Quantifiers, Anaphora, and Intensionality

Mary Dalrymple*
John Lamping*
Fernando Pereira†
Vijay Saraswat*

September 16, 1999

The relationship between Lexical-Functional Grammar (LFG) *functional structures* (f-structures) for sentences and their semantic interpretations can be expressed directly in a fragment of linear logic in a way that correctly explains the constrained interactions between quantifier scope ambiguity, bound anaphora and intensionality.

The use of a deductive framework to account for the compositional properties of quantifying expressions in natural language obviates the need for additional mechanisms, such as Cooper storage, to represent the different scopes that a quantifier might take. Instead, the semantic contribution of a quantifier is recorded as a logical formula whose use in a proof will establish the scope of the quantifier. Different proofs will in general lead to different scopes. In each complete proof, the properties of linear logic will ensure that each quantifier is properly scoped.

The interactions between quantified NPs and intensional verbs such as 'seek' are also accounted for in this deductive setting. A single specification in linear logic of the argument requirements of intensional verbs is sufficient to derive the correct reading predictions for intensional-verb clauses both with nonquantified and with quantified direct objects. In particular, both *de dicto* and *de re* readings are derived for quantified objects. The effects of type-raising or quantifying-in rules in other frameworks here just follow as linear-logic theorems.

While our approach resembles current categorial approaches in important ways (Moortgat, 1988; Moortgat, 1992a; Morrill, 1993; Carpenter, 1993), it differs from them in allowing the greater type flexibility of categorial semantics (van Benthem, 1991) while maintaining a precise connection to syntax. As a result, we are able to provide derivations for certain readings of sentences with intensional verbs and complex direct objects that are not derivable in current purely categorial accounts of the syntax-semantics interface.

*Xerox PARC, Palo Alto CA 94304; {dalrymple,lamping,saraswat}@parc.xerox.com
†AT&T Bell Laboratories, Murray Hill NJ 07974; pereira@research.att.com

1 Introduction

This paper describes a part of our ongoing investigation in the use of formal deduction to explicate the relationship between syntactic analyses in Lexical-Functional Grammar (LFG) and semantic interpretations. We use *linear logic* (Girard, 1987) to represent the connection between two dissimilar linguistic levels: LFG f-structures and their semantic interpretations.

F-structures provide a uniform representation of syntactic information relevant to semantic interpretation that abstracts away from the varying details of phrase structure and linear order in particular languages. As Halvorsen (1988) notes, however, the flatter, unordered functional structure of LFG does not fit well with traditional semantic compositionality, based on functional abstraction and application, which mandates a rigid order of semantic composition. We are thus led to a more relaxed form of compositionality, in which, as in more traditional ones, the semantics of each lexical entry in a sentence is used exactly once in interpretation, but without imposing a rigid order of composition. Approaches to semantic interpretation that encode semantic representations in attribute-value structures (Pollard and Sag, 1987; Fenstad et al., 1987; Pollard and Sag, 1993) offer such a relaxation of compositionality, but are unable to properly represent constraints on variable binding and scope (Pereira, 1990).

The present approach, in which linear logic is used to specify the relation between f-structures and their meanings, provides exactly what is required for a calculus of semantic composition for LFG. It can directly represent the constraints on the creation and use of semantic units in sentence interpretation, including those pertaining to variable binding and scope, without forcing a particular hierarchical order of composition, except as required by the properties of particular lexical entries.

The use of formal deduction in semantic interpretation was implicit in deductive systems for categorial syntax (Lambek, 1958), and has been made explicit through applications of the Curry-Howard parallelism between proofs and terms in more recent work on categorial semantics (van Benthem, 1988; van Benthem, 1991), labeled deductive systems (Moortgat, 1992b) and flexible categorial systems (Hendriks, 1993). Accounts of the syntax-semantics interface in the categorial tradition require that syntactic and semantic analyses be formalized in parallel algebraic structures of similar signatures, based on generalized application and abstraction (or residuation) operators and structure-preserving relations between them. Those accounts therefore force the adoption of categorial syntactic analyses, with an undesirably strong dependence on phrase structure and linear order.

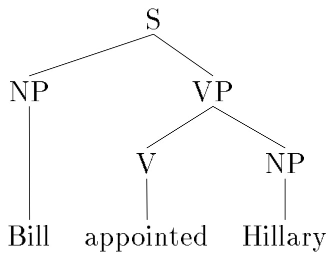
We have previously shown that the linear-logic formalization of the syntax-semantics interface for LFG provides simple and general analyses of modification, functional completeness and coherence, and complex predicate formation (Dalrymple, Lamping, and Saraswat, 1993; Dalrymple et al., 1993). In the present paper, the analysis is extended to the interpretation of quantified NPs. After an overview of the approach, we present our analysis of the compositional properties of quantified NPs, and we show that the analysis correctly accounts for scope ambiguity and its interactions with bound anaphora. We also present an analysis of intensional verbs, which take quantified arguments, and show that our approach predicts the full range of acceptable readings without appealing to additional machinery.

2 LFG and Linear Logic

Syntactic framework LFG assumes two syntactic levels of representation. Constituent structure (*c-structure*) encodes phrasal dominance and precedence relations, and is represented as a phrase structure tree. Functional structure (*f-structure*) encodes syntactic predicate-argument structure, and is represented as an attribute-value matrix. The c-structure and f-structure for sentence (1) are given in (2):

(1) Bill appointed Hillary.

(2) C-structure:



F-structure:

$$\begin{bmatrix} \textsc{pred} & \text{`}\textsc{appoint}\text{'} \\ \textsc{subj} & \begin{bmatrix} \textsc{pred} & \text{`}\textsc{Bill}\text{'} \end{bmatrix} \\ \textsc{obj} & \begin{bmatrix} \textsc{pred} & \text{`}\textsc{Hillary}\text{'} \end{bmatrix} \end{bmatrix}$$

As illustrated, an f-structure consists of a collection of attributes, such as PRED, SUBJ, and OBJ, whose values can, in turn, be other f-structures.

The relationship between c-structure trees and the corresponding f-structures is given by a *functional projection* function ϕ from c-structure nodes to f-structures. More generally, LFG analyses involve several levels of linguistic representation called *projections* related by means of *projection functions* (Kaplan, 1987; Halvorsen and Kaplan, 1988). For instance, phonological, morphological, or discourse structure might be represented by a phonological, morphological, or discourse projection, related to other projections by means of functional specifications.

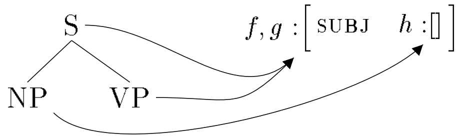
The functional projection of a c-structure node is the solution of constraints associated with the phrase-structure rules and lexical entries used to derive the node. In each rule or lexical entry constraint, the $\uparrow$ metavariable refers to the ϕ-image of the mother c-structure node, and the $\downarrow$ metavariable refers to the ϕ-image of the nonterminal labeled by the constraint (Kaplan and Bresnan, 1982, page 183). For example, the following annotated phrase-structure rules were used in the analysis of sentence (1):

(3) $$\begin{array}{lccc} \text{S} & \longrightarrow & \text{NP} & \text{VP} \\ & & (\uparrow \textsc{subj}) = \downarrow & \uparrow = \downarrow \end{array}$$

The annotations on the rule indicate that the f-structure for the S ($\uparrow$ in the annotation on the NP node) has a SUBJ attribute whose value is the f-structure for the NP daughter ($\downarrow$ in the annotation on the NP node), and that the S node corresponds to an f-structure which is the same as the f-structure for the VP daughter.

When the phrase-structure rule for S is used in the analysis of a particular sentence, the metavariables $\uparrow$ and $\downarrow$ are instantiated to particular f-structures placed in correspondence with nodes of the c-structure. We will refer to actual f-structures by giving them names such as f, g, and h. The instantiated phrase structure rule is given in (4), with the ϕ correspondence between c-structure nodes and f-structures indicated by the directed arcs from phrase-structure nodes to attribute-value matrices:

(4) $$\text{S} \longrightarrow \begin{array}{cc} \text{NP} & \text{VP} \\ (f\ \textsc{subj}) = h & f = g \end{array}$$



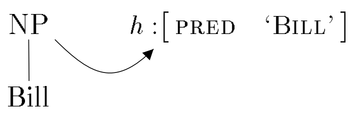
Lexical entries also use the metavariables $\uparrow$ and $\downarrow$ to encode information about the f-structures of the preterminal nodes that immediately dominate them. A partial lexical entry for the word 'Bill' is:

(5) Bill NP $(\uparrow\ \textsc{pred}) = \text{`}\textsc{Bill}\text{'}$

This entry states that 'Bill' has syntactic category NP. The constraint $(\uparrow\ \textsc{pred}) = \text{`}\textsc{Bill}\text{'}$ states that the preterminal node immediately dominating the terminal symbol 'Bill' has an f-structure whose value for the attribute PRED is 'BILL'. In this paper, we will provide only the most minimal f-structural representations, leaving aside all details of syntactic specification; in this example, for instance, agreement and other syntactic features of 'Bill' have been omitted.

For a particular instance of use of the word 'Bill', the following c-structure and f-structure configuration results:

(6) $(h\ \textsc{pred}) = \text{`}\textsc{Bill}\text{'}$



Other lexical entries similarly specify features of the f-structure of the immediately dominating preterminal node. The following is a list of the phrase structure rules and lexical entries used in the analysis of example (1):[1]

(7) $\text{S} \longrightarrow \underset{(\uparrow \text{SUBJ}) = \downarrow}{\text{NP}} \quad \underset{\uparrow = \downarrow}{\text{VP}}$

$\text{VP} \longrightarrow \underset{\uparrow = \downarrow}{\text{V}} \quad \underset{(\uparrow \text{OBJ}) = \downarrow}{\text{NP}}$

(8) Bill NP $(\uparrow \text{PRED}) =$ 'BILL'

appointed V $(\uparrow \text{PRED})=$ 'APPOINT'

Hillary NP $(\uparrow \text{PRED}) =$ 'HILLARY'

For a more complete explication of the syntactic assumptions of LFG, see Bresnan (1982), Levin, Rappaport, and Zaenen (1983), and the references cited there.

Lexically-specified semantics A distinguishing feature of our work (and of other work within the LFG framework) is that semantic composition does not take syntactic dominance and precedence relations as the main input. Instead, we follow other work in LFG (Kaplan and Bresnan 1982, Halvorsen 1983, Halvorsen and Kaplan 1988, and many others) in assuming that the functional syntactic information encoded by f-structures determines semantic composition. That is, we believe that meaning composition is mainly determined by syntactic relations such as *subject-of*, *object-of*, *modifier-of*, and so on. Those relations are realized by different c-structure forms in different languages, but are represented directly and uniformly in the f-structure.

In LFG, syntactic predicate-argument structure is projected from lexical entries. Therefore, its effect on semantic composition will for the most part – in fact, in all the cases considered in this paper – be determined by lexical entries, not by phrase-structure rules. In particular, the two phrase-structure rules given above for S and VP need not encode semantic information, but only specify how grammatical functions such as SUBJ are expressed in English. In some cases, the constituent structure of a syntactic construction may make a direct semantic contribution, as when properties of the construction as a whole and not just of its lexical elements are responsible for the interpretation of the construction. Such cases include, for instance, relative clauses with no complementizer ('the man Bill met'). We will not discuss construction-specific interpretation rules in this paper.

In the same way as the functional projection function ϕ associates f-structures to c-structures as described above, we will use a *semantic* or *σ-projection* function σ to map f-structures to *semantic* or *σ-structures* encoding information about f-structure meaning.

[1]Those familiar with other analyses within the LFG framework will notice that we have not included a list of grammatical functions subcategorized for by the verb 'appoint'; this is because we assume a different treatment of the LFG requirements of completeness and coherence. We return to this point below.

For instance, the following lexical entry for 'Bill' extends (8) with a suitable constraint on semantic structure:

(9) Bill NP $(\uparrow \textsc{pred}) = \text{`}\textsc{Bill}\text{'}$
$\uparrow_\sigma \rightsquigarrow \mathit{Bill}$

The additional constraint

$$\uparrow_\sigma \rightsquigarrow \mathit{Bill}$$

is what we call the *meaning constructor* of the entry. The expression $\uparrow_\sigma$ stands for the σ *projection* of the f-structure $\uparrow$. The σ projection is an attribute-value matrix like the f-structure. For simple entries such as this, the σ projection has no internal structure; below, we will examine cases in which the σ projection is structured with several different attributes.

As above, for a particular use of 'Bill', the metavariable $\uparrow$ will be replaced by a particular f-structure h, with semantic projection h_σ:

(10) $(h \ \textsc{pred}) = \text{`}\textsc{Bill}\text{'}$

NP $\qquad h : [\ \textsc{pred} \quad \text{`}\textsc{Bill}\text{'}\ ] \qquad h_\sigma : [\ ] \rightsquigarrow \mathit{Bill}$
Bill

More generally, the association between the semantic structure h_σ and a meaning P is represented by the atomic formula $h_\sigma \rightsquigarrow P$, where $\rightsquigarrow$ is an otherwise uninterpreted binary predicate symbol. (In fact, we use not one but a family of relations $\rightsquigarrow_\tau$ indexed by the semantic type of the intended second argument, although for simplicity we will omit the type subscript whenever it is determinable from context.) We can now explain the meaning constructor in (9). If a particular occurrence of 'Bill' in a sentence is associated with f-structure h, the syntactic constraint in the lexical entry *Bill* will be instantiated as:

$$(h \ \textsc{pred}) = \text{`}\textsc{Bill}\text{'}$$

and the semantic constraint will be instantiated as:

$$h_\sigma \rightsquigarrow \mathit{Bill}$$

representing the association between h_σ and the constant *Bill* representing its meaning.

We will often informally say that P is h's meaning without referring to the role of the semantic structure h_σ in $h_\sigma \rightsquigarrow P$. We will see, however, that f-structures and their semantic projections must be distinguished, because semantic projections can carry more information than just the association to the meaning for the corresponding f-structure.

