Prediction in Chart Parsing Algorithms for Categorial Unification Grammar

Gosse Bouma
Computational Linguistics Department
University of Groningen, P.O. box 716
NL-9700 AS Groningen, The Netherlands
e-mail:gosse@let.rug.nl

Abstract

Natural language systems based on Categorial Unification Grammar (CUG) have mainly employed bottom-up parsing algorithms for processing. Conventional prediction techniques to improve the efficiency of the parsing process, appear to fall short when parsing CUG. Nevertheless, prediction seems necessary when parsing grammars with highly ambiguous lexicons or with non-canonical categorial rules. In this paper we present a lexicalist prediction technique for CUG and show that this may lead to considerable gains in efficiency for both bottom-up and top-down parsing.

1 Preliminaries

CATEGORIAL UNIFICATION GRAMMAR. Unification-based versions of Categorial Grammar, known as CUG or UCG, have attracted considerable attention recently (see, for instance, Uszkoreit, 1986, Karttunen, 1986, Bouma, 1988, Bouma et al., 1988, and Calder et al., 1988). The categories of Categorial Grammar (CG) can be encoded easily as feature-structures, in which the attribute $< cat >$ dominates either an atomic value (in case of an atomic category) or a structure with attributes $< val >$, $< dir >$ and $< arg >$ (in case of a complex category). Morphosyntactic information can be added by introducing additional labels. An example of such a category represented as attribute-value matrix is presented below.

$$NP[+nom]/N[+nom,+sg] =$$

$$\left[cat : \left[\begin{array}{l} val : \left[\begin{array}{l} cat : np \\ case : nom \end{array} \right] \\ dir : right \\ arg : \left[\begin{array}{l} cat : n \\ case : nom \\ num : sg \end{array} \right] \end{array} \right] \right]$$

The combinatory rules of classical CG, $A \rightarrow A/B \;\; B$ (rightward application) and $A \rightarrow B \;\; B\backslash A$ (leftward application), can be encoded as highly schematic rewrite rules associated with an attribute-value graph:

Rightward Application Rule :

$$X_0 \rightarrow X_1 \;\; X_2$$
$$\left[\begin{array}{l} X_0 :< 1 > \\ X_1 : \left[cat : \left[\begin{array}{l} val :< 1 > \\ dir : right \\ arg :< 2 > \end{array} \right] \right] \\ X_2 :< 2 > \end{array} \right]$$

Leftward Application Rule :

$$X_0 \rightarrow X_1 \;\; X_2$$
$$\left[\begin{array}{l} X_0 :< 1 > \\ X_1 :< 2 > \\ X_2 : \left[cat : \left[\begin{array}{l} val :< 1 > \\ dir : left \\ arg :< 2 > \end{array} \right] \right] \end{array} \right]$$

CUG is a lexicalist theory: language specific information about word order, subcategorization, agreement, case-assignment, etc., is stored primarily in the lexicon. Whereas in classical CG functor-argument structure is the only means available for describing linguistic phenomena, in CUG additional features may be used to account for phenomena such as agreement and case-marking (see Bouma 1988). Also, whereas in classical CG all rules are in principle universal (i.e. not language-specific), in CUG there is a tendency to supplement generic categorial rules with language or construction specific rules For instance, a rule

$$NP \rightarrow N[+plu]$$

may be added to account for the occurence of bare plural NPs, and specific rules may be added to account for unbounded dependency constructions (Bouma

1987). Finally, instead of fully instantiated category-structures, one may choose to work with polymorphic categories (Karttunen 1989, Zeevat et al. 1987). Consequently, CUG not only shows resemblances with traditional categorial grammar, but also with Head-driven Phrase Structure Grammar (Pollard & Sag, 1987), another lexicalist and unification-based framework.

CHART PARSING OF UNIFICATION GRAMMAR (UG). Parsing methods for context-free grammar can be extended to unification-based grammar formalisms (see Shieber, 1985 or Haas, 1989), and therefore they can in principle be used to parse CUG. A chart-parser scans a sentence from left to right, while entering items, representing (partial) derivations, in a chart. Assume that items are represented as Prolog terms of the form $item(Begin, End, LHS, Parsed, ToParse)$, where LHS is a feature-structure and $Parsed$ and $ToParse$ contain lists of feature-structures. An $item(0, 1, [S], [NP], [V, NP])$ represents a partial derivation ranging from position 0 to 1 of a constituent with feature-structure S, of which a daughter NP has been found and of which daughters V and NP are still to be parsed. A word with lexical entry $Word : Cat$ at position $Begin$, leads to addition of an item $item(Begin, Begin + 1, Cat, [Word], [\ ])$. Next, completion and prediction steps are called until no further items can be added to the chart.

