The Complexity of Parsing with Extended Categorial Grammars

Esther König, Universität Stuttgart, Institute for Computational Linguistics,
Keplerstrasse 17, D-7000 Stuttgart 1, FRG, koenig@ds0lilog.bitnet

Abstract

Instead of incorporating a gap-percolation mechanism for handling certain "movement" phenomena, the extended categorial grammars contain special inference rules for treating these problems. The Lambek categorial grammar is one representative of the grammar family under consideration. It allows for a restricted use of hypothetical reasoning. We define a modification of the Cocke-Younger-Kasami (CKY) parsing algorithm which covers this additional deductive power and analyze its time complexity.

1 Introduction

Categorial grammars have become attractive in the last decade. One reason might be the rediscovery of their conceptual elegance: the combinatory potential of a lexeme is stated in direct association with the lexeme itself - in the lexicon. A *basic (bidirectional) categorial grammar* **B** is defined by a set of categories $C := C_0 \cup \{z \,|\, z = x/y \; or \; z = x\backslash y; \; x, y \in C\}$ (C_0 a finite set of ***basic categories***, the category y is referred to as the ***argument category***, the category x is called ***value category***, complex categories are named ***functor categories***), a ***goal category*** g (the start symbol) which is a basic category, a ***lexicon*** L which is a function from a finite set of lexemes onto a set of finite sets of categories, and the two *combination rules* "leftward application" ($app_\backslash$) and "rightward application" ($app_/$) which state how argument positions are filled:

$(app_\backslash)\quad y,\; x\backslash y \rightarrow x$
$(app_/)\quad x/y,\; y \rightarrow x$

An object $U \rightarrow x$ where U is a *sequence* of categories is called a *sequent*.

This basic concept of categorial grammar may be extended by adding more combinatory rules. E.g. the rule of *functional composition* is the incarnation of the idea how to handle certain phenomena of unbounded dependencies: If a functor category x/y finds only an incomplete argument category y/z, i.e. which, itself, still lacks an argument z, then these two partial categories can be, nevertheless, combined into the category x/z, which expresses the fact that z is still missing.

$(fc_\backslash)\quad y\backslash z, x\backslash y \rightarrow x\backslash z$
$(fc_/)\quad x/y, y/z \rightarrow x/z$

In general, functional composition cannot do its job by itself. One needs rules which allow for changing the order in which a functor category takes its left arguments with regard to its right arguments. The use of the *type raising* rules is one way to obtain this goal:

$(tr_\backslash)\quad y \rightarrow x\backslash(x/y)$
$(tr_/)\quad y \rightarrow x/(x\backslash y)$

The first extensive use of the concept of functional composition in syntax appears in the paper [Ades, Steedman 1982]. The rules of functional composition and type raising have already been mentioned in [Lambek 1958] as being theorems of the Lambek calculus.

To obtain a Lambek categorial grammar **L** from a basic categorial grammar, one has to add the two rules of *functional abstraction* (T is a non-empty sequence of categories):

$$(abstr_\backslash)\quad \frac{y,\, T \rightarrow x}{T \rightarrow x\backslash y}$$
$$(abstr_/)\quad \frac{T,\, y \rightarrow x}{T \rightarrow x/y}$$

The rule ($abstr_\backslash$) reads: *If* x can be derived from the category sequence T under the assumption that y has been put in front of T *then* it holds that $x\backslash y$ is derivable from T alone. An example of an L-derivation in a somewhat abbreviated notation[1] is given in figure 2[2]. This analysis is an adaptation of ideas due to [Hepple 1990a].

In the remainder of the paper, we first give a brief overview on other approaches to parsing with extended categorial grammars. Then the modified CKY-algorithm is presented.

2 General Processing Issues

In connection with parsers for extended categorial grammars, the problem of "spurious ambiguities" or derivational equivalence has appeared in a massive way: One syntactic (or semantic) reading of a sentence can be derived in many, many ways. There have been several proposals in the literature how to tackle this problem. The first group of approaches uses repeated equivalence tests such that the spurious ambiguities stay local and do not carry over

[1]Only the categories on the righthand sides of the $\rightarrow$-signs are shown.

