CATEGORIAL SEMANTICS AND SCOPING

Fernando C. N. Pereira

AT & T Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974

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; Alshawi et al. 1989) 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

$$\exists w.\text{woman}(w) \wedge (\forall m.\text{man}(m) \Rightarrow \text{saw}(w, m)) \wedge \text{disliked}(w, m)$$

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 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 free (q) in each raisable (storable) term q and when $x \in$ 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 given 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 being addressed here.

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 a free as an argument of book-by in

$$\forall b.\text{book-by}(b, a) \Rightarrow (\exists a.\text{author}(a) \wedge \text{has-read}(\text{john}, b) \wedge \text{arrived}(a))$$

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

$$[_{\text{NP}\,i} \cdots \text{N}[_{\overline{\text{S}}} \cdots [_{\text{NP}j} \cdots \text{X}_i \cdots] \cdots] \cdots]$$

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 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.

Conversely, it could be argued that the free-variable constraint is responsible for forcing a quantifier to raise outside a relative clause in examples such as

5. The slush fund that *every minister* needs is kept by *his* private secretary.

Here, the coindexing of *every minister* and *his* forces the former to be scoped outside the main clause, and thus outside the relative clause in which it occurs.

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 following two rules:

$$\frac{\alpha \quad \alpha \rightarrow \beta}{\beta} \qquad \frac{\begin{array}{c}(\alpha)\\ \vdots \\ \beta\end{array}}{\alpha \rightarrow \beta}$$

The first rule states that the result of applying a function from objects of type α to objects of type β (a function of type $\alpha \rightarrow \beta$) to an object of type α is an object of type β. The second rule states that if from an arbitrary object of type α it is possible to construct an object of type β, then one has a function from objects of type α to objects of type β. 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 them as follows:

$$[\text{app}] : \frac{u : \alpha \quad v : \alpha \rightarrow \beta}{v(u) : \beta} \qquad [\text{abs}] : \frac{\begin{array}{c}(x : \alpha)\\ u : \beta\end{array}}{\lambda x.u : \alpha \rightarrow \beta}$$

A formula $u: \alpha$ will be called a *type assignment*, assigning the type α to term u. Thus, the two rules above state that if u has type α and v has type $\alpha \rightarrow \beta$ then $v(u)$ has type β, and if by assuming that x has type α, we can show that u (possibly containing x) has type β, then the function represented by $\lambda x.u$ has type $\alpha \rightarrow \beta$. In the rest of this paper, the term *functionality rules* will refer to these rules.

Notation As usual in categorial analyses of natural language, types are built from the basic types e for individuals and t for propositions. In function types, the constructor $\rightarrow$ associates to the right. Following this type system (no tuple types) expressions will be written in a "curried" notation. For example, a binary relation r over individuals will have type $\text{e} \rightarrow \text{e} \rightarrow \text{t}$ and its application to arguments x and y will be written $r(x)(y)$. Since the usual semantic analysis has a transitive verb combining first with its object and then with its subject, the meaning of sentence such as "John loves Mary" will be represented by loves(m)(j).

For a first example of how the functionality rules may be used in semantic interpretation, I will consider the derivation in Figure 1 of a simplified meaning for the topicalized

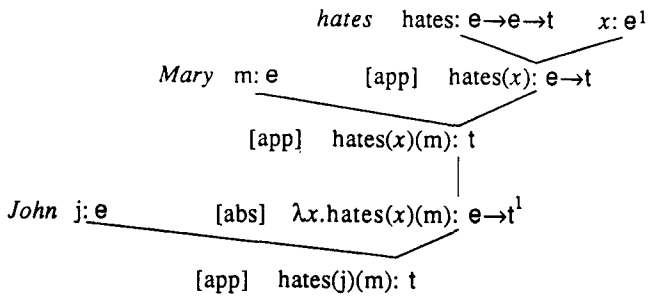

Figure 1 Using the Functionality Rules.

sentence "John, Mary hates".[3] The derivation is represented by a tree in which leaf nodes are labeled by assumptions and nonleaf nodes are labeled by the results of rule applications. Several aspects of the overall approach appear in this example. Each type assignment in the derivation is associated to one phrase whose meaning it represents, and is derived by applying the functionality rules to type assignments associated with the constituents of the phrase. The meanings of words in the sentence enter as assumptions in the derivation, which are left undischarged. In the figure, type assignments are labeled either by the lexical item that led to their introduction as assumptions or by the name of the rule applied to derive the type assignment. Superscripts are used to link the result of an application of rule [abs] to the assumptions discharged by that application. The derivation follows closely the syntax of the sentence. Specifically, the assumptions j : e, m : e, and hates : e → e → t correspond to the words "John," "Mary," and "hates" in the sentence, and the assumption x : e corresponds to the topicalization trace.

