Managing scope ambiguities in Glue via multistage proving

Jamie Y. Findlay
University of Oslo

Dag T. T. Haug
University of Oslo

Proceedings of the LFG'22 Conference

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

2022

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

It is well known that vanilla LFG+Glue overgenerates when it comes to scope ambiguities. This has both theoretical and practical implications: it generates unattested readings, and also produces many spurious ambiguities which do not correspond to linguistically relevant differences. We propose a solution in keeping with the LFG philosophy: a new level of the parallel projection architecture, called *proof-structure*, which can be used to limit scopal interactions. Unlike earlier attempts to solve this problem, the modularity of our approach means that we do not need to alter the underlying linear logic or proof algorithm of Glue Semantics, with the effect that existing analyses and tools continue to be relevant and usable.

1 Introduction

Glue Semantics (Glue: Dalrymple et al. 1993, Dalrymple 1999, Asudeh 2022) is the *de facto* standard theory of the syntax-semantics interface in LFG (see e.g. Dalrymple et al. 2019, ch. 8). Glue offers elegant analyses of a number of phenomena such as anaphora (Dalrymple et al. 1999, Asudeh 2005, Belyaev and Haug 2014), tense and aspect (Haug 2008, Bary and Haug 2011), and argument structure/argument realisation (Asudeh and Giorgolo 2012, Findlay 2016, 2020). But it is its unique approach to quantification (Dalrymple et al. 1999) for which Glue is perhaps best known: in Glue there is no syntactic ambiguity associated with scope ambiguity (there is no operation of Quantifier Raising, for instance). Rather, different scopes correspond to the different proofs which can be obtained from the same set of meaning constructor premises.

However, despite its strengths, this approach to scope ambiguities is ultimately too permissive: it overgenerates, producing both unattested readings and large numbers of spurious ambiguities which turn out to be logically equivalent. Existing proposals aiming to solve this problem involve either complexifying the linear logic component of Glue (Gotham 2019, 2021), or modifying the algorithm which delivers the linear logic proofs (Crouch and van Genabith 1999).[1] In this paper, we present a different solution: in keeping with the modular philosophy of LFG, we introduce a new level of representation, which we call *proof-structure*, where certain structural constraints on the linear logic proof can be enforced. This enables us to put limitations on composition that restrict the number of readings available, and to do so in linguistically-principled ways. It also means that we leave the logic

[†]We thank audiences at the 32nd SE-LFG meeting and at the LFG'22 conference for their feedback, especially Ash Asudeh, Miriam Butt, Mary Dalrymple, and Davide Mocci. This work was supported by the Research Council of Norway, grant number 300495 "Universal Natural Language Understanding", which we gratefully acknowledge.

[1]Andrews (2018, 141f.) also offers a solution to a subset of the problems discussed in this paper, but this a) is not as general a solution as the others mentioned in the text, or what we propose below; and b) involves recapitulating c-structure information at f-structure, thus violating the modular philosophy of LFG's parallel projection architecture.

itself and the proof algorithm untouched, which has the advantage that existing analyses and tools (such as the Glue Semantics Workbench – Meßmer and Zymla 2018) remain valid and usable.

We begin, in Section 2, by presenting four scope-related problems for Glue. Section 3 introduces proof-structure, and Section 4 shows how it can be used to solve these problems. Finally, Section 5 addresses some technical questions of implementation, and shows how our proposal still permits linguistic analysis to be viewed declaratively and not procedurally.

2 The problems

2.1 Modifier scope

In many languages, including English, the order in which modifiers like attributive adjectives appear determines their relative scope. For example, a nominal containing two intensional adjectives, like *alleged former racketeer*, is interpreted with the first adjective having scope over the second:

(1) $[\![\text{alleged former racketeer}]\!]$ (Andrews and Manning 1999)
$= \mathbf{alleged}(\mathbf{former}(\mathbf{racketeer}))$
$\neq \mathbf{former}(\mathbf{alleged}(\mathbf{racketeer}))$

That is, an alleged former racketeer is someone alleged to be a former racketeer, not someone formerly alleged to be a racketeer.

The same ordering effects are observed when we mix intensional and intersective adjectives:[2]

(2) $[\![\text{counterfeit American coin}]\!]$ (Campbell 2002)
$= \mathbf{counterfeit}(\mathbf{american}(\mathbf{coin}))$
$\neq \mathbf{american}(\mathbf{counterfeit}(\mathbf{coin}))$

(3) $[\![\text{trustworthy former chairman}]\!]$ (Lowe 2015, 441)
$= \mathbf{trustworthy}(\mathbf{former}(\mathbf{chairman}))$
$\neq \mathbf{former}(\mathbf{trustworthy}(\mathbf{chairman}))$

Example (2) refers to a counterfeit instance of an American coin, and not, say, to an American attempt at counterfeiting some other currency; while (3) refers to a trustworthy person who used to be a chairman, not someone who was formerly a trustworthy chairman.

[2] It is notoriously difficult to find any adjectives that are truly and unambiguously intersective. Here we use *American* and *trustworthy* (and, below, *Scottish*); to the extent that *any* adjectives are genuinely intersective, we feel these are good representatives, but the reader should feel free to replace them with others if they disagree. In fact, nothing we say here requires there to be a real distinction between intensional and intersective adjectives; as we point out below, the apparent lack of an ordering effect in sentences involving multiple intersective adjectives, like (4), follows from the meanings of the adjectives, not anything structural – a position which is wholly compatible with both adjectives being intensional to some greater or lesser extent.

When both adjectives are intersective, order does not appear to impose such a scopal restriction:

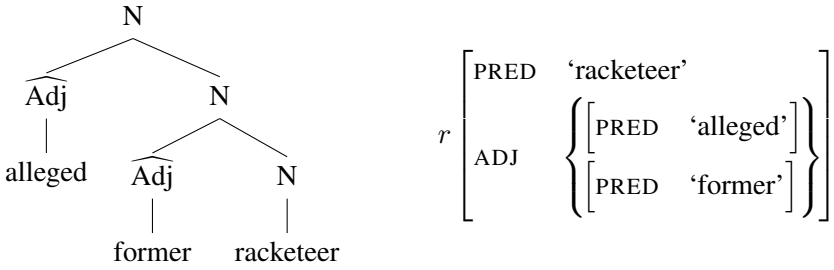
(4) $[\![\text{trustworthy Scottish chairman}]\!]$
$= \mathbf{trustworthy}(\mathbf{scottish}(\mathbf{chairman}))$
$= \mathbf{scottish}(\mathbf{trustworthy}(\mathbf{chairman}))$

However, this is because the two different scopes are logically equivalent: since intersection is represented in most meaning languages via conjunction, and conjunction is commutative, it makes no difference in which order we combine two intersective adjectives. In other words, since the equivalence in (5) holds, we cannot really infer anything about the order of composition in (4): it could be either way round and we would be unable to tell by considering the meaning, since both are equivalent.

