Head-Corner Parsing of Unification Grammars:
A Case Study

Margriet Verlinden

Department of Computer Science, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
email: margriet@cs.utwente.nl

ABSTRACT

The head-corner chart parsing algorithm by Sikkel and Op den Akker in [SA92] defines a new way to grammatically analyse sentences of a context-free language. In this algorithm sentences are parsed in another order than as usual from left to right. In informal terms, head-driven parsing is a strategy for syntactic analysis in which we start by looking for the main word. There is a linguistic motivation for the notion head, but in this paper the subject is looked at only from a computing science perspective.

The head-corner parsing algorithm has been extended to unification grammars and an implementation of the extended algorithm has been written. In this paper will be presented: the original algorithm, the extension, some aspects of the implementation and results of comparisons of the head-corner parser with some other parsers.

1 THE ALGORITHM

The algorithm for head-corner chart parsing from [SA92] was the basis for the implementation of a parser. A formal description of this algorithm will be given in this section. In [SA92] more versions of the algorithm and proofs of correctness can be found. A shorter version has been published as [SA93].

1.1 NOTATIONAL CONVENTIONS

We use the following notations. Nonterminals are denoted by $A, B, \ldots \in N$; terminals by $a, b, \ldots \in \Sigma$. We write $V$ for $N \cup \Sigma$, with $X, Y, \ldots$ as typical elements. Strings in $V^*$ are denoted by $\alpha, \beta, \ldots$.

A context-free grammar $G$ is a 4-tuple $(N, \Sigma, P, S)$, with $P$ a set of productions and $S$ the start symbol. A lexicon $L$ for $G$ is a 3-tuple $(\Sigma, E, W)$, with $W$ the words of a natural language, $E$ a set of lexical entries of the form $b \rightarrow \text{word}$, with $b \in \Sigma$ and $\text{word} \in W$. The sentence to be parsed is often denoted by the lexical categories of its words: $a_1 \ldots a_n$. Place markers $i, j, k, \ldots$ are used to indicate positions in the sentence. The symbol $a_i$ is located between positions $i-1$ and $i$. The derivation arrow $\Rightarrow$ is defined by:

$\alpha A \beta \Rightarrow \alpha \gamma \beta$ if there is a production $p \in P$ of the form $A \rightarrow \gamma$.

Furthermore $\Rightarrow^*$ is the transitive and reflexive closure of $\Rightarrow$.

1.2 HEADS

To define the algorithm for head-corner chart parsing we need to define the notions head, head grammar and head-corner first:

- A context-free head grammar is a 5-tuple $(N, \Sigma, P, S, r)$, with $r$ a function that assigns a natural number to each production in $P$. Let $|p|$ denote the length of the right-hand side of $p$. Then $r$ is constrained to $r(p) = 0$ for $|p| = 0$ and $1 \leq r(p) \leq |p|$ for $|p| > 0$.

- The head of a production $p$ is the $r(p)$-th symbol of the right-hand side; the head is empty if and only if the production is empty.

- The head-corner relation $\succ_h$ on $N \times (V \cup \{\varepsilon\})$ is defined by:

$A \succ_h U$ if there is a production $p = A \rightarrow \alpha \in P$ with $U$ the head of $p$.

More use will be made of the transitive and reflexive closure of $\succ_h$, denoted as $\succ_h^*$. 
In a much more practical notation for head grammars, the function r is not defined explicitly, but the head of each production is simply underlined. An example is given below.

\[
S \rightarrow NP VP \\
VP \rightarrow \text{verb} \ NP \\
NP \rightarrow \text{det} \ noun
\]

1.3 ITEMS

The head-corner chart algorithm is related to the Earley chart parser ([Ear70]). Both are based on the recognition of items. Items represent parts of a phrase that have been or have to be parsed. For the head-corner parser we distinguish the following kinds of items:

\[
[l, r, A], \ predict \ items, \ \ [l, r, A; B \rightarrow \alpha \beta \gamma, i, j], \ head\text{-}corner \ items, \ \ [a, j - 1, j], \ terminal \ items.
\]

Recognition of items should be interpreted as follows. A predict item \([l, r, A]\) is recognized if a constituent \(A\) is being looked for, located somewhere between \(l\) and \(r\). A head-corner item \([l, r, A; B \rightarrow \alpha \beta \gamma, i, j]\) is recognized if \([l, r, A]\) is looked for, \(A \rightarrow^* B\), and \(\beta \Rightarrow^* a_{i+1} \ldots a_{j}\) has been established. A terminal item \([a, j - 1, j]\) is recognized if the \(j\)-th word of the sentence belongs to lexical category \(a\). An item is called recognizable for a given sentence if a chart parser (following rules that will be introduced next) will add the item to the chart sometime or other. A sentence is correct if and only if \([0, n, S; S \rightarrow^* 0, n]\) is recognizable. For a more formal treatment and correctness of the head-corner parser see [SA92].

