

Polymorphism in English Logical Grammar

Elias Ponvert
The University of Texas at Austin
efp@uts.cc.utexas.edu

Abstract

The quantifiers and connectives of English present difficulties to standard (Church-style) type theory in linguistics in that they can apply systematically to objects of different form. This paper sketches solutions to this problem with some extensions to the simply typed lambda calculus, including Damas-Milner types and System F or the polymorphic lambda calculus. The goal is to maintain the “proofs-as-readings” spirit of the logical grammar tradition while allowing flexible types for the logical terms.

1 Background and Outline

Type theory in linguistics grew out of the categorial investigations of Bar-Hillel [1], Lambek [15, 16, 17] and Curry [5]. Later, Montague’s PTQ semantics [18] utilized a simple Church-style type system to accompany the simple applicative categorial grammar used to define the fragment of English under investigation. This unification and the rediscovery of Lambek’s work led van Benthem [26, 27] and many others to develop the tradition of type-logical grammar. In this tradition (substructural) fragments of propositional intuitionistic logic are used to characterize both natural language (NL) syntax and as the type-theory accompanying lambda terms used to describe NL denotations. The easily defined (injective) functors from the associative syntactic calculus (often Lambek’s **L**) to the type theory (often Lambek’s associative syntactic calculus with permutations, **LP**, or the full positive propositional intuitionistic calculus **J**[\rightarrow^+]) became the generalized format of Montague’s rule-to-rule notion of compositionality. Also, implicitly or explicitly, type-logical grammar was inspired by the Curry-Howard-de Bruijn isomorphism between typed lambda terms and formulae of intuitionistic logic—the tradition characterized by the slogan “formulae as types, proofs as terms”. Applied in the setting of NL semantics, and the above theories of **LP** and **J**[\rightarrow^+], the isomorphism is relaxed to a *correspondence* [19, p.116].

Within the Montagovian tradition, Partee and Rooth [20] addressed the problem that the boolean connectives “and” and “or” in English induce the same basic intuition or reading irrespective of the types of arguments, e.g.

- (1) Serge and Jane sing \approx Serge sings and Jane sings
- (2) Serge sings and smokes \approx Serge sings and Serge smokes

- (3) One or two girls are in the studio \approx
 One girl is in the studio or two girls are in the studio

The monomorphic theory adopted by Montague and the type-logical tradition that followed did not have the expressive power to characterize polymorphic terms like connectives. Partee and Rooth sketched a solution in the following way: let CT be the set of conjoinable types, that is, the set that satisfies

$$(4) \quad \text{CT} ::= \mathbf{t} \mid \mathbf{T} \rightarrow \text{CT}$$

where \mathbf{t} is the type of propositions and $t \rightarrow u$ is the type of functional terms from t to u . Thus,

$$(5) \quad \phi \mathbf{and} \psi = \begin{cases} \phi \wedge \psi & \text{if } \phi \text{ and } \psi \text{ are of type } \mathbf{t} \\ \lambda x^A (\phi \cdot x) \mathbf{and} (\psi \cdot x) & \text{if } \phi \text{ and } \psi \text{ are of type } A \rightarrow B \end{cases}$$

$$(6) \quad \phi \mathbf{or} \psi = \begin{cases} \phi \vee \psi & \text{if } \phi \text{ and } \psi \text{ are of type } \mathbf{t} \\ \lambda x^A (\phi \cdot x) \mathbf{or} (\psi \cdot x) & \text{if } \phi \text{ and } \psi \text{ are of type } A \rightarrow B \end{cases}$$

Along similar lines, more recently several authors [7, 8, 27] have noted that quantifiers like “every”, “one”, “two” etc. exhibit similarly polymorphic tendencies. Consider¹:

- (7) A present pleases every child. [27, 8]
 (8) Every time Serge smokes, women swoon.
 (9) Serge only sings for fun.
 (10) Only Serge sings for fun.
 (11) Serge sings only for fun.

The aim of this paper is to provide interpretations for these and other terms, that are both concise (ideally postulating a single interpretation for terms that exhibit a natural class of truth conditions) and expressive (flexible enough to apply to the variety of arguments that they seem to take).

In what follows, I sketch a monomorphic theory based on Lambek’s categorial **LP**, using an operational semantics to characterize the relationship between terms and types. Then I extend it with sum types, Damas-Milner style polymorphic types and System F polymorphic types, proposing the use of the increasingly powerful systems for certain NL expressions, focusing on the quantifiers “every” and “only” and the basic connectives “and” and “or”.

