# 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

Proceedings of NODALIDA 1995, pages 69-80

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$_?$*

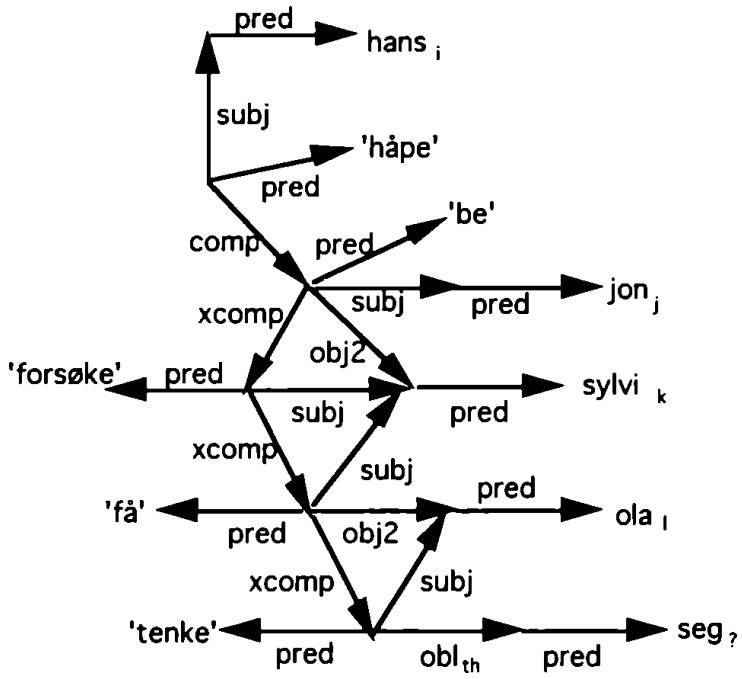


Fig. 1: A graph for the sentence in (1).

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.

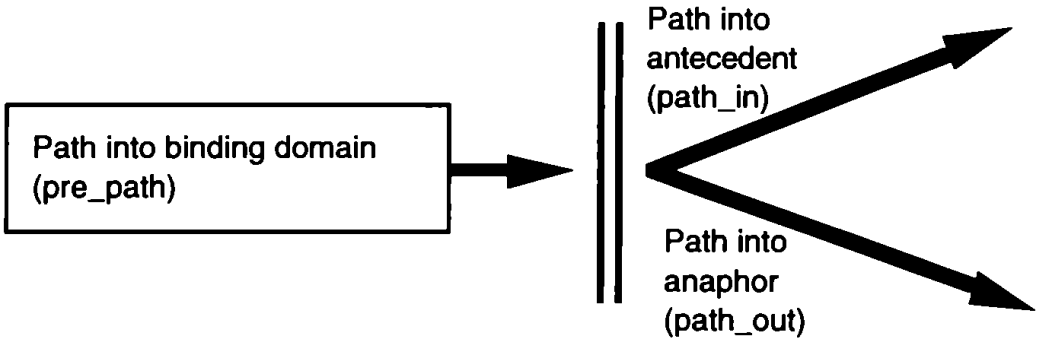


Fig. 2: The uncertainty point in the functional graph

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

<σ> represent the mapping between syntax and semantics, ↑A the f-structure of the anaphor, (PathOut ↑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 ↑A, and from which an antecedent f-structure *ant* is reachable via a path in the set of strings PathIn, and *ant* and ↑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) $\neg \quad ( \; ant : \mathrm{GF}^* \; = \; \uparrow \; )$

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 | 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) $\neg \quad [ \quad int : \mathrm{GF}^+ \; = \; \uparrow$
$b \; : \mathrm{GF}^+ \; = \; int$
$int : \mathrm{TENSE} \quad ]$

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):

$((\mathrm{DomainPath} \quad \mathrm{GF} \quad \uparrow) \; \mathrm{AntecedentPath})_\sigma \quad = \quad \uparrow_\sigma$
$\neg$ (-> 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 | OBJ2 | $OBL_{\theta}$. 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:

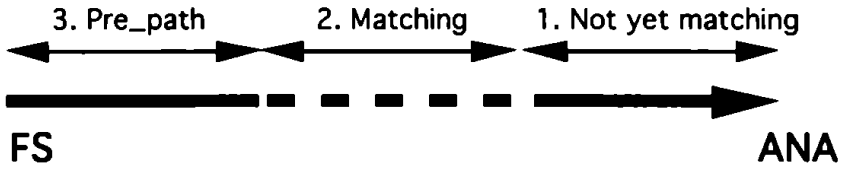


Fig. 3: The three possible situations

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.