Making Logical Form type-logical

Glue Semantics for Minimalist syntax

Matthew Gotham

This is a post-peer-review, pre-copyedit version of an article published in *Linguistics and Philosophy*. The final authenticated version is available online at: <http://dx.doi.org/10.1007/s10988-018-9229-z>.

Abstract Glue Semantics is a theory of the syntax-semantics interface according to which the syntactic structure of a sentence produces premises in a fragment of linear logic, and the semantic interpretation(s) of the sentence correspond to the proof(s) derivable from those premises. This paper describes how Glue can be connected to a Minimalist syntactic theory and compares the result with the more mainstream approach to the syntax-semantics interface in Minimalism, according to which the input to semantic interpretation is a syntactic structure (Logical Form) derived by covert movement operations. I argue that the Glue approach has advantages that make it worth exploring.

Keywords syntax-semantics interface · Glue Semantics · linear logic · Minimalism

1 Introduction

In the mainstream approach to the syntax-semantics interface for broadly Chomskyan syntactic theories, of which Minimalism (Chomsky 1995) is the most recent iteration, semantic interpretation is fed directly by syntactic structures, in combination with lexical semantics.¹ Glue Semantics (henceforth Glue) is an alternative view, according to which the syntax-semantics interface takes the form of deduction in a fragment of linear logic (Girard 1987), with syntax in a sense forming the ‘input’ to this deduction and semantics the ‘output’. Glue is the mainstream approach to the syntax-semantics interface within LFG (Dalrymple et al 1993, 1999), for which it was originally developed. However, implementations also exist for HPSG (Asudeh and Crouch 2002c) and LTAG (Frank and van Genabith 2001), and in principle it is compatible with any syntactic framework. This paper implements Glue for Minimalism, and argues that the implementation overcomes some conceptual and empirical difficulties inherent in the mainstream approach.

Matthew Gotham
Department of Philosophy, Classics, History of Art and Ideas, University of Oslo, PO Box 1020 Blindern,
0315 Oslo, Norway
E-mail: matthew.gotham@ifikk.uio.no

¹ Some popular textbooks that take this approach are Larson and Segal (1995); Heim and Kratzer (1998); Chierchia and McConnell-Ginet (2000); Zimmermann and Sternefeld (2013).

The paper is structured as follows. In Section 2 I will outline some key properties of the mainstream approach, and give an indication of where the Glue approach to be presented will diverge from these. Section 3 gives a basic introduction to Glue. In Section 4 I define a toy version of Minimalism and show how Glue can be connected to it. The Glue approach so defined is compared with related proposals in Section 5. In Section 6 I show how Minimalism with Glue accounts for the interpretation of nested DPs, and in so doing overcomes difficulties inherent in the mainstream approach. Section 7 concludes.

2 The mainstream approach

In this section I will lay out some properties of the mainstream approach to the syntax-semantics interface, and then outline the ways in which the account to be presented in this paper differs from them.

2.1 Compositional rules

The first property, which has already been alluded to, is that syntax produces structures (normally, trees) that serve as the input to semantic compositional rules. A very widely-adopted concrete proposal is that the workhorse of these is the rule of function application, defined over local trees by Heim and Kratzer (1998, 44) as in (1) (where $\llbracket X \rrbracket$ is the denotation of X).

- (1) If α is a branching node, $\{\beta, \gamma\}$ the set of its daughters, and $\llbracket \beta \rrbracket$ is a function whose domain contains $\llbracket \gamma \rrbracket$, then $\llbracket \alpha \rrbracket = \llbracket \beta \rrbracket (\llbracket \gamma \rrbracket)$.

For example, the idea is that syntax produces structures such as (2), which is interpreted as shown in (3). (2) is interpretable because $\llbracket a \rrbracket$ is a function that has $\llbracket man \rrbracket$ in its domain.

- (3) $\llbracket DP \rrbracket = \llbracket D \rrbracket (\llbracket N \rrbracket)$
 $= \llbracket a \rrbracket (\llbracket man \rrbracket)$

2.2 Movement and assignments

The second property is that syntax is taken to involve rules of *movement* in the derivation of some structures. For example, the structure of the relative clause ‘who Jacob sees’ is formed by movement of the DP ‘who’ from the complement of the verb to the specifier of CP, as indicated schematically in (4).

- (4) who Jacob sees ~~who~~
-

In the structure that is the input to interpretation, a moving constituent leaves a trace in its original position, and indices encode the dependencies created by movement. In (5) I have assumed, following Heim and Kratzer (1998), that the (arbitrarily chosen) index (in this case, 4) is not on the moved constituent itself, but rather adjoins to the sister of the moved constituent.

In effect, the interpretative rules treat the trace like a variable, and the index associated with the moved constituent as a binder for that variable. Making this work requires taking denotations *relative to an assignment*, here shown as $\llbracket X \rrbracket^g$ (for assignment g), where g is a function with domain the set of natural numbers $\mathbb{N}$ and range the domain of discourse D . Most denotations are assignment-invariant, but for any assignment g and any trace t_n , $\llbracket t_n \rrbracket^g = g(n)$. A compositional principle adequate for structures like (5) is given in (6), based on (Heim and Kratzer 1998, 186). Here, $g[n \mapsto x]$ is the assignment h such that $h(n) = x$ and for any $m \in \mathbb{N}$ such that $m \neq n$, $h(m) = g(m)$.

- (6) If α is a branching node, $\{n, \beta\}$ the set of its daughters, and n is a numerical index, then for any assignment g , $\llbracket \alpha \rrbracket^g$ = the function f such that for any $o \in D$, $f(o) = \llbracket \beta \rrbracket^{g[n \mapsto o]}$.

For example, (5) is interpreted as shown in (7), where I have assumed for the sake of simplicity that $\llbracket_{DP} \text{who} \rrbracket$, $\llbracket_C \rrbracket$ and $\llbracket_T -s \rrbracket$ are all semantically inert.

- (7) $\llbracket CP \rrbracket^g$ = the $f \in \{0, 1\}^D$ such that for any $o \in D$, $f(o) = \llbracket TP \rrbracket^{g[4 \mapsto o]}$
 = the $f \in \{0, 1\}^D$ such that for any $o \in D$, $f(o) = \llbracket VP \rrbracket^{g[4 \mapsto o]}(\llbracket Jacob \rrbracket^{g[4 \mapsto o]})$
 = the $f \in \{0, 1\}^D$ such that for any $o \in D$,
 $f(o) = (\llbracket see \rrbracket^{g[4 \mapsto o]}(\llbracket t_4 \rrbracket^{g[4 \mapsto o]}))(\llbracket Jacob \rrbracket^{g[4 \mapsto o]})$
 = the $f \in \{0, 1\}^D$ such that for any $o \in D$,
 $f(o) = (\llbracket see \rrbracket^{g[4 \mapsto o]}(g[4 \mapsto o](4)))(\llbracket Jacob \rrbracket^{g[4 \mapsto o]})$
 = the $f \in \{0, 1\}^D$ such that for any $o \in D$,
 $f(o) = (\llbracket see \rrbracket^{g[4 \mapsto o]}(o))(\llbracket Jacob \rrbracket^{g[4 \mapsto o]})$

In the first line of (7), we see how the index associated with the moved constituent manipulates the assignment relative to which TP is interpreted: it is no longer interpreted relative to g but rather to $g[4 \mapsto o]$. This then affects the assignment relative to which every constituent contained in TP is interpreted including, crucially, the trace. Since $g[4 \mapsto o](4) = o$ (for any $o \in D$), the interpretation of the trace will co-vary with the argument to the relative clause, and so the trace behaves like a bound variable.

2.3 Covert movement and Logical Form

The third property is that that movement can be *covert*, i.e. not reflected in the structure that is pronounced. For example, the structure that is the input to the semantic interpretation of (8), (9), is one derived from surface structure by a covert movement operation of *Quantifier Raising* (QR) (May 1977).

(8) Jacob sees a ladder.

The interpretation of (8), then, is as shown in (10).

$$(10) \quad \llbracket a \text{ ladder} \rrbracket^g (\text{the } f \in \{0, 1\}^D \text{ such that for any } o \in D, \\ f(o) = (\llbracket see \rrbracket^{g[6 \mapsto o]}(o)) (\llbracket Jacob \rrbracket^{g[6 \mapsto o]}))$$

This approach is partly motivated by the fact that, on the simplest assumptions about what the denotations are, $\llbracket a \text{ ladder} \rrbracket^g$ is actually not in the domain of $\llbracket see \rrbracket^g$ —but the (assignment-dependent) denotation of the trace is. It is also partly motivated by the idea that covert movement can feature in an explanation for scope ambiguity. On this view, scope ambiguity is syntactic ambiguity at a level of representation after QR has happened, called *Logical Form* (LF). For example, the ambiguous sentence (11) is taken to have two possible LFs: one corresponding to the surface scope interpretation (‘there is someone who sees everyone’), and one to the inverse scope interpretation (‘everyone is seen by someone or other’), as shown in (12-a) and (12-b) respectively.

(11) Someone sees everyone.

(12) a.

b.

In the Glue account to be presented in this paper, the structures over which rules of semantic interpretation are defined are not trees as such, but rather structures (specifically, proofs) that are in a sense derived from trees.² Function application is still the workhorse of the process of semantic interpretation; this follows from the logic underlying the proofs. The logic underlying the proofs also provides the variable-binding mechanism needed to interpret movement, and so there is no need for traces in syntax. As I will argue in Section 4.3.1, this accords well with the desire, from a syntactic perspective, to do away with traces for independent reasons. The account also has the property that there is no need for covert movement in order to account for ambiguities of scope.

² Or syntactic objects like them. See fn. 11.

1. Function, given by Glue implementation (Section 4.2)
2. Relation, given by linear logic proof theory (Section 3.2)
3. Function, given by Curry-Howard correspondence (Section 3.3)

Fig. 1: The syntax-semantics interface according to Glue

3 Basics of Glue semantics

3.1 Conceptual introduction

The mainstream approach as outlined in the previous section adopts the assumption that the mapping from syntactic structure to semantic interpretation is functional; that is to say, that structure plus lexical semantics *determines* interpretation. From this it follows that if a sentence is ambiguous, such as (11), then that ambiguity must be either lexical or syntactic. As noted above, the mainstream position is that the ambiguity is syntactic, at LF.³

According to the Glue approach, however, the mapping from syntactic structure to semantic interpretation is relational, which is to say structure plus lexical semantics *constrains* interpretation. An informal statement of the relevant constraints for (11) would be as follows:

- (13)
- $\llbracket \text{see} \rrbracket$ applies to X , then Y , to form Z .
 - $\llbracket \text{someone} \rrbracket$ applies to (something that applies to Y to form Z) to form Z .
 - $\llbracket \text{everyone} \rrbracket$ applies to (something that applies to X to form Z) to form Z .

