COPiL CAMBRIDGE OCCASIONAL PAPERS *in* LINGUISTICS
Volume **8**, Article 4: 56–83, 2015 | ISSN 2050-5949

TOWARDS GLUE SEMANTICS FOR MINIMALIST SYNTAX*

MATTHEW GOTHAM
University College London

ABSTRACT Glue Semantics is a theory of the syntax/semantics interface according to which syntax generates premises in a fragment of linear logic, and semantic interpretation proceeds by deduction from those premises. Glue was originally developed within Lexical-Functional Grammar and is now the mainstream view of semantic composition within LFG, but it is in principle compatible with any syntactic framework. In this paper I present an implementation of Glue for Minimalism, and show how it can bring certain advantages in comparison with approaches to the syntax/semantics interface more conventionally adopted.

1 INTRODUCTION

Take a simple example of quantifier scope ambiguity like (1).

(1) A boy trains every dog.

There are various theoretical options available when deciding where the ambiguity of (1) should be located. One option would be to say that that the lexical entry of one (or more) of the words in the sentence is polymorphic (Hendriks 1987). Another would be to say that there is more than one syntactic analysis of (1): either in terms of constituent structure (May 1977) or derivational history (Montague 1973). There is a third option, however, which is to say that the ambiguity resides in the nature of the connection between syntax and semantic composition. This is the approach taken in Glue Semantics.[1]

What the lexical polymorphism and syntactic ambiguity accounts have in common is the view that, given a syntactic analysis of a sentence and a choice

* This paper is based on material that was presented at the event *Interactions between syntax and semantics across frameworks* at Cambridge University on 8 January 2015, and to the Oxford Glue Group on 19 February and 5 March 2015. Thanks to Laura Aldridge, Ash Asudeh, Bob Borsley, Mary Dalrymple, Patrick Elliott, Dag Haug and Mark Steedman for questions, suggestions and comments.

1 To this extent, Glue is similar to the 'storage' approach set out by Cooper (1983).

of lexical semantic entries for the words in it, the way in which the meanings of the words will combine is determined. In contrast, in the Glue approach there are many cases in which lexical semantics and syntactic analysis *constrain* the way in which meanings combine without *determining* it. In the case of (1), an informal statement of what the relevant constraints would be is shown in (2).

(2) Constraints for (1).

a. $[\![loves]\!]$ combines with X, then with Y, to form Z.

b. $[\![a]\!]$ combines with $[\![boy]\!]$, then with (something that combines with Y to form Z), to form Z.

c. $[\![every]\!]$ combines with $[\![dog]\!]$, then with (something that combines with X to form Z), to form Z.

There are two ways to put $[\![some]\!]$, $[\![boy]\!]$, $[\![trains]\!]$, $[\![every]\!]$ and $[\![dog]\!]$ together, while satisfying constraints (2a)–(2c), to form Z. Those two ways give the two different interpretations of (1): namely, the surface scope interpretation and the inverse scope interpretation. Formally, the constraints correspond to premises in a fragment of linear logic, and the different ways of putting the meanings together while satisfying those constraints correspond to different proofs from those premises.

Glue was originally developed within Lexical-Functional Grammar (Dalrymple 1999) and is now the mainstream view of the syntax/semantics interface within LFG. However, it is in principle compatible with any syntactic framework. Asudeh & Crouch (2002) and Frank & van Genabith (2001) have defined Glue for Head-driven Phrase Structure Grammar (HPSG) and Lexicalized Tree-Adjoining Grammar (LTAG), respectively. However, to my knowledge no implementation yet exists for Minimalist syntax. In this paper I will define an implementation for Minimalism and show how it has certain advantages in comparison with other approaches to the syntax/semantics interface.

The rest of this paper is structured as follows. In Section 2 I introduce a fragment of linear logic, and explain how deduction in this fragment can be connected to meaning composition in Glue Semantics. In Section 3 I lay out the assumptions that I make about the form of syntactic theory involved and provide an implementation of Glue in Minimalism given those assumptions. In Section 4 I compare this treatment of semantic composition with more conventional views, in particular with quantifier-raising-based accounts, and then go on to discuss how constraints on scope-taking that have been noted in the literature can be implemented in this system. Section 5 concludes.

2 Linear logic and Glue

2.1 Linear logic

Linear logic (Girard 1987) is often called a 'logic of resources' (Crouch & van Genabith 2000: p. 5). Perhaps the best way to explain what is meant by this is by comparison with classical logic.

Sequent (3) is valid in classical logic, as shown by the natural deduction proof in (4).

(3) $A \to (B \to C), A \to B \vdash A \to C$

(4)
$$\dfrac{\dfrac{A \to (B \to C) \quad [A]^1}{B \to C}\,{\to_E} \quad \dfrac{A \to B \quad [A]^1}{B}\,{\to_E}}{\dfrac{C}{A \to C}\,{\to_I}^1}\,{\to_E}$$

Proof (4) has the property that in the final step two instances of the same hypothesis are discharged. In the sequent calculus, this property of the proof is reflected explicitly by the use of the structural rule of contraction, as shown in (5).

(5)
$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,\text{Axiom} \quad \dfrac{\dfrac{}{B \vdash B}\,\text{Axiom} \quad \dfrac{}{C \vdash C}\,\text{Axiom}}{B \to C, B \vdash C}\,{\to_L}}{A \to (B \to C), A, B \vdash C}\,{\to_L}}{A \to (B \to C), B, A \vdash C}\,\text{Permutation}_L}{A \to (B \to C), A \to B, A, A \vdash C}\,{\to_L}}{A \to (B \to C), A \to B, A \vdash C}\,\textbf{Contraction}_L}{A \to (B \to C), A \to B \vdash A \to C}\,{\to_R}$$

In other words, in classical logic, if you have a premise then you may use that premise as many times as you like in your proof.

Sequent (6) is also valid in classical logic.

(6) $P, Q \vdash Q$

In (6), one of the premises is not used in deriving the conclusion. In the sequent calculus proof shown in (7), this fact is reflected explicitly by the use of the structural rule of weakening.

(7)
$$\dfrac{\dfrac{}{Q \vdash Q}\,\text{Axiom}}{P, Q \vdash Q}\,\textbf{Weakening}_L$$

In classical logic, you need not use all the premises you have.

Linear logic (LL) does not have the structural rules of weakening or contraction, and so neither (3) nor (6) is valid in LL. Linear logic keeps a strict accounting of the number of times a premise is used in a proof;[2] premises can be viewed as resources that are used up by being involved in inferential steps. This makes it an ideal logic for semantic composition, because in computing the meaning of a sentence based on the meanings of the words in it, each word meaning must be used (no weakening) and used only once (no contraction) (Asudeh 2004: Ch. 3).

2.2 Interpretation as deduction

In practice, no Glue implementation uses more than a small fragment of linear logic. The only connective of propositional LL to be discussed in this paper is $\multimap$ (linear implication), and in the extension to first-order LL only $\forall$ will be added.

The natural deduction elimination and introduction rules for $\multimap$ are as shown in (8) and (9), respectively.

(8) $\multimap$ elimination (linear modus ponens)

$$\frac{A \multimap B \qquad A}{B} \multimap_E$$

(9) $\multimap$ introduction (linear conditional proof)

$$\begin{array}{c} [A]^n \\ \vdots \\ B \end{array} \over {A \multimap B} \multimap_I{}^n$$

The essence of Glue Semantics is that expressions of a meaning language (in this case, the lambda calculus) are paired with formulae in a fragment of linear logic (the glue language), and that steps of deduction carried out using those formulae correspond to operations performed on the meaning terms. For the first fragment of LL that we will consider, the appropriate correspondence is shown in Table 1.[3] In what follows I will sometimes refer to a pairing of a meaning m with a linear logic formula A, shown as $m : A$, as a 'meaning

2 But it doesn't care about how they are ordered, because it has the structural rule of permutation. If you take a fragment of LL and remove that rule, then you (more or less) end up with the Lambek Calculus **L**, which underlies much work done in categorial grammar in the type-logical tradition (Morrill 1994, 2011, Carpenter 1998).

