Natural Language Processing
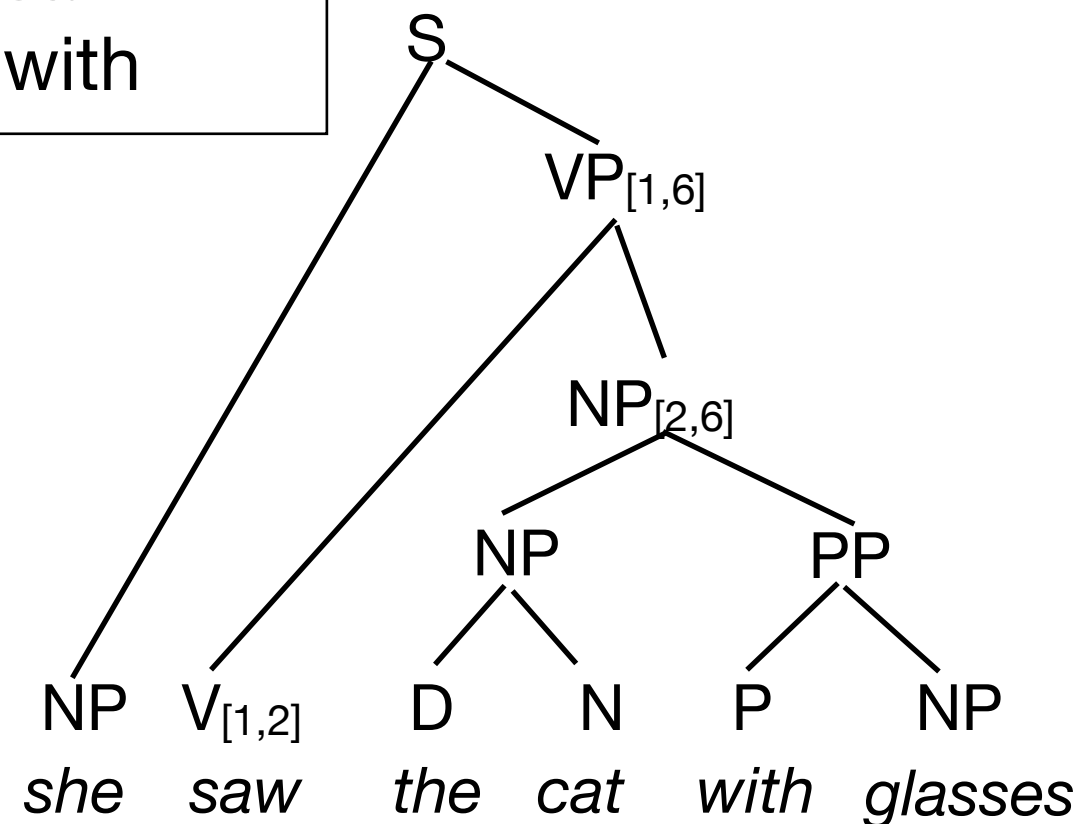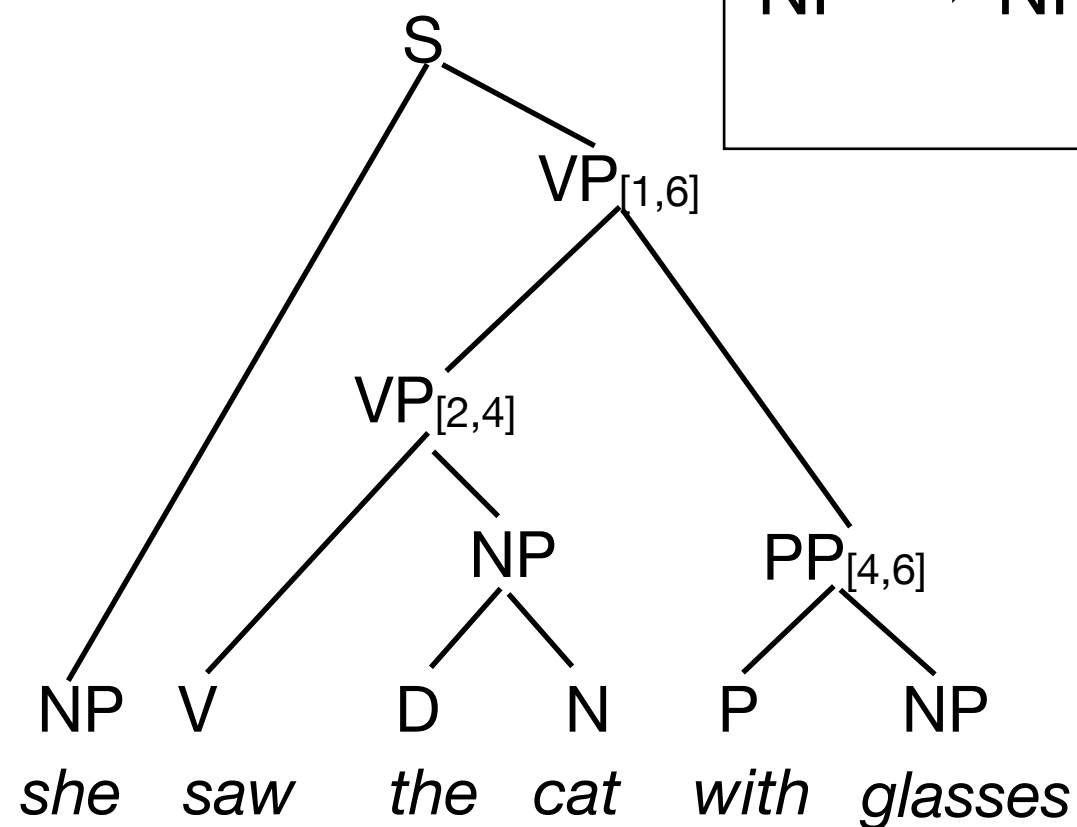
Lecture 7: Parsing with Context Free Grammars II. CKY for PCFGs. Earley Parser.

10/02/2018

COMS W4705

Daniel Bauer

Recall: Syntactic Ambiguity

S	→ NP VP	NP	→ she
VP	→ V NP	NP	→ glasses
VP	→ VP PP	D	→ the
PP	→ P NP	N	→ cat
NP	→ D N	N	→ glasses
NP	→ NP PP	V	→ saw
		P	→ with

Which parse tree is "better"? More probable?

Probabilities for Parse Trees

- Let $\mathcal{T}_G$ be the (possibly infinite) set of all parse trees generated by grammar G.

- We want a model that assigns a probability to each parse tree, such that $\sum_{t \in \mathcal{T}_G} P(t) = 1$.

- We can use this model to select the most probable parse tree compatible with an input sentence.

  - This is another example of a generative model!

