Parsing Categorial Grammars in Natural Deduction

José Antonio Jiménez Millán
Escuela Superior de Ingenieria de Cadiz
Calle Sacramento, 82
11003 - CADIZ (SPAIN)

Antonio Frias Delgado
Facultad de Filosofia y Letras
Avda. Gomez Ulla, s/n
11003 - CADIZ (SPAIN)

{joseantonio.jimenez,antonio.frias}@uca.es
Universidad de Cadiz, Spain

April 1999

Área temática:
4. Gramáticas y formalismos para el análisis morfológico y sintáctico

Abstract

We present an algorithm for parsing in the product-free fragment of Categorial Grammars (CG) using a Natural Deduction (ND) Calculus *à la* Prawitz. Since this is not the usual formalism of choice, we must first introduce the rules of the calculus and some of the geometric properties of deductions in which lies the parsing process.

The resulting algorithm generates only natural deductions in normal form so that it is devoid of spurious ambiguity. The parsing is quite efficient with some top-down (goal directed) steps followed by bottom-up ones.

1 Introduction

What have we done: We present an algorithm for parsing in the product-free fragment of Categorial Grammars (CG) using the Natural Deduction (ND) Calculus *à la* Prawitz.

Closely following Prawitz [Pra65], we prove the *inversion principle*—as well as other several geometric properties of deductions in the ND version of CG—that allows us to build a quite efficient mixed parser, with some bottom-up (data directed) steps followed by top-down (goal directed) ones. The parser generates only natural deductions in normal form, so that it is devoid of spurious ambiguity.

Why is it important: Gentzen developed the sequent calculus for logic [Gen34], with its *Hauptsatz* property, to cope with the problem that posseses the natural deduction formalism such that it does not admit of a goal-directed algorithm for deductions. So when Lambek presented his Lambek–Gentzen sequent calculus [Lam58], giving a formalism for syntax in a logical format, he used the Gentzen sequent calculus that has become, since then, the formalism of choice for categorial grammars, see [Moo88]. Later Prawitz [Pra65] demonstrated that natural deduction calculus—in normal form—owns one kind of *Hauptsatz* property after all, so that it is possible to build deductions in a top down—goal directed—fashion. Since then several authors have claimed the ND formalism to be more 'natural' for natural language than the sequent's one (see [Kön94], an updated version of [Kön89], and what she calls hypothetical reasoning). Nowadays we can say that there is an awakening of interest in this calculus—see [Ben91], [Kön94], so as [Bus97] and [Moo97] in [BM97]—but there are much less studies about parsing in ND than there are about Lambek–Gentzen systems and, above all, we miss the formal development of the culculus with its properties.

Here we introduce an adapted version of the Natural Deduction Calculus for the product-free fragment of Categorial Grammars, and then we prove the *inversion principle* for this calculus that permits us to follow some goal-directed (top down) steps during parsing.

This is the only complete algorithm we know of, and it is completelly based in the formal study of geometric properties of this ND cal-

culus.

State of the art in ND parsing: As we noted before, most work in categorial grammar deals with proofs in the Lambek-Gentzen sequent formalism. Although some authors have developed one 'normal form' for proof trees in the Lambek-Gentzen sequent calculus and some algorithms to generate only proof trees in that 'normal' form, they are not "directly" related with our work in natural deduction calculus.

The closer works to ours are that of König [Kön94], and that of Carpenter [Car94].

König presents two different algorithms for parsing the product-free fragment of categorial grammars in the natural deduction calculus: A direct one, and another chart-based one. Both two are exclusivelly data–directed (bottom-up) algorithms and they are explained in terms of operations over one kind of stack. Those algorithms are not complete as they can not explain the type-rising rule $[a \Rightarrow (n/a)\backslash n]$—we belive this is a direct consequence of the management of that sort of a stack in a bottom up (from data to conclusion) fashion so that the goal $(n/a)\backslash n$ is not used at all—Moreover, because the individual steps are decomposed in several ones that manage the stack, those algorithms have the drawback of derivational equivalence.

The algorithm of Carpenter is another data–directed one and the user must signal the place he/she wants to introduce a supposition (a gap). Besides it generates non normal proofs that are managed by an independent normalization step.

We have not found any work that developed the geometric properties of natural deductions in a formal way (or at all) except some hints in Johan Van Benthem, Esther König, and others.

