

Språkbehandling och datalingvistik

Projektarbeten 2002



Handledare: Pierre Nugues



LUNDS UNIVERSITET

Institutionen för Datavetenskap

<http://www.cs.lth.se>

Printed in Sweden
Lund 2004

Innehåll

Florian Eisl: Discovery of Morphemes in Swedish	5
Patricia Grudziecka, Björn Isaksson: Using salience to rank documents	11
Jon Hasselgren, Erik Montnemery and Markus Svensson: HMS: A Predictive Text Entry Method Using Bigrams	16
Mattias Johansson and Jonas Håkansson: The Artificial Librarian - A Database Dialogue Systems	23
Richard Johansson A Morphological Parser for Estonian	33
Erik Lindvall and Johan Nilsson: Extracting Information from Sport Articles in Swedish Using Pattern Recognition	43
Hossein Motallebipour, August Bering: A Spoken Dialogue System to Control Robots	53
Klas Skogmar and Johan Olsson,,: Clustering documents with vector space model using n-grams	59
Marcus Uneson, HORACE—an artificial columnist	65



LUNDS UNIVERSITET

Institutionen för Datavetenskap

<http://www.cs.lth.se>

Discovery of Morphemes in Swedish

Florian Eisl

Lund Institute of Technology
Department of Computer Science
Box 118
S-221 00 Lund, Sweden
x02fe@efd.lth.se

Abstract

This document describes an overview of two methods which allows to discover syntactic structures from an untagged corpus.

The first part describes the algorithm by Hervé Déjean – how it works, and shows the result of the algorithm applied to a Swedish corpus. A text by Selma Lagerlöf and a Swedish dictionary.

The second part shows the basics of the algorithms developed by Patrick Schone and Daniel Jurafsky.

1 Introduction

The morphological analysis is basically the segmentation of words into components that form the word by concatenation. From a practical point of view, the development of a fully automated morphology generator would be of considerable interest, since we still need good morphologies of many European languages and to produce a morphology of a given language by hand can take weeks or months. With the fact that a lot of text

is available online it is of great interest to develop morphologies of particular stages of a language, and the process of automatic morphology writing can simplify this stage, where there are no native speakers available.¹

2 Hervé Déjean – Morphemes

The idea of Hervé is based on the approach by Harris and is characterized by two facts: (a) the use of corpora and (b) the use of the notion of distribution instead of the sense of elements. The distribution of an element is the set of environments in which the element occurs.

Only untagged and non artificial corpora without specific knowledge about the studied language is used. They try to discover the structures of a natural language from raw texts of this language. This kind of discovery is possible if there are some expectations of the structure of the Natural Language and some formal properties are used.

The method relies on structural linguistic concepts: the morpheme, the chunk and the linearity of the language, i.e. the corpus is

¹ Goldsmith, John: Unsupervised Learning of the Morphology of a Natural Language

composed of a unidimensional sequence of elements.²

2.1 Morpheme Discovery – How it works

The algorithm is based on the number of different letters which follow a given sequence of letters. The increase of this number indicates a morpheme boundary. For instance, after the English sequence *direct*, we only find, in our corpus, one letter *t*. After *direct*, we find four letters: *i*, *l*, *o*, and *e* (*directly*, *director*, *directed*, *direction*). This increase indicates a boundary between the root (*direct*) and the suffixes (*-ion*, *-ly*, *-or* and *-ed*). The algorithm works well when the corpus contains enough occurrences of a stem family. But, it may generate wrong segmentations. For example from the list started, *startled*, *startling*, the algorithm outputs this segmentation: *start-ed*, *start-led*, *start-ling*. The errors occur when two kinds of stem families are used for the segmentation.³

The new idea for improving the segmentation now is to divide this operation into three steps. The first step computes the list of the most frequent morphemes. The second step is to extend this list by using the discovered morphemes already generated. And the third and last step is the segmentation of the words using the before produced morphemes. The illustration is only done for the segmentation of the suffixes but to get the prefixes the same algorithm can be used just with the reverse letters of the words.

2.1.1 Discover the most frequent morphemes

The aim is to find beginnings or endings of words which have the following property: after a given sequence of letters, we count the number of different letters. If this number is higher than a threshold (e.g. half the letters of

the alphabet), we got a so called morpheme boundary, expect in the case that we are in the sequence which corresponds to another, to a longer morpheme, a case which can be detected. This can be illustrate by simple example, before the sequence “*on*” we found 20 different letters therefore “*on*” may be the morpheme. But 154 of these words in the used corpus end with “*ion*” out of 293 which and end with “*on*”. Now it can be seen that the longest sequence “*ion*” represents more then 50% of the words ended by “*on*” and due to this it can be considered that the morpheme is not “*on*”. “*on*” is only a part of the morpheme “*ion*”.

The most frequent morphemes of the English and German language can be seen in the following table:

English	German
-e	-en
-s	-e
-ed	-te
-ing	-ten
-al	-er
-ation	-es
-ly	-lich
-ic	-el
-ent	

Table 1: The most frequent morphemes of English and German⁴

² Déjean, Hervé: Morphemes as Necessary Concept for Structures Discovery from Untagged Corpora

³ Déjean, Hervé: Morphemes as Necessary Concept for Structures Discovery from Untagged Corpora

⁴ Déjean, Hervé: Morphemes as Necessary Concept for Structures Discovery from Untagged Corpora

The most frequent morphemes of the Swedish language:

Swedish (selma)	Swedish (dictionary)
-ar	-erna
-er	-ningarna
-en	-ade
	-ar
	-ligt
	-ligast
	-ningarnas
	-ernas
	-ades
	-nings
	-ens
	-ers
	-ets
	-ad
	-ning
	-en

Table 2: The most frequent morphemes of the Swedish language

The first column is the result of text written by Thelma Lagerlöf. This corpus consists out of about 1.000.000 words.

The result in the second column is received using a Swedish dictionary as the corpus. The dictionary consists of about 120.000 words.

2.1.2 Discover other morphemes

After the most frequent morphemes of a language are found this morphemes can be used to find out other morphemes. This can be done using the following rule: For a given sequence of letters it can be checked if the next sequences of letters correspond to morphemes already found. If half of them belongs to the morphemes found, then the others can also be considered as morphemes of the language. This can be seen in the following table for the English language:

Morphemes found	words	New Morphemes
	light	
-s	lights	
-ed	lighted	
-ing	lighting	
-ly	lightly	
-er	lighter	
	lightness	-ness
	lightest	-est
	lighten	-en

Table 3: Table of other morphemes of the English⁵

This algorithm is not perfect and also wrong morphemes are generated, but their frequency is very low. To make sure that we get only correct morphemes we use a threshold (five in practice). The morphemes with a frequency lower than the threshold are not found. The list of the received morphemes may greatly depend on the type of corpus used. The number of morphemes depends on the morphology of the language. What can be found out is, that morphemes have a similar behavior as words, a small number of them possesses a high frequency and corresponds to the mayor occurrences of the corpus.

2.1.3 Segmentation of the words

After all morphemes are found we use this morphemes to segment all the words in the corpus. The segmentation is done be using the longest match algorithm. This means that we segment each word with the longest morpheme that matches the beginning or ending of the word.

3 Patrick Schone and Daniel Jurafsky - Morphemes

A knowledge free algorithm which automatically induce the morphology structures of a language. The algorithm takes as input a

⁵ Déjean, Hervé: Morphemes as Necessary Concept for Structures Discovery from Untagged Corpora

large corpus and produces as output a set of conflation sets indicating the various inflected and derived forms for each word in the language. An example for this can be the word “abuse”. The result would contain the following words: “abuse”, “abused”, “abuses”, “abusive”, “abusively” and so on. The algorithm extends earlier approaches to morphology induction by combining various induced information sources: the semantic relatedness of the affixed forms using a Latent Semantic Analysis approach to corpus-based semantics, affix frequency, syntactic context and transitive closure. The algorithm achieves an F-score of 88.1% on the task of identifying conflation sets in English. The algorithm is also applied to German and Dutch and evaluated on its ability to find prefixes, suffixes and circumfixes in these languages.⁶

3.1 Morpheme Discovery – How it works

In the picture below an overview over this approach is shown.

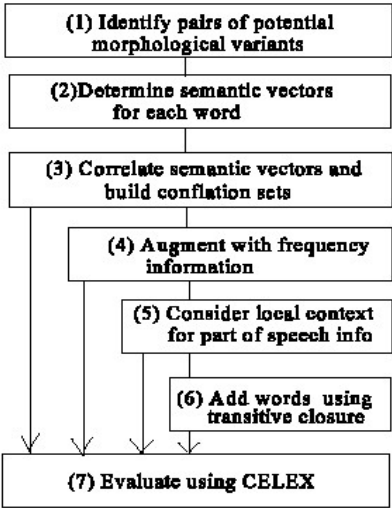


Figure 1: Overview how the algorithm works⁷

⁶ Schone, Patrik and Jurafsky, Daniel: Knowledge-Free Induction of Inflectional Morphologies

⁷ Schone, Patrik and Jurafsky, Daniel: Knowledge-Free Induction of Inflectional Morphologies

3.1.1 Identify pairs of potential morphological variants

The first goal is to find word endings which could serve as suffixes. A useful tool to find these suffixes is the so called character tree. Yet using this approach, there may be circumfixes whose endings will be overlooked in the search for suffixes unless we first remove all candidate prefixes. Therefore a lexicon of all the words in the corpus is built and all word beginnings are identified with frequencies in excess of some threshold (T1), so called pseudo-prefixes. All the pseudo-prefixes are stripped and the word residuals are added back to the lexicon. To show how the search for the suffixes works consider the following example. The following words are contained in the lexicon: align, real, aligns, realign, realigned, react, reacts, and reacted. Due to the high frequency occurrence of “re-“ it is supposed to be a pseudo-prefix. If all the words are stripped of the “re-“ and the residuals are added to a character tree the branch of the tree of words beginning with “a” can be seen in figure 2.

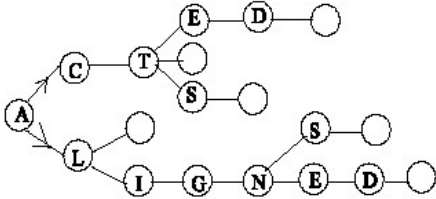


Figure 2: Character tree⁸

Out of the generated character trees rules can be received, but not all of these rules are correct and in the next step, incorporating semantics can help to determine the validity of each rule.

⁸ Schone, Patrik and Jurafsky, Daniel: Knowledge-Free Induction of Inflectional Morphologies

3.1.2 Determine semantic vectors for each word

In order to obtain semantic representations of each word a singular value decomposition SVD is performed to a $N \times 2N$ term-term matrix. The N represents the $N-1$ most-frequent words as well as a glob position to account for all other words not in the top $N-1$. The matrix is structured such that for a given word w 's row, the first N columns denote words that precede w by up to 50 words, and the second N columns represent those words that follow by up to 50 words. Then the SVD is computed and the top 300 singular values to form semantic vectors for each word are kept.⁹

3.1.3 Correlate semantic vectors and build conflation sets

To make a correlation between these semantic vectors normalized cosine scores NCS are used. Out of these scores it is possible to get the probability that an NCS is random or not and it is possible to estimate the distribution of true correlations and number of terms in that distribution. These numbers are needed in the following step.

3.1.4 Augment with frequency information

If just a purely semantic-based approach is used the tendency is to select only the relationships with contextually similar meanings. To overcome this weaknesses of the semantic-based morphology induction the analysis can be improved by supplementing semantic probabilities with orthographic-based probabilities.

The motivation is now to use an approach based on minimum edit distance MED. Minimum edit distance determines the minimum-weighted set of insertions, substitutions and deletions required to transform one word

into another. For example, only a single deletion is needed to transform rates into rate whereas two substitutions and an insertion are required to transform it into rating.

If this method for achieving the task is used the number of correct pairs of potential morphological variants PPMV can be increased by 3% than semantics alone had provided for the $-s$ rule.¹⁰

3.1.5 Consider local context for part of speech info

There is no guarantee that two words which are morphological variants need to share similar semantic properties. Due to this it is possible to improve the performance if the induction process took advantage of local, syntactic contexts around words in addition to the more global, large-window contexts used in semantic processing.

There is an added benefit from following this approach. It can be also be used to find rules that though different, seem to convey similar information. This could be clearly be of use for part-of-speech induction.¹¹

3.1.6 Add words using transitive closure

The algorithm contains semantic, orthographic and syntactic components but there are still valid pairs of potential morphological variants which may seem unrelated due to the corpus choice or weak distributional properties. In Figure 3 this property is demonstrated in greater detail.

By semantics only eight connections can be found starting at Abuse, abuse, abusers, abusing, ...

⁹ Schone, Patrik and Jurafsky, Daniel: Knowledge-Free Induction of Inflectional Morphologies

¹⁰ Schone, Patrik and Jurafsky, Daniel: Knowledge-Free Induction of Inflectional Morphologies

¹¹ Schone, Patrik and Jurafsky, Daniel: Knowledge-Free Induction of Inflectional Morphologies

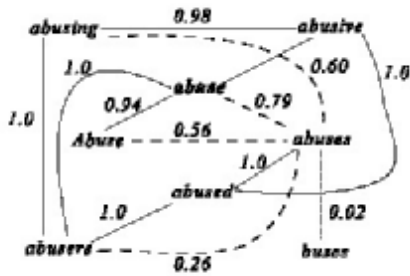


Figure 3: Semantic strengths¹²

References

- John Goldsmith. 2001. *Unsupervised Learning of the Morphology of a Natural Language*, University of Chicago, Association for Computational Linguistics.
- Hervé Déjean, 1998. *Morphemes as Necessary Concept for Structures Discovery from Untagged Corpora*. Université de Caen, Basse Normandie,
- Partick Schone and Daniel Jurafsky. 2001. *Knowledge-Free Induction of Inflectional Morphologies*, University of Colorado at Boulder.

3.1.7 Evaluate using CELEX

The algorithms are only applied to the words out of the corpus which have a frequency higher than 10. This cutoff slightly limits the generality of the results but it also greatly decreases processing time for all of the algorithms tested against it.¹³

4 Conclusion

Using Déjean's algorithm it is very important of which type the corpus is. As it can be seen in Table 2 two completely different results are archived using tow different corpora.

The corpus should be balanced and the result is becoming better the bigger the corpus is.

¹² Schone, Patrik and Jurafsky, Daniel: Knowledge-Free Induction of Inflectional Morphologies

¹³ Schone, Patrik and Jurafsky, Daniel: Knowledge-Free Induction of Inflectional Morphologies

Using salience to rank documents

Patricia Grudziecka, Björn Isaksson

Department of Computer Science

Lund Institute of Technology

Box 118

SE-221 00 Lund, Sweden

d98ng@efd.lth.se, d98bis@efd.lth.se

Abstract

The idea described in this paper is to improve searches on the Internet by using the syntactic structure of sentences. We implemented a method which with help of context factors is able to capture the importance of a word better than a simple occurrence count. The prototype developed is slow and with the functionality limited to simple cases. Further suggestions of improvement of the method are given.

1 Introduction

Often when you search on the Internet you will be frustrated when you can't find the information you want. The search engines will give you popular documents that contain the words you search for. But you don't search for words, you search for content. I.e how the words are used in the document is more important than how often they are used.

The idea is to use the syntactic structure of the sentences in documents to rank the importance of words in a document. E.g. a word that is the subject of a sentence is more important to the content of the document than an object is.

A way to capture that idea is described in this paper.

2 Vector Space Model

A common way to rank documents is to use the Inverse Document Frequency which is based on the

Vector Space Model. The idea of the vector space model is to represent documents and queries in a multi-dimensional space. Semantic equivalence of the query and document is said to be correlated with the proximity of the query and document vectors.

The coordinates or term weights are derived from occurrence counts as described below.

2.1 Term weights

The important question is how to weight words in the vector space model. The essential information used in term weighting is term frequency and document frequency.

The term frequency shows how salient a word is

Quantity	Symbol	Definition
term freq.	$tf_{i,j}$	number of occurrences of w_i in d_j
document freq.	df_i	number of documents in a collection that w_i occurs in

Figure 1: Two commonly used quantities in information retrieval. w_i stands for word i and d_j stands for document j

in a given document. The higher term frequency means a higher likelihood that the word is a good description of the content of the document. The relative importance of a word is often not a linear function of the occurrences of the word, but is taken as a logarithmic function (or another dampening function) of the term frequency. A docu-

ment with three occurrences of a word is more important than a document with one occurrence, but not three times as important.

Document frequency indicates the informativeness of the word. If a word occurs in many documents in the collection its relative importance is less than if it occurs only in a few documents. Therefore one can take the importance of the word as an inverse function of the document frequency. A way to combine a word's term frequency and document frequency into a single weight is as follows:

$$weight(i, j) = \begin{cases} (1 + \ln(tf_{i,j})) \ln \frac{N}{df_i} & \text{if } tf_{i,j} \geq 1 \\ 0 & \text{if } tf_{i,j} = 0 \end{cases}$$

where N is total number of documents in the collection. This form $\ln \frac{N}{df_i}$ is often called *inverse document frequency*.

3 The use of salience to give term weights

In the idea of improving information retrieval, given in the introduction, the syntactic structure of a sentence is the deciding factor of how salient a word is instead of the term frequency. A new number describing the salience of the word instead of the term frequency is used in the calculation of the term weights.

This new number calculated by the weight of the context factors, we call the *aggregated context factor weights* (ACFW). When a word in a sentence is within the scope of the context factor its weight is added to the ACFW of the word. The salience value of an individual word in a document is obtained by adding the weights of the context factors which have that word in their scope:

$$SV(word) = \sum_i weight(CF_i^{word})$$

where SV is the salience value, CF is the context factor (see Figure 3).

After each sentence the word's ACFW is updated by the salience value of that sentence. Figure 2 shows how the ACFW of two individual words changes in the short text.

	cat	dog
The cat and the dog ran.	3+2=5	3+2=5
The dog chased the cat.	5+3=8	5+3+2=10

Figure 2: Example of ACFW calculation.

Context Factors	Objects in scope	Weights
Major-constituent	Subjects and objects	3
Subject	Subject	2
Nested-term	Noun phrase modifiers	1
Relation	Relative clause	3

Figure 3: Context factors and their weights

4 Implementation

A prototype has been developed to rank documents according to the method above. The prototype, called SalRank written in Java, is at this stage not able to rank documents on the Internet, but you have to supply it with text-files. This has been a conscious choice to avoid all the technical pitfalls of the real world and to concentrate on the basic idea. For the same reason there are only rudimentary user interfaces. The program consists of two major parts, the data processing part and the user interfaces.

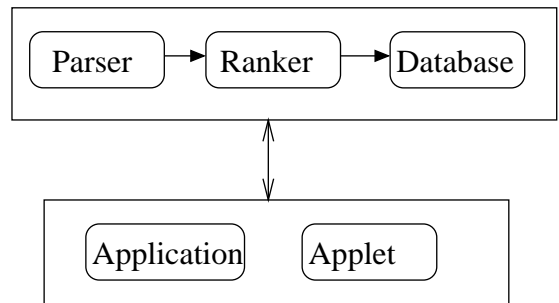


Figure 4: The data processing part above and the user interfaces below.

The data processing part consists of the Parser, Ranker and Database.

4.1 Parser

The parser parses a given text into sentences. At present it can only handle uncomplicated sentences and has trouble with abbreviations. It is

suitable for text-files and it can't read HTML-tags. Thus it is the practical obstacle to run the prototype on the Internet.

4.2 Ranker

In order to rank a document the Ranker has to obtain the context factors in a sentence. This it does with a grammatical parser, Link grammar. Link grammar treats the words of a sentence as blocks with connectors. Every block has connectors pointing to the right or to the left, every connector is of specific type. A left-pointing connector connects with a right-pointing connector of the same type. The two connectors together form a "link". These links are used to decide the context factor of a word (see Figure 5).

Context Factors	Link types
Major-constituent	S, SI, J, O
Subject	S, SI
Nested-term	AN
Relation	R

Figure 5: Link types and corresponding context factors

Figure 6 gives an example of a parsed sentence with link types. The lower-case letters are connector subscripts that are not used in the implementation of SalRank (link type D connects a determiner with a noun).

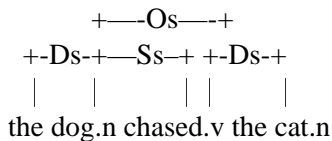


Figure 6: Example of link grammar representation of a sentence.

Depending on the link type of word it a different weight is added to its ACFW. In the example of Figure 6 3+2 (major+subject) is added to the ACFW of "dog" and 3 (major) is added to the ACFW of "cat". The ACFW is the weight of a word in a specific document.

4.3 Database

The Database is implemented as a hashtable with linked lists. Every entry in the hashtable corresponds with one word and contains a linked list. Every link in the list consists of an url and a rank for the specific word in that document.

The rank of every word is computed by the inverse document frequency using the ACFW instead of the term frequency. If a document does not contain a queried word the rank for that word is 0. The document is ranked by summing the ranks of the words according to:

$$r_D = \sum_i \begin{cases} (1 + \ln(a_{D,i})) \ln \frac{N}{df_i} & \text{if } a_{D,i} \geq 1 \\ 0 & \text{if } a_{D,i} = 0 \end{cases}$$

Where r_D is the rank of document D , i runs over the queried words and $a_{D,i}$ is the ACFW for word i in document D . Thus the document with the highest total rank is the best match for the query. It is possible to save and load the Database.

4.4 Application & Applet

The application itself, SalRank, is mainly a user interface. The collection of text documents to be indexed will have to be in a specified directory. The documents are fetched by the application and are then passed on to the data processing part. When the data processing is finished one is able to query the database and get a presentation of the result, in form of a simple list with rank and url for the documents matching the query.

