Proceedings of the 10th Nordic Conference of Computational Linguistics NODALIDA-95 Helsinki 29-30 May 1995

Edited by
Kimmo Koskenniemi

University of Helsinki
Department of General Linguistics
P.O. Box 4 (Keskuskatu 8)
FIN-00014 University of Helsinki
Finland

PUBLICATIONS
NO. 26
1995

Preface

The Department of General Linguistics at the University of Helsinki arranges this tenth Scandinavian Conference of Computational Linguistics or "10:e Nordiska Datalingvistikdagarna", in short NODALIDA-95.

The event is directed to the scholars, students and professionals in various fields of computational linguists as well as to all other interested parties. Most of the participants which have preregistered are from the Nordic countries.

Papers were invited from all areas of computational linguistics, including but not restricted to: morphological, syntactic, semantic and textual analysis and generation, speech processing, machine translation and processing of monolingual or multilingual text corpora.

The conference is committed to meet two goals:

- To be a forum for presenting high quality scientific papers. Such papers were refereed and selected by a program committee and referees.

- To be an occasion for computational linguists in the Nordic Countries meet and get to know each other, as well as inform each other about their interests, projects and status of their research. Short papers or posters are intended to accomplish this task.

A program committee was established consisting of prof. Lauri Carlson, doc. Gunnel Källgren, director Bente Mægård, prof. Torbjørn Nordgård, prof. Anna Sågvall Hein, and Ph.D. Atro Voutilainen, and Kimmo Koskenniemi as the chairman. In addition to the members of the program committee, PhD Maria Vilkuna and PhD Lars Borin acted as additional referees.

Altogether 32 submissions were received by the program committee. About 18 papers were considered as potential long papers, others being explicitly submitted as posters or short papers. All candidates for long papers were reviewed by at least to referees independently. Finally, six papers were accepted to be printed in this volume and presented in the conference as long plenum papers. The long papers are arranged in the alphabetical order of the authors.

Short papers and abstracts of posters will be copied and distributed separately to the participants of the conference.

This conference was arranged almost exclusively using electronic mail and the Internet. No hardcopy "call for papers" or the like were ever mailed. Most of the refereeing also proceeded electronically, and most of the camera ready papers were transmitted via the net. We consider the possibility of placing the material in a suitable FTP site for easy access by the community.

May, 1995 Kimmo Koskenniemi

Contents

Tagging the Teleman Corpus

Thorsten Brants and Christer Samuelsson
Universität des Saarlandes
FR 8.7, Computerlinguistik, Postfach 151150
D-66041 Saarbrücken, Germany
Internet: {thorsten,christer}@coli.uni-sb.de

Abstract

Experiments were carried out comparing the Swedish Teleman and the English Susanne corpora using an HMM-based and a novel reductionistic statistical part-of-speech tagger. They indicate that tagging the Teleman corpus is the more difficult task, and that the performance of the two different taggers is comparable.

1 Introduction

The experiments reported in the current article continue a line of research in the field of part-of-speech tagging using self-organizing models that was presented at the previous (9th) Scandinavian Conference on Computational Linguistics. Then, the well-established HMM-based Xerox tagger, see [Cutting 1994], was compared with some less known taggers, namely a neural-network tagger described in [Eineborg & Gambäck 1994], and a Bayesian tagger presented in [Samuelsson 1994]. The Xerox tagger performs lexical generalizations by clustering words based on their distributional patterns, while the latter two utilize the morphological information present in Swedish by generalizing over word suffixes.

This time, another HMM-based approach is compared with a novel reductionistic statistical tagger inspired by the successful Constraint Grammar system, [Karlsson *et al* 1995].

The performed experiments do not only serve to evaluate the two taggers, but also shed some new light on the Teleman corpus as an evaluation domain for part-of-speech taggers compared to other, English, corpora.

The paper is organized as follows: Section 2 discusses the Teleman corpus and the tagsets used. Section 3 describes the HMM-based tagger and Section 4 the reductionistic statistical one. The vital issue of handling sparse data is addressed in Section 5 and the experimental results are presented in Section 6.

2 The Teleman Corpus

The Teleman corpus [Teleman 1974] is a corpus of contemporary Swedish, representing a mixture of different text genres like information brochures on military service and medical care, novels, etc. It comprises 85,408 words (tokens; here, words is a collective denotation of proper words, numbers, and punctuation). There are 14,191 different words (types); the most frequent one is ".", which occurs 4,662 times; the most frequent proper word is "*och*" (*and*), which occurs 2,217 times. 8,458 of the words occur exactly once, which is 60% of the types but only 10% of the tokens.

Table 1: Comparison of Teleman and Susanne corpora

	Teleman	Susanne
size	85,408 words	156,644 words
word types	14,191 words	14,732 words
most freq. word	4662×"."	9641×"*the*"
one occurrence	8,458 words	6,820 words
unknown words	10% expected	4% expected
tagset	258 tags	424 tags[1]
max. tags/word	15 (for "*för*")	14 (for "*as*")
reduced tagset	19 tags	62 tags
max. tags/word	7 ("*för*", "*i*")	6 ("*a*", "*no*")

[1]Tags in the Susanne corpus with indices are counted as separate tags.

For the experiments, we used two different tagsets. First, we used the original tagset, consisting of 258 tags. Each of the 14,191 word types can have between one and 15 of the 258 tags (the highly ambiguous word "*för*" (*for, stern, lead, too, ...*) has the maximum number of tags). We then used a reduced tagset, consisting of 19 tags, which represent common syntactic categories and punctuation. This tagset is identical to that used in the publications mentioned above. Each of the word types then has between one and 7 tags ("*för*" and "*i*" have the maximum number of tags).

2.1 Comparison with an English Corpus

Since 10% of the words in the Teleman corpus occur only once, we expect from the Good-Turing formula [Good 1953] that 10% of the words in new text be unknown, which is a very high percentage. Other publications typically report 5%. Since most of the work in this area is on English corpora, we compared the Teleman corpus with an English corpus, namely the Susanne corpus [Sampson 1995], which is a re-annotated part of the Brown Corpus [Francis & Kucera 1982], comprising different text genres. The relevant facts are summarized and compared in Table 1. The major difference (apart from corpus size and tagsets used) is the percentage of words that occur exactly once: 10% for Teleman vs. 4% for Susanne. According to the Good-Turing formula, this percentage is identical to the expected percentage of unknown words. Actual counts by dividing the corpora into training and test parts yield around 14% and 7%, respective. This indicates that unseen Swedish text will have substantially more unknown words than unseen English, which is most likely due to the higher degree of morphological variation in Swedish.

A further difficulty with the Swedish corpus is the higher degree of ambiguity. In the Teleman corpus, each word in the running text has in average 2.38 tags for the small tagset, and 3.69 for the large tagset. These numbers are 2.07 and 2.61 for the Susanne corpus, despite the fact that the tagsets for the Susanne corpus are larger than those for the Teleman corpus. Thus, there is much more work for the tagger to do in the Teleman corpus. Some more numbers: in the running text, 54.5%/64.2% of the words in the Teleman corpus are ambiguous, and only 44.3%/48.9% in the Susanne corpus (small/large tagset, resp.; see Table 2 for further details).

3 The HMM Approach

A Hidden Markov Model (HMM) consists of a set of states, a set of output symbols and a set of transitions. For each state and each symbol, the probability that this symbol is emitted by that state is given. Also, a probability is associated with

Table 2: Distribution of number of categories per word in running text for the Teleman and Susanne corpus, small and large tagsets.

	Teleman		Susanne	
	small	large	small	large
1	45.5%	35.8%	55.7%	51.1%
2	16.2%	12.2%	17.4%	19.2%
3	17.7%	14.1%	4.8%	4.2%
4	6.4%	10.0%	11.1%	4.5%
5	9.0%	5.3%	8.4%	9.2%
6	1.7%	6.2%	2.6%	2.2%
7	3.5%	4.3%	–	2.2%
8	–	1.1%	–	4.8%
9	–	2.9%	–	–
10	–	2.7%	–	2.0%
11	–	1.6%	–	–
12	–	2.9%	–	–
13	–	–	–	–
14	–	–	–	0.6%
15	–	0.9%	–	–
> 1	54.5%	64.2%	44.3%	48.9%

Table 3: Teleman corpus parts

	total words	unknown words[2]
part A	67,402	—
part B	9,262	1,421 (15.3%)
part C	8,774	1,198 (13.7%)
Σ	85,408	

[2]Unknown words are words that occur only in the test set, but not in the training set.

Table 4: Susanne corpus parts

	total words	unknown words
part A	127,385	—
part B	9,752	714 (7.4%)
part C	9,684	563 (5.8%)
Σ	146,821[3]	

[3]The remaining 9,823 words of the Susanne corpus were not used in the experiments.

each transition between states (see [Rabiner 1989] for a good introduction). The transition probability, and thus the probability of the following state, depends only on the previous state for first order HMMs, or on k previous states for HMMs of kth order. HMM approaches to part-of-speech tagging make the well-known assumption that the current category or part-of-speech of a word depends only on the previous $(n-1)$ categories (Markov assumption), thus they assume that natural language is a Markov process of order $(n-1)$, which of course is not true, but a successful approximation. $n = 3$ is chosen in most of the cases, resulting in a trigram model (i.e., always working with a window of size 3), since it yields the best compromise between size of corpora needed for training and tagging accuracy. Furthermore, the current word (symbol) depends only on the current category (state). Thus, instead of calculating and maximizing $P(T_1 \ldots T_k \mid W_1 \ldots W_k)$, with T_i tags and W_i words, which is impossible in all practical cases, one calculates and maximizes

$$\prod_{i=1}^{k} P(T_i \mid T_{i-n+1} \ldots T_{i-1}) P(W_i \mid T_i) \tag{1}$$

to find the best sequence of tags for a given sequence of words.

The parameters of an HMM can be estimated directly from a pretagged corpus via maximum-likelihood estimation (MLE). But MLE sets a lot of the transition probabilities to zero, and if one of the multiplied probabilities in (1) is zero, the product becomes zero, leaving no means to distinguish between different products that contain a zero probability. This results in poor estimates for the probabilities of new sequences of words. This problem is addressed in Section 5.

Another way of estimating the parameters of an HMM is to use an untagged corpus, a lexicon with parts-of-speech lists for the words and the Baum-Welch algorithm [Baum 1972]. This approach has the advantage of avoiding the tedious work of manually annotating a corpus, but it requires a sophisticated choice of initial biases, and generally, the performance is worse than that achieved with annotated corpora.

When using an HMM for tagging, the system gets a string of words and has to find the most probable sequence of tags that could have produced the string of words. This is done with a dynamic programming method, the Viterbi algorithm [Viterbi 1967]. The algorithm finds the most probable sequence of states in time linear in the length of the input string.

4 The Reductionistic Statistical Approach

Although not yet fully realized, the basic philosophy behind the reductionistic statistical approach is to give it the same expressive power as the Constraint Grammar system.

4.1 Constraint Grammar

The Constraint Grammar system performs remarkably well; [Voutilainen & Heikkilä 1994] report 99.7% recall, or 0.3% error rate, which is ten times smaller than that of the best statistical taggers. These impressive results are achieved by:

1. Utilizing a number of different information sources, and not only the stereotyped lexical statistics and n-gram tag statistics that have become the de facto standard in statistical part-of-speech tagging.

2. Not fully resolving all ambiguities when this would jeopardize the recall.

Property 2 means that the system trades precision for recall, which makes it ideal as a preprocessor for natural language systems performing deeper analysis.

The Constraint Grammar system works as follows: First, the input string is assigned all possible tags from the lexicon, or rather, from the morphological analyzer. Then, tags are removed iteratively by repeatedly applying a set of rules, or constraints, to the tagged string. When no more tags are removed by the last iteration, the process terminates, and morphological disambiguation is concluded. Then a set of syntactic tags are assigned to the tagged input string and a similar process is performed for syntactic disambiguation. This method is often referred to as *reductionistic tagging*.

The rules are sort-of formulated as finite state automata [Tapanainen, personal communication], which allows very fast processing.

Each rule applies to a current word with a set of candidate tags. The structure of a rule is typically:

> "In the following context, discard the following tags."

or

> "In the following context, commit to the following tag."

We will call discarding or committing to tags the *rule action*. A typical *rule context* is:

> "There is a word to the left that is unambiguously tagged with the following tag, and there are no intervening words tagged with such and such tags."

4.2 The New Approach

The structure of the Constraint Grammar rules readily allows their contexts to be viewed as the conditionings of conditional probabilities, and the actions have an obvious interpretation as the corresponding probabilities.

Each context type can be seen as a separate information source, and we will combine information sources $S_1, \ldots, S_n$ by multiplying the scaled probabilities:

$$\frac{P(T \mid S_1, \ldots, S_n)}{P(T)} \approx \prod_{i=1}^{n} \frac{P(T \mid S_i)}{P(T)}$$

This formula can be established by Bayesian inversion, then performing the independence assumptions, and renewed Bayesian inversion:

$$\begin{aligned} & P(T \mid S_1, \ldots, S_n) = \\ & = \frac{P(T) \cdot P(S_1, \ldots, S_n \mid T)}{P(S_1, \ldots, S_n)} \approx \\ & \approx P(T) \cdot \prod_{i=1}^{n} \frac{P(S_i \mid T)}{P(S_i)} = \\ & = P(T) \cdot \prod_{i=1}^{n} \frac{P(T) \cdot P(S_i \mid T)}{P(T) \cdot P(S_i)} = \\ & = P(T) \cdot \prod_{i=1}^{n} \frac{P(T \mid S_i)}{P(T)} \end{aligned}$$

In standard statistical part-of-speech tagging there are only two information sources — the lexical probabilities and the tags assigned to neighbouring words.

We thus have:

$$\begin{aligned} P(\text{Tag} \mid \text{Lexicon and } n\text{-grams}) &= \\ &= \frac{P(\text{Tag} \mid \text{Lexicon}) \cdot P(\text{Tag} \mid \text{N-grams})}{P(\text{Tag})} \end{aligned}$$

The context will in general not be fully disambiguated. Rather than employing dynamic programming over the lattice of remaining candidate tags, the new approach uses the weighted average over the remaining candidate tags to estimate the probabilities:

$$\begin{aligned} P(T \mid \cup_{i=1}^{n} C_i) &= \\ &= \sum_{i=1}^{n} P(T \mid C_i) \cdot P(C_i \mid \cup_{i=1}^{n} C_i) \end{aligned}$$

It is assumed that $\{C_i : i = 1, \ldots, n\}$ constitutes a partition of the context C, i.e., that $C = \cup_{i=1}^{n} C_i$ and that $C_i \cap C_j = \emptyset$ for $i \neq j$. In particular, trigram probabilities are combined as follows:

$$\begin{aligned} P(T \mid C) &= \\ &= \sum_{(T_l, T_r) \in C} P(T \mid T_l, T_r) \cdot P((T_l, T_r) \mid C) \end{aligned}$$

Here T denotes a candidate tag of the current word, T_l denotes a candidate tag of the immediate left neighbour, and T_r denotes a candidate tag of the immediate right neighbour. C is the set of ordered pairs (T_l, T_r) drawn from the set of candidate tags of the immediate neighbours. $P(T \mid T_l, T_r)$ is the symmetric trigram probability.

The tagger is reductionistic since it repeatedly removes low-probability candidate tags. The probabilities are then recalculated, and the process terminates when the probabilities have stabilized and no more tags can be removed without jeopardizing the recall; candidate tags are only removed if their probabilities are below some threshold value.

5 Sparse Data

Handling sparse data consists of two different tasks:

1. Estimating the probabilities of events that do not occur in the training data.
2. Improving the estimates of conditional probabilities where the number of observations under this conditioning is small.

Coping with unknown words, i.e., words not encountered in the training set, is an archetypical example of the former task. Estimating probability distributions conditional on small contexts is an example of the latter task. We will examine several approaches to these tasks.

For the HMM, it is necessary to avoid zero probabilities. The most straightforward strategy is employing the expected-likelihood estimate (ELE), which simply adds 0.5 to each frequency count and then constructs a maximum-likelihood estimate (MLE), (see e.g. [Gale & Church 1990]). The MLE of the probability is the relative frequency r. Another possibility is the Good-Turing method [Good 1953], where each frequency f is replaced by $f^* = (f+1)N_{f+1}/N_f$, where N_f denotes the frequency of frequency f. Alternatively, one can use linear interpolation of the probabilities obtained by MLE, $\hat{P}(c \mid a, b) = \lambda_1 r(c) + \lambda_2 r(c \mid b) + \lambda_3 r(c \mid a, b)$.

[Brown *et al* 1992] let the λ values dependent on the context, which improves the tagging accuracy. This is related to the idea of successive abstraction presented in Section 5.1. To achieve improved estimates of lexical probabilities, words can be clustered together, see [Cutting *et al* 1992].

There are several ways to handle unknown words. These include:

1. Making every tag a possible tag for that word with equal probability and finding the most probable tag solely based on context probabilities. The results can be slightly improved by trying only open-class tags for unknown words.

2. As an extension to case 1, choosing different but again constant probabilities for each possible tag. This constitutes an a priori distribution for unknown words, reflecting for example that most of the unknown words are nouns. The probabilities could be obtained from a separate training part, or from the distribution of words that occur only once in the training corpus. These words reflect the distribution of unknown words according to the formula presented in [Good 1953].

3. Exploiting word-form information as proposed in [Samuelsson 1994]. Here, the probability distributions are determined from the last n characters of the word, and the remaining number of syllables. This method has been proven successful for Swedish text.

4. Utilizing orthographical cues such as capitalization.

5.1 Successive Abstraction

Assume that we want to estimate the probability $P(E \mid C)$ of the event E given a context C from the number of times N_E it occurs in $N = |C|$ trials, but that this data is sparse. Assume further that there is abundant data in a more general context $C' \supset C$ that we want to use to get a better estimate of $P(E \mid C)$.

If there is an obvious linear order $C = C_m \subset C_{m-1} \subset \cdots \subset C_1 = C'$ of the various generalizations C_k of C, we can build the estimates of $P(E \mid C_k)$ on the relative frequency $r(E \mid C_k)$ of event E in context C_k and the previous estimates of $P(E \mid C_{k-1})$. We call this method *linear successive abstraction*. A simple example is estimating the probability $P(T \mid l_n, \ldots, l_{n-j})$ of a tag T given $l_{n-j}, \ldots, l_n$, the last $j+1$ letters of the word. In this case, the estimate will be based on the relative frequencies $r(T \mid l_n, \ldots, l_{n-j}), r(T \mid l_n, \ldots, l_{n-j+1}), \ldots, r(T \mid l_n), r(T)$.

Previous experiments [Samuelsson 1994] indicate that the following is a suitable formula:

$$\hat{P}(E \mid C) = \frac{\sqrt{N}\, r(E \mid C) + \hat{P}(E \mid C')}{\sqrt{N} + 1} \tag{2}$$

This formula simply up-weights the relative frequency r by a factor $\sqrt{N}$, the square root of the size of context C, which is the active ingredient of the standard deviation of r.

If there is only a partial order of the various generalizations, the scheme is still viable. For example, consider generalizing symmetric trigram statistics, i.e., statistics of the form $P(T \mid T_l, T_r)$. Here, both T_l and T_r are one-step generalizations of the context T_l, T_r, and both have in turn the common generalization Ω. We modify Equation 2 accordingly:

$$\begin{aligned}\hat{P}(T \mid T_l, T_r) &= \\ &= \frac{\sqrt{|T_l, T_r|}\, r(T|T_l, T_r) + \hat{P}(T|T_l) + \hat{P}(T|T_r)}{\sqrt{|T_l, T_r|} + 2}\end{aligned}$$

and

$$\begin{aligned}\hat{P}(T \mid T_l) &= \frac{\sqrt{|T_l|}\, r(T \mid T_l) + \hat{P}(T)}{\sqrt{|T_l|} + 1} \\ \hat{P}(T \mid T_r) &= \frac{\sqrt{|T_r|}\, r(T \mid T_r) + \hat{P}(T)}{\sqrt{|T_r|} + 1}\end{aligned}$$

We call this *partial successive abstraction.*

6 Experiments

For the experiments, both corpora were divided into three sets, one large set and two small sets. We used three different divisions into training and testing sets. First, all three sets were used for both training and testing. In the second and third case, training and test sets were disjoint, the large set and one of the small sets were used for training, the remaining small set was used for testing. As a baseline to indicate what is gained by taking the context into account, we performed an additional set of experiments that used lexical probabilities only, and ignored the context.

6.1 HMM Approach

The experiments of this section were performed with a trigram tagger as described in Section 3. Zero frequencies were avoided by using expected-likelihood estimation. Unknown words were handled by a mixture of methods 2 and 3 listed in Section 5: If the suffix of 4 characters (3 characters for the Susanne corpus) of the unknown words was found in the lexicon, the tag distribution for that suffix was used. Otherwise we used the distribution of tags for words that occurred only once in the training corpus.

As opposed to trigram tagging, lexical tagging ignores context probabilities and is based solely on lexical probabilities. Each word is assigned its most frequent tag from the training corpus. Unknown words were assigned the most frequent tag of words that occurred exactly once in the training corpus. The most frequent tags for single occurrence words are for the Teleman corpus NNSS (indefinite noun-noun compound) and noun (large and small tagset, resp.), for the Susanne corpus NN2 (plural common noun) and NN (common noun; again large and small tagset resp.).

Tagging speed was generally between 1000 and 2000 words per second on a SparcServer 1000; most of this variation was due to variations in the number of unknown words.

The results for the Teleman corpus are shown in Table 5 and the results for the Susanne corpus in Table 6.

What immediately attracts attention is the remarkably low performance of the trigram approach for the Teleman corpus. Already the baseline obtained by lexical tagging is below 80% for new text, usual results are around 90%. Normal results can be obtained only for known words or when using the small tagset, the latter being in fact a very simple task, since the algorithm has to choose from only 19 tags. For the large tagset, trigram tagging achieves only 83% accuracy. This low figure is due to the unusually high number of unknown words and the larger degree of ambiguity compared to English corpora, as is discussed in Section 2. Using a large Swedish lexicon or morphological analyzer should improve the results significantly.

Another interesting result is that accuracy increases when the size of the tagset increases for the cases where known text is tagged and context probabilities are taken into account. This means that the additional information about the context in the larger tagset is very helpful for disambiguation, but only when disambiguating

Table 5: Results of the HMM experiments with the Teleman corpus

		Training	Testing	total correct	known correct	unknown correct
Small Tagset	19 Tags	Lexical Tagging				
		A, B, C	A, B, C	95.13%	95.13%	—
		A, B	C	89.27%	94.18%	58.35%
		A, C	B	90.42%	94.20%	69.60%
		Trigram Tagging				
		A, B, C	A, B, C	96.22%	96.22%	—
		A, B	C	92.88%	94.51%	82.55%
		A, C	B	92.81%	94.62%	82.83%
Large Tagset	258 Tags	Lexical Tagging				
		A, B, C	A, B, C	90.65%	90.65%	—
		A, B	C	78.84%	89.07%	14.44%
		A, C	B	78.05%	88.20%	22.03%
		Trigram Tagging				
		A, B, C	A, B, C	98.35%	98.35%	—
		A, B	C	83.78%	89.99%	44.66%
		A, C	B	81.01%	89.40%	34.69%

Table 6: Results of the HMM experiments with the Susanne corpus

		Training	Testing	total correct	known correct	unknown correct
Small Tagset	62 Tags	Lexical Tagging				
		A, B, C	A, B, C	95.28%	95.28%	—
		A, B	C	91.48%	94.80%	49.72%
		A, C	B	91.20%	94.44%	38.37%
		Trigram Tagging				
		A, B, C	A, B, C	98.65%	98.65%	—
		A, B	C	95.76%	96.95%	80.81%
		A, C	B	95.18%	96.58%	72.29%
Large Tagset	424 Tags	Lexical Tagging				
		A, B, C	A, B, C	93.98%	93.98%	—
		A, B	C	86.98%	93.04%	10.78%
		A, C	B	88.16%	92.59%	15.81%
		Trigram Tagging				
		A, B, C	A, B, C	99.80%	99.80%	—
		A, B	C	92.61%	95.66%	54.20%
		A, C	B	93.07%	95.46%	53.83%

known text. This could arise from the fact that a large number (> 50%) of the trigrams that occur in the training text occur exactly once. And most of the possible trigrams do not occur at all (generally more than 90%, depending on the size of the tagset). Now, the trigram approach has a distinct bias to those trigrams that occurred once over those that never occurred. These happen to be the right ones for known text but not necessarily for new text, thus the positive effect of a larger tagset vanishes for fresh text.

The results for the Susanne corpus are similar to those reported in other publications for (other) English corpora.

6.2 Reductionistic Approach

The reductionistic statistical tagger described in Section 4 was tested on the same data as the HMM tagger. The information sources employed in the experiments were lexical statistics and contextual information, which consisted of symmetric trigram statistics. Unknown words were handled by creating a decision tree of the four last letters from words with three or less occurrences. Each node in the tree was associated with a probability distribution (over the tagset) extracted from these words, and the probabilities were smoothened through linear successive abstraction, see Section 5.1.

There were two cut-off values for contexts: Firstly, any context with less than 10 observations was discarded. Secondly, any context where the probability distributions did not differ substantially from the unconditional one was also discarded. Only the remaining ones were used for disambiguation. Due to the computational model employed, omitted contexts are equivalent to backing off to whatever the current probability distribution is. The distributions conditional on contexts are however susceptible to the problem of sparse data. This was handled using partial successive abstraction as described in Section 5.1.

The results are shown in Tables 7 and 8. They clearly indicate that:

- The employed treatment of unknown words is quite effective.
- Using contextual information, i.e., trigrams, improves tagging accuracy.
- The performance is on pair with the HMM tagger and comparable to state-of-the-art statistical part-of-speech taggers.
- Teleman is a considerably tougher nut to crack than Susanne.

