CSE 490 U
Natural Language Processing
Spring 2016

Dependency Parsing
And Other Grammar Formalisms

Yejin Choi - University of Washington

Dependency Grammar

For each word, find one parent.

Child Parent

A child is dependent on the parent.

- A child is an argument of the parent.
- A child modifies the parent.

I shot an elephant

For each word, find one parent.

Child Parent

A child is dependent on the parent.

- A child is an argument of the parent.
- A child modifies the parent.

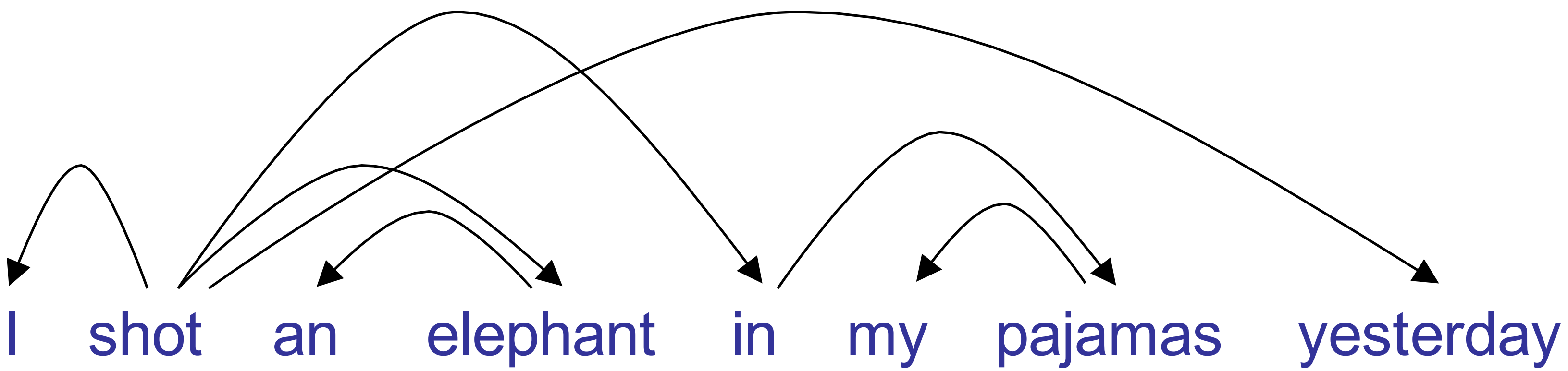
I shot an elephant in my pajamas

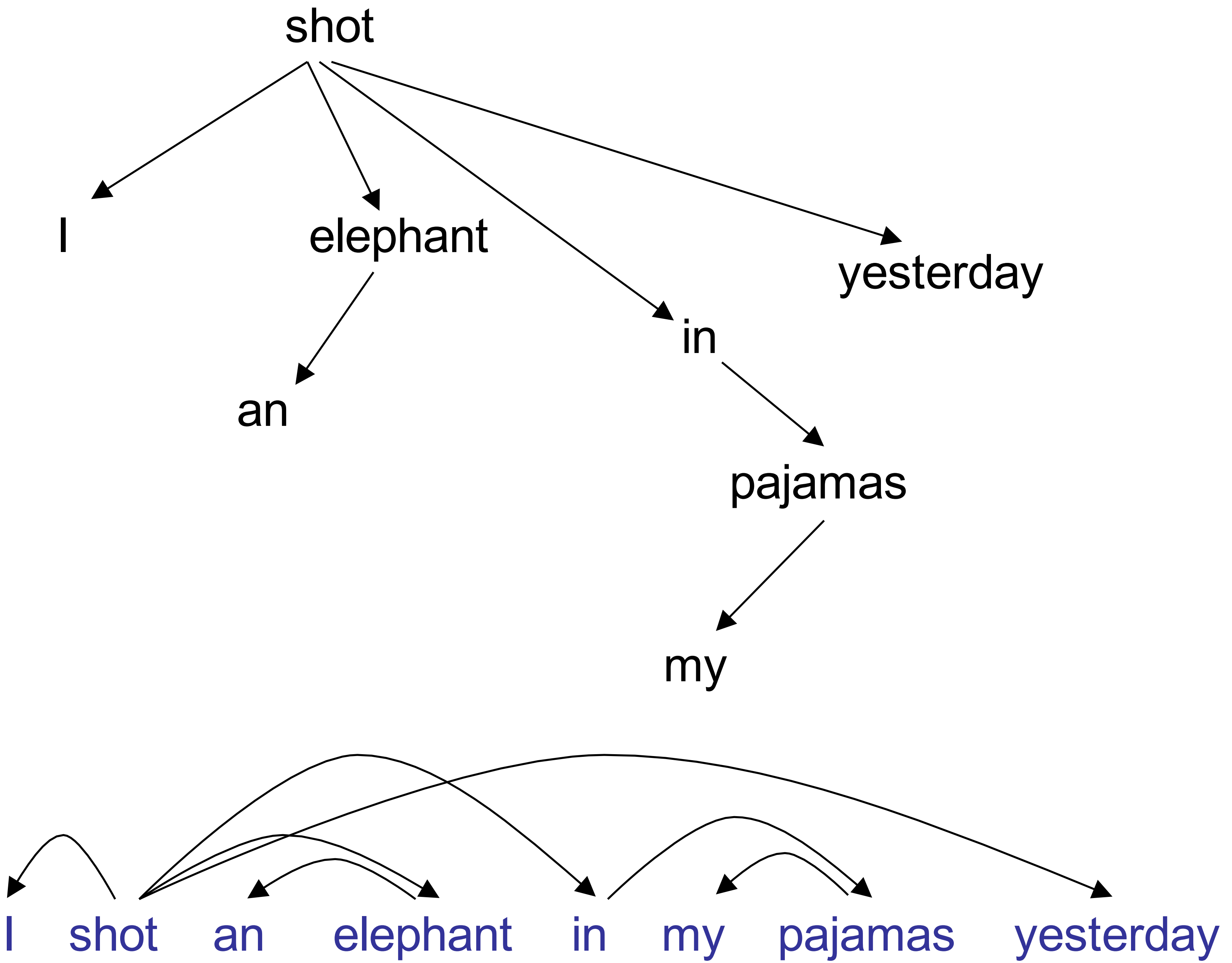
For each word, find one parent.

Child Parent

A child is dependent on the parent.

- A child is an argument of the parent.
- A child modifies the parent.

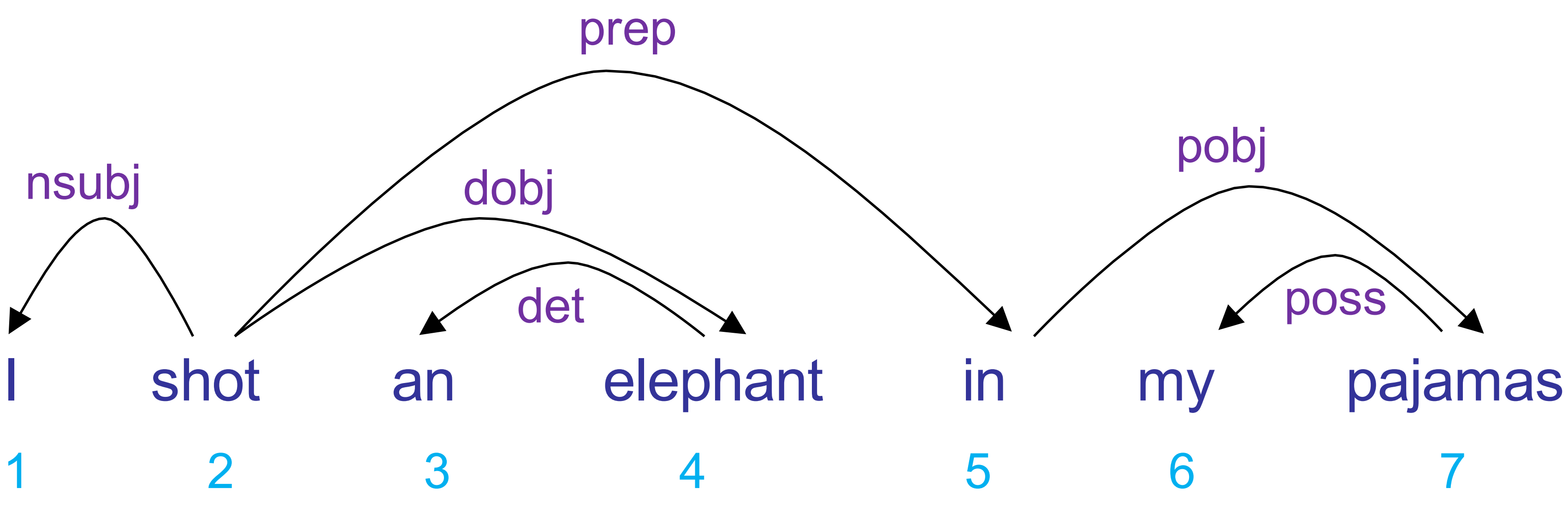



shot
I
elephant
yesterday
in
an
pajamas
my
I shot an elephant in my pajamas yesterday

Typed Depedencies

nsubj(shot-2, i-1)
root(ROOT-0, shot-2)
det(elephant-4, an-3)
dobj(shot-2, elephant-4)

