CS447: Natural Language Processing

http://courses.engr.illinois.edu/cs447

Lecture 18: Expressive Grammars

Julia Hockenmaier

juliahmr@illinois.edu

3324 Siebel Center

Lecture 18:
Expressive Grammars

Part 1: Grammars in NLP: what and why

What is grammar?

Grammar formalisms

(= linguists' programming languages)

A precise way to define and describe the structure of sentences.

(N.B.: There are many different formalisms out there, which each define their own data structures and operations)

Specific grammars

(= linguists' programs)

Implementations (in a particular formalism) for a particular language (English, Chinese,….)

(NB: any practical parser will need to also have a model/scoring function to identify which grammatical analysis should be assigned to a given sentence)

Why study grammar?

Linguistic questions:

What kind of constructions occur in natural language(s)?

Formal questions:

Can we define formalisms that allow us to characterize which strings belong to a language?

Those formalisms have appropriate weak generative capacity

Can we define formalisms that allow us to map sentences to their appropriate structures?

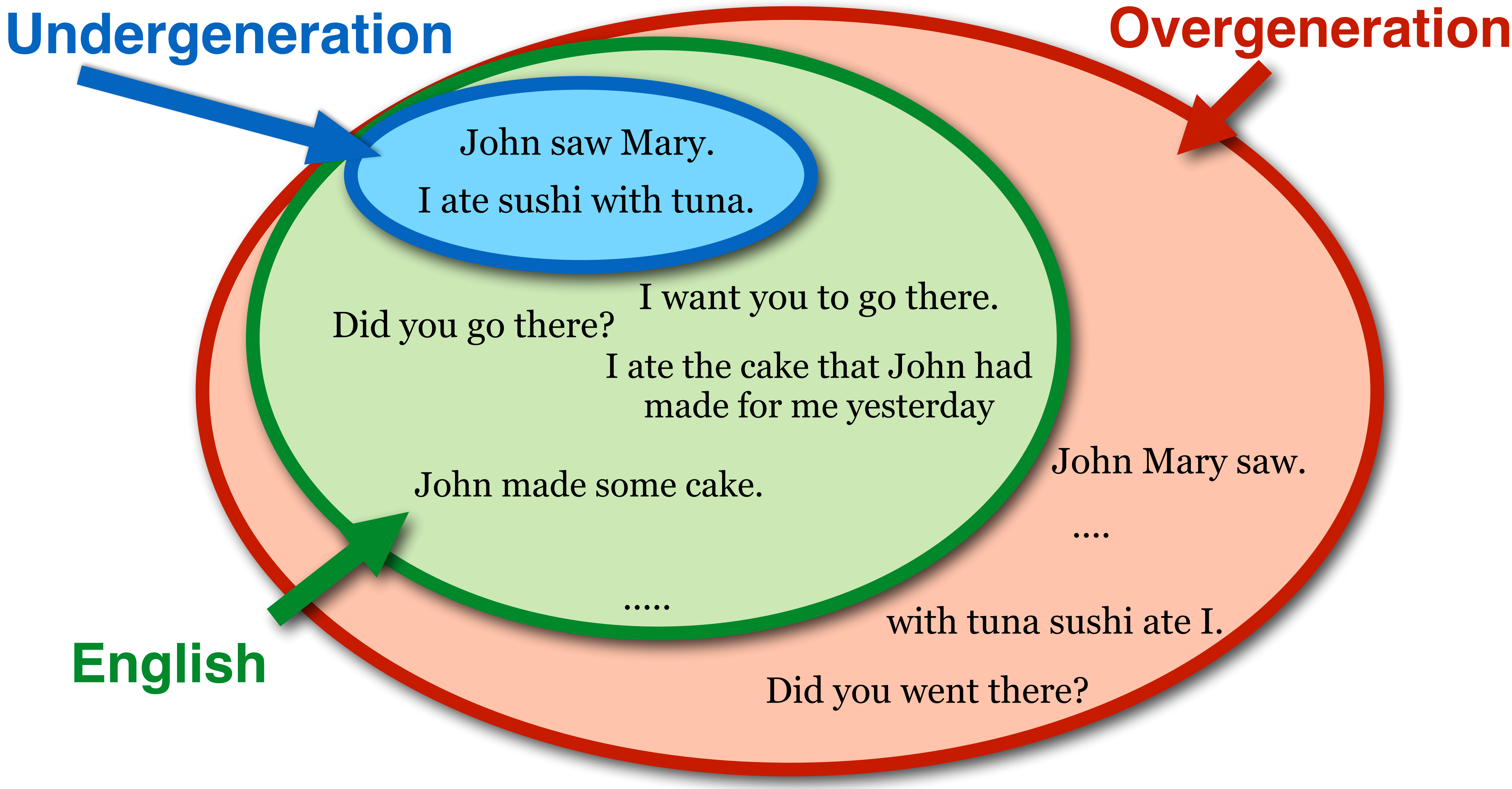
Those formalisms have appropriate strong generative capacity

Practical applications (Syntactic/Semantic Parsing):

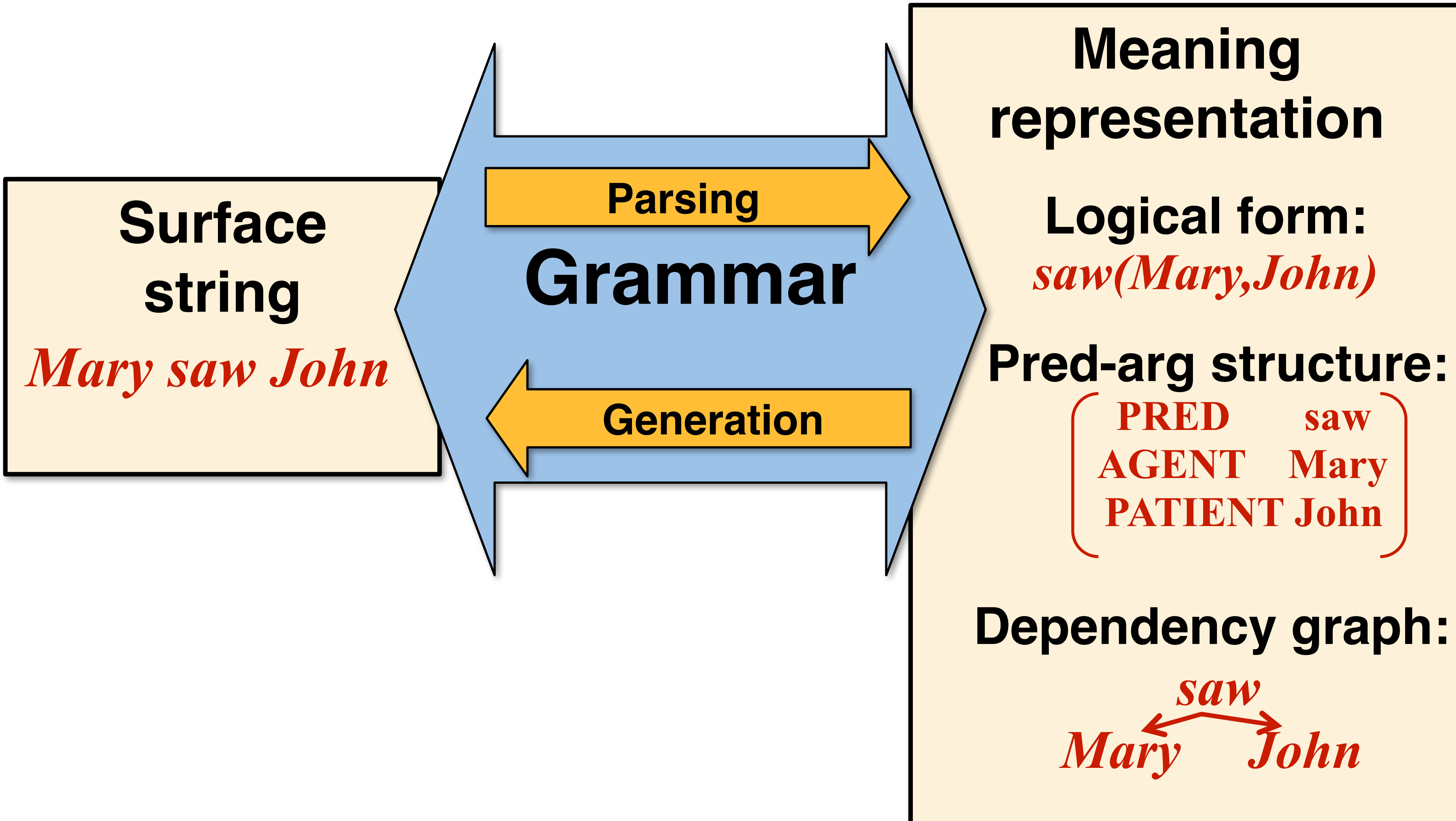
Can we identify the grammatical structure of sentences?

Can we translate sentences to appropriate meaning representations?

Can we define a program that generates all English sentences?



Syntax as an interface to semantics



Grammar formalisms

Formalisms provide a formal **language** in which linguistic theories can be expressed and implemented

Formalisms define **elementary objects**
(trees, strings, feature structures)
and **recursive operations** which generate complex objects from simple objects.

Different formalisms may impose different **constraints**
(e.g. on the kinds of dependencies they can capture)

What makes a formalism "expressive"?

"Expressive" formalisms are richer than context-free grammars.

Different formalisms use different mechanisms, data structures and operations to go beyond CFGs

Examples of expressive grammar formalisms

Tree-adjoining Grammar (TAG):
Fragments of phrase-structure trees

Combinatory Categorial Grammar (CCG):
Syntactic categories paired with meaning representations

Lexical-functional Grammar (LFG):
Annotated phrase-structure trees (c-structure)
linked to feature structures (f-structure)

Head-Driven Phrase Structure Grammar(HPSG):
Complex feature structures (Attribute-value matrices)

Lecture 18:
Expressive Grammars

Part 2: Why go beyond CFGs?

The dependencies so far:

Arguments:

Verbs take arguments: subject, object, complements, ...

