LECTURE 19: LINGUISTICALLY EXPRESSIVE GRAMMARS

Mehmet Can Yavuz, PhD.

Adapted from Julia Hockenmaier, NLP S2023 - course material
https://courses.grainger.illinois.edu/cs447/sp2023/

PART 3: FEATURE STRUCTURE GRAMMARS

WHY FEATURE STRUCTURES

Feature structures form the basis for many grammar formalisms used in computational linguistics.

Feature structure grammars (aka attribute-value grammars, or unification grammars) can be used as

- a more compact way of representing rich CFGs
- a way to represent more expressive grammars

SIMPLE GRAMMARS OVERGENERATE

$$
\begin{aligned}
S &\rightarrow NP \;\; VP \\
VP &\rightarrow Verb \;\; NP \\
NP &\rightarrow Det \;\; Noun \\
Det &\rightarrow the \mid a \mid these \\
Verb &\rightarrow eat \mid eats \\
Noun &\rightarrow cake \mid cakes \mid student \mid students
\end{aligned}
$$

This generates ungrammatical sentences like
"these student eats a cakes"

We need to capture (number/person) agreement

REFINING THE NONTERMINALS

$$\begin{array}{rcl}
S & \rightarrow & NPsg \quad VPsg \\
S & \rightarrow & NPpl \quad VPpl \\
VPsg & \rightarrow & VerbSg \quad NP \\
VPpl & \rightarrow & VerbPl \quad NP \\
NPsg & \rightarrow & DetSg \quad NounSg \\
DetSg & \rightarrow & the \mid a \\
\ldots & \ldots & \ldots
\end{array}$$

- This yields very large grammars.
- What about person, case, ...?
- Difficult to capture generalizations (Subject and verb have to have number agreement)
- *NPsg*, *NPpl* and *NP* are three distinct nonterminals

FEATURE STRUCTURES

- Replace atomic categories with feature structures:

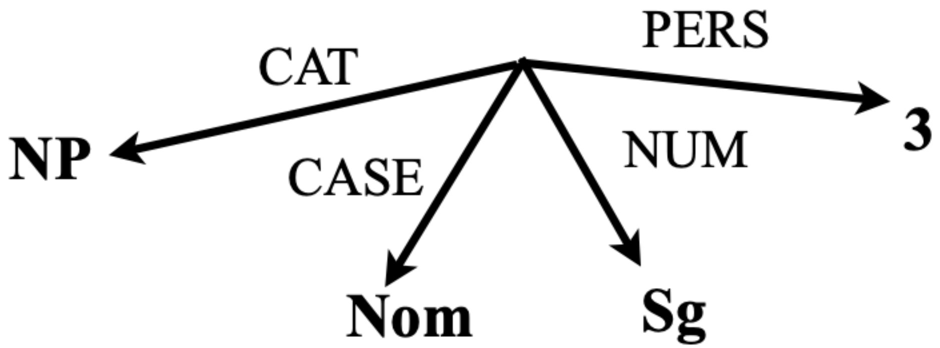
$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUM} & \text{SG} \\ \text{PERS} & 3 \\ \text{CASE} & \text{NOM} \end{bmatrix} \qquad \begin{bmatrix} \text{CAT} & \text{VP} \\ \text{NUM} & \text{SG} \\ \text{PERS} & 3 \\ \text{VFORM} & \text{FINITE} \end{bmatrix}$$

A feature structure is a list of features (= attributes, e.g. CASE), and values (e.g. NOM).

We often represent feature structures as attribute value matrices (AVMs)
Usually, values are typed (to avoid CASE:SG)

FEATURE STRUCTURES AS DIRECTED GRAPHS

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUM} & \text{SG} \\ \text{PERS} & 3 \\ \text{CASE} & \text{NOM} \end{bmatrix} =$$



COMPLEX FEATURE STRUCTURES

- We distinguish between atomic and complex feature values.
- A complex value is a feature structure itself.
- This allows us to capture better generalizations.

Only atomic values:

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUM} & \text{SG} \\ \text{PERS} & 3 \\ \text{CASE} & \text{NOM} \end{bmatrix}$$

Complex values:

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{AGR} & \begin{bmatrix} \text{NUM} & \text{SG} \\ \text{PERS} & 3 \\ \text{CASE} & \text{NOM} \end{bmatrix} \end{bmatrix}$$

FEATURE PATHS

$$\textbf{NP:}\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{AGR} & \begin{bmatrix} \text{NUM} & \text{SG} \\ \text{PERS} & 3 \\ \text{CASE} & \text{NOM} \end{bmatrix} \end{bmatrix}$$

A feature path allows us to identify particular values in a feature structure:

$\langle$**NP** CAT$\rangle$ = NP

$\langle$**NP** AGR CASE$\rangle$ = NOM

UNIFICATION

- Two feature structures A and B **unify** (A ⊔ B) if they can be merged into one consistent feature structure C:

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUM} & \text{SG} \\ \text{CASE} & \text{NOM} \end{bmatrix} \sqcup \begin{bmatrix} \text{CAT} & \text{NP} \\ \text{PERS} & 3 \end{bmatrix} = \begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUM} & \text{SG} \\ \text{PERS} & 3 \\ \text{CASE} & \text{NOM} \end{bmatrix}$$

- Otherwise, unification fails:

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUM} & \text{SG} \\ \text{CASE} & \text{NOM} \end{bmatrix} \sqcup \begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUM} & \text{PL} \end{bmatrix} = \emptyset$$

PATR-II STYLE FEATURE STRUCTURES

- CFG rules are augmented with constraints:
- **A0 →A1...An**

  {set of constraints}
- There are two kinds of constraints:
  - Unification constraints:
    - $\mathbf{A_i}$ feature-path□ = $\mathbf{A_j}$ feature-path□
- Value constraints:
  - $\mathbf{A_i}$ feature-path□ = atomic value

A GRAMMAR WITH FEATURE STRUCTURES

S	→	**NP VP**		**Grammar rule**
		⟨**NP** *NUM*⟩	= ⟨**VP** *NUM*⟩	**Constraints**
		⟨**NP** *CASE*⟩	= *nom*	

NP	→	**DT NOUN**		**Grammar rule**
		⟨**NP** *NUM*⟩	= ⟨**NOUN** *NUM*⟩	**Constraints**
		⟨**NP** *CASE*⟩	= ⟨**NOUN** *CASE*⟩	

NOUN	→	*cake*		**Lexical entry**
		⟨**NOUN** *NUM*⟩	= *sg*	**Constraints**

WITH COMPLEX FEATURE STRUCTURES

S	$\rightarrow$	**NP VP**		**Grammar rule**
		⟨**NP** *AGR*⟩	= ⟨**VP** *AGR*⟩	**Constraints**
		⟨**NP** *CASE*⟩	= *nom*	

NP	$\rightarrow$	**DT NOUN**		**Grammar rule**
		⟨**NP** *AGR*⟩	= ⟨**NOUN** *AGR*⟩	**Constraints**

NOUN	$\rightarrow$	*cake*		**Lexical entry**
		⟨**NOUN** AGR NUM⟩	= *sg*	**Constraints**

Complex feature structures can capture **better generalizations** (and hence require fewer constraints) — cf. the previous slide

THE HEAD FEATURE

- Instead of implicitly specifying heads for each rewrite rule, let us define a **head feature**.
- The head of a VP has the same agreement feature as the VP itself:

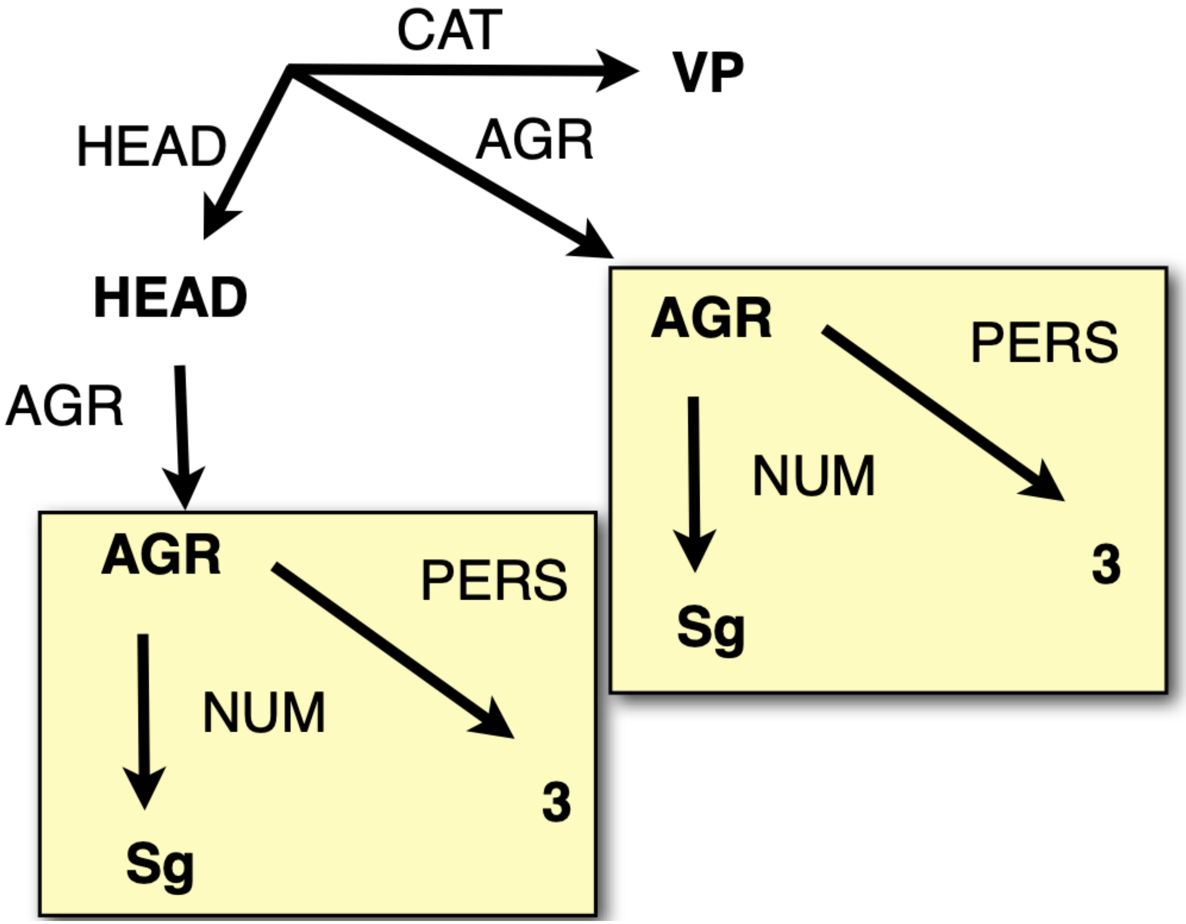
$$\begin{bmatrix} \text{CAT} & \text{VP} \\ \text{AGR} & \begin{bmatrix} \text{NUM} & \text{SG} \\ \text{PERS} & 3 \end{bmatrix} \\ \text{HEAD} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{NUM} & \text{SG} \\ \text{PERS} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

RE-ENTRANCIES

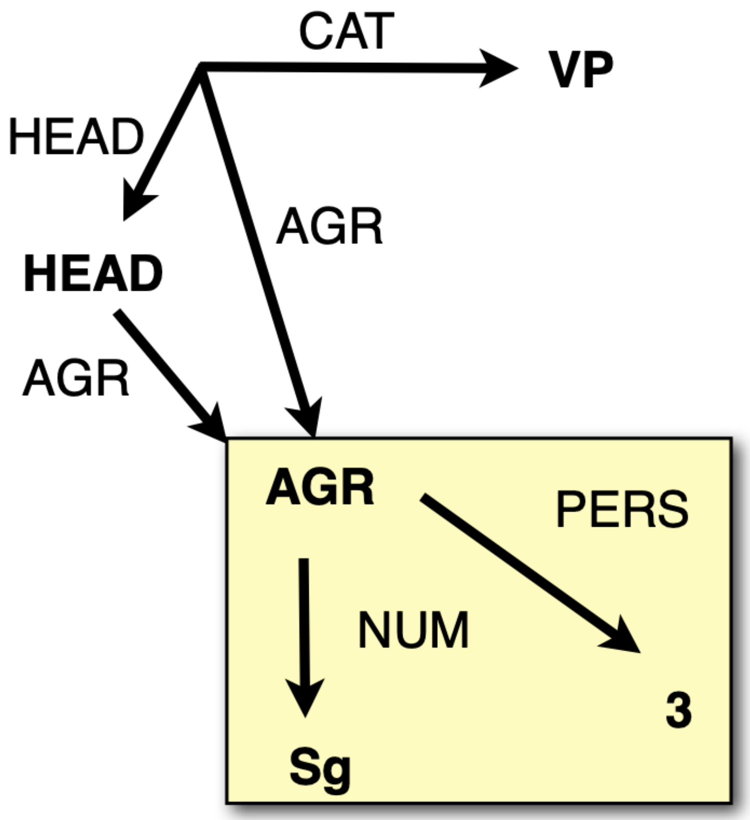
- What we *really* want to say is that the agreement feature of the head is *identical* to that of the VP itself.
- This corresponds to a re-entrancy in the FS (indicated via coindexation $\boxed{1}$)

$$\begin{bmatrix} \mathrm{CAT} & \mathrm{VP} \\ \mathrm{AGR} & \boxed{1} \begin{bmatrix} \mathrm{NUM} & \mathrm{SG} \\ \mathrm{PERS} & 3 \end{bmatrix} \\ \mathrm{HEAD} & \begin{bmatrix} \mathrm{AGR} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

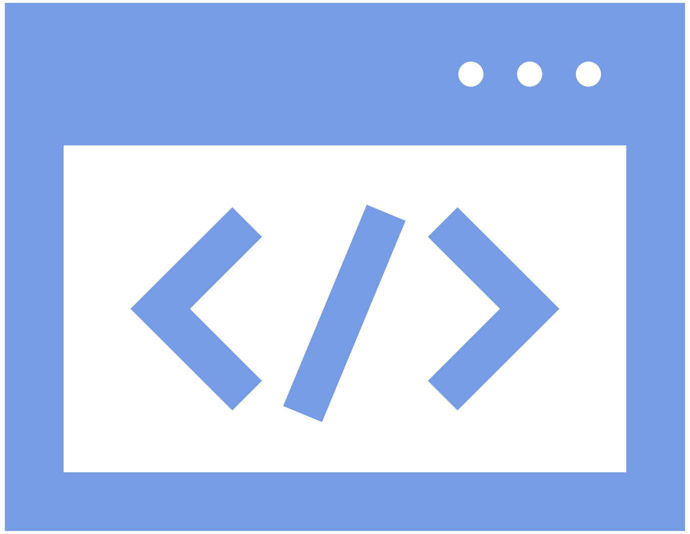
RE-ENTRANCIES — NOT LIKE THIS:



RE-ENTRANCIES — BUT LIKE THIS:



ATTRIBUTE-VALUE GRAMMARS AND CFGS



- If every feature can only have a finite set of values, any attribute-value grammar can be compiled out into a (possibly huge) context-free grammar

GOING BEYOND CFGS

- The power-of-2 language: $L_2 = \{w_i \mid i \text{ is a power of } 2\}$
- L_2 is a (fully) context-sensitive language.

