Grammar as Logic

Glyn Morrill

July 14, 1989

Abstract

The notion of 'parsing as deduction' appears to presuppose that of 'grammar as logic', though in fact the former has largely involved *embedding* grammars in logics using a fragment conducive to theorem-proving. It is well-known that categorial grammars take the form of implicational logics; in this case the grammar is not embedded, but simply *is* is a logic. The paper argues that many further operations may be added to such a grammar, the result having close relations with linear logic. Thus categorial grammar emerges as the implicational fragment of a much more general logical grammar. A Prolog implementation illustrates applications to polymorphism, optionality, intensionality, bounded and unbounded extraction, and coordination reduction.

1 Implication

The notion of 'parsing as deduction' appears to presuppose that of 'grammar as logic', though in fact the former has largely involved *embedding* grammars in powerful logics using a fragment conducive to theorem-proving, e.g. embedding context-free grammars in the Horn clause fragment of first order logic (cf. Pereira and Warren 1983 and references therein). In categorial grammars, the slash connectives behave directly as implication. The terms of the language of categories are defined starting from a set of basic categories as follows:

(1) a. If X is a basic category
then X is a category.

b. If X and Y are categories
then X/Y and $X\backslash Y$ are categories.

A term X/Y represents expressions which apply to expressions of category Y on their right to form expressions of category X; a term $X\backslash Y$ represents expressions which apply to expressions of category Y on their left to form expressions of category X. Thus the following will be valid:

(2) a. $>$: $X/Y \quad Y \Rightarrow X$

b. $<$: $Y \quad X\backslash Y \Rightarrow X$

Note that a uniform-orientation notation is used, with arguments appearing on the same side of their value, irrespective of directionality.

The inference from X/Y and Y to X is analogous to the inference from $Y \rightarrow X$ and Y to X. Consider in particular the (product-free) Associative Lambek Calculus (ALC), which has a Gentzen-style sequent axiomatisation as follows (Lambek 1958):

(3) $X \Rightarrow X$

(4) $$\frac{\Gamma_1, X, \Gamma_2 \Rightarrow Z \quad \Delta \Rightarrow Y}{\Gamma_1, X/Y, \Delta, \Gamma_2 \Rightarrow Z}/L \qquad \frac{\Gamma_1, X, \Gamma_2 \Rightarrow Z \quad \Delta \Rightarrow Y}{\Gamma_1, \Delta, X\backslash Y, \Gamma_2 \Rightarrow Z}\backslash L$$

(5) $$\frac{\Gamma, Y \Rightarrow X}{\Gamma \Rightarrow X/Y}/R \qquad \frac{Y, \Gamma \Rightarrow X}{\Gamma \Rightarrow X\backslash Y}\backslash R$$

(6) $$\frac{\Gamma \Rightarrow X \qquad \Delta_1, X, \Delta_2 \Rightarrow Y}{\Delta_1, \Gamma, \Delta_2 \Rightarrow Y} CUT$$

Cut elimination holds for the system, i.e. every theorem has a cut-free proof, so the cut rule (6) can be omitted without effecting the theory. Because each rule introduces a connective, searching backwards without cut reduces complexity at every stage and gives a decision procedure for theoremhood. The theory defined corresponds to the implicational fragment of non-commutative linear logic (for linear logic in general see Girard 1987; Girard 1988, p14 refers specifically to the Lambek calculus). There are two entailments: one from the right and one from the left. When directionality is ignored (i.e. premises are multisets as opposed to lists), the result is commutative linear logic with one, nondirectional, implication and a structural rule of exchange. The absence of structural rules appears appropriate for grammar, where the order of premises (word categories) is crucial.

Terms of the ALC can be compositionally interpreted as string-sets thus:

(7) $$[[\mathrm{X/Y}]]_{\mathrm{syn}} = \{x | \forall y \in [[\mathrm{Y}]]_{\mathrm{syn}}, xy \in [[\mathrm{X}]]_{\mathrm{syn}}\}$$
$$[[\mathrm{X\backslash Y}]]_{\mathrm{syn}} = \{x | \forall y \in [[\mathrm{Y}]]_{\mathrm{syn}}, yx \in [[\mathrm{X}]]_{\mathrm{syn}}\}$$

A sequent $X_1, \ldots, X_n \Rightarrow X_0$ means that every expression formed by concatenating expressions of categories $X_1, \ldots, X_n$ is an expression of category X_0.

