Incremental Syntactic and Semantic Processing

Esther Konig*
Department of Electrical Engineering
Kyoto University, Sakyo-ku,
Kyoto 606, JAPAN
e-mail: esther@kuee.kyoto-u.ac.jp

Abstract

The construction of the semantic representation for a natural language sentence or a piece of discourse cannot be covered by the so-called "compositional semantics" alone. In the general case, the non-compositional construction steps of generating quantifier scoping and of anaphora resolution have to be included. In order to filter out unnecessary information as soon as possible it is desirable to merge these three phases into one processing step. We describe how the rules for "extended compositional semantics" as presented in ([Pereira, 1990]) can be integrated into a parser for categorial grammar. The inspection of the data flow shows where concurrency can come into play.

1 Introduction

The construction of a semantic representation for a natural language sentence involves at least the three steps of

1. "compositional" semantics (constructing a semantic form by the use of β-conversion)
2. determining the scopes of the quantifiers
3. anaphora resolution

The most obvious sequential order of the three steps, which causes the least technical problems for the implementation, is the one given above (cf. [Kamp and Reyle, 1991, Asher and Wada, 1988]). But there are also proposals to merge these operations during processing. [Johnson and Klein, 1986] describe how discourse representation structures (DRS's, see [Kamp and Reyle, 1991]) are constructed during the syntactic processing and how anaphora resolution can be integrated into this construction process[1]. However, they ignore the problem of generating different quantifier scopings. In [Johnson and Kay, 1990], the second phase of the construction of semantic representations (i.e. the generation of quantifier scopings) has been included into the construction procedure where a Definite Clause Grammar (see e.g. [Clocksin and Mellish, 1987]) serves as the syntactic basis.

Pereira describes the integration of compositional semantics, quantifier scope generation, and anaphora resolution on an abstract level [Pereira, 1990]. The key idea is that not only one half of a functional calculus should be used, which is the functional application operation (β-conversion) but that the other half, i.e. functional abstraction deserves to be taken under consideration, as well. Functional abstraction, so to speak, allows for hooking up the compositional semantics with the contextual information. The context of an utterance can be considered as a set of additional assumptions which help to derive the current sentence [Pollack and Pereira, 1988]. Additional assumptions have to be "discharged" after successful derivation in order to perform a sound reasoning. The discharging of assumptions is exactly the task of functional abstraction. Thus, a broadened view on compositionality can be taken. In particular, Pereira's suggestions imply that the mechanisms for dealing with contextual information no longer are mere procedural appendage of the logically well-understood step of compositional semantics but that these mechanisms are part of the logical system which specifies how to construct a semantic representation for a sentence (or a text).

In the following, we investigate how Pereira's suggestions can be implemented in the framework of the currently fashionable lexicalized approaches to grammar [Pollard and Sag, 1987, and many others], in our case a basic categorial grammar[2].

*This research has been made possible by a Monbusho-scholarship (Japanese Ministry of Education). I wish to thank Makoto Nagao, Yuji Matsurnoto, and the two anonymous referees. The responsibility for errors resides with me.

[1] Similar approaches have been implemented by other people, see e.g. [Geurts, 1990].

[2] Basic categorial grammars are a subsystem of Combinatory Categorial Grammars, see e.g. [Steedman, 1989].

$$(app_{\backslash}) \quad \frac{y : S_a \qquad (x\backslash y) : S_f}{x : (S_f S_a)} \qquad (app_{/}) \quad \frac{(x/y) : S_f \qquad y : S_a}{x : (S_f S_a)}$$

$$(lic_p) \quad \frac{pron : u}{np : u} \qquad (lic_q) \quad \frac{np_q : S \qquad quant : \langle u, S \rangle}{np : u}$$

$$(abs_p) \quad \frac{\begin{array}{c} quant : \langle u_q, S_q \rangle \qquad pron : u_p \\ \vdots \qquad\qquad \vdots \\ x : S \end{array}}{x : [u_p/u_q]S} \qquad (abs_q) \quad \frac{\begin{array}{c} quant : \langle u, S_q \rangle \\ \vdots \\ x : S \end{array}}{x : S_q(\lambda u.S)}$$

