Constraint-Based Categorial Grammar

Gosse Bouma and Gertjan van Noord
Alfa-informatica and
Behavorial and Cognitive Neurosciences,
Rijksuniversiteit Groningen
{gosse,vannoord}@let.rug.nl

Abstract

We propose a generalization of Categorial Grammar in which lexical categories are defined by means of recursive constraints. In particular, the introduction of relational constraints allows one to capture the effects of (recursive) lexical rules in a computationally attractive manner. We illustrate the linguistic merits of the new approach by showing how it accounts for the syntax of Dutch cross-serial dependencies and the position and scope of adjuncts in such constructions. Delayed evaluation is used to process grammars containing recursive constraints.

1 Introduction

Combinations of Categorial Grammar (CG) and unification naturally lead to the introduction of polymorphic categories. Thus, Karttunen (1989) categorizes NP's as X/X, where X is a verbal category, Zeevat *et al.* (1987) assign the category X/(NP\X) to NP's, and Emms (1993) extends the Lambek-calculus with polymorphic categories to account for coordination, quantifier scope, and extraction.

The role of polymorphism has been restricted, however, by the fact that in previous work categories were defined as feature structures using the simple, non-recursive, constraints familiar from feature description languages such as PATR. Relational constraints can be used to define a range of polymorphic categories that are beyond the expressive capabilities of previous approaches.

In particular, the introduction of relational constraints captures the effects of (recursive) lexical rules in a computationally attractive manner. The addition of such rules makes it feasible to consider truly 'lexicalist' grammars, in which a powerful lexical component is accompanied by a highly restricted syntactic component, consisting of application only.

2 Recursive Constraints

In CG, many grammatical concepts can only be defined recursively. Dowty (1982) defines grammatical functions such as *subject* and *object* as being the ultimate and penultimate 'argument-in' of a verbal category. Hoeksema (1984) defines verbs as exocentric categories reducible to S. Lexical rules frequently refer to such concepts. For instance, a categorial lexical rule of passive applies to *verbs* selecting an *object* and must remove the *subject*.

In standard unification-based formalisms, these concepts and the rules referring to such concepts cannot be expressed directly.

2.1 Subject-verb agreement

Consider a categorial treatment of subject-verb agreement with intransitive (NP[NOM]\S) and transitive ((NP[NOM]\S)/NP[ACC]) verbs defined as follows:

(1) *lex*(walks, X) :-
  iv(X).
lex(kisses, X) :-
  tv(X).

$$iv(\begin{bmatrix} \mathit{val} & \begin{bmatrix} \mathit{cat}\ \mathrm{s} \end{bmatrix} \\ \mathit{dir} & \text{`}\backslash\text{'} \\ \mathit{arg} & \begin{bmatrix} \mathit{cat}\ \mathrm{np} \\ \mathit{case}\ \mathrm{nom} \end{bmatrix} \end{bmatrix}).$$

$$tv(\begin{bmatrix} \mathit{val} & \begin{bmatrix} \mathit{val} & \begin{bmatrix} \mathit{cat}\ \mathrm{s} \end{bmatrix} \\ \mathit{dir} & \text{`}\backslash\text{'} \\ \mathit{arg} & \begin{bmatrix} \mathit{cat}\ \mathrm{np} \\ \mathit{case}\ \mathrm{nom} \end{bmatrix} \end{bmatrix} \\ \mathit{dir} & \text{`/'} \\ \mathit{arg} & \begin{bmatrix} \mathit{cat}\ \mathrm{np} \\ \mathit{case}\ \mathrm{acc} \end{bmatrix} \end{bmatrix}).$$

Subject-verb agreement can be incorporated easily if one reduces agreement to a form of subcategorization.

If, however, one wishes to distinguish these two pieces of information (to avoid a proliferation of subcategorization types or for morphological reasons, for instance), it is not obvious how this could be done without recursive constraints. For intransitive verbs one needs the constraint that $\langle arg\ agr \rangle = Agr$ (where Agr is some agreement value), for transitive verbs that $\langle val\ arg\ agr \rangle = Agr$, and for ditransitive verbs that $\langle val\ val\ arg\ agr \rangle = Agr$. The generalization is captured using the recursive constraint *sv_agreement* (2). In (2) and below, we use definite clauses to define lexical entries and constraints. Note that lexical entries relate words to feature structures that are defined indirectly as a combination of simple constraints (evaluated by means of unification) and recursive constraints.[1]

(2) *lex*(walks, X) :-
    iv(X),
    sv_agreement(sg3, X).
  lex(kisses, X) :-
    tv(X),
    sv_agreement(sg3, X).

$$sv\_agreement(\mathrm{Agr}, \begin{bmatrix} cat\ \mathrm{np} \\ agr\ \mathrm{Agr} \end{bmatrix} \backslash \mathrm{S}).$$

  sv_agreement(Agr, Y\X) :-
    sv_agreement(Agr, X).

