SFU NatLangLab

CMPT 825: Natural Language Processing

Constituency Parsing

Spring 2020

2020-03-24

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

Project Milestone

- Project Milestone due Tuesday 3/31
- PDF (2-4 pages) in the style of a conference (e.g. ACL/EMNLP) submission
  - https://2020.emnlp.org/files/emnlp2020-templates.zip
- Milestone should include:
  - Title and Abstract - motivate the problem, describe your goals, and highlight your findings
  - Approach - details on your main approach and baselines. Be specific. Make clear what part is original, what code you are writing yourself, what code you are using
  - Experiment - describe dataset, evaluation metrics, what experiments you plan to run, any results you have so far. Also provide training details, training times, etc.
  - Future Work - what is your plan for the rest of the project
  - Reference - provide references using BibTex
- Milestone will be graded based on progress and writing quality

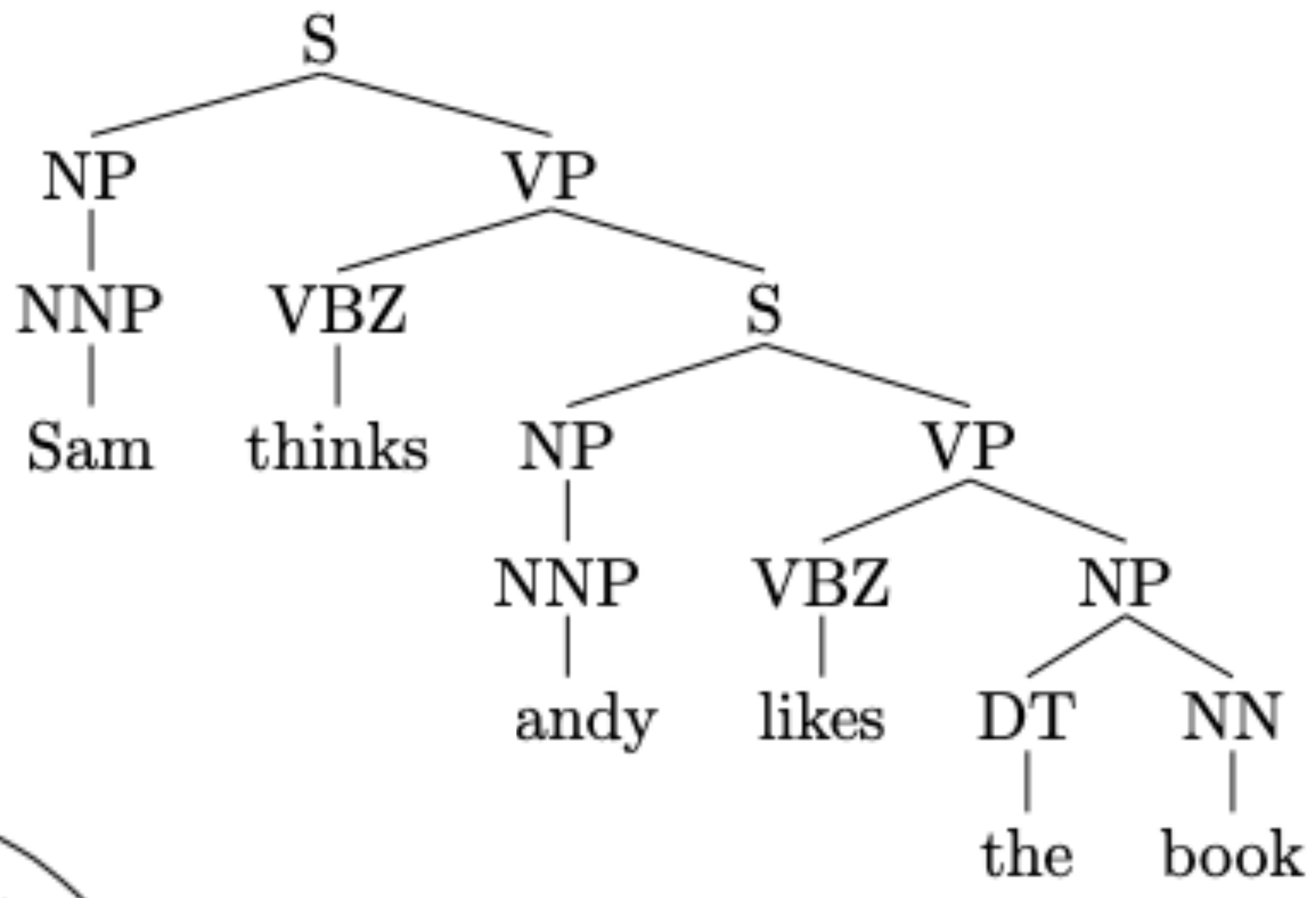
Overview

- 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

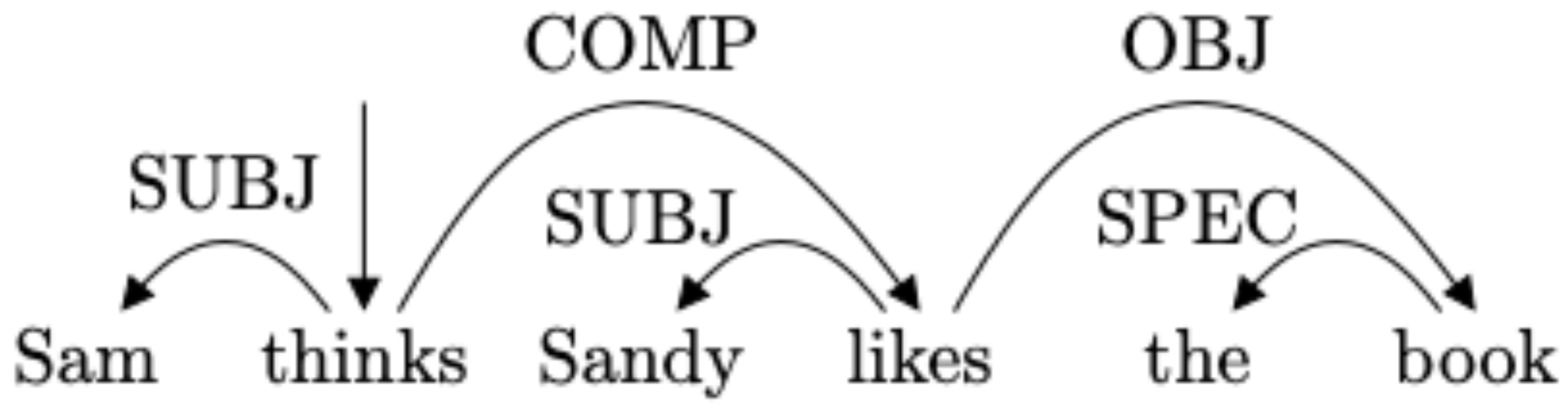
Syntactic structure: constituency and dependency

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

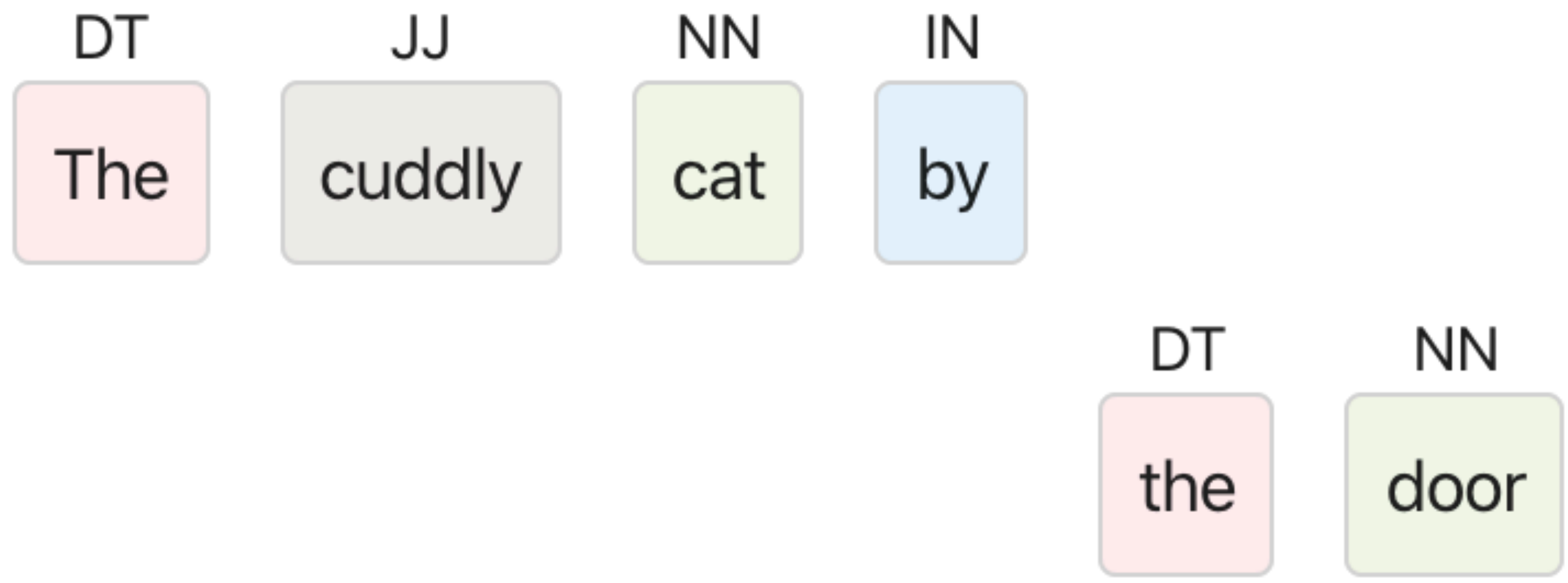
- Phrases can combine into bigger phrases recursively

the cuddly cat, by the door

NP PP→IN NP

the cuddly cat by the door

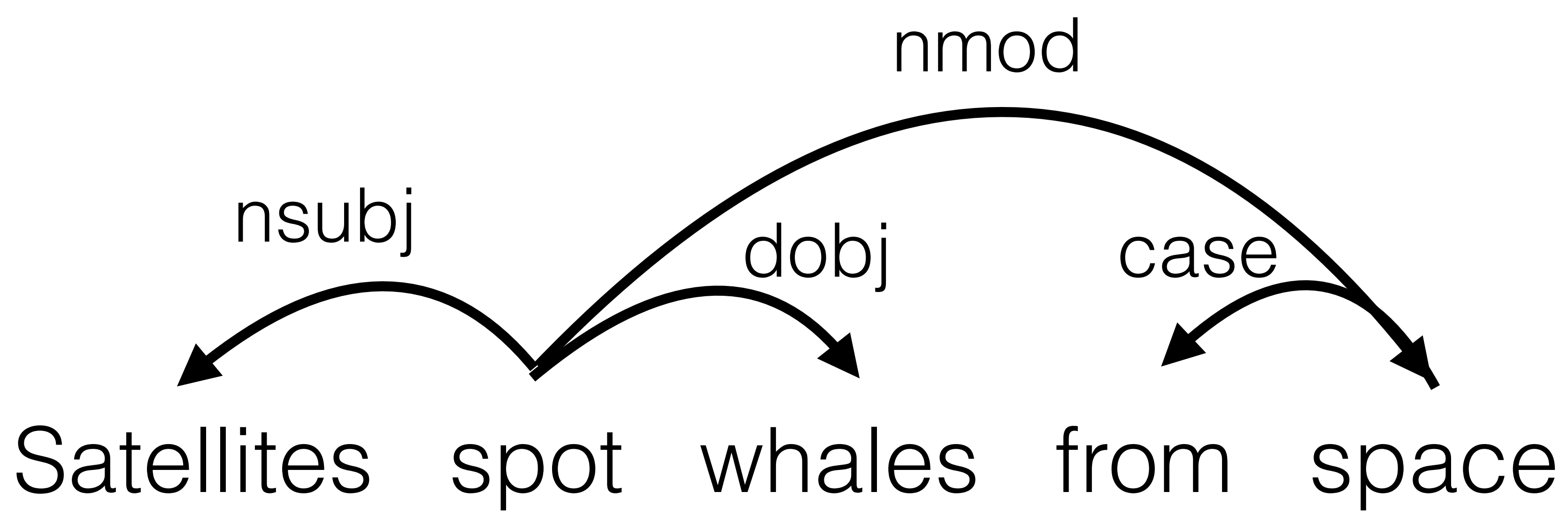
NP→ NP PP

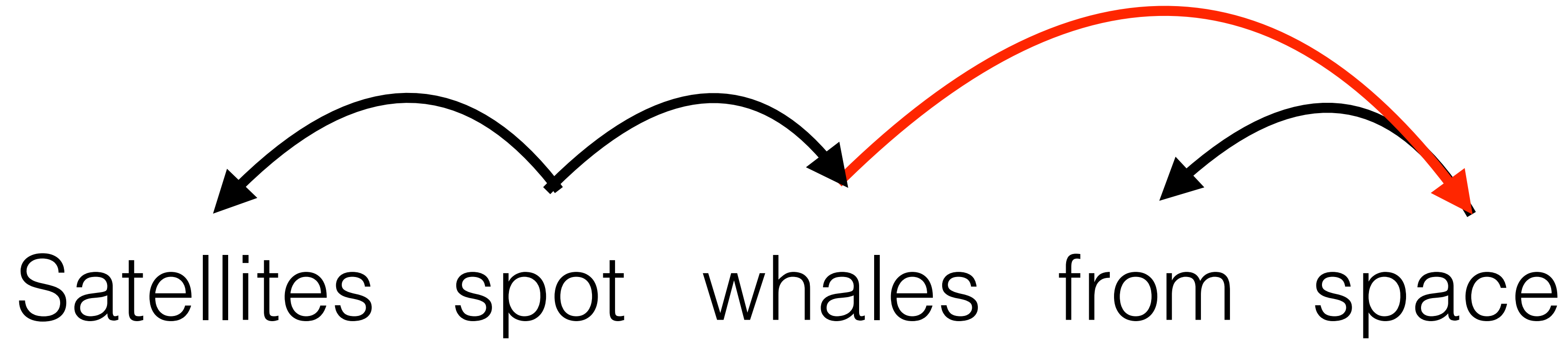


This Thursday

Dependency structure

- Dependency structure shows which words depend on (modify or are arguments of) which other words.