[2]Ignore the extra decoration in this figure for a moment.

globally to the final parse results ([Karttunen 1989], [Hepple, Morrill 1989]). But the presence of local spurious ambiguities means that still some superfluous work has to be done by the parser. The second group of methods uses top-down information to avoid equivalent derivations completely. This has been illustrated for the method of *predictive combination* in [Wall, Wittenburg 1989] where, instead of functional composition, rules like the following one are used:

$$(pfc_{/}) \quad x_1/(x_2/x_3), x_2/x_4 \rightarrow x_1/(x_4/x_3)$$

This method has the shortcoming of changing the set of derivable sequents with regard to the original rule set. Check e.g. the sequent $x_1/(x_2/x_3)$, $x_2/(x_4/x_5)$, x_4/x_6, $(x_6/x_5)/x_3 \rightarrow x_1$.

Since those extended categorial grammars which are based on the explicit use of functional composition and type raising seem to be reluctant to leave behind the problem of spurious ambiguity, we have chosen the Lambek calculus as a different setup for an extended categorial grammar where those rules are theorems but not axioms. In the Lambek calculus, the problem of spurious ambiguities disappears since one can define a normal form for derivations which can be carried out on the fly (cf. [Hepple 1990b], [König 1989]). One essential characteristic of this normal form is the following: If there is a choice between using an application rule and an abstraction rule, the abstraction rule is preferred. Another requirement is that no spontaneous uses of the abstraction rules do occur (cf. Prawitz normal form, [Prawitz 1965]). This means that an abstraction rule can only be triggered by lexical material.

3 A chart parser for extended categorial grammar

The CKY-algorithm ([Aho, Ullman 1972]) works bottom-up. It uses a chart of well-formed substrings in order to avoid redundant work. By defining a parser for the basic categorial grammars, we introduce our particular representation of chart parsing.

3.1 Chart parsing with a basic categorial grammar

A chart is a set of items. An *item* is a triple $[x, i, j]$ which states the fact that the category x has been derived from the continuous part of the input string between the positions i and j. We use a slightly alienated sequent notation where the lefthand side of a sequent is the current chart, and the righthand side is the goal category g of the grammar. The $\bullet$-sign marks the current scanning position.

Figure 1 shows the processing steps which can be performed with a chart. The rule (*axiom*) describes the conditions for successful termination of the parsing process: The chart has been processed until its end and there exists an item which covers the whole input string of length n. The rule (*scan*) replaces a lexeme in the input string by one of the categories it is assigned to in the lexicon. The rule (*complete*$_\backslash$) implements leftward functional application: a leftward looking functor which follows immediately its argument causes the addition of a new item with the value category. For rightward functional application, this works symmetrically. The inference rule-style representation of our algorithm abstracts away from details dealing with the appropriate control mechanism which guarantees that e.g. all possible (*complete*$_\backslash$)-steps are performed with one item.

The completer step is the part of the CKY-algorithm which determines its cubic time complexity. The applications of the rules (*complete*$_\backslash$) and (*complete*$_/$) stay within the same time bound because the search for a pair of reducible items works as in the original algorithm. In particular, the number of items which span the same section of the input string has a constant bound which is the number of subcategories of the categories occuring in the lexicon. This number is proportional to the size of the finite lexicon.

(L lexicon; w lexeme; α string; I set of items)

(*axiom*)

$$I, \begin{bmatrix} z \\ 0 \\ n \end{bmatrix} \bullet \rightarrow z$$

(*scan*)

$$\frac{x \in L(w) \qquad I, \begin{bmatrix} z \\ i \\ i+1 \end{bmatrix} \bullet \alpha \rightarrow z}{I \bullet w, \alpha \rightarrow z}$$

(*complete*$_\backslash$)

$$\frac{I, \begin{bmatrix} y \\ i \\ j \end{bmatrix}, \begin{bmatrix} x\backslash y \\ j \\ k \end{bmatrix}, \begin{bmatrix} x \\ i \\ k \end{bmatrix} \bullet \alpha \rightarrow z}{I, \begin{bmatrix} y \\ i \\ j \end{bmatrix}, \begin{bmatrix} x\backslash y \\ j \\ k \end{bmatrix} \bullet \alpha \rightarrow z}$$

