

Making Logical Form type-logical

Glue Semantics for Minimalist syntax

Matthew Gotham

University of Oslo

UiO Forum for Theoretical Linguistics
12 October 2016



Slides available at <http://folk.uio.no/matthegg/research#talks>

What this talk is about

Slides available at <http://folk.uio.no/mattheegg/research#talks>

- An implementation of **Glue Semantics**

- an approach that treats the syntax-semantics interface as deduction in a type logic—

- for Minimalist syntax,

- i.e. syntactic theories in the $ST \rightarrow EST \rightarrow REST \rightarrow GB \rightarrow \dots$ ‘Chomskyan’ tradition.

Q How Minimalist, as opposed to (say) GB-ish?

A Not particularly, *but* the factoring together of subcategorization and structure building (in the mechanism of feature-checking) is, if not crucial to this analysis, then certainly useful.

and

- a comparison of this approach with more mainstream approaches to the syntax-semantics interface.

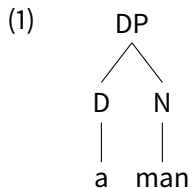
Outline

- 1 The mainstream approach
- 2 A fast introduction to Glue Semantics
- 3 Implementation in Minimalism
 - The form of syntactic theory assumed
 - The connection to Glue
- 4 Comparison with the mainstream approach
 - Interpreting (overt) movement
 - Problems with the mainstream approach
 - Glue analysis
 - Nested DPs
 - Scope islands

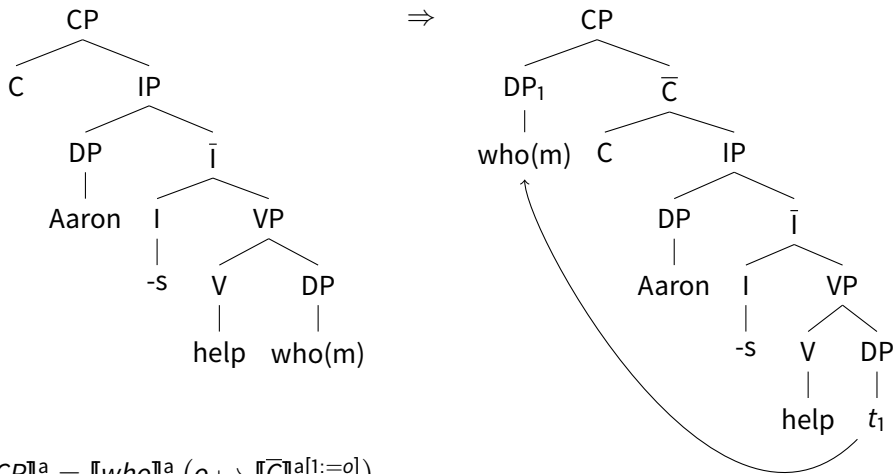
How semantics tends to be done for broadly GB/P&P/Minimalist syntax

After Heim & Kratzer (1998)

- Syntax produces structures that are interpreted recursively according to compositional rules, primarily the rule of function application.
- For example, in (1), $\llbracket DP \rrbracket = \llbracket D \rrbracket (\llbracket N \rrbracket) = \llbracket a \rrbracket (\llbracket man \rrbracket)$



Syntax is taken to involve *transformational* rules, for example *movement*:



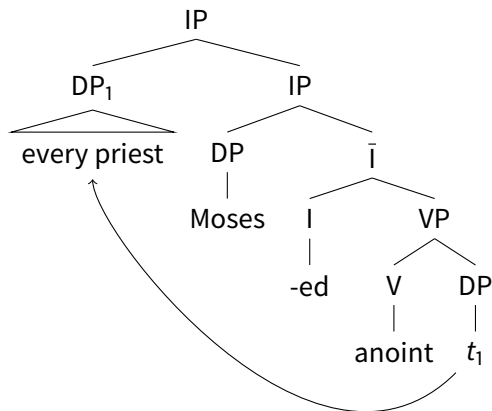
$$\llbracket CP \rrbracket^a = \llbracket who \rrbracket^a (o \mapsto \llbracket \bar{C} \rrbracket^{a[1:=o]})$$

The interpretative rules treat the trace as a variable, and the moved constituent coindexed with it in such a way that it binds that variable.

Covert movement

It is widely assumed that movement can be *covert*, i.e. that the structure that is the input to semantics can be one derived from the pronounced structure by further movement processes, e.g. *Quantifier Raising* (QR):

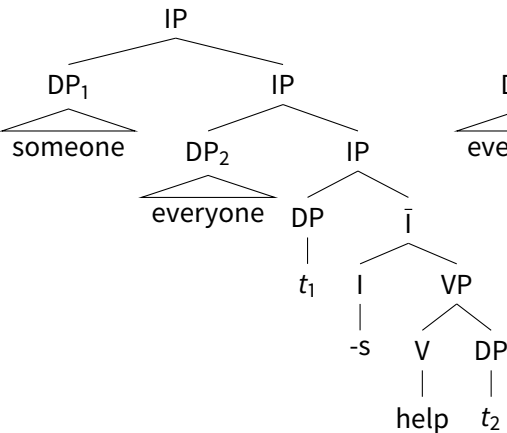
Logical Form



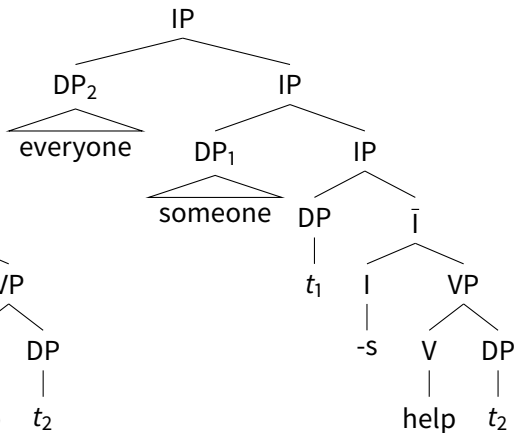
Quantifier scope ambiguity is therefore syntactic ambiguity at a level of representation after covert movement, called *Logical Form* (LF).

(2) Someone helps everyone.

LF1: Surface scope



LF2: Inverse scope



Features of the Glue analysis

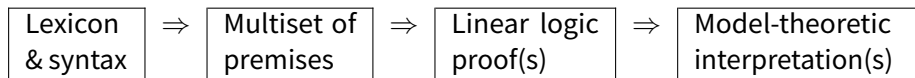
to be presented

- Function application is still centre stage.
- The variable-binding mechanism needed to interpret movement comes for free; there is no need for traces in syntax.
- It fits just as nicely with copy- or remerge-based theories of movement.
- There is no need for covert movement or LF in order to account for scope ambiguity.

A fast introduction to Glue Semantics

Glue Semantics is a theory of the syntax-semantics interface according to which

- syntactic analysis produces a multiset of premises in a fragment of linear logic (Girard 1987), and
- semantic interpretation consists in constructing a proof using those premises.



The syntax-semantics interface according to Glue

- Glue is the mainstream view of the syntax-semantics interface in LFG (Dalrymple et al. 1993, Dalrymple, Gupta, et al. 1999), for which it was originally developed.
- It has also been applied to HPSG (Asudeh & Crouch 2002) and LTAG (Frank & van Genabith 2001).