3 By 'implicational proposition' I mean a proposition that has (linear) implication as the main connective.

	LL	λ calculus	
propositions as types	implicational proposition	functional type	
rules as operations	$\multimap$ elimination	application	$\dfrac{f : A \multimap B \qquad x : A}{f(x) : B}\ \multimap_E$
	$\multimap$ introduction	abstraction	$\dfrac{\begin{array}{c}[x : A]^n \\ \vdots \\ f : B\end{array}}{\lambda x.f : A \multimap B}\ \multimap_I{}^n$

Table 1 Curry-Howard correspondence for the implicational fragment of propositional linear logic

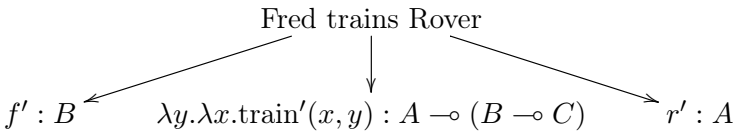
constructor'.

As a simple first example, let us consider (10).

(10) Fred trains Rover.

The job of a Glue implementation is to specify how syntax pairs the meanings of the words in (10) with LL formulae. The implementation proposed in this paper is in Section 3.2. For now, let us assume the pairing shown in (11).

(11)



Given the premises that are produced, the sentence is interpreted by deduction as shown in (12). Note that the form of the premises requires 'Rover' to be the object of 'trains' (A), and requires 'Fred' to be the subject of 'trains' (B).

(12)
$$\dfrac{\dfrac{\lambda y.\lambda x.\text{train}'(x, y) : A \multimap (B \multimap C) \quad r' : A}{\lambda x.\text{train}'(x, r') : B \multimap C}\ \multimap_E \qquad f' : B}{\text{train}'(f', r') : C}\ \multimap_E$$

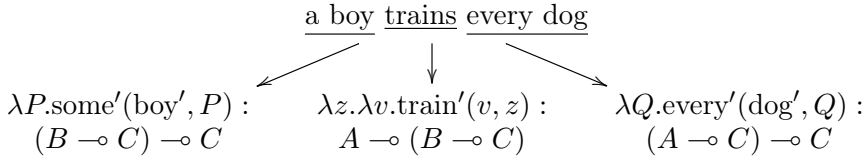
As stated in Table 1, each step of $\multimap$ elimination on the LL side corresponds to a step of function application on the λ calculus side. By deduction from the premises we thereby end up at the correct interpretation of (10).

Now let us consider again our initial example of quantifier scope ambiguity.

(1) A boy trains every dog.

The premises produced will be as shown in (13). Note that the form of the premises requires 'every dog' to be the object of 'trains' (A), and requires 'a boy' to be the subject of 'trains' (B).[4]

(13)



This time there are two distinct proofs that can be computed from these premises resulting in C as the conclusion. One proof gives the surface scope interpretation of (1), and the other gives the inverse scope interpretation of (1). These are shown in (14) and (15), respectively.

(14)

$$\dfrac{\lambda P.\text{some}'(\text{boy}', P) : (B \multimap C) \multimap C \qquad \dfrac{\dfrac{\dfrac{\dfrac{\lambda z.\lambda v.\text{train}'(v, z) : A \multimap (B \multimap C) \quad [y : A]^1}{\lambda v.\text{train}'(v, y) : B \multimap C} \multimap_E \quad [x : B]^2}{\text{train}'(x, y) : C} \multimap_E}{\lambda y.\text{train}'(x, y) : A \multimap C} \multimap_I{}^1 \qquad \lambda Q.\text{every}'(\text{dog}', Q) : (A \multimap C) \multimap C}{\text{every}'(\text{dog}', (\lambda y.\text{train}'(x, y))) : C} \multimap_E}{\lambda x.\text{every}'(\text{dog}', (\lambda y.\text{train}'(x, y))) : B \multimap C} \multimap_I{}^2}{\text{some}'(\text{boy}', \lambda x.\text{every}'(\text{dog}', \lambda y.\text{train}'(x, y))) : C} \multimap_E$$

(15)

$$\dfrac{\dfrac{\lambda P.\text{some}'(\text{boy}', P) : (B \multimap C) \multimap C \qquad \dfrac{\lambda z.\lambda x.\text{train}'(x, z) : A \multimap (B \multimap C) \quad [y : A]^1}{\lambda x.\text{train}'(x, y) : B \multimap C} \multimap_E}{\text{some}'(\text{boy}', \lambda x.\text{train}'(x, y)) : C} \multimap_E}{\lambda y.\text{some}'(\text{boy}', \lambda x.\text{train}'(x, y)) : A \multimap C} \multimap_I{}^1 \qquad \lambda Q.\text{every}'(\text{dog}', Q) : (A \multimap C) \multimap C}{\text{every}'(\text{dog}', \lambda y.\text{some}'(\text{boy}', \lambda x.\text{train}'(x, y))) : C} \multimap_E$$

As stated in Table 1, each step of $\multimap$ elimination on the LL side corresponds to a step of function application on the λ calculus side, and also each step of

4 For simplicity's sake I omit discussion of how $[\![a\ boy]\!]$ is put together from $[\![a]\!]$ and $[\![boy]\!]$, for example.

$\multimap$ introduction on the LL side corresponds to a step of lambda abstraction on the λ calculus side.

3 Minimalist syntax and Glue

In this section I present a toy grammar to serve as the formalization of Minimalism (in Section 3.1), and then give an implementation of Glue for that grammar (in Section 3.2). The toy grammar is based on ideas gleaned from Adger (2003, 2010), Stabler (1997) and Retoré & Stabler (2004).

This implementation is based on the idea, common in the Minimalist literature, that the syntactic structure-building operations (merge, move, agree) are driven by the matching of syntactic features introduced by lexical items. The connection to Glue is made by (tokens of) the features themselves bearing indices that also have to match for structure to be built, and those indices feeding into the meaning constructors for each lexical item.

3.1 The feature system

3.1.1 Syntactic features

I assume a set of features as syntactic primitives, with the following properties:

- Every feature is specified for interpretability, either *inpretable* or *uninterpretable*. In the latter case it is shown with the prefix '*u*', so for example if D is an (interpretable) determiner feature, then *u*D is an uninterpretable determiner feature.
- Every uninterpretable feature is specified for strength, either *weak* or *strong*. In the latter case it is shown with the suffix '*', so for example if *u*D is a (weak) uninterpretable determiner feature, then *u*D* is a strong uninterpretable determiner feature.
- The set of *categorial* features is a proper subset of the set of features. The categorial features are N(oun), V(erb), D(eterminer), A(djective), P(reposition), C(omplementizer) and T(ense),[5] and their (weak or strong) uninterpretable counterparts.

3.1.2 Hierarchy of projections

Every interpretable categorial feature belongs to at most one hierarchy of projections (HoPs). The hierarchies used are:[6]

5 This list is not supposed to be exhaustive, but is sufficient for the fragment considered in this paper.

6 This is a stripped-down version of the hierarchies found in (Adger 2003):

Clausal: C > T > V
Nominal: D > N

3.1.3 Lexical items

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 categorial feature.
- B is a (possibly empty) sequence of uninterpretable features.

A lexical item is a two-node tree in which a feature structure dominates a phonological form. So here are some possible lexical items:

$\langle\{\text{V}\}, \langle u\text{D}, u\text{3rd}, u\text{sg}, u\text{pres}\rangle\rangle$ | smiles	which I'll often represent as:	V $\langle u\text{D}, u\text{pres}, u\text{3rd}, u\text{sg}\rangle$ | smiles
$\langle\{\text{T}, \text{pres}\}, \langle u\text{D*}\rangle\rangle$ | ϵ	(phonologically empty), often:	T[pres] $\langle u\text{D*}\rangle$
$\langle\{\text{D}, \text{sg}, \text{3rd}\}, \langle\rangle\rangle$ | every	often:	D[sg,3rd] | every

The form of representation on the right hand side is chosen so as to highlight the single interpretable categorial feature in each feature structure. Formally, however, the structures are exactly as shown on the left hand side.