Heads subcategorize for their arguments

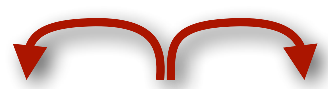
Adjuncts/Modifiers:

Adjectives modify nouns, adverbs modify VPs or adjectives,

PPs modify NPs or VPs

Modifiers subcategorize for the head

Typically, these are *local* dependencies: they can be expressed *within individual CFG rules*

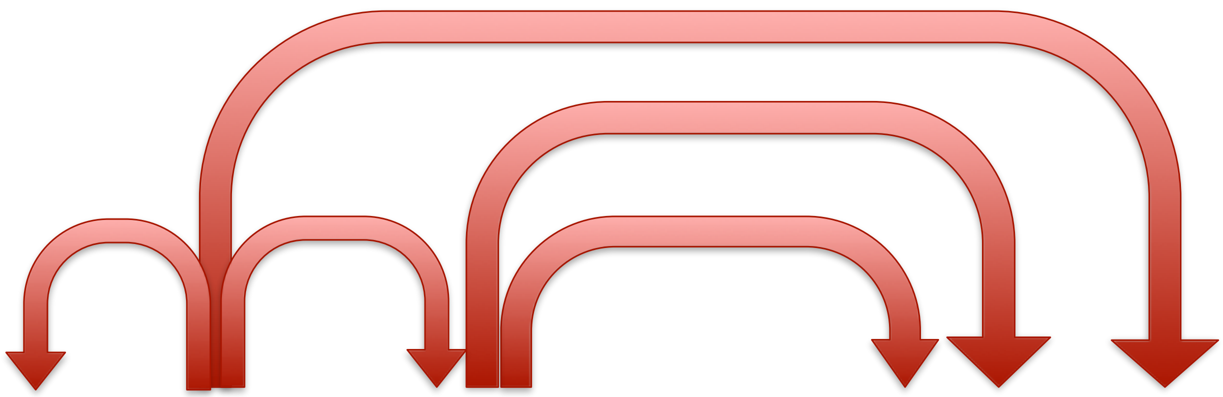


VP → Adv Verb NP

Context-free grammars

CFGs capture only **nested** dependencies

- The dependency graph is a **tree**
- The dependencies **do not cross**



German: center embedding

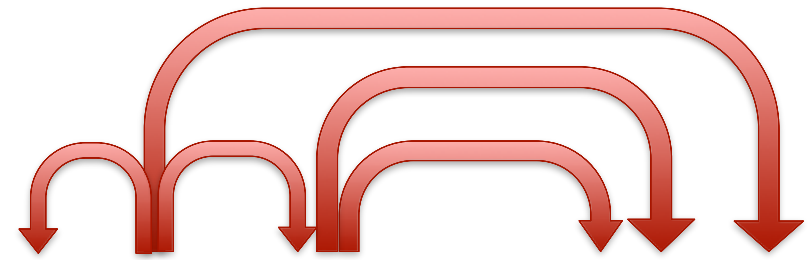
...daß ich [Hans schwimmen] sah
...that I Hans swim saw
...that I saw [Hans swim]

...daß ich [Maria [Hans schwimmen] helfen] sah
...that I Maria Hans swim help saw
...that I saw [Mary help [Hans swim]]

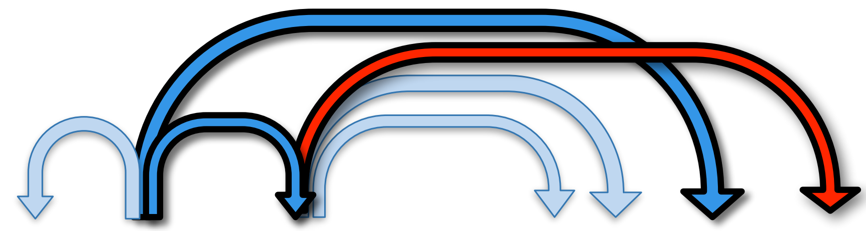
...daß ich [Anna [Maria [Hans schwimmen] helfen] lassen] sah
...that I Anna Maria Hans swim help let saw
...that I saw [Anna let [Mary help [Hans swim]]]

Dependency structures in general

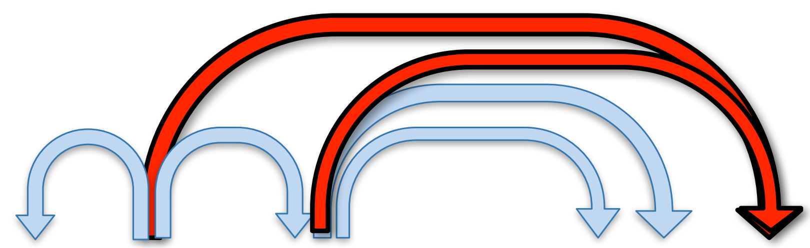
Nested (projective) dependency trees (CFGs)



Non-projective dependency trees

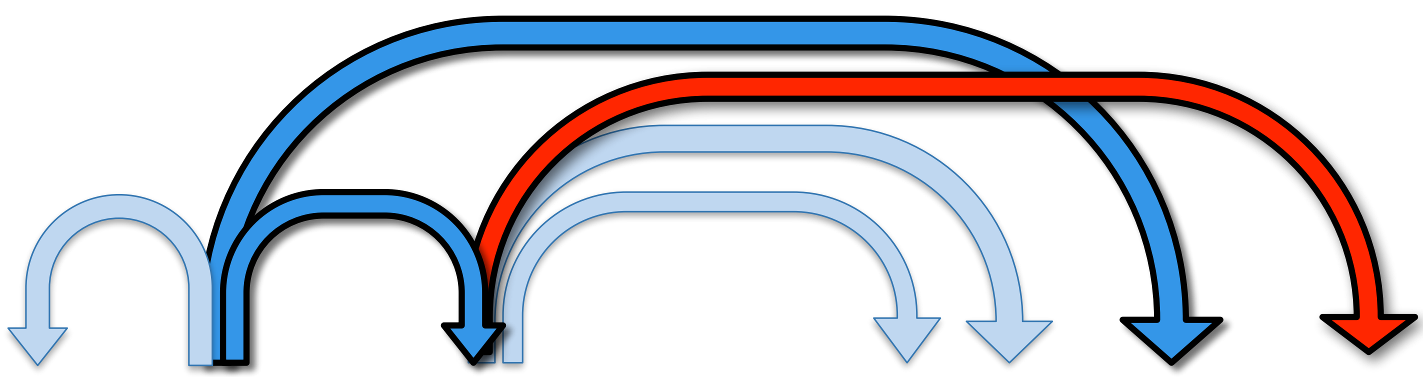


Non-local dependency graphs



Beyond CFGs:
Nonprojective dependencies

Dependencies form a **tree with crossing branches**



Dutch: Cross-Serial Dependencies

...dat ik Hans zag zwemmen
...that I Hans saw swim
...that I saw [Hans swim]

...dat ik Maria Hans zag helpen zwemmen
...that I Maria Hans saw help swim
...that I saw [Mary help [Hans swim]]

...dat ik Anna Maria Hans zag laten helpen zwemmen
...that I Anna Maria Hans saw let help swim
...that I saw [Anna let [Mary help [Hans swim]]]

Such cross-serial dependencies require
mildly context-sensitive grammars

Other crossing (non-projective) dependencies

(Non-local) scrambling: In a sentence with multiple verbs, the argument of a verb appears in a different clause from that which contains the verb (arises in languages with freer word order than English)

Die Pizza *hat Klaus versprochen zu* ***bringen***
The pizza has Klaus promised to bring
Klaus has promised to bring the pizza

Extraposition: Here, a modifier of the subject NP is moved to the end of the sentence

The ***guy*** *is coming* ***who is wearing a hat***
Compare with the non-extraposed variant
The ***[guy [who is wearing a hat]]*** *is coming*

Topicalization: Here, the argument of the embedded verb is moved to the front of the sentence.

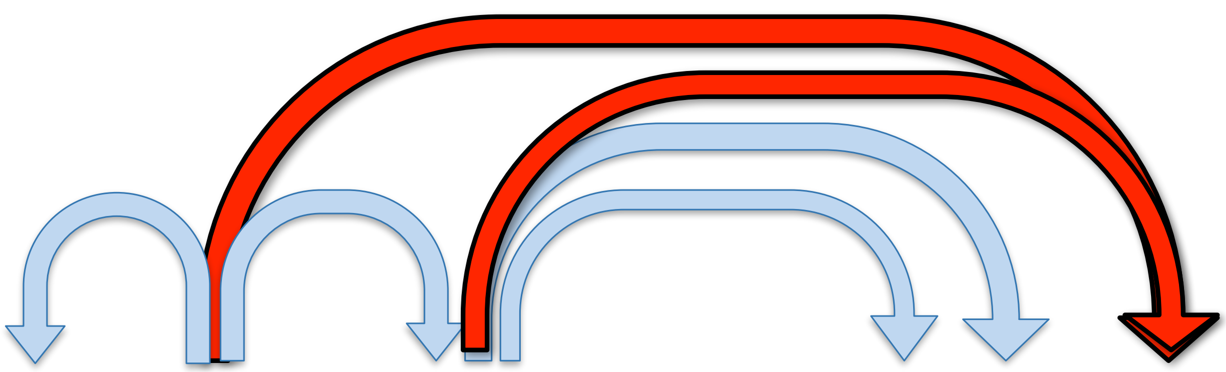
Cheeseburgers*, I [thought [he* ***likes****]]*

Beyond CFGs: Nonlocal dependencies

Dependencies form a **DAG**
(a node may have **multiple incoming edges**)

Arise in the following constructions:

- **Control** *(**He** has **promised** me to **go**)*, **raising** (**He seems** to **go**)
- **Wh-movement** *(the **man** who you **saw** yesterday **is** here again)*,
- **Non-constituent** coordination
  (right-node raising, gapping, argument-cluster coordination)



Wh-Extraction (e.g. in English)

Relative clauses:

the sushi that [*you told me* [*John saw* [*Mary eat*]]]’

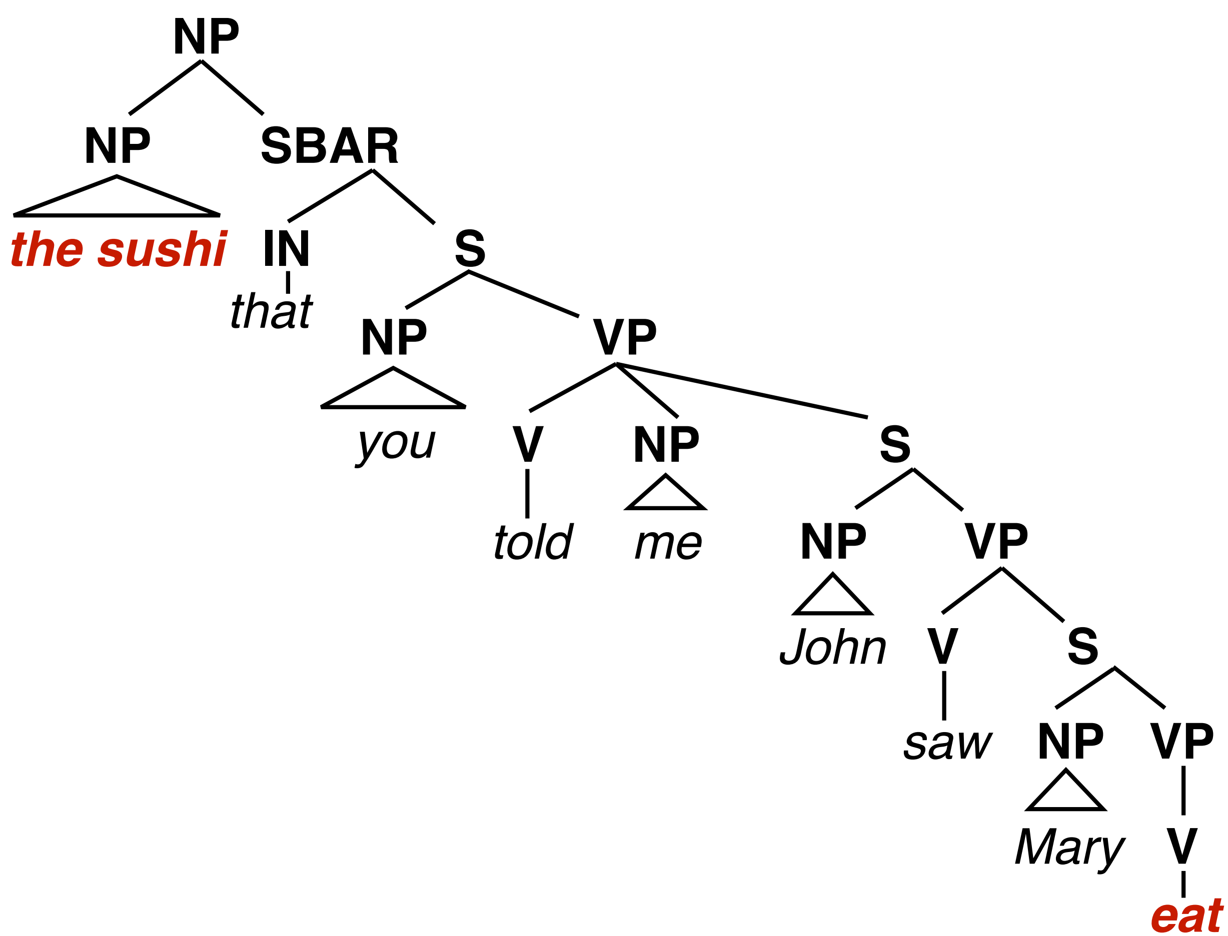
Wh-Questions:

‘*what* [*did you tell me* [*John saw* [*Mary eat*]]]?’