Logical representation of semantic compositionality We now turn to an examination of the lexical entry for 'appointed'. In this case, the meaning constructor is more complex, as it relates the meanings of the subject and object of a clause to the clause's meaning:

(11) appointed V $(\uparrow \text{PRED}) = \text{`APPOINT'}$
$\forall X, Y.\ (\uparrow \text{SUBJ})_\sigma \rightsquigarrow X \otimes (\uparrow \text{OBJ})_\sigma \rightsquigarrow Y \multimap \uparrow_\sigma \rightsquigarrow \mathit{appoint}(X, Y)$

The meaning constructor is the linear-logic formula:

$$\forall X, Y.\ (\uparrow \text{SUBJ})_\sigma \rightsquigarrow X \otimes (\uparrow \text{OBJ})_\sigma \rightsquigarrow Y \multimap \uparrow_\sigma \rightsquigarrow \mathit{appoint}(X, Y)$$

in which the linear-logic connectives of multiplicative conjunction $\otimes$ and linear implication $\multimap$ are used to specify how the meaning of a clause headed by the verb is composed from the meanings of the arguments of the verb. For the moment, we can think of the linear connectives as playing the same role as the analogous classical connectives conjunction $\wedge$ and implication $\rightarrow$, but we will soon see that the specific properties of the linear connectives are essential to guarantee that lexical entries bring into the interpretation process all and only the information provided by the corresponding words.

The meaning constructor for 'appointed' asserts, then, that if the subject (SUBJ) of a clause with main verb 'appointed' means X and its object (OBJ) means Y, then the whole clause means $\mathit{appoint}(X, Y)$.[2] The meaning constructor can thus be thought of as a linear definite clause, with the variables X and Y playing the same role as Prolog variables.

A particular instance of use of 'appointed' produces the following f-structure and meaning constructor:

(12) $(f\ \text{PRED}) = \text{`APPOINT'}$

V
appointed

$$f : \begin{bmatrix} \text{PRED} & \text{`APPOINT'} \\ \text{SUBJ} & [\,] \\ \text{OBJ} & [\,] \end{bmatrix} \qquad f_\sigma : [\,]$$

$\forall X, Y.\ (f\ \text{SUBJ})_\sigma \rightsquigarrow X \otimes (f\ \text{OBJ})_\sigma \rightsquigarrow Y \multimap f_\sigma \rightsquigarrow \mathit{appoint}(X, Y)$

[2] In fact, we believe that the correct treatment of the relation between a verb and its arguments requires the use of *mapping principles* specifying the relation between the array of semantic arguments required by a verb and their possible syntactic realizations (Bresnan and Kanerva, 1989; Alsina, 1993; Butt, 1993). A verb like 'appoint', for example, might specify that one of its arguments is an agent and the other is a theme. Mapping principles would then specify that agents can be realized as subjects and themes as objects.

Here we make the simplifying assumption (valid for English) that the arguments of verbs have already been linked to syntactic functions and that this linking is represented in the lexicon. In the case of *complex predicates* this assumption produces incorrect results, as shown by Butt (1993) for Urdu. Mapping principles are very naturally incorporated into the framework discussed here; see Dalrymple, Lamping, and Saraswat (1993) and Dalrymple et al. (1993) for discussion and illustration.

The instantiated meaning constructor asserts that f is the f-structure for a clause with predicate (PRED) 'APPOINT', and:

- if f's subject (f SUBJ) has meaning X
- and ($\otimes$) f's object (f OBJ) has meaning Y
- then ($\multimap$) f has meaning $appoint(X, Y)$.

It is not an accident that the form of the meaning constructor for *appointed* is analogous to the type $(e \times e) \rightarrow t$ which, in its curried form $e \rightarrow e \rightarrow t$, is the standard type for a transitive verb in a compositional semantics setting (Gamut, 1991). In general, the propositional structure of the meaning constructors of lexical entries will parallel the types assigned to the meanings of the same words in compositional analyses.

As mentioned above, in most cases phrase-structure rules make no semantic contributions of their own. Thus, all the semantic information for a sentence like 'Bill appointed Hillary' is provided by the lexical entries for 'Bill', 'appointed', and 'Hillary':

(13) Bill NP $(\uparrow \text{PRED}) = \text{`BILL'}$
$\uparrow_\sigma \rightsquigarrow Bill$

appointed V $(\uparrow \text{PRED}) = \text{`APPOINT'}$
$\forall X, Y.\ (\uparrow \text{SUBJ})_\sigma \rightsquigarrow X \otimes (\uparrow \text{OBJ})_\sigma \rightsquigarrow Y \multimap \uparrow_\sigma \rightsquigarrow appoint(X, Y)$

Hillary NP $(\uparrow \text{PRED}) = \text{`HILLARY'}$
$\uparrow_\sigma \rightsquigarrow Hillary$

Assembly of meanings via deduction We have now the ingredients for building semantic interpretations by deductive means. To recapitulate the development so far, lexical entries provide semantic constructors, which are linear-logic formulas specifying how the meanings of f-structures are built from the meanings of their substructures. Thus, linear logic serves as a *glue language* to assemble meanings. Certain terms in the glue language represent (open) formulas of an appropriate *meaning language*, which for the present purposes will be a version of Montague's intensional logic (Montague, 1974).[3] Other terms in the glue language represent semantic projections. The glue-language formula $f \rightsquigarrow t$, with f a term representing a semantic projection and t a term representing a meaning-language formula, expresses the association between the semantic projection denoted by f and the meaning fragment denoted by t.

The fragment of linear logic we use as glue language will be described incrementally as we discuss examples, and is summarized in Appendix A. The semantic contribution of each lexical entry is a linear-logic formula, its meaning constructor, that can be understood

[3]The reader familiar with Montague may be surprised by the apparently purely extensional form of the meaning terms in the examples that follow, in contrast with Montague's use of intensional expressions even in purely extensional cases to allow for uniform translation rules. The reasons for this divergence are explained in Section 4.

as "instructions" for combining the meanings of the lexical entry's syntactic arguments to obtain the meaning of the f-structure headed by the entry. In the case of the verb 'appointed' above, the meaning constructor is a glue language formula consisting of instructions on how to assemble the meaning of a sentence with main verb 'appointed', given the meanings of its subject and object.

We will now show how meanings are assembled by linear-logic deduction. The full set of proof rules relevant to this paper is given in Appendix B. For readability, however, we will present derivations informally in the main body of the paper. As a first example, consider the lexical entries in (13) and let the constants f, g and h name the following f-structures:

(14) $$f{:}\begin{bmatrix} \textsc{pred} & \text{`}\textsc{appoint}\text{'} \\ \textsc{subj} & g{:}\begin{bmatrix} \textsc{pred} & \text{`}\textsc{Bill}\text{'} \end{bmatrix} \\ \textsc{obj} & h{:}\begin{bmatrix} \textsc{pred} & \text{`}\textsc{Hillary}\text{'} \end{bmatrix} \end{bmatrix}$$

Instantiating the lexical entries for 'Bill', 'Hillary', and 'appointed' appropriately, we obtain the following meaning constructors, abbreviated as **bill**, **hillary**, and **appointed**:

$$\begin{array}{ll} \textbf{bill}: & g_\sigma \rightsquigarrow \mathit{Bill} \\ \textbf{hillary}: & h_\sigma \rightsquigarrow \mathit{Hillary} \\ \textbf{appointed}: & \forall X, Y.\ g_\sigma \rightsquigarrow X \otimes h_\sigma \rightsquigarrow Y \multimap f_\sigma \rightsquigarrow \mathit{appoint}(X, Y) \end{array}$$

These formulas show how the generic semantic contributions in the lexical entries are instantiated to reflect their participation in this particular f-structure. Since the entry 'Bill' gives rise to f-structure g, the meaning constructor for 'Bill' provides a meaning for g_σ. Similarly, the meaning constructor for 'Hillary' provides a meaning for h_σ. The verb 'appointed' requires two pieces of information, the meanings of its subject and object, in no particular order, to produce a meaning for the clause. As instantiated, the f-structures corresponding to the subject and object of the verb are g and h, respectively, and f is the f-structure for the entire clause. Thus, the instantiated entry for 'appointed' shows how to combine a meaning for g_σ (its subject) and h_σ (its object) to generate a meaning for f_σ (the entire clause).

In the following, assume that the formula **bill-appointed** is defined thus:

$$\textbf{bill-appointed}:\quad \forall Y.\ h_\sigma \rightsquigarrow Y \multimap f_\sigma \rightsquigarrow \mathit{appoint}(\mathit{Bill}, Y)$$

Then the following derivation is possible in linear logic ($\vdash$ stands for the linear-logic entailment relation):

(15) $$\begin{array}{lll} & \textbf{bill} \otimes \textbf{hillary} \otimes \textbf{appointed} & (\mathit{Premises.}) \\ \vdash & \textbf{bill-appointed} \otimes \textbf{hillary} & X \mapsto \mathit{Bill} \\ \vdash & f_\sigma \rightsquigarrow \mathit{appoint}(\mathit{Bill}, \mathit{Hillary}) & Y \mapsto \mathit{Hillary} \end{array}$$

Each formula is annotated with the variable substitutions (universal instantiations) required to derive it from the preceding one by the modus ponens rule $A \otimes (A \multimap B) \vdash B$.

Of course, another derivation is also possible. Assume that the formula **appointed-hillary** is defined as:

$$\textbf{appointed-hillary}: \quad \forall X.\, g_\sigma \rightsquigarrow X \multimap f_\sigma \rightsquigarrow appoint(X, Hillary)$$

Then we have the following derivation:

$$\begin{array}{lll}
(16) & \textbf{bill} \otimes \textbf{hillary} \otimes \textbf{appointed} & (Premises.) \\
\vdash & \textbf{bill} \otimes \textbf{appointed-hillary} & Y \mapsto Hillary \\
\vdash & f_\sigma \rightsquigarrow appoint(Bill, Hillary) & X \mapsto Bill
\end{array}$$

In summary, each word in a sentence contributes a linear-logic formula, its meaning constructor, relating the semantic projections of specific f-structures in the LFG analysis to representations of their meanings. From these glue language formulas, the interpretation process attempts to deduce an atomic formula relating the semantic projection of the whole sentence to a representation of the sentence's meaning. Alternative derivations may yield different such conclusions, corresponding to ambiguities of semantic interpretation.

Linear logic As we have just outlined, we use deduction in linear logic to assign meanings to sentences, starting from information about their functional structure and about the semantic contributions of their words. Traditional compositional approaches depend on a strict separation between functors and arguments, typically derived from a binary-branching phrase-structure tree. In contrast, our linear-logic-based approach allows the premises carrying semantic information to commute while keeping their connection to the f-structure, and is thus more compatible with the flat and relatively free form organization of functional structure.

An important motivation for using linear logic is that it allows us to directly capture the intuition that lexical items and phrases each contribute exactly once to the meaning of a sentence. As noted by Klein and Sag (1985, page 172):

> Translation rules in Montague semantics have the property that the translation of each component of a complex expression occurs exactly once in the translation of the whole. ...That is to say, we do not want the set S [of semantic representations of a phrase] to contain *all* meaningful expressions of IL which can be built up from the elements of S, but only those which use each element exactly once.

In our terms, the semantic contributions of the constituents of a sentence are not context-independent assertions that may be used or not in the derivation of the meaning of the sentence depending on the course of the derivation. Instead, the semantic contributions are *occurrences* of information which are generated and used exactly once. For example, the formula $g_\sigma \rightsquigarrow Bill$ can be thought of as providing one occurrence of the meaning *Bill* associated to the semantic projection g_σ. That meaning must be consumed exactly once (for example, by **appointed** in (15)) in the derivation of a meaning of the entire utterance.

It is this "resource-sensitivity" of natural language semantics—an expression is used exactly once in a semantic derivation—that linear logic can model. The basic insight underlying linear logic is that logical formulas are *resources* that are produced and consumed in the deduction process. This gives rise to a resource-sensitive notion of implication, the *linear implication* $\multimap$: the formula $A \multimap B$ can be thought of as an action that can *consume* (one copy of) A to produce (one copy of) B. Thus, the formula $A \otimes (A \multimap B)$ linearly entails B. It does not entail $A \otimes B$ (because the deduction consumes A), and it does not entail $(A \multimap B) \otimes B$ (because the linear implication is also consumed in doing the deduction). This resource-sensitivity not only disallows arbitrary duplication of formulas, but also disallows arbitrary deletion of formulas. Thus the linear multiplicative conjunction $\otimes$ is sensitive to the multiplicity of formulas: $A \otimes A$ is not equivalent to A (the former has two copies of the formula A). For example, the formula $A \otimes A \otimes (A \multimap B)$ linearly entails $A \otimes B$ (there is still one A left over) but does not entail B (there must still be one A present). In this way, linear logic checks that a formula is used once and only once in a deduction, enforcing the requirement that each component of an utterance contributes exactly once to the assembly of the utterance's meaning.

A direct consequence of the above properties of linear logic is that the constraints of functional *completeness* and *coherence* hold without further stipulation[4] (Dalrymple, Lamping, and Saraswat, 1993). In the present setting, the feature structure f corresponding to the utterance is associated with the $(\otimes)$ conjunction ϕ of all the formulas associated with the lexical items in the utterance. The conjunction is said to be *complete* and *coherent* iff $Th \vdash \phi \multimap f_\sigma \rightsquigarrow t$ (for some term t), where Th is the background theory of general linguistic principles. Each t is to be thought of as a valid meaning for the sentence. This guarantees that the entries are used exactly once in building up the denotation of the utterance: no syntactic or semantic requirements may be left unfulfilled, and no meaning may remain unused.

Our glue language needs to be only a fragment of higher-order linear logic, the *tensor fragment*, that is closed under conjunction, universal quantification, and implication. This fragment arises from transferring to linear logic the ideas underlying the concurrent constraint programming scheme of Saraswat (1989).[5]

---

[4] 'An f-structure is *locally complete* if and only if it contains all the governable grammatical functions that its predicate governs. An f-structure is *complete* if and only if all its subsidiary f-structures are locally complete. An f-structure is *locally coherent* if and only if all the governable grammatical functions that it contains are governed by a local predicate. An f-structure is *coherent* if and only if all its subsidiary f-structures are locally coherent.' (Kaplan and Bresnan, 1982, pages 211–212)

To illustrate:

(a) *John devoured. [incomplete]

(b) *John arrived Bill the sink. [incoherent]

[5] Saraswat and Lincoln (1992) provide an explicit formulation for the higher-order version of the linear concurrent constraint programming scheme. Scedrov (1993) gives a tutorial introduction to linear logic itself; Saraswat (1993) supplies further background on computational aspects of linear logic relevant to the implementation of the present proposal.

