Computational Linguistics

CSC 485/2501
Fall 2023

8

8. Mildly Context-Sensitive Grammar Formalisms

Gerald Penn
Department of Computer Science, University of Toronto

Based on slides by David Smith, Dan Klein, Stephen Clark and Eva Banik

Combinatory Categorial Grammar

Combinatory Categorial Grammar (CCG)

- Categorial grammar (CG) is one of the oldest grammar formalisms
- *Combinatory* Categorial Grammar now well established and computationally well founded (Steedman, 1996, 2000)
  - Account of syntax; semantics; prosody and information structure; automatic parsers; generation

Combinatory Categorial Grammar (CCG)

- CCG is a lexicalized grammar
- An elementary syntactic structure – for CCG a lexical category – is assigned to each word in a sentence

  walked: S\NP "give me an NP to my left and I return a sentence"
- A small number of rules define how categories can combine
  - Rules based on the combinators from Combinatory Logic

CCG Lexical Categories

- Atomic categories: S , N , NP , PP , . . . (not many more)
- Complex categories are built recursively from atomic categories and slashes, which indicate the directions of arguments
- Complex categories encode subcategorisation information
  - intransitive verb: S \NP *walked*
  - transitive verb: (S \NP)/NP *respected*
  - ditransitive verb: ((S \NP)/NP)/NP *gave*
- Complex categories can encode modification
  - PP nominal: (NP \NP)/NP
  - PP verbal: ((S \NP)\(S \NP))/NP

Simple CCG Derivation

$$
\begin{array}{cccl}
\mathit{interleukin-10} & \mathit{inhibits} & \mathit{production} & \\
\hline
NP & (S\backslash NP)/NP & NP & \\
\cline{2-3}
 & \multicolumn{2}{c}{S\backslash NP} & > \\
\cline{1-3}
\multicolumn{3}{c}{S} & <
\end{array}
$$

$>$ forward application

$<$ backward application

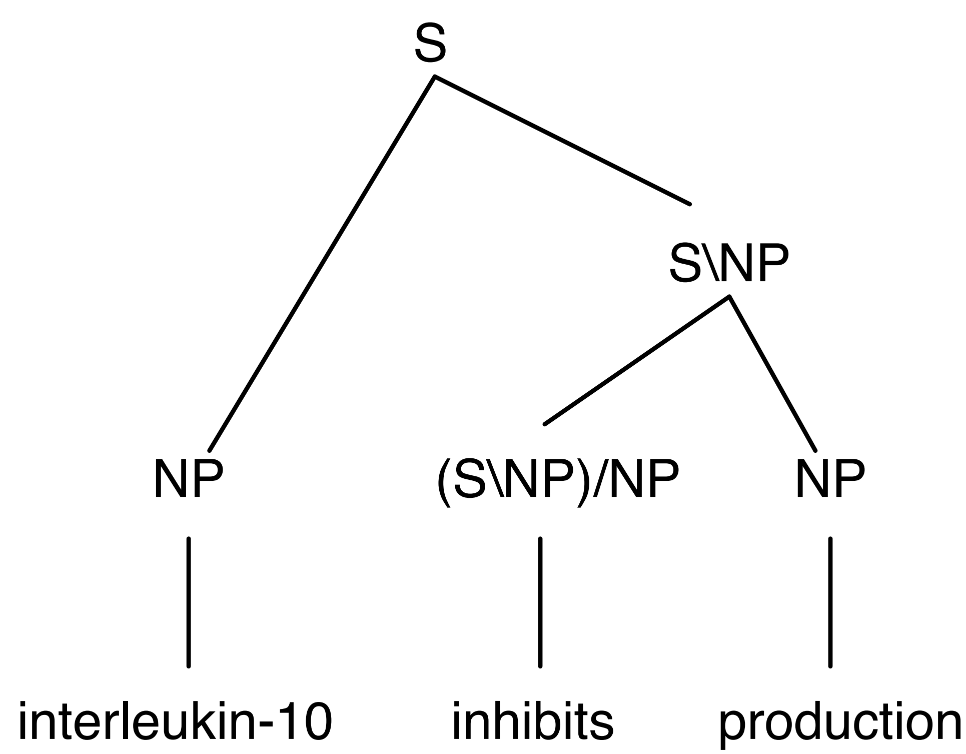
Function Application Schemata

- Forward ($>$) and backward ($<$) application:

$$
\begin{array}{rcll}
X/Y \quad Y & \Rightarrow & X & (>) \\
Y \quad X\backslash Y & \Rightarrow & X & (<)
\end{array}
$$