3.1.4 Structure-building operations

The core structure-building operation is *Merge*. It comes in different forms:

- Hierarchy of Projections-driven. (HoPs merge)
- Selection-driven.
  - External. (external merge)

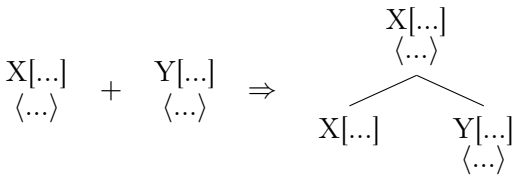
Clausal: C > T > (Neg) > (Perf) > (Prog) > (Pass) > v > V
Nominal: D > (Poss) > n > N
Adjectival: (Deg) > A

– Internal. (internal merge, remerge or move)[7]

I'll also assume an operation *Agree*.

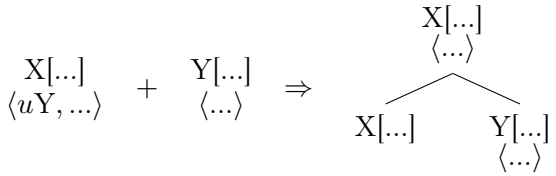
Each of the structure-building operations takes one or two graphs as input and produces a graph as output.[8] The rules in (16)–(19) are to be read as saying that if you have (a) graph(s) rooted in the feature structure(s) shown on the input side, then you can combine them in the way shown on the output 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 by a separate module.

(16) HoPs merge



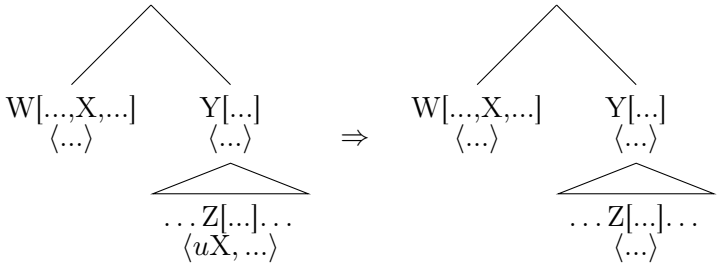
Where X and Y are in the same hierarchy of projections (HoPs) and X is immediately higher on that HoPs than Y

(17) Select merge: External



In this case, the uninterpretable Y feature is no longer present in the output structure, and the rest of the uninterpretable features introduced by the X node are shifted up to its mother.

(18) Agree



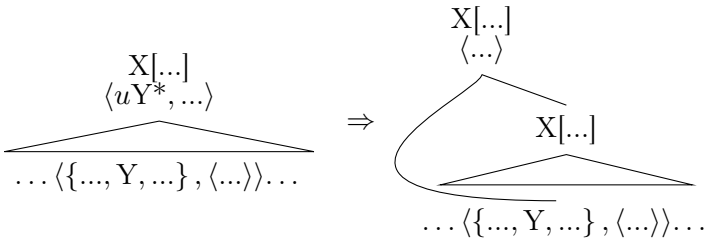
What this diagram is supposed to represent is that the node containing the interpretable X feature c-commands the node containing the uninterpretable X

7 I use these designations interchangeably.

8 Specifically, each of the structures is a directed acyclic graph (DAG) with exactly one root (a node with no mother). They are not trees because a node may have more than one mother.

feature. A node A c-commands a node B iff A's sister dominates B (on the understanding that dominance is reflexive). The uninterpretable X feature is no longer present in the output structure.

(19) Select merge: Internal



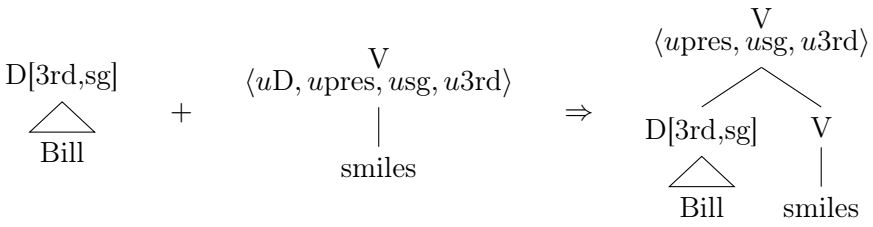
Where the re-merged constituent is a maximal projection of the feature Y.

As the diagram indicates, I understand 'movement' as the creation of structures of multidominance.[9] Note that there is nothing here to say that Y is (or is not) a categorial feature.[10]

3.1.5 An example derivation

I will now show how the rules given in (16)–(19) conspire to generate a derivation for the simple sentence 'Bill smiles'.

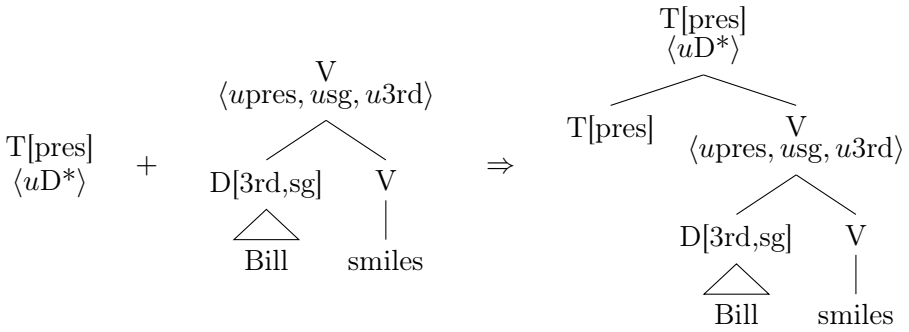
External merge:



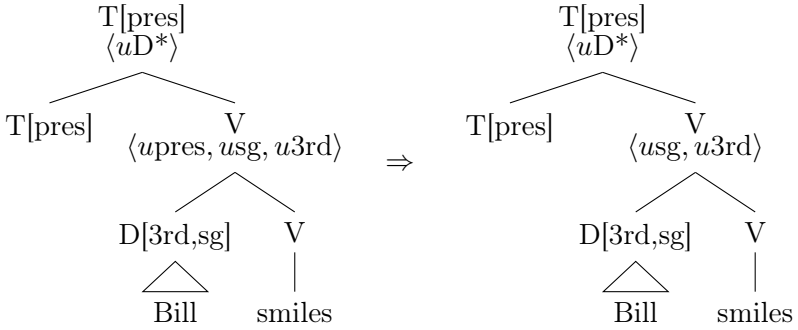
9 This is not crucial, but it is the simplest way to ensure that no duplication of semantic resources is caused by any process of copying in the syntax.

10 Agree and internal merge should also incorporate some principle to the effect that the constituent bearing the uninterpretable feature uF is closer to the one bearing the interpretable feature F than any other constituent bearing uF, for example the 'Locality of Matching' principle given by Adger (2003: 218).

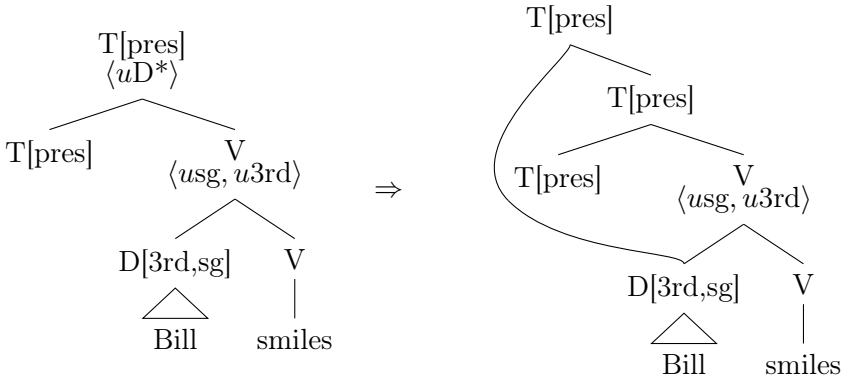
HoPs merge:



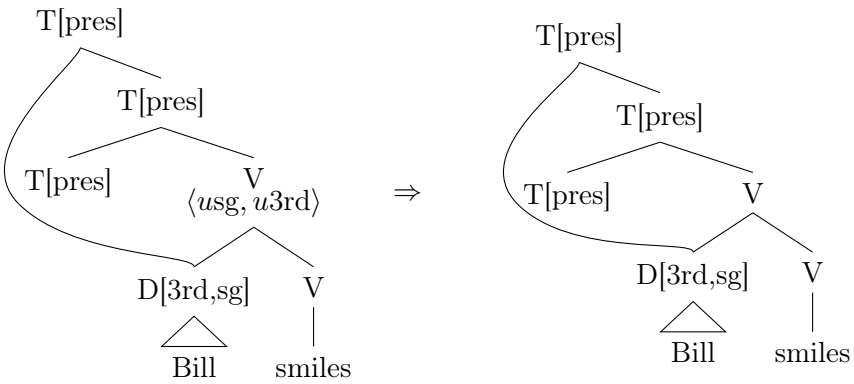
Agree:



Internal merge:



Agree ×2*:*



Note that I am assuming VP-internal subjects. In the final structure there are no uninterpretable features left on any node.

