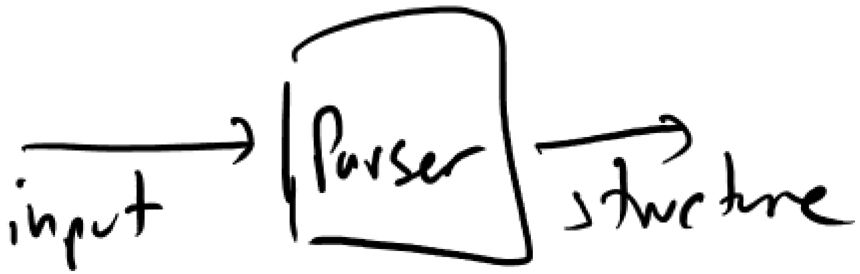
Parsing

Boğaziçi Linguistics
LING 488

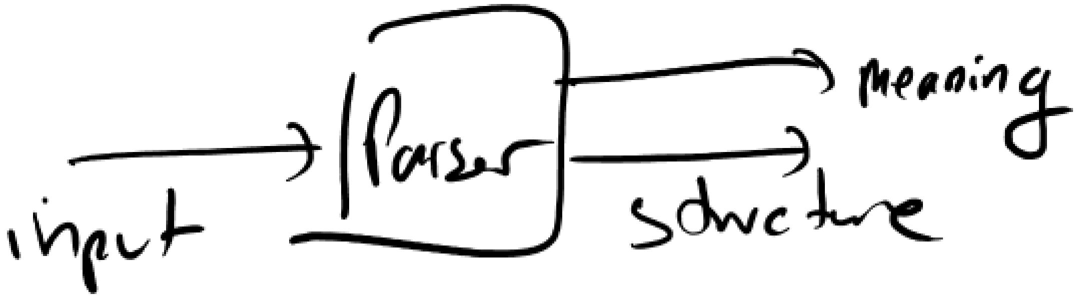
Cem Bozşahin

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

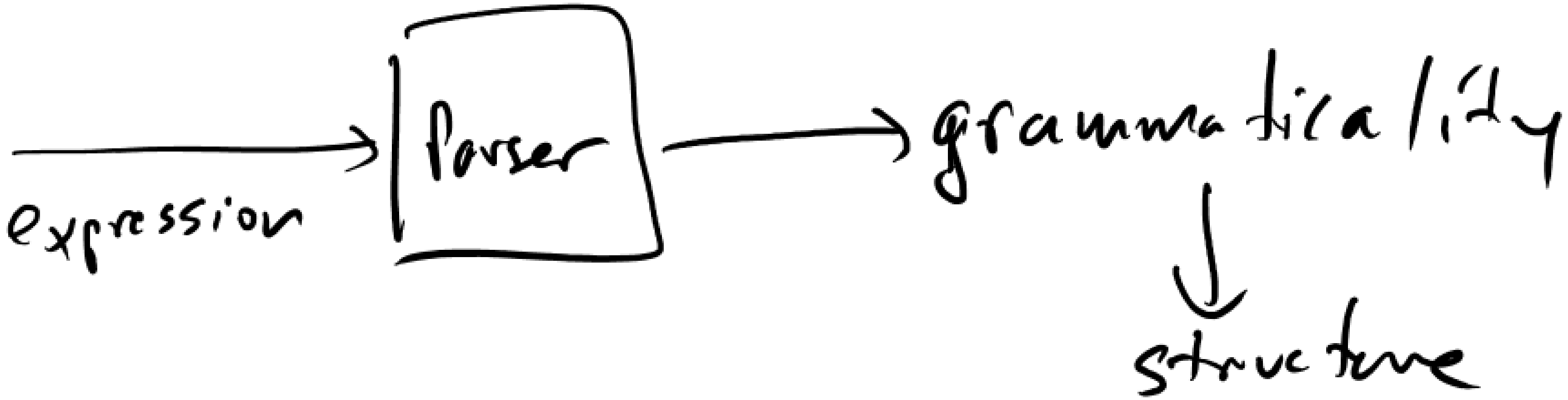
view 1

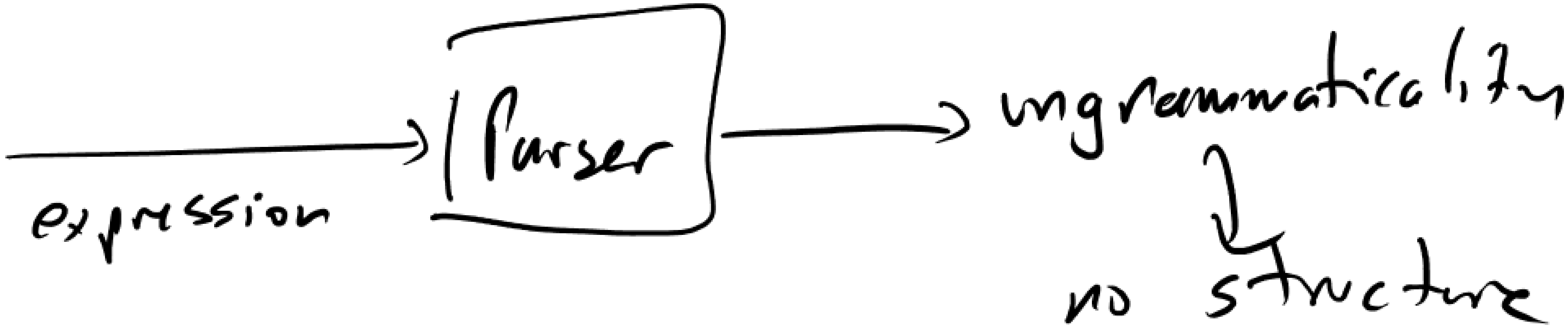


view 2



- 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.