  (*Mildly* context-sensitive languages have the **constant growth property** (the length of words always increases by a constant factor c))

Here is a feature grammar which generates L_2:

$$
\begin{aligned}
\mathrm{A} \rightarrow \quad & a \\
& \langle \mathrm{A}\ F \rangle = 1 \\
\mathrm{A} \rightarrow \quad & \mathrm{A}_1 \quad \mathrm{A}_2 \\
& \langle \mathrm{A}\ F \rangle = \langle \mathrm{A}_1 \rangle \\
& \langle \mathrm{A}\ F \rangle = \langle \mathrm{A}_2 \rangle
\end{aligned}
$$

PART 4: TREE-ADJOINING GRAMMAR

(LEXICALIZED) TREE-ADJOINING GRAMMAR

TAG is a tree-rewriting formalism:
TAG defines operations (**substitution**, **adjunction**) on trees. The **elementary objects** in TAG are trees (not strings)

TAG is lexicalized:
Each elementary tree is **anchored** to a lexical item (word)

"Extended domain of locality":
The elementary tree contains all arguments of the anchor.

TAG requires a linguistic theory which specifies the shape of these elementary trees.

TAG is mildly context-sensitive:
can capture Dutch cross-serial dependencies but is still efficiently parseable

AK Joshi and Y Schabes (1996) Tree Adjoining Grammars. In G. Rosenberg and A. Salomaa, Eds., Handbook of Formal

MILDLY CONTEXT-SENSITIVE GRAMMARS

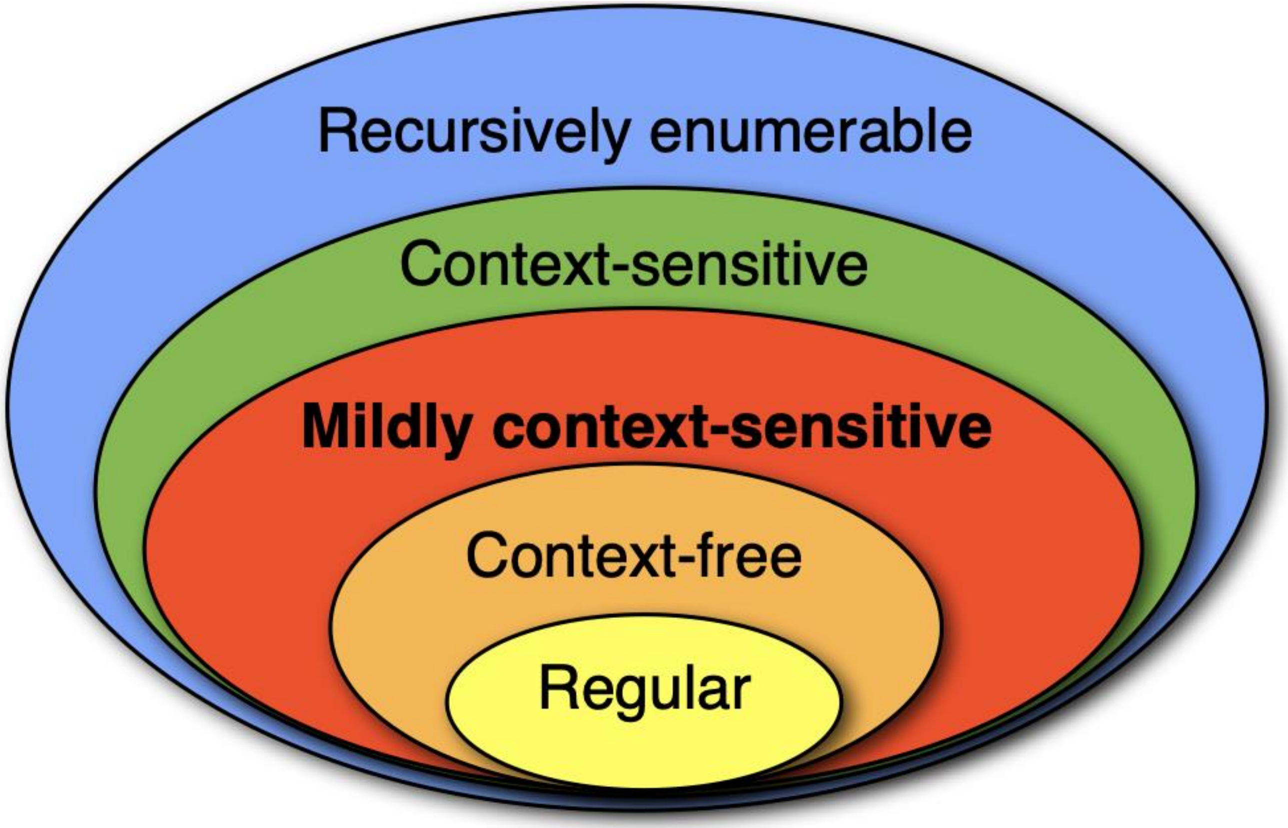
Contain all context-free grammars/languages

Can be **parsed in polynomial time** (TAG/CCG: O(n6))

(*Strong* generative capacity) capture certain kinds of dependencies: **nested** (like CFGs) and **cross-serial** (like the Dutch example), but not the MIX language:
MIX: the set of strings $\boldsymbol{w} \in \{a, b, c\}^*$ that contain equal numbers of *a*s, *b*s and *c*s

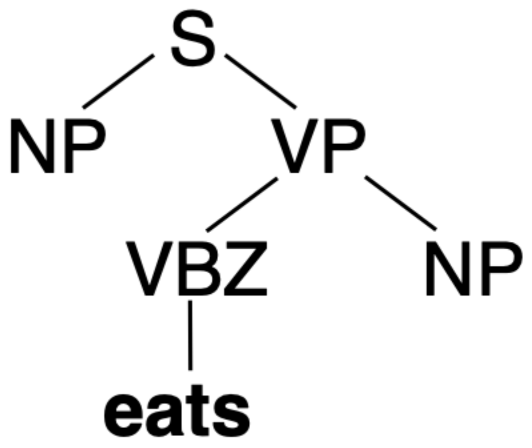
Have the **constant growth** property:
the length of strings grows in a linear way
The power-of-2 language {*a2n*} does not have the constant growth propery.