2 A Natural Deduction Calculus *à la* Prawitz for CG

In a more or less 'classical' logic, we use to say that in the ND calculus a proof is build by one *deduction* that is defined like an ordered list of formulae. This definition is not appropiate in Categorial Grammars as we need, beside the sequential order of formulae that reflects the order of application of the rules of inference as above, another mechanism to reflect the left-to-right relative position of lexical items (hypotheses) in the sentence as well as the left-to-right relative position of phrasal constituents (intermediate lemas). This mechanism should, also, reflect the resource-sensitivity of this kind of substructural logic, so that resouces can not be reused once consumed.

2.1 Deduction as a two-dimensional structure

We will define one deduction (a proof in ND calculus) as a two-dimensional structure that looks like a tree but that is a rooted directed acyclic graph really, with arcs in sequential left to right order. The relative vertical position, of hypotheses (lexical items) and premisses (phrasal constituents) in the tree, shows the order of application of the rules of inference; but there relative horizontal position reflects the relative left to right order of words in the sentence. But before we needs some definitions.

We define the *fringe* of one deduction tree as the left-to-right ordered sequence of the leaves of the tree. □

A deduction is made up of one only conclusion (the root of the graph) that is obtained out of certain premisses using some deduction rules. Each premiss is obtained, in the same way, out of other premisses using other deduction rules, and so on until we reach the hypotheses. These hypotheses make a left-to-right ordered sequence that we call the *fringe* of the deduction graph.

Each hypothesis B may be in one of two states: it may be *active* — it is the usual state and it indicates that it is an active part of the deduction such that the conclusion A is dependant upon B: $[\alpha, B, \beta \vdash A]$ — or it may be *cancelled* meaning that, from one point on, the conclusion is not dependant upon the cancelled hypothesis (then we say the hypothesis was a *supposition*). We will use the below notation to indicate a deduction (a proof in ND), such that $\alpha \vdash A$, of the conclusion A out of the sequence of hypotheses α that builds the *fringe* of the leaves of the graph:

α

∇

A.

This notation lets us infer certain properties of deductions from the geometric properties of the tree in a rather perspicacious way. Note that α is a sequence of hypotheses in which we care for the left-to-right order of hypotheses as well as for the number of times they occurs.

Base case in the definition of deductions:

A (*hypothesis*)

This is the most simple deduction we can build, meaning that we can deduce the conclusion A just because we have considered it to be a hypothesis.

Introduction rules of logical connectives in the conclusion:

$$\frac{\begin{array}{c}[A]_i\ \gamma \\ \bigtriangledown \\ B\end{array}}{A\backslash B_i}\ (Intro,\backslash) \qquad \frac{\begin{array}{c}\gamma\ [A]_j \\ \bigtriangledown \\ B\end{array}}{B/A_j}\ (Intro,/)$$

The bracketing $[A]$ shows the hypothesis A being cancelled (it is a supposition), while the indexing reflects that the supposition is only active in the *path*[1], $[A_i \ldots\ A\backslash B_i]$ or $[A_j \ldots\ B/A_j]$, between the first occurrence of the index until the second occurrence of the index. Out of this path such hypothesis is cancelled.

The index makes deductions to be a graph and not a real tree. These rules are suitable to use in a goal-directed automatic parser because they fulfil the subformula property[2]—*à la* Gentzen—Premisses are subformulae of the conclusion.

Elimination rules of logical connectives in the conclusion

$$\frac{\begin{array}{cc}\alpha & \beta \\ \bigtriangledown & \bigtriangledown \\ A & A\backslash B\end{array}}{B}\ (Elim,\backslash) \qquad \frac{\begin{array}{cc}\beta & \alpha \\ \bigtriangledown & \bigtriangledown \\ B/A & A\end{array}}{B}\ (Elim,/)$$

These rules do not present the above subformula property as the premiss A does not belong to the conclusion. This is a serious drawback for the implementation of a goal-directed automatic parser for it causes an infinite ramification in the search. Below we will see that there is another kind of 'subformula property' for deductions in normal form that allows us to solve that problem.

In an instance of the $(Elim,\backslash)$ or $(Elim,/)$ rule, we say 'A' to be the *minor premiss*. We say the other premisses—of these and other rules— to be the *major premisses*. In the case of the elimination rules, major premisses are of the form '$A\backslash B$' or 'B/A'. We will show below that we can impose several restrictions to the form of major and minor premisses of deductions in normal form.