1950s

- This is called the Chomsky Normal form.

1970s

- X-bar theory is a linguistic version of that.
- 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	$S \to NP\ VP$		
not in CNF	$VP \to V\ NP\ NP$	$\Longrightarrow$	$S \to NP\ VP$ $VP \to X\ NP$ $X \to V\ NP$ $V \to$ gave
not branching	$V \to$ gave		all in CNF

- Categorial grammars are already one-argument-at-a-time.
- That is not mathematical simplicity; 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) (also CYK)
- It is a rediscovery. Hiroo Sakai invented it in 1961.

- It works on MODELS of grammar, using bottom-up parsing and dynamic programming.

The idea:

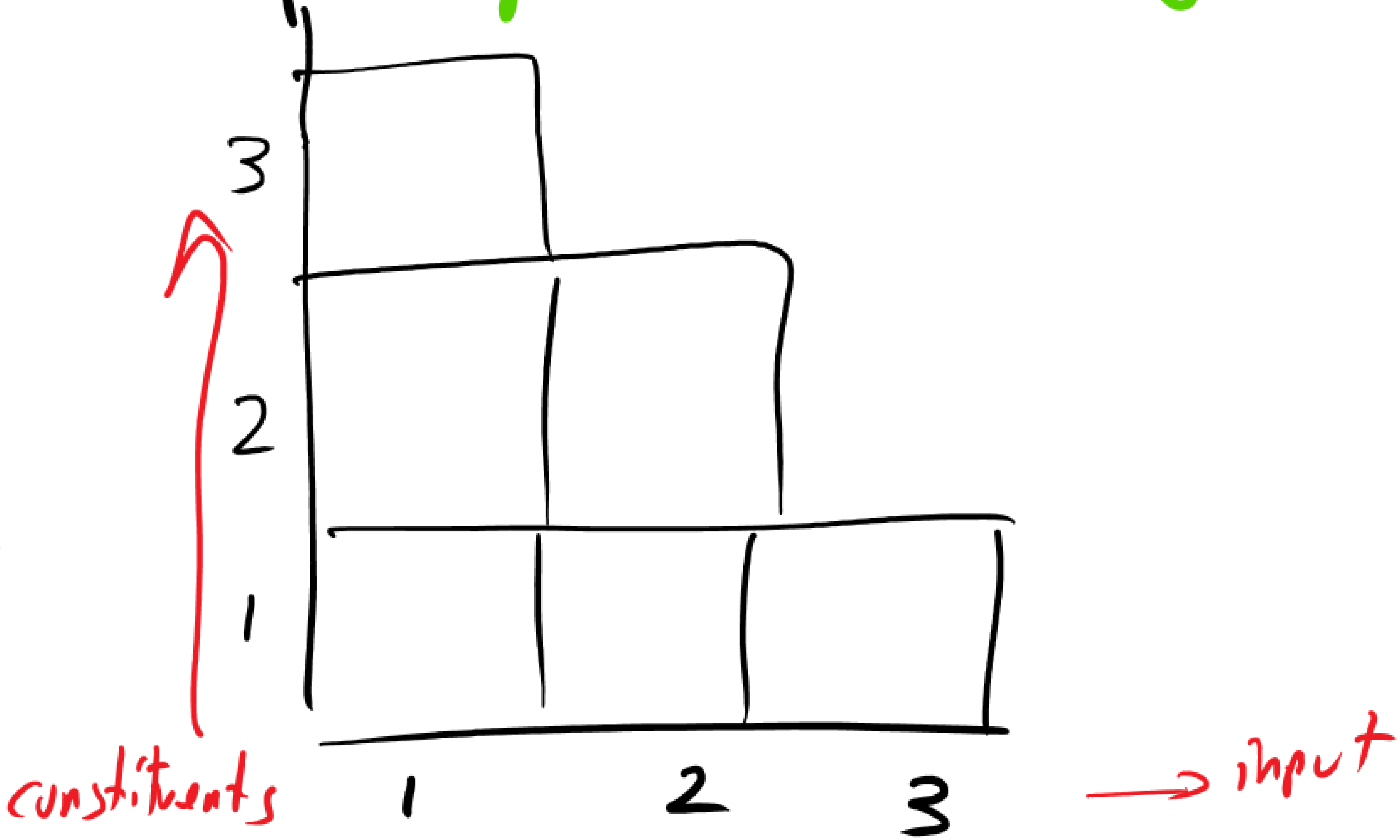
- For an expression of length n, make a matrix of size n.
- It is enough for the table to be a lower-triangular matrix, since the upper-triangle is not needed — we will see why.
- Fill every entry with combinations of $1+k$ or $k+1$, $k = 1, \dots, n-1$.