(*complete*$_/$)

$$\frac{I, \begin{bmatrix} x/y \\ i \\ j \end{bmatrix}, \begin{bmatrix} y \\ j \\ k \end{bmatrix}, \begin{bmatrix} x \\ i \\ k \end{bmatrix} \bullet \alpha \rightarrow z}{I, \begin{bmatrix} x/y \\ i \\ j \end{bmatrix}, \begin{bmatrix} y \\ j \\ k \end{bmatrix} \bullet \alpha \rightarrow z}$$

Figure 1: A chart-parser $\mathbf{B}_C$ for basic categorial grammars

3.2 Extending the basic chart parser

In the traditional concept of a chart every item is visible to every other item in the chart. For the Lambek calculus, general accessability is inadequate because of its use of hypothetical reasoning: The abstraction rules assert hypothetical constituents which must only be used in the subproof for the premise sequent. Thus, the interaction of items must be regulated in a more fine-grained way.

Fig. 3 shows a section of an L-derivation where a use of an abstraction rule occurs[3]. The complex argument of the *emitter* category

$x/[\ldots(y/x_{2q})\backslash x_{1p})\ldots\backslash x_{11})/x_{21}]$

triggers the act of asserting its own arguments as hypothetical categories (in a fixed order) in addition to a given non-empty category sequence $x_i \ldots x_j$. We call the sequence of hypothetical categories which are asserted on the lefthand side the *left mini-chart*, its symmetric counterpart is a *right mini-chart*. A subproof is carried out on the basis of these additional premise categories. If this subproof yields a category y, then x spans the whole sequence of premises under consideration.

To perform an abstraction rule, obviously an intermingling of top-down and bottom-up directed actions is needed. The sequence $x_i \ldots x_j$, i.e. the positions where the mini-charts are *attached*, has to be chosen non-deterministically. Instead of asserting copies of the mini-charts at all positions in the current chart, it is more convenient to put the mini-charts apart and to keep them accessible from all chart positions. An adapted completer rule will built additional items on the basis of the extended chart. Since there might be several emitter categories in the original sequent, it is important to provide the asserted mini-charts with enough information in order to control their use. In particular, *attachment chains* can arise because mini-charts can be attached to other mini-charts.

We assume that each value category of a complex argument of an emitter is marked with a unique number m. Items also have unique numbers a. A (left) index of an item is now a quadruple $\langle t, m, i, p\rangle$ where t stands for the type of mini-chart the item belongs to: `l` (left), `r` (right), or `n` (none); m is the number of the subcategory which caused the assertion of this item; i is a position number relative to the (mini-)chart with type t and number m; p is the number of an item in another part of the chart where the mini-chart is attached to. For items which have been derived on the basis of the items given by the input string, $t =$ `n`, $m = 0$, and $p = 0$.

In figure 4, the extended algorithm is presented. From the following discussion, various details and, in particular, a proof of correctness, are omitted due to the lack of space. The rule ($compl_\backslash s$) is roughly equivalent to ($complete_\backslash$) in the basic algorithm: two items which are adjacent in the *same* piece of the chart are combined. The rule ($compl_\backslash r$) allows an item at the beginning of a right mini-chart to combine with some other item with number $a1$ on its left. The function `newlast` adds the information about this new attachment point at the end of the right attachment chain of item $a2$ (see item 6 in figure 2 where the item id 0 is replaced by the id 5 because of a ($compl_\backslash r$)-step which combines items 5 and 9). In order to avoid cycles, the intersection of the mini-chart numbers which are found by traversing all the four attachment `chain`'s has to be empty. The work of the abstraction rule ($abstr_\backslash$) is split into the two sub-tasks: The rule ($emit_\backslash$) performs the assertion of the mini-charts. The rule ($disc_\backslash$) ("discharge") is a specialization of the completer rule for functor categories which are emitters. The conditions in the rule guarantee the following: Both mini-charts which are due to the current emitter must have been used completely in deriving y. The right index of the attachment point $p4$ of y (determined by a function `ri`) must be equal to the left index of the emitter. The value category x must not bear any information about the involvement of the current mini-charts in its derivation. This is achieved by going back to the attachment point of the left mini-chart.

