Constraining scope in Glue Semantics: the linear logic approach

Mary Dalrymple
University of Oxford

Jamie Y. Findlay
University of Oslo

Dag T. T. Haug
University of Oslo

Proceedings of the LFG'25 Conference

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

2025

PubliKon

lfg-proceedings.org

Abstract

A hallmark of the Glue approach to the syntax-semantics interface is its clean and simple treatment of quantifiers and quantifier scope ambiguity, without the need to assume different syntactic structures for different scopes. One weakness, which has received renewed attention recently, is Glue's inability to *constrain* such scope ambiguities when they should *not* occur. We propose a new means of encoding constraints on the relative scope of quantifiers, building on Fry's (1997) analysis of negative polarity items. We believe that our proposal also holds promise for the analysis of other constraints on relative scope, including modifier scope and scope islands.

1 Introduction

Glue semantics is the standard theory of the syntax–semantics interface and of semantic composition in LFG (Dalrymple et al. 1993; Dalrymple et al. 2019: ch. 12; Asudeh 2022, 2023; Asudeh & Dalrymple in prep.), though it can be combined with any syntactic theory (Asudeh & Crouch 2001; Frank & van Genabith 2001; Gotham 2018; Haug & Findlay 2023).[†] The basic idea is that words (and phrasal configurations which contribute meaning) encode an association between the meaning which they express and a logical expression which specifies how that meaning can combine with other meanings in the sentence. This logical approach makes Glue very flexible, and enables it to describe syntax–semantics interactions more straightforwardly and/or elegantly than other theories can. In particular, it allows Glue to model quantifier scope ambiguity, as witnessed in sentences like (1), without recourse to an otherwise unmotivated syntactic ambiguity (often brought about via "Quantifier Raising" as in Chomsky 1976, May 1977, 1985, Heim & Kratzer 1998: ch. 7–8, *inter alia*).

(1) *Someone chose everything.*
$\rightsquigarrow$ There is a person who chose everything. $[\exists \gg \forall]$
$\rightsquigarrow$ Everything was chosen by someone. $[\forall \gg \exists]$

The sentence in (1) has two interpretations in English depending on the order in which the quantifiers are combined, as indicated above. In Glue, these two interpretations correspond to two distinct logical proofs which can be constructed from the same lexically-contributed premises (we will see how this works in more detail in the following section).

However, the Glue approach to quantifier scope is sometimes *too* flexible. Although some sentences with two quantifiers, like (1), are ambiguous, others, like (2), are not:

(2) *The teacher gave one student every problem.*
$\rightsquigarrow$ One particular student was given every problem. $[\textit{one} \gg \textit{every}]$
$\not\rightsquigarrow$ Each problem was given to a possibly different student. $[\textit{every} \not\gg \textit{one}]$

In the double object construction in English (and in many languages), when both objects denote quantifiers, they can only be interpreted in the order in which they appear.

[†]We would like to thank the audience at the 2025 LFG conference, the members of the Glue mailing list, the two abstract reviewers, and the two manuscript reviewers for their helpful comments and feedback. Any remaining errors herein are of our own doing.

However, the naïve Glue approach will instead predict that both readings are possible, contrary to fact. We therefore need some way to constrain the general permissiveness of Glue in certain contexts.

In this paper, we offer a solution to this problem from a formal perspective, and provide a technique that allows the theorist to enforce a particular scope relation between any two quantifiers using the existing resources of Glue semantics. We begin in Section 2 with a brief introduction to Glue semantics and a more detailed look at how it handles quantifiers. Section 3 further illustrates the problem of quantifier scope rigidity, before Section 4 presents our solution. Section 5 examines some previous attempts to tackle this problem, and compares them to our approach, before Section 6 concludes.

2 Glue semantics and quantifier scope

As mentioned above, in Glue semantics, the meaning contributions of linguistic objects are given as pairs consisting of a meaning expression (on the left-hand side) and a logical expression (on the right-hand side). Such combinations are known as *meaning constructors*. We will often refer to the left-hand side as the *meaning side* and the right-hand side as the *glue side* or *glue type*. Meaning expressions must support function abstraction and application, but are otherwise unconstrained; for simplicity and generality, we use expressions of predicate logic (with generalized quantifiers) in the following, but many recent Glue analyses use some variant of Compositional Discourse Representation Theory (Muskens 1996). The logical expressions of the glue side are formulated in a fragment of linear logic (Girard 1987), a resource logic which ensures that each meaning contribution is used exactly once in semantic composition.

In (3), the meaning constructor contributed by the proper name *Fred* associates the meaning **fred** with the resource f, and the meaning constructor contributed by the intransitive verb *danced* associates the meaning $\lambda x.\mathbf{dance}(x)$ with the implicational resource $f \multimap d$:[1]

(3) a. $\mathbf{fred} : f$ (meaning constructor for *Fred*)
b. $\lambda x.\mathbf{dance}(x) : f \multimap d$ (meaning constructor for *danced*)

In our LFG setting, f is the f-structure (or its semantic projection) for the subject *Fred*, and d is the f-structure for the sentence *Fred danced*, as shown by the italic labels in the f-structure in (4):

(4) $$_{d}\begin{bmatrix} \textsc{pred} & \text{‘}\textsc{dance}\langle\textsc{subj}\rangle\text{’} \\ \textsc{subj} & _{f}\begin{bmatrix} \textsc{pred} & \text{‘}\textsc{Fred}\text{’} \end{bmatrix} \end{bmatrix}$$ (f-structure for *Fred danced*)

Thus, we can think of the glue (right-hand) side of the verb's meaning constructor as requiring a resource for the subject f, and producing a resource for the sentence d. These meaning constructors serve as premises in a linear logic proof, driven by the glue side. This proof is successful if the result is complete (all of the necessary resources are present) and coherent (we can derive a resource for the sentence with no unused

[1]The symbol $\multimap$ represents linear implication, the correspondent in linear logic of classical logic's material conditional, $\rightarrow$.

resources left over). Logical inference steps on the glue side correspond to operations in the lambda calculus on the meaning side, following the Curry-Howard correspondence (Curry & Feys 1958; Howard 1980). For example, *modus ponens*, a.k.a. implication elimination, corresponds to function application. Using this correspondence, we build up the meaning of the sentence compositionally as a side effect of the logical deduction. Let us illustrate this with a simple example.

For the sentence *Fred danced* we have the proof in (5).[2] Given the meaning constructor $\lambda x.\mathbf{dance}(x) : f \multimap d$ contributed by *danced* and the meaning constructor $\mathbf{fred} : f$ contributed by *Fred*, we can derive, via one instance of implication elimination, the expression $\mathbf{dance}(\mathbf{fred}) : d$, which associates the meaning $\mathbf{dance}(\mathbf{fred})$ with the resource d for the sentence *Fred danced*:

(5)
$$\dfrac{\begin{array}{ll}\textbf{[Fred]} & \textbf{[danced]}\\ \mathbf{fred} : f & \lambda x.\mathbf{dance}(x) : f \multimap d\end{array}}{\mathbf{dance}(\mathbf{fred}) : d}$$

The proof for a transitive sentence proceeds in the same manner, but involves two instances of implication elimination/function application:

(6)
$$\dfrac{\dfrac{\begin{array}{ll}\textbf{[Fred]} & \textbf{[hit]}\\ \mathbf{fred} : f & \lambda x.\lambda y.\mathbf{hit}(x,y) : f \multimap (g \multimap h)\end{array}}{\lambda y.\mathbf{hit}(\mathbf{fred},y) : g \multimap h} \qquad \begin{array}{l}\textbf{[George]}\\ \mathbf{george} : g\end{array}}{\mathbf{hit}(\mathbf{fred},\mathbf{george}) : h}$$

In Section 1, we noted that sentences like (1), repeated here as (7), are ambiguous, which each reading corresponding to a different order of combination for the quantifiers. We represent this explicitly here, using unanalysed meanings for *someone* and *everything*:

(7) *Someone chose everything.*
$\rightsquigarrow \mathbf{someone}(\lambda x.\mathbf{everything}(\lambda y.\mathbf{choose}(x,y)))$ $[\equiv \exists x.\forall y.\mathbf{choose}(x,y)]$
$\rightsquigarrow \mathbf{everything}(\lambda y.\mathbf{someone}(\lambda x.\mathbf{choose}(x,y)))$ $[\equiv \forall y.\exists x.\mathbf{choose}(x,y)]$

We also noted that the standard Glue approach to quantifiers models this semantic ambiguity successfully, without requiring a corresponding syntactic ambiguity. Each scope interpretation instead corresponds to a different proof derived from the same f-structure and the same meaning constructor premises. Let us illustrate how this works.

