Head Corner Parsing

Gertjan van Noord

Lehrstuhl für Computerlinguistik
Universität des Saarlandes
Im Stadtwald 15
D-6600 Saarbrücken 11, FRG
vannoord@coli.uni-sb.de

Abstract

I describe a head-driven parser for a class of grammars that handle discontinuous constituency by a richer notion of string combination than ordinary concatenation. The parser is a generalization of the left-corner parser and can be used for grammars written in powerful formalisms such as non-concatenative versions of UCG and HPSG.

1 Introduction

Although most constraint-based formalisms in computational linguistics assume that phrases are built by concatenation (eg. as in PATR II, GPSG, LFG and most versions of Categorial Grammar) this assumption is sometimes challenged by allowing more powerful operations to construct strings. For example, Pollard (1984) proposes several versions of 'head wrapping'. In the analysis of the Australian free word-order language Guugu Yimidhirr, Mark Johnson uses a 'combine' predicate in a DCG-like grammar that corresponds to the union of words (Johnson, 1985). Mike Reape uses an operation called 'sequence union' to analyse Germanic semi-free word order constructions (Reape, 1989; Reape, 1990). Other examples outside the family of constraint-based formalisms include Tree Adjoining Grammars (Joshi *et al.*, 1975; Vijay-Shankar and Joshi, 1988), versions of Categorial Grammar (Dowty, 1990) and references cited there, and some approaches within more traditional branches of generative grammar (Blevins, 1990).

The linguistic motivation for such alternative conceptions of string combination are so-called discontinuous constituency constructions. Furthermore, in formalisms that provide more powerful string operations it is much easier to define analyses in which the semantics is built compositionally. This in turn may simplify generation algorithms.

In the next section I describe the proposals by Pollard (1984); Johnson (1985) and Reape (1989). Based on work by Vijay-Shanker *et al.* (1987) I introduce the notion of Linear Context-free Rewriting systems, augmented with Feature structures (F-LCFRS). In F-LCFRS we abstract away from the actual construction of strings; we only require that these operations are *linear* and *non-erasing*. The three approaches mentioned previously are examples of F-LCFRS.

Most 'standard' parsing algorithms for constraint-based grammars (Matsumoto *et al.*, 1983; Pereira and Shieber, 1987; Shieber, 1989; Haas, 1989; Gerdemann, 1991) are not applicable in general for F-LCFRS because in these algorithms the assumption that phrases are constructed by concatenation is 'built-in'. I describe a head-driven parsing algorithm, based on the head-driven parser by Martin Kay (Kay, 1989). The parser is generalized in order to be applicable to any F-LCFR grammar. The disadvantages Kay noted for his parser do not carry over to this generalized version, as redundant search paths for CF-based grammars turn out to be genuine parts of the search space for F-LCFR grammars.

The algorithm is closely related to head-driven generators (van Noord, 1989; Calder *et al.*, 1989; Shieber *et al.*, 1989; van Noord, 1990; Shieber *et al.*, 1990). The algorithm proceeds in a bottom-up, head-driven fashion, which provides for bottom-up and top-down filtering in a simple and straightforward way. In modern linguistic theories very much information is defined in lexical entries, whereas rules are reduced to very general (and very uninformative) schemata. More information usually implies less search space, hence it is sensible to parse bottom-up in order to obtain useful information as soon as possible. Furthermore, in many linguistic theories a special daughter called the head determines what kind of other daughters there may be. Therefore, it is also sensible to start with the head in order to know what else you have to look for next. As the parser proceeds from head to head it is furthermore possible to use powerful top-down predictions based on the usual head feature percolations.

This chapter is organized as follows. Firstly I discuss the proposals for more powerful string operations, as presented by Pollard (1984), Johnson (1985) and Reape (1990). Then I define two restrictions on possible string combinations for constraint-based grammars, based on Vijay-Shanker *et al.* (1987). As an example I define a simple grammar for Dutch in which strings are combined by a technique quite similar to Pollard's head wrapping. The main part of the chapter is section 4, in which a parsing strategy for F-LCFR grammars is proposed. Some of its properties and possible modifications are discussed in the final section.

2 Beyond concatenation

2.1 Concatenative systems

In formalisms such as PATR II the string associated with a derivation is the sequence of terminal nodes of the corresponding derivation tree in left-to-right order. For example, the sentence

(1) Kim is easy to please

may be analyzed in some PATR grammar in a way that gives rise to the derivation tree in figure 1. Hence, the string associated with the derivation

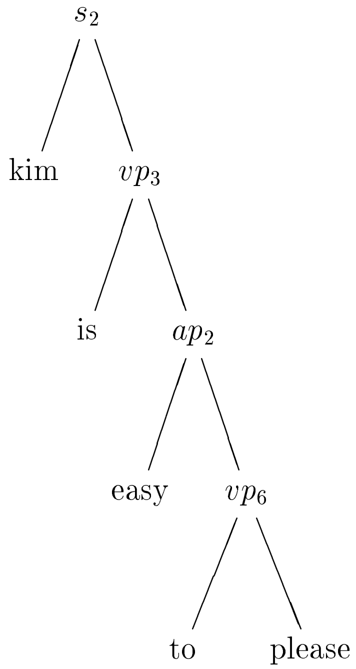

Figure 1: PATR derivation tree

is the sequence 'kim is easy to please'. Note though that in PATR this string is not (necessarily) part of the feature structures.

In sign-based approaches such as in UCG and HPSG the string does not have any special status but is part of an attribute of each feature structure (sign). The attribute is usually called 'phon', 'string', 'graph' or 'orth' (I will use 'string' in the following). Hence, the string associated with a construction is simply the value of the 'string' feature of the sign that is assigned to the construction. In UCG there is a condition called 'adjacency' condition, which says that signs can combine only if they are adjacent. In other words, the value of the 'string' feature of a mother node in a parse tree is always the concatenation of the 'string' features of the daughter nodes. Hence, the UCG parse tree for the foregoing example presumably would be someting like figure 2.

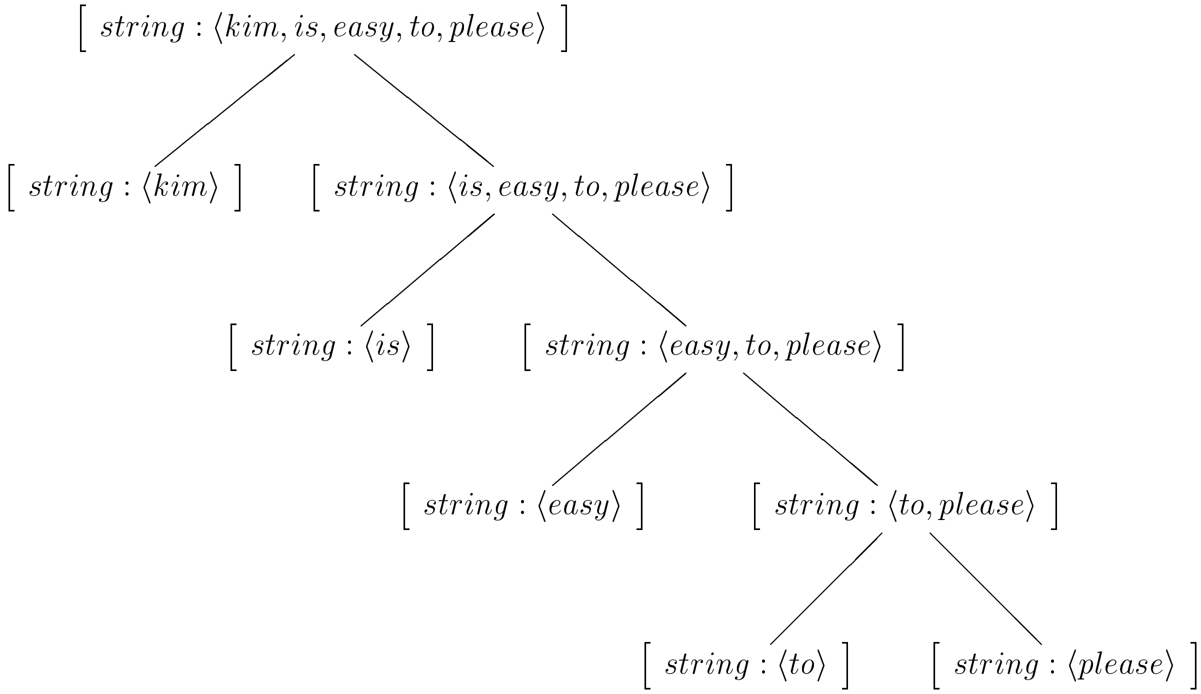

Figure 2: UCG parse tree

The two approaches are formally equivalent, but the second approach has the advantage that it at least becomes easier to think of other 'modes' of combination of the value of the 'string' attribute. As an example consider

(2) Kim is an easy person to please

Suppose that there is linguistic motivation that in this sentence, as in sentence (1), the sequence 'easy to please' should be regarded as a (discontinuous) constituent. Such an analysis can not be defined directly in PATR or UCG. If no adjancency condition applied we could have a parse tree of 'easy person to please' as in figure 3. In the next three subsections I describe some proposals which allow such a direct implementation of discontinuous constituents.