(5) $\mathbf{trustworthy}(\mathbf{scottish}(\mathbf{chairman})) \Leftrightarrow \mathbf{scottish}(\mathbf{trustworthy}(\mathbf{chairman}))$

The basic observations regarding word order and scope are handled straightforwardly in a theory where semantic composition is based on the phrase-structure tree: semantic scope can be made to follow directly from c-structure scope. But this is not possible 'out of the box' in LFG+Glue, since Glue meaning constructors are connected to the syntax via f-structure, not c-structure, and f-structure flattens the relevant scopal relationships:[3]

(6) C- and f-structure for *alleged former racketeer*:



Given the f-structure in (6), we would end up with the following instantiated meaning constructors for this phrase:

[3]In many versions of LFG+Glue (e.g. that presented in Dalrymple et al. 2019, ch. 8), meaning constructors are hooked up to s(emantic)-structures, not f-structures, but this has no bearing on the problem presented here, since s-structure is itself a projection of f-structure, and so cannot recover information that is not recoverable at f-structure. In this paper, we make use of *first-order Glue* (Kokkonidis 2008), whereby the linear logic types are provided by unary predicate type constructors which take structures (in this case f-structures) as their arguments. This provides for more readable meaning constructors, and removes the need for otherwise unmotivated s-structure attributes like VAR and RESTR (Kokkonidis 2008, 63f.; Findlay 2019, 182f.). Nothing hinges on this choice, however.

(7) $\mathbf{racketeer} : (e_r \multimap t_r)$
$\mathbf{alleged} : (e_r \multimap t_r) \multimap (e_r \multimap t_r)$
$\mathbf{former} : (e_r \multimap t_r) \multimap (e_r \multimap t_r)$

But these impose no constraints on the order in which the second and third meaning constructors apply to the first, so from the premises in (7), we can obtain proofs of *both* scopal interpretations, i.e. both (8a) and (8b), thereby overgenerating:

(8) a. $\mathbf{former}(\mathbf{alleged}(\mathbf{racketeer}))$
b. $\mathbf{alleged}(\mathbf{former}(\mathbf{racketeer}))$

2.2 Scope freezing

From the modifier scope data, we might think that the problem is that c-structure information needs to be visible at f-structure (this is the motivation behind Andrews's 2018 solution mentioned in fn. 1, for example). But the scopal structure of a sentence does not simply recapitulate c-structure scope; it can also introduce different/additional structure. This is apparent in so-called *scope freezing*. Some constructions, such as the English double object construction (David Lebeaux *apud* Larson 1990, 603) or German SVO clauses (Gotham 2019, 115), allow only one scopal interpretation, unlike other, similar constructions in these languages.

For example, (9) can only mean that there is a student whom Hilary gave every grade to (where the existential scopes over the universal quantifier, $\exists > \forall$), not that every grade is such that it was given to at least one student ($\forall > \exists$):

(9) Hilary gave a student every grade.

But of course in English simple transitive sentences, such scope ambiguity does occur. (10) is ambiguous between precisely the two scenarios described above:[4]

(10) A student received every grade.

In German, we observe a similar pattern in the difference between SVO and OVS word orders. (11) has only one scopal interpretation, where there is at least one officer who is individually responsible for guarding every exit him/herself ($\exists > \forall$). By contrast, (12) is ambiguous and permits the reading where each exit is such that a (potentially different) officer guards it ($\forall > \exists$).

(11) Ein Polizist bewacht jeden Ausgang.
*a.*NOM *police.officer guards every.*ACC *exit*
'A police officer guards every exit.'

(12) Jeden Ausgang bewacht ein Polizist.
*every.*ACC *exit guards a.*NOM *police.officer*

[4]Note that here c-structure scope explicitly does *not* constrain quantifier scope: the subject NP outscopes the object at c-structure in (10), but this is irrelevant for semantic interpretation. This once again shows that semantic scope is orthogonal to c-structure scope, and not simply reducible to it.

Once again, standard LFG+Glue has no way of enforcing a constraint like scope freezing, since the f-structures for all these examples will be flat, with no scopal relationship between the arguments (and the f-structures of (11) and (12) will in fact be identical).

2.3 Scope islands

Some combinations of embedding verbs and quantifiers can result in subordinate clauses becoming what are called *scope islands* (Barker 2022); quantifiers inside such islands cannot scope outside of them. For example, in (13), the universal quantifier introduced by *every* cannot scope over the existential quantifier introduced by *a* in the higher clause:

(13) A warden thinks [that every prisoner escaped]. (Gotham 2021, 150)

a. = There is a warden who thinks that every prisoner escaped.
($\exists > \forall$)

b. $\neq$ For every prisoner, there is a warden who thinks they escaped.
($^{*}\forall > \exists$)

This is not determined by syntactic structure alone, but rather by poorly understood interactions between embedding operators and the quantifiers they embed. For instance, (14) has apparently exactly the same syntactic structure as (13), and involves the same two quantifiers, but contains a different embedding verb, and here the universal *can* outscope the existential, 'escaping' from the downstairs clause, which is evidently no longer an island:

(14) An accomplice ensured [that every prisoner escaped]. (Gotham 2021, 151)

a. = There is an accomplice who ensured that every prisoner escaped.
($\exists > \forall$)

b. = For every prisoner, there is an accomplice who ensured they escaped.
($\forall > \exists$)

Vanilla LFG+Glue predicts that there should be no scope islands, and that inverse scope readings should always be available, contrary to fact. Gotham (2021) provides a solution to this problem, but he does so by introducing (new) modalities to the linear logic. We would prefer a solution that leaves the underlying mechanisms of Glue unchanged.

2.4 Sublexical meanings

Recent work in LFG+Glue (e.g. Asudeh et al. 2014, Przepiórkowski 2017, Findlay 2020) has proposed to break down lexical meaning (using *templates*) so that common parts can be shared across lexical entries. (15) shows an example lexical entry for the passive participle *crushed*, based on Asudeh et al. (2014, 79):

(15) crushed V $(\uparrow \textsc{pred}) = \text{`crush'}$
@AGENT
@PATIENT
@PASSIVE
$\lambda e.\mathbf{crush}(e) : v_\uparrow \multimap t_\uparrow$

The meaning contribution of this verb is factored out into several components: the last line provides the only idiosyncratic meaning, a predicate which describes the kind of event expressed by *crushed*. The preceding three lines each provide a meaning constructor that will also recur in other verbs. The first template provides a meaning constructor which adds an Agent argument to the event predicate expressed by the verb; the second does the same for a Patient argument; and the third provides an optional meaning constructor which existentially closes a dependency on the Agent argument, to be used when it is unexpressed (as in the short passive). These templates are spelled out below:[5]

(16) AGENT $\coloneqq$
$\lambda P.\lambda x.\lambda e.P(e) \wedge \mathbf{agent}(e,x) : (v_\uparrow \multimap t_\uparrow) \multimap e_{(\uparrow_\sigma \text{ARG1})} \multimap v_\uparrow \multimap t_\uparrow$

(17) PATIENT $\coloneqq$
$\lambda P.\lambda x.\lambda e.P(e) \wedge \mathbf{patient}(e,x) : (v_\uparrow \multimap t_\uparrow) \multimap e_{(\uparrow_\sigma \text{ARG2})} \multimap v_\uparrow \multimap t_\uparrow$