2 A Type-theoretic Sketch

For a very long time linguists and philosophers have recognized that linguistic denotations can be characterized as mathematical functions or operations; for example Russell explicitly treated the logical constants as truth functions. As such, it is common to utilize the lambda calculus, a theory based around the λ operator for reifying functions anonymously, to express the form of linguistic

¹Thanks are due to the anonymous reviewer who suggested investigating “only” in this context

denotations. As only certain combinations of linguistic terms are *grammatical*, so correspondingly only certain combinations of denotational terms are sensible; thus *type theory*, which characterizes such combinatorial constraints in terms of type-membership from Russell onwards, is commonly utilized in keeping with the Curry-Howard-de Bruijn correspondence described above.

2.1 Syntax

A theory well suited for this is the associative syntactic calculus with permutations **LP** of Lambek [17]. The correspondence between lambda terms and types in **LP** is given in the following schematization: Types T are

$$(12) \quad \mathsf{T} ::= \mathbf{e} \mid \mathbf{t} \mid \mathsf{T} \rightarrow \mathsf{T} \mid \mathsf{T} \circ \mathsf{T}$$

where \mathbf{e} is the type of entities of the model and \mathbf{t} the type of propositions. Terms trm are

$$(13) \quad \mathsf{trm} ::= \mathbf{var} \mid \mathbf{const} \mid \lambda \mathbf{var}^{\mathsf{T}} \mathsf{trm} \mid \mathsf{trm} \cdot \mathsf{trm}$$

for a class \mathbf{var} of term variables and \mathbf{const} of constants. Types and terms are related by the type-tagging on variables in λ expressions, (thus this is a loosely-typed or Church-typed system), but more generally by type judgments. A typing context Γ is a string of variable-type pairs:

$$(14) \quad \Gamma ::= (\mathbf{var}_{\mathsf{T}})^*$$

You can treat contexts as a symbol-type-lookup table, searching from right to left, and as such a partial function $\mathbf{var} \rightarrow \mathsf{T}$. Γ, Δ means Γ and Δ are concatenated. Concatenation is associative, so parentheses do not matter. $\Gamma(\Delta)$ means Δ is a substring of Γ and in particular,

$$\frac{\Gamma(\Delta)}{\Gamma(\Delta')}$$

means Δ' is substituted for Δ in Γ .

2.2 Judgments

LP typing judgments are given in the following natural deduction format:

$$\begin{aligned} & (\mathbf{Ax}) \quad x_A \vdash x : A \\ & (\circ\mathbf{E}) \quad \frac{\Gamma \vdash a : A \circ B \quad \Delta(x_A, y_B) \vdash c : C}{\Delta(\Gamma) \vdash c[x \cdot y/a] : C} \\ & (\rightarrow \mathbf{I}) \quad \frac{\Gamma(x_A) \vdash f : B}{\Gamma \vdash \lambda x^A f : A \rightarrow B} \\ & (\rightarrow \mathbf{E}) \quad \frac{\Gamma \vdash a : A \quad \Delta \vdash f : A \rightarrow B}{\Gamma(\Delta) \vdash (f \cdot a) : B} \\ & (\circ\mathbf{I}) \quad \frac{\Gamma \vdash a : A \quad \Delta \vdash b : B}{\Gamma(\Delta) \vdash a \cdot b : A \circ B} \end{aligned}$$

Constants are assumed to have types already, that is, for $\mathbf{cn} \in \mathbf{const}$

$$(\mathbf{Given}) \vdash \mathbf{cn} : A$$

for some A . Taken together, call this system **LP**_{ND}.

2.3 Gentzen Presentation and Decidability

There is also the Gentzen presentation \mathbf{LP}_G [15, 17, 19], where, together with (Ax) and for what it's worth (Given),

$$\begin{aligned}
(\circ\text{L}) & \frac{\Gamma(x_A(y_B)) \vdash c : C}{\Gamma(z_{A \circ B}) \vdash c[z/x \cdot y] : C} \\
(\circ\text{R}) & \frac{\Gamma \vdash a : A \quad \Delta \vdash b : B}{\Gamma(\Delta) \vdash a \cdot b : A \circ B} \\
(\rightarrow \text{L}) & \frac{\Gamma \vdash a : A \quad \Delta(x_B) \vdash c : C}{\Delta(y_{A \rightarrow B}(\Gamma)) \vdash c[(y \cdot a)/z] : C} \\
(\rightarrow \text{R}) & \frac{\Gamma(x_A) \vdash f : B}{\Gamma \vdash \lambda x^A f : A \rightarrow B}
\end{aligned}$$

Plus the familiar

$$(\text{Cut}) \frac{\Gamma \vdash a : A \quad \Delta(x_A) \vdash c : C}{\Gamma(\Delta) \vdash c[x/a] : C}$$

Lambek [15] and Došen [6] show the equivalence of the two systems, the eliminability of (Cut), and consequently the decidability of both.