THE CHOMSKY HIERARCHY

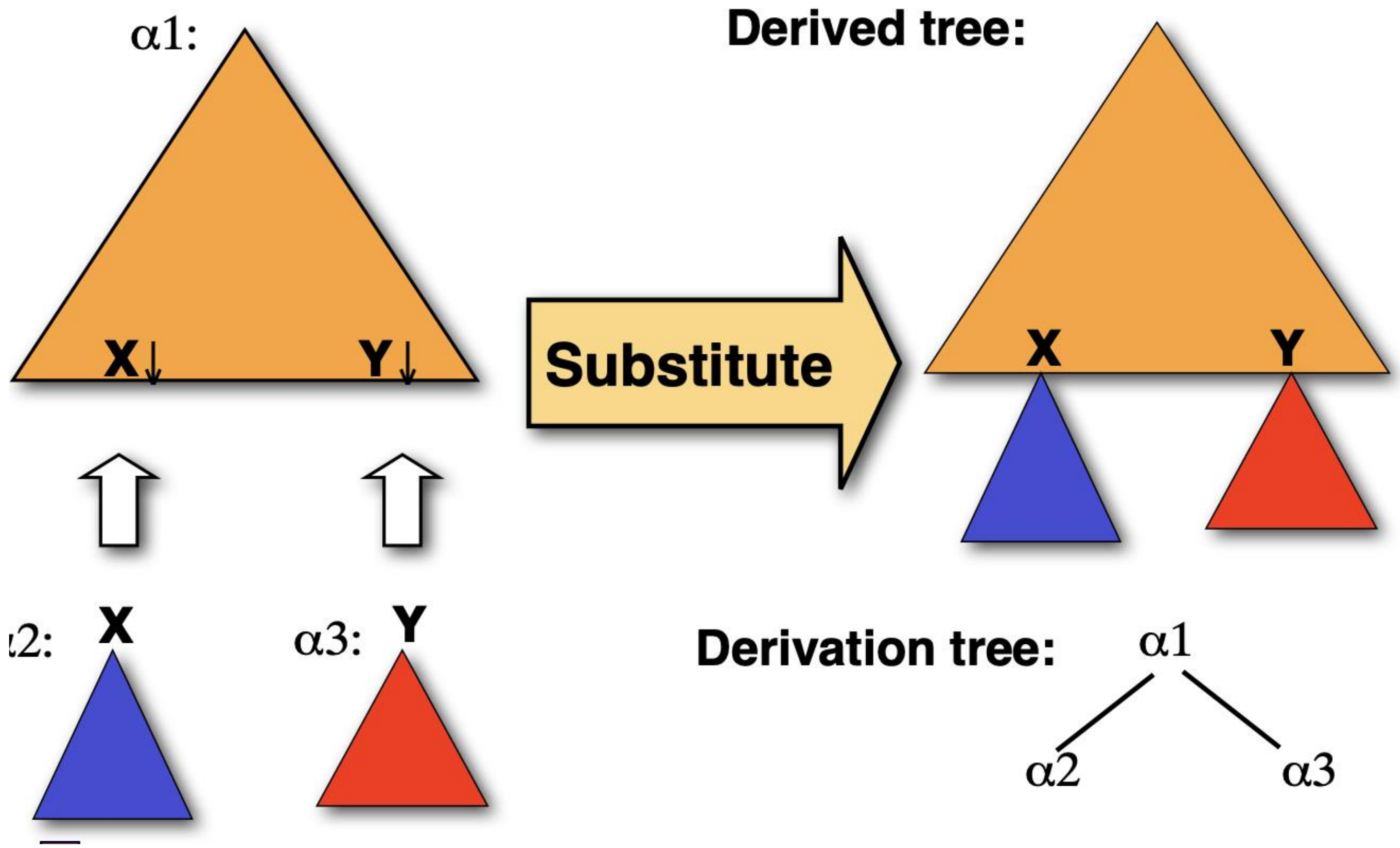


EXTENDED DOMAIN OF LOCALITY

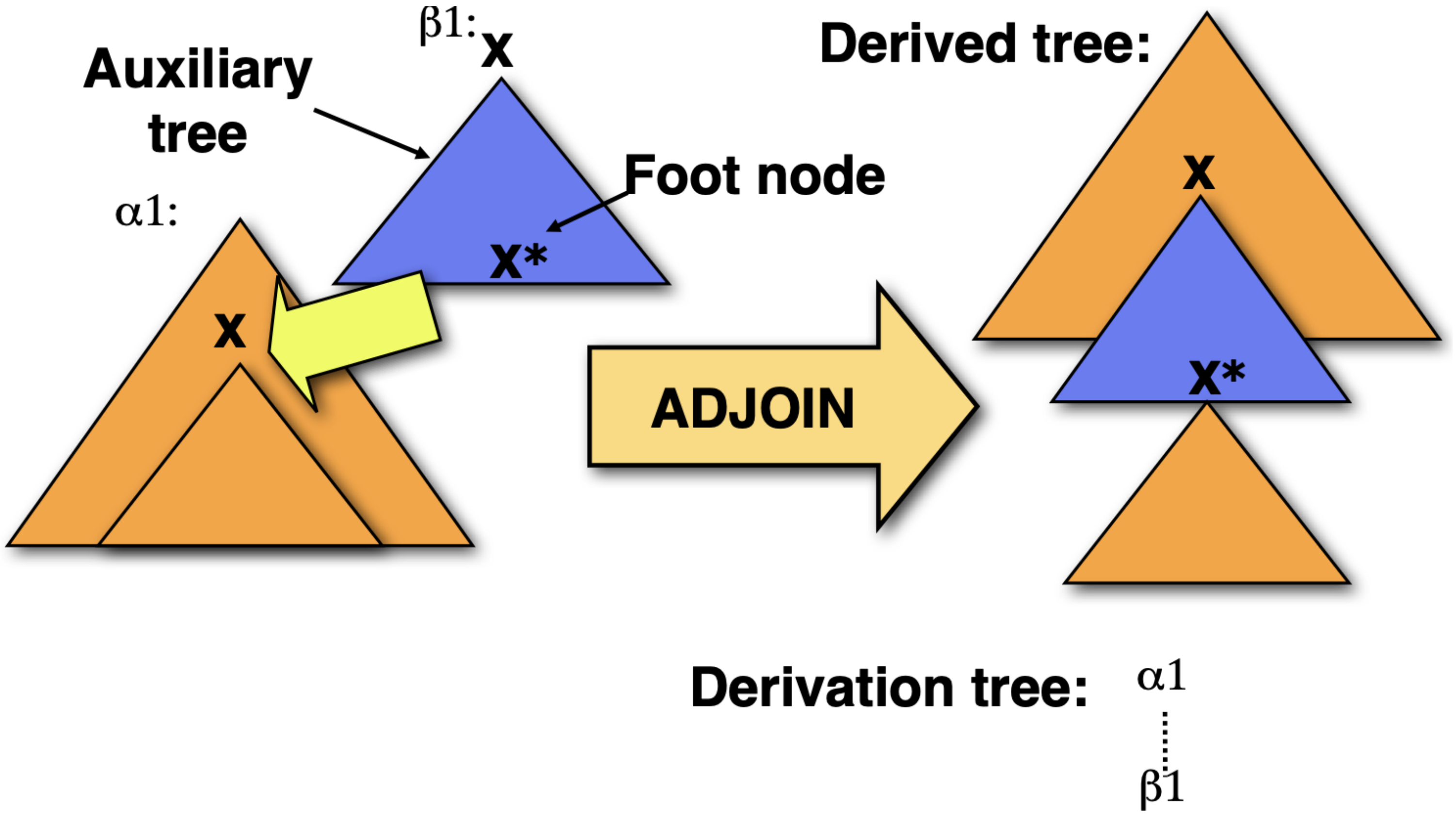
- We want to capture **all arguments of a word** in a **single elementary object**.
- We also want to retain certain syntactic structures (e.g. VPs).
- Our elementary objects are **tree fragments:**