(18) PASSIVE $\coloneqq$
$(\uparrow \textsc{voice}) = \textsc{passive}$
$\Big(\lambda P.\exists x[P(x)] : (e_{(\uparrow_\sigma \text{ARG1})} \multimap t_\uparrow) \multimap t_\uparrow\Big)$

This factorisation serves an important theoretical purpose, and enables generalisations across the lexicon to be described in a way which is otherwise not possible. However, by breaking up meaning constructors in this way, we also introduce *a lot* of spurious ambiguity into the proofs. As an example, let us consider the analysis which Asudeh et al. (2014) give for *Kim was crushed last night*. They provide seven lexically contributed meaning constructors, and use them to obtain the proof shown in Figure 1. However, from these seven premises, we can in fact derive no fewer than *twenty* distinct proofs, though all of them are logically equivalent, corresponding to the reading shown as the conclusion of Figure 1. This ambiguity stems from the fact that the Glue proofs provide answers to linguistic non-questions like whether *last night* is applied to the event description before or after the Patient argument is introduced. Since the meaning is built up conjunctively, and conjunction is commutative, it makes no difference which order the conjuncts are combined in – but logically speaking, the different combinations correspond to different proofs. And this problem will escalate exponentially with longer sentences or more complexly

[5] We allow the type constructors in our first-order Glue to take s-structures as well as f-structures as arguments: here $(\uparrow_\sigma \text{ ARG1})$ and $(\uparrow_\sigma \text{ ARG2})$ refer to semantic arguments of the predicate which may or may not be realised overtly in the syntax, following whatever version of Mapping Theory is adopted (see Findlay 2016, 2020 for discussion of mapping within the theoretical framework assumed by Asudeh et al. 2014).

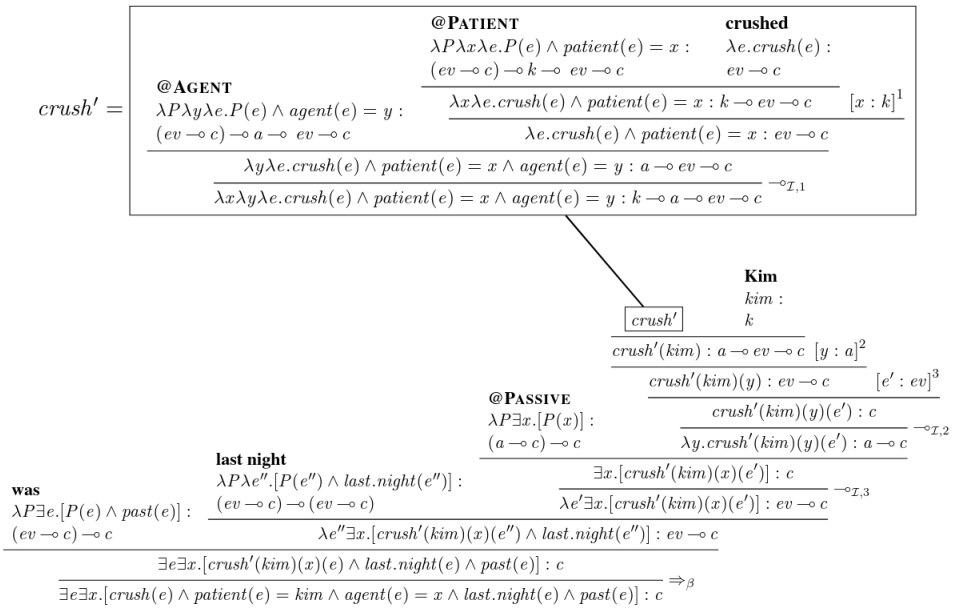


Figure 1: Glue proof for *Kim was crushed last night* (Asudeh et al. 2014, 86)

subdivided lexical entries. Such a proliferation of spurious ambiguities may seem merely inelegant from a theoretical perspective, but from a practical perspective it can be disastrous, since it will drastically increase the time it takes to parse a sentence in any computational implementation.

The solution is apparent from the diagram shown in Figure 1: the authors draw a box around the lexical meaning and the two argument-adding meaning constructors, indicating that these meaning constructors are to be combined first. This would indeed reduce the number of spurious ambiguities produced, in this case from twenty to three. But there is actually no way of doing this in standard LFG+Glue. In this paper, we provide a general means of enforcing just such a 'boxing off'.

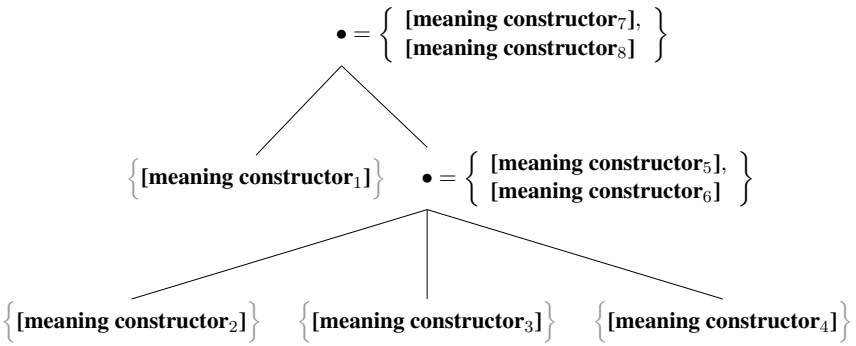
3 Proof-structure

In brief, the solution to all of the problems presented above is to control the structure of the Glue proofs, so as to control the order of composition. For all four, we require some meaning constructors to combine before others, or to the exclusion of others. In other words, some meaning constructors should be 'boxed off', as in Figure 1. As mentioned above, existing proposals have achieved this aim via additions to the linear logic (Gotham 2019, 2021), or via constraints on the proof algorithm (Crouch and van Genabith 1999). We propose a different solution, exploiting the LFG projection architecture and introducing a new level of representation which can be used to enforce constraints on the structure of Glue proofs. We call this level of representation *proof-structure*.

Proof-structure is *not* a level of representation for proofs: the proof itself remains an epiphenomenon – merely the process by which we arrive at a meaning, not a part of the linguistic representation itself. Rather, proof-structure can be seen as representing the grammatically-encoded scopal structure of a sentence: it is a level of representation at which we can express constraints on what kinds of proofs are admitted by the grammar, by grouping together meaning constructors that 'go together' to the exclusion of others, but this underspecifies the form of the actual proof(s) involved. Formally speaking, proof-structure is a tree, where each sub-tree corresponds to a sub-proof. The nodes in the tree are (sets of) premises (i.e. meaning constructors), and each meaning constructor in a mother node is obtained from a proof which makes use of exactly one meaning constructor from each of its daughters. Notice again that the structure underspecifies the actual proof(s), since we allow n-ary branching, whereas proof rules are generally at most binary.

A schematic proof-structure is shown in (19):

(19)



We adopt the convention of suppressing set brackets when the sets are singletons, so meaning constructors contributed from the initial parse (be it lexically or from an annotated phrase-structure rule) will appear as terminal nodes in proof-structures, without surrounding set brackets (shown greyed out in (19)). Higher nodes in the tree consist of sets of meaning constructors obtainable by combining their daughters. These mother nodes can be multi-member sets because there can be multiple ways of combining their daughter meaning constructors, and we want to record those ambiguities and propagate them up the tree. However, in many cases in this paper we simply represent these nodes as bullets (•), since what is important is the structure they impose, rather than their content.