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

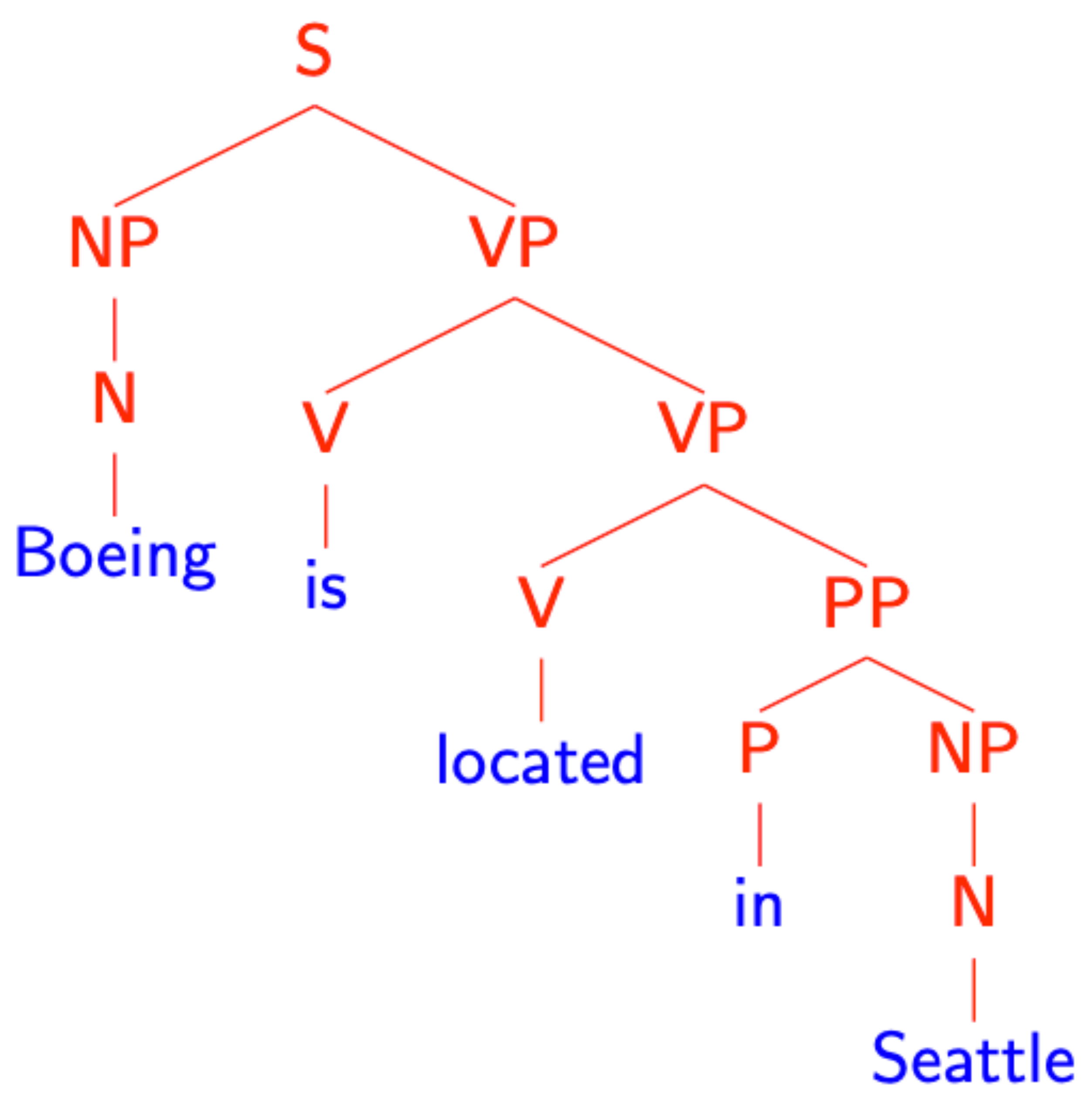
Syntactic parsing

- 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

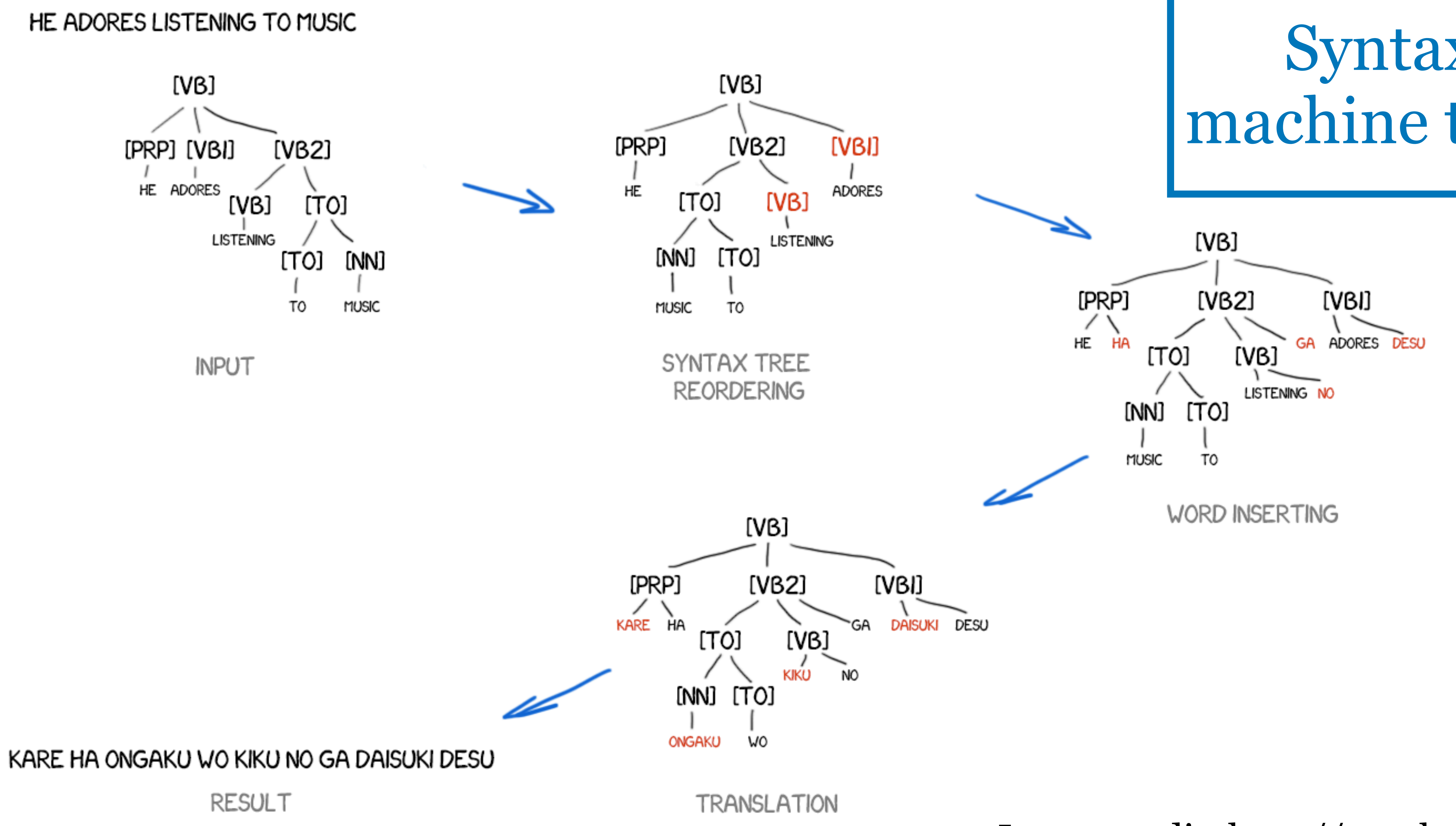
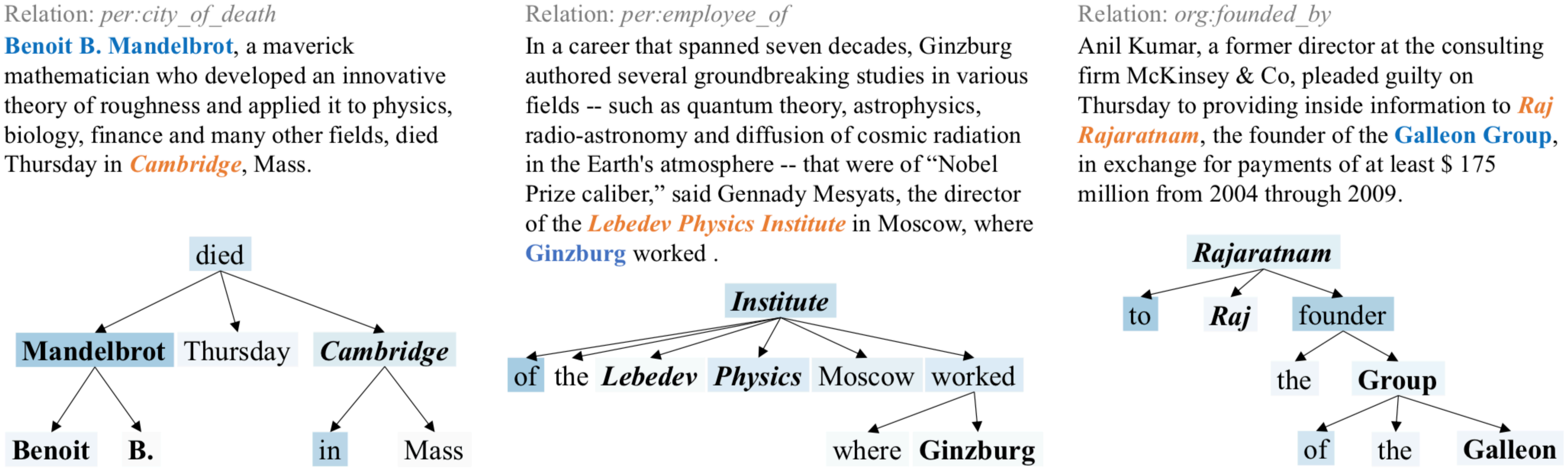


Image credit: http://vas3k.com/blog/machine_translation/

Syntactic parsing

- Used as intermediate representation for downstream applications

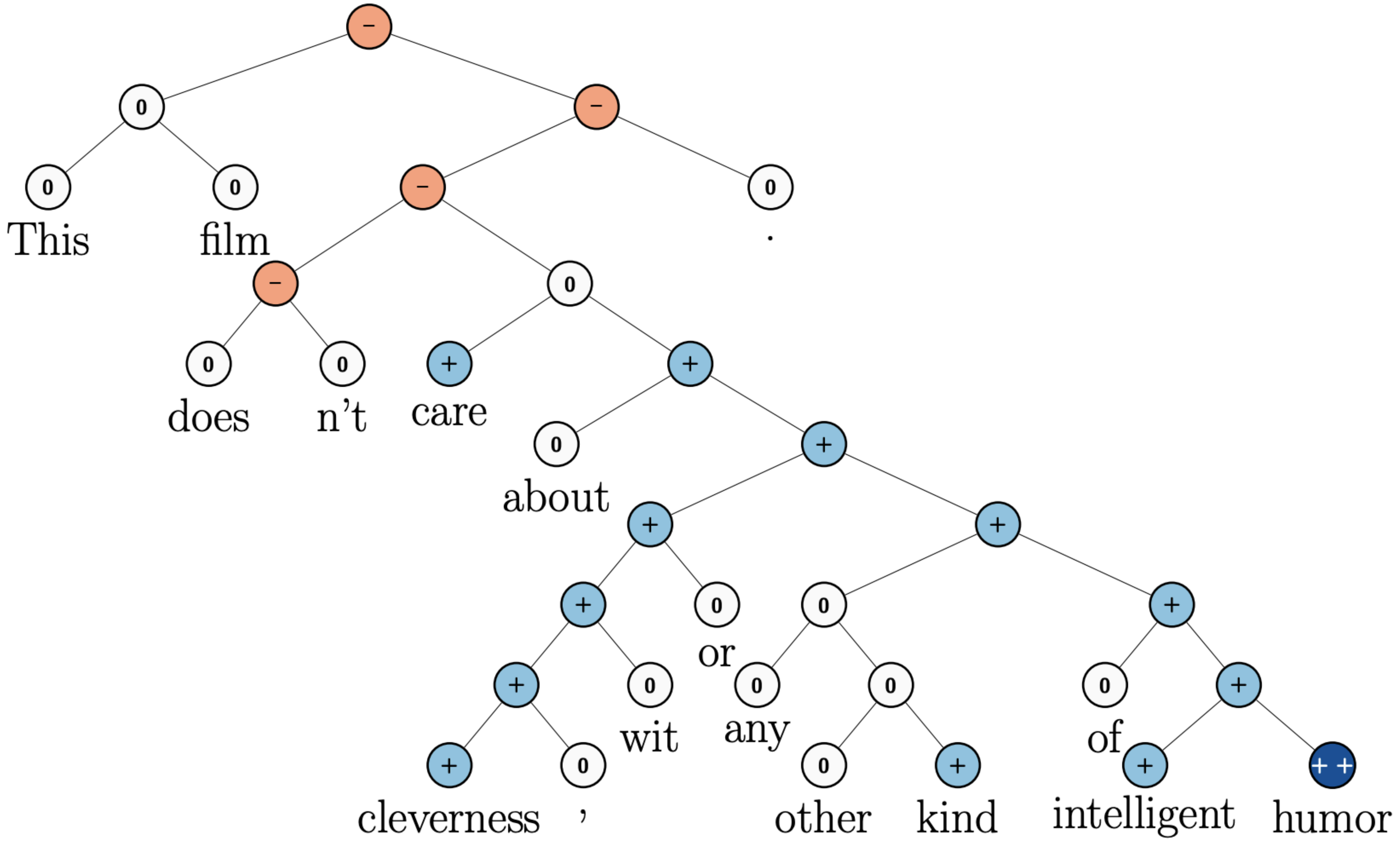


Relation Extraction

Image credit: (Zhang et al, 2018)