Clearly it is possible to define such an analysis in PATR or UCG in an indirect way by the usual threading of information through intermediate nodes (remember that both systems are Turing equivalent). However, these threading techniques usually become rather complex and make it difficult to define a semantics which is compositional. Furthermore, it is easier to define generation algorithms if the semantics is built in a systematically constrained

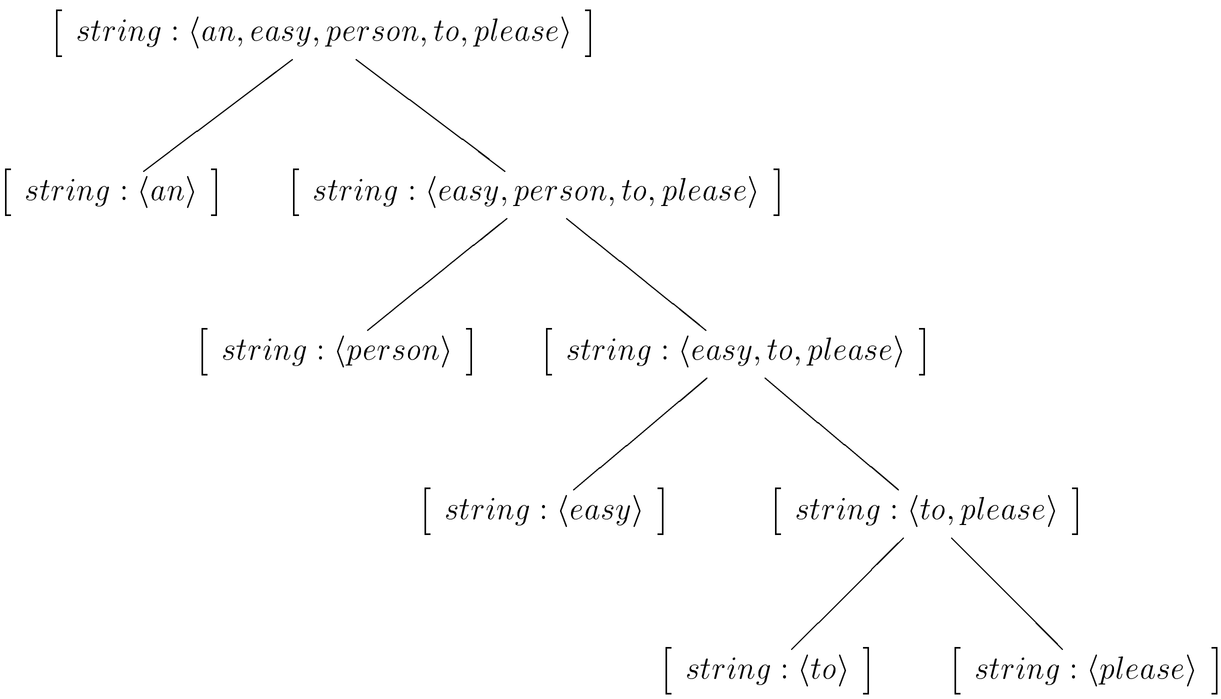

Figure 3: Discontinuous Constituency

way. The semantic-head-driven generation strategy discussed in the previous chapter faces problems in case semantic heads are 'displaced', and this displacement is analyzed using threading. However, in this chapter I sketch a simple analysis of verb-second (an example of a displacement of semantic heads) by an operation similar to head wrapping which a head-driven generator processes without any problems (or extensions) at all.

2.2 More powerful string operations

2.2.1 Head wrapping

Pollard (1984) proposes a grammatical formalism called Head Grammar (HG). HG is a slightly more powerful formalism than context-free grammar. The extra power is available through *head wrapping* operations. A head wrapping operation manipulates strings which contain a distinguished element (its *head*). Such *headed strings* are a pair of an ordinary string, and an index (the pointer to the head), for example $\langle w_1w_2w_3w_4, 3\rangle$ is the string $w_1w_2w_3w_4$ whose head is w_3. Ordinary grammar rules define operations on such strings. Such an operation takes n headed strings as its arguments and returns a headed string. A simple example is the operation which takes two headed strings and concatenates the first one to the left of the second one, and where the head of the second one is the head of the result (this shows that the operations subsume ordinary concatenation). The rule is labelled LC2 by Pollard:

$$LC2(\langle \sigma, j \rangle, \langle \tau, i \rangle) := \langle \sigma\tau, i \rangle$$

The following example shows that head-wrapping operations are in general more powerful than concatenation. In this example the second argument is 'wrapped' around the first argument:

$$RL2(\langle \sigma, j \rangle, \langle t_1 \dots t_n, i \rangle) := \langle t_1 \dots t_i \sigma t_{i+1} \dots t_n, i \rangle$$

As an example, Pollard presents a rule for English auxiliary inversion:

$$S[+INV] \rightarrow RL2(NP, VP[+AUX])$$

which may combine the noun phrase 'Kim' and the verb phrase 'must go' to yield 'Must Kim go', with head 'must'.

The motivation Pollard presents for extending context-free grammars (in fact, GPSG), is of a linguistic nature. Especially so-called discontinuous constituencies can be handled by HG whereas they constitute typical puzzles for GPSG. Apart from the above mentioned subject-auxiliary inversion he discusses the analysis of 'transitive verb phrases' based on Bach (1979). The idea is that in sentences such as

(3) Sandy persuaded Kim to leave

'persuaded' + 'to leave' form a (VP) constituent, which then combines with the NP object ('Kim') by a wrapping operation.

Yet another example of the use of head-wrapping in English are the analyses of the following sentences.

(4) a. Kim is much taller than Sandy
b. Kim is a much taller person than Sandy

(5) a. Kim is very easy to please
b. Kim is a very easy person to please

where in the first two cases 'taller than Sandy' is a constituent, and in the latter examples 'easy to please' is a constituent.

Breton and Irish are VSO languages, for which it has been claimed that the V and the O form a constituent. Such an analysis is readily available using head wrapping, thus providing a non-transformational account of McCloskey (1983).

Finally, Pollard also provides a wrapping analysis of Dutch cross-serial dependencies.

2.2.2 Johnson's 'combines'

Johnson (1985) discusses an extenstion of DCG in order to analyse the Australian free word-order language 'Guugu Yimidhirr'. In ordinary DCG a category is associated with a pair indicating which location the constituent occupies. Johnson proposes that constituents in the extended version of DCG be associated with a set of such pairs. A constituent thus 'occupies' a set of continuous locations. The following is a sentence of Guugu Yimidhirr:

(6) Yarraga-aga-mu-n gudaa dunda-y biiba-ngun
boy-GEN-mu-ERG do+ABS hit-PAST father-ERG
The boy's father hit the dog

In this sentence, the discontinuous constituent 'Yarraga-aga-mu-n ... biiba-ngun' (boy's father) is associated with the set of locations:

$$\{[0,1],[3,4]\}$$

Johnson notices that such expressions can be represented with bit vectors. In a grammar rule the sets of locations of the daughters of the rule are 'combined' to construct the set of locations associated with the daughters. The predicate $combines(s_1, s_2, s)$ is true iff s is equal to the (bit-wise) union of s_1 and s_2, and the (bit-wise) intersection of s_1 and s_2 is null (ie. s_1 and s_2 must be non-overlapping locations). Grammars which exclusively use this predicate, are permutation-closed. For Guugu Yimidhirr Johnson also proposes a concatenative rule for possesive noun constructions in which the possesive is identified by position rather than by inflectional markings. Apart from these constructions Guugu Yimidhirr is said to be permutation-closed (Johnson, quoting Haviland (1979)).

Earley deduction (Pereira and Warren, 1983) is used as a general proof procedure for such extended DCG grammars.