Linear logic

Linear logic is often called a ‘logic of resources’(Crouch & van Genabith 2000: 5). The reason for this is that, in linear logic, for a sequent

premise(s) \vdash conclusion

to be valid, every premise in premise(s) must be ‘used’ exactly once. So for example,

$A \vdash A$ and $A, A \multimap B \vdash B$, but
 $A, A \not\vdash A$ and $A, A \multimap (A \multimap B) \not\vdash B$

(\multimap is linear implication)

Interpretation as deduction

In Glue,

- expressions of a meaning language (in this case, the lambda calculus) are paired with formulae in a fragment of linear logic (the glue language), and
- steps of deduction carried out using those formulae correspond to operations performed on the meaning terms, according to the Curry-Howard correspondence (Howard 1980).

Linear implication and functional types

Rules for \multimap and their images under the Curry-Howard correspondence	
Elimination...	Introduction...
$\frac{f : X \multimap Y \quad a : X}{f(a) : Y} \multimap E$	<div style="display: flex; align-items: center;"> <div style="flex: 1;"> $\frac{\begin{array}{c} [v : X]^n \\ \vdots \\ f : Y \end{array}}{\lambda v(f) : X \multimap Y} \multimap I, n$ </div> <div style="flex: 1; padding-left: 20px;"> <p>Exactly one hypothesis must be discharged in the introduction step.</p> </div> </div>
...corresponds to ...	
...application.	...abstraction.

In this paper, $m : \Phi$...

- ... is the pairing of meaning m with linear logic formula Φ
- ... will sometimes be referred to as a ‘meaning constructor’

Two logics

Meaning constructors

$$m : \Phi$$

lambda calculus

■ connectives:

 λ
 $=$
 $\neg, \wedge, \vee, \leftrightarrow$
 \rightarrow
 \exists
 \forall

a fragment of linear logic

■ connectives:

 \multimap
 \boxplus^1

¹This choice of notation is somewhat idiosyncratic, but see Morrill 1994: Chapter 6.

Two logics

Meaning constructors

$$m : \Phi$$

lambda calculus

- higher order
- constants and variables in every type

a fragment of linear logic

- first order (and monadic)
- predicates:
e and t
- constants:
1, 2, 3, ...
- variables:
X, Y, Z, X₁, X₂, ...

To save space, I'll write e.g. e_1 and t_Y instead of $e(1)$ and $t(Y)$ respectively.

Type map

For any meaning constructor $m : \Phi$, m is of type $\text{Ty}(\Phi)$, where

- (3) a. For any term α :
- (i) $\text{Ty}(t_\alpha) = t$
- (ii) $\text{Ty}(e_\alpha) = e$
- b. For any formulae A and B , and any variable X :
- (i) $\text{Ty}(A \multimap B) = \text{Ty}(A) \rightarrow \text{Ty}(B)$
- (ii) $\text{Ty}(\forall X(A)) = \text{Ty}(A)$

So for example,

if $x : e_7$	then x is of type e
if $f : e_4 \multimap t_5$	then f is of type $e \rightarrow t$
if $c : \prod Y. t_Y \multimap t_Y$	then c is of type $t \rightarrow t$

An example

Sentence:

(4) Aaron helps Moses.

+

Analysis:

label	assigned to	
1	the object argument of <i>helps</i>	<i>Moses</i>
2	the subject argument of <i>helps</i>	<i>Aaron</i>
3	the sentence as a whole	

⇓

Meaning constructors:

m : e_1 **a** : e_2 **help** : $e_1 \multimap (e_2 \multimap t_3)$

Interpretation

Premises:

$$\mathbf{m} : e_1 \quad \mathbf{a} : e_2 \quad \mathbf{help} : e_1 \multimap (e_2 \multimap t_3)$$

\Downarrow

Proof:

$$\frac{\frac{\mathbf{help} : e_1 \multimap (e_2 \multimap t_3) \quad \mathbf{m} : e_1}{\mathbf{help}(\mathbf{m}) : e_2 \multimap t_3} \multimap E \quad \mathbf{a} : e_2}{\mathbf{help}(\mathbf{m})(\mathbf{a}) : t_3} \multimap E$$

back

More rules

Rules for \Box	
Elimination	Introduction
$\frac{f : \Box X(A)}{f : A[t/X]} \Box_E$ <p>t free for X</p>	$\frac{f : A}{f : \Box X(A)} \Box_I$ <p>X not free in any open premise</p>

- These are rules on the linear logic side only, without effect on meaning.
- For example:

$$\frac{\lambda p. \neg p : \Box X. t_X \multimap t_X}{\lambda p. \neg p : t_1 \multimap t_1} \Box_E$$

Another example

Sentence:

(5) Someone helps everyone.

+

Analysis:

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

⇓

Meaning constructors:

$$\lambda P.\forall x.\mathbf{person}(x) \rightarrow P(x) : \prod X.(e_1 \multimap t_X) \multimap t_X$$

$$\mathbf{help} : e_1 \multimap (e_2 \multimap t_3)$$

$$\lambda Q.\exists y.\mathbf{person}(y) \wedge Q(y) : \prod Y.(e_2 \multimap t_Y) \multimap t_Y$$

Surface scope interpretation

Premises:

$$\lambda P.\forall x.\mathbf{person}(x) \rightarrow P(x) : \prod X.(e_1 \multimap t_X) \multimap t_X$$

$$\mathbf{help} : e_1 \multimap (e_2 \multimap t_3)$$

$$\lambda Q.\exists y.\mathbf{person}(y) \wedge Q(y) : \prod Y.(e_2 \multimap t_Y) \multimap t_Y$$

Proof:

$$\begin{array}{c}
 \lambda Q.\exists y.\mathbf{person}(y) \wedge Q(y) : \prod Y.(e_2 \multimap t_Y) \multimap t_Y \\
 \hline
 \lambda Q.\exists y.\mathbf{person}(y) \wedge Q(y) : \prod Y.(e_2 \multimap t_Y) \multimap t_Y \quad \prod_E \\
 \hline
 (e_2 \multimap t_3) \multimap t_3 \\
 \hline
 \exists y.\mathbf{person}(y) \wedge \forall x.\mathbf{person}(x) \rightarrow \mathbf{help}(x)(y) : t_3
 \end{array}
 \quad
 \begin{array}{c}
 \lambda P.\forall x.\mathbf{person}(x) \rightarrow P(x) : \prod X.(e_1 \multimap t_X) \multimap t_X \\
 \hline
 \lambda P.\forall x.\mathbf{person}(x) \rightarrow P(x) : \prod X.(e_1 \multimap t_X) \multimap t_X \quad \prod_E \\
 \hline
 (e_1 \multimap t_3) \multimap t_3 \\
 \hline
 \forall x.\mathbf{person}(x) \rightarrow \mathbf{help}(x)(v) : t_3 \\
 \hline
 \lambda v.\forall x.\mathbf{person}(x) \rightarrow \mathbf{help}(x)(v) : e_2 \multimap t_3 \\
 \hline
 \exists y.\mathbf{person}(y) \wedge \forall x.\mathbf{person}(x) \rightarrow \mathbf{help}(x)(y) : t_3
 \end{array}
 \quad
 \begin{array}{c}
 \mathbf{help} : e_1 \multimap (e_2 \multimap t_3) \quad \frac{[z:]^1}{[e_1]} \multimap_E \\
 \hline
 \mathbf{help}(z) : e_2 \multimap t_3 \\
 \hline
 \mathbf{help}(z)(v) : t_3 \quad \multimap_{I,1} \\
 \hline
 \lambda z.\mathbf{help}(z)(v) : e_1 \multimap t_3 \\
 \hline
 \mathbf{help}(z)(v) : t_3 \quad \multimap_E, \Rightarrow_\beta \\
 \hline
 \forall x.\mathbf{person}(x) \rightarrow \mathbf{help}(x)(v) : t_3 \\
 \hline
 \lambda v.\forall x.\mathbf{person}(x) \rightarrow \mathbf{help}(x)(v) : e_2 \multimap t_3 \\
 \hline
 \exists y.\mathbf{person}(y) \wedge \forall x.\mathbf{person}(x) \rightarrow \mathbf{help}(x)(y) : t_3
 \end{array}$$