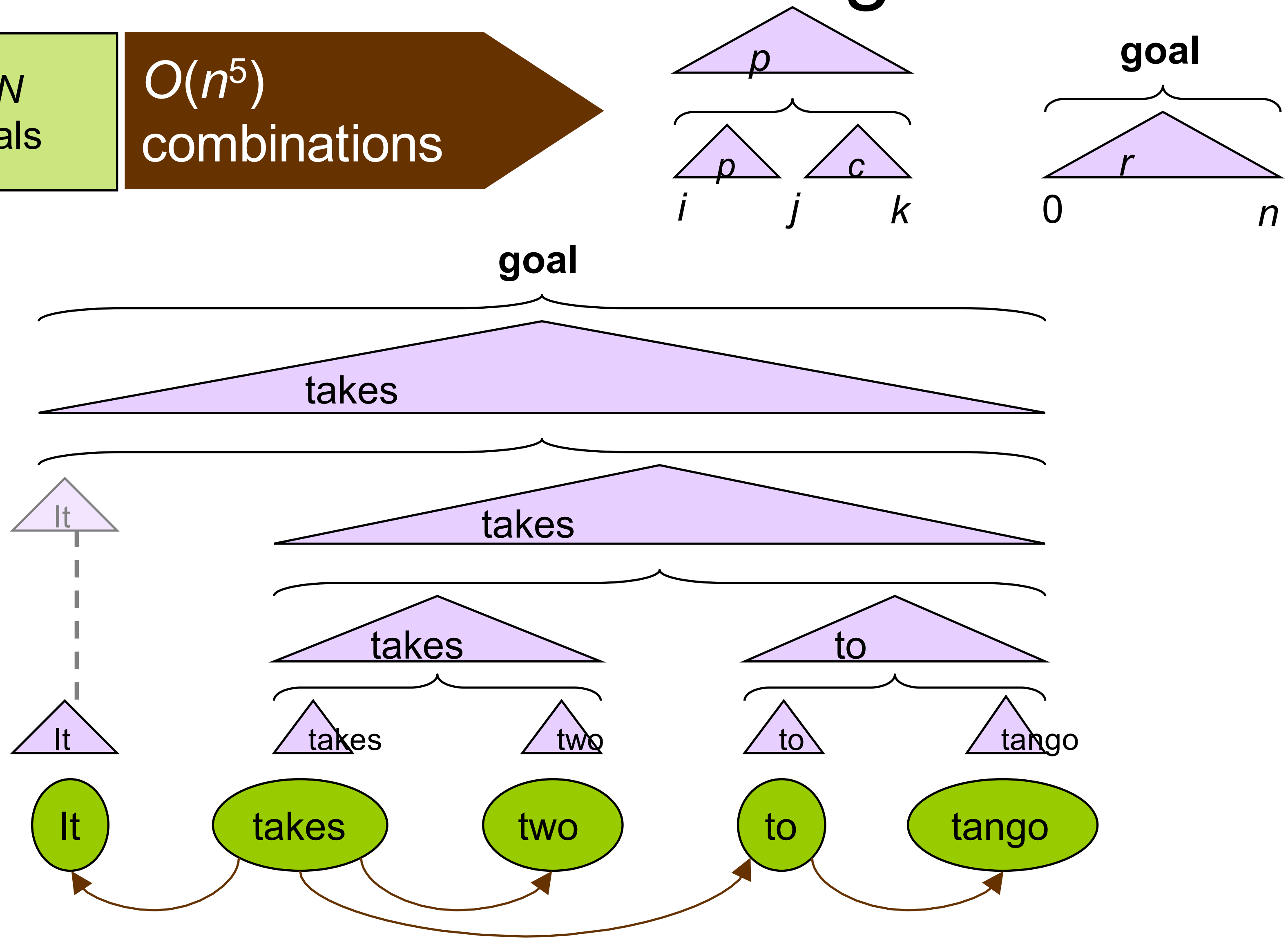
prep(shot-2, in-5)
poss(pajamas-7, my-6)
pobj(in-5, pajamas-7)



Naïve CKY Parsing

$O(n^5N^3)$ if N nonterminals

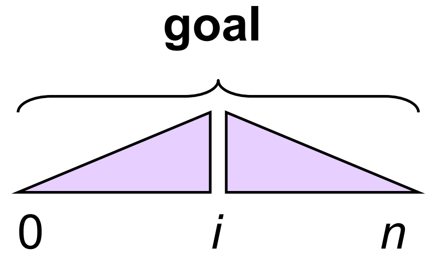
$O(n^5)$ combinations



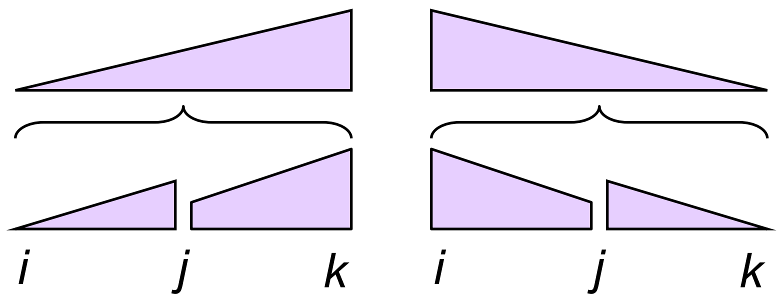
slides from Eisner & Smith

Eisner Algorithm (Eisner & Satta, 1999)

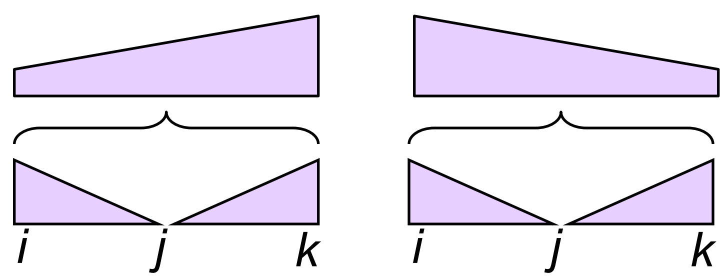
This happens only once as the very final step



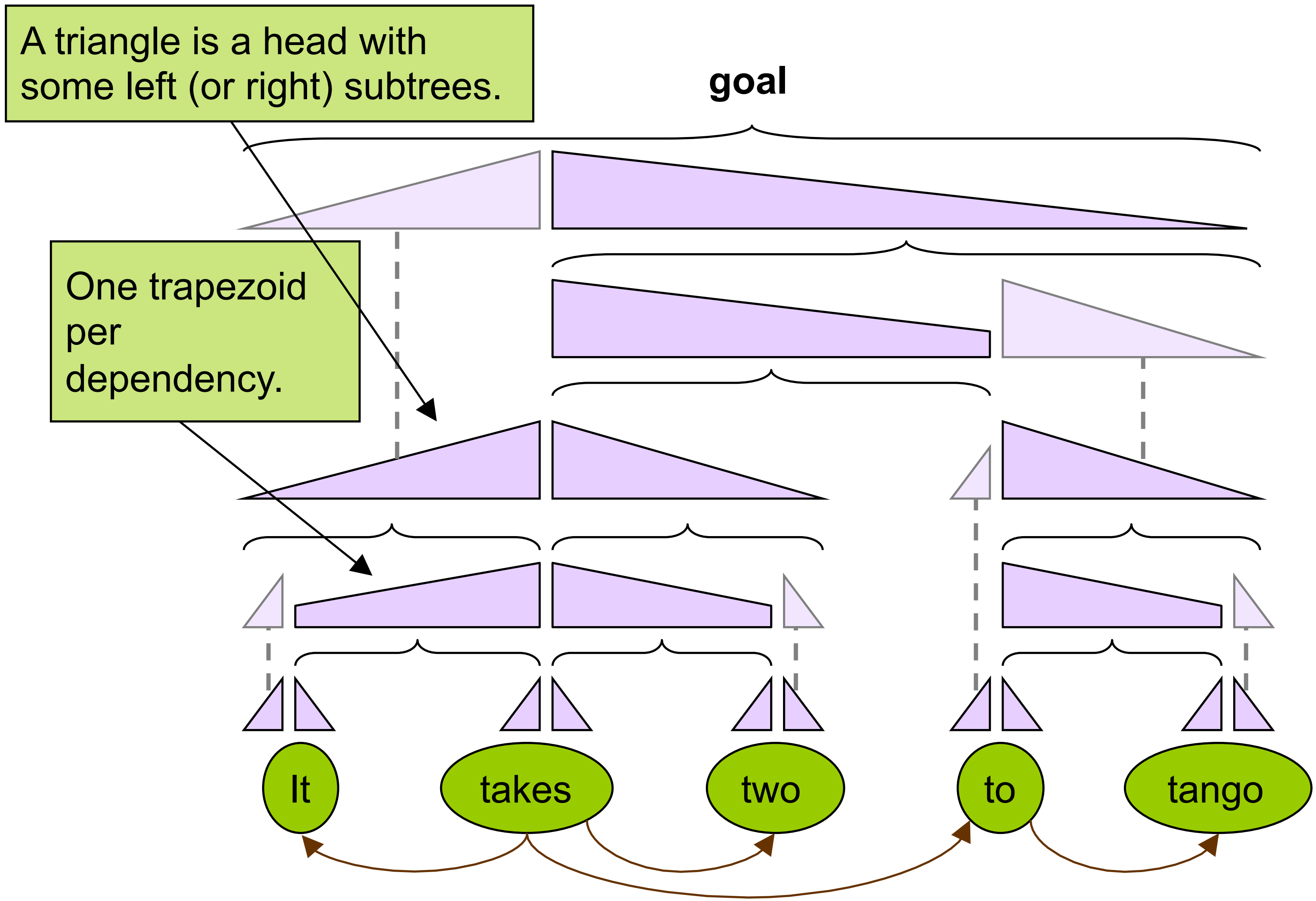
Without adding a dependency arc



When adding a dependency arc (head is higher)



Eisner Algorithm (Eisner & Satta, 1999)



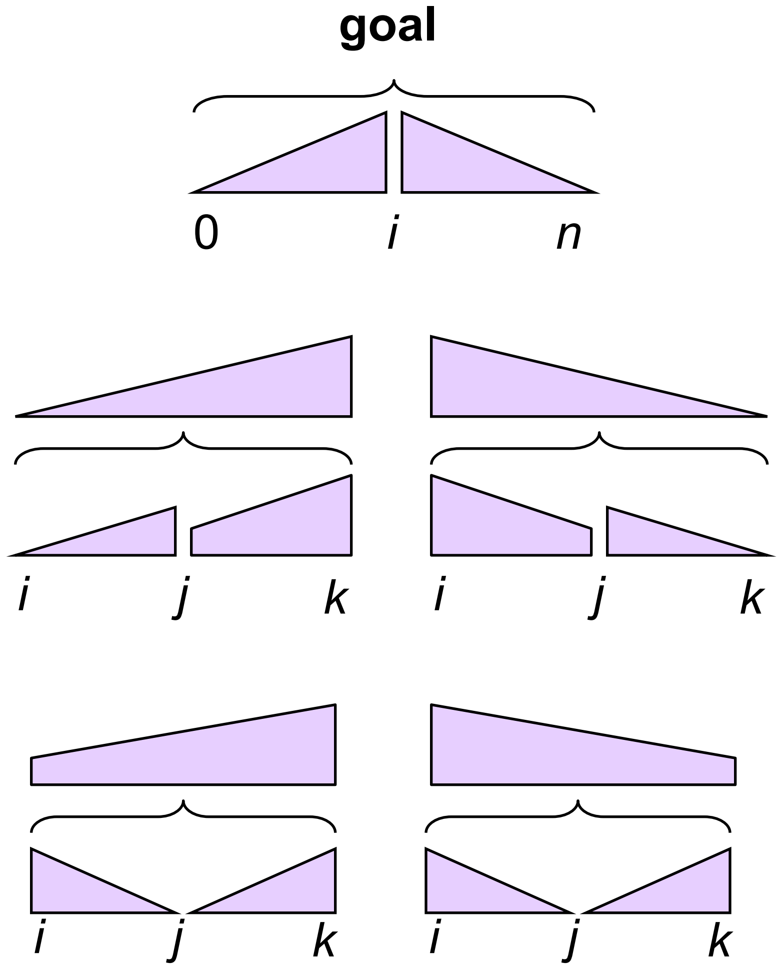
slides from Eisner & Smith

Eisner Algorithm (Eisner & Satta, 1999)

