EFFICIENT PROCESSING OF FLEXIBLE CATEGORIAL GRAMMAR

Gosse Bouma
Research Institute for Knowledge Systems
Postbus 463, 6200 AL Maastricht
The Netherlands
e-mail(earn): exriksgb@hmarl5

ABSTRACT*

From a processing point of view, however, flexible categorial systems are problematic, since they introduce *spurious ambiguity*. In this paper, we present a flexible categorial grammar which makes extensive use of the product-operator, first introduced by Lambek (1958). The grammar has the property that for every reading of a sentence, a strictly left-branching derivation can be given. This leads to the definition of a subset of the grammar, for which the spurious ambiguity problem does not arise and efficient processing is possible.

1. Flexibility vs. Ambiguity

Categorial Grammars owe much of their popularity to the fact that they allow for various degrees of flexibility with respect to constituent structure. From a processing point of view, however, flexible categorial systems are problematic, since they introduce *spurious ambiguity*.

The best known example of a flexible categorial grammar is a grammar containing the reduction rules *application* and *composition*, and the category changing rule *raising* [1] :

(1)

application :	A/B	B	==>	A
	B	B\A	==>	A
composition :	A/B	B/C	==>	A/C
	C\B	B\A	==>	C\A
raising :	A		==>	(B/A)\B
	A		==>	B/(A\B)

With this grammar many alternative constituent structures for a sentence can be generated, even where this does not correspond to semantic ambiguities. From a linguistic point of view, this has many advantages. Various kind of arguments for giving up traditional conceptions of constituent structure can be given, but the most convincing and well-documented case in favour of flexible constituent structure is coordination (see Steedman (1985), Dowty (1988), and Zwarts (1986)).

The standard assumption in generative grammar is that coordination always takes place between between constituents. Right-node raising constructions and other instances of non-constituent conjunction are problematic, because it is not clear what the status of the coordinated elements in these constructions is. Flexible categorial grammar presents an elegant solution for such cases, since, next to canonical constituent structures, it also admits various other constituent structures. Therefore, the sentences in (2) can be considered to be ordinary instances of coordination (of two categories **s/np** and **(vp/np)\vp**, respectively).

(2) a. *John sold* and *Mary bought* a book
s/vp vp/np s/vp vp/np np
s/np s/np

* I would like to thank Esther König, Erik-Jan van der Linden, Michael Moortgat, Adriaan van Paassen and the participants of the Edinburgh Categorial Grammar Weekend, who made useful comments to earlier presentations of this material. All remaining errors and misconceptions are of course my own.

[1] Throughout this paper we will be using the notation of Lambek (1958), in which **A/B** and **B\A** are a right-directional and a left-directional functor respectively, looking for an argument of category **B**.

b. J. loves *Mary madly* and *Sue wildly*
vp/np np vp\vp np vp\vp

Mary madly
np vp\vp
____
(vp/np)\vp
_____
(vp/np)\vp

A somewhat different type of argument for flexible phrase structure is based on the way humans process natural language. In Ades & Steedman (1982) it is pointed out that humans process natural language in a left-to-right, incremental, manner. This processing aspect is accounted for in a flexible categorial system, where constituents can be built for any part of a sentence. Since syntactic rules operate in parallel with semantic interpretation rules, building a syntactic structure for an initial part of a sentence, implies that a corresponding semantic structure can also be constructed.

These and other arguments suggest that there is no such thing as a fixed constituent structure, but that the order in which elements combine with eachother is rather free.

From a parsing point of view, however, flexibility appears to be a disadvantage. Flexible categorial grammars produce large numbers of, often semantically equivalent, derivations for a given phrase. This *spurious ambiguity problem* (Wittenburg (1986)) makes efficient processing of flexible categorial grammar problematic, since quite often there is an exponential growth of the number of possible derivations, relative to the length of the string to be parsed.

There have been two proposals for eliminating spurious ambiguity from the grammar. The first is Wittenburg (1987). In this paper, a categorial grammar with composition and heavily restricted versions of raising (for subject **np**'s only) is considered. Wittenburg proposes to eliminate spurious ambiguity by redefining composition. His *predictive* composition rules apply only in those cases where they are really needed to make a derivation possible. A disadvantage of this method, noticed by Wittenburg, is that one may have to add special *predictive* composition rules for all general combinatory rules in the grammar. Some careful rewriting of the original grammar has to take place, before things work as desired.