Inverse scope interpretation

Premises:

$$\lambda P.\forall x.\mathbf{person}(x) \rightarrow P(x) : \Pi X.(e_1 \multimap t_X) \multimap t_X$$

$$\mathbf{help} : e_1 \multimap (e_2 \multimap t_3)$$

$$\lambda Q.\exists y.\mathbf{person}(y) \wedge Q(y) : \Pi Y.(e_2 \multimap t_Y) \multimap t_Y$$

Proof:

$$\begin{array}{c}
 \lambda P.\forall x.\mathbf{person}(x) \rightarrow P(x) : \Pi X.(e_1 \multimap t_X) \multimap t_X \\
 \hline
 \lambda P.\forall x.\mathbf{person}(x) \rightarrow P(x) : \Pi_E \\
 \hline
 (e_1 \multimap t_3) \multimap t_3
 \end{array}
 \quad
 \begin{array}{c}
 \lambda Q.\exists y.\mathbf{person}(y) \wedge Q(y) : \Pi Y.(e_2 \multimap t_Y) \multimap t_Y \\
 \hline
 \lambda Q.\exists y.\mathbf{person}(y) \wedge Q(y) : \Pi_E \\
 \hline
 (e_2 \multimap t_3) \multimap t_3
 \end{array}
 \quad
 \begin{array}{c}
 \mathbf{help} : \\
 e_1 \multimap (e_2 \multimap t_3) \quad [z : e_1]^1 \\
 \hline
 \mathbf{help}(z) : e_2 \multimap t_3 \\
 \hline
 \multimap_E, \Rightarrow_\beta \\
 \hline
 \exists y.\mathbf{person}(y) \wedge \mathbf{help}(z)(y) : t_3 \\
 \hline
 \multimap_{I,1} \\
 \hline
 \lambda z.\exists y.\mathbf{person}(y) \wedge \mathbf{help}(z)(y) : e_1 \multimap t_3 \\
 \hline
 \multimap_E, \Rightarrow_\beta \\
 \hline
 \forall x.\mathbf{person}(x) \rightarrow \exists y.\mathbf{person}(y) \wedge \mathbf{help}(x)(y) : t_3
 \end{array}$$

Implementation in Minimalism

Basic ideas

- Syntactic objects have features.
- The structure-building operation(s) (Merge) is/are based on the matching of features.
- Every feature bears an index, and when two features match their indices must also match.
- Those indices are used for the labels on linear logic formulae paired with interpretations, thereby providing the syntax/semantics connection.

Features

Largely based on Adger (2003, 2010)

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

- Every feature is specified for interpretability, either *interpretable* or *uninterpretable*.
 - Interpretable features describe what an LI *is*.
 - Uninterpretable features describe what an LI *needs*.
- Every uninterpretable feature is specified for strength, either *weak* or *strong*. Strong features trigger movement.
- The set of *categorial* features is a proper subset of the set of features. For this paper, the categorial features are N(oun), V(erb), D(eterminer), P(reposition), C(omplementizer) and T(ense).

For example:

interpretable	uninterpretable		Determiner features
	weak	strong	
D	<i>uD</i>	<i>uD*</i>	

Hierarchy of projections

Every interpretable categorial feature belongs to at most one hierarchy of projections (HoPs). Adger (2003) has:

Clausal: $C > T > (\text{Neg}) > (\text{Perf}) > (\text{Prog}) > (\text{Pass}) > v > V$

Nominal: $D > (\text{Poss}) > n > N$

Adjectival: $(\text{Deg}) > A$

We'll use:

Clausal: $C > T > V$

Nominal: $D > N$

Lexical items

A *feature structure* is an ordered pair $\langle A, B \rangle$ where:

- A is a set of interpretable features, exactly one of which is a categorial feature.
- B is a (possibly empty) sequence of uninterpretable features.

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

$$\langle \{V\}, \langle uD, uD \rangle \rangle$$

|
help

which I'll often represent as:

$$\begin{array}{c} V \\ \langle uD, uD \rangle \end{array}$$

|
help

$$\langle \{T, \text{pres}\}, \langle uD^* \rangle \rangle$$

|
-S

often:

$$\begin{array}{c} T[\text{pres}] \\ \langle uD^* \rangle \end{array}$$

|
-S

Structure-building operation(s)

Merge.

- Hierarchy of Projections-driven.
- Selectional features-driven.
 - External.
 - Internal.

HoPs merge

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

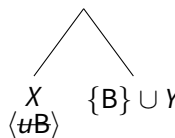
Where A and B are in the same hierarchy of projections (HoPs) and A is higher on that HoPs than B

In this and the following slides,

- A and B stand for arbitrary (interpretable) features,
- X and Y stand for arbitrary sets of (interpretable) features, and
- Σ stands for an arbitrary sequence of (uninterpretable) features.

Select merge

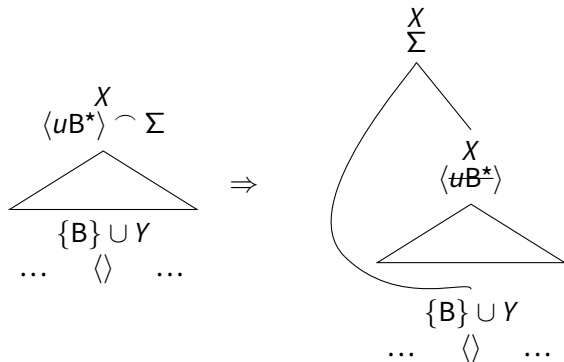
External

$$\langle uB \rangle \overset{X}{\frown} \Sigma + \{B\} \cup Y \Rightarrow$$


\frown indicates sequence concatenation.

Select merge

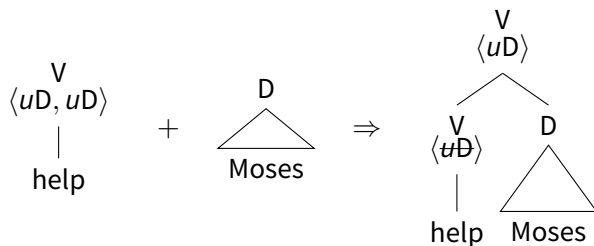
Internal



This requires an additional constraint to the effect that the constituent that remerges is the closest matching one to the head of the input tree.