Terms can also be interpreted as meaning-classes, e.g. by translating into types and interpreting these as function spaces in the usual way:

(8) $$\tau(X/Y) = \tau(X\backslash Y) = (\tau(Y), \tau(X))$$

(9) $$[[\mathrm{(a,b)}]]_{\mathrm{sem}} = [[\mathrm{b}]]_{\mathrm{sem}}^{[[\mathrm{a}]]_{\mathrm{sem}}}$$

Van Benthem (1983) shows how to associate with each proof of a sequent $X_1, \ldots, X_n \Rightarrow X_0$ in the ALC a lambda-term mapping from meanings in categories $X_1, \ldots, X_n$ into meanings in category X_0. The left rules are functional application, and the right rules, functional abstraction.

Purely applicative analyses give canonical word orders; the semantics of discontinuity is managed by functional abstraction over the meaning of extracted elements.

2 Product

Lambek (1958) contains already a product connective representing juxtaposition. The clause (10) is added to the definition of categories; (11) will be valid.

(10) If X and Y are categories
then $X * Y$ is a category.

(11) $\times$: $X \qquad Y \Rightarrow X * Y$

The sequent rules are as follows:

(12) $$\frac{\Gamma_1, X, Y, \Gamma_2 \Rightarrow Z}{\Gamma_1, X * Y, \Gamma_2 \Rightarrow Z} * L$$

$$\frac{\Gamma \Rightarrow X \qquad \Delta \Rightarrow Y}{\Gamma, \Delta \Rightarrow X * Y} * R$$

These are the sequent rules for times/multiplicative conjunction/intensional conjunction/tensor product in linear logic (Girard and Lafont 1986, p3). Product can be interpreted syntactically as concatenation and semantically as pair formation:

(13) $[[\text{X*Y}]]_{\text{syn}} = \{xy | x \in [[\text{X}]]_{\text{syn}} \text{ and } y \in [[\text{Y}]]_{\text{syn}}\}$
$[[\text{X*Y}]]_{\text{sem}} = [[\text{X}]]_{\text{sem}} \times [[\text{Y}]]_{\text{sem}}$

We will assume that product binds more tightly than implication: $* < / = \backslash$. Products have been invoked in relation to coordination. As an illustration of possible linguistic application here, consider the small clauses posited in Government-Binding:

(14)
$$\dfrac{\dfrac{\text{Mary}}{\text{NP}} \quad \dfrac{\dfrac{\dfrac{\text{considers}}{\text{S}\backslash\text{NP}/\text{NP*(N/N)}} \quad \dfrac{\dfrac{\text{John}}{\text{NP}} \quad \dfrac{\text{lucky}}{\text{N/N}}}{\text{NP*(N/N)}}\times}{\text{S}\backslash\text{NP}}>}{\text{S}}<$$

3 Intersection and Union

Van Benthem (1989) suggests intersection and union as natural operations on string-sets; the same is true as operations on function-spaces. The following clauses are added to the definition of categories:

(15) If X and Y are categories
then $X\&Y$ and $X + Y$ are categories.

For intersection (boolean conjunction) & the following will be valid:

(16) a. $\cap_1$: $X\&Y \Rightarrow X$
b. $\cap_2$: $X\&Y \Rightarrow Y$

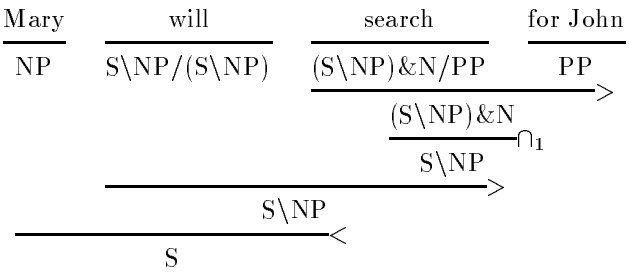
We propose the following sequent rules:

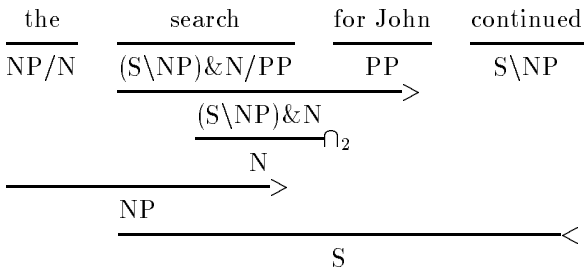
(17)
$$\frac{\Gamma_1, X, \Gamma_2 \Rightarrow Z}{\Gamma_1, X\&Y, \Gamma_2 \Rightarrow Z}\&L_1 \qquad \frac{\Gamma_1, Y, \Gamma_2 \Rightarrow Z}{\Gamma_1, X\&Y, \Gamma_2 \Rightarrow Z}\&L_2$$

$$\frac{\Gamma \Rightarrow X \quad \Gamma \Rightarrow Y}{\Gamma \Rightarrow X\&Y}\&R$$

These are the rules for with/extensional conjunction/direct product in linear logic (Girard and Lafont 1986, p3). We will assume that intersection binds exactly as tightly as product: $\& = * < / = \backslash$. Intersection of categories expresses polymorphism. For example assigning an expression to category NP&SP indicates that it is both a noun phrase and a complementized sentence. Intersection in the value part of a category expresses value polymorphism. Thus if it is believed that *search* is in fact the same when it takes a PP*for* complement to form an untensed verb phrase, and when it takes a PP*for* to form a noun, we might express this by assigning it to the category (S\NP)&N/PP:

(18) a. Mary will search for John.
b. The search for John continued.