The results using the Susanne corpus are similar to those reported for the Lancaster-Oslo-Bergen (LOB) corpus in [de Marcken 1990], where a statistical n-best-path approach was employed to trade precision for recall.

The tagging speed was typically a couple of hundred words per second on a SparcServer 1000, but varied with the size of the tagset and the amount of remaining ambiguity.

7 Conclusions

The experiments with the HMM approach show that it is much harder to process the Swedish than the English corpus. Although the two corpora are not fully comparable because of the differences in size and tagsets used, they reveal a strong tendency. The difficulty in processing is mostly due to the rather large number of unknown words in the Swedish corpus and the higher degree of ambiguity despite having smaller tagsets. These effects mainly arise from the higher morphological variation

Table 7: Results of the reductionistic experiments with the Teleman corpus

Training	Testing	Threshold:	0.00	0.05	0.075	0.10	0.15	0.20	0.30	0.50
		Small Tagset								
		Trigram and lexical statistics								
A,B,C	A,B,C	Recall (%)	100.00	99.02	98.66	98.35	97.78	97.37	96.65	95.55
		Tags/word	2.38	1.15	1.12	1.10	1.07	1.05	1.03	1.00
		Lexical statistics only								
		Recall (%)	100.00	98.96	98.53	98.29	97.69	97.28	96.36	95.10
		Tags/word	2.38	1.25	1.17	1.14	1.09	1.07	1.03	1.00
		Trigram and lexical statistics								
A,B	C	Recall (%)	98.98	97.72	97.25	96.81	96.20	95.53	94.67	93.34
		Tags/word	2.54	1.21	1.17	1.14	1.10	1.07	1.04	1.00
		Lexical statistics only								
		Recall (%)	98.98	97.61	97.14	96.87	96.15	95.63	94.26	92.55
		Tags/word	2.54	1.34	1.25	1.21	1.14	1.11	1.04	1.00
		Trigram and lexical statistics								
A,C	B	Recall (%)	98.99	97.80	97.44	96.94	96.34	95.84	98.81	93.50
		Tags/word	2.51	1.23	1.18	1.15	1.11	1.08	1.04	1.00
		Lexical statistics only								
		Recall (%)	98.99	97.67	97.33	97.07	96.45	95.84	94.34	92.52
		Tags/word	2.51	1.34	1.26	1.21	1.14	1.10	1.04	1.00
		Large Tagset								
		Trigram and lexical statistics								
A,B,C	A,B,C	Recall (%)	100.00	98.36	97.92	97.54	97.03	96.41	95.31	93.75
		Tags/word	3.69	1.23	1.18	1.15	1.11	1.08	1.04	1.00
		Lexical statistics only								
		Recall (%)	100.00	98.30	97.63	97.20	96.67	95.57	93.65	90.59
		Tags/word	3.69	1.43	1.31	1.26	1.22	1.16	1.08	1.00
		Trigram and lexical statistics								
A,B	C	Recall (%)	97.46	94.93	93.94	93.35	92.35	91.15	88.53	85.56
		Tags/word	4.16	1.47	1.37	1.31	1.24	1.18	1.08	1.00
		Lexical statistics only								
		Recall (%)	97.46	95.23	94.24	93.69	92.93	91.51	87.92	83.62
		Tags/word	4.16	1.69	1.53	1.44	1.34	1.26	1.11	1.00
		Trigram and lexical statistics								
A,C	B	Recall (%)	96.64	94.04	93.00	92.09	90.92	89.46	86.94	83.58
		Tags/word	4.18	1.48	1.38	1.32	1.24	1.18	1.08	1.00
		Lexical statistics only								
		Recall (%)	96.64	94.51	93.27	92.50	91.02	89.68	85.86	81.69
		Tags/word	4.18	1.71	1.54	1.44	1.34	1.24	1.10	1.00

Table 8: Results of the reductionistic experiments with the Susanne corpus

Training	Testing	Threshold:	0.00	0.05	0.075	0.10	0.15	0.20	0.30	0.50
Small Tagset										
A,B,C	A,B,C	Trigram and lexical statistics								
		Recall (%)	100.00	99.46	99.35	99.23	99.03	98.82	98.43	97.75
		Tags/word	2.07	1.08	1.07	1.06	1.04	1.03	1.02	1.00
		Lexical statistics only								
		Recall (%)	100.00	99.33	99.20	98.94	98.67	98.10	97.43	95.28
		Tags/word	2.07	1.18	1.16	1.14	1.11	1.08	1.05	1.00
A,B	C	Trigram and lexical statistics								
		Recall (%)	99.22	98.43	98.28	98.11	97.78	97.43	96.91	95.99
		Tags/word	2.23	1.14	1.11	1.09	1.07	1.05	1.02	1.00
		Lexical statistics only								
		Recall (%)	99.22	98.27	98.03	97.78	97.45	96.80	96.15	93.42
		Tags/word	2.23	1.25	1.23	1.19	1.15	1.11	1.08	1.00
A,C	B	Trigram and lexical statistics								
		Recall (%)	99.22	98.46	98.22	97.99	97.58	97.15	96.49	95.54
		Tags/word	2.17	1.13	1.10	1.09	1.06	1.05	1.02	1.00
		Lexical statistics only								
		Recall (%)	99.22	98.21	97.88	97.61	97.35	96.47	95.46	92.87
		Tags/word	2.17	1.24	1.21	1.17	1.15	1.10	1.06	1.00
Large Tagset										
A,B,C	A,B,C	Trigram and lexical statistics								
		Recall (%)	100.00	99.25	99.12	98.96	98.74	98.44	98.04	96.87
		Tags/word	2.61	1.10	1.08	1.07	1.06	1.04	1.03	1.00
		Lexical statistics only								
		Recall (%)	100.00	99.05	98.88	98.59	98.20	97.58	96.72	93.98
		Tags/word	2.61	1.23	1.20	1.17	1.14	1.10	1.07	1.00
A,B	C	Trigram and lexical statistics								
		Recall (%)	98.31	96.94	96.52	96.19	95.68	95.02	94.21	92.70
		Tags/word	3.01	1.22	1.18	1.15	1.11	1.08	1.04	1.00
		Lexical statistics only								
		Recall (%)	98.31	96.91	96.49	95.94	95.50	94.40	93.42	90.26
		Tags/word	3.01	1.41	1.35	1.28	1.20	1.14	1.08	1.00
A,C	B	Trigram and lexical statistics								
		Recall (%)	98.49	97.03	96.72	96.41	95.88	95.16	94.29	92.71
		Tags/word	2.83	1.21	1.18	1.15	1.11	1.08	1.04	1.00
		Lexical statistics only								
		Recall (%)	98.49	96.95	96.55	96.05	95.57	94.44	93.26	90.31
		Tags/word	2.83	1.36	1.31	1.25	1.19	1.13	1.08	1.00

of Swedish which calls for additional strategies to be applied. These could be the use of a large corpus-independent lexicon and a separate morphological analysis.

It is reassuring to see that the reductionistic tagger performs as well as the HMM tagger, indicating that the new framework is as powerful as the conventional one when using strictly conventional information sources. The new framework also enables using the same sort of information as the highly successful Constraint Grammar approach, and the hope is that the addition of further information sources can advance state-of-the-art performance of statistical taggers.

Viewed as an extension of the Constraint Grammar approach, the new scheme allows making decisions on the basis of not fully disambiguated portions of the input string. The absolute value of the probability of each tag can be used as a quantitative measure of when to remove a particular candidate tag and when to leave in the ambiguity. This provides a tool to control the tradeoff between recall (accuracy) and precision (remaining ambiguity).

Acknowledgements

We wish to thank Björn Gambäck for providing information on previous work with the Teleman corpus.

References

[Baum 1972]
L. E. Baum. "An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes", *Inequalities III*, pp. 1-8, 1972.

[Brown *et al* 1992]
P. F. Brown, V. J. Della Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer and P. S. Roossin. "Class-based *n*-gram models of natural language", *Computational Linguistics 18(4)* pp. 467–479, 1992.

[Cutting 1994]
Douglass Cutting. "A Practical Part-of-Speech Tagger", in *Procs. 9th Scandinavian Conference on Computational Linguistics*, pp. 65–70, Stockholm University, 1994.

[Cutting *et al* 1992] Douglass R. Cutting, Julian Kupiec, Jan Pedersen and Penelope Sibun. "A Practical Part-of-Speech Tagger". in *Procs. 3rd Conference on Applied Natural Language Processing*, pp. 133–140, ACL, 1992.

[Eineborg & Gambäck 1994]
Martin Eineborg and Björn Gambäck. "Tagging Experiments Using Neural Networks", in *Procs. 9th Scandinavian Conference on Computational Linguistics*, pp. 71–82, Stockholm University, 1994.

[Francis & Kucera 1982]
N. W. Francis and H. Kucera. *Frequency Analysis of English Usage*, Houghton Mifflin, Boston, 1982.

[Gale & Church 1990]
W. A. Gale and K. W. Church. "Poor Estimates of Context are Worse than None", in *Proc. of the Speech and Natural Language Workshop*, pp. 283-287, Morgan Kaufmann, 1990.

[Good 1953]
I. J. Good. "The population frequencies of species and the estimation of population parameters", *Biometrika 40*, pp. 237–264, 1953.

[Karlsson *et al* 1995]
Fred Karlsson, Atro Voutilainen, Juha Heikkilä and Arto Anttila (eds). *Constraint Grammar. A Language-Independent System for Parsing Unrestricted Text*, Mouton de Gruyter, Berlin / New York, 1995.

[de Marcken 1990]
Carl G. de Marcken. "Parsing the LOB Corpus", in *Procs. 28th Annual Meeting of the Association for Computational Linguistics*, pp. 243–251, ACL 1990.

[Rabiner 1989]
L. R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition", in *Proceedings of the IEEE 77(2)*, pp. 257–285, 1989.

[Sampson 1995]
Geoffrey Sampson. *English for the Computer*, Oxford University Press, Oxford, 1995.

[Samuelsson 1994]
Christer Samuelsson. "Morphological Tagging Based Entirely on Bayesian Inference", in *Procs. 9th Scandinavian Conference on Computational Linguistics*, pp. 225–238, Stockholm University, 1994.

[Teleman 1974]
Ulf Teleman. *Manual för grammatisk beskrivning av talad och skriven svenska*, (in Swedish), Studentlitteratur, Lund, Sweden 1974.

[Viterbi 1967]
A. Viterbi. "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm", in *IEEE Transactions on Information Theory*, pp. 260–269, 1967.

[Voutilainen & Heikkilä 1994]
Atro Voutilainen and Juha Heikkilä. "An English constraint grammar (ENGCG): a surface-syntactic parser of English", in *Procs. 14th International Conference on English Language Research on Computerized Corpora*, pp. 189–199, Zürich, 1994.

Indexed Languages and Unification Grammars*

Tore Burheim†

Abstract

Indexed languages are interesting in computational linguistics because they are the least class of languages in the Chomsky hierarchy that has not been shown not to be adequate to describe the string set of natural language sentences. We here define a class of unification grammars that exactly describe the class of indexed languages.

1 Introduction

The occurrence of purely syntactical cross-serial dependencies in Swiss-German shows that context-free grammars can not describe the string sets of natural language [Shi85]. The least class in the Chomsky hierarchy that can describe unlimited cross-serial dependencies is indexed grammars [Aho68]. Gazdar discuss in [Gaz88] the applicability of indexed grammars to natural languages, and show how they can be used to describe different syntactic structures. We are here going to study how we can describe the class of indexed languages with a unification grammar formalism. After defining indexed grammars and a simple unification grammar framework we show how we can define an equivalent unification grammar for any given indexed grammar. Two grammars are equivalent if they generate the same language. With this background we define a class of unification grammars and show that this class describes the class of indexed languages.

2 Indexed grammars

Indexed grammars is a grammar formalism with generative capacity between context-free grammars and context-sensitive grammars. Context-free grammars can not describe cross-serial dependencies due to the pumping lemma, while indexed grammars can. However, the class of languages generated by indexed grammars, –the indexed languages, is a proper subset of context-sensitive languages [Aho68].

Indexed grammars can be seen as a context-free grammar where we add a string –or stack, of indices to the nonterminal nodes in the phrase structure trees, or derivation trees as we will call them. Some production rules add an index to the beginning of the string, while the use of other production rules is dependent on the first index in the string. When such a production rule is applied the index of which it is dependent, is removed, and the rest of the index-string is kept by the daughter(s). In this way we may distribute information from one part of the derivation tree to another. The original definition of indexed grammars was given

*This work has been supported by grant 100437/410 from Norwegian Research Council.

†University of Bergen, Department of Informatics, N-5020 Bergen, Norway *and* University of the Saarland, Computational Linguistic, Postfach 1150, D-66041 Saarbrücken, Germany. Email: Tore.Burheim@ii.uib.no

by Aho [Aho68]. We are here using the definition used by Hopcroft and Ullman [HU79] with some minor notational variations:

Definition 1 *An* INDEXED GRAMMAR G *is a 5-tuple;* $G = \langle N, T, I, P, S\rangle$ *where*

N *is a finite set of symbols, called* nonterminals,

T *is a finite set of symbols, called* terminals,

I *is a finite set of symbols, called* indices,

P *is a finite set of ordered pairs, each on one of the forms* $\langle A, Bf\rangle$, $\langle Af, \alpha\rangle$ *or* $\langle A, \alpha\rangle$ *where* A *and* B *are nonterminal symbols in* N, α *is a finite string in* $(N \cup T)^*$, *and* f *is an index in* I. *An element in* P *is called a* production rule *and is written* $A \rightarrow Bf$, $Af \rightarrow \alpha$ *or* $A \rightarrow \alpha$.

S *is a symbol in* N, *and is called the* start symbol.

and such that N, T *and* I *are pairwise disjoint.*

An indexed grammar $G = \langle N, T, I, P, S\rangle$ *is on* REDUCED FORM *if each production in* P *is on one of the forms*

a) $A \rightarrow Bf$

b) $Af \rightarrow B$

c) $A \rightarrow BC$

d) $A \rightarrow t$

where A, B, C *are in* N, f *is in* I, *and* t *is in* $(T \cup \{\varepsilon\})$.

Aho showed in his original paper [Aho68] that for every indexed grammar there exists an indexed grammar on reduced form which generates the same language.

To define constituent structures and derivation trees we are going to use tree domains: Let $\mathcal{N}_+$ be the set of all integers greater than zero. A *tree domain* D is a set $D \subseteq \mathcal{N}_+^*$ of number strings so that if $x \in D$ then all prefixes of x are also in D, and for all $i \in \mathcal{N}_+$ and $x \in \mathcal{N}_+^*$, if $xi \in D$ then $xj \in D$ for all j, $1 \leq j < i$. The *out degree* $d(x)$ of an element x in a tree domain D is the cardinality of the set $\{i \mid xi \in D, i \in \mathcal{N}_+\}$. The set of terminals of D is $term(D) = \{x \mid x \in D, d(x) = 0\}$. The elements of a tree domain are totally ordered lexicographically as follows: $x \preceq y$ if x is a prefix of y, or there exist strings $z, z', z'' \in \mathcal{N}_+^*$ and $i, j \in \mathcal{N}_+$ with $i < j$, such that $x = ziz'$ and $y = zjz''$. We also define that $x \prec y$ if $x \preceq y$ and $x \neq y$.[1]

A tree domain D can be viewed as a tree graph in the following way: The elements of D are the nodes in the tree, ε is the root, and for every $x \in D$ the element $xi \in D$ is x's child number i. A tree domain may be infinite, but we shall restrict attention to finite tree domains. A finite tree domain can also describe the topology of a derivation tree. This representation provides a name for every node in the derivation tree directly from the definition of a tree domain. Our definition of derivation trees for indexed grammars with the use of tree domains is based on Hayashi [Hay73]:

Definition 2 *A* DERIVATION TREE *based on an indexed grammar* $G = \langle N, T, I, P, S\rangle$ *is a pair* $\langle D, C_{\mathcal{I}}\rangle$ *of a finite tree domain* D *and a function* $C_{\mathcal{I}} : D \rightarrow (NI^* \cup T \cup \{\varepsilon\})$ *where*

i) $C_{\mathcal{I}}(\varepsilon) = S$

[1]See Gallier [Gal86] for more about tree domains.

ii) *$C_{\mathcal{I}}(x) \in NI^*$ for every node x in D with $d(x) > 0$. Moreover if $C_{\mathcal{I}}(x) = A\gamma$ for $A \in N$ and $\gamma \in I^*$ and $C_{\mathcal{I}}(xi) = B_i\theta_i$ with $B_i \in (N \cup T \cup \{\varepsilon\})$ and $\theta_i \in I^*$ for every $i : 1 \leq i \leq d(x)$ then either*

a) *$A \rightarrow B_1 f$ is a production rule in P such that $d(x) = 1$, $f \in I$, and $\theta_1 = f\gamma$, or*

b) *$Af \rightarrow B_1 \ldots B_{d(x)}$ is a production rule in P such that $f \in I$ where $\gamma = f\gamma'$, and $\theta_i = \gamma'$ if $B_i \in N$ and $\theta_i = \varepsilon$ if $B_i \in (T \cup \{\varepsilon\})$, or*

c) *$A \rightarrow B_1 \ldots B_{d(x)}$ is a production rule in P such that $\theta_i = \gamma$ if $B_i \in N$ and $\theta_i = \varepsilon$ if $B_i \in (T \cup \{\varepsilon\})$.*

iii) *$C_{\mathcal{I}}(x) \in (T \cup \{\varepsilon\})$ for every node in D with $d(x) = 0$,*

The SYMBOL FUNCTION*; $C_{\mathcal{I}}^{sym} : D \rightarrow (N \cup T)$, and the* INDEX STRING FUNCTION*; $C_{\mathcal{I}}^{idx} : D \rightarrow I^*$, are total functions on D such that if $C_{\mathcal{I}}(x) = A\gamma$ where $A \in (N \cup T \cup \{\varepsilon\})$ and $\gamma \in I^*$ then $C_{\mathcal{I}}^{sym}(x) = A$ and $C_{\mathcal{I}}^{idx}(x) = \gamma$ for all $x \in D$.*

The TERMINAL STRING *of a derivation tree $\langle D, C_{\mathcal{I}} \rangle$ is the string $C_{\mathcal{I}}(x_1)...C_{\mathcal{I}}(x_n)$ where $\{x_1, ..., x_n\} = term(D)$ and $x_i \prec x_{i+1}$ for all $i, 1 \leq i \leq n-1$.*

We also define the LICENSE FUNCTION*; $license : (D - term(D)) \rightarrow P$, such that if $A \rightarrow \alpha$ is a production rule according to ii) a), b) or c) for a node x in D, then $license(x) = A \rightarrow \alpha$.*

Informally this is a traditional derivation tree. If we have a node with label $A\gamma$ where A is a nonterminal symbol and γ is a string of indices, and we use a production rule $A \rightarrow Bf$, then the node's only child gets the label $Bf\gamma$. If we instead use a production rule $A \rightarrow BC$ on the same node it gets two children labeled $B\gamma$ and $C\gamma$ respectively, or if we use a production rule $A \rightarrow t$ where t is a terminal symbol, then we remove all the indices and the node's only child gets the label t. If we have a node labeled with $Af\gamma$, where f is a index and we use a production rule $Af \rightarrow B$ then the node's only child gets the label $B\gamma$. We also see that the terminal string is a string in T^* since $C_{\mathcal{U}}(x) \in (T \cup \{\varepsilon\})$ for all $x \in term(D)$.

Definition 3 *A string w is* GRAMMATICAL *with respect to an indexed grammar G if and only if there exists a derivation tree based on G with w as the terminal string. The language generated by G, $L(G)$ is the set of all grammatical strings with respect to G.*

Example 1 Let $G = \langle N, T, I, P, S \rangle$ be an indexed grammar where $T = \{a, b, c\}$ is the set of terminal symbols, $N = \{S, S', A, B, C\}$ is the set of nonterminal symbols, $I = \{f, g\}$ is the set of indices and P is the least set containing the following production rules:

$$\begin{array}{lll} S \rightarrow S'f & Ag \rightarrow aA & Af \rightarrow a \\ S' \rightarrow S'g & Bg \rightarrow bB & Bf \rightarrow b \\ S' \rightarrow ABC & Cg \rightarrow cC & Cf \rightarrow c \end{array}$$

Figure 1 shows the derivation tree for the string *"aabbcc"* based on this grammar. The language $L(G)$ generated by this grammar is $\{a^n b^n c^n \mid n \geq 1\}$.

We close this presentation of indexed grammars by showing a simple technical observation that we will use in later proofs.

Definition 4 *An indexed grammar $G = \langle N, T, I, P, S \rangle$ has a* MARKED INDEX-END *if and only if it has one and only one production rule where the start symbol occurs and this rule is on the form $S \rightarrow A\$$ where $A \in N$ and the index \$ does not occur in any other production rule.*

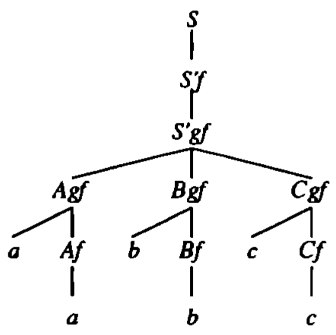

Figure 1: *Derivation tree for the string "aabbcc" based on the grammar in Example 1*

If an indexed grammar has a marked index-end then in any derivation tree every nonterminal node except the root gets a \$ at the end of the index list. Since no rule requires that there is an empty index list, and neither \$ nor the start symbol occurs in any other production rule, it is straight forward to construct an equivalent grammar with a marked index-end for any indexed grammar.

Lemma 1 *For every indexed grammar G there exists an indexed grammar with a marked index-end $G_\$$ such that $L(G) = L(G_\$)$.*

Proof: Let $G = \langle N, T, I, P, S\rangle$ be an indexed grammar, and assume that S_0 and \$ do not occur in G. $G_\$$ is defined from G by adding the production rule $S_0 \rightarrow S\$$ such that S_0 becomes the new start symbol and is added to the set of nonterminal symbols, and \$ is added to the set of indices. Formally, if $G = \langle N, T, I, P, S\rangle$ and $S_0, \$ \notin (N \cup T \cup I)$, then $G_\$ = \langle N \cup \{S_0\}, T, I \cup \{\$\}, P \cup \{\langle S_0, S\$\rangle\}, S_0\rangle$. Then $G_\$$ has a marked index-end, and we have to show that for any string w, $w \in L(G)$ if and only if $w \in L(G_\$)$.

($\Longrightarrow$) Let $\langle D, C_\mathcal{I}\rangle$ be any derivation tree based on G and assume that w is its terminal string. From this we construct a derivation tree $\langle D', C'_\mathcal{I}\rangle$ based on $G_\$$ as follows: First let $D' = \{1x \mid x \in D\} \cup \{\varepsilon\}$. Then let $C'_\mathcal{I}(\varepsilon) = S_0$ and let $C'_\mathcal{I}(1x) = C_\mathcal{I}(x)\$$ for all $x \in (D - \mathit{term}(D))$. Let also $C'_\mathcal{I}(1x) = C_\mathcal{I}(x)$ for all $x \in \mathit{term}(D)$. The derivation tree $\langle D', C'_\mathcal{I}\rangle$ has then the same terminal string as $\langle D, C_\mathcal{I}\rangle$. Since no rule requires that there is an empty index list, and \$ does not occur in any production rule in G, a production rule that is licensing a node x in $\langle D, C_\mathcal{I}\rangle$, will license the node $1x$ in $\langle D', C'_\mathcal{I}\rangle$. The rule $S_0 \rightarrow S\$$ licenses the root. Then $\langle D', C'_\mathcal{I}\rangle$ is a valid derivation tree according to Definition 2.

($\Longleftarrow$) Let $\langle D', C'_\mathcal{I}\rangle$ be any derivation tree based on $G_\$$ and assume that w is its terminal string. Since $S_0 \rightarrow S\$$ must license the root and \$ does not occur in any other production rule the index symbol \$ occurs at the end of the index list at every nonterminal node except the root in $\langle D', C'_\mathcal{I}\rangle$. From this derivation tree we construct a derivation tree $\langle D, C_\mathcal{I}\rangle$ based on G as follows: First let $D = \{x \mid 1x \in D'\}$. Then for all $x \in (D - \mathit{term}(D))$ let $C_\mathcal{I}(x) = \beta$ where $C'_\mathcal{I}(1x) = \beta\$$. Let also $C_\mathcal{I}(x) = C'_\mathcal{I}(1x)$ for all $x \in \mathit{term}(D)$. The derivation tree $\langle D, C_\mathcal{I}\rangle$ has then the same terminal string as $\langle D', C'_\mathcal{I}\rangle$. Since every production rule in $G_\$$ except $S_0 \rightarrow S\$$ also is a production rule in G, the rule $S_0 \rightarrow S\$$ only can license the root, and \$ does not occur in any other production rule, a production rule that licenses a node $1x$ in $\langle D', C'_\mathcal{I}\rangle$ will license the node x in $\langle D, C_\mathcal{I}\rangle$. Then $\langle D, C_\mathcal{I}\rangle$ is a valid derivation tree according to Definition 2. □

Notice in the proof that if G is on reduced form then $G_\$$ is also on reduced form. Then for any indexed grammar on reduced form there also exists an indexed grammar on reduced form with a marked index-end.

3 Unification grammars

We are here going to give a description of a very simple unification grammar formalism. The formalism itself is not particularly interesting, and it is only meant as a framework for the rest of this paper. The formalism is just a notational variant of the basic formalism used by Colban in his work on restrictions on unification grammars [Col91]. It should be easy to reformulate this in most of the known formalisms available. We give an informal description of feature structures in the way they are used here before we define the grammar formalism.