Relational constraints can also be used to capture the effect of lexical rules. In a lexicalist theory such as CG, in which syntactic rules are considered to be universally valid scheme's of functor-argument combination, lexical rules are an essential tool for capturing language-specific generalizations. As Carpenter (1991) observes, some of the rules that have been proposed must be able to operate recursively. Predicative formation in English, for instance, uses a lexical rule turning a category reducible to VP into a category reducing to a VP-modifier (VP\VP). As a VP-modifier is reducible to VP, the rule can (and sometimes must) be applied recursively.

2.2 Adjuncts as arguments

Miller (1992) proposes a lexical rule for French nouns which adds an (modifying) adjective to the list of arguments that the noun subcategorizes for. Since a noun can be modified by any number of adjectives, the rule must be optional as well as recursive. The advantages of using a lexical rule in this case is that it simplifies accounting for agreement between nouns and adjectives and that it enables an account of word order constraints between arguments and modifiers of a noun in terms of *obliqueness*.

The idea that modifiers are introduced by means of a lexical rule can be extended to verbs. That is, adjuncts could be introduced by means of a recursive rule that optionally adds these elements to verbal categories. Such a rule would be an alternative for the standard categorial analysis of adjuncts as (endocentric) functors. There is reason to consider this alternative.

In Dutch, for instance, the position of verb modifiers is not fixed. Adjuncts can in principle occur anywhere to the left of the verb:[2]

(3) *a.* dat Johan opzettelijk een ongeluk veroorzaakt
  that J. deliberately an accident causes
  that J. deliberately causes an accident

 b. dat Johan Marie opzettelijk geen cadeau geeft
  that J. M. deliberately no present gives
  that J. deliberately gave M. no present

There are several ways to account for this fact. One can assign multiple categories to adjuncts or one can assign a polymorphic category X/X to adjuncts, with X restricted to 'verbal projections' (Bouma, 1988).

Alternatively, one can assume that adjuncts are not functors, but arguments of the verb. Since adjuncts are optional, can be iterated, and can occur in several positions, this implies that verbs must be polymorphic. The constraint *add_adjuncts* has this effect, as it optionally adds one or more adjuncts as arguments to the 'initial' category of a verb:

(4) *lex*(veroorzaken, X):-
    add_adjuncts(X, NP\(NP\S)).
  lex(geven, X) :-
    add_adjuncts(X, NP\(NP\(NP\S))).

  add_adjuncts(S, S).
  add_adjuncts(Adj\X, Y) :-
    add_adjuncts(X, Y).

$$add\_adjuncts(\begin{bmatrix} val\ \mathrm{X} \\ dir\ \mathrm{D} \\ arg\ \mathrm{A} \end{bmatrix}, \begin{bmatrix} val\ \mathrm{Y} \\ dir\ \mathrm{D} \\ arg\ \mathrm{A} \end{bmatrix}) \text{ :-}$$

    add_adjuncts(X, Y).

[1] We use X/Y and $Y\backslash X$ as shorthand for $\begin{bmatrix} val\ \mathrm{X} \\ dir\ \text{'/'} \\ arg\ \mathrm{Y} \end{bmatrix}$ and $\begin{bmatrix} val\ \mathrm{X} \\ dir\ \text{'\textbackslash'} \\ arg\ \mathrm{Y} \end{bmatrix}$, respectively and S, NP, and Adj as 'typed variables' of type $\begin{bmatrix} cat\ \mathrm{s} \end{bmatrix}$, $\begin{bmatrix} cat\ \mathrm{np} \end{bmatrix}$, and $\begin{bmatrix} cat\ \mathrm{adj} \end{bmatrix}$, respectively.

[2] As we want to abstract away from the effects of 'verb-second', we present only examples of subordinate clauses.

