On Internal Merge

Steedman (2023) on CCGs and Minimalism

Isabella Senturia

Hopf Algebra seminar

October 9, 2023

Introduction

Introduction

Inclusiveness Condition (Chomsky, 1995): The output of a system must not contain anything beyond its input, meaning syntactic derivation rules cannot add information (indices, traces, syntactic categories, etc.) to whatever has been specified in the categories that the derivation begins with.

Introduction

Inclusiveness Condition (Chomsky, 1995): The output of a system must not contain anything beyond its input, meaning syntactic derivation rules cannot add information (indices, traces, syntactic categories, etc.) to whatever has been specified in the categories that the derivation begins with.

- This is taken as a core tenet of Steedman (2023): a derivation begins with inputs specifying exactly the combinatory potential to be used.

Introduction

Inclusiveness Condition (Chomsky, 1995): The output of a system must not contain anything beyond its input, meaning syntactic derivation rules cannot add information (indices, traces, syntactic categories, etc.) to whatever has been specified in the categories that the derivation begins with.

- This is taken as a core tenet of Steedman (2023): a derivation begins with inputs specifying exactly the combinatory potential to be used.
- The universal, type–dependent and language–independent rules are then applied to combine the inputs, which do not alter information.

Introduction (cont.)

- How to deal with long–distance dependencies, such as long–distance agreement or movement?

Introduction (cont.)

- How to deal with long–distance dependencies, such as long–distance agreement or movement?
- Unite internal merge with external merge without adding discontinuity into the grammar, using as minimal expressive power as possible.

Introduction (cont.)

- How to deal with long–distance dependencies, such as long–distance agreement or movement?
- Unite internal merge with external merge without adding discontinuity into the grammar, using as minimal expressive power as possible.
  - Contrary to other proposals that eliminate movement (GPSG trace–feature passing, CG–nonstandard constituency, LFG lexicalization of locality, HPSG structural unification, etc.).

Introduction (cont.)

- How to deal with long–distance dependencies, such as long–distance agreement or movement?
- Unite internal merge with external merge without adding discontinuity into the grammar, using as minimal expressive power as possible.
  - Contrary to other proposals that eliminate movement (GPSG trace–feature passing, CG–nonstandard constituency, LFG lexicalization of locality, HPSG structural unification, etc.).
  - These are too restrictive (grammar collapses to CFG) or too expressive (require additional constraints to then limit this).

CCG solution to Internal Merge

- The link between the internally merged item and its long–distance relationship item must be established before the start of the derivation.

CCG solution to Internal Merge

- The link between the internally merged item and its long–distance relationship item must be established before the start of the derivation.
- Do this in the logical lexical form as binder $\lambda\alpha$ and variable α.

CCG solution to Internal Merge

- The link between the internally merged item and its long–distance relationship item must be established before the start of the derivation.
- Do this in the logical lexical form as binder $\lambda\alpha$ and variable α.
  - Thus, internal merge (IM) is not different than external merge (EM).

CCG solution to Internal Merge

- The link between the internally merged item and its long–distance relationship item must be established before the start of the derivation.
- Do this in the logical lexical form as binder $\lambda\alpha$ and variable α.
  - Thus, internal merge (IM) is not different than external merge (EM).
- Expressive power of this system: “low near–context–free".

CCG solution to Internal Merge

- The link between the internally merged item and its long–distance relationship item must be established before the start of the derivation.
- Do this in the logical lexical form as binder $\lambda\alpha$ and variable α.
  - Thus, internal merge (IM) is not different than external merge (EM).
- Expressive power of this system: "low near–context–free".
- Categories are defined functionally and semantically with functions and arguments, and IM discontinuities base–generated in the logical forms.

Topics

- Components of CCG
  - Lexical items
  - Function application/composition
  - Type raising
- External Merge
- Internal Merge
- Linguistic examples

Defining CCGs and External Merge

Components of CCG

(1) Bare Phrase Structural notation for linguistic categories

Components of CCG

(1) Bare Phrase Structural notation for linguistic categories

- *The Categorial Assumption:* Linguistic Categories are defined syntactically and semantically as functions and/or arguments.

Components of CCG

(1) Bare Phrase Structural notation for linguistic categories

- *The Categorial Assumption:* Linguistic Categories are defined syntactically and semantically as functions and/or arguments.

(2) Contiguous merger universal rules

Components of CCG

(1) Bare Phrase Structural notation for linguistic categories

- *The Categorial Assumption:* Linguistic Categories are defined syntactically and semantically as functions and/or arguments.

(2) Contiguous merger universal rules

- *The Adjacency Assumption:* Rules are purely functional binary operations, limited to application, composition, and substitution, applying to strictly adjacent, phonologically-realized categories, which synchronously and monotonically compose logical forms (lf) and concatenate phonological forms (pf).

(1) Linguistic category labels of lexical items

Every lexical item has two components:

(1) Linguistic category labels of lexical items

Every lexical item has two components:

- CG lexical category: function (specified for linearization order, which is itself a formal feature) X/Y or X\Y or argument (with agreement features)

(1) Linguistic category labels of lexical items

Every lexical item has two components:

- CG lexical category: function (specified for linearization order, which is itself a formal feature) X/Y or X\Y or argument (with agreement features)
  - X, Y variables over categories ($S, NP_{3pl}, S \backslash NP_{3pl}$)

(1) Linguistic category labels of lexical items

Every lexical item has two components:

- CG lexical category: function (specified for linearization order, which is itself a formal feature) X/Y or X\Y or argument (with agreement features)
  - X, Y variables over categories ($S, NP_{3pl}, S \backslash NP_{3pl}$)
    - walk S\NP

(1) Linguistic category labels of lexical items

Every lexical item has two components:

- CG lexical category: function (specified for linearization order, which is itself a formal feature) X/Y or X\Y or argument (with agreement features)
  - X, Y variables over categories ($S, NP_{3pl}, S\backslash NP_{3pl}$)
    - ⋆ walk S\NP
    - ⋆ show (S\NP)/NP

(1) Linguistic category labels of lexical items

Every lexical item has two components:

- CG lexical category: function (specified for linearization order, which is itself a formal feature) X/Y or X\Y or argument (with agreement features)
  - X, Y variables over categories ($S, NP_{3pl}, S\backslash NP_{3pl}$)
    - walk S\NP
    - show (S\NP)/NP
  - Comparable to lexical categories in Bare Phrase Structure Minimalism:

(1) Linguistic category labels of lexical items

Every lexical item has two components:

- CG lexical category: function (specified for linearization order, which is itself a formal feature) X/Y or X\Y or argument (with agreement features)
  - X, Y variables over categories ($S, NP_{3pl}, S\backslash NP_{3pl}$)
    - walk S\NP
    - show (S\NP)/NP
  - Comparable to lexical categories in Bare Phrase Structure Minimalism:
    - walk [V, *u*N]