Completion step: [1] For each *item(B, E, LHS, Parsed, [Next|ToParse])* and *item(E, End, Next, Parsed, [])*, add an *item(B, End, LHS, Parsed+Next, ToParse)*.

Bottom-up Prediction step: For each *item(B, E, Next, Parsed, [])*, and each rule *(LHS* $\rightarrow$ *[Next | RHS])*, add *item(B, E, LHS, [Next], RHS)*.

The prediction step causes the algorithm to work bottom-up.

2 The Problem

In a bottom-up chart parser, applicable rules are predicted bottom-up, and thus, lexical information is used to constrain the addition of active items (i.e. items representing partial derivations). At first sight, this method appears to be ideal for CUG, as in CUG the lexical items contain syntactic information which is language and grammar specific, whereas the rules are generic in nature. Note, however, that although bottom-up parsing is certainly attractive for CUG, there are also a number of potential inefficiencies:

- In many cases useless items will be predicted. Consider, for instance, a grammar with a lexicon containing only the categories NP/N, N, and $NP\backslash S$, and with application as the only combinatory rules. When encountering a determiner, prediction of an $item(i, i, X, [np/n], [(np/n)\backslash X])$ is superfluous, since there is simply no way that the grammar could ever produce a category $(np/n)\backslash X$ [2].

- If the lexicon is highly ambiguous, many useless (partial) derivations may take place. Consider, for instance, the syntax of NPs in German, where determiners and adjectives are ambiguous with respect to case, declension pattern, gender and number (see Zwicky, 1986, for an analysis in terms of GPSG). The sentence *die junge Frau schläft* has only one derivation, but a bottom-up parser has to consider 11 possible analyses for the word *junge*, 6 for the phrase *junge Frau*, 4 for *die* and 2 for *die junge Frau*. This example shows that even in a pure categorial system, there may be situations where top-down prediction has its merits.

- If the grammar contains language or construction specific rules, bottom-up prediction may be less efficient. Relevant examples are the rule for forming bare plurals mentioned in the previous section and rules which implement a categorial version of *gap-threading* (see Pereira & Shieber, 1986 : 114 ff). The rule shemata below allow for the derivation of sentences with a preposed element and for the extraction of arguments:

Gap-elimination:	$S \rightarrow X \quad S[gap : X]$
Gap-introduction:	$X[gap : Y] \rightarrow X/Y$
	$X[gap : Y] \rightarrow Y\backslash X$

  Gap-introduction will be used every time a functor category is encountered. Again, some form of top-down prediction could improve this situation.

In the following sections, we will consider top-down parsing, as an alternative for the bottom-up approach, and we will consider the possibility of improving the predictive capabilities of a bottom-up parser.

[1] In these and following definitions, we assume, unless otherwise indicated, that feature-structures denoted by identical prolog variables are unified by means of feature-unification.

[2] The example may suggest that prediction should be eliminated all together. This option is feasible only if the rule set is restricted to application.

3 Top-down Parsing

Top-down chart parsing differs from the algorithm described above only in the prediction-step, which predicts applicable rules top-down. Contrary to bottom-up parsing, however, the adaptation of a top-down algorithm for UG requires some special care. For UGs which lack a so-called *context-free back-bone*, such as CUG, the top-down prediction step can only be guaranteed to terminate if we make use of *restriction*, as defined in Shieber (1985).

Top-down prediction with a restrictor R (where R is a (finite) set of paths through a feature-structure) amounts to the following:

Restriction The restriction of a feature-structure F relative to a restrictor R is the most specific feature-structure $F' \sqsubseteq F$, such that every path in F' has either an atomic value or is an element of R.

Predictor Step For each *item(_ , End, LHS, Parsed, [Next | ToParse])* such that R_{Next} is the restriction of $Next$ relative to R, and each rule $R_{Next} \rightarrow RHS$, add *item(i,i,* R_{Next}*, [], RHS)*.

Restriction can be used to develop a top-down chart parser for CUG in which the (top-down) prediction step terminates. The result is unsatisfactory, however, for the following two reasons. First, as a consequence of the generic and language independent nature of categorial rules, the role of top-down prediction as a constraint on possible derivation steps is lost completely. Second, many useless items will be predicted due to the fact that the LHS of both rightward and leftward application always match with R_{Next} in the prediction step (note that a bottom-up parser has a similar inefficiency for leftward application only). Therefore, the overhead which is introduced by top-down prediction does not pay-off. We conclude that, eventhough the introduction of restriction make it possible to parse CUG top-down, in practice, such a method has no advantages over a bottom-up approach.