3.2 Implementation

3.2.1 The glue language

In this section I will show how the features introduced in the previous section can be used to define the connection between lexical items and their meaning constructors.

Following Kokkonidis (2008), a fragment of (monadic) first-order linear logic will be used as the glue language (GL). The reason that a first-order system is needed will become clear shortly.

The fragment has two predicates: e and t. I will represent the constants of the GL with natural numbers, and the variables of the GL with uppercase letters. In addition to the connective $\multimap$ that we have already seen, the fragment contains the universal quantifier $\forall$. So for example, $e(1)$, $e(Y)$, $t(2)$ and $\forall X(e(X) \multimap t(X))$ are well-formed formulae.

There is a type map $\textsc{Ty}$ between meanings and formulae in the glue language such that, if $m : g$ is a meaning constructor, then m is of type $\textsc{Ty}(g)$. The type map is shown in (20).

(20) Type map $\textsc{Ty}$ for GL

a. For any terms α and β:
   (i) $\textsc{Ty}(t(\alpha)) = t$
   (ii) $\textsc{Ty}(e(\alpha)) = e$
   (iii) $\textsc{Ty}(\alpha \multimap \beta) = \textsc{Ty}(\alpha) \rightarrow \textsc{Ty}(\beta)$

b. For any formula Φ and any variable X:
   (i) $\textsc{Ty}(\forall X \Phi) = \textsc{Ty}(\Phi)$

	LL	λ calculus	
propositions as types	implicational proposition	functional type	
rules as operations	$\multimap$ elimination	application	$\dfrac{f : A \multimap B \qquad x : A}{f(x) : B}\ \multimap_E$
	$\multimap$ introduction	abstraction	$\dfrac{\begin{array}{c}[x : A]^n \\ \vdots \\ f : B\end{array}}{\lambda x.f : A \multimap B}\ \multimap_I{}^n$
	$\forall$ elimination	—	$\dfrac{f : \forall X.A}{f : A[X \leftarrow c]}\ \forall_E$ c free for X
	$\forall$ introduction	—	$\dfrac{f : A}{f : \forall X.A}\ \forall_I$ X not free in any open leaf

Table 2 Curry-Howard correspondence for the $(\multimap, \forall)$ fragment of first-order linear logic

So for example, if we have the meaning constructor $f : e(4) \multimap t(4)$, then the type of f is $e \to t$. Note that there is a one-to-one correspondence between the GL predicates and basic semantic types. For more complex analyses it may become necessary to add more predicates and make this correspondence many-to-one, but for the current fragment this is not required.[11]

We will need an additional rule of inference for the quantifier in addition to those shown in Table 1. The rule for $\forall$ elimination is shown in (21).[12] Note that, unlike the rules shown in Table 1, no operation is carried out on the meaning side. The updated Curry-Howard correspondence for the fragment of linear logic used is shown in Table 2.

(21) $$\frac{f : \forall X.A}{f : A[X \leftarrow c]}\ \forall_E \quad c \text{ free for } X$$

11 Thanks to Dag Haug for clarification on this point.

12 $\forall$ introduction will not be used in this paper, but it is included in Table 2 for the sake of completeness.

3.2.2 Connecting lexical items to premises

We now come to the point at which lexical entries will be connected with linear logic formulae. The first step is to associate a numerical index with every feature (interpretable or uninterpretable) in lexical entries as described in Section 3.1. The second step is to share those indices with the appropriate terms in linear logic formulae in the semantic side of those lexical entries. This principle is best illustrated with an example. The proposed lexical entry for the verb form 'trains' is shown in (22).

(22)

	trains
Syntax	$\langle \{\mathrm{V}_i\}, \langle u\mathrm{D}_j, u\mathrm{D}_k, u\mathrm{pres}_l, u3\mathrm{rd}_m, u\mathrm{sg}_n \rangle \rangle$
Semantics	$\lambda x.\lambda y.\mathrm{train}'(y, x) : e(j) \multimap (e(k) \multimap t(i))$
	$i \neq j \neq k$

i, j, k etc. are variables over indices. They can be instantiated to any value, subject to constraints listed in the lexical entry (in this case, i, j and k must be instantiated to distinct values). In (22), the sharing of the index variable j between the first uninterpretable D feature and the argument of the e predicate in the antecedent in the linear logic formula ensures that the first DP merged with the verb will be interpreted as the object, and similarly the sharing of the index k ensures that the second DP merged with the verb will be interpreted as the subject.

Some other example lexical entries are shown in (24)–(29). These all adhere to general constrains on index variables described in (23).

(23) In any lexical item:

a. The index variables on all interpretable features must be identical.

b. The index variables on all uninterpretable features must be distinct, and distinct from the index variable on the interpretable features.

(24)

	every
Syntax	$\langle \{\mathrm{D}_i, \mathrm{sg}_i, 3\mathrm{rd}_i\}, \langle\rangle \rangle$
Semantics	$\lambda P.\lambda Q.\mathrm{every}'(P, Q) :$
	$(e(i) \multimap t(i)) \multimap \forall X((e(i) \multimap t(X)) \multimap t(X))$

(25)

	boy
Syntax	$\langle \{\mathrm{N}_i\}, \langle u\mathrm{sg}_j \rangle \rangle$
Semantics	$\lambda x.\mathrm{boy}'(x) : e(i) \multimap t(i)$

(26)

	Bill
Syntax	$\langle \{\mathrm{N}_i\}, \langle u\mathrm{sg}_j \rangle \rangle$
Semantics	$b'(x) : e(i)$

(27)

	smiles
Syntax	$\langle \{ \mathrm{V}_i \}, \langle u\mathrm{D}_j, u\mathrm{pres}_k, u3\mathrm{rd}_l, u\mathrm{sg}_m \rangle \rangle$
Semantics	$\lambda x.\mathrm{smile}'(x) : e(j) \multimap t(i)$
	$i \neq j$

(28)

	ϵ
Syntax	$\langle \{ \mathrm{T}_i, \mathrm{pres}_i \}, \langle u\mathrm{D}^*_j \rangle \rangle$
Semantics	—

(29)

	ϵ
Syntax	$\langle \{ \mathrm{T}_i, \mathrm{past}_i \}, \langle u\mathrm{D}^*_j \rangle \rangle$
Semantics	$\lambda p.\mathrm{past}'(p) : t(i) \multimap t(i)$

The structure-building operations described in Section 3.1.4 are sensitive to indices in the general sense that if two features are required to match, then their indices are required to be identical.

(30) Index matching in structure-building operations.

a. In HoPs Merge as shown in (16), the indices on X and Y must be identical.

b. In External Merge as shown in (17), the indices on uY and Y must be identical.

c. In Agree as shown in (18), the indices on X and uX must be identical.

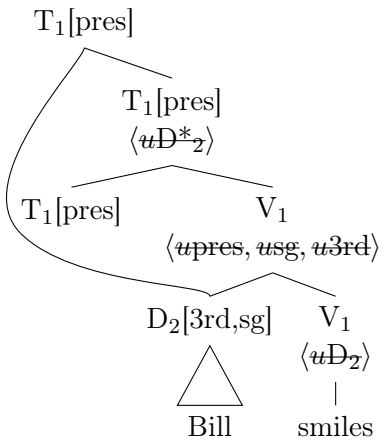
d. In Internal Merge as shown in (19), the indices on uY* and Y must be identical.