A *feature structure* over a set of attribute symbols A and value symbols V is a four-tuple $\langle Q, \delta, \alpha, m_D \rangle$ where Q is a finite set of nodes, $\delta : Q \times A \rightarrow Q$ is a partial function, called the transition function, $\alpha : Q \rightarrow V$ is a partial function called the atomic value function, and $m_D : D \rightarrow Q$ is a function, called the name mapping. We will mostly omit the name-domain from the notation, so m will alone denote the name mapping. We extend the transition function to be a function from pairs of nodes and *strings* of attribute symbols: For every $q \in Q$ let $\delta(q, \varepsilon) = q$. If $\delta(q_1, \psi) = q_2$ and $\delta(q_2, a) = q_3$ then let $\delta(q_1, \psi a) = q_3$ for every $q_1, q_2, q_3 \in Q$, $\psi \in A^*$ and $a \in A$.

A feature structure is *describable* if there for every node is a path from a named node to the node. This means that for every $q \in Q$ there is an $x \in D$ and a $\psi \in A^*$ such that $\delta(m(x), \psi) = q$. A feature structure is *atomic* if every node with an atomic value has no out-edges. This means that for every node $q \in Q$, $\delta(q, a)$ is not defined for any $a \in A$ if $\alpha(q)$ is defined. A feature structure is *acyclic* if it does not contain attribute cycles. This means that for every node $q \in Q$, $\delta(q, \psi) = q$ if and only if $\psi = \varepsilon$. A feature structure is *well defined* if it is describable, atomic and acyclic. When nothing else is said we require that feature structures are well defined in the rest of this paper.

We are going to use equations to describe feature structures, in a way where feature structure satisfies equations. A feature structure *satisfies* the equation

$$x_1\psi_1 \doteq x_2\psi_2 \tag{1}$$

if and only if $\delta(m(x_1), \psi_1) = \delta(m(x_2), \psi_2)$, and the equation

$$x_1\psi_1 \doteq v \tag{2}$$

if and only if $\alpha(\delta(m(x_1), \psi_1)) = v$, where $x_1, x_2 \in D$, $\psi_1, \psi_2 \in A^*$ and $v \in V$. We only allow equations on those two forms. This means that there is no typing, quantification, implication, negation, or explicit disjunction as we may find in other unification grammars and feature logics.

If E is a set of equations of the above form and M is a well defined feature structure such that M satisfies every equation in E then we say that M *satisfies* E and we write

$$M \models E \tag{3}$$

A set of equations E is *consistent* if there exists a well defined feature structure that satisfies E.

The notation of the grammar formalism is borrowed from Lexical Functional Grammar [KB82].

Definition 5 *A* SIMPLE UNIFICATION GRAMMAR G *over a set of attribute symbols A and value symbols V is a 5-tuple $\langle N, T, P, L, S \rangle$ where*

N is a finite set of symbols, called nonterminals,

T is a finite set of symbols, called terminals,

P is a finite set of production rules

$$\begin{array}{ccccc} A_0 & \longrightarrow & A_1 & \dots & A_n \\ & & E_1 & & E_n \end{array} \tag{4}$$

where $n \geq 1$, $A_0, ..., A_n \in N$, and for all i, $1 \leq i \leq n$, E_i is a finite set with equations on the forms

$$\uparrow\mid\downarrow \psi \doteq \uparrow\mid\downarrow \psi' \tag{5}$$

$$\uparrow\mid\downarrow \psi'' \doteq v \tag{6}$$

where $\psi, \psi' \in A^*$, $\psi'' \in A^+$ and $v \in V$.[2]

L is a finite set of lexicon rules

$$\begin{array}{ccc} A & \longrightarrow & t \\ & & E \end{array} \tag{7}$$

where $A \in N$, $t \in (T \cup \{\varepsilon\})$, and E is a finite set of equations on the form

$$\uparrow\mid\downarrow \psi'' \doteq v \tag{8}$$

where $\psi'' \in A^+$ and $v \in V$.

S is a symbol in N, called start symbol.

As an example (9) is a production rule.

$$\begin{array}{ccccc} A & \longrightarrow & B & C & C \\ & & \uparrow \doteq \downarrow & \uparrow \doteq \downarrow a_1 & \downarrow \doteq \downarrow a_3\, a_4 \\ & & \uparrow c \doteq v_1 & \uparrow a_2\, a_3 \doteq \downarrow a_1 & \uparrow a_3 \doteq v_2 \end{array} \tag{9}$$

Definition 6 *A* CONSTITUENT STRUCTURE *(c-structure) based on a simple unification grammar $G = \langle N, T, P, L, S \rangle$ is a triple $\langle D, C_{\mathcal{U}}, E_{\mathcal{U}} \rangle$ where*

D is a finite tree domain,

$C_{\mathcal{U}} : D \longrightarrow (N \cup T \cup \{\varepsilon\})$ is a function,

$E_{\mathcal{U}} : (D - \{\varepsilon\}) \longrightarrow \Gamma$ is a function where Γ is the set of all equation sets in P and L,

such that $C_{\mathcal{U}}(x) \in (T \cup \{\varepsilon\})$ for all $x \in term(D)$, $C_{\mathcal{U}}(\varepsilon) = S$, and for all $x \in (D - term(D))$, if $d(x) = n$ then

$$\begin{array}{ccccc} C_{\mathcal{U}}(x) & \longrightarrow & C_{\mathcal{U}}(x1) & \dots & C_{\mathcal{U}}(xn) \\ & & E_{\mathcal{U}}(x1) & & E_{\mathcal{U}}(xn) \end{array} \tag{10}$$

is a production or lexicon rule in G.

The TERMINAL STRING *of a constituent structure is the string $C_{\mathcal{U}}(x_1)...C_{\mathcal{U}}(x_n)$ where $\{x_1, ..., x_n\} = term(D)$ and $x_i \prec x_{i+1}$ for all i, $1 \leq i < n$.*

To get equations that can be satisfied by a feature structure we must instantiate the up and down arrows in the equations from the rule set. We substitute them with nodes from the c-structure such that the nodes become the domain of the name mapping. For this purpose we define the $'$-function such that $E'_{\mathcal{U}}(xi) = E_{\mathcal{U}}(xi)[x/\uparrow, xi/\downarrow]$. We see that the value of the function $E'_{\mathcal{U}}$ is a set of equations that feature structures may satisfy.

[2] $\uparrow\mid\downarrow$ denotes here a $\uparrow$ or a $\downarrow$

Definition 7 *The c-structure $\langle D, K, E\rangle$* GENERATES *the feature structure M if and only if*

$$M \models \bigcup_{x \in D} E'_{\mathcal{U}}(x) \tag{11}$$

A c-structure may generate different feature structures. The tree domain will form a name set for feature structures that this union generates. A string is grammatical if this union is consistent.

Definition 8 *A string w is* GRAMMATICAL *with respect to a simple unification grammar G if and only if there exists a c-structure based on G with w as the terminal string and which generates a well defined feature structure. The language generated by G, $L(G)$ is the set of all grammatical strings with respect to G.*

4 From Indexed Grammars to Unification Grammars

We are here going to define a simple unification grammar that is equivalent to a given indexed grammar. The main idea is that we use feature structures to represent the index string more or less like a (nested) stack. The use of feature structures to represent stacks for indexed grammars is also used by Gazdar and Mellish [GM89] although they do not go into much details. Here we define a function that transforms any indexed grammar on reduced form with a marked index-end to a simple unification grammar, such that the new grammar generates the same language.

Definition 9 *Let $G_\$ = \langle N, T, I, P, S\rangle$ be an indexed grammar on reduced form with a marked index-end. We then define the simple unification grammar $\mathcal{U}(G_\$)$ as $\langle N, T, P', L', S\rangle$ where P' and L' are the least sets where*

a) *For each rule on the form $A \rightarrow Bf$ in P, P' has a production rule on the form*

$$A \;\rightarrow\; \begin{array}{c} B \\ \downarrow next \doteq \uparrow \\ \downarrow idx \doteq f \end{array} \tag{12}$$

b) *For each rule on the form $Af \rightarrow B$ in P, P' has a production rule on the form*

$$A \;\rightarrow\; \begin{array}{c} B \\ \uparrow next \doteq \downarrow \\ \uparrow idx \doteq f \end{array} \tag{13}$$

c) *For each rule on the form $A \rightarrow BC$ in P, P' has a production rule on the form*

$$A \;\rightarrow\; \begin{array}{cc} B & C \\ \uparrow \doteq \downarrow & \uparrow \doteq \downarrow \end{array} \tag{14}$$

d) For each rule on the form $A \longrightarrow a$ in P, L' has a lexicon rule on the form

$$\begin{array}{ccc} A & \longrightarrow & a \\ & & \emptyset \end{array} \tag{15}$$

If p is a production rule in $G_\$$ then $\mathcal{U}(p)$ is the production or lexicon rule in $\mathcal{U}(G_\$)$ defined by a), b) c) or d).

Notice that there is a one-to-one relation between the production rules in $G_\$$, and production/lexicon-rules in $\mathcal{U}(G_\$)$. We will later define a class of unification grammars which can be defined by production and lexicon rules on the forms used here. But first we will show that $G_\$$ and $\mathcal{U}(G_\$)$ are equivalent.

Lemma 2 *For every indexed grammar $G_\$$ on reduced form with a marked index end, $L(G_\$) = L(\mathcal{U}(G_\$))$.*

Proof: We have to show that for any string w, $w \in L(G_\$)$ if and only if $w \in L(\mathcal{U}(G_\$))$.

($\Longrightarrow$) For every $w \in L(G_\$)$ there exists a derivation tree $\langle D, C_{\mathcal{I}} \rangle$ for w based on $G_\$$. We have to show that based on $\mathcal{U}(G_\$)$ there exist c-structure with w as the terminal string which generates a well defined feature structure. We define the c-structure $\langle D, C_{\mathcal{U}}, E_{\mathcal{U}} \rangle$ on the same tree domain D.

For every nonterminal node x in D we have a unique production rule *license*(x) in the indexed grammar, and for each production rule in the indexed grammar we have a unique corresponding production or lexicon rule $\mathcal{U}(\mathit{license}(x))$ in $\mathcal{U}(G_\$)$ according to Definition 9. If

$$\begin{array}{ccccccc} \mathcal{U}(\mathit{license}(x)) & = & A_0 & \longrightarrow & A_1 & \ldots & A_n \\ & & & & E_1 & & E_n \end{array} \tag{16}$$

then let $C_{\mathcal{U}}(xi) = A_i$ and $E_{\mathcal{U}}(xi) = E_i$ for all $1 \leq i \leq n$, and let $C_{\mathcal{U}}(x) = A_0$. Then we have a valid c-structure and since $C_{\mathcal{U}}(x) = C_{\mathcal{I}}^{sym}(x)$ for all $x \in D$, it also has w as terminal string. Now we only have to show that all the equations in the c-structure are satisfied by a well defined feature structure.

For any finite string γ over an alphabet I we may define a feature structure where the node set is the union of all suffixes of γ and all symbols occurring in γ. Here we make a distinction between the singleton string of a symbol, and the symbol itself, such that they are regarded as two distinct nodes. For all non-empty string nodes, let the *idx* attribute point to the first symbol of the string and let the *next* attribute point to the rest of the string when we remove the first symbol, ie. $\delta(f\gamma', idx) = f$ and $\delta(f\gamma', next) = \gamma'$ for every non-empty suffix $f\gamma'$ of γ where $f \in I$. Let also the atomic value of each symbol-node be the symbol itself, ie. $\alpha(f) = f$. Else, let no more attributes or atomic values be defined, and in particular let $\delta(\varepsilon, next)$, $\delta(\varepsilon, idx)$ and $\alpha(\varepsilon)$ be undefined. We extend the definition directly to any finite set of strings over an alphabet. With any name-mapping to the string nodes defined from this finite set, this is a well defined feature structure since each nonempty string has a unique first symbol, and a unique suffix with length one less than the string itself.

Let M be the feature structure defined as described on the set of all index strings that occur in the derivation tree $\langle D, C_{\mathcal{I}} \rangle$, with the mapping of each nonterminal node in the tree domain to the index-string of that node: $m(x) = C_{\mathcal{I}}^{idx}(x)$. This is a well defined feature structure. We now have to show that all the equations in the c-structure are satisfied by the feature structure M. We have three different cases to consider:

Assume for a node x that $C_{\mathcal{I}}(x) = A\gamma$ where γ is an index-string and that $license(x) = A \rightarrow Bf$. Then $C_{\mathcal{I}}(x1) = Bf\gamma$, $m(x) = \gamma$ and $m(x1) = f\gamma$. From $\mathcal{U}(license(x))$ we have that $E'_{\mathcal{U}}(x1) = \{x1\; next \doteq x,\; x1\; idx \doteq f\}$, which is satisfied by the feature structure M since $\delta(f\gamma, next) = \gamma$, and $\alpha(\delta(f\gamma, idx)) = f$.

Assume for a node x that $C_{\mathcal{I}}(x) = Af\gamma$ where $f\gamma$ is an nonempty index-string and that $license(x) = Af \rightarrow B$. Then $C_{\mathcal{I}}(x1) = B\gamma$, $m(x) = f\gamma$ and $m(x1) = \gamma$. From $\mathcal{U}(license(x))$ we have that $E'_{\mathcal{U}}(x1) = \{x\; next \doteq x1,\; x\; idx \doteq f\}$, which is satisfied by the index-string feature structure M since $\delta(f\gamma, next) = \gamma$, and $\alpha(\delta(f\gamma, idx)) = f$.

Assume for a node x that $C_{\mathcal{I}}(x) = A\gamma$ where γ is an index-string and that $license(x) = A \rightarrow BC$. Then $C_{\mathcal{I}}(x1) = B\gamma$, $C_{\mathcal{I}}(x2) = C\gamma$ and $m(x) = m(x1) = m(x2) = \gamma$. From $\mathcal{U}(license(x))$ we have that $E'_{\mathcal{U}}(x1) = \{x \doteq x1\}$ and $E'_{\mathcal{U}}(x2) = \{x \doteq x2\}$, which is satisfied by the index-string feature structure M.

We do not have to consider the nodes which license production rules with terminal symbols since all the terminal nodes have empty equation sets. Then all the equations in the c-structure are satisfied by the feature structure M and then $w \in L(\mathcal{U}(G_\$))$.

($\Longleftarrow$) We will here use the function $idx\text{-}lst : Q \rightarrow V^*$ defined on any well defined acyclic feature structure as follows: $idx\text{-}lst(q) = \alpha(q)$ if $\alpha(q)$ is defined. If $\delta(q, idx)$ and $\delta(q, next)$ are both defined then $idx\text{-}lst(q)$ is the concatenation of $idx\text{-}lst(\delta(q, idx))$ followed by $idx\text{-}lst(\delta(q, next))$. Else $idx\text{-}lst(q) = \varepsilon$. We restrict our attention to its prefix with \$ as last symbol: Let $idx\text{-}lst_\$: Q \rightarrow V^*$ be the function such that: $idx\text{-}lst_\$(q)$ is the smallest prefix of $idx\text{-}lst(q)$ with \$ as the last symbol. If $idx\text{-}lst(q)$ does not contain any \$ then $idx\text{-}lst_\$(q) = \varepsilon$.

For every $w \in L(\mathcal{U}(G_\$))$ there exists a c-structure $\langle D, C_{\mathcal{U}}, E_{\mathcal{U}}\rangle$ for w based on $\mathcal{U}(G_\$)$ which generates a well defined feature structure. We define the derivation tree $\langle D, C_{\mathcal{I}}\rangle$ for w based on $G_\$$ on the same tree domain D. Let $C_{\mathcal{I}}^{sym}(x) = C_{\mathcal{U}}(x)$ for all nodes in D and $C_{\mathcal{I}}^{idx}(x) = idx\text{-}lst_\$(m(x))$ for all nonterminal nodes in D except for the root ε for which we define $C_{\mathcal{I}}^{idx}(\varepsilon)$ to be the empty string. This derivation tree has w as terminal string, and we just have to show that this is a valid derivation tree according to Definition 2.

Since $G_\$$ has a marked index-end, the only production rule where the start symbol occurs is $S \rightarrow A\$$, for an $A \in N$. This gives the following corresponding production rule in $\mathcal{U}(G_\$)$:

$$\begin{array}{ccc} S & \rightarrow & A \\ & & \downarrow next \doteq \uparrow \\ & & \downarrow idx \doteq \$ \end{array} \qquad (17)$$

which is the only production rule in $\mathcal{U}(G_\$)$ where the start symbol occurs. Then $C_{\mathcal{I}}(\varepsilon) = S$ which is the start symbol of $G_\$$. Here we also have that $idx\text{-}lst_\$(m(1)) = \$$ and $C_{\mathcal{U}}(1) = A$ so that $C_{\mathcal{I}}(1) = A\$$ and $S \rightarrow A\$$ licenses the root node. For all the other nonterminal nodes in the tree domain we have four cases to consider:

Assume for a nonterminal node x except for the root node that $C_{\mathcal{U}}(x) = A$ and $idx\text{-}lst_\$(m(x)) = \gamma$. Then $C_{\mathcal{I}}(x) = A\gamma$. Assume also that there exists a production rule in $\mathcal{U}(G_\$)$ from Definition 9 *a*), such that $C_{\mathcal{U}}(x1) = B$, $E'_{\mathcal{U}}(x1) = \{x1\; next \doteq x,\; x1\, idx \doteq f\}$ and $x1$ has no sister nodes. Since \$ only occurs in the one production rule with the start symbol, $f \neq \$$. Then $idx\text{-}lst_\$(m(x1)) = f\gamma$ and $C_{\mathcal{I}}(x1) = Bf\gamma$. From the reverse of Definition 9 *a*), there exists a production rule $A \rightarrow Bf$ in $G_\$$, which licenses x.

Assume for a nonterminal node x except for the root node that $C_{\mathcal{U}}(x) = A$ and $idx\text{-}lst_\$(m(x)) = f\gamma$. Then $C_{\mathcal{I}}(x) = Af\gamma$. Assume also that there exists a production rule in $\mathcal{U}(G_\$)$ from Definition 9 *b*), such that $C_{\mathcal{U}}(x1) = B$, $E'_{\mathcal{U}}(x1) = \{x\; next \doteq x1,\; x\; idx \doteq f\}$ and $x1$ has no sister nodes. Since \$ only occur in the one production rule with the start symbol, $f \neq \$$. Then $idx\text{-}lst_{\$}(m(x1)) = \gamma$ and $C_{\mathcal{I}}(x1) = B\gamma$. By the reverse of Definition 9 *b*), there exist a production rule $Af \rightarrow B$ in $G_{\$}$, which licenses x.

Assume for a nonterminal node x except for the root node that $C_{\mathcal{U}}(x) = A$ and $idx\text{-}lst_{\$}(m(x)) = \gamma$. Then $C_{\mathcal{I}}(x) = A\gamma$. Assume also that there exist a production rule in $\mathcal{U}(G_{\$})$ from Definition 9 *c*), such that $d(x) = 2$, $C_{\mathcal{U}}(x1) = B$, $C_{\mathcal{U}}(x2) = C$, $E'_{\mathcal{U}}(x1) = \{x \doteq x1\}$ and $E'_{\mathcal{U}}(x2) = \{x \doteq x2\}$. Then $idx\text{-}lst_{\$}(m(x1)) = idx\text{-}lst_{\$}(m(x2)) = \gamma$, $C_{\mathcal{I}}(x1) = B\gamma$ and $C_{\mathcal{I}}(x2) = C\gamma$ By the reverse of Definition 9 *c*), there exist a production rule $A \rightarrow BC$ in $G_{\$}$, which licenses x.

Assume for a nonterminal node x except for the root node that $C_{\mathcal{U}}(x) = A$ and $idx\text{-}lst_{\$}(m(x)) = \gamma$. Then $C_{\mathcal{I}}(x) = A\gamma$. Assume also that there exists a lexicon rule in $\mathcal{U}(G_{\$})$ from Definition 9 *d*), such that $d(x) = 1$, $C_{\mathcal{U}}(x1) = t$ and $E'_{\mathcal{U}}(x1) = \emptyset$. Then $C_{\mathcal{I}}(x1) = t$. By the reverse of Definition 9 *d*), there exist a production rule $A \rightarrow t$ in $G_{\$}$ which licenses x.

We then have a valid derivation tree with the same terminal string as the c-structure and then $w \in L(G_{\$})$. □

Example 2 Let $G = \langle N, T, I, P, S\rangle$ be an indexed grammar where $T = \{d\}$ is the set of terminal symbols, $N = \{S, A, B, C, C', D\}$ is the set of nonterminal symbols, $I = \{\$, f, g\}$ is the set of indices and P is the least set containing the following production rules:

$$\begin{array}{lll} S \rightarrow A\$ & B \rightarrow CC & \\ A \rightarrow Bf & Cg \rightarrow C' & C' \rightarrow CC \\ B \rightarrow Bg & Cf \rightarrow D & D \rightarrow d \end{array}$$

This grammar is on reduced form with a marked index-end. The simple unification grammar $\mathcal{U}(G)$ as given in Definition 9 is then the 5-tuple $\langle N, T, P', L', S\rangle$ where P' is the least set containing the following production rules:

$$\begin{array}{lll} S \rightarrow \begin{array}{c} A \\ \downarrow next \doteq \uparrow \\ \downarrow idx \doteq \$ \end{array} & B \rightarrow \begin{array}{cc} C & C \\ \uparrow \doteq \downarrow & \uparrow \doteq \downarrow \end{array} & \\ A \rightarrow \begin{array}{c} B \\ \downarrow next \doteq \uparrow \\ \downarrow idx \doteq f \end{array} & C \rightarrow \begin{array}{c} C' \\ \uparrow next \doteq \downarrow \\ \uparrow idx \doteq g \end{array} & C' \rightarrow \begin{array}{cc} C & C \\ \uparrow \doteq \downarrow & \uparrow \doteq \downarrow \end{array} \\ B \rightarrow \begin{array}{c} B \\ \downarrow next \doteq \uparrow \\ \downarrow idx \doteq g \end{array} & C \rightarrow \begin{array}{c} D \\ \uparrow next \doteq \downarrow \\ \uparrow idx \doteq f \end{array} & \end{array}$$

and L' contains one single lexicon rule:

$$D \rightarrow \begin{array}{c} d \\ \emptyset \end{array}$$

Figure 2 shows the derivation tree for the string *"dddd"* based on the indexed grammar G together with the c-structure and the feature structure for the same string string based on the simple unification grammar $\mathcal{U}(G)$. This shows that the string *"dddd"* is both in $L(G)$ and in $L(\mathcal{U}(G))$. The language generated by G and $\mathcal{U}(G)$ is $\{d^{2^n} \mid n \geq 1\}$.

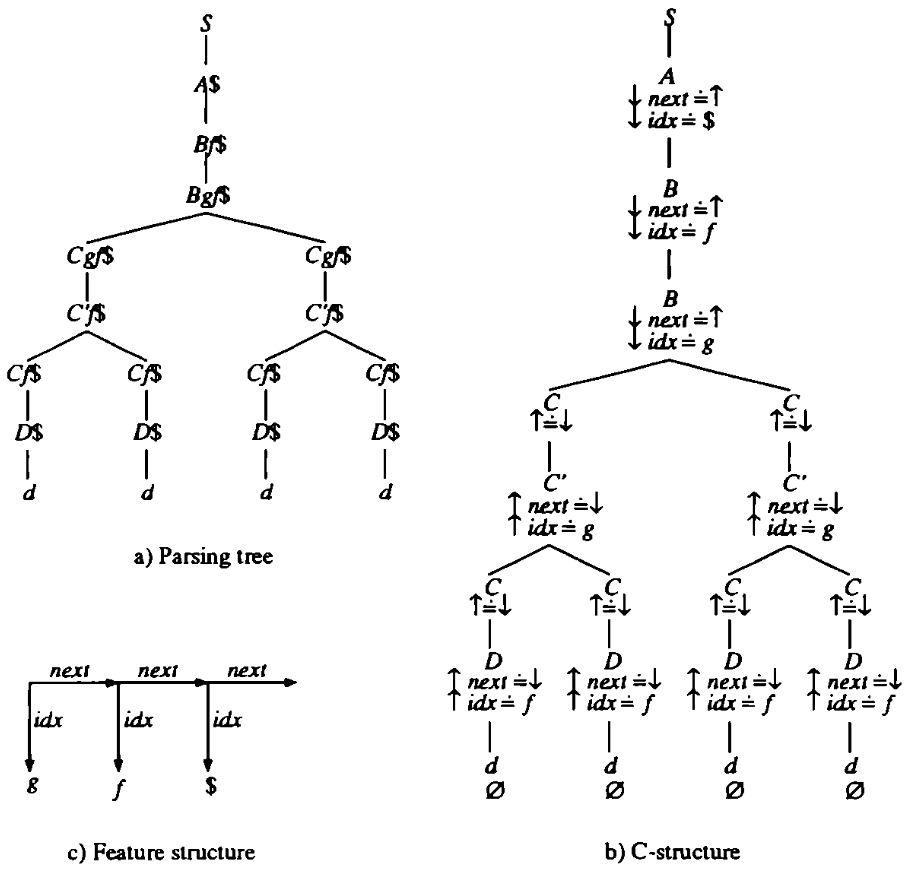

Figure 2: *derivation tree (a) for the string "dddd" based on the grammar G in Example 2, together with the c-structure (b) and feature structure (c) for the same string based on the grammar $\mathcal{U}(G)$.*

5 A Unification Grammar Formalism for Indexed Languages

We are here going to define a version of the simple unification grammar that describes the class of indexed languages. Just to be precise, a *class of languages*, $\mathcal{C}_\Gamma$ over a countable set Γ of symbols is a set of languages, such that each language $L \in \mathcal{C}_\Gamma$ is a subset of Σ^* where Σ is a finite subset of Γ. The class $\mathcal{C}_\Gamma(GF)$ of languages that a grammar formalism GF describes is the set of all languages L' over Γ such that there exists a grammar G in GF where $L(G) = L'$. The class of indexed languages is then the set of languages such that there for each language exist a indexed grammar that generates the language. We assume that Γ is the set of all terminal symbols that we use and drop Γ as subscript.