The applet is only able to query a database previously created by the application. The results are presented by the applet in a similar way to the application.

5 Evaluation

The data processing takes extremely long time. In our experiments we have had a collection of 33 files with a total of about 47000 words. It takes several hours to process this collection of documents on a Sparc 167MHz machine. The main bottleneck here is the Link grammar parser. We tested the ACFW against the term frequency using inverse document frequency and an example of the result is presented in Figure 7. In the figure

there is also a comparison with Google.

Ranking results on: house decoration

ACFW	tf	Google*	Filename
13.81	12.01	4	chinadaily.txt
10.94	6.61	2	speel.txt
7.71	8.70	1	burrows.txt
7.07	8.14	3	philamuseum.txt
4.96	9.96	5	cleanairgardening.txt
3.03	3.96	-	house5.txt
1.79	3.19	-	house3.txt

*Only relative order. “house5” and “house3” are not in the top 100 on Google.

chinadaily.txt: An article in China Daily about the mishaps of a customer trying to hire a house decoration company.

speel.txt: An historical review of Lord Leighton’s decoration of Arab Hall, Leighton House.

burrows.txt: Oscar Wilde’s lecture on house decoration given in 1882.

philamuseum.txt: A presentation of House Decoration Themes by the Park House Guides of Philadelphia Museum of Art.

cleanairgardening.txt: A commercial page selling bird- and bat-houses. The houses are “not decoration” according to the text.

house5.txt: Encyclopedia Britannica’s entry on the house sparrow.

house3.txt: Encyclopedia Britannica’s entry on the house mouse.

Figure 7: Test result of ranking with ACFW and term frequency with inverted document frequency.

Our subjective ordering of the documents is entirely consistent with the ordering given by SalRank. Although the first and the two last documents are rated in the same order there are some differences inbetween. The second highest rated document using term frequency, “cleanairgardening”, does not contain any information about “house decoration”. The term “not decoration” increases the rating for this document with term frequency. With the way the term is mostly used in the text e.g. “Bluebird house for bluebirds, not decoration!”, this does not increase the ACFW for “decoration”.

When it comes to the “speel”-text it is rated high

with ACFW mostly because the word “house” in the ‘right’ position in the sentences.

The differences in ratings between documents can intuitively be described as the relative importance of the documents. The difference between the highest and lowest ranking documents is larger with the ACFW method than with the term frequency method implying that the ACFW method is better in lifting important documents and suppressing irrelevant documents. But this is a fairly untested assumption.

6 Conclusions

The idea of SalRank was to improve the perceived correlation between the page content and the search query, on the Internet. We feel we have achieved this, but we would have benefited from a larger test collection to draw this conclusion for certain. At this stage SalRank most often returns a ranking list in the expected order of *our* subjective rating of the documents.

Some improvements of the program can obviously be made. The Parser should be able to fetch real Internet documents and to crawl the net for new documents, like a real search engine. To rank the documents one could exchange the Link grammar parser to a specially written sentence parser to extract the context factors. The database could be replaced by a regular database, like SQL.

A further improvement of our method to calculate the salience could be to take references of the words into account. One’s scope would then no longer be just one sentence at a time, but several on each other following sentences. A word with many references in following sentences would then receive a higher ACFW, than with the present method. The advantage of this method is that even if a word is only used once (or seldom) in a page, but referred to a lot, it is still important to the content of the page.

7 Acknowledgements

In our program we have used java-files written by Ola Åkerberg and Hans Svennson. The most important file was the java interface to Link grammar that is written in C.

We have also used Link grammar from <http://www.link.cs.cmu.edu/link/>.

References

- Christopher D. Manning and Hinrich Schütze, 1999, *Foundations of statistical natural language processing*, MIT Press.
- Carla Huls, Edwin Bos and Wim Claassen, 1995, *Automatic Referent Resolution of Deictic and Anaphoric Expressions*, Association for Computational Linguistics
- Megumi Kameyama, 1997, *Recognizing Referential Links: An Information Extraction Perspective*, <http://acl.ldc.upenn.edu/W/W97/W97-1307.pdf>
- Pierre Nugues, 2002, *Introduction to Language Processing and Computational Linguistics, Lecture notes*, Lund Institute of Technology
- Davy Temperly, 1999, *An Introduction to the Link Grammar Parser*, <http://www.link.cs.cmu.edu/link/dict/introduction.html>
- John Lafferty, 2000, *The Link Parser Application Program Interface (API)*, <http://www.link.cs.cmu.edu/link/api/index.html>

HMS: A Predictive Text Entry Method Using Bigrams

Jon Hasselgren Erik Montnemery Markus Svensson

Lund Institute of Technology
Department of Computer Science

Box 118

S-221 00 Lund, Sweden

{d99jh, d99em, d99msv}@efd.lth.se

Abstract

Due to the emergence of SMS messages, the significance of effective text entry on limited-size keyboards has increased. In this paper, we describe and discuss a new method to enter text more efficiently using a mobile telephone keyboard. This method, which we called HMS, predicts words from a sequence of keystrokes using a dictionary and a function combining bigram frequencies and word length.

We implemented the HMS text entry method on a software-simulated mobile telephone keyboard and we compared it to a widely available commercial system. We trained the language model on a corpus of Swedish news and we evaluated the method. Although the training corpus does not reflect the language used in SMS messages, the results show a decrease by 7 to 13 percent in the number of keystrokes needed to enter a text. These figures are very encouraging even though the implementation can be optimized in several ways. The HMS text entry method can easily be transferred to other languages.

1 Introduction

The entry of text in computer applications has traditionally been carried out using a 102-key keyboard. These keyboards allow to input characters in a completely unambiguous way using single keys or sometimes key combinations.

However, in the last few years, mobile telephones have introduced a new demand for text entry methods. Mobile telephones are usually optimized in size and weight. As a result, the keyboard is reduced to a minimal 12-button keyboard (Figure 1).

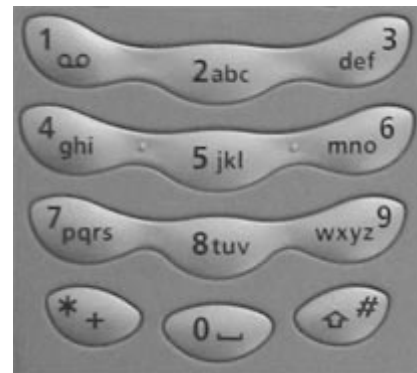


Figure 1: The 12-button keyboard of a Nokia 3410.

The reduced keyboard makes it hard for the user to enter text in an efficient way because s/he has to use multiple tapping or long key combinations to display and disambiguate the characters. Albeit tedious, the multiple tapping method was the most commonly implemented in mobile telephones until some time ago. To spare the user these elements of frustration, a new class of text entry methods has appeared. It uses dictionaries in an attempt to resolve the word ambiguity and requires, in most cases, only one keystroke per character.

This paper proposes a method that supplements the dictionary with word and bigram probabilities. The method uses the last written word to improve the prediction of the current word and to decrease the number of needed keystrokes even further. This method that we refer to as HMS in the

rest of the text, uses the frequencies of common bigrams that we extracted from a corpus of texts.

2 Current Text Entry Methods

In this section, we summarize the text entry methods currently in use and some methods under development. All the mentioned methods use a keyboard with 12 buttons.

As a measurement of the efficiency of the different text entry methods, we will use the number of keystrokes per character or *KSPC* (MacKenzie, 2002). A completely unambiguous keyboard enables a *KSPC* of 1, text prediction methods may reduce this number even further.

2.1 Multi-Press Methods

The multi-press methods require more than one keystroke to enter a character. These methods allow for unambiguous typing of characters. They can be used alone or as a fallback for systems using more complex text entry methods. The multi-press methods are well suited to type words not contained in the dictionary.

2.1.1 The Multi-Tap Method

The first and still most common way to enter text on a mobile telephone is the multi-tap method. Since ‘a’, ‘b’ and ‘c’ share the same key, the user presses it once to enter an ‘a’, twice to enter a ‘b’, and three times to enter a ‘c’. To enter the word *dog*, the user presses the sequence of keys “36664”.

As two consecutive characters of a word can share a same key, as for example the word “no” where both ‘n’ and ‘o’ are assigned to 6, a timeout is needed to determine when to stop shifting the letters and display a new character.

This method results in a *KSPC* of 2.0342 if English text is entered (MacKenzie, 2002).

2.1.2 Remapped Keyboard

On current mobile telephone keyboards, characters are assigned alphabetically to keys. This is not optimal given that, for instance, the most frequent character in English, ‘e’, is displayed using two taps. Remapped keyboards assign a single key to the most frequent characters. The remaining characters are grouped into sets that share

a same key. This method decreases the *KSPC* because frequent characters are entered with only one keystroke.

The program *MessageEase* (Saied, 2001) of EX-ideas uses the idea of the remapped keyboard technique. *MessageEase* results in a *KSPC* at 1.8210 (MacKenzie, 2002).

2.2 Single-Press Methods

The single-press methods try to reduce the *KSPC* to roughly one. They resort to a dictionary as a mean of resolving the ambiguity of the input.

2.2.1 The Predictive Text Entry Method

With the predictive text entry method, the user presses one key per character and the program matches the key sequence to words in a dictionary (Hastrup, 2001). Even if several characters are mapped to the same key, in many cases, only one word is possible given the sequence. This method makes it possible to reduce the *KSPC* to roughly 1. If the key sequence corresponds to two or more words, the user can browse through the resulting word list and choose the word s/he intended to write.

The user, for example, enters the word *come*, by first pressing 2. The program will then propose the word *a* because it matches the entered sequence. When the user presses 6, 6, and 3, the program might propose the words *an*, *con* and finally *come*. The words *bone*, *bond*, and *anod* (and some more), also fit the given sequence. The user can access these words by pressing a next-key.

Many new mobile telephones use this method. The most widely used implementation is T9 by Tegic (Grover et al., 1998). Other implementations are eZiText by Zi Corporation (Zi Corporation, 2002) and iTAP by Motorola (Lexicus Division, 2002). Most implementations only match words with the same length as the key sequence, resulting in a *KSPC* of slightly greater than 1 when the user types words that are contained in the dictionary.

Some implementations propose words longer than the tapped sequence based on probability information for the words. These implementations can reach a *KSPC* < 1.

2.2.2 WordWise

WordWise developed by Eatoni Ergonomics uses an auxiliary key. A character on a key is selected explicitly by simultaneously pressing the key corresponding to the character and the auxiliary key indicating the position of the character on the key. This decreases the number of matching words for a key sequence considerably because the user explicitly disambiguates some characters in the sequence.

A drawback is that two keys must be pressed concurrently. With a limited space keyboard, this can prove difficult to some users.

2.2.3 LetterWise

LetterWise (MacKenzie et al., 2001), also by Eatoni Ergonomics, is a different approach, which eliminates the need for a large dictionary. It only considers the letter digram probabilities. In English, the letter ‘t’ is often followed by ‘h’ and hardly ever by ‘g’. The program selects the most probable letter knowing the previous one. The user can browse and change the characters by pressing a ‘Next’ key.

The LetterWise method has a *KSPC* of 1.1500 (MacKenzie, 2002). One of its main advantages is the small amount of memory needed. Another advantage is the fact that it is just as easy to enter words, which are not in a dictionary. Therefore this could be a suitable fallback method instead of the multi-tap methods, to produce faster text input.

3 Predictive Text Entry Using Bigrams

Prediction may further improve the performance of text entry with a limited keyboard. With it, the suggested words may be longer than the currently typed input.

We propose to use word bigrams, i.e. two consecutive words, to give a better text prediction, see *inter alia* (Shannon, 1948), (Jelinek, 1997), and (Manning and Schütze, 1999). The list of bigrams is stored in memory together with their frequency of occurrence and it is accessed simultaneously with the character input.

Given a previously written word, the most probable subsequent words are extracted from the bigram list. Using the maximum of likelihood, the probability of the bigram w_{n-1}, w_n given the word

w_{n-1} is computed as:

$$P_{MLE}(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n)}{C(w_{n-1})} \quad (1)$$

Since the previously written word w_{n-1} is always known and constant, it is sufficient to use the frequency of the bigrams and set aside $C(w_{n-1})$.

In practice, bigrams must be combined with a dictionary. Sparse data from the development corpus and memory constraints make it impossible to store an exhaustive list of bigrams. To choose the words to propose, we used a variation of the Katz model (Katz, 1987). The Katz model takes the longest available N-gram and uses correction terms to normalize the probabilities. In the case of bigrams, the probabilities can be expressed as:

$$\begin{aligned} P(w_n|w_{n-1}) &= \\ &= \begin{cases} P(w_n|w_{n-1}) & C(w_{n-1}, w_n) \neq 0 \\ \alpha P(w_n) & C(w_{n-1}, w_n) = 0 \end{cases} \quad (2) \end{aligned}$$

where α is the correction term.

In our implementation, the bigrams are always prioritized over the unigrams. The Katz back-off model is well suited for our implementation as it allows for a small memory footprint of the bigrams list, while still ensuring that the system will support entering of all words in the dictionary.

In addition to the bigram frequencies, the word length is a useful criterion to present the matching words to the user. This additional parameter is justified by the navigation through a list of words with the keys available on mobile telephones.

Bigram probabilities used alone produce a list of possible words and rank them without regard to the effort needed to select the intended word. Since browsing the list is carried out using one scrolling key, it may take a couple of keystrokes to reach the word. Even, if corpus frequencies suggest a longer word being preferred to a shorter one, a presentation by decreasing frequencies may be inadequate.

The list navigation is in fact easier in some cases using character input keys. A single keystroke can resolve a great deal of ambiguity because there is a total of 8 keys to choose compared to the unique scrolling key to cycle the list of suggested words.

That’s why the list of proposed words is rescored and short words are given an additional weight.

4 Implementation

We implemented a software prototype of the HMS method we described in this paper. We chose the Java programming language because of its extensive packages that allow for rapid development. Another advantage is Java’s platform independence, which should, in theory, make it possible to run the program on any modern mobile telephone.

The program was designed to run on a handheld device i.e. on the client side of the mobile network. The memory of a mobile telephone is very limited and a disadvantage of this strategy is the memory footprint of the language models we use. A possible workaround would be to implement the HMS software on an application server. All the users would then share the language models with possible customizations. Modern mobile telephone infrastructures enable a real-time round trip of the typed characters and thus the interactive suggestion of matching words.

The program computes a list of word suggestions every time a key is pressed and the best suggestion is displayed simultaneously on the screen: The top white window in our Java program (Figure 2). The user can browse the list of suggestions using the up and down keys.

4.1 Program Design

The program is divided into two parts: a user interaction module and a lexical database module.

The user interaction module currently consists of a Graphical User Interface (GUI) whose layout closely resembles that of a mobile telephone. The simulated keyboard layout makes it possible to compare the HMS prototype with software running on mobile telephones.

The lexical database module contains the core of the program. It is responsible for the generation of a list of suggested words given the user input so far. The modules communicate with each other using an interface. Thus, the two parts are independent and one may modify the user interaction module in particular to fit different platforms



Figure 2: Screenshot of the HMS Java prototype.

without having to modify the module concerning the word guessing algorithm.

4.2 Data Structures

A compact encoding structure of the bigram and unigram lists has a significant impact to achieve an efficient word proposal.

The data structure we used is comparable to that of a letter tree or *trie* (de la Briandais, 1959). However, the nodes of the new tree structure correspond to an input key instead of a character as in the classical tries. For instance, the characters $(a, b, c, 2)$ are associated to a single node. Thus, the tree structure enables to represent the keystroke ambiguity and makes it easier to traverse the tree. It also introduces the need to store a complete list of words that match a keystroke sequence in the leaves resulting in a somewhat higher memory overhead.

Searching this type of tree is straightforward. The keys pressed so far by the user are used as input and the tree is traversed one level down based on every key pressed. When the traversal is completed the resulting sub-tree includes all possible suggested words for the typed key combination.

For the bigrams, a slightly different structure is needed. Since the previously written word has

been chosen from the list of suggested words, it can no longer be considered ambiguous. One cannot simply build a tree of bigrams using the proposed structure because the tree itself is ambiguous. A collection of trees, one tree for each preceding word, was used. For performance reasons, a hash table was used to manage the collection.

4.3 Training the Language Model

We trained the language model – unigrams and bigrams – on the Stockholm-Umeå (SU) Corpus (Ejerhed et al., 1992). The SU corpus is a POS annotated, balanced corpus of Swedish news reports, novels, etc. The SU corpus does not reflect the language of SMS messages that differs greatly from that of the “classical” written Swedish. This results in a non-optimal language model. We chose it because of the unavailability of a large-enough public SMS corpus.

When the input of a single word is completed, its corresponding bigram and unigram probabilities are updated. It results in a learning system, which adapts to every user’s style of writing. To increase the speed of adaptation, language frequencies derived from the user input have higher priorities than what has been learned from the training corpus.

All corpora and dictionaries used with the software have been in Swedish so far. However, the HMS program does not carry out any language-specific parsing or semantic analysis. Hence, the method could be transferred to any language provided that a sufficient corpus exists.

5 Evaluation

As an evaluation of the efficiency of our implementation, we made an initial comparative test between the HMS program and the Nokia 3410, which uses the T9 system.

As we said in the previous section, we could not train a language model optimized for an SMS application. This certainly biased the evaluation of the entry methods in our disfavor. Therefore, we chose to evaluate both programs with a test set consisting of a sample of SMS messages and short texts from newspapers.

A total of nine testers entered the texts. They first had the possibility to get accustomed to both

the HMS and the T9 methods. The testers were encouraged to compose a short arbitrary SMS message of 50-100 characters containing everyday language. They also chose an excerpt of a newspaper article of approximately the same length as the typed SMS message from the *Aftonbladet* Swedish newspaper website. The keystroke count was recorded and used to calculate the *KSPC* parameter.

The entry of new words, i.e. missing from the dictionary, uses the same technique in the HMS and T9 methods. We selected texts where all the words were in the dictionary of both systems.

Table 1 shows the results we obtained in keystrokes per character.

Table 1: Test results.

Method	Type of text	<i>KSPC</i>
T9	SMS	1.0806
HMS Bigrams	SMS	1.0108
T9	News	1.0088
HMS Bigrams	News	0.8807

The HMS entry method shows a *KSPC* smaller than that of the T9 system in both tests: news and SMS texts. The improvement is of, respectively, 7 and 13 percent. The better result for the bigram method is mainly due to two reasons. First, the utilization of the previously written word to predict the next word results in an improvement of the prediction compared to the methods relying only on dictionaries such as T9. Secondly, the fact that words are actually predicted before all characters are entered improves even further the performance of HMS over T9.

6 Discussion

The difference in *KSPC* between the SMS and news text with our method is to a large extent due to the corpus, which does not fit the more casual language of the typical SMS texts. The T9 method, on the other hand, is optimized for typing SMS texts.

Another reason for the difference may be that the news texts in general contain longer words. The mean word length in our test is about 4 characters for the SMS texts and 5 characters

for the news texts. In general longer words can be predicted earlier relatively to the wordlength, since less words are possible given a sequence of keystrokes. This should imply a smaller $KSPC$ for longer words. Figure 3 shows the $KSPC$ according to the word length and the falling curve for longer words.

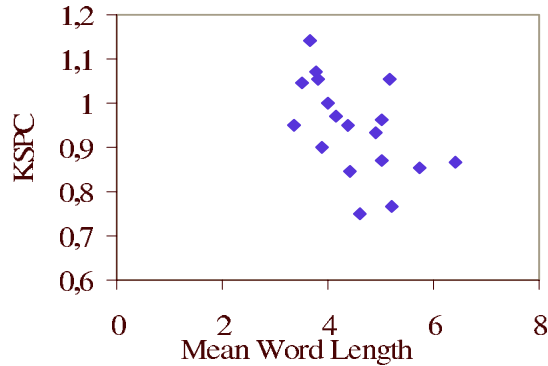


Figure 3: $KSPC$ versus the mean word length in the HMS bigram method.

A longer word often resolves some ambiguities and the possible words for a given key sequence are often fewer than for a short sequence. This explains why the T9 system also shows a better result for the news text. However, the T9 can never reach a $KSPC$ less than 1 since it doesn't predict words longer than the given sequence.

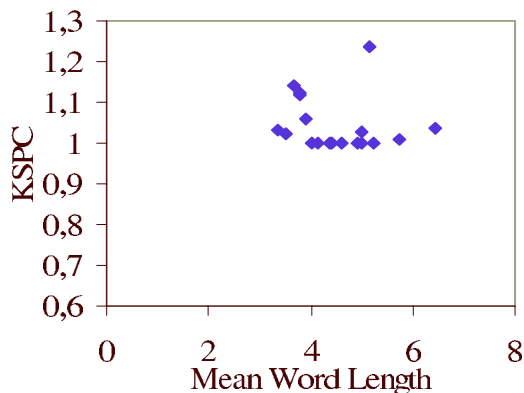


Figure 4: $KSPC$ versus mean word length in the T9 system.

Other significant differences between the SMS and news texts play a role in the final results. For example, the SMS texts show a higher frequency of certain characters such as the question marks, slashes and exclamation marks, which results in a higher $KSPC$. This fact can explain the surprisingly high $KSPC$ for some texts. This property affects both methods to the same extent though.

7 Conclusion and Perspectives

We implemented a new text entry method adapted to mobile telephone keyboards and we compared it to the T9 method widely available on commercial devices. The HMS method is based on language models that we trained on the SU corpus.

The training corpus was, to a great extent, collected from Swedish news wires and didn't fit our application very well. This is heavily related to the language used in SMS messages, which tends to include abbreviations and slang absent from the SU corpus. However, the results we obtained with the HMS method show a decrease by 7 to 13 percent in the number of keystrokes needed to enter a text. These figures are very encouraging even though the implementation can be optimized in several ways.

