# Robust Parsing with the Head-corner Parser

Gertjan van Noord
Vakgroep Alfa-informatica & BCN
vannoord@let.rug.nl
Rijksuniversiteit Groningen

July 18, 1996

## 1 Introduction

The head-corner parser is currently being used in the the NWO Priority Programme on Language and Speech Technology. This programme aims at the construction of a Dutch spoken dialog system for public transport information: OVIS. For this purpose a robust and efficient parser is required. Elsewhere I have shown that the head-corner parser can be a very efficient parser for constraint-based natural language grammars (Bouma and van Noord, 1993; van Noord, 1996). This paper discusses how the head-corner parser is used for robust processing.

Certain approaches towards robust parsing take as a starting point the 'inactive items' that were found by the parser. Such an approach requires that the parser discovers all categories anywhere in the input in a bottom-up fashion. It is therefore natural to consider a bottom-up parser in such a context, since otherwise the set of inactive items the parser constructs in case of failure will not be complete.

Head-corner parsing is a mixture of bottom-up and top-down parsing. Therefore it cannot be guaranteed that the parser finds all possible categories. In this paper I will argue that the use of underspecification can be exploited to obtain a version of the head-corner parser which is guaranteed to discover all *relevant* categories anywhere in the input. The grammar explicitly defines what categories are relevant in this sense. It is shown how this property is used in the design of the robustness component of the OVIS system.

An important issue in the context of robustness concerns evaluation. In the context of the OVIS system a new and simple evaluation criterion is available. In OVIS the input for the parser consists of the output of the speech recognizer. Given that the speech recognizer often is uncertain about what is actually said, it produces a word-graph of possible sentences (Oerder and Ney, 1993). This makes it possible to measure the success of the parser by comparing the path in the word-graph on which the parser bases its best analysis with the actual utterance. Different comparison operators will be discussed.

This paper is organised as follows. I briefly introduce the head-corner parser and some of the techniques to make the parser more efficient in the following section. A more detailed discussion can be found in van Noord (1996). Section 3 discusses the use of underspecification to make sure that all relevant categories anywhere in the input are found. In the fourth section I discuss the internal evaluation method that is being used. Section 5 presents some preliminary results.

## 2 Head-corner Parsing for Natural Language Understanding

### 2.1 Representation

I assume that grammars are defined in the Definite Clause Grammar formalism (Pereira and Warren, 1980). Without any loss of generality I assume that no external Prolog calls (the ones that are defined within { and }) are used. Moreover I will assume that such a grammar is represented in a somewhat different manner to make the definition of the parser easier, and to make sure that rules are indexed appropriately. This representation will in practice be compiled from a representation in user-friendly notation.

More specifically, I will assume that grammar rules are represented by the predicate `headed_rule/4` in which the first argument is the head of the rule, the second argument is the mother node of the rule, the third argument is the reversed list of daughters left of the head, and the fourth argument is a list of the daughters right of the head.

As an example, the DCG rule

```
x(A,E) --> a(A), b(B,A), x(C,B), d(C,D), e(D,E).        (1)
```

of which the third daughter constitutes the head, is represented now as:

```
headed_rule( x(C,B), x(A,E), [b(B,A), a(A)], [d(C,D), e(D,E)]).        (2)
```

I will be assuming furthermore that lexical lookup has been performed already by another module. This module has asserted clauses for the predicate `lexical_analysis/3` where the first two arguments are the string positions and the third argument is the (lexical) category. For an input sentence 'Time flies like an arrow' this module may produce the following set of clauses:

```
lexical_analysis(0,1,verb).   lexical_analysis(0,1,noun).
lexical_analysis(0,2,noun).   lexical_analysis(1,2,noun).
lexical_analysis(1,2,verb).   lexical_analysis(2,3,prep).        (3)
lexical_analysis(2,3,verb).   lexical_analysis(3,4, det).
lexical_analysis(4,5,noun).
```