Selecting Parse Trees

- Let $\mathcal{T}_G(s)$ be the set of trees generated by grammar G whose *yield* (sequence of leafs) is string s.

- The most likely parse tree produced by G for string s is

$$\arg\max_{t \in \mathcal{T}_G(s)} P(t)$$

  - How do we define *P(t)?*

  - How do we learn such a model from training data (annotated or un-annotated).

  - How do we find the highest probability tree for a given sentence? (*parsing/decoding*)

Probabilistic Context Free Grammars (PCFG)

- A PCFG consists of a Context Free Grammar $G=(N, \Sigma, R, S)$ and a probability $P(A \rightarrow \beta)$ for each production $A \rightarrow \beta \in R$.

  - The probabilities for all rules with the same left-hand-side sum up to 1:

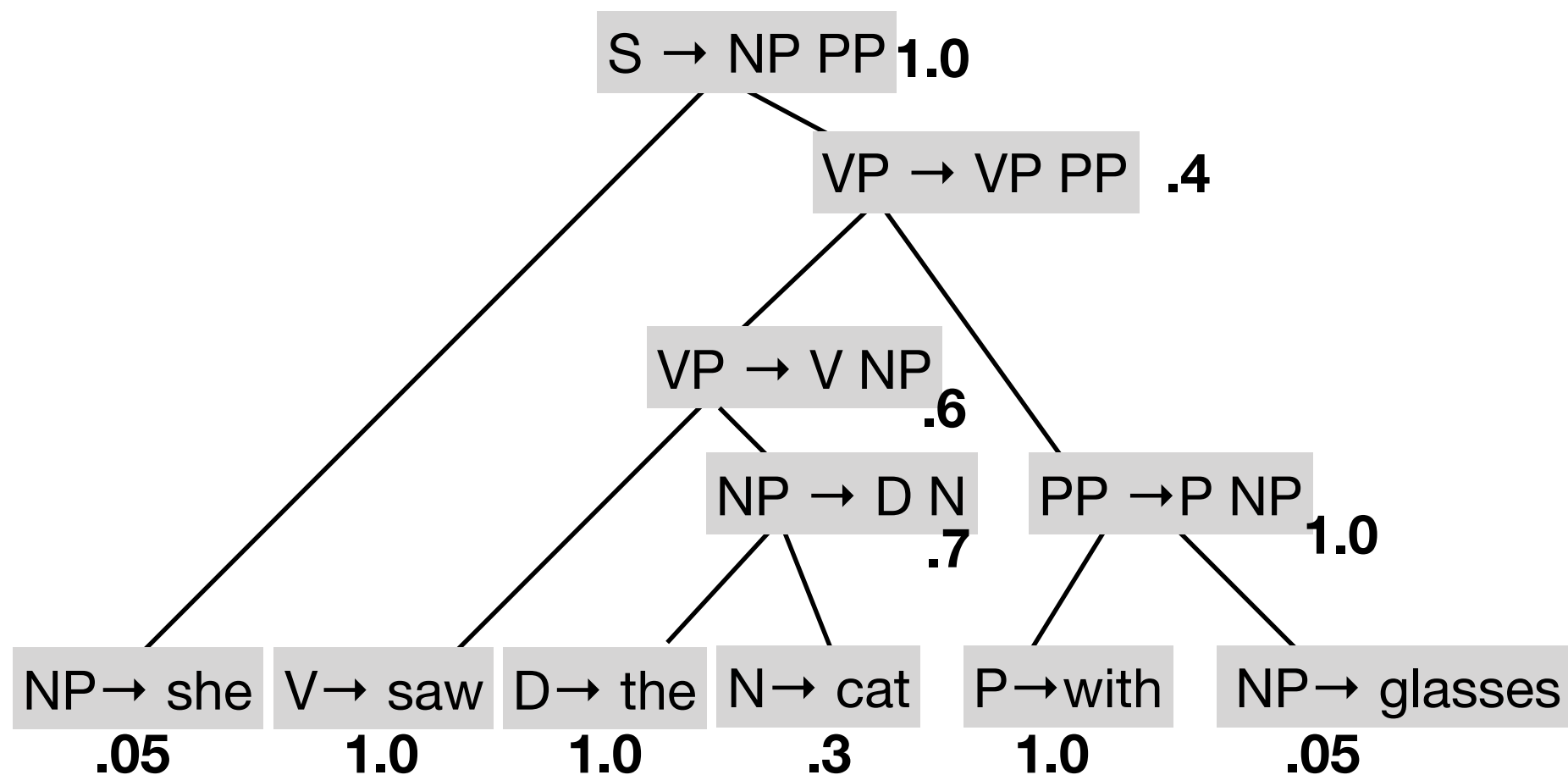$$\sum_{A \rightarrow \beta : A = X} P(A \rightarrow \beta) = 1 \text{ for all } X \in N$$

  - Think of this as the conditional probability for $A \rightarrow \beta$, given the left-hand-side nonterminal A.

