Top-Down Predictive Linking and Complex-Feature-Based Formalisms

James Kilbury
Seminar für Allgemeine Sprachwissenschaft
Heinrich-Heine-Universität Düsseldorf
Universitätsstraße 1
D-40225 Düsseldorf, Germany
kilbury@ling.uni-duesseldorf.de

Abstract

Automatic compilation of the linking relation employed in certain parsing algorithms for context-free languages is examined. Special problems arise in the extension of these algorithms to the possibly infinite domain of feature structures. A technique is proposed which is designed specifically for left-recursive categories and is based on the generalization of their occurrences in a derivation. Particular attention is drawn to the top-down predictive character of the linking relation and to its significance not only as a filter for increasing the efficiency of syntactic analysis but as a device for the top-down instantiation of information, which then serves as a key to the directed analysis of inflected forms as well as "unknown" or "new" words.

1 Introduction

Complex-feature-based formalisms are understood here as equivalent to *unification-based formalisms* as exemplified by PATR-II, HPSG, and others (cf Shieber 1986, Carpenter 1992). Such formalisms typically include a context-free (CF) base, which allows the use of parsing algorithms designed for CF languages despite the fact that complex-feature-based formalisms are essentially more powerful than CF grammars. However, such an adaptation of CF algorithms involves their extension to possibly infinite nonterminal domains, which, as Shieber (1985) and Haas (1989) have shown, is nontrivial.

Various CF algorithms make use of a binary relation between a goal category and the category of a constituent (phrase or word) which either has just been parsed or is to be parsed next. Different terms have been used to designate this relation; Kay (1980) speaks of *reachability*, while Pereira/Shieber (1987) and others before them use the term *linking* for the relation.

Whatever term one takes, an important aspect of the relation is that it can be used to reduce the search space of possible syntactic analyses at an earlier point in parsing and thus serves to improve the efficiency of a parser. Shieber (1985, 1992) follows established terminology in speaking of *top-down filtering* in connection with the *prediction* step of the Earley algorithm. His central notion of *restriction*, whereby a *restrictor* is a finite subset of the paths specified in a feature structure, is related to the technique we introduce here, since both guarantee the finiteness of an otherwise possibly infinite domain of complex categories, but Shieber's restrictors are specified manually.

We propose a general algorithmic method of compilation that avoids manual specification. The focus of this discussion is on the linking relation used to extend left-corner parsers, rather than on the prediction step of the Earley algorithm as with Shieber, although the results carry over.

Whereas Shieber et al. (1990) have discussed similar techniques in the context of semantic-head-driven generation, we are concerned here with parsing. We view the linking relation not simply as a filter to increase efficiency within the domain of syntactic analysis--this aspect is stressed by Shieber (1985) and other investigators such as Bouma (1991)--but rather as a device for the top-down predictive instantiation of information, as Shieber et al. (1990) have shown for semantic-head-driven generation. In this paper we are concerned especially with morphosyntactic information and illustrate the relevance of predictive linking for morphological analysis and for the analysis of "unknown" or "new" lexical items.

2 Left-Corner Parsing and Linking

2.1 The Left-Corner Parsing Algorithm

The so-called *left-corner* (LC) parsing algorithm is generally credited to Rosenkrantz/ Lewis (1970). It has been presented so often since and is now so well-known that a brief informal statement of the algorithm should suffice here:

The algorithm applies to CF grammars in general; it is both correct and, with the exception of derivations of the form $A \rightarrow^* A$, where A is a nonterminal, is complete. It can be used either to compile a given CF grammar into a parser or to interpret it.

The principle is simple. To parse a string, the current word form is first parsed, i.e. looked up in the lexicon. Whenever a constituent, be it a word form or a phrase, is successfully parsed, the syntax rules are chosen which have the category of the identified constituent as their *left corner*, i.e. the left-most category in the right-hand side of the rule. If the remaining sister categories of the left corner can be parsed, then the mother category of the rule is the result for the corresponding substring, and the algorithm continues recursively until the entire string is covered by a category; if the category of an expectation was specified, it must match the category found.

2.2 The Linking Relation

It was soon noted that the efficiency of the algorithm could be improved significantly through the use of a *reachability* or *linking* relation compiled out of the grammar before parsing; this consists of the reflexive and transitive closure of the relation defined by the pairs of mother and LC categories specified in the set of syntax rules of the grammar. Whenever a lexical entry is found that assigns a category C to a given word form, the linking relation is used to determine whether C can be useful in reaching the goal category C', i.e. whether C is a (transitive) left corner of C'; this test is carried out *before* the parser looks for rules having C as the left corner. Of course, C may itself be the category C' sought, hence the reflexive closure. Likewise, when a rule is found in which C is the left corner, the relation tests whether the mother C_0 of C can be used to reach the goal C' before an attempt is made to parse the sisters of C.