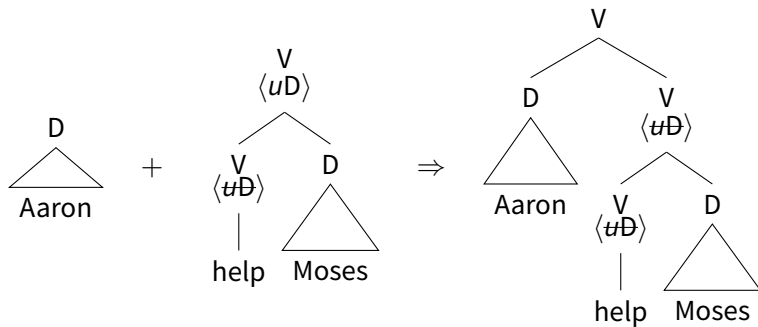
External merge

An example



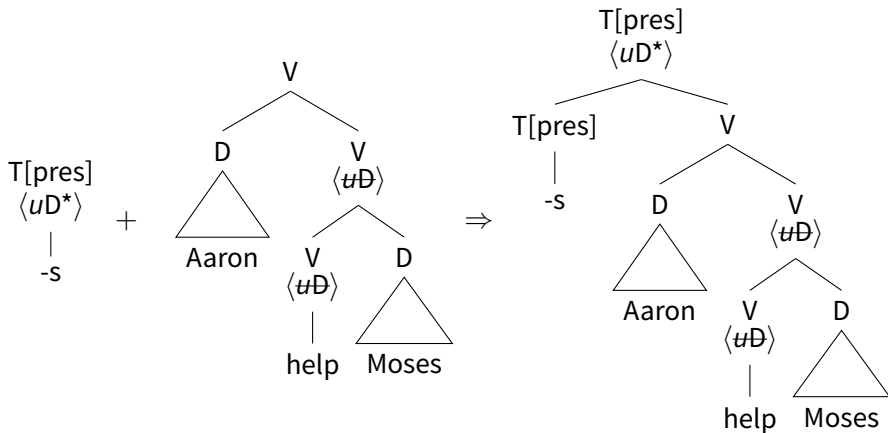
External merge

An example



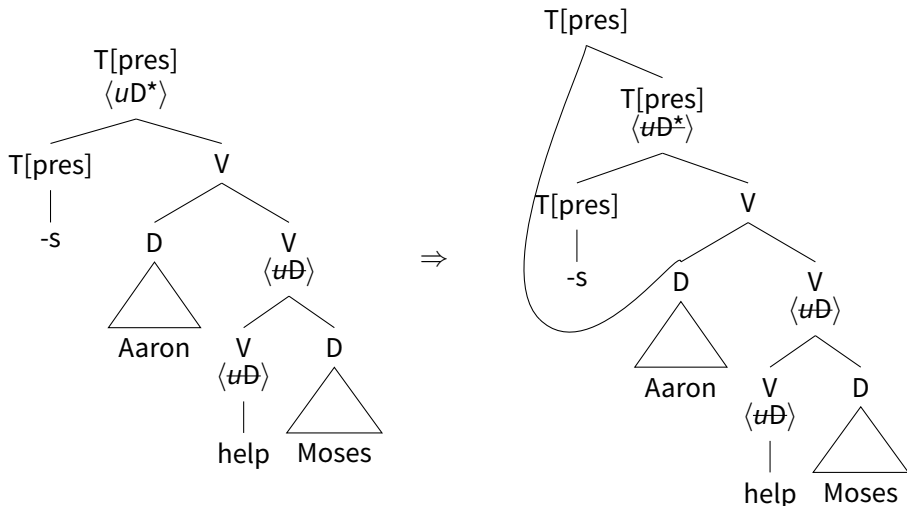
HoPs merge

An example



Internal merge

An example



Indices on features

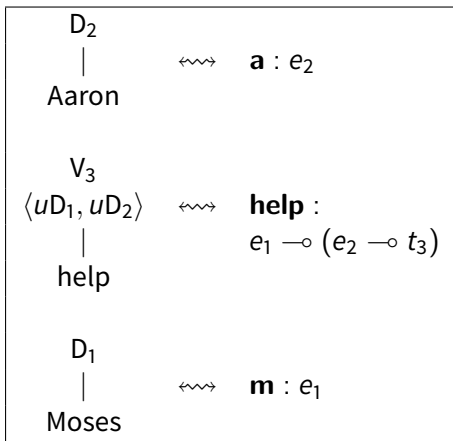
To provide the connection to (Glue) semantics, the features contained in lexical items bear indices (subject to constraints). For example:

$$\begin{array}{c}
 V_j \\
 \langle uD_j, uD_k \rangle \\
 | \\
 \text{help}
 \end{array}
 \quad \longleftrightarrow \quad
 \text{help} : e_j \multimap (e_k \multimap t_j)$$

$$i \neq j, i \neq k, j \neq k$$

Structure-building operations are sensitive to indices in that the indices on the matching features must also match.

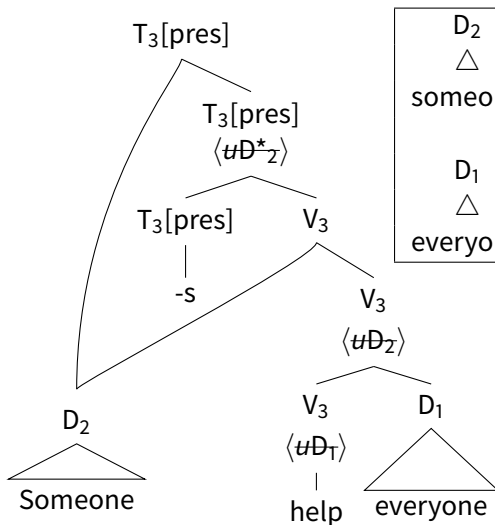
with indices



37 / 63

Someone helps everyone

with indices



D_2	$\lambda Q. \exists y. \text{person}(y) \wedge Q(y) :$
\triangle	$\iff \Box Y. (e_2 \multimap t_Y) \multimap t_Y$
someone	
D_1	$\lambda P. \forall x. \text{person}(y) \rightarrow Q(y) :$
\triangle	$\iff \Box X. (e_1 \multimap t_X) \multimap t_X$
everyone	

interpretations

Comparison with the mainstream approach

Can we have traces?

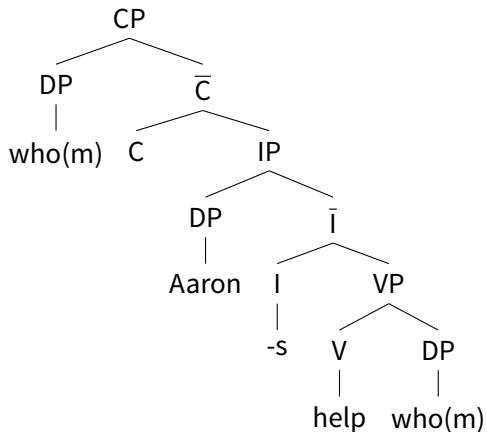
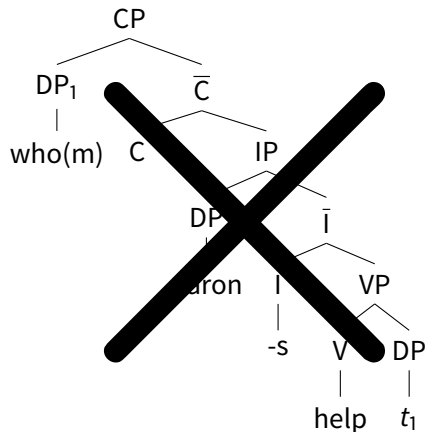
A “perfect language” should meet the condition of inclusiveness: any structure formed by the computation [...] is constituted of elements already present in the lexical items selected for N; no new objects are added in the course of computation apart from rearrangements of lexical properties (in particular, no indices [...]).