Relationship with Categorial Syntax and Semantics As suggested above, there are close connections between our approach and various systems of categorial syntax and semantics. The Lambek calculus (Lambek, 1958), introduced as a logic of syntactic combination, turns out to be a fragment of noncommutative multiplicative linear logic. If permutation is added to Lambek's system, its left- and right-implication connectives (\ and /) collapse into a single implication connective with behavior identical to $\multimap$. This undirected version of the Lambek calculus was developed by van Benthem (1988; 1991) to account for the semantic combination possibilities of phrase meanings.

Those systems and related ones (Moortgat, 1988; Hepple, 1990; Morrill, 1990) were developed as calculi of syntactic/semantic types, with propositional formulas representing syntactic categories or semantic types. Given the types for the lexical items in a sentence as assumptions, the sentence is syntactically well-formed in the Lambek calculus if the type of the sentence can be derived from the assumptions arranged as an ordered list. Furthermore, the Curry-Howard isomorphism between proofs and terms (Howard, 1980) allows the extraction of a term representing the meaning of the sentence from the proof that the sentence is well-formed (van Benthem, 1986). However, the Lambek calculus and its variants carry with them a particular view of the syntax-semantics interface which is not obviously compatible with the flatter f-structures of LFG. In Section 5, we will examine more closely the differences between those approaches and ours.

On the other hand, categorial semantics in the undirected Lambek calculus and other related commutative calculi provide an analysis of the possibilities of meaning combination independently of the syntactic realizations of those meanings, but does not offer a mechanism for relating semantic combination possibilities to the corresponding syntactic combination possibilities.

Our system follows categorial semantics in using the "propositional skeleton" of glue formulas to encode the types of phrase meanings and thus their composition potential. In addition, however, first-order quantification over semantic projections maintains the connection between those types and the corresponding syntactic objects, while quantification over semantic terms is used to build the meanings of those syntactic objects. This tripartite organization reflects the three linked systems of representation that participate in semantic interpretation: syntactic structure, semantic types and semantic interpretations themselves. In this way, we can take advantage of the principled description of potential meaning combinations arising from categorial semantics without losing track of the constraints imposed by syntax on the possible combinations of those meanings.

3 Quantification

Our treatment of quantification, and in particular of quantifier scope ambiguities and of the interactions between scope and bound anaphora, follows the analysis of Pereira (1990; 1991), but offers in addition a formal account of the syntax-semantics interface, which was treated only informally in that earlier work.

3.1 Quantifier meanings

The basic idea for the analysis can be seen as a logical counterpart at the glue level of the standard type assignment for generalized quantifiers (Barwise and Cooper, 1981). The generalized quantifier meaning of a natural language determiner has the following type:

(17) $(e \to t) \to (e \to t) \to t$

that is, the type of functions from two properties, the quantifier's restriction and scope, to propositions. At the semantic glue level, we can understand that type as follows. For any determiner, if for arbitrary x we can construct a meaning $R(x)$ for the quantifier's restriction, and again for arbitrary x we can construct a meaning $S(x)$ for the quantifier's scope, where R and S are suitable properties (functions from entities to propositions), then we can construct the meaning $Q(R, S)$ for the whole sentence containing the determiner, where Q is the meaning of the determiner.

Assume for the moment that we have determined the following semantic structures: *restr* for the restriction (a common noun phrase), *restr-arg* for its implicit argument, *scope* for the scope of quantification, and *scope-arg* for the grammatical function filled by the quantified NP. Then the foregoing analysis can be represented in linear logic by the following schematic formula:[6]

(18) $$\begin{aligned} \forall R, S.\ & (\forall x.\ \mathit{restr\text{-}arg} \rightsquigarrow x \multimap \mathit{restr} \rightsquigarrow R(x)) \\ & \otimes (\forall x.\ \mathit{scope\text{-}arg} \rightsquigarrow x \multimap \mathit{scope} \rightsquigarrow S(x)) \\ & \multimap \mathit{scope} \rightsquigarrow Q(R, S) \end{aligned}$$

Given the equivalence between $A \otimes B \multimap C$ and $A \multimap (B \multimap C)$, the propositional part of (18) parallels the generalized quantifier type (17).

In addition to providing a semantic type assignment for determiners, (18) uses glue language quantification to express how the meanings of the restriction and scope of quantification are determined and combined into the meaning of the quantified clause. The subformula

$$\forall x.\ \mathit{restr\text{-}arg} \rightsquigarrow x \multimap \mathit{restr} \rightsquigarrow R(x)$$

specifies that *restr* has meaning $R(x)$ if for arbitrary x *restr-arg* has meaning x, that is, it gives the dependency of the meaning of a common noun phrase on its implicit argument. Property R is the representation of that dependency as a function in the meaning language. Similarly, the subformula

$$\forall x.\ \mathit{scope\text{-}arg} \rightsquigarrow x \multimap \mathit{scope} \rightsquigarrow S(x)$$

[6] We use lower-case letters for *essentially universal* variables, that is, variables that stand for new local constants in a proof. We use capital letters for *essentially existential* variables, that is, Prolog-like variables that become instantiated to particular terms in a proof. In other words, essentially existential variables stand for specific but as yet unspecified terms, while essentially universal variables stand for arbitrary constants, that is, constants that could be replaced by *any* term while still maintaining the validity of the derivation. In the linear-logic fragment we use here, essentially existential variables arise from universal quantification with outermost scope, while essentially universal variables arise from universal quantification whose scope is a conjunct in the antecedent of an outermost implication.

specifies the dependency of the meaning $S(x)$ of a semantic structure *scope* on the meaning x of one of its arguments *scope-arg*. If both dependencies hold, then R and S are an appropriate restriction and scope for the determiner meaning Q.

Computationally, the nested universal quantifiers substitute unique new constants (eigenvariables) for the quantified variable x, and the nested implications try to prove their consequents with their antecedents added to the current set of assumptions. For the restriction (the case of the scope is similar), this will in particular involve solving an equation of the form $R(x) = t$, where $\mathit{restr} \rightsquigarrow t$ has been derived. The equation must be solved modulo α-, β- and η-conversion, and any solution R must not contain occurrences of x, since R's scope is wider than x's. Higher-order unification (Huet, 1975) is a procedure suitable for solving such equations.[7]

3.2 Quantifier restrictions

We have seen that since the meaning of the restriction of a quantifier is a property (type $e \to t$), its meaning constructor has the form of an implication, just like a verb. In (18), the first line of the determiner's semantic constructor

$$(\forall x.\ \mathit{restr\text{-}arg} \rightsquigarrow x \multimap \mathit{restr} \rightsquigarrow R(x))$$

requires a meaning x for *restr-arg* to produce the meaning $R(x)$ for *restr*, defining the restriction R of the quantifier. We need thus to identify the semantic projections *restr-arg* and *restr*.

The f-structure of a quantified NP has the general form:

(19) $$f\!:\!\begin{bmatrix} \textsc{spec} & q \\ \textsc{pred} & n \end{bmatrix}$$

where q is the determiner f-structure and n the noun f-structure. None of the f-structures f, q or n is a natural syntactic correlate of the argument or result of the quantifier restriction. This contrasts with the treatment of verbs, whose semantic contributions and argument dependencies are directly associated with appropriate syntactic units of the clauses they head. Therefore, we take the semantic projection f_σ of the quantified NP to be structured with two attributes (f_σ VAR) and (f_σ RESTR):

(20) $$\begin{array}{c}\text{Det}\\ | \\ \text{every}\end{array} \qquad f : \begin{bmatrix} \textsc{spec} & \text{`}\textsc{every}\text{'} \end{bmatrix} \qquad f_\sigma : \begin{bmatrix} \textsc{var} & [\ ] \\ \textsc{restr} & [\ ] \end{bmatrix}$$

[7] While higher-order unification is in general undecidable, the unification problems involved here are of one of the forms $F(x) = t$ or $p(X) = t$ where t is a closed term, F and X essentially existential variables and x and p essentially universal variables. These cases fall within the $l\lambda$ fragment of Miller (1990), which is a decidable extension of first-order unification.

The value of VAR will play the role of *restr-arg*, supplying an entity-type variable, and the value of RESTR will play the role of *restr* in the meaning constructor of the determiner. For a preliminary version of the lexical entry for 'every', we replace the relevant portions of our canonical determiner entry appropriately:

(21) Preliminary lexical entry for 'every':

$$\begin{array}{lll} \text{every} & \text{Det} & (\uparrow \textsc{spec}) = \text{`}\textsc{every}\text{'} \\ & & \forall R, S.\ (\forall x.\ (\uparrow_\sigma \textsc{var}) \rightsquigarrow x \multimap (\uparrow_\sigma \textsc{restr}) \rightsquigarrow R(x)) \\ & & \qquad \otimes (\forall x.\ \mathit{scope\text{-}arg} \rightsquigarrow x \multimap \mathit{scope} \rightsquigarrow S(x)) \\ & & \quad \multimap \mathit{scope} \rightsquigarrow \mathit{every}(R, S) \end{array}$$

The restriction property R should of course be derived from the semantic contribution of the nominal part of the noun phrase. Therefore, semantic constructors for nouns must connect appropriately to the VAR and RESTR components of the noun phrase's semantic projection, as we shall now see.

3.3 Noun meanings

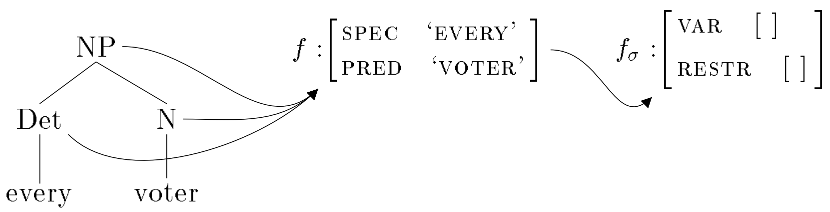
We will use the following phrase structure rule for simple noun phrases:

(22) $\text{NP} \longrightarrow \begin{array}{cc} \text{Det} & \text{N} \\ \uparrow = \downarrow & \uparrow = \downarrow \end{array}$

This rule states that the determiner Det and noun N contribute equally to the f-structure for the NP. Lexical specifications ensure that the noun contributes the PRED attribute and its value, and the determiner contributes the SPEC attribute and its value.

The c-structure, f-structure, and semantic structure for 'every voter', together with the functional relations between them, are:

(23)



In rule (22), the meaning constructors of the noun 'voter' and the determiner 'every' make reference to the same semantic structure, and in particular to the same semantic projections VAR and RESTR. The noun will supply appropriate values for the VAR and RESTR attributes of the NP, and these will be consumed by the determiner's meaning constructor. Thus, the semantic constructor for a noun will have the general form

$$\forall x.\ (\uparrow_\sigma \textsc{var}) \rightsquigarrow x \multimap (\uparrow_\sigma \textsc{restr}) \rightsquigarrow Px$$

where P is the meaning of the noun.[8] In particular, the lexical entry for 'voter' is:

(24) voter N $(\uparrow \text{PRED}) = \text{'VOTER'}$
$\forall X.\ (\uparrow_\sigma \text{VAR}) \rightsquigarrow X \multimap (\uparrow_\sigma \text{RESTR}) \rightsquigarrow \mathit{voter}(X)$

Given this entry and the one for 'every' in (21), we obtain the following instantiated semantic constructors for (23):

$$\begin{array}{ll} \textbf{every}: & \forall R, S.\ (\forall x.\ (f_\sigma\ \text{VAR}) \rightsquigarrow x \multimap (f_\sigma\ \text{RESTR}) \rightsquigarrow R(x)) \\ & \quad \otimes (\forall x.\ \mathit{scope\text{-}arg} \rightsquigarrow x \multimap \mathit{scope} \rightsquigarrow S(x)) \\ & \quad \multimap \mathit{scope} \rightsquigarrow \mathit{every}(R, S) \\ \textbf{voter}: & \forall X.\ (f_\sigma\ \text{VAR}) \rightsquigarrow X \multimap (f_\sigma\ \text{RESTR}) \rightsquigarrow \mathit{voter}(X) \end{array}$$

Applying the variable substitutions $X \mapsto x, R \mapsto \mathit{voter}$ and modus ponens to those two premises, we obtain the semantic constructor for 'every voter':

(25) **every-voter**: $\forall S.\ (\forall x.\ \mathit{scope\text{-}arg} \rightsquigarrow x \multimap \mathit{scope} \rightsquigarrow S(x))$
$\multimap \mathit{scope} \rightsquigarrow \mathit{every}(\mathit{voter}, S)$

In keeping with the parallel noted earlier between our semantic constructors and compositional types, the propositional part of this formula corresponds to the standard type for NP meanings, $(e \rightarrow t) \rightarrow t$.

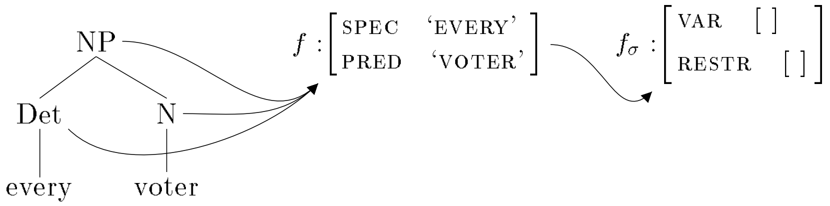
3.4 Quantified NP meanings

To complete our analysis of the semantic contribution of determiners, we need to characterize how a quantified NP contributes to the semantics of a sentence in which it appears, by specifying the semantic projections *scope-arg* and *scope* in quantified NP semantic constructors like (25).

Individual-type contribution First, we require the meaning of the scope to depend on the meaning of (the position filled by) the quantifier itself. Thus, *scope-arg* is the semantic projection for the quantified NP itself:

[8] Of course, the derivation would be more complicated if the NP included adjective phrases or other noun modifiers; for the sake of brevity, we will not discuss the contribution of noun modifiers in this paper. Intuitively, the function of modifiers is to consume the meaning of the phrase they modify and produce a new, modified meaning of the same semantic shape, which can play the same semantic role as the unmodified phrase can play. Dalrymple, Lamping, and Saraswat (1993) provide a general discussion of modification in this framework.

