

Approaches to scope islands in LFG+Glue

Matthew Gotham

University of Oxford

Proceedings of the LFG'21 Conference

On-Line

Miriam Butt, Jamie Y. Findlay, Ida Toivonen (Editors)


2021

CSLI Publications

pages 146–166

<http://csli-publications.stanford.edu/LFG/2021>

Keywords: scope, scope islands, quantification, Glue semantics, type logic

Gotham, Matthew. 2021. Approaches to scope islands in LFG+Glue. In Butt, Miriam, Findlay, Jamie Y., & Toivonen, Ida (Eds.), *Proceedings of the LFG'21 Conference, On-Line*, 146–166. Stanford, CA: CSLI Publications. 

Abstract

In this paper I examine two possible approaches to scope islands in LFG with Glue semantics: one in which constraints on scope level are imposed by means of constraints off the path of an inside-out functional uncertainty, and one in which they are imposed through the structural rules of the fragment of linear logic used for meaning composition, by making the fragment multi-modal. For each approach, I show how it could be made to account for novel empirical arguments made by Barker (2021), and go on to argue in favour of the multi-modal Glue approach.

1 Introduction

At the outset of theory design in formal linguistics, the theorist is faced with a fundamental choice. Do you start with something relatively constrained, and then find ways to loosen it as the evidence demands? Or do you start with something relatively *unconstrained*, and then find ways to constrain it as required? As a theory of the syntax/semantics interface, Glue semantics is towards the unconstrained end of the spectrum. This paper is addressed at the need to constrain Glue with respect to the phenomenon of quantifier scope, in particular the (non-)ability of a quantified noun phrase to take scope outside of its minimal clause.

1.1 Background

It is a feature of the Glue approach to semantic composition that many instances of quantifier scope ambiguity are resolved purely at the level of linear logic proofs. For example, the two interpretations of (1) shown below, surface scope and inverse scope respectively, can both be derived from the same f-structure and the same associated meaning constructors, as shown in (2) and (3) respectively.¹

- (1) Someone sees everything.
⇒ someone(λx .everything(λy .see(x, y)))
⇒ everything(λy .someone(λx .see(x, y)))

[†]I thank the audience of LFG'21 for helpful and encouraging feedback. This research is funded by an Early Career Fellowship from the Leverhulme Trust.

¹The subscripts e (entities) and p (propositions) represent types. Semantically, we can think of p as equivalent to $s \rightarrow t$. I use \boxed{n} and \mathbf{n} interchangeably for f-structure labels / linear logic formulae. We have the following logical constants on the meaning side:

every, some :: $((e \rightarrow p) \times (e \rightarrow p)) \rightarrow p$
everything, something, someone :: $(e \rightarrow p) \rightarrow p$
not :: $p \rightarrow p$

$$\begin{array}{c}
\lambda v. \lambda u. \text{see}(u, v) : \\
\frac{\mathbf{2}_e \multimap (\mathbf{1}_e \multimap \mathbf{0}_p) \quad [y : \mathbf{2}_e]^1}{\lambda u. \text{see}(u, y) : \mathbf{1}_e \multimap \mathbf{0}_p} \quad [x : \mathbf{1}_e]^2 \\
\text{everything} : \frac{\text{see}(x, y) : \mathbf{0}_p}{\lambda y. \text{see}(x, y) : \mathbf{2}_e \multimap \mathbf{0}_p} \multimap \text{I}^1 \\
\frac{(\mathbf{2}_e \multimap \mathbf{0}_p) \multimap \mathbf{0}_p}{\text{everything}(\lambda y. \text{see}(x, y)) : \mathbf{0}_p} \\
\text{someone} : \frac{(\mathbf{1}_e \multimap \mathbf{0}_p) \multimap \mathbf{0}_p}{\lambda x. \text{everything}(\lambda y. \text{see}(x, y)) : \mathbf{1}_e \multimap \mathbf{0}_p} \multimap \text{I}^2 \\
\frac{}{\text{someone}(\lambda x. \text{everything}(\lambda y. \text{see}(x, y))) : \mathbf{0}_p}
\end{array}$$

Figure 1: Derivation of the surface scope interpretation of (1) from (3)

$$\begin{array}{c}
\lambda v. \lambda x. \text{see}(x, v) : \\
\text{someone} : \frac{\mathbf{2}_e \multimap (\mathbf{1}_e \multimap \mathbf{0}_p) \quad [y : \mathbf{2}_e]^1}{(\mathbf{1}_e \multimap \mathbf{0}_p) \multimap \mathbf{0}_p \quad \lambda x. \text{see}(x, y) : \mathbf{1}_e \multimap \mathbf{0}_p} \\
\text{everything} : \frac{(\mathbf{2}_e \multimap \mathbf{0}_p) \multimap \mathbf{0}_p}{\lambda y. \text{someone}(\lambda x. \text{see}(x, y)) : \mathbf{2}_e \multimap \mathbf{0}_p} \multimap \text{I}^1 \\
\frac{}{\text{everything}(\lambda y. \text{someone}(\lambda x. \text{see}(x, y))) : \mathbf{0}_p}
\end{array}$$

Figure 2: Derivation of the inverse scope interpretation of (1) from (3)

$$\begin{array}{l}
(2) \quad \left[\begin{array}{l} \text{PRED} \quad \text{'see'}(\boxed{1}, \boxed{2}) \\ \text{TENSE} \quad \text{PRES} \\ \boxed{0} \text{ SUBJ} \quad \boxed{1} \left[\text{PRED} \quad \text{'someone'} \right] \\ \text{OBJ} \quad \boxed{2} \left[\text{PRED} \quad \text{'everything'} \right] \end{array} \right] \\
(3) \quad \text{someone} : (\mathbf{1}_e \multimap \mathbf{0}_p) \multimap \mathbf{0}_p \\
\lambda y. \lambda x. \text{see}(x, y) : \mathbf{2}_e \multimap (\mathbf{1}_e \multimap \mathbf{0}_p) \\
\text{everything} : (\mathbf{2}_e \multimap \mathbf{0}_p) \multimap \mathbf{0}_p
\end{array}$$

The proofs deriving these interpretations are shown in Figures 1 and 2, respectively.² However, not all instances of scope ambiguity can be handled quite as simply as this.

1.2 Scope level

Consider (4), which has the surface scope and inverse linking interpretations shown below, and has the (simplified) f-structure shown in (5).

$$\begin{array}{l}
(4) \quad \text{A member of every board resigned.} \\
\Rightarrow \text{some}(\lambda x. \text{every}(\text{board}, \lambda y. \text{member-of}(x, y)), \text{resign})
\end{array}$$

²Throughout this paper, un-annotated steps of inference should be read as instances of \multimap elimination, to save space.