(19)


(20)


For union (boolean disjunction) $+$, (21) will be valid.

(21) a. $\cup_1$: $X \Rightarrow X + Y$

b. $\cup_2$: $Y \Rightarrow X + Y$

The following sequent axiomatisation is proposed:

(22)
$$\frac{\Gamma_1, X, \Gamma_2 \Rightarrow Z \qquad \Gamma_1, Y, \Gamma_2 \Rightarrow Z}{\Gamma_1, X + Y, \Gamma_2 \Rightarrow Z} +L$$

$$\frac{\Gamma \Rightarrow X}{\Gamma \Rightarrow X + Y} +R_1 \qquad \frac{\Gamma \Rightarrow Y}{\Gamma \Rightarrow X + Y} +R_2$$

This is plus/extensional disjunction/direct sum in linear logic (Girard and Lafont 1987, p3). Union is assumed to bind exactly as tightly as intersection (and product): $+ = \& = * < / = \backslash$. Union in the argument part of a category expresses argument polymorphism. If it is assumed that it is the same verb *see* which combines with noun phrases and small clauses, we may assign it to the category S\NP/NP+(NP*(S\NP)):

(23) a. Mary saw the man.

b. Mary saw John yawn.

(24)
$$\dfrac{\dfrac{\text{Mary}}{\text{NP}} \quad \dfrac{\dfrac{\text{saw}}{\text{S}\backslash\text{NP}/\text{NP}+(\text{NP}^*(\text{S}\backslash\text{NP}))} \quad \dfrac{\dfrac{\text{the man}}{\text{NP}}}{\text{NP}+(\text{NP}^*(\text{S}\backslash\text{NP}))}\cup_1}{\text{S}\backslash\text{NP}}>}{\text{S}}<$$

(25)
$$\dfrac{\dfrac{\text{Mary}}{\text{NP}} \quad \dfrac{\dfrac{\text{saw}}{\text{S}\backslash\text{NP}/\text{NP}+(\text{NP}^*(\text{S}\backslash\text{NP}))} \quad \dfrac{\dfrac{\text{John}}{\text{NP}} \quad \dfrac{\text{yawn}}{\text{S}\backslash\text{NP}}}{\text{NP}^*(\text{S}\backslash\text{NP})}\times}{\text{NP}+(\text{NP}^*(\text{S}\backslash\text{NP}))}\cup_2}{\text{S}\backslash\text{NP}}>}{\text{S}}<$$

4 Exponentials

Linear logic contains unary operators ! ('of-course!') and ? ('why-not?'), sometimes compared to the modalities necessity and possibility. They reintroduce in a controlled manner the structural rules that are lost in the transition from intuitionistic logic to linear logic. From a linguistic point of view, we may look to $!X$ to mean 'one or more Xs', and in this way approach iterated coordination: *John, Bill, Mary, Suzy, and Fred*; we may look to $?X$ to mean 'one or no Xs' and so express *optionality*. For the latter (27) will be valid.

(26) If X is category
then $!X$ and $?X$ are categories.

(27) a. $()_1$: $X \Rightarrow ?X$
b. $()_0$: $\Rightarrow ?X$

There could be the following axiom and sequent rule:

(28) $\Rightarrow ?X$

(29)
$$\dfrac{\Gamma \Rightarrow Y}{\Gamma \Rightarrow ?Y}?\text{R}$$

The implementation in the appendix includes ? but not !. A unary operator associates to the right and is assumed to bind tighter than the binary operators: $? < + = \& = * < / = \backslash$. The optionality of the object of e.g. *eat* is thus characterised by assignment to S\NP/?NP:

(30)
$$\dfrac{\dfrac{\text{John}}{\text{NP}} \quad \dfrac{\dfrac{\text{ate}}{\text{S}\backslash\text{NP}/?\text{NP}} \quad \dfrac{\dfrac{\text{the cake}}{\text{NP}}}{?\text{NP}}()_1}{\text{S}\backslash\text{NP}}>}{\text{S}}<$$

(31)
$$\dfrac{\dfrac{\text{John}}{\text{NP}} \quad \dfrac{\dfrac{\text{ate}}{\text{S\NP/?NP}} \quad \dfrac{}{\text{?NP}}()_0}{\text{S\NP}}>}{\text{S}}<$$

5 Modality

In Morrill (1989) it is suggested that intensionality be represented in the language of categories by means of a one-place operator, and it is observed that when this is done, the combinatorics of intensional types is given by modal necessity; that is, the logic of intensional types is the logic of necessity. Where X is a category, $\Box X$ is the corresponding intensional category. Clause (32) is added to the definition of terms; there is the type-mapping in (33), where s represents the set of indices.