Figure 1: *Construction calculus*

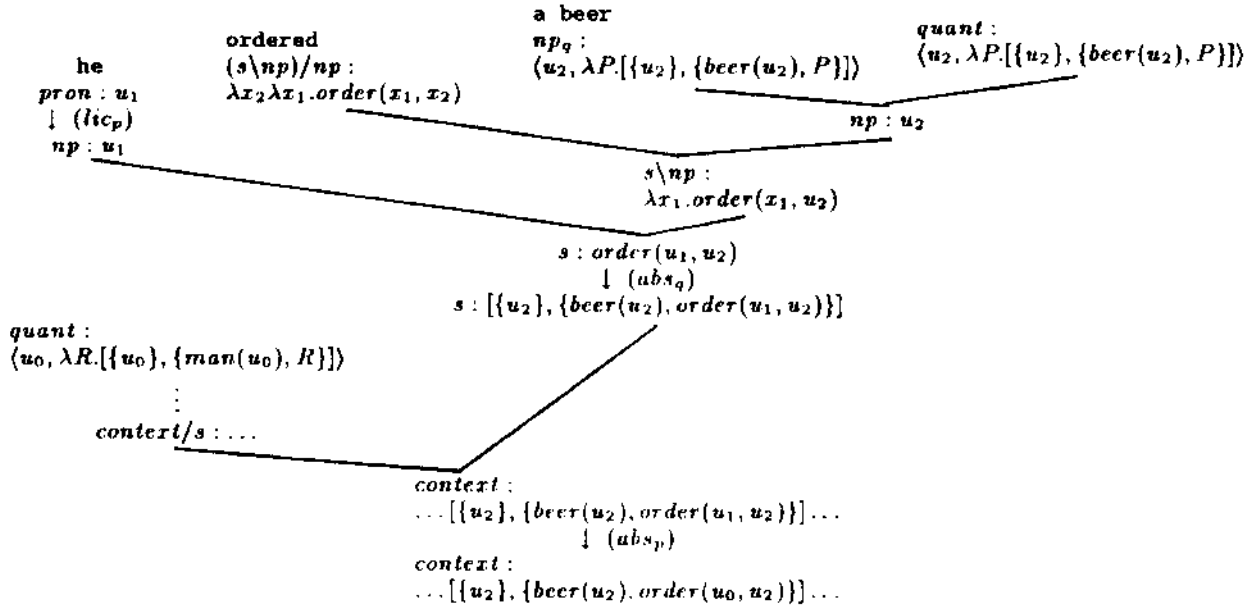

Figure 2: *Sample derivation*

2 The construction calculus

An abbreviated version of Pereira's construction calculus is given in figure 1 in a modified notation which allows to turn it easily into a parsing algorithm. Thus, instead of semantic types, we use syntactic ones (i.e. those of categorial grammar). We assume that a proposition is a pair which consists of a syntactic category x_i and a (partial) semantic representation S_i (a special case are variables u_i). The symbols *quant* and *pron* are distinguished categories which designate "stored" quantified expressions and pronouns respectively[3]. The partial semantic representation of a quantified phrase is prefixed by the variable u_i which is bound by its main quantifier.

The *application rules (app\)* and *(app/)* perform the purely compositional part of the construction of a semantic **representation. The *quantifier licensing rule (lic$_q$)*** "stores" the semantic form of a quantified noun phrase as a hypothetical proposition with category *quant*. In the *quantifier abstraction rule (abs$_q$)* such a hypothesis is discharged, i.e. the quantified expression is removed from the store and applied to an intermediate parse result. The *pronoun licensing rule (lic$_p$)* transforms a pronoun into a regular noun phrase which can be used in the subsequent derivation. The initial pron-proposition is considered as a hypothetical one, which has to be discharged by applying the *pronoun abstraction rule (abs$_p$)* to perform anaphora resolution. A derivation is successful if all stored quantified expressions and all pronouns have been discharged. For more details see [Pereira, 1990],

For the sake of simplicity we assume that pronouns are the only kind of anaphoric expressions, and that anaphora can only take quantified expressions as their antecedents.