It would be very interesting to evaluate the $KSPC$ of the bigram method after training the system with a better-suited corpus. We expect the $KSPC$ to be significantly lower than with the present corpus. It is worth once again pointing out that even with the non-optimal corpus, the results of the bigram method are on par or superior.

We also observed that the language model adapts quicker to the users' individual ways of expressing themselves than other systems. It thus increases the gain over time.

At the time we wrote this paper, we could not gain access to a large corpus of SMS messages. However, we intend to collect texts from Internet chat rooms and message boards, where the language shows strong similarities to SMS language. We expect a better language model and an improved $KSPC$ from this new corpus.

A problem with the bigram method is its large memory footprint compared to that of dictionary-based systems. However, this should not be a problem on the next generation of mobile tele-

phones like GPRS and 3G. The language models could be off-loaded on an application server and the low round-trip time of the network system should enable a real-time interaction between the server and the user terminal to carry out the word selection.

References

- Zi Corporation. 2002. eZiText. Technical report, <http://www.zicorp.com>.
- R. de la Briandais. 1959. File searching using variable length keys. In *Proceedings of the Western Joint Computer Conference*, volume 15, pages 285–298, New York. Institute of Radio Engineers.
- Lexicus Division. 2002. iTap. Technical report, Motorola, <http://www.motorola.com/lexicus>, December.
- Eva Ejerhed, Gunnel Källgren, Ola Wennstedt, and Magnus Åström. 1992. The linguistic annotation system of the Stockholm-Umeå project. Technical report, University of Umeå, Department of General Linguistics.
- Dale L. Grover, Martin T. King, and Clifford A. Kushler. 1998. Reduced keyboard disambiguating computer. U.S. Patent no. 5,818,437.
- Jan Haestrup. 2001. Communication terminal having a predictive editor application. U.S. Patent no. 6,223,059.
- Frederick Jelinek. 1997. *Statistical Methods for Speech Recognition*. The MIT Press, Cambridge, Massachusetts.
- Slava M. Katz. 1987. Estimation of probabilities from sparse data for a language model component of a speech recognizer. *IEEE Transaction on Acoustics, Speech, and Signal Processing*, 35(3):400–401, March.
- I. Scott MacKenzie, Hedy Kober, Derek Smith, Terry Jones, and Eugene Skepner. 2001. Letterwise: Prefix-based disambiguation for mobile text input. Technical report, Eatoni, <http://www.eatoni.com/research/lw-mt.pdf>.
- I. Scott MacKenzie. 2002. KSPC (keystrokes per character) as a characteristic of text entry techniques. In *Proceedings of the Fourth International Symposium on Human Computer Interaction with Mobile Devices*, pages 195–210, Heidelberg, Germany. Springer-Verlag.
- Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Language Processing*. MIT Press, Cambridge, Massachusetts.
- Nesbat B Saied. 2001. Fast, full text entry using a physical or virtual 12-button keypad. Technical report, EXideas, <http://www.exideas.com/ME/whitepaper.pdf>.
- Claude E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July-October.

The Artificial Librarian

A Database Dialogue System

Mattias Johansson and Jonas Håkansson
Department of Computer Science
Lund Institute of Technology
Lund University, Sweden

January 17, 2003

1. Introduction

This project is the second part of the course “An Introduction to Language Processing and Computational Linguistics” at the Department of Computer Science. It was defined by us together with our teacher Pierre Nugues. The Artificial Librarian is a dialogue system which is connected to a database. It parses the user input and searches the database for the information. The user can for example ask for a book by a specific author, who has written a specific book or where in the library the book is located. The librarian can sometimes ask for additional information, for example to narrow down the search. The application is web based and can be found at <http://www.efd.lth.se/~e98mj/librarian/>. It may not always be up and running though. There is a screenshot in appendix B to give an idea on how the system looks.

2. Technology overview

The application was decided to be connected to a public book database to make the query constraints as few as possible. Since the application was to be connected to the database through internet it felt natural to make an internet interface for the whole application. Our teacher suggested JSP, Java Server Pages. We had never used this technique, but most of our programming courses are based on Java, so it seemed like a good idea.

3. Language processing

The language processing consists of two parts; one that interprets the user input and one that generates the response from the librarian. The main technique we use for the user input interpretation is decision trees, which simplified works like this: classify the first word, and then take the next word and check if it fits with some of the parts-of-speech that are allowed after the first one. This way, word by word in the sentence is traversed. The dialogue processor, that generates the response from the librarian, is explained in chapter 3.3.

3.1 Parts-of-speech

When a word is to be classified it is compared to the list of words in the part-of-speech table. Examples of the words are shown in the table below.

Part-of-speech	Examples
Auxiliary	do, can, has
Interjection	yes, no, ok, okay
Noun	shelf, books
Preposition	by, on
Pronoun	you, I, anything, something, any
Proper noun	any proper noun, e.g. Stephen King
Verb	have, find, written, wrote
Wh-word	what, where, who, which

3.2 Possible decision trees

The tree consists of part-of-speech phrases. These phrases consist of words of the same part-of-speech. Which phrases or words that are allowed as the next, are shown in the table below. Every sentence begins with the sentence phrase. The first word of a sentence is compared to the list in the second column in the table. If it fits, the next word is checked in the same way. Not all parts-of-speech are allowed as a last word in a sentence. The rules are shown in the third column. Since the phrases not are words, there is no information on those rows. As given from that column the only two parts-of-speech that are allowed as last word are proper nouns and interjections. There is a special case when the last word not is a proper noun or an interjection. When the librarian has asked for something and the user answers with “yes, [phrase]”, the phrase part is interpreted, but if the phrase part doesn’t fit any of the possible types of sentences, the phrase part is neglected and the sentence is interpreted as only “yes”. This means that all words are allowed as a last word if it is a phrase that follows a “yes” or “no” and the phrase couldn’t be interpreted.

Root or current leaf	Possible subsequent leafs	Allowed as last word
Sentence (phrase)	wh-word, preposition phrase, interjection, proper noun phrase, auxiliary	-
Noun phrase	pronoun, noun	-
Preposition phrase	preposition	-
Proper noun phrase	proper noun	-
Verb phrase	verb	-
Auxiliary	noun phrase, verb phrase	no
Interjection	proper noun phrase, preposition phrase, wh-word	yes
Noun	auxiliary, preposition phrase	no
Preposition	proper noun phrase, wh-word	no
Pronoun	verb phrase, preposition phrase, noun phrase	no
Proper noun	proper noun phrase, preposition phrase	yes
Verb	preposition phrase, proper noun phrase, noun phrase	no
Wh-word	auxiliary, noun phrase, verb phrase	no

3.3 Dialogue processor

The user input is searched for special words so that the system knows what the user wants. For example if the user begins the sentence with “who”, the system knows that the user is looking for authors. When a successful interpretation of the user input is done, the query is sent to the database. Otherwise, if it could not be interpreted, the librarian has to ask the user to type something else. When the answer is received from the database, and the result is extracted, the system can decide what the answer should be. The system now checks how many results the database returned and builds different answers based on that information. On some occasions the systems gives more information then the user asked for, e.g. on what shelf the books are located. All possible dialogues are listed in a tree structure in appendix A.

4. Information extraction from web database

The web database we decided to use was Amazon.com. This is an online book store which provides a huge amount of books, so that it will look like there are that many books in the library. The draw back of using a book store is that there are sometimes different editions of a book. If you for example ask “What do you have by Tolkien?”, the answer would be:

I found more than 25 titles. The five most popular, I think, are:

- ✘ *The Hobbit and The Lord of the Rings [BOX SET]*
- ✘ *The Lord of the Rings [BOX SET]*
- ✘ *The Lord of the Rings (Leatherette Collector's Edition)*
- ✘ *The Hobbit (Leatherette Collector's Edition)*
- ✘ *The Silmarillion*

Is it any special book you are interested in?

This is not the answer you expect when it is a library. It should only answer with the title and not with multiple versions of the same book.

4.1 Retrieve result from web page

Amazon.com only provides an html based version of the database available to people outside the company. This means that when we send our database queries we get an html page as an answer. Since Amazon.com is one of the largest online book stores, it has a lot of commercial banners and special offers on the site. This made it a little more difficult to extract the significant results from the answer. If Amazon.com doesn't find any book or it only finds one, the answer page looks completely different, but we managed to find some patterns to recognise the significant results, even in such cases. For example, there was a number on the page that told how many results that was found, but the number only showed if there were two or more results. Another potential problem with getting the result page in html format is that it shows a maximum of only 25 hits per page. As the system is built now that is not a problem, but if we sometime are going to develop it further it might be a limitation. To get the most likely answer when the user asks for an author or a title, we chosen to use Amazon.com's feature of sorting the results by best selling. Then, when the system suggests an author or a title it picks the top item of the list.

5. Development

To give an overview of the way our work proceeded, we have put together some milestones.

- **Selected techniques:** Since we have programmed a lot using Java earlier, we chose to use a Java based environment; Apache Tomcat which is a web server supporting JSP (Java Server Pages). As we run the server on our school account, we can not have it up and running when we are not logged on. We looked around for an online book store to use as the database for our project, and found that Amazon.com would be the best choice, even though it sometimes could be a little slow, because of its large number of books and the consistent way it presents the search results.
- **Extracted search results:** In the beginning we had some difficulties to decode the result pages, because of the large amount of commercials and special offers on the result pages, but managed to solve the problems and wrote code for searching by author, title and both.
- **The Language Processing module:** Initially we wrote code that could handle a couple of types of sentences. Before we decided to use decision trees to represent the allowed sentence structures, we tried a couple of other homemade variants. This was probably the part of the project that chewed the most time.
- **Connected the modules:** We developed the Language Processing module and the Web Database module separately to have better control over eventual misbehaviours. As we got them both to work independently, we connected them and for the first time got a working web application.
- **The dialog history:** To be able to get an overview of the dialog (since each turn can take quite a while to process, see selected techniques above), we added the history to returned web pages.
- **The multi-turn dialogs:** To make a more interesting application, we implemented multi-turn dialogs, so that the user would be able to give answers to the librarian's following-up questions. The recognized answers would be "yes", "no" and "okay". The user may also continue a "yes"-answer with a supported sentence that would be used if correct.
- **Conversation enders:** To reset the dialog and its history, we added conversation enders like "goodbye" and "thank you"
- **Map of library:** We added a simple map as a complement to the referred sections of the library. It makes the visualisation of the artificial library easier.
- **Testing the application:** We let people not involved in the project try The Artificial Librarian to get opinions about it, to see what kind of sentences they asked, and to find eventual bugs.

6. Examples

To give you a picture of how The Artificial Librarian works, we have provided a demonstration with four examples.

Example 1:

U: Who has written Pippi Långstrump?

L: I found over 25 matching titles. Could it perhaps be Astrid Lindgren, Michael Chesworth (Illustrator)? Otherwise, you have to be more specific.

U: Yes, that's right

L: Great, I'm glad I could help you!

Example 2:

U: What books do you have by King?

L: I found more than 25 titles. The five most popular, I think, are:

- ☒ From a Buick 8: A Novel
- ☒ Everything's Eventual: 14 Dark Tales
- ☒ On Writing: A Memoir of the Craft
- ☒ Brunelleschi's Dome: How a Renaissance Genius Reinvented Architecture
- ☒ The Foster's Market Cookbook

Is it any special book you are interested in?

U: Yes

L: Which one would that be?

U: Everything's Eventual

L: It would be to your left in the 'K' section.

Example 3:

U: What do you have by Jan Guillou?

L: I found 18 titles. The most popular ones are:

- ☒ Enemy's Enemy
- ☒ Væagen till Jerusalem
- ☒ Tempelriddaren
- ☒ Irak--det nya Arabien
- ☒ Artister : intervjuer och porträtt

Is it any special book you are interested in?

U: Yes, on which shelf can I find Tempelriddaren?

L: It would be to your left in the 'G' section.

Example 4:

U: Where can I find books by Långstrump?

L: I'm sorry, but it seems like we don't have anything by Långstrump.

U: Okay

L: What else can I do for you?

U: Nothing

7. Conclusions

The problem discussed in chapter 4, that the system answers with different versions of the same book, should be dealt with. If we some day would decide to develop The Artificial Librarian further, we would probably use a library database to make it more realistic. Another solution is that we could change the library to an information desk in a book store. This would be much easier of course, but it could have other features as price information and price limitation in a search.

When we started with this project we did not really know what result we could expect because we were all new to computational linguistics. We started with a not very high ambition level, but our teacher encouraged us to make it more and more extensive. Now when we are finished, the most important lesson we have learned is that it doesn't take a huge program to make an interactive dialogue system which work fairly well, although there are some linguistic tricks that make our system look smarter then it is.

Overall we are satisfied with the result of this project. Of course there are things that could be better, but it was only a two university points (two weeks) project.

8. References

J Boye, M Wirén, M Rayner, I Lewin, D Carter and R Becket, July 1999, **Language-Processing Strategies and Mixed-Initiative Dialogues**, IJCAI-99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems

S Abney, M Collins and A Singhal, 2000, **Answer Extraction**, AT&T Shannon Laboratory

P Nugues, 2002, **Lecture notes: Introduction to Language Processing and Computational Linguistics**, Department of Computer Science, Lund Institute of Technology

Apache Tomcat of The Apache Jakarta Project, <http://jakarta.apache.org/tomcat/>

Amazon.com, <http://www.amazon.com/>

Appendix A. Structure of implemented dialogs

This appendix shows the structure of all implemented types of possible and successful dialogs. A "new" dialog (beginning with one of the three main types below) can be started at any point in the current dialog.

Upper case words within brackets indicate information supplied by the user or other dynamic information generated by the program. Below follows a list of descriptions of such words.

[TITLE]	Title supplied by user or found during search
[AUTHOR]	Author supplied by user or found during search
[SECTION]	First letter in author's last name indicating a section
[HITS]	Number of hits found during search
[DIRECTION]	Side of the artificial library (left or right)
[LOCATION_SENTENCE]	A possible sentence of the third main type of dialog starters below
[ANYTHING]	Any sentence/text

- Asking for a title (e.g. "What do you have by Stephen King?")

Search result returns no titles:

- I'm sorry, but it seems like we have no books by [AUTHOR].
 - Okay
 - What else can I do for you?

Search result returns just one title:

- I can only find [TITLE] by [AUTHOR]. Is this the one you are looking for?
 - Yes
 - You will find it in the '[SECTION]' section.
 - Okay
 - I am glad I could help you!
 - No **OR** No, [ANYTHING]
 - I'm sorry but it seems like we don't have any other books by [AUTHOR].
 - Okay
 - What else can I do for you?

Search result returns more than one but fewer than six titles:

- I found [HITS] titles:
[TITLE] ... [TITLE]
Is it any special book you are interested in?
 - Yes
 - Which one would that be?
 - [TITLE]
 - It would be to your [DIRECTION] in the '[SECTION]' section.
 - Okay
 - I am glad I could help you!
 - Yes, [TITLE] **OR** Yes, [LOCATION_SENTENCE]
 - It would be to your [DIRECTION] in the '[SECTION]' section.
 - Okay
 - I am glad I could help you!
 - No **OR** No, [ANYTHING]
 - Okay, I hope you find them interesting!

Search result returns more than five but fewer than twenty six titles:

- I found [HITS] titles. The most popular ones are:
[TITLE] ... [TITLE]
Is it any special book you are interested in?
 - Yes
 - Which one would that be?
 - [TITLE]
 - It would be to your [DIRECTION] in the '[SECTION]'
 - Okay
 - I am glad I could help you!
 - Yes, [TITLE] **OR** Yes, [LOCATION_SENTENCE]
 - It would be to your [DIRECTION] in the '[SECTION]'
 - Okay
 - I am glad I could help you!
- No **OR** No, [ANYTHING]
 - Okay, I hope you find them interesting!

Search result returns more than twenty five titles:

- I found more than 25 titles. The five most popular, I think, are:
[TITLE] ... [TITLE]
Is it any special book you are interested in?
 - Yes
 - Which one would that be?
 - [TITLE]
 - It would be to your [DIRECTION] in the '[SECTION]'
 - Okay
 - I am glad I could help you!
 - Yes, [TITLE] **OR** Yes, [LOCATION_SENTENCE]
 - It would be to your [DIRECTION] in the '[SECTION]'
 - Okay
 - I am glad I could help you!
- No **OR** No, [ANYTHING]
 - Okay, I hope you find them interesting!

• Asking for an author (e.g. "Who has written The Hitchhiker's Guide To The Galaxy?")

Search result returns no titles:

- I'm sorry, but I don't know who the author is of [TITLE].
 - Okay
 - What else can I do for you?

Search result returns one or more but fewer than twenty six titles:

- I found [HITS] matching titles. Could it perhaps be [AUTHOR]?
Otherwise, you have to be more specific.
 - Yes
 - Great, I'm glad I could help you!
 - No **OR** No, [ANYTHING]
 - You have to be more specific.
Otherwise, that is the only author I can find on [TITLE].
 - Okay
 - What else can I do for you?

Search result returns more than twenty five titles:

- I found over 25 matching titles. Could it perhaps be [AUTHOR]?
Otherwise, you have to be more specific.
 - Yes
 - Great, I'm glad I could help you!
 - No **OR** No, [ANYTHING]
 - You have to be more specific.
Otherwise, that is the only author I can find on [TITLE].
 - Okay
 - What else can I do for you?

• Asking for a location (e.g. "On which shelf can I find The Testament by John Grisham?")

Search result returns no titles:

User supplied both the title and the author in the question:

- I'm sorry, but it seems like we do not have [TITLE] by [AUTHOR].
- Okay
 - What else can I do for you?

User only supplied the title in the question:

- I'm sorry, but I can not find any books named [TITLE].
- Okay
 - What else can I do for you?

User only supplied the author in the question:

- I'm sorry, but it seems like we don't have anything by [AUTHOR].
- Okay
 - What else can I do for you?

Search result returns one or more titles:

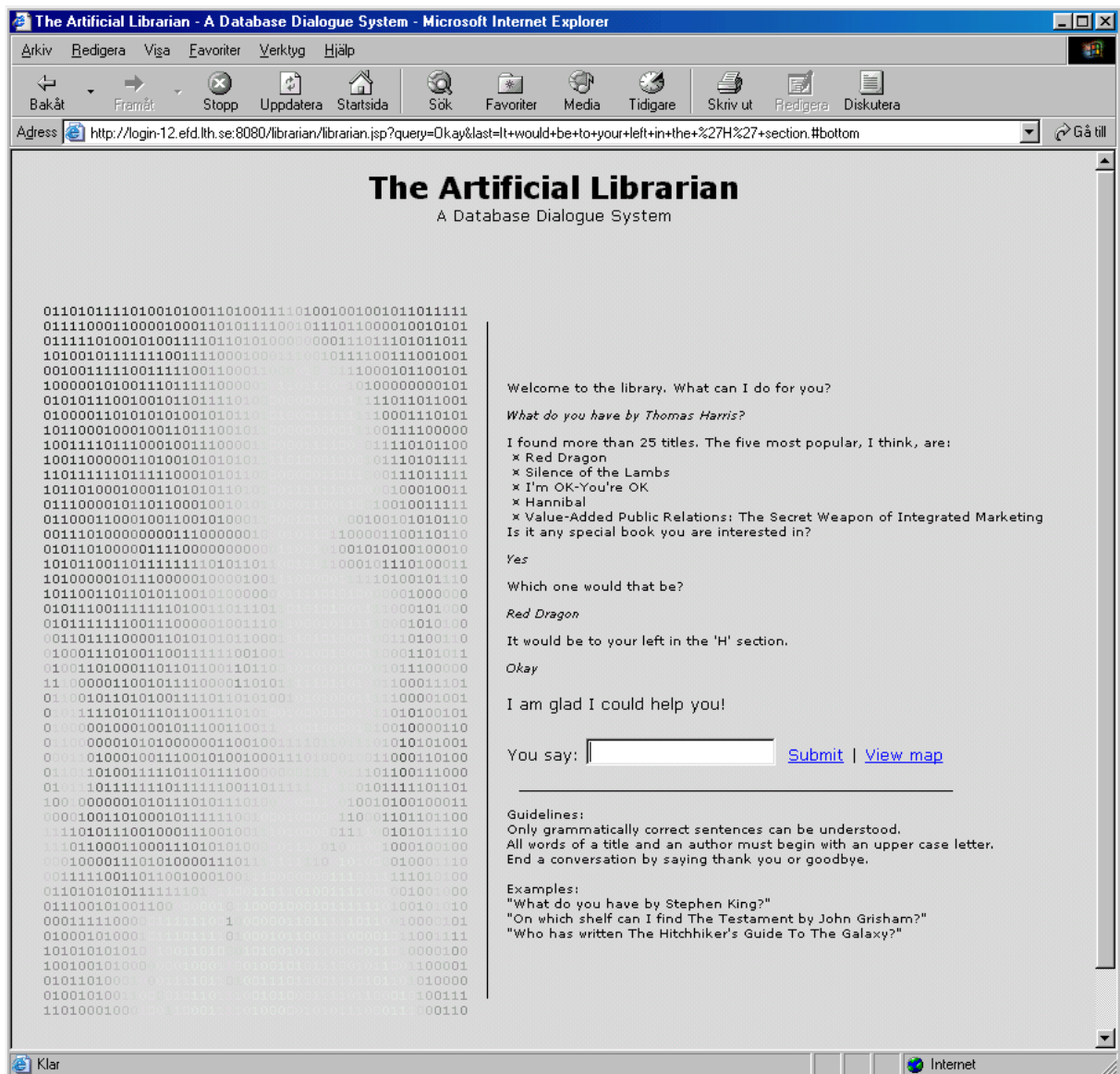
User supplied both the title and the author or just the author in the question:

- It would be to your [DIRECTION] in the '[SECTION]' section.
- Okay
 - I am glad I could help you!