To make precise what inferences are possible in the calculus of functionality, we need a precise notion of derivation. It turns out that seemingly small differences in this have important consequences as to what type assignments can be derived. The following definition is adapted from that of Prawitz (1965). A *derivation* is a tree with each node n labeled by a type assignment $\phi(n)$ (the *conclusion* of the node) and by a set $\Gamma(n)$ of type assignments giving the *assumptions of* $\phi(n)$. In addition, a derivation D satisfies the following conditions:

1. Each leaf node n has its labeling type assignment as single assumption, that is $\Gamma(n) = \{\phi(n)\}$.
2. Each nonleaf node n corresponds either to an application of [app], in which case it has two daughters m and m' with $\phi(m) \equiv u : \alpha$, $\phi(m') \equiv v : \alpha \rightarrow \beta$, $\phi(n) \equiv v(u) : \beta$ and $\Gamma(n) = \Gamma(m) \cup \Gamma(m')$, or to an application of [abs], in which case n has a single daughter m, and $\phi(m) \equiv u : \beta$, $\phi(n) \equiv \lambda x.u : \alpha \rightarrow \beta$, and $\Gamma(n) = \Gamma(m) - \{x : \alpha\}$.
3. No node assumption set contains two assumptions $x : \alpha$ and $x : \beta$ for the same variable x.

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

Condition 2 allows empty abstraction, that is, the application of rule [abs] to some type assignment $u : \beta$ even if $x : \alpha$ is not one of the assumptions of $u : \beta$. This is necessary for the Curry calculus, which describes all typed λ-terms, including those with vacuous abstraction, such as the polymorphic **K** combinator $\lambda x.\lambda y.x : \alpha \rightarrow (\beta \rightarrow \alpha)$. 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 2 can be easily modified to block vacuous abstraction by requiring that $x : \alpha \in \Gamma(m)$ for the application of the [abs] rule to a derivation node m.

Condition 3 is a technical restriction to ensure that a given variable is not abstracted more than once. That this restriction does not affect the class of well-typed functions follows immediately from the observation that any λ-term can be α-converted to an equivalent one in which all bound variables are distinct.

The definition of derivation 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 type assignments and assumption sets. For example, elements of the [app] relation would have the general form $\langle\langle u : \alpha, \Gamma_1\rangle, \langle v : \alpha \rightarrow \beta, \Gamma_2\rangle, \langle v(u) : \beta, \Gamma_1 \cup \Gamma_2\rangle\rangle$, while elements of the [abs] rule without vacuous abstraction would have the form $\langle\langle u : \beta, \Gamma\rangle, \langle \lambda x.u : \alpha \rightarrow \beta, \Gamma - \{x : \alpha\}\rangle\rangle$ whenever $x : \alpha \in \Gamma$. This definition should be kept in mind when reading the derived rules of inference presented informally in the rest of the paper.

The natural-deduction format we have been using is intuitively quite appealing but does not make it easy to discriminate among different possible treatments of assumptions, and their effects on what types are derivable. For this, it is better to use a Gentzen-style sequent format in which the assumptions for a type assignment are carried explicitly. In a fairly general setting, a sequent $\Gamma \vdash A$ is a pair of a sequence Γ of assumptions and a type assignment A, with the intuitive meaning that A follows from the assumptions Γ. The rules of application and abstraction take then the form

$$\frac{x : \alpha, \Gamma \vdash u : \beta}{\Gamma \vdash \lambda x.u : \alpha \rightarrow \beta} \qquad \frac{\Gamma \vdash u : \alpha \quad \Delta \vdash v : \alpha \rightarrow \beta}{\Gamma, \Delta \vdash v(u) : \beta}$$

