Head-driven Parsing for Lexicalist Grammars: Experimental Results

Gosse Bouma & Gertjan van Noord
vakgroep Alfa-informatica, University of Groningen
Postbus 716
NL 9700 AS Groningen

Abstract

We present evidence that head-driven parsing strategies lead to efficiency gains over standard parsing strategies, for lexicalist, concatenative and unification-based grammars. A head-driven parser applies a rule only after a phrase matching the head has been derived. By instantiating the head of the rule important information is obtained about the left-hand-side and the other elements of the right-hand-side. We have used two different head-driven parsers and a number of standard parsers to parse with lexicalist grammars for English and for Dutch. The results indicate that for important classes of lexicalist grammars it is fruitful to apply parsing strategies which are sensitive to the linguistic notion 'head'.

1 Introduction

Lexicalist grammar formalisms, such as Head-driven Phrase Structure Grammar (HPSG) and Categorial Unification Grammar (CUG) have two characteristic properties. Lexical elements and phrases are associated with categories that have considerable internal structure. Second, instead of construction specific rules, a small set of generic rule schemata is used. Consequently, the set of constituent structures defined by a grammar cannot be 'read off' the rule set directly, but is defined by the interaction of the rule schemata and the lexical categories.

Applying standard parsing algorithms to such grammars is unsatisfactory for a number of reasons. Earley parsing is intractable in general, as the rule set is simply too general. For some grammars, naive top-down prediction may even fail to terminate. [Shieber, 1985] therefore proposes a modified version of the Earley-parser, using *restricted* top-down prediction. While this modification leads to termination of the prediction step, in practice it easily leads to a trivial top-down prediction step, thus leading to inferior performance.

Bottom-up parsing is far more attractive for lexicalist formalisms, as it is driven by the syntactic information associated with lexical elements. Certain inadequacies remain, however. Most importantly, the selection of rules to be considered for application may not be very efficient. Consider, for instance, the following DCG rule:

$$s([\,]) \rightarrow Arg, vp([Arg]). \tag{1}$$

A parser in which application of a rule is driven by the left-most daughter, as it is for instance in a standard bottom-up active chart parser, will consider the application of rule (1) each time an arbitrary constituent *Arg* is derived. For a bottom-up active chart parser, for instance, this may lead to the introduction of large amounts of active items. Most of these items will be useless. For instance, if a determiner is derived, there is no need to invoke the rule in (1), as there are simply no VP's selecting a determiner as subject.

Parsers in which the application of a rule is driven by the rightmost daughter, such as shift-reduce and inactive bottom-up chart parsers, encounter a similar problem for rules such as (2).

$$vp(Args) \rightarrow vp([Arg|Args]), Arg. \tag{2}$$

Each time an arbitrary constituent *Arg* is derived, the parser will consider applying rule (2), and a search for a matching VP-constituent will be carried out. Again, in many cases (if *Arg* is instantiated as a determiner or preposition, for instance) this search is doomed to fail, as a VP subcategorizing for a category *Arg* may simply not be derivable by the grammar. The problem may seem less acute than that posed by uninstantiated left-most daughters for an active chart parser, as only a search of the chart is carried out and no additional items are added to it. Note, however, that the amount of search required may grow exponentially, if more than one uninstantiated daughter is present (3) or if the number of daughters is not specified by the rule (4), as appears to be the case for some of the rule-schemata used in HPSG:

$$vp(Args) \rightarrow vp([A_1, A_2|Args]), A_1, A_2. \quad (3)$$

$$vp[A_0]) \rightarrow vp([A_0, \ldots, A_n]), A_1, \ldots, A_n. \quad (4)$$

Several authors have suggested parsing algorithms which appear to be more suitable for lexicalist grammars. [Kay, 1989] discusses the concept of *head-driven* parsing. The key idea underlying this concept is that the linguistic notion *head* can be used to obtain parsing algorithms which are better suited for typical natural language grammars. Most linguistic formalisms assume that among the daughters introduced by a rule or rule-schema there is one daughter which can be identified as the *head* of that rule. There are several criteria for deciding which daughter is the head. Two of these criteria seem relevant for parsing. First of all, the head of a rule determines to a large extent what other daughters may or must be present, as the head *subcategorizes* for the other daughters. Second, the syntactic category and morphological properties of the mother node are, in the default case, identical to the category and morphological properties of the head daughter. These two properties suggest that it might be possible to design a parsing strategy in which one first identifies a potential head of a rule, before starting to parse the non-head daughters. By starting with the head, important information about the remaining daughters is obtained. Furthermore, since the head is to a large extent identical to the mother category, effective top-down identification of a potential head should be possible. A head-driven parsing strategy is particularly interesting for lexicalist grammars, as these grammars normally suffer most from the problem that rules or rule-schemata hardly constrain the search-space of the parser.

