A CALCULUS FOR SEMANTIC COMPOSITION AND SCOPING

Technical Note 467

May 5, 1989

By: Fernando C. N. Pereira

Artificial Intelligence Center
Computer and Information Sciences Division
and
Center for the Study of Language and Information

To appear in the *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, June, 1989.

The preparation of this paper and the research reported herein were supported in part by a grant from the System Development Foundation to the Center for the Study of Language and Information, and in part under contract with the Nippon Telegraph and Telephone Company.

A Calculus for Semantic Composition and Scoping

Fernando C. N. Pereira
Artificial Intelligence Center, SRI International
333 Ravenswood Ave., Menlo Park, CA 94025, USA

Abstract

Certain restrictions on possible scopings of quantified noun phrases in natural language are usually expressed in terms of formal constraints on binding at a level of logical form. Such reliance on the form rather than the content of semantic interpretations goes against the spirit of compositionality. I will show that those scoping restrictions follow from simple and fundamental facts about functional application and abstraction, and can be expressed as constraints on the derivation of possible meanings for sentences rather than constraints of the alleged forms of those meanings.

1 An Obvious Constraint?

Treatments of quantifier scope in Montague grammar (Montague, 1973; Dowty et al., 1981; Cooper, 1983), transformational grammar (Reinhart, 1983; May, 1985; Heim, 1982; Roberts, 1987) and computational linguistics (Hobbs and Shieber, 1987; Moran, 1988) have depended implicitly or explicitly on a constraint on possible logical forms to explain why examples[1] such as

(1) * A woman who saw *every man* disliked *him*

are ungrammatical, and why in examples such as

(2) *Every man* saw a friend of *his*
(3) *Every admirer* of a picture of *himself* is vain

the *every*... noun phrase must have wider scope than the *a*... noun phrase if the pronoun in each example is assumed to be *bound* by its antecedent. What exactly counts as bound anaphora varies between different accounts of the phenomena, but the rough intuition is that semantically a bound pronoun plays the role of a variable bound by the logical form (a quantifier) of its antecedent. Example (1) above is then "explained" by noting that its logical form would be something like

$$\begin{aligned}&\exists w.\textsc{woman}(w)\&\\&\quad(\forall m.\textsc{man}(m) \Rightarrow \textsc{saw}(w,m))\&\\&\quad\textsc{disliked}(w,m)\end{aligned}$$

but this is "ill-formed" because variable m occurs as an argument of DISLIKED outside the scope of its binder $\forall m$.[2] As for Examples (2) and (3), the argument is similar: wide scope

[1]In all the examples that follow, the pronoun and its intended antecedent are italicized. As usual, starred examples are supposed to be ungrammatical.

[2]In fact, this is a perfectly good *open* well-formed formula and therefore the precise formulation of the constraint is more delicate than seems to be realized in the literature.

for the logical form of the *a*... noun phrase would leave an occurrence of the variable that the logical form of *every*... binds outside the scope of this quantifier. For lack of an official name in the literature for this constraint, I will call it here the *free-variable* constraint.

In accounts of scoping possibilities based on quantifier raising or storage (Cooper, 1983; van Eijck, 1985; May, 1985; Hobbs and Shieber, 1987), the free-variable constraint is enforced either by keeping track of the set of free variables $\textsc{free}(q)$ in each raisable (storable) term q and when $x \in \textsc{free}(q)$ blocking the raising of q from any context $Bx.t$ in which x is bound by some binder B, or by checking after all applications of raising (unstoring) that no variable occurs outside the scope of its binder.

The argument above is often taken to be so obvious and uncontroversial that it warrants only a remark in passing, if any (Cooper, 1983; Reinhart, 1983; Partee and Bach, 1984; May, 1985; van Riemsdijk and Williams, 1986; Williams, 1986; Roberts, 1987), even though it depends on nontrivial assumptions on the role of logical form in linguistic theory and semantics.

First of all, and most immediately, there is the requirement for a logical-form level of representation, either in the predicate-logic format exemplified above or in some tree format as is usual in transformational grammar (Heim, 1982; Cooper, 1983; May, 1985; van Riemsdijk and Williams, 1986; Williams, 1986; Roberts, 1987).

Second, and most relevant to Montague grammar and related approaches, the constraint is formulated in terms of restrictions on formal objects (logical forms) which in turn are related to meanings through a denotation relation. However, compositionality as it is commonly understood requires meanings of phrases to be functions of the *meanings* rather than the forms of their constituents. This is a problem even in accounts based on quantifier storage (Cooper, 1983; van Eijck, 1985), which are precisely designed, as van Eijck puts it, to "avoid all unnecessary reference to properties of ... formulas" (van Eijck, 1985, p. 214). In fact, van Eijck proposes an interesting modification of Cooper storage that avoids Cooper's reliance on forbidding vacuous abstraction to block out cases in which a noun phrase is unstored while a noun phrase contained in it is still in store. However, this restriction does not deal with the case I have been discussing.