Quantifiers in Glue are given meaning constructors like those in (8). In words, these meaning constructors look for some dependency on the argument position the quantifier occupies, consume that dependency, and return a meaning corresponding to its output:[3]

[2]We write proofs in natural deduction style, and leave steps corresponding to implication elimination unannotated, but indicate which introduction or elimination rule is being used in other cases. Linguistically contributed meanings are indicated by writing the word or construction introducing them in square brackets.

[3]The universal quantifier $\forall$ on the glue side of the meaning constructors for *someone* and *everything* in (8) quantifies over possible scope structures S, allowing these quantifiers to choose different scope points in a complex sentence, and to choose different relative scopes when there is more than one quantifier; this is not relevant in the examples we discuss, so we could simply fix S as the resource corresponding to the clause, which in our running example is c.

(8) Standard meaning constructors for *someone* and *everything* (to be revised):

a. $\lambda P.\mathbf{someone}(P) : \forall S.(s \multimap S) \multimap S$

b. $\lambda P.\mathbf{everything}(P) : \forall S.(e \multimap S) \multimap S$

To derive a meaning for *Someone chose everything*, we also need an f-structure for the sentence, shown in (9), and the meaning constructor for *chose*, given in (10):

(9)
$${}_{c}\left[\begin{array}{ll}\text{PRED} & \text{‘CHOOSE}\langle\text{SUBJ, OBJ}\rangle\text{’}\\ \text{SUBJ} & {}_{s}\left[\text{PRED ‘SOMEONE’}\right]\\ \text{OBJ} & {}_{e}\left[\text{PRED ‘EVERYTHING’}\right]\end{array}\right]$$

(10) Meaning constructor for *chose*:

$\lambda x.\lambda y.\mathbf{choose}(x,y) : s \multimap (e \multimap c)$

We now have three premises (the two in (8) and the one in (10)), corresponding to each of the three words in the sentence. From these premises, we can construct two distinct proofs of the resource c corresponding to the meaning of the clause. These arise through the use of *hypothetical reasoning*: hypothesising a resource and then subsequently discharging that hypothesis at a later point in the proof, which corresponds to implication introduction on the glue side and lambda abstraction on the meaning side. Using this rule of inferenec, we can construct the first argument of either quantifier from the meaning of the verb, thereby allowing the quantifiers to be combined in either order, ultimately giving the two scope readings. Figures 1–2 show the two proofs.[4]

$$\dfrac{\begin{array}{c}\textbf{[someone]}\\ \mathbf{someone} : (s \multimap c) \multimap c\end{array} \qquad \dfrac{\dfrac{\begin{array}{c}\textbf{[everything]}\\ \mathbf{everything} : (e \multimap c) \multimap c\end{array} \qquad \dfrac{\begin{array}{c}\textbf{[chose]}\\ \lambda x.\lambda y.\mathbf{choose}(x,y) : s \multimap (e \multimap c)\end{array} \qquad [x : s]^1}{\lambda y.\mathbf{choose}(x,y) : e \multimap c}}{\mathbf{everything}(\lambda y.\mathbf{choose}(x,y)) : c}}{\lambda x.\mathbf{everything}(\lambda y.\mathbf{choose}(x,y)) : s \multimap c}\multimap_{I_1}}{\mathbf{someone}(\lambda x.\,\mathbf{everything}(\lambda y.\mathbf{choose}(x,y))) : c}$$

Figure 1: Proof for *Someone chose everything* with wide scope for *someone*

$$\dfrac{\begin{array}{c}\textbf{[everything]}\\ \mathbf{everything} : (e \multimap c) \multimap c\end{array} \qquad \dfrac{\dfrac{\begin{array}{c}\textbf{[someone]}\\ \mathbf{someone} : (s \multimap c) \multimap c\end{array} \qquad \dfrac{\dfrac{\dfrac{\begin{array}{c}\textbf{[chose]}\\ \lambda x.\lambda y.\mathbf{choose}(x,y) : s \multimap (e \multimap c)\end{array} \qquad [x : s]^1}{\lambda y.\mathbf{choose}(x,y) : e \multimap c} \qquad [y : e]^2}{\mathbf{choose}(x,y) : c}}{\lambda x.\mathbf{choose}(x,y) : s \multimap c}\multimap_{I_1}}{\mathbf{someone}(\lambda x.\mathbf{choose}(x,y)) : c}}{\lambda y.\mathbf{someone}(\lambda x.\mathbf{choose}(x,y)) : e \multimap c}\multimap_{I_2}}{\mathbf{everything}(\lambda y.\mathbf{someone}(\lambda x.\mathbf{choose}(x,y))) : c}$$

Figure 2: Proof for *Someone chose everything* with wide scope for *everything*

In Figure 1, we first hypothesise a resource corresponding to the subject argument, s, allowing us to compose this with the meaning for the verb and produce a resource of type $e \multimap c$, which is exactly what the meaning constructor for *everything* is looking for as its scope. Once this meaning has been combined, we then discharge the hypothesis

[4]Hypotheses are decorated with numbers, which are then referred to when annotating implication introduction steps to indicate which hypothesis is being discharged at that point.

we were entertaining, introducing a new dependency on s on the glue side, and lambda abstracting over the variable it corresponds to (x) on the meaning side. Now we have a resource of type $s \multimap c$, which is what the the meaning constructor for *someone* is looking for, and so this applies next. Thus, we introduce the meaning for *everything* before the meaning for *someone*, and the latter therefore takes scope over the former.

For the inverse scope reading, we start in the same way, by hypothesising a resource for the subject, s, but then continue by hypothesising a resource for the object, e, as well. By then immediately discharging the hypothesis of s, we obtain the type $s \multimap c$ resource that *someone* is looking for, and so can combine this with the verb before we combine *everything*. Once we then discharge the hypothesis of e, we obtain the type $e \multimap c$ resource that *everything* is looking for, and complete the proof. This time *someone* was applied first and so *everything* takes wider scope.

The important point to note here is that both scope readings were available simply as a result of the properties of the composition logic, and did not require there to be a distinct syntactic parse for each reading, nor any difference in the meanings of the quantifiers from their use in an intransitive sentence where there is no interaction between different quantifiers.

3 The problem: constraining scope ambiguity

The fact that quantifier scope ambiguity "comes for free" in Glue is generally seen as a strength of the approach, and early work (for example, Dalrymple et al. 1997) took it for granted that it was desirable to always allow all possible quantifier scopes. However, there are in fact many situations where not all quantifier scopes *are* available, and we then need a way to *constrain* the scope possibilities otherwise provided by the glue logic.

Our running example in this paper will be the double-object construction in English. Larson (1990: 603–605) observes that in this construction, between the two complements, only the surface scope reading is available – the OBJ must scope over the $\text{OBJ}_{\text{THEME}}$, and the reverse scope reading is not possible.