(1) Linguistic category labels of lexical items

Every lexical item has two components:

- CG lexical category: function (specified for linearization order, which is itself a formal feature) X/Y or X\Y or argument (with agreement features)
  - X, Y variables over categories ($S, NP_{3pl}, S \backslash NP_{3pl}$)
    - ⋆ walk S\NP
    - ⋆ show (S\NP)/NP
  - Comparable to lexical categories in Bare Phrase Structure Minimalism:
    - ⋆ walk [V, *u*N]
    - ⋆ show [V, *u*N, *u*N]

(1) Linguistic category labels of lexical items

Every lexical item has two components:

- CG lexical category: function (specified for linearization order, which is itself a formal feature) X/Y or X\Y or argument (with agreement features)
  - X, Y variables over categories ($S, NP_{3pl}, S\backslash NP_{3pl}$)
    - walk S\NP
    - show (S\NP)/NP
  - Comparable to lexical categories in Bare Phrase Structure Minimalism:
    - walk [V, *u*N]
    - show [V, *u*N, *u*N]
  - Merge/unification possible in function composition when values on all attributes are equal or one is less specified than the other (merged item has most specific attribute).

(1) Linguistic category labels of lexical items

Every lexical item has two components:

- CG lexical category: function (specified for linearization order, which is itself a formal feature) X/Y or X\Y or argument (with agreement features)
  - X, Y variables over categories ($S, NP_{3pl}, S\backslash NP_{3pl}$)
    - ★ walk S\NP
    - ★ show (S\NP)/NP
  - Comparable to lexical categories in Bare Phrase Structure Minimalism:
    - ★ walk [V, *u*N]
    - ★ show [V, *u*N, *u*N]
  - Merge/unification possible in function composition when values on all attributes are equal or one is less specified than the other (merged item has most specific attribute).
- Logical form: $f\ a$ application of f to a, left association, variable–binding λ operator

(1) Linguistic category labels of lexical items

Every lexical item has two components:

- CG lexical category: function (specified for linearization order, which is itself a formal feature) X/Y or X\Y or argument (with agreement features)
  - X, Y variables over categories ($S, NP_{3pl}, S\backslash NP_{3pl}$)
    - walk S\NP
    - show (S\NP)/NP
  - Comparable to lexical categories in Bare Phrase Structure Minimalism:
    - walk [V, *u*N]
    - show [V, *u*N, *u*N]
  - Merge/unification possible in function composition when values on all attributes are equal or one is less specified than the other (merged item has most specific attribute).
- Logical form: $f\ a$ application of f to a, left association, variable–binding λ operator
  - f, a variables over logical forms corresponding to categories $\lambda y.pres(walk\ y)$

(2) Merge: Function application

Contiguous Merge I: Application Rules

(2) Merge: Function application

Contiguous Merge I: Application Rules

a. Forward application ($>$)
$X/_*Y : f \quad Y : a \Rightarrow X : f \ a$

b. Backward application ($<$)
$Y : a \quad X\backslash_*Y : f \Rightarrow X : f \ a$

(2) Merge: Function application (cont.)

The Principle of Consistency: All rules linearize their inputs consistently with the directionality specified in the governing category.

(2) Merge: Function application (cont.)

The Principle of Consistency: All rules linearize their inputs consistently with the directionality specified in the governing category.
The Principle of Inheritance: Any category that appears in an input that also appears in the output of a rule must be feature-identical in both, including its slash-features, if any.

(2) Merge: Function application (cont.)

The Principle of Consistency: All rules linearize their inputs consistently with the directionality specified in the governing category.

The Principle of Inheritance: Any category that appears in an input that also appears in the output of a rule must be feature-identical in both, including its slash-features, if any.

a. $Y : a \quad X/Y : f \not\Rightarrow X : f \; a$

b. $(X/Y)/W : f \quad Y : a \not\Rightarrow X/W : \lambda w.f \; w \; a$

c. $X_i/Y : f \quad Y : a \not\Rightarrow X_j : f \; a$

Examples of Merge

1.
$$\dfrac{\dfrac{\text{Mary}}{\mathtt{NP}_{3s} : \mathit{mary}} \quad \dfrac{\text{walks.}}{\mathtt{S}\backslash\mathtt{NP}_{3s} : \lambda y.\mathit{pres}(\mathit{walk}\ \ y)}}{\mathtt{S} : \mathit{pres}(\mathit{walk}\ \ \mathit{mary})}<$$

Examples of Merge

1.
$$
\dfrac{\dfrac{\text{Mary}}{\mathtt{NP}_{3s} : \mathit{mary}} \quad \dfrac{\text{walks.}}{\mathtt{S}\backslash\mathtt{NP}_{3s} : \lambda y.\mathit{pres}(\mathit{walk}\ y)}}{\mathtt{S} : \mathit{pres}(\mathit{walk}\ \mathit{mary})}<
$$

2.
$$
\dfrac{\dfrac{\dfrac{\text{A}}{\mathtt{NP}_{3s}/\mathtt{N}_{3s} : a} \quad \dfrac{\text{child}}{\mathtt{N}_{3s} : \mathit{child}}}{\mathtt{NP}_{3s} : \mathit{achild}}> \quad \dfrac{\dfrac{\text{held}}{(\mathtt{S}\backslash\mathtt{NP}_{agr})/\mathtt{NP} : \lambda x \lambda y.\mathit{past}(\mathit{hold})xy} \quad \dfrac{\dfrac{\text{a}}{\mathtt{NP}_{3s}/\mathtt{N}_{3s} : a} \quad \dfrac{\text{balloon.}}{\mathtt{N}_{3s} : \mathit{balloon}}}{\mathtt{NP}_{3s} : \mathit{aballoon}}>}{\mathtt{S}\backslash\mathtt{NP}_{3s} : \lambda y.\mathit{past}(\mathit{hold}(\mathit{aballoon})y)}>}{\mathtt{S} : \mathit{past}(\mathit{hold}(\mathit{aballoon})\mathit{achild})}<
$$

Merge: Function Composition

Contiguous Merge IIa: Function Composition

Merge: Function Composition

Contiguous Merge IIa: Function Composition

a. Forward composition (>**B**)
$X/_{\diamond}Y : f \quad Y/Z : g \Rightarrow X/Z : \lambda z.f(gz)$

b. Backward composition (<**B**)
$Y\backslash Z : g \quad X\backslash_{\diamond}Y : f \Rightarrow X\backslash Z : \lambda z.f(gz)$

Merge: Function Composition

Contiguous Merge IIa: Function Composition

a. Forward composition ($>\mathbf{B}$)
$X/_{\diamond}Y : f \quad Y/Z : g \Rightarrow X/Z : \lambda z.f(gz)$

b. Backward composition ($<\mathbf{B}$)
$Y\backslash Z : g \quad X\backslash_{\diamond}Y : f \Rightarrow X\backslash Z : \lambda z.f(gz)$

c. Forward crossing composition ($>\mathbf{B}_{\times}$)
$X/_{\times}Y : f \quad Y\backslash Z : g \Rightarrow X\backslash Z : \lambda z.f(gz)$

d. Backward crossing composition ($<\mathbf{B}_{\times}$)
$Y/Z : g \quad X\backslash_{\times}Y : f \Rightarrow X/Z : \lambda z.f(gz)$

Merge: Function Composition (cont.)