Beyond syntactic parsing

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

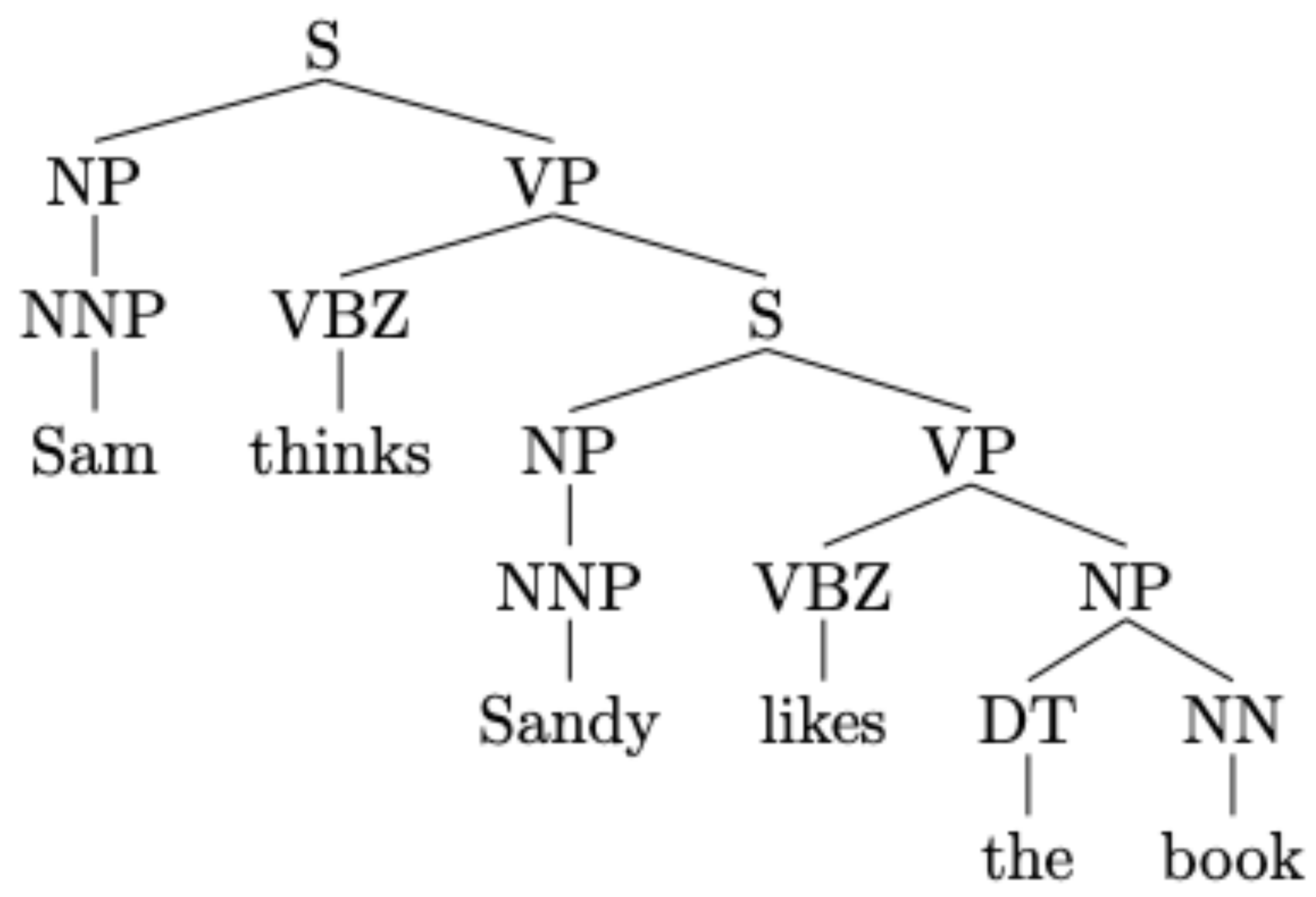


Nested Sentiment Analysis

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

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

$R =$

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

Grammar

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

Lexicon

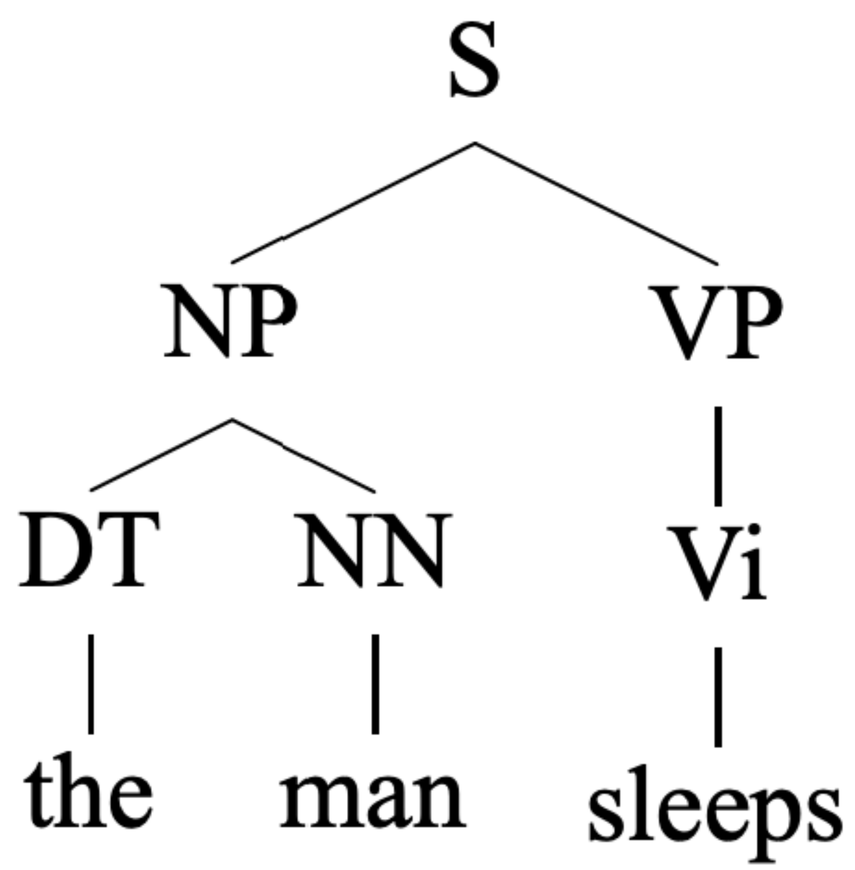
S:sentence, VP:verb phrase, NP: noun phrase, PP:prepositional phrase,
DT:determiner, Vi:intransitive verb, Vt:transitive verb, NN: noun, IN:preposition

(Left-most) Derivations

- Given a CFG G, a left-most derivation is a sequence of strings $s_1, s_2, \ldots, s_n$, where
  - $s_1 = S$
  - $s_n \in \Sigma^*$: all possible strings made up of words from Σ
  - Each s_i for $i = 2, \ldots, n$ is derived from s_{i-1} by picking the left-most non-terminal X in s_{i-1} and replacing it by some β where $X \to \beta \in R$
- s_n: yield of the derivation

(Left-most) Derivations

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



$R =$

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

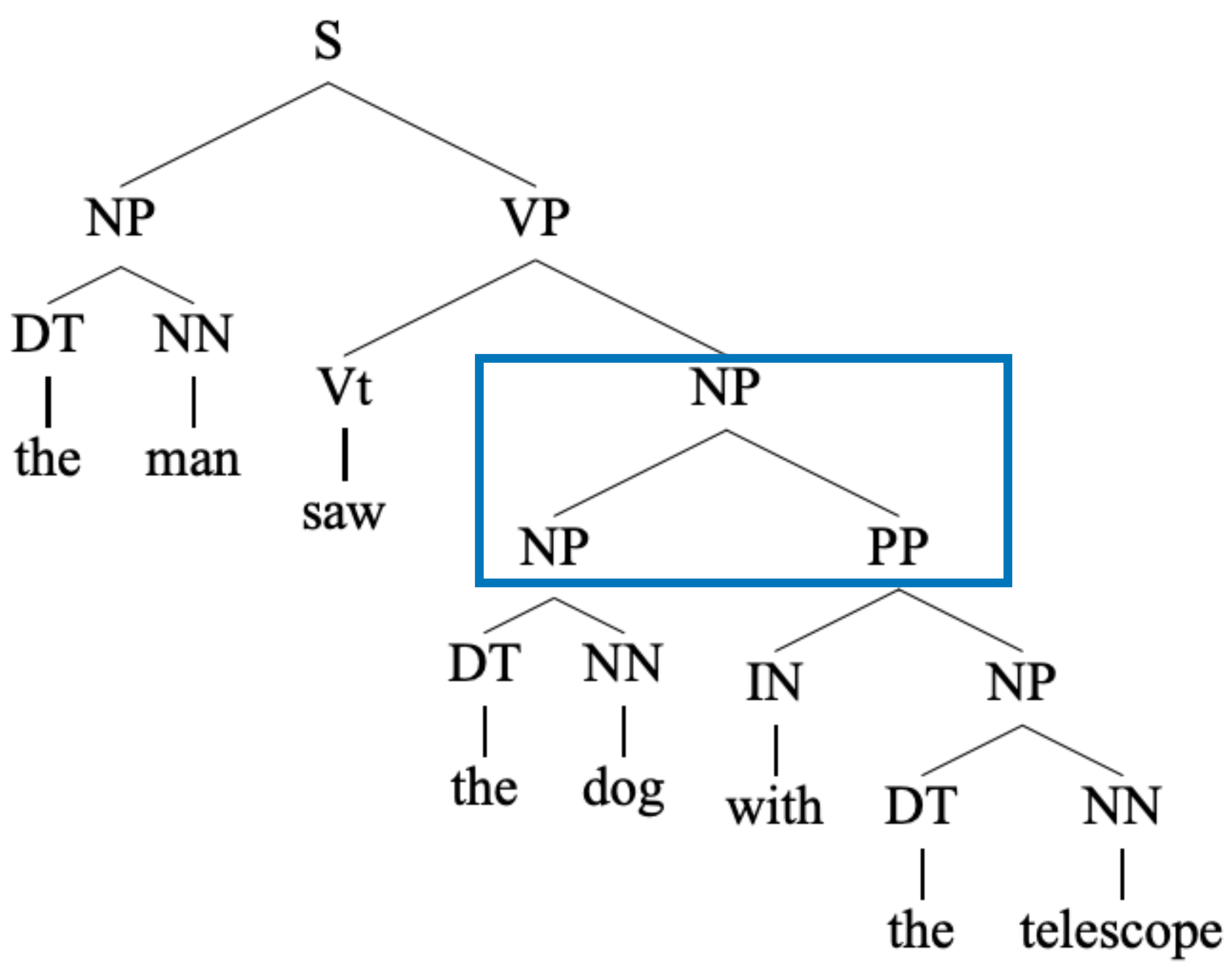
Vi	$\rightarrow$	sleeps
Vt	$\rightarrow$	saw
NN	$\rightarrow$	man
NN	$\rightarrow$	woman
NN	$\rightarrow$	telescope
NN	$\rightarrow$	dog
DT	$\rightarrow$	the
IN	$\rightarrow$	with
IN	$\rightarrow$	in

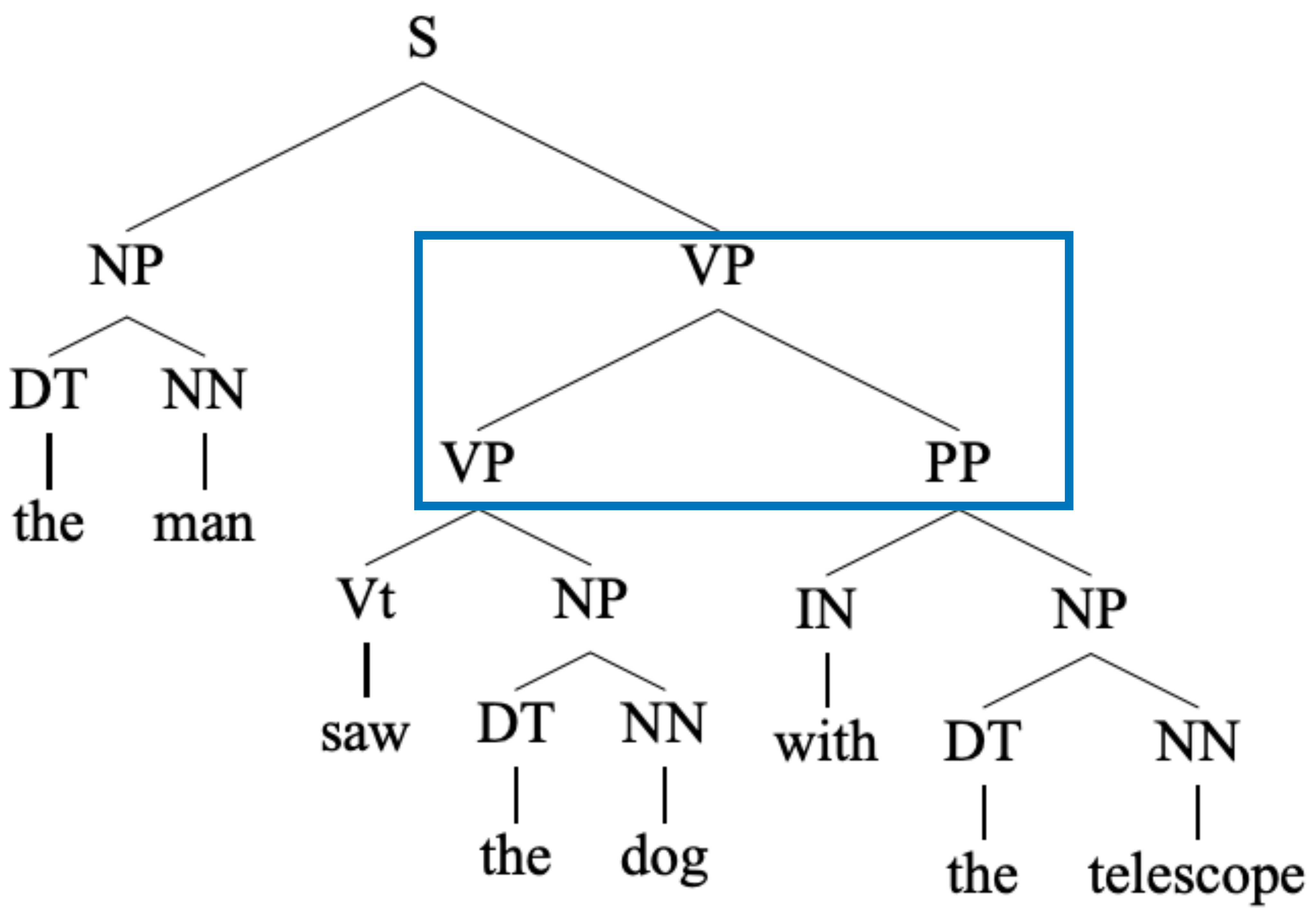
<u>A derivation can be represented as a parse tree!</u>

- 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 trees!).





"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].

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

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

- 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!