Thm 1 ($\mathbf{LP}_{\text{nd}} \approx \mathbf{LP}_g$.) *If there is a proof of $\Gamma \vdash a : A$ in \mathbf{LP}_{ND} then there is a proof of $\Gamma \vdash a : A$ in \mathbf{LP}_G . (Lambek [15])*

Thm 2 (Cut-elimination.) *If there is a proof of $a \vdash A$ in \mathbf{LP}_G with (Cut), there is also a proof without (Cut). (Lambek [15], Došen [6])*

2.4 Reduction and Safety

Term reduction rules follow Pierce [21]'s presentation. *Values* in the system, a subset of terms, are given by

$$(15) \text{ val} ::= \text{const} \mid \lambda \text{var}^\top \text{trm}$$

and reduction rules are, for $a, b, c \in \text{trm}$ and $v \in \text{val}$,

$$\begin{aligned}
(\text{app1}) & \frac{a \rightsquigarrow b}{(a \cdot c) \rightsquigarrow (b \cdot c)} \\
(\text{app2}) & \frac{a \rightsquigarrow b}{(v \cdot a) \rightsquigarrow (v \cdot b)} \\
(\text{beta}) & (\lambda x^A f \cdot a) \rightsquigarrow f[x/a]
\end{aligned}$$

The following two theorems relate term reductions to type judgments (c.f. Pierce [21]):

Thm 3 (Progress) *When $\vdash a : A$ then either $a \in \text{val}$ or $\exists b, a \rightsquigarrow b$.*

Thm 4 (Preservation) *If $\Gamma \vdash a : A$ and $a \rightsquigarrow b$ then $\Gamma \vdash b : A$.*

Taken together, the theorems show that \mathbf{LP} admits the slogan “Safety = Progress + Preservation”.

2.5 Logical Constants

Following Turner [25] we introduce the following constant term to the system,

$$(16) \quad \vdash \Rightarrow : \mathbf{t} \rightarrow \mathbf{t} \rightarrow \mathbf{t}$$

and term schema

$$(17) \quad \vdash \Pi_A : A \rightarrow \mathbf{t} \rightarrow \mathbf{t}$$

The language \mathbf{val} is interpreted by a model $\mathfrak{M} = \langle D, P, T, I, \sqsubseteq, \Pi_A \rangle$. $\llbracket \cdot \rrbracket^{\mathfrak{M}}$ maps types \mathbf{T} into D s.t.

$$(18) \quad \begin{aligned} \llbracket \mathbf{e} \rrbracket^{\mathfrak{M}} &\subseteq D \\ \llbracket \mathbf{t} \rrbracket^{\mathfrak{M}} &= P \\ \llbracket A \rightarrow B \rrbracket^{\mathfrak{M}} &= (\llbracket B \rrbracket^{\mathfrak{M}})^{\llbracket A \rrbracket^{\mathfrak{M}}} \end{aligned}$$

For a model $\mathfrak{M} = \langle D, P, T, I, \sqsubseteq, \Pi_A \rangle$ the following holds,

1. D is a set, the universe,
2. $P \subseteq D$ the set of propositions,
3. $T \subseteq P$ the true propositions,
4. $I : \mathbf{const} \rightarrow D$,
5. \sqsubseteq a function on $P \times P \rightarrow P$ s.t. $x \sqsubseteq y \in T$ if $x \in T$ only if $y \in T$
6. $\Pi_A : P^{\llbracket A \rrbracket} \rightarrow P$ s.t. for $f : \llbracket A \rrbracket \rightarrow P$ $\llbracket \Pi_A(f) \rrbracket \in T$ if for all $a \in \llbracket A \rrbracket$, $f(a) \in T$.

Equipped with a partial function $g : \mathbf{var} \rightarrow D$, $\llbracket \cdot \rrbracket^{\mathfrak{M},g} : \mathbf{val} \cup \mathbf{var} \rightarrow D$ s.t.

$$(19) \quad \begin{aligned} \llbracket \lambda x^A f \rrbracket^{\mathfrak{M},g} &= d \mapsto \llbracket f \rrbracket_{g[x/d]}^{\mathfrak{M}} \text{ for } d \in \llbracket A \rrbracket^{\mathfrak{M}} \\ \llbracket x \rrbracket^{\mathfrak{M},g} &= g(x) \text{ for } x \in \mathbf{var} \\ \llbracket a \rrbracket^{\mathfrak{M},g} &= I(a) \text{ for } a \in \mathbf{const}, \text{ in particular} \\ \llbracket a \Rightarrow b \rrbracket^{\mathfrak{M},g} &= \llbracket a \rrbracket^{\mathfrak{M},g} \sqsubseteq \llbracket b \rrbracket_g^{\mathfrak{M}} \\ \llbracket \Pi_A \rrbracket^{\mathfrak{M},g} &= \Pi_A \end{aligned}$$

