Table 1.2: Relations in WordNet. The column "Abbreviation" show the relation names, as they are called in the Prolog distribution of the WordNet database files. "Type" shows the type of relation, and "Word Forms" tells possible constraints on which word forms the relation holds between.

substance and part meronymy.

An ER diagram of WordNet is show in figure 1.2. Even though the participle relation connects verbs and adjectives, it only consists of 90 relations. This makes it difficult to use all the information stored in WordNet, because of this weak connection between different syntactic categories.

1.2 Previous research

1.2.1 Text Retrieval

In Information Retrieval, many underlying techniques have been tried on a wide variety of applications. Most commons are models based on symbol manipulation [Yan99, MSS]. This type of models relies on a correlation between words as symbols and their meaning, which only to some extend is present. In other words, the models do not take into account the different meanings of the words, which also is reflected in the results of their performance.

The problem of text retrieval is traditionally viewed as matching a query string against a set of texts, and thereby finding the documents that are relevant to the query [Voo98]. One common way of doing this is by using a vector space model [Voo98]. Here each text

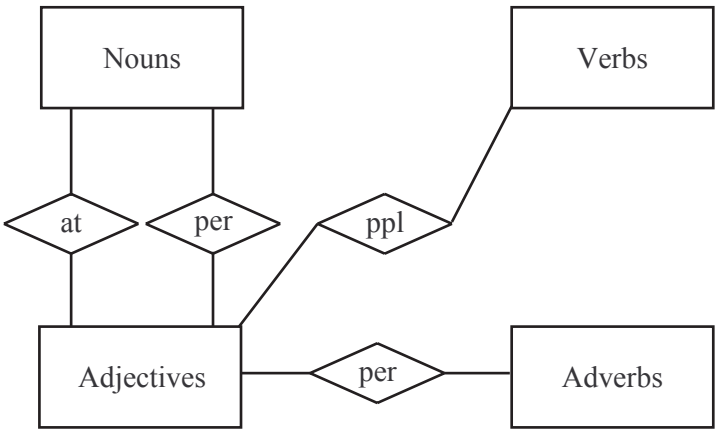

Figure 1.2: ER diagram showing relations between different syntactic categories in WordNet.

is represented by a T-dimensional vector, where T is the number of different words in the text. The length of the vector in a given direction, is given by the number of times a particular word occurs in the text. Given a query, the vector for that query is calculated, and matched against each vector of the text collection. Now only the text vectors that are similar to the query vector, is returned as results. Obvious advantages of using simple symbol-based approaches like this are:

- They are fast, robust, and well-known [Coh95a, Coh95b, Qui96].
- They can be made general, i.e. they do not need supervised configuration for use in different areas [MSS].

The biggest disadvantage of this kind of model, is that no better than half of the searched documents can be found. The best results of this kind of model, has achieved to find about half of the relevant documents. Using a precision/recall evaluation scheme, this means that no more than 50% of the found articles of a test set is in fact relevant, and no more than 50% of the truly relevant documents are found [Voo98].

A way to enable the vector space model to take into account the different senses of words, is to apply WordNet for resolving the senses of the Words. George Miller has done exactly that, by adding a new WordNet construct called a *hood* [Voo98]. The idea is to assign a number of categories to each word, and then to disambiguate the categories, so only one is left for each word. The category labeling is defined by the hood, which itself

is defined by the hyponymy relation. A hood is the synset that the word synset is a descendant of, as long as no other relations exists from the hood and down to the word synset. By assigning and disambiguating hoods automatically, the vector space model can be applied on the hood categories instead of the word symbols.

Unfortunately the results are at best, no better than the symbol based approach. For this reason I have chosen to base the developed system on another technique for automatic word sense resolution, namely lexical chains.

1.2.2 Text Categorization

In the previous section a system using hoods derived from WordNet as categories was discussed. These category labels was used for matching queries against texts, to find documents related to the query. The prototype system I am developing, is to be used for a category browsing system, which is quite different from the use of the hoods. Finding a hood of a word or synset does not indicate anything about the structure or the context the word is used in.

An attempt to use WordNet and lexical chaining for categorization, is Al-Halimi and Kazman's lexical trees [AHK98]. These trees are build in a manner like lexical chains, but instead of finding several chains, only one is build, preserving a tree structure between the words.

Another algorithm is QUESCOT by Stairmand [Sta97] also based on lexical chains. The chains are identified using Morris and Hirst's algorithm, but a new additional concept is introduced, namely lexical clusters, which should correspond to the context at fixed points in the analyzed documents. Here they define context as *"context can be specified by a word set consisting of keywords of the context."* This is done by considering the distribution of the terms found in the lexical chains throughout a document. It is assumed that there exists a tight relation between the contexts in the document, and the main topic. Therefore the most dominant context is selected as the topic of the document.

A system for categorization of email messages is described in [Tak95]. It is not based on WordNet, but describes a dynamic system, based on the semantic distance of the documents in a given category. When the distance becomes too big, the category is split into several sub categories. In this way the system is able to extend its structure while adding new documents. Unfortunately the paper only covers a preliminary study.

Chapter 2

Categorization

As the field of automatic categorization is very limited on published research, I have chosen to select techniques successfully applied to other applications of text retrieval, namely lexical chaining. I have then made some initial experiments on how to use it for text categorization.

The general idea is first to identify all lexical chains by using an adapted version of Morris and Hirst's original algorithm [MH91, SO95]. Hereafter I will try to identify relations between the found chains, to find out which are general/specific in meaning. Finally it may be possible to place the article in one or several sub categories, see figure 2.1.

I will start with a description of lexical chaining, and then continue with how the identified chains can be used for categorizing text.

2.1 Lexical chaining

The lexical chaining algorithm was developed by Morris and Hirst. The algorithm group the words of a given text, so that the words of each group has a close semantic relation. The purpose of the algorithms was to correctly disambiguate meanings of words, but also to give an indication of the text structure [SO95]. Because chains are limited in scope, they tend to indicate the structure of the text [MH91]. However, as stated by Stairmand, it is not clear whether or not there is a relation between the identified chains and the concept of text structure [Sta97].

The algorithm was developed by Morris and Hirst [MH91] based on Roget's International Thesaurus, which is a classification of words and phrases into ideas and concept. However, at the time of development, Roget's was not available electronically, and therefore the algorithm was never implemented. This was later done by St-Onge [SO95, HSO98] and Stairmand [Sta97], both adapted for WordNet.

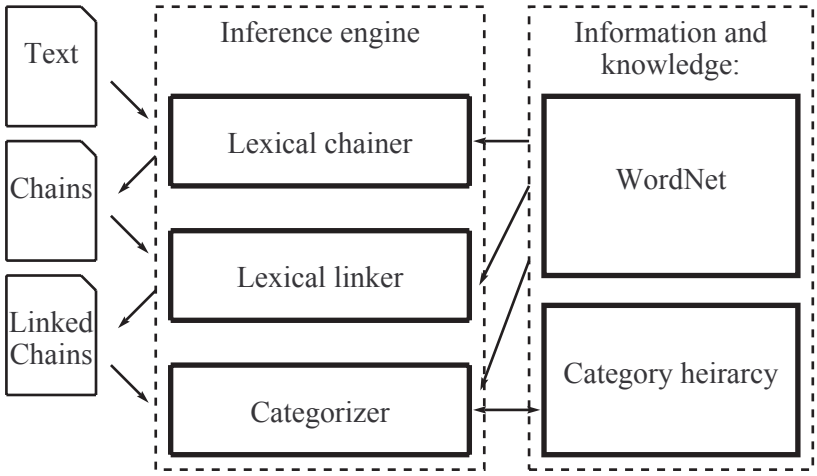

Figure 2.1: Overview of the prototype system. The lexical chainer and linker are the modules that are already developed, and presented in section 3.