The contribution of syntactic structure is to provide constraints like this which, in combination with the underlying logic for meaning composition (see Section 3.2), guarantee that all and only the right interpretations of a sentence are derivable. So for example, there are *two* ways of combining $\llbracket \text{someone} \rrbracket$, $\llbracket \text{see} \rrbracket$ and $\llbracket \text{everyone} \rrbracket$, while respecting the constraints given in (13), and arriving at Z ; and we can think of ‘arriving at Z ’ as ‘deriving an interpretation of type t ’, which is what we want to do. Those two ways give us the two interpretations that we want, namely the surface scope interpretation and the inverse scope interpretation. The constraints rule out, for example, the derivation of an interpretation that has $\llbracket \text{everyone} \rrbracket$ as the agent of seeing.

Formally, the constraints are formulae of linear logic (Girard 1987), and the constraint-respecting ways of putting interpretations together correspond to linear logic proofs using all and only those formulae as premises. The architecture of the syntax-semantics interface is therefore as shown in Fig. 1: scope ambiguity arises because of the possibility that distinct proofs may be constructed from the same premises, i.e. because the mapping labelled 2 in Fig. 1 is a relation. However, every proof corresponds to exactly one interpretation, and so the proofs play a similar role in this approach to that played by LFs in the mainstream approach (more on this in Section 4.3.2).

³ Lexical ambiguity has also been considered, e.g. by Hendriks (1987), and sometimes both approaches are combined.

Explicating the mappings labelled 2 and 3 in Fig. 1 is the task of the remainder of Section 3. The mapping labelled 1 is the properly linguistic part of the setup, and constitutes the novel step in this paper. It is the subject matter of Section 4, particularly Section 4.2.

3.2 Introduction to linear logic

Linear logic is a **substructural logic**. What that means is that it lacks certain **structural rules** that are valid in classical logic, specifically the rules of **contraction** (14) and **weakening** (15).

$$(14) \quad \frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B} \text{ Contraction (not valid in linear logic)}$$

$$(15) \quad \frac{\Gamma \vdash B}{\Gamma, A \vdash B} \text{ Weakening (not valid in linear logic)}$$

That is to say, in linear logic premises may not be duplicated (no contraction) or discarded (no weakening) at will; every premise must be ‘used’ exactly once for a proof to be valid. For this reason, linear logic is often described as a ‘logic of resources’ (Crouch and van Genabith 2000, 5).

Examples of sequents that would be valid in classical logic but which are not valid in linear logic are given in (16), which would require contraction, and (17), which would require weakening. N.B. $\multimap$ is linear implication.

$$(16) \quad A, A \multimap (A \multimap B) \not\vdash B$$

$$(17) \quad A, B \not\vdash A$$

The sequent (16) is invalid because it would require the premise A to be ‘used’ twice, when in fact it is only present once. The sequent (17) is invalid because it would require the premise B not to be ‘used’. As Asudeh (2004) argues, these considerations make linear logic an ideal logic for semantic composition, because in computing the meaning of some expression we want every lexical item to contribute its meaning (if it has one) exactly once to the meaning of the expression as a whole.

In practice, only a small fragment of linear logic is used for any Glue implementation. The fragment used in this paper will be a first-order language (following Kokkonidis (2008)). There are two predicates, both one-place: e and t . These have specifically been chosen to reflect the types of lambda calculus expressions (see (20) below). Constants will be natural numbers and variables will be uppercase letters. Only two connectives will be used: the implication $\multimap$ and the universal quantifier $\Box$.⁴ In order to save space, subscript notation will be used, e.g. I will write e_1 instead of $e(1)$. So for example, e_1, e_Y, t_2 and $\Box X(e_X \multimap t_X)$ are well-formed formulae of the fragment.

The connectives used in the fragment have the rules of inference shown below in tree-style natural deduction format.

$$(18) \quad \text{Rules of inference for } \multimap$$

$$\text{a. } \frac{A \multimap B \quad A}{B} \multimap \text{ elimination (linear modus ponens)}$$

⁴ This choice of notation is non-standard (but see (Morrill 1994, Chapter 6)). It is more common to use the symbol $\forall$, but since this symbol will be used on the meaning side, I have decided to use a different one on the glue side.

- b.
$$\frac{\begin{array}{c} [A]^n \\ \vdots \\ B \end{array}}{A \multimap B} \multimap \text{ introduction, } n \text{ (linear conditional proof)}$$
- (19) Rules of inference for $\Box$
- a.
$$\frac{\Box X(A)}{A[m/X]} \Box \text{ elimination (linear universal instantiation)}$$

$$m \text{ a constant or variable free for } X$$
- b.
$$\frac{A}{\Box X(A)} \Box \text{ introduction (linear universal generalization)}$$

$$X \text{ a variable not free in any open premise}$$

3.3 Connection to semantic interpretation

The core idea of Glue is that the process of deriving an interpretation corresponds to the process of constructing a proof in a fragment of linear logic, with the premises to the proof being contributed by syntax (see Fig. 1). Each premise is paired with an expression in the simply-typed lambda calculus (Church 1940) according to the lexical semantics of the individual words, and every step of deduction corresponds to an operation in the λ -calculus.

In what follows, the pairing of a λ -calculus expression m with a linear logic formula Φ will be displayed as $m : \Phi$, which will often be referred to as a ‘meaning constructor’. The left of the colon will often be referred to as the ‘meaning side’, and the right as the ‘glue side’. Possible meaning constructors are constrained by the type map defined in (20).

- (20) TYPE MAP FOR MEANING CONSTRUCTORS. For any meaning constructor $m : \Phi$, m is of type $\text{TY}(\Phi)$, where
- a. For any constant or variable α :
- (i) $\text{TY}(t_\alpha) = t$
 - (ii) $\text{TY}(e_\alpha) = e$
- b. For any formulae A and B , and any variable X :
- (i) $\text{TY}(A \multimap B) = \text{TY}(A) \multimap \text{TY}(B)$
 - (ii) $\text{TY}(\Box X(A)) = \text{TY}(A)$

So for example, if we have the meaning constructor $f : e_2 \multimap (e_7 \multimap t_9)$, then we know that f is of type $e \multimap (e \multimap t)$; and if we have the meaning constructor $g : \Box Z((e_4 \multimap t_Z) \multimap t_Z)$, then we know that g is of type $(e \multimap t) \multimap t$.

We therefore have the two pairings shown in (21).

- (21) λ -calculus expression linear logic formula
 operation in the λ -calculus linear logic rule of inference

Each of the rules of inference given in (18)–(19), then, corresponds to an operation in the lambda calculus. The correspondence is shown in Fig. 2. $\multimap$ elimination on the glue side corresponds to function application on the meaning side, and $\multimap$ introduction on the glue side corresponds to abstraction on the meaning side. The rules for $\Box$ are semantically inert, i.e. they have no effect on meaning.⁵

⁵ Which makes this an instance of the Curry-Howard ‘correspondence’ and not ‘isomorphism’: distinct proofs may correspond to the same lambda calculus expression, but not *vice versa*.

	Elimination	Introduction
$\multimap$	$\frac{\begin{array}{c} \vdots \\ f : A \multimap B \end{array} \quad \begin{array}{c} \vdots \\ a : A \end{array}}{f(a) : B} \multimap_E$	$\frac{\begin{array}{c} [v : A]^n \\ \vdots \\ f : B \end{array}}{\lambda v(f) : A \multimap B} \multimap_{I,n}$ <p>exactly one hypothesis discharged</p>
$\Box$	$\frac{\begin{array}{c} \vdots \\ f : \Box X(A) \end{array}}{f : A[m/X]} \Box_E$ <p>m a constant or variable free for X</p>	$\frac{\begin{array}{c} \vdots \\ f : A \end{array}}{f : \Box X(A)} \Box_I$ <p>X a variable not free in any open premise</p>

Fig. 2: Rules of inference for the fragment of linear logic used, and their images under the Curry-Howard correspondence

We therefore have an instance of the Curry-Howard correspondence (Howard 1980) between proofs (in the fragment of linear logic) and programs (in the λ -calculus), as shown in Fig. 2, and between formulae (of linear logic) and types (of λ -calculus expressions), as shown in (20). The correspondence is summarized in (22).⁶

(22)	Linear logic		λ -calculus
	proofs	as	programs
	$\multimap$ elimination		function application
	$\multimap$ introduction		λ abstraction
	($\Box$ elimination		identity)
	($\Box$ introduction		identity)
	formulae	as	types
	implicational formula		functional type
	(universally quantified formula		the same as the type of the formula it's prefixed to)

The existence of a Curry-Howard correspondence for it makes the fragment of linear logic used in this paper a **type logic** (see (Jäger 2005, 31–42) for discussion). In a type-logical (categorical) grammar, however, the entire linguistic theory would be characterized by (a) correspondence(s) like this. That is not the case here; the syntactic theory of the natural language is stated separately, and connected to the type logic (and hence the interpretation) by another mechanism. A general outline of how this works is given in the next section, then stated specifically and in detail in Section 4.

3.4 Connection to syntactic analysis

As stated above, in the Glue setup syntax has to provide the linear logic formulae that serve as premises for the linear logic proof(s). Lexical semantics provides the λ -calculus expres-

⁶ By ‘implicational formula’, I mean a formula that has (linear) implication as its main connective.

sions paired with those formulae (thus giving us meaning constructors), and the interpretation for each proof is determined by the Curry-Howard correspondence.

Practically, what we need from syntax for a Glue implementation is some specification of how the argument positions of different words in the sentence have to match up. For example, in (23),

(23) Jacob sees Rachel.

syntax has to provide the information that *Rachel* is the object argument, and *Jacob* the subject argument, of *sees*. One way of making this idea explicit, which will be adopted in this paper, involves assigning labels to argument positions of predicates. An illustration is given in (24).