It is easy to see that, except for the use of sequences rather than sets, these two rules correspond directly to the operations of type assignments and assumptions sets described under condition 2 above. Furthermore, condition 1 corresponds in this system to making each sequent $A \vdash A$, where A is some type assignment, an *axiom*. Finally, the fact that we are interested in assumptions sets rather than assumption sequences is encoded by the following three *structural rules* (Girard et al. 1989):

$$\text{[exchange]} : \frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}$$

$$[\text{contraction}] : \frac{A, A, \Gamma \vdash B}{A, \Gamma \vdash B}$$

$$[\text{weakening}] : \frac{\Gamma \vdash B}{A, \Gamma \vdash B}$$

The exchange rule allows us to ignore assumption order, so the collection of assumptions forms a bag rather than a set. Adding contraction permits us to ignore the number of occurrences of an assumption, that is, the assumption collection is treated as a set. Without contraction, no variable can have more than one occurrence. Finally, weakening allows irrelevant elements to be added to the assumptions without changing what follows from them. Without it, vacuous abstraction is not possible, since weakening provides the only means of introducing a variable in an assumption without having it also in the conclusion (as axioms require).

Choices of structural rules and other constraints on allowed sequents lead to a *categorial hierarchy* whose members are systems with varying semantic powers of semantic combination (Moortgat 1988; van Bentham 1989). If we ignore the associated λ-expressions and consider only the types, the types derivable using the full set of rules are exactly the consequences of the three axioms $\alpha \rightarrow \alpha$, $\alpha \rightarrow (\beta \rightarrow \alpha)$, and $(\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow ((\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma))$, 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 $\rightarrow$ as implication, these theorems are exactly those of the pure implicational fragment of intuitionistic propositional logic (Curry and Feys 1968; Anderson and Belnap Jr. 1975). In contrast, without weakening (vacuous abstraction) we have the weaker system of pure relevant implication $\mathbf{R}_{\rightarrow}$ (Anderson and Belnap 1975).

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 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, 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 e the type of the meaning of the NP and e $\rightarrow$ t 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 the 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

For the derived rules, we enrich the type system with a formal type constructor quant (q), where q is a *quantifier,* that is, a value of type (e $\rightarrow$ t) $\rightarrow$ t, and two type constants (nullary constructors) pron for pronoun assumptions and trace for traces in relative clauses. It is important to observe that the types resulting from the application of these constructors are not to be taken as being subtypes of e, and thus the rules involving them should not be seen as type subsumption rules. Instead, the new formal type constructors serve to constrain possible derivations in a similar way to the directed function type constructors of Lambek calculus (Lambek 1958) or the extraction constructor used by Moortgat to handle long-distance dependencies (Moortgat 1988).

Because of their particular nature, the formal type constructors are handled in a special way diverging somewhat

from the usual structure of natural-deduction proof systems. Specifically, each formal type constructor is introduced as an assumption by appropriate syntactic licensing. However, the types resulting from the formal constructors do not combine with any other types, so the only way of having the associated variables participate in a derivation is to apply immediately *licensing* rules that replace the formal type by an appropriate semantic type (e in all cases considered here). Paired with these rules we have *abstraction* rules that discharge formal type assumptions by abstraction. Nevertheless, we will see that from a derivation involving these rules where all the formal type assumptions have been discharged, it is straightforward to construct a derivation in the functionality calculus starting from the same lexical assumptions and yielding the same result.

4.1 TRACE LICENSING AND ABSTRACTION

The following two rules deal with traces and the meaning of relative clauses:

$$[\text{trace-lic}]: \frac{x : \text{trace}}{x : \text{e}} \qquad [\text{trace-abs}]: \frac{\begin{array}{c}(x : \text{trace}) \\ \vdots \\ r : \text{t}\end{array}}{\lambda x.r : \text{e} \rightarrow \text{t}}$$

Rule [trace-lic] is licensed by the occurrence of a trace in the syntax, and rule [trace-abs] by the construction of a relative clause from a sentence containing a trace.[4]