Since each of the $O(n)$ mini-charts for an input string of length n can only be used once in a derivation, and since we currently do not know any reasonable restrictions, we assume that any one of the $O(n!)$ permutations of the mini-charts possibly can be used in a specific derivation. For every item in the initial chart, $O(n!)$ completer-steps can be possible. This means that the whole parser has a time complexity of $O(n^2 \times n!)$. In order to evaluate the result for the given algorithm, one has to be aware of the fact that this parsing procedure can handle n-fold extraction from phrases - to speak linguistically.

If we restrict ourselves to single extraction (this can be implemented by checking the length of the attachment chain before performing ($compl_\backslash r$)) then the time complexity reduces drastically: The formula $n!/(n-k)!$ for the variations of length k of a set of length n equals n for $k = 1$. Thus, the time complexity of the algorithm is $O(n^3)$. If we want to describe two-fold extraction like in figure 2, i.e. use a "2-restricted version of $\mathbf{L}_C$", the algorithm needs $O(n^4)$ steps for deriving an input string of length n.

Conjecture: 1-restricted $\mathbf{L}_C$ covers the rules of functional composition and type raising, whereas 2-restricted $\mathbf{L}_C$ suffices to derive the rules of predictive combination.

4 Conclusion

We have presented an extension of the CKY-algorithm for an instance of the family of extended categorial grammars which uses hypothetical reasoning, i.e. for the bidirectional Lambek categorial grammars. Although the current algorithm has an undesirable time complexity in its general case, one can define restricted versions which cover the kind of structures found in linguistic examples with time complexity $O(n^3)$ or $O(n^4)$ depending on the fact

[3]In the following, symmetric cases are mostly omitted.

whether one wishes to describe single or double extraction from phrases. It is an interesting question whether better algorithms for the general case can be found.

There are many ways to improve this new algorithm concerning its "average complexity" in practical applications. For example, the search for a partner item in a completer-step could be reduced by using appropriate heuristics. Further increases in efficiency can be expected, if one allows for arbitrary embedding of disjunctions and slashes in categories (cf. [Benthem 1989], [Morrill 1989]) instead of using the flat set notation which amounts to a disjunctive normal form of categories in the lexicon.

Theoretical complexity results for a group of categorial grammars reaching beyond context-free languages are known, as well. From the fact that certain extended versions of categorial grammars (CCG) are equivalent with the Tree Adjoining Grammars (TAG) ([Weir, Joshi 1988]), the $O(n^4 \log n)$-time complexity of TAG-parsers ([Harbusch 1989]), in principle, carries over to CCG.

The connection of the propositional grammar calculi as discussed above with a unification device for first order terms is done by taking care of the variable bindings in the hypothetical categories which are asserted by the abstraction rules. The recognition problem is still decidable (cf. [Pareschi 1988]) although surely not well-behaved as this is the case for all non-propositional grammars. For example, the recognition problem for a basic categorial grammar with feature unification is NP-complete ([Morrill 1988]). Higher order versions of categorial grammars like the ones being produced in the CUG/UCG-frameworks ([Uszkoreit 1986],[Zeevat et al. 1986]) where variables can range over categories are in general undecidable because higher order unification itself is undecidable.

5 Acknowledgements

The research reported in this paper is supported by the LILOG project, and a doctoral fellowship, both from IBM Deutschland GmbH, and by the Esprit Basic Research Action Project 3175 (DYANA). I thank Andreas Eisele for discussion. The responsibility for errors resides with me.

References

[Ades, Steedman 1982] Anthony E. Ades, Mark J. Steedman. On the order of words. *Linguistics and Philosophy*, 4:517–558, 1982.

[Aho, Ullman 1972] Alfred V. Aho, Jeffrey D. Ullman. *The Theory of Parsing, Translation and Compiling.* Prentice Hall, 1972.

[Benthem 1989] Johan van Benthem. Categorial grammar and type theory. *Linguistics and Philosophy*, 1989.

[Harbusch 1989] Karin Harbusch. *Effiziente Strukturanalyse natürlicher Sprache mit Tree Adjoining Grammars (Efficient Parsing of Natural Language with Tree Adjoining Grammars).* PhD thesis, Universität des Saarlandes, Saarbrücken, 1989.