Consistency and Inheritence apply here too:

Merge: Function Composition (cont.)

Consistency and Inheritence apply here too:

$$X/_{\diamond}Y : f \quad Y/Z : g \not\Rightarrow X\backslash Z : \lambda z.f(gz)$$

Merge: Function Composition (cont.)

Consistency and Inheritence apply here too:

$$X/_{\diamond}Y : f \quad Y/Z : g \not\Rightarrow X\backslash Z : \lambda z.f(gz)$$

Function composition yields constituency effects:

Merge: Function Composition (cont.)

Consistency and Inheritence apply here too:

$$X/_{\diamond}Y : f \quad Y/Z : g \not\Rightarrow X\backslash Z : \lambda z.f(gz)$$

Function composition yields constituency effects:

$$\dfrac{\dfrac{\text{will}}{(\mathtt{S}\backslash\mathtt{NP})/\mathtt{VP}\ \lambda p\lambda y.will(py)} \quad \dfrac{\text{wear}}{\mathtt{VP}/\mathtt{NP} : \lambda x\lambda y.(wear\ xy)}}{(\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP} : \lambda x\lambda y.will(wear\ xy)}{>}\mathsf{B}$$

Merge: Second-level Function Composition

Contiguous Merge IIb: Second-level Function Composition

Merge: Second-level Function Composition

Contiguous Merge IIb: Second-level Function Composition

a. Forward level-2 composition ($>\mathbf{B}^2$)
$X/_{\diamond}Y : f \quad (Y/Z)|W : g \Rightarrow (X/Z)|W : \lambda w \lambda z.f(gwz)$

b. Backward level-2 composition ($<\mathbf{B}^2$)
$(Y\backslash Z)|W : g \quad X\backslash_{\diamond}Y : f \Rightarrow (X\backslash Z)|W : \lambda w \lambda z.f(gwz)$

Merge: Second-level Function Composition

Contiguous Merge IIb: Second-level Function Composition

a. Forward level-2 composition ($>\mathbf{B}^2$)
$X/_\diamond Y : f \quad (Y/Z)|W : g \Rightarrow (X/Z)|W : \lambda w \lambda z.f(gwz)$

b. Backward level-2 composition ($<\mathbf{B}^2$)
$(Y\backslash Z)|W : g \quad X\backslash_\diamond Y : f \Rightarrow (X\backslash Z)|W : \lambda w \lambda z.f(gwz)$

c. Forward crossing level-2 composition ($>\mathbf{B}^2_\times$)
$X/_\times Y : f \quad (Y\backslash Z)|W : g \Rightarrow (X\backslash Z)|W : \lambda w \lambda z.f(gwz)$

d. Backward crossing level-2 composition ($<\mathbf{B}^2_\times$)
$(Y/Z)|W : g \quad X\backslash_\times Y : f \Rightarrow (X/Z)|W : \lambda w \lambda z.f(gwz)$

Merge: Second-level Function Composition

Contiguous Merge IIb: Second-level Function Composition

a. Forward level-2 composition ($>\mathbf{B}^2$)
$X/_{\diamond}Y : f \quad (Y/Z)|W : g \Rightarrow (X/Z)|W : \lambda w \lambda z.f(gwz)$

b. Backward level-2 composition ($<\mathbf{B}^2$)
$(Y\backslash Z)|W : g \quad X\backslash_{\diamond}Y : f \Rightarrow (X\backslash Z)|W : \lambda w \lambda z.f(gwz)$

c. Forward crossing level-2 composition ($>\mathbf{B}^2_{\times}$)
$X/_{\times}Y : f \quad (Y\backslash Z)|W : g \Rightarrow (X\backslash Z)|W : \lambda w \lambda z.f(gwz)$

d. Backward crossing level-2 composition ($<\mathbf{B}^2_{\times}$)
$(Y/Z)|W : g \quad X\backslash_{\times}Y : f \Rightarrow (X/Z)|W : \lambda w \lambda z.f(gwz)$

$$X/Y : f \quad (Y/Z)/W : g \not\Rightarrow (X/Z)\backslash W : \lambda w \lambda z.f(gwz)$$

Merge: Second-level Function Composition (cont.)

$$\dfrac{\dfrac{\text{may}}{(\mathtt{S}\backslash\mathtt{NP})/\mathtt{VP}\ \lambda p\lambda y.may(py)} \quad \dfrac{\text{give}}{(\mathtt{VP}/\mathtt{PP})/\mathtt{NP} : \lambda w\lambda x\lambda y.(give\ xwy)}}{((\mathtt{S}\backslash\mathtt{NP})/\mathtt{PP})/\mathtt{NP} : \lambda w\lambda x\lambda y.may(give\ xwy)}{>}\mathbf{B}^2$$

Merge: Second-level Function Composition (cont.)

$$\dfrac{\dfrac{\text{may}}{(\mathtt{S}\backslash\mathtt{NP})/\mathtt{VP}\ \lambda p \lambda y.may(py)} \quad \dfrac{\text{give}}{(\mathtt{VP}/\mathtt{PP})/\mathtt{NP} : \lambda w \lambda x \lambda y.(give\ xwy)}}{((\mathtt{S}\backslash\mathtt{NP})/\mathtt{PP})/\mathtt{NP} : \lambda w \lambda x \lambda y.may(give\ xwy)} >\mathbf{B}^2$$

- Categories can grow completely unbounded, meaning CCG here is non–context–free.

CPP

The Combinatory Projection Principle (CPP): Combinatory rules apply to contiguous categories (“Adjacency"), must respect the linearization specified in the slash direction for the governing category (“Consistency"), and must project unchanged onto the resulting category any further categorial, selectional, and linearization information specified in either the governing or the dependent category (“Inheritance").

Internal Merge

Internal Merge: Raising

- Long–range dependency, i.e. IM, governed by the same function/argument category composition as local–dependency (EM).