User only supplied the title in the question:

- If you mean [TITLE] by [AUTHOR], it's to your [DIRECTION] in the '[SECTION]' section. Otherwise, you have to be more specific.
- Okay
 - I am glad I could help you!

Appendix B. Screenshot



A Morphological Parser for Estonian

Richard Johansson
d99rj@efd.lth.se

Abstract

This report describes how a prototype morphological parser for Estonian, a language with a relatively complex morphology, was implemented. The implementation uses a two-level model which was hand-coded in Prolog. Some background material on the Estonian language and the two-level model is presented.

1 Introduction

A morphological parser is a computer tool that accepts a given inflected word (or sequence of words) and determines which its root word is and which inflections it has undergone. It can be used alone, as a kind of intelligent dictionary for the learner of a language, or as a morphological backend for a high-level natural language tool such as a dialogue system.

Morphology has been a neglected area of research in computational linguistics, mainly due to the fact that English has a trivial morphology. A simple morphology eliminates the need for morphological parsers since the inflectional forms are few enough to be stored in the dictionary along with their root words. But in a language with a complex morphology, where a noun or verb can take hundreds of shapes, an automatic tool is needed to extract the root word and interpret the inflectional information.

This report describes the implementation of a morphological parser for Estonian, a Finno-Ugric language with a relatively complex morphology. Although quite incomplete, the morphological parser is currently running for everyone to try at its web interface at <http://www.df.lth.se/~richardj/parser>.

The report is organized as follows: In Section 2 a short introduction to the Estonian language is given. The implementation of the parser is described in Section 3. In the following section, Section 4, the parser is evaluated and compared to existing tools on the web, and in Section 5 we discuss why the result is imperfect. Finally, a conclusion is given in Section 6.

Linguistic terms are for the sake of brevity only explained only when they are specific to Estonian. When using linguistic terminology, I have tried to stick to the Glossary of Linguistic Terms by SIL (<http://www.sil.org/linguistics/GlossaryOfLinguisticTerms/>).

2 An Introduction to the Estonian Language

2.1 Background

Estonian is spoken by approximately 1100000 people in the world. Most of them live in Estonia, located on the east side of the Baltic Sea in the north of Europe, but there is also a large diaspora. The language is a Finno-Ugric language closely related to Finnish and some nearly extinct languages in the St. Petersburg area in Russia.

Some characteristics of the Finno-Ugric languages compared to the Indo-European are:

- absence of gender (the same pronoun for both he and she),
- absence of definite-indefinite articles (a and the in English),
- long words due to the structure of the language,
- numerous noun cases,

- postpositions rather than prepositions, and
- no syntactic equivalent of the verb “to have”.

Estonian has been more influenced by Indo-European languages (above all German) than Finnish has been, which is visible not only in the numerous loan words but also in the German-influenced word order and the use of genitive forms instead of possessive suffixes. Another striking feature of Finnish, the vowel harmony, has also disappeared in Estonian.

When the nationalist movement in Estonia emerged, and Estonian needed to be remade from a peasant’s language into a language for all levels of society, the language was artificially reconstructed grammatically and many new words were invented to purge the language from unwanted loan words and to cover concepts for which Estonian earlier had no suitable words.

A readable and thorough Estonian language course (written in Swedish) is [8]. A linguistic overview can be found in [1]. The dictionary data and inflection tables used when implementing the morphological parser were taken from *Õigekeelsussõnaraamat* [2] (from now on called the *ÕS*), the standard reference dictionary used in Estonia. The best Estonian-English dictionary is [7].

2.2 Letters and Sounds

Estonian uses the Latin alphabet with the following non-ASCII additions: *ä, ö, ü, õ, š* and *ž*. The letters *c, f, g, w, x, y, z, š* and *ž* are only used in words of foreign origin. In computer systems various encodings are used. This morphological parser, like most other computer systems using Estonian, uses the ISO-8859-1 encoding.

One interesting feature of Estonian phonology is that there are three degrees of quantity (length) of vowels and consonants. To indicate the first degree of quantity, a single letter is used. Unfortunately, the distinction between the second and third degree is not always indicated in the spelling. Here are two examples where this leads to ambiguity:

<i>saada</i>	(second)	send!	<i>linna</i>	(second)	of the town
<i>saada</i>	(third)	to get	<i>linna</i>	(third)	into town

Beside quantity, there is another source of ambiguity in Estonian spelling: palatalization of consonants. This means that dental consonants are pronounced with the tongue in the position of an *i*-sound, slightly similar to the sound *ñ* in Spanish. The letters *d, t, l, s* and *n* can all denote an unpalatalized as well as a palatalized sound.

Except for the quantity and the palatalization, Estonian spelling is simple. One letter generally stands for one sound. A single letter denotes a short sound and a doubled a long (second or third quantity).

The pronunciation of letters in Estonian is usually similar to the pronunciation in German, its greatest influence. There are although some differences: *b, d* and *g* are distinguished from *p, t* and *k* mainly in quantity (the former are pronounced with the first degree of quantity and the latter with the second) rather than voicing. The letter *v* is always pronounced voiced and *s* always voiceless. The unlauded letters *ö* and *ü* are pronounced like their German counterparts, but *ä* is open like the *a* in the English word *fan*. The letter *õ* is an indescribable sound specific to Estonian, but it can be approximated by the *u* in the Swedish word *Mumintrollen* with a deep Finland-Swedish pronunciation. The letters *z, ž* and *š*, which are used in loan words only, denote sounds similar to the English *zoo*, *measure* and *shoe*, respectively.

2.3 General Picture of Morphology

Originally Estonian, like its neighbour language Finnish still is, was a highly agglutinative language where separately identifiable and independent morphemes were added to the word stem. But due to influence from Indo-European languages and phonological attrition, morphemes in word-endings are not always separable in present-day Estonian. For example, the present tense person and number endings are today different from the past tense endings. But Estonian still has some agglutinative features, like most of the noun case endings (which are simply added to the genitive endings) and the clitic *-gi/-ki*.

Central to the morphology of Estonian is the concept of gradation. This is a process where a word changes phonetically, usually in the form of a syllable shortening or loss of a stop consonant, when the word is put into certain inflections. For example, the verb *õppima*, “to study”, (where *pp* signifies a third degree stop) becomes *õpin* (with a second degree stop) in the present tense first person singular, and the noun *lind*, “bird” (with a short stop at the end) becomes *linnu* in the genitive singular (the stop is lost).

2.4 Morphology of Verbs

The verb morphology is more complex in Estonian than in English but comparable in structure and complexity to Romanic languages like French or Spanish. The finite forms of the verbs are constructed according to the following morphological features:

- *Mood*: Indicative, conditional, imperative or oblique (reported speech),
- *Voice*: Active or passive,
- *Tense*: Present or past (perfect and pluperfect are constructed with participle forms),
- *Polarity*: Affirmative or negative,
- *Person*: First, second or third,
- *Number*: Singular or plural.

As one can see, the Estonian verb features are roughly the same as in many Indo-European languages, with the main difference being that the verb has an affirmative – negative polarity feature. In the negative polarity, the verbs lose their person and number endings (except for in the imperative mood). The oblique mood, which tells what one is said to or supposed to do, is also notable.

There are also nonfinite forms such as participles, a gerund and two kinds of infinitive. The dictionary form for verbs is the first infinitive.

In Appendix A we can see an example verb *minema*, which is the most irregular verb of the Estonian language, conjugated in most forms.

2.5 Morphology of Nouns and Adjectives

While the verb morphology of Estonian is relatively similar to those of some Indo-European languages, the morphology of nouns is radically different. As mentioned above, Estonian has a wealth of noun cases. From a morphological point of view, the cases can roughly be divided as follows:

- *Fundamental*: Nominative (denoting the subject of the sentence), genitive (denoting possessor or the object of the sentence) and partitive (denoting a quantity or the object of the sentence),
- *Nonfundamental*: Illative (into what?), inessive (in what?), elative (from the inside of what?), allative (onto what?), adessive (on what?), ablative (from the outside of what?), translative (into the state of what?), terminative (until what?), essive (as what?), abessive (without what?) and comitative (with what?).

The nonfundamental cases are usually formed by adding their respective ending to the genitive form. For example, if we wish to construct the inessive plural we add the ending *-s* to the genitive plural form.

Adjectives in Estonian are from a morphological point of view just nouns which can form the comparative or superlative forms. The dictionary form for nouns and adjectives is the nominative singular. The nouns and adjectives are declined in singular and plural. Many alternate forms are possible.

In Appendix B we can see the nouns *raamat* and *mägi* in all inflectional forms. For *raamat*, alternate plural forms are available for many cases – the case ending may be added both to the genitive plural and to the partitive plural (which is considered somewhat archaic). For the other example noun, *mägi*, we see that a short form of the illative case is possible. If the short form is available, it is generally preferred.

Nouns in the nominative singular or genitive singular or plural may form a compound with another noun.

2.6 Syntax

As mentioned above, Estonian word order is influenced by the German one. But that is more a question of preferred style than a rigid rule, and generally speaking the word order is free for the speaker to vary according to taste and style or for emphasis or clarity.

One of the fundamental concepts in Estonian syntax, which is also notable from a morphological point of view, is the distinction between total and partial objects, which roughly means that the object was affected “totally” or “partially”. This manifests itself in the use of the genitive or nominative case for the total object and partitive case for the partial object. For example:

Ma lugesein raamatut. I read (or was reading) a book.
Ma lugesein raamatu läbi. I read the book through.

Here *raamat*, “book”, is in the partitive case in the first example and in the genitive case in the second. The word *läbi*, “through”, is one of a number of particles used to emphasize that the action is complete. These particles have come into use relatively recently and the reason they are used is that for many words in present-day Estonian, the genitive and the partitive forms coincide. For example:

Koer söi vana muna. The dog ate (or was eating) an old egg.
Koer söi vana muna üra. The dog ate the old egg (up).

Here *vana*, “old”, and *muna*, “egg”, have identical genitive and partitive forms.

3 Implementation

3.1 Goals

The following goals of the parser project were set:

- The parser should be able to parse all inflected forms of nouns, pronouns, adjectives and verbs,
- it should be reasonably fast,
- it should be able to handle all possible compounds, not just the most frequent ones,
- it should be able to handle the most regularly appearing derivations.

3.2 Two-level Morphology

The model used in this implementation is the two-level model introduced by Kimmo Koskenniemi, which presently is the most widespread one for implementations of morphology systems. This model is useful because it is conceptually very simple and because it easily can be implemented efficiently using finite-state techniques. In [4], an introduction to the subject is given by its original inventor.

In the two-level model, a word is thought to have a *surface form* – what appears to the speaker – and an *underlying form* – the theoretical form consisting of root word and inflection information. The relations between these two forms are called *two-level rules*. The two-level rules are bidirectional – one can go from the surface form to the underlying form or vice versa.

The two-level rules usually operate symbol by symbol. To handle the case where a symbol is present in the underlying form but not in the surface form (or vice versa), a *zero symbol* is inserted.

Here is an example of a two-level rule for a group of Estonian nouns. For these nouns, which consist of two vowels separated by *b*, *d* or *g*, the surface genitive singular form is formed from the underlying nominative form by removing the separating consonant and mutating the vowels in the following way: *i* becomes *e*, *u* becomes *o* and *ü* becomes *õ*. Here we can see how a two-level rule could relate the surface form *sea* (“of the pig”) to the underlying form *siga* (“pig”) Singular Genitive.

Surface form	s e 0 a 0 0
Underlying form	s i g a S G

3.3 Finite-state Transducers

The two-level rules are usually implemented using finite-state transducers, which are nondeterministic finite-state automata where the arcs are labelled with pairs of surface-underlying symbols.

In [6] and [5], there are introductions to the theory of finite-state transducers, with theorems about their basic properties presented.

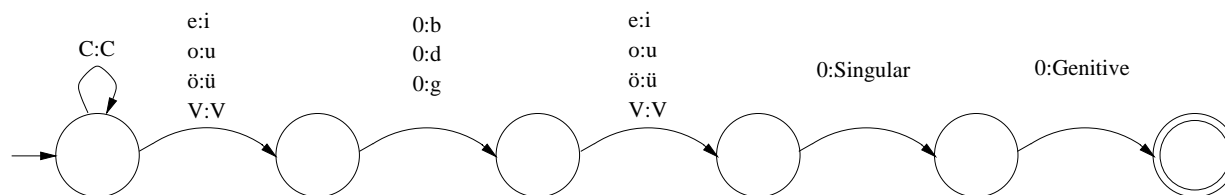


Figure 1: A finite-state transducer

In figure 1 we see a transducer which implements the two-level rule for forming the genitive for the noun group mentioned in Subsection 3.2. Here, C means any consonant and (for the sake of brevity) V means any vowel except *i*, *u* or *ü*.

To make a transduction from the surface string *sea* to the underlying string *si* Singular Genitive, we first follow the arc from the start state back to itself, while consuming the *s* in both the surface and the underlying form. Then we jump to the next state, this time consuming the *e* in the surface form and the *i* in the underlying form. We follow arcs and consume symbol pairs in this way until there are no more symbols left, and if we are then in a final state (marked in the figure as a double circle), the transduction was successful.

Note that in this transducer, the transduction from surface form to underlying form is not a function, that is the underlying form is not uniquely determined by the surface form. In this example five additional underlying forms, for example the nonexistent **seba* Singular Genitive, are produced by the transducer. When trying to find correct underlying forms, the results need to be checked with a dictionary.

3.4 Implementation of Transducers

The transducers for this morphological parser were hand-written in Prolog. Using Prolog is particularly convenient from a programming point of view: the nondeterminism of the transducers and their bidirectional nature can both be implemented almost without effort.

In Appendix C a Prolog code example implementing the described transducer is given.

The reason that no existing two-level morphology system was used instead of hand-writing was mainly the time constraints – there was simply no time to learn how to use for example the Kimmo system [3].

3.5 The Dictionary

As mentioned above, the transducers produce lots of hypothetical forms which usually are incorrect. This means that for every word, many dictionary lookup need to be made. An efficient dictionary data structure is therefore essential.

The dictionary was implemented with a *letter tree* or *trie*. In a letter tree the words are stored as paths in a tree, with the nodes representing letters. In the leaf nodes, grammatical information such as conjugation pattern numbers could be stored. Two words which begin with the same letters share a common entry path. This leads to a compact dictionary data structure, where the search time is proportional to the number of letters of the word. In figure 2 a letter tree with two words is shown.

The dictionary used for the morphological parser is the online version of the $\tilde{O}S$. Since the transducers should be able to construct any possible compound word, those compound words which were already present in the dictionary could be removed. This reduced the size of the dictionary from about 120000 words to 50087

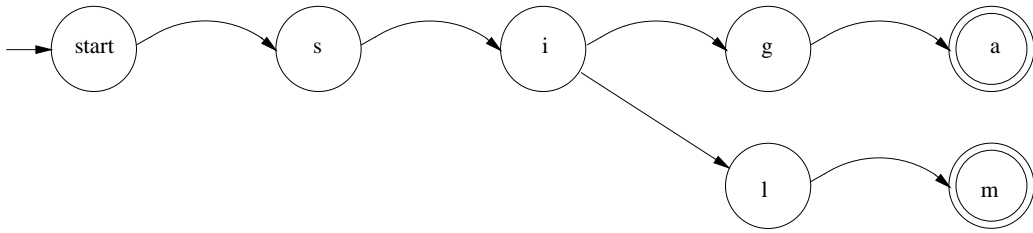


Figure 2: A letter tree

words.

3.6 Other Implementation Details

The parser system consists of a client containing the transducer code and a server handling the dictionary. This division was made for development reasons since the startup of the dictionary takes several minutes, and while developing and debugging the transducers the program frequently needs to be restarted.

The executables are built as compiled Prolog code and currently run on an UltraSPARC-II 300 MHz machine.

4 Evaluation

To evaluate the quality of the implementation, the parser was compared to two existing parsers on the web: Filosoft's at <http://www.filosoft.ee> and the one at Estonian Language Institute, <http://kiisu.eki.ee>. The prototype was run once with compound words enabled and once with compound words disabled. The 100-word test data used can be seen in Appendix D. The results were the following:

- When compound word handling was enabled the parser produced a result after 1 minute and 40 seconds with 69 percent correct parses.
- When compound word handling was disabled the result was produced in 20 seconds with 63 percent correct parses.
- The parsers at Filosoft and at Estonian Language Institute both parse the test data in a couple of seconds.

The benefit of compound words is questionable since it may lead to an explosion of possible constituent words of the compound. For example, when the word *patsiendiorganisatsiooniks* (“patient organization” in the translative case, which is just a two-word compound) was input, the parser produced no output at all (at least the report author did not have the time to wait for it).

5 Improvements

5.1 Program Speed

The main problem with the parser is, as we have seen in the previous section, its speed. There are a number of reasons for this:

The client-server division of the program should not have been done between the transducers and the dictionary, since all the dictionary lookups will lead to excessive process communication. This is currently not a big issue since the processes run on the same machine (and time measurements have shown that communication is only a small part of the execution time), but making another division could lead to a more efficient program. But as mentioned previously, the reason for this division was the development cycle time and this should only be used as a last resort.

A less attractive implementation improvement could be to abandon Prolog as implementation program language and fine-tune the implementation in a low-level language such as C. The current implementation uses Prolog features such as atom unification and backtracking extensively. Abandoning Prolog would however surely increase the development time drastically.

The parser behaves acceptably when compound word handling is disabled, but when compound words are handled the output may be produced after many minutes. The naïve implementation of compound word handling used here (just connecting nominative and genitive final states to the start of the noun transducer) is obviously not a good strategy. Limiting the number of allowed constituents is a strategy taken in the implementation of the two parsers which were compared to the prototype – they both allow a maximum of four nouns to be compounded. Memoization is obviously also a necessity when handling compound words. Another simple option is to disallow arbitrary compound words and just use the ones given in the $\tilde{O}S$.

5.2 Maintainability

Another obvious problem seen when inspecting the code is that it is unreadable, which is the result of the decision to hand-write the transducers in Prolog. This leads to code that is difficult to change and debug. It is obvious that such code could be generated by an automatic tool (for example a graph-drawing program), which could be produced in relatively little time. Another option could be to use existing rule-based two-level morphology systems such as Kimmo [3].

6 Conclusion

6.1 Goals Met

A prototype of the parser was implemented and set up to run with a web user interface. The verb morphology was completely implemented (except for some nonstandard forms), but due to time constraints the noun morphology was only partially implemented. The most common personal pronouns are handled, but no adjective features (comparative and superlative). Derivations are not handled except for the verbal noun forms (like “the eating”). The clitic *-gi/-ki* is not handled. The success rate of the prototype the test data in Appendix D was 63 percent when compound words were disabled.

As we saw in the Section 4, the parser prototype is acceptably fast when compound words are disabled but still much slower than the other ones tested. Compound words can be handled but expose a severe implementation problem, as the number of possible constituent words of the compound becomes enormous for long words.

6.2 Final Words

The parser project has been relatively successful and its goals have to a certain degree been met. But to overcome the speed and maintainability problems and produce a high-quality product such as the two existing parsers compared to the prototype, much work would need to be done.

A A Verb Example: *minema*, “to go”

A.1 Finite forms

<i>Mood</i>	<i>Voice</i>	<i>Tense</i>	<i>Affirmative polarity</i>	<i>Negative polarity</i>
			1st Person Singular, ..., 3rd Person Plural	
Indicative	Active	Present	lähen, lähed, läheb, läheme, lähete, lähevad	lähe
		Past	läksin, läksid, läks, läksime, läksite, läksid	läinud
	Passive	Present	minnakse	mindu
		Past	mindu	mindud
Conditional	Active	Present	läheksin, läheksid, läheks, läheksime, läheksite, läheksid	läheks
		Past	läinuksin, läinuksid, läinuks, läinuksime, läinuksite, läinuksid	läinuks
Imperative			mingu, mine, mingu, mingem, minge, mingu	mingu, mine, ...
Oblique	Active	Present	minevat	minevat
	Passive	Present	mindavat	mindavat

A.2 Nonfinite forms

<i>First infinitive</i>	minema
<i>Second infinitive</i>	minna
<i>Progressive form</i>	minev
<i>Gerund</i>	minnes
<i>Active past participle</i>	läinud
<i>Passive past participle</i>	mindud

B Two Noun Examples

B.1 *raamat*, “book”

	<i>Singular</i>	<i>Plural</i>	
<i>Nominative</i>	raamat	raamatud	
<i>Genitive</i>	raamatu	raamatute	
<i>Partitive</i>	raamatut	raamatuid	
<i>Illative</i>	raamatusse	raamatutesse	raamatuisse
<i>Inessive</i>	raamatus	raamatutes	raamatuis
<i>Elicative</i>	raamatust	raamatutest	raamatuist
<i>Allative</i>	raamatule	raamatutele	raamatuile
<i>Adessive</i>	raamatul	raamatutel	raamatuil
<i>Ablative</i>	raamatult	raamatutelt	raamatuilt
<i>Translative</i>	raamatuks	raamatuteks	raamatuiks
<i>Terminative</i>	raamatuni	raamatuteni	raamatuini
<i>Essive</i>	raamatuna	raamatutena	raamatuina
<i>Abessive</i>	raamatuta	raamatuteta	
<i>Comitative</i>	raamatuga	raamatutega	

B.2 *mägi*, “mountain”

	<i>Singular</i>	<i>Plural</i>	
<i>Nominative</i>	mägi	mäed	
<i>Genitive</i>	mäe	mägede	
<i>Partitive</i>	mäge	mägesid	
<i>Illative</i>	mäesse	mäkke	mägede
<i>Inessive</i>	mäes	mägedes	
<i>Elative</i>	mäest	mägedest	
<i>Allative</i>	mälele	mägedele	
<i>Adessive</i>	mäel	mägedel	
<i>Ablative</i>	mäelt	mägedelt	
<i>Translative</i>	mäeks	mägedeks	
<i>Terminative</i>	mäeni	mägedeni	
<i>Essive</i>	mäena	mägedena	
<i>Abessive</i>	mäeta	mägedeta	
<i>Comitative</i>	mäega	mägedega	

C Prolog Code Example

```

arc(start, start, C, C) :- consonant(C).

arc(start, state1, i, e).
arc(start, state1, u, o).
arc(start, state1, ü, ö).
arc(start, state1, V, V) :- member(V, [a, e, o, y, ä, ö, õ]).

arc(state1, state2, b, 0).
arc(state1, state2, d, 0).
arc(state1, state2, g, 0).

arc(state2, state3, i, e).
arc(state2, state3, u, o).
arc(state2, state3, ü, ö).
arc(state2, state3, V, V) :- member(V, [a, e, o, y, ä, ö, õ]).

arc(state3, state4, 'Singular', 0).
arc(state4, final, 'Genitive', 0).

final_state(final).

transduce(Start, Final, [U | UnderlyingString], SurfaceString) :-
    arc(Start, Next, U, 0),
    U \== 0,
    transduce(Next, Final, UnderlyingString, SurfaceString).
transduce(Start, Final, UnderlyingString, [S | SurfaceString]) :-
    arc(Start, Next, 0, S),
    S \== 0,
    transduce(Next, Final, UnderlyingString, SurfaceString).
transduce(Start, Final, Under, Surface) :-
    arc(Start, Next, 0, 0),
    transduce(Next, Final, Under, Surface).
transduce(Start, Final, [U | UnderlyingString], [S | SurfaceString]) :-
    arc(Start, Next, U, S),
    U \== 0,
    S \== 0,
    transduce(Next, Final, UnderlyingString, SurfaceString).