Figure 2 shows the derivation of the sentence **"he ordered a beer"** under the assumption that the preceding sentence reads "a man entered." Instead of constructing a traditional predicate logic formula, a discourse representation structure (DRS) is built. A DRS is either a pair of a list of discourse referents and a list of conditions, or a two-place term with functor "⇒" and two DRS's as arguments. Intuitively, an undischarged quantified expression corresponds to what is meant by

$$(app_{\backslash}) \quad \frac{[y : S_a]\ Q_1 \qquad [(x \backslash y) : S_f]\ Q_2}{[x : (S_f S_a)]\ Q_1, Q_2} \qquad (app_{/}) \quad \frac{[(x/y) : S_f]\ Q_1 \qquad [y : S_a]\ Q_2}{[x : (S_f S_a)]\ Q_1, Q_2}$$

$$(lic_p) \quad \frac{[pron : u]\ Q}{[np : u]\ Q} \qquad (lic_q) \quad \frac{[np_q : S]\ Q}{[np : u]\ Q, quant : \langle u, S \rangle}$$

$$(abs_p) \quad \frac{\begin{array}{c}[pron : u_p]\ Q_p \\ \vdots \\ [x : S]\ Q, quant : \langle u_q, S_q \rangle\end{array}}{[x : [u_p/u_q]S]\ Q, quant : \langle u_q, S_q \rangle} \qquad (abs_q) \quad \frac{[x : S]\ Q, quant : \langle u, S_q \rangle}{[x : S_q(\lambda u.S)]\ Q}$$

Figure 3: *Modified construction calculus - accumulation method*

the term *accessible discourse referent* in Discourse Representation Theory. If a quantified expression is still undischarged at a certain node in a derivation tree, then its scope covers at least the phrase at the leaves of this subtree.

3 Implementation issues

In the following, we explain how the construction calculus can be implemented. Nevertheless, we will stick to the natural deduction format to represent various possible algorithms. Because of the similarity of natural deduction rules and Horn clauses (several premises and one conclusion), a Prolog implementation can be derived in a straightforward way.

The construction calculus in definition 1 is not yet a specification of an efficient algorithm. A theorem prover which is derived from this definition has to exploit the pecularities of the individual rules in order to increase the efficiency. One of its main tasks is to keep track of the additional propositions which are brought into the derivation by the licensing rules. We will focus on the handling of the set of undischarged quantified expressions. In order to ease the access, they have to be put into a separate data structure.

3.1 Accumulation method

The most obvious method is to simply accumulate the information about undischarged quantified expressions starting from the leaves of the tree. For this purpose, propositions are now indexed by a set Q of undischarged quantified expressions (or to be more efficient, by a set of pointers to undischarged quantified expressions):

$$[x : S]\ Q$$

Set union and set membership is indicated by a comma. Lexical entries are assumed to come with empty sets. In the application rules the quantifier sets of the two daughter are simply combined. The quantifier licensing rule adds a quantified expression to the set, the quantifier abstraction rule removes an element. The pronoun manipulation rules do not change the set. Figure 3 shows the revised calculus.

The suggested flow of data makes clear that under certain circumstances anaphora can be resolved *before* the quantifier scopes in a sentence have been completely determined. In the example

> "Every man who loves a woman follows her to some country."'

the pronoun "her" may find its antecedent before the relative scopes of the quantifiers "every" and "some" are known.

3.2 Threading method

If we assume that we are dealing with anaphora only, i.e. that cataphora do not occur, a further change of the data structure will allow for a slight simplification of the calculus. Anaphora want to know their left context, therefore it is sufficient to collect information from left to right in the well-known manner of information "threading". Information is not only accumulated from the leaves towards the root of a derivation tree, but propagated down to the leaves which are located further right than the nodes where the information was originated.