Computation of the linking relation from a set of CF syntax rules is straightforward. Since the nonterminal symbols are atomic, one merely needs to check for left recursive symbols so that the computation terminates.

2.3 Complex-Feature-Based Formalisms

As noted above, the extension of the LC algorithm to a potentially infinite nonterminal domain, i.e. complex feature structures, is nontrivial. An example of the pitfalls awaiting naive attempts at such an extension is provided by the grammars illustrating the list technique for subcategorization introduced by Shieber (1986: 32, 77-78); also see the similar example of Haas (1989: 227). We quote Shieber's syntax rules for his second analysis of subcategorization (p.84), in which the subject of a verb appears as the first element of the subcategorization list:

```
S ---> NP VP : <S head> = <VP head>
               <S head form> = finite
               <VP subcat first> = <NP>
               <VP subcat rest> = end.

VP ---> V : <VP head> = <V head>
            <VP subcat> = <V subcat>.

VP_1 ---> VP_2  X :
     <VP_1 head> = <VP_2 head>
     <VP_2 subcat first> =
        <VP_1 subcat first>
     <VP_2 subcat rest first> = <X>
     <VP_2 subcat rest rest> =
        <VP_1 subcat rest>.
```

The difficulties clearly lie in the last syntax rule: *VP* (or *VP_1* and *VP_2*) seems to be left recursive--whatever we may mean by that at this point--so perhaps no link arises at all from this rule. On the other hand, the two feature structures are unifiable, but their unification produces a *cyclic* feature structure, which raises additional problems for the definition of linking and possibly for implementation. The difficulty, of course, is that *VP_1* and *VP_2* are *schematic* and that these rules recursively generate a denumerably infinite set of *VP*-type categories, all of which may give rise to distinct elements in the linking relation. Whether this is linguistically important, which is improbable, or merely a mathematical game is beside the point: the formal problem is there, and we cannot individually specify infinitely many links.

Consider the LC analysis of the sentence *John loves Mary*. After analyzing $[John]_{NP}$, the parser expects a *VP[1]*, where *VP[n]* is used as an informal alias for a *VP* that subcategorizes for *n* complements. Now, there is an entry for $[loves]_V$, so a link <*VP[1]*, *V[2]*> is needed since *loves* subcategorizes for a subject and an object. Indeed, since the grammar allows verbs to subcategorize for any finite number of complements, we need an infinite number of links between *VP[1]* and *V[n]* categories. Moreover, once we have these links we need the same number between *VP[1]* and *VP[n]* categories since the first *VP* expansion simply unifies the subcategorization of the *V* daughter with that of the *VP* mother.

Other problems arise in grammars with *indirect* left recursion. This is linguistically plausible in the following example, where both *NP* and *Det* are indirectly left recursive:

```
S ---> NP VP : <NP agr> = <VP agr>.

NP ---> Det N : <NP agr> = <N agr>
                <NP agr> = <Det agr>.

Det ---> NP Posessive_Marker
```

The rules account for sentences like *The child's father sleeps*. We must take care not to exclude *A mother's children sleep*, which will happen if the linking relation is defined so that any determiner--also in a possessive construction--must have the same agreement features as the sentence subject of which it is a left corner.

2.4 Top-Down Predictive Linking

The aim of our proposal is to define equivalence relations that keep the linking relation finite while also preventing it from being too restrictive; this turns the linking relation into a *weak prediction table* in the sense of Haas (1989: 227ff). Like Shieber (1985, 1992) with the notion of *restriction*, we confine our attention to a subset of specifications; in particular, we can define a feature structure that subsumes all *VP*-type feature structures of Shieber's recursive subcategorization rules. But unlike Shieber, our restrictors are computed automatically by building the generalization of the occurrences of left-recursive categories in a grammar.

The intuitive idea is that we consider categories to be left recursive if their tokens can be unified (rather than being identical, as in the case of atoms); we then use their generalization, or greatest lower bound, as a common denominator defining an equivalence relation.