‘Someone who is a member of every board resigned.’
 \Rightarrow every(board, $\lambda y.$ some($\lambda x.$ member-of(x, y), resign))
 ‘For every board, someone member of that board resigned.’

$$(5) \quad \left[\begin{array}{l} \text{PRED} \quad \text{‘resign’} \langle \boxed{1} \rangle \\ \text{SUBJ} \quad \boxed{1} \left[\begin{array}{l} \text{PRED} \quad \text{‘member’} \langle \boxed{2} \rangle \\ \text{SPEC} \quad \left[\text{PRED} \quad \text{‘a’} \right] \\ \text{OBJ} \quad \boxed{2} \left[\text{“every board”} \right] \end{array} \right] \end{array} \right]$$

Unlike (1), the difference between the two interpretations of (4) *does* depend on a difference in meaning constructors. Specifically, the meaning constructor associated with *every board* in (5) is as shown schematically in (6), where \square is a placeholder.

$$(6) \quad \lambda P.\text{every}(\text{board}, P) : (2_e \multimap \square_p) \multimap \square_p$$

To derive the surface scope interpretation, the formula shown as \square in (6) has to be $\boxed{1}$, while to derive the inverse linking interpretation, it has to be $\boxed{0}$. I will refer to this choice as the choice of ‘scope level’ for a meaning constructor. Examples like (5) differ from those like (2) in that there *is* a choice of scope level for at least one quantifier—and this choice moreover matters.

In the literature, there are essentially two approaches to resolving scope level. The first, adopted e.g. by Lev (2007), Andrews (2010) and Gotham (2019), is to treat this as an instance of functional uncertainty. The lexical entry for *every* could contain the description shown in (7), where we leave SCOPEPATH unspecified for now but note that it should at least include OBJ (for $\boxed{1}$) and SUBJ OBJ (for $\boxed{0}$).³

$$(7) \quad \%A = (\text{SCOPEPATH} \uparrow) \\ \lambda P.\lambda Q.\text{every}(P, Q) : (\uparrow_e \multimap \uparrow_p) \multimap ((\uparrow_e \multimap \%A_p) \multimap \%A_p)$$

The second approach, which is more widely adopted (including in Dalrymple et al. (2019)), is to use quantification in the linear logic fragment to express the various possible scope levels. The meaning constructor for *every* would then look something like (8).

$$(8) \quad \lambda P.\lambda Q.\text{every}(P, Q) : (\uparrow_e \multimap \uparrow_p) \multimap \forall X((\uparrow_e \multimap X_p) \multimap X_p)$$

³In the fragment of second-order linear logic most often assumed for Glue, as described e.g. in (Dalrymple et al. 2019, Chapter 8), it would be strictly speaking incoherent to have both 2_e and 2_p , since the subscripts are supposed to be sort labels and a formula cannot belong to more than one sort. Nevertheless, you *can* do this in the XLE+Glue implementation (Dalrymple et al. 2020). I am not able to speak to what is going on under the hood in the implementation, but one coherent way to interpret the notation would be to take the subscripts to be unary propositional functions, as in Gotham and Haug (2018). Another would be to switch from a second-order to a first-order system, and treat e and p as predicates to which the f-structure labels are arguments (Kokkonidis 2008).

In the proof, X can then be instantiated to either $\mathbf{0}$ or $\mathbf{1}$, deriving the respective interpretations, as shown below.

$$\frac{\lambda Q.\text{every}(\text{board}, Q) : \forall X.(\mathbf{2}_e \multimap X_p) \multimap X_p}{\lambda Q.\text{every}(\text{board}, Q) : (\mathbf{2}_e \multimap \mathbf{0}_p) \multimap \mathbf{0}_p} \forall_E \quad \text{or} \quad \frac{\lambda Q.\text{every}(\text{board}, Q) : \forall X.(\mathbf{2}_e \multimap X_p) \multimap X_p}{\lambda Q.\text{every}(\text{board}, Q) : (\mathbf{2}_e \multimap \mathbf{1}_p) \multimap \mathbf{1}_p} \forall_E$$

2 Scope islands

The point of departure for this paper is the fact that this choice of scope level is not entirely free. Consider, for example, (9), which has the surface scope interpretation, but *not* the inverse scope interpretation—it can only mean that there is a particular warden who thinks that every prisoner escaped, and not that for every prisoner, there is some warden or other who thinks that prisoner escaped.

- (9) A warden thinks that every prisoner escaped.
 \Rightarrow some(warden, $\lambda x.\text{think}(x, \text{every}(\text{prisoner}, \text{escape}))$)
 $\not\Rightarrow$ every(prisoner, $\lambda y.\text{some}(\text{warden}, \lambda x.\text{think}(x, \text{escape}(y)))$)

Given the (simplified) f-structure for (9) shown in (10), that would amount to saying that the meaning constructor associated with *every* (or *every prisoner*) can take $\boxed{2}$ as its scope level, but not $\boxed{0}$.

$$(10) \quad \boxed{0} \left[\begin{array}{l} \text{PRED} \quad \text{'think}(\boxed{1}, \boxed{2}) \\ \text{TENSE} \quad \text{PRES} \\ \text{SUBJ} \quad \boxed{1} \left[\text{'a warden'} \right] \\ \text{COMP} \quad \boxed{2} \left[\begin{array}{l} \text{PRED} \quad \text{'escape}(\boxed{3}) \\ \text{TENSE} \quad \text{PAST} \\ \text{SUBJ} \quad \boxed{3} \left[\text{'every prisoner'} \right] \end{array} \right] \end{array} \right]$$

The received wisdom (May 1977) about examples like these is that the inverse scope interpretation is unavailable because finite clauses are ‘scope islands’, meaning that no quantifier inside of one can take scope out of it. The received wisdom seems to favour the functional uncertainty approach to scope level, as this constraint can be imposed by appropriately defining SCOPEPATH from (7), e.g. as shown in (11). By contrast, it is harder to see how such a constraint could be stated in the approach using quantification in the linear logic fragment to fix scope level.

$$(11) \quad \text{SCOPEPATH} \equiv \left(\begin{array}{cc} \text{GF}^* & \text{GF} \\ \neg(\rightarrow \text{TENSE}) & \end{array} \right)$$

The well-known fact that indefinites are not so constrained—that they can take

‘exceptional scope’ (Charlow 2014), as in (12)—can then be accounted for by allowing their scope level to be fixed by a less constrained path.⁴

- (12) Every warden thinks that a prisoner escaped.
 \Rightarrow every(warden, λx .think(x , some(prisoner, escape)))
 \Rightarrow some(prisoner, λy .every(warden, λx .think(x , escape(y))))

2.1 Varieties of scope island

However, it is becoming increasingly clear that the received wisdom is too simplistic. As pointed out by Barker (2021), not all finite clauses are scope islands for all quantifiers. For example, (13) *does* have an interpretation where *every prisoner* takes widest scope, as shown—it *can* mean that for every prisoner, some accomplice or other ensured that that prisoner escaped.