```

```
transduce(Final, Final, [], []) :-  
    final_state(Final).
```

D Test Data

The following test data was used in the performance tests. The text consists of 100 words and was taken from the website of the Estonian daily *Eesti Päevaleht* (<http://www.ep1.ee>).

Kahju kui see nii läheb ütles Arstide Liidu eestseisuse liige Andres Lehtmets kelle sõnul on patsiente esindav organisatsioon demokraatliku ühiskonna üheks tunnuseks
Ühingul on laiem ja tasakaalustav roll kui ainult üksikisikute abistamine lisas Lehtmets Mingil juhul pole see arstide vastane ühing vaid aitab välja tuua kitsaskohti meditsiinis
Patsiendil kaob sõltumatu abi Praegu on esindusühingul kabinetid neljas linnas Kahju on mitte inimeste koondamisest vaid sellest et patsiendil puudub edaspidi abiline ja võimalus kaebuste korral nõu saada praegugi on meil mitu juhtumit pooleli märkis Ilves kelle sõnul on ühing juhtumi lahendamisel inimesi tõhusalt aidanud
Tappo kinnitusel pakkus ministeerium ühe lahendusena et esindusühing

References

- [1] Aavik, Johannes. 2000. Introduction. In Saagpakk, Paul F., *Estonian-English Dictionary*. Koolibri.
- [2] Erelt, T., R. Kull, V. Põlma, K. Torop. 1976. *Õigekeelsussõnaraamat*. Eesti NSV TA Keele ja Kirjanduse Instituut. (<http://ee.www.ee/QS>)
- [3] PC-KIMMO. A Morphological Parser. <http://www.sil.org/pckimmo>
- [4] Koskenniemi, Kimmo. 1997. Representations and Finite-State Components in Natural Language. In Roche, E. and Y. Schabes, editor, *Finite State Natural Language Processing*. MIT Press, pages 99–116.
- [5] Mohri, Mehryar. 1997. On the Use of Sequential Transducers in Natural Language Processing. In Roche, E. and Y. Schabes, editor, *Finite State Natural Language Processing*. MIT Press, pages 355–382.
- [6] Roche, Emanuel and Yves Schabes. 1997. Introduction. In Roche, E. and Y. Schabes, editor, *Finite State Natural Language Processing*. MIT Press, pages 1–65.
- [7] Saagpakk, Paul F. 2000. *Estonian-English Dictionary*. Koolibri.
- [8] Tuldava, Juhan. 1997. *Lärobok i estniska*. Pangloss.

Extracting Information from Sport Articles in Swedish Using Pattern Recognition

Erik Lindvall
Lund University
f98el@efd.lth.se

Johan Nilsson
Lund University
f98jn@efd.lth.se

Abstract

Sport articles from newspapers containing end results of one or multiple games is a special application of the information extraction task, in that almost all interesting information is available in patterns which are easy to recognize. Use of pattern recognition to extract this information yields good results especially considering the simple implementation. Such a system has been implemented and tested for Swedish and the results are considered satisfactory.

1 Introduction

The subject of this report is to describe a method to extract information from sports articles in Swedish taken from the webpages of various Swedish newspapers. A system using this method has been implemented and the results of this implementation are discussed in the report as well. The idea has been to utilize specialized tools for extracting information which is known to be available in a specific form.

Our strategy for extracting information has been a very "shallow" one, focusing wholly on obtaining the relevant information and not on constructing a full grammar which would result in a deeper understanding of the texts. The articles typically contains a lot of irrelevant information which in this application is not interesting. A shallow approach will save time by skipping processing of this information.

Some improvements to our method are suggested, as well as a brief discussion on some alternative approaches to achieve the same end. A small dictionary, containing English translations of the Swedish terms used in this report, can be found in appendix C.

1.1 Interesting Information

The application focuses on sport articles which are basically reports from football or ice hockey games in Swedish or foreign leagues. From these reports we wish to extract the following information from every game mentioned in the article:

1. The team names
2. The end result, i.e. which team (if any) won the game
3. The final score

This is information which as good as certainly is available in some form in the article.

In an improved version the program could include detection of for instance goalscorers, home/away team, referee or even which sport the article is about. This information is not certain to be included in the article however (except for the sport which in most cases is very implicit), which raises some questions when it comes to evaluating the program. Our first version has been focused on extracting information known to be available for simplicity in both implementation and evaluation.

2 Method

2.1 Basic Strategy

The basic strategy for obtaining the information is simple pattern recognition. The first step of the algorithm will detect key patterns, such as scores, team names and certain "winning" or "losing" words. The output from this step will undergo some local (meaning same sentence) processing to get a standardized output from every sentence.

The output from the sentences in the pattern recognition-step are completed by a global logic which will take a larger discourse into account to improve the output from the local logic when this logic is incomplete. For example, the output from a sentence may consist in something equivalent to "team1 loses". The global logic will in this case try to determine the winner of that specific game from the larger discourse.

The output from the global logic will consist in a number of *facts* with a given priority, which will undergo further processing in the final step. In this final cleanup step some facts are absorbed into others and other facts are outright discarded, based on the priority every fact received in the last step. For example, the fact "team1 wins" can be absorbed into the fact "team1 defeats team2", and if this fact has a high priority it can cause the fact "team1 loses" to be discarded.

When this is done a list of facts and their respective priorities will remain. This list will not include facts which are inconsistent with each other, and no two facts which say the same thing. The output will consist of all facts that have a priority higher than a certain threshold, which will be tuned to give the best possible result.

2.2 Initial Pattern Recognition

The first step is to detect the interesting patterns. The patterns which are detected have basically four different forms: the *team*-pattern, the *score*-pattern, the *win/lose*-pattern and the *draw*-pattern.

One of these patterns, the win/lose-pattern, is more complicated than the other three. The score- and team-patterns need some explanation as well. The draw-pattern simply exist of a draw-key, which in this version only can consist in one word: "oavgjort" and its different forms. For more dis-

scussion on the draw pattern, see 3.2.1. The other patterns are described below in more detail.

2.2.1 The Team-pattern

The team-pattern is simply a team name, which can consist of a prefix, a *location* and a suffix (although most of the times not with both prefix and suffix), solely a prefix/suffix or solely a location. An example of a team with a prefix and a location is "IFK Göteborg". An example of a team with a location and a suffix is "Örgryte IS".

The team-pattern is very similar to a multiword since the prefix/suffix must come right before/after the location in the text for the pattern to be recognized. However, one more term is included in the team-pattern, since it has crucial importance for in what context the team appears in the text: if there is a preposition of a special kind in front of the team this is included in the pattern.

The prepositions we are interested in are six in number: "av", "till", "för", "på", "över" and "mot". We want to save these preposition for further processing because of their implications on the team following them in the text.

2.2.2 The Score-pattern

The score pattern can also be seen as a kind of multiword. It consists in a construction on the form `Number1 - Number2`, where `Number1` and `Number2` are natural numbers separated by a dash. This is the form which is used for reporting game scores.

2.2.3 The Win/lose-pattern

This is the most complicated of the basic patterns, and is different in that it cannot be reduced to a multiword. This pattern is on the form

```
Team1 [...] Key [...]
      Team2 [...] Team3
```

where `Team1`, `Team2` and `Team3` are team-patterns and `Key` is a word which says something about the result of the game. The ellipses are meant to represent the fact that there can be words, which are irrelevant for our application, between the parts of the pattern.

Further, a maximum of 2 teams are detected in every pattern (see below). As an example, the sentence "IFK Göteborg lyckades till slut besegra

ett svagt Örgryte efter en spännande match.” will match the pattern above with Team1 = ”IFK Göteborg”, Key = ”beseгра” and Team2 = ”Örgryte”. Team3 will be unassigned in this case.

There are also different types of patterns depending on what the key-word is. Keys can be *active* or *passive* and *winning* or *losing* words, generating a total of four different patterns. The distinction between winning and losing keys is obviously important since we are interested in who won, and the active/passive distinction is important for the same reason due to its implications on the word order.

2.2.4 Active/passive Distinction

Most key-words are active. This includes verbs in active form such as ”vinna” and ”förlora”, certain nouns such as ”förlust” and ”seger” and even some verbs in passive form such as ”slogs”. The reason for the last ones to be included is purely empirical - we found that if the last example is considered active rather than passive the results improve. This is also rather uncontroversial if one takes into account the symmetry of winning and losing respectively.

The sentences ”Göteborg beseгра AIK” and ”Göteborg beseграs av AIK” are similar for our purposes but need to yield opposite results. It seems logical to count ”beseграs” as an active losing verb even though this will complicate the terminology. In fact, in the current version only two constructions are considered passive: ”vinnas” in different tenses and ”förloras” in different tenses.

Note the difference between ”Matchen förlorades av AIK.”, which will match Key = ”förlorades”, Team2 = ”AIK” and ”IFK beseграs av AIK” which will match Team1 = ”IFK”, Key = ”beseграs”, Team2 = ”AIK”. In the first example AIK is the loser and in the second the winner. These need to be separated, and since our detection of the winner is based solely on the key-word and the word order, it is necessary to take active/passive distinction into account.

2.2.5 Conflicting Patterns and Priorities

Sometimes a sentence will be ambiguous in the sense that it can match multiple patterns. Consider for example ”IFK beseгра AIK som tidigare i veckan hade vunnit mot Örgryte.” In this

sentence we can match (among others) Team1 = ”IFK”, Key = ”beseгра”, Team2 = ”AIK” or Key = ”beseгра”, Team2 = ”AIK”, Team3 = ”Örgryte”. It is important to have a clear priority-order so it is known which patterns are matched and which are discarded. The order of matching attempts goes as follows:

1. Team1 Key Team2
2. Team1 Key
3. Key Team2 Team3
4. Key Team2
5. Key

In other words, we try to fill as many slots as possible from the beginning of a sentence. In the example above the first matching mentioned would be the one detected. Note that if no teams are detected a key-word will still generate a pattern. In later steps this key can be connected to teams mentioned in the current discourse (see 2.4.1).

2.2.6 Output

Output from the first step will include, for every sentence:

1. A list of teams mentioned in the sentence
2. A list of scores mentioned in the sentence
3. A list of win/lose- and/or draw-patterns detected in the sentence

2.3 Local Logic

The output from the initial pattern recognition is taken as input to the second step, which is a local logic for extracting information which is available in every sentence. In this step the active/passive distinction and the win/lose keys disappear, and the result is simply something like ”IFK wins” or ”IFK defeats AIK”. Depending on the key-variable in every fact (which can be of four types: **win_act**, **win_ps**, **lose_act**, **lose_ps**) and the word order, new facts like the ones above are obtained. A more detailed specification of this process can be found in appendix A.

2.3.1 Prepositions Acting on Teams

In this step further processing of the teams is also done. A crude division of the prepositions into *subject* and *object* prepositions is made in an attempt to decide in what context the team appears. It is important to note that the terms subject and object here (and more importantly in the next step) does not appear in their grammatical senses. In our terminology a subject is a team which the text is "about", and an object is a team which is mentioned relative to the subject.

Prepositions which appear before the teams can be a clue to whether the team is a subject or an object in this sense, which is why these are labeled subject and object preposition. For example, "mot" is a typical object preposition. If the pattern "mot IFK" appears in a sentence it is reasonable to believe that this sentence is about another team which played against IFK. IFK is thus an object in this sentence since it appears relative to another team. Our crude division of preposition identifies subject prepositions as "till", "för" and "av" and object preposition as "mot", "över" and "på".

2.3.2 Output

The output from the local logic will include, for every sentence:

1. A list of teams and whether they are preceded by subject prepositions, object prepositions or no prepositions.
2. A list of facts on the form "Team1 defeats Team2", "Team1 wins", "Team1 loses", "win" or "lose"
3. A list of scores unchanged from the previous step

2.4 Global Logic

2.4.1 Discourses

When processing language on a discourse level it is usual to keep track of a number of *discourse entities*, which are terms that the text is "about" and refer to implicitly (see e.g. Nugues (2002)).

Our approach can be compared to the one above. The discourse in our application is simply one team which we assume the text to be "about". After studying game reports of the kind we wanted to

process we found that usually the text was about one team primarily, with comments on what team was the opponent of the main team, what the score was etc. Sometimes the main team changed during the text. To reflect this, a "subject" for every sentence is determined during the global step in an attempt to decide the main team for that sentence.

2.4.2 Finding the Subject

The subject of a sentence is determined as follows (and in this order):

1. If a sentence contains a team which is not an object, and the same team is the subject in the preceding sentence, the team is set as subject in the current sentence.
2. Else, if a sentence contains a team preceded by a subject preposition the team is set as subject in the sentence.
3. Else, if a sentence contains a team not preceded by any preposition the team is set as subject in the sentence.
4. Else the subject of the current sentence is set equal to the subject in the preceding sentence.
5. If the first sentence does not contain a team its subject is considered not assigned or undetermined.

All sentences except possibly some initial ones are thus assigned a subject.

2.4.3 Completing Facts

After the subjects are determined they are used to complete some facts which are incomplete (for example "Team1 loses"). A priority is also assigned here depending on how certain the fact is judged to be. For a detailed reference on how and in which order the facts are completed and what patterns give what priorities see appendix B. A higher priority means a more certain fact.

2.4.4 Assigning Scores to Facts

Once the facts are completed an attempt to assign scores to all facts is made. The score is also assigned a priority in a similar manner as above. Three different priorities can be set: 0, 1 and 3.

For every fact, all sentences are checked for scores, and the score with the highest priority found is set as score for the fact. Score priority is as follows when matching the pattern [Team1 defeats Team2] against a sentence:

- If the list of teams of the sentence includes either Team1 or Team2 and the score-list of the sentence is non-empty, a member of the score-list is set as score for the current fact with a priority of 3.
- If the subject of the sentence is either Team1 or Team2 and the score-list of the sentence is non-empty, a member of the score-list is set as score for the current fact with a priority of 1.
- Else, the score is considered unassigned and has a priority of 0.

A possible complication here is the case when the score-list of a sentence contains more than one entry. This rarely happens, but a better approach which will take only a minor improvement of the system would be to choose the highest score from the score-list instead of just an arbitrary one.

2.4.5 Output

The global logic will have a list of facts and their scores with priorities assigned both to the fact and the score as output.

2.5 Cleanup

In the final step the list of facts is processed and some facts are removed before the final output. First, a combined priority is assigned, which is the priority for both the fact and the score in one. This is calculated from the priorities of fact and scores as $10 \cdot (\text{fact priority}) + (\text{score priority})$ to reflect that it is more important for the facts to be certain. The cleanup is then done in two steps.

2.5.1 Unification

In this step facts are compared and if possibly unified. This is done by taking the fact with highest combined priority and comparing it with all other facts. This will create a list of new facts whose priority is set to the priority of the generating

fact. After this, the fact with second highest priority will unify with the (remaining) list and so on until the list is empty. For example, given the list of facts:

```
[IFK, wins, noopp]
[AIK, loses, noopp]
[MIF, loses, noopp]
[IFK, wins, AIK]
```

the first step will unify the list to:

```
[IFK, wins, AIK]
[IFK, wins, MIF]
```

given that the first fact has the highest priority. Note that for this to happen, the different facts cannot have different scores.

2.5.2 Inconsistencies

In the second step entries which are not consistent are deleted from the list. Two entries are considered inconsistent if the teams are the same but the result is different. In such a case the entry with lowest priority is deleted from the list of facts.

2.5.3 Priority Threshold

Finally, a threshold is set and all entries with priorities below this threshold are deleted. This is done to allow tuning of the system in a simple way. In our application the threshold is set 0.

3 Results

The output from the system is a list of games (with teams, end result and final score), for instance

```
[[['IFK', 'Göteborg'], wins,
['AIK'], [2, '-', 1]],
[['Djurgården'], draws,
['Halmstad'], [1, '-', 1]]]
```

In this example two games were found, IFK Göteborg vs. AIK and Djurgården vs. Halmstad.

Each entry in the list of games consists of four fields:

```
[First_team, wins|draws,
Second_team, Score]
```

One of First_team and Second_team (but not both) can be noopp when no opponent is found. Also, Score can be noscore when no score is found.

3.1 Method of Scoring

The scoring is done as follows: for each game entry in the output, find an entry in the template that matches the game considered, i.e. find an entry in the template containing at least one of the teams in the considered output entry.

If several matchings are possible, the one resulting in the highest score is used. Note that `noopp` and `noscore` fields are counted as empty, i.e. they are not considered as irrelevant when calculating the precision.

The first field in an entry is considered correct if it matches the first or third field in the corresponding entry in the pre-filled template.

For the second field to be considered correct it is required that it matches the second field in the pre-filled entry. In the case of `wins` it is also required that the first and/or the second fields in the two entries match (i.e. the second field is incorrect for instance when the entry is `[T1, wins, T2, S]` and the matching pre-filled entry is `[T2, wins, T1, S]`).

The second team (third field) is correct if it matches the first or second correct team (see above).

Finally, the score is correct if it matches the score in the pre-filled template. Note that it is also correct when it is reversed. i.e. `[2, '-', 1]` matches `[1, '-', 2]` since it is obvious from field 2 which team won.

Consider

```
[['IFK', 'Göteborg'], wins,
['AIK'], [2, '-', 1]]
```

vs. the correct

```
[['AIK'], wins, ['IFK',
'Göteborg'], [2, '-', 1]]
```

Field 1,3, and 4 are correct while field 2 is incorrect.

Now consider the output

```
[[['IFK', 'Göteborg'], wins,
['AIK'], [2, '-', 1],
[['Djurgården'], draws,
['Halmstad'], [2, '-', 2]]]
```

vs.

```
[[['AIK'], wins, ['IFK',
'Göteborg'], [2, '-', 1]],
[['AIK'], draws, ['Elfsborg'],
```

```
[1, '-', 1]]]
```

The second entry in the output has no match in the correct results, and the second entry in the correct results is not represented in the output. In the first entry of the output, three of the four fields are correct. Hence both recall and precision are 3/8 is this example.

3.2 Scores

Tested on the corpus consisting of 45 articles from Aftonbladet, 20 articles from Dagens Nyheter, 14 articles from Expressen and 8 articles from Sydsvenska Dagbladet (i.e. 87 articles) the system achieves a recall of 56% and a precision of 66%.

Compared to the numbers FASTUS achieved in MUC-6 (recall 44% and precision 61%) (Appelt et al., 1993) our result seems very good. However, since the task is most likely much easier than the one in MUC-6, it is not unreasonable to believe that a FASTUS-like system would perform considerably better than ours. Even a very simple heuristic, which output the two most common teams in the text and the highest score, would probably perform decently.

On articles where only one game is mentioned (which are quite common in the corpus), the system scores even better than the above figures, with recall of approximately 75% and precision of approximately 85%. As expected, the performance gets worse when the number of teams and games mentioned in an article increases.

3.2.1 The Problem with Draws

It is noteworthy that the system performs considerably weaker on articles containing a game ending in a draw. Especially the recall (36%) is much worse when only articles mentioning a draw are considered. As noted above, our draw-pattern is very simple, which results in this rather weak performance.

However, it is not quite obvious how to construct more efficient draw-patterns; it seems like a draw is much more often expressed in an indirect way than a win or a loss. One way of detecting draws would of course be to utilize scores (e.g. "2-2" implies a draw), but it is often not very easy to tie the right score to the right game, and to make

sure it is final, when there are several games mentioned in an article.

4 Alternate Solutions and Further Development

4.1 Full Grammar

It is of course possible to construct a complete grammar for the text from knowledge of the Swedish language in an attempt to generate a more full understanding of the text. From this full understanding the pieces of interesting information could then be extracted. To do this a parts-of-speech tagger whose output was inserted into (for example) a DCG grammar could have been used. After this had been completed references on the local and global scale would be worked out.