$O(n)$ combinations

$O(n^3)$ combinations

$O(n^3)$ combinations



Gives $O(n^3)$ dependency grammar parsing

slides from Eisner & Smith

Eisner Algorithm

- Base case:

$$\forall t \in \{\unlhd, \unrhd, \lhd, \rhd\},\ \pi(i,i,t) = 0$$

- Recursion:

$$\pi(i,j,\unlhd) = \max_{i \le k < j} \Big(\pi(i,k,\rhd) + \pi(k+1,j,\lhd) + \phi(w_j, w_i)\Big)$$

$$\pi(i,j,\unrhd) = \max_{i \le k < j} \Big(\pi(i,k,\rhd) + \pi(k+1,j,\lhd) + \phi(w_i, w_j))\Big)$$

$$\pi(i,j,\lhd) = \max_{i \le k < j} \Big(\pi(i,k,\lhd) + \pi(k+1,j,\unlhd)\Big)$$

$$\pi(i,j,\rhd) = \max_{i \le k < j} \Big(\pi(i,k,\unrhd) + \pi(k+1,j,\rhd)\Big)$$

- Final case:

$$\pi(1,n,\lhd\rhd) = \max_{1 \le k < n} \Big(\pi(1,k,\lhd) + \pi(k+1,n,\rhd)\Big)$$

CFG vs Dependency Parse I

- CFG focuses on “**constituency**” (i.e., phrasal/clausal structure)
- Dependency focuses on “**head**” relations.

- CFG includes non-terminals. CFG edges are not typed.
- No non-terminals for dependency trees. Instead, dependency trees provide “dependency types” on edges.

- Dependency types encode “**grammatical roles**” like
  - nsubj -- nominal subject
  - dobj – direct object
  - pobj – prepositional object
  - nsubjpass – nominal subject in a passive voice

CFG vs Dependency Parse II

- Can we get “heads” from CFG trees?
  - Yes. In fact, modern statistical parsers based on CFGs use hand-written “head rules” to assign “heads” to all nodes.
- Can we get constituents from dependency trees?
  - Yes, with some efforts.

- Can we transform CFG trees to dependency parse trees?
  - Yes, and transformation software exists. (stanford toolkit based on [de Marneffe et al. LREC 2006])
- Can we transform dependency trees to CFG trees?
  - Mostly yes, but (1) dependency parse can capture ***non-projective*** dependencies, while CFG cannot, and (2) people rarely do this in practice

CFG vs Dependency Parse III

- Both are context-free.
- Both are used frequently today, but dependency parsers are more recently popular.

- CKY Parsing algorithm:
  - O (N^3) using CKY & unlexicalized grammar
  - O (N^5) using CKY & lexicalized grammar (O(N^4) also possible)
- Dependency parsing algorithm:
  - O (N^5) using naïve CKY
  - O (N^3) using Eisner algorithm
  - O (N^2) based on minimum directed spanning tree algorithm (arborescence algorithm, aka, Edmond-Chu-Liu algorithm – see edmond.pdf)
- Linear-time O (N) Incremental parsing (shift-reduce parsing) possible for both grammar formalisms

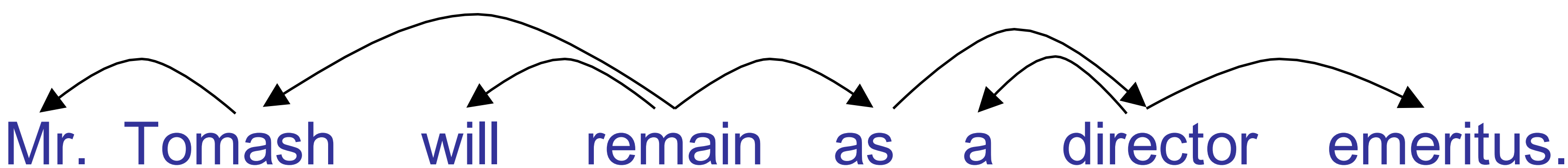
Non Projective Dependencies

- Mr. Tomash will remain as a director emeritus.
- A hearing is scheduled on the issue today.

Non Projective Dependencies

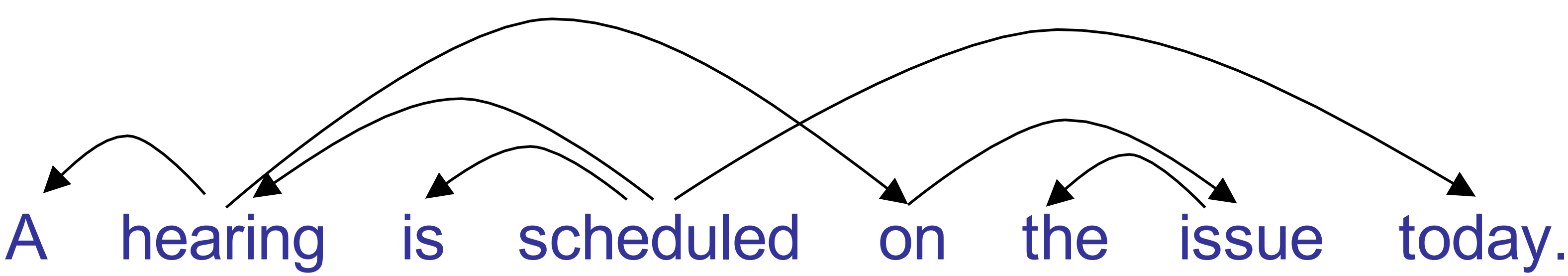
- Projective dependencies: when the tree edges are drawn directly on a sentence, it forms a tree (without a cycle), and there is no crossing edge.

- Projective Dependency:
- Eg:



Non Projective Dependencies

- Projective dependencies: when the tree edges are drawn directly on a sentence, it forms a tree (without a cycle), and there is no crossing edge.
- Non-projective dependency:
- Eg:



Non Projective Dependencies

- which word does "on the issue" modify?
  - We scheduled a meeting on the issue today.
  - A meeting is scheduled on the issue today.
- CFGs capture only projective dependencies (why?)

Coordination across Constituents

- Right-node raising:
  - [[She bought] and [he ate]] bananas.
- Argument-cluster coordination:
  - I give [[you an apple] and [him a pear]].
- Gapping:
  - She likes sushi, and he sashimi

➔ CFGs don't capture coordination across constituents:

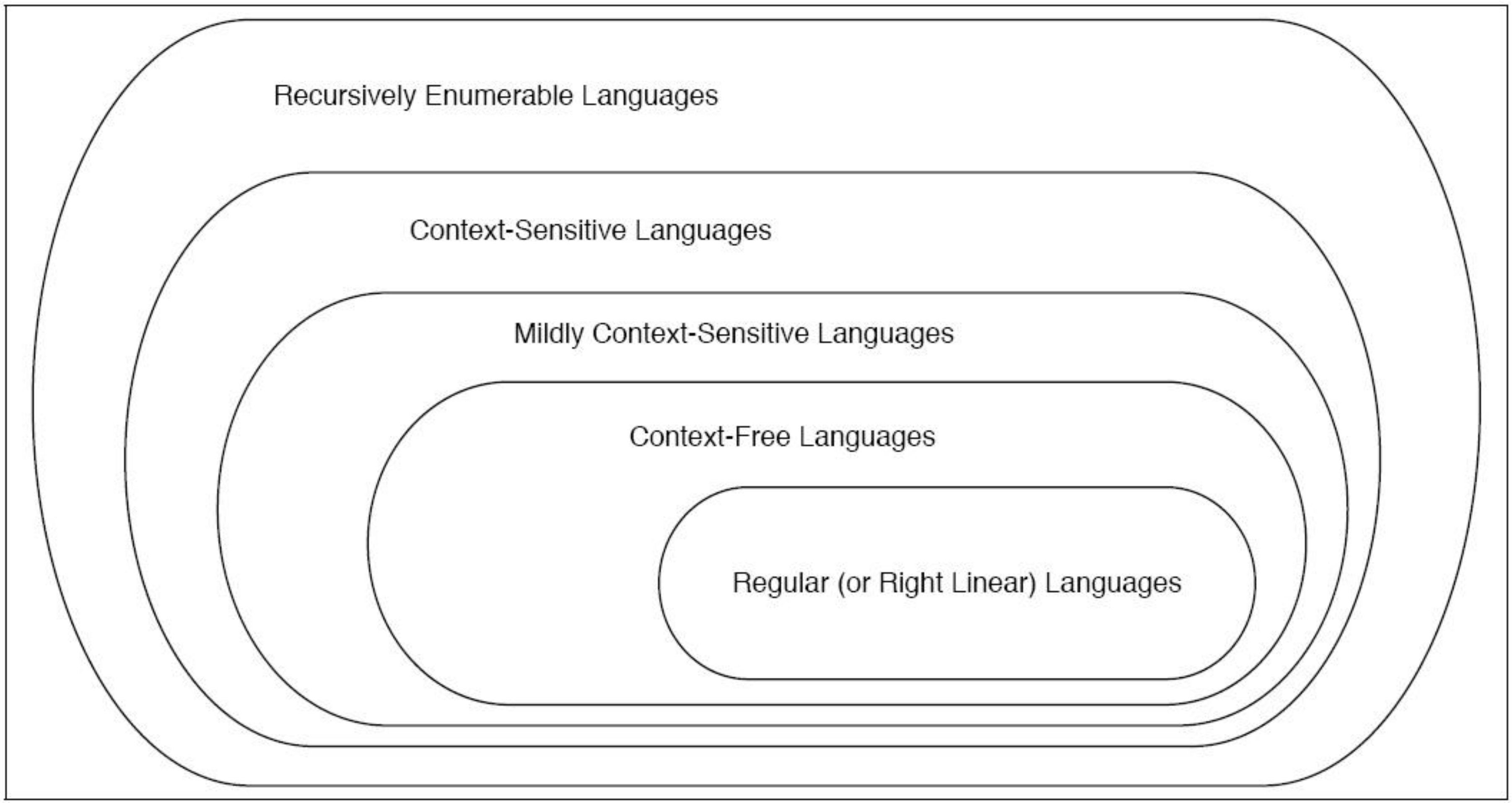
Coordination across Constituents

- She bought and he ate bananas.
- I give you an apple and him a pear.

Compare above to:

- She bought and ate bananas.
- She bought bananas and apples.
- She bought bananas and he ate apples.