The rest of Turner's Intuitionistic Higher Order Logic (IHOL) is given by the following definitions

$$\begin{aligned} (\forall x^T) &\equiv \lambda \phi^{T \rightarrow \mathbf{t}} \Pi_T(\lambda x^T \phi) \\ \wedge &\equiv \lambda \phi^{\mathbf{t}} \lambda \psi^{\mathbf{t}} ((\forall \chi^{\mathbf{t}}) \cdot (\phi \rightarrow (\psi \rightarrow \chi)) \rightarrow \chi) \\ \vee &\equiv \lambda \phi^{\mathbf{t}} \lambda \psi^{\mathbf{t}} ((\forall \chi^{\mathbf{t}}) \cdot (\phi \rightarrow \chi) \rightarrow ((\psi \rightarrow \chi) \rightarrow \chi)) \\ (\exists x^T) &\equiv \lambda \phi^{T \rightarrow \mathbf{t}} \lambda \chi^{\mathbf{t}} ((\forall y^T) \cdot p)(\chi[y/x] \rightarrow \chi) \rightarrow \chi \\ \perp &\equiv ((\forall \phi^{\mathbf{t}}) \cdot \phi) \\ \sim &\equiv \lambda \phi^{\mathbf{t}} \phi \rightarrow \perp \\ =_T &\equiv \lambda a^T \lambda b^T ((\forall f^{T \rightarrow \mathbf{t}}) \cdot (f \cdot a) \rightarrow (f \cdot b)) \end{aligned}$$

It follows that the proof theory of IHOL (made into classical HOL with the inclusion of the law of excluded middle) applies here too.

3 Ad hoc Polymorphism: First Pass

Constructive conjunction and disjunction are added to the system as follows:

Extend \top

$$(20) \quad \top ::= \dots \mid \top \times \top \mid \top + \top$$

with new term schemata, for $i \in \{0, 1\}$,

$$(21) \quad \text{trm} ::= \dots \mid \langle \text{trm}, \text{trm} \rangle \mid \pi_i \text{trm} \mid \iota_i \text{trm} \mid (\text{trm} ? \text{trm} : \text{trm})$$

Typing judgments are extended, for \mathbf{LP}_{ND} ,

$$\begin{aligned} (\times\text{I}) & \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \langle a, b \rangle : A \times B} \\ (\times\text{E}) & \frac{\Gamma \vdash a : A_0 \times A_1}{\Gamma \vdash \pi_i a : A_i} \\ (+\text{I}) & \frac{\Gamma \vdash a : A_i}{\iota_i a \vdash A_0 + A_1 :} \\ (+\text{E}) & \frac{\Gamma \vdash c : A + B \quad \Gamma \vdash a : A \rightarrow C \quad \Gamma \vdash b : B \rightarrow C}{\Gamma \vdash (c ? a : b) : C} \end{aligned}$$

Alternatively, for \mathbf{LP}_{G} ,

$$\begin{aligned} (\times\text{L}) & \frac{\Gamma(x_{A_i}) \vdash b : B}{\Gamma(y_{A_0 + A_1}) \vdash b[\pi_i y/x] : B} \\ (\times\text{R}_i) & \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \langle a, b \rangle : A \times B} \\ (+\text{L}) & \frac{\Gamma(x_A) \vdash a : C \quad \Gamma(y_B) \vdash b : C}{\Gamma(z_{A+B}) \vdash (z ? \lambda x^A a : \lambda y^B b) :} \\ (+\text{R}_i) & \frac{\Gamma \vdash a : A_i}{\Gamma \vdash \iota_i a : A_0 + A_1} \end{aligned}$$

The continued equivalence of \mathbf{LP}_{ND} and \mathbf{LP}_{G} is relatively easy to show. For example, here is $(\times\text{R})$'s derivation in \mathbf{LP}_{ND}

$$(22) \quad \frac{\frac{\Gamma(x_{A_i}) \vdash b : B}{\Gamma \vdash \lambda x^{A_i} b : A \rightarrow B} \quad \frac{y_{A_0 \times A_1} \vdash y : A_0 \times A_1}{y_{A_0 \times A_1} \vdash \pi_i y A_i :}}{\frac{\Gamma, y_{A_0 \times A_1} \vdash (\lambda x^{A_i} b \cdot \pi_i y) : B}{\Gamma, y_{A_0 \times A_1} \vdash b[\pi_i y/x] : B}}$$

