Syntax And Parsing

Jonathan May
CSCI 662
October 14/16, 2014

Credits/Further Reading

- Michael Collins' notes:
  - http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/pcfgs.pdf
  - http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/lexpcfgs.pdf
- Luke Zettlemoyer's/Dan Klein's notes:
  - http://homes.cs.washington.edu/~lsz/lectures/parsing.pdf
- Yoav Goldberg's notes:
  - http://u.cs.biu.ac.il/~89-680/parsing-and-cky.pdf
- Jason Eisner's notes:
  - http://u.cs.biu.ac.il/~89-680/eisner-parse.pdf
- Chris Manning's lectures:
  - http://nlp.stanford.edu/courses/lsa354/
- Klein and Manning on parsing:
  - http://www.cs.berkeley.edu/~klein/papers/unlexicalized-parsing.pdf

Syntax

Syntax

- A system of arrangement of words and phrases of (human) language

Syntax

- A system of arrangement of words and phrases of (human) language
- There's a lot of ways to communicate but we can still identify verbs, nouns, phrases acting like verbs and nouns, modifiers, etc.

Syntax

- A system of arrangement of words and phrases of (human) language
- There’s a lot of ways to communicate but we can still identify verbs, nouns, phrases acting like verbs and nouns, modifiers, etc.
- The structure helps us create sentences and inherently knowing the structure helps us interpret the unfamiliar.

Syntax

Syntax

- “Hexy planars garumphed by the snox three zamfirs ago.”

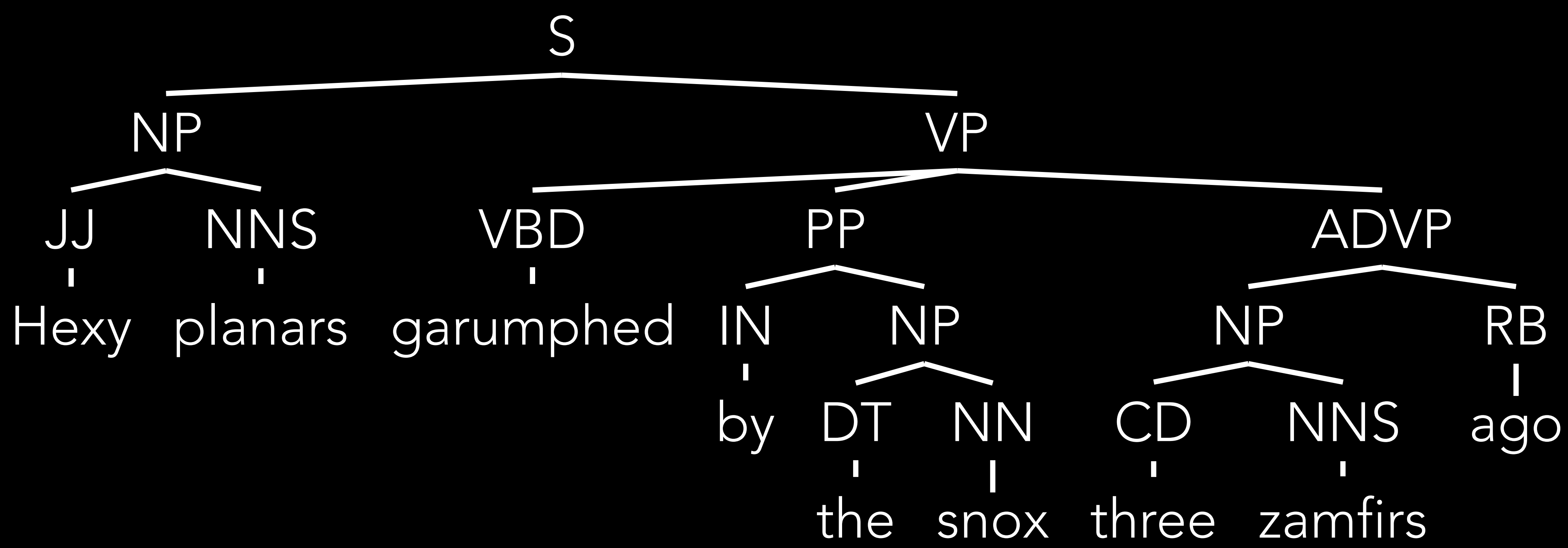
Syntax

- “Hexy planars garumphed by the snox three zamfirs ago.”
- What kind of planars are being discussed?

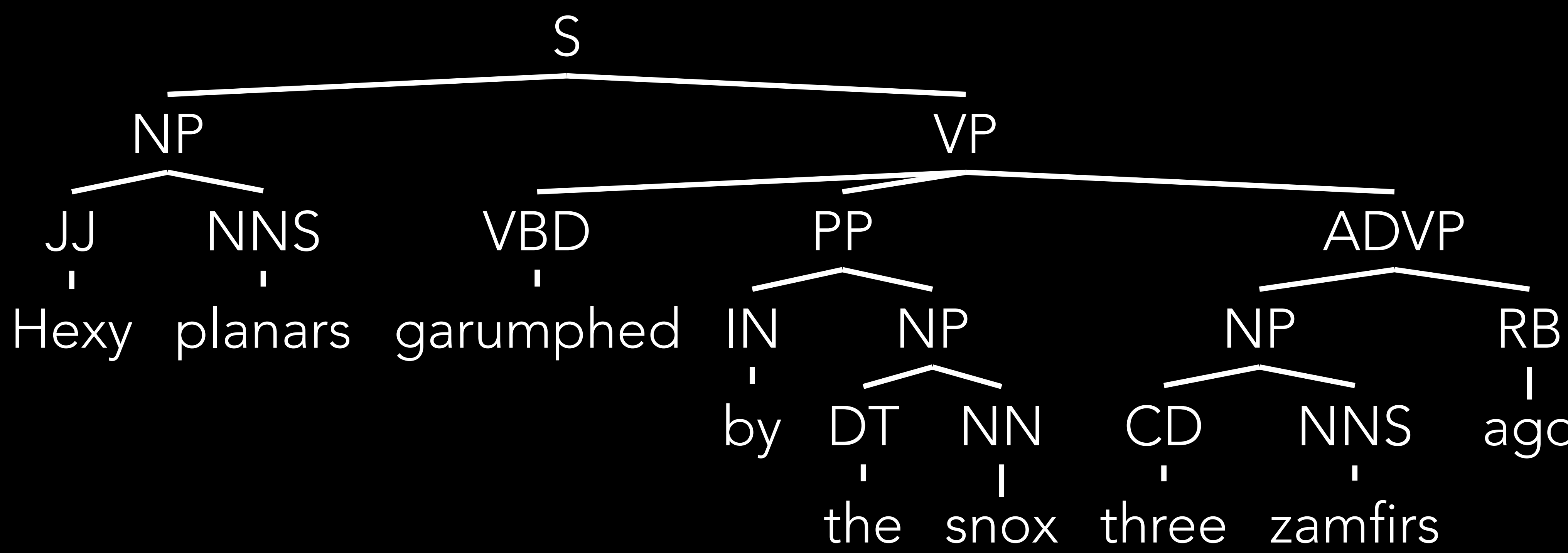
Syntax

- “Hexy planars garumphed by the snox three zamfirs ago.”
- What kind of planars are being discussed?
- Why do you know that?

Syntax

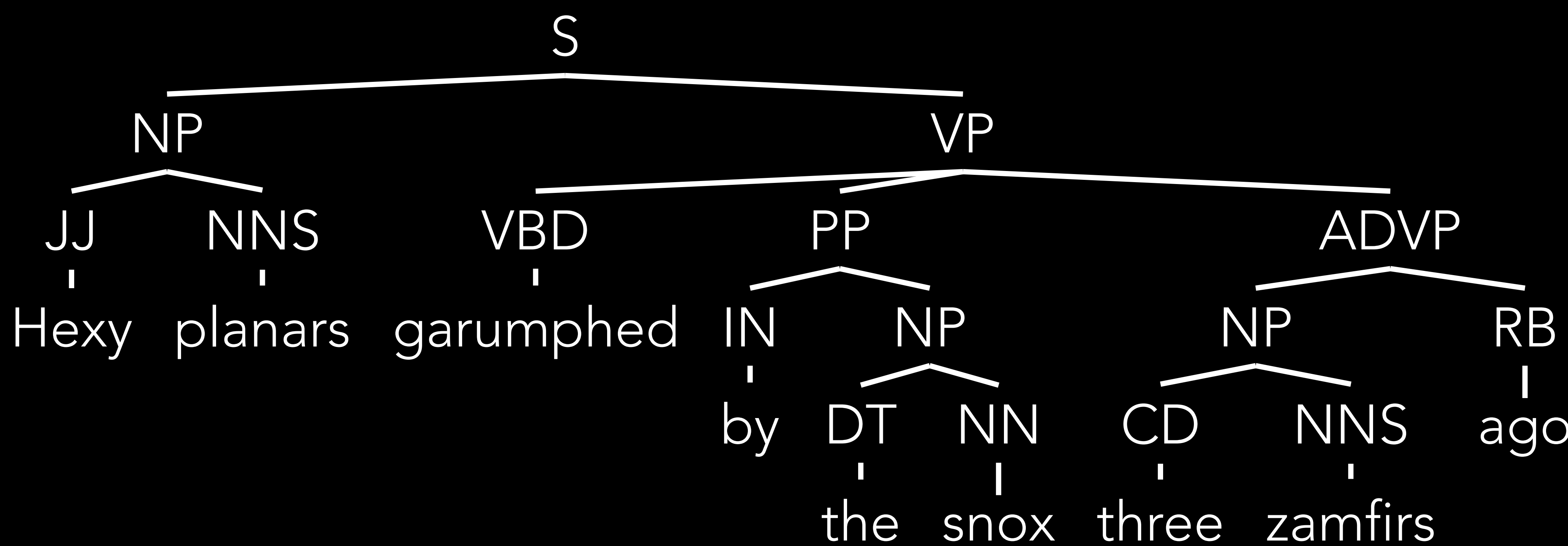

S
NP
VP
JJ
NNS
VBD
PP
ADVP
Hexy
planars
garumphed
IN
NP
NP
RB
by
DT
NN
CD
NNS
ago
the
snox
three
zamfirs

Syntax



- The characteristics of the words and their relative location to each other allow us to generate this interpretation even though we don't really understand the sentence

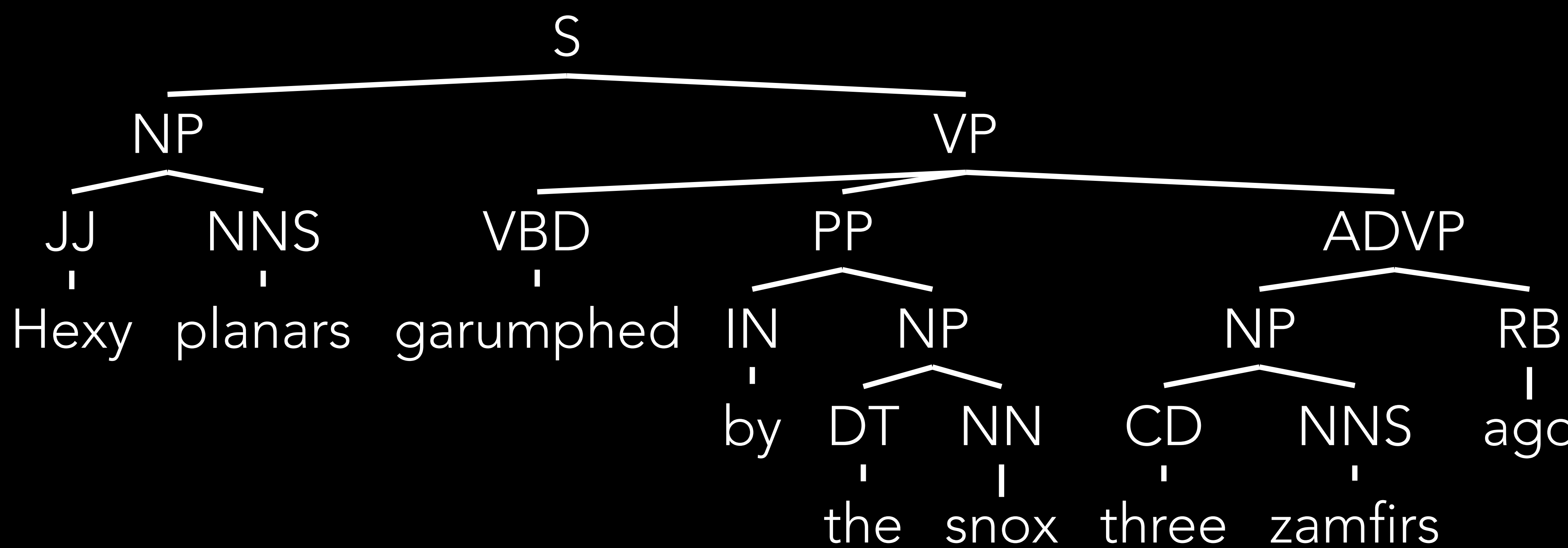
Syntax



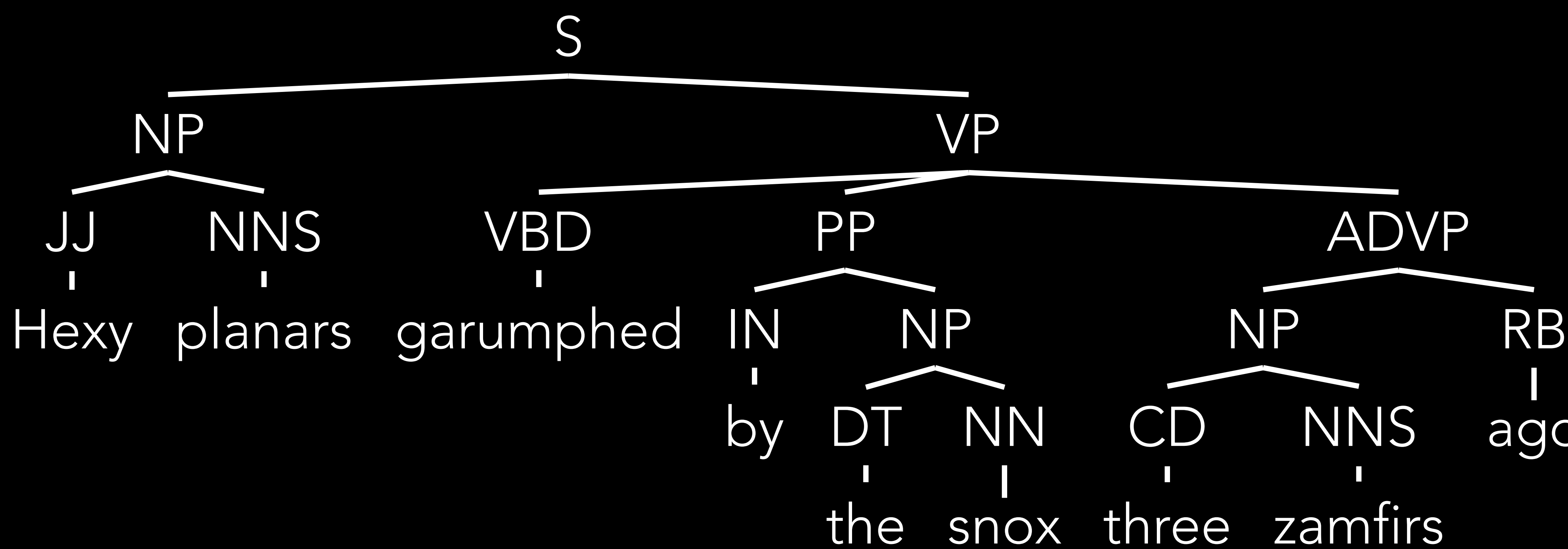
- The characteristics of the words and their relative location to each other allow us to generate this interpretation even though we don't really understand the sentence

- Goal of *parsing* is to write a computer program to generate this automatically.

Uses of Syntax

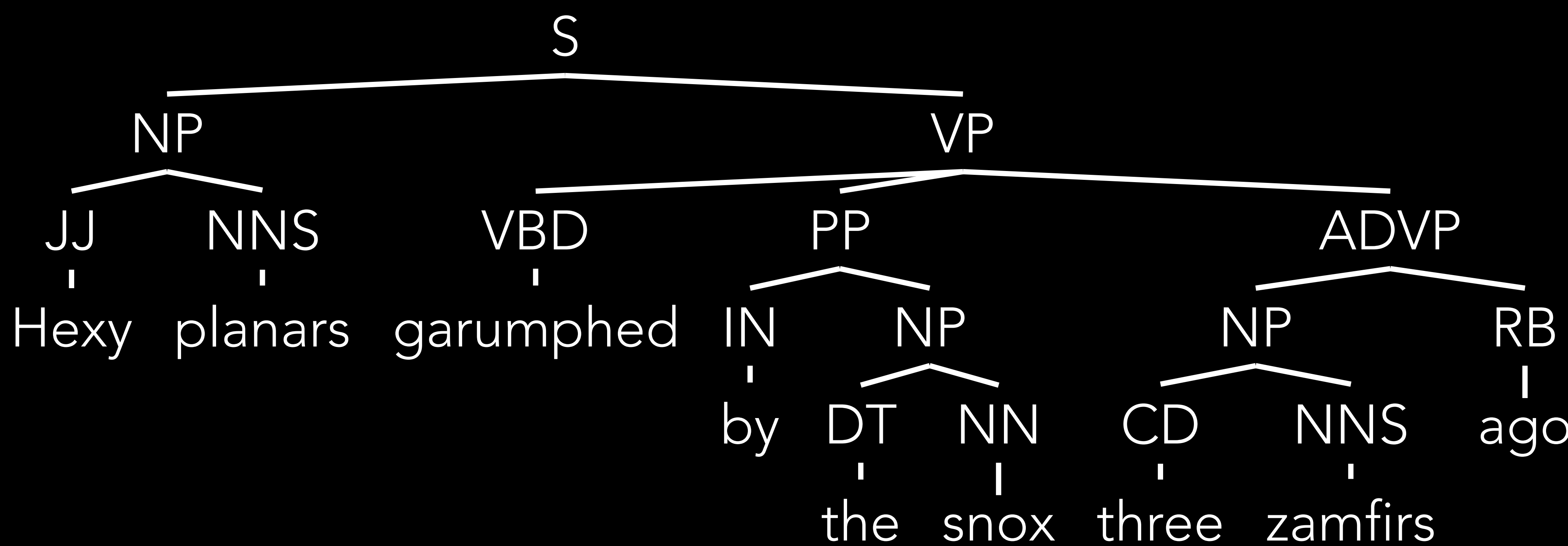


Uses of Syntax



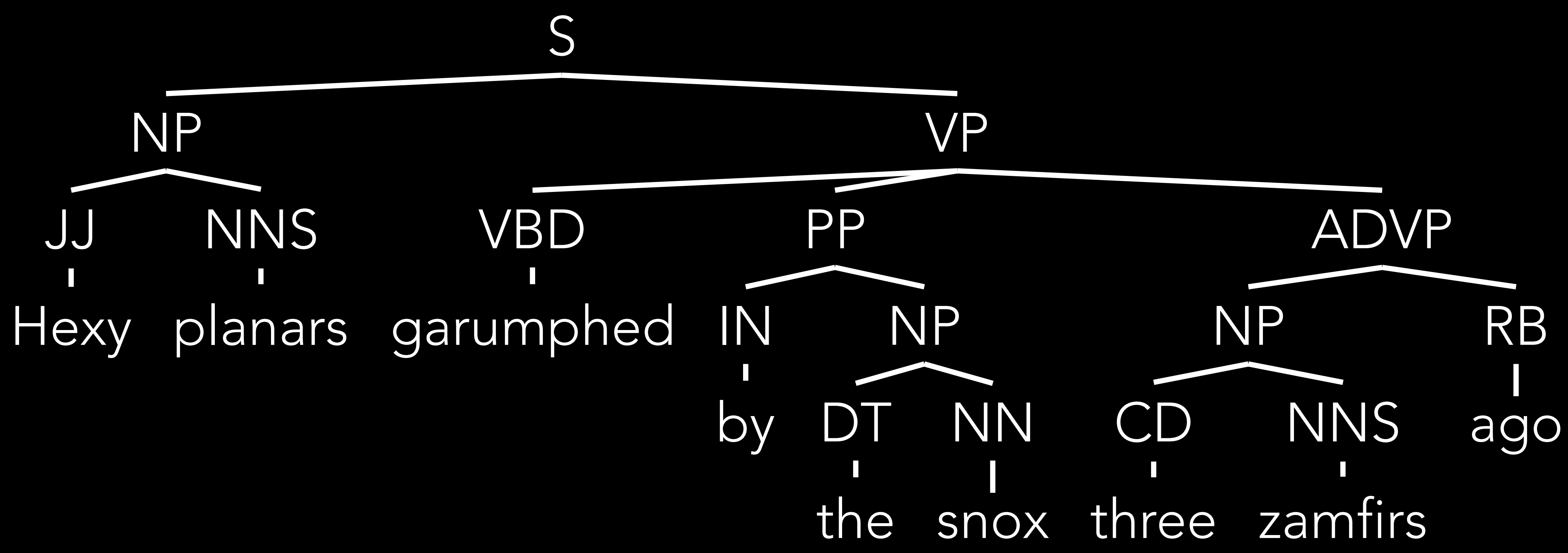
- Question answering (When did hexy planars garumph?)

Uses of Syntax



- Question answering (When did hexy planars garumph?)
- Sentence compression (Hexy planars garumphed.)

Uses of Syntax



- Question answering (When did hexy planars garumph?)
- Sentence compression (Hexy planars garumphed.)
- Machine translation (Planares hexos …)

Penn Treebank
(Marcus et al., ’93)

Penn Treebank (Marcus et al., ’93)

- Goal: investigate the syntactic phenomena that naturally occur

Penn Treebank (Marcus et al., ’93)

- Goal: investigate the syntactic phenomena that naturally occur
- 1 million words of English Wall Street Journal text in 40,000 sentences, annotated with syntactic structure

Penn Treebank (Marcus et al., ’93)

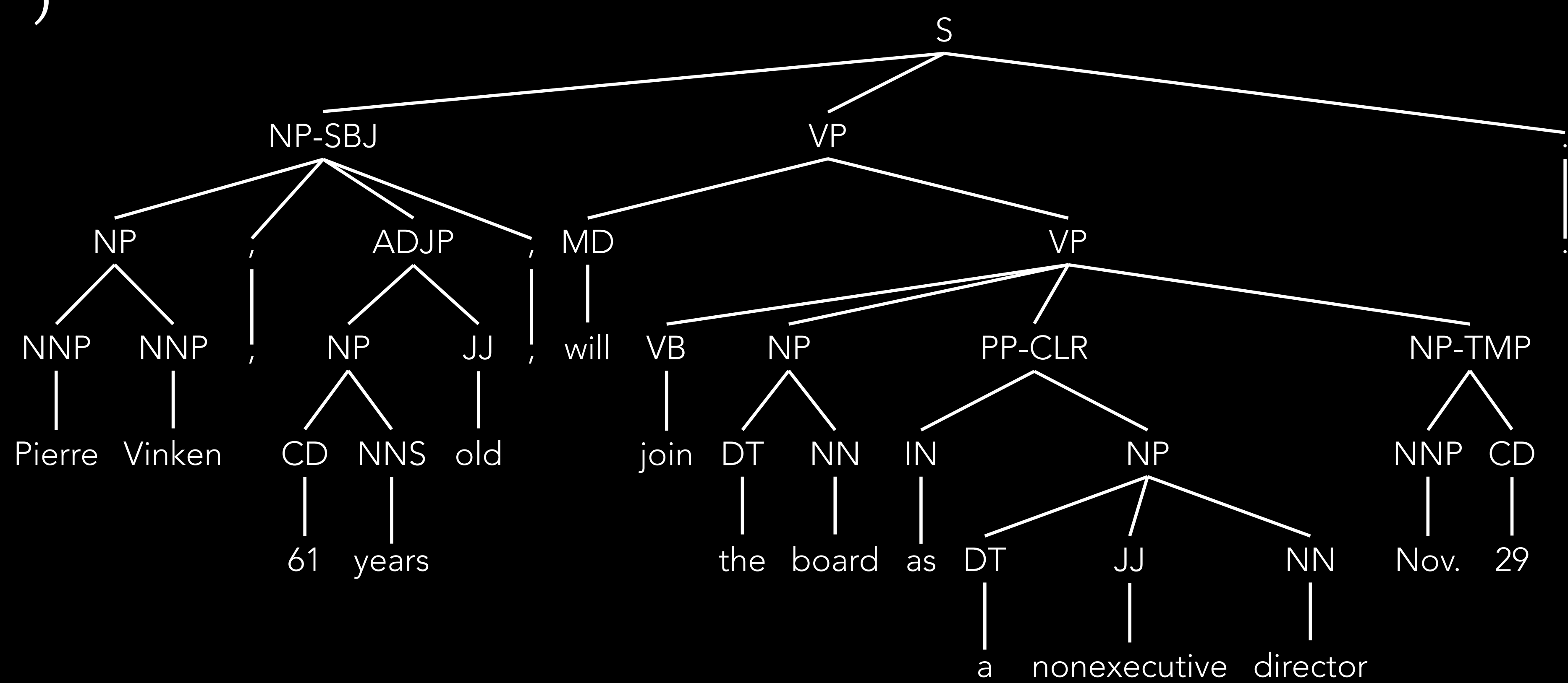
- Goal: investigate the syntactic phenomena that naturally occur
- 1 million words of English Wall Street Journal text in 40,000 sentences, annotated with syntactic structure
- $1/word

Penn Treebank (Marcus et al., ’93)

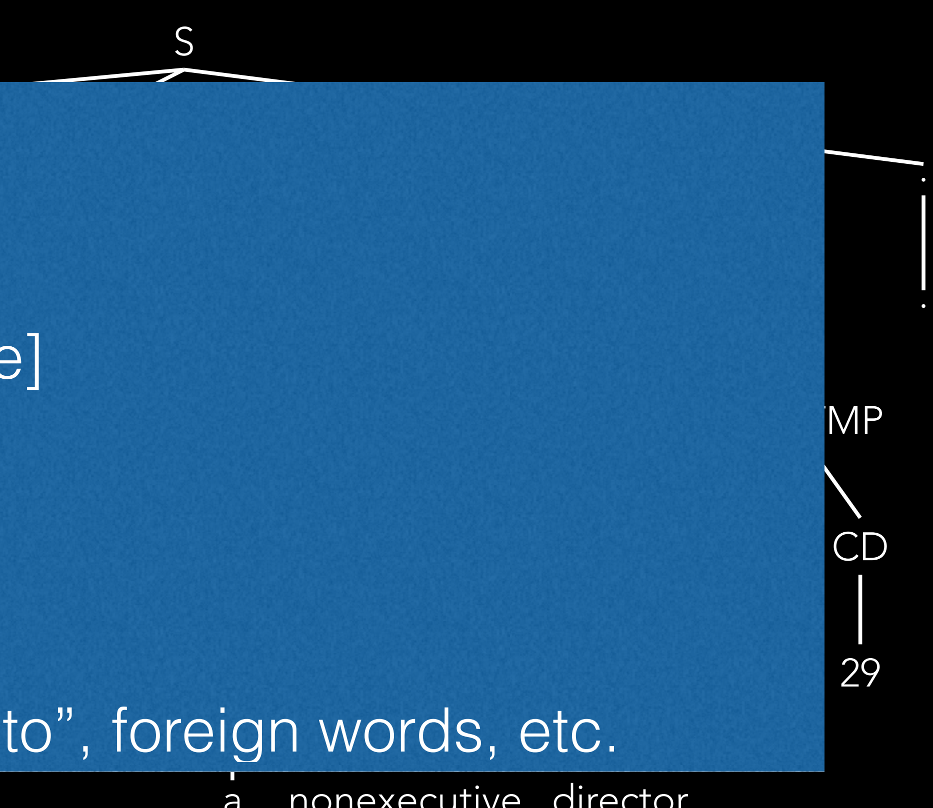
- Goal: investigate the syntactic phenomena that naturally occur
- 1 million words of English Wall Street Journal text in 40,000 sentences, annotated with syntactic structure
- $1/word
- Since then, treebanks have been created for many other languages

```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (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) )))
  (. .) )
```

```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (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) )))
  (. .) )
```



```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (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) )))
  (. .) )
```



36 POS (part of speech) tags:

NN[P][S] = noun [proper] [plural]
JJ[R|S] = adjective [comparative|superlative]
VB[D|G|N|P|Z] = verbs [various kinds]
MD = modal IN = preposition
CD = cardinal number
DT = determiner
punctuation, pronouns, adverbs, the word “to”, foreign words, etc.

```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (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) )))
  (. .) )
```

36 POS (part of speech) tags:

NN[P][S] = noun [proper] [plural]
JJ[R|S] = adjective [comparative|superlative]
VB[D|G|N|P|Z] = verbs [various kinds]
MD = modal IN = preposition
CD = cardinal number
DT = determiner
punctuation, pronouns, adverbs, the word “to”, foreign words, etc.

```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (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) )))
  (. .) )
```

36 POS (part of speech) tags:

NN[P][S] = noun [proper] [plural]
JJ[R|S] = adjective [comparative|superlative]
VB[D|G|N|P|Z] = verbs [various kinds]
MD = modal IN = preposition
CD = cardinal number
DT = determiner
punctuation, pronouns, adverbs, the word “to”, foreign words, etc.

```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (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) )))
  (. .) )
```

36 POS (part of speech) tags:

NN[P][S] = noun [proper] [plural]
JJ[R|S] = adjective [comparative|superlative]
VB[D|G|N|P|Z] = verbs [various kinds]
MD = modal IN = preposition
CD = cardinal number
DT = determiner
punctuation, pronouns, adverbs, the word “to”, foreign words, etc.

```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (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) )))
  (. .) )
```

36 POS (part of speech) tags:

NN[P][S] = noun [proper] [plural]
JJ[R|S] = adjective [comparative|superlative]
VB[D|G|N|P|Z] = verbs [various kinds]
MD = modal IN = preposition
CD = cardinal number
DT = determiner
punctuation, pronouns, adverbs, the word “to”, foreign words, etc.

```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (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) )))
  (. .) )
```

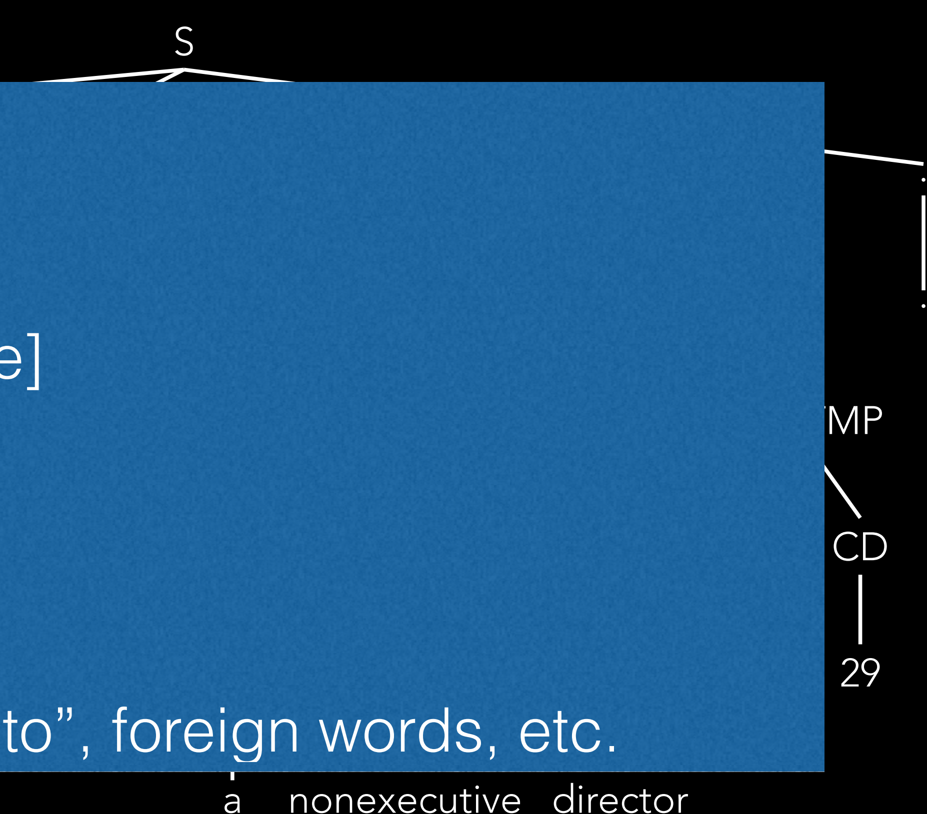
36 POS (part of speech) tags:

NN[P][S] = noun [proper] [plural]
JJ[R|S] = adjective [comparative|superlative]
VB[D|G|N|P|Z] = verbs [various kinds]
MD = modal IN = preposition
CD = cardinal number
DT = determiner
punctuation, pronouns, adverbs, the word “to”, foreign words, etc.

```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (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) )))
  (. .) )
```

36 POS (part of speech) tags:

NN[P][S] = noun [proper] [plural]
JJ[R|S] = adjective [comparative|superlative]
VB[D|G|N|P|Z] = verbs [various kinds]
MD = modal IN = preposition
CD = cardinal number
DT = determiner
punctuation, pronouns, adverbs, the word “to”, foreign words, etc.



```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (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) )))
  (. .) )
```

36 POS (part of speech) tags:

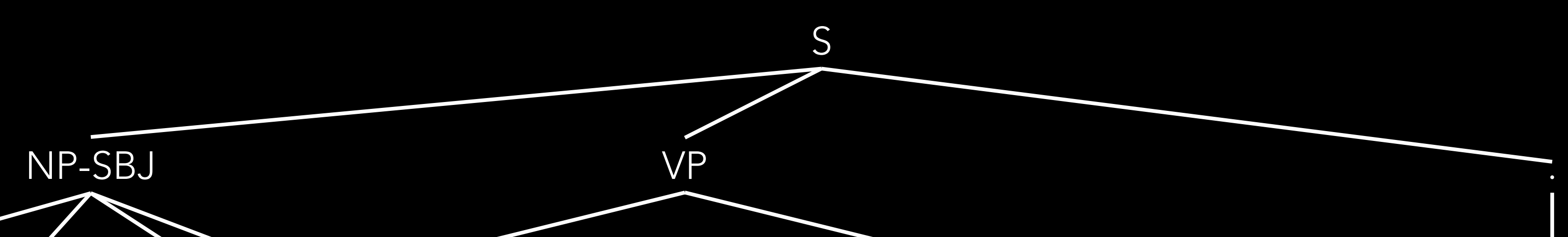
NN[P][S] = noun [proper] [plural]
JJ[R|S] = adjective [comparative|superlative]
VB[D|G|N|P|Z] = verbs [various kinds]
MD = modal IN = preposition
CD = cardinal number
DT = determiner
punctuation, pronouns, adverbs, the word “to”, foreign words, etc.

```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (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) )))
  (. .) )
```

36 POS (part of speech) tags:

NN[P][S] = noun [proper] [plural]
JJ[R|S] = adjective [comparative|superlative]
VB[D|G|N|P|Z] = verbs [various kinds]
MD = modal IN = preposition
CD = cardinal number
DT = determiner
punctuation, pronouns, adverbs, the word “to”, foreign words, etc.