(26)



$$\textbf{every-voter}\text{: } \forall S.\ (\forall x.\ f_\sigma \rightsquigarrow x \multimap \mathit{scope} \rightsquigarrow S(x)) \multimap \mathit{scope} \rightsquigarrow \mathit{every}(\mathit{voter}, S)$$

Informally, the constructor for 'every voter' can be read as follows: if by giving the arbitrary meaning x of type e to f, the f-structure for 'every voter', we can derive the meaning $S(x)$ of type t for the scope of quantification *scope*, then S can be the property that the quantifier requires as its scope, yielding the meaning $\mathit{every}(\mathit{voter}, S)$ for *scope*. The quantified NP can thus be seen as providing two contributions to an interpretation: locally, a *referential import* x, which must be discharged when the scope of quantification is established; and globally, a *quantificational import* of type $(e \to t) \to t$, which is applied to the meaning of the scope of quantification to obtain a quantified proposition.

Notice also that the assignment of a meaning to *scope* appears on both sides of the implication, and that in fact the meaning is not the same in the two instances. Linear logic allows for the *consumption* of the preliminary meaning in the antecedent of the implication, *producing* the final meaning for *scope* in the conclusion.

Scope of quantification To complete our account of quantified NP interpretation, we need to explain how to select the possible scopes of quantification, for which we used the place-holder *scope* in (26).

As is well known, the scope of a quantifier is not syntactically fixed. While syntactic effects may play a significant role in scope preferences, most claims of scope islands (eg. May's (1985)) are defeasible given appropriate choices of lexical items and context. Therefore, we will take as possible quantifier scopes all semantic projections for which a meaning of proposition type can be derived. But even this liberal notion of scope is subject to indirect constraints from syntax, as those that we will see arise from interaction of coreference relations and quantification.

Previous work on scope determination in LFG (Halvorsen and Kaplan, 1988) defined possible scopes at the f-structure level, using *inside-out functional uncertainty* to nondeterministically choose a scope f-structure for quantified noun phrases. That approach requires the scope of a quantified NP to be an f-structure which contains the NP f-structure. In contrast, our approach depends only on the logical form of semantic constructors to yield just the appropriate scope choices. Within the constraints imposed by that logical form, the actual scope can be freely chosen. Logically, that means that the semantic constructor for an NP should quantify universally over scopes, as follows:

(27) $\textbf{every-voter}\text{: } \forall H, S.\ (\forall x.\ f_\sigma \rightsquigarrow x \multimap H \rightsquigarrow S(x)) \multimap H \rightsquigarrow \mathit{every}(\mathit{voter}, S)$

The foregoing argument leads to the following general semantic constructor for a determiner with meaning Q:

(28) $\forall H, R, S.$

$$
\begin{array}{ll}
(\forall x.\ (\uparrow_\sigma \textsc{var}) \rightsquigarrow x & \left\{\begin{array}{l}\text{if, by assuming an arbitrary meaning } x \\ \text{for } (\uparrow_\sigma \textsc{var}),\end{array}\right. \\
\quad \multimap (\uparrow_\sigma \textsc{restr}) \rightsquigarrow R(x)) & \left\{\begin{array}{l}\text{a meaning } R(x) \text{ for } (\uparrow_\sigma \textsc{restr}) \text{ can be} \\ \text{derived,}\end{array}\right. \\
(\forall x.\ \uparrow_\sigma \rightsquigarrow x & \left\{\begin{array}{l}\text{and if, by assuming an arbitrary mean-} \\ \text{ing } x \text{ for } \uparrow,\end{array}\right. \\
\quad \multimap H \rightsquigarrow_t S(x)) & \left\{\begin{array}{l}\text{a meaning } S(x) \text{ for some scope } H \text{ can} \\ \text{be derived,}\end{array}\right. \\
\quad \multimap H \rightsquigarrow_t Q(R, S) & \left\{\begin{array}{l}\text{then we can derive a possible complete} \\ \text{meaning for } H\end{array}\right.
\end{array}
$$

where H ranges over semantic structures associated with meanings of type t.

Note that the VAR and RESTR components of the semantic projection for a quantified NP in our analysis play a similar role to the // category constructor in PTQ (Montague, 1974), that of distinguishing syntactic configurations with identical semantic types but different contributions to the interpretation. The two PTQ syntactic categories t/e for intransitive verb phrases and $t/\!/e$ for common noun phrases correspond to the single semantic type $e \to t$; similarly, the two conjuncts in the antecedent of (28) correspond to the same semantic type, encoded with a linear implication, but to two different syntactic contexts, one relating the predication of an NP to its implicit argument and one relating a clause to an embedded argument.

3.5 Simple example of quantification

Before we look at quantifier scope ambiguity and interactions between scope and bound anaphora, we demonstrate the basic operation of our proposed meaning constructor for quantified NPs with a singly quantified, unambiguous sentence:

(29) Bill convinced every voter.

To carry out the analysis, we need a lexical entry for 'convinced':

(30) convinced V $(\uparrow \textsc{pred}) = \text{‘}\textsc{convince}\text{’}$
$\forall X, Y.\ (\uparrow \textsc{subj})_\sigma \rightsquigarrow X \otimes (\uparrow \textsc{obj})_\sigma \rightsquigarrow Y \multimap \uparrow_\sigma \rightsquigarrow \mathit{convince}(X, Y)$

The f-structure for (29) is:

(31)
$$
f{:}\begin{bmatrix}
\textsc{pred} & \text{‘}\textsc{convince}\text{’} \\
\textsc{subj} & g{:}\begin{bmatrix}\textsc{pred} & \text{‘}\textsc{Bill}\text{’}\end{bmatrix} \\
\textsc{obj} & h{:}\begin{bmatrix}\textsc{spec} & \text{‘}\textsc{every}\text{’} \\ \textsc{pred} & \text{‘}\textsc{voter}\text{’}\end{bmatrix}
\end{bmatrix}
$$

The premises for the derivation are appropriately instantiated meaning constructors for 'Bill' and 'convinced' together with the instantiated meaning constructor derived earlier for the quantified NP 'every voter':

$$\begin{array}{ll}
\textbf{bill}: & g_\sigma \rightsquigarrow \mathit{Bill} \\
\textbf{convinced}: & \forall X, Y.\ g_\sigma \rightsquigarrow X \otimes h_\sigma \rightsquigarrow Y \multimap f_\sigma \rightsquigarrow \mathit{convince}(X, Y) \\
\textbf{every-voter}: & \forall H, S.\ (\forall x.\ h_\sigma \rightsquigarrow x \multimap H \rightsquigarrow_t S(x)) \\
& \quad \multimap H \rightsquigarrow_t \mathit{every}(\mathit{voter}, S)
\end{array}$$

Giving the name **bill-convinced** to the formula

$$\textbf{bill-convinced}:\ \forall Y.\ h_\sigma \rightsquigarrow Y \multimap f_\sigma \rightsquigarrow \mathit{convince}(\mathit{Bill}, Y)$$

we have the derivation:

$$\begin{array}{lll}
& \textbf{bill} \otimes \textbf{convinced} \otimes \textbf{every-voter} & \text{(Premises.)} \\
\vdash & \textbf{bill-convinced} \otimes \textbf{every-voter} & X \mapsto \mathit{Bill} \\
\vdash & f_\sigma \rightsquigarrow \mathit{every}(\mathit{voter}, \lambda z.\mathit{convince}(\mathit{Bill}, z)) & H \mapsto f_\sigma, Y \mapsto x \\
& & S \mapsto \lambda z.\mathit{convince}(\mathit{Bill}, z)
\end{array}$$

No derivation of a different formula $f_\sigma \rightsquigarrow_t P$ is possible. The formula **bill-convinced** represents the semantics of the scope of the determiner 'every'. The derivable formula

$$\forall Y.\ h_\sigma \rightsquigarrow_e Y \multimap h_\sigma \rightsquigarrow_e Y$$

could at first sight be considered another possible, but erroneous, scope. However, the type subscripting of the $\rightsquigarrow$ relation used in the determiner lexical entry requires the scope to represent a dependency of a proposition on an individual, while this formula represents the dependency of an individual on an individual (itself). Therefore, it does not provide a valid scope for the quantifier.

3.6 Quantifier scope ambiguities

When a sentence contains more than one quantifier, scope ambiguities are of course possible. In our system, those ambiguities will appear as alternative successful derivations. We will take as our example this sentence:[9]

(32) Every candidate appointed a manager.

We need the following additional lexical entries:

(33) a Det $(\uparrow \text{SPEC}) = \text{'A'}$

$$\begin{array}{l}
\forall H, R, S.\ (\forall x.\ (\uparrow_\sigma \text{VAR}) \rightsquigarrow x \multimap (\uparrow_\sigma \text{RESTR}) \rightsquigarrow R(x)) \\
\qquad \otimes (\forall x.\ \uparrow_\sigma \rightsquigarrow x \multimap H \rightsquigarrow S(x)) \\
\quad \multimap H \rightsquigarrow a(R, S)
\end{array}$$

[9] To allow for apparent scope ambiguities, we adopt a scoping analysis of indefinites, as proposed, for example, by Neale (1990).

(34) candidate N $(\uparrow \text{PRED}) = \text{`CANDIDATE'}$
$\forall X.\ (\uparrow_\sigma \text{VAR}) \rightsquigarrow X \multimap (\uparrow_\sigma \text{RESTR}) \rightsquigarrow \mathit{candidate}(X)$

(35) manager N $(\uparrow \text{PRED}) = \text{`MANAGER'}$
$\forall X.\ (\uparrow_\sigma \text{VAR}) \rightsquigarrow X \multimap (\uparrow_\sigma \text{RESTR}) \rightsquigarrow \mathit{manager}(X)$

The f-structure for sentence (32) is:

(36)
$$f\!:\left[\begin{array}{ll} \text{PRED} & \text{`APPOINT'} \\ \text{SUBJ} & g\!:\left[\begin{array}{ll} \text{SPEC} & \text{`EVERY'} \\ \text{PRED} & \text{`CANDIDATE'} \end{array}\right] \\ \text{OBJ} & h\!:\left[\begin{array}{ll} \text{SPEC} & \text{`A'} \\ \text{PRED} & \text{`MANAGER'} \end{array}\right] \end{array}\right]$$

We can derive meaning constructors for 'every candidate' and 'a manager' in the way shown in Section 3.4. Further derivations proceed from those contributions together with the contribution of 'appointed':

$$\begin{array}{ll} \textbf{every-candidate}: & \forall G, R.\ (\forall x.\ g_\sigma \rightsquigarrow x \multimap G \rightsquigarrow R(x)) \\ & \quad \multimap G \rightsquigarrow \mathit{every}(\mathit{candidate}, R) \\ \textbf{a-manager}: & \forall H, S.\ (\forall y.\ h_\sigma \rightsquigarrow y \multimap H \rightsquigarrow S(y)) \\ & \quad \multimap H \rightsquigarrow a(\mathit{manager}, S) \\ \textbf{appointed}: & \forall X, Y.\ g_\sigma \rightsquigarrow X \otimes h_\sigma \rightsquigarrow Y \multimap f_\sigma \rightsquigarrow \mathit{appoint}(X, Y) \end{array}$$

As of yet, we have not made any commitment about the scopes of the quantifiers; the scope and scope meaning variables in **every-candidate** and **a-manager** have not been instantiated. Scope ambiguities are manifested in two different ways in our system: through the choice of different semantic structures G and H, corresponding to different scopes for the quantified NPs, or through different relative orders of application for quantifiers that scope at the same point. For this example, the second case is relevant, and we must now make a choice to proceed. The two possible choices correspond to two equivalent rewritings of **appointed**:

$$\begin{array}{ll} \textbf{appointed}_1: & \forall X.\ g_\sigma \rightsquigarrow X \multimap (\forall Y.\ h_\sigma \rightsquigarrow Y \multimap f_\sigma \rightsquigarrow \mathit{appoint}(X, Y)) \\ \textbf{appointed}_2: & \forall Y.\ h_\sigma \rightsquigarrow Y \multimap (\forall X.\ g_\sigma \rightsquigarrow X \multimap f_\sigma \rightsquigarrow \mathit{appoint}(X, Y)) \end{array}$$

These two equivalent forms correspond to the two possible ways of "currying" a two-argument function $f : \alpha \times \beta \to \gamma$ as one-argument functions:

$$\lambda u.\lambda v.f(u, v) : \alpha \to (\beta \to \gamma)$$

$$\lambda v.\lambda u.f(u, v) : \beta \to (\alpha \to \gamma)$$

We select 'a manager' to take narrower scope by using the variable instantiations

$$H \mapsto f_\sigma, Y \mapsto y, S \mapsto \lambda v.appoint(X, v)$$

and transitivity of implication to combine **appointed**$_1$ with **a-manager** into:

$$\textbf{appointed-a-manager}: \ \forall X.\ g_\sigma \rightsquigarrow X \\ \multimap f_\sigma \rightsquigarrow_t a(manager, \lambda v.appoint(X, v))$$

We have thus the derivation

$$\begin{array}{ll} & \textbf{every-candidate} \otimes \textbf{appointed}_1 \otimes \textbf{a-manager} \\ \vdash & \textbf{every-candidate} \otimes \textbf{appointed-a-manager} \\ \vdash & f_\sigma \rightsquigarrow_t every(candidate, \lambda u.a(manager, \lambda v.appoint(u, v))) \end{array}$$

of the $\forall\exists$ reading of (32), where the last step uses the substitutions

$$G \mapsto f_\sigma, X \mapsto x, R \mapsto \lambda u.a(manager, \lambda v.appoint(u, v))$$

Alternatively, we could have chosen 'every candidate' to take narrow scope, by combining **appointed**$_2$ with **every-candidate** to produce:

$$\textbf{every-candidate-appointed}: \ \forall Y.\ h_\sigma \rightsquigarrow Y \\ \multimap f_\sigma \rightsquigarrow_t every(candidate, \lambda u.appoint(u, Y))$$

This gives the derivation

$$\begin{array}{ll} & \textbf{every-candidate} \otimes \textbf{appointed}_2 \otimes \textbf{a-manager} \\ \vdash & \textbf{every-candidate-appointed} \otimes \textbf{a-manager} \\ \vdash & f_\sigma \rightsquigarrow_t a(manager, \lambda v.every(candidate, \lambda u.appoint(u, v))) \end{array}$$

for the $\exists\forall$ reading. These are the only two possible outcomes of the derivation of a meaning for (32), as required.

3.7 Constraints on quantifier scoping

Sentence (37) contains two quantifiers and therefore might be expected to show a two-way ambiguity analogous to the one described in the previous section:

(37) Every candidate appointed an admirer of his.