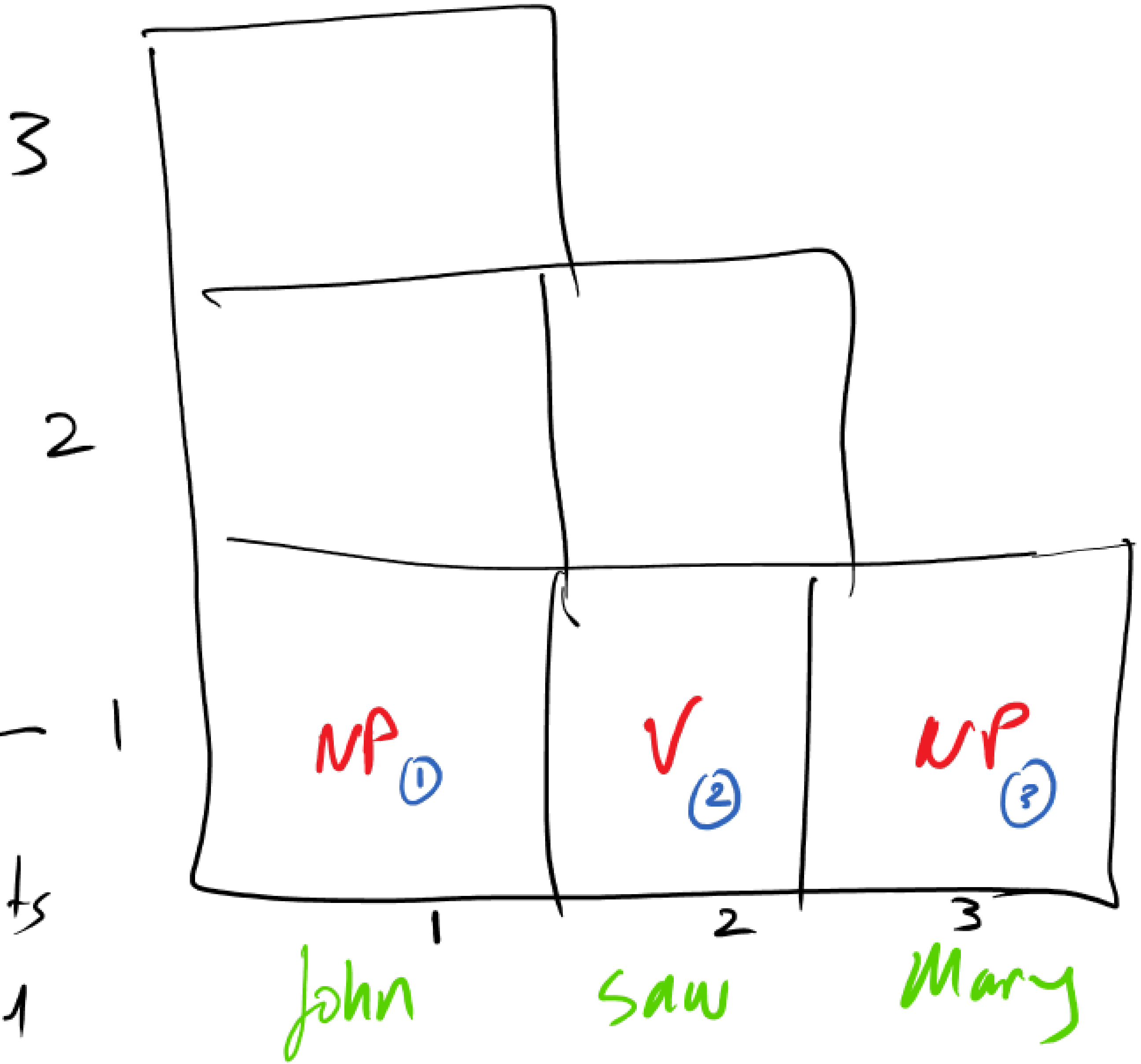
Phrase-structure example:

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

input: John saw Mary

Start like this:





constituents of length 1

S → NP VP

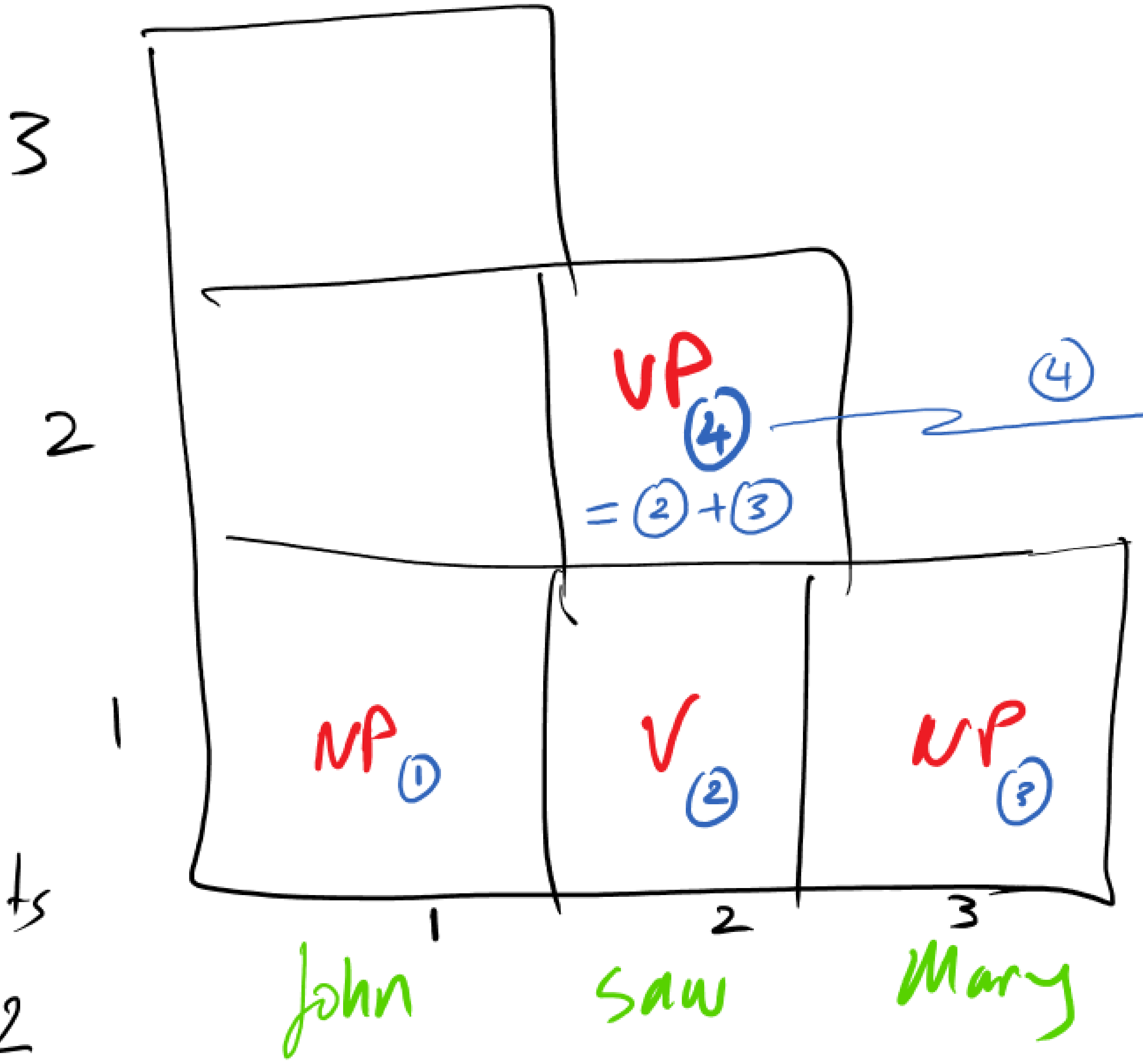
(1) NP → John

(3) NP → Mary

VP → V NP

(2) V → saw

now build constituents of length 2 →



constituents of length 2

S → NP VP

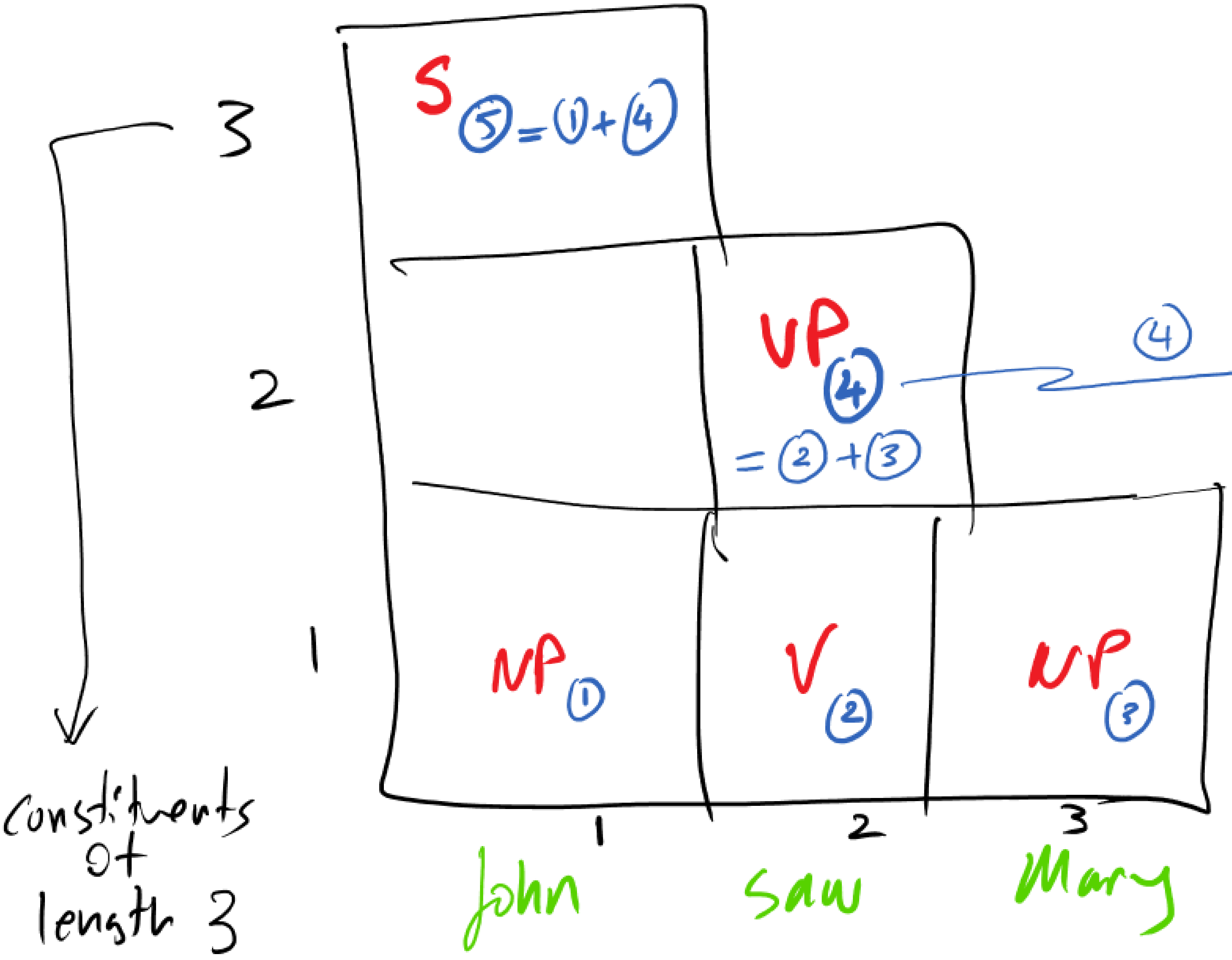
1 NP → John

3 NP → Mary

4 VP → V NP

2 V → saw

now build constituents of length 3 →



$S \to NP\ VP$

1 $NP \to John$

3 $NP \to Mary$

4

$VP \to V\ NP$

2 $V \to saw$

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 :: $np : j'$ 1