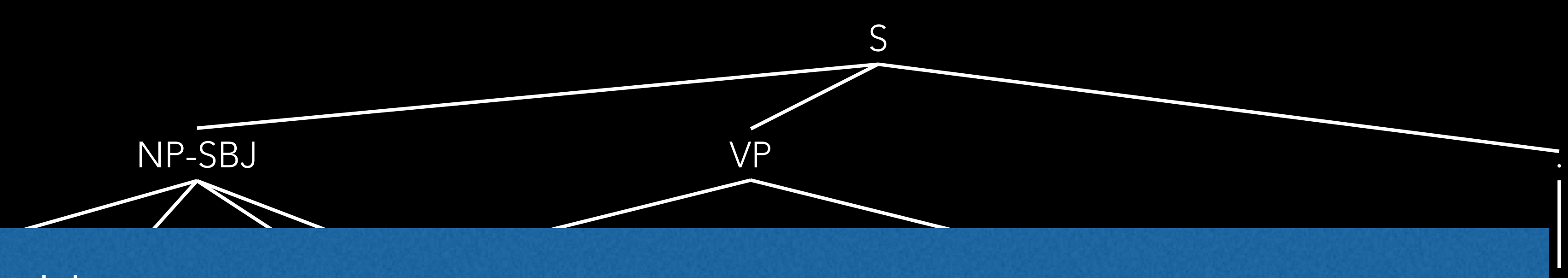
```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (CD 61) (NNS years) )
      (JJ old) )
    (, ,) )
  (VP (MD will)
    (VP (VB join)
      (NP (DT the) (NN board) )
      (PP-CLR (IN as)
        (NP (DT a) (JJ nonexecutive) (NN director) ))
      (NP-TMP (NNP Nov.)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



26 constituent tags:

S = declarative sentence NP = noun phrase
VP = verb phrase ADJP = adjective phrase
PP = prepositional phrase SBAR = sentential clause
SQ = question SINV = inverted sentence FRAG = fragment
…

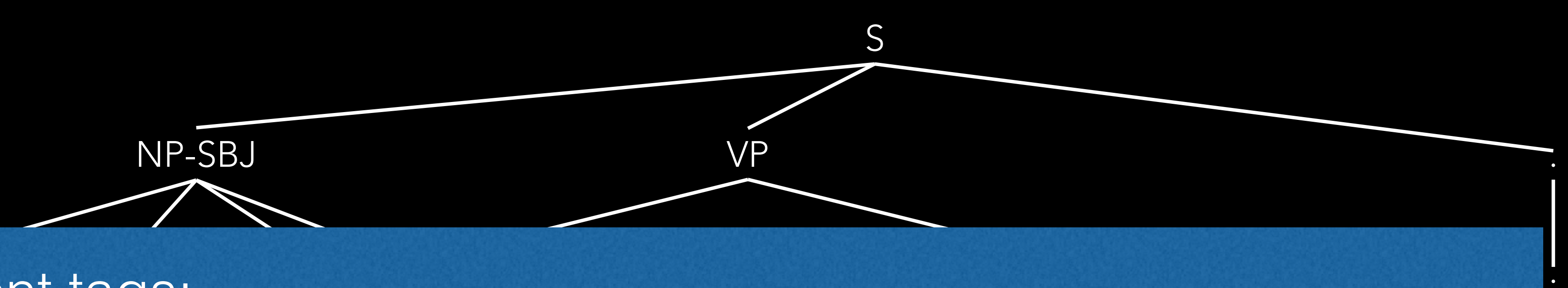
```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (CD 61) (NNS years) )
      (JJ old) )
    (, ,) )
  (VP (MD will)
    (VP (VB join)
      (NP (DT the) (NN board) )
      (PP-CLR (IN as)
        (NP (DT a) (JJ nonexecutive) (NN director) ))
      (NP-TMP (NNP Nov.)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



26 constituent tags:

S = declarative sentence NP = noun phrase
VP = verb phrase ADJP = adjective phrase
PP = prepositional phrase SBAR = sentential clause
SQ = question SINV = inverted sentence FRAG = fragment
…

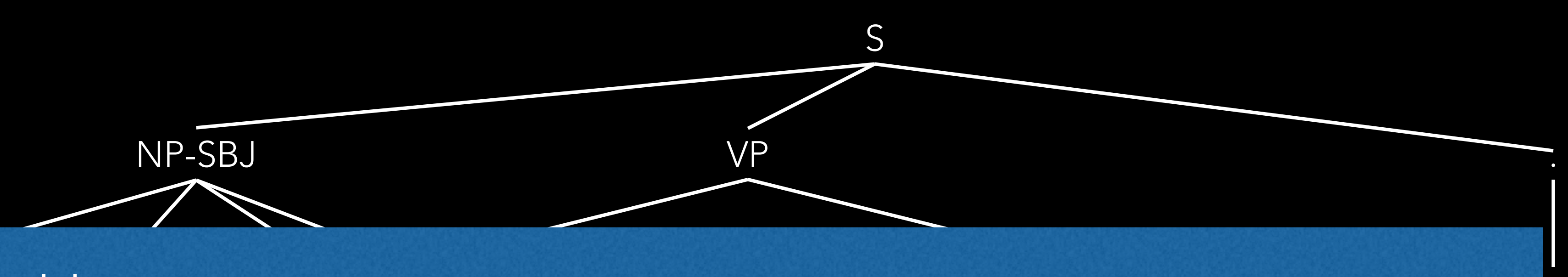
```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (CD 61) (NNS years) )
      (JJ old) )
    (, ,) )
  (VP (MD will)
    (VP (VB join)
      (NP (DT the) (NN board) )
      (PP-CLR (IN as)
        (NP (DT a) (JJ nonexecutive) (NN director) ))
      (NP-TMP (NNP Nov.)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



26 constituent tags:

S = declarative sentence NP = noun phrase
VP = verb phrase ADJP = adjective phrase
PP = prepositional phrase SBAR = sentential clause
SQ = question SINV = inverted sentence FRAG = fragment
…

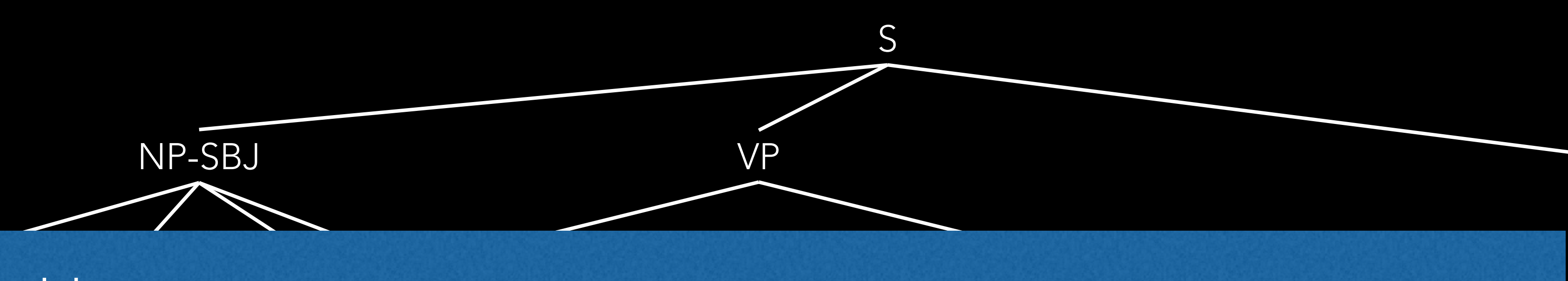
```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (CD 61) (NNS years) )
      (JJ old) )
    (, ,) )
  (VP (MD will)
    (VP (VB join)
      (NP (DT the) (NN board) )
      (PP-CLR (IN as)
        (NP (DT a) (JJ nonexecutive) (NN director) ))
      (NP-TMP (NNP Nov.)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



26 constituent tags:

S = declarative sentence   NP = noun phrase
VP = verb phrase   ADJP = adjective phrase
PP = prepositional phrase   SBAR = sentential clause
SQ = question   SINV = inverted sentence   FRAG = fragment
…

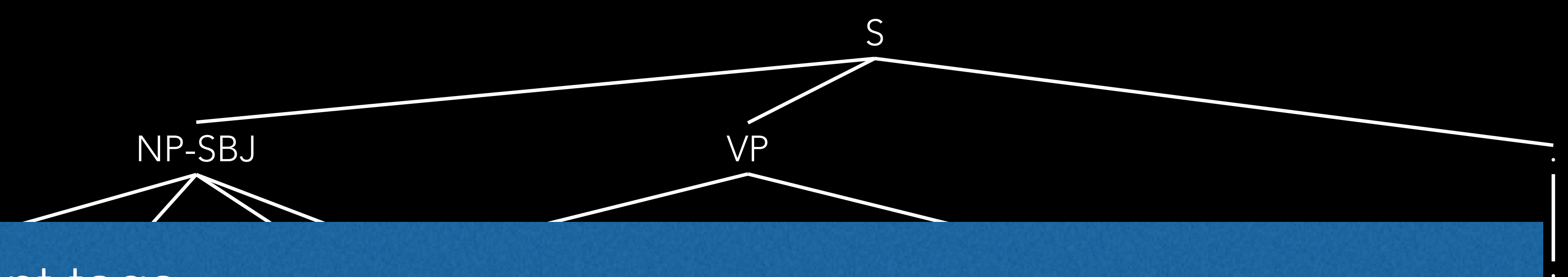
```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (CD 61) (NNS years) )
      (JJ old) )
    (, ,) )
  (VP (MD will)
    (VP (VB join)
      (NP (DT the) (NN board) )
      (PP-CLR (IN as)
        (NP (DT a) (JJ nonexecutive) (NN director) ))
      (NP-TMP (NNP Nov.)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



26 constituent tags:

S = declarative sentence   NP = noun phrase
VP = verb phrase   ADJP = adjective phrase
PP = prepositional phrase   SBAR = sentential clause
SQ = question   SINV = inverted sentence   FRAG = fragment
…

```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (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) )))
  (. .) )
```



26 constituent tags:

S = declarative sentence   NP = noun phrase
VP = verb phrase   ADJP = adjective phrase
PP = prepositional phrase   SBAR = sentential clause
SQ = question   SINV = inverted sentence   FRAG = fragment
…