2.2.3 Sequence union

Reape (1989), and Reape (1990) discuss an operation called *sequence union* to analyze discontinuous constituents. The sequence union of two sequences s_1 and s_2 is the sequence s_3 iff each of the elements in s_1 and s_2 occur in s_3, and moreover, the originial order of the elements in s_1 and s_2 is preserved. For example, the sequence union of the sequences $\langle a,b\rangle$ and $\langle c,d\rangle$ is any of the sequences:

$$\langle a,b,c,d\rangle$$
$$\langle a,c,b,d\rangle$$
$$\langle a,c,d,b\rangle$$
$$\langle c,d,a,b\rangle$$

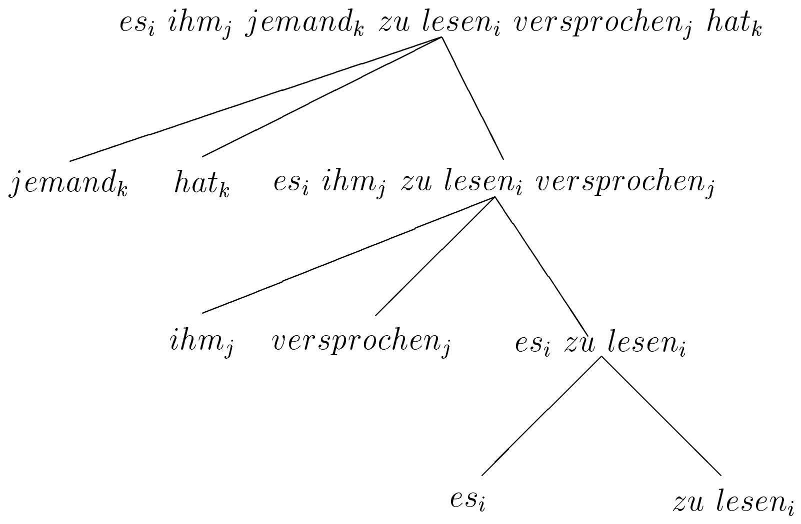

Figure 4: Parse tree of sequence union example

$\langle c, a, d, b\rangle$
$\langle c, a, b, d\rangle$

Reape presents an HPSG-style grammar (Pollard and Sag, 1987) for German and Dutch which uses the sequence union relation on *word-order domains*. The grammar handles several word-order phenomena in German and Dutch. Word-order domains are sequences of *signs*. The phonology of a sign is the concatenation of the phonology of the elements of its word-order domain. In 'rules', the word-order domain of the mother sign is defined in terms of the word-order domains of its daughter signs. For example, in the ordinary case the word-order domain of the mother simply consists of its daughter signs. However, in specific cases it is also possible that the word-order domain of the mother is the sequence union of the word-order domains of its daughters.

The following German example by Reape clarifies the approach, where I use indices to indicate to which verb an object belongs.

(7) ... es$_i$ ihm$_j$ jemand$_k$ zu lesen$_i$ versprochen$_j$ hat$_k$
... it him someone to read promised had
(i.e. someome had promised him to read it)

Figure 4 shows a parse tree of this sentences where the nodes of the derivation tree are labelled by the string associated with that node. Note that strings are defined *with respect to* word-order domains. Sequence union is defined on such domains. The strings of the derivation tree are thus only indirectly related through the corresponding word-order domains. Linear precedence statements are defined with respect to word-order domains. These statements can be thought of either as well-formedness conditions on totally ordered sequences, or alternatively as constraints further limiting possible orders of a word-order domain. Note that order information is monotonic; the sequence union relation can not 'change' the order of two ordered items.

2.3 F-LCFRS

I will restrict the attention to a class of constraint-based formalisms, in which operations on strings are defined that are more powerful than concatenation, but which operations are restricted to be *nonerasing*, and *linear*. The resulting class of systems can be characterized as Linear Context-Free Rewriting Systems (LCFRS), augmented with feature-structures (F-LCFRS). For a discussion of the properties of LCFRS without feature-structures, see Vijay-Shanker *et al.* (1987) and Weir (1988). Note though that these properties do not carry over to the current system, because of the augmention with feature structures. The proposals discussed in the previous section can be seen as examples of F-LCFRS.

As in LCFRS, the operations on strings in F-LCFRS can be characterized as follows. First, derived structures will be mapped onto a string of words; i.e. each derived structure 'knows' which string it 'dominates'. For example, each derived feature structure may contain an attribute 'string' whose value is a list of atoms representing the string it dominates. Furthermore, I will identify this string with the set of occurrences of words (or bag of words) in the string. I will write $w(F)$ for the set of occurrences of words that the derived structure F dominates. Rules combine structures $D_1 \ldots D_n$ into a new structure M. Nonerasure requires that the union of w applied to each daughter is a subset of $w(M)$:

$$\bigcup_{i=1}^{n} w(D_i) \subseteq w(M)$$

Linearity requires that the difference of the cardinalities of these sets is a constant factor; i.e. a rule may only introduce a fixed number of words syncategorematically:

$$|w(M)| - |\bigcup_{i=1}^{n} w(D_i)| = c, c \text{ } a \text{ } constant$$

CF-based formalisms clearly fulfill this requirement, as do Head Grammars, grammars using sequence union, Johnson's Australian grammar, and TAG's. Unlike in the definition of LCFRS I do not require that these operations are operations in the strict sense, i.e. the operations do not have to be functional. Note that sequence union is relational; the others are functional. For a discussion of the consequences of this difference, refer to

Reape (1991). I assume here furthermore that $\bigcup_{i=1}^{n} w(D_i) = w(M)$, for all rules other than lexical entries (i.e. all words are introduced on a terminal). Note though that a simple generalization of the algorithm presented below handles the general case (along the lines of Shieber *et al.* (1989); Shieber *et al.* (1990) by treating rules that introduce extra lexical material as non-chain-rules).

Furthermore, I will assume that each rule has a designated daughter, called the head. Although I will not impose any restrictions on the head, it will turn out that the parsing strategy to be proposed will be very sensitive to the choice of heads, with the effect that F-LCFRS's in which the notion 'head' is defined in a systematic way (Pollard's Head Grammars, Reape's version of HPSG, Dowty's version of Categorial Grammar), may be much more efficiently parsed than other grammars. The notion *lexical head* of a parse tree is defined recursively in terms of the head. The lexical head of a tree will be the lexical head of its head. The lexical head of a terminal will be that terminal itself. I will write the head of a definite clause as the leftmost daughter of that clause.

A grammar will be a set of definite clauses over a constraint language, following Höhfeld and Smolka (1988). The constraint language consists of path equations in the manner of PATR. Hence, instead of first-order terms we are using feature structures. The relation defined by the grammar will be the relation 'sign'. I assume furthermore that no other recursive relations are being defined. Hence a grammar rule will be something like

$$sign(M)\texttt{:-}sign(D_1)\ldots sign(D_n), \phi.$$

where ϕ are constraints on the variables. However, each nonunit rule may be associated with one extra relation-call. This extra relation is thought of as defining how the string of the mother node is built from the strings of the daughters. For example, to implement a simple grammar rule using concatenation, we may write

(8) $sign(M)\texttt{:-}$
$\quad sign(D_1),$
$\quad sign(D_2),$
$\quad concatenate\_strings(D_1, D_2, M)$
$\quad \phi.$

where the relation 'concatenate_strings' states that the string of the mother is the concatenation of the strings of the daughters. The extra relation should be defined in such a way that given the two instantiated daughter signs, the extra relation is guaranteed to terminate (assuming Prolog like procedural semantics).

Furthermore, each grammar should provide a definition of the predicates *head/2* and *string/2*. The first one defines the relation between the mother sign of a rule and its head; the second one defines for a sign the string (as a list of atoms) associated with that sign.

In the meta-interpreter I use for a nonunit clause

$$sign(Mother)\,\texttt{:-}\,sign(Head), sign(D_1) \ldots sign(D_n), Call, \phi.$$

the predicate *rule/4*:

$$rule(Head, Mother, \langle D_1 \ldots D_n \rangle, Call)\,\texttt{:-}\,\phi.$$

Unit clauses are represented as

$$lex(M)\,\texttt{:-}\,\phi.$$

I assume the meta interpreter has the predicate 'call/1' available which may be used as in Prolog.

3 A sample grammar

In this section I present a simple F-LCFR grammar for a (tiny) fragment of Dutch. As a caveat I want to stress that the purpose of the current section is to provide an *example* of possible input for the parser to be defined in the next section, rather than to provide an account that is completely satisfactory from a linguistic point of view.

