LOGICAL GRAMMAR

Glyn Morrill

1 FORMAL GRAMMAR

The canonical linguistic process is the cycle of the speech-circuit [Saussure, 1915]. A speaker expresses a psychological idea by means of a physiological articulation. The signal is transmitted through the medium by a physical process incident on a hearer who from the consequent physiological impression recovers the psychological idea. The hearer may then reply, swapping the roles of speaker and hearer, and so the circuit cycles.

For communication to be successful speakers and hearers must have shared associations between forms (*signifiers*) and meanings (*signifieds*). De Saussure called such a pairing of signifier and signified a *sign*. The relation is one-to-many (ambiguity) and many-to-one (paraphrase). Let us call a stable totality of such associations a *language*. It would be arbitrary to propose that there is a longest expression (where would we propose to cut off *I know that you know that I know that you know* ...?) therefore language is an infinite abstraction over the finite number of acts of communication that can ever occur.

The program of formal syntax [Chomsky, 1957] is to define the set of all and only the strings of words which are well-formed sentences of a natural language. Such a system would provide a map of the space of expression of linguistic cognition. The methodological idealisations the program requires are not unproblematic. How do we define what is a 'word'? Speaker judgements of well-formededness vary. Nevertheless there are extensive domains of uncontroversial and robust data to work with. The greater scientific prize held out is to realize this program 'in the same way' that it is done psychologically, i.e. to discover principles and laws of the language faculty of the mind/brain. Awkwardly, Chomskyan linguistics has disowned formalisation as a means towards such higher goals.

The program of formal semantics [Montague, 1974] is to associate the meaningful expressions of a natural language with their logical semantics. Such a system would be a characterisation of the range and means of expression of human communication. Again there are methodological difficulties. Where is the boundary between linguistic (dictionary) and world (encyclopedic) knowledge? Speaker judgements of readings and entailments vary. The program holds out the promise of elucidating the mental domain of linguistic ideas, thoughts and concepts and relating it to the physical domain of linguistic articulation. That is, it addresses a massive, pervasive and ubiquitous mind/body phenomenon.

Handbook of the Philosophy of Science. Volume 14: Philosophy of Linguistics.
Volume editors: Ruth Kempson, Tim Fernando, and Nicholas Asher. General editors: Dov M. Gabbay, Paul Thagard and John Woods.

It could be argued that since the program of formal syntax is hard enough in itself, it's pursuit should be modularised from the further challenges of formal semantics. That is, that syntax should be pursued autonomously from semantics. On the other hand, attention to semantic criteria may help guide our path through the jungle of syntactic possibilities. Since the raison d'être of language is to express and communicate, i.e. to have meaning, it seems more reasonable to posit the syntactic reality of a syntactic theory if it supports a semantics. On this view, it is desirable to pursue formal syntax and formal semantics in a single integrated program of formal grammar.

We may speak of syntax, semantics or grammar as being *logical* in a weak sense when we mean that they are being systematically studied in a methodologically rational inquiry or scientific (hypothetico-deductive) fashion. But when the formal systems of syntax resemble deductive systems, we may speak of *logical syntax* in a strong sense. Likewise, when formal semantics models in particular the logical semantics of natural language, we may speak of *logical semantics* in a strong sense. Formal grammar as comprising a syntax which is logical or a semantics which is logical may then inherit the attribute *logical*, especially if it is logical in both of the respects.

In section 1 of this article we recall some relevant logical tools: predicate logic, sequent calculus, natural deduction, typed lambda calculus and the Lambek calculus. In section 2 we comment on transformational grammar as formal syntax and Montague grammar as formal semantics. In section 3 we take a tour through some grammatical frameworks: Lexical-Functional Grammar, Generalized Phrase Structure Grammar, Head-driven Phrase Structure Grammar, Combinatory Categorial Grammar and Type Logical Categorial Grammar. There are many other worthy approaches and no excuses for their omission here will seem adequate to their proponents, but reference to these formalisms will enable us to steer towards what we take to be the 'logical conclusion' of logical grammar.

2 LOGICAL TOOLS

2.1 Predicate logic

Logic advanced little in the two millennia since Aristotle. The next giant step was Frege's [1879] *Begriffsschrift* ('idea writing' or 'ideagraphy'). Frege was concerned to provide a formal foundation for arithmetic and to this end he introduced quantificational logic. Peano called Frege's theory of quantification 'abstruse' and at the end of his life Frege considered that he had failed in his project; in a sense it was proved shortly afterwards in Gödel's incompleteness theorem that the project could not succeed. But Frege had laid the foundations for modern logic and already in the *Begriffsschrift* had effectively defined a system of predicate calculus that would turn out to be complete. Frege used a graphical notation; in the textual notation that has come to be standard the language of first-order logic is as follows:

$$\begin{array}{rcll}
[c]^g & = & F(c) & \text{for } c \in C \\
[x]^g & = & g(x) & \text{for } x \in V \\
[f(t_1, \ldots, t_i)]^g & = & F(f)([t_1]^g, \ldots, [t_i]^g) & \text{for } f \in F^i, i > 0 \\
[Pt_1 \ldots t_i]^g & = & \left\{ \begin{array}{ll} \{\emptyset\} & \text{if } \langle [t_1]^g, \ldots, [t_i]^g \rangle \in F(P) \\ \emptyset & \text{otherwise} \end{array} \right. & \text{for } P \in P^i, i \geq 0 \\
[\neg A]^g & = & \overline{[A]^g}^{\{\emptyset\}} & \\
[(A \wedge B)]^g & = & [A]^g \cap [B]^g & \\
[(A \vee B)]^g & = & [A]^g \cup [B]^g & \\
[(A \rightarrow B)]^g & = & \left\{ \begin{array}{ll} \{\emptyset\} & \text{if } [A]^g \subseteq [B]^g \\ \emptyset & \text{otherwise} \end{array} \right. & \\
[\forall x A]^g & = & \bigcap_{d \in D} [A]^{(g - (x, g(x))) \cup \{(x,d)\}} & \\
[\exists x A]^g & = & \bigcup_{d \in D} [A]^{(g - (x, g(x))) \cup \{(x,d)\}} &
\end{array}$$

Figure 1. semantics of first-order logic

(1) **Definition** (*language of first-order logic*)

Let there be a set C of (individual) *constants*, a denumerably infinite set V of (individual) *variables*, a set F^i of *function letters* of arity i for each $i > 0$, and a set P^i of *predicate letters* of arity i for each $i \geq 0$. The set $\mathcal{T}$ of *first-order terms* and the set $\mathcal{F}$ of *first-order formulas* are defined recursively as follows:

$$\begin{array}{lcl}
\mathcal{T} & ::= & C \mid V \mid F^i(\mathcal{T}_1, \ldots, \mathcal{T}_i), i > 0 \\
\mathcal{F} & ::= & P^i \mathcal{T}_1 \ldots \mathcal{T}_i, i \geq 0 \\
& & \mid \neg\mathcal{F} \mid (\mathcal{F} \wedge \mathcal{F}) \mid (\mathcal{F} \vee \mathcal{F}) \mid (\mathcal{F} \rightarrow \mathcal{F}) \mid \forall V \mathcal{T} \mid \exists V \mathcal{T}
\end{array}$$

The standard semantics of first-order logic was given by Tarski [1935]; here we use $\{\emptyset\}$ and $\emptyset$ for the truth values true and false respectively, so that the connectives are interpreted by set-theoretic operations. An *interpretation of first-order logic* is a structure (D, F) where *domain* D is a non-empty set (of *individuals*) and *interpretation function* F is a function mapping each individual constant to an individual in D, each function letter of arity $i > 0$ to an i-ary operation in D^{D^i}, and each predicate letter of arity $i \geq 0$ to an i-ary relation in D^i. An *assignment function* g is a function mapping each individual variable to an individual in D. Each term or formula ϕ receives a semantic value $[\phi]^g$ relative to an interpretation (D, F) and an assignment g as shown in figure 1.

A formula A *entails* a formula B, or B is a *logical consequence* of A, if and only if $[A]^g \subseteq [B]^g$ in every interpretation and assignment. Clearly the entailment relation inherits from the subset relation the properties of reflexivity (A entails A) and transitivity (if A entails B and B entails C, then A entails C).

2.2 Sequent calculus

First-order entailment is an infinitary semantic notion since it appeals to the class of all interpretations. Proof theory aims to capture such semantic notions as entailment in finitary syntactic formal systems. Frege's original proof calculus had proofs as sequences of formulas (what are often termed Hilbert systems). Such systems have axiom schemata (that may relate several connectives) and rules that are sufficient to capture the properties of entailment. However, Gentzen [1934] provided a great improvement by inventing calculi, both sequent calculus and natural deduction, which aspire to deal with single occurrences of single connectives at a time, and which thus identify in a modular way the pure inferential properties of each connective.

A classical *sequent* $\Gamma \Rightarrow \Delta$ comprises an *antecedent* Γ and a *succedent* Δ which are finite, possibly empty, sequences of formulas. A sequent is read as asserting that the conjunction of the antecedent formulas (where the empty sequence is the conjunctive unit true) entails the disjunction of the succedent formulas (where the empty sequence is the disjunctive unit false). A sequent is called *valid* if and only if this assertion is true; otherwise it is called *invalid*. The sequent calculus for the propositional part of classical logic can be presented as shown in figure 2. Each rule has the form $\frac{\Sigma_1 \;\ldots\; \Sigma_n}{\Sigma_0}, n \geq 0$ where the Σ_i are sequent schemata; $\Sigma_1, \ldots, \Sigma_n$ are referred to as the *premises*, and Σ_0 as the *conclusion*.

The identity axiom *id* and the *Cut* rule are referred to as the *identity group*; they reflect the reflexivity and transitivity respectively of entailment. All the other rules are left (L) introduction rules, introducing the active formula on the left (antecedent) of the conclusion, or right (R) introduction rules, introducing the active formula on the right (succedent) of the conclusion.