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)

The Penn Treebank Project (Marcus et al, 1993)

Treebanks

- Standard setup (WSJ portion of Penn Treebank):
  - 40,000 sentences for training
  - 1,700 for development
  - 2,400 for testing
- Why building a treebank instead of a grammar?
  - Broad coverage
  - Frequencies and distributional information
  - A way to evaluate systems

Probabilistic context-free grammars (PCFGs)

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

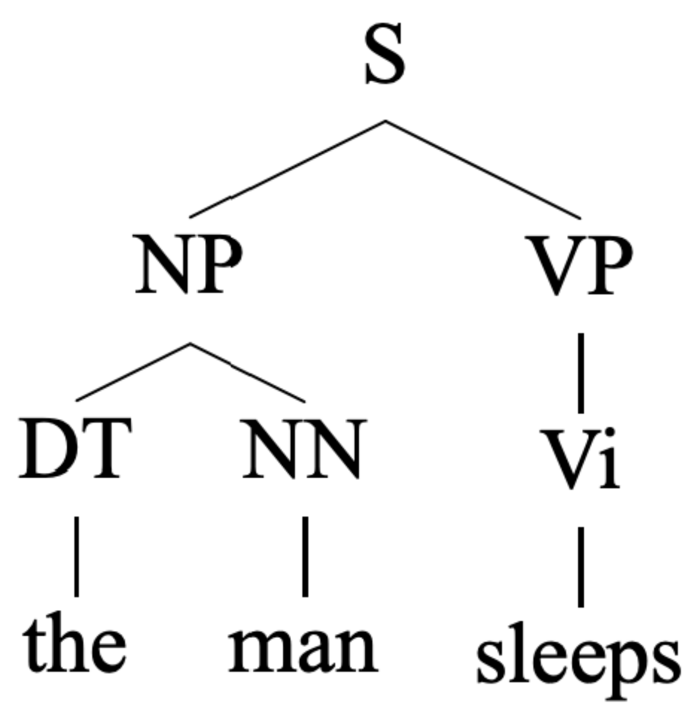
- A probabilistic context-free grammar (PCFG) consists of:
  - A context-free grammar: $G = (N, \Sigma, R, S)$
  - For each rule $\alpha \to \beta \in R$, there is a parameter $q(\alpha \to \beta) \geq 0$. For any $X \in N$,

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

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:

$$\prod_{i=1}^{l} q(\alpha_i \rightarrow \beta_i)$$



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

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

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

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

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 the 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, $q(\text{VP} \to \text{Vt NP}) = 0.105$

CFG vs PCFG

- 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.

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)

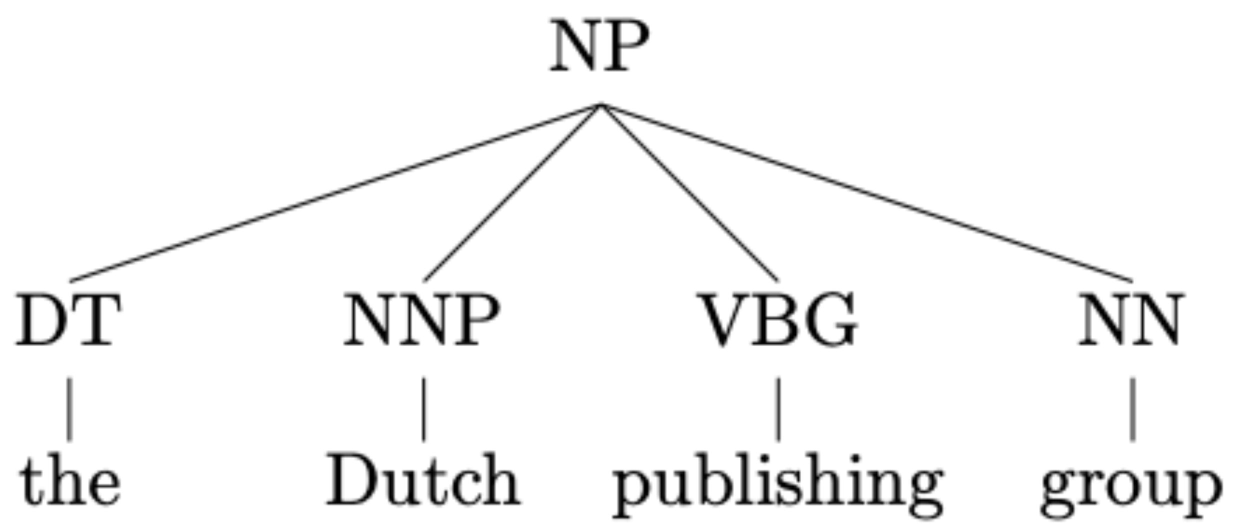
- **Chomsky Normal Form (CNF)**: all the rules take one of the two following forms:

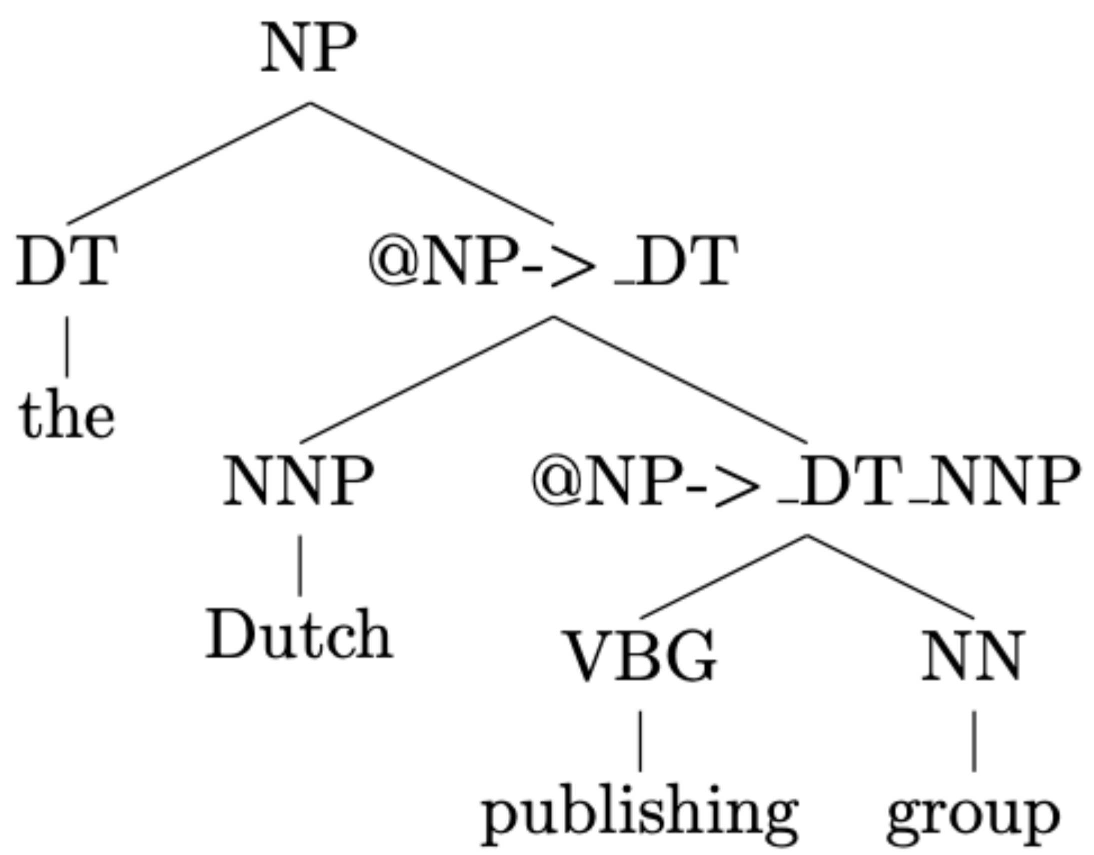
    - $X \rightarrow Y_1 Y_2$ where $X \in N, Y_1 \in N, Y_2 \in N$ Binary
    - $X \rightarrow Y$ where $X \in N, Y \in \Sigma$ Unary

- 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 $\rightarrow$ DT NNP VBG NN





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

The CKY algorithm

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

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

$_0$ Book $_1$ the $_2$ flight $_3$ through $_4$ Houston $_5$

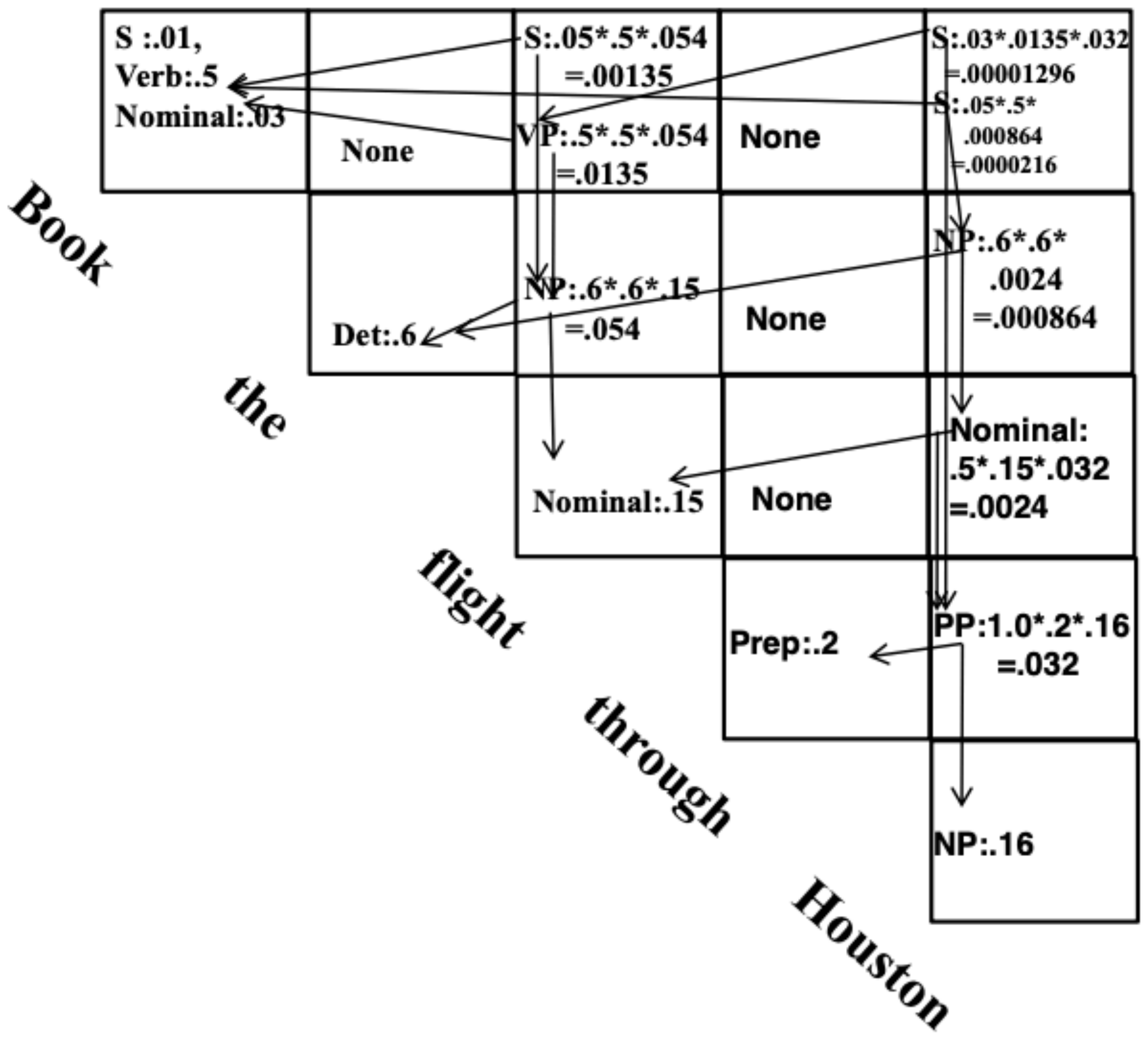
The CKY algorithm

- 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



Cells contain:

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

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 $X \in N$,

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

Algorithm:

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

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

and

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

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

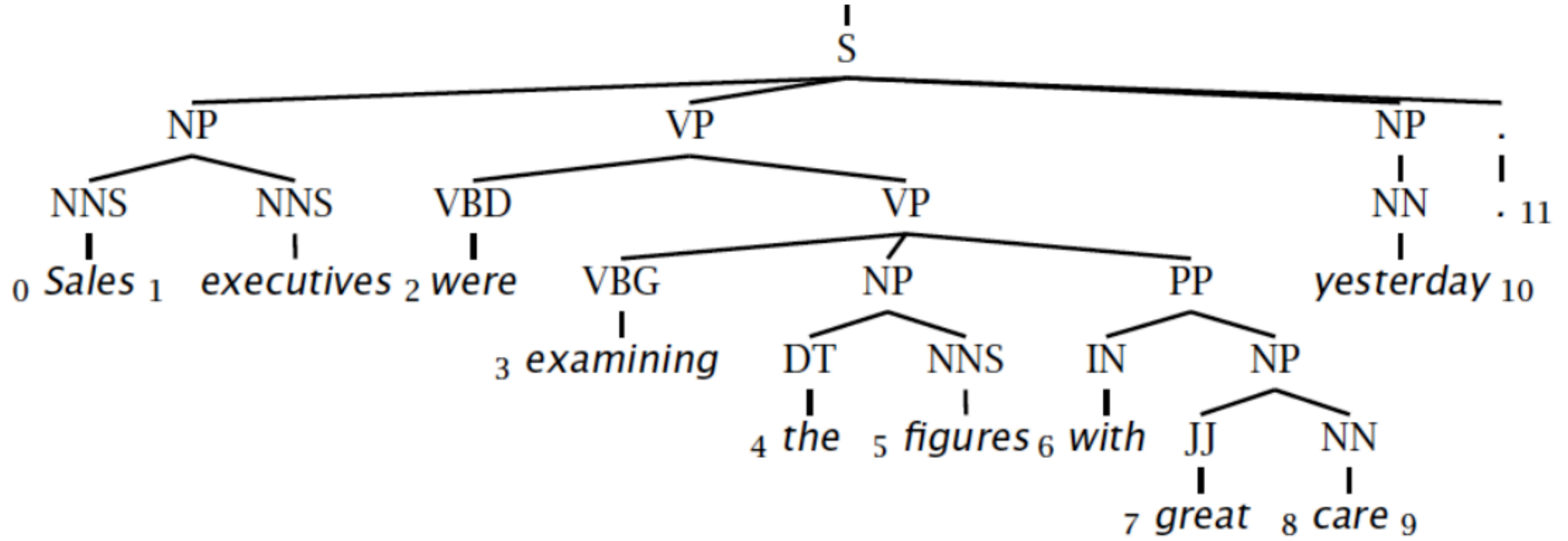
How does this change CKY?