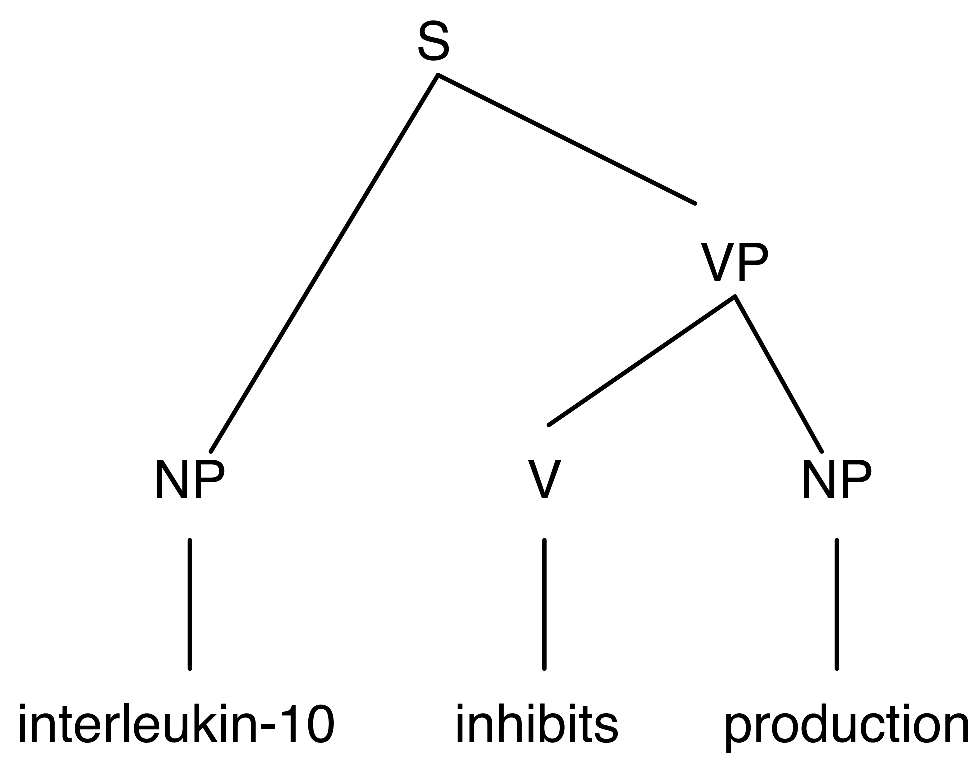
Classical Categorial Grammar

- 'Classical' Categorial Grammar only has application rules
- Classical Categorial Grammar is context free



Classical Categorial Grammar

- ‘Classical’ Categorial Grammar only has application rules
- Classical Categorial Grammar is context free



Extraction out of a Relative Clause

$$\frac{The}{NP/N} \quad \frac{company}{N} \quad \frac{which}{(NP\backslash NP)/(S/NP)} \quad \frac{Microsoft}{NP} \quad \frac{bought}{(S\backslash NP)/NP}$$

Extraction out of a Relative Clause

$$
\begin{array}{ccccc}
\textit{The} & \textit{company} & \textit{which} & \textit{Microsoft} & \textit{bought} \\
\overline{NP/N} & \overline{N} & \overline{(NP\backslash NP)/(S/NP)} & \overline{NP} & \overline{(S\backslash NP)/NP} \\
 & & & \overline{S/(S\backslash NP)}^{>\mathbf{T}} &
\end{array}
$$

$>\mathbf{T}$ type-raising

Extraction out of a Relative Clause

$$\dfrac{\textit{The}}{NP/N} \quad \dfrac{\textit{company}}{N} \quad \dfrac{\textit{which}}{(NP\backslash NP)/(S/NP)} \quad \dfrac{\dfrac{\dfrac{\textit{Microsoft}}{NP}}{S/(S\backslash NP)}{>}\mathbf{T} \quad \dfrac{\textit{bought}}{(S\backslash NP)/NP}}{S/NP}{>}\mathbf{B}$$

$>$ **T** type-raising

$>$ **B** forward composition

Extraction out of a Relative Clause

$$
\begin{array}{ccccc}
\mathit{The} & \mathit{company} & \mathit{which} & \mathit{Microsoft} & \mathit{bought} \\
\overline{NP/N} & \overline{N} & \overline{(NP\backslash NP)/(S/NP)} & \overline{NP} & \overline{(S\backslash NP)/NP} \\
 & & & \overline{S/(S\backslash NP)}^{>\mathbf{T}} & \\
 & & & \multicolumn{2}{c}{\overline{S/NP}^{>\mathbf{B}}} \\
 & & \multicolumn{3}{c}{\overline{NP\backslash NP}^{>}}
\end{array}
$$