By way of an example, I repeat the derived structure for 'Bill smiles' shown in Section 3.1.5 above as (31) below, with the following additions:

i. Indices that have semantic relevance are shown.

ii. Eliminated uninterpretable features are shown struck out at the point at which they are eliminated, in order to show derivational history.

(31)

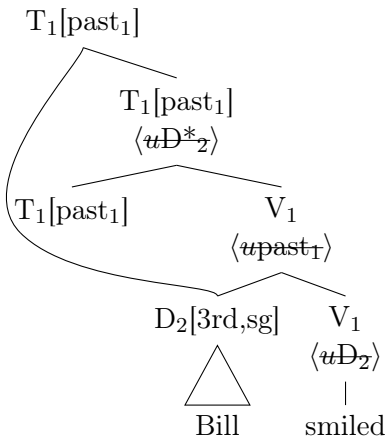


Note that the index on D matches the index on uD and on uD*, as required by (30b) and (30d). Likewise, the index on V matches the index on T, as required by (30a). Given this instantiation of the index values, and the lexical entries for 'Bill' and 'smiles', the sentence is interpreted as shown in (32) below.

(32) $$\frac{\lambda x.\text{smile}'(x) : e(2) \multimap t(1) \qquad b' : e(2)}{\text{smile}'(b') : t(1)} \multimap_E$$

The derived structure for 'Bill smiled' would be as shown in (33), and the interpretation therefore as shown in (34). To save space and improve readability, from now on I will use subscript notation in the LL formulae, e.g. instead of $e(2) \multimap t(1)$ I will write $e_2 \multimap t_1$ (Kokkonidis 2008: p. 53). I will also sometimes omit the '$\multimap_E$' annotation in proofs.

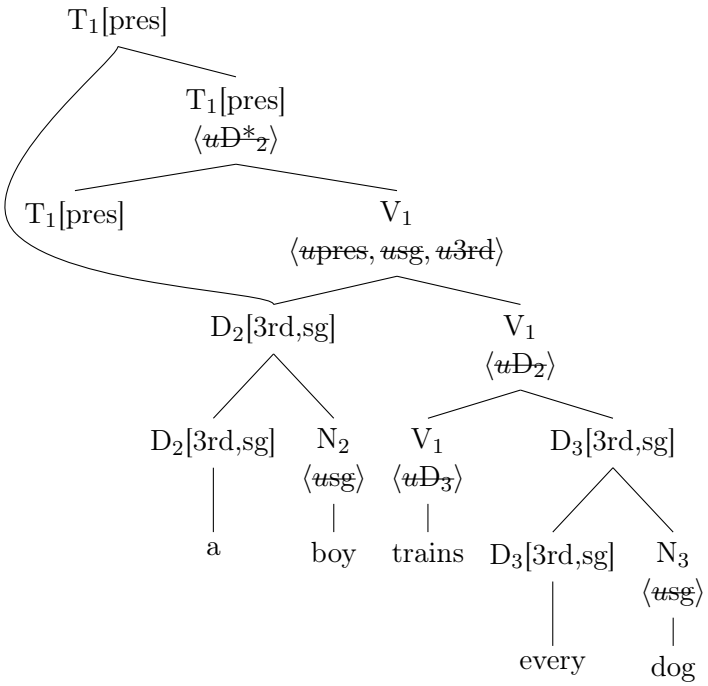
(33)



(34)
$$\dfrac{\lambda p.\text{past}'(p) : t_1 \multimap t_1 \qquad \dfrac{\lambda x.\text{smile}'(x) : e_2 \multimap t_1 \qquad b' : e_2}{\text{smile}'(b') : t_1}\ \multimap_E}{\text{past}'(\text{smile}'(b')) : t_1}\ \multimap_E$$

Now let us look again at quantifier scope ambiguity. The derived structure for (1) would be as shown in (35) below.

(35)



Instantiation of indices on the lexical entries produces the multiset of premises shown in (36).

(36) Premises for the interpretation of (1).

a. $\lambda P.\lambda Q.\text{some}'(P, Q) : (e_2 \multimap t_2) \multimap \forall X((e_2 \multimap t_X) \multimap t_X)$
b. $\text{boy}' : e_2 \multimap t_2$
c. $\lambda x.\lambda y.\text{train}'(y, x) : e_3 \multimap (e_2 \multimap t_1)$
d. $\lambda F.\lambda G.\text{every}'(F, G) : (e_3 \multimap t_3) \multimap \forall Y((e_3 \multimap t_Y) \multimap t_Y)$
e. $\text{dog}' : e_3 \multimap t_3$

The DPs 'a boy' and 'every dog' are interpreted as shown in (37) and (38), respectively. Note that the proofs make use of $\forall$ elimination, and in both cases

the t variable is instantiated to 1.

(37)
$$\dfrac{\dfrac{\lambda P.\lambda Q.\text{some}'(P,Q) : (e_2 \multimap t_2) \multimap \forall X((e_2 \multimap t_X) \multimap t_X) \qquad \text{boy}' : e_2 \multimap t_2}{\lambda Q.\text{some}'(\text{boy}', Q) : \forall X((e_2 \multimap t_X) \multimap t_X)}\ \multimap_E}{\lambda Q.\text{some}'(\text{boy}', Q) : (e_2 \multimap t_1) \multimap t_1}\ \forall_E$$

(38)
$$\dfrac{\dfrac{\lambda F.\lambda G.\text{every}'(F,G) : (e_3 \multimap t_3) \multimap \forall Y((e_3 \multimap t_Y) \multimap t_Y) \qquad \text{dog}' : e_3 \multimap t_3}{\lambda G.\text{every}'(\text{dog}', G) : \forall Y((e_3 \multimap t_Y) \multimap t_Y)}\ \multimap_E}{\lambda G.\text{every}'(\text{dog}', G) : (e_3 \multimap t_1) \multimap t_1}\ \forall_E$$

We can now derive the surface scope and inverse scope readings of (1), shown in (39) and (40), respectively. With e_3 in place of A, e_2 in place of B and t_1 in place of C, these are identical to the proofs shown in (14) and (15).

(39)
$$\dfrac{\lambda Q.\text{some}'(\text{boy}', Q) : (e_2 \multimap t_1) \multimap t_1 \qquad \dfrac{\dfrac{\dfrac{\dfrac{\lambda z.\lambda v.\text{train}'(v,z) : e_3 \multimap (e_2 \multimap t_1) \quad [y : e_3]^1}{\lambda v.\text{train}'(v,y) : e_2 \multimap t_1} \quad [x : e_2]^2}{\text{train}'(x,y) : t_1}}{\lambda y.\text{train}'(x,y) : e_3 \multimap t_1}\ \multimap_I{}^1 \qquad \lambda G.\text{every}'(\text{dog}', G) : (e_3 \multimap t_1) \multimap t_1}{\text{every}'(\text{dog}', (\lambda y.\text{train}'(x,y))) : t_1}}{\lambda x.\text{every}'(\text{dog}', (\lambda y.\text{train}'(x,y))) : e_2 \multimap t_1}\ \multimap_I{}^2}{\text{some}'(\text{boy}', \lambda x.\text{every}'(\text{dog}', \lambda y.\text{train}'(x,y))) : t_1}$$

(40)
$$\dfrac{\dfrac{\lambda Q.\text{some}'(\text{boy}', Q) : (e_2 \multimap t_1) \multimap t_1 \qquad \dfrac{\lambda z.\lambda x.\text{train}'(x,z) : e_3 \multimap (e_2 \multimap t_1) \quad [y : e_3]^1}{\lambda x.\text{train}'(x,y) : e_2 \multimap t_1}}{\text{some}'(\text{boy}', \lambda x.\text{train}'(x,y)) : t_1}}{\lambda y.\text{some}'(\text{boy}', \lambda x.\text{train}'(x,y)) : e_3 \multimap t_1}\ \multimap_I{}^1 \qquad \dfrac{\lambda G.\text{every}'(\text{dog}', G) : (e_3 \multimap t_1) \multimap t_1}{\text{every}'(\text{dog}', \lambda y.\text{some}'(\text{boy}', \lambda x.\text{train}'(x,y))) : t_1}$$

We therefore have two distinct interpretations for (1), even though the sentence was given a single syntactic analysis (shown in (35)) and no lexical polymorphism was postulated.