Pareschi & Steedman (1987) propose an efficient chart-parsing algorithm for categorial grammars with spurious ambiguity. Instead of the usual strategy, in which all possible subconstituents are added to the chart, Pareschi & Steedman restrict themselves to adding only those constituents that may lead to a difference in semantics. Thus, in (3) only the underlined constituents are in the chart. The "---" constituent is not.

(3) John loves Mary madly
s/vp vp/np np vp\vp
_____
_____

Combining 'madly' with the rest would be impossible or lead to backtracking in the normal case. Here, the Pareschi & Steedman algorithm starts looking for a constituent left adjacent of *madly*, which contains an element **X/vp** as a leftmost category. If such a constituent can be found, it can be concluded that the rest of that constituent must (implicitly) be a **vp**, and thus the validity of combining **vp\vp** with this constituent has been established. Therefore, Pareschi & Steedman are able to work with only a minimal amount of items in the chart.

Both Wittenburg and Pareschi & Steedman work with categorial grammars, which contain restricted versions of composition and raising. Although they can be processed efficiently, there is linguistic evidence that they are not fully adequate for analysis of such phenomena as coordination. Since atomic categories can in general not be raised in these grammars, sentence (2b) (in which the category **np** has to be raised) cannot be derived. Furthermore, since composition is not generalized, as in Ades & Steedman (1982), a sentence such as *John sold but Mary donated a book to the library* would not be derivable. The possibilities for left-to-right, incremental, processing are also limited. Therefore, there is reason to look for a more flexible system, for which efficient parsing is still possible.

2. Structural Completeness

In the next section we present a grammatical calculus, which is more flexible than the systems considered by Wittenburg (1987) and Pareschi & Steedman (1987), and therefore is attractive for linguistic purposes. At the same time, it offers a solution to the spurious ambiguity problem.

Spurious ambiguity causes problems for parsing in the systems mentioned above, because there is no systematic relationship between syntactic structures and semantic representations. That is, there is no way to identify in advance, for a given sentence S, a proper subset of the set of all possible syntactic structures and associated semantic representations, for which it holds that it will contain all possible semantic representations of S.

Consider now a grammar for which the following property holds:

(4) **Structural Completeness**
If a sequence of categories $X_1 .. X_n$ reduces to Y, there is a reduction to Y for any bracketing of $X_1..X_n$ into constituents. (Moortgat, 1987:5)[2]

Structural complete grammars are interesting linguistically, since they are able to handle, for instance, all kinds of non-constituent conjunction, and also because they allow for strict left-to-right processing (see Moortgat, 1988).

The latter observation has consequences for parsing as well,, since, if we can parse every sentence in a strict left-to-right manner (that is, we produce only strictly left-branching syntax trees), the parsing algorithm can be greatly simplified. Notice, however, that such a parsing strategy is only valid, if we also guarantee that all possible readings of a sentence can be found in this way. Thus, instead of (4), we are looking for grammars with the following, slightly stronger, property:

(5) **Strong Structural Completeness**
If a sequence of categories $X_1 .. X_n$ reduces to Y, with semantics Y', there is a reduction to Y, with semantics Y', for any bracketing of $X_1..X_n$ into constituents.

Grammars with this property, can potentially circumvent the spurious ambiguity problem, since for these grammars, we only have to inspect all left-branching syntax trees, to find all possible readings. This method will only fail if the set of left-branching trees itself would contain spurious ambiguous derivations. In section 4 we will show that these can be eliminated from the calculus presented below.

3. The P-calculus

The P(roduct)-calculus is a categorial grammar, based on Lambek (1958), which has the property of strong structural completeness.

In Lambek (1958), the foundations of flexible categorial grammar are formulated in the form of a calculus for syntactic categories. Well-known categorial rules, such as application, composition and category-raising, are theorems of this calculus. A largely neglected aspect of this calculus, is the use of the product-operator.

The calculus we present below, was developed as an alternative for Moortgat's (1988) M-system. The M-system is a subset of the Lambek-calculus, which uses, next to application, only a very general form of composition. Since it has no raising, it seems to be an attractive candidate for investigating the possibilities of left-associative parsing for categorial grammar. It is not completely satisfactory, however, since structural completeness is not fully guaranteed, and also, since it is unknown whether the strong structural completeness property holds for this system. In our calculus, we hope to overcome these problems, by using product-introduction and -elimination rules instead of composition.

The kernel of the P-calculus is right- and left-application, as usual. Next to these,

---

2 Buszkowski (1988) provides a slightly different definition in terms of functor-argument structures.

we use a rule for introducing the product-operator, and two inference rules for eliminating products :

(6) RA : A/B B => A
LA : B B\A => A

(product) introduction:

I : A B => A*B

inference rules :

P : A B => C, D C => E

D*A B => E

P' : A B => C, C D => E

A B*D => E

We can use this calculus to produce left-branching syntax trees for any given (grammatical) sentence. (7) is a simple example[3] .