```
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken) )
    (, ,)
    (ADJP
      (NP (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) )))
  (. .) )
```

20 function tags:

-SBJ = subject of sentence
-CLR = closely related (in this case, to the verb)
-TMP = temporal
…

These are usually not considered in parsing. I've never paid any attention to them.

Why make the Treebank?

Why make the Treebank?

- Before this it was common for linguists to declare what constituted a legitimate sentence structure in a language (this is a gross simplification).

Why make the Treebank?

- Before this it was common for linguists to declare what constituted a legitimate sentence structure in a language (this is a gross simplification).

- The Treebank provides a documentation of how (a very specific subset of) people actually structure English.

Why make the Treebank?

- Before this it was common for linguists to declare what constituted a legitimate sentence structure in a language (this is a gross simplification).
- The Treebank provides a documentation of how (a very specific subset of) people actually structure English.
- Constructed with this goal: given a new sentence, can we automatically put the tree on top? (i.e. syntactic parsing)

How to Use The Treebank

How to Use The Treebank

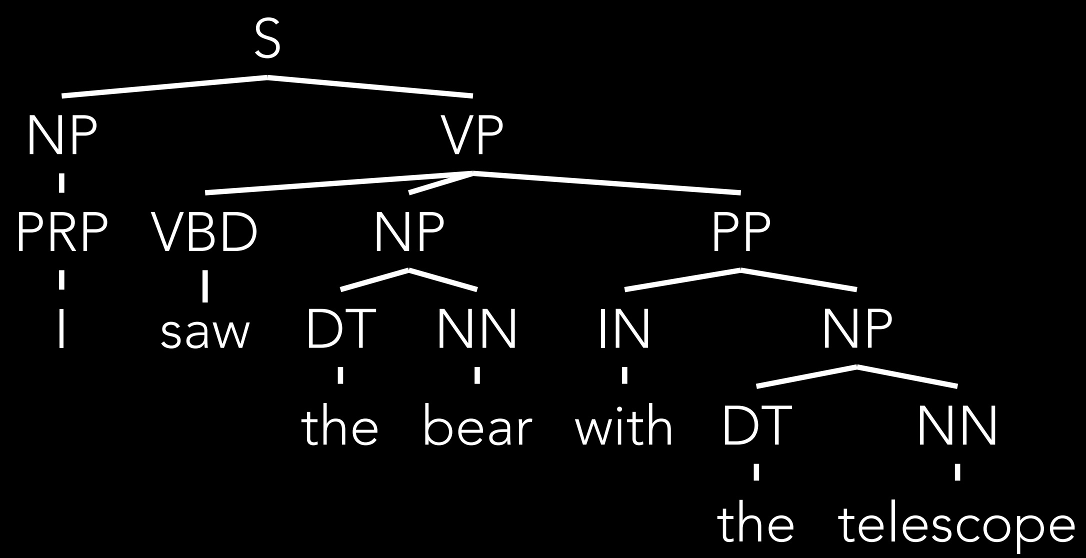
- Evaluation:
  - I’ll give you some of the sentences from the treebank (without trees).
  - You give me the trees your computer parser produced.
  - I’ll tell you how close you were to the human trees.

How to Use The Treebank

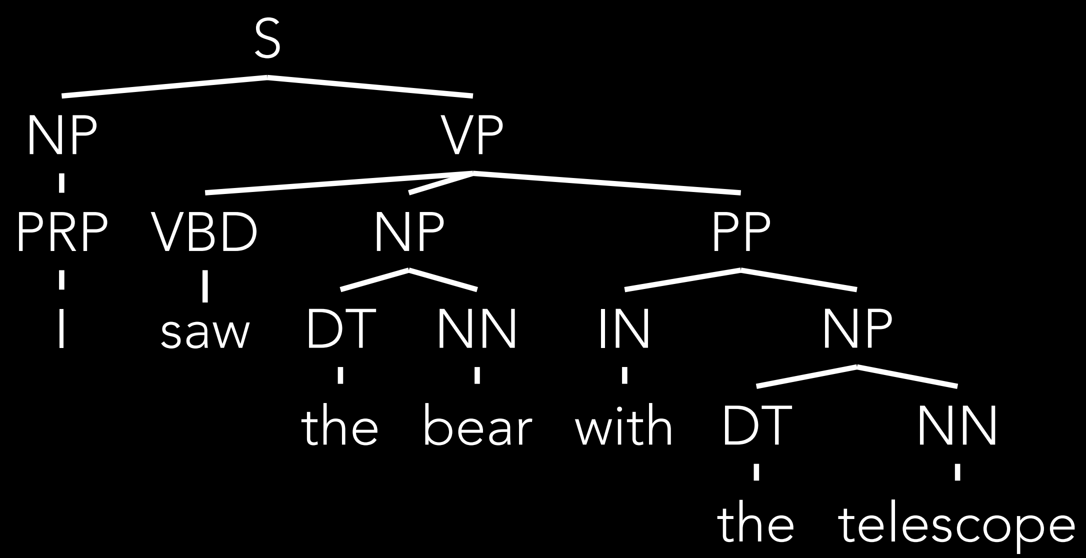
- Evaluation:
  - I’ll give you some of the sentences from the treebank (without trees).
  - You give me the trees your computer parser produced.
  - I’ll tell you how close you were to the human trees.
- Training:
  - Use the treebank trees as input to some data-driven parsing algorithm (just don’t use the same trees to evaluate)
  - Standard split: sections 2-21 for training, section 23 for test

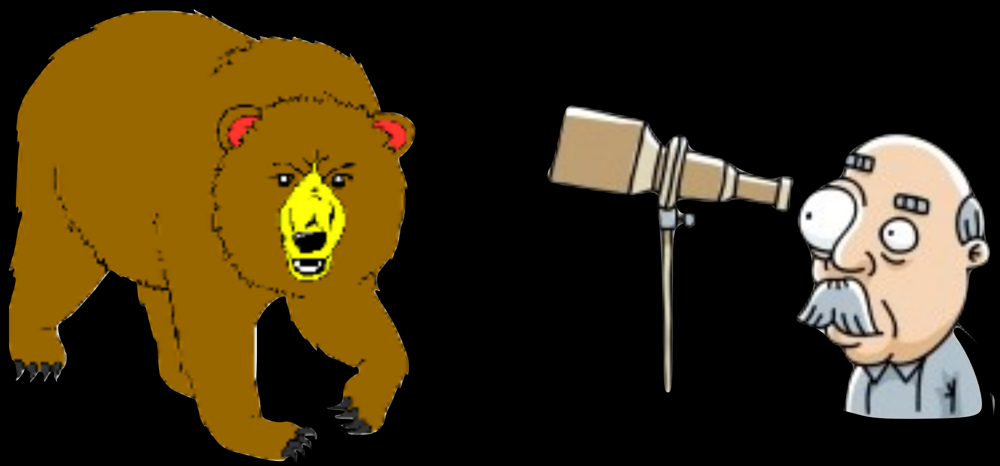
Parser Evaluation

Parser Evaluation

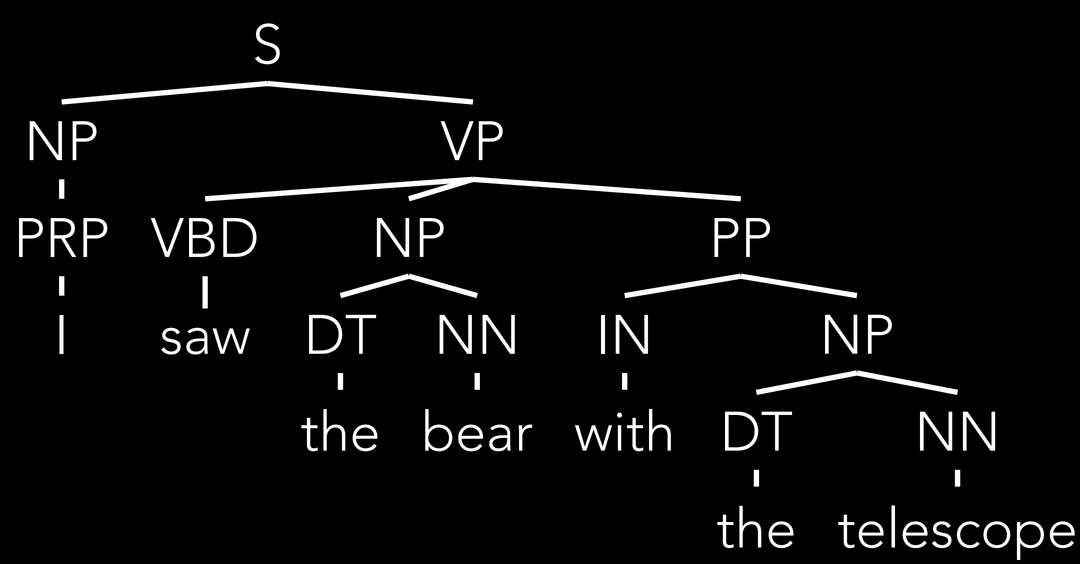


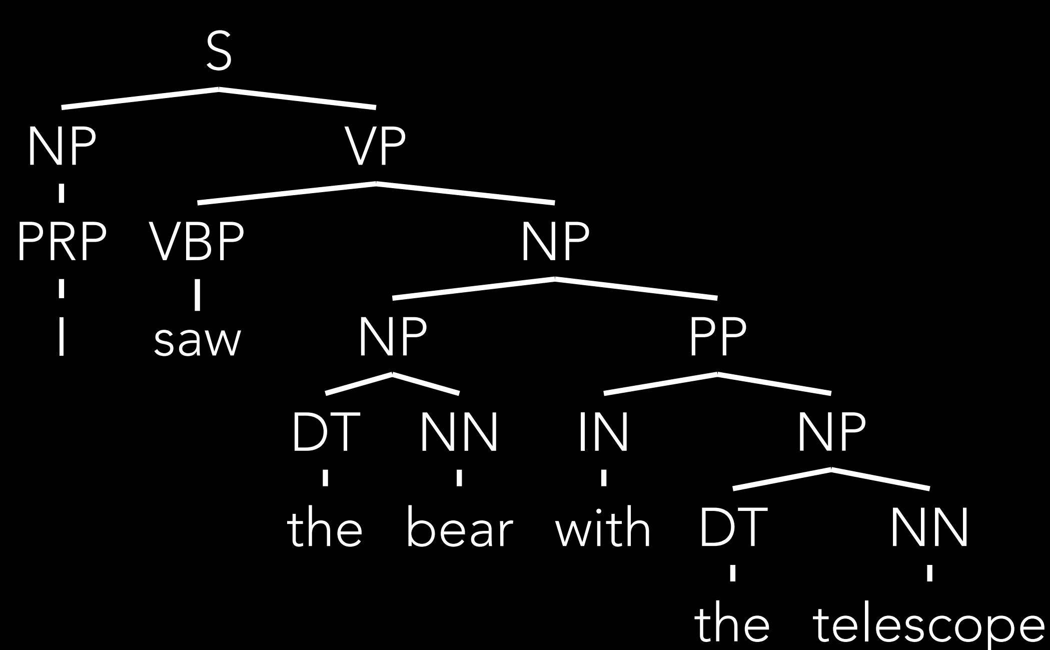
Parser Evaluation

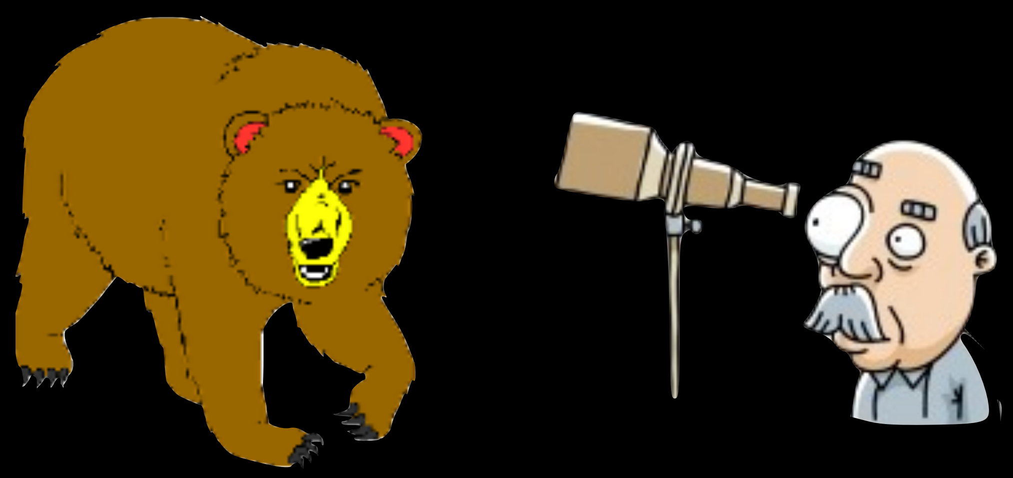




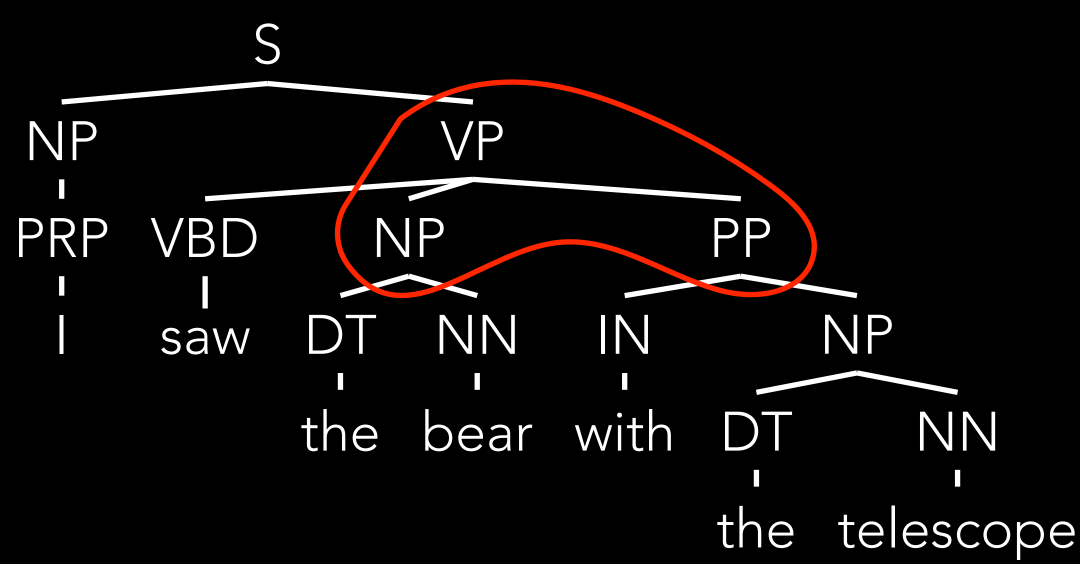
Parser Evaluation

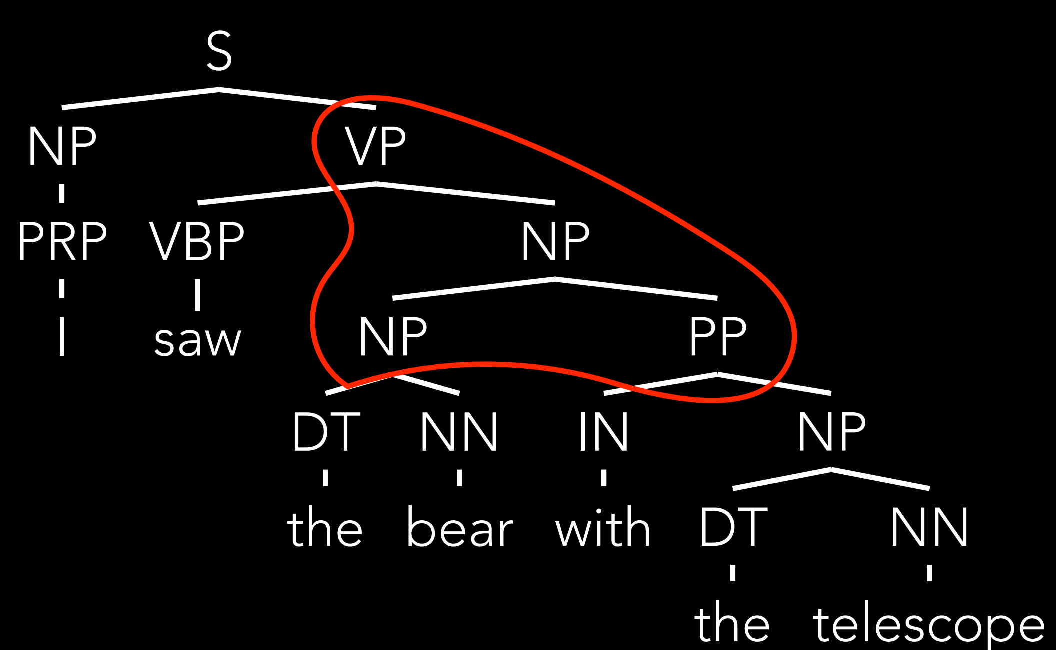




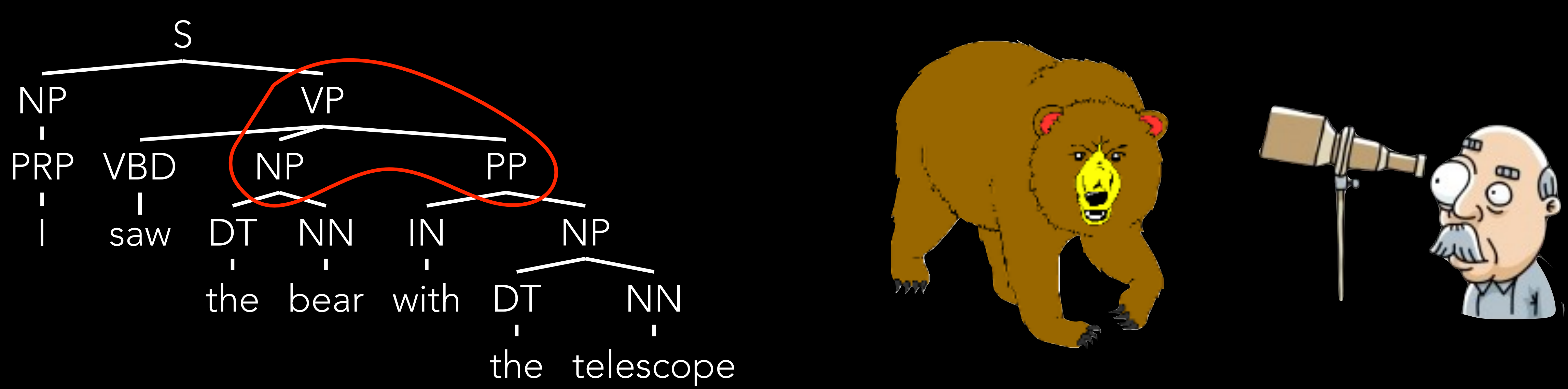


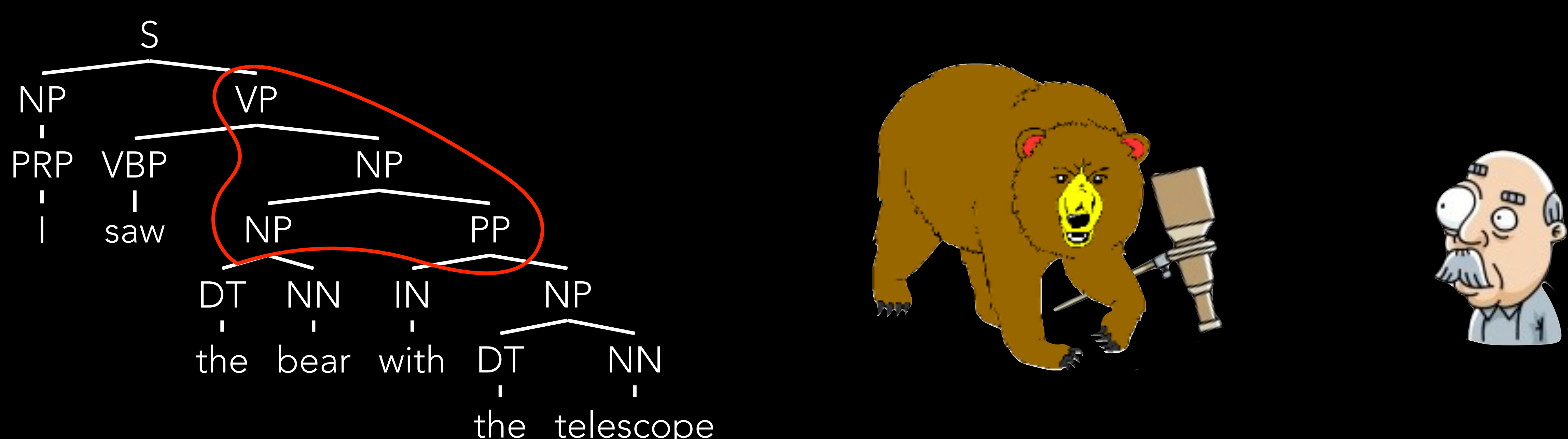
Parser Evaluation



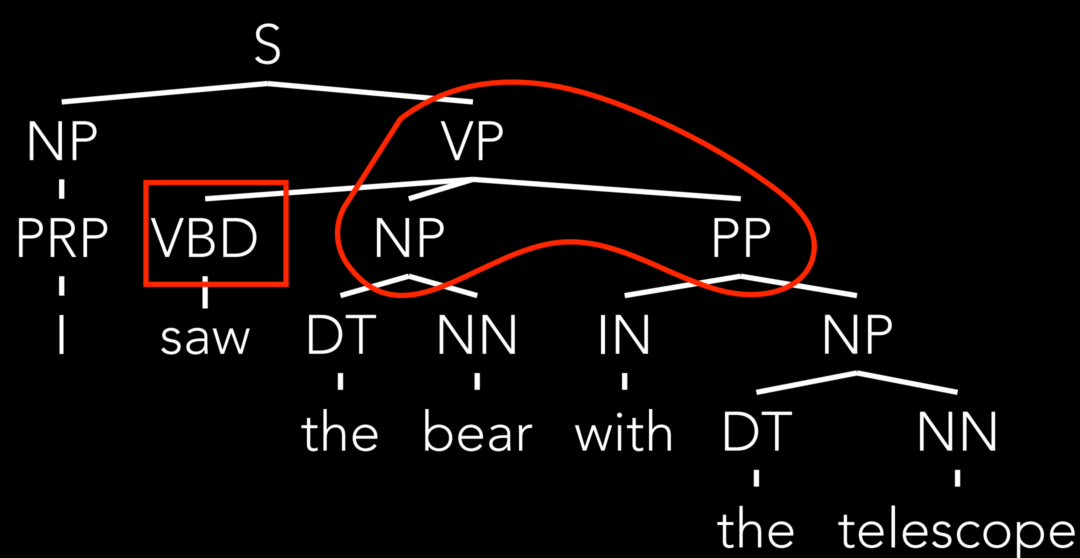


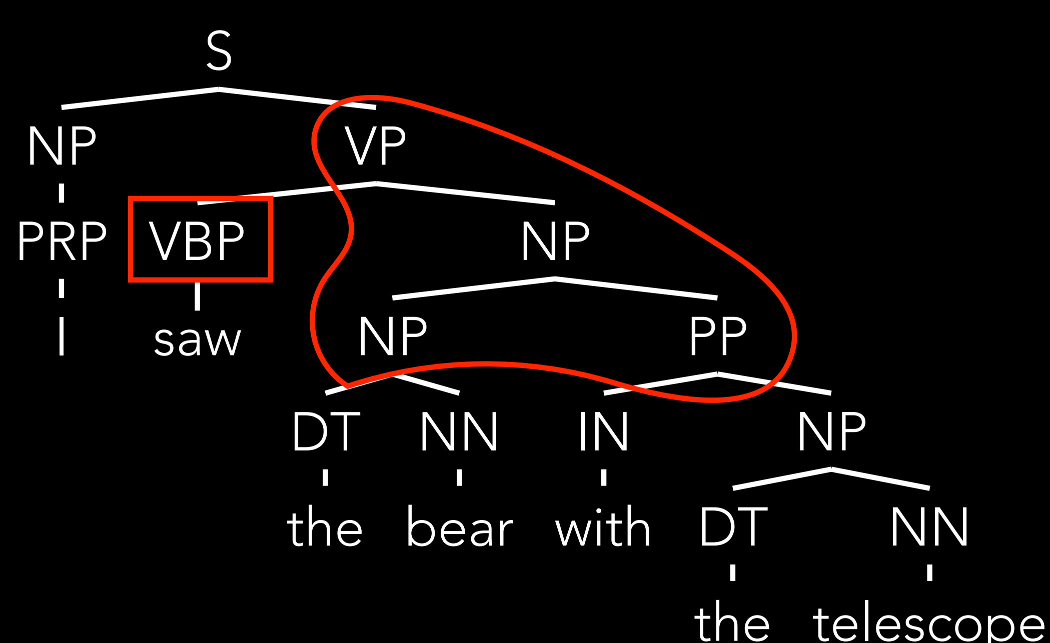
Parser Evaluation

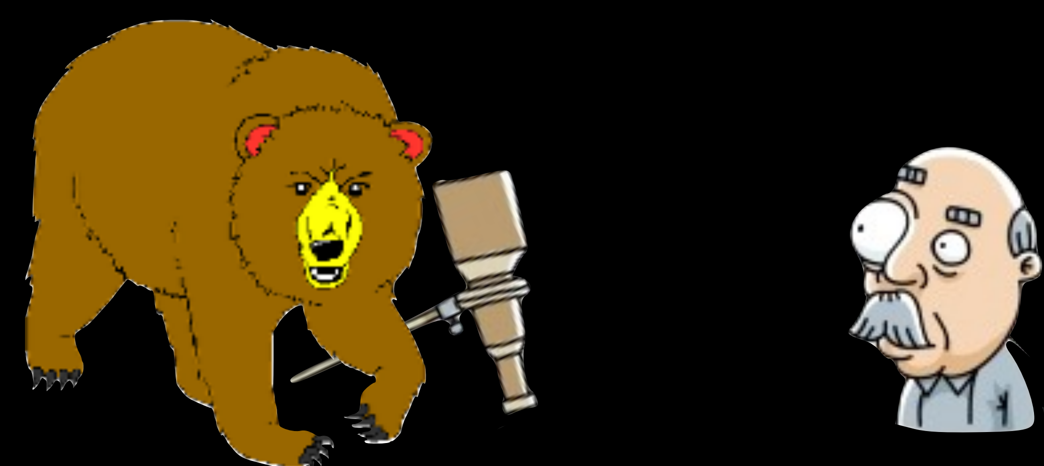




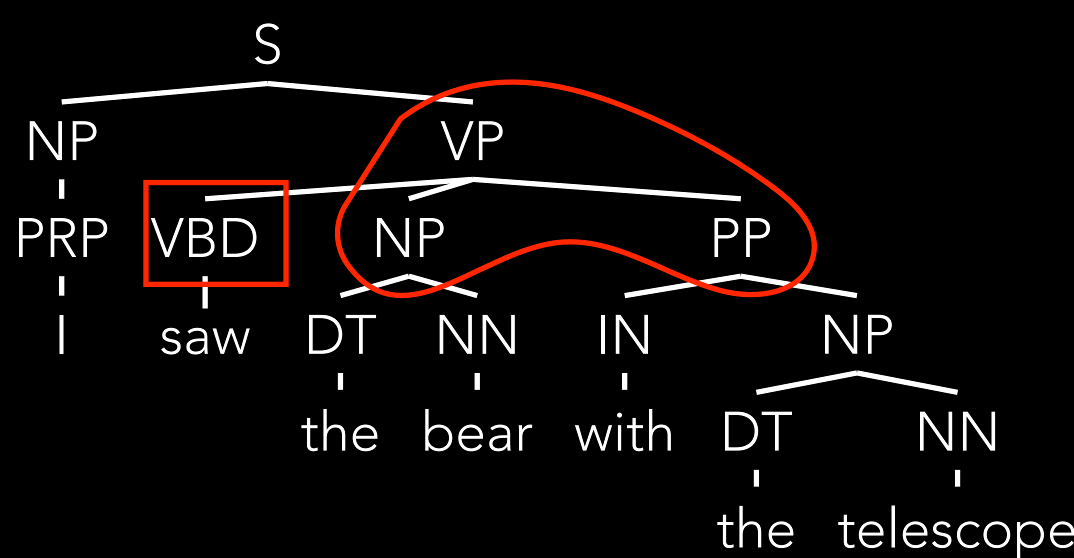
Parser Evaluation

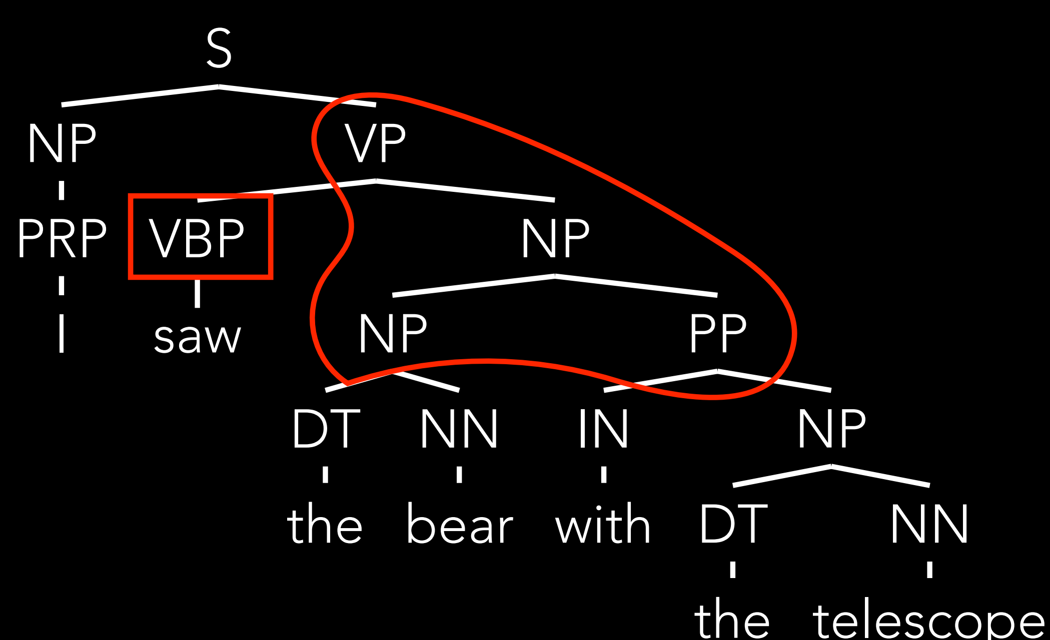


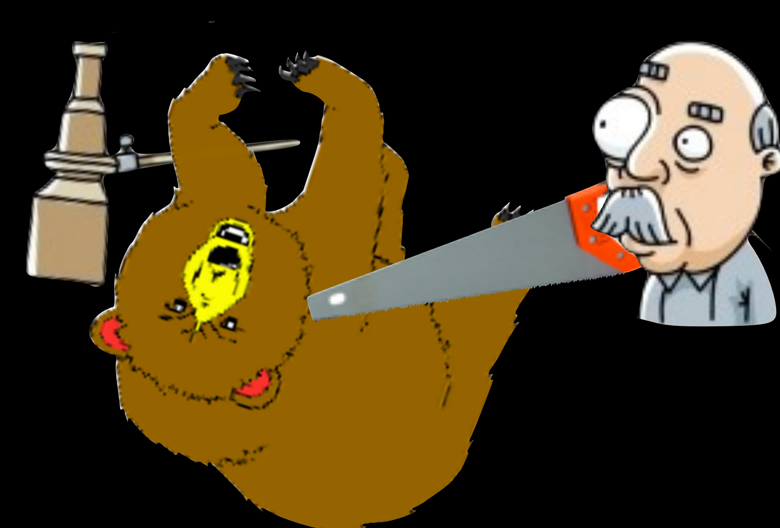




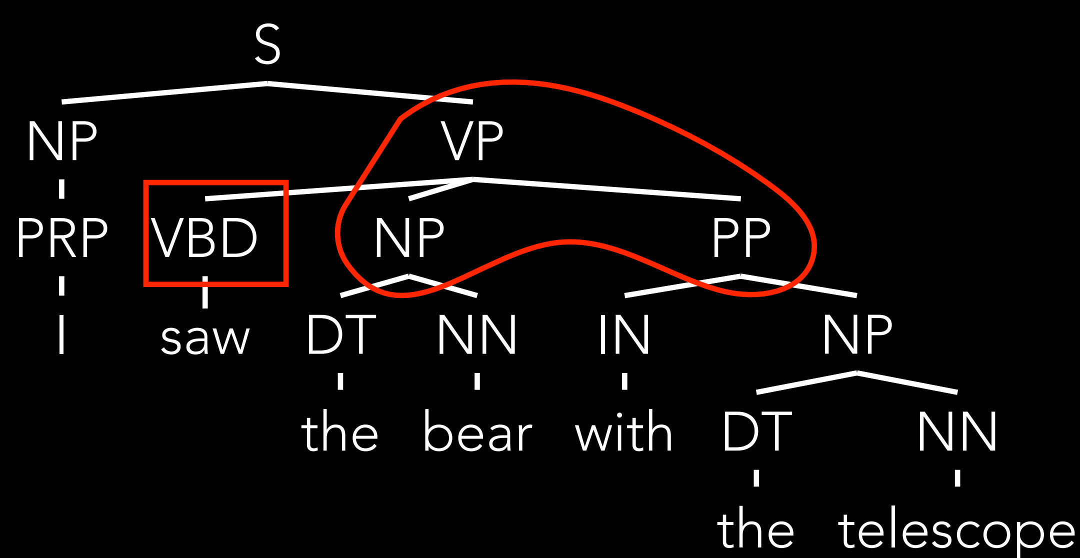
Parser Evaluation

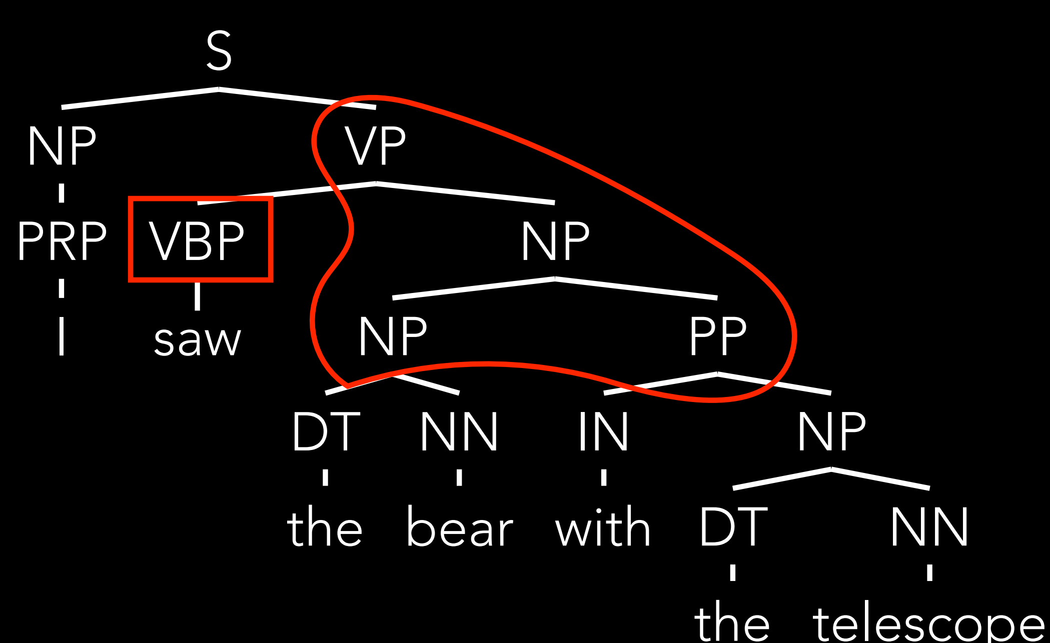


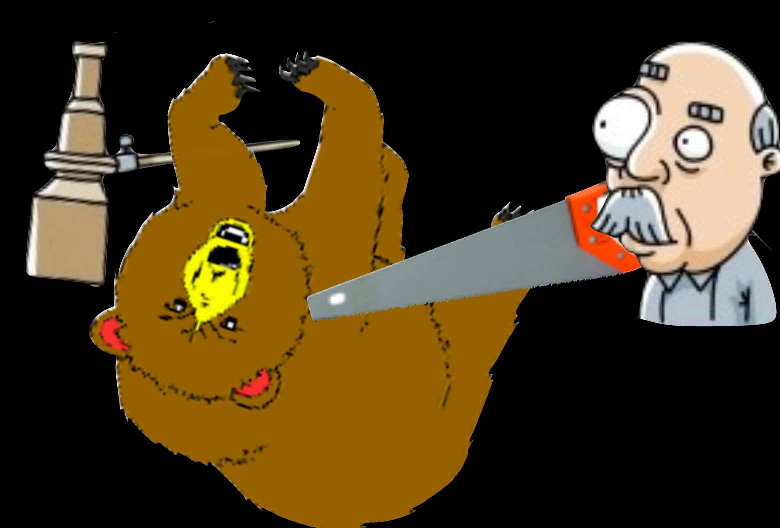




Parser Evaluation







How do we quantify parse errors?

Parser Evaluation

Parseval (Black et al. ’91)

Parser Evaluation

Parseval (Black et al. ’91)

1. Decompose tree into labeled *constituent* spans (note: POS tags not considered!)

Parser Evaluation

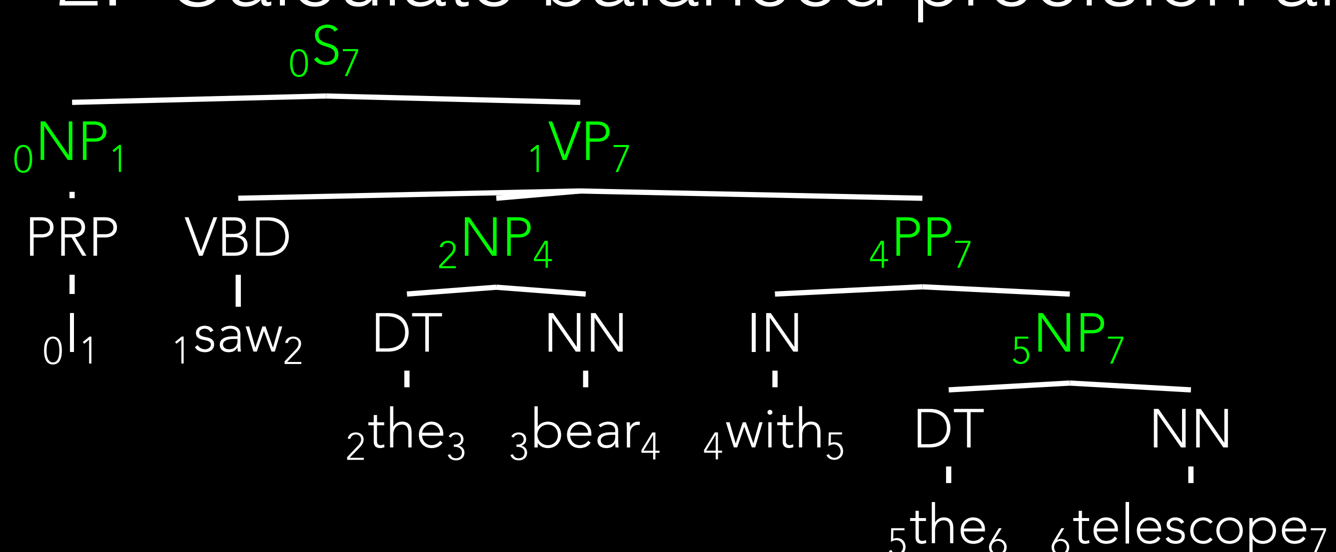
Parseval (Black et al. ’91)

1. Decompose tree into labeled *constituent* spans
   (note: POS tags not considered!)
2. Calculate balanced precision and recall (f-measure)

Parser Evaluation

Parseval (Black et al. ’91)

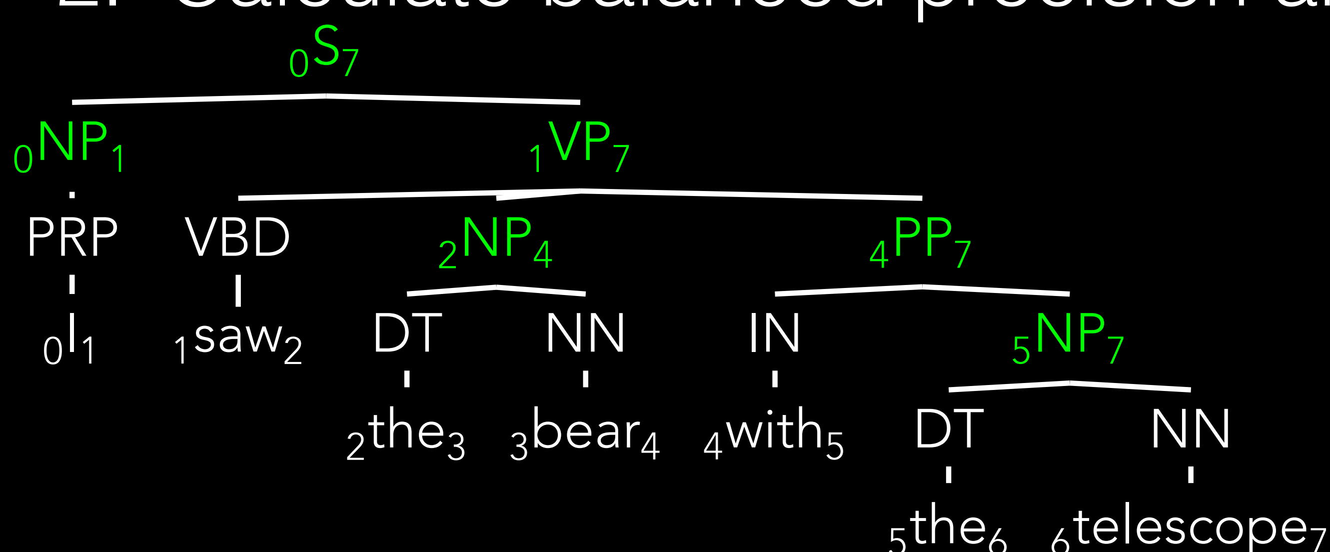
1. Decompose tree into labeled *constituent* spans
   (note: POS tags not considered!)
2. Calculate balanced precision and recall (f-measure)



Parser Evaluation

Parseval (Black et al. '91)

1. Decompose tree into labeled *constituent* spans
   (note: POS tags not considered!)
2. Calculate balanced precision and recall (f-measure)

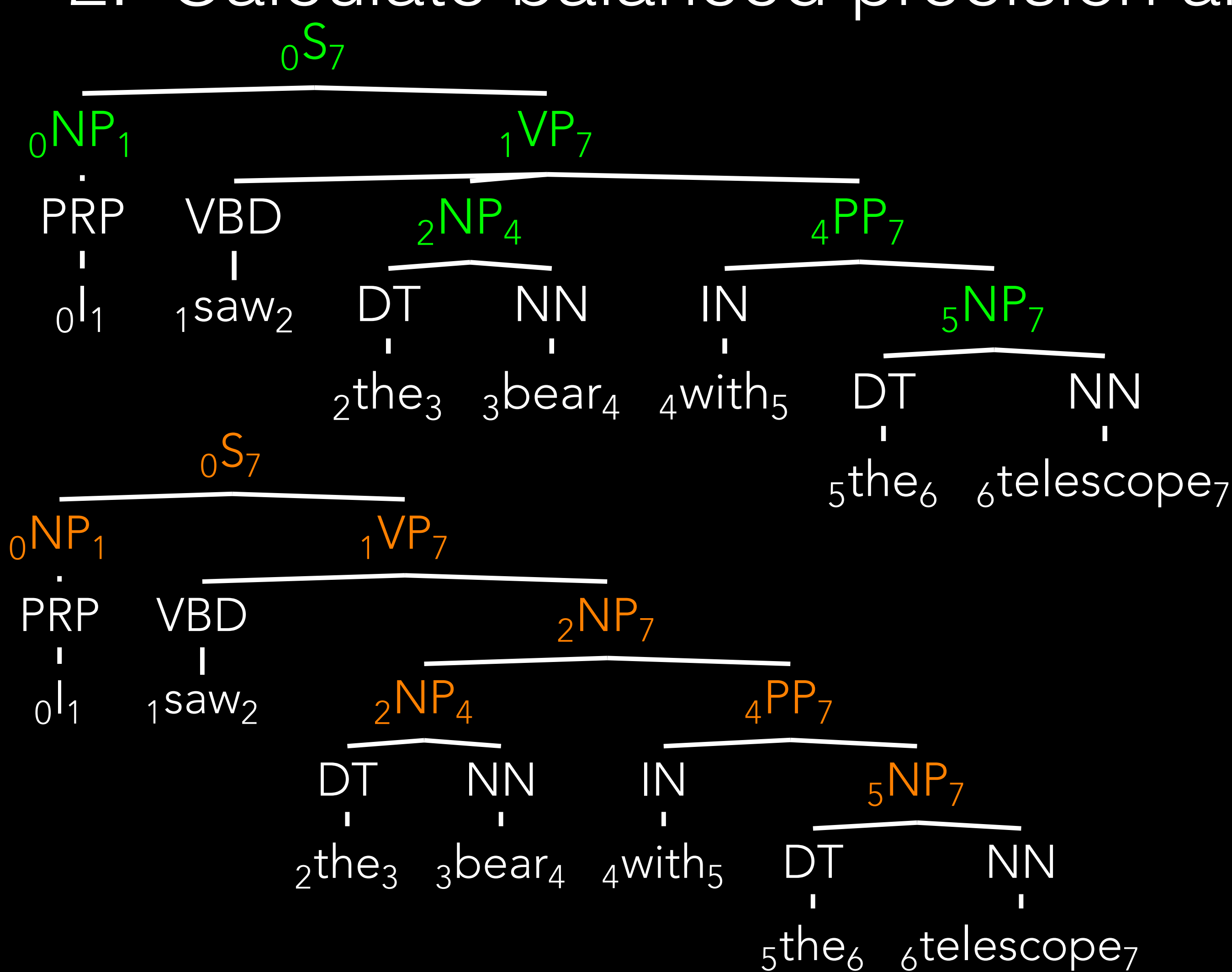


(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

Parser Evaluation

Parseval (Black et al. ’91)

1. Decompose tree into labeled *constituent* spans (note: POS tags not considered!)
2. Calculate balanced precision and recall (f-measure)

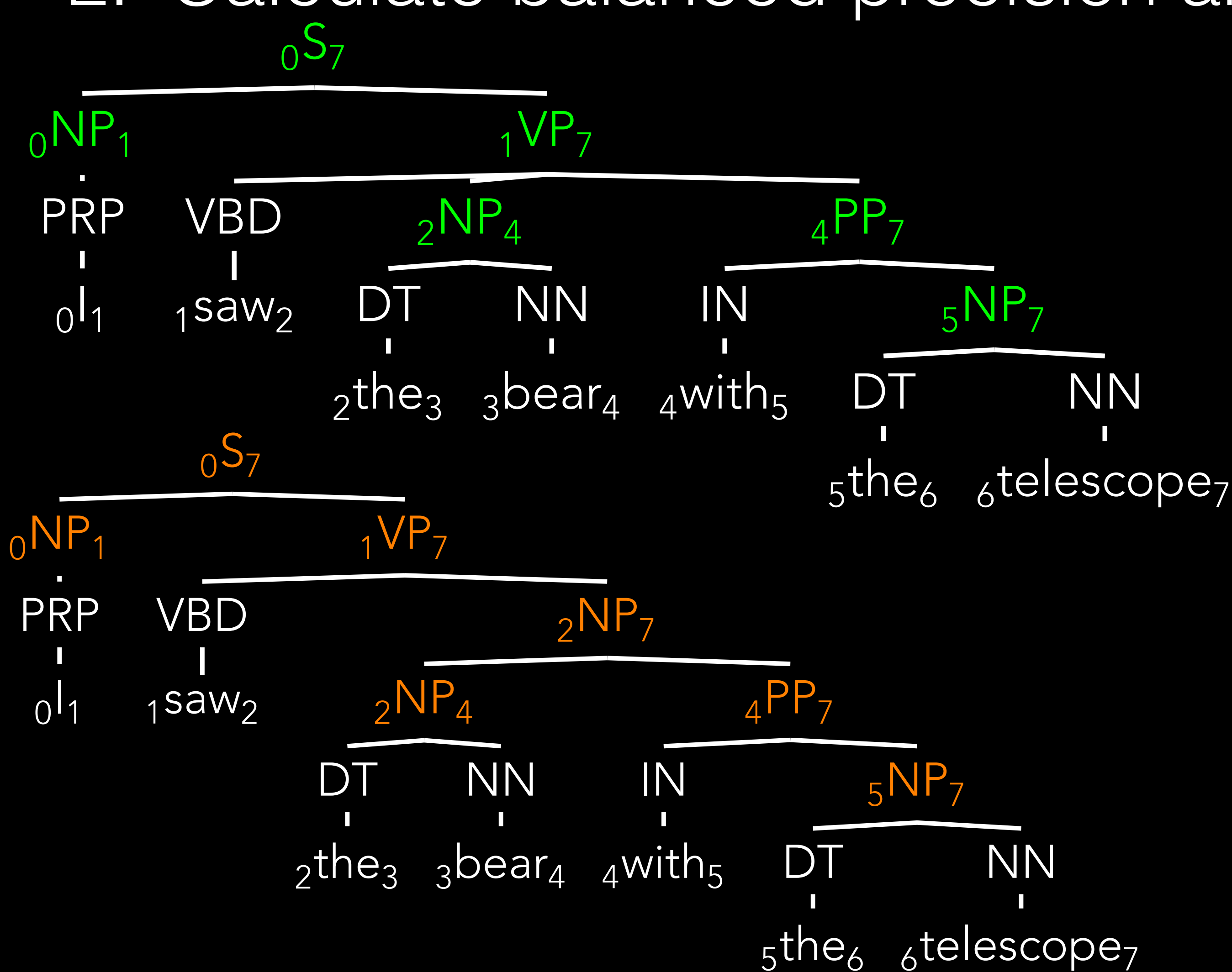


(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

Parser Evaluation

Parseval (Black et al. '91)

1. Decompose tree into labeled *constituent* spans
   (note: POS tags not considered!)
2. Calculate balanced precision and recall (f-measure)



(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

Parser Evaluation

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

A = answers

C = correct

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

Parser Evaluation

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

A = answers   C = correct

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

Precision: Of the answers I gave, what percent were correct?

Parser Evaluation

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

A = answers

C = correct

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

Precision: Of the answers I gave, what percent were correct?

$$\frac{|A \cap C|}{|A|}$$

Parser Evaluation

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

A = answers

C = correct

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

Precision: Of the answers I gave, what percent were correct?

$$\frac{|A \cap C|}{|A|}$$

Recall: What percent of the correct answers did I give?

Parser Evaluation

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

A = answers   C = correct

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

Precision: Of the answers I gave, what percent were correct?

$$\frac{|A \cap C|}{|A|}$$

Recall: What percent of the correct answers did I give?

$$\frac{|A \cap C|}{|C|}$$

Parser Evaluation

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

A = answers   C = correct

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

Precision: Of the answers I gave, what percent were correct?

$$\frac{|A \cap C|}{|A|}$$

Recall: What percent of the correct answers did I give?

$$\frac{|A \cap C|}{|C|}$$

F1: Harmonic mean of Precision and Recall (reciprocal of mean of reciprocals)

Parser Evaluation

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

A = answers   C = correct

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

Precision: Of the answers I gave, what percent were correct?

$$\frac{|A \cap C|}{|A|}$$

Recall: What percent of the correct answers did I give?

$$\frac{|A \cap C|}{|C|}$$

F1: Harmonic mean of Precision and Recall (reciprocal of mean of reciprocals)

$$\frac{2|A \cap C|}{|C| + |A|}$$

Parser Evaluation

A = answers

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

$|A| = 7$

C = correct

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

$|C| = 6$

Parser Evaluation

A = answers

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

$|A| = 7$

$|A \cap C| = 6$

C = correct

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

$|C| = 6$

Parser Evaluation

A = answers

(S 0 7)
(NP 0 1)
(VP 1 7)
extra! $\boxed{\text{(NP 2 7)}}$
(NP 2 4)
(PP 4 7)
(NP 5 7)

$|A| = 7$

$|A \cap C| = 6$

C = correct

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

$|C| = 6$

Parser Evaluation

A = answers

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

C = correct

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

$|A \cap C| = 6$

$|A| = 7$

$|C| = 6$

$$R = \frac{|A \cap C|}{|C|} = 1.0$$

Parser Evaluation

A = answers

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

C = correct

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

$|A \cap C| = 6$

$|A| = 7$

$|C| = 6$

$$R = \frac{|A \cap C|}{|C|} = 1.0$$

$$P = \frac{|A \cap C|}{|A|} = .857$$

Parser Evaluation

A = answers

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

C = correct

(S 0 7)
(NP 0 1)
(VP 1 7)
(NP 2 4)
(PP 4 7)
(NP 5 7)

$|A \cap C| = 6$

$|A| = 7$

$|C| = 6$

$$R = \frac{|A \cap C|}{|C|} = 1.0 \qquad P = \frac{|A \cap C|}{|A|} = .857$$

$$F1 = \frac{2|A \cap C|}{|C| + |A|} = .923$$

How to parse a sentence?

How to parse a sentence?

- Input: word-separated sentence

How to parse a sentence?

- Input: word-separated sentence
- Output: syntactic tree with POS tag/word labels

How to parse a sentence?

- Input: word-separated sentence
- Output: syntactic tree with POS tag/word labels
- How you get there is up to you!

How to parse a sentence?

- Input: word-separated sentence
- Output: syntactic tree with POS tag/word labels
- How you get there is up to you!
- So far in this class we've used wFSAs and wFSTs. Can we keep using them?

How to parse a sentence?

- Input: word-separated sentence
- Output: syntactic tree with POS tag/word labels
- How you get there is up to you!
- So far in this class we've used wFSAs and wFSTs. Can we keep using them?
- NO! (Why?)

Context-Free Grammars

Context-Free Grammars

S
NP
VP
NNS
PP
VB
...

Nonterminals
(start)

Context-Free Grammars

S NP
VP NNS
PP
VB
…

Nonterminals
(start)

cats I
mice the
telescope
… eat

Terminals

Context-Free Grammars

Nonterminals (start)	Terminals	Rules
S, NP, VP, NNS, PP, VB, …	cats, I, mice, the, telescope, eat, …	S → NP VP VP → VB NP NP → NNS NNS → cats NNS → mice VB → eat

Context-Free Grammars

S NP VP NNS PP VB …

Nonterminals (start)

cats I mice the telescope eat …

Terminals

S → NP VP
VP → VB NP
NP → NNS
NNS → cats
NNS → mice
VB → eat

Rules

S

Extend leftmost nonterminal by RHS of matching rule until done

Context-Free Grammars

S NP VP NNS PP VB …

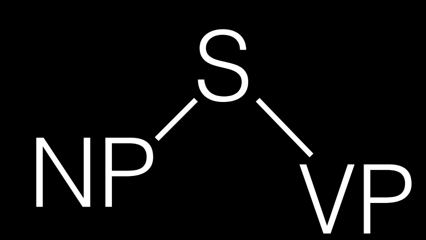
Nonterminals (start)

cats I mice the telescope eat …

Terminals

S → NP VP
VP → VB NP
NP → NNS
NNS → cats
NNS → mice
VB → eat

Rules



Extend leftmost nonterminal by RHS of matching rule until done

Context-Free Grammars

S NP
VP NNS
PP
VB
…

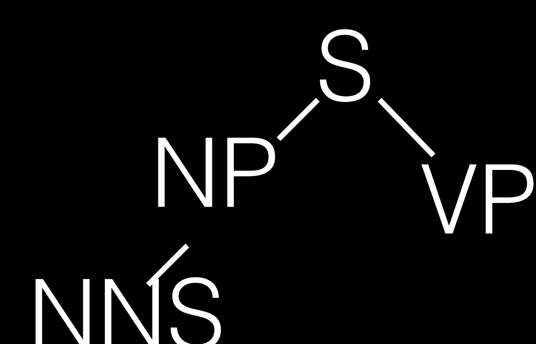
Nonterminals
(start)

cats I
mice the
telescope
… eat

Terminals

S → NP VP
VP → VB NP
NP → NNS
NNS → cats
NNS → mice
VB → eat

Rules



Extend leftmost nonterminal by RHS of matching rule until done

Context-Free Grammars

S NP
VP NNS
PP
VB
…

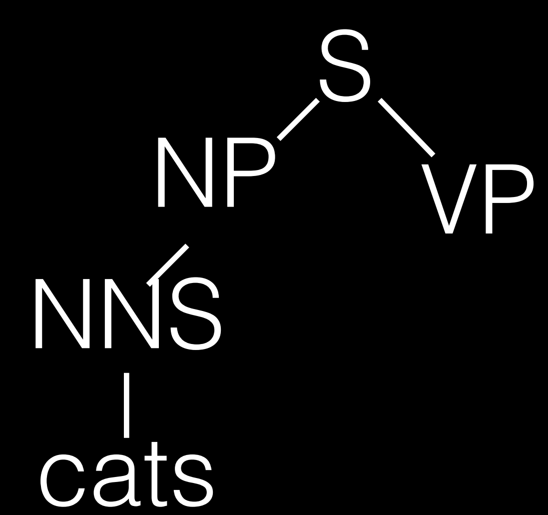
Nonterminals
(start)

cats I
mice the
telescope
eat
…

Terminals

S → NP VP
VP → VB NP
NP → NNS
NNS → cats
NNS → mice
VB → eat

Rules



Extend leftmost nonterminal by RHS of matching rule until done

Context-Free Grammars

Nonterminals (start)

S

NP

VP

NNS

PP

VB

…

Terminals

cats

I

mice

the

telescope

eat

…

Rules

S → NP VP

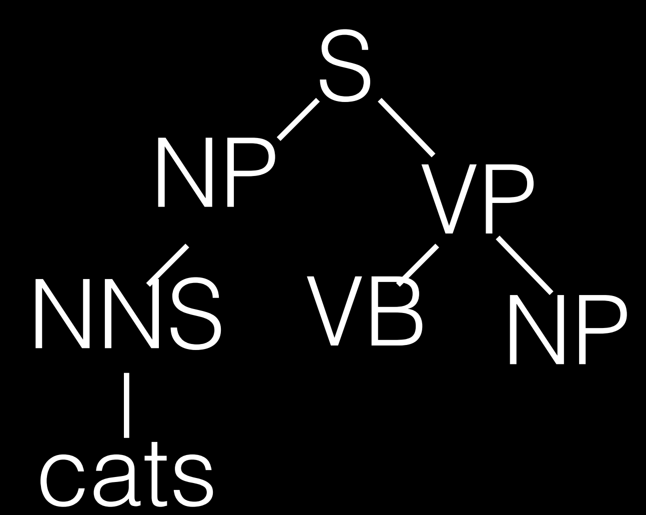
VP → VB NP

NP → NNS

NNS → cats

NNS → mice

VB → eat



Extend leftmost nonterminal by RHS of matching rule until done

Context-Free Grammars

S NP
VP NNS
PP
VB
…

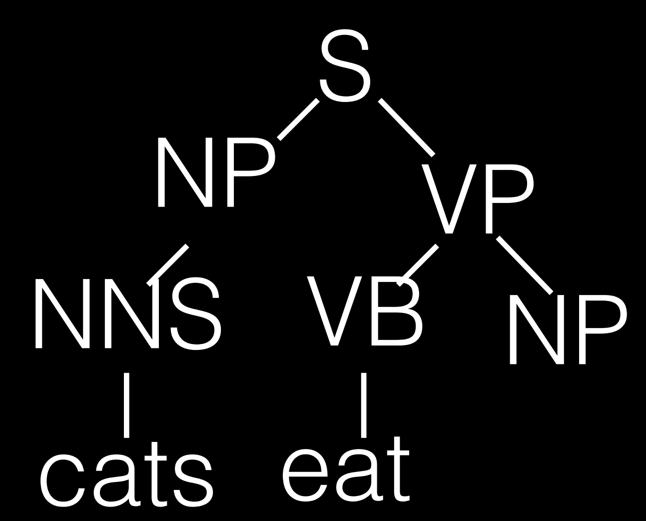