Definition 10 *A* UNIFICATION GRAMMAR FOR INDEXED LANGUAGES, $\mathcal{UGI}$ *is a simple unification grammar where*

a) each equation set in the production rules is on one of the three forms

- $E = \{\uparrow \doteq \downarrow\}$,
- $E = \{\downarrow \mathit{next} \doteq \uparrow, \downarrow \mathit{idx} \doteq f\}$,
- $E = \{\uparrow \mathit{next} \doteq \downarrow, \uparrow \mathit{idx} \doteq f\}$

where f is any value symbol, and next and idx are the same two attribute symbols for all equations in all production rules in $\mathcal{UGI}$, *b) each lexicon rule has en empty equation set.*

Lemma 3 *The class of languages $C(\mathcal{UGI})$ contains the class of indexed languages.*

Proof: Aho [Aho68] showed that for every indexed language there exists an indexed grammar on reduced form which generates the language. From Lemma 1 and its proof we have that for every indexed grammar G on reduced form there exists an indexed grammar on reduced form with a marked index-end $G_\$$, such that $L(G) = L(G_\$)$. The simple unification grammar $\mathcal{U}(G_\$)$ defined from the indexed grammar on reduced form with a marked index-end in Definition 9 is an $\mathcal{UGI}$ grammar. From Lemma 2 we have that $L(G_\$) = L(\mathcal{U}(G_\$))$. Then every indexed language can be generated by an $\mathcal{UGI}$ grammar. □

We shall now show that every $\mathcal{UGI}$ grammar generates an indexed language, but to do this we need some technical results. First it is easy to see that every $\mathcal{UGI}$ grammar can be formulated with rules only on the forms used in Definition 9 *a)*-*d)*. We define the reduced form for this.

Definition 11 *A $\mathcal{UGI}$ grammar is on* REDUCED FORM *if and only if every production rule is on one of the three following forms:*

$$\begin{array}{cccccccccc} A & \longrightarrow & B & \quad A & \longrightarrow & B & \quad A & \longrightarrow & B & C \\ & & \downarrow next \doteq \uparrow & & & \uparrow next \doteq \downarrow & & & \uparrow \doteq \downarrow & \uparrow \doteq \downarrow \\ & & \downarrow idx \doteq f & & & \uparrow idx \doteq f & & & & \end{array} \tag{18}$$

Lemma 4 *For every $\mathcal{UGI}$ grammar there is an equivalent grammar on reduced form.*

Proof: Using the techniques from the standard proof for normal form for context-free grammars, it is straight forward to replace each production rule in the original grammar not on reduced form with a set of new lexicon rules and production rules on reduced form. This can be done such that one instance of an original rule corresponds to the net effect of combining one ore more of the new rules. This is possible since we allow the empty string in lexicon rules. □

To make this formalism more directly comparable to indexed grammars with a marked index-end we use what we will call a *sink-mapped root*:

Definition 12 *A $\mathcal{UGI}$ grammar $\langle N, T, P, L, S\rangle$ has a* SINK-MAPPED ROOT *if and only if it has one and only one production rule where the start symbol occurs and this rule is on the form*

$$\begin{array}{ccc} S & \longrightarrow & A \\ & & \downarrow next \doteq \uparrow \\ & & \downarrow idx \doteq \$ \end{array} \tag{19}$$

where $A \in N$ and the value symbol \$ does not occur in any other production rule.

The value symbol \$ will form some kind of a blockade in the feature structure since it does not occur in any other production rule, hence no other node in the c-structure will be mapped to the same node in the feature structure as the root of the c-structure.

What we are doing here is to put a mark at the bottom of the stack of indices, in the way the nested stack is represented as a feature structure. We also want to map the root of the c-structure to the "sink" of the feature structure when we follow the *next* attribute.

Lemma 5 *For every $\mathcal{UGI}$ grammar G there exists a $\mathcal{UGI}$ grammar with a sink-mapped root G' such that $L(G) = L(G')$.*

Proof: First we show how we from any $\mathcal{UGI}$ grammar G may define a $\mathcal{UGI}$ grammar with a sink-mapped root G'. After this we show that for any string w, $w \in L(G)$ if and only if $w \in L(G')$.

Let any $\mathcal{UGI}$ grammar $G = \langle N, T, P, L, S \rangle$ be given, and assume that S_0, S' and S_ε are neither terminal nor nonterminal symbols in G, and that \$ is a value symbol not used in G. The grammar G' is defined by adding the following production and lexicon rules to the rules we have in G:

i) Let the following be two production rules:

$$\begin{array}{ccc} S_0 & \rightarrow & \begin{array}{c} S' \\ \downarrow next \doteq \uparrow \\ \downarrow idx \doteq \$ \end{array} \end{array} \tag{20}$$

$$\begin{array}{ccc} S' & \rightarrow & \begin{array}{cc} S & S_\varepsilon \\ \uparrow \doteq \downarrow & \uparrow \doteq \downarrow \end{array} \end{array} \tag{21}$$

ii) For each $f \in V$ used in any production rule in G, let the following be a production rule:

$$\begin{array}{ccc} S' & \rightarrow & \begin{array}{c} S' \\ \downarrow next \doteq \uparrow \\ \downarrow idx \doteq f \end{array} \end{array} \tag{22}$$

iii) Let the following be a lexicon rule:

$$\begin{array}{ccc} S_\varepsilon & \rightarrow & \begin{array}{c} \varepsilon \\ \emptyset \end{array} \end{array} \tag{23}$$

Complete G' by adding S_0, S' and S_ε to the nonterminal symbols, and let S_0 be the start symbol of G'. We see that G' is a $\mathcal{UGI}$ grammar with a SINK-MAPPED ROOT. Notice also that if G is on reduced form so is the new grammar.[3]

Now we have to show that for any string w, $w \in L(G)$ if and only if $w \in L(G')$.

($\Longrightarrow$) We show this direction in two steps: First we define something that we call a *canonical* feature structure for c-structures based on $\mathcal{UGI}$ grammars. This is done such that if the c-structure generates a well defined feature structure at all, then it is also generating the canonical feature structure. After this definition we show how we from a c-structure based on G, together with its canonical feature structure can construct a c-structure together with a feature structure based on the grammar G'. This is done such that the two c-structures have the same terminal string and if the terminal string is in $L(G)$ so is it in $L(G')$ also.

Let $\langle D, K, E \rangle$ be any c-structure based on a $\mathcal{UGI}$ grammar G such that it generates a feature structure. The *canonical feature structure* $\langle Q, \delta, \alpha, m \rangle$ for the c-structure is defined as follows: Let first Q_+ be the set of all sequences of nodes from the c-structure with at most $2n + 1$ nodes in each sequence, where n is the height of the c-structure. Then let the name mapping function m be defined on Q_+ by top-down induction on the nodes in the c-structure: First let the mapping of the root node, $m(\varepsilon)$ be the sequence of $n + 1$ ε's, $<\varepsilon, \varepsilon, ..., \varepsilon>$, where again n is

[3]The use of S_ε in rule (21) together with rule (23) where it will label the mother of a node with the empty string is only done because we want to stay in the domain of grammars on reduced form when G is on reduced form. This definition could be simplified if we did not want this.

the height of the c-structure. Now assume that $m(x)$ is defined for a node x in the c-structure. Then for each daughter xi of x, let

$$\begin{array}{lll} m(xi) = m(x) & \text{if} & \uparrow \doteq \downarrow \in E'(xi) \\ m(xi) = pop(m(x)) & \text{if} & \uparrow next \doteq \downarrow \in E'(xi) \\ m(xi) = add(xi, m(x)) & \text{if} & \downarrow next \doteq \uparrow \in E'(xi) \end{array} \tag{24}$$

where *pop* of any nonempty sequence is the sequence we get by removing the first element, $pop(<x_1, x_2, ..., x_k>) =<x_2, ..., x_k>$, and *add* of a single element and a sequence is the sequence we get by adding the single element to the beginning of the sequence, $add(x, <x_1, ..., x_k>) =<x, x_1, ..., x_k>$. Since the root node is mapped to the sequence of $n+1$ ε's, *pop* and *add* may not go out of their domain and therefore is m well defined.

Extend now the set Q_+ such that all the value symbol used in the c-structure also are elements in Q_+. Then let the partial function $\delta_+ : Q_+ \times \{next, idx\} \longrightarrow Q_+$ be defined such that $\delta_+(q, next) = pop(q)$ for all nonempty sequences $q \in Q_+$, and let $\delta_+(q, next)$ be undefined when q is the empty sequence. Moreover let $\delta_+(q, idx) = f$ for the value symbol f if and only if there exists a node x in the c-structure such that either $\downarrow idx \doteq f \in E(x)$, or $\uparrow idx \doteq f \in E(xi)$ for a daughter xi of x. This is the only place where inconsistency may occur and we will later see that it will not occur if the c-structure generates any feature structure at all. We extend the definition of the δ_+ to pairs of nodes and strings of the attribute symbols as described in the definition of feature structures in the beginning of section 3.

Now, let us shrink the definitions of Q_+ and δ_+ such that we get a well defined feature structure. First let $Q \subseteq Q_+$ be the set of all nodes that is reachable from a named node, formally $Q = \{q \mid \exists x \in D, \psi \in \{next, idx\}^* : \delta_+(m(x), \psi) = q\}$. Then, we restrict δ to the new domain: $\delta = \delta_+ \cap (Q \times \{next, idx\} \times Q)$. Finally, let $\alpha(f) = f$ for all value symbol used in the c-structure. We now have a feature structure and it is describable and acyclic directly from the definition of Q and δ. It is also atomic since δ is not defined on any feature symbol node, and α is only defined on feature symbol nodes. Moreover, it satisfies all the equations from the c-structure after we have instantiated the up and down arrows. We will now show that if the c-structure generates any well defined feature structure so will it generate the well defined canonical one also.

Let $M' = \langle Q', \delta', \alpha', m' \rangle$ be any well defined feature structure which the c-structure generates, and assume that we have the canonical feature structure as described. From the fact that the c-structure generates a feature structure, and from the definition of the canonical feature structure we have that if $m(x) = m(y)$ for any two nodes x and y in the c-structure then $m'(x) = m'(y)$. Now we may define a function $h : Q \longrightarrow Q'$ from the nodes in the canonical feature structure to the nodes in M', such that $m'(x) = h(m(x))$ for all nodes x in the c-structure. Assume then that we don't have a well defined canonical feature structure because of inconsistency in it definition. This means that there exist two instantiated equations, $x\ idx \doteq f$ and $y\ idx \doteq f'$ from the c-structure where $m(x) = m(y)$ but $f \neq f'$. However, then $m'(x) = m'(y)$, and inconsistency must also occur with respect to M' and the c-structure can not generate any well defined feature structure. Then the canonical feature structure must be consistent defined, and since it is also describable, acyclic and atomic it is well defined. Since it also satisfies all the equations in the c-structure it is generated by the c-structure.

Now we have a well defined canonical feature structure for each c-structure based on any $\mathcal{UGI}$ grammar if the c-structure generates a feature structure. Notice that $\delta(<\varepsilon>, idx)$ is not defined for the canonical feature structure. This due to the mapping of the root in the c-structure to the sequence of $n+1$ ε's, where n is the height of the c-structure. With this height it is only possible to pop of $n-1$ ε's according to definition of the name mapping (24), and since $\delta(q, idx)$ is only defined for q if there exist a node x such that $m(x) = q$, $\delta(<\varepsilon>, idx)$ can not be defined.

Assume now that $w \in L(G)$ for a grammar G. Then we have a c-structure for w based on G which generates a well defined feature structure. Then it is also generating a canonical feature structure $M = \langle Q, \delta, \alpha, m\rangle$ as described above. For this feature structure we extend the definition of δ and α as follows: First let $\delta(<\varepsilon>, idx) = \$$ and let $\alpha(\$) = \$$. For all sequences q of ε's such that $\delta(q, idx)$ is not defined, let $\delta(q, idx) = f$ for any value symbol f which occurs in the c-structure. When we construct the new c-structure based on G' the old nodes keep their mapping values.

We construct a new c-structure for w based on G' by the following steps: First add a new node on the top of the c-structure by applying the production rule (21). This give us also a new sister node for the old root node. Map the two new nodes to the same node in the extended canonical feature structure as the old root node. This secures that the equations in the production rule (21) is satisfied by the extended feature structure. The new sister node labeled with S_ε may only be a mother of a terminal node labeled with the empty string such that the terminal string is still w. Now add n nodes above the present root node by applying the generic production rule (22) $n - 1$ times and production rule (20) on the topmost node. This top node will be the root node in the new c-structure and it is now labeled with the start symbol in G'. When applying the generic production rule (22), let $f = \alpha(\delta(m(x1), idx))$ for each new node x where it is applied. The new nodes are each mapped to the sequence of k ε's, where k is the node's distance from the new root node. In this way the new root node is mapped to the empty sequence, the daughter of the root node is mapped to $<\varepsilon>$, and so on. Since $\delta(<\varepsilon>, idx) = \$$ the equations in production rule (20) is satisfied by the feature structure. Moreover since $f = \alpha(\delta(m(x1), idx))$ for each node x where the production rule (22) is applied and $\delta(q, next) = pop(q)$, all the equations is satisfied by the feature structure. We then have a c-structure based on G' with w as terminal string, and this c-structure generates a well defined feature structure. Then $w \in L(G')$.

($\Longleftarrow$) Assume that $w \in L(G')$ for a grammar G. Then there is a c-structure with category S_0 in the root, and a sequence of derivations down to a node with category S, where each intermediate node has category S'. This has been constructed by first using production rule (20) and then a sequence of zero or more applications of production rule (22) before production rule (21) gives the node with category S. Every node above the first node with category S has only one child, except the first which has an additional daughter, labeled with S_ε. This daughter is the mother of a single terminal node labeled with the empty string. Then we can remove all nodes above the node labeled S and still have the same terminal string w in the c-structure. The new c-structure will have a root-node with category S, and only production rules from the grammar G are used. Since the original c-structure generates a feature structure, so does the new one. Then $w \in L(G)$. □

Now we have the necessary technical results to show that every language in $\mathcal{C}(\mathcal{UGI})$ is an indexed language. We do this in two steps.

Lemma 6 *For any $\mathcal{UGI}$ grammar G on reduced form with a sink-mapped root, there exists an indexed grammar $G_\mathcal{I}$ such that $\mathcal{U}(G_\mathcal{I}) = G$.*

Proof: Assume that $G = \langle N, T, P, L, S\rangle$ is a $\mathcal{UGI}$ grammar on reduced form with a sink-mapped root. Then let $G_\mathcal{I} = \langle N, T, I', P', S\rangle$ be an indexed grammar where I' is all the value symbols occurring in G, and P' is constructed from P and L by reversing Definition 9 *a*)-*d*). This can bee done since G is on reduced form and there exist a one to one relation between the production rules in the indexed grammar and the production and lexicon rules in the unification grammar defined there. Since G

has a sink-mapped root the start symbol will occur in one and only one production rule together with a unique value symbol. Then $G_{\mathcal{I}}$ has a marked index-end and $\mathcal{U}(G_{\mathcal{I}}) = G$. □

Lemma 7 *Every language in $\mathcal{C}(\mathcal{UGI})$ is an indexed language.*

Proof: From Lemma 4 and Lemma 5 we have for any language in $\mathcal{C}(\mathcal{UGI})$ that there exist a $\mathcal{UGI}$ grammar G on reduced form with a sink-mapped root that generates the language. From Lemma 6 we have an indexed grammar $G_{\mathcal{I}}$ such that $\mathcal{U}(G_{\mathcal{I}}) = G$. By Lemma 2 we have that $L(G_{\mathcal{I}}) = L(G)$. Then we have an indexed grammar for all languages in $\mathcal{C}(\mathcal{UGI})$. □

From Lemma 3 and Lemma 7 we then have the following result:

Theorem 1 : *The class $\mathcal{C}(\mathcal{UGI})$ is the class of indexed languages.*

Acknowledgments

I would like to thank Tore Langholm for his advice during the work that this paper is based on, and for extended comments on earlier versions of this paper.

References

[Aho68] Alfred V. Aho. Indexed grammars —an extension of context-free grammars. *Journal of the Association of Computing Machinery*, 15(4):647–671, October 1968.

[Col91] Erik A. Colban. *Three Studies in Computational Semantics.* Dr.scient thesis, University of Oslo, 1991.

[Gal86] Jean H. Gallier. *Logic for Computer Science.* Harper & Row, Publishers, New York, 1986.

[Gaz88] Gerald Gazdar. Applicability of indexed grammars to natural languages. In Uwe Reye and Christian Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, pages 69–94. D. Reidel Publishing Company, Dordrecht, Holland, 1988.

[GM89] Gerald Gazdar and Chris Mellish. *Natural Language Processing in LISP.* Addison-Wesley Publishing Company, 1989. Also in Prolog version.

[Hay73] Takeshi Hayashi. On derivation trees of indexed grammars —an extension of the uvwxy-teorem—. *Publications of the Research Institute for Mathematical Sciences, Kyoto University*, 9(1):61–92, 1973.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[KB82] Ronald M. Kaplan and Joan Bresnan. Lexical functional grammar: A formal system of grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Gramatical Relations.* MIT-Press, 1982.

[Shi85] Stuart M. Shieber. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343, 1985.

Swedish Language Processing in the Spoken Language Translator

Björn Gambäck
Natural Language Processing Group
Swedish Institute of Computer Science
Box 1263, S - 164 28 KISTA
Stockholm, Sweden
`gamback@sics.se`

Abstract

The paper describes the Swedish language components used in the Spoken Language Translator (SLT) system. SLT is a multi-component system for translation of spoken English into spoken Swedish. The language processing parts of the system are the English Core Language Engine (CLE) and its Swedish counterpart, the S-CLE. The S-CLE is a general purpose natural language processing systems for Swedish which in the SLT project was tuned towards the register of the air travel information (ATIS) domain. The peculiarities and the coverage of the resulting Swedish grammar are the main topics of the paper, even though the overall SLT system also is briefly described.

1 Introduction

The Swedish Core Language Engine (or S-CLE for short) (Gambäck and Rayner, 1992) is a general purpose natural language processing system for Swedish developed by the Swedish Institute of Computer Science from its English counterpart, the SRI Core Language Engine (CLE) (Alshawi, 1992). The key idea behind the system is indicated by the word "core": the S-CLE was intended to be used as a building block in a broad range of applications and has already been tested as part of a database query system (Gambäck and Ljung, 1993) and as a text-to-speech front-end (Gambäck and Eineborg, 1995). The two copies of the CLE have also been used together to form a machine translation system for a car-hire domain (Alshawi *et al.*, 1991).

In the Spoken Language Translator, described in the next section, the English CLE performed as a back-end to a speech recognition system, the S-CLE as a front-end to a speech synthesis system, and the two CLEs together formed a (text) translation system in the air travel information domain. In the course of the project, the previous Swedish system was completely redesigned and the general-purpose grammar expanded, but also tuned to cover the peculiarities of the register (sublanguage) of a particular domain.

The present paper starts out by describing the overall SLT system architecture in Section 2 and briefly introduces the different components of the system. Section 3 is the main focus of the paper, describing the different modules of the present Swedish language processing component in detail by giving examples of the rules used for the treatment of some specific phenomena.

Section 4 details the coverage issues and how the Swedish coverage was improved during the first year of the project. The final section of the paper looks into the future, describes the ongoing work on making the system completely bidirectional, and sums up the previous discussion.

2 The SLT system

The Spoken Language Translator (SLT) is a system prototype which can translate queries from spoken English to spoken Swedish in the domain of air travel planning (ATIS). The system was developed as a joint effort by the Swedish Institute of Computer Science, SRI International (Menlo Park, US and Cambridge, UK), and Telia Research AB (Haninge, Sweden). Most of the first-year prototype was constructed from previously existing pieces of software, which were adapted for use in the speech translation task with as few changes as possible. The overall architecture of the current version of SLT system is described shortly in this section, for a complete description see (Rayner *et al.*, 1993) or (Agnäs *et al.*, 1994).

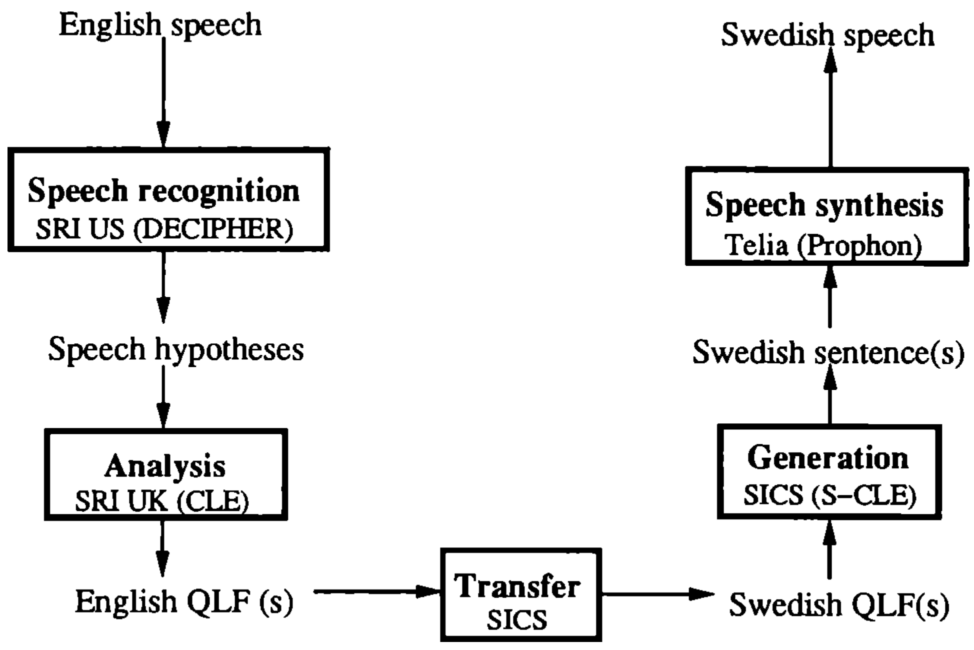

Figure 1: Top-level architecture of the Spoken Language Translator

The main components of the SLT system are connected together in a pipelined sequence as shown in Figure 1. The input signal is processed by SRI Menlo Park's DECIPHER(TM) (Murveit *et al.*, 1991), a speaker-independent continuous speech recognition system based on Hidden Markov Model technology. It produces a set of speech hypotheses which is passed to the English-language processor, the SRI Cambridge Core Language Engine (Alshawi, 1992).

The CLE grammar associates each speech hypothesis with a set of possible quasi-logical forms, QLFs (Alshawi and van Eijck, 1989), typically producing 5 to 50 QLFs per hypothesis. In order to allow fast processing of a large number of hypotheses, a scaled-down version of the grammar induced with the machine-learning technique "Explanation-Based Learning" (Samuelsson and Rayner, 1991) is first invoked and parsed with an LR-parser (Samuelsson, 1994). Only if this restricted-coverage grammar fails is the general-purpose grammar tried on the (by the speech recognizer) most preferred hypothesis.

A preference component is then used to give each QLF a numerical score reflecting its linguistic plausibility (Alshawi and Carter, 1994). When the preference component has made its choice, the highest-scoring logical form is passed to the transfer component, which uses a set of simple non-deterministic recursive pattern-matching rules to rewrite it into a set of possible corresponding Swedish representations (Alshawi *et al.*, 1991; Gambäck and Bretan, 1994).

The preference component is now invoked again, to select the most plausible transferred logical form. The result is fed to a second copy of the CLE, which uses a Swedish-language grammar and lexicon developed at SICS (Gambäck and Rayner, 1992) to convert the form into a Swedish string and an associated syntax tree. Finally, the string and tree are passed to the Telia Prophon speech synthesizer, which utilizes polyphone synthesis to produce the spoken Swedish utterance (Bäckström *et al.*, 1989).

The SLT system's current performance figures measured on previously unseen data (the 1001-utterance December 1993 ATIS corpus) are: 78.8% of all utterances are such that the top-scoring speech hypothesis is an acceptable one. If the speech hypothesis is correct, then an acceptable translation is produced in 68.3% of the cases and the overall performance of the system is 53.8%. Limiting the test corpus to sentences of 10 words or less (688 utterances), these figures move up to 83.9% for speech recognition and 74.2% for language processing, with a 62.2% overall performance.

For about 10% of the correctly recognized utterances, an unacceptable translation is produced. Nearly all of these are incorrect due to their containing errors in grammar or naturalness of expression, with errors due to divergence in meaning between the source and target sentences accounting for less than 1% of all translations. SLT performance is discussed at length in (Rayner *et al.*, 1994).

3 Swedish Language Processing

As noted above, the S-CLE is a general purpose natural language processing system for Swedish. The main object of the system is to map certain natural language expressions into appropriate predicates in quasi-logical form. The system is based completely on unification and has a fairly large bidirectional phrase-structure type grammar (i.e., the grammar can be used both for analysis and generation) covering most of the common Swedish constructions. There is a good treatment of inflectional morphology, covering all main inflectional classes of nouns, verbs and adjectives.

The S-CLE has been developed from the original English CLE by replacing English-specific modules (grammar, morphology, lexicon and lexicon acquisition) with corresponding Swedish-language versions, exploiting the large overlap between the structures of the two languages. Most of the Swedish grammar is thus completely equivalent to the English one; this section will concentrate on the parts that differ for interesting reasons. (So, even though the grammars indeed differ in several ways not described here, most of the differences are for rather uninteresting reasons more reflecting different tastes on the side of the grammarians than real grammatical differences and will thus be left out from the discussion here.)

A previous version of the Swedish grammar and how it was developed was described in (Gambäck and Rayner, 1992). There we also went into some detail on the (at least for a translation task) most vital differences between English and Swedish, both at the morphology and syntax levels. The present paper will thus refrain from recapitulating that discussion and only give an overview of the most important phenomena and their present treatment in the system.