It is also interesting to observe that a wider class of examples of forbidden scopings would have to be considered if raising out of relative clauses were allowed, for example in

(4) An author who John has read every book by arrived

In this example, if we did not assume the restriction against raising from relative clauses, the *every*... noun phrase could in principle be assigned widest scope, but this would be blocked by the free-variable constraint as shown by the occurrence of b free as an argument of $\textsc{book-by}$ in

$$\begin{aligned}&\forall b.\textsc{book-by}(b,a) \Rightarrow\\ &\quad(\exists a.\textsc{author}(a)\&\\ &\qquad\textsc{has-read}(\textsc{john},b)\&\textsc{arrived}(a))\end{aligned}$$

That is, the alleged constraint against raising from relatives, for which many counterexamples exist (Vanlehn, 1978), blocks some derivations in which otherwise the free-variable constraint would be involved, specifically those associated to syntactic configurations of the form

$$[_{\mathrm{NP}_i} \cdots \mathrm{N} [_{\overline{\mathrm{S}}} \cdots [_{\mathrm{NP}_j} \cdots \mathrm{X}_i \cdots] \cdots] \cdots]$$

where X_i is a pronoun or trace coindexed with NP_i and NP_j is a quantified noun phrase. Since some of the most extensive Montague grammar fragments in the literature (Dowty

$$\frac{A \quad A \rightarrow B}{B} \qquad \frac{\overset{(A)}{B}}{A \rightarrow B}$$

Figure 1: Curry Rules

et al., 1981; Cooper, 1983) do not cover the other major source of the problem, PP complements of noun phrases (replace $\overline{\text{S}}$ by PP in the configuration above), the question is effectively avoided in those treatments.

The main goal of this paper is to argue that the free-variable constraint is actually a consequence of basic semantic properties that hold in a semantic domain allowing functional application and abstraction, and are thus independent of a particular logical-form representation. As a corollary, I will also show that the constraint is better expressed as a restriction on the *derivations* of meanings of sentences from the meanings of their parts rather than a restriction on logical forms. The resulting system is related to the earlier system of conditional interpretation rules developed by Pollack and Pereira (1988), but avoids that system's use of formal conditions on the order of assumption discharge.

2 Curry's Calculus of Functionality

Work in combinatory logic and the λ-calculus is concerned with the elucidation of the basic notion of functionality: how to construct functions, and how to apply functions to their arguments. There is a very large body of results in this area, of which I will need only a very small part.

One of the simplest and most elegant accounts of functionality, originally introduced by Curry and Feys (1968) and further elaborated by other authors (Stenlund, 1972; Lambek, 1980; Howard, 1980) involves the use of a logical calculus to describe the *types* of valid functional objects. In a natural deduction format (Prawitz, 1965), the calculus can be simply given by the two rules in Figure 1. The first rule states that the result of applying a function from objects of type A to objects of type B (a function of type $A \rightarrow B$) to an object of type A is an object of type B. The second rule states that if from an arbitrary object of type A it is possible to construct an object of type B, then one has a function from objects of type A to objects of type B. In this rule and all that follow, the parenthesized formula at the top indicates the discharge of an assumption introduced in the derivation of the formula below it. Precise definitions of assumption and assumption discharge are given below.

The typing rules can be directly connected to the use of the λ-calculus to represent functions by restating the rules as shown in Figure 2. That is, if u has type A and v has type $A \rightarrow B$ then $v(u)$ has type B, and if by assuming that x has type A we can show that u (possibly containing x) has type B, then the function represented by $\lambda x.u$ has type $A \rightarrow B$.

To understand what inferences are possible with rules such as the ones in Figure 2, we need a precise notion of derivation, which is here adapted from the one given by Prawitz (1965). A *derivation* is a tree with each node n labeled by a formula $\phi(n)$ (the *conclusion* of the node) and by a set $\Gamma(n)$ of formulas giving the *assumptions* of $\phi(n)$. In addition, a

$$[\mathsf{app}] : \frac{u : A \qquad v : A \to B}{v(u) : B} \qquad [\mathsf{abs}] : \frac{\begin{array}{c}(x : A)\\ u : B\end{array}}{\lambda x.u : A \to B}$$

Figure 2: Curry Rules for Type Checking

derivation D satisfies the following conditions:

i. For each leaf node $n \in D$, either $\phi(n)$ is an *axiom*, which in our case is a formula giving the type and interpretation of a lexical item, and then $\Gamma(n)$ is empty, or $\phi(n)$ is an assumption, in which case $\Gamma(n) = \{\phi(n)\}$

ii. Each nonleaf node n corresponds either to an application of [app], in which case it has two daughters m and m' with $\phi(m) \equiv u : A$, $\phi(m') \equiv v : A \to B$, $\phi(n) \equiv v(u) : B$ and $\Gamma(n) = \Gamma(m) \cup \Gamma(m')$, or to an application of [abs], in which case n has a single daughter m, and $\phi(m) \equiv u : B$, $\phi(n) \equiv \lambda x.u : A \to B$, and $\Gamma(n) = \Gamma(m) - \{x : A\}$

If n is the root node of a derivation D, we say that D is a derivation of $\phi(n)$ from the assumptions $\Gamma(n)$.