## 2.2 The Basics

A simple non-deterministic version of the head-corner parser is now given as follows. The predicate visible to the rest of the world will be the predicate `parse/3`. This predicate is defined in terms of the `parse/5` predicate. The extra arguments introduce a pair of indices which represent the extreme positions between which a parse should be found. This is explained in more detail in (van Noord, 1996).

```
% parse(?Cat,+P0,+P)
parse(Cat,P0,P) :- parse(Cat,P0,P,P0,P).        (4)
```

A goal category can be parsed if a predicted lexical category can be shown to be a head-corner of that goal.

```
% parse(?Cat,?P0,?P,+E0,+E)
% there is a category Cat from P0-P within the interval E0-E
parse(Cat,P0,P,E0,E) :-        (5)
    predict(Cat,P0,P,SmallCat,Q0,Q),
    head_corner(SmallCat,Q0,Q,Cat,P0,P,E0,E).
```

The head-corner predicate constructs in a bottom-up fashion larger and larger head-corners.

```
% head_corner(?Small,+P0,+P,?Cat,?Q0,?Q,+E0,+E)
% Small from P0-P is a head-corner of Cat from Q0-Q
% where Q0-Q occurs within E0-E
head_corner(Cat,P0,P,Cat,P0,P,_,_).
head_corner(Small,Q0,Q,Cat,P0,P,E0,E) :-        (6)
    headed_rule(Small,Mother,RevLeftDs,RightDs),
    parse_left_ds(RevLeftDs,QL,Q0,E0),  parse_right_ds(RightDs,Q,QR,E),
    head_corner(Mother,QL,QR,Cat,P0,P,E0,E).
```

To parse a list of daughter categories we have to parse each daughter category in turn.

```
% parse_left_ds(+RevLeftDs,-Qleft,+Qright,+ExtremeLeft)
% there are categories LeftDs from Qleft to Qright
% s.t. RevLeftDs is reverse of LeftDs, and ExtremeLeft < Qleft.
parse_left_ds([],Q,Q,_).
parse_left_ds([H|T],Q0,Q,E) :-
    parse(H,Q1,Q,E,Q), parse_left_ds(T,Q0,Q1,E).
                                                                    (7)
% parse_right_ds(+RightDs,+Qleft,-Qright,+ExtremeRight)
% there are categories RightDs from Qleft to Qright s.t.
% Qright < ExtremeRight.
parse_right_ds([],Q,Q,_).
parse_right_ds([H|T],Q0,Q,E) :-
    parse(H,Q0,Q1,Q0,E),  parse_right_ds(T,Q1,Q,E).
```

A predicted category must be a lexical category that lies somewhere between the extreme positions.

```
% predict(?Cat,-P0,-P,+E0,+E,-SmallCat,-Q0,-Q)
% SmallCat from Q0-Q is a lexical category and possible
% head-corner of Cat from P0-P, where P0-P is within E0-E.
predict(_Cat,_P0,_P,E0,E,Small,Q0,Q) :-                             (8)
    lexical_analysis(Q0,Q,Small),  between(Q0,Q,E0,E).
% between(+P0,+P,+E0,+E) P0-P occurs within E0-E
between(P0,P,E0,E) :-  E0 =< P0, P =< E.
```

## 2.3 Improvements

The simple and non-deterministic version of the head-corner parser given above is turned into a very efficient parser by making use of the following techniques. Each of these techniques is explained in (van Noord, 1996) in more detail:

- the use of top-down information by a table representing the head-corner relation. Furthermore indexing is used for efficient table lookup. The head-corner relation includes information on the start and end positions (e.g. the requirement that the head of an sbar is a complementizer *at the left-most position* of that sbar phrase.
- the use of underspecification of position information in the case of empty productions (epsilon rules).
- the (limited) use of memo-ization. Memo-ization is only applied for the parse/5 predicate. This implies that each *maximal projection* is computed only once; partial projections of a head can be constructed during a parse any number of times, as can sequences of categories (considered as sisters to a head). Active chart parsers 'memo' everything; inactive chart parsers only memo categories, but not sequences of categories. In our proposal, we memo only those categories that are 'maximal projections', i.e. projections of a head which unify with the top category (start symbol) or with a non-head daughter of a rule.