Crucially, we require that the proofs by which mother nodes are produced are *complete*, in the sense that there are no undischarged hypotheses. This has the effect that each sub-tree in proof-structure is a scope island: quantifier scoping in Glue involves hypothetical reasoning, and by limiting the span over which such reasoning is allowed to occur, we also limit the potential domains of scopal interaction.

Proof-structure is connected to the LFG architecture by a function γ,[6] projected from c-structure. By projecting proof-structure from c-structure, we maintain access

[6]The label γ is essentially arbitrary, but we intend it to be vaguely mnemonic for 'Glue' and/or 'Girard', for Jean-Yves Girard, the inventor of linear logic (Girard 1987).

to configurational information that is lost at f-structure but which may be relevant to determining scope (in the case of modifier scope, for example), i.e. the "relationship between c-structure and the semantics that is not mediated by f-structure" mentioned by Andrews and Manning (1999, 9).

Lexically contributed meaning constructors are introduced as daughters of their c-structure pre-terminal's proof structure. This is written in lexical entries as follows:

(20) $\hat{*}_\gamma \lhd$ **[meaning constructor]**

In words, this says that this node's mother (i.e. the pre-terminal node dominating the lexical item) has a proof-structure which immediately dominates ($\lhd$) whatever meaning constructor is being introduced.[7]

Phrase-structure rules will now bear annotations describing their contribution to proof-structure as well as the other levels of representation in the parallel projection architecture. In the default case, all phrase-structure rules will simply bear the annotation $\hat{*}_\gamma = *_\gamma$, equating mother and daughter's proof-structures. For example, the IP rule in English might look like this:

(21)
$$\begin{array}{lcccc} \text{IP} & \rightarrow & \text{NP} & & \text{I}' \\ & & (\uparrow \textsc{subj}) = \downarrow & & \uparrow = \downarrow \\ & & \hat{*}_\gamma = *_\gamma & & \hat{*}_\gamma = *_\gamma \end{array}$$

If we add this annotation to every right-hand element in every phrase-structure rule, we will obtain a totally flat proof-structure for every sentence. This is our baseline, and corresponds to the 'vanilla' LFG+Glue position: there are no scope islands and so no constraints on the order of composition. But we now also have the ability to describe a more articulated proof-structure, which means we can selectively interrupt this flat structure to enforce the constraints we need in order to solve the problems discussed in Section 2.

4 Solutions to the problems

4.1 Modifier scope

To enforce the required scope ordering for English pre-nominal adjectives, we annotate the relevant phrase-structure rule as follows:

(22)
$$\begin{array}{lcccc} \text{N} & \rightarrow & \widehat{\text{Adj}} & & \text{N} \\ & & \downarrow \in (\uparrow \textsc{adj}) & & \uparrow = \downarrow \\ & & \hat{*}_\gamma = *_\gamma & & \hat{*}_\gamma \lhd *_\gamma \end{array}$$

That is, the proof-structure for the N head is made a *daughter* of the mother N's proof-structure, rather than being identified with it. This has the effect that things

[7] Once again, this should really be the set containing the meaning constructor in question, but we suppress set brackets around singleton sets.

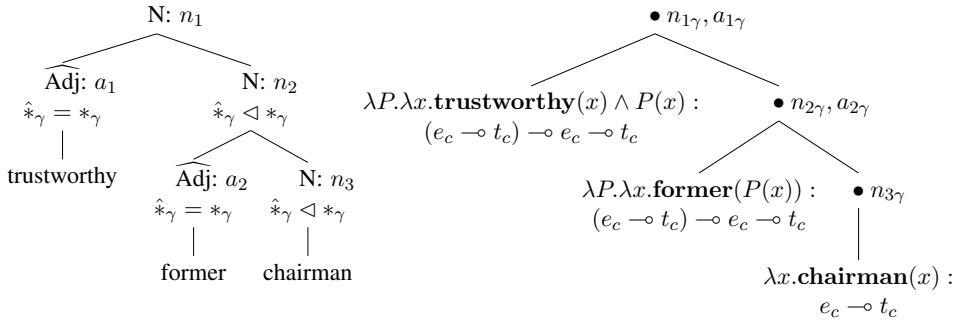


Figure 2: C-structure and proof-structure for *trustworthy former chairman*

below the nominal head must be composed before things above, so that adjectives closer to the head will have their meanings applied before adjectives further away. In other words, it enforces a scope order based on c-structure. Let us see how this works.

Assuming the following lexical entries, we obtain the c-structure and proof-structure shown in Figure 2 for the phrase *trustworthy former chairman*.[8]

(23) trustworthy $\widehat{\text{Adj}}$ $(\uparrow \text{PRED}) = \text{`trustworthy'}$
$\%N = (\text{ADJ} \in \uparrow)$
$\hat{*}_\gamma \lhd \lambda P \lambda x.\mathbf{trustworthy}(x) \wedge P(x) :$
$(e_{\%N} \multimap t_{\%N}) \multimap e_{\%N} \multimap t_{\%N}$

(24) former $\widehat{\text{Adj}}$ $(\uparrow \text{PRED}) = \text{`former'}$
$\%N = (\text{ADJ} \in \uparrow)$
$\hat{*}_\gamma \lhd \lambda P \lambda x.\mathbf{former}(P(x)) :$
$(e_{\%N} \multimap t_{\%N}) \multimap e_{\%N} \multimap t_{\%N}$

(25) chairman N $(\uparrow \text{PRED}) = \text{`chairman'}$
$\hat{*}_\gamma \lhd \lambda x.\mathbf{chairman}(x) : e_\uparrow \multimap t_\uparrow$

As we can see, the proof-structure node corresponding to the middle N node, i.e. $n_{2\gamma}$, is dominated by the proof-structure node corresponding to the higher N node, $n_{1\gamma}$. The effect of this is that the meanings for *former* and *chairman* must combine with each other before they combine with *trustworthy*, since their meaning constructors are dominated by $n_{2\gamma}$ while *trustworthy*'s is not. Thus, we obtain the correct, c-structure-mediated scoping, where the meaning of the phrase is (26a), not (26b).

(26) a. ✓$\mathbf{trustworthy}(\mathbf{former}(\mathbf{chairman}))$
b. ✗$\mathbf{former}(\mathbf{trustworthy}(\mathbf{chairman}))$.

Notice that by adding the annotation which creates an articulated proof-structure to the phrase-structure rule in (22), we apply this analysis to all cases of pre-nominal

[8]Here and throughout we use mnemonic labels for the appropriate f-structures in the linear logic types of meaning constructors.

adjective modification, including those involving intersective adjectives where order of combination is immaterial. That is, the proof-structure for *trustworthy Scottish chairman* looks identical to that for *trustworthy former chairman*, and similarly requires that the meanings of *Scottish* and *chairman* combine with each other before combining with *trustworthy*. Although there is no theoretical reason to enforce such an ordering, there is no reason to avoid it either, and in fact a good practical reason to prefer it: as mentioned above, generating spurious ambiguities adds to the difficulty of the computational task involved in parsing. So the analysis presented here produces both theoretical and practical gains – it prevents us from deriving some unattested readings, and it also eliminates some spurious ambiguity.