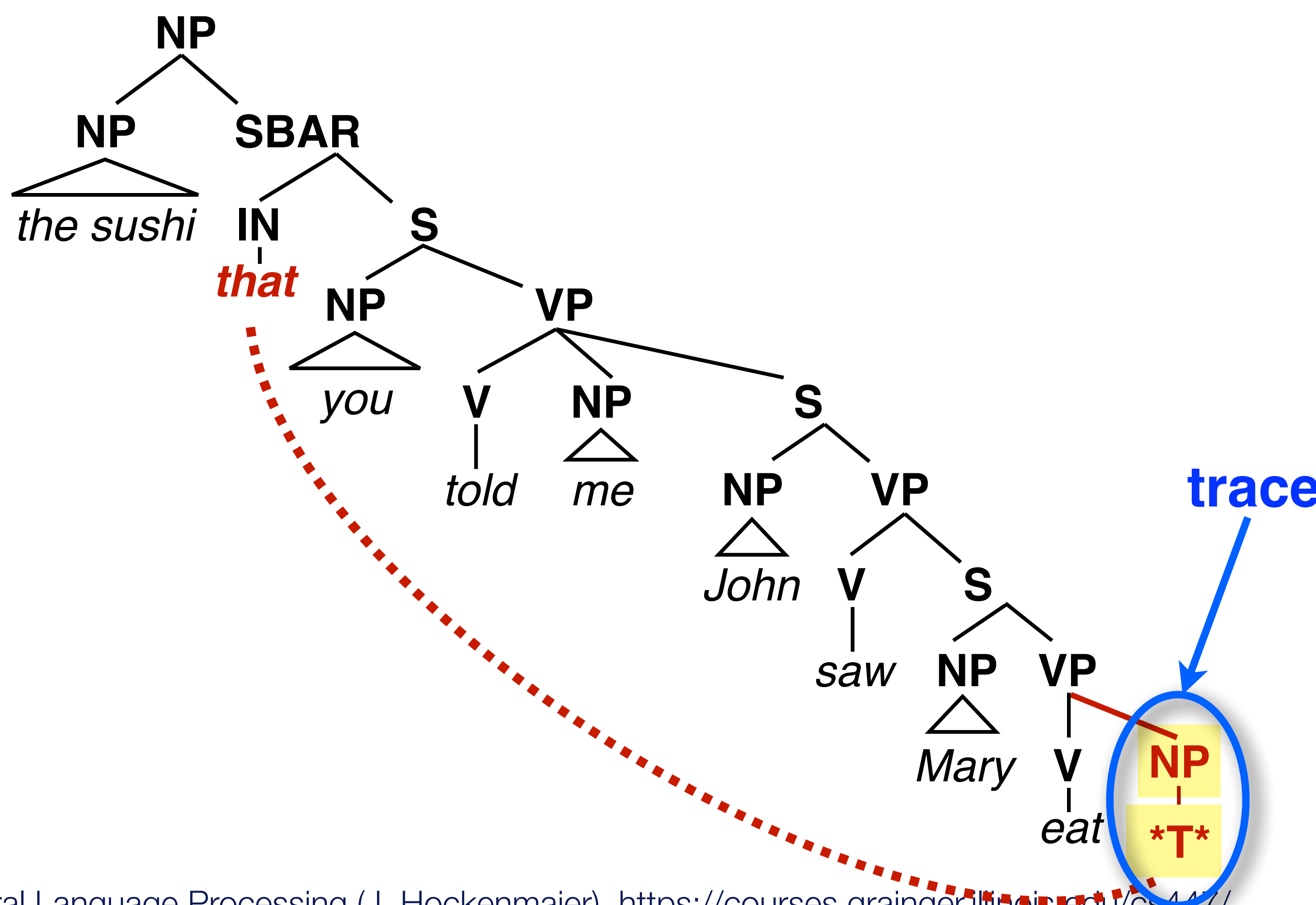
Wh-questions (what, who, …) and relative clauses contain so-called *unbounded* nonlocal dependencies because the verb that subcategorizes for the moved NP may be arbitrarily deeply embedded in the tree

Linguists call this phenomenon **wh-extraction** (wh-movement).

As a phrase structure tree:



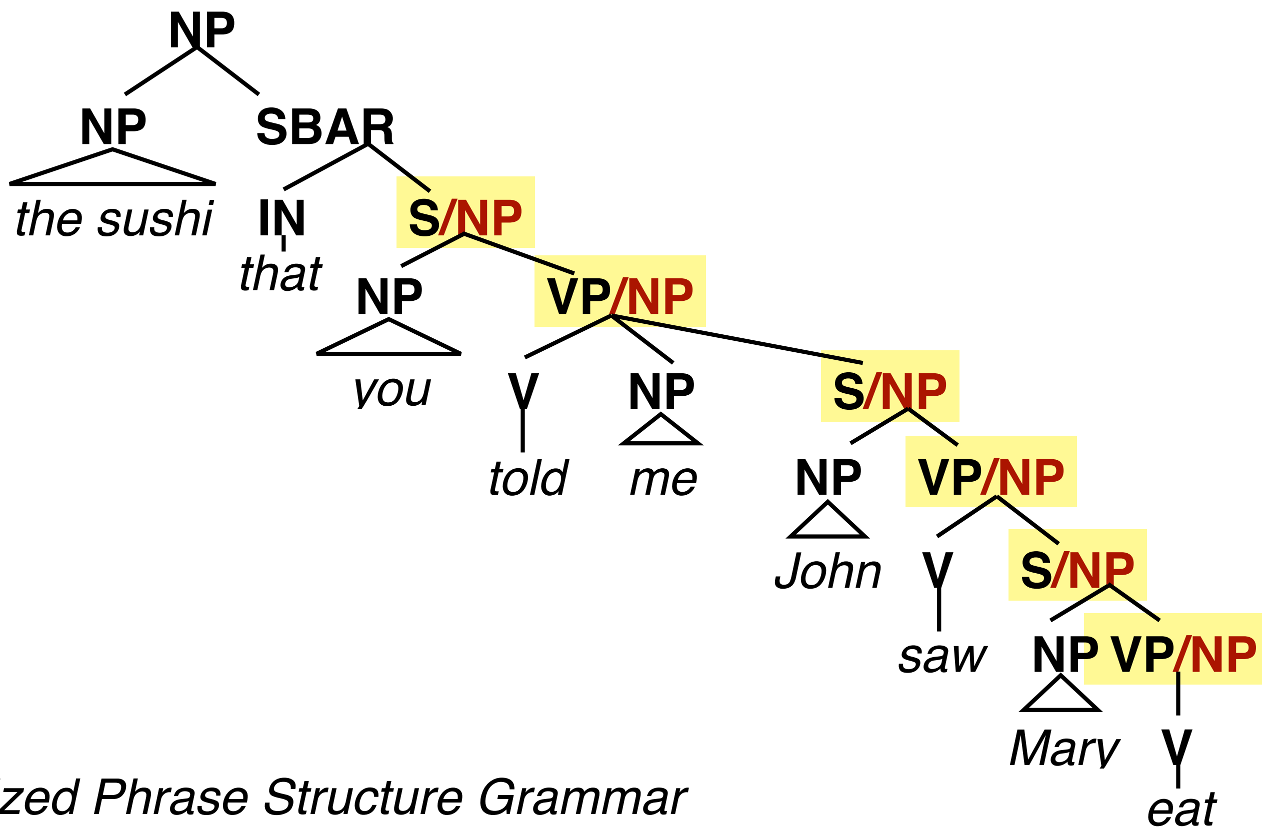
The trace analysis of wh-extraction



Slash categories for wh-extraction

Because only one element can be extracted,
we can use **slash categories.**

This is still a CFG: the set of nonterminals is finite.



Generalized Phrase Structure Grammar
(GPSG), Gazdar et al. (1985)

Part 3: 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 Languages

Mildly context-sensitive grammars

Contain all context-free grammars/languages

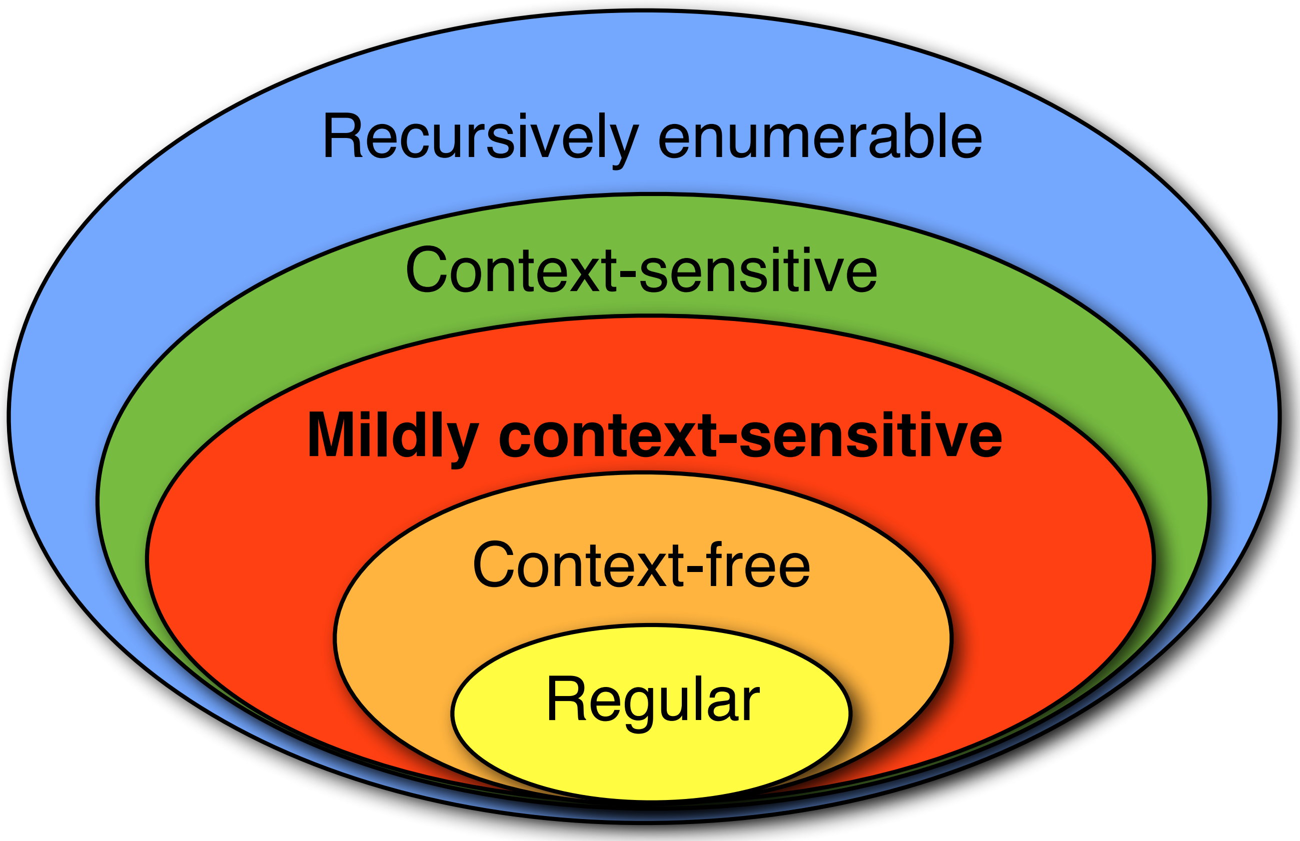
Can be **parsed in polynomial time** (TAG/CCG: $O(n^6)$)