We shall say that two categories build a left-recursive link, i.e. $\langle X, X' \rangle \in L_1$ iff on the basis of the given grammar there is a derivation $A \rightarrow^* A'\alpha'$ (where α' is a string of categories and terminal symbols) such that the unification $A \sqcup A'$ exists, whether or not it is cyclic, and there is no A'' such that $A \rightarrow^* A''\alpha'' \rightarrow^* A'\alpha'$, where $A \sqcup A''$ and $A'' \sqcup A'$ exist. Let A_{gen} be the generalization $A \sqcap A'$ of A and A'; then we define X and X' as distinct copies of A_{gen} such that for every path π where $A@\pi = A'@\pi$, it also holds that $X@\pi = X'@\pi$, where $F@\pi$ is "the value of the feature structure F at some path π at which it is defined" (Carpenter 1992:38) and '=' denotes token identity. We thus expressly allow reentrancies between the distinct feature structures X and X'; as we shall see below, this is essential in order for us to use the linking relation to instantiate information during parsing.

A second relation, corresponding directly to the conventional notion of links as compiled directly from rules, can now be defined: two categories build a rule link, i.e. $\langle A_0', A_n' \rangle \in L_2$ iff on the basis of the given grammar there is a finite derivation $A_0 \rightarrow A_1\alpha_1 \ldots A_{n-1}\alpha_{n-1} \rightarrow A_n\alpha_n$ with $1 \leq n$ such that for all i with $1 \leq i \leq n$ it is the case that $A_0 \sqcup A_i$ is undefined (i.e. the derivation is nonrecursive), and for all i with $0 \leq i \leq n$ it is the case that if $\langle X, X' \rangle \in L_1$ and $A_i \sqcup X$ exists, then A_i' also exists and is the extension $X' \sqsubseteq A_i'$ that arises when A_i is unified with X. Intuitively, L_2 is given by the set of nonrecursive derivations licensed by the grammar, where each left-recursive left-most category in the derivation is replaced by (i.e. restricted to) the generalization of the instances of that left-recursive category.

The overall linking relation L is then defined so that $\langle B, B' \rangle \in L$ iff (1) there is a $\langle X, X' \rangle \in (L_1 \cup L_2)$ such that $B \sqcup X$ and $B' \sqcup X'$ exist, or else (2) $B \sqcup B'$ exists (the reflexive case where B' satisfies the parse goal B).

Moore and Alshawi (1992: 134ff) describe a similar algorithm that compiles the linking relation for complex-feature-based formalisms. But rather than computing the generalization of left-recursive categories to avoid the possibility of generating an infinite relation, they instead "impose a cutoff after two nested occurrences of the same functor in a feature specification,

substituting new unique variables for the arguments of the inner occurrence of the functor, so that any constituent with a more complex feature description will be accepted." Moreover, they discuss linking only with regard to filtering.

Our use of the linking relation with destructive unification in parsing requires special comment. If B is a goal and B' is a parsed left corner such that $\langle X, X' \rangle \in L$ and $B \sqcup X$ and $B' \sqcup X'$ exist, then there is a link between B and B'; we can stop here with a mere test of unification if we only want to use linking as a filter to reduce the search space. But if B and B' are actually unified with X and X', respectively, information may become shared between B and B'. In the reflexive case of linking for completion, B and B' are unified with each other. Since the information that becomes shared via L_I is subsumed by that of the reflexive case, completion works correctly for left-recursive categories, but B and B' must still be unified in the actual completion step.

We have employed generalization in the definition of linking to make a kind of mask allowing just the appropriate information to become shared between B and B'. Thus, linking ceases to be a mere test or filter and can instead function as an independent device for the transmission of information in parsing.

Returning to Shieber's rules for subcategorization, the definition of L given here allows instantiated head features of a *VP* goal to be passed top-down to a verb. Moreover, since the treatment of subcategorization of Shieber (1986: 84) is adopted in which the subject *NP* appears consistently as the *first* element of the subcategorization list in all projections of *V*, then instantiated agreement features and other information about the subject can be passed top-down as well. The linking relation compiled from the grammar in (2.3) above is listed here:

```
l1([cat:vp,head:A,subcat:[first:B,rest:C]],
   [cat:vp,head:A,subcat:[first:B,rest:D]]).

l2([cat:s], [cat:np]).

l2([cat:vp,head:A,subcat:[first:B,rest:C]],
   [cat:v, head:A,subcat:[first:B,rest:D]]).
```

Finally, we list the linking relation compiled from the example with left-recursive categories *Det* and *NP*:

```
l1([cat:np],  [cat:np]).
l1([cat:det], [cat:det]).

l2([cat:s],   [cat:np]).
l2([cat:np],  [cat:det]).
l2([cat:det], [cat:np]).
l2([cat:s],   [cat:det]).
```

In contrast to the previous example, agreement specifications have been compiled out of the relation, but no additional convention whereby **cat** specifications define a context-free skeleton is involved here.