Internal Merge: Raising

- Long–range dependency, i.e. IM, governed by the same function/argument category composition as local–dependency (EM).
- IM is achieved via function composition, just need to extend set of lexical types to include second-order functions which themselves can take functions (semantic type $e \to t$).

Internal Merge: Raising

- Long–range dependency, i.e. IM, governed by the same function/argument category composition as local–dependency (EM).
- IM is achieved via function composition, just need to extend set of lexical types to include second-order functions which themselves can take functions (semantic type $e \to t$).
- Working examples:

Internal Merge: Raising

- Long–range dependency, i.e. IM, governed by the same function/argument category composition as local–dependency (EM).
- IM is achieved via function composition, just need to extend set of lexical types to include second-order functions which themselves can take functions (semantic type $e \to t$).
- Working examples:
  1. **Raising**: John seems to walk.

Internal Merge: Raising

- Long–range dependency, i.e. IM, governed by the same function/argument category composition as local–dependency (EM).
- IM is achieved via function composition, just need to extend set of lexical types to include second-order functions which themselves can take functions (semantic type $e \to t$).
- Working examples:
    1. **Raising**: John seems to walk.
       John$_i$ seems t_i to walk.

Internal Merge: Raising

- Long–range dependency, i.e. IM, governed by the same function/argument category composition as local–dependency (EM).
- IM is achieved via function composition, just need to extend set of lexical types to include second-order functions which themselves can take functions (semantic type $e \to t$).
- Working examples:
  1. **Raising**: John seems to walk.
     John$_i$ seems t_i to walk.
  2. **Control**: John tries to walk.

Internal Merge: Raising

- Long–range dependency, i.e. IM, governed by the same function/argument category composition as local–dependency (EM).
- IM is achieved via function composition, just need to extend set of lexical types to include second-order functions which themselves can take functions (semantic type $e \to t$).
- Working examples:
  1. **Raising**: John seems to walk.
     John$_i$ seems t_i to walk.
  2. **Control**: John tries to walk.
     John$_i$ tries PRO_i to walk.

IM via raising and control

Raising-to-subject

IM via raising and control

Raising-to-subject

seem: $VP/VP_{to} : \lambda p \lambda y.seem(p\ y)$

IM via raising and control

Raising-to-subject

seem: $VP/VP_{to} : \lambda p \lambda y.seem(p\ y)$

$$\dfrac{\dfrac{\dfrac{\text{John}}{\mathtt{NP} : john} \quad \dfrac{\dfrac{\text{seems}}{(\mathtt{S}\backslash\mathtt{NP})/\mathtt{VP}_{\mathrm{to}} : \lambda p \lambda y.pres(seem(py))} \quad \dfrac{\dfrac{\text{to}}{\mathtt{VP}_{\mathrm{to}}/\mathtt{VP} : \lambda p \lambda y.py} \quad \dfrac{\text{walk.}}{\mathtt{VP} : \lambda y.walk(y)}}{\mathtt{VP}_{\mathrm{to}} : \lambda y.walk(y)}>}{\mathrm{S}\backslash\mathtt{NP} : \lambda y.pres(seem(walk y))}>}{}}{\mathrm{S} : pres(seem(walk john))}<$$

IM via raising and control

Raising-to-subject

seem: $VP/VP_{to} : \lambda p \lambda y.seem(p\ y)$

$$
\begin{array}{cccc}
\text{John} & \text{seems} & \text{to} & \text{walk.} \\
\hline
\mathtt{NP} : john & (\mathtt{S}\backslash\mathtt{NP})/\mathtt{VP}_{\mathrm{to}} : \lambda p \lambda y.pres(seem(py)) & \mathtt{VP}_{\mathrm{to}}/\mathtt{VP} : \lambda p \lambda y.py & \mathtt{VP} : \lambda y.walk(y)
\end{array}
$$

$$
\frac{\mathtt{VP}_{\mathrm{to}}/\mathtt{VP} : \lambda p \lambda y.py \quad \mathtt{VP} : \lambda y.walk(y)}{\mathtt{VP}_{\mathrm{to}} : \lambda y.walk(y)}>
$$

$$
\frac{(\mathtt{S}\backslash\mathtt{NP})/\mathtt{VP}_{\mathrm{to}} : \lambda p \lambda y.pres(seem(py)) \quad \mathtt{VP}_{\mathrm{to}} : \lambda y.walk(y)}{\mathtt{S}\backslash\mathtt{NP} : \lambda y.pres(seem(walk y))}>
$$

$$
\frac{\mathtt{NP} : john \quad \mathtt{S}\backslash\mathtt{NP} : \lambda y.pres(seem(walk y))}{\mathtt{S} : pres(seem(walk john))}<
$$

Subject-control

IM via raising and control

Raising-to-subject

seem: $VP/VP_{to} : \lambda p \lambda y.seem(p\ y)$

$$
\dfrac{\dfrac{\text{John}}{\mathtt{NP} : john} \quad \dfrac{\dfrac{\text{seems}}{(\mathtt{S}\backslash\mathtt{NP})/\mathtt{VP}_{\mathrm{to}} : \lambda p \lambda y.pres(seem(py))} \quad \dfrac{\dfrac{\text{to}}{\mathtt{VP}_{\mathrm{to}}/\mathtt{VP} : \lambda p \lambda y.py} \quad \dfrac{\text{walk.}}{\mathtt{VP} : \lambda y.walk(y)}}{\mathtt{VP}_{\mathrm{to}} : \lambda y.walk(y)}>}{\mathsf{S} \backslash \mathtt{NP} : \lambda y.pres(seem(walk y))}>}{\mathsf{S} : pres(seem(walk john))}<
$$

Subject-control

try: $VP_{+a}/VP_{to} : \lambda p \lambda y.try(\ p\ y)y$

IM via raising and control

Raising-to-subject

seem: $VP/VP_{to} : \lambda p \lambda y.seem(p\ y)$

$$
\dfrac{\dfrac{\text{John}}{\mathtt{NP} : john} \quad \dfrac{\dfrac{\text{seems}}{(\mathtt{S}\backslash\mathtt{NP})/\mathtt{VP}_{\mathrm{to}} : \lambda p \lambda y.pres(seem(py))} \quad \dfrac{\dfrac{\text{to}}{\mathtt{VP}_{\mathrm{to}}/\mathtt{VP} : \lambda p \lambda y.py} \quad \dfrac{\text{walk.}}{\mathtt{VP} : \lambda y.walk(y)}}{\mathtt{VP}_{\mathrm{to}} : \lambda y.walk(y)}>}{\mathrm{S}\ \backslash\mathtt{NP}\ : \lambda y.pres(seem(walk y))}>}{\mathrm{S} : pres(seem(walkjohn))}<
$$