The Chomsky Hierarchy



The Chomsky Hierarchy

Type	Common Name	Rule Skeleton	Linguistic Example
0	Turing Equivalent	$\alpha \rightarrow \beta$, s.t. $\alpha \neq \epsilon$	HPSG, LFG, Minimalism
1	Context Sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$, s.t. $\gamma \neq \epsilon$	
–	Mildly Context Sensitive		TAG, CCG
2	Context Free	$A \rightarrow \gamma$	Phrase-Structure Grammars
3	Regular	$A \rightarrow xB$ or $A \rightarrow x$	Finite-State Automata

- Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1987, 1994)
- Lexical Functional Grammar (LFG) (Bresnan, 1982)
- Minimalist Grammar (Stabler, 1997)

- Tree-Adjoining Grammars (TAG) (Joshi, 1969)
- Combinatory Categorial Grammars (CCG) (Steedman, 1986)

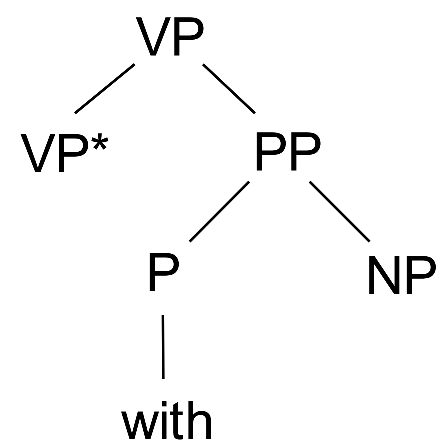
Mildly Context-Sensitive Grammar Formalisms

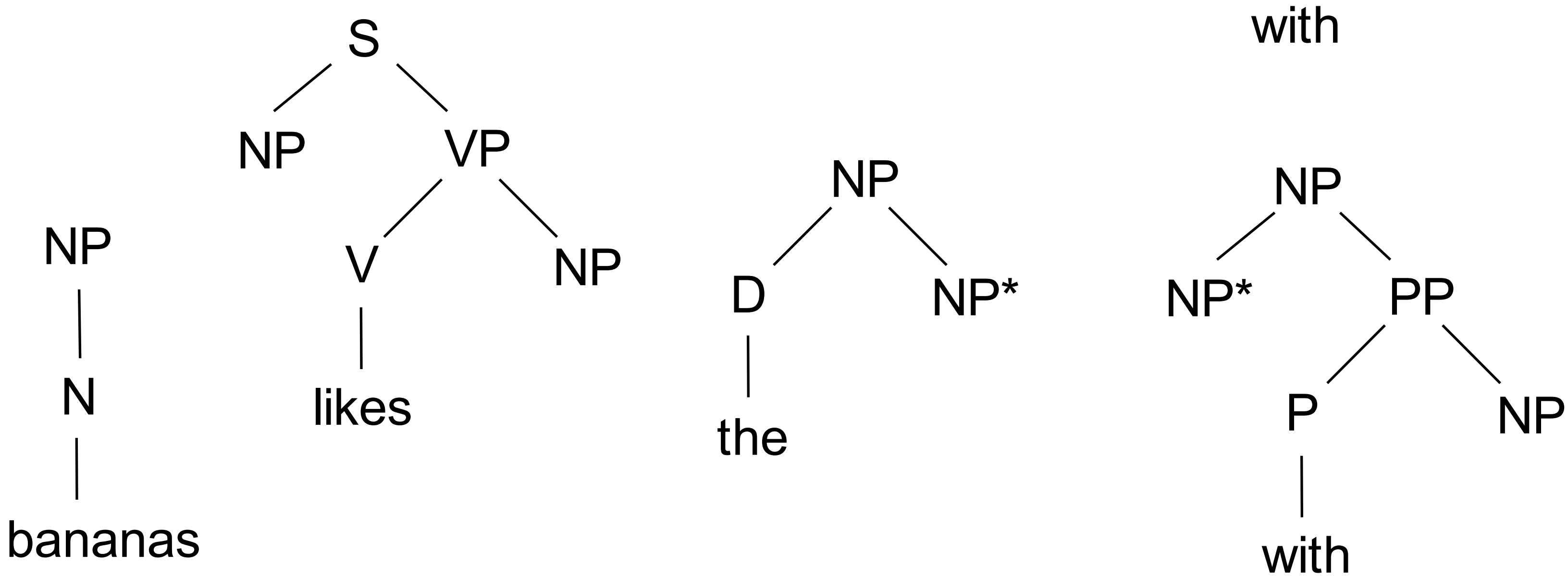
I. Tree Adjoining Grammar (TAG)

Some slides adapted from Julia Hockenmaier’s

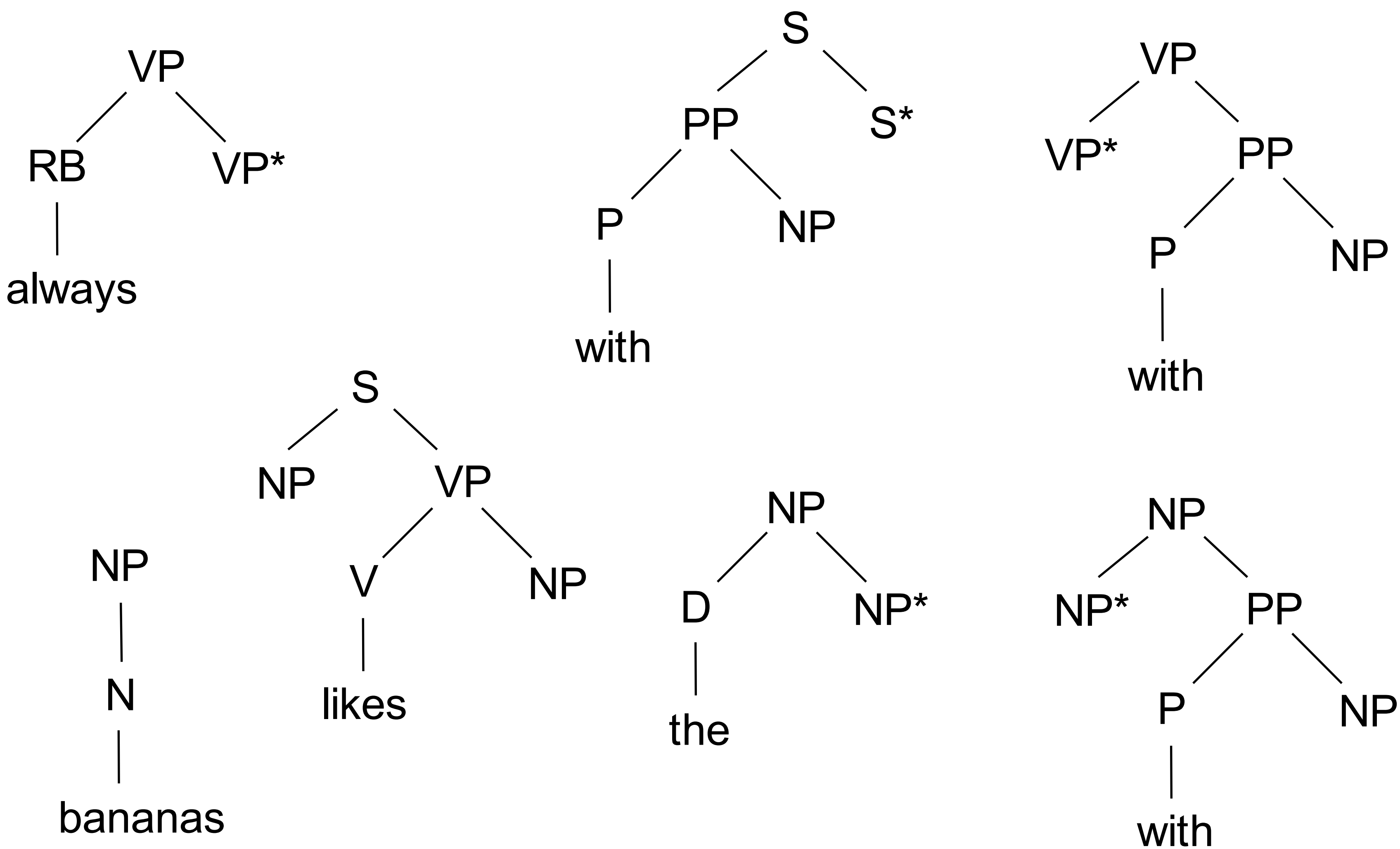
TAG Lexicon (Supertags)

- Tree-Adjoining Grammars (TAG) (Joshi, 1969)
- "*… **super parts of speech (supertags): almost parsing***" (Joshi and Srinivas 1994)
- POS tags enriched with syntactic structure
- also used in other grammar formalisms (e.g., CCG)



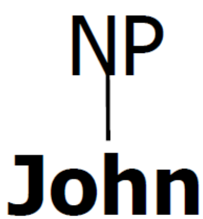


TAG Lexicon (Supertags)

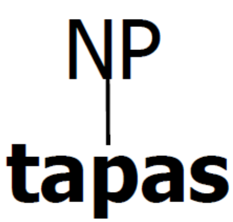


Example: TAG Lexicon

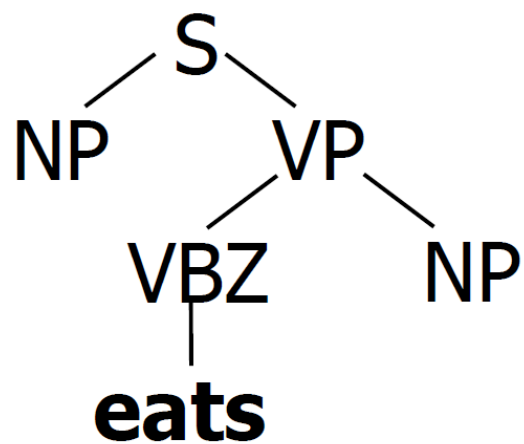
$\alpha 2$:



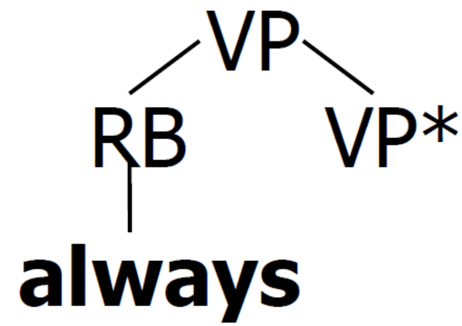
$\alpha 3$:



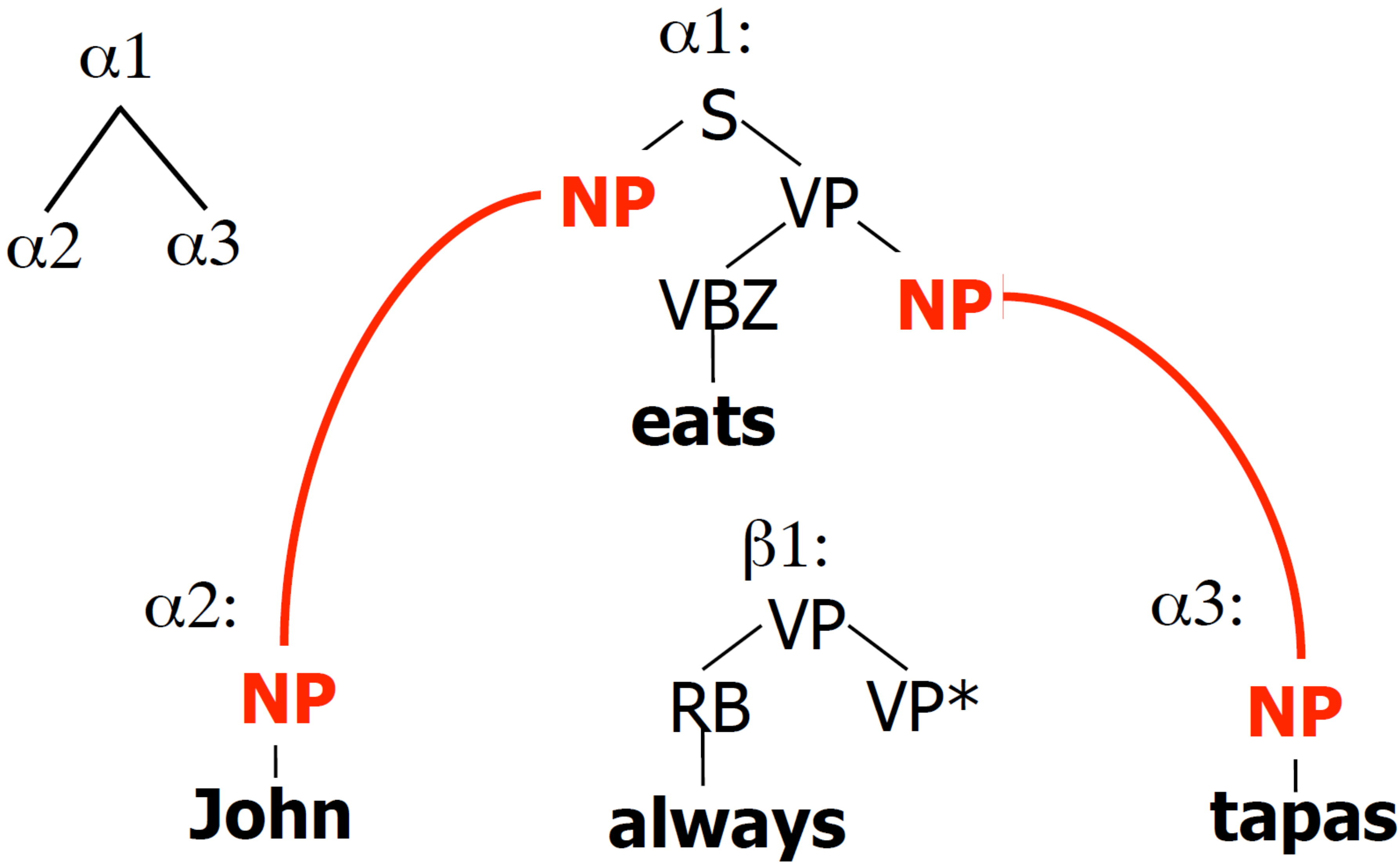
$\alpha 1$:



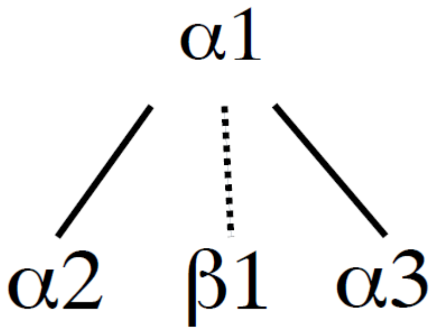
$\beta 1$:

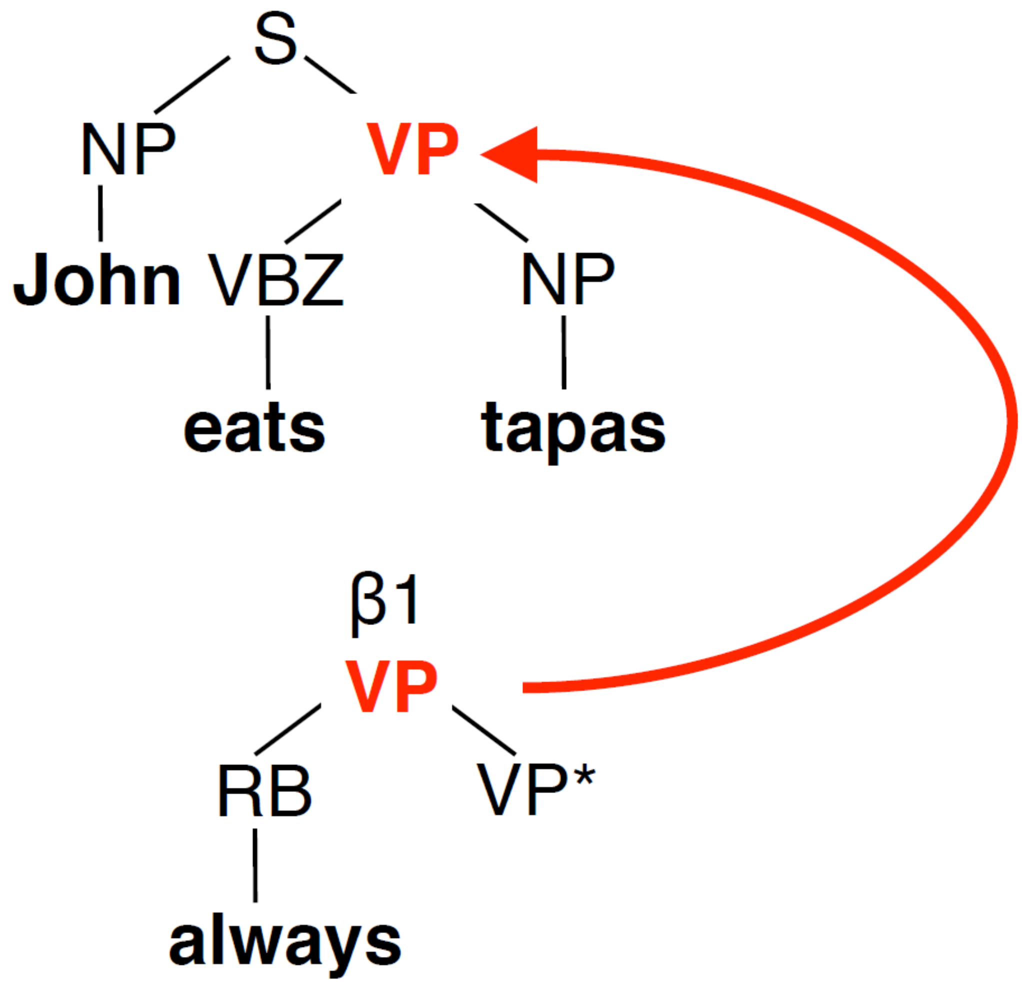


Example: TAG Derivation

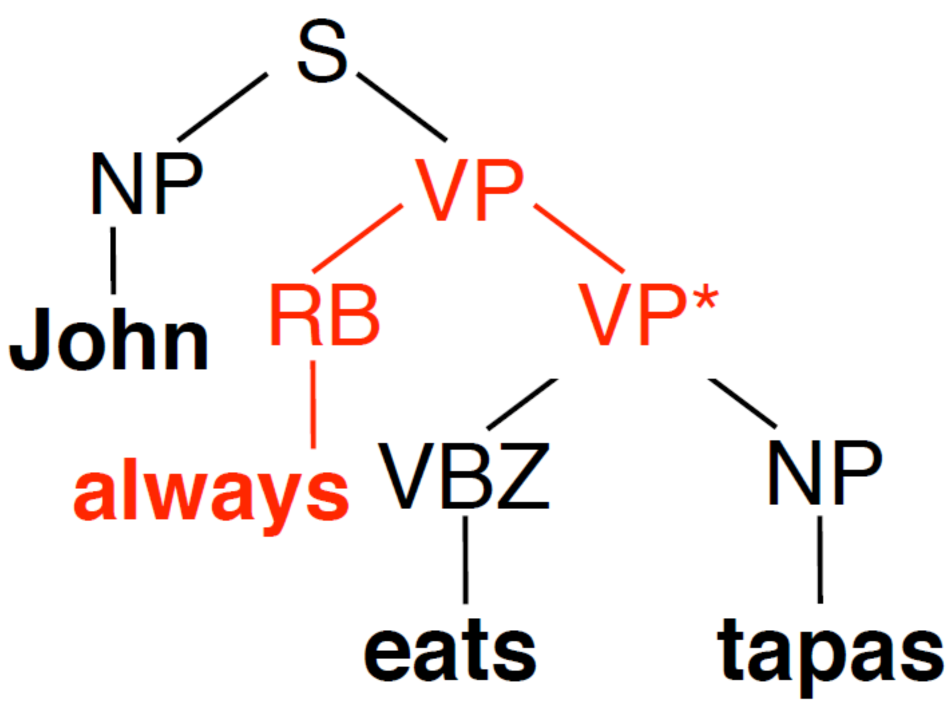


Example: TAG Derivation

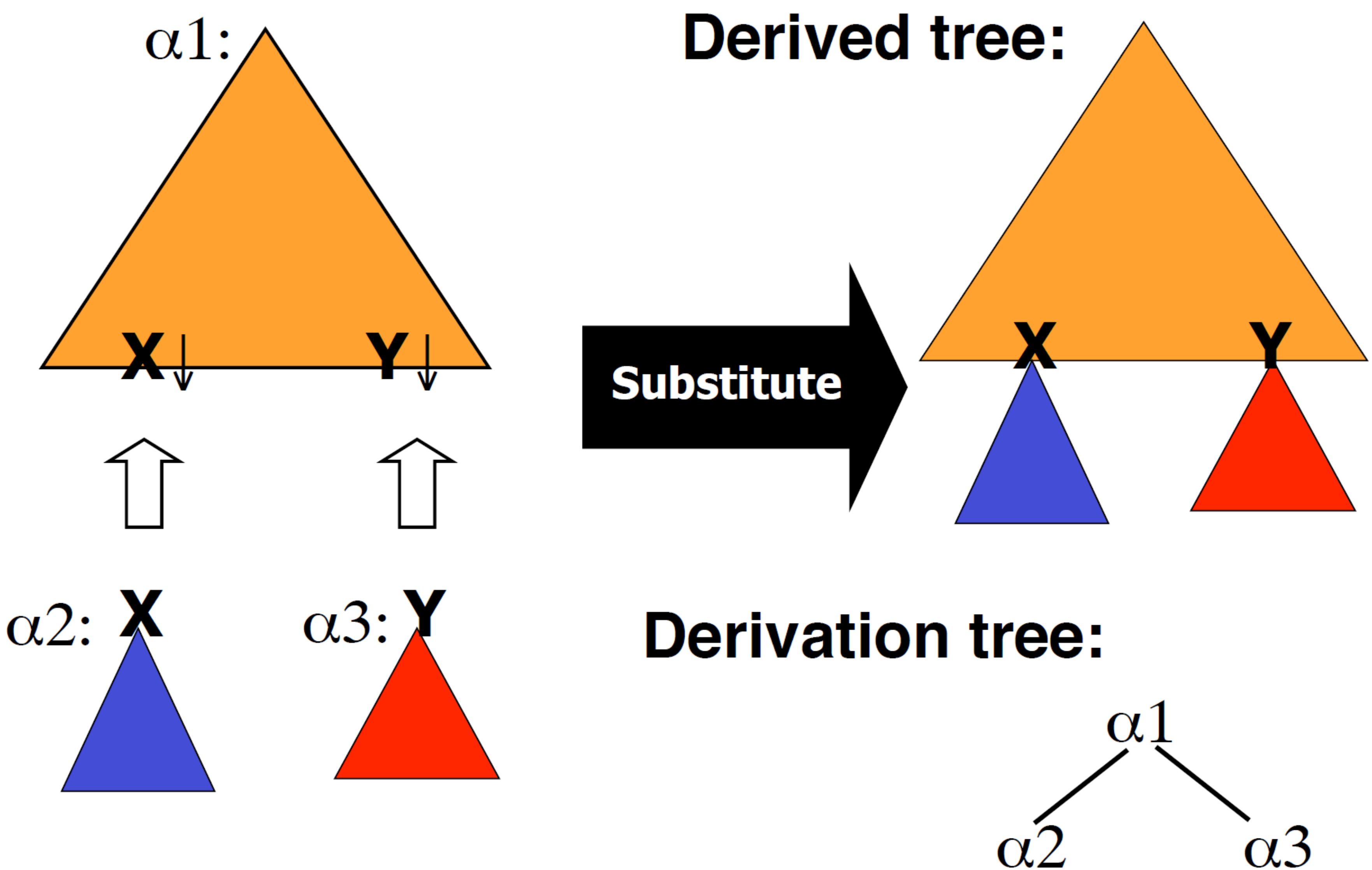




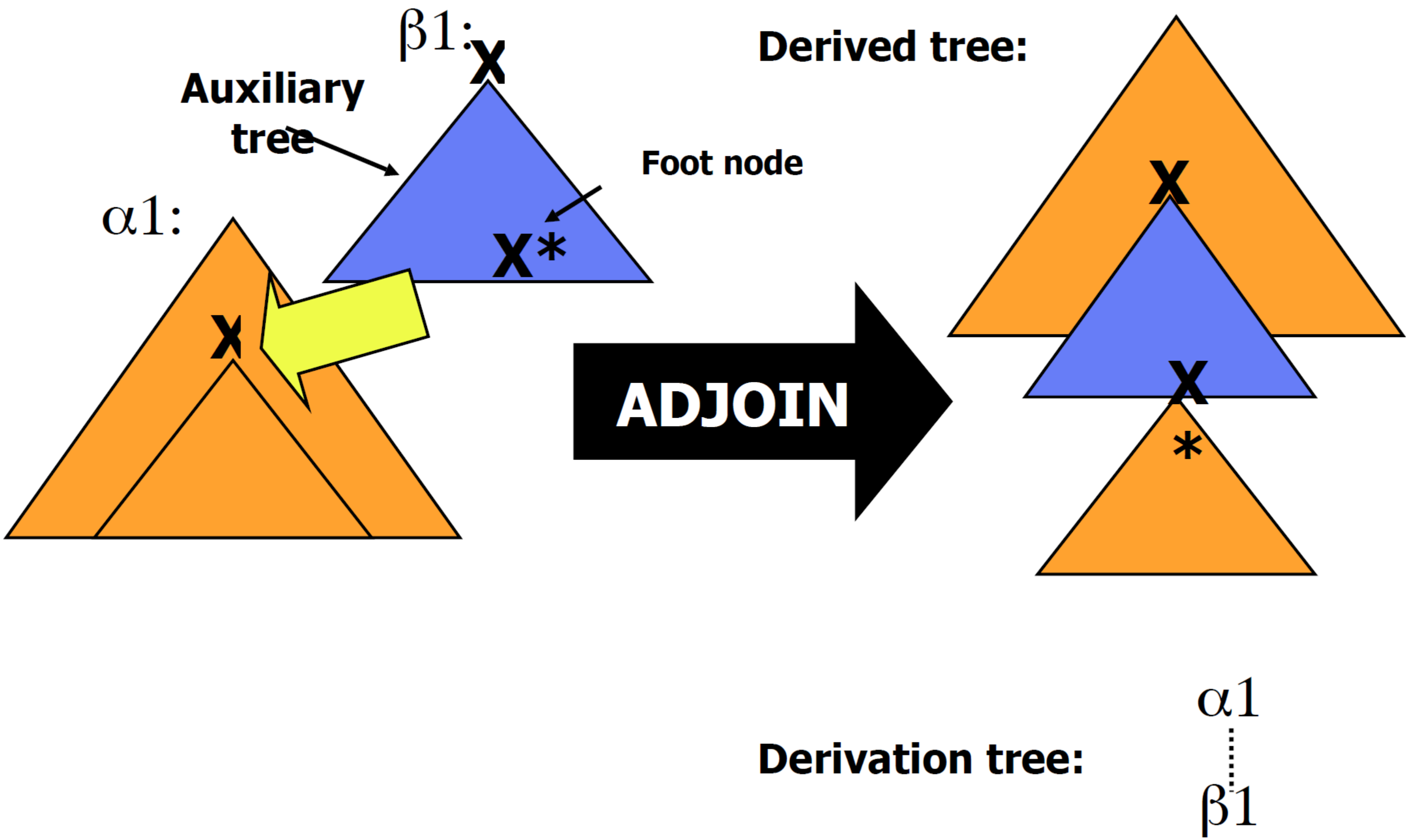
Example: TAG Derivation



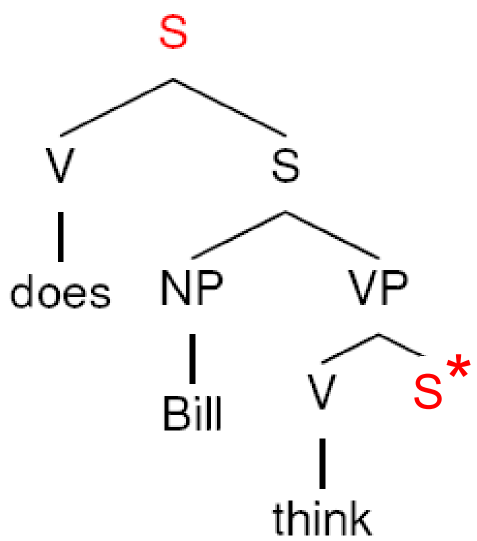
TAG rule 1: **Substitution**

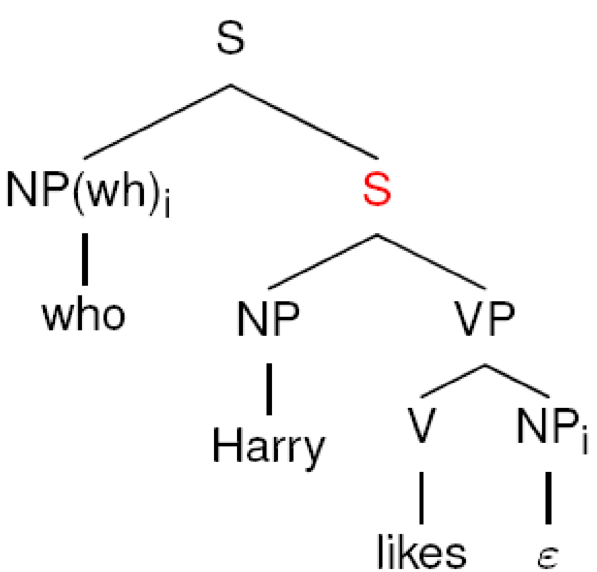


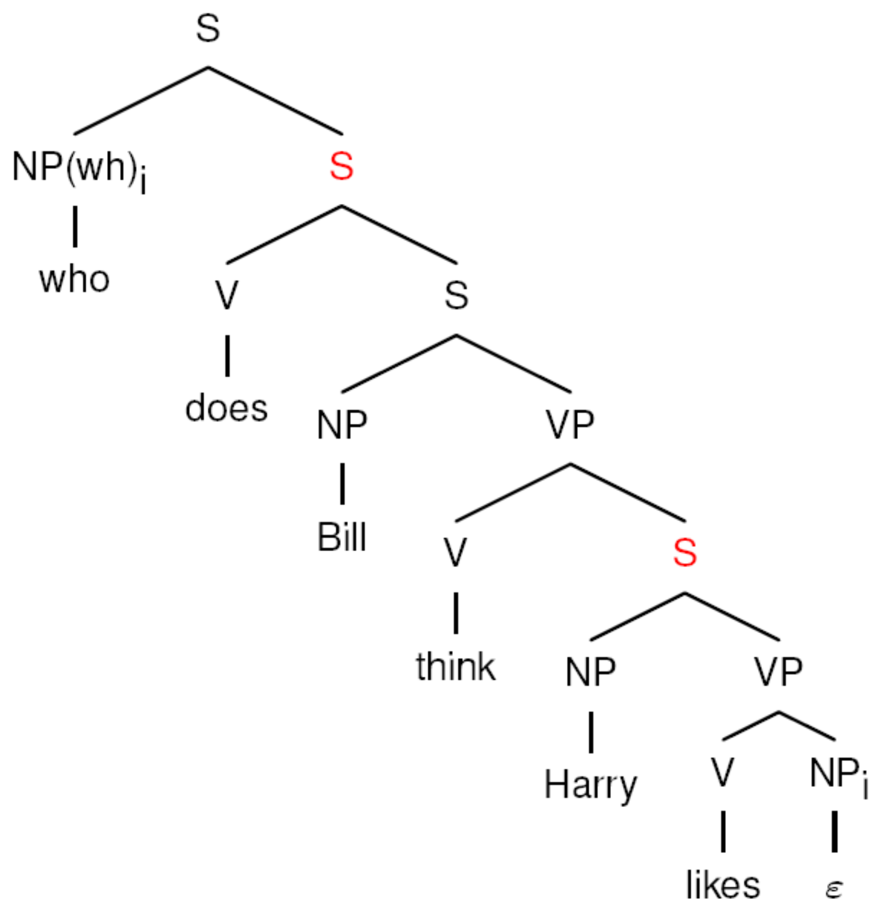
TAG rule 2: **Adjunction**



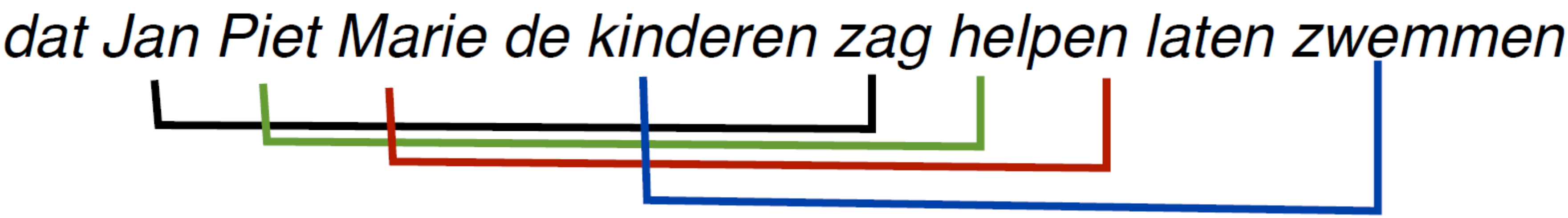
(1) Can handle long distance dependencies







(2) Cross-serial Dependencies

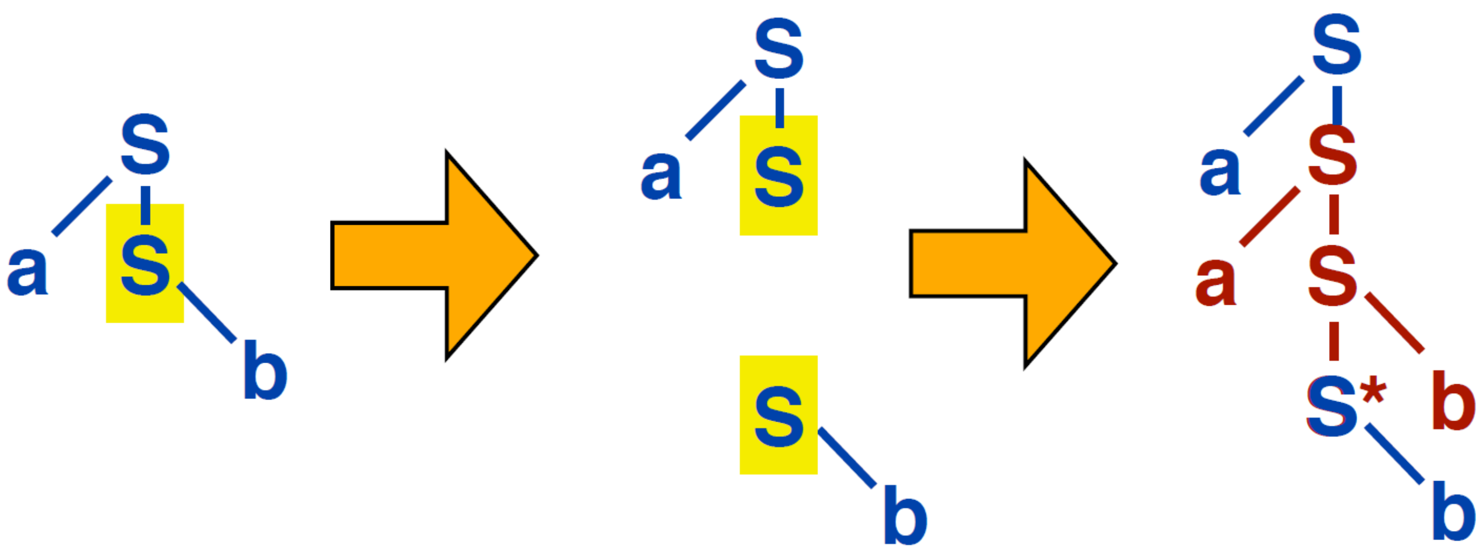


- Dutch and Swiss-German
- Can this be generated from context-free grammar?

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

Elementary trees:

Deriving aabb

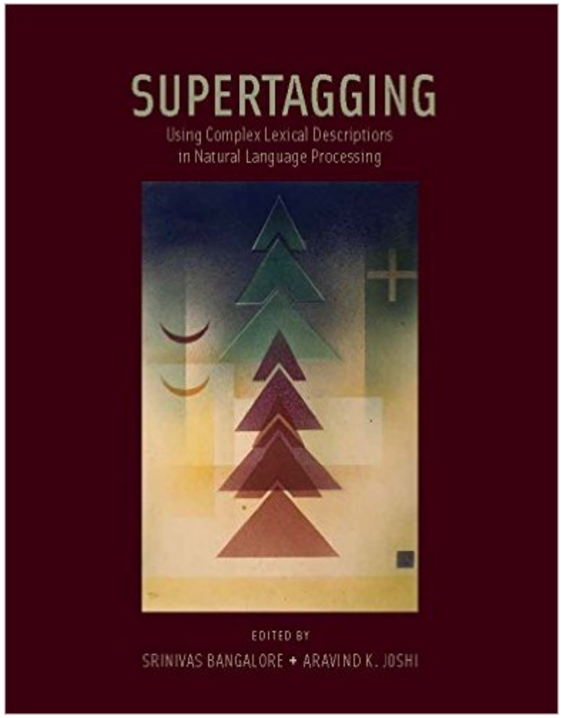


Tree Adjoining Grammar (TAG)

- TAG: Aravind Joshi in 1969
- Supertagging for TAG: Joshi and Srinivas 1994

- Pushing grammar down to lexicon.
- With just two rules: substitution & adjunction

- Parsing Complexity:
  - O(N^7)



- Xtag Project (TAG Penntree) ([http://www.cis.upenn.edu/~xtag/)](http://www.cis.upenn.edu/~xtag/)

- Local expert!