Extraction out of a Relative Clause

$$
\begin{array}{ccccc}
\mathit{The} & \mathit{company} & \mathit{which} & \mathit{Microsoft} & \mathit{bought} \\
\hline
NP/N & N & (NP\backslash NP)/(S/NP) & NP & (S\backslash NP)/NP \\
\multicolumn{2}{c}{\overline{\qquad\qquad\qquad}{>}} & & \overline{\qquad\quad}{>}\mathbf{T} & \\
\multicolumn{2}{c}{\color{blue}{NP}} & & S/(S\backslash NP) & \\
 & & & \multicolumn{2}{c}{\overline{\qquad\qquad\qquad}{>}\mathbf{B}} \\
 & & & \multicolumn{2}{c}{S/NP} \\
 & & \multicolumn{3}{c}{\overline{\qquad\qquad\qquad\qquad\qquad}{>}} \\
 & & \multicolumn{3}{c}{NP\backslash \color{blue}{NP}} \\
\multicolumn{5}{c}{\overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad}{<}} \\
\multicolumn{5}{c}{NP}
\end{array}
$$

Forward Composition and Type-Raising

- Forward composition ($>_{\mathbf{B}}$):

$$X/Y \quad Y/Z \quad \Rightarrow \quad X/Z \qquad (>_{\mathbf{B}})$$

- Type-raising ($\mathbf{T}$):

$$\begin{aligned} X &\Rightarrow T/(T\backslash X) \qquad (>_{\mathbf{T}}) \\ X &\Rightarrow T\backslash(T/X) \qquad (<_{\mathbf{T}}) \end{aligned}$$

- Extra combinatory rules increase the weak generative power to mild context -sensitivity

"Non-constituents" in CCG – Right Node Raising

$$
\begin{array}{cccccc}
\textit{Google} & \textit{sells} & \textit{but} & \textit{Microsoft} & \textit{buys} & \textit{shares} \\
\hline
NP & (S\backslash NP)/NP & conj & NP & (S\backslash NP)/NP & NP \\
\overline{}{>}\mathbf{T} & & & \overline{}{>}\mathbf{T} & & \\
S/(S\backslash NP) & & & S/(S\backslash NP) & &
\end{array}
$$

$>\mathbf{T}$ type-raising

"Non-constituents" in CCG – Right Node Raising

Google	*sells*	*but*	*Microsoft*	*buys*	*shares*
NP	$(S\backslash NP)/NP$	$conj$	NP	$(S\backslash NP)/NP$	NP
$S/(S\backslash NP)$ $>\mathbf{T}$			$S/(S\backslash NP)$ $>\mathbf{T}$		
S/NP $>\mathbf{B}$			S/NP $>\mathbf{B}$		