PCFG Example

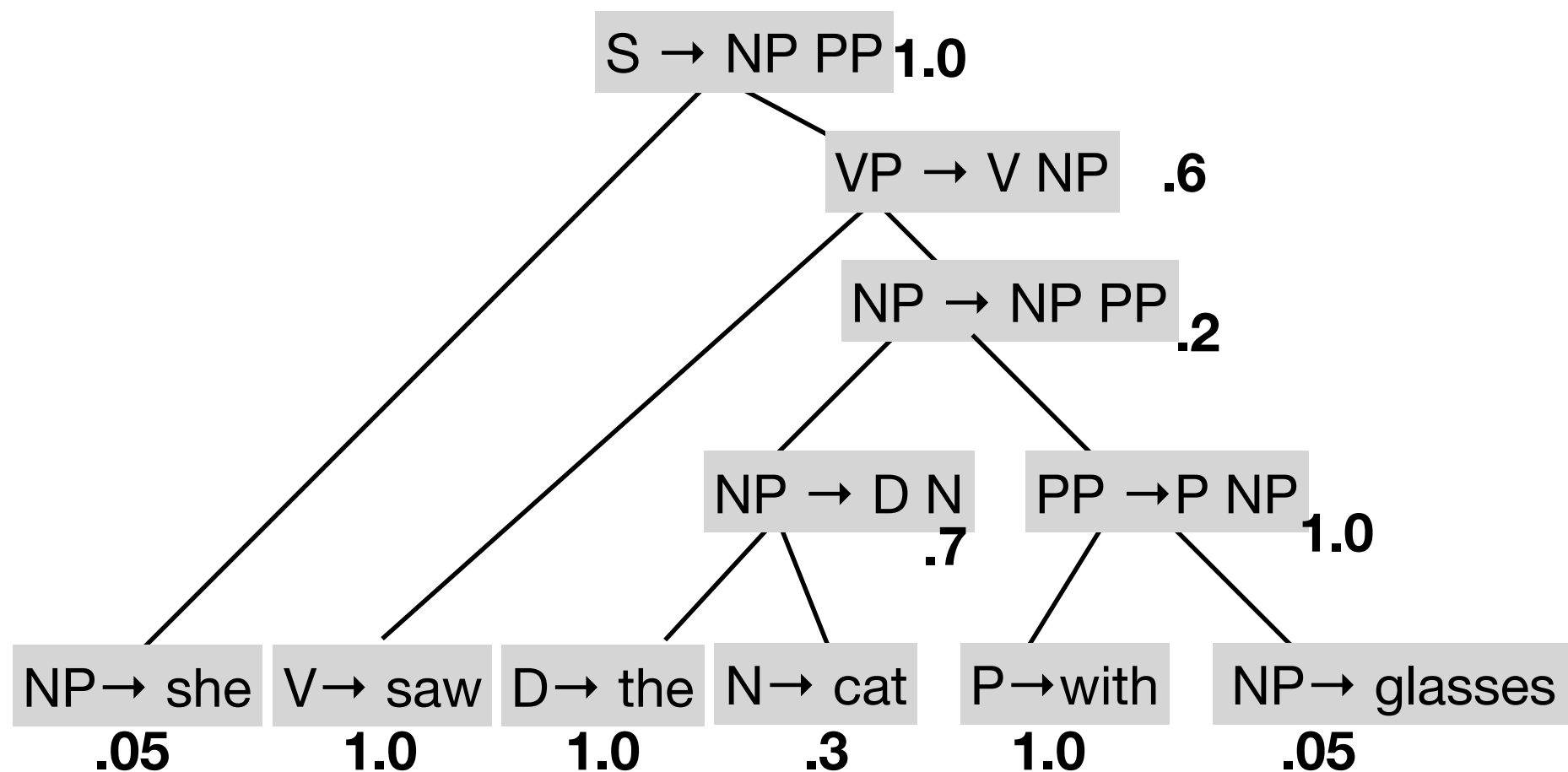|		
S → NP VP	[1.0]	NP → she [0.05]
VP → V NP	[0.6]	NP → glasses [0.05]
VP → VP PP	[0.4]	D → the [1.0]
PP → P NP	[1.0]	N → cat [0.3]
NP → D N	[0.7]	N → glasses [0.7]
NP → NP PP	[0.2]	V → saw [1.0]
		P → with [1.0]

Parse Tree Probability

- Given a parse tree $t \in \mathcal{T}_G$, containing rules $A_1 \to \beta_1, \ldots, A_n \to \beta_n$ the probability of t is

$$P(t) = \prod_{i=1}^{n} P(A_i \to \beta_i)$$

$$S \to NP\ PP \quad \textbf{1.0}$$
$$VP \to VP\ PP \quad \textbf{.4}$$
$$VP \to V\ NP \quad \textbf{.6}$$
$$NP \to D\ N \quad \textbf{.7}$$
$$PP \to P\ NP \quad \textbf{1.0}$$
$$NP \to she \quad \textbf{.05}$$
$$V \to saw \quad \textbf{1.0}$$
$$D \to the \quad \textbf{1.0}$$
$$N \to cat \quad \textbf{.3}$$
$$P \to with \quad \textbf{1.0}$$
$$NP \to glasses \quad \textbf{.05}$$

1 x .05 x .4 x .6 x 1 x 0.7 x 1 x 0.3 x 1 x 1 x .05 = .000126

Parse Tree Probability

- Given a parse tree $t \in \mathcal{T}_G$, containing rules $A_1 \to \beta_1, \ldots, A_n \to \beta_n$ the probability of t is

$$P(t) = \prod_{i=1}^{n} P(A_i \to \beta_i)$$

S → NP PP **1.0**

VP → V NP **.6**

NP → NP PP **.2**

NP → D N **.7** PP →P NP **1.0**

NP→ she **.05** V→ saw **1.0** D→ the **1.0** N→ cat **.3** P→with **1.0** NP→ glasses **.05**

1 x .05 x .6 x 1 x .2 x .7 x 1 x .3 x 1 x 1 x .05 = 0.000063 < 0.000126

Estimating PCFG probabilities

- Supervised training: We can estimate PCFG probabilities from a *treebank,* a corpus manually annotated with constituency structure using maximum likelihood estimates:

$$P(A \rightarrow \beta) = \frac{count(A \rightarrow \beta)}{count(A)}$$

- Unsupervised training:

  - What if we have a grammar and a corpus, but no annotated parses?

  - Can use the **inside-outside** algorithm for parsing and do EM estimation of the probabilities (not discussed in this course)

The Penn Treebank

- Syntactically annotated corpus of newspaper text (1989 Wall Street Journal Articles).

- The source text is naturally occurring but the treebank is not:

  - Assumes a specific linguistic theory (although a simple one).

  - Very flat structure (NPs, Ss, VPs).