  Note that nothing would prevent us to memo other predicates as well. Experience suggests that the cost of maintaining tables for e.g. the `head_corner` relation is (much) higher than the associated profit. The use of memoization for only the `parse/5` goals implies that the memory requirements of the head-corner parser in terms of the number of items that is being recorded is much smaller than in ordinary chart parsers. Not only do we refrain from asserting so-called *active* items, but we also refrain from asserting *inactive* items for non-maximal projections of heads. In practice the difference in space requirements is enormous (2 orders of magnitude). This difference may in turn be a significant reason for the practical efficiency of the head-corner parser. [1]

[1] Even if the table of items were indexed properly (which is somehwat difficult in Prolog) a non-trivial overhead remains because items must be compared with the subsumption check.

- the use of *goal-weakening*. The insight behind 'goal weakening' in the context of memoization is that we may combine a number of slightly different goals into a single more general goal. Very often it is much cheaper to solve this single (but more general) goal, than to solve each of the specific goals in turn. Clearly, one must be careful not to remove essential information from the goal (in the worst case this may even lead to non-termination of otherwise well-behaved programs).

  Depending on the properties of a particular grammar, it may for example be worthwile to *restrict* a given category to its syntactic features before we attempt to solve the parse goal of that category. Shieber's restriction operator (Shieber, 1985) can be used here. Thus we essentially throw some information away before an attempt is made to solve a (memoized) goal. For example, the category

  ```
  x(A,B,f(A,B),g(A,h(B,i(C))))
  ```
  (9)

  may be weakened into:

  ```
  x(A,B,f(_,_),g(_,_))
  ```
  (10)

  Note that goal weakening is sound. An answer $a$ to a weakened goal $g$ is only considered as an answer for $g$ if $a$ and $g$ unify. Also note that goal-weakening is complete in the sense that for an answer $a$ to a goal $g$ there will always be an answer $a'$ to the weakening of $g$ such that $a'$ subsumes $a$. For practical implementations the use of goal weakening can be extremely important. It is my experience that a well-chosen goal weakening operator may reduce parsing times with an order of magnitude.

- Compact representation of parse trees by *packing ambiguities*.

## 2.4 Parsing Word-graphs

The head-corner parser is one of the parsers developed within the NWO Priority Programme on Language and Speech Technology. In this program a spoken dialog system is developed for public transportation information (Boves et al., 1995)

In this system the input for the parser is not a simple list of words, but rather a word-graph: a directed, acyclic graph where the states are points in time, and the edges are labelled with word hypotheses and their corresponding probability. Thus, such word-graphs are acyclic weighted finite-state automata.

In certain approaches toward ill-formed input processing the desire to generalize from input strings to input finite-state automata is also clearly present. E.g., in Lang (1989) a framework for ill-formed input processing is described in which certain common errors are modelled as (weighted) finite-state transducers. The composition of an input sentence with these transducers produces a (weighted) finite state automaton which is then input for the parser.

The generalization from strings to weighted automata introduces essentially two complications. Firstly, we cannot use string indices anymore. Secondly we need to keep track of the probabilities of the words used in a certain derivation.

Parsing on the basis of a finite-state automaton can be seen as the computation of the intersection of that automaton with the grammar. It can be shown that if the definite-clause grammar is off-line parsable, and if the finite-state automaton is acyclic, then this computation can be guaranteed to terminate (van Noord, 1995). Moverover, existing techniques for parsing based on strings can be generalized easily by using the names of states in the automaton instead of the usual string indices.

In the head-corner parser, this leads to a change in the definition of the predicate `between/4`. Rather than a simple integer comparison, we now need to check that a derivation from `P0` to `P` can be extended to a derivation from `E0` to `E` by checking that there are paths in the word-graph from `E0` to `P0` and from `P` to `E`.