(32) If X is a category
then $\Box X$ is a category.

(33) $\tau(\Box X) = (s, \tau(X))$

The main part of that paper uses minimal modal logic, though addition of further axioms is suggested at the end and there are reasons to believe that this is appropriate. Nevertheless, we use just minimal modal logic here, axiomatised thus:

(34)
$$\dfrac{\Gamma \Rightarrow X}{\Box\Gamma \Rightarrow \Box X}\Box$$

The $\Box$ is again assumed to bind more tightly than the binary operators: $\Box =? < + = \& = * < / = \backslash$. Morrill (1989) describes how at the same time as doing the book-keeping for an intensional semantics, the modal apparatus specifies intensional domains in terms of which it is possible to express boundedness of reflexivisation as contrasted with unboundedness of relativisation; these points will not be repeated here, though they are illustrated in the appendix.

As remarked above, the linear logic exponentials are comparable to modals, though for the current application the role of ! as representing Kleene plus iteration, and $\Box$ as representing intensionality, are quite different. This may indicate that a distinction is to be drawn in the context of grammar; alternatively, the suitablity of $\Box$ to intensionality may draw from its relatedness to universal quantification, in which case the latter might be used.

6 Coordination

To assign a coordinator a category such as S\S/S presents it rather as a subordinator. We introduce a compound connective /\ indicating simultaneous forward and backward application (cf. Geach 1972, p485). I am not aware of a logical analogue. There is the following sequent rule:

(35)
$$\dfrac{\Gamma_{12}, \Delta_1, \Gamma_{21} \Rightarrow Y \qquad \Gamma_{12}, \Delta_2, \Gamma_{21} \Rightarrow Y \qquad \Gamma_{11}, X, \Gamma_{22} \Rightarrow Z}{\Gamma_{11}, \Gamma_{12}, \Delta_1, X/\backslash Y, \Delta_2, \Gamma_{21}, \Gamma_{22} \Rightarrow Z}$$

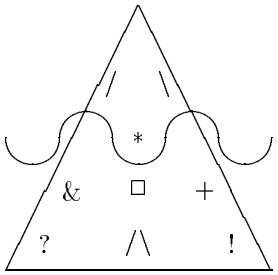
The /\ binds more loosely than all the other connectives: $\Box =? < + = \& = * < / = \backslash < /\backslash$.

7 Discussion

The appendix contains an implementation which assumes cut-elimination, and which associates a functional semantics with proofs (cf. also 'formulae as types', Howard 1980). It only seems appropriate to require the cut rule to be valid, since it expresses the idea of consequence which is fundamental to logic. A cut-elimination proof is then required to ensure the completeness of cut-free analysis of the kind in the appendix. In this connection, the cut-elimination results from linear logic should prove helpful, as should the method of proof-nets and normalisation developed in that context. At this point, further logical operations may still be expected to be found, and full work on their axiomatisation and semantics remains to be carried out.

That implication has been most central in grammar need come as no surprise, since entailment is the "heart of logic". But the picture emerging according to this paper is that categorial grammar forms just the implicational fragment of a much larger *logical grammar*, with close relations to linear logic.

(36)



The implementation in the appendix illustrates applications to polymorphism, optionality, intensionality, bounded and unbounded extraction, and coordination reduction.

8 Commentary on the Program and Log

In the translations, application is represented by ' which associates to the left, and abstraction is represented by @, which takes the abstractor variable to its left and which associates to the right. Sometimes the translations obtained could undergo eta-reduction. Analysis halts after one derivation has been found; exhaustive search would yield a large (but finite) number of derivations assigning the same meaning.

Examples 1 and 2 illustrate simple sentences. In these, all words are evaluated at the same index; this is indicated by the fact that each word meaning (assumed to be intensional) applies initially to the same index. The translation as a whole is the abstraction over the indices, i.e. the proposition expressed by the sentence. Example 3 exhibits an intensional domain: the elements of the embedded clause are evaluated at an index different form that of the superordinate clause, and the sentence-embedding verb applies to embedded sentence interpretation abstracted over indices — it applies to the intension of the embedded clause. Examples 4 and 5 illustrate how intensional verbs, and prepositions (assumed to take intensional objects), can extensionalise by applying to Lifted arguments.