Each constituent can be characterized by the states of the set of accessible quantified expressions before and after it has been derived. Hence, each proposition will be indexed by two sets Q_1 ,Q_2 *of* undischarged quantified expressions:

$$Q_1\ [x : S]\ Q_2$$

The set Q_1 is the *input set,* and Q_2 the *output set.* The similarity to difference lists in Prolog is not accidental. For lexical entries, the input and the output sets are equal. The input set of the first word of a sentence might either be empty or equal the output set of the preceding sentence. The newly revised calculus is presented in figure 4. An application rule connects the output set of its left premise with the input set of the right premise. The output set of the conclusion of the quantifier licensing rule contains one element more than the output set of its premise; for the quantifier abstraction rule, the inverse relation holds. The rules *(lic_p)* and *(abs_p)* would read in the "threading" notation:

$$(lic_p) \quad \frac{Q_1\ [pron : u]\ Q_2}{Q_1\ [np : u]\ Q_2}$$

$$(abs_p) \quad \frac{\begin{array}{c}Q_1\ [pron : u_p]\ Q_2 \\ \vdots \\ Q_3\ [x : S]\ Q_4, quant : \langle u_q, S_q \rangle\end{array}}{Q_3\ [x : [u_p/u_q]S]\ Q_4, quant : \langle u_q, S_q \rangle}$$

$$(app_{\backslash}) \quad \frac{Q_1\,[y : S_a]\,Q_2 \qquad Q_2\,[(x\backslash y) : S_f]\,Q_3}{Q_1\,[x : (S_f S_a)]\,Q_3} \qquad (app_{/}) \quad \frac{Q_1\,[(x/y) : S_f]\,Q_2 \qquad Q_2\,[y : S_a]\,Q_3}{Q_1\,[x : (S_f S_a)]\,Q_3}$$

$$(pron) \quad \frac{Q_1, quant : (u_q, S_q)\,[pron : u_p]\,Q_2}{Q_1, quant : (u_q, S_q)\,[np : u_q]\,Q_2} \qquad (lic_q) \quad \frac{Q_1\,[np_q : S]\,Q_2}{Q_1\,[np : u]\,Q_2, quant : \langle u, S\rangle}$$

$$(abs_q) \quad \frac{Q_1\,[x : S]\,Q_2, quant : \langle u, S_q\rangle}{Q_1\,[x : S_q(\lambda u.S)]\,Q_2}$$

Figure 4: *Modified construction calculus - threading method*

However, a quantified expression can only qualify as a potential antecedent of an anaphor if it occurs in the left context of the anaphor, i.e. if the quantified expression is a member of the input set of the anaphor. Hence, the pronoun can access directly the set of its potential antecedents without making reference to a larger subtree of the derivation. The combined rule must perform the two tasks of the original rules *(lic_p)* and *(abs_p)* which are the formation of a result proposition *(np : u)* and the substitution of the variable u_p by the variable u_q:

$$(pron) \quad \frac{Q_1, quant : \langle u_q, S_q\rangle\,[pron : u_p]\,Q_2}{Q_1, quant : \langle u_q, S_q\rangle\,[np : u_q]\,Q_2}$$

The modified rule works fine on the logical level. It will cause problems with the processing, because, in general, the input set will not be instantiated at the moment when the pronoun is encountered by the parser. The search in the set of possible antecedents has to be *frozen* as long as the variable which represents the set is still uninstantiated. The task of waiting until the appropriate part of the derivation has been built up is delegated to an underlying mechanism instead of mentioning it explicitly in the rule *(abs_p).* In the framework of concurrent logic programming languages, which provide a suspension mechanism, the set of undischarged quantified expressions may be considered as a *stream.*