Subject-control

try: $VP_{+a}/VP_{to} : \lambda p \lambda y.try(\ p\ y)y$

$$
\dfrac{\dfrac{\text{John}}{\mathtt{NP} : john} \quad \dfrac{\dfrac{\text{tries}}{(\mathtt{S}_{+\mathrm{a}}\backslash\mathtt{NP})/\mathtt{VP}_{\mathrm{to}} : \lambda p \lambda y.pres(try(py)y)} \quad \dfrac{\dfrac{\text{to}}{\mathtt{VP}_{\mathrm{to}}/\mathtt{VP} : \lambda p \lambda y.py} \quad \dfrac{\text{walk.}}{\mathtt{VP} : \lambda y.walk(y)}}{\mathtt{VP}_{\mathrm{to}} : \lambda y.walk(y)}>}{\mathtt{S}_{+\mathrm{a}}\ \backslash\mathtt{NP}\ : \lambda y.pres(try(walk y)y)}>}{\mathtt{S}_{+\mathrm{a}} : pres(try(walkjohn)john)}<
$$

IM via raising and control (cont.)

- These result in composition of the If content with the predicate of their complement.

IM via raising and control (cont.)

- These result in composition of the If content with the predicate of their complement.
- In the raising-to-subject, the variable y has no role outside of predicate p.

IM via raising and control (cont.)

- These result in composition of the If content with the predicate of their complement.
- In the raising-to-subject, the variable y has no role outside of predicate p.
- For subject-control, the variable has two roles, one inside of the predicate p and one outside of it, i.e. once as a subject controller and once as the controllee.

IM via raising and control (cont.)

- These result in composition of the If content with the predicate of their complement.
- In the raising-to-subject, the variable y has no role outside of predicate p.
- For subject-control, the variable has two roles, one inside of the predicate p and one outside of it, i.e. once as a subject controller and once as the controllee.
- Thus, the c-command relationship used for operator binding is here *only* established at If, which is why IM is effectively reduced to EM with additional If machinery.

Type-raising

Universal quantifiers and type-raising

In order for universally quantified NPs to take scope over the verbal domain, they themselves need to bear the category of a functor.

Universal quantifiers and type-raising

In order for universally quantified NPs to take scope over the verbal domain, they themselves need to bear the category of a functor.

$$\dfrac{\dfrac{\text{Everything}}{\mathtt{NP} : \mathit{everything}} \quad \dfrac{\text{flows}}{\mathtt{S\backslash NP} : \lambda y.\mathit{pres}(\mathit{flow}\ \ y)}}{\mathtt{S} : \mathit{pres}\ (\mathit{flow\ everything})}<$$

Universal quantifiers and type-raising

In order for universally quantified NPs to take scope over the verbal domain, they themselves need to bear the category of a functor.

$$\dfrac{\dfrac{\text{Everything}}{\mathtt{NP} : \textit{everything}} \quad \dfrac{\text{flows}}{\mathtt{S\backslash NP} : \lambda y.\textit{pres}(\textit{flow}\ y)}}{\mathtt{S} : \textit{pres (flow everything)}}<$$

$$\dfrac{\dfrac{\text{Everything}}{\mathtt{S/(S\backslash NP)} : \lambda p\forall y[\textit{thing}\ y \Rightarrow py]} \quad \dfrac{\text{flows}}{\mathtt{S\backslash NP} : \lambda y.\textit{pres}(\textit{flow}\ y)}}{\mathtt{S} : \forall y[\textit{thing}\ y \Rightarrow \textit{pres}(\textit{flow}\ y)]}>$$

Type-raising

- This then results in having to allow all NPs and transitive verbs to “type raise”, e.g. objects to take scope over matrix domain.

Type-raising

- This then results in having to allow all NPs and transitive verbs to "type raise", e.g. objects to take scope over matrix domain.

$$
\begin{array}{ccc}
\text{Someone} & \text{sees} & \text{everything} \\
\hline
\mathtt{S}/(\mathtt{S}\backslash\mathtt{NP}) & (\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP} & (\mathtt{S}\backslash\mathtt{NP})\backslash((\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP}) \\
: \lambda p.p(\mathit{some}\ \mathit{person}) & : \lambda x\lambda y.\mathit{pres}(\mathit{see}\ xy) & : \lambda p\lambda y.\forall x[\mathit{thing}\ x \Rightarrow p\ x\ y] \\
\cline{2-3}
 & \multicolumn{2}{c}{\mathtt{S}\backslash\mathtt{NP}\ : \lambda y.\forall x[\mathit{thing}\ x \Rightarrow \mathit{pres}(\mathit{see}\ xy)] \quad <} \\
\hline
\multicolumn{3}{c}{\mathtt{S}\ : \forall x[\mathit{pres}(\mathit{see}\ x(\mathit{some}\ \mathit{person}))] \quad >}
\end{array}
$$

Type-raising (cont.)

Type-raised generalized quantifier is the only way for universal quantifiers to take wide-scope without quantifier-raising.

Type-raising (cont.)

Type-raised generalized quantifier is the only way for universal quantifiers to take wide-scope without quantifier-raising.

$$
\begin{array}{ccccc}
\text{A} & \text{girl} & \text{held} & \text{every} & \text{chipmunk.} \\
\hline
(\mathtt{S}/(\mathtt{S}\backslash\mathtt{NP}_{3s}))/\mathtt{N}_{3s} & \mathtt{N}_{3s} & (\mathtt{S}\backslash\mathtt{NP}_{agr})/\mathtt{NP} & ((\mathtt{S}\backslash\mathtt{NP}_{agr})\backslash((\mathtt{S}\backslash\mathtt{NP}_{agr})/\mathtt{NP}))/\mathtt{N}_{3s} & \mathtt{N}_{3s} \\
: \lambda n \lambda p.p(a\ n) & : \mathit{girl} & : \lambda x \lambda y.\mathit{past}(\mathit{hold}\ xy) & : \lambda n \lambda p \lambda y.\forall x[n\ x \rightarrow p\ x\ y] & : \mathit{chipmunk}
\end{array}
$$

$$\mathtt{S}/(\mathtt{S}\backslash\mathtt{NP}_{3s}) \ : \lambda p.p(a\ \mathit{girl}) \quad (>)$$

$$(\mathtt{S}\backslash\mathtt{NP}_{agr})\backslash((\mathtt{S}\backslash\mathtt{NP}_{agr})/\mathtt{NP}) \ : \lambda p \lambda y.\forall x[\mathit{chipmunk}\ x \rightarrow p\ x\ y)] \quad (>)$$

$$\mathtt{S}\backslash\mathtt{NP}_{agr} \ : \lambda y.\forall x[\mathit{chipmunk}\ x \rightarrow \mathit{past}(\mathit{hold}\ xy)] \quad (<)$$

$$\mathtt{S} \ : \forall x[\mathit{chipmunk}\ x \rightarrow \mathit{past}(\mathit{hold}\ x(a\ \mathit{girl}))] \quad (>)$$

Type-raising for generalized quantifiers

- Type-raising changes nothing about the derivations except reverses the direction of function application.