The rules W (weakening), C (contraction) and P (permutation) are referred to as *structural* rules; they apply to properties of all formulas with respect to the metalinguistic comma (conjunction in the antecedent, disjunction in the succedent). Weakening corresponds to the *monotonicity* of classical logic: that conjoining premises, or disjoining conclusions, preserves validity. Contraction and Permutation correspond to the idempotency and commutativity of conjunction in the antecedent and disjunction in the succedent. They permit each side of a succedent to be read, if we wish, as a set rather than a list, of formulas.

Then there are the *logical* rules, dealing with the connectives themselves. For each connective there is a left rule and a right rule introducing single principal connective occurrences in the active formula in the antecedent (L) or the succedent (R) of the conclusion respectively.

A sequent which has a proof is a *theorem*. The sequent calculus is *sound* (every theorem is a valid sequent) and *complete* (every valid sequent is a theorem).

All the rules except *Cut* have the property that all the formulas in the premises are either in the conclusion (the *side*-formulas in the contexts $\Gamma_{(i)}/\Delta_{(i)}$, and the active formulas of structural rules), or else are the (immediate) subformulas of the active formula (in the logical rules). In the Cut rule, the *Cut formula* A is a

$$\frac{}{A \Rightarrow A}\,id \qquad \frac{\Gamma_1 \Rightarrow \Delta_1, A \qquad A, \Gamma_2 \Rightarrow \Delta_2}{\Gamma_1, \Gamma_2 \Rightarrow \Delta_1, \Delta_2}\,Cut$$

$$\frac{\Delta_1, \Delta_2 \Rightarrow \Delta}{\Delta_1, A, \Delta_2 \Rightarrow \Delta}\,WL \qquad \frac{\Delta \Rightarrow \Delta_1, \Delta_2}{\Delta \Rightarrow \Delta_1, A, \Delta_2}\,WR$$

$$\frac{\Delta_1, A, A, \Delta_2 \Rightarrow \Delta}{\Delta_1, A, \Delta_2 \Rightarrow \Delta}\,CL \qquad \frac{\Delta \Rightarrow \Delta_1, A, A, \Delta_2}{\Delta \Rightarrow \Delta_1, A, \Delta_2}\,CR$$

$$\frac{\Delta_1, A, B, \Delta_2 \Rightarrow \Delta}{\Delta_1, B, A, \Delta_2 \Rightarrow \Delta}\,PL \qquad \frac{\Delta \Rightarrow \Delta_1, A, B, \Delta_2}{\Delta \Rightarrow \Delta_1, B, A, \Delta_2}\,PR$$

$$\frac{\Gamma \Rightarrow A, \Delta}{\neg A, \Gamma \Rightarrow \Delta}\,\neg L \qquad \frac{\Delta, A \Rightarrow \Gamma}{\Delta \Rightarrow \neg A, \Gamma}\,\neg R$$

$$\frac{\Delta_1, A, B, \Delta_2 \Rightarrow \Delta}{\Delta_1, A \wedge B, \Delta_2 \Rightarrow \Delta}\,\wedge L \qquad \frac{\Delta \Rightarrow \Delta_1, A, \Delta_2 \qquad \Delta \Rightarrow \Delta_1, B, \Delta_2}{\Delta \Rightarrow \Delta_1, A \wedge B, \Delta_2}\,\wedge R$$

$$\frac{\Delta_1, A, \Delta_2 \Rightarrow \Delta \qquad \Delta_1, B, \Delta_2 \Rightarrow \Delta}{\Delta_1, A \vee B, \Delta_2 \Rightarrow \Delta}\,\vee L \qquad \frac{\Delta \Rightarrow \Delta_1, A, B, \Delta_2}{\Delta \Rightarrow \Delta_1, A \vee B, \Delta_2}\,\vee R$$

$$\frac{\Gamma \Rightarrow A \qquad \Delta_1, B, \Delta_2 \Rightarrow \Delta}{\Delta_1, \Gamma, A \rightarrow B, \Delta_2 \Rightarrow \Delta} \rightarrow L \qquad \frac{\Delta_1, A, \Delta_2 \Rightarrow \Gamma_1, B, \Gamma_2}{\Delta_1, \Delta_2 \Rightarrow \Gamma_1, A \rightarrow B, \Gamma_2} \rightarrow R$$

Figure 2. Sequent calculus for classical propositional logic

new unknown reading from conclusion to premises. Gentzen proved as his *Hauptsatz* (main clause) that every proof has a Cut-free equivalent (Cut-elimination). Gentzen's Cut-elimination theorem has as a corollary that every theorem has a proof containing only its subformulas (the *subformula property*), namely any of its Cut-free proofs.

Computationally, the contraction rule is potentially problematic since it (as well as Cut) introduces material in backward-chaining proof search reading from conclusion to premises. But such Cut-free proof search becomes a decision procedure for classical propositional logic when antecedents and succedents are treated as sets. First-order classical logic is not decidable however.

2.3 Natural deduction

Intuitionistic sequent calculus is obtained from classical sequent calculus by restricting succedents to be non-plural. Observe for example that the following derivation of the law of excluded middle is then blocked, since the middle sequent has two formulas in its succedent: $A \Rightarrow A \ / \Rightarrow A, \neg A \ / \Rightarrow A \vee \neg A$. Indeed, the law of excluded middle is not derivable at all in intuitionistic logic, the theorems of which are a proper subset of those of classical logic.

Natural deduction is a single-conclusioned proof format particularly suited to intuitionistic logic. A natural deduction proof is a tree of formulas with some coindexing of leaves with dominating nodes. The leaf formulas are called *hypotheses*: *open* if not indexed, *closed* if indexed. The root of the tree is the *conclusion*: a natural deduction proof asserts that the conjunction of its open hypotheses entails its conclusion. A trivial tree consisting of a single formula is a proof (from itself, as open hypothesis, to itself, as conclusion, corresponding to the identity axiom of sequent calculus). Then the proofs of $\{\rightarrow, \wedge, \vee\}$-intuitionistic logic are those further generated by the rules in figure 3. Hypotheses become indexed (closed) when the dominating inference occurs, and any number of hypotheses (including zero) can be indexed/closed in one step, cf. the interactive effects of Weakening and Contraction.

2.4 Typed lambda calculus

The untyped lambda calculus was introduced as a model of computation by Alonzo Church. It uses a variable binding operator (the λ) to name functions, and forms the basis of functional programming languages such as LISP. It was proved equivalent to Turing machines, hence the name Church-Turing Thesis for the notion that Turing machines (and untyped lambda calculus) capture the notion of algorithm.

Church [1940] defined the simply, i.e. just functionally, typed lambda calculus, and by including logical constants, higher-order logic. Here we add also Cartesian product and disjoint union types.

(2) **Definition** (*types*)

$$\dfrac{\begin{matrix}\vdots & \vdots\\ A & A\to B\end{matrix}}{B}\,E\to \qquad \dfrac{\begin{matrix}A^i\\ \vdots\\ B\end{matrix}}{A\to B}\,I\to^i$$

$$\dfrac{\begin{matrix}\vdots\\ A\wedge B\end{matrix}}{A}\,E\wedge_1 \qquad \dfrac{\begin{matrix}\vdots\\ A\wedge B\end{matrix}}{B}\,E\wedge_2 \qquad \dfrac{\begin{matrix}\vdots & \vdots\\ A & B\end{matrix}}{A\wedge B}\,I\wedge$$

$$\dfrac{\begin{matrix} & A^i & B^i\\ \vdots & \vdots & \vdots\\ A\vee B & C & C\end{matrix}}{C}\,E\vee^i \qquad \dfrac{\begin{matrix}\vdots\\ A\end{matrix}}{A\vee B}\,I\vee_1 \qquad \dfrac{\begin{matrix}\vdots\\ B\end{matrix}}{A\vee B}\,I\vee_2$$

Figure 3. Natural deduction rules for $\{\to,\wedge,\vee\}$-intuitionistic logic

The τ set of *types* is defined on the basis of a set δ of *basic types* as follows:

$$\tau \quad ::= \quad \delta \mid \tau\to\tau \mid \tau\&\tau \mid \tau+\tau$$

(3) **Definition** (*type domains*)

The *type domain* D_τ of each type τ is defined on the basis of an assignment d of sets (*basic type domains*) to the set δ of basic types as follows:

$$\begin{array}{rcll} D_\tau & = & d(\tau) & \text{for } \tau\in\delta\\ D_{\tau_1\to\tau_2} & = & D_{\tau_2}^{D_{\tau_1}} & \text{i.e. the set of all functions from } D_{\tau_1} \text{ to } D_{\tau_2}\\ D_{\tau_1\&\tau_2} & = & D_{\tau_1}\times D_{\tau_2} & \text{i.e. } \{\langle m_1,m_2\rangle \mid m_1\in D_{\tau_1}\ \&\ m_2\in D_{\tau_2}\}\\ D_{\tau_1+\tau_2} & = & D_{\tau_1}\uplus D_{\tau_2} & \text{i.e. } (\{1\}\times D_{\tau_1})\cup(\{2\}\times D_{\tau_2}) \end{array}$$

(4) **Definition** (*terms*)