[Hepple 1990a] Mark Hepple. Grammatical relations and the Lambek calculus. In *Proc. of the Symposium on Discontinous Constituency*, Tilburg, The Netherlands, Institute for Language Technology and Artificial Intelligence, 1990.

[Hepple 1990b] Mark Hepple. Normal form theorem proving for the Lambek calculus. *This volume.*

[Hepple, Morrill 1989] Mark Hepple, Glyn Morrill. Parsing and derivational equivalence. In *Proc. EACL* , Manchester, 1989.

[Karttunen 1989] Lauri Karttunen. Radical lexicalism. In Mark Baltin, Anthony Kroch, eds., *Alternative Conceptions of Phrase Structure*, University of Chicago Press, 1989.

[König 1989] Esther König. Parsing as natural deduction. In *Proc. ACL* , Vancouver, B.C., 1989.

[Lambek 1958] Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170, 1958.

[Morrill 1988] Glyn Morrill. *Extraction and Coordination in Phrase Structure Grammar and Categorial Grammar.* PhD thesis, University of Edinburgh, Centre for Cognitive Science, Edinburgh, Scotland, 1988.

[Morrill 1989] Glyn Morrill. *Grammar as Logic.* Technical Report EUCCS/RP-34, Centre for Cognitive Science, University of Edinburgh, Edinburgh, Scotland, 1989.

[Pareschi 1988] Remo Pareschi. A definite clause version of categorial grammar. In *Proc. ACL* , Buffalo, N.Y., 1988.

[Prawitz 1965] Dag Prawitz. *Natural Deduction.* Alqvist and Wiksell, Uppsala, 1965.

[Uszkoreit 1986] Hans Uszkoreit. Categorial unification grammars. In *Proc. COLING* , Bonn, 1986.

[Wall, Wittenburg 1989] Robert E. Wall, Kent Wittenburg. Predictive normal forms for function composition in categorial grammars. In *Proc. of the International Workshop on Parsing Technologies*, Carnegie Mellon University, 1989.

[Weir, Joshi 1988] D.J. Weir, A.K. Joshi. Combinatory categorial grammmars: Generative power and relationship to linear context-free rewriting systems. In *Proc. ACL* , Buffalo, N.Y., 1988.

[Zeevat et al. 1986] Henk Zeevat, Ewan Klein, Jo Calder. *Unification Categorial Grammar.* Technical Report EUCCS/RP-21, Centre for Cognitive Science, University of Edinburgh, Edinburgh, Scotland, 1986.