In [Kay, 1989] two different head-driven parsers are presented. First, a 'head-driven' shift-reduce parser is presented which differs from a standard shift-reduce parser in that it considers the application of a rule (i.e. a reduce step) only if a category matching the head of the rule has been found. Furthermore, it may shift elements onto the parse-stack which are in a sense similar to the active items (or 'dotted rules') of active chart parsers. By using the head of rule to determine whether a rule is applicable, the head-driven shift-reduce parser avoids the disadvantages of parsers in which either the leftmost or rightmost daughter is used to drive the selection of rules.

Kay also presents a 'head-corner' parser. The striking property of this parser is that it does not parse a phrase from left to right, but instead operates 'bidirectionally'. It starts by locating a potential head of the phrase and then proceeds by parsing the daughters to the left and the right of the head. Again, this strategy avoids the disadvantages of parsers in which rule selection is uniformly driven by either the leftmost or rightmost daughter. Furthermore, by selecting potential heads on the basis of a 'head-corner table' (comparable to the left-corner table of a left-corner parser) it may use top-down filtering to minimize the search-space. Head-corner parsing has also been considered elsewhere. In [Satta and Stock, 1989; Sikkel and op den Akker, 1992] chart-based head-corner parsing for context-free grammar is considered. It is shown that, in spite of the fact that bidirectional parsing seemingly leads to more overhead than left-to-right parsing, the worst-case complexity of a head-corner parser does not exceed that of an Earley parser. [van Noord, 1991; van Noord, 1993] argues that head-corner parsing is especially useful for parsing with non-concatenative grammar formalisms. In [Lavelli and Satta, 1991] a head-driven parsing strategy for Lexicalized Tree Adjoining Grammars is presented.

Although it has been suggested that head-driven parsing has benefits for lexicalist grammars, this has not been established in practice. The potential efficiency gains of a head-driven parser are often outbalanced by the cost of additional overhead. This is particularly true for the (bidirectional) head-corner parser. The results of the experiment we describe in section 3 establish that efficient head-driven parsing is possible. That is, we show that for a radical lexicalist grammar (based on CUG) a bottom-up head-driven chart parser (a chart-based breadth-first implementation of Kay's head-driven shift-reduce parser) is more efficient than standard pure bottom-up chart parsers. Also, we show that for a lexicalist (definite clause) grammar in which the rules still contain a substantial amount of information, (bidirectional) head-corner parsing, in which a bottom-up parsing strategy is guided by top-down prediction, is more efficient than pure bottom-up parsing as well as left-corner parsing.

Before discussing the experiment, however, we first discuss the two head-driven parsers used in the experiment, and how they relate to standard parsing algorithms.

2 Two Head-driven Parsers

In this section we present two head-driven parsing algorithms. Prolog code for simplifications of the algorithms is included in the appendix. For each grammar rule $LHS \rightarrow D_1, \ldots, D_h, \ldots, D_n$, it is assumed

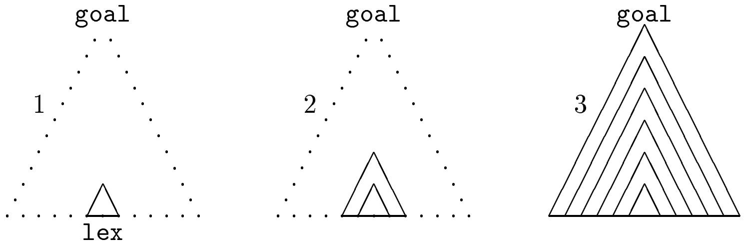


Figure 1: The head-corner parser.

that there is one daughter D_h which has been identified (by the grammar writer) as the head of that rule.

2.1 Head-driven Chart Parsing

The head-driven chart parser scans a sentence from left to right, storing items representing (partial) derivations in a chart. Items are of the form $item(Cat, ToParse, BeginPos, EndPos)$. If $ToParse$ is empty, the item is *inactive*, otherwise it is *active*. The parser is a bottom-up active chart parser without prediction, in which the addition of an active item based on a rule R is considered whenever an inactive item H is entered into the chart which matches the head of R. More precisely, if $item(Cat, [\ ], B, E)$ is derived, and there is a rule $LHS \rightarrow D_1, \ldots, D_{h-1}, Cat, D_{h+1}, \ldots, D_n$ and there are inactive items matching $D_1 \ldots D_{h-1}$, ranging from B_0 to B, an $item(LHS, D_{h+1} \ldots D_n, B_0, E)$ is added to the chart.