This constraint captures the effect of applying the following (schematic) lexical rule recursively:

(5)
$$X_1\backslash\ldots\backslash X_i\backslash X_{i+1}\backslash\ldots\backslash S/Y_1\ldots Y_n$$
$$\Downarrow$$
$$X_1\backslash\ldots\backslash X_i\backslash\mathrm{Adj}\backslash X_{i+1}\backslash\ldots\backslash S/Y_1\ldots Y_n$$

The derivation of (3a) is given below (where $X \Rightarrow Y$ indicates that *add_adjuncts*(Y,X) is satisfied, and IV = NP\S).

(6)
$$\begin{array}{cccc}
\ldots\mathit{J.} & \mathit{opzettelijk} & \mathit{een\ ongeluk} & \mathit{veroorzaakt} \\
\mathrm{NP} & \mathrm{ADJ} & \mathrm{NP} & \mathrm{NP}\backslash\mathrm{IV} \\
 & & & \Downarrow \\
 & & & \mathrm{NP}\backslash(\mathrm{ADJ}\backslash\mathrm{IV}) \\
\hline
 & & \mathrm{ADJ}\backslash\mathrm{IV} & \\
\hline
 & \mathrm{IV} & & \\
\hline
\mathrm{S} & & &
\end{array}$$

An interesting implication of this analysis is that in a categorial setting the notion 'head' can be equated with the notion 'main functor'. This has been proposed by Barry and Pickering (1990), but they are forced to assign a category containing Kleene-star operators to verbal elements. The semantic counterpart of such category-assignments is unclear. The present proposal is an alternative for such assignments which avoids introducing new categorial operators and which does not lead to semantic complications (the semantics of *add_adjuncts* is presented in section 3.3). Below we argue that this analysis also allows for a straightforward explanation of the distribution and scope of adjuncts in verb phrases headed by a verbal complex.

3 Cross-Serial Dependencies

In Dutch, verbs selecting an infinitival complement (e.g. modals and perception verbs) give rise to so called cross-serial dependencies. The arguments of the verbs involved appear in the same order as the verbs in the 'verb cluster':

(7) *a.* dat An_1 Bea_2 wil_1 kussen_2.
dat An Bea wants to kiss
that An wants to kiss Bea

b. dat An_1 Bea_2 Cor_3 wil_1 zien_2 kussen_3.
dat An Bea Cor wants to see kiss
that An wants to see Bea kiss Cor

The property of forming cross-serial dependencies is a lexical property of the matrix verb. If this verb is a 'trigger' for cross-serial word order, this order is obligatory, whereas if it is not, the infinitival complement will follow the verb:

(8) *a.* *dat An wil Bea kussen.

b. dat An zich voornam Bea te kussen.
that An Refl. planned Bea to kiss
that An. planned to kiss Bea

c. *dat An zich Bea voornam te kussen.

3.1 Generalized Division

Categorial accounts of cross-serial dependencies initially made use of a syntactic rule of composition (Steedman, 1985). Recognizing the lexical nature of the process, more recent proposals have used either a lexical rule of composition (Moortgat, 1988) or a lexical rule of 'division' (Hoeksema, 1991). Division is a rule which enables a functor to inherit the arguments of its argument:[3]

$$X/Y \Rightarrow (X/Z_1\ldots/Z_n)/(Y/Z\ldots/Z_n)$$

To generate cross-serial dependencies, a 'disharmonic' version of this rule is needed:

(9) $X/Y \Rightarrow (Z_1\backslash\ldots Z_n\backslash X)/(Z_1\backslash\ldots Z_n\backslash Y)$

Hoeksema proposes that verbs which trigger cross-serial word order are subject to (9):

(10)
$$\begin{array}{cccc}
\ldots\mathit{An} & \mathit{Bea} & \mathit{wil} & \mathit{kussen} \\
\mathrm{NP} & \mathrm{NP} & \mathrm{IV}/\mathrm{IV} & \mathrm{NP}\backslash\mathrm{IV} \\
 & & \Downarrow & \\
 & & (\mathrm{NP}\backslash\mathrm{IV})/(\mathrm{NP}\backslash\mathrm{IV}) & \\
\hline
 & & \mathrm{NP}\backslash\mathrm{IV} & \\
\hline
 & \mathrm{IV} & &
\end{array}$$

In a framework using recursive constraints, generalized disharmonic division can be implemented as a recursive constraint connecting the initial category of such verbs with a derived category:

(11) *lex*(willen, X) :-
cross_serial(X, (NP\S)/(NP\S)).
lex(zien, X) :-
cross_serial(X, (NP\(NP\S))/(NP\S)).
lex(voornemen, $(\mathrm{NP}_{\mathrm{refl}}\backslash(\mathrm{NP}\backslash\mathrm{S}))/(\mathrm{NP}\backslash\mathrm{S})$).