(24)	label	assigned to
	1	the object argument of <i>sees</i> <i>Rachel</i>
	2	the subject argument of <i>sees</i> <i>Jacob</i>
	3	the sentence as a whole

We can think of the information contained in (24) as licensing the transformation of the underspecified meaning constructors at the top of (25), containing information that can be derived from lexical semantics alone, into the fully specified meaning constructors at the bottom of (25).⁷

Equipped with fully specified meaning constructors, we are in a position to construct a proof and, with it, an interpretation for the sentence, as shown in (26).

$$\begin{array}{c}
 (26) \quad \frac{\text{see} : e_1 \multimap (e_2 \multimap t_3) \quad \text{r} : e_1}{\text{see}(\text{r}) : e_2 \multimap t_3} \multimap_E \quad \text{j} : e_2 \multimap_E \\
 \hline
 \text{see}(\text{r})(\text{j}) : t_3
 \end{array}$$

For simple cases like (23), the Glue implementation seems like an unnecessarily complicated one in comparison with the mainstream approach. What are far more interesting are cases in which more than one proof can be constructed from the same premises, thereby deriving the different interpretations of an ambiguous sentence. For example, we can consider a scopally ambiguous sentence like (11). Let us suppose that the lexicon provides the underspecified meaning constructors shown in (27).

⁷ These ‘underspecified meaning constructors’ are for illustration only; I will describe what they are more precisely in Section 4.2.

$$(27) \quad \begin{array}{ccc} \text{someone} & \text{sees} & \text{everyone} \\ \Downarrow & \Downarrow & \Downarrow \\ \lambda P(\exists x(\text{person}(x) \wedge P(x))) : & \text{see} : & \lambda Q(\forall y(\text{person}(y) \rightarrow Q(y))) : \\ \Box X((e_- \multimap t_X) \multimap t_X) & e_- \multimap (e_- \multimap t_-) & \Box Y((e_- \multimap t_Y) \multimap t_Y) \end{array}$$

The effect of the universal quantifier on the glue side in these (underspecified) meaning constructors is to say that these expressions can take scope anywhere. Nevertheless, we still need syntax to resolve the argument positions relative to the verb. Following the example above, syntactic analysis of (11) can take the form shown in (28), thereby deriving the fully specified meaning constructors shown in (29).

(28)	label	assigned to
	1	the object argument of <i>sees</i> <i>everyone</i>
	2	the subject argument of <i>sees</i> <i>someone</i>
	3	the sentence as a whole

$$(29) \quad \begin{array}{ccc} \Downarrow & \Downarrow & \Downarrow \\ \lambda P(\exists x(\text{person}(x) \wedge P(x))) : & \text{see} : & \lambda Q(\forall y(\text{person}(y) \rightarrow Q(y))) : \\ \Box X((e_2 \multimap t_X) \multimap t_X) & e_1 \multimap (e_2 \multimap t_3) & \Box Y((e_1 \multimap t_Y) \multimap t_Y) \end{array}$$

From the premises in (29), two proofs to a type- t conclusion are possible. These correspond to the surface scope interpretation of the sentence, shown in Fig. 3, and the inverse scope interpretation, shown in Fig. 4.⁸ In both of these proofs, the two universally quantified linear logic formulae (on the glue side) are both instantiated to the same value: 3, indicating that both quantified noun phrases take scope in the same clause. Different scopal interpretations nevertheless arise because they are able to take different scope orders with respect to each other. Examples where quantified DPs take scope at different positions, and hence where the first-order character of the linear logic fragment is actually used, will be given in Section 6.

4 Implementation in Minimalism

In this section a toy grammar for Minimalism will be defined, and then a Glue implementation will be given for it. The key ideas are that

1. syntactic objects have features,
2. the structure-building operations are based on the matching of features,
3. every feature bears an index, and when two features match their indices must also match, and
4. those indices are ‘shared’ with the linear logic formulae in meaning constructors, thereby providing the syntax-semantics connection.

Points 1 and 2 are widely-shared assumptions in Minimalist syntax, and the format chosen in Section 4.1 below is heavily based on Adger (2003, 2010).⁹ Points 3 and 4 constitute the novel steps necessary for Glue implementation. They will be discussed in Section 4.2.

⁸ In these proofs and in the rest of this paper, lambda calculus expressions are normalized without comment. I will also often use a dot following a variable binder to indicate unbounded scope to the right, separately for each side of the meaning constructor. So for example, in these proofs $\lambda Q.\exists y.\text{person}(y) \wedge Q(y) : \Box Y.(e_2 \multimap t_Y) \multimap t_Y$ is the same as $\lambda Q(\exists y(\text{person}(y) \wedge Q(y))) : \Box Y((e_2 \multimap t_Y) \multimap t_Y)$.

⁹ An anonymous reviewer suggests instead using a Minimalist Grammar in the style of Stabler (1997), which is in many ways like Adger’s system but has the advantages of being better formally understood and

Alongside the interpretable features we have **uninterpretable** features, which describe what a syntactic object *needs* (in a sense to be explained shortly). For any interpretable feature F , uF is the **weak** uninterpretable version of F , and uD^* is the **strong** uninterpretable version of F . The strong uninterpretable features are the ones that trigger movement. As an example, (31) shows the various forms of determiner feature.

(31)

interpretable	uninterpretable	
	weak	strong
D	uD	uD^*

Preliminarily, a **feature structure** is an ordered pair $\langle A, B \rangle$, where A is a set of interpretable features, exactly one of which is a category feature, and B is a (possibly empty) sequence of uninterpretable features. A **lexical item** is a two-node directed graph in which a node labelled by a feature structure dominates a node labelled by a phonological form. Some example lexical items are shown in (32).¹¹

(32)

$\langle \{V\}, \langle uD, uD \rangle \rangle$	$\langle \{T, \text{pres}\}, \langle uD^* \rangle \rangle$
see	-s

which will often be represented as:

V $\langle uD, uD \rangle$	T[pres] $\langle uD^* \rangle$
see	-s

The representation at the bottom of (32) has been chosen so as to highlight the category feature in each feature structure.

4.1.2 Structure-building rules

Each of the structure-building operations takes one or two graphs as input and produces a graph as output.¹² The rules in (33)–(35) are to be read as saying that if you have (a) graph(s) rooted in the feature structure(s) shown on the input side, then you can combine them in the way shown on the output side. In no case is linear order crucial, either of the inputs or of the daughter nodes in the output—I take linear order to be determined separately.

In each of the rules (33)–(35), A and B stand for arbitrary (interpretable) features, X and Y stand for arbitrary sets of (interpretable) features, and Σ and Γ stand for arbitrary sequences of (uninterpretable) features.

(33) HoPs merge

¹¹ These definitions are preliminary and need to be revised in order to take account of indices and meaning constructors. The revision is outlined informally in Section 4.2 and defined formally in Appendix A.2.

¹² Specifically, each of the structures is a rooted, directed, acyclic graph. They are not trees because of the way that (35) has been formulated, meaning that paths from root to any leafs are not necessarily unique.

$$\begin{array}{c} \{A\} \cup X \\ \Sigma \end{array} + \begin{array}{c} \{B\} \cup Y \\ \Gamma \end{array} \Rightarrow \begin{array}{c} \{A\} \cup X \\ \Sigma \\ \swarrow \quad \searrow \\ \{A\} \cup X \quad \{B\} \cup Y \\ \Gamma \end{array}$$

Where A and B are in the same hierarchy of projections (HoPs) and A is higher on that HoPs than B

(34) Select merge: External

$$\begin{array}{c} X \\ \langle uB \rangle \frown \Sigma \end{array} + \begin{array}{c} \{B\} \cup Y \\ \Gamma \end{array} \Rightarrow \begin{array}{c} X \\ \Sigma \\ \swarrow \quad \searrow \\ X \quad \{B\} \cup Y \\ \langle uB \rangle \quad \Gamma \end{array}$$

$\frown$ indicates sequence concatenation. The struck-out uninterpretable feature is not present (at all), but is shown struck-out simply as a visual aide.

(35) Select merge: Internal

$$\begin{array}{c} X \\ \langle uB^* \rangle \frown \Sigma \\ \triangle \\ \{B\} \cup Y \\ \Gamma \end{array} \Rightarrow \begin{array}{c} X \\ \Sigma \\ \swarrow \quad \searrow \\ X \quad \{B\} \cup Y \\ \langle uB^* \rangle \quad \Gamma \\ \triangle \\ \{B\} \cup Y \\ \Gamma \end{array}$$

Where the re-merged constituent is a maximal projection of the feature B.

Rule (35) requires an additional constraint to the effect that the constituent that remerges is the closest matching one to the root of the input graph, such as the ‘Locality of Matching’ principle given by Adger (2003, 218). As the diagram indicates, I understand ‘movement’ as the creation of structures of multidominance. This is not crucial, but it is the simplest way to ensure that no duplication of semantic resources is caused by any process of copying in the syntax. This issue will be raised again in Section 4.3.1.

An example derivation Given the rules defined in (33)–(35), the syntactic object pronounced as (23) can be derived as shown in Fig. 5. Note that no uninterpretable features are present in the final output structure; this I take to be a requirement of candidate sentential syntactic objects.

4.2 The connection to Glue

In order to establish a connection to meaning constructors that will do the job for compositional semantics, the definitions of feature structures and lexical items from Section 4.1 need to be revised. Formal definitions are given in Appendix A.2; in the rest of this section I will lay out all the crucial points informally.

In the lexicon, every feature (interpretable or uninterpretable) on a lexical item bears an **index variable**. The set of index variables is the same as the set of variables of the linear logic fragment, i.e. X , Y , etc. At the same time, within each lexical item there are free variables on the glue side of each meaning constructor. These are chosen so as to match the

External merge:

External merge:

HoPs merge:

Internal merge:

Fig. 5: Derivation of (23) according to the rules in (33)–(35)

index variables on features. For example, we might have the lexical entry for *see* shown in (36).

$$\begin{array}{c}
 (36) \quad V_X \\
 \langle uD_Z, uD_Y \rangle \\
 | \\
 \text{see} \\
 \text{see} : e_Z \multimap (e_Y \multimap t_X)
 \end{array}$$

In (36), the same (index) variable (Z) is on the first uninterpretable D feature, and is the argument to the e predicate in the antecedent of the linear logic formula on the glue side of the meaning constructor. This sharing of (index) variables encodes the fact that the first DP to merge with the verb is the object argument.