If the leftmost daughter of each grammar rule is the head of the rule, then the head-driven chart parser reduces to an ordinary bottom-up active chart parser. If the rightmost daughter of each rule is the head, then the head-driven chart parser reduces to an inactive bottom-up chart parser (i.e. a breadth-first implementation of a shift-reduce parser).

The head-driven strategy has a potential advantage over active bottom-up chart parsers, as it will assert substantially less active items for grammars that contain rules with an underspecified leftmost daughter (as in rule 1). In particular it avoids entering active items into the chart for which it is clear that the missing daughters cannot be derived.

The head-driven parser also has a potential advantage over inactive bottom-up chart parsers for grammars that contain rules with an underspecified rightmost daughter. An inactive chart parser must search in the chart for items matching the remaining daughter of such a rule each time an arbitrary category is derived. The head-driven parser on the other hand only needs to search for matching active items. The difference may lead to important efficiency improvements, especially if searching the chart is expensive. For example this is the case if the unification operation is expensive.

2.2 Head-corner Parsing

Head-corner parsing is a more radical approach to head-driven parsing in that it gives up the idea that parsing should proceed form left to right. Rather, the order of processing in a head-corner parser is bidirectional, starting from a head outward ('island'-driven). A head-corner parser can be thought of as a generalization of the left-corner parser [Rosenkrantz and Lewis-II, 1970]. As in the left-corner parser, the flow of information in a head-corner parser is both bottom-up *and* top-down.

The basic idea of the head-corner parser is illustrated in figure 1. The parser selects the head of the string (1), and proves that this element is the head-corner of the goal. To this end, a rule is selected of which this lexical entry is the head daughter. Then the other daughters of the rule are parsed recursively in a bidirectional fashion: the daughters left of the head are parsed from right to left (starting from the head), and the daughters right of the head are parsed from left to right (starting from the head). The result is a slightly larger head-corner (2). This process repeats itself until a head-corner is constructed which dominates the whole string (3).

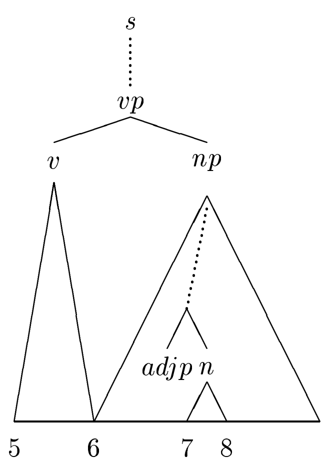
Note that a rule is triggered only with a fully instantiated head-daughter. The 'generate-and-test' behavior observed for example 1 is avoided in a head-corner parser, because the rule is applied only if the VP is found, and hence Arg is instantiated. For example if $Arg = np(sg3, [\,], Subj)$, the parser continues to search for a singular NP, and need not consider other categories.

The head-relation holds between two categories h and m with respect to a grammar G iff G contains a rule with left hand side m and head daughter h. The relation 'head-corner' is the reflexive and transitive closure of the head relation. As in the left-corner parser, a 'linking' table is maintained which represents important aspects of this head-corner relation. For some grammars this table simply represents the fact that the HEAD features of a category and its head-corner are shared.

Note that unlike the left-corner parser, the head-corner parser may need to consider alternative words as a possible head-corner of a phrase, e.g. when parsing a sentence which contains several verbs. This problem is reduced because of the following three observations.

The Quicksort Effect. A simplified version of the head-corner parser is provided in the appendix. The main difference with a simple version of the left-corner parser is — apart from the head-driven selection of rules — the use of two pairs of indices, to implement the bidirectional way in which the parser proceeds through the string.

Observe that each parse-goal in the left-corner parser is provided with a category and a left-most position. In the head-corner parser a parse-goal is provided either with a begin or end position (depending on whether we parse from the head to the left or to the right) but also with the extreme positions between which the category should be found. In general the parse predicate is thus provided with a category and two pair of indices. The first pair indicates the begin and end position of the category, the second pair indicates the extreme positions between which the first pair should lay. The following figure illustrates this point with an example:



Suppose we found for a goal category *s* a possible head-corner *v* from position 5 to 6. In order to construct a complete tree *s* for this head-corner, a rule is selected which dictates that a category *np* should be parsed to the right, starting from position 6. To parse *np*, we predict the head-corner *n* between 7 and 8. Suppose furthermore that in order to connect *n* to *np* a rule is selected which requires a category *adjp* to the left of *n*. It will be clear that this category should end in position 7, but can never start before position 6. Hence the only candidate head-corner of this phrase is to be found between 6 and 7. This example illustrates that the use of two pairs of string positions reduces the number of possible head-corners for a given goal.