- (13) An accomplice ensured that every prisoner escaped.
 \Rightarrow some(accomplice, λx .ensure(x , every(prisoner, escape)))
 \Rightarrow every(prisoner, λy .some(accomplice, λx .ensure(x , escape(y))))

So, *every N* can take scope out of a finite clause, provided that clause is embedded by *ensured*. Note, however, that this does not mean that the clause embedded by *ensured* is not a scope island at all. As shown in (14), *it is* a scope island for *no N*. I have marked (14) as questionable because the one interpretation it does have conflicts with world knowledge about what it means to be an accomplice; it can only mean (implausibly) that there is a particular accomplice who ensured that no prisoner escaped, and not (more plausibly) that no prisoner is such that some accomplice or other ensured that that prisoner escaped.

- (14) ?An accomplice ensured that no prisoner escaped.
 \Rightarrow some(accomplice, λx .ensure(x , not(some(prisoner, escape))))
 \Rightarrow not(some(prisoner, λy .some(accomplice, λx .ensure(x , escape(y))))))

These observations invite the hypotheses that, in some sense, (i) *think* induces a stronger scope island than *ensure*, and (ii) *every N* is a stronger island-escaper than *no N*. The hypotheses are confirmed by filling in the gap in the paradigm: since the complement of *think* is a scope island for *every N*, if *every N* is a stronger island-escaper than *no N*, then we expect the complement of *think* to be a scope island for *no N* as well. This prediction is borne out, as shown in (15).

- (15) A warden thinks that no prisoner escaped.
 \Rightarrow some(warden, λx .think(x , not(some(prisoner, escape))))
 \Rightarrow not(some(prisoner, λy .some(warden, λx .think(x , escape(y))))))

⁴We ignore the possibility of treating indefinites as something other than quantifiers, semantically.

Meanwhile, being a strong enough island-escaper to take scope out of the complement of *think*, an *N* can certainly take scope out of the complement of *ensure*:

- (16) Every accomplice ensured that a prisoner escaped.
 \Rightarrow every(accomplice, λx .ensure(x , some(prisoner, escape)))
 \Rightarrow some(prisoner, λy .every(accomplice, λx .ensure(x , escape(y))))

These data imply an implicational relationship, which Barker (2021) dubs the ‘Scope Island Subset Constraint’ (SISC):

SISC Given any two scope takers, the set of scope islands that trap one is a subset of the set of scope islands that trap the other.

So far we have only looked at three scope-takers and two clause-embedders, but a further piece of evidence in favour of the SISC comes from the behaviour of negative polarity items (NPIs). To be licensed, an NPI must be interpreted within the scope of an appropriate ‘negative’ licenser—Fry (1999) shows a method for ensuring this in LFG+Glue. However, as is acknowledged by Fry (1999), this method has the shortcoming that it does not ensure that an NPI be interpreted in the scope of its *closest* relevant licenser. For example, in (17) there are two potential licensors for the NPI *anyone*—*surprised* and *didn’t*—but the NPI has to be interpreted as scoping under both of them, as shown.

- (17) Martha is surprised that Mary didn’t help anyone.
 \Rightarrow surprise(not(someone(λx .help(mary, x))), martha)
 \Rightarrow surprise(someone(λx .not(help(mary, x))), martha)

That is to say, (17) can mean that Martha is surprised that there’s no-one that Mary helped, but not that Martha is surprised that there’s someone that Mary didn’t help (or equivalently, that Martha is surprised that Mary didn’t help *everyone*). A natural explanation for this distinction would be that, in addition to being *licensors* for NPIs, at least some such expressions—such as overt negation—also induce *scope islands* for NPIs.

Meanwhile, like *a N* but unlike *every N* and *no N*, *any N* can take scope out of a clause embedded by *thinks*, as (18) shows.

- (18) If Mary thinks anyone is to blame, that person is Bob.
 \Rightarrow if(someone(λx .think(mary, blame(x))), think(mary, blame(bob)))

Here, the antecedent of the conditional provides the relevant context for NPI licensing. The form of the consequent is chosen so as to privilege the interpretation of the antecedent according to which *there is someone* that Mary thinks is to blame, i.e. in which *anyone* takes scope over *thinks* (but under *if*, which licenses it).

So, *any N* seems to be a weaker island-escaper than *a N*, but a stronger island-escaper than *every N* and *no N*. The SISC therefore predicts, given the fact that negation induces a scope island for *any N*, that it also induces a scope island for

clause embedder	quantifier				island strength
	<i>an N</i>	<i>any N</i>	<i>every N</i>	<i>no N</i>	
<i>not</i>		*	*	*	3
<i>think</i>			*	*	2
<i>ensure</i>				*	1
escaper strength	3	2	1	0	

Table 1: Relative strength of islands and escapers

every N and *no N*. Once again, the prediction is borne out, as shown in (19) and (20) respectively.⁵

- (19) Jesus didn't heal everyone.
 \Rightarrow $\text{not}(\text{everyone}(\lambda x.\text{heal}(\text{jesus}, x)))$
 \equiv $\text{someone}(\lambda x.\text{not}(\text{heal}(\text{jesus}, x)))$
 \nRightarrow $\text{everyone}(\lambda x.\text{not}(\text{heal}(\text{jesus}, x)))$
 \equiv $\text{not}(\text{someone}(\lambda x.\text{heal}(\text{jesus}, x)))$
- (20) Simon didn't receive nothing.
 \Rightarrow $\text{not}(\text{not}(\text{something}(\lambda x.\text{receive}(\text{simon}, x))))$
 \equiv $\text{something}(\lambda x.\text{receive}(\text{simon}, x))$
 \nRightarrow $\text{not}(\text{something}(\lambda x.\text{not}(\text{receive}(\text{simon}, x))))$
 \equiv $\text{everything}(\lambda x.\text{receive}(\text{simon}, x))$

We can summarise the empirical landscape in Table 1, adapted from (Barker 2021, Table 1). An asterisk in a cell means that the relevant scope taker is unable to take scope out of the island induced by the relevant clause embedder.⁶ In the following two sections I will outline and evaluate two possible approaches to these data.

3 Blocking features and off-path constraints

It is still possible to impose some of the relevant constraints on scope level using the kind of inside-out functional uncertainty technique exemplified in (11). However, additional difficulties arise with the attempt. First of all, it is not clear how the scope island induced by negation can be accounted for, since the mainstream view of negation in LFG is that it is represented in f-structure either by the value of a NEG or POL feature, or as a member of the ADJ set, at the matrix level (Dalrymple et al. 2019, 67–69). The point is that in none of these accounts does negation embed the f-structure representing the negatum, and so the issue of scope level does not

⁵In some varieties of English, (20) has a negative concord interpretation, where *nothing* is interpreted as equivalent to the NPI *anything*. This is a separate issue which does not affect the discussion.

⁶The table in Barker (2021) is somewhat different, partly because he considers issues that there is not space to address here, for example the semantics of focus.

arise.

