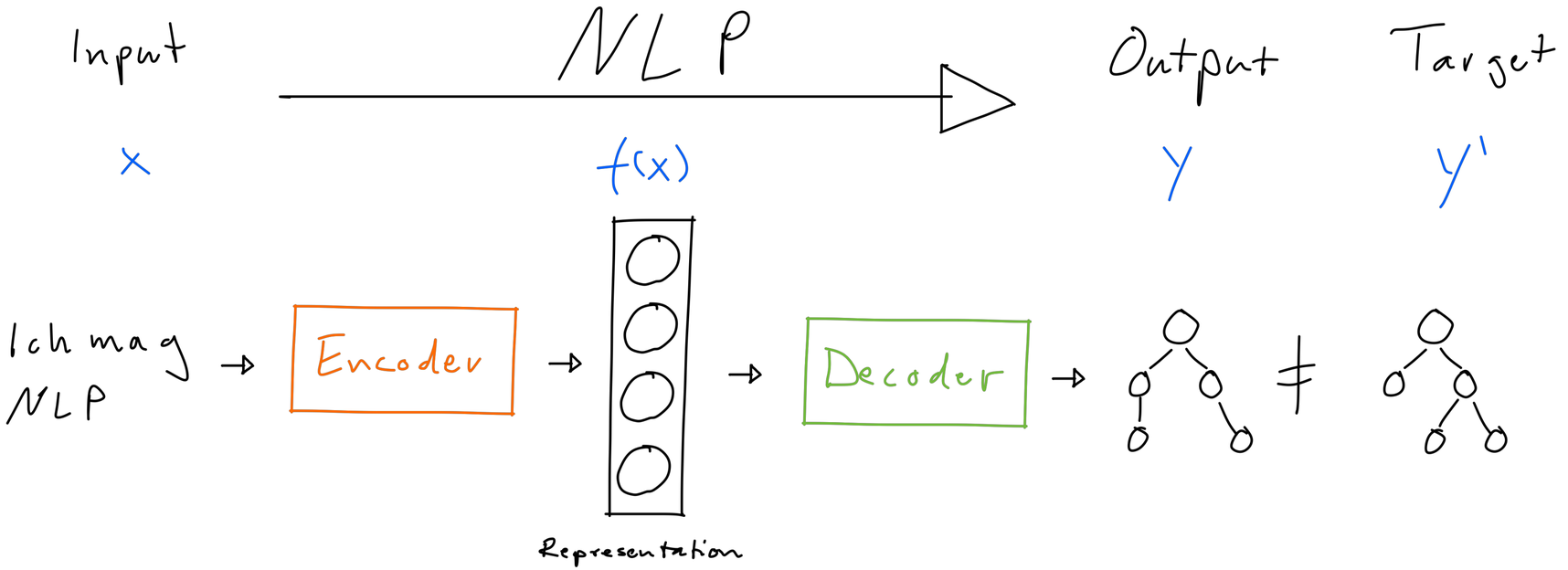
Constituency Parsing

Tim Rocktäschel & **Sebastian Riedel**
COMP0087 Natural Language Processing

NLP in a Nutshell



- Sentences
- Documents
- Languages
- Domains
- Databases

- Manual
- Learnt
- Pretrained
- RNN
- CNN

- symbolic
- sparse
- dense
- attention

- softmax
- sequence decoder
- tree decoder

- labels
- sequences
- trees
- graphs

- supervised
- self supervised
- weak
- unsup.
- semi-sup.

Motivation

- Say you want to automatically build a database of this form

Brand	Parent
Kit Kat	Nestle
Lipton	Unilever
⋮	⋮

Supervised Relation Extraction

Training Set

Dechra Pharmaceuticals has made its second acquisition after purchasing Genitrix

Trinity Mirror plc is the largest British newspaper after purchasing Local World

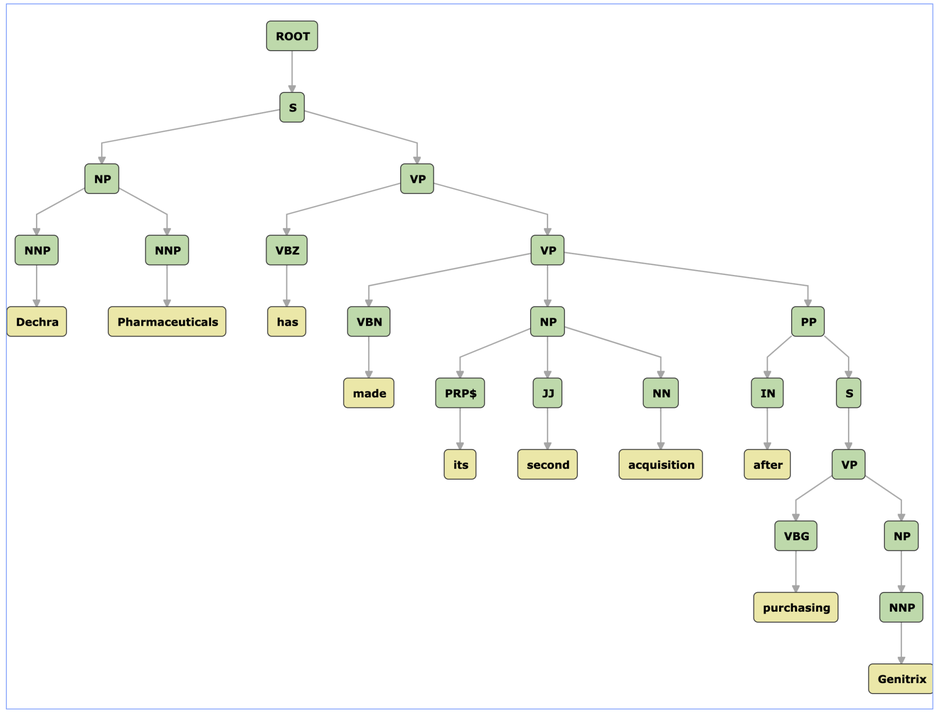
---

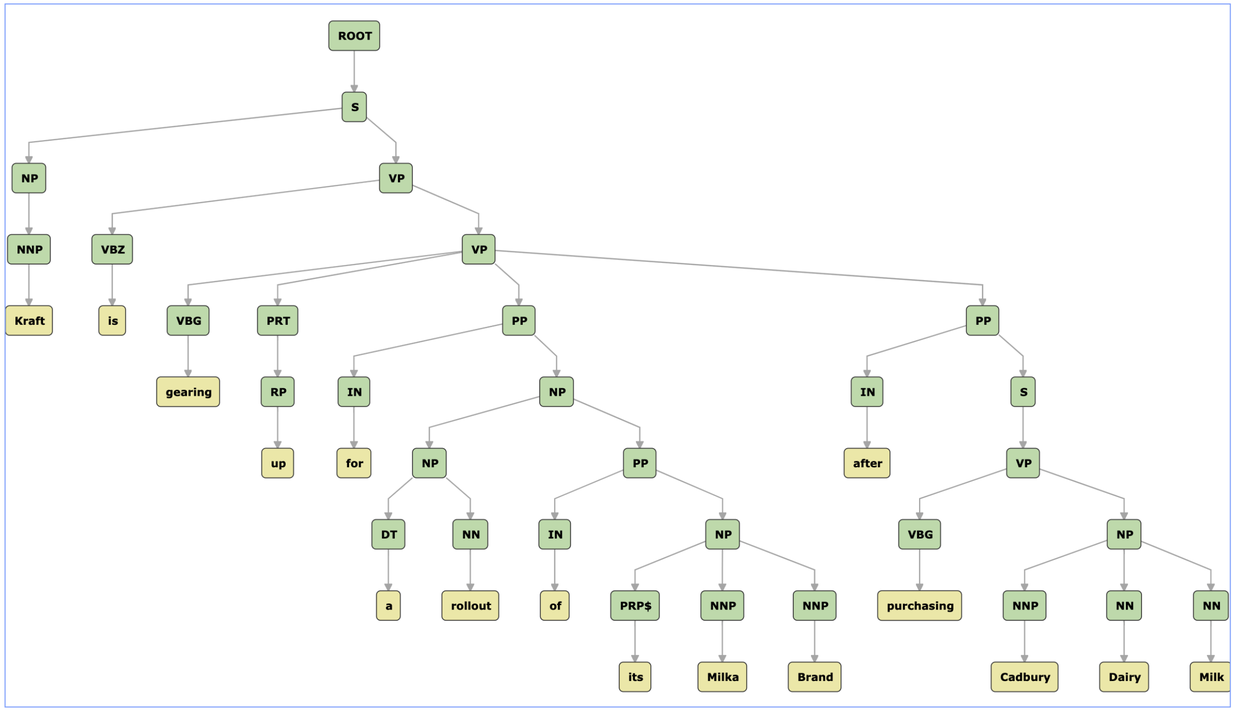
Test Set

Kraft is gearing up for a roll-out of its Milka brand after purchasing Cadbury Dairy Milk

Parse Trees

http://corenlp.run





Parsing

- Parsing is is the process of finding these trees:
  - important for downstream applications
  - the "celebrity" sub-field of NLP
  - partly because it marries linguistics and NLP
  - researched in academia and industry

Syntax

- from the Greek syntaxis (arrangement):
  - **Constituency**: groups of words act as single units.
  - **Grammatical Relations**: object, subject, direct object etc.
  - **Subcategorization**: restrictions on the type of phrases that go with certain words.

Constituency

- **Noun Phrase** (NP)
  - a roll-out of its Milka brand
  - Cadbury Dairy Milk
  - a roll-out
- **Verb Phrase** (VP)
  - is gearing up
  - purchasing Cadbury Dairy Milk
- **Prepositional Phrase** (PP)
  - of its Milka brand
  - after purchasing Cadbury Dairy Milk

Grammatical Relations

- *Subject* of purchasing: Kraft
- *Object* of purchasing: Cadbury Dairy Milk

Subcategorization

- There are more complex (sub) categories of verbs (and other types of words)
  - Intransitive Verbs: must not have objects
    - the student works
  - Transitive Verbs: must have exactly one object
    - Kraft purchased Cadbury Dairy Milk
  - Ditransitive Verbs: must have two objects
    - Give me a break!

Context Free Grammars

A Context Free Grammar (CFG) is a 4-tuple (N, Σ, R, S)

- N : set of non-terminal symbols
- Σ : set of terminal symbols
- R : rules $X \to Y_1 \dots Y_n$ where $X \in N$ and $Y_i \in N \cup \Sigma$
- S : $\in N$, start symbol

Example Grammar

$S \rightarrow NP_p \; VP_p$

$S \rightarrow NP_s \; VP_s$

$NP_p \rightarrow$ "Tim raps"

$VP_p \rightarrow$ "are" ADJ

$NP_s \rightarrow$ "Tim"

$VP_s \rightarrow$ "raps in NLP"

$ADJ \rightarrow$ "silly"

Left-Most Derivation

- The structure of a sentence with respect to a grammar can be described by its derivation (if it exists)

Sequence of sequences $S_1 \dots S_n$ such that

- $S_1 = S$: first sequence is the start symbol
- $S_n \in \Sigma^*$: last sequence consists of only terminals
- S_i for $i > 1$: replace left most non-terminal α in S_{i-1} with right-hand of $\alpha \rightarrow \beta_1 \dots \beta_n$

Example Derivation

$S \rightarrow NP_p \; VP_p$

$S \rightarrow NP_s \; VP_s$

$NP_p \rightarrow$ "Tim raps"

$VP_p \rightarrow$ "are" ADJ

$NP_s \rightarrow$ "Tim"

$VP_s \rightarrow$ "raps in NLP"

ADJ $\rightarrow$ "silly"

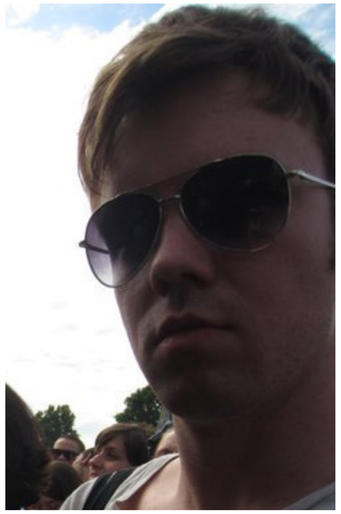
$S_1 = S$

$S_2 = NP_p \; VP_p$

$S_3 =$ "Tim raps" VP_p

$S_4 =$

Parse Trees



- Represent derivations as trees

$S \rightarrow NP_p \; VP_p$

$S \rightarrow NP_s \; VP_s$

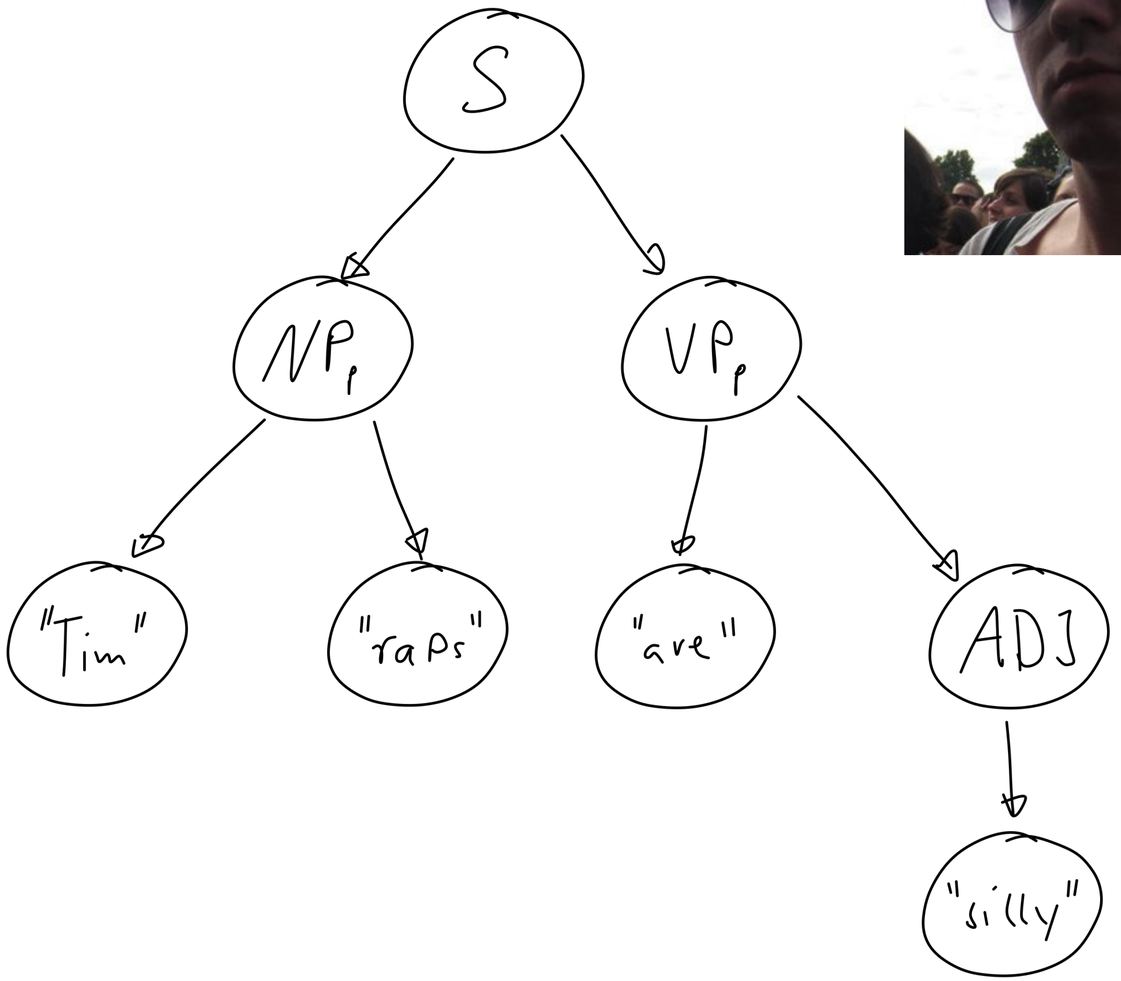
$NP_p \rightarrow$ "Tim raps"

$VP_p \rightarrow$ "are" ADJ

$NP_s \rightarrow$ "Tim"

$VP_s \rightarrow$ "raps in NLP"

ADJ $\rightarrow$ "silly"



Parse Tree Example 2

- Represent derivations as trees

$S \rightarrow NP_p \; VP_p$

$S \rightarrow NP_s \; VP_s$

$NP_p \rightarrow$ "Tim raps"

$VP_p \rightarrow$ "are" ADJ

$NP_s \rightarrow$ "Tim"

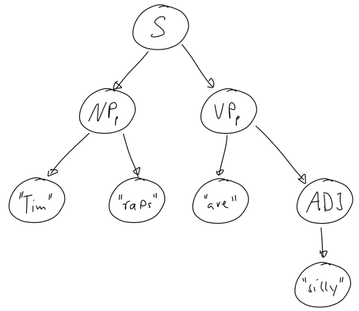
$VP_s \rightarrow$ "raps in NLP"

ADJ $\rightarrow$ "silly"

How to find Parse Tree

Syntactic Parsing: given grammar & sentence, get parse tree

"Tim raps are silly" →



- Top Down
  - Start at S, generate trees and keep only those that match
  - wastes no effort in generating non-S trees
  - but generates a lot of trees inconsistent with input
- Bottom Up
  - Start at sentence, and compose trees until an S is reached
  - wastes no effort in generating inconsistent trees
  - but generates a lot trees that don’t lead to S

Bottom up Parsing

$S \rightarrow NP_p \; VP_p$

$S \rightarrow NP_s \; VP_s$

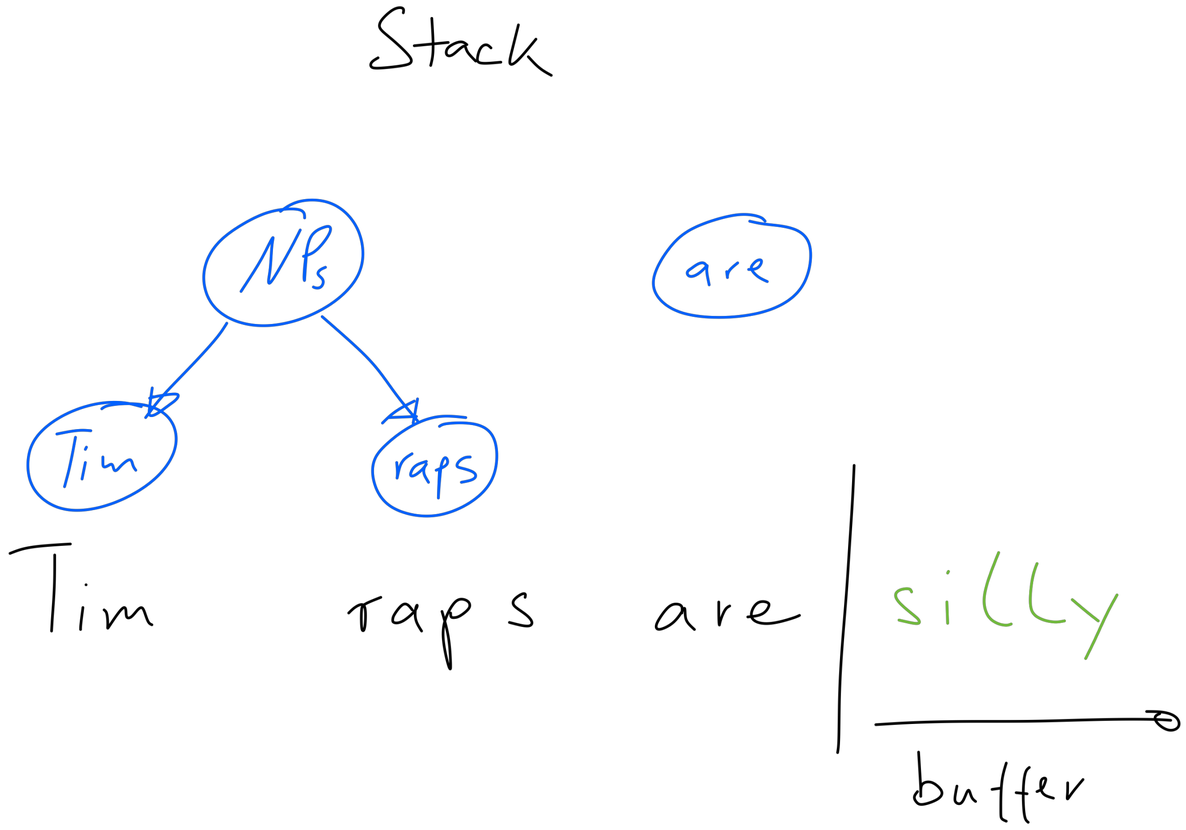
$NP_p \rightarrow$ "Tim raps"

$VP_p \rightarrow$ "are" ADJ

$NP_s \rightarrow$ "Tim"

$VP_s \rightarrow$ "raps in NLP"

ADJ $\rightarrow$ "silly"



Shift

$S \rightarrow NP_p \; VP_p$

$S \rightarrow NP_s \; VP_s$

$NP_p \rightarrow$ "Tim raps"

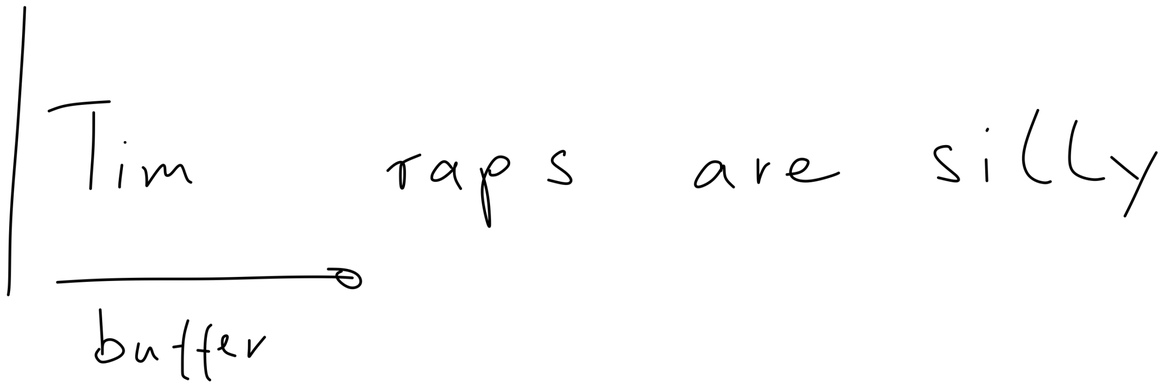
$VP_p \rightarrow$ "are" ADJ

$NP_s \rightarrow$ "Tim"

$VP_s \rightarrow$ "raps in NLP"

ADJ $\rightarrow$ "silly"

Stack



Shift

$S \rightarrow NP_p \; VP_p$

$S \rightarrow NP_s \; VP_s$

$NP_p \rightarrow$ "Tim raps"

$VP_p \rightarrow$ "are" ADJ

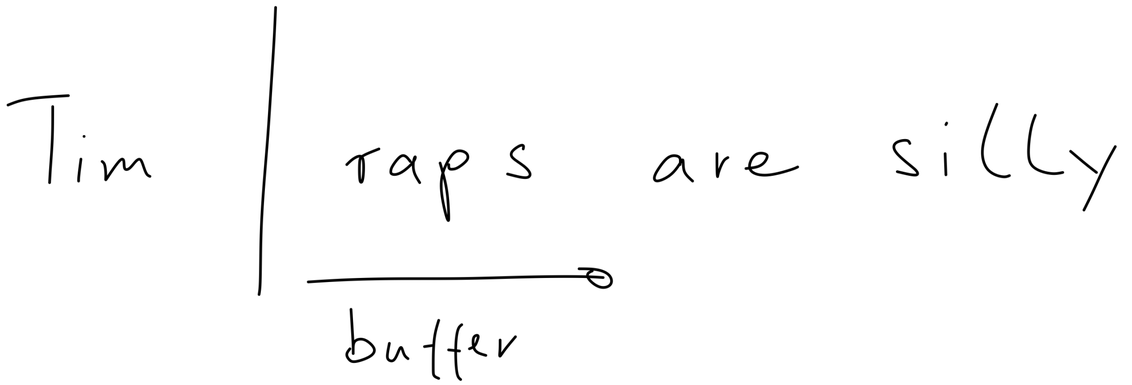
$NP_s \rightarrow$ "Tim"

$VP_s \rightarrow$ "raps in NLP"

ADJ $\rightarrow$ "silly"

Stack

Tim



Reduce

$S \rightarrow NP_p \; VP_p$

$S \rightarrow NP_s \; VP_s$

$NP_p \rightarrow$ "Tim raps"

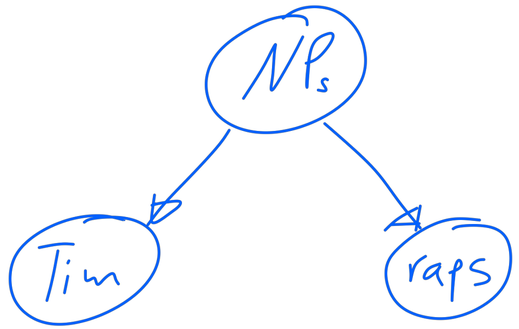
$VP_p \rightarrow$ "are" ADJ

$NP_s \rightarrow$ "Tim"

$VP_s \rightarrow$ "raps in NLP"

ADJ $\rightarrow$ "silly"

Stack



Tim raps | are silly

buffer

Backtracking

$S \rightarrow NP_p \; VP_p$

$S \rightarrow NP_s \; VP_s$

$NP_p \rightarrow$ "Tim raps"

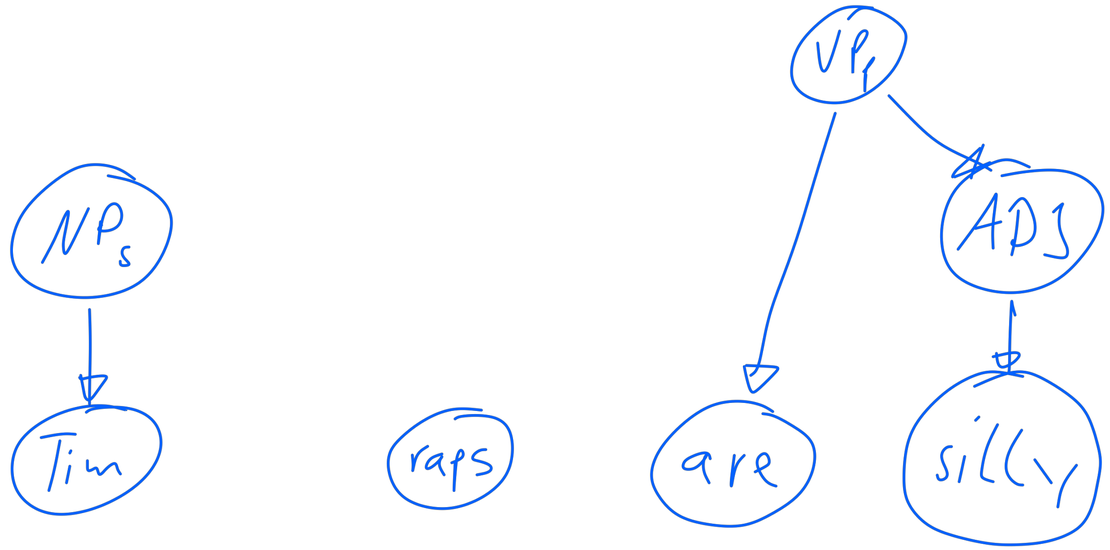
$VP_p \rightarrow$ "are" ADJ

$NP_s \rightarrow$ "Tim"

$VP_s \rightarrow$ "raps in NLP"

ADJ $\rightarrow$ "silly"

Stack



Tim raps are silly |

Backtracking

Stack

$S \rightarrow NP_p \; VP_p$

$S \rightarrow NP_s \; VP_s$

$NP_p \rightarrow$ "Tim raps"

$VP_p \rightarrow$ "are" ADJ

$NP_s \rightarrow$ "Tim"

$VP_s \rightarrow$ "raps in NLP"

ADJ $\rightarrow$ "silly"

Tim

Tim | raps are silly

Backtracking

Stack

$S \rightarrow NP_p \; VP_p$

$S \rightarrow NP_s \; VP_s$

$NP_p \rightarrow$ "Tim raps"

$VP_p \rightarrow$ "are" ADJ

$NP_s \rightarrow$ "Tim"

$VP_s \rightarrow$ "raps in NLP"

ADJ $\rightarrow$ "silly"

Tim raps | are silly

Full Parse

Stack

$S \rightarrow NP_p \ VP_p$

$S \rightarrow NP_s \ VP_s$

$NP_p \rightarrow$ "Tim raps"

$VP_p \rightarrow$ "are" ADJ

$NP_s \rightarrow$ "Tim"

$VP_s \rightarrow$ "raps in NLP"

ADJ $\rightarrow$ "silly"

Tim raps are silly

Dynamic Programming

- Cache parse trees for parts (spans!) of the sentence
- Build them up incrementally by span size
- First convert grammar into Chomsky Normal Form

Chomsky Normal Form

Rules have the form:

- $\alpha \rightarrow \beta\gamma \in N \setminus S \times N \setminus S$
  - ← exactly 2 nonterminals
- $\alpha \rightarrow t \in \Sigma$
  - ← exactly 1 terminal

Conversion to CNF

- $\alpha \to \beta\gamma\delta \Rightarrow \alpha \to \beta\alpha', \alpha' \to \gamma\delta$
- $\alpha \to \beta t \Rightarrow \alpha \to \beta\alpha', \alpha' \to t \in \Sigma$
- $\alpha \to \beta \quad \beta \to \gamma\delta \Rightarrow \alpha \to \gamma\delta, \beta \to \gamma\delta$

Example

S → NP VP PP

VP → are ADJ

Full Conversion

$S \rightarrow NP_p \; VP_p$

$S \rightarrow NP_s \; VP_s$

$NP_p \rightarrow$ "Tim raps"

$VP_p \rightarrow$ "are" ADJ

$NP_s \rightarrow$ "Tim"

$VP_s \rightarrow$ "raps in NLP"

$ADJ \rightarrow$ "silly"

$S \rightarrow NP_p \; VP_p$

$S \rightarrow NP_s \; VP_s$

$NP_p^0 \rightarrow$ "Tim"

$NP_p^1 \rightarrow$ "raps"

$NP_p \rightarrow NP_p^0 \; NP_p^1$

$VP_p^2 \rightarrow$ "are"

$VP_p \rightarrow VP_p^2 \; ADJ$

$NP_s \rightarrow$ "Tim"

$VP_s^4 \rightarrow$ "raps"

$VP_s \rightarrow VP_s^4 \; VP_s^3$

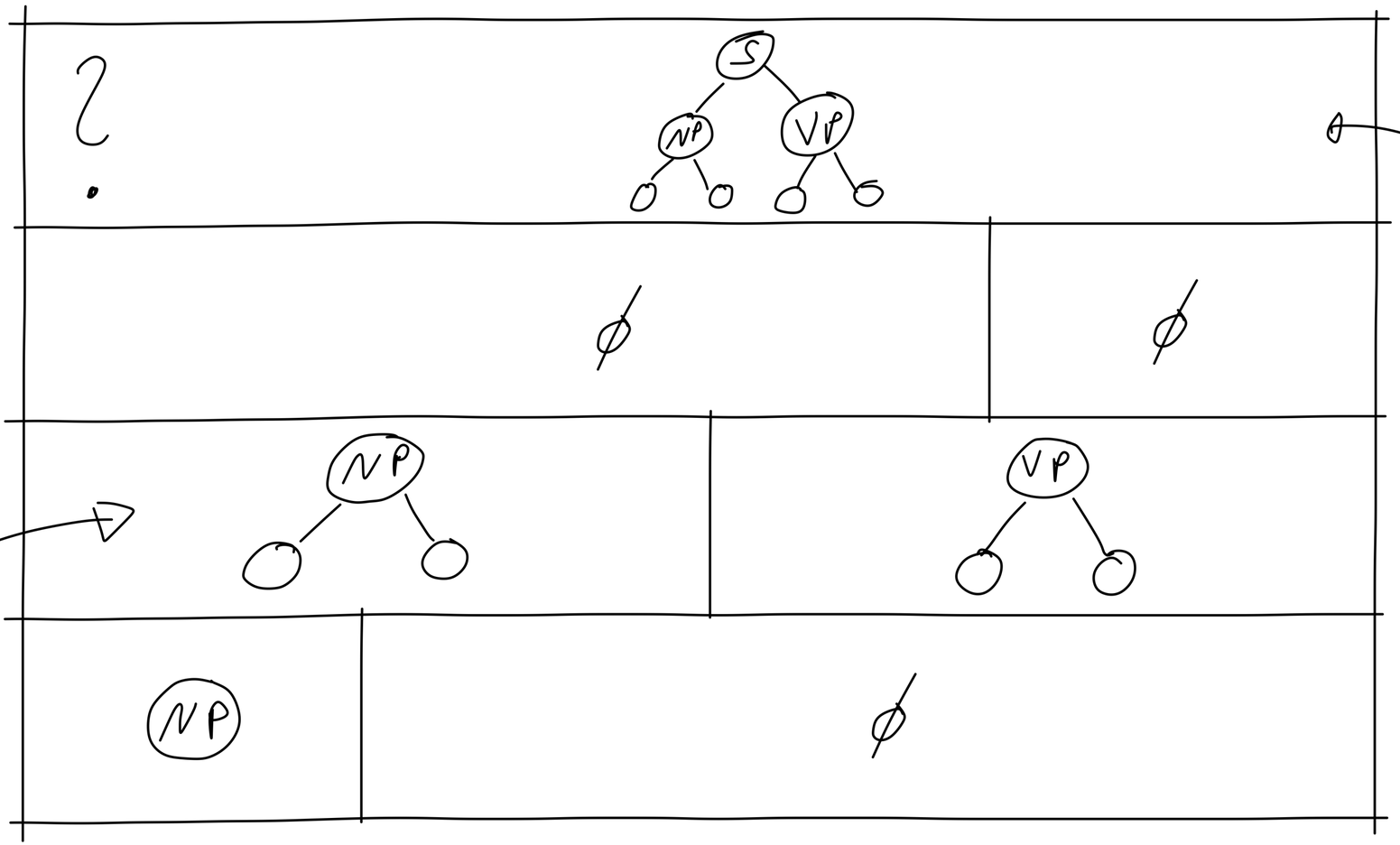
$VP_s^{3,5} \rightarrow$ "in"

$VP_s^{3,6} \rightarrow$ "NLP"

$VP_s^3 \rightarrow VP_s^{3,5} \; VP_s^{3,6}$

$ADJ \rightarrow$ "silly"

Intuition

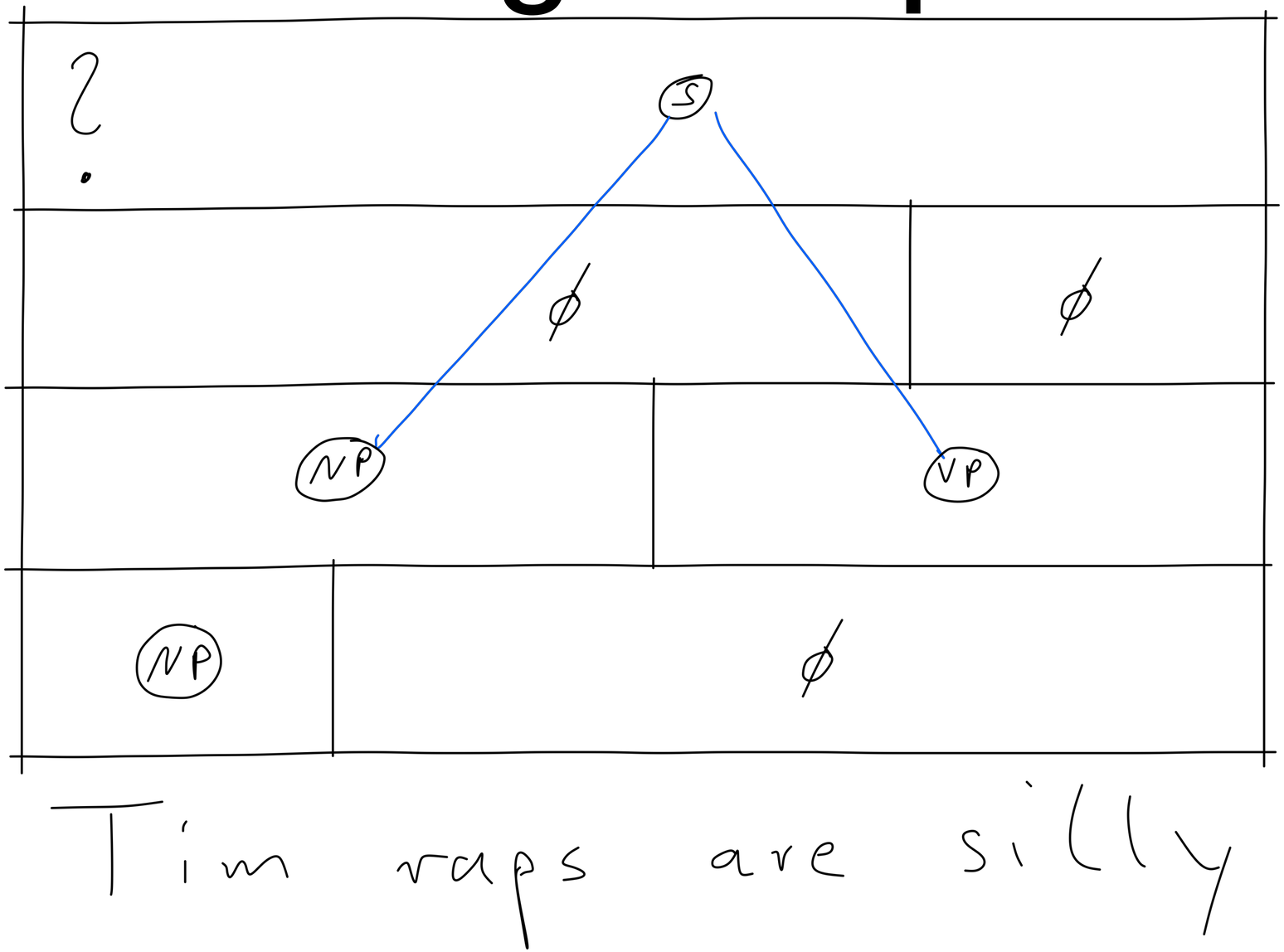


all trees covering "Tim raps"

all trees covering "Tim raps are silly"

Tim raps are silly

Maintaining Backpointers

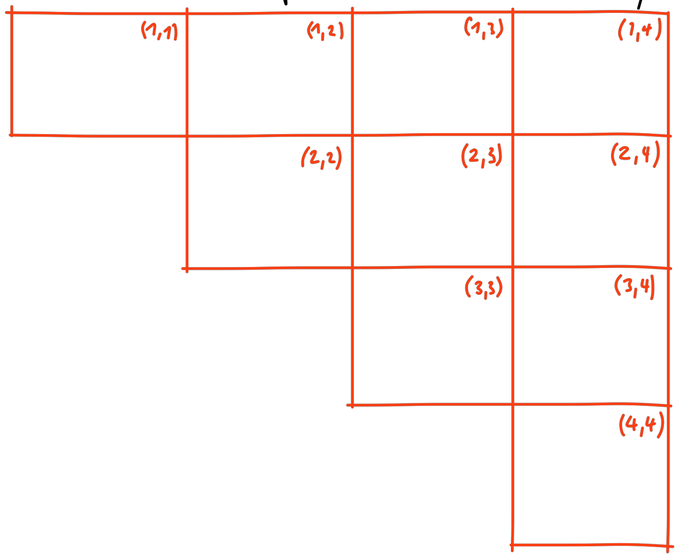


Cocke-Younger-Kasami

for $i \in 1..n$:
  $S(i,i) = \{\alpha \mid \alpha \to w_i \in R\}$

for $\ell = 1..n-1$:
  for $i = 1..n-\ell$:
    $S(i, i+\ell) = \phi$
    for $m = i..i+\ell-1$:
      $S(i, i+\ell) = S(i, i+\ell) \cup \{\alpha \mid \alpha \to \beta\gamma \in R, \beta \in S(i,m), \gamma \in S(m, i+\ell)\}$

Tim
raps
ave
silly



"Chart"

$A \to$ Tim
$B \to$ raps
$A \to$ ave
$B \to$ silly
$A \to AB$
$B \to BA$

Example

$S \rightarrow NP_p\ VP_p$

$S \rightarrow NP_s\ VP_s$

$NP_p^0 \rightarrow$ "Tim"

$NP_p^1 \rightarrow$ "raps"

$NP_p \rightarrow NP_p^0\ NP_p^1$

$VP_p^2 \rightarrow$ "are"

$VP_p \rightarrow VP_p^2\ ADJ$

$NP_s \rightarrow$ "Tim"

$VP_s^4 \rightarrow$ "raps"

$VP_s \rightarrow VP_s^4\ VP_s^3$

$VP_s^{3,5} \rightarrow$ "in"

$VP_s^{3,6} \rightarrow$ "NLP"

$VP_s^3 \rightarrow VP_s^{3,5}\ VP_s^{3,6}$

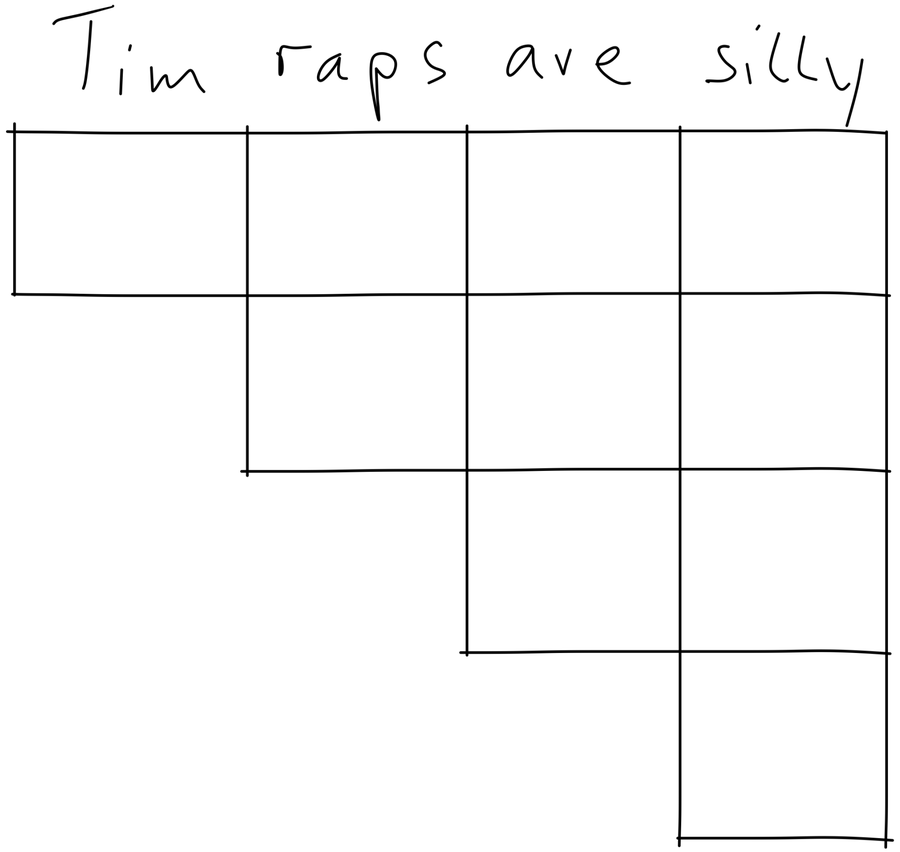
$ADJ \rightarrow$ "silly"

Tim

raps

are

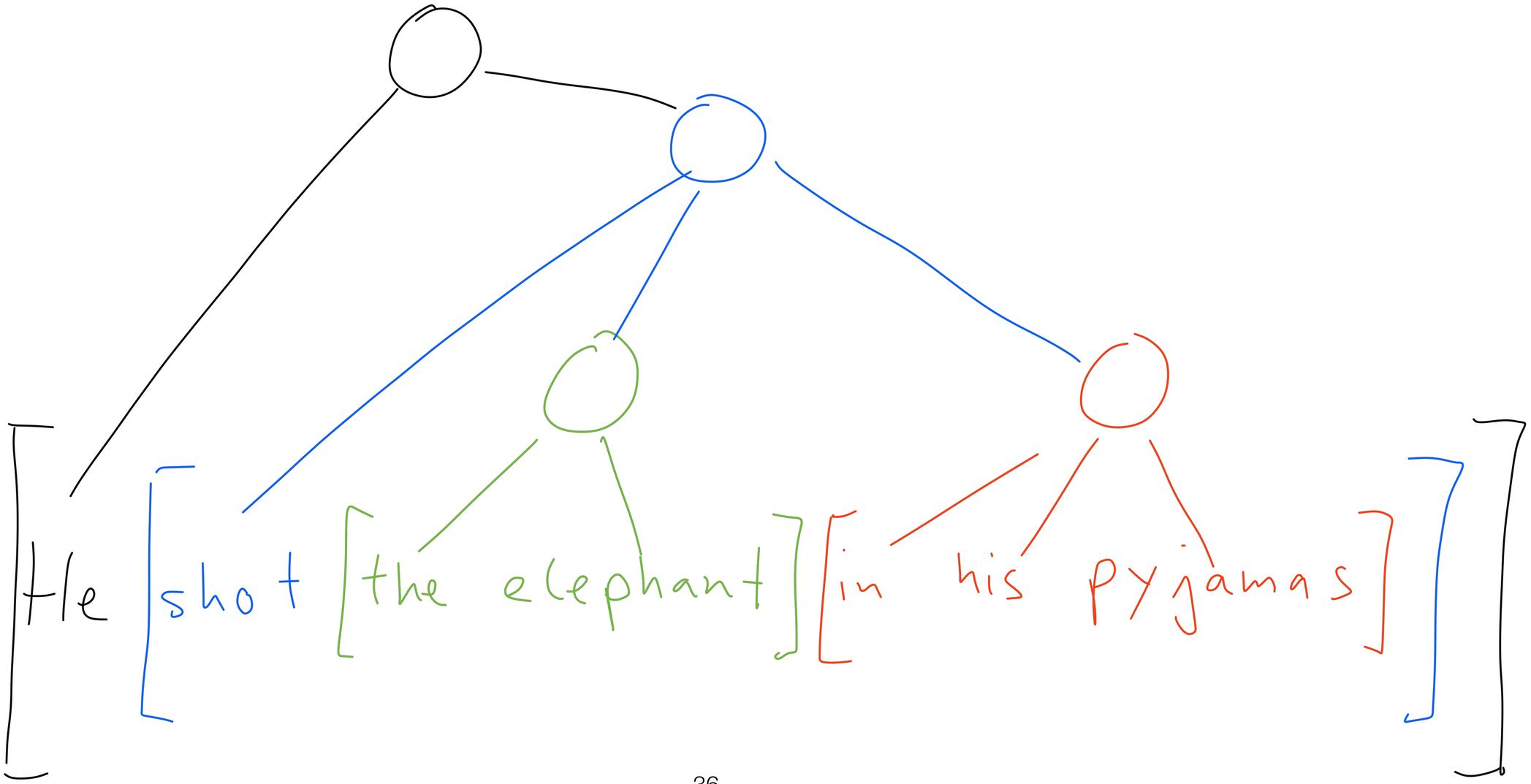
silly



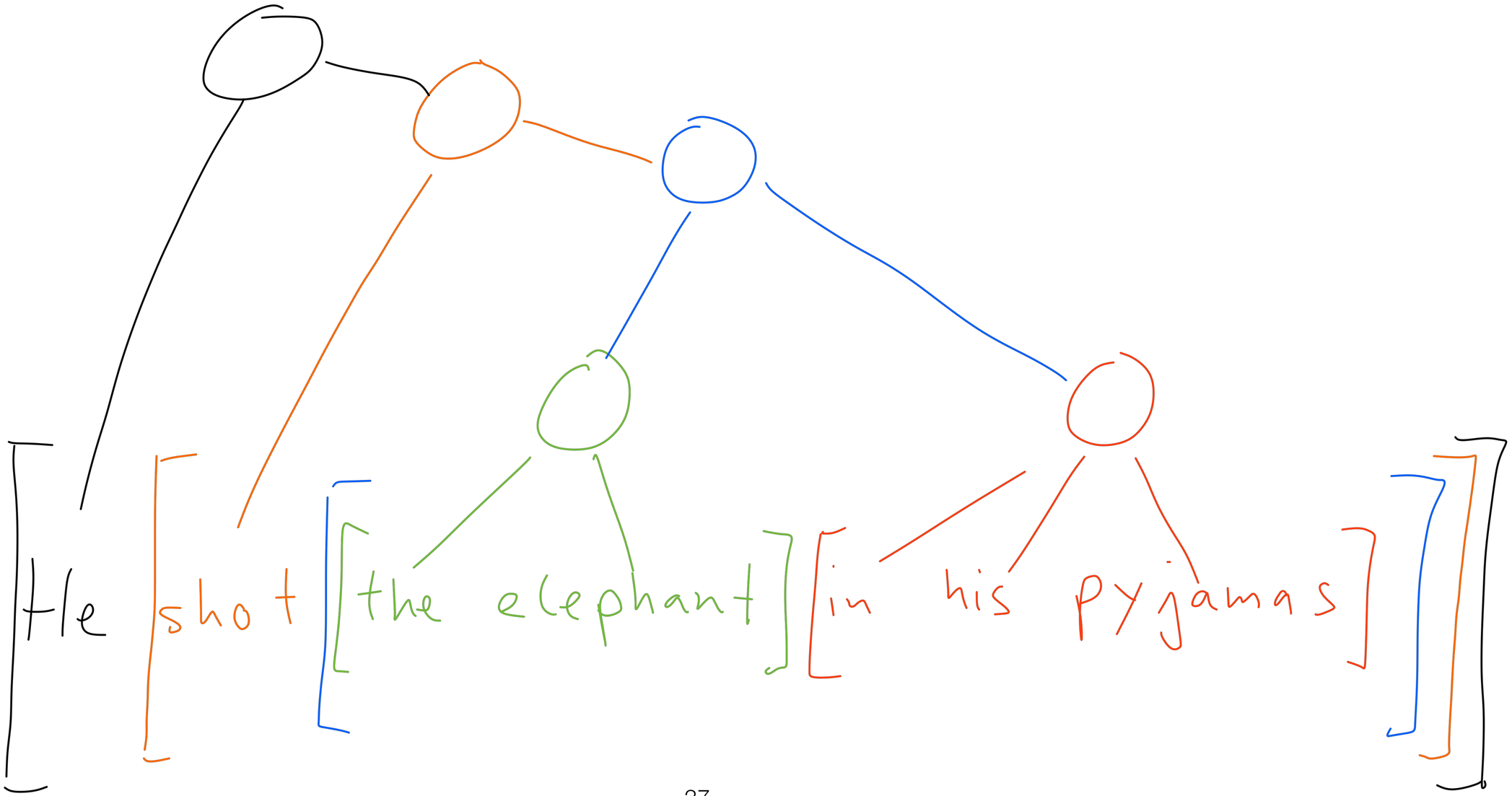
Ambiguity

- Usually there are more than one legal parse trees per sentence
- How to score these to indicate that one is more likely than another?

Legal Parse 1



Legal Parse 2

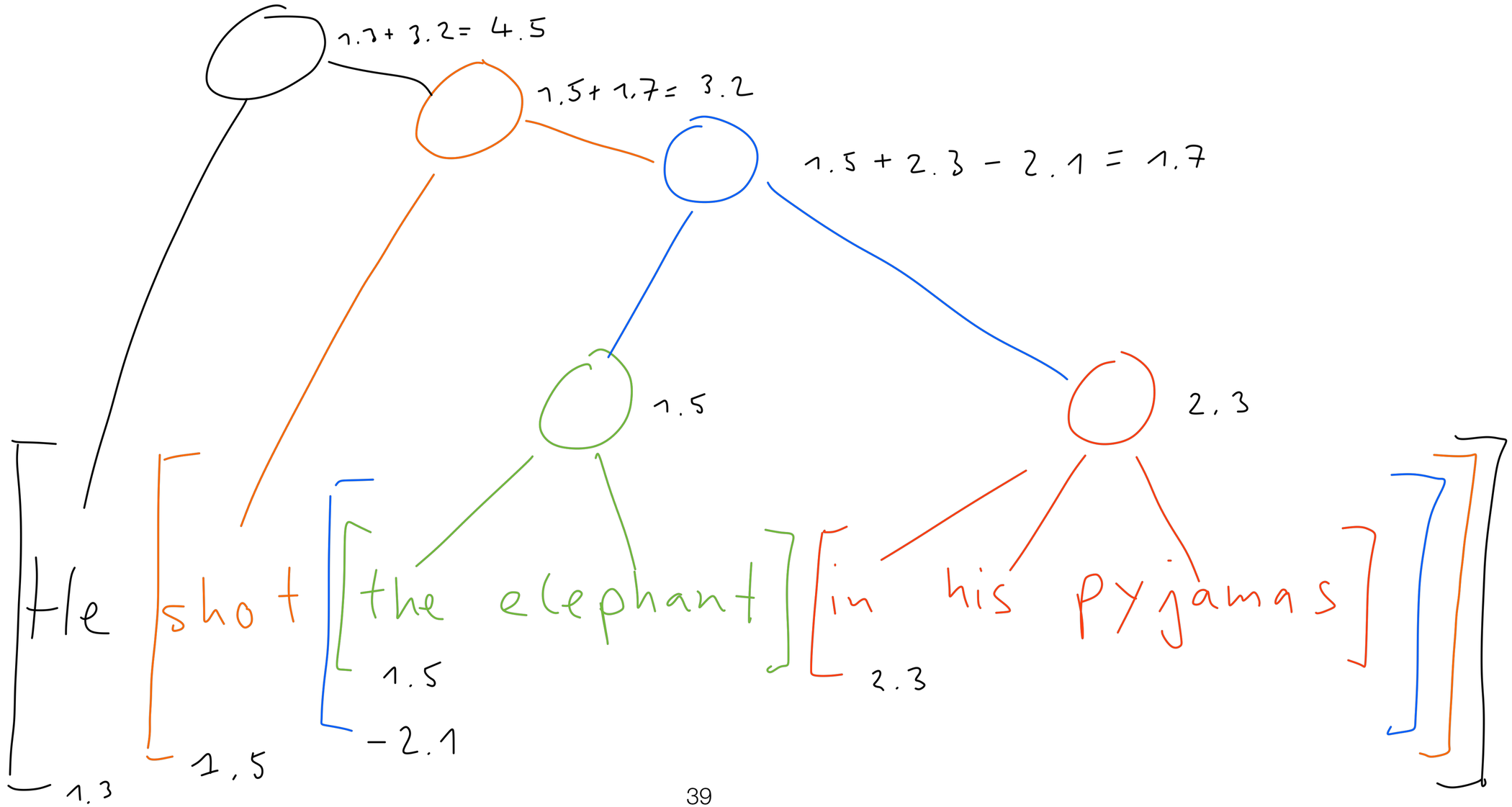


Scoring Trees

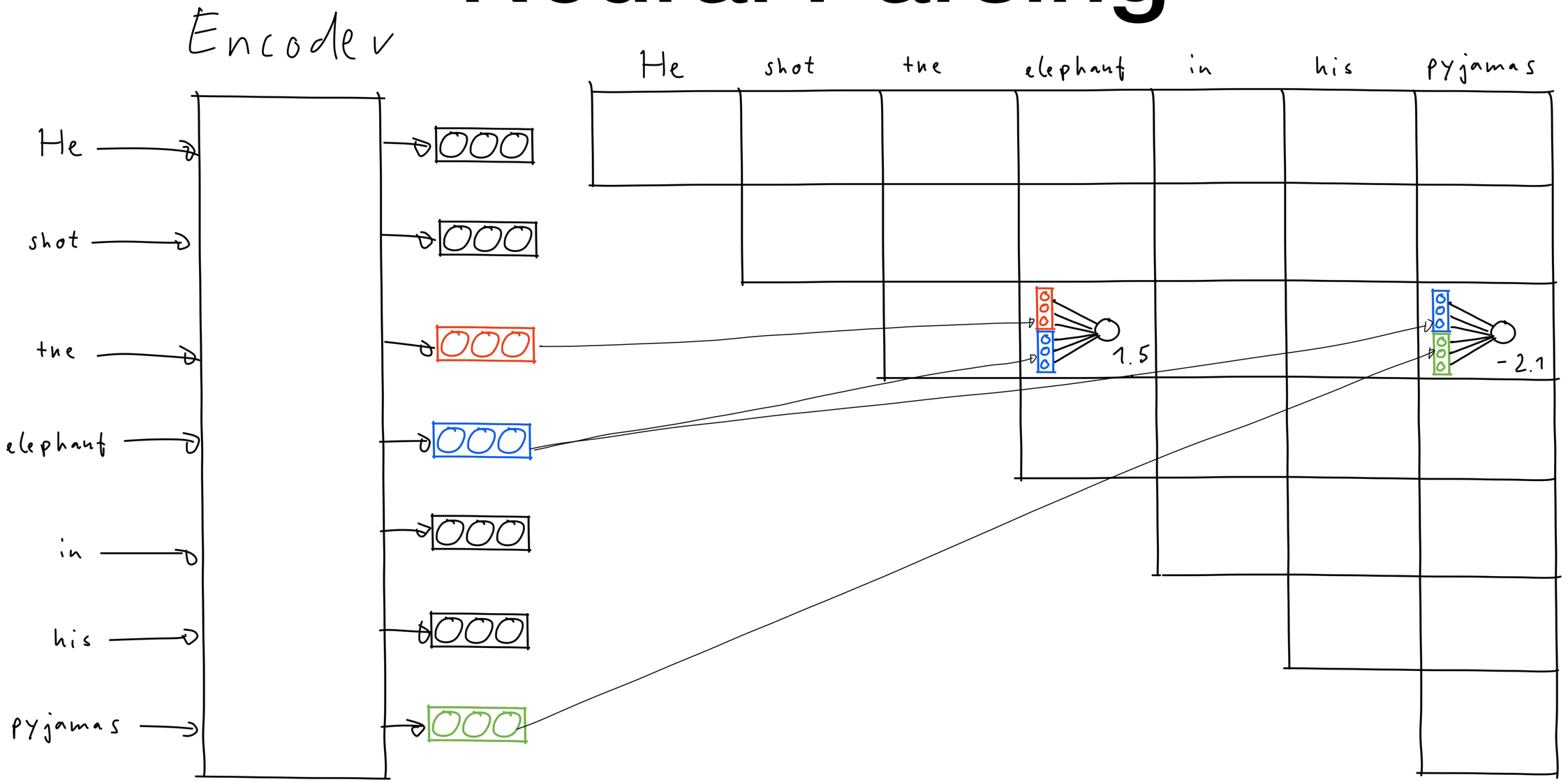
- Define a tree score based on a neural network that scores its brackets
- Train such that correct trees have highest score using labelled trees

$$s(\text{tree}) = \sum_{\text{bracket} \in \text{tree}} nn(\text{bracket})$$

Legal Parse 2



Neural Parsing



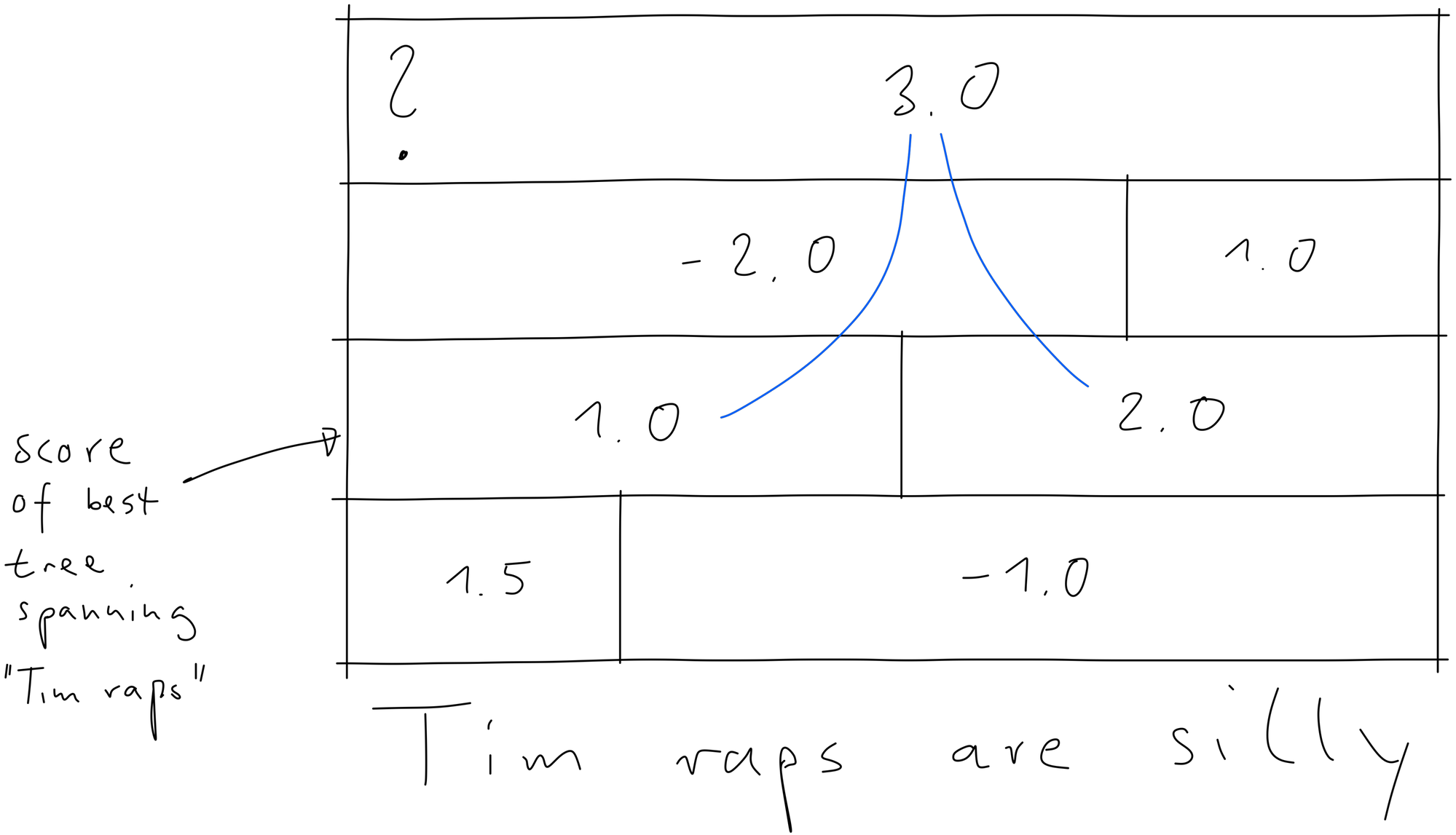
Score-based Parsing

- Assuming trained neural network, find highest scoring tree

$$\operatorname{argmax}_{tree \in \text{Legal Trees}} s(tree)$$

- no overlaps
- no missing words

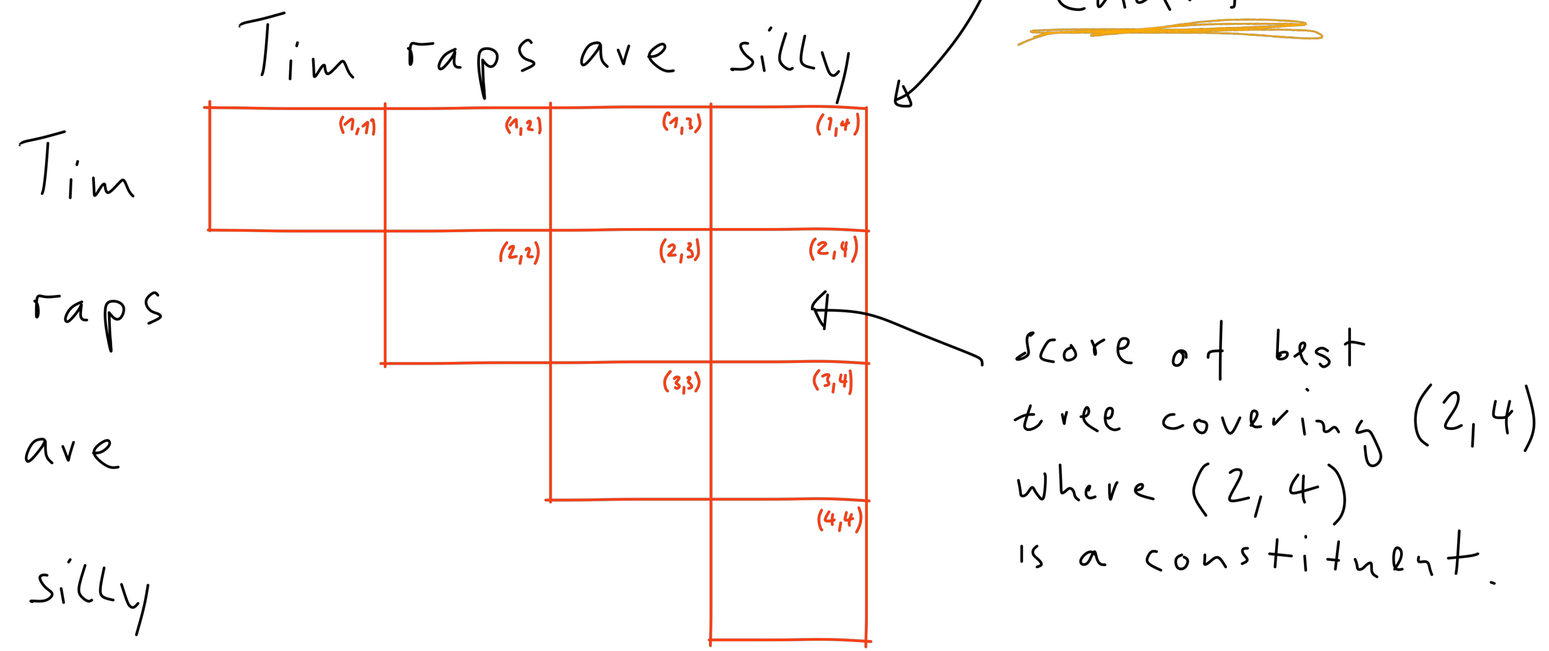
Intuition for Score-based CYK



Score-based CYK

for $i \in 1 \ldots n$:
  $s(i,i) = nn(i,i)$

for $\ell = 1 \ldots n-1$:
  for $i = 1 \ldots n-\ell$:
    $s(i, i+\ell) = -\infty$
    for $m = i \ldots i+\ell-1$:
      $s(i, i+\ell) = \max\left(s(i, i+\ell),\ nn(i, i+\ell) + s(i, m) + s(m, i+\ell) \right)$



References

- What's Going On in Neural Constituency Parsers? An Analysis, David Gaddy et al., NAACL 2018
- Mike Collins PCFG Lecture
- Stat NLP Book: Constituency Parsing
- J&M: Chapter 11