(11) *Bertie gave* [*someone*]$_{\text{OBJ}}$ [*everything*]$_{\text{OBJ}_{\text{THEME}}}$.
$\rightsquigarrow$ One particular individual was given everything. $[\exists \gg \forall]$
$\not\rightsquigarrow$ Each thing was given to a possibly different person. $[\forall \not\gg \exists]$

As it stands, the Glue approach predicts that both scope readings should be possible here, contrary to fact. In Section 4 we therefore propose a novel approach to quantifiers in Glue, which gives a very general means of enforcing relative scope between two arbitrary scope-taking items. The goal of our proposal is to provide a way to constrain the relative scope of quantifiers while adding as little extra machinery as possible. Previous approaches to this problem, which we discuss in Section 5, have either been unsuccessful, unduly limited in their coverage, or involve complexifying the underlying logic or grammatical architecture in ways which may be seen as undesirable.

We will restrict our attention in this paper to the double object construction in English for expository ease, but the phenomenon of scope rigidity is widespread. Indeed, it has been noted that there is a general typological trend for languages with freer word order to be concomitantly less permissive of scope ambiguity (Bobaljik & Wurmbrand 2012),[5] albeit this connection is far from absolute.

For example, sentences using the unmarked SOV word order in Japanese exhibit scope rigidity, and can only have the surface scope interpretation where the SUBJ scopes over the OBJ; the inverse scope reading is not available (Hoji 1985):

(12) **SOV:**

Dareka-ga	*daremo-o*	*aisiteiru.*
someone-NOM	everyone-ACC	love

‘Someone loves everyone.’ $[\exists \gg \forall, \forall \not\gg \exists]$

When the alternative OSV word order is used, though, the sentence *is* scopally ambiguous, just like an English transitive sentence with two quantifiers:

(13) **OSV:**

Daremo-o	*dareka-ga*	*aisiteiru.*
everyone-ACC	someone-NOM	love

‘Someone loves everyone.’ $[\exists \gg \forall, \forall \gg \exists]$

A similar pattern obtains in Polish, where Karnowski (2001: 433) claims that SVO word order is unambiguous, with only the surface scope reading allowed, while OVS order is ambiguous, with both surface and inverse scope readings possible:

(14) **SVO:**

(Przynajmniej)	*jeden*	*z*	*germanistów*	*przeczytał*	*każdą*
at.least	one.NOM	of	Germanists.NOM	read	every.ACC

powieść	*Bölla.*
novel.ACC	Böll.GEN

‘(At least) one Germanist read every novel by Böll.’

$\rightsquigarrow$ There is a Germanist who read every Böll novel. $[\exists \gg \forall]$

$\not\rightsquigarrow$ Every Böll novel is such that at least one Germanist has read it. $[\forall \not\gg \exists]$

(15) **OVS:**

Każdą	*powieść*	*Bölla*	*przeczytał*	*(przynajmniej)*	*jeden*	*z*
every.ACC	novel.ACC	Böll.GEN	read	at.least	one.NOM	of

germanistów.
Germanists.NOM

‘(At least) one Germanist read every novel by Böll.’

$\rightsquigarrow$ There is a Germanist who read every Böll novel. $[\exists \gg \forall]$

$\rightsquigarrow$ Every Böll novel is such that at least one Germanist has read it. $[\forall \gg \exists]$

However, judgements about scope possibilities are notoriously delicate, and disagreements over the data are common. In the case of Polish, for example, Szczegielniak (2005: 25) reports different judgements about sentences like (14), and claims that SVO word order *does* give rise to scopal ambiguity, *contra* Karnowski (2001) (and Abels &

[5] In the generativist literature where LF is taken to be the locus of the syntax–semantics interface, such languages are said to “wear their LF on their sleeves” – see e.g. Szabolcsi (1997) on Hungarian.

Grabska 2022 provide suggestive experimental evidence in support of this position).[6]

Individual constructions or even lexical items can also impose scope rigidity. Cross-linguistically, double object constructions very commonly permit only a surface scope interpretation of their two complements, as we saw for English above. There are also several verbs that make "scope islands" out of their complements, preventing quantifiers inside from scoping over quantifiers outside. Gotham (2021: 150–151) gives the following examples highlighting the lexical nature of this phenomenon, since while *think* forms a scope island out of its complement, *ensure* does not:

(16) *A warden thinks that every prisoner escaped.*
$\rightsquigarrow$ There is a warden who thinks that every prisoner escaped. $[\exists \gg \forall]$
$\not\rightsquigarrow$ Every prisoner has a warden who thinks they escaped. $[\forall \not\gg \exists]$

(17) *An accomplice ensured that every prisoner escaped.*
$\rightsquigarrow$ There is an accomplice who ensured that every prisoner escaped. $[\exists \gg \forall]$
$\rightsquigarrow$ Every prisoner has an accomplice who ensured they escaped. $[\forall \gg \exists]$

In fact, it is not simply that some predicates create scope islands and others don't, but rather that the *combination* of certain predicates and certain quantifiers creates scope islands, while other combinations do not (Barker 2022). For example, while the complement of *ensure* is not a scope island for *every*, it is for *no* (Gotham 2021: 151):

(18) *An accomplice ensured that no prisoner escaped.*
$\rightsquigarrow$ There is an accomplice who ensured no prisoner escaped. $[\exists \gg \neg\exists]$
$\not\rightsquigarrow$ No prisoner is such that an accomplice ensured their escape. $[\neg\exists \not\gg \exists]$

And indefinites can escape from the complements of verbs that trap universals (Gotham 2021: 151):

(19) *Every warden thinks that a prisoner escaped.*
$\rightsquigarrow$ Every warden has a prisoner they think escaped. $[\exists \gg \forall]$
$\rightsquigarrow$ There is a prisoner that every warden thinks escaped. $[\forall \gg \exists]$

(This ability of indefinites to scope high is part of the motivation for treating them as distinct from other quantifiers in dynamic theories of semantics like Discourse Representation Theory: Kamp & Reyle 1993; Kamp et al. 2011, so this particular set of facts may or may not be probative here.)

The question of scope rigidity extends beyond quantifiers to other scope-taking items as well. A well-known example is English attributive adjectives, which take scope in the order they appear. An *alleged former gangleader* is different from a *former alleged gangleader*, for example (see Findlay & Haug 2022: sec. 2.1 for more examples and discussion). We will limit our analysis in this paper to quantifiers, but the techniques we develop can be generalised and extended to other cases such as adjectives too (though we leave demonstration of this to future work).

As this section has shown, there are various contexts in which we want to limit the scopal interactions of two quantifiers (or other scope-taking elements). The standard Glue analysis of quantifiers does not predict this, and instead assumes that all logically

[6]We thank Sebastian Zawada for discussion of the Polish data.

possible scope combinations are actually attested. However, we have also seen that the empirical situation is complex, and the basic facts of the matter are often far from settled. For this reason, what we present in this paper is relatively modest on the theoretical front. We offer a formal technique which allows the relative scope of any two quantifiers to be fixed, but we do not attempt to explain why this pattern should obtain in one particular construction and not another.