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

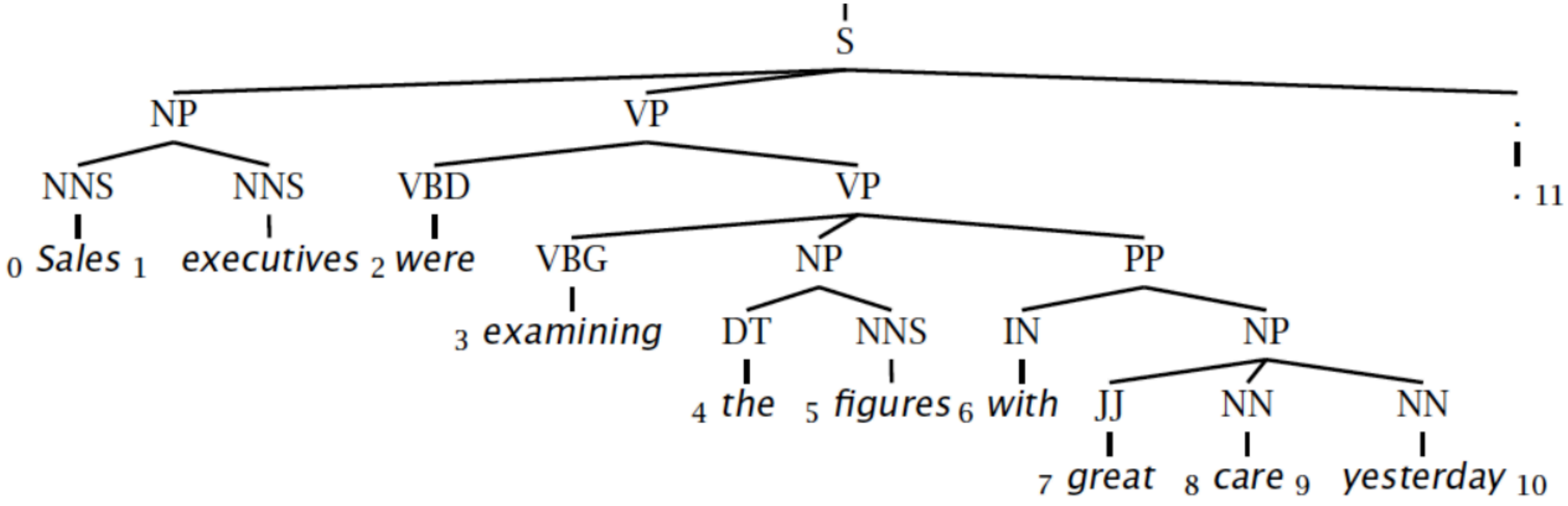
- Compute unary closure: if there is a rule chain $X \rightarrow Y_1, Y_1 \rightarrow Y_2, \ldots, Y_k \rightarrow Y$, add $q(X \rightarrow Y) = q(X \rightarrow Y_1) \times \cdots \times q(Y_k \rightarrow Y)$
- Update unary rule once after the binary rules

Evaluating constituency parsing

Gold standard brackets: **S-(0:11)**, **NP-(0:2)**, VP-(2:9), VP-(3:9), **NP-(4:6)**, PP-(6-9), NP-(7,9), NP-(9:10)



Candidate brackets: **S-(0:11)**, **NP-(0:2)**, VP-(2:10), VP-(3:10), **NP-(4:6)**, PP-(6-10), NP-(7,10)

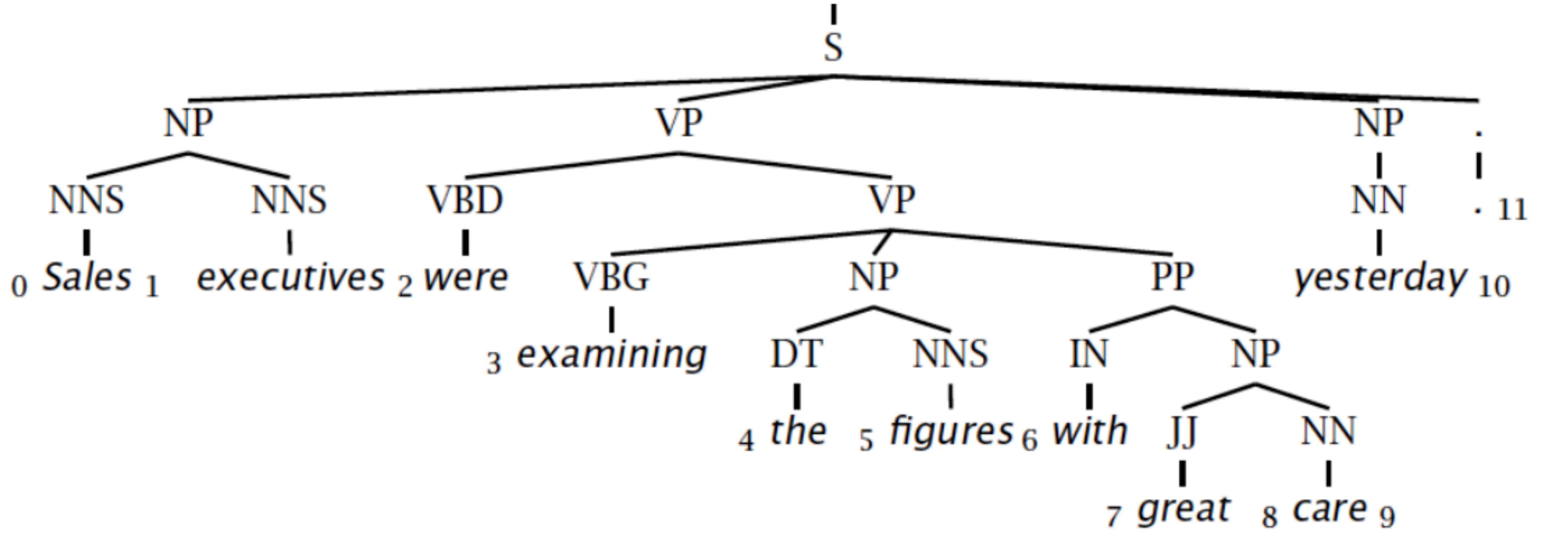


Evaluating constituency parsing

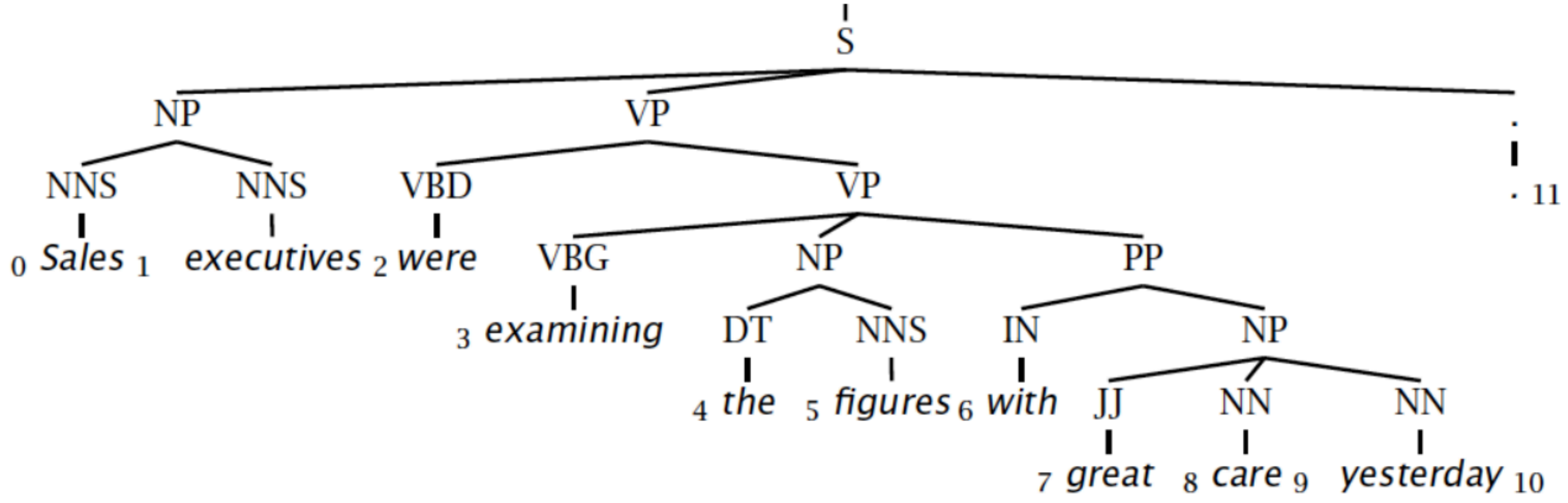
- Recall: (# correct constituents in candidate) / (# constituents in gold tree)
- Precision: (# correct constituents in candidate) / (# constituents in candidate)
- Labeled precision/recall require getting the non-terminal label correct
- F1 = (2 * precision * recall) / (precision + recall)

Evaluating constituency parsing

Gold standard brackets: **S-(0:11)**, **NP-(0:2)**, VP-(2:9), VP-(3:9), **NP-(4:6)**, PP-(6-9), NP-(7,9), NP-(9:10)



Candidate brackets: **S-(0:11)**, **NP-(0:2)**, VP-(2:10), VP-(3:10), **NP-(4:6)**, PP-(6-10), NP-(7,10)



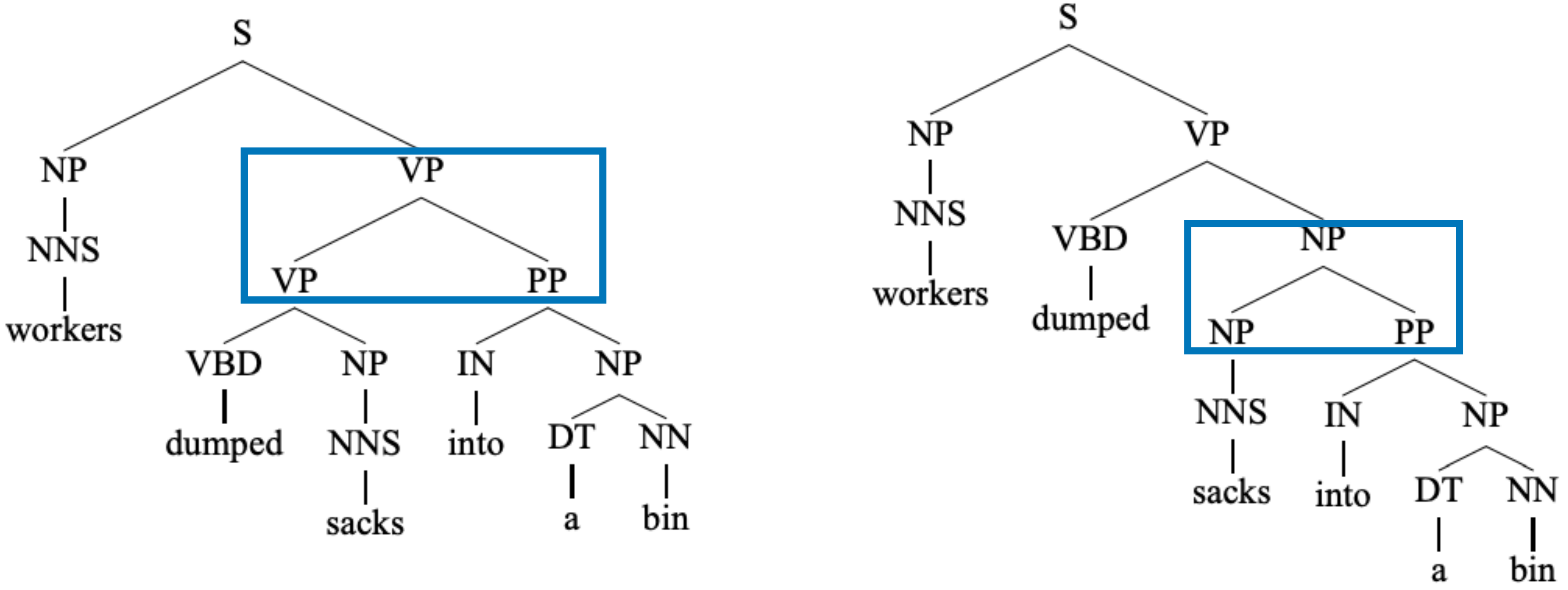
- Precision: 3/7 = 42.9%
- Recall: 3/8 = 37.5%
- F1 = 40.0%
- Tagging accuracy: 100%

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)



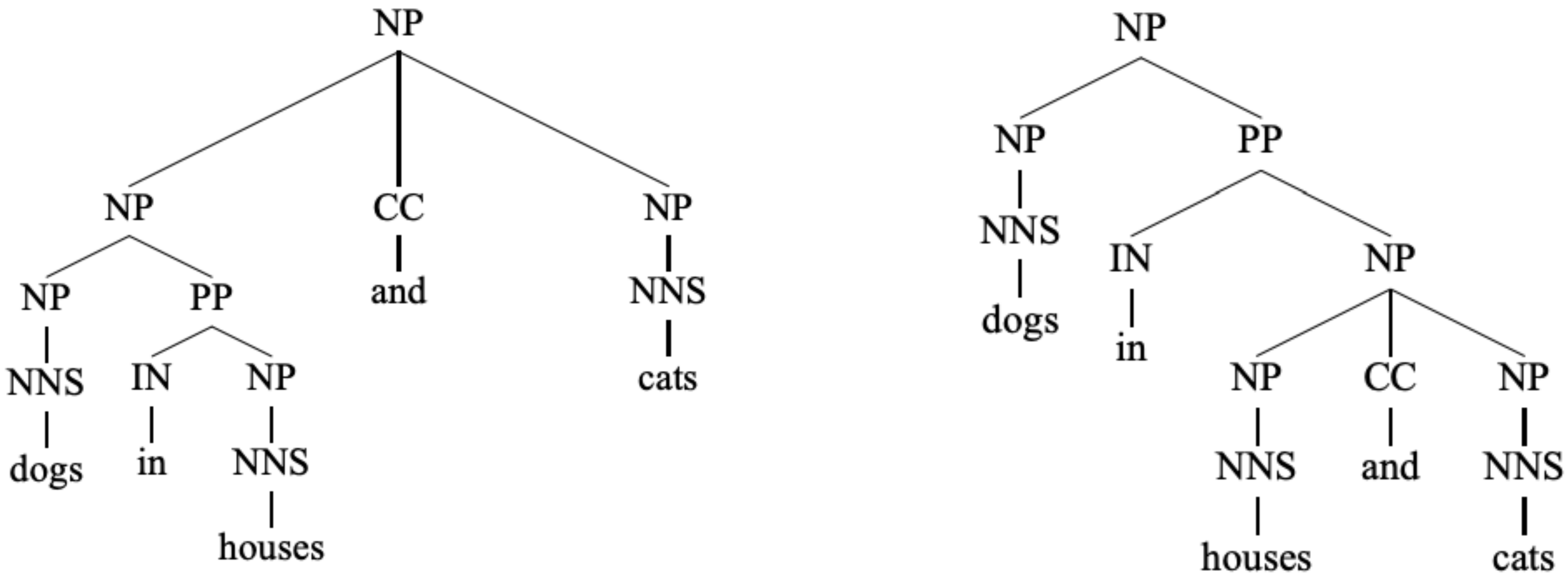
The only difference between these two parses:

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

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

Weaknesses of PCFGs

- Lack of sensitivity to lexical information (words)