String positions in head-corner table. Secondly, the head-corner table includes information about begin and end positions, following an idea in [Sikkel and op den Akker, 1992]. For example, if the goal is to parse a phrase with category *sbar* from position 7, and within positions 7 and 12, then for some grammars it can be concluded that the only possible head-corner for this goal should be a complementizer starting at position 7. Such information is compiled into the table as well. Hence the number of possible head-corners is reduced.

Well-formed substring tables. Thirdly, the problem of multiple possible heads is reduced because a well-formed substring table is maintained. This is implemented by a memo-ization technique. This reduces the problem because even if the wrong head-corner is predicted for a given goal, it may turn out to be the case that the computations based on this wrong prediction may be useful later (each lexical category usually is the head of *some* projection).

The well-formed substring table is implemented using an interesting generalization of the subsumption relation. A goal need not be investigated anymore if a more general goal has already been completed. It is easy to see that a certain goal with extreme positions 3 to 6 is more general than an otherwise identical goal with extreme positions 4 and 6.

Head-driven vs. functor-driven parsing. For categorial unification grammars in which we choose the functor as the head of a rule, the head-corner table is not going to be discriminating, because the grammar rules in such a grammar may simply be (in DCG notation, given appropriate operator definitions): [1]

$$\begin{array}{l} Val \rightarrow Val/Arg, Arg \\ Val \rightarrow Arg, Arg\backslash Val \end{array} \qquad (5)$$

As no information about word-class or morphology is stated in the rules, such information will not be found in the head-corner table.

A possibly useful approach here is to compile some lexical information into the rule set, along the lines proposed in [Bouma, 1991]. In that paper it is proposed to compile lexical information into the rule-set, and parse with this 'enriched' rule-set. What seems to be most useful here, is to use this enriched grammar only for the compilation of the head-corner table. The parser then uses the general rule schemata themselves.

However, given the usual analysis of modifiers as functors, even this approach may fail to yield an interesting head-corner table. Note that some analyses in categorial grammar prescribe that even in such cases certain morphological features are shared between the functor and its resulting value [Bouma, 1993].

2.3 Comparison

The important differences between both head-driven parsing algorithms can be summarized as follows (see also table 1). Firstly the head-driven chart parser proceeds from left-to-right as usual, whereas the head-corner parser proceeds bidirectionally. Secondly, the head-driven chart parser is an active chart parser (i.e. it also stores partial analyses of phrases);

[1] the second author prefers to write the second rule as

$$Val \rightarrow Arg, Val\backslash Arg$$

the head-corner parser uses memo-ization of the parse predicate and the head-corner predicate (i.e. it only stores complete analyses of phrases, and partial analyses of head-corners).

We also implemented an active head-corner chart parser along the lines of [Sikkel and op den Akker, 1992], but preliminary experiments indicate that (our implementation of) this parser is not useful for the grammars used in the experiments to be discussed in the next section. Note that it is not possible to incorporate top-down filtering in the head-driven chart parser in a simple way, because the necessary active items may not be available yet.

Thirdly, although in both algorithms the way rules are applied is bottom-up in an important sense, there is an important flow of information in top-down direction in the head-corner parser. For grammars in which the head-corner table is discriminating, this should have important effects in practice. This expectation is confirmed in the experiments discussed in the next section.

3 The experiment

This section describes experimental results for the parsing algorithms discussed above, in comparison with some obvious alternative strategies. The experiment consists of two parts.

The first part of the experiment compares parsing strategies which proceed in a bottom-up fashion without the use of any top-down prediction. For CUG such parsers are suitable as no top-down information can be compiled from the rule schemata in a simple way.[2] It turns out that the head-driven bottom-up chart parser performs better than both an inactive and an active bottom-up chart parser, for a particular CUG for English. If the cost of unification is relatively high, the use of the head-driven chart parser pays off. If unification is cheap, then the inactive chart parser may still be the most efficient choice.

The second part of the experiment concentrates on the comparison between the head-corner parser and the left-corner parser. Both of these parsers proceed in a bottom-up fashion, but use important top-down prediction. Such parsers are interesting for grammars in which interesting top-down information can be extracted from the rule schemata. It can be concluded from the experiment that for a specific lexicalist Definite Clause Grammar for Dutch the head-corner parser performs much better than the left-corner parser.

These results indicate that at least for some grammars it is fruitful to apply parsing strategies which are sensitive to the linguistic notion 'head'.