However, no such ambiguity is found if the pronoun 'his' is taken to corefer with the subject 'every candidate'. In this case, only one reading is available, in which 'an admirer of his' takes narrow scope. Intuitively, this NP may not take wider scope than the quantifier 'every candidate', on which its restriction depends.

As we will soon see, the lack of a wide scope 'a' reading follows automatically from our formulation of the meaning constructors for quantifiers and anaphors without further stipulation. In Pereira's earlier work on deductive interpretation (Pereira 1990, 1991), the same result was achieved through constraints on the relative scopes of glue-level universal quantifiers representing the dependencies between meanings of clauses and the meanings of their arguments. Here, although universal quantifiers are used to support the extraction of properties representing the meanings of the restriction and scope (the variables R and S in the semantic constructors for determiners), the blocking of the unwanted reading follows from the propositional structure of the glue formulas, specifically the nested linear implications. This is more satisfactory, since it does not reduce the problem of proper quantifier scoping in the object language to the same problem in the metalanguage.

The lexical entry for 'admirer' is:

(38) admirer N $(\uparrow \text{PRED}) = \text{`ADMIRER'}$
$$\forall X, Y.\ (\uparrow_\sigma \text{VAR}) \rightsquigarrow X \otimes (\uparrow \text{OBL}_{\text{OF}})_\sigma \rightsquigarrow Y$$
$$\multimap (\uparrow_\sigma \text{RESTR}) \rightsquigarrow \mathit{admirer}(X, Y)$$

Here, 'admirer' is a relational noun taking as its oblique argument a phrase with prepositional marker 'of', as indicated in the f-structure by the attribute OBL_{OF}. The meaning constructor for a relational noun has, as expected, the same propositional form as the binary relation type $e \times e \rightarrow t$: one argument is the admirer, and the other is the admiree.

We assume that the semantic projection for the antecedent of the pronoun 'his' has been determined by some separate mechanism and recorded as the ANT attribute of the pronoun's semantic projection.[10] The meaning constructor of the pronoun is, then, a formula that consumes the meaning of its antecedent and then reintroduces that meaning, simultaneously assigning it to its own semantic projection:

(39) his N $(\uparrow \text{PRED}) = \text{`PRO'}$
$$\forall X.\ (\uparrow_\sigma \text{ANT}) \rightsquigarrow X \multimap (\uparrow_\sigma \text{ANT}) \rightsquigarrow X \otimes \uparrow_\sigma \rightsquigarrow X$$

In other words, the semantic contribution of a pronoun copies the meaning X of its antecedent as the meaning of the pronoun itself. Since the left-hand side of the linear implication "consumes" the antecedent meaning, it must be reinstated in the consequent of the implication.

The f-structure for example (37) is:

[10] The determination of appropriate values for ANT requires a more detailed analysis of other linguistic constraints on anaphora resolution, which would need further projections to give information about, for example, discourse relations and salience. Dalrymple (1993) discusses in detail LFG analyses of anaphoric binding.

(40)
$$f{:}\left[\begin{array}{ll}
\text{PRED} & \text{`APPOINTED'} \\
\text{SUBJ} & g{:}\left[\begin{array}{ll}\text{SPEC} & \text{`EVERY'} \\ \text{PRED} & \text{`CANDIDATE'}\end{array}\right] \\
\text{OBJ} & h{:}\left[\begin{array}{ll}\text{SPEC} & \text{`A'} \\ \text{PRED} & \text{`ADMIRER'} \\ \text{OBL}_{\text{OF}} & i{:}\left[\begin{array}{ll}\text{PRED} & \text{`PRO'}\end{array}\right]\end{array}\right]
\end{array}\right]$$

with $(i_\sigma\ \text{ANT}) = g_\sigma$.

We will begin by illustrating the derivation of the meaning of 'an admirer of his', starting from the following premises:

$$\begin{array}{ll}
\textbf{a}: & \forall H, R, S.\ (\forall x.\ (h_\sigma\ \text{VAR}) \rightsquigarrow x \multimap (h_\sigma\ \text{RESTR}) \rightsquigarrow R(x)) \\
& \quad \otimes (\forall x.\ h_\sigma \rightsquigarrow x \multimap H \rightsquigarrow S(x)) \\
& \quad \multimap H \rightsquigarrow a(R, S) \\
\textbf{admirer}: & \forall Z, X.\ (h_\sigma\ \text{VAR}) \rightsquigarrow Z \otimes i_\sigma \rightsquigarrow X \\
& \quad \multimap (h_\sigma\ \text{RESTR}) \rightsquigarrow \mathit{admirer}(Z, X) \\
\textbf{his}: & \forall X.\ g_\sigma \rightsquigarrow X \multimap g_\sigma \rightsquigarrow X \otimes i_\sigma \rightsquigarrow X
\end{array}$$

First, we rewrite **admirer** into the equivalent form

$$\forall X.\ i_\sigma \rightsquigarrow X \multimap (\forall Z.\ (h_\sigma\ \text{VAR}) \rightsquigarrow Z \multimap (h_\sigma\ \text{RESTR}) \rightsquigarrow \mathit{admirer}(Z, X))$$

We can use this formula to rewrite the second conjunct in the consequent of **his**, yielding

$$\begin{array}{l}
\textbf{admirer-of-his}: \\
\quad \forall X.\ g_\sigma \rightsquigarrow X \multimap \\
\quad\quad g_\sigma \rightsquigarrow X \otimes \\
\quad\quad (\forall Z.\ (h_\sigma\ \text{VAR}) \rightsquigarrow Z \multimap (h_\sigma\ \text{RESTR}) \rightsquigarrow \mathit{admirer}(Z, X))
\end{array}$$

In turn, the second conjunct in the consequent of **admirer-of-his** matches the first conjunct in the antecedent of **a** given appropriate variable substitutions, allowing us to derive

$$\begin{array}{l}
\textbf{an-admirer-of-his}: \\
\quad \forall X.\ g_\sigma \rightsquigarrow X \multimap \\
\quad\quad g_\sigma \rightsquigarrow X \otimes (\forall H, S.\ (\forall x.\ h_\sigma \rightsquigarrow x \multimap H \rightsquigarrow S(x)) \multimap \\
\quad\quad\quad H \rightsquigarrow a(\lambda z.\mathit{admirer}(z, X), S))
\end{array}$$

At this point the other formulas available are:

$$\begin{array}{l}
\textbf{every-candidate}: \\
\quad \forall H, S.\ (\forall x.\ g_\sigma \rightsquigarrow x \multimap H \rightsquigarrow S(x)) \\
\quad \multimap H \rightsquigarrow \mathit{every}(\mathit{candidate}, S) \\
\textbf{appointed}: \\
\quad \forall Z, Y.\ g_\sigma \rightsquigarrow Z \otimes h_\sigma \rightsquigarrow Y \multimap f_\sigma \rightsquigarrow \mathit{appoint}(Z, Y)
\end{array}$$

We have thus the meanings of the two quantified NPs. The antecedent implication of **every-candidate** has an atomic conclusion and hence cannot be satisfied by **an-admirer-of-his**, which has a conjunctive conclusion. Therefore, the only possible move is to combine **appointed** and **an-admirer-of-his**. We do this by first putting **appointed** in the equivalent form

$$\forall Z.\ g_\sigma \rightsquigarrow Z \multimap (\forall Y.\ h_\sigma \rightsquigarrow Y \multimap f_\sigma \rightsquigarrow appoint(Z, Y))$$

After substituting X for Z, this can be used to rewrite the first conjunct in the consequent of **an-admirer-of-his** to derive

$$\begin{array}{l} \forall X.\ g_\sigma \rightsquigarrow X \multimap \\ \quad (\forall Y.\ h_\sigma \rightsquigarrow Y \multimap f_\sigma \rightsquigarrow appoint(X, Y)) \otimes \\ \quad (\forall H, S.\ (\forall x.\ h_\sigma \rightsquigarrow x \multimap H \rightsquigarrow S) \multimap H \rightsquigarrow a(\lambda z.admirer(z, X), S)) \end{array}$$

Applying the substitutions

$$Y \mapsto x, H \mapsto f_\sigma, S \mapsto \lambda z.appoint(X, z)$$

and modus ponens with the two conjuncts in the consequent as premises, we obtain

$$\forall X.\ g_\sigma \rightsquigarrow X \multimap f_\sigma \rightsquigarrow_t a(\lambda z.admirer(z, X), \lambda z.appoint(X, z))$$

Finally, this formula can be combined with **every-candidate** to give the meaning of the whole sentence:

$$f_\sigma \rightsquigarrow_t every(candidate, \lambda w.a(\lambda z.admirer(z, w), \lambda z.appoint(w, z)))$$

In fact, this is the only derivable conclusion, showing that our analysis blocks those putative scopings in which variables occur outside the scope of their binders.

3.8 Adequacy

We will now argue that our analysis is *sound* in that all variables occur in the scope of their binders, and *complete* in that all possible sound readings can be generated.

More precisely, soundness requires that all occurrences of a meaning-level variable x representing the argument positions filled by a quantified NP or anaphors bound to the NP are within the scope of the quantifier meaning of the NP binding x. As argued by Pereira (1990), treatments of quantification based on storage or quantifier raising either fail to guarantee soundness or enforce it by stipulation. In contrast, deductive frameworks based on a suitable type logic for meanings, such as those arising from categorial semantics, achieve soundness as a by-product of the soundness of their underlying type logics.

In the present setting, meaning terms are explicitly constructed rather than read out from well-typing proofs using the Curry-Howard connection between proofs and terms, but the particular form of our glue-logic formulas follows that of typing rules closely and thus guarantees soundness, as we will now explain.

Recall first that quantifiers can only be introduced into meaning terms by quantified NPs with semantic contributions of the form

(41) $\forall H, S.\ (\forall x.\ f \rightsquigarrow x \multimap H \rightsquigarrow S(x)) \multimap H \rightsquigarrow Q(S)$

where f is the semantic projection of the NP and Q is the meaning of the NP. Since S outscopes x, any instance of S in a valid derivation will be a meaning term of the form $\lambda z.T$, with x not free in T. The free occurrences of z in T will be precisely the positions quantified over by Q. We need thus to show that f and all semantic projections coreferential f have z as its interpretation. But f itself is given interpretation x in (41), while coreferential projections must by lexical entry (39) also have interpretation x. Since $S(x) = T[z \mapsto x]$ with x not free in T, any free occurrence of x in $S(x)$ must arise from substituting x for z in T. That is, the interpretation of f and those of any other projections which corefer with f are quantified over by Q as required.

As seen in the above argument, the dependency of anaphors on their antecedents is encoded by the propositional structure and quantification over semantic projections of the semantic contributions of anaphors. That encoding alone is sufficient to generate all and only the possible derivations, but quantification over meaning terms is needed to extract the appropriate meaning terms from the derivations. The scope of glue language variables ranging over meaning terms guarantees that all variables in meaning terms are properly bound.

Turning now to completeness, we need to consider the correlations between the deductive patterns and the functional structure. With one exception, the glue-logic formulas from which deduction starts respect the functional structure of meanings in that implications that conclude the meaning of a phrase depend on the meanings of all immediate subphrases which can have meanings, or depend on the phrase itself, but on nothing else. The exception is anaphors, whose meanings depend on that of their antecedents. Thus, the meaning of a phrase will, transitively, depend on the meanings of all its subphrases that can have meanings and on the meanings of the antecedents of its anaphoric pronouns.

Now we can consider the possible scopings of a quantified NP in terms of phrase structure. The linearity of the implication in the antecedent of the NP's constructor requires the meaning of the scope to depend on the meaning of the noun phrase and that nothing else depend on that meaning. But the above argument shows that this will be true exactly of every containing phrase, unless there is a bound anaphor not contained in the containing phrase that has the NP as its antecedent. So all the containing phrases that also contain all coreferring anaphors are, indeed, candidates for scope of the quantified NP.

It is worth noting that the quantificational structure of semantic constructors is enough on its own to ensure soundness of the resulting meaning terms. In particular, the nested implication form of quantified NP constructors could be replaced by the flatter

$$\exists x . g_\sigma \rightsquigarrow x \otimes (\forall H, S.\ H \rightsquigarrow S(x) \multimap H \rightsquigarrow Q(S))$$

A quantifier lexical entry would then look like:

$$\begin{array}{l} \exists x.\ (\uparrow_\sigma \text{VAR}) \rightsquigarrow x \\ \quad \otimes \forall H, R.\ (\uparrow_\sigma \text{RESTR}) \rightsquigarrow R(x) \multimap (\uparrow_\sigma \rightsquigarrow x \otimes \forall S.\ (H \rightsquigarrow S(x)) \multimap H \rightsquigarrow Q(z, Rz, Sz)) \end{array}$$

This formulation just asserts that there is a generic entity, x, which stands for the meaning of the quantified phrase, and also serves as the argument of the restriction. The derivations of the restriction and scope are then expected to consume this information. By avoiding nested implications, this formulation may be computationally more desirable.

However, the logical structure of this formulation is not as restrictive as that of (28), as it can allow additional derivations where information intended for the restriction can be used by the scope. This cannot happen in our analyses, however, since all the dependencies in semantic constructors respect syntactic dependencies expressed in the f-structure. As long as that principle is observed, the formulation above is equivalent to (28). Despite this, we prefer to stay closer to categorial semantics and thus capture explicitly quantifier and anaphoric dependencies in the propositional structure. We will therefore continue with the formulation (28).

4 Intensional Verbs

Following Montague (1974),we will give an intensional verb like *seek* a meaning that takes as direct object an NP meaning intension. Montague's method for assembling meanings by function application forces the meanings of all expressions of a given syntactic category to be raised to their lowest common semantic type. In particular, every transitive verb meaning, whether intensional or not, must take a quantified NP meaning intension as direct-object argument. In contrast, our approach allows the semantic contributions of verbs to be of as low a type as possible. Nonetheless, the uniformity of the translation process is preserved because any required type changes are derivable within the glue language, along lines similar to type change in the undirected Lambek calculus (van Benthem, 1988).

We will not represent intensional types explicitly at the glue level, in contrast to categorial treatments of intensionality such as Morrill's (1990; 1993). Instead, semantic constructors will correspond to the appropriate extensional types. The Montagovian intension and extension operators $\hat{}$ and $\check{}$ will appear only at term level, to the right of $\rightsquigarrow$ in our derivation formulas. Thus, while the meaning of *seek* has type $e \to (s \to ((e \to t) \to t)) \to t$, the corresponding semantic constructor in (43) parallels the type $e \to ((e \to t) \to t) \to t$.