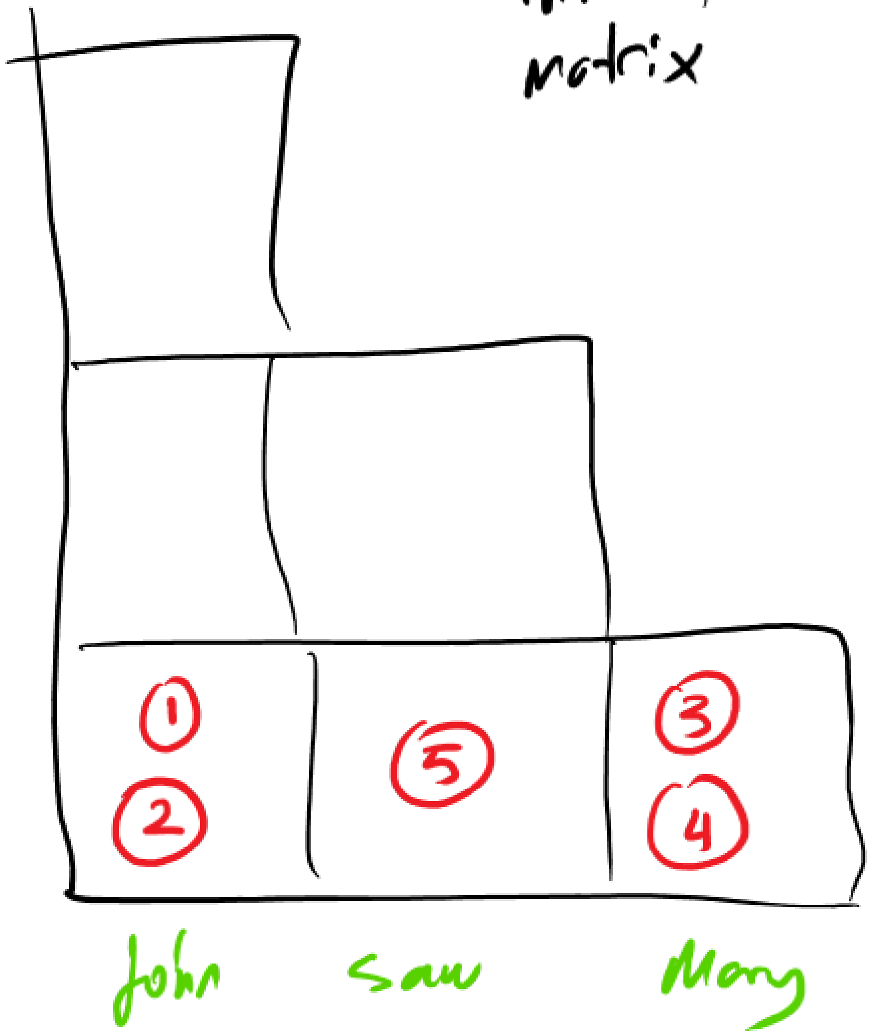
:: $s/(s\backslash np) : \lambda p.\, p\, j'$ 2

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

initial matrix



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

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

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

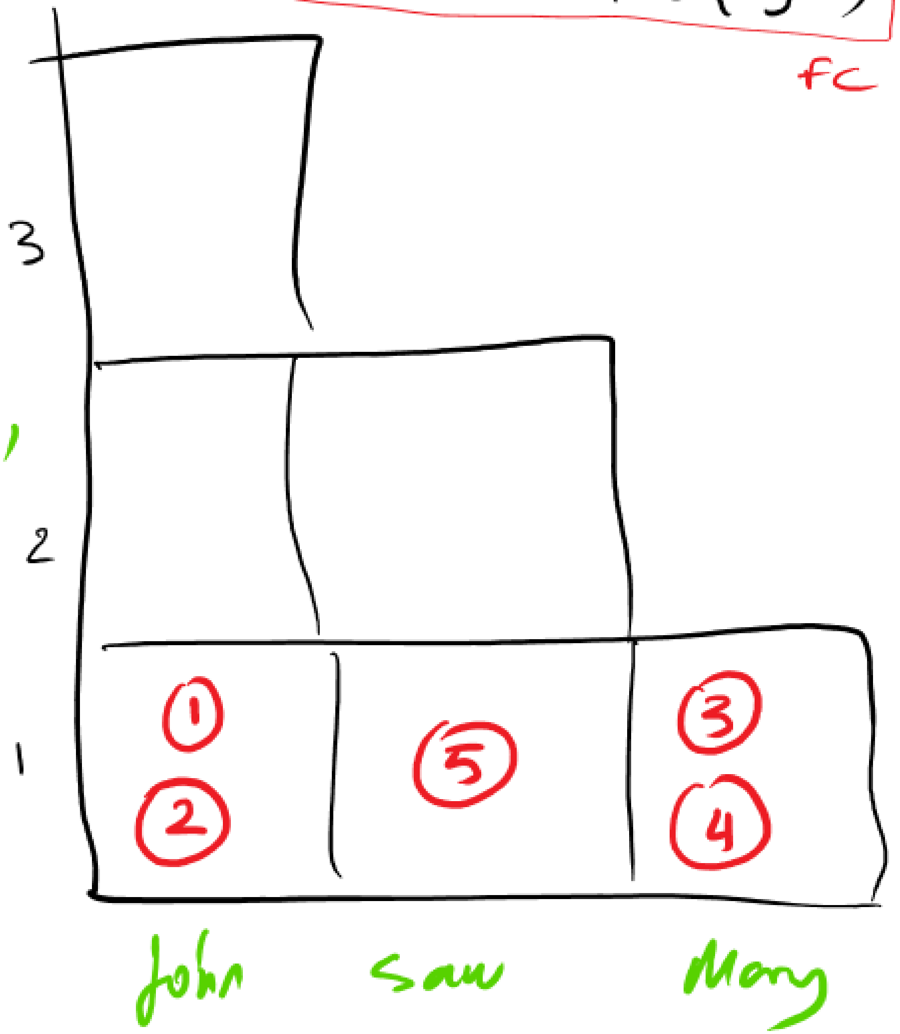
John :: $NP: j'$ 1

:: $S/(S\backslash NP): \lambda p.p\, j'$ 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 "