We start the parser with a chart that contains all recognizable terminal items. When the \(j\)-th word belongs to different categories, say \(a\) and \(b\), then both items \([a, j - 1, j]\) and \([b, j - 1, j]\) are present in the initial chart. Furthermore, the item \([0, n, S]\) is needed initially. New items can be derived from already recognized items by applying one of the operators, defined in Section 1.4. The chart parsing algorithm stops when all recognizable items have been added to the chart (and no other items).

1.4 OPERATORS

The definition of the operators is given in Figure 1. The operators predict, scan and complete can work in two directions from the parsed part of an item. In Figure 1 both possibilities are given for each operator. The turnstyle \(\rightarrow\) notation is a convenient shorthand, to be interpreted as:

"if each of the the arguments (i.e. items left of \(\rightarrow\)) has been recognized then the item right of \(\rightarrow\) can be recognized also"

As mentioned in Section 1.3 the operators are used to recognize new items and add them to the chart. The conventional way to preserve that a deterministically defined set of items is added to the chart, is to make use of a second data structure called agenda. Initially, the chart contains only the terminal items; item \([0, n, S]\) is on the agenda. At each step of the algorithm, an item (the current item) is taken from the agenda and added to the chart. For each operator it is checked whether new items can be recognized from the current item and other relevant items present in the chart. These new items are added to the agenda. In Figure 2 this general chart parsing algorithm is written schematically.

The head-corner operators make that the order of parsing is determined by the heads of the productions. The analysis of the sentence 'The cat catches a mouse' for the grammar in Section 1.2 illustrates this. The completed chart for this example is shown in Figure 3. From a completed chart the parse tree of the parsed sentence can be derived. For the example the parse tree is given in Figure 4.

2 EXTENSION TO UNIFICATION GRAMMARS

In unification grammars symbols of the grammar may have features and productions may have feature rules or feature constraints. These constraints determine how the values of the features are to be computed from other values by unification, or what the conditions on feature values are. A parser for unification grammars not only computes parse trees for a sentence but it also computes the feature values of all words and checks the conditions. After the grammar and lexicon are extended with feature constraints also the roles of feature structures in the items and operators have to be defined. All these adaptations of the algorithm will be described in this section.

2.1 GRAMMAR AND LEXICON

In Figures 5 and 6 examples of a grammar and lexicon with feature constraints are given. Linguistically much more interesting features can be
head-corner: there are three distinguished cases for head-corners $b, C, α$.

(i) for $A →^*_b B, \ B → abγ ∈ P, \ l < j ≤ r$:

\[ [l, r, A], [b, j - 1, j] ⊢ [l, r, A; B → αabγ, j - 1, j]; \]

(ii) for $A →^*_γ B, \ B → αCγ ∈ P$:

\[ [l, r, A; C → δγ, i, j] ⊢ [l, r, A; B → αCγ, i, j]; \]

(iii) for $A →^*_ε B, \ B → ε ∈ P, \ l ≤ j ≤ r$:

\[ [l, r, A], ⊢ [l, r, A; B → ε, j, j]; \]

\[ [l, r, A; B → αCγ, i, j] ⊢ [l, i, C]; \]

\[ [l, r, A; B → αCγ, i, j] ⊢ [j, r, C]; \]

\[ [a, j - 1, j], [l, r, A; B → αaβγ] ⊢ [l, r, A; B → αaβγ, j - 1, k]; \]

\[ [l, r, A; B → αaβγ, i, j], [a, j, j + 1] ⊢ [l, r, A; B → αaβγ, i, j + 1]; \]

\[ [l, j, C; C → δγ, i, j], [l, r, A; B → αCγ, j, k] ⊢ [l, r, A; B → αCγ, i, k]; \]

\[ [l, r, A; B → αCγ, i, j], [j, r, C; C → δγ, j, k] ⊢ [l, r, A; B → αCγ, i, k]. \]

Figure 1: The operators for head-corner chart parsing.

program chart parser
begin
  create initial chart and agenda;
  while agenda is not empty do
    delete (arbitrarily chosen) current item from agenda;
    if current ∉ chart then
      add current to chart;
      add all items to agenda that can be recognized by any operator using current and items in chart
    fi
  od
end.

Figure 2: General schema for a chart parser.
Figure 3: The completed head-corner chart for the sentence ‘The cat catches a mouse’.