Notice that condition (ii) above allows empty abstraction, that is, the application of rule [abs] to some formula $u : B$ even if $x : A$ is not one of the assumptions of $u : B$. This is necessary for the Curry calculus, which describes all typed λ-terms, including those with vacuous abstraction, such as the polymorphic K combinator $\lambda x.\lambda y.x : A \to (B \to A)$. However, in the present work, every abstraction needs to correspond to an actual functional dependency of the interpretation of a phrase on the interpretation of one of its constituents. Condition (ii) can be easily modified to block vacuous abstraction by requiring that $x : A \in \Gamma(m)$ for the application of the [abs] rule to a derivation node m. [3]

The definition of derivation above can be generalized to arbitrary rules with n premises and one conclusion by defining a rule of inference as a $n+1$-place relation on pairs of formulas and assumption sets. For example, elements of the [app] relation would have the general form $\langle\langle u : A, \Gamma_1\rangle, \langle v : A \to B, \Gamma_2\rangle, \langle v(u) : B, \Gamma_1 \cup \Gamma_2\rangle\rangle$, while elements of the [abs] rule without vacuous abstraction would have the form $\langle\langle u : B, \Gamma\rangle, \langle \lambda x.u : A \to B, \Gamma - \{x : A\}\rangle\rangle$ whenever $x : A \in \Gamma$. This definition should be kept in mind when reading the derived rules of inference presented informally in the rest of the paper.

3 Semantic Combinations and the Curry Calculus

In one approach to the definition of allowable semantic combinations, the possible meanings of a phrase are exactly those whose type can be derived by the rules of a semantic calculus

[3] Without this restriction to the abstraction rule, the types derivable using the rules in Figure 2 are exactly the consequences of the three axioms $A \to A$, $A \to (B \to A)$ and $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$, which are the polymorphic types of the three combinators I, K and S that generate all the closed typed λ-calculus terms. Furthermore, if we interpret $\to$ as implication, these theorems are exactly those of the pure implicational fragment of intuitionistic propositional logic (Curry and Feys, 1968; Stenlund, 1972; Anderson and Belnap, 1975). In contrast, with the restriction we have the weaker system of pure relevant implication $\mathbf{R}_{\to}$ (Prawitz, 1965; Anderson and Belnap, 1975).

from axioms giving the types of the lexical items in the phrase. However, this is far too liberal in that the possible meanings of English phrases do not depend only on the types involved but also on the syntactic structure of the phrases. A possible way out is to encode the relevant syntactic constraints in a more elaborate and restrictive system of types and rules of inference. The prime example of a more constrained system is the Lambek calculus (Lambek, 1958) and its more recent elaborations within categorial grammar and semantics (van Benthem, 1986a; van Benthem, 1986b; Hendriks, 1987; Moortgat, 1988). In particular, Hendriks (1987) proposes a system for quantifier raising, which however is too restrictive in its coverage to account for the phenomena of interest here.

Instead of trying to construct a type system and type rules such that free application of the rules starting from appropriate lexical axioms will generate all and only the possible meanings of a phrase, I will instead take a more conservative route related to Montague grammar and early versions of GPSG (Gazdar, 1982) and use syntactic analyses to control semantic derivations.

First, a set of *derived* rules will be used in addition to the basic rules of application and abstraction. Semantically, the derived rules will add no new inferences, since they will merely codify inferences already allowed by the basic rules of the calculus of functionality. However, they provide the semantic counterparts of certain syntactic rules.

Second, the use of some semantic rules must be *licensed* by a particular syntactic rule and the premises in the antecedent of the semantic rule must correspond in a rule-given way to the meanings of the constituents combined by the syntactic rule. As a simple example using a context-free syntax, the syntactic rule S $\rightarrow$ NP VP might license the function application rule [app] with A the type of the meaning of the NP and $A \rightarrow B$ the type of the meaning of the VP.

Third, the domain of types will be enriched with a few new type constructors, in addition to the function type constructor $\rightarrow$. From a purely semantic point of view, these type constructors add no new types, but allow a convenient encoding of rule applicability constraints motivated by syntactic considerations. This enrichment of the formal universe of types for syntactic purposes is familiar from Montague grammar (Montague, 1973), where it is used to distinguish different syntactic realizations of the same semantic type, and from categorial grammar (Lambek, 1958; Steedman, 1987), where it is used to capture syntactic word-order constraints.

Together, the above refinements allow the syntax of language to restrict what potential semantic combinations are actually realized. Any derivations will be sound with respect to [app] and [abs], but many derivations allowed by these rules will be blocked.