4 Degrees of freedom in scope-taking

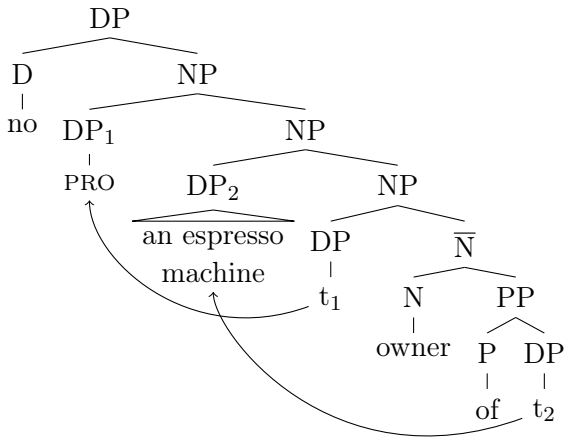
4.1 Taking scope within non-clausal categories

The Glue approach is particularly useful when it comes to the interpretation of sentences in which a quantified DP (QP) is embedded inside another QP. For example, consider (41) (Heim & Kratzer 1998: 229).

(41) No owner of an espresso machine drinks tea.

In order to derive the surface scope interpretation of (41) in an approach based on quantifier raising (QR), Heim & Kratzer (1998: § 8.5.3) postulate the existence of a subject position within the NP that is filled by a phonologically and semantically null pronoun PRO, so that the DP as a whole can be interpreted according to the LF shown in (42).

(42)



If one is committed to a purely QR-based account of scope ambiguity then this kind of manoeuvre is unavoidable: in order for the QP 'an espresso machine' to take scope within the NP containing it, that NP has to be made clause-like so that the QP has a node of type t to adjoin to. The alternative is to allow some kind of type-shifting operation so that the embedded DP can be interpreted in situ, but once this kind of type-shifting is added to the system then the motivation for QR in general is weakened.

In contrast, nothing needs to be added for the Glue approach to account for examples such as (41). Assuming the additional lexical entries shown in (43)–(44) and therefore the structure of the subject DP shown in (45), the surface scope interpretation of the subject DP is derived as shown in (46)–(47) (where (46) feeds into (47)).

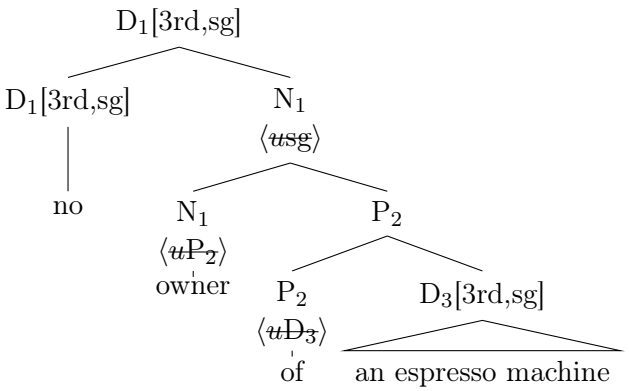
(43)

	owner
Syntax	$\langle \{\mathrm{N}_i\}, \langle u\mathrm{D}_j, u\mathrm{sg}_k \rangle \rangle$
Semantics	$\lambda x.\lambda y.\mathrm{own}'(y,x) : e_j \multimap (e_i \multimap t_i)$
	$i \neq j$

(44)

	of
Syntax	$\langle \{\mathrm{P}_i\}, \langle u\mathrm{D}_j \rangle \rangle$
Semantics	$\lambda x.x : e_j \multimap e_i$

(45)



(46)

$$\dfrac{\dfrac{\text{an espresso machine} \Downarrow \lambda F.\mathrm{some}'(\mathrm{machine}', F) : \forall X((e_3 \multimap t_X) \multimap t_X)}{\lambda F.\mathrm{some}'(\mathrm{machine}', F) : (e_3 \multimap t_1) \multimap t_1}\ \forall_E \qquad \dfrac{\dfrac{\dfrac{\text{owner} \Downarrow \lambda w.\lambda u.\mathrm{own}'(u,w) : e_2 \multimap (e_1 \multimap t_1) \quad \dfrac{\text{of} \Downarrow \lambda v.v : e_3 \multimap e_2 \quad \left[\begin{smallmatrix} y : \\ e_3 \end{smallmatrix}\right]^1}{y : e_2}}{\lambda u.\mathrm{own}'(u,y) : e_1 \multimap t_1} \quad \left[\begin{smallmatrix} x : \\ e_1 \end{smallmatrix}\right]^2}{\mathrm{own}'(x,y) : t_1}}{\lambda y.\mathrm{own}'(x,y) : e_3 \multimap t_1}\ \multimap_I{}^1}{\mathrm{some}'(\mathrm{machine}', \lambda y.\mathrm{own}'(x,y)) : t_1}$$

$$\dfrac{\mathrm{some}'(\mathrm{machine}', \lambda y.\mathrm{own}'(x,y)) : t_1}{\lambda x.\mathrm{some}'(\mathrm{machine}', \lambda y.\mathrm{own}'(x,y)) : e_1 \multimap t_1}\ \multimap_I{}^2$$

(47)

$$\dfrac{\dfrac{\text{no} \Downarrow \lambda P.\lambda Q.\neg\mathrm{some}'(P,Q) : (e_1 \multimap t_1) \multimap \forall Y((e_1 \multimap t_Y) \multimap t_Y)}{\lambda Q.\neg\mathrm{some}'(\lambda x.\mathrm{some}'(\mathrm{machine}', \lambda y.\mathrm{own}'(x,y)), Q) : \forall Y((e_1 \multimap t_Y) \multimap t_Y)}\ \forall_E \qquad \dfrac{\text{owner of an espresso machine} \Downarrow \lambda x.\mathrm{some}'(\mathrm{machine}', \lambda y.\mathrm{own}'(x,y)) : e_1 \multimap t_1}{\lambda Q.\neg\mathrm{some}'(\lambda x.\mathrm{some}'(\mathrm{machine}', \lambda y.\mathrm{own}'(x,y)), Q) : \forall Y((e_1 \multimap t_Y) \multimap t_Y)}$$

Note that nothing in this analysis depends on 'owner' having an internal argument, as has been assumed in (43). The surface scope interpretations of (48) and (49) (Heim & Kratzer 1998: Ch. 8) are derivable analogously, without the need for a subject position within NP, PP or AP.

(48) No student [$_{\text{PP}}$ from a foreign country] was admitted.

(49) No student [$_{\text{AP}}$ interested in more than one topic] showed up.

In each case, what is crucial is that the interpretation of the head noun ('owner' or 'student') is a function into objects of type t, thus providing the necessary scope point for the embedded QP without the need for the embedding NP as a whole to have an interpretation within type t.

4.2 Imposing constraints on scope-taking

The system described in Section 3.2, on its own, will allow a QP to take scope arbitrarily high. So for example, an interpretation for (50) would be derivable according to which for every boy there is some teacher or other (not necessarily the same one) who thinks that that boy smokes (the $\forall > \exists$ reading).

(50) A teacher thinks that every boy smokes.

It is commonly thought that this reading is not available for (50), and that the reason for this is that the tensed clause is an 'island' for scope, i.e. no constituent within it can take scope outside of it.[13]

In the Glue system as described in this paper, one way of making a constituent X a scope island is by saying that the root of X is a point at which a proof must be carried out, and hence the premises introduced by lexical items within that constituent used up. If we think of the interpretative mechanism working bottom-up through the syntactic structure collecting premises, until it reaches a point at which there is an instruction to compute a proof with a particular proof goal, then the idea that the tensed clause is a scope island can be expressed as shown in (51).

(51) At the maximal projection of T[tense], find a proof with conclusion $m : t_\alpha$, for some meaning m and some term α.