CKY for Categorial grammar

John :: $np : j'$ ①

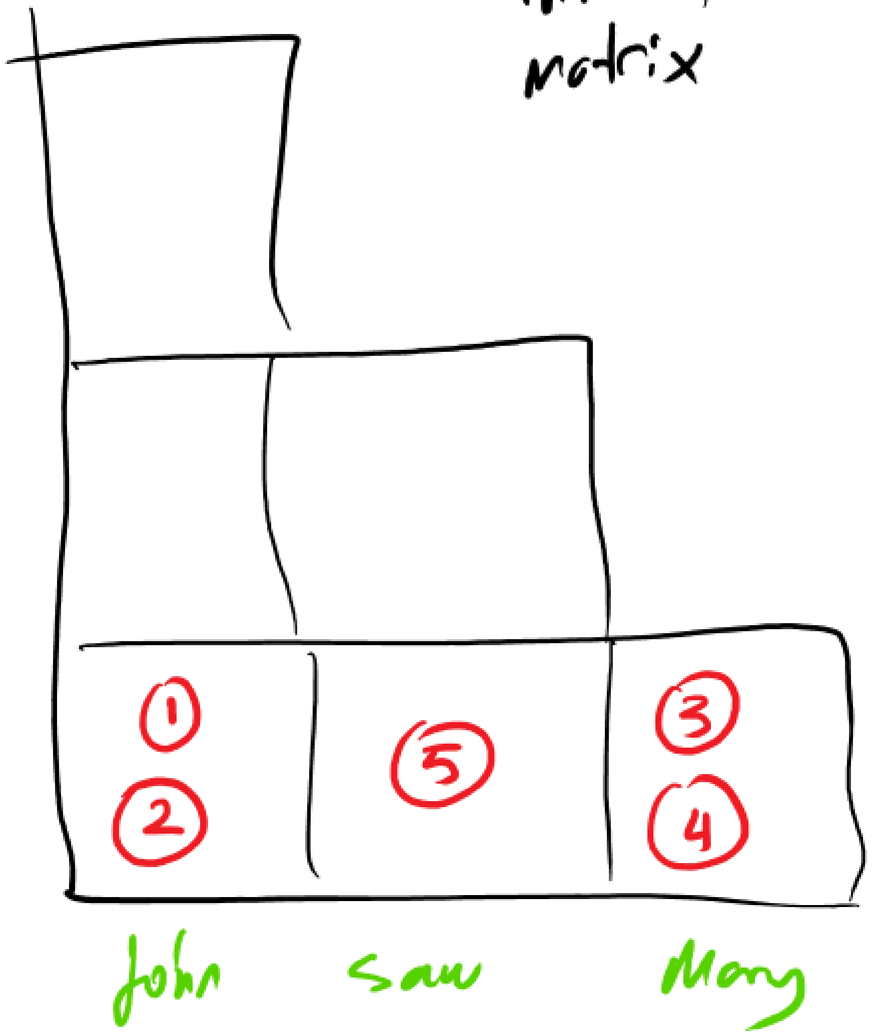
:: $s/(s\backslash np) : \lambda p.p\ j'$ ②

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

initial matrix



CKY for Categorial grammar

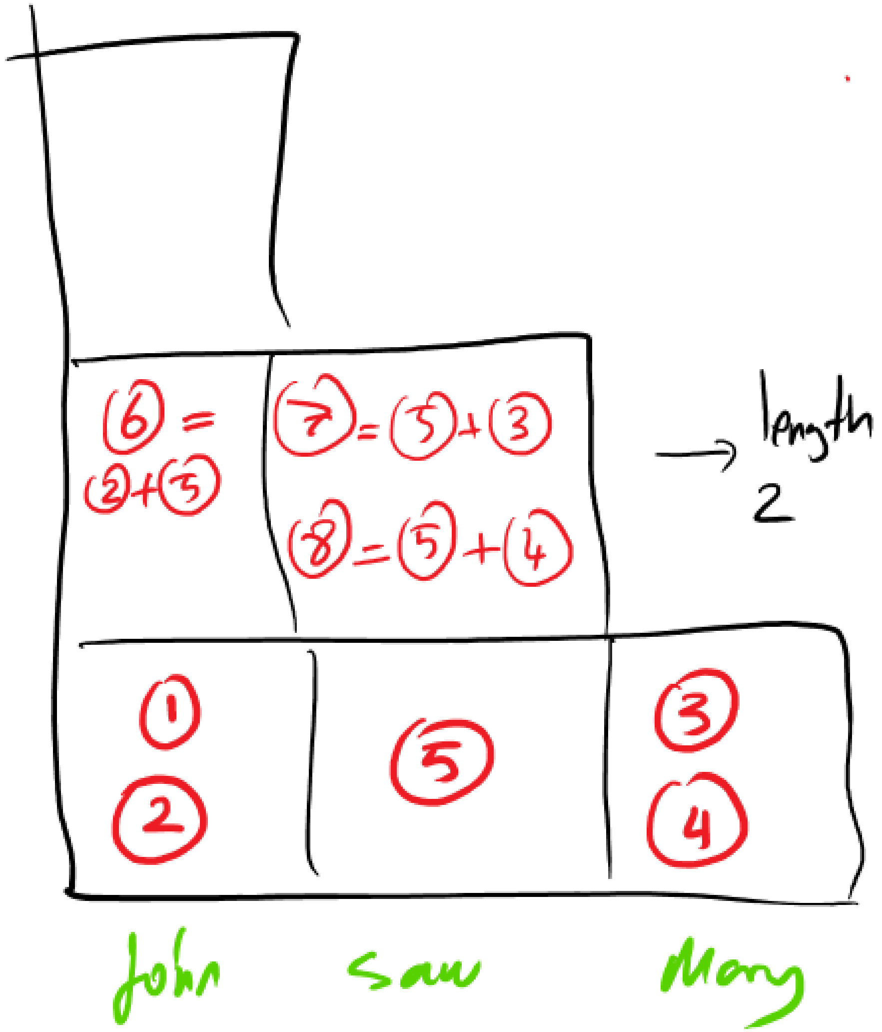
John :: $np : j'$ (1)