(Chomsky 1995: 228)

Trace theory was abandoned in early minimalism in favor of the so-called copy theory of movement. Indices were deemed incompatible with the principle of Inclusiveness, which restricts the content of tree structures to information originating in the lexicon. Because indices of phrases cannot be traced back to any lexical entry, they are illegitimate syntactic objects.

(Neeleman & van de Koot 2010: 331)

The copy theory of movement



If we assume that the computational system of syntax doesn't use variables, variables are introduced at the point where the LF-structure of a sentence is translated into a semantic representation.

(Sauerland 1998: 196)

the semantic component can treat lower copies as variables

(Fox 2002: 66)

But how?

For the identity of copies to replace coindexation, it must be possible to distinguish between identical constituents that stand in a movement relation and identical constituents that are merged separately. However, the copy theory provides no way of doing so. [...] Chomsky, during a keynote address in 2004, suggested that the computational system “knows” which copies have been created by movement. One implication of this position is that the input to the interface with semantics is not a tree, but an ordered set of trees. If taken seriously, this requires an additional— nontrivial— mechanism that extracts the relevant information from this set.

(Neeleman & van de Koot 2010: 332)

- (6) a. Some man arrived.
 b. $[_{IP} [_{DP} \text{some man}] [_{I'} I [_{VP} \text{arrived} [_{DP} \text{some man}]]]]$

*Let semantic composition proceed in a bottom-up manner, starting from the lower instance of man. When the DP node dominating some man is reached (yielding, say, $\lambda X \exists x [\text{man}'(x) \wedge X(x)]$ as a translation), it is discovered that this DP is a movement trace. **(Exactly how it is determined that an item is a trace—that is, the bottom element of a chain—is a technical question inherent in the copy theory** that is not particular to my proposal; assume for concreteness that the presence of unchecked uninterpretable features [...] indicates that the element is (part of) a trace.)*

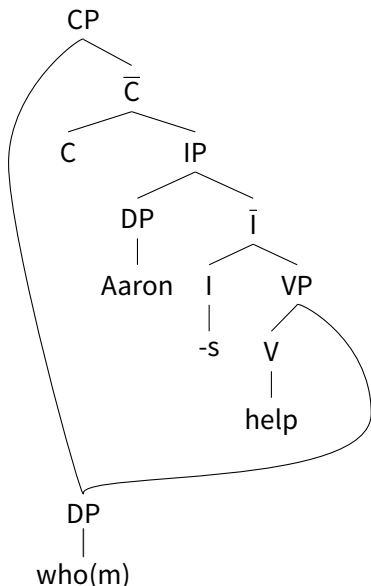
(Ruys 2015: 458)
(emphasis mine)

- (7) a. Some man arrived.
 b. $[_{IP} [_{DP} \text{some man}] [_{I'} I [_{VP} \text{arrived} [_{DP} \text{some man}]]]]$

*Therefore, what composes with arrived' is not the regular translation $(\lambda X \exists x [\text{man}'(x) \wedge X(x)])$ computed so far; this is discarded in favor of a simple variable. The variable so obtained is subsequently λ -bound at the landing site I' , and the resulting λ -expression composes with the translation of the upstairs copy of some man. **This process of first calculating and then discarding the regular semantics of a trace copy may appear superfluous, but it is difficult to avoid under the copy theory.***

(Ruys 2015: 458–9)
 (**emphasis** mine)

An alternative: multidominance

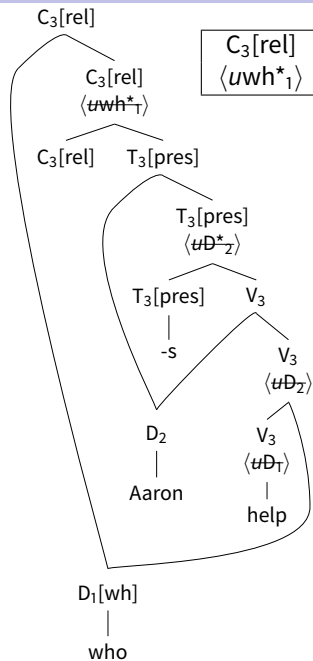


The issue now is that we can't apply definitions like this (Heim & Kratzer 1998: 95):

Functional Application (FA)

If α is a branching node and $\{\beta, \gamma\}$ the set of its daughters, then, for any assignment a , if $\llbracket \beta \rrbracket^a$ is a function whose domain contains $\llbracket \gamma \rrbracket^a$, then $\llbracket \alpha \rrbracket^a = \llbracket \beta \rrbracket^a (\llbracket \gamma \rrbracket^a)$

The reason being that you now have to know whether or not β and γ have other mothers besides α .



$$C_3[\text{rel}] \quad \longleftrightarrow \quad \lambda P. \lambda Q. \lambda x. P(x) \wedge Q(x) :$$

$$\langle \text{wh}^*_1 \rangle \quad (e_1 \multimap t_3) \multimap ((e_1 \multimap t_3) \multimap (e_1 \multimap t_3))$$

Interpretation

$$\begin{array}{c}
 \lambda P.\lambda Q.\lambda x.P(x) \wedge Q(x) : \\
 (e_1 \multimap t_3) \multimap \\
 ((e_1 \multimap t_3) \multimap (e_1 \multimap t_3)) \\
 \hline
 \lambda Q.\lambda x.\mathbf{help}(x)(\mathbf{a}) \wedge Q(x) : (e_1 \multimap t_3) \multimap (e_1 \multimap t_3)
 \end{array}
 \quad
 \begin{array}{c}
 \mathbf{help} : \\
 \frac{e_1 \multimap (e_2 \multimap t_3) \quad [y : e_1]^1}{\mathbf{help}(y) : e_2 \multimap t_3} \multimap_E \\
 \frac{\mathbf{help}(y)(\mathbf{a}) : t_3}{\lambda y.\mathbf{help}(y)(\mathbf{a}) : e_1 \multimap t_3} \multimap_{I,1} \\
 \frac{\lambda y.\mathbf{help}(y)(\mathbf{a}) : e_1 \multimap t_3 \quad \mathbf{a} : e_2}{\lambda y.\mathbf{help}(y)(\mathbf{a}) : e_1 \multimap t_3} \multimap_E \\
 \multimap_E, \Rightarrow_\beta
 \end{array}$$

But there are still indices!

Yes, but,

- They are ‘already present in the lexical items selected’.
- They are not ‘added in the course of computation’...
- ...They are *resolved* in the course of computation.
- So Inclusiveness is respected.

Scope within DP

(8) A fan of every band cheered.