[1] Below we will define in a formal way what is meant to be a 'path' between two given formulae.

[2] We will see below that there is another kind of subformula property for deductions in normal form.

In a natural deduction tree Π, we define a *thread* as the ordered sequence of occurrence of formulae $[F_1, F_2, \ldots, F_n]$, taken from that tree, such that: 1) F_1 is one of the leaves of the tree—one hypothesis of the proof— 2) F_n is the root of the tree—the conclusion of the proof— 3) $\forall_{i\in[1\ldots n)} F_i$ is one premiss in an instance of some deduction rule such that F_{i+1} is its conclusion.

In the natural deduction tree Π we define a *branch* as the initial segment $[F_1, F_2, \ldots, F_m]$ of a *thread* such that: 1) F_m is the first formula in the *thread* that plays the role of a minor premiss in an instance of the aplication of some elimination rule. 2) Otherwise F_m should be the root of the tree—the conclusion in the deduction tree—in the case there is no minor premiss in the thread. We say a *branch* that is also a *thread* to be a *main branch.*

We define a *path* between the occurrence of two formulae 'A' and 'B' in a deduction tree Π, as the initial segment $[A = F_1, F_2, \ldots, F_m]$ of the *thread* $[A = F_1, F_2, \ldots, F_m, F_{m+1} = B, \ldots, F_n]$. Notice that we can not always find a path between whatever two formulae in a deduction.

Conditions for the application of the rules: We should emphasize that: 1) **Each and every hypothesis can be used only once** and that **they may be cancelled only once.** 2) We can only operate with hypotheses that are consecutive in the horizontal ordering and that are at the very same vertical height. 3) Hypotheses makes up a sequence —the fringe of the rooted directed acyclic graph with ordered arcs— Hence we care for the left to right order of the hypotheses as well as for the number of times they occurs.

3 Normalization. The inversion principle

Given a deduction tree, Π, such that $\alpha \vdash_{ND} A$, we can introduce in it an endless chain of introduction and elimination rules that yields the very same conclusion A out of the very same hypotheses α. We say such a family of deductions to be equivalent. Among them there is a simplified one that we say is in *normal form.* This is an instance of the more general inversion principle that states that, in deductions of the form:

$$\dfrac{\dfrac{\beta\ [A]_i}{\dfrac{\nabla}{B}}\ (Intro,/) \quad \dfrac{\alpha}{\dfrac{\nabla}{A}}}{B}\ (Elim,/)$$ it is implicit a previous proof of B in the proofs of the premises $\dfrac{\vdots}{B/A}$ and $\dfrac{\vdots}{A}$. Such deductions may be reduced to the simplified following form: $\dfrac{\dfrac{\alpha}{\dfrac{\nabla}{\beta}}\ \dfrac{\nabla}{A}}{B}$ so that there is no introduction rule followed by an elimination one. And analogously for the symmetric (\) operator.

The inversion principle asses that we do not obtain any gain if we infer the formula $B|A$ as the conclusion of one rule of introduction, and then we use that very same formula $B|A$ as the major premiss in an elimination rule. This drives to the following theorem.

THEOREM 1 Inversion Theorem: *If* $\dfrac{\gamma}{\dfrac{\nabla}{A}}$ $[\gamma \vdash_{ND} A]$, *then there is a deduction tree* $\dfrac{\gamma}{\dfrac{\nabla}{A}}$ *such that no (occurrence of one) formula is at the same time the conclusion of an introduction rule and the major premiss of an elimination rule.*

The proof is based upon the inversion principle above. □

Hence, we get a family of equivalent deductions with only one normal form. Normalization fulfil the Church-Rosser property, thus normal forms are unique.

4 Some properties of deductions in normal form

Deductions in normal form own a very peculiar structure that we may take advantage of for building an automatic parser.

THEOREM 2 *Given a deduction tree* Π, *in normal form, and one sequence of formulae* $\beta = [F_1, F_2, \ldots, F_n]$ *that makes a* branch *in* Π, *then there exists one formula* $F_i \in \beta$*—that we will call the* **minimum formula in** β*—that splits the sequence* β *in two (may be empty) parts called the* elimination part *and the* introduction part. *These parts are such that: 1) Each formula* $F_{j\in[1\ldots i)}$ *from the* Elim–part *is the major premiss in some* $(Elim,\star)$ *rule and it contains* F_{j+1} *as a subformula. 2) If* $i \neq n$ *then* F_i *is the premiss of one* $(Intro,\star)$ *rule. 3) Each formula* $F_{j\in(1\ldots n)}$ *from the* Intro–part *(with the exception of the last one) is the premiss of some* $(Intro,\star)$ *rule and it is a subformula of* F_{j+1}.