The sets Φ_τ of *terms* of type τ for each type τ are defined on the basis of a set C_τ of constants of type τ and an denumerably infinite set V_τ of variables of type τ for each type τ as follows:

$$\begin{array}{rcll} \Phi_\tau & ::= & C_\tau \mid V_\tau & \\ & & \mid (\Phi_{\tau'\to\tau}\ \Phi_{\tau'}) & \text{functional application}\\ & & \mid \pi_1\Phi_{\tau\&\tau'} \mid \pi_2\Phi_{\tau'\&\tau} & \text{projection}\\ & & \mid (\Phi_{\tau_1+\tau_2}\to V_{\tau_1}.\Phi_\tau;\ V_{\tau_2}.\Phi_\tau) & \text{case statement}\\ \Phi_{\tau\to\tau'} & ::= & \lambda V_\tau\Phi_{\tau'} & \text{functional abstraction}\\ \Phi_{\tau\&\tau'} & ::= & (\Phi_\tau,\Phi_{\tau'}) & \text{pair formation}\\ \Phi_\tau & ::= & \iota_1\Phi_{\tau+\tau'} \mid \iota_2\Phi_{\tau'+\tau} & \text{injection} \end{array}$$

$$\begin{array}{rcll}
[c]^g & = & f(c) & \text{for } c \in C_\tau \\
[x]^g & = & g(x) & \text{for } x \in V_\tau \\
[(\phi\ \psi)]^g & = & [\phi]^g([\psi]^g) & \\
[\pi_1\phi]^g & = & \text{the first projection of } [\phi]^g & \\
[\pi_2\phi]^g & = & \text{the second projection of } [\phi]^g & \\
[(\phi \to y.\psi; z.\chi]^g & = & \left\{ \begin{array}{ll} [\psi]^{(g-\{(y,g(y))\})\cup\{(y,d)\}} & \text{if } [\phi]^g = \langle 1, d\rangle \\ [\chi]^{(g-\{(z,g(z))\})\cup\{(z,d)\}} & \text{if } [\phi]^g = \langle 2, d\rangle \end{array} \right. & \\
[\lambda x_\tau \phi]^g & = & D_\tau \ni d \mapsto [\phi]^{(g-\{(x,g(x))\cup\{(x,d)\}} & \\
[(\phi, \psi)]^g & = & \langle [\phi]^g, [\psi]^g \rangle & \\
[\iota_1\phi]^g & = & \langle 1, [\phi]^g \rangle & \\
[\iota_2\phi]^g & = & \langle 2, [\phi]^g \rangle &
\end{array}$$

Figure 4. Semantics of typed lambda calculus

Each term $\phi \in \Phi_\tau$ receives a semantic value $[\phi]^g \in D_\tau$ with respect to a valuation f which is a mapping sending each constant in C_τ to an element in D_τ, and an assignment g sending each variable in V_τ to an element in D_τ, as shown in figure 4.

An occurrence of a variable x in a term is called *free* if and only if it does not fall within any part of the term of the form $\lambda x\cdot$ or $x.\cdot$; otherwise it is *bound* (by the closest variable binding operator within the scope of which it falls). The result $\phi\{\psi/x\}$ of substituting term ψ (of type τ) for variable x (of type τ) in a term ϕ is the result of replacing by ψ every free occurrence of x in ϕ. The application of the substitution is *free* if and only if no variable in ψ becomes bound in its new location. Manipulations can be pathological if substitution is not free. The laws of lambda conversion in figure 5 obtain (we omit the so-called commuting conversions for the case statement $\cdot \to x.\cdot; y.\cdot$).

The Curry-Howard correspondence [Girard *et al.*, 1989] is that intuitionistic natural deduction and typed lambda calculus are isomorphic. This *formulas-as-types* and *proofs-as-programs* correspondence takes place at the following three levels:

(5)

intuitionistic natural deduction	typed lambda calculus
formulas	types
proofs	terms
proof normalisation	lambda reduction

Overall, the laws of lambda reduction are the same as the natural deduction proof normalisations (elimination of detours) of Prawitz [1965]. For the calculi we have given we have formulas-as-types correspondence $\to\cong\to, \wedge \cong \&, \vee \cong +$. By way of illustration, the β- and η-proof reductions for conjunction are as shown in figures 6 and 7 respectively.

$$\begin{array}{rcl}
\lambda y\phi & = & \lambda x(\phi\{x/y\}) \\
\multicolumn{3}{l}{\text{if } x \text{ is not free in } \phi \text{ and } \phi\{x/y\} \text{ is free}} \\
\phi \to y.\psi; z.\chi & = & \phi \to x.(\psi\{x/y\}); z.\chi \\
\multicolumn{3}{l}{\text{if } x \text{ is not free in } \psi \text{ and } \psi\{x/y\} \text{ is free}} \\
\phi \to y.\psi; z.\chi & = & \phi \to y.\psi; x.(\chi\{x/z\}) \\
\multicolumn{3}{l}{\text{if } x \text{ is not free in } \chi \text{ and } \chi\{x/z\} \text{ is free}} \\
\multicolumn{3}{c}{\textit{α-conversion}}
\end{array}$$

$$\begin{array}{rcl}
(\lambda x\phi\ \psi) & = & \phi\{\psi/x\} \\
 & & \text{if } \phi\{\psi/x\} \text{ is free} \\
\pi_1(\phi, \psi) & = & \phi \\
\pi_2(\phi, \psi) & = & \psi \\
\iota_1\phi \to y.\psi; z.\chi & = & \psi\{\phi/y\} \\
\multicolumn{3}{l}{\text{if } \psi\{\phi/y\} \text{ is free}} \\
\iota_2\phi \to y.\psi; z.\chi & = & \chi\{\phi/z\} \\
\multicolumn{3}{l}{\text{if } \chi\{\phi/z\} \text{ is free}} \\
\multicolumn{3}{c}{\textit{β-conversion}}
\end{array}$$

$$\begin{array}{rcl}
\lambda x(\phi\ x) & = & \phi \\
\multicolumn{3}{l}{\text{if } x \text{ is not free in } \phi} \\
(\pi_1\phi, \pi_2\phi) & = & \phi \\
\multicolumn{3}{c}{\textit{η-conversion}}
\end{array}$$

Figure 5. Laws of lambda-conversion

$$\cfrac{\cfrac{\begin{array}{cc}\phi & \psi \\ \vdots & \vdots \\ A & B\end{array}}{A \wedge B}\,I\wedge}{A}\,E\wedge_1 \quad\Rightarrow\quad \begin{array}{c}\phi \\ \vdots \\ A\end{array} \qquad\qquad \cfrac{\cfrac{\begin{array}{cc}\phi & \psi \\ \vdots & \vdots \\ A & B\end{array}}{A \wedge B}\,I\wedge}{B}\,E\wedge_2 \quad\Rightarrow\quad \begin{array}{c}\psi \\ \vdots \\ B\end{array}$$

Figure 6. β-reduction for conjunction

$$\cfrac{\cfrac{\begin{array}{c}\phi \\ \vdots \\ A \wedge B\end{array}}{A}\,E\wedge_1 \quad \cfrac{\begin{array}{c}\phi \\ \vdots \\ A \wedge B\end{array}}{B}\,E\wedge_2}{A \wedge B}\,I\wedge \quad\Rightarrow\quad \begin{array}{c}\phi \\ \vdots \\ A \wedge B\end{array}$$

Figure 7. η-reduction for conjunction

In contrast to the untyped lambda calculus, the normalisation of terms (evaluation of ‘programs’) of our typed lambda calculus is *terminating*: every term reduces to a normal form in a finite number of steps.

2.5 The Lambek calculus

The Lambek calculus [Lambek, 1958] is a predecessor of linear logic [Girard, 1987]. It can be presented as a sequent calculus without structural rules and with single formulas (*types*) in the succedents. It is retrospectively identifiable as the multiplicative fragment of non-commutative intuitionistic linear logic without empty antecedents.

(6) **Definition** (*types of the Lambek calculus*)

The set $\mathcal{F}$ of *types* of the Lambek calculus is defined on the basis of a set $\mathcal{P}$ of primitive types as follows:

$$\mathcal{F} ::= \mathcal{P} \mid \mathcal{F}\bullet\mathcal{F} \mid \mathcal{F}\backslash\mathcal{F} \mid \mathcal{F}/\mathcal{F}$$

The connective $\bullet$ is called product, $\backslash$ is called under, and $/$ is called over.

(7) **Definition** (*standard interpretation of the Lambek calculus*)

A *standard interpretation of the Lambek calculus* comprises a semigroup $(L, +)$ and a function $[[\cdot]]$ mapping each type $A \in \mathcal{F}$ into a subset of L such that:

$$\begin{array}{rcl} [[A\backslash C]] & = & \{s_2 \mid \forall s_1 \in [[A]], s_1{+}s_2 \in [[C]]\} \\ [[C/B]] & = & \{s_1 \mid \forall s_2 \in [[B]], s_1{+}s_2 \in [[C]]\} \\ [[A\bullet B]] & = & \{s_1{+}s_2 \mid s_1 \in [[A]] \ \&\ s_2 \in [[B]]\} \end{array}$$

A *sequent* $\Gamma \Rightarrow A$ of the Lambek calculus comprises a finite non-empty antecedent sequence of types (*configuration*) Γ and a succedent type A. We extend the standard interpretation of types to include configurations as follows:

$$[[\Gamma_1, \Gamma_2]] \quad = \quad \{s_1{+}s_2 \mid s_1 \in [[\Gamma_1]] \ \&\ s_2 \in [[\Gamma_2]]\}$$

A sequent $\Gamma \Rightarrow A$ is *valid* iff $[[\Gamma]] \subseteq [[A]]$ in every standard interpretation. The *Lambek sequent calculus* is as shown in figure 8 where $\Delta(\Gamma)$ indicates a configuration Δ with a distinguished subconfiguration Γ. Observe that for each connective there is a left (L) rule introducing it in the antecedent, and a right (R) rule introducing it in the succedent. Like the sequent calculus for classical logic, the sequent calculus for the Lambek calculus fully modularises the inferential properties of connectives: it deals with a single occurrence of a single connective at a time.

$$\frac{}{A \Rightarrow A}\,id \qquad \frac{\Gamma \Rightarrow A \qquad \Delta(A) \Rightarrow B}{\Delta(\Gamma) \Rightarrow B}\,Cut$$

$$\frac{\Gamma \Rightarrow A \qquad \Delta(C) \Rightarrow D}{\Delta(\Gamma, A\backslash C) \Rightarrow D}\,\backslash L \qquad \frac{A, \Gamma \Rightarrow C}{\Gamma \Rightarrow A\backslash C}\,\backslash R$$

$$\frac{\Gamma \Rightarrow B \qquad \Delta(C) \Rightarrow D}{\Delta(C/B, \Gamma) \Rightarrow D}\,/L \qquad \frac{\Gamma, B \Rightarrow C}{\Gamma \Rightarrow C/B}\,/R$$

$$\frac{\Delta(A, B) \Rightarrow D}{\Delta(A\bullet B) \Rightarrow D}\,\bullet L \qquad \frac{\Gamma \Rightarrow A \qquad \Delta \Rightarrow B}{\Gamma, \Delta \Rightarrow A\bullet B}\,\bullet R$$

Figure 8. Lambek sequent calculus

(8) **Proposition** (*soundness of the Lambek calculus*)

In the Lambek calculus, every theorem is valid.