4.2 Scope freezing

As scope freezing illustrates, sometimes we do not want proof-structure to simply recapitulate c-structure, but rather add structure which is not present elsewhere. But this is straightforward to do as well; for the English double-object construction, for example, we add the following annotations to the relevant phrase-structure rule:

(27)
$$\begin{array}{lcccc} \mathrm{V}' & \rightarrow & \mathrm{V} & \mathrm{NP} & \mathrm{NP} \\ & & \uparrow{=}\downarrow & (\uparrow \textsc{obj}) = \downarrow & (\uparrow \textsc{obj}_\theta) = \downarrow \\ & & \hat{*}_\gamma \lhd *_\gamma & \hat{*}_\gamma = *_\gamma & \hat{*}_\gamma \lhd *_\gamma \end{array}$$

This adds a new level of proof-structure below the V′ proof-structure, which contains the meaning constructors contributed by the verb and by the second object.[9] The first object, though, has the same proof structure as the V′, and so its meaning constructors are in the higher part of the proof-structure. This means that the first object will always outscope the second, as desired. To illustrate, the c-structure and proof-structure for *Hilary gave a student every grade* are shown in Figure 3.

4.3 Scope islands

Since quantifiers cannot scope outside of the proof-structure node they appear under, we have a straightforward way of enforcing scope islands: we simply add articulation to the proof-structure at the desired location. For example, we could make all CP COMPs scope islands by adding the following annotations to the relevant phrase-structure rule:

(28)
$$\begin{array}{lcccc} \mathrm{V}' & \rightarrow & \mathrm{V} & \mathrm{CP} \\ & & \uparrow = \downarrow & (\uparrow \textsc{comp}) = \downarrow \\ & & \hat{*}_\gamma = *_\gamma & \hat{*}_\gamma \lhd *_\gamma \end{array}$$

[9] We assume there is a similar 'minimal solution' constraint on proof-structure as is standard elsewhere in LFG, so that, given the constraints in (27), a proof-structure with one daughter that merges the proof-structures of the V and second NP is preferred to one with two distinct daughters that keeps them apart. If we actually do need two distinct proof-structures here, we would need to explicitly differentiate them – e.g. by making use of a local name (Crouch et al. 2017) for one and stating that the other is not equal to it.

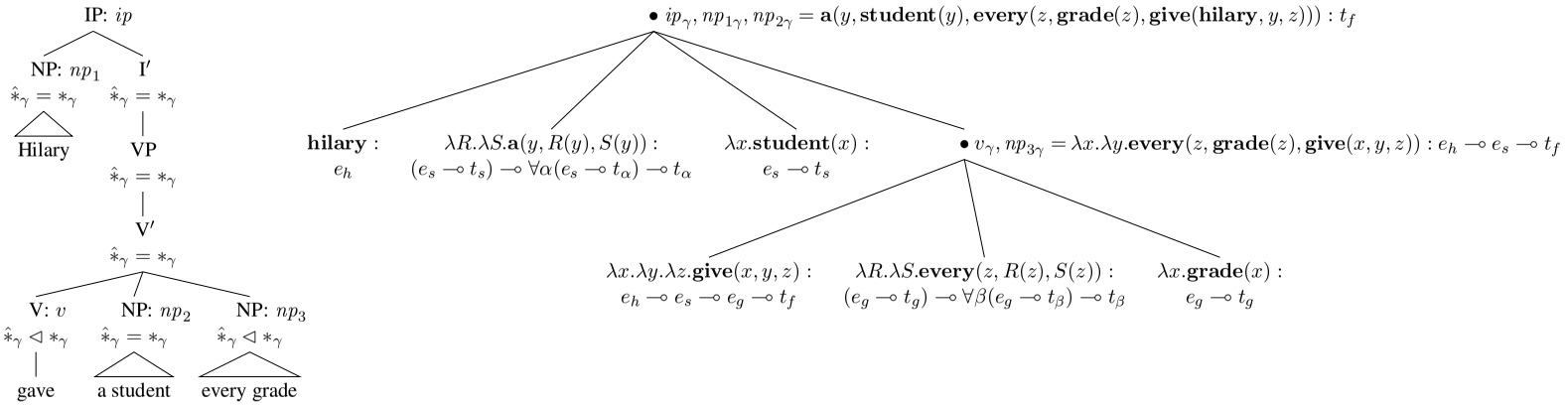


Figure 3: C-structure and proof-structure for *Hilary gave a student every grade*

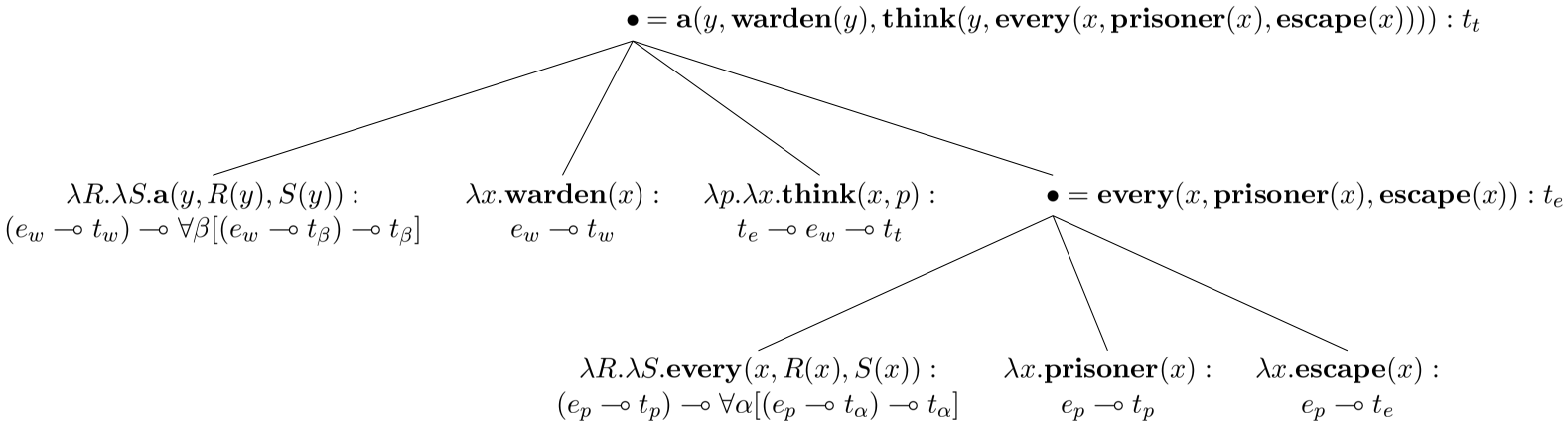


Figure 4: Proof-structure for *A warden thinks every prisoner escaped*

The proof-structure for *A warden thinks every prisoner escaped* would then be as shown in Figure 4. As can be seen, the two quantifiers appear under different nodes, and therefore cannot interact. This means that each of the mother nodes is only a singleton set, containing the one reading shown.