4 Lexicalist Prediction

Instead of customizing existing top-down parsing algorithms for CUG, we can also try to take the opposite track. That is, we will try to represent a CUG in such a way that non-trivial forms of top-down prediction are possible.

Top-down prediction, as described in the previous section, relies wholly on the syntactic information encoded in the syntactic rules. For CUG, this is an akward situation, as most syntactic information which could be relevant for top-down prediction is located in the lexicon. In order to make this information accessible to the parser, we precompile the grammatical rules into a set of ***instantiated rules***. The instantiated rules are more restrictive than the generic categorial rules, as they take lexical information into account.

The following algorithm computes a set of instantiated syntactic rules, given a set of generic rules and a lexicon.

Compilation For every category C, where C is either a lexical category or the LHS of an instantiated rule, and every (generic) rule GR, if C is unifiable with the head-daughter of GR, add GR' (the result of the unification) to the set of instantiated rules. [3]

We assume that there is some way of distinguishing head-daughters from non-head daughters (for instance, by means of a feature). The head daughter should be the daughter which has the most influence on the instantiation of the rule. For the application rules, for instance, the functor is the most natural choice, as the functor both determines the instantiation of the resultant category and of the argument category.

The compilation step is correct and complete for arbitrary UGs, that is, a string is derivable using the instantiated rules if and only if it is derivable using the generic rules. Note, however, that the compilation procedure does not necessarily terminate. Consider for instance a categorial grammar with category raising $(X/(Y\backslash X) \rightarrow Y)$. In such a grammar, arbitrarily complex instantiations of this rule can be compiled. To avoid the creation of an infinite set of rules, we may again employ restriction:

Compilation with restriction Let R be a restrictor. For every category C, where C is either a lexical category or the LHS of an instantiated rule, and every (generic) rule GR, if the restriction of C relative to R is unifiable with the head-daughter of GR, add GR' (the result of the unification) to the set of instantiated rules.

The compilation step is guaranteed to terminate as long as R is finite (cf. Shieber, 1985). The compilation procedure is not specific to a certain grammar formalism or rule set, and thus can be used to compile arbitrary UGs. Such a compilation step will give rise to a substantially more instantiated rule set in all cases

[3]Note that for classical CG, an algorithm of this kind can be used to compute the phrase-structure equivalent of the input grammar.

where schematic grammar rules are used in combination with highly structured lexical items.

For the compiled grammar, a standard top-down algorithm (such as the one in section 3) can be used. Prediction for CUG is now significant, as only rules which have a functor category that is actually derivable by the grammar will be predicted. So, starting from a category S, we will not predict leftmost categories such as S/NP, $(S/NP)/NP$, if no such categories can be derived from the lexical categories. Also, a leftmost argument category A will only be predicted if the grammar contains a matching functor category $A\backslash S$. Finally, since we are working with the instantiated rules, morphosyntactic information can effectively be predicted top-down.

Restriction is not only useful to guarantee termination of the compilation procedure. The precompilation procedure can in principle lead to an instantiated grammar that is considerably larger than the input grammar. For instance, given a grammar which distinguishes between plural and singular and between first, second and third person NPs, six versions of the rule $S \rightarrow NP \quad NP\backslash S$ might be derivable. Such a multiplication is unnecessary, however, as it does not provide any information which is useful for the top-down prediction step. Choosing a restrictor which filters out all distinctions that are irrelevant to top-down prediction, can prevent an explosion of the rule set.

5 Bottom-Up Parsing with Prediction

The compilation procedure described in section 4 was developed to improve the performance of top-down parsing-algorithms for lexicalist grammars of the CUG-variety. In this section, we argue that replacing a generic CUG with its instantiated equivalent also has advantages for bottom-up parsing. There are two reasons to believe that this is so: first, predictions based on leftward application will be less frequent and second, to an instantiated grammar non-trivial forms of top-down prediction can be added.

In section 2 we pointed out that a bottom-up parser will predict many useless instances of leftward application. This is due to the fact that the leftmost daughter of leftward application is completely general and thus, given an *item(B, E, Cat,Parsed,[])*, an *item(B,E, X, [Cat], [Cat\X])* will always be predicted. The compilation procedure presented in the previous section replaces leftward application with instantiated versions of this rule, in which the leftmost argument of the rule is instantiated. Although the instantiated rule set of a grammar is bound to be larger than the original rule set, which is a potential disadvantage, the chart will grow less fast if we use the instantiated grammar. It is therefore worthwhile to investigate the performance of a bottom-up parser which uses a compiled grammar as opposed to a bottom-up parser working with a generic rule set.