St-Onge's algorithm is based on the original algorithm, along with the notion of *salience*, which was introduced by Okumura and Honda (see [SO95] for further reference.) Salience means that both recency and length of a chain is taken into account when building chains.

The algorithm works by looking up a word at a time, and then try to establish relations between the word and one of the chains. The relations can either be between the word symbols, or the synsets assigned to the words.

```
- Read word
- Skip word if found in stop list.
- Find semantically equal term in WordNet, by performing
  morphologic analysis
- Find relation between word and one of the words in the
  already initialized chains.
  - If none is found, make a new chain, and initialize it with
    the new word.
  - If relation is found, disambiguate synset senses of words
    in chain, by pruning all senses not used to find the
    relations between the words.
```

This continues until no more words is available. To use the salience concept, the chains

are searched in order of recency, i.e. the read word, is compared to the words in the chains most recently updated. If a chain has not been updated according to a special threshold value, the chain is not updated any more. Finding relations between words is done by applying one of several rules, each assigned a weight. Here each relation in WordNet has been given a *direction*, see table 3.1.

Extra strong The words are equal.

Strong There exists one or more direct horizontal relations between the synsets of the two words.

Medium A relation can be made using the following rules:

- The number of relations between the two synset pairs must be no greater than five.
- No more than one change in direction is allowed, unless it is horizontal.
- Upward relations is only allowed if no other direction changes has been made before.

The medium relations are further weighted, by their length and number of changes in direction. When a relation is found, synsets that are not directly connected to other synsets in the chain are pruned. In this way disambiguation is done incrementally whenever new words is added, and more information about what senses of the words is available.

As mentioned earlier, it can be difficult to make tight relations between lexical chains and other semantic entities. The word disambiguation might be coarse grained, but still lexical chaining has many useful purposes in information retrieval. Compared to other methods in this field, lexical chaining has with success been applied to detection of malapropisms [HSO98], and also initial experiments in categorization has been applied [Sta97].

St-Onge's implementation is based on version 1.4 of WordNet. The most important difference between this version and version 1.6, at least when lexical chaining is considered, is that the 1.4 lacks relations between verbs and nouns. This lead St-Onge to limit the analysis to nouns – When a word was found that could not be looked up in WordNet's noun database it was skipped. With the use of version 1.6 it may therefore be possible to improve performance of the output.

Another aspect of the algorithm is the running time – It is not discussed in the literature I know of. It is important when comparing application based on other IR methods, also to compare the running time. Many of these tools are used where large amount of information needs to be processed, and therefore the performance matters. The issue is discussed in section 3.3 and 3.6.

2.2 Categorization

The idea is to match lexical chains against a network of categories. As an example this could be the network of categories you can browse through at Yahoo's portal. Instead of representing the nodes by words the system should use synsets from WordNet.

When the chains of a document has been identified, they can be matched against the category network in several ways (see figure 2.2):

- Find relations between chains and category nodes. Hereafter try to find a path in the matched categories, and thereby selecting the one that is semantically close to the text.
- Find a path of relations between chains, and then search for a similar path in the category network.

Both methods gives the possibility to add new categories to the network. This can be done whenever a chain cannot be matched against a node in the network. Of course it is not a trivial process to connect the chain to the right nodes in the network.

A way to examine weather this is a good idea or not, could be to extract the category networks from e.g. Yahoo or the Open Directory Project[1], and then search for links between the categories. The lexical chaining implementation from this system could easily be adapted for this use. In this way it may be possible to discover properties of the relational structure in the network. This information could then be used to insert new nodes automatically.

However, due to time limitations, I have chosen to do another test. By implementing the lexical linker from figure 2.1, it will be possible to examine if links between the chains can be found. However, since the lexical chainer already have found relations between the words of the individual chains, the relation paths between chains are probably exhibiting a different structure. In the prototype system I will allow longer path, and path only containing upward or horizontal directions. In this way, I hope to find a path which have some resemblance to a category hierarchy.

[1] See `http://www.yahoo.com` and `http://dmoz.org` respectively.

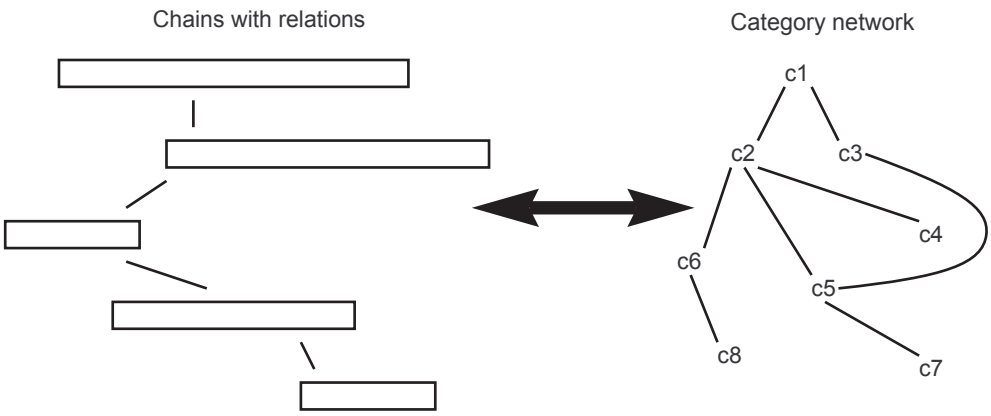

Figure 2.2: The categorization system surrounding the lexical chaining and linking system is searching for a path in the category hierarchy similar to the one found between the chains. An alternative could be to find relations between chains and category labels, and hereafter search for a path through the identified labels.

Chapter 3

Implementation

The prototype system implements the following components:

- Database access to WordNet.
- Morphologic analysis
- Lexical chainer
- Lexical chain linker
- Front-end

The system is developed in Sicstus Prolog [Swe98], using the Object Prolog extension of Sicstus. Object Prolog is an object oriented extension of the Edinburgh dialect of Prolog. Object Prolog is very similar to Prolog++ [Swe98]. The reason why I choose to develop the system in Object Prolog is primary because the description of the lexical chaining made by St-Onge relied on object oriented concepts. Futhermore I wanted to develop a program in an object oriented extension to Prolog, as I have never tried this before.

The source code for the lexical chainer developed by St-Onge and Stairmand is not public available. My implementation is primary based on [SO95], although some relations used in the chaining process might be different. The following description of the implementation is primary discussing implementation details, only important for the developed system.

3.1 Accessing the WordNet database

WordNet is distributed both as databases with native browsing interfaces for Windows, Unix and MacOS, and as Prolog readable files. I used the Prolog distribution for this

project, along with some files from the native distribution which were not included in the Prolog distribution. The total size of the files are about 19 Mb. To load and compile this, requires heavy computation. To avoid this I converted the files to external Prolog database files. In this way the programs using the database loads faster, and I could easily adjust the indexing mechanism to my needs.

The convert program relies on the files from the WordNet distribution to be available in the same directory as the program runs from. Also some of the files from the native distribution requires preprocessing, before they can be read by the converter program.

For accessing the WordNet database from Prolog, I have written a static class called `wordnet`, appendix A. References to the databases are opened when the file containing the class definition is loaded. The class contains two specialized methods:

`dbc` This method takes as argument a WordNet relation (see table 1.2 for predicate names) and instantiates its parameters.