Thus, each *branch* in a deduction in normal form may be splitted into one (may be empty) sequence of elimination rules in which conclusions are more and more simple—with less and less conectors—followed by another (may be empty) sequence of introduction rules in which formulae are more and more complex—with more and more connectors—. The formula that separates both parts is called the *minimum formula* in the branch and owns the more simple structure of the branch so that it is a subformula of every formulae above and it is also a subformula of every formula below.

Proof: Let us remind first that, by definition, no formula in the branch (with the possible exception of the last one) may be a minor premiss in one elimination rule. Then let us observe that, in any branch β that belongs to a deduction in normal form, the formulae that play the role of major premiss of an elimination rule must precede those formulae that are premisses in any introduction rule. If this was not the case we will find us with one situation that is subject of normalization and hence it is in a contradiction with our initial supposition of the deduction to be in normal form.

Thus, let us consider the first formula 'F_i' in the branch that is a premiss of an $(Intro,\star)$ rule, or otherwise let $F_i = F_n$ if there is no such intro rule in the branch. This formula 'F_i' is the minimum formula in the branch β and it fulfils the above three properties (1), (2) and (3) in the theorem. □

THEOREM 3 Subformula property for deductions in normal form: *Every occurence of a formula* F_j *in a normalized deduction* $\dfrac{\gamma}{\dfrac{\nabla}{A}}$, *is a subformula of the conclusion* A *or, otherwise, it must be a subformula of any hypothesis in* γ.

This theorem is but a conclusion of the former one. □

5 An algorithm to obtain deductions in normal form

Although the elimination rules forbid the design of a goal directed algorithm for parsing in natural deduction—for they produce an infinite ramification of the search tree—, the above theorem 3 states that, if we want to obtain deductions in normal form, we should only try with subformulae of the original sequence of hypotheses or with subformulae of the conclusion. This leads us to build the following inductive algorithm.

We will call *goal* to one sequence of categories, α (the antecedent), followed by the target category, O (the consequent), and this we will write as $[\alpha \Rightarrow O]$.

Procedure 'proof one goal' Input: one goal $[\alpha \Rightarrow O]$. **Output:** A sequence of steps that builds the proof tree (deduction). **Process:**
Initial state. Initial goal: $[\alpha \Rightarrow O]$, where α = the original sequence of categories of lexical items, and O is the original target category for the whole phrase.
Base case. If goal = $[O \Rightarrow O]$. then end procedure 'proof one goal' (success).
Elimination step. Let goal = $[\alpha \Rightarrow O]$. Execute function 'Eliminate connectives' with the given goal.
Introduction step. We should follow the above steps for each and every result from the previous function for 'Eliminate connectives': a) IF antecedent = consequent, THEN end procedure 'proof one goal' (success). b) OTHERWISE execute procedure for 'Insert connectives'.
End of procedure 'proof one goal' □

Function 'Eliminate connectives': Input: one goal. **Output:** a set of goals. **Process:** Execute every possible combination of the following steps and concatenate the results.
1) Output = original unaltered goal.
2) IF goal = $[\alpha, A/B, B, \gamma \Rightarrow O]$. THEN output = $[\alpha, A, \gamma \Rightarrow O]$. This step corresponds with the application of the $(Elim, /)$ rule.
3) IF goal = $[\alpha, B, B\backslash A, \gamma \Rightarrow O]$. THEN output = $[\alpha, A, \gamma \Rightarrow O]$. This step corresponds with the aplication of the $(Elim, \backslash)$ rule.
End of the function 'Eliminate connectives' □