Our implicit treatment of intensional types imposes certain constraints on the use of functional abstraction and application in meaning terms, since β-reduction is only valid for intensional terms if the argument is *intensionally closed*, that is, if the free occurrences of the bound variable do not occur in intensional contexts (Gamut, 1991, p. 131). As we will see, that constraint is verified by all the semantic terms in our semantic constructors. Thus, in carrying out proofs we will be justified in solving for free variables in meaning terms modulo the $\check{}\hat{}$-elimination schema $\check{}(\hat{}P) = P$ and α-, β- and η-conversion.

Generalized quantifier meanings in Montague grammar are given type $(s \to e \to t) \to (s \to e \to t) \to t$, that is, their are relations between properties. While we maintain the propositional form of glue-level formulas corresponding to the extensional generalized quantifier meanings discussed earlier, the semantic terms in determiner semantic constructors must be adapted to match the new intensionalized generalized quantifier type:

$$\begin{aligned}\forall H, R, S.\ & (\forall x.\ (\uparrow_\sigma \text{VAR}) \rightsquigarrow x \multimap (\uparrow_\sigma \text{RESTR}) \rightsquigarrow ({}^\vee R)(x))\\ & \otimes (\forall x.\ \uparrow_\sigma \rightsquigarrow x \multimap H \rightsquigarrow ({}^\vee S)(x))\\ & \multimap H \rightsquigarrow a(R, S)\end{aligned}$$

Therefore, the meaning of a sentence such as (37) will now be written:

$$every({}^\wedge candidate, {}^\wedge \lambda w.a({}^\wedge \lambda z.admirer(z, w), {}^\wedge \lambda z.appoint(w, z)))$$

The type-changing potential of the linear-logic formulation allows us to give an intensional verb a single semantic constructor, and yet have the expected *de re/de dicto* ambiguities follow without further stipulation. For example, we will see that for sentence

(42) Bill seeks a unicorn.

we can derive the two readings:

de dicto reading:	$seek(Bill, {}^\wedge \lambda Q.a({}^\wedge unicorn, Q))$
de re reading:	$a({}^\wedge unicorn, \lambda u.seek(Bill, {}^\wedge \lambda Q.({}^\vee Q)(u)))$

Given the foregoing analysis, the lexical entry for *seek* is:

(43) seek $(\uparrow \text{PRED}) = \text{‘SEEK’}$

$$\begin{aligned}\forall Z, Y.\ & (\uparrow \text{SUBJ})_\sigma \rightsquigarrow Z\\ & \otimes(\forall s, p.\ (\forall X.\ (\uparrow \text{OBJ})_\sigma \rightsquigarrow X \multimap s \rightsquigarrow ({}^\vee p)(X)) \multimap s \rightsquigarrow Y(p))\\ & \multimap \uparrow_\sigma \rightsquigarrow seek(Z, {}^\wedge Y)\end{aligned}$$

which can be paraphrased as follows:

$\forall Z, Y.\ (\uparrow \text{SUBJ})_\sigma \rightsquigarrow Z \otimes$	The verb *seek* requires a meaning Z for its subject and	
$(\forall s, p.\ (\forall X.\ (\uparrow \text{OBJ})_\sigma \rightsquigarrow X \multimap s \rightsquigarrow ({}^\vee p)(X)) \multimap s \rightsquigarrow Y(p))$	a meaning ${}^\wedge Y$ for its object, where Y is an NP meaning applied to the meaning p of an arbitrarily-chosen ‘scope’ s,	$(*)$
$\multimap \uparrow_\sigma \rightsquigarrow seek(Z, {}^\wedge Y)$	to produce the clause meaning $seek(Z, {}^\wedge Y)$.	

Rather than looking for an entity type meaning for its object, the requirement expressed by the subformula labeled $(*)$ describes semantic constructors of quantified NPs. Such a constructor takes as input the constructor for a scope, which by itself maps an arbitrary meaning X to the meaning $p(X)$ for an arbitrary scope s. From that input, the quantified NP constructor will produce a final quantified meaning M for s. That meaning is required to satisfy the equation $M = Y(p)$, and thus ${}^\wedge Y$ is the property of properties (predicate intensions) that *seek* requires as second argument. Note that the argument p of Y in the equation will be an intension given the new semantic constructors for determiners. Therefore, β-conversion with the abstraction Y as functor is allowed.

The f-structure for (42) is:

(44)
$$f\!:\!\begin{bmatrix} \textsc{pred} & \text{`SEEK'} \\ \textsc{subj} & g\!:\!\begin{bmatrix} \textsc{pred} & \text{`Bill'} \end{bmatrix} \\ \textsc{obj} & h\!:\!\begin{bmatrix} \textsc{spec} & \text{`A'} \\ \textsc{pred} & \text{`UNICORN'} \end{bmatrix} \end{bmatrix}$$

The semantic constructors associated with this f-structure are then:

$$\begin{array}{ll} \textbf{seeks:} & \forall Z, Y.\quad g_\sigma \rightsquigarrow Z \\ & \qquad\qquad \otimes(\forall s, p.(\forall X.\, h_\sigma \rightsquigarrow X \multimap s \rightsquigarrow ({}^{\vee}p)(X)) \multimap s \rightsquigarrow Y({}^{\wedge}p)) \\ & \quad \multimap f_\sigma \rightsquigarrow seek(Z, {}^{\wedge}Y) \\ \textbf{Bill:} & g_\sigma \rightsquigarrow Bill \\ \textbf{a-unicorn:} & \forall H, S.\, (\forall x.\, h_\sigma \rightsquigarrow x \multimap H \rightsquigarrow ({}^{\vee}S)(x)) \multimap H \rightsquigarrow a({}^{\wedge}unicorn, S) \end{array}$$

These are the premises for the deduction of the meaning of sentence (42). From the premises **Bill** and **seeks** and the instantiation $Z \mapsto Bill$ we can conclude by modus ponens:

$$\begin{array}{ll} \textbf{Bill-seeks:} & \forall Y.\, (\forall s, p.\, (\forall X.\, h_\sigma \rightsquigarrow X \multimap s \rightsquigarrow ({}^{\vee}p)(X)) \multimap s \rightsquigarrow Y(p)) \\ & \qquad \multimap f_\sigma \rightsquigarrow seek(Bill, {}^{\wedge}Y) \end{array}$$

Different derivations starting from the premises **Bill-seeks** and **a-unicorn** will yield the alternative readings of *Bill seeks a unicorn*, as we shall now see.

4.1 De Dicto Reading

The formula **a-unicorn** is exactly what is required by the antecedent of **Bill-seeks** provided that the following substitutions are performed:

$$\begin{aligned} H &\mapsto s \\ S &\mapsto p \\ X &\mapsto x \\ Y &\mapsto \lambda P.a({}^{\wedge}unicorn, P) \end{aligned}$$

We can thus conclude the desired *de dicto* reading:

$$f_\sigma \rightsquigarrow seek(Bill, {}^{\wedge}\lambda P.a({}^{\wedge}unicorn, P))$$

To show how the premises also support a *de re* reading, we consider first the simpler case of nonquantified direct objects.

$$\cfrac{\cfrac{\cfrac{\cfrac{h_\sigma \rightsquigarrow Al \vdash h_\sigma \rightsquigarrow Al \qquad s \rightsquigarrow (\check{\ }P)(Al) \vdash s \rightsquigarrow (\check{\ }P)(Al)}{h_\sigma \rightsquigarrow Al, h_\sigma \rightsquigarrow Al \multimap s \rightsquigarrow (\check{\ }P)(Al) \vdash s \rightsquigarrow (\check{\ }P)(Al)}}{h_\sigma \rightsquigarrow Al, (\forall x. h_\sigma \rightsquigarrow x \multimap s \rightsquigarrow (\check{\ }P)(x)) \vdash s \rightsquigarrow (\check{\ }P)(Al)}}{h_\sigma \rightsquigarrow Al \vdash (\forall x. h_\sigma \rightsquigarrow x \multimap s \rightsquigarrow (\check{\ }P)(x)) \multimap s \rightsquigarrow (\check{\ }P)(Al)}}{h_\sigma \rightsquigarrow Al \vdash \forall P. (\forall x. h_\sigma \rightsquigarrow x \multimap s \rightsquigarrow (\check{\ }P)(x)) \multimap s \rightsquigarrow (\check{\ }P)(Al)}$$

Figure 1: Proof that **Al** can function as a quantifier

4.2 Nonquantified Objects

The meaning constructor for *seek* also allows for nonquantified objects as arguments, without needing a special type-raising rule. Consider the f-structure for the sentence *Bill seeks Al*:

(45)
$$f{:}\left[\begin{array}{ll} \text{PRED} & \text{`SEEK'} \\ \text{SUBJ} & g{:}\left[\begin{array}{ll}\text{PRED} & \text{`BILL'}\end{array}\right] \\ \text{OBJ} & h{:}\left[\begin{array}{ll}\text{PRED} & \text{`AL'}\end{array}\right] \end{array}\right]$$

The lexical entry for *Al* is analogous to the one for *Bill*. We begin with the premises **Bill-seeks** and **Al**:

$$\begin{array}{ll} \textbf{Bill-seeks:} & \forall Y. (\forall s, p. (\forall X.\ h_\sigma \rightsquigarrow X \multimap s \rightsquigarrow (\check{\ }p)(X)) \multimap s \rightsquigarrow Y(p)) \\ & \quad \multimap f_\sigma \rightsquigarrow seek(Bill, \hat{\ }Y) \\ \textbf{Al:} & h_\sigma \rightsquigarrow Al \end{array}$$

For the derivation to proceed, **Al** must supply the NP meaning constructor that **Bill-seeks** requires. This is possible because **Al** can map a proof Π of the meaning for s from the meaning for h into a meaning for s, simply by supplying $h_\sigma \rightsquigarrow Al$ to Π. Formally, from **Al** we can prove (Figure 1):

(46) $\forall P.\ (\forall x.\ h_\sigma \rightsquigarrow x \multimap s \rightsquigarrow (\check{\ }P)(x)) \multimap s \rightsquigarrow (\check{\ }P)(Al)$

This corresponds to the Montagovian type-raising of a proper name meaning to an NP meaning, and also to the undirected Lambek calculus derivation of the sequent $e \Rightarrow (e \rightarrow t) \rightarrow t$.

Formula (46) with the substitutions

$$P \mapsto p, Y \mapsto \lambda P.(\check{\ }P)(Al)$$

can then be used to satisfy the antecedent of **Bill-seeks** to yield the desired result:

$$f_\sigma \rightsquigarrow seek(Bill, \hat{\ }\lambda P.(\check{\ }P)(Al))$$

It is worth contrasting the foregoing derivation with treatments of the same issue in a λ-calculus setting. The function $\lambda x.\lambda P.(\check{\ }P)(x)$ raises a term like Al to the quantified

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{I \rightsquigarrow Z \vdash I \rightsquigarrow Z \qquad S \rightsquigarrow ({}^{\vee}P)(Z) \vdash S \rightsquigarrow ({}^{\vee}P)(Z)}{I \rightsquigarrow Z, I \rightsquigarrow Z \multimap S \rightsquigarrow ({}^{\vee}P)(Z) \vdash S \rightsquigarrow ({}^{\vee}P)(Z)}}{I \rightsquigarrow Z, (\forall x.\, I \rightsquigarrow x \multimap S \rightsquigarrow ({}^{\vee}P)(x)) \vdash S \rightsquigarrow ({}^{\vee}P)(Z)}}{I \rightsquigarrow Z \vdash (\forall x.\, I \rightsquigarrow x \multimap S \rightsquigarrow ({}^{\vee}P)(x)) \multimap S \rightsquigarrow ({}^{\vee}P)(Z)}}{I \rightsquigarrow Z \vdash \forall S, P.\, (\forall x.\, I \rightsquigarrow x \multimap S \rightsquigarrow ({}^{\vee}P)(x)) \multimap S \rightsquigarrow ({}^{\vee}P)(Z)}}{\vdash I \rightsquigarrow Z \multimap \forall S, P.\, (\forall x.\, I \rightsquigarrow x \multimap S \rightsquigarrow ({}^{\vee}P)(x)) \multimap S \rightsquigarrow ({}^{\vee}P)(Z)}}{\vdash \forall I, Z.\, I \rightsquigarrow Z \multimap \forall S, P.\, (\forall x.\, I \rightsquigarrow x \multimap S \rightsquigarrow ({}^{\vee}P)(x)) \multimap S \rightsquigarrow ({}^{\vee}P)(Z)}$$

Figure 2: General Type-Raising Theorem

NP form $\lambda P.({}^{\vee}P)(Al)$, so it is easy to modify Al to make it suitable for **seek**. Because a λ-term must specify exactly how functions and arguments combine, the conversion must be explicitly applied somewhere, either in a meaning postulate or in an alternate definition for *seek*. Thus, it is impossible to write a function term that is indifferent with respect to whether its argument is Al or $\lambda P.({}^{\vee}P)(Al)$.

In our deductive framework, on the other hand, the exact way in which different propositions can interact is not prescribed, although it is constrained by their logical structure. Thus $h_\sigma \rightsquigarrow Al$ can function as any logical consequence of itself, in particular as:

$$\forall S, P.\, (\forall x.\, h_\sigma \rightsquigarrow x \multimap S \rightsquigarrow ({}^{\vee}P)(x)) \multimap S \rightsquigarrow ({}^{\vee}P)(Al)$$

This flexibility, which is also found in syntactic-semantic analyses based on the Lambek calculus and its variants (Moortgat, 1988; Moortgat, 1992b; van Benthem, 1991), seems to align well with some of the type flexibility in natural language.