PTB Example

```
( (S (NP-SBJ (NP (NNP Pierre) (NNP Vinken))
      (, ,)
      (ADJP (NML (CD 61) (NNS years))
      (JJ old))
      (, ,))
    (VP (MD will)
  (VP (VB join)
      (NP (DT the) (NN board))
      (PP-CLR (IN as)
        (NP (DT a) (JJ nonexecutive) (NN director)))
      (NP-TMP (NNP Nov.) (CD 29))))
    (. .)))
```
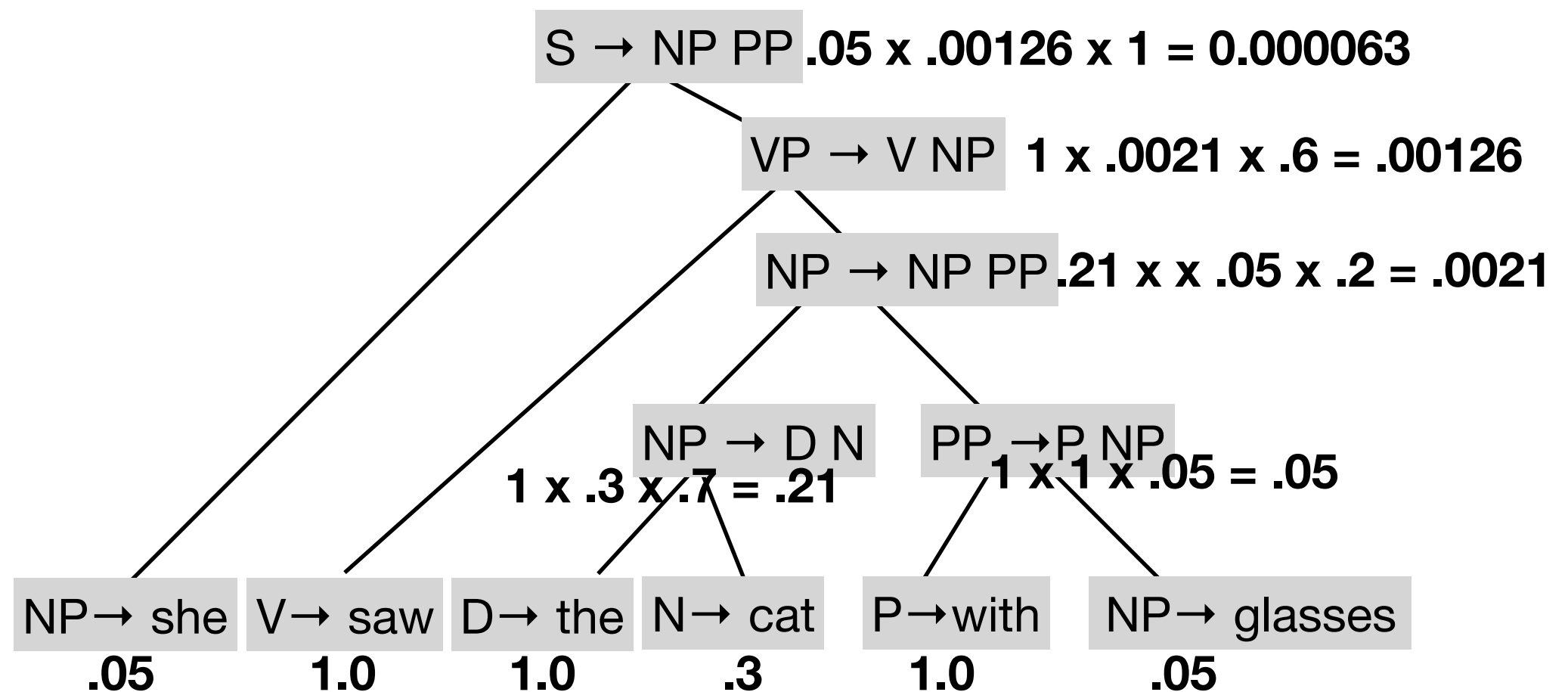
PTB Example
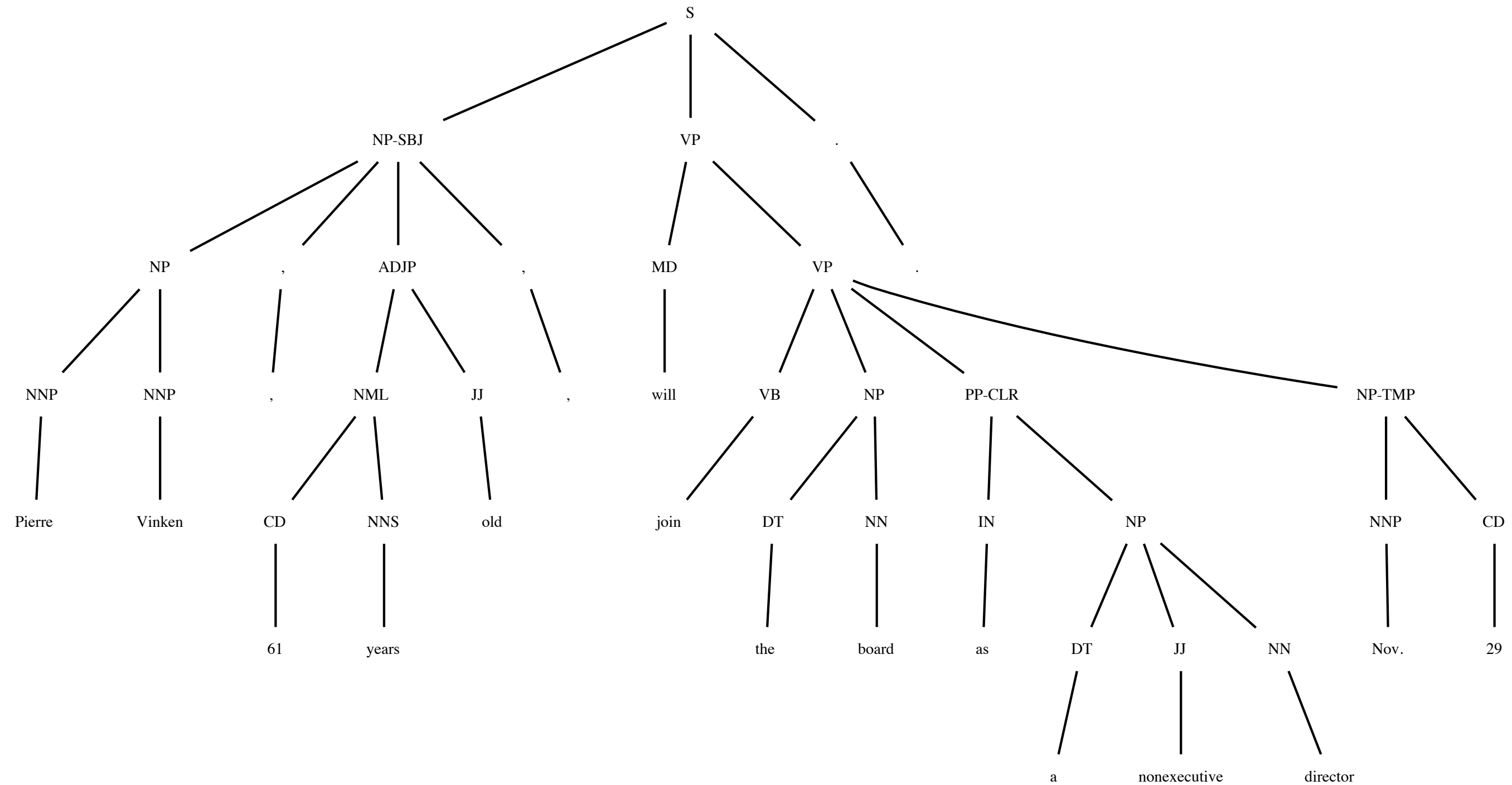
Parsing with PCFG

- We want to use PCFG to answer the following questions:

  - What is the total probability of the sentence under the PCFG?

  - What is the most probable parse tree for a sentence under the PCFG? *(decoding/parsing)*

- We can modify the CKY algorithm.
  Basic idea: Compute these probabilities bottom-up using dynamic programming.