$>$ **T** type-raising
$>$ **B** forward composition

"Non-constituents" in CCG – Right Node Raising

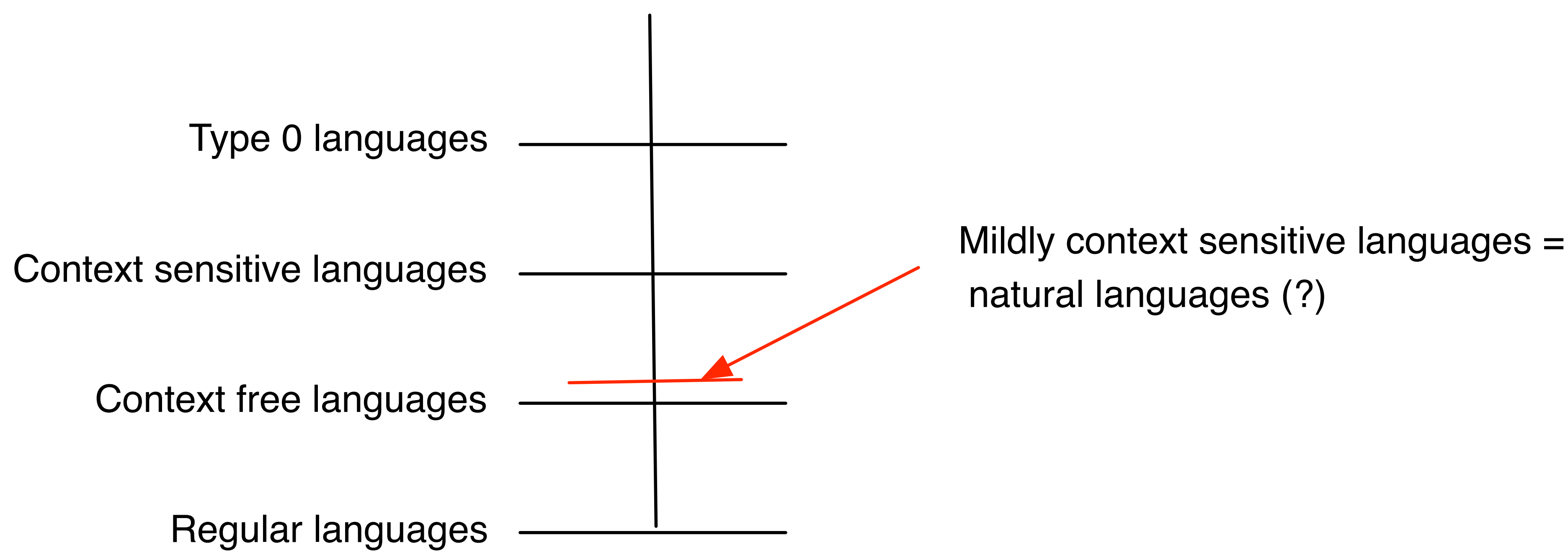
$$
\begin{array}{cccccc}
\mathit{Google} & \mathit{sells} & \mathit{but} & \mathit{Microsoft} & \mathit{buys} & \mathit{shares} \\
\hline
NP & (S\backslash NP)/NP & conj & NP & (S\backslash NP)/NP & NP \\
\cline{1-1}\cline{4-4}
S/(S\backslash NP)\ {>}\mathbf{T} & & & S/(S\backslash NP)\ {>}\mathbf{T} & & \\
\cline{1-2}\cline{4-5}
\multicolumn{2}{c}{S/NP\ {>}\mathbf{B}} & & \multicolumn{2}{c}{S/NP\ {>}\mathbf{B}} & \\
\cline{1-5}
\multicolumn{5}{c}{S/NP\ {<}\Phi{>}} &
\end{array}
$$

"Non-constituents" in CCG – Right Node Raising

$$
\begin{array}{cccccc}
\textit{Google} & \textit{sells} & \textit{but} & \textit{Microsoft} & \textit{buys} & \textit{shares} \\
\overline{NP} & \overline{(S\backslash NP)/NP} & \overline{conj} & \overline{NP} & \overline{(S\backslash NP)/NP} & \overline{\color{blue}{NP}} \\
\overline{S/(S\backslash NP)}{>}\mathbf{T} & & & \overline{S/(S\backslash NP)}{>}\mathbf{T} & & \\
\multicolumn{2}{c}{\overline{\qquad\qquad S/NP \qquad\qquad}{>}\mathbf{B}} & & \multicolumn{2}{c}{\overline{\qquad\qquad S/NP \qquad\qquad}{>}\mathbf{B}} & \\
\multicolumn{5}{c}{\overline{\qquad\qquad\qquad\qquad S/\color{blue}{NP} \qquad\qquad\qquad\qquad}{<}\Phi{>}} & \\
\multicolumn{6}{c}{\overline{\qquad\qquad\qquad\qquad\qquad S \qquad\qquad\qquad\qquad\qquad}{>}}
\end{array}
$$

Combinatory Categorial Grammar

- CCG is *mildly* context sensitive
- Natural language is provably non-context free
- Constructions in Dutch and Swiss German (Shieber, 1985) require more than context free power for their analysis
    - these have *crossing* dependencies (which CCG can handle)



CCG Semantics

- Categories encode argument sequences
- Parallel syntactic combinator operations and lambda calculus semantic operations