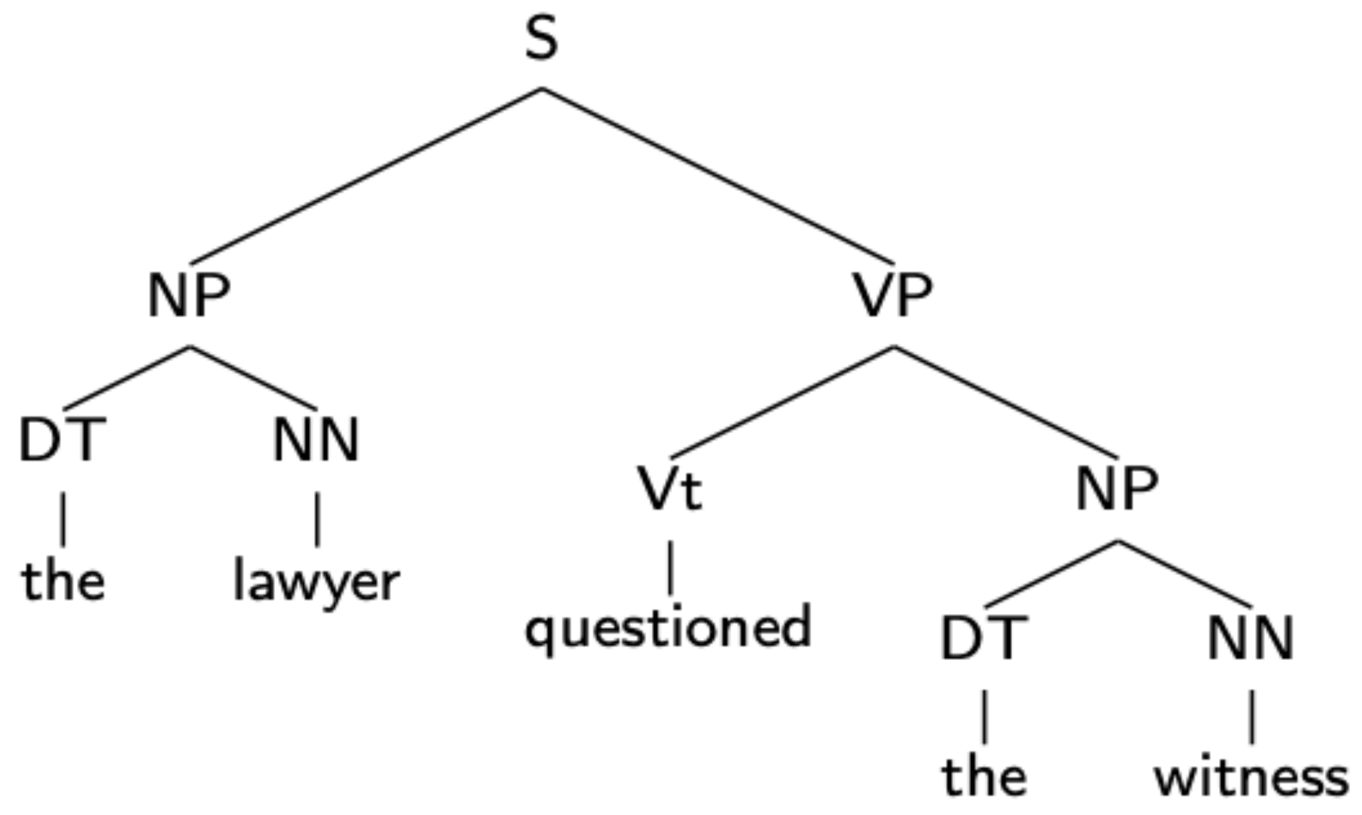


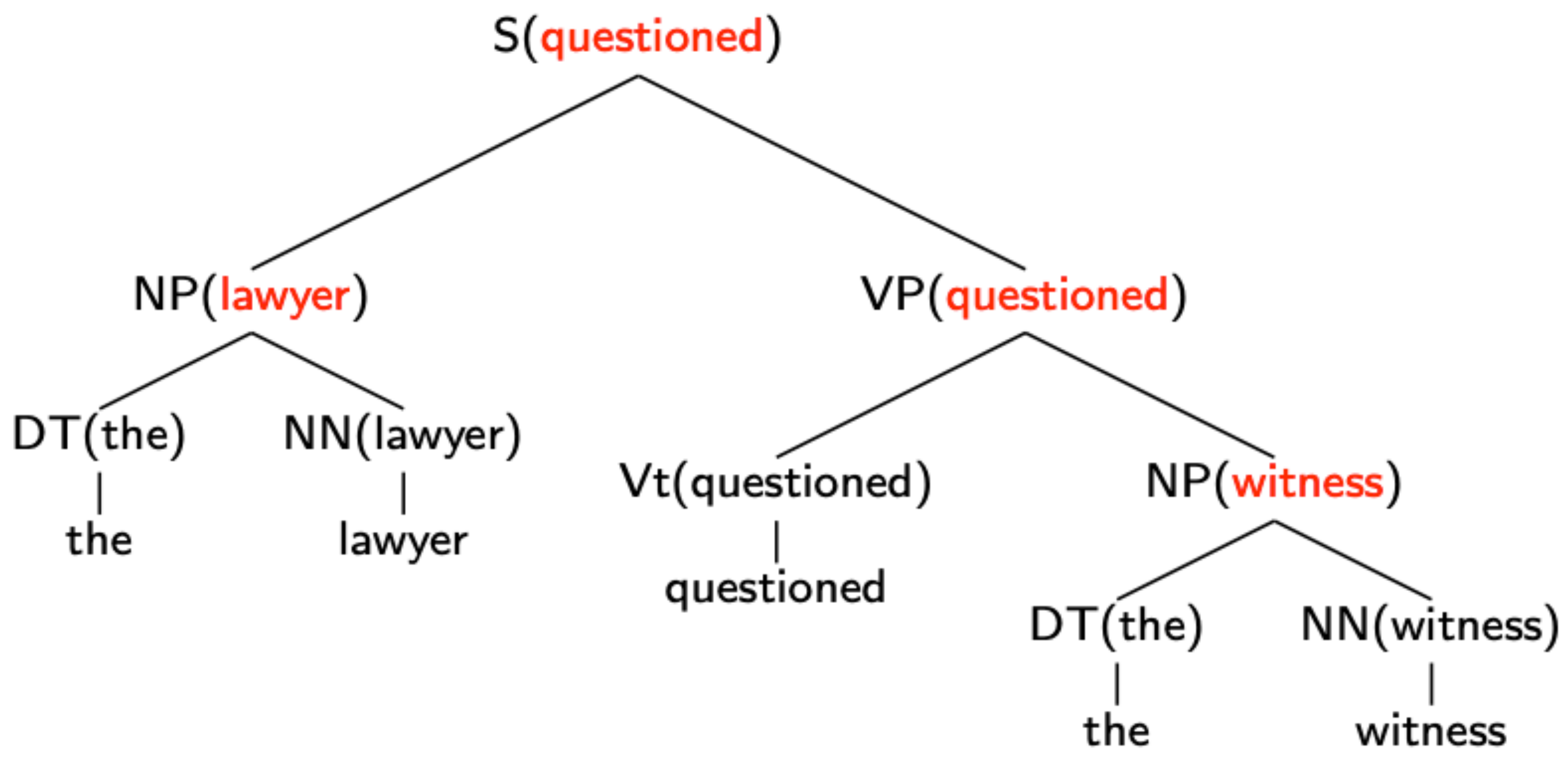
Exactly the same set of context-free rules!

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)

S	$\Rightarrow$	NP	VP		(VP is the head)
VP	$\Rightarrow$	Vt	NP		(Vt is the head)
NP	$\Rightarrow$	DT	NN	NN	(NN is the head)

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

S(saw)	$\rightarrow_2$	NP(man)	VP(saw)
VP(saw)	$\rightarrow_1$	Vt(saw)	NP(dog)
NP(man)	$\rightarrow_2$	DT(the)	NN(man)
NP(dog)	$\rightarrow_2$	DT(the)	NN(dog)
Vt(saw)	$\rightarrow$	saw	
DT(the)	$\rightarrow$	the	
NN(man)	$\rightarrow$	man	
NN(dog)	$\rightarrow$	dog	

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

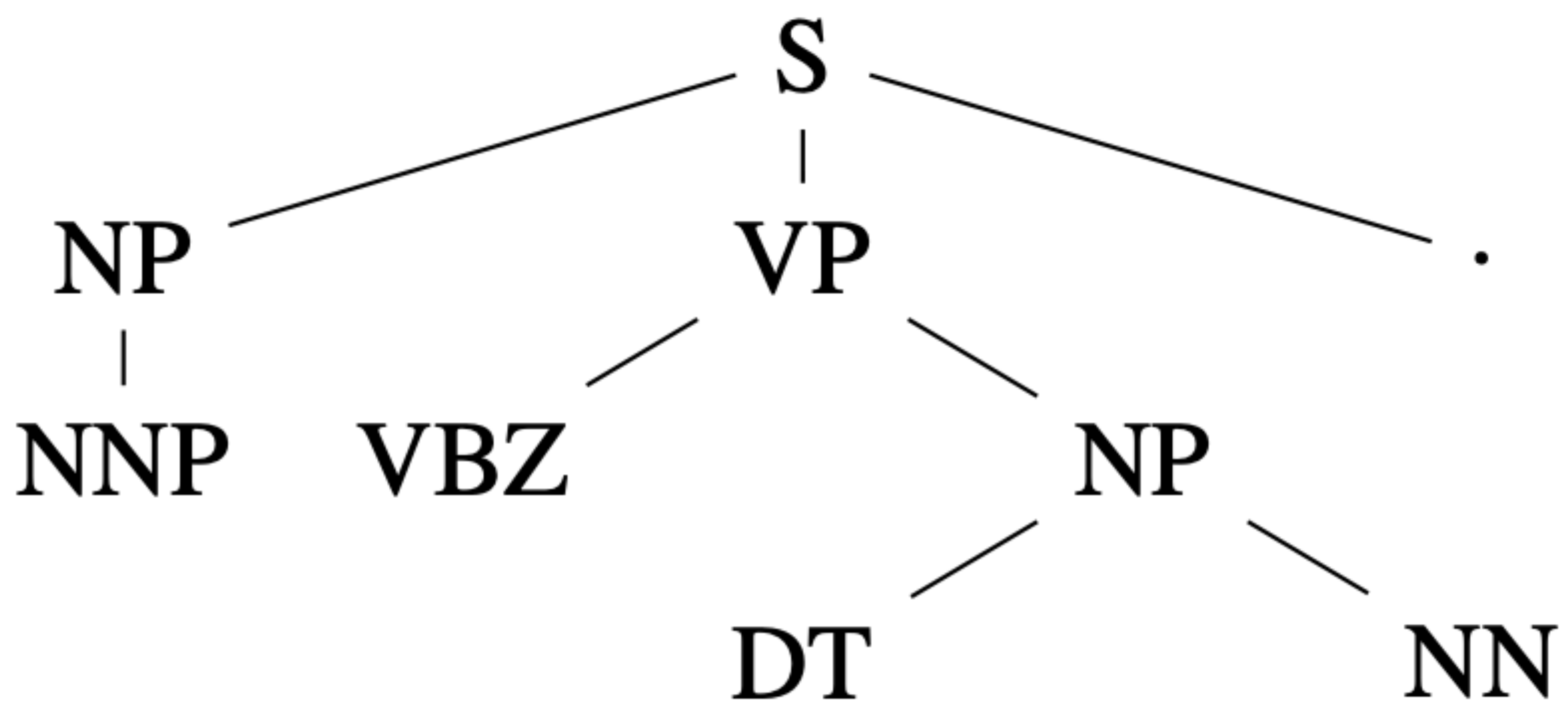
Parsing with Neural Networks

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)

John has a dog . $\rightarrow$



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

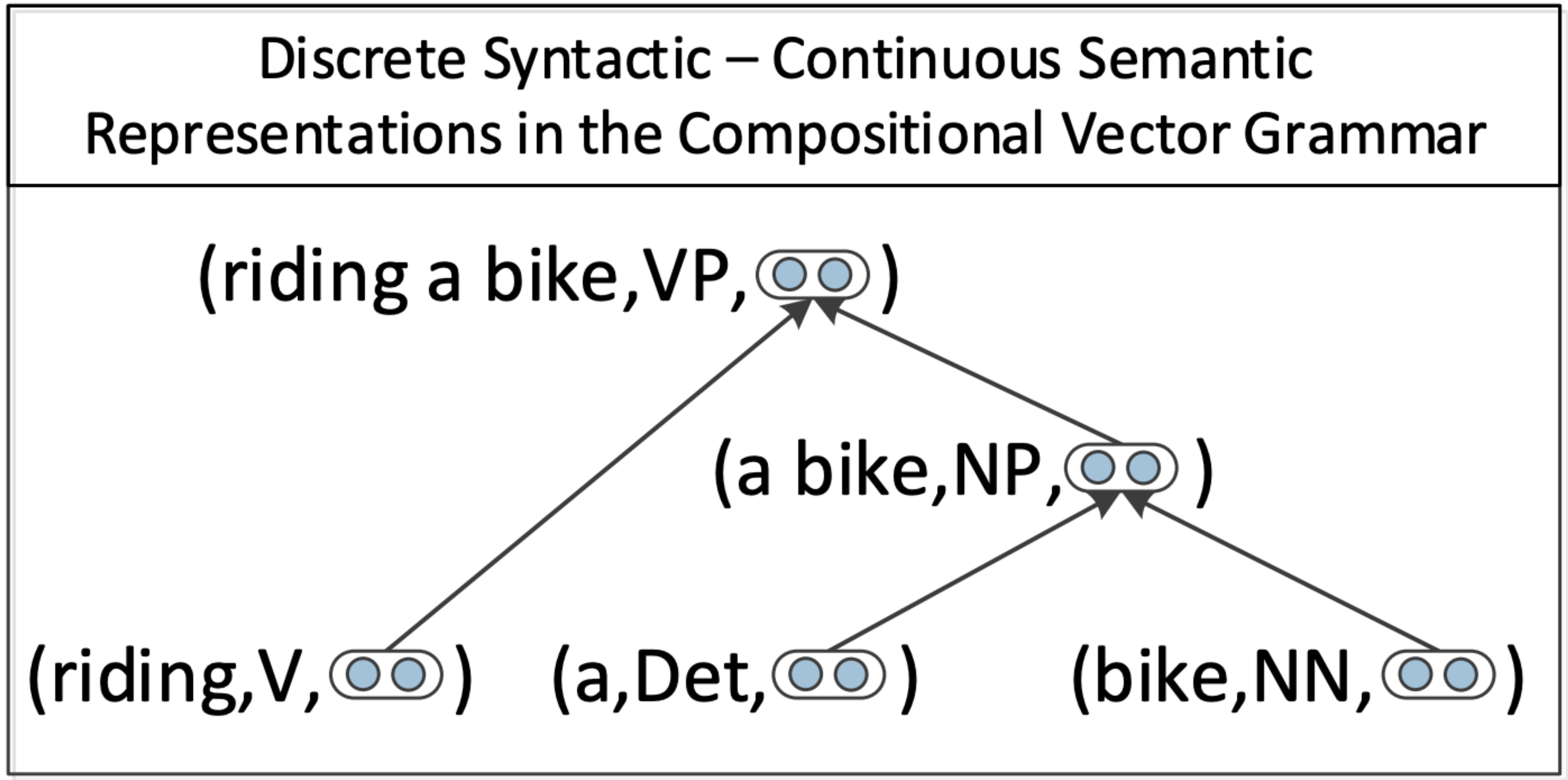
May not be structural correct
(i.e. unbalanced parenthesis)