The predicate `between/4` is implemented using memo-ization as follows. It is assumed that state names are integers; to rule out cyclic word-graphs we also require that for all transitions from `P0` to `P` it is the case that `P0` < `P`. Transitions in the word-graph are represented by clauses of the form `wordgraph:trans(P0,Sym,P,Score)` which indicate that there is a transition from state `P0` to `P` with symbol `Sym` and probability `Score`.

```
between(L0,R0,L,R) :-  connection(L,L0), connection(R0,R).

connection(A,B):-
        (  var(A)          -> true
        ;  var(B)          -> true
        ;  A=:=B           -> true
        ;  B < A           -> fail   % word-graphs are acyclic
        ;  ok_conn(A,B)    -> true                                        (11)
        ;  fail_conn(A,B)  -> fail
        ;  wordgraph:trans(A,_,X,_),
           connection(X,B) -> assertz(ok_conn(A,B))
        ;  assertz(fail_conn(A,B)),
           fail
        ).
```

### 2.5 Accounting for Word-graph Probabilities.

In order to account for the probabilities of a derivation (defined as the product of the probabilites associated with all transitions from the word-graph involved in the derivation) we assume that the predicate `lexical_analysis` represents the probability of the piece of the word-graph that it covers by an extra argument. During the first phase probabilities are ignored. During the second phase (when a particular derivation is constructed) the probabilities are combined.

## 3 Head-corner Parsing and Robustness

Certain approaches towards robust parsing use the partial results of the parser. In order for such approaches to work it is often assumed that a bottom-up parser is essential: parsers that use top-down information (such as the head-corner parser) may fail to recognize relevant sub-parsers in the context of an ungrammaticality. We show how we use underspecification to obtain all relevant partial analyses for robustness.

Normally the head-corner parser will be called as e.g. :

```
?- parse(s(Sem),0,12).                                                   (12)
```

indicating that we want to parse a sentence from position 0 to 12 with category `s(Sem)` (a sentence with a semantic representation that is yet to be discovered). Suppose however that a specific robustness module is interested in all 'maximal projections' anywhere in the sentence. Such a maximal projection may be represented by a term `xp(Sem)`. Furthermore there may be unary grammar rules rewriting such an `xp` into appropriate categories, e.g.:

```
xp(Sem) --> np(Sem,_).   xp(Sem) --> s(Sem).
                                                                         (13)
xp(Sem) --> pp(Sem,_).   xp(Sem) --> advp(Sem).
```

If we want to recognize all maximal projections at all positions in the input, then we can simply give the following parse goal:

```
?- parse(xp(Sem),_,_).                                                   (14)
```

Now it may seem that such an underspecified goal will dramatically slow down the head-corner parser, but this turns out to be a false expectation. In actual fact we have experienced an increase of efficiency using underspecification. This can only be understood in the light of the use of memoization (van Noord, 1996). Even though we now have a much more general goal, the number of different goals that we need to solve is much smaller.

Note that the call to the parse predicate has variable extreme positions. This does not imply that all power of top-down prediction is lost; recursive calls to the parse predicate may still have instantiated left and/or right extreme positions. The same applies with even more force for top-down information on categories.

An example clarifies how the extreme position information can still be used even if the parse predicate is called with un-instantiated position arguments. For example, if at some point the parser is trying to find a *vp* and it predicts that its head is a verb between position $p_2$ and $p_3$ then, if a rule is selected which requires a *pp* to the left of that head, then it will be clear that for this *pp* the extreme right position will be $p_2$. Moreover, suppose that the head of this *pp* is found between position $p_0$ and $p_1$ then the parser can conclude that the *np*-complement of that preposition should be found between $p_1$ and $p_2$.

# 4 The Robustness Component in OVIS

In an attempt to obtain a robust natural language understanding component we have experimented in OVIS with the techniques mentioned in the preceding paragraph. The top category (start symbol) of the OVIS grammar is defined as the category `max(Sem)`. Moreover there are unary rules such as `max(Sem)` $\rightarrow$ `np(Sem,..)` for NP, S, PP, ADVP.