Type-raising for generalized quantifiers

- Type-raising changes nothing about the derivations except reverses the direction of function application.
- Raised arguments pass their values to the lf by binding a second-order variable p, rather than the verb itself, similar to IM.

Type-raising for generalized quantifiers

- Type-raising changes nothing about the derivations except reverses the direction of function application.
- Raised arguments pass their values to the lf by binding a second-order variable p, rather than the verb itself, similar to IM.
  - someone: $NP^{\uparrow} : \lambda p \lambda \hat{y}.p(some\ person)\hat{y}$
  - an: $NP^{\uparrow}/N : \lambda n \lambda p \lambda \hat{y}.p(a\ n)\hat{y}$
  - every: $NP^{\uparrow}/N : \lambda n \lambda p \lambda \hat{y}.\forall x(n\ x \rightarrow p\ x\ \hat{y})$

Type-raising for generalized quantifiers

- Type-raising changes nothing about the derivations except reverses the direction of function application.
- Raised arguments pass their values to the lf by binding a second-order variable p, rather than the verb itself, similar to IM.
  - someone: $NP^{\uparrow} : \lambda p \lambda \hat{y}.p(some\ person)\hat{y}$
  - an: $NP^{\uparrow}/N : \lambda n \lambda p \lambda \hat{y}.p(a\ n)\hat{y}$
  - every: $NP^{\uparrow}/N : \lambda n \lambda p \lambda \hat{y}.\forall x(n\ x \rightarrow p\ x\ \hat{y})$
- Type-raising allows for scrambling word orders.

Type-raising for generalized quantifiers

- Type-raising changes nothing about the derivations except reverses the direction of function application.
- Raised arguments pass their values to the lf by binding a second-order variable p, rather than the verb itself, similar to IM.
  - someone: $NP^{\uparrow} : \lambda p \lambda \hat{y}.p(some\ person)\hat{y}$
  - an: $NP^{\uparrow}/N : \lambda n \lambda p \lambda \hat{y}.p(a\ n)\hat{y}$
  - every: $NP^{\uparrow}/N : \lambda n \lambda p \lambda \hat{y}.\forall x(n\ x \to p\ x\ \hat{y})$
- Type-raising allows for scrambling word orders.
- Caveat: not a free combinatory rule to be used for syntactic derivations, only available as a morpho-lexical schema.

Type-Raising: Scrambling in Germanic

Scrambling in Germanic

- Scrambling of NP arguments is not freely allowed in most English dialects.

Scrambling in Germanic

- Scrambling of NP arguments is not freely allowed in most English dialects.
- CCGs ensure case-raised $NP^{\uparrow}$s cannot combine by specifying slash-types to exclude crossing composition: $|_{\diamond *}$

Scrambling in Germanic

- Scrambling of NP arguments is not freely allowed in most English dialects.
- CCGs ensure case-raised $NP^{\uparrow}$s cannot combine by specifying slash-types to exclude crossing composition: $|_{\diamond *}$

$$
\begin{array}{cccc}
\text{*Sara} & \text{gave} & \text{cake} & \text{her sister.} \\
\hline
\mathtt{S}/_{\diamond *}(\mathtt{S}\backslash\mathtt{NP}) & ((\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP})/\mathtt{NP} & (\mathtt{S}\backslash\mathtt{NP})\backslash_{\diamond *}((\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP}) & (\mathtt{S}\backslash\mathtt{NP})\backslash_{\diamond *}((\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP}) \\
 & \multicolumn{2}{c}{\overline{\qquad\qquad\qquad\qquad}\, * <\mathrm{B}_{\times}} & \\
 & \multicolumn{2}{c}{***} &
\end{array}
$$

Scrambling in Germanic

- Scrambling of NP arguments is not freely allowed in most English dialects.
- CCGs ensure case-raised $NP^{\uparrow}$s cannot combine by specifying slash-types to exclude crossing composition: $|_{\diamond *}$

$$\begin{array}{cccc}
\text{*Sara} & \text{gave} & \text{cake} & \text{her sister.} \\
\hline
\mathtt{S}/_{\diamond *}(\mathtt{S}\backslash\mathtt{NP}) & ((\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP})/\mathtt{NP} & (\mathtt{S}\backslash\mathtt{NP})\backslash_{\diamond *}((\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP}) & (\mathtt{S}\backslash\mathtt{NP})\backslash_{\diamond *}((\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP}) \\
\cline{2-3}
 & \multicolumn{2}{c}{***} \quad {*}{<}\mathrm{B}_{\times} &
\end{array}$$

- Scrambling involving heavy NP-shift allowed with English VP adjuncts like yesterday:

Scrambling in Germanic

- Scrambling of NP arguments is not freely allowed in most English dialects.
- CCGs ensure case-raised $NP^{\uparrow}$s cannot combine by specifying slash-types to exclude crossing composition: $|_{\diamond *}$

$$
\begin{array}{cccc}
\text{*Sara} & \text{gave} & \text{cake} & \text{her sister.} \\
\hline
\mathtt{S}/_{\diamond *}(\mathtt{S}\backslash\mathtt{NP}) & ((\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP})/\mathtt{NP} & (\mathtt{S}\backslash\mathtt{NP})\backslash_{\diamond *}((\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP}) & (\mathtt{S}\backslash\mathtt{NP})\backslash_{\diamond *}((\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP}) \\
\cline{2-3}
 & \multicolumn{2}{c}{***} \quad {*}{<}\mathrm{B}_{\times} &
\end{array}
$$

- Scrambling involving heavy NP-shift allowed with English VP adjuncts like yesterday:

$$
\begin{array}{cccc}
\text{Sara} & \text{saw} & \text{today} & \text{a really large dog.} \\
\hline
\mathtt{S}/_{\diamond *}(\mathtt{S}\backslash\mathtt{NP}) & (\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP} & (\mathtt{S}\backslash\mathtt{NP})\backslash(\mathtt{S}\backslash\mathtt{NP}) & (\mathtt{S}\backslash\mathtt{NP})\backslash_{\diamond *}((\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP}) \\
\cline{2-3}
 & \multicolumn{2}{c}{(\mathtt{S}\backslash\mathtt{NP})/\mathtt{NP} \quad {<}\mathrm{B}_{\times}} & \\
\cline{2-4}
 & \multicolumn{3}{c}{\mathtt{S}\backslash\mathtt{NP} \quad {<}} \\
\hline
\multicolumn{4}{c}{\mathtt{S} \quad {>}}
\end{array}
$$

German scrambling

Type-raising of arguments (morpho-lexical and order-preserving) can interact with function composition to derive scrambling effects in languages with freer word-order.

(1) *Johann hat [auf dem Markt] [die Lebensmittel gekauft].*
Johann has at the market the groceries bought.

'Johann bought the groceries at the market.'

(2) *Johann hat [die Lebensmittel*] *[auf dem Markt] gekauft.*
Johann has the groceries at the market bought.

'Johann bought the groceries at the market.'

(3) **Johann [die Lebensmittel*] *hat [auf dem Markt] gekauft.*
Johann the groceries has at the market bought.

'Johann bought the groceries at the market.'

CCG and German scrambling

Johann	hat	die Lebensmittel	auf dem Markt	gekauft.
Johann	has	the groceries	at the market	bought.
$S/(S\backslash NP)$	$(S_{main}\backslash NP_{nom})/VP$	$VP/(VP\backslash NP)$	VP/VP	$VP\backslash NP_{acc}$

$$VP/VP \quad VP\backslash NP_{acc} \;\xrightarrow{>B_\times}\; VP\backslash NP_{acc}$$

$$VP/(VP\backslash NP) \quad VP\backslash NP_{acc} \;\xrightarrow{>}\; VP$$

$$(S_{main}\backslash NP_{nom})/VP \quad VP \;\xrightarrow{>}\; S\backslash NP_{nom}$$

$$S/(S\backslash NP) \quad S\backslash NP_{nom} \;\xrightarrow{>}\; S$$

CCG and German scrambling (cont.)

$$
\dfrac{\text{*Johann}}{\text{Johann}}\quad \dfrac{\text{die Lebensmittel}}{\text{the groceries}}\quad \dfrac{\text{hat}}{\text{has}}\quad \dfrac{\text{auf dem Markt}}{\text{at the market}}\quad \dfrac{\text{gekauft.}}{\text{bought.}}
$$

$$
\texttt{S}/(\texttt{S}\backslash\texttt{NP}) \quad \texttt{VP}/(\texttt{VP}\backslash\texttt{NP}_{\mathrm{acc}}) \quad (\texttt{S}_{\mathrm{main}}\backslash\texttt{NP}_{\mathrm{nom}})/\texttt{VP} \quad \texttt{VP}/\texttt{VP} \quad \texttt{VP}\backslash\texttt{NP}_{\mathrm{acc}}
$$

$$
\texttt{VP}/\texttt{VP} \quad \texttt{VP}\backslash\texttt{NP}_{\mathrm{acc}} \xrightarrow{>\mathsf{B}_\times} \texttt{VP}\backslash\texttt{NP}_{\mathrm{acc}}
$$

$$
(\texttt{S}_{\mathrm{main}}\backslash\texttt{NP}_{\mathrm{nom}})/\texttt{VP} \quad \texttt{VP}\backslash\texttt{NP}_{\mathrm{acc}} \xrightarrow{>\mathsf{B}_\times} (\texttt{S}_{\mathrm{main}}\backslash\texttt{NP}_{\mathrm{nom}})\backslash\texttt{NP}_{\mathrm{acc}}
$$

$$
\xrightarrow{*} \; ***
$$

IM: Wh-Movement

Wh-movement

- Unbounded wh-movement is analyzed as merger by application of second-order function to the residue left over after the wh-word has moved out.

Wh-movement

- Unbounded wh-movement is analyzed as merger by application of second-order function to the residue left over after the wh-word has moved out.
- Fronted elements have to have a non-order preserving raised type.

Wh-movement

- Unbounded wh-movement is analyzed as merger by application of second-order function to the residue left over after the wh-word has moved out.
- Fronted elements have to have a non-order preserving raised type.
- So, we have both type-raising and If alteration.

Wh-movement

- Unbounded wh-movement is analyzed as merger by application of second-order function to the residue left over after the wh-word has moved out.
- Fronted elements have to have a non-order preserving raised type.
- So, we have both type-raising and If alteration.

Topicalization:

Wh-movement

- Unbounded wh-movement is analyzed as merger by application of second-order function to the residue left over after the wh-word has moved out.
- Fronted elements have to have a non-order preserving raised type.
- So, we have both type-raising and lf alteration.

Topicalization:

$$
\dfrac{\dfrac{\text{Books,}}{\mathtt{S_t/(S/NP)} : \lambda p.p\ \mathit{books}} \quad \dfrac{\dfrac{\text{she}}{\mathtt{S/(S\backslash NP_{3s})} : \lambda p.p\ \mathit{her}} \quad \dfrac{\text{hates.}}{\mathtt{(S\backslash NP_{3s})/NP} : \lambda x \lambda y.\mathit{pres}(\mathit{hate}\ x\ y)}}{\mathtt{S/NP} : \lambda x.\mathit{pres}(\mathit{hate}\ x\ \mathit{her})}{>}\mathbf{B}}{\mathtt{S_t} : \mathit{pres}(\mathit{hate\ books\ her})}{>}
$$

Wh-question formation

- Auxiliaries (in English) specify the subject-inversion, not the subject.

Wh-question formation

- Auxiliaries (in English) specify the subject-inversion, not the subject.

What	does	she	hate?
$\mathtt{S}_{\mathrm{whq}}/(\mathtt{S}_{\mathrm{inv}}/\mathtt{NP})$	$(\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})/\mathtt{NP}_{3s}$	$(\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})\backslash((\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})/\mathtt{NP}_{3s})$	$\mathtt{VP}/\mathtt{NP}$
$: \lambda p \lambda wh.p\ wh$	$: \lambda y \lambda p.pres(p\ y)$	$: \lambda p.p\ her$	$: \lambda x \lambda y.hate\ x\ y$

$$\overline{\quad\quad\quad\quad\quad\quad\quad\quad}^{<}$$
$$\mathtt{S}_{\mathrm{inv}}/\mathtt{VP} : \lambda p.pres(p\ her)$$
$$\overline{\quad\quad\quad\quad\quad\quad\quad\quad}^{>\mathrm{B}}$$
$$\mathtt{S}_{\mathrm{inv}}/\mathtt{NP} : \lambda x.pres(hate\ x\ her)$$
$$\overline{\quad\quad\quad\quad\quad\quad\quad\quad}^{>}$$
$$\mathtt{S}_{\mathrm{whq}} : \lambda wh.pres(hate\ wh\ her)$$

Wh-question formation

- Auxiliaries (in English) specify the subject-inversion, not the subject.