Proof. By induction on the length of proofs. ■

(9) **Theorem** (*completeness of the Lambek calculus*)

In the Lambek calculus, every valid sequent is a theorem.

Proof. [Buszkowski, 1986]. ■

Soundness and completeness mean that the Lambek calculus is satisfactory as a logical theory.

(10) **Theorem** (*Cut-elimination for the Lambek calculus*)

In the Lambek calculus, every theorem has a Cut-free proof.

Proof. [Lambek, 1958]. ■

(11) **Corollary** (*subformula property for the Lambek calculus*)

In the Lambek calculus, every theorem has a proof containing only its subformulas.

Proof. Every rule except Cut has the property that all the types in the premises are either in the conclusion (side formulas) or are the immediate subtypes of the active formula, and Cut itself is eliminable. ■

(12) **Corollary** (*decidability of the Lambek calculus*)

In the Lambek calculus, it is decidable whether a sequent is a theorem.

Proof. By backward-chaining in the finite Cut-free sequent search space. ■

3 FORMAL SYNTAX AND FORMAL SEMANTICS

3.1 Transformational grammar

Noam Chomsky's short book *Syntactic Structures* published in 1957 revolutionised linguistics. It argued that the grammar of natural languages could be characterised by formal systems, so-called generative grammars, as models of the human capacity to produce and comprehend unboundedly many sentences, regarded as strings. There, and in subsequent articles, he defined a hierarchy of grammatical production/rewrite systems, the Chomsky hierarchy, comprising type 3 (regular), type 2 (context-free), type 1 (context-sensitive) and type 0 (unrestricted/Turing powerful) grammars. He argued formally that regular grammars cannot capture the structure of English, and informally that context-free grammars, even if they could in principle define the string-set of say English, could not do so in a scientifically satisfactory manner. Instead he forwarded *transformational grammar* in which a deep structure phrase-structure base component feeds a system of 'transformations' to deliver surface syntactic structures.

To emphasize the link with logical formal systems, we describe here a 'proto-transformational grammar' like sequent calculus in which base component rules are axiomatic rules and transformational rules are structural rules.

Let there be *modes* n (nominal), v (verbal), a (adjectival) and p (prepositional). Let there be *types* PN (proper name), NP (noun phrase), VP (verb phrase), TV (transitive verb), COP (copula), TPSP (transitive past participle), Pby (preposition *by*), CN (count noun), ... Let a *configuration* be an ordered tree the leaves of which are labelled by types and the mothers of which are labelled by modes. Then we may have base component rules:

(13)
$$\begin{array}{l} [_vTV, NP] \Rightarrow VP \\ [_vNP, VP] \Rightarrow S \\ [_nDET, CN] \Rightarrow NP \\ [_nPN] \Rightarrow NP \end{array}$$

There may be the following agentive passive transformational rule:

(14)
$$\frac{[_v[_n\Gamma_1], [_vTV, [_n\Gamma_2]]] \Rightarrow S}{[_v[_n\Gamma_2], [_vCOP, TPSP, [_pPby, [_n\Gamma_1]]]] \Rightarrow S}\, Agpass$$

Then the sentence form for *The book was read by John* is derived as shown in figure 9. This assumes lexical insertion after derivation whereas transformational grammar had lexical insertion in the base component, but the proto-

$$\dfrac{\dfrac{\dfrac{[_nDET, CN] \Rightarrow NP \qquad [_vTV, NP] \Rightarrow VP}{[_vTV, [_nDET, CN]] \Rightarrow VP}\,Cut \qquad \dfrac{[_nPN] \Rightarrow NP \qquad [_vNP, VP] \Rightarrow S}{[_v[_nPN], VP]] \Rightarrow S}\,Cut}{[_v[_nPN], [_vTV, [_nDET, CN]]] \Rightarrow S}\,Cut}{[v[_nDET, CN], [_vCOP, TPSP, [_pPby, [_nPN]]]] \Rightarrow S}\,Agpass$$

Figure 9. Proto-transformational derivation of agentive passivization

transformational formulation shows how transformations could have been seen as structural rules of sequent calculus.

3.2 Montague grammar

Montague [1970b; 1970a; 1973] were three papers defining and illustrating a framework for grammar assigning logical semantics. The contribution was revolutionary because the general belief at the time was that the semantics of natural language was beyond the reaches of formalisation.

'Universal Grammar' (UG) formulated syntax and semantics as algebras, with compositionality a homomorphism from the former to the latter. The semantic algebra consisted of a hierarchy of function spaces built over truth values, entities, and possible worlds.

'English as a Formal Language' (EFL) gave a denotational semantics to a fragment of English according to this design. Since denotation was to be defined by induction on syntactic structure in accordance with compositionality as homomorphism, syntax was made an absolutely free algebra using various kinds of brackets, with a '(dis)ambiguating relation' erasing the brackets and relating these to ambiguous forms.

'The Proper Treatment of Quantification' (PTQ) relaxed the architecture to generate directly ambiguous forms, allowing itself to assume a semantic representation language known as (Montague's higher order) Intensional Logic (IL) and including an ingenious rule of term insertion (S14) for quantification (and pronoun binding) which is presumably the origin of the paper's title.

4 GRAMMATICAL FRAMEWORKS

4.1 Lexical-Functional Grammar

The formal theory of Lexical-Functional Grammar [Kaplan and Bresnan, 1982; Bresnan, 2001] is a framework which takes as primitive the grammatical functions of traditional grammar (subject, object, ...). It separates, amongst other levels of representation, *constituent-structure* (c-structure) which represents category

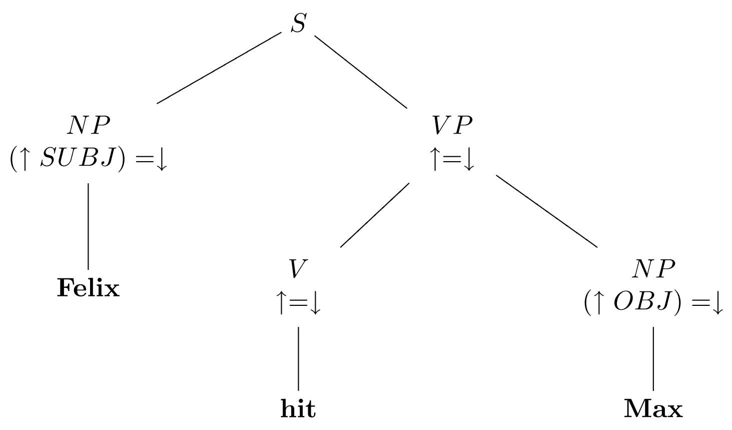

Figure 10. LFG c-structure for *Felix hit Max*

and ordering information, and *functional-structure* (f-structure) which represents grammatical functions and which feeds semantic interpretation.

The phrase-structural c-structure rules are productions with regular expressions on their right-hand side, and which have have 'functional annotations' defining the correspondence between c-structure nodes and their f-structure counterparts, which are attribute-value matrices providing the solution to the c-structure constraints. The functional annotations, which also appear in lexical entries, are equations containing ↑ meaning my mother's f-structure and ↓ meaning my own f-structure:

(15) a. $$\textbf{hit} : V,\ \begin{array}{lcl} (\uparrow\ TENSE) & = & PAST \\ (\uparrow\ PRED) & = & \text{‘}hit\langle(SUBJ, OBJ)\rangle\text{’} \end{array}$$

b. $$S \rightarrow \begin{array}{cc} NP & VP \\ (\uparrow\ SUBJ) =\downarrow & \uparrow=\downarrow \end{array}$$

$$VP \rightarrow \begin{array}{cc} V & NP \\ \uparrow=\downarrow & (\uparrow\ OBJ) =\downarrow \end{array}$$

Then *Felix hit Max* receives the c-structure and f-structure in figures 10 and 11 respectively.

One of the first LFG analyses was the lexical treatment of passive in Bresnan [1982]. She argued against its treatment in syntax, as since Chomsky [1957]. Since around 1980 there has been a multiplication of grammar formalisms also treating other local constructions such as control by lexical rule. More recently Bresnan's LFG treatment of lexical rules such as passive have been refined under 'lexical mapping theory' with a view to universality.

Kaplan and Zaenan [1989] propose to treat long-distance dependencies in LFG by means of functional annotations extended with regular expressions: so-called

$$\begin{bmatrix} PRED & \text{‘}hit\langle(SUBJ, OBJ)\rangle\text{’} \\ SUBJ & \begin{bmatrix} PRED & \text{‘}Felix\text{’} \\ PER & 3 \\ NUM & SG \end{bmatrix} \\ TENSE & PAST \\ OBJ & \begin{bmatrix} PRED & \text{‘}Max\text{’} \\ PER & 3 \\ NUM & SG \end{bmatrix} \end{bmatrix}$$

Figure 11. LFG f-structure for *Felix hit Max.*

functional uncertainty. Consider an example of topicalization:

(16) Mary John claimed that Bill said that Henry telephoned.

They propose to introduce the topic *Mary* and establish the relation between this and *telephoned* by a rule such as the following:

(17) $$S' \rightarrow \begin{array}{c} XP \\ (\uparrow\ TOPIC) =\downarrow \\ (\uparrow\ TOPIC) = (\uparrow\ COMP^{*}\ OBJ) \end{array} \quad S$$

Here, * is the Kleene star operator, meaning an indefinite number of iterations.