In the first phase, the parser finds all occurrences of the top category in the input word-graph. Thus, we obtain items for all possible maximal projections anywhere in the input graph. To obtain the best analysis for the word-graph as a whole the following procedure is being used. This robustness procedure consists of a best-first search from the beginning of the graph to the end of the graph. A path in the input graph can be constructed by taking steps of the following two types. To move from position $P$ to $Q$ you can either:

1. use a maximal projection from $P$ to $Q$ (as constructed by the parser)
2. use a transition from $P$ to $Q$. In this case we say that we 'skip' that transition.

In order to compare paths in the best-first search method, we define a score function which includes:

1. the number of skips. We prefer paths with a smaller number of such skips.
2. the number of maximal projections. We prefer paths with a smaller number of such projections.
3. the combined acoustic score as defined in the word-graph.

This best-first search method can be implemented in linear time; it is essentially the Viterbi algorithm for finding the most likely path through an HMM.

The resulting 'best' path in general consists of a number of maximal projections (connected by skips). In the OVIS application these often are simple time or place expressions. The pragmatic module is able to deal with such pieces of information and will perform better if it is given such partial parse results (as compared with an approach in which parser failure does not produce any information).

In practice, the best-first search will not try to use all categories as constructed by the parser. For this reason I have implemented the second phase of the parser in a lazy fashion: the second phase is only applied for a certain category if that category is considered in the best-first search. A table is maintained to make sure that for each category the second phase is only applied once.

We are experimenting with other criteria to be included in the scoring function:

- The semantic score. The resulting semantic representation is compared with the question asked previously (in the application all user utterances are reactions to the questions asked by the system). If the answer matches the question then a better score is obtained.
- The bigram score.

### 4.1 Evaluation of the Robustness component

In order to evaluate the appropriate combination of the factors determining the scoring function, and to evaluate this approach with respect to other approaches, we use a corpus of word-graphs for which we know the corresponding actual utterances. We compare the sentence associated with the 'best' path in the word-graph with the sentence that was actually spoken. Clearly if the robustness component more often uses the information that was actually uttered then we have more confidence in that component. The string comparison is defined by the minimal number of deletions and insertions that is required to turn the first string into the second (Levenshtein distance), although it may be worthwile to invest other measures.

For example, it seems likely that for our application it is less problematic to 'miss' information, whereas "hallucination" is a more severe problem. If information is missed, then the dialogue might become somewhat longer than neccessary. But if information is assumed for which no evidence existed in the actual utterences then misunderstandings might easily lead to confusion. This intuition could be formalized by a scoring function in which insertion is cheaper than deletion.

Currently the string that is being compared with the actual utterance is defined as the best path through the word-graph, given the best-first search procedure defined previously. However note that this implies that 'skips' are also part of that string. This seems undesirable because such skips have no effect on the semantic representations that result. Therefore, it may be better (at least for internal evaluation purposes) to count such skips as 'deletions' even if those words were actually spoken.

A final issue concerns the evaluation of the evaluation criterion. Clearly the task of the NLP component in the current framework is not to predict which sentence was actually spoken, but to construct the best possible semantic representations of it. Therefore the current evaluation technique should only be used together with more sophisticated techniques which should take into account the actual semantic representations that are constructed.

## 5 Some Results

This section presents a number of preliminary results to indicate how well the system currently performs. I used a corpus of 500 word-graphs, output of a preliminary version of the speech recognizer, and typical of the intended application. The size of these word-graphs is given in table 1

Currently the grammar assigns the following number of readings to each of these word-graphs (table 2). It can be concluded from this table that in about 20% of the cases the grammar cannot assign a full parse to the input.

The cputime and memory requirements of three different parsers are listed in the tables 3 and 4.

The final table lists the penalty score (as explained above) for the set of 500 word-graphs ( 5).