4 Our solution: licences for scope-taking items

4.1 Background: NPI licensing

Our proposal has its basis in Fry's (1997) analysis of negative polarity items (NPIs) such as *yet* or *anyone*, which are constrained to appear in the scope of negation or another downward-entailing operator which can act as its licensor:

(20) a. *Fred did* $\underline{\textit{not}}$ *dance* $\underline{\underline{\textit{yet}}}$*.* / **Fred danced* $\underline{\underline{\textit{yet}}}$*.*
b. $\underline{\textit{Nobody}}$ *chose* $\underline{\underline{\textit{anyone}}}$*.* / **Fred chose* $\underline{\underline{\textit{anyone}}}$*.*

Fry proposed that the meaning constructors for NPIs require the presence of a licence, ℓ, in the glue logic, and that such a licence is made available within the scope of NPI licensors like *not* or *nobody*. Fry assumes the meaning constructors for *nobody* and *anyone* in (21); we have rewritten his meaning constructors into the 'new glue' format, rather than the 'old glue' format used in Fry (1997).[7]

(21) a. Meaning constructor for *nobody* as NPI licensor:
$\mathbf{nobody} : \forall S.((n \otimes \ell) \multimap (S \otimes \ell)) \multimap S$
b. Meaning constructor for *anyone* as NPI:
$\mathbf{anyone} : \forall S.(a \multimap (S \otimes \ell)) \multimap (S \otimes \ell)$

These share the basic structure of the quantifiers we have already seen, but add extra constraints by conjoining certain resources with an NPI licensing resource, ℓ.[8] According to Fry's analysis, NPI licensors like *nobody* provide this licence within their scope – it is conjoined with both resources within the quantifier's (implicational) argument, which corresponds to the quantifier's scope. This licensing resource must be present in order for an NPI like *anyone* to appear in a well-formed proof: the argument of *anyone* only has ℓ on the right-hand side of the implication, and so cannot provide the resource itself. What is more, because the meaning constructor for *anyone* still has ℓ associated with its output, the licence persists so that a single negative polarity licensor can license more than one NPI in its scope, as in $\underline{\textit{Nobody}}$ *chose* $\underline{\underline{\textit{anyone}}}$ $\underline{\underline{\textit{yet}}}$. By contrast, the licence ℓ does not survive outside the scope of the NPI licensor: in (21a), ℓ is associated with S on the left-hand side of the outermost linear implication $\multimap$, but not on the right-hand

[7]See Asudeh & Dalrymple (in prep.) for an overview and history of notations for meaning constructors and the difference between 'old glue' and 'new glue'. We also give the quantifier meanings in their η-reduced form, postponing questions about the meaning side correspondents of the glue-side licences until Section 4.2.

[8]The symbol $\otimes$ represents multiplicative conjunction, one of the correspondents in linear logic of classical logic's conjunction, $\wedge$.

$$\dfrac{\begin{array}{c}\textbf{[nobody]}\\ ((n\otimes\ell)\multimap(c\otimes\ell))\multimap c\end{array} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[n]^2 \quad \dfrac{[a]^1 \quad \begin{array}{c}\textbf{[chose]}\\ a\multimap n\multimap c\end{array}}{n\multimap c}}{c} \quad [\ell]^3}{c\otimes\ell}\otimes_I}{a\multimap(c\otimes\ell)}\multimap_{I_1} \quad \begin{array}{c}\textbf{[anyone]}\\ (a\multimap(c\otimes\ell))\multimap(c\otimes\ell)\end{array}}{c\otimes\ell} \quad [n\otimes\ell]^4}{c\otimes\ell}\otimes_{E_{2,3}}}{(n\otimes\ell)\multimap(c\otimes\ell)}\multimap_{I_4}}{c}$$

Figure 3: The linear logic side of the proof corresponding to *Nobody chose anyone*, using Fry's NPI-licensing resource, ℓ

side. Thus, the NPI licensor licenses any number of NPIs *within* its scope, but "cleans up" at the end by consuming the licensing resource.

Figure 3 shows the linear logic side of the proof for *Nobody chose anyone*, to illustrate how this works.[9] This proof uses two rules of inference which may not be so familiar to readers, since the use of multiplicative conjunction has fallen out of favour in recent Glue analyses. These are the rules of conjunction introduction and elimination, which correspond on the meaning side to pair construction and pairwise substitution respectively:

(22) **Conjunction introduction:**

$$\dfrac{x : A \qquad y : B}{\langle x, y\rangle : A\otimes B}\otimes_I$$

This first rule says if we have two resources A and B, then we are entitled to derive a new resource $A\otimes B$, while on the meaning side we form a pair out of the two meanings x and y which correspond to A and B respectively.

(23) **Conjunction elimination:**

$$\dfrac{\begin{array}{c}[x : A]^1 \;\; [y : B]^2\\ \vdots\\ f : C\end{array} \qquad \begin{array}{c}\vdots\\ \langle a, b\rangle : A\otimes B\end{array}}{f[x\mapsto a, y\mapsto b] : C}\otimes_{E_{1,2}}$$

This second rule says that if assuming a resource A and a resource B allows us to derive a resource C, then, if we can independently prove $A\otimes B$, we are permitted to derive C. In other words, if we show that having A and having B would allow us to produce C, and then we show that we have A and B, we have shown that we can have C. On the meaning side, this corresponds to pairwise substitution: replacing the meanings x and y associated with our hypothesised resources with the meanings a and b from the pair associated with the actual conjoined resource.

Now, the specific details of this analysis of NPIs is not important for the proposal in this paper. But there is one crucial observation from the proof in Figure 3 which

[9]We omit the meaning sides at this stage to make the proofs easier to follow, since there are a number of complexities which arise, and which we will discuss in Section 4.2.

should be highlighted. The meaning constructor for *nobody* consumes resources which have been conjoined with the licence resource, whereas the meaning constructor for *anyone* produces such a resource. Since we need to eventually eliminate the licenses from the proof, *nobody* must enter the proof after *anyone*, and this therefore accounts for the scope behaviour, whereby *nobody* must outscope *anyone*. This is exactly the same pattern we will make use of in managing quantifier scope more generally.

4.2 Licensing quantifiers: introducing licensors

Our analysis adopts Fry's intuition that the presence of licensors is important in controlling scope relations. We propose that quantifiers all require a licence, but that, in general, they also provide that licence, and so license themselves. However, scope-constraining environments can restrict the availability of the relevant licence, "taking it hostage" and only making it available within another quantifier's scope. In this section, we will show how this can be achieved formally.

Updating the standard meaning constructors for the quantifiers *someone* and *everything* in (8), we propose the revised meaning constructors in (24). Each quantifier makes two contributions: a meaning constructor containing a licensing requirement ℓ^e or ℓ^s specific to each quantifier, and a licence resource of the same form which fulfils that licensing requirement.

(24) Licence-dependent meaning constructors for *someone* and *everything* and their licences:

a. $\mathbf{someone} : \forall S.((s \multimap S) \otimes \ell^s) \multimap S$
$\_ : \ell^s$

b. $\mathbf{everything} : \forall S.((e \multimap S) \otimes \ell^e) \multimap S$
$\_ : \ell^e$

We represent the meaning side of the licence as "_", since it serves only an ancillary role in constraining composition, and will not appear in the final meaning for the sentence.

On the glue side, the first meaning constructors contributed by each quantifier differ from the "standard" meaning constructors given above in (8) only in that there is an additional requirement that the relevant licence be available alongside whatever dependency the quantifier takes as its argument. Since the second meaning contribution of each quantifier *provides* exactly this licence, the cumulative effect is exactly the same as the meaning constructors given in (8), so for now this change gains us nothing. In the next section, however, we will show how this new setup allows us to impose constraints on the relative order of pairs of quantifiers.