88.3 F1

- Linearize parse tree and train LSTM seq2seq model with attention

Recursive Neural Networks (Socher et al, 2013)

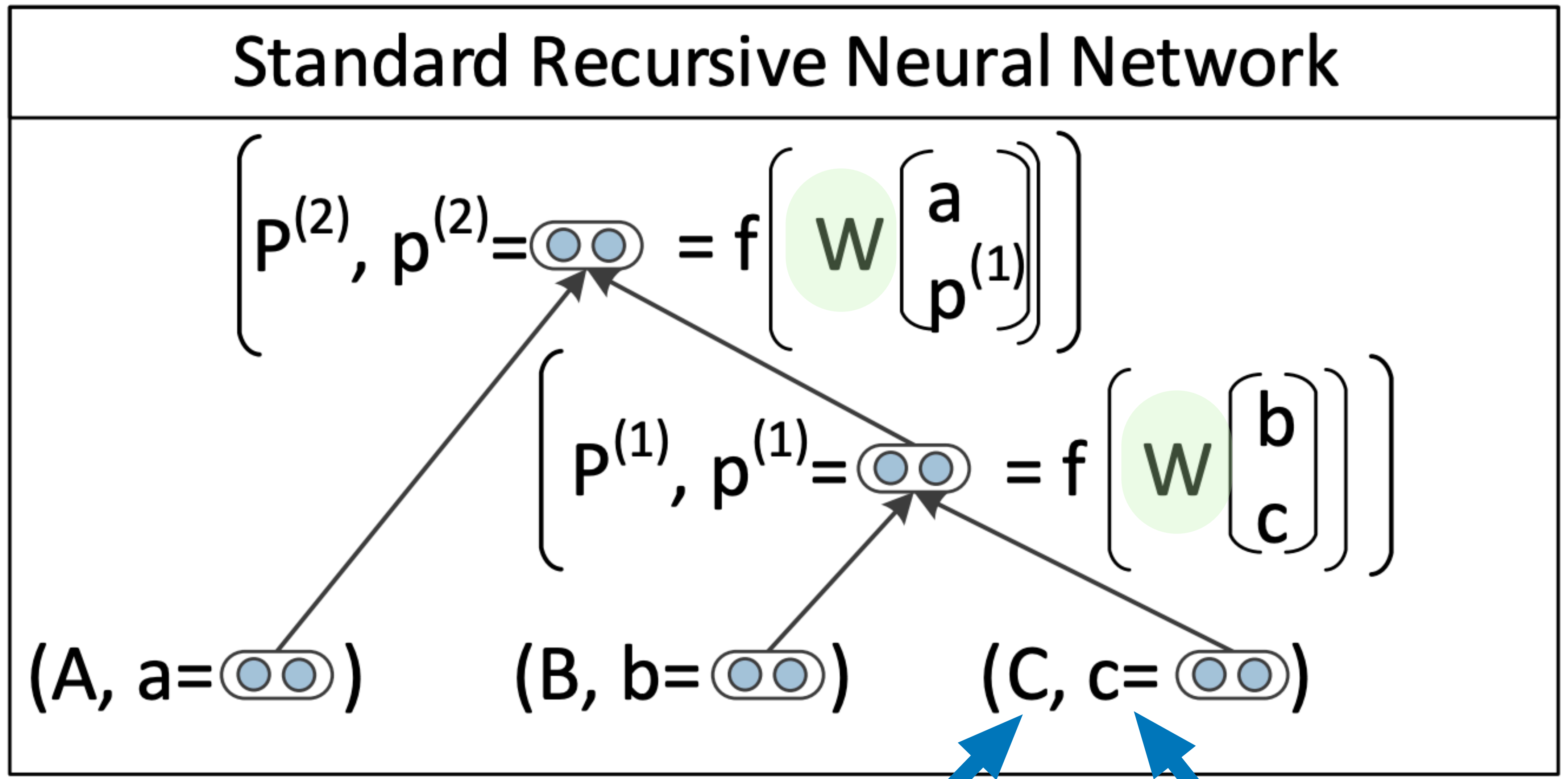
- Continuous representations for words and non-terminal nodes
- Compositional representations for non-terminal nodes
- Use neural networks to get compositional representations as well as scores for composition



Compositional Vector Grammar = PCFG + TreeRNN

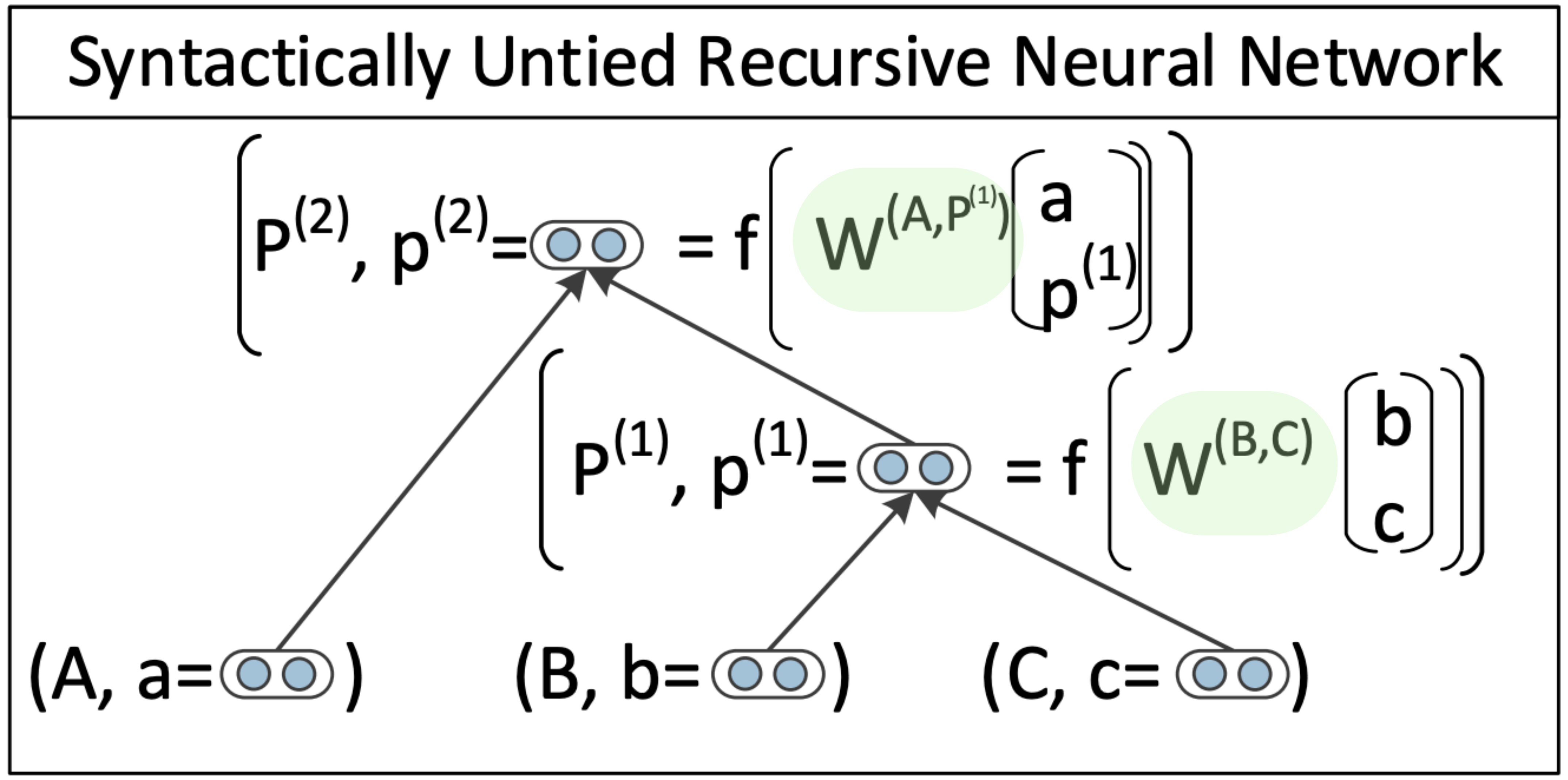
Recursive Neural Networks
(Socher et al, 2013)

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



Node label Node embedding

Weights can be tied



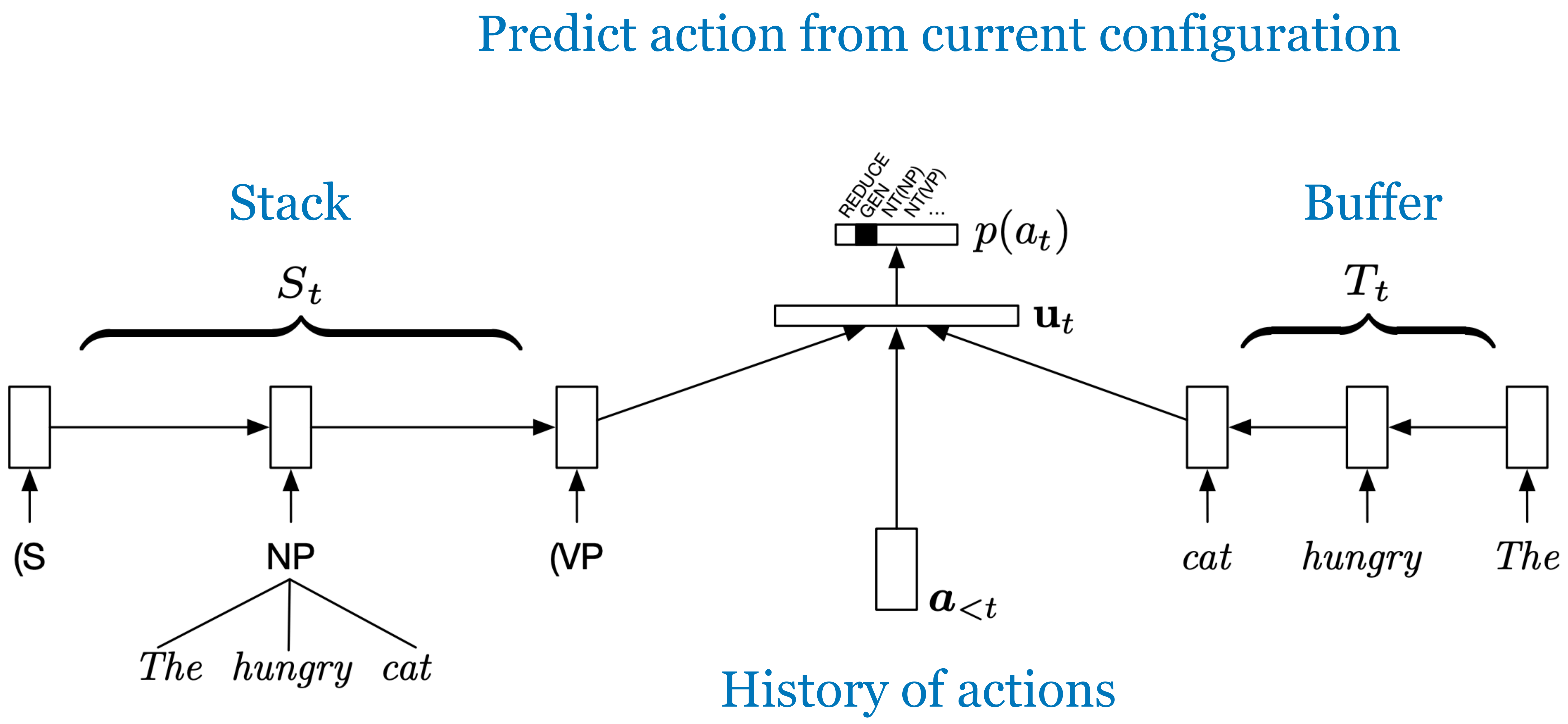
or parameterized by constituency type

90.4 F1

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:

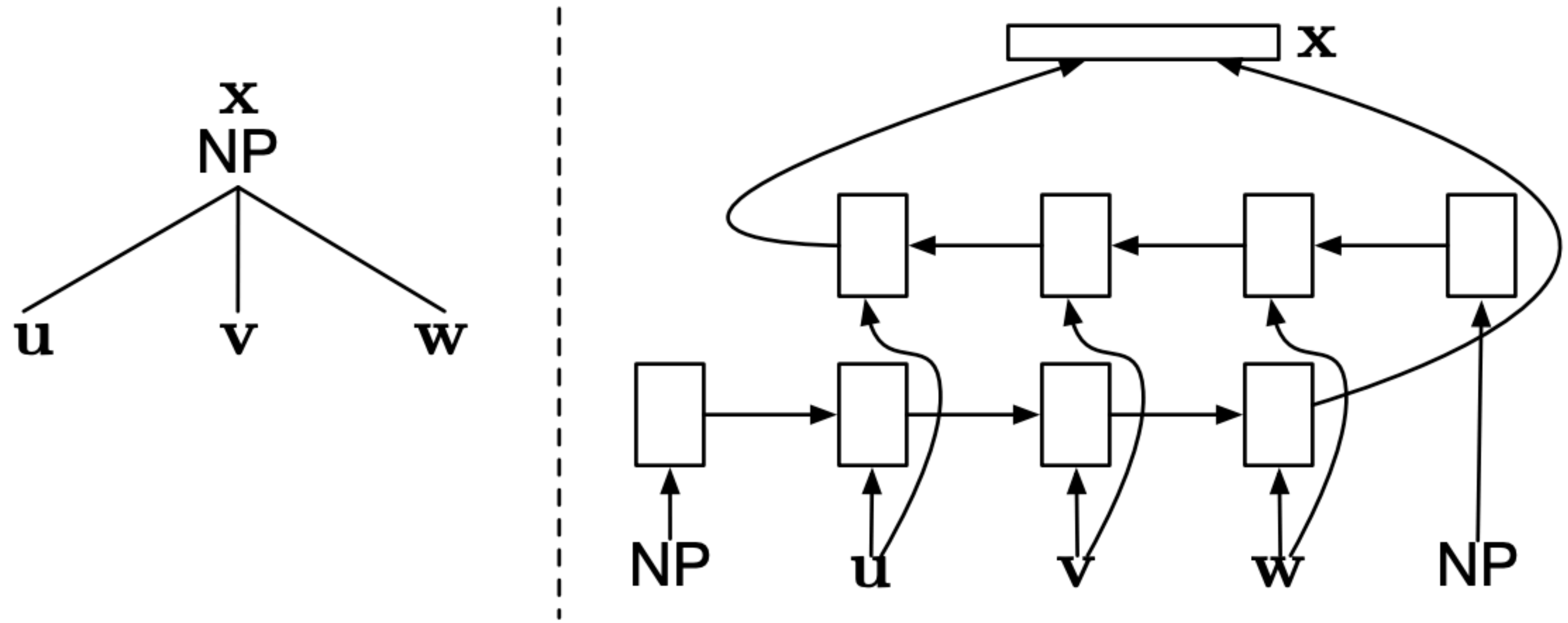
NT(X): Open (create) a new non-terminal of type X