Computing Probabilities Bottom-Up

S → NP PP **.05 x .00126 x 1 = 0.000063**

VP → V NP **1 x .0021 x .6 = .00126**

NP → NP PP **.21 x x .05 x .2 = .0021**

NP → D N **1 x .3 x .7 = .21**

PP→P NP **1 x 1 x .05 = .05**

NP→ she **.05**

V→ saw **1.0**

D→ the **1.0**

N→ cat **.3**

P→with **1.0**

NP→ glasses **.05**

CKY for PCFG Parsing

- Let $T_{\mathcal{G}}(s, A)$ be the set of trees generated by grammar G starting at nonterminal A, whose *yield* is string s

- Use a chart π so that $\pi[i,j,A]$ contains the probability of the highest probability parse tree for string s[i,j] starting in nonterminal A.

$$\pi[i, j, A] = \max_{t \in T_{\mathcal{G}}(s[i,j],A)} P(t)$$

- We want to find $\pi[0,lenght(s),S]$ -- the probability of the highest-scoring parse tree for s rooted in the start symbol S.

CKY for PCFG Parsing

- To compute $\pi[0,lenght(s),S]$ we can use the following recursive definition:

Base case:
$$\pi[i, i + 1, A] = \begin{cases} P(A \to s_i) & \text{if } A \to s_i \in R \\ 0 & \text{otherwise} \end{cases}$$

$$\pi[i, j, A] = \max_{\substack{k=i+1\ldots j-1, \\ A \to BC \in R}} P(A \to BC) \cdot \pi[i, k, B] \cdot \pi[k, j, C]$$

- Then fill the chart using dynamic programming.

CKY for PCFG Parsing

- **Input:** PCFG $G=(N, \Sigma, R, S)$, input string s of length n.

- for i=0…n-1:   initialization

$$\pi[i, i+1, A] = \begin{cases} P(A \to s_i) & \text{if } A \to s_i \in R \\ 0 & \text{otherwise} \end{cases}$$

- for *length=2…n*:   main loop
    for *i=0…(n-length)*:
    j = i+length
    for *k=i+1…j-1*:
        for $A \in N$:
        $$\pi[i, j, A] = \max_{\substack{k=i+1\ldots j-1, \\ A \to BC \in R}} P(A \to BC) \cdot \pi[i, k, B] \cdot \pi[k, j, C]$$

Use **backpointers** to retrieve the highest-scoring parse tree (see previous lecture).

Probability of a Sentence

- What if we are interested in the probability of a sentence, **not** of a single parse tree (for example, because we want to use the PCFG as a language model).

- Problem: Spurious ambiguity. Need to sum the probabilities of **all** parse trees for the sentence.

- How do we have to change CKY to compute this?

$$\pi[i, j, A] = \sum_{\substack{k=i+1\ldots j-1, \\ A \to BC \in R}} P(A \to BC) \cdot \pi[i, k, B] \cdot \pi[k, j, C]$$

Earley Parser

- CKY parser starts with words and builds parse trees bottom-up; requires the grammar to be in CNF.

- The Earley parser instead starts at the start symbol and tries to "guess" derivations top-down.

  - It discards derivations that are incompatible with the sentence.

  - The early parser sweeps through the sentence left-to-right only once. It keeps partial derivations in a table ("chart").

  - Allows arbitrary CFGs, no limitation to CNF.

Parser States

- Earley parser keeps track of partial derivations using parser **states / items.**

- State represent hypotheses about constituent structure based on the grammar, taking into account the input.

- Parser states are represented as **dotted rules with spans.**

  - The constituents to the left of the · have already been seen in the input string *s* (corresponding to the span)

S → · NP VP [0,0]   *"According to the grammar, there may be an NP starting in position 0. "*

NP → D A · N [0,2]   *"There is a determiner followed by an adjective in s[0,2]"*

NP → NP PP · [3,8] *"There is a complete NP in s[3,8], consisting of an NP and PP"*

Earley Parser (sketch)

```
 S  → NP VP        V  → saw
VP  → V  NP        P  → with
VP  → VP PP        D  → the
PP  → P NP         N  → cat
NP  → D N          N  → tail
NP  → NP PP        N  → student
```

S → · NP VP [0,0]

NP → · NP PP [0,0]   NP → · D N [0,0]

D → · the [0,0]

Three parser operations:

1. **Predict** new subtrees top-down.

0 *the* 1 *student* 2 *saw* 3 *the* 4 *cat* 5 *with* 6 *the* 7 *tail*

Earley Parser (sketch)

S → NP VP
VP → V NP
VP → VP PP
PP → P NP
NP → D N
NP → NP PP

V → saw
P → with
D → the
N → cat
N → tail
N → student

S → · NP VP [0,0]

NP → · NP PP [0,0]   NP → · D N [0,0]

D → the · [0,1]

Three parser operations:

1. Predict new subtrees top-down.

2. **Scan** input terminals.

0 *the* 1 *student* 2 *saw* 3 *the* 4 *cat* 5 *with* 6 *the* 7 *tail*

Earley Parser (sketch)

S → NP VP
VP → V NP
VP → VP PP
PP → P NP
NP → D N
NP → NP PP

V → saw
P → with
D → the
N → cat
N → tail
N → student

S → · NP VP [0,0]

NP → · NP PP [0,0]   NP → · D N [0,0]

D → the · [0,1]   passive state

Three parser operations:

1. Predict new subtrees top-down.

2. **Scan** input terminals.

0 *the* 1 *student* 2 *saw* 3 *the* 4 *cat* 5 *with* 6 *the* 7 *tail*

Earley Parser (sketch)

S → NP VP
VP → V NP
VP → VP PP
PP → P NP
NP → D N
NP → NP PP

V → saw
P → with
D → the
N → cat
N → tail
N → student

S → · NP VP [0,0]

NP → · NP PP [0,0]   NP → D · N [0,1]

D → the · [0,1]   passive state

Three parser operations:

1. Predict new subtrees top-down.

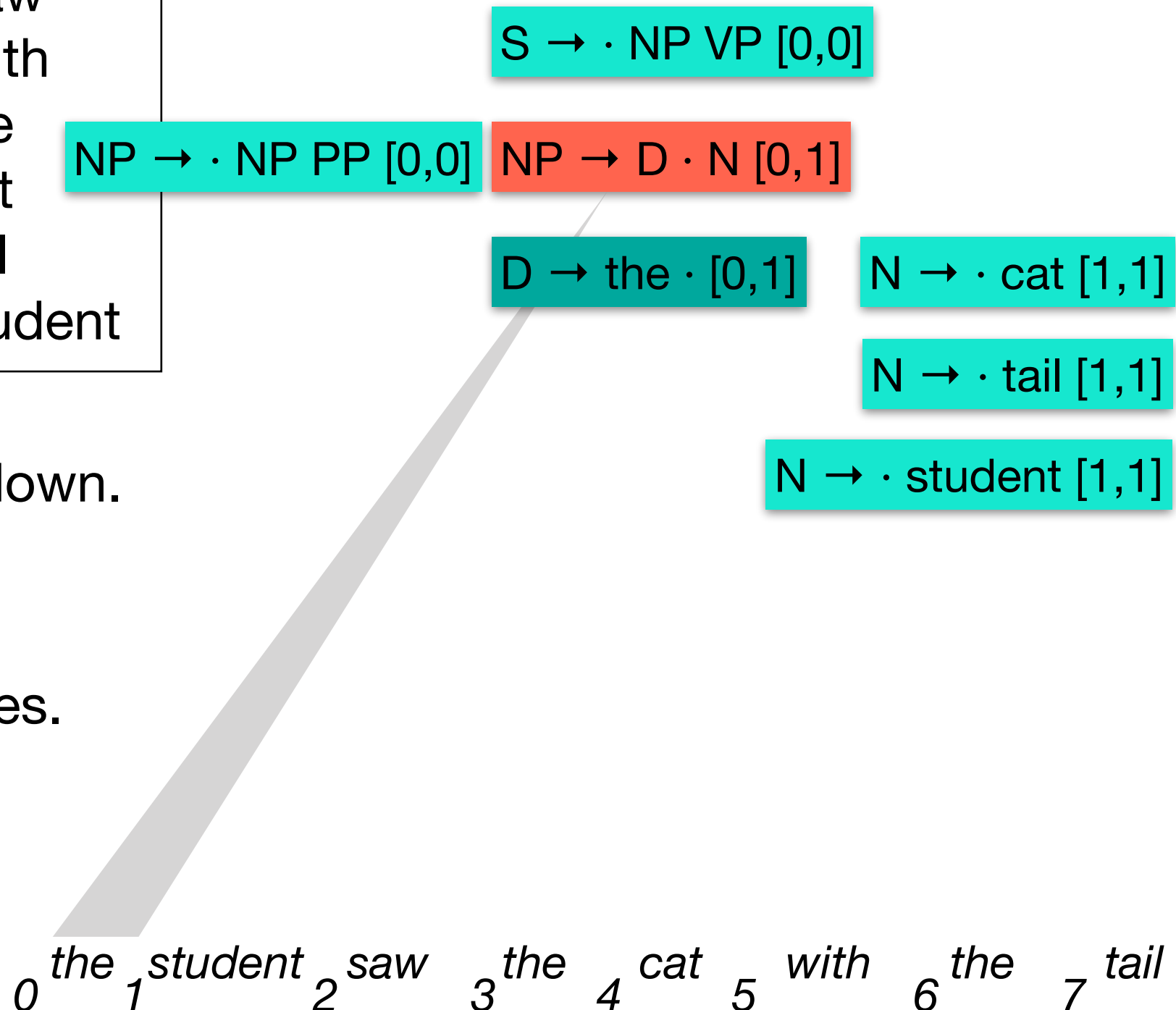2. Scan input terminals.

3. **Complete** with passive states.

0 *the* 1 *student* 2 *saw* 3 *the* 4 *cat* 5 *with* 6 *the* 7 *tail*

Earley Parser (sketch)

S → NP VP   V → saw
VP → V NP   P → with
VP → VP PP   D → the
PP → P NP   N → cat
NP → D N   N → tail
NP → NP PP   N → student

Three parser operations:

1. **Predict** new subtrees top-down.

2. Scan input terminals.

3. Complete with passive states.

S → · NP VP [0,0]

NP → · NP PP [0,0]   NP → D · N [0,1]

D → the · [0,1]   N → · cat [1,1]

N → · tail [1,1]

N → · student [1,1]

0 *the* 1 *student* 2 *saw* 3 *the* 4 *cat* 5 *with* 6 *the* 7 *tail*

Earley Parser (sketch)

S → NP VP   V → saw
VP → V NP   P → with
VP → VP PP   D → the
PP → P NP   N → cat
NP → D N   N → tail
NP → NP PP   N → student

Three parser operations:

1. Predict new subtrees top-down.

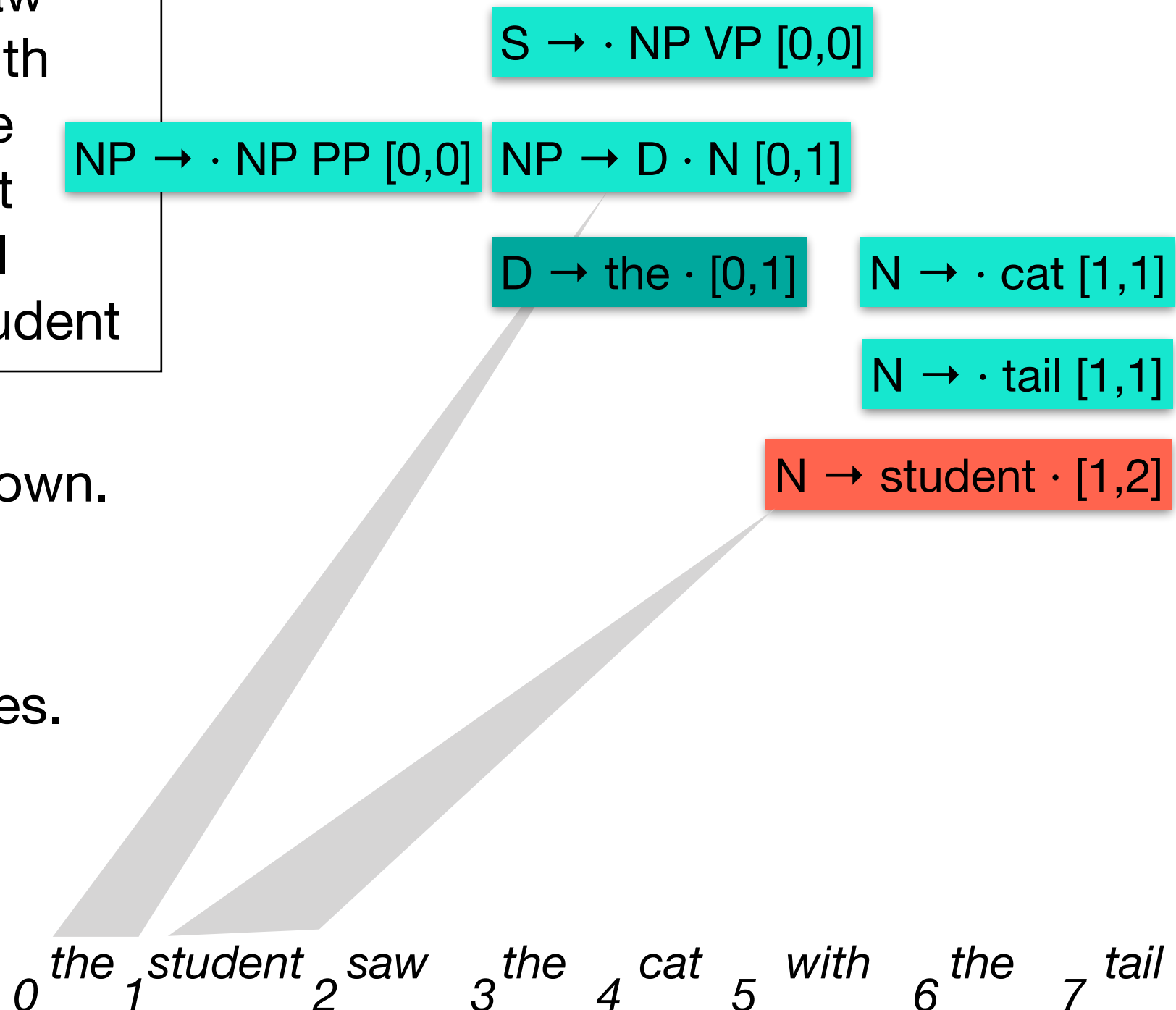2. **Scan** input terminals.