`synsets` Given a word, returns a list of valid synset for that particular word. If the word cannot be found in the database, the method fails.

Furthermore it contains template descriptions and definitions of the relations used in lexical chaining.

3.2 Morphologic analysis

Morphologic analysis is in general divided in two different processes [Cov94]:

- Inflection which is the processes of transforming between different forms of a word, and
- Derivation which is a transformation between words of different syntactic categories.

For practical implementation purposes, it is always a tradeoff, of how much you want to list in the dictionary, and how much should be handled by a program. In the case of English, there are many regular inflections of words, and therefore it would be possible to write an inflection algorithm that can handle a large part of the morphologic transformation. On the other hand it would be hard to cover all inflections without listing some of them in lexicon. For example, there is no rule telling that the plural of *child* is *children* [Cov94].

The front-end of the native distribution of WordNet, handles the problem by doing inflection, and if that fails it tries to look the word up in a list of irregular word forms. This list is not supplied in the Prolog distribution, but I have chosen to convert the files from

the native version to Prolog, and then implementing my own inflectional transformation system. This is done in the `morph` object. Sending a `checkword` message to the `morph` object with a word as input, instantiates the second parameter with the same word, but in a possible different form, which can be found in the WordNet database. `validate` is used by `checkword` to ensure that the derivation can be found in WordNet's noun index. Furthermore it searches for compound words, and transform words from upper to lower case when needed.

3.3 Lexical chainer

The lexical chainer is implemented using the two dynamic classes `chain` and `lexchain`. When used, `lexchain` fetches a sentence at a time, and then try to add each word to one of the chains. The lexical chaining is done in `chain` and `lexchain`, by performing the morphologic analysis on the words, and allocating `chain` objects, containing the actual chains. By sending an `add` message to a `chain` object, the object responds by either accepting the word, or failing. If it fails the lexchain object backtrack and tries another chain object. If no chain will accept the word, a new chain is created, by sending a `new` message to the chain class.

The implementation differs from St-Onge's, in the way medium strength relations are found. Instead of finding all of them, then calculating a weight, and selecting the optimal, I perform a breath first search to find the relation with shortest path first. When the first relation is found it is selected, and the search is not continued.

The disambiguation is done in `chain` by first removing superfluous relations between synsets `prune_synrel/2`, and then removing the unconnected synsets, `prune_synsets/2`. The class uses `assert` and `retract` to store the relations between the words and synset.

As described in section 2.1 extra strong relations are sought throughout all chains before seeking for strong or medium relations. Therefore the running time must at least be $O(N \log N)$ where N is the size of the text. Strong and medium relations are only sought in a limited scope of chains. However, as will be seen from the next section, a typical analysis have one or two longer chains, which continues to be updated throughout the whole analysis. Therefore the running time of the strong and medium analysis will probably be somewhere near to that of the extra strong search. It is quite difficult to give an average time analysis, since it depends on which of the WordNet relations is used for constructing the medium relation search.

Name	Direction
Antonymy	Horizontal
Attribute	Horizontal
Cause	Down
Entailment	Down
Group	Horizontal
Holonymy	Down
Hypernymy	Up
Hyponymy	Down
Meronymy	Up
Participle	Horizontal
Pertain	Horizontal
See also	Horizontal
Synonymy	Horizontal

Table 3.1: Directions assigned to the WordNet relation, as it is done by St-Onge.

3.4 Lexical chain linker

The linking of the identified chains is done in the `lexlink` class. The process is very similar to what is done in the `chain` class, only the relation types allowed is different.

Each synset of a chain is matched against the synsets of the other chains using the `relation` method. `relation` is only allowed to use upward or horizontal links to find a path between the two synsets. The path must be no longer than ten relations.

It is important to note that the object identity of both the `lexchain` and the `chain` classes are of crucial importance for its success. Both classes use value assignment by specialized version of the `get/set` predicates. Another solution could be to pass the object values as parameters to the objects in a frame-like manner. This would keep the declarative semantics intact, but instead it might be confusing to keep passing long lists of parameters between the objects.

3.5 Front-end

The front-end consists of the predicate `test`. The article to be analyzed should be placed in the file `article.pl` before loading. The system gives a progression indication while searching for the chains, and finally prints out the chains and their interrelations.

No	Heading	Source	No./words
1	China warns Taiwan about making two states theory legal	CNN (web), Jan. 31. 2000	383
2	Bananas - Cultural directions	From www.plants.com	330
3	Tae kwon do	Encyclopedia Britannica, 1999 electronic edition	224

Table 3.2: The table show the four articles used in the testing of the developed prototype.

3.6 Testing

The testing of the system is done by applying three different articles as input to the system. A description of the articles are show in table 3.2. The results is printed in appendix B.

An initial test of the chainer was first done using the following words: Pear, apple, carrot, melon, tree, apples, blue, red, green and yellow. Here we would expect two chains as result, one containing fruits and vegetables, and one containing colors. In fact the result is:

```
[apple,melon_tree,carrot,apple,pear]
[yellow,green,red,blue]
```

Furthermore we see that the morph class seems to work in that the word 'apples' has been transformed to 'apple', and 'melon' and 'tree' has been made a compound word. Looking at the chains produced from the example texts, also shows that the words of the chains have good relations to each other.

However, the most crucial parameter for its success, I have found to be which relations is used. In the tests presented here I have restricted the analysis to the use of meronymy/holonymy, hypernymy/hyponymy and antonyms. When trying to include some of the other upward/downward relations, the search time was dramatically increased, without any better results. Also when trying to relax the constraints of the use of horizontal relations it was found that only one or two chains was found, containing words without any close semantic relations. These experiments indicate that the restrictions on the medium strength relation search, is of crucial importance for the algorithms success.

The results of the experiments shows that a number of lexical chains is found for each article. In all three tests one chain grows very big (covers about 30% of the article). However the words in the chains are all naturally related somehow. It is of course not an objective measure, weather my personal opinion is that they are naturally related or not. A better measure could be to make a large number of English speaking persons group the

words of the documents. By comparing the results with the lexical chains found by the developed system, a better measure of its success may be found.

Applying the lexical linker to the chains found for each article, showed that no relations between the chains could be found. In the lexical linker it is allowed to construct chains of length 10. However, if it is not at all possible to construct chains of that length using the selected relations, it is likely that no relations is found. This is because relations of length 5 have already been sought when constructing the chains. This problem could maybe be solved by using other relations than in the chaining process. An examination of which ones should be used, along with which constraints could be a possible continuation of this project.

Chapter 4

Discussion

Compared to the traditional ways of analyzing natural language, the use of lexical chains can be described as a hackers approach. Instead of trying to analyze a whole sentence by some predefined rules, everything which immediately can be assigned a meaning is used, and everything else is thrown away. The structure of sentences is not considered and only nouns are used. However, as the testing of the prototype system shows, the model is in fact quite robust, identifying words of similar meaning. In text categorization, the only thing of interest is to assign one, maybe two labels to the text, thereby placing it in a network of category labels. For this purpose, it is not needed to understand every single sentence of the text to be analyzed. Instead the rather coarse grained view of the lexical chains might be adequate.

An interesting property is the chaining algorithm's ability of specialization when analyzing a text. A text describing animals, humans, and plants, may be able to place all these references in the same chain, whereas a document describing humans in one context and animals in another, may be able to distinguish these things by placing the words in different chains. Although not tested here, this should be possible because of the dynamic word sense resolution. In other words this means that a system based on this resolution mechanism can be applied to many different areas without further adaption, because it is able to set a level of detail.