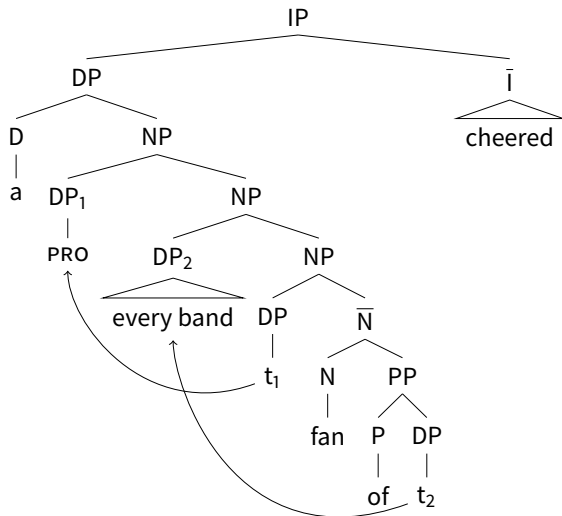
Two readings:

(sur) $\exists x. \forall y (\mathbf{band}(y) \rightarrow \mathbf{fan-of}(y)(x)) \wedge \mathbf{cheer}(x)$

(inv) $\forall y. \mathbf{band}(y) \rightarrow \exists x. \mathbf{fan-of}(y)(x) \wedge \mathbf{cheer}(x)$

The surface scope reading

According to Heim & Kratzer (1998)

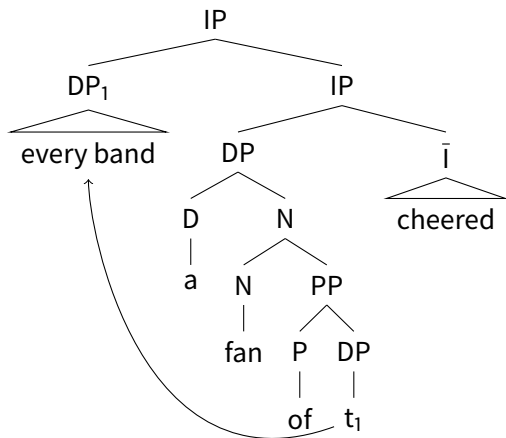


Discussion

- If one is committed to a purely QR-based account of scope ambiguity then this kind of manoeuvre is unavoidable: in order for the DP ‘every band’ to take scope within the NP containing it, that NP has to be made clause-help so that the DP has a node of type t to adjoin to.
- But what independent evidence is there for the existence of a subject position within NP filled by a phonologically and semantically null pronoun?
- The alternative is to allow some kind of type-shifting operation so that the embedded DP can be interpreted *in situ*, but once this kind of type-shifting is added to the system then the motivation for QR in general is weakened.

The inverse-linking reading

According to May (1977)



Discussion

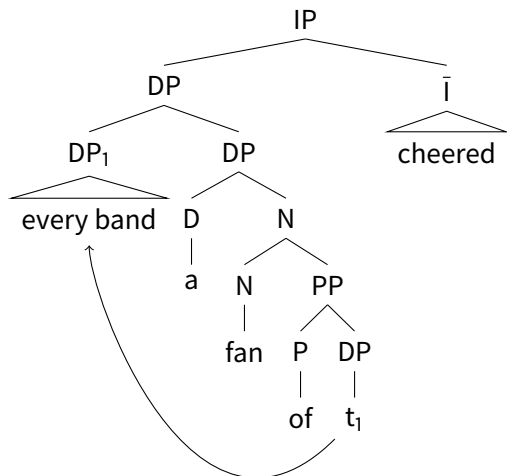
Based on May & Bale 2007

- This involves (covert) movement out of the subject DP, weakening the analogy between covert and overt movement (and hence the plausibility of covert movement overall).
- Furthermore, if this covert movement is possible then the interpretation of (9-a) shown in (9-b) should be possible, but it isn't (Larson's generalization).

- (9) a. A fan of every band sang no songs.
b. $\forall y.\mathbf{band}(y) \rightarrow \neg\exists z.\mathbf{song}(z) \wedge \exists x.\mathbf{fan-of}(y)(x) \wedge \mathbf{sing}(z)(x)$

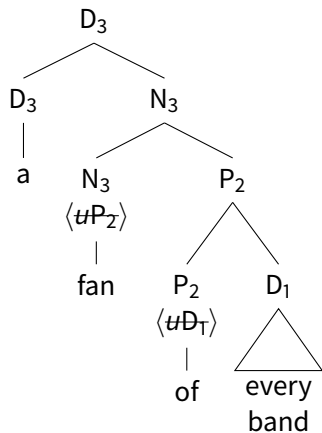
The inverse-linking reading

According to May (1985)



For this structure to be interpreted requires additional compositional principles and/or type-shifting operations, thereby reducing the motivation for QR in general.

A Glue analysis



D ₃ a	$\Leftarrow \rightsquigarrow$	$\lambda P. \lambda Q. \exists x. P(x) \wedge Q(x) : (e_3 \multimap t_3) \multimap \prod X. (e_3 \multimap t_X) \multimap t_X$
N ₃ $\langle \#P_2 \rangle$ fan	$\Leftarrow \rightsquigarrow$	fan-of : $e_2 \multimap (e_3 \multimap t_3)$
P ₂ $\langle \#D_1 \rangle$ of	$\Leftarrow \rightsquigarrow$	$\lambda v. v : e_1 \multimap e_2$
D ₁ △ every band	$\Leftarrow \rightsquigarrow$	$\lambda F. \forall y. \mathbf{band}(y) \rightarrow F(y) : \prod Y. (e_1 \multimap t_Y) \multimap t_Y$

Surface scope interpretation

Part I

$$\begin{array}{c}
 \frac{\lambda F.\forall y.\mathbf{band}(y) \rightarrow F(y) : \quad \frac{\frac{\frac{}{\Pi.(e_1 \multimap t_y) \multimap t_y}}{\lambda F.\forall y.\mathbf{band}(y) \rightarrow F(y) :} \Pi_E \quad (e_1 \multimap t_3) \multimap t_3}{\lambda F.\forall y.\mathbf{band}(y) \rightarrow F(y) :} \Pi_E}{\lambda F.\forall y.\mathbf{band}(y) \rightarrow F(y) :} \Pi_E \\
 \frac{\lambda F.\forall y.\mathbf{band}(y) \rightarrow F(y) : \quad \frac{\frac{\frac{\mathbf{fan-of} : \quad \frac{\lambda v.v : \quad \frac{e_1 \multimap e_2 \quad [u:]^1}{u : e_2} \multimap_E, \Rightarrow_\beta}{e_2 \multimap (e_3 \multimap t_3)} \multimap_E}{\mathbf{fan-of}(u) : e_3 \multimap t_3} \multimap_E}{\mathbf{fan-of}(u)(v) : t_3} \multimap_{I,1}}{\lambda u.\mathbf{fan-of}(u)(v) : e_1 \multimap t_3} \multimap_{I,1}}{\lambda v.\forall y.\mathbf{band}(y) \rightarrow \mathbf{fan-of}(y)(v) : t_3} \multimap_{I,2}}{\lambda v.\forall y.\mathbf{band}(y) \rightarrow \mathbf{fan-of}(y)(v) : e_3 \multimap t_3} \multimap_{I,2}
 \end{array}$$

Surface scope interpretation

Part II

$$\begin{array}{c}
 \text{[Part I]} \\
 \Downarrow \\
 \frac{\lambda P. \lambda Q. \exists x. P(x) \wedge Q(x) : (e_3 \multimap t_3) \multimap \prod X. (e_3 \multimap t_X) \multimap t_X \quad \lambda v. \forall y. \mathbf{band}(y) \rightarrow \mathbf{fan-of}(y)(v) : e_3 \multimap t_3}{\lambda Q. \exists x. \forall y (\mathbf{band}(y) \rightarrow \mathbf{fan-of}(y)(x)) \wedge Q(x) : \prod X. (e_3 \multimap t_X) \multimap t_X} \multimap_E, \Rightarrow_\beta
 \end{array}$$

Remarks

- We now have a surface scope interpretation of $[_{DP}$ a fan of every band] without the need for a subject position within **the NP** in **the syntax**.
- The reason is that, effectively, we can have a subject position for **the NP interpretation** within **the proof**, in the form of an auxiliary hypothesis (in this case, $v : e_3$) that is later discharged.
- This requires no additions to the Glue system that has already been set up for the interpretation of other structures.

