CS 498 Expressive Grammars for Natural Language Processing: Theory and Applications

Lecture 15

Julia Hockenmaier
juliahmr@cs.uiuc.edu

Where we are

- **Formalisms we have covered so far:**
  Context-free grammars,
  Dependency Grammars,
  Tree-Adjoining Grammars

- **Today: Combinatory Categorial Grammar (CCG)**
  Categories, derivations, non-local dependencies

- **The next lectures:**
  Extracting CCGs from treebanks, building a CCG parser,
  showing the (weak) equivalence of TAG and CCG

Why categorial grammar?

- Phrase-structure grammar stipulates arbitrary categories and rules.
- Can we define a **calculus over syntactic categories** that
  - does not rely on arbitrary categories or rewrite rules?
  - describes how the meaning of sentences is built compositionally from the meaning of words?

Why CCG?

- **CCG is mildly context-sensitive** (like TAG)

  Captures crossing dependencies, but is still efficiently parseable

- **CCG has a flexible constituent structure:**
  - Simple, unified treatment of extraction and coordination
  - Psycholinguistic motivation: allows incremental processing

- **CCG has a transparent syntax-semantics interface**

  If we know the syntax of a sentence, we also know its meaning.

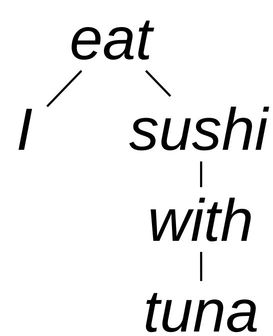
- **CCG requires no traces or null elements**

  Parsing algorithms (e.g. CKY) are straightforward to apply.

What is ’the meaning’ of a sentence?

I eat sushi with tuna

- A dependency structure:



- A logical formula:
  $eat(x,y) \wedge I(x) \wedge sushi(y) \wedge with(y,z) \wedge tuna(z)$

Overview

- **Part I: The basics**
  (local dependencies)
  - The ingredients: categories, rules, derivations
  - Simple syntactic phenomena: modification, simple coordination
  - Syntactic derivations and semantic interpretations
- **Part II: The interesting stuff**
  (long-range dependencies)
  - Bounded dependencies: control and raising
  - Unbounded dependencies: extraction and coordination
  - Scrambling

Part I: The basics

CCG: The machinery

- **Categories** specify subcat lists of words/constituents.
- **Combinatory rules** specify how constituents can combine.
- **The lexicon** specifies which categories a word can have.
- **Derivations** spell out process of combining constituents.

CCG categories

- **Simple categories**: **NP**, **S**, **PP**.
- **Complex categories**: **S\NP**, **(S\NP)/NP**,**(NP\NP)/NP**
  Functions which return a **result** if they get an **argument**:

NP	**S\NP**	$\Longrightarrow$	**S**
He	drinks coffee		He drinks coffee
(S\NP)/NP	**NP**	$\Longrightarrow$	**S\NP**
drinks	coffee		drinks coffee
(NP\NP)/NP	**NP**	$\Longrightarrow$	**NP\NP**
with	milk		with milk

CCG rules

Function application:	**(S\NP)/NP NP**	$\Rightarrow$	**S\NP**
	NP S\NP	$\Rightarrow$	**S**
Type raising:	**NP**	$\Rightarrow$	**S/(S\NP)**
Function composition:	**S/(S\NP) (S\NP)/NP**	$\Rightarrow$	**S/NP**
Coordination:	**NP conj NP**	$\Rightarrow$	**NP**

CCG rules

These are really rule schemas, ie.:

Function application:	$X/Y \;\; Y$	$\Rightarrow$	X
	$Y \;\; X\backslash Y$	$\Rightarrow$	X
Type raising:	X	$\Rightarrow$	$T/(T\backslash X)$
	X	$\Rightarrow$	$T\backslash(T/X)$
Function composition:	$X/Y \;\; Y/Z$	$\Rightarrow$	X/Z
	$Y\backslash Z \;\; X\backslash Y$	$\Rightarrow$	$X\backslash Z$
Coordination:	X **conj** X	$\Rightarrow$	X

CCG rules

These are really rule schemas, ie.:

Function application:	$X/Y \;\; Y$	$\Rightarrow$	X
	$Y \;\; X\backslash Y$	$\Rightarrow$	X
Type raising:	X	$\Rightarrow$	$T/(T\backslash X)$
	X	$\Rightarrow$	$T\backslash(T/X)$
Function composition:	$X/Y \;\; Y/Z$	$\Rightarrow$	X/Z
	$Y\backslash Z \;\; X\backslash Y$	$\Rightarrow$	$X\backslash Z$
Coordination:	X conj X	$\Rightarrow$	X

These are order-preserving rules (context-free only).

More later...

CCG derivations

$$\frac{\textit{I}}{\textbf{NP}} \quad \frac{\textit{like}}{\textbf{(S}\backslash\textbf{NP)/NP}} \quad \frac{\textit{coffee}}{\textbf{NP}}$$

CCG derivations

$$\begin{array}{ccc} \textit{I} & \textit{like} & \textit{coffee} \\ \hline \mathbf{NP} & \mathbf{(S\backslash NP)/NP} & \mathbf{NP} \\ \cline{2-3} & \multicolumn{2}{c}{\mathbf{S\backslash NP} \quad >} \end{array}$$

CCG derivations

$$
\begin{array}{ccc}
\textit{I} & \textit{like} & \textit{coffee} \\
\hline
\mathbf{NP} & \mathbf{(S\backslash NP)/NP} & \mathbf{NP} \\
\cline{2-3}
 & \multicolumn{2}{c}{\mathbf{S\backslash NP}} \quad > \\
\hline
\multicolumn{3}{c}{\mathbf{S}} \quad <
\end{array}
$$