Example 6 demonstrates the product analysis of small clauses; in the translation, pairs are represented by commas and brackets. In 7 and 8, *search* has a single value-polymorphic category covering its role as an untensed verb and a noun. In 9 and 10, *saw* has a single argument-polymorphic category covering its role as a governer of noun phrases or small clauses; the latter is again encoded by means of product. Examples 11 and 12 indicate optionality.

Examples 13 and 14 show simple subject and object relativisation respectively. In 15 and 16 there is object relativisation of an intensional argument, and out of an intensional domain, respectively; in 17 there is both.

18 and 19 show simple agreement for reflexives, implemented by featural analysis of basic nominal categories; 20 and 21 show agreement for reflexives with relativisation. 22 and 23 show

reflexives in positions which can be intensional; 24 shows reflexivisation across an auxiliary and with the value-polymorphic *search*. Example 25 shows that reflexivisation cannot pass out of an embedded clauses, which form intensional domains.

Examples 26 to 30 show sentence coordination, intransitive verb phrase coordination, transitive verb phrase coordination, subject coordination, and object coordination, respectively; 31 is not generated, reflecting a problem with agreement and coordination.

The right node raising in 32, and the across-the-board extraction in 33, are obtained, but the embedded right node raising in 34 is not generated. This involves right node raising from an intensional domain; presumably related is the failure to generate the left node raising in 34, involving adverbials also assumed to create intensional domains. The left node raising in 36, not involving intensionality, it obtained.

Appendix A: Prolog Implementation

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

:- op(350,yfx,*).

:- op(350,yfx,&).
:- op(350,yfx,+).

:- op(330,fy,#).

:- op(430,yfx,'/\').

:- op(330,fy,?).
:- op(330,fy,!).

:- op(600,xfx,':=').

:- op(600,xfx,'=>').

:- op(500,xfx,:).

:- op(450,yfx,`).
:- op(450,xfy,@).

top(Str,Trans) :-
        lex_entries(Str,Premises),
        theorem(Premises => Trans: #s).

lex_entries([],[]).

lex_entries([Word|Words],[Word:Cat|Cats]) :-
        Word := Cat,
        lex_entries(Words,Cats).

% Axiom

theorem([A:X] => A:X) :-
        basic_cat(X).
```

```
% /L

theorem(Premises => CZ) :-
        appendn([Gamma1,[A:X/Y],Delta,Gamma2],Premises),
        theorem(Delta => B:Y),
        appendn([Gamma1,[A‘B:X],Gamma2],Gamma),
        theorem(Gamma => CZ).

% \L

theorem(Premises => CZ) :-
        appendn([Gamma1,Delta,[A:X\Y],Gamma2],Premises),
        theorem(Delta => B:Y),
        appendn([Gamma1,[A‘B:X],Gamma2],Gamma),
        theorem(Gamma => CZ).

% /R

theorem(Premises => B@A:X/Y) :-
        append(Premises,[B:Y],Gamma),
        theorem(Gamma => A:X).

% \R

theorem(Premises => B@A:X\Y) :-
        append([B:Y],Premises,Gamma),
        theorem(Gamma => A:X).

% *L

theorem(Premises => CZ) :-
        appendn([Gamma1,[(A,B):X*Y],Gamma2],Premises),
        appendn([Gamma1,[A:X],[B:Y],Gamma2],Gamma),
        theorem(Gamma => CZ).

% *R

theorem(Premises => (A,B):X*Y) :-
        append(Gamma,Delta,Premises),
        theorem(Gamma => A:X),
        theorem(Delta => B:Y).

% &L1

theorem(Premises => CZ) :-
        appendn([Gamma1,[A:X&_],Gamma2],Premises),
        appendn([Gamma1,[A:X],Gamma2],Gamma),
        theorem(Gamma => CZ).

% &L2

theorem(Premises => CZ) :-
        appendn([Gamma1,[A:_&Y],Gamma2],Premises),
```

```
        appendn([Gamma1,[A:Y],Gamma2],Gamma),
        theorem(Gamma => CZ).

% &R

theorem(Premises => A:X&Y) :-
        theorem(Premises => A:X),
        theorem(Premises => A:Y).

% +L

theorem(Premises => CZ) :-
        appendn([Gamma1,[A:X+Y],Gamma2],Premises),
        appendn([Gamma1,[A:X],Gamma2],GammaX),
        theorem(GammaX => CZ),
        appendn([Gamma1,[A:Y],Gamma2],GammaY),
        theorem(GammaY => CZ).