First, however, we should note that the simple methodology outlined for developing a system for a new language has also been shown to be successful for other languages. A full-scale version of the CLE for French has recently been developed by ISSCO, Geneva. It has a coverage at roughly the same level as the Swedish one (Rayner and Bouillon, 1995) and is also used as a part of a spoken language translation system (English-to-French), which was demonstrated at the CeBIT fair in Hannover, March 1995.

Small-scale versions of the CLE are also under development by the University of Cambridge: for German (Parkinson, 1992), mainly for testing the grammar formalism on a language with a different word order; for Polish (Styś, 1995), testing the morphology component on the intricacies having to do with case, gender and number variation on nouns, as well as the noun phrase part of the grammar on some of the problems associated with a "free" word order; and for Korean.

The rest of this section will in turn go through the different processing steps used when forming a QLF in the S-CLE and describe the rule sets used in each of them: first the morphological processing where the rulebase is divided into morphophonological "spelling" rules and morphosyntactic "production" rules. Then the grammatical processing which in turn is divided into two steps, syntactic parsing and semantic analysis. The rules of the grammar proper are thus divided into two different rule sets, one with the syntax and another with the (compositional) semantics. The main processing chain is as shown in Figure 2.

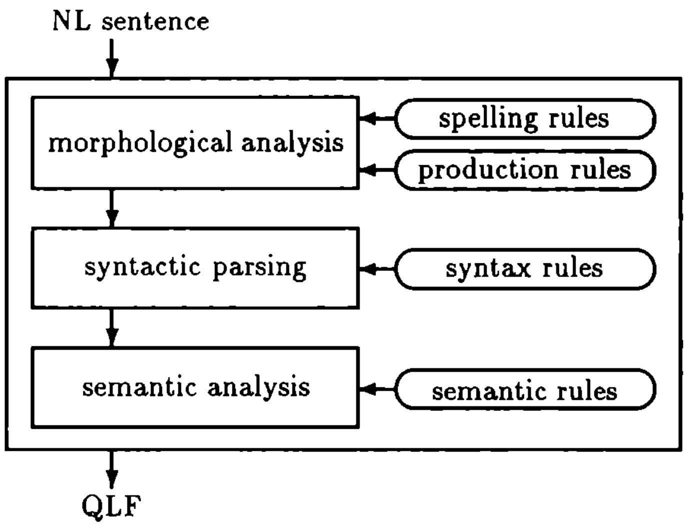

Figure 2: The analysis steps of the S-CLE

3.1 Morphology

Given that Swedish is an inflectional language, the treatment of the inflectional morphology by simple affix-stripping used in the original English CLE was far from sufficient. A "lazy" version of the two-level morphology (Koskenniemi, 1983) was thus implemented (Carter, 1995). This version is "lazy" in that it does not account for general changes of the stems of words.

A typical spelling rule is the following which shows that when the affix `er` is added to a stem ending with an `o` or an `e` followed by an `l` or an `r`, the stem vowel may be dropped unless it is stressed (i.e., formler = formel + er, manövrer = manöver + er, etc.):

```
spell(plur_LRer_eLR,
      "||", =>, "|2|1+er",
      [2/"oe",1/"lr"],
      [stresslast=n]).
```

In all the rule formalisms of the CLE, the first argument (here, `plur_LRer_eLR`) is simply a rule name mainly used for debugging. The main parts of the rule appear on the different sides of the arrow (=>): these are the surface and lexical forms, respectively. The vertical bars (|) indicate which letters may be changed in the rule. If the arrow is bidirectional (<=>), the rule *must* apply; here it may optionally apply. The final two lists put restrictions on the "variables" `1` and `2` in the rule, and on possible feature settings on the stem.

In the current version of the Swedish morphology (which is still under development), 58 such spelling rules appear and are complemented by another set of 4 interword rules used in the derivational morphology, which in Swedish is also quite complex; however, since the current version of the system cannot handle derivational morphology in general, we will not go into too much detail here, but concentrate on the — for the task at hand — most important part of it, namely the production of noun compounds, which are extremely common in the ATIS domain.

While noun compounds in English are formed simply as groups of words, the Swedish compounds are formed by actually compounding the words together. In general, this can be done in a wide variety of fashions, but in present-day Swedish mainly in two ways only: either by just "gluing" the words together, or by inserting an *-s-* between the words in the compound, as described in for example (Kiefer, 1970). Compounds can in general be of almost all word-classes, but the most common ones are noun compounds, in which the last word of the compound is a noun; the other words in the compound can be of other classes (e.g., adjectives or adverbs), but are normally nouns, as well.

As a rule-of-thumb, noun compounds are formed first without inserting an infix *s*, but if the compound consists of more than two words, an *s* will be inserted for every second word added to the compound, so for example the following sequence would give the words for "father", "grand-father" (father's father), "great grand-father", etc.:

far, *farfar*, *farfarsfar*, *farfarsfarfar*, ...

Whether a particular noun will form compounds by inserting an *s* or not depends on the word in question and is thus lexicalized.

To implement the noun compound formation, a feature nn_infix on an $\overline{\text{N}}$ (nouns are lexicalized as $\overline{\text{N}}$s) indicates whether or not it can be post-modified with a complex $\overline{\text{N}}$ compound. The feature can currently take the values s (for an infix '-s-') or n (for no infix, in practice '--', i.e., just a hyphen) and is lexicalized on the $\overline{\text{N}}$, thus indicating the fact that some lexicon $\overline{\text{N}}$s take an s-infix when forming a compound, and some do not.

The following morphological production rule for noun-noun compounds (there are similar rules for other types of noun compounds) show together with the two rules for infixes ('-s-' or '--') how the nn_infix feature propagates as 00+01=01, 01+10=00, i.e., an $\overline{\text{N}}$ that takes no infix has nn_infix(0,0) as its lexicon value and meets the '--'-infix which has nn_infix(0,1) to produce an $\overline{\text{N}}$ with nn_infix(0,1), which in turn can produce an $\overline{\text{N}}$ that takes the null-infix if it meets the '-s-'-infix (nn_infix(1,0)), etc.

```
   nbar:[nn_infix=(I,0), simple=n, ...]
    -->
   nbar:[nn_infix=(I,N), ...]
    +
   'INFIX':[nn_infix=(N,0)]
    +
   nbar:[simple=y, ...]

  lex('-s-',['INFIX':[nn_infix=(1,0)]]).
  lex('--',['INFIX':[nn_infix=(0,1)]]).
```

The setting of the feature simple force complex compounds to form in a left-branching fashion; the right-most daughter may not itself be a compound (must have simple=y).

Production rules like the one above currently number 27 in the system, only 4 of which are used for forming compounds. These production rules are actually used by the syntactic morphological processing and are more or less paralleled by 33 semantic morphological derivation rules.

3.2 Syntax

On the syntactic side, the English and Swedish grammars differ on many accounts. Firstly, several extra rules appear in the Swedish system, mainly to capture different kinds of movements, in particular the fact that Swedish allows for topicalization of just about any type of constituent. Space considerations prevent a full account of these rules from being included in this paper; they will be reported on elsewhere (Gambäck, 1995). Here, we will thus only concentrate on some prototypical cases.

Secondly, a number of new features had to be added or the value ranges or relevant rules for old features had to be extended. Most notably since the more complex agreement structure of Swedish means that the features indicating agreement and definiteness must be propagated to many more constituents.

An example is the three-valued definiteness feature, which ranges over values for "indefinite", "definite" and "possessive", the last one being used on genitive NPs. These are treated as forming complex determiners, so that '*en mans fru*' (a man's wife), '*mannens fru*' (the man's wife), and '*Kalles fru*' (Kalle's wife) are all interpreted as having the structure [NP [DET $\overline{\text{N}}$]] as examplified in Figure 3.

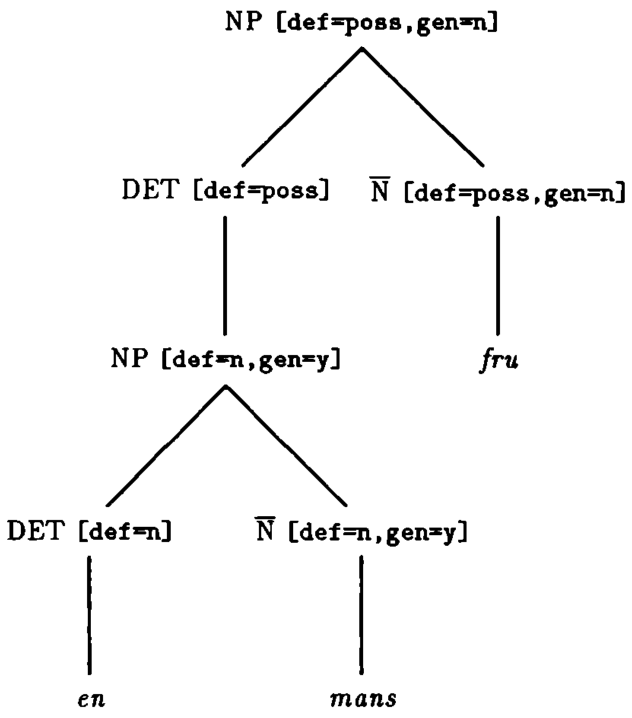

Figure 3: The tree structure for the noun phrase '*en mans fru*'

This is obtained by using the following two rules (here quite simplified with most features removed):

```
syn(det_np_Genitive,
[det:[def=poss],
  np:[def=_, gen=y]
]).

syn(np_det_nbar,
[np:[def=D, gen=G],
  det:[def=D],
  nbar:[def=D, gen=G],
]).
```

The first rule specifically forms determiners from genitive NPs (with the feature setting `gen=y`) regardless of the NP's definiteness (`def=_`), giving the newly formed determiner a possessive definiteness. The second rule forms NPs from determiners and nouns as long as the definiteness values on the daughters unify. This rule may be used on a wide range of determiner and noun types, including genitives.

3.3 Semantics

Most of the differences between English and Swedish syntax is only mirrored at the (QLF, i.e., compositional) semantic level without any interesting additions. The most notable exception is the verb-phrases. Already at the syntax-level, most word-order differences stem from the strongly verb-second nature of Swedish: formation of both YN- and WH-questions is by simple inversion of the subject and verb, without the introduction of an auxiliary. This is illustrated in the following examples:

Han såg Maria.	*He saw Mary.*
Såg han Maria?	*Did he see Mary?*
Vem såg han?	*Who did he see?*

This difference of verb syntax can actually be factored away. However, we will not dwell in too much detail on the rather special unification-based treatment of verb-phrases used in the system — for that, the reader is referred to (Gambäck, 1993a; Gambäck, 1993b) — but will note that the main trick used is *lexicalization*: information regarding for example verb subcategorization schemes (i.e., the number and type of verbal complements, such as objects, particles, etc.) is removed from the grammar and put in the lexicon instead. Syntactically, this enables us to treat both English and Swedish verb-phrases of different kinds with a rule like the following:

```
syn(vp_v_comp_Normal,
[vp:[tense_aspect=TA],
  v:[aux=_, tense_aspect=TA,
     subcat=Complements]
  | Complements
]).
```

where the value of the subcat feature of the verb has to unify with the rest of the verb-phrase. The value of subcat is specified for a particular verb in its lexical entry and can of course be empty (for intransitives, etc.). Our current Swedish grammar treats 48 different main verb complement patterns plus copulas and auxiliaries. Without claiming this to be the absolute number of Swedish verb types in any sense, it is easily understandable that without the strategy outlined above, we would have been forced to state specific instances of the verb-phrase formation rule for a vast number of cases.

In the CLE, each syntactic rule is paralleled by (at least) one semantic rule. For all English verbs and for Swedish main verbs, the verb-phrase rule above has a simple counterpart, but even for Swedish auxiliaries the treatment causes no problems, even though an extra case of the semantic rule had to be added in order to pass tense and aspect information properly, given that for main verbs, the tense information of the verb-phrase is the same as the one of the daughter verb and is simply unified up together with the other semantic information, while in the auxiliary case, the semantic interpretation of the mother verb-phrase still is the one of the daughter verb-phrase, but the tense is to be taken from the auxiliary.

Thus we get the following two (indeed very simplified!) semantic rules:

```
sem(vp_v_comp_Normal, mainv,
[(V,vp:[tense_and_aspect=TA],
  (V,v:[aux=n, tense_aspect=TA,
        subcat=Complements])
  | Complements
]).

sem(vp_v_comp_Normal, aux,
[(V,vp:[tense_and_aspect=TA],
  (Aux,v:[aux=y, tense_aspect=TA,
            subcat=(V,vp:[])]),
  (V,vp:[])
]).
```

Note that each constituent in the semantic rules is a pair with the first part holding the semantic logical-form fragment and the second part holding the (basically) syntactic information.

3.4 Negation

A specific case where the English and Swedish grammar differs significantly is in the treatment of negation. Negation in Swedish is expressed with the particle '*inte*' (not), which is placed after the main verb in a main clause, but before it in a subordinate clause, thus:

Han snarkade inte.	*He did not snore.*
...att han inte snarkade.	*...that he did not snore.*

Similar considerations also apply to a number of other common adverbials (so-called "mobile adverbs"), including '*ofta*' (often), '*alltid*' (always) and '*troligen*' (probably).

Even though negation tends to be used to a very small degree in the ATIS domain, a serious natural-language processing system must of course treat it, however, it does cause some problems both for English-Swedish transfer and for the QLF-formalism as such. The design choice in the English CLE was to treat negation semantically as an operator on the sentence structure which at the syntactic level pre-modifies a verb-phrase forming a new verb-phrase, the rule thus being schematically:

```
VP -> not VP
```

In Swedish such a treatment does not suffice; negation is still viewed as an operator at the semantic level, but instead of modifying verb-phrases, it is taken as modifying the verb itself in the syntax. Since whether the modification is pre- or post- depends on the type of clause, this has been treated by adding a `subordinate` feature to S, VP and V.

Three rules for verbs are needed, the first treating main clause negation, the second treating subordinate clause negation and the third treating a special case of main clause negation with a pronoun as object:

1. *mannen [gillade inte] Maria/mig* *the man did not like Mary/me*

```
v:[subordinate=n, ...]
 -->
v:[...]
 +
neg:[]
```

2. *att mannen [inte gillade] Maria/mig* *that the man did not like Mary/me*

```
v:[subordinate=y, ...]
 -->
neg:[]
 +
v:[vform=(\(att)), ...]
```

3. *mannen [gillade mig inte]*　　　　*the man did not like me*

```
v:[subordinate=n, subcat=Rest, ...]
 -->
v:[subcat=[Pro|Rest], ...]
 +
Pro
 +
neg:[]
```

At the semantic level, treating negation as an operator causes some problems. Mainly since all mobile adverbs ought to be treated in the same way, but introducing QLF-operators for all of them would hardly be feasible. Thus negation is actually the only mobile adverb treated by the present version of the Swedish grammar. This problem and the fact that while modification of English verb phrases occurs external to the VP, Swedish modifiers are internal can be taken as an argument against having a VP node at all in Swedish, or as basis for introducing a $\overline{V}$ node. The above treatment goes a bit along the way of the second alternative.

4 Swedish grammar coverage

Without going into more details of the Swedish grammar, we should note that its coverage on the ATIS task was increased substantially during the project.

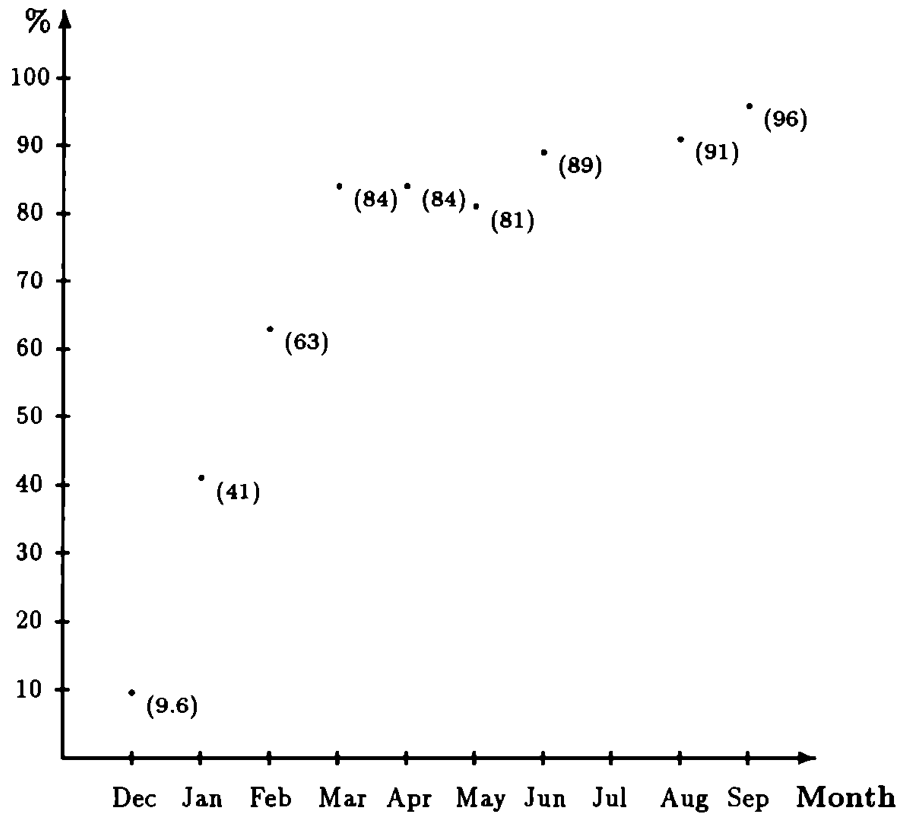

Figure 4: Transfer and generation coverage increase

Tests on a representative 281 sentence corpus showed an increase in coverage of the transfer and generation components combined from a mere 9.6% in mid-December 1992 to 96% in mid-September 1993, as can be seen in Figure 4.

As could be expected the main coverage increases were obtained early on in the project. After awhile, the coverage stabilized around 80%; to further increase the coverage, some major changes had to be undertaken, changes which at first actually lead to a slight coverage drop (as shown by the figure for mid-June).

Note that the figures in the graph refer to sentences that obtained a translation, *any* translation. For a discussion of the translation quality, see (Agnäs *et al.*, 1994).

5 Future Work and Conclusions

In the paper, the Swedish language processing component of the SLT English-to-Swedish spoken language translation system has been described. The main emphasis has been on the grammar and its coverage, but the other modules of the language processing part have also been described. The overall SLT system prototype and its coverage after the first year of the project has only been briefly discussed, while the paper has focused on the different modules of the Swedish processing component. These have been described mainly on a pro-example type level, showing the various rule formalisms at work.

At the date of writing, work has just begun on a second phase of the SLT project. We intend to reverse the system, so that translation of spoken Swedish into spoken English will be possible. Even though the main part of the work needed for that will be on producing a Swedish speech recognition system, the language processing components will be extended quite a lot at the same time. Partly because the Swedish part of the system has not been extensively tried for language processing as opposed to just generation for awhile, partly because the new version of SLT also will include extended processing in a new spoken language database query task, as well as allowing for some translations in a computer mediated person-to-person dialogue setup.

In parallel, work will be undertaken on systematically testing how the grammar coverage of the Swedish system can be tuned towards a new domain (Berglund and Gambäck, 1995) and whether the system is robust enough to be used as the basis for building a tree-bank of Swedish analyses (Santamarta *et al.*, 1995). Both these tests will use the representative Swedish "Stockholm-Umeå corpus" (SUC) (Ejerhed *et al.*, 1992).

6 Acknowledgements

The work reported here was funded by the Swedish Institute of Computer Science and Telia Networks.

I would like to thank Ivan Bretan, Jussi Karlgren and Christer Samuelsson who all were involved in the SLT project at SICS, my other colleagues in the SICS NLP-group, everybody else working on SLT related issues at the various sites, in particular Manny Rayner and Malgorzata Styś for some useful suggestions. Charlotta Berglund, Nikolaj Lindberg and Lena Santamarta are worthy of many thanks for (unwillingly) having had to help out in debugging the grammar while doing their BA and MA Thesis work.

References

Agnäs, M.-S., Alshawi, H., Bretan, I., Carter, D., Ceder, K., Collins, M., Crouch, R., Digalakis, V., Ekholm, B., Gambäck, B., Kaja, J., Karlgren, J., Lyberg, B., Price, P., Pulman, S., Rayner, M., Samuelsson, C., and Svensson, T. 1994. Spoken Language Translator: First-Year Report. Joint Research Report R94:03 and CRC-043, SICS and SRI International, Stockholm, Sweden and Cambridge, England.

Alshawi, H., editor. 1992. *The Core Language Engine.* The MIT Press, Cambridge, Massachusetts.

Alshawi, H. and Carter, D. 1994. Training and Scaling Preference Functions for Disambiguation. *Computational Linguistics*, 20:635–648.

Alshawi, H. and van Eijck, J. 1989. Logical Forms in the Core Language Engine. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 25–32, Vancouver, British Columbia.

Alshawi, H., Carter, D. M., Gambäck, B., and Rayner, M. 1991. Translation by Quasi Logical Form Transfer. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 161–168, University of California, Berkeley, California.

Bäckström, M., Ceder, K., and Lyberg, B. 1989. Prophon — An Interactive Environment for Text-to-Speech Conversion. In *Proceedings of the European Conference on Speech Communication and Technology*, pages 144–147, Paris, France.

Berglund, C. and Gambäck, B. 1995. On Testing Domain Adaptability. In *Proceedings of the 10th Scandinavian Conference on Computational Linguistics*, Helsinki University, Helsinki, Finland. (presentation).

Carter, D. 1995. Rapid Development of Morphological Descriptions for Full Language Processing Systems. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, pages 202–209, University College of Dublin, Dublin, Ireland.

Ejerhed, E., Källgren, G., Wennstedt, O., and Åström, M. 1992. The Linguistic Annotation System of the Stockholm–Umeå Corpus Project. Report 33, Department of General Linguistics, University of Umeå, Umeå, Sweden.

Gambäck, B. 1993a. On Implementing Swedish Tense and Aspect. In *Proceedings of the 9th Scandinavian Conference on Computational Linguistics*, pages 97–109, Stockholm University, Stockholm, Sweden.

Gambäck, B. 1993b. Towards a Uniform Treatment of Swedish Verb Syntax and Semantics. In *Proceedings of the 14th Scandinavian Conference of Linguistics and the 8th Conference of Nordic and General Linguistics*, pages 123–134, University of Gothenburg, Gothenburg, Sweden.

Gambäck, B. 1995. *Processing Swedish Sentences: A Unification-Based Swedish Grammar and some Applications for It.* Doctor of Engineering Thesis, Stockholm University/Royal Institute of Technology, Stockholm, Sweden.

Gambäck, B. and Bretan, I. 1994. Complex Verb Transfer Phenomena in the SLT System. In *Proceedings of the 1st Conference of the Association for Machine Translation in the Americas*, pages 89–96, Columbia, Maryland.

Gambäck, B. and Eineborg, M. 1995. A Grammar-Based Rule Formalism for a Text-to-Speech Interface System. In *Proceedings of the 5th Scandinavian Conference on Artificial Intelligence*, Trondheim, Norway. (to appear).

Gambäck, B. and Ljung, S. 1993. Question Answering in the Swedish Core Language Engine. In *Proceedings of the 4th Scandinavian Conference on Artificial Intelligence*, pages 212–225, Stockholm, Sweden. Also available as SICS Research Report, R92014, Stockholm, Sweden.

Gambäck, B. and Rayner, M. 1992. The Swedish Core Language Engine. In *Papers from the 3rd Nordic Conference on Text Comprehension in Man and Machine*, pages 71–85, Linköping University, Linköping, Sweden. Also available as SICS Research Report, R92013, Stockholm, Sweden.

Kiefer, F. 1970. *Swedish Morphology*. Skriptor, Stockholm, Sweden.

Koskenniemi, K. 1983. *Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production*. Doctor of Philosophy Thesis, University of Helsinki, Helsinki, Finland.

Murveit, H., Butzberger, J., and Weintraub, M. 1991. Speech Recognition in SRI's Resource Management and ATIS Systems. In *Proceedings of the 4th Speech and Natural Language Workshop*. DARPA, Morgan Kaufmann.

Parkinson, S. 1992. A Computational Grammar for Use in Machine Translation. Master of Philosophy Thesis, Cambridge University, Cambridge, England.

Rayner, M. and Bouillon, P. 1995. Hybrid Transfer in an English–French Spoken Language Translator. (In manuscript.).

Rayner, M., Alshawi, H., Bretan, I., Carter, D. M., Digalakis, V., Gambäck, B., Kaja, J., Karlgren, J., Lyberg, B., Pulman, S. G., Price, P., and Samuelsson, C. 1993. A Speech to Speech Translation System Built from Standard Components. In *Proceedings of the Workshop on Human Language Technology*, Princeton, New Jersey. ARPA, Morgan Kaufmann.

Rayner, M., Carter, D. M., Price, P., and Lyberg, B. 1994. Estimating Performance of Pipelined Spoken Language Translation Systems. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Kyoto, Japan.

Samuelsson, C. 1994. Notes on LR Parser Design. In *Proceedings of the 15th International Conference on Computational Linguistics*, volume 1, pages 386–390, Kyoto, Japan.

Samuelsson, C. and Rayner, M. 1991. Quantitative Evaluation of Explanation-Based Learning as an Optimization Tool for a Large-Scale Natural Language System. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 609–615, Sydney, Australia.

Santamarta, L., Lindberg, N., and Gambäck, B. 1995. Towards Building a Swedish Treebank. In *Proceedings of the 10th Scandinavian Conference on Computational Linguistics*, Helsinki University, Helsinki, Finland. (presentation).

Styś, M. 1995. Incorporating Discourse Aspects in English–Polish MT: Towards Robust Implementation. (In manuscript.).

Locally Tree-shaped Sentence Automata and Resolution of Ambiguity

Jussi Piitulainen
Research Unit for Multilingual Language Technology
Department of General Linguistics
University of Helsinki
jpiitula@ling.helsinki.fi

Abstract

The framework of finite state intersection grammars is directed towards practical representation and parsing of running text. A problem in parsing has been that intersection with successive rule automata can produce prohibitively large intermediate sentence automata. See (Koskenniemi, Tapanainen and Voutilainen 1992) and (Koskenniemi 1990) and (Voutilainen and Tapanainen 1993).