## References

Bouma, Gosse and Gertjan van Noord. 1993. Head-driven parsing for lexicalist grammars: Experimental results. In *Sixth Conference of the European Chapter of the Association for Computational Linguistics*, Utrecht.

| number of transitions | number of graphs |
|---:|---:|
| 0-5 | 250 |
| 6-10 | 103 |
| 11-15 | 50 |
| 16-20 | 26 |
| 21-30 | 23 |
| 31-40 | 16 |
| 41-50 | 8 |
| 51-75 | 15 |
| $> 75$ | 9 |

Table 1: The size of the test set of 500 word graphs

| number of readings | number of graphs |
|---:|---:|
| 0 | 104 |
| 1 | 261 |
| $\geq 2$ | 135 |
| 2 | 76 |
| 3 | 18 |
| 4 | 20 |
| 5 | 3 |
| 6 | 4 |
| 8 | 4 |
| 9 | 1 |
| 10 | 1 |
| 11 | 1 |
| 14 | 1 |
| 16 | 2 |
| 18 | 1 |
| 24 | 1 |
| 48 | 1 |
| 84 | 1 |

Table 2: Number of readings assigned to the word-graphs by the grammar

| parser | total (msec) | average | maximum |
|---|---:|---:|---:|
| head-corner | 50983 | 101 | 2820 |
| inactive-chart | 348867 | 697 | 37761 |
| bottom-up Earley | 443364 | 886 | 51460 |

Table 3: Cpu-time (in milliseconds) needed to analyse the test set of 500 word-graphs. Timings include the robustness component. Timings were obtained on a fast HP-UX 735. This machine is about three times faster than a Pentium 75MHz. The first parser is the head-corner parser as described in this paper. The second parser is an inactive bottom-up chart parser with packing. The third parser is a bottom-up Earley parser with packing.

| parser | total (Kbytes) | average | maximum |
|---|---|---|---|
| head-corner | 7280 | 14 | 299 |
| inactive-chart | 34025 | 68 | 1912 |
| bottom-up Earley | 84295 | 169 | 5403 |

Table 4: Memory requirements (in kilo-bytes) of the parsers. The figures listed indicate the size of the parse-tables.

| | total score | number of hits |
|---|---|---|
| speech (no bigram) best path | 771 | 277 |
| speech (no bigram) best possible | 170 | 402 |
| speech (with bigram) | 508 | 353 |
| nlp (robustness, no bigram, speech for skips) | 532 | 349 |
| nlp (robustness, no bigram, none for skips) | 557 | 341 |
| nlp (no robustness, no bigram, speech for skips) | 663 | 315 |
| nlp (no robustness, no bigram, none for skips) | 869 | 288 |

Table 5: Penalty scores for the set of 500 word-graphs. The best and worst possibilities are defined by the best path through the word-graph (which ignores bigrams) and the best possible path through this word-graph. Using bigrams currenlty works only slightly better than the NLP-component including the robustness component.


Boves, Lou, Jan Landsbergen, Remko Scha, and Gertjan van Noord. 1995. *Language and Speech Technology*. NWO Den Haag. Project plan for the NWO Priority Programme 'Language and Speech Technology'.

Lang, Bernard. 1989. A generative view of ill-formed input processing. In *ATR Symposium on Basic Research for Telephone Interpretation (ASTI)*, Kyoto Japan.

van Noord, Gertjan. 1995. The intersection of finite state automata and definite clause grammars. In *33th Annual Meeting of the Association for Computational Linguistics*, MIT Boston.

van Noord, Gertjan. 1996. An efficient implementation of the head-corner parser. Report available from `http://www.let.rug.nl/~vannoord/papers/`.

Oerder, Martin and Hermann Ney. 1993. Word graphs: An efficient interface between continuous-speech recognition and language understanding. In *ICASSP Volume 2*, pages 119–122.

Pereira, Fernando C.N. and David Warren. 1980. Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13.

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