The allocation of index variables to features in a lexical item is subject to the general constraints that (i) the same index variable goes on every interpretable feature, and (ii) a different index variable goes on every uninterpretable feature.

Next, in preparation for going into any larger structure, index variables must be resolved into **indices**. The set of indices is the same as the set of constants of the linear logic fragment, i.e. 1, 2, etc. The mapping from (index) variables to indices/constants must be uniform within a lexical item. So for example, (36) can be resolved to (37-a), but *not* (37-b), because in (37-b) the resolution has transformed the Z on the uninterpretable D feature into 1, but has not transformed the free occurrence of Z in the meaning constructor to 1—it has transformed it to 4. This requirement, together with the constraints on the allocation of index variables, keeps the connection between features and semantic argument positions intact and thereby serve to prevent the ‘mixing up’ of argument positions in proofs.

$$\begin{array}{ll}
 (37) \quad \text{a.} & \begin{array}{c} V_3 \\ \langle uD_1, uD_2 \rangle \\ | \\ \text{see} \\ \text{see} : e_1 \multimap (e_2 \multimap t_3) \end{array} \\
 \text{b.} \quad * & \begin{array}{c} V_3 \\ \langle uD_1, uD_2 \rangle \\ | \\ \text{see} \\ \text{see} : e_4 \multimap (e_2 \multimap t_3) \end{array}
 \end{array}$$

The final piece of the jigsaw is the requirement that, whenever the features in some structure-building rule must match, the indices on those features must also match. So for example, in HoPs merge as outline in (33) the indices on the features A and B must be the same, in external merge as outlined in (34) the indices on uB and B must be the same, and in internal merge as outlined in (35), the indices on uB^* and B must be the same.

(38)	a.	$\begin{array}{c} D_X \\ \Delta \\ \text{Jacob} \\ \mathbf{j} : e_X \end{array}$	$\begin{array}{c} T_X[\text{pres}_X] \\ \\ -s \end{array}$	$\begin{array}{c} V_X \\ \langle uD_Z, uD_Y \rangle \\ \\ \text{see} \\ \text{see} : e_Z \multimap (e_Y \multimap t_X) \end{array}$	$\begin{array}{c} D_X \\ \Delta \\ \text{Rachel} \\ \mathbf{r} : e_X \end{array}$
	b.	$\begin{array}{c} \Downarrow \\ D_2 \\ \Delta \\ \text{Jacob} \\ \mathbf{j} : e_2 \end{array}$	$\begin{array}{c} \Downarrow \\ T_3[\text{pres}_3] \\ \\ -s \end{array}$	$\begin{array}{c} \Downarrow \\ V_3 \\ \langle uD_1, uD_2 \rangle \\ \\ \text{see} \\ \text{see} : e_1 \multimap (e_2 \multimap t_3) \end{array}$	$\begin{array}{c} \Downarrow \\ D_1 \\ \Delta \\ \text{Rachel} \\ \mathbf{r} : e_1 \end{array}$

In the same way, we can imagine the resolution shown in (40) (with the resolution of T and V the same as in (38)), from which the the structure of (11) can be derived, as shown in (41).

¹³ Well, maybe not lexical, as there may be more internal structure to the proper name, as the 'coathanger' indicates. But this concern is tangential to the main point.

$$(40) \quad \begin{array}{cc} \begin{array}{c} D_Y \\ \Delta \\ \text{someone} \\ \lambda P.\exists x.\mathbf{person}(x) \wedge P(x) : \\ \Box X.(e_Y \multimap t_X) \multimap t_X \end{array} & \begin{array}{c} D_Z \\ \Delta \\ \text{everyone} \\ \lambda Q.\forall y.\mathbf{person}(y) \rightarrow Q(y) : \\ \Box Y.(e_Z \multimap t_Y) \multimap t_Y \end{array} \end{array}$$

$$\Downarrow$$

$$\Downarrow$$

$$(41) \quad \begin{array}{cc} \begin{array}{c} D_2 \\ \Delta \\ \text{someone} \\ \lambda P.\exists x.\mathbf{person}(x) \wedge P(x) : \\ \Box X.(e_2 \multimap t_X) \multimap t_X \end{array} & \begin{array}{c} D_1 \\ \Delta \\ \text{everyone} \\ \lambda Q.\forall y.\mathbf{person}(y) \rightarrow Q(y) : \\ \Box Y.(e_1 \multimap t_Y) \multimap t_Y \end{array} \end{array}$$

The meaning constructors contributed in (41) are the same as those listed in (29), and so there are two proofs available: the surface scope interpretation shown in Fig. 3, and the inverse scope interpretation shown in Fig. 4. We therefore have an account of the ambiguity of (11) that is based neither on any syntactic ambiguity, nor on any lexical ambiguity, nor on any ad-hoc type-shifting rules.

4.3 Remarks

4.3.1 The interpretation of movement

Covert movement has been dispensed with but, of course, overt movement has not. In fact, we have already seen examples of the interpretation of structures involving A-movement in the present account, as a consequence of adopting the VP-internal subject hypothesis. We can now see how $\bar{A}$ -movement is treated by examining once more the relative clause ‘who Jacob sees’. Let us assume the additional lexical items shown in (42).

With appropriate resolutions of the index variables, these lexical items can combine with other lexical items that we have already seen to give the syntactic structure shown in Fig. 6. And with the meaning constructors from the structure shown in Fig. 6, the relative clause can be interpreted as shown in (43).

$$(43) \quad \frac{\frac{\frac{\text{see} : e_1 \multimap (e_2 \multimap t_3) \quad \left[\begin{smallmatrix} y : \\ e_1 \end{smallmatrix} \right]^1}{\text{see}(y) : e_2 \multimap t_3} \multimap_E \quad \mathbf{j} : e_2}{\text{see}(y)(\mathbf{j}) : t_3} \multimap_{I,1} \quad \frac{\lambda P. \lambda Q. \lambda x. P(x) \wedge Q(x) : (e_1 \multimap t_3) \multimap ((e_3 \multimap t_3) \multimap (e_3 \multimap t_3))}{\lambda Q. \lambda x. \text{see}(x)(\mathbf{j}) \wedge Q(x) : (e_3 \multimap t_3) \multimap (e_3 \multimap t_3)} \multimap_E}{\lambda y. \text{see}(y)(\mathbf{j}) : e_1 \multimap t_3} \multimap_E$$

The first point to note here is that there is no trace in Fig. 6, nor (consequently) any binder for any trace. Semantically, the closest analogue to the trace in the mainstream approach (see (5)) is the auxiliary hypothesis ($y : e_1$) in (43). The analogue to the binder for the trace is the step in the proof at which that hypothesis is discharged, by $\multimap$ introduction. So it's $\multimap$ introduction that guarantees that long-distance dependencies can be interpreted. And of course, $\multimap$ introduction is an inherent part of the logic underlying meaning composition generally.¹⁴

The reason that interpretation is guaranteed to work out is a combination of the logic, and the requirement for indices on features to be identical when those features match. For example, we get the right interpretation for the movement dependencies for the structure shown in Fig. 6 because the index on the feature uwh^* matches that on the feature wh , and the index on the feature uD^* matches that on the feature D . There are *no special compositional principles* for interpreting structures formed by movement. The only part of the grammar that deals especially with movement is the rule for internal merge (35) that makes movement possible. The requirement in that rule that the indices on matching features match is shared by all the other structure-building rules.

What this means is that the system of compositional semantics is unaffected by the choice of formulation of syntactic rule for movement, with the sole caveat that the rule must not duplicate meaning constructors. So for example, if instead of (35) we had a rule that copied the embedded constituent and then deleted the phonological and semantic information in the base position (one way of implementing the ‘copy theory’ of movement), then instead of the structure shown in Fig. 6 we would have the structure shown in Fig. 7. Importantly, the structures shown in Figs. 6 and 7 contribute exactly the same meaning constructors, and so both are interpreted as shown in (43).

4.3.2 Whither LF?

Since the Glue implementation eliminates the need for covert movement, the natural conclusion to draw is that it also, therefore, eliminates the need for an additional level of representation such as LF. This, surely, is a result in the spirit of the Minimalist Program.

Alternatively, one could think of the linear logic proofs themselves as LFs, since every one of them is associated with exactly one interpretation. We therefore would have a notion of Logical Form that is truly (type-)logical.

However, if we do adopt the ‘proofs as LFs’ perspective then we need not, and in fact should not, identify the proofs with the particular natural deduction representations given here, since proofs can be represented in many different ways. As Corbalán and Morrill (2016, fn. 4) put it,

Gentzen calculus, labelled and unlabelled natural deductions, proof nets, categorical calculus, etc. are all of repute, all have their respective advantages and disadvantages, and are all notations for the same theory.

What this means in practice is that, even if we think of the proofs as LFs, we are not free to state arbitrary constraints on the forms of representations of the proofs as part of

¹⁴ With the qualification inherent in Section 4.3.2. Strictly speaking, it's the availability of conditional reasoning that guarantees that long-distance dependencies can be interpreted, and $\multimap$ introduction is how conditional reasoning is implemented in the natural deduction proof format.

Fig. 7: A relative clause: alternative structure

our linguistic theory.¹⁵ And so the Glue approach is in this sense more constrained than the mainstream approach, as we have removed ‘structure at LF’ as a possible locus of explanation for linguistic phenomena.

A further corollary of this perspective is that certain worries that one might have about the structure of LFs turn out to be non-issues in the proofs-as-LFs approach. As an example, consider the inverse-scope interpretation of (11). The LF for this interpretation given in (12-b), and repeated below as (44-a), follows from the assumption that quantified DPs *must* undergo QR. There is an alternative perspective, though, according to which QR is optional. On that assumption, the correct LF for this interpretation would (or could) be as shown in (44-b) below, instead.

¹⁵ Here I part company with some of the LFG literature, in which such constraints on natural deduction derivations have sometimes been proposed in order to account for parallel constraints on scope orderings; some examples are Asudeh and Crouch (2002a,b); Crouch and van Genabith (1999). A linguistic theory should not be forcing us to use natural deduction rather than, say, proof nets to write out our proofs.

(44) a.

b.

Which perspective is right? The natural deduction analogues of (44-a) and (44-b) are shown below as (45-a) and (45-b) (repeated, pared down, from Figure 4), respectively. To make comparison with (44-a)–(44-b) as simple as possible, only linear logic formulae are shown and the steps of $\Box$ -elimination have been elided.