4 Derived Rules

In the rules below, we will use the two basic types $\mathtt{e}$ for individuals and $\mathtt{t}$ for propositions, the function type constructor $\rightarrow$ associating to the right, the formal type constructor $\mathtt{quant}(q)$, where q is a *quantifier*, that is, a value of type $(\mathtt{e} \rightarrow \mathtt{t}) \rightarrow \mathtt{t}$, and the two formal types pron for pronoun assumptions and $\mathtt{trace}$ for traces in relative clauses. For simplicity in examples, I will adopt a "reverse Curried" notation for the meanings of verbs, prepositions and relational nouns. For example, the meaning of the verb *to love* will be $\textsc{love} : \mathtt{e} \rightarrow \mathtt{e} \rightarrow \mathtt{t}$, with x the lover and y the loved one in $\textsc{love}(y)(x)$. The assumptions corresponding to lexical items in a derivation will be appropriately labeled.

$$[\mathsf{trace+}] : \frac{x : \mathtt{trace}}{x : \mathtt{e}} \qquad [\mathsf{trace-}] : \frac{\begin{array}{c}(x : \mathtt{trace})\\ r : \mathtt{t}\end{array}}{\lambda x.r : \mathtt{e} \to \mathtt{t}}$$

Figure 3: Rules for Relative Clauses

$$[\mathsf{pron+}] : \frac{x : \mathtt{pron}}{x : \mathtt{e}} \qquad [\mathsf{pron-}] : \frac{\begin{array}{cc}(x : \mathtt{pron}) & \\ s : A & y : B\end{array}}{(\lambda x.s)(y) : A}$$

Figure 4: Bound Anaphora Rules

4.1 Trace Introduction and Abstraction

The two derived rules in Figure 3 deal with traces and the meaning of relative clauses. Rule [trace+] is licensed by the the occurrence of a trace in the syntax, and rule [trace−] by the construction of a relative clause from a sentence containing a trace. Clearly, if $n : \mathtt{e} \to \mathtt{t}$ can be derived from some assumptions using these rules, then it can be derived using rule [abs] instead.

For an example of use of [trace+] and [trace−], assume that the meaning of relative pronoun *that* is THAT $\overset{\text{def}}{=} \lambda r.\lambda n.\lambda x.n(x)\&r(x) : (\mathtt{e} \to \mathtt{t}) \to (\mathtt{e} \to \mathtt{t}) \to (\mathtt{e} \to \mathtt{t})$. Given appropriate syntactic licensing, Figure 5 shows the derivation of a meaning for *car that John owns.* Each nonleaf node in the derivation is labeled with the rule that was used to derive it, and leaf nodes are labeled according to their origin (lexical entries for words in the phrase or syntactic traces). The assumptions at each node are not given explicitly, but can be easily computed by looking in the subtree rooted at the node for undischarged assumptions.

4.2 Bound Anaphora Introduction and Elimination

Another pair of rules, shown in Figure 4, is responsible for introducing a pronoun and resolving it as bound anaphora. The pronoun resolution rule [pron−] applies only when B is $\mathtt{trace}$ or quant(q) for some quantifier q. Furthermore, the premise $y : B$ does not belong to an immediate constituent of the phrase licensing the rule, but rather to some undischarged assumption of $s : A$, which will remain undischarged.

These rules deal only with the construction of the meaning of phrases containing bound anaphora. In a more detailed grammar, the licensing of both rules would be further restricted by linguistic constraints on coreference — for instance, those usually associated with c-command (Reinhart, 1983), which seem to need access to syntactic information (Williams, 1986). In particular, the rules as given do not by themselves enforce any constraints on the possible antecedents of reflexives.