Also, new evaluation rules:

$$\begin{aligned}
& (\text{proj1}_i) \pi_i \langle a_0, a_1 \rangle \rightsquigarrow a_i \\
& (\text{proj2}_i) \frac{a \rightsquigarrow a'}{\pi_i a \rightsquigarrow \pi_i a'} \\
& (\text{pair1}) \frac{a \rightsquigarrow a'}{\langle a, b \rangle \rightsquigarrow \langle a', b \rangle} \\
& (\text{pair2}) \frac{a \rightsquigarrow a'}{\langle v, a \rangle \rightsquigarrow \langle v, a' \rangle} \\
& (\text{inj-case}_i) (\iota_i v ? a_0 : a_1) \rightsquigarrow (a_i \cdot v) \\
& (\text{case}) \frac{a \rightsquigarrow a'}{(a ? b : c) \rightsquigarrow (a' ? b : c)} \\
& (\text{inj}_i) \frac{a \rightsquigarrow a'}{\iota_i a \rightsquigarrow \iota_i a'}
\end{aligned}$$

The properties sketched above, Decidability and Safety still hold for **LP** with $\{\times, +\}$ [22, 21]. Also, it is standard to define arbitrary lists and variants directly rather than constructing them out of pairs and sums.

3.1 Conjunction

Ad hoc polymorphism can be applied to a first pass at the definition of generalized conjunction **and** and **or**. Let

$$\begin{aligned}
(23) \quad & \mathbf{and}_t \quad =_{\text{def}} \quad \lambda \phi^{t \times t} \pi_0 \phi \wedge \pi_1 \phi \\
& \mathbf{and}_e \quad =_{\text{def}} \quad \lambda x^{e \times e} (X \cdot \pi_0 x) \wedge (X \cdot \pi_1 x) \\
& \mathbf{or}_t \quad =_{\text{def}} \quad \lambda \phi^{t \times t} \pi_0 \phi \vee \pi_1 \phi \\
& \mathbf{or}_e \quad =_{\text{def}} \quad \lambda x^{e \times e} (X \cdot \pi_0 x) \vee (X \cdot \pi_1 x)
\end{aligned}$$

Polymorphic terms **and** and **or** are given by

$$\begin{aligned}
(24) \quad & \mathbf{and} \quad =_{\text{def}} \quad \lambda x^{e \times e + t \times t} (x ? \mathbf{and}_e : \mathbf{and}_t) \\
& \mathbf{or} \quad =_{\text{def}} \quad \lambda x^{e \times e + t \times t} (x ? \mathbf{or}_e : \mathbf{or}_t)
\end{aligned}$$

where \times binds more closely than $+$. So, for example, “Serge and Brigitte” as in “Serge and Brigitte sing” is derived as follows. Given $\vdash \mathbf{serge} : e$ and $\vdash \mathbf{brigitte} : e$,

$$\begin{aligned}
(25) \quad & \frac{x_{e \rightarrow t} \vdash \mathbf{serge} : e \quad x_{e \rightarrow t} \vdash \mathbf{brigitte} : e}{x_{e \rightarrow t} \vdash \langle \mathbf{serge}, \mathbf{brigitte} \rangle : e \times e} \\
& \frac{x_{e \rightarrow t} \vdash \iota_1 \langle \mathbf{serge}, \mathbf{brigitte} \rangle : t \times t + e \times e}{x_{e \rightarrow t} \vdash (\mathbf{and} \cdot \iota_1 \langle \mathbf{serge}, \mathbf{brigitte} \rangle) : t} \\
& \frac{x_{e \rightarrow t} \vdash (x \cdot \mathbf{serge}) \wedge (x \cdot \mathbf{brigitte}) : t}{\vdash \lambda x^{e \rightarrow t} (x \cdot \mathbf{serge}) \wedge (x \cdot \mathbf{brigitte}) :}
\end{aligned}$$

\mathbf{and}_t and \mathbf{or}_t are legitimate combinators of **LP**: they would not be if \cdot were substituted for \times above.

However, \mathbf{and}_e is not even a combinator, as it contains a free variable.