Before that, however, we must consider the meaning sides of the quantifiers' meaning constructors. Given their more complicated glue types, we must also complexify their meanings in order to preserve the Curry-Howard correspondence that connects expressions in the glue logic to expressions in the lambda calculus. Specifically, since a conjunction in the linear logic corresponds to a pair on the meaning side, the argument that our quantifier meaning is looking for is now not just a function from individuals to truth values, but rather a pair containing that function as the first element and a licence meaning as the second. Given this, the meaning side of the first meaning constructor in (24a), for example, should be (25a) rather than (25b), where π_1 is the projection function that returns the first element of a pair:

(25) a. $\lambda\mathcal{P}.\textbf{someone}(\pi_1(\mathcal{P}))$
b. $\lambda P.\textbf{someone}(P)$

However, since the licensing machinery disappears in the final meaning, nothing is lost by ignoring this additional complexity throughout, and instead retaining the simpler meaning shown in (25b). This is what we do, therefore, disregarding the meaning contribution of licence resources in proofs, thus saving space and improving readability. Note that this does, however, mean that identical meaning sides will sometimes appear with different glue types, because the differences on the meaning side are being omitted.

We will now switch our running example to the double object construction containing two quantifiers in (11), repeated below as (26):

(26) *Bertie gave someone everything.*
$\rightsquigarrow$ One particular individual was given everything. $[\exists \gg \forall]$
$\not\rightsquigarrow$ Each thing was given to a possibly different person. $[\forall \not\gg \exists]$

Figures 4–5 illustrate the fact that, without further constraints, the new meaning contributions of the quantifiers allow both scopes, just as in standard Glue, even though we only want the surface scope reading. These proofs use the labelled f-structure in (27) and the additional meaning constructors in (28) for *Bertie* and *gave*:[10]

(27) F-structure for *Bertie gave someone everything*:

$$_{g}\begin{bmatrix} \text{PRED} & \text{`GIVE}\langle\text{SUBJ, OBJ, OBJ}_{\text{THEME}}\rangle\text{'} \\ \text{SUBJ} & _{b}\begin{bmatrix}\text{PRED} & \text{`BERTIE'}\end{bmatrix} \\ \text{OBJ} & _{s}\begin{bmatrix}\text{PRED} & \text{`SOMEONE'}\end{bmatrix} \\ \text{OBJ}_{\text{THEME}} & _{e}\begin{bmatrix}\text{PRED} & \text{`EVERYTHING'}\end{bmatrix} \end{bmatrix}$$

(28) Meaning constructors for *Bertie* and *gave*:
a. $\mathbf{b} : b$
b. $\lambda x.\lambda y.\lambda z.\textbf{give}(x, y, z) : b \multimap (s \multimap (e \multimap g))$

4.3 Licensing quantifiers: imposing a licensing constraint

To enforce a constraint on scope ordering, we include a special scope-constraining meaning constructor (SCMC) which requires that the licence of the narrower scoping quantifier is conjoined with the resource corresponding to the wider scoping quantifier. You can think of this meaning constructor as "hijacking" the licence of the narrower scoping quantifier and only making it available within the scope of the wider scoping quantifier. In our running example, we want to enforce the surface scope order *someone* $\gg$ *everything*, so the meaning constructor in question has the following form:

[10]To save space in the proof, we use **b** as the meaning for *Bertie*. We also continue to ignore tense, as we have implicitly been doing throughout. The expression $\textbf{give}(x, y, z)$ is to be interpreted as meaning "x gave y z".

$$
\dfrac{
\begin{array}{l}\textbf{[someone-1]}\\ \mathbf{someone} : ((s \multimap g) \otimes \ell^s) \multimap g\end{array}
\quad
\dfrac{
\begin{array}{c}\textbf{[someone-2]}\\ \_ : \ell^s\end{array}
\quad
\dfrac{
\dfrac{
\begin{array}{l}\textbf{[everything-1]}\\ \mathbf{everything} : ((e \multimap g) \otimes \ell^e) \multimap g\end{array}
\quad
\dfrac{
\begin{array}{c}\textbf{[everything-2]}\\ \_ : \ell^e\end{array}
\quad
\dfrac{
\dfrac{
\begin{array}{l}\textbf{[gave]}\\ \mathbf{give} : b \multimap (s \multimap (e \multimap g))\end{array}
\quad
\begin{array}{l}\textbf{[Bertie]}\\ \mathbf{b} : b\end{array}
}{\mathbf{give}(\mathbf{b}) : s \multimap (e \multimap g)}
\quad
[y : s]^1
}{\mathbf{give}(\mathbf{b})(y) : e \multimap g}
}{\mathbf{give}(\mathbf{b})(y) : (e \multimap g) \otimes \ell^e}\ \otimes_I
}{\mathbf{everything}(\mathbf{give}(\mathbf{b})(y)) : g}
}{\lambda y.\mathbf{everything}(\mathbf{give}(\mathbf{b})(y)) : s \multimap g}\ \multimap_{I_1}
}{\lambda y.\mathbf{everything}(\mathbf{give}(\mathbf{b})(y)) : (s \multimap g) \otimes \ell^s}\ \otimes_I
}{
\dfrac{\mathbf{someone}(\lambda y.\mathbf{everything}(\mathbf{give}(\mathbf{b})(y))) : g}{\mathbf{someone}(\lambda y.\mathbf{everything}(\lambda z.\mathbf{give}(\mathbf{b})(y)(z))) : g}\ =_\eta
}
$$

Figure 4: *Bertie gave someone everything*, no scope constraint – surface scope reading, *someone* $\gg$ *everything*

$$
\dfrac{\begin{matrix}\textbf{[everything-1]}\\ \mathbf{everything} : ((e \multimap g) \otimes \ell^e) \multimap g\end{matrix}
\quad
\dfrac{\begin{matrix}\textbf{[everything-2]}\\ \_ : \ell^e\end{matrix}
\quad
\dfrac{\dfrac{\begin{matrix}\textbf{[someone-1]}\\ \mathbf{someone} : ((s \multimap g) \otimes \ell^s) \multimap g\end{matrix}
\quad
\dfrac{\begin{matrix}\textbf{[someone-2]}\\ \_ : \ell^s\end{matrix}
\quad
\dfrac{\dfrac{\dfrac{\dfrac{\begin{matrix}\textbf{[gave]}\\ \mathbf{give} : b \multimap (s \multimap (e \multimap g))\end{matrix} \quad \begin{matrix}\textbf{[Bertie]}\\ \mathbf{b} : b\end{matrix}}{\mathbf{give}(\mathbf{b}) : s \multimap (e \multimap g)} \quad [y : s]^1}{\mathbf{give}(\mathbf{b})(y) : e \multimap g} \quad [z : e]^2}{\mathbf{give}(\mathbf{b})(y)(z) : g}}{\lambda y.\mathbf{give}(\mathbf{b})(y)(z) : s \multimap g}\multimap_{I_1}}
{\lambda y.\mathbf{give}(\mathbf{b})(y)(z) : (s \multimap g) \otimes \ell^s}\otimes_I}
{\mathbf{someone}(\lambda y.\mathbf{give}(\mathbf{b})(y)(z)) : g}}
{\lambda z.\mathbf{someone}(\lambda y.\mathbf{give}(\mathbf{b})(y)(z)) : e \multimap g}\multimap_{I_2}}
{\lambda z.\mathbf{someone}(\lambda y.\mathbf{give}(\mathbf{b})(y)(z)) : (e \multimap g) \otimes \ell^e}\otimes_I}
{\mathbf{everything}(\lambda z.\mathbf{someone}(\lambda y.\mathbf{give}(\mathbf{b})(y)(z))) : g}
$$

Figure 5: *Bertie gave someone everything*, no scope constraint – unwanted inverse scope reading, *everything* $\gg$ *someone*

(29) **Scope-constraining meaning constructor (SCMC):**

$$\lambda\_.\lambda x.\langle\_, x\rangle \;:\; \underbrace{\ell^e}_{\text{license for narrow scope quantifier}} \multimap (\underbrace{s}_{\text{resource for wide scope quantifier}} \multimap (\ell^e \otimes s))$$