The system show robustness in the quality of output, but another aspect is the running time. As discussed in the tests the result is dependent on the number of relations used to find medium strength relations. However, I believe the running time can be reduced by better knowledge, on which combinations is likely to give a valid relation, and also the inclusion of verbs in the analysis. To give a better understanding of which paths are worth searching for, a statistical analysis could be performed.

The lexical linking process, is obvious not a success in its present implementation. However, I believe that experiments on finding relations in an existing category network,

will give more insight to what constraints should be used, when searching for relations between chains. But a better lexical linker is not the only thing that is needed. Also a matching against the existing network is not a trivial process to implement. This part should preferably also include feedback to the category network, in form of added categories.

An overall view of the subject shows that there are great possibilities in a database like WordNet, covering a wide variety of the English language. Although extension of the database is needed to include better connections of the different syntactic categories, more knowledge on how to use these relations is also needed. The developed prototype makes it possible to investigate these properties of WordNet. I think that what is done today using lexicographic databases like WordNet, is only the top of the iceberg. By using a system based on e.g. lexical chaining, I believe it is possible to make advanced language based interfaces, without a complete knowledge on the theory of natural language.

Bibliography

[AHK98] Reem Al-Halimi and Rick Kazman. Temporal indexing through lexical chains. In Fellbaum [Fel98], chapter 14, pages 333–351.

[Coh95a] William W. Cohen. Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference*, 1995.

[Coh95b] William W. Cohen. Text categorization and relational learning. In *Machine Learning: Proceedings of the Twelfth International Conference*, 1995.

[Cov94] Michael A. Covington. *Natural Language Processing for Prolog Programmers.* Prentice Hall, 1994.

[Fel98] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database.* MIT Press, 1998.

[Hir00] Graeme Hirst. Context as a spurious concept. In Alexander F. Gelbukh, editor, *CICLing-2000*, pages 273–285, 2000.

[HSO98] Graeme Hirst and David St-Onge. Lexical chains as representations of context for the detection and correction of malapropisms. In Fellbaum [Fel98], chapter 13, pages 305–332.

[KMF96] Jim Cowie Kavi Mahesh, Sergei Nirenburg and David Farwell. An assessment of cyc for natural language processing. Technical Report MCCS-96-302, Computing Research Laboratory, New Mexico State University, 1996.

[MH91] J. Morris and Graeme Hirst. Lexical cohesion computed by thesaural relations as an indicator of the structure of text. *Computational Linguistics*, 17(1):21–48, 1991.

[MSS] Marti Hearst Mehran Sahami and Eric Saund. Applying the multiple cause mixture model to text categorization. *?*, ?

[Qui96] J. R. Quinlan. Learning first-order definitions of functions. *Journal of Artificial Intelligence Research*, (5):139–161, 1996.

[SO95] David St-Onge. Detecting and correcting malapropisms with lexical chains. Master's thesis, Department of Computer Science, University of Toronto, 1995.

[Sta97] Mark A. Stairmand. Textual context analysis for information retrieval. In *SIGIR*, Philadelphia, 1997.

[Sti99] Rune R. Stilling. Using natural language to search the internet. Master's thesis, Computer Science, Roskilde University, 1999.

[Swe98] Swedish Institute of Computer Science. *SICStus Prolog User's Manual*, September 1998.

[Tak95] Juha Takkinen. An adaptive approach to text categorization and understanding – a preliminary study. Presented at the Fifth IDA Graduate Conference on Computer and Information Science, Linköping, November 1995.

[Voo98] Ellen M. Voorhees. Using wordnet for text retrieval. In Fellbaum [Fel98], chapter 12, pages 285–303.

[Yan99] Yiming Yang. An evaluation of statistical approaches to text categorization. *?*, 1999.

Appendix A

Source code

File: `main.pl`

```
% ============================================================================
% Main file for the lexical chainer and linker system.
%
% By Tue Haste Andersen <haste@diku.dk>, February 1999.
% Tested in Sicstus Prolog 3.7.1.
% ============================================================================

?- use_module(library(objects)).

?- consult(util).
?- consult(wordnet).
?- consult(morph).
?- consult(article).
?- consult(sentence).
?- consult(chain).
?- consult(lexchain).
?- consult(lexlink).


test :-
        lexlink::new(example),
        example::print_chains.
```

File: `lexlink.pl`

```
% ============================================================================
% Part of the lexical chainer and linker system.
%
% By Tue Haste Andersen <haste@diku.dk>, February 1999.
% Tested in Sicstus Prolog 3.7.1.
% ============================================================================

lexlink :: {
        super(lexchain) &

        dynamic chain_rel/2 &

        :- :use_module(library(ordsets), [list_to_ord_set/2, ord_member/2,
                                          ord_del_element/3, ord_union/2]) &

        new(Instance) :-
                super <: new(Instance) &
                :write('Linking...'), :nl,
                Instance::get(chain_ids(X)),
                Instance::chain_synsets(X,ChainSynsets),
                Instance::find_chain_rel(ChainSynsets,ChainSynsets) &

        % -------------------------------------------------------
        % Initiates the process of finding links between the
        % lexical chains.
        % -------------------------------------------------------
        find_chain_rel([],_) &
        find_chain_rel([X|Rest],ChainSynsets) :-
                :ord_del_element(ChainSynsets,X,C1),
                :ord_union(C1,C2),
                ::synset_rel(X,C2),
                ::find_chain_rel(Rest,ChainSynsets) &

        % -------------------------------------------------------
        % Succeeds if a relation exists between S1 and S2, where
        % S1 is a synset of C1 and S2 is a synset of C2.
        % -------------------------------------------------------
        synset_rel([],_) &
        synset_rel([C|C1],C2) :-
                :member(S1,C),
                ::relation(S1,S2),
```

```
        :member(S2,C2),
        :write('chain_rel '), :write(S1), :write(S2), :nl,
        assert(chain_rel(S1,S2)) &
synset_rel([_|C1],C2) :-
        ::synset_rel(C1,C2) &


% ------------------------------------------------------
% relation is true if there exists an upward link between
% the two synsets.
% ------------------------------------------------------
relation(From,To) :-
        ::relation(From,To,[]) &
relation(From,To,Old) :-
        :length(Old,L), :(L<10),
        wordnet::direction(Type,Dir), :member(Dir,[up,horizontal]),
        wordnet::link(Type,From,Temp,Pred),
        wordnet::dbc(Pred),
        (   Temp = To ;
            ::relation(From,To,[Temp|Old])) &

% ------------------------------------------------------
% Given a list of chain id's, returns a list of ordered
% synset sets. Only sets of lengths greater than one are
% included in the output list.
% ------------------------------------------------------
chain_synsets([],[]) &
chain_synsets([ID|R1],[S3|R2]) :-
        ID::get(chain(S1)),
%       :length(S1,L), :(L > 1), !,
        <:chain_extract_synsets(S1,S2),
        :list_to_ord_set(S2,S3),
        ::chain_synsets(R1,R2) &
chain_synsets([_,R1],R2) :-
             ::chain_synsets(R1,R2)
}.
```