CCG derivations

$$\frac{\textit{I}}{\textbf{NP}} \quad \frac{\textit{like}}{\textbf{(S}\backslash\textbf{NP)/NP}} \quad \frac{\textit{coffee}}{\textbf{NP}}$$

CCG derivations

$$
\begin{array}{ccc}
\dfrac{\textit{I}}{\mathbf{NP}} & \dfrac{\textit{like}}{\mathbf{(S\backslash NP)/NP}} & \dfrac{\textit{coffee}}{\mathbf{NP}} \\
\overline{\mathbf{S/(S\backslash NP)}}^{\,>\mathbf{T}} & &
\end{array}
$$

CCG derivations

$$
\begin{array}{ccc}
\textit{I} & \textit{like} & \textit{coffee} \\
\hline
\mathbf{NP} & \mathbf{(S\backslash NP)/NP} & \mathbf{NP} \\
\hline \; >\mathbf{T} & & \\
\mathbf{S/(S\backslash NP)} & & \\
\hline \multicolumn{2}{c}{>\mathbf{B}} & \\
\multicolumn{2}{c}{\mathbf{S/NP}} &
\end{array}
$$

CCG derivations

I | *like* | *coffee*

NP | (S\NP)/NP | NP

$\overline{}{>}\mathbf{T}$

S/(S\NP)

$\overline{}{>}\mathbf{B}$

S/NP

$\overline{}{>}$

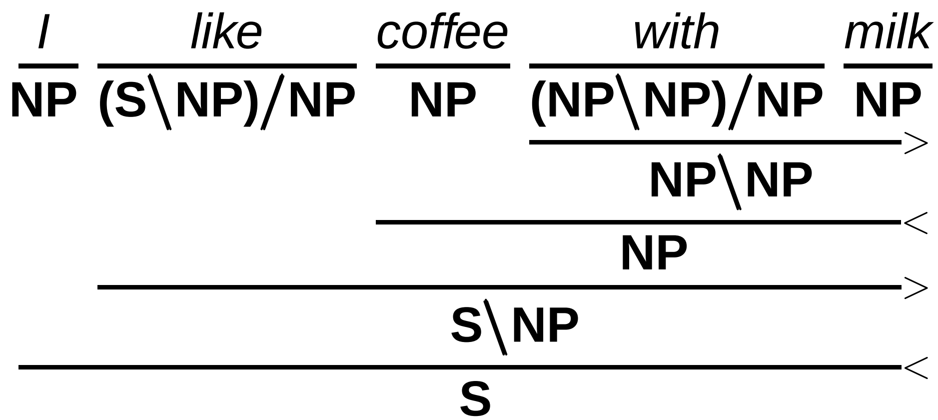
S

CCG derivations

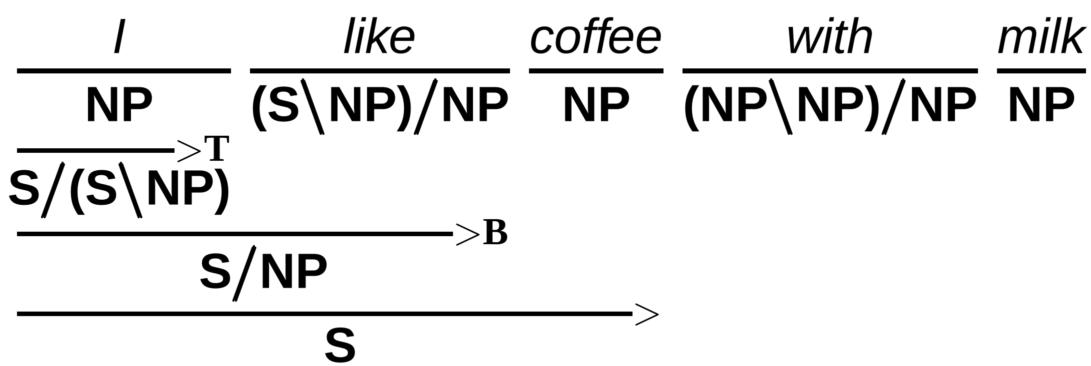
$$
\begin{array}{ccc}
\textit{I} & \textit{like} & \textit{coffee} \\
\hline
\textbf{NP} & \textbf{(S\backslash NP)/NP} & \textbf{NP} \\
\hline \text{(>T)} & & \\
\textbf{S/(S\backslash NP)} & & \\
\hline \text{(>B)} & & \\
\textbf{S/NP} & & \\
\hline \text{(>)} & & \\
\textbf{S} & &
\end{array}
$$

- Type-raising and composition permit alternative derivations.
  This is an example of incremental derivation.
- Here, ***I like*** is a constituent.
  (If you don’t like that, we’ll later see examples where that makes a lot more sense.)
- But in general, not every substring can be a constituent.

CCG derivations



but not a fully incremental derivation:



The syntax-semantics interface

Every syntactic rule has a **semantic interpretation**:

Function application	**X/Y**:$\lambda x.f(x)$	**Y**:a	$\Rightarrow$**X**:$f(a)$
Function composition	**X/Y**:$\lambda x.f(x)$	**X/Y**:$\lambda x.g(x)$	$\Rightarrow$**X/Z**:$\lambda x.f(gx)$
Type-raising	**X**:a		$\Rightarrow$ **T/(T\X)**:$\lambda f.f(a)$

I — *like* — *coffee*

$\mathbf{NP}{:}I' \quad \mathbf{(S\backslash NP)/NP}{:}\lambda x.\lambda y.like'xy \quad \mathbf{NP}{:}coffee'$

$\mathbf{S\backslash NP}{:}\lambda y.like'coffee'y \quad >$

$\mathbf{S}{:}like'coffee'I' \quad <$

The CCG lexicon

- This requires a **lexicon** which pairs words with their syntactic categories and semantic interpretations, e.g:
  eat:**(S$\backslash$NP)/NP**: $\lambda x.\lambda y.$*eat'(x,y)*, **S$\backslash$NP**:$\lambda x.eat'(x)$
  sushi: **NP:***sushi'*
  ...