Of course, complement clauses are not always islands; as discussed above, this depends on so far poorly understood interactions between embedding verbs and embedded quantifiers. Formally, such dependency is no problem, however: we can make the added articulation in proof structure dependent on the presence/absence of other features. For example, let us assume that there is an s-structure feature SCOPEISLAND, whose value is $+$ when the combination of embedding verb and quantifier produces such an island, and $-$ when it does not. Then we can have a more complex CP-introducing rule, which only creates an island when the value of the CP's SCOPEISLAND feature is $+$:

(29) $\mathrm{V}' \rightarrow$ V: $\uparrow = \downarrow$, $\hat{*}_\gamma = *_\gamma$; $\left\{ \hat{*}_\gamma = *_\gamma \;\middle|\; \begin{array}{c} \text{CP: } (\uparrow \text{COMP}) = \downarrow \\ \hat{*}_\gamma \lhd *_\gamma \\ (\downarrow_\sigma \text{SCOPEISLAND}) =_c + \end{array} \right\}$

In actual fact, we will probably need something more fine-grained, if, for example, some quantifiers can escape such an island at the same time as others in the same clause cannot. We reserve judgement until the facts can be better established; our goal here is merely to illustrate that we now have the formal tools required to describe such an island.

4.4 Sublexical meanings

For the sublexical meanings problem, we need to 'box off' the lexical item itself before it enters the larger semantic composition. We achieve this by making reference to the proof-structure of the c-structure terminal node containing the word. That is, in lexical entries, sublexical meanings are made daughters of $*_\gamma$ rather than of $\hat{*}_\gamma$. The new version of the lexical entry for *crushed* is shown below. Note that we must also connect the proof-structure of the terminal node to the larger proof-structure, which is achieved by the second line of (30) – this makes the lexical proof-structure a daughter of the pre-terminal node's proof-structure, thus ensuring that all sublexical material is assembled first.[10]

[10]Note that the passive meaning constructor is made a daughter of $\hat{*}_\gamma$ like other normal meaning constructors, rather than a daughter of $*_\gamma$ like other sublexical ones. This is done for alignment with Asudeh et al.'s (2014) diagram in Figure 1, where they box off the meaning constructors contributed by the valency templates and the lexical meaning, but leave the contribution of the passive outside of the box. One would need to consider the empirical facts to decide whether it was ever necessary for the quantification over the implicit Agent introduced by the passive meaning to scope outside of the lexical meaning, as this permits, or whether it could be added to the set of sublexical meanings like the others.

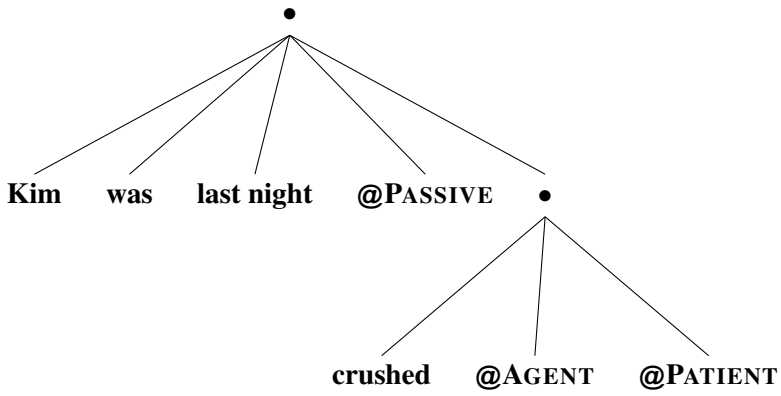


Figure 5: Proof-structure for *Kim was crushed last night*

(30) crushed V $(\uparrow \text{PRED}) = \text{`crush'}$
$\hat{*}_\gamma \lhd *_\gamma$
@AGENT
@PATIENT
@PASSIVE
$*_\gamma \lhd \lambda e.\mathbf{crush}(e) : v_\uparrow \multimap t_\uparrow$

(31) AGENT $\coloneqq$
$*_\gamma \lhd \lambda P.\lambda x.\lambda e.P(e) \wedge \mathbf{agent}(e,x) :$
$(v_\uparrow \multimap t_\uparrow) \multimap e_{(\uparrow_\sigma \text{ARG1})} \multimap v_\uparrow \multimap t_\uparrow$

(32) PATIENT $\coloneqq$
$*_\gamma \lhd \lambda P.\lambda x.\lambda e.P(e) \wedge \mathbf{patient}(e,x) :$
$(v_\uparrow \multimap t_\uparrow) \multimap e_{(\uparrow_\sigma \text{ARG2})} \multimap v_\uparrow \multimap t_\uparrow$

(33) PASSIVE $\coloneqq$
$(\uparrow \text{VOICE}) = \text{PASSIVE}$
$\left(\hat{*}_\gamma \lhd \lambda P.\exists x[P(x)] : (e_{(\uparrow_\sigma \text{ARG1})} \multimap t_\uparrow) \multimap t_\uparrow\right)$

Figure 5 shows a schematic version of the proof-structure for the sentence *Kim was crushed last night* which this lexical entry gives rise to – for readability, we omit the details of the meaning constructors, but they are the same as those in Figure 1. As we can see, the valency templates and the lexical meaning form a sub-proof, and so their meanings are combined first. This reduces the number of proofs for the sentence as a whole from twenty to three – and if the passive meaning constructor were also included in the set of sublexical meanings, the number of proofs would be reduced to just one.[11]

[11] Obviously this approach is limited to decomposed meanings below the level of the word – it will not carry over to e.g. complex predicates (although our proposal should help with the overgeneration problem mentioned by Lowe 2015, 439 in his LFG+Glue analysis of complex predicates).

* * *

In summary, this section has shown how adding a notion of articulation to Glue proofs, whereby certain sets of meaning constructors are combined before others, enables us to find straightforward solutions to several theoretical and practical problems. We feel that using a new projection in the LFG architecture for this purpose is quite natural, and fits LFG's modular philosophy in that we do not need to change the workings of the Glue Semantics component itself.

One concern might be that the way we have presented our solution so far makes it seem quite procedural: certain proofs are finished 'before' others. However, this can be understood entirely metaphorically, and proofs can be processed in parallel. In the next section, we describe our approach to implementing the current proposal computationally, and show how this enables us to maintain a declarative rather than procedural view of parsing.

5 Implementation

The intuition behind our work – and as far as we know all previous work on restricting scope ambiguities in Glue semantics – is that we need to restrict the order in which premises can combine. As mentioned above, previous work has achieved this by either changing the logic (by adding modalities) or changing the proof algorithm. By contrast, we keep the logic and the proof system as is, and instead control the way proofs are built by using the projection architecture to gather premises into groups, where each group is then processed just like a standard Glue proof. That is, there is nothing special in the way the meaning of *every prisoner escaped* is composed from the words *every*, *prisoner* and *escaped* in the bottom right sub-tree of Figure 4, nor in the way that, once composed, that meaning then combines with the meanings of *a*, *warden* and *thinks* to yield the overall meaning of the sentence.