For example, take (21), representing a simplified version of the f-structure of (19) according to the INESS XLE-WEB (Rosén et al. 2012). Since the f-structure introduced by negation does not lie on the path between the f-structure for *everyone* and any possible scope level, it cannot be used to constrain quantifier scope—there is only one possible scope level for *everyone*: $\boxed{0}$.

$$(21) \quad \left[\begin{array}{l} \text{PRED} \quad \text{'heal'}(\boxed{1}, \boxed{2}) \\ \text{SUBJ} \quad \boxed{1} \left[\text{PRED} \quad \text{'Jesus'} \right] \\ \boxed{0} \text{ OBJ} \quad \boxed{2} \left[\text{PRED} \quad \text{'everyone'} \right] \\ \text{ADJ} \quad \left\{ \boxed{3} \left[\text{PRED} \quad \text{'not'} \right] \right\} \end{array} \right]$$

In fact, we could view this issue with negation as an instance of a more general question, namely what the connection is between extra- and intra-clausal scope rigidity phenomena: ‘scope islands’ and ‘scope freezing’, respectively. If we use features in f-structure to impose constraints on scope level and thus account for scope islands, we need a completely different account of scope freezing. In many languages simple two-quantifier sentences like (1) are *not* ambiguous, for instance—an empirical fact that needs accounting for given Glue’s general unconstrainedness. In Gotham (2019) I proposed an account of scope freezing for examples like this, which could certainly be combined with a blocking features-based account of scope islands (as I suggested there), but perhaps it would be preferable to account for both kinds of scope rigidity within the same framework. I will return to this issue in Section 5.

With respect to the difference between *think* and *ensure*-type verbs, we *can* impose the relevant constraints by introducing different types of blocking feature. One way of doing so is exemplified in Figure 3, where we have sentence-embedding verbs projecting a SCOPEISLAND feature into their complement f-structure, and scope-takers sensitive to those features.

If we leave aside negation, the feature specifications in Figure 3 capture the facts in Table 1. But even so, problems remain. For one thing, it remains an open question as to whether or not the SCOPEISLAND feature can be independently motivated. Verbs of attitude and perception seem to pattern with *think*, so one could argue that there is a semantic generalization, but at the moment the only thing for this feature to do would be to enforce scope-islandhood.

Another potential problem relates to the SISC. Given the setup in Figure 3, there is nothing about the theory that prevents us from giving a lexical entry for a quantifier *nunone* that can take scope out of a clause embedded by *think* but *not*

$$\begin{array}{l}
\textit{thinks} \quad V \\
(\uparrow \text{COMP SCOPEISLAND}) = 2 \\
\textit{ensured} \quad V \\
(\uparrow \text{COMP SCOPEISLAND}) = 1 \\
\textit{someone} \quad N \\
\%X = (\text{GF}^* \text{GF} \uparrow) \\
\textit{someone} : (\uparrow_e \multimap \%X_p) \multimap \%X_p \\
\textit{everyone} \quad N \\
\%Y = \left(\begin{array}{c} \text{GF}^* \qquad \text{GF} \uparrow \\ (\rightarrow \text{SCOPEISLAND}) \neq 2 \end{array} \right) \\
\textit{everyone} : (\uparrow_e \multimap \%Y_p) \multimap \%Y_p \\
\textit{no-one} \quad N \\
\%Z = \left(\begin{array}{c} \text{GF}^* \qquad \text{GF} \uparrow \\ (\rightarrow \text{SCOPEISLAND}) \neq \{1 | 2\} \end{array} \right) \\
\lambda P.\text{not}(\textit{someone}(P)) : (\uparrow_e \multimap \%Z_p) \multimap \%Z_p
\end{array}$$

Figure 3: Possible lexical constraints on scope

out of a clause embedded by *ensure*, as shown in (22).

$$\begin{array}{l}
(22) \quad \textit{nunone} \quad N \\
\%C = \left(\begin{array}{c} \text{GF}^* \qquad \text{GF} \uparrow \\ (\rightarrow \text{SCOPEISLAND}) \neq 1 \end{array} \right) \\
\textit{nunone} : (\uparrow_e \multimap \%C_p) \multimap \%C_p
\end{array}$$

If we wanted to state the SISC as a grammar-wide constraint, then, we would have to do so by stating a constraint on the form of possible descriptions, to rule out lexical entries like (22). This is not impossible, but the relevant constraint would in all likelihood be quite messy and it is an open question exactly what form it would take. In view of these limitations, it is worth considering an alternative.

4 Multi-modal Glue semantics

An alternative approach is to impose the relevant constraints within Glue semantics itself. This requires some complication of the linear logic fragment used in Glue, but it can be argued on the basis of the data we have seen that the complication is linguistically motivated.

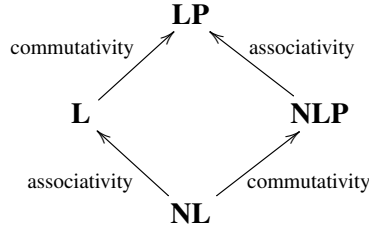


Figure 4: Substructural type logics (Moot and Retoré 2012, 111)

Within the family of substructural type logics, the base fragment⁷ of linear logic used in Glue is equivalent to the Lambek calculus with permutation **LP**. It is thus both commutative and associative, and so relates to the others as shown in Figure 4. Commutativity means that the premises in a proof have no particular order (or, equivalently, can be freely reordered), as shown schematically in (23), and associativity means that the premises in a proof have no particular grouping (or, equivalently, can be freely regrouped), as shown schematically in (24).

$$(23) \quad \frac{(\Gamma, \Delta) \vdash A}{(\Delta, \Gamma) \vdash A}$$

$$(24) \quad \frac{((\Gamma, \Delta), \Sigma) \vdash A}{(\Gamma, (\Delta, \Sigma)) \vdash A}$$

So far, **LP** has been a good choice of logic for Glue: unlike in categorial grammar, the logic is not meant to account for word order and so it makes sense for it to be commutative. So far it has also made sense for the logic to be associative, but scope islands may actually give us a reason to care about how premises are grouped, and so restrict associativity. We can do so selectively by combining elements of **LP** (as before) and **NLP** (which is non-associative) in a multimodal system, where the modes correspond to the island/escaper strengths outlined in Table 1. An implementation of these ideas is given by the rules of inference shown in Figure 5, in combination with the lexical entries shown in Figure 6.⁸ In the lexical entries, \multimap (without a mode index) is shorthand for $\multimap_{|0}$, and \multimap_i means the choice of index is free (so these can be seen as parameterized lexical entries).

The idea behind the rules in Figure 5 is that the \downarrow modes show blocking strength, and the \uparrow modes show escaping strength. Note that now, because we no longer assume generalized associativity, there is bracketing on the left hand side of sequents. The mode indices on those brackets correspond to mode indices on occurrences of \multimap . Commutativity is ensured by the structural rule **P** (for *permutation*), and we have restricted associativity thanks to the rule **MA** (*mixed associativity*). The modes

⁷By ‘base fragment’ I mean without quantification such as was discussed in connection with (8).