(7)
John	loves	Mary	madly
s/vp	vp/np	np	vp\vp
	---I---		
	s/vp*vp/np		
	---(a)---		
	s/vp*vp		
	---(b)---		
	s		

(a) vp/np np => vp, s/vp vp => s/vp*vp
---
s/vp*vp/np np => s/vp*vp

(b) vp vp\vp => vp , s/vp vp => s
---
s/vp*vp vp => s

The first step in the derivation of (7) is the application of rule I. The other two reductions ((a) and (b)) are instantiations of the inference rule P. As the example shows, the *-operator (more in particular its use in I) does something like concatenation, but whereas such operations are normally associated with particular grammatical rules (i.e. you may concatenate two elements of category **NP** and **VP**, respectively, if there is a rule **S --> NP VP**), or (in CG) with a reduction rule (application or composition, for instance), we now have the freedom to concatenate arbitrary categories, completely irrespective of their internal structure.

The P-calculus is structurally complete. To prove this, we prove that for any four categories **A, B, C, D**, it holds that : **(AB)C --> D <==> A(BC) --> D**, where **-->** is the derivability relation. From this, structural completeness may be concluded, since any bracketing (or branching of syntax trees) can be obtained by applying this equivalence an arbitrary number of times.

Proof : From **(AB)C --> D** it follows that there exists a category **E** such that **AB --> E** and **EC --> D**. **BC --> B*C**, by I. Now **A(B*C) --> D**, by P', since **AB --> E** and **EC --> D**. Therefore, by transitivity of **-->**, **A(BC) --> D**. To prove that **A(BC) --> D ==> (AB)C --> D** use P instead of P'.

Semantics can be added to the grammar, by giving a semantic counterpart (in lower case) for each of the rules in (6):

(8) RA : A/B:a B:b => A:a(b)
LA : B:b B\A:a => A:a(b)

(product) introduction:

I : A:a B:b => A*B:a*b

inference rules :

P : A:a B:b => C:c, D:d C:c => E:e

D*A:d*a B:b => E:e

P' : A:a B:b => C:c, C:c D:d => E:e

A:a B*D:b*d => E:e

We can now include semantics in the proof given above, and from this, we may conclude that strong structural completeness holds for the P-calculus as well.

4. Eliminating Spurious Ambiguity

In this section we outline a subset of the P-calculus, for which efficient processing is possible. As was noted above, in the P-calculus there is always a strictly left-branching derivation for any reading of a

3 To improve readability, we assume that the operators / and \ take precedence over * (**X*Y/Z** should be read as **X*(Y/Z)**).

sentence S. The restrictions we add in this section are needed to eliminate spurious ambiguities from these left-branching derivations.

Restricting a parser so that it will only accept left-branching derivations will not directly lead to an efficient parsing procedure for the P-calculus. The reason is twofold.

First, nothing in the P-calculus excludes spurious ambiguity which occurs within the set of left-branching analysis trees. Consider again example (7). This sentence is unambiguous, but nevertheless we can give a left-branching derivation for it which differs from the one given earlier :

(9) John loves Mary madly
s/vp vp/np np vp\vp
I
s/vp*vp/np
I
(s/vp*vp/np)*np
(**)
s

The inference step (**) can be proven to be valid, if we use P' as well as P.

An even more serious problem is caused by the interaction between I and P,P'.

(10)
A B => A*B A*B C => D
P'
B C => B*C A B*C => D
P
A*B C => D

If we try to prove that **A*B** and **C** can be reduced to a category **D**, we could use P, with I in the left premise. To prove the second premise, we could use P', also with using I in the left premise. But now the right premise of P' is identical to our initial problem, and thus we have made a useless loop, which could even lead to an infinite regress.

These problems can be eliminated, if we restrict the grammar in two ways. First of all, we consider only derivations of the form $C_1,...,C_n$ ==> S, where $C_1,...,C_n$,S do not contain the product-operator. This means we require that the start symbol of the grammar, and the set lexical categories must be product-free. Notice that this restriction can be easily made, since most categorial lexicons do not contain the product-operator anyway.