As a result of this, existing tools like the Glue Semantics Workbench (GSW: Meßmer and Zymla 2018) can be used off the shelf. While we have not attempted to implement our theory in XLE+Glue (Dalrymple et al. 2020), we have implemented it in a setting where meaning constructors arise from the interpretation of dependency parses rather than LFG structures (cf. Gotham and Haug 2018). In this setting, all that is needed is a script that controls which groups of premises are sent to the prover together: this is done by chopping the dependency tree into sub-trees according to which meaning constructors belong together in the same sub-proof, following rules that we provide. In the XLE+Glue setting it would be necessary to recursively build proof-structure trees according to the dominance constraints given by the annotated c-structure rules and lexical entries, then send each set of sibling nodes to the prover separately. Two issues arise: First, how can we prove non-atomic proof goals? Second, if we are to send sets of sibling nodes to the prover, does that impose an order on the proofs? If possible we want to avoid having to compute the value of a

non-leaf node based on its daughters before it is sent to the prover with its siblings. As we will see, both issues require us to know in advance what the proof goal is.

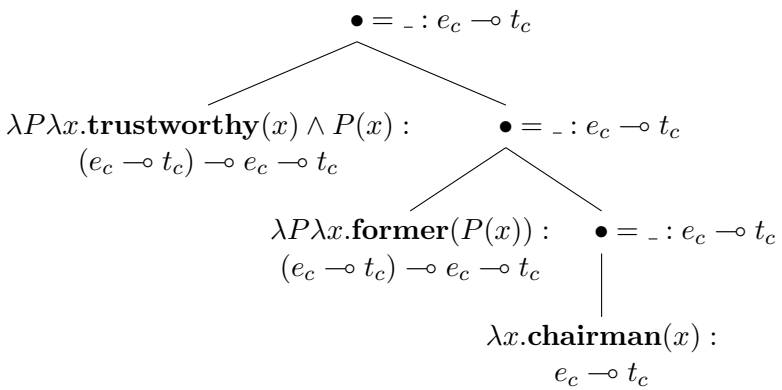
The first issue arises because the Hepple-Lev algorithm (Hepple 1996, Lev 2007) that underlies the Glue Semantics Workbench is only complete with respect to proofs of atomic formulae (i.e. if the formula to be proved contains '$\multimap$' we are not guaranteed to find a proof even if there is one). However, in cases like *trustworthy former chairman*, shown in Figure 2, the sub-proofs we require should end in $e_c \multimap t_c$. And although in this particular (simple) case GSW does in fact find such a proof from the two premises $e_c \multimap t_c$ and $(e_c \multimap t_c) \multimap e_c \multimap t_c$, this is not the case in general for non-atomic goals.

If we know in advance what the proof goal should be, we can work around this problem by supplying an artificial meaning constructor that consumes the intended goal and produces an atomic resource; in the case at hand it would be $GOAL : (e_c \multimap t_c) \multimap G$. GSW then straightforwardly returns the proof that we represent in natural deduction format in (34) for the sub-proof of *former chairman*:

(34)
$$\dfrac{\dfrac{\begin{array}{ll}\lambda P\lambda x.\mathbf{former}(P(x)): & \lambda x.\mathbf{chairman}(x): \\ (e_c \multimap t_c) \multimap e_c \multimap t_c & e_c \multimap t_c\end{array}}{\begin{array}{ll}\lambda x.\mathbf{former}(\mathbf{chairman}(x)): & GOAL: \\ e_c \multimap t_c & (e_c \multimap t_c) \multimap G\end{array}}}{GOAL(\lambda x.\mathbf{former}(\mathbf{chairman}(x))) : G}$$

From (34) it is easy to extract the semantic resource we want by removing the $GOAL$ predicate and using the goal type we need, to yield the meaning constructor $\lambda x.\mathbf{former}(\mathbf{chairman}(x)) : (e_c \multimap e_t)$.

Moreover, if we know what the proof goal is, we can produce proof-structures decorated with explicit goal types for the non-terminal nodes. The proof-structure from Figure 2 could then be represented as (35) without any proofs being carried out:

(35)


This suggests a solution to the second issue. As we noted above, the tree format in (35) may suggest that we first prove the meaning of *former chairman* and then

plug the result into a new proof which also involves the meaning constructor of *trustworthy*. But if we already know that the meaning of *trustworthy* combines with something of the linear logic type $e_c \multimap t_c$, , we do not have to proceed in this way, but can perform the required proof with a placeholder meaning constructor of the form shown in (36):

(36) $PLACEHOLDER : e_c \multimap t_c$.

This proof can be performed in parallel with the proof in (34) and yields (37):

(37) $\lambda x.\mathbf{trustworthy}(x) \wedge PLACEHOLDER(x) : e_c \multimap t_c$

Given that the type of $PLACEHOLDER$ was set to the goal type of the proof in (34), we can – once we are done with all proofs – replace it with the meaning side of the goal from (34) (removing the $GOAL$ wrapper). This yields (38), the correct meaning constructor for the whole phrase:

(38) $\lambda x.\mathbf{trustworthy}(x) \wedge \mathbf{former}(\mathbf{chairman}(x)) : e_c \multimap t_c$

Aside from the performance gains from parallelisation, this procedure also serves to reduce the number of proofs that we must compute in cases where one or more sub-trees are ambiguous, i.e. have several proofs yielding several meaning sides (which evidently all have the same linear logic type). Consider a situation where we have one sub-proof inside another, and both sub-trees contain three quantifiers. Such structures are not unexpected in a Glue setting. They could in fact easily occur in a more detailed representation of a sentence like *A guard thinks every prisoner escaped* if an event and time variable were also included.[12] Computing proofs corresponding to all possible permutations of the six quantifiers would yield $6! = 720$ proofs. By boxing off each clause, we reduce the space of readings to $3! \times 3! = 36$ readings. But to get those 36 readings, we only need to perform 6 proofs for each clause $= 12$ proofs.

We have seen that the key both to proving non-atomic goals and to achieving parallelisation is to know what the goal type of a given proof-structure node should be. For the system to know this without actually inspecting the premises, we need to state proof goals along with the dominance constraints for the proof structure. In many cases, this is easily done. For example, we can augment the pre-nominal adjective rule in (22) as follows to provide the goal types used in (35):

(39)

N $\rightarrow$	$\widehat{\text{Adj}}$	N
	$\downarrow \in (\uparrow \text{ADJ})$	$\uparrow = \downarrow$
	$\hat{*}_\gamma = *_\gamma$	$\hat{*}_\gamma \lhd *_\gamma$
		$\boxed{\pi_2(*_\gamma) = e_\uparrow \multimap t_\uparrow}$

[12] It is part of the design of Glue that we do not look at the meaning side when computing scopes and hence it is easily possible that we end up computing six permutations of the three quantifiers in a sentence with three existentially quantified variables for individuals, times and events, even if they are all semantically equivalent.

Here we treat a meaning constructor as a pair consisting of a lambda expression and a linear logic type, and use the projection function π_2 to refer to the second member of that pair, i.e. the type, so that we can then constrain it appropriately.[13]

In our own implementation this approach generalises, because we use a neo-Davidsonian semantics in the style of Champollion (2015), where all arguments are treated semantically as modifiers of verbal meanings. This means that they leave the same type as they consume, making it straightforward to identify goal types and state them in the rules. However, as a reviewer points out, it is not trivial to compute what the type of v_γ in Figure 3 is, where we have not used a neo-Davidsonian semantics. v_γ must therefore have the type of a consumer (of the remaining verbal arguments) rather than a consumee. A discussion of the different possible solutions here would take us too far afield, and so we leave for future work to determine how this can be achieved.