⁸The meaning constructor given for *not* is of type $(e \rightarrow p) \rightarrow (e \rightarrow p)$ rather than $p \rightarrow p$ in order to allow (and in fact, require) a quantifier in subject position to take scope over negation. An alternative way of achieving this aim will be explored in Section 5. Note that we are now able to fix scope level with linear logic quantification, although the IOFU method is also still available.

$\overline{x : A \vdash x : A}$ axiom

For modes $i, j \in \{\downarrow 0, \uparrow 1, \downarrow 2, \uparrow 3, \downarrow 1, \uparrow 2, \downarrow 3\}$:

$$\frac{\Gamma \vdash x : A \quad \Delta \vdash f : A \multimap_i B}{(\Gamma, \Delta)^i \vdash f(x) : B} \multimap_i E \quad \frac{(x : A, \Gamma)^i \vdash y : B}{\Gamma \vdash \lambda x. y : A \multimap_i B} \multimap_i I$$

$$\frac{(\Gamma, \Delta)^i \vdash x : A}{(\Delta, \Gamma)^i \vdash x : A} P$$

$$\frac{((\Gamma, \Delta)^i, \Sigma)^j \vdash x : A}{(\Gamma, (\Delta, \Sigma)^j)^i \vdash x : A} MA$$

\leftarrow provided
that j does
not block i

j blocks $i \Leftrightarrow j = \downarrow m, i = \uparrow n$ and $m > n$

Figure 5: Proposed rules of inference for multi-modal Glue

not Adv

$\%A = (\text{ADJ} \in \uparrow)$

$\lambda P. \lambda x. \text{not}(P(x)) : ((\%A \text{ SUBJ})_e \multimap_i \%A_p) \multimap_{\downarrow 3} ((\%A \text{ SUBJ})_e \multimap_i \%A_p)$

thinks V

$\lambda p. \lambda x. \text{think}(x, p) : (\uparrow \text{COMP})_p \multimap_{\downarrow 2} ((\uparrow \text{SUBJ})_e \multimap_i \uparrow_p)$

ensured V

$\lambda p. \lambda x. \text{ensure}(x, p) : (\uparrow \text{COMP})_p \multimap_{\downarrow 1} ((\uparrow \text{SUBJ})_e \multimap_i \uparrow_p)$

a D

$\lambda P. \lambda Q. \text{some}(P, Q) : (\uparrow_e \multimap \uparrow_p) \multimap \forall X ((\uparrow_e \multimap_{\downarrow 3} X_p) \multimap X_p)$

any D

$\lambda P. \lambda Q. \text{some}(P, Q) : (\uparrow_e \multimap \uparrow_p) \multimap \forall X ((\uparrow_e \multimap_{\downarrow 2} X_p) \multimap X_p)$

every D

$\lambda P. \lambda Q. \text{every}(P, Q) : (\uparrow_e \multimap \uparrow_p) \multimap \forall X ((\uparrow_e \multimap_{\downarrow 1} X_p) \multimap X_p)$

no D

$\lambda P. \lambda Q. \text{not}(\text{some}(P, Q)) : (\uparrow_e \multimap \uparrow_p) \multimap \forall X ((\uparrow_e \multimap X_p) \multimap X_p)$

Figure 6: Some partial lexical entries for the fragment

$$\begin{array}{c}
\vdots \\
\text{[escaped]} \vdash \quad \text{[every prisoner]} \vdash \\
\text{escape} : \lambda P.\text{every}(\text{prisoner}, P) : \\
\mathbf{3}_e \multimap_{\circ 1} \mathbf{2}_p \quad (\mathbf{3}_e \multimap_{\circ 1} \mathbf{2}_p) \multimap \mathbf{2}_p \quad \text{[ensured]} \vdash \quad \vdots \\
\hline
(\text{[escaped]}, \text{[every prisoner]}) \vdash \quad \lambda p.\lambda x.\text{ensure}(x, p) : \quad \text{[an accomplice]} \vdash \\
\text{every}(\text{prisoner}, \text{escape}) : \mathbf{2}_p \quad \mathbf{2}_p \multimap_{\circ 1} (\mathbf{1}_e \multimap_{\circ 13} \mathbf{0}_p) \lambda P.\text{some}(\text{accomplice}, \\
\hline
((\text{[escaped]}, \text{[every prisoner]}), \text{[ensured]})^{\uparrow 1} \vdash \quad P) : \\
\lambda x.\text{ensure}(x, \text{every}(\text{prisoner}, \text{escape})) : \mathbf{1}_e \multimap_{\circ 13} \mathbf{0}_p \quad (\mathbf{1}_e \multimap_{\circ 13} \mathbf{0}_p) \multimap \mathbf{0}_p \\
\hline
(((\text{[escaped]}, \text{[every prisoner]}), \text{[ensured]})^{\uparrow 1}, \text{[an accomplice]}) \vdash \\
\text{some}(\text{accomplice}, \lambda x.\text{ensure}(x, \text{every}(\text{prisoner}, \text{escape}))) : \mathbf{0}_p
\end{array}$$

Figure 7: Surface scope interpretation of (13)

interact in the MA rule in such a way that, in combination with the lexicon shown in Figure 6, just the right scope takers are able to escape from just the right islands.

4.1 Multi-modal Glue in action

The time has come to look at some examples. We have the space to go through two: (13), which permits an inverse scope interpretation, and (9), which does not. Given the (simplified) f-structure of (13) shown in (25) and the appropriately instantiated meaning constructors shown in (26), both the surface scope and inverse scope interpretations are available, as shown in Figures 7 and 8 respectively.

(13) An accomplice ensured that every prisoner escaped.

$$(25) \quad \left[\begin{array}{l} \text{PRED} \quad \text{'ensure'} \langle \text{[1]}, \text{[2]} \rangle \\ \text{SUBJ} \quad \text{[1]} \left[\text{"an accomplice"} \right] \\ \text{COMP} \quad \text{[2]} \left[\begin{array}{l} \text{PRED} \quad \text{'escape'} \langle \text{[3]} \rangle \\ \text{SUBJ} \quad \text{[3]} \left[\text{"every prisoner"} \right] \end{array} \right] \end{array} \right]$$

$$(26) \quad \begin{array}{l} \text{[an accomplice]} := \lambda P.\text{some}(\text{accomplice}, P) : (\mathbf{1}_e \multimap_{\circ 13} \mathbf{0}_p) \multimap \mathbf{0}_p \\ \text{[ensured]} := \lambda p.\lambda x.\text{think}(x, p) : \mathbf{2}_p \multimap_{\circ 12} (\mathbf{1}_e \multimap_{\circ 13} \mathbf{0}_p) \\ \text{[every prisoner]} := \lambda P.\text{every}(\text{prisoner}, P) : \forall X((\mathbf{3}_e \multimap_{\circ 11} X_p) \multimap X_p) \\ \text{[escaped]} := \text{escape} : \mathbf{3}_e \multimap_{\circ 11} \mathbf{2}_p \end{array}$$

The proof in Figure 8 depends on two instances of mixed associativity in order to ‘move’ the variable y to the outside of the premise structure so that it can be abstracted at the step of $\multimap_{\circ 11}$ introduction. The crucial MA step is the first one, having the schematic form shown in (27).