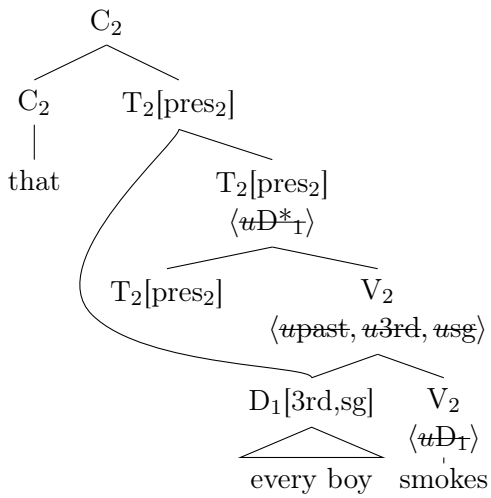
In (51), 'tense' is meant to be an abbreviation for some (possibly disjunctive) feature specification that distinguishes tensed clauses from untensed ones. So for example, T[past] and T[pres] are both instances of T[tense] in this sense.

13 There is some disagreement about these judgements. Perhaps (51) could or should be thought of as part of a parsing strategy rather than a constraint in the grammar.

Note that the type map given in (20) guarantees that m in (51) will be of type t.[14,15]

Given the structure of the embedded clause shown in (52), the premises collected are as shown in (53).

(52)



(53) Premises from the TP in (52):

a. $\lambda F.\text{every}'(\text{boy}', F) : \forall X((e_1 \multimap t_X) \multimap t_X)$

b. $\lambda x.\text{smoke}'(x) : e_1 \multimap t_2$

As the root of the TP in (52) is tensed, (51) requires a proof to be found from the premises shown in (53) to a conclusion of type t. One such proof is available, as shown in (54).

(54)
$$\dfrac{\dfrac{\lambda F.\text{every}'(\text{boy}', F) \quad : \forall X((e_1 \multimap t_X) \multimap t_X)}{\lambda F.\text{every}'(\text{boy}', F) \quad : (e_1 \multimap t_2) \multimap t_2}\,\forall_E \qquad \lambda x.\text{smoke}'(x) \quad : e_1 \multimap t_2}{\text{every}'(\text{boy}', \text{smoke}') : t_2}\,\multimap_E, \eta$$

14 I am sticking with an extensional system, although it is strictly speaking inaccurate for examples like (50). The best way to to move on from an extensional system would probably be to change the type map such that the LL predicate t maps to the type $s \to t$ for an intensional system, or *prop* for a hyperintensional one (Pollard 2008), and adapt the lambda terms accordingly.

15 Because of the interaction with (overt) movement, the restriction to a conclusion of the form $m : t_\alpha$ may have to be relaxed to allow conclusions of the form $m : e_\beta \multimap t_\alpha$ if the clause is one out of which some constituent is moving. Relaxing (51) in this way will not allow scope to escape from the clause: for that, you would need a conclusion of the form $m : (t_\alpha \multimap t_\beta) \multimap t_\gamma$ or some higher type.

The conclusion of (54) can now be used as a premise in computing the interpretation of (50), together with the premises contributed by 'a', 'teacher', 'thinks' and 'that'. Given the form of the conclusion of (54), there is no way for 'every boy' to take scope over 'a teacher' (or 'thinks') in that interpretation.

Other constraints like (51) may be useful in the grammar. To see this, consider an adapted form of (41), shown below in (55).

(55) Two baristas served an owner of every espresso machine.

From the working in Section 4.1 it should be clear how the $2 > \exists > \forall$ and $\exists > \forall > 2$ readings of (55) can be derived. The $\exists > 2 > \forall$ 'reading' is nonsensical (and cannot be derived in Glue: see (Asudeh & Crouch 2002: §4.1.2)). That leaves the $2 > \forall > \exists$, $\forall > 2 > \exists$ and $\forall > \exists > 2$ readings. The system described so far, on its own, will allow all of these readings to be derived. But it is thought that the $\forall > 2 > \exists$ reading is actually unavailable (May & Bale 2007).[16] That is to say, there is no reading of (55) on which for every espresso machine, two baristas each served a (possibly different) owner of that espresso machine.

To capture this restriction we would need to make DP a scope island, as shown in (56).[17]

(56) At the maximal projection of D, find a proof with conclusion $m : (e_\alpha \multimap t_\beta) \multimap t_\beta$, for some meaning m and some terms α and β.

Adhering to (56) ensures that there is no way to derive an interpretation for (55) in which the scope of 'two baristas' intervenes between that of 'every espresso machine' and 'an owner'. However, it does not prevent the derivation of an interpretation in which 'every espresso machine' takes scope over 'an owner'—the inverse linking reading (May & Bale 2007). That interpretation can be derived given the structure in (57), the resulting premises in (58) and the proof in (59).

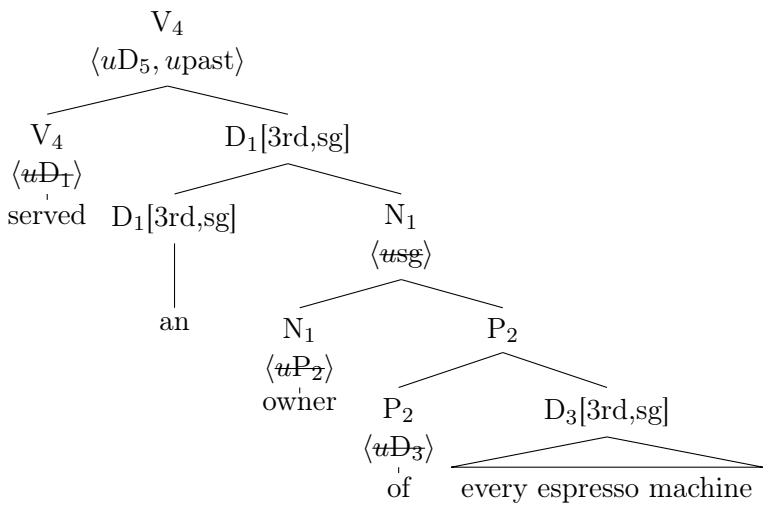
16 Thanks to Patrick Elliott for pushing me on this point.

17 As stated, (56) would require proper names to type raise from type e to type $(e \to t) \to t$. This is not a problem, since type raising is a theorem of the underlying logic, as the following proof shows.

$$\dfrac{\dfrac{\dfrac{[f : e_1 \multimap t_X]^1 \quad j' : e_1}{f(j') : t_X}\ \multimap_E}{\lambda f.f(j') : (e_1 \multimap t_X) \multimap t_X}\ \multimap_I{}^1}{\lambda f.f(j') : \forall X((e_1 \multimap t_X) \multimap t_X)}\ \forall_I$$

But perhaps it would still be worth exploring the idea that constraints like (51) and (56) describe the highest type the conclusion can have, rather than its only type.

(57)



(58) Premises from the DP complement of V in (57):

a. $\lambda P.\lambda Q.\text{some}'(P,Q) : (e_1 \multimap t_1) \multimap \forall X((e_1 \multimap t_X) \multimap t_X)$
b. $\lambda x.\lambda y.\text{own}'(y,x) : e_2 \multimap (e_1 \multimap t_1)$
c. $\lambda v.v : e_3 \multimap e_2$
d. $\lambda F.\text{every}'(\text{machine}', F) : (e_3 \multimap t_4) \multimap t_4$[18]

(59)