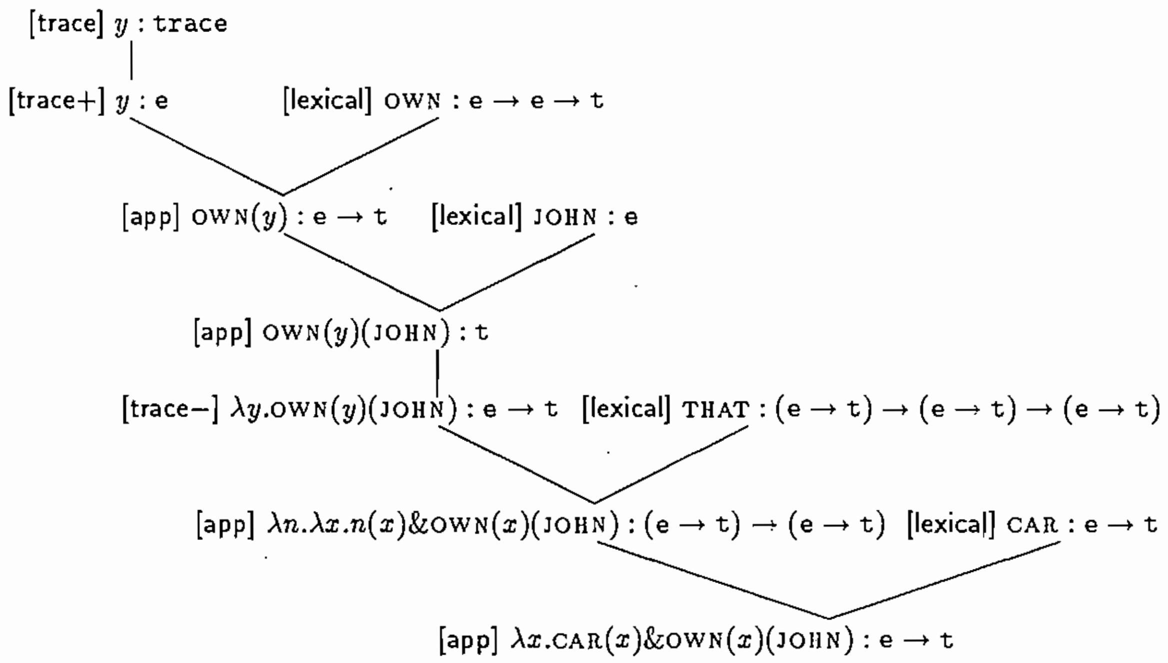

[trace] y : trace
[trace+] y : e
[lexical] OWN : e → e → t
[app] OWN(y) : e → t
[lexical] JOHN : e
[app] OWN(y)(JOHN) : t
[trace−] λy.OWN(y)(JOHN) : e → t
[lexical] THAT : (e → t) → (e → t) → (e → t)
[app] $\lambda n.\lambda x.n(x)$&OWN(x)(JOHN) : (e → t) → (e → t)
[lexical] CAR : e → t
[app] λx.CAR(x)&OWN(x)(JOHN) : e → t

Figure 5: Using Derived Rules

The soundness of the rules can be seen by noting that the schematic derivation

$$\dfrac{\dfrac{\dfrac{(x : \mathtt{pron})}{x : \mathsf{e}}}{\vdots} \atop s : A \qquad y : B}{(\lambda x.s)(y) : A}$$

corresponds to a special case of the schematic derivation

$$\dfrac{y : e \qquad \dfrac{\begin{array}{c}(x : e)\\ \vdots\\ s : A\end{array}}{\lambda x.s : e \rightarrow A}}{(\lambda x.s)(y) : A}$$

The example derivation in Figure 7, which will be explianed in more detail later, shows the application of the anaphora rules in deriving an interpretation for example sentence (2).

4.3 Quantifier Raising

The rules discussed earlier provide some of the auxiliary machinery required to illustrate the free-variable constraint. However, the main burden of enforcing the constraint falls on

$$[\mathsf{quant+}] : \frac{q : (\mathtt{e} \to \mathtt{t}) \to \mathtt{t} \quad x : \mathtt{quant}(q)}{x : \mathtt{e}} \qquad [\mathsf{quant-}] : \frac{\begin{array}{c}(x : \mathtt{quant}(q)) \\ s : \mathtt{t}\end{array}}{q(\lambda x.s) : \mathtt{t}}$$

Figure 6: Quantifier Rules

the rules responsible for quantifier raising, and therefore I will cover in somewhat greater detail the derivation of those rules from the basic rules of functionality.

I will follow here the standard view (Montague, 1973; Barwise and Cooper, 1981) that natural-language determiners have meanings of type $(\mathtt{e} \to \mathtt{t}) \to (\mathtt{e} \to \mathtt{t}) \to \mathtt{t}$. For example, the meaning of *every* might be $\lambda r.\lambda s.\forall x.r(x) \Rightarrow s(x)$, and the meaning of the noun phrase *every man* will be $\lambda s.\forall x.\mathrm{MAN}(x) \Rightarrow s(x)$. To interpret the combination of a quantified noun phrase with the phrase containing it that forms its scope, we apply the meaning of the noun phrase to a property s derived from the meaning of the scope. The purpose of devices such as quantifying-in in Montague grammar, Cooper storage or quantifier raising in transformational grammar is to determine a scope for each noun phrase in a sentence. From a semantic point of view, the combination of a noun phrase with its scope, most directly expressed by Montague's quantifying-in rules,[4] corresponds to the following schematic derivation in the basic calculus (rules [app] and [abs] only):

$$\frac{\dfrac{\begin{array}{c}(x : \mathtt{e}) \\ \vdots \\ s : \mathtt{t}\end{array}}{\lambda x.s : \mathtt{e} \to \mathtt{t}} \qquad q : (\mathtt{e} \to \mathtt{t}) \to \mathtt{t}}{q(\lambda x.s) : \mathtt{t}}$$

where the assumption $x : \mathtt{e}$ is introduced in the derivation at a position corresponding to the occurrence of the noun phrase with meaning q in the sentence. In Montague grammar, this correspondence is enforced by using a notion of syntactic combination that does not respect the syntactic structure of sentences with quantified noun phrases. Cooper storage was in part developed to cure this deficiency, and the derived rules presented below address the same problem.