(45) a.

b.

$$\begin{array}{c}
 \text{everyone} \\
 \downarrow \\
 (e_1 \multimap t_3) \multimap t_3 \\
 \hline
 t_3
 \end{array}
 \quad
 \begin{array}{c}
 \text{someone} \\
 \downarrow \\
 (e_2 \multimap t_3) \multimap t_3 \\
 \hline
 t_3
 \end{array}
 \quad
 \begin{array}{c}
 \text{see} \\
 \downarrow \\
 e_1 \multimap (e_2 \multimap t_3) \\
 \hline
 e_2 \multimap t_3
 \end{array}
 \quad
 \begin{array}{c}
 [e_1]^2 \\
 \hline
 \multimap_E
 \end{array}
 \quad
 \begin{array}{c}
 \multimap_E
 \end{array}$$

Crucially, there is a very strong sense in which the deductions shown in (45-a) and (45-b) are actually the same proof. More specifically, the deduction shown in (45-a) is not in normal form;¹⁶ its normal form is the deduction shown in (45-b). The reason is that (45-a) contains the sub-derivation

$$\frac{\frac{e_2 \multimap t_3 \quad [e_2]^1}{t_3} \multimap_E}{e_2 \multimap t_3} \multimap_{I,1}$$

5 Comparison with related approaches

It has already been noted that in Glue, the mapping from syntactic structure to semantic interpretation is relational rather than functional. To this extent, Glue is similar to the ‘storage’ approach set out by Cooper (1983), which has been adapted specifically for Minimalist Grammars in the sense of Stabler (1997) by Kobele (2006, 2012). In the approach taken by Kobele (2006, 62–81), our scopally ambiguous sentence (11) would have a structure like that shown in Figure 8.¹⁸

Informally, the analysis of scope ambiguity is that a moving expression can be interpreted in any of the positions it appears in (provided that that configuration is in fact interpretable). In Figure 8, this allows for the object DP to be interpreted in Spec, vP and the subject DP to be interpreted in its base position, giving us the inverse scope interpretation of the sentence. All other possible configurations give us the surface scope interpretation of the sentence.

However, derived trees such as Figure 8 are not the best vehicles for thinking about the syntax-semantics interface of a Minimalist Grammar in this sense. It is better to use a derivation tree, which shows the history of how an expression is put together. A (simplified) derivation tree for Figure 8 is shown in Figure 9.

¹⁶ I am grateful to an anonymous reviewer for pointing out the relevance of proof normalization for a comparison of proofs with LFs.

¹⁷ One might wonder whether there is an alternative proof format in which it is not even possible to write out two different-looking proofs that are actually the same. The answer is yes: proof nets, which were introduced by Girard (1987) in the course of introducing linear logic. See Moot (2002); Moot and Retoré (2012) for discussion in a linguistic context.

¹⁸ This is a simplification in various ways, some of which will be alluded to, and the node labels have been chosen for the sake of familiarity.

Fig. 9: Derivation tree of (11) according to Kobele (2006)

For the most part, features will be suppressed in the following discussion. The reader can see which constituent is moving at each stage in Figure 9 by comparing it with Figure 8. The important point in the current context is that interpretive rules can be defined over derivation trees like Figure 9.

Before stating those rules, some preliminary remarks on Kobele's system are in order. In this system, instead of taking denotations *relative to* an assignment, assignments are *part of* denotations. Sentence meanings are consequently taken to be sets of verifying assignments. This makes it possible to give a direct denotation of abstraction operators, rather than having to treat them via a syncategorematic rule such as (6). The following abbreviations are adopted:

- $G := D^{\mathbb{N}}$ (the set of assignments)
- $E := D^G$
- For any $n \in \mathbb{N}$, $x_n :=$ the $f \in E$ such that for any $g \in G$, $f(g) = g(n)$.
- For any $n \in \mathbb{N}$, $\lambda_n :=$ the $f \in ((\emptyset G)^E)^{(\emptyset G)}$ such that for any $H \in \emptyset G$, any $a \in E$ and any $g \in G$, $g \in f(H)(a)$ iff $g[n \mapsto a(g)] \in H$.

x s and λ s therefore behave as variables and variable-binders respectively, as the notation suggests. These changes also, of course, mean that we have to ‘lift’ all our other denotations to adjust to the fact that the interpretation of the simplest possible argument is not longer in D but rather in $D^{(D^{\mathbb{N}})} = E$. So for example, instead of

$$\begin{aligned} \llbracket \text{everyone} \rrbracket^g &= \text{the } f \in \{0, 1\}^{\{0, 1\}^D} \text{ such that for any } A \in \{0, 1\}^D, \\ f(A) &= 1 \text{ iff } \{o \in D : o \text{ is a person}\} \subseteq \{o \in D : A(o) = 1\} \end{aligned}$$

we have

$$\begin{aligned} \llbracket \text{everyone} \rrbracket &= \text{the } f \in (\emptyset G)^{(\emptyset G)^E} \text{ such that for any } d \in (\emptyset G)^E, \\ f(d) &= \{g \in G : \{a \in E : a(g) \text{ is a person}\} \subseteq \{a \in E : g \in d(a)\}\} \end{aligned}$$

In order to give a semantics for movement in this system we need more than denotations, however. Semantic values of constituents are ordered pairs $\langle a, b \rangle$, where a is a model-theoretic object and b is a sequence of model-theoretic objects (a **store**). Borrowing notation from Larson and Segal (1995), I will write $Val(m, c)$ to say that m is a semantic value of c . Then, the rules for interpreting derivations can be stated as shown in (47)–(48).

- (47) $Val(x, \text{merge}(\alpha, \beta))$ iff:
- a. $x = \langle y(z), s \cap t \rangle$ and $Val(\langle y, s \rangle, \alpha)$ and $Val(\langle z, t \rangle, \beta)$ (forward application),²⁰ or
 - b. $x = \langle z(y), s \cap t \rangle$ and $Val(\langle y, s \rangle, \alpha)$ and $Val(\langle z, t \rangle, \beta)$ (backward application), or
 - c. For some $i \in \mathbb{N}$, $x = \langle y(x_i), s \cap \langle z \circ \lambda_i \rangle \cap t \rangle$ and $Val(\langle y, s \rangle, \alpha)$ and $Val(\langle z, t \rangle, \beta)$ (storage).
- (48) $Val(x, \text{move}(\alpha))$ iff:
- a. $Val(x, \alpha)$ (identity), or
 - b. $x = \langle Q(y), s \cap t \rangle$ and $Val(\langle y, s \cap Q \cap t \rangle, \alpha)$ (retrieval).

Subject to the constraints that:

- x_i is ‘fresh’ in (47-c).
- In (48-b), Q was contributed to the store by the moving expression.
- Once an expression has finished moving, anything it contributed to the store has been taken out again.

These constraints follow from the precise formulation of the rules (Kobele 2006, 120–132), which tie stores to the features that drive movement.

Suppose that we have the lexical information that

- $Val(\langle \llbracket \text{someone} \rrbracket, \langle \rangle \rangle, \text{someone})$,
- $Val(\langle \llbracket \text{see} \rrbracket, \langle \rangle \rangle, \text{see})$, and
- $Val(\langle \llbracket \text{everyone} \rrbracket, \langle \rangle \rangle, \text{everyone})$

, and suppose furthermore that for every other lexical entry ent used in Figure 9, $Val(\langle I, \langle \rangle \rangle, ent)$, where I is the identity function. Then the surface scope interpretation and the inverse scope interpretation of (11) can be arrived at given the derivation shown in Figure 9 and the interpretive rules given in (47)–(48), as shown below.

²⁰ ‘Forward’ and ‘backward’ in this context refers not to the linear order of constituents, but rather to which constituent bears the feature that the other checks. ‘ $\text{merge}(\alpha, \beta)$ ’ means that α is the head of the structure created, and not necessarily on the left.

Surface scope:

1. $Val(\langle \llbracket see \rrbracket(X_1), \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle), merge(see, everyone))$ (storage)
2. $Val(\langle \llbracket see \rrbracket(X_1), \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle), merge(v, 1))$ (forward application)
3. $Val(\langle \llbracket see \rrbracket(X_1), \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle), move(2))$ (identity)
4. $Val(\langle \llbracket see \rrbracket(X_1)(X_2), \langle \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle, \langle \llbracket someone \rrbracket \circ \lambda_2 \rangle \rangle), merge(3, someone))$ (storage)
5. $Val(\langle \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle(\llbracket see \rrbracket(X_1)(X_2)), \langle \langle \llbracket someone \rrbracket \circ \lambda_2 \rangle \rangle), move(4))$ (retrieval)
6. $Val(\langle \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle(\llbracket see \rrbracket(X_1)(X_2)), \langle \langle \llbracket someone \rrbracket \circ \lambda_2 \rangle \rangle), merge(Prog, 5))$ (forward application)
7. $Val(\langle \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle(\llbracket see \rrbracket(X_1)(X_2)), \langle \langle \llbracket someone \rrbracket \circ \lambda_2 \rangle \rangle), merge(Perf, 6))$ (forward application)
8. $Val(\langle \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle(\llbracket see \rrbracket(X_1)(X_2)), \langle \langle \llbracket someone \rrbracket \circ \lambda_2 \rangle \rangle), merge(-s, 7))$ (forward application)
9. $Val(\langle \langle \llbracket someone \rrbracket \circ \lambda_2 \rangle(\langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle(\llbracket see \rrbracket(X_1)(X_2))), \langle \rangle), move(8))$ (retrieval)
10. $Val(\langle \langle \llbracket someone \rrbracket \circ \lambda_2 \rangle(\langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle(\llbracket see \rrbracket(X_1)(X_2))), \langle \rangle), move(9))$ (identity)

Inverse scope:

1. $Val(\langle \llbracket see \rrbracket(X_1), \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle), merge(see, everyone))$ (storage)
2. $Val(\langle \llbracket see \rrbracket(X_1), \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle), merge(v, 1))$ (forward application)
3. $Val(\langle \llbracket see \rrbracket(X_1), \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle), move(2))$ (identity)
4. $Val(\langle \llbracket someone \rrbracket(\llbracket see \rrbracket(X_1)), \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle), merge(3, someone))$ (backward application)
5. $Val(\langle \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle(\llbracket someone \rrbracket(\llbracket see \rrbracket(X_1))), \langle \rangle), move(4))$ (retrieval)
6. $Val(\langle \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle(\llbracket someone \rrbracket(\llbracket see \rrbracket(X_1))), \langle \rangle), merge(Prog, 5))$ (forward application)
7. $Val(\langle \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle(\llbracket someone \rrbracket(\llbracket see \rrbracket(X_1))), \langle \rangle), merge(Perf, 6))$ (forward application)
8. $Val(\langle \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle(\llbracket someone \rrbracket(\llbracket see \rrbracket(X_1))), \langle \rangle), merge(-s, 7))$ (forward application)
9. $Val(\langle \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle(\llbracket someone \rrbracket(\llbracket see \rrbracket(X_1))), \langle \rangle), move(8))$ (identity)
10. $Val(\langle \langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle(\llbracket someone \rrbracket(\llbracket see \rrbracket(X_1))), \langle \rangle), move(9))$ (identity)

By the definition of function composition,

- $(\llbracket someone \rrbracket \circ \lambda_2)(\langle \llbracket everyone \rrbracket \circ \lambda_1 \rangle(\llbracket see \rrbracket(X_1)(X_2)))$ is equivalent to $\llbracket someone \rrbracket(\lambda_2(\llbracket everyone \rrbracket(\lambda_1(\llbracket see \rrbracket(X_1)(X_2))))$, and
- $(\llbracket everyone \rrbracket \circ \lambda_1)(\llbracket someone \rrbracket(\llbracket see \rrbracket(X_1)))$ is equivalent to $\llbracket everyone \rrbracket(\lambda_1(\llbracket someone \rrbracket(\llbracket see \rrbracket(X_1))))$,

so we have derivations for the surface scope and inverse scope interpretations of the sentence.

As stated above, this theory shares with the Glue approach the property that syntactic structure constrains interpretation without (necessarily) determining it. Moreover, it makes no use of covert movement. However, in this approach interpretation is still tied to structure to an extent that forces decisions about structure that wouldn't otherwise be made. If we look at Figure 8, we can see that both DPs have moved twice after first merging. This in fact is forced by a combination of the formalism and semantic considerations. The object DP has to move to a position higher than the subject's base position in order to be interpretable (Kobele 2006, 77), a semantic consideration. This in turn means that the object DP has to have moved once already before the subject DP merges, since it follows from the SMC that

basically have to be thought up on a case-by-case basis. The application, storage and retrieval rules given in this section are sufficient for simple sentences, but for more complex structures such as those involving inverse linking (see Section 6), topicalization or remnant movement (Kobele 2012, 109–110), additional rules must be stipulated. In contrast, in the Glue approach all the compositional rules needed follow precisely from the underlying logic and the Curry-Howard correspondence.

6 Nesting quantified DPs

In this section I will compare the Glue approach with the mainstream approach, and one recent alternative, with respect to how they handle the interpretation of sentences in which quantified determiner phrases (DPs) are embedded within other DPs. As an example, consider (52).

(52) A fan of every band cheers.

(52) has two interpretations, paraphraseable as shown in (53-a) (the surface scope interpretation) and (53-b) (the inverse-linking interpretation).

- (53) a. There is someone who is a fan of every band and who cheers.
b. For every band, there is someone who is a fan of that band and who cheers.

6.1 Problems with the mainstream approach

Both (53-a) and (53-b) pose problems for the mainstream approach as outlined in Section 2. Let us examine (53-a) first. Heim and Kratzer (1998) present an analysis of nested DPs according to which the LF that gives this interpretation would be as shown in (54).

This analysis involves the postulation of a subject position within NP that is filled by a phonologically and semantically null pronoun (PRO), which then moves at LF. The point to note here is that this kind of manoeuvre is made unavoidable by the approach to the interpretation of scope-taking constituents outlined in Section 2: in order for the DP ‘every band’ to take scope within the NP containing it, that NP has to be made clause-like so that the DP has a node of the appropriate type to adjoin to. This effect is achieved in (54) because, although PRO is semantically null, the trace left behind when it moves isn’t—it is interpreted as an assignment-dependent individual, meaning that the NP containing it is interpreted as an assignment-dependent proposition, and is therefore an appropriate target for QR. But it is very difficult to motivate, on independent grounds, a subject position within NP filled by a phonologically and semantically null pronoun.

Alternatively, one could say that QR is *not* the only means by which scope-taking constituents can be interpreted, and allow some kind of meaning-changing operation (Hendriks 1987) or additional compositional principle (Winter 2001, 131) so that the embedded DP can be interpreted *in situ*. However, once this kind of operation or principle is added to the system then the motivation for QR in general is weakened, as invoking an additional principle or operation leaves QR with one fewer thing to do.

The inverse-linking interpretation of (52) poses a different problem. At a first look, the natural thing to say would be that the LF needed is the one shown in (55) (May 1977).

(55) involves (covert) movement out of the subject DP. Since overt movement from that position is impossible, adopting the analysis shown in (55) requires weakening the analogy between covert and overt movement—and hence the plausibility of covert movement overall, since a significant argument in favour of covert movement is that quantifier scope relationships are supposed to be constrained in the same way that movement operations are (May 1977, §3.4).

A more serious problem with (55) is that, if QR out of the subject DP *is* possible, then there can be nothing to stop a quantified DP in the VP taking scope between the two quantifiers in the subject. For example, (57) should be a possible LF of (56).

(56) A fan of every band sings no songs.

If (57) were a possible LF of (56), then (56) should have an interpretation paraphraseable as indicated in (58). But such an interpretation is not possible.²⁴

(58) For every band, there are no songs that any fan of that band sings.

Faced with this problem, May (1985) proposed that a DP undergoing QR can adjoin to another DP node. That would mean that that LF for the inverse-linking interpretation of (52) would be as shown in (59).

However, on the standard assumptions about what the denotations are, (59) would not be interpretable according to (1) and (6), because (the $f \in \{0,1\}^D$ such that for any $o \in D$, $f(o) = \llbracket a \text{ fan of } t_1 \rrbracket^g[1 \mapsto o]$) is not in the domain of $\llbracket \text{every band} \rrbracket^g$ (nor *vice versa*). Therefore, for the structure shown in (59) to be interpreted requires additional compositional principles and/or meaning-changing operations, which once again reduces the motivation for QR in general.

6.2 Glue analysis

The Glue approach extends without any further stipulations to the analysis of nested DPs, as was anticipated in the earliest papers on quantification in Glue such as Dalrymple et al (1997). Given the lexical items shown in (60), the DP ‘a fan of every band’ can have the structure shown in Fig. 10. From the meaning constructors shown in Fig. 10, both the surface scope and the inverse-linking interpretations of the DP can be deduced. These are shown in Figs. 11 and 12, respectively.

²⁴ This is commonly known as ‘Larson’s generalization’, after Richard Larson. See May and Bale (2007) for discussion.

Fig. 10: The structure of a complex DP

One way of thinking about what's going on in the proof shown in Fig. 11 would be to say that the reason there is no need for a subject position within the NP in the *syntax* (In Fig. 10) is because there is, in effect, a subject position available within the NP interpretation in the *proof*, in the form of an auxiliary hypothesis (in this case, $v : e_3$). Another way of thinking about it would be to say that the additional meaning-changing operation or compositional principle that would be required to interpret everything *in situ* in the mainstream approach follows as a theorem of the underlying logic in the Glue approach.

A very similar point can be made regarding the inverse linking interpretation as shown in Fig. 12. The underlying logic allows for this interpretation to be derived using only premises contributed by words in Fig. 10, i.e. from within the DP. This becomes important when we consider how to state the constraint that DP is a scope island in the current approach, and thereby rule out the interpretation of (56) paraphrased in (58).

6.3 Scope islands

The basic idea behind the treatment of scope islands that I want to defend is as follows. A **scope island** is a constituent that, semantically, behaves like a single lexical item in the

interpretation of structures containing it. Premises contributed by words within the island can't directly enter into proofs with premises contributed by words outside of the island; all the meaning constructors from within the island have to be 'used up' in the derivation of a single meaning constructor (the conclusion of the proof) that represents the semantic contribution of the island as a whole. For example, to say that DP is a scope island is to say that you have to derive a conclusion of type e or $(e \rightarrow t) \rightarrow t$ from all and only the premises available from within DP, and then that conclusion can serve as a premise in the interpretation of the sentence as a whole, in combination with premises contributed by the rest of the sentence. The idea is defined formally in Appendix A.3.

Both the proofs shown in Figs. 11 and 12 meet the requirement of treating the subject DP as a scope island, so defined: they are both proofs to a conclusion of type $(e \rightarrow t) \rightarrow t$ from the meaning constructors shown in Fig. 10.²⁵ However, there is no way to derive a conclusion of type e or $(e \rightarrow t) \rightarrow t$ from those meaning constructors in such a way as to make the interpretation paraphrased in (58) derivable afterwards. Any proof that made that interpretation derivable would have to have a conclusion of higher type, to wit $((e \rightarrow t) \rightarrow t) \rightarrow t$ or higher. There is, therefore, no tension in the Glue framework between the scope island-hood of DP and the possibility of inverse linking. Again, one way of thinking about this result would be to say that the additional meaning-changing operation or compositional principle that would be required to interpret (59) in the mainstream approach follows as a theorem of the underlying logic in the Glue approach.

This way of defining scope islands is conceptually similar to the notion of a 'phase' that has been much discussed in the Minimalist literature. For example, consider this discussion of phases from (Chomsky 2008, 142 f.):

at various stages of computation there are Transfer operations: one hands the SO [syntactic object] already constructed to the phonological component, which maps it to the SM interface ("Spell-Out"); the other hands SO to the semantic component, which maps it to the C-I interface. [...] For minimal computation, as soon as the information is transferred it will be forgotten, not accessed in subsequent stages of derivation

Thereafter, not being 'accessed in subsequent stages of derivation' serves as an explanation for various locality effects. In the present context, we can think of 'hand[ing] SO to the semantic component' as requiring that the meaning constructors contributed by items within that SO serve as premises in a proof. They are 'not accessed' subsequently because the information contained in them is represented in the conclusion of that proof.²⁶

²⁵ Of course, the DP 'every band' should also be a scope island, and this isn't reflected in the proofs shown in Figs. 11 and 12. However, that is easily fixable, as in both cases the meaning constructors contributed by 'every' and 'band' derive an independent sub-proof of the final proof, with a conclusion of type $(e \rightarrow t) \rightarrow t$.

