# Parsing

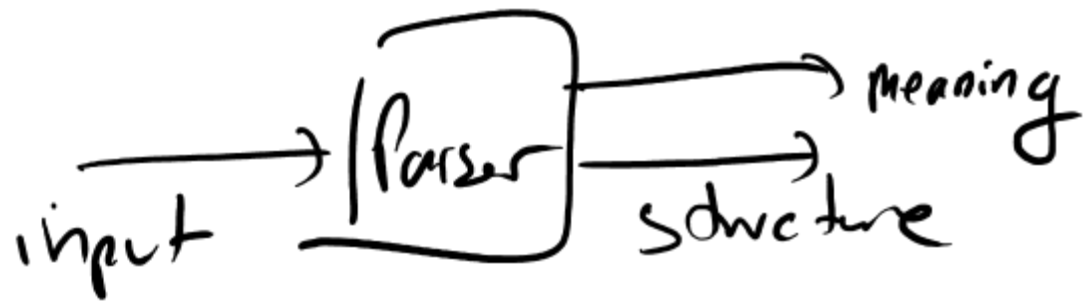
Boğaziçi Linguistics
LING 488

Cem Bozşahin

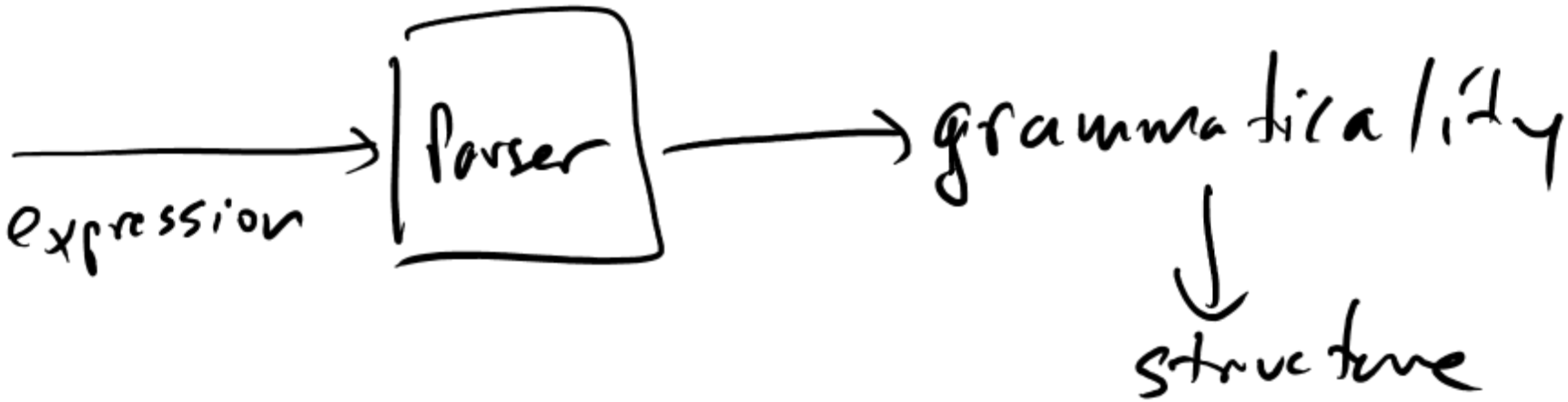
- Parsing is the process of revealing the structure of an expression.

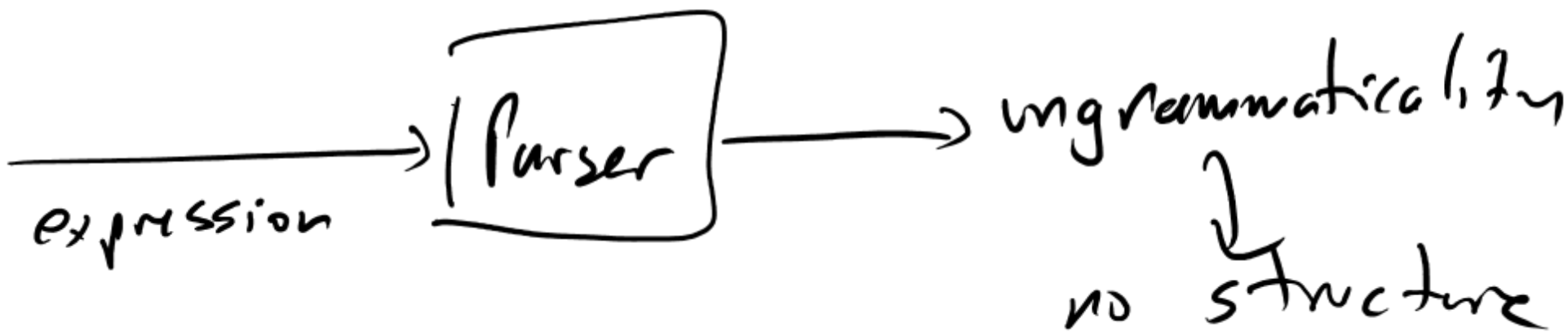
view 1


view 2




- View 2 also asks why we build that structure


expression
Parser
grammaticality
structure


expression
Parser
ungrammaticality
no structure

# Linguistic takes on parsing

- Chomsky (2000): Parsing is not a reflex.
  - It does not have to have an algorithm.
  - Recently, there are efforts to take numeration to spell-out in the Minimalist Program (Berwick & Stabler 2019).
- Garrett, Fodor (1980+): Parsing is a reflex. Try turning it off!

- what would it look like if it had an algorithm?

- Parsing is a major cottage industry in Computational linguistics.

- It is a field by itself in Computer Science.

Long story short: Any phrase-structure grammar can be turned into something with binary branching.

- This is called the Chomsky Normal form. 1950s
- X-bar theory is a linguistic version of that. 1970s
- This means one-argument-at-a-time, if there is branching.

- We have known this since Schönfinkel (1920)!

# A formal exercise in binarizing a PSG:

in CNF: $S \rightarrow NP\ VP$

not in CNF: $VP \rightarrow V\ NP\ NP$

not branching: $V \rightarrow gave$

$\Rightarrow$

$S \rightarrow NP\ VP$

$VP \rightarrow X\ NP$

$X \rightarrow V\ NP$

$V \rightarrow gave$

all in CNF

- Categorial grammars are already one-argument-at-a-time.