$$
\begin{array}{cccc}
\text{What} & \text{does} & \text{she} & \text{hate?} \\
\hline
\mathtt{S}_{\mathrm{whq}}/(\mathtt{S}_{\mathrm{inv}}/\mathtt{NP}) & (\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})/\mathtt{NP}_{3s} & (\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})\backslash((\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})/\mathtt{NP}_{3s}) & \mathtt{VP}/\mathtt{NP} \\
:\lambda p \lambda wh.p\ wh & :\lambda y \lambda p.pres(p\ y) & :\lambda p.p\ her & :\lambda x \lambda y.hate\ x\ y
\end{array}
$$

$$\dfrac{(\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})/\mathtt{NP}_{3s} \quad (\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})\backslash((\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})/\mathtt{NP}_{3s})}{\mathtt{S}_{\mathrm{inv}}/\mathtt{VP} : \lambda p.pres(p\ her)}<$$

$$\dfrac{\mathtt{S}_{\mathrm{inv}}/\mathtt{VP} \quad \mathtt{VP}/\mathtt{NP}}{\mathtt{S}_{\mathrm{inv}}/\mathtt{NP} : \lambda x.pres(hate\ x\ her)}>\mathrm{B}$$

$$\dfrac{\mathtt{S}_{\mathrm{whq}}/(\mathtt{S}_{\mathrm{inv}}/\mathtt{NP}) \quad \mathtt{S}_{\mathrm{inv}}/\mathtt{NP}}{\mathtt{S}_{\mathrm{whq}} : \lambda wh.pres(hate\ wh\ her)}>$$

- Wh-item has copies of what it has moved out of, p, and λ-binder.

Wh-question formation

- Auxiliaries (in English) specify the subject-inversion, not the subject.

$$
\begin{array}{cccc}
\text{What} & \text{does} & \text{she} & \text{hate?} \\
\hline
\mathtt{S}_{\mathrm{whq}}/(\mathtt{S}_{\mathrm{inv}}/\mathtt{NP}) & (\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})/\mathtt{NP}_{3s} & (\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})\backslash((\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})/\mathtt{NP}_{3s}) & \mathtt{VP}/\mathtt{NP} \\
: \lambda p \lambda wh.p\ wh & : \lambda y \lambda p.pres(p\ y) & : \lambda p.p\ her & : \lambda x \lambda y.hate\ x\ y
\end{array}
$$

$$\underline{\qquad\qquad\qquad\qquad}{<}$$
$$\mathtt{S}_{\mathrm{inv}}/\mathtt{VP} : \lambda p.pres(p\ her)$$
$$\underline{\qquad\qquad\qquad\qquad}{>\mathrm{B}}$$
$$\mathtt{S}_{\mathrm{inv}}/\mathtt{NP} : \lambda x.pres(hate\ x\ her)$$
$$\underline{\qquad\qquad\qquad\qquad}{>}$$
$$\mathtt{S}_{\mathrm{whq}} : \lambda wh.pres(hate\ wh\ her)$$

- Wh-item has copies of what it has moved out of, p, and λ-binder.
- Its syntactic category is specified as at the left edge of the sentence.

Wh-question formation

- Auxiliaries (in English) specify the subject-inversion, not the subject.

$$
\begin{array}{cccc}
\text{What} & \text{does} & \text{she} & \text{hate?} \\
\hline
\mathtt{S}_{\mathrm{whq}}/(\mathtt{S}_{\mathrm{inv}}/\mathtt{NP}) & (\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})/\mathtt{NP}_{3s} & (\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})\backslash((\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})/\mathtt{NP}_{3s}) & \mathtt{VP}/\mathtt{NP} \\
: \lambda p \lambda wh.p\ wh & : \lambda y \lambda p.pres(p\ y) & : \lambda p.p\ her & : \lambda x \lambda y.hate\ x\ y
\end{array}
$$

$$\dfrac{(\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})/\mathtt{NP}_{3s} \quad (\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})\backslash((\mathtt{S}_{\mathrm{inv}}/\mathtt{VP})/\mathtt{NP}_{3s})}{\mathtt{S}_{\mathrm{inv}}/\mathtt{VP} : \lambda p.pres(p\ her)} <$$

$$\dfrac{\mathtt{S}_{\mathrm{inv}}/\mathtt{VP} \quad \mathtt{VP}/\mathtt{NP}}{\mathtt{S}_{\mathrm{inv}}/\mathtt{NP} : \lambda x.pres(hate\ x\ her)} >\mathrm{B}$$

$$\dfrac{\mathtt{S}_{\mathrm{whq}}/(\mathtt{S}_{\mathrm{inv}}/\mathtt{NP}) \quad \mathtt{S}_{\mathrm{inv}}/\mathtt{NP}}{\mathtt{S}_{\mathrm{whq}} : \lambda wh.pres(hate\ wh\ her)} >$$

- Wh-item has copies of what it has moved out of, p, and λ-binder.
- Its syntactic category is specified as at the left edge of the sentence.
- Its argument is thus passed into the wh-item as p.

Conclusion

- Steedman shows all different types of surface syntactic discontinuity are able to be accounted for in CCG in terms of contiguous merger of local domains that itself is strictly local.

Conclusion

- Steedman shows all different types of surface syntactic discontinuity are able to be accounted for in CCG in terms of contiguous merger of local domains that itself is strictly local.
- λ-binding in lexical logical forms creates potentially unbounded dependency domains without requirement of cyclic mediation.

Conclusion

- Steedman shows all different types of surface syntactic discontinuity are able to be accounted for in CCG in terms of contiguous merger of local domains that itself is strictly local.
- λ-binding in lexical logical forms creates potentially unbounded dependency domains without requirement of cyclic mediation.
- IM is thus reduced at the level of *syntactic* derivation to EM, which reduces Move to purely local contiguous Merge. This is in comparison to Minimalist systems which are not entirely synchronous, as opposed to CCG here.

Conclusion

- Steedman shows all different types of surface syntactic discontinuity are able to be accounted for in CCG in terms of contiguous merger of local domains that itself is strictly local.
- λ-binding in lexical logical forms creates potentially unbounded dependency domains without requirement of cyclic mediation.
- IM is thus reduced at the level of *syntactic* derivation to EM, which reduces Move to purely local contiguous Merge. This is in comparison to Minimalist systems which are not entirely synchronous, as opposed to CCG here.
- Probes, goals, valuations, feature deletion, and visibility conditions can all be eliminated.

References

Chomsky, Noam. 1995. *The Minimalist Program*. Cambridge, MA: MIT Press.

Steedman, Mark. 2023. On Internal Merge. *Linguistic Inquiry*; doi: https://doi.org/10.1162/ling_a_00521.