% +R1

theorem(Premises => A:X+_) :-
        theorem(Premises => A:X).

% +R2

theorem(Premises => A:_+Y) :-
        theorem(Premises => A:Y).

% ? Axiom

theorem([] => _: ?_).

% ?R

theorem(Premises => C: ?A) :-
        theorem(Premises => C:A).

% #

theorem(Premises => I@A: #X) :-
        mod(DeModPremises,Premises,I),
        theorem(DeModPremises => A:X).

% /\

theorem(Premises => CZ) :-
        appendn([Gamma11,Gamma12,Delta1,[A:X/\Y],Delta2,Gamma21,Gamma22],Premises),
        appendn([Gamma12,Delta1,Gamma21],Gamma1),
        theorem(Gamma1 => B1:Y),
        appendn([Gamma12,Delta2,Gamma21],Gamma2),
        theorem(Gamma2 => B2:Y),
        appendn([Gamma11,[A‘(B1,B2):X],Gamma22],Gamma),
        theorem(Gamma => CZ).
```

```
mod([],[],_).

mod([A‘I:X|DeMods],[A: #X|Mods],I) :-
        mod(DeMods,Mods,I).

appendn([],[]).

appendn([L1|Ls],L) :-
        append(L1,L2,L),
        appendn(Ls,L2).

append([],L,L).

append([H|L1],L2,[H|L]) :-
        append(L1,L2,L).

basic_cat(s).
basic_cat(sp).
basic_cat(n(_)).
basic_cat(np(_)).
basic_cat(pp).

and := #(s/\s).
ate := #(s\np(_)/ ?np(_)).
arrived := #(s\np(_)).
book := #n(n).
cake := #n(n).
considers := #(s\np(_)/np(_)*(n(G)/n(G))).
continued := #(s\np(_)).
dislikes := #(s\np(_)/np(_)).
for := #(pp/(#s\(#s/np(_)))).
give := #(s\np(_)/np(_)/np(_)).
herself := #(s\np(f)\(s\np(f)/np(f))).
himself := #(s\np(m)\(s\np(m)/np(m))).
john := #np(m).
left := #(s\np(_)).
likes := #(s\np(_)/np(_)).
lucky := #(n(G)/n(G)).
man := #n(m).
married := #(s\np(_)/np(_)).
mary := #np(f).
met := #(s\np(_)/np(_)).
record := #n(_).
saw := #(s\np(_)/np(_)+(np(G)*(s\np(G)))).
search := #((s\np(_))&n(n)/pp).
sent := #(s\np(_)/np(_)/np(_)).
sought := #(s\np(_)/(#s\(#s/np(_)))).
suzy := #np(_).
the := #(np(G)/n(G)).
thinks := #(s\np(_))/ #s.
today := #(s\np(G))\ #(s\np(G)).
voted := #(s\np(_)/pp).
who := #(n(G)\n(G))/(#s\ #np(G)).
who := #(n(G)\n(G))/(#s/ #np(G)).
```

```
will := #(s\np(G)/(s\np(G))).
yawn := #(s\np(_)).
yesterday := #(s\np(G))\ #(s\np(G)).

% Simple sentences

str(1,[john,left]).
str(2,[mary,will,give,john,the,lucky,book]).

% Modality: intensional domains

str(3,[suzy,thinks,mary,met,john]).

% Abstraction: extensionalisation of prepositions and intensional verbs

str(4,[mary,sought,john]).
str(5,[mary,voted,for,the,man]).

% Product: small clauses

str(6,[mary,considers,john,lucky]).

% Intersection: value polymorphism

str(7,[mary,will,search,for,john]).
str(8,[the,search,for,john,continued]).

% Union: argument polymorphism

str(9,[mary,saw,the,man]).
str(10,[mary,saw,john,yawn]).

% Why not: optionality

str(11,[john,ate,the,cake]).
str(12,[john,ate]).

% Abstraction and Modality: relativisation and reflexivisation

str(13,[the,man,who,saw,mary,left]).
str(14,[the,man,who,suzy,met,left]).
str(15,[the,man,who,suzy,sought,left]).
str(16,[the,man,who,mary,thinks,suzy,met,left]).
str(17,[the,man,who,mary,thinks,suzy,voted,for,left]).
str(18,[mary,saw,herself]).
str(19,[mary,saw,himself]).
str(20,[the,man,who,saw,himself,left]).
str(21,[the,man,who,saw,herself,left]).
str(22,[mary,sought,herself]).
str(23,[mary,voted,for,herself]).
str(24,[mary,will,search,for,herself]).
str(25,[mary,thinks,john,met,herself]).