:: $s/(s\backslash np) : \lambda p.\, p\, j'$ (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 :: $np : j'$ (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) = (2)+(5)	(7) = (5)+(3) (8) = (5)+(4)	→ length 2
(1) (2)	(5)	(3) (4)
John	saw	Mary

why is (2)+(5) not here?

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

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

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

↑ syntax + semantics?

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

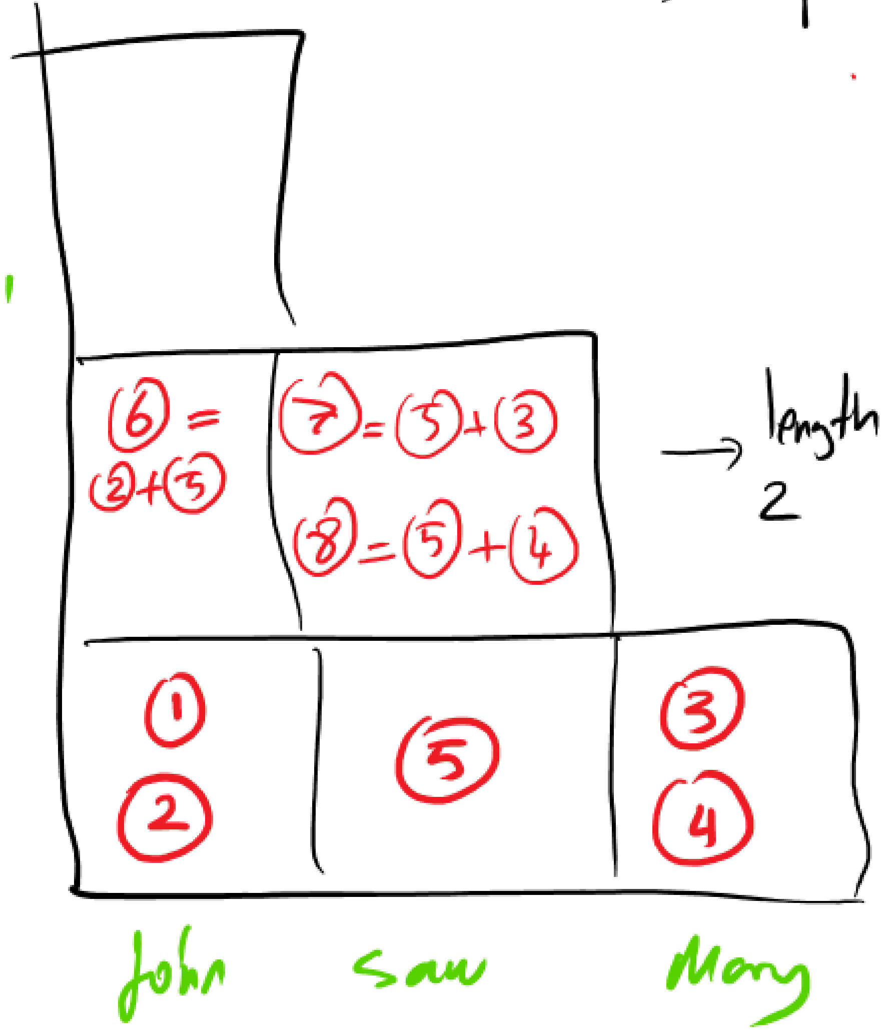
$: \lambda p.\, p\, J'$ $\quad : \lambda x.\lambda y.\, see'\, x\, y \quad \lambda z.\, see'\, z\, J'$