- CCG is a **lexicalized** formalism:
  The lexicon is where all the language-specific information is represented.

Approximating predicate-argument structure with word-word dependencies

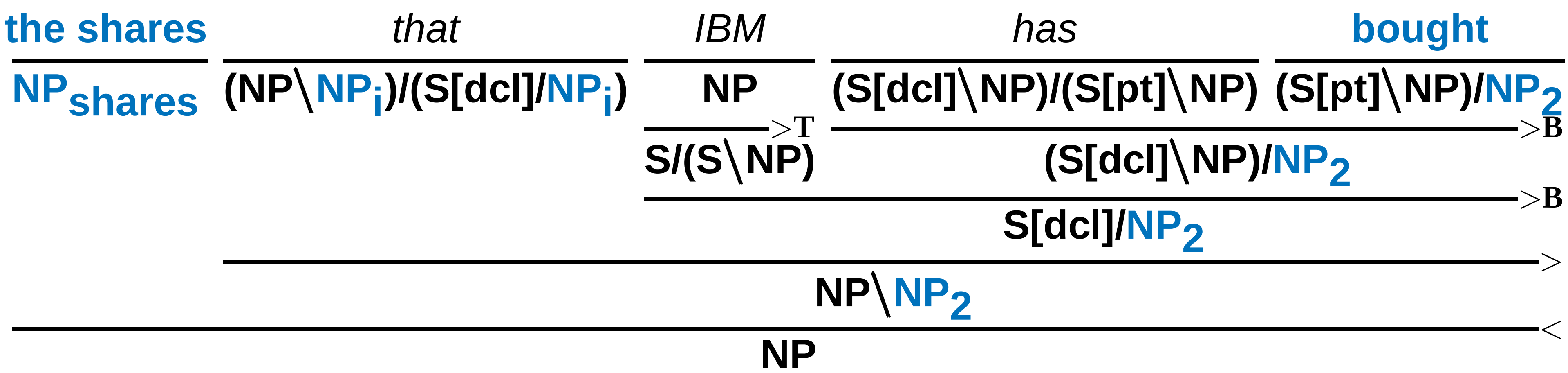
- The **argument slots** of functor categories define dependencies:

$$\cfrac{\cfrac{\textit{like}}{(\mathbf{S[dcl]}\backslash \mathbf{NP}_1)/\mathbf{NP}_2} \qquad \cfrac{\textit{coffee}}{\mathbf{NP}}}{\mathbf{S[dcl]}\backslash \mathbf{NP}_1}\,>$$

$$\langle \textbf{\textit{like}}, (\mathbf{S[dcl]}\backslash \mathbf{NP}_1)/\mathbf{NP}_2, \mathbf{2}, \textbf{\textit{coffee}} \rangle$$

- This also includes **long-range dependencies**.

Long-range dependencies in CCG



$\langle$***bought***, **(S[pt]\NP$_1$)/NP$_2$, 2,** ***shares***$\rangle$

- Long-range dependencies are **local**.
- They are **projected from the lexicon** by the derivation.

Derivations and interpretations

- **Syntactic derivations**...

  ... describe **constituency**

  ... account for **unbounded dependencies**
  that arise through extraction and coordination
  (but don't require traces to do so)

  ... are **not a level of representation** in the theory,
  just a record of the process which builds the interpretation
  (the interface between spoken form and its meaning)

- **Semantic interpretations**...

  ... account for **bounded dependencies**
  that arise in binding, raising and control
  (c-command is defined here)

Part II: The interesting parts

A. Bounded dependencies

Control and raising

- Bounded dependencies:
  co-index arguments within the lexical category of the verb
  (and re-use the corresponding variable in the semantic interpretation)

tries	$((S[dcl]\backslash NP_i)/(S[to]\backslash NP_i))$:	$\lambda p.\lambda y.try'(p(ana'y)\,y)$
persuades	$((S[dcl]\backslash NP)/(S[to]\backslash NP_i))/NP_i$:	$\lambda x.\lambda p.\lambda y.persuade'(p(ana'x)\,x\,y)$

- Modals and auxiliaries are like subject control:

might	$((S[dcl]\backslash NP_i)/(S[b]\backslash NP_i))$:	$\lambda p.\lambda y.might'(p(ana'y)\,y)$

B. Unbounded dependencies

Wh-extraction

- Use type-raising and composition to form “incomplete” constituents.
- Wh-words subcategorize for “incomplete” constituents
  (and use co-indexation to pass the dependencies)

Wh-extraction

- Use type-raising and composition to form "incomplete" constituents.
- Wh-words subcategorize for "incomplete" constituents
  (and use co-indexation to pass the dependencies)

that	***John***	***buys***
$\mathbf{(N\backslash N_i)/(S[dcl]/NP_i)}$	$\mathbf{NP}$	$\mathbf{(S[dcl]\backslash NP)/NP}$

Wh-extraction

- Use type-raising and composition to form “incomplete” constituents.
- Wh-words subcategorize for “incomplete” constituents
  (and use co-indexation to pass the dependencies)

that	***John***	***buys***
$(N\backslash N_i)/(S[dcl]/NP_i)$	NP	$(S[dcl]\backslash NP)/NP$
	$\overline{}{>}T$ $S/(S\backslash NP)$	

Wh-extraction

- Use type-raising and composition to form "incomplete" constituents.
- Wh-words subcategorize for "incomplete" constituents
  (and use co-indexation to pass the dependencies)

$$
\begin{array}{ccc}
\textit{that} & \textit{John} & \textit{buys} \\
\hline
(N\backslash N_i)/(S[dcl]/NP_i) & NP & (S[dcl]\backslash NP)/NP \\
 & \overline{S/(S\backslash NP)}^{\,>T} & \\
 & \multicolumn{2}{c}{\overline{\qquad S[dcl]/NP \qquad}^{\,>B}}
\end{array}
$$