A CUG for English. The first grammar is a CUG for English which includes rules for leftward and rightward application and four construction specific rules to implement gap-threading. The grammar covers the basic sentence types (declaratives, WH and yes-no questions, and relative clauses) and a wide range of verbal and adjectival subcategorization types. PPs may modify nouns as well as VPs, leading to so-called PP-attachment ambiguities. The syntax of unbounded dependency constructions is treated rather extensively, including accounts of constraints on extraction, pied-piping, and the possibility of nested dependencies (as in *which violin is this sonata easy to play on*). The grammar is defined in terms of feature-structures, which may be combined using feature-unification. Furthermore, the treatment of nested dependencies uses lists of *gaps*. The interaction of these lists with certain lexical entries (such as *easy*) as well as the interaction of these lists with the checking of island-constraints requires that attempts at cyclic unifications must be detected and must fail. Therefore, the feature-unification procedure includes an occurs check.

If the standard techniques for compiling a left-corner resp. a head-corner table are applied for this grammar, then, at best, the 'trivial' link would result, because the rule schemata do not specify any interesting information about morphological features etc.

A lexicalist DCG for Dutch. This grammar is a definite clause grammar for Dutch, in which subcategorization requirements are implemented using subcat-lists. The grammar handles topicalization using gap-threading. Verb-second is accounted for by a feature-based simulation of head-movement. The grammar analyses cross-serial dependencies by concatenating subcategorization lists (implemented as difference lists). As opposed to the CUG grammar, the second grammar uses actual 'empty elements' to introduce the traces corresponding to the topicalized phrases and verbs occurring in second position. Another difference with the first grammar is that first-order terms are used, rather than feature structures.

The compilation of the left-corner resp. the head-corner table was done using the same restrictor. The left-corner table contained 94 entries, and the head-corner table contained 25 entries.

The parsers. The parsers used in the experiment have a number of important properties in common (see table 1). First of all, they all use a chart to represent (partially or fully developed) analyses of substrings. Second, as categories are feature-structures or terms, rather than atomic symbols, special requirements are needed to ensure that the chart is always 'minimal'. That is, items are only added to the chart if no subsuming item exists, and, if an item is added to the chart, all more specific items are deleted from the chart. Finally, information about the derivational history of phrases is added to the chart in such a way that parse-trees can be recovered.

[2] but see the discussion on head-driven vs. functor-driven parsing in the previous section.

	inact	hdc	act	lc	hc
well-formed substrings	+	+	+	+	+
packing	+	+	+	+	+
subsumption-checking	+	+	+	+	+
active items	-	+	+	+	-
left-to-right processing	+	+	+	+	-
top-down filtering	-	-	-	+	+
head-driven processing	-	+	-	-	+

Table 1: The parsers used in the experiment

n	parses	items			recognition			parsing		
		hdc	inact	act	hdc	inact	act	hdc	inact	act
		#	%	%	sec	%	%	sec	%	%
3	1	25	67	170	.8	63	191	1.1	67	168
6	1	43	73	180	.9	87	199	1.4	90	164
9	2	89	74	179	2.5	91	208	3.6	94	179
12	3	141	75	181	4.0	102	211	6.2	101	175
15	4	193	79	184	5.5	111	214	8.2	109	180
18	6	254	82	184	7.0	124	215	10.8	113	175
21	32	369	84	181	10.9	135	224	30.0	117	147
24	98	452	86	181	13.7	140	225	87.0	106	120
27	55	472	87	185	14.1	142	233	29.7	119	164
30	95	592	87	179	19.9	144	218	172.7	107	120

Table 2: Results for the English grammar

This is done by using 'packed structures' (also called 'parse-forests') to obtain structure sharing in the case of ambiguities; semantic constraints (if present) are only evaluated when the syntactic analysis phase is completed. Our implementation of 'packing' follows that of [Moore and Alshawi, 1992], who implement it for a (unification-based) left-corner parser.

Three different bottom-up chart parsers are implemented. The first one (*hdc*) is the head-driven chart parser presented above, in which the head of the rule is given by the grammar writer. The active chart parser (*act*) is the same as the head-chart parser, but now it is assumed that for each rule the left-most daughter is the head (*active chart*). The inactive chart parser (*inact*) is a version of the head-corner parser where each right-most daughter is assumed to be the head of the rule. Since the parser does not use active items, some (slight) simplifications of the head-driven chart parser were possible.

The left-corner parser is a generalized version of the chart-based left-corner parser of [Rosenkrantz and Lewis-II, 1970]. As we also add items to construct parse-trees using 'packing', the resulting parser should be comparable to the CLE parser [Moore and Alshawi, 1992]. The head-corner parser is the parser discussed in the previous section.[3]

[3] We also implemented a generalized Earley parser. This parser was extremely slow for all sentences of both grammars.