There is only one parameterized, binary branching, rule in the grammar:

$$(9)\quad sign(\begin{bmatrix} syn : Syn \\ sc : Tail \\ sem : Sem) \end{bmatrix}_M \;\texttt{:-}$$
$$sign(\begin{bmatrix} syn : Syn \\ sc : \langle Arg|Tail\rangle \\ sem : Sem \end{bmatrix}_H),$$
$$sign(Arg),$$
$$cb(M, H, Arg).$$

In this grammar rule, heads select arguments using a subcat list. Argument structures are specified lexically and are percolated from head to head. Syntactic features are shared between heads (hence I make the simplifying assumption that head = functor, which may have to be revised in order to treat modification). The relation 'cb' defines how the string of the mother is constructed from its daughters. In the grammar I use revised versions of Pollard's head wrapping operations to analyse *cross serial dependency* and *verb second* constructions. For a linguistic background of these constructions and analyses, cf. Evers (1975), Koster (1975) and many others. The value of the attribute *phon* consists of three parts, to implement the idea of Pollard's 'headed strings'. The parts $left$ and $right$ represent the strings left and right of the head. The part *head* represent the head string. Hence, the string associated with such a term is the concatenation of the three arguments from left to right. The predicate *cb* is defined as follows:

$$(10)\quad cb(\begin{bmatrix} phon : Mphon \\ string : String \end{bmatrix}, \begin{bmatrix} phon : Hphon \end{bmatrix}, \begin{bmatrix} phon : Aphon \\ rule : Rule \end{bmatrix}) :\text{-}$$
$$wrap(Rule, Hphon, Aphon, Mphon),$$
$$phon\_string(Mphon, String).$$

Here, the values of the attribute *phon* associated with the two daughters of the rule are to be combined by the *wrap* predicate. Several versions of this predicate will be defined below. The value of 'string' of the mother node is defined with respect to its 'phon' value by the predicate *phon_string*. This predicate is defined in terms of the predicate *append*/3. As an abbreviation I write $A \cdot B$ for C such that $append(A, B, C)$. The definitions of both predicates follow:

$$(11)\quad phon\_string(\begin{bmatrix} left : L \\ head : H \\ right : R \end{bmatrix}, L \cdot H \cdot R).$$

$$append(\langle\rangle, X, X).$$
$$append(\langle H|T\rangle, L_1, \langle H|L\rangle) :\text{-}$$
$$append(T, L_1, L)$$

There are a few versions of the predicate *wrap* to illustrate the idea that different string operations can be defined. Each version of the predicate will be associated with an atomic identifier to allow lexical entries to subcategorize for their arguments under the condition that a specific version of this predicate be used. The purpose of this feature is similar to the 'order' feature found in UCG (Zeevat *et al.*, 1987). For example, a verb may select an object to its left, and an infinite verb phrase which has to be raised. For simple (left or right) concatenation the predicate is defined as follows:

$$(12)\quad wrap(left, \begin{bmatrix} left : L \\ head : H \\ right : R \end{bmatrix}, \begin{bmatrix} left : ArgL \\ head : ArgH \\ right : ArgR \end{bmatrix}, \begin{bmatrix} left : ArgL \cdot ArgH \cdot ArgR \cdot L \\ head : H \\ right : R \end{bmatrix}).$$

$$wrap(right, \begin{bmatrix} left : L \\ head : H \\ right : R \end{bmatrix}, \begin{bmatrix} left : ArgL \\ head : ArgH \\ right : ArgR \end{bmatrix}, \begin{bmatrix} left : L \\ head : H \\ right : R \cdot ArgL \cdot ArgH \cdot ArgR \end{bmatrix}).$$

In the first case the string associated with the argument is appended to the left of the string left of the head; in the second case this string is appended to the right of the string right of the head. Lexical entries for intransitive verbs such as 'ontwaakt' (wakes up) are defined as follows:

$$(13)\quad \begin{bmatrix} syn : v \\ sc : \langle \begin{bmatrix} syn : n \\ sc : \langle\rangle \\ sem : Subj \\ rule : left \end{bmatrix} \rangle \\ phon : \begin{bmatrix} left : \langle\rangle \\ head : \langle ontwaakt \rangle \\ right : \langle\rangle \end{bmatrix} \\ sem : ontwaakt(Subj) \\ string : \langle ontwaakt \rangle \end{bmatrix}$$

I assume that lexical entries also specify that their *string*-value is functionally dependent of the *phon* value. Furthermore, the values of the $left$ and *right* attributes of *phon* are the empty list. Henceforth, I will not specify the value of *string* explicitly, but assume that each lexical entry extends

$$\begin{bmatrix} phon : \begin{bmatrix} left : \langle\rangle \\ head : Head \\ right : \langle\rangle \end{bmatrix} \\ string : Head \end{bmatrix}.$$

Hence, bitransitive verbs such as 'vertelt' (tells) are abbreviated as follows:

$$(14)\quad \begin{bmatrix} syn : v \\ sc : \langle \begin{bmatrix} syn : n \\ sc : \langle\rangle \\ sem : Obj \\ rule : left \end{bmatrix}, \begin{bmatrix} syn : n \\ sc : \langle\rangle \\ sem : Iobj \\ rule : left \end{bmatrix}, \begin{bmatrix} syn : n \\ sc : \langle\rangle \\ sem : Subj \\ rule : left \end{bmatrix} \rangle \\ phon : \begin{bmatrix} head : \langle vertelt \rangle \end{bmatrix} \\ sem : vertelt(Subj, Iobj, Obj) \end{bmatrix}$$

A different version of this lexical entry selects an SBAR to the right:

$$(15)\quad \begin{bmatrix} syn : v \\ sc : \langle \begin{bmatrix} syn : comp \\ sc : \langle\rangle \\ sem : Obj \\ rule : right \end{bmatrix}, \begin{bmatrix} syn : n \\ sc : \langle\rangle \\ sem : Iobj \\ rule : left \end{bmatrix}, \begin{bmatrix} syn : n \\ sc : \langle\rangle \\ sem : Subj \\ rule : left \end{bmatrix} \rangle \\ phon : \begin{bmatrix} head : \langle vertelt \rangle \end{bmatrix} \\ sem : vertelt(Subj, Iobj, Obj) \end{bmatrix}$$

Proper nouns such as 'Arie' are simply defined as:

$$(16) \quad \left[\begin{array}{l} syn : n \\ sc : \langle\rangle \\ phon : \left[\ head : \langle arie \rangle\ \right] \\ sem : arie \end{array}\right]$$

For the sake of the example I assume several other NP's to have such a definition.

The choice of datastructure for the value of the attribute *phon* allows a simple definition of the verb raising (*vr*) version of the *wrap* predicate that may be used for Dutch cross serial dependencies:

$$(17) \quad wrap(vr, \left[\begin{array}{l} left : \langle\rangle \\ head : H \\ right : \langle\rangle \end{array}\right], \left[\begin{array}{l} left : ArgL \\ head : ArgH \\ right : ArgR \end{array}\right], \left[\begin{array}{l} left : ArgL \\ head : H \\ right : ArgH \cdot ArgR \end{array}\right]).$$

Here the head and right string of the argument are appended to the right, whereas the left string of the argument is appended to the left. A raising verb, eg. 'hoort' (hears) is defined as:

$$(18) \quad \left[\begin{array}{l} syn : v \\ sc : \langle \left[\begin{array}{l} syn : inf \\ sc : \langle \left[\ sem : InfSubj\ \right] \rangle \\ sem : Obj \\ rule : vr \end{array}\right], \left[\begin{array}{l} syn : n \\ sc : \langle\rangle \\ sem : InfSubj \\ rule : left \end{array}\right], \left[\begin{array}{l} syn : n \\ sc : \langle\rangle \\ sem : Subj \\ rule : left \end{array}\right] \rangle \\ phon : \left[\ head : \langle hoort \rangle\ \right] \\ sem : hoort(Subj, Obj) \end{array}\right]$$

In this entry 'hoort' selects — apart from its NP-subject — two objects, a NP and a VP (with category INF). The INF still has an element in its subcat list; this element is controlled by the NP (this is performed by the sharing of *InfObj*). To derive the subordinate phrase

(19) dat Jan Arie Bob leugens hoort vertellen
(that Jan hears that Arie tells lies to Bob)

the main verb 'hoort' first selects the infinitival 'bob leugens vertellen'. These two strings are combined into 'bob leugens hoort vertellen' (using the *vr* version of the *wrap* predicate). After the selection of the object, resulting in 'arie bob leugens hoort vertellen', the subject is selected resulting in the string 'jan arie bob leugens hoort vertellen'. This string is selected by the complementizer, resulting in 'dat jan arie bob leugens hoort vertellen'. The argument structure will be instantiated as *dat(hoort(jan,vertelt(arie,bob,leugens)))*.