Inverse linking interpretation

Part I

$$\begin{array}{c}
 \lambda P. \lambda Q. \exists x. P(x) \wedge Q(x) : \\
 (e_3 \multimap t_3) \multimap \\
 \Pi X. (e_3 \multimap t_X) \multimap t_X \\
 \hline
 \lambda Q. \exists x. \mathbf{fan-of}(u)(x) \wedge Q(x) : \Pi X. (e_3 \multimap t_X) \multimap t_X \\
 \hline
 \lambda Q. \exists x. \mathbf{fan-of}(u)(x) \wedge Q(x) : (e_3 \multimap t_Z) \multimap t_Z
 \end{array}
 \quad
 \begin{array}{c}
 \mathbf{fan-of} : \\
 e_2 \multimap (e_3 \multimap t_3) \\
 \hline
 \mathbf{fan-of}(u) : e_3 \multimap t_3 \\
 \hline
 \multimap_E
 \end{array}
 \quad
 \begin{array}{c}
 \lambda v. v : \\
 e_1 \multimap e_2 \quad \frac{[u :]^1}{e_1} \\
 \hline
 u : e_2 \\
 \hline
 \multimap_E, \Rightarrow_\beta
 \end{array}
 \quad
 \begin{array}{c}
 \multimap_E \\
 \hline
 \Pi_E
 \end{array}$$

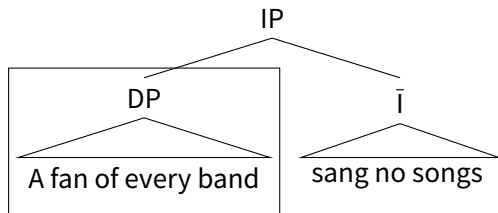
Part II

$$\begin{array}{c}
\text{[Part I]} \\
\Downarrow \\
\frac{\lambda F. \forall y. \mathbf{band}(y) \rightarrow F(y) : \prod. (e_1 \multimap t_Y) \multimap t_Y}{\lambda F. \forall y. \mathbf{band}(y) \rightarrow F(y) : (e_1 \multimap t_Z) \multimap t_Z} \prod_E \quad \frac{\lambda Q. \exists x. \mathbf{fan-of}(u)(x) \wedge Q(x) : (e_3 \multimap t_Z) \multimap t_Z \quad \left[\frac{P : }{e_3 \multimap t_Z} \right]^2}{\exists x. \mathbf{fan-of}(u)(x) \wedge P(x) : t_Z} \multimap_E, \Rightarrow_\beta \\
\frac{\lambda F. \forall y. \mathbf{band}(y) \rightarrow F(y) : (e_1 \multimap t_Z) \multimap t_Z \quad \frac{\exists x. \mathbf{fan-of}(u)(x) \wedge P(x) : t_Z}{\lambda u. \exists x. \mathbf{fan-of}(u)(x) \wedge P(x) : e_1 \multimap t_Z} \multimap_{I,1}}{\lambda u. \exists x. \mathbf{fan-of}(u)(x) \wedge P(x) : e_1 \multimap t_Z} \multimap_E, \Rightarrow_\beta \\
\frac{\lambda u. \exists x. \mathbf{fan-of}(u)(x) \wedge P(x) : e_1 \multimap t_Z}{\forall y. \mathbf{band}(y) \rightarrow \exists x. \mathbf{fan-of}(y)(x) \wedge P(x) : t_Z} \multimap_{I,2} \\
\frac{\forall y. \mathbf{band}(y) \rightarrow \exists x. \mathbf{fan-of}(y)(x) \wedge P(x) : t_Z}{\lambda P. \forall y. \mathbf{band}(y) \rightarrow \exists x. \mathbf{fan-of}(y)(x) \wedge P(x) : (e_3 \multimap t_Z) \multimap t_Z} \multimap_{I,1} \\
\frac{\lambda P. \forall y. \mathbf{band}(y) \rightarrow \exists x. \mathbf{fan-of}(y)(x) \wedge P(x) : (e_3 \multimap t_Z) \multimap t_Z}{\lambda P. \forall y. \mathbf{band}(y) \rightarrow \exists x. \mathbf{fan-of}(y)(x) \wedge P(x) : \prod Z. (e_3 \multimap t_Z) \multimap t_Z} \prod_I
\end{array}$$

We now have an inversely-linked interpretation of the DP ‘a fan of every band’, *using only premises contributed by words within the DP*.

Scope islands and proof goals

A possible way of stating the claim that DP is a scope island.



- (10) From all and only the premises contributed from within this constituent, construct a proof with conclusion of type $(e \rightarrow t) \rightarrow t \dots$

...which can then serve as a premise in the proofs for the interpretations of the IP.

Observations

- Both the surface scope and inverse-linking interpretations of [_{DP} a fan of every band] given above conform to the constraint given in (10).
- Neither one makes the (apparently impossible) interpretation of (9-a) given in (9-b) possible.
- There is no conflict between inverse linking, and the claim that DP is a scope island, in the Glue framework.

References I

- Adger, David. 2003. *Core syntax: A Minimalist approach*. (Core Linguistics). Oxford: Oxford University Press.
- Adger, David. 2010. A Minimalist theory of feature structure. In Anna Kibort & Greville G. Corbett (eds.), *Features: Perspectives on a key notion in linguistics*, 185–218. Oxford: Oxford University Press.
- Andrews, Avery D. 2010. Propositional glue and the projection architecture of LFG. *Linguistics and Philosophy* 33. 141–170.
- Asudeh, Ash & Richard Crouch. 2002. Glue Semantics for HPSG. In Frank van Eynde, Lars Hellan & Dorothee Beermann (eds.), *Proceedings of the 8th international HPSG conference*. Stanford, CA: CSLI Publications.
- Chomsky, Noam. 1995. *The minimalist program*. Cambridge, MA: MIT Press.

References II

- Crouch, Richard & Josef van Genabith. 2000. Linear logic for linguists. ESSLLI 2000 course notes. Archived 2006-10-19 in the Internet Archive at http://web.archive.org/web/20061019004949/http://www2.parc.com/istl/members/crouch/esslli00_notes.pdf.
- Dalrymple, Mary, Vineet Gupta, John Lamping & Vijay Saraswat. 1999. Relating resource-based semantics to categorial semantics. In Mary Dalrymple (ed.), *Semantics and syntax in Lexical Functional Grammar: The resource logic approach*, 261–280. Cambridge, MA: MIT Press.
- Dalrymple, Mary, John Lamping & Vijay Saraswat. 1993. LFG semantics via constraints. In Steven Krauwer, Michael Moortgat & Louis des Tombe (eds.), *Eacl 1993: Proceedings of the sixth conference of the European chapter of the Association for Computational Linguistics*, 97–105. Universiteit Utrecht.
- Fox, Danny. 2002. Antecedent-contained deletion and the copy theory of movement. *Linguistic Inquiry* 33(1). 63–96.