Given this restriction, the inference rule P can be restricted: we require that the left premise of this rules always is an instance of either left- or right-application. Consider what would happen if we used I here :

(11)
B C => B*C A B*C => D
***
P
A*B C => D

Since the lexicon is product-free, and we are interested in strictly left-branching derivations only, we know that **C** must be a product-free category. If we combine **B** and **C** through I, we are faced with the problem in ***. At this point we could use I again for instance, thereby instantiating **D** as **A*(B*C)**. But this will lead to a spurious ambiguity, since we know that:

A*(B*C) E => F iff (A*B)*C E => F[4] .

A category **(A*B)*C** can be obtained by applying I directly to **A*B** and **C**.

If we apply P' at point ***, we find ourselves trying to find a solution for **A B => E**, and then **E C => D**. But this is nothing else than trying to find a left-branching derivation for **A,B,C => D**, and therefore, the inference step in (11) has not led to anything new.

In fact, given that the lexicon is product free and only application may be used in the left premise of P, P' is never needed to derive a left-branching tree.

As a result, we get (12), where we have made a distinction between reduction rules (right and left-application) and other rules. This enables us to restrict the left premise of P. The fact that every reduction rule is also a general rule of the grammar, is expressed by R. P' has been eliminated.

4 In the P-calculus, this follows from the fact that E must be product-free. It is a theorem of the Lambek-calculus as well.

(12) RA : A/B B -> A
LA : B B\A -> A

(product) introduction:

I : A B => A*B

inference rules :

P : A B -> C, D C => E

D*A B => E

R : A B -> C

A B => C

The system in (12) is a subset of the P-calculus, which is able to generate a strictly left-branching derivation for every reading of a given sentence of the grammar.

The Prolog fragment in (13) shows how the restricted system in (12) can be used to define a simple left-associative parsing algorithm.

```
(13)  parse([C] ==> C) :- !.
      parse([C1,C2|Rest] ==> S) :-
              rule([C1,C2] ==> C3),
              parse([C3|Rest] ==> S).
      % R :
      rule(X ==> Y) :-
              reduction_rule(X ==> Y).

      % P :
      % '+' is used instead of '*' to avoid
      % unnecessary bracketing
      rule([X+Y,Z] ==> W) :-
              reduction_rule([Y,Z] ==> V),
              rule([X,V] ==> W).
      % I:
      rule([X,Y] ==> X+Y).

      % application :
      reduction_rule([X/Y,Y] ==> X).
      reduction_rule([Y,Y\X] ==> X).
```

5. Shift-reduce parsing

It has sometimes been noted that a derivation tree in categorial grammar (such as (7)) does not really reflect constituent structure in the traditional sense, but that it reflects a particular parse process. This may be true for categorial systems in general, but it is particularly clear for the P-calculus. Consider for instance how one would parse a sentence like (7) using a shift-reduce parsing technique, and having only right- and left-application as syntax rules.

(14)	(remaining) input	stack
	s/vp,vp/np,np,vp\vp	_
S	vp/np,np,vp\vp	s/vp
S	np,vp\vp	s/vp,vp/np
S	vp\vp	s/vp,vp/np,np
R	vp\vp	s/vp,vp
S	_	s/vp,vp,vp\vp
R	_	s/vp,vp
R	_	s

Shifting an element onto the stack (apart form the first one maybe) seems to be equivalent to combining elements by means of I. The stack is after all nothing but a somewhat different representation of the product types we used earlier. The fact that adding one element to the stack (**vp\vp**) induces two reduction steps, is comparable to the fact that the inference rule P may have the effect of eliminating more than one slash at time.

The similarity between shift-reduce parsing and the derivations in P brings in another interesting aspect. The shift-reduce algorithm is a correct parsing strategy, because it will produce all (syntactic) ambiguities for a given input string. This means that in the example above, a shift-reduce parser would only produce one syntax tree (assuming that the grammar has only application).

If the input was potentially ambiguous, as in (15), there are two different derivations.

(15) a/a a a\a

It is after shifting **a** on the stack that a difference arises. Here, one can either reduce or shift one more step. The first choice leads to the left-branching derivation, the second to the right-branching one.

The choice between shifting or reducing has a categorial equivalent. In the P-calculus, one can either produce a left-branching derivation tree for (15) by using application only, or as indicated in (16).

(16) a/a a a\a

_____I

a/a*a

_____P

a

Note that the P-calculus thus is able to find genuine syntactic (or potentially semantic) ambiguities, without producing a different branching phrase structure. The correspondence to shift-reduce parsing already suggests this of course, since we should consider the phrase structure produced by a structurally complete grammar much more as a record of the parse process than as a constituent structure in the traditional sense.