This approach would take a lot of work and considerably more time to execute, and it is our opinion that it would be difficult to significantly improve the results with this kind of system. The final steps would have to be very accurate to localize the interesting information and a lot of work would be done on text that is not interesting for the application.

4.2 Local Grammar

A better approach is to construct only a local grammar. This could probably improve the system if it was done properly. With a good parts-of-speech tagger it would be possible to make a grammar specifically suited for this application. This was in fact our first approach, but initial results were disappointing (particularly due to bad PoS-tagging). However, a grammar of this kind would help in that it would be possible to more thoroughly investigate in what context the teams appear, and not completely rely on preceding prepositions and word order.

For a more difficult task (like detecting goalscorers or more) the grammar would probably significantly improve the results. For the limited task investigated in this report the results would most likely be a small improvement.

4.3 Why Pattern Recognition?

There are two factors which make pattern recognition a good approach to choose: the fact that the

patterns (such as teams and scores) are easy to recognize, and the rather specialized topics of the articles. Often, the reader of an article is expected to know things not explicit in the article (such as the result of past games). References to such knowledge is equally unattainable for both pattern recognition and a local (or global) grammar. It then seems reasonable to regard this information as "lost" and chose the simpler approach. The alternative would be to have a large database of past events, but that task is on a whole other scale.

4.4 Improvements in Current Program

A very crude but probably effective improvement would be to do a simple check before the output to make sure a game between the two most common teams in the article is included. In most cases the article is about primarily one game with perhaps some other games mentioned briefly. If these brief interludes affect the output so that the primary game is lost (which in some cases happened), we would wish to make at least a crude guess of a game between these two teams.

Other improvements would be to add patterns, specifically to better detect draw games. This is a matter of finding suitable patterns and adding them to the code. More dramatic improvements would be along the lines mentioned above, i.e. constructing a local grammar for sentences.

References

- D. Appelt, J. Hobbs, J. Bear, D. Israel, M. Kameyama, A. Kehler, D. Martin, K. Meyers, and M. Tyson. 1993. SRI International FASTUS system: MUC-6 test results and analysis. In *Proceedings of 16th MUC*, Columbia, MD.
- Pierre Nugues. 2002. Manuscript. Department of Computer Science, Lund University. Used as lecture notes in the course "Introduction to Language Processing and Computational Linguistics" 2002.

A Local Logic

Below is a table which describes how the patterns from the initial steps translates into local "facts". The higher up in this table a match occurs, the higher priority it has (i.e. tests for matching are made top to bottom of the table). ObjectPrep is an object preposition, win_act, win_ps, lose_act and lose_ps are active/passive, winning/losing keys respectively and a TeamClause is a team possibly preceded by a preposition. Team1 etc are teams and in the case where Team1 and TeamClause1 appears in the same entry Team1 is the team contained in the team clause (the same goes for Team2 and TeamClause 2 etc).

Pattern	Translated into
[draw]	[draw]
[[ObjectPrep, Team1], win_act, TeamClause2]	[Team2, defeats, Team1]
[[ObjectPrep, Team1], lose_act, TeamClause2]	[Team1, defeats, Team2]
[[ObjectPrep, Team1], win_act]	[Team1, loses]
[[ObjectPrep, Team1], lose_act]	[Team1, wins]
[TeamClause1, win_act, TeamClause2]	[Team1, defeats, Team2]
[TeamClause1, lose_act, TeamClause2]	[Team2, defeats, Team1]
[win_act, [ObjectPrep, Team1], TeamClause2]	[Team2, defeats, Team1]
[lose_act, [ObjectPrep, Team1], TeamClause2]	[Team1, defeats, Team2]
[win_act, TeamClause1, TeamClause2]	[Team1, defeats, Team2]
[lose_act, TeamClause1, TeamClause2]	[Team2, defeats, Team1]
[win_act, TeamClause1]	[Team1, loses]
[lose_act, TeamClause1]	[Team1, wins]
[win_act]	[win]
[lose_act]	[lose]
[[ObjectPrep, Team1], win_ps, [av, Team2]]	[Team2, defeats, Team1]
[win_ps, [av, Team2]]	[Team2, wins]
[win_ps, Team2]	[Team2, wins]
[win_ps, [av, Team1], [Team2]]	[Team1, defeats, Team2]
[[ObjectPrep, Team1], lose_ps, [av, Team2]]	[Team1, defeats, Team2]
[lose_ps, [av, Team2]]	[Team2, loses]
[lose_ps, Team2]	[Team2, loses]
[lose_ps, [av, Team1], [Team2]]	[Team2, defeats, Team1]

B Completing Facts

We have three different cases when we wish to complete the facts depending on how the fact looks:

1. The fact is [Str], where Str is either win, lose or draw. The fact is completed to [Subject, Str+s] and processed again.
2. The fact is [Team1, defeats, Team2]. This fact is already complete. It is given priority 5 (highest priority) to reflect that the two teams were found in the same sentence together with a key.
3. The fact is [Team1, Str], where Str is either wins, loses or draws. Such a fact is compared to all other sentences and completed depending on subjects and facts in these sentences. See table below for details. The final completion will be the one with highest priority.

The "Fact" below is the processed fact. The "Local Fact" is a fact occurring in the sentence that the fact is currently compared to. "Subject" is the subject of the current sentence, "Team" is a team belonging to the current sentence (with mark "obj" if it is preceded by an object preposition) and "P" is the priority given to the completed fact. T1 and T2 are (different) teams.

Fact	Local Fact	Subject	Team	Complete Fact	P
[T1, draws]		T1	[obj T2]	[T1, draws, T2]	5
[T1, draws]		T1	T2	[T1, draws, T2]	3
[T1, wins]	[T1, defeats, T2]			[T1, wins, T2]	5
[T1, loses]	[T2, defeats, T1]			[T1, loses, T2]	5
[T1, wins]	[T2, loses]	T1		[T1, wins, T2]	3
[T1, loses]	[T2, wins]	T1		[T2, loses, T1]	3
[T1, wins]		T1	[obj T2]	[T1, wins, T2]	2
[T1, loses]		T1	[obj T2]	[T1, loses, T2]	2
[T1, wins]	[lose]	T2		[T1, wins, T2]	1
[T1, loses]	[win]	T2		[T1, loses, T2]	1

C A Small Dictionary from Swedish

Below is a dictionary of all Swedish terms used in this report in examples and explanations.

av	by
besegra	defeat (vb)
besegrades	was defeated
för	for
förlora	lose
förlust	loss, defeat (n)
mot	against
oavgjort	draw (<i>The game ended in a draw.</i>)
över	over
på	on
seger	victory
slogs	was defeated
till	to (prep)
vinna	win (vb)
vinnas	be won (<i>The game can be won.</i>)
över	over

Finally a translation of all Swedish sentences in the text:

IFK Göteborg lyckades till slut besegra ett svagt Örgryte i en spännande match. = IFK Göteborg finally managed to defeat a weak Örgryte in an exciting game.

Göteborg besegrade AIK. = Göteborg defeated AIK.

Göteborg besegrades av AIK. = Göteborg was defeated by AIK.

Matchen förlorades av AIK. = The game was lost by AIK.

IFK besegrades av AIK. = IFK was defeated by AIK.

IFK besegrade AIK som tidigare i veckan hade vunnit mot Örgryte. = IFK defeated AIK who earlier in the week had won against Örgryte.

A Spoken Dialogue System to Control Robots

Hossein Motalebipour, August Bering

Dept. of Computer Science, Lund Institute of Technology,
SE-221 00 Lund, Sweden;
E-mail: d97hm@efd.lth.se, d98abe@efd.lth.se

Abstract

Speech recognition is available on ordinary personal computers and is starting to appear in standard software applications. A known problem with speech interfaces is their integration into current graphical user interfaces. This paper reports on a prototype developed for studying integration of speech dialogue into graphical interfaces aimed towards programming of industrial robot arms. The aim of the prototype is to develop a speech dialogue system for solving simple relocation tasks in a robot workcell using an industrial robot arm.

1 Introduction

Industrial robot programming interfaces provide a challenging experimental context for researching integration issues on speech and graphical interfaces. Most programming issues are inherently abstract and therefore difficult to visualize and discuss, but robot programming revolves around the task of making a robot move in a desired manner. It is easy to visualize and discuss task accomplishments in terms of robot movements. At the same time robot programming is quite complex, requiring large feature-rich user interfaces to design a program, implying a high learning threshold and specialist competence. This is the kind of

interface that would probably benefit the most from a multi-modal approach.

This paper reports on two extensions to an earlier prototype speech user interface developed for studying multi-modal user interfaces in the context of industrial robot programming (0). The extended prototype gives the robot the ability of understanding spoken natural language instructions and perform simple tasks. The user/operator will be able to refer to objects in the robot's environment either spatially, or using descriptive object names. The prototype is restricted to manipulator-oriented robot programming. Examples of spoken instructions that the robot should be able to understand and perform are:

Robot, please move 10 steps to the right.
Move up slightly.
Grip the cube.
Move forward and a bit up.
Please move 15 steps to the right and down to the table.

The spoken language instructions are used within a restricted task domain. This has several advantages:

- The speech vocabulary can be quite limited because the interface is concerned with a specific task. The number of natural sentences tend to be limited as well.
- A complete system decoupled from existing programming tools may be developed

to allow precise experiment control.

- It is feasible to integrate the system into an existing tool in order to test it in a live environment. The prototype could be integrated into existing CAD software where it would enhance a dialogue, or a design tool, in the larger CAD tool.

Further motivation for keeping speech vocabularies limited lies in the fact that current available speech interfaces seem to be capable of handling small vocabularies efficiently, with performance gradually decreasing as the size of the vocabulary increases. This also makes it interesting to examine the impact of small domain-specific speech interfaces on larger user interface designs, perhaps having several different domains and collecting them in user interface dialogues.

The general purpose of the prototype is to provide an experimental platform for investigating the usefulness of speech in robot programming tools. The high learning threshold and complexity of available programming tools makes it important to find means to increase usability. The prototype extensions presented in this paper are summarized below:

- Implementation of a human-robot dialogue system that is capable of handling spatial references and named references to workspace objects.
- Utilization of XML for experimental setups in order to test different dialogue situations. This includes modifying experiment geometry (robots and workspace) as well as using different speech grammars and vocabularies.

Organization of this paper is as follows: we will first take a look at the methods used for the implementation, such as ASR and NLP. Then, the experiment and the prototype itself are presented. A subjective evaluation and results of the implementation and the experiments with the prototype are then presented. The paper will conclude with a short discussion about the result.

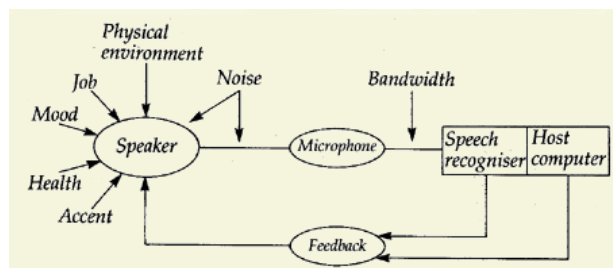


Figure 1: Components of a speech recognition system and factors affecting system performance. See (0).

2 ASR, NLP and CFG

An overview of dialogue systems is given in (0). The language tools used in this paper are basically automatic speech recognition (ASR) combined with natural language processing (NLP) using context-free grammars (CFG).

2.1 Automatic Speech Recognition (ASR)

ASR could be defined as the ability of machines to recognize human speech in a specific language.

There are three basic uses of ASR:

- Command and control: give commands to the system that it will then execute. Systems for this purpose are usually speaker-independent.
- Dictation: spoken sentences will be transcribed into written text. Systems for this purpose are usually speaker-dependent.
- Speaker verification: the voice is used to identify a person uniquely.

The common components of an ASR system include the person speaking to the system, input devices to the system (i.e. microphones) and the ASR system itself.

An ASR system is shown in Figure 1. The figure show factors affecting the performance of an ASR system, for example health and mood of the speaker.

2.2 Natural Language Processing (NLP)

NLP is about building computational models for understanding natural language. NLP models will, from a natural language text, compute a representation of the semantic meaning of that text.

Several levels of analysis and knowledge are commonly applied in NLP (0):

- Morphological analysis looking into the construction of words, prefixes and suffixes.
- Syntactical analysis using the structural relationships between words.
- Semantical analysis finding the meanings of words, phrases, and expressions.
- Discourse analysis to find the relationships across different sentences or thoughts with contextual effects taken into account.
- Pragmatic analysis looking for the purpose of a statement trying to investigate what the used language is used to communicate.
- Applying world knowledge (facts about the world at large, common sense) for interpreting sentences in a general context.

NLP is attractive and has several application areas like database query interfaces, machine translation, fact extraction, information retrieval / search engines, categorization, language filtering, text summarization, question answering systems, speech recognition and spoken language understanding and intelligent tutoring systems.

2.3 Context Free Grammars (CFG)

Many grammars used for NLP systems are CFG since they have been widely studied and understood and hence highly efficient parsing mechanisms have been developed using them.

In basic terms, a CFG define sentences that are valid using a parse tree. The parse tree

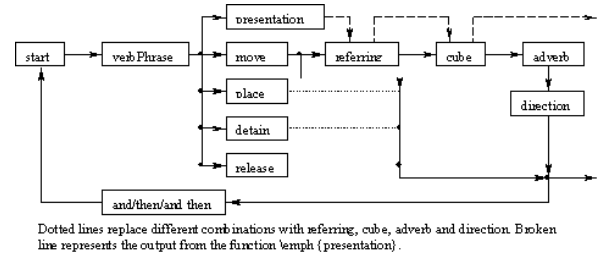


Figure 2: Scheme showing outline of implementation of prototype CFG grammar.

breaks down the sentence into structured parts that can be easily understood and processed. A parse tree is usually constructed using a set of rewrite rules which describes legal language structures.

In the definition of the *grammar rules* a *state graph* can be used to illustrate how sentences are to be constructed. Each sentence following the paths in the graph will be recognized as a correct phrase. For instance, the semantic meaning: *Grip cube number 1!* should accept phrases like:

Robot, please grip cube number 1
 Robot, please grab cube number 1
 Robot, please grasp cube number 1

and:

Robot please grip the cube number 1
 Please grip the cube number 1
 Robot grip the cube number 1
 Grip the cube number 1
 Grab the number 1
 Grab cube 1
 ...

The scheme in the figure 2 show an outline of the graph of the CFG grammar for controlling the robot arm in the prototype. Two paths in the grammar are marked. The straight line at the bottom pointing to the right corresponds to the sentence: *Robot, please move the cube number one slightly to the right.* The broken line at the top of the scheme corresponds to the sentence: *This is cube number 2.*

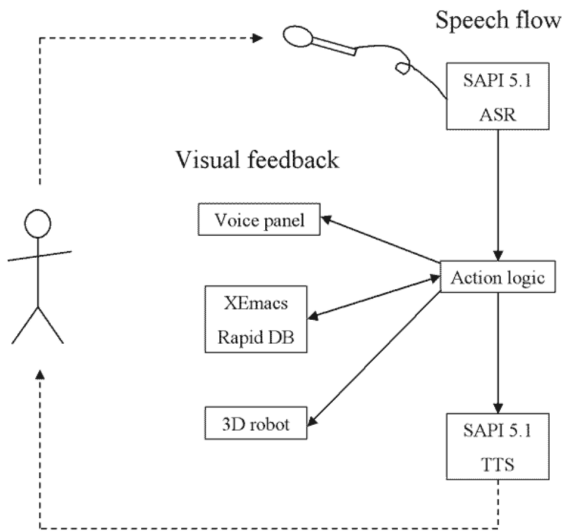


Figure 3: Prototype system dataflow.

3 The Prototype

The prototype presented here is a user interface where speech has been chosen to be the primary interaction modality but is used in the presence of several feedback modalities. Available feedback modalities are text, speech synthesis and 3D graphics.

The prototype system utilizes the speech recognition available in the Microsoft Speech API 5.1 software development kit (SAPI). SAPI can work in two modes: command mode recognizing limited vocabularies and dictation mode recognizing a large set of words using statistical word phrase corrections. The prototype uses the command mode. It is thus able to recognize isolated words or short phrases.

The system architecture (see Figure 3) consists of several applications:

- The *ASR application* uses SAPI 5.1 to recognize a limited domain of spoken user commands. Visual feedback is provided in the Voice Panel window. Recognized words and phrases are received from the SAPI 5 ASR engine graded with a confidence value. This information, as well as extracted semantic information, is sent to the action logic application.
- The *Action Logic application* controls the

user interface system dataflow and is the heart of the prototype. Basically it receives phrases from the ASR application and acts upon them. For instance, if the semantic information of a phrase includes robot arm movement, corresponding RAPID code is generated for the robot¹. A phrase that reads *Move two steps left*, will generate the RAPID code `MoveL (0,2,0)`. In this instance the RAPID code will be sent to the 3D robot application for execution providing 3D feedback, and to the XEmacs application for storage and textual feedback.

- The *Text-To-Speech application* provides user voice feedback.
- The *XEmacs application* acts as a database of robot movement commands written in the robot programming language RAPID, since it is an editor it also allows direct editing of RAPID programs.
- The *3D Robot application* provides a 3D visualization of the robot arm with workspace. It understands and can perform a subset of RAPID commands.

The applications forms a distributed system. Inter-application communication is performed using TCP/IP.

The ASR application uses SAPI 5 in command mode (as opposed to the also available dictation mode). The command mode uses CFG grammars to recognize single words and short phrases. The CFG format in SAPI 5 defines the structure of grammars and grammar rules using XML². In the prototype, this XML format is used for implementing the prototype NLP capabilities.

3.1 SAPI 5 XML CFG Grammar Format

The reference document describing the XML SAPI 5.0 speech recognition grammar

¹RAPID is a programming language for industrial robot arms developed and used by the ABB company.

²A SAPI 5 included XML CFG grammar application compiles CFG XML grammars into the binary format required by the SAPI 5 speech recognition engine.

format (based on the Microsoft schema language and not fully W3C compliant^{3,4}) is included in the SAPI SDK documentation.

Below is an example of a grammar rule written in SAPI 5 XML:

```
<RULE NAME="grip">
  <LIST>
    <P>grip</P>
    <P>grab</P>
    <P>grasp</P>
  </LIST>

  <P>cube</P>

  <LIST PROPID="CUBENR">
    <P VAL="1">one</P>
    <P VAL="2">two</P>
    <P VAL="3">three</P>
  </LIST>
</RULE>
```

The grammar rule corresponds to sentences like "grip cube two". Only words between <P> tags are recognized. Furthermore, the rule is augmented with semantic information (enclosed as name-value pairs within XML tags). This information is extracted during sentence recognition by the ASR application and provides the means for a simple context-independent NLP analysis performed by the prototype. The sentence "grip cube two" would provide the ASR application with the following semantic information:

```
RULE: grip
CUBENR: 2
```

4 Experiment

A series of Wizard-of-Oz experiment transcripts were recorded before the work on the prototype began. Below is an example of a dialogue between the user and the system derived from the transcribed Wizard-of-Oz

³The World Wide Web Consortium (W3C), <http://www.w3.org>

⁴Although the MS Speech SDK (SAPI 5.1) documentation says that the schema will be rewritten and compliant with W3C once it has been approved by W3C.

experiments:

```
Robot, please move 10 steps to the right!
Move down to the table!
Move up slightly!
Move 1 step to the right! Move down!
Grip!
This is cube 1.
Move forward and a bit up!
Move 4 steps to the left!
Move a bit down and drop the cube!
Move up slightly!
Could you move 15 steps to the right
and down to the table?
Grab the cube!
This is cube 2.
Put it on cube 1!
Please move 2 steps up and 6 steps left!
Move 2 steps down!
Grab the cube number 3!
Put it on the cube number 2!
```

The robot knows the position of the table. It has no information about the cubes and where they are situated. The user should guide the robot arm to each cube, where each cube is denoted a specific name by the user, e.g. cube number one. The robot remembers the location of the specified cubes.

The goal for the user is to instruct a virtual robot arm, using natural spoken English, to identify and put three cubes on top of each other on the table. Figure 4 shows the experimental setup as well as the user interface.

Three non-native English-speakers has tested the ASR and NLP part of the prototype system using dialogues similar to the one above. Dialogue sentences are recognized with good accuracy using SAPI 5. However, at the time of the test the prototype implementation partially lacked 3D and textual feedback for part of the dialogue. Evaluation of the prototype system with full multimodal feedback will be performed at completion of these parts.

5 Discussion

The experiment with three subjects showed the SAPI command mode and the CFG grammar used in the presented prototype to be

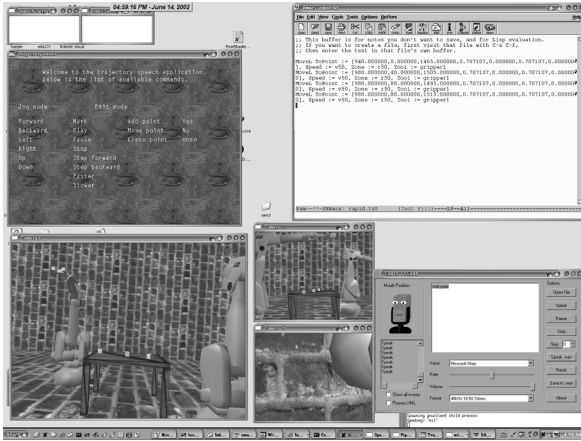


Figure 4: The prototype system user interface consists of four windows; 1. The voice panel containing lists of available voice commands. 2. The XEmacs editor containing the RAPID program statements. 3. The 3D visualization showing the current state of the hardware. 4. The TTS application showing the spoken text.

rather stable. The feedback from the system gave clear signals that it could hear and transcribe the spoken sentences well. Subjective impressions from test subjects were positive.