There is a second reason for considering instantiated grammars. It is possible in bottom-up parsing to speed up the parsing process by adding top-down prediction. Top-down prediction is implemented with the help of a table containing items of the form $left\_corner(Ancestor, LeftCorner)$, which lists the *left-corner* relation for the grammar at hand. The *left-corner* relation is defined as follows:

Left-corner Category C_1 is a left-corner of an ancestor category A if there is a rule $A \rightarrow C_1....C_n$. The relation is transitive: if A is a left-corner of B and B a left-corner of C, A is a left-corner of C.

Top-down filtering is now achieved by modifying the prediction step as follows :

Bottom-up Prediction with Top-down Filtering: For each *item(B, E, Cat, Parsed, [])*, and each rule *($X_0 \rightarrow$ [Cat | RHS])*, such that there is an *item(_, B, _, _, [Next|ToParse])* with X_0 a left-corner of $Next$, add *item(B, E, X_0, [Cat], RHS)*[4].

For CUG it makes little sense to compute a left-corner relation according to this definition, since any category X is a left-corner of any category Y (according to leftward application), and thus the left-corner relation can never have any predictive power.

For an instantiated grammar, the situation is more promising. For instance, given the fact that only nominative NPs occur as left-corner of S, and that every determiner which is the left-corner of NP, has a case feature which is compatible (unifiable) with that NP, it can be concluded that only nominative determiners can be left-corners of S.

Computing the left-corner relation mechanichally for a UG will not always lead to the most economical representation of the left-corner table. For example, in German the left-corner of an NP with case and number features X will be a determiner with identical features. If we compute this, using a sufficiently

[4]The bottom-up parsing algorithm extended with left-corner prediction is closely related to the *BUP*-parser of Matsumoto et al. (1983). The *BUP*-parser is based on definite clause grammar and thus, may backtrack. Minimal use is made of a chart (in which successful and failed parse attempts are stored). Our algorithm assigns a more important role to the chart and thus avoids backtracking.

instantiated grammar, we get 8 versions (i.e. 4 cases times 2 possible values for number) of this relation. Similar observations can be made for adjectives that are left-corners of N (where things are even worse, as we would like to take declension classes into account as well). This multiplication may lead to a needlessly large left-corner table, which, if used in the prediction step, may in fact lead to sharp decreases in parsing performance (see also Haas, 1989, who encountered similar problems). Note that checking a left-corner table containing feature-structures is in general expensive, as unification, rather than identity-tests, have to be carried out.

To avoid this problem we have found it necessary to construct the left-corner table by hand, using linguistic *meta-knowledge* about what is relevant, given a particular left-corner relation, to top-down prediction to compress the table to an absolute minimum. It turns out to be the case that only in this way the effect of top-down filtering will pay-off against the increased overhead of having to check the left-corner table.

6 Some Results

The performance of the parsing algorithms discussed in the preceding sections (a bottom-up parser for UG (BU), a top-down parser for UG (cf Shieber, 1985) (TD), a top-down parser operating on an instantiated grammar (TD/I), and a bottom-up parser with top-down filtering operating on an instantiated grammar (BU/LC)) were tested on two experimental CUGs, one implementing the morphosyntactic features of German NPs, and one implementing the syntax of WH-questions in Dutch by means of a gap-threading mechanism. Some illustrative results are listed in Tables 1 and 2.

	Sentence1		Sentence2	
	items	secs	items	secs
TD:	93	5.9	160	10.5
TD/I:	45	2.0	68	2.5
BU :	68	2.0	120	3.0
BU/LC:	12	0.6	23	0.9

Table1: German

For German, an ideal restrictor R was $\{< l^* > | l = cat, val, arg,$ or $dir\}$. This restrictor effectively filters out all morphosyntactic information, in as far as it is not repeated in the categorial rules. The resulting precompiled grammar is much smaller than in the case where no restriction was used or where morphosyntactic information was not completely filtered out. A categorial lexicon for German, for instance, containing only determiners, adjectives, nouns, and transitive and intransitive verbs, will give rise to more than 60 instantiated rules if precompiled without restriction, whereas only four rules are computed if R is used (i.e. only two more than in the uncompiled (categorial) grammar). The improvement in efficiency of TD/I over TD is due to the fact that no useless instances of leftward application are predicted and to the fact that no restriction is needed during parsing with an instantiated grammar. Thus, prediction based on already processed material can be maximal. As soon as we have parsed a category $NP/N[+sg, +wk, +dat, +fem]$, for instance, top-down prediction will add only those items that have $N[+sg, +wk, +dat, +fem]$ as LHS.