4.3 Type Raising and Quantifying In

The derivation in Figure 1 can be generalized as shown in Figure 2 to prove the general type-raising theorem:

(47) $\forall I, Z.\, I \rightsquigarrow Z \multimap (\forall S, P.\, (\forall x. I \rightsquigarrow x \multimap S \rightsquigarrow ({}^{\vee}P)(x)) \multimap S \rightsquigarrow ({}^{\vee}P)(Z))$

This theorem can be used to raise meanings of e type to $(e \rightarrow t) \rightarrow t$ type, or, dually, to quantify into verb argument positions. For example, with the variable instantiations

$$\begin{aligned} I &\mapsto h_\sigma \\ X &\mapsto x \\ P &\mapsto p \\ S &\mapsto s \\ Y &\mapsto \lambda R.({}^{\vee}R)(Z) \end{aligned}$$

we can use transitivity of implication to combine (47) with **Bill-seeks** to derive:

$$\textbf{Bill-seeks}'{:}\, \forall Z.\, h_\sigma \rightsquigarrow Z \multimap f_\sigma \rightsquigarrow seek(Bill, {}^{\wedge}\lambda R.({}^{\vee}R)(Z))$$

This formula can then be combined with arguments of type e to produce a meaning for f_σ. For instance, it will take the non-type-raised $h_\sigma \rightsquigarrow Al$ to yield the same result

$$f_\sigma \rightsquigarrow seek(Bill, {}^\wedge\lambda R.({}^\vee R)(Al))$$

as the combination of **Bill-seeks** with the type-raised version of **Al**. In fact, **Bill-seeks′** corresponds to type $e \to t$, and can thus be used as the scope of a quantifier, which would then quantify into the intensional direct object argument of *seek*. As we will presently see, that is exactly what is needed to derive *de re* readings.

4.4 De Re Reading

We have just seen how theorem (47) provides a general mechanism for quantifying into intensional argument positions. In particular, it allowed the derivation of **Bill-seeks′** from **Bill-seeks**. Now, given the premises

Bill-seeks′: $\forall Z.\ h_\sigma \rightsquigarrow Z \multimap f_\sigma \rightsquigarrow seek(Bill, {}^\wedge\lambda R.({}^\vee R)(Z))$
a-unicorn: $\forall H, S.\ (\forall x.\ h_\sigma \rightsquigarrow x \multimap H \rightsquigarrow ({}^\vee S)(x)) \multimap H \rightsquigarrow a({}^\wedge unicorn, S)$

and the variable substitutions

$$\begin{aligned} Z &\mapsto x \\ H &\mapsto f_\sigma \\ S &\mapsto {}^\wedge\lambda z.seek(Bill, {}^\wedge\lambda R.({}^\vee R)(z)) \end{aligned}$$

we can apply modus ponens to derive the *de re* reading of *Bill seeks a unicorn*:

$$f_\sigma \rightsquigarrow a({}^\wedge unicorn, {}^\wedge\lambda z.seek(Bill, {}^\wedge\lambda R.({}^\vee R)(z)))$$

5 Comparison with Categorial Syntactic Approaches

In recent work, multidimensional and labeled deductive systems (Moortgat, 1992b; Morrill, 1993) have been proposed as refinements of the Lambek systems that are able to represent synchronized derivations involving multiple levels of representation: for instance, a level of head-dependent representations and a level of syntactic functor-argument representations. However, these systems do not seem yet able to represent the connection between a flat syntactic representation in terms of grammatical functions, such as the f-structure of LFG, and a function-argument semantic representation. The problem in those systems is that they cannot express at the type level the link between particular syntactic structures (f-structures in our case) and particular contributions to meaning. The extraction of meanings from derivations following the Curry-Howard isomorphism that is standard in categorial systems demands that the order of syntactic combination coincide with the order of semantic combination so that functor-argument relations at the syntactic and semantic level are properly aligned.

Nevertheless, there are strong similarities between the analysis of quantification that we present and analyses of the same phenomena discussed by Morrill (1993) and Carpenter (1993). Following Moortgat (1992a), they add to an appropriate version of the Lambek calculus (Lambek, 1958) the *scope* connective $\Uparrow$, subject to the following proof rules:

$$\frac{\Gamma, v : A, \Gamma' \Rightarrow u : B \qquad \Delta, t(\lambda v.u) : B, \Delta' \Rightarrow C}{\Delta, \Gamma, t : A \Uparrow B, \Gamma', \Delta' \Rightarrow C} \quad [\mathrm{QL}]$$

$$\frac{\Gamma \Rightarrow u : A}{\Gamma \Rightarrow \lambda v.v(u) : A \Uparrow B} \quad [\mathrm{QR}]$$

In terms of the scope connective, a quantified NP is given the category N $\Uparrow$ S, which semantically corresponds to the type $(e \to t) \to t$ and agrees with the propositional structure of our linear formulas for quantified NPs. A phrase of category $N \Uparrow S$ is an infix functor that binds a variable of type e, the type of individual NPs N, within a scope of type t, the type of sentences S. An intensional verb like 'seek' has, then, category (N \ (S)/(N $\Uparrow$ S), with corresponding type $((e \to t) \to t) \to e \to t$. [11] Thus the intensional verb will take as direct object a quantified NP, as required.

A problem arises, however, with sentences such as

(48) Bill seeks a conversation with every unicorn.

This sentence has five possible interpretations:

(49) a. $\mathit{seek}(\mathit{Bill}, \hat{}\lambda P.\mathit{every}(\hat{}\mathit{unicorn}, \hat{}\lambda u.a(\hat{}\lambda z.\mathit{conv\text{-}with}(z,u), P)))$

b. $\mathit{seek}(\mathit{Bill}, \hat{}\lambda P.a(\hat{}\lambda z.\mathit{every}(\hat{}\mathit{unicorn}, \hat{}\lambda u.\mathit{conv\text{-}with}(z,u)), P))$

c. $\mathit{every}(\hat{}\mathit{unicorn}, \hat{}\lambda u.\mathit{seek}(\mathit{Bill}, \hat{}\lambda P.a(\hat{}\lambda z.\mathit{conv\text{-}with}(z,u), P)))$

d. $\mathit{every}(\hat{}\mathit{unicorn}, \hat{}\lambda u.a(\hat{}\lambda z.\mathit{conv\text{-}with}(z,u), \hat{}\lambda z.\mathit{seek}(\mathit{Bill}, \hat{}\lambda P.(\check{}P)(z))))$

e. $a(\hat{}\lambda z.\mathit{every}(\hat{}\mathit{unicorn}, \hat{}\lambda u.\mathit{conv\text{-}with}(z,u)), \hat{}\lambda z.\mathit{seek}(\mathit{Bill}, \hat{}\lambda P.(\check{}P)(z)))$

Both our approach and the categorial analysis using the scope connective have no problem in deriving interpretations (49b), (49c), (49d) and (49e). In those cases, the scope of 'every unicorn' interpreted as an appropriate term of type $e \to t$. However, the situation is different for interpretation (49a), in which both the conversations and the unicorn are *de dicto*, but the conversations sought may be different for different unicorns sought. As we will show below, this interpretation can be easily derived within our framework. However, a similar derivation does not appear possible in terms of the categorial scoping connective.

The difficulty for the categorial account is that the category N $\Uparrow$ S represents a phrase that plays the role of a category N phrase where it appears, but takes an S (dependent

[11] These category and type assignments are an oversimplification since intensional verbs like *seek* require a direct object of type $s \to ((e \to t) \to t)$, but for the present discussion the simpler category and type are sufficient. Morrill (1993) provides a full treatment.

on the N) as its scope. In the derivation of (49a), however, the scope of 'every unicorn' is 'a conversation with', which is not of category S. Semantically, 'a conversation with' is represented by:

(50) $\lambda P.\hat{}\lambda u.a(\hat{}\lambda z.conv\text{-}with(z,u), P) : (s \to e \to t) \to (s \to e \to t)$

The *undirected* Lambek calculus (van Benthem, 1991) allows us to compose (50) with the interpretation of 'every unicorn':

(51) $\lambda Q.every(\hat{}unicorn, Q) : (s \to e \to t) \to t$

to yield:

(52) $\lambda P.every(\hat{}unicorn, \hat{}\lambda u.a(\hat{}\lambda z.conv\text{-}with(z,u), P)) : (s \to e \to t) \to t$

As we will see below, our linear logic formulation also allows that derivation step.

In contrast, as Moortgat (1992a) points out, the categorial rule [QR] is not powerful enough to raise $N \Uparrow S$ to take as scope any functor whose result is a S. In particular, the sequent

(53) $\mathrm{N} \Uparrow \mathrm{S} \Rightarrow \mathrm{N} \Uparrow (\mathrm{N} \Uparrow \mathrm{S})$

is not derivable, whereas the corresponding "semantic" sequent (up to permutation)

(54) $q : (e \to t) \to t \Rightarrow$
$\lambda R.\lambda P.q(\lambda x.R(P)(x)) : ((e \to t) \to (e \to t)) \to (e \to t) \to t$

is derivable in the undirected Lambek calculus. Sequent (54) will in particular raise (51) to a function that, applied to (50), produces (52), as required.

Furthermore, the solution proposed by Morrill (1993) to make the scope calculus complete is to restrict the intended interpretation of $\Uparrow$ so that (53) is not valid. Thus, *contra* Carpenter (1993), Morrill's logically more satisfying account of $\Uparrow$ is not a step towards making reading (49a) available.

We now give the derivation of the interpretation (49a) in our framework. The f-structure for (48) is:

(55)
$$f:\left[\begin{array}{ll} \text{PRED} & \text{'SEEK'} \\ \text{SUBJ} & g:\left[\begin{array}{ll}\text{PRED} & \text{'BILL'}\end{array}\right] \\ \text{OBJ} & h:\left[\begin{array}{ll} \text{SPEC} & \text{'A'} \\ \text{PRED} & \text{'CONVERSATION'} \\ \text{OBL}_{\text{WITH}} & i:\left[\begin{array}{ll} \text{SPEC} & \text{'EVERY'} \\ \text{PRED} & \text{'UNICORN'} \end{array}\right] \end{array}\right] \end{array}\right]$$

The two formulas **Bill-seeks** and **every-unicorn** can be derived as described before:

$$\textbf{Bill-seeks:}\quad \forall Y.\ (\forall s,p.\ (\forall X.\ h_\sigma \rightsquigarrow X \multimap s \rightsquigarrow ({}^\vee p)(X)) \multimap s \rightsquigarrow Y(p)) \\ \multimap f_\sigma \rightsquigarrow seek(Bill, {}^\wedge Y)$$

$$\textbf{every-unicorn:}\quad \forall G,S.\ (\forall x.\ i_\sigma \rightsquigarrow x \multimap G \rightsquigarrow ({}^\vee S)(x)) \\ \multimap G \rightsquigarrow every({}^\wedge unicorn, S)$$

The remaining lexical premises for (55) are:

$$\textbf{a:}\quad \forall H,R,T.\ ((\forall x.\ (h_\sigma \mathrm{VAR}) \rightsquigarrow x \multimap (h_\sigma \mathrm{RESTR}) \rightsquigarrow ({}^\vee R)(x)) \\ \otimes (\forall x.\ h_\sigma \rightsquigarrow x \multimap H \rightsquigarrow ({}^\vee T)(x))) \\ \multimap H \rightsquigarrow a(R,T)$$

$$\textbf{conv-with:}\quad \forall Z,X.\ (h_\sigma \mathrm{VAR}) \rightsquigarrow Z \otimes i_\sigma \rightsquigarrow X \\ \multimap (h_\sigma \mathrm{RESTR}) \rightsquigarrow conv\text{-}with(Z,X)$$

From these premises we immediately derive

$$\forall X,H,T.\ i_\sigma \rightsquigarrow X \otimes (\forall x.\ h_\sigma \rightsquigarrow x \multimap H \rightsquigarrow ({}^\vee T)(x)) \\ \multimap H \rightsquigarrow a({}^\wedge \lambda z.conv\text{-}with(z,X), T)$$

which can be rewritten as:

$$(56)\quad \forall H,T.\ (\forall x.\ h_\sigma \rightsquigarrow x \multimap H \rightsquigarrow ({}^\vee T)(x)) \multimap \\ \forall X.\ (i_\sigma \rightsquigarrow X \multimap H \rightsquigarrow a({}^\wedge \lambda z.conv\text{-}with(z,X), T))$$

If we apply the substitutions

$$X \mapsto x, G \mapsto H, S \mapsto {}^\wedge \lambda u.a({}^\wedge \lambda z.conv\text{-}with(u,v), T),$$

formula (56) can be combined with **every-unicorn** to yield the required quantifier-type formula:

$$(57)\quad \forall H,T.\ (\forall x.\ h_\sigma \rightsquigarrow x \multimap H \rightsquigarrow ({}^\vee T)(x)) \multimap \\ H \rightsquigarrow every({}^\wedge unicorn, {}^\wedge \lambda u.a({}^\wedge \lambda z.conv\text{-}with(z,u), T))$$

Using substitutions

$$\begin{array}{lll} H & \mapsto & s \\ T & \mapsto & p \\ Y & \mapsto & \lambda R.every({}^\wedge unicorn, {}^\wedge \lambda u.a({}^\wedge \lambda z.conv\text{-}with(z,u), R)) \end{array}$$

and modus ponens, we then combine (57) with **Bill-seeks** to obtain the desired final result:

$$f_\sigma \rightsquigarrow seek(Bill, {}^\wedge \lambda R.every({}^\wedge unicorn, {}^\wedge \lambda u.a({}^\wedge \lambda z.conv\text{-}with(z,u), R)))$$

Thus, we see that our more flexible connection between syntax and semantics permits the full range of type flexibility provided categorial *semantics* without losing the rigorous connection to syntax. In contrast, current categorial accounts of the syntax-semantics interface do not appear to offer the needed flexibility when syntactic and semantic composition are more indirectly connected, as in the present case.

Recently, Oehrle (1993) independently proposed a multidimensional categorial system with types indexed so as to keep track of the syntax-semantic connections that we represent with $\rightsquigarrow$. Using proof net techniques due to Moortgat (1992b) and Roorda (1991), he maps categorial formulas to first-order clauses similar to our meaning constructors, except that the formulas arising from determiners lack the embedded implication. Oehrle's system models quantifier scope ambiguities in a way similar to ours, but it is not clear that it can account correctly for the interactions with anaphora, given the lack of implication embedding in the clausal representation used.

6 Conclusion

Our approach exploits the f-structure of LFG for syntactic information needed to guide semantic composition, and also exploits the resource-sensitive properties of linear logic to express the semantic composition requirements of natural language. The use of linear logic as the glue language in a deductive semantic framework allows a natural treatment of quantification which automatically gives the right results for quantified NPs, their scopes and bound anaphora, and allows for a clean and natural treatment of extensional verbs and their arguments.