In Dutch main clauses, there usually is no overt complementizer; instead the finite verb occupies the first position (in yes-no questions), or the second position (right after the topic; ordinary declarative sentences). In the following analysis an empty complementizer selects an ordinary (finite) v; the resulting string is formed by the following definition of *wrap*:

$$(20)\quad wrap(v2, \begin{bmatrix} left : \langle\rangle \\ head : \langle\rangle \\ right : \langle\rangle \end{bmatrix}, \begin{bmatrix} left : L \\ head : H \\ right : R \end{bmatrix}, \begin{bmatrix} left : \langle\rangle \\ head : H \\ right : L \cdot R \end{bmatrix}).$$

The 'empty' finite complementizer is defined as:

$$(21)\quad \begin{bmatrix} syn : comp \\ sc : \langle \begin{bmatrix} syn : v \\ sc : \langle\rangle \\ sem : Obj \\ rule : v2 \end{bmatrix} \rangle \\ phon : \begin{bmatrix} head : \langle\rangle \end{bmatrix} \\ sem : dat(Obj) \end{bmatrix}$$

whereas an ordinary complementizer, eg. 'dat' (that) is defined as:

$$(22)\quad \begin{bmatrix} syn : comp \\ sc : \langle \begin{bmatrix} syn : v \\ sc : \langle\rangle \\ sem : Obj \\ rule : right \end{bmatrix} \rangle \\ phon : \begin{bmatrix} head : \langle dat \rangle \end{bmatrix} \\ sem : dat(Obj) \end{bmatrix}$$

Note that this analysis captures the special relationship between complementizers and (fronted) finite verbs in Dutch. The sentence

(23) Hoort Arie Jan Bob vertellen dat Claire ontwaakt?
(hears Arie Jan Bob tell that Claire wakes up?
i.e. Does Arie hear that Jan tells Bob that Claire wakes up?)

is derived as in figure 5 (where the head of a string is represented in capitals).

What remains to be done is to define the two grammar specific predicates *head/2* and *string/2*. These are simply defined as follows:

$$(24)\quad head(\begin{bmatrix} syn : Syn \\ sem : Sem \end{bmatrix}, \begin{bmatrix} syn : Syn \\ sem : Sem \end{bmatrix}).$$

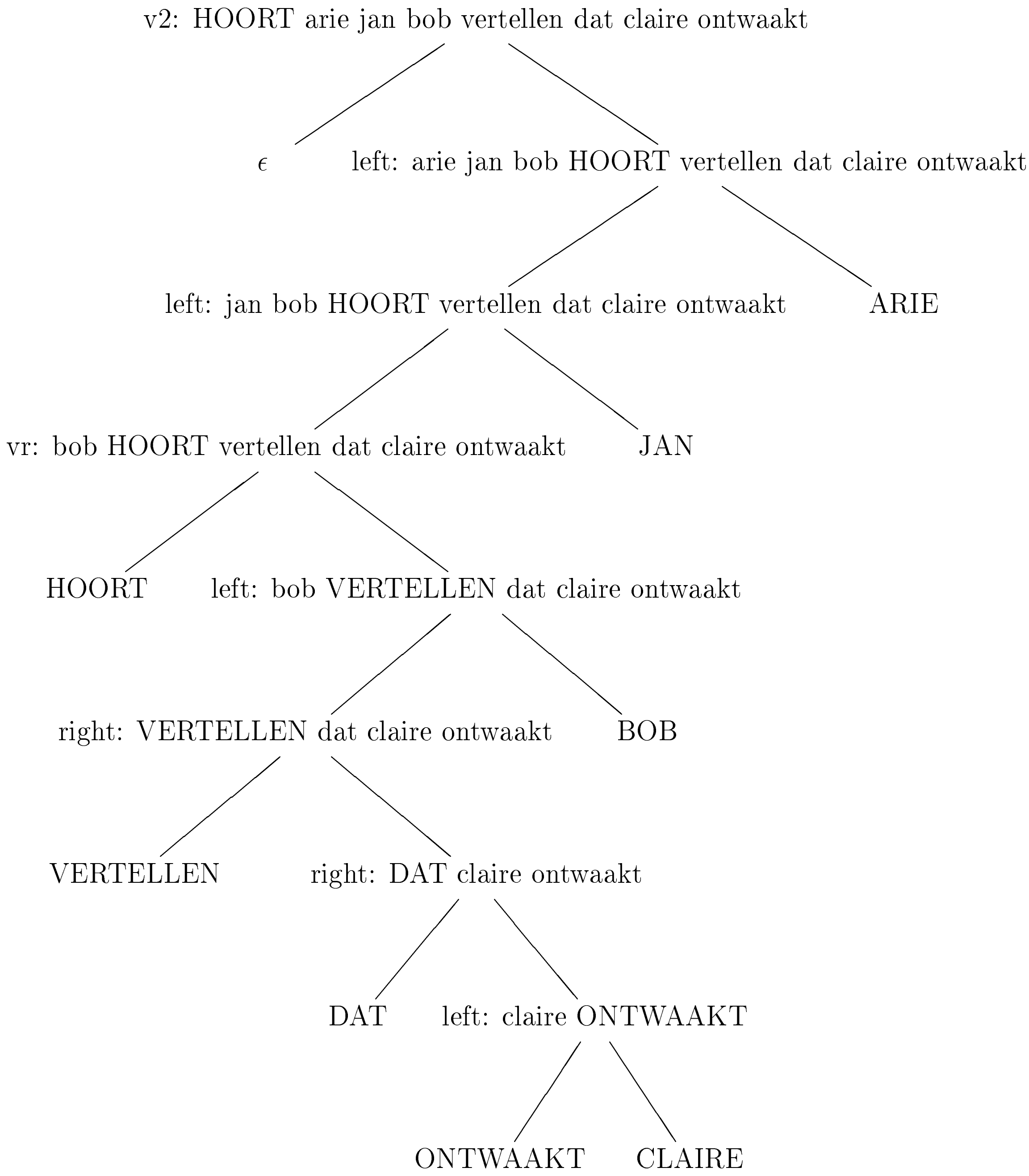

Figure 5: Deriving 'Hoort Arie Jan Bob vertellen dat Claire ontwaakt'

$$string(\left[\ string : String\ \right], String).$$

4 The head corner parser

This section describes the head-driven parsing algorithm for the type of grammars described above. The parser is a generalization of Kay's head-driven parser, which in turn is a modification of a left-corner parser. The parser, which may be called a 'head-corner' parser,[1] proceeds in a bottom-up way. Because the parser proceeds from head to head it is easy to use powerful top-down predictions based on the usual head feature percolations, and subcategorization requirements that heads require from their arguments.
In fact, the motivation for this approach to parsing discontinuous constituency is already hinted at by Mark Johnson (Johnson, 1985):

> My own feeling is that the approach that would bring the most immediate results would be to adopt some of the "head driven" aspects of Pollard's (1984) Head Grammars. In his conception, heads contain as lexical information a list of the items they subcategorize for. This strongly suggests that one should parse according to a "head-first" strategy: when one parses a sentence, one looks for its verb first, and then, based on the lexcial form of the verb, one looks for the other arguments in the clause. Not only would such an approach be easy to implement in a DCG framework, but given the empirical fact that the nature of argument NP's in a clause is strongly determined by that clause's verb, it seems a very reasonable thing to do.

It is clear from the context that Johnson believes this strategy especially useful for non-configurational languages.

In left-corner parsers (Matsumoto *et al.*, 1983) the first step of the algorithm is to select the left-most word of a phrase. The parser then proceeds by proving that this word indeed can be the left-corner of the phrase. It does so by selecting a rule whose leftmost daughter unifies with the category of the word. It then parses other daughters of the rule recursively and then continues by connecting the mother category of that rule upwards, recursively. The left-corner algorithm can be generalized to the class of grammars under consideration if we start with the *lexical head* of a phrase, instead of its leftmost word. Furthermore the *head_corner* predicate then connects smaller categories upwards by unifying them with the *head* of a rule. The first step of the algorithm consists of the prediction step: which lexical entry is the lexical head of the phrase? The first thing to note is that the words

[1] This name is due to Pete Whitelock.

introduced by this lexical entry should be part of the input string, because of the nonerasure requirement. Therefore we use the bag of words as a 'guide' (Dymetman *et al.*, 1990) as in a left-corner parser, but we change the way in which lexical entries 'consume the guide'. For the moment I assume that the bag of words is simply represented with the string, but I introduce a better datastructure for bags in the next section. Hence to get things going I define the predicate *start_parse/1* as follows:

(25) $start\_parse(Sign)$:-
$\quad string(Sign, String)$,
$\quad parse(Sign, String, \langle\rangle)$.