To deliver logical semantics in LFG, Dalrymple [1999] adopts linear logic as a ‘glue language’ to map f-structure to *semantic-structure* (s-structure), for example to compute alternative quantifier scopings under Curry-Howard proofs-as-programs. The multistratality of the c/f/s-structure of LFG is seen by its proponents as a strength in that it posits a level of f(unctional)-structure in relation to which universalities can be posited. But consider the non-standard constituent conjunts and coordination in say right node raising (RNR):

(18) John likes and Mary dislikes London.

It seems that in view of its traditional c(onstituent)-structure LFG could not characterise such a construction without treating *likes* in c-structure as an intransitive verb. How could this be avoided?

4.2 Generalized Phrase Structure Grammar

Generalized Phrase Structure Grammar (GPSG; [Gazdar, 1981; Gazdar *et al.*, 1985]) aimed to develop a congenial phrase structure formalism without exceeding context-free generative power.

Let there be a basic context-free grammar:

(19) $S \rightarrow NP\ VP$
$VP \rightarrow TV\ NP$
$VP \rightarrow SV\ CP$
$CP \rightarrow C\ S$

(20) **Bill** := NP
claimed := SV
Henry := NP
John := NP
Mary := NP
said := SV
telephoned := TV
that := C

To treat unbounded dependencies, Gazdar [1981] proposed to extend categories with 'slash' categories B/A signifying a B 'missing' an A. Then further rules may be derived from basic rules by *metarules* such as the following:[1]

(21) $\dfrac{B \to \Gamma\ A}{B/A \to \Gamma}$ slash introduction $\qquad \dfrac{C \to \Gamma\ B}{C/A \to \Gamma\ B/A}$ slash propagation

Then assuming also a topicalisation rule (23), left extraction such as (22) is derived as shown in figure 12.

(22) Mary John claimed that Henry telephoned.

(23) $S' \to XP\ S/XP$

The phrase structure schema (24) will generate standard constituent coordination.

(24) $X \to X\ CRD\ X$

But furthermore, if we assume the slash elimination rule (25), non-standard constituent RNR coordination such as (18) is also generated; see figure 13.

(25) $B \to B/A\ A$

However, if GPSG needs to structure categories with slashes to deal with extraction and coordination, why not structure categories also to express subcategorization valencies?

4.3 Head-driven Phrase Structure Grammar

The framework of Head-driven Phrase Structure Grammar (HPSG; [Pollard and Sag, 1987; Pollard and Sag, 1994]) represents all linguistic objects as attribute-value matrices: labelled directed (but acyclic) graphs. Like LFG and GPSG, HPSG is a unification grammar, meaning that the matching of formal and actual parameters is not required to be strict identity, but merely compatibility/unifiability.

[1]Gazdar *et al.* [1985] delegated slash propagation to principles of feature percolation, but the effect is the same.

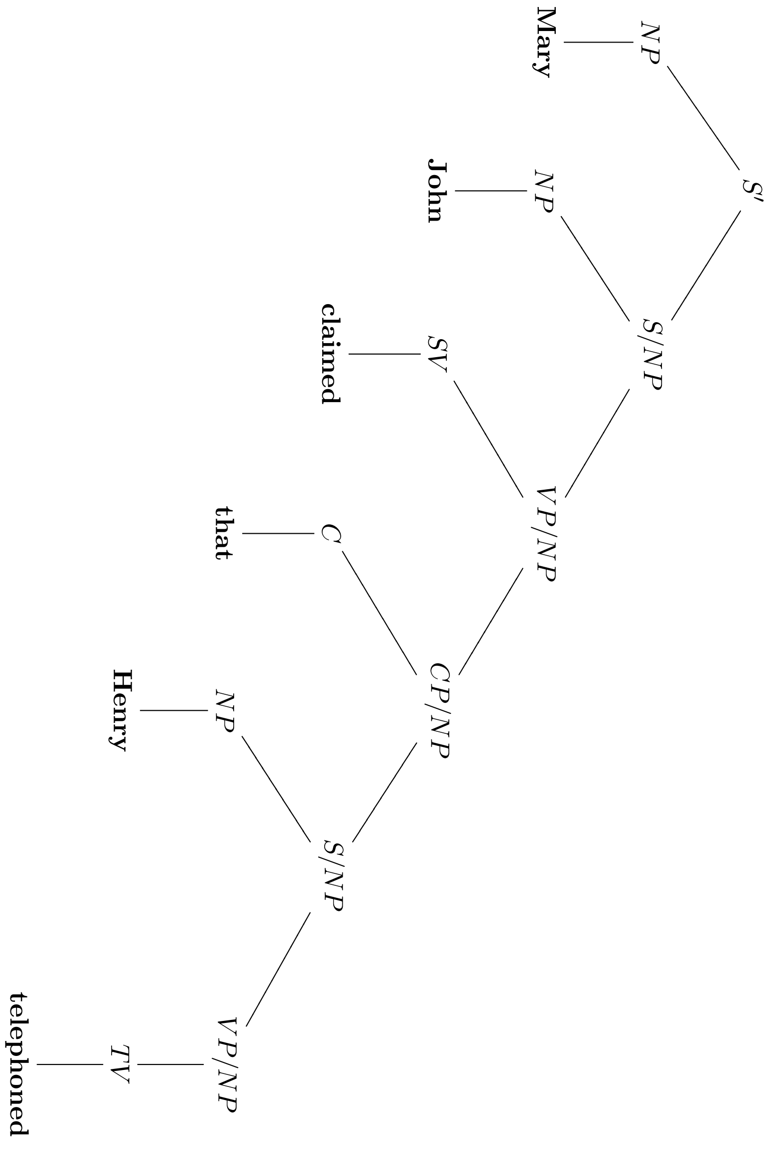

Figure 12. Left extraction in GPSG

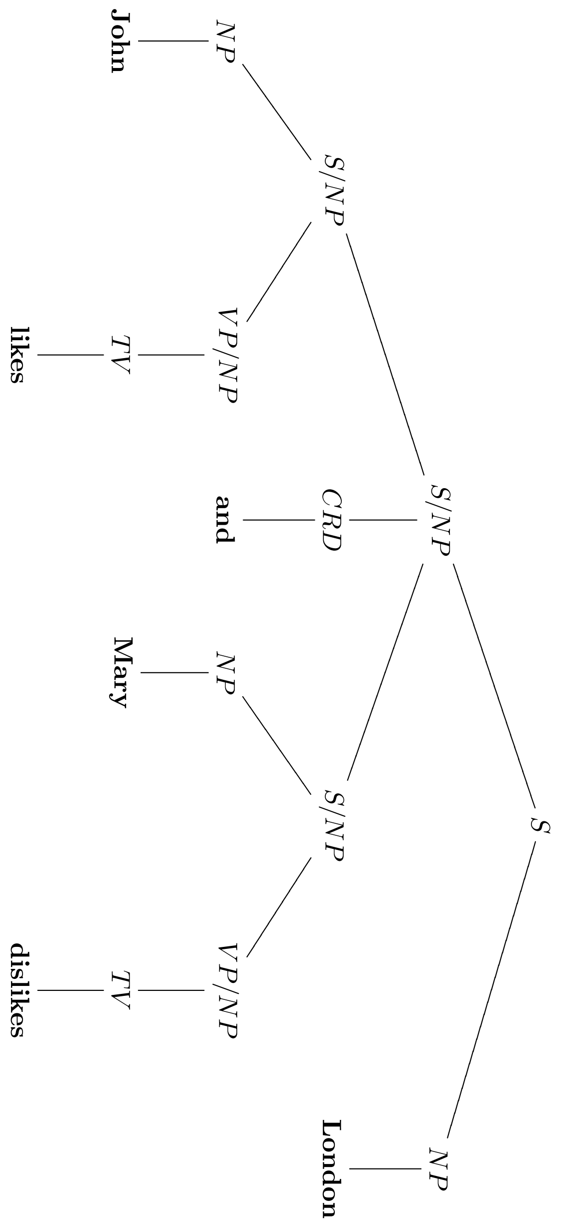

Figure 13. Right node raising in GPSG

The form (signifier) associated with a sign is represented as the value of a PHON(OLOGY) attribute and the meaning (signified) associated with a sign as the value of a CONTENT attribute. Subcategorization is projected from a lexical stack of valencies on heads: the stack-valued SUBCAT(EGORIZATION) feature (there are additional stack-valued features such as SLASH, for gaps). Thus there is a subcategorization principle:

(26) $H[SUBCAT\ \langle\ldots\rangle] \rightarrow H[SUBCAT\ \langle X,\ldots\rangle], X$

where the phonological order is to be encoded by linear precedence rules, or by reentrancy in PHON attributes. See figure 14. HPSG is entirely encoded as typed feature logic [Kasper and Rounds, 1990; Johnson, 1991; Carpenter, 1992]. The grammar is a system of constraints, and the signs in the language model defined are those which satisfy all the constraints.

HPSG can treat left extraction and right node raising much as in GPSG, but what about left node raising (LNR) non-standard constituent coordination such as the following?

(27) Mary gave John a book and Sue a record.

Since it is the *head* which is left node raised out of the coordinate structure in LNR it is unclear how to categorize the conjuncts and derive them as constituents in *Head*-driven Phrase Structure Grammar.

4.4 Combinatory Categorial Grammar

Combinatory Categorial Grammar (CCG; [Steedman, 1987; Steedman, 2000]) extends the categorial grammar of Adjukiewicz [1935] and Bar-Hillel [1953] with a small number of additional combinatory schemata. Let there be forward- and backward-looking types B/A and $A\backslash B$ defined recursively as in the Lambek calculus.[2] Then the classical cancellation schemata are:

(28) $>: B/A, A \Rightarrow B$
$<: A, A\backslash B \Rightarrow B$

Thus:

(29)
$$\cfrac{\cfrac{\textbf{Felix}}{N} \qquad \cfrac{\cfrac{\textbf{hit}}{(N\backslash S)/N} \quad \cfrac{\textbf{Max}}{N}}{N\backslash S}\,>}{S}\,<$$

CCG adds combinatory schemata such as the following:

[2]CCG writes $B\backslash A$ to mean "looks for an A to the left to form a B", but we keep to the original Lambek notation here.