Results for CUG. One hundred arbitrarily chosen sentences (10 of length 3, 10 of length 6, etc.) were parsed, using the three pure bottom-up parsers (*hdc, inact*, and *act*). The columns in table 2 give, for each sentence length (column 1), the average number of readings (column 2), the average number of items produced by *hdc*, and the average percentage of items produced by *inact* and *act*, when compared with *hdc* (columns 3-6), the average time it took *hdc* to parse a sentence without recovering the different analyses and the average percentage of time needed for *inact* and *act* to do that (columns 7-9), and finally the average time it took to parse a sentence and recover all analysis trees for *hdc* and the average percentage of time needed by *inact* and *act* to do that.

The number of chart items illustrate clearly that *hdc* combines features of an inactive chart parser with that of an active chart-parser. Note that, in spite of the fact that English is mostly a head-initial language, *act* produces 80% more items than *hdc*, whereas *inact* almost produces 80% of the items produced by *hdc*. For languages which are predominantly head-final, the difference between *act* and *hdc* will probably be larger, whereas that between *inact* and *hdc* should be smaller.

The recognition times show that an active bottom-up chart parser is two-times slower for this grammar than a head-driven chart parser. The difference between the inactive chart parser and the head-driven

n	parses	recognition					parsing				
		hc	hdc	lc	act	inact	hc	hdc	lc	act	inact
		sec	%	%	%	%	sec	%	%	%	%
3	1	.3	2647	80	2804	390	.5	1699	79	1759	428
6	2	.8	5407	343	5968	1044	1.6	3698	215	4265	1300
9	6	1.2		550		1170	2.9		334		1474
12	5	2.0		428		2333	3.8		285		
15	9	3.1		355		2521	6.7		210		
18	16	5.1		248		2408	10.7		160		
21	20	7.4		195		1918	15.3		127		
24	23	10.2		147			19.8		104		
27	61	13.8		209			34.3		131		
30	87	17.3		145			62.4		102		

Table 3: Results for the Dutch grammar. For parsers which did not succeed within a given period, the entry in the table has not been filled in.

parser is less extreme, and is notably in favor of the head-driven parser only for relatively long and complex (in terms of number of analyses) sentences. Nevertheless, the difference is of enough significance to establish the superiority of a head-driven strategy in this case.

The final three columns show that if recovery of parse trees is taken into account as well, the differences are much less extreme. The reason for this difference is simply that recovery (for which we used an Earley-style top-down algorithm which reconstructs explicit analysis trees on the basis of inactive items) may take up to eight times as long as doing parsing without recovery. Since the amount of time needed for recovery is (approximately) equal for all three parsers, this explains why the relative differences are much smaller in this case.

The head-corner parser was applied to the same grammar and sentence set as well. It behaves much worse (up to 100 times as slow for recognition of 24-words sentences) than the parsers listed in the tables due to the lack of guiding top-down information. The left-corner parser without top-down prediction reduces to the active chart parser.

We also applied the same sentence set to a compiled version of the same CUG. In this compiled version first-order terms were used, rather than feature structures. Furthermore, we used ordinary Prolog unification on such terms rather than the previously mentioned feature unification including occurs check. This implied that we had to forbid multiple extractions in the compiled version of the grammar. Experiments indicate that in such cases the inactive chart parser performs consistently better than both the head-driven chart parser and the active chart parser. This should not come as a surprise given the discussion in section 2.1 where we expected the head-driven chart parser to be useful for grammars with an 'expensive' unification operation.

Results for the DCG. The next table encodes the results for the Dutch grammar (cf. table 3). Again, one hundred sentences were chosen (ten of three words, ten of six words, etc).

The head-corner parser improved with a well-formed substring table and packing beats the bottom-up chart parsers. This is explained by the fact that these parsers proceed strictly bottom-up, whereas the left-corner and head-corner parser employ both top-down and bottom-up information. The top-down information is available through a left-corner resp. head-corner table, which turn out to be quite informative for this grammar.

The head-corner parser performs considerably better than the left-corner parser on average, especially if we only take the recognition phase into account. For longer sentences the differences are somewhat less extreme than for shorter sentences. This difference is due to the fact that the left-corner parser seems somewhat better suited for grossly ambiguous sentences. Furthermore, the number of items used for the representation of parse trees is not the same for the left-corner and head-corner parser. For ambiguous sentences the head-corner parser produces more useless items, in the sense that such items can never be used for the construction of an actual parse tree. As a consequence, it is more expensive to recover the parse trees based on this representation, than it is for the recovery of parse trees based on the smaller representation built by the left-corner parser. A few numbers for three typical (long) sentences are shown in table 4.