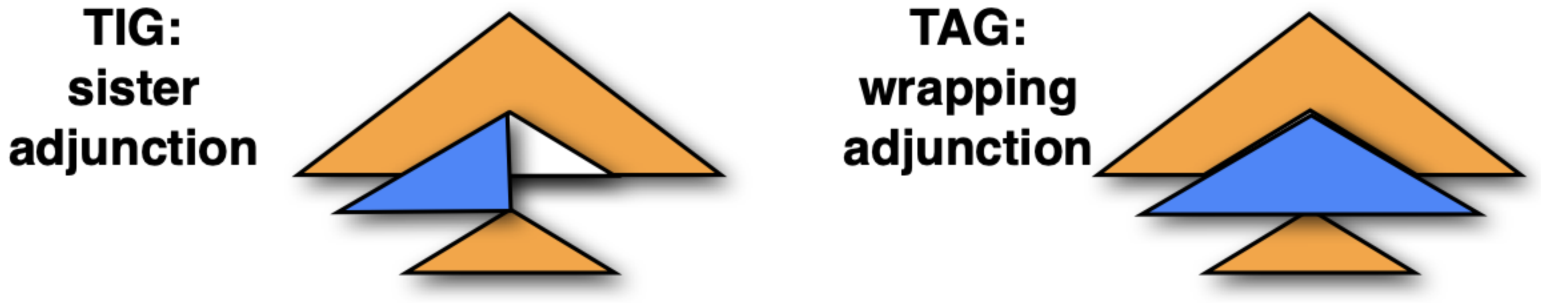
TAG SUBSTITUTION (ARGUMENTS)



TAG ADJUNCTION



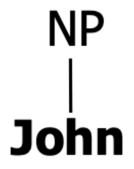
THE EFFECT OF ADJUNCTION



- No adjunction: TSG (Tree substitution grammar)
  TSG is context-free
- Sister adjunction: TIG (Tree insertion grammar)
  TIG is also context-free, but has a linguistically more adequate treatment of modifiers
- Wrapping adjunction: TAG (Tree-adjoining grammar)
  TAG is mildy context-sensitive

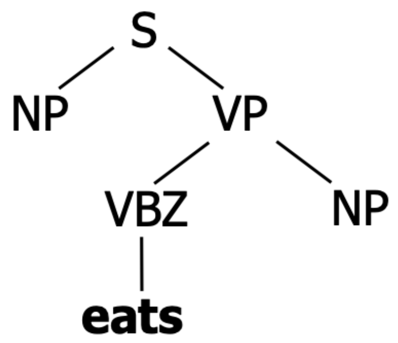
A SMALL TAG LEXICON

α2:

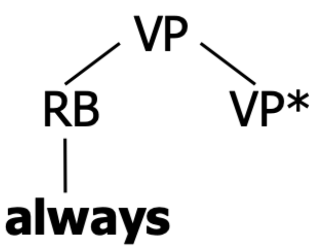


α3:

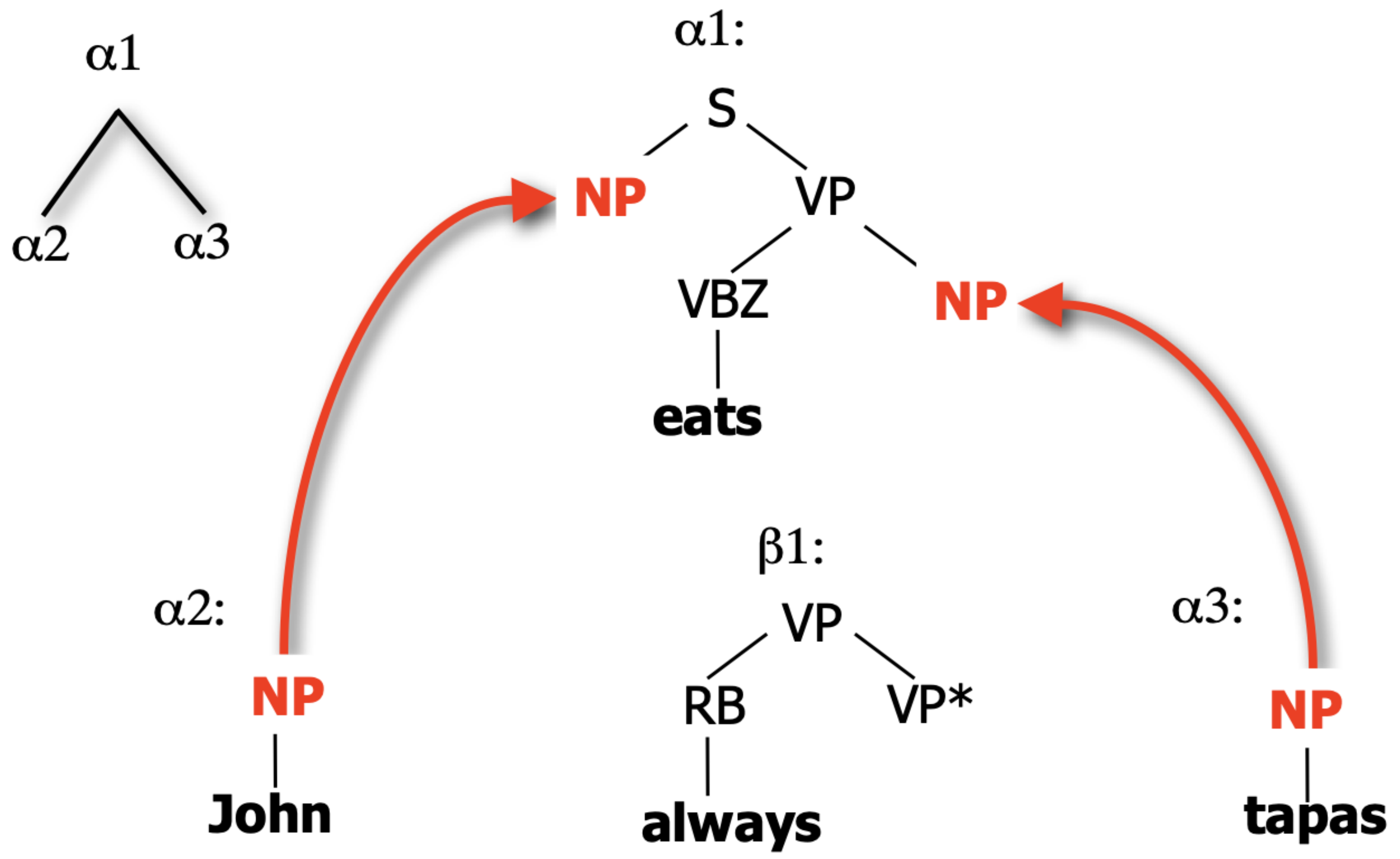
α1:



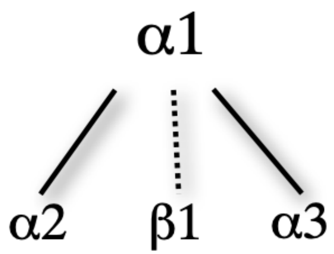
β1:

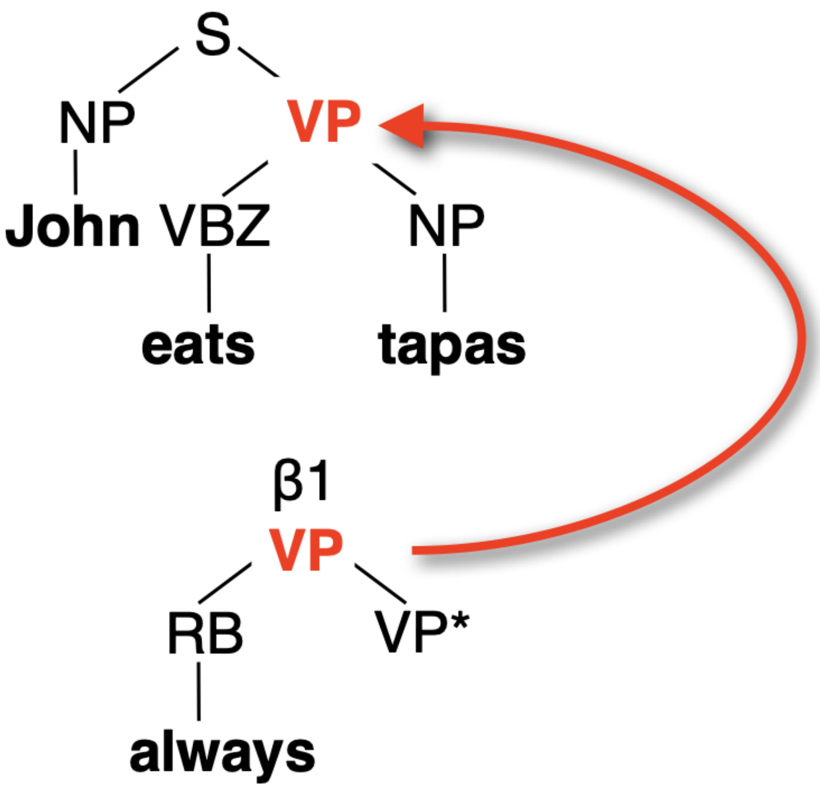


A TAG DERIVATION

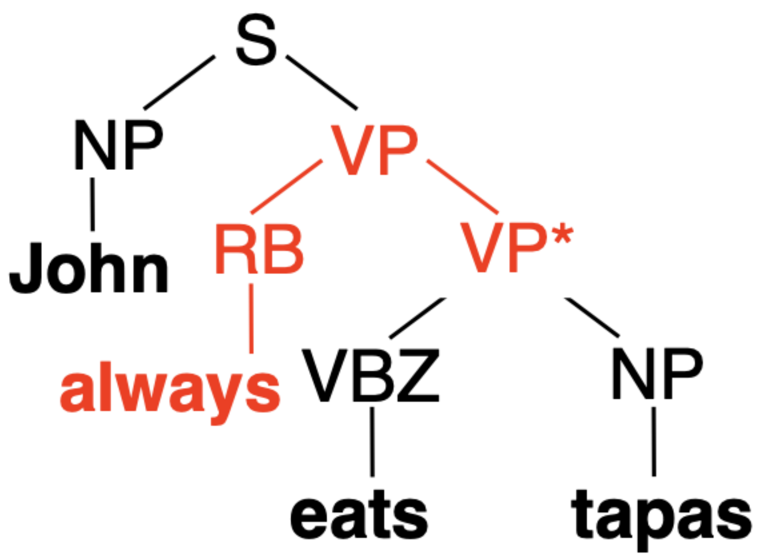


A TAG DERIVATION





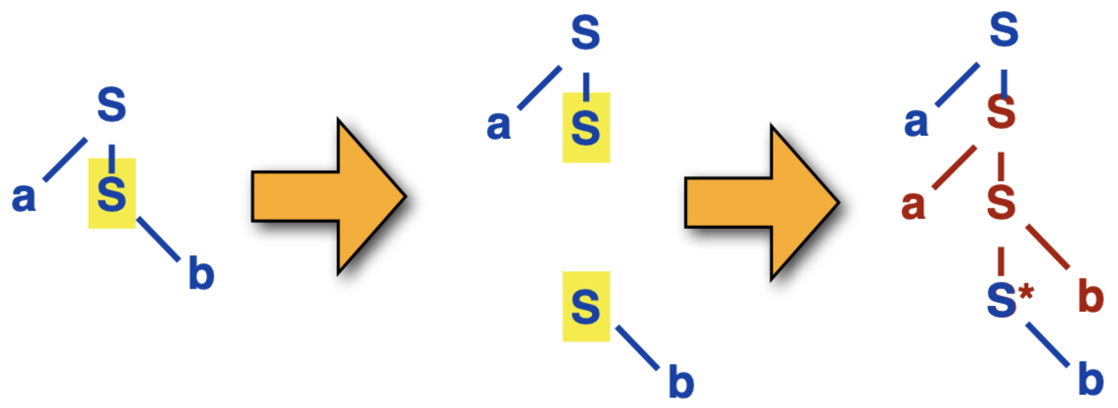
A TAG DERIVATION



$A^{N}B^{N}$: CROSS-SERIAL DEPENDENCIES

Elementary trees:

Deriving aabb



PART 5: (COMBINATORY) CATEGORIAL GRAMMAR

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 (atomic) categories: **NP, S, PP**

Complex categories (functions):

Return a **result** when combined with an **argument**

VP, intransitive verb	**S\NP**
Transitive verb	**(S\NP)/NP**
Adverb	**(S\NP)\(S\NP)**
Prepositions	**((S\NP)\(S\NP))/NP** **(NP\NP)/NP** **PP/NP**