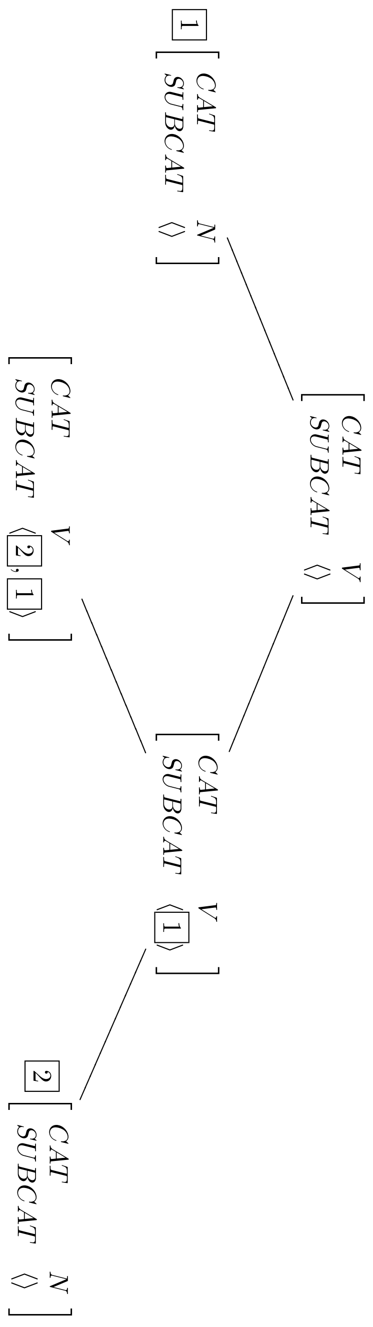

Figure 14. HPSG derivation of *Felix hit Max*.

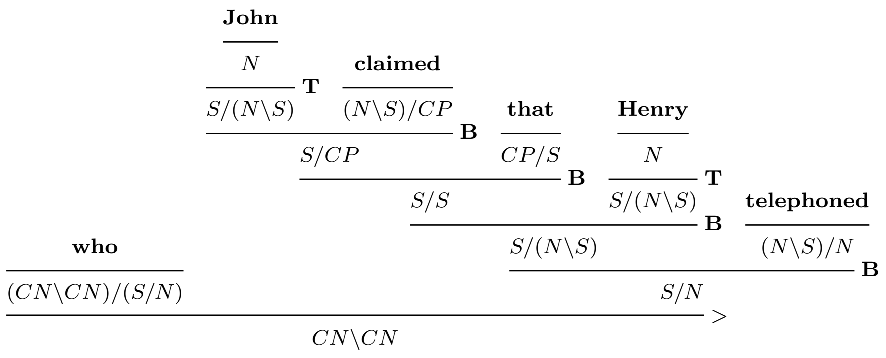

Figure 15. Left extraction in CCG

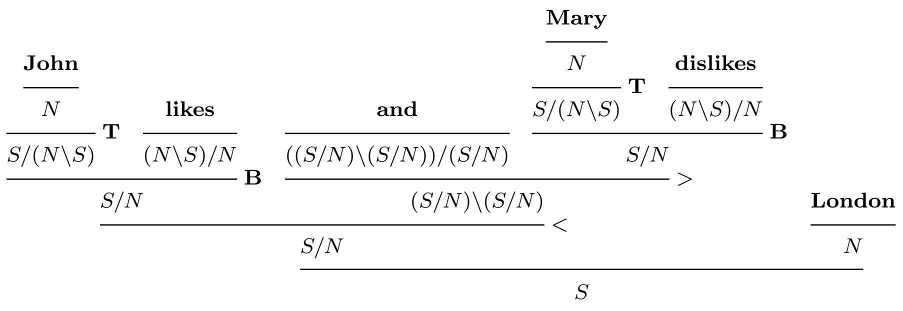

Figure 16. Right node raising in CCG

(30) $\mathbf{T} : A \Rightarrow B/(A\backslash B)$
$\mathbf{B} : C/B, B/A \Rightarrow C/A$

(The combinator names define the associated semantics: $\mathbf{T} = \lambda x \lambda y(y\ x); \mathbf{B} = \lambda x \lambda y \lambda z(x\ (y\ z))$.) This allows left extraction and right node raising to be derived as shown in figures 15 and 16 [Steedman, 1987].

Dowty [1988] observes that backward counterparts of (30) derive left node raising; see figure 17.

(31) $\mathbf{T} : A \Rightarrow (B/A)\backslash B$
$\mathbf{B} : A\backslash B, B\backslash C \Rightarrow A\backslash C$

However, multiple right node raising will require additional type shifts:

(32) a. John gave and Mary sent a book to Bill.
$N, ((N\backslash S)/PP)/N \Rightarrow (S/PP)/N$

b. John bet and Mary also wagered Sue \$10 that it would rain.
$N, (((N\backslash S)/CP)/N)/N \Rightarrow ((S/CP)/N)/N$

Likewise, combined left and right node raising:

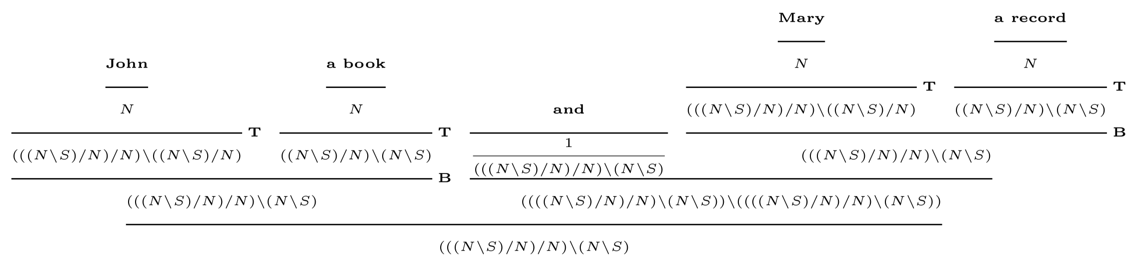
John
N
$(((N\backslash S)/N)/N)\backslash((N\backslash S)/N)$ T
a book
N
$((N\backslash S)/N)\backslash(N\backslash S)$ T
$(((N\backslash S)/N)/N)\backslash(N\backslash S)$ B
and
1
$(((N\backslash S)/N)/N)\backslash(N\backslash S)$
Mary
N
$(((N\backslash S)/N)/N)\backslash((N\backslash S)/N)$ T
a record
N
$((N\backslash S)/N)\backslash(N\backslash S)$ T
$(((N\backslash S)/N)/N)\backslash(N\backslash S)$ B
$((((N\backslash S)/N)/N)\backslash(N\backslash S))\backslash((((N\backslash S)/N)/N)\backslash(N\backslash S))$
$(((N\backslash S)/N)/N)\backslash(N\backslash S)$

Figure 17. Left node raising in CCG

(33) John gave Mary a book and John a record about bird song.
$N, N/PP \Rightarrow ((((N\backslash S)/N)/N)\backslash(N\backslash S))/PP$

It seems unfortunate to have to posit new combinatory schemata adhoc on an example-by-example basis. All the above type shifts are derivable in the Lambek calculus, and type-logical categorial grammar takes that as its basis.

4.5 Type Logical Categorial Grammar

The framework of Type Logical Categorial Grammar (TLCG; [van Benthem, 1991; Morrill, 1994; Moortgat, 1997]) is an enrichment of Lambek calculus with additional connectives, preserving the character of the latter as a non-commutative intuitionistic linear logic. For our illustration here, let the set $\mathcal{F}$ of syntactic types be defined on the basis of a set $\mathcal{A}$ of primitive syntactic types as follows:

(34) $\mathcal{F} ::= \mathcal{A} \mid [\,]^{-1}\mathcal{F} \mid \langle\rangle\mathcal{F} \mid \mathcal{F} \wedge \mathcal{F} \mid \mathcal{F} \vee \mathcal{F} \mid \mathcal{F}\backslash\mathcal{F} \mid \mathcal{F}/\mathcal{F} \mid \mathcal{F}\bullet\mathcal{F} \mid \triangle\mathcal{F}$

We define sequent antecedents as well-bracketed sequences of types; neither sequents nor brackets may be empty. The sequent calculus is as shown in figure 18.

The connectives $\langle\rangle$ and $[\,]^{-1}$ are bracket operators [Morrill, 1994; Moortgat, 1995]. They may be used to project bracketed domains; in our examples these will be domains which are islands to extraction. We refer to this as structural inhibition since the brackets may block association and permutation. $\wedge$ and $\vee$ are additives in the terminology of linear logic. They can express polymorphism [Lambek, 1961; Morrill, 1990]. The Lambek connectives $\backslash, \bullet, /$ are multiplicatives. The structural operator or modality $\triangle$ [Barry *et al.*, 1991] licenses the structural rule of permutation and is inspired by the exponentials of linear logic.

Consider a mapping as follows from our TLG syntactic types to the types of the lambda calculus of section 2.4:

(35)
$$\begin{array}{rcl}
T(\langle\rangle A) & = & T(A) \\
T([\,]^{-1}A) & = & T(A) \\
T(A \wedge B) & = & T(A)\&T(B) \\
T(A \vee B) & = & T(A) + T(B) \\
T(A\bullet B) & = & T(A)\&T(B) \\
T(A\backslash C) & = & T(A) \rightarrow T(C) \\
T(C/B) & = & T(B) \rightarrow T(C) \\
T(\triangle A) & = & T(A)
\end{array}$$

Under this mapping, every TLG proof has a reading as a proof in $\{\rightarrow, \wedge, \vee\}$-intuitionistic logic. This categorial semantics is called Curry-Howard type-logical semantics. Lambda-term lexical semantics is substituted into the lambda reading of a syntactic proof/derivation to deliver the semantics of derived expressions.