Since we presented the algorithm in the natural deduction format, one might assume that there are the usual *structural rules* silently working in the background. However, for a parser, the presence of the structural rules is harmful. Propositions should not be arbitrarily omitted, duplicated or permuted. The *weakening rule* which allows for the omission of premises will be never needed unless vacuous abstraction is considered a legal phenomenon. In the case of the original construction calculus, the task of the *permutation rule* is to guarantee the arbitrary access to the *quant-* and pron-propositions. A first step has been done by representing the "syntactic" propositions and the purely semantic ones in two different data structures. In the threading method, the need for a separate data structure for the undischarged pronouns has disappeared, since pronoun licensing and pronoun abstraction have been merged into one rule. Only for the handling of the list of undischarged quantified expressions, the permutation rule must be available in order to turn the list into a set. Hence, the permutation rule need not be used in the calculus itself. The *contraction rule* does its work tacitly to prepare the way for applications of the rule *(pron).* Obviously, a quantified expression might serve as the antecedent of several pronouns. This is the only the case where the use of a structural rule cannot be eliminated. But fortunately, by coupling it with the responsible rule *(pron),* the properties of the parser are not affected and the number of occurrences of the contraction rule is restricted to the number of pronouns in a sentence or a text.

The limited use of the contraction rule (i.e. copy rule) implies that the global stream of undischarged quantified expressions is finite. In addition, the stream is acyclic because it represents a traversal of an acyclic tree structure. All frozen tasks (of pronoun resolution) are arranged along this finite, acyclic stream, hence there will be neither the trap of an infinite search space nor the danger of a deadlock. Thus, the algorithm is quaranteed to terminate for any input.

3.3 Concurrent parsing

Concerning the processing methods which have been presented in the last sections, full search for all possible derivations is most easily performed by using the Prolog backtracking mechanism. This leads to two known drawbacks. Firstly, usually a lot of redundant work has to be done because the backtracking mechanism does not allow to reuse successful subderivations. Secondly, since all data is local to one proof structure, there is hardly any basis to attempt several proofs in parallel. Standard solutions to these problems exist already in the form of table-based parsing, or chart-parsing. For Prolog implementations, the more conventional approaches realize the chart as a set of Prolog data base entries, see e.g. [Pereira and Shieber, 1987].

For a parsing method which also handles the processing of contextual information, the Prolog data base approach becomes quite expensive. Every time when an application step has been carried out, the data base entries which represent the nodes in the right daughter tree have to be exchanged in order to update their input sets. If one adopts, however, the somewhat more unconventional approach of parsing with *nested streams,* cf. [Okumura and Matsumoto, 1987, Matsumoto and Sugimura, 1987], the destructive operation of updating the context lists is done inexpensively by the Prolog unification operation.

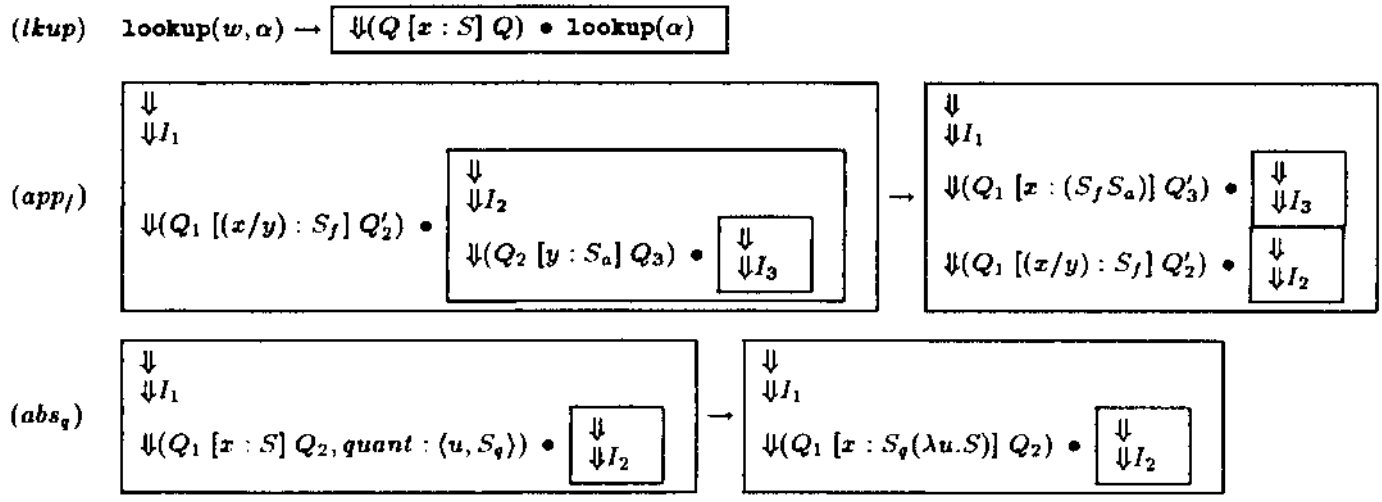

Figure 5: Stream-based syntactic and semantic processing (analoguous rule variants are omitted)