Figure 4: The parse tree of the sentence ‘The cat catches a mouse’.
chosen (see e.g. [StS93]) but here these examples will do. In this paper a notation is used that is almost identical to PATR-II ([Shi86]). As in PATR, we may replace formal variables $X_0, X_1, X_2, \ldots$ by the syntactic categories (i.e. cat features) of these variables. The categories are not mentioned again in the feature constraints. This way the context-free parts of the grammar and lexicon are the same as in Section 1.

$$S \rightarrow NP\ VP$$
$$S : sem = VP : sem$$
$$S : syn = NP : syn$$
$$VP : syn = number = NP : syn : number$$
$$VP : syn = person = NP : syn : person$$

$$VP \rightarrow \text{verb}\ NP$$
$$VP : sem = action = verb : sem$$
$$VP : sem = object = NP : sem$$
$$VP : syn = number = verb : number$$
$$NP : syn = number = verb : number$$

$$NP \rightarrow \text{det}\ noun$$
$$NP : sem = noun : sem$$
$$NP : syn = number = noun : number$$
$$det : number = noun : number$$

Figure 5: Grammar with feature constraints

$$\text{det} \rightarrow \text{the}$$
$$\text{det} \rightarrow \text{a}$$
$$\text{noun} \rightarrow \text{cat}$$
$$\text{noun} : sem = cat$$
$$\text{noun} : number = singular$$
$$\text{verb} : person = 3rd$$
$$\text{noun} \rightarrow \text{mouse}$$
$$\text{noun} : sem = mouse$$
$$\text{noun} : number = singular$$
$$\text{noun} : person = 3rd$$
$$\text{verb} \rightarrow \text{catches}$$
$$\text{verb} : sem = catching$$
$$\text{verb} : number = singular$$
$$\text{verb} : person = 3rd$$

Figure 6: Lexicon with feature constraints

One important difference with the context-free case is that a grammar production or a lexical entry can occur more than once with a different set of constraints. This double occurrence can be necessary because in PATR-II it is the only possibility to describe certain phenomena like disjunction and negation. So the distinction between grammar productions or lexical entries is decided by both the syntactical categories and the feature constraints.

For practical reasons the feature constraints are put together in feature structures, one structure for each constituent in a production or entry (see also [Shi86]). A common way to noteate feature structures are attribute value matrices (AVMs). For example in Figure 7 the feature structures belonging to (the constituents in) the production $NP \rightarrow \text{det}\ noun$ are represented as such matrices. In Figure 8 the feature structure belonging to (the noun in) the entry $noun \rightarrow \text{mouse}$ is given.

$$NP \rightarrow \begin{bmatrix}
\text{sem} & 1 \\
\text{syn} & \begin{bmatrix}
nr & 2 \\
\end{bmatrix}
\end{bmatrix}$$

$$det \rightarrow \begin{bmatrix}
noun & 2 
\end{bmatrix}$$

$$noun \rightarrow \begin{bmatrix}
\text{sem} & \text{mouse} \\
\text{number} & \text{singular} \\
\text{person} & \text{3rd}
\end{bmatrix}$$

Figure 7: Feature structures in $NP \rightarrow \text{det}\ noun$

Figure 8: Feature structure of the noun 'mouse'

If nothing is known about a structure the structure is empty, denoted as $\begin{bmatrix} \end{bmatrix}$. If a (partial) feature structure is reachable by different paths, this is denoted with a numbered box; the same number means same structure. This so-called coreferenceootnote{reentrancy in [Shi86] on path-equivalence in [KR86]} is the main mechanism of unification grammars. Via coreferences information can be synthesized from the lexicon and grammar into the final feature structure of a sentence $S$.

2.2 Items and Operators

In Section 2.1 the input for a head-corner parser for unification grammars has been described. In this section the main adaptations of the items and operators will be given.
First all constituents of the items get feature structures like the constituents in the grammar and lexicon have. The terminal items get their feature structures straight from the lexicon. If a word has two entries in the lexicon we have to create two terminal items, possibly with the same context-free part, but with different feature structures. In the initial predict item \([n, q, S]\) we know nothing yet about the features of \(S\), so \(S\) has an empty structure. Now the feature structures of all initial items are fixed.

All other items on the chart are results of operators applied to already recognized items. Therefore it is sufficient to give a description for each operator how the feature structures in the new items are determined by the ones in the old items. To get a compact description a few short notations are introduced:

\[
X_f \text{ in } p \text{ : the feature structure of constituent } X \text{ in production } p \text{ (as explained in Section 2.1)};
\]

\[
X_f \text{ in } it \text{ : the feature structure of constituent } X \text{ in item } it;
\]

\[
f_1 \sqcup f_2 \text{ : the result of unification of feature structure } f_1 \text{ with feature structure } f_2.
\]

In Figure 9 the description is given of constituents in newly recognized items. With this description the values of all features in the new items are fixed. Note that coreferences between constituents within an item have to be retained, because these embody constraints that are imposed by the lexicon and the productions. But the coreferences across items are undesirable. An item or a production can be used more than once for the recognition of a new item. In most cases the new items belong to different parses of one sentence. Therefore we do not want the new items to be connected via the old item respectively the production.\footnote{More information on this problem can be found in §7.7.1 of [GM89].}

The simplest solution to this problem is to make copies of the old feature structures. Only in case of unification we do not copy the operands of the unification but we use a non-destructive unification algorithm. For all feature structures that depend on only one other feature structure we still have to make copies. As can be seen in Figure 9 many copies have to be made.