3.2 Quantification

Lets examine English quantifiers “every” and “only”. Let \mathbf{i} be a new primitive type (hence, $\mathbb{T} ::= \dots \mid \mathbf{i}$) of time-indices and have it that verbs generally quantify over time-indices as sketched in [3], here oversimplified. That said, for the purposes of this presentation, assume that unless specified, verbal terms are applied to temporal constant **now**—i.e. in the present tense:

$$(26) \quad ((\mathbf{loves} \cdot \mathbf{serge}) \cdot \mathbf{brigitte}) =_{\text{def}} (((\mathbf{loves} \cdot \mathbf{now}) \cdot \mathbf{serge}) \cdot \mathbf{brigitte})$$

For now, set $vp =_{\text{def}} \mathbf{e} \rightarrow \mathbf{i} \rightarrow \mathbf{t}$. Consider ²

$$(27) \quad \llbracket \text{Every man sleeps} \rrbracket = ((\mathbf{every} \cdot \mathbf{man}) \cdot \mathbf{sleeps})$$

$$(28) \quad \llbracket \text{Every time Serge sleeps Serge dreams} \rrbracket = \\ ((\mathbf{every} \cdot (\lambda t^{\mathbf{i}} \mathbf{sleeps} \cdot \mathbf{serge})) \cdot (\lambda t^{\mathbf{i}} \mathbf{dreams} \cdot \mathbf{serge}))$$

$$(29) \quad \llbracket \text{Only Serge loves Brigitte} \rrbracket = ((\mathbf{only} \cdot \mathbf{serge}) \cdot ((\mathbf{loves} \cdot \mathbf{brigitte})))$$

$$(30) \quad \llbracket \text{Serge only loves Brigitte} \rrbracket = (((\mathbf{only} \cdot \mathbf{loves}) \cdot \mathbf{serge}) \cdot \mathbf{brigitte})$$

$$(31) \quad \llbracket \text{Serge sings only for fun} \rrbracket = ((\mathbf{only} \cdot \mathbf{forfun}) \cdot ((\mathbf{sings} \cdot \mathbf{serge})))$$

Ascribing a denotation for **every** is standard and throughout the literature [2]. It can be glossed as

$$(32) \quad \mathbf{every} =_{\text{def}} \lambda x^{X-\mathbf{t}} \lambda y^{X-\mathbf{t}} (\forall z^T)(x \cdot z) \Rightarrow (y \cdot z)$$

only is trickier. “Only Serge loves Brigitte” suggests that Serge loves Brigitte, and no one else does, hence

$$(33) \quad \mathbf{only} =_{\text{def}} \lambda x^{\mathbf{e}} \lambda Z^{\mathbf{e}-\mathbf{t}} (Z \cdot x) \wedge (\forall Y^{\mathbf{e}-\mathbf{t}})(Y \cdot x) \Rightarrow Y =_{\mathbf{e}-\mathbf{t}} Z$$

but, “Serge only likes Brigitte” (in the sense that he doesn’t love her or hate her)

$$(34) \quad \mathbf{only} =_{\text{def}} \lambda x^{\mathbf{e}-\mathbf{e}-\mathbf{t}} \lambda z^{\mathbf{e}} \lambda y^{\mathbf{e}} ((x \cdot z) \cdot y) \wedge (\forall w^{\mathbf{e}-\mathbf{e}-\mathbf{t}})((w \cdot z) \cdot y) \Rightarrow w =_{\mathbf{e}-\mathbf{e}-\mathbf{t}} x$$

The problem is not only in choosing a type for the full term, but for the sub-terms as well. Here, categorial fusion and its associativity and permutability, helps

$$(35) \quad \mathbf{only} =_{\text{def}} \lambda x^A \lambda y^B (x \cdot y) \wedge (\forall z^B)(x \cdot z) \Rightarrow y =_B z$$

However, rather than terms for “every” and “only” we have term schemata.

4 From Schemata to Terms

The goal of this section is an account of how to reify the schemata sketched above (32, 35) into terms. This will introduce parametric polymorphism into the lexicon.

²Special thanks to the anonymous commenter on this paper who suggested looking into “only” here.

4.1 Damas-Milner Polymorphism

Add to the system new terms

$$(36) \text{ trm} ::= \dots \mid \text{let } x \leftarrow \text{trm} \text{ in trm}$$

the new typing rule

$$(\text{Let}) \frac{\Gamma \vdash a[x/b] : A \quad \Gamma \vdash b : B}{\Gamma \vdash \text{let } x \leftarrow b \text{ in } a : A}$$

and evaluation rules

$$\begin{aligned} (\text{let1}) & \frac{v \in \mathbf{val}}{\text{let } x \leftarrow v \text{ in } a \rightsquigarrow a[x/v]} \\ (\text{let2}) & \frac{a \rightsquigarrow b}{\text{let } x \leftarrow a \text{ in } c \rightsquigarrow \text{let } x \leftarrow b \text{ in } c} \end{aligned}$$

New *type-place-holders* $?X$ are added to the system and allowed to serve as type-tags on lambda terms. This is feasible in the system because an effective type-reconstruction algorithm exists for the system, choosing concrete types (i.e. members of \mathbf{T}) when a typing exists for a term. In fact, given Γ a given term has a *principal* or *most general solution* to the question of how to type its constituent terms. In effect, this allows terms that do not have type-tags labeling them. Pierce [21] covers this topic.