% Coordination
```

```
str(26,[john,arrived,and,mary,left]).
str(27,[suzy,arrived,and,left]).
str(28,[mary,met,and,married,john]).
str(29,[suzy,and,mary,left]).
str(30,[john,met,mary,and,suzy]).
str(31,[john,and,mary,left]).
str(32,[john,likes,and,mary,dislikes,suzy]).
str(33,[the,man,who,john,likes,and,mary,dislikes,left]).
str(34,[mary,thinks,john,likes,and,suzy,thinks,john,dislikes,the,man]).
str(35,[john,saw,mary,yesterday,and,suzy,today]).
str(36,[john,sent,suzy,the,book,and,mary,the,record]).

test(N) :-
        str(N,Str),
        nl, nl, write(N), tab(2), write(Str), nl,
        test1(Str).

test1(Str) :-
        top(Str,Trans),
        numbervars(Trans,0,_), nl, write(Trans), !, fail.
```

Appendix B: Log of Terminal Session

```
Script started on Fri Jul 14 02:53:07 1989
% qprolog

Quintus Prolog Release 2.2 (Sun-3, Unix 3.2)
Copyright (C) 1987, Quintus Computer Systems, Inc.  All rights reserved.
1310 Villa Street, Mountain View, California  (415) 965-7700

| ?- compile(tp18).
[compiling /home/user3/glyn/Sequent/tp18...]
[tp18 compiled 20.550 sec 10,048 bytes]

yes
| ?- test(_).


1  [john,left]

A@left`A` (john`A)

2  [mary,will,give,john,the,lucky,book]

A@will`A` (B@give`A` (john`A)` (the`A` (lucky`A` (book`A)))`B)` (mary`A)

3  [suzy,thinks,mary,met,john]

A@thinks` (B@met`B` (john`B)` (mary`B))`A` (suzy`A)
```

```
4  [mary,sought,john]

A@sought‘A‘ (B@C@B‘ (john‘A)‘C)‘ (mary‘A)

5  [mary,voted,for,the,man]

A@voted‘A‘ (for‘A‘ (B@C@B‘ (the‘A‘ (man‘A))‘C))‘ (mary‘A)

6  [mary,considers,john,lucky]

A@considers‘A‘ (john‘A,B@lucky‘A‘B)‘ (mary‘A)

7  [mary,will,search,for,john]

A@will‘A‘ (B@search‘A‘ (for‘A‘ (C@D@C‘ (john‘A)‘D))‘B)‘ (mary‘A)

8  [the,search,for,john,continued]

A@continued‘A‘ (the‘A‘ (search‘A‘ (for‘A‘ (B@C@B‘ (john‘A)‘C))))

9  [mary,saw,the,man]

A@saw‘A‘ (the‘A‘ (man‘A))‘ (mary‘A)

10  [mary,saw,john,yawn]

A@saw‘A‘ (john‘A,B@yawn‘A‘B)‘ (mary‘A)

11  [john,ate,the,cake]

A@ate‘A‘ (the‘A‘ (cake‘A))‘ (john‘A)

12  [john,ate]

A@ate‘A‘B‘ (john‘A)

13  [the,man,who,saw,mary,left]

A@left‘A‘ (the‘A‘ (who‘ (B@C@saw‘C‘ (mary‘C)‘ (B‘C))‘A‘ (man‘A)))

14  [the,man,who,suzy,met,left]

A@left‘A‘ (the‘A‘ (who‘ (B@C@met‘C‘ (B‘C)‘ (suzy‘C))‘A‘ (man‘A)))

15  [the,man,who,suzy,sought,left]

A@left‘A‘ (the‘A‘ (who‘ (B@C@sought‘C‘ (D@E@D‘ (B‘C)‘E)‘ (suzy‘C))‘A‘ (man‘A)))

16  [the,man,who,mary,thinks,suzy,met,left]

A@left‘A‘ (the‘A‘ (who‘ (B@C@thinks‘ (D@met‘D‘ (B‘D)‘ (suzy‘D))‘C‘ (mary‘C))‘A‘ (man‘A)))

17  [the,man,who,mary,thinks,suzy,voted,for,left]
```

```
A@left‘A‘ (the‘A‘ (who‘ (B@C@thinks‘ (D@voted‘D‘ (for‘D‘ (E@F@E‘ (B‘D)‘F))‘ (suzy‘D))‘C‘ (mary‘C))‘A

18  [mary,saw,herself]

A@herself‘A‘ (B@C@saw‘A‘B‘C)‘ (mary‘A)

19  [mary,saw,himself]


20  [the,man,who,saw,himself,left]