(*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 $\mathbf{w} \in \{a, b, c\}^*$ that contain equal numbers of as, bs and cs

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

The Chomsky Hierarchy

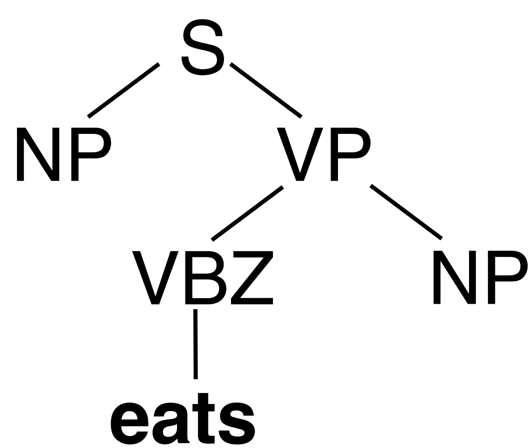


Extended domain of locality

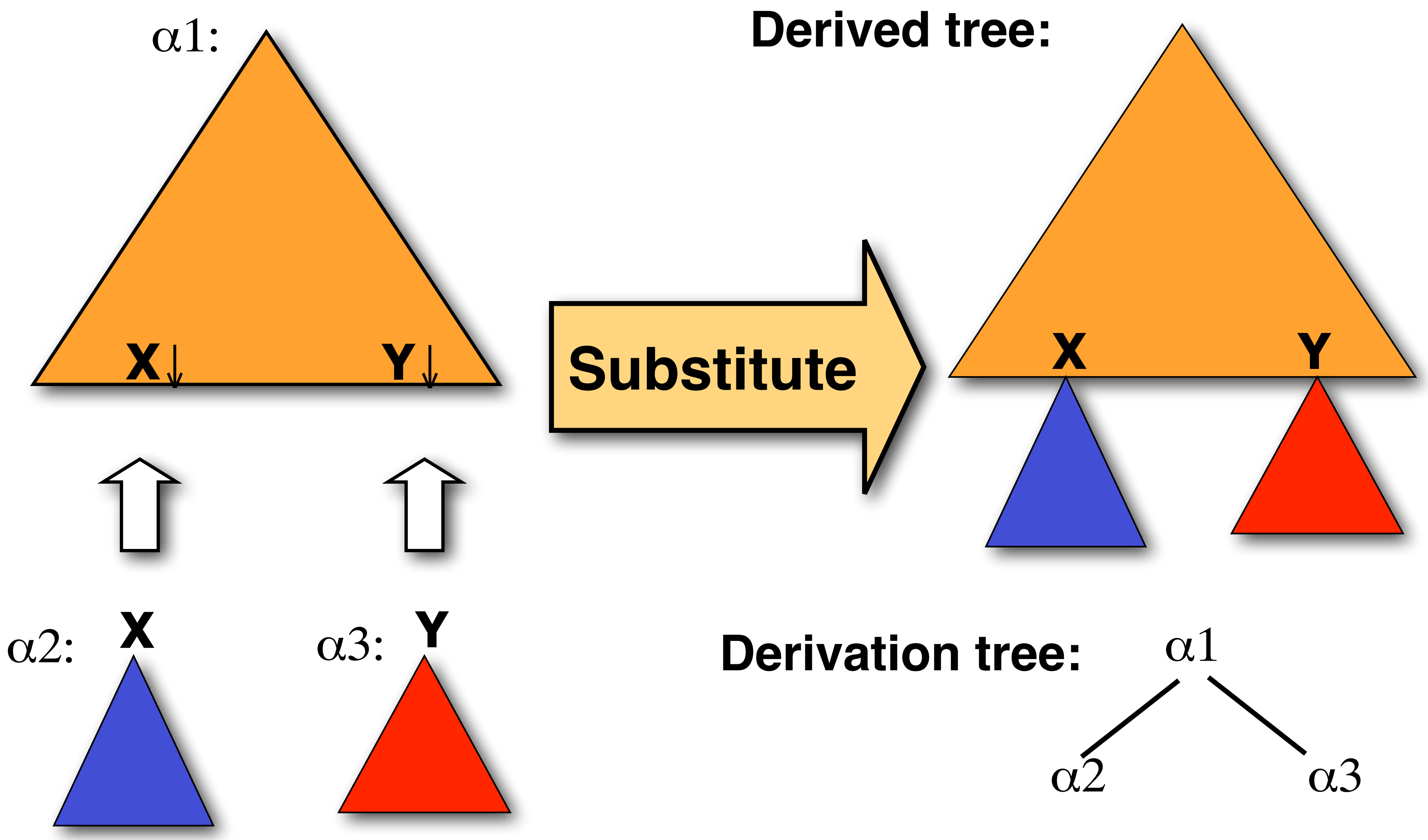
We want to capture **all arguments of a word** in a **single elementary object**.

We also want to retain certain syntactic structures (e.g. VPs).

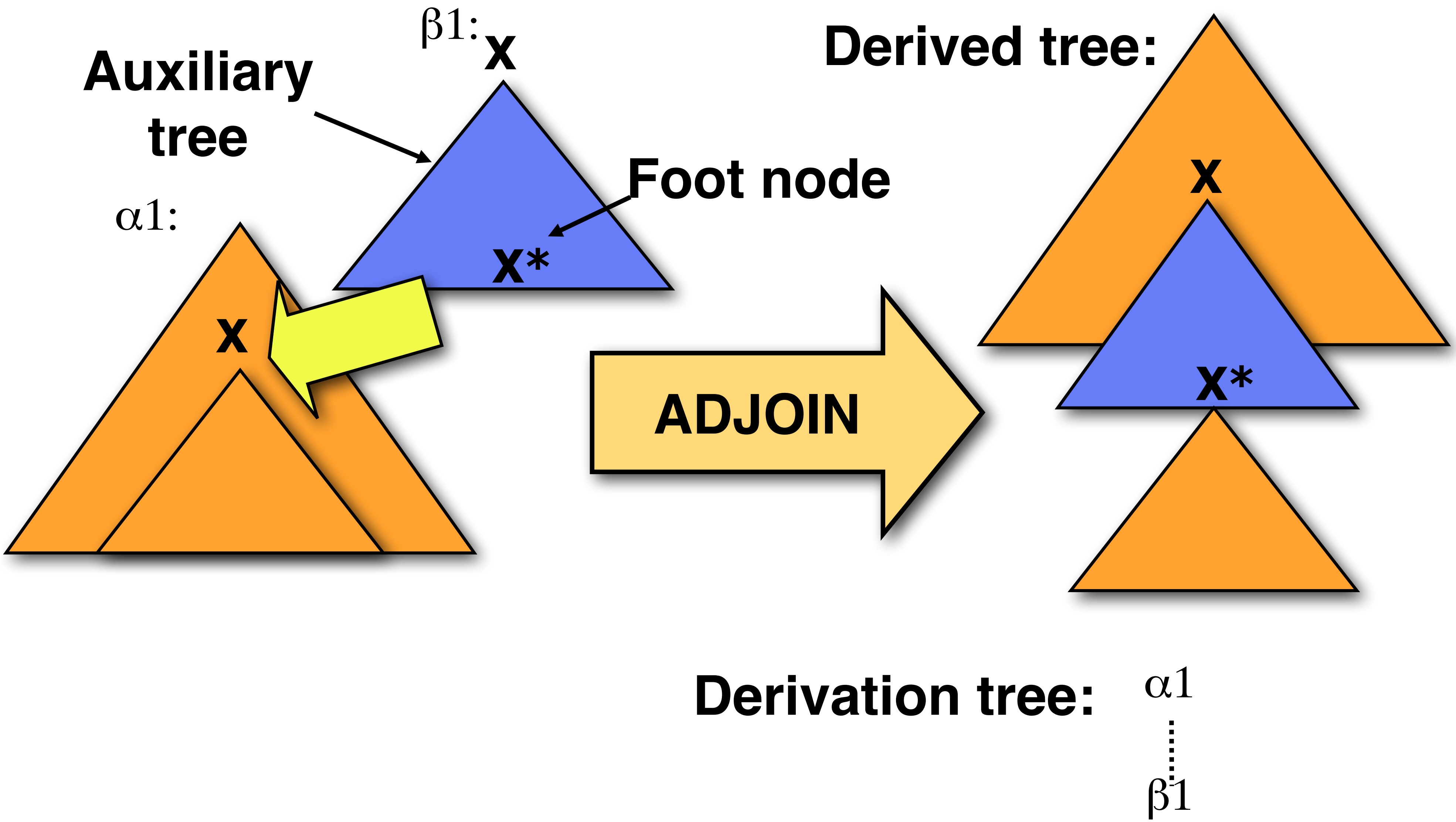
Our elementary objects are **tree fragments:**



TAG substitution (arguments)

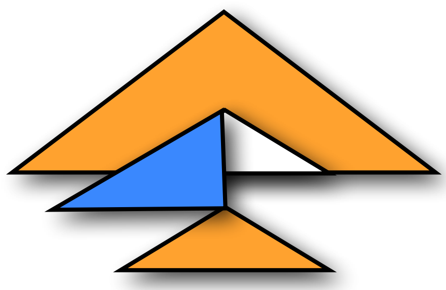


TAG adjunction

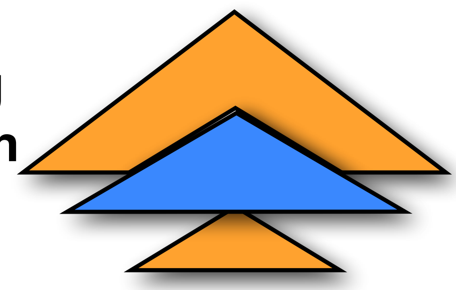


The effect of adjunction

TIG:
sister
adjunction



TAG:
wrapping
adjunction



No adjunction: TSG (Tree substitution grammar)