Now, the free-variable constraint is involved in situations in which the quantifier q itself depends on assumptions that must be discharged. The relevant incomplete schematic derivation (again in terms of [app] and [abs] only) is

$$\frac{\dfrac{\begin{array}{c}\text{(a)}\quad (x : \mathtt{e}) \\ \vdots \\ s : \mathtt{t}\end{array}}{\lambda x.s : \mathtt{e} \to \mathtt{t}} \qquad \begin{array}{c}\text{(b)}\quad y : \mathtt{e} \\ \vdots \\ q : (\mathtt{e} \to \mathtt{t}) \to \mathtt{t} \\ ?\end{array}}{\begin{array}{c}q(\lambda x.s) : \mathtt{t} \\ ?\end{array}} \tag{5}$$

[4]In general, quantifying-in has to apply not only to proposition-type scopes but also to property-type scopes (meanings of common-noun phrases and verb-phrases). Extending the argument that follows to those cases offers no difficulties.

Given that the assumption $y:\mathrm{e}$ has not been discharged in the derivation of $q:(\mathrm{e}\to\mathtt{t})\to\mathtt{t}$, that is, $y:\mathrm{e}$ is an undischarged assumption of $q:(\mathrm{e}\to\mathtt{t})\to\mathtt{t}$, the question is how to complete the whole derivation. If the assumption were discharged before q had been combined with its scope, the result would be the semantic object $\lambda y.q:\mathrm{e}\to(\mathrm{e}\to\mathtt{t})\to\mathtt{t}$, which is of the wrong type to be combined by [app] with the scope $\lambda x.s$. Therefore, there is no choice but to discharge (b) *after* q is combined with its scope. Put in another way, q cannot be raised outside the scope of abstraction for the variable y occurring free in q, which is exactly what is going on in Example (4) ('An author who John has read every book by arrived'). A correct schematic derivation is then

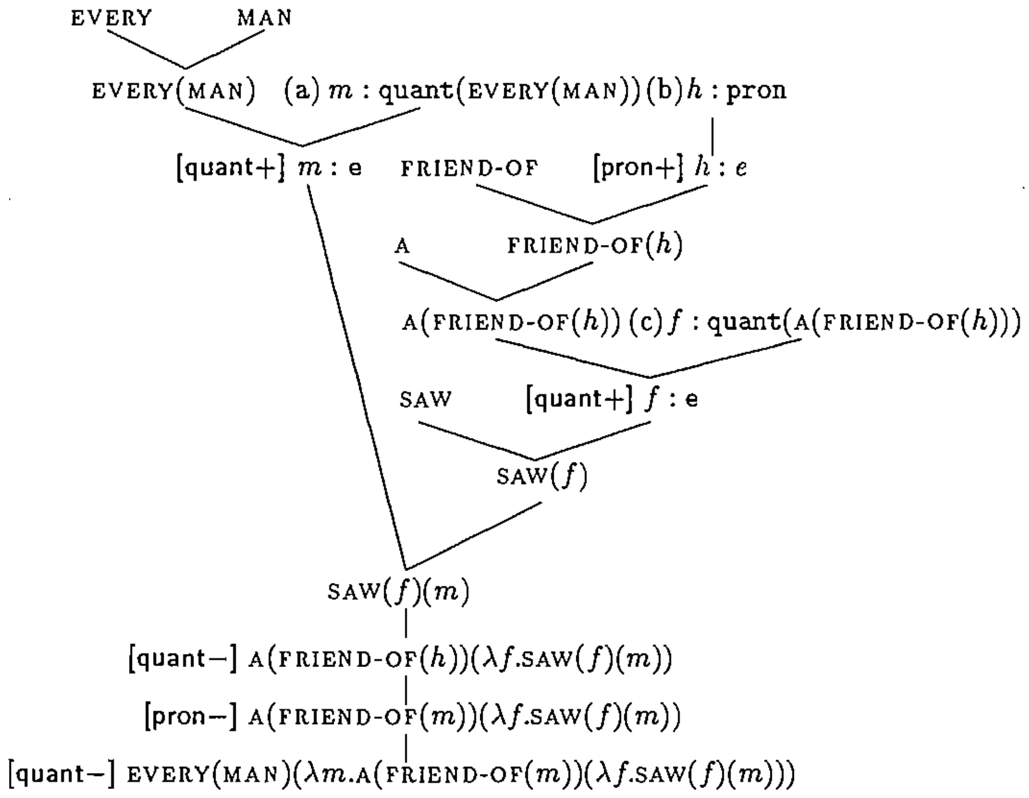
$$\dfrac{\begin{array}{c}\dfrac{\dfrac{\begin{array}{c}\text{(a)}\quad (x:\mathrm{e})\\ \vdots\\ s:\mathtt{t}\end{array}}{\lambda x.s:\mathrm{e}\to\mathtt{t}}\qquad\begin{array}{c}\text{(b)}\quad (y:\mathrm{e})\\ \vdots\\ q:(\mathrm{e}\to\mathtt{t})\to\mathtt{t}\end{array}}{q(\lambda x.s):\mathtt{t}}\\ \vdots\\ u:A\end{array}}{\lambda y.u:\mathrm{e}\to A}$$

In the schematic derivations above, nothing ensures the association between the syntactic position of the quantified noun phrase and the introduction of assumption (a). To do this, we need the the derived rules in Figure 6. Rule [quant+] is licensed by a quantified noun phrase. Rule [quant−] is not keyed to any particular syntactic construction, but instead may be applied whenever its premises are satisfied. It is clear that any use of [quant+] and [quant−] in a derivation

$$\dfrac{\begin{array}{c}\dfrac{\begin{array}{c}\vdots\\ q:(\mathrm{e}\to\mathtt{t})\to\mathtt{t}\end{array}\qquad (x:\mathrm{quant}(q))}{x:\mathrm{e}}\\ \vdots\\ s:\mathtt{t}\end{array}}{q(\lambda x.s):\mathtt{t}}$$

can be justified by translating it into an instance of the schematic derivation (5).