2.5 Notes on Implementation in Prolog

Implementation of the LC algorithm in Prolog has been discussed by Matsumoto et al. (1982) for the BUP system, by Pereira/Shieber (1987), Kilbury (1990), and Covington (1994). Here we present, with minor changes, the LC-based interpreter with linking for a modified DCG formalism as formulated by Pereira/Shieber (1987: 180ff); note that the interpreter itself is encoded as a DCG:

```
parse(NT) --> leaf(LC, NT), lc(LC, NT).

leaf(Cat,NT) --> [Word],
                 {lex(Word,Cat), link(NT,Cat)}.

lc(LC, NT) --> [], {unify(LC, NT)}.
lc(LC, NT) --> {C0 ---> [C1|Cats],
                  unify(LC, C1),
                  link(NT,C0)},
                 right(Cats),
                 lc(C0,NT).

right([]) --> [].
right([Cat|Cs]) --> parse(Cat), right(Cs).
```

Our version stated here tranfers the call to **link** from the definition of **parse** to that of **leaf**; the motivation for this change stems from our use of top-down information in the morphological analysis and in the treatment of missing lexical entries.

Moreover, our implementation uses the open Prolog lists of Eisele/Dörre (1986) to encode the feature structures of PATR-II (also see Gazdar/Mellish 1989: 228ff). Since simple variable sharing does not capture the unification of these objects, we instead employ **unify/2**.

The transitive subset of the linking relation can be implemented as follows:

```
link(C0,C) :- (l1(X,Y) ; l2(X,Y)),
              unify(X,C0), unify(Y,C).
```

Of course, such clauses lead to a highly inefficient search for Prolog. A better solution, which we have adopted from Kilbury (1990), is to introduce rule numbers, which are then used to define a purely filtering linking relation. This amounts to the simplest case of the restriction technique of Shieber (1985). Only when a link between numerical pointers is first found is the linking relation between feature structures used to instantiate information.

3 Consequences of Predictive Linking

What is the advantage of predictive linking as discussed above in (2)? In the previous parsing literature attention has been drawn mainly to linking as a *filter* employed to reduce the search space as early as possible in a syntactic analysis. But we have seen that linking as defined above in terms of feature structures and used in parsing with destructive unification leads to the top-down instantiation of information. This has far-reaching consequences for the analysis of inflectional morphology and lexical items for which no entry at all or no adequate entry is found in the parser's lexicon. We briefly address these areas separately.

3.1 Inflectional Morphology

If we ignore capitalization, the German word form *runden* 'round' can be assigned the category *N*, *A*, or *V*; the corresponding inflected forms are too numerous to list here but include e.g. accusative plural for *N*, genitive singular weak inflection for *A*, and first person plural present indicative for *V*. The facts of German inflection lead to massively disjunctive analyses in conventional systems of morphological analysis that simply take an isolated inflected word form and consider what it might be. But if we are given a top-down prediction or expectation of the lexical category from the linking relation, then we can first find a lexical entry for the stem, use linking to confirm the appropriateness of the entry (in our example, the appropriateness of the category *N*, *A*, or *V*) for the context, simultaneously use linking to further instantiate features (in particular for agreement) on the category, and then check the given inflection within a highly restricted search space. This captures our intuition that an inflection like *-n* in German in itself bears practically no information and is functional only because we normally have expectations and can greatly reduce the range of inflections possible in a given context.

3.2 Analysis of Unknown Words

A fundamental issue for parsing lies in the area of "unknown" or "new" words. This involves cases where no entry at all is found for a given word form, but also cases where an entry for the form is found, which, however, does not fit the given context; the missing lexical entries may simply have been omitted from the lexicon of a system, or may reflect novel lexical creations. The theoretical and practical significance of such unknown forms is great; see the discussion in Kilbury/Barg/Renz (1994).

Even without the linking relation, unification, together with backtracking through a space of possible analyses (or corresponding use of a chart) gives us information for the missing entry; this in itself is not novel. But simple extensions of well-known parsing algorithms presuppose a *random* search through the space of open lexical classes to get the candidate category. The procedure is roughly as follows: (1) once it is established that no appropriate entry is in the lexicon, (2) *arbitrarily* select a category for an open lexical class, (3) check whether it fits the given context, and (4) if it fits, take the final feature structure for the form which is instantiated in the course of a successfully completed parse.