Parallel to the new lexicon and grammar an important difference with the context-free case is that an item can now occur more than once in the chart only with different feature structures. So the distinction between items is determined by both their context-free parts and the feature structures of their constituents. This distinction is important to determine whether a newly recognized item is on the chart already or should be added to the chart.

Finally the chart will contain one or more items of the form \([n, q, S; \alpha] \rightarrow \delta_0, 0, n\). In each item of this form the feature structure of \(S\) (right of the semicolon) is a feature structure of the sentence \(\alpha \ldots \alpha_n\) that has been parsed.

With the grammar and lexicon of Section 2.1 the chart for the sentence ‘the cat catches a mouse’ is the same as the one in Figure 3 apart from the feature structures. There were no words that occurred more than once in the lexicon (Figure 8) so the terminal items are the same. In the grammar (Figure 7) no production occurred more than once either, so there are no extra possibilities to apply a head-corner operator. The feature structure of the sentence (second \(S\) in item (15)) is given in Figure 10.

\[
S \rightarrow \begin{bmatrix}
\text{syn : } [\text{voice : active}] \\
\text{sem : } [\text{scene : } [\text{action : catching}] \\
\text{subject : cat} \\
\end{bmatrix}
\]

Figure 10: Feature structure of the sentence ‘The cat catches a mouse’.

3 THE IMPLEMENTATION

After adapting the head-corner algorithm I implemented a head-corner parser for unification grammars. As usual still many decisions had to be made to do so. A few of those decisions will be described in this section. I wrote the code for the head-corner parser in the imperative programming language Modula-2.

3.1 INPUT PROCESSING

A lot of information from the lexicon and grammar has to be made explicit. For example we say easily ‘\(a \in \Sigma\)’, ‘\(A \Rightarrow^* B\)’ or ‘for all \(p \in P\)’. To be able to handle all expressions used in the description of the algorithm we need to put the lexicon
\textbf{head corner (i)}: newitem = \{l, r, A; B \rightarrow \alpha \beta \gamma, j - 1, j\} and p = B \rightarrow \alpha \beta \gamma

\text{A}_f \text{ in newitem is as } A_f \text{ in } \{l, r, A\}

\text{B}_f \text{ in newitem is as } B_f \text{ in } p

\text{X}_f \text{ in newitem is as } X_f \text{ in } p, \text{ for all } X \in \alpha \gamma

\text{b}_f \text{ in newitem is as } (b_f \text{ in } \{b, j - 1, j\}) \cup (b_f \text{ in } p);

\textbf{head corner (ii)}: newitem = \{l, r, A; B \rightarrow \alpha C \gamma, i, j\} and p = B \rightarrow \alpha C \gamma

\text{A}_f \text{ in newitem is as } A_f \text{ in } \{l, r, A; C \rightarrow \delta \xi, i, j\}

\text{B}_f \text{ in newitem is as } B_f \text{ in } p

\text{X}_f \text{ in newitem is as } X_f \text{ in } p, \text{ for all } X \in \alpha \gamma

\text{C}_f \text{ in newitem is as } (C_f \text{ in } \{l, r, A; C \rightarrow \delta \xi, i, j\}) \cup (C_f \text{ in } p);

\textbf{head corner (iii)}: newitem = \{l, r, A; B \rightarrow \ast, j, j\} and p = B \rightarrow \varepsilon

\text{A}_f \text{ in newitem is as } A_f \text{ in } \{l, r, A\}

\text{B}_f \text{ in newitem is as } B_f \text{ in } p;

\textbf{predict (left)}: newitem = \{l, i, C\}

\text{C}_f \text{ in newitem is as } C_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta \gamma, i, j\};

\textbf{predict (right)}: newitem = \{j, r, C\}

\text{C}_f \text{ in newitem is as } C_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta \gamma, i, j\};

\textbf{scan (left)}: newitem = \{l, r, A; B \rightarrow \alpha \beta \gamma, j - 1, k\}

\text{A}_f \text{ in newitem is as } A_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta \gamma, j, k\}

\text{B}_f \text{ in newitem is as } B_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta \gamma, j, k\}

\text{X}_f \text{ in newitem is as } X_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta \gamma, j, k\}, \text{ for all } X \in \alpha \beta \gamma