Wh-extraction

- Use type-raising and composition to form "incomplete" constituents.
- Wh-words subcategorize for "incomplete" constituents
  (and use co-indexation to pass the dependencies)

$$
\begin{array}{ccc}
\textit{that} & \textit{John} & \textit{buys} \\
\hline
\mathbf{(N\backslash N_i)/(S[dcl]/NP_i)} & \mathbf{NP} & \mathbf{(S[dcl]\backslash NP)/NP} \\
 & \overline{\mathbf{S/(S\backslash NP)}}^{>T} & \\
 & \multicolumn{2}{c}{\overline{\mathbf{S[dcl]/NP}}^{>B}} \\
\multicolumn{3}{l}{\overline{\mathbf{N\backslash N}}^{>}}
\end{array}
$$

Questions

Does	*he*	*seem*	*to*	*like*	*coffee?*
$(S[q]/(S[b]\backslash NP_i))/NP_i$	NP	(S[b]\NP)/(S[to]\NP)	(S[to]\NP)/(S[b]\NP)	(S[b]\NP)/NP	NP

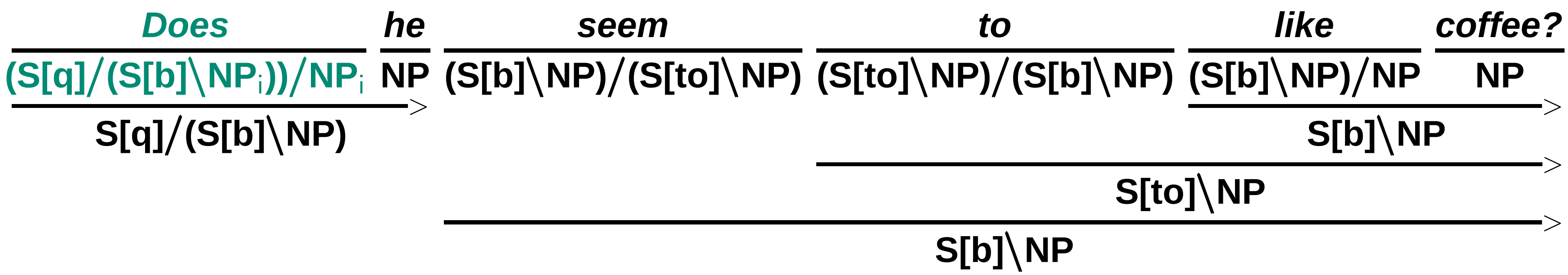
Questions

Does	*he*	*seem*	*to*	*like*	*coffee?*
(S[q]/(S[b]\NP$_i$))/NP$_i$	**NP**	**(S[b]\NP)/(S[to]\NP)**	**(S[to]\NP)/(S[b]\NP)**	**(S[b]\NP)/NP**	**NP**
S[q]/(S[b]\NP) >				**S[b]\NP** >	

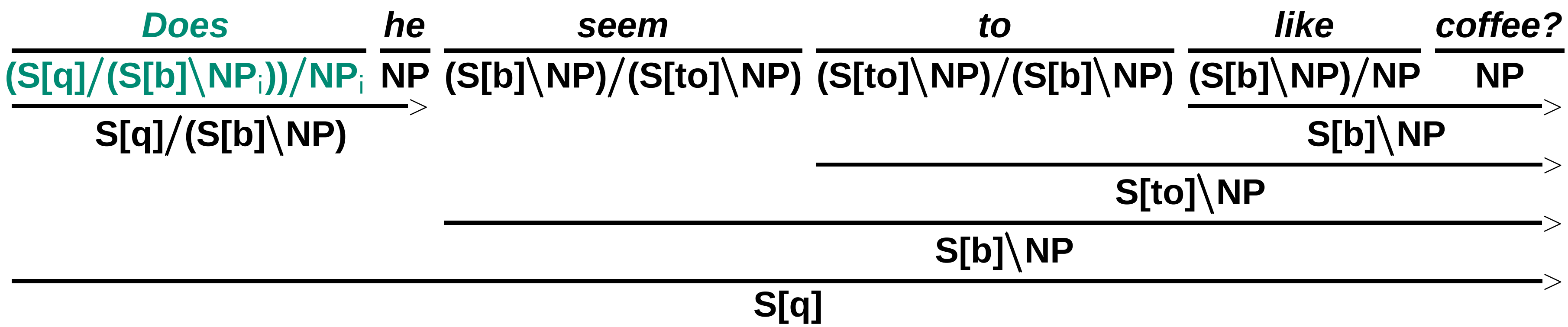
Questions

Does	*he*	*seem*	*to*	*like*	*coffee?*
$(S[q]/(S[b]\backslash NP_i))/NP_i$	NP	(S[b]\NP)/(S[to]\NP)	(S[to]\NP)/(S[b]\NP)	(S[b]\NP)/NP	NP
S[q]/(S[b]\NP) (>)				S[b]\NP (>)	
			S[to]\NP (>)		

Questions


Does
he
seem
to
like
coffee?
(S[q]/(S[b]\NPi))/NPi
NP
(S[b]\NP)/(S[to]\NP)
(S[to]\NP)/(S[b]\NP)
(S[b]\NP)/NP
NP
>
S[q]/(S[b]\NP)
>
S[b]\NP
>
S[to]\NP
>
S[b]\NP

Questions


Does he seem to like coffee?
(S[q]/(S[b]\NP_i))/NP_i
NP
(S[b]\NP)/(S[to]\NP)
(S[to]\NP)/(S[b]\NP)
(S[b]\NP)/NP
NP
S[q]/(S[b]\NP)
S[b]\NP
S[to]\NP
S[b]\NP
S[q]