The purpose of this meaning constructor is only to constrain quantifier scope, and it makes no other semantic contribution, so in proofs we will omit the complex meaning side and instead represent it using the symbol ◯, to avoid notational clutter:

(30) **SCMC (abbreviated):**

$$\bigcirc : \ell^e \multimap (s \multimap (\ell^e \otimes s))$$

We will demonstrate shortly how this scope-constraining meaning constructor works, but first we address the question of where it comes from. Put simply, this will vary from phenomenon to phenomenon/construction to construction. Whatever it is that imposes the constraint on scope order, it is precisely by contributing a meaning constructor like (30) that it does so. If we assume that scope rigidity is a property of the double object construction generally, then we can introduce the SCMC via the relevant phrase-structure rule:

(31) $\textsc{OutScopes}(A, B) := \;\; \bigcirc : \ell^B \multimap [A \multimap [\ell^B \otimes A]]$

(32)
$$\begin{array}{lccc} \mathrm{V}' \rightarrow & \mathrm{V} & \mathrm{NP} & \mathrm{NP} \\ & \uparrow = \downarrow & (\uparrow \textsc{obj}) = \downarrow & (\uparrow \textsc{obj}_{\textsc{th}}) = \downarrow \\ & @\textsc{OutScopes}((\uparrow \textsc{obj}), (\uparrow \textsc{obj}_{\textsc{th}})) & & \end{array}$$

But we could also introduce such a constraint lexically, in the entry for a verb like *gave*:

(33) *gave* V $(\uparrow \textsc{pred})$ = ‘give$\langle$SUBJ,OBJ,OBJ$_{\textsc{th}}\rangle$’

$\lambda x.\lambda y.\lambda z.\mathbf{gave}(x, y, z) :$
$(\uparrow \textsc{subj}) \multimap [(\uparrow \textsc{obj}) \multimap [(\uparrow \textsc{obj}_{\textsc{th}}) \multimap \uparrow]]$

$@\textsc{OutScopes}((\uparrow \textsc{obj}), (\uparrow \textsc{obj}_{\textsc{th}}))$

In certain contexts one approach may be more obviously correct than the other. For instance, in the case of scope rigidity arising from word order, presumably a lexical approach would be much more complex than a c-structure-based one. In addition, there is nothing stopping this meaning constructor being associated with other components of the grammar, such as prosody, as well. This might be the right way to capture the effects of stress/focus in disambiguating scopally ambiguous sentences, although that may more properly be interpreted indirectly, via information structure (see e.g. Jackendoff 1972; Kadmon & Roberts 1986; Martí 2001).

However it is introduced to the sentence, the SCMC works the same. Let us see how this works for our running example. With the introduction of the meaning constructor in (30), the licence corresponding to *everything* is bundled together with the resource s associated with *someone*. This means that *everything* now requires both of these resources to be present before it can enter the proof. The s resource can be introduced via hypothetical reasoning, but once that hypothesis is discharged, a dependency on s will be introduced into the proof instead. The only way to remove such a dependency is via the application of the meaning constructor for *someone*, which consumes just such a dependency. So, after the application of the meaning of *everything*, a dependency only resolvable by application of the meaning of *someone* exists. The net result of this is that *everything* must enter the proof before *someone*, and therefore scope under it, or otherwise we will be left with an unresolved dependency on s and no way to remove it – thus, no valid proof. With the introduction of the SCMC, there is therefore now only one successful proof of the meaning of *Bertie gave someone everything*, where *everything* takes narrow scope. This is shown in Figure 6.

5 Alternative proposals for constraining quantifier scope

The need to impose constraints on quantifier scope in a Glue setting has been acknowledged in previous Glue work, and there have been several proposals for how this can be done.

5.1 Constraints on the form of proofs

Crouch & van Genabith (1999) propose to constrain scope order by constraining what counts as an admissible proof. According to their proposal, a node ordering constraint of the form $f{>}g$ requires that no instance of g occurs lower down in the proof than every instance of f, i.e. f outscopes g.

Gotham (2019: 118f.) criticizes Crouch & van Genabith's (1999) approach on two grounds. First, he notes that it relies on a particular (natural deduction-style) representation of proofs – the predicates "higher" and "lower" describe properties of the way natural deduction proofs are written, and are not defined for other styles such as proof nets – and points out that it is undesirable to build into the theory a requirement for proofs to be represented in a particular way. As he stresses (p. 119), "natural deduction derivations are representations of proofs, not the proofs themselves". Second, he notes that the approach in question implements a non-monotonic generate-then-filter strategy, where all proofs are generated, and then some are discarded as not of interest, despite being well-formed. With Gotham, we conclude that Crouch & van Genabith's (1999) proposal is not a desirable augmentation of Glue; see Gotham (2019: 118f.) for more detailed discussion.

5.2 Hard-coded scope orders

Gotham (2019) makes an alternative proposal to constrain quantifier scope, which involves expanding the fragment of linear logic that is used by adding a new type of term (integers) and a new function symbol, s, representing the successor function. To some extent, our proposal also expands the fragment of linear logic from the purely implicational fragment often assumed in modern work, by also including the ⊗ connective. However, we do note that this connective is a standard part of linear logic, and has previously been used in Glue analyses, e.g. of anaphora (such as Dalrymple 2001: ch. 11).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{l}\textbf{[everything-2]}\\ \_ : \ell^e\end{array} \quad \begin{array}{l}\textbf{[SCMC]}\\ \bigcirc : \ell^e \multimap (s \multimap (\ell^e \otimes s))\end{array}}{\bigcirc : s \multimap (\ell^e \otimes s)} \quad [y : s]^1}{\bigcirc : \ell^e \otimes s} \quad \dfrac{\dfrac{\dfrac{\begin{array}{l}\textbf{[gave]}\\ \mathbf{gave} : b \multimap (s \multimap (e \multimap g))\end{array} \quad \begin{array}{l}\textbf{[Bertie]}\\ \mathbf{b} : b\end{array}}{\mathbf{gave}(\mathbf{b}) : s \multimap (e \multimap g)} \quad [y : s]^2}{\mathbf{gave}(\mathbf{b})(y) : e \multimap g} \quad [\_ : \ell^e]^3}{\mathbf{gave}(\mathbf{b})(y) : (e \multimap g) \otimes \ell^e}\otimes_I}{\mathbf{gave}(\mathbf{b})(y) : (e \multimap g) \otimes \ell^e}\otimes_{E_{2,3}} \quad \begin{array}{l}\textbf{[everything-1]}\\ \mathbf{everything} : ((e \multimap g) \otimes \ell^e) \multimap g\end{array}}{\mathbf{everything}(\mathbf{gave}(\mathbf{b})(y)) : g}}{\lambda y.\mathbf{everything}(\mathbf{gave}(\mathbf{b})(y)) : s \multimap g}\multimap_{I_1} \quad \begin{array}{l}\textbf{[someone-2]}\\ \_ : \ell^s\end{array}}{\lambda y.\mathbf{everything}(\mathbf{gave}(\mathbf{b})(y)) : (s \multimap g) \otimes \ell^s} \quad \begin{array}{l}\textbf{[someone-1]}\\ \mathbf{someone} : ((s \multimap g) \otimes \ell^s) \multimap g\end{array}}{\mathbf{someone}(\lambda y.\mathbf{everything}(\mathbf{gave}(\mathbf{b})(y))) : g}}{\mathbf{someone}(\lambda y.\mathbf{everything}(\lambda z.\mathbf{gave}(\mathbf{b})(y)(z))) : g}=_\eta$$

Figure 6: Proof for *Bertie gave someone everything* with licences and scope constraint forcing *someone* ≫ *everything*

In Gotham's (2019) proposal, the linear logic resources on the glue side of meaning constructors are redefined as predicates of numerical terms, which bear some conceptual similarity to the licensing resources which we have proposed, in that their purpose is to constrain the proof space. A predicate can specify the relative scope of its arguments by requiring the resource for one of its arguments to be associated with the same numerical term as itself: for example, if a transitive verb specifies that the object resource and the resource for the verb are associated with the same variable, then the object must take narrow scope relative to the subject. In this way, restrictions on the relative scope of arguments are hard-coded into the meaning constructor of the predicate that selects them.