TSG is context-free

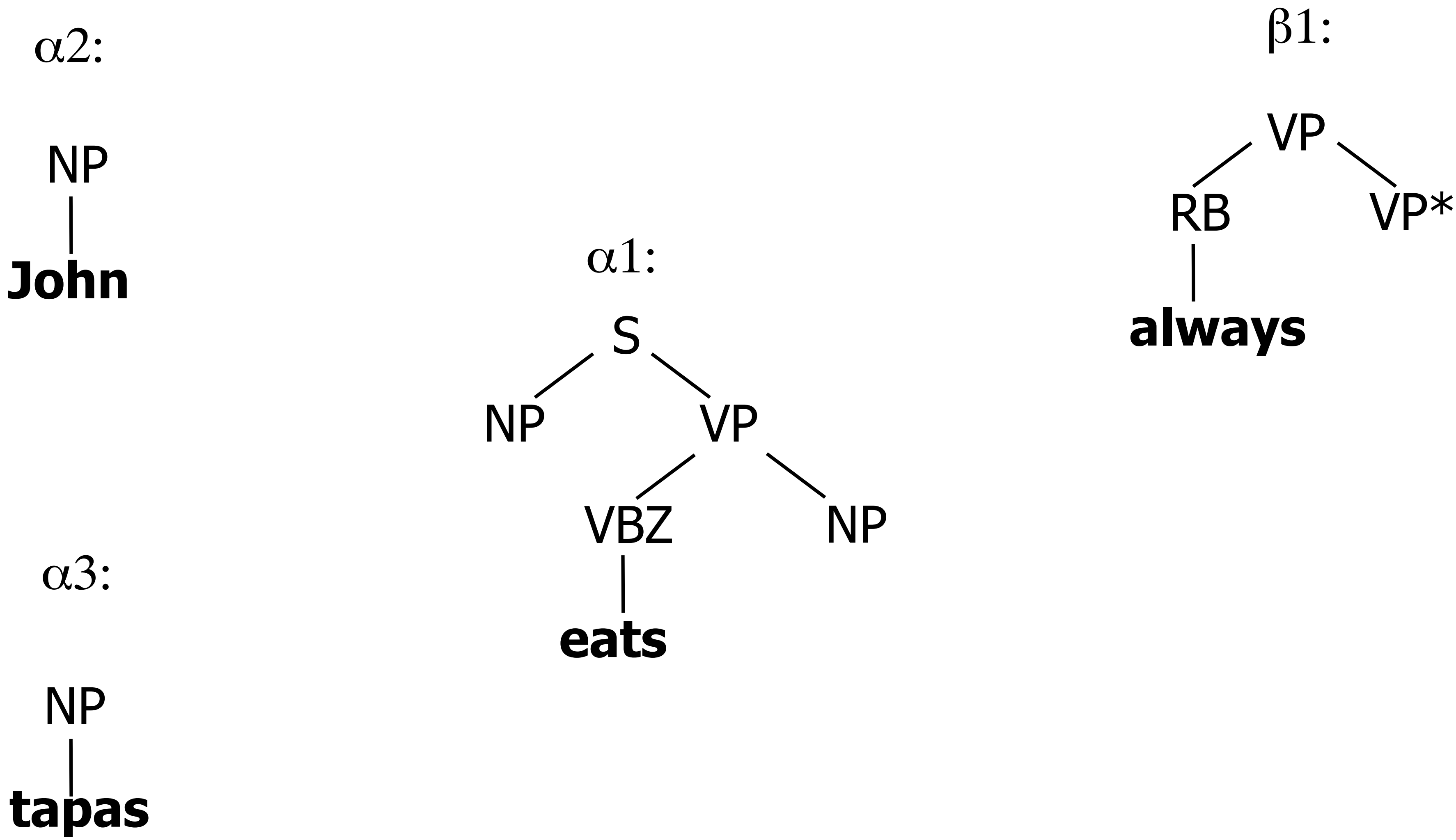
Sister adjunction: TIG (Tree insertion grammar)

TIG is also context-free, but has a linguistically more adequate treatment of modifiers

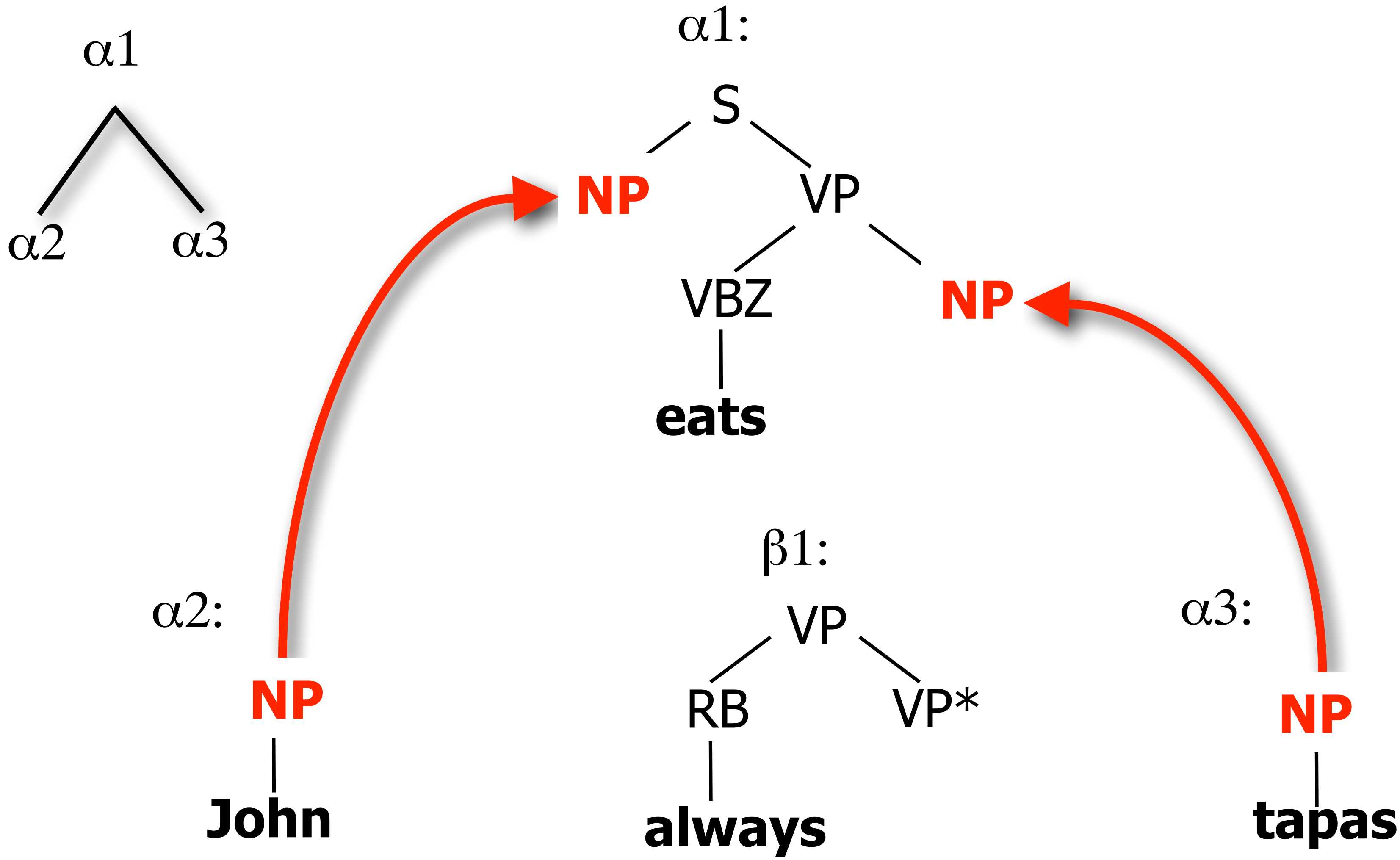
Wrapping adjunction: TAG (Tree-adjoining grammar)