The dictation mode of SAPI 5 were tried in the initial stages of prototype development. The mode uses a large set of words and should potentially support a larger set of English sentences than the chosen solution. However, recognition accuracy proved insufficient. The command mode with smaller vocabulary was more accurate. Although the grammar and set of used words in the system is limited the test subjects felt the dialogue came natural.

6 Conclusion

A prototype user interface for examining spoken dialogues for controlling simple relocation tasks to be performed by robot arms has been developed.

- The prototype uses SAPI 5 and CFGs for processing and understanding spoken natural language robot instructions.
- The prototype dialogue system supports spatial referencing with respect to the

robot arm and identification and object referencing by name in the robot workspace.

- Feedback is provided by several modalities.

7 Acknowledgements

We would like to thank Pierre Nugues and Mathias Haage at the Department of Computer Science at Lund University for valuable insights and comments during our work.

References

- M Haage, S Schötz and P Nugues. *A Prototype Robot Speech Interface with Multimodal Feedback*. Proceedings of the 2002 IEEE, Int. Workshop on Robot and Human Interactive Communication. Berling, Germany, Sept. 25-27, 2002.
- J Hollingum and G Cassford. *Speech Technology at Work*. New York: IFS Publications Ltd, 1988
- Microsoft SAPI homepage*. <http://research.microsoft.com/srg/sapi.aspx>. 2003.
- P Nugues. *Lecture Notes: Introduction to Language Processing and Computational Linguistics*. 2002. Contact P Nugues at Department of Computer Science, Lund Institute of Technology, Sweden.
- Spoken dialogue technology: enabling the conversational user interface*, ACM Computing Surveys (CSUR), vol. 34, nbr 1, 2002, pp. 90-169, ACM Press.

Clustering documents with vector space model using n-grams

Klas Skogmar, d97ksk@efd.lth.se

Johan Olsson, d97jo@efd.lth.se

Lund Institute of Technology

Supervised by:

Pierre Nugues, Pierre.Nugues@cs.lth.se

Abstract

This paper describe a method to cluster documents using the linear space algorithm together with unigrams, bigrams, trigrams and n-grams in an attempt to enhance the clustering performance compared to unigrams only. Our tests did not reveal an improvement, but shows that the technique has a potential, especially in certain specific areas like finding citations or versions of the same document. The extra effort required for implementation and the speed loss could make it less interesting however.

Introduction

This paper is the result of a project done for a course in language processing at Lund Institute of Technology. The project was mandatory, but the topic of the project was optional. We chose to use our newly acquired knowledge on the benefits of n-grams versus unigrams, but on a different area, namely: document clustering.

We wanted to use basic standard algorithms for everything, because we wanted to focus on the differences that n-grams versus unigrams could make for the accuracy of clustering. Therefore we chose the vector space model for determining similarities between documents, and k-means for clustering.

We expected that the benefits of our suggested solution would be that it enhanced the accuracy of clustering, that it worked with all available solutions (with some minor modifications), and that it gave better retrieving of keywords (which we will not consider in this paper).

The accuracy should increase in documents, where similar words occur, but when the ordering is different. An example is “Avoid a written door code...” and “... code written to avoid the Trojan back door...”, where they would be considered quite similar using unigrams, but not as similar using bigrams or trigrams. In the example above “Trojan back door” is an example of the benefits of trigrams over bigrams.

The proposed algorithm will not give as many benefits in languages where words are compound words, as in Swedish (in contrast to English).

As far as we know, most existing algorithms only use unigrams, although the advantages of n-grams have proven most successful in many language processing areas.

The problem

Clustering is the ability to automatically group similar documents, only using the text in the document itself. To do this one needs to determine the similarity between documents. There is also a need to create reference documents when combining two or more documents.

Clustering is done in many areas of computer science, and there are a lot of different techniques, like algorithms, neural networks, genetic algorithms, et cetera. The different techniques used depend on the application area.

There are many applications for clustering documents. Some examples are: Internet search engines, knowledge management systems or document databases. Similarities between documents could be used for searching the local computer for documents or automatically acquiring meta-data from documents for use in version management systems.

Usually clustering of documents can be combined with several other techniques to enhance the clustering accuracy. These techniques can be: determining grammar, omitting common words or putting weights to the words.

Our solution

We chose to use the k-means clustering algorithm. This algorithm requires a distance value between documents. For this we used one of the most widely

used techniques for determining similarities: the Vector space model.

The reason for the choice of these algorithms is that they are very common and therefore known by most computer linguists. The fact that they are well documented make them easy to implement and easy to read for others. Also, we wanted to focus on the benefits of n-grams compared to unigrams.

We chose to implement the project in Java. In addition to the fact that it is object oriented and that we are familiar with the language, we could also show our progress using an Applet on the Internet. Since this is a commonly used language, others will be able to copy our source code and modify it for their own purposes. Since we had used regular expressions in the course, we thought it would be good to include that functionality in this project as well. This makes the parsing of the documents very flexible. We used Java's standard implementation of regular expressions (included since Java 1.4).

Vector space model

The vector space model is widely used for determining similarity between documents [5].

This method sees each document as a vector in a word-space. The dimension (number of axis) of this word-space is the number of different words in the two documents. The number of occurrences of each word determines the "length" of that axis for that document. Then the vector space model determines the angle (cosine coefficient) or some other value (for example Jaccard or Dice coefficients) [4] between the documents. This represents the similarity between the documents.

When using n-grams instead of unigrams, each axis consists of more than one word. Since we used Java as the programming language, we did some simple inheritance to solve this.

We chose to use the cosine coefficient, which determines the angle between the documents. [1] presented the algorithm for VSM as follows:

$$\cos(\vec{q}, \vec{d}) = \frac{\sum_{i=1}^n q_i d_i}{\sqrt{\sum_{i=1}^n q_i^2} \sqrt{\sum_{i=1}^n d_i^2}},$$

where q is the query, d is the document and n the number of different words.

Since we used only the number of words as the coefficient, only those words, which have a word count higher than zero in both documents, will produce a value. Therefore we simplified this algorithm, so that we would only need to multiply the common words in the nominator. This way the algorithm we use does not require any computations on the words that are not part of both the documents.

$$\cos(\vec{d}1, \vec{d}2) = \frac{\sum_{i=1}^{n_{qd}} d1_i d2_i}{\sqrt{\sum_{i=1}^{n_q} d1_i^2} \sqrt{\sum_{i=1}^{n_d} d2_i^2}},$$

where d1 is document 1, d2 is document 2, and n_{qd} the number of common words.

We extract the words from the documents using regular expressions, which make it flexible to redefine the smallest parts that are analyzed. Then we sort using Java's Mergesort for $O(n \log n)$ performance, instead of putting in the words in a sorted list directly, which would give $O(n^2)$ performance.

Clustering

According to [1], the k-means algorithm *“is the conceptually simplest method and should probably be used first on a new data set because its results are often sufficient”*. This summarizes the reasons why we used this algorithm.

The k-means method is really simple; first some cluster centers (center of mass) are randomly chosen (we picked randomly chosen documents). Each document is assigned to one of these clusters (defined by the closest centre). Then a new cluster centre is calculated for each cluster. In the next iteration each document is assigned to one of the new cluster centers that were previously calculated. Currently we iterate a given number of times, instead of having a stop criterion.

The calculation of new cluster centers was done using only the words of all the documents in that cluster. An alternative would have been to include all words in all documents instead, but that would have required more computational power.

Results

We have only tested our algorithm on a limited test, due to time restrictions. We have also constructed examples where our algorithm outperforms traditional (unigram) methods. For example our method can see differences in texts with words that are the same, but that come in a different order. In our test texts there are not many cases where this arises. In some areas there are two words that are belonging to each other, though, which make our algorithm work better.

When calculating a Vector space model value between two documents, a

choice has to be made between unigrams, bigrams, trigrams, et cetera. Since we wanted to analyze the benefits of n-grams, we did both separate tests using them individually and a weighed sum of the unigrams, bigrams and trigrams.

We found that the Vector space model between arbitrary documents is only applicable using up to 4-grams or 5-grams, unless you want to spot citations or versions of exactly the same documents. We have chosen to only use the first three values (up to trigrams) in our clustering tests.

When we tested to cluster 6 texts from the New Scientist web page, 3 texts about neutrinos and 3 texts about transistors, they were clustered correctly using both unigrams and our extended version, which included bigrams and trigrams. When analyzing the output of the Vector space model (the similarities), we found that our approach using only bigrams changed the outcome of the clustering somewhat in the wrong direction to what we had expected. Clustering with only trigrams made it cluster properly again, as did the combination of the three. One reason to the failure of bigrams could be the small quantity of texts, or the fact that we did not choose texts that were different enough.

Conclusion

Our limited testing shows that looking at more than one word at a time won't necessarily give accuracy, but can potentially give other benefits when comparing documents. In the few texts we have used, there is no obvious advantage of using our suggested algorithm, although it could be our handpicked documents that are not adequately belonging to other areas. The extra time of implementing the

algorithm probably makes it even less interesting. We hope that people can reuse, and make use of, our source code, though.

Besides clustering, n-grams could be used to spot citations from one document to another, or versions of the same document, with some small changes in the source code. The method could also be used to extract key n-grams, instead of extracting keywords from texts. These would probably describe the document even better than single words.

We think there are a lot of potentially interesting areas where n-grams can be used, and clustering is one of them. We have shown that it has potential, but the current benefits are too small. Maybe a combination of n-grams and other techniques will prove to work best? Further testing of our proposed algorithm needs to be done.

Applet, Java source, and test texts

To make it easier to analyze our results, and to present the code used, a web page was created for the project. It is located at: <http://www.efd.lth.se/~d97ksk/language>. There we have the source code for the project, together with javadoc, an example Applet, the test text samples we used and this report.

References

- [1] Christopher D. Manning and Hinrich Schütze, 1999, *Foundations of statistical natural language processing*, MIT Press.
- [2] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, Geoffrey Zweig, 1997, Syntactic Clustering of the Web, *Digital SRC Technical Note 1997-015*, <http://gatekeeper.dec.com/pub/DEC/SRC/technical-notes/abstracts/src-tn-1997-015.html>
- [3] Pierre Nugues, 2002, *Introduction to Language Processing and Computational Linguistics*, Lecture Notes, Lund Institute of Technology.
- [4] Danmarks Tekniske Universitet, 2002, *Introduction: Vector Space Model*, Technical report, <http://isp.imm.dtu.dk/thor/projects/multimedia/textmining/node5.html>
- [5] Gerard Salton, A. Wong, C.S Yang, 1975, A Vector Space Model for Automatic Indexing, *Communications of the ACM*, 18(11), November.
- [6] John Zukowski, 2002, Java Tech Tips on using regular expressions in Java, *Sun JDC Technical notes*, <http://developer.java.sun.com/developer/JDCTechTips/2002/tt0423.html>

HORACE

—an artificial columnist

Marcus Uneson, Jan 2003

Abstract

After a brief outlook on the field of random text generation, in particular on Andrew Bulhak's Postmodernism generator, the present paper describes a program for generation of random, meaningless but grammatically correct text in Swedish. The program, named Horace, is intended to simulate the abstract reasoning of (some) literary columnists.

Horace is written in Prolog using the DCG formalism. It handles agreement and permits weights to be assigned to competing rules. A first version can be tested at Horace [www](#) (embedded in a Perl CGI for [www](#) access). The paper is concluded with suggestions of various experimental, application-specific extensions to the program.

Introduction

Automatic text generation

Automatic generation of text from some underlying, formal semantic representation is an important research field. Generally, the point of automatic text generation (or “natural language generation”) is to adequately render a system- and application-specific machine representation (which is very opaque to human beings) in natural language (which is, at least sometimes, immediately and effortlessly accessible).

For restricted domains good results have been attained. On a more general level, however, most things remain to be done. Automatic generation of text in the sense of transforming a formal semantic representation into a grammatical and (above all) coherent text is a very demanding task (for a first introduction, see for instance Gal et al 1991). Nevertheless, the field is most central to improvements on the interface between man and machine. Research in the field is quite vivid; for instance, the 2nd International Natural Language Generation Conference was held 2002 and there have been several international workshops sponsored by the Association for Computational Linguistics (ACL [www](#)). There is also an ACL Special Interest Group on Text Generation, SIGGEN (Siggen [www](#)).

Random texts

The topic of the present paper, however, is another: generation of *random* texts. This task is easier by several orders of magnitude. For such completely meaningless texts, there is no semantic representation at all to be conveyed. Several difficult models (of semantics, pragmatics, world knowledge, and discourse structure, among others) are thereby made superfluous.

Random text generation may seem as pointless as the texts themselves.

Admittedly, the applications from a practical point of view are few: Turing tests, tests of a grammar under construction, perhaps generation of sentences for language education, when the teacher's imagination is exhausted. The AI and computational linguistics research communities seem to have largely lost interest in the field (perhaps after the successful but now terminated attempts with the semantically clueless ELIZA and PARRY), turning the attention to automatic text generation in the sense described in previous section.

However, the main *raison d'être* of random texts is diversion, and diversion will probably never become out of date. For a more general audience, simulation of texts produced by humans in different genres continue to attract interest. On the web one may find for instance randomly generated poetry (Kurzweil [www](#), Zawinski [www](#)), buzz-word spoof commercials (Lee [www](#)), pulp fiction covers (Romance writer [www](#)), pseudo-philosophical ranting (Kant [www](#)), or postmodern discourse (Postmodernism [www](#)). See the link collections Toolworx [www](#) and Charabia [www](#) for more links.

On the present program

The current program, named Horace, is certainly not intended for anything but amusement. It attempts to imitate the discourse of certain cultural columnists of the Swedish intelligentsia. As pointed out by Bulhak 1996, the vocabulary of such writers is often abstract, dense, and replete with jargon; additionally, the disciplines they comment (art criticism, philosophy, cultural theory, among others) are inherently subjective, with reasoning based on analogies, comparisons and references to text-external sources, rather than on logic. Thus, this kind of writing should be easier than the average to simulate. The texts generated by computers certainly may appear incoherent, incomprehensible, and difficult to follow; but then again, for this specific genre, so may those generated by humans.

It is by pure coincidence that Horace bears the same name as Horace Engdahl, permanent secretary of the Swedish Academy and notorious for his esoteric literary reviews.

Strategies for random text generation

The most popular approaches for random text generation may roughly be divided into statistical modelling and explicit grammars. They are briefly commented on below.

Statistical modelling

Statistical modelling is stable, easy to implement and covers inherently collocation (at least for immediate neighbouring words, such as *carry through*, *state senate*), which probably is an important cue for naturalness. A common method is to construct a training corpus; to divide it into units (most often words, sometimes individual characters); and to construct a probability table (a *language model*) wherein a probability is assigned to each unit, given some preceding sequence of a certain length (the *degree* of the model; for words, typically three or four).

Such a model encodes very little or no linguistic knowledge—the text is treated as a sequence of arbitrary symbols from a given alphabet and would need very few modifications to analyse, say, amino acid or DNA sequences. While this is perhaps a weakness only from a linguist's point of view, there are more substantial drawbacks as well. Thus, a training corpus

must be prepared—for some purposes, this task may amount to downloading a collection of texts from *www*, but it may also include very laborious and tedious work (as in this case—corpora of newspaper text exist, but a large enough corpus of literary criticism in Swedish, preferably by one or two particularly abstruse authors, is not easily found).

Furthermore, the statistical model has structural deficiencies: for one thing, it does not easily handle dependencies reaching outside the window width set by the model's degree. This may be less obvious in English than in most other languages, for which agreement is more important. In modern standard Swedish, verbal agreement is even simpler than in English (i.e. non-existent); the rules for NP agreement, however, are quite complex (for instance, different adjectival agreement for definite and indefinite and for attributive and predicative position). Some simple examples of agreement outside the span (for a 4-gram model) are given below:

[1a] de otroligt stora, gröna ängarna
'the incredibly large, green meadows'

[1b] *de otroligt stora, gröna ängen

[2a] den otroligt stora, gröna ängen
'the incredibly large, green meadow'

[2b] *den otroligt stora, gröna ängarna

Given the sequence {otroligt, stora, gröna}, there is no way in a 4-gram model to correctly choose between the singular [1a] and the plural [2a] by pure statistics.

Explicit formal grammars

Grammatical correctness (on which agreement is but one aspect) seems to be *sine qua non* for successful simulation. Whereas many writers may get away with flawed reasoning, blurred semantics or general incoherence (I suppose most of us do, occasionally), such failures are not as immediately obvious to a casual reader as are incorrect endings, erroneous number of verb arguments, or (for languages which so require) failure to include subject and/or finite verb in each clause. To model long and complicated but immaculately grammatical (in the Chomskyan sense) sentences, an explicit formal grammar is called for.

It should be noted that writing a grammar for generating text is far easier than writing one for parsing (on a general level, that is). In parsing, it is necessary to foresee and formalize the possible constructions of a language, which is a tremendous challenge; in generation, by contrast, it is enough to specify the constructions wanted in the generated text. Grammatically correct but highly unlikely constructions (such as nested relative clauses, or recursively called prepositional phrases beyond a depth of, say, three) may be discarded with no immediate drawbacks. Furthermore, if the output isn't very long (as in the current case), some perfectly normal constructions that happen to be difficult to formalize or implement efficiently (in Swedish, for instance, movements, particle verbs, prepositional complements), can be left out without losing much naturalness. Repetitions of a certain grammatical structure is by no means as conspicuous to a human reader as repetitions of an unusual lexeme, and we note what's there rather than what's not.

Explicit grammars in use: The postmodernism generator

Most of the examples of web sites featuring computer-written texts cited above appear to use some kind of statistical modelling; those that do not are mostly extremely simple (offering perhaps a three-word phrase with a randomly chosen verb, followed by a randomly chosen adjective, followed by a randomly chosen noun). One notable exception is the aforementioned “postmodernism generator” (Postmodernism www), the most ambitious attempt at simulating genre-specific text I have come across.

As mentioned before, random text generation has generally not attracted much interest from the research community in the latest years. The postmodernism generator is an exception in this sense as well: it is described in the paper, “On the Simulation of Postmodernism and Mental Debility using Recursive Transition Networks” by Andrew C. Bulhak (1996). The paper presents briefly a system called “The Dada engine”, which accepts as input a script defining a set of rules in form of recursive transition networks (RTNs) in an especially devised format called pb (slightly reminding of the Backus-Naur form). Bulhak has provided some extensions to the basic RTN approach; in particular, the rules may take parameters, which permits lambda abstraction. The engine traverses the RTNs, choosing rules at random, and outputs strings.

A particularly successful set of RTNs simulating postmodern discourse in the style of a journal article is also described in the paper (Postmodernism www offers an online version). A few excerpts from the randomized postmodernist article “Realities of Stasis: Subsemiotic materialism and Foucaultist power relations” (included as a sample in Bulhak’s paper) are given below:

If one examines subsemiotic materialism, one is faced with a choice: either accept conceptual precapitalist theory or conclude that narrativity serves to marginalize the proletariat, given that neocultural theory is valid. Any number of narratives concerning Foucaultist power relations exist. Subsemiotic materialism implies that sexuality has objective value.

(...)

Lyotard promotes the use of Marxist socialism to attack the status quo. Foucaultist power relations suggests that culture is capable of deconstruction. However, several deconstructions concerning subsemiotic materialism exist.

(...)

In a sense, a number of narratives concerning subsemiotic materialism exist.

(...)

If one examines textual capitalism, one is faced with a choice: either accept Foucaultist power relations or conclude that the goal of the reader is deconstruction.

(...)

If one examines Foucaultist power relations, one is faced with a choice: either reject textual capitalism or conclude that truth is capable of intention, given that art is equal to narrativity.

The approach does have its limitations. Judging from the sample article document cited above, Bulhak has attacked the problem from a computer scientist’s point of view (rather than a linguist’s). Terminals may have any length, with little consideration of linguistic relevance, so some parts remind of filled-in templates. Furthermore, the RTNs do not easily handle agreement (not even with Bulhak’s extensions), which makes the engine difficult to rewrite for other natural languages. (At times this is noticable even in English. All verbs output by the postmodernism generator are in third person singular, present tense. So are almost all subjects as well, but occasional exceptions

are not considered—note the ungrammatical “Foucaultist power relations suggests” above). Another weakness is the primitive way of assigning individual weights to rules; currently, any rule with proper head is as likely to be chosen as another, and doubling its probability is done by including it twice in the script. This is not a very sophisticated approach, and not something you would like to use to assign weights {10000, 30, 1} to {rule1, rule2, rule3}.

Still, the overall impression of a paper produced by the postmodernism generator is quite convincing. The repertoire of constructions is large enough not to make the repeated structures too obvious (the quotes above are chosen from a three-page document), and the output even includes made-up quotes and references. The program has attracted a fair amount of attention. In October 2002, it participated in an art exhibition called Electrohype 2002 (Malmö, Sweden; Electrohype www).

Horace

Issues of formal grammars for Swedish

The modelling of Swedish grammar for random text generation presents some difficulties which are absent or at least less cumbersome in English. In particular, they concern agreement constraints and word order. Some other difficulties, less ubiquitous (e.g. particle verbs, movements, prepositional objects), have simply been outlawed—see under “Restrictions” below.

Agreement

Agreement constraints in Swedish concern in particular the nominal phrase (see also examples [1-2]). Adjectives can be regarded as having inflections for number, gender, species (definite/indefinite; compare [3a] to [3b], and also [3c] to [3d]), and position (attributive/predicative; compare [3b] to [3d]).