Gotham's (2019) proposal adds terms to the derivation whose purpose is to eliminate proofs of unwanted scopings, using the standard rules of linear logic. However, the approach has limitations. Though it can successfully constrain the relative scope of the arguments of a two-place predicate, Gotham (2019: 123–126) points out that it must be augmented with additional constraints if scope possibilities also depend on other factors such as linear order, and the logic itself must be further expanded with additional function symbols to handle scope relations involving arguments of a predicate with arity greater than two. Additionally, it is not clear that his proposal could (straightforwardly) handle scoping constraints that do not involve arguments of the same predicate, as in (18), where the two quantifiers of interest are arguments of two different predicates. By contrast, our approach simply expresses a scope ordering between any two quantifiers, quite independently of their relation to any predicate, the number of arguments their governor might have, or of their argument/adjunct status. Thus, we conclude that Gotham's (2019) approach is insufficiently general in comparison to the proposal presented in this paper.

5.3 Multistage proving

Other recent work also allows for the imposition of constraints on the scope of operators, but in a rather different way. Findlay & Haug (2022) propose a new level of representation in the projection architecture which groups meaning premises together to manage interactions between them. Zymla (2024) builds on this work in a computational implementation. In their approach, premises are grouped together in a structure called *proof structure*, which imposes limits on the interactions between the premises. In essence, the proof structure partitions the semantic resources into disjoint sets. Inside each set, no scope constraints are imposed: a proof is constructed from the premises according to the normal inference rules of linear logic. But the sets themselves are ordered, so that no resource from a lower set can scope over a resource in a higher set.

In this setup, it is easy to impose *total* scope islands. For example, if we want to impose the constraint that some clause is an absolute scope island, we make the proof structure group all the resources coming from material in that clause, forcing all that material to combine together before it interacts with the material governing that clause (see Findlay & Haug 2022: 154–156 for an example). It is not trivial to impose such a constraint with the licensing machinery described in this paper, though it is possible.

On the other hand, there is no very natural way to impose *pairwise* ordering constraints in the multistage proving approach of Findlay & Haug (2022). This would be important in a situation where we had at least three scope-taking elements, but where the relative ordering was only fixed between two of them, with the third item free to scope at any point. Imagine, for example, that we have a sentence with three scope-taking elements Q_1, Q_2, Q_3 and a verb V, and we want to impose the constraint that Q_2 outscopes Q_3 but that otherwise any scoping is permissible (i.e. we can have the order $Q_1 \gg Q_2 \gg Q_3$, the order $Q_2 \gg Q_1 \gg Q_3$, or the order $Q_2 \gg Q_3 \gg Q_1$). It is possible to achieve this effect in multistage proving by using disjunctive descriptions of proof structure, such that the sentence is associated with two distinct proof structures (either $\{Q_2, \{Q_1, Q_3, V\}\}$ or $\{Q_1, Q_2, \{Q_3, V\}\}$). But this is less natural than simply stating a constraint between Q_2 and Q_3 only, as can be done with the licensing approach described here.

Whether this is an advantage of our approach then depends on whether we find such situations in natural language. This empirical question is not at all straightforward to answer, since, as we saw above, judgements can already be quite fragile when just two scope-taking items are involved, and here we must necessarily have at least three. But here is a potential example of the structure in question:

(34) *Bertie didn't give someone everything.*

This is the negated version of our running example, where we know that the direct object *something* must outscope the secondary object *everything* ($\exists \gg \forall$). We have not discussed negation in this paper, but it too involves a scope-taking element ($\neg$) which can scope over or under other operators, and our treatment of it will be very similar to how we handle quantifiers. We suggest that (34) illustrates exactly the pattern described in the previous paragraph, with a fixed scope order between *something* and *everything*, but no such constraint on the negation, which can then scope at any point relative to the two quantifiers. This gives rise to three distinct readings (each of which can be brought out by pronouncing the sentence with a particular stress pattern):

(35) a. It's not the case that Bertie gave someone everything. $[\neg \gg \exists \gg \forall]$
(*Bertie* DIDN'T *give someone everything.*)
b. There's someone Bertie didn't give everything too. $[\exists \gg \neg \gg \forall]$
(*Bertie didn't give* SOMEONE *everything.*)
c. There's someone Bertie gave nothing to. $[\exists \gg \forall \gg \neg]$
(*Bertie didn't give someone* EVERYTHING.)

We leave to further research the difficult task of clarifying the empirical facts more rigorously, e.g. via behavioural or neurophysiological experiments, but if contrasts such as those described in (35) are valid, then the licensing approach offers a more natural explanation than the multistage proving approach does.

Before we leave the topic of multistage proving, it should be noted that Gotham (2021) presents a system where he achieves an effect similar to that of multistage proving by restricting the associativity of the logic, i.e. by not allowing in the general case the inference from $(\Gamma, (\Delta, \Sigma))$ to $((\Gamma, \Delta), \Sigma)$, thereby forcing Δ, Σ to combine first. Instead, brackets are assigned ordered mode indices (via indices on the implication connective) and rebracketing is allowed only when the inner bracket's mode is stronger than that of the outer bracket. Gotham's (2021) approach does have the advantage of being able to capture the interaction of islands and escapers of different strengths, as mentioned above (see Barker 2022), which is not straightforwardly possible in the approach of Findlay & Haug (2022). Nevertheless, we feel his general strategy is guilty of "generalising to the worst case". In the approach of Findlay & Haug (2022) and that of the present paper, associativity, i.e. scope ambiguity, is assumed to be present unless explicitly excluded; for Gotham (2021), the situation is reversed: every interaction of quantifiers is mediated by the mode numbers, and so any scope ambiguity must be explicitly licensed. We do not feel this is correct, but concede this may amount to little more than aesthetics. More substantively, Gotham's (2021) approach, just like the multistage proving approach of Findlay & Haug (2022), has no natural way to state arbitrary pairwise scope constraints.

6 Conclusion

Glue has the desirable property of modelling quantifier scope ambiguities as arising at the syntax–semantics interface itself, rather than requiring an otherwise unmotivated syntactic ambiguity. However, we do not always want to allow all such scope orderings – some constructions restrict them in various ways. In this paper, we have shown that vanilla (LFG+)Glue is in fact able to enforce arbitrary pairwise scope orderings, without modifications to the proof algorithm or underlying logic (save for the use of multiplicative conjunction, $\otimes$), and without complexification of the grammatical architecture. This makes it possible to impose constraints on scope ambiguity on a construction-by-construction basis.

References

Abels, Klaus & Dagmara Grabska. 2022. On the distribution of scope ambiguities in Polish. *Glossa: a journal of general linguistics* 7(1). https://doi.org/10.16995/glossa.8170.

Asudeh, Ash. 2022. Glue Semantics. *Annual Review of Linguistics* 8. 321–341. https://doi.org/10.1146/annurev-linguistics-032521-053835.

Asudeh, Ash. 2023. Glue semantics. In Mary Dalrymple (ed.), *The handbook of Lexical Functional Grammar* (Empirically Oriented Theoretical Morphology and Syntax 13), 651–697. Language Science Press. https://doi.org/10.5281/zenodo.10185964.