Nonterminals
(start)

cats I
mice the
telescope
… eat

Terminals

S → NP VP
VP → VB NP
NP → NNS
NNS → cats
NNS → mice
VB → eat

Rules



Extend leftmost
nonterminal
by RHS of matching
rule until done

Context-Free Grammars

S
NP
VP
NNS
PP
VB
…

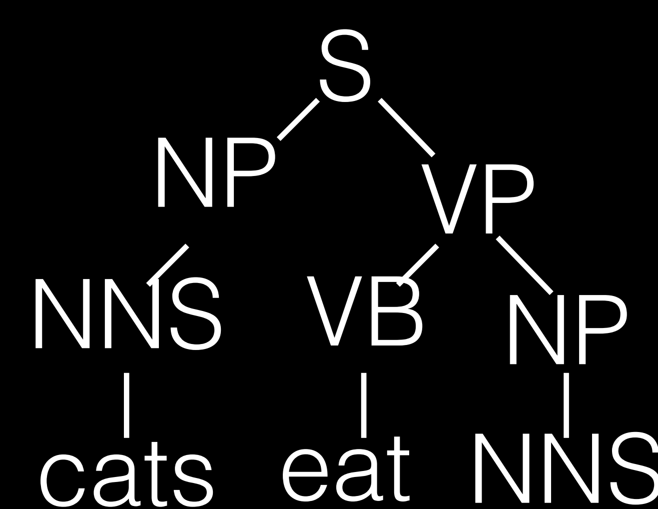
Nonterminals
(start)

cats
I
mice
the
telescope
eat
…

Terminals

S → NP VP
VP → VB NP
NP → NNS
NNS → cats
NNS → mice
VB → eat

Rules



Extend leftmost nonterminal by RHS of matching rule until done

Context-Free Grammars

S NP
VP NNS
PP
VB
…

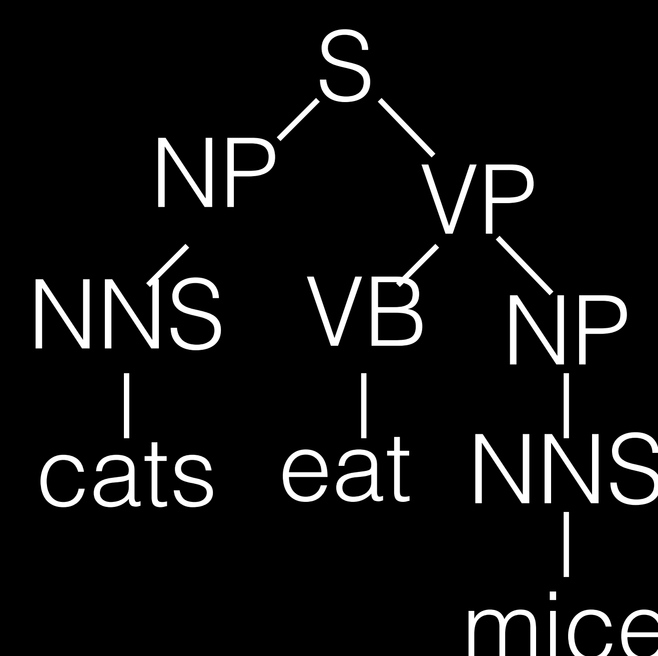
Nonterminals
(start)

cats I
mice the
telescope
… eat

Terminals

S → NP VP
VP → VB NP
NP → NNS
NNS → cats
NNS → mice
VB → eat

Rules



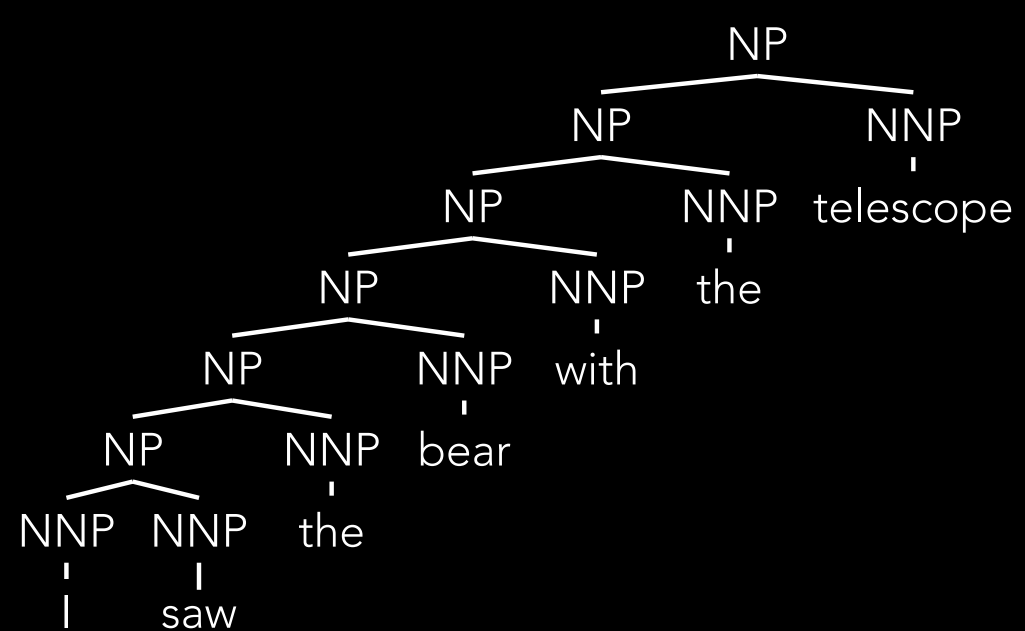
Extend leftmost nonterminal by RHS of matching rule until done

Baseline Parser

Left-branching chain of NPs

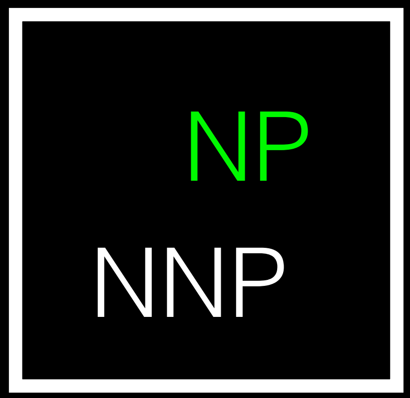
Baseline Parser

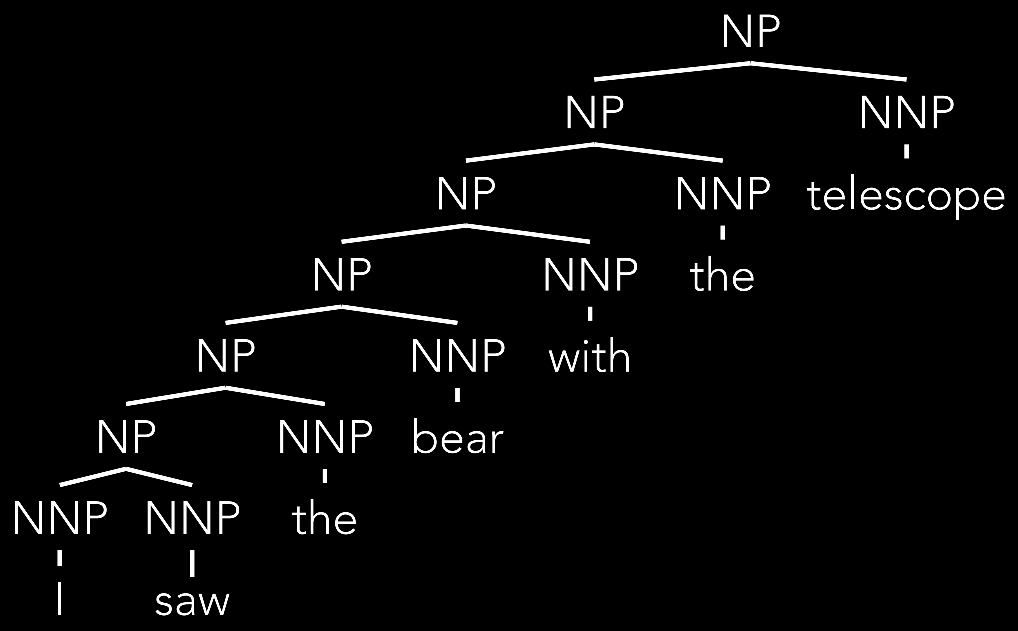
Left-branching chain of NPs



Baseline Parser

Left-branching chain of NPs



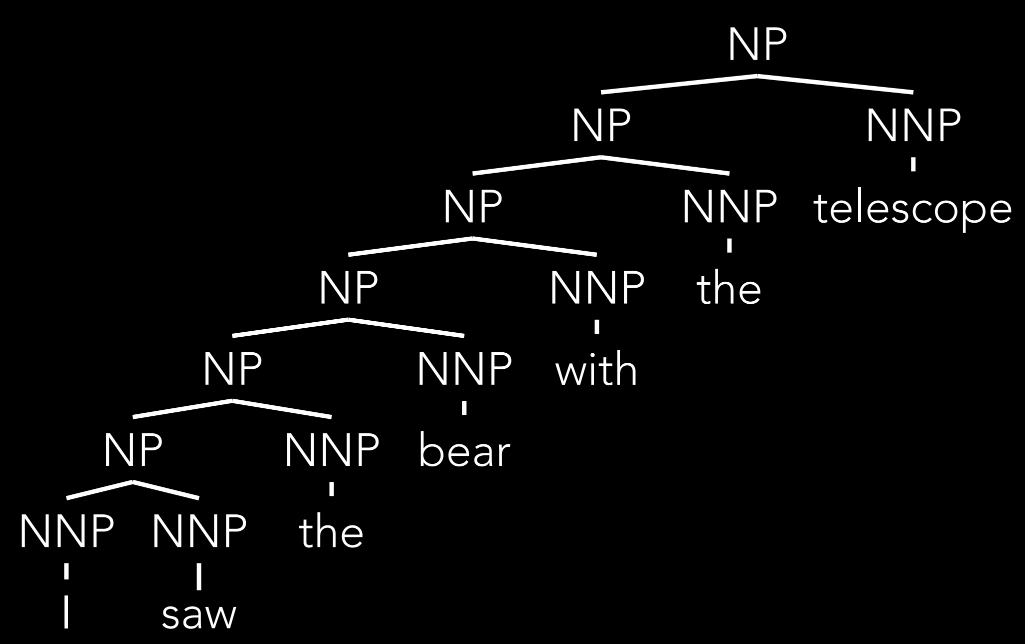


Baseline Parser

Left-branching chain of NPs

NP
NNP

$$\begin{aligned} NP &\rightarrow NP\ NNP \\ NP &\rightarrow NNP\ NNP \\ NNP &\rightarrow \text{I/saw/the/…} \end{aligned}$$



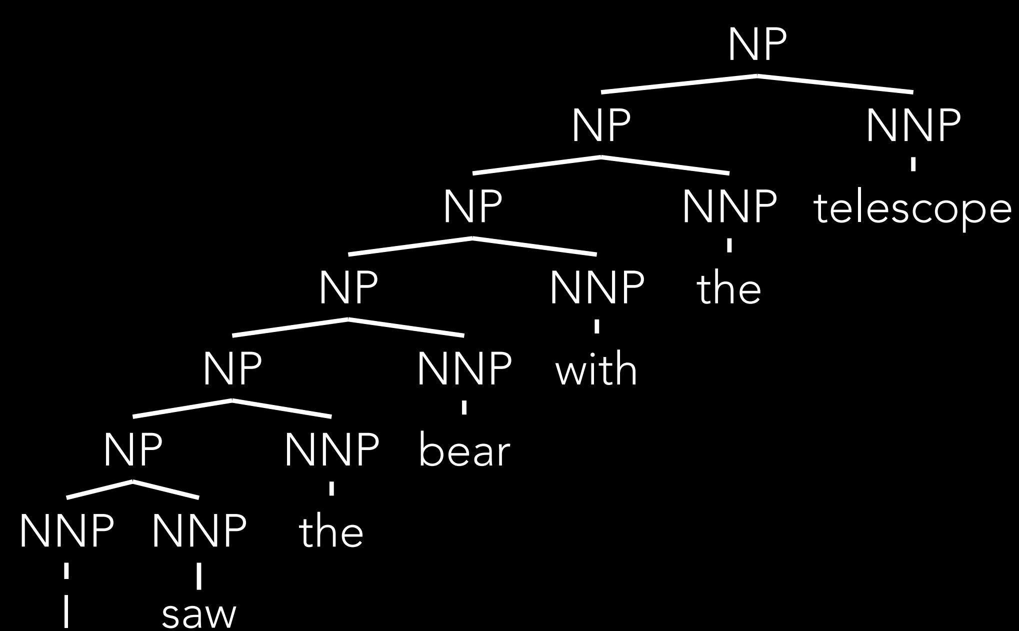
Baseline Parser

Left-branching chain of NPs

NP

NNP

NP → NP NNP
NP → NNP NNP
NNP → I/saw/the/…



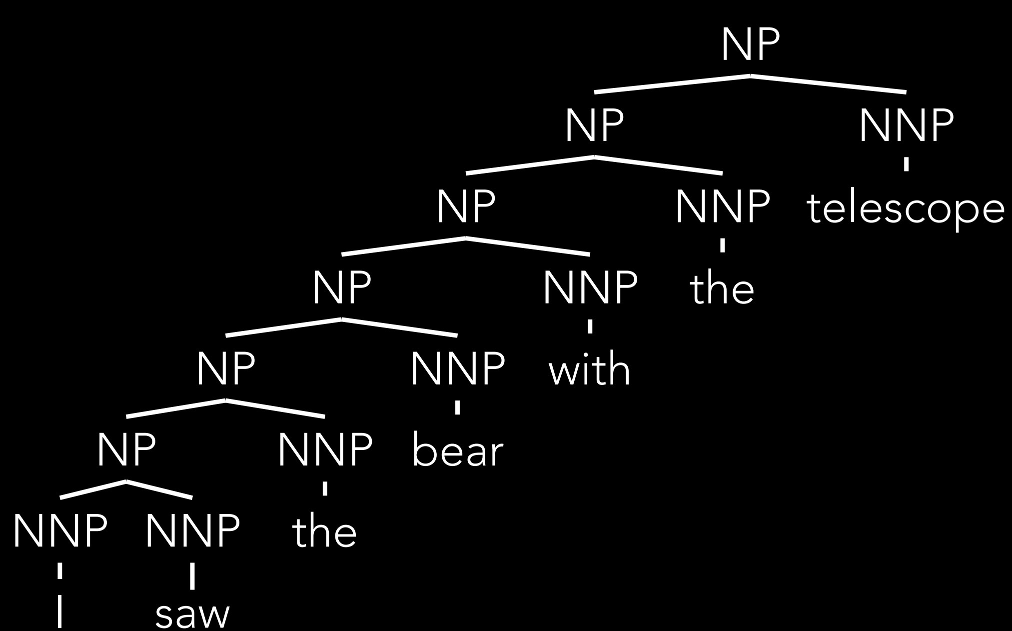
- Data-driven! (Most popular tag/constituent in PTB)

Baseline Parser

Left-branching chain of NPs

NP
NNP

NP → NP NNP
NP → NNP NNP
NNP → I/saw/the/…



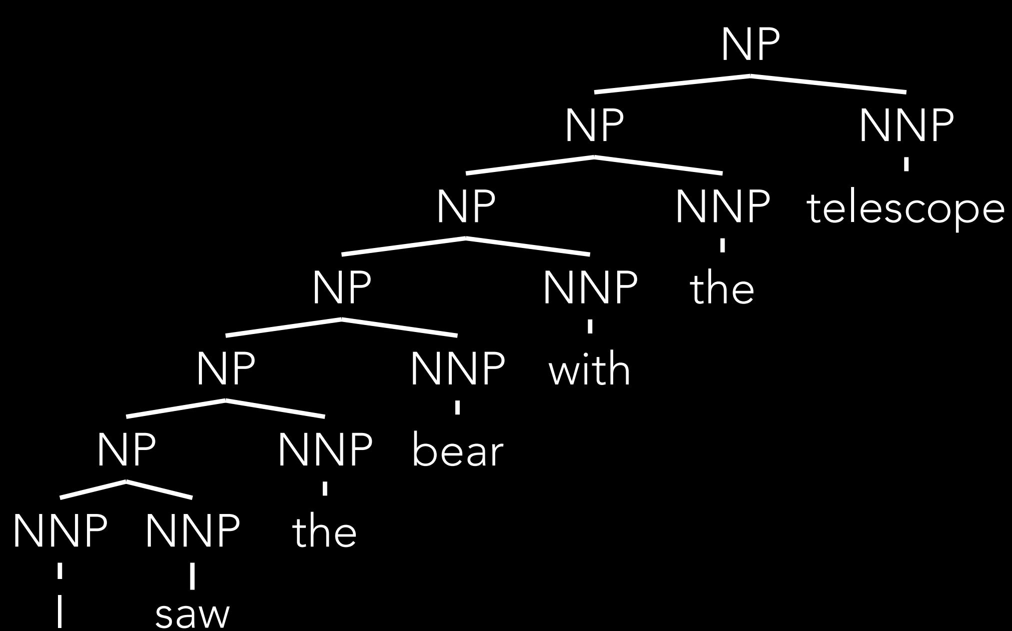
- Data-driven! (Most popular tag/constituent in PTB)
- Easy!

Baseline Parser

Left-branching chain of NPs

NP
NNP

NP → NP NNP
NP → NNP NNP
NNP → I/saw/the/…



- Data-driven! (Most popular tag/constituent in PTB)
- Easy!
- F1 of around 0.03

But how do you actually *parse*?

But how do you actually *parse*?

- CFG operations showed how to *generate* a tree, not put a tree on a given string.

But how do you actually *parse*?

- CFG operations showed how to *generate* a tree, not put a tree on a given string.

- We’re intersecting two languages, so we can use the *Bar-Hillel* construction: annotate the CFG rules with indices corresponding to the words in the string.

But how do you actually *parse*?

- CFG operations showed how to *generate* a tree, not put a tree on a given string.

- We're intersecting two languages, so we can use the *Bar-Hillel* construction: annotate the CFG rules with indices corresponding to the words in the string.

$$\begin{aligned} NP &\rightarrow NP\ NNP \\ NP &\rightarrow NNP\ NNP \\ NNP &\rightarrow \text{I} \mid \text{saw} \mid \text{the} \mid \text{bear} \end{aligned}$$

But how do you actually *parse*?

- CFG operations showed how to *generate* a tree, not put a tree on a given string.
- We're intersecting two languages, so we can use the *Bar-Hillel* construction: annotate the CFG rules with indices corresponding to the words in the string.

$$\boxed{\begin{array}{l} \mathrm{NP} \rightarrow \mathrm{NP}\ \mathrm{NNP} \\ \mathrm{NP} \rightarrow \mathrm{NNP}\ \mathrm{NNP} \\ \mathrm{NNP} \rightarrow \mathrm{I} \mid \mathrm{saw} \mid \mathrm{the} \mid \mathrm{bear} \end{array}}$$

I saw the bear

But how do you actually *parse*?

- CFG operations showed how to *generate* a tree, not put a tree on a given string.

- We're intersecting two languages, so we can use the *Bar-Hillel* construction: annotate the CFG rules with indices corresponding to the words in the string.

$$\mathrm{NP} \rightarrow \mathrm{NP\ NNP}$$
$$\mathrm{NP} \rightarrow \mathrm{NNP\ NNP}$$
$$\mathrm{NNP} \rightarrow \mathrm{I} \mid \mathrm{saw} \mid \mathrm{the} \mid \mathrm{bear}$$

I saw the bear
0 1 2 3

But how do you actually *parse*?

- CFG operations showed how to *generate* a tree, not put a tree on a given string.

- We're intersecting two languages, so we can use the *Bar-Hillel* construction: annotate the CFG rules with indices corresponding to the words in the string.

$$\boxed{\begin{array}{l} NP \rightarrow NP\ NNP \\ NP \rightarrow NNP\ NNP \\ NNP \rightarrow I \mid saw \mid the \mid bear \end{array}}$$

I saw the bear
0 1 2 3

$NNP_0 \rightarrow I$

But how do you actually *parse*?

- CFG operations showed how to *generate* a tree, not put a tree on a given string.

- We're intersecting two languages, so we can use the *Bar-Hillel* construction: annotate the CFG rules with indices corresponding to the words in the string.

$$\boxed{\begin{array}{l} NP \rightarrow NP\ NNP \\ NP \rightarrow NNP\ NNP \\ NNP \rightarrow \text{I} \mid \text{saw} \mid \text{the} \mid \text{bear} \end{array}}$$

I saw the bear
0 1 2 3

$NNP_0 \rightarrow$ I

$NNP_1 \rightarrow$ saw

But how do you actually *parse*?

- CFG operations showed how to *generate* a tree, not put a tree on a given string.

- We're intersecting two languages, so we can use the *Bar-Hillel* construction: annotate the CFG rules with indices corresponding to the words in the string.

$$\boxed{\begin{array}{l} NP \rightarrow NP\ NNP \\ NP \rightarrow NNP\ NNP \\ NNP \rightarrow \text{I} \mid \text{saw} \mid \text{the} \mid \text{bear} \end{array}}$$

I saw the bear
0 1 2 3

$NNP_0 \rightarrow$ I

$NNP_1 \rightarrow$ saw

$NNP_2 \rightarrow$ the

But how do you actually *parse*?

- CFG operations showed how to *generate* a tree, not put a tree on a given string.

- We're intersecting two languages, so we can use the *Bar-Hillel* construction: annotate the CFG rules with indices corresponding to the words in the string.

$$NP \rightarrow NP\ NNP$$
$$NP \rightarrow NNP\ NNP$$
$$NNP \rightarrow I \mid saw \mid the \mid bear$$

I saw the bear
0 1 2 3

$NNP_0 \rightarrow$ I $\quad NNP_2 \rightarrow$ the

$NNP_1 \rightarrow$ saw $\quad NNP_3 \rightarrow$ bear

But how do you actually *parse*?

- CFG operations showed how to *generate* a tree, not put a tree on a given string.
- We're intersecting two languages, so we can use the *Bar-Hillel* construction: annotate the CFG rules with indices corresponding to the words in the string.

$NP \rightarrow NP\ NNP$
$NP \rightarrow NNP\ NNP$
$NNP \rightarrow$ I | saw | the | bear

I saw the bear
0 1 2 3

$NP_{i\_j} \rightarrow NP_{i\_j-1}\ NNP_j$

$NNP_0 \rightarrow$ I
$NNP_1 \rightarrow$ saw
$NNP_2 \rightarrow$ the
$NNP_3 \rightarrow$ bear

But how do you actually *parse*?

- CFG operations showed how to *generate* a tree, not put a tree on a given string.
- We're intersecting two languages, so we can use the *Bar-Hillel* construction: annotate the CFG rules with indices corresponding to the words in the string.

$\mathrm{NP} \rightarrow \mathrm{NP}\ \mathrm{NNP}$
$\mathrm{NP} \rightarrow \mathrm{NNP}\ \mathrm{NNP}$
$\mathrm{NNP} \rightarrow$ I | saw | the | bear

I saw the bear
0 1 2 3

$\mathrm{NP}_{i\_j} \rightarrow \mathrm{NP}_{i\_j-1}\ \mathrm{NNP}_j$

$\mathrm{NP}_{i\_i+1} \rightarrow \mathrm{NNP}_i\ \mathrm{NNP}_{i+1}$

$\mathrm{NNP}_0 \rightarrow$ I $\quad \mathrm{NNP}_2 \rightarrow$ the
$\mathrm{NNP}_1 \rightarrow$ saw $\quad \mathrm{NNP}_3 \rightarrow$ bear

But how do you actually *parse*?

- CFG operations showed how to *generate* a tree, not put a tree on a given string.

- We're intersecting two languages, so we can use the *Bar-Hillel* construction: annotate the CFG rules with indices corresponding to the words in the string.

$$0 \le i < j \le 3$$

$$\boxed{\begin{array}{l} NP \rightarrow NP\ NNP \\ NP \rightarrow NNP\ NNP \\ NNP \rightarrow \text{I} \mid \text{saw} \mid \text{the} \mid \text{bear} \end{array}}$$

I saw the bear
0 1 2 3

$$NP_{i\_j} \rightarrow NP_{i\_j-1}\ NNP_j$$

$$NP_{i\_i+1} \rightarrow NNP_i\ NNP_{i+1}$$

$NNP_0 \rightarrow \text{I}$ $\quad$ $NNP_2 \rightarrow \text{the}$

$NNP_1 \rightarrow \text{saw}$ $\quad$ $NNP_3 \rightarrow \text{bear}$

But how do you actually *parse*?

- CFG operations showed how to *generate* a tree, not put a tree on a given string.
- We're intersecting two languages, so we can use the *Bar-Hillel* construction: annotate the CFG rules with indices corresponding to the words in the string.

$NP \rightarrow NP\ NNP$

$NP \rightarrow NNP\ NNP$

$NNP \rightarrow I \mid saw \mid the \mid bear$

I saw the bear
0 1 2 3

NP_{0_3} $\quad 0 \leq i < j \leq 3$

$NP_{i\_j} \rightarrow NP_{i\_j-1}\ NNP_j$

$NP_{i\_i+1} \rightarrow NNP_i\ NNP_{i+1}$

$NNP_0 \rightarrow I$ $\quad NNP_2 \rightarrow the$

$NNP_1 \rightarrow saw$ $\quad NNP_3 \rightarrow bear$

CKY Parsing

CKY Parsing

- Reasonably efficient — $O(n^3)$ for n words

CKY Parsing

- Reasonably efficient — $O(n^3)$ for n words
- Easy to code

CKY Parsing

- Reasonably efficient — $O(n^3)$ for n words
- Easy to code
- Easily extendable to weighted case

CKY Parsing

- Reasonably efficient — $O(n^3)$ for n words
- Easy to code
- Easily extendable to weighted case
- Requires *Chomsky Normal Form*

CKY Parsing

- Reasonably efficient — $O(n^3)$ for n words
- Easy to code
- Easily extendable to weighted case
- Requires *Chomsky Normal Form*

$$X \rightarrow Y\ Z$$

nonterminal rule

CKY Parsing

- Reasonably efficient — $O(n^3)$ for n words
- Easy to code
- Easily extendable to weighted case
- Requires *Chomsky Normal Form*

$X \rightarrow Y\ Z$	$X \rightarrow x$
nonterminal rule	terminal rule

CKY Parsing

CKY Parsing

- Input: sentence of length n+1 = <w0, w1…wn>, CFG

CKY Parsing

- Input: sentence of length n+1 = <w0, w1…wn>, CFG
- Build states of the form [X, i, j] = “I can cover from words i through j with at least one tree rooted in X”

CKY Parsing

- Input: sentence of length n+1 = <w0, w1…wn>, CFG
- Build states of the form [X, i, j] = "I can cover from words i through j with at least one tree rooted in X"
- States contain backpointers of the form {R, k} = "To satisfy my state I will use R (X -> YZ) and states [Y, i, k] and [Z, k, j]"

CKY Parsing

CKY Parsing

- Initialize: build [X, i, i+1] for every lexical rule R = X -> wi; add backpointer {R, ()}

CKY Parsing

- Initialize: build [X, i, i+1] for every lexical rule R = X -> wi; add backpointer {R, ()}

- Recursively: build [X, i, j] for every nonlexical rule R = X -> Y Z and pair of states ([Y, i, k], [Z, k, j]) where i < k < j; add backpointer {R, k}

CKY Parsing

- Initialize: build [X, i, i+1] for every lexical rule R = X -> wi; add backpointer {R, ()}
- Recursively: build [X, i, j] for every nonlexical rule R = X -> Y Z and pair of states ([Y, i, k], [Z, k, j]) where i < k < j; add backpointer {R, k}
- If state is already built, just add backpointer

CKY Parsing

- Initialize: build [X, i, i+1] for every lexical rule R = X -> wi; add backpointer {R, ()}