- That is not mathematical reductionism; it is the most natural way to treat ALL categories as FUNCTIONS (not labels).

- The added benefit is the ability to derive the meaning as we parse.

- The simplest and most common binary parser is CKY.
- Cocke-Kasami-Younger (1967-1970)
- It is a rediscovery. Hiroo Sakai invented it in 1961. (also, CYK)
- It works on MODELS of grammar, using bottom-up parsing and dynamic programming.

## The idea:

- For an expression of length $n$, make a matrix of size $n$.
- It is enough for the table to be a lower-triangular matrix, since the upper-triangle is not needed — we will see why.
- Fill every entry with combinations of the form $n = x + y$ but not $n = y + y + z$ etc.

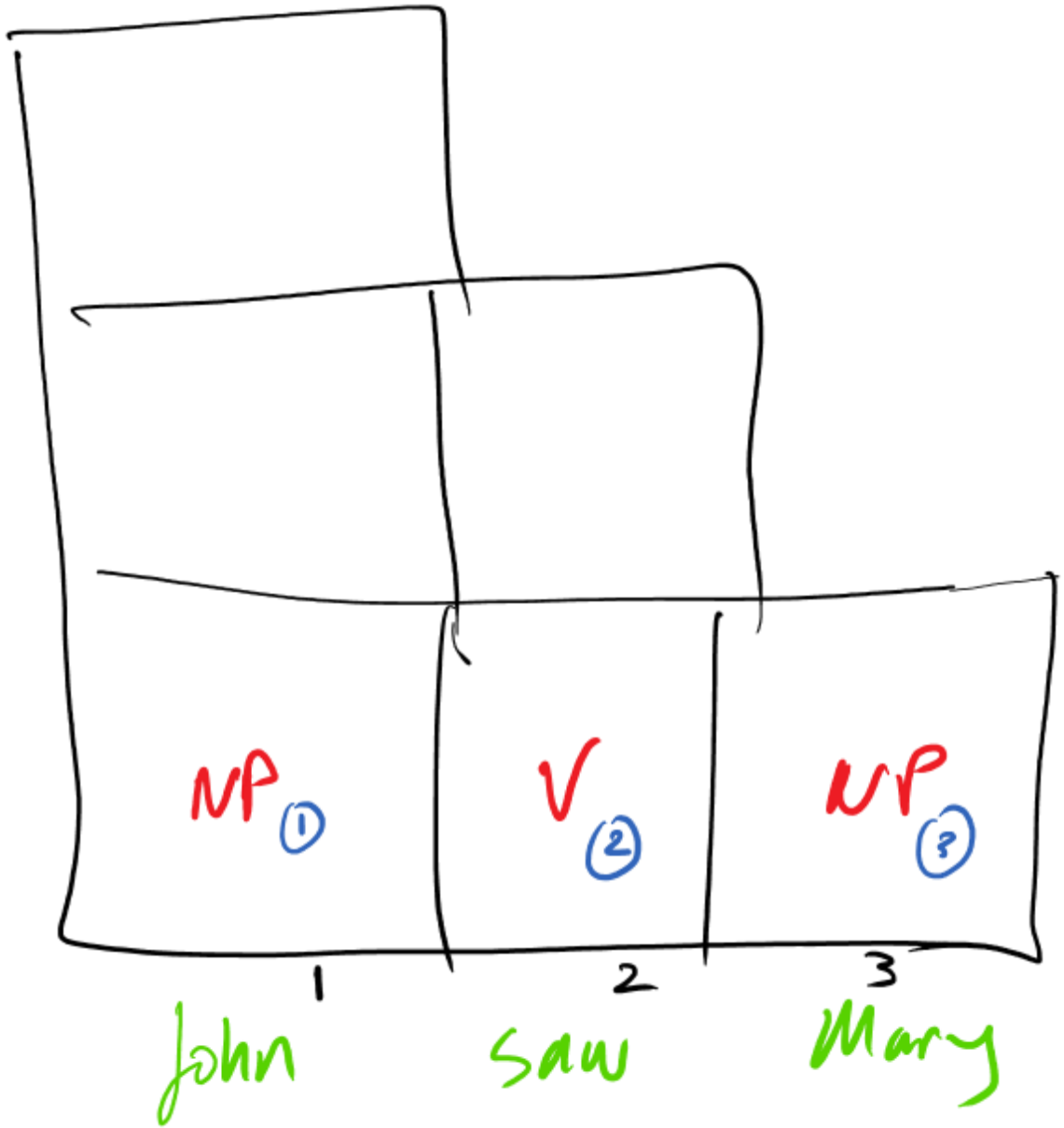
For example, if we have an expression of length 5:

- constituents of length 1: lexical categories
- " " " 2: 1+1 (from pos. 1)
- " " " 3: 1+2, 2+1 starting from position 1 and 2. (But not 1+1+1)
- " " " 4: 2+2, 1+3, 3+1 (but not 2+1+1, 1+2+1 etc.)
- " " " 5: 2+3, 3+2, 1+4, 4+1

# Phrase structure example:

S → NP VP
NP → John | Mary
VP → V NP
V → saw

Start like this:




S → NP VP

① NP → John

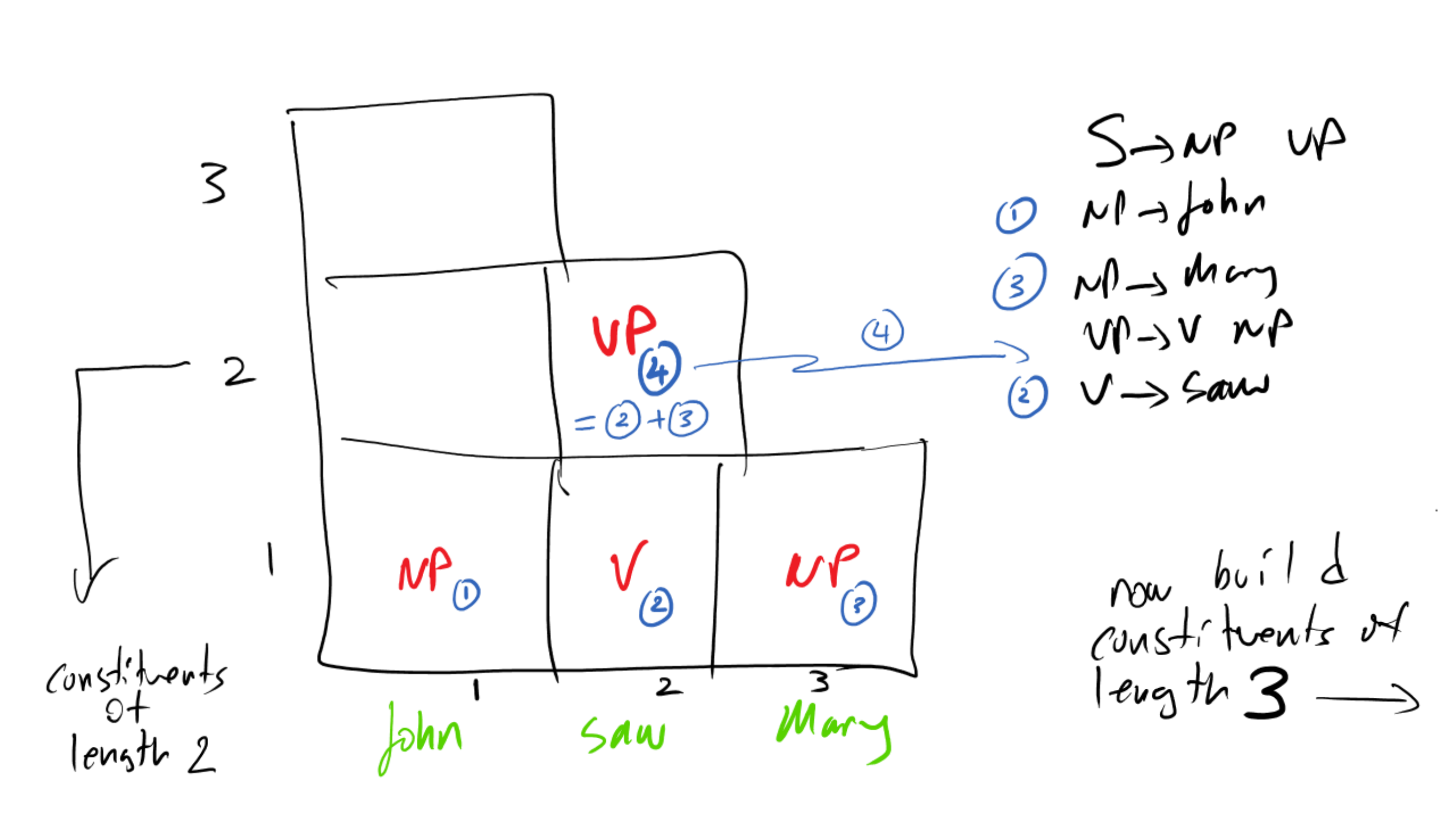
③ NP → Mary

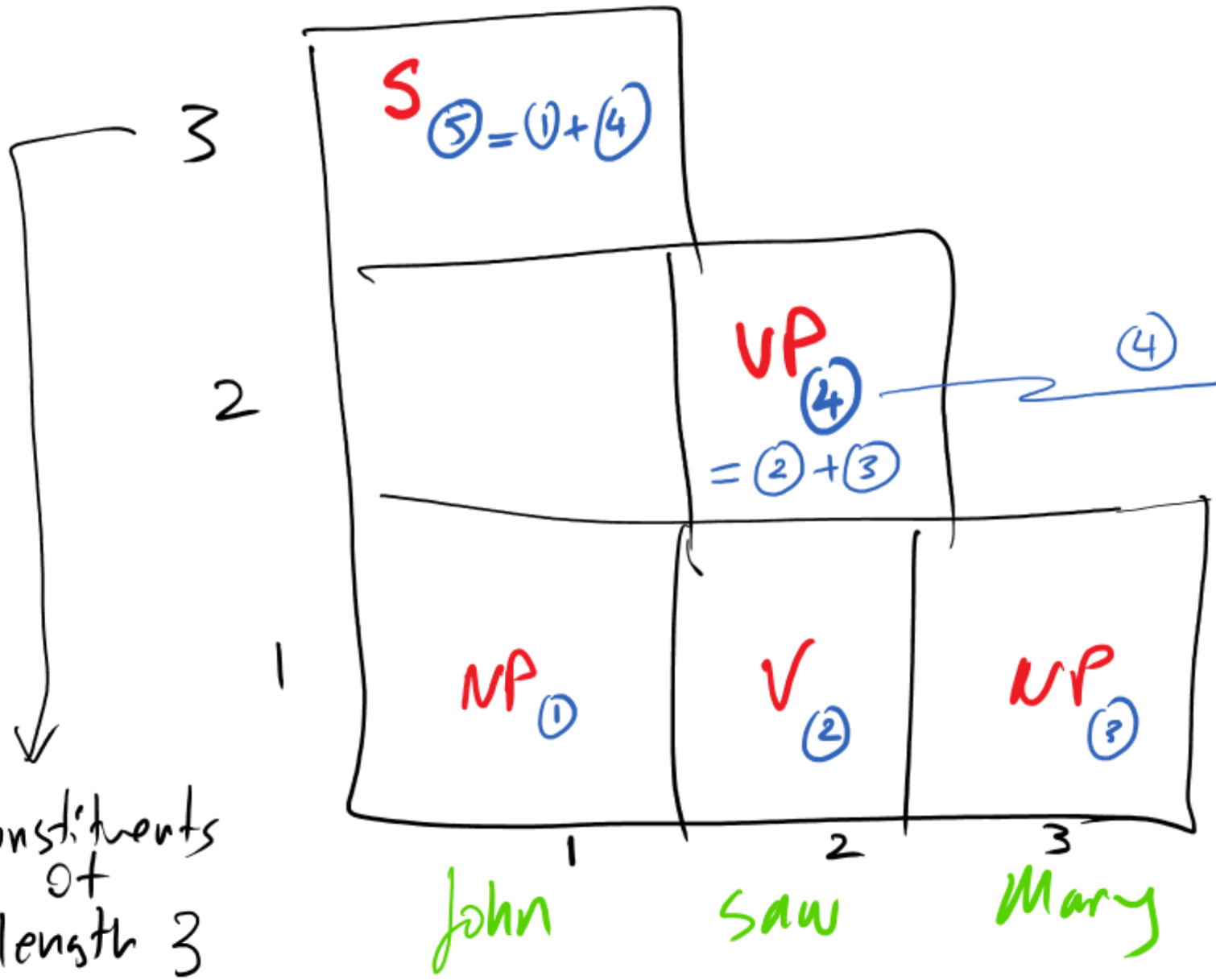
VP → V NP

② V → saw

constituents of length 1

now build constituents of length 2 →


3
2
1
VP
(4)
= (2) + (3)
NP (1)
V (2)
NP (3)
1
2
3
John
saw
Mary
S → NP VP
(1) NP → John
(3) NP → Mary
VP → V NP
(2) V → saw
(4)
constituents of length 2
now build constituents of length 3 →




S → NP VP

(1) NP → John