File: `lexchain.pl`

```
% ============================================================================
% Part of the lexical chainer and linker system.
%
% By Tue Haste Andersen <haste@diku.dk>, February 1999.
% Tested in Sicstus Prolog 3.7.1.
% ============================================================================

% ----------------------------------------------------------------------------
% The lexchain class implements the chaining of an article, by reading
% each sentence, and construct lexical chains.
% ----------------------------------------------------------------------------
lexchain :: {
        super(object) &

        :- :use_module(library(lists), [delete/3]) &

        attributes([chain_ids([]), % ID's of all chains.
                    sentences([])])  & % ID's of all sentence objects.

        new(Instance) :-
                super <: instance(Instance),
                Instance :: chainer &

        % ----------------------------------------------------
        % Print out the values of the identified chains.
        % ----------------------------------------------------
        print_chains :-
                <: get(chain_ids(IDs)),
                <: print_chains(IDs) &

        print_chains([]) &
        print_chains([X|Rest]) :-
                X::get(chain(Y)),
                <: chain_words(Y,Z),
                :write(Z), :nl,
                <: print_chains(Rest) &

        % Given a chain list, with each element being [Word,Synsets],
        % returns a list of [Word,No-of-synsets] elements.
        chain_words([],[]) &
        chain_words([[W,S]|R1],[[W,N]|R2]) :-
```

```
        :length(S,N),
        <: chain_words(R1,R2) &

% ---------------------------------------------------
% Given a list of [Word,Synsets] pairs, returns a list
% of [Synsets] elements.
% ---------------------------------------------------
chain_extract_synsets([],[]) &
chain_extract_synsets([[_,S]|R1],[S|R2]) :-
        ::chain_extract_synsets(R1,R2) &


% ---------------------------------------------------
% Calls chainer/4 for each sentence in the article.
% ---------------------------------------------------
chainer :-
        sentence::get_words(SentenceNo,Words), !,
        get(chain_ids(C1)),
        <: chainer(Words,SentenceNo,C1,C2),
        set(chain_ids(C2)),
        <: chainer &
chainer :-
        :write('Done.'), :nl &

% Given a list of valid sentence words, updates chains.
chainer([],_,C,C) &
chainer([Word|Rest],Sentence,Cin,Cout) :-
        :write('Searching relation to '), :write(Word), :write(': '),
        <: addword(Word,Sentence,Cin,ID), !, :nl,
        :delete(Cin,ID,CTemp),
        <: chainer(Rest,Sentence,[ID|CTemp],Cout) &
chainer([Word|Rest],Sentence,Cin,Cout) :-
        !,
        :write('fail. '),
        chain::new(ID,Word),
        :write('Creating chain '), :write(ID),
        :write(', '), :write(Word), :nl,
        <: chainer(Rest,Sentence,[ID|Cin],Cout) &

% ---------------------------------------------------
% Given a word, sentence no, and a list of chain id's,
% addword tries to add the word to one of the chains.
```

```
        % ---------------------------------------------------
        addword(Word,Sentence,Chain,ID) :-
                <: addword(Word,Sentence,Chain,ID,extrastrong) &
        addword(Word,Sentence,Chain,ID) :-
                <: addword(Word,Sentence,Chain,ID,strong) &
        addword(Word,Sentence,Chain,ID) :-
                <: addword(Word,Sentence,Chain,ID,medium) &

        addword(Word,Sentence,[ID|_],ID,RelationType) :-
                ID::add(Word,Sentence,RelationType),
                :write('found '), :write(RelationType),
                :write(' to '), :write(ID), ! &
        addword(Word,Sentence,[_|Rest],ID,RelationType) :-
                <: addword(Word,Sentence,Rest,ID,RelationType)
}.
```