- Recursively: build [X, i, j] for every nonlexical rule R = X -> Y Z and pair of states ([Y, i, k], [Z, k, j]) where i < k < j; add backpointer {R, k}

- If state is already built, just add backpointer

- Goal: build [X, 0, n+1] for start symbol X

time	flies	like	an	arrow
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	[1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

time	flies	like	an	arrow
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	[1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

Length 1

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

time	flies	like	an	arrow
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	[1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

Length 2

S → NP VP

NP → DT NN

NP → time | fruit

NP → NN NNS

VP → VBP NP

VP → flies

VP → VP PP

PP → IN NP

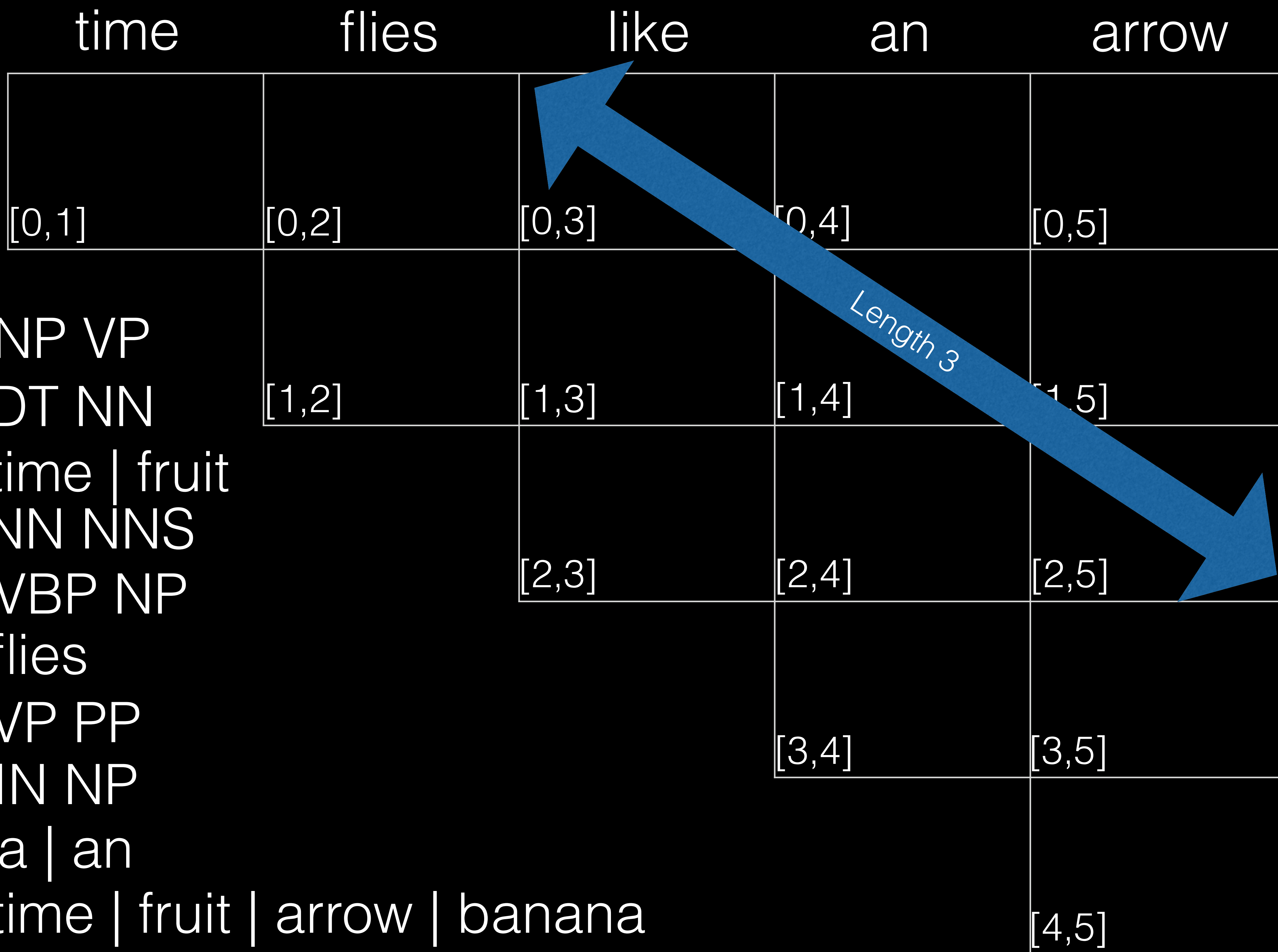
DT → a | an

NN → time | fruit | arrow | banana

NNS → flies

VBP → like

IN → like



S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

time	flies	like	an	arrow
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	[1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

Length 4

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

time	flies	like	an	arrow
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	[1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

Length 5

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

time	flies	like	an	arrow
NP NN [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	[1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

Seed length-1 cells
with lexical coverage

time	flies	like	an	arrow
NP NN [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

S → NP VP

NP → DT NN

NP → time | fruit

NP → NN NNS

VP → VBP NP

VP → flies

VP → VP PP

PP → IN NP

DT → a | an

NN → time | fruit | arrow | banana

NNS → flies

VBP → like

IN → like

Seed length-1 cells
with lexical coverage

time	flies	like	an	arrow
NP NN [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

Seed length-1 cells
with lexical coverage

time	flies	like	an	arrow
NP NN [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	[3,5]
				[4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

Seed length-1 cells
with lexical coverage

time	flies	like	an	arrow
NP NN [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	[3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | $\boxed{\text{arrow}}$ | banana
NNS → flies
VBP → like
IN → like

Seed length-1 cells
with lexical coverage

time	flies	like	an	arrow
NP NN [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	[3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

Now check length-2 spans.
Match rules to subspans

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	[3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

Two new states!

	time	flies	like	an	arrow
	NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
		VP NNS [1,2]	[1,3]	[1,4]	[1,5]
			IN VBP [2,3]	[2,4]	[2,5]
				DT [3,4]	[3,5]
					NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

Annotate with backpointer
for future reconstruction (not part of state)

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	[3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

No matching rule

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	[3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

No matching rule

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]

S → NP VP

NP → DT NN

NP → time | fruit

NP → NN NNS

VP → VBP NP

VP → flies

VP → VP PP

PP → IN NP

DT → a | an

NN → time | fruit | arrow | banana

NNS → flies

VBP → like

IN → like

New state!

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

Check every split point for rule

	time	flies	like	an	arrow
	NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
		VP NNS [1,2]	[1,3]	[1,4]	[1,5]
			IN VBP [2,3]	[2,4]	[2,5]
				DT [3,4]	NP [3,5]
					NN [4,5]

$S \rightarrow NP\ VP$

$NP \rightarrow DT\ NN$

$NP \rightarrow$ time | fruit

$NP \rightarrow NN\ NNS$

$VP \rightarrow VBP\ NP$

$VP \rightarrow$ flies

$VP \rightarrow VP\ PP$

$PP \rightarrow IN\ NP$

$DT \rightarrow$ a | an

$NN \rightarrow$ time | fruit | arrow | banana

$NNS \rightarrow$ flies

$VBP \rightarrow$ like

$IN \rightarrow$ like

Empty cell

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

No matching rule

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

(Skipping)

	time	flies	like	an	arrow
	NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
		VP NNS [1,2]	[1,3]	[1,4]	[1,5]
			IN VBP [2,3]	[2,4]	PP VP [2,5]
				DT [3,4]	NP [3,5]
					NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

Two new states!

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

Empty cell

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

(Skipping)

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
$\boxed{\text{VP → VP PP}}$
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

New state!

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

Empty Cell

	time	flies	like	an	arrow
	NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
		VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
			IN VBP [2,3]	[2,4]	PP VP [2,5]
				DT [3,4]	NP [3,5]
					NN [4,5]

S $\rightarrow$ NP VP
NP $\rightarrow$ DT NN
NP $\rightarrow$ time | fruit
NP $\rightarrow$ NN NNS
VP $\rightarrow$ VBP NP
VP $\rightarrow$ flies
VP $\rightarrow$ VP PP
PP $\rightarrow$ IN NP
DT $\rightarrow$ a | an
NN $\rightarrow$ time | fruit | arrow | banana
NNS $\rightarrow$ flies
VBP $\rightarrow$ like
IN $\rightarrow$ like

Empty Cell

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]

S → NP VP

NP → DT NN

NP → time | fruit

NP → NN NNS

VP → VBP NP

VP → flies

VP → VP PP

PP → IN NP

DT → a | an

NN → time | fruit | arrow | banana

NNS → flies

VBP → like

IN → like

New state!

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]

S → NP VP

NP → DT NN

NP → time | fruit

NP → NN NNS

VP → VBP NP

VP → flies

VP → VP PP

PP → IN NP

DT → a | an

NN → time | fruit | arrow | banana

NNS → flies

VBP → like

IN → like

Alternate (no new state)

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

Empty Cell

	time	flies	like	an	arrow
	NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
		VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
			IN VBP [2,3]	[2,4]	PP VP [2,5]
				DT [3,4]	NP [3,5]
					NN [4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

Empty Cell

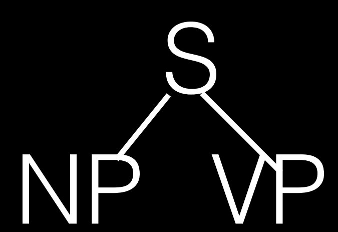
time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]

Start at top of the chart
Follow backpointers
to build tree

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]

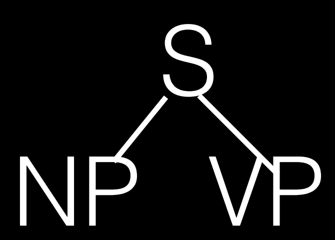
S

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]



Backpointer: $S \rightarrow NP\ VP$,1

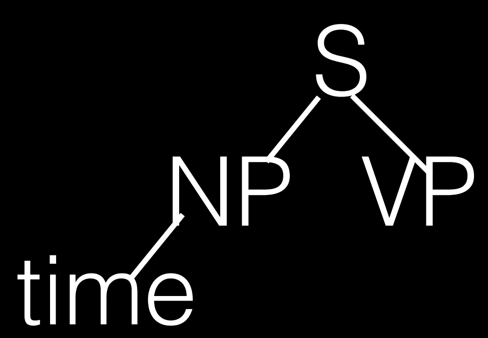
time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]



(could have used $S \rightarrow NP\ VP$,2
to generate the other tree)

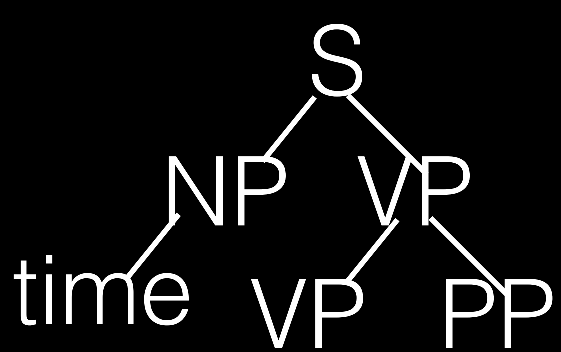
Backpointer: $S \rightarrow NP\ VP$,1

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]



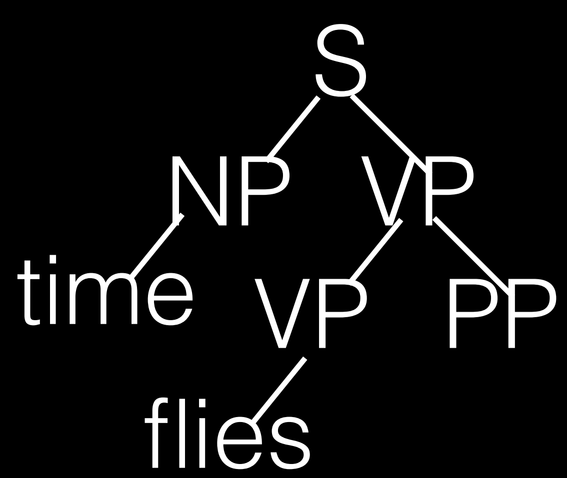
Backpointer: NP → time

	time	flies	like	an	arrow
	NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
		VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
			IN VBP [2,3]	[2,4]	PP VP [2,5]
				DT [3,4]	NP [3,5]
					NN [4,5]



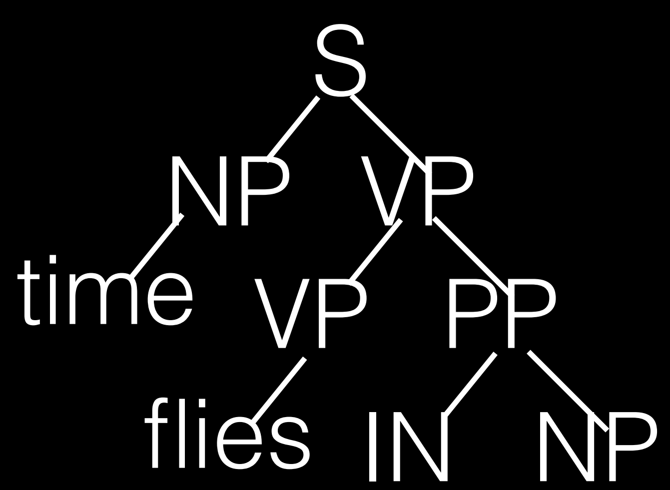
Backpointer: VP → VP PP ,2

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]



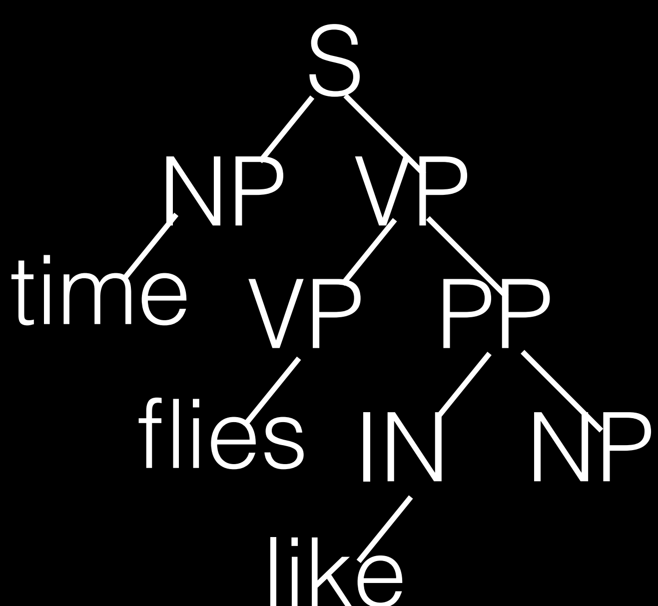
Backpointer: VP → flies

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]



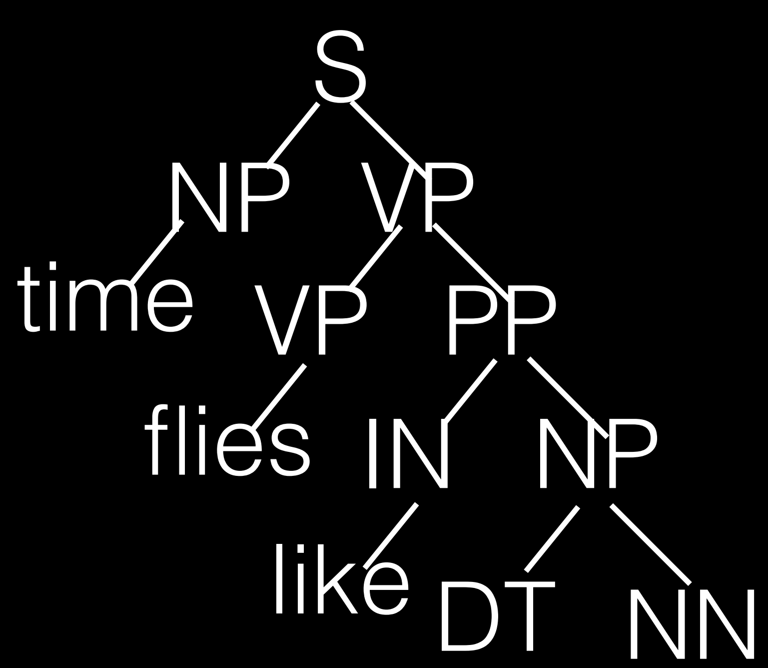
Backpointer: PP → IN NP ,3

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]



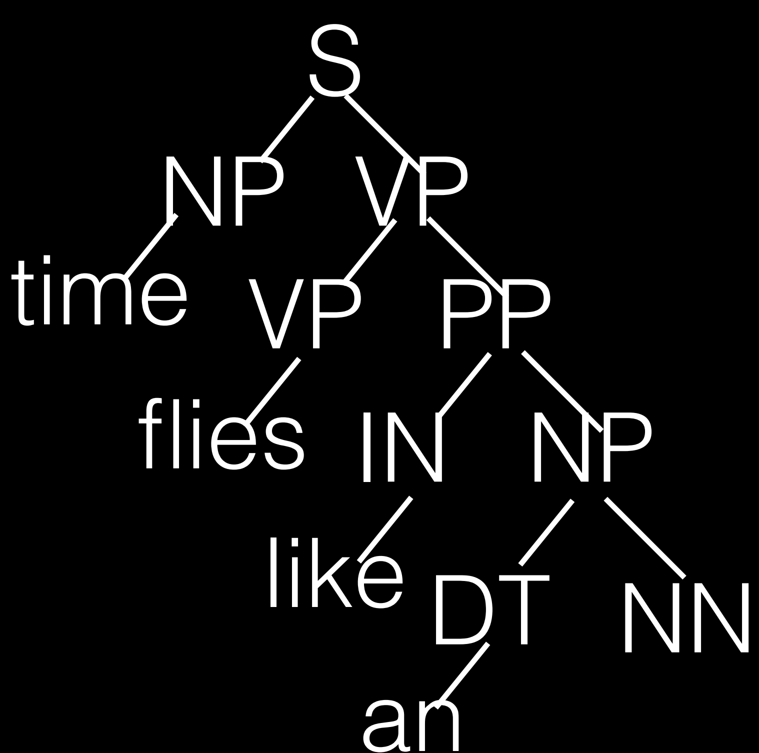
Backpointer: IN → like

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	$\boxed{\text{NP}}$ [3,5]
				NN [4,5]



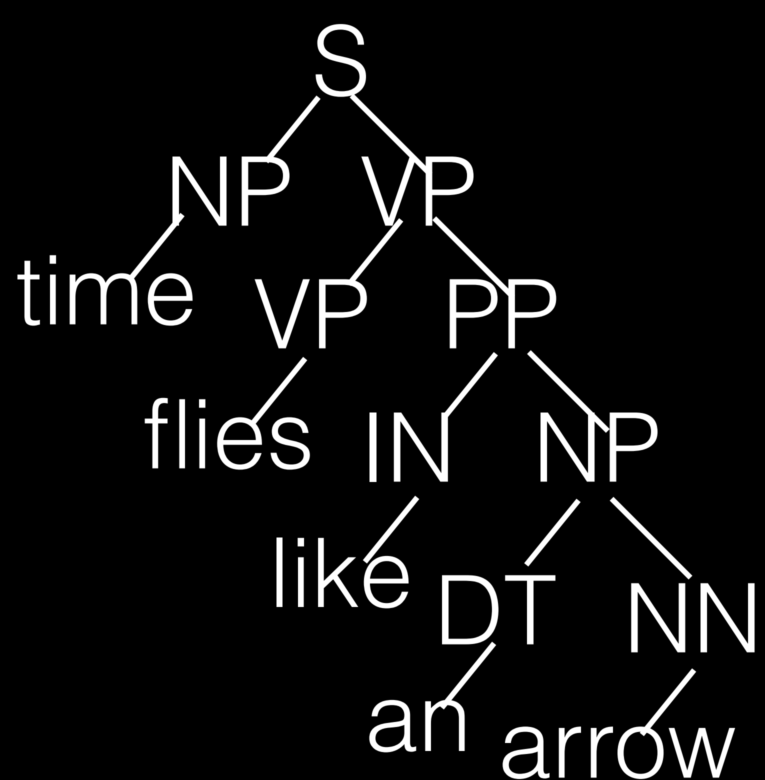
Backpointer: NP → DT NN ,4

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]



Backpointer: DT → an

time	flies	like	an	arrow
NP NN [0,1]	S NP [0,2]	[0,3]	[0,4]	S [0,5]
	VP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]



Backpointer: NN$\rightarrow$ arrow

Other things to do with CKY

Other things to do with CKY

- Generate entire forest of trees: Keep all backpointers instead of just one

Other things to do with CKY

- Generate entire forest of trees: Keep all backpointers instead of just one
- Just determine coverage: No need to keep any backpointers

Other things to do with CKY

- Generate entire forest of trees: Keep all backpointers instead of just one
- Just determine coverage: No need to keep any backpointers
- Count trees:

Other things to do with CKY

- Generate entire forest of trees: Keep all backpointers instead of just one
- Just determine coverage: No need to keep any backpointers
- Count trees:

  $$C(X[i:i+1]) = |X \to w_i|$$

Other things to do with CKY

- Generate entire forest of trees: Keep all backpointers instead of just one
- Just determine coverage: No need to keep any backpointers
- Count trees:

$$C(X[i:i+1]) = |X \to w_i|$$

$$C(X[i:j]) = \sum_{k=i+1}^{j-1} \sum_{X \to YZ} C(Y[i:k]) \times C(Z[k:j])$$

Where do grammars come from?

Where do grammars come from?

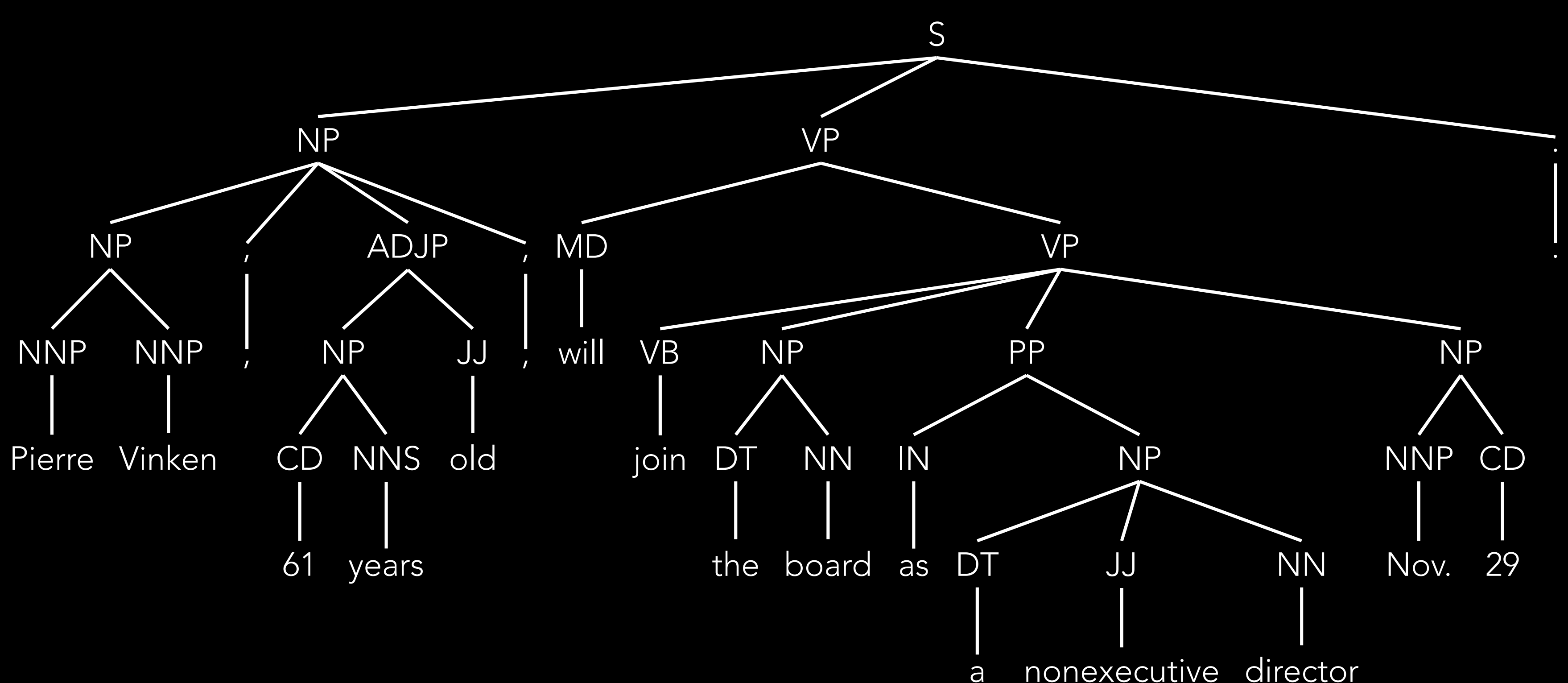
- You could write them by hand…

Where do grammars come from?

- You could write them by hand…
- But why not read them off of real trees?

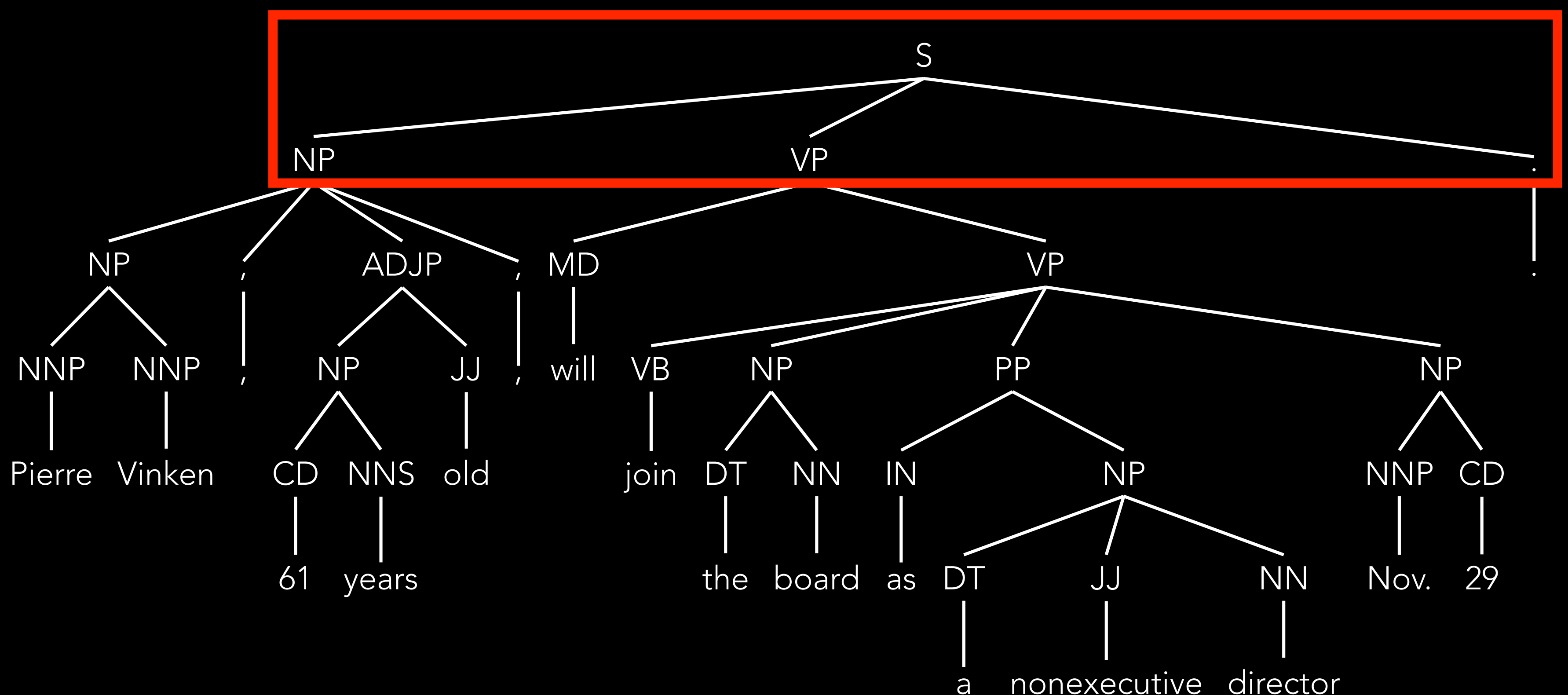
Where do grammars come from?

- You could write them by hand…
- But why not read them off of real trees?



Where do grammars come from?

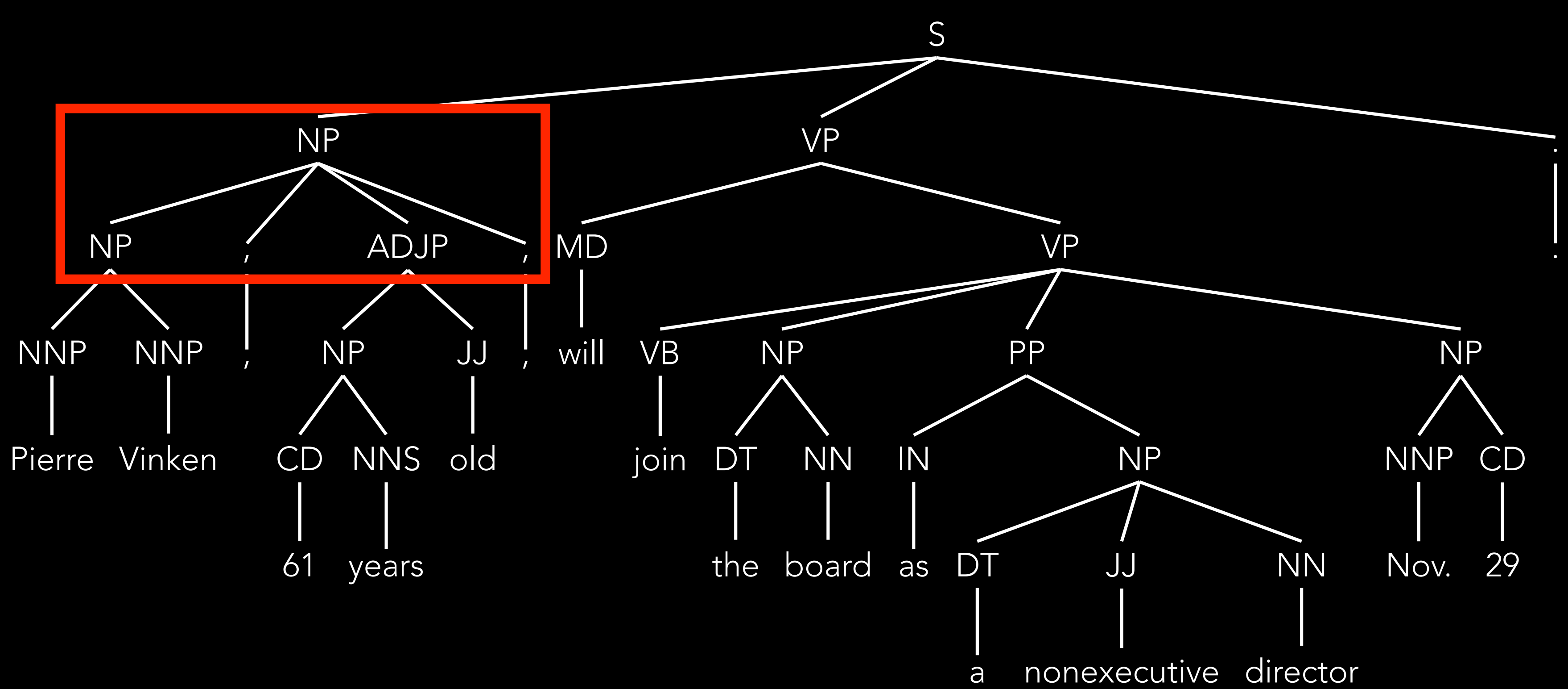
- You could write them by hand…
- But why not read them off of real trees?



$S \rightarrow NP\ VP\ .$

Where do grammars come from?

- You could write them by hand…
- But why not read them off of real trees?

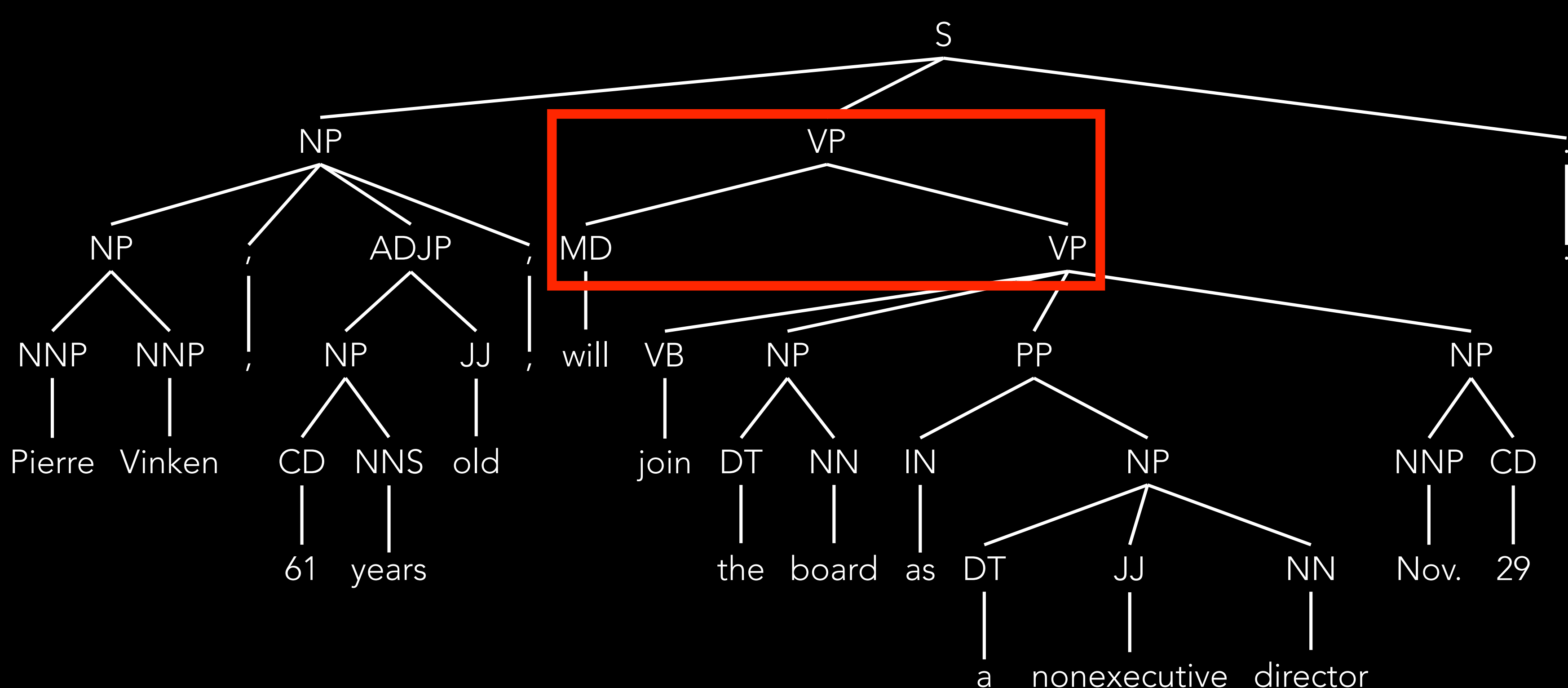


S → NP VP .

NP → NP , ADJP ,

Where do grammars come from?

- You could write them by hand…
- But why not read them off of real trees?



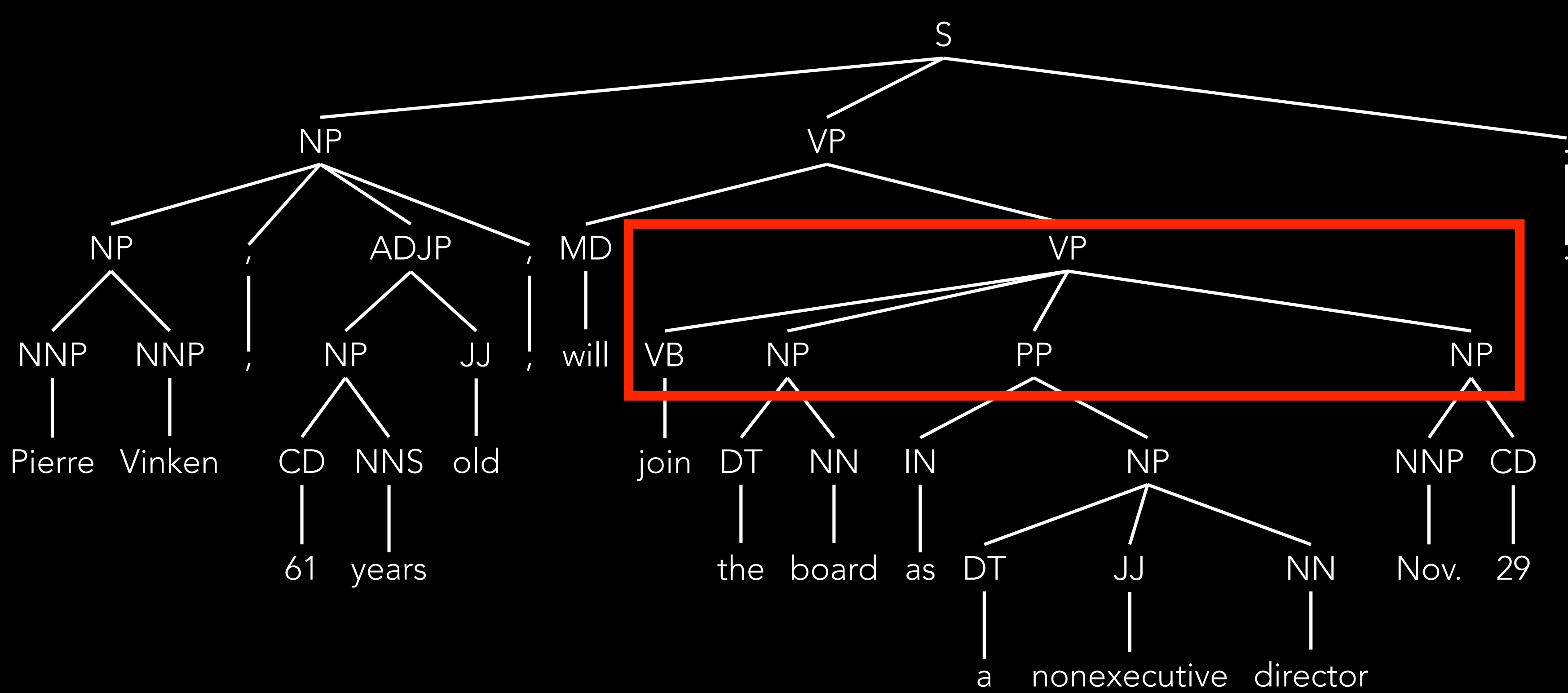
S → NP VP .

NP → NP , ADJP ,

VP → MD VP

Where do grammars come from?

- You could write them by hand…
- But why not read them off of real trees?



$$S \rightarrow NP\ VP\ .$$

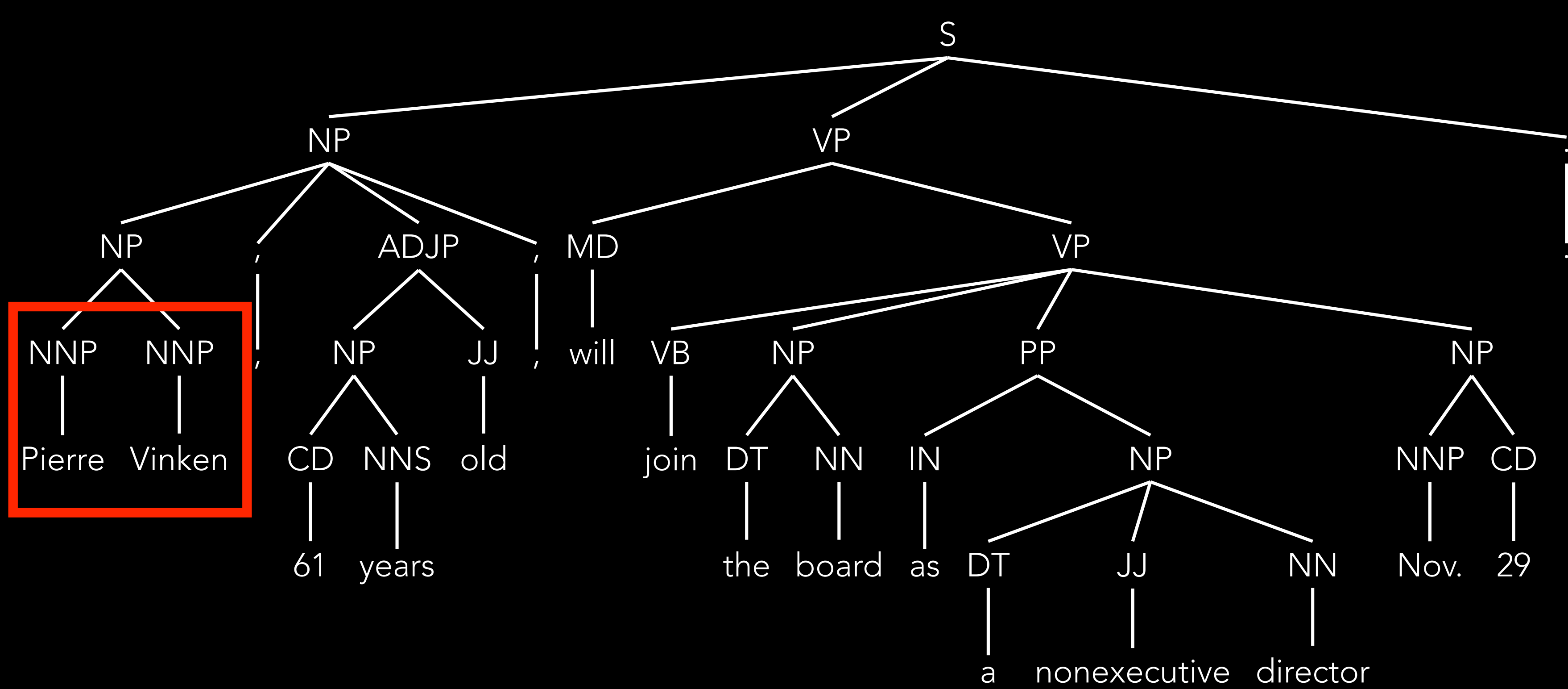
$$NP \rightarrow NP\ ,\ ADJP\ ,$$

$$VP \rightarrow MD\ VP$$

$$VP \rightarrow VB\ NP\ PP\ NP$$

Where do grammars come from?

- You could write them by hand…
- But why not read them off of real trees?



S → NP VP .

NP → NP , ADJP ,