[3] Argument inheritance is used in HPSG to account for verb clustering in German (Hinrichs and Nakazawa, 1989). The HPSG analysis is essentially equivalent to Hoeksema's account.

(12) *cross_serial*(Out, In) :-
division(Out, In),
verb_cluster(Out).

$division(X, X).$
$division((Z\backslash X)/(Z\backslash Y), X'/Y')$:-
$division(X/Y, X'/Y').$

$verb\_cluster(\begin{bmatrix} arg & [\ vc + \ ] \end{bmatrix}).$

Only verbs that trigger the cross-serial order are subject to the *division* constraint. This accounts immediately for the fact that cross-serial orders do not arise with all verbs selecting infinitival complements.

3.2 Verb Clusters

The *verb_cluster* constraint ensures that cross-serial word order is obligatory for verbs subject to *cross_serial*. To rule out the ungrammatical (8a), for instance, we assume that *Bea kussen* is not a verb cluster. The verb *kussen* by itself, however, is unspecified for VC, and thus (7a) is not excluded.

We do not assume that cross-serial verbs take lexical arguments (as has sometimes been suggested), as that would rule out the possibility of complex constituents to the right of cross-serial verbs altogether. If one assumes that a possible bracketing of the verb cluster in (7b) is *[wil [zien kussen]]* (coordination and fronting data have been used as arguments that this is indeed the case), a cross-serial verb must be able to combine with non-lexical verb clusters. Furthermore, if a verb selects a particle, the particle can optionally be included in the verb cluster, and thus can appear either to the right or to the left of a governing cross-serial verb. For a verb cluster containing two cross-serial verbs, for instance, we have the following possibilities:

(13) *a.* dat An Bea heeft durven aan te spreken
that An Bea has dared part. to speak
that An has dared to speak to Bea.
b. dat An Bea heeft aan durven te spreken.
c. dat An Bea aan heeft durven te spreken.

A final piece of evidence for the the fact that cross-serial verbs may take complex phrases as argument stems from the observation that certain adjectival and prepositional arguments can also appear as part of the verb cluster:

(14) dat An dit aan Bea had duidelijk gemaakt
that An this to Bea has clear made
that An had made this clear to Bea

Cross-serial verbs select a +VC argument. Therefore, all phrases that are not verb clusters must be marked -VC. In general, in combining a (verbal) functor with its argument, it is the argument that determines whether the resulting phrase is -VC. For instance, NP-arguments always give rise to -VC phrases, whereas particles and verbal arguments do not give rise to -VC phrases. This suggests that NP's must be marked -VC, that particles and verbs can remain unspecified for this feature, and that in the syntactic rule for application the value of the feature VC must be reentrant between argument and resultant.

3.3 The distribution and scope of adjuncts

The analysis of cross-serial dependencies in terms of argument inheritance interacts with the analysis of adjuncts presented in section 2.2. If a matrix verb inherits the arguments of the verb it governs, it should be possible to find modifiers of the matrix verb between this verb and one of its inherited arguments. This prediction is borne out (15a). However, we also find structurally similar examples in which the adjunct modifies the governed verb (15b). Finally, there are examples that are ambiguous between a wide and narrow scope reading (15c). We take it that the latter case is actually what needs to be accounted for, i.e. examples such as (15a) and (15b) are cases in which there is a strong preference for a wide and narrow scope reading, respectively, but we will remain silent about the (semantic) factors determining such preferences.

(15) *a.* dat Frits Marie volgens mij lijkt te ontwijken.
that F. M. to me seems to avoid
It seems to me that F. avoids M.
b. dat Frits Marie opzettelijk lijkt te ontwijken.
that F. M. deliberately seems to avoid
It seems that F. deliberately avoids M.
c. dat Frits Marie de laatste tijd lijkt te ontwijken.
that F. M. lately seems to avoid
It seems lately as if F. avoids M.
It seems as if F. avoids M. lately

On the assumption that the lexical entries for *lijken* en *ontwijken* are as in (16), example (15c) has two possible derivations ((17) and (18)). Procedurally speaking, the rule that adds adjuncts can be applied either to the matrix verb (after division has taken place) or to the governed verb. In the latter case, the adjunct is 'inherited' by the matrix verb. Assuming that adjuncts take scope over the verbs introducing them, this accounts for the ambiguity observed above.

