

CS378: Natural Language Processing

Lecture 20: Trees II



Eunsol Choi

Some slides adapted from Greg Durrett, Yoav Artzi, Yejin Choi, Michael Collins, Princeton NLP, Graham Neubig



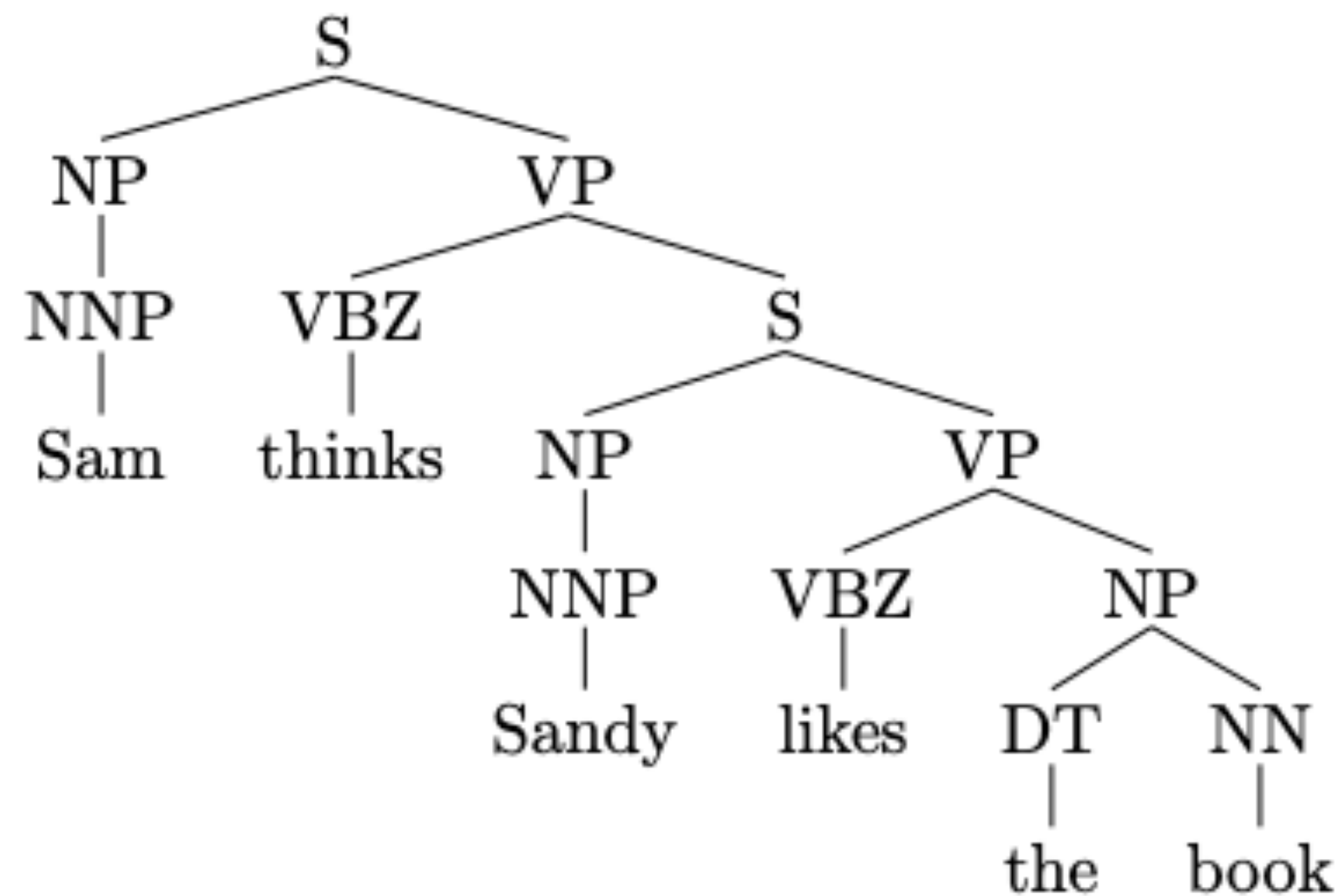
Trees

- ▶ Overview
- ▶ Dependency Parsing [Tuesday]
- ▶ Constituency Parsing [Today]

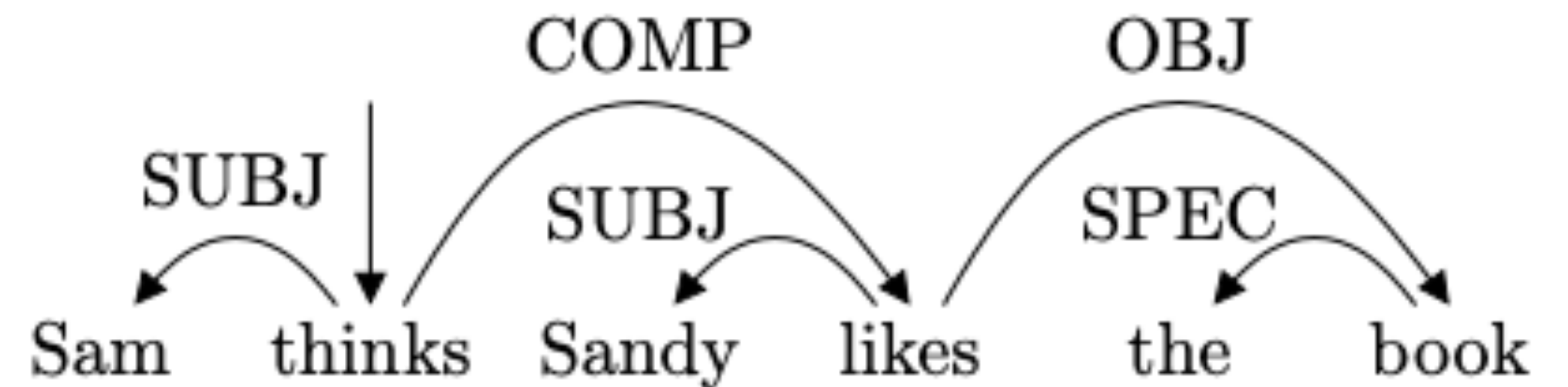


Recap: Syntax

- ▶ Study of word order and how words form sentences
 - ▶ Constituency Parsing
 - ▶ Dependency Parsing



Words organized as nested structure of constituents



which words depend on (modify or are arguments of) which other words.



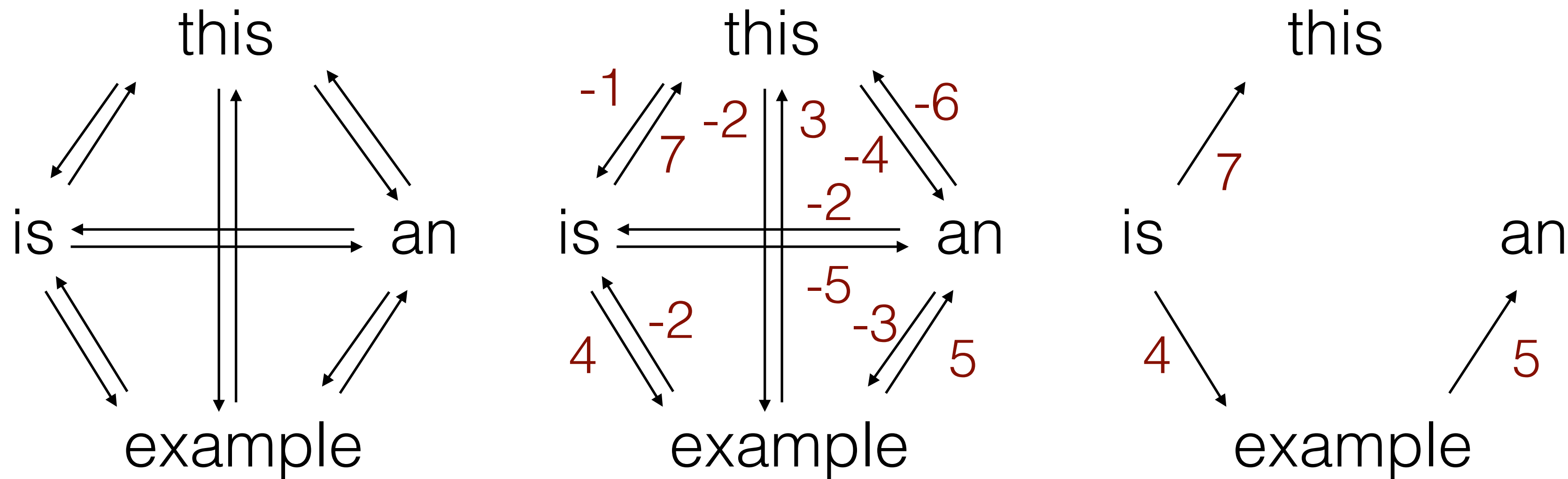
Recap: Why do we care about syntax?

- ▶ Clarifies the ambiguities in language
 - ▶ Recognize verb-argument structures (who is doing what to whom?)
 - ▶ Recognize modifier scopes (e.g., plastic cup holder)
 - ▶ Coordination scope (e.g., small rats and mice)
- ▶ Provide higher level of abstraction beyond words: some languages are SVO, some are VSO, some are SOV
- ▶ Sometimes can be used to help downstream tasks

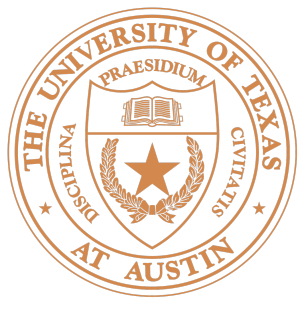


Recap: Graph-based dependency parsing

- ▶ Train a classifier to score each edge
- ▶ Find the maximum spanning tree

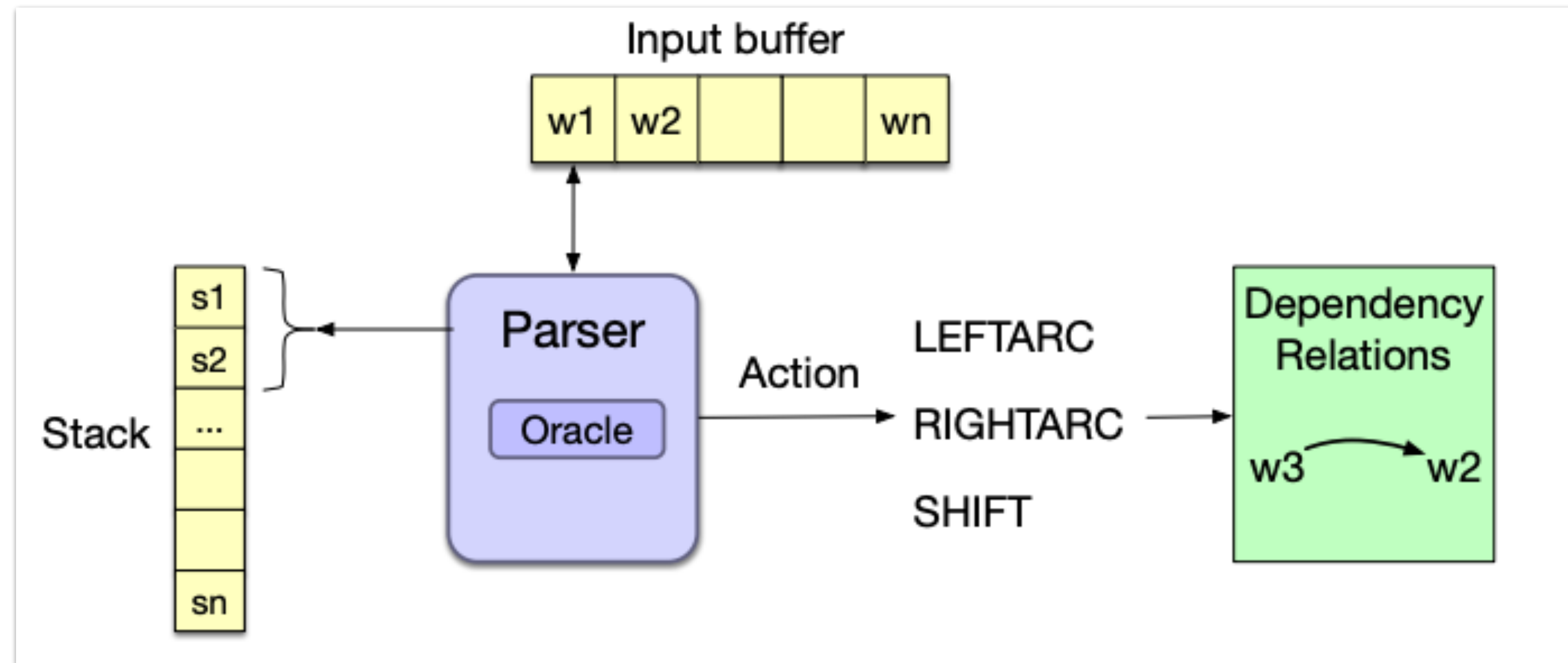


- ▶ Pros: can handle non-projective trees, guaranteed best-scoring parse
- ▶ Cons: relatively slow $O(n^3)$, n = length of the sentence, individual arc scoring



Today: Transition-based dependency parsing

- ▶ Process a sentence sequentially word by word



- ▶ As you process, you can either:
 - ▶ Shift: Move a word from the buffer to a stack
 - ▶ Left Arc: The top of the stack is the head of the second word on stack
 - ▶ Right Arc: The second word on stack is head of top word



Transition Based Parsing

- ▶ Initially, the stack has root, the buffer has sentence's words, and no edges
 - ▶ Eg) **stack contains [ROOT]**, **buffer contains [I ate some spaghetti bolognese]**
- ▶ At the end, **stack contains [ROOT]**, **buffer is empty []**

```
function DEPENDENCYPARSE(words) returns dependency tree  
  
state ← { [root], [words], [] } ; initial configuration  
while state not final  
    t ← ORACLE(state) ; choose a transition operator to apply  
    state ← APPLY(t, state) ; apply it, creating a new state  
return state
```

Figure 14.6 A generic transition-based dependency parser



Arc-Standard Parsing

ROOT



- ▶ Start: **stack contains [ROOT]**, **buffer contains [I ate some spaghetti bolognese]**
- ▶ Three operations
 - ▶ Let σ denote the stack
 - ▶ Shift: top of buffer \rightarrow top of stack
 - ▶ Left-arc: “Pop two elements, add a left arc, put them back on the stack”
 $\sigma | w_{-2}, w_{-1} \rightarrow \sigma | w_{-1}$ w_{-2} is now a child of w_{-1}
 - ▶ Right-arc: “Pop two elements, add a right arc, put them back on the stack”
 $\sigma | w_{-2}, w_{-1} \rightarrow \sigma | w_{-2}$, w_{-1} is now a child of
- ▶ End: **stack contains [ROOT]**, **buffer is empty []**



Transition Based Parsing

ROOT

I ate some spaghetti bolognese

Shift top of **buffer** -> top of **stack**
LA **pop two**, left arc between them
RA **pop two**, right arc between them

Shift
Stack: [ROOT] Buffer: [I ate some spaghetti bolognese]

Stack: [ROOT I] Buffer: [ate some spaghetti bolognese]

Shift
Stack: [ROOT I ate] Buffer: [some spaghetti bolognese]

LA
Stack: [ROOT ate] Buffer: [some spaghetti bolognese]

Shift
Stack: [ROOT ate some] Buffer: [spaghetti bolognese]



Transition Based Parsing

ROOT

I ate some spaghetti bolognese

- Shift** top of **buffer** -> top of **stack**
- LA** **pop two**, left arc between them
- RA** **pop two**, right arc between them

► Stack: [ROOT ate some] Buffer: [spaghetti bolognese]

Shift

► Stack: [ROOT ate some spaghetti] Buffer: [bolognese]

LA

► Stack: [ROOT ate spaghetti] Buffer: [bolognese]

Shift

► Stack: [ROOT ate spaghetti bolognese] Buffer: []

↓
|

↓
|

↓
|

↓
|

some

some



Transition Based Parsing

ROOT

I ate some spaghetti bolognese

Shift

top of **buffer** -> top of **stack**

LA

pop two, left arc between them

RA

pop two, right arc between them

- Stack: [ROOT ate spaghetti bolognese] Buffer: []

RA

I some

- Stack: [ROOT ate spaghetti] Buffer: []

RA

I some bolognese

- Stack: [ROOT ate] Buffer: []

RA

I spaghetti
 some bolognese

Final state:

[ROOT]

[]

ate

I

spaghetti

some

bolognese



Complexity

- ▶ A word can only enter the stack once.
- ▶ So complexity is $O(2n)$, where n is the length of the sentence.



Train a classifier to predict actions

- ▶ Following rule will give a valid sequence for the annotated parse tree.

If the second element in the stack is the child of the top of the stack, then make a left edge.
If the top of the stack is the child of the second element in the stack, and the top of the stack has no children that have yet to be added to the tree, then make a right edge.
Otherwise, shift

- ▶ For each x_i with n words, we can construct a transition sequence of length $2n$ which generates y_i , so we can generate $2n$ training examples: $\{(c_k, a_k)\}$

c_k : configuration, a_k : action

- ▶ Space of actions:

- ▶ 3 (if untyped edges)

- ▶ $|R| * 2 + 1$ (if typed edges, R the number of types)

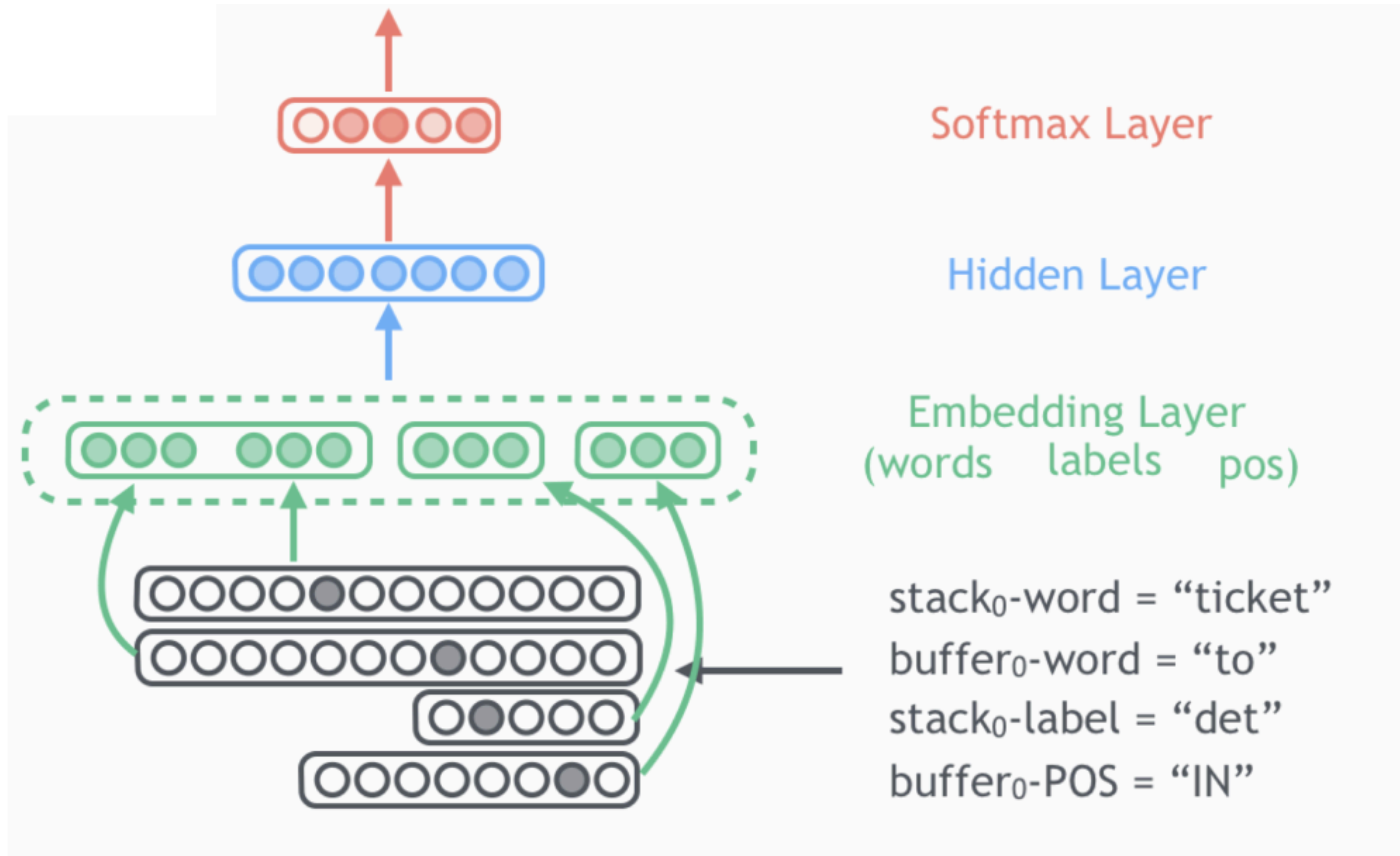


Features in Transition Based Parsing

- ▶ The top 3 words on the stack and buffer (6 features)
 $s_1, s_2, s_3, b_1, b_2, b_3$
- ▶ The two leftmost/rightmost children of the top two words on the stack (8 features)
 $lc_1(s_i), lc_2(s_i), rc_1(s_i), rc_2(s_i) \ i=1,2$
- ▶ leftmost and rightmost grandchildren (4 features)
 $lc_1(lc_1(s_i)), rc_1(rc_1(s_i)) \ i=1,2$
- ▶ POS tags of all of the above (18 features)
- ▶ Arc labels of all children/grandchildren (12 features)