(3) NP → Mary

VP → V NP

(2) V → saw

(4)

Length 3 constituents:

3 = 1 + length 2

3 = length 2 + 1

start position

end pos.

# CKY for Categorial grammar

: categories are pairs of command rel.

John :: $np : \bar{J}'$ ①

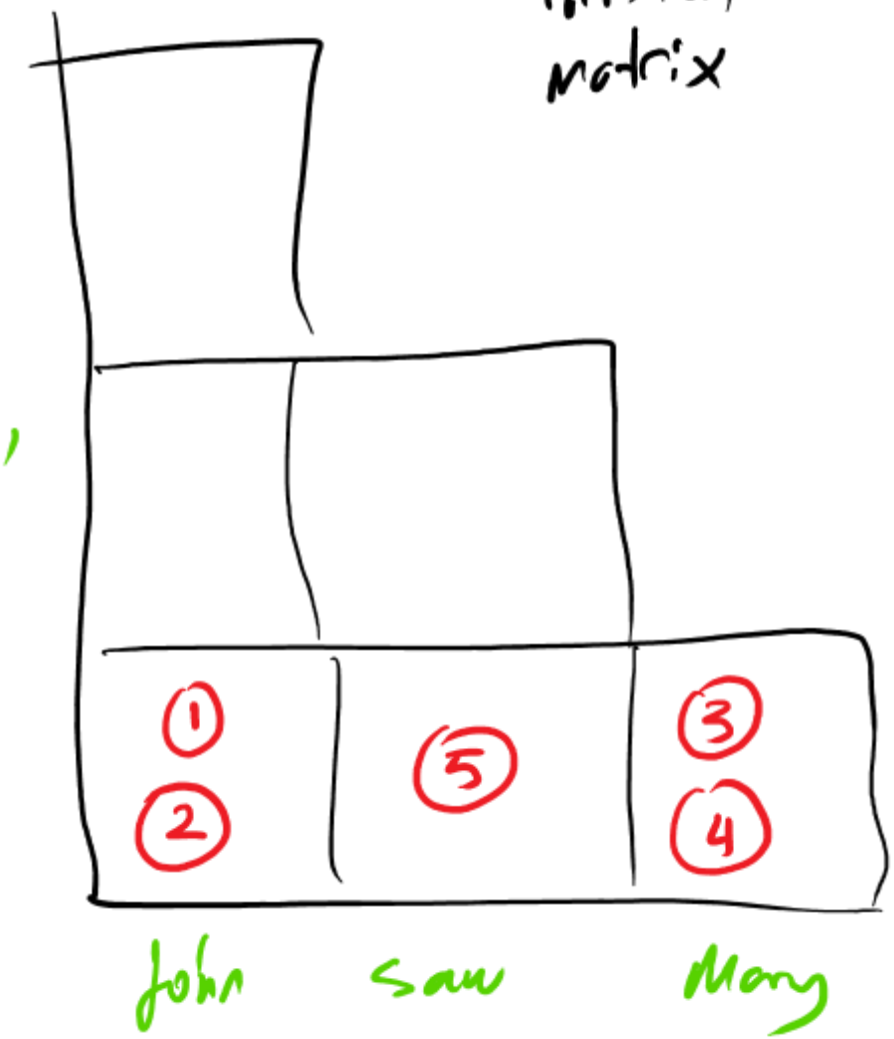
:: $s/(s\backslash np) : \lambda p.\, p\ \bar{J}'$ ②

Mary :: $np : m'$ ③

:: $(s\backslash np)\backslash((s\backslash np)/np) : \lambda p.\, p\ \bar{J}'$ ④