TAG is mildy context-sensitive

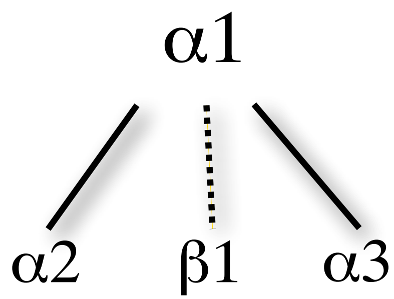
A small TAG lexicon

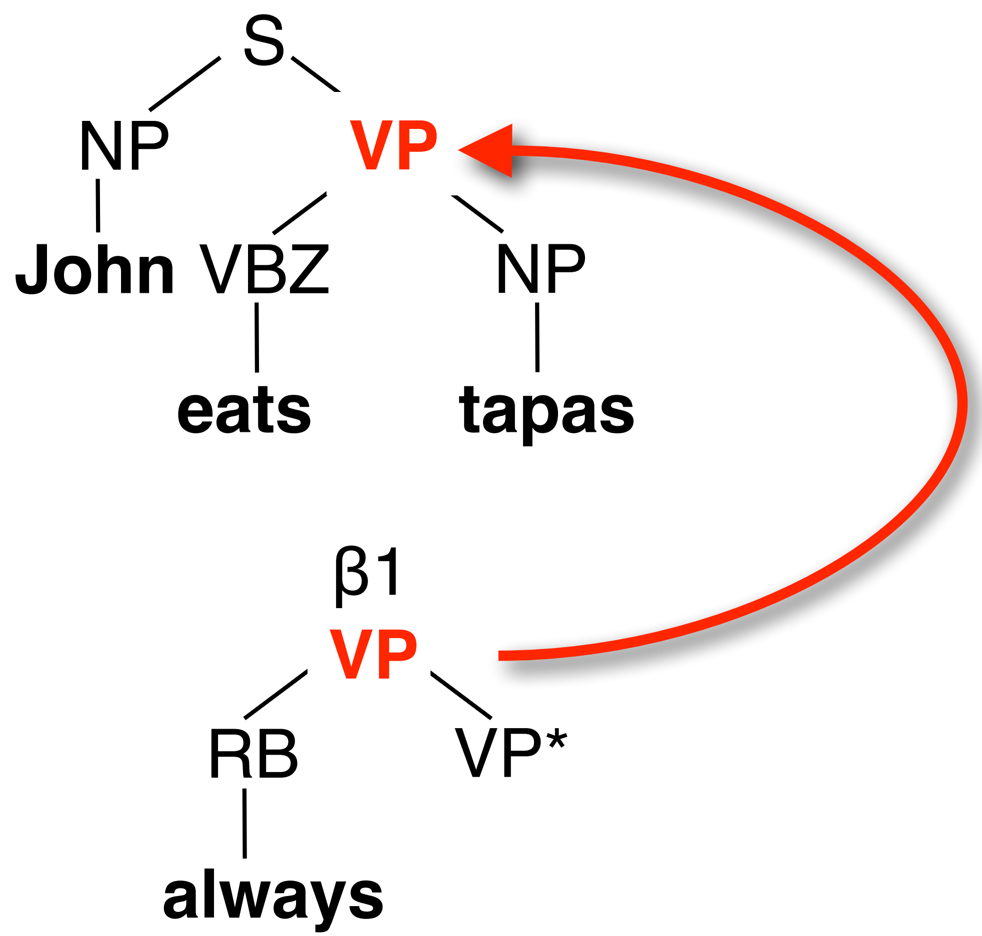


A TAG derivation

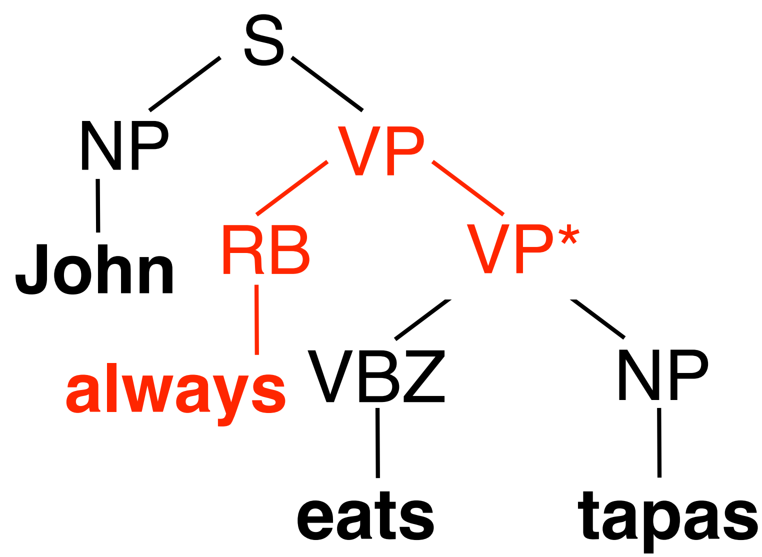


A TAG derivation





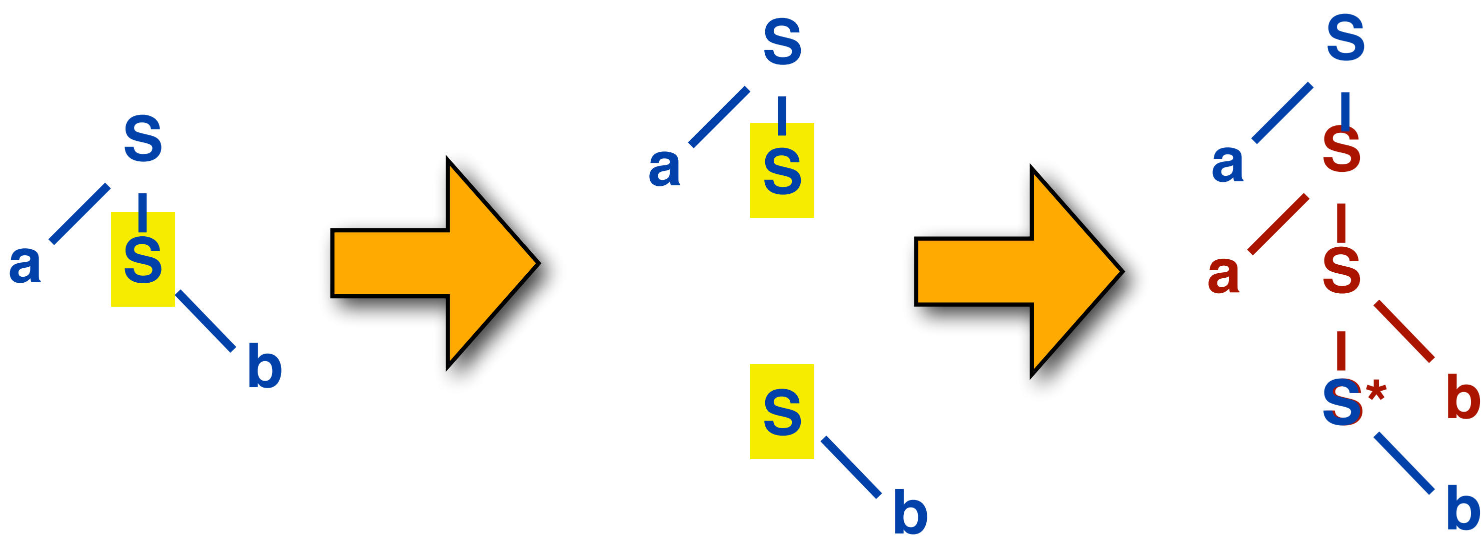
A TAG derivation



$a^n b^n$: Cross-serial dependencies

Elementary trees:

Deriving aabb



Part 4: (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**

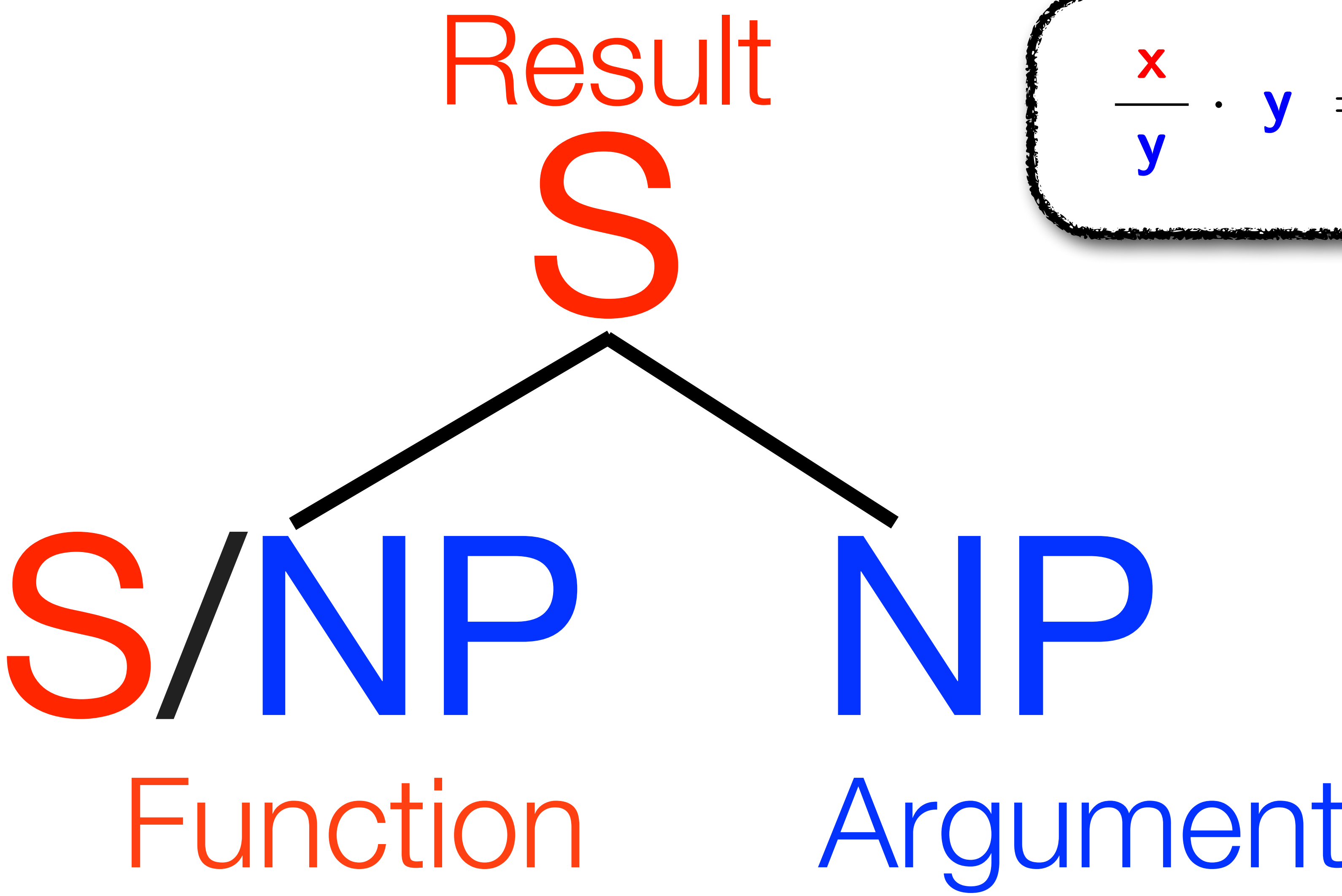
S, NP, PP

All other CCG categories are **functions**:

S / NP

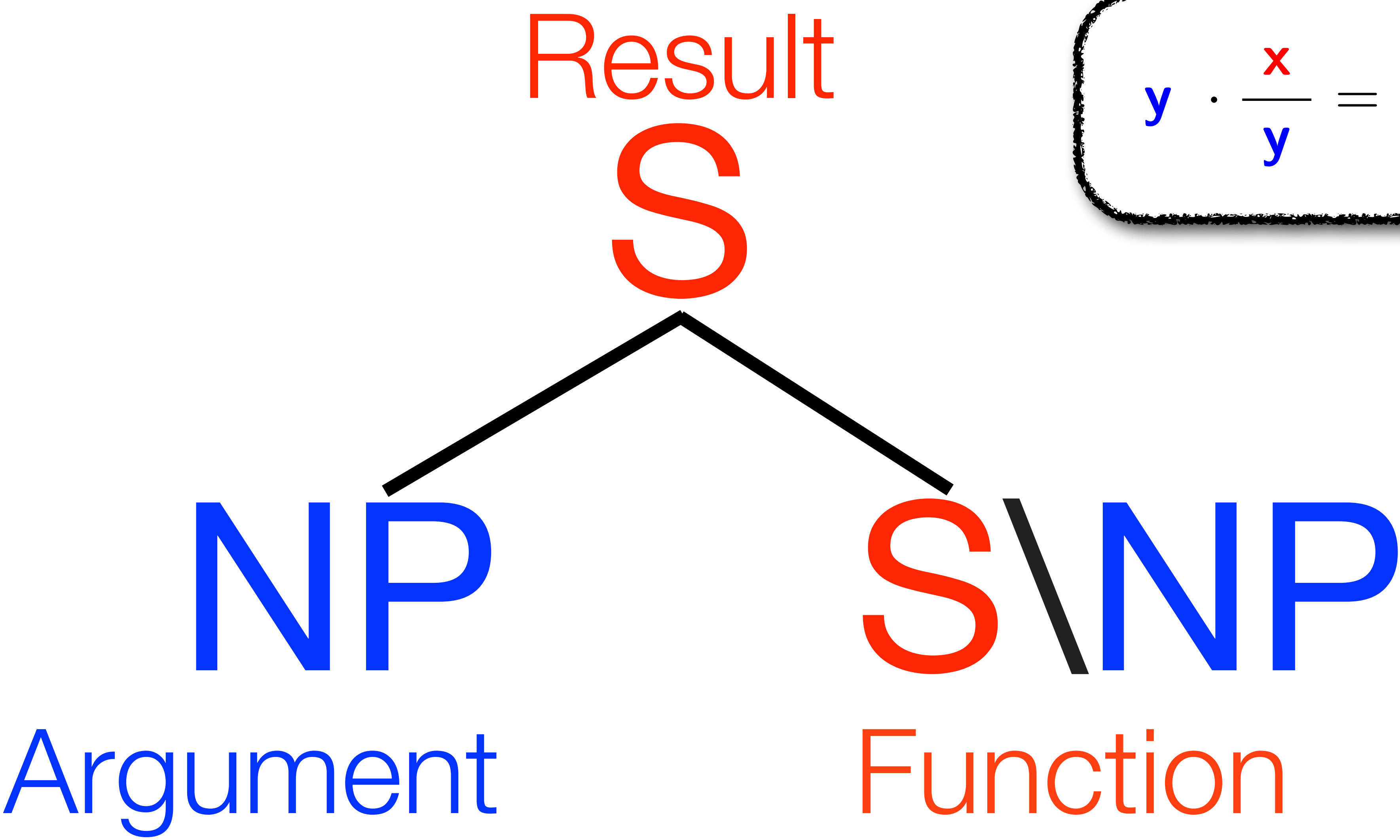
Result Dir. Argument

Rules: Function application



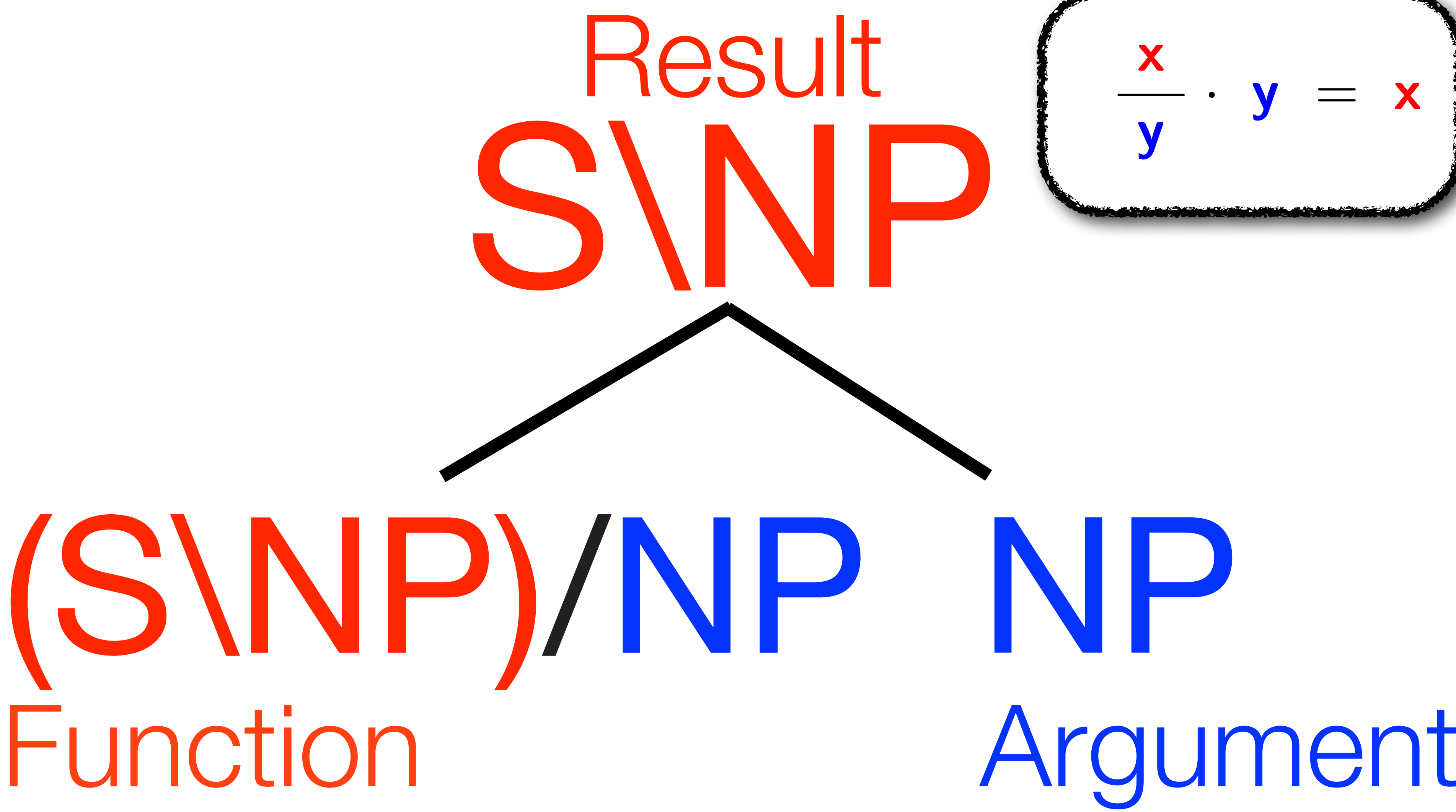
$$\frac{x}{y} \cdot y = x$$

Rules: Function application



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

Rules: Function application



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

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

John eats tapas

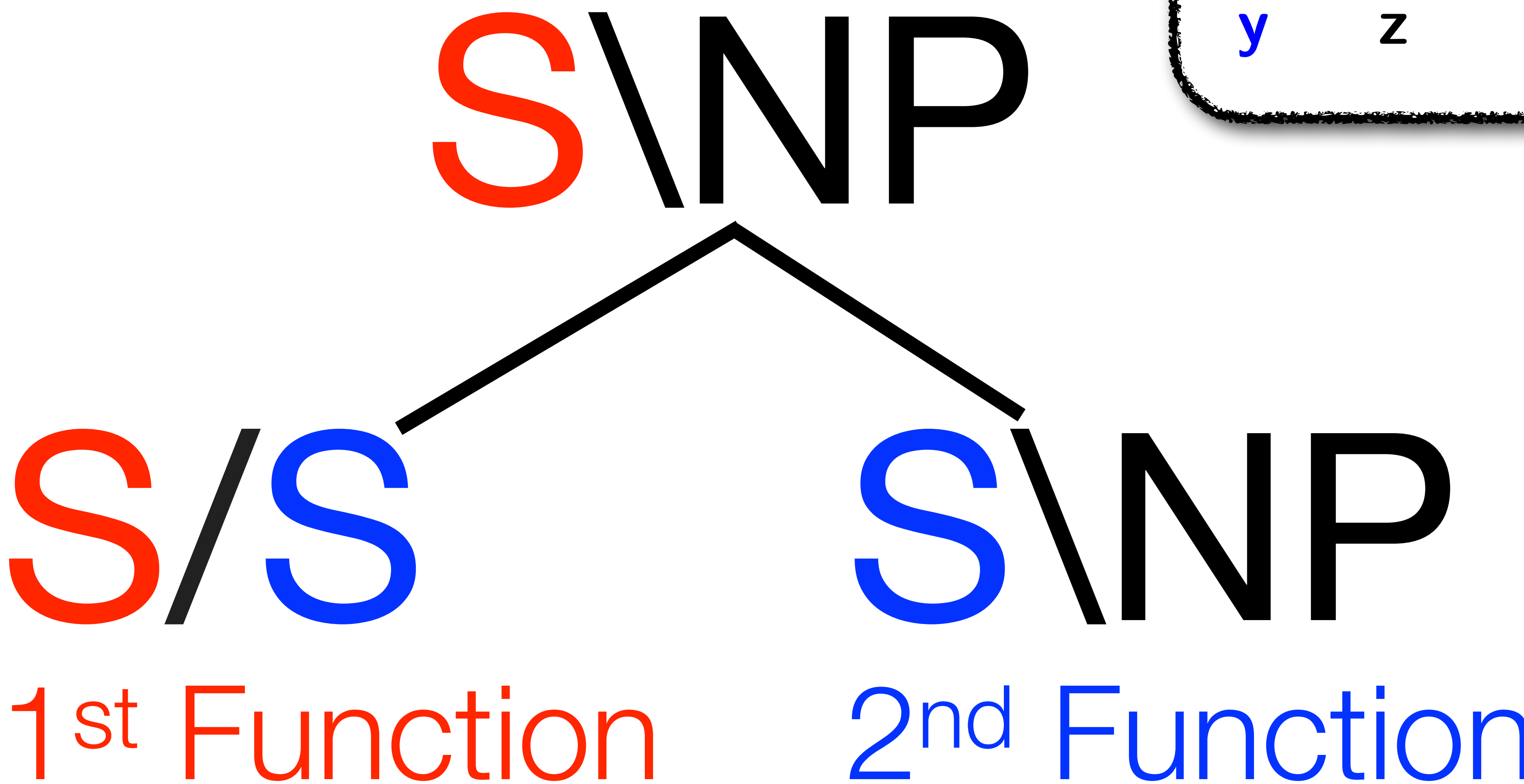
$\text{NP} \quad (\text{S}\backslash\text{NP})/\text{NP} \quad \text{NP}$

$\text{S}\backslash\text{NP}$ (>)

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 → T/(T\X)

Turns an argument into a function.

NP → S/(S\NP) (subject)

NP → (S\NP)\((S\NP)/NP) (object)

Harmonic composition: X/Y Y/Z → X/Z

Composes two functions (complex categories), same slashes

(S\NP)/PP PP/NP → (S\NP)/NP

S/(S\NP) (S\NP)/NP → S/NP

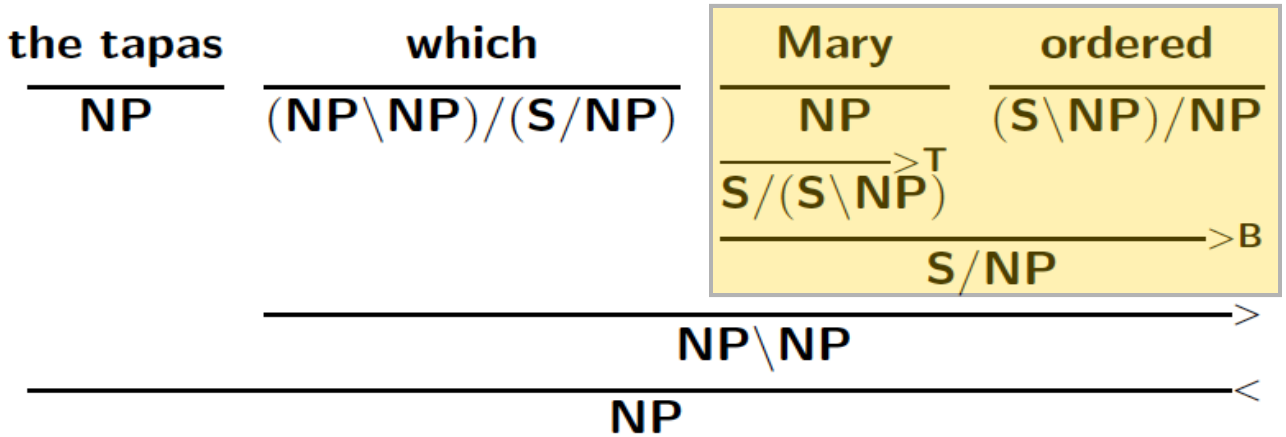
Crossing composition: X/Y Y\Z → X\Z

Composes two functions (complex categories), different slashes

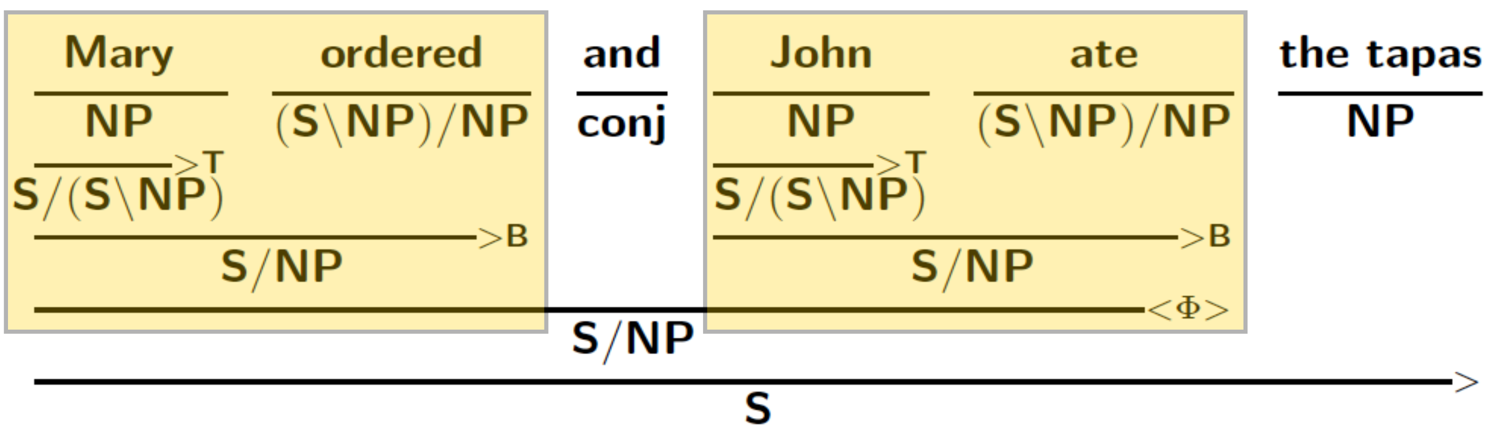
(S\NP)/S S\NP → (S\NP)\NP

Type-raising and composition

Wh-movement (relative clause):



Right-node raising:



Part 5: 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

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

This generates ungrammatical sentences like
“these student eats a cakes”

We need to capture (number/person) agreement

Refining the nonterminals

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

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).