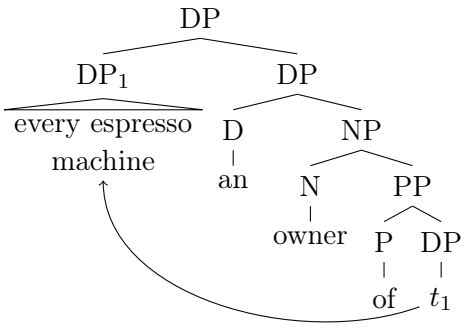
$$\dfrac{\dfrac{\dfrac{\dfrac{\lambda P.\lambda Q.\text{some}'(P,Q) : (e_1 \multimap t_1) \multimap \forall X((e_1 \multimap t_X) \multimap t_X) \quad \dfrac{\lambda x.\lambda y.\text{own}'(y,x) : e_2 \multimap (e_1 \multimap t_1) \quad \dfrac{\lambda v.v : e_3 \multimap e_2 \quad \left[\begin{smallmatrix} z : \\ e_3 \end{smallmatrix}\right]^1}{z : e_2}}{\lambda y.\text{own}'(y,z) : e_1 \multimap t_1}}{\lambda Q.\text{some}'(\lambda y.\text{own}'(y,z), Q) : \forall X((e_1 \multimap t_X) \multimap t_X)}}{\lambda Q.\text{some}'(\lambda y.\text{own}'(y,z), Q) : (e_1 \multimap t_4) \multimap t_4}\ \forall_E \quad \left[\begin{smallmatrix} P : \\ e_1 \multimap t_4 \end{smallmatrix}\right]^2}{\dfrac{\text{some}'(\lambda y.\text{own}'(y,z), P) : t_4}{\lambda z.\text{some}'(\lambda y.\text{own}'(y,z), P) : e_3 \multimap t_4}\ \multimap_I{}^1}$$

$$\dfrac{\lambda F.\text{every}'(\text{machine}', F) : (e_3 \multimap t_4) \multimap t_4 \quad \lambda z.\text{some}'(\lambda y.\text{own}'(y,z), P) : e_3 \multimap t_4}{\dfrac{\text{every}'(\text{machine}', \lambda z.\text{some}'(\lambda y.\text{own}'(y,z), P)) : t_4}{\lambda P.\text{every}'(\text{machine}', \lambda z.\text{some}'(\lambda y.\text{own}'(y,z), P)) : (e_1 \multimap t_4) \multimap t_4}\ \multimap_I{}^2}$$

18 So as to adhere to (56), 'every espresso machine' is itself shown as a single premise of the form $m : (e_\alpha \multimap t_\beta) \multimap t_\beta$.

The proof in (59) shows the derivation of an interpretation for the DP 'an owner of every espresso machine', of type $(e \to t) \to t$, in which 'every espresso machine' takes scope over 'an owner'. To gain this interpretation in a simple QR system, 'every espresso machine' would have to move out of it its containing DP and adjoin at clause level—which in turn would make the $\forall > 2 > \exists$ reading possible. The alternative is to allow embedded QPs undergoing QR to adjoin to their containing DP in the manner shown in (60), which again would necessitate additional type-shifting principles (or lexical polymorhism) in order for the sentence to be interpretable at all.

(60)



This discussion gives some idea of how constraints on scope-taking can be addressed in a Glue approach to semantic composition. Many more issues need to be tackled, however. For example, while the 'find a proof' strategy for constraining available readings as illustrated in (51) and (56) may work well for scope islands, it is doubtful that it could be adapted to account for 'scope freezing' effects such as can be seen in (61).

(61) Mr. Smith asked a candidate every question.

(61) has no reading on which for every question, Mr. Smith posed that question to a (possibly different) candidate; in the double object construction in (61), 'every question' cannot take cope over 'a candidate'. It may well be that in cases like these, the relevant constraint should not be stated at the level of the sentence but rather at the level of the linear logic proof itself, as described by Crouch & van Genabith (1999).

5 Conclusion

In this paper I have outlined the basics of Glue semantics and given what is, to my knowledge, the first implementation of Glue in a Minimalist syntactic framework. In this approach, syntactic analysis produces premises in a fragment of linear logic, and semantic interpretation consists in finding a proof from

those premises. In some circumstances more than one proof can be constructed from the same multiset of premises, which accounts for ambiguities of scope. A Glue analysis therefore removes the need for covert movement or ad-hoc type-shifting rules, insofar as these are motivated by scope ambiguities.

I have shown that in some respects the Glue approach to relative quantifier scope is preferable to the quantifier raising approach most commonly assumed in Minimalist analyses, particularly in cases where one quantified DP is embedded inside another. I have also discussed some ways of imposing constraints to limit possible readings when working in this framework. The next stage of investigation is to examine how this approach to meaning composition interacts with (overt) movement.

References

Adger, David. 2003. *Core syntax: A Minimalist approach* Core Linguistics. Oxford: Oxford University Press.

Adger, David. 2010. A Minimalist theory of feature structure. In Anna Kibort & Greville G. Corbett (eds.), *Features: Perspectives on a key notion in linguistics*, 185–218. Oxford: Oxford University Press.

Asudeh, Ash. 2004. *Resumption as resource management*: Stanford University dissertation.

Asudeh, Ash & Richard Crouch. 2002. Glue Semantics for HPSG. In Frank van Eynde, Lars Hellan & Dorothee Beermann (eds.), *Proceedings of the 8th international HPSG conference*, Stanford, CA: CSLI Publications.

Carpenter, Bob. 1998. *Type-logical semantics*. Cambridge, MA: MIT Press.

Cooper, Robin. 1983. *Quantification and syntactic theory* (Studies in Linguistics and Philosophy 21). Dordrecht: D. Reidel.

Crouch, Richard & Josef van Genabith. 1999. Context change, underspecification and the structure of Glue language derivations. In Mary Dalrymple (ed.), *Semantics and syntax in Lexical Functional Grammar: The resource logic approach*, 117–189. Cambridge, MA: MIT Press.

Crouch, Richard & Josef van Genabith. 2000. Linear logic for linguists. ESSLLI 2000 course notes. http://www.cs.bham.ac.uk/%7Evdp/DickCrouch.html.

Dalrymple, Mary (ed.). 1999. *Semantics and syntax in Lexical Functional Grammar: The resource logic approach*. Cambridge, MA: MIT Press.

Frank, Anette & Josef van Genabith. 2001. GlueTag: Linear logic-based semantics for LTAG—and what it teaches us about LFG and LTAG. In Miriam Butt & Tracy Holloway King (eds.), *Proceedings of the LFG01 conference*, Stanford, CA: CSLI Publications.

Girard, Jean-Yves. 1987. Linear logic. *Theoretical Computer Science* 50(1). 1–101. doi:10.1016/0304-3975(87)90045-4.

Heim, Irene & Angelika Kratzer. 1998. *Semantics in generative grammar* (Blackwell Textbooks in Linguistics 13). Oxford: Wiley-Blackwell.

Hendriks, Herman. 1987. Type change in semantics. In Ewan Klein & Johan van Bentham (eds.), *Categories, polymorphism and unification*, 95–119. Amsterdam: Instituut voor Taal, Logica en Informatie.

Kokkonidis, Miltiadis. 2008. First-order Glue. *Journal of Logic, Language and Information* 17(1). 43–68.

May, Robert. 1977. *The grammar of quantification*: Massachusetts Institute of Technology dissertation.

May, Robert & Alan Bale. 2007. Inverse linking. In Martin Everaert, Henk Van Riemsdijk, Rob Goedemans & Bart Hollebrandse (eds.), *The Blackwell companion to syntax*, vol. 2, chap. 36. Oxford: Blackwell.

Montague, Richard. 1973. The proper treatment of quantification in ordinary English. In Patrick Suppes, Julius Moravcsik & Jaakko Hintikka (eds.), *Approaches to natural language*, 221–242. Dordrecht: D. Reidel. Reprinted in (Thomason 1974: Chapter 8).

Morrill, Glynn V. 1994. *Type logical grammar: Categorial logic of signs*. Dordrecht, Netherlands: Kluwer Academic Publishers.

Morrill, Glynn V. 2011. *Categorial grammar: Logical syntax, semantics and processing*. Oxford: Oxford University Press.

Pollard, Carl. 2008. Hyperintensions. *Journal of Logic and Computation* 18(2). 257–282.

Retoré, Christian & Edward Stabler. 2004. Generative grammars in resource logics. *Research on Language and Computation* 2(1). 3–25. doi:10.1023/A:1025455403876.

Stabler, Edward. 1997. Derivational minimalism. In Christian Retoré (ed.), *Logical aspects of computational linguistics* (Lecture Notes in Computer Science 1328), 68–95. Berlin/Heidelberg: Springer. doi:10.1007/BFb0052152.

Thomason, Richmond H. (ed.). 1974. *Formal philosophy: Selected papers of Richard Montague*. New Haven/London: Yale University Press.

Towards Glue for Minimalism

Matthew Gotham
Research Department of Linguistics
University College London
2 Wakefield Street
London, WC1N 1PF
United Kingdom

m.gotham@ucl.ac.uk