saw :: $(s\backslash np)/np : \lambda x\, \lambda y.\, see'\, x\ y$ ⑤

initial matrix



"Rules" : $X/Y:f \quad Y:a \rightarrow X:fa$ FA

$Y:a \quad X\backslash Y:f \rightarrow X:fa$ BA

$X/Y:f \quad Y/Z:g \rightarrow X/Z: \lambda z.f(gz)$ FC

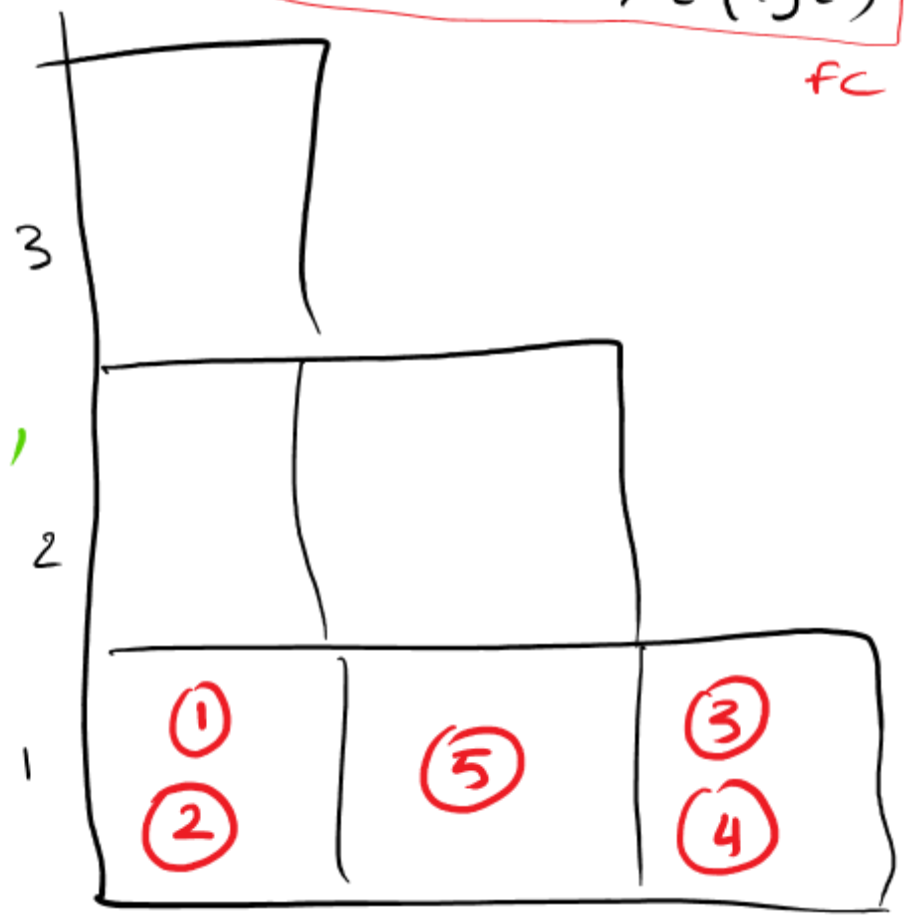
John :: $NP:\bar{J}'$ ①

:: $S/(S\backslash NP):\lambda p.p\,\bar{J}'$ ②

Mary :: $NP:\bar{m}'$ ③

:: $(S\backslash NP)\backslash((S\backslash NP)/NP):\lambda p.p\,\bar{J}'$ ④

saw :: $(S\backslash NP)/NP:\lambda x\lambda y.see'\,x\,y$ ⑤




FA: forward application FC: forward composition
BA: backward "

# CKY for Categorial grammar

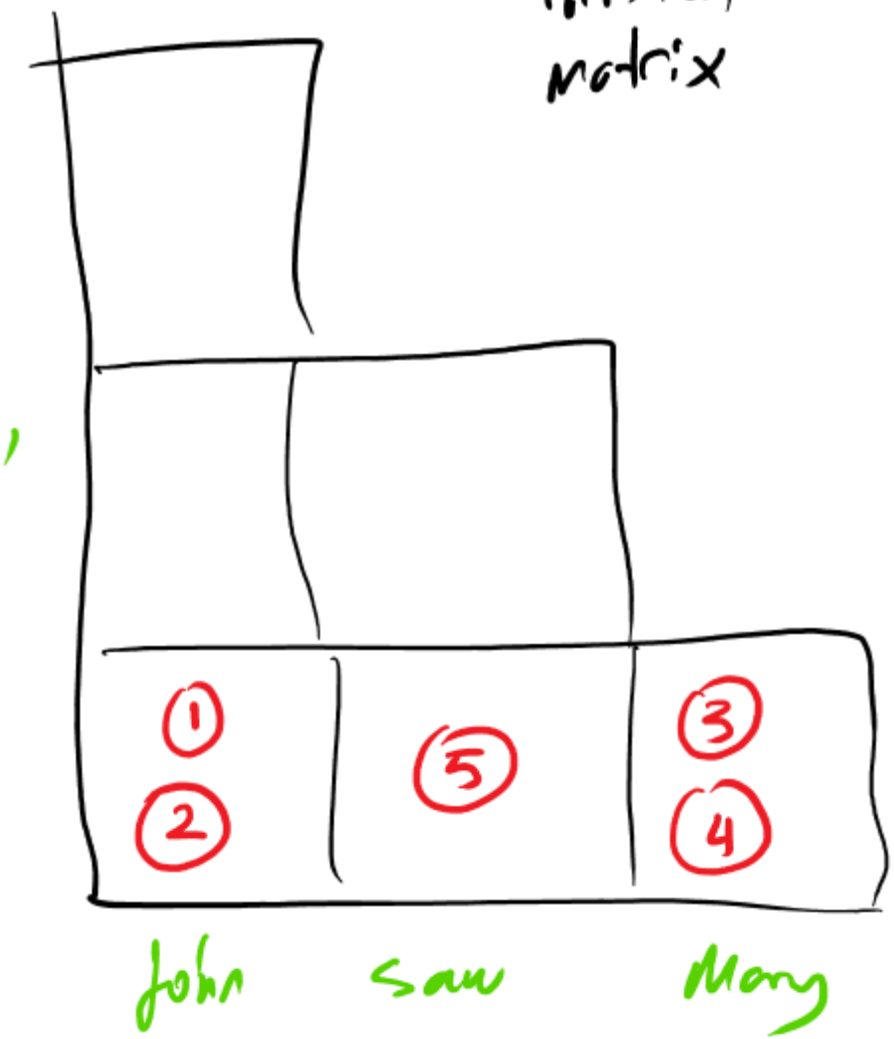
John :: $np : \bar{J}'$ ①

:: $s/(s\backslash np) : \lambda p.\, p\ \bar{J}'$ ②

Mary :: $np : m'$ ③

:: $(s\backslash np)\backslash((s\backslash np)/np) : \lambda p.\, p\ \bar{J}'$ ④