CCG CATEGORIES ARE FUNCTIONS

CCG has **a few atomic categories, e.g**

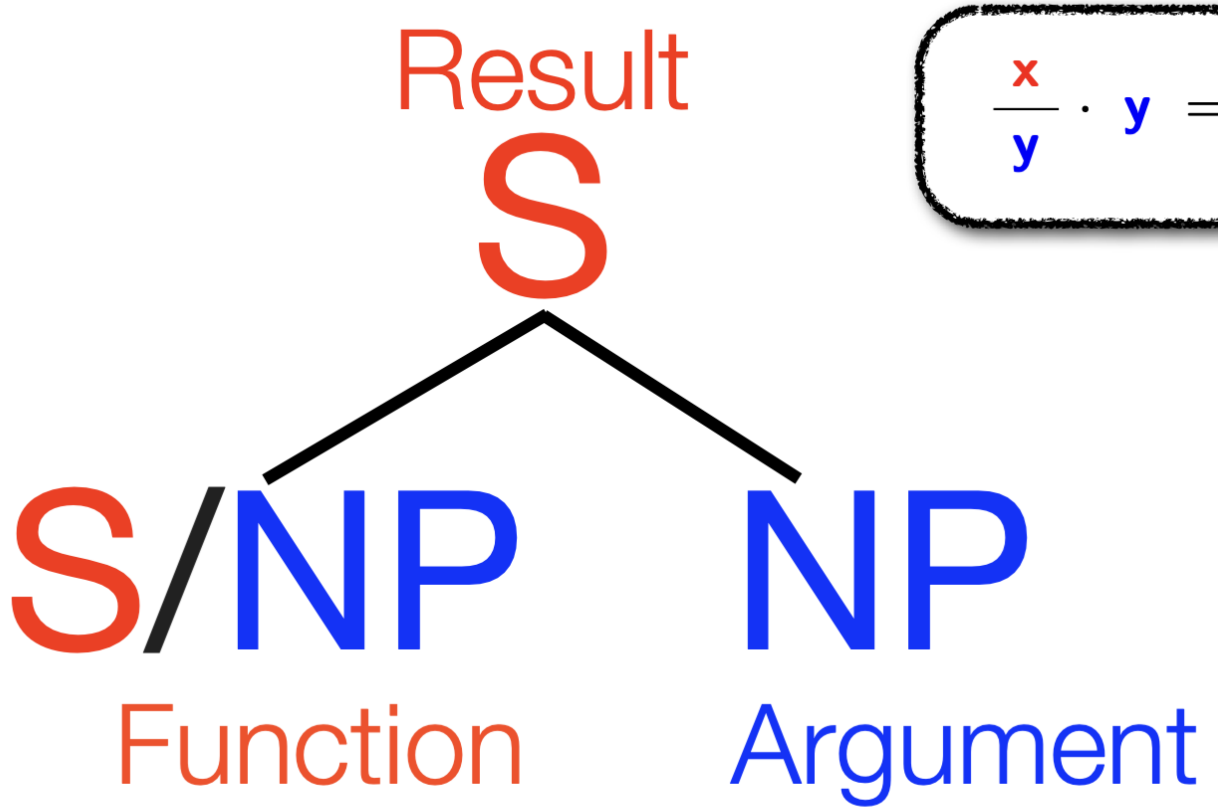
S, NP, PP

All other CCG categories are **functions:**

S / NP

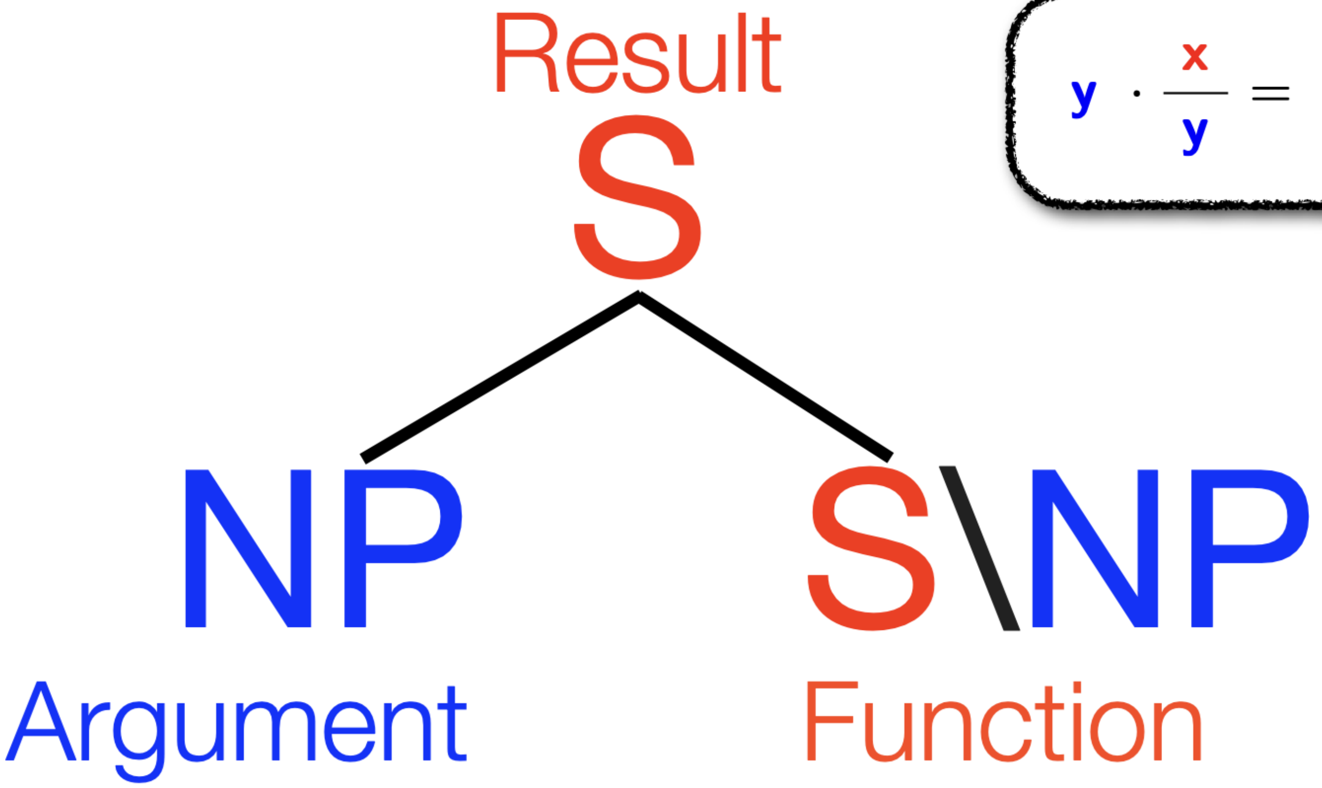
Result Dir. Argument

RULES: FUNCTION APPLICATION



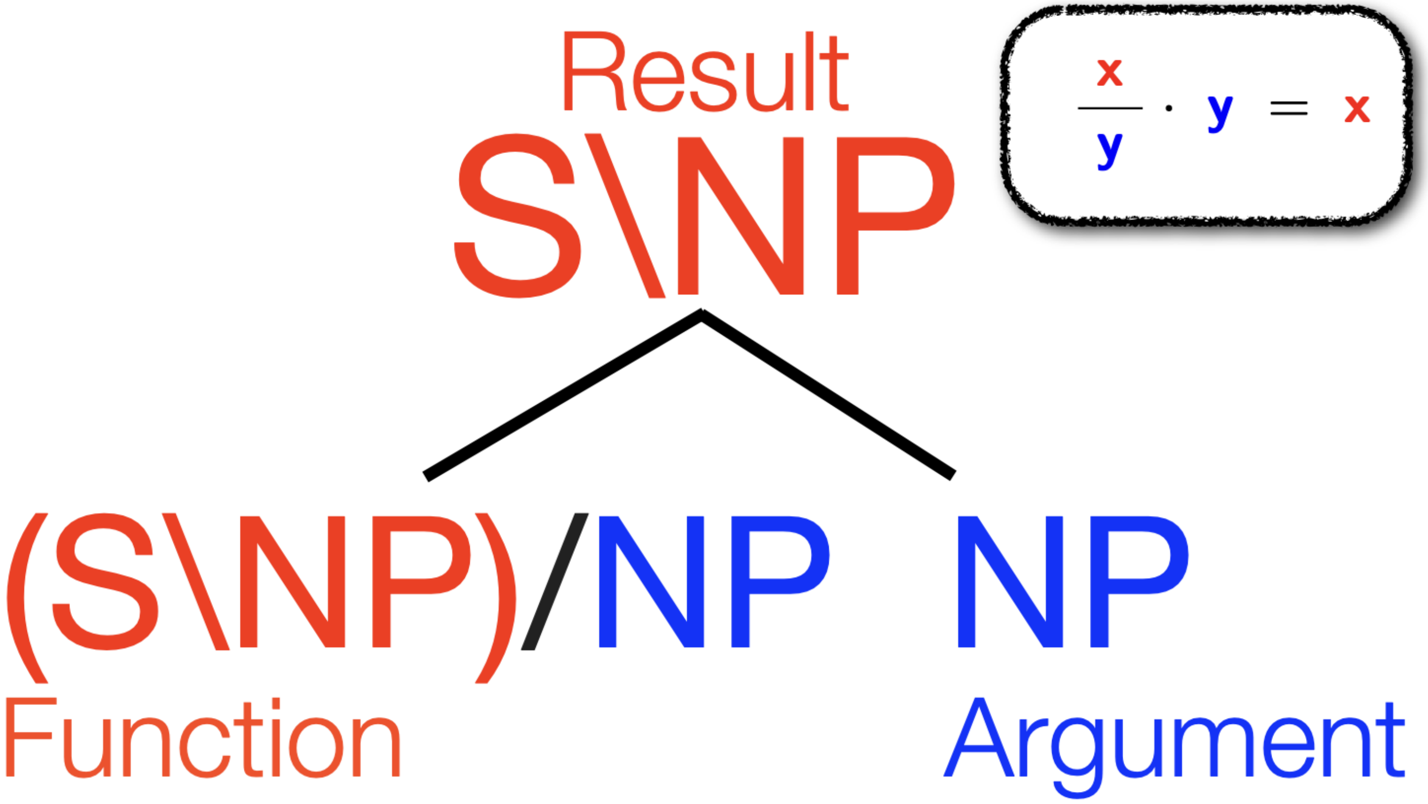
$$\boxed{\frac{x}{y} \cdot y = x}$$

RULES: FUNCTION APPLICATION



$$y \cdot \frac{x}{y} = x$$

RULES: FUNCTION APPLICATION



FUNCTION APPLICATION

Forward application (>):

(S\NP)/NP	NP	$\Rightarrow_{>}$	S\NP
eats	tapas		eats tapas

Backward application (<):

NP	S\NP	$\Rightarrow_{<}$	S
John	eats tapas		John eats tapas

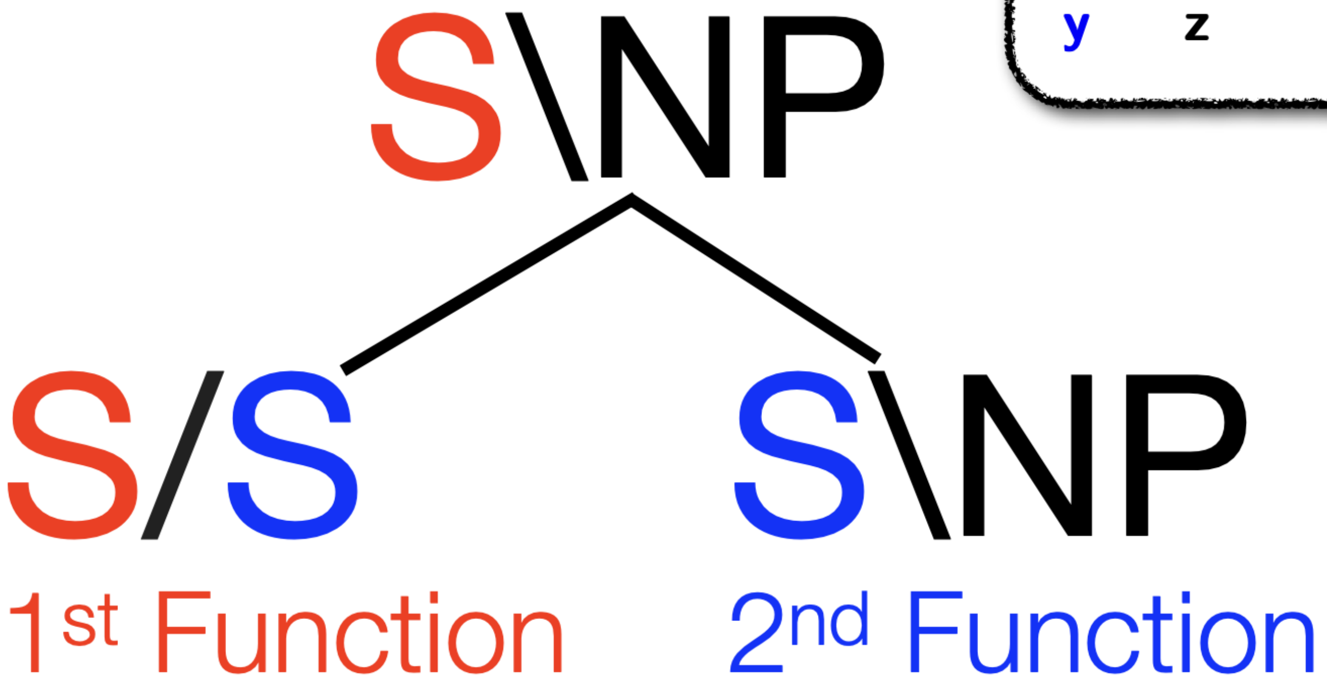
Combines function X/Y or X\Y with argument Y to yield result X
Used in all variants of categorial grammar

A (C)CG DERIVATION

$$\dfrac{\dfrac{\textbf{John}}{\mathsf{NP}} \qquad \dfrac{\dfrac{\textbf{eats}}{(\mathsf{S}\backslash \mathsf{NP})/\mathsf{NP}} \qquad \dfrac{\textbf{tapas}}{\mathsf{NP}}}{\mathsf{S}\backslash \mathsf{NP}}{>}}{\mathsf{S}}{<}$$