This paper sketches a data structure and parsing method that may help to compute the intersection of a sentence automaton with all rule automata while keeping the size of intermediate sentence automata under control.

1 Representation of sentence readings

Sentence readings are finite sequences of word forms, morphosyntactic labels and boundary labels (Voutilainen 1994, Voutilainen and Tapanainen 1993). The structure of a sentence reading is such that each word form reading contains the word form, a base form, a few morphological labels and one or two syntactic labels, and word form readings are separated by boundary labels. For example, the correct reading of the sentence 'And time began seriously to pass.' in figure 1 identifies 'time' as a subject `@SUBJ`, and 'began' as a main verb in a main clause `@MV MAINC@`, and so on. Other readings might misidentify 'time' as a verb or 'to' as a preposition, or correctly identify 'time' as a noun but misidentify it syntactically as an object or an apposition. An incorrect choice for a word boundary label would indicate the presence of more than one finite verb form.

There are five alternative word boundary labels: `@@` begins and ends a sentence, embedded finite clauses are enclosed between `@<` and `@>`, a boundary where a finite clause ends and a new one starts is labeled with `@/`, and `@` is used for others.

A finite set of readings is represented as an acyclic finite state automaton called a sentence automaton. Word forms in the sentence have all lexically possible morphosyntactic readings as alternatives, and each word boundary is made four or sometimes five ways ambiguous. The initial number of readings in a sentence automaton is the product of the lexical and word boundary ambiguities in the sentence.

```
                                                @@
<and> and CC                     @CC            @
<time> time N NOM SG             @SUBJ          @
<began> begin V PAST VFIN        @MV MAINC@     @
<seriously> serious ADV          @ADVL          @
<to> to INFMARK                  @aux           @
<pass> pass V INF                @mv OBJ@       @
<.>                              @fullstop      @@
```

Figure 1 A sentence reading

The grammar is represented as a number of finite state automata called rule automata. Each rule automaton is constructed to accept all readings that are to be grammatical and to reject some readings that are to be ungrammatical. In other words, the grammar is taken to be the intersection of individual rules.

Sentence automata will be drawn with the the start state on the left, the direction of edges from left to right, and the final state as a double circle on the right.

2 The problem

The task is to compute the intersection $S \cap R_0 \cap \cdots \cap R_{n-1}$ of a sentence automaton S with all rule automata $R_0, \ldots, R_{n-1}$. The main practical problem is that sometimes intersection with successive rule automata produces prohibitively big intermediate sentence automata. Any method that computes the same final result can be used instead of the straightforward intersection; this paper proposes one such method.

The number of readings represented by the sentence automaton decreases monotonically during the parsing process. The number of states in the sentence automaton tends to increase at first; the initial automaton can be very compact precisely because it contains all alternatives, and a smaller set may be represented by a bigger automaton.

The removal of the dashed edges in figure 2 leaves two minimal automata of which the bigger represents the smaller set of readings. This shows how removal of readings can require addition of states and edges.

The two automata in figure 2 represent a set of four readings. The upper automaton is minimal and the lower automaton is (almost) maximal. Consider now a rule that rejects the sequence $\langle x, a, y, a, z\rangle$ and accepts the other three. No edge can be removed from the minimal automaton since every edge belongs to one of $\langle x, a, y, b, z\rangle$ and $\langle x, b, y, a, z\rangle$ and $\langle x, b, y, b, z\rangle$. In the maximal automaton, the dashed edges and the state between them belong only to the rejected sequence and can be removed.

The final sentence automaton $S \cap R_0 \cap \cdots \cap R_{n-1}$ represents the set of correct readings for the sentence. If the grammar is accurate, this automaton is again quite small because it represents a very small set of readings.

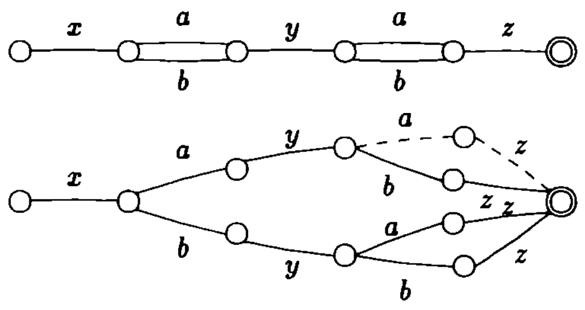

Figure 2 A minimal automaton and a maximal automaton

3 New representation of sentence automata

This section introduces a new representation for sentence automata. The new data structure is designed to facilitate separation of two operations that intersection would combine, namely removal of readings and expansion of the sentence automaton. This separation should help keep the size of intermediate automata under control.

Edges and states can be removed if they are not parts of any reading that should remain. In a tree-shaped automaton, all removable readings can be removed by removing such edges and states. Since it is strictly impractical to make the whole sentence automaton tree-shaped, the automaton is made tree-shaped only locally. Initially, the local trees span readings of word forms, but they can be expanded to span longer intervals.

There must be states with more than one immediate predecessor. These are immediate successors of the leaves of local trees. Since the trees initially span word forms, the word boundaries correspond to collections of the states that may have several immediate predecessors. These collections will be called slices.

Each slice of a sentence automaton contains exactly one state of every path between the start state and the final state. The first slice contains the state that immediately follows the start state, and the last slice contains the final state. Initially slices contain only one state.

The two automata in figure 3 represent a set of a few readings of the sentence 'Time passed'. Punctuation and some labels, notably **MAINC@**, are omitted to save space. The upper automaton is locally tree-shaped; the lower automaton is an ordinary minimal automaton.

The rectangles in the upper automaton represent the three slices corresponding to the three word boundaries in the sentence.

Each state in a slice is a root of a local tree. The leaves of the tree are immediate predecessors of some roots in the next slice. Note that a state that is properly between slices has only one immediate predecessor, so that the trees do not share structure.

The data structure representing a sentence automaton should give easy access to the information needed in the algorithms to be described, in addition to the underlying automaton structure. First, the containing slice, if any, should be available given a state. Second, the previous state or states should be available given a

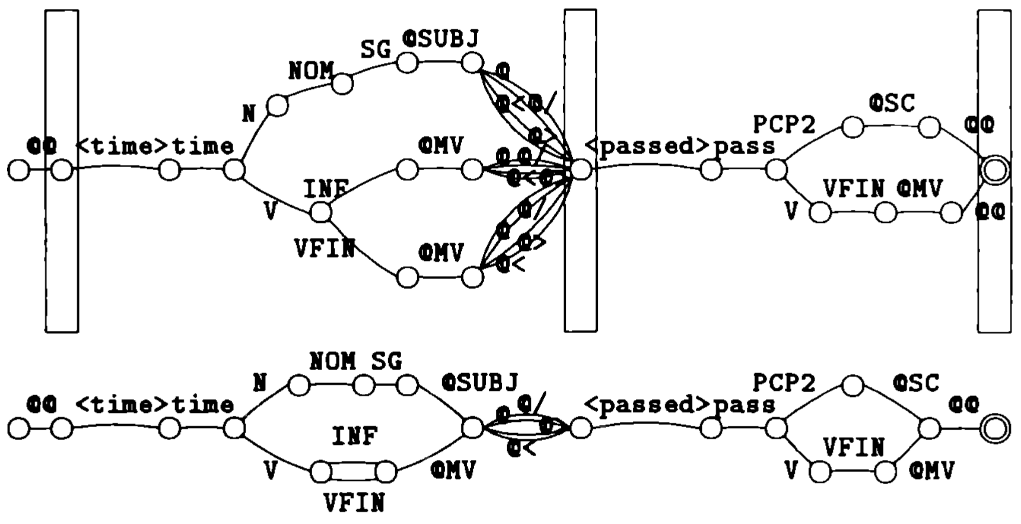

Figure 3 Locally tree-shaped vs. minimal

state. Third, the states in slices and immediate predecessors of states in slices will be assigned sets of rule states that should be available given such a state. It should be easy to iterate over all edges that leave a state.

4 Operations on the new data structure

The new data structure is designed so that removal and addition of states and edges are separate operations. This is in contrast to intersection that not only removes readings but also expands the sentence automaton in a way that is difficult to control.

In contrast to sentence automata, next to nothing is assumed about the representation of rule automata. All that is required is that it is possible to compute the state where a given state takes a given symbol, and to decide whether there is any way to reach a final state from a given state.

A state is called a sink if there is no way to reach a final state from it. A rule can fail by going into a sink long before reaching the end of an input sequence.

4.1 Initialization and propagation

In the new representation, sentence automata contain two special kinds of states, namely the roots of the local trees and the leaves of the local trees. The roots are in slices, and each leaf is an immediate predecessor of a root. (The start state is also thought of as a leaf.) Readings are removed in two steps called propagation and removal.

The propagation step assigns sets of states of a rule automaton to the roots and leaves of the local trees. Initially, a set containing the start state of the rule automaton is assigned to the start state of the sentence automaton. There is only one path from the start state of the sentence automaton to the first root, the one labeled **@@**; the set containing the state to which the start state of the rule automaton takes the label **@@** is assigned to the first root.

Once initialized, the propagation step continues by assignment of sets of rule states to the roots and leaves of each local tree in turn. For example, a certain rule automaton goes to a sink state when it encounters the label @MV after it has already encountered @MV but not yet encountered a clause boundary. Let state 1 be the start state, state 2 be the state after the first @MV and state 5 be the sink state. In the upper automaton in figure 4, the root has been reached through paths that contain one @MV and paths that do not contain @MV. Therefore, the root is assigned the set of two rule states $\{1,2\}$. When this set is propagated further, the rule automaton stays in these two states until it encounters the @MV label; then the state set would be $\{2,5\}$ but the sink state 5 is simply ignored.

In the lower automaton in figure 4, the root has been reached only through paths that contain one @MV. The state set after the second @MV contains only the sink state and becomes effectively empty.

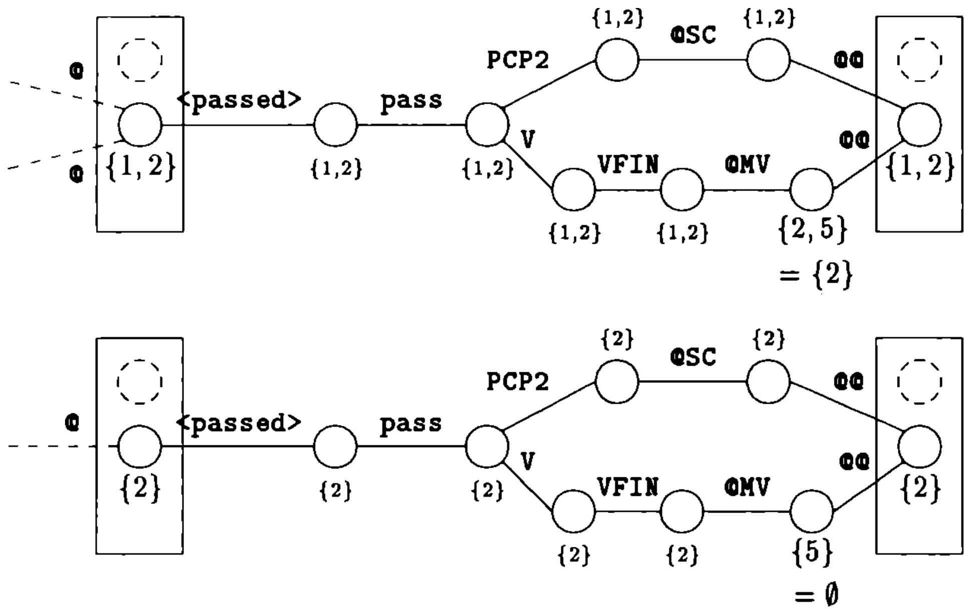

Figure 4 Propagation of a rule state set

The first root has been assigned in the initialization step. There is a unique path from the root of a local tree to a leaf of that tree. This path and the rule states in the root determine the set of rule states that are assigned to the leaf. Rule states in the leaf are those to which some rule state in the root takes the path from the root to the leaf, excluding sinks. When all the rule states would be sinks, this set is empty.

Unlike leaves, the roots in a slice can have several predecessors. A root is assigned the set of rule states that are reachable from some rule state in some of its predecessors by some path from the leaf to the root.

4.2 Removing paths that block propagation

The sets of rule states in the leaves help to remove states and edges. First, if a leaf is assigned the empty set of rule states, there is no way for the rule to reach this leaf. The leaf can be removed, and edges and states preceding it can be removed back to the first state with more than one immediate successor. All boundary edges forward can be removed, and if the following root had no other immediate predecessor, removal can continue forward. This means that the edges labeled `V VFIN @MV @@` in the lower automaton in figure 4 can be removed.

Second, if all the rule states in some leaf take some boundary edge to a sink, the edge can be removed. If all the edges from the leaf to a root are removable this way, removal can continue backward and forward.

Third, a non-final state in the last slice can be treated as if it were a sink. There is no way to reach another final state from the last state in the sentence automaton.

Only such readings are removed as would be removed by intersection. However, not all such readings can be removed this way. To allow removal of further readings, the sentence automaton may need to be expanded.

4.3 Expanding the sentence automaton locally

In the new method, the sentence automaton cannot grow while readings are removed. A separate operation of local expansion is provided to open up removal opportunities.

Observe that the initial sentence automaton is tree-shaped between any pair of successive slices. The expansion makes it tree-shaped (rather forest-shaped) between some given pair of slices. Each state that is properly between the two slices will have exactly one immediate predecessor.

The example sentence 'Time passed' is so short that there is only one way to choose a pair of slices for expansion. The initial automaton is shown in figure 3; the expanded automaton is shown in figure 5. For longer sentences, such maximal expansion is not feasible.

The first word form 'time' is three ways ambiguous and the word boundary after it is four ways ambiguous; together 'time' and the boundary are twelve ways ambiguous. Expansion makes twelve copies of the state in the slice between the readings of 'time' and 'passed', and the tree of 'passed' is turned into a forest of twelve identical trees.

Much of the new structure is easy to remove. Particularly interesting are the three trees of 'passed' that follow the boundary symbol `@/`. This boundary symbol separates finite clauses: there should be the label `VFIN` both before and after it. This constraint is not enough to remove anything in the automaton of figure 3 since `VFIN` occurs in one of the readings of 'time', but in the automaton of figure 5, two of the three trees can be removed. For example, after `@@ <time> time N NOM SG @SUBJ @/` the rule should be in a sink state.

Another impossibility is to have `@<` and `@@` or `@<` and `@<` as successive boundaries. The former constraint would allow removal of much of the structure in the expanded automaton: the label `@@` would take a rule to a sink.

Parts of the lower four trees of 'passed' are removable by the constraint that the boundary symbol between two finite verb forms `VFIN` can not be `@`, so that the second `VFIN` would take a rule to a sink.

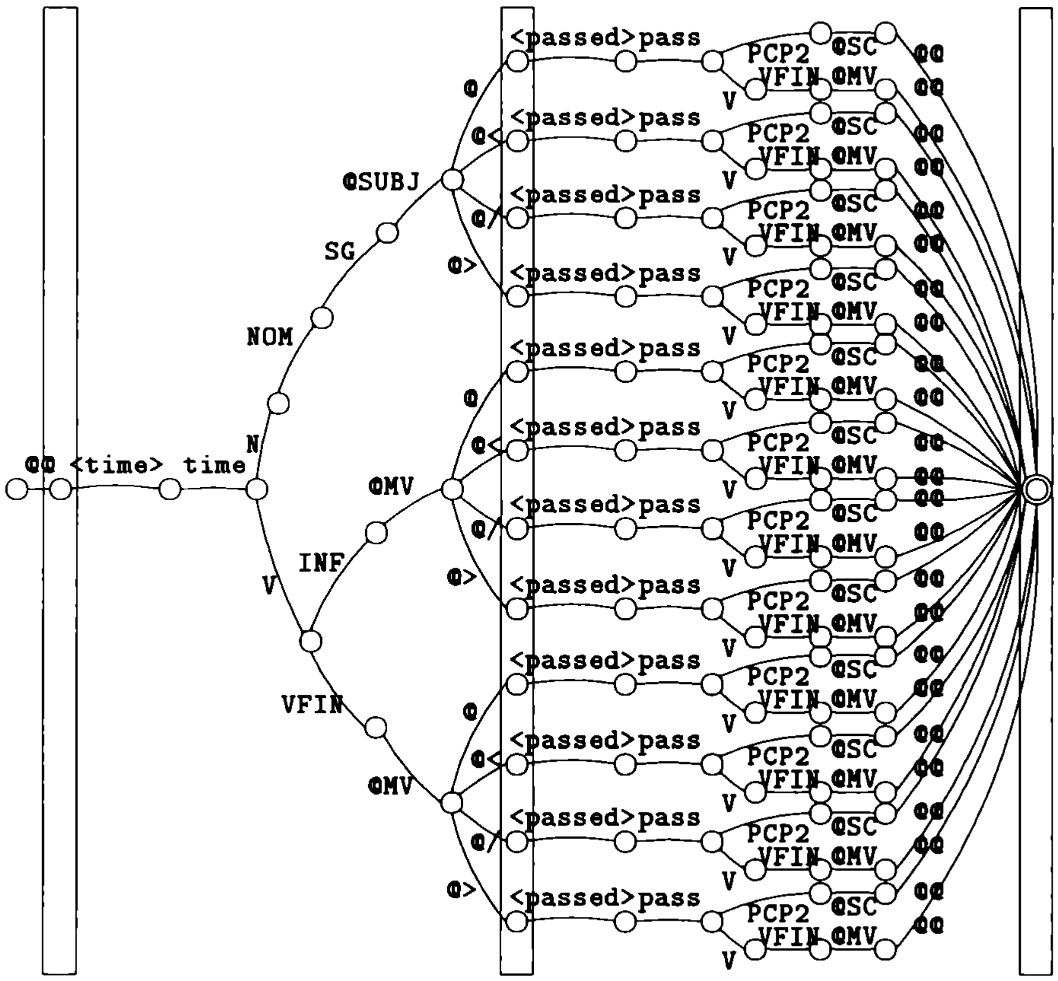

Figure 5 An expanded sentence automaton

5 Computation strategies

The high level strategy to use is an open question. The operations of propagation, removal and expansion can be used in different orders. In this method, rules need to be used more than once, and the order of expansions and removals may affect efficiency significantly.

The simple minded but prohibitively expensive strategy would be to first maximize the sentence automaton and then just use every rule in turn.

Another strategy would be to exercise all rule automata on a short interval in the beginning of the sentece automaton first, and then lengthen the interval. A third strategy would be to expand short intervals and then propagate and remove.

More intelligent strategies might take into account special properties of each sentence automaton, or special properties of some rules. Or such intervals might be expanded that contain relatively few readings.

Kimmo Koskenniemi (personal communication) suggested first propagating a rule into the sentence automaton and then studying the sets of rule states to see whether this rule can allow removals after expansion. In the positive case, expand and then propagate again.

6 Some details

This section gives a more detailed description of the operations on the locally tree-shaped automata than was given before. All automata here are deterministic finite state automata.

An automaton is a finite set of uniquely labeled states, partitioned into final and non-final states, with one state designated as the start state.

All states will be seen as total functions taking sequences to states, as follows. When s and t are states and a path from s to t is labeled by a sequence of alphabet symbols w, the notation $s\,w$ designates the state t. The set of all prefixes of such sequences w that s takes to a final state is called the domain of s. A state with empty domain is called a sink; if w is not in the domain of s, then $s\,w$ is a sink.

Each state s satisfies the equations $s\ \langle\rangle = s$ and $s\ \langle a, b, \ldots\rangle = (s\ \langle a\rangle)\ \langle b, \ldots\rangle$ for all $a, b, \ldots$ in the alphabet. In a minimal automaton, a sink s is a non-final state that satisfies $s\,w = s$ for all sequences w.

If $s\ \langle a\rangle = t$ for some symbol a, then s is said to be an immediate predecessor of t, and t is said to be an immediate successor of s.

In the process to be specified, a sentence automaton $\mathcal{S}$ is represented as a sequence of slices $\langle \mathcal{S}_0, \ldots, \mathcal{S}_{n-1}\rangle$. It must be replaced by successive sentence automata so that the final result represents the intersection of the original sentence automaton with a number of rule automata $R_0, \ldots, R_{k-1}$.

6.1 Propagation

Rule states can be propagated from $\mathcal{S}_i$ to $\mathcal{S}_{i+1}$. An intermediate step is to propagate them to the leaves of each local tree with root in $\mathcal{S}_i$. When a path from a root to a leaf is labeled w and the set of rule states assigned to the root is R, the set of rule states in the leaf will be $\{\, r\,w \mid r \in R \,\}$ excluding sinks.

After propagation to the leaves, an immediate predecessor p of a state t in $\mathcal{S}_{i+1}$ contains the correct set R' of rule states. If D is the domain of p, the set of rule states in t will be a superset of $\{\, r\ \langle d\rangle \mid r \in R' \text{ and } \langle d\rangle \in D \,\}$ excluding sinks. Other immediate predecessors of t produce other states.

Rule states can be propagated from the start state to each slice in turn. If the start state of a rule is r_0, the state in the first slice will contain just the state $r_0\ \langle @@\rangle$.

6.2 Removal

When rule states have been propagated from $\mathcal{S}_0$ to $\mathcal{S}_i$, it may be possible to remove edges and states that precede states in $\mathcal{S}_i$.

If p is an immediate predecessor of some state s in $\mathcal{S}_i$ and the set of rule states in p is empty, then the rule takes to a sink all sequences that label paths from the start state to p. The state p can be removed together with preceding edges and states back to the first state that has more than one edge in its domain.

If the set of rule states in p is not empty but there are several edges from p to s, all the rule states can take some of the edges to a sink. Such edges can be removed.

6.3 Expansion

When the sentence automaton is expanded between $\mathcal{S}_i$ and $\mathcal{S}_{i+2}$, and some state s in $\mathcal{S}_{i+1}$ has k immediate predecessors, the state s and its local tree are replaced with k copies, each with its own immediate predecessor out of the immediate predecessors of the original s.

7 Summary

Intersection of finite state automata reduces readings but may add structure in the automata. Separation of the two effects may help to keep the size of a sentence automaton under control. Keeping the sentence automaton locally tree-shaped makes this separation possible.

Future work will involve implementation of the parser and empirical study of various prasing strategies.

Acknowledgements

Special thanks to Anssi Yli-Jyrä, Kari Granö, Atro Voutilainen and Kimmo Koskenniemi, and thanks to Timo Lahtinen, Timo Järvinen, Mikko Lounela and the two referees, for reading and comments.

The two example sentences are from the Douglas Adams novel, Dirk Gently's Holistic Detective Agency, first published in 1987 by William Heinemann Ltd.

References

Kimmo Koskenniemi. 1990. Finite-state Parsing and Disambiguation. In Hans Karlgren (ed). *COLING-90: Papers Presented to the 13th International Conference on Computational Linguistics, Vol. 2.* Helsinki, Finland.

Kimmo Koskenniemi, Pasi Tapanainen and Atro Voutilainen. 1992. Compiling and Using Finite-State Syntactic Rules. In *Proceedings of COLING-92, Vol. I.* Nantes, France.

Atro Voutilainen and Pasi Tapanainen. 1993. Ambiguity resolution in a reductionistic parser. In *Proceedings of EACL-93.* Utrecht, The Netherlands.

Atro Voutilainen. 1994. Designing a Parsing Grammar. In *Three Studies of Grammar-Based Surface Parsing of Unrestricted English Text.* Ph.D. dissertation. Helsinki, Finland.

Sense Extension Functions in Lexical Semantics

Peter Rossen Skadhauge
Department of General and Applied Linguistics
University of Copenhagen
E-mail: rossen@cphling.dk

Abstract

Representing polysemy in an economical way is an issue of major importance within lexical semantics. Polysemy is found both within single lexical entries, and systematically in some lexical classes with common semantic properties. Prepositions in various languages are generally considered highly polysemic in an unpredictable way. The latter participate in what can be called systematic polysemy. This work is highly inspired by work as different as Pustejovsky [Pus91], Copestake and Briscoe [CB95], and Lakoff [Lak94].

I will sketch a framework or the fundamentals of a formalism in which important polysemic properties can be described. The interpretational semantics is built as typed lambda-calculus. This choice is not essential to the formalism, which might be extended to situation-theoretical notation and interpretation. Currently, situation-theoretical issues are not discussed within the framework.

It is briefly outlined how the lexical semantics as construed in this paper can be implemented in a typed feature structure formalism compatible to HPSG [PS94]. Accounts of various aspects of prepositional semantics are given in this formalism, with special emphasis on the Danish preposition **med**.

1 Systematic polysemy

Certain phenomena are usually referred to as polysemy. One such example is the well-known example by Pustejovsky [Pus91]:

(1) a. Mary enjoys the movie

b. Mary enjoys watching the movie

The sentences (1a) and (1b) are synonymous, and in order to maintain compositionality and avoid multiple lexical entries for the verb **enjoy**, the semantics is accounted for by claiming that **enjoy**'s semantics ENJOY is a two-place predicate taking an event as its second argument. The noun **movie** belongs to a class of complex lexical entries that enables it to act semantically both as an event involving some watching and as a simple object that can be watched. The two senses are related by a process called *type-raising*. **Movie** and similar nouns obviously form a class, which can be represented in a hierarchical lexicon as being marked for susceptibility to type-raising.

The phenomenon is referred to as *logical metonymy* because the relation between a movie and the event of watching a movie can be judged to be familiar with usual metonymic relations pictured in (2), where the underlined NP's can be said to be interpreted identically in certain contexts.