(16) $lex(\text{lijken}, \text{Verb})$:-
$add\_adjuncts(\text{Verb}, \text{Verb}')$,
$cross\_serial(\text{Verb}', (\text{NP}\backslash\text{S})/(\text{NP}\backslash\text{S}))$.
$lex(\text{ontwijken}, \text{Verb})$:-
$add\_adjuncts(\text{Verb}, \text{NP}\backslash(\text{NP}\backslash\text{S}))$.

(17)
$$\begin{array}{ccc}
\textit{... de laatste tijd} & \textit{lijkt} & \textit{te ontwijken} \\
\text{ADJ} & \text{IV/IV} & \text{TV} \\
 & & \Downarrow \\
 & & \text{TV/TV} \\
 & & \Downarrow \\
 & & (\text{ADJ}\backslash\text{TV})/\text{TV} \\
\hline
 & \multicolumn{2}{c}{\text{ADJ}\backslash\text{TV}} \\
\hline
\multicolumn{3}{c}{\text{TV}}
\end{array}$$

(18)
$$\begin{array}{ccc}
\textit{... de laatste tijd} & \textit{lijkt} & \textit{te ontwijken} \\
\text{ADJ} & \text{IV/IV} & \text{TV} \\
 & \Downarrow & \Downarrow \\
 & (\text{ADJ}\backslash\text{TV})/(\text{ADJ}\backslash\text{TV}) & \text{ADJ}\backslash\text{TV} \\
\hline
 & \multicolumn{2}{c}{\text{ADJ}\backslash\text{TV}} \\
\hline
\multicolumn{3}{c}{\text{TV}}
\end{array}$$

The assumption that adjuncts scope over the verbs introducing them can be implemented as follows. We use a unification-based semantics in the spirit of Pereira and Shieber (1987). Furthermore, the semantics is *head-driven*, i.e. the semantics of a complex constituent is reetrant with the semantics of its head (i.e. the functor). The feature structure for a transitive verb including semantics (taking two NP's of the generalized quantifier type $\langle\langle e,t\rangle,t\rangle$ as argument and assigning wide scope to the subject) is:

$$(19)\quad \left[\begin{array}{ll}
\textit{val} & \left[\begin{array}{ll}
\textit{val} & [\ \textit{cat}\ \text{s}\ ] \\
\textit{dir} & `\backslash' \\
\textit{arg} & \left[\begin{array}{l} \textit{cat}\ \text{np} \\ \textit{sem}\ (\text{X}\hat{\ }\text{S}_{Obj})\hat{\ }\text{S}_{Subj} \end{array}\right]
\end{array}\right] \\
\textit{dir} & `\backslash' \\
\textit{arg} & \left[\begin{array}{l} \textit{cat}\ \text{np} \\ \textit{sem}\ (\text{Y}\hat{\ }\text{kiss(X,Y)})\hat{\ }\text{S}_{Obj} \end{array}\right] \\
\textit{sem} & \text{S}_{Subj}
\end{array}\right]_{TV}$$

Thus, a lexical entry for a transitive verb can be defined as follows (where TV refers to the feature structure in 19):

(20) $lex(\text{kussen}, X)$:-
$add\_adjuncts(\text{X}, \text{TV})$.

The lexical rule for adding adjuncts can now be extended with a semantics:

(21) $add\_adjuncts([\ \textit{sem}\ \text{S}_X\ ]_X, [\ \textit{sem}\ \text{S}_Y\ ]_Y)$:-
$add\_adj(\text{X}, \text{Y}, \text{S}_X, \text{S}_Y)$.