VP → MD VP

VP → VB NP PP NP

…

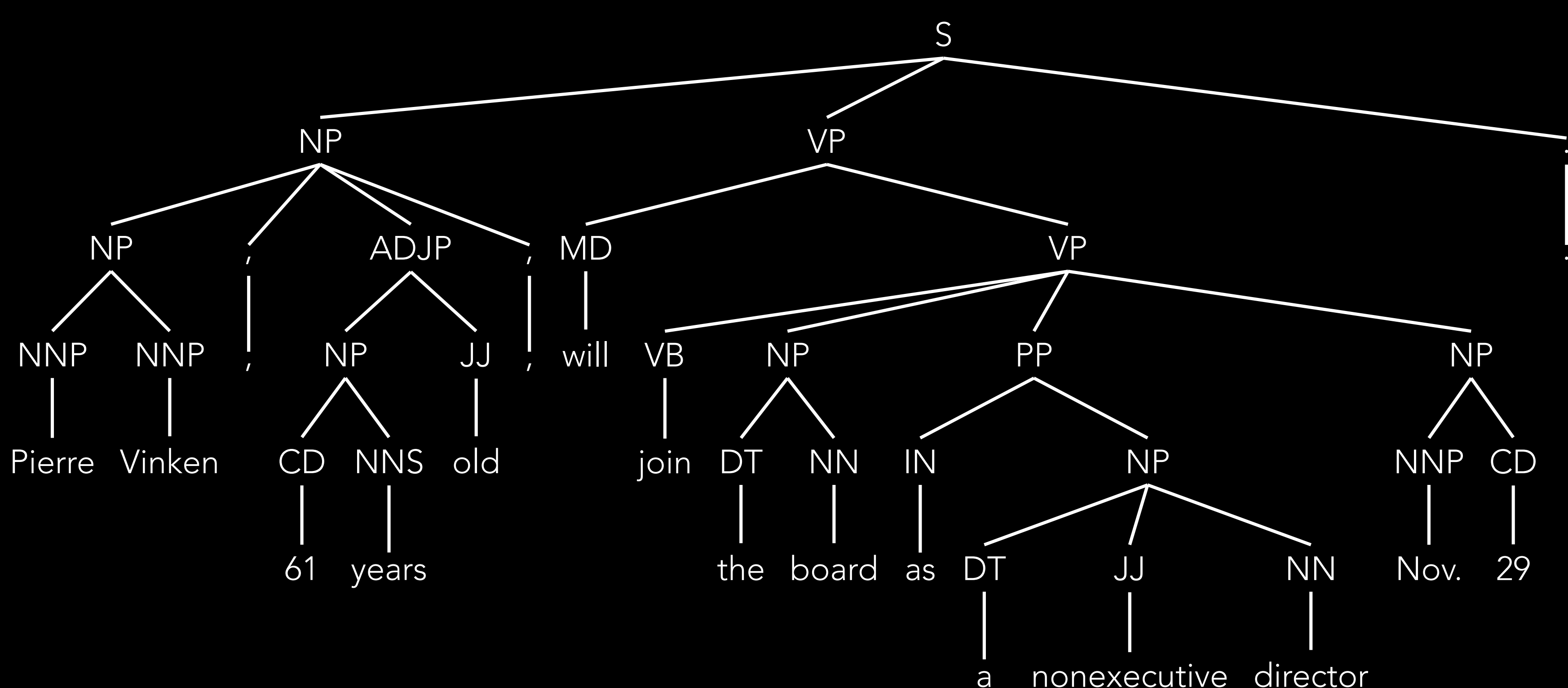
NNP → Pierre

NNP → Vinken

…

Where do grammars come from?

- You could write them by hand…
- But why not read them off of real trees?



S → NP VP .

NP → NP , ADJP ,

VP → MD VP

VP → VB NP PP NP

…

NNP → Pierre

NNP → Vinken

…

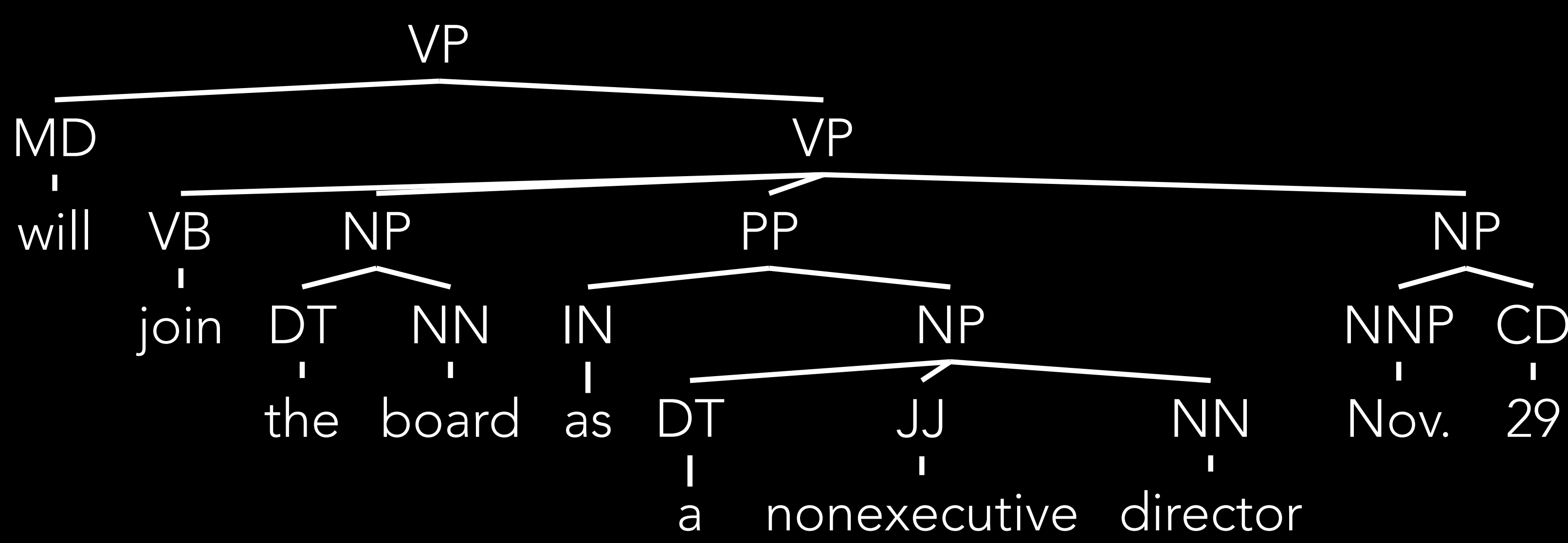
Chomsky Normal Form Violation!

CFG Normalization?

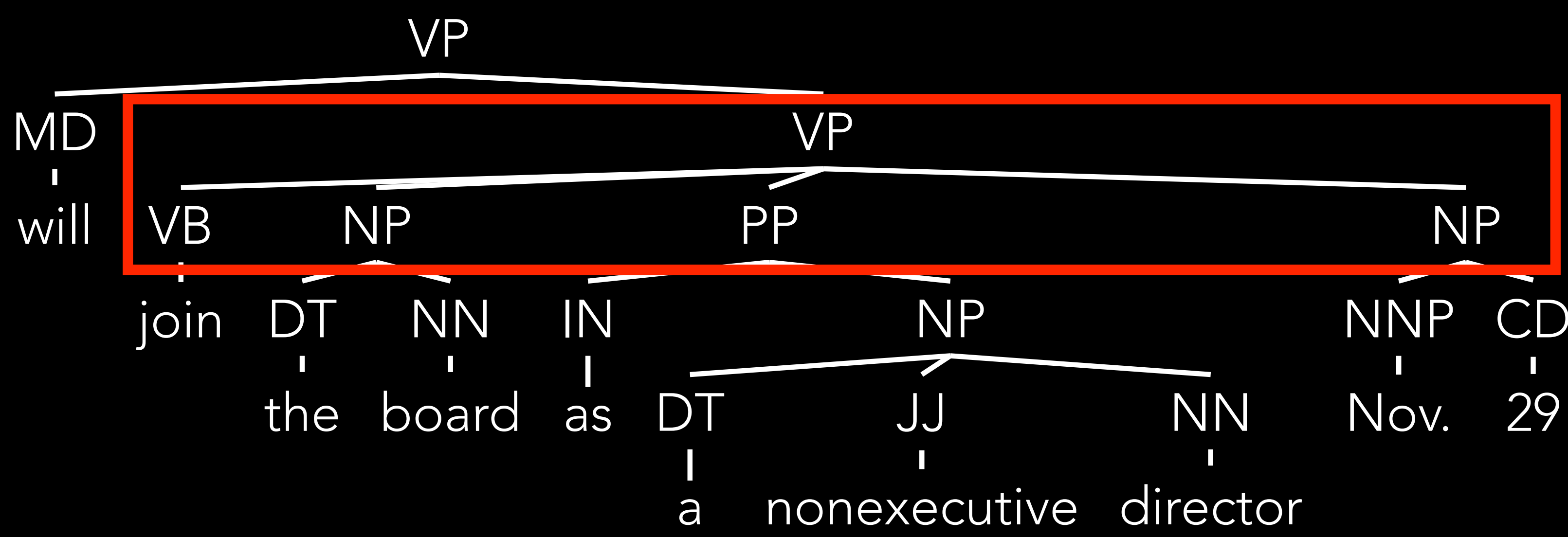
- There are well known approaches to normalizing CFGs into CNF
- However, it’s far simpler to just modify the *trees*

CFG Normalization

CFG Normalization



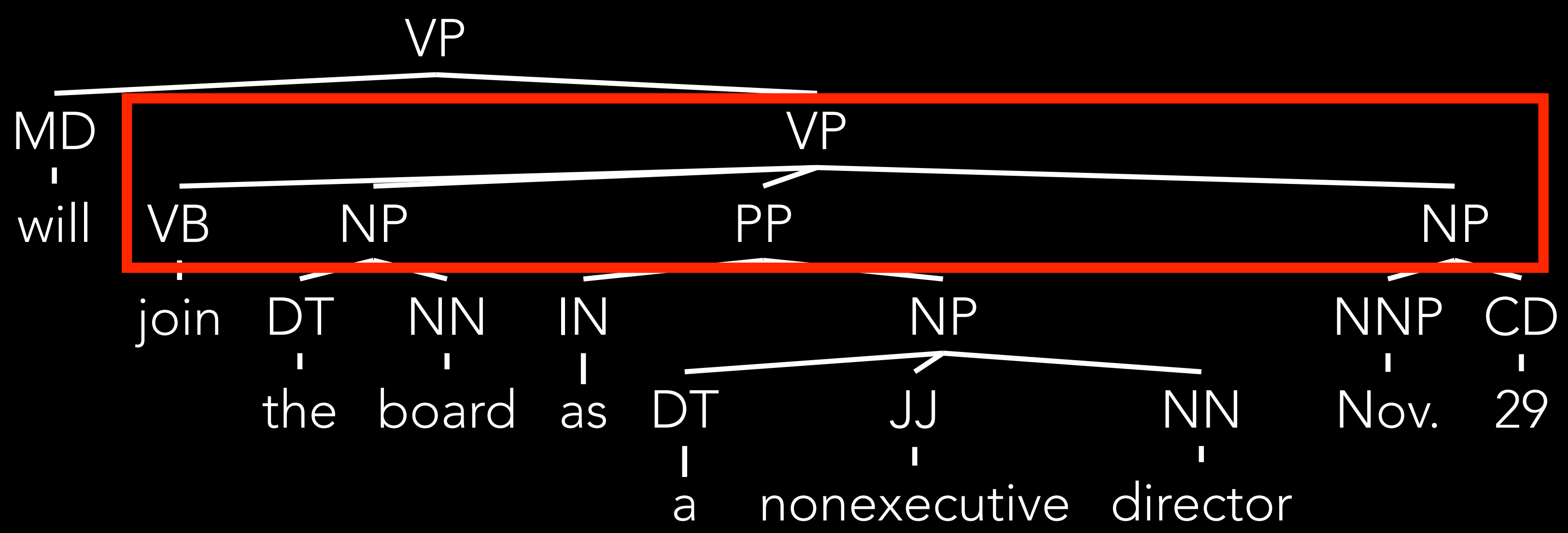
CFG Normalization



VP -> VB NP PP NP

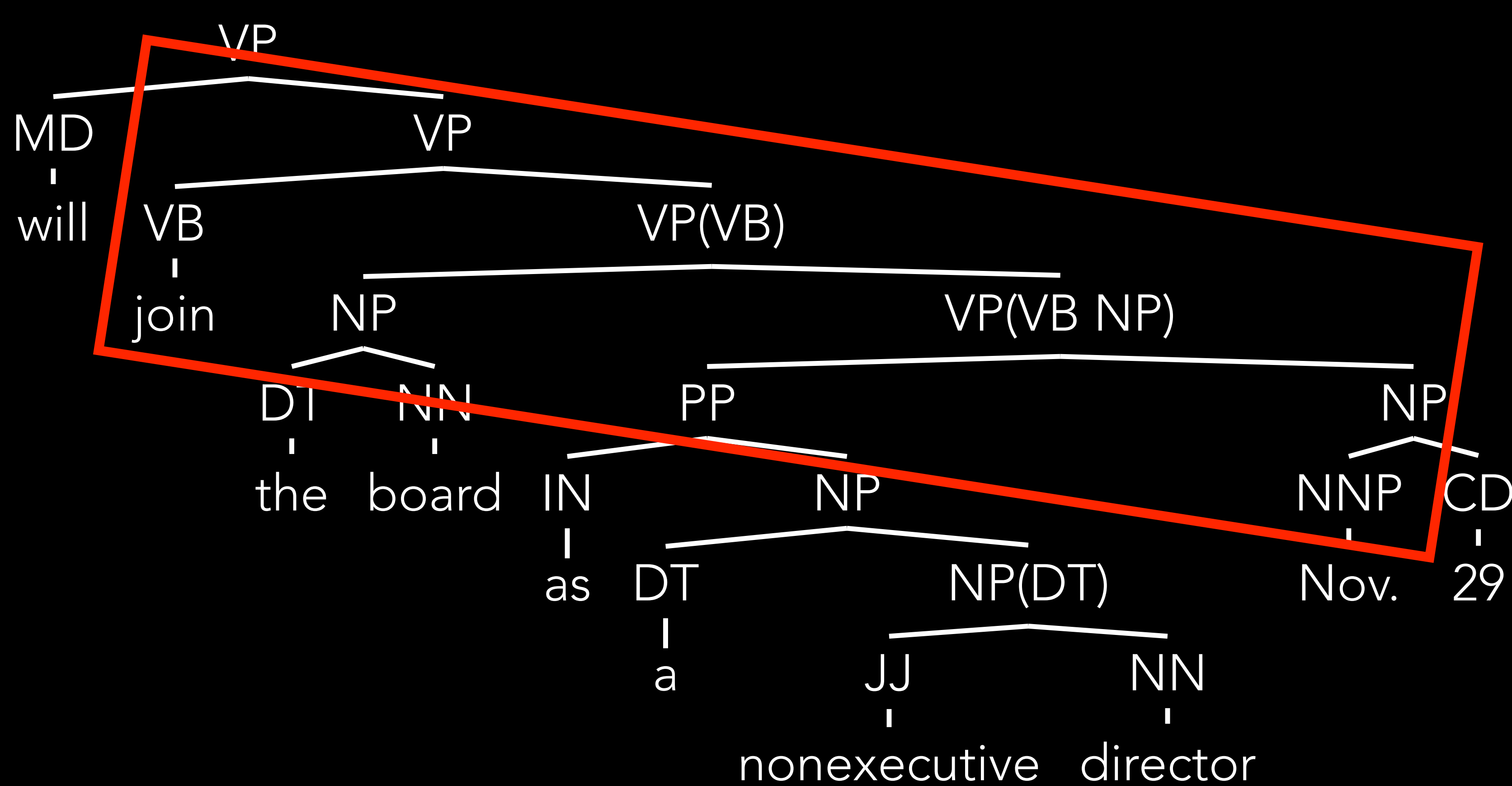
not ok for CKY

CFG Normalization



VP -> VB NP PP NP

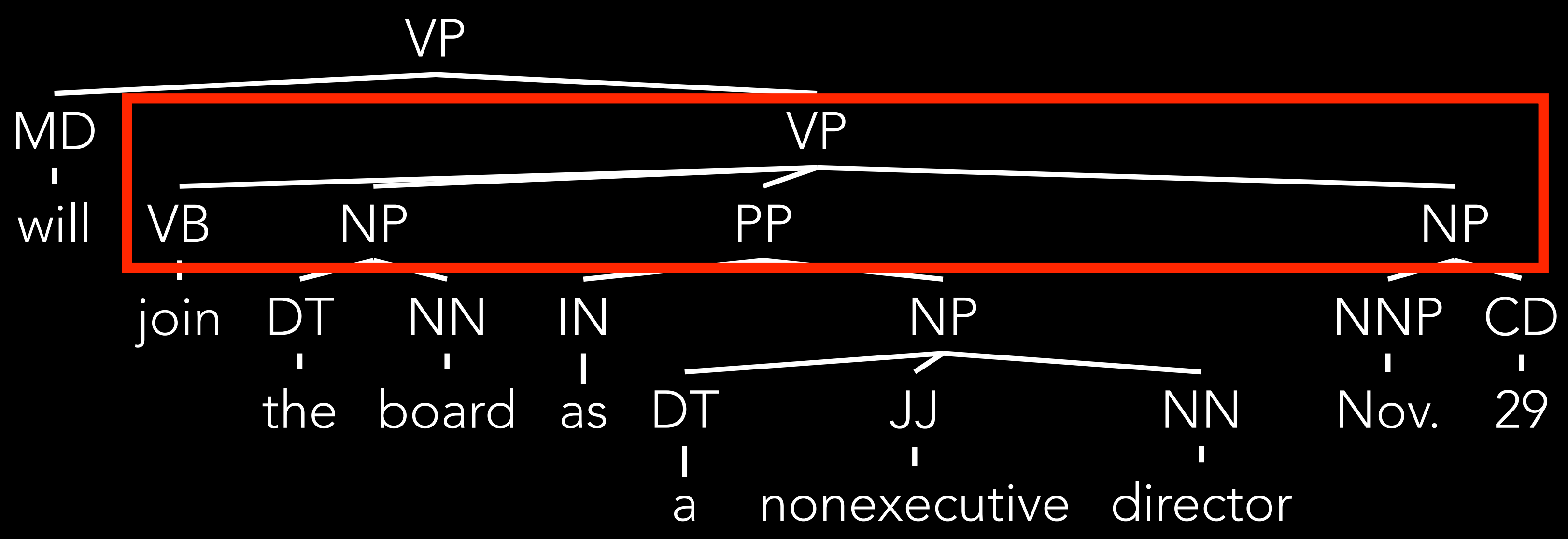
not ok for CKY



Three binary rules

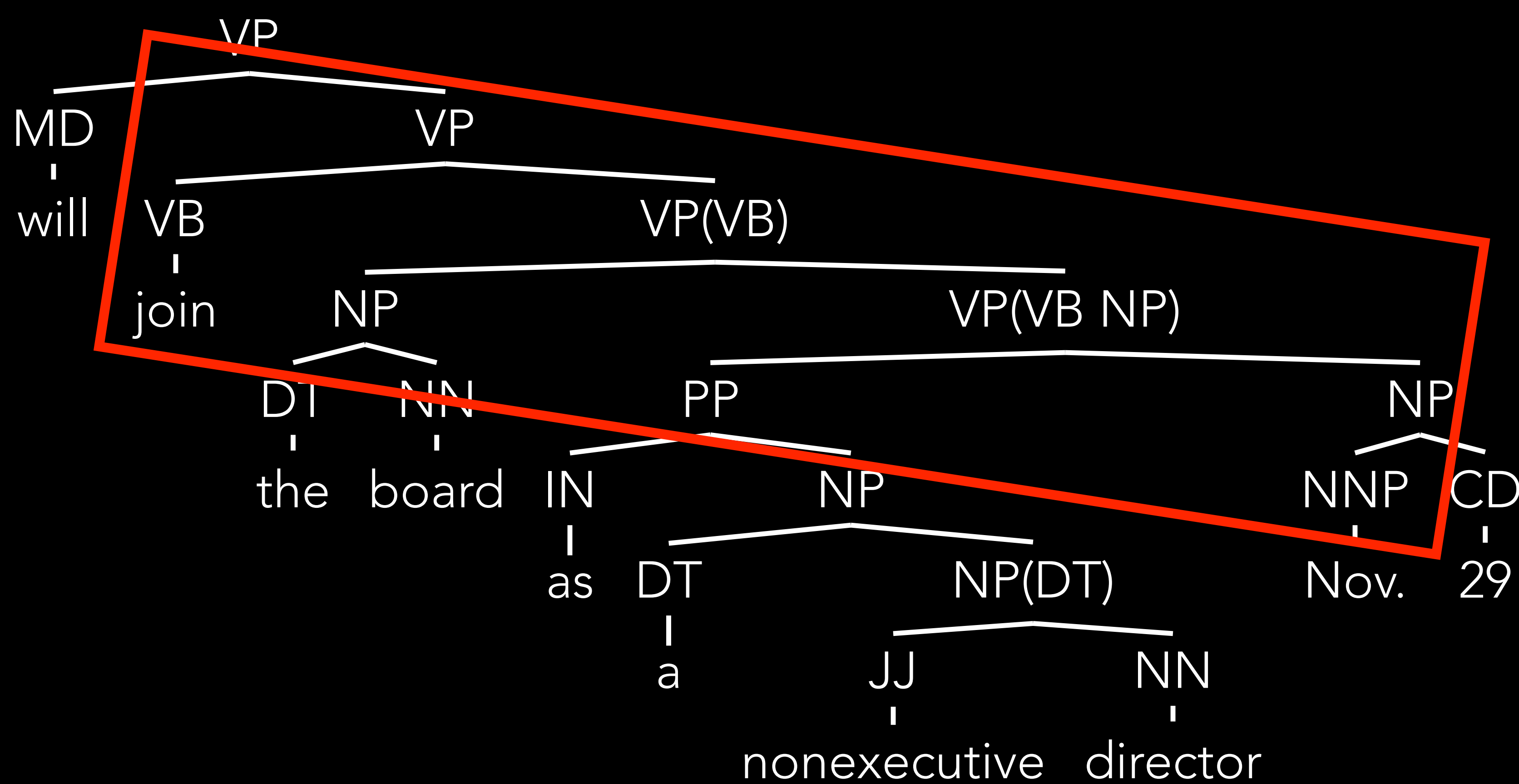
ok for CKY

CFG Normalization



VP -> VB NP PP NP

not ok for CKY

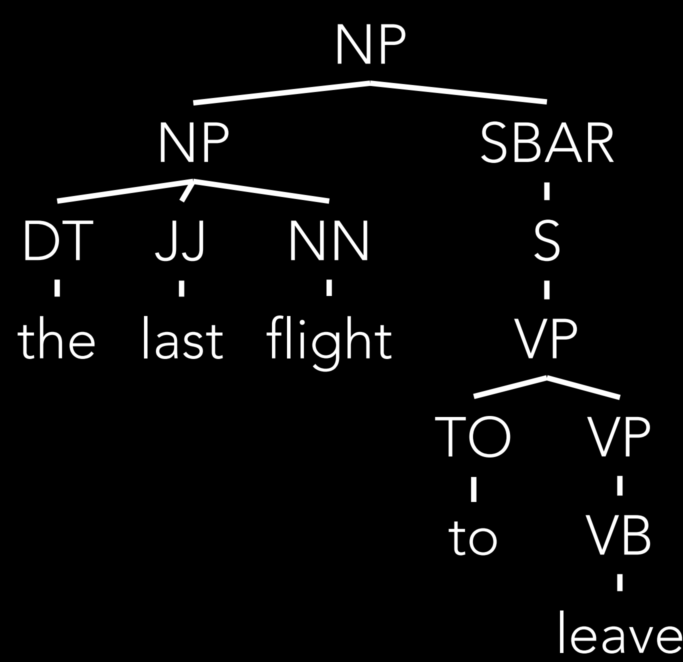


Three binary rules ok for CKY

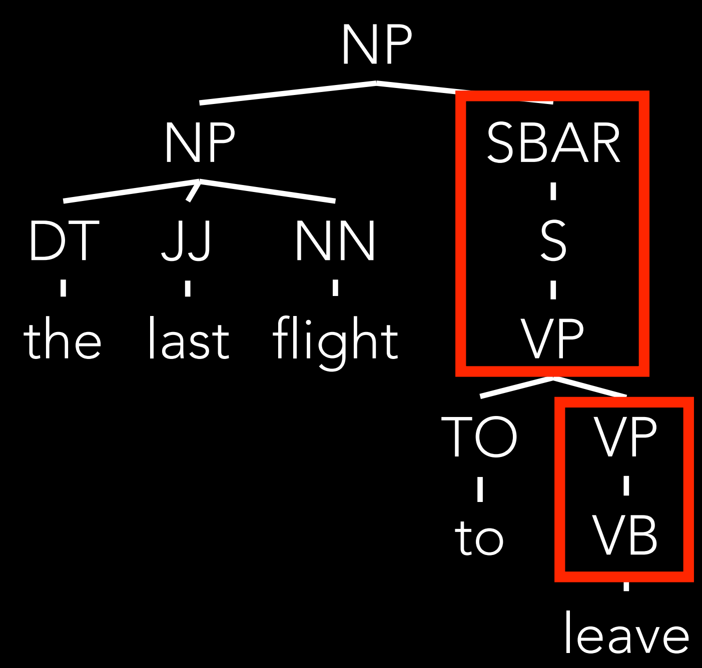
Why are the labels important?

What about unary trees?

What about unary trees?

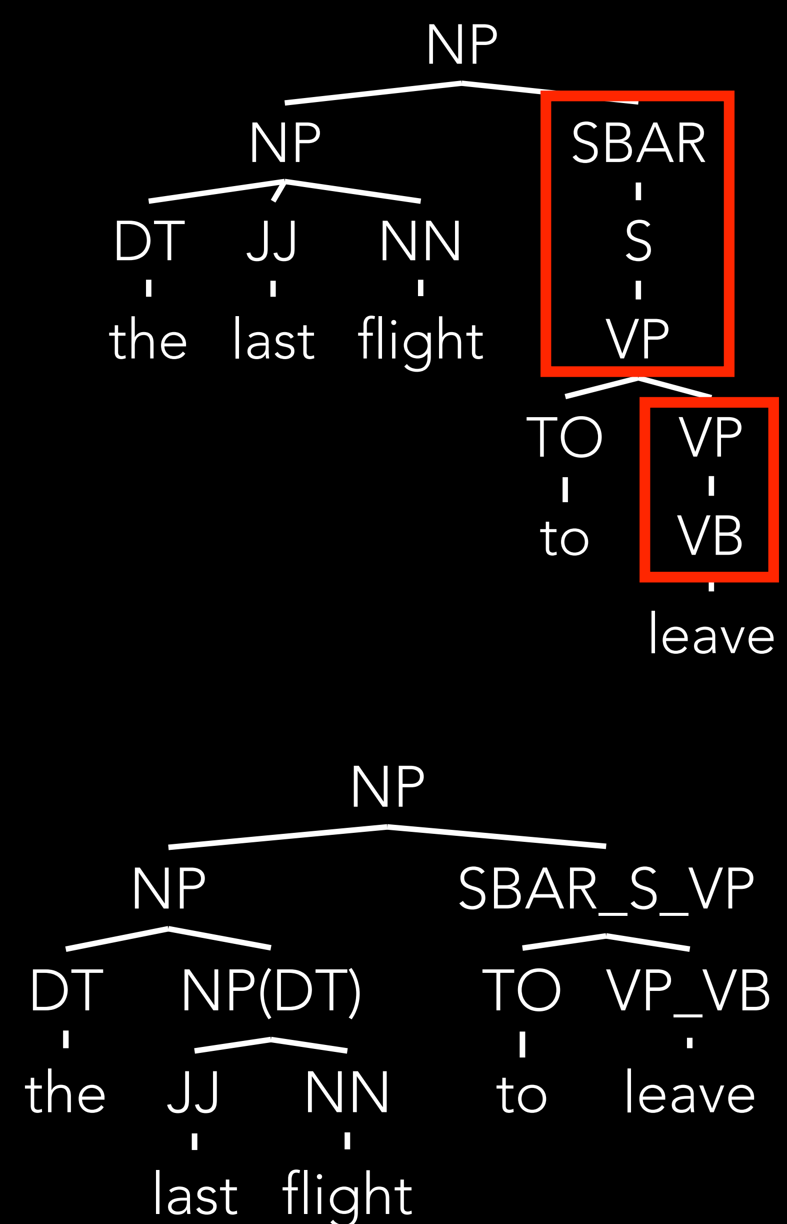


What about unary trees?



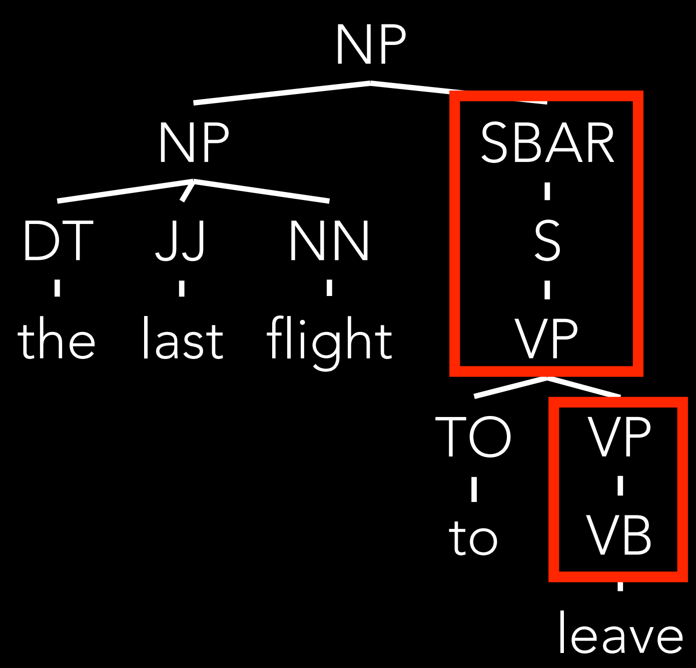
SBAR -> S, S->VP,
VP-> VB
not good for CKY

What about unary trees?

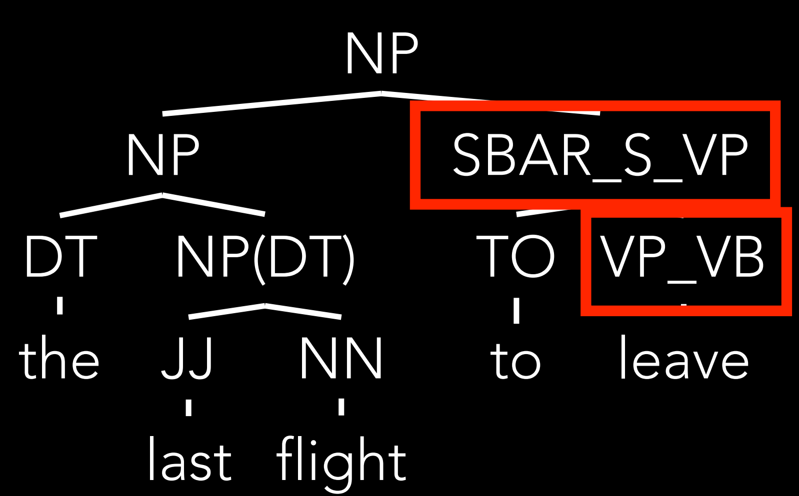


SBAR -> S, S->VP,
VP-> VB
not good for CKY

What about unary trees?



SBAR -> S, S->VP,
VP-> VB
not good for CKY



Collapse into single label

Review

Review

- Parsing (generating syntax trees) is a useful task

Review

- Parsing (generating syntax trees) is a useful task
- The Treebank is a resource of real-world syntax trees

Review

- Parsing (generating syntax trees) is a useful task
- The Treebank is a resource of real-world syntax trees
- Parseval metric for scoring parse trees

Review

- Parsing (generating syntax trees) is a useful task
- The Treebank is a resource of real-world syntax trees
- Parseval metric for scoring parse trees
- Context-Free Grammars

Review

- Parsing (generating syntax trees) is a useful task
- The Treebank is a resource of real-world syntax trees
- Parseval metric for scoring parse trees
- Context-Free Grammars
- CKY parsing

Review

- Parsing (generating syntax trees) is a useful task
- The Treebank is a resource of real-world syntax trees
- Parseval metric for scoring parse trees
- Context-Free Grammars
- CKY parsing
- Extracting grammars from data

Review

- Parsing (generating syntax trees) is a useful task
- The Treebank is a resource of real-world syntax trees
- Parseval metric for scoring parse trees
- Context-Free Grammars
- CKY parsing
- Extracting grammars from data
- Tree normalization

Review

- Parsing (generating syntax trees) is a useful task
- The Treebank is a resource of real-world syntax trees
- Parseval metric for scoring parse trees
- Context-Free Grammars
- CKY parsing
- Extracting grammars from data
- Tree normalization
- CKY extension to unary nonterminal rules

Review

- Parsing (generating syntax trees) is a useful task
- The Treebank is a resource of real-world syntax trees
- Parseval metric for scoring parse trees
- Context-Free Grammars
- CKY parsing
- Extracting grammars from data
- Tree normalization
- CKY extension to unary nonterminal rules

Alternate: Modify CKY!

- Initialize: build [X, i, i+1] for every lexical rule R = X -> w_i; add backpointer {R, ()}
- Recursively: build [X, i, j] for every nonlexical rule R = X -> Y Z and pair of states ([Y, i, k], [Z, k, j]) where i < k < j; add backpointer {R, k}
- If state is already built, just add backpointer

Alternate: Modify CKY!

- Initialize: build [X, i, i+1] for every lexical rule R = X -> w_i; add backpointer {R, ()}
- Recursively:
  - build [X, i, j] for every nonlexical rule R = X -> Y Z and pair of states ([Y, i, k], [Z, k, j]) where i < k < j; add backpointer {R, k}
  - build [X, i, j] for every nonlexical rule R = X -> Y and state [Y, i, j] **if [X, i, j] does not exist**; add backpointer {R, ()}
- If state is already built, just add backpointer

time	flies	like	an	arrow
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	[1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

S → NP VP
NP → DT NN
NP → time | fruit
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

time	flies	like	an	arrow
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	[1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

More realistic!

time	flies	like	an	arrow
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	[1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

Not typical, but for illustration.

	time	flies	like	an	arrow
	NN [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
		VP NNS [1,2]	[1,3]	[1,4]	[1,5]
			IN VBP [2,3]	[2,4]	[2,5]
				DT [3,4]	[3,5]
					NN [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

After seeding the chart.
We need to check length-1 spans
for nonlexical rules

time	flies	like	an	arrow
NP NN [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	[3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

New state!

time	flies	like	an	arrow
NP PP NN [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	[3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

New state!

	time	flies	like	an	arrow
	NP PP NN VP [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
		VP NNS [1,2]	[1,3]	[1,4]	[1,5]
			IN VBP [2,3]	[2,4]	[2,5]
				DT [3,4]	[3,5]
					NN [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

New state!

time	flies	like	an	arrow
NP PP NN VP [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	VP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	[3,5]
				NN [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

No new state, and no backpointer
(would lead to infinite loop)

	time	flies	like	an	arrow
	NP PP NN VP [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
		PP VP NP NNS [1,2]	[1,3]	[1,4]	[1,5]
			IN VBP [2,3]	[2,4]	[2,5]
				DT [3,4]	[3,5]
					NN [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

$\boxed{\text{PP → NP}}$
VP → PP
$\boxed{\text{NP → VP}}$

Two new states!

time	flies	like	an	arrow
NP PP NN VP [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	PP VP NP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	[3,5]
				NP PP NN VP [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

Three new states!

time	flies	like	an	arrow
NP PP NN VP [0,1]	S NP [0,2]	[0,3]	[0,4]	[0,5]
	PP VP NP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	[3,5]
				NP PP NN VP [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

This is what we had before…

time	flies	like	an	arrow
NP PP NN VP [0,1]	S NP VP [0,2]	[0,3]	[0,4]	[0,5]
	PP VP NP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	[3,5]
				NP PP NN VP [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
$\boxed{\text{VP → VP PP}}$
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

…but now we can add one more
with a binary rule

time	flies	like	an	arrow
NP PP NN VP [0,1]	S NP VP PP [0,2]	[0,3]	[0,4]	[0,5]
	PP VP NP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	[2,5]
			DT [3,4]	[3,5]
				NP PP NN VP [4,5]

S → NP VP

NP → DT NN

NP → NN

NP → NN NNS

VP → VBP NP

VP → flies

VP → VP PP

PP → IN NP

DT → a | an

NN → time | fruit | arrow | banana

NNS → flies

VBP → like

IN → like

$\boxed{\text{PP → NP}}$

VP → PP

NP → VP

…and can take a unary step

	time	flies	like	an	arrow
	NP PP NN VP [0,1]	S NP VP PP [0,2]	[0,3]	[0,4]	[0,5]
		PP VP NP NNS [1,2]	[1,3]	[1,4]	[1,5]
			IN VBP [2,3]	[2,4]	[2,5]
				DT [3,4]	NP [3,5]
					NP PP NN VP [4,5]

$S \rightarrow NP\ VP$

$\boxed{NP \rightarrow DT\ NN}$

$NP \rightarrow NN$

$NP \rightarrow NN\ NNS$

$VP \rightarrow VBP\ NP$

$VP \rightarrow$ flies

$VP \rightarrow VP\ PP$

$PP \rightarrow IN\ NP$

$DT \rightarrow$ a | an

$NN \rightarrow$ time | fruit | arrow | banana

$NNS \rightarrow$ flies

$VBP \rightarrow$ like

$IN \rightarrow$ like

$PP \rightarrow NP$

$VP \rightarrow PP$

$NP \rightarrow VP$

This is what we had before

	time	flies	like	an	arrow
	NP PP NN VP [0,1]	S NP VP PP [0,2]	[0,3]	[0,4]	[0,5]
		PP VP NP NNS [1,2]	[1,3]	[1,4]	[1,5]
			IN VBP [2,3]	[2,4]	[2,5]
				DT [3,4]	NP PP VP [3,5]
					NP PP NN VP [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

Now we have two more

time	flies	like	an	arrow
NP PP NN VP [0,1]	S NP VP PP [0,2]	[0,3]	[0,4]	[0,5]
	PP VP NP NNS [1,2]	[1,3]	[1,4]	[1,5]
		IN VBP [2,3]	[2,4]	PP VP [2,5]
			DT [3,4]	NP PP VP [3,5]
				NP PP NN VP [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

This is what we had before

	time	flies	like	an	arrow
	NP PP NN VP [0,1]	S NP VP PP [0,2]	[0,3]	[0,4]	[0,5]
		PP VP NP NNS [1,2]	[1,3]	[1,4]	[1,5]
			IN VBP [2,3]	[2,4]	PP VP NP [2,5]
				DT [3,4]	NP PP VP [3,5]
					NP PP NN VP [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

Can add one from unary step

time	flies	like	an	arrow
NP PP NN VP [0,1]	S NP VP PP [0,2]	[0,3]	[0,4]	[0,5]
	PP VP NP NNS [1,2]	[1,3]	[1,4]	VP [1,5]
		IN VBP [2,3]	[2,4]	PP NP VP [2,5]
			DT [3,4]	NP PP VP [3,5]
				NP PP NN VP [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
$\boxed{\text{VP → VP PP}}$
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

This is what we had before

time	flies	like	an	arrow
NP PP NN VP [0,1]	S NP VP PP [0,2]	[0,3]	[0,4]	[0,5]
	PP VP NP NNS [1,2]	[1,3]	[1,4]	VP S [1,5]
		IN VBP [2,3]	[2,4]	PP NP VP [2,5]
			DT [3,4]	NP PP VP [3,5]
				NP PP NN VP [4,5]

$\boxed{S \rightarrow NP\ VP}$

NP → DT NN

NP → NN

NP → NN NNS

VP → VBP NP

VP → flies

VP → VP PP

PP → IN NP

DT → a | an

NN → time | fruit | arrow | banana

NNS → flies

VBP → like

IN → like

PP → NP

VP → PP

NP → VP

We can add one more non-unary now

time	flies	like	an	arrow
NP PP NN VP [0,1]	S NP VP PP [0,2]	[0,3]	[0,4]	[0,5]
	PP VP NP NNS [1,2]	[1,3]	[1,4]	VP S PP NP [1,5]
		IN VBP [2,3]	[2,4]	PP NP VP [2,5]
			DT [3,4]	NP PP VP [3,5]
				NP PP NN VP [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

And two more with unary steps.

time	flies	like	an	arrow
NP PP NN VP [0,1]	S NP VP PP [0,2]	[0,3]	[0,4]	S [0,5]
	PP VP NP NNS [1,2]	[1,3]	[1,4]	VP S PP NP [1,5]
		IN VBP [2,3]	[2,4]	PP NP VP [2,5]
			DT [3,4]	NP PP VP [3,5]
				NP PP NN VP [4,5]

$\boxed{S \rightarrow NP\ VP}$

$NP \rightarrow DT\ NN$

$NP \rightarrow NN$

$NP \rightarrow NN\ NNS$

$VP \rightarrow VBP\ NP$

$VP \rightarrow$ flies

$VP \rightarrow VP\ PP$

$PP \rightarrow IN\ NP$

$DT \rightarrow$ a | an

$NN \rightarrow$ time | fruit | arrow | banana

$NNS \rightarrow$ flies

$VBP \rightarrow$ like

$IN \rightarrow$ like

$PP \rightarrow NP$

$VP \rightarrow PP$