$$(27) \quad \frac{((\Gamma, \Delta)^{\uparrow 1}, \Sigma)^{\uparrow 1} \vdash \dots}{(\Gamma, (\Delta, \Sigma)^{\uparrow 1})^{\uparrow 1} \vdash \dots} \text{MA}$$

$$\begin{array}{c}
\frac{y : \mathbf{3}_e \vdash \text{[escaped]} \vdash}{y : \mathbf{3}_e \quad \mathbf{3}_e \multimap_{\downarrow 1} \mathbf{2}_p} \quad \frac{\text{[thinks]} \vdash}{\lambda p. \lambda x. \text{think}(x, p) : \mathbf{2}_p \multimap_{\downarrow 2} (\mathbf{1}_e \multimap_{\circ 3} \mathbf{0}_p)} \\
\frac{\frac{(y : \mathbf{3}_e, \text{[escaped]})^{\downarrow 1} \vdash \quad \text{escape}(y) : \mathbf{2}_p}{((y : \mathbf{3}_e, \text{[escaped]})^{\downarrow 1}, \text{[thinks]})^{\downarrow 2} \vdash \quad \lambda x. \text{think}(x, \text{escape}(y)) : \mathbf{1}_e \multimap_{\circ 3} \mathbf{0}_p}}{\text{[a warden]} \vdash \quad \lambda P. \text{some}(\text{warden}, P) : (\mathbf{1}_e \multimap_{\circ 3} \mathbf{0}_p) \multimap \mathbf{0}_p}}{\frac{((y : \mathbf{3}_e, \text{[escaped]})^{\downarrow 1}, \text{[thinks]})^{\downarrow 2}, \text{[a warden]}) \vdash}{\text{some}(\text{warden}, \lambda x. \text{think}(x, \text{escape}(y))) : \mathbf{0}_p}}
\end{array}$$

Figure 9: Failed attempt to derive an inverse scope interpretation for (9)

This step of MA is licit because $\downarrow 1$, representing the blocking strength of *ensured*, does not block $\downarrow 1$, representing the escaping strength of *every prisoner*. Thus, the inverse scope interpretation is possible.

By contrast, consider (9), with the (simplified) f-structure shown in (10) and the appropriately instantiated meaning constructors shown in (28).

(9) A warden thinks that every prisoner escaped.

$$(10) \quad \left[\begin{array}{l} \text{PRED} \quad \text{‘think}(\underline{1}, \underline{2})\text{’} \\ \text{SUBJ} \quad \underline{1} \left[\text{‘a warden’} \right] \\ \text{COMP} \quad \underline{2} \left[\begin{array}{l} \text{PRED} \quad \text{‘escape}(\underline{3})\text{’} \\ \text{SUBJ} \quad \underline{3} \left[\text{‘every prisoner’} \right] \end{array} \right] \end{array} \right]$$

$$\begin{aligned}
(28) \quad \text{[a warden]} &:= \lambda P. \text{some}(\text{warden}, P) : (\mathbf{1}_e \multimap_{\circ 3} \mathbf{0}_p) \multimap \mathbf{0}_p \\
\text{[thinks]} &:= \lambda p. \lambda x. \text{think}(x, p) : \mathbf{2}_p \multimap_{\downarrow 2} (\mathbf{1}_e \multimap_{\circ 3} \mathbf{0}_p) \\
\text{[every prisoner]} &:= \lambda P. \text{every}(\text{prisoner}, P) : \forall X. (\mathbf{3}_e \multimap_{\downarrow 1} X_p) \multimap X_p \\
\text{[escaped]} &:= \text{escape} : \mathbf{3}_e \multimap_{\downarrow 1} \mathbf{2}_p
\end{aligned}$$

The surface scope interpretation is derived in an entirely analogous manner to the surface scope interpretation of (13) as shown in Figure 7, with \multimap elimination the only rule of inference used. In order to derive an inverse scope interpretation, one would have to proceed as shown in Figure 9, introducing an auxiliary assumption early in order to be abstracted later. However, if you do that then at some point the derivation cannot proceed, as shown. In order to get an inverse scope interpretation, y would have to be moved to the outside of the premise structure so that it can be abstracted. But this is not possible because the relevant portion of the structure has the schematic form shown in (29).

$$(29) \quad ((\Gamma, \Delta)^{\downarrow 1}, \Sigma)^{\downarrow 2}$$

Mixed associativity cannot apply because $\downarrow 2$, the blocking strength of *thinks*,

blocks $\uparrow 1$, the escaping strength of *every prisoner*. Thus, no inverse scope interpretation is possible.

It should be clear from these examples how the rules round out the SISC: neither *any N*, *every N* nor *no N* can take scope over negation because MA cannot apply to a structure of the form shown in (30), and in fact *no N* cannot take scope from out of the complement of *think* or *ensure* either because MA cannot apply to a structure of the form shown in (31).

$$(30) \quad ((\Gamma, \Delta)^{\uparrow 0/1/2}, \Sigma)^{\downarrow 3}$$

$$(31) \quad ((\Gamma, \Delta), \Sigma)^{\downarrow 1/2}$$

4.2 Reflections

The choice of available modes in this multi-modal Glue system, and the way they interact in the MA rule, are obviously ad-hoc to an extent. As with the SCOPEISLAND features considered in Section 3, these modes can be viewed as placeholders for whatever the comparative strengths of various scope island inducers and escapers turn out to be. I intend to leave open the possibility, for example, that there could be an island inducer stronger than *ensure* but weaker than *think*, or an island escaper stronger than *every N* but weaker than *any N*; I am also open to the possibility that these modes could be predictable from some syntactic or semantic feature.⁹

As a choice of *formal system*, however, multi-modal Glue has one major advantage: given the MA rule and a natural order on the modes (here represented by $\langle \rangle$), the scope island subset constraint follows automatically. Unlike in the system outlined in Section 3, there is no way to give a lexical entry like (22) for a quantifier which can take scope out of the complement of *thinks* but not out of the complement of *ensured*, for example.

On the other hand, it does complicate the underlying logic considerably to move to a multi-modal system, whereas the blocking features-based approach only makes use of established LFG+Glue technology. In the following section we will look at some potential additional motivations for adopting a multi-modal approach.