\text{a}_f \text{ in newitem is as } (a_f \text{ in } \{a, j - 1, j\}) \cup (a_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta \gamma, j, k\});

\textbf{scan (right)}: newitem = \{l, r, A; B \rightarrow \alpha \beta \gamma, i, j + 1\}

\text{A}_f \text{ in newitem is as } A_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta \gamma, i, j\}

\text{B}_f \text{ in newitem is as } B_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta \gamma, i, j\}

\text{X}_f \text{ in newitem is as } X_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta \gamma, i, j\}, \text{ for all } X \in \alpha \beta \gamma

\text{a}_f \text{ in newitem is as } (a_f \text{ in } \{a, j, j + 1\}) \cup (a_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta \gamma, i, j\});

\textbf{complete (left)}: newitem = \{l, r, A; B \rightarrow \alpha \beta C \gamma, i, k\}

\text{A}_f \text{ in newitem is as } A_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta C \gamma, j, k\}

\text{B}_f \text{ in newitem is as } B_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta C \gamma, j, k\}

\text{X}_f \text{ in newitem is as } X_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta C \gamma, j, k\}, \text{ for all } X \in \alpha \beta \gamma

\text{C}_f \text{ in newitem is as } (C_f \text{ in } \{l, j, C_1\}) \cup (C_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta C \gamma, j, k\});

\textbf{complete (right)}: newitem = \{l, r, A; B \rightarrow \alpha \beta C \gamma, i, k\}

\text{A}_f \text{ in newitem is as } A_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta C \gamma, i, j\}

\text{B}_f \text{ in newitem is as } B_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta C \gamma, i, j\}

\text{X}_f \text{ in newitem is as } X_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta C \gamma, i, j\}, \text{ for all } X \in \alpha \beta \gamma

\text{C}_f \text{ in newitem is as } (C_f \text{ in } \{j, r, C_1; C_2 \rightarrow \delta \xi, j, k\}) \cup (C_f \text{ in } \{l, r, A; B \rightarrow \alpha \beta C \gamma, i, j\}).

Figure 9: Extension of the operators to feature structures.
and grammar into datastructures and determine several properties.

**Building Feature Structures**
We can think of feature structures as directed acyclic graphs (DAGs). For example, the feature structure of *S* in Figure 10 can be represented as the graph in Figure 11 (see also [Shi86]). Those feature graphs were implemented with pointers.

![Figure 11: The feature graph of the sentence 'The cat catches a mouse'.](image)

**Properties of the Categories**
After putting the grammar and lexicon in datastructures, first a list of all syntactic categories was made. For each category it was determined whether it was a terminal, a nonterminal, or even both. During testing with realistic grammars it turned out that a grammar writer possibly wants to use a lexical category as a nonterminal as well. The Plinius grammar (see Section 4) contains for example the production `noun → noun noun`.

Next the head-corner relation and its closure was determined. The transitive and reflexive closure of this relation (>{1}) was derived using an algorithm of Yellin ([Ye88]).

### 3.2 Operations on Feature Structures
We have seen in Section 3.1 that the implementation of the feature structures was not straightforward. Also during parsing the feature structures needed special attention.

**Copying**
For the independence between items we have to copy feature structures. In order to retain the coreferences within one feature structure, the copying was done as follows. To copy one feature structure a list was made of all the vertices in that structure. For each vertex it was noted whether it had been copied and which vertex was its copy. Walking through the original feature structure a new feature structure was built. If a vertex had been copied, the copy found in the list was used and otherwise a copy was made and used, and the list was updated.

By application of some operators we have to copy a whole production (out of the grammar or an item) together with all its interconnected feature structures. In these cases the feature structures were copied one by one using only one vertex-list for all feature structures. This way the coreferences between the feature structures belonging to that one production were retained.

**Unification**
We use a non-destructive adaption of Huet's unification algorithm ([Hu76]), that is similar to the congruence closure algorithm of Nelson and Oppen ([NO80]), cf. Wroblewski ([W87]). See Chapter 9 of [Sik83] for more details.

### 3.3 An Alternative Distinction Between Items
In Section 2.2 we talked about the distinction between items with feature structures. The most straightforward way to determine this distinction seems comparison of the context-free parts and of the feature structures. But the feature structures in the grammar production of a head-corner item can be connected by coreferences. This makes the comparison of two head-corner items with both a set of interconnected feature structures rather complicated. That is why I worked out another solution. This easier way to compare items will be described in this section.