Since no rule can derive an expression of type trace, expressions of that type can only appear as assumptions. Furthermore, no rule except [trace-lic] accepts premises of that type. Therefore, in a completed derivation any occurrence of an expression of type trace must be a premise of [trace-lic] and be later discharged by a use of [trace-abs]. That is, use of the trace rules will always lead to derivations matching the following schema

$$\frac{\begin{array}{c}\dfrac{x : \text{trace}^1}{x : \text{e}} \\ \vdots \\ r : \text{t}\end{array}}{\lambda x.r : \text{e} \rightarrow \text{t}^1}$$

But this can be mapped directly into the schematic derivation

$$\frac{\begin{array}{c}x : \text{e}^1 \\ \vdots \\ r : \text{t}\end{array}}{\lambda x.r : \text{e} \rightarrow \text{t}^1}$$

Consequently, any type assignment derived with the help of the trace rules could already have been derived with [app] and [abs] alone, as was claimed in the previous section.

Figure 2 shows the application of the trace rules to the derivation of an interpretation for the $\overline{N}$ "car that John owns," with the assumption that the relative pronoun "that" has type that: $(\text{e} \rightarrow \text{t}) \rightarrow (\text{e} \rightarrow \text{t}) \rightarrow (\text{e} \rightarrow \text{t})$, that is, a function that combines two properties into a property. With the further assumption that "that" corresponds in this case to property conjunction

$$\text{that} \stackrel{\text{def}}{=} \lambda r.\lambda n.\lambda x.n(x) \wedge r(x),$$

the result of the derivation can be reduced to the more familiar form

$$\lambda x.\text{car}(x) \wedge \text{own}(x)(\text{j})$$

that is, the property of being a car that John owns.

4.2 BOUND ANAPHORA LICENSING AND ABSTRACTION

The analysis of bound anaphora brings up a wide range of issues in syntax, semantics, and pragmatics, most of which I will ignore in this paper. I will assume that possible coreferences are determined elsewhere and that the role of the bound anaphora rules here is simply to derive the appropriate semantic interpretation for phrases involving pronouns. Two approaches are possible here. Working with the functionality rules alone, a noun phrase will be associated to an assumption of the form u : e. The interpretation of a pronoun coindexed with that noun phrase will be another occurrence of the assumption. When the antecedent is a trace or a quantified noun phrase (interpretation of quantified noun phrases is discussed in the next section), the assumption will eventually be discharged. The definition of derivation in Section 2 ensures that all occurrences of u will be bound by the same application of [abs].

The second approach relies on a pair of derived rules, pronoun licensing and abstraction. These rules of course do not add new semantic consequences, but facilitate the representation of the syntactic licensing of bound ana-

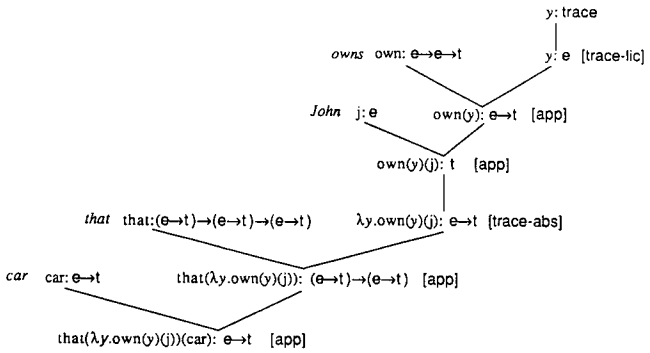

Figure 2 Using the Trace Rules.

phora. The two rules are as follows:

$$[\text{pron-lic}]: \frac{\begin{array}{c}(x:\text{pron})\\ \vdots\end{array}}{x:\text{e}} \qquad [\text{pron-abs}]: \frac{s:\alpha \quad u:\beta}{(\lambda x.s)(u):\alpha}$$

The pronoun resolution rule [pron-abs] applies only when $u:\beta$ is an undischarged assumption of $s:\alpha$ such that either β is trace or quant (q) for some quantifier q, or the assumption is licensed by some proper noun.

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.