Furthermore in most linguistic theories it is assumed that certain features are shared between the mother and the head. I assume that the predicate *head/2* defines these feature percolations; for the grammar of the foregoing section this predicate was defined as:

(26) $head(\begin{bmatrix} syn : Syn \\ sem : Sem \end{bmatrix}, \begin{bmatrix} syn : Syn \\ sem : Sem \end{bmatrix}).$

As we will proceed from head to head these features will also be shared between the lexical head and the top-goal; hence we can use this definition to restrict lexical lookup by top-down prediction. [2] The first step in the algorithm is defined as:

(27) $parse(Goal, P_0, P)$:-
$\quad predict\_head(Goal, Lex, P_0, P_1)$,
$\quad head\_corner(Lex, Goal, P_1, P)$.

$predict\_head(Goal, Lex, P_0, P)$:-
$\quad head(Goal, Lex)$,
$\quad lex(Lex)$,
$\quad string(Lex, Words)$,
$\quad subset(Words, P_0, P)$.

Instead of taking the first word from the current input string, the parser may select a lexical entry dominating a subset of the words occuring in the input string, provided this lexical entry can be the *lexical head* of the current goal. The predicate $subset(L_1, L_2, L_3)$ is true in case L_1 is a subset of L_2 with complement L_3. Later we will improve on the indexing of lexical entries.

The second step of the algorithm, the *head_corner* part, is identical to the *left_corner* part of the left-corner parser, but instead of selecting the

[2]In the general case we need to compute the transitive closure of (restrictions of (Shieber, 1985)) possible mother-head relationships.

leftmost daughter of a rule the head-corner parser selects the head of a rule. Remember that nonunit rules are represented as:

$$rule(Head, Mother, Ds, Call)$$

where $Head$ is the head of the rule, $Mother$ is the mother node, Ds is a list of daughter nodes and $Call$ is the call to the extra relation.

(28) $head\_corner(X, X, P, P).$
$head\_corner(Small, Big, P_0, P)$:-
  $rule(Small, Mid, Others, Call),$
  $parse\_list(Others, P_0, P_1),$
  $call(Call),$
  $head\_corner(Mid, Big, P_1, P).$

$parse\_list([], P, P).$
$parse\_list([H|T], P_0, P)$:-
  $parse(H, P_0, P_1),$
  $parse\_list(T, P_1, P).$

4.0.1 Example.

To parse the sentence 'dat jan ontwaakt', the head corner parser will proceed as follows. The first call to *parse* is:

$$\texttt{?-}\quad parse(\begin{bmatrix} syn : comp \\ sc : \langle\rangle \\ string : \langle dat, jan, ontwaakt\rangle \end{bmatrix}, \langle dat, jan, ontwaakt\rangle, \langle\rangle).$$

The prediction step selects the lexical entry 'dat'. The next goal is to show that this lexical entry is the lexical head of the top goal; furthermore the string that still has to be covered is now $\langle jan, ontwaakt\rangle$. Leaving details out the *head_corner* clause looks as :

$$\texttt{?-}\quad head\_corner(\begin{bmatrix} syn : comp \\ sc : \langle \begin{bmatrix} syn : v \\ sc : \langle\rangle \\ sem : Sem \\ rule : right \end{bmatrix} \rangle \\ phon : \begin{bmatrix} left : \langle\rangle \\ head : \langle dat\rangle \\ right : \langle\rangle \end{bmatrix} \\ string : \langle dat\rangle \\ sem : dat(Sem) \end{bmatrix},$$

$$\begin{bmatrix} syn : comp \\ sc : \langle\rangle \\ string : \langle dat, jan, ontwaakt\rangle \end{bmatrix},$$
$$\langle jan, ontwaakt\rangle, \langle\rangle).$$

The category of *dat* has to be matched with the head of a rule. Notice that *dat* subcategorizes for a v with rule feature *right*. Hence the *right* version of the *wrap* predicate applies, and the next goal is to parse the v for which this complementizer subcategorizes, with input 'jan, ontwaakt':

$$\texttt{?-}\quad parse(\begin{bmatrix} syn : v \\ sc : \langle\rangle \\ sem : Sem \\ rule : right \end{bmatrix}, \langle jan, ontwaakt\rangle, P).$$

Lexical lookup selects the word *ontwaakt* from this string. The word *ontwaakt* has to be shown to be the head of this v node, by the *head_corner* predicate:

$$\texttt{?-}\quad head\_corner(\begin{bmatrix} syn : v \\ sc : \langle \begin{bmatrix} syn : n \\ sc : \langle\rangle \\ sem : Subj \\ rule : left \end{bmatrix} \rangle \\ phon : \begin{bmatrix} left : \langle\rangle \\ head : \langle ontwaakt\rangle \\ right : \langle\rangle \end{bmatrix} \\ sem : ontwaakt(Subj) \\ string : \langle ontwaakt\rangle \end{bmatrix},$$
$$\begin{bmatrix} syn : v \\ sc : \langle\rangle \\ sem : Sem \\ rule : right \end{bmatrix}, \langle jan\rangle, P).$$

This time the *left* combination rule applies and the next goal consists in parsing a NP (for which *ontwaakt* subcategorizes) with input string *jan*. This goal succeeds with an empty output string. Hence the argument of the rule has been found successfully and hence we need to connect the mother of the rule up to the v node. This succeeds trivially, instantiating the *head_corner* call above to:

$$
\texttt{?-}\quad head\_corner(\begin{bmatrix} syn : v \\ sc : \langle \begin{bmatrix} syn : n \\ sc : \langle\rangle \\ sem : jan \\ rule : left \end{bmatrix} \rangle \\ phon : \begin{bmatrix} left : \langle\rangle \\ head : \langle ontwaakt \rangle \\ right : \langle\rangle \end{bmatrix} \\ sem : ontwaakt(jan) \\ string : \langle ontwaakt \rangle \end{bmatrix},
$$

$$
\begin{bmatrix} syn : v \\ sc : \langle\rangle \\ sem : ontwaakt(jan) \\ phon : \begin{bmatrix} left : \langle jan \rangle \\ head : \langle ontwaakt \rangle \\ right : \langle\rangle \end{bmatrix} \\ string : \langle jan, ontwaakt \rangle \\ rule : right \end{bmatrix}, \langle jan \rangle, \langle\rangle).
$$

and therefore we now have found the v for which *dat* subcategorizes, instantiating the parse goal above to:

$$
\texttt{?-}\quad parse(\begin{bmatrix} syn : v \\ sc : \langle\rangle \\ sem : ontwaakt(jan) \\ phon : \begin{bmatrix} left : \langle jan \rangle \\ head : \langle ontwaakt \rangle \\ right : \langle\rangle \end{bmatrix} \\ string : \langle jan, ontwaakt \rangle \\ rule : right \end{bmatrix}, \langle jan, ontwaak \rangle, \langle\rangle).
$$

Hence the next goal is to connect the complementizer with an empty subcat list up to the topgoal; again this succeeds trivially. Hence we obtain the instantiated version of the first *head_corner* call:

$$\text{?- } head\_corner(\begin{bmatrix} syn : comp \\ sc : \langle \begin{bmatrix} syn : v \\ sc : \langle\rangle \\ sem : ontwaakt(jan) \\ rule : right \end{bmatrix} \rangle \\ phon : \begin{bmatrix} left : \langle\rangle \\ head : \langle dat \rangle \\ right : \langle\rangle \end{bmatrix} \\ sem : dat(Sem_2) \end{bmatrix},$$

$$\begin{bmatrix} syn : comp \\ sc : \langle\rangle \\ sem : dat(ontwaakt(jan)) \end{bmatrix}, \langle jan, ontwaakt \rangle, \langle\rangle).$$

and the first call to parse will succeed, yielding:

$$\text{?- } parse(\begin{bmatrix} syn : comp \\ sc : \langle\rangle \\ sem : dat(ontwaakt(jan)) \\ phon : \begin{bmatrix} left : \langle\rangle \\ head : \langle dat \rangle \\ right : \langle jan, ontwaakt \rangle \end{bmatrix} \\ string : \langle dat, jan, ontwaakt \rangle \end{bmatrix}, \langle dat, jan, ontwaakt \rangle, \langle\rangle).$$

The flow of control of the parsing process for this example is shown in figure 6. In this picture, the integers associated with the arrows indicate the order of the steps of the parsing process. Predict steps are indicated with a framed integer, head_corner steps are indicated with a bold face integer, and recursive parse steps are indicated with a slanted integer. The nodes represent (some of) the information available at the moment this step is performed.