Let there be the lexicon in figure 19. Then *Felix hit Max* is derived as follows with semantics (*hit max felix*):

$$\frac{}{A \Rightarrow A}\,id \qquad \frac{\Gamma \Rightarrow A \qquad \Delta(A) \Rightarrow B}{\Delta(\Gamma) \Rightarrow B}\,Cut$$

$$\frac{\Delta(A) \Rightarrow C}{\Delta([[\,]^{-1}A]) \Rightarrow C}\,[\,]^{-1}L \qquad \frac{[\Gamma] \Rightarrow A}{\Gamma \Rightarrow [\,]^{-1}A}\,[\,]^{-1}R$$

$$\frac{\Delta([A]) \Rightarrow C}{\Delta(\langle\rangle A) \Rightarrow C}\,\langle\rangle L \qquad \frac{\Gamma \Rightarrow A}{[\Gamma] \Rightarrow \langle\rangle A}\,\langle\rangle R$$

$$\frac{\Delta(A) \Rightarrow C}{\Delta(A \wedge B) \Rightarrow C}\,\wedge L \qquad \frac{\Delta(B) \Rightarrow C}{\Delta(A \wedge B) \Rightarrow C}\,\wedge L \qquad \frac{\Delta \Rightarrow A \qquad \Delta \Rightarrow B}{\Delta \Rightarrow A \wedge B}\,\wedge R$$

$$\frac{\Delta(A) \Rightarrow C \qquad \Delta(B) \Rightarrow C}{\Delta(A \vee B) \Rightarrow C}\,\vee L \qquad \frac{\Delta \Rightarrow A}{\Delta \Rightarrow A \vee B}\,\vee R \qquad \frac{\Delta \Rightarrow B}{\Delta \Rightarrow A \vee B}\,\vee R$$

$$\frac{\Gamma \Rightarrow A \qquad \Delta(C) \Rightarrow D}{\Delta(\Gamma, A\backslash C) \Rightarrow D}\,\backslash L \qquad \frac{A, \Gamma \Rightarrow C}{\Gamma \Rightarrow A\backslash C}\,\backslash R$$

$$\frac{\Gamma \Rightarrow B \qquad \Delta(C) \Rightarrow D}{\Delta(C/B, \Gamma) \Rightarrow D}\,/L \qquad \frac{\Gamma, B \Rightarrow C}{\Gamma \Rightarrow C/B}\,/R$$

$$\frac{\Delta(A, B) \Rightarrow D}{\Delta(A{\bullet}B) \Rightarrow D}\,{\bullet}L \qquad \frac{\Gamma \Rightarrow A \qquad \Delta \Rightarrow B}{\Gamma, \Delta \Rightarrow A{\bullet}B}\,{\bullet}R$$

$$\frac{\Delta(A) \Rightarrow B}{\Delta(\triangle A) \Rightarrow B}\,\triangle L \qquad \frac{\triangle\Delta \Rightarrow B}{\triangle\Delta \Rightarrow \triangle B}\,\triangle R \qquad \frac{\Delta(A, B) \Rightarrow C}{\Delta(B, A) \Rightarrow C}\,\triangle P, A \text{ or } B\ \triangle\text{-ed}$$

Figure 18. TLCG sequent calculus

and	$-$	$\lambda x \lambda y[y \wedge x]$
	$:=$	$(S\backslash[\,]^{-1}S)/S$
annoys	$-$	$annoy$
	$:=$	$(\langle\rangle CP\backslash S)/N$
felix	$-$	f
	$:=$	N
from	$-$	$\lambda x((from_{adn}\ x), (from_{adv}\ x))$
	$:=$	$((CN\backslash CN) \wedge ((N\backslash S)\backslash(N\backslash S)))/N$
hit	$-$	hit
	$:=$	$(N\backslash S)/N$
is	$-$	$\lambda x \lambda y(x \to z, [y = z]; w.((w\ \lambda u[u = y])\ y))$
	$:=$	$(N\backslash S)/(N \vee (CN\backslash CN))$
max	$-$	m
	$:=$	N
that	$-$	$\lambda x \lambda y \lambda z[(y\ z) \wedge (x\ z)]$
	$:=$	$(CN\backslash CN)/(S/\triangle N)$
that	$-$	$\lambda x x$
	$:=$	CP/S

Figure 19. TLG lexicon

(36)
$$\dfrac{N \Rightarrow N \qquad \dfrac{N \Rightarrow N \qquad S \Rightarrow S}{N, N\backslash S \Rightarrow S}\,\backslash L}{N, (N\backslash S)/N, N \Rightarrow S}\,/L$$

Left extraction such as *man that John thinks Mary loves* is derived as shown in figure 20 with semantics $\lambda z[(man\ z) \wedge (think\ (love\ z\ m)\ j)]$.

The role of the permutation modality is to allow medial extraction such as *man that Mary met today* as follows, where ADN and ADV abbreviate CN\CN and (N\S)\(N\S) respectively:

(37)
$$\dfrac{\dfrac{\dfrac{\dfrac{N, (N\backslash S)/N, N, ADV \Rightarrow S}{N, (N\backslash S)/N, \triangle N, ADV \Rightarrow S}\,\triangle L}{N, (N\backslash S)/N, ADV, \triangle N \Rightarrow S}\,\triangle P}{N, (N\backslash S)/N, ADV \Rightarrow S/\triangle N}\,/R \qquad ADN \Rightarrow ADN}{ADN/(S/\triangle N), N, (N\backslash S)/N, ADV \Rightarrow ADN}\,/L$$

The use of the bracket operators in figure 19 marks coordinate structures and sentential subjects as islands:

(38) a. *man that John likes Suzy and Mary loves
b. *man who that Mary likes annoys Bill

$$\dfrac{\dfrac{\dfrac{N \Rightarrow N}{\triangle N \Rightarrow N}\,\triangle L \qquad \dfrac{N \Rightarrow N \qquad \dfrac{S \Rightarrow S \qquad \dfrac{N \Rightarrow N \qquad S \Rightarrow S}{N, N\backslash S \Rightarrow S}\,\backslash L}{N, (N\backslash S)/S, S \Rightarrow S}\,/L}{N, (N\backslash S)/S, N, N\backslash S \Rightarrow S}\,\backslash L}{\dfrac{N, (N\backslash S)/S, N, (N\backslash S)/N, \triangle N \Rightarrow S}{N, (N\backslash S)/S, N, (N\backslash S)/N \Rightarrow S/\triangle N}\,/R}\,/L \qquad \dfrac{CN \Rightarrow CN \qquad CN \Rightarrow CN}{CN, CN\backslash CN \Rightarrow CN}\,\backslash L}{CN, (CN\backslash CN)/(S/\triangle N), N, (N\backslash S)/S, N, (N\backslash S)/N \Rightarrow CN}\,/L$$

Figure 20. Left extraction in TLG

First, note how bracketed domains are induced. For, say, *Mary talks and Suzy talks*:

(39)
$$\dfrac{N, N\backslash S \Rightarrow S \qquad \dfrac{N, N\backslash S \Rightarrow S \qquad \dfrac{S \Rightarrow S}{[[\,]^{-1}S] \Rightarrow S}\,[\,]^{-1}L}{[N, N\backslash S, S\backslash[\,]^{-1}S] \Rightarrow S}\,\backslash L}{[N, N\backslash S, (S\backslash[\,]^{-1}S)/S, N, N\backslash S] \Rightarrow S}\,/L$$

And for, say, *That Mary talks annoys Bill*:

(40)
$$\dfrac{N \Rightarrow N \qquad \dfrac{\dfrac{CP/S, N, N\backslash S \Rightarrow CP}{[CP/S, N, N\backslash S] \Rightarrow \langle\rangle CP}\,\langle\rangle R \qquad S \Rightarrow S}{[CP/S, N, N\backslash S], \langle\rangle CP\backslash S \Rightarrow S}\,\backslash L}{[CP/S, N, N\backslash S], (\langle\rangle CP\backslash S)/N, N \Rightarrow S}\,/L$$

Second, observe that the coordinator type $(\mathrm{S}\backslash[\,]^{-1}\mathrm{S})/\mathrm{S}$ and the sentential subject verb type $(\langle\rangle\mathrm{CP}\backslash\mathrm{S})/\mathrm{N}$ will block the overgeneration in (38) because the brackets projected will block the conditionalised gap subtype from associating and permuting into the islands.

5 WHY MIGHT GRAMMAR AND PROCESSING BE LOGICAL?

The formalisms we have considered have particular empirical and/or technical characteristic features. LFG: grammatical functions; GPSG: context-freeness; HPSG: heads and feature logic; CCG: combinators; TLCG: type logic. We have traced a path leading from each to the next. Young science does not readily renounce treasured key concepts, but our own 'logical conclusion' of logical grammar, indeed formal grammar, is enrichment of non-commutative intuitionistic linear logic. This latter was already in existence at the time of Syntactic Structures in the form of the Lambek calculus.

One may question whether formal grammar is a good linguistic program at all. All grammars leak, and logical semantics has little to say about allegory, metaphor, or poetry. But that is not to say that grammaticality and truth conditions are not real. It seems to me that formal grammar has been tried but not really tested: after an initial euphoria, the going got heavy. But we have an opportunity to develop linguistic formalism in the paradigm of modern mathematical logic.

We conclude by considering why it might have been expected that grammar would take the form of a logic and processing would take the form of deduction. We consider the engineering perspective of language engineering and the scientific perspective of cognitive science.

On the engineering perspective, linguistic formalisms can be seen as construction kits for building formal languages which are like, or resemble, fragments of natural language. The charting of natural language syntax and semantics is then a massive information engineering task. It seems likely that logic would be a helpful tool/organisational principle for this. Indeed, if the mapping strategy were not logical, on what basis could it succeed?