  - Fei Xia @ Linguistics (https://faculty.washington.edu/fxia/)

II. Combinatory Categorial Grammar (CCG)

Some slides adapted from Julia Hockenmaier’s

Categories

- Categories = types
  - Primitive categories
    - N, NP, S, etc
  - Functions
    - a combination of primitive categories
    - S/NP,   (S/NP) / (S/NP),   etc
    - V, VP, Adverb, PP, etc

Combinatory Rules

- Application
  - forward application: x/y y → x
  - backward application: y x\y → x
- Composition
  - forward composition: x/y y/z → x/z
  - backward composition: y\z x\y → x\z
  - (forward crossing composition: x/y y\z → x\z)
  - (backward crossing composition: x\y y/z → x/z)
- Type-raising
  - forward type-raising: x → y / (y\x)
  - backward type-raising: x → y \ (y/x)
- Coordination <&>
  - x conj x → x

Combinatory Rules 1 : Application

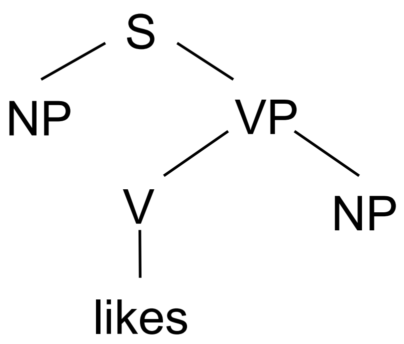
- Forward application “>”
  - X/Y   Y   → X
  - (S\NP)/NP   NP   → S\NP

- Backward application “<“
  - Y   X\Y   → X
  - NP   S\NP   →   S

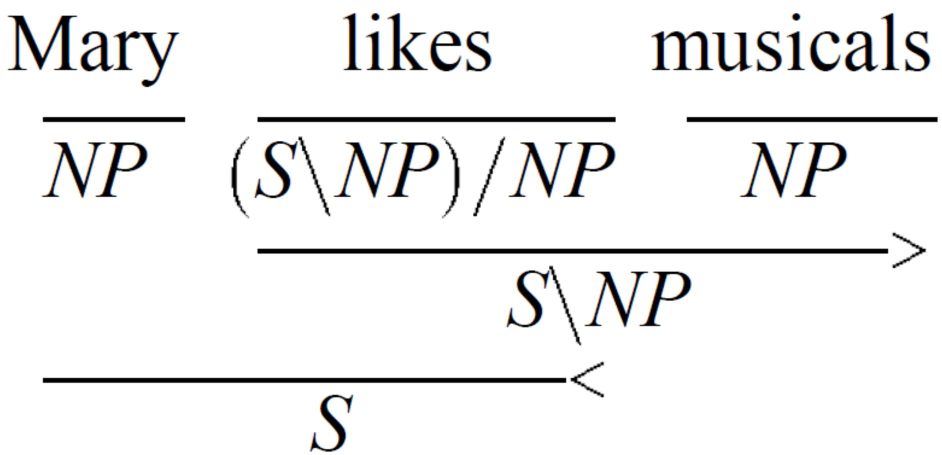
Function

- likes := (S\NP) / NP
  - A transitive verb is a function from NPs into predicate S. That is, it accepts two NPs as arguments and results in S.