Questions

Does	*he*	*seem*	*to*	*like*	*coffee?*
(S[q]/(S[b]\NP$_i$))/NP$_i$	NP	(S[b]\NP)/(S[to]\NP)	(S[to]\NP)/(S[b]\NP)	(S[b]\NP)/NP	NP

Does he: S[q]/(S[b]\NP) (>)

like coffee?: S[b]\NP (>)

to like coffee?: S[to]\NP (>)

seem to like coffee?: S[b]\NP (>)

Does he seem to like coffee?: S[q] (>)

What	*does*	*he*	*seem*	*to*	*like?*
	(S[q]/(S[b]\NP$_i$))/NP$_i$	NP	(S[b]\NP)/(S[to]\NP)	(S[to]\NP)/(S[b]\NP)	(S[b]\NP)/NP

Questions

Does	*he*	*seem*	*to*	*like*	*coffee?*
$(S[q]/(S[b]\backslash NP_i))/NP_i$	NP	$(S[b]\backslash NP)/(S[to]\backslash NP)$	$(S[to]\backslash NP)/(S[b]\backslash NP)$	$(S[b]\backslash NP)/NP$	NP

Does he → (>) $S[q]/(S[b]\backslash NP)$

like coffee? → (>) $S[b]\backslash NP$

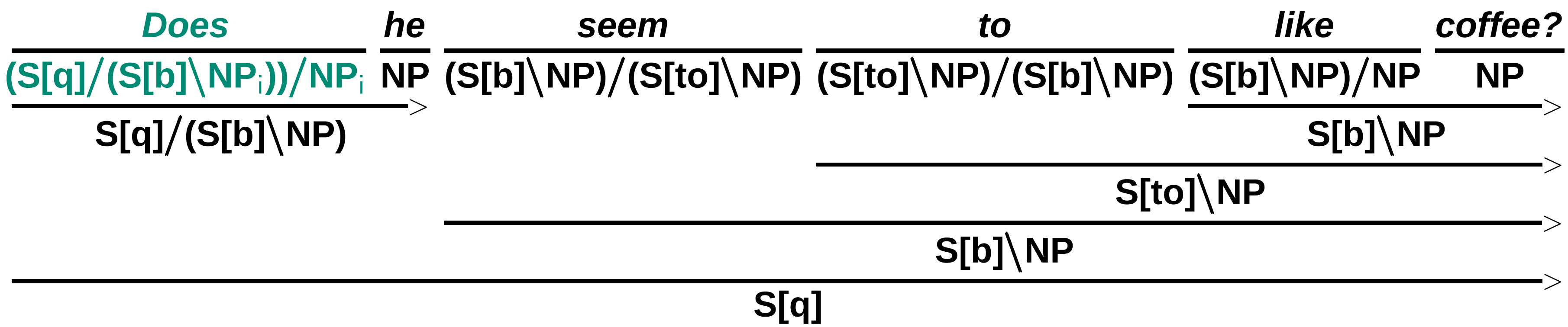
to like coffee? → (>) $S[to]\backslash NP$

seem to like coffee? → (>) $S[b]\backslash NP$

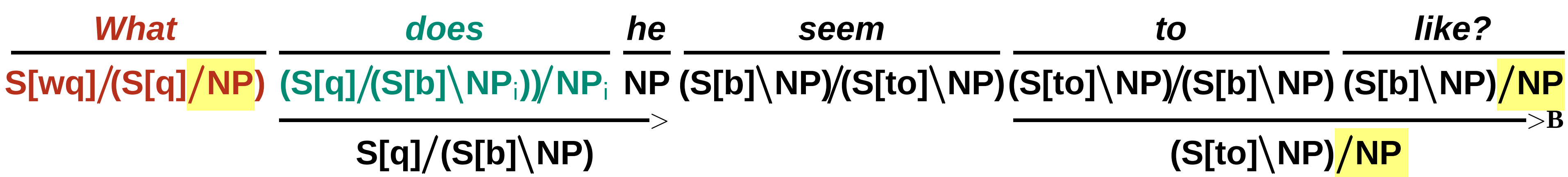
Does he seem to like coffee? → (>) $S[q]$

What	*does*	*he*	*seem*	*to*	*like?*
$S[wq]/(S[q]/NP)$	$(S[q]/(S[b]\backslash NP_i))/NP_i$	NP	$(S[b]\backslash NP)/(S[to]\backslash NP)$	$(S[to]\backslash NP)/(S[b]\backslash NP)$	$(S[b]\backslash NP)/NP$