$NP \rightarrow VP$

We built S this way before

time	flies	like	an	arrow
NP PP NN VP [0,1]	S NP VP PP [0,2]	[0,3]	[0,4]	S VP [0,5]
	PP VP NP NNS [1,2]	[1,3]	[1,4]	VP S PP NP [1,5]
		IN VBP [2,3]	[2,4]	PP NP VP [2,5]
			DT [3,4]	NP PP VP [3,5]
				NP PP NN VP [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

We can also build a VP at this split point.

time	flies	like	an	arrow
NP PP NN VP [0,1]	S NP VP PP [0,2]	[0,3]	[0,4]	S VP [0,5]
	PP VP NP NNS [1,2]	[1,3]	[1,4]	VP S PP NP [1,5]
		IN VBP [2,3]	[2,4]	PP NP VP [2,5]
			DT [3,4]	NP PP VP [3,5]
				NP PP NN VP [4,5]

S → NP VP

NP → DT NN

NP → NN

NP → NN NNS

VP → VBP NP

VP → flies

VP → VP PP

PP → IN NP

DT → a | an

NN → time | fruit | arrow | banana

NNS → flies

VBP → like

IN → like

PP → NP

VP → PP

NP → VP

Before, we built a second backpointer to S

time	flies	like	an	arrow
NP PP NN VP [0,1]	S NP VP PP [0,2]	[0,3]	[0,4]	S VP [0,5]
	PP VP NP NNS [1,2]	[1,3]	[1,4]	VP S PP NP [1,5]
		IN VBP [2,3]	[2,4]	PP NP VP [2,5]
			DT [3,4]	NP PP VP [3,5]
				NP PP NN VP [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

Now we have a second backpointer to VP, too

time	flies	like	an	arrow
NP PP NN VP [0,1]	S NP VP PP [0,2]	[0,3]	[0,4]	S VP NP PP [0,5]
	PP VP NP NNS [1,2]	[1,3]	[1,4]	VP S PP NP [1,5]
		IN VBP [2,3]	[2,4]	PP NP VP [2,5]
			DT [3,4]	NP PP VP [3,5]
				NP PP NN VP [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

$\boxed{\text{PP → NP}}$
VP → PP
$\boxed{\text{NP → VP}}$

Finally, we have new states from unary steps.

time	flies	like	an	arrow
NP PP NN VP [0,1]	S NP VP PP [0,2]	[0,3]	[0,4]	S VP NP PP [0,5]
	PP VP NP NNS [1,2]	[1,3]	[1,4]	VP S PP NP [1,5]
		IN VBP [2,3]	[2,4]	PP NP VP [2,5]
			DT [3,4]	NP PP VP [3,5]
				NP PP NN VP [4,5]

S → NP VP
NP → DT NN
NP → NN
NP → NN NNS
VP → VBP NP
VP → flies
VP → VP PP
PP → IN NP
DT → a | an
NN → time | fruit | arrow | banana
NNS → flies
VBP → like
IN → like

PP → NP
VP → PP
NP → VP

Do we have any new trees?
Actually, no,
because S is the only start symbol.

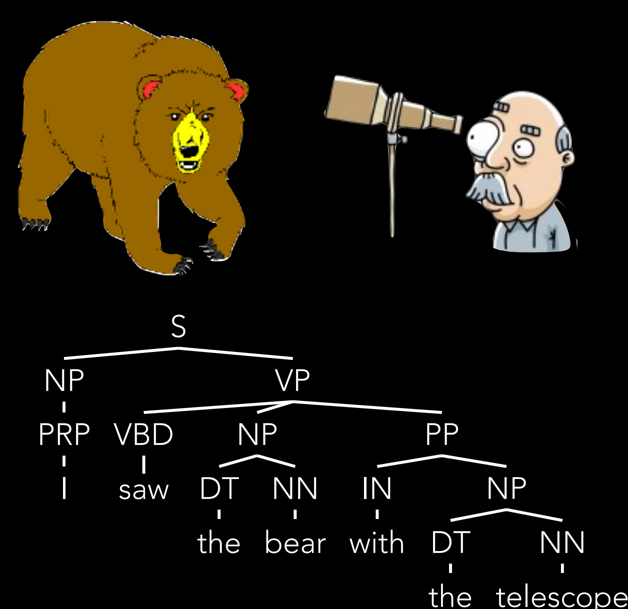
But Which Parse Is Best?

But Which Parse Is Best?

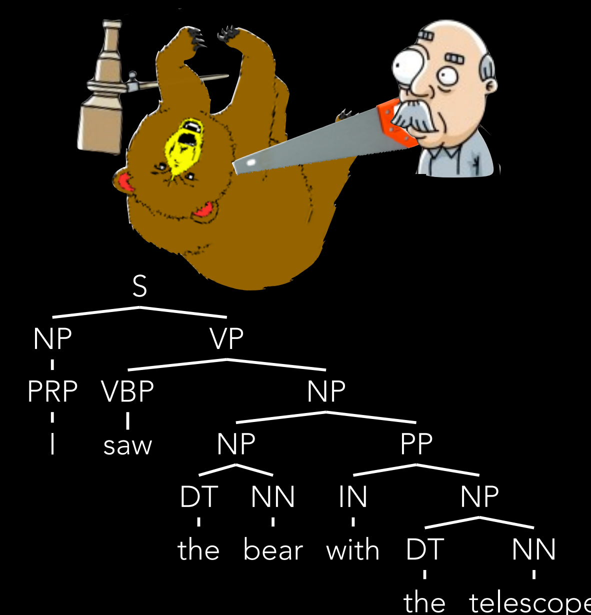
- How does our parser make decisions about which tree to produce?

But Which Parse Is Best?

- How does our parser make decisions about which tree to produce?

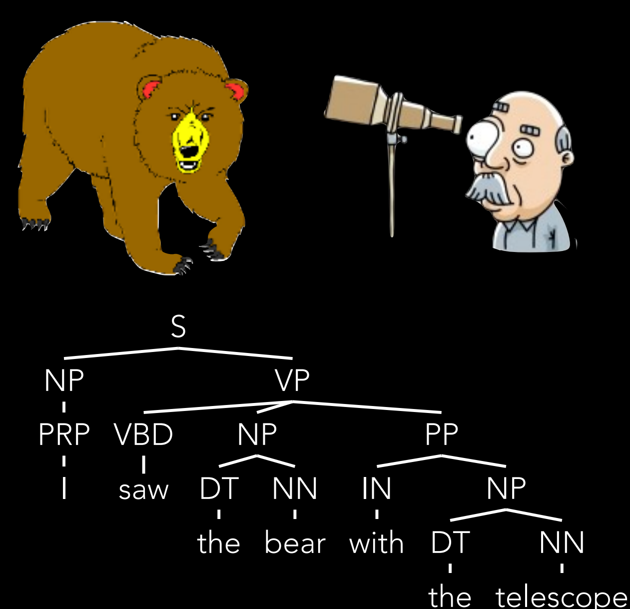


VS

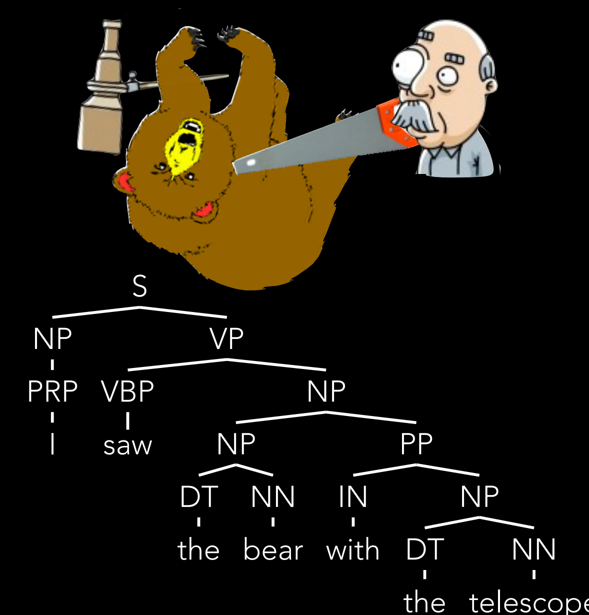


But Which Parse Is Best?

- How does our parser make decisions about which tree to produce?



VS



- We need a probabilistic CFG and a probability model of trees

Probabilistic Context-Free Grammars

S NP
VP NNS
PP
VB
DT
…

Nonterminals
(start)

cats I
mice the
telescope
… eat

Terminals

S → NP VP
VP → VB NP
NP → NNS
NP → DT NNS
NNS → cats
NNS → mice
VB → eat

Rules

Probabilistic Context-Free Grammars

S NP
VP NNS
PP
VB
DT
…

Nonterminals
(start)

cats I
mice the
telescope
eat
…

Terminals

$S \rightarrow NP\ VP$		1.0
$VP \rightarrow VB\ NP$		1.0
$NP \rightarrow NNS$		0.2
$NP \rightarrow DT\ NNS$		0.8
$NNS \rightarrow$ cats		0.6
$NNS \rightarrow$ mice		0.4
$VB \rightarrow$ eat		1.0

Rules

Probabilistic Context-Free Grammars

S NP
VP NNS
PP
DT VB
…

Nonterminals (start)

cats I
mice the
telescope
… eat

Terminals

S → NP VP	1.0
VP → VB NP	1.0
NP → NNS	0.2
NP → DT NNS	0.8
NNS → cats	0.6
NNS → mice	0.4
VB → eat	1.0

Rules

Each rule has a weight w, 0 < w <= 1

Probabilistic Context-Free Grammars

Nonterminals (start)	Terminals	Rules	
S, NP, VP, NNS, PP, VB, DT, …	cats, I, mice, the, telescope, eat, …	$S \rightarrow NP\ VP$	1.0
		$VP \rightarrow VB\ NP$	1.0
		$NP \rightarrow NNS$	0.2
		$NP \rightarrow DT\ NNS$	0.8
		$NNS \rightarrow$ cats	0.6
		$NNS \rightarrow$ mice	0.4
		$VB \rightarrow$ eat	1.0

Each rule has a weight w, 0 < w <= 1

Weights of all rules with the same LHS sum to 1

Probabilistic Context-Free Grammars

S NP
VP NNS
PP
VB
… DT

Nonterminals
(start)

cats I
mice the
telescope
… eat

Terminals

S →	NP VP	1.0
VP →	VB NP	1.0
NP →	NNS	0.2
NP →	DT NNS	0.8
NNS →	cats	0.6
NNS →	mice	0.4
VB →	eat	1.0

Rules

Each rule has a weight w, $0 < w <= 1$

Weights of all rules with the same LHS sum to 1

Probability of tree is product of
probability of rules in its derivation

Probabilistic Context-Free Grammars

S NP
VP NNS
PP
VB
DT
…

Nonterminals
(start)

cats I
mice the
telescope
eat
…

Terminals

S →	NP VP	1.0
VP →	VB NP	1.0
NP →	NNS	0.2
NP →	DT NNS	0.8
NNS →	cats	0.6
NNS →	mice	0.4
VB →	eat	1.0

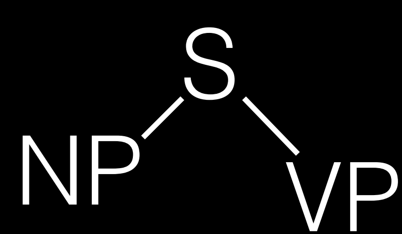
Rules

S

1.0

Probabilistic Context-Free Grammars

Nonterminals (start)	Terminals	Rules
S, NP, VP, NNS, PP, VB, DT, …	cats, I, mice, the, telescope, eat, …	S → NP VP 1.0 VP → VB NP 1.0 NP → NNS 0.2 NP → DT NNS 0.8 NNS → cats 0.6 NNS → mice 0.4 VB → eat 1.0



1.0

Probabilistic Context-Free Grammars

S NP
VP NNS
PP
VB
… DT

Nonterminals
(start)

cats I
mice the
telescope
… eat

Terminals

S → NP VP	1.0
VP → VB NP	1.0
NP → NNS	0.2
NP → DT NNS	0.8
NNS → cats	0.6
NNS → mice	0.4
VB → eat	1.0

Rules

S
NP VP
NNS

0.2

Probabilistic Context-Free Grammars

S NP VP NNS PP VB DT …

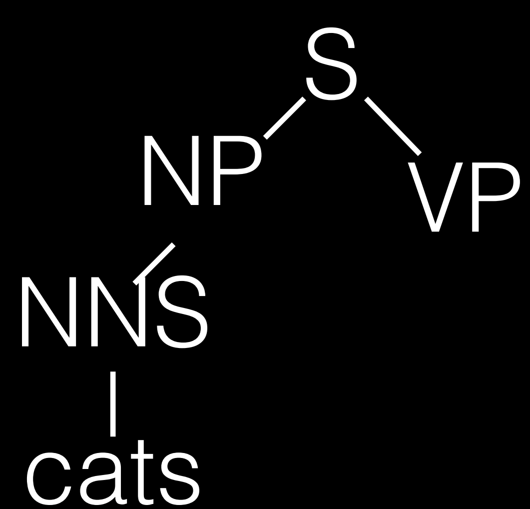
Nonterminals (start)

cats I mice the telescope eat …

Terminals

S → NP VP		1.0
VP → VB NP		1.0
NP → NNS		0.2
NP → DT NNS		0.8
NNS → cats		0.6
NNS → mice		0.4
VB → eat		1.0

Rules



0.12

Probabilistic Context-Free Grammars

S NP
VP NNS
PP
VB
… DT

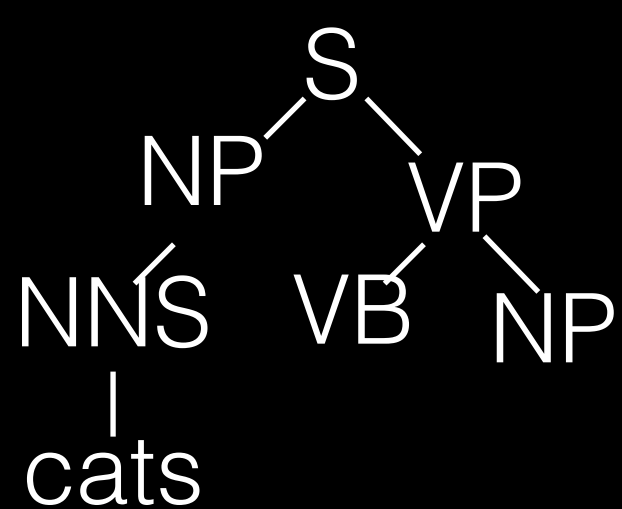
Nonterminals
(start)

cats I
mice the
telescope
… eat

Terminals

S	→ NP VP	1.0
VP	→ VB NP	1.0
NP	→ NNS	0.2
NP	→ DT NNS	0.8
NNS	→ cats	0.6
NNS	→ mice	0.4
VB	→ eat	1.0

Rules



0.12

Probabilistic Context-Free Grammars

S NP
VP NNS
PP
VB
... DT

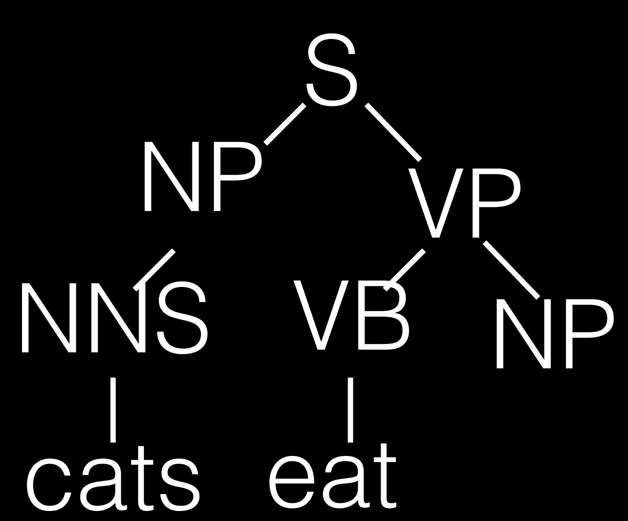
Nonterminals
(start)

cats I
mice the
telescope
... eat

Terminals

S	→ NP VP	1.0
VP	→ VB NP	1.0
NP	→ NNS	0.2
NP	→ DT NNS	0.8
NNS	→ cats	0.6
NNS	→ mice	0.4
VB	→ eat	1.0

Rules



0.12

Probabilistic Context-Free Grammars

S NP
VP NNS
PP
VB
… DT

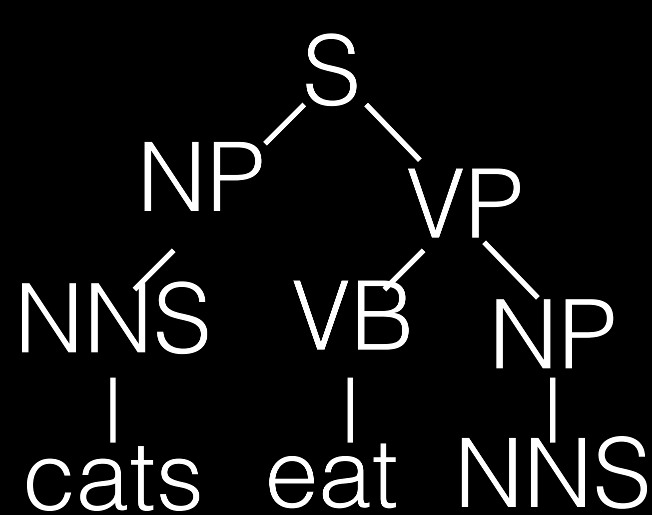
Nonterminals
(start)

cats I
mice the
telescope
… eat

Terminals

S → NP VP	1.0
VP → VB NP	1.0
NP → NNS	0.2
NP → DT NNS	0.8
NNS → cats	0.6
NNS → mice	0.4
VB → eat	1.0

Rules



0.024

Probabilistic Context-Free Grammars

S NP VP NNS PP VB DT …

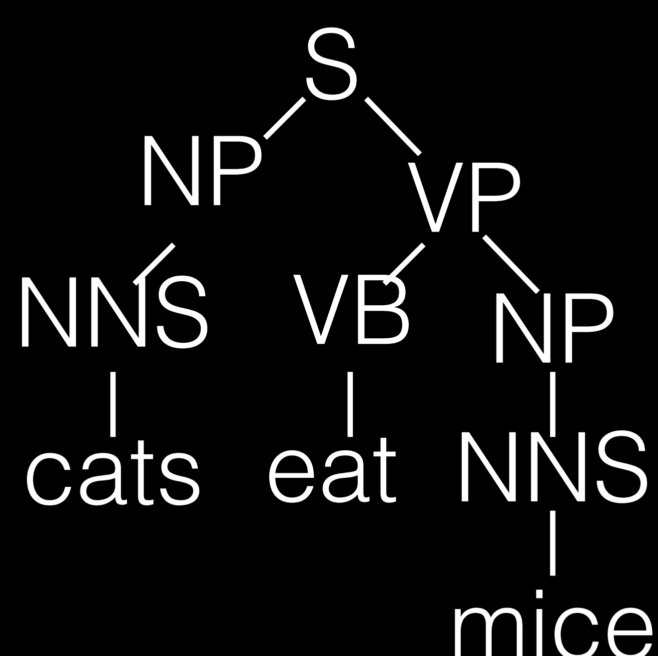
Nonterminals (start)

cats I mice the telescope eat …

Terminals

S → NP VP		1.0
VP → VB NP		1.0
NP → NNS		0.2
NP → DT NNS		0.8
NNS → cats		0.6
NNS → mice		0.4
VB → eat		1.0

Rules



0.0096

Viterbi (one-best) CKY for PCFG

Viterbi (one-best) CKY for PCFG

- Before:
  - we built states [X, i, j] denoting coverage of span from i to j with symbol X
  - we stored all {X->YZ, k} backpointer

Viterbi (one-best) CKY for PCFG

- Before:
    - we built states [X, i, j] denoting coverage of span from i to j with symbol X
    - we stored all {X->YZ, k} backpointer
- New:
    - when deriving via {R, k}, set best([X, i, j]) = p(R) * best([Y, i, k]) * best([Z, k, j]) if this is greater than current value for best([X, i, j])
    - only keep backpointer if best is updated
    - chart contains only the best derivation at end of algorithm

time	flies	like	an	arrow
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	[1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

$S \rightarrow NP\ VP$ 1.0

$NP \rightarrow DT\ NN$ 0.3

$NP \rightarrow time$ 0.05 $|$ $fruit$ 0.05

$NP \rightarrow NN\ NNS$ 0.6

$VP \rightarrow VBP\ NP$ 0.7

$VP \rightarrow flies$ 0.1

$VP \rightarrow VP\ PP$ 0.2

$PP \rightarrow IN\ NP$ 1.0

$DT \rightarrow a$ 0.5 $|$ an 0.5

$NN \rightarrow time$ 0.25 $|$ $fruit$ 0.25 $|$ $arrow$ 0.25 $|$ $banana$ 0.25

$NNS \rightarrow flies$ 1.0

$VBP \rightarrow like$ 1.0

$IN \rightarrow like$ 1.0

time	flies	like	an	arrow
NP 0.05 NN 0.25 [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	[1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

S → NP VP 1.0
NP → DT NN 0.3
NP → $\boxed{\text{time}^{0.05}}$ | fruit 0.05
NP → NN NNS 0.6
VP → VBP NP 0.7
VP → flies 0.1
VP → VP PP 0.2
PP → IN NP 1.0
DT → a 0.5 | an 0.5
NN → $\boxed{\text{time}^{0.25}}$ | fruit 0.25 | arrow 0.25 | banana 0.25
NNS → flies 1.0
VBP → like 1.0
IN → like 1.0

best() for bottom of the chart
is simply the rule probability

time	flies	like	an	arrow
NP 0.05 NN 0.25 [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	VP 0.1 NNS 1.0 [1,2]	[1,3]	[1,4]	[1,5]
		[2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

- $S \rightarrow NP\ VP$ 1.0
- $NP \rightarrow DT\ NN$ 0.3
- $NP \rightarrow$ time (0.05) | fruit (0.05)
- $NP \rightarrow NN\ NNS$ 0.6
- $VP \rightarrow VBP\ NP$ 0.7
- $\boxed{VP \rightarrow \text{flies}}$ 0.1
- $VP \rightarrow VP\ PP$ 0.2
- $PP \rightarrow IN\ NP$ 1.0
- $DT \rightarrow$ a (0.5) | an (0.5)
- $NN \rightarrow$ time (0.25) | fruit (0.25) | arrow (0.25) | banana (0.25)
- $\boxed{NNS \rightarrow \text{flies}}$ 1.0
- $VBP \rightarrow$ like 1.0
- $IN \rightarrow$ like 1.0

best() for bottom of the chart
is simply the rule probability

time	flies	like	an	arrow
NP $^{0.05}$ NN $^{0.25}$ [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	VP $^{0.1}$ NNS $^{1.0}$ [1,2]	[1,3]	[1,4]	[1,5]
		IN $^{1.0}$ VBP $^{1.0}$ [2,3]	[2,4]	[2,5]
			[3,4]	[3,5]
				[4,5]

S → NP VP $^{1.0}$

NP → DT NN $^{0.3}$

NP → time $^{0.05}$ | fruit $^{0.05}$

NP → NN NNS $^{0.6}$

VP → VBP NP $^{0.7}$

VP → flies $^{0.1}$

VP → VP PP $^{0.2}$

PP → IN NP $^{1.0}$

DT → a $^{0.5}$ | an $^{0.5}$

NN → time $^{0.25}$ | fruit $^{0.25}$ | arrow $^{0.25}$ | banana $^{0.25}$

NNS → flies $^{1.0}$

VBP → like $^{1.0}$

IN → like $^{1.0}$

best() for bottom of the chart is simply the rule probability

time	flies	like	an	arrow
NP $^{0.05}$ NN $^{0.25}$ [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	VP $^{0.1}$ NNS $^{1.0}$ [1,2]	[1,3]	[1,4]	[1,5]
		IN $^{1.0}$ VBP $^{1.0}$ [2,3]	[2,4]	[2,5]
			DT $^{0.5}$ [3,4]	[3,5]
				[4,5]

$S \rightarrow NP\ VP^{1.0}$

$NP \rightarrow DT\ NN^{0.3}$

$NP \rightarrow time^{0.05} \mid fruit^{0.05}$

$NP \rightarrow NN\ NNS^{0.6}$

$VP \rightarrow VBP\ NP^{0.7}$

$VP \rightarrow flies^{0.1}$

$VP \rightarrow VP\ PP^{0.2}$

$PP \rightarrow IN\ NP^{1.0}$

$DT \rightarrow a^{0.5} \mid \boxed{an}^{0.5}$

$NN \rightarrow time^{0.25} \mid fruit^{0.25} \mid arrow^{0.25} \mid banana^{0.25}$

$NNS \rightarrow flies^{1.0}$

$VBP \rightarrow like^{1.0}$

$IN \rightarrow like^{1.0}$

best() for bottom of the chart is simply the rule probability

time	flies	like	an	arrow
NP 0.05 NN 0.25 [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	VP 0.1 NNS 1.0 [1,2]	[1,3]	[1,4]	[1,5]
		IN 1.0 VBP 1.0 [2,3]	[2,4]	[2,5]
			DT 0.5 [3,4]	[3,5]
				NN 0.25 [4,5]

$S \rightarrow NP\ VP$ 1.0

$NP \rightarrow DT\ NN$ 0.3