This is a somewhat puzzling result. Useless items are asserted only in case the parser is following a dead-end. However, the fact that the number of useless items is larger for the head-corner parser than for the left-corner parser implies that the head-corner parser follows more dead-ends, yet the head-corner parser is much faster during the recognition phase. A possible explanation for this puzzling fact may be the overhead involved in keeping track of the ac-

# parses	hc				lc			
	items #	recognition sec	recovery sec	total sec	items #	recognition sec	recovery sec	total sec
26	768	12	12	24	503	33	8	41
100	1420	20	37	57	831	43	29	72
30	543	9	10	19	430	20	8	28

Table 4: Comparison of the size of the parse forest for the left-corner and head-corner parser for a few (longer) sentences.

tive items in the left-corner parser whereas no active items are asserted for the head-corner parser. Clearly for grammars with rules that contain many daughters (unlike the grammar under consideration) the use of active items may start to pay off.

Note that we also implemented a version of the head-corner parser that asserts less useless items by delaying the assertion of items until a complete head-corner has been found. However, given the fact that this technique leads to a more complex implementation of the memo-ization of the head-corner relation, it turned out that this immediately leads to longer recognition times, and an overall worse behavior.

4 Conclusion

The main conclusion to be drawn from the experiments discussed above is that the influence of the grammar can hardly be underestimated. The parser that works best for one grammar may easily turn out to be the most inefficient one for a different grammar. This observation also holds for the grammars discussed above even though these are both lexicalist grammars.

Head-corner parsing appears to be superior for grammars in which the head-corner table contains discriminating information. A typical DCG grammar for a head-final language such as Dutch is an example of such a grammar. On the other hand, for grammars in which top-down filtering is difficult to implement, strictly bottom-up parsing strategies are more useful, especially if the number of active items can be reduced, either by a lazy strategy which never enters active items in the chart or, even more successful for the CUG grammar for English we considered, a head-driven strategy.

Clearly many other factors may be relevant in finding the best parser for a particular grammar. For example the cost of unification turns out to be an important factor. As indicated above a cheap unification procedure may favor an inactive chart parser, even if in that parser many useless reductions are attempted. However, if the cost of unification is relatively high, the cost of the use of active items to reduce the number of useless reductions, for example by a head-driven strategy, may be worthwhile.

Another result we obtained during the experiments is that the use of a head-corner and left-corner table may also lead to *inefficiency*. It may be the case that on the basis of the left-corner table (resp. head-corner table) very little derivations are actually filtered out. Furthermore, the use in the table may even lead to *more* derivations as now certain subcases are considered which are considered as a single derivation in a parser without prediction. An important problem thus is to come up with the most useful left-corner (resp. head-corner) table for a given grammar.

A final factor in determining the best parser is the actual use we want to make of the parser. For example, are we interested in the times needed to do recognition, or do we need to consider the times used for the recovery of parse trees as well. In some systems these different parse trees are never actually built but the semantic and pragmatic components directly work on the items built by the parser [Moore and Alshawi, 1992]. We conjecture that even in such applications it is probably a good thing to limit the size of the parse forest, but the importance may vary from application to application.

References

[Bouma, 1991] Gosse Bouma. Prediction in chart parsing algorithms for categorial unification grammar. In *Fifth Conference of the European Chapter of the Association for Computational Linguistics*, Berlin, 1991.

[Bouma, 1993] Gosse Bouma. *Nonmonotonicity and Categorial Unification Grammar*. PhD thesis, University of Groningen, 1993.

[Kay, 1989] Martin Kay. Head driven parsing. In *Proceedings of Workshop on Parsing Technologies*, Pittsburg, 1989.

[Lavelli and Satta, 1991] Alberto Lavelli and Giorgio Satta. Bidirectional parsing of lexicalized tree adjoining grammar. In *Fifth Conference of the European Chapter of the Association for Computational Linguistics*, Berlin, 1991.

[Moore and Alshawi, 1992] Robert C. Moore and Hiyan Alshawi. Syntactic and semantic processing. In Hiyan Alshawi, editor, *The Core Language Engine*, pages 129–148. ACL-MIT press, 1992.

[Rosenkrantz and Lewis-II, 1970] D.J. Rosenkrantz and P.M. Lewis-II. Deterministic left corner parsing. In *IEEE Conference of the 11th Annual Symposium on Switching and Automata Theory*, pages 139–152, 1970.

[Satta and Stock, 1989] Giorgio Satta and Oliviero Stock. Head-driven bidirectional parsing. a tabular method. In *Proceedings of the Workshop on Parsing Technologies*, pages 43–51, Pittsburg, 1989.

[Shieber, 1985] Stuart M. Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *23th Annual Meeting of the Association for Computational Linguistics*, Chicago, 1985.

[Sikkel and op den Akker, 1992] Klaas Sikkel and Rieks op den Akker. Head-corner chart parsing. In *Proceedings Computing Science in the Netherlands (CSN '92)*, Utrecht, 1992.

[van Noord, 1991] Gertjan van Noord. Head corner parsing for discontinuous constituency. In *29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, 1991.