Asudeh, Ash & Richard Crouch. 2001. Glue semantics for HPSG. In Frank Van Eynde, Lars Hellan & Dorothee Beermann (eds.), *Proceedings of the 8th International Conference on Head-Driven Phrase Structure Grammar*, 1–19. Stanford: CSLI Publications. http://cslipublications.stanford.edu/HPSG/2/hpsg01.html.

Asudeh, Ash & Mary Dalrymple. In prep. Glue semantics. In *Handbook of linguistic semantics: Bridging theory, philosophy and cognition*, Springer Nature.

Barker, Chris. 2022. Rethinking scope islands. *Linguistic Inquiry* 53(4). 633–661. https://doi.org/10.1162/ling_a_00419.

Bobaljik, Jonathan David & Susi Wurmbrand. 2012. Word order and scope: transparent interfaces and the 3/4 signature. *Linguistic Inquiry* 43(3). 371–421. https://muse.jhu.edu/article/481121.

Chomsky, Noam. 1976. Conditions on rules of grammar. *Linguistic Analysis* 2. 303–351.

Crouch, Richard & Josef van Genabith. 1999. Context change, underspecification and the structure of Glue language derivations. In *Semantics and syntax in Lexical Functional Grammar*, 117–189. MIT Press.

Curry, Haskell B. & Robert Feys. 1958. *Combinatory logic: volume I*. Amsterdam: North Holland.

Dalrymple, Mary. 2001. *Lexical Functional Grammar* (Syntax and Semantics 34). San Diego, CA: Academic Press.

Dalrymple, Mary, John Lamping, Fernando C. N. Pereira & Vijay A. Saraswat. 1997. Quantifiers, anaphora, and intensionality. *Journal of Logic, Language and Information* 6(3). 219–273. https://doi.org/10.1023/A:1008224124336.

Dalrymple, Mary, John Lamping & Vijay Saraswat. 1993. LFG semantics via constraints. In Steven Krauwer, Michael Moortgat & Louis des Tombe (eds.), *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics (EACL 1993)*, 97–105. Association for Computational Linguistics. https://doi.org/10.3115/976744.976757.

Dalrymple, Mary, John J. Lowe & Louise Mycock. 2019. *The Oxford Reference Guide to Lexical Functional Grammar*. Oxford: Oxford University Press. https://doi.org/10.1093/oso/9780198733300.001.0001.

Findlay, Jamie Y. & Dag T. T. Haug. 2022. Managing scope ambiguities in Glue via multistage proving. In Miriam Butt, Jamie Y. Findlay & Ida Toivonen (eds.), *Proceedings of the LFG'22 Conference*, 143–163. Stanford, CA: CSLI Publications. https://lfg-proceedings.org/lfg/index.php/main/article/view/18.

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 LFG'01 Conference*, 104–126. Stanford, CA: CSLI Publications. https://typo.uni-konstanz.de/lfg-proceedings/LFGprocCSLI/LFG2001/pdfs/lfg01frankgenabith.pdf.

Fry, John. 1997. Negative polarity licensing at the syntax–semantics interface. In *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics (ACL-EACL 97)*, 144–150. Madrid: Association for Computational Linguistics. https://doi.org/10.3115/976909.979636.

Girard, Jean-Yves. 1987. Linear logic. *Theoretical Computer Science* 50(1). 1–102. https://doi.org/10.1016/0304-3975(87)90045-4.

Gotham, Matthew. 2018. Making Logical Form type-logical: Glue semantics for Minimalist syntax. *Linguistics & Philosophy* 41(5). 511–556. https://doi.org/10.1007/s10988-018-9229-z.

Gotham, Matthew. 2019. Constraining scope ambiguity in LFG+Glue. In Miriam Butt, Tracy Holloway King & Ida Toivonen (eds.), *Proceedings of the LFG'19 Conference*, Stanford, CA: CSLI Publications. https://typo.uni-konstanz.de/lfg-proceedings/LFGprocCSLI/LFG2019/lfg2019-gotham.pdf.

Gotham, Matthew. 2021. Approaches to scope islands in LFG+Glue. In Miriam Butt, Jamie Y. Findlay & Ida Toivonen (eds.), *Proceedings of the LFG'21 Conference*. Stanford, CA: CSLI Publications. https://typo.uni-konstanz.de/lfg-proceedings/LFGprocCSLI/LFG2021/lfg2021-gotham.pdf.

Haug, Dag T. T. & Jamie Y. Findlay. 2023. Formal semantics for Dependency Grammar. In *Proceedings of the Seventh International Conference on Dependency Linguistics (Depling 2023)*, 22–31. Association for Computational Linguistics. https://aclanthology.org/2023.depling-1.3/.

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

Hoji, Hajime. 1985. *Logical form constraints and configurational structures in Japanese*. Ph.D. thesis, University of Washington.

Howard, William A. 1980. The formulae-as-types notion of construction. In *To H. B. Curry: essays on combinatory logic, lambda calculus, and formalism*, 479–490. London: Academic Press. Circulated in unpublished form from 1969.

Jackendoff, Ray S. 1972. *Semantic interpretation in generative grammar* (Current Studies in Linguistics). Cambridge, MA: MIT Press.

Kadmon, Nirit & Craige Roberts. 1986. Prosody and scope: the role of discourse structure. In Anne M. Farley, Peter T. Farley & Karl-Erik McCullough (eds.), *Proceeding from the 22nd meeting of the Chicago Linguistics Society [Part 2]: Papers from the parasession on pragmatics and grammatical theory*, 16–28. Chicago, IL: Chicago Linguistic Society.

Kamp, Hans, Josef van Genabith & Uwe Reyle. 2011. Discourse Representation Theory. In Dov M. Gabbay & Franz Guenthner (eds.), *Handbook of philosophical logic* (2nd edn.), vol. 15, 125–394. Berlin: Springer.

Kamp, Hans & Uwe Reyle. 1993. *From discourse to logic*. Dordrecht: Kluwer.

Karnowski, Pawel. 2001. Zum relativen Quantorenskopus im Polnischen. In Gerhild Zybatow, Uwe Junghanns, Grit Mehlhorn & Luka Szucsich (eds.), *Current issues in formal Slavic linguistics* (Linguistik International 5), 426–435. Berlin: Peter Lang.

Larson, Richard K. 1990. Double objects revisited: reply to Jackendoff. *Linguistic Inquiry* 21(4). 589–632. https://www.jstor.org/stable/4178697.

Martí, Luisa. 2001. Intonation, scope, and restrictions on quantifiers. In Karine Megerdoomian & Leora Anne Bar-el (eds.), *Proceedings of the 20th West Coast Conference on Formal Linguistics (WCCFL 20)*, 372–385. Somerville, MA: Cascadilla Press.

May, Robert. 1977. *The grammar of quantification*. Ph.D. thesis, MIT.

May, Robert. 1985. *Logical form: its structure and derivation* (Linguistic Inquiry Monographs 12). Cambridge, MA: MIT Press.

Muskens, Reinhard. 1996. Combining Montague Semantics and discourse representation. *Linguistics and Philosophy* 19(2). 143–186. https://doi.org/10.1007/bf00635836.

Szabolcsi, Anna. 1997. Strategies for scope taking. In Anna Szabolcsi (ed.), *Ways of scope taking*, 109–154. Dordrecht: Kluwer.

Szczegielniak, Adam. 2005. *Relativization that you did ...* (MIT Occasional Papers in Linguistics 24). Cambridge, MA: MIT Press.

Zymla, Mark-Matthias. 2024. Ambiguity management in computational Glue semantics. In Miriam Butt, Jamie Y. Findlay & Ida Toivonen (eds.), *Proceedings of the LFG'24 Conference*, 285–310. Konstanz: PubliKon. https://lfg-proceedings.org/lfg/index.php/main/article/view/59.