saw :: $(s\backslash np)/np : \lambda x\, \lambda y.\, see'\, x\ y$ ⑤



# CKY for Categorial grammar

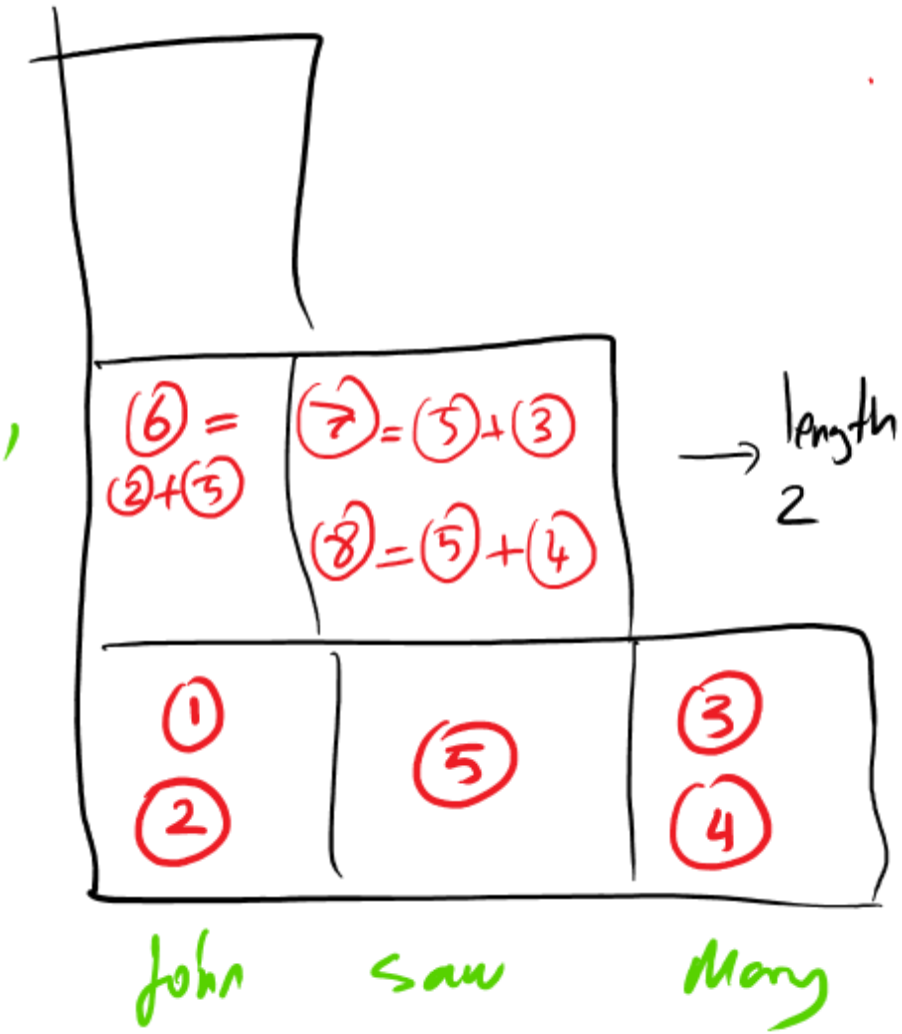
John :: $np : j'$ ①

:: $s/(s\backslash np) : \lambda p.\, p\, j'$ ②

Mary :: $np : m'$ ③

:: $(s\backslash np)\backslash((s\backslash np)/np) : \lambda p.\, p\, j'$ ④

saw :: $(s\backslash np)/np : \lambda x\, \lambda y.\, see'\, x\, y$ ⑤



⑥ $^{FC}$ = $s/(s\backslash np)\ (s\backslash np)/np \rightarrow s/np$

⑦ $^{FA}$ = $(s\backslash np)/np\ \ np \rightarrow s\backslash np$

⑧ $^{BA}$ = $(s\backslash np)/np\ \ (s\backslash np)\backslash((s\backslash np)/np)$
$\rightarrow s\backslash np$

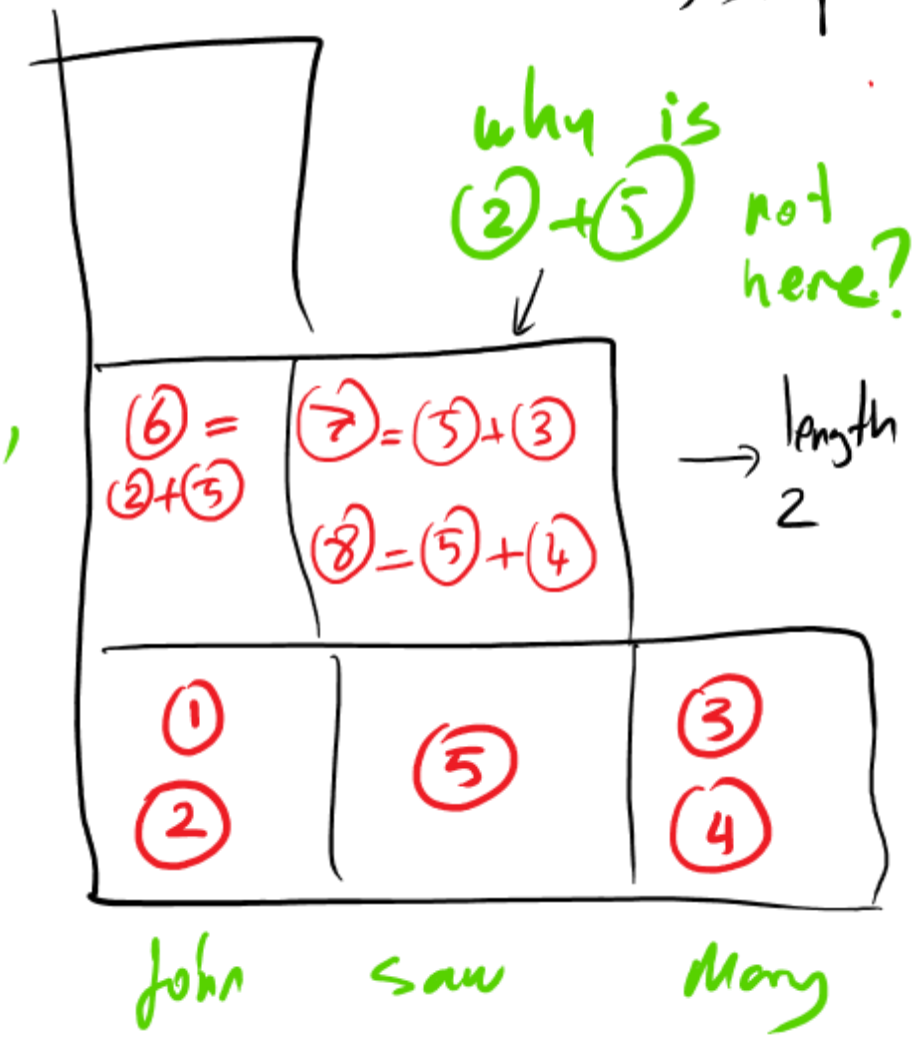
John :: $np : \bar{J}'$ ①

:: $s/(s\backslash np) : \lambda p.\, p\ \bar{J}'$ ②

Mary :: $np : m'$ ③

:: $(s\backslash np)\backslash((s\backslash np)/np) : \lambda p.\, p\ \bar{J}'$ ④

saw :: $(s\backslash np)/np : \lambda x\, \lambda y.\, see'\, x\ y$ ⑤



⑥ (FC) = $s/(s\backslash np)\ \ (s\backslash np)/np \rightarrow s/np$

↑ syntax + semantics?