[van Noord, 1993] Gertjan van Noord. *Reversibility in Natural Language Processing*. PhD thesis, University of Utrecht, 1993.

A A head-driven chart parser

The main omission consists of the administration concerning the packed items, for the recovery of parse-trees. Also this version assumes that no empty productions occur in the grammar.

Rules are of the form `rule(Head, LHS, LeftDs, RightDs)`, where `LeftDs` is in reversed order. The predicate `lex(Cat,P0,P)` is true if the word connecting the positions `P0` and `P` has category `Cat`.

The chart consists of (dynamically asserted) facts of the form `item(Cat,ToParse,P0,P)`, indicating that if there is a list of categories `ToParse` from position `P` to `Q` then there is category `Cat` from position `P0` to `Q`. The predicate `assertz_check` is used to asserts such items. That predicate asserts its argument only if no more general clause exists; furthermore it deletes all more specific clauses.

```
% scan(+P0,+P) parses from P0 to P,
% P is current position
scan(P,P).
scan(P0,P) :-
   P1 is P0 + 1,
   ( lex(Cat,P0,P1),
     add_item(Cat,[],P0,P1),
     fail
   ; scan(P1,P)
   ).

% add_item(+Cat,+ToParse,+Begin,+End)
% asserts item and computes all its
% consequences, if inactive item
add_item(Cat,[],B,E) :-
   assertz_check(item(Cat,[],B,E)),
   closure(Cat,B,E).
add_item(Cat,[H|T],B,E) :-
   assertz_check(item(Cat,[H|T],B,E)).

% closure(+Cat,+Begin,+End)
% computes all the items on basis
% of item Cat from Begin to End
closure(Cat,P1,P) :-
   item(Lhs,[Cat|ToParse],P0,P1),
   add_item(Lhs,ToParse,P0,P),
   fail.
closure(Cat,P1,P) :-
   rule(Cat,Lhs,Left,Right),
   left(Left,P0,P1),
   add_item(Lhs,Right,P0,P),
   fail.
closure(_,_,_).

% left(+Ds,?Begin,+End) if there are Ds
% from right from Begin to End
left([],B0,B0).
left([D|Ds],B0,E) :-
   item(D,[],B,E),
   left(Ds,B0,B).
```

B A head-corner parser

The main omission of the following version of the head-corner parser is the administration concerning the well-formed substring table, packing and the possibility of rules with an empty right hand side. In the head-corner parser used in the experiment the parse predicate and the head-corner predicate are memoized. Furthermore items for the parse forest are asserted in the head-corner predicate. Finally some special arrangements are made to allow for rules with an empty right hand side, by allowing underspecification of the string position in the comparison predicates.

The relation `hc_table(Cat,P0,P,Goal,Q0,Q)` implements the head-corner table. If `P0=Q0` the phrase is head-initial; if `P=Q` the phrase is head-final. Rules and lexical entries are represented as before.

```
% parse(Cat,P0,P,E0,E) if there is
% Cat from P0 to P, within range E0,E
parse(Goal,P0,P,E0,E) :-
   predict(Goal,P0,P,Lex,Q0,Q,E0,E),
   head_corner(Lex,Q0,Q,Goal,P0,P,E0,E).

% head_corner(Cat,C0,C,Goal,G0,G,E0,E)
% if Cat from C0 to C is a head-corner of
% Goal from G0 to G within E0 to E.
head_corner(Cat,Q0,Q,Cat,Q0,Q,_,_).
head_corner(Small,Q1,Q2,Goal,P0,P,E0,E) :-
   rule(Small,Mid,Left,Right),
   left(Left,Q0,Q1,E0),
   right(Right,Q2,Q,E),
   hc_table(Mid,Q0,Q,Goal,P0,P),
   head_corner(Mid,Q0,Q,Goal,P0,P,E0,E).

% predict(Goal,P0,P,Lex,Q0,Q,E0,E)
% if Lex from Q0 to Q may be head-corner
% of Goal from P0 to P within E0, E.
predict(Goal,P0,P,Lex,Q0,Q,E0,E) :-
   hc_table(Lex,Q0,Q,Goal,P0,P),
   lex(Lex,Q0,Q),
   E0 =< Q0,
   Q =< E.

% left(Ds,P0,P,E0) if (reversed) Ds exist
% from P to P0 with left-extreme E0
left([],P,P,_).
left([H|T],P0,P,E0) :-
   parse(H,P1,P,E0,P),
   left(T,P0,P1,E0).

% right(Ds,P0,P,E) if Ds exist from
% P0 to P with right-extreme E
right([],P,P,_).
right([H|T],P0,P,E):-
   parse(H,P0,P1,P0,E),
   right(T,P1,P,E).
```