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

$$\cfrac{\cfrac{\begin{array}{c}x:\text{pron}^1\\ \vdots\end{array}}{x:\text{e}} \quad \begin{array}{c}s:\alpha\\ \vdots\end{array}}{(\lambda x.s)(u):\alpha^1}$$

corresponds simply to a schematic derivation involving multiple uses of the assumption $u:\beta$

$$\cfrac{\begin{array}{c}u:\beta \cdots u:\beta\\ \vdots\end{array}}{s[x/u]:\alpha}$$

where $s[x/u]$ denotes the result of substituting u for every free occurrence of x in s.

Figure 3 shows a simple derivation involving the pronoun rules. The last derivation node in the figure is the application of [pron-abs] to the assumption to be discharged x : pron and the antecedent assumption j : e, with result $(\lambda x.\text{bored}\ (x)(\text{j}))(\text{j}) \equiv \text{bored}\ (\text{j})(\text{j})$. A more interesting case, involving interactions between pronoun and quantifier assumptions, occurs in the derivation of Figure 5 for sentence (2).

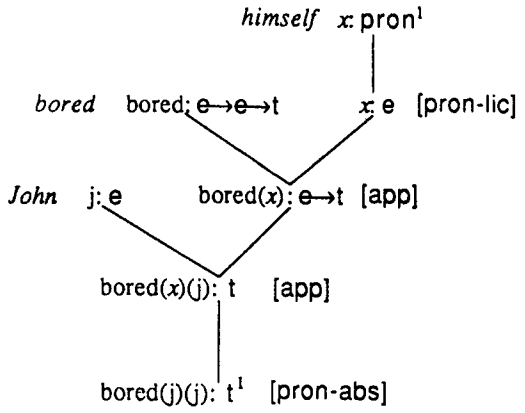

Figure 3 Using the Pronoun Rules.

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 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 the meanings of natural language determiners are *generalized quantifiers,* with type $(\text{e} \to \text{t}) \to (\text{e} \to \text{t}) \to \text{t}$. For example, the meaning of *every* would be $\lambda r.\lambda s.\forall x.r(x) \Rightarrow s(x)$, and the meaning of the noun phrase *every man* $\lambda s.\forall x.man(x) \Rightarrow s(x)$. 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,[5] corresponds to the following schematic derivation in the functionality calculus

$$6.\quad \cfrac{\cfrac{\begin{array}{c}x:\text{e}^1\\ \vdots\end{array}}{s:\text{t}} \quad q:(\text{e}\to\text{t})\to\text{t}}{q(\lambda x.s):\text{t}}$$

where the assumption x : e is introduced in the derivation at a position corresponding to the occurrence in the sentence of the noun phrase with meaning q. 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. The following derived rules achieve the same effect:

$$[\text{quant-lic}]: \frac{q : (e \to t) \to t \quad x : \text{quant}(q)}{x : e}$$

$$(x : \text{quant}(q))$$

$$\vdots$$

$$[\text{quant-abs}]: \frac{s : t}{q(\lambda x.s) : t}$$

Rule [quant-lic] is licensed by a quantified noun phrase. Rule [quant-abs] is not keyed to any particular syntactic construction, but instead may be applied whenever its premises are satisfied. It is easy to see that any use of [quant-lic] and [quant-abs] in a derivation

$$\vdots$$

$$\frac{q : (e \to t) \to t \quad x : \text{quant}(q)^1}{x : e}$$

$$\vdots$$

$$\frac{s : t}{q(\lambda x.s) : t^1}$$

can be justified by translating it into an instance of the schematic derivation (6). Furthermore, quantifier assumptions can only arise and be discharged in this way.

Figure 4 shows the use of the quantification rules in a derivation for the preferred reading of the sentence "Every guest brought a dish." The other reading could be derived in a similar manner.

Now, the free-variable constraint plays a role in situations in which the quantifier itself depends on assumptions that must be discharged, and forbids derivations of the form

$$y : \alpha^1$$

$$\vdots$$

$$\frac{q : (e \to t) \to t \quad x : \text{quant}(q)^2}{x : e}$$

$$\vdots$$

$$u : \beta^1$$

$$\vdots$$

$$\frac{s : t}{q(\lambda x.s) : t^2}$$

that is, derivations in which an assumption is discharged after assumptions it depends on. But such a derivation maps to the following in the functionality calculus:

$$x : e^2$$

$$\vdots$$

$$\frac{v : \gamma}{\lambda y.v : \alpha \to \gamma^1} \quad \text{(a)}$$

$$\vdots$$

$$u : \beta \qquad \text{(b) } y : \alpha^1$$

$$\vdots \qquad\qquad \vdots$$

$$\frac{s : t}{\lambda x.s : e \to t^2} \quad q : (e \to t) \to t$$

$$q(\lambda x.s) : t$$

The problem with this derivation is that assumption $y : \alpha$ is discharged at (a), but it is not an assumption for the premise $v : \gamma$ of (a). Even if we allow vacuous abstraction so that an assumption $y : \alpha$ can be discharged at that point, that discharge will not include the assumption $y : \alpha$ (b) for the quantifier q, which will thus be left undischarged. In the Gentzen-style rules given in Section 2, step (a) can only arise after an application of weakening to introduce $y : \beta$, but on the other hand, assumption (b) can only be eliminated independently by abstraction, since the alternative of combining it by contraction with the other occurrence of $y : \alpha$ is only available after the assumptions for the left and right premises of the application of q to its scope. But that is too late, because the other occurrence of $y : \beta$ has been discharged by then. Therefore, there is no choice but to

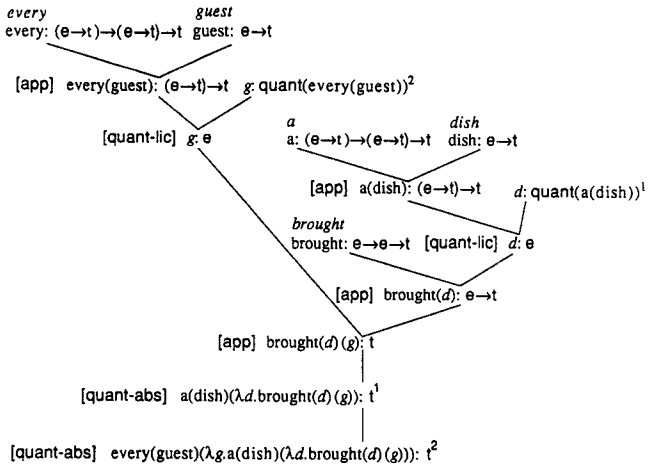

Figure 4 Using the Quantifier Rules.

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

$$\dfrac{\dfrac{\dfrac{x : \mathrm{e}^1 \qquad (\mathrm{b})\ y\ \alpha^2}{\vdots \qquad\quad \vdots}}{\dfrac{s : \mathrm{t}}{\lambda x.s : \mathrm{e} \rightarrow \mathrm{t}^1} \quad q : (\mathrm{e} \rightarrow \mathrm{t}) \rightarrow \mathrm{t}}}{q(\lambda x.s) : \mathrm{t}}$$

$$\vdots$$

$$\dfrac{u : \beta}{\lambda y.u : \alpha \rightarrow \beta^2}$$

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 5 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*.

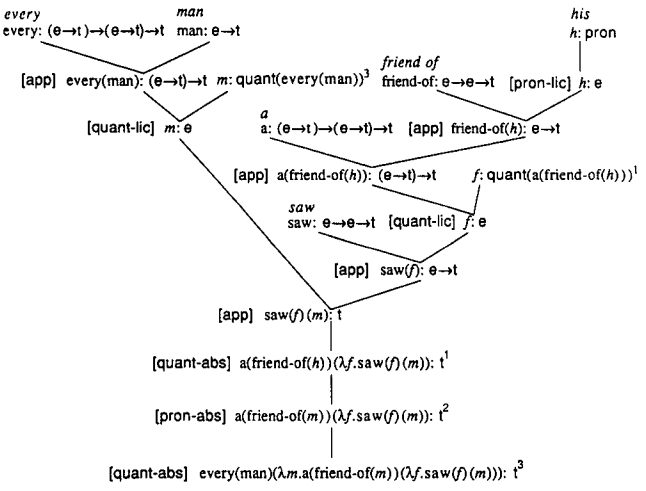

Figure 5 Derivation Involving Anaphora and Quantification.

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 seems the most promising so far, but it is at present too restrictive to support raising from noun-phrase complements.