Neural Features





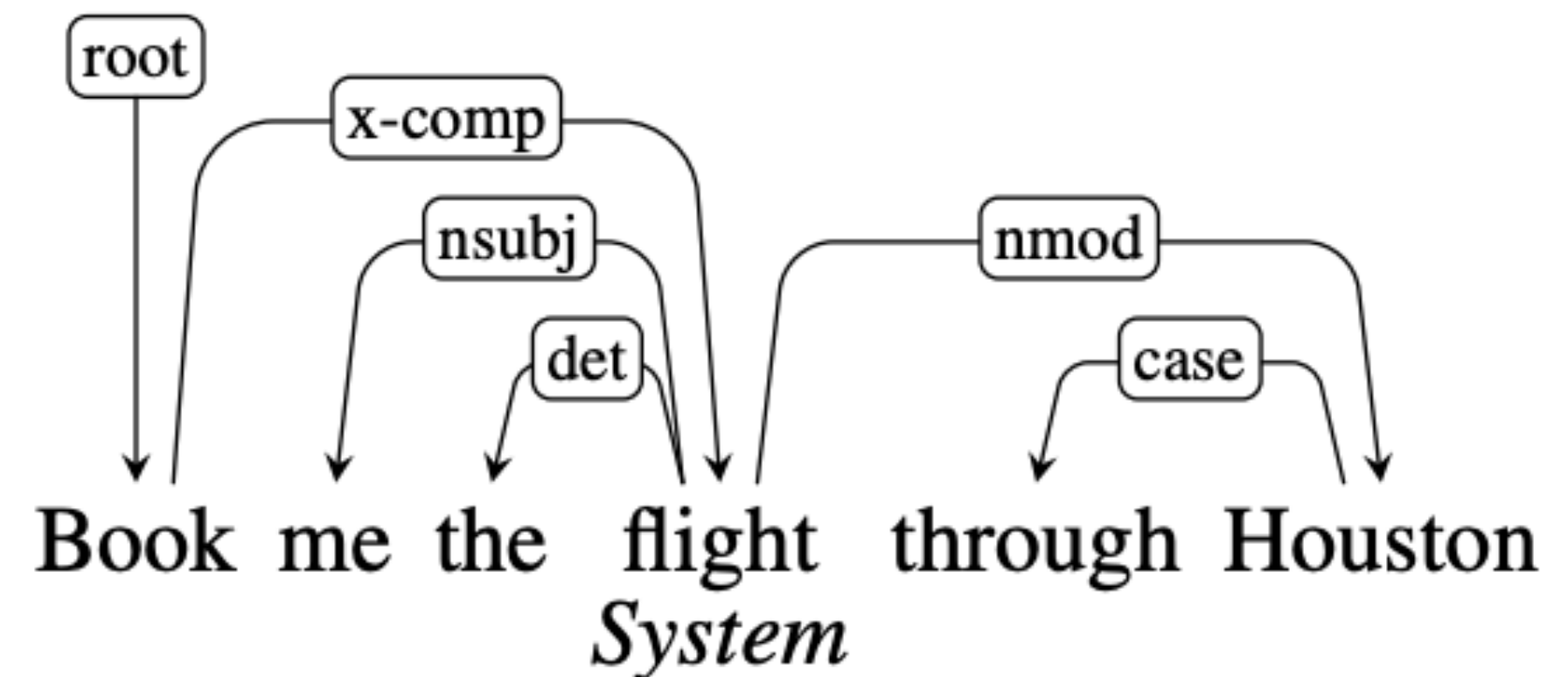
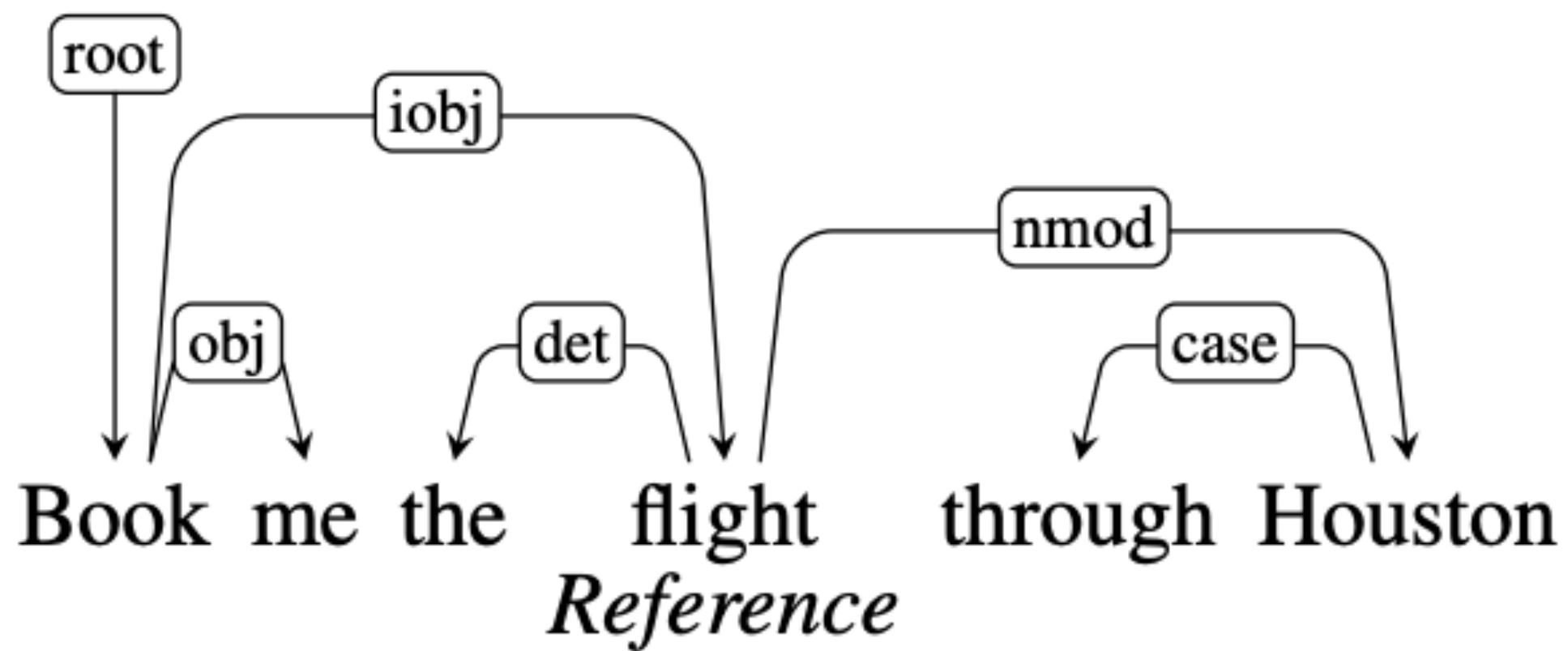
Graph-based vs. Transition-based

- ▶ Graph based:
 - ▶ Can find exact best global solution with dynamic programming
 - ▶ Local independence assumption
- ▶ Transition based:
 - ▶ Greedy algorithm
 - ▶ Cannot model non-projective trees
 - ▶ Can condition on longer tree context



Evaluation

- ▶ Unlabeled attachment score (UAS)
 - ▶ Percentage of words that have been assigned the correct head
- ▶ Labeled attachment score (LAS)
 - ▶ Percentage of words that have been assigned the correct head & edge label





Results

Type	Model	English PTB-SD 3.3.0		Chinese PTB 5.1	
		UAS	LAS	UAS	LAS
Transition	Ballesteros et al. (2016)	93.56	91.42	87.65	86.21
	Andor et al. (2016)	94.61	92.79	—	—
	Kuncoro et al. (2016)	95.8	94.6	—	—
Graph	Kiperwasser & Goldberg (2016)	93.9	91.9	87.6	86.1
	Cheng et al. (2016)	94.10	91.49	88.1	85.7
	Hashimoto et al. (2016)	94.67	92.90	—	—
	Deep Biaffine	95.74	94.08	89.30	88.23

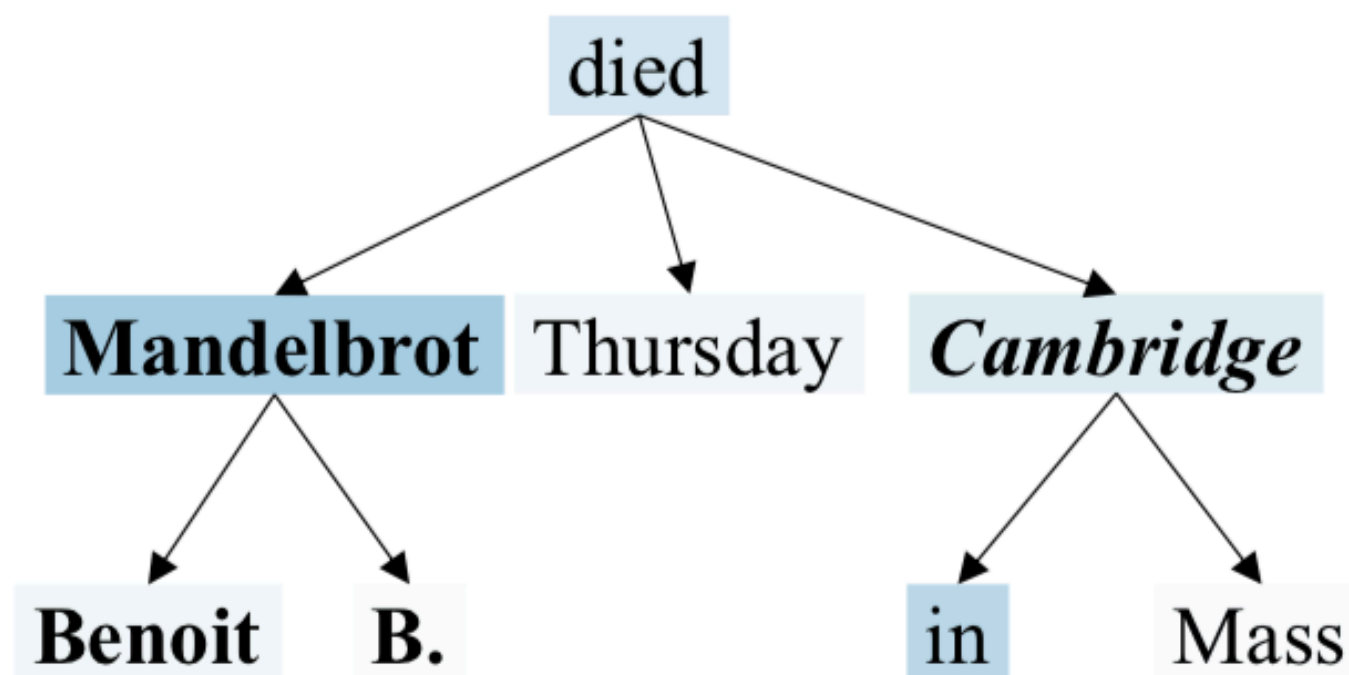


Using Dependency Parsers

- ▶ For information extraction
- ▶ Features capturing dependency path between entities

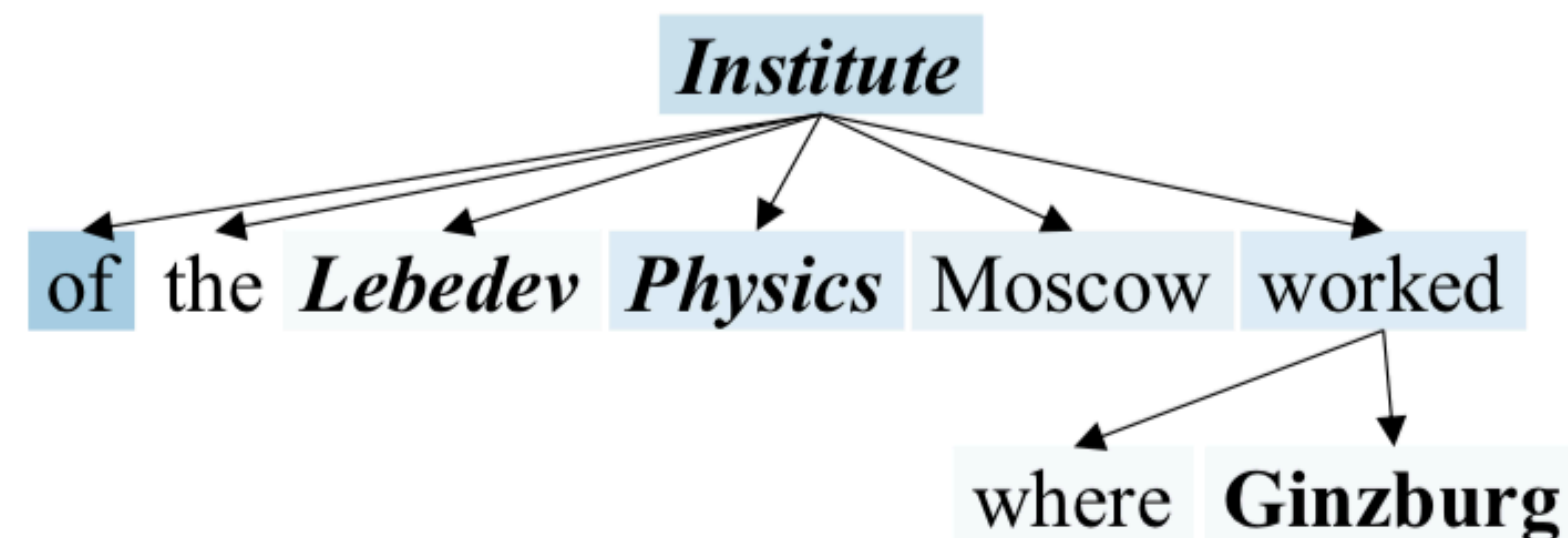
Relation: *per:city_of_death*

Benoit B. Mandelbrot, a maverick mathematician who developed an innovative theory of roughness and applied it to physics, biology, finance and many other fields, died Thursday in **Cambridge**, Mass.



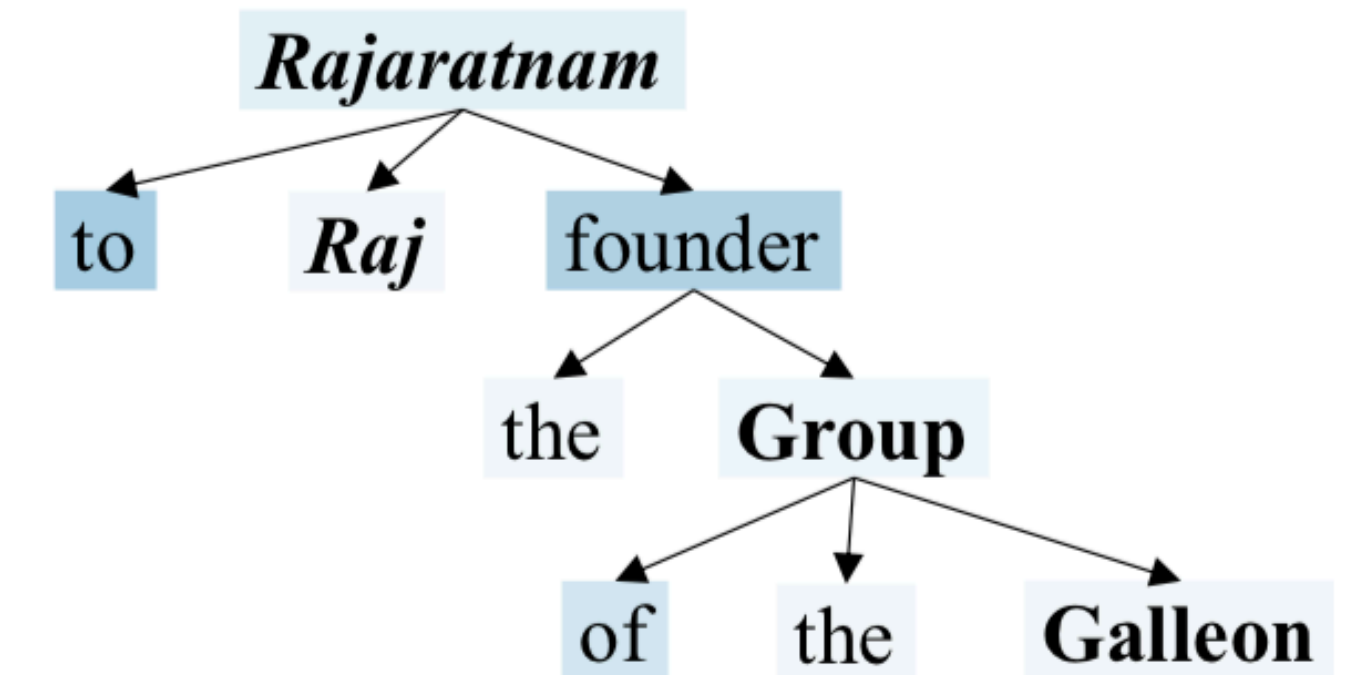
Relation: *per:employee_of*

In a career that spanned seven decades, Ginzburg authored several groundbreaking studies in various fields -- such as quantum theory, astrophysics, radio-astronomy and diffusion of cosmic radiation in the Earth's atmosphere -- that were of "Nobel Prize caliber," said Gennady Mesyats, the director of the **Lebedev Physics Institute** in Moscow, where **Ginzburg** worked .



Relation: *org:founded_by*

Anil Kumar, a former director at the consulting firm McKinsey & Co, pleaded guilty on Thursday to providing inside information to **Raj Rajaratnam**, the founder of the **Galleon Group**, in exchange for payments of at least \$ 175 million from 2004 through 2009.





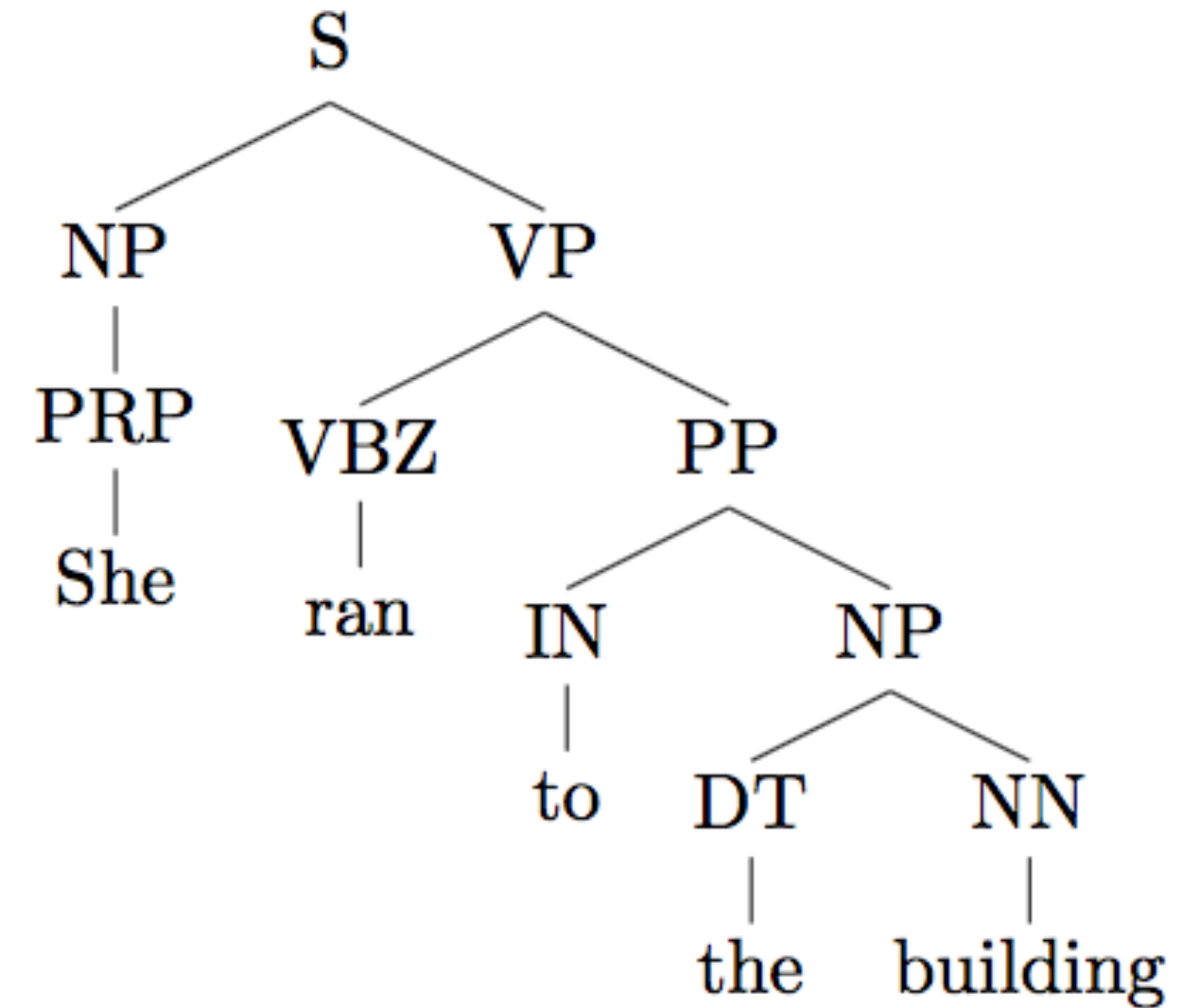
Trees

- ▶ Overview
- ▶ Dependency Parsing [Tuesday]
- ▶ Constituency Parsing [Today]
 - ▶ Overview
 - ▶ Probabilistic context-free grammar
 - ▶ Constituency parsing / Inference
 - ▶ Comparison with dependency parsing



Constituency Parsing

- ▶ Constituent: a ***unit*** that can appear in different places
 - ▶ Common constituents: noun phrases, verb phrases, prepositional phrases
- ▶ Phrase structure organizes words into nested constituents





Constituency structure

- ▶ Starting units: **words** are given a **category** (part-of-speech tags)

the, cuddly, cat, by, the, door

Det, Adj, N, P, Det, N

- ▶ Words combine into **phrases** with **categories**

the cuddly cat, by the door

$NP \rightarrow \text{Det Adj N}$ $PP \rightarrow \text{P NP}$

- ▶ Phrases can combine into **bigger phrases** (with category) recursively

the cuddly cat by the door

$NP \rightarrow \text{NP PP}$

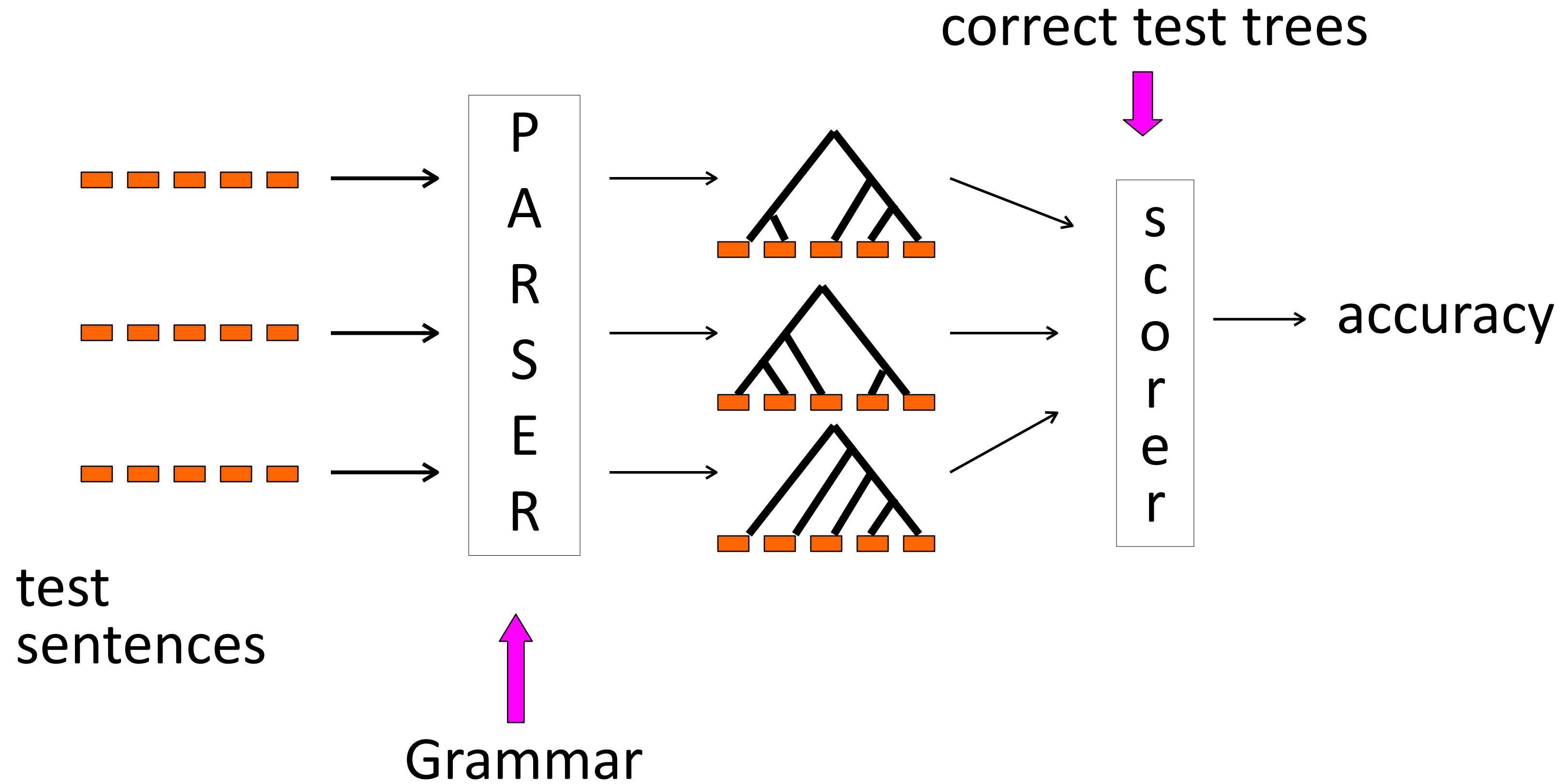


Constituency Tests

- ▶ A constituent behaves as a ***unit*** that can appear in different places
 - ▶ John talked to the children about drugs.
 - ▶ John talked [to the children] [about drugs].
 - ▶ John talked [about drugs] [to the children].
 - ▶ *John talked drugs to the children about
- ▶ Substitution by *proform* (e.g., pronoun)
- ▶ Rewriting: (*It was with a spoon that...*)
- ▶ Question Answering
 - ▶ (What did they eat? *the cake*)
 - ▶ (How? *with a spoon*)



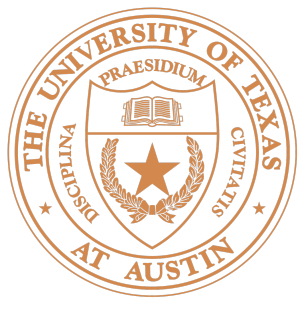
The Parsing Problem





Context-Free Grammar (CFG)

- ▶ Formal system for modeling constituency structure in language
- ▶ A context-free grammar is a tuple $\langle N, \Sigma, S, R \rangle$
 - ▶ N : a set of non terminal symbols
 - ▶ Σ : a set of terminal symbols
 - ▶ R : set of rules
 - ▶ S : a start symbol



Context-Free Grammar (CFG)

N : the set of non-terminal symbols

Phrasal categories: S, NP, VP, ADJP, etc.

Parts-of-speech (pre-terminals): NN, JJ, DT, VB

Σ : the set of terminal symbols (the words)

R : the set of rules

Of the form $X \rightarrow Y_1 Y_2 \dots Y_n$, with $X \in N$, $n \geq 0$, $Y_i \in (N \cup \Sigma)$

Examples: $S \rightarrow NP VP$, $VP \rightarrow VP CC VP$

S : the start symbol



Example

Writing parsing rules:

$N \rightarrow \text{Fed}$

$V \rightarrow \text{raises}$

$NP \rightarrow N$

$S \rightarrow NP VP$

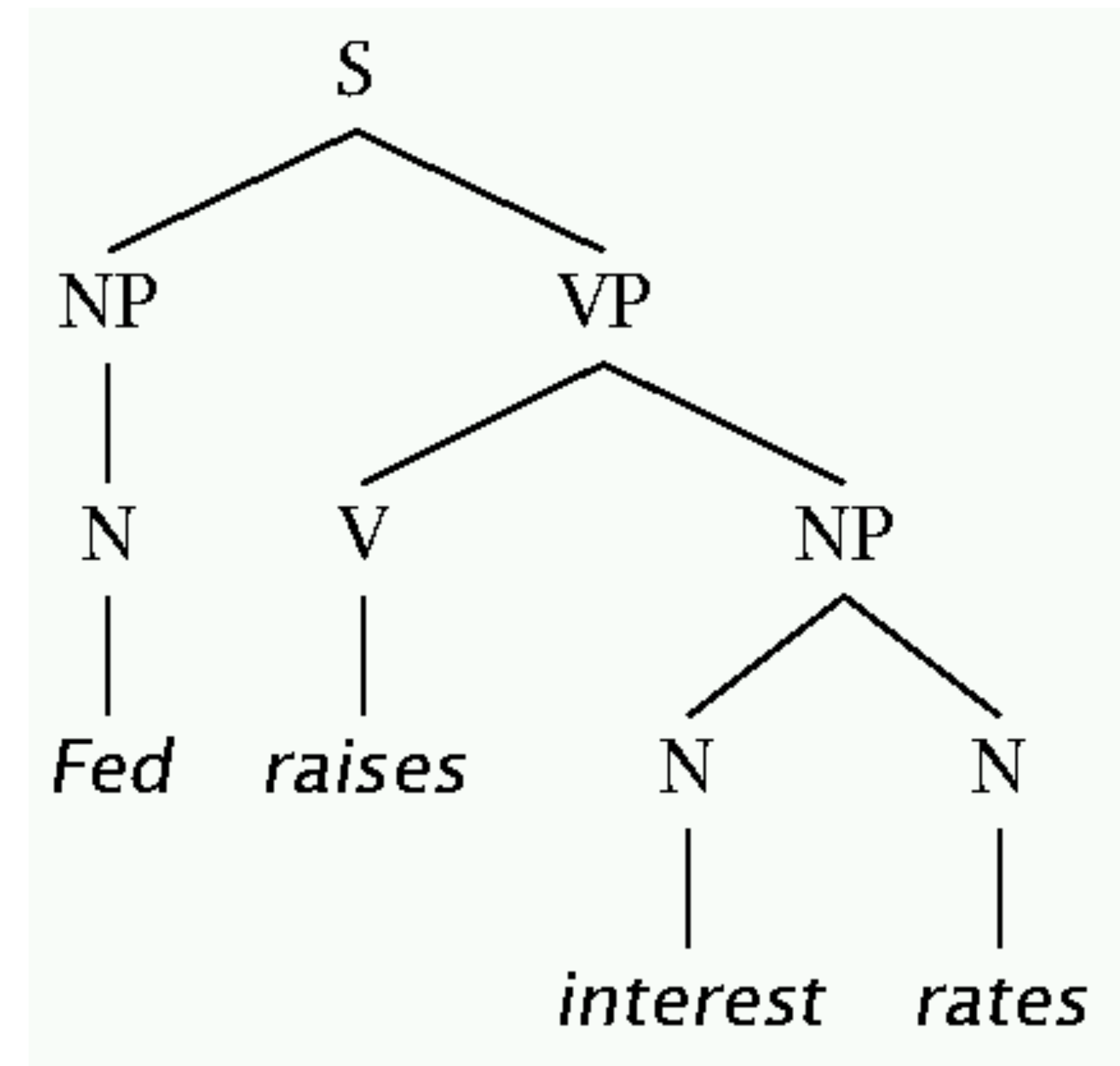
$VP \rightarrow V NP$

$NP \rightarrow N N$

$NP \rightarrow NP PP$

$N \rightarrow \text{interest}$

$N \rightarrow \text{raises}$





Example Grammar

$N = \{S, NP, VP, PP, DT, Vi, Vt, NN, IN\}$

$S = S$

$\Sigma = \{\text{sleeps, saw, man, woman, telescope, the, with, in}\}$

$R =$

S	→	NP	VP
VP	→	Vi	
VP	→	Vt	NP
VP	→	VP	PP
NP	→	DT	NN
NP	→	NP	PP
PP	→	IN	NP

Grammar

Vi	→	sleeps
Vt	→	saw
NN	→	man
NN	→	woman
NN	→	telescope
NN	→	dog
DT	→	the
IN	→	with
IN	→	in

Lexicon

S=sentence, VP=verb phrase, NP=noun phrase, PP=prepositional phrase,
DT=determiner, Vi=intransitive verb, Vt=transitive verb, NN=noun, IN=preposition



Example Parses

$N = \{S, NP, VP, PP, DT, Vi, Vt, NN, IN\}$

$S = S$

$\Sigma = \{\text{sleeps, saw, man, woman, telescope, the, with, in}\}$

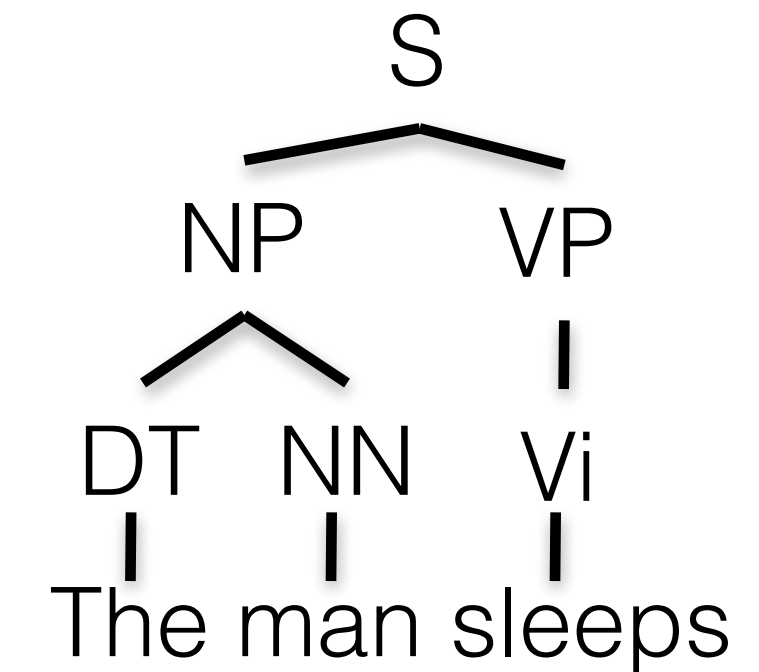
$R =$

S	→	NP	VP
VP	→	Vi	
VP	→	Vt	NP
VP	→	VP	PP
NP	→	DT	NN
NP	→	NP	PP
PP	→	IN	NP

Grammar

Vi	→	sleeps
Vt	→	saw
NN	→	man
NN	→	woman
NN	→	telescope
NN	→	dog
DT	→	the
IN	→	with
IN	→	in

Lexicon



S=sentence, VP=verb phrase, NP=noun phrase, PP=prepositional phrase,
DT=determiner, Vi=intransitive verb, Vt=transitive verb, NN=noun, IN=preposition



Example Parses

$N = \{S, NP, VP, PP, DT, Vi, Vt, NN, IN\}$

$S = S$

$\Sigma = \{\text{sleeps, saw, man, woman, telescope, the, with, in}\}$

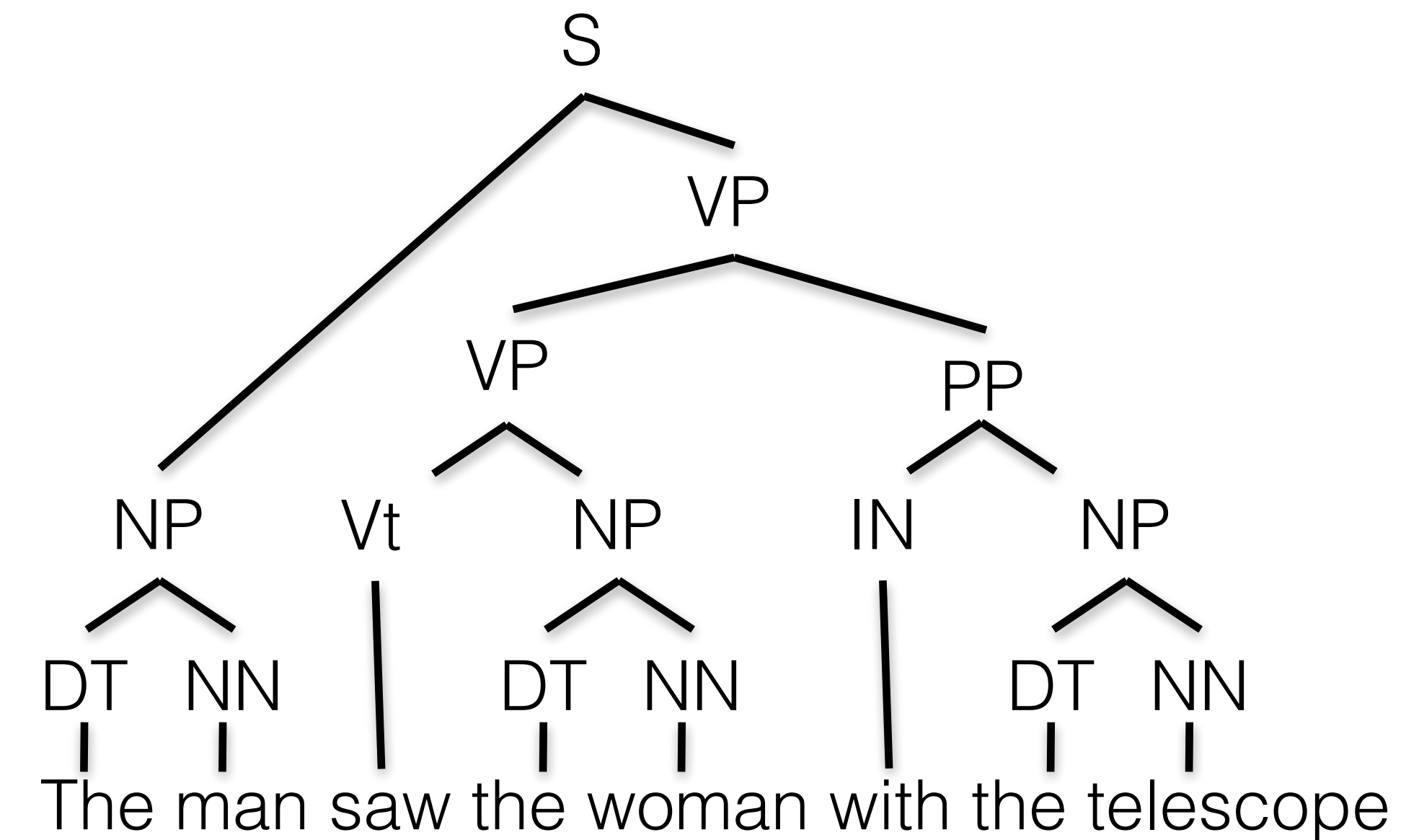
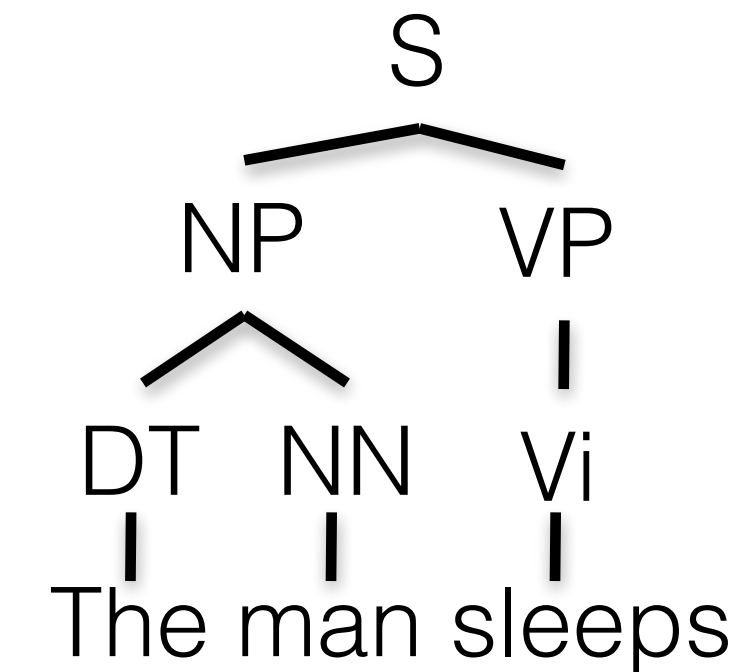
$R =$

S	→	NP	VP
VP	→	Vi	
VP	→	Vt	NP
VP	→	VP	PP
NP	→	DT	NN
NP	→	NP	PP
PP	→	IN	NP

Grammar

Vi	→	sleeps
Vt	→	saw
NN	→	man
NN	→	woman
NN	→	telescope
NN	→	dog
DT	→	the
IN	→	with
IN	→	in

Lexicon



S=sentence, VP=verb phrase, NP=noun phrase, PP=prepositional phrase,
DT=determiner, Vi=intransitive verb, Vt=transitive verb, NN=noun, IN=preposition



“Classical” NLP parsing

- ▶ Sentences can have a very large number of possible parses

$((ab)c)d$ $(a(bc))d$ $(ab)(cd)$ $a((bc)d)$ $a(b(cd))$

- ▶ We can add constraints to the grammar to limit unlikely parses, but then we have a coverage, unable to parse some sentences
- ▶ Less constrained grammar \rightarrow High coverage — blow up of the number of possible parses



Statistical Parsing

- ▶ Learning from data: Treebanks
- ▶ Adding probabilities to the rules: probabilistic CFGs (PCFGs)

Treebanks: a collection of sentences paired with their parse trees

```
((S
  (NP-SBJ (DT That)
    (JJ cold) (, ,)
    (JJ empty) (NN sky) )
  (VP (VBD was)
    (ADJP-PRD (JJ full)
      (PP (IN of)
        (NP (NN fire)
          (CC and)
          (NN light) ))))
  (. .) ))
```

(a)

```
((S
  (NP-SBJ The/DT flight/NN )
  (VP should/MD
    (VP arrive/VB
      (PP-TMP at/IN
        (NP eleven/CD a.m/RB ))
      (NP-TMP tomorrow/NN )))))
```

(b)

50K annotated sentences

The Penn Treebank Project (Marcus et al, 1993)



Probabilistic context free grammar

S	\Rightarrow	NP	VP	1.0
VP	\Rightarrow	Vi		0.4
VP	\Rightarrow	Vt	NP	0.4
VP	\Rightarrow	VP	PP	0.2
NP	\Rightarrow	DT	NN	0.3
NP	\Rightarrow	NP	PP	0.7
PP	\Rightarrow	P	NP	1.0

Vi	\Rightarrow	sleeps	1.0
Vt	\Rightarrow	saw	1.0
NN	\Rightarrow	man	0.7
NN	\Rightarrow	woman	0.2
NN	\Rightarrow	telescope	0.1
DT	\Rightarrow	the	1.0
IN	\Rightarrow	with	0.5
IN	\Rightarrow	in	0.5

A context-free grammar: $G = (N, \Sigma, R, S)$

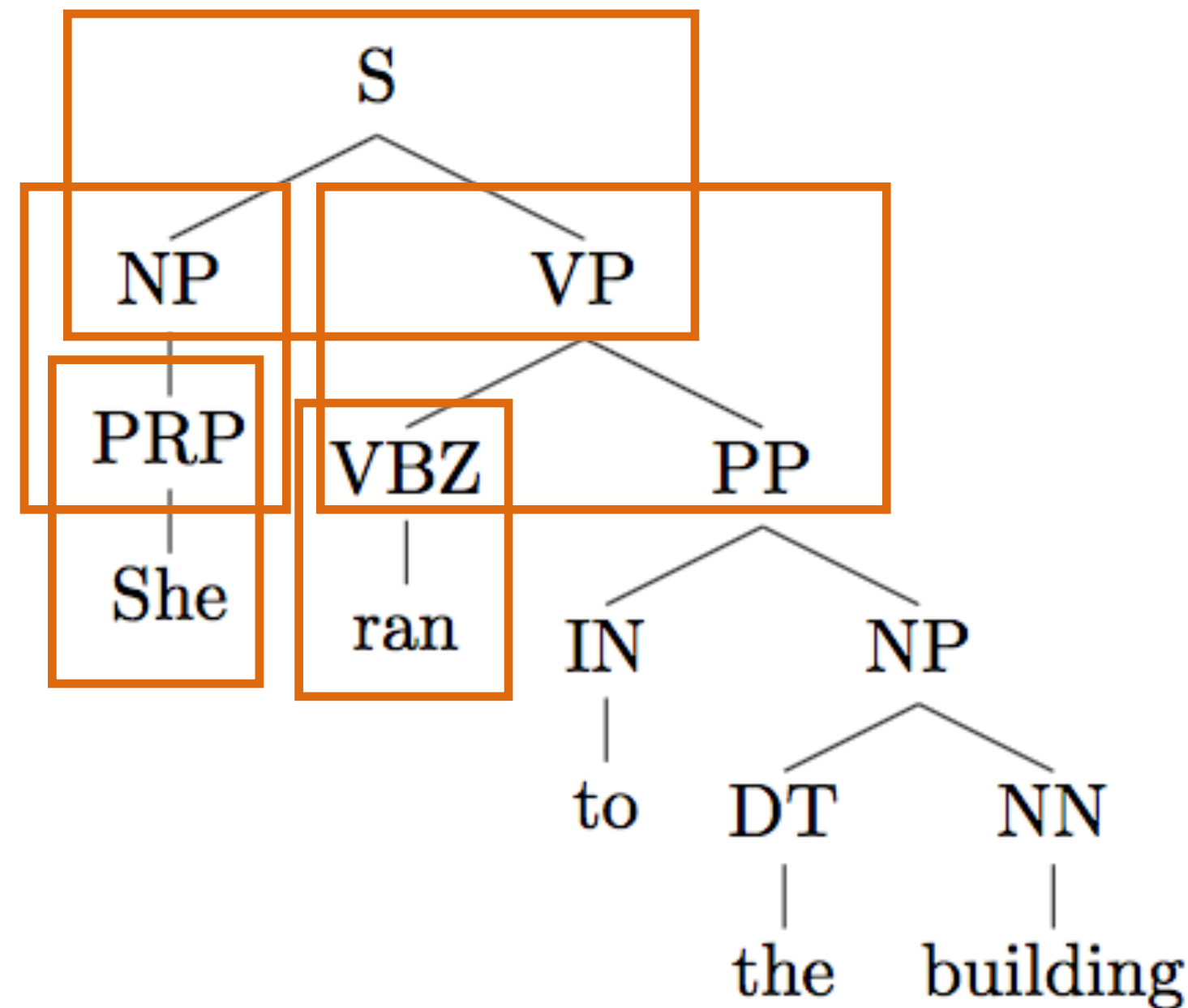
For each rule $\alpha \rightarrow \beta \in R$, there is a parameter $q(\alpha \rightarrow \beta) \geq 0$. For any $X \in N$,

$$\sum_{\alpha \rightarrow \beta: \alpha = X} q(\alpha \rightarrow \beta) = 1$$



Probabilistic context free grammar

- ▶ Tree T is a series of rule applications r .



$S \rightarrow NP VP$ 1.0

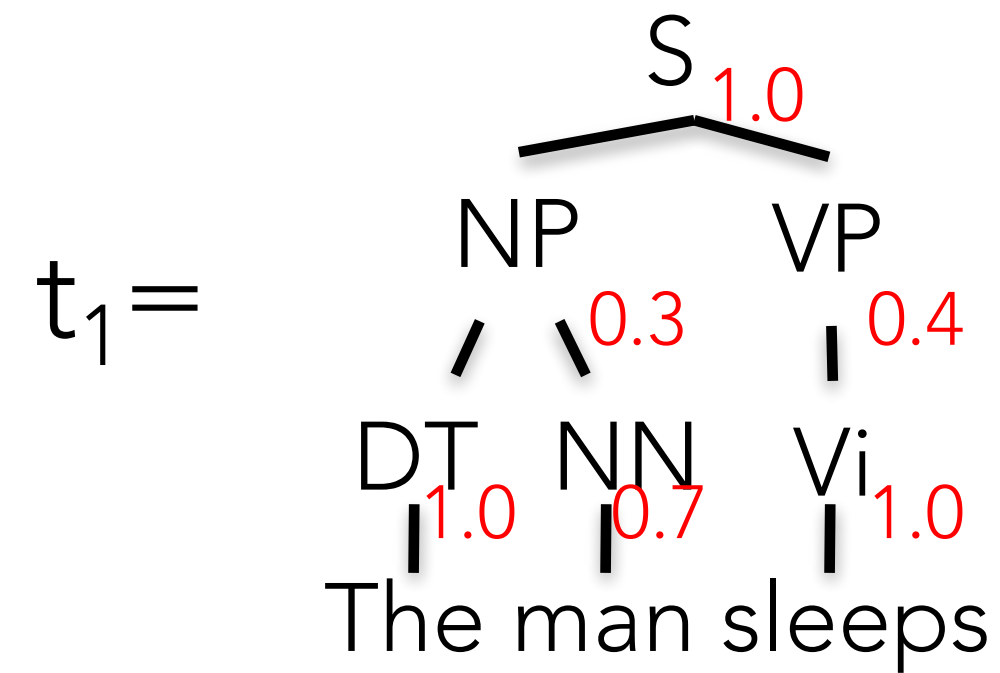
$NP \rightarrow PRP$ 0.5

$NP \rightarrow DT NN$ 0.5

...

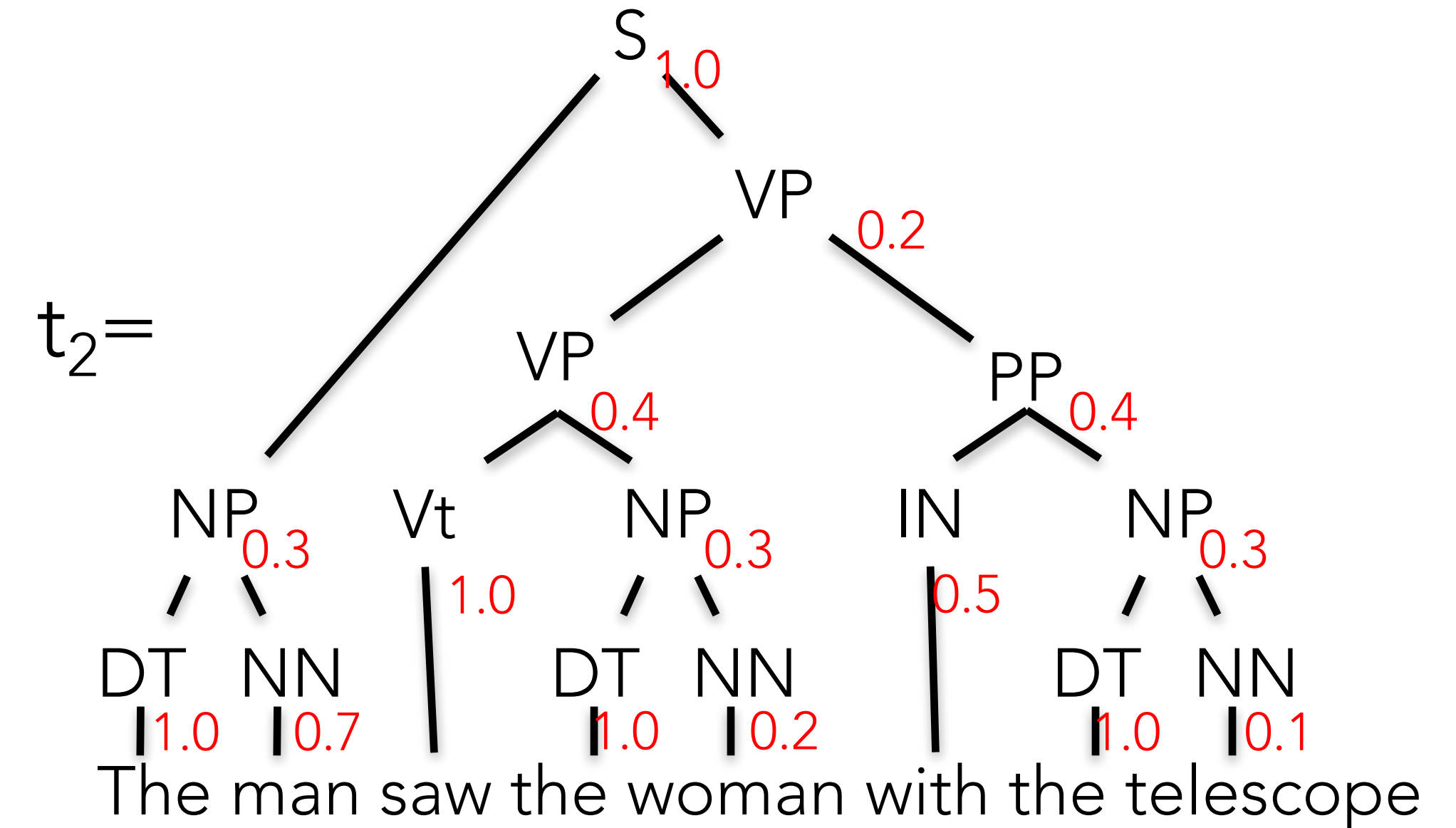


Tree Scorer



$$P(t_1) = q(S \rightarrow NP VP) \times q(NP \rightarrow DT NN) \times q(DT \rightarrow \text{the}) \\ \times q(NN \rightarrow \text{man}) \times q(VP \rightarrow Vi) \times q(Vi \rightarrow \text{sleeps})$$

$$= 1.0 \times 0.3 \times 1.0 \times 0.7 \times 0.4 \times 1.0 = 0.084$$



$$p(t_s) = 1.0 * 0.3 * 1.0 * 0.7 * 0.2 * 0.4 * 1.0 * 0.3 * 1.0 * 0.2 * 0.4 * 0.5 * 0.3 * 1.0 * 0.1$$



Deriving a PCFG from a treebank

- ▶ Training data: a set of parse trees t_1, t_2, \dots, t_m
- ▶ A PCFG (N, Σ, S, R, q) :
 - ▶ N is the set of all non-terminals seen in the trees
 - ▶ Σ is the set of all words seen in the trees
 - ▶ S is taken to be S .
 - ▶ R is taken to be the set of all rules $\alpha \rightarrow \beta$ seen in the trees
- ▶ The maximum-likelihood parameter estimates are:

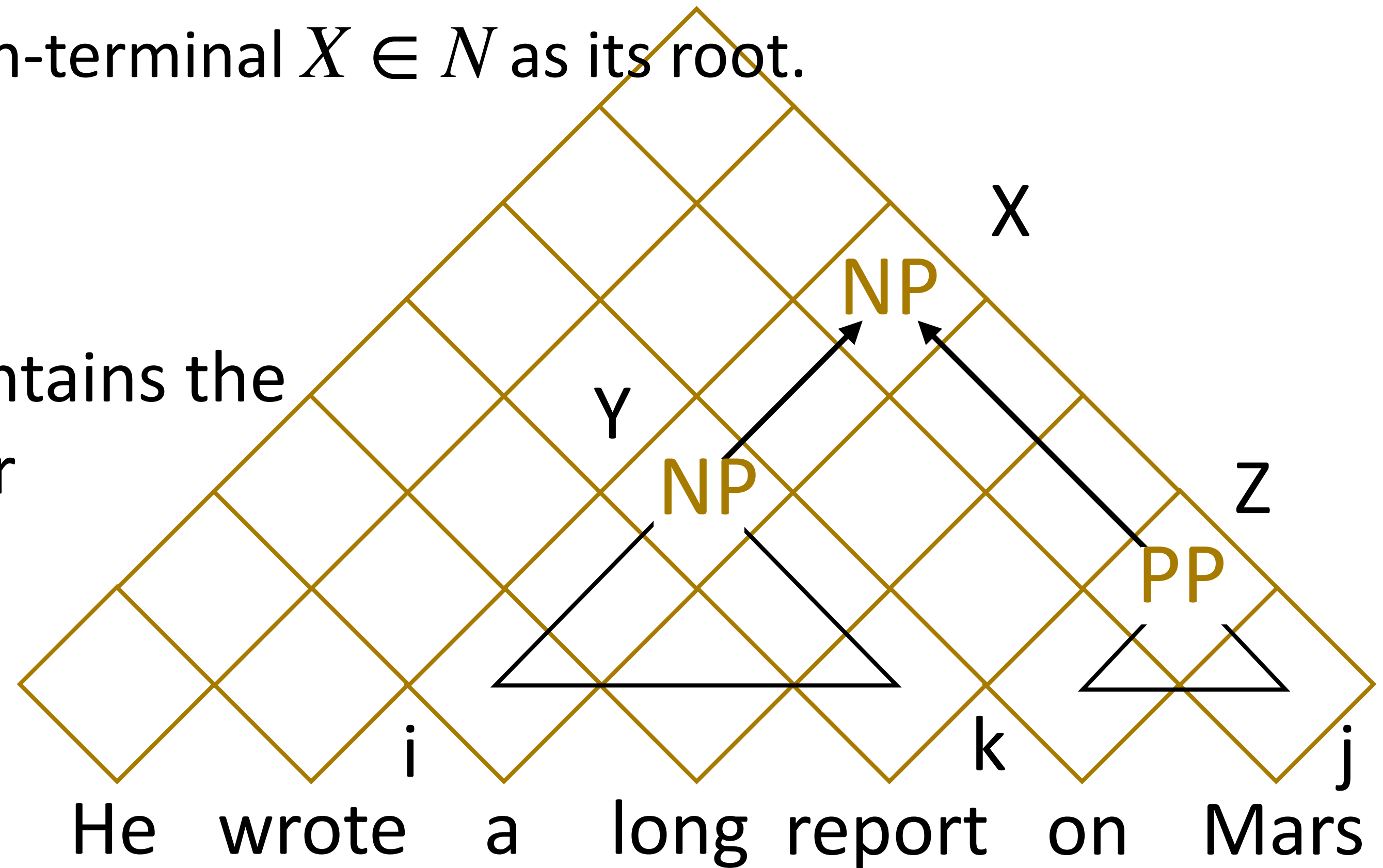
$$q_{ML}(\alpha \rightarrow \beta) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

If we have seen the rule $VP \rightarrow Vt\ NP$ 105 times, and the the non-terminal VP 1000 times, $q(VP \rightarrow Vt\ NP) = 0.105$

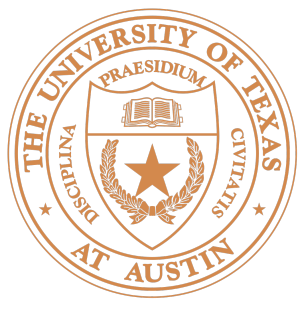


Inference

- ▶ Given a sentence x_1, x_2, \dots, x_n , denote $T[i,j,X]$ as the highest score for any parse tree that covers words x_i, \dots, x_{j-1} with non-terminal $X \in N$ as its root.
- ▶ Find $T[1, n+1, S]$
- ▶ Dynamic programming: chart maintains the best way of building symbol X over span (i, j)
- ▶ Same idea with Viterbi algorithm!

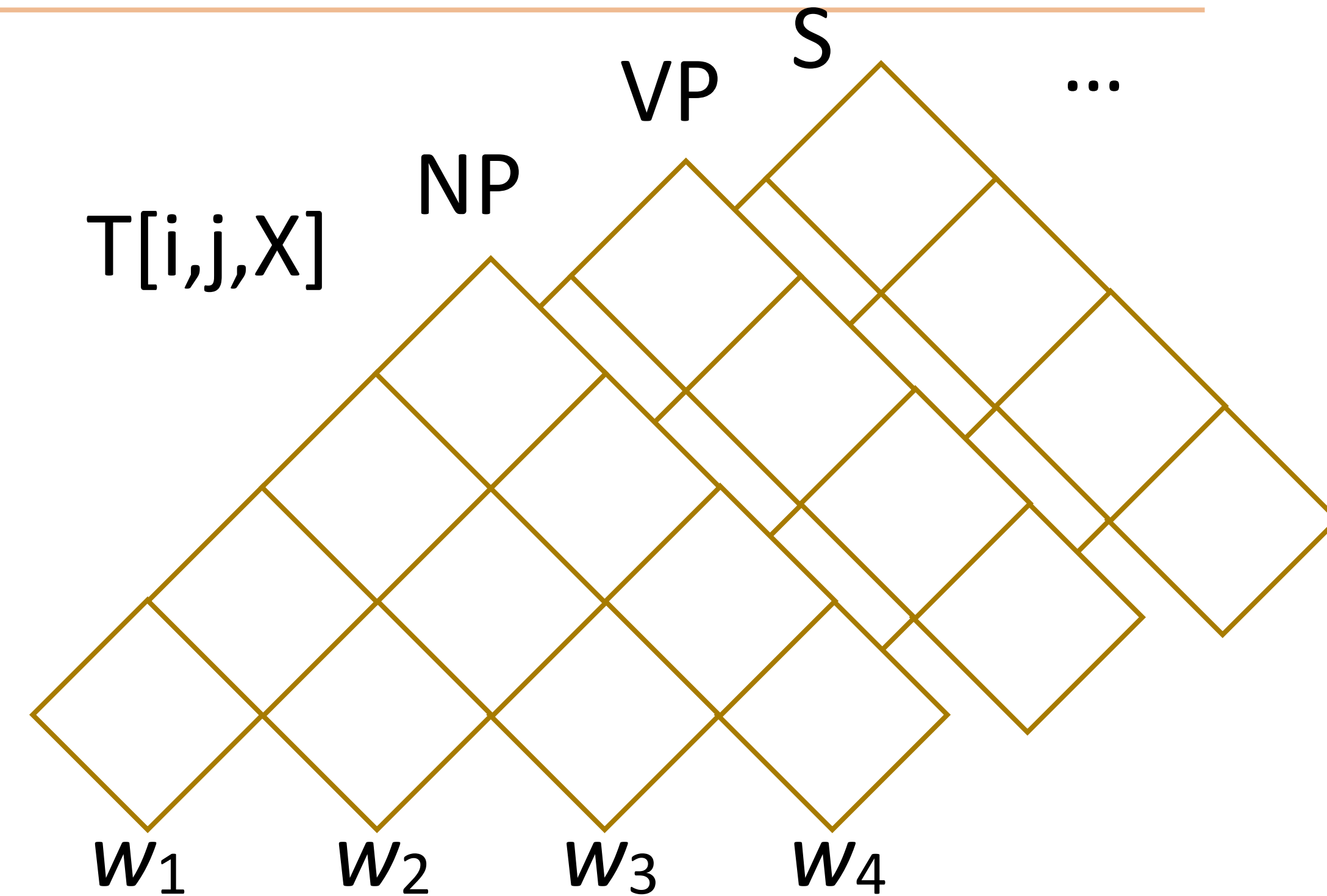


Cocke-Kasami-Younger (CKY algorithm)



Cocke-Kasami-Younger (CKY algorithm)

- ▶ Chart: $T[i,j,X] = \text{best score}$
- ▶ Base: $T[i,i+1,X] = \log P(X \rightarrow w_i)$
- ▶ Loop over all split points k ,
apply rules $X \rightarrow Y Z$ to build
 X in every possible way
- ▶ Recurrence:
$$T[i,j,X] = \max_k \max_{r: X \rightarrow X1 X2} T[i,k,X1] + T[k,j,X2] + \log P(X \rightarrow X1 X2)$$

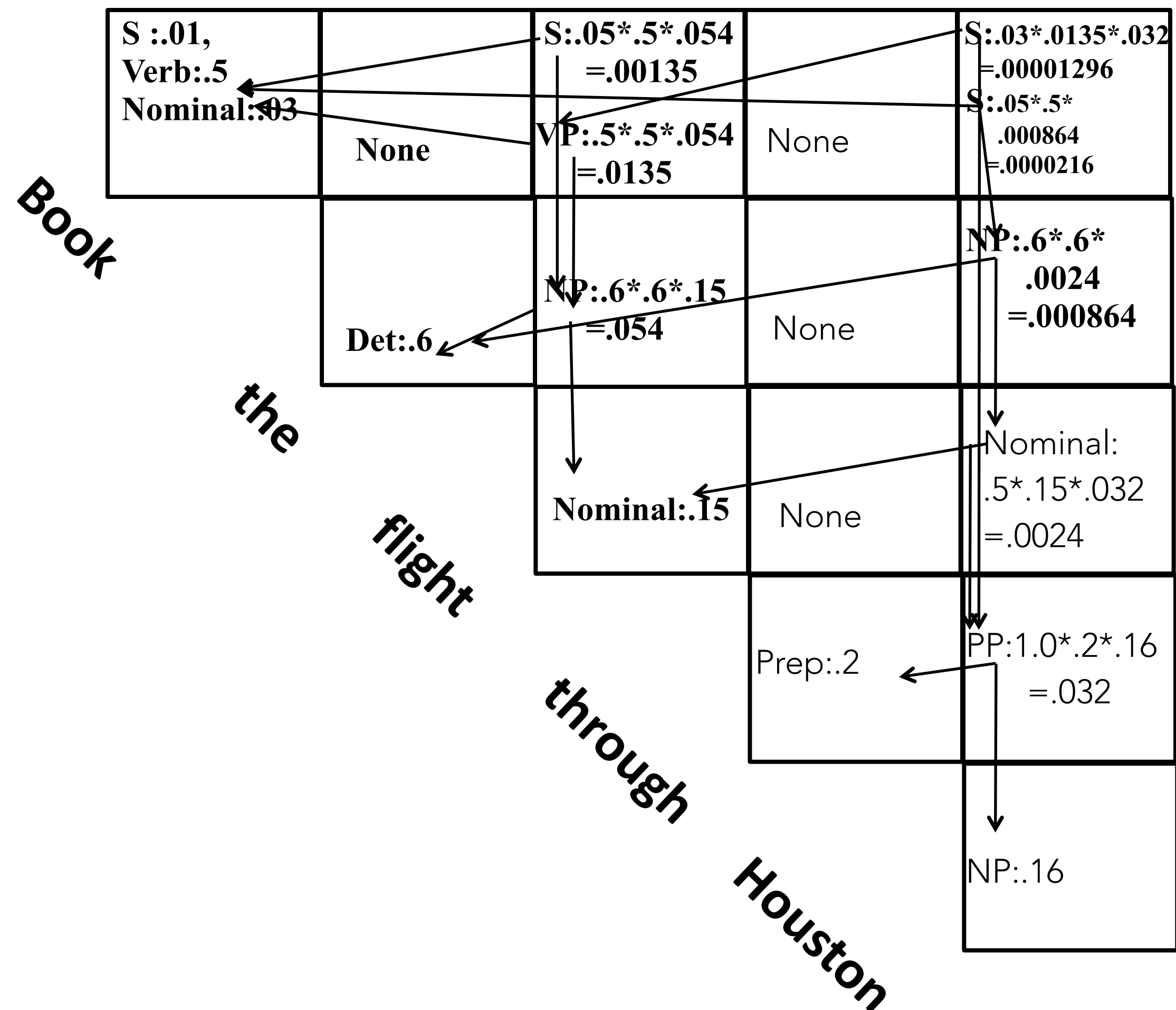


$S[0,4] \Rightarrow NP[0,2] VP[2,4]$



CKY [Example]

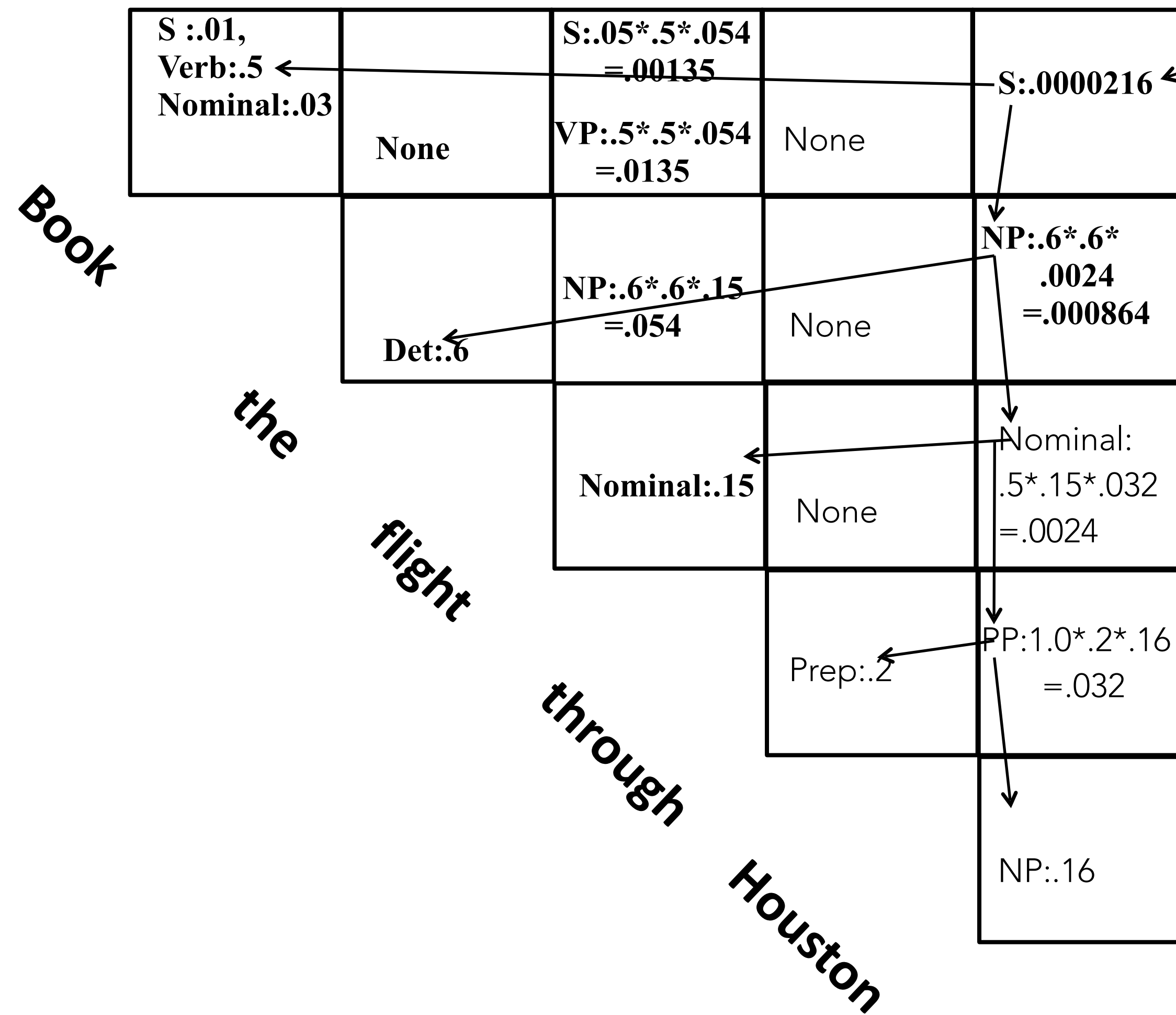
$S \rightarrow NP VP$	0.8
$S \rightarrow X1 VP$	0.1
$X1 \rightarrow Aux NP$	1.0
$S \rightarrow \text{book} \mid \text{include} \mid \text{prefer}$ 0.01 0.004 0.006	
$S \rightarrow \text{Verb NP}$	0.05
$S \rightarrow VP PP$	0.03
$NP \rightarrow I \mid \text{he} \mid \text{she} \mid \text{me}$ 0.1 0.02 0.02 0.06	
$NP \rightarrow \text{Houston} \mid \text{NWA}$ 0.16 0.04	
$Det \rightarrow \text{the} \mid \text{a} \mid \text{an}$ 0.6 0.1 0.05	
$NP \rightarrow Det Nominal$	0.6
$Nominal \rightarrow \text{book} \mid \text{flight} \mid \text{meal} \mid \text{money}$ 0.03 0.15 0.06 0.06	
$Nominal \rightarrow Nominal Nominal$	0.2
$Nominal \rightarrow Nominal PP$	0.5
$Verb \rightarrow \text{book} \mid \text{include} \mid \text{prefer}$ 0.5 0.04 0.06	
$VP \rightarrow Verb NP$	0.5
$VP \rightarrow VP PP$	0.3
$Prep \rightarrow \text{through} \mid \text{to} \mid \text{from}$ 0.2 0.3 0.3	
$PP \rightarrow Prep NP$	1.0





CKY [Example]

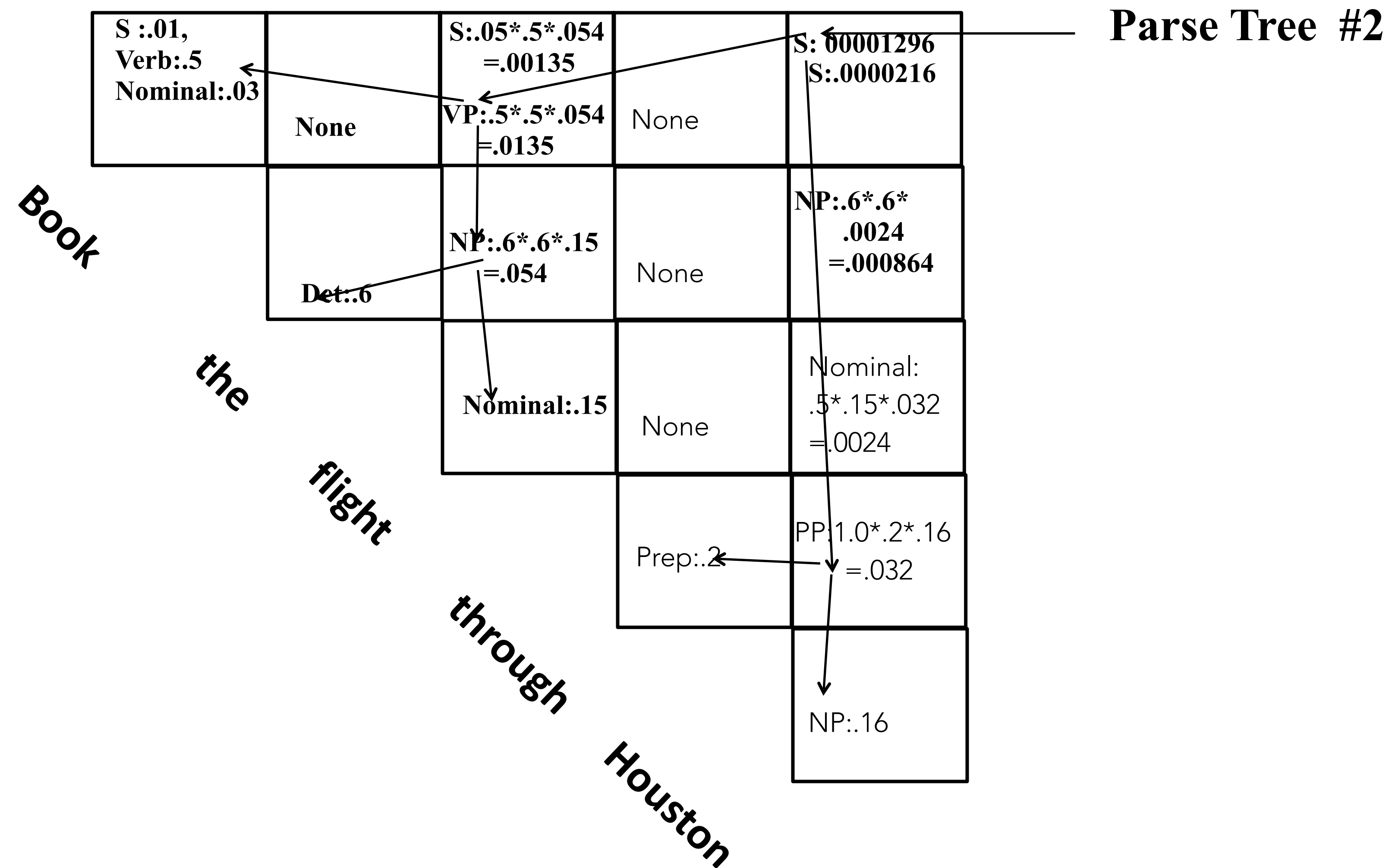
Parse Tree #1



Pick most probable parse, i.e. take max to combine probabilities of multiple derivations of each constituent in each cell.



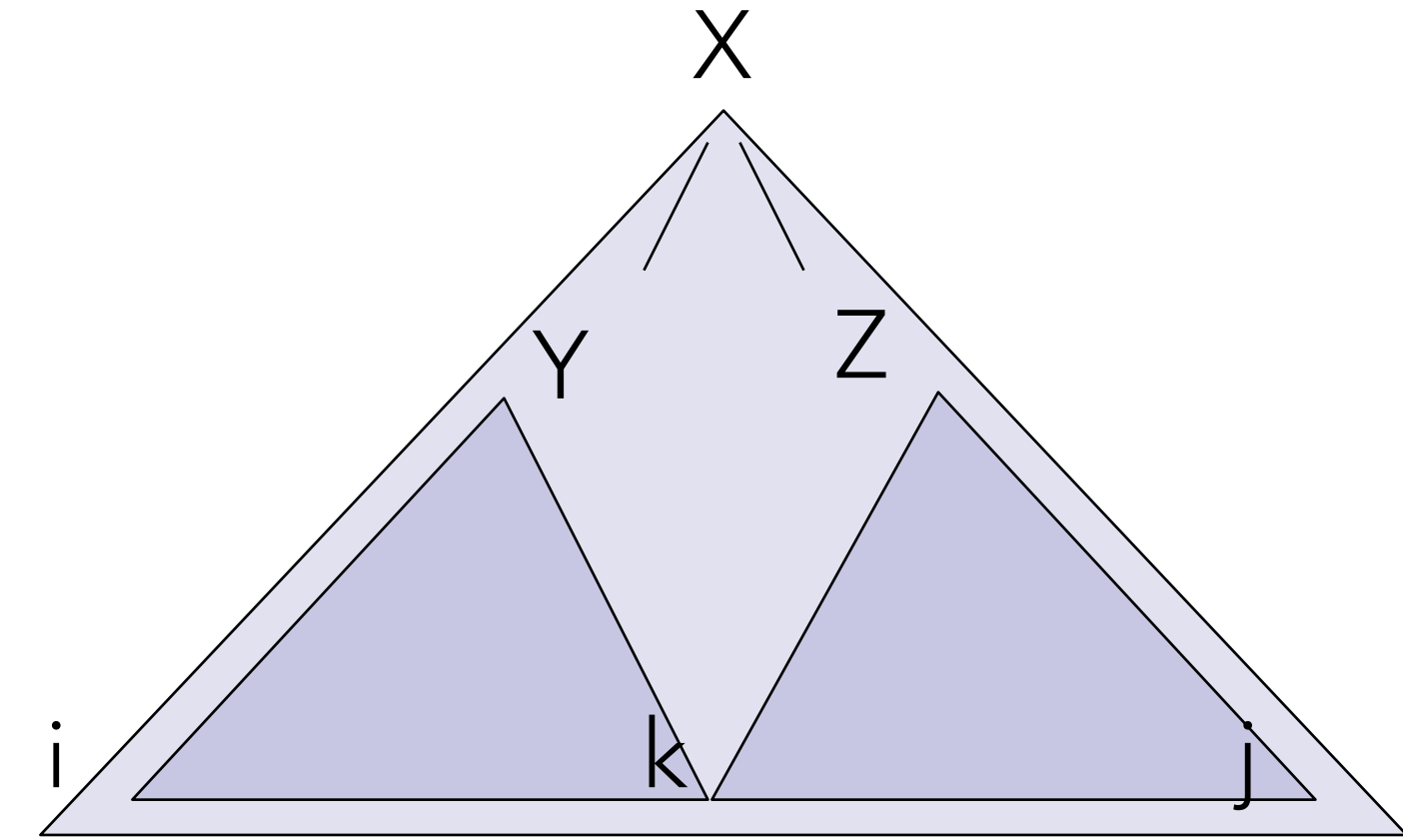
CKY [Example]





Efficiency?

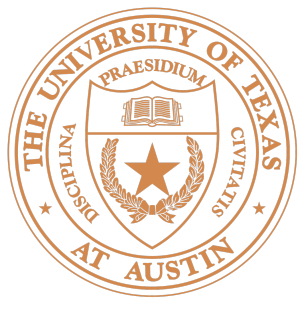
- ▶ Runtime: $O(n^3 * |R|)$ R = grammar constant
 - For each length ($\leq n$)
 - For each i ($\leq n$)
 - For each split point k
 - For each rule $X \rightarrow Y Z$
 - » Do constant work
- ▶ Memory:
 - ▶ Need to store score caches
 - ▶ Cache size: $|Symbols| * n^2$



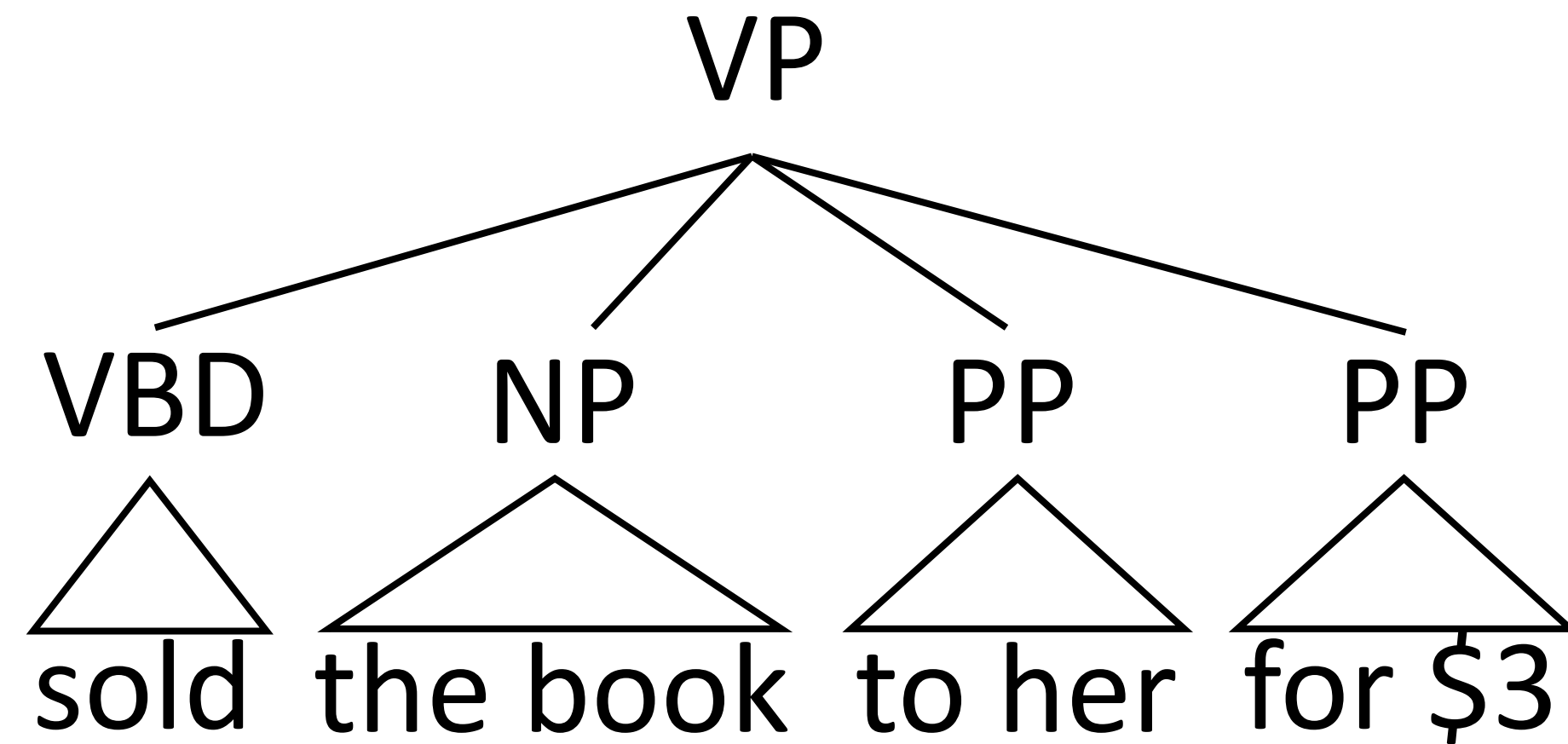


Efficiency?

- ▶ Can we keep N-ary ($N > 2$) rules and still do dynamic programming?
- ▶ Can we keep unary rules and still do dynamic programming?
- ▶ Binary trees over n words have at most $n-1$ nodes, but you can have unlimited numbers of nodes with unaries ($S \rightarrow \text{SBAR} \rightarrow \text{NP} \rightarrow S \rightarrow \dots$)
- ▶ We introduce Chomsky Normal Form (CNF):
 - ▶ All rules are either $X \rightarrow YZ$ or $X \rightarrow w$
 - ▶ Rewriting PCFG grammar into equivalent CNF is possible



Binarization

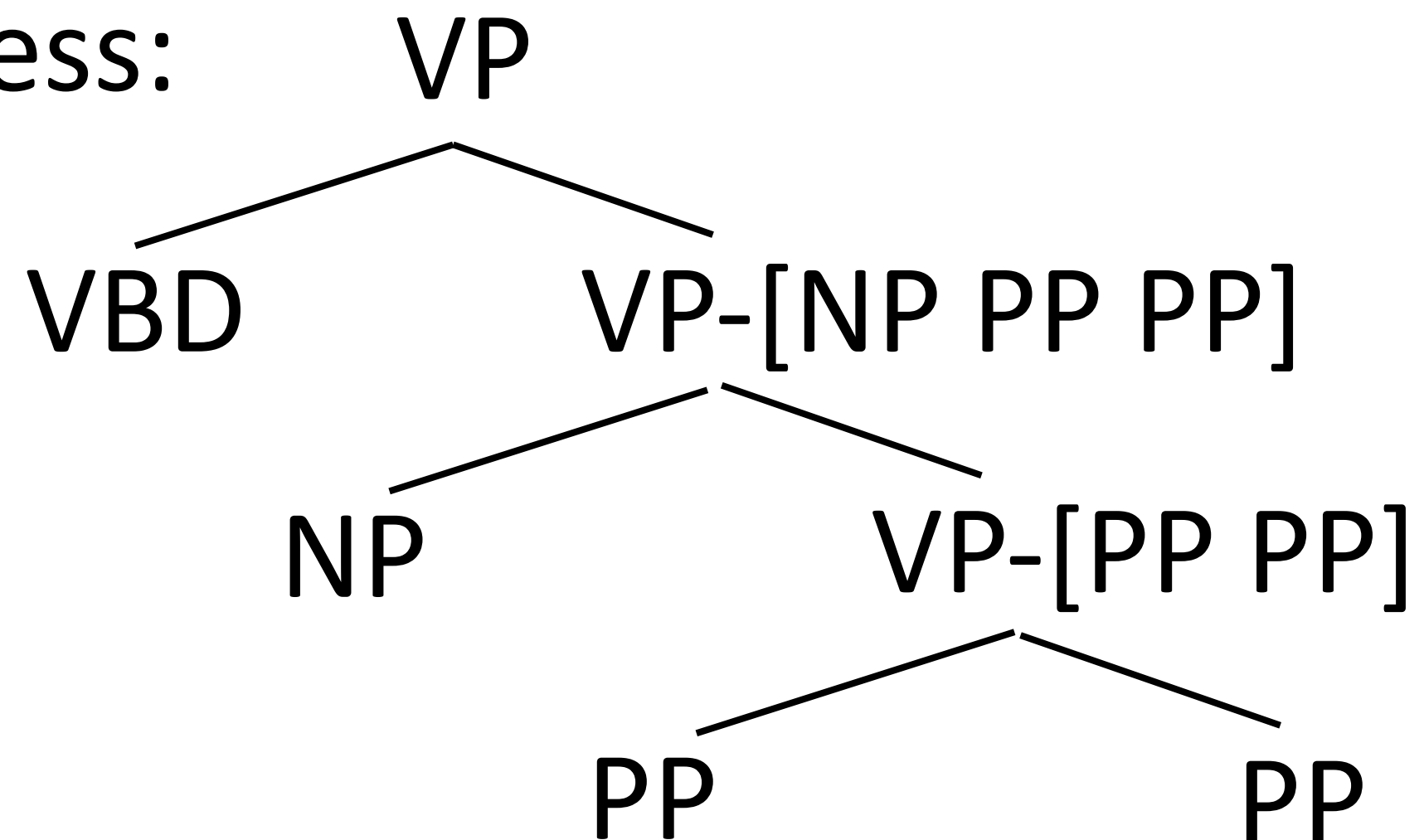


$$P(\text{VP} \rightarrow \text{VBD NP PP PP}) = 0.2$$

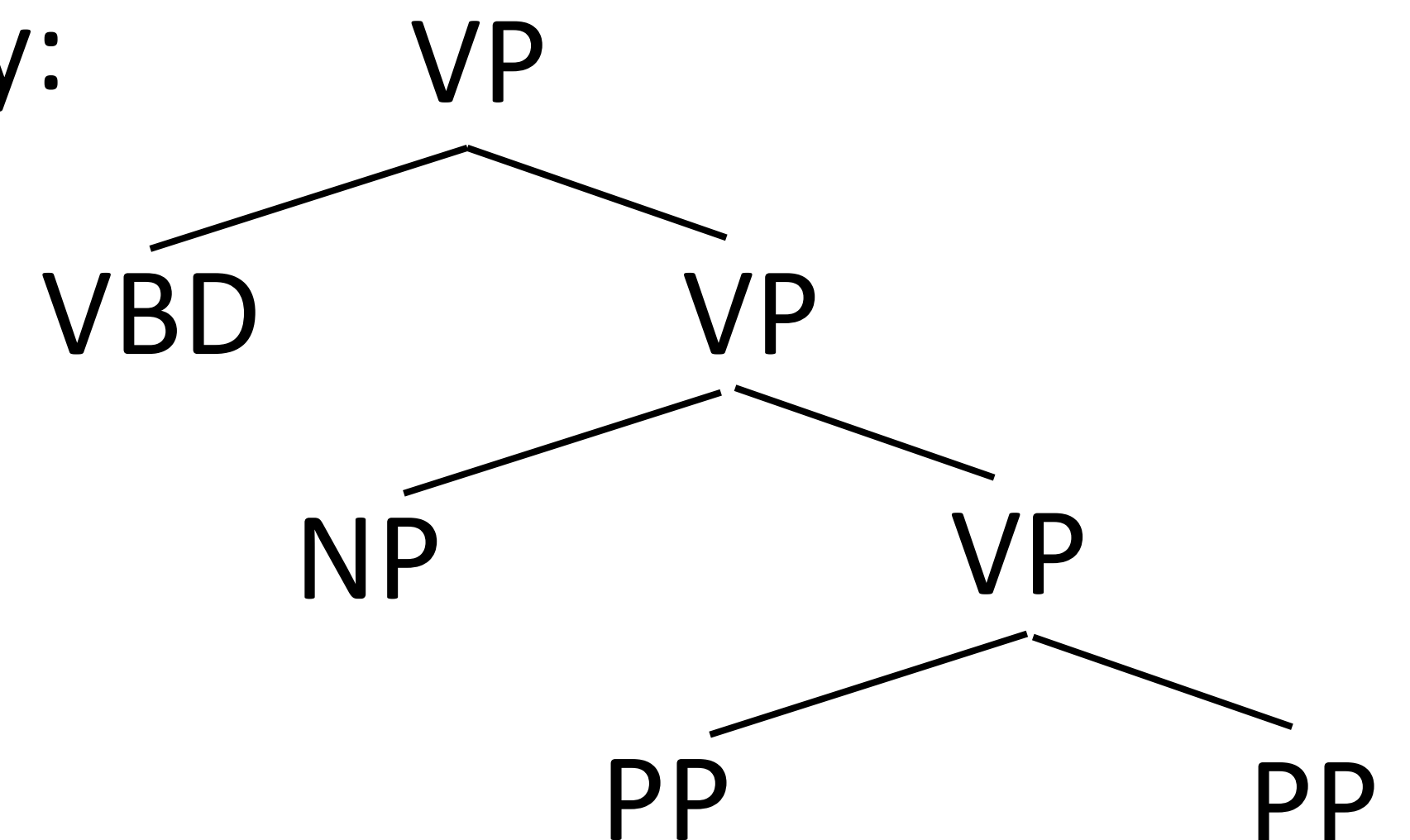
$$P(\text{VP} \rightarrow \text{VBZ PP}) = 0.1$$

...

► Lossless:



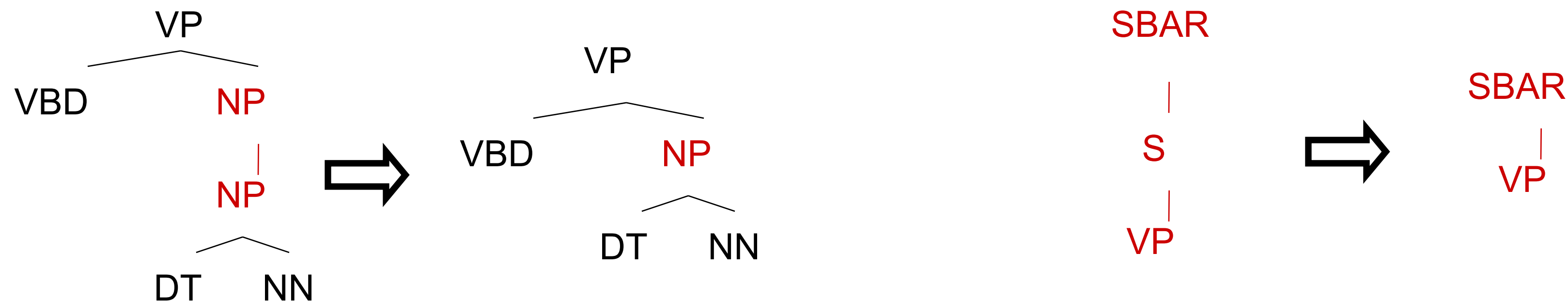
► Lossy:





Unary Rules

- Enforce at most one unary over each span by modifying grammar

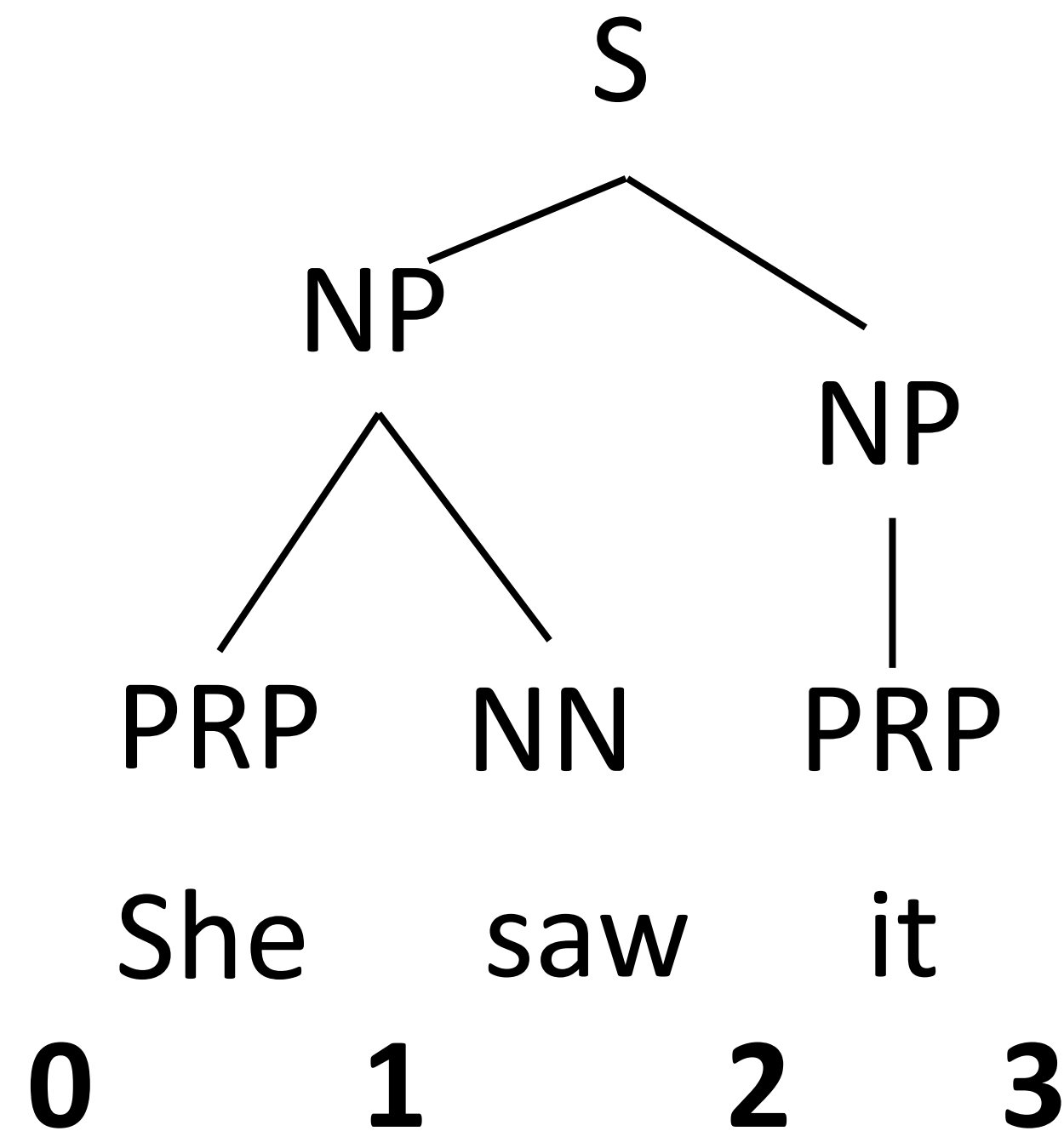


Compute unary closure: if there is a rule chain $X \rightarrow Y_1, Y_1 \rightarrow Y_2, \dots, Y_k \rightarrow Y$, add $q(X \rightarrow Y) = q(X \rightarrow Y_1) \times \dots \times q(Y_k \rightarrow Y)$

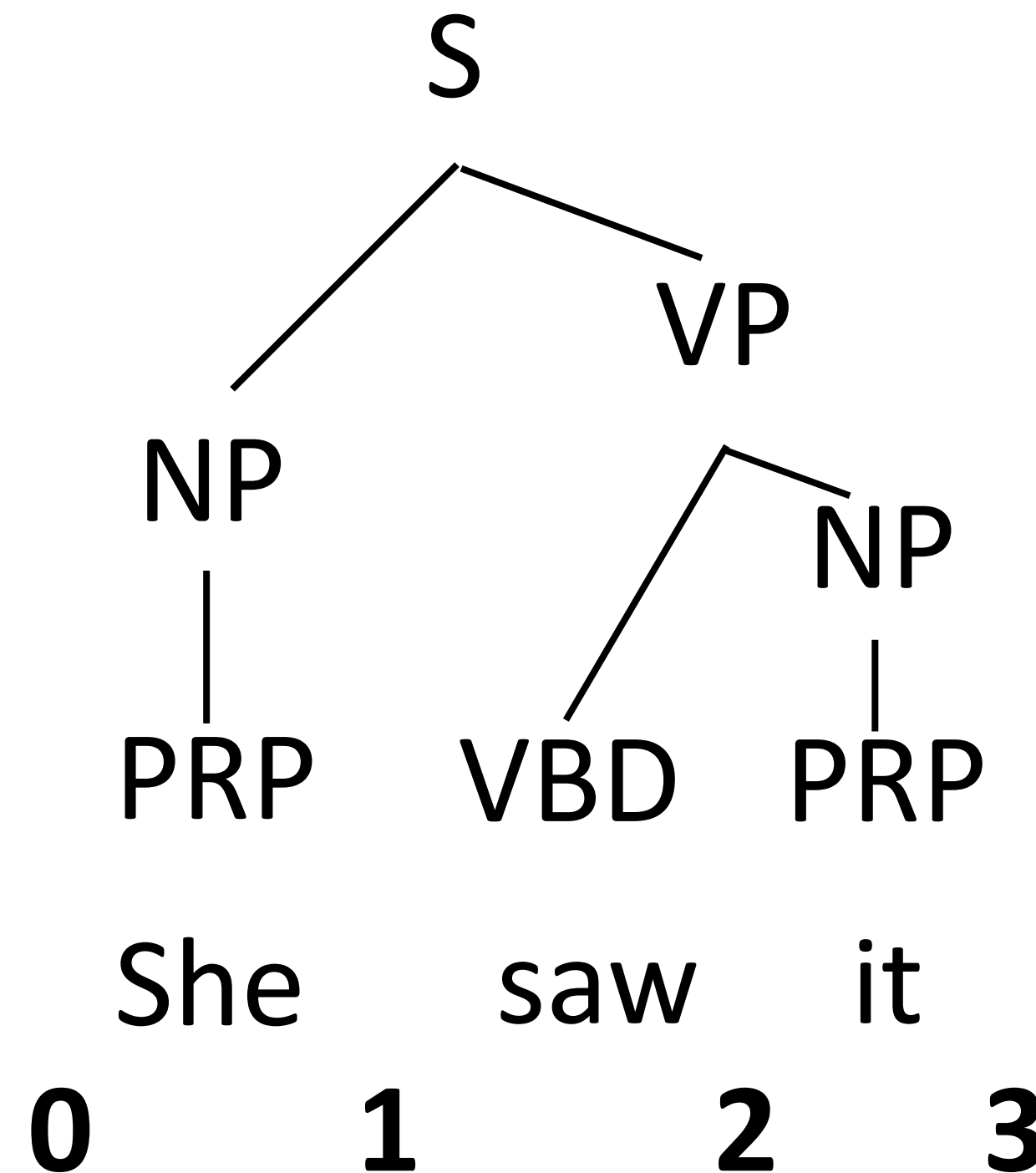
- In CKY: Update unary rule once after the binary rules



Parser Evaluation



S(0,3),
NP(0,2),
NP(2,3),
~~PRP(0,1),~~
~~NN(1,2),~~
~~PRP(2,3)~~



S(0,3),
NP(0,1),
VP(1,3),
NP(2,3),
~~PRP(0,1),~~
~~VBD(1,2),~~
~~PRP(2,3)~~

- Precision: number of correct brackets / num pred brackets = 2/3
- Recall: number of correct brackets / num of gold brackets = 2/4
- F1: harmonic mean of precision and recall = $(1/2 * ((2/4)^{-1} + (2/3)^{-1}))^{-1}$
= 0.57



Results

- ▶ Standard dataset for English: Penn Treebank (Marcus et al., 1993)
 - ▶ Evaluation: F1 over labeled constituents of the sentence
- ▶ Vanilla PCFG: ~75 F1
- ▶ Best PCFGs for English: ~90 F1
- ▶ SOTA (discriminative models): 95 F1
- ▶ Other languages: results vary widely depending on annotation + complexity of the grammar

Klein and Manning (2003)



Lexicalized Parsers

- ▶ Annotate each grammar symbol with its “head word” of the phrase

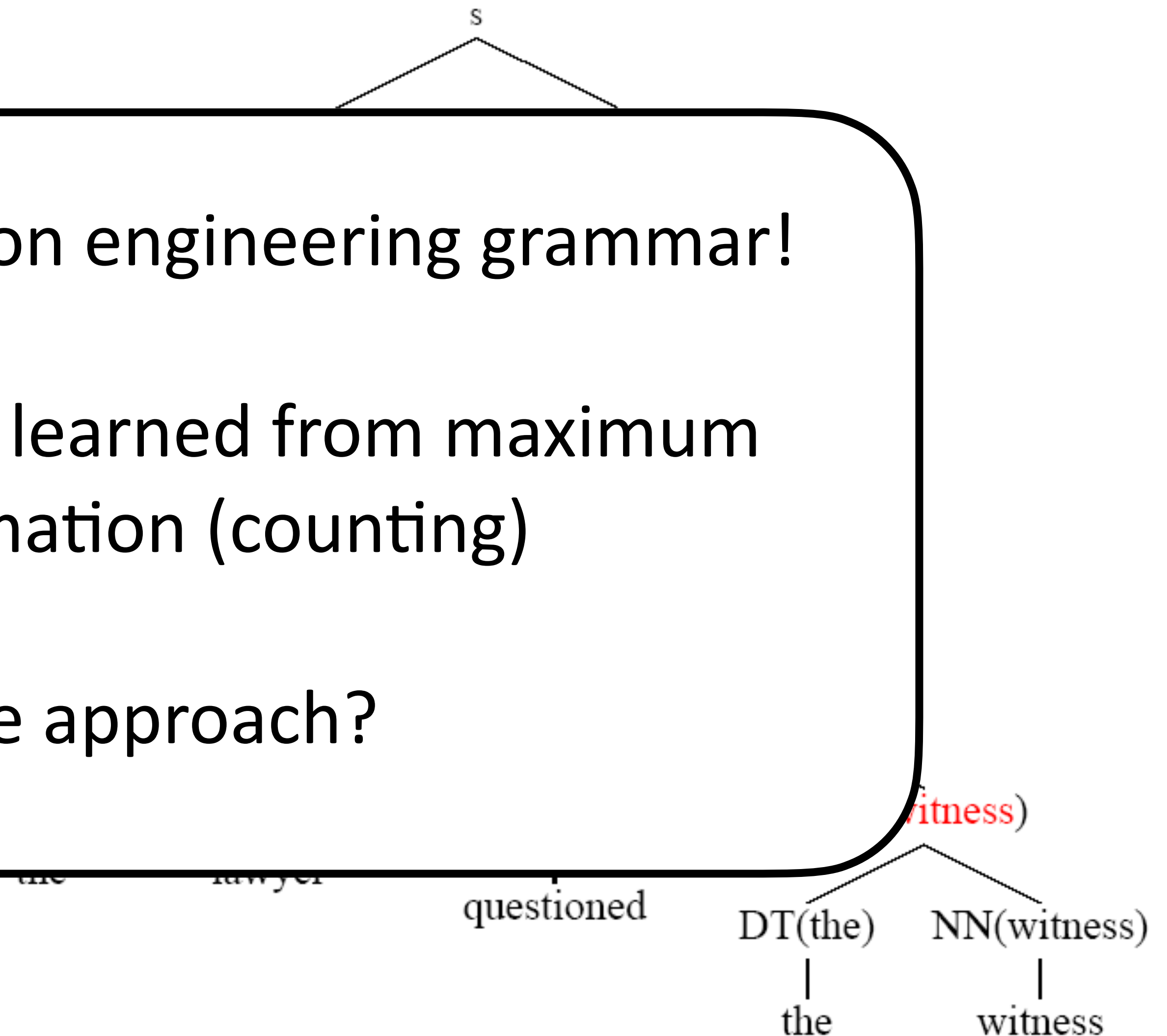
- ▶ Rules for the last word in the phrase are prepositional

- ▶ Collins and Manning achieved ~89 F1 with these

All these work focuses on engineering grammar!

Constituency parser is learned from maximum likelihood estimation (counting)

Alternative approach?

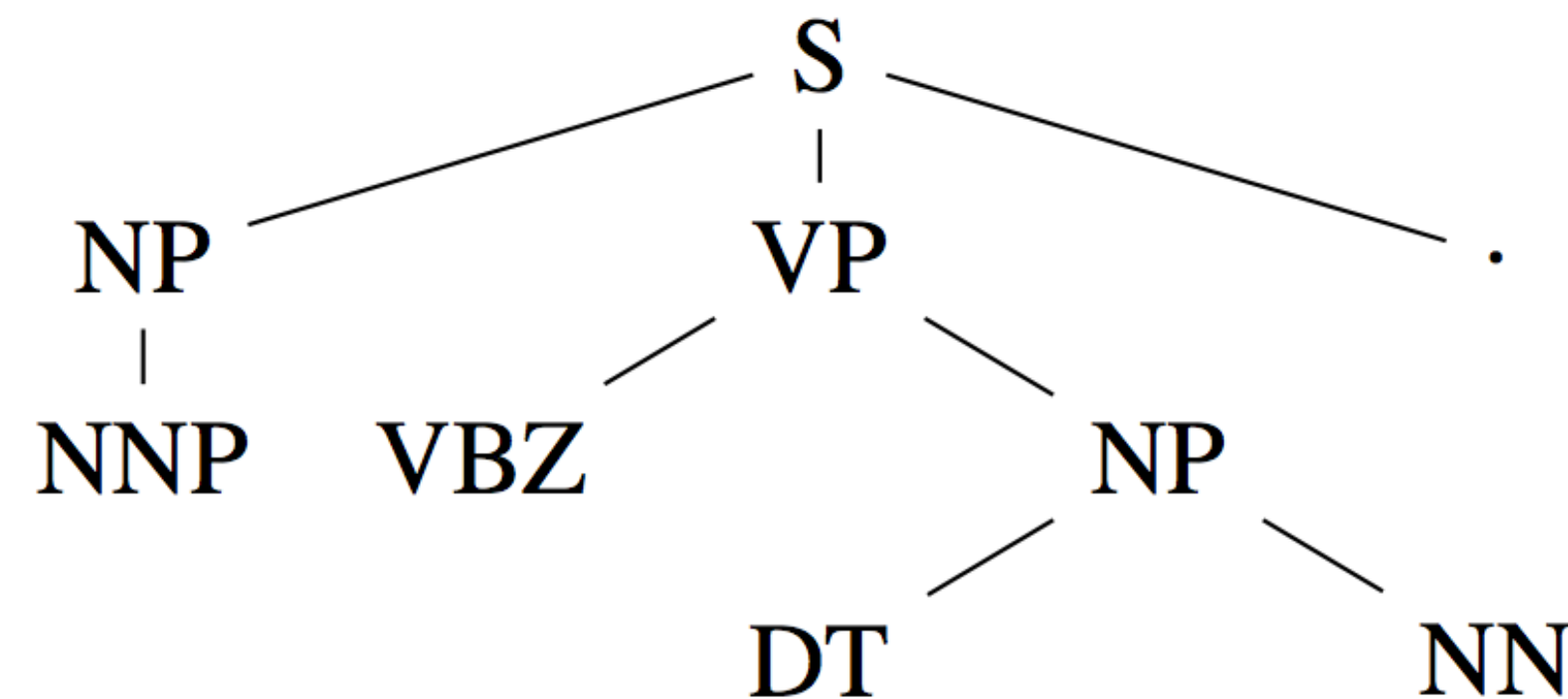




“Grammar as Foreign Language” (deep learning)

Vinyals et al., 2015

John has a dog →



John has a dog →

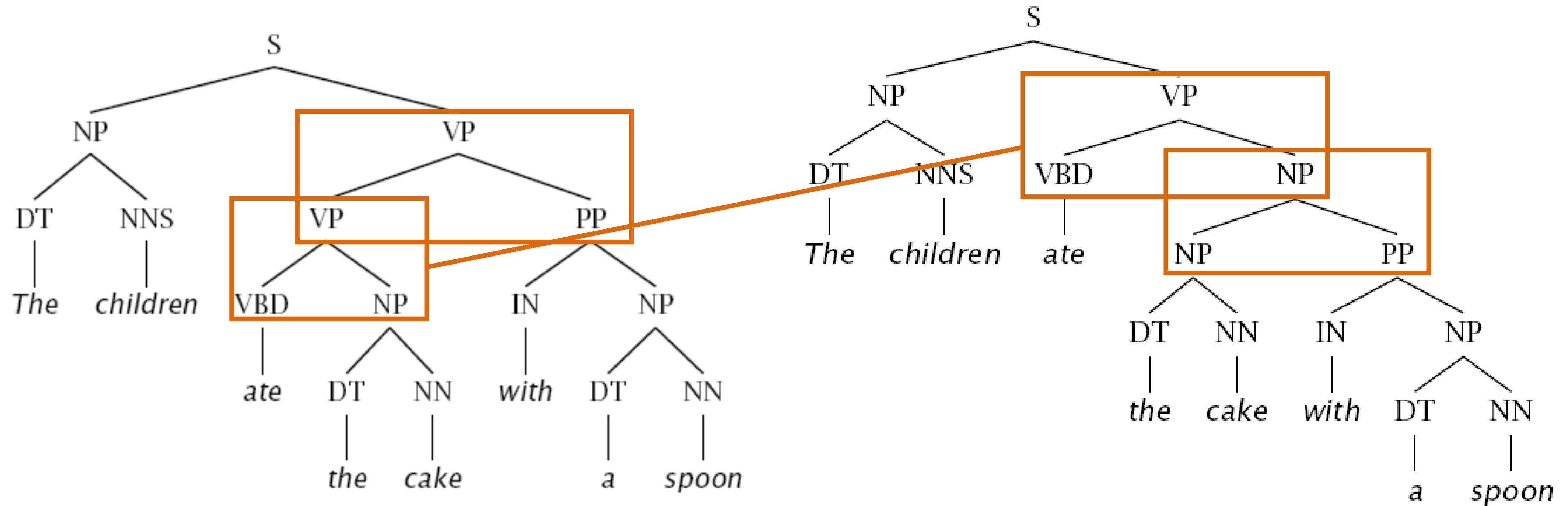
$(S (NP NNP)_{NP} (VP VBZ (NP DT NN)_{NP})_{VP} .)_{S}$

- ▶ Linearize parse trees into a sequence
- ▶ Then parsing becomes similar to machine translation, takes a sentence as an input sequence and output a parse tree sequence
- ▶ Data augmentation tricks, gets up to 92 F1



Dependency vs. Constituency: PP Attachment

- Constituency: several rule productions need to change





Dependency vs. Constituency: PP Attachment

- ▶ Dependency: one word (with) assigned a different parent

the children ate the cake with a spoon

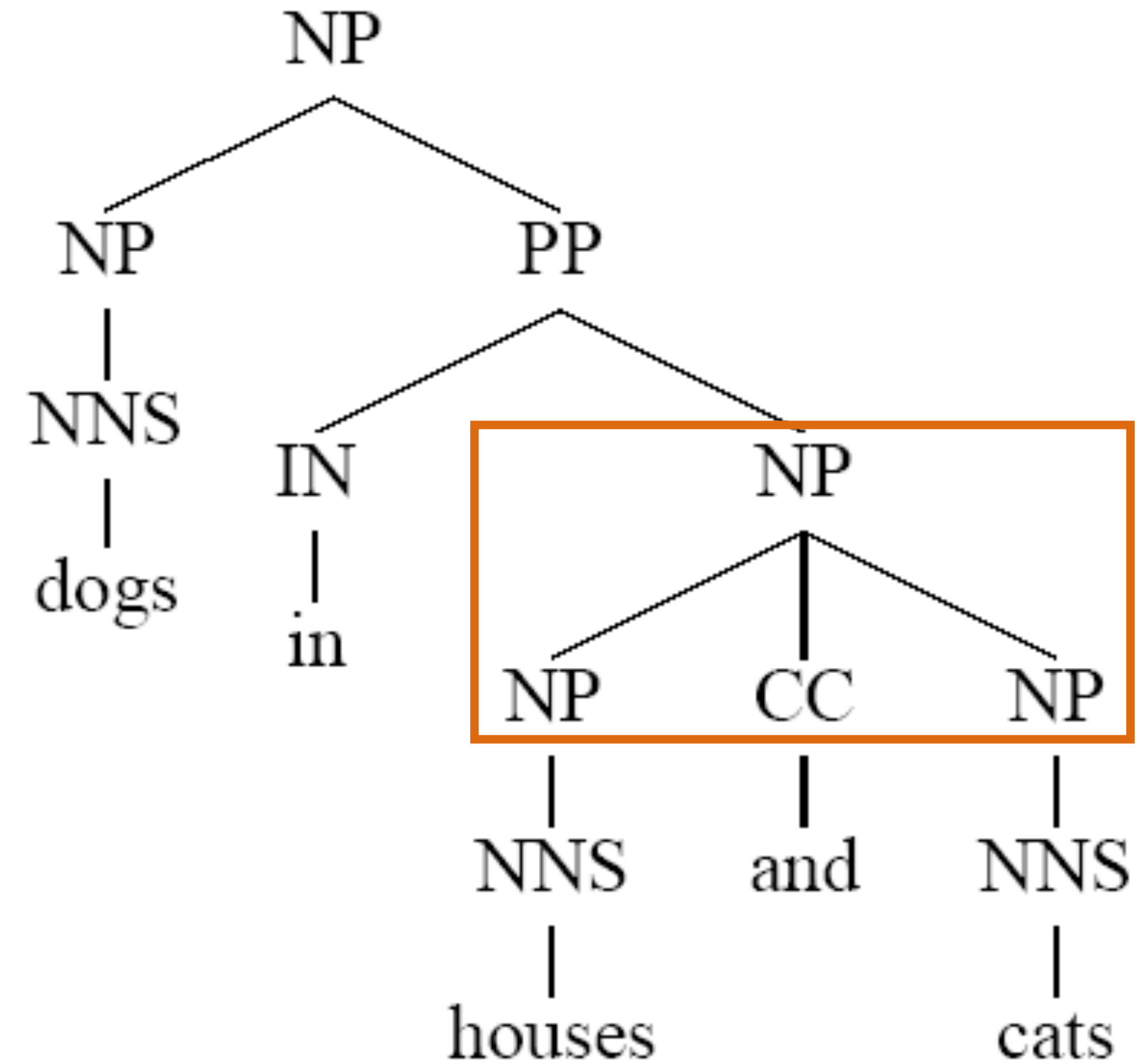
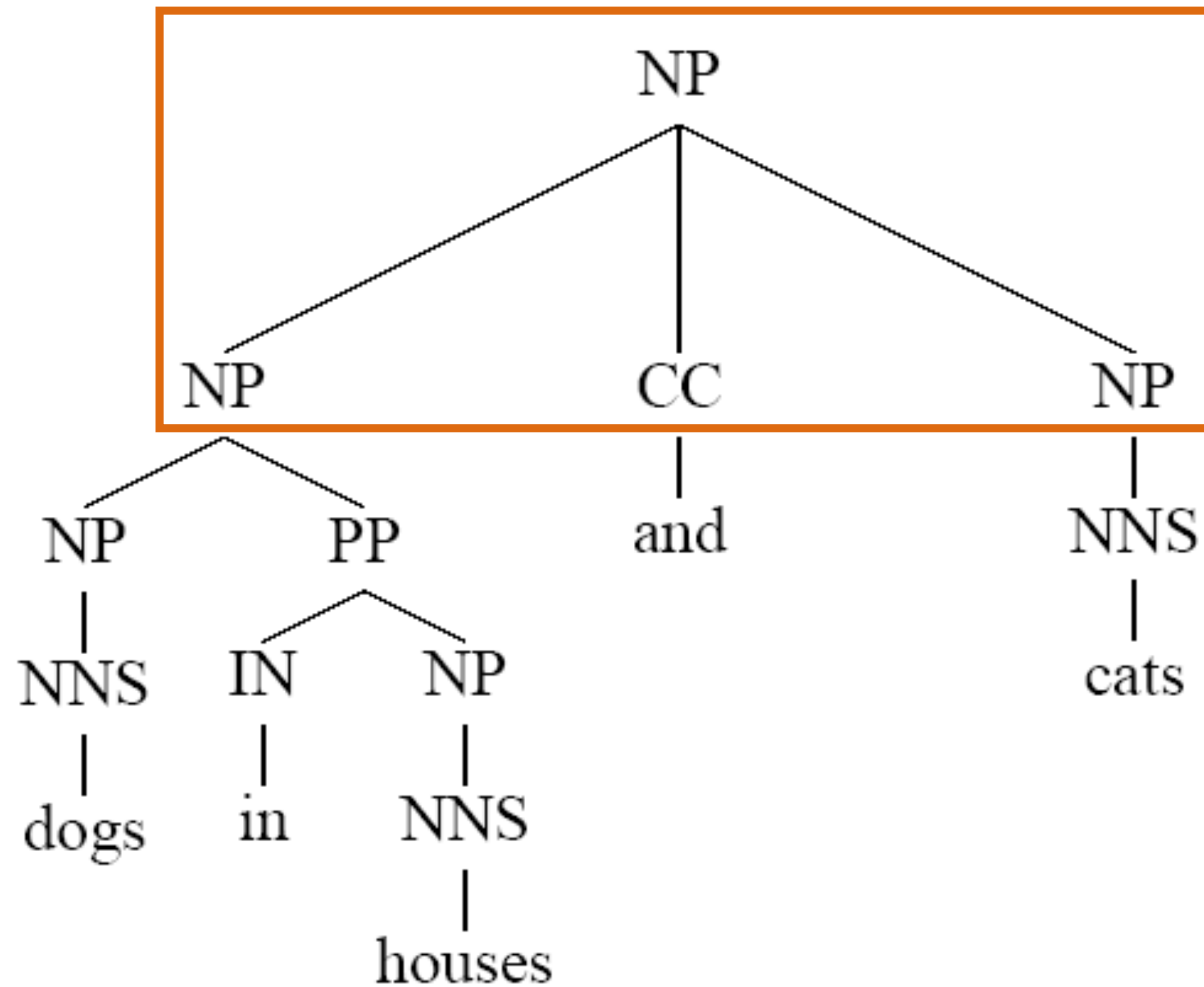
The diagram illustrates dependency arcs for the sentence "the children ate the cake with a spoon". Two orange curved arrows represent dependencies: one from "the" to "ate" and another from "with" to "ate".

- ▶ More predicate-argument focused view of syntax
- ▶ “What’s the main verb of the sentence? What is its subject and object?”
— easier to answer under dependency parsing



Dependency vs. Constituency: Coordination


- Constituency: ternary rule NP → NP CC NP





Dependency vs. Constituency: Coordination

- ▶ Dependency: first item is the head

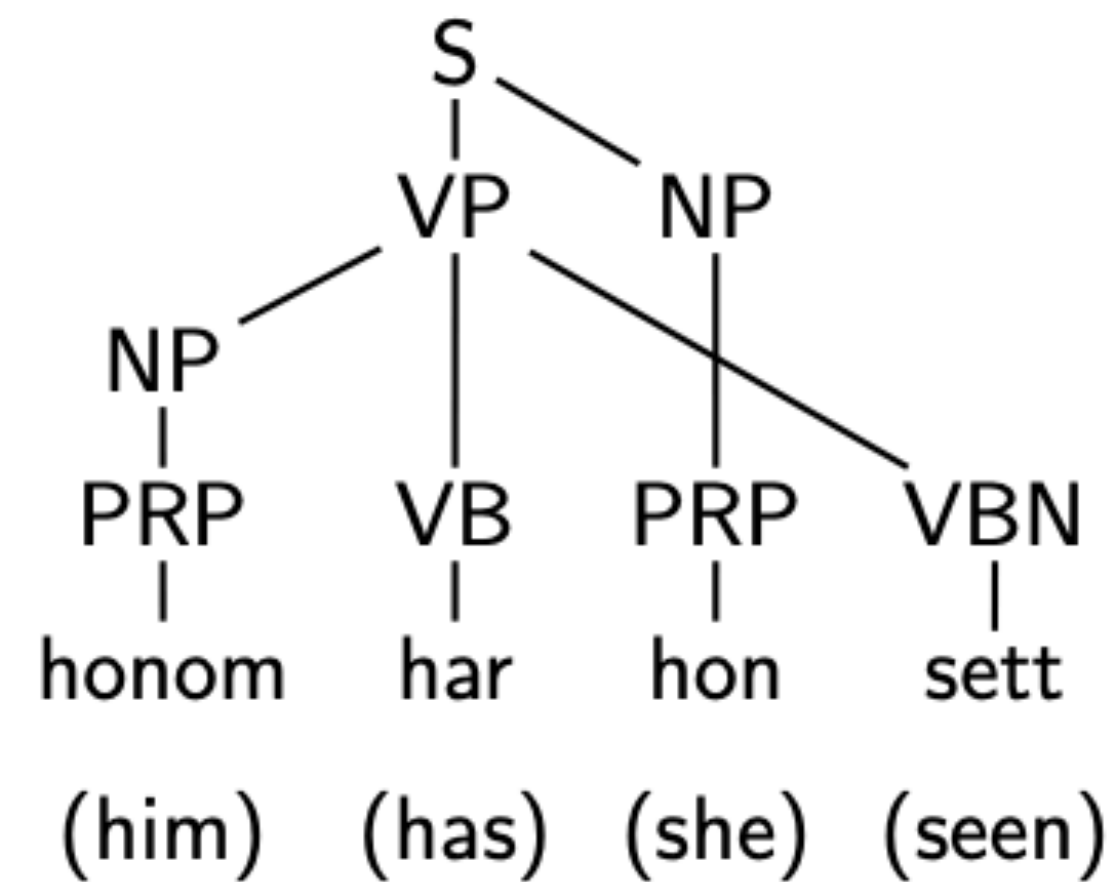
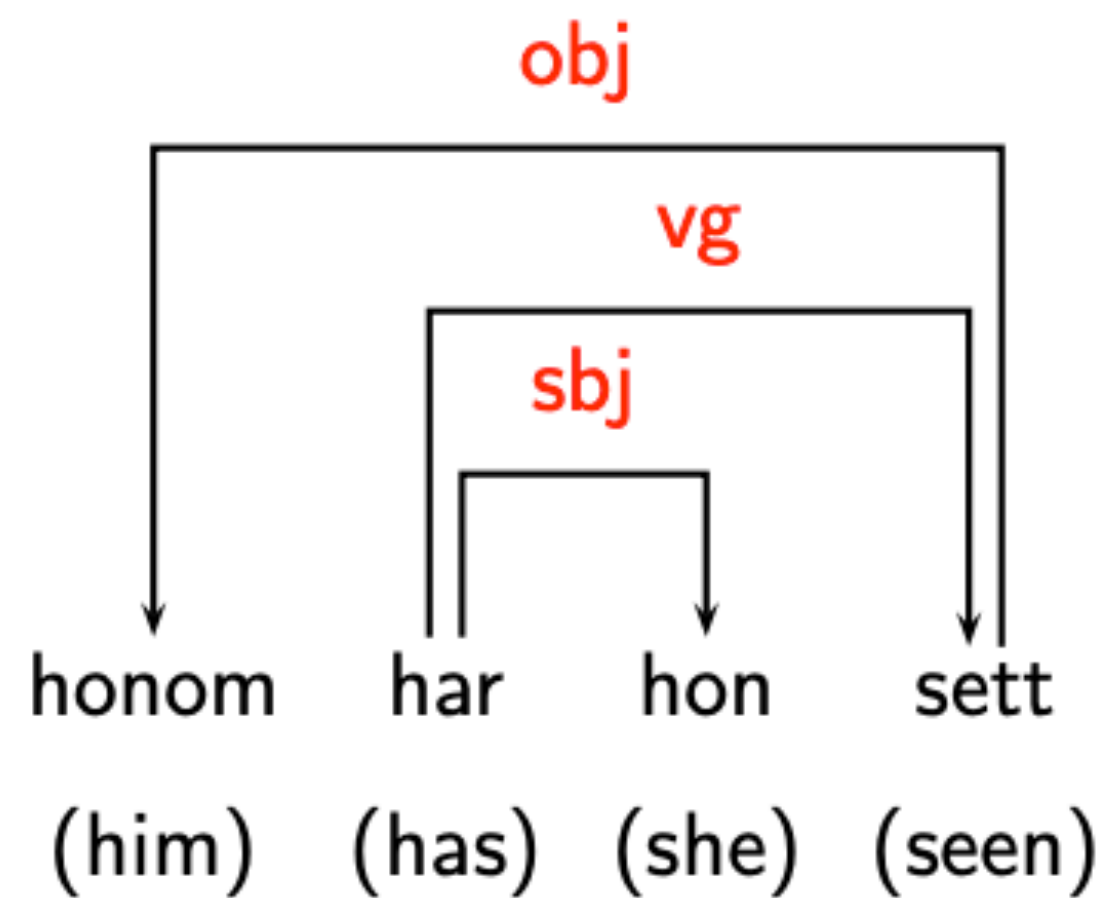
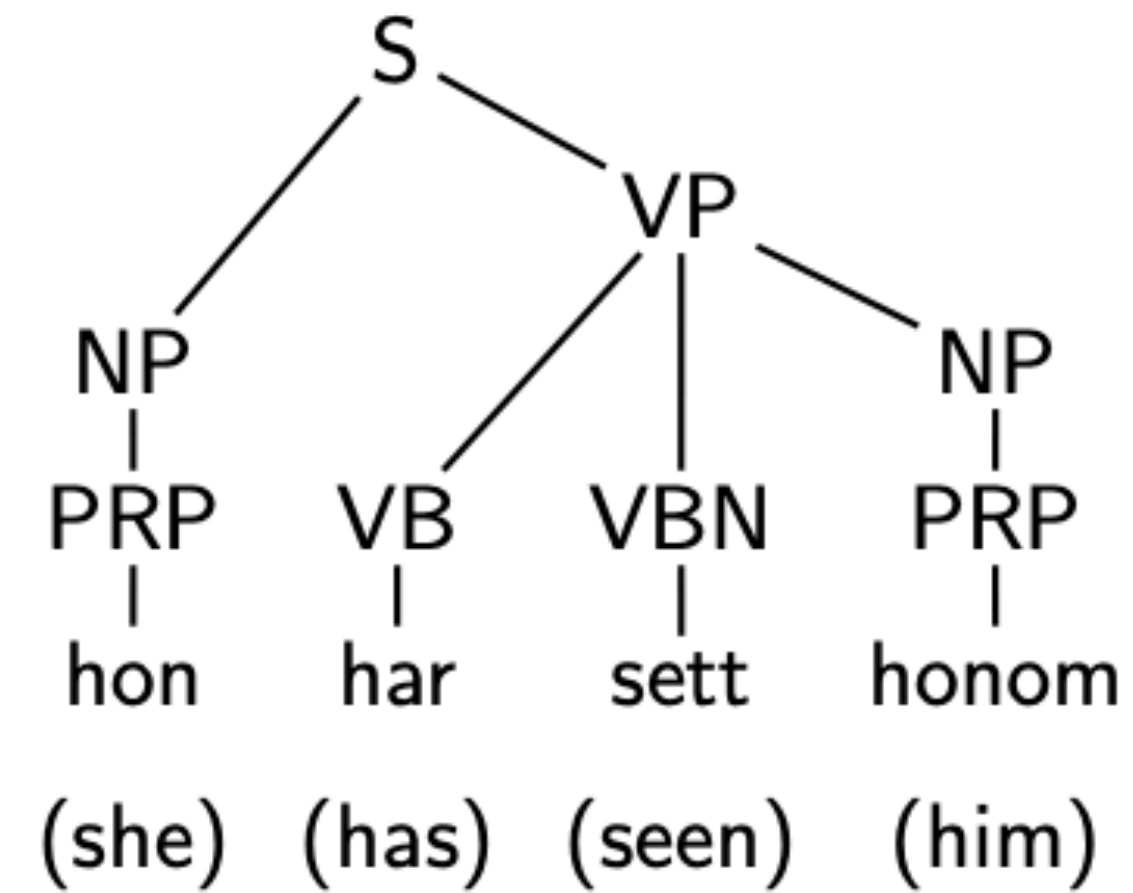
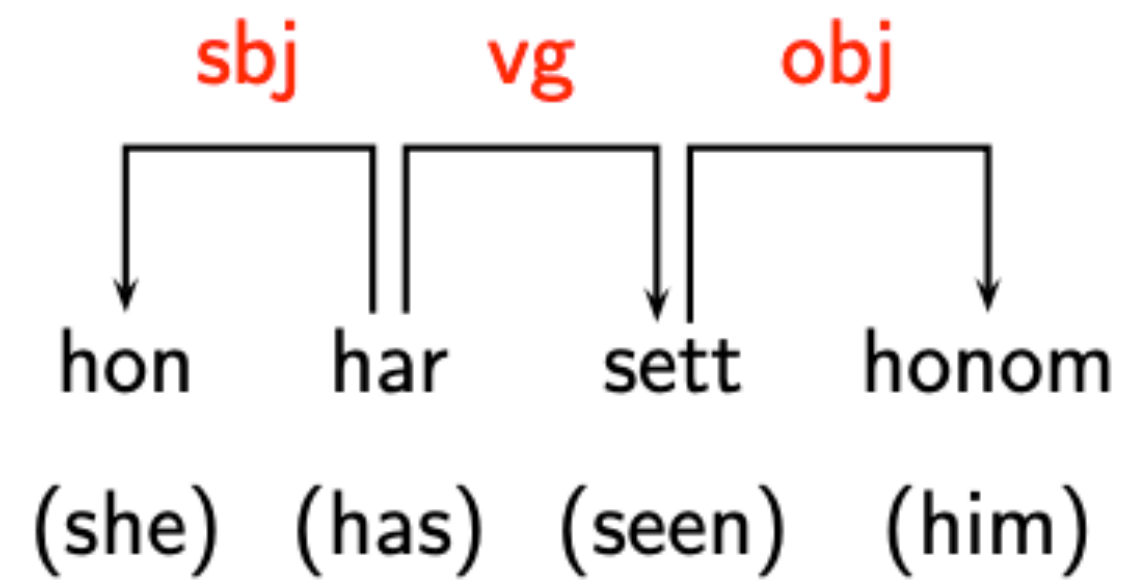

dogs in houses **and** cats
[dogs in houses] and cats


dogs in **houses and cats**
dogs in [houses and cats]

- ▶ Coordination is decomposed across a few arcs as opposed to being a single rule production as in constituency
- ▶ Can also choose *and* to be the head
- ▶ In both cases, headword doesn't really represent the phrase — constituency representation makes more sense



Advantages of dependency tree





Dependency vs. Constituency

- ▶ Dependency is often more useful in practice (models predicate argument structure)
- ▶ Dependency parsers are easier to build: no “grammar engineering”, no unaries, easier to get structured discriminative models working well
- ▶ Dependency parsers are usually faster (we will learn next class!)
- ▶ Dependencies are more universal cross-lingually



Dependency vs. Constituency

- ▶ Constituency includes non-terminals, and their edges are not typed.
- ▶ Dependency types encode “grammatical roles”.
- ▶ Can we get transform dependency parse into constituency parse? How about the other way around?
 - ▶ Mostly yes, with some caveats
 - ▶ Dependency parse can capture non-projective dependencies, while constituency parse cannot
 - ▶ Mapping from constituency to dependency edges and head is heuristic, somewhat lossy.