The situation relevant to the free-variable constraint arises when q in [quant+] depends on assumptions. It is straightforward to see that the constraint on a sound derivation according to the basic rules discussed earlier in this section turns now into the constraint that an assumption of the form $x:\mathrm{quant}(q)$ must be discharged before any of the assumptions on which q depends. Thus, the free-variable constraint is reduced to a constraint on derivations imposed by the basic theory of functionality, dispensing with a logical-form representation of the constraint. Figure 7 shows a derivation for the only possible scoping of sentence (2) when *every man* is selected as the antecedent of *his*. To allow for the selected coreference, the pronoun assumption must be discharged before the quantifier assumption (a) for *every man*. Furthermore, the constraint on dependent assumptions requires that the quantifier assumption (c) for *a friend of his* be discharged before the pronoun assumption (b) on which it depends. It then follows that assumption (c) will be discharged before assumption (a), forcing wide scope for *every man*.


EVERY
MAN
EVERY(MAN)
(a) m : quant(EVERY(MAN))
(b) h : pron
[quant+] m : e
FRIEND-OF
[pron+] h : e
A
FRIEND-OF(h)
A(FRIEND-OF(h))
(c) f : quant(A(FRIEND-OF(h)))
SAW
[quant+] f : e
SAW(f)
SAW(f)(m)
[quant−] A(FRIEND-OF(h))(λf.SAW(f)(m))
[pron−] A(FRIEND-OF(m))(λf.SAW(f)(m))
[quant−] EVERY(MAN)(λm.A(FRIEND-OF(m))(λf.SAW(f)(m)))

Most interpretation types and the inference rule label on uses of [app] have been omitted for simplicity.

Figure 7: Derivation Involving Anaphora and Quantification

5 Discussion

The approach to semantic interpretation outlined above avoids the need for manipulations of logical forms in deriving the possible meanings of quantified sentences. It also avoids the need for such devices as distinguished variables (Gazdar, 1982; Cooper, 1983) to deal with trace abstraction. Instead, specialized versions of the basic rule of functional abstraction are used. To my knowledge, the only other approaches to these problems that do not depend on formal operations on logical forms are those based on specialized logics of type change, usually restrictions of the Curry or Lambek systems (van Benthem, 1986a; Hendriks, 1987; Moortgat, 1988). In those accounts, a phrase P with meaning p of type T is considered to have also alternative meaning p' of type T', with the corresponding combination possibilities, if $p' : T'$ follows from $p : T$ in the chosen logic. The central problem in this approach is to design a calculus that will cover all the actual semantic alternatives (for instance, all the possible quantifier scopings) without introducing spurious interpretations. For quantifier raising, the system of Hendriks (1987) seems the most promising so far, but it is at present too restrictive to support raising from noun-phrase complements.

An important question I have finessed here is that of the compositionality of the proposed semantic calculus. It is clear that the application of semantic rules is governed only by the existence of appropriate syntactic licensing and by the availability of premises of the appropriate types. In other words, no rule is sensitive to the *form* of any of the meanings appearing in its premises. However, there may be some donbt as to the status of the basic abstraction rule and those derived from it. After all, the use of λ-abstraction in the consequent of those rules seems to imply the constraint that the abstracted object should formally be a variable. However, this is only superficially the case. I have used the formal operation of λ-abstraction to represent functional abstraction in this paper, but functional abstraction itself is independent of its formal representation in the λ-calculus. This can be shown either by using other notations for functions and abstraction, such as that of de Bruijn's (Barendregt, 1984; Huet, 1986), or by expressing the semantic derivation rules in λ-Prolog (Miller and Nadathur, 1986) following existing presentations of natural deduction systems (Felty and Miller, 1988).

Acknowledgments

This research was supported by a contract with the Nippon Telephone and Telegraph Corp. and by a gift from the Systems Development Foundation as part of a coordinated research effort with the Center for the Study of Language and Information, Stanford University. I thank Mary Dalrymple and Stuart Shieber for their helpful discussions regarding this work.

Bibliography

Alan Ross Anderson and Nuel D. Belnap, Jr. 1975. *Entailment: the Logic of Relevance and Necessity, Volume I.* Princeton University Press, Princeton, New Jersey.

Hank P. Barendregt. 1984. *The Lambda Calculus: its Syntax and Semantics.* North-Holland, Amsterdam, Holland.