⑦ (FA) = $(s\backslash np)/np\ \ np \rightarrow s\backslash np$

⑧ (BA) = $(s\backslash np)/np\ \ (s\backslash np)\backslash((s\backslash np)/np) \rightarrow s\backslash np$

⑥ =

$s/(s\backslash np)\ \ (s\backslash np)/np \rightarrow s/np$

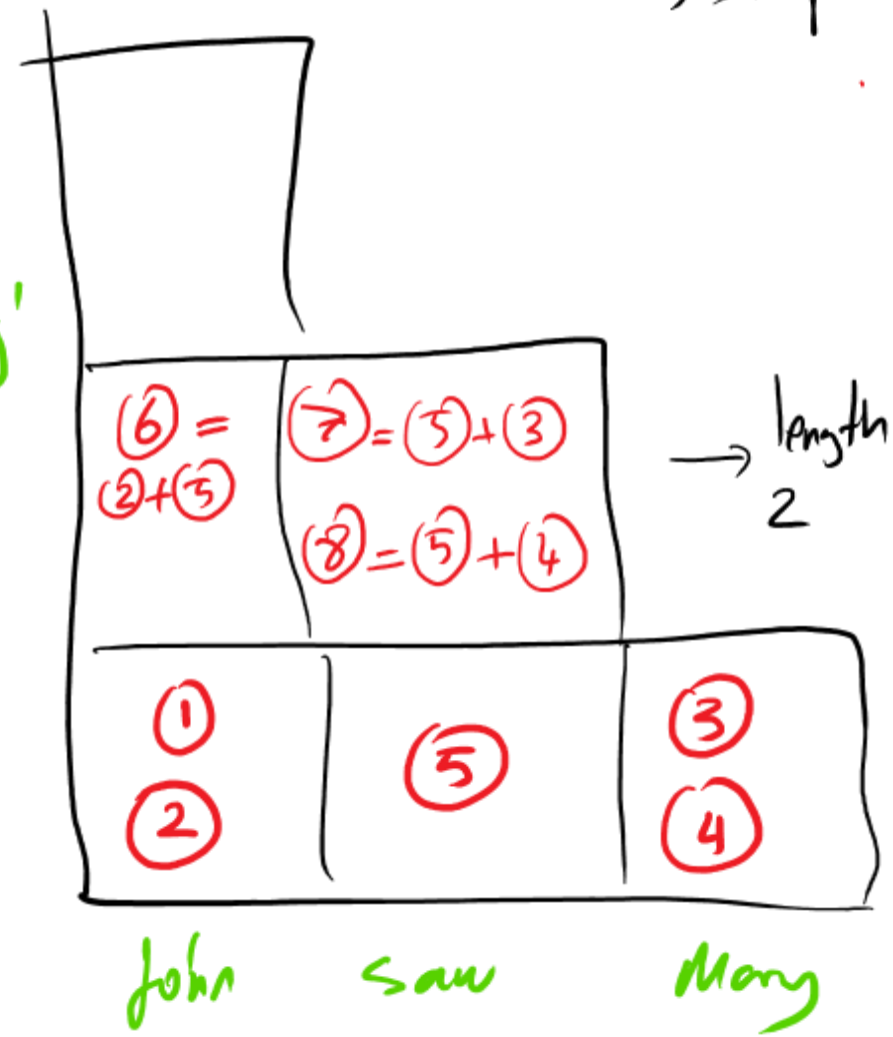
: $\lambda p.\, p\, j'$ : $\lambda x.\lambda y.\, see'\, x\, y$ $\lambda z.\, see'\, z\, j'$

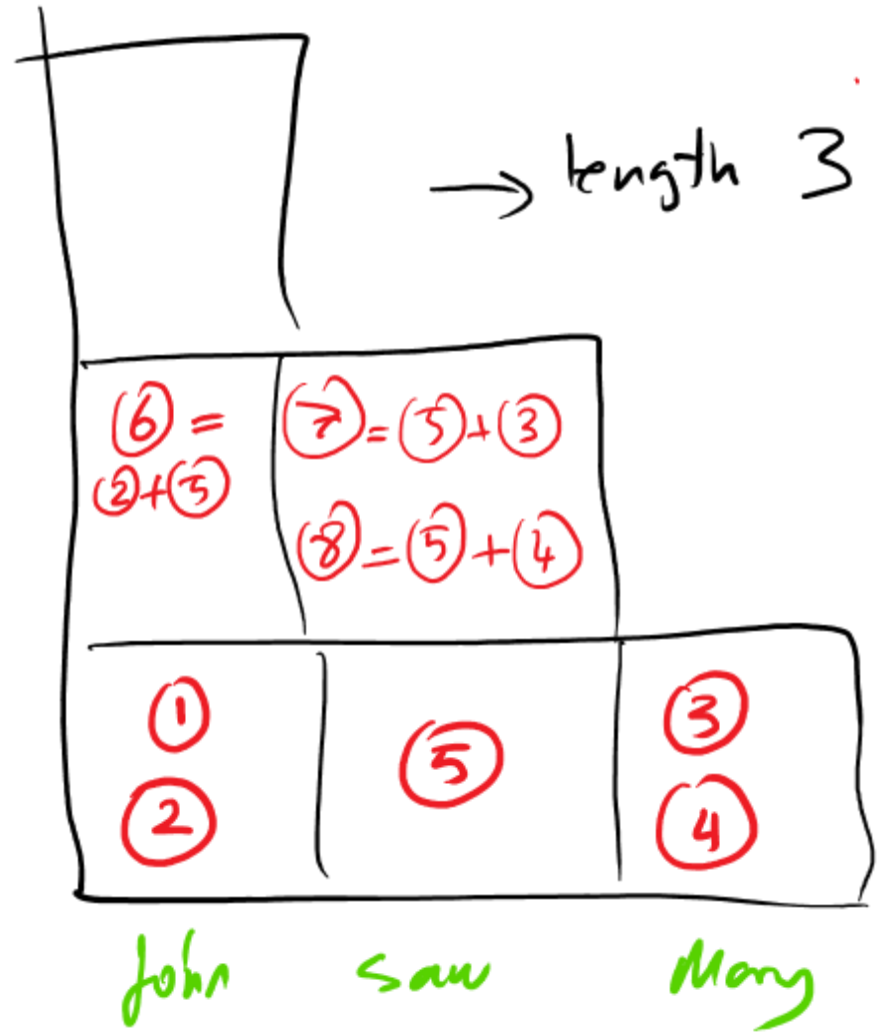
how do we get that?