6. Coordination

The P-calculus is structurally complete, and therefore, all the arguments that have been presented in favour of a categorial analysis of coordination, hold for the P-calculus as well. Coordination introduces polymorphism in the grammar, however, and this leads to some complications for the restricted P-calculus presented in (12).

Adding a category **X\(X/X)** for coordinators to the P-calculus, enables us to handle non-constituent conjunction, as is exemplified in (17) and (18).

(17) John loves and Peter adores Sue

s/vp vp/np X\(X/X) s/vp vp/np np

_____ _____I

s/vp*vp/np s/vp*vp/np

_____

s/vp*vp/np

_____P

s

(18) J. loves Mary madly and Sue wildly

vp/np np vp\vp X\(X/X) np vp\vp

_____ _____

np*vp\vp np*vp\vp

_____

np*vp\vp

_____P'

vp

The restricted-calculus of (12) was designed to enable efficient left-associative parsing. We assumed that lexical categories would always be product-free, but this assumption no longer holds, if we add **X\(X/X)** to the grammar (since **X** can be instantiated, for instance as **s/vp*vp/np**). This means that left-associative derivations are not always possible for coordinated sentences.

Our solution to this problem, is to add rules such as (19) to the grammar, which can transform certain product-categories into product-free categories.

(19) A/(B*C) ==> (A/C)/B

A number of such rules are needed to restore left-associativity.

Next to syntactical additions, some modifications to the semantic part of the inference rule P had to be made, in order to cope with the polymorphic semantics proposed for coordination by Partee & Rooth (1983).

7. Concluding remarks.

The spurious ambiguity problem has been solved in this paper in a rather paradoxical manner. Whereas Wittenburg (1987) tries to do away with ambiguous phrase structure as much as possible (it only arises where you need it) and Pareschi & Steedman (1987) use a chart parsing technique to recover implicit constituents efficiently, the strategy in this paper has been to go for complete ambiguity. It is in fact this massive ambiguity, which trivializes constituent structure to such an extent that one might as well ignore it, and choose a constituent structure that fits ones purposes best (left-branching in this case). It seems that as far as processing is concerned, the half-way flexible systems of Steedman (having generalized composition, and heavily restricted forms of raising) are in fact the hardest case. Simple AB-grammars are in all respects similar to CF-grammars, and can directly be parsed by any bottom-up algorithm. For strong structurally complete systems such as P, spurious ambiguity can be eliminated by inspecting left-branching trees only. For flexible but not structurally complete systems, it is much harder to predict which derivations are interesting and which ones are not, and therefore the only solution is often to inspect all possibilities.

References

Ades, Antony & Mark Steedman (1982), On the Order of Words, *Linguistics & Philosophy 4*, 517-518.

Buszkowski, Wojciech (1988). Generative Power of Categorial Grammars. In R. Oehrle, E. Bach, and D. Wheeler (eds.), *Categorial Grammars and Natural Language Structures*, Reidel, Dordrecht, 69-94.

Dowty, David (1988), Type Raising, Functional Composition, and Non-constituent Conjunction. In R. Oehrle, E. Bach, and D. Wheeler (eds.), *Categorial Grammars and Natural Language Structures*, Reidel, Dordrecht, 153-197.

Lambek, Joachim (1958), The mathematics of sentence structure. *American Mathematical Monthly 65*, 154-170.

Moortgat, Michael (1987), Lambek Categorial Grammar and the Autonomy Thesis. *INL working papers 87-03*.

Moortgat, Michael (1988), *Categorial Investigations : Logical and Linguistic Aspects of the Lambek Calculus*. Dissertation, University of Amsterdam.

Pareschi, Remo and Mark Steedman (1987), A Lazy Way to Chart-Parse with Categorial Grammars. *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford University, 81-88.

Partee, Barbara and Mats Rooth (1983), Generalized Conjunction and Type Ambiguity. In R. Bäuerle, C. Schwarze and A. von Stechow (eds.) *Meaning, Use, and Interpretation of Language*, de Gruyter, Berlin, 361-383.

Steedman, Mark (1985), Dependency and Coordination in the Grammar of Dutch and English. *Language 61*, 523-568.

Wittenburg, Kent (1986), *Natural Language Parsing with Combinatory Categorial Grammars in a Graph-Unification-Based Formalism*. Ph. D. dissertation, University of Texas at Austin.

Wittenburg, Kent (1987), Predictive Combinators: A Method for Efficient Processing of Combinatory Categorial Grammars. *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford University, 73-80.

Zwarts, Frans (1986), *Categoriale Grammatica en Algebraische Semantiek*, Dissertation, State University Groningen.