The propositions which cover portions of the derivation starting at the same position in the sentence form a stream I_1. Furthermore, each proposition is connected to the stream I_2 of its alternative right neighbor constituents:

$$\{(Q_1\ [x : S]\ Q_2) \bullet I_2\}, I_1$$

or more pictorially:

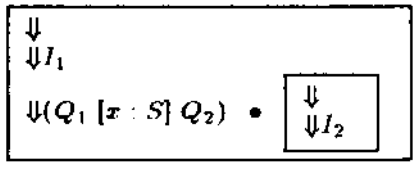

Figure 5 shows the rules of the stream-based parsing algorithm. The rule (lkup) realizes the lexical lookup of for a word w with category x and its right neighbor string (x. The applications rules do their usual work. However, one important feature of the Prolog data base approach must be simulated here. Each data-base lookup of an entry returns a new copy of this entry. Looking up a proposition in the stream, however, returns the identical proposition for all accesses. In order to avoid variable clashes, the new proposition $Q_1\ [x : (S_f S_a)]\ Q'_3$ must result from combining a c $Q'_2\ [y : S_a]\ Q'_3$ f the proposition $Q_2\ [y : S_a]\ Q_3$ with the functor proposition. This kind of copying does not lead to the appearance of two copies of the same proposition in one (virtual) construction calculus derivation. The use of copies is reserved for alternative derivations, only[4]. The rules (pron), (lic_q), and (abs_p) add new elements to the stream. Since this happens "spontaneously", the appropriate control structure must be provided to let these rules apply at most once to each stream element, e.g. by creating a process per rule and per stream.

[4]A more comprehensive argument for the soundness of this style of copying with regard to a chart parsing algorithm which handles hypothetical propositions has been given in [Konig, 1990].

4 Conclusion

Based on Pereira's semantic construction calculus [Pereira, 1990], we have defined algorithms which perform the three phases of the construction of a semantic representation (composition, quantifier scoping, and anaphora resolution) in an interwoven manner. The investigation of the data flow lead to a stream-based algorithm which allows for a high degree of AND-parallel processing.

In previous approaches, at least the anaphora resolution module had to access the semantic form, cf. [Johnson and Kay, 1990]. Now, the semantic construction operations work independently of the underlying semantic representation. The only link between the construction algorithm and the specific semantic representation is the A-conversion operation, which could be replaced by other, possibly weaker constructors, e.g. based on unification.

What we have suggested so far is an algorithmic skeleton of extended compositional semantics. This skeleton will only become a full-fledged body if one adds the wealth of heuristics about anaphora resolution and constraints on quantifier scopes. Further investigations concerning efficient structure sharing methods for semantic representations and contextual information are necessary in order to improve the overall performance when parsing highly ambiguous input. The algorithms seem still rather far away from the presumed way of cognitive language processing. Information about possible antecedents for anaphora flows only along the paths of the derivation tree. Improvements should be made towards a word-to-word transport of information which, of course, has to be filtered by the structural constraints.

Based on the notion of extended compositionaiity as it is realized in the construction calculus, even semantic representation languages like Kamp's Discourse Representation Structures which have the reputation of being non-compositional can be handled in a compositional manner.

Constructing semantic representations for a natural language utterance (and interpreting it) has some similarity to the evaluation of a computer program: programs are processed with regard to a given context (i.e. machine state). The similarity between programs and natural language has been observed in the literature, e.g. [van Benthem, 1989], and has given rise to research programs concerning "dynamic logic" for natural language, e.g. [Groenendijk and Stokhof, 1990, Muskens, 1990]. Further investigation of the dynamics of natural language might give insight how to use parallel processing mechanisms for natural language analysis in a more sophisticated way.