Predict items and the predict part (before the semicolon) of head-corner items were compared just by looking at the context-free parts and the feature structures. For each terminal item and for the production part (after the semicolon) of each head-corner item a parse tree was created. These parse trees are only intuitively introduced.
In fact the trees are the usual parse trees (see Figure 4) annotated with the productions and lexical entries involved in the parsing. To annotate the trees we number the productions and the entries (up from 1). The parse tree for item (13) from the chart of the running example is given in Figure 12. Earlier items like (13) and (11) have unparsed strings outside the dots. All constituents in these strings are represented by empty branches in the parse trees. Terminal items get parse trees consisting of only one leaf, the lexical entry.

This way all sources of feature information about the parsed phrase (all used grammar productions and lexical entries) are referred to in the annotated parse tree. Differences between the feature structures of items can only occur by use of different grammar productions or lexical entries, so all differences are noticed by comparison of the annotated parse trees.

In exceptional cases the feature structures of two items can be the same while their annotated parse trees are different. In these cases the grammar writer should think well about what he wants to achieve with the different productions and entries that lead to the same feature structures. If only the annotations in the parse trees are different it is a warning to the grammar writer for possible redundancy in the grammar or lexicon. If the parse trees differ in the constituents it is good to make a distinction between the items, because they belong to totally different parses.

Here we have arrived at another advantage (besides the easier comparison) of keeping the annotated parse trees in the items. We do not have to walk through the completed chart to find the parse trees. After parsing, each item of the form \([0, n, 5; 5 \rightarrow \cdot, 0, n]\) is associated with one parse tree. For convenience a list of references to final items is also kept during parsing.

4 TEST RESULTS

Within the Department of Computer Science at the University of Twente there is a project on knowledge acquisition, called 'Plinius' (see also [StS93]). In this project a grammar and a lexicon with feature constraints are being developed. The Plinius corpus is a set of 400 English abstracts of technical articles on the mechanical properties of ceramic materials. Sentences from this corpus are being parsed with a simple parser. Sentences, grammar and lexicon of this project were used to test the head-corner parser.

4.1 HEAD-CORNER PARSER VERSUS PLINIUS PARSER

The Plinius parser is based on work of Gazdar and Mellish ([GM89]). It is also a parser for unification grammars. The parser works bottom-up and from left to right and has been written in PROLOG. All solutions are being looked for by means of backtracking. This mechanism makes the worst-case efficiency of the parser rather bad. That is why a more efficient parser was desired.
<table>
<thead>
<tr>
<th></th>
<th>large lexicon (52 words)</th>
<th>small lexicon (14 words)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>large grammar (20 productions)</strong></td>
<td>sent.1, sent.2, sent.3</td>
<td>sent.1, sent.2, sent.3</td>
</tr>
<tr>
<td><strong>small grammar (11 productions)</strong></td>
<td>sent.1, sent.2</td>
<td></td>
</tr>
</tbody>
</table>

sent.1 = 'the material exhibited elongation', 1 parse  
sent.2 = 'the material exhibited elongation in a test at temperatures', 5 parses  
sent.3 = 'the material exhibited superplastic elongation in a tension test at temperatures and at strain rates', 7 parses

Figure 13: Test plan Plinius- versus head-corner parsing

<table>
<thead>
<tr>
<th><strong>Plinius</strong></th>
<th>sent.1</th>
<th>sent.2</th>
<th>sent.3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>large lexicon</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>large grammar</td>
<td>2.70</td>
<td>40.85</td>
<td>2903,10 (=48,4 min.)</td>
</tr>
<tr>
<td>small lexicon</td>
<td>0.28</td>
<td>40.66</td>
<td>2893,95 (=48,2 min.)</td>
</tr>
<tr>
<td>large lexicon</td>
<td>1.62</td>
<td>4.65</td>
<td>—</td>
</tr>
<tr>
<td>small grammar</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Head-corner</strong></th>
<th>sent.1</th>
<th>sent.2</th>
<th>sent.3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>large lexicon</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>large grammar</td>
<td>1.54</td>
<td>15,50</td>
<td>140,09 (=2,33 min.)</td>
</tr>
<tr>
<td>small lexicon</td>
<td>1.38</td>
<td>15,02</td>
<td>139,38 (=2,32 min.)</td>
</tr>
<tr>
<td>large lexicon</td>
<td>1,04</td>
<td>12,44</td>
<td>—</td>
</tr>
<tr>
<td>small grammar</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 14: Parsing times of the Plinius- and the head-corner parser in seconds
<table>
<thead>
<tr>
<th>sent.1</th>
<th>sent.2</th>
<th>sent.3</th>
<th>sent.4</th>
<th>sent.4*</th>
<th>sent.5</th>
<th>sent.5*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left-corner</td>
<td>32</td>
<td>126</td>
<td>381</td>
<td>28</td>
<td>69</td>
<td>32</td>
</tr>
<tr>
<td>Head-corner</td>
<td>37</td>
<td>205</td>
<td>962</td>
<td>34</td>
<td>35</td>
<td>36</td>
</tr>
<tr>
<td>change</td>
<td>+17%</td>
<td>+63%</td>
<td>+152%</td>
<td>+21%</td>
<td>-49%</td>
<td>+13%</td>
</tr>
</tbody>
</table>

* with a lexical ambiguity (‘yield’ as verb and as noun in the lexicon)

sent. 4 = 'polycrystals yield hydroxyapatite', 1 parse
sent. 5 = 'hydroxyapatite increases the yield', 1 parse

Figure 15: The number of created items at left-corner- and head-corner parsing

After implementation of the head-corner parser a comparison has been made of the Plinius parser and the head-corner parser.