Procedure 'Insert connectives': Input: one goal. **Output:** a sequence of steps that builds a deduction tree. **Process:** Try each of the following cases in a nondeterministic way:
1) IF goal = $[\alpha, A/(B/C), \beta \Rightarrow O]$ THEN 1.a) Split—non deterministically—the sequence β in one left part β_1 and another right part β_2 (β_1 should not be empty). 1.b) Execute procedure 'proof one goal' with both two new goals: $[\beta_1, C \Rightarrow B]$ and $[\alpha, A, \beta_2 \Rightarrow O]$. This step corresponds with the application of one introduction rule followed by the application of an elimination rule.
2) IF goal = $[\alpha, A/(C\backslash B), \beta \Rightarrow O]$ THEN The symmetrical situation to (1)
3) IF goal = $[\alpha, (C\backslash B)\backslash A, \beta \Rightarrow O]$ THEN The symmetrical situation to (1)
4) IF goal = $[\alpha, (B/C)\backslash A, \beta \Rightarrow O]$ THEN The symmetrical situation to (1)
5) IF goal = $[\alpha \Rightarrow A/B]$ THEN execute procedure 'proof one goal' with the new goal $[\alpha, B \Rightarrow A]$. This step corresponds with the application of some $(Intro, /)$ rule.
6) IF goal = $[\alpha \Rightarrow B\backslash A]$ THEN The symmetrical situation to (5)
7) OTHERWISE end procedure 'Insert connectives' (failure).
End of procedure 'Insert connectives' □

Steps (5) and (6) are top-down (goal directed) steps fron the conclusion to data, while steps (1), (2), (3) and (4) are mixed ones, from data to conclusion and back from conclusion to data.

6 Properties of the parsing algorithm

The algorithm converges. Because every step reduces the complexity (number of connectives) of the goal. □

The algorithm is correct. Every goal that may be constructed by this algorithm can be deduced in the original natural deduction calculus.
The proof follows since we can check that each and every step in the algorithm corresponds with the application of some of the rules of the original calculus or, otherwise, it corresponds with some combination of rules. This we have shown in each step of the algorithm. □

The algorithm is complete for deductions in normal form. Every deduction in normal form that may be infered in the original cal-

culus corresponds with some goal that may be build with this algorithm. This is a more hard to see property than the previous one. The proof is based upon the rather peculiar structure of deductions in normal form because of the theorem 2 above. Branches are made with a (possibly empty) initial sequence of elimination steps—function for 'eliminate connectives' in our algorithm—followed by another (possibly empty) sequence of introduction steps that ends up in the minor premiss of some elimination step—our (1), (2), (3) and (4) cases of the procedure for 'insert connectives' in our algorithm—or, otherwise, it must end in the conclusion of the goal—our (5) and (6) cases of the procedure for 'insert connectives' in our algorithm—and these are the only cases that we can find in deductions in normal form. □

The algorithm only builds deductions in normal form. Every deductions with that structure are normal ones. □

The algorithm is devoid of espurious ambiguity. As the algorithm only constructs deductions in normal form, and as we may assimilate the concept of one different normal form with that of one different 'semantic' form, thus the algorithm is devoid of spurious ambiguity. □