Indeed, the same basic facts are also accounted for in other recent treatments of compositionality, in particular categorial analyses with discontinuous constituency connectives (Moortgat, 1992a). These results suggest the advantages of a generalized form of compositionality in which the meaning constructors of phrases are represented by logical formulas rather than by functional abstractions as in traditional compositionality. The fixed application order and fixed type requirements of lambda terms are just too restrictive when it comes to encoding the freer order of information presentation in natural language.

In this observation, our treatment is closely related to systems of syntactic and semantic type assignment based on the Lambek calculus and its variants. However, we differ from those categorial approaches in providing an explicit link between functional structures and semantic derivations that does not depend on linear order and constituency in syntax to keep track of predicate-argument relations. Thus we avoid the need to force syntax and semantics into an uncomfortably tight categorial embrace.

Acknowledgments

Portions of this work were originally presented at the Second CSLI Workshop on Logic, Language, and Computation, Stanford University, and published as Dalrymple et al. (1994a); and at the Conference on Information-Oriented Approaches to Logic, Language and Computation, held at Saint Mary's College, Moraga, California, and published as Dalrymple et al. (1994b). We are grateful to the audiences at these venues for helpful comments.

We would particularly like to thank Johan van Benthem, Bob Carpenter, Jan van Eijck, Kris Halvorsen, Angie Hinrichs, David Israel, Ron Kaplan, Chris Manning, John Maxwell, Michael Moortgat, John Nerbonne, Stanley Peters, Henriette de Swart and an anonymous reviewer for the Second CSLI Workshop on Logic, Language, and Computation for comments and discussion. They are not responsible for any remaining errors, and we doubt that they will endorse all our analyses and conclusions, but we are sure that the end result is much improved for their help.

A Syntax of the Meaning and Glue Languages

The meaning language is based on Montague's intensional higher-order logic, with the following syntax:

$$
\begin{array}{llcll}
\text{(M-terms)} & M & ::= & c & \text{(Constants)} \\
 & & \mid & x & \text{(Lambda-variables)} \\
 & & \mid & \lambda x\, M & \text{(Abstraction)} \\
 & & \mid & M\, M & \text{(Application)} \\
 & & \mid & X & \text{(Glue-language variables)} \\
 & & \mid & \hat{} & \text{(``cap'' operator)} \\
 & & \mid & \check{} & \text{(``cup'' operator)}
\end{array}
$$

Terms are typed in the usual way; logical connectives such as *every* and *a* are represented by constants of appropriate type. The "cap" operator is polymorphic, and of type $\alpha \rightarrow (s \rightarrow \alpha)$; similarly the "cup" operator is of type $(s \rightarrow \alpha) \rightarrow \alpha$.

For readability, we will often "uncurry" $MN_1 \cdots N_m$ as $M(N_1, \ldots, N_m)$. Note that we allow variables in the glue language to range over meaning terms.

The glue language refers to three kinds of terms: meaning terms, f-structures, and semantic or σ-structures. f- and σ-structures are feature structures in correspondence (through projections) with constituent structure. Conceptually, feature structures are just functions which, when applied to attributes (a set of constants), return constants or other feature structures. In the following we let A range over some pre-specified set of attributes.

$$
\begin{array}{llcll}
\text{(F-terms)} & F & ::= & \uparrow & \text{(Indexical reference)} \\
 & & \mid & f \mid g \mid h \mid \cdots & \text{(F-structure constants)} \\
 & & \mid & (F A) & \text{(Attribute selection)}
\end{array}
$$

$$
\begin{array}{llcll}
(\sigma\text{-terms}) & S & ::= & F_\sigma & \text{(Semantic projection)} \\
 & & \mid & (S A) & \text{(Attribute selection)} \\
 & & \mid & H & \text{(Glue-language variable)}
\end{array}
$$

Glue-language formulas are built up using linear connectives from atomic formulas of the form $S \rightsquigarrow_\tau M$, whose intended interpretation is that the meaning associated with σ-structure S is denoted by term M of type τ. We omit the type subscript τ when it can be determined from context.

$$
\begin{array}{llcll}
\text{(Glue formulas)} & G & ::= & S \rightsquigarrow_\tau M & \text{(Basic assertion)} \\
 & & \mid & G \otimes G & \text{(Linear conjunction)} \\
 & & \mid & G \multimap G & \text{(Linear implication)} \\
 & & \mid & \Pi \lambda X.\, G & \text{(Quantification over M-terms)} \\
 & & \mid & \Pi \lambda H.\, G & \text{(Quantification over } \sigma\text{-terms)}
\end{array}
$$

We usually write $\Pi \lambda X.\, G$ as $\forall X.\, G$, and similarly for $\Pi \lambda H.\, G$.

B Proof rules for intensional higher-order linear logic

Identity $$\frac{}{F \vdash F}$$ $$\frac{\Gamma_1 \vdash F \qquad \Gamma_2, F \vdash D}{\Gamma_1, \Gamma_2 \vdash D}$$ **Cut**

Exch. Left $$\frac{\Gamma_1, F, G, \Gamma_2 \vdash D}{\Gamma_1, G, F, \Gamma_2 \vdash D}$$

λ **Left** $$\frac{\Gamma, F' \vdash D \qquad F \rightarrow_\lambda F'}{\Gamma, F \vdash D}$$ $$\frac{\Gamma \vdash D \qquad D \rightarrow_\lambda D'}{\Gamma \vdash D'}$$ λ **Right**

$\otimes$ **Left** $$\frac{\Gamma, F, G \vdash D}{\Gamma, (F \otimes G) \vdash D}$$ $$\frac{\Gamma_1 \vdash F \qquad \Gamma_2 \vdash G}{\Gamma_1, \Gamma_2 \vdash (F \otimes G)}$$ $\otimes$ **Right**

$\multimap$ **Left** $$\frac{\Gamma_1 \vdash F \qquad \Gamma_2, G \vdash D}{\Gamma_1, \Gamma_2, (F \multimap G) \vdash D}$$ $$\frac{\Gamma, F \vdash G}{\Gamma \vdash (F \multimap G)}$$ $\multimap$ **Right**

Π **Left** $$\frac{\Gamma, Pt \vdash D}{\Gamma, \Pi P \vdash D}$$ $$\frac{\Gamma \vdash Py}{\Gamma \vdash \Pi P}$$ Π **Right**

The Π **Right** rule only applies if y is not free in Γ, Σ, and any nonlogical theory axioms. We write $M \rightarrow_\lambda N$ to indicate that N can be obtained from M by one or more applications of $\alpha-$ or $\beta-$ reduction, or by the application of the rule:

$$\check{}(\hat{}(Q)) \rightarrow Q$$

to a sub-term of M.

References

Alsina, Alex. 1993. *Predicate Composition: A Theory of Syntactic Function Alternations.* Ph.D. thesis, Stanford University.

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

Bresnan, Joan, editor. 1982. *The Mental Representation of Grammatical Relations.* The MIT Press, Cambridge, MA.

Bresnan, Joan and Jonni M. Kanerva. 1989. Locative inversion in Chicheŵa: A case study of factorization in grammar. *Linguistic Inquiry*, 20(1):1–50. Also in E. Wehrli and T. Stowell, eds., Syntax and Semantics 26: Syntax and the Lexicon. New York: Academic Press.

Butt, Miriam. 1993. *The Structure of Complex Predicates.* Ph.D. thesis, Stanford University.

Carpenter, Bob. 1993. Quantification and scoping: a deductive account. Submitted for publication.

Dalrymple, Mary. 1993. *The Syntax of Anaphoric Binding.* Number 36 in CSLI Lecture Notes. Center for the Study of Language and Information.

Dalrymple, Mary, Angie Hinrichs, John Lamping, and Vijay Saraswat. 1993. The resource logic of complex predicate interpretation. In *Proceedings of the 1993 Republic of China Computational Linguistics Conference (ROCLING)*, Hsitou National Park, Taiwan, September. Computational Linguistics Society of R.O.C.

Dalrymple, Mary, John Lamping, Fernando C. N. Pereira, and Vijay Saraswat. 1994a. A deductive account of quantification in LFG. In Makoto Kanazawa, Christopher J. Piñón, and Henriette de Swart, editors, *Quantifiers, Deduction, and Context.* Center for the Study of Language and Information, Stanford, California.

Dalrymple, Mary, John Lamping, Fernando C. N. Pereira, and Vijay Saraswat. 1994b. Intensional verbs without type-raising or lexical ambiguity. In *Conference on Information-Oriented Approaches to Logic, Language and Computation*, Moraga, California. Saint Mary's College.

Dalrymple, Mary, John Lamping, and Vijay Saraswat. 1993. LFG semantics via constraints. In *Proceedings of the Sixth Meeting of the European ACL*, University of Utrecht, April. European Chapter of the Association for Computational Linguistics.

Fenstad, Jens Erik, Per-Kristian Halvorsen, Tore Langholm, and Johan van Benthem. 1987. *Situations, Language and Logic.* D. Reidel, Dordrecht.

Gamut, L. T. F. 1991. *Logic, Language, and Meaning*, volume 2: Intensional Logic and Logical Grammar. The University of Chicago Press, Chicago.

Girard, J.-Y. 1987. Linear logic. *Theoretical Computer Science*, 45:1–102.

Halvorsen, Per-Kristian. 1988. Situation Semantics and semantic interpretation in constraint–based grammars. In *Proceedings of the International Conference on Fifth Generation Computer Systems, FGCS-88*, pages 471–478, Tokyo, Japan, November. Also published as CSLI Technical Report CSLI-TR-101, Stanford University, 1987.

Halvorsen, Per-Kristian and Ronald M. Kaplan. 1988. Projections and semantic description in Lexical-Functional Grammar. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 1116–1122, Tokyo, Japan. Institute for New Generation Systems.

Hendriks, Herman. 1993. *Studied Flexibility: Categories and Types in Syntax and Semantics.* ILLC dissertation series 1993—5, University of Amsterdam, Amsterdam, Holland.

Hepple, Mark. 1990. *The Grammar and Processing of Order and Dependency: a Categorial Approach.* Ph.D. thesis, University of Edinburgh.

Howard, W.A. 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.* Academic Press, London, England, pages 479–490.

Huet, Gérard. 1975. A unification algorithm for typed $\overline{\lambda}$-calculus. *Theoretical Computer Science*, 1:27–57.

Kaplan, Ronald M. 1987. Three seductions of computational psycholinguistics. In Peter Whitelock, Harold Somers, Paul Bennett, Rod Johnson, and Mary McGee Wood, editors, *Linguistic Theory and Computer Applications.* Academic Press, London, pages 149–188.

Kaplan, Ronald M. and Joan Bresnan. 1982. Lexical-Functional Grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations.* The MIT Press, Cambridge, MA, pages 173–281.

Klein, Ewan and Ivan A. Sag. 1985. Type-driven translation. *Linguistics and Philosophy*, 8:163–201.

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

Levin, Lori S., Malka Rappaport, and Annie Zaenen, editors. 1983. *Papers in Lexical-Functional Grammar.* Indiana University Linguistics Club, Bloomington, IN.

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

Miller, Dale A. 1990. A logic programming language with lambda abstraction, function variables and simple unification. In Peter Schroeder-Heister, editor, *Extensions of Logic Programming*, Lecture Notes in Artificial Intelligence. Springer-Verlag.

Montague, Richard. 1974. The proper treatment of quantification in ordinary English. In Richmond H. Thomason, editor, *Formal Philosophy*. Yale University Press, New Haven, Connecticut.

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

Moortgat, Michael. 1992a. Generalized quantifiers and discontinuous type constructors. In W. Sijtsma and H. van Horck, editors, *Discontinuous Constituency*. Mouton de Gruyter, Berlin, Germany. To appear.

Moortgat, Michael. 1992b. Labelled deductive systems for categorial theorem proving. In P. Dekker and M. Stokhof, editors, *Proceedings of the Eighth Amsterdam Colloquium*, pages 403–423, Amsterdam. Institute for Logic, Language and Computation.

Morrill, Glyn. 1990. Intensionality and boundedness. *Linguistics and Philosophy*, 13(6):699–726.

Morrill, Glyn V. 1993. *Type Logical Grammar: Categorial Logic of Signs*. Studies in Linguistics and Philosophy. Kluwer Academic Publishers, Dordrecht, Holland. To appear.

Neale, Stephen. 1990. *Descriptions*. The MIT Press, Cambridge, MA.

Oehrle, Richard T. 1993. String-based categorial type systems. Workshop "Structure of Linguistic Inference: Categorial and Unification-Based Approaches," European Summer School in Logic, Language and Information, Lisbon, Portugal.

Pereira, Fernando C. N. 1990. Categorial semantics and scoping. *Computational Linguistics*, 16(1):1–10.

Pereira, Fernando C. N. 1991. Semantic interpretation as higher-order deduction. In Jan van Eijck, editor, *Logics in AI: European Workshop JELIA'90*, pages 78–96, Amsterdam, Holland. Springer-Verlag.

Pollard, Carl and Ivan A. Sag. 1987. *Information-Based Syntax and Semantics, Volume I*. Number 13 in CSLI Lecture Notes. Center for the Study of Language and Information, Stanford University.

Pollard, Carl and Ivan A. Sag. 1993. *Head-Driven Phrase Structure Grammar*. The University of Chicago Press, Chicago.

Roorda, Dirk. 1991. *Resource Logics: Proof-theoretical Investigations*. Ph.D. thesis, University of Amsterdam.

Saraswat, Vijay A. 1989. *Concurrent Constraint Programming Languages.* Ph.D. thesis, Carnegie-Mellon University. Reprinted by MIT Press, Doctoral Dissertation Award and Logic Programming Series, 1993.

Saraswat, Vijay A. 1993. A brief introduction to linear concurrent constraint programming. Technical report, Xerox Palo Alto Research Center, April.

Saraswat, Vijay A. and Patrick Lincoln. 1992. Higher-order, linear concurrent constraint programming. Technical report, Xerox Palo Alto Research Center, August.

Scedrov, Andre. 1993. A brief guide to linear logic. In G. Rozenberg and A. Salomaa, editors, *Current Trends in Theoretical Computer Science*, pages 377–394. World Scientific Publishing Co.

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

van Benthem, Johan. 1988. The Lambek calculus. In Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures.* D. Reidel, Dordrecht, pages 35–68.

van Benthem, Johan. 1991. *Language in Action: Categories, Lambdas and Dynamic Logic.* North-Holland, Amsterdam.