Formal types and derived rules are a rather special-purpose mechanism for constraining semantic derivations. A more general view, based on a notion of the possible relationships between syntactic and semantic algebra is desirable here, maybe following from the notion of projection proposed for lexical-functional grammar (Halvorsen and Kaplan 1988).

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 doubt 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. All that is required is a calculus of functional objects in which those objects satisfy their usual algebraic properties, such as what is provided by λ-Prolog (Miller and Nadathur 1986). For instance, in a λ-Prolog realization of the proposed system, abstractions arising from the [abs] rule or its derived rule surrogates could be calculated through a combination of universal quantification in the metalanguage (the language in which the proof system is described) and higher-order functional equations solved by higher-order unification (Felty and Miller 1988). Syntactic manipula-

tion of object-level variables and binders would be neither necessary nor possible.

The investigation reported in this paper was motivated by the use of the free-variable constraint in an earlier system of semantic-interpretation rules (Pollack and Pereira 1988; Pereira and Pollack in press). Those rules operate on objects formally analogous to sequents, with assumptions for quantified noun phrases and pronouns as well as for some other purposes. It would be worth seeing to what extent those formal operations on sequent-like objects can be mapped onto the sequent rules for some version of the functionality calculus.

Verb-phrase ellipsis and gapping constructions provide another possible area of application for the kind of categorial semantics sketched in this paper. A semantic account of those constructions requires the extraction of the meaning of elided material from the meaning of a source clause. Intermediate results of appropriate type in derivations of the meaning of the source clause may provide the possible meanings of the elided material. In other words, possible antecedent meanings would be obtained by semantic reanalysis of the source clause, where a semantic reanalysis of the source clause is just an alternative categorial derivation of the source clause's meaning. Initial results along these lines will be reported elsewhere (Dalrymple et al. 1990).

Acknowledgments

The work described here was carried out in part at SRI International and supported by a contract with the Nippon Telephone and Telegraph Corporation and 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, Stanford, CA. Mary Dalrymple, David Israel, Aravind Joshi, Dale Miller, Martha Pollack, and Stuart Shieber made useful suggestions and comments. This paper is a revised and expanded version of "A Calculus for Semantic Composition and Scoping" presented at the 1989 meeting of the Association for Computational Linguistics (Pereira 1989).

References

Anderson, A. R. and Belnap, Jr., N. D. 1975 *Entailment: The Logic of Relevance and Necessity,* Volume I. Princeton University Press, Princeton, NJ.

Alshawi, H.; Carter, D. M.; van Eijck, J.; Moore, R. C.; Moran, D. B.; Pereira, F. C. N.; Pulman, S. G.; and Smith, A. G. 1989 "Research Programme in Natural Language Processing: Final Report." Technical report, Cambridge Research Centre, SRI International, Cambridge, U.K.

Barwise, J. and Cooper, R. 1981 "Generalized Quantifiers and Natural Language." *Linguistics and Philosophy,* 4:159–219.

Curry, H. B. and Feys, R. 1968 *Combinatory Logic, Volume I,* North-Holland, Amsterdam, Netherlands.

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

Dalrymple, M.; Shieber, S. M.; and Pereira, F. C. N. 1990 "Ellipsis and Higher-Order Unification," Unpublished paper, 1990.

Dowty, D. R.; Wall, R. E.; and Peters, S. 1981 "Introduction to Montague Semantics," Volume 11 of *Synthese Language Library.* D. Reidel, Dordrecht, Netherlands.

Felty, A. and Miller, D. 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, PA.

Gazdar, G. 1982 "Phrase Structure Grammar," In P. Jacobson and G. K. Pullum (eds.), *The Nature of Syntactic Representation,* D. Reidel, Dordrecht, Netherlands, 131–186.

Girard, J.-Y.; Lafont, Y.; and Taylor, P. 1989 *Proofs and Types,* Volume 7 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, Cambridge, U.K.

Heim, I. R. 1982 *The Semantics of Definite and Indefinite Noun Phrases.* Ph.D. Thesis, Department of Linguistics, University of Massachusetts, Amherst, MA.

Hendriks, H. 1987 "Type Change in Semantics: The Scope of Quantification and Coordination," In E. Klein and J. van Benthem (eds.), *Categories, Polymorphism and Unification,* Centre for Cognitive Science, University of Edinburgh, Edinburgh, U.K., 95–120.