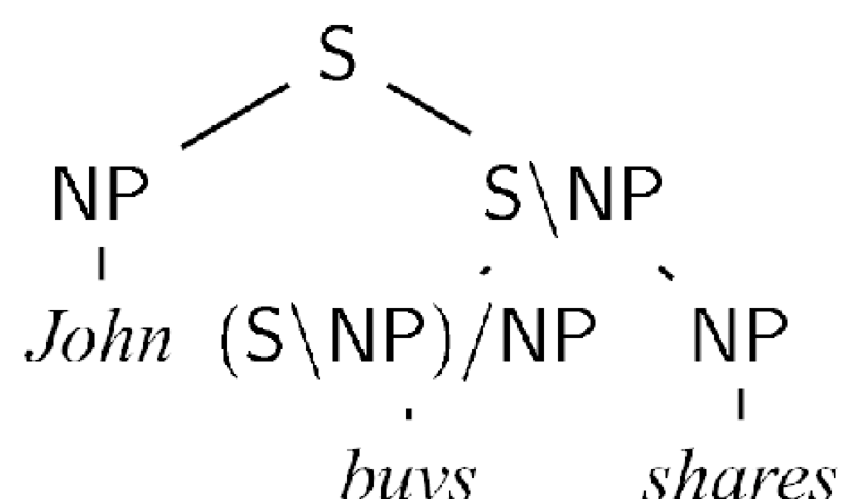
$John \vdash \mathsf{NP} : john'$

$shares \vdash \mathsf{NP} : shares'$

$buys \vdash (\mathsf{S}\backslash\mathsf{NP})/\mathsf{NP} : \lambda x.\lambda y.buys'xy$

$sleeps \vdash \mathsf{S}\backslash\mathsf{NP} : \lambda x.sleeps'x$

$well \vdash (\mathsf{S}\backslash\mathsf{NP})\backslash(\mathsf{S}\backslash\mathsf{NP}) : \lambda f.\lambda x.well'(fx)$



CCG Semantics

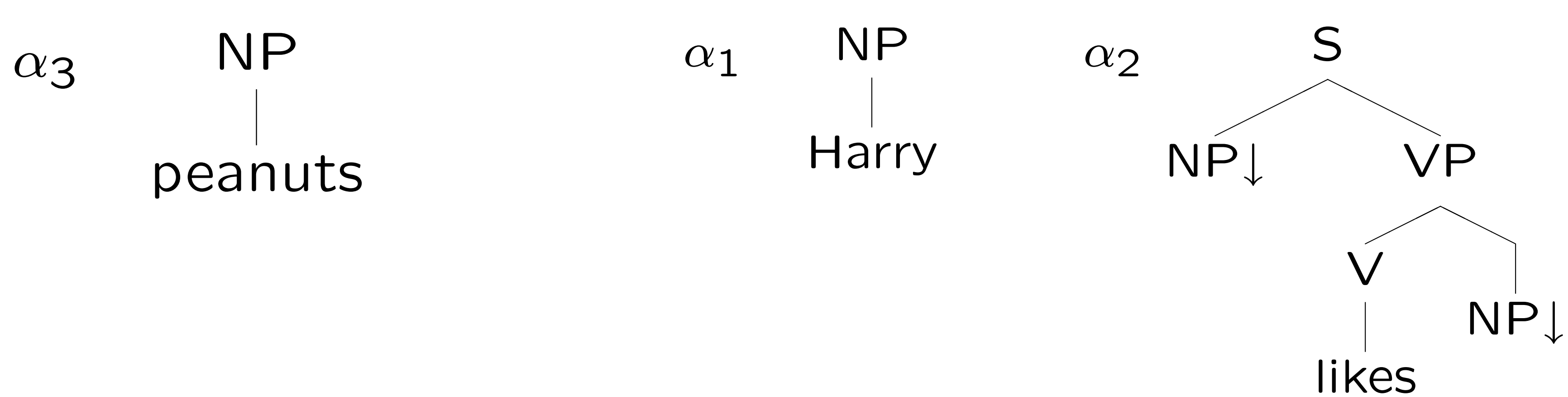
Left arg.	Right arg.	Operation	Result
X/Y : f	Y : a	Forward application	X : f(a)
Y : a	X\Y : f	Backward application	X : f(a)
X/Y : f	Y/Z : g	Forward composition	X/Z : λx.f(g(x))
X : a		Type raising	T/(T\X) : λf.f(a)

etc.