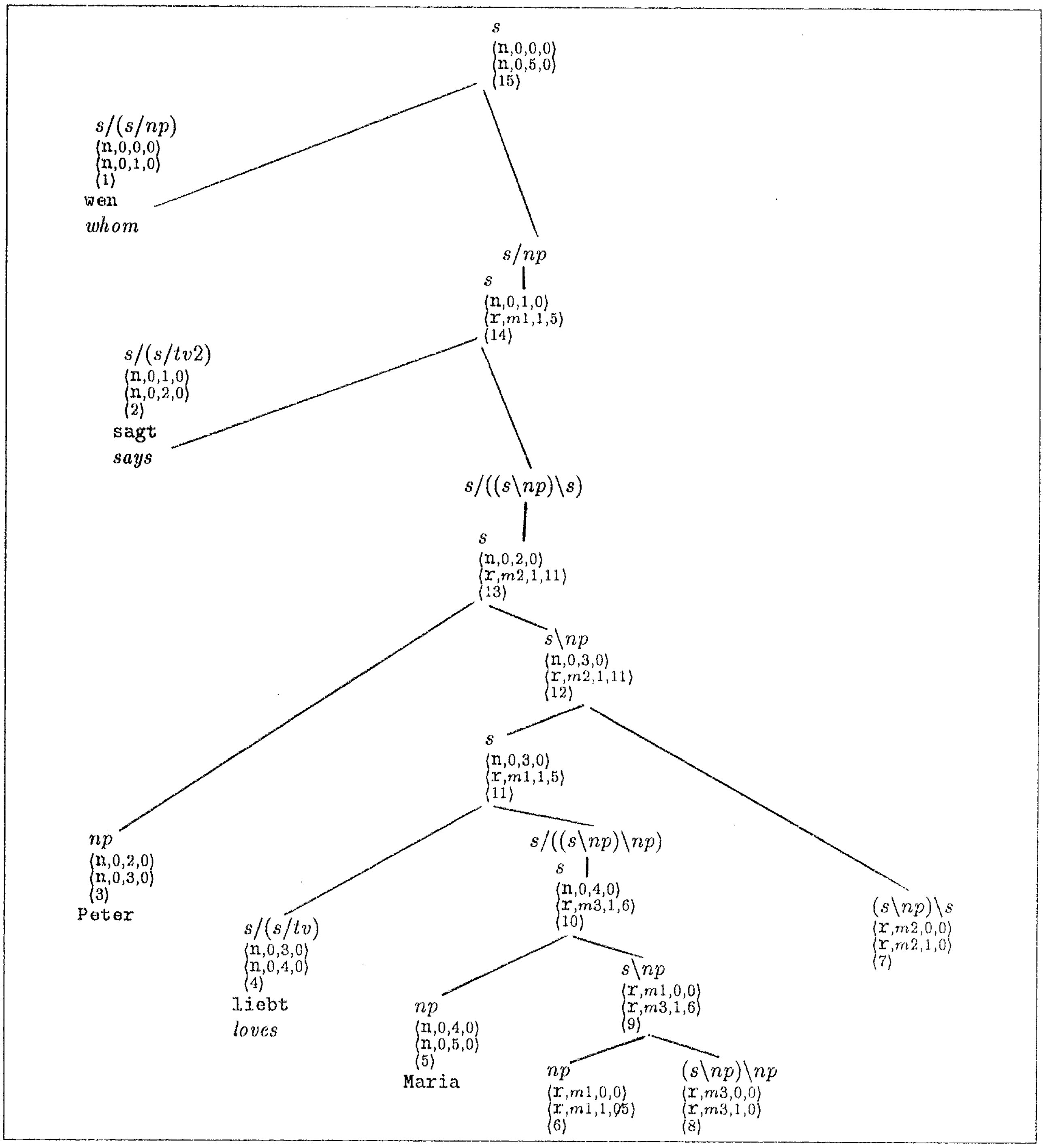

Figure 2: Long distance dependency [$tv = (s\backslash np)\backslash np$, $tv2 = (s\backslash np)\backslash s$]

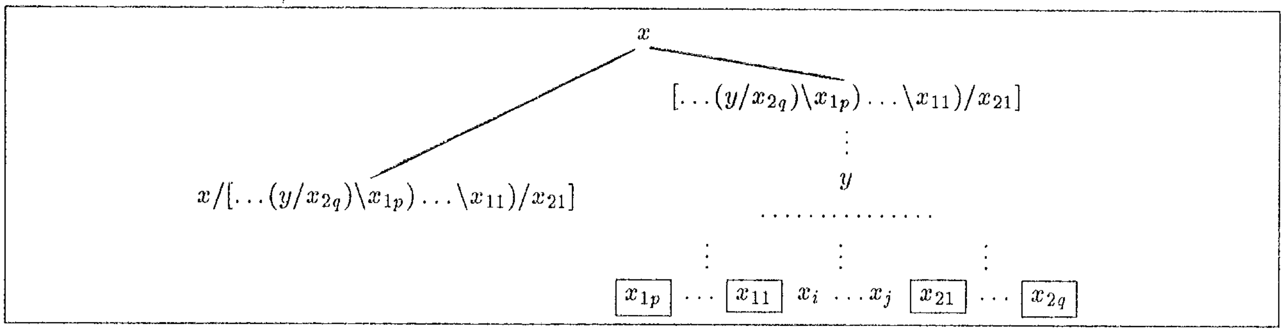
x
$[\ldots(y/x_{2q})\backslash x_{1p})\ldots\backslash x_{11})/x_{21}]$
$x/[\ldots(y/x_{2q})\backslash x_{1p})\ldots\backslash x_{11})/x_{21}]$
y
$\boxed{x_{1p}}\ \ldots\ \boxed{x_{11}}\ x_i\ \ldots x_j\ \boxed{x_{21}}\ \ldots\ \boxed{x_{2q}}$