Composition is:

$\lambda z.\ (\lambda p.\, p\, j')\,[\,(\lambda x.\lambda y.\, see'\, x\, y)\, z\,] =$

$\lambda z.\ (\lambda p.\, p\, j')\,[\,\lambda y.\, see'\, z\, y\,] = \lambda z.\ (\lambda y.\, see'\, z\, y)\, j'$




→ length 3
⑥ = ②+③
⑦ = ⑤+③
⑧ = ⑤+④
①
②
⑤
③
④
John
saw
Mary

"Rules" : $X/Y:f \quad Y:a \rightarrow X:fa$ FA

$Y:a \quad X\backslash Y:f \rightarrow X:fa$ BA

$X/Y:f \quad Y/Z:g \rightarrow X/Z: \lambda z.f(gz)$ FC

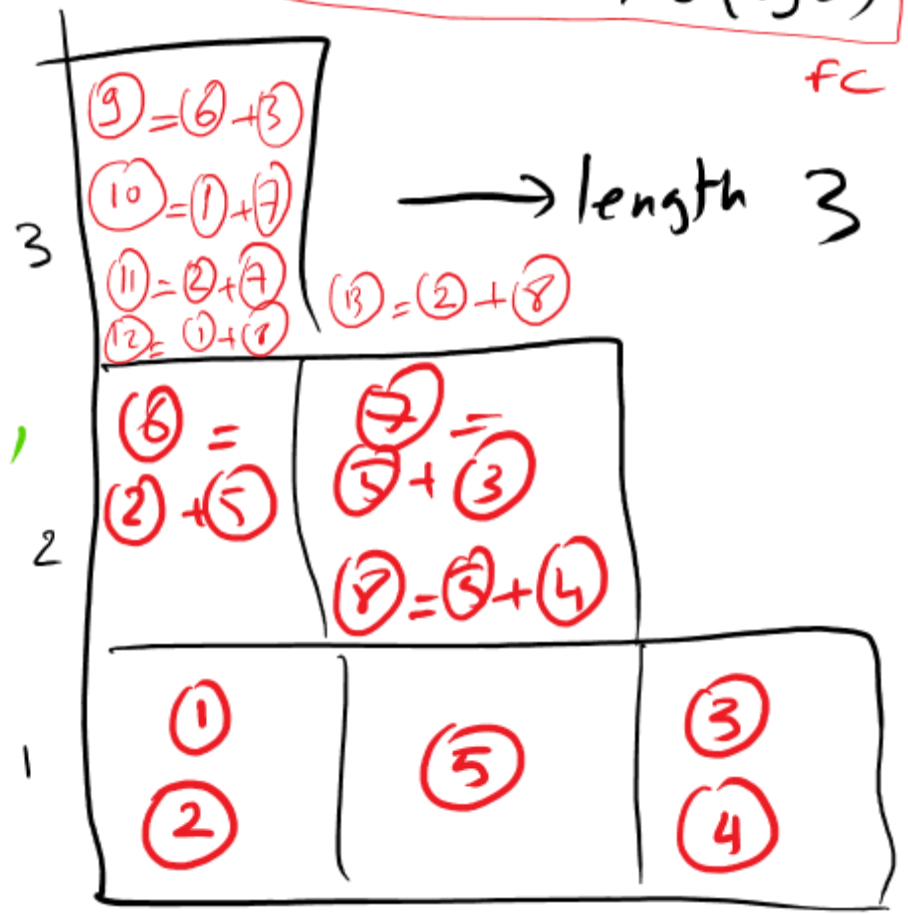
John :: $np: J'$ ①

:: $s/(s\backslash np): \lambda p.p\,J'$ ②

Mary :: $np: m'$ ③

:: $(s\backslash np)\backslash((s\backslash np)/np): \lambda p.p\,J'$ ④

saw :: $(s\backslash np)/np: \lambda x \lambda y.see'\,x\,y$ ⑤




FA: forward application FC: forward composition

BA: backward "

for example,

⑫ = ① + ⑧ = $np: J'$ $\quad s\backslash np: \lambda x.\ see'\ m'\ x$

$\xrightarrow[BA]{}$ $S: see\ m'\ J'$

⑪ = ② + ⑦ = $s/(s\backslash np): \lambda p.\ p\ J'$ $\quad s\backslash np: \lambda x.\ see'\ m'\ x$

$\xrightarrow[FA]{}$ $S: see'\ m'\ J'$

$\overset{\equiv}{=}$ $(\lambda p.\ p\ J')\ (\lambda x.\ see'\ m'\ x)$

$= (\lambda x.\ see'\ m'\ x)\ J = see'\ m'\ J'$

- CKY produces a PARSE FOREST. (all results)
- All binary combinations can be read off the CKY matrix. (dynamic programming)
- It is ONE MODEL of mechanics of linguistic analysis:
  - Bottom-up parsing

(analysis is a theoretical concept. Combination and decomposition are MODELING concepts)

- Parsers can solve mechanical aspects of who does what to whom, GIVEN category choice!!

- They are objects of modeling, NOT objects of grammar.