$add\_adj(\text{S}, \text{S}, \text{Sem}, \text{Sem})$.

$$add_adj(\left[\begin{array}{ll}
\textit{val} & \text{X} \\
\textit{dir} & `\backslash' \\
\textit{arg} & \left[\begin{array}{l} \textit{cat}\ \text{adj} \\ \textit{sem}\ \text{S}_Y\hat{\ }\text{S}_A \end{array}\right]
\end{array}\right], \text{Y}, \text{S}_X, \text{S}_Y) \text{ :-}$$
$add\_adj(\text{X}, \text{Y}, \text{S}_X, \text{S}_A)$.

$$add\_adj(\left[\begin{array}{l} \textit{val}\ \text{X} \\ \textit{dir}\ \text{D} \\ \textit{arg}\ \text{A} \end{array}\right], \left[\begin{array}{l} \textit{val}\ \text{Y} \\ \textit{dir}\ \text{D} \\ \textit{arg}\ \text{A} \end{array}\right], \text{S}_X, \text{S}_Y) \text{ :-}$$
$add\_adj(\text{X}, \text{Y}, \text{S}_X, \text{S}_Y)$.

Each time an adjunct is added to the subcategorization frame of a verb, the semantics of the adjunct is 'applied' to the semantics as it has been built up so far (S_Y), and the result (S_A) is passed on. The final step in the recursion unifies the semantics that is constructed in this way with the semantics of the 'output' category. As an adjunct A_1 that appears to the left of an adjunct A_2 in the string will be added to the subcategorization frame of the governing verb after A_2 is added, this orders the (sentential) scope of adjuncts according to left-to-right word order. Furthermore, since the scope of adjuncts is now part of a verb's lexical semantics, any functor taking such a verb as argument (e.g. verbs selecting for an infinitival complement) will have the semantics of these adjuncts in its scope.

Note that the alternative treatments of adjuncts mentioned in section 2.2 cannot account for the distribution or scope of adjuncts in cross-serial dependency constructions. Multiple (i.e. a finite number of) categorizations cannot account for all possible word orders, since division implies that a trigger for cross-serial word order may have any number of arguments, and thus, that the number of 'subcategorization frames' for such verbs is not fixed. The polymorphic solution (assigning adjuncts the category X/X) does account for word order, but cannot account for narrow scope readings, as the adjunct will always modify the whole verb cluster (i.e the matrix verb) and cannot be made to modify an embedded verb only.

4 Processing

The introduction of recursive lexical rules has repercussions for processing as they lead to an infinite number of lexical categories for a given lexical item or, if one considers lexical rules as unary syntactic rules, to non-branching derivations of unbounded length. In both cases, a parser may not terminate. One of the main advantages of modeling lexical rules by means of constraints is that it suggests a solution for this problem. A control strategy which delays the evaluation of constraints until certain crucial bits of information are filled in avoids non-termination and in practice leads to grammars in which all constraints are fully evaluated at the end of the parse-process.

Consider a grammar in which the only recursive constraint is *add_adjuncts*, as defined in section 2.2. The introduction of recursive constraints in itself does not solve the non-termination problem. If all solutions for *add_adjuncts* are simply enumerated during lexical look-up an infinite number of categories for any given verb will result.

During processing, however, it is not necessarily the case that we need to consider all solutions. Syntactic processing can lead to a (partial) instantiation of the arguments of a constraint. If the right pieces of information are instantiated, the constraint will only have a finite number of solutions.

Consider, for instance, a parse for the following string.

(22)
$$
\begin{array}{cccc}
\ldots J. & opzettelijk & een\ ongeluk & veroorzaakt \\
\textsc{np} & \textsc{adj} & \textsc{np} & Verb \\
 & & & \approx \\
 & & & \textsc{np}\backslash(\textsc{adj}\backslash\textsc{iv}) \\
\hline
 & & \textsc{adj}\backslash\textsc{iv} & \\
\hline
 & \textsc{np}\backslash\textsc{s} & & \\
\hline
 & \textsc{s} & &
\end{array}
$$

Even if the category of the verb is left completely open initially, there is only one derivation for this string that reduces to S (remember that the syntax uses application only). This derivation provides the information that the variable *Verb* must be a transitive verb selecting one additional adjunct, and with this information it is easy to check whether the following constraint is satisfied:

$$ add\_adjuncts(\textsc{np}\backslash(\textsc{adj}\backslash(\textsc{np}\backslash\textsc{s})), \textsc{np}\backslash(\textsc{np}\backslash\textsc{s})). $$

This suggests that recursive constraints should not be evaluated during lexical look-up, but that their evaluation should be delayed until the arguments are sufficiently instantiated.