Tree Adjoining Grammar

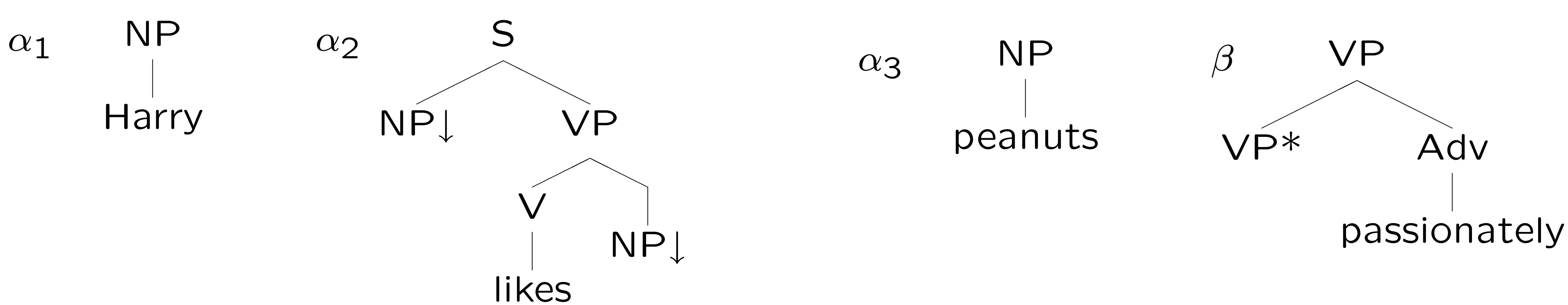
TAG Building Blocks

- Elementary trees (of many depths)
- Substitution at ↓
- Tree *Substitution* Grammar equivalent to CFG

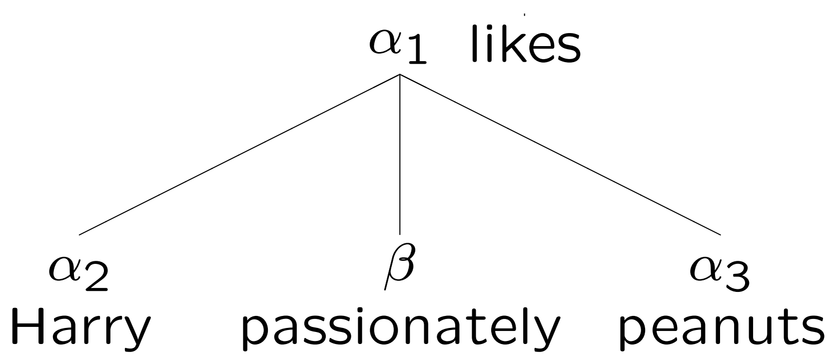


TAG Building Blocks

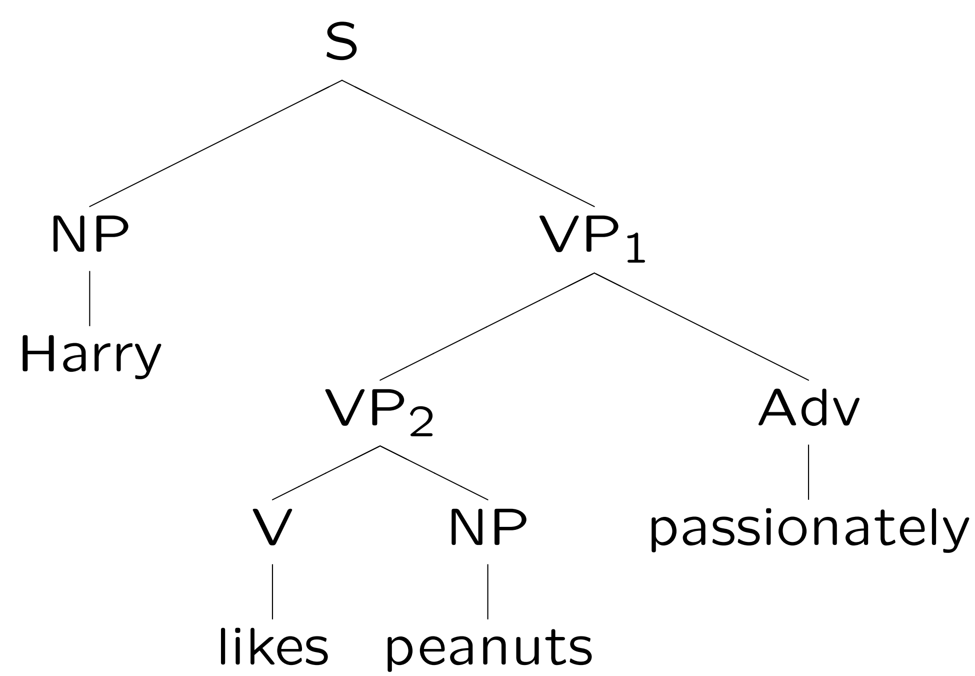
- Auxiliary trees for *adjunction*
- Adds extra power beyond CFG



Derivation Tree



Derived Tree



Semantics

$$Harry(x) \wedge likes(e, x, y) \wedge peanuts(y) \wedge passionately(e)$$