So, now we have

$$(37) \mathbf{every} =_{\text{def}} \lambda x \lambda y (\forall z)(x \cdot z) \Rightarrow (y \cdot z)$$

$$(38) \mathbf{only} =_{\text{def}} \lambda x \lambda y (x \cdot y) \wedge (\forall z)(z \cdot x) \Rightarrow y = z$$

So, for example, “Every time Serge sleeps Serge dreams” is given by

$$(39) \text{let } x \leftarrow \lambda t^i ((\mathbf{sleeps} \cdot t) \cdot \mathbf{serge}) \text{ in let } y \leftarrow \lambda t^i ((\mathbf{dreams} \cdot t) \cdot \mathbf{serge}) \text{ in } (\forall z^i)(x \cdot z) \Rightarrow (y \cdot z)$$

Note that replacing x and then y with $\lambda t^i ((\mathbf{sleeps} \cdot t) \cdot \mathbf{serge})$ and then $\lambda t^i ((\mathbf{dreams} \cdot t) \cdot \mathbf{serge})$ reduces to

$$(40) ((\mathbf{every} \cdot \lambda t^i ((\mathbf{sleeps} \cdot t) \cdot \mathbf{serge})) \cdot \lambda t^i ((\mathbf{dreams} \cdot t) \cdot \mathbf{serge}))$$

Since it is fairly straightforward, though tedious, to provide a typing for **every** s.t. $\vdash (40) : \mathbf{t}$, it follows that

$$(41) (\text{Let}) \frac{\vdash (40) : \mathbf{t}}{\vdash (39) : \mathbf{t}}$$

4.2 Polymorphic Lambda Calculus

System F or the polymorphic lambda calculus [11, 12, 23, 24] adds to the type system a universal type schema and an abstraction operation on types. Now, in the object language there are two classes of variables, type variables $\mathbf{T}\mathbf{y}\mathbf{V}\mathbf{a}\mathbf{r}$ and term variables $\mathbf{T}\mathbf{m}\mathbf{V}\mathbf{a}\mathbf{r}$. Now, typing constants have type variables occur in them

$$(42) \Gamma ::= (\mathbf{var}_T \mid \mathbf{T}\mathbf{y}\mathbf{V}\mathbf{a}\mathbf{r})^*$$

and there are new terms that range over type symbols

$$(43) \quad \mathsf{T} ::= \mathbf{e} \mid \mathbf{t} \mid \mathbf{i} \mid \mathsf{TyVar} \mid \mathsf{T} \rightarrow \mathsf{T} \mid \forall \mathsf{T}. \mathsf{T}$$

The other term schemata defined above are now unnecessary, as System F can define general terms for pairs, sums, even arithmetic via Church-encodings.

$$(44) \quad \mathsf{TmVar} ::= \mathsf{TmVar} \mid \mathbf{const} \mid \lambda \mathsf{TmVar}^{\mathsf{T}} \mathbf{trm} \mid (\mathbf{trm} \cdot \mathbf{trm}) \mid \Lambda \mathsf{TyVar} \mathbf{trm} \mid (\mathbf{trm} \cdot \mathsf{T})$$

New typing rules for F

$$\begin{aligned} (\forall\mathsf{I}) \quad & \frac{\Gamma, X \vdash f : A}{\Gamma \vdash \Lambda X \ f : \forall X. A} \\ (\forall\mathsf{E}) \quad & \frac{\Gamma \vdash f : \forall X. A}{\Gamma \vdash f_B : A[B/X]} \end{aligned}$$

and reduction rules

$$\begin{aligned} (\mathbf{tyapp}) \quad & \frac{a \rightsquigarrow b}{a_A \rightsquigarrow b_A} \\ (\mathbf{tybeta}) \quad & \Lambda X \ a_A \rightsquigarrow a[X/A] \end{aligned}$$

System F is safe in the sense of “Preservation + Progress” [21], proofs are cut admissible and F has the property of normalization, that all well-typed terms terminate [11, 12].

The semantics of System F are strictly outside those we sketched for **LP**, as quantifiers range over propositional variables, then to comprehend a type $\forall X. A$, it is necessary to comprehend all $A[X/B]$, where B may be much more complicated than A . In this sense F is *impredicative*. There are two known classes of models of System F [24]. In one, types are mapped to partial equivalence relations on a model of the untyped lambda calculus. In the other, the meaning of type is a Scott domain.