5 Possible extensions

As alluded to in Section 3, it is an open question what the connection is between the explanations for scope islands and scope freezing. As an example of the latter in English, consider (32), which has a surface scope interpretation but not an inverse scope interpretation, as shown.

$$(32) \quad \text{Every warden checked no prisoner(s).} \\ \Rightarrow \text{every(warden, } \lambda x.\text{not(some(prisoner, } \lambda y.\text{check}(x, y))))$$

⁹My thanks to a reviewer for pressing this point.

$$\frac{
\frac{
y : \mathbf{2}_e \vdash \quad \text{[checked]} \vdash \quad \lambda v. \lambda u. \text{check}(u, v) : \mathbf{2}_e \multimap (\mathbf{1}_e \multimap_{|1} \mathbf{0}_p)
}{
(y : \mathbf{2}_e, \text{[checked]}) \vdash \quad \lambda u. \text{check}(u, y) : \mathbf{1}_e \multimap_{|1} \mathbf{0}_p
}
\quad
\frac{
\vdots \quad \text{[every warden]} \vdash \quad \lambda P. \text{every}(\text{warden}, P) : \quad (\mathbf{1}_e \multimap_{|1} \mathbf{0}_p) \multimap_{|1} \mathbf{0}_p
}{
((y : \mathbf{2}_e, \text{[checked]}), \text{[every warden]})^{\uparrow 1} \vdash \quad \text{every}(\text{warden}, \lambda u. \text{check}(u, y)) : \mathbf{0}_p
}$$

Figure 10: Failed attempt to derive an inverse scope interpretation of (32)

$$\Rightarrow \text{not}(\text{some}(\text{prisoner}, \lambda y. \text{every}(\text{warden}, \lambda x. \text{check}(x, y))))$$

Because there is no embedded clausal f-structure in the f-structure of (32), shown in (33), there is no choice of scope level and hence no way to account for scope freezing in the blocking features-based approach. Both *every warden* and *no prisoner* have to take $\boxed{0}$ as their scope level.

$$(33) \quad \boxed{0} \left[\begin{array}{l} \text{PRED} \quad \text{'check}(\boxed{1}, \boxed{2})\text{' } \\ \text{SUBJ} \quad \boxed{1} \text{ [“every warden”]} \\ \text{OBJ} \quad \boxed{2} \text{ [“no prisoner”]} \end{array} \right]$$

In Gotham (2019) I proposed an account of scope freezing in Glue but, as I mentioned in Section 3, it requires yet another complication of the linear logic fragment used, of a different kind to that discussed in Section 4. Perhaps more seriously, it is not ideally suited to the kind of quantifier-determined scope rigidity exhibited by (32). What I mean by that is that it is not the case in general that direct objects cannot scope over subjects in English—unlike in e.g. German main clauses with canonical SVO order, which is more the point of Gotham (2019). Rather, it seems to be the case that downward-monotonic objects (such as *no N*) cannot scope over upward-monotonic subjects (such as *every N*).

Multi-modal Glue suggests a way we could approach this issue. Look at the proposed meaning constructors for *every* and *no* below, and compare them with those given in Figure 6.

$$\begin{aligned}
\text{every} &\rightsquigarrow \lambda P. \lambda Q. \text{every}(P, Q) : (\uparrow_e \multimap \uparrow_p) \multimap \forall X ((\uparrow_e \multimap_{|1} X_p) \multimap_{|1} X_p) \\
\text{no} &\rightsquigarrow \lambda P. \lambda Q. \text{not}(\text{some}(P, Q)) : (\uparrow_e \multimap \uparrow_p) \multimap \forall X ((\uparrow_e \multimap X_p) \multimap X_p)
\end{aligned}$$

In the meaning constructor shown above, *every* has been given a blocking mode index on its final \multimap . This makes an inverse scope interpretation of (32) unavailable, as shown by the failed attempt to derive one in Figure 10. A premise structure of the general form shown in (34) is created, meaning that MA cannot apply.

$$(34) \quad ((\Gamma, \Delta), \Sigma)^{\uparrow 1}$$

However, this strategy for explaining the non-ambiguity of (32)—of effectively

making *every N* induce a scope island from which *no N* cannot escape—quickly runs into problems. The same structure as (34) would be created in any attempt to derive a *surface* scope interpretation of (35), for example.

(35) No warden checked every prisoner.

The modes in our fragment effectively have two parameters:¹⁰ \uparrow vs. \downarrow to express blocking vs. escaping, and 0–3 to express strength thereof. In order to differentiate between (32) and (35), and allow the *no* $>$ *every* scope order in the latter but not the former, there would need to be an additional parameter keeping track of some relevant property, presumably either argument structure, linear order or c-structure embeddedness.

5.1 Extending the fragment further

Suppose that we have the linear logic fragment as defined in Figure 5, except that we have the expanded list of modes shown in (36), and the definition of blocking for the MA rule is as shown in (37).¹¹ Once again, the use of *i* or *j* in a mode index means that choice of parameter is free.

(36) $a\uparrow 0, b\downarrow 0, c\uparrow 0, d\downarrow 0, a\uparrow 1, b\downarrow 1, c\uparrow 1, d\downarrow 1, a\uparrow 2, b\downarrow 2, c\uparrow 2, d\downarrow 2, a\uparrow 3, b\downarrow 3, c\uparrow 3, d\downarrow 3,$
 $a\downarrow 1, b\uparrow 1, c\downarrow 1, d\uparrow 1, a\downarrow 2, b\uparrow 2, c\downarrow 2, d\uparrow 2, a\downarrow 3, b\uparrow 3, c\downarrow 3, d\uparrow 3$

(37) j blocks $i \Leftrightarrow j = x\downarrow m, i = y\uparrow n, m > n$ and $x < y$.

The idea is to use *a/b/c/d* to encode in the lexical entry for a verb the relative prominence of its arguments. The definition in (37) stipulates that for blocking to occur, the prospective blocker must (in the relevant sense) outrank the relevant escaper on both the alphabetical and numerical parameters. That means we can account for the contrast between (32) and (35) by means of the lexicon shown in Figure 11.^{12,13}

No N then can take scope over *every N* when that corresponds to a surface scope interpretation, e.g. of (35). The crucial inferential step is as shown in (38).

¹⁰This talk of parameters need not be taken literally. In reality, the modes can be simple, with a blocking order defined on them directly. But the notational use of parameters helps with exposition.

¹¹For the purposes of (37), $a < b < c < d$, since alphabetical order is ‘ascending’.

¹²To retain the account of scope islands from Section 4, it follows that the complement argument must be given an alphabetical parameter that outranks every other, as shown in the lexical entry for *thinks* in Figure 11. Therefore, this is not quite the same notion of syntactic rank as expressed in LFG binding theory, although perhaps the definitions could be changed to bring the two notions into line.