Automated language processing divides mainly into parsing (computing meanings/signifieds from forms/signifiers) and generation (computing forms/signifiers from meanings/signifieds). When grammar is a logic, these computational tasks take the form of parsing-as-deduction and generation-as-deduction. The setting up of grammar as logic and processing as the corresponding deduction seems to augur well for verificaton: the transparency of the correctness of processing with respect to grammar.

We know something of the macroscopic and microscopic physiology of the brain, and where the language faculty is normally located; and it is usual to view cognitive processes as computations, or at least unconscious and automatic cognition such as human language processing. We want to express our cognitive theories in terms of algorithms, representations and processes eventually implemented neuronally. But there is a huge gap in our knowledge of these concepts at the level at which we want to theorise. We do not know how to define algorithms, representations or processes except in ways dependent on arbitrary features of models of computation like neural nets, RAMs, or Turing machines which we have no basis to posit as characteristic of the levels of the higher cognitive functions of our psychological theories.

Surely an eventual understanding of such concepts will come at least partly from logic. As well as with knowledge and semantics, logic has deep relations with computation (Cut-elimination, logic programming, resolution, computation as proof-search, functional programming, computation as proof normalisation). A natural theory of algorithms, representations and processes would be one akin to logic. Pending such theory it seems reasonable to express our models of knowledge of language —grammar— at a logical level of type formulas and proof terms.

As cognitive phenomena, parsing and generation are termed comprehension and production. In TLCG syntactic structures are proofs (of grammaticality) and semantic structures are also proofs: meanings are the *way* in which grammaticality is proved. So interpreted psychologically, TLCG models production and comprehension as synthesis and analysis of *proofs*. Not just manipulation of arbitrary or language-specific structures and representations, but the resonance of *reason* in the dynamics of words and concepts: *logical* grammar and processing.

ACKNOWLEDGEMENTS

This work was partially funded by the DGICYT project TIN2008–06582–C03–01 (SESAAME-BAR). Thanks to Hiroyuki Uchida for comments. All errors are my own.

BIBLIOGRAPHY

[Ajdukiewicz, 1935] Kazimierz Ajdukiewicz. Die syntaktische konnexität. *Studia Philosophica*, 1:1–27, 1935. Translated in S. McCall, editor, 1967, *Polish Logic: 1920–1939*, Oxford University Press, Oxford, 207–231.

[Bar-Hillel, 1953] Yehoshua Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58, 1953.

[Barry *et al.*, 1991] Guy Barry, Mark Hepple, Neil Leslie, and Glyn Morrill. Proof Figures and Structural Operators for Categorial Grammar. In *Proceedings of the Fifth Conference of the European Chapter of the Association for Computational Linguistics*, Berlin, 1991.

[Bresnan, 1982] Joan Bresnan. The passive in lexical theory. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 3–86. MIT Press, Cambridge, MA, 1982.

[Bresnan, 2001] Joan Bresnan. *Lexical-Functional Syntax.* Number 16 in Blackwell Textbooks in Linguistics. Blackwell Publishers, Oxford, 2001.

[Buszkowski, 1986] W. Buszkowski. Completeness results for Lambek syntactic calculus. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 32:13–28, 1986.

[Carpenter, 1992] Bob Carpenter. *The Logic of Typed Feature Structures.* Number 32 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, 1992.

[Chomsky, 1957] Noam Chomsky. *Syntactic Structures.* Mouton, The Hague, 1957.

[Church, 1940] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[Dalrymple, 1999] Mary Dalrymple, editor. *Semantics and Syntax in Lexical Functional Grammar: The Resource Logic Approach.* MIT Press, Cambridge, MA, 1999.

[Dowty, 1988] David Dowty. Type Raising, Functional Composition, and Non-Constituent Conjunction. In Richard T. Oehrle, Emmon Bach, and Deidre Wheeler, editors, *Categorial Grammars and Natural Language Structures*, volume 32 of *Studies in Linguistics and Philosophy*, pages 153–197. D. Reidel, Dordrecht, 1988.

[Frege, 1879] G. Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.* Nebert Verlag, Halle a.S., 1879.

[Gazdar *et al.*, 1985] Gerald Gazdar, Ewan Klein, Geoffrey Pullum, and Ivan Sag. *Generalized Phrase Structure Grammar.* Basil Blackwell, Oxford, 1985.

[Gazdar, 1981] Gerald Gazdar. Unbounded dependencies and coordinate structure. *Linguistic Inquiry*, 12:155–184, 1981.

[Gentzen, 1934] G. Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210 and 405–431, 1934. Translated in M.E. Szabo, editor, 1969, *The Collected Papers of Gerhard Gentzen*, North-Holland, Amsterdam, 68–131.

[Girard *et al.*, 1989] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*, volume 7. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambridge, 1989.

[Girard, 1987] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Johnson, 1991] Mark Johnson. Features and formulae. *Computational Linguistics*, 17:131–151, 1991.

[Kaplan and Bresnan, 1982] Ronald M. Kaplan and Joan Bresnan. Lexical-functional grammar: a formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, MA, 1982. Reprinted in Mary Dalrymple, Ronald M. Kaplan, John T. Maxwell III and Annie Zaenen, editors, 1995, Formal Issues in Lexical-Functional Grammar, CSLI, Stanford, CA, 29–130.

[Kaplan and Zaenen, 1989] Ronald M. Kaplan and Annie Zaenen. Long-Distance Dependencies, Constituent Structure, and Functional Uncertainty. In Mark R. Baltin and Anthony S. Kroch, editors, *Alternative Conceptions of Phrase Structure*, pages 17–42. The University of Chicago Press, Chicago, 1989.

[Kasper and Rounds, 1990] R.T. Kasper and W.C. Rounds. The logic of unification in grammar. *Linguistics and Philosophy*, 13(1):35–58, 1990.

[Lambek, 1958] Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170, 1958. Reprinted in Buszkowski, W., W. Marciszewski, and J. van Benthem, editors, 1988, *Categorial Grammar*, Linguistic & Literary Studies in Eastern Europe volume 25, John Benjamins, Amsterdam, 153–172.

[Lambek, 1961] J. Lambek. On the Calculus of Syntactic Types. In Roman Jakobson, editor, *Structure of Language and its Mathematical Aspects, Proceedings of the Symposia in Applied Mathematics XII*, pages 166–178. American Mathematical Society, Providence, Rhode Island, 1961.

[Montague, 1970a] Richard Montague. English as a formal language. In B. Visentini et al., editor, *Linguaggi nella Società e nella Tecnica*, pages 189–224. Edizioni di Comunità, Milan, 1970. Reprinted in R.H. Thomason, editor, 1974, *Formal Philosophy: Selected Papers of Richard Montague*, Yale University Press, New Haven, 188–221.

[Montague, 1970b] Richard Montague. Universal grammar. *Theoria*, 36:373–398, 1970. Reprinted in R.H. Thomason, editor, 1974, *Formal Philosophy: Selected Papers of Richard Montague*, Yale University Press, New Haven, 222–246.

[Montague, 1973] Richard Montague. The Proper Treatment of Quantification in Ordinary English. In J. Hintikka, J.M.E. Moravcsik, and P. Suppes, editors, *Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*, pages 189–224. D. Reidel, Dordrecht, 1973. Reprinted in R.H. Thomason, editor, 1974, *Formal Philosophy: Selected Papers of Richard Montague*, Yale University Press, New Haven, 247–270.

[Montague, 1974] Richard Montague. *Formal Philosophy: Selected Papers of Richard Montague.* Yale University Press, New Haven, 1974. R. H. Thomason (ed.).

[Moortgat, 1995] Michael Moortgat. Multimodal linguistic inference. *Journal of Logic, Language and Information*, 5:349–385, 1995. Also in *Bulletin of the IGPL*, 3(2,3):371–401, 1995.

[Moortgat, 1997] Michael Moortgat. Categorial Type Logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 93–177. Elsevier Science B.V. and The MIT Press, Amsterdam and Cambridge, Massachusetts, 1997.

[Morrill, 1990] Glyn Morrill. Grammar and Logical Types. In Martin Stockhof and Leen Torenvliet, editors, *Proceedings of the Seventh Amsterdam Colloquium*, pages 429–450, 1990. Also in G. Barry and G. Morrill, editors, *Studies in Categorial Grammar*, Edinburgh Working Papers in Cognitive Science, Volume 5, pages 127–148: 1990. Revised version published as Grammar and Logic, *Theoria*, LXII, 3:260–293, 1996.

[Morrill, 1994] Glyn V. Morrill. *Type Logical Grammar: Categorial Logic of Signs.* Kluwer Academic Press, Dordrecht, 1994.

[Pollard and Sag, 1987] Carl Pollard and Ivan A. Sag. *Information-based Syntax and Semantics.* Number 13 in CSLI Lecture Notes. CSLI, Stanford, CA, 1987.

[Pollard and Sag, 1994] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar.* The University of Chicago Press, Chicago, 1994.

[Prawitz, 1965] D. Prawitz. *Natural Deduction.* Almqvist & Wiksell, Stockholm, 1965.

[Saussure, 1915] F. de Saussure. *cours de linguistique générale.* English translation published in 1959 by McGraw Hill, New York, 1915.

[Steedman, 1987] Mark Steedman. Combinatory Grammars and Parasitic Gaps. *Natural Language and Linguistic Theory*, 5:403–439, 1987.

[Steedman, 2000] Mark Steedman. *The Syntactic Process.* Bradford Books. MIT Press, Cambridge, Massachusetts, 2000.

[Tarski, 1935] A. Tarski. Der Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, 1:261–405, 1935. English translation in John Corcoran, editor, 1956, Logic, Semantics, Metamathematics, Alfred Tarski, trans. by J.H. Woodger, Oxford University Press, 1956, second edition Hackett Publishing Company, 1983, 152–278.

[van Benthem, 1991] J. van Benthem. *Language in action: Categories Lambdas, and Dynamic Logic.* Number 130 in Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1991. Revised student edition printed in 1995 by MIT Press.