Questions

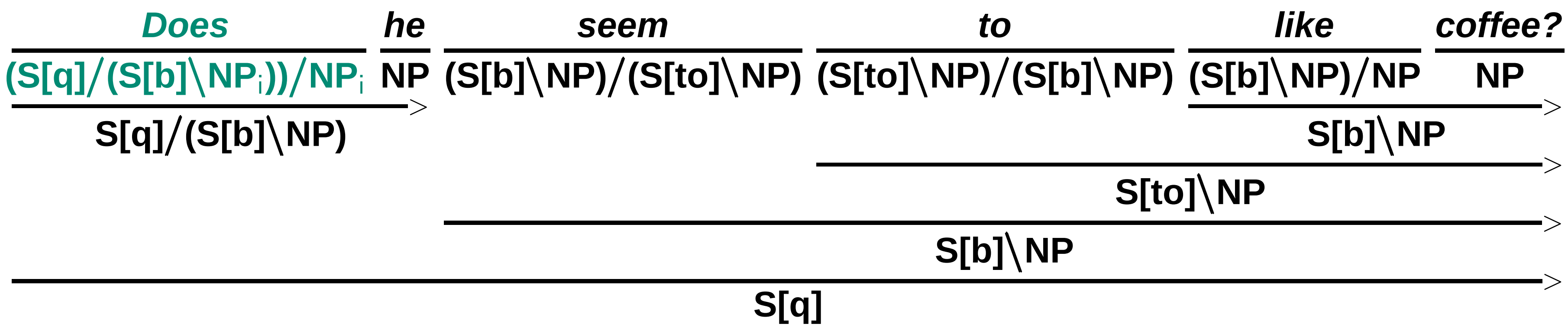


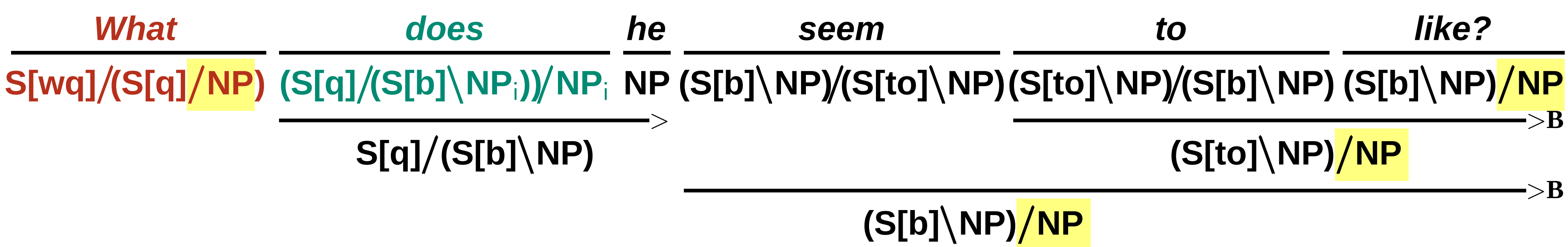
Does he seem to like coffee?
(S[q]/(S[b]\NP_i))/NP_i
NP
(S[b]\NP)/(S[to]\NP)
(S[to]\NP)/(S[b]\NP)
(S[b]\NP)/NP
NP
>
S[q]/(S[b]\NP)
S[b]\NP
S[to]\NP
S[b]\NP
S[q]



What does he seem to like?
S[wq]/(S[q]/NP)
(S[q]/(S[b]\NP_i))/NP_i
NP
(S[b]\NP)/(S[to]\NP)
(S[to]\NP)/(S[b]\NP)
(S[b]\NP)/NP
>
>B
S[q]/(S[b]\NP)
(S[to]\NP)/NP

Questions


Does
he
seem
to
like
coffee?
(S[q]/(S[b]\NPᵢ))/NPᵢ
NP
(S[b]\NP)/(S[to]\NP)
(S[to]\NP)/(S[b]\NP)
(S[b]\NP)/NP
NP
>
S[q]/(S[b]\NP)
>
S[b]\NP
>
S[to]\NP
>
S[b]\NP
>
S[q]


What
does
he
seem
to
like?
S[wq]/(S[q]/NP)
(S[q]/(S[b]\NPᵢ))/NPᵢ
NP
(S[b]\NP)/(S[to]\NP)
(S[to]\NP)/(S[b]\NP)
(S[b]\NP)/NP
>
S[q]/(S[b]\NP)
>B
(S[to]\NP)/NP
>B
(S[b]\NP)/NP

Questions

Does *he* *seem* *to* *like* *coffee?*

(S[q]/(S[b]\NP$_i$))/NP$_i$ NP (S[b]\NP)/(S[to]\NP) (S[to]\NP)/(S[b]\NP) (S[b]\NP)/NP NP

S[q]/(S[b]\NP) (>)

S[b]\NP (>)

S[to]\NP (>)

S[b]\NP (>)

S[q] (>)

What *does* *he* *seem* *to* *like?*

S[wq]/(S[q]/NP) (S[q]/(S[b]\NP$_i$))/NP$_i$ NP (S[b]\NP)/(S[to]\NP) (S[to]\NP)/(S[b]\NP) (S[b]\NP)/NP

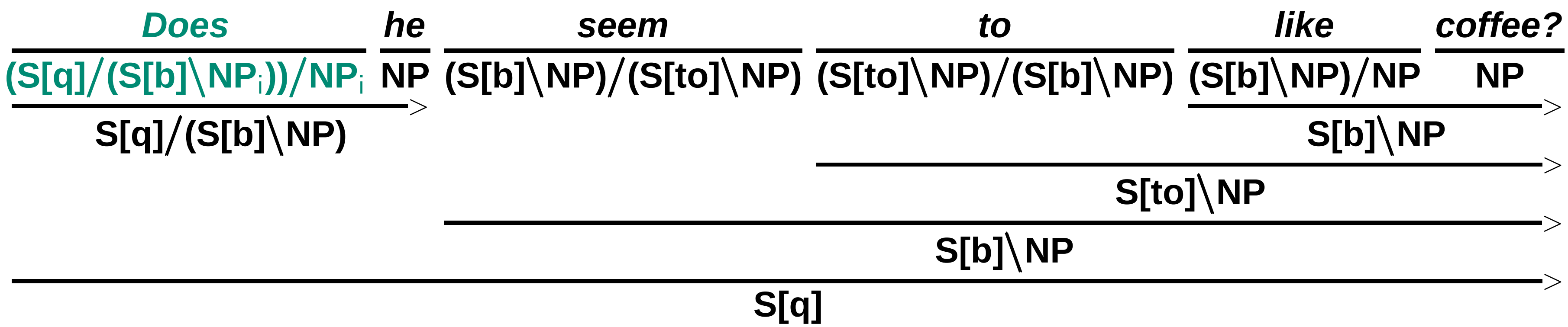
S[q]/(S[b]\NP) (>)

(S[to]\NP)/NP (>B)