Figure 3: Schema for the use of the abstraction rules

$$(\mathit{axiom}) \qquad I, \begin{bmatrix} (z)_{\langle a \rangle} \\ \langle \mathtt{n},0,0,0 \rangle \\ \langle \mathtt{n},0,n,0 \rangle \end{bmatrix} \bullet \longrightarrow z$$

$$(\mathit{scan}) \qquad \frac{x \in L(w) \qquad I, \begin{bmatrix} (x)_{\langle a \rangle} \\ \langle \mathtt{n},0,i,0 \rangle \\ \langle \mathtt{n},0,i{+}1,0 \rangle \end{bmatrix} \bullet \alpha \longrightarrow z}{I \bullet w, \alpha \longrightarrow z}$$

$$(\mathit{compl}\backslash s) \qquad \frac{\begin{array}{l} p2 = 0 \vee p3 = 0 \\ I, \begin{bmatrix} y \\ \langle t1,m1,i1,p1 \rangle \\ \langle t2,m2,i2,p2 \rangle \end{bmatrix}, \begin{bmatrix} x \backslash y \\ \langle t2,m2,i2,p3 \rangle \\ \langle t4,m4,i4,p4 \rangle \end{bmatrix}, \begin{bmatrix} x \\ \langle t1,m1,i1,p1 \rangle \\ \langle t4,m4,i4,p4 \rangle \end{bmatrix}, \bullet \alpha \longrightarrow z \end{array}}{I, \begin{bmatrix} y \\ \langle t1,m1,i1,p1 \rangle \\ \langle t2,m2,i2,p2 \rangle \end{bmatrix}, \begin{bmatrix} x \backslash y \\ \langle t2,m2,i2,p3 \rangle \\ \langle t4,m4,i4,p4 \rangle \end{bmatrix} \bullet \alpha \longrightarrow z}$$

$$(\mathit{compl}\backslash l) \qquad \frac{\begin{array}{l} \mathtt{chain}(p1) \cap \mathtt{chain}(p2) \cap \mathtt{chain}(p3) \cap \mathtt{chain}(p4) = \{\} \\ \mathtt{newlast}(p1, a2) \\ I, \begin{bmatrix} (y)_{\langle a1 \rangle} \\ \langle t1,m1,i1,p1 \rangle \\ \langle \mathtt{l},m2,0,p2 \rangle \end{bmatrix}, \begin{bmatrix} (x \backslash y)_{\langle a2 \rangle} \\ \langle t3,m3,i3,p3 \rangle \\ \langle t4,m4,i4,p4 \rangle \end{bmatrix}, \begin{bmatrix} (x)_{\langle a3 \rangle} \\ \langle t1,m1,i1,p1 \rangle \\ \langle t4,m4,i4,p4 \rangle \end{bmatrix}, \bullet \alpha \longrightarrow z \end{array}}{I, \begin{bmatrix} (y)_{\langle a1 \rangle} \\ \langle t1,m1,i1,p1 \rangle \\ \langle \mathtt{l},m2,0,p2 \rangle \end{bmatrix}, \begin{bmatrix} (x \backslash y)_{\langle a2 \rangle} \\ \langle t3,m3,i3,p3 \rangle \\ \langle t4,m4,i4,p4 \rangle \end{bmatrix}, \bullet \alpha \longrightarrow z}$$