¹³The lexical entry given for *not* in Figure 11 is now of type $p \rightarrow p$, and permits, but does not require, a quantifier in subject position to take scope over negation. This can be seen as an improvement over the treatment of negation given in Figure 6.

checked V
 $\lambda y.\lambda x.\text{check}(x, y) : (\uparrow \text{OBJ})_e \multimap_{c|i} ((\uparrow \text{SUBJ})_e \multimap_{b|j} \uparrow_p)$

every D
 $\lambda P.\lambda Q.\text{every}(P, Q) : (\uparrow_e \multimap \uparrow_p) \multimap \forall X((\uparrow_e \multimap_{i|1} X_p) \multimap_{i|1} X_p)$

no D
 $\lambda P.\lambda Q.\text{not}(\text{some}(P, Q)) : (\uparrow_e \multimap \uparrow_p) \multimap \forall X((\uparrow_e \multimap_{i|0} X_p) \multimap_{i|1} X_p)$

warden N
 $\text{warden} : \uparrow_e \multimap \uparrow_p$

prisoner N
 $\text{prisoner} : \uparrow_e \multimap \uparrow_p$

thinks V
 $\lambda p.\lambda x.\text{think}(x, p) : (\uparrow \text{COMP})_p \multimap_{a|2} ((\uparrow \text{SUBJ})_e \multimap_{b|i} \uparrow_p)$

not Adv
 $\text{not} : (\text{ADJ} \in \uparrow)_p \multimap_{b|3} (\text{ADJ} \in \uparrow)_p$

Figure 11: Partial lexical entries for scope freezing

$$(38) \quad \frac{((x : \mathbf{1}_e, [\text{checked}])^{b|0}, [\text{every prisoner}])^{c|1} \text{every}(\text{prisoner}, \lambda y.\text{check}(x, y)) : \mathbf{0}_p \vdash}{(x : \mathbf{1}_e, ([\text{checked}], [\text{every prisoner}])^{c|1})^{b|0} \vdash \text{every}(\text{prisoner}, \lambda y.\text{check}(x, y)) : \mathbf{0}_p} \text{MA}$$

Mixed associativity is applicable in (38) because $c \not< b$ and so blocking does not occur according to the revised definition in (37). By contrast, if we attempt to derive an inverse scope interpretation of (32) then we end up with a premise structure of the form shown in (39), to which MA cannot apply.

$$(39) \quad ((x : \mathbf{1}_e, [\text{checked}])^{c|0}, [\text{every prisoner}])^{b|1} \vdash \text{every}(\text{prisoner}, \lambda y.\text{check}(x, y)) : \mathbf{0}_p$$

Since *every* N outranks *no* N both according to strength ($1 > 0$) and, in this case, according to argument position ($b < c$), MA is blocked.

6 Discussion

The previous section has shown that it is at least feasible for scope islands and scope freezing to both be accounted for using the same formal tools, but it remains to be seen whether or not this is the best approach. It also remains to be seen what the predictions of any specific implementation of this idea might be. For example, languages that are scope rigid in the sense that inverse scope interpretations are disallowed in general, and not just based on the particular quantifiers involved, could be accommodated within the particular formulation of the modes and structural

rules given in (36)–(37) by assigning to every quantifier a meaning constructor of the general form shown in (40).

$$(40) \quad \text{quant} : \forall X((\uparrow_e \multimap_{i|0} X_p) \multimap_{j|1} X_p)$$

This will ensure that the blocking strength of any quantifier will always be greater than the escaping strength of any other, meaning that the argument position parameter alone will be decisive. But it remains to be seen what the implications of this assumption would be for the kind of extra-clausal scope interactions that our discussion began with (‘scope islands’), and whether or not they are borne out.¹⁴ For learnability reasons, our default assumption really ought to be that every language uses the same fragment of linear logic for meaning composition, which makes it crucial to push these kinds of questions early on if we decide that multi-modal Glue is the way to go.

As became progressively clear from Sections 4–5, analysing the data using multi-modal Glue involves incorporating a significant amount of syntactic information into meaning constructors, to the point where many LFG practitioners may feel that we are doing too much categorial grammar within LFG. What I would say to that is that the data force us to do so to some degree, because in Glue meaning constructors are standardly defined based on either f- or s-structure: levels at which certain properties that seem to be crucial for scope possibilities are not defined.

It is still an open question what the best way to account for scope islands in LFG+Glue is, partly because the empirical landscape is not entirely clear, despite decades of work from researchers working in a variety of frameworks. That said, it seems highly likely that the proper explanation for at least some forms of scope rigidity will require a complication of the fragment of linear logic used in Glue beyond simply **LP** (with or without quantification to fix scope level), for the reasons discussed in Section 3 and in Gotham (2019). Just what form that complication should take, though, and what data it should cover, are also open questions.

References

- Andrews, Avery D. 2010. Propositional glue and the projection architecture of LFG. *Linguistics and Philosophy* 33, 141–170.
- Barker, Chris. 2021. Rethinking Scope Islands. *Linguistic Inquiry* Advance publication.
- Charlow, Simon. 2014. *On the semantics of exceptional scope*. Ph.D.thesis, New York University.

¹⁴There is also the issue of how argument position, which this paper has focussed on, conspires with other factors, such as word order and c-structure prominence, to determine what constitutes ‘inverse scope’. In (Gotham 2019, §4.1) this issue was addressed for German by means of a template allowing particular word order configurations to ‘reset’ the scope constraints. Such an approach could be added to multi-modal Glue as well. My thanks to a reviewer for picking up on this.

- Dalrymple, Mary, Lowe, John J. and Mycock, Louise. 2019. *The Oxford Reference Guide to Lexical Functional Grammar*. Oxford: Oxford University Press.
- Dalrymple, Mary, Patejuk, Agnieszka and Zymła, Mark-Matthias. 2020. XLE+Glue – A new tool for integrating semantic analysis in XLE. In Miriam Butt and Ida Toivonen (eds.), *Proceedings of the LFG'20 Conference, On-Line*, pages 89–108, Stanford, CA: CSLI Publications.
- Fry, John. 1999. Proof Nets and Negative Polarity Licensing. In Mary Dalrymple (ed.), *Semantics and Syntax in Lexical Functional Grammar*, pages 91–116, Cambridge, MA: MIT Press.
- Gotham, Matthew. 2019. Constraining Scope Ambiguity in LFG+Glue. In Miriam Butt, Tracy Holloway King and Ida Toivonen (eds.), *Proceedings of the LFG'19 Conference, Australian National University*, pages 111–129, Stanford, CA: CSLI Publications.
- Gotham, Matthew and Haug, Dag Trygve Truslew. 2018. Glue semantics for Universal Dependencies. In Miriam Butt and Tracy Holloway King (eds.), *Proceedings of the LFG'18 Conference, University of Vienna*, pages 208–226, Stanford, CA: CSLI Publications.
- Kokkonidis, Miltiadis. 2008. First-Order Glue. *Journal of Logic, Language and Information* 17(1), 43–68.
- Lev, Iddo. 2007. *Packed Computation of Exact Meaning Representations*. Ph.D.thesis, Stanford University.
- May, Robert. 1977. *The Grammar of Quantification*. Ph.D.thesis, Massachusetts Institute of Technology.
- Moot, Richard and Retoré, Christian. 2012. *The Logic of Categorical Grammars*. Lecture Notes in Computer Science, No. 6850, Berlin/Heidelberg: Springer.
- Rosén, Victoria, De Smedt, Koenraad, Meurer, Paul and Dyvik, Helge. 2012. An open infrastructure for advanced treebanking. In Jan Hajič, Koenraad De Smedt, Marko Tadić and António Branco (eds.), *META-RESEARCH Workshop on Advanced Treebanking at LREC2012*, pages 22–29, Istanbul.