Halvorsen, P.-K. and Kaplan, R. M. 1988 "Projections and Semantic Description in Lexical-Functional Grammar," In *Proceedings of the International Conference on Fifth Generation Computer Systems,* Tokyo, Japan, 1116–1122.

Howard, W. A. 1980 The Formulae-As-Types Notion of Construction," In J. P. Seldin and J. R. Hindley (eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism,* Academic Press, London, 479–490.

Hobbs, J. R. and Shieber, S. M. 1987 "An Algorithm for Generating Quantifier Scopings." *Computational Linguistics,* 13:47–63.

Lambek, J. 1958 "The Mathematics of Sentence Structure." *American Mathematical Monthly,* 65:154–170.

Lambek, J. 1980 "From λ-Calculus to Cartesian Closed Categories," In J. P. Seldin and J. R. Hindley (eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism,* Academic Press, London, 375–402.

May, R. 1985 *Logical Form: Its Structure and Derivation,* Volume 12 of *Linguistic Inquiry Monographs.* MIT Press, Cambridge, MA.

Miller, D. A. and Nadathur, G. 1986 "Higher-Order Logic Programming," In E. Shapiro (ed.), *Third International Conference on Logic Programming,* Springer-Verlag, Berlin.

Montague, R. 1973 "The Proper Treatment of Quantification in Ordinary English," In R. H. Thomason (ed.), *Formal Philosophy.* Yale University Press, New Haven, CT.

Moortgat, M. 1988 *Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus.* Ph.D. Thesis, University of Amsterdam, Amsterdam, Netherlands.

Moran, D. B. 1988 "Quantifier Scoping in the SRI Core Language Engine," *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics,* 33–47.

Partee, B. and Bach, E. 1984 "Quantification, Pronouns and VP Anaphora," In J. A. G. Groenendijk, T. M. V. Janssen, and M. B. J. Stokhof (eds.), *Truth, Interpretation and Information,* Foris, Dordrecht, Netherlands, 99–130.

Pereira, F. C. N. 1989 "A Calculus for Semantic Composition and Scoping," *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics,* 152–160.

Pollack, M. E. and Pereira, F. C. N. 1988 "An Integrated Framework for Semantic and Pragmatic Interpretation," *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics,* 75–86.

Pereira, F. C. N. and Pollack, M. E. "Incremental Interpretation," *Artificial Intelligence,* in press.

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

Reinhart, T. 1983 *Anaphora and Semantic Interpretation.* Croom Helm, London.

Roberts, C. 1987 *Modal Subordination, Anaphora and Distributivity.* Ph.D. Thesis, Department of Linguistics, University of Massachusetts, Amherst, MA.

Stenlund, S. 1972 *Combinators, λ-Terms and Proof Theory.* D. Reidel, Dordrecht, Netherlands.

Steedman, M. 1987 "Combinatory Grammars and Parasitic Gaps." *Natural Language and Linguistic Theory,* 5(3):403–439.

Vanlehn, K. A. 1978 *Determining the Scope of English Quantifiers.* M.S. thesis, Massachusetts Institute of Technology, Cambridge, MA.

van Benthem, J. 1986a "Categorial Grammar and Lambda Calculus," In D. Skordev (ed.), *Mathematical Logic and its Application,* Plenum Press, New York, 39–60.

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

van Benthem, J. 1989 "Categorial Grammar and Type Theory." *Journal of Philosophical Logic,* In press, 1990.

van Eijck, J. 1985 *Aspects of Quantification in Natural Language.* Ph.D. Thesis, University of Groningen, Groningen, Netherlands.

van Riemsdijk, H. and Williams, E. 1986 *Introduction to the Theory of Grammar,* Volume 12 of *Current Studies in Linguistics.* MIT Press, Cambridge, MA.

Williams, E. 1986 "A Reassignment of the Functions of LF." *Linguistic Inquiry,* 17(2):265–299.

Notes

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.
3. This particular example and its analysis were chosen just as the shortest plausible example requiring both application and abstraction, not as making substantive linguistic or semantic claims.
4. These rules are quite similar to the *extraction introduction* rule of Moortgat (1988).
5. 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.