References III

- Frank, Anette & Josef van Genabith. 2001. GlueTag: linear logic-based semantics for LTAG—and what it teaches us about LFG and LTAG. In Miriam Butt & Tracy Holloway King (eds.), *Proceedings of the LFG01 conference*. Stanford, CA: CSLI Publications.
- Girard, Jean-Yves. 1987. Linear logic. *Theoretical Computer Science* 50(1). 1–101.
- Heim, Irene & Angelika Kratzer. 1998. *Semantics in generative grammar*. (Blackwell Textbooks in Linguistics 13). Oxford: Wiley-Blackwell.
- Howard, W.A. 1980. The formulae-as-types notion of construction. In J.P. Seldin & J.R. Hindley (eds.), *To H.B. Curry: Essays on combinatory logic, lambda calculus and formalism*, 479–490. New York: Academic Press.
- May, Robert. 1977. *The grammar of quantification*. Massachusetts Institute of Technology dissertation.
- May, Robert. 1985. *Logical form: Its structure and derivation*. (Linguistic Inquiry Monographs 12). Cambridge, MA: MIT Press.

References IV

- May, Robert & Alan Bale. 2007. Inverse linking. In Martin Everaert, Henk van Riemsdijk, Rob Goedemans & Bart Hollebrandse (eds.), *The Blackwell companion to syntax*, vol. 2, chap. 36. Oxford: Blackwell.
- Moot, Richard. 2002. *Proof nets for linguistic analysis*. University of Utrecht dissertation.
- Morrill, Glyn V. 1994. *Type logical grammar: Categorical logic of signs*. Dordrecht, Netherlands: Kluwer Academic Publishers.
- Neeleman, Ad & Hans van de Koot. 2010. A local encoding of syntactic dependencies and its consequences for the theory of movement. *Syntax* 13(4). 331–372.
- Ruys, E.G. 2015. A Minimalist condition on semantic reconstruction. *Linguistic Inquiry* 46(3). 453–488.
- Sauerland, Uli. 1998. *The meaning of chains*. Massachusetts Institute of Technology dissertation.

Whither LF?

- The Glue implementation eliminates the need for covert movement, and hence for a level of syntax formed as the result of covert movement, such as Logical Form (at least, to the extent that these are motivated by considerations of scope).
- An alternative conception would be to take *the proofs themselves* as LFs, since every one of them is associated with exactly one interpretation.
- However, we needn't/shouldn't identify the proofs with the particular natural deduction representations given here, since proofs can be represented in many different ways.

Surface scope interpretation

Matthew Gotham (Oslo)

Sequent calculus

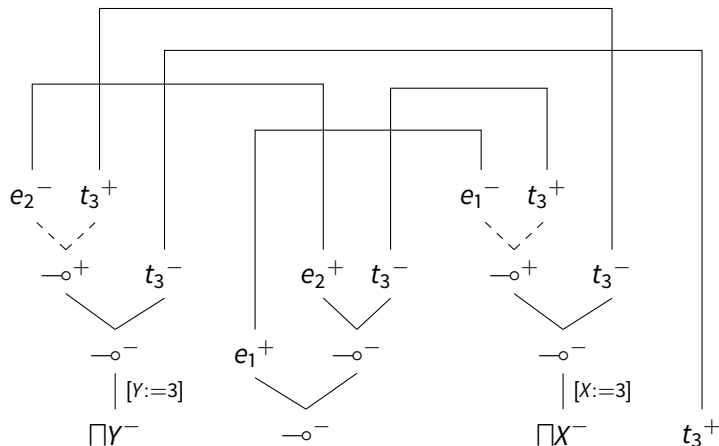
Inverse scope interpretation

$$\begin{array}{c}
 \frac{\frac{\frac{}{e_1 \vdash e_1} \quad \frac{\frac{}{e_2 \vdash e_2} \quad \frac{}{t_3 \vdash t_3}}{e_2, e_2 \multimap t_3 \vdash t_3} \multimap_L}{e_2, e_1, e_1 \multimap (e_2 \multimap t_3) \vdash t_3} \multimap_L}{e_1, e_1 \multimap (e_2 \multimap t_3) \vdash e_2 \multimap t_3} \multimap_R \quad \frac{}{t_3 \vdash t_3} \multimap_L \\
 \frac{e_1, (e_2 \multimap t_3) \multimap t_3, e_1 \multimap (e_2 \multimap t_3) \vdash t_3}{(e_2 \multimap t_3) \multimap t_3, e_1 \multimap (e_2 \multimap t_3) \vdash e_1 \multimap t_3} \multimap_R \quad \frac{}{t_3 \vdash t_3} \multimap_L \\
 \frac{(e_2 \multimap t_3) \multimap t_3, e_1 \multimap (e_2 \multimap t_3), (e_1 \multimap t_3) \multimap t_3 \vdash t_3}{(e_2 \multimap t_3) \multimap t_3, e_1 \multimap (e_2 \multimap t_3), \Box X((e_1 \multimap t_X) \multimap t_X) \vdash t_3} \multimap_L \\
 \frac{(e_2 \multimap t_3) \multimap t_3, e_1 \multimap (e_2 \multimap t_3), \Box X((e_1 \multimap t_X) \multimap t_X) \vdash t_3}{\Box Y((e_2 \multimap t_Y) \multimap t_Y), e_1 \multimap (e_2 \multimap t_3), \Box X((e_1 \multimap t_X) \multimap t_X) \vdash t_3} \Box_L
 \end{array}$$

Proof net I

(Moot 2002: Chapter 5)

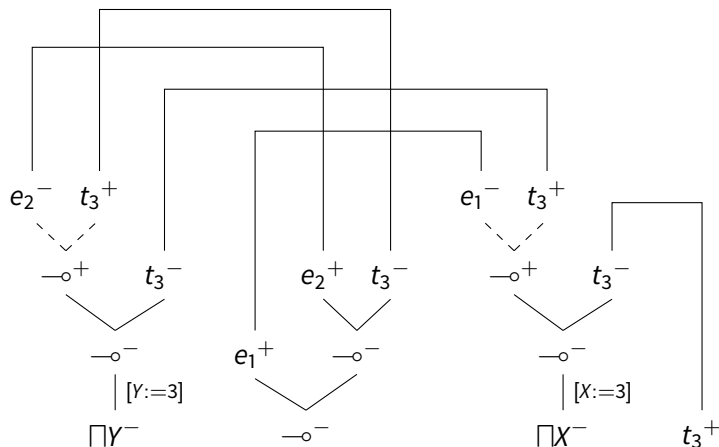
Surface scope interpretation



Proof net I

(Moot 2002: Chapter 5)

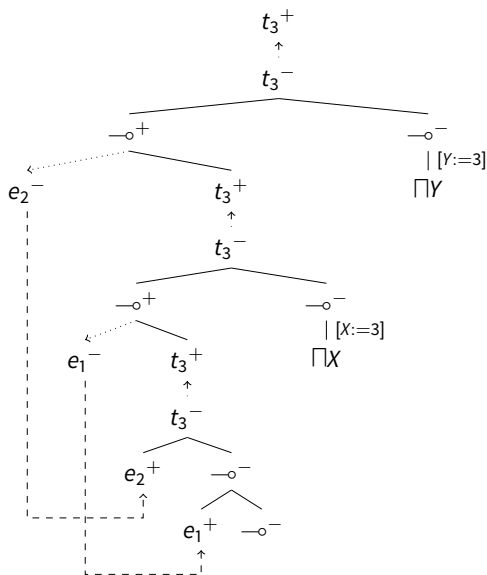
Inverse scope interpretation



Proof net II

(Adapted from Andrews 2010)

Surface scope interpretation



Proof net II

(Adapted from Andrews 2010)

Inverse scope interpretation