²⁶ I should acknowledge that taking this approach to scope islands essentially complicates the syntax-semantics interface in comparison with what has been assumed so far, since we are now talking about an interpretation of a sentence being mediated not necessarily by a single proof but possibly by an ordered collection of proofs. An alternative worth exploring in order to maintain the single-proof architecture, without attempting to impose constraints on the form of proof representations (as criticized in Section 4.3.2), would be to enrich the fragment of linear logic with unary modal connectives. This approach is commonplace in categorial grammar and is explored in a Glue setting in Gotham (2017).

6.4 On the use of function composition

The Glue approach to nested DPs has some similarities with that of Kobele (2010), which is a QR-based account of scope-taking that nonetheless deviates from the mainstream in the following ways:

1. Instead of taking denotations *relative to* an assignment, assignments are *part of* denotations.
2. The stock of semantic compositional rules includes function composition.

Point 1 is implemented in that, for any $n \in \mathbb{N}$,

- we retain the definitions of x_n and λ_n from Section 5 above, and the accompanying ‘lifting’ of denotations,
- for any trace t_n , $\llbracket t_n \rrbracket = x_n$,
- indices of movement do not occupy their own syntactic positions but rather are attached to the moved constituent, and for any XP, $\llbracket XP_n \rrbracket = \llbracket XP \rrbracket \circ \lambda_n$.

The LF structures for the surface scope reading and the inverse linking reading of (52) would then be as shown in (61) and (62) respectively.

So far, neither structure (61)–(62) would be interpretable: in (61), neither $\llbracket fan\ of\ t_1 \rrbracket$ nor $\llbracket every\ band \rrbracket \circ \lambda_1$ is in the domain of the other, and in (62), neither $\llbracket a\ fan\ of\ t_1 \rrbracket$ nor $\llbracket every\ band \rrbracket \circ \lambda_1$ is in the domain of the other. This is where the significance of point 2 comes in: Kobele argues that functional composition is not restricted to the interpretation of indexed constituents but rather is a generally available semantic compositional principle. With function

composition available, (61) would be interpreted as shown in (63), and (62) and shown in (64).

$$(63) \quad \llbracket a \rrbracket ((\llbracket \text{every band} \rrbracket \circ \lambda_1) \circ (\llbracket \text{fan} \rrbracket (x_1))) (\llbracket \text{cheer} \rrbracket) \\ \equiv \quad \llbracket a \rrbracket (\lambda_2 (\llbracket \text{every band} \rrbracket (\lambda_1 (\llbracket \text{fan} \rrbracket (x_1) (x_2)))) (\llbracket \text{cheer} \rrbracket))$$

$$(64) \quad ((\llbracket \text{every band} \rrbracket \circ \lambda_1) \circ (\llbracket a \rrbracket (\llbracket \text{fan} \rrbracket (x_1)))) (\llbracket \text{cheer} \rrbracket) \\ = \quad (\llbracket \text{every band} \rrbracket \circ \lambda_1) ((\llbracket a \rrbracket (\llbracket \text{fan} \rrbracket (x_1))) (\llbracket \text{cheer} \rrbracket)) \\ = \quad \llbracket \text{every band} \rrbracket (\lambda_1 ((\llbracket a \rrbracket (\llbracket \text{fan} \rrbracket (x_1))) (\llbracket \text{cheer} \rrbracket)))$$

As the steps of reasoning in (64) show, from Kobele's lexical entries and the definition of function composition it follows that the interpretation of (62) with function application and composition available is the same as that of (55) with function application available only. We therefore get the semantic effect of moving *every band* out of the DP without actually having to move it out of the DP, which means that DP can still be an island for movement.

The similarity between this account and the Glue account comes from the fact that under the Curry-Howard correspondence, function composition corresponds to a hypothetical syllogism. It is therefore a theorem of the linear logic fragment used in this paper, as (65) shows.

$$(65) \quad \frac{\frac{h : B \multimap C \quad \frac{f : A \multimap B \quad [x : A]^1}{f(x) : B} \multimap_E}{h(f(x)) : C} \multimap_E}{\lambda x. h(f(x)) : A \multimap C} \multimap_{I,1}$$

There is an important difference between the two accounts, however. As discussed in Section 3, in the Glue approach, labelling of argument positions ensures that argument structure is preserved under operations like (65). By contrast, allowing function composition as a generally-available compositional principle *without* such a constraint has the potential to wreck argument structure. For example, consider the quantifier-in-situ structure shown in (66).

Assume that $\llbracket \text{see} \rrbracket \in ((\wp G)^E)^E$ and $\llbracket \text{everyone} \rrbracket \in (\wp G)^{((\wp G)^E)}$ (as above). Then (66) is interpretable by function composition but the interpretation it gives us, $\llbracket \text{everyone} \rrbracket \circ \llbracket \text{see} \rrbracket$, is *not* the one we want: it has everyone as the *agent* of seeing!

Kobele (2010, 189) is well aware of this problem, remarking that ‘by allowing function composition as a legitimate mode of semantic combination, I am forced to the position that quantifier raising is not ‘type driven’, but is rather syntactic feature driven’. In other words, (66) is ruled out as an LF for syntactic reasons. In a fuller account, we would of course need to know what those reasons are, and how well-motivated they are independently of the desire to rule out an interpretation that would otherwise be predicted. The same point can be made for other examples, such as (67).

(67) Every boy who dances, smiles.

Historically, two different structures have been proposed for relative clause examples like (67), shown below in (68-a–b).

If our compositional rules are limited to function application and abstraction, then only (68-a) is an interpretable structure, and that on the assumption that $\llbracket \text{who dances} \rrbracket \in ((\wp G)^E)^{((\wp G)^E)}$. This leads to the conclusion, for languages where the syntactic evidence is in favour of (68-b), either that some other compositional principle is required for these structures (Bach and Cooper 1978) or that, despite having the structure shown in (68-b) on the surface, they have the structure shown in (68-a) at LF. However, if function composition is available and (as above) $\llbracket \text{who dances} \rrbracket \in ((\wp G)^E)^{((\wp G)^E)}$, then (68-b) is interpretable after all, as shown in (69).

$$(69) \quad ((\llbracket \text{every} \rrbracket (\llbracket \text{boy} \rrbracket)) \circ \llbracket \text{who dances} \rrbracket) (\llbracket \text{smiles} \rrbracket) \\ = \llbracket \text{every} \rrbracket (\llbracket \text{boy} \rrbracket) (\llbracket \text{who dances} \rrbracket (\llbracket \text{smiles} \rrbracket))$$

The problem, though, is that (69) is *not* the right interpretation of (67); it says that every boy both smiles and dances.

Furthermore, generally-available function composition can cause trouble even in some structures that surely can't be disputed. Look at (70).

Let us assume that $\llbracket \text{the} \rrbracket \in E^{((\wp G)^E)}$ and $\llbracket \text{boy} \rrbracket, \llbracket \text{smiles} \rrbracket \in (\wp G)^E$ —which is natural, given Kobele's system. Then, of course, the interpretation $\llbracket \text{smiles} \rrbracket (\llbracket \text{the} \rrbracket (\llbracket \text{boy} \rrbracket))$ would be available as an interpretation, meaning that the unique boy smiles. But if function composition is a generally-available mode of semantic composition, then $(\llbracket \text{boy} \rrbracket \circ \llbracket \text{the} \rrbracket) (\llbracket \text{smiles} \rrbracket)$ would also be available as an interpretation, meaning that the unique smiler is a boy.

There are of course ways out: in this case, we could say that there is a ranking on the compositional rules such that function composition is only available if function application isn't, or that $\llbracket \text{the} \rrbracket \notin E^{((\wp G)^E)}$ after all. But the general point to be taken from these

examples is that if function composition is to be introduced as a compositional principle, it must be accompanied by thoroughly thought-through restrictions on its applicability so as to guarantee that no undesirable consequences emerge. The glue approach provides just such a restriction.

7 Conclusion

I have given an implementation of Glue Semantics for Minimalist syntax, adding to the implementations that Glue has for LFG, HPSG and LTAG. In Glue, the syntax-semantics interface takes the form of deduction in a type logic, specifically a fragment of linear logic, with syntax contributing the premises to the deduction and semantics determined by the Curry-Howard correspondence between proofs and programs. The key aspect of this implementation is the idea that the process—matching of features—that drives syntactic structure-building also provides the connection to linear logic that Glue requires. The implementation dispenses with the need for covert movement in the syntax, at least to the extent that this is motivated by concerns of scope, as different scopings correspond to different proofs from the same premises. It also dispenses with the need for ad-hoc meaning-changing rules or compositional principles. I have shown that in some respects the Glue approach to relative quantifier scope is preferable to the mainstream approach, particularly in cases where one quantified DP is embedded inside another, and in so doing have given a definition of scope islands that is compatible with this implementation. I have also argued that this implementation provides a natural account of the interpretation of structures formed by (overt) movement as well, without the need for traces in the syntax, which is neutral between copy-based and remerge-based conceptions of movement.

Acknowledgements The material in this paper was presented at various stages of development: at the workshop *Interactions between syntax and semantics across frameworks* at University of Cambridge in January 2015 (which appeared as Gotham (2015)), to the international Glue group in cyberspace in February 2015, at the annual meeting of the LAGB in August 2015 and at the University of Oslo Forum for Theoretical Linguistics in October 2016. I want to thank the participants in the these meetings for many helpful suggestions and challenges. I am also grateful to two anonymous reviewers for the improvements that were made to this paper in the review process.

A Definitions

A.1 Linear logic fragment

A.1.1 Formulae

Given a set $\mathbf{C}$ of constants and a set $\mathbf{V}$ of variables, the set $\mathbf{F}$ of formulae of the fragment of linear logic used is defined recursively as follows:

$$\mathbf{F} ::= \mathbf{P}_{\mathbf{C}} \mid \mathbf{P}_{\mathbf{V}} \mid \mathbf{F} \multimap \mathbf{F} \mid \Box \mathbf{V}(\mathbf{F})$$

$$\mathbf{P} ::= e \mid t$$

A.1.2 Glue semantics assignment

A **meaning constructor** is an ordered pair $\langle m, \Phi \rangle$ (written $m : \Phi$), where Φ is a formula of the linear logic fragment and m is an expression of the simply-typed lambda calculus of the type determined by the type map given in (20) in the main paper.