To implement this delayed evaluation strategy, we used the `block` facility of Sicstus Prolog. For each recursive constraint, a `block` declaration defines what the conditions are under which it may be evaluated. The definition of *add_adjuncts* (with semantics omitted for readability), for instance, now becomes:

(23)
$$
\begin{array}{l}
add\_adjuncts(\begin{bmatrix} arg\ \mathrm{Arg} \end{bmatrix}_X, \mathrm{Y}) \text{ :-} \\
\quad add\_adjuncts(\mathrm{X}, \mathrm{Y}, \mathrm{Arg}). \\
\\
\text{:- } \mathtt{block}\ add\_adjuncts(?, ?, -). \\
\\
add\_adjuncts(\mathrm{S}, \mathrm{S}, \_). \\
add\_adjuncts(\mathrm{Adj}\backslash\mathrm{X}, \mathrm{Y}, \_) \text{ :-} \\
\quad add\_adjuncts(\mathrm{X}, \mathrm{Y}). \\
add\_adjuncts(\begin{bmatrix} val\ \mathrm{X} \\ dir\ \mathrm{D} \\ arg\ \mathrm{A} \end{bmatrix}, \begin{bmatrix} val\ \mathrm{Y} \\ dir\ \mathrm{D} \\ arg\ \mathrm{A} \end{bmatrix}, \mathrm{A}) \text{ :-} \\
\quad add\_adjuncts(\mathrm{X}, \mathrm{Y}).
\end{array}
$$

We use *add_adjuncts*/2 to extract the information that determines when *add_adjuncts*/3 is to be evaluated. The `block` declaration states that *add_adjuncts*/3 may only be evaluated if the third argument (i.e. the argument of the 'output' category) is not a variable. During lexical look-up, this argument is uninstantiated, and thus, no evaluation takes place. As soon as a verb combines with an argument, the argument category of the verb is instantiated and *add_adjuncts*/3 will be evaluated. Note, however, that calls to *add_adjuncts*/3 are recursive, and thus one evaluation step may lead to another call to *add_adjuncts*/3, which in its turn will be blocked until the argument has been instantiated sufficiently. Thus, the recursive constraint is evaluated incrementally, with each syntactic application step leading to a new evaluation step of the blocked constraint. The recursion will stop if an atomic category S is found.

Delayed evaluation leads to a processing model in which the evaluation of lexical constraints and the construction of derivational structure is completely intertwined.

4.1 Other strategies

The delayed evaluation techniques discussed above can be easily implemented in parsers which rely on backtracking for their search. For the grammars that we have worked with, a simple bottom-up (shift-reduce) parser combined with delayed evaluation guarantees termination of the parsing process.

To obtain an efficient parser more complicated search strategies are required. However, chart-based search techniques are not easily generalized for grammars which make use of complex constraints. Even if the theoretical problems can be solved (Johnson, 1993; Dörre, 1993) severe practical problems might surface, if the constraints are as complex as the ones proposed here.

As an alternative we have implemented chart-based parsers using the 'non-interleaved pruning' strategy (terminology from (Maxwell III and Kaplan, 1994)).

Using this strategy the parser first builds a parse-forest for a sentence on the basis of the context-free backbone of the grammar. In a second processing phase parses are recovered on the basis of the parse forest and the corresponding constraints are applied. This may be advantageous if the context-free backbone of the grammar is 'informative' enough to filter many unsuccessful partial derivations that the parser otherwise would have to check.

As clearly a CUG grammar does not contain such an informative context-free backbone a further step is to use 'selective feature movement' (cf. again (Maxwell III and Kaplan, 1994)). In this approach the base grammar is compiled into an equivalent modified grammar in which certain constraints from the base grammar are converted to a more complex context-free backbone in the modified grammar.

Again, this technique does not easily give good results for grammars of the type described. It is not clear at all where we should begin extracting appropriate features for such a modified grammar, because most information passing is simply too 'indirect' to be easily compiled into a context-free backbone.

We achieved the best results by using a 'hand-fabricated' context-free grammar as the first phase of parsing. This context-free grammar builds a parse forest that is then used by the 'real' grammar to obtain appropriate representation(s) for the input sentence. This turned out to reduce parsing times considerably.

Clearly such a strategy raises questions on the relation between this context-free grammar and the CUG grammar. The context-free grammar is required to produce a superset of the derivations allowed by the CUG. Given the problems mentioned above it is difficult to show that this is indeed the case (if it were easy, then it probably would also be easy to obtain such a context-free grammar automatically).