6 Conclusion

We have presented four problems for standard LFG+Glue relating to scope: modifier scope, scope freezing, scope islands, and sublexical meanings. The first three involve empirical failings – LFG+Glue predicts readings which are unattested. The last is a largely practical problem – the atomisation of lexical meanings results in a large number of spurious ambiguities in the Glue proofs. Both kinds of problems are ultimately of the same nature: namely, Glue is too permissive insofar as it places no constraints on which scope-taking meanings can interact. The solution is therefore to provide a means of constraining the order in which meaning constructors can combine, so that some interactions are ruled out. Unlike previous proposals, our solution takes advantage of LFG's modular approach to the grammar, and defines a new level of the projection architecture, proof-structure, which can be used to 'box off' certain parts of the Glue proof, turning them into their own self-contained sub-proofs. This means that the Glue Semantics component itself remains unchanged, and existing tools and analyses can continue to be used. We demonstrated that proof-structure offers straightforward solutions to the four problems discussed here, and lastly showed that it can be implemented in such a way that we retain the declarative status of LFG+Glue. We believe that this offers the least problematic solution yet to a number of long-standing issues in Glue Semantics.

References

Andrews, Avery D. 2018. Sets, heads and spreading in LFG. *Journal of Language Modelling* 6(1), 131–174.

Andrews, Avery D. and Manning, Christopher D. 1999. *Complex predicates and information spreading in LFG*. Stanford, CA: CSLI Publications.

[13] We commit a formal fudge here, since $*_\gamma$ is actually a *set* of meaning constructors; we assume the definition of π_2 is modified in such a way as to make it distributive.

Asudeh, Ash. 2005. Relational nouns, pronouns, and resumption. *Linguistics and Philosophy* 28(4), 375–446.

Asudeh, Ash. 2022. Glue Semantics. *Annual Review of Linguistics* 8, 321–341.

Asudeh, Ash and Giorgolo, Gianluca. 2012. Flexible composition for optional and derived arguments. In Miriam Butt and Tracy Holloway King (eds.), *Proceedings of the LFG'12 Conference*, pages 64–84, Stanford, CA: CSLI Publications.

Asudeh, Ash, Giorgolo, Gianluca and Toivonen, Ida. 2014. Meaning and valency. In Miriam Butt and Tracy Holloway King (eds.), *Proceedings of the LFG'14 Conference*, pages 68–88, Stanford, CA: CSLI Publications.

Barker, Chris. 2022. Rethinking scope islands. *Linguistic Inquiry* 53(4), 633–661.

Bary, Corien and Haug, Dag. 2011. Temporal anaphora across and inside sentences: the function of participles. *Semantics & Pragmatics* 4(8).

Belyaev, Oleg and Haug, Dag. 2014. Pronominal coreference in Ossetic correlative and the syntax-semantics interface. In Miriam Butt and Tracy Holloway King (eds.), *Proceedings of the LFG'14 Conference*, pages 89–109, Stanford, CA: CSLI Publications.

Campbell, Richard. 2002. Computation of modifier scope in NP by a language-neutral method. In *COLING 2002: the 19th International Conference on Computational Linguistics*.

Champollion, Lucas. 2015. The interaction of compositional semantics and event semantics. *Linguistics and Philosophy* 38(1), 31–66.

Crouch, Dick, Dalrymple, Mary, Kaplan, Ronald M., King, Tracy Holloway, Maxwell III, John T. and Newman, Paula. 2017. *XLE documentation*. Palo Alto Research Center (PARC), Palo Alto, CA.

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

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

Dalrymple, Mary, Lamping, John and Saraswat, Fernando Pereira Vijay. 1999. Quantification, anaphora, and intensionality. In Mary Dalrymple (ed.), *Semantics and syntax in Lexical Functional Grammar: the resource logic approach*, pages 39–90, Cambridge, MA: MIT Press.

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

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 Zymla, 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*, pages 89–108, Stanford,

CA: CSLI Publications.
Findlay, Jamie Y. 2016. Mapping theory without argument structure. *Journal of Language Modelling* 4(2), 293–338.
Findlay, Jamie Y. 2019. *Multiword expressions and the lexicon*. Ph.D. thesis, University of Oxford.
Findlay, Jamie Y. 2020. Mapping Theory and the anatomy of a lexical entry. In Miriam Butt and Ida Toivonen (eds.), *Proceedings of the LFG'20 Conference*, pages 127–147, Stanford, CA: CSLI Publications.
Girard, Jean-Yves. 1987. Linear logic. *Theoretical Computer Science* 50(1), 1–102.
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*, pages 111–129, Stanford, CA: CSLI Publications.
Gotham, Matthew. 2021. Approaches to scope islands in LFG+Glue. In Miriam Butt, Jamie Y. Findlay and Ida Toivonen (eds.), *Proceedings of the LFG'21 Conference*, pages 146–166, Stanford, CA: CSLI Publications.
Gotham, Matthew and Haug, Dag T. T. 2018. Glue semantics for Universal Dependencies. In Miriam Butt and Tracy Holloway King (eds.), *Proceedings of the LFG'18 Conference*, pages 208–226, Stanford, CA: CSLI Publications.
Haug, Dag T. T. 2008. Tense and aspect for Glue Semantics: the case of participial XADJs. In Miriam Butt and Tracy Holloway King (eds.), *Proceedings of the LFG'08 Conference*, pages 291–311, Stanford, CA: CSLI Publications.
Hepple, Mark. 1996. A compilation-chart method for linear categorial deduction. In *COLING '96: Proceedings of the 16th conference on computational linguistics*, volume 1, pages 537–542.
Kokkonidis, Miltiadis. 2008. First-order Glue. *Journal of Logic, Language and Information* 17(1), 43–68.
Larson, Richard K. 1990. Double objects revisited: reply to Jackendoff. *Linguistic Inquiry* 21(4), 589–632.
Lev, Iddo. 2007. *Packed computation of exact meaning representations*. Ph.D. thesis, Stanford University, Stanford, CA.
Lowe, John J. 2015. Complex predicates: an LFG+glue analysis. *Journal of Language Modelling* 3(2), 413–462.
Meßmer, Moritz and Zymla, Mark-Matthias. 2018. The Glue Semantics Workbench: a modular toolkit for exploring linear logic and Glue Semantics. In Miriam Butt and Tracy Holloway King (eds.), *Proceedings of the LFG'18 Conference*, pages 249–263, Stanford, CA: CSLI Publications.
Przepiórkowski, Adam. 2017. A full-fledged hierarchical lexicon in LFG: the FrameNet approach. In Victoria Rosén and Koenraad De Smedt (eds.), *The very model of a modern linguist: in honor of Helge Dyvik*, Bergen Language and Linguistics Studies, No. 8, pages 202–219, Bergen: University of Bergen.