(2) a. <u>Denmark</u> voted against the treaty

b. The majority of the Danish voters voted against the treaty

By accounting for the phenomenon in semantic terms, one does not have to posit that the syntactic difference between **the movie** and **watching the movie** should trigger different lexical entries of **enjoy**. Furthermore, enjoying different sorts of events can still be described by the same verb entry:

(3) Franz enjoyed the sausage

By inserting the semantics for typical events involving sausages into the semantics for **sausage**, one can infer that the semantics for the clause (3) contains some eating event.

2 Extending the scope: Lexical metaphor

Within theoretical linguistics, polysemy, metonymy and metaphor are traditionally regarded as if not out of bounds, then at least as marginal phenomena not worth paying too much attention to when describing the language system as it is typically construed by linguists. In computational linguistics it is often thought that such topics should be treated in an AI fashion, without employing the known structures of the linguistic system. In my view, polysemy constitutes an at least empirically indistinguishable part of the language systems. It is the norm rather than the exception that words are used in different but related senses. It is a lexicological challenge to account for a system within which the senses of every lexeme are related instead of just listing the various senses of the individual lexemes, in much the same sense as it is a challenge for the phonologist to state and arrange the phonemes of a language in a system stating generalizations on properties across sounds instead of listing the individual sounds.

The most general system in which all senses of all lexemes can be represented is not interesting for my purpose: construed as a feature-structure system the number of primitive features would have the same magnitude as the number of lexemes.

What one needs is a limited system with a few dimensions along which the sense extensions take place. As it is the case for any kind of linguistic categorization, such dimensions or features must be empirically motivated. This restricts the domain of the functions in question to be a quite narrow one. The sense extensions must be testable either in the cognitive/physical system or as a means to underpin grammatical generalizations.

The set of sense extension functions that apply to the whole range of lexical items is believed to be a very small, general one. One such function is the meronymic PART-OF function, which is present in Ray Jackendoff's [Jac91] and several other authors' accounts. Other, more special functions apply to special domains. Lexemes whose semantics have an inherent spatial and/or temporal structure like activity verbs and prepositions can have spatial functions applied to them.

I shall restrict myself to treat lexical metaphor. It is not yet clear to me whether or how phrasal metaphor should be described as a linguistic process.

According to [Lak94], one must distinguish between metaphor and metaphor *use*. Metaphor is here construed as a function between structured sense domains. Metaphor use is a pair containing a metaphor and a source sense. The metaphor is a mapping from source senses to target senses. Both source and target senses are linked to the same lexeme, i.e., they are expressible with the same phonology. Some target senses seem to recur more often than others; a linguist might judge some recurrent senses as *lexicalized metaphorical uses*, because they can be conventional and have achieved unpredictable connotations, use of the latter can be characterized as *creative metaphorical uses.* I emphasize that I can state nothing about the psychological status of the senses nor whether the sense extension functions mirror how any senses come about in the mind of the language user. Just like phrase-structure rules in formal grammar are often understood as acceptability constraints on syntax, sense extension functions should be conceived of as formal relations between interpretations.

The lexicalized uses are parts of the language system according to which computational linguists might consider it worth to enable the computer to parse as well as generate.

The creative uses are highly relevant for parsing, but probably of minor importance with respect to computational generation, because computer users do not expect any kind of nonconventional creative behaviour from computers.

The important point is that both lexicalized and creative uses arguably can be described with the same set of functions, and only statistical methods can distinguish between the two.

The cognitivist approach states some metaphors in a quite elaborate hierarchical system [Lak94]. This system is highly interesting, certainly not because the cognitivist positions reflect the observations of vagueness that many linguists strive to account for, but simply because it forms an informal version of a strong hypothesis about interesting parts of the language system.

One well-known metaphor is TIME IS SPACE. I conceive of this metaphor as a sense extension function which is no less formally describable than what is usually the case within formal linguistics. Furthermore, it fulfills the testability requirement. The function is an instance of a general sense extension function that connects two partially isomorphic domains. In this case, the time domain is mapped onto one of the dimensions in the spatial domain. Actually, the time domain can be construed as a particular instance of a one-dimensional subdomain of the general three-dimensional spatial domain. With this construal, TIME IS SPACE is a reflexive metaphorical function, i.e., a function from a domain to the domain itself or a subdomain thereof.

Because the function is reflexive, the inverse metaphor SPACE IS TIME is automatically present. In other words, it is possible to view time as a trajectory in space, and thus communicate about time in spatial terms. Vice versa, one can communicate about space in temporal terms. You can express a distance by referring to the time it conventionally takes to travel it.

Because of the reflexive nature of the sense extension function, it is not very well argued that either time or space is the basic domain of the relevant prepositions. One could as well postulate that the basic domain is a one-dimensional ordered space, and apply trivial sense extension functions to evoke the attested uses. With the sketched method it is not possible to devise one basic sense. In this case, the choice of basic sense is arbitrary.

When applying sense-extension function to prepositional arguments, one can account for basic senses of prepositions in the following examples.

(4) a. Jeg bor 10 minutter fra universitetet
'I live 10 minutes from the university'

b. Toget standsede 8 minutter efter Helsinki
'The train stopped 8 minutes after Helsinki'

c. Vandet varede 10 kilometer
'The water lasted 10 kilometres'

All the uses in (4) — which I cannot determine as lexicalized or creative — can be accounted for using the functions in (4′).

(4′) a. $\text{PERIOD}(x) \rightarrow \text{DISTANCE}(\varphi_a(x))$

b. $\text{LOCATION}(x) \rightarrow \text{MOMENT}(\varphi_b(x))$

c. $\text{DISTANCE}(x) \rightarrow \text{PERIOD}(\varphi_c(x))$

The first of these functions simply states that a sense of the type PERIOD is mapped to a derived sense of type DISTANCE. The argument x is the period, and the function φ_a is a physically dermined function that maps a certain period of time to the distance usually covered in that periode of time. The other functions can be described in a parallel way.

Polysemy is possible according to the TIME IS SPACE metaphor. (5) is ambiguous in the context of a train voyage that Peter performs every day. The PP '10 minutter efter Køge' can refer to (a) a location at which the train is usually located 10 minutes after leaving the station of Køge, and to (b) the time 10 minutes after the train leaves Køge, regardless of how far the train has actually gone.[1]

(5) Peter spiser altid sin madpakke 10 minutter efter Køge
'Peter always eats his lunch 10 minutes after Køge'

The lexeme **fra** ('from') is stated to have the sense

(6) $([\lambda(a : \text{distance}).\lambda(b : \text{location}).(c : \text{location})|\mathit{distance}(b,c) = a] : \text{sense})$

The lexeme **efter** ('after') is stated to have the sense

(7) $([\lambda(a : \text{period}).\lambda(b : \text{moment}).(c : \text{moment})|\mathit{period}(b,c) = a] : \text{sense})$

The **til** ('to') and **før** ('before') senses are stated parallel to that.

(6) is nothing more than a typed lambda-expression stating that the mapping from pairs of lengths of paths and locations to locations with a certain cognitive or physical relation between them is a sense. The physical relation here is the one of *distance*.

Senses of spatio-temporal prepositions used without measures can be derived from the above senses.

2.1 Representing semantics and metaphor

Compositional rules combining syntax and semantics serve to fill in the proper arguments. Unfortunately, this paper leaves no space to describe the details concerning the syntax. However, I shall give a brief outline of the metaphor representation system implemented in a feature-structure calculus compatible with HPSG[2]

The tricky part is to give a formally operational definition of the notion of domains. I shall just sketch how the domains work in this paper.

Domains are not parts of semantics, but structured concepts to which the lexical semantics must adhere. Domains contain conditions that domain members must fulfill, and mappings between linguistically stated relations between items belonging to the domain and real-world relations. Domains are arranged in multiple inheritance hierarchies. This calls for an example.

[1](4b) suffers from the same structural ambiguity. Since the event only occurred once, it is not decidable whether one thinks of the time or the place of the event.

[2]The semantics is slightly deviant from the HPSG standard. For simplicity, it is assumed that a suitable representation of the interpretable semantics is a string of λ-expressions to be evaluated by a grammar-external device. The semantics of a phrase is built by concatenating the semantics of its constituents.

(8)
$$\begin{tikzcd}[column sep=small]
 & {}_{\textit{1-dimension}}\left[\begin{array}{ll}\text{ORDER} & \top \\ \text{INTERVAL} & \top\end{array}\right] \arrow[dl, no head] \arrow[dr, no head] & \\
{}_{\textit{1-dim-space}}\left[\begin{array}{ll}\text{ORDER} & \lambda ab.a <_{spatial} b \\ \text{INTERVAL} & \lambda abc.|a -_{spatial} b| = c\end{array}\right] & & {}_{\textit{time}}\left[\begin{array}{ll}\text{ORDER} & \lambda ab.a <_{temporal} b \\ \text{INTERVAL} & \lambda abc.|a -_{temporal} b| = c\end{array}\right]
\end{tikzcd}$$

Domains are represented as typed feature structures. The above definition (8) ensures — due to a general wellformedness criterion on typed feature structures — that all subdomains of *1-dimension* have the features ORDER and INTERVAL. The values of these features are interpretation functions relevant to the individual domains. The actual interpretation functions are here represented as λ-abstractions of logical expressions with domain-specific operators and real-world arguments. These arguments are projections of locations and times on domain-specific scales. The example is simplified, the interpretation functions should be typed λ-expressions.

The isomorphy crucial to the metaphorical sense extensions is represented in the wellformedness criterium to which the typed feature structures must conform. That is, metaphors exploiting 1-dimensional structure are functions between subdomains of the *1-dimension* domain.

The lexical entry of **før** is stated in (9). The DOMAIN feature constrains the semantic content of **før** to a function relevant to a particular domain. I have (arbitrarily) stated that the relational content of the lexical entry must adhere to the domain *time*. Structure-sharing (by the index $\boxed{1}$) implies that the value of the CONTENT feature is $\lambda abc.a <_{temporal} b$.

ARG1 and ARG2 contain the semantics of the subject and object of the preposition, respectively. We shall presently look away from the inconsistency regarding that subjects of temporal prepositions usually denote events, and the proposed representation is some representation of a moment.

The metaphor is represented as an HPSG-style *lexical rule* in (10). (10) applied to (9) yields the derived lexical entry (11), which contains the semantics of the spatial metaphorical use of **før**.

(9)
$$\left[\begin{array}{ll}\text{PHON} & \textbf{før} \\ \text{SEM} & \left[\begin{array}{ll}\text{DOMAIN} & {}_{\textit{time}}\left[\begin{array}{ll}\text{ORDER} & \boxed{1}\end{array}\right] \\ \text{CONTENT} & \boxed{1}\,\boxed{2}\,\boxed{3} \\ \text{ARG1} & \boxed{2} \\ \text{ARG2} & \boxed{3}\end{array}\right]\end{array}\right]$$

(10)
$$\left[\text{SEM} \mid \text{DOMAIN} \quad {}^{\textit{1-dimension}}\right] \Rightarrow \left[\text{SEM} \mid \text{DOMAIN} \quad {}^{\textit{1-dim-space}}\right]$$

(11)
$$\left[\begin{array}{ll}\text{PHON} & \textbf{før} \\ \text{SEM} & \left[\begin{array}{ll}\text{DOMAIN} & {}_{\textit{1-dim-space}}\left[\begin{array}{ll}\text{ORDER} & \boxed{1}\lambda ab.a <_{spatial} b\end{array}\right] \\ \text{CONTENT} & \boxed{1}\,\boxed{2}\,\boxed{3} \\ \text{ARG1} & \boxed{2} \\ \text{ARG2} & \boxed{3}\end{array}\right]\end{array}\right]$$

The use of the type system ensures a parsimonious description, because the features values specific to every subdomain and common to various subdomains are stated only once, and because the domains and subdomains are common to several lexical entries.

The definition of the domain type system is in fact a cognitive theory itself, because it has to reflect the semantic features to which metaphor is judged to apply.

It stands as a link between the pure linguistic semantics and the truth-conditional interpretation functions by which the language describes the world.

As a first approximation of a theory of lexical metaphor, all lexical metaphors must be represented as instances of the lexical rule (12). The condition ensures that the domains α and β have one or more features in common, to secure isomorphy.

(12) $[\text{SEM} \mid \text{DOMAIN} \quad \alpha] \Rightarrow [\text{SEM} \mid \text{DOMAIN} \quad \beta]$, $\mathit{features}(\alpha) \cap \mathit{features}\beta \neq \emptyset$.

The use of HPSG-like lexical rules in computer implementations is not unproblematic for known reasons, but such issues are outside the scope of the current treatment.

The arbitrary choice of basic sense of **før** has no ramifications for the account. If one chooses the abstract *1-dimension* as the basic domain for **før**, metaphors derived from (12) do the job of deriving the specific spatial and temporal senses. I am currently ignorant of how a theory should respond to these questions.

3 Sense extension in grammar

The highly polysemic Danish preposition **med** — largely translatable as English 'with' — is used in several syntactic contexts with different senses attached to it. The semantics is traditionally accounted for by positing either homonymy or combining very different semantic content with each syntactic construction. The latter approach is motivated by the fact that one typical sense of **med** resembles the INSTRUMENTAL sense of the instrumental case in e.g. Slavonic languages. Often, if one sense is grammaticalized in one influential language, the scientific community will soon regard that sense as primitive in the semantic description of a number of other languages, whether or not that particular sense plays a role in the individual grammar. Instrumentality is not attested as a semantic element in Danish morphology. One cannot derive the other senses of **med** from the instrumental sense, so if the lexeme has one basic sense, it cannot be the instrumental sense.

Introspectively and by examining parts of a Danish written language corpus [Ber],I state the following senses.

(13)

a	Han slog hende med en hammer 'He hit her with a hammer'	Instrumental
b	Han læssede vognene med ost 'He loaded the trucks with cheese'	Incremental theme
c	Han vendte tilbage med succes 'He returned with success'	Circumstantial subjective predicate
d	Han øvede med sangerne[3] 'He had the singers practice' (lit. 'He practiced with the singers')	Causative
e	Han gik til bal med sine venner 'He went to a ball with his friends'	Coagentive
f	konen med æggene 'The woman with the eggs'	Control, Proximity
g	pigen med det lyse hår 'The girl with the blond hair'	Part/Whole
h	Dansen med den slemme pige 'The dance with the naughty girl'	(Deverbal) Coagentive
i	Ulykken med den væltede tankvogn 'The accident with the overturned tanker'	Unspecified participation

The upper block (13a-e) shows use of **med** in prepositional phrases in verb-modifying positions, typically in the final part of clauses. The lower block shows (13f-i) prepositional phrases modifying a noun, i.e., contributing to a noun phrase.

The clausal uses of **med** often serve to express the state of a secondary agent to be related to the expressed action. This is quite contrary to the use of **med** in noun phrases, where control or contiguity between things is expressed.

But, the exceptions show that the interpretive choice between the event-participative and the noun-modifying sense is semantic rather than syntactic. In (13c) the object of the preposition clearly modifies the subject of the phrase, although it can also be said to describe the circumstances of the event. The fact that 'success' cannot really be counted as a participant triggers the adnominal sense.

In (13i), on the other hand, a tanker can only metaphorically be said to be 'a part of' an accident. The tanker is rather interpreted as participating in an unspecified role of the event of an accident. I cannot say whether the tanker caused the accident or its turning over was caused by it.

Med with deverbal nouns clearly can function as controlling the original prepositional object as in (13h). The particular sense is the same as if the preposition had been controlled by the original verb.

The instrumental sense is special, because the 'instrument' is often both participating (non-voluntarily) in an event and a part of the agent. That is the case when the agent uses a part of his body as instrument, as in (14).

(14) Hun sparkede professoren med sin højre fod
'She kicked the professor with her right foot'

But what is expressed as new information is not that the foot is a part of the person, but that the foot is that part of the person which is utilized to perform the action. Thus, the controlling sense is the most important one.

Basically, the senses of **med** fall into two groups: Participating in events and describing physical objects.

Thus, superficially **med** is a preposition very different from the well-studied spatial ones, which keep their spatial sense in the adnominal use as well as in the adverbial and verbal-complementary uses.

3.1 Participative and adnominal senses

Interestingly, the predicative senses of **med** can be singled out by substituting [... A ... med B] with [B har A] — in English: [B 'has' A].

The rest of the senses are participative in the sense that they relate a *secondary participant* to the expressed action, which I will describe with semantic roles of a 'chain of action' as in [Cro93] and others.

The question is, of course, what has the 'have' relation got in common with the relation of a secondary participant.

The link between the senses becomes more obvious if one employs the notion of *accompaniment*. I shall shortly list some important circumstances of the participating senses.

Coagentivity Secondary participants often occur together with agents, which either control them or perform the action in company with them. Thus, the secondary participant can be part of the 'agentivity side' in the activity. When the secondary participant is coagentive with the agent (which is expressed as subject or object), the secondary participant is undergraded only for pragmatic reasons.

[3]This construction also has the Coagentive sense.

The expressed agent has no control of the secondary participant, and the agent and secondary participant are interchangeable.

The accompaniment relation between agent and secondary participant is clearly a meronymic relation between groups of individuals, and as such a subcase of a general PART-OF relation.

Instrumentality and incremental themes When the secondary participant is controlled by the agent and cannot be ascribed intentionality, it is harder to employ the notion of being part of the active side. The control part is more important than contiguity. Being an instrument also involves that employing the instrument for the intended purpose actually forms a part of the action. Thus, 'He hit her with a hammer' and 'The man with the hammer hit her' differ exactly on this point. The instrument interpretation is triggered by the fact that the prepositional phrase is grammatically linked to a verb which is lexically specified for taking instruments of the relevant kind. '?He read the paper with a hammer' is not that reasonable.

The semantic role of *incremental theme* is mentioned in [Dow91]. Traditionally, separate accounts have been made of instrumentality and incremental themes, though it is very difficult to single out the differences between them. The verbal semantics decide whether a secondary participant is an instrument or an incremental theme. They have the same primitive semantic properties with respect to the agent, i.e., they are controlled by the agent and often propelled by the agent. This position is supported by the ambiguity of 'They loaded the trucks with shovels'.

The distinction between instrumentality and incremental thematicity is a lexical matter of the controlling verb, and I have not yet fulfilled the major lexical-semantic task of partitioning the verbs according to this distinction.

3.2 Causativity

The causative **med** constitutes a special problem. [Han71] states an extreme example:

(15) Jensen er nede i postkassen med et brev
litt. 'Jensen is down in the mailbox with a letter'
meaning 'Jensen performs an action making the letter go into the mailbox'

The secondary participant is simply undergoing the trajectory described in the predicate, leaving the agent with the only function of causing the event, if one does not consider the unlikely interpretation where the agent physically ends up in the mailbox.

The secondary participant overtakes the part of the agentive role from the agent, which the agent cannot possibly fulfill. This is a very special subcase of coagentivity, but it is governed by principles not related to the lexical semantics of **med**.

3.3 Adnominal senses

The important features in adnominal modification are contiguity, part/whole and control.

Typically the prepositional subject cognitively controls the prepositional object, as in (13f). Otherwise physical contiguity or attachment is present, as in (13g). In such cases the prepositional object is often part of the prepositional subject.

In cases when no control is involved, subject and object can be interchanged. This is a very fundamental issue that applies also to the coagentive cases. The use of **med** in this case is simply pragmatically determined, and has the effect of undergrading the entity chosen as the prepositional object.

I am working on a more detailed account of these issues.

3.4 The lexical entry of med

I state the basic lexical sense of **med** as follows:

(16) $([\lambda(a : \text{entity}).\lambda(b : \text{entity}).a \ \mathit{controls}\ b \vee \mathit{contiguous}(a,b)] : \text{sense})$

The sense of **med** is the disjunction of a cognitive relation of *control* and a physical relation of *contiguity*. It is yet to be determined how control is to be accounted for. As for contiguity, a formal treatment of contiguity is stated in [Aur91].

The sense in (16) is capable of encompassing events, when the subject argument (the event) is type-raised in the style of Pustejovsky. The event must simply be type-raised to yield the agent of the event.

3.5 Representing med in a typed feature structure system

I shall give an overview of the implementation of **med** in a system like the one sketched in section 2.1.

To maintain a unified account of the semantics of adjuncts to clauses and nouns compatible to HPSG [PS94], let us assume that the basic template for preposition semantics is as in (17). The compositional semantics of the adjunction is the conjunction of the semantics of the head ($\boxed{2}$) (or subject) and the semantics of the adjunct. Let us furthermore assume that the basic lexical entry of **med** is as (18), which is a particular instance of (17). The somewhat clumsy λ-expression owes its disgrace to the general adjunction account. The entry in (18) is the one that accounts for adjunction to nouns. This very simple treatment does not account for more subtle differences between senses of **med**. It does not rely on any metaphorical description. Thus, the domain stated in (18) is just an all-purpose domain containing typical relations among animate and inanimate physical objects. Again, all typing in the interpretational λ-calculus is abstracted from. We shall not deal further with the actual interpretation functions. The variables ν_1, ν_2 are purely symbolic devices which feed the arguments with indices $\boxed{2}$ and $\boxed{3}$ into both interpretation functions $\boxed{1a}$ and $\boxed{1b}$.

(17)
$$\left[\begin{array}{ll}\text{SEM} & \left[\begin{array}{ll}\text{CONT} & \boxed{2} \wedge \boxed{1}\,\boxed{2}\,\boxed{3}\\ \text{ARG1} & \boxed{2}\\ \text{ARG2} & \boxed{3}\end{array}\right]\end{array}\right]$$

(18)
$$\left[\begin{array}{ll}\text{PHON} & \textbf{med}\\ \text{SEM} & \left[\begin{array}{ll}\text{DOMAIN} & \left[\begin{array}{ll}\text{ACCOMPANIMENT} & \boxed{1a}\\ \text{CONTROL} & \boxed{1b}\end{array}\right]_{\mathit{phys\text{-}obj}}\\ \text{CONT} & \boxed{2} \wedge \lambda\nu_1\nu_2.(\boxed{1a}\nu_1\nu_2 \vee \boxed{1b}\nu_1\nu_2)\ \boxed{2}\ \boxed{3}\\ \text{ARG1} & \boxed{2}\\ \text{ARG2} & \boxed{3}\end{array}\right]\end{array}\right]$$

When **med**-phrases describe events, a metonymic sense extension function must be applied to raise the agent semantics up from the semantics of the subject clause. Such a sense extension function can be represented as in (19).

(19) $$\left[\begin{array}{ll}\text{SEM} & \left[\begin{array}{ll}\text{CONT} & \boxed{2} \wedge \boxed{1}\,\boxed{2}\,\boxed{3} \\ \text{ARG1} & \boxed{2}\left[\text{AGT}\ \boxed{4}\right]\end{array}\right]\end{array}\right] \Rightarrow \left[\text{SEM} \mid \text{CONT}\quad \boxed{2} \wedge \boxed{1}\,\boxed{4}\,\boxed{3}\right]$$

This metonymic function seems applicable with other prepositions. One can imagine other metonymic functions. To account for the ambiguity of (20) related to the Danish preposition **i**, one needs a function which assumes e.g. a recipient role, and type-raises its semantics.

(20) Han sendte hende et brev i Stockholm
'He sent her a letter in Stockholm'

4 Conclusion

The treatment of some lexical semantic properties as a very restricted part of metaphoric and metonymic theory accounts for some obvious problems in the current state of lexical semantics. This can to a large extent be done with Pustejovsky-style type-raising functions. Making the cognitivist assumptions more precise opens the possibility of covering much larger aspects of posited language systems than are usually done in computational linguistics.

Accounting for metaphors as a part of a NLP system seems to be valuable mainly in the parsing realm. Devices of the sketched kind will probably be judged as highly over-generating. This is due to the current public expectations to the language use in computer systems. Most creative metaphorical language is highly stilistically marked, and it is hard to imagine any commercial need for a system generating metaphorical language.

References

[Aur91] M. Aurnague. *Contribution à l'étude de la sémantique formelle de l'espace et du raisonnement spatial : la localisation interne en français, sémantique et structures inférentielles.* PhD thesis, L'université Paul Sabatier de Toulouse, 1991.

[Ber] H. Bergenholtz. DK87-90. Corpus.

[CB95] A. Copestake and T. Briscoe. Sense extensions and the lexicon. *Journal of Semantics*, 1995.

[Cro93] W. Croft. Case marking and the semantics of mental verbs. In J. Pustejovsky, editor, *Semantics and the Lexicon.* Kluwer Academic Publishers, Dordrecht, 1993.

[Dow91] D. Dowty. Thematic proto-roles and argument selection. *Language*, 1991.

[Han71] Erik Hansen. Jensen er nede i postkassen med et brev. *Danske Studier*, pages 5–36, 1971.

[Jac91] R. Jackendoff. Parts and boundaries. *Cognition*, 41:9–45, 1991.

[Lak94] G. Lakoff. The contemporary theory of metaphor. In A. Ortony, editor, *Metaphor and Thought.* Cambridge University Press, 2 edition, 1994?

[PS94] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar.* University of Chicago Press, Chicago, 1994.

[Pus91] J. Pustejovsky. The generative lexicon. *Computational Linguistics*, 17(4), 1991.

An Application of Inside-out Functional Uncertainty to Anaphora Resolution

Kjetil Strand
University of Oslo
Department of Linguistics
Postboks 1102 BLINDERN
0317 OSLO
NORWAY
e-mail: kjetil.strand@ilf.uio.no

Abstract

For some time now, it has been known that in unification grammars we can use *functional uncertainty* to model certain linguistic phenomena (Kaplan and Zaenen 1989). Dalrymple et al. (1990) introduced the notion of *inside-out functional uncertainty*, and showed how this concept could account for the description of syntactic constraints on anaphoric binding. So far, however, no method for computing inside-out functional uncertainty equations has been described in the literature. This paper presents an algorithm and a Prolog implementation for the computation of a subset of equations involving inside-out functional uncertainty. To illustrate the details, the method is applied to resolution of the Norwegian long-distance reflexive [seg]. Prolog is well suited to model the inside-out functional uncertainty in question, although it is not a functional programming language. The main reasons for this are the use of logical variables, the inherent searching behavior of the Prolog machine, and the backtracking to alternative continuations while failing.