BU is almost as efficient as TD/I, eventhough it works with a generic grammar, and thus produces (significantly) more chart-items. Once we replace the generic grammar by an instantiated grammar, and add left-corner relationships (BU/LC), the predictive capacities of the parser are maximal, and a sharp decrease in the number of chart items and parse times occurs.

	Sentence1		Sentence2		Sentence3	
	items	secs	items	secs	items	secs
TD:	255	32.2	225	27.9	358	47.2
TD/I:	48	3.2	71	6.0	129	11.9
BU :	78	1.8	74	1.7	131	3.6
BU/LC:	40	1.7	45	2.1	69	3.9

Table1: Gap-threading

For the grammar with gap-threading (table 2), we used a restrictor $R = \{< l^* > | l = cat, val, arg, dir, gap, in$ or $out\}$. The TD parser encounters serious difficulties in this case, whereas TD/I performs significantly better, but still is rather inefficient. There is a distinct difference between BU and BU/LC if we look at the number of chart items, although the difference is less marked than in the case of German. In terms of parse times the two algorithms are almost equivalent.

Comparing our results with those of Shieber (1985) and Haas (1989), we see that in all cases top-down filtering may reduce the size of the chart significantly. Whereas Haas (1989) found that top-down filtering never helps to actually decrease parse times in a bottom-up parser, we have found at least one example (German) where top-down filtering is useful.

7 Conclusions

There is a trend in modern linguistics to replace grammars that are completely language specific by grammars which combine universal rules and principles with language specific parameter settings, lexicons, etc. This trend can be observed in such diverse frameworks as Lexical Functional Grammar, Government-Binding Theory, Head-driven Phrase Structure Grammar and Categorial Grammar. In parsing with such formalisms, especially those formalisms that are unification-based, we find that traditional parsing-techniques, eventhough they may be applicable to UG, are no longer satisfactory. In particular, prediction techniques which may be efficient for phrase structure grammar do not always carry over easily to UG. The present paper shows that if a grammar uses only schematic combinatory principles instead of phrase-structure rules, prediction is only possible if we replace the generic rules by grammar-specific instances of these rules.

8 Literature

Bouma, G. 1987. A Unification-based Analysis of Unbounded Dependencies in Categorial Grammar, in J. Groenendijk, M. Stokhof, & F. Veltman (eds.) *Proceedings of the sixth Amsterdam Colloquium*, University of Amsterdam, Amsterdam, 1-19.

Bouma, G., 1988, Modifiers and Specifiers in Categorial Unification Grammar, *Linguistics*, vol 26, 21-46.

Bouma, G., E. König, & H. Uszkoreit, 1988. A Flexible Graph-Unification Formalism and its Application to Natural Language Processing, *IBM Journal of Research and Development*, 32, 170-184.

Calder, J., E. Klein, & H. Zeevat 1988. Unification Categorial Grammar: a concise, extendable grammar for natural language processing. *Proceedings of Coling 1988, Hungarian Academy of Sciences*, Budapest, 83-86.

Haas, A. 1989. A Parsing Algorithm for Unification Grammar. *Computational Linguistics* 15-4, 219-232.

Karttunen, L. 1989. Radical Lexicalism. In M. Baltin & A. Kroch (eds.), *Alternative Conceptions of Phrase Structure*, Chicago University Press, Chicago, 43-66.

Matsumoto, Y., H. Tanaka, H. Hirakawa, H. Miyoshi, & H. Yasukawa, 1983, BUP : A Bottom-Up Parser embedded in Prolog. *New Generation Computing*, vol 1, 145-158.

Pereira, F., & S. Shieber (1986). *Prolog and Natural Language Analysis*. CSLI Lecture Notes 10, University of Chicago Press, Chicago.

Pollard, C. & I. Sag, 1987, *Information-Based Syntax and Semantics, vol 1 : Fundamentals*, CSLI Lecture Notes 13, University of Chicago Press, Chicago.

Shieber, S. 1985. Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Algorithms. *Proceedings of the 22nd Annual Meeting of the Association for Computational Linguistics*, University of Chicago, Chicago, 145-152.

Uszkoreit, H. 1986. Categorial Unification Grammars. *Proceedings of COLING 1986*. Institut für angewandte Kommunikations- und Sprachforschung, Bonn, 187-194.

Zeevat, H., E. Klein, & J. Calder, 1987. An Introduction to Unification Categorial Grammar. In N. Haddock, E. Klein, & G. Morill (eds.), *Categorial Grammar, Unification grammar, and Parsing*, Edinburgh Working Papers in Cognitive Science, Vol. 1.

Zwicky, A. 1986. German Adjective Agreement in GPSG. *Linguistics*, vol 24, 957-990.