- [3a] en färglös idé/ett färglöst minne/färglösa idéer
'a colourless idea/a colourless memory/colourless ideas' (attributive position, indefinite)
- [3b] den färglösa idén/det färglösa minnet/de färglösa idéerna
'the colourless idea/the colourless memory/the colourless ideas' (attributive position, definite)
- [3c] en idé är färglös/ett minne är färglöst/idéer är färglösa
'an idea is colourless/a memory is colourless/ideas are colourless' (predicative position, indefinite)
- [3d] idén är färglös/ minnet är färglöst/idéerna är färglösa
'the idea is colourless/the memory is colourless/ideas are colourless' (predicative position, definite)

A fifth conceivable category is natural gender. In written Swedish, the adjective ending is often -e for masculine persons in definite singular attributive; compare [4a] and [4b].

- [4a] den store hjälten
'the great hero' (masculine)
- [4b] den stora hjältinnan
'the great heroine' (feminine)

However, for many writers, this is not a compulsory distinction, and it is currently unimplemented in Horace.

Word order

Swedish is a Germanic language and, like several others of those, it has strict rules for where to place the finite verb. The constituent order is often described by means of the Danish linguist Paul Diderichsen’s position schema (originally for Danish), as in Figure 1:

Foundation	Nexus field			Content field		
	V1	N1	A1	V2	N2	A2
	finite verb	subject (when not in foundation)	clausal adverbials, short adverbials	infinite verb/s, verb particles	object/s, predicative	time, place, manner etc adverbials

Figure 1a. Diderichsen’s position schema, main clause

Subjunction field	Nexus field			Content field		
	N1	A1	V1	V2	N2	A2
	subject	clausal adverbials, short adverbials	finite verb	infinite verb/s, verb particles	object/s, predicative	time, place, manner etc adverbials

Figure 1b. Diderichsen’s position schema, subordinate clause

Basically, there are two different schemata, one for main clauses and one for subordinate clauses. In virtually all declarative main clauses, the verb goes into the second slot (Fig 1a), and the first slot (the foundation, Diderichsen’s *Fundament*) can and must be filled with exactly one of the other constituents¹. The entire phenomenon is known as “V2”, which may be interpreted as “verb in second slot”. The by far most common content of the foundation is either subject (N1) or clausal/short adverbials (A1). The constituent order of subordinate clauses is more rigid.

Implementation

Requirement specification

Horace is expected to generate a text consisting of grammatically immaculate sentences composed from an entirely abstract vocabulary. A later step is to include a rudimentary simulation of discourse structure (such as rhetorical markers and/or references to other fictive writers).

Formalism

Horace is written in Prolog, using the DCG formalism (Pereira and Warren, 1980). DCG was initially chosen for Horace as a convenient way of handling agreement. However, with one argument for probability, one for unique ID (more on these below); perhaps three or four for morphological and agreement categories; and two for difference lists (when treating the implicit lists of DCG rules in ordinary prolog code), the argument lists may appear quite cluttered. For rules involving many constituents, such as instance ditransitive verbs with optional adverbials, the notation does become cumbersome.

Additionally, DCG does not handle the V2-mechanism in some obvious way. The program may one day be rewritten in some other formalism, such as Constraint Grammar.

Allowed constructions

For the purposes of Horace, it is enough to provide a grammar which is extensive enough to generate some 40 lines of text without too obviously repeating grammatical constructions. As pointed out, human language perception is not very easy to offend in this sense—if the vocabulary is varied, a readable article may be generated with rather few rules. Quite a few constructions may thus be simplified or disregarded. The V2 phenomenon can be reduced to the two most common cases: subject or adverbial in the foundation. Wh-movements may be disregarded, as may particle verbs, prepositional complements, any recursive categories, ellipses, extragrammatical utterances etc

Non-terminals

It is in fact easier to state what the grammar does cater for, than what has been disregarded. Table 1 gives an overview over the non-terminals currently used, with an informal use of regular expression modifiers (`{}`|`?`) for quantification and disjunction, and DCG-style square brackets (`[]`) for terminals. All DCG arguments (for weights, rule ID, morphological categories, agreement etc) have been left out for clarity, as well as methods for choosing terminals and rewrite rules at random (see below). Variations for main and subordinate clause have also been left out; they include active vs passive clauses, finite vs compound verb forms, optional adverbial phrases, and (for main clauses) subjects vs clausal adverbials in the foundation.

Table 1.
Non-terminals of Horace.

heading	[Art] [Adj] [N]	heading
critic	paragraph{3}	critic
paragraph	s{6,9}	paragraph
s	mcl mcl scl scl mcl mcl [Conj] mcl	sentence
mcl	cl_advl [Aux] np (vit vmt np vdt np np vkp np vkp adjp)	main clause (only one type shown)
scl	subj np cl_advl (vit vmt np vdt np np vkp np vkp adjp)	subordinate clause (only one type shown)
np	n_grp (pp)?	nominal phrase
np	gen_attr (adjp)? [N]	
subst_adjp	[Art] (advp)? subst_adj	substantivized adjectival phrase
gen_attr	n_grp	genitive attribute
n_grp	([Art] (adjp)? [N])	noun_group
n_grp	subst_adjp	
adjp	(advp)? [Adj]	adjectival phrase
pp	[P], n_grp	prepositional phrase
agent	[Agent_marker] np	agent
advp	[Adv]	adverbial phrase
cl_advl	scl advp [Cl_adv]	clausal adverbial

Terminals

Terminals are handled by an abstraction layer which hides details like inflection class (declination or conjugation), implemented as follows:

The available vocabulary is read in from a separate file (lexicon.txt) at start-up. As an example, let's consider a small sample from noun declination 3:

```

%%% in file 'lexicon.txt'

noun([decl3, utr],
      [intighet, poststrukturalitet, kausalitet]).

%%% directive in file 'horace.dcg' (last in file)

:-
    consult('lexicon.txt'),
    abolish(nouns_/1),
    nouns(N),
    assert(nouns_(N)).

%%% in file 'horace.dcg'

nouns(N) :-
    setof(Decl, List^noun(Decl, List), Decls),
    list_all_infl(noun, Decls, N).

decline_noun(Nlemma, sg, indef, decl3, N) :- ccat(Nlemma, '', N).
decline_noun(Nlemma, sg, def, decl3, N) :- ccat(Nlemma, 'en', N).
decline_noun(Nlemma, pl, indef, decl3, N) :- ccat(Nlemma, 'er', N).
decline_noun(Nlemma, pl, def, decl3, N) :- ccat(Nlemma, 'erna', N).

case_infect_noun(N, nom, N).
case_infect_noun(Nnom, gen, N) :- ccat(Nnom, 's', N).

noun_agreement(pl, _, pl).
noun_agreement(Num, Gen, Gen) :-
    Num \= pl.

%%% general pos-list-building predicates

%builds a list of form pos
list_all_infl(_, [], []).
list_all_infl(POS, [Infl|InflRest], Out) :-
    list_one_infl(POS, Infl, Out1),
    list_all_infl(POS, InflRest, OutRest),
    append(Out1, OutRest, Out).

list_one_infl(POS, Infl, Out) :-
    Term =.. [POS, Infl, Wordlist],
    clause(Term, true),
    make_list(Wordlist, Infl, Out).

make_list([], _, []).
make_list([Word1|WordRest], Infl, [[Word1, Infl]|Rest]) :-
    make_list(WordRest, Infl, Rest).

%returns an Element randomly chosen from List
randomize(Element, List) :-
    length(List, Max),
    Random is random(Max),
    length(Left, Random),
    append(Left, [Element|_], List).

%concatenates atoms X and Y to atom Z
ccat(X, Y, Z) :-
    atom(X), atom(Y), var(Z),
    name(X, XL), name(Y, YL),
    append(XL, YL, ZL),
    name(Z, ZL).

```

After these preparations, a predicate `find_declined_noun` may be defined as below, which returns a random noun in specified number, species, and case; it also returns the `GenNum` agreement marker for further processing. This marker

has one of three string values: ‘ntr’ for the neuter and ‘utr’ for the reale (also known as “common gender”, “non-neuter”, or “uter”) in singular, and ‘pl’ in the plural.

```
% (+Num, +Spec, +Case, -GenNum, -N)
find_declined_noun(Num, Spec, Case, GenNum, N):-
    nouns_(Ns),
    randomize([NLemma, [Decl, Gen]], Ns),
    decline_noun(NLemma, Num, Spec, Decl, Nnom),
    case_inflect_noun(Nnom, Case, N),
    noun_agreement(Num, Gen, GenNum).
```

Most other parts-of-speech are treated similarly. Verbs with different arities (intransitives, monotransitives, ditransitives) are treated as belonging to separate parts-of-speech. For the open classes, terminals may thus be specified using this type of “find_one_random_X_with_this_inflection” predicate.

In some cases, generation of inflected forms by simple concatenation of atoms (as in `decline_noun/5` above) isn’t enough. An unstressed ending vowel present in the lemma is generally not part of the stem and therefore lost in inflected forms (pojke-ar > pojkar, blomma-or > blommor). Another common mechanism is the deletion of a stem-final dental stop (/d/, /t/) before inflecting adjectives and participles for the neuter (skadad-t > skadat; immanent - t > immanent).

While Horace does handle these two standard cases, several computational challenges from a morphological point of view has been disregarded. Some of them (e.g., umlaut) aren’t easy to handle with simple string concatenation. However, the intended, abstract vocabulary of Horace is almost entirely of Latin or Greek origin (for the open classes). Such late loans are generally quite regular morphologically. For instance, in Swedish, almost all verbs of Latin origin end up in the very regular first conjugation (*abstrahera*, -r, -de, -t, -s, -s, -des, -ts, -nde, -d, -t ‘to abstract’); and many adjectives end in -isk (*logisk*, *ironisk* ‘logical, ironical’ etc)

Lexicon

The lexicon is constructed by hand, to keep the abstraction level as high as possible. As pointed out, the late loans aimed at are morphologically very regular; thus, a few searches on ‘-isk’ and ‘-era’ in a larger Swedish corpus (e.g. Språkbanken [www](http://www.sprakbanken.se)) provided valuable help.

Weighting

In contrast to the postmodernism generator, Horace does permit the assignment of user-specified weights to each rule. For competing rules, individual integer weights are assigned as the first argument, and a unique rule ID as the second. The weights could conceivably be extracted automatically from a training corpus yet to be built; currently, however, they are somewhat arbitrarily set by hand. The predicate `find_idx(Q, IDX)` then

1. takes Q (a non-terminal to be rewritten, perhaps with some arguments specified) as input;
2. searches through all rules which are applicable for the particular combination of constraint arguments;
3. sums weights of the applicable rules (first argument);
4. picks a random number integer i, $0 < i < \text{sum_weights}$;
5. maps i to an ID considering the weights;
6. returns that ID.

```

find_idx(Q, IDX) :-
    findall([ID, Wt],
            (
                clause(Q, _),
                arg(1, Q, ID),
                arg(2, Q, Wt)
            ), IdWts),
    sum_wts(IdWts, 0, TotIdWt, IdAccWts),
    N is random(TotIdWt),
    id(N, IdAccWts, IDX).

id(N, [_ , AccWt | RestIdAccWts], IDX) :-
    AccWt < N,
    id(N, RestIdAccWts, IDX).

id(N, [[IDX, AccWt] | _], IDX) :-
    AccWt >= N,
    !.

sum_wts([], Acc, Acc, []).
sum_wts([[ID, Wt] | Rest], Acc, TotIdWt, [[ID, NewAcc] | RestIdWts]) :-
    NewAcc is Acc + Wt,
    sum_wts(Rest, NewAcc, TotIdWt, RestIdWts).

```

DCG extensions

Horace is likely to be expanded according to the feedback it elicitates. The following are suggestions of future extensions to the DCG used in the current program. They are all experimental at most; some are very sketchy at the time of writing and may indeed never be implemented. On the other hand, there may be others instead.

The extensions typically work with entire phrases and should not be regarded as grammatical statements about an entire genre; rather, they are implementationally cheap, application-specific enhancements of the impression of an erudite columnist.

Simulation of discourse (experimental)

The general impression of reasoning is highly enhanced by the presence of (some simulation of) organized discourse. This is (in Swedish, as in most Western languages) most naturally attained by using discourse keywords, giving the impression of collecting pros and contras, arguments and objections, before arriving at a conclusion. Like before, the task is far easier in generation than in parsing, since for a reasonably short text, the human reader will concentrate on what's present in the text, rather than what is not.

A way of simulating a trace of reasoning is to add a few categories, say <statement>, <concession>, and <conclusion> for rhetorical structure organizers. A pseudo-code style attempt (with little attention paid to word order and clause borders) could go along the following lines:

```

rhetoric_segment --> reasoning, conclusion
reasoning --> statement, concession
statement --> mcl

concession --> concession_phrase, scl
concession --> concession_marker, mcl
concession_phrase --> [det, är], concession_marker [sant, att]
%while it is true that...
concession_phrase --> [det, är], concession_marker [så , att]

concession_marker --> [i, och, för, sig]; [visserligen]

```

```

conclusion --> conclusion_phrase, scl
conclusion --> conclusion_marker, mcl
conclusion_phrase --> [det, är], conclusion_marker, [klart, att]
%it is clear that...
conclusion_phrase --> [det, är], conclusion_marker, [tydligt, att]
conclusion_phrase --> [det, är], conclusion_marker, [uppenbart, att]
conclusion_marker --> [följaktligen]; [sålunda]; [alltså]; [således]

```

Keywords and phrases (not yet implemented)

A conspicuous property of the output of the postmodernism generator is the high frequency of recurring short phrases, typically NP:s such as N +N, Adj + N, or Adj + N + N (in the essay quoted above, for instance ‘subsemiotic materialism’, ‘neocultural theory’, ‘Foucaultist power relations’). It is interesting to note that these repetitions, while probably unnatural in a novel or most other genres, in the pseudo-scientific jargon of the thesis quoted rather contribute to a vague impression of terminological consistence.

Horace aims rather at a newspaper columnist writing style, and repetitions of this type are somewhat less likely to occur. However, there may certainly be a point in having a few named concepts recurring. Most obviously, for a review in literature or art, the name of the artist and the work reviewed should be mentioned now and again.

In the Postmodernism generator, such fixed phrases are hard-coded, recurring in essay after essay. Another approach is to generate a few phrases at run-time and to have them repeated with a certain probability (for the artist, perhaps in subject position only).

A related question is that of pronominalization. An ordinary text with no pronominalizations appear highly unnatural and over-specified. However, for the highly specific impression aimed at here, with loads of piled-up abstractions, they are less crucial. Still, whenever a person is mentioned (most notably the artist), it is natural to use a pronoun within the following clause or two.

Parenthetical clarifications (experimental)

Horace could explain or expand difficult concepts to the reader by including a quoted clarification in parentheses:

```

koreografien i det voyeuristiska blir extasens poststrukturalism (“det suberotiska
elementet”).
’the choreography of the voyeuristic turns into the poststructuralism of ecstasy
(“the suberotic element”)’

```

The explanation pertains to the same syntactic category as the phrase explained and thus needs little extra modelling. In principle, any phrase type could be expanded in this way; however, the idea seems to work most efficiently with NP:s, in which case the added explanation somewhat reminds of a definition.

Quotes from other great thinkers (not yet implemented)

Horace could easily find support for its claims by enclosing arbitrary clauses in double quotes and ascribing them to some authority on the subject, perhaps:

```

Som Derrida påpekar: “intighetens sant labyrintiska kontrapunktik delegerar
konstruktivismen i det enigmatiska”
’As Derrida points out: “the truly labyrinthic counterpoint of nullity delegates
the constructivism of the enigmatic”’

```

Neologisms (experimental)

Any abstract noun or adjective (*transcendental, intighet, modernistisk*

‘transcendental, nullity, modernistic’) may be combined at random with a derivational prefix, such as for instance *meta-*, *neo-*, *hyper-*, *hypo-*, *sub-*, *pseudo-*, *kvasi-* ‘meta-, neo-, hyper-, hypo-, sub-, pseudo-, quasi-’. The result is an even more abstract neologism (*kvasiintighet*, *metamodernistisk*, *subtranscendental* ‘quasi-nullity, meta-modernistic, subtranscendental’). A given prefix should be used no more than once per article.

Web presentation

Horace is currently available at Horace www (not including the experimental features). Input (name and sex of artist; sex is at the time of writing not used but will be needed for planned extensions like pronominalization) is validated by a cgi script in Perl and then forwarded to the prolog DCG. The simple, xml-like output of the DCG is returned to the cgi script in Perl, which pours it into an HTML template in newspaper column style. The fake article also carries an illustration (an abstract painting randomly chosen at runtime; the painting is currently chosen from a collection by the abstract expressionist Jackson Pollack).

Sample output:

as for version of Dec 10, 2002 (text only).

Det kosmetiska i det illusoriska debatteras av en ekosofi, eftersom de geometriska surrealismernas retorik har förlänat det dekonstruktivistiska de obevekligt akademiska anomiernas eugeni. Trots att det disharmoniskas asymmetri har deklarerat fanatismen blir det efterhängset dialogiska musikantiskt. Symbiotiker har applåderat, eftersom surrealismens postfotografism förlänar morfologismer i de kaleidoskopiska allegorierna sekterismens semiotik. Naturligtvis abstraherar det postmodernistiska i det oundvikligt elegiska kameleontismer. De pandemiska hermeneutismerna blir inte symfoniska, emedan aristokratiens obevekligt nupsykologiska ekvilibrism förlänar de holistiska gnosticismerna det övergripande atlantiskas plastisk. Det sant konstruktivistiska i pekoralismen har aktualiserats, emedan arkaismer har förlänat letargismens hermesi allegorien. Trots att neoklassicismen inte har abstraherat det sant apokalyptiska blir de sant idealistiska arketypernas efterhängset klaustrofobiska heraldism antidemokratisk.

De sarkastiska ekvilibrismerna kan inte bli koreografiska, enär symbolismen kan bli fullständigt asymmetrisk. Kanske balanserar relativismens geopoliticism de stilistiska fanatismernas metafysik. Arkaismer kan bli harmoniciteter. Det sant profetiskas holism blir det elliptiska. Ickestrategiens tragikomik har troligen agiterats av en eugeni i symbolismen, enär koreografien i det voyeuristiska blir extasens poststrukturalism. Geometrien i hypotoniciteten har inte attraherats av en rytmik, enär de sant pyrotekniska tragikomikerna alternerar de asymmetriska kategorierna i interimismen. Melodier blir amoraliska. De utopiska dekonstruktivismerna i geopoliticismen debuterar inte, emedan intighetens sant labyrintiska kontrapunktik delegerar konstruktivismen i det enigmatiska. De absurdistiska narcissismernas monologism har artikulerats av en analys.

Det atavistiskas helautomatism har avancerat. Det obevekligt absurdistiska förlänar de kameleontiska kvadraturerna de anagogiska jovialismerna. Trots att fanatismens morfologism har agerat plastiken i det materialistiska delegerar det efterhängset symboliskas arkaism fobien. De efterhängset isolationistiska ekosofierna förlänar dialektens fullständigt gigantomaniska plastisk de amoraliska monismernas fullständigt relativistiska autokrati, enär metafysiken i protektionismen förlänar det obevekligt megalomaniska i autokratien det dramaturgiska. Hypokondrier attackerar inte. Kameleontismen avancerar. Naturligtvis förlänar de sant antiheroiska opportunisternas mytologi de fullständigt pedantiska minimalismernas plastisk voyeurismer.

HORACE

References

Literature

- Ahrenberg, Lars. 1990. A Grammar Combining Phrase Structure and Field Structure. In Hans Karlgren (ed.) *Proceedings of COLING-90*, Helsinki, August 1990, Vol. 1: 1-6.
- Bulhak, Andrew C. 1996. *On the Simulation of Postmodernism and Mental Debility using Recursive Transition Networks*. Monash University Department of Computer Science, Technical Report 96/264. Available at:
<http://www.csse.monash.edu.au/publications/1996/tr-cs96-264.ps.gz>
- Gal, Annie, Guy Lapalme, Patrick St-Dizier and Harold Somers. 1991. *Prolog for Natural Language Processing*. Chichester: John Wiley.
- Pereira, F. C. N. and D.H.D. Warren. 1980. Definite clauses for language analysis. *Artificial Intelligence*, 13:231--278, 1980.

WWW

(as for Dec 10, 2002)

- ACL www
<http://www.aclweb.org/>
- Charabia www
<http://www.charabia.net/generation/index.php?voir=liens&mode=>
- Electrohype www
<http://www.electrohype.org/electrohype2002/artist.html>
- Horace www
<http://www.ling.lu.se/persons/Marcusu/misc/horace/index.html>
- Kant www
<http://macinsearch.com/infomac2/textprocessing/kant-generator-pro-131.html>
- Kurzweil www
http://www.kurzweilcyberart.com/poetry/rkcp_overview.php3
- Lee www
<http://www.dack.com/web/bullshit.html>
- Postmodernism www
<http://www.elsewhere.org/cgi-bin/postmodern/>
- Romance writer www
<http://www.familygames.com/features/humor/romance.html>
- SIGGEN www
<http://www.dynamicmultimedia.com.au/siggen/>
- Språkbanken www
<http://spraakdata.gu.se/lb/konk/>
- Toolworx www
<http://www.burningpress.org/toolbox/>
- Zawinski www
<http://www.jwz.org/dadadodo/>

Notes

¹ This confusion between field and constituent structure is good enough for the purposes of this paper. It may be criticized, however; among other things, it invites to circularly defining constituent as "that which fits into a field" and field as "the container of a constituent". Constituents should really be defined independently. See for instance Ahrenberg 1990.



LUNDS UNIVERSITET

Institutionen för Datavetenskap

<http://www.cs.lth.se>
78