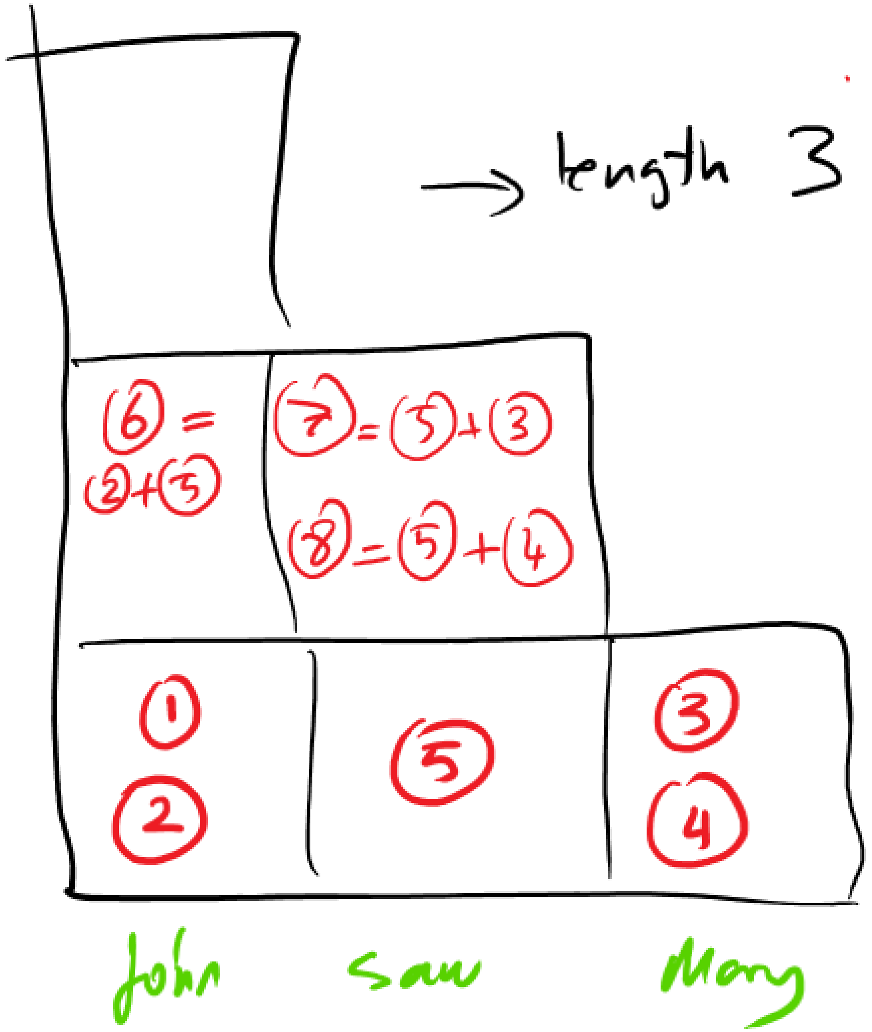
how do we get that?

Composition is:

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

$$\lambda z.\, (\lambda p.\, p\, J')\left[\lambda y.\, see'\, z\, y\right] = \lambda z.\, (\lambda y.\, see'\, z\, y)\, J'$$




length 3
6 = 2 + 5
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

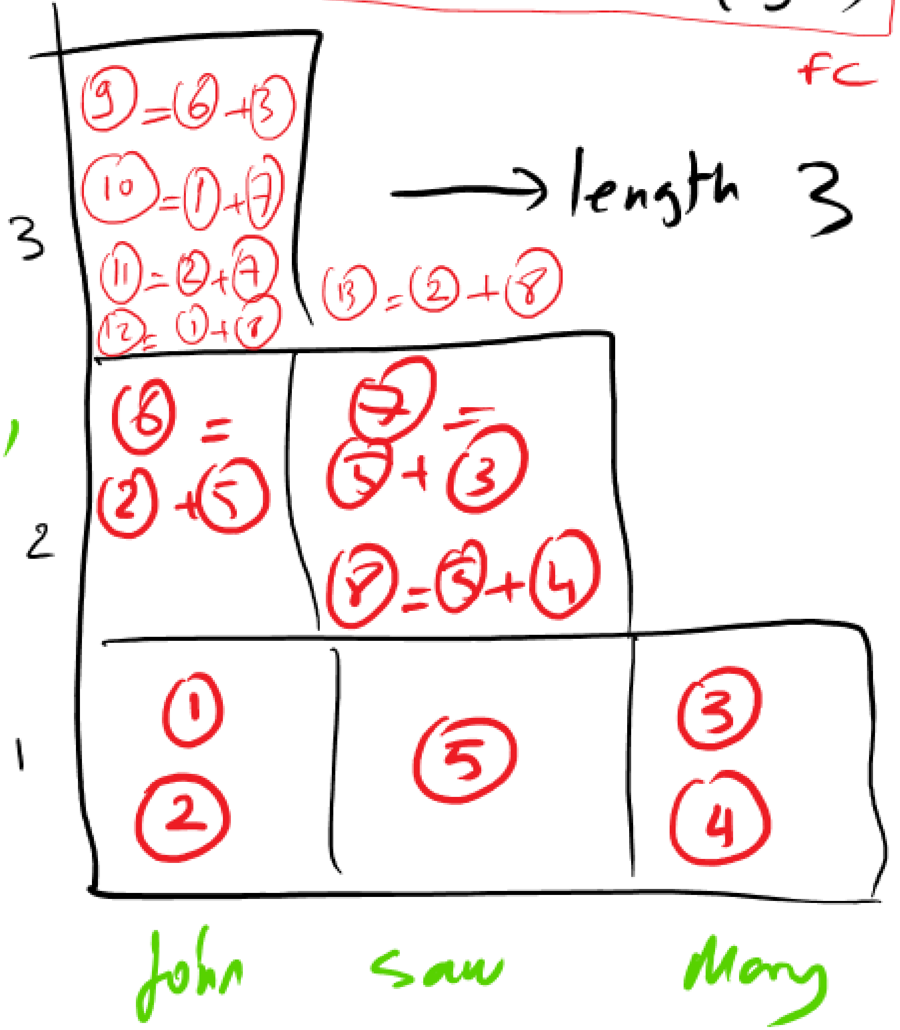
John :: $NP : j'$ (1)

:: $S/(S\backslash NP) : \lambda p.\, p\, j'$ (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) = $s/(s\backslash np) : \lambda p.\, p\, J'$ $\quad s\backslash np : \lambda x.\, see'\, m'\, x$

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

$\Downarrow$

$\equiv$

$(\lambda p.\, p\, J')\,(\lambda x.\, see'\, m'\, x)$

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

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

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