Jon Barwise and Robin Cooper. 1981. Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4:159–219.

Robin Cooper. 1983. *Quantification and Syntactic Theory.* D. Reidel, Dordrecht, Netherlands.

Haskell B. Curry and Robert Feys. 1968. *Combinatory Logic, Volume I.* Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, Holland. Second printing.

David R. Dowty, Robert E. Wall, and Stanley Peters. 1981. *Introduction to Montague Semantics*, Volume 11 of *Synthese Language Library.* D. Reidel, Dordrecht, Holland.

Amy Felty and Dale Miller. 1988. Specifying theorem provers in a higher-order logic programming language. Technical Report MS-CIS-88-12, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania.

Gerald Gazdar. 1982. Phrase structure grammar. In P. Jacobson and G.K. Pullum, editors, *The Nature of Syntactic Representation*, pages 131–186. D. Reidel, Dordrecht, Holland.

Irene R. Heim. 1982. *The Semantics of Definite and Indefinite Noun Phrases.* Ph.D. thesis, Department of Linguistics, University of Massachusetts, Amherst, Massachusetts (September).

Herman Hendriks. 1987. Type change in semantics: the scope of quantification and coordination. In Ewan Klein and Johan van Benthem, editors, *Categories, Polymorphism and Unification*, pages 95–120. Centre for Cognitive Science, University of Edinburgh, Edinburgh, Scotland.

Jerry R. Hobbs and Stuart M. Shieber. 1987. An algorithm for generating quantifier scopings. *Computational Linguistics*, 13:47–63.

W.A. Howard. 1980. The formulae-as-types notion of construction. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, London, England.

Gérard Huet. 1986. Formal structures for computation and deduction. First edition of the lecture notes of a course given in the Computer Science Department of Carnegie-Mellon University during the Spring of 1986 (May).

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

Joachim Lambek. 1980. From λ-calculus to cartesian closed categories. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 375–402. Academic Press, London, England.

Robert May. 1985. *Logical Form: its Structure and Derivation*, Volume 12 of *Linguistic Inquiry Monographs.* MIT Press, Cambridge, Massachusetts.

Dale A. Miller and Gopalan Nadathur. 1986. Higher-order logic programming. In Ehud Shapiro, editor, *Third International Conference on Logic Programming*, Berlin, Germany. Springer-Verlag.

Richard Montague. 1973. The proper treatment of quantification in ordinary English. In Richmond H. Thomason, editor, *Formal Philosphy.* Yale University Press.

Michael Moortgat. 1988. *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus.* Ph.D. thesis, University of Amsterdam, Amsterdam, Holland (October).

Douglas B. Moran. 1988. Quantifier scoping in the SRI Core Language Engine. In *26th Annual Meeting of the Association for Computational Linguistics*, pages 33–47, Morristown, New Jersey. Association for Computational Linguistics.

Barbara Partee and Emmon Bach. 1984. Quantification, pronouns and VP anaphora. In J.A.G. Groenendijk, T.M.V. Janssen, and M.B.J. Stokhof, editors, *Truth, Interpretation and Information*, pages 99–130. Foris, Dordrecht, Holland.

Martha E. Pollack and Fernando C.N. Pereira. 1988. An integrated framework for semantic and pragmatic interpretation. In *26th Annual Meeting of the Association for Computational Linguistics*, pages 75–86, Morristown, New Jersey. Association for Computational Linguistics.

Dag Prawitz. 1965. *Natural Deduction: A Proof-Theoretical Study.* Almqvist and Wiksell, Uppsala, Sweden.

Tanya Reinhart. 1983. *Anaphora and Semantic Interpretation.* Croom Helm, London, England, corrected and revised printing, 1987 edition.

Craige Roberts. 1987. *Modal Subordination, Anaphora and Distributivity.* Ph.D. thesis, Department of Linguistics, University of Massachusetts, Amherst, Massachusetts (February).

Mark Steedman. 1987. Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory*, 5(3):403–439.

Sören Stenlund. 1972. *Combinators, λ-Terms and Proof Theory.* D. Reidel, Dordrecht, Holland.

Johan van Benthem. 1986a. Categorial grammar and lambda calculus. In D. Skordev, editor, *Mathematical Logic and its Application*, pages 39–60. Plenum Press, New York, New York.

Johan van Benthem. 1986b. *Essays in Logical Semantics*, Volume 29 of *Studies in Linguistics and Philosophy.* D. Reidel, Dordrecht, Holland.

Jan van Eijck. 1985. *Aspects of Quantification in Natural Language.* Ph.D. thesis, University of Groningen, Groningen, Holland (February).

Henk van Riemsdijk and Edwin Williams. 1986. *Introduction to the Theory of Grammar*, Volume 12 of *Current Studies in Linguistics.* MIT Press, Cambridge, Massachusetts.

Kurt A. Vanlehn. 1978. Determining the scope of English quantifiers. Master's thesis, M.I.T. (June).

Edwin Williams. 1986. A reassignment of the functions of LF. *Linguistic Inquiry*, 17(2):265–299.