A.2.4 Resolved lexical items

Let $\mathbf{I}$ be a set of indices, identical to the set $\mathbf{C}$ as discussed in A.1.1, and let f be an injective function with domain $\mathbf{XV}$ as discussed in A.2.2 ($=\mathbf{V}$ as discussed in A.1.1), and range $\mathbf{I}$ ($=\mathbf{C}$). Given any lexical item L in which a node labelled by the feature structure FS dominates a node labelled by the ordered pair $\langle P, M \rangle$, a **resolved lexical item** is the result of replacing every index variable i in FS with $f(i)$, and replacing every free variable v on the glue side of every member of M with $f(v)$.

A.2.5 Syntactic structures

Every resolved lexical item is a **syntactic structure**, and the structure-building rules as defined below are all (unary or binary) operations on syntactic structures.

HOPS MERGE

$$\begin{array}{c} \{A_i\} \cup X \\ \Sigma \end{array} + \begin{array}{c} \{B_i\} \cup Y \\ \Gamma \end{array} \Rightarrow \begin{array}{c} \{A_i\} \cup X \\ \Sigma \\ \swarrow \quad \searrow \\ \{A_i\} \cup X \quad \{B_i\} \cup Y \\ \Gamma \end{array}$$

Where A and B are in the same hierarchy of projections (HoPs) and A is higher on that HoPs than B

SELECT MERGE: EXTERNAL

$$\begin{array}{c} X \\ \langle uB_i \rangle \cap \Sigma \end{array} + \begin{array}{c} \{B_i\} \cup Y \\ \Gamma \end{array} \Rightarrow \begin{array}{c} X \\ \Sigma \\ \swarrow \quad \searrow \\ X \quad \{B_i\} \cup Y \\ \langle \#B_i \rangle \quad \Gamma \end{array}$$

SELECT MERGE: INTERNAL

$$\begin{array}{c} X \\ \langle uB^*_i \rangle \cap \Sigma \\ \triangle \\ \{B_i\} \cup Y \\ \Gamma \end{array} \Rightarrow \begin{array}{c} X \\ \Sigma \\ \swarrow \quad \searrow \\ X \quad \langle \#B^*_i \rangle \\ \triangle \\ \{B_i\} \cup Y \\ \Gamma \end{array}$$

Where the re-merged constituent is a maximal projection of the feature B, and subject to locality of matching.

Nothing else is a syntactic structure.

A.3 Interpretations of structures

Given a syntactic structure S :

- Every terminal node in S is a **scope island**.
- Any constituent in S rooted in a maximal projection of D is a **scope island**.
- [any other scope islands]
- For any constituent C and any scope island I in S , C **interpretatively contains** I if and only if
 - C properly contains I , and
 - there is no scope island I' such that C properly contains I' and I' properly contains I .
- For any constituent C in S and meaning constructor $m : \Phi$, $m : \Phi$ is an **interpretation** of C if and only if
 - C is a terminal node and $m : \Phi$ is derivable from the multiset of meaning constructors in C , or
 - C is a non-terminal node and $m : \Phi$ is derivable from a multiset M of meaning constructors, such that there is an bijection f from M to the multiset of scope islands interpretatively contained in C , and for any $x \in M$, x is an interpretation of $f(x)$.
 - $\text{TY}(\Phi)$ is a **licensed type** for C .
- For any constituent C in S , if C is a maximal projection of D then the (only) licensed types for C are e and $(e \rightarrow t) \rightarrow t$.
- [licensed types for any other scope islands]

References

- Adger D (2003) Core Syntax. No. 1 in Core Linguistics, Oxford
- Adger D (2010) A Minimalist theory of feature structure. In: Kibort A, Corbett GG (eds) *Features*, Oxford, pp 185–218
- Asudeh A (2004) Resumption as resource management. PhD thesis, Stanford University
- Asudeh A, Crouch R (2002a) Coordination and parallelism in Glue Semantics. In: Butt M, King TH (eds) *Proceedings of the LFG02 Conference*, CSLI Publications, Stanford, CA, pp 19–39
- Asudeh A, Crouch R (2002b) Derivational parallelism and ellipsis parallelism. In: Mikkelsen L, Potts C (eds) *WCCFL 21 Proceedings*, Cascadia Press, Somerville, MA, pp 1–14
- Asudeh A, Crouch R (2002c) Glue Semantics for HPSG. In: van Eynde F, Hellan L, Beermann D (eds) *Proceedings of the 8th International HPSG Conference*, CSLI Publications, Stanford, CA
- Bach E, Cooper R (1978) The NP-S analysis of relative clauses and compositional semantics. *Linguistics and Philosophy* 2:145–150
- Chierchia G, McConnell-Ginet S (2000) *Meaning and Grammar*, 2nd edn. MIT press, Cambridge, MA
- Chomsky N (1995) *The Minimalist Program*. MIT Press, Cambridge, MA
- Chomsky N (2008) On phases. In: Freidin R, Zubizarreta ML, Otero CP (eds) *Foundational Issues in Linguistic Theory*, pp 133–166
- Church A (1940) A formulation of the simple theory of types. *The Journal of Symbolic Logic* 5(2):56–68
- Cooper R (1983) Quantification and Syntactic Theory. No. 21 in *Studies in Linguistics and Philosophy*, D. Reidel, Dordrecht
- Corbalán ML, Morrill G (2016) Overtly anaphoric control in type logical grammar. In: Foret A, Morrill G, Muskens R, Osswald R, Pogodalla S (eds) *Formal Grammar*, Springer, Berlin, Heidelberg, no. 9804 in *Lecture Notes in Computer Science*, pp 183–199, DOI 10/bv25
- Crouch R, van Genabith J (1999) Context change, underspecification and the structure of Glue language derivations. In: Dalrymple M (ed) *Semantics and Syntax in Lexical Functional Grammar*, MIT Press, Cambridge, MA, pp 117–189
- Crouch R, van Genabith J (2000) Linear logic for linguists, ESSLLI 2000 course notes
- Dalrymple M (2001) *Lexical Functional Grammar*. No. 34 in *Syntax and Semantics*, Emerald, Bingley, UK
- Dalrymple M, Lamping J, Saraswat V (1993) LFG semantics via constraints. In: Krauwer S, Moortgat M, des Tombe L (eds) *EACL 1993*, pp 97–105
- Dalrymple M, Lamping J, Pereira F, Saraswat V (1997) Quantifiers, anaphora, and intensionality. *Journal of Logic, Language, and Information* 6(3):219–273
- Dalrymple M, Gupta V, Lamping J, Saraswat V (1999) Relating resource-based semantics to categorial semantics. In: Dalrymple M (ed) *Semantics and Syntax in Lexical Functional Grammar*, MIT Press, Cambridge, MA, pp 261–280
- Frank A, van Genabith J (2001) GlueTag: Linear logic-based semantics for LTAG—and what it teaches us about LFG and LTAG. In: Butt M, King TH (eds) *Proceedings of the LFG01 Conference*, CSLI Publications, Stanford, CA
- Girard JY (1987) Linear logic. *Theoretical Computer Science* 50(1):1–101, DOI 10.1016/0304-3975(87)90045-4
- Gotham M (2015) Towards Glue semantics for Minimalist syntax. *Cambridge Occasional Papers in Linguistics* 8:56–83
- Gotham M (2017) Glue Semantics and locality. In: *Proceedings of the LFG17 Conference*, forthcoming
- Heim I, Kratzer A (1998) *Semantics in Generative Grammar*. No. 13 in *Blackwell Textbooks in Linguistics*, Wiley-Blackwell, Oxford
- Hendriks H (1987) Type change in semantics. In: Klein E, van Benthem J (eds) *Categories, Polymorphism and Unification*, Instituut voor Taal, Logica en Informatie, Amsterdam, pp 95–119
- Howard W (1980) The formulae-as-types notion of construction. In: Seldin J, Hindley J (eds) *To H.B. Curry*, Academic Press, New York, pp 479–490
- Jäger G (2005) *Anaphora and Type Logical Grammar*. No. 24 in *Trends in Logic*, Springer, Dordrecht
- Kobelev GM (2006) *Generating copies*. PhD thesis, University of California Los Angeles
- Kobelev GM (2010) Inverse linking via function composition. *Natural Language Semantics* 18:183–196, DOI 10.1007/s11050-009-9053-7
- Kobelev GM (2012) Importing Montagovian dynamics into Minimalism. In: Béchet D, Dikovsky A (eds) *Logical Aspects of Computational Linguistics*, Springer, Berlin, Heidelberg, no. 7351 in *Lecture Notes in Computer Science*, pp 103–118
- Kokkonidis M (2008) *First-order Glue*. *Journal of Logic, Language and Information* 17(1):43–68
- Larson R, Segal G (1995) *Knowledge of Meaning*. MIT Press, Cambridge, MA
- May R (1977) *The grammar of quantification*. PhD thesis, Massachusetts Institute of Technology

- May R (1985) Logical Form. No. 12 in *Linguistic Inquiry Monographs*, MIT Press, Cambridge, MA
- May R, Bale A (2007) Inverse linking. In: Everaert M, van Riemsdijk H, Goedemans R, Hollebrandse B (eds) *The Blackwell Companion to Syntax*, vol 2, Blackwell, Oxford, chap 36
- Moot R (2002) Proof nets for linguistic analysis. PhD thesis, University of Utrecht
- Moot R, Retoré C (2012) The Logic of Categorical Grammars. No. 6580 in *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg
- Morrill G (1994) Type Logical Grammar. Kluwer Academic Publishers, Dordrecht, Netherlands
- Prawitz D (1965) Natural Deduction. No. 3 in *Stockholm Studies in Philosophy*, Almqvist & Wicksell, Uppsala, Sweden
- Stabler E (1997) Derivational Minimalism. In: Retoré C (ed) *Logical Aspects of Computational Linguistics*, Springer, Berlin, Heidelberg, no. 1328 in *Lecture Notes in Computer Science*, pp 68–95, DOI 10.1007/BFb0052152
- Winter Y (2001) Flexibility Principles in Boolean Semantics. No. 37 in *Current Studies in Linguistics*, MIT Press, Cambridge, MA
- Zimmermann TE, Sternefeld W (2013) *Introduction to Semantics*. De Gruyter Mouton, Berlin