As the programs of the Plinius parser and the head-corner parser differ a lot, they can only be compared on the basis of the time they need to parse a certain sentence. The cpu-time is chosen because the real time varies strongly with the number of users on the machine. The needed cpu-time can be different on different machines so the programs have been tested on the same machine. Moreover, all tests have run ten times and the measurements have been averaged. The combinations of sentence, grammar and lexicon that have been tested are given in a schema (Figure 13). As testing was not the main purpose of the implementation, a limited number of cases has been tested. Testing with the large grammar and large lexicon is the most realistic situation. This situation can be simplified by reducing either the grammar or the lexicon. To learn about the influence of the size of grammar and lexicon, there have been testing in the two alternative situations too. The smaller grammar does not have enough productions to parse sentence 3 with. The Plinius grammar did not have heads so they were given in a linguistically usual way. Often the constituent in the righthand side that has the most feature information in common with the lefthand side, is taken as the head.

The results of the tests are in Figure 14. As we can see, the parsing time of the head-corner parser is much shorter than the time of the Plinius parser in most cases. The reduction varies from 36% to 95%. In two cases the Plinius parser is faster. But these are no realistic situations because either the grammar or the lexicon had been reduced artificially to reduce the parsing time of a specific sentence.

### 4.2 Head-corner Parsing versus Left-Corner Parsing

In [SA92] the head-corner parser is introduced by analogy to a left-corner parser. Left-corner parsers have been described by many authors. A left-corner parser parses sentences from left to right without jumping up and down the sentence. Usually left-corner parsing is described in simpler notations than used in this article. Less administration is needed because there are less possible orders for parsing. Still it is possible to describe left-corner parsing in the same notation as head-corner parsing. This looks a lot like head-corner parsing with a grammar that has all its heads extremely left. This way we can compare the efficiency of the left- and the head-corner parser by counting the number of created items.

Once the head-corner parser had been implemented it was easy to adapt it to use it as a left-corner parser. For the comparison the same three sentences were used as in the comparison with the Plinius parser, plus two sentences to learn about the effect of lexical ambiguities. Figure 15 shows the result of these tests. Sentences 4 and 5 have first been parsed with a lexicon with only
the right entry for 'yield' and again with a lexicon
with two entries (*-marked column). As we can
see, in most cases the left-corner parser gives a
better result.

Still, the head-corner parser can be of value.
By introducing heads in the grammar produc-
tions the parsing can be more guided. If there
are more discriminating parts in a phrase they
can be parsed first in order to avoid false parses.
Only the head-corner parser should be optimized
in one very important respect. The head-corner
parser described in this article suffers from the
fact that it does not use alignment information.
We take for example the sentence 'the elastic ma-
terial bends in the test'. For every constituent
'the elastic material ...' similar constituents 'elas-
tic material ...' and 'material ...' are recognized
as well. An optimized head-corner parser should
take only subjects that start at position 0. Im-
portant here is that the left-corner parser can be
seen as a special case of the optimized head-corner
parser, rather than the implemented head-corner
parser: the left-corner parser does not leave gaps.

5 CONCLUDING REMARKS

Everything that has been noticed while working
with the parser and that can be used for further
work, will be discussed in this last section. First,
a few stray observations are mentioned under the
title 'Experiences'. Ways to improve the parser
are given in Section 5.2. In the final section con-
clusions will be given.

5.1 EXPERIENCES

One limitation of the parser appeared when it
was used in the Plinius project. The Plinius
parser searches automatically for every possible
category. The head-corner parser looks only for
the category it is asked to look for by the ini-
tial predict item, in general \([0, n, S]\). The head-
corner parser can be adapted so that it searches
all possible categories between the place markers
0 and n. But when this adaption had been made,
the parser had become very slow. However, this
adaption is probably not that much needed.