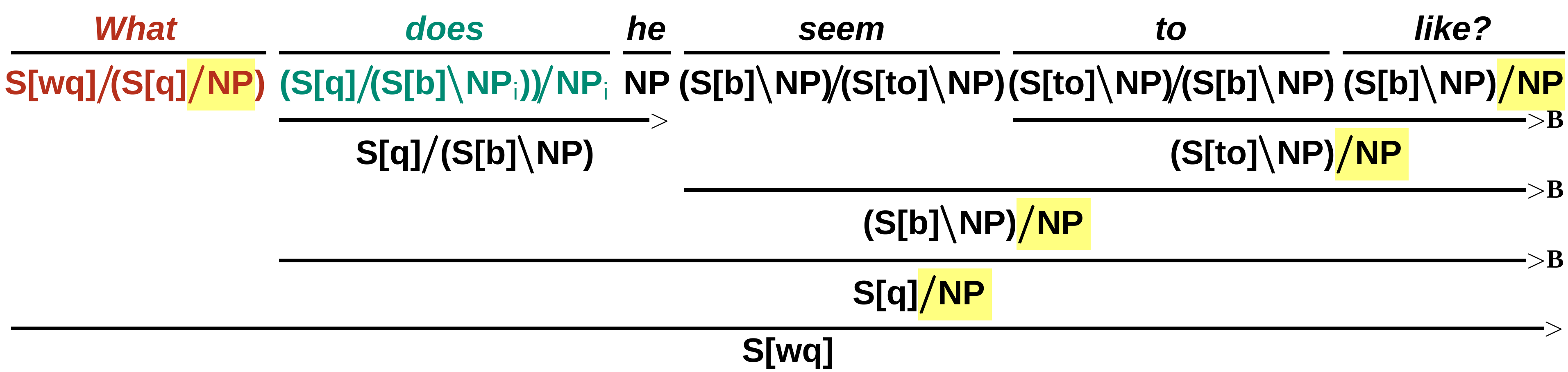
(S[b]\NP)/NP (>B)

S[q]/NP (>B)

Questions



Does
he
seem
to
like
coffee?
(S[q]/(S[b]\NPᵢ))/NPᵢ
NP
(S[b]\NP)/(S[to]\NP)
(S[to]\NP)/(S[b]\NP)
(S[b]\NP)/NP
NP
>
S[q]/(S[b]\NP)
S[b]\NP
S[to]\NP
S[b]\NP
S[q]



What
does
he
seem
to
like?
S[wq]/(S[q]/NP)
(S[q]/(S[b]\NPᵢ))/NPᵢ
NP
(S[b]\NP)/(S[to]\NP)
(S[to]\NP)/(S[b]\NP)
(S[b]\NP)/NP
>B
S[q]/(S[b]\NP)
(S[to]\NP)/NP
(S[b]\NP)/NP
S[q]/NP
S[wq]

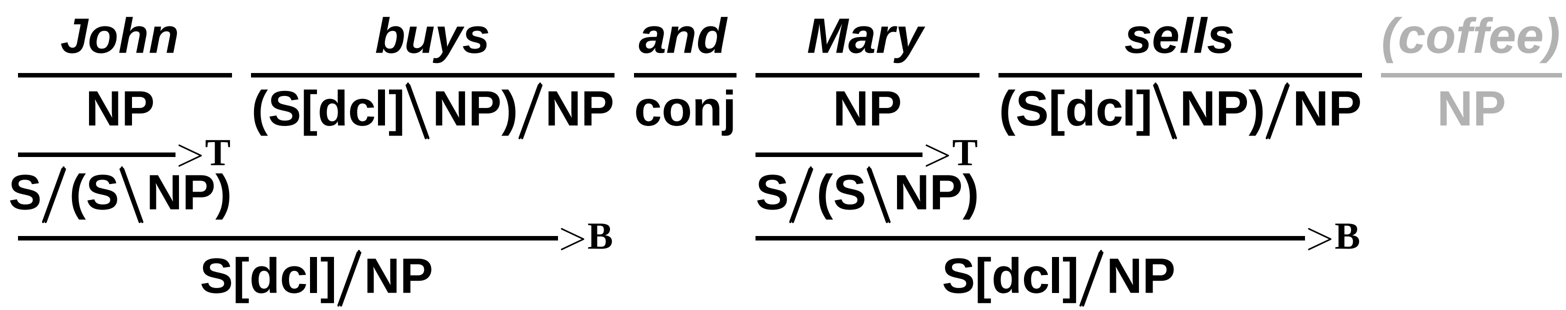
Right node raising

- Use type-raising and composition to form “incomplete” constituents.
- RNR is just coordination of such “incomplete” constituents:

John	*buys*	*and*	*Mary*	*sells*	*(coffee)*
NP	$(S[dcl]\backslash NP)/NP$	conj	NP	$(S[dcl]\backslash NP)/NP$	NP

Right node raising

- Use type-raising and composition to form “incomplete” constituents.
- RNR is just coordination of such “incomplete” constituents:



Argument cluster coordination

I	*give*	*her*	*flowers*	*and*	*him*	*whisky*
NP	**((S\NP)/NP)/NP**	**NP**	**NP**	**conj**	**NP**	**NP**

Argument cluster coordination

- Use type-raising and composition to combine the argument clusters.

I	*give*	*her*	*flowers*	*and*	*him*	*whisky*
NP	((S\NP)/NP)/NP	NP	NP	conj	NP	NP

Argument cluster coordination

- Use type-raising and composition to combine the argument clusters.

I	*give*	*her*	*flowers*	*and*	*him*	*whisky*
NP	((S\NP)/NP)/NP	NP	NP	conj	NP	NP
		<T	<T		<T	<T
		T\(T/NP)	U\(U/NP)		T\(T/NP)	U\(U/NP)

Argument cluster coordination

- Use type-raising and composition to combine the argument clusters.