The strategy can be described in somewhat more detail as follows. The context-free phase of processing builds a number of items defining the parse forest, in a format that can be used by the second processing phase. Such items are four-tuples

$$\langle R, P_0, P, D \rangle$$

where R is a rule name (consistent with the rule names from the CUG), P_0, P are string positions and D describes the string positions associated with each daughter of the rule (indicating which part of the string is covered by that daughter).

Through a head-driven recursive descent the second processing phase recovers derivations on the basis of these items. Note that the delayed evaluation technique for complex constraints is essential here. Alternative solutions are obtained by backtracking. If the first phase has done a good job in pruning many failing search branches then this is not too expensive, and we do not have to worry about the interaction of caching and complex constraints.

5 Final Remarks

In sections 2 and 3 we have sketched an analysis of cross-serial dependency constructions and its interaction with the position and scope of adjuncts. The rules given there are actually part of a larger fragment that covers the syntax of Dutch verb clusters in more detail. The fragment accounts for cross-serial dependencies and extraposition constructions (including cases of 'partial' extraposition), *infinitivus-pro-participio*, modal and participle inversion, the position of particles in verb clusters, clitic climbing, partial VP-topicalization, and verb second. In the larger fragment, additional recursive constraints are introduced, but the syntax is still restricted to application only.

The result of Carpenter (1991) emphasizes the importance of lexical rules. There is a tendency in both CG and HPSG to rely more and more on mechanisms (such as inheritance and lexical rules or recursive constraints) that operate in the lexicon. The unrestricted generative capacity of recursive lexical rules implies that the remaining role of syntax can be extremely simple. In the examples above we have stressed this by giving an account for the syntax of cross-serial dependencies (a construction that is, given some additional assumptions, not context-free) using application only. In general, such an approach seems promising, as it locates the sources of complexity for a given grammar in one place, namely the lexicon.

References

Guy Barry and Martin Pickering. 1990. Dependency and coordination in categorial grammar. In G. Barry and G. Morrill, editors, *Studies in Categorial Grammar*. University of Edinburgh. Edinburgh Working Papers in Cognitive Science 5.

Gosse Bouma. 1988. Modifiers and specifiers in categorial unification grammar. *Linguistics*, 26:21–46.

Bob Carpenter. 1991. The generative power of categorial grammars and head-driven phrase structure grammars with lexical rules. *Computational Linguistics*, 17(3):301–313.

Jochen Dörre. 1993. Generalizing earley deduction for constraint-based grammars. In Jochen Doërre, editor, *Computational Aspects of Constraint-Based Linguistic Description*. Dyana-2 Deliverable R1.2.A.

David R. Dowty. 1982. Grammatical relations and montague grammar. In Pauline Jacobson and Geoffrey Pullum, editors, *The Nature of Syntactic Representation*, pages 79–130. Reidel, Dordrecht.

Martin Emms. 1993. Parsing with polymorphism. In *Sixth Conference of the European Chapter of the Association for Computational Linguistics*, pages 120–129.

Erhard Hinrichs and Tsuneko Nakazawa. 1989. Flipped out: AUX in German. In *Papers from the 25th Annual Regional Meeting of the Chicago Linguistic Society*, pages 187–202. Chicago Linguistics Society, Chicago.

Jack Hoeksema. 1984. *Categorial Morphology.* Ph.D. thesis, Groningen University.

Jack Hoeksema. 1991. A categorial theory of reanalysis phenomena. ms. RUG Groningen.

Mark Johnson. 1993. Memoization in constraint logic programming. Paper presented at the First Workshop on Principles and Practice of Constraint Programming, April 28-30, Newport, Rhode Island.

Lauri Karttunen. 1989. Radical lexicalism. In Mark R. Baltin and Anthony S. Kroch, editors, *Alternative Conceptions of Phrase Structure*, pages 43–65. University of Chicago Press.

John T. Maxwell III and Ronald M. Kaplan. 1994. The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–90.

Philip Miller. 1992. *Clitics and Constituents in Phrase Structure Grammar.* Garland, New York.

Michael Moortgat. 1988. *Categorial Investigations.* Ph.D. thesis, University of Amsterdam.

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

Mark Steedman. 1985. Dependency and coordination in the grammar of Dutch and English. *Language*, 61:523–68.

Henk Zeevat, Ewan Klein, and Jo Calder. 1987. 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. Volume 1 of Working Papers in Cognitive Science.