Appendix: Prolog Program listing of the algorithm

```
:- op(400,yfx,'\').

variable(1).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Automatic Theorem prover in Natural Deduction
%%%% For Categorial Grammars
%%%%
%%%% Try the consults:
%%%% ? probar([someone:s/(n\s),bore:(n\s)/n,
%%%%          everyone:(s/n)\s],ProofTree:s)
%%%% ? probar([a:a],ProofTree:(b/a)\b)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

probar([Demo:Obj], Demo:Obj) :- !.
probar(Xs,Zs) :-
  elimina(Xs,Ys),
  probar(Ys,Zs), !.

probar(Xs,Ys) :-
  insertar(Xs,Ys).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% introduction rules
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

insertar([Demo:Obj], Demo:Obj) :- !.
insertar(Xs,Demo:Obj) :-
  busca(Xs,K,Dcba:(C\B)\A,M),
  append(K1,[X|K2],K),
  variable(Z),
  Z1 is Z+1,
  abolish(variable/1),
  assert(variable(Z1)),
  Eps = eps(Z),
  append([Eps:C],[X|K2],K2s),
  probar(K2s,Demo2:B),
  append(K1,[elimBack(introBack(Eps:C,Demo2:B):C\B,
        Dcba:(C\B)\A):A|M],Ks),
  probar(Ks,Demo:Obj).

insertar(Xs,Demo:Obj) :-
  busca(Xs,K,Dcba:(B/C)\A,M),
  append(K1,[X|K2],K),
  variable(Z),
  Z1 is Z+1,
  abolish(variable/1),
  assert(variable(Z1)),
  Eps = eps(Z),
  append([X|K2],[Eps:C],K2s),
  probar(K2s,Demo2:B),
  append(K1,[elimBack(introSlash(Demo2:B,Eps:C):B/C,
        Dcba:(B/C)\A):A|M],Ks),
  probar(Ks,Demo:Obj).

insertar(Xs,Demo:Obj) :-
  busca(Xs,K,Dcba:A/(B/C),M),
  append([X|M1],M2,M),
  variable(Z),
  Z1 is Z+1,
  abolish(variable/1),
  Eps = eps(Z),
  assert(variable(Z1)),
  append([X|M1],[Eps:C],M1s),
  probar(M1s,Demo2:B),
  append(K,[elimSlash(Dcba:A/(B/C),
        introSlash(Demo2:B,Eps:C):B/C):A|M2],Ms),
  probar(Ms,Demo:Obj).

insertar(Xs,Demo:Obj) :-
  busca(Xs,K,Dcba:A/(C\B),M),
  append([X|M1],M2,M),
  variable(Z),
  Z1 is Z+1,
  abolish(variable/1),
  Eps = eps(Z),
  assert(variable(Z1)),
  append([Eps:C],[X|M1],M1s),
  probar(M1s,Demo2:B),
  append(K,[elimSlash(Dcba:A/(C\B),
        introBack(Eps:C,Demo2:B):C\B):A|M2],Ms),
  probar(Ms,Demo:Obj).

insertar(Xs,Demo:A/B) :-
  !,
  variable(Z),
  Z1 is Z+1,
```

```
  abolish(variable/1),
  Eps = eps(Z),
  assert(variable(Z1)),
  append(Xs,[Eps:B],Xss),
  probar(Xss,Demo1:A),
  Demo = introSlash(Demo1:A,Eps:B).

insertar(Xs,Demo:B\A) :-
  !,
  variable(Z),
  Z1 is Z+1,
  abolish(variable/1),
  Eps = eps(Z),
  assert(variable(Z1)),
  append([Eps:B],Xs,Xss),
  probar(Xss,Demo1:A),
  Demo = introBack(Eps:B,Demo1:A).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Look for a pattern in a string
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

busca([X|Xs],[],X,Xs).
busca([X|Xs],[X|Inic],Y,Final) :-
   busca(Xs,Inic,Y,Final).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Loop to insert elimination rules
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

elimina(Xs,Ys) :-
  elim1(Xs,Ys),
  Xs \= Ys.

elim1([],[]) :- !.
elim1([Dab:A/B,Db:B|Xs],
     [elimSlash(Dab:A/B,Db:B):A|Xs]).
elim1([Db:B,Dab:B\A|Xs],
     [elimBack(Db:B,Dab:B\A):A|Xs]).
elim1([X|Xs],[X|Ys]) :- elim1(Xs,Ys).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

References

[Ben91] J. Van Benthem. *Language in Action: Categories, Lambdas, and Dynamic Logic.* North–Holland, 1991.

[BM97] J. Van Benthem and A. Ter Meulen, editors. *Handbook of Logic and Language.* Elsevier Science B.V. and MIT Press, 1997.

[Bus97] W. Buszkowski. Mathematical linguistics and proof theory. In Benthem and Meulen [BM97].

[Car94] Bob Carpenter. A natural deduction theorem prover for type-theoretic categorial grammars. Technical report, Carnegie Mellon Laboratory for Computational Linguistics, 1994.

[Gen34] G. Gentzen. Untersuchungen über das logische schielessen. *Mathematische Zeitschrift*, 34, 1934.

[Kön89] E. König. Parsing as natural deduction. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics. Vancouver*, 1989.

[Kön94] E. König. A hypothetical reasoning algorithm for linguistic analysis. *Journal of Logic and Computation*, 4(1):1–19, 1994.

[Lam58] J. Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–169, 1958.

[Moo88] Michael Moortgat. *Categorial Investigations: Logical and Linguistics Aspects of the Lambek Calculus.* Foris Publications, 1988.

[Moo97] M. Moortgat. Categorial type logics. In Benthem and Meulen [BM97], pages 93–178.

[Pra65] Dag Prawitz. *Natural Deduction.* Almqvist and Wiksell, Uppsala, 1965.