In contrast, top-down predictive linking provides for a *directed* search since top-down instantiation can *propose* a category. The efficiency can be further increased by partitioning the linking relation according to lexical and phrasal categories for the left corner and, among the lexical left corners, open and closed lexical classes.

In implementation, missing lexical entries can be dealt with in a first approximation by extending the interpreter with a second clause in the definition of **leaf**:

```
leaf(Cat,NT) --> [Word],
   {(setof(C,lex(Word,C),Cs), !; Cs = []),
    \+ (member(Old,Cs), link(NT,Old)),
    open_lexical_link(NT,Cat),
    new_word(Word,Cat)}.
```

Obviously, more needs to be said about the control strategy of the modified interpreter since garden paths and structural ambiguity must be dealt with before new entries are postulated, but that goes beyond the scope of this paper.

4 Conclusion

We have presented a technique based on the operation of generalization that provides for the automatic computation of a linking relation for use with complex-feature based grammar formalisms of the kind that are currently favored in computational linguistics. Although our proposal was developed for left-corner parsing with backtracking, it obviously carries over to other parsers with top-down prediction. A further goal of this paper is to motivate a shift of attention from linking as a mere filter to genuinely predictive linking as a valuable device for the top-down transport of information essential for other aspects of natural language parsing.

Acknowledgements

The research project *Dynamische Erweiterung des Lexikons* (DYNALEX) is supported by the German Research Foundation (DFG) in the Sonderforschungsbereich 282 *Theorie des Lexikons*. I wish to thank Petra Barg, Christof Rumpf, Sebastian Varges and anonymous referees for their comments and suggestions.

References

Alshawi, Hiyan (ed.) (1992) *The Core Language Engine*. Cambridge, Mass. & London: MIT Press.

Bouma, Gosse (1991) Prediction in Chart Parsing Algorithms for Categorial Unification Grammar, *Proceedings of the 5th EACL Conference*, 179-184.

Carpenter, Bob (1992) *The Logic of Typed Feature Structures*. Cambridge: CUP.

Covington, Michael A. (1994) *Natural Language Processing for Prolog Programmers*. Englewood Cliffs: Prentice-Hall.

Eisele, Andreas / Jochen Dörre (1986) A Lexical Functional Grammar System in Prolog, *Proceedings of COLING-86*, 551-553.

Gazdar, Gerald / Chris Mellish (1989) *Natural Language Processing in Prolog: An Introduction to Computational Linguistics*. Wokingham et al.: Addison-Wesley.

Haas, Andrew (1989) A Parsing Algorithm for Unification Grammar. *Computational Linguistics* 15: 219-232.

Kay, Martin (1980) *Algorithm Schemata and Data Structures in Syntactic Processing* (= Report CSL-80-12). Palo Alto: Xerox Corp.

Kilbury, James (1990) QPATR and Constraint Threading, *Proceedings of COLING-90*, Vol. 3, 282-284.

Kilbury, James / Petra Barg / Ingrid Renz (1994) Simulation lexikalischen Erwerbs, S. W. Felix / Chr. Habel / G. Rickheit (eds), *Kognitive Linguistik: Repräsentation und Prozesse*, 251-271. Opladen: Westdeutscher Verlag.

Matsumoto, Y. / H. Tanaka / H. Hirakawa / H. Miyoshi / H. Yasukawa (1983) BUP: A Bottom-Up Parser Embedded in Prolog. *New Generation Computing* 1: 145-158.

Moore, Robert C. / Alshawi, Hiyan (1992) Syntactic and Semantic Processing, in Alshawi (1992), 129-148.

Pereira, Fernando C. N. / Stuart M. Shieber (1987) *Prolog and Natural-Language Analysis* (= *CSLI Lecture Notes* 10). Stanford: CSLI.

Rosenkrantz, D. J / P. M. Lewis (1970) Deterministic Left Corner Parser, *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata Theory*, 139-152.

Shieber, Stuart M. (1985) Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms, *Proceedings of the 23rd ACL Conference*, 145-152.

Shieber, Stuart M. (1986) *An Introduction to Unification-Based Approaches to Grammar* (= *CSLI Lecture Notes* 4). Stanford: CSLI.

Shieber, Stuart M. (1992) *Constraint-Based Grammar Formalisms*. Cambridge, Mass. & London: MIT Press.

Shieber, Stuart M. / Gertjan van Noord / Fernando C. N. Pereira / Robert C. Moore (1990) Semantic-Head-Driven Generation. *Computational Linguistics* 16: 30-42.