$$(\mathit{compl}\backslash r) \qquad \frac{\begin{array}{l} \mathtt{chain}(p1) \cap \mathtt{chain}(p2) \cap \mathtt{chain}(p3) \cap \mathtt{chain}(p4) = \{\} \\ \mathtt{newlast}(p4, a1) \\ I, \begin{bmatrix} (y)_{\langle a1 \rangle} \\ \langle t1,m1,i1,p1 \rangle \\ \langle t2,m2,i2,p2 \rangle \end{bmatrix}, \begin{bmatrix} (x \backslash y)_{\langle a2 \rangle} \\ \langle \mathtt{r},m3,0,p3 \rangle \\ \langle t4,m4,i4,p4 \rangle \end{bmatrix}, \begin{bmatrix} (x)_{\langle a3 \rangle} \\ \langle t1,m1,i1,p1 \rangle \\ \langle t4,m4,i4,p4 \rangle \end{bmatrix}, \bullet \alpha \longrightarrow z \end{array}}{I, \begin{bmatrix} (y)_{\langle a1 \rangle} \\ \langle t1,m1,i1,p1 \rangle \\ \langle t2,m2,i2,p2 \rangle \end{bmatrix}, \begin{bmatrix} (x \backslash y)_{\langle a2 \rangle} \\ \langle \mathtt{r},m3,0,p3 \rangle \\ \langle t4,m4,i4,p4 \rangle \end{bmatrix}, \bullet \alpha \longrightarrow z}$$

$$(\mathit{emit}\backslash) \qquad \frac{\begin{array}{l} I, \begin{bmatrix} (x_{1p})_{\langle a1p \rangle} \\ \langle \mathtt{l},m3,p,0 \rangle \\ \langle \mathtt{l},m3,p-1,0 \rangle \end{bmatrix}, \ldots, \begin{bmatrix} (x_{11})_{\langle a11 \rangle} \\ \langle \mathtt{l},m3,1,0 \rangle \\ \langle \mathtt{l},m3,0,0 \rangle \end{bmatrix}, \begin{bmatrix} (x_{21})_{\langle a21 \rangle} \\ \langle \mathtt{r},m3,0,0 \rangle \\ \langle \mathtt{r},m3,1,0 \rangle \end{bmatrix}, \ldots, \begin{bmatrix} (x_{2q})_{\langle a2q \rangle} \\ \langle \mathtt{r},m3,q-1,0 \rangle \\ \langle \mathtt{r},m3,q,0 \rangle \end{bmatrix}, \\ \begin{bmatrix} x \backslash [\ldots (y_{\langle m3 \rangle} / x_{2q}) \backslash x_{1p}) \ldots \backslash x_{11}) / x_{21}] \\ \langle t1,m1,i1,p1 \rangle \\ \langle t2,m2,i2,p2 \rangle \end{bmatrix} \bullet \alpha \longrightarrow z \end{array}}{I, \begin{bmatrix} x \backslash [\ldots (y_{\langle m3 \rangle} / x_{2q}) \backslash x_{1p}) \ldots \backslash x_{11}) / x_{21}] \\ \langle t1,m1,i1,p1 \rangle \\ \langle t2,m2,i2,p2 \rangle \end{bmatrix} \bullet \alpha \longrightarrow z}$$

$$(\mathit{disc}\backslash) \qquad \frac{\begin{array}{l} \mathtt{ri}(p4) = \langle t1, m1, i1, p1 \rangle \\ I, \begin{bmatrix} y \\ \langle \mathtt{l},m3,p,p3 \rangle \\ \langle \mathtt{r},m3,q,p4 \rangle \end{bmatrix}, \begin{bmatrix} x \backslash [\ldots (y_{\langle m3 \rangle} / x_{2q}) \backslash x_{1p}) \ldots \backslash x_{11}) / x_{21}] \\ \langle t1,m1,i1,p1 \rangle \\ \langle t2,m2,i2,p2 \rangle \end{bmatrix}, \begin{bmatrix} x \\ \mathtt{li}(p3) \\ \langle t2,m2,i2,p2 \rangle \end{bmatrix} \bullet \alpha \longrightarrow z \end{array}}{I, \begin{bmatrix} y \\ \langle \mathtt{l},m3,p,p3 \rangle \\ \langle \mathtt{r},m3,q,p4 \rangle \end{bmatrix}, \begin{bmatrix} x \backslash [\ldots (y_{\langle m3 \rangle} / x_{2q}) \backslash x_{1p}) \ldots \backslash x_{11}) / x_{21}] \\ \langle t1,m1,i1,p1 \rangle \\ \langle t2,m2,i2,p2 \rangle \end{bmatrix} \bullet \alpha \longrightarrow z}$$

Figure 4: Chart-parser L_C for Lambek categorial grammars (L lexicon; w lexeme; α string; a item id; I set of items)