3. Complete with passive states.

S → · NP VP [0,0]

NP → · NP PP [0,0]   NP → D · N [0,1]

D → the · [0,1]   N → · cat [1,1]

N → · tail [1,1]

N → student · [1,2]

0 the 1 student 2 saw 3 the 4 cat 5 with 6 the 7 tail

Earley Parser (sketch)

S → NP VP
VP → V NP
VP → VP PP
PP → P NP
NP → D N
NP → NP PP

V → saw
P → with
D → the
N → cat
N → tail
N → student

Three parser operations:

1. Predict new subtrees top-down.

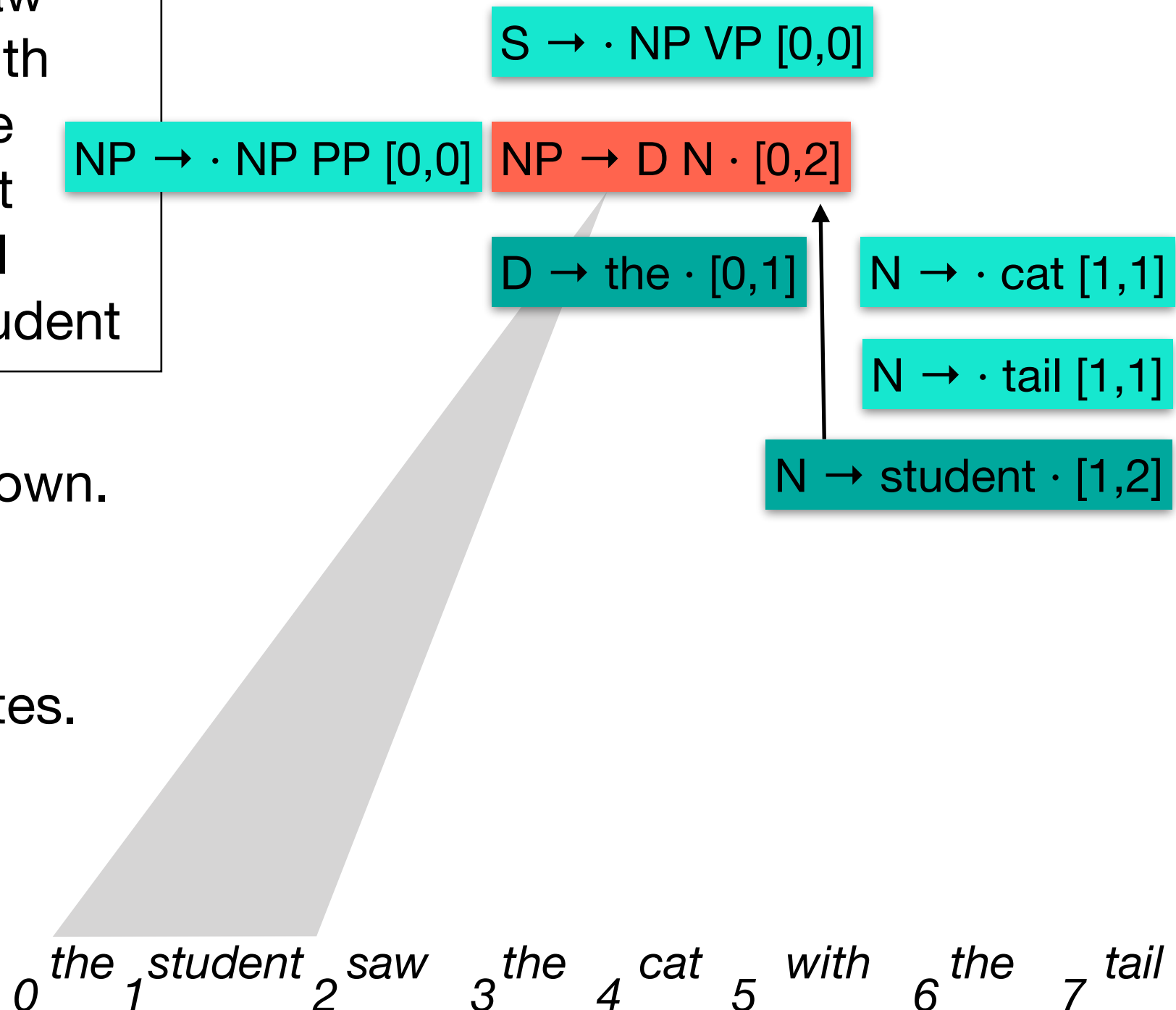2. Scan input terminals.

3. **Complete** with passive states.

S → · NP VP [0,0]

NP → · NP PP [0,0]   NP → D N · [0,2]

D → the · [0,1]   N → · cat [1,1]

N → · tail [1,1]

N → student · [1,2]

0 *the* 1 *student* 2 *saw* 3 *the* 4 *cat* 5 *with* 6 *the* 7 *tail*

Earley Parser (sketch)

S → NP VP
VP → V NP
VP → VP PP
PP → P NP
NP → D N
NP → NP PP

V → saw
P → with
D → the
N → cat
N → tail
N → student

Three parser operations:

1. Predict new subtrees top-down.

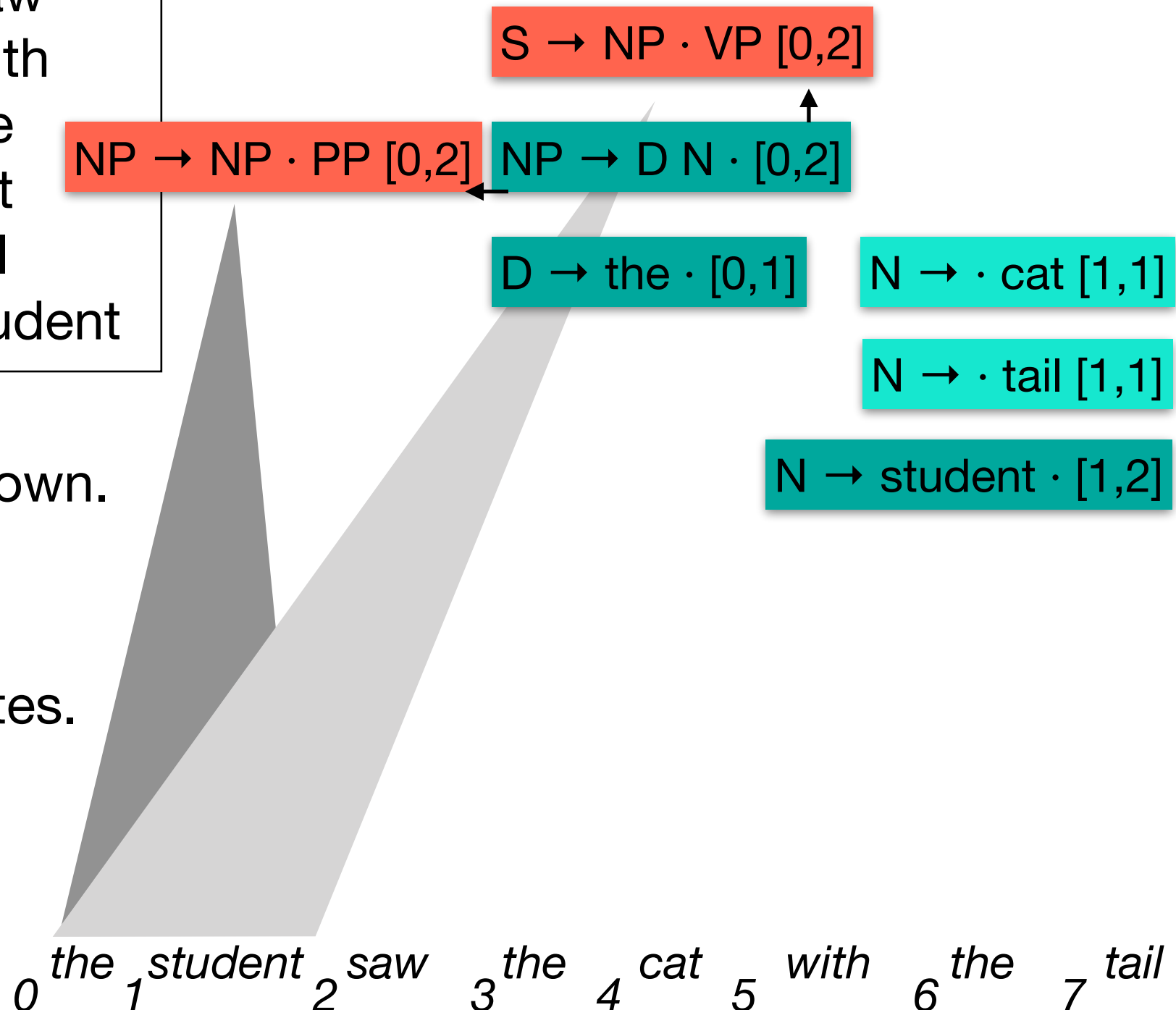2. Scan input terminals.

3. **Complete** with passive states.

S → NP · VP [0,2]

NP → NP · PP [0,2] NP → D N · [0,2]

D → the · [0,1] N → · cat [1,1]

N → · tail [1,1]

N → student · [1,2]

0 *the* 1 *student* 2 *saw* 3 *the* 4 *cat* 5 *with* 6 *the* 7 *tail*

Earley Algorithm

- Keep track of parser states in a table ("chart"). *Chart[k]* contains a set of all parser states that end in position k.

- **Input:** Grammar $G=(N, \Sigma, R, S)$, input string s of length n.

- **Initialization:** For each production $S \rightarrow \alpha \in R$
    add a state $S \rightarrow \cdot\alpha[0,0]$ to Chart[0].

- for $i = 0$ to n:

    - for each *state* in Chart[i]:

        - if *state* is of form $A \rightarrow \alpha \cdot s[i] \beta [k,i]$:
          scan(*state*)

        - elif *state* is of form $A \rightarrow \alpha \cdot B \beta [k,i]$:
          predict(*state*)

        - else: // *state* is of form $A \rightarrow \alpha \cdot [k,i]$
          complete(*state*)

Earley Algorithm - Scan

- The scan operation can only be applied to a state if the dot is in front of a terminal symbol that matches the next input terminal.

- function **scan(**state**):**   *// state* is of form $A \rightarrow \alpha \cdot s[i] \beta [k,i]$

    - Add a new state $A \rightarrow \alpha s[i] \cdot \beta [k,i+1]$
      to $Chart[i+1]$

Earley Algorithm - Predict

- The predict operation can only be applied to a state if the dot is in front of a non-terminal symbol.

- function **predict(**state**):** *// state* is of form $A \rightarrow \alpha \cdot B \, \beta \, [k,i]$:

  - Add a new state $B \rightarrow \cdot \, \gamma \, [i,i]$
    to *Chart[i]*

- Note that this modifies Chart[i] **while** the algorithm is looping through it.

- No duplicate states are added (Chart[i] is a set)

Earley Algorithm - Complete

- The complete operation may only be applied to a passive item.

- function **complete(**state**):**   // *state* is of form A $\rightarrow \alpha \cdot$ *[k,j]*

  - for each state $B \rightarrow \beta \cdot A \gamma$ *[i,k]* add a new state
    $B \rightarrow \beta A \cdot \gamma$ *[i,j]* to Chart[j]

- Note that this modifies Chart[i] **while** the algorithm is looping through it.

- Note that it is important to make a copy of the old state before moving the dot.

- This operation is similar to the combination operation in CKY!

Earley Algorithm - Runtime

- The runtime depends on the number of items in the chart (each item is "visited" exactly once).

- We proceed through the input exactly once, which takes O(N).

  - For each position on the chart, there are O(N) possible split points where the dot could be.

  - Each complete operation can produce O(N) possible new items (with different starting points).

- Total: $O(N^3)$

Earley Algorithm - Some Observations

- How do we recover parse trees?

  - What happens in case of ambiguity?

    - Multiple ways to Complete the same state.

  - Keep back-pointers in the parser state objects.

  - Or use a separate data structure (CKY-style table or hashed states)

- How do we make the algorithm work with PCFG?

  - Easy to compute probabilities on Complete. Follow back pointer with max probability.