I	*give*	*her*	*flowers*	*and*	*him*	*whisky*
NP	((S\NP)/NP)/NP	NP	NP	conj	NP	NP
		T\(T/NP) <T	U\(U/NP) <T		T\(T/NP) <T	U\(U/NP) <T
		(U\((U/NP)/NP)) <B			(U\((U/NP)/NP)) <B	

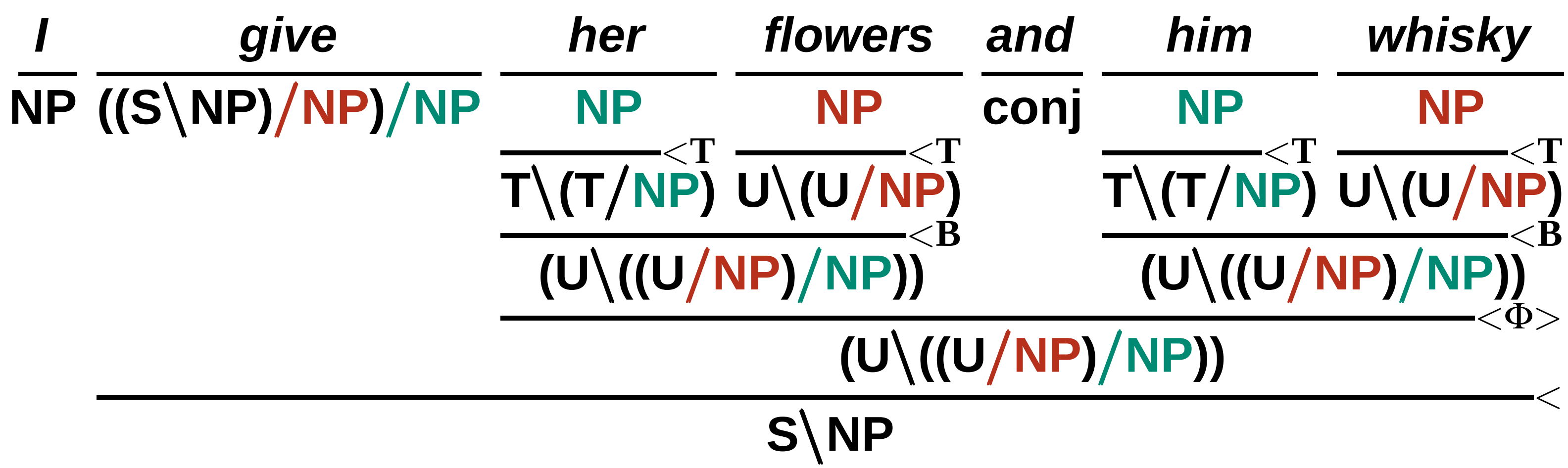
Argument cluster coordination

- Use type-raising and composition to combine the argument clusters.

I	*give*	*her*	*flowers*	*and*	*him*	*whisky*
NP	((S\NP)/NP)/NP	NP	NP	conj	NP	NP
		T\(T/NP) <T	U\(U/NP) <T		T\(T/NP) <T	U\(U/NP) <T
		(U\((U/NP)/NP)) <B			(U\((U/NP)/NP)) <B	
		(U\((U/NP)/NP)) <Φ>				

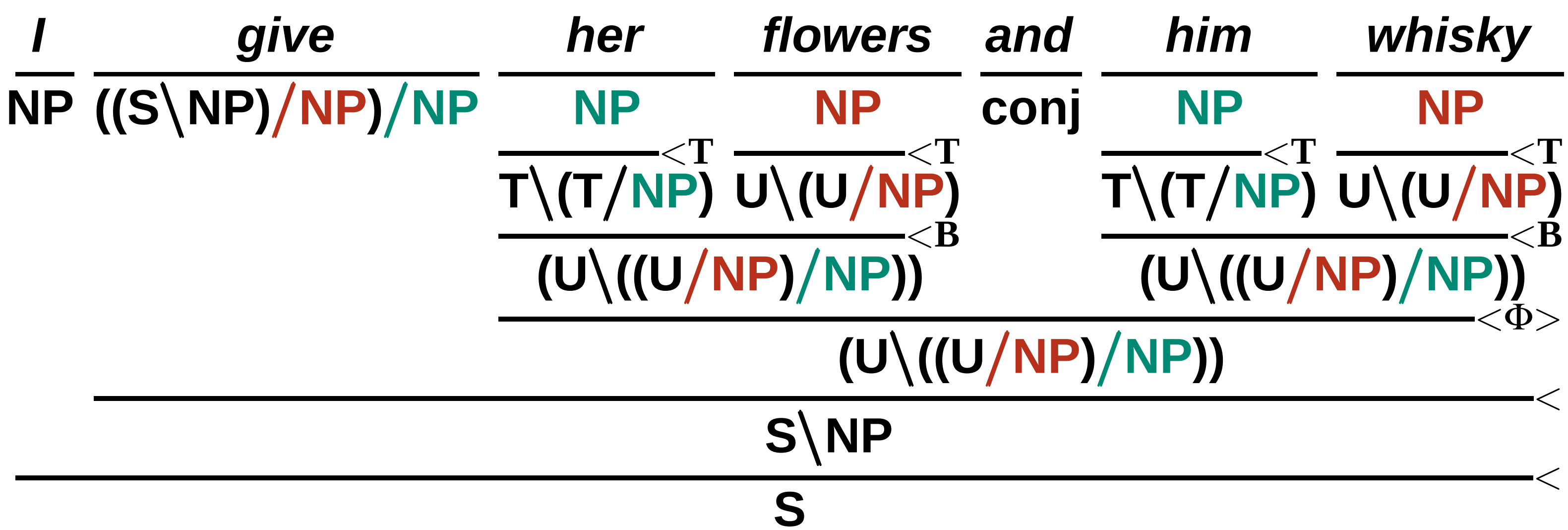
Argument cluster coordination

- Use type-raising and composition to combine the argument clusters.



Argument cluster coordination

- Use type-raising and composition to combine the argument clusters.



Scrambling

Verb has standard category. Use type-raising and (crossing) composition