A slightly more complex example is shown in figure 7, for the sentence

(29) Hoort Arie Bob sla bestellen?

5 Discussion and Extensions

5.1 Sound and Complete.

The algorithm as it is defined is *sound* and *complete* in the usual depth-first, backtrack search sense. Clearly the parser may enter an infinite loop (in case non branching rules are defined that may feed themselves or in case a grammar makes a heavy use of empty categories). However, in case the parser *does* terminate one can be sure that it has found all solutions.

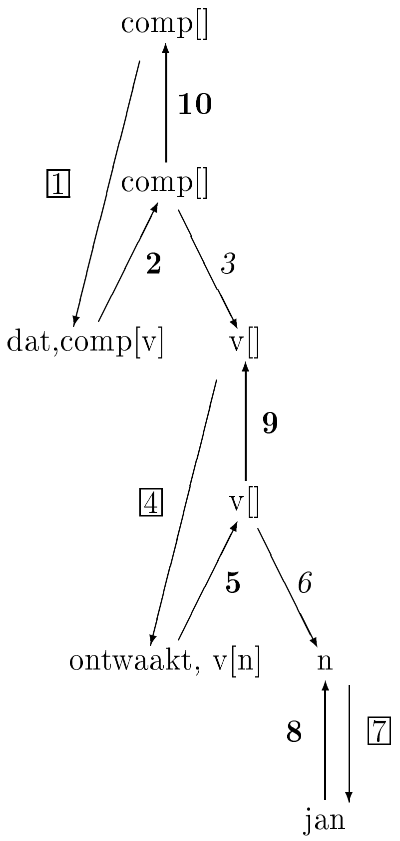

Figure 6: Parsing 'dat jan ontwaakt'

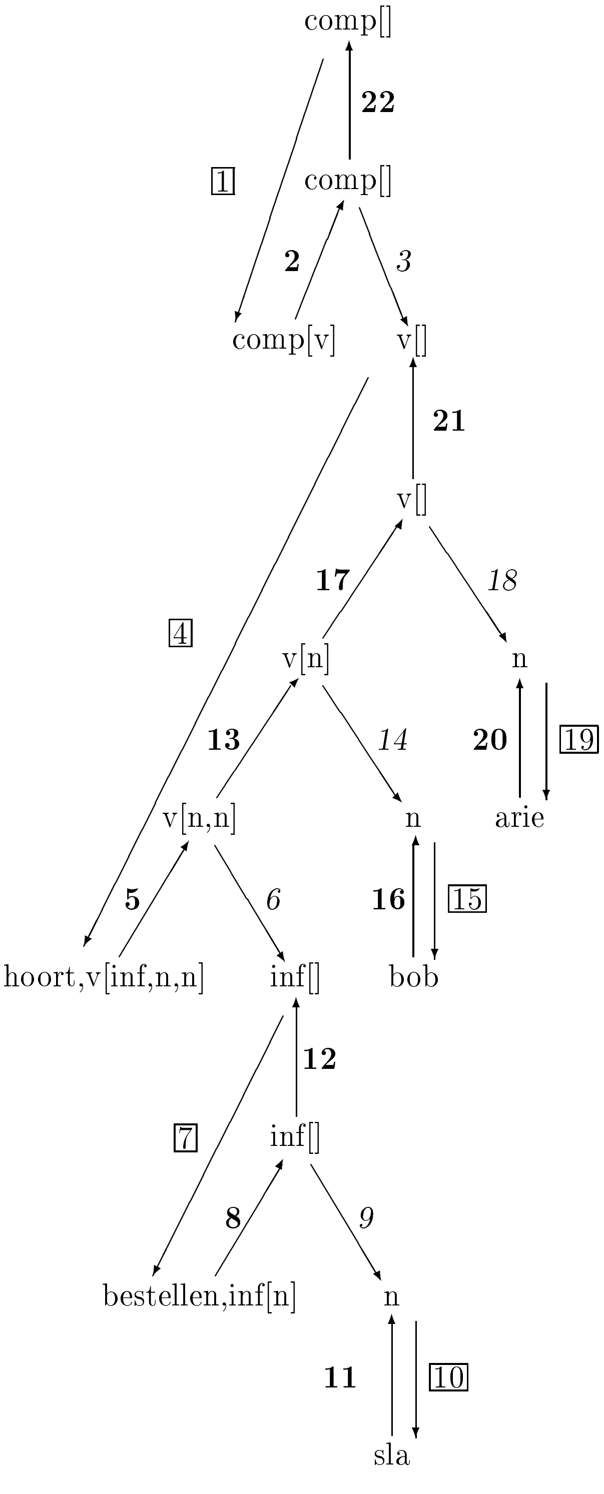

Figure 7: Parsing ‘Hoort Arie Bob sla bestellen’

5.2 Minimality.

A parser is called *minimal* iff it returns one solution for each possible derivation. As it stands the parser is not minimal in this sense. In fact, if the same word occurs more than once in the input string then we may find a solution several times. The problem comes about because a list is not an appropriate data structure for bags. For example, removing an element from a bag should not be non deterministic, but it is if we use lists (in Prolog this problem may be repaired using the cut). It is straightforward to encode bags with a more appropriate data structure such as the one found in some Prolog libraries, which we will adopt with slight modifications. [3]

An empty bag is represented by the constant 'bag'. A nonempty bag consists of three parts: an element, a (representation of a) number indicating how often this element occurs in the bag, and the rest of the bag. Furthermore the bag is ordered in such a way that an element precedes another element in the bag iff it precedes that element in some standard order. For example, in Prolog the bag $\{a, b, a, a, b, c\}$ is represented as:

```
bag(a,3,bag(b,2,bag(c,1,bag)))
```

For clarity I sometimes write such a bag as:

$$\langle a/3, b/2, c/1\rangle$$

and use the usual list notation.

The parser is modified as follows. The predicate which calls the parser will contain an extra predicate, *list_to_bag/2*, which encodes a list as a bag. Furthermore this bag is given as the input argument *to parse/3*; the output argument is the empty bag, the constant *bag*.

(30) $start\_parse(Sign)$:-
    $string(Sign, String)$,
    $list\_to\_bag(String, Bag)$,
    $parse(Sign, Bag, bag)$.

The 'subset' predicate is now defined for such bags, and for this reason it is called 'subbag'.

(31) $\%subbag(+SubBag, +TotalBag, ?ComplementBag)$.
$subbag(\langle\rangle, Bag, Bag)$.
$subbag(\langle El/0|R\rangle, Bag, Rest)$:-
    $subbag(R, Bag, Rest)$.
$subbag(\langle El/s(X)|R\rangle, \langle El/s(Y)|R2\rangle, Rest)$:-
    $subbag(\langle El/X|R\rangle, \langle El/Y|R2\rangle, Rest)$.

[3]For example in the file 'bags.pl' of the Quintus library, by Richard O'Keefe.

$$subbag(\langle El/s(I)|R\rangle, \langle X/Y|Bag\rangle, \langle X/Y|Rest\rangle) :\!-$$
$$\quad subbag(\langle El/s(I)|R\rangle, Bag, Rest).$$

This definition of the predicate 'subbag' is deterministic given the first two arguments, and given the fact that the elements of the bag are always constants. The modified version of the parser is minimal.

5.3 Efficiency.

The parser is quite efficient if the notion 'syntactic head' implies that much syntactic information is shared between the head of a phrase and its mother. In this case the prediction step in the algorithm will be much better at 'predicting' the head of the phrase. If on the other hand the notion 'head' does not imply such feature percolations, then the parser must predict the head randomly from the input string as no top-down information is available.

The efficiency of the parser can be improved by common logic programming and parsing techniques. Firstly, it is possible to compile the grammar rules, lexical entries and parser a bit further by (un)folding. Secondly it is possible to integrate well-formed and ill-formed subgoal tables in the parser, following the technique described by Matsumoto *et al.* (1983). The usefulness of this technique strongly depends on the actual grammars that are being used.

5.3.1 Indexing of lexical entries

It is possible to use a more clever indexing of lexical entries. Firstly, it is possible to extract the elements from the lexicon that can be used to parse some given sentence, before the parser starts properly. That is, for each sentence the parser first selects those lexical entries that possibly could be used in the analysis of that sentence. The parsing algorithm proper then only takes these lexical entries into account when it searches the lexicon in the prediction step.