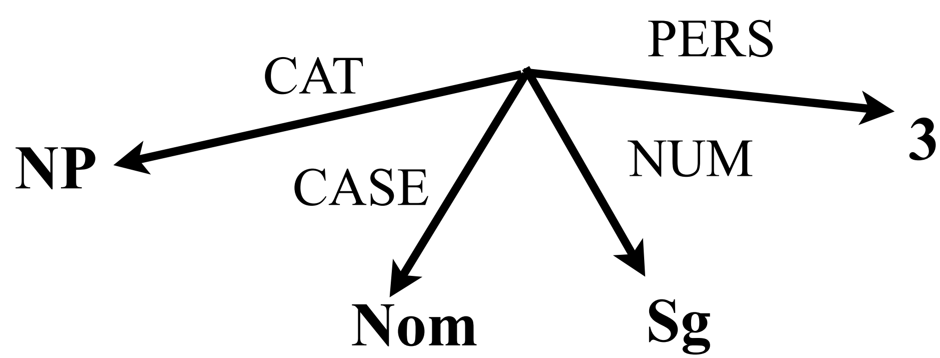
We often represent feature structures as attribute value matrices (AVMs)

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.

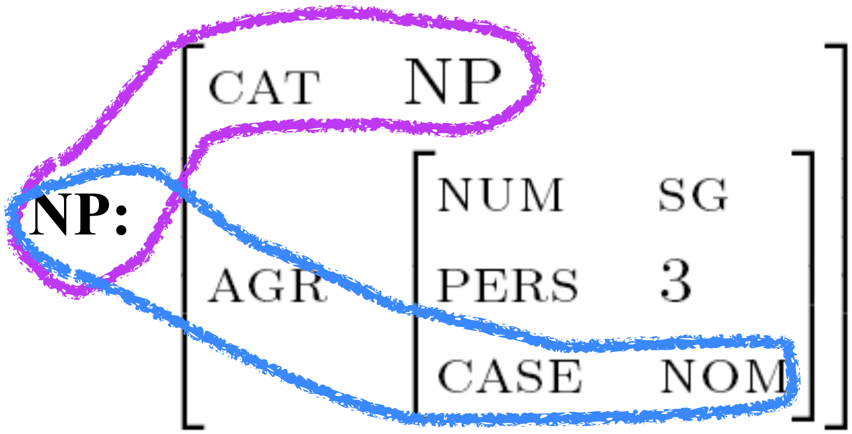
Only atomic values:

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

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



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:

$$\mathbf{A_0 \rightarrow A_1 \ldots A_n}$$

$$\{\text{set of constraints}\}$$

There are two kinds of constraints:

Unification constraints:

$$\langle \mathbf{A_i} \text{ feature-path} \rangle = \langle \mathbf{A_j} \text{ feature-path} \rangle$$

Value constraints:

$$\langle \mathbf{A_i} \text{ feature-path} \rangle = \text{atomic value}$$

A grammar with feature structures

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

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

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

With complex feature structures

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

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:

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

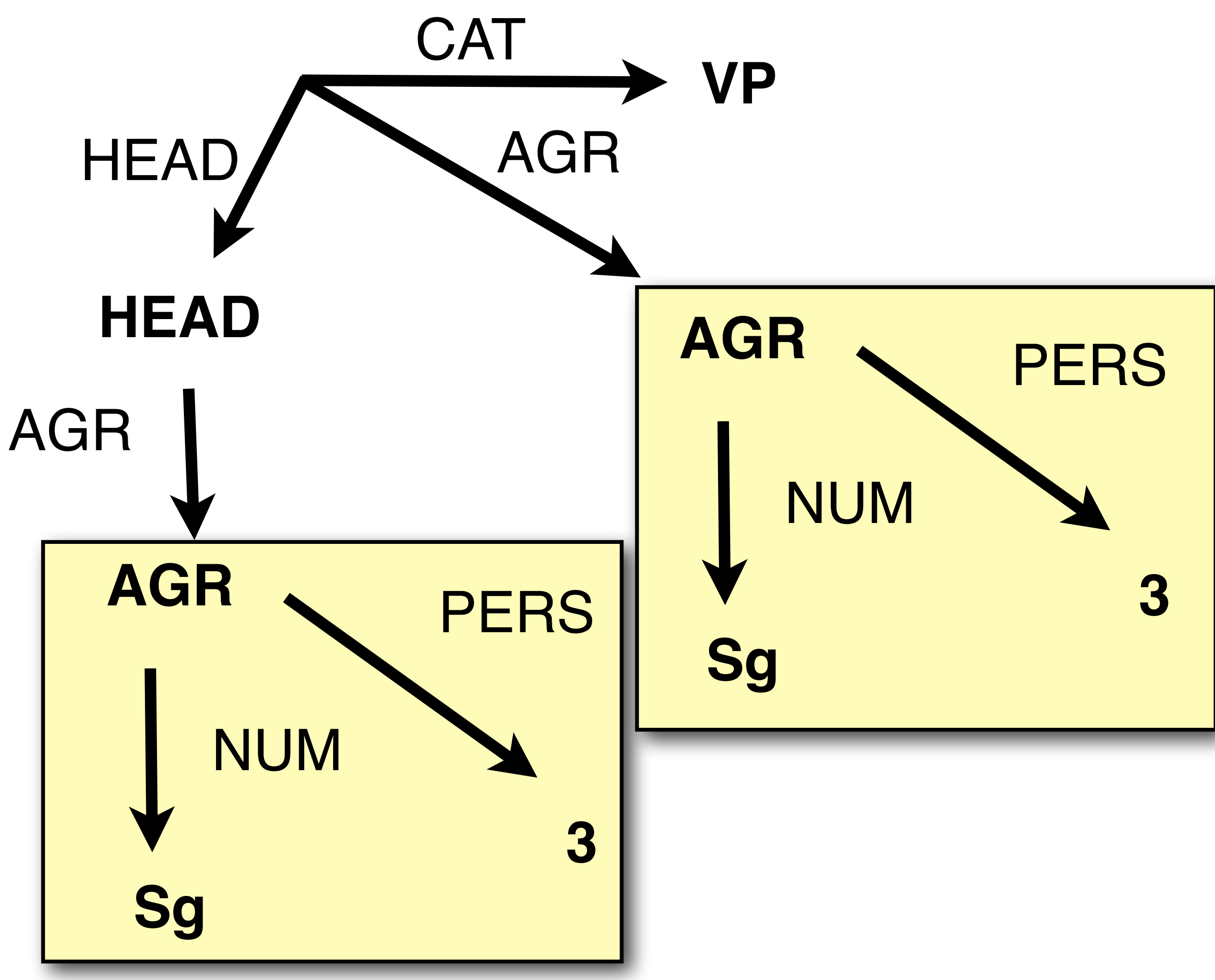
Re-entrancies

What we *really* want to say is that the agreement feature of the head is *identical* to that of the VP itself.

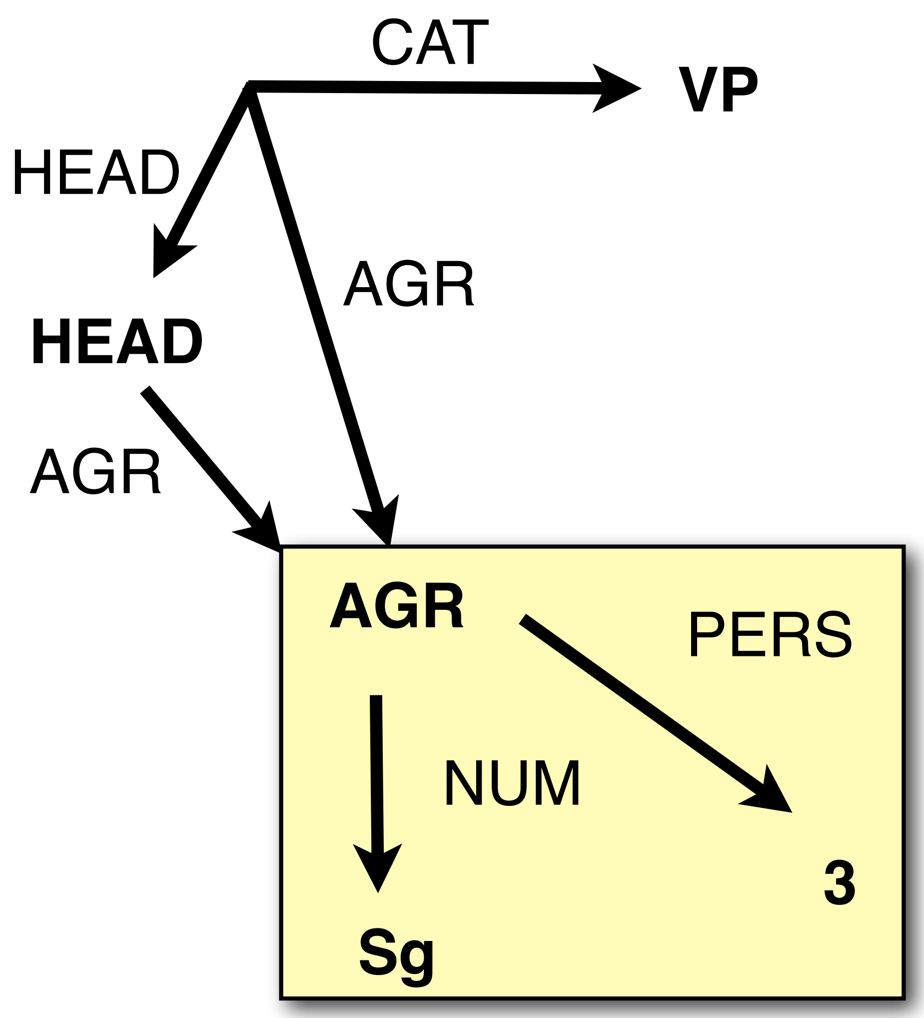
This corresponds to a re-entrancy in the FS (indicated via coindexation $\boxed{1}$)

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

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:

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