SHIFT: move x from buffer to stack

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

Recurrent Neural Network Grammars (Dyer et al, 2016)

REDUCE

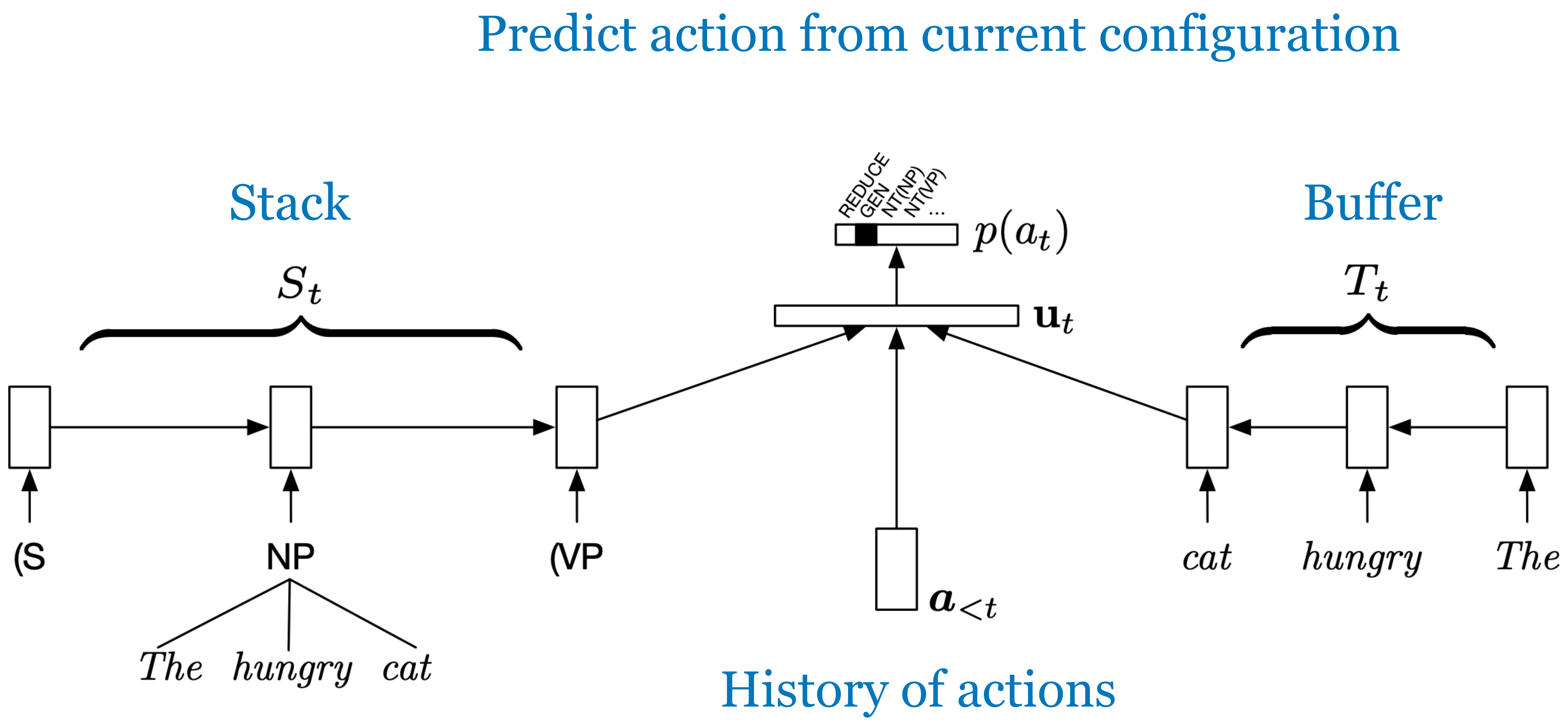


- BiLSTM to get composite representation of non-terminal

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



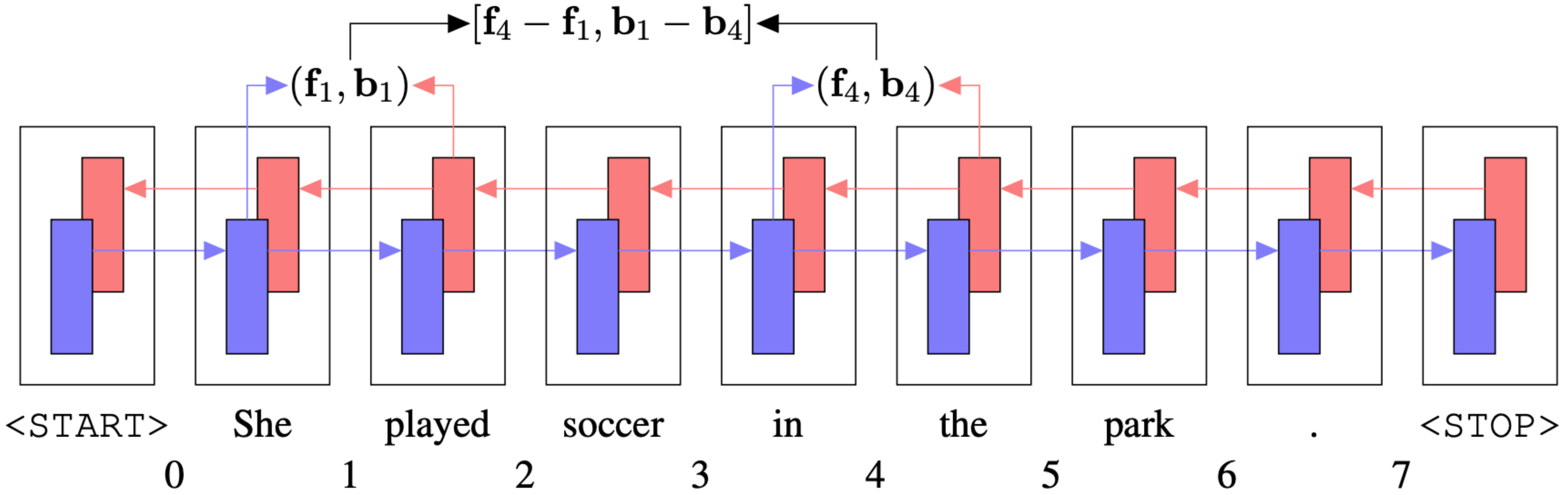
91.2 F1

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**

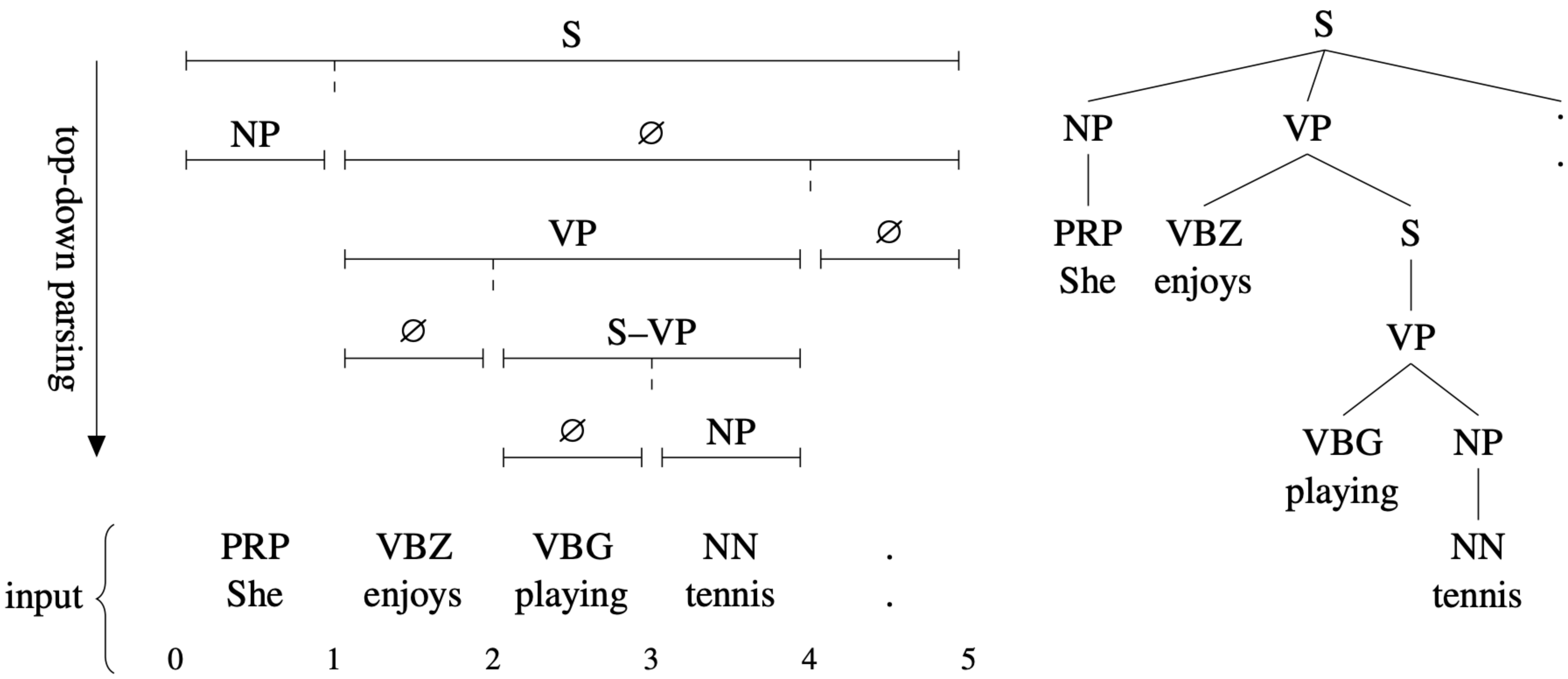
- Recursively for each span:
  - Assign a label
  - Pick a split point

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

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

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

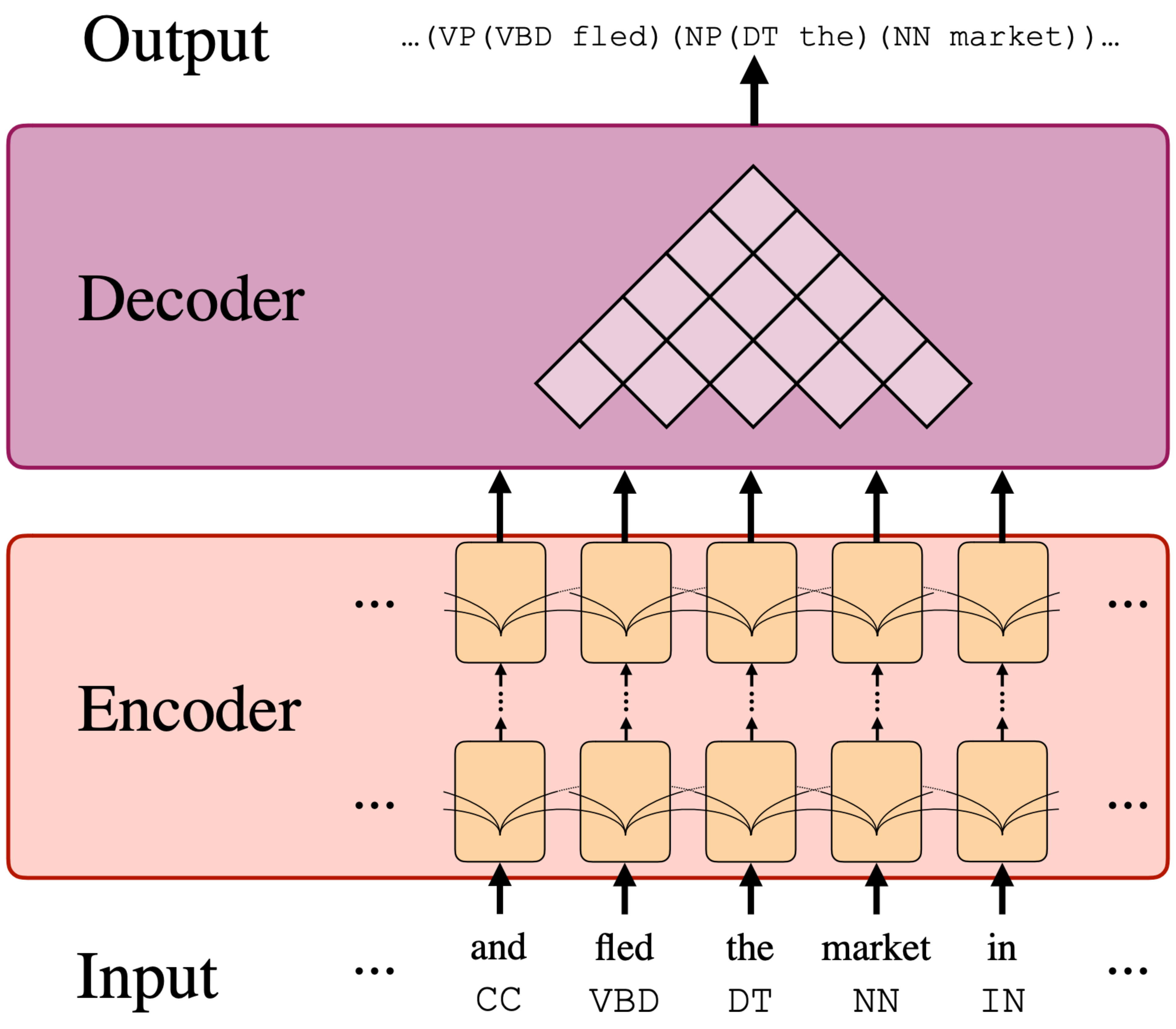
91.8 F1



(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, 95.1 (+ELMo)