File: chain.pl

```
% ============================================================================
% Part of the lexical chainer and linker system.
%
% By Tue Haste Andersen <haste@diku.dk>, February 1999.
% Tested in Sicstus Prolog 3.7.1.
% ============================================================================

% ----------------------------------------------------------------
% This class holds the words of a chain along with the relations
% between them.
% ----------------------------------------------------------------
chain :: {
        super(wordnet) &

        dynamic word_rel/3 &
        dynamic syn_rel/3 &

        :- :use_module(library(lists), [member/2, non_member/2]) &
        :- :use_module(library(ordsets), [list_to_ord_set/2,
                                          ord_intersection/3]) &

        attributes([chain([]),     % Chain, containing word id's
                    recency(0)]) & % Holds the sentence number of the most
                                   % recent added word in the chain.

        new(Instance,Word) :-
                super <: instance(Instance),
                <: synsets(Word,Synsets),
                Instance :: set(chain([[Word,Synsets]])) &

        % ------------------------------------------------------
        % Adds the word Word to the chain and returns the
        % relation type for the added word.
        % ------------------------------------------------------
        add(Word,SentenceNo,RelationType) :-
                <: synsets(Word,Synsets),
                get(chain(C1)), get(recency(RecencyNo)),
                :member([W2,S2],C1),
                <: relation(RelationType,RecencyNo,SentenceNo,[Word,Synsets],
                            [W2,S2],Rel), !,
```

```
        % Update chain recency
        chain_recency(RecencyNo,SentenceNo),

        % Update chain
        set(chain([[Word,Synsets]|C1])),
        assert(word_rel(RelationType,Word,W2)),
        assert_synrel(Rel),
        prune_synrel(Word,W2),
        prune_synsets([[Word,Synsets]|C1],C2),
        set(chain(C2)) &


% -------------------------------------------------------
% Updates the recency of the chain if necessary.
% -------------------------------------------------------
chain_recency(RecencyNo,SentenceNo) :-
        :(SentenceNo > RecencyNo),
        set(recency(SentenceNo)) &
chain_recency(_,_) &

% -------------------------------------------------------
% Given a word of chain, return associated synsets. This
% method differs from wordnet::synsets in that only the
% synsets which have not been removed by the word sense
% disambiguation is returned.
% -------------------------------------------------------
chain_synsets(W,S) :-
        get(chain(C)),
        :member([W,S],C) &

% -------------------------------------------------------
% Asserts a list of new syn_rel's.
% -------------------------------------------------------
assert_synrel([]) &
assert_synrel([X|R]) :-
        assert(X),
        <: assert_synrel(R) &

% -------------------------------------------------------
% Lookup a synset relation
% -------------------------------------------------------
synset_rel(X,Y) :- <: syn_rel(_,X,Y) &
```

```
synset_rel(X,Y) :- <: syn_rel(_,Y,X) &

% -------------------------------------------------------
% Retract synset relation
% -------------------------------------------------------
retract_synset_rel(X,Y) :-
        <: syn_rel(_,X,Y), retract(syn_rel(_,X,Y)) &
retract_synset_rel(X,Y) :-
        <: syn_rel(_,Y,X), retract(syn_rel(_,Y,X)) &

% -------------------------------------------------------
% When a chain is updated with new relations,
% prune_synsets is called to remove superfluous synsets.
% -------------------------------------------------------
prune_synrel(W1,W2) :-

        % Find a relation from W2 to W3, using a synset S2,
        % which also connects W1 (S2a,S3a).
        <: chain_synsets(W1,R1),
        <: chain_synsets(W2,R2),
        :member(S1a,R1),
        :member(S2a,R2),
        <: synset_rel(S1a,S2a),
        <: chain_synsets(W3,R3), :non_member(W3,[W1,W2]),
        :member(S3a,R3),
        <: synset_rel(S2a,S3a),

        % Find a relation between a synset in W2
        % and W3 not connected to W1 (S2b,S3b)
        :member(S2b,R2),
        <:not((<:synset_rel(S1b,S2b), :member(S1b,R1))),
        <: synset_rel(S2b,S3b), :member(S3b,R3),

        % Ensure that another relation than (S2b,S3b)
        % exists from W3.
        : member(S3c,R3), <: synset_rel(S3c,Sx),
        (   :(S3c \== S3b) ; :(Sx \== S2b) ),

        % Delete (S2b,S3b)
        <: retract_synset_rel(S2b,S3b), !,

        % Try to find other relations
```

```
        <: prune_synrel(W2,W3),
        <: prune_synrel(W1,W2) &

prune_synrel(_,_) &

not(X) :- ((X, !, :fail) ; :true ) &

% -------------------------------------------------------
% After prune_synrel, prune_synsets removes synsets with
% no relations to other synsets.
% -------------------------------------------------------
prune_synsets([],[]) &
prune_synsets([[W,R1]|C1],[[W,R2]|C2]) :-
        <: prune_synsets1(R1,R2),
        <: prune_synsets(C1,C2) &

prune_synsets1([],[]) &
prune_synsets1([X|R1],[X|R2]) :-
        <: syn_rel(_,X,_),
        <: prune_synsets1(R1,R2) &
prune_synsets1([X|R1],[X|R2]) :-
        <: syn_rel(_,_,X),
        <: prune_synsets1(R1,R2) &
prune_synsets1([X|R1],R2) :-
        <: prune_synsets1(R1,R2) &

% -------------------------------------------------------
% Given two words with corresponding synsets, succedes if
% there is a relation between the two words, and return a
% list of synset relations of the form:
% rel(Weight,Synset_id1,Synset_id2).
% -------------------------------------------------------

% The words are equal
relation(extrastrong,_,_,[W,_],[W,S],Rel) :-
        <: relation_list(extrastrong,S,S,Rel), ! &

% The words share one or several synsets
relation(strong,RecencyNo,SentenceNo,[_,S1],[_,S2],Rel) :-
        :(Temp is RecencyNo+7), :(Temp >= SentenceNo),
        :ord_intersection(S1,S2,S3),
        :(S3 \== []), !,
```

```
        <: relation_list(strong,S3,S3,Rel) &

% Horizontal link exists between the two.
relation(strong,RecencyNo,SentenceNo,[_,S1],[_,S2],Rel) :-
        :(Temp is RecencyNo+7), :(Temp >= SentenceNo),
        <: relation_strong(S1,S2,Rel),
        :(Rel \== []), ! &

relation_strong([],_,[]) &
relation_strong([S1|R1],R2,[syn_rel(strong,S1,S2)|Rel]) :-
        <: direction(Type,horizontal),
        <: link(Type,S1,S2,Pred),
        <: dbc(Pred),
        :member(S2,R2),
        <: relation_strong(R1,R2,Rel) &
relation_strong([_|R1],R2,Rel) :-
        <: relation_strong(R1,R2,Rel) &

% Medium-strenght relations
relation(medium,RecencyNo,SentenceNo,[_,S1],[_,S2],Rel) :-
        :(Temp is RecencyNo+3), :(Temp >= SentenceNo),
        <: relation_med(S1,S2,Rel),
        :(Rel \== []), ! &

relation_med([],_,[]) &
relation_med([S1|R1],R2,[syn_rel(medium,S1,S2)|Rel]) :-
        <: relation_med([[S1,[]]],R2,_,S2), !,
        <: relation_med(R1,R2,Rel) &
relation_med([_|R1],R2,Rel) :-
        <: relation_med(R1,R2,Rel) &

% Perform breath first search, to ensure that shortest
% path is found first!
relation_med(R1,R2,Path,S3) :-
        :member([_,TestPath],R1), :length(TestPath,L), :(L < 5),
        self(Self), :setof(S2, Self::breath_rel(R1,S2),Set), !,
        (   (   :member(S3,R2), :member([S3,Path],Set)) ;
            (   <: relation_med(Set,R2,Path,S3))) &

breath_rel(R1,[S2,[Dir|Path]]) :-
        :member([S1,Path],R1),
        <: analyze_path(Path,Type,Dir),
```

```
        <: link(Type,S1,S2,Pred),
        <: dbc(Pred) &

% Given a weight, two lists of synsets, returns a list of
% relations for the synsets.
relation_list(_,[],[],[]) &
relation_list(Weight,[S1|R1],[S2|R2],[syn_rel(Weight,S1,S2)|Rest]) :-
        relation_list(Weight,R1,R2,Rest) &


% -----------------------------------------------------
% analyze_path returns a valid relation type to apply to
% the given path (Type).
% -----------------------------------------------------
analyze_path([],Type,Dir) :-
        <: direction(Type,Dir) &
analyze_path(Path,Type,Dir) :-
        <: upward_direction(Path),
        <: direction(Type,Dir) &
analyze_path(Path,Type,Dir) :-
        <: one_direction(Path),
        <: direction(Type,Dir),
        :(Dir \== up) &
analyze_path([Dir|_],Type,Dir) :-
        <: direction(Type,Dir) &
analyze_path([horizontal|_],Type,down) :-
        <: direction(Type,down) &

% -----------------------------------------------------
% Ensures that the given path only contains upward
% directions.
% -----------------------------------------------------
upward_direction([up]) &
upward_direction([up|Rest]) :-
        <: upward_direction(Rest) &

% -----------------------------------------------------
% Ensures that only one direction is taken in the given
% path.
% -----------------------------------------------------
one_direction(Path) :-
        <: one_direction(Path,_) &
```

```
        one_direction([Dir],Dir) &
        one_direction([Dir|Rest],Dir) :-
                <: one_direction(Rest,Dir)
}.
```