RULES: FUNCTION COMPOSITION

$$\frac{x}{y} \cdot \frac{y}{z} = \frac{x}{z}$$



RULES: TYPE-RAISING

$$S/(S\backslash NP)$$

$$NP$$

$$y = \frac{x}{x} \cdot y = \frac{x}{\left(\frac{x}{y}\right)}$$

TYPE-RAISING AND COMPOSITION

Type-raising: $X \rightarrow T/(T\backslash X)$

Turns an argument into a function.

$NP \rightarrow S/(S\backslash NP)$ (subject) $NP \rightarrow (S\backslash NP)\backslash((S\backslash NP)/NP)$ (object)

Harmonic composition: $X/Y \ Y/Z \rightarrow X/Z$

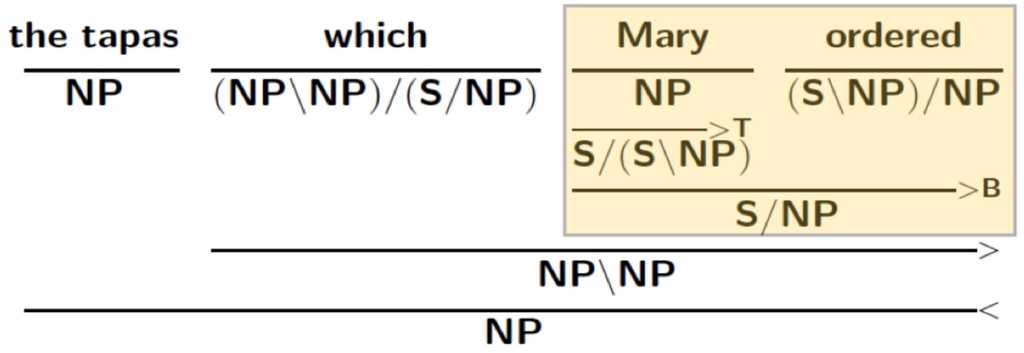
Composes two functions (complex categories), same slashes
$(S\backslash NP)/PP \ PP/NP \rightarrow (S\backslash NP)/NP$
$S/(S\backslash NP) \ (S\backslash NP)/NP \rightarrow S/NP$

Crossing composition: $X/Y \ Y\backslash Z \rightarrow X\backslash Z$

Composes two functions (complex categories), different slashes
$(S\backslash NP)/S \ S\backslash NP \rightarrow (S\backslash NP)\backslash NP$

TYPE-RAISING AND COMPOSITION

Wh-movement (relative clause):



Right-node raising: