SFU NatLangLab

CMPT 413/713: Natural Language Processing

Constituency Parsing

Spring 2026
2026-03-16

Adapted from slides from Danqi Chen and Karthik Narasimhan
(with some content from Anoop Sarkar, David Bamman, Chris Manning, Mike Collins, and Graham Neubig)

Parsing

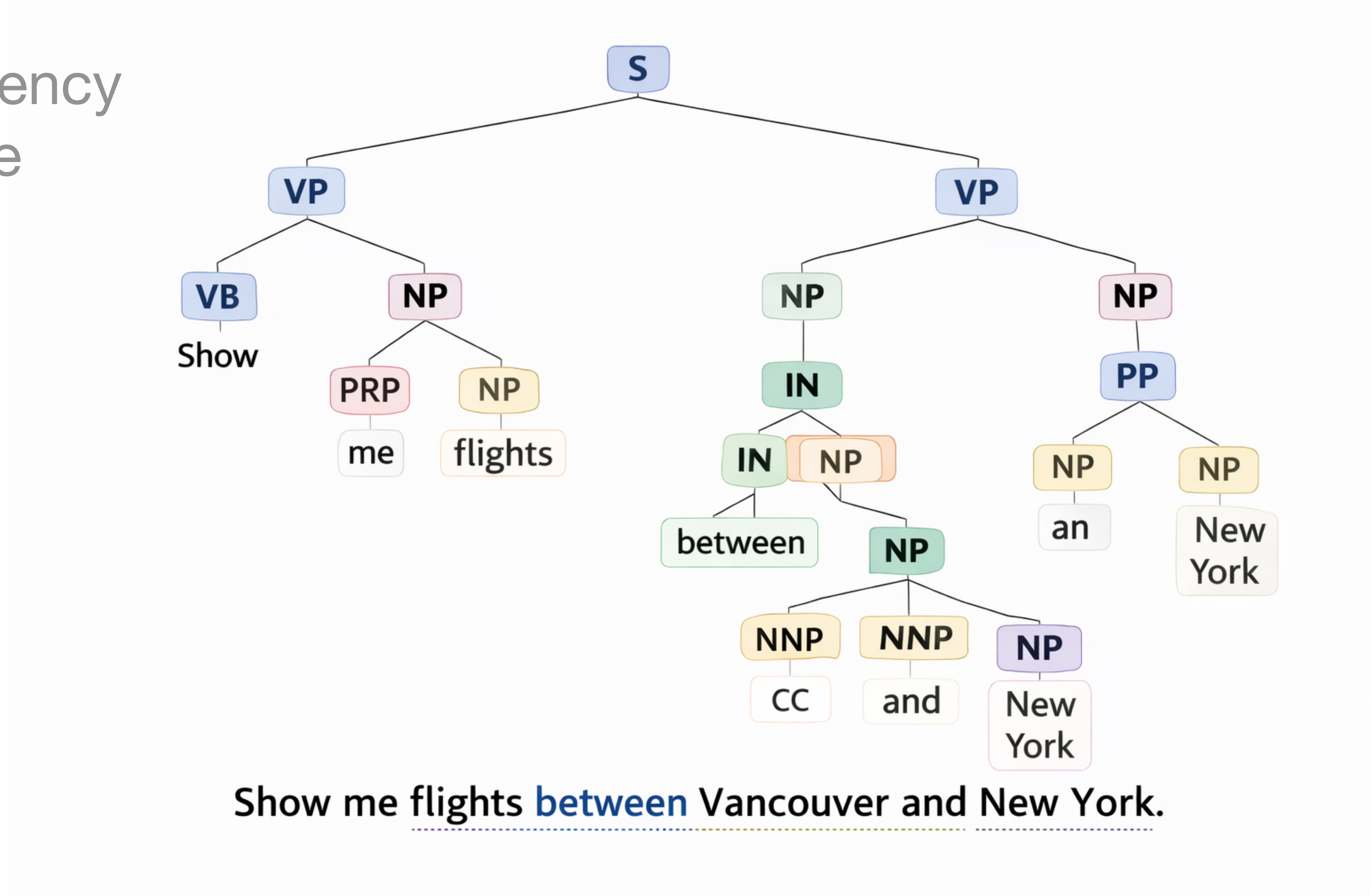
- Sentence to structured representation
- Useful to extract information for various applications

Show me flights between Vancouver and New York.

ChatGPT (Mar 2026)

Syntactic parsing

Constituency parse



- Understand the syntactic structure of a sentence
- Useful for analyzing sentences

Semantic parsing

SQL

```
SELECT f.*
FROM flights f
JOIN airports a1 ON f.origin_airport = a1.airport_code
JOIN airports a2 ON f.destination_airport = a2.airport_code
WHERE a1.city = 'Vancouver'
  AND a2.city = 'New York';
```

- Extract “semantics” (meaning) of the sentence
- Machine interpretable form (code, frames)
- Potentially executable

Parsing to extract information

- **Frame** - knowledge structure representing user intentions
  - Contains a set of **slots**, to be filled with information of a given **type.**
  - Each associated with a **question** to the user

Slot	Type	Question
ORIGIN	city	What city are you leaving from?
DEST	city	Where are you going?
DEP DATE	date	What day would you like to leave?
DEP TIME	time	What time would you like to leave?
AIRLINE	Airline	What is your preferred airline?

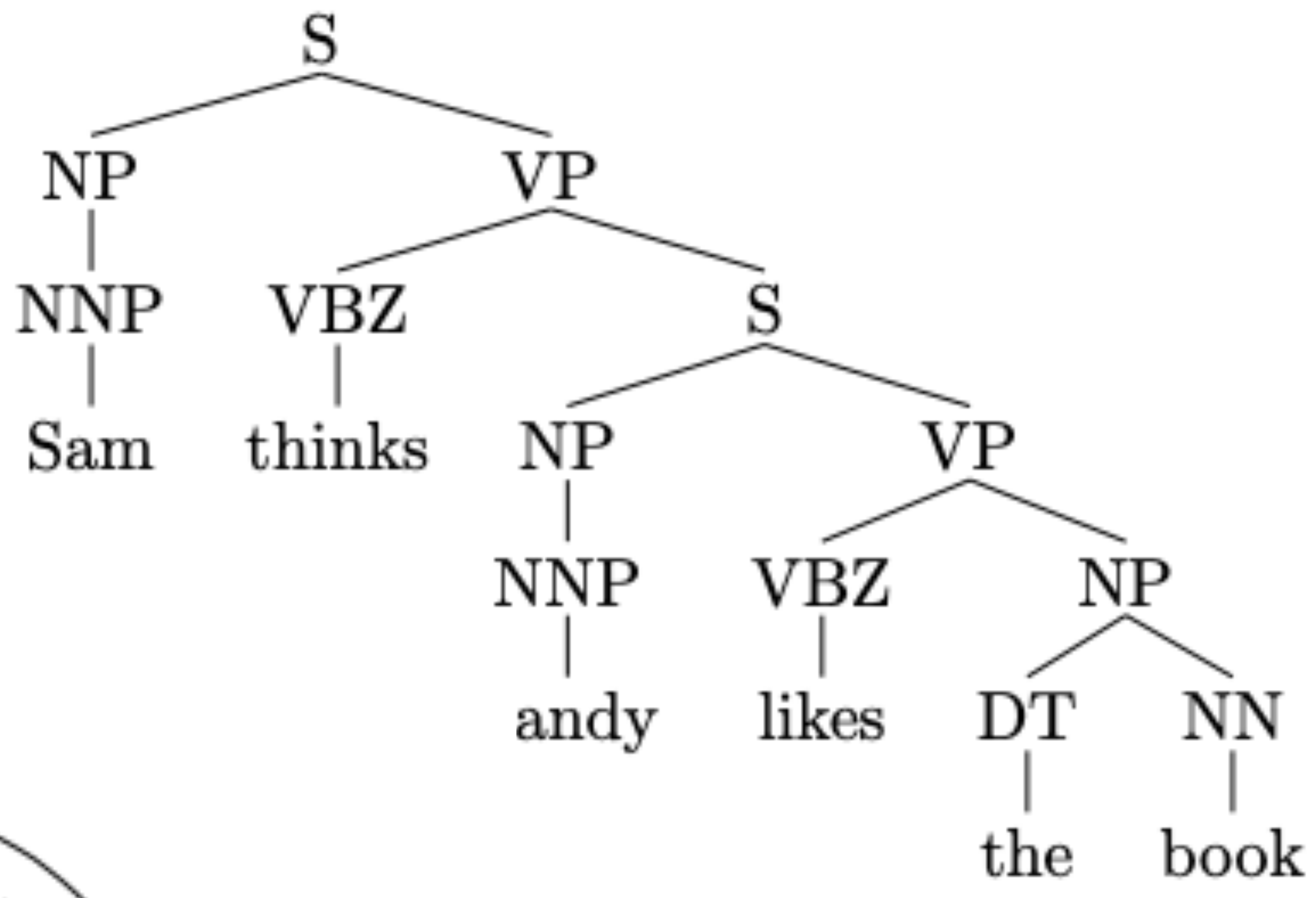
Syntactic parsing

- Constituency structure vs dependency structure
- Context-free grammar (CFG)
- Probabilistic context-free grammar (PCFG)
- The CKY algorithm
- Evaluation
- Lexicalized PCFGs
- Neural methods for constituency parsing
- LLMs for constituency parsing

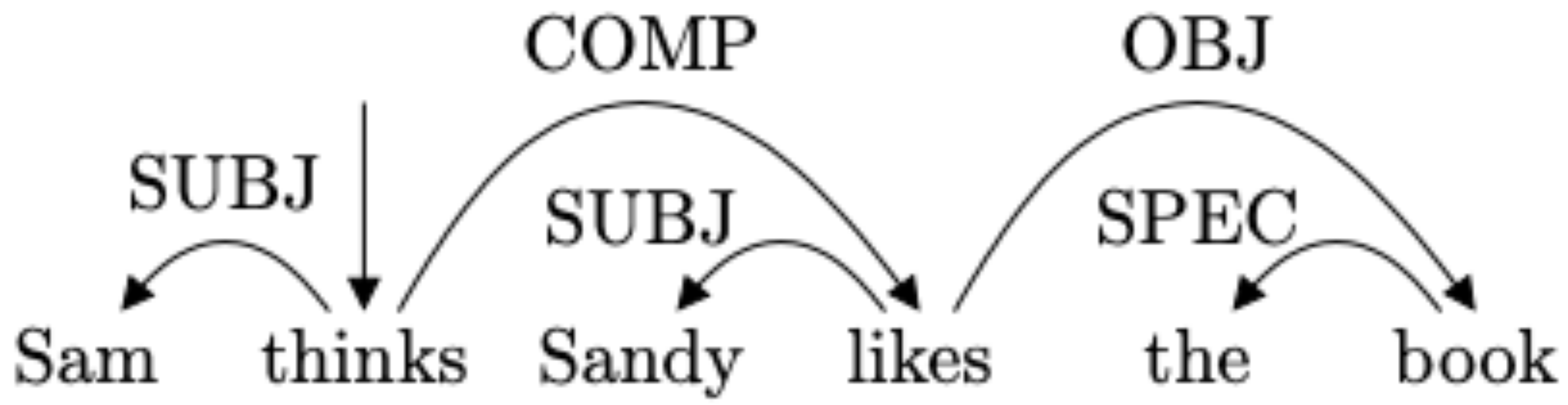
Syntactic structure: constituency and dependency

Two views of linguistic structure

- **Constituency**
  - = phrase structure grammar
  - = context-free grammars (CFGs)



- **Dependency**



Constituency structure

- **Phrase structure** organizes words into **nested constituents**
- Starting units: words are given a category: part-of-speech tags

  the, cuddly, cat, by, the, door

  DT, JJ, NN, IN, DT, NN

- Words combine into phrases with categories

  the cuddly cat, by, the door

  NP→DT JJ NN IN NP →DT NN

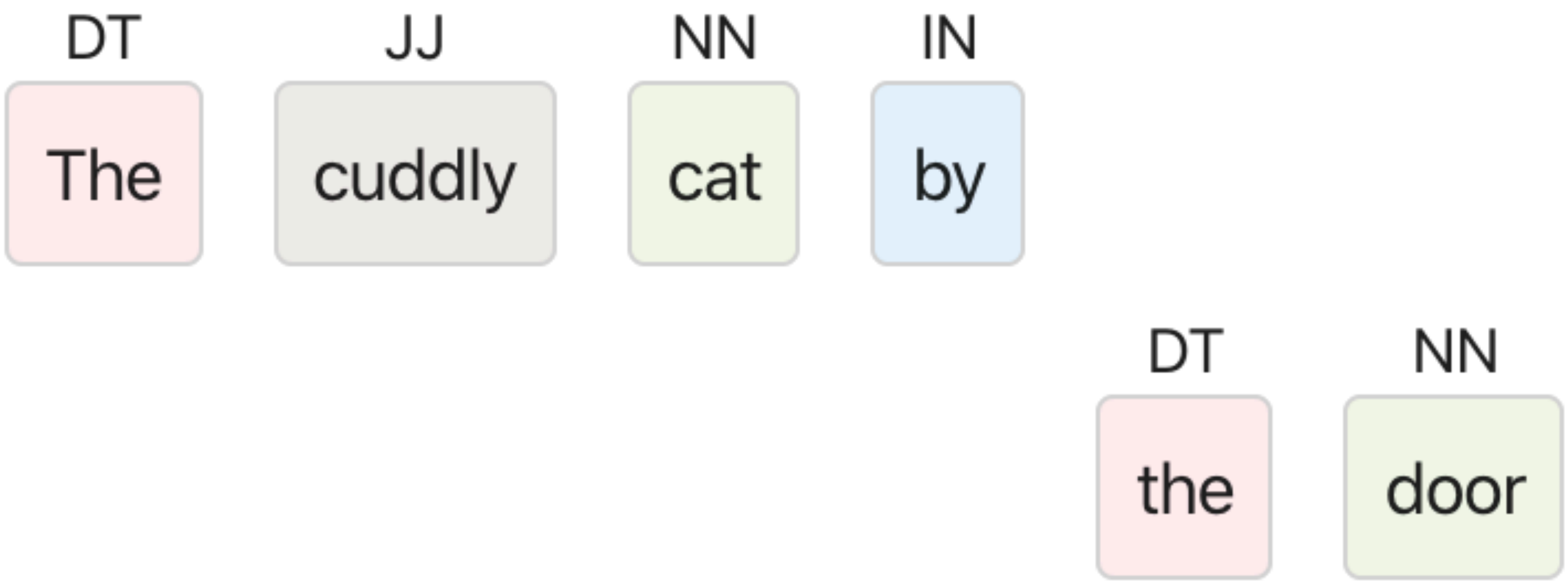
- Phrases can combine into bigger phrases recursively

  the cuddly cat, by the door

  NP PP→IN NP

  the cuddly cat by the door

  NP→ NP PP



Why do we need sentence structure?

- We need to understand sentence structure in order to be able to interpret language correctly

- Human communicate complex ideas by composing words together into bigger units

- We need to know what is connected to what

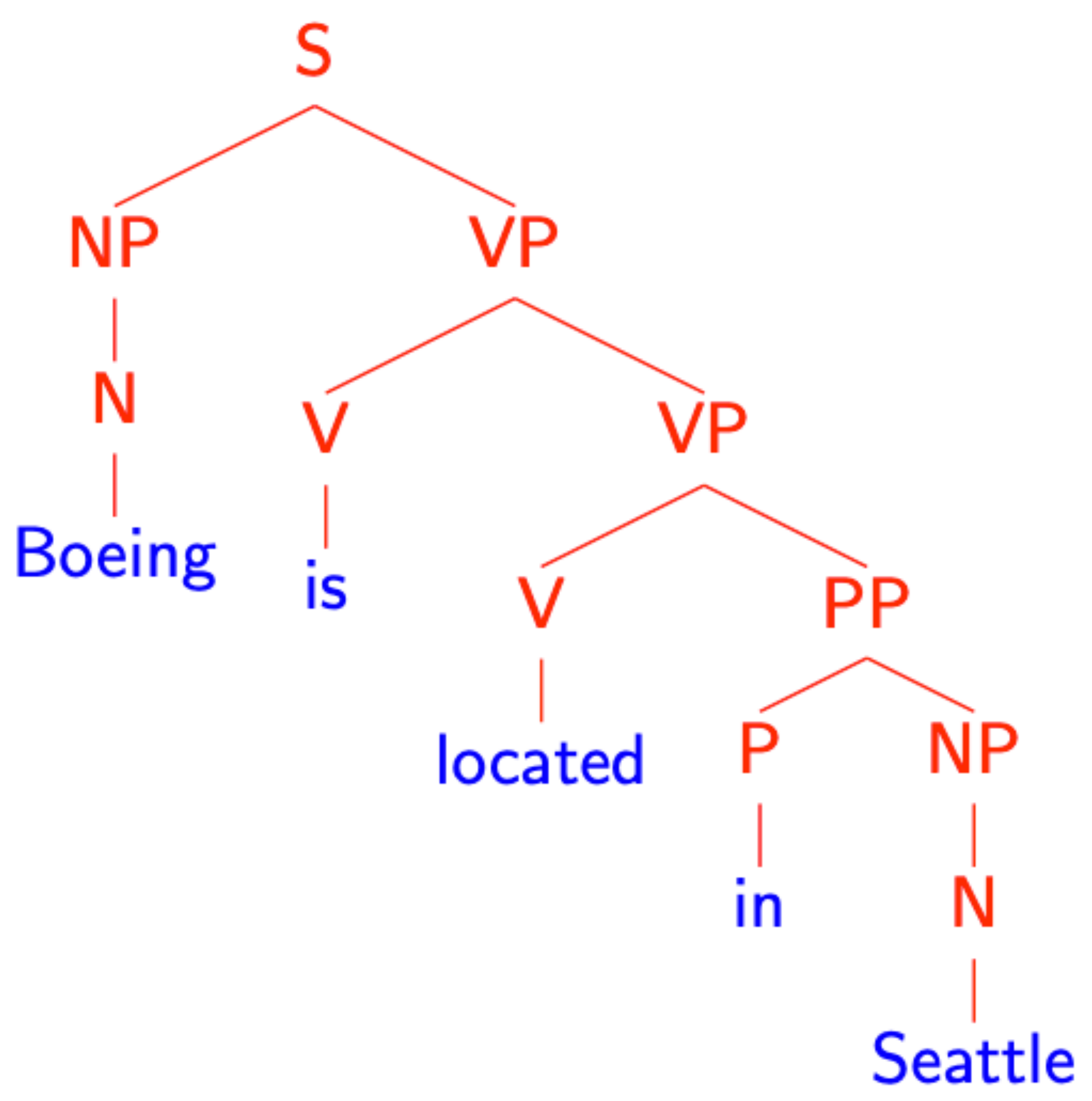
Syntactic parsing

- Syntactic parsing is the task of recognizing a sentence and assigning a structure to it.

Input:

Boeing is located in Seattle.

Output:

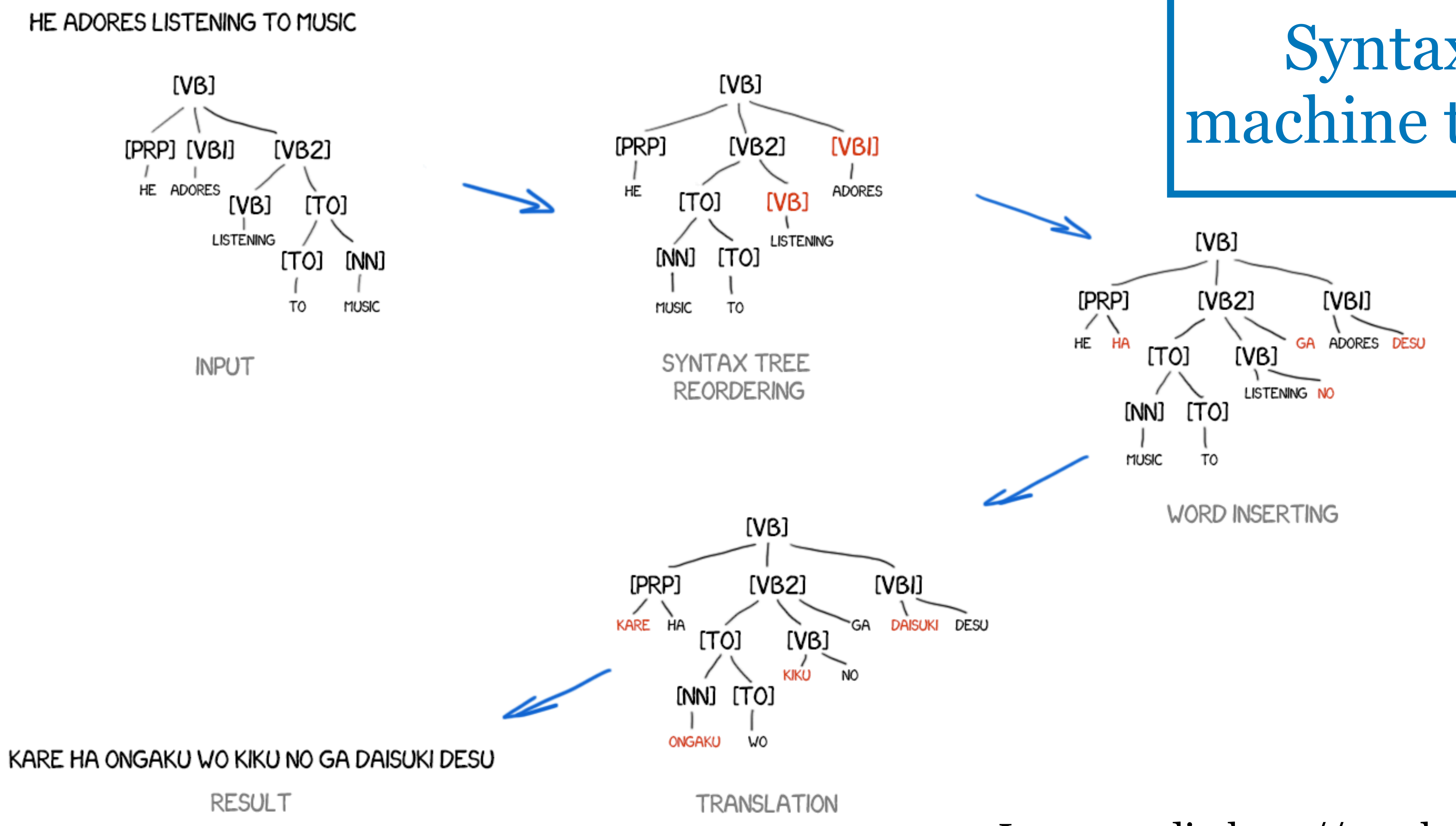


Syntactic parsing

- Used as intermediate representation for downstream applications

English word order: subject — verb — object

Japanese word order: subject — object — verb

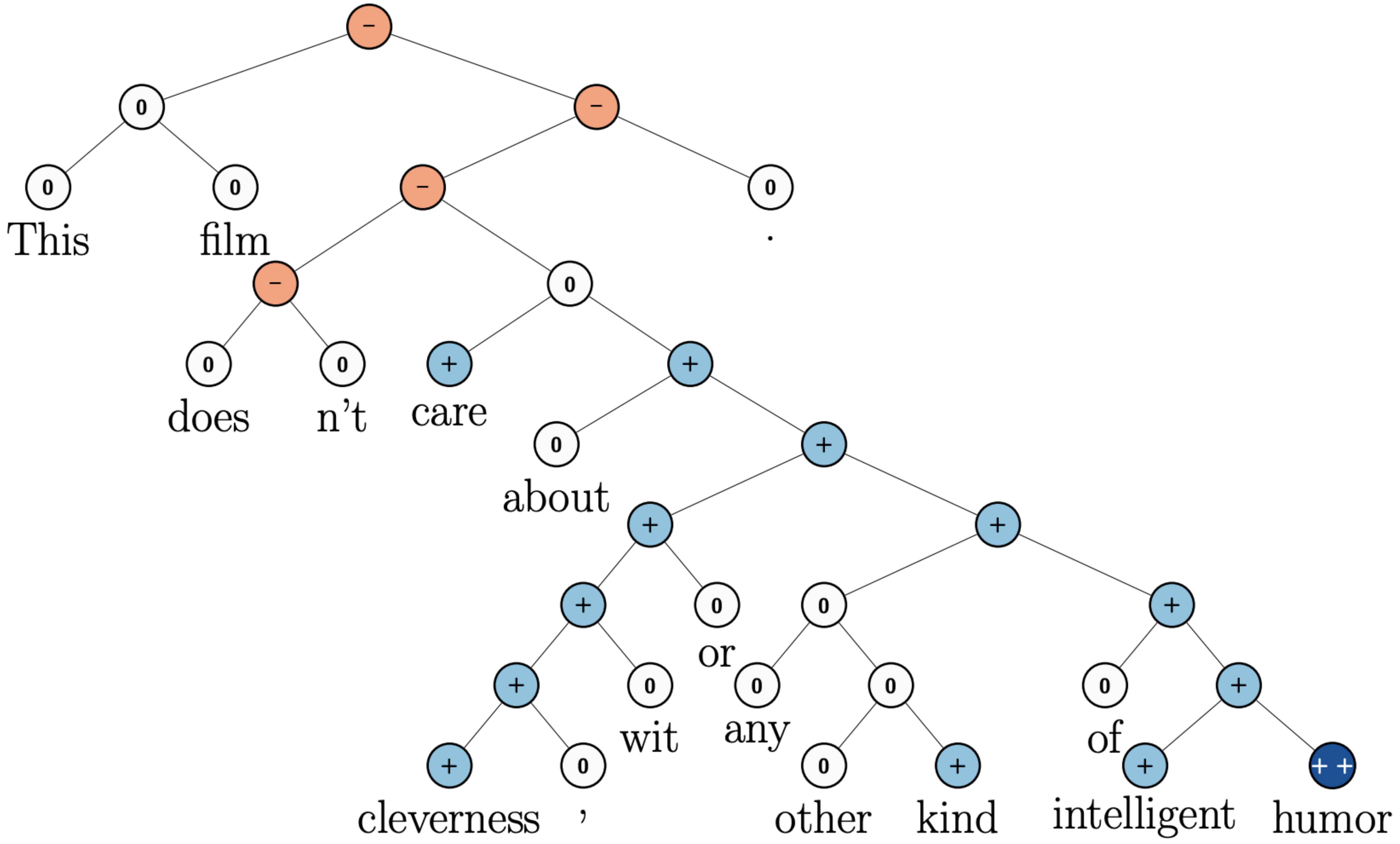


Syntax based machine translation

Image credit: http://vas3k.com/blog/machine_translation/

Beyond syntactic parsing

This file doesn’t care about cleverness, wit or any other kind of intelligent humor. Negative



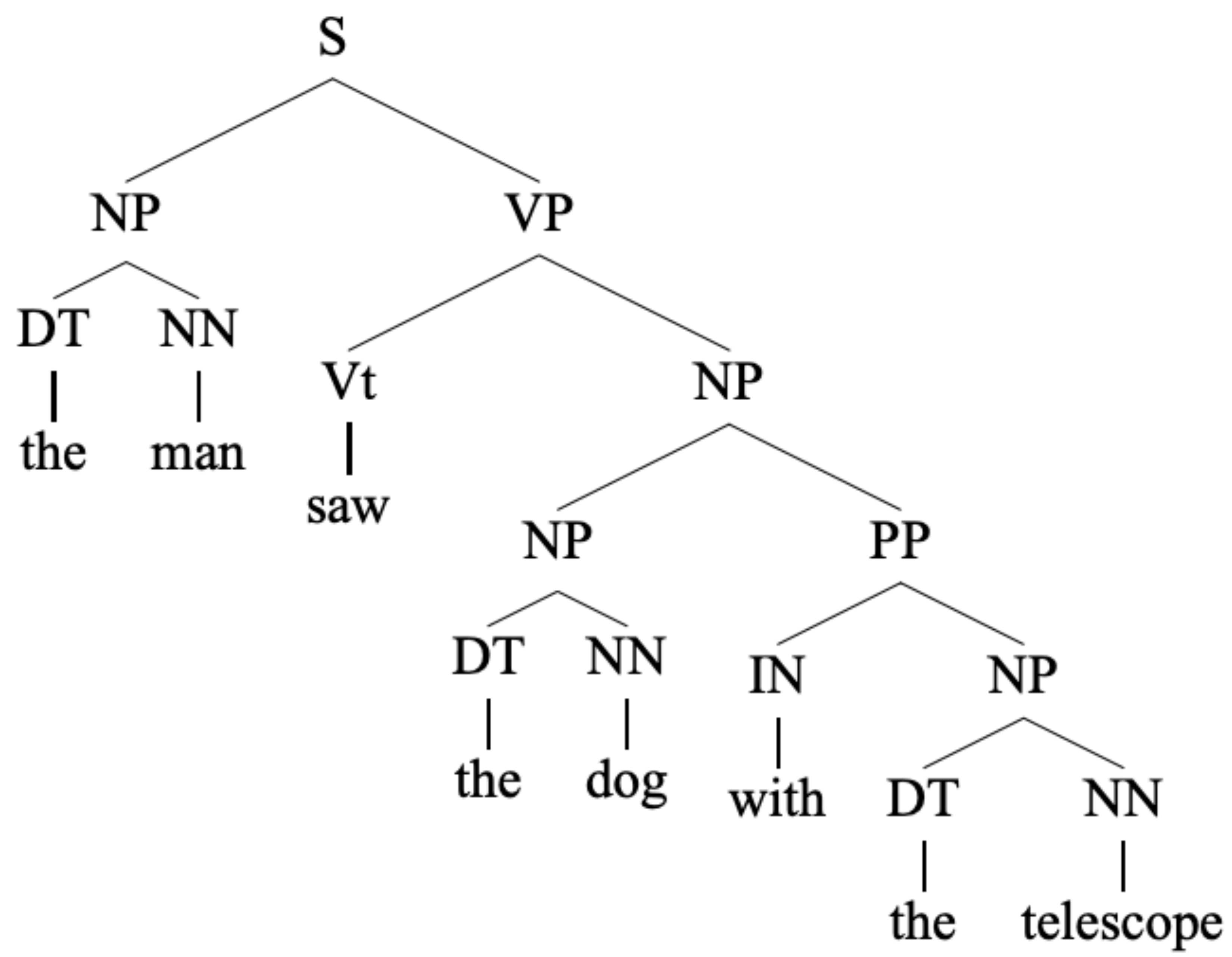
Nested Sentiment Analysis

Recursive deep models for semantic compositionality over a sentiment treebank
Socher et al, EMNLP 2013

Context-free grammars (CFG)

- Widely used formal system for modeling constituency structure in English and other natural languages

- A context free grammar $G = (N, \Sigma, R, S)$ where
  - N is a set of non-terminal symbols
  - Σ is a set of terminal symbols
  - R is a set of rules of the form $X \rightarrow Y_1 Y_2 \ldots Y_n$ for $n \geq 1, X \in N, Y_i \in (N \cup \Sigma)$
  - $S \in N$ is a distinguished start symbol



A Context-Free Grammar for English

N = {S, NP, VP, PP, DT, Vi, Vt, NN, IN}
S = S
Σ = {sleeps, saw, man, woman, telescope, the, with, in}

$R =$

Grammar

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

Lexicon

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

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

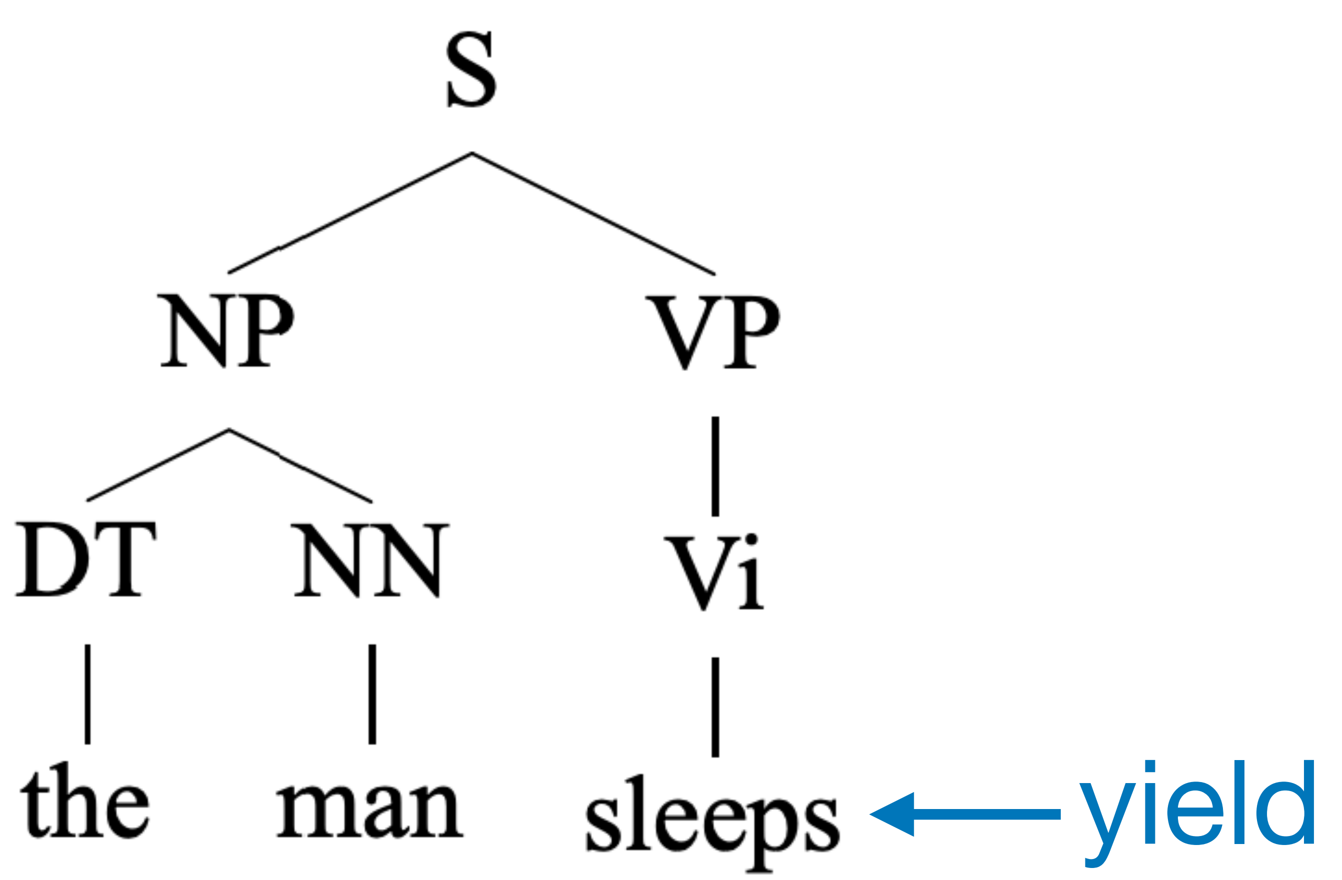
Derivations

- Given a CFG G, a derivation is sequence of rule-expansions starting from the start symbol to a string consisting of terminal symbols
- It can be expressed as a sequence of strings $s_1, s_2, \ldots, s_n$, where
  - $s_1 = S$ start symbol
  - $s_n \in \Sigma^*$ where Σ^* is all the possible strings made up of words from Σ
  - Each s_i for $i = 2, \ldots, n$ is derived from s_{i-1} by picking some non-terminal X in s_{i-1} and replacing it by some β where $X \to \beta \in R$

Left-most derivation:
pick left-most non-terminal

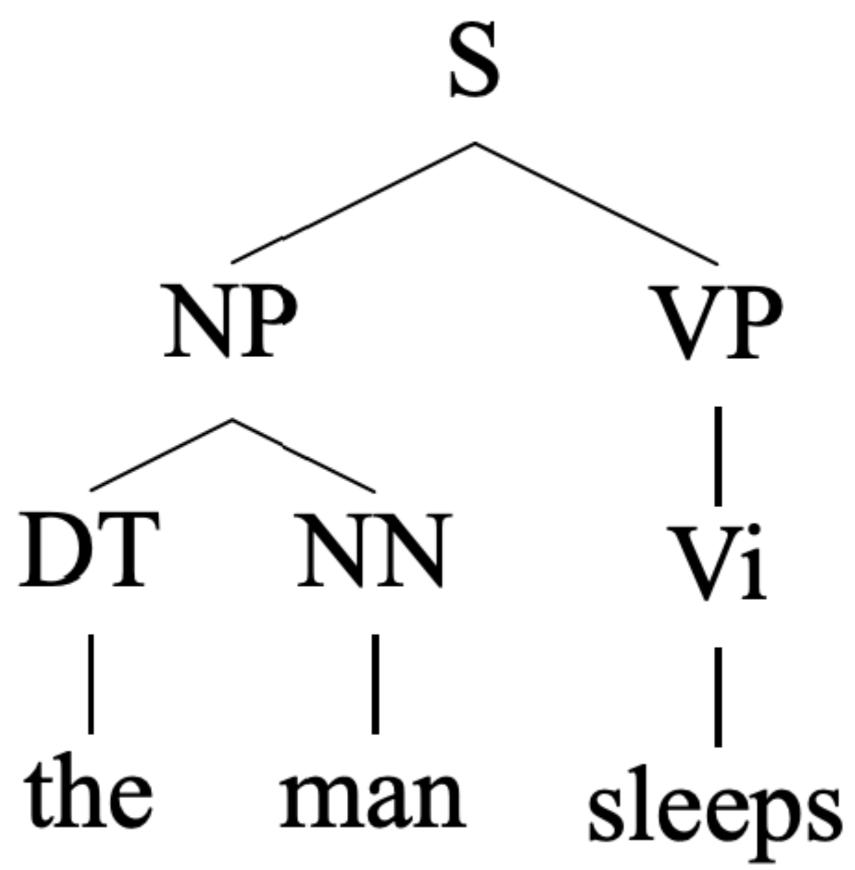
- s_n: yield of the derivation

A derivation can be represented as a parse tree!



(Left-most) Derivation

- $s_1 = \text{S}$
- $s_2 = \text{NP VP}$
- $s_3 = \text{DT NN VP}$
- $s_4 = \text{the NN VP}$
- $s_5 = \text{the man VP}$
- $s_6 = \text{the man Vi}$
- $s_7 = \text{the man sleeps}$



a parse tree

$R =$

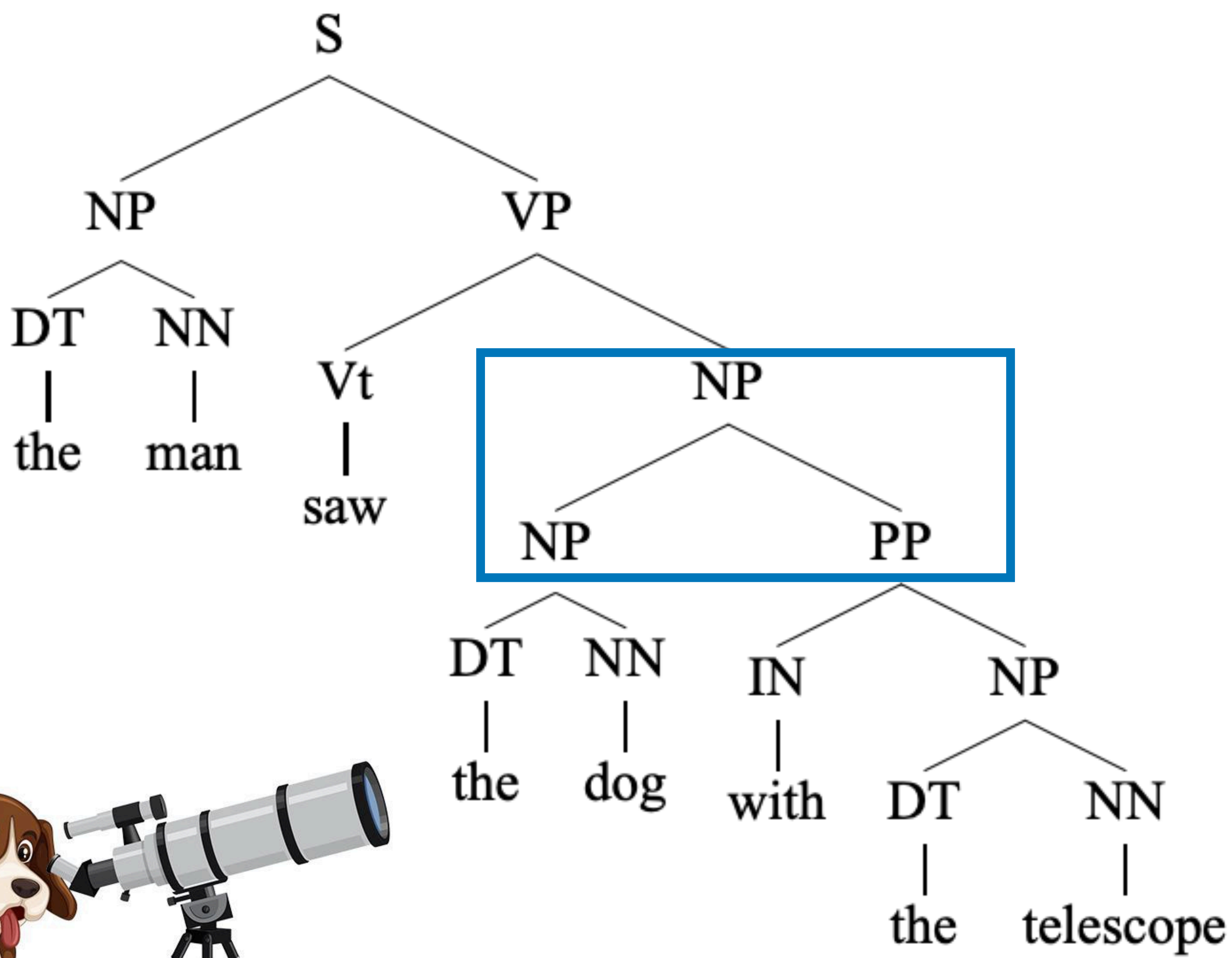
S	$\rightarrow$	NP	VP
VP	$\rightarrow$	Vi	
VP	$\rightarrow$	Vt	NP
VP	$\rightarrow$	VP	PP
NP	$\rightarrow$	DT	NN
NP	$\rightarrow$	NP	PP
PP	$\rightarrow$	IN	NP

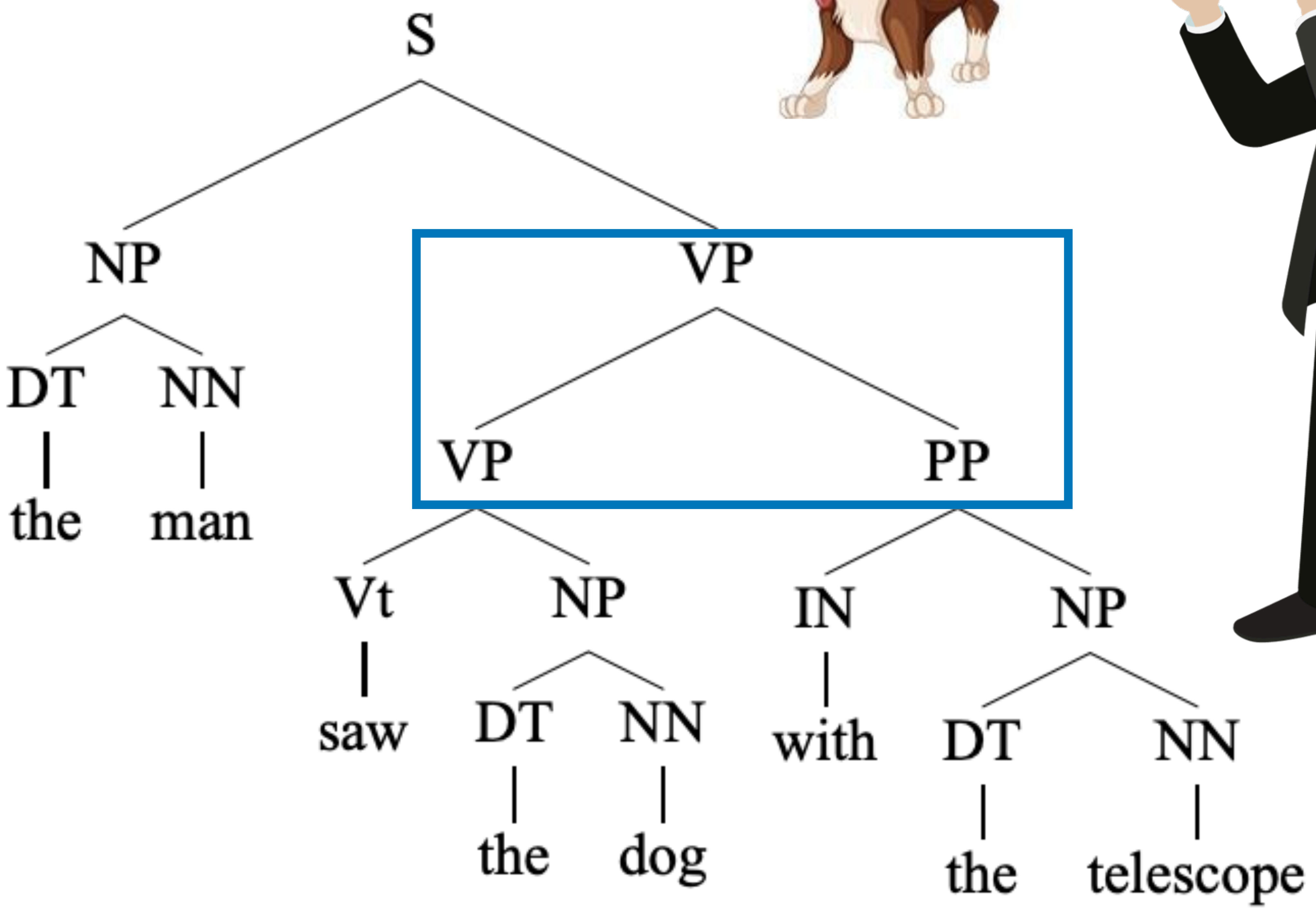
Vi	$\rightarrow$	sleeps
Vt	$\rightarrow$	saw
NN	$\rightarrow$	man
NN	$\rightarrow$	woman
NN	$\rightarrow$	telescope
NN	$\rightarrow$	dog
DT	$\rightarrow$	the
IN	$\rightarrow$	with
IN	$\rightarrow$	in

- A string $s \in \Sigma^*$ is in the language defined by the CFG if there is at least one derivation whose yield is s
- The set of possible derivations may be finite or infinite

Ambiguity

- Some strings may have more than one derivations (i.e. more than one parse tree!).





“Classical” NLP Parsing

- In fact, sentences can have a very large number of possible parses

  The board approved [its acquisition] [by Royal Trustco Ltd.] [of Toronto] [for $27 a share] [at its monthly meeting].

- How many parses for sentence of length n?

```
((natural language) (learning course))
(((natural language) learning) course)
((natural (language learning)) course)
(natural (language (learning course)))
(natural ((language learning) course))
```

$n : a^n$	number of parses
1	1
2	1
3	2
4	5
5	14
6	42
7	132
8	429
9	1430
10	4862
11	16796

“Classical” NLP Parsing

- In fact, sentences can have a very large number of possible parses

The board approved [its acquisition] [by Royal Trustco Ltd.] [of Toronto] [for $27 a share] [at its monthly meeting].

$n : a^n$	number of parses
1	1
2	1
3	2
4	5
5	14
6	42
7	132
8	429
9	1430
10	4862
11	16796

- The number of (binary) parses happen to follow the Catalan numbers

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

For a sentence of length *n*, can form constituents by placing parenthesis.

Catalan number: $C_n = \frac{1}{n+1}\binom{2n}{n}$

Number of parses = number of ways to parenthesize expression such that
- there are equal number of open/close parenthesis
- they are properly nested with open before close

unlabeled parses for a sentence of n words

See Church and Patil (CL Journal, 1982) or TAOCP VI pp 388-389 (Knuth, 1975)

"Classical" NLP Parsing

- In fact, sentences can have a very large number of possible parses

- It is also difficult to construct a grammar with enough coverage
  - A less constrained grammar can parse more sentences but result in more parses for even simple sentences
  - There is no way to choose the right parse!

Binary notion:
in or not in language

- Need to be able to assign scores to parses

Statistical parsing

- **Adding probabilities to the rules**: probabilistic CFGs (PCFGs)

- **Learning from data**: treebanks

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)

The Penn Treebank Project (Marcus et al, 1993)

Probabilistic context-free grammars (PCFGs)

- A CFG tells us whether a sentence is in the language it defines

- A PCFG gives us a mechanism for assigning scores (here, probabilities) to different parses for the same sentence.

Probabilistic context-free grammars (PCFGs)

S	$\rightarrow$	NP	VP	1.0
VP	$\rightarrow$	Vi		0.3
VP	$\rightarrow$	Vt	NP	0.5
VP	$\rightarrow$	VP	PP	0.2
NP	$\rightarrow$	DT	NN	0.8
NP	$\rightarrow$	NP	PP	0.2
PP	$\rightarrow$	IN	NP	1.0

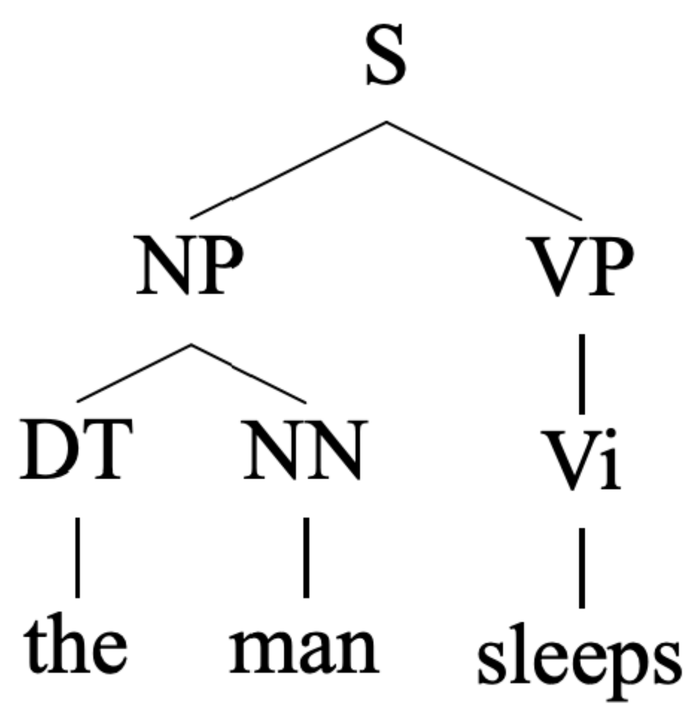
Vi	$\rightarrow$	sleeps	1.0
Vt	$\rightarrow$	saw	1.0
NN	$\rightarrow$	man	0.1
NN	$\rightarrow$	woman	0.1
NN	$\rightarrow$	telescope	0.3
NN	$\rightarrow$	dog	0.5
DT	$\rightarrow$	the	1.0
IN	$\rightarrow$	with	0.6
IN	$\rightarrow$	in	0.4

- A probabilistic context-free grammar (PCFG) consists of:
  - 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 grammars (PCFGs)

For any derivation (parse tree) containing rules:
$\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \ldots, \alpha_l \rightarrow \beta_l$, the probability of the parse is: $\prod_{i=1}^{l} q(\alpha_i \rightarrow \beta_i)$



R				q
S	→	NP	VP	1.0
VP	→	Vi		0.3
VP	→	Vt	NP	0.5
VP	→	VP	PP	0.2
NP	→	DT	NN	0.8
NP	→	NP	PP	0.2
PP	→	IN	NP	1.0

Vi	→	sleeps	1.0
Vt	→	saw	1.0
NN	→	man	0.1
NN	→	woman	0.1
NN	→	telescope	0.3
NN	→	dog	0.5
DT	→	the	1.0
IN	→	with	0.6
IN	→	in	0.4

$$P(t) = q(\text{S} \rightarrow \text{NP VP}) \times q(\text{NP} \rightarrow \text{DT NN}) \times q(\text{DT} \rightarrow \text{the})$$
$$\times\, q(\text{NN} \rightarrow \text{man}) \times q(\text{VP} \rightarrow \text{Vi}) \times q(\text{Vi} \rightarrow \text{sleeps})$$
$$= 1.0 \times 0.8 \times 1.0 \times 0.1 \times 0.3 \times 1.0 = 0.024$$

Why do we want $\sum_{\alpha \rightarrow \beta : \alpha = X} q(\alpha \rightarrow \beta) = 1$?

Treebanks

English

- Standard setup (WSJ portion of Penn Treebank):
  - 40,000 sentences for training
  - 1,700 for development
  - 2,400 for testing

Penn Treebank (1989-1996)
- Syntactic annotation of text for POS tagging, parses, predicate-arguments, and speech disfluencies
- WSJ articles from 3 years

- Why building a treebank instead of a grammar?
  - Broad coverage
  - Frequencies and distributional information
  - A way to evaluate systems

Penn Treebank

Phrasal categories

ADJP	Adjective phrase
ADVP	Adverb phrase
NP	Noun phrase
PP	Prepositional phrase
S	Simple declarative clause
SBAR	Subordinate clause
SBARQ	Direct question introduced by *wh*-element
SINV	Declarative sentence with subject-aux inversion
SQ	Yes/no questions and subconstituent of SBARQ excluding *wh*-element
VP	Verb phrase
WHADVP	Wh-adverb phrase
WHNP	Wh-noun phrase
WHPP	Wh-prepositional phrase
X	Constituent of unknown or uncertain category
*	"Understood" subject of infinitive or imperative
0	Zero variant of *that* in subordinate clauses
T	Trace of wh-Constituent

Part-of-speech tagset

Penn Treebank

CC	Coordinating conj.	TO	infinitival *to*
CD	Cardinal number	UH	Interjection
DT	Determiner	VB	Verb, base form
EX	Existential there	VBD	Verb, past tense
FW	Foreign word	VBG	Verb, gerund/present pple
IN	Preposition	VBN	Verb, past participle
JJ	Adjective	VBP	Verb, non-3rd ps. sg. present
JJR	Adjective, comparative	VBZ	Verb, 3rd ps. sg. present
JJS	Adjective, superlative	WDT	Wh-determiner
LS	List item marker	WP	Wh-pronoun
MD	Modal	WP$	Possessive *wh*-pronoun
NN	Noun, singular or mass	WRB	Wh-adverb
NNS	Noun, plural	#	Pound sign
NNP	Proper noun, singular	$	Dollar sign
NNPS	Proper noun, plural	.	Sentence-final punctuation
PDT	Predeterminer	,	Comma
POS	Possessive ending	:	Colon, semi-colon
PRP	Personal pronoun	(	Left bracket character
PP$	Possessive pronoun	)	Right bracket character
RB	Adverb	"	Straight double quote
RBR	Adverb, comparative	‘	Left open single quote
RBS	Adverb, superlative	“	Left open double quote
RP	Particle	’	Right close single quote
SYM	Symbol	”	Right close double quote

Deriving a PCFG from a treebank

- Training data: a set of parse trees $t_1, t_2, \ldots, t_m$
- 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 the start symbol S.
  - R is taken to be the set of all rules $\alpha \to \beta$ seen in the trees
  - The maximum-likelihood parameter estimates are:

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

Can add smoothing

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

What if there is no annotated parses?

- Use Expectation Maximization.
- For learning parameters for PCFGs
  - E-Step: compute expectation over trees with fixed model weights (probabilities)
  - M-Step: determine model weights (probabilities) that maximize likelihood of expected parses
- Use the inside-outside algorithm (a dynamic programming algorithm) to compute these probabilities efficiently.

Parsing with PCFGs

- Given a sentence s and a PCFG, how to find the highest scoring parse tree for s?

$$argmax_{t \in \mathcal{T}(s)} P(t)$$

- **The CKY algorithm**: applies to a PCFG in Chomsky normal form (CNF)

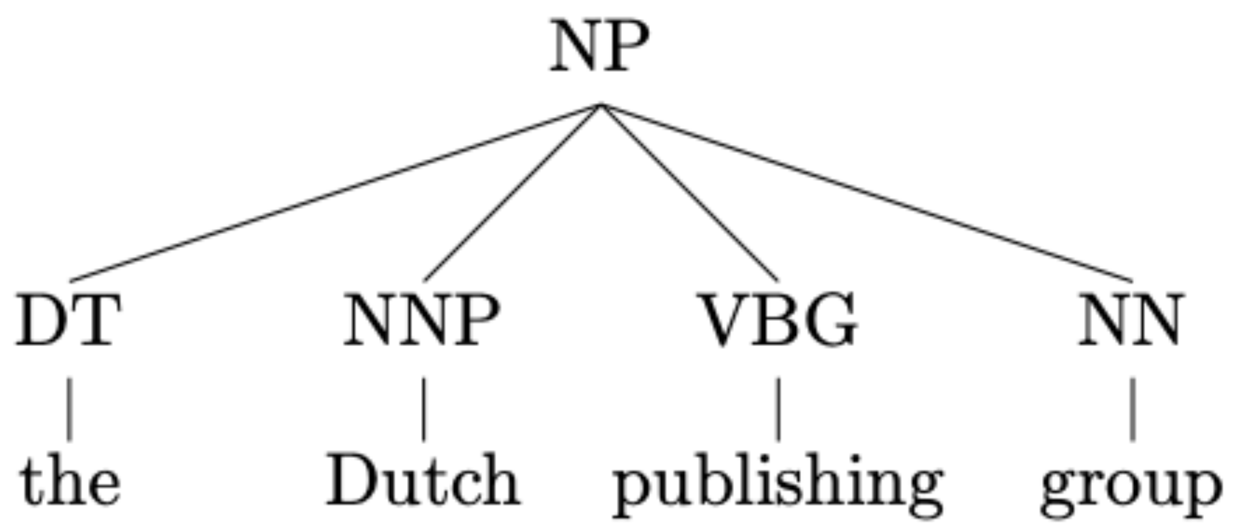
- **Chomsky Normal Form (CNF)**: all the rules take one of the two following forms:

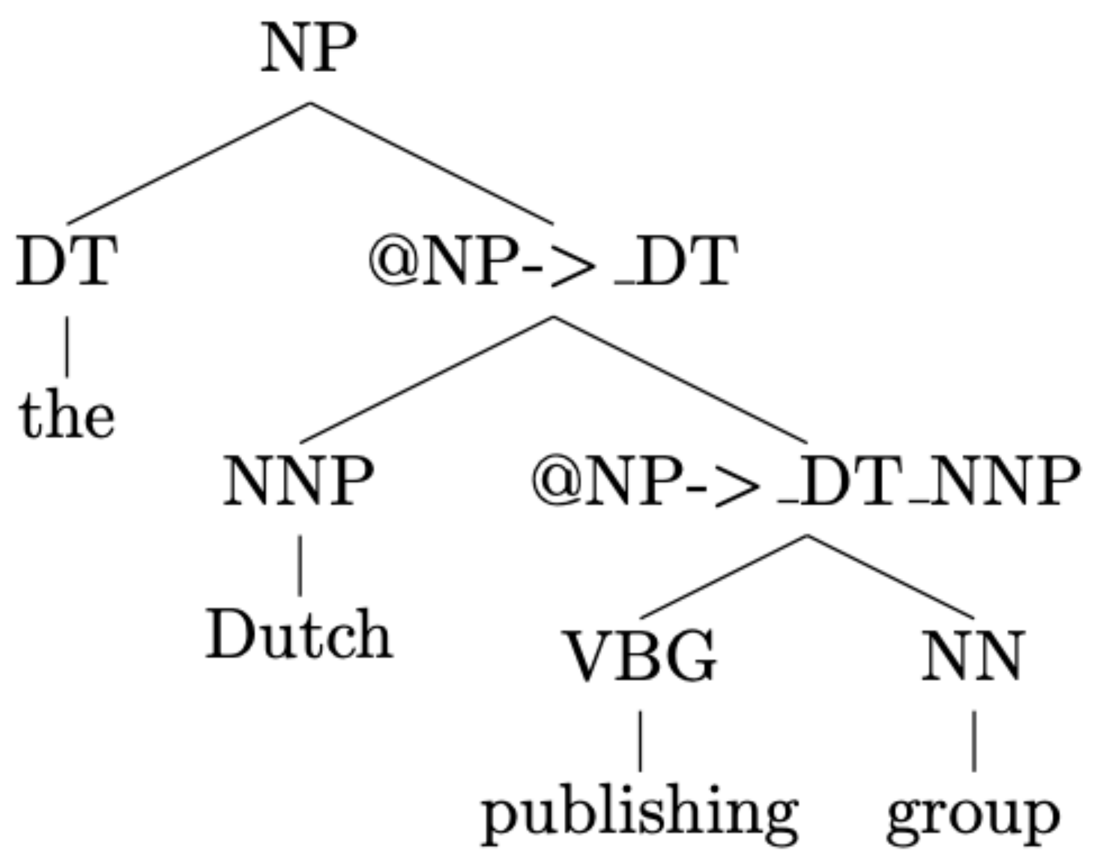
    - $X \rightarrow Y_1 Y_2$ where $X \in N, Y_1 \in N, Y_2 \in N$ Binary
    - $X \rightarrow Y$ where $X \in N, Y \in \Sigma$ Unary

- Can convert any PCFG into an equivalent grammar in CNF!
    - However, the trees will look differently
    - Possible to do "reverse transformation"

Converting PCFGs into a CNF grammar

- n-ary rules ($n > 2$): NP → DT NNP VBG NN





- Unary rules: VP → Vi, Vi → sleeps
  - Eliminate all the unary rules recursively by adding VP → sleeps
  - We will come back to this later!

The CKY algorithm

Cocke–Kasami-Younger

- Dynamic programming

- Given a sentence $x_1, x_2, \ldots, x_n$, denote $\pi(i, j, X)$ as the highest score for any parse tree that dominates words $x_i, \ldots, x_j$ and has non-terminal $X \in N$ as its root.

- Output: $\pi(1,n, S)$

- Initially, for $i = 1,2,\ldots, n$,

$$\pi(i, i, X) = \begin{cases} q(X \to x_i) & \text{if } X \to x_i \in R \\ 0 & \text{otherwise} \end{cases}$$

Book $_1$ the $_2$ flight $_3$ through $_4$ Houston $_5$

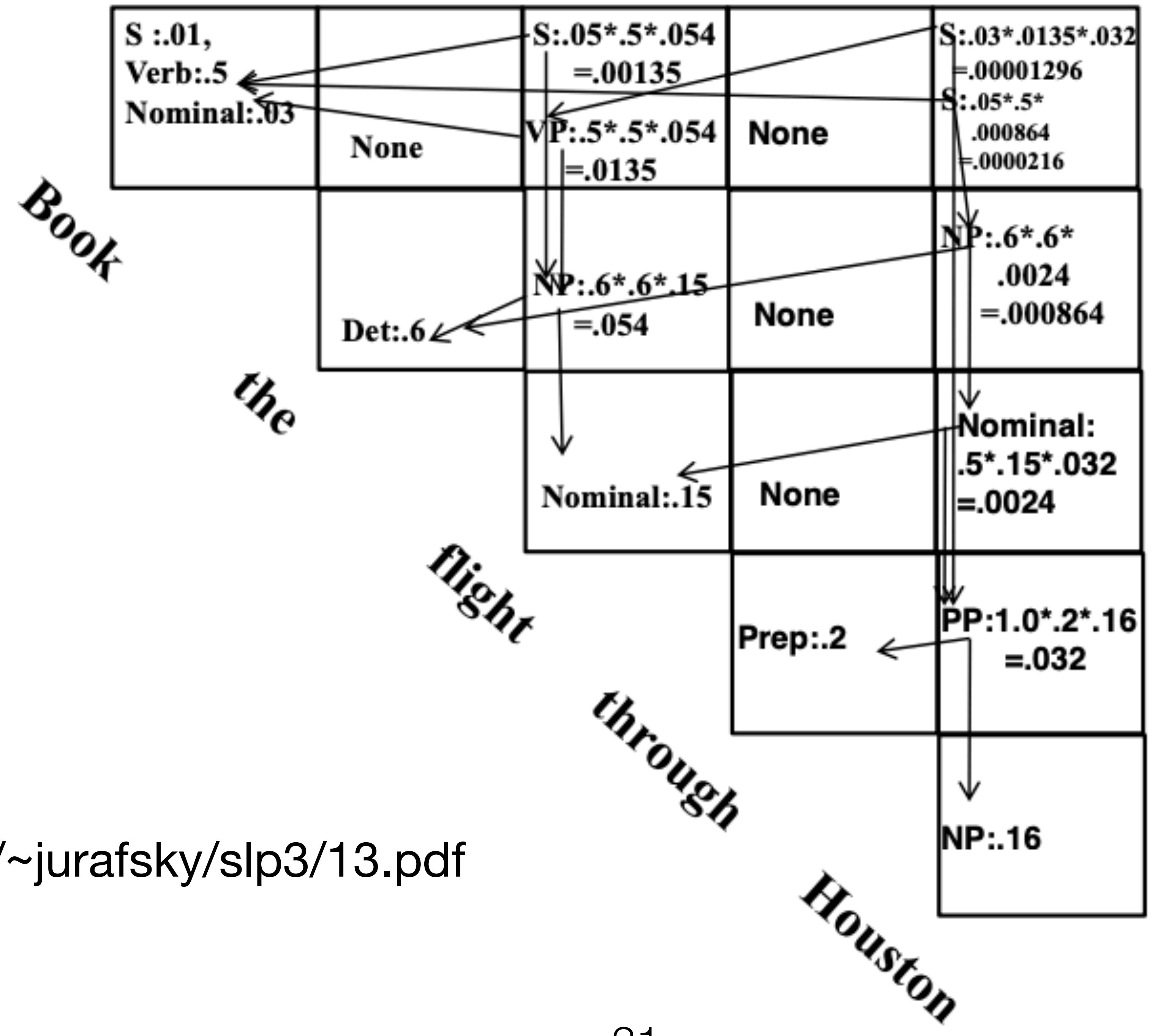
The CKY algorithm

- For all (i, j) such that $1 \leq i < j \leq n$ for all $X \in N$,

$$\pi(i,j,X) = \max_{X \to YZ \in R, i \leq k < j} q(X \to YZ) \times \pi(i,k,Y) \times \pi(k+1,j,Z)$$

Consider all ways span (i,j) can be split into 2 (k is the split point)

Also stores backpointers which allow us to recover the parse tree



https://web.stanford.edu/~jurafsky/slp3/13.pdf

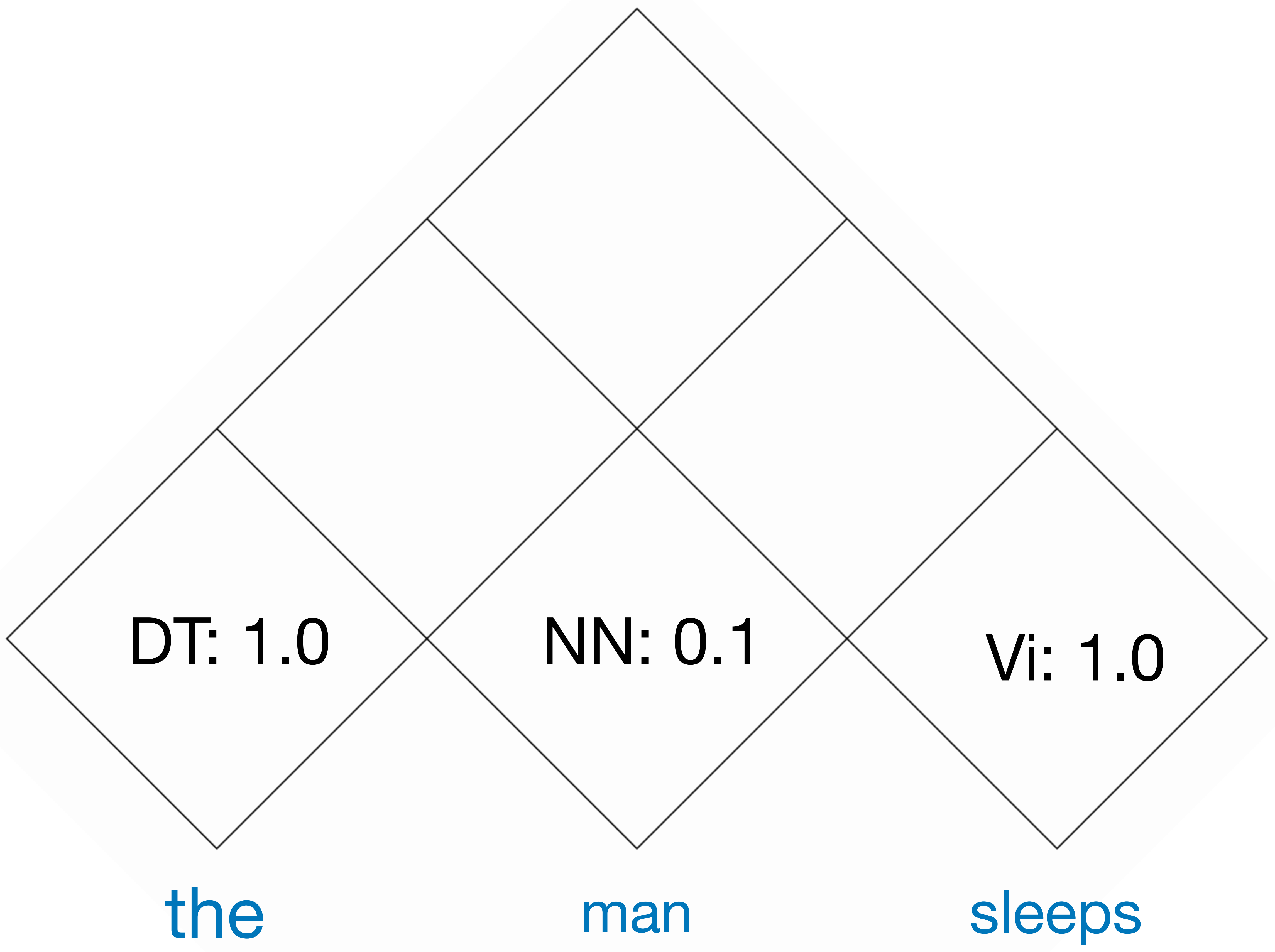
Cells contain:

- Best score for parse of span (i,j) for each non-terminal X
- Backpointers

Example of CKY parsing

S	→	NP	VP	1.0
VP	→	Vi		0.3
VP	→	Vt	NP	0.5
VP	→	VP	PP	0.2
NP	→	DT	NN	0.8
NP	→	NP	PP	0.2
PP	→	IN	NP	1.0

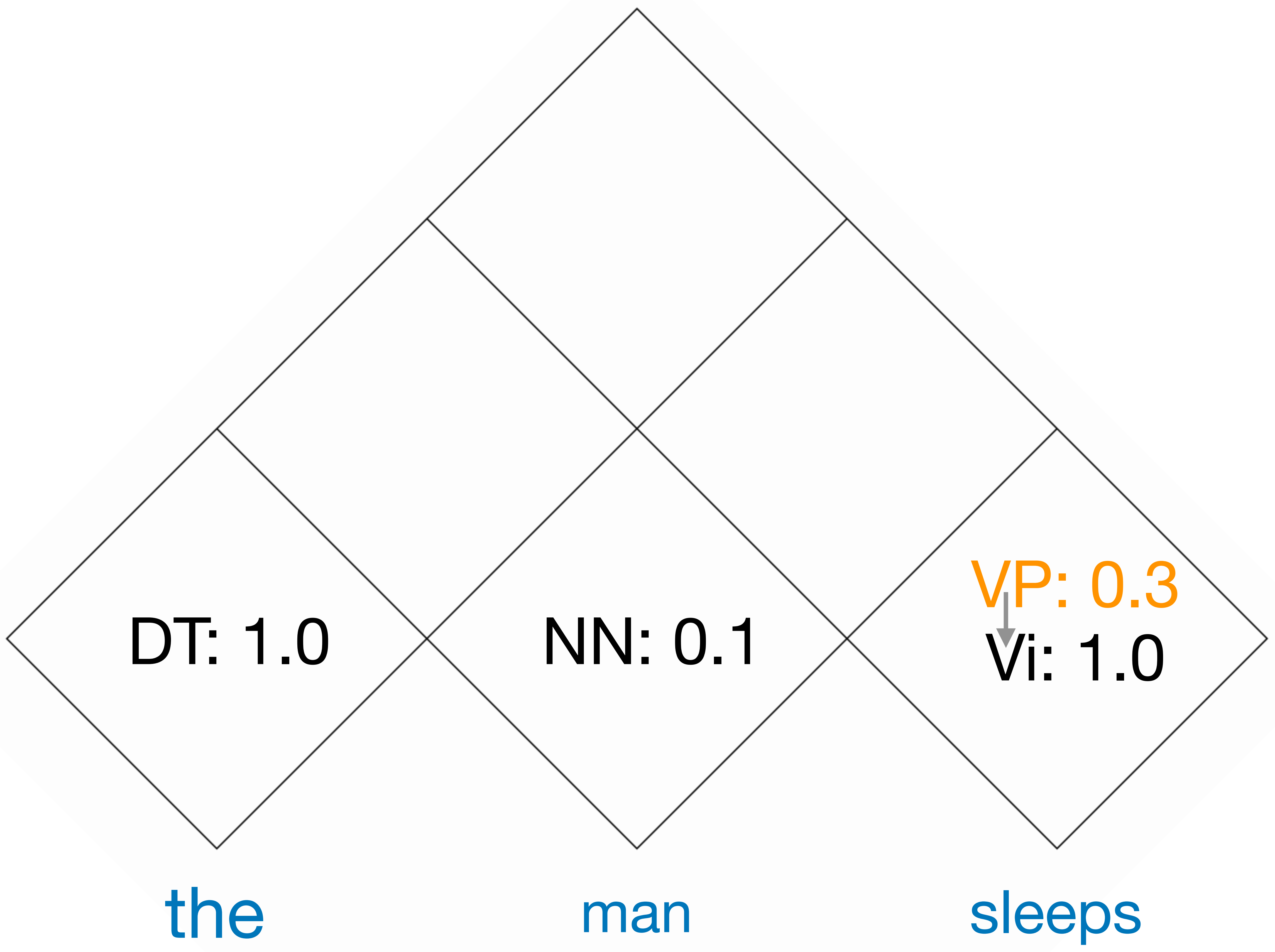
Vi	→	sleeps	1.0
Vt	→	saw	1.0
NN	→	man	0.1
NN	→	woman	0.1
NN	→	telescope	0.3
NN	→	dog	0.5
DT	→	the	1.0
IN	→	with	0.6
IN	→	in	0.4



Example of CKY parsing

S	→	NP	VP	1.0
VP	→	Vi		0.3
VP	→	Vt	NP	0.5
VP	→	VP	PP	0.2
NP	→	DT	NN	0.8
NP	→	NP	PP	0.2
PP	→	IN	NP	1.0

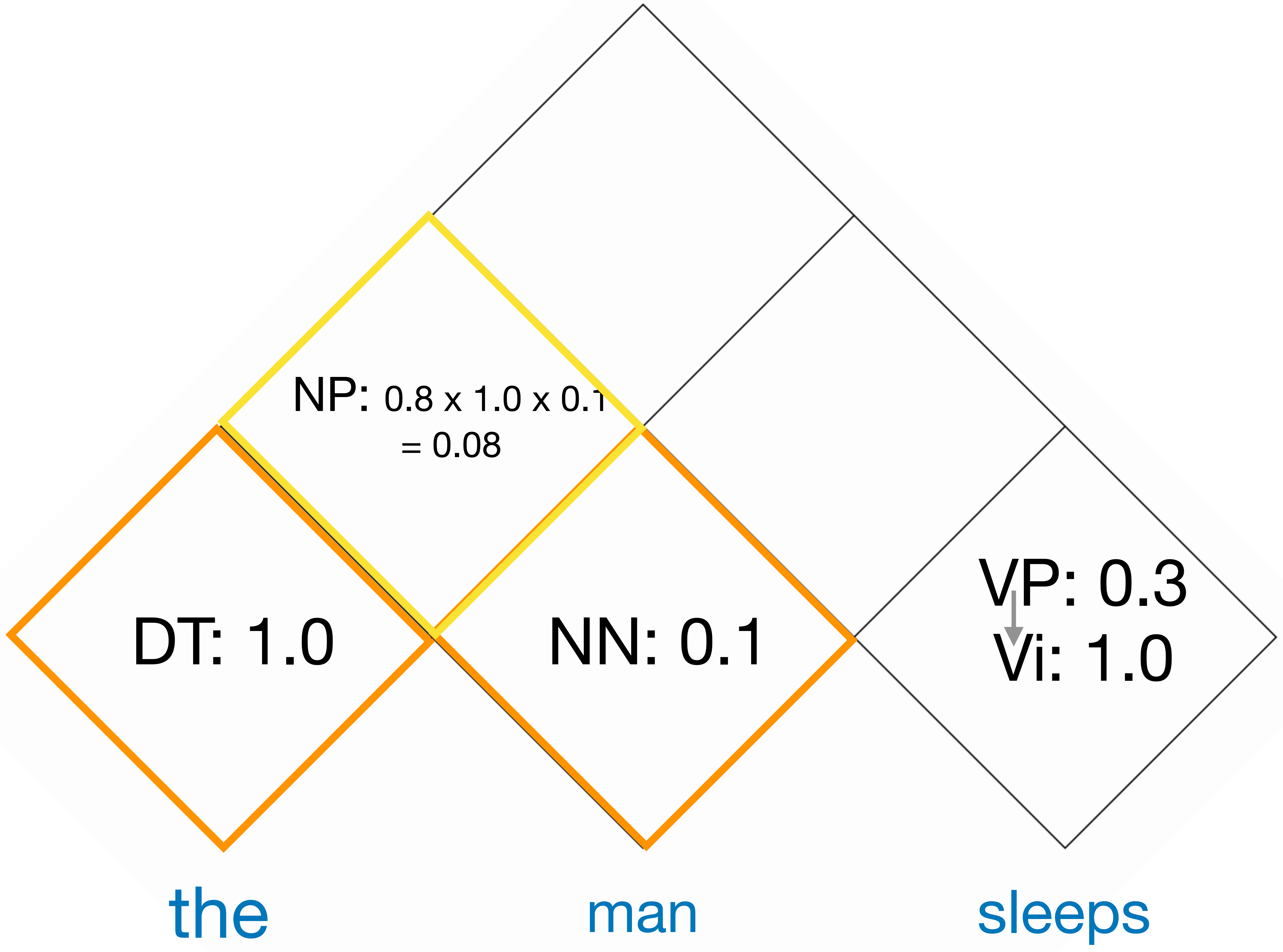
Vi	→	sleeps	1.0
Vt	→	saw	1.0
NN	→	man	0.1
NN	→	woman	0.1
NN	→	telescope	0.3
NN	→	dog	0.5
DT	→	the	1.0
IN	→	with	0.6
IN	→	in	0.4



Example of CKY parsing

S	→	NP	VP	1.0
VP	→	Vi		0.3
VP	→	Vt	NP	0.5
VP	→	VP	PP	0.2
NP	→	DT	NN	0.8
NP	→	NP	PP	0.2
PP	→	IN	NP	1.0

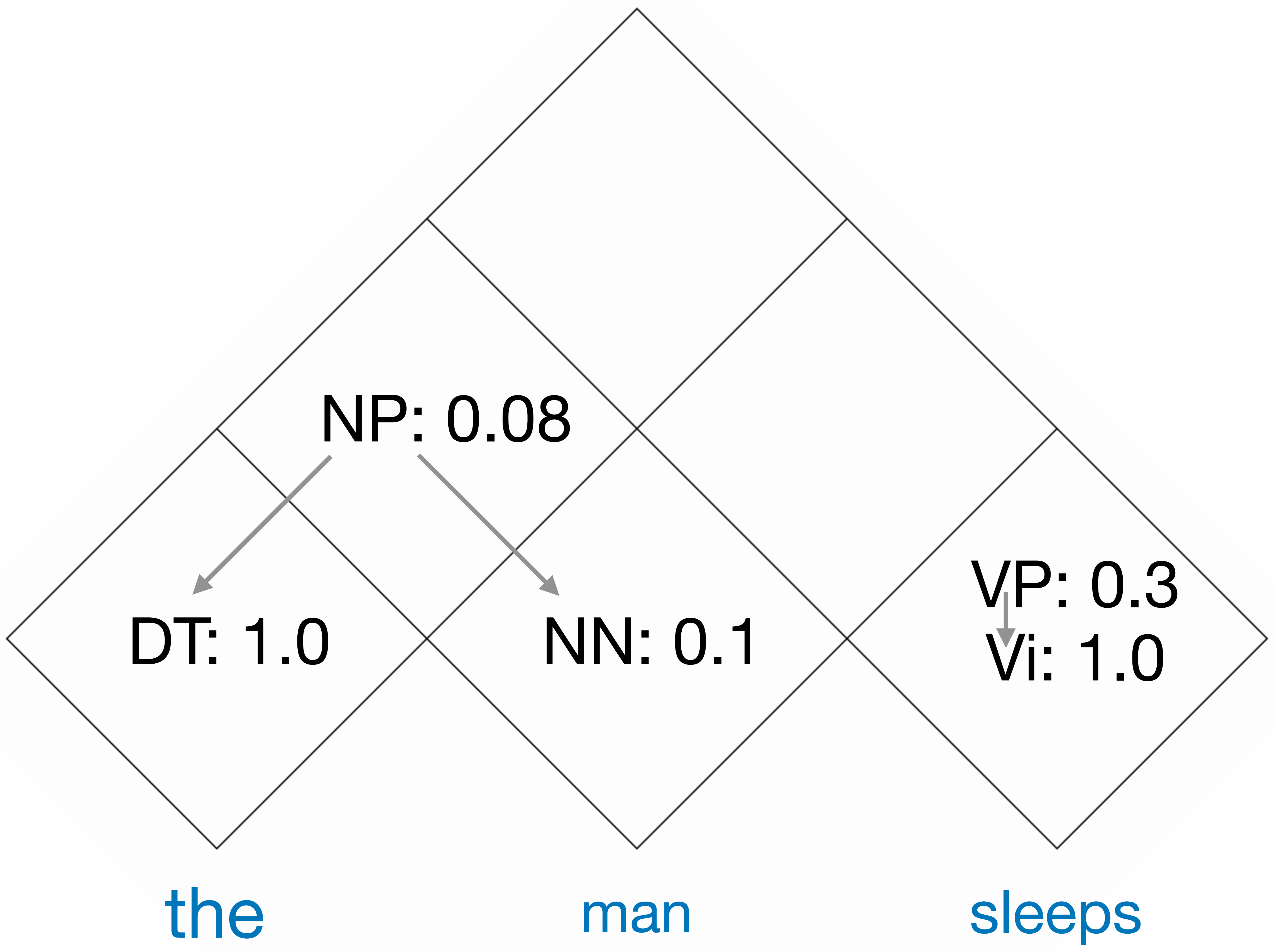
Vi	→	sleeps	1.0
Vt	→	saw	1.0
NN	→	man	0.1
NN	→	woman	0.1
NN	→	telescope	0.3
NN	→	dog	0.5
DT	→	the	1.0
IN	→	with	0.6
IN	→	in	0.4



Example of CKY parsing

S	→	NP	VP	1.0
VP	→	Vi		0.3
VP	→	Vt	NP	0.5
VP	→	VP	PP	0.2
NP	→	DT	NN	0.8
NP	→	NP	PP	0.2
PP	→	IN	NP	1.0

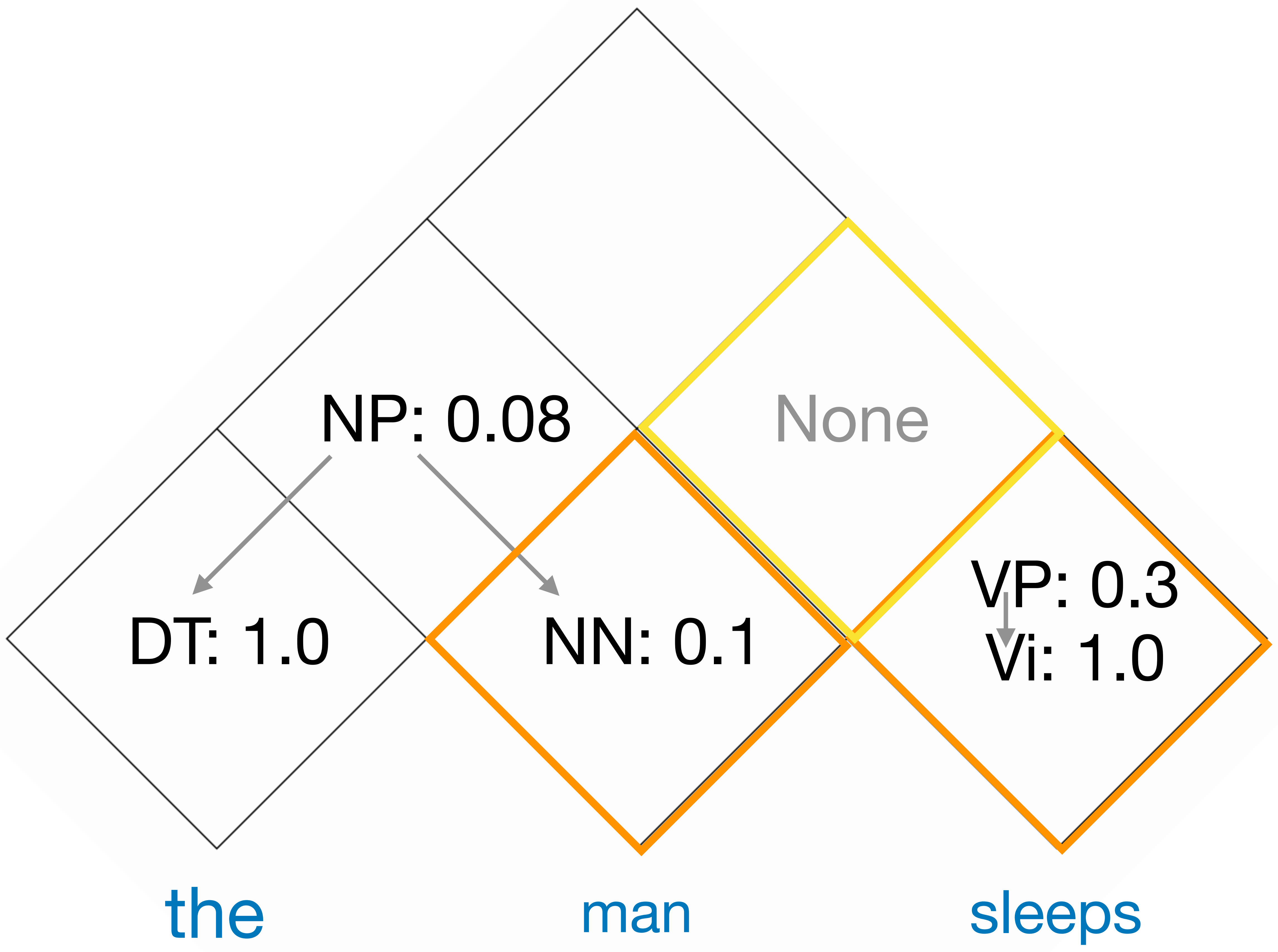
Vi	→	sleeps	1.0
Vt	→	saw	1.0
NN	→	man	0.1
NN	→	woman	0.1
NN	→	telescope	0.3
NN	→	dog	0.5
DT	→	the	1.0
IN	→	with	0.6
IN	→	in	0.4



Example of CKY parsing

S	→	NP	VP	1.0
VP	→	Vi		0.3
VP	→	Vt	NP	0.5
VP	→	VP	PP	0.2
NP	→	DT	NN	0.8
NP	→	NP	PP	0.2
PP	→	IN	NP	1.0

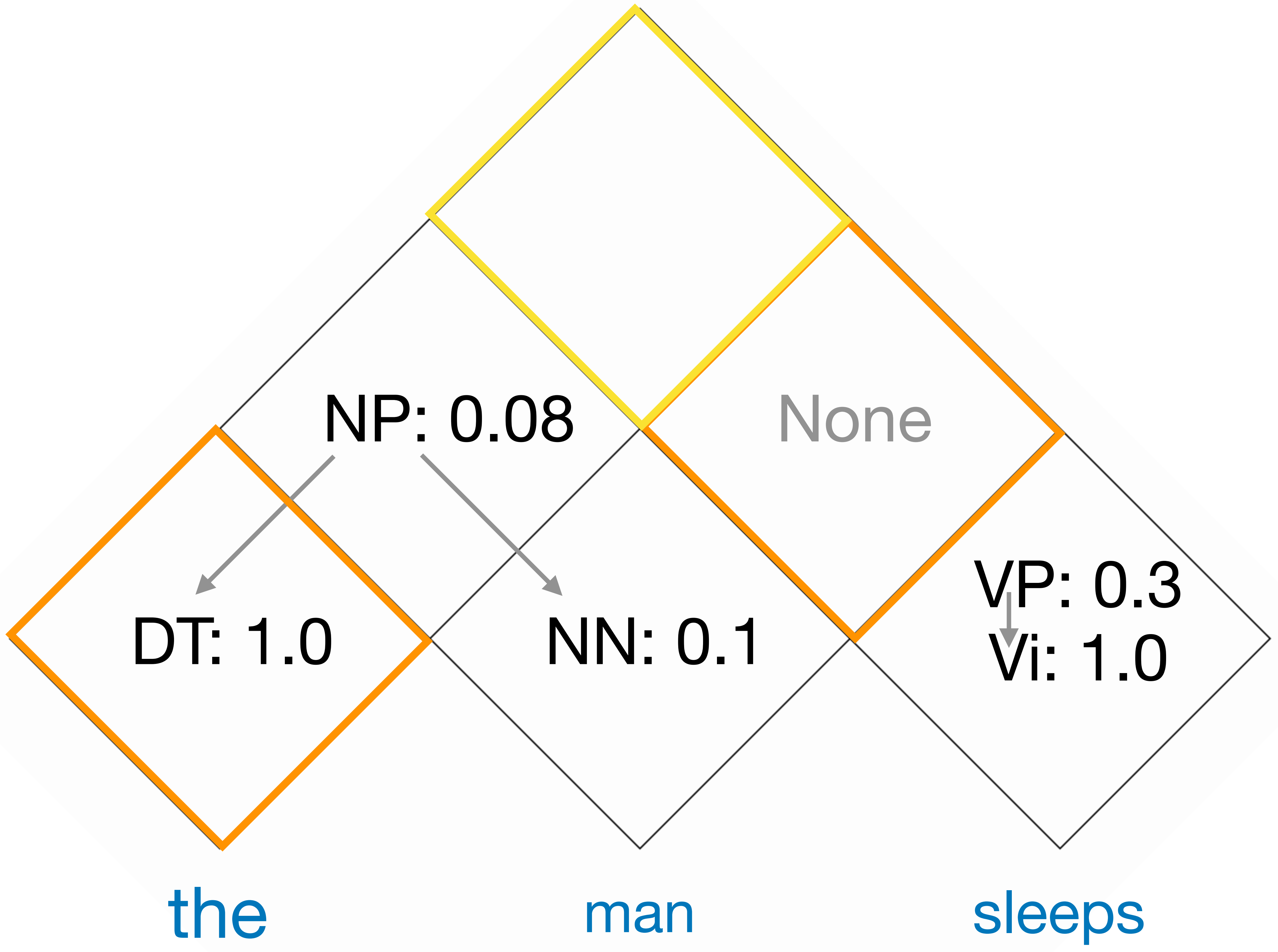
Vi	→	sleeps	1.0
Vt	→	saw	1.0
NN	→	man	0.1
NN	→	woman	0.1
NN	→	telescope	0.3
NN	→	dog	0.5
DT	→	the	1.0
IN	→	with	0.6
IN	→	in	0.4



Example of CKY parsing

S	→	NP	VP	1.0
VP	→	Vi		0.3
VP	→	Vt	NP	0.5
VP	→	VP	PP	0.2
NP	→	DT	NN	0.8
NP	→	NP	PP	0.2
PP	→	IN	NP	1.0

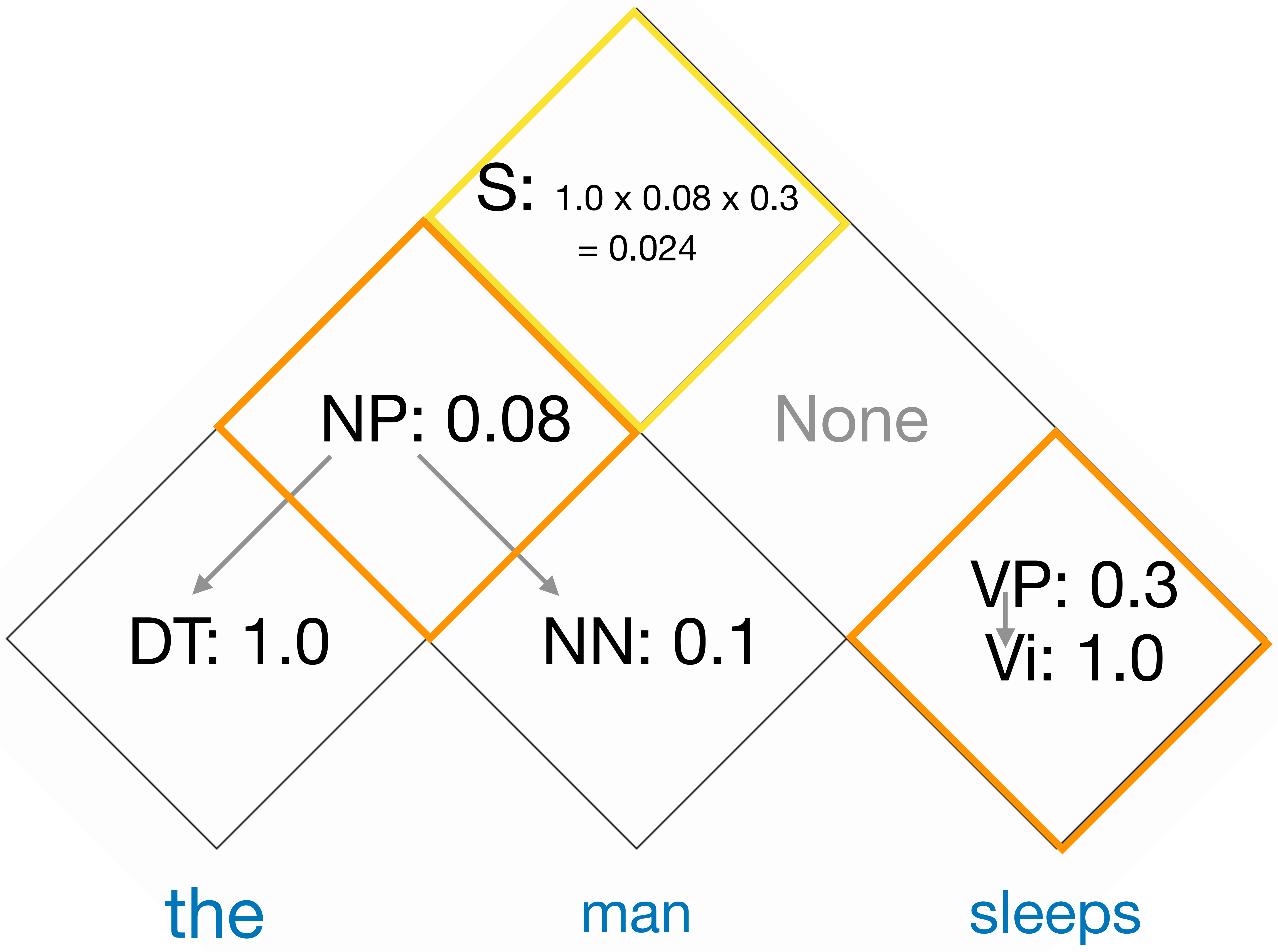
Vi	→	sleeps	1.0
Vt	→	saw	1.0
NN	→	man	0.1
NN	→	woman	0.1
NN	→	telescope	0.3
NN	→	dog	0.5
DT	→	the	1.0
IN	→	with	0.6
IN	→	in	0.4



Example of CKY parsing

S	→	NP	VP	1.0
VP	→	Vi		0.3
VP	→	Vt	NP	0.5
VP	→	VP	PP	0.2
NP	→	DT	NN	0.8
NP	→	NP	PP	0.2
PP	→	IN	NP	1.0

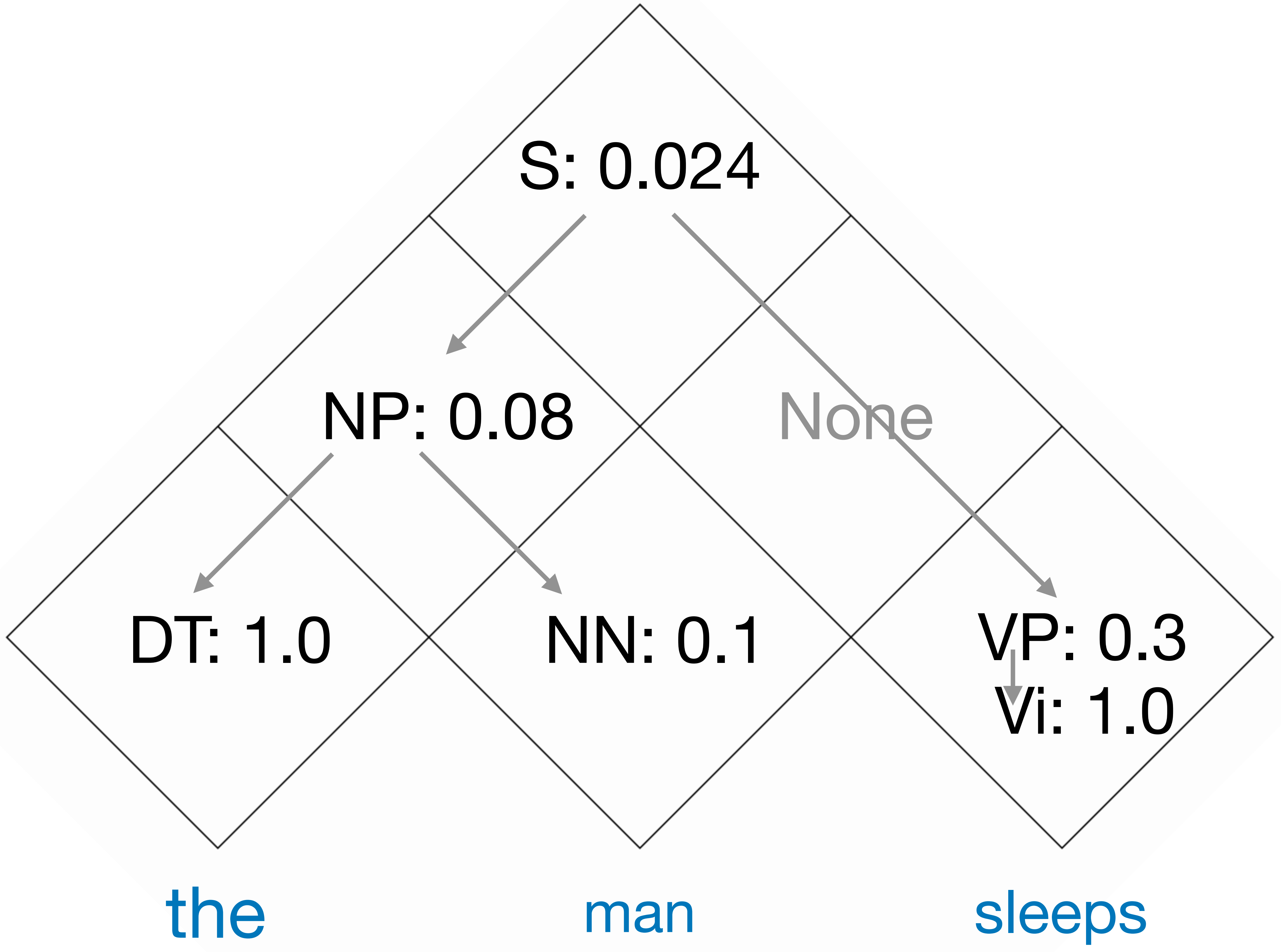
Vi	→	sleeps	1.0
Vt	→	saw	1.0
NN	→	man	0.1
NN	→	woman	0.1
NN	→	telescope	0.3
NN	→	dog	0.5
DT	→	the	1.0
IN	→	with	0.6
IN	→	in	0.4



Example of CKY parsing

S	→	NP	VP	1.0
VP	→	Vi		0.3
VP	→	Vt	NP	0.5
VP	→	VP	PP	0.2
NP	→	DT	NN	0.8
NP	→	NP	PP	0.2
PP	→	IN	NP	1.0

Vi	→	sleeps	1.0
Vt	→	saw	1.0
NN	→	man	0.1
NN	→	woman	0.1
NN	→	telescope	0.3
NN	→	dog	0.5
DT	→	the	1.0
IN	→	with	0.6
IN	→	in	0.4



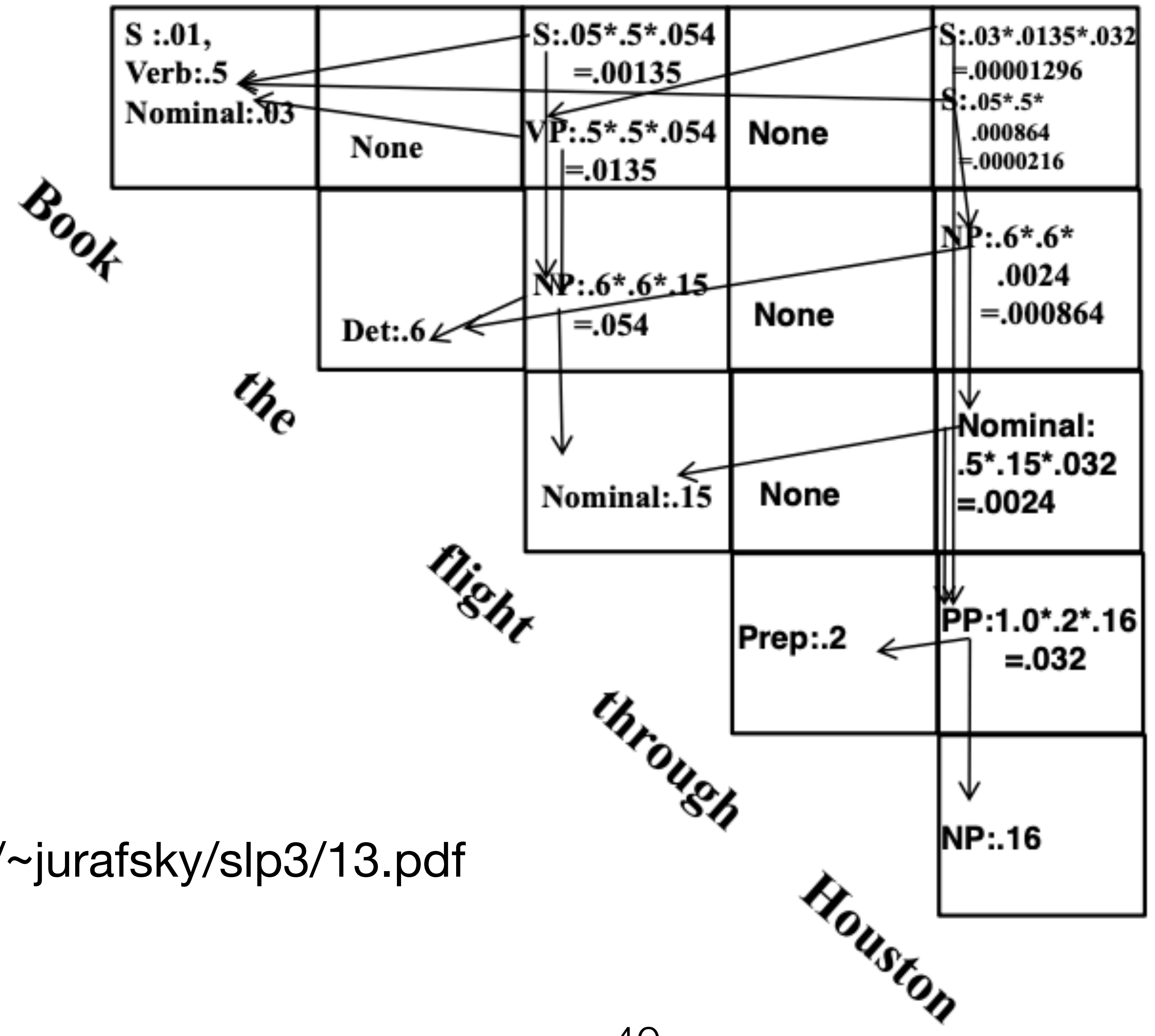
The CKY algorithm

- For all (i, j) such that $1 \leq i < j \leq n$ for all $X \in N$,

$$\pi(i, j, X) = \max_{X \to YZ \in R, i \leq k < j} q(X \to YZ) \times \pi(i, k, Y) \times \pi(k+1, j, Z)$$

Consider all ways span (i,j) can be split into 2 (k is the split point)

Also stores backpointers which allow us to recover the parse tree



Cells contain:

- Best score for parse of span (i,j) for each non-terminal X
- Backpointers

https://web.stanford.edu/~jurafsky/slp3/13.pdf

The CKY algorithm

Input: a sentence $s = x_1 \ldots x_n$, a PCFG $G = (N, \Sigma, S, R, q)$.
Initialization:
For all $i \in \{1 \ldots n\}$, for all $X \in N$,

$$\pi(i, i, X) = \begin{cases} q(X \to x_i) & \text{if } X \to x_i \in R \\ 0 & \text{otherwise} \end{cases}$$

Algorithm:

- For $l = 1 \ldots (n-1)$
  - For $i = 1 \ldots (n-l)$
    - Set $j = i + l$
    - For all $X \in N$, calculate

$$\pi(i, j, X) = \max_{\substack{X \to YZ \in R, \\ s \in \{i \ldots (j-1)\}}} (q(X \to YZ) \times \pi(i, s, Y) \times \pi(s+1, j, Z))$$

and

$$bp(i, j, X) = \arg \max_{\substack{X \to YZ \in R, \\ s \in \{i \ldots (j-1)\}}} (q(X \to YZ) \times \pi(i, s, Y) \times \pi(s+1, j, Z))$$

Output: Return $\pi(1, n, S) = \max_{t \in \mathcal{T}(s)} p(t)$, and backpointers bp which allow recovery of $\arg\max_{t \in \mathcal{T}(s)} p(t)$.

Running time?

$O(n^3 |R|)$

CKY with unary rules

- In practice, we also allow unary rules:

$$X \rightarrow Y \text{ where } X, Y \in N$$

conversion to/from the normal form is easier

$$\pi(i, j, X) = \max_{X \rightarrow Y \in R} q(X \rightarrow Y) \times \pi(i, j, Y)$$

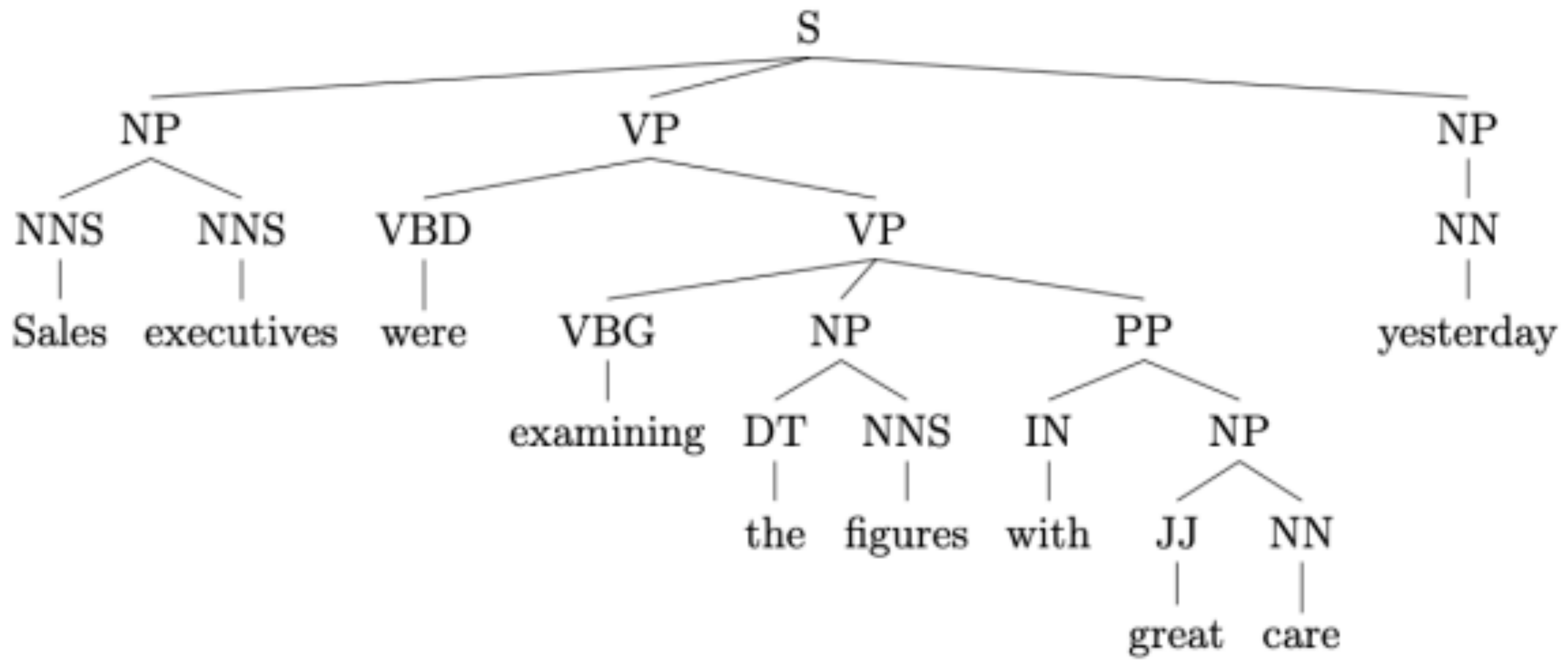
- Compute unary closure: if there is a rule chain $X \rightarrow Y_1, Y_1 \rightarrow Y_2, \ldots, Y_k \rightarrow Y$, add $q(X \rightarrow Y) = q(X \rightarrow Y_1) \times \cdots \times q(Y_k \rightarrow Y)$
- Update unary rule once after the binary rules

Constituency Parsing

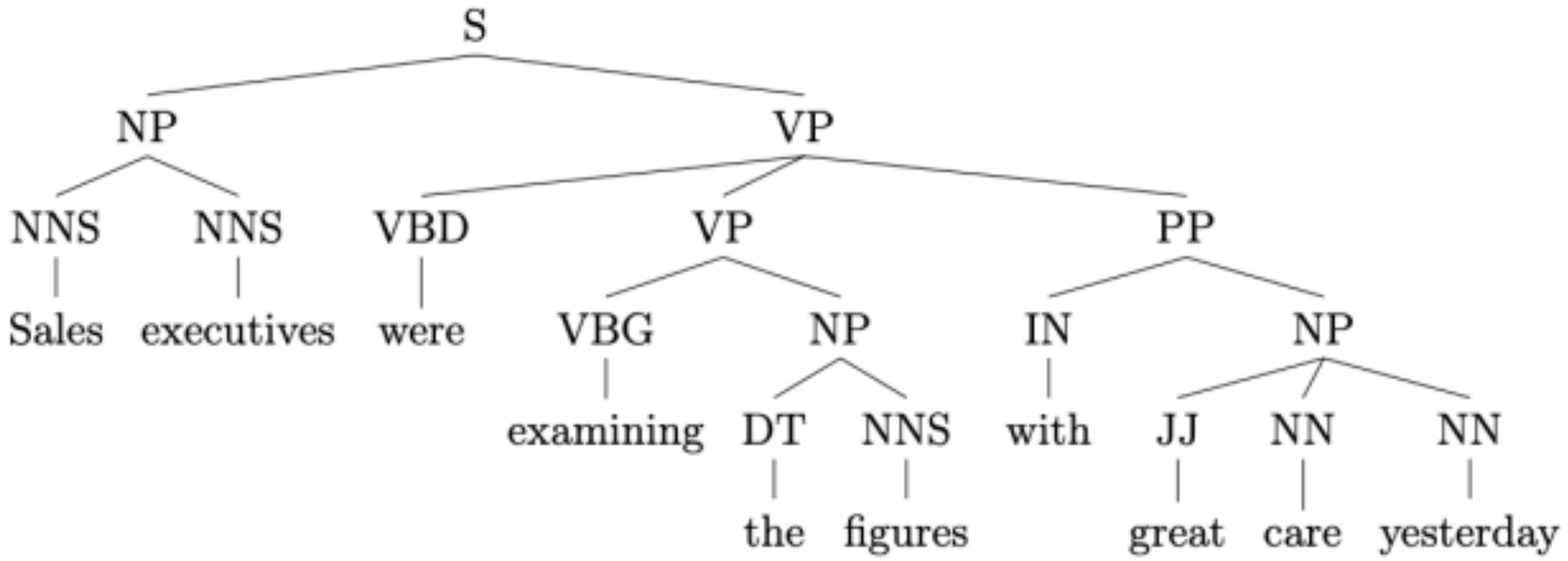
- Borealis AI Tutorials
  - Parsing I (https://www.borealisai.com/en/blog/tutorial-15-parsing-i-context-free-grammars-and-cyk-algorithm/)
    - CFGs and the CKY algorithm
    - CNF and number of parses
  - Parsing II (https://www.borealisai.com/en/blog/tutorial-18-parsing-ii-wcfgs-inside-algorithm-and-weighted-parsing/)
    - Weighted CFGs and CKY algorithm for parsing Weighted CFGs
  - Parsing III (https://www.borealisai.com/en/blog/tutorial-19-parsing-iii-pcfgs-and-inside-outside-algorithm/)
    - PCFGs
    - Parameter estimation for both supervised and unsupervised cases
      - Inside-Outside algorithm for unsupervised learning of parameters

Evaluating constituency parsing

Gold: **(1, 10, S), (1, 2, NP)**, (3, 9, VP), (4, 9, VP), **(5, 6, NP)**, (7, 9, PP), (8, 9, NP), (10, 10, NP)



Predicted: **(1, 10, S), (1, 2, NP)**, (3, 10, VP), (4, 6, VP), **(5, 6, NP)**, (7, 10, PP), (8, 10, NP)

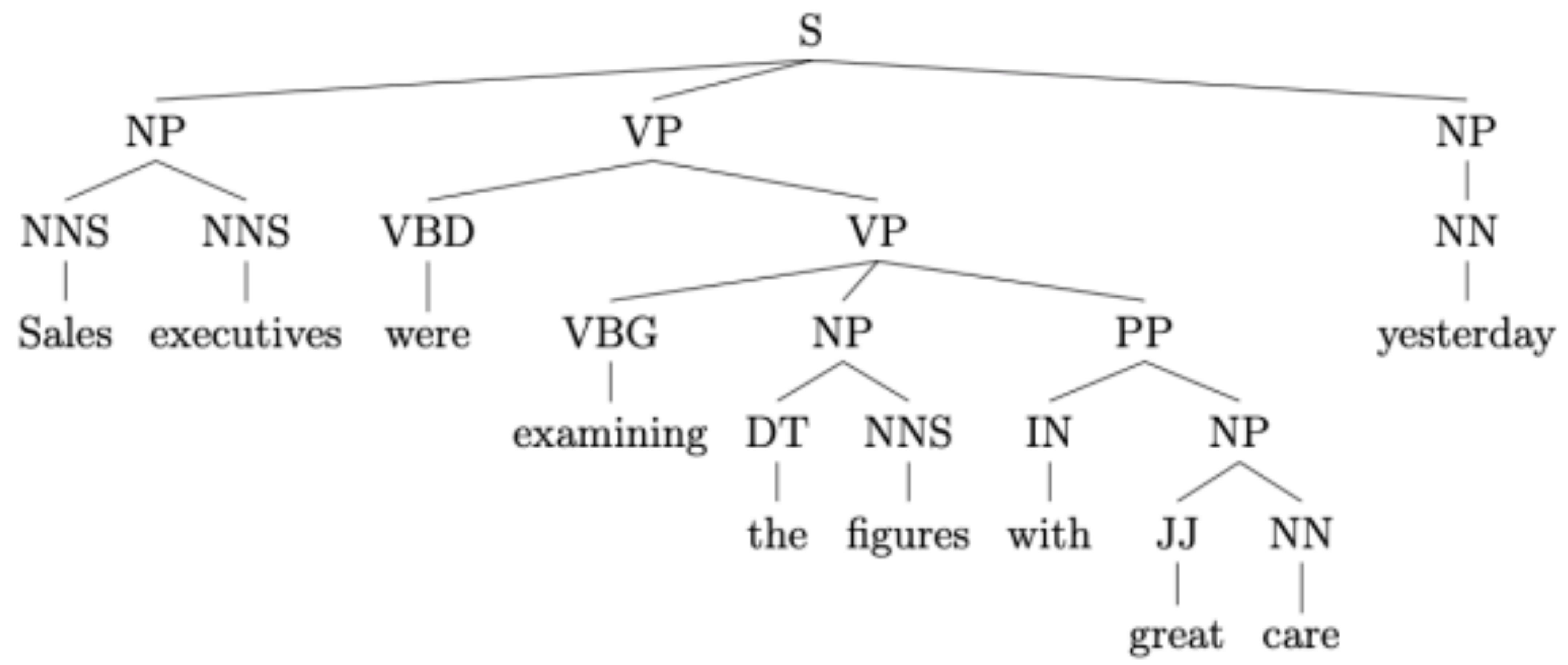


Evaluating constituency parsing

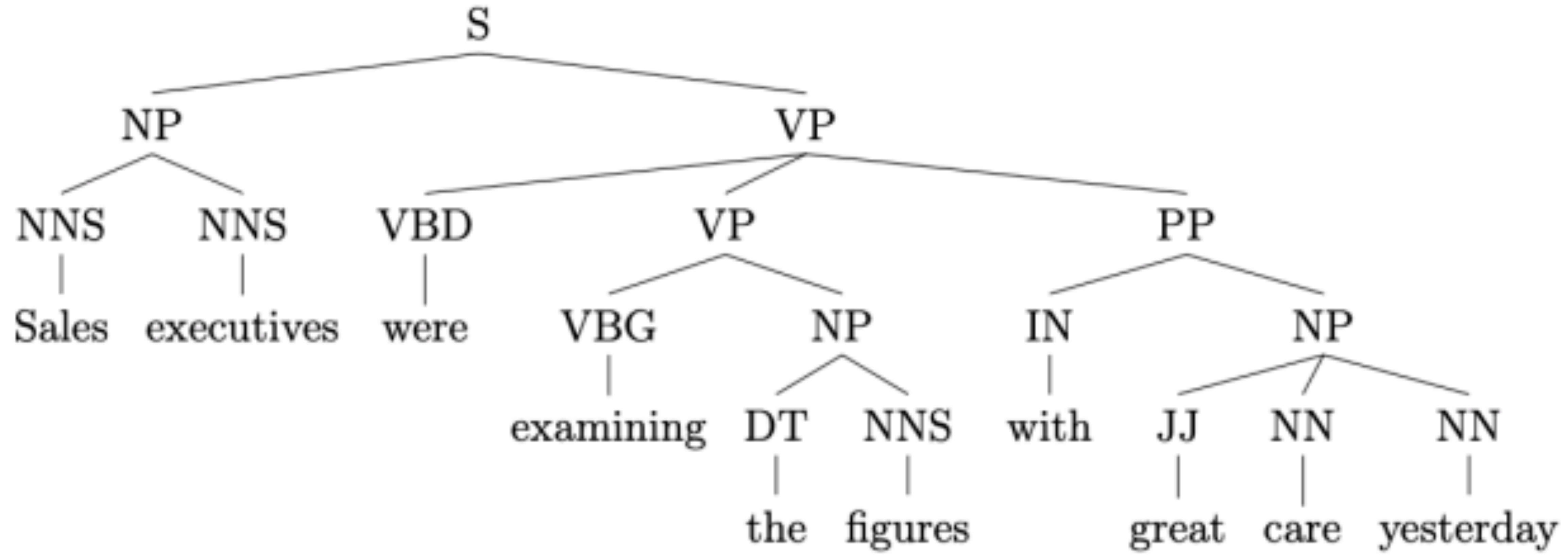
- Recall: (# correct constituents in candidate) / (# constituents in gold tree)
- Precision: (# correct constituents in candidate) / (# constituents in candidate)
- Labeled precision/recall require getting the non-terminal label correct
- F1 = (2 * precision * recall) / (precision + recall)
- Part-of-speech tagging accuracy is evaluated separately

Evaluating constituency parsing

Gold: **(1, 10, S), (1, 2, NP)**, (3, 9, VP), (4, 9, VP), **(5, 6, NP)**, (7, 9, PP), (8, 9, NP), (10, 10, NP)



Predicted: **(1, 10, S), (1, 2, NP)**, (3, 10, VP), (4, 6, VP), **(5, 6, NP)**, (7, 10, PP), (8, 10, NP)



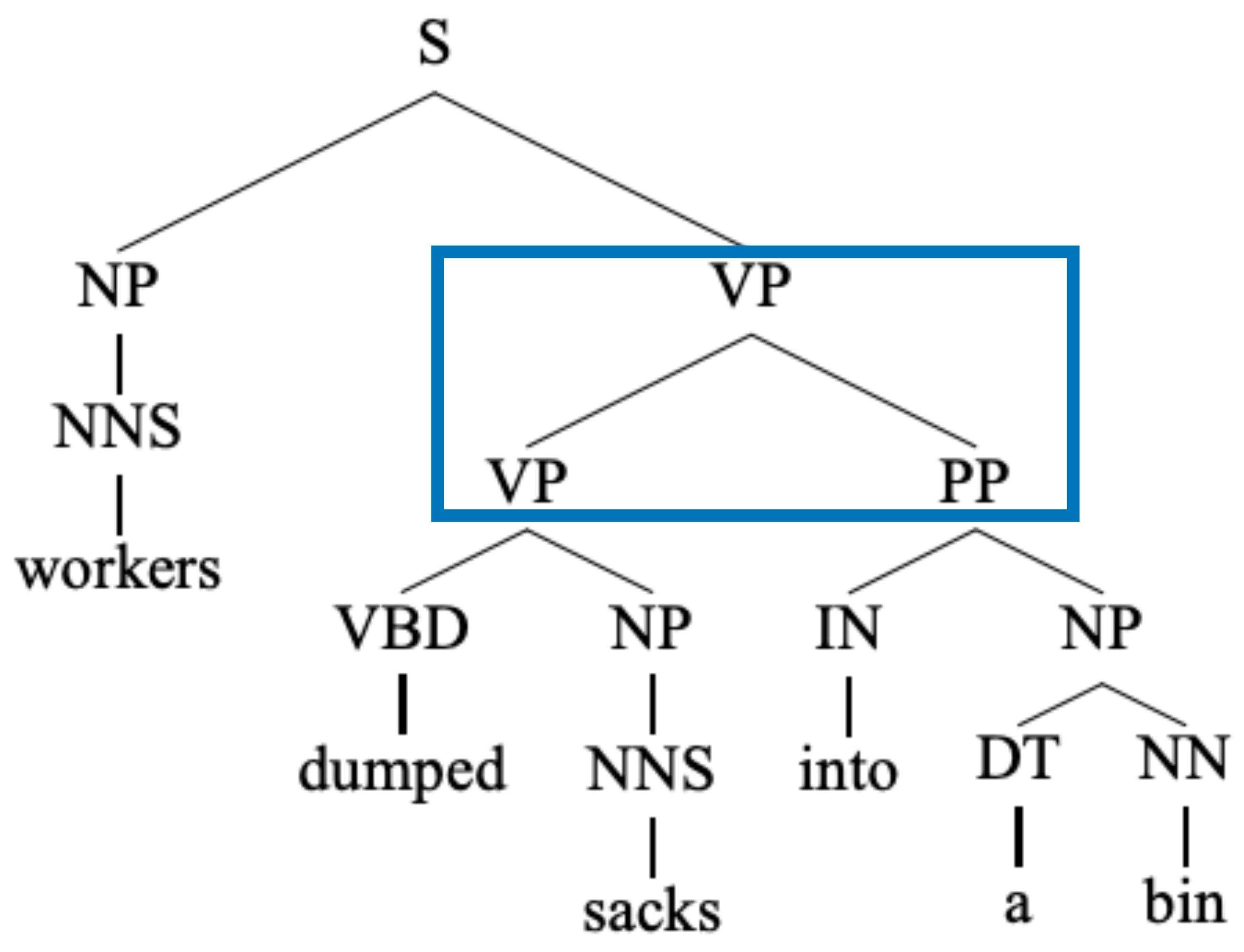
- Precision: 3/7 = 42.9%
- Recall: 3/8 = 37.5%
- F1 = 40.0%
- Tagging accuracy: 100%

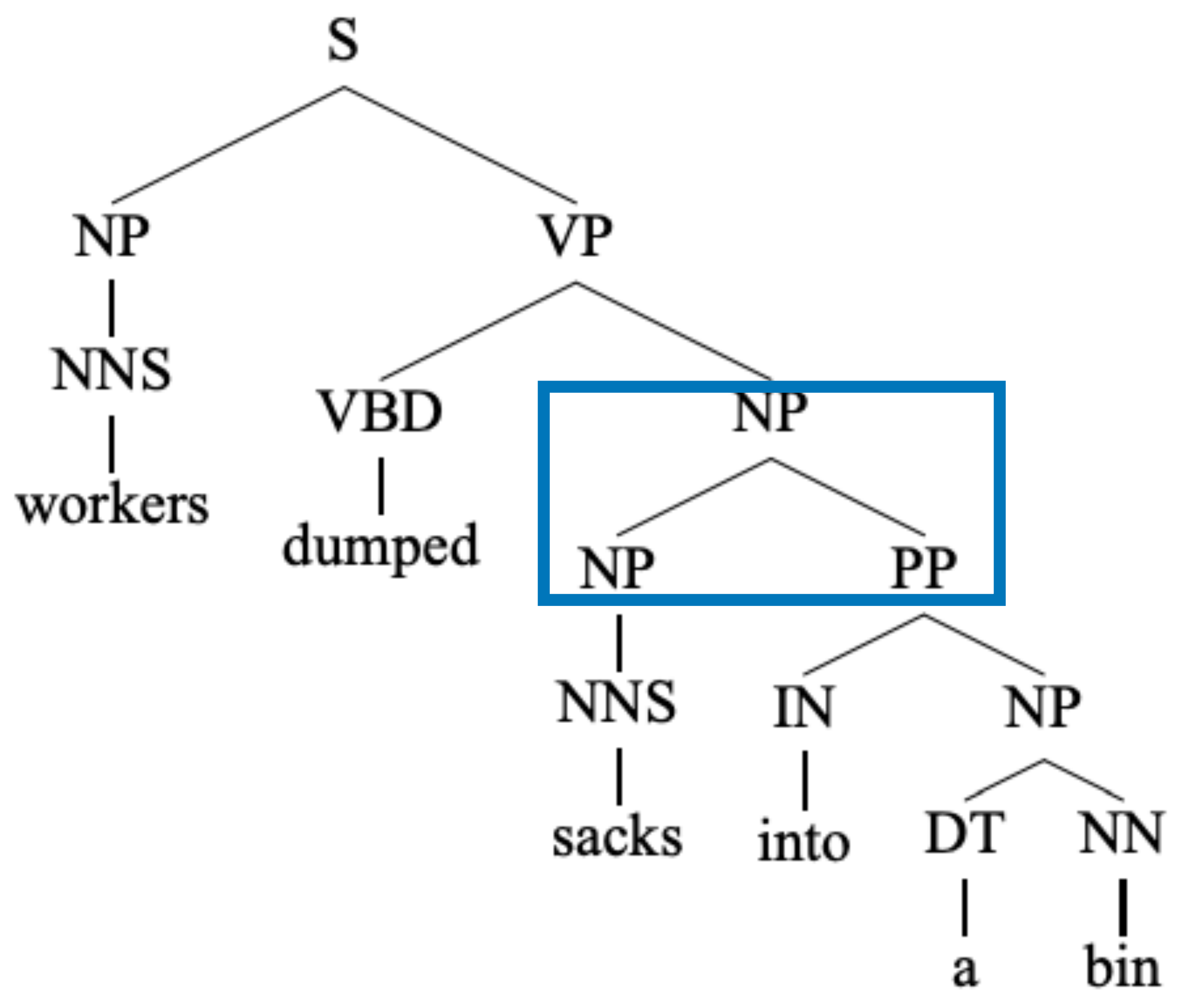
Weaknesses of PCFGs

- Strong independence assumption
  - Each production (e.g., NP -> DT NN) is independent of the rest of the tree
- Lack of sensitivity to context (where is the non-terminal in the tree, is it a subject or object)
- Lack of sensitivity to lexical information (words)

Weaknesses of PCFGs

- Lack of sensitivity to lexical information (words)





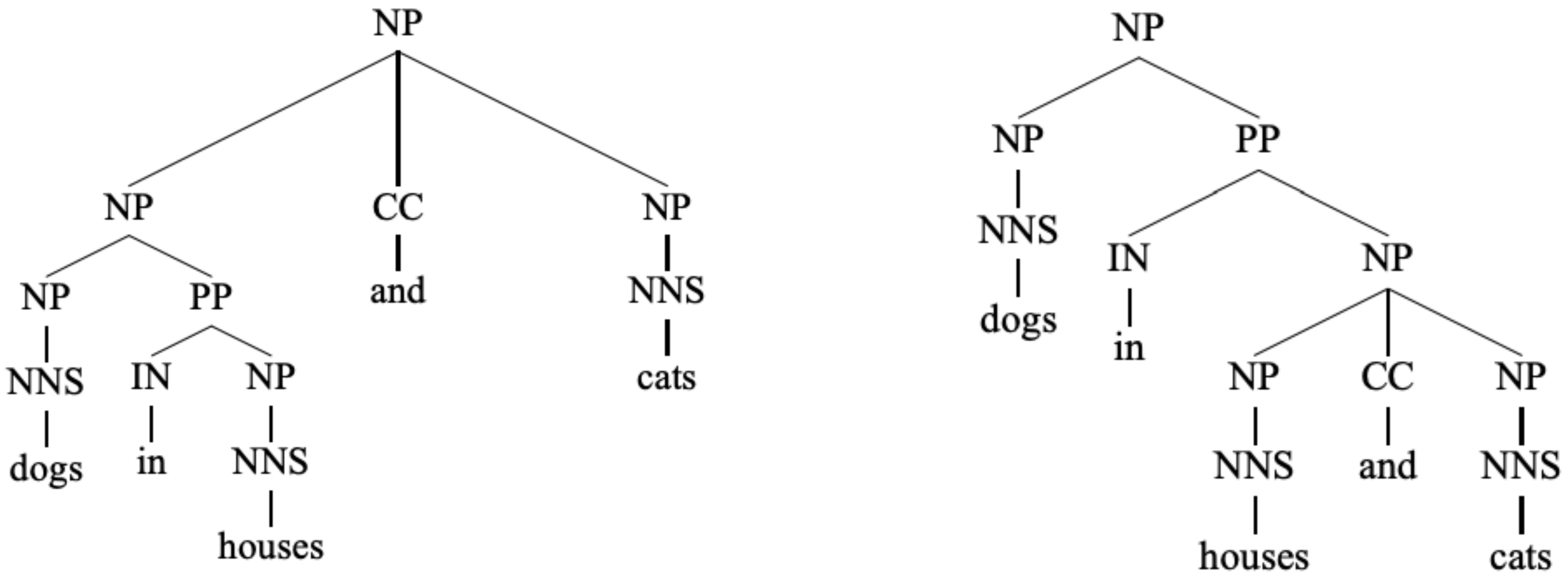
The only difference between these two parses:

$$q(\text{VP} \rightarrow \text{VP PP}) \text{ vs } q(\text{NP} \rightarrow \text{NP PP})$$

Difficult to determine the correct parse without looking at the words!

Weaknesses of PCFGs

- Lack of sensitivity to lexical information (words)

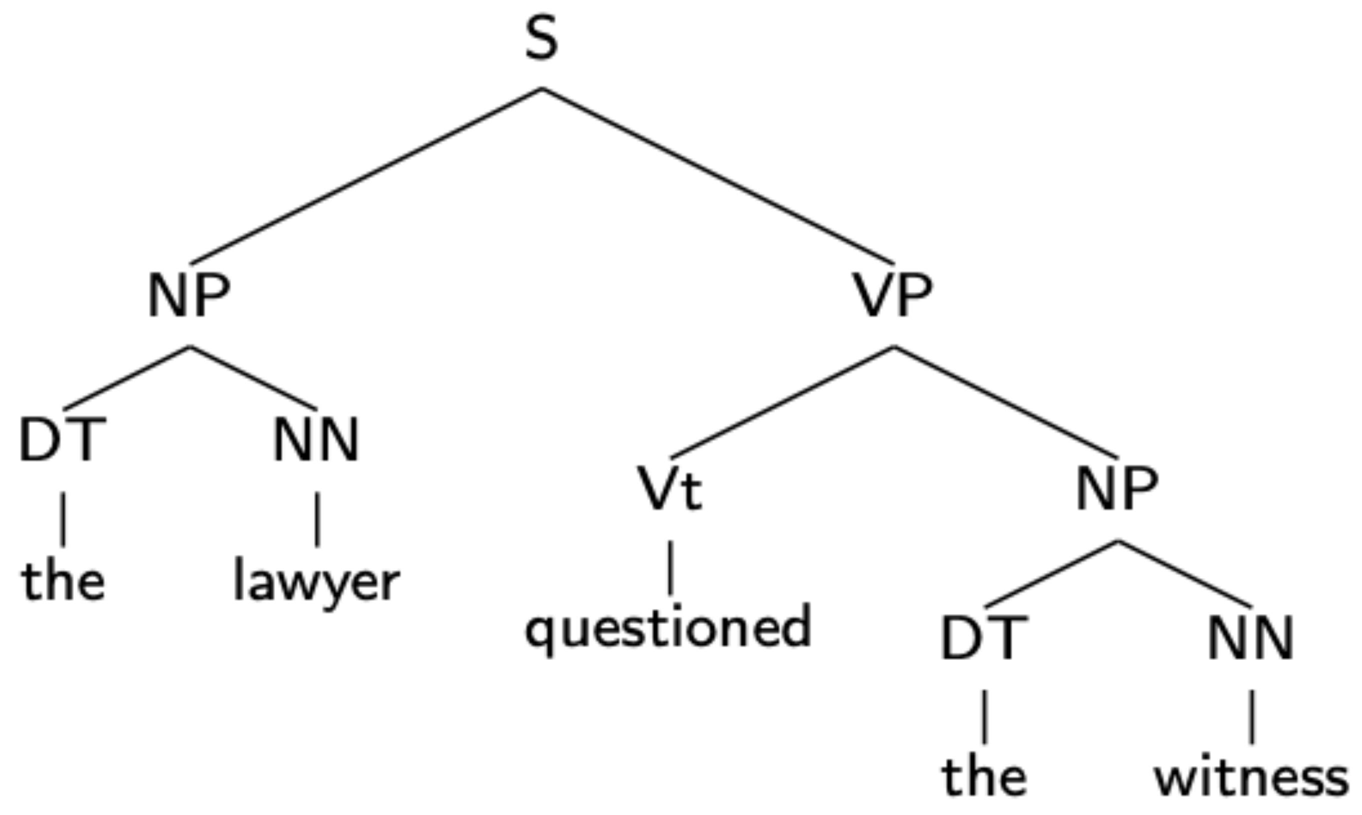


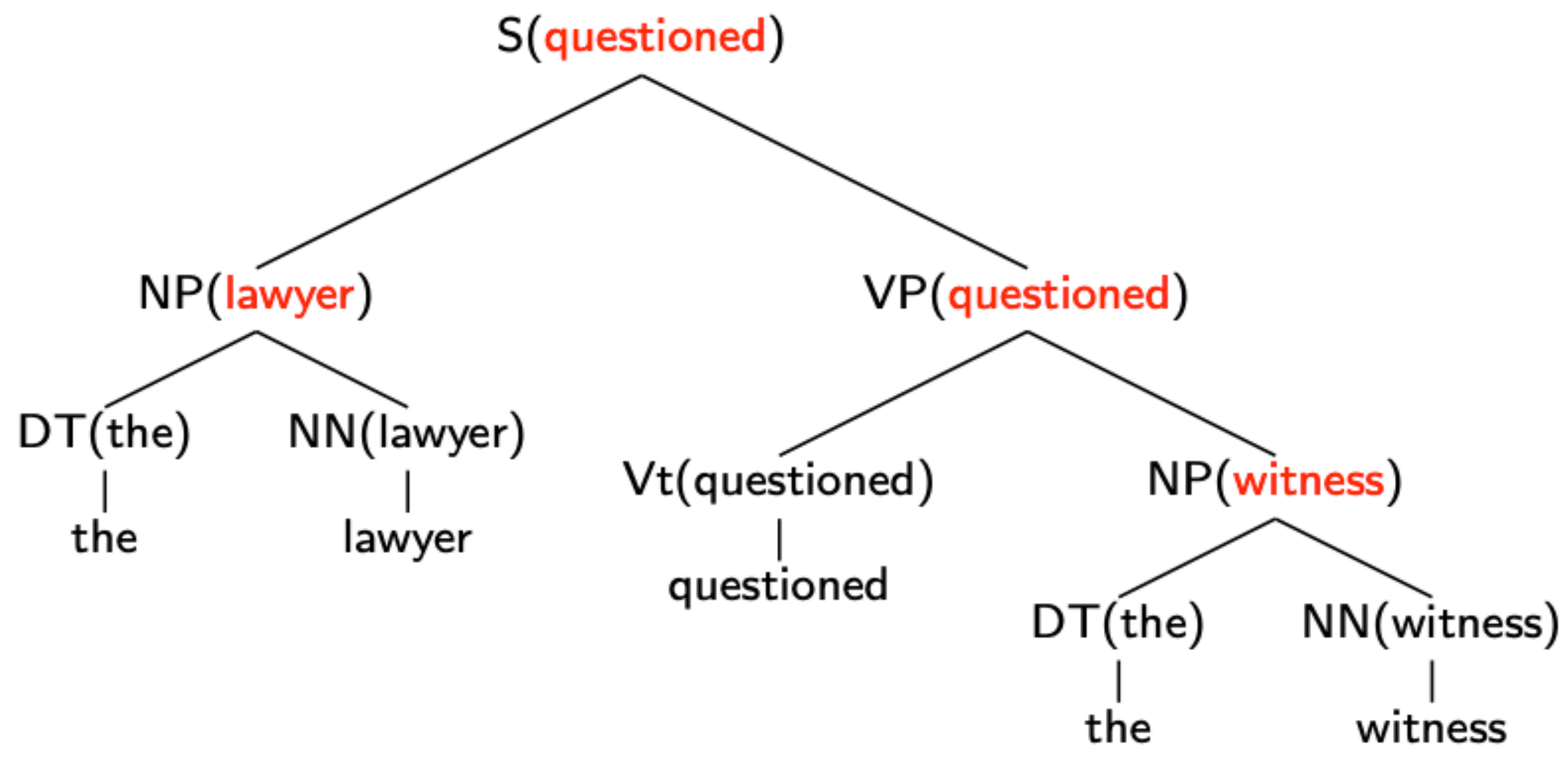
Exactly the same set of context-free rules!

Lexicalized PCFGs

- Key idea: add **headwords** to trees

Annotate parent with more information





- Each context-free rule has one special child that is the head of the rule (a core idea in syntax)

$$
\begin{array}{llllll}
\mathsf{S} & \Rightarrow & \mathsf{NP} & \mathsf{VP} & & \text{(VP is the head)} \\
\mathsf{VP} & \Rightarrow & \mathsf{Vt} & \mathsf{NP} & & \text{(Vt is the head)} \\
\mathsf{NP} & \Rightarrow & \mathsf{DT} & \mathsf{NN} & \mathsf{NN} & \text{(NN is the head)}
\end{array}
$$

Head finding rules

If the rule contains NN, NNS, or NNP:
    Choose the rightmost NN, NNS, or NNP

Else If the rule contains an NP: Choose the leftmost NP

Else If the rule contains a JJ: Choose the rightmost JJ

Else If the rule contains a CD: Choose the rightmost CD

Else Choose the rightmost child

If the rule contains Vi or Vt: Choose the leftmost Vi or Vt

Else If the rule contains a VP: Choose the leftmost VP

Else Choose the leftmost child

Lexicalized PCFGs

$$
\begin{array}{llll}
\text{S(saw)} & \rightarrow_2 & \text{NP(man)} & \text{VP(saw)} \\
\text{VP(saw)} & \rightarrow_1 & \text{Vt(saw)} & \text{NP(dog)} \\
\text{NP(man)} & \rightarrow_2 & \text{DT(the)} & \text{NN(man)} \\
\text{NP(dog)} & \rightarrow_2 & \text{DT(the)} & \text{NN(dog)} \\
\text{Vt(saw)} & \rightarrow & \text{saw} & \\
\text{DT(the)} & \rightarrow & \text{the} & \\
\text{NN(man)} & \rightarrow & \text{man} & \\
\text{NN(dog)} & \rightarrow & \text{dog} &
\end{array}
$$

Drawbacks:

- Dramatically increases the size of the grammar -> less training data for each production
- Increase the complexity of the model (running time and memory)

- Further reading: *Michael Collins. 2003. Head-Driven Statistical Models for Natural Language Parsing*.
- Results for a PCFG: 70.6% recall, 74.8% precision
- Results for a lexicalized PCFG: 88.1% recall, 88.3% precision

Further improvements to parsing

- Discriminative **reranking**
  - PCFG is a generative model
  - Use discriminative models with more global features to score parses and rerank candidate parses from the PCFG
- **Self-training** (incorporate unlabeled data)
  - Train on some data to get initial good model
  - Then run model on unlabeled data and combine newly labeled data with gold labeled data and retrain
- **Ensemble**
  - Combine multiple models

Charniak parser w/ self-train+rerank:

(McClosky et al 2006)

92.1 F1

Beyond supervised learning:

Grammar Induction = learn grammar from unlabeled data

Using Neural Networks for Constituency Parsing

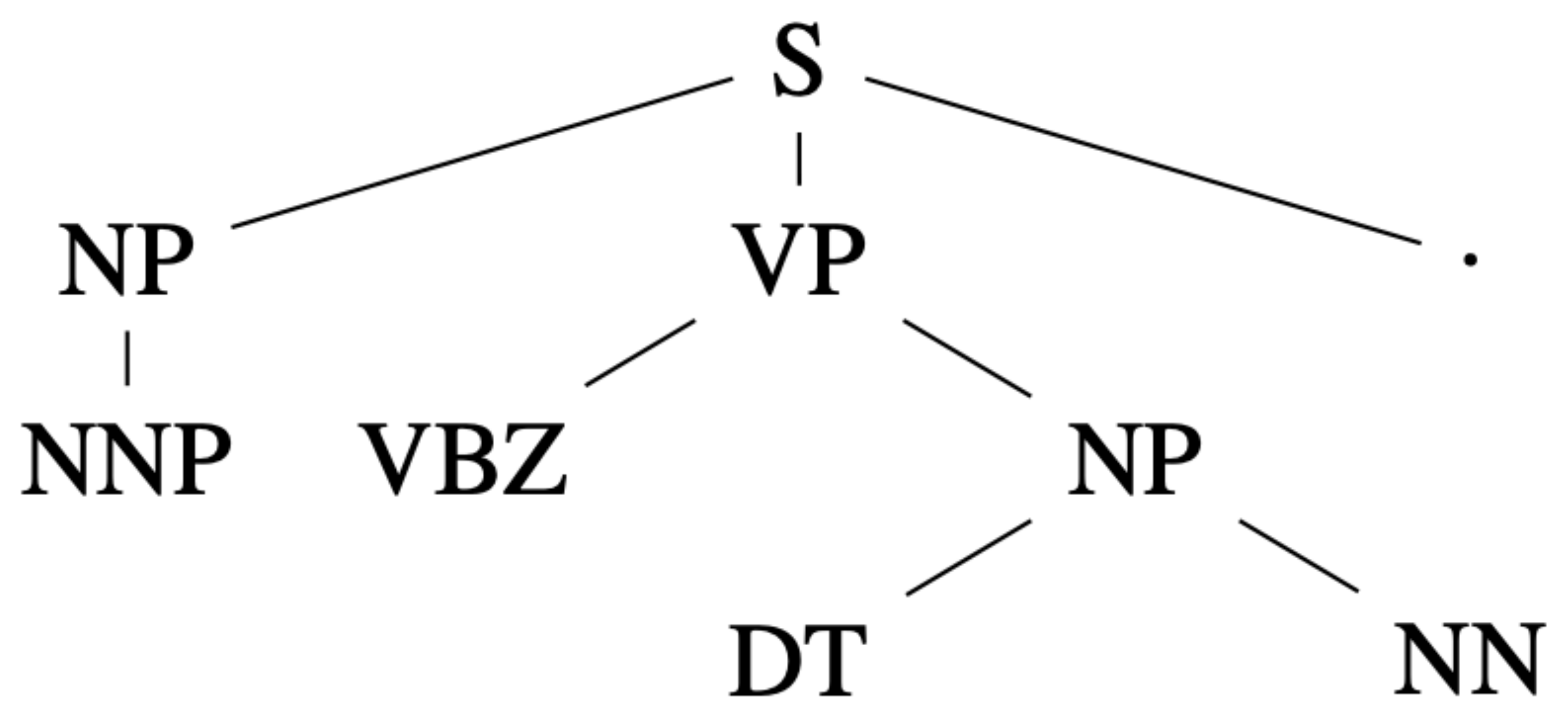
Parsing with Neural Networks

What can neural networks bring?

- Better phrase representations
  - Embeddings for words, tags, and nodes
  - Leverage pretrained embeddings
- Learned scoring functions
- Less independence assumptions

Parsing as Seq2Seq
(Vinyals et al, 2015; Vaswani et al, 2017)

John has a dog . $\rightarrow$



John has a dog . $\rightarrow$ (S (NP NNP $)_{NP}$ (VP VBZ (NP DT NN $)_{NP}$ $)_{VP}$. $)_{S}$

May not be structural correct
(i.e. unbalanced parenthesis)

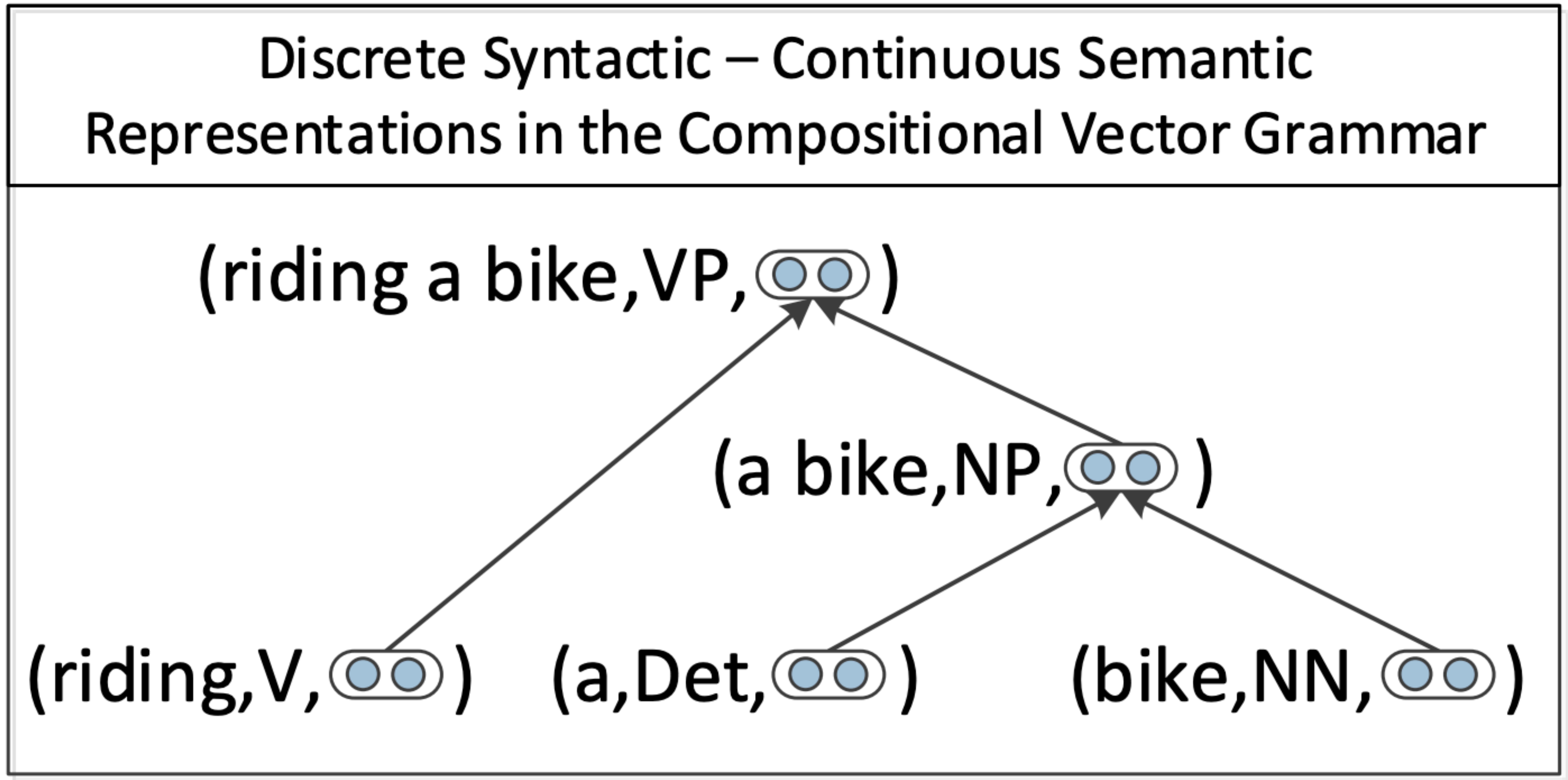
88.3 F1

91.3 F1

- Linearize parse tree and train LSTM seq2seq model with attention
- With transformers

Recursive Neural Networks (Socher et al, 2013)

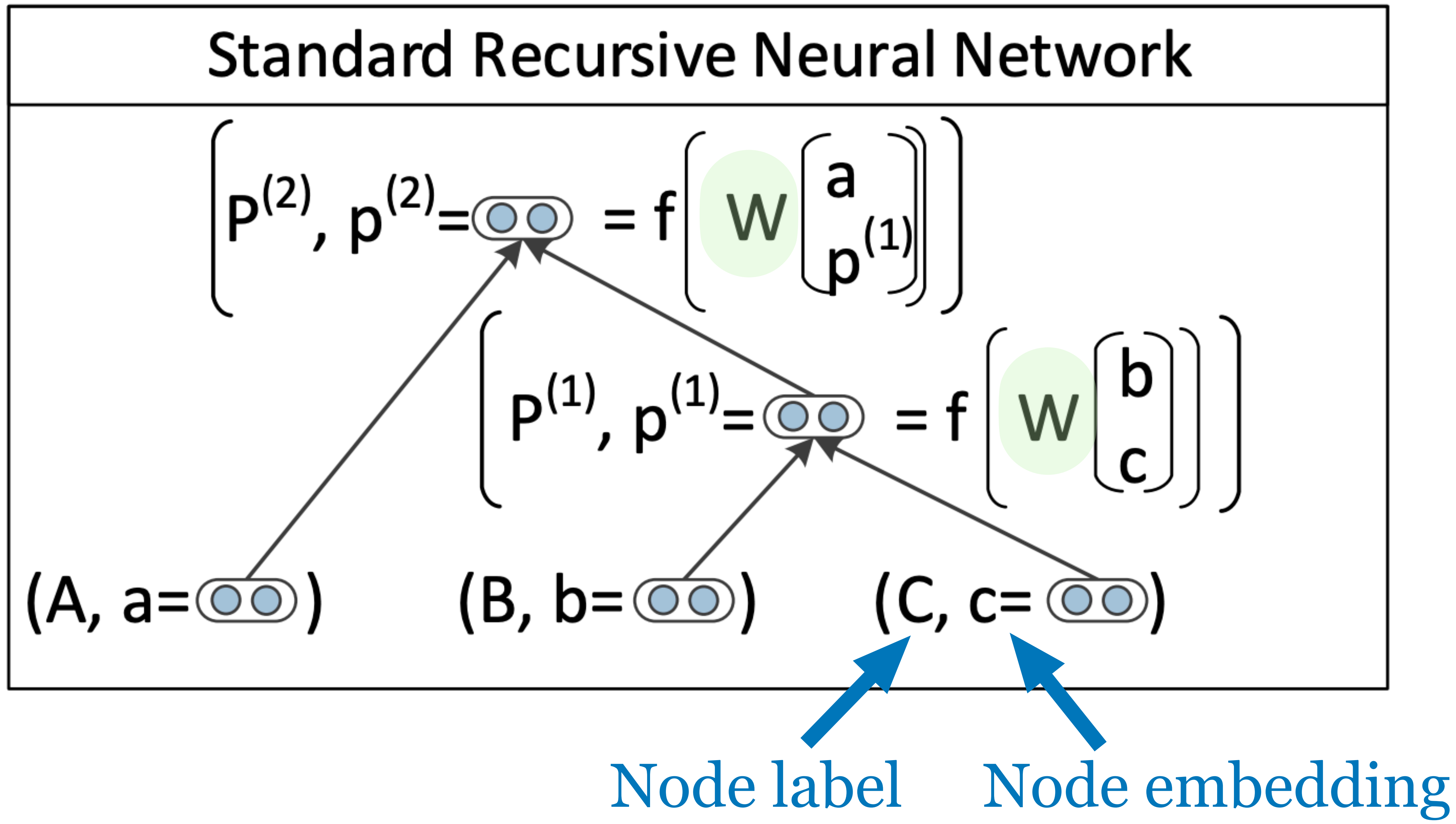
- Continuous representations for words and non-terminal nodes
- Compositional representations for non-terminal nodes
- Use neural networks to get compositional representations as well as scores for composition

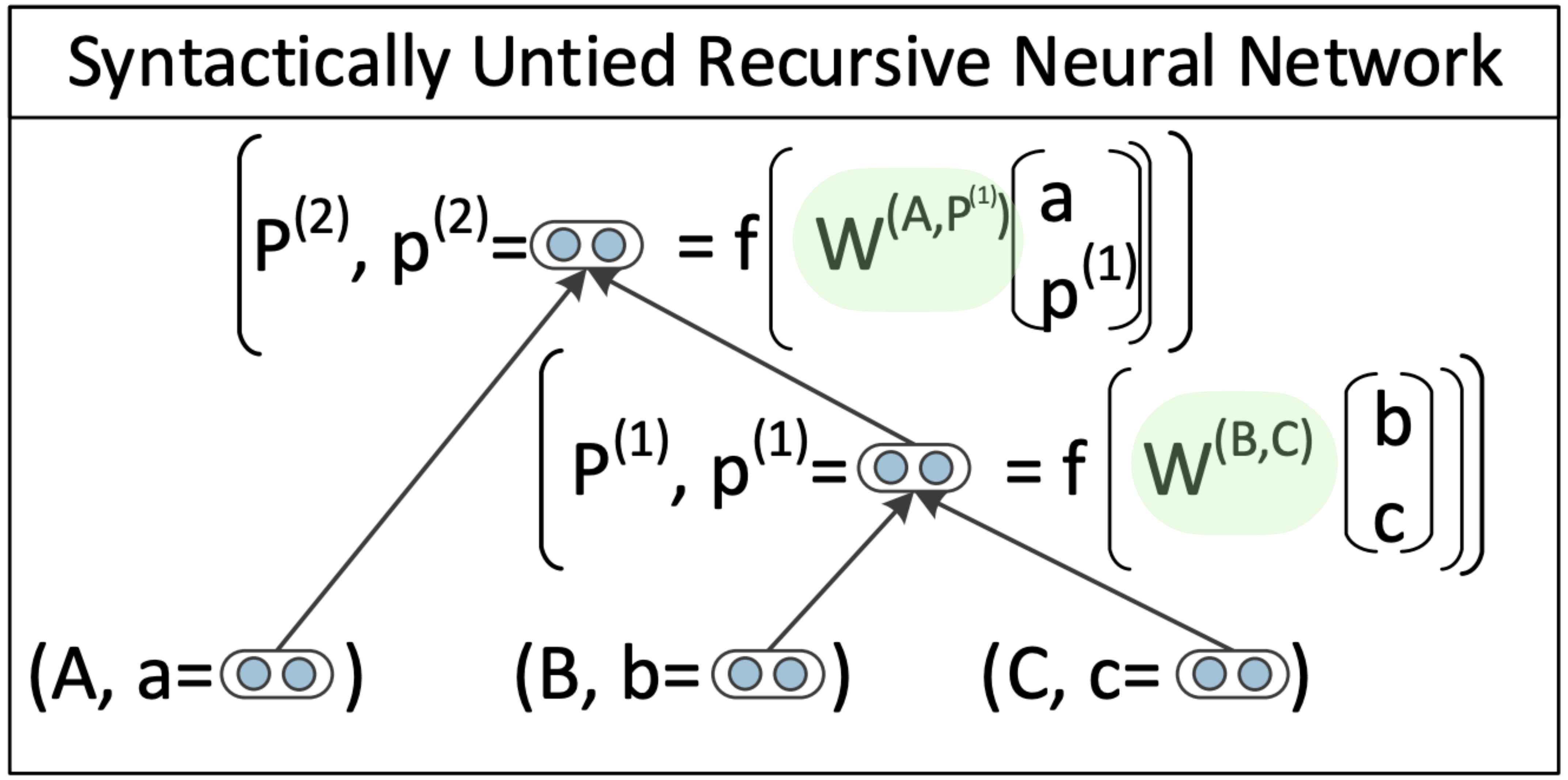


Compositional Vector Grammar = PCFG + TreeRNN

Recursive Neural Networks (Socher et al, 2013)

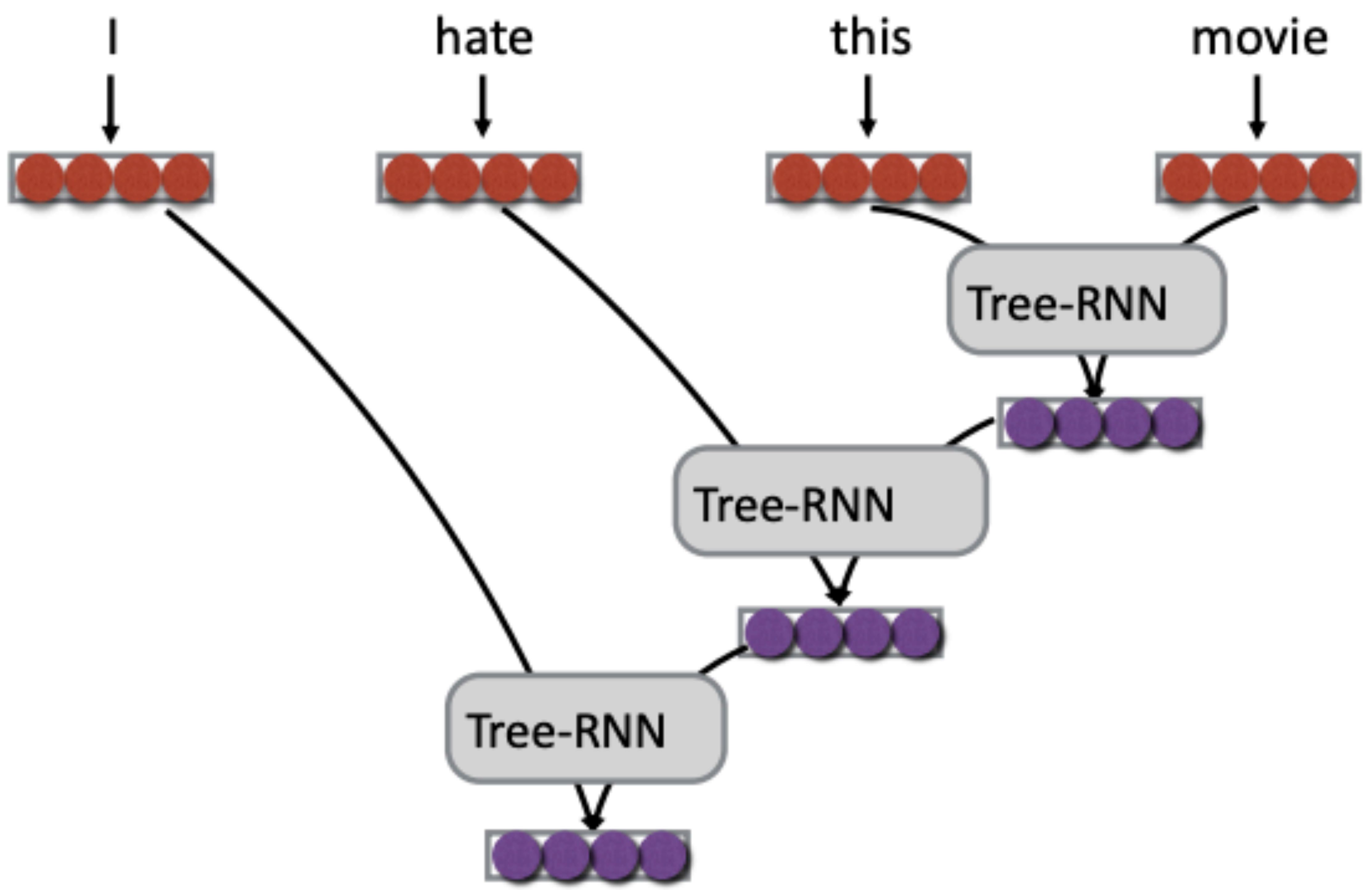
Weights depend on discrete category of children (NP, VP)





Weights can be tied or parameterized by constituency type

Recursive Neural Networks (Socher et al, 2013)



$$\text{tree-rnn}(\boldsymbol{h}_1, \boldsymbol{h}_2) = \tanh(W[\boldsymbol{h}_1; \boldsymbol{h}_2] + \boldsymbol{b})$$

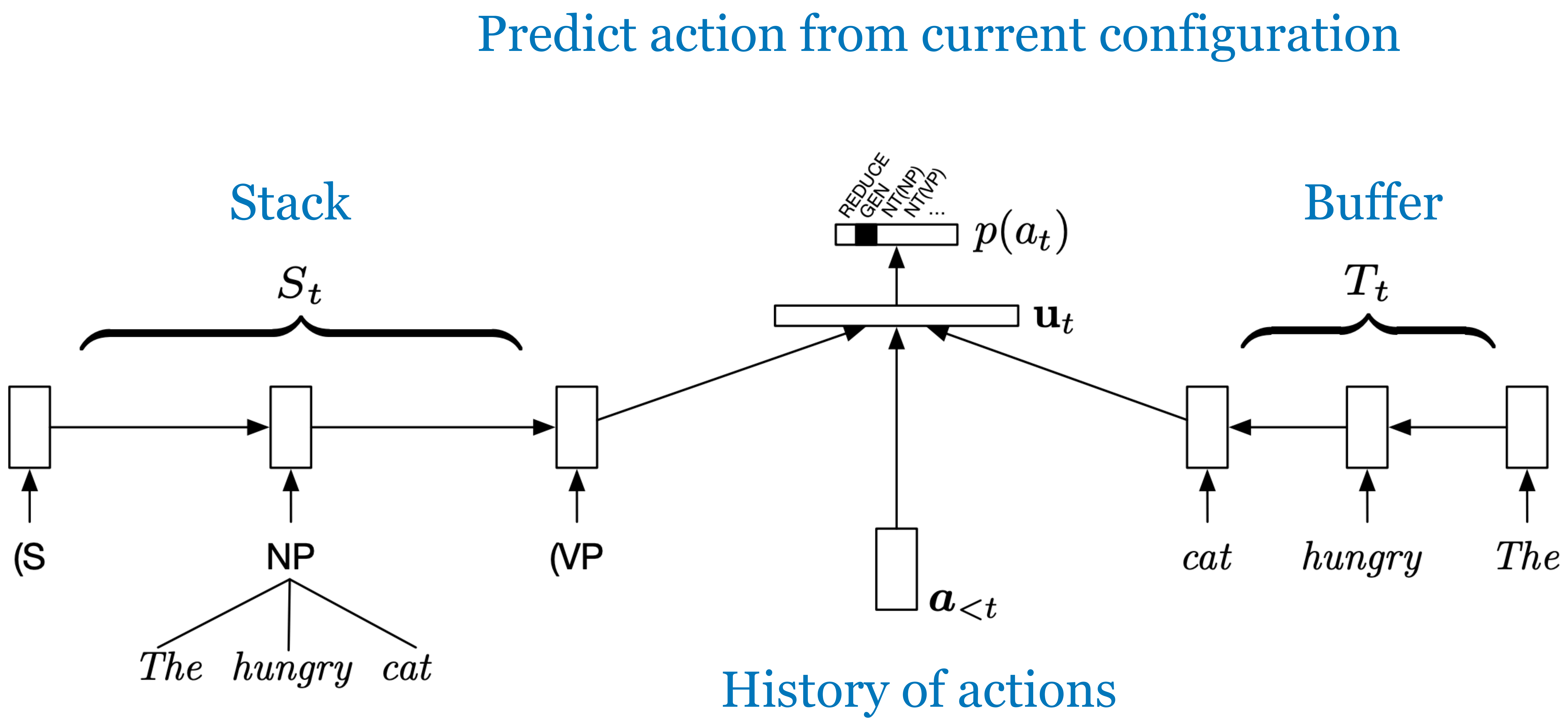
90.4 F1

(figure credit: CMU CS 11-747, Graham Neubig)

Recurrent Neural Network Grammars (Dyer et al, 2016)

Transition Parsers

- Like Seq2Seq but output is a sequence of operations that builds the tree incrementally
- The sequence can guarantee structural consistency



Recurrent Neural Network Grammars (Dyer et al, 2016)

S: stack of open nonterminals and completed subtrees

B: buffer of unprocessed terminal symbols

x: terminal symbol

X: Non-terminal symbol

τ: completed subtree

Parser transitions

Before action / After action

Stack$_t$	**Buffer$_t$**	**Open NTs$_t$**	**Action**	**Stack$_{t+1}$**	**Buffer$_{t+1}$**	**Open NTs$_{t+1}$**
S	B	n	NT(X)	$S \mid$ (X	B	$n+1$
S	$x \mid B$	n	SHIFT	$S \mid x$	B	n
$S \mid$ (X $\mid \tau_1 \mid \ldots \mid \tau_\ell$	B	n	REDUCE	$S \mid$ (X $\tau_1 \ldots \tau_\ell$)	B	$n-1$

Top-down parsing

Input: *The hungry cat meows .*

	Stack	**Buffer**	**Action**
0		*The* | *hungry* | *cat* | *meows* | .	NT(S)
1	(S	*The* | *hungry* | *cat* | *meows* | .	NT(NP)
2	(S | (NP	*The* | *hungry* | *cat* | *meows* | .	SHIFT
3	(S | (NP | *The*	*hungry* | *cat* | *meows* | .	SHIFT
4	(S | (NP | *The* | *hungry*	*cat* | *meows* | .	SHIFT
5	(S | (NP | *The* | *hungry* | *cat*	*meows* | .	REDUCE
6	(S | (NP *The hungry cat*)	*meows* | .	NT(VP)
7	(S | (NP *The hungry cat*) | (VP	*meows* | .	SHIFT
8	(S | (NP *The hungry cat*) | (VP *meows*	.	REDUCE
9	(S | (NP *The hungry cat*) | (VP *meows*)	.	SHIFT
10	(S | (NP *The hungry cat*) | (VP *meows*) | .		REDUCE
11	(S (NP *The hungry cat*) (VP *meows*) .)		

Actions:

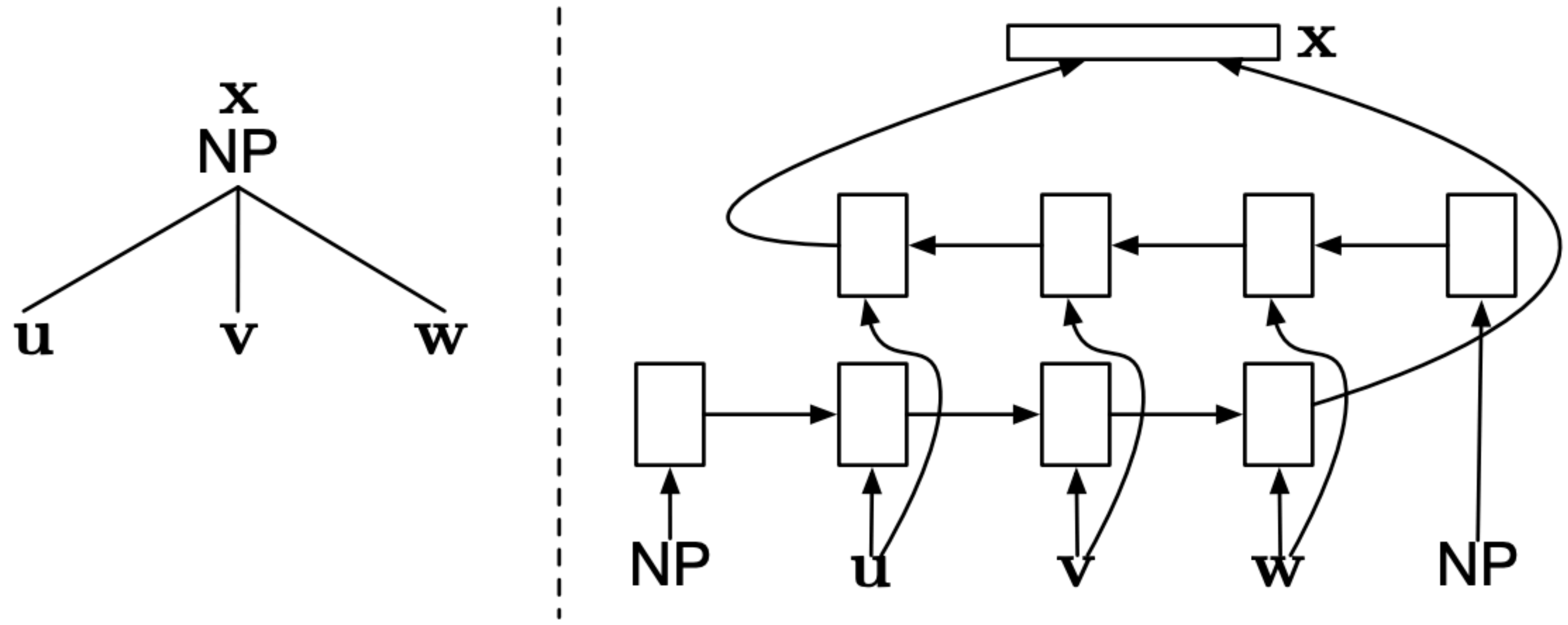
NT(X): Open (create) a new non-terminal of type X

SHIFT: move x from buffer to stack

REDUCE: Close(finish) open non-terminal on stack

Recurrent Neural Network Grammars (Dyer et al, 2016)

REDUCE

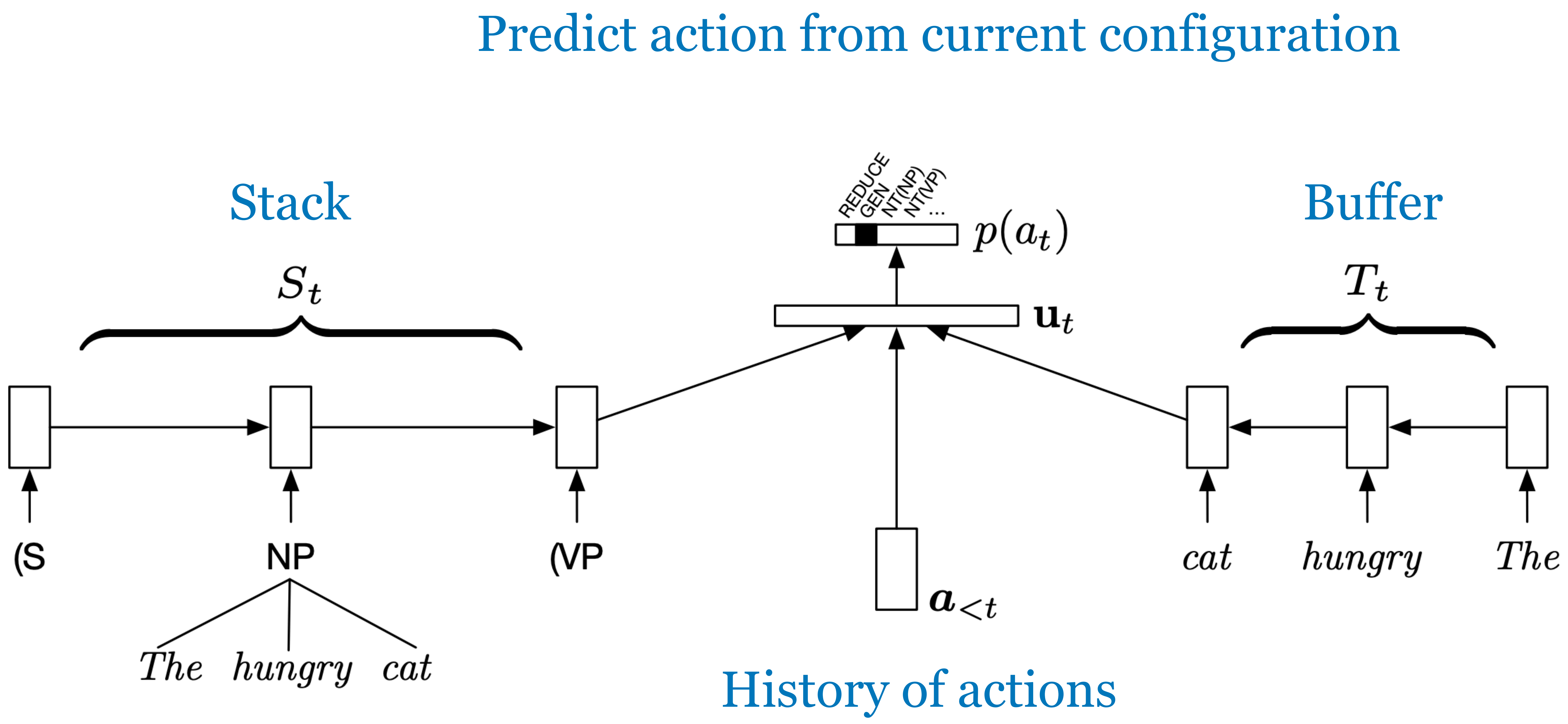


- BiLSTM to get composite representation of non-terminal

Recurrent Neural Network Grammars (Dyer et al, 2016)

Transition Parsers

- Like Seq2Seq but output is a sequence of operations that builds the tree incrementally
- The sequence can guarantee structural consistency



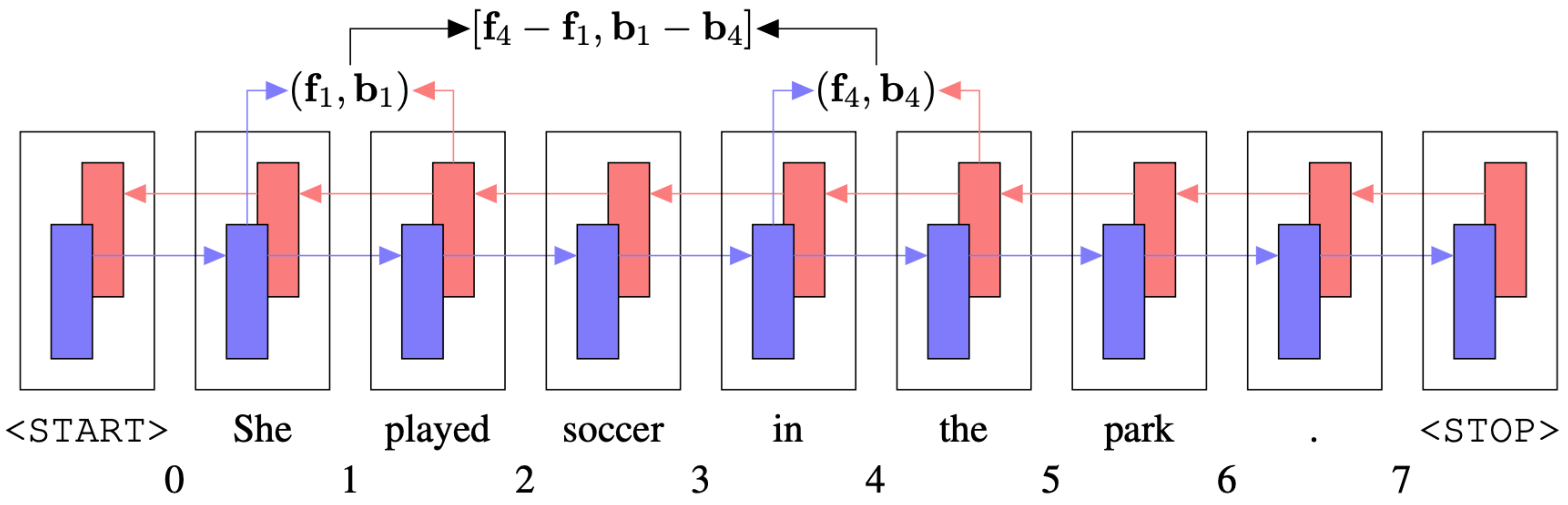
91.2 F1

Span Labeling
(Stern et al. 2017)

- Simple idea: decide whether span is constituent in tree or not
- Scores labels and spans independently
- Allows for various loss functions (local vs structured), inference algorithms (CKY vs topdown)

- Word representation
- Span representation
- Label scoring

Span Labeling (Stern et al. 2017)



(Gaddy et al, 2018)

- Bidirectional LSTM to get forward/backward encodings (f_i, b_i) for position i
- Span (i, j) representation: concat vector differences $[f_j - f_i, b_i - b_j]$
- Feedforward neural networks to predict scores for labels and spans

$$s_{\text{labels}}(i,j) = \mathbf{V}_l g(\mathbf{W}_l \mathbf{s}_{ij} + b_l)$$ vector

$$s_{\text{label}}(i,j,l) = l \text{ th element of } s_{\text{labels}}$$

$$s_{\text{span}}(i,j) = \mathbf{v}_s^\top g(\mathbf{W}_s \mathbf{s}_{ij} + b_s)$$ scalar

Span Labeling (Stern et al. 2017)

Running time?

$O(n^2)$

Greedy **top down parsing**

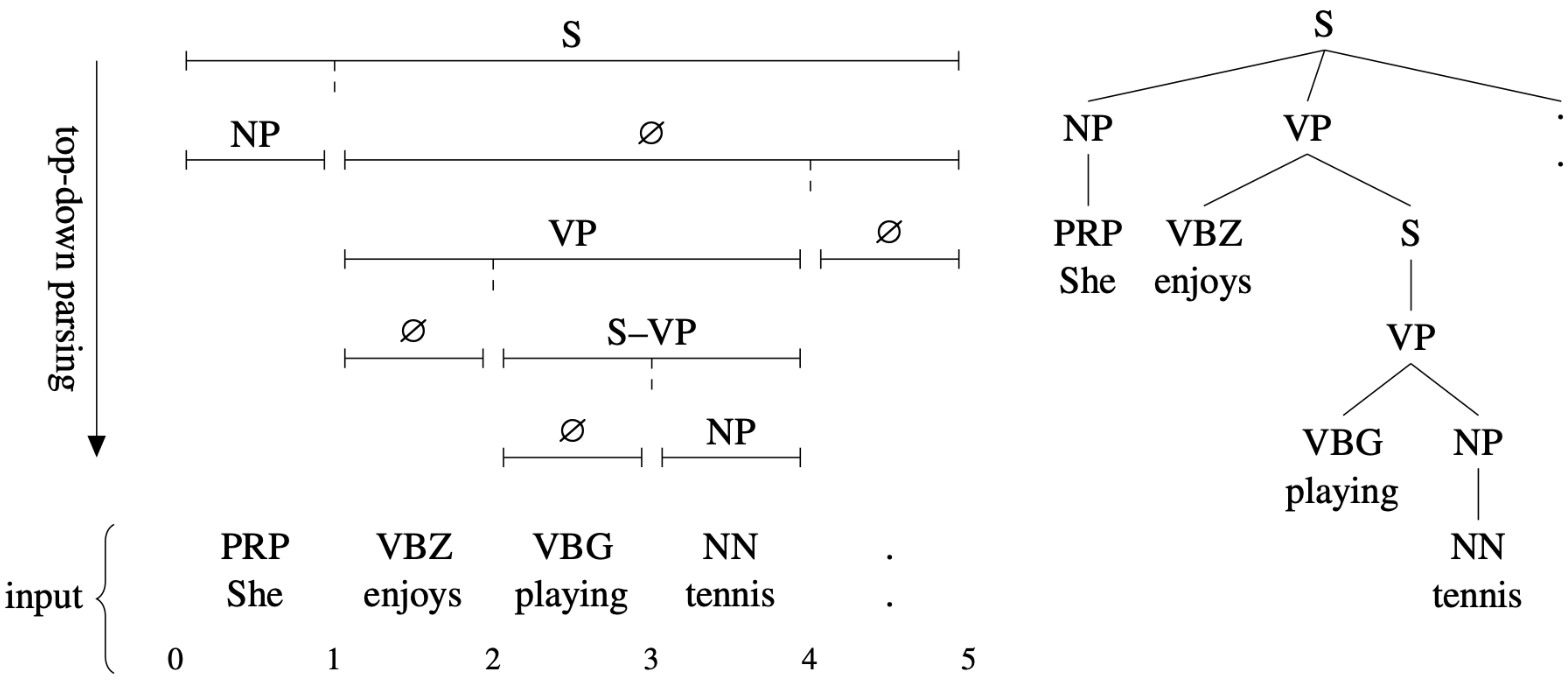
- Recursively for each span:
  - Assign a label
  - Pick a split point

$$\hat{l} = \arg\max_k S_{\text{label}}(i, j, l)$$

$$\hat{k} = \arg\max_k S_{\text{split}}(i, k, j)$$

$$S_{\text{span}}(i, k) + S_{\text{span}}(k, j)$$

91.8 F1

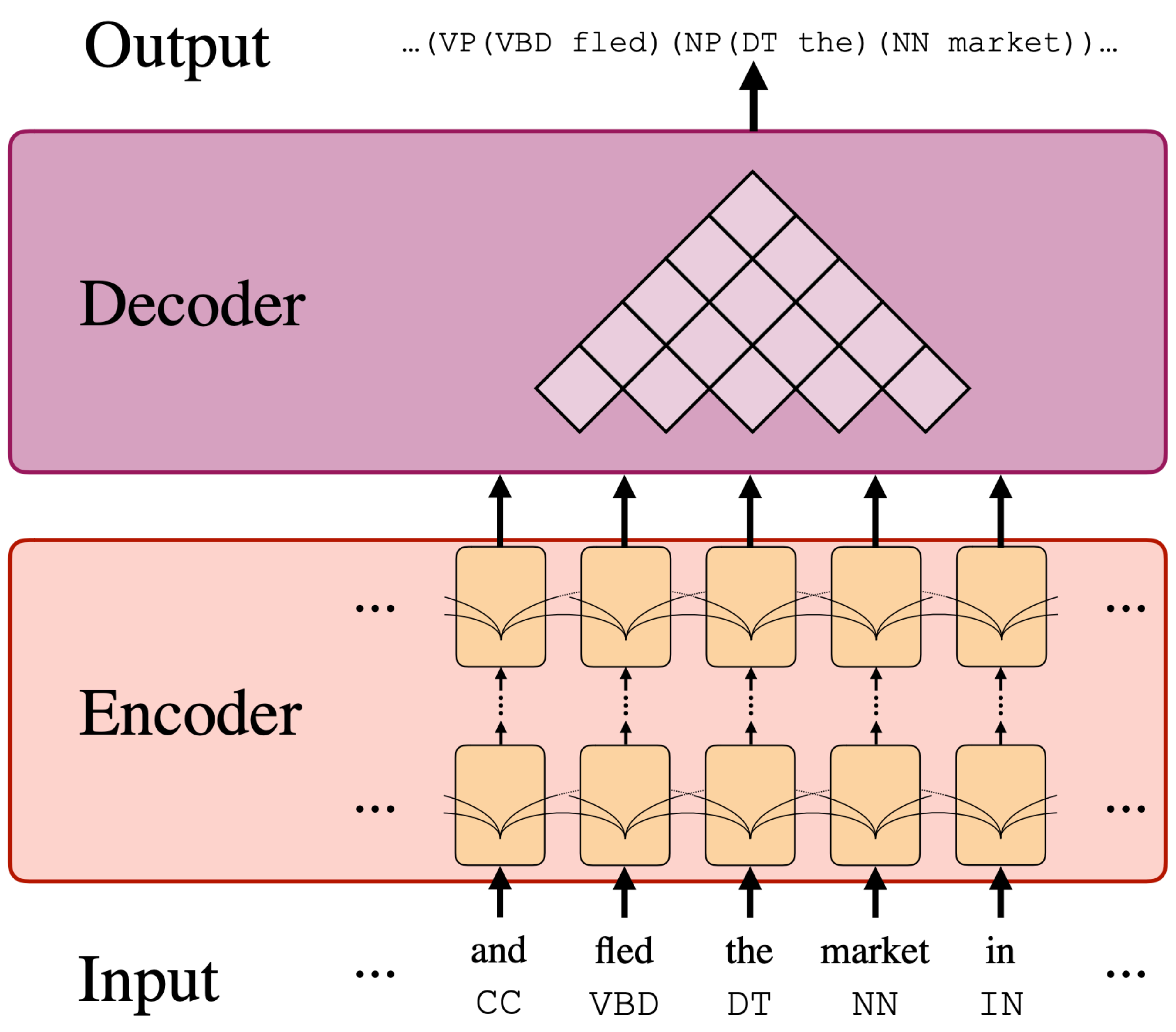


(a) Execution of the top-down parsing algorithm.

(b) Output parse tree.

Self-Attentional Encoding (Kitaev and Klein, 2018)

- Self-attention based encoding
- Learned scoring $s(i, j, l)$ function for each span from token i to token j with label l
- CKY for decoding to find the best tree
- Berkeley neural parser: https://github.com/nikitakit/self-attentive-parser

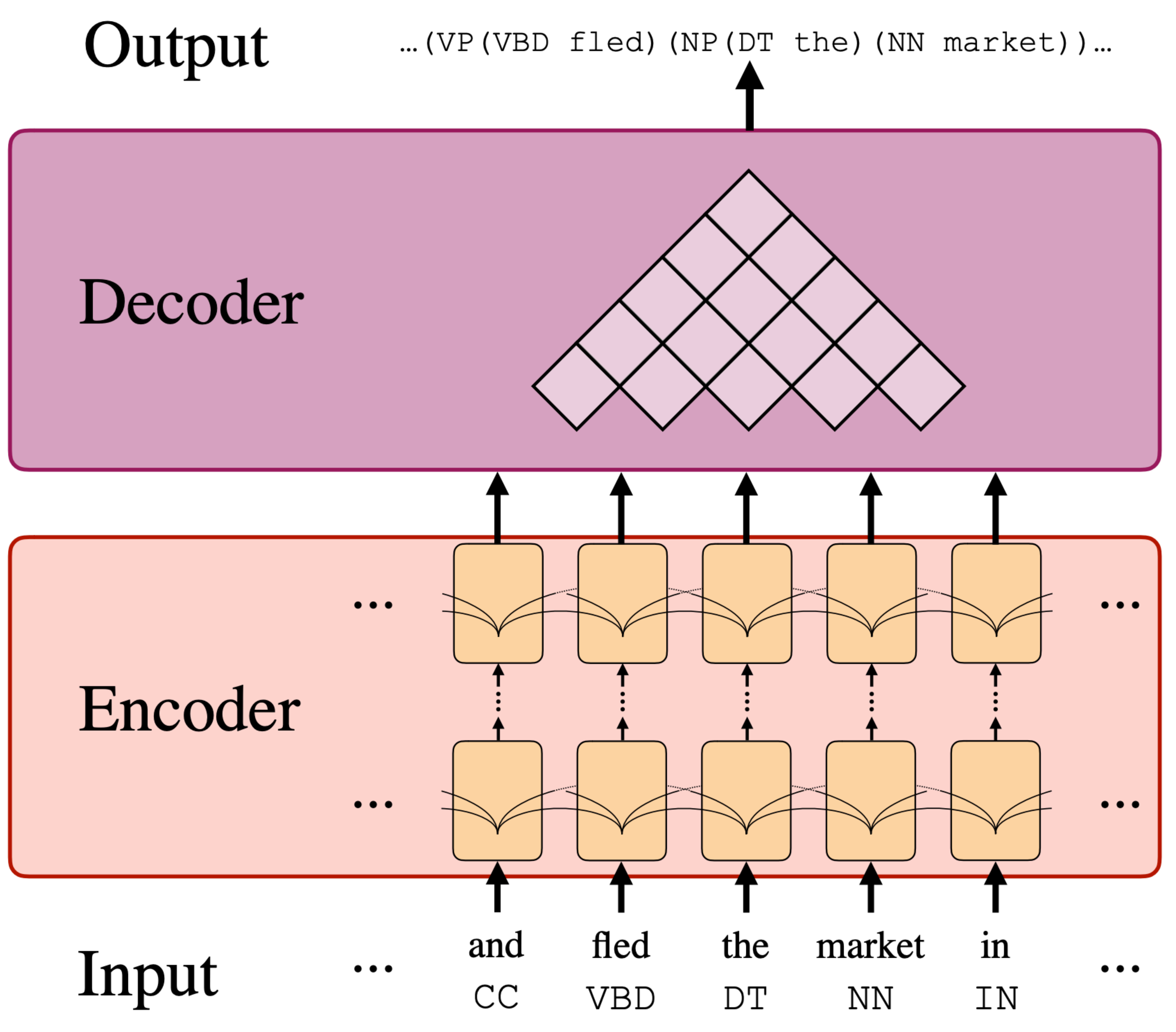


93.6 F1

Self-Attentional Encoding (Kitaev and Klein, 2018)

- Improvements with pretrained representations

F1
93.6 (no pretraining)
93.7 (w/ FastText)
95.2 (w/ ELMo),
95.7 (w/ BERT LARGE cased),
95.8 (Ensemble w/ BERT BASE/LARGE, cased/uncased



LLMs for constituency parsing

Provide a constituency parse for "The dog and cat enjoy playing together"

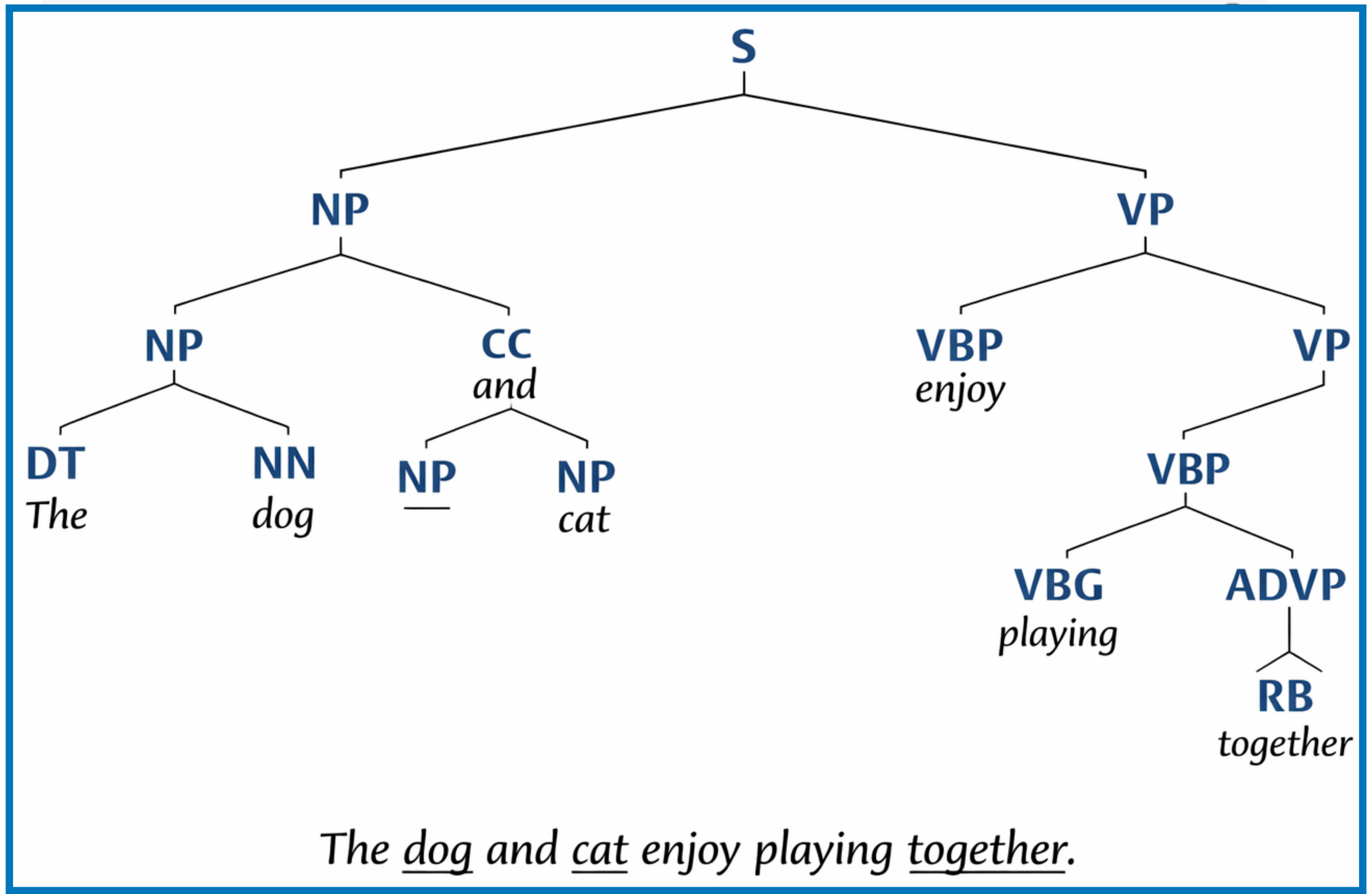
ChatGPT (Mar 2026)

A **constituency (phrase structure) parse** groups words into hierarchical phrases such as noun phrases (NP) and verb phrases (VP).

For the sentence:

> **"The dog and cat enjoy playing together."**

Bracketed Constituency Parse



Key Points

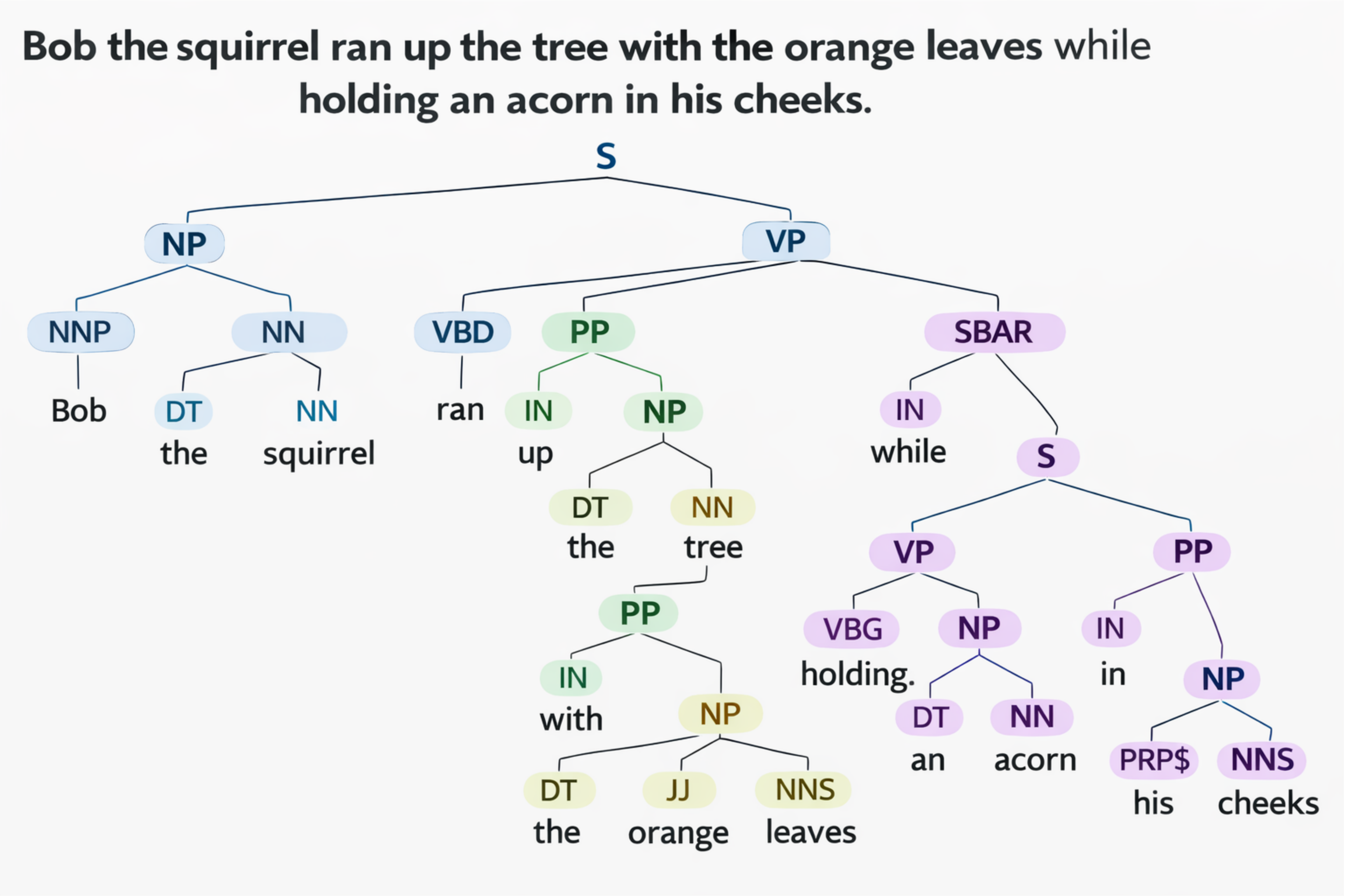
- **S**: Sentence
- **NP**: Subject noun phrase with coordination (*dog and cat*)
- **CC**: Coordinating conjunction (*and*)
- **VP**: Verb phrase headed by **enjoy**
- **Embedded VP**: *playing together* functioning as the complement of *enjoy*
- **ADVP**: Adverbial phrase modifying *playing*

If you'd like, I can also:

- Show the **Penn Treebank style** parse exactly as produced by common parsers, or
- Generate a **visual parse tree diagram** suitable for slides or teaching.

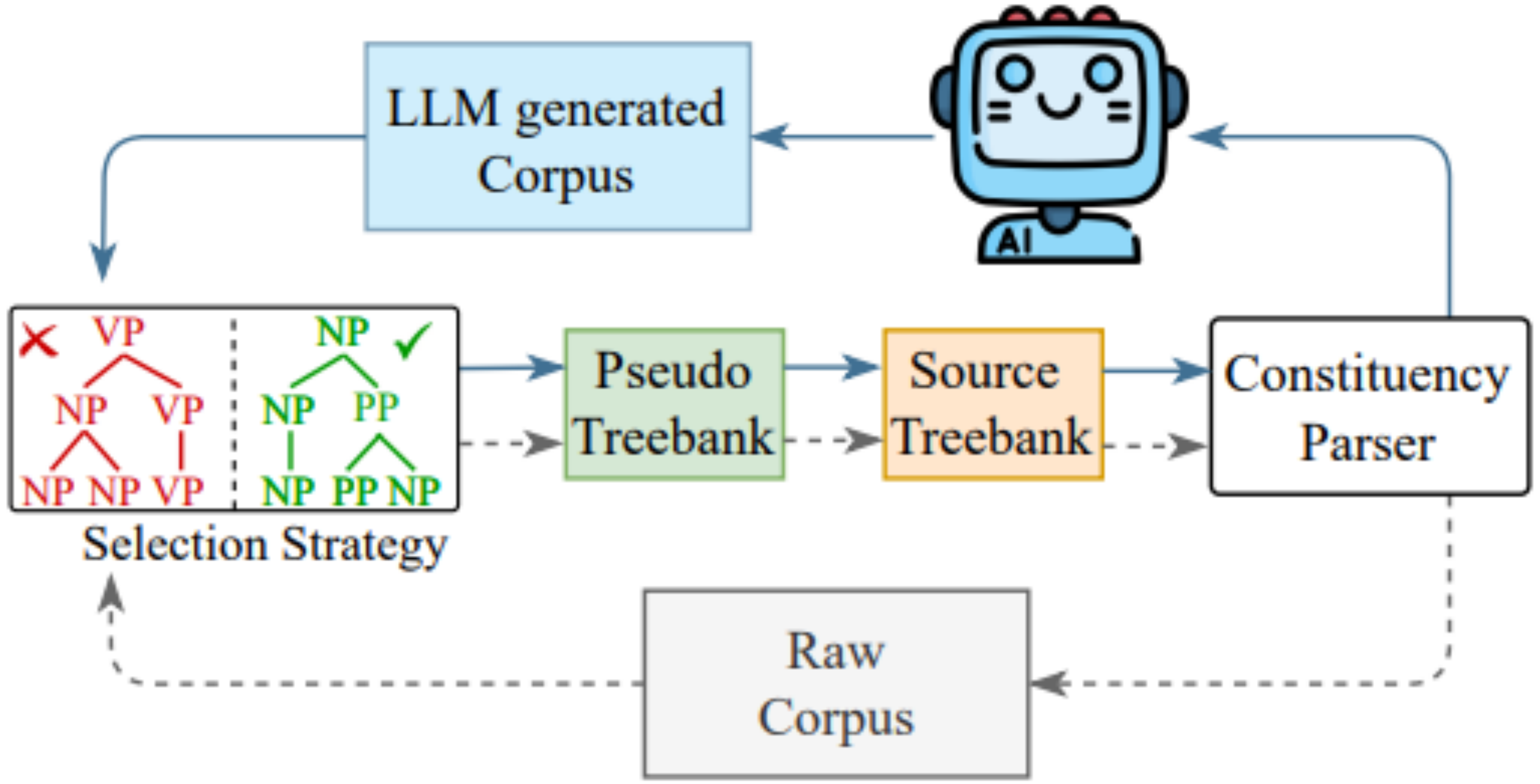
LLMs for constituency parsing

Provide a constituency parse for "Bob the squirrel ran up the tree with the orange leaves while holding an acorn in his cheeks."



LLMs for constituency parsing

- Use LLM to generate training data
  - Given grammar rules and examples, generate parse and sentences



- Select useful samples to add to the corpus

LLM Prompt

As a language assistant, you excel at creating sentences of a specific length while adhering to N grammar rules provided above. Please consider the M examples, generate one sentence of $L1 \sim L2$ words:

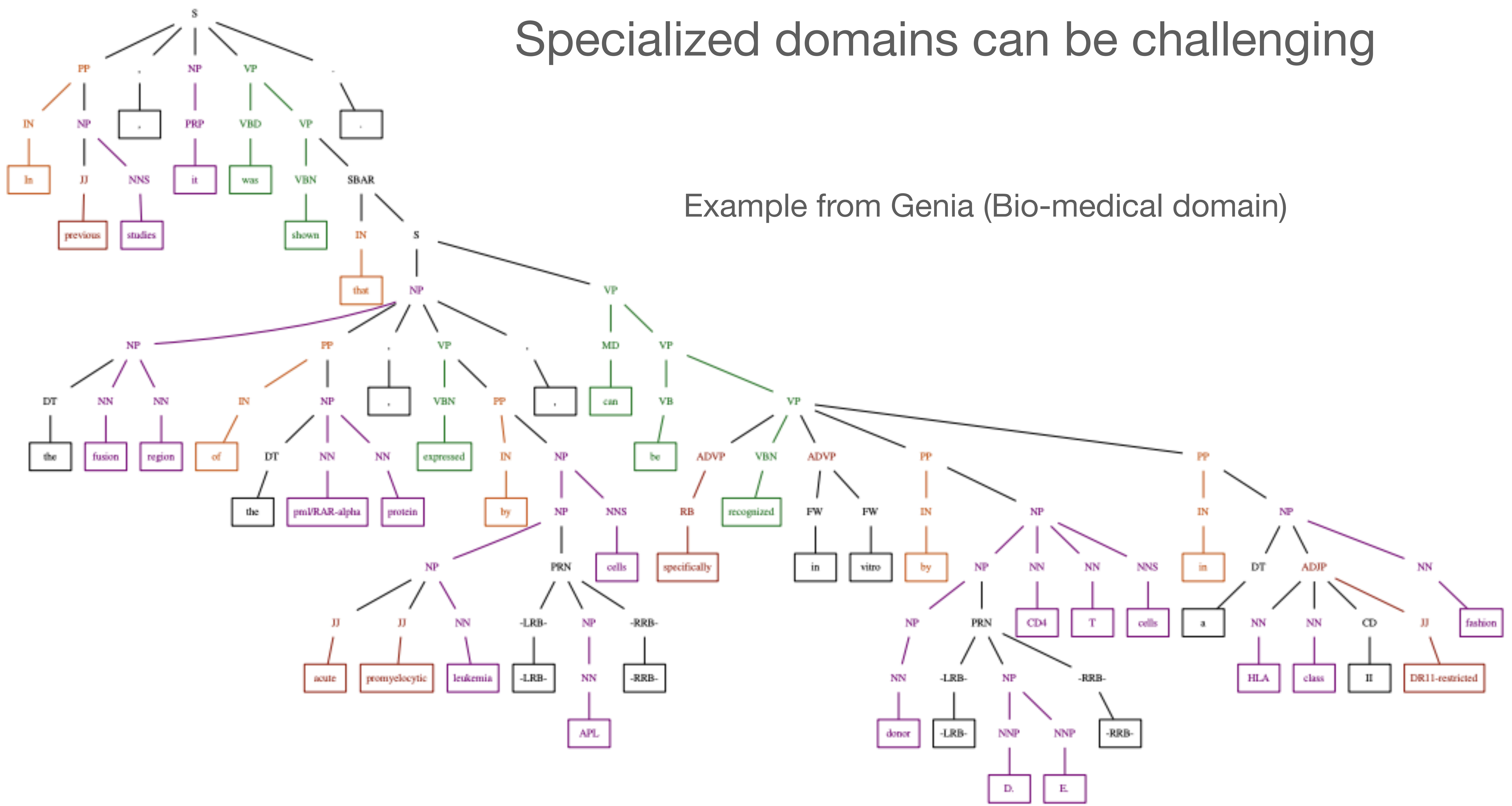
GRs: S→PP NP VP, PP→IN NP, VP→VP NP, ...

Snts: 1. History brings inspiration to peace . 2. It is the jack of all trades and master of none . 3. ... ...

- Used for "cross-domain" parsing
  - adapt model trained on a source domain to a target domain

Specialized domains can be challenging

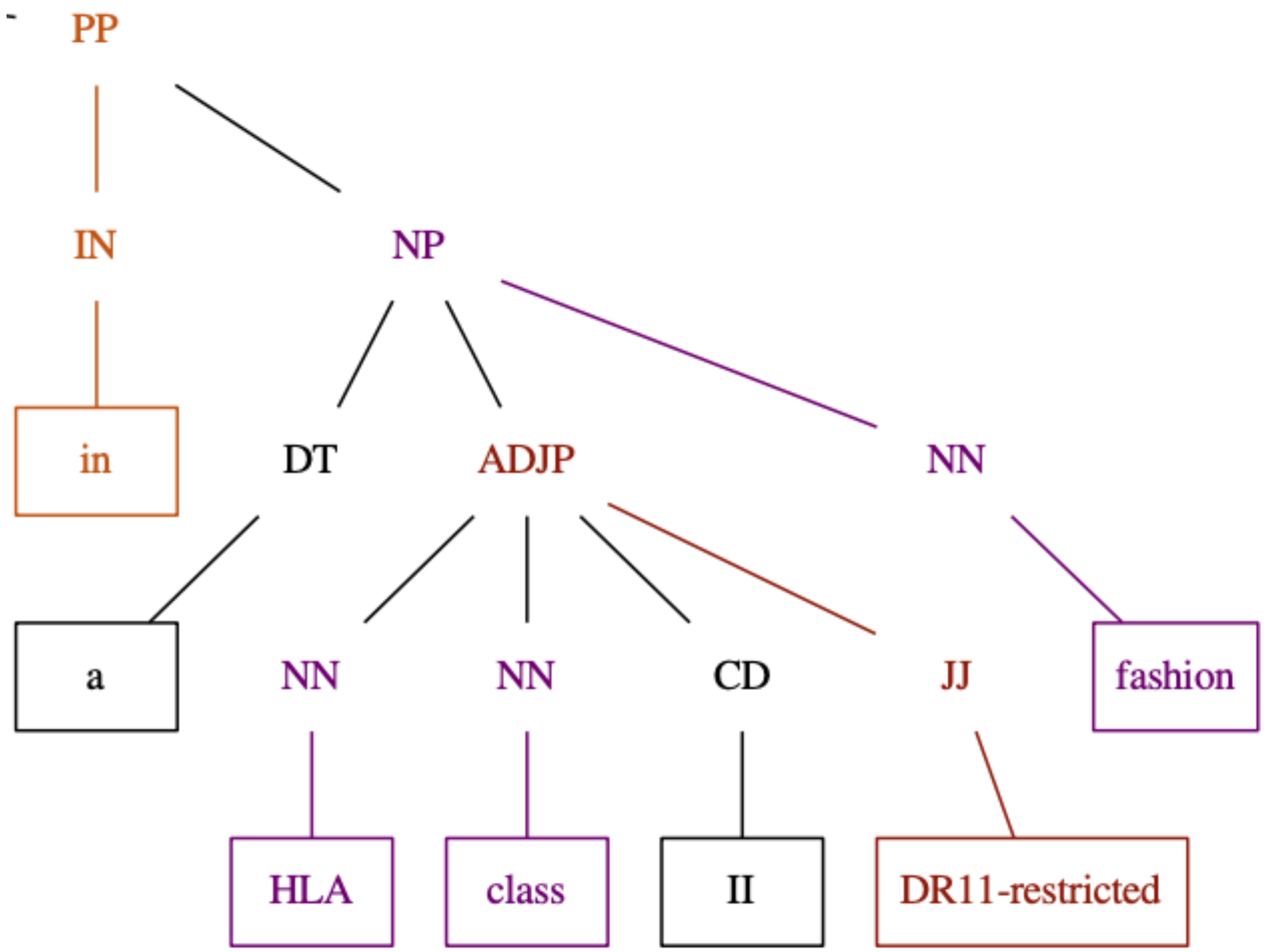
Example from Genia (Bio-medical domain)



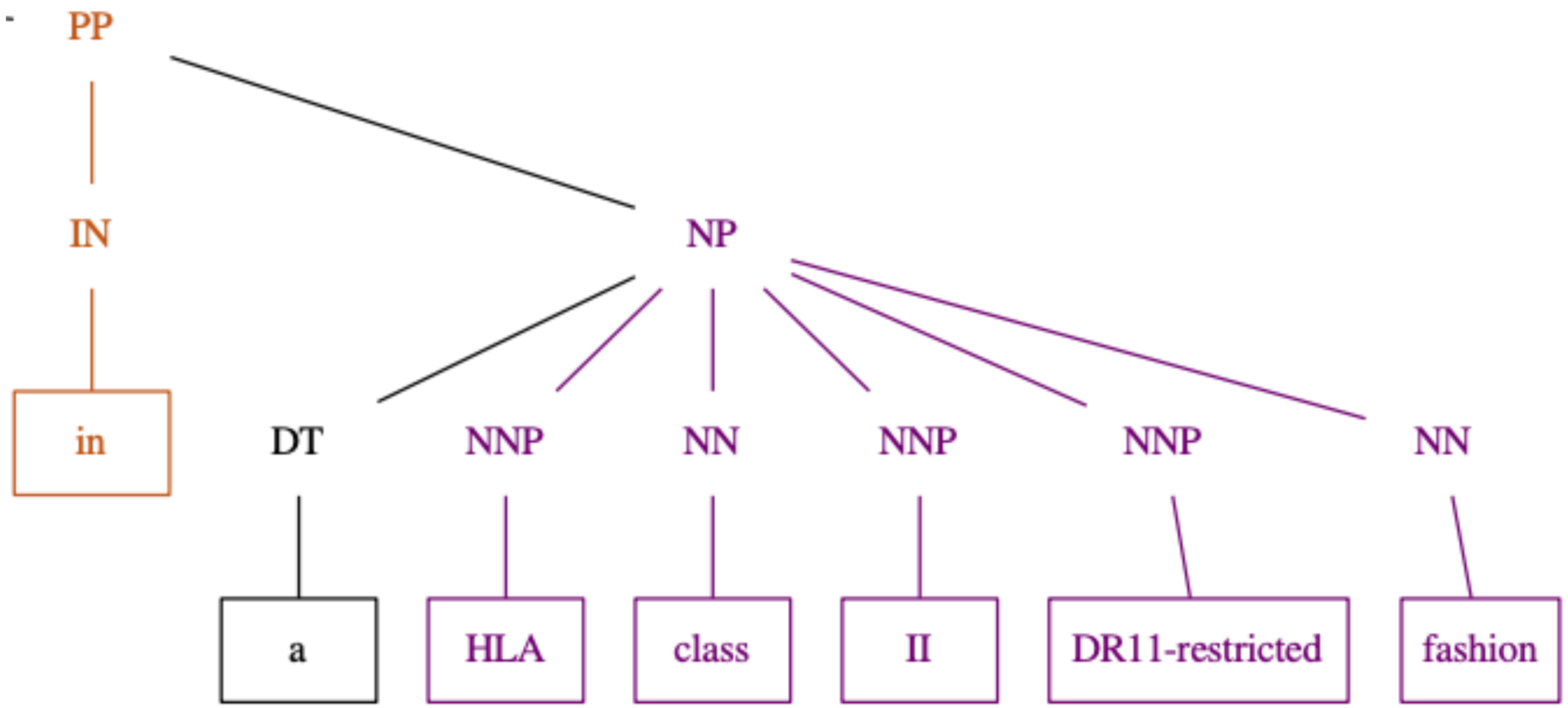
[Challenges to Open-Domain Constituency Parsing](#) [Yang et al. ACL Findings 2022]

Specialized terminology can be challenging

Gold annotation



Predicted



Challenges to Open-Domain Constituency Parsing [Yang et al. ACL Findings 2022]

LLMs for constituency parsing

Method	Criteria	Dialogue	Forum	Law	Literature	Review	Avg
gpt-3.5-turbo	-	70.70*42.4%	71.56*21.2%	80.72*27.6%	72.83*11.4%	71.24*36.7%	73.41*27.9%
Kitaev and Klein (2018)	-	85.92	86.00	91.71	85.04	83.51	86.44
Vanilla ST	Token	86.05	86.52	91.75	85.53	83.96	86.77
	Conf	86.03	86.54	91.94	85.60	83.93	86.81
	GRs	86.04	86.49	91.92	85.63	83.94	86.80
	GRsConf	86.13	86.57	91.93	85.76	84.02	86.88
LLM-enhanced ST	Token	86.01	86.52	91.69	85.55	83.92	86.74
	Conf	86.26	86.67	92.00	85.90	84.07	86.98
	GRs	86.18	86.82	91.97	86.00	84.09	87.01
	GRsConf ‡	**86.71**	**87.10**	**92.30**	**86.17**	**84.34**	**87.32**
Liu and Zhang (2017)* †	-	85.56	86.33	91.50	84.96	83.89	86.45
Kitaev and Klein (2018)* †	-	86.30	87.04	92.06	86.26	84.34	86.20
LLM-enhanced ST*	GRsConf ‡	**87.59**	**87.55**	**93.29**	**87.54**	**85.58**	**88.31**

†BERT Large

LLM-enhanced Self-training for Cross-domain Constituency Parsing [Li et al. EMNLP 2023]

Summary

- Two types of structured representations: constituency vs dependency
- Formalism for context free grammars (CFG) and probabilistic context free grammars (PCFGs)
  - CFGs have terminals (leafs), non-terminals, and production rules
  - PCFGs are CFGs with probabilities on the rules
- Estimating probabilities for PCFGs and decoding (parsing)
- How to use neural networks for constituency parsing