File: morph.pl

```
% ============================================================================
% Part of the lexical chainer and linker system.
%
% By Tue Haste Andersen <haste@diku.dk>, February 1999.
% Tested in Sicstus Prolog 3.7.1.
% ============================================================================

% ----------------------------------------------------------------------------
% This class performs lookup in wordnet, by trying to make compound words,
% transform case from upper to lower, inflection and derivation.
% ----------------------------------------------------------------------------
morph :: {
        super(object) &

        :- :use_module(library(lists), [append/3]) &

        % ------------------------------------------------
        % Given a list of words, returns a term found in
        % wordnet, and the rest of the words. The term is made
        % by first concatenating the first three words of the
        % list, the the two first, and finally the first. If
        % none is found, the first word is ignored.
        % ------------------------------------------------
        get_next_word(List,Word,Rest) :-
                <: compound_word(List,WordTemp,Rest),
                <: checkword(WordTemp,Word), ! &
        get_next_word(List,Word,Rest) :-
                <: compound_word(List,WordTemp1,Rest),
                :lower_case(WordTemp1,WordTemp2),
                <: checkword(WordTemp2,Word), ! &
        get_next_word([_|List],Word,Rest) :-
                <: get_next_word(List,Word,Rest) &

        % ------------------------------------------------
        % Concatenation of terms, by the way described above.
        % ------------------------------------------------
        compound_word([X,Y,Z|Rest],Word,Rest) :-
                :atom_concat(X,'_',T1),
                :atom_concat(T1,Y,T2),
                :atom_concat(T2,'_',T3),
                :atom_concat(T3,Z,Word) &
```

```
compound_word([X,Y|Rest],Word,Rest) :-
        :atom_concat(X,'_',T),
        :atom_concat(T,Y,Word) &
compound_word([Word|Rest],Word,Rest) &

% ------------------------------------------------------
% checkword, takes as input a noun, and transforms it
% into the baseform. If the predicate fails it is
% either because the input word is not a noun, or it
% was not identified as such.
% ------------------------------------------------------

% Check the word as it is.
checkword(Word,Word) :-
        <: validate(Word), ! &

% Try to look it up in the exception list, of irregular
% forms.
checkword(WordIn,WordOut) :-
        wordnet::dbc(exc(noun,WordIn,WordOut)),
        <: validate(WordOut), ! &

% Try morphing of word - remove "ful" ending before
% doing so.
checkword(WordIn,WordOut) :-
        <: ending(WordIn,Base,ful),
        <: morphword(Base,WordTemp),
        <: ending(WordOut,WordTemp,ful),
        <: validate(WordOut), ! &

% Try morphing words which is not ending on "ss".
checkword(WordIn,WordOut) :-
        (   <: ending(WordIn,_,ss), !, :fail ) % Stop if word
        ;                                      % ends on ss.
        (   <: morphword(WordIn,WordOut), % Otherwise try to morph
            <: validate(WordOut), !) &

% Again, try morphing of word, by removing what is after
% possible "-" characters in the word.
checkword(WordIn,WordOut) :-
        <: beforedelimeter(WordIn,WordTemp),
        <: morphword(WordTemp,WordOut),
```

```
        <: validate(WordOut), ! &

    % ------------------------------------------------------
    % Succeds if the word is a noun represented in wordnet,
    % and the word is not in the stoplist.
    % ------------------------------------------------------
    validate(Word) :-
        wordnet::dbc(stop(Word)), !, :fail &
    validate(Word) :-
        wordnet::dbc(s(_,_,Word,n,_,_)) &

    morphword(WordIn,WordOut) :-
        <: suffix(noun,EndOld,EndNew),
        <: ending(WordIn,Base,EndOld),
        <: ending(WordOut,Base,EndNew) &

    suffix(noun,'s','') &
    suffix(noun,'ses','s') &
    suffix(noun,'xes','x') &
    suffix(noun,'zes','z') &
    suffix(noun,'ches','ch')&
    suffix(noun,'shes','sh')&

    ending(Word,Base,End) :-
        :ground(Word), :ground(End),
        :name(Word,WordList),
        :name(End,EndList),
        :append(BaseList,EndList,WordList),
        :name(Base,BaseList) &
    ending(Word,Base,End) :-
        :ground(Base), :ground(End),
        :name(Base,BaseList),
        :name(End,EndList),
        :append(BaseList,EndList,WordList),
        :name(Word,WordList) &

    beforedelimeter(WordIn,WordOut) :-
        :ground(WordIn),
        :name(WordIn,WordInList),
        :append(WordOutList,['-'|_],WordInList),
        :name(WordOut,WordOutList)
}.
```

File: `wordnet.pl`

```
% ========================================================================
% WordNet access for the lexical chainer and linker system.
%
% By Tue Haste Andersen <haste@diku.dk>, February 1999.
% Tested in Sicstus Prolog 3.7.1.
% ========================================================================
?- use_module(library(db)).
?- use_module(library(objects)).

databases([[s_db,s],[sim_db,sim],[ant_db,ant],[stop_db,stop],[exc_db,exc],
           [mm_db,mm],[mp_db,mp],[hyp_db,hyp],[sa_db,sa],[at_db,at],
           [per_db,per],[cs_db,cs],[ent_db,ent],[g_db,g],[ppl_db,ppl]]).

open_database :-
        databases(DB),
        open_database(DB).
open_database([]).
open_database([[DB,Name]|Rest]) :-
        db_open(DB,read,_,Ref),
        assert(db_ref(Name,Ref)),
        open_database(Rest).

?- open_database.

% ------------------------------------------------------------
% wordnet is a static object containing methods for accessing the
% WordNet database in a convenient way.
% ------------------------------------------------------------
wordnet :: {
        super(object) &

        :- :use_module(library(lists), [non_member/2]) &
        :- :use_module(library(ordsets), [list_to_ord_set/2]) &

        % ------------------------------------------------------
        % DataBase Consult. Consults the given term in the
        % associated external database.
        % ------------------------------------------------------
        dbc(Term) :-
                :functor(Term,Head,_),
                :db_ref(Head,R),
```

```
        :db_fetch(R,Term,_) &

% -------------------------------------------------------
% Given a word, return a list of corresponding synsets.
% Fails if no noun synsets is found.
% -------------------------------------------------------
synsets(Word,List) :-
        <: synsets(Word,Temp,[]),
        :length(Temp,X), :'>'(X,0),
        :list_to_ord_set(Temp,List) &
synsets(Word,[Sid|Rest],Retreived) :-
        <: dbc(s(Sid,_,Word,n,No,_)),
        :non_member(No,Retreived), !,
        <: synsets(Word,Rest,[No|Retreived]) &
synsets(_,[],_) &

% -------------------------------------------------------
% Specifies a direction on each WordNet relation type.
% -------------------------------------------------------
direction(ant,horizontal) &
%direction(at,horizontal) &
%direction(per,horizontal) &
%direction(sim,horizontal) &
%direction(sa,horizontal) &
%direction(per2,horizontal) &

direction(hyper,up) &
direction(mm,up) &
direction(ms,up) &
direction(mp,up) &
%direction(cs2,up) &

direction(hypo,down) &
direction(mh,down) &
direction(hs,down) &
direction(hp,down) &
%direction(cs,down) &
%direction(ent,down) &

% -------------------------------------------------------
% Templates for different relation types. Instantiating
% R1 or R2 to a synset, returns a template for use
```

```
        % with wordnet::dbc.
        % ------------------------------------------------------
        link(sa,R1,R2,sa(R1,_,R2,_)) &
        link(ant,R1,R2,ant(R1,_,R2,_)) &
        link(at,R1,R2,at(R1,R2)) &
        link(per,R1,R2,per(R1,_,R2,_)) &
        link(per2,R1,R2,per(R2,_,R1,_)) &
        link(sim,R1,R2,sim(R1,R2)) &
        link(cs,R1,R2,cs(R1,R2)) &
        link(cs2,R1,R2,cs(R2,R1)) &
        link(ent,R1,R2,ent(R1,R2)) &
        link(hyper,R1,R2,hyp(R1,R2)) &
        link(hypo,R1,R2,hyp(R2,R1)) &
        link(mm,R1,R2,mm(R1,R2)) &
        link(mh,R1,R2,mm(R2,R1)) &
        link(ms,R1,R2,ms(R1,R2)) &
        link(hs,R1,R2,ms(R2,R1)) &
        link(mp,R1,R2,mp(R1,R2)) &
        link(hp,R1,R2,mp(R2,R1))
}.
```

File: sentence.pl

```
% ============================================================================
% Part of the lexical chainer and linker system.
%
% By Tue Haste Andersen <haste@diku.dk>, February 1999.
% Tested in Sicstus Prolog 3.7.1.
% ============================================================================

% ----------------------------------------------------------------------
% Class holding the words of a sentence. When the object is created
% a sentence number is given as paramter. The words of the sentence
% is morphologically trasformed, and verified to be represented in the
% nuon index of WordNet. All the words satisfiying these conditions are
% stored in this object.
% ----------------------------------------------------------------------
sentence :: {
        super(article) &

        attributes([number(0)]) & % Sentence number

        get_words(Number,Words) :-
                get(number(X)),
                :(Y is X+1),
                <: sentence(Y,SentenceWords),
                (   (   <: normalize_words(SentenceWords,Words),
                        set(number(Y)),
                        Number = Y ) ;
                    (   set(number(Y)), get_words(Number,Words))) &


        normalize_words([],[]) &
        normalize_words(Words,[VerifiedWord|Rest2]) :-
                morph::get_next_word(Words,VerifiedWord,Rest1),
                normalize_words(Rest1,Rest2)
}.
```