Also, in general type reconstruction fails, that is, given a term a of the untyped lambda calculus, whether there is a term a' in System F s.t. $a = a'$ with all the type-tags removed [21, 29].

Finally, in linguistics Martin Emms [8] discovered that in general the associative syntactic calculus **L** equipped with higher-order types is undecidable (in spite of showing that it is cut-admissible).

4.3 Examples of $\forall \rightarrow \forall$ Polymorphism?

The general impredictativity of System F, and the results regarding its lack of type reconstruction and undecidability when constrained to **L** raise the question whether F is too powerful for linguistic tasks. Many linguists think so; van Benthem [27] prefers Damas-Milner style types, which he call “substitutional polymorphism” which are also studied in Emms [9] and elsewhere.

Several authors [4, 14, 10] have pointed out the impredictativity of certain English terms, such as “is fun”, and have generalized the type theory of their respective theories (or, in Chierchia’s case, removed it altogether).

In keeping with the emphasis on quantifiers and connectives in this paper, note the connectives can coordinate (polymorphic) quantifiers themselves:

$$(45) \quad \text{All and only men love Brigitte.}$$

- [8] Martin Emms. Parsing with polymorphism. In *Proceedings of the sixth conference on European chapter of the Association for Computational Linguistics*, pages 120–129, Morristown, NJ, 1993. Association of Computational Linguistics.
- [9] Martin Emms. Models of the polymorphic lambek calculus. In Christian Retoré, editor, *LAFL*, volume 1328 of *Lecture Notes in Computer Science*, pages 168–187. Springer, 1996.
- [10] Chris Fox and Shalom Lappin. An expressive first-order logic with flexible typing for natural language semantics. *Logic Journal of the Interest Group in Pure and Applied Logics*, 12:135–168, 2004.
- [11] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures dans l’arithmétique d’ordre supérieure*. Thèse de doctorat d’état, Université de Paris VII, 1972.
- [12] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. CUP, Cambridge, 1989. Page references cite online version.
- [13] Roman Jakobson, editor. *Studies of Language and Its Mathematical Aspects*, Providence, 1961. AMS.
- [14] Fairouz Kammareddine. Important issues in foundational formalisms. *Bulletin of the IGPL*, 3:291–317, 1995.
- [15] Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170, 1958.
- [16] Joachim Lambek. Contributions to a mathematical analysis of the english verb phrase. *Journal of the Canadian Linguistic Association*, 5:183–189, 1959.
- [17] Joachim Lambek. On the calculus of syntactic types. In Jakobson [13], pages 166–178.
- [18] Richard Montague. The proper treatment of quantification in ordinary english. In J. Hintikka, J. Moravcsik, and P. Suppes, editors, *Approaches to Natural Language*, Dordrecht, 1973. D. Reidel.
- [19] Michael Moortgat. Categorical type logics. In van Bentham and ter Meulen [28].
- [20] Barbara H. Partee and Matts Rooth. Generalized conjunction and type ambiguity. In Rainer Bäuerle, Christoph Schwarze, and Arnim von Stechow, editors, *Meaning, use and the interpretation of language*, pages 361–93. Walter de Gruyter, 1983.
- [21] Benjamin J. Pierce. *Types and Programming Languages*. MIT, Cambridge, Mass., 2002.
- [22] Greg Restall. *An Introduction to Substructural Logics*. Routledge, London and New York, 2000.

- [23] John C. Reynolds. Toward a theory of type structure. In J. E. Fenstad, editor, *Proceedings, Colloque sur la programmation*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425, Berlin, 1974. Springer-Verlag.
- [24] John C. Reynolds. Introduction to part ii. In Gérard Huet, editor, *Logical Foundations of Functional Programming*, The UT Year of Programming Series, pages 77–86, Reading, Massachusetts, 1990. Addison-Wesley.
- [25] Raymond Turner. Types. In van Benthem and ter Meulen [28].
- [26] Johan van Benthem. The semantics of variety in categorial grammar. Report 83–29, Department of Mathematics, Simon Fraser University, Burnaby, 1983.
- [27] Johan van Benthem. *Language in Action: Categories, Lambdas and Dynamic Logic*. MIT, Cambridge, Mass., 1995.
- [28] Johan van Benthem and Alice ter Meulen, editors. *Handbook of Logic and Language*, Amsterdam and Cambridge, Mass., 1997. Elsevier and MIT.
- [29] Joe B. Wells. Typability and type checking in the second-order lambda-calculus are equivalent and undecidable. In *Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 176–185, 1994.