1 Inside-out functional uncertainty

An LFG grammar for a particular language yields a complete and coherent functional structure for a single sentence if the sentence is well-formed according to the annotated phrase structure rules and the lexical entries for the respective words occurring in that sentence. LFG makes use of a finite set of grammatical functions to give a functional description of an utterance. If we present the functional structure as a directed graph, it is possible to reach all the relevant syntactic information for each phrase in the utterance via the edges labelled with these grammatical functions. Focusing only on the functional information, a representation of the sentence (1) will yield a graph like that in figure 1.

(1) Hans$_i$ håpet at Jon$_j$ ville be Sylvi$_k$
forsøke å få Ola$_l$ til å tenke på seg$_?$.

Hans$_i$ hoped that Jon$_j$ would ask Sylvi$_k$
to try to make Ola$_l$ think of himself/herself$_?$

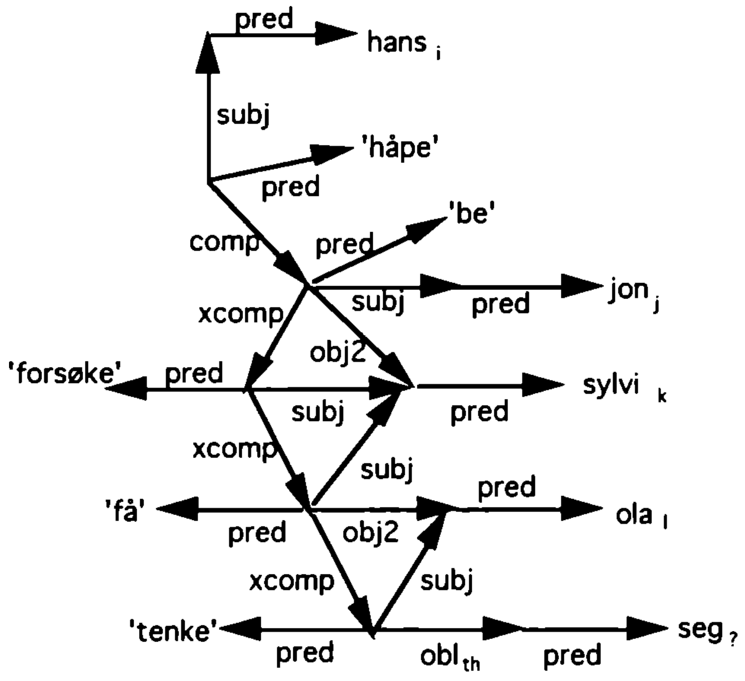

Fig. 1: A graph for the sentence in (1).

In LFG, it is convenient to let the arguments in the functional equations denote a set of paths over grammatical functions. Such a set can be described by a regular expression with the grammatical functions as alphabet. Functional uncertainty has been used to account for long distance dependencies (Kaplan and Zaenen 1989), quantifier scope (Halvorsen and Kaplan 1988) and modelling of syntactic constraints on anaphoric binding (Dalrymple et al. 1990, Dalrymple 1993). In the latter two cases, the notion of *inside-out functional uncertainty* is used. Given a functional description of a sentence, we can draw a picture of the binding relation as in figure 2.

The two vertical strokes are meant to model the variable point in the graph at which the actual binding domain for the anaphor in question starts. The path from the global f-structure and all the way into the anaphor is the concatenation of pre_path and path_out.

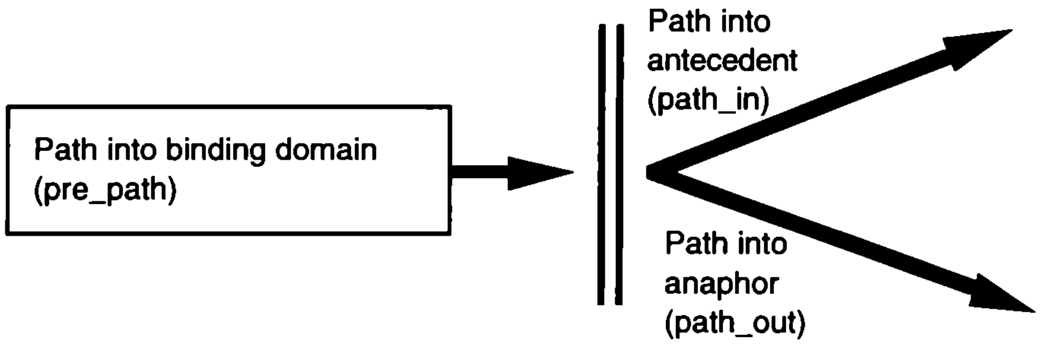
Path into binding domain (pre_path)
Path into antecedent (path_in)
Path into anaphor (path_out)

Fig. 2: The uncertainty point in the functional graph

The uncertainty point in the graph can vary for the same anaphor. This depends on the syntactic constraints on the binding domain. In figure 1 we will have more than one of these points. If the path_out is described by a regular expression like XCOMP^{+} : (ADJ) : OBJ | OBJ2 | OBL_{θ}[1], and the total path from the global f-structure and into the anaphor is COMP: XCOMP : XCOMP : XCOMP : OBL_{th}, we will have three possible uncertainty points in this graph. The notion of functional uncertainty is due to this variation, and the notion "inside- out" is due to the fact that the uncertainty is rooted at the f-structure of the anaphor.

2 Descriptions containing uncertainty equations

Dalrymple et al. (1990) propose an equation like (2) to model the anaphoric relationship between antecedent and anaphor:

(2) $$<\sigma>((\text{PathOut} \uparrow\text{A})\ \text{PathIn}) \quad = \quad <\sigma>\uparrow\text{A}$$

$<\sigma>$ represent the mapping between syntax and semantics, $\uparrow$A the f-structure of the anaphor, (PathOut $\uparrow$A) picks out the set of f-structures that contain the anaphor and in which the antecedent must be located, and PathIn characterizes the set of possible paths into the antecedent from these domains. The equation should be read as follows: There should exist an f-structure *b* from which there is a path in the set of strings PathOut leading to $\uparrow$A, and from which an antecedent f-structure *ant* is reachable via a path in the set of strings PathIn, and *ant* and $\uparrow$A should map to the same semantic

[1] In this notation, ":" means concatenation, "*" means zero, one, or more repetitions, "+" means repetition one or more times, "|" means disjunction and round parentheses means optionality.

projection. The equation could very well be satisfiable in more than one way, depending on the choice of paths from the sets PathOut and PathIn, respectively. But the contribution of (2) to the global functional description, is that we stick to one of these ways in the final representation.

In this paper, I express the anaphoric relationship through unification of the AGR features in the f-structure. These AGR features project from the lexical entries of nominal heads, and consist in turn of the index, gender, number and person features.

Norwegian has a rich inventory of reflexives. They comprise 1) [seg selv], which has to be bound to a subjective noun phrase in the minimal nucleus ([+sb], [+ncl], in the LFG terminology (Sells 1985)); 2) [seg], which has to be bound to a subjective noun phrase outside the minimal nucleus, but inside the minimal finite tensed domain ([+sb], [-ncl]); and 3) [ham selv], [henne selv], [den selv] and [det selv], which have to be bound in the minimal complete nucleus, but at the same time disjoint from the subjective phrase in this domain ([-sb], [+ncl])[2]. If the PathOut for [seg] is stated as $(GF - COMP)^+ : OBJ \mid OBJ2 \mid OBL_\theta$[3], and the PathIn as SUBJ | POSS, the equation in (2) may be expressed in the following way when applied to [seg]:

(3) $(((GF - COMP)^+ : OBJ|OBJ2|OBL_\theta : \uparrow) SUBJ|POSS:AGR) = (\uparrow:AGR)$

Both to make (3) more readable and to foresee some of the implementational matters, I divide (3) into a conjuction of equations. To achieve this, we introduce existential quantified variables ranging over f-structures. The equation in (3) will thus be transposed to the three equations in (4), where *b* and *ant* are existentially bound f-structure variables.

(4)
$$\begin{aligned} b : (GF - COMP)^+ : OBJ \mid OBJ2 \mid OBL_\theta &= \uparrow \\ b : SUBJ \mid POSS &= ant \\ ant : AGR &= \uparrow : AGR \end{aligned}$$

The resolution process amounts to instantiating the index value of the anaphor (or of sharing the variable index with the antecedent). Reflexives are also often underspecified regarding the other morphosyntactic features in the AGR feature (e.g., [seg] is underspecified with respect to syntactic gender and number). Unification of the AGR features will thus, under normal circumstances, add information to the linguistic description of the anaphoric phrase, in addition to the index feature.

---

[2] This analysis of the Norwegian anaphors has been questioned, e.g., by Lødrup (1985). In this paper I will stick to the analysis given by Sells (1985). The parts of this analysis relevant for the implementation described here, is also supported by Hellan (1988) and Dalrymple (1993).

[3] As in Dalrymple et al. (1990) I take GF to denote the set of grammatical function labels.

Under the lexical entry for [seg] is also inserted the constraint in (5)[4]:

(5) $\neg \; (\, ant : \mathrm{GF}^* \; = \; \uparrow \,)$

The "non containment condition" in (5) says that no path (including the null path, i.e., identity) exists between the antecedent and the anaphor f-structures. This ensures the first requirement of *f-command*: "For any occurrences of the functions α, β, in an f-structure F, α *f-commands* β if and only if α does not contain β and every f-structure of F that contains α contains β" (Bresnan 1982; 334).The PathIn in the second line of (4) is the disjunction SUBJ I POSS, which means that this path has length one. This ensures the other requirement of f-command (Dalrymple 1993; 156).

The problem with (5) is that it involves *universal* quantification over paths in the set of strings GF*. This is the effect of negating an equation involving functional uncertainty (Dalrymple 1993; 123). Such equations only make sense if related to completed f-structures, where they will be evaluated as true or not. This is accounted for in LFG by treating negation nonconstructively (Kaplan and Bresnan 1982; 210, Dalrymple 1993; 123). The last equation is thus only **constraining**, i.e., it will be checked for satisfaction in a complete and coherent f-structure.

The equations in (4) and (5) have to be further elaborated to account for all the syntactic constraints on anaphoric binding.The [+sb] anaphors must be bound inside the minimal tensed domain. This means that no intervening f-structure in the PathOut should contain the feature TENSE. This can be expressed in the following way[5]:

(6)
$$\neg \; \left[\begin{array}{lcl} int : \mathrm{GF}^+ & = & \uparrow \\ b : \mathrm{GF}^+ & = & int \\ & int : \mathrm{TENSE} & \end{array} \right]$$

The [-ncl] feature states that all non-reflexive arguments inside the minimal nucleus should be disjoint from the anaphor in question. As (7) shows, it is perfectly possible for the [-ncl] anaphor [seg] to corefer with a *reflexive* argument inside the minimal nucleus:

[4] The constraints in (5) and (6) are assumed to be expressed as conjuncts to the existential quantified constraints in (4), so the variables *ant* and *b* will be properly bound.

[5] Dalrymple (1993; 136) uses the following notation to express the constraint in (6):

$$((\mathrm{DomainPath} \quad \mathrm{GF} \quad \uparrow) \; \mathrm{AntecedentPath})_\sigma \quad = \quad \uparrow_\sigma$$
$$\neg \, (\text{->} \; \mathrm{TENSE})$$

This should be read as follows: None of the f-structures that is picked out along the path DomainPath, should contain the feature TENSE. The equation in (6) involves universal quantification of the variable *int*, as the negation sign has wider scope.

(7) Martin$_i$ ba oss snakke til seg$_i$ om [seg selv]$_i$/seg$_i$.
Martin$_i$ asked us to talk to him$_i$ about himself$_i$.[6]

The minimal nucleus is the minimal f-structure containing both the anaphor and a feature PRED, that is, no intervening f-structure between this f-structure and the anaphor should contain PRED. In this domain all the non-reflexive co-argumenting f-structures should have an index disjoint from the index of the anaphor. This so-called *co-argument disjointness condition* can be stated with the help of a constraint like that in (6).

3 The inside-out algorithm

Kaplan and Maxwell (1988) showed that the verification problem for equations containing outside-in functional uncertainty was trivial, while the satisfiability problem was decidable in the acyclic case. Whether these results hold also for equations involving inside-out uncertainty, is not obvious. Imagine, for instance, the case where an equation enables us to build ever more comprehensive f-structures by adding ADJ on our way out. This cancels the property of *rootedness* which is usually presupposed for linguistic descriptions.

In the application considered in this paper, however, this problem need not arise. This will be evident by a closer inspection of the equations comprising inside-out functional uncertainty, and in particular the first two equations in (4). None of these equations could be **defining**, using the LFG terminology, in that they allow information to be added in a monotonic way during construction time. The antecedent *ant* for an intrasentential anaphor always has to be realized by other means in the global f-structure, either as an element subcategorized for by an existing PRED, or otherwise realized in terms of a projection from the c-structure. This is also the case for the f-structure representing the binding domain *b*, and for all grammatical functions in PathOut and PathIn. Only the third equation in (4), unifying the AGR features, adds information, in that the anaphor is attached with its antecedent by sharing of index values. Thus only the latter equation is defining in the LFG sense. The consequence of this argument, is that we can treat the inside-out uncertainties with respect to the final coherent and complete f-structure for the sentence as a whole. For this application we only have to consider the grammatical functions in the final global representation as candidates for possible steps in the uncertainty paths.

The algorithm is called IO, as it is based upon a true, "inside-out" recursive traversal of a finite graph. Input to the algorithm are the global f-structure FS, the f-structure for the anaphoric element ***ana*** and the regular expression ***reg_exp*** describing the PathOut from ***ana*** to the possible

[6] This example is from Dalrymple (1993; 150), where it is used to illustrate binding asymmetries.

binding domains. We start by assigning FS to the variable parameter ***fstruc***. Then we traverse into ***ana***, one step (***gf***) at the time. At each step ***gf*** we instantiate the corresponding part of ***path*** by concatenating ***gf*** in front of the variable ***path*** rest. On each corresponding step during withdrawal we check for a match between ***reg_exp*** and ***path***, and try to resolve the anaphor if we have a match. Output is either success, with the AGR feature of ***ana*** unified with the antecedent, or failure.

Above, we stated that ***reg_exp*** in the [seg] case could be described by the regular expression (GF - COMP)$^+$: OBJ I OBJ2 I OBL$_\theta$. Only f-structures at this "distance" from ***ana*** should be taken into account as possible binding domains. If some suffix of ***path*** satisfies this description at all, we will be in one of three situations on our way out of the graph, starting from ***ana*** at the innermost level:

1) We have not reached the variable point in the graph where ***path*** matches ***reg_exp***. In this case ***path*** should be a suffix of ***reg_exp***. (e.g., ***path*** = OBL$_{th}$).

2) We have a situation where ***path*** matches ***reg_exp***. (e.g., ***path*** = XCOMP : XCOMP : OBL$_{th}$).

3) We have passed the variable point where ***path*** matches ***reg_exp***. (e.g., ***path*** = COMP: XCOMP : XCOMP : XCOMP : OBL$_{th}$).

These three situations can be illustrated as in figure 3:

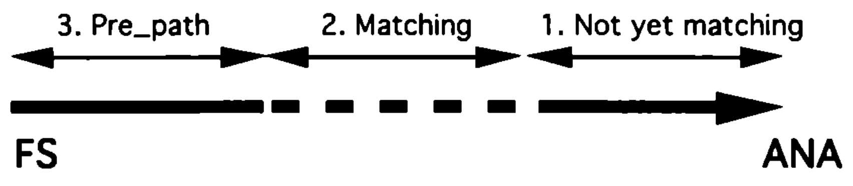

Fig. 3: The three possible situations

The matching process can be done on each level while withdrawing in the recursion. To keep track of which of the three situations we are in, we introduce two flags: ***match?*** and ***resolved?***. These flags can be in one of two states: **true** or **false**. Initially, and in situation 1 above, they are both **false**. From the first level of matching an onwards (situation 2) ***match?*** = **true**. We should try to resolve the anaphor only on this level. If we succeed in resolving the anaphor, ***resolved?*** is set to **true**, and all higher levels of IO succeed without any further ado. If we are in situation 2, and no longer have a match between ***path*** and ***reg_exp***, IO should fail on this level.

Now we are in a position to describe the algorithm:

IO 1. [We have reached the anaphor]
If ***fstruc*** is identical with ***ana***, terminate with ***path*** set to the empty path.

IO 2. [Search a step further down in the f-structure]
Follow a grammatical function ***gf*** from ***fstruc***. Call the f-structure which ***gf*** picks out for ***temp***. Set ***path*** to the concatenation of ***gf*** and the variable ***path_rest*** and call IO recursively with ***temp*** and ***path_rest***, and the other parameters unchanged. Let us assume that we seek depth first. If ***fstruc*** contains no grammatical function, continue in the last possible alternative continuation. Eventually we will reach the anaphor ***ana***.
On each level on the way back in the recursion we do the following:

a) If ***resolved?*** is **true**, terminate with success.

b) Make a resolution try if
   1) the flag ***resolved?*** is **false** and
   2) we are in a legal binding domain (there is a match between ***path*** and ***reg_exp***, and ***match?*** is set to **true**)

   If the resolution succeeds, set ***resolved?*** to **true**. In any case, terminate with success (if resolution fails on this level, we should anyway try on a higher level).

c) If ***match?*** is **true**, and we don't have a match between ***path*** and ***reg_exp***, terminate with failure (we are in situation 3 above, and have not succeeded in resolution of the anaphor inside the legal binding domain).

d) Otherwise, if ***path*** is a suffix of ***reg_exp***, terminate with success.
   (Both ***resolved?*** and ***match?*** are **false**. This means that we have not yet reached a legal binding domain on our way out)

e) Otherwise, terminate with failure. (No suffix of this path between FS and ***ana*** will ever satisfy ***reg_exp***.)

4 Advantages using Prolog

There are two obvious advantages in using Prolog to implement the described algorithm[7]. First, there is the depth-first searching machine inherent in the proof resolution of Prolog. This can be utilized in step IO2, where we pick out a reachable f-structure ***temp***, following an edge labelled ***gf*** from ***fstruc***. Prolog will search through the graph until the anaphoric element is found, without any special machinery.

The search mechanism of Prolog has also one further advantage. Backtracking to the last alternative continuation will give us all possible solutions, one at a time, in a "don't know" indeterministic way. This goes together well with the notion of functional uncertainty.

Second, the logical variables of Prolog are utilized in the recursive call on IO with the variable dynamic parameter ***path_rest*** in IO2. This parameter will be instantiated through unification if the goal succeeds. We would have difficulties in modelling this dynamic instantiation with the same elegance and descriptive power in any other programming language.

The Prolog variable, preliminarily distributed as the value of the IND feature from the lexicon, ensures that structures sharing this variable will continue to share this property, regardless of the course of the processing history: syntactic and semantic analysis, noun phrase processing including anaphoric resolution, foci and knowledge base updating, etc.

Indexing is done on the functional description of the clause. But as the representations of the discourse referents share the same Prolog variable as value of the index feature both in the functional and the semantic representation, the instantiation will affect all structures simultaneously and in the same way.

Sometimes we try to unify two AGR features where neither have an instantiated index. This happens if an anaphor has a pronominal as its antecedent. If the unification succeeds, the two structures in question will share the same Prolog variable as value of the IND feature. When the pronominal is resolved later on, both structures will be instantiated with the same index value.

Logical variables can also be utilized to model the two flags in IO. Both flags are first **false**, then eventually **true**, but never changing back to **false** again. This we can model in Prolog, letting **false** be a logical variable, and **true** the constant **true**. We check the value of a flag asking if it is a variable

[7] I have troughout this paper considered *acyclic* f-structures only. Jan Tore Lønning (p.c.) has pointed out to me, that Prolog is not the best programming language for representing cyclic graphs, in that it is impossible for a Prolog variable to contain itself. The algorithm presented in the preceding section might handle cyclicity by naming the f-structures in the path, checking in each step that we do not pass through the same f-structure twice. The interaction of cycles with uncertainty paths may pose other problems, however.

or not. Once instantiated to **true**, a flag will keep this value in the actual environment.

5 The Prolog code

The program consists of two predicates: inside_out/6 and check/6. Inside_out/6 has two entries, while check/6 has three entries. The arguments in both cases come in this order: ***fstruc***, ***ana***, ***path***, ***reg_exp***, ***match?***, ***resolved?***.

```
inside_out(Ana, Ana, [], RegExp, Match, Resolved) :- !.

inside_out(FS, Ana, [GF | Path], RegExp, Match, Resolved) :-
                follow(FS, GF, Temp),
                inside_out(Temp, Ana, Path, RegExp, Match, Resolved),
                check(FS, Ana, [GF | Path], RegExp, Match, Resolved).
```

The flags (and the resolution of the anaphor) are handled inside the check/6 goal. If Resolved is set to **true** on an earlier level, we continue to withdraw:

```
check(_, _, _, _, _, Resolved) :- nonvar(Resolved), !.
```

We are in situation 2: We have a match between Path and RegExp. We set Match to **true**, and try to resolve the anaphor. If resolution succeeds on this level, Resolved is set to **true**, otherwise it remains a variable. In any case, the goal will succeed, so we can continue to withdraw:

```
check(FS, Ana, Path, RegExp, Match, Resolved) :-
                                match(Path, RegExp),
                                !,
                                Match = true,
                                resolve(FS, Ana, Path, Resolved).
```

We are in situation 1: Both Resolved and Match are variables, and we do not have a match yet. The Path so far is a suffix of one of the described paths in RegExp. The check/6 goal succeeds, and we continue to withdraw:

```
check(_, _, Path, RegExp, Match, _) :-
                                var(Match),
                                suffix(Path, RegExp).
```

This procedure tries out a solution on the level nearest to the anaphor first. If this solution is in conflict with other constraints, we back-track to the continuation inside entry 2, where Resolved remains a variable. On the successive levels, entry 2 will be evoked as long as we have a match.

If we reach situation 3 without any resolution (Match is **true**, and Resolved is still a variable) check/6 fails on this level, and due to this, inside_out/6 fails on the same level.

If we reach a situation where both Resolved and Match are still variables, and Path neither matches RegExp nor a suffix of RegExp, check/6 fails immediately.

To give a flavor of the approach taken, I include an example of the resolve/4 goal in the [seg] case:

```
resolve(FS, Ana, _, Resolved) :-
                    (follow(FS, subj, Ant); follow(FS, poss, Ant)),
                    not(contained(Ana, Ant)),
                    Ant : agr === Ana : agr,
                    Resolved = true.

resolve(_, _, _, _).
```

The f-command restriction is guaranteed by the first two lines: In line one we follow a path of length one (**subj** or **poss**) to identify the f-structure of the antecedent (Ant), and in line two the goal fails if Ana is contained in (or identical to) Ant. Resolution amounts to unification of the AGR features, and if all these goals succeed, Resolved is set to **true**.

If any of the goals fails, the second entry for resolve/4 succeeds, and the Resolve flag remains a Prolog variable.

6 Conclusion

I have presented a Prolog implementation of inside-out functional uncertainty with an application to intrasentential anaphora resolution. The main predicate inside_out/6 and the subgoal check/6 take care of the binding constraints. It turns out that the inside-out functional uncertainty approach is well suited for an efficient implementation of intrasentential anaphora resolution. This is so because, for this application, we only have to concern ourselves with the f-structures and the grammatical functions legitimated by other defining equations in the linguistic description, as they are projected from the c-structure tree and the lexical entries of the morphemes occuring in the string. Thus we can take the complete and coherent f-structures as input to the resolution algorithm. Although not a functional programming language, Prolog is well suited for implementation of the algorithm in question.

7 Acknowledgements

I am grateful to Mary Dalrymple for stimulating discussions and helpful comments on an earlier version of this paper. I would particularly like to thank Jan Tore Lønning for extensive and very helpful discussion of the issues presented here. I of course take responsibility for all flaws in the reasoning, errors or confusions in the paper.

8 References

Bresnan, Joan 1982: *Control and Complementation.* In Joan Bresnan (Ed.): The Mental Representation of Grammatical Relations. Cambridge, Massachusetts: The MIT Press. (pp. 282 - 390)

Dalrymple, Mary 1993: *The Syntax of Anaphoric Binding.* CSLI Lecture Notes. Number 36. Stanford.

Dalrymple, Mary, John Maxwell and Annie Zaenen 1990: *Modeling Syntactic Constraints on Anaphoric Binding.* In Proceedings of COLING 90, Volume II. Helsinki. (pp. 72 - 76).

Halvorsen, Per-Kristian and Ronald M. Kaplan 1988: *Projections and Semantic Description in Lexical-Functional Grammar.* In Proceedings of the International Conference on Fifth Generation Computer Systems. Tokyo, Japan. Edited by ICOT. (pp. 1116 - 1122).

Hellan, Lars 1988: *Anaphora in Norwegian and the Theory of Grammar.* Foris, Dordrecht.

Kaplan, Ronald M. and Joan Bresnan 1982: Lexical-Functional Grammar: A Formal System for Grammatical Representation. In Joan Bresnan (Ed.): The Mental Representation of Grammatical Relations. Cambridge, Massachusetts: The MIT Press. (pp. 173 - 281).

Kaplan, Ronald M. and John Maxwell 1988: *An Algorithm for Functional Uncertainty.* In Proceedings of COLING 88, Volume I. Budapest. (pp. 297 - 302).

Kaplan, Ronald M. and Annie Zaenen 1989: *Long-distance Dependencies, Constituent Structure, and Functional Uncertainty.* I Baltin, M. and Kroch, A. (Eds.): Alternative Conceptions of Phrase Structure. Chicago University Press. (pp. 17 - 42).

Lødrup, Helge 1985: *En note om seg og seg selv.* (A note on *seg* and *seg selv*). In Skriftserie No. 21, University of Bergen.

Sells, Peter 1985: *Lectures on Contemporary Syntactic Theories.* CSLI Lecture Notes. Number 3. Stanford.