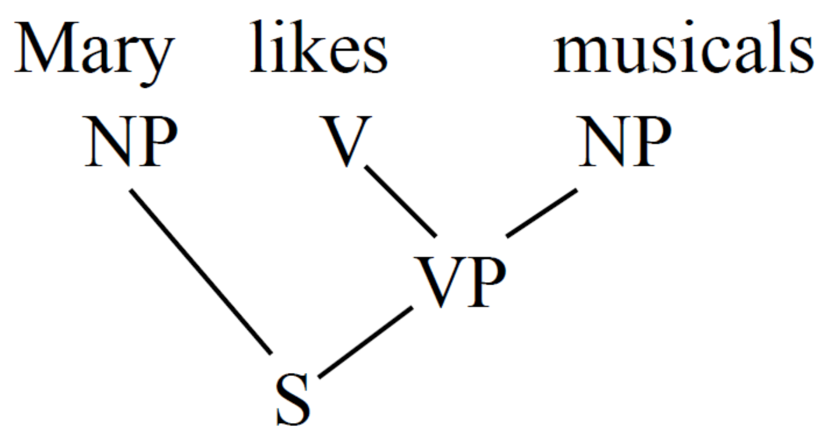
- Transitive verb: (S\NP) / NP
- Intransitive verb: S\NP
- Adverb: (S\NP) \ (S\NP)
- Preposition: (NP\NP) / NP
- Preposition: ((S\NP) \ (S\NP)) / NP



CCG Derivation:



CFG Derivation:



Combinatory Rules

- Application
  - forward application: x/y y ➔ x
  - backward application: y x\y ➔ x
- Composition
  - forward composition: x/y y/z ➔ x/z
  - backward composition: y\z x\y ➔ x\z
  - forward crossing composition: x/y y\z ➔ x\z
  - backward crossing composition: x\y y/z ➔ x/z
- Type-raising
  - forward type-raising: x ➔ y / (y\x)
  - backward type-raising: x ➔ y \ (y/x)
- Coordination <&>
  - x conj x ➔ x

Combinatory Rules 4 : Coordination

- X conj $X \rightarrow X$
- Alternatively, we can express coordination by defining conjunctions as functions as follows:
- and := $(X\backslash X) / X$

Coordination with CCG

$$\begin{array}{ccccc}
\text{I} & \text{loathe} & \text{and} & \text{detest} & \text{opera} \\
\overline{NP} & \overline{(S\backslash NP)/NP} & \overline{CONJ} & \overline{(S\backslash NP)/NP} & \overline{NP} \\
 & \multicolumn{3}{c}{\overline{\qquad\qquad\qquad (S\backslash NP)/NP \qquad\qquad\qquad}^{<\&>}} & \\
 & \multicolumn{4}{c}{\overline{\qquad\qquad\qquad\qquad S\backslash NP \qquad\qquad\qquad\qquad}^{>}} \\
\multicolumn{4}{c}{\overline{\qquad\qquad\qquad\qquad\qquad S \qquad\qquad\qquad\qquad\qquad}^{<}} &
\end{array}$$

Examples from Prof. Mark Steedman

Coordination with CCG

Marcel	conjectured	and	proved	completeness
NP	(S\NP)/NP	(X\X)/X	(S\NP)/NP	NP

- Application
  - forward application: x/y y ➔ x
  - backward application: y x\y ➔ x

Coordination with CCG

Marcel	conjectured	and	proved	completeness
NP	$(S\backslash NP)/NP$	$(X\backslash X)/X$	$(S\backslash NP)/NP$	NP

$$\dfrac{(X\backslash X)/X \quad (S\backslash NP)/NP}{((S\backslash NP)/NP)\backslash((S\backslash NP)/NP)} >$$

$$\dfrac{(S\backslash NP)/NP \quad ((S\backslash NP)/NP)\backslash((S\backslash NP)/NP)}{(S\backslash NP)/NP} <$$

$$\dfrac{(S\backslash NP)/NP \quad NP}{S\backslash NP} >$$

$$\dfrac{NP \quad S\backslash NP}{S} <$$

- Application
  - forward application: x/y y ➔ x
  - backward application: y x\y ➔ x

Combinatory Rules

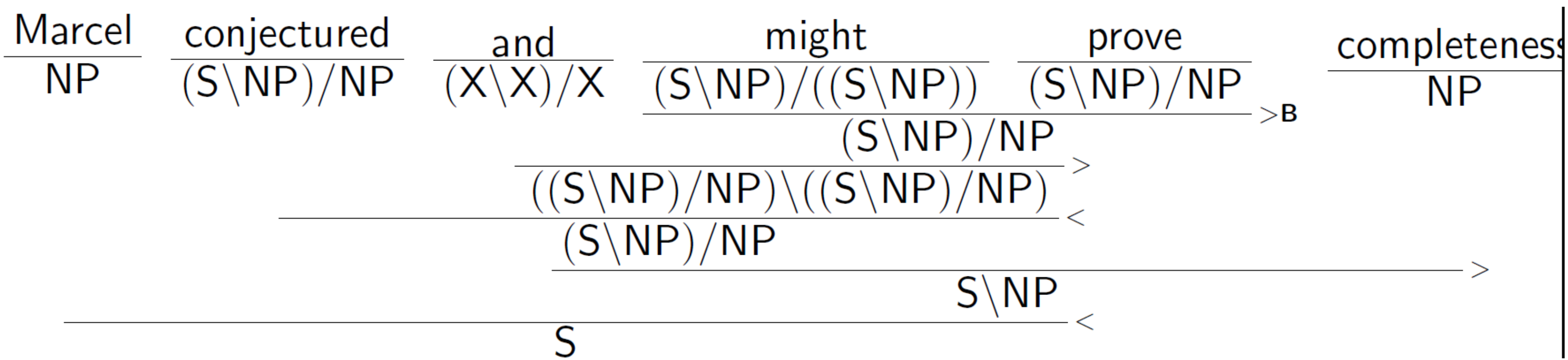
- Application
  - forward application: x/y y ➔ x
  - backward application: y x\y ➔ x
- Composition
  - forward composition: x/y y/z ➔ x/z
  - backward composition: y\z x\y ➔ x\z
  - forward crossing composition: x/y y\z ➔ x\z
  - backward crossing composition: x\y y/z ➔ x/z
- Type-raising
  - forward type-raising: x ➔ y / (y\x)
  - backward type-raising: x ➔ y \ (y/x)
- Coordination <&>
  - x conj x ➔ x

Coordination with CCG

Marcel	conjectured	and	might	prove	completeness
NP	(S\NP)/NP	(X\X)/X	(S\NP)/((S\NP))	(S\NP)/NP	NP

- Application
  - forward application: x/y y ➔ x
  - backward application: y x\y ➔ x
- Composition
  - forward composition: x/y y/z ➔ x/z
  - backward composition: y\z x\y ➔ x\z
  - forward crossing composition: x/y y\z ➔ x\z
  - backward crossing composition: x\y y/z ➔ x/z

Coordination with CCG



- Application
  - forward application: x/y y ➔ x
  - backward application: y x\y ➔ x
- Composition
  - forward composition: x/y y/z ➔ x/z
  - backward composition: y\z x\y ➔ x\z
  - forward crossing composition: x/y y\z ➔ x\z
  - backward crossing composition: x\y y/z ➔ x/z

Combinatory Rules

- Application
  - forward application: x/y y → x
  - backward application: y x\y → x
- Composition
  - forward composition: x/y y/z → x/z
  - backward composition: y\z x\y → x\z
  - forward crossing composition: x/y y\z → x\z
  - backward crossing composition: x\y y/z → x/z
- Type-raising
  - forward type-raising: x → y / (y\x)
  - backward type-raising: x → y \ (y/x)
- Coordination <&>
  - x conj x → x

Combinatory Rules 3 : Type-Raising

- Turns an argument into a function

- Forward type-raising: X ➔ T / (T\X)
- Backward type-raising: X ➔ T \ (T/X)

For instance…

- Subject type-raising: NP ➔ S / (S \ NP)
- Object type-raising: NP ➔ (S\NP) \ ((S\NP) / NP)

Combinatory Rules 3 : Type-Raising

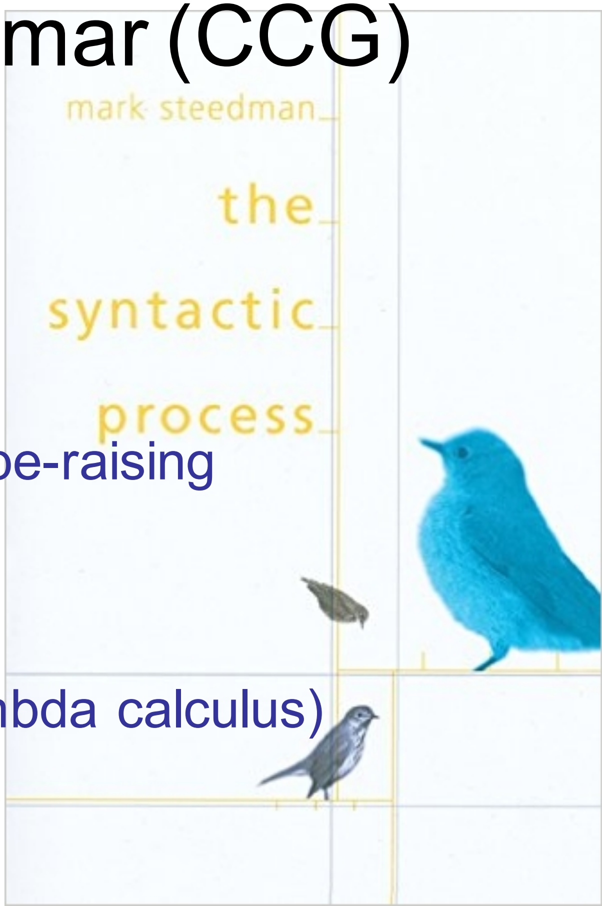
I	dislike	and	Mary	likes	musicals
NP	$(S\backslash NP)/NP$	$CONJ$	NP	$(S\backslash NP)/NP$	NP

- Application
  - forward application: x/y y ➔ x
  - backward application: y x\y ➔ x
- Type-raising
  - forward type-raising: x ➔ y / (y\x)
  - backward type-raising: x ➔ y \ (y/x)
  - Subject type-raising: NP ➔ S / (S \ NP)
  - Object type-raising: NP ➔ (S\NP) \ ((S\NP) / NP)
- Coordination <&>
  - x conj x ➔ x

Combinatory Rules 3 : Type-Raising

$$
\begin{array}{cccccc}
\text{I} & \text{dislike} & \text{and} & \text{Mary} & \text{likes} & \text{musicals} \\
\hline
NP & (S\backslash NP)/NP & CONJ & NP & (S\backslash NP)/NP & NP \\
 & & & :mary' & :\lambda x.\lambda y.like'xy & \\
\overline{S/(S\backslash NP)}^{>T} & & & \overline{S/(S\backslash NP)}^{>T} & & \\
 & & & :\lambda f.f\ mary' & & \\
\multicolumn{2}{c}{\overline{\qquad S/NP \qquad}^{>B}} & & \multicolumn{2}{c}{\overline{\qquad S/NP \qquad}^{>B}} & \\
 & & & \multicolumn{2}{c}{:\lambda x.like'x\ mary'} & \\
\multicolumn{5}{c}{\overline{\qquad\qquad S/NP \qquad\qquad}^{<\&>}} & \\
 & & \multicolumn{4}{c}{\overline{\qquad\qquad S \qquad\qquad}^{>}}
\end{array}
$$

Combinatory Categorial Grammar (CCG)



- CCG: Steedman in 1986

- Pushing grammar down to lexicon.
- With just a few rules: application, composition, type-raising

- We've looked at only syntactic part of CCG
- A lot more in the semantic part of CCG (using lambda calculus)

- Parsing Complexity:
  - O(N^6)

- Local expert!
  - Luke Zettlemoyer (https://www.cs.washington.edu/people/faculty/lsz)