One benefit of the implementation is the cal-
culated waste factor. The waste factor can be
found empirically now and this may be helpful in
developing theories on parsing. After counting all
items that were needed to find the final item(s)
the waste factor is calculated by:

\[
\text{waste factor} = \frac{\text{total} \# - \text{needed} \#}{\text{total} \#}
\]

For head-corner parsing of the sentences in Fig-
ure 15 the waste factor varies from 33% to 87%.
The factor is higher for the longer sentences and
charts. One thing noticed while calculating the
waste factors is that the number of needed items
differs sometimes for the left- and head-corner
parsing of one sentence. Sometimes one or more
extra predict items are used to achieve the same
result.

5.2 POSSIBLE IMPROVEMENTS

During the eight months working with, reading
and writing about the head-corner parser several
suggestions for improvement of the parser have
come up. They were found in literature or arose
by 'playing' with the parser. Successively im-
provement of the preprocessing, typical improve-
ments of a chart parser by using the place mark-
ers, and improvements with respect to the feature
structures are discussed in this section.

Preprocessing

Of course the grammar and lexicon—among
others—determine the performance of the parser.
Actually we have to consider the grammar and
lexicon a given input. But, we could ask the
grammar writer to take care of a few things or
we could adapt the input before parsing.

There are two sources of inefficiency in the
grammar. Syntactic categories that are both ter-
minimal and nonterminal should be avoided. Head-
recursive productions also lead to inefficiency.\(^4\)
An interesting question for further study is
whether the efficiency can be improved by spe-
cial treatment of these constructions.

Moreover, we can improve the results of the
parser by reducing the lexical ambiguity. One
way to do so is to derive from the grammar
precedence- and follower-relations between all
categories. When we have a word with several
lexical entries we can possibly forget about a few
categories and shorten the initial chart. This is
a simple adaption that can lead to a major effi-
ciency gain.

Place markers in chart parsers

In [SA92] two suggestions for improvement of the
head-corner chart parser are given. First, when

\(^4\)A head recursive production is of the form \(A \rightarrow \alpha \ A \beta\),
in analogy to left recursive productions.
an item \( \ldots; A \rightarrow \alpha B \tau, i, j \) has been recognized by the head-corner rule, with \( \alpha \neq \varepsilon \neq \tau \), it should be expanded either to \( \ldots; A \rightarrow \alpha B \tau, j, i \) or to \( \ldots; A \rightarrow \alpha B \tau, i, k \) but not in both directions; from either one a completed item \( \ldots; A \rightarrow \alpha B \tau, h, i \) or \( \alpha B \tau, h, k \) can be obtained. This was first suggested by Satta and Stock ([SS89]).

Second, the parser should use alignment information in order to avoid parsed parts of one sentence that do not fit together (see also Section 4.2).

Furthermore, the place markers of items can be used to organize the chart. This organization (like any arrangement) of the recognized items speeds up searches for items that — together with the current item — can be used as operands for an operation.

**Feature structures**

Finally, some actions on feature structures can be made more efficient and the feature structures themselves can be used to improve the performance of the parser.

Copying and unification of feature structures can be made more efficient by subgraph sharing (see [Kog90]). Subgraph sharing means that a vertex is only copied when it is actually used for different things. Until this double use, the vertex (and the feature graph connected to it) is shared by different feature structures. This way the number of vertices to be copied is reduced.

Besides being a source of inefficiency, feature structures can also be of help. Feature structures can be used for a form of top-down filtering as will be explained below. In the head-corner parser the predict items and the predict parts of the head-corner items contain feature information. This information could be used to check in an early stage whether the constituent that is being build can ever contribute to what is predicted. If it can not, we can stop (or ‘filter’) such parses that are of no use. In order to do so, it is necessary to know what features from the lefthand side are inherited\(^5\) by the heads. In general this does not count for all features, so realizing the outlined filtering involves more than just unification of a predicted constituent with a found one. In order to improve the head-corner parser, top-down filtering is an option. But as long as this filtering is not realized (and it still is not) another choice of heads may be better. In stead of choosing heads on the basis of correspondence of feature infor-

\(^5\) to inherit a feature: to have the same feature and with the same value.

**5.3 Conclusions**

We have successfully applied the head-corner parsing algorithm for unification grammars to the Plinius grammar. This confirms that head-corner parsing, previously described mainly on theoretical level, can be practically applied.

The left-corner parser, however, seems to be slightly more efficient than the head-corner parser. But it should be noticed that there is much room for optimization in the implementation of the head-corner parser, that will tip the balance (for this grammar) in favor of head-corner parsing.

As for unification grammars, in general the usefulness of head-corner parsing will strongly depend on the properties of the grammar. See also Bouma and Van Noord ([BV93]), who found that it depends on the grammar whether head-corner parsing is useful.

**Acknowledgements**

I would like to thank Mark Moll for writing the programming code for the unification of feature structures, and Wim van der Veer for programming an organized chart. Riks op den Akker, Toine Anderman, and Klaas Sikkels have given helpful comments on a draft version of this paper.

**References**