A@left‘A‘ (the‘A‘ (who‘ (B@C@himself‘C‘ (D@E@saw‘C‘D‘E)‘ (B‘C))‘A‘ (man‘A)))

21  [the,man,who,saw,herself,left]


22  [mary,sought,herself]

A@herself‘A‘ (B@C@sought‘A‘ (D@E@D‘B‘E)‘C)‘ (mary‘A)

23  [mary,voted,for,herself]

A@herself‘A‘ (B@C@voted‘A‘ (for‘A‘ (D@E@D‘B‘E))‘C)‘ (mary‘A)

24  [mary,will,search,for,herself]

A@will‘A‘ (B@herself‘A‘ (C@D@search‘A‘ (for‘A‘ (E@F@E‘C‘F))‘D)‘B)‘ (mary‘A)

25  [mary,thinks,john,met,herself]


26  [john,arrived,and,mary,left]

A@and‘A‘ (arrived‘A‘ (john‘A),left‘A‘ (mary‘A))

27  [suzy,arrived,and,left]

A@and‘A‘ (arrived‘A‘ (suzy‘A),left‘A‘ (suzy‘A))

28  [mary,met,and,married,john]

A@and‘A‘ (met‘A‘ (john‘A)‘ (mary‘A),married‘A‘ (john‘A)‘ (mary‘A))

29  [suzy,and,mary,left]

A@and‘A‘ (left‘A‘ (suzy‘A),left‘A‘ (mary‘A))

30  [john,met,mary,and,suzy]

A@and‘A‘ (met‘A‘ (mary‘A)‘ (john‘A),met‘A‘ (suzy‘A)‘ (john‘A))

31  [john,and,mary,left]
```

```
32  [john,likes,and,mary,dislikes,suzy]

A@and‘A‘ (likes‘A‘ (suzy‘A)‘ (john‘A),dislikes‘A‘ (suzy‘A)‘ (mary‘A))

33  [the,man,who,john,likes,and,mary,dislikes,left]

A@left‘A‘ (the‘A‘ (who‘ (B@C@and‘C‘ (likes‘C‘ (B‘C)‘ (john‘C),dislikes‘C‘ (B‘C)‘ (mary‘C)))‘A‘ (man‘

34  [mary,thinks,john,likes,and,suzy,thinks,john,dislikes,the,man]


35  [john,saw,mary,yesterday,and,suzy,today]


36  [john,sent,suzy,the,book,and,mary,the,record]

A@and‘A‘ (sent‘A‘ (suzy‘A)‘ (the‘A‘ (book‘A))‘ (john‘A),sent‘A‘ (mary‘A)‘ (the‘A‘ (record‘A))‘ (john
no
| ?- ^Z
[ End of Prolog execution ]
%

script done on Fri Jul 14 03:07:10 1989
```

References

van Benthem, Johan: 1983, *The semantics of Variety in Categorial Grammar*, Report 83-29, Department of Mathematics, Simon Fraser University. Also in W. Buszkowski *et al.* (eds.), 1988, *Categorial Grammar*, Volume 25, Linguistic & Literary Studies in Eastern Europe, John Benjamins, Amsterdam/Philadelphia.

van Benthem, Johan: 1989, *Language in Action*, ms., Faculteit Wiskunde en Informatica, Universiteit van Amsterdam.

Geach, P. T.: 1972, 'A program for syntax', in D. Davidson and G. Harman (eds.) *Semantics of Natural Language*, D. Reidel, Dordrecht.

Girard, Jean-Yves: 1987, 'Linear Logic', *Theoretical Computer Science* **50**, 1–102.

Girard, Jean-Yves, 1988, 'Geometry of Interaction', unpublished, Paris.

Girard, Jean-Yves and Yves Lafont: 1986, 'Linear Logic and Lazy Computation', Rapports de Recherche No. 588, Institut National de Recherche en Informatique et en Automatique, Domaine de Voluceau, Rocquencourt B.P.105, 78153 Le Chesnay Cedex, France.

Howard, W.A.: 1980, 'The formulae-as-types notion of construction, in J.R. Hindley and J.P Seldin (eds.) *To H. B. Curry, Essays on Combinatory Logic, Lambda Calculus, and Formalism*, Academic Press.

Lambek, J.: 1958, 'The mathematics of sentence structure', *American Mathematical Monthly* **65**, 154–170.

Morrill, Glyn: 1989, 'Intensionality, Boundedness, and Modal Logic, Research Paper EUCCS/RP-32, Centre for Cognitive Science, University of Edinburgh.

Pereira, Fernando C. N. and David H. D. Warren: 1983, 'Parsing as deduction', in *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, Massachusetts Institute of Technology, Cambridge, Massachusetts.