Furthermore, we can get rid of the *subbag/3* predicate in the *predict* clause. Given the precompilation step mentioned above, we can use a slightly different representation of bags where the elements of the bag are not explicitly mentioned, but are implicitly represented by the position in the bag. To make this work we also allow bags where elements occur zero times. For example, given the sentence 'a b b c a a', the corresponding bag will simply be:

$$\langle 3, 2, 1\rangle$$

Furthermore, the bag that consists of two a's is given the representation

$$\langle 2, 0, 0\rangle$$

with respect to that sentence. The idea is that each lexical entry which has been found to be a possible candidate for a given sentence is asserted as a clause *lex(Node,InBag,OutBag)* where the *subbag* predicate is already partially evaluated. For example, given the sentence 'a b b c a a' the indexing step of the parser may find that the lexical entries 'a', 'b' and 'c' are applicable. These entries are then asserted as:

(32) $lex(ANode, \langle s(A), B, C\rangle, \langle A, B, C\rangle).$
$lex(BNode, \langle A, s(B), C\rangle, \langle A, B, C\rangle).$
$lex(CNode, \langle A, B, s(C)\rangle, \langle A, B, C\rangle).$

The guide is instantiated as follows. The in-part will simply be the bag representation shown above. More precisely:

$$\langle s(s(s(0))), s(s(0)), s(0)\rangle$$

and the out-part now simply is:

$$\langle 0, 0, 0\rangle$$

The 'predict_head' clause is re-defined as:

(33) $predict\_head(Goal, Lex, P_0, P)$:-
$head(Goal, Lex),$
$lex(Lex, P_0, P).$

5.3.2 'Order-monotonic' grammars

In some grammars the string operations that are defined are not only monotonic with respect to the words they dominate, but also with respect to the order constraints that are defined between these words ('order-monotonic'). For example in Reape's sequence union operation the linear precedence constraints that are defined between elements of a daughter are by definition part of the linear precedence constraints of the mother. Note though that the analysis of verb second in the foregoing section uses a string operation that does not satisfy this restriction. For grammars that do satisfy this restriction it is possible to extend the top-down prediction possibilities by the incorporation of an extra clause in the 'head_corner' predicate which will check that the phrase that has been analysed up to that point can become a substring of the top string. To this purpose, the input string is percolated through the parser as an extra argument. Each time a rule has applied the parser checks whether the string derived up to that point can be a subsequence of the complete string. This is achieved using the *subseq/2* predicate. In that case the revised 'head_corner' clause looks as follows:

(34) $head\_corner(Sign, Sign, P, P, String).$
$head\_corner(Small, Big, P_0, P, String)$:-
$rule(Small, Mid, Others, Call),$
$parse\_rest(Others, P_0, P_1, String),$
$call(Call),$
$string(Mid, Sub),$
$subseq(Sub, String),$
$head\_corner(Mid, Big, P_1, P, String).$

$subseq(\langle\rangle, Seq).$
$subseq(\langle H|T\rangle, \langle H|T_2\rangle)$:-
$subseq(T, T_2).$
$subseq(\langle H|T\rangle, \langle H_2|T_2\rangle)$:-
$subseq(\langle H|T\rangle, T_2).$

5.3.3 Delaying the extra constraint

In some cases it is very useful to delay the constraint which defines the operations on strings until the parser is finished. For example, if the constraints are disjunctive then it may be useful to wait as long as possible before a choice in a certain direction is to be made. The important information for the parser is percolated anyway through the bag of words; the actual order of the words has (usually) not much influence on other choices of the parser. Hence a lot of uninteresting non determinism (from the parser's point of view) can thus be delayed. In practice this increased the efficiency of the parser for some grammars by a factor 3. Clearly this technique is incompatible with the improvement suggested above for order-monotonic grammars.

Acknowledgements

This research was supported by SFB 314, Project N3 BiLD. I am grateful to Mike Reape for useful comments.

References

Emmon Bach. Control in Montague grammar. *Linguistic Inquiry*, 10:515–553, 1979.

James Blevins. Derived constituent structure, 1990. Ms. University of Texas at Austin & MCC.

Jonathan Calder, Mike Reape, and Henk Zeevat. An algorithm for generation in unification categorial grammar. In *Fourth Conference of the European*

Chapter of the Association for Computational Linguistics, pages 233–240, Manchester, 1989.

David Dowty. Towards a minimalist theory of syntactic structure. In *Proceedings of the Symposium on Discontinuous Constituency*, ITK Tilburg, 1990.

Marc Dymetman, Pierre Isabelle, and Francois Perrault. A symmetrical approach to parsing and generation. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, Helsinki, 1990.

Arnold Evers. *The Transformational Cycle in Dutch and German.* PhD thesis, Rijksuniversiteit Utrecht, 1975.

Dale Douglas Gerdemann. *Parsing and Generation of Unification Grammars.* PhD thesis, University of Illinois at Urbana-Champaign, 1991. Cognitive Science technical report CS-91-06 (Language Series).

Andrew Haas. A parsing algorithm for unification grammar. *Computational Linguistics*, 15(4), 1989.

J. Haviland. Guugu yimidhirr. In R. Dixon and B.Blake, editors, *Handbook of Australian Languages.* Benjamins Amsterdam, 1979.

Markus Höhfeld and Gert Smolka. Definite relations over constraint languages. Technical report, 1988. LILOG Report 53; to appear in Journal of Logic Programming.

Mark Johnson. Parsing with discontinuous constituents. In *23th Annual Meeting of the Association for Computational Linguistics*, Chicago, 1985.

A.K. Joshi, L.S. Levy, and M. Takahashi. Tree adjunct grammars. *Journal Computer Systems Science*, 10(1), 1975.

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

Jan Koster. Dutch as an SOV language. *Linguistic Analysis*, 1, 1975.

Y. Matsumoto, H. Tanaka, H. Hirakawa, H. Miyoshi, and H. Yasukawa. BUP: a bottom up parser embedded in Prolog. *New Generation Computing*, 1(2), 1983.

J. McCloskey. A VP in a VSO language? In Gerald Gazdar, Ewan Klein, and Geoffrey K. Pullum, editors, *Order, Concord and Constituency.* Foris, 1983.

Fernando C.N. Pereira and Stuart M. Shieber. *Prolog and Natural Language Analysis*. Center for the Study of Language and Information Stanford, 1987.

Fernando C.N. Pereira and David Warren. Parsing as deduction. In *21st Annual Meeting of the Association for Computational Linguistics*, Cambridge Massachusetts, 1983.

Carl Pollard and Ivan Sag. *Information Based Syntax and Semantics, Volume 1*. Center for the Study of Language and Information Stanford, 1987.

Carl Pollard. *Generalized Context-Free Grammars, Head Grammars, and Natural Language*. PhD thesis, Stanford, 1984.

Mike Reape. A logical treatment of semi-free word order and bounded discontinuous constituency. In *Fourth Conference of the European Chapter of the Association for Computational Linguistics*, UMIST Manchester, 1989.

Mike Reape. Getting things in order. In *Proceedings of the Symposium on Discontinuous Constituency*, ITK Tilburg, 1990.

Mike Reape. Parsing bounded discontinuous constituents: Generalisations of some common algorithms. In *Proceedings of the first CLIN dag*. OTS RUU Utrecht, 1991. to appear.

Stuart M. Shieber, Gertjan van Noord, Robert C. Moore, and Fernando C.N. Pereira. A semantic-head-driven generation algorithm for unification based formalisms. In *27th Annual Meeting of the Association for Computational Linguistics*, Vancouver, 1989.

Stuart M. Shieber, Gertjan van Noord, Robert C. Moore, and Fernando C.N. Pereira. Semantic-head-driven generation. *Computational Linguistics*, 16(1), 1990.

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.

Stuart M. Shieber. *Parsing and Type Inference for Natural and Computer Languages*. PhD thesis, Menlo Park, 1989. Technical note 460.

Gertjan van Noord. BUG: A directed bottom-up generator for unification based formalisms. *Working Papers in Natural Language Processing, Katholieke Universiteit Leuven, Stichting Taaltechnologie Utrecht*, 4, 1989.

Gertjan van Noord. An overview of head-driven bottom-up generation. In Robert Dale, Chris Mellish, and Michael Zock, editors, *Current Research in Natural Language Generation*. Academic Press, 1990.

K. Vijay-Shankar and A. Joshi. Feature structure based tree adjoining grammar. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING)*, Budapest, 1988.

K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In *25th Annual Meeting of the Association for Computational Linguistics*, Stanford, 1987.

David J. Weir. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1988.

Henk Zeevat, Ewan Klein, and Jo Calder. Unification categorial grammar. In Nicholas Haddock, Ewan Klein, and Glyn Morrill, editors, *Categorial Grammar, Unification Grammar and Parsing*. Centre for Cognitive Science, University of Edinburgh, 1987. Volume 1 of Working Papers in Cognitive Science.