$NP \rightarrow$ time 0.05 | fruit 0.05

$NP \rightarrow NN\ NNS$ 0.6

$VP \rightarrow VBP\ NP$ 0.7

$VP \rightarrow$ flies 0.1

$VP \rightarrow VP\ PP$ 0.2

$PP \rightarrow IN\ NP$ 1.0

$DT \rightarrow$ a 0.5 | an 0.5

$NN \rightarrow$ time 0.25 | fruit 0.25 | arrow 0.25 | banana 0.25

$NNS \rightarrow$ flies 1.0

$VBP \rightarrow$ like 1.0

$IN \rightarrow$ like 1.0

best() for bottom of the chart is simply the rule probability

	time	flies	like	an	arrow
	NP $^{0.05}$ NN $^{0.25}$ [0,1]	[0,2]	[0,3]	[0,4]	[0,5]
		VP $^{0.1}$ NNS $^{1.0}$ [1,2]	[1,3]	[1,4]	[1,5]
			IN $^{1.0}$ VBP $^{1.0}$ [2,3]	[2,4]	[2,5]
				DT $^{0.5}$ [3,4]	[3,5]
					NN $^{0.25}$ [4,5]

S → NP VP $^{1.0}$

NP → DT NN $^{0.3}$

NP → time $^{0.05}$ | fruit $^{0.05}$

NP → NN NNS $^{0.6}$

VP → VBP NP $^{0.7}$

VP → flies $^{0.1}$

VP → VP PP $^{0.2}$

PP → IN NP $^{1.0}$

DT → a $^{0.5}$ | an $^{0.5}$

NN → time $^{0.25}$ | fruit $^{0.25}$ | arrow $^{0.25}$ | banana $^{0.25}$

NNS → flies $^{1.0}$

VBP → like $^{1.0}$

IN → like $^{1.0}$

Now multiply rule by best() of descendants

time	flies	like	an	arrow
NP 0.05 NN 0.25 [0,1]	S 0.005 NP 0.15 [0,2]	[0,3]	[0,4]	[0,5]
	VP 0.1 NNS 1.0 [1,2]	[1,3]	[1,4]	[1,5]
		IN 1.0 VBP 1.0 [2,3]	[2,4]	[2,5]
			DT 0.5 [3,4]	[3,5]
				NN 0.25 [4,5]

$\boxed{S \rightarrow NP\ VP}$ 1.0
NP → DT NN 0.3
NP → time 0.05 | fruit 0.05
$\boxed{NP \rightarrow NN\ NNS}$ 0.6
VP → VBP NP 0.7
VP → flies 0.1
VP → VP PP 0.2
PP → IN NP 1.0
DT → a 0.5 | an 0.5
NN → time 0.25 | fruit 0.25 | arrow 0.25 | banana 0.25
NNS → flies 1.0
VBP → like 1.0
IN → like 1.0

$$1.0 * .05 * .1 = .005$$
$$.6 * .25 * 1.0 = .15$$

	time	flies	like	an	arrow
	NP 0.05 NN 0.25 [0,1]	S 0.005 NP 0.15 [0,2]	[0,3]	[0,4]	0.0000375 S [0,5]
		VP 0.1 NNS 1.0 [1,2]	[1,3]	[1,4]	0.00075 VP [1,5]
			IN 1.0 VBP 1.0 [2,3]	[2,4]	PP 0.0375 VP 0.02625 [2,5]
				DT 0.5 [3,4]	0.0375 NP [3,5]
					NN 0.25 [4,5]

$S \rightarrow NP\ VP$ 1.0

$NP \rightarrow DT\ NN$ 0.3

$NP \rightarrow$ time 0.05 | fruit 0.05

$NP \rightarrow NN\ NNS$ 0.6

$VP \rightarrow VBP\ NP$ 0.7

$VP \rightarrow$ flies 0.1

$VP \rightarrow VP\ PP$ 0.2

$PP \rightarrow IN\ NP$ 1.0

$DT \rightarrow$ a 0.5 | an 0.5

$NN \rightarrow$ time 0.25 | fruit 0.25 | arrow 0.25 | banana 0.25

$NNS \rightarrow$ flies 1.0

$VBP \rightarrow$ like 1.0

$IN \rightarrow$ like 1.0

best(S) = 1*.05*.00075 = .0000375

time	flies	like	an	arrow
NP 0.05 NN 0.25 [0,1]	S 0.005 NP 0.15 [0,2]	[0,3]	[0,4]	0.0039375 S [0,5]
	VP 0.1 NNS 1.0 [1,2]	[1,3]	[1,4]	0.00075 VP [1,5]
		IN 1.0 VBP 1.0 [2,3]	[2,4]	PP 0.0375 VP 0.02625 [2,5]
			DT 0.5 [3,4]	0.0375 NP [3,5]
				NN 0.25 [4,5]

$\boxed{S \rightarrow NP\ VP}$ 1.0

NP → DT NN 0.3

NP → time 0.05 | fruit 0.05

NP → NN NNS 0.6

VP → VBP NP 0.7

VP → flies 0.1

VP → VP PP 0.2

PP → IN NP 1.0

DT → a 0.5 | an 0.5

NN → time 0.25 | fruit 0.25 | arrow 0.25 | banana 0.25

NNS → flies 1.0

VBP → like 1.0

IN → like 1.0

can also derive as

1*.15*.02625 = .0039375

so update best and change backpointer

time	flies	like	an	arrow
NP 0.05 NN 0.25 [0,1]	S 0.005 NP 0.15 [0,2]	[0,3]	[0,4]	S 0.0039375 [0,5]
	VP 0.1 NNS 1.0 [1,2]	[1,3]	[1,4]	VP 0.00075 [1,5]
		IN 1.0 VBP 1.0 [2,3]	[2,4]	PP 0.0375 VP 0.02625 [2,5]
			DT 0.5 [3,4]	NP 0.0375 [3,5]
				NN 0.25 [4,5]

S → NP VP 1.0

NP → DT NN 0.3

NP → time 0.05 | fruit 0.05

NP → NN NNS 0.6

VP → VBP NP 0.7

VP → flies 0.1

VP → VP PP 0.2

PP → IN NP 1.0

DT → a 0.5 | an 0.5

NN → time 0.25 | fruit 0.25 | arrow 0.25 | banana 0.25

NNS → flies 1.0

VBP → like 1.0

IN → like 1.0

If we wanted the probability of ALL parses of this sentence, sum multiple derivations instead of max (= .0039375+ .0000375)

How to get the weights

How to get the weights

- We discovered our rule set by reading training trees.

How to get the weights

- We discovered our rule set by reading training trees.
- We can do the same to learn weights

How to get the weights

- We discovered our rule set by reading training trees.
- We can do the same to learn weights
- Maximum likelihood estimate = fancy name for count and divide

How to get the weights

- We discovered our rule set by reading training trees.

- We can do the same to learn weights

- Maximum likelihood estimate = fancy name for count and divide

- If we see NP 1000 times in training, and we see NP -> DT NN 150 times, then p(NP -> DT NN) = 150/1000 = .15

Problems with PCFGs

Problems with PCFGs

- Using the approach just described ("vanilla PCFG"), a CKY parser scores about 73% F1. State of the art parsers are in the mid-90s

Problems with PCFGs

- Using the approach just described (“vanilla PCFG”), a CKY parser scores about 73% F1. State of the art parsers are in the mid-90s
- Why so bad?

Problems with PCFGs

- Using the approach just described ("vanilla PCFG"), a CKY parser scores about 73% F1. State of the art parsers are in the mid-90s

- Why so bad?

  - Rules are too specific

Problems with PCFGs

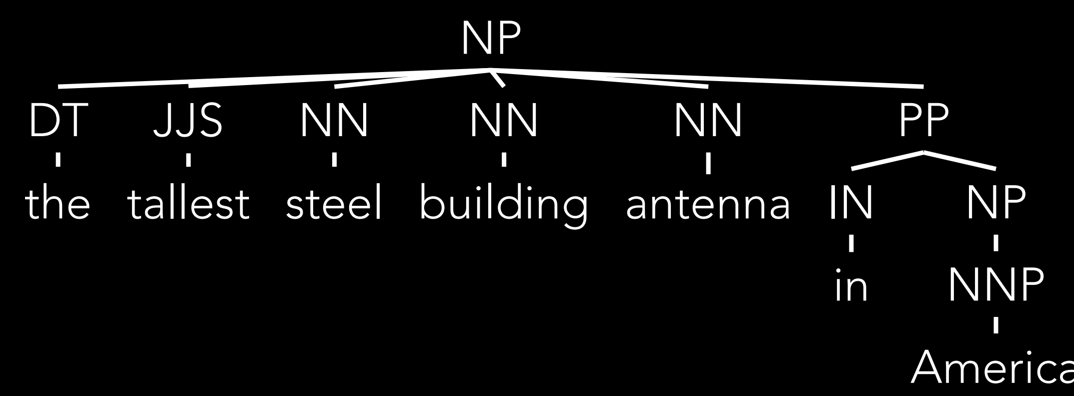
- Using the approach just described ("vanilla PCFG"), a CKY parser scores about 73% F1. State of the art parsers are in the mid-90s

- Why so bad?

  - Rules are too specific

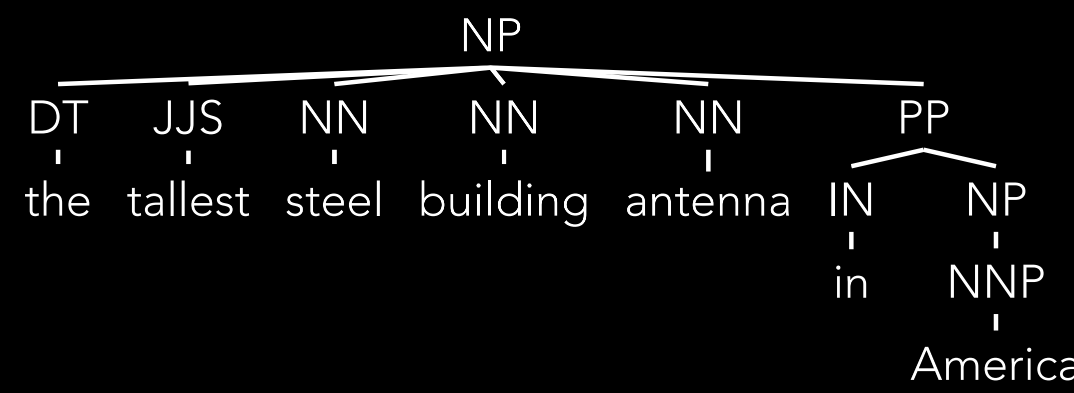
  - Rules are not specific enough

Too Specific



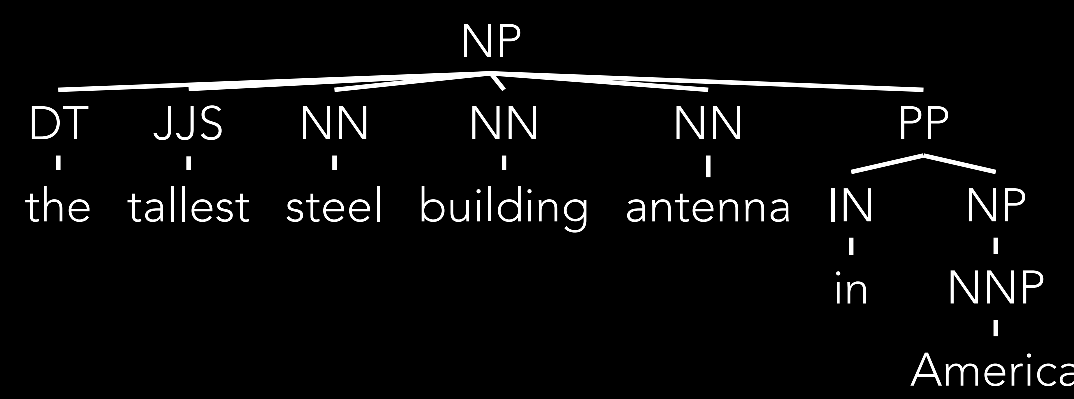
Too Specific

If we see this tree:



Too Specific

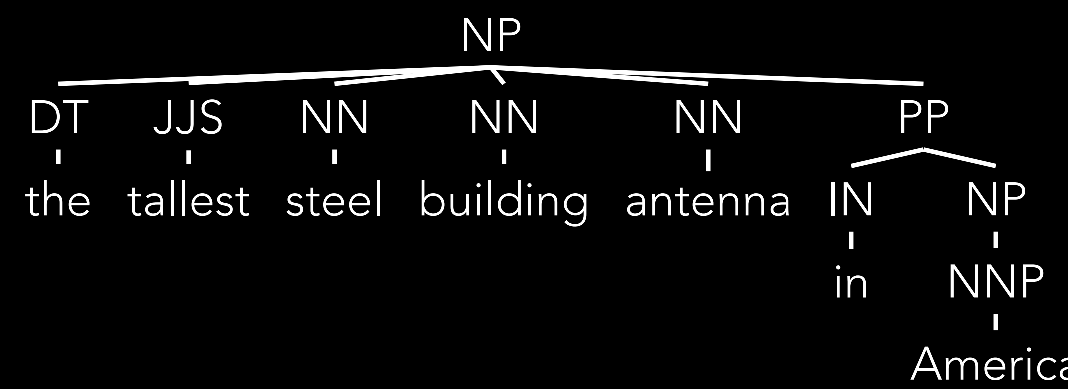
If we see this tree:



We get this rule: NP -> DT JJS NN NN NN PP

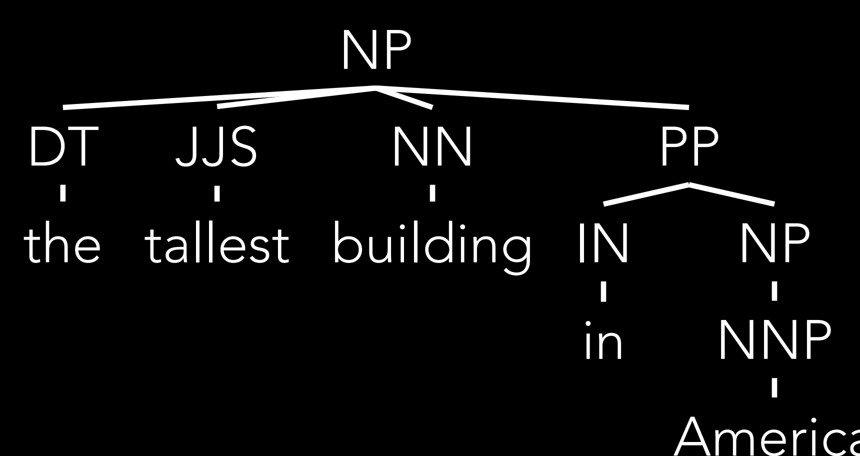
Too Specific

If we see this tree:



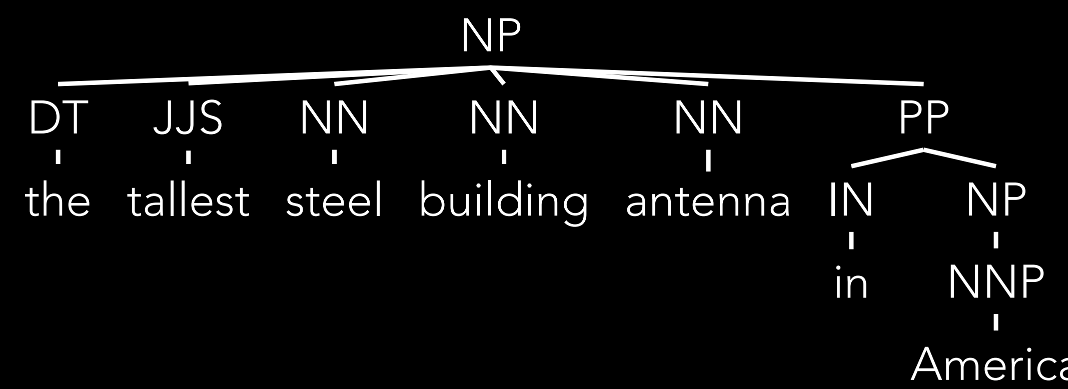
We get this rule: NP -> DT JJS NN NN NN PP

But we can't parse this tree:



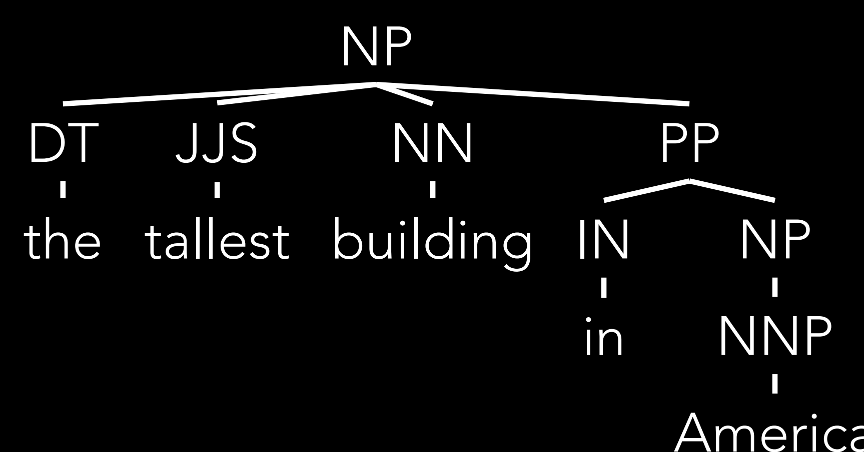
Too Specific

If we see this tree:



We get this rule: NP -> DT JJS NN NN NN PP

But we can't parse this tree:



without this rule: NP -> DT JJS NN PP

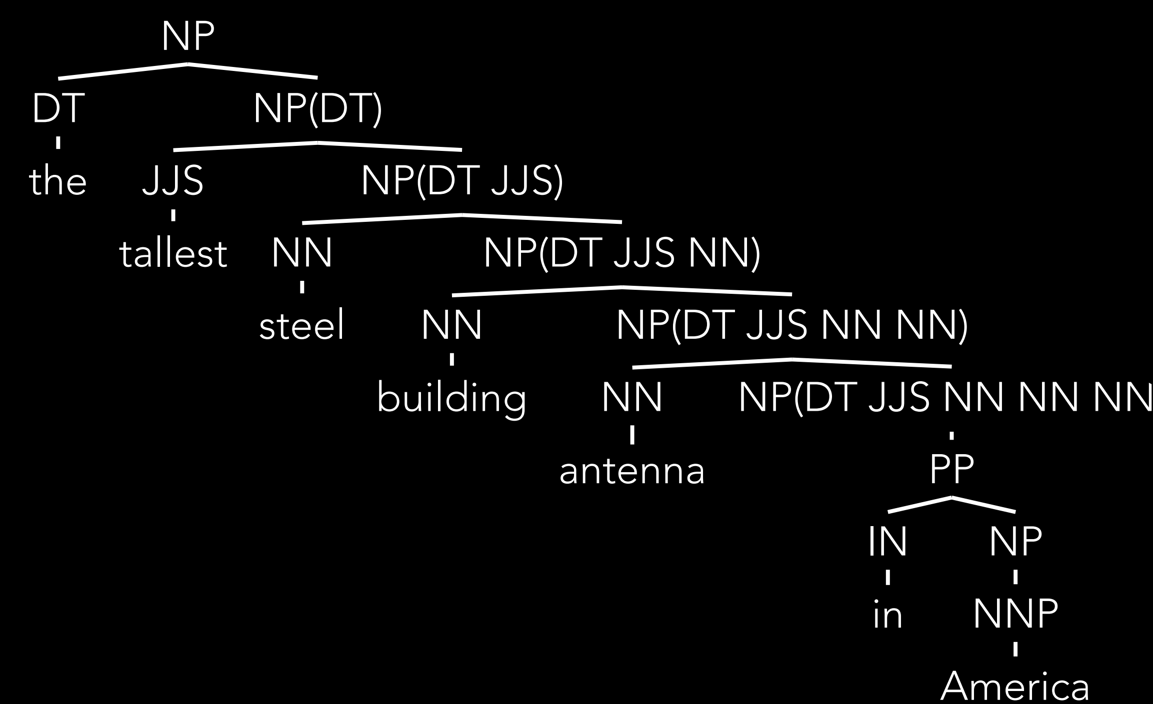
Too Specific

Too Specific

Recall that for
CKY purposes
we binarized

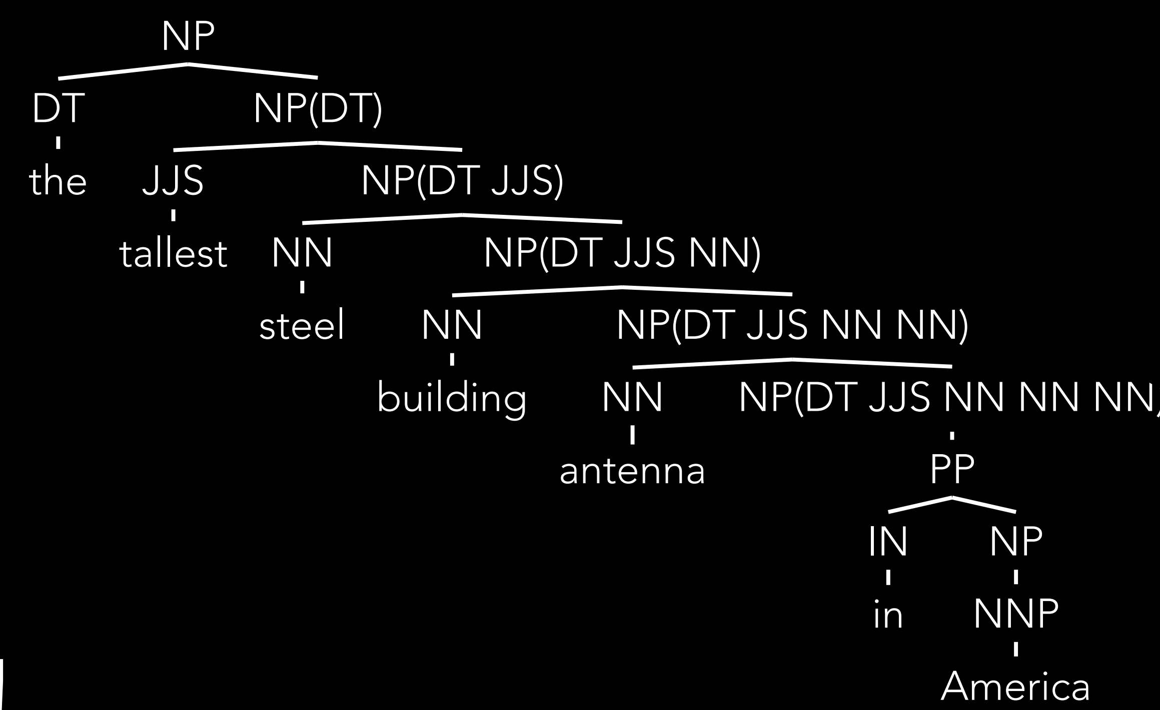
Too Specific

Recall that for CKY purposes we binarized



Too Specific

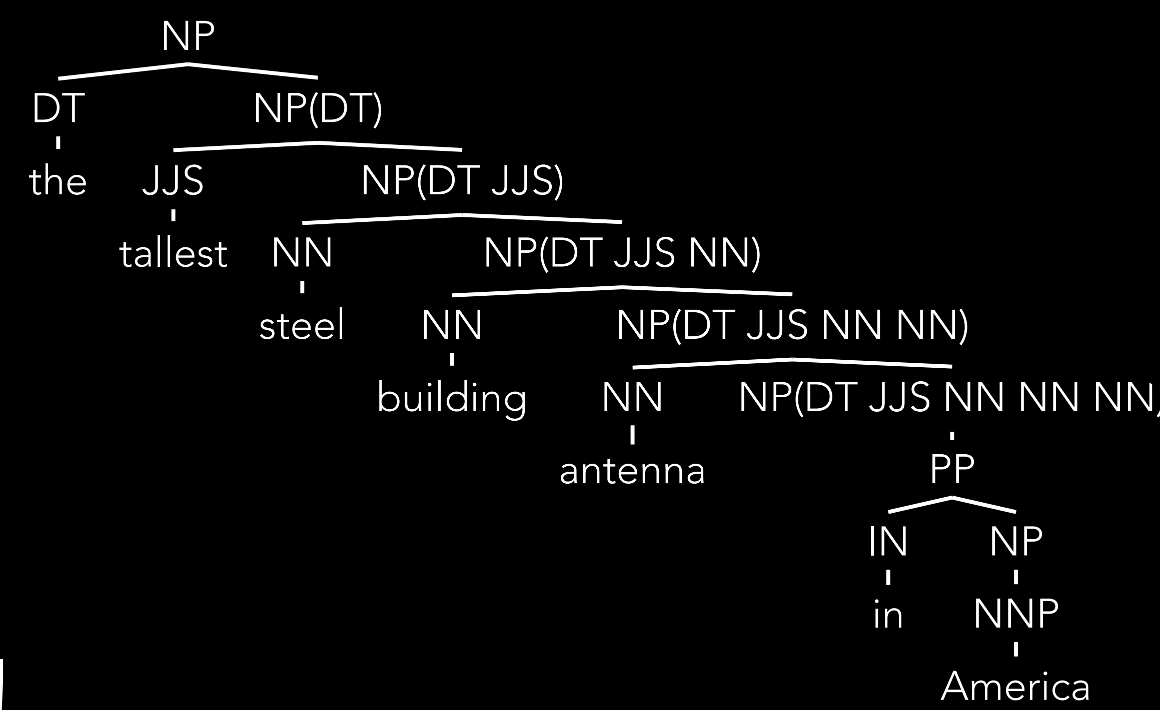
Recall that for CKY purposes we binarized



That still doesn't help!

Too Specific

Recall that for CKY purposes we binarized

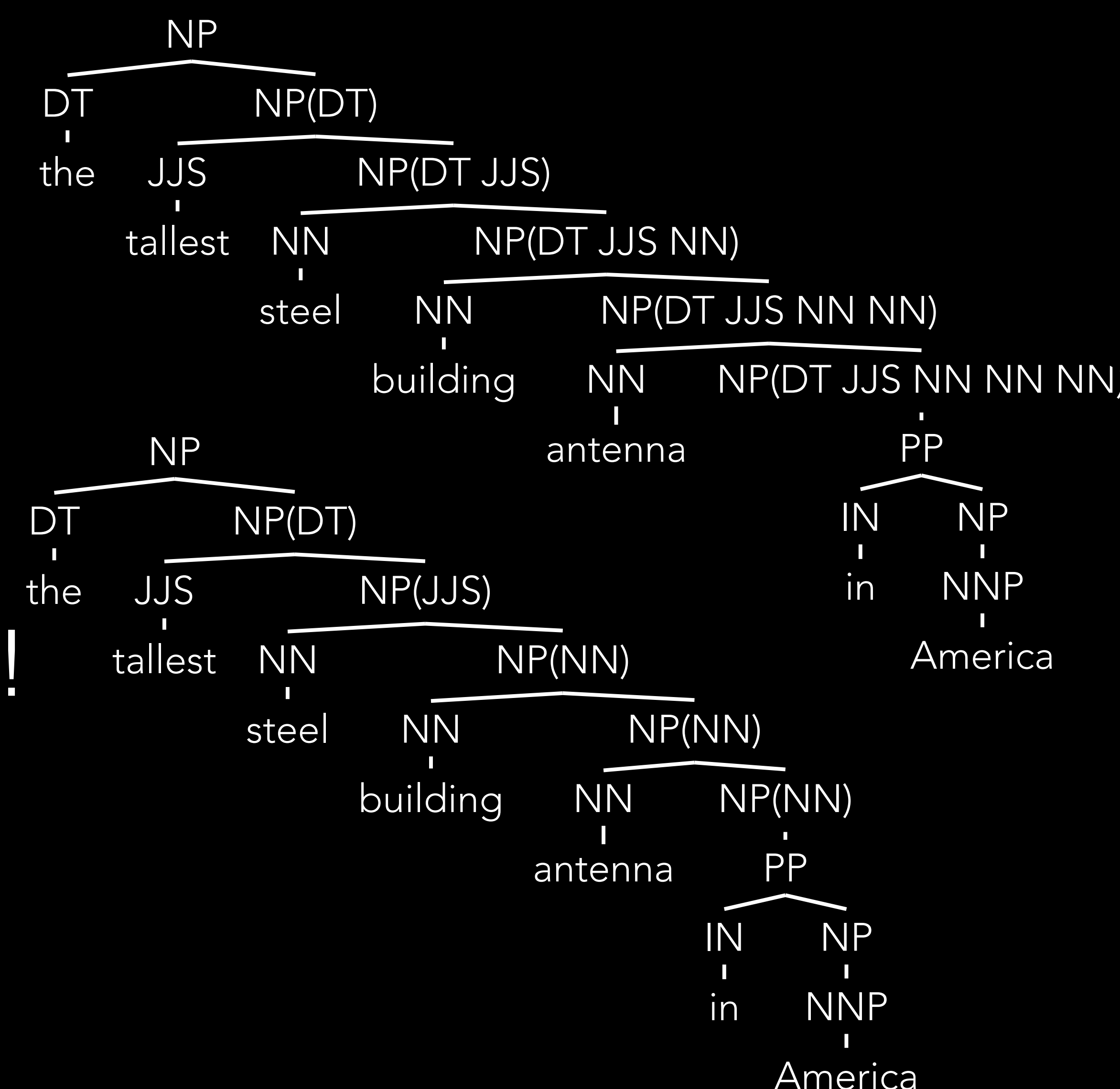


That still doesn't help!

What if we only remembered one sibling?

Too Specific

Recall that for CKY purposes we binarized



That still doesn't help!

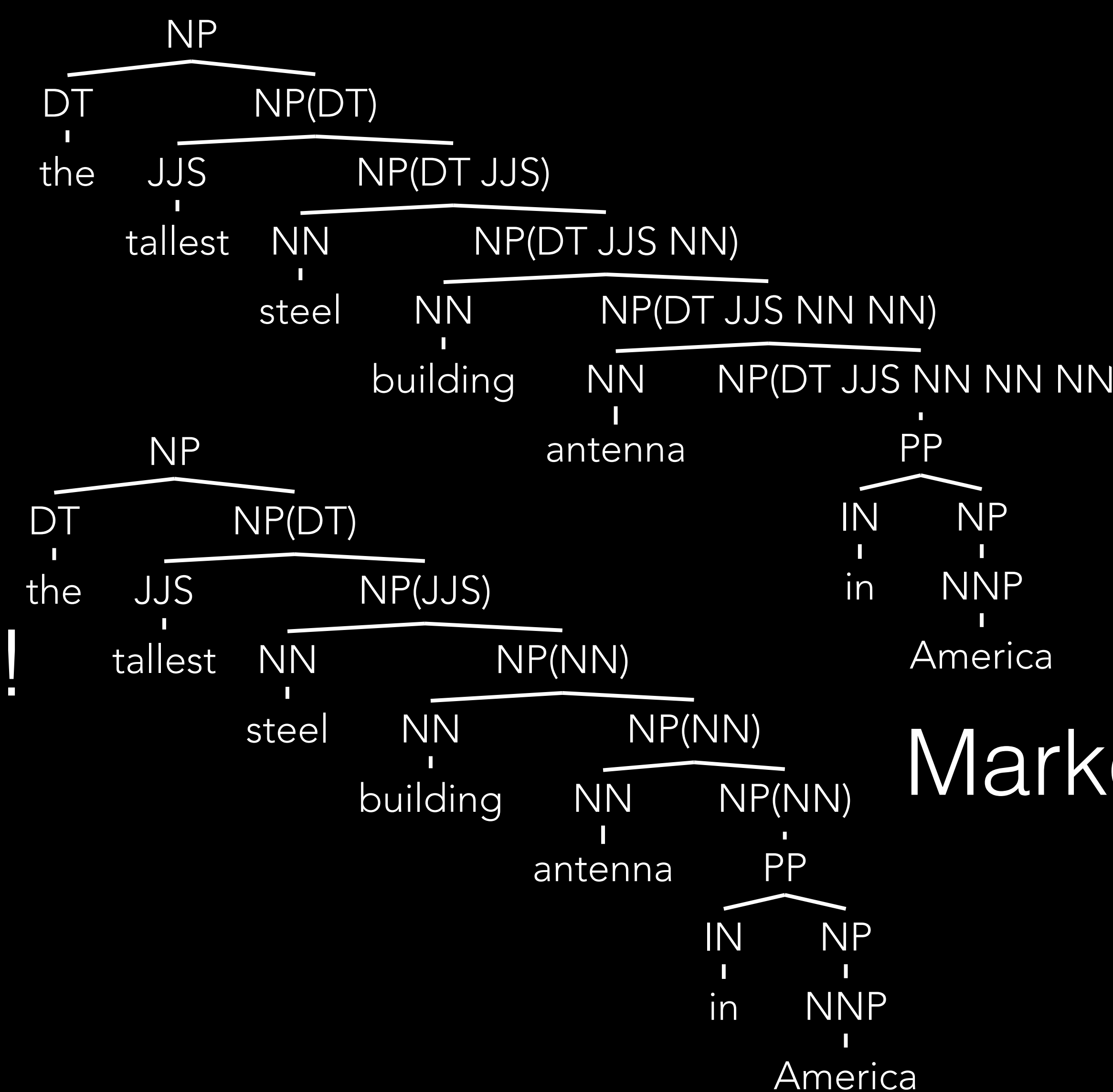
What if we only remembered one sibling?

Too Specific

Recall that for CKY purposes we binarized

That still doesn't help!

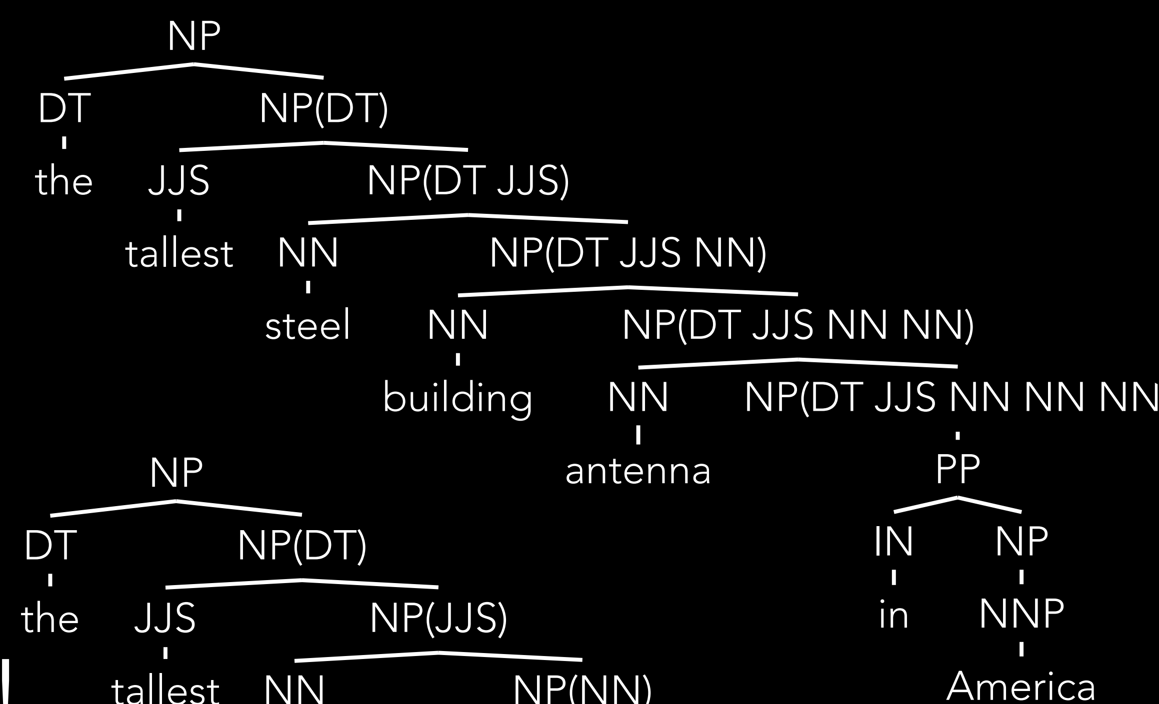
What if we only remembered one sibling?



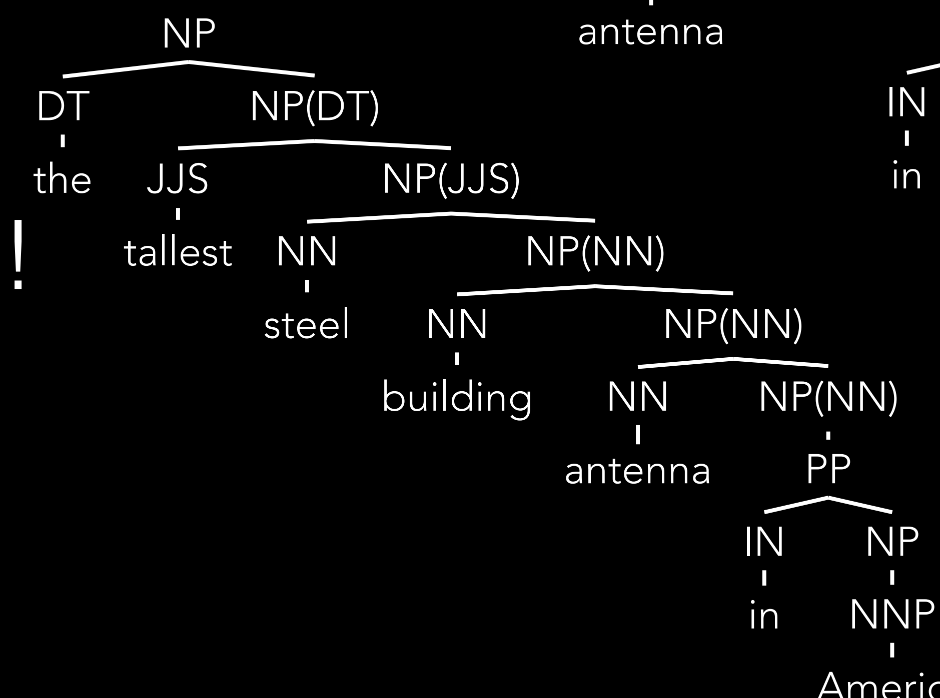
Markovization!

Too Specific

Recall that for CKY purposes we binarized



That still doesn't help!



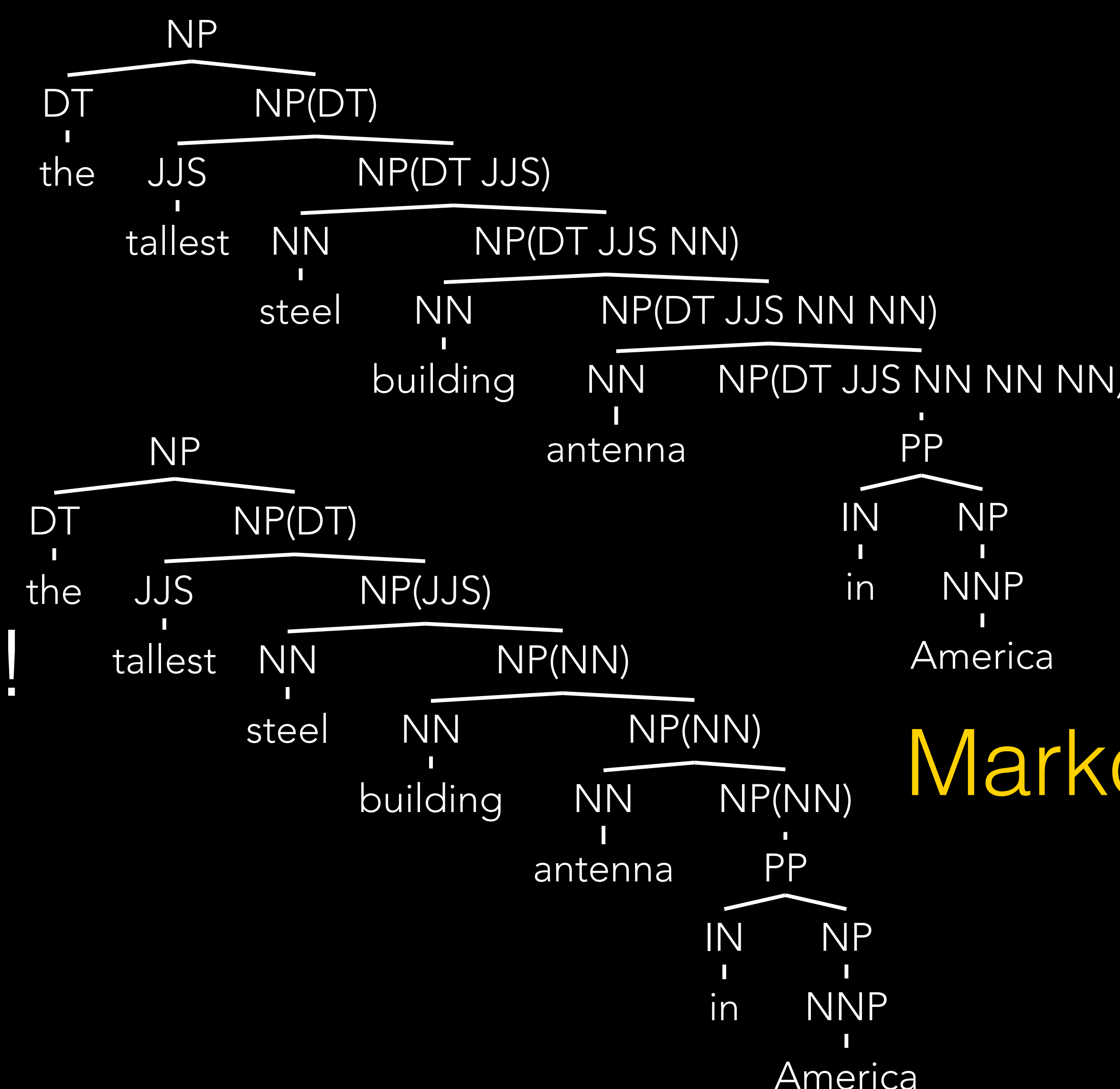
Markovization!

What if we only remembered one sibling?

Now we have the rules we need

Too Specific

Recall that for CKY purposes we binarized



That still doesn't help!

What if we only remembered one sibling?

Markovization!

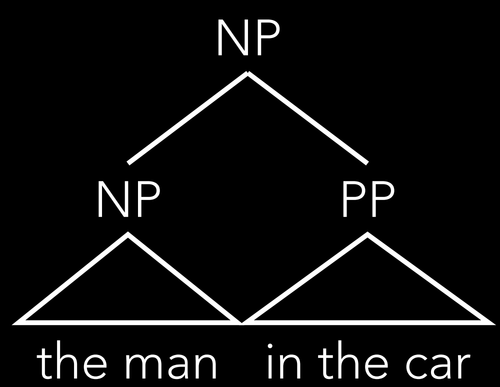
Now we have the rules we need

Can experiment with keeping more or less context

Not Specific Enough (I)

If we saw

Not Specific Enough (I)

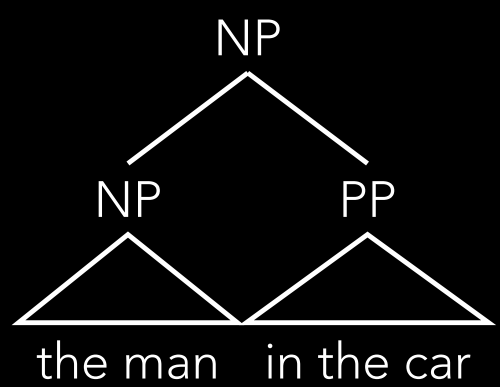


If we saw

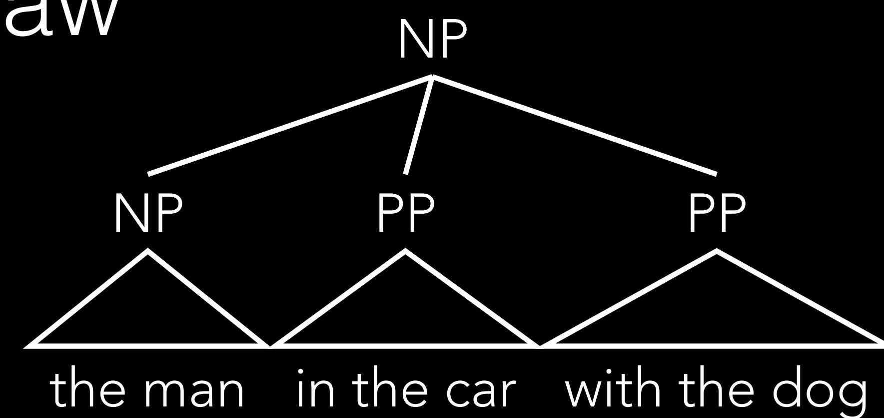
90
times

Not Specific Enough (I)

If we saw



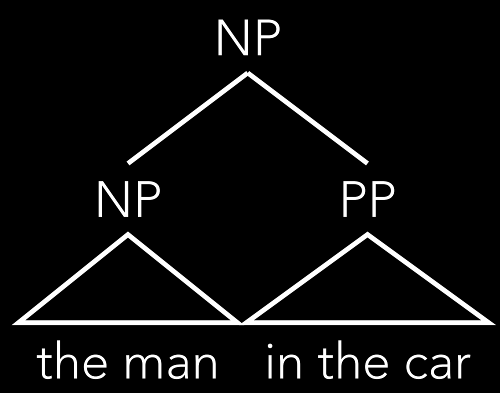
90 times



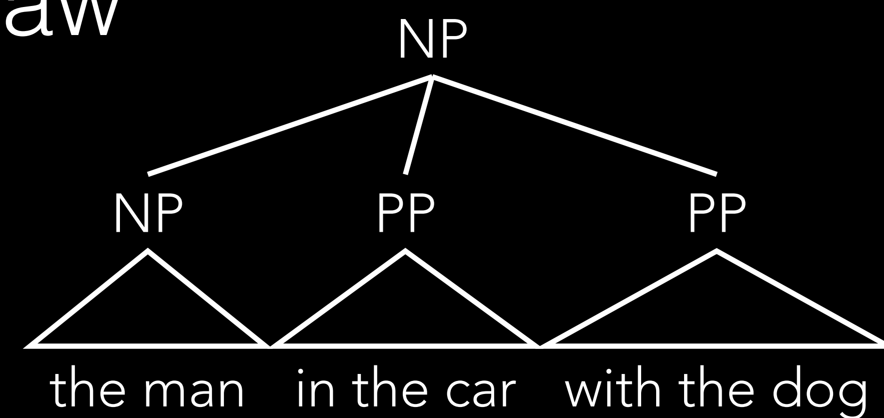
10 times

Not Specific Enough (I)

If we saw



90 times

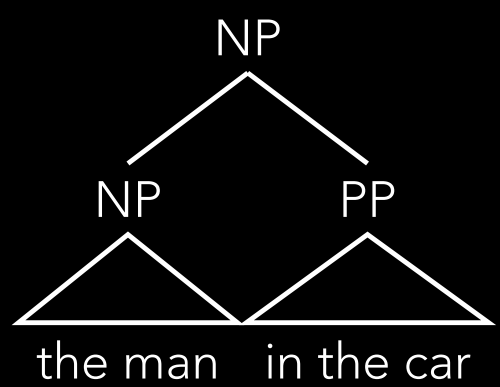


10 times

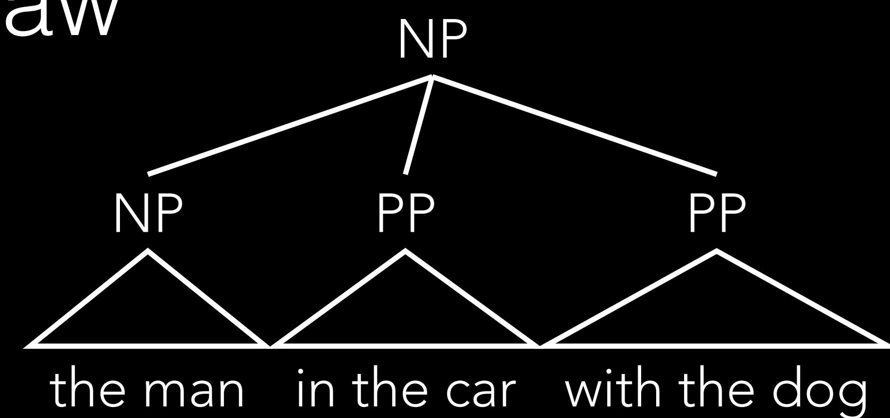
p(NP -> NP PP)
= 90/200 = .45

Not Specific Enough (I)

If we saw



90 times



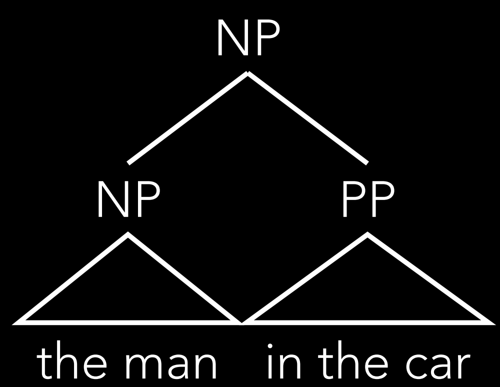
10 times

p(NP -> NP PP)
= 90/200 = .45

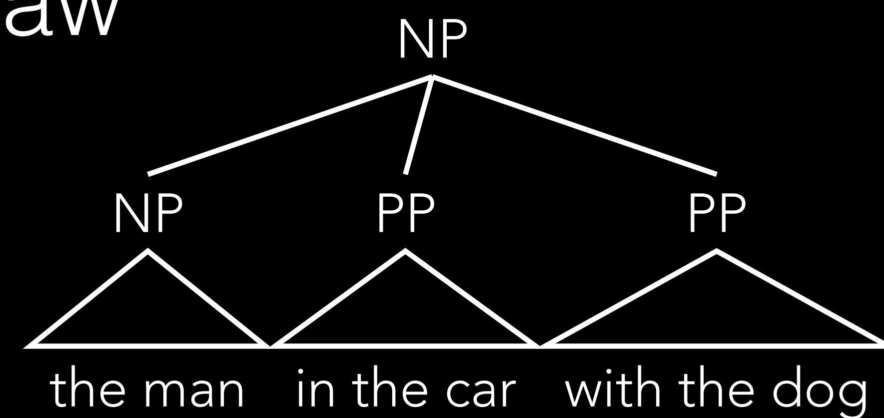
p(NP -> NP PP PP)
= 10/200 = .05

Not Specific Enough (I)

If we saw



90 times



10 times

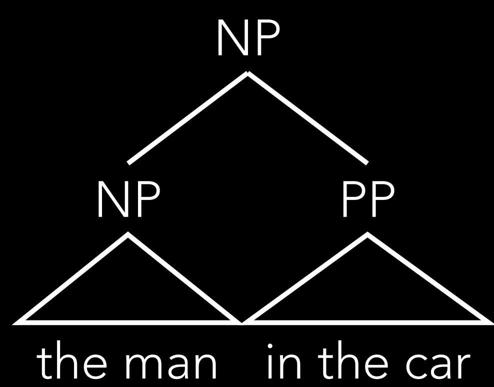
p(NP -> NP PP)
= 90/200 = .45

p(NP -> NP PP PP)
= 10/200 = .05

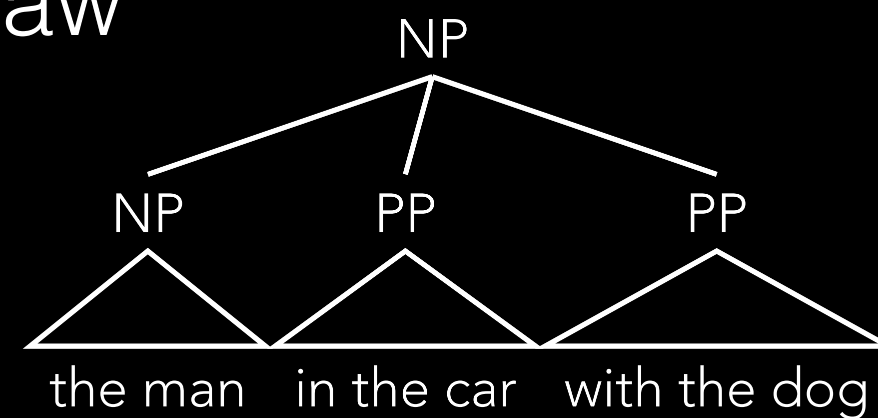
How would we score parses of “the man in the car with the dog”?

Not Specific Enough (I)

If we saw



90 times

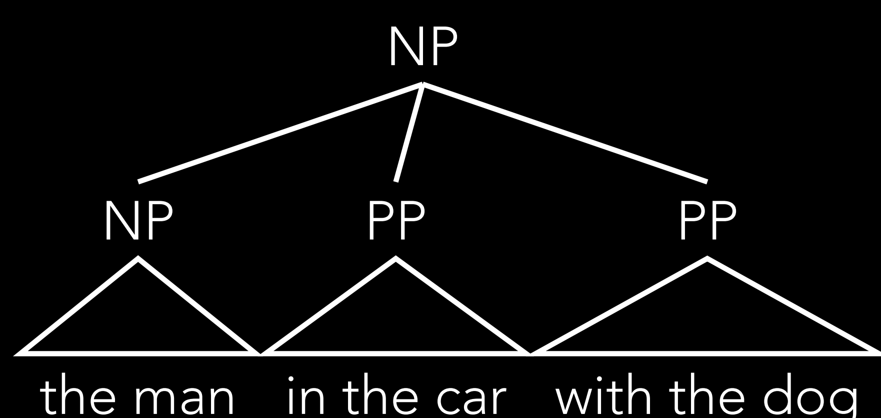


10 times

p(NP -> NP PP)
= 90/200 = .45

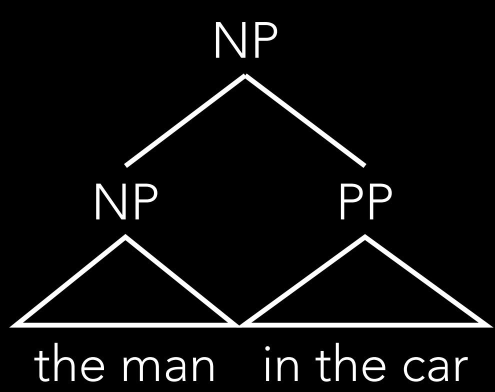
p(NP -> NP PP PP)
= 10/200 = .05

How would we score parses of “the man in the car with the dog”?

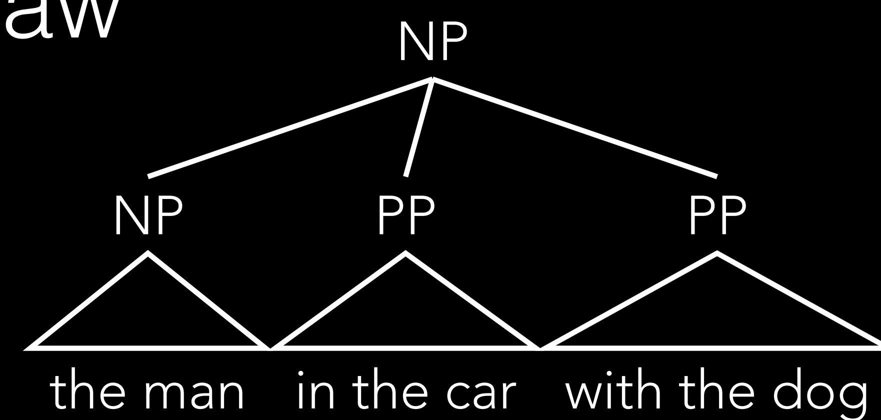


Not Specific Enough (I)

If we saw



90 times

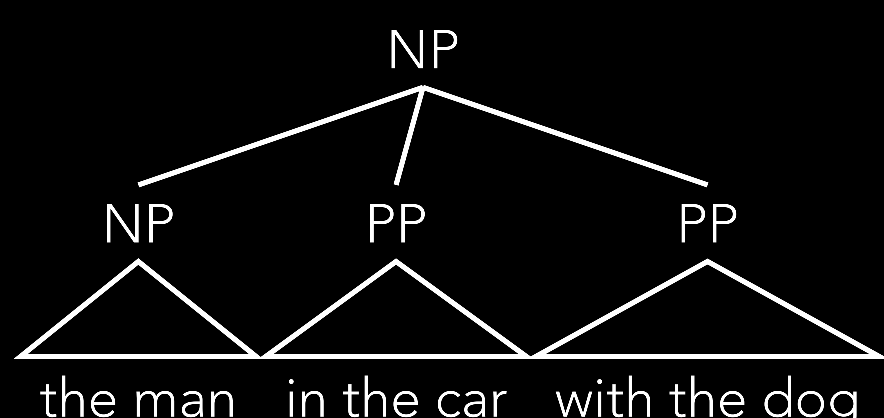


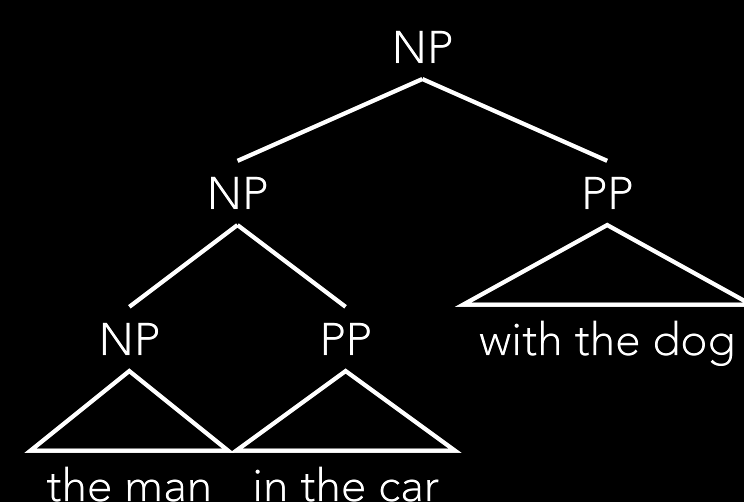
10 times

p(NP -> NP PP)
= 90/200 = .45

p(NP -> NP PP PP)
= 10/200 = .05

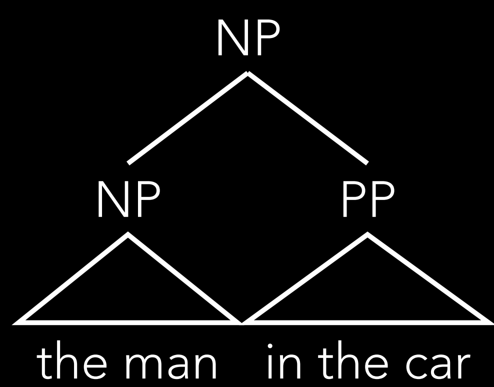
How would we score parses of “the man in the car with the dog”?



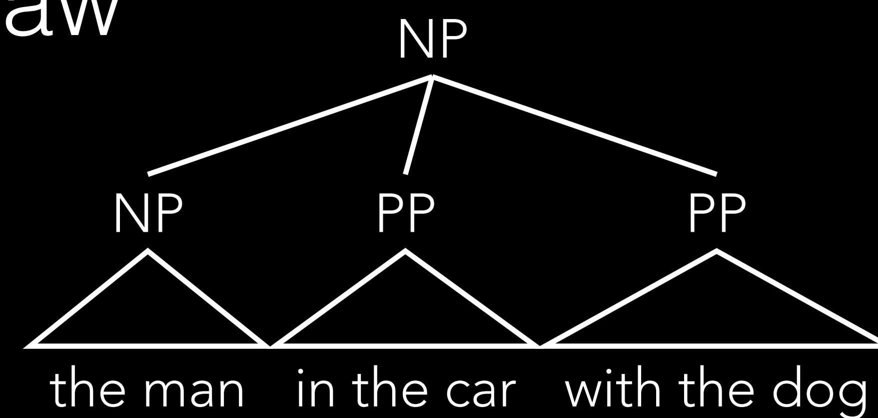


Not Specific Enough (I)

If we saw



90 times

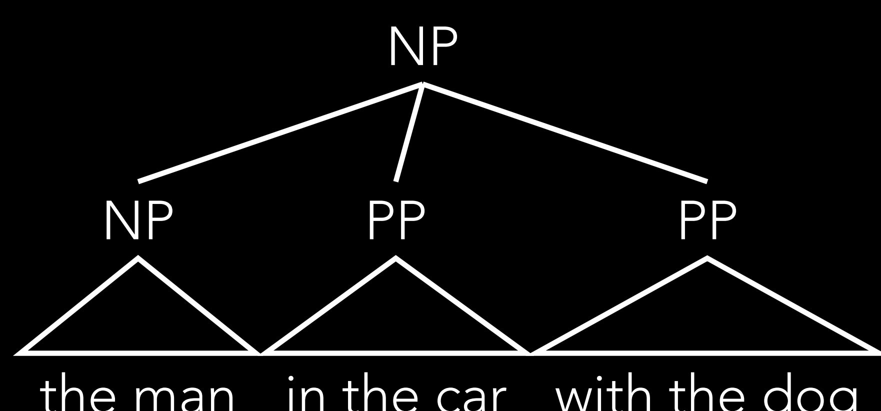


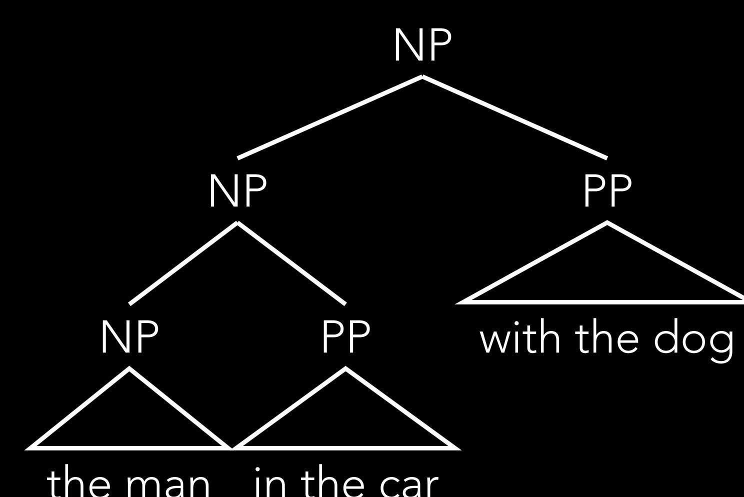
10 times

p(NP -> NP PP) = 90/200 = .45

p(NP -> NP PP PP) = 10/200 = .05

How would we score parses of “the man in the car with the dog”?

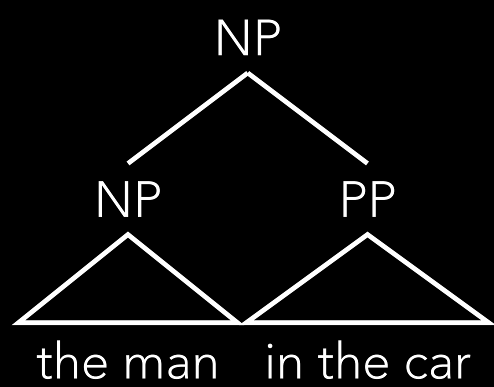




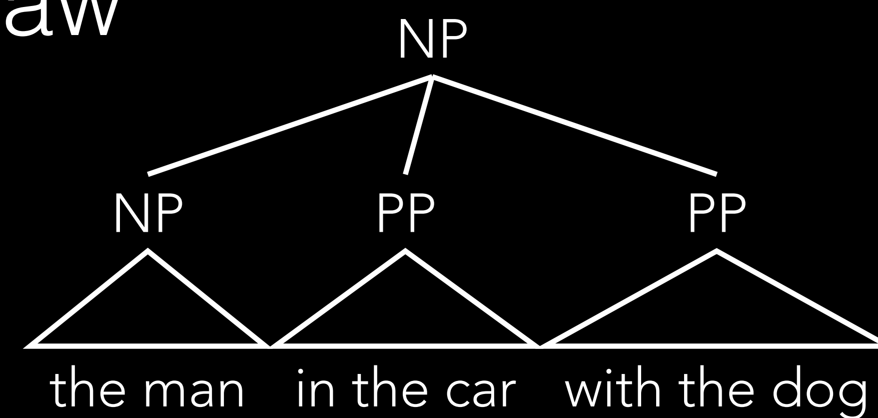
.05x…

Not Specific Enough (I)

If we saw



90 times

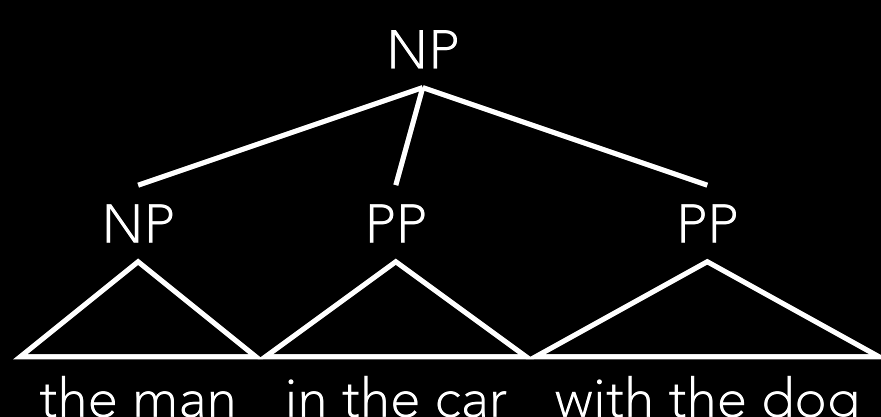


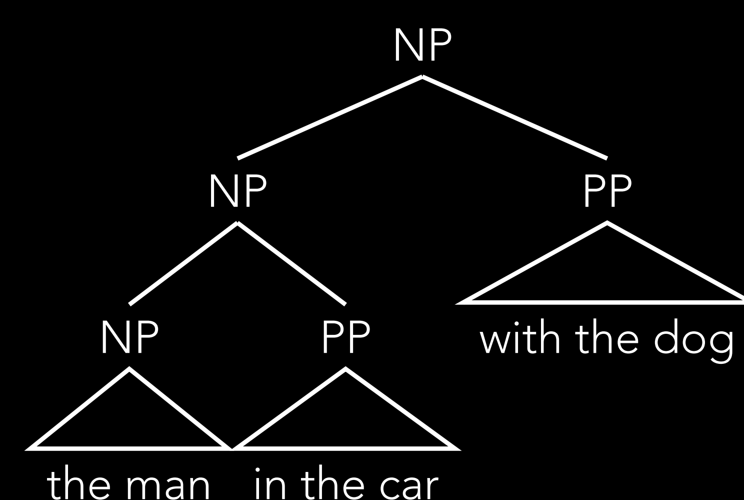
10 times

p(NP -> NP PP) = 90/200 = .45

p(NP -> NP PP PP) = 10/200 = .05

How would we score parses of “the man in the car with the dog”?



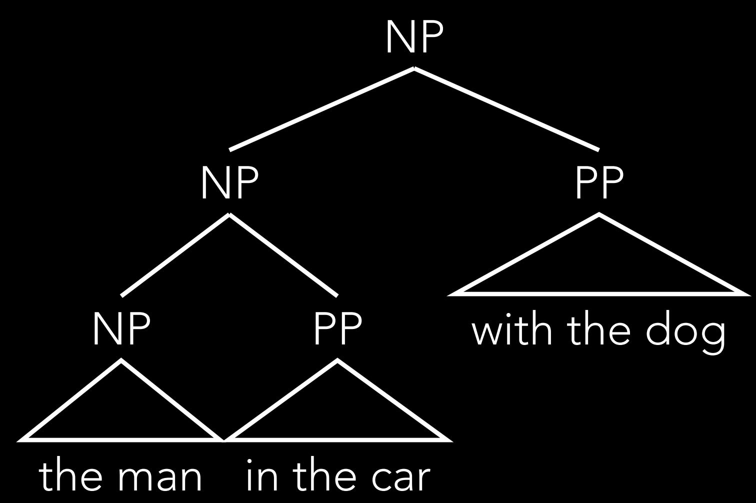


.05x…

.45x.45x… = .2025x…

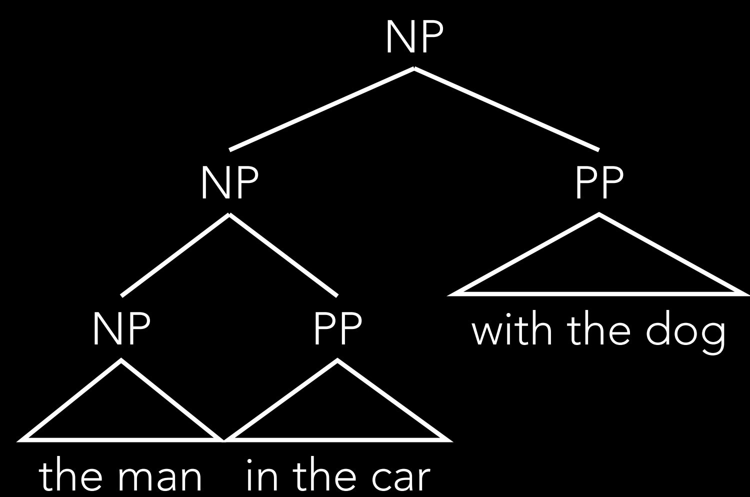
Not Specific Enough (I)

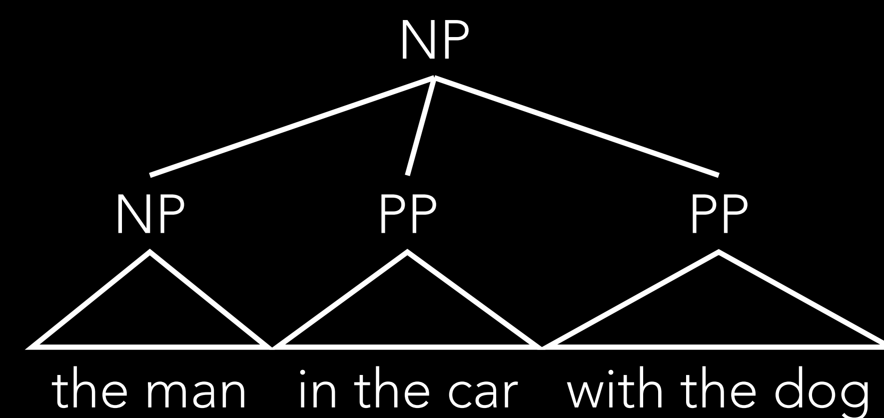
Not Specific Enough (I)



This was never seen in training

Not Specific Enough (I)

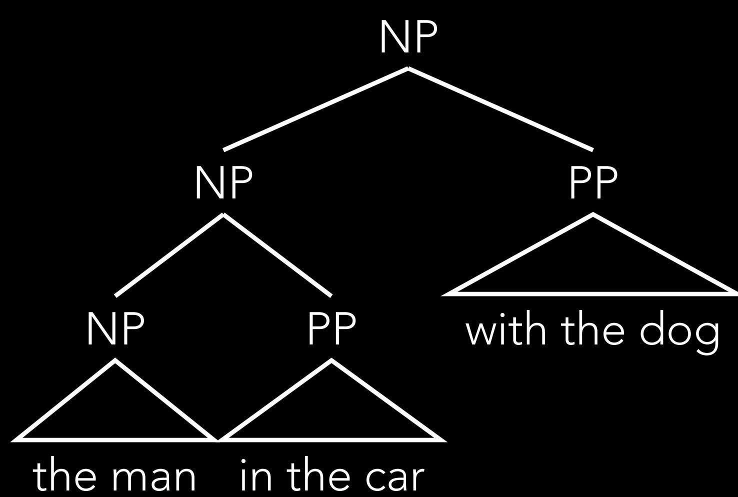




This was never seen in training

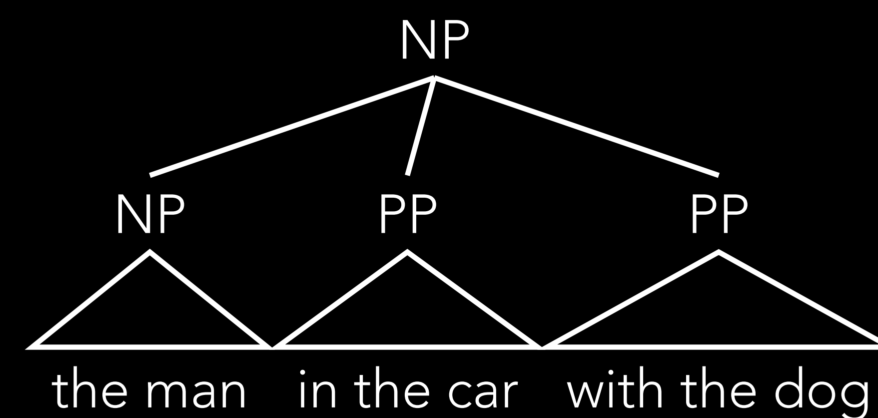
This was!

Not Specific Enough (I)



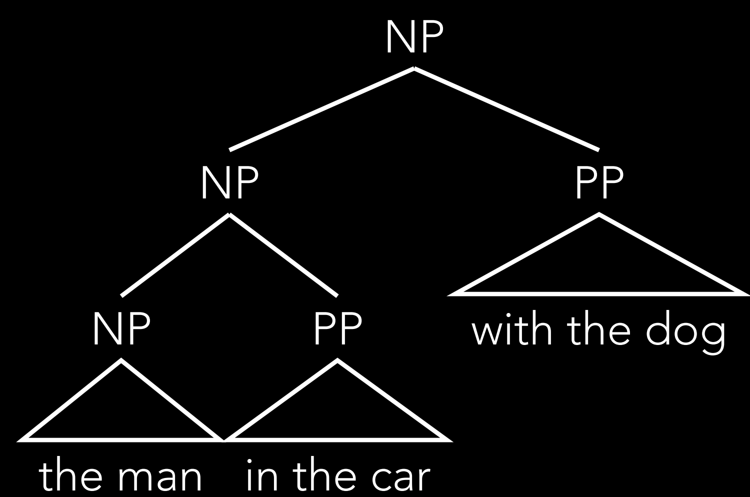
.2025

This was never seen in training



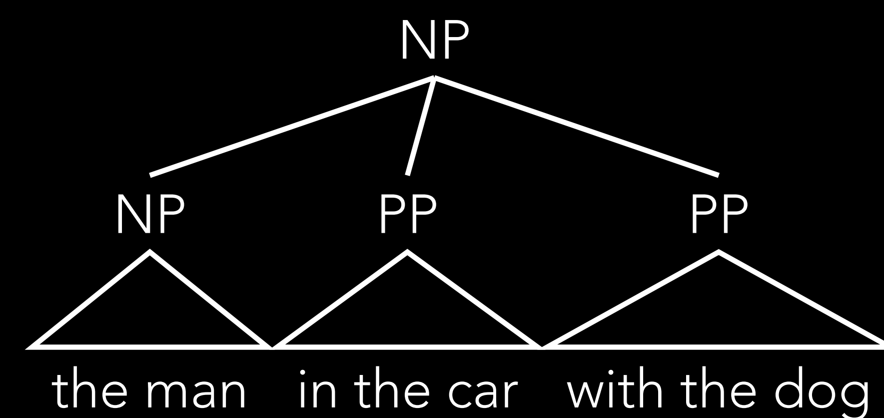
This was!

Not Specific Enough (I)



.2025

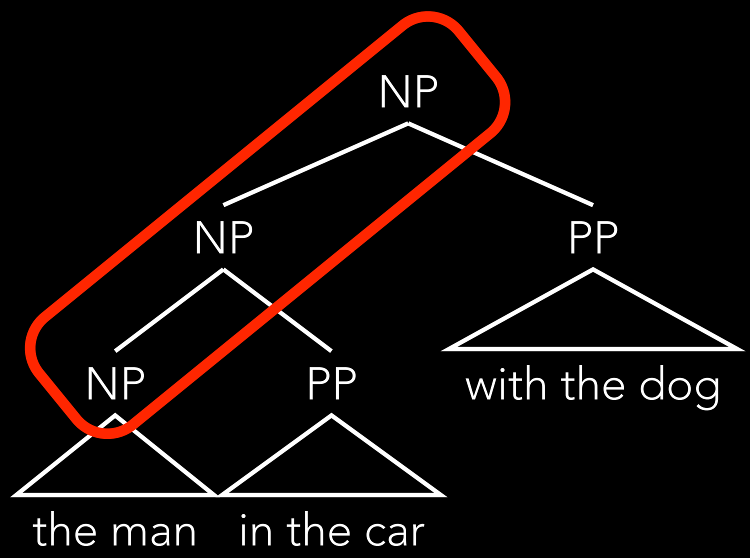
This was never seen in training

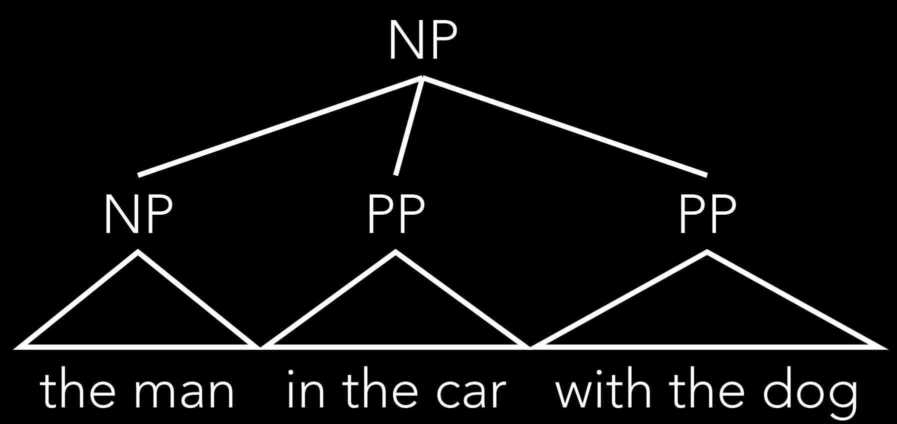


.05

This was!

Not Specific Enough (I)





.2025

This was never seen in training

.05

This was!

We should model parent-child behavior

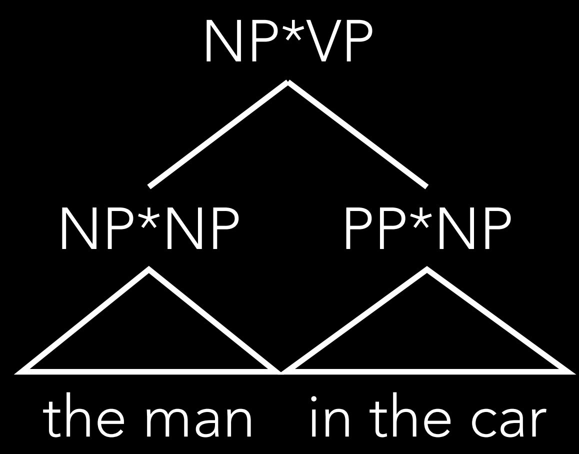
Not Specific Enough (I)

Not Specific Enough (I)

Add parent symbol annotation

Not Specific Enough (I)

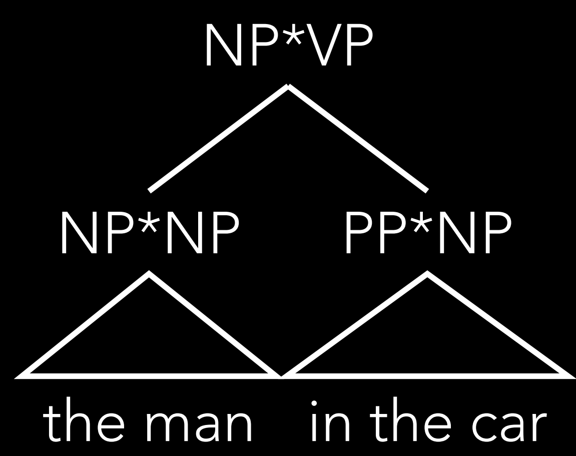
Add parent symbol annotation



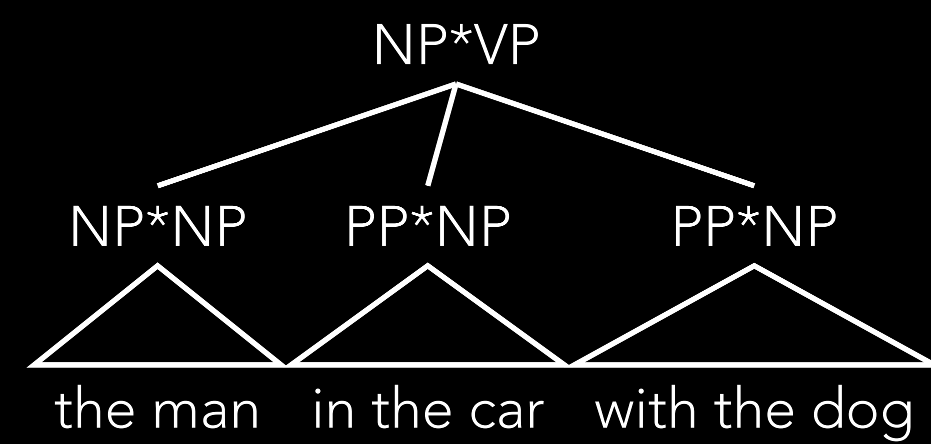
90 times

Not Specific Enough (I)

Add parent symbol annotation