References

[Asher and Wada, 1988] Nicholas Asher and Hajime Wada. A computational account of syntactic, semantic and discourse principles for anaphora resolution. Journal of Semantics, 6:309 344, 1988.

[Clocksin and Mellish, 1987] W. F. Clocksin and C.S. Mellish. Programming in Prolog. Springer, Berlin, 3 edition, 1987.

[Gallier, 1986] Jean H. Gallier. Logic for Computer Science. Foundations of Automatic Theorem Proving. Harper and Row, New York, 1986.

[Geurts, 1990] Bart Geurts (ed.). Natural Language Understanding in LILOG: An Intermediate Overview. Technical Report 137, IBM Deutschland GmbH, Institute for Knowledge-Based Systems, Stuttgart, Baden-Wurttemberg, 1990.

[Groenendijk and Stokhof, 1990] Jeroen Groenendijk and Martin Stokhof. Dynamic Predicate Logic. Report 2.1.A, ESPRIT Basic Research Action 3175, Dynamic Interpretation of Natural Language (DYANA), 1990.

[Hobbs and Shieber, 1987] Jerry Ilobbs and Stuart M. Shieber. An algorithm for generating quantifier scopings. Computational Linguistics, 13:47-63, 1987.

[Johnson and Kay, 1990] Mark Johnson and Martin Kay. Semantic abstraction and anaphora. In Proceedings of the 13th COLING, pages 17-27, Helsinki, 1990.

[Johnson and Klein, 1986] Mark Johnson and Ewan Klein. Discourse, anaphora and parsing. In Proceedings of the 11th COLING, Bonn, Fed. Rep. of Germany, 1986.

[Konig, 1990] Esther Konig. Parsing Categorial Grammar. Report 1.2.C, ESPRIT Basic Research Action 3175, Dynamic Interpretation of Natural Language (DYANA), 1990.

[Kamp and Reyle, 1991] Hans Kamp and Uwe Reyle. From Discourse to Logic. Kluwer Academic Publishers, 1991.

[Matsumoto and Sugimura, 1987] Yuji Matsumoto and Ryoichi Sugimura. A parsing system based on logic programming. In Proceedings of the International Conference on Artificial Intelligence, Milano, Italy, 1987.

[Muskens, 1990] Reinhard Muskens. Anaphora and the logic of change. In Jan van Eijck, editor, Proceedings of the European Workshop on Logics in AI, pages 414-430, Springer, Berlin, 1990.

[Okumura and Matsumoto, 1987] Akira Okumura and Yuji Matsumoto. Parallel programming with layered streams. In Proceedings of the Logic Programming Symposium, San Fransisco, CA., 1987.

[Pereira, 1990] Fernando C.N. Pereira. Categorial semantics and scoping. Computational Linguistics, 16(1):1-10, 1990.

[Pereira and Shieber, 1987] Fernando C.N. Pereira and Stuart M. Shieber. Prolog and Natural-Language Analysts. CSLI Lecture Notes, Center for the Study of Language and Information, Stanford, Ca., 1987.

[Pollack and Pereira, 1988] Martha E. Pollack and Fernando C.N. Pereira. An integrated framework for semantic and pragmatic interpretation. In Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics, pages 75-86, Buffalo, New York, 1988.

[Pollard and Sag, 1987] Carl Pollard and Ivan Sag. An Information-Based Syntax and Semantics I Fundamentals. Volume 13 of Lecture Notes, Center for Study of Language and Information, Stanford, Ca., 1987.

[Steedman, 1989] Mark Steedman. Grammar, interpretation, and processing from the lexicon. In W. Marslen-Wilson, editor, Lexical Representation and Process, MIT Press, 1989. (in press 1989).

[van Benthem, 1989] Johan van Benthem. Semantic parallels in natural language and computation. In H.-D. Ebbinghaus, editor, Logic Colloquium 1981, pages 331-375, Elsevier Science Publishers, 1989.