File: `util.pl`

```
% ============================================================================
% Misc. predicates for the lexical chainer and linker system.
%
% By Tue Haste Andersen <haste@diku.dk>, February 1999.
% Tested in Sicstus Prolog 3.7.1.
% ============================================================================

% ----------------------------------------------------------------
% This predicate is included in Sicstus 3.8, but to run the
% program under 3.7 it is needed.
% ----------------------------------------------------------------
atom_concat(A,B,AB) :-
        ground(A), ground(B),
        name(A,Alist),
        name(B,Blist),
        append(Alist,Blist,ABlist),
        name(AB,ABlist).

% ----------------------------------------------------------------
% Converts the term T1 to a term T2, only consisting of lower
% cases.
% ----------------------------------------------------------------
lower_case(T1,T2) :-
        name(T1,Upcase),
        convert_case(Upcase,Lowcase),
        name(T2,Lowcase).

% ----------------------------------------------------------------
% Given a list of character codes, convert those of upper case to
% lower case.
% ----------------------------------------------------------------
convert_case([],[]).
convert_case([U|R1],[L|R2]) :-
        U > 64, U <91,   % Verify that the char is uppercase.
        L is U + 32,
        convert_case(R1,R2).
convert_case([L|R1],[L|R2]) :-
        convert_case(R1,R2).
```

File: `article.pl`

```
% ============================================================================
% Test data for the lexical chainer and linker system.
%
% By Tue Haste Andersen <haste@diku.dk>, February 1999.
% Tested in Sicstus Prolog 3.7.1.
% ============================================================================

?- use_module(library(objects)).

article :: {
  super(object) &

  name(einstein) &


  sentence(1,[pear,apple,carrot,melon,tree,apples,blue,red,green,yellow])
}.
```

Appendix B

Test results

The following shows the found lexical chains for the three examples (see table 3.2.) For each word the number of associated synsets is shown next to it. Because the example texts are copyrighted they are not included here.

```
Title: Tae kwon do (running time: 54 minutes on Intel Pentium, 100MHz)

Chains:

[[martial_art,1],[tae_kwon_do,1],[tae_kwon_do,1],[martial_art,1],
 [tae_kwon_do,1],[karate,1],[tae_kwon_do,1]],
[[contact,1],[short,1],[opponent,1],[sparring,1],[free,1],[practice,1],
 [student,1],[attack,1],[sequence,1],[set,1],[sparring,1],[step,1],[id,1],
 [combination,1],[sparring,1],[basic,1],[practice,1],[student,1],[grade,1],
 [rank,1],[series,1],[development,1],[spiritual,1],[founder,1],[principal,3],
 [general,1],[South_Korean,1],[defense,1],[form,1],[art,4],[art,4]]
[[blow,1],[counter,1],[punch,1],[kick,1],[jump,1],[standing,1],[combat,1],
 [kicking,1]]
[[In,1]]
[[lower,1]]
[[proficiency,2]]
[[high,7]]
[[Korean,2],[Korean,2],[Korean,2]]


Title: China warns Taiwan about making two states theory legal
       (Running time: 55 minutes on Intel Pentium II).

[[democracy,1],[mainland,1],[society,1],[formula,2],[Taiwan,1],
 [capitalist,1],[run,1],[Beijing,1],[system,1],[country,1],[formula,2],
 [China,1],[part,1],[Taiwan,1],[state,1],[back,1],[Taiwan,1],[policy,1],
```

```
 [official,1],[leadership,1],[senior,1],[member,1],[people,1],[people,1],
 [Taiwanese,1],[safety,2],[property,1],[life,1],[fire,1],[action,1],
 [people,1],[prospect,1],[Taiwan,1],[constitution,1],[state,1],[force,1],
 [Taiwan,1],[China,1],[Taiwan,1],[split,1],[feature,1],[state,1],
 [people,1],[state,1],[treat,1],[side,1],[President,1],[Taiwanese,1],
 [China,1],[voter,1],[party,1],[moderate,1],[voter,1],[separatist,1],
 [candidate,1],[threat,1],[China,1],[Taiwan,1],[people,1],[threat,1],
 [premier,2],[Taiwan,1],[China,1],[threat,1],[leader,1],[Chinese,1],
 [top,2],[Communist_Party,2],[people,1],[fire,1],[island,1],
 [Taiwanese,1],[China,1],[Taiwan,1]]
[[embrace,3]]
[[wait,2]]
[[unification,2]]
[[Macau,1]]
[[Hong_Kong,1]]
[[autonomy,2]]
[[theory,3],[theory,3],[theory,3]]
[[editorial,1],[editorial,1],[daily,1],[daily,1],[daily,1],[daily,1],
 [newspaper,1],[daily,1],[editorial,1],[separate,1]]
[[Sunday,1], [March,1],[days,1],[Sunday,1]]
[[playing,1],[election,2],[warning,1],[warning,1],[playing,1]]
[[reunification,1]]
[[basis,1],[law,1],[sentiment,1],[law,1],[claim,1]]
[[independence,1],[independence,1],[tension,2],[tension,2],
 [independence,1],[scare,1],[independence,1],[status,1]]
[[put,1]]
[[attempt,2]]
[[Lee,1],[Lee,1],[Lee,1]]
[[plot,4]]
[[true,1]]
[[equal,1],[equal,1]]
[[July,1]]
[[fear,2]]
[[stance,1]]
[[Chen,1]]
[[vice,2]]
[[war,2],[ruling,1]]
[[wage,1]]
[[leaders,1]]


Title: Bananas - Cultural directions.
```

[[result,1],[leave,2],[leave,2],[leave,2],[receipt,1]]
[[blend,3],[blend,3],[blend,3]]
[[foliage,2],[fertilizer,1],[liquid,1],[green,1],[ground,1],[air,4],
 [growth,1],[plant,1],[banana,2],[water,2],[production,1],[fruit,1],
 [growth,1],[fall,1],[In,1],[drinker,2],[feeder,1],[plant,1],
 [banana,2],[soil,1],[plant,1],[banana,2],[sun,1],[fruit,1],
 [produce,1],[banana,2],[plant,1],[banana,2],[rapid,1],[growth,1],
 [plant,1],[banana,2],[shoot,1],[produce,1],[corm,1],[dry,1],
 [water_plant,1],[ground,1],[pot,1],[plant,1],[stem,1],[change,1],
 [level,1],[plant,1],[banana,2],[plant,1],[soil,1],[peat,1],
 [mixture,1],[soil,1],[soil,1],[plant,1],[plant,1],[root,2],
 [plant,1],[banana,2],[plant,1]]
[[flowering,2]]
[[present,1],[long,1],[month,2],[month,2]]
[[temperature,2]]
[[winter,1]]
[[week,3]]
[[daily,1]]
[[summer,1]]
[[heavy,2],[heavy,2]]
[[normal,1]]
[[yellow,1]]
[[turn,12]]
[[resumption,1]]
[[lateral,1]]
[[times,2]]
[[soak,1],[planting,1],[shipping,1],[planting,1],[damage,1]]
[[season,3]]
[[days,1]]
[[coloration,1]]
[[growing,1]]
[[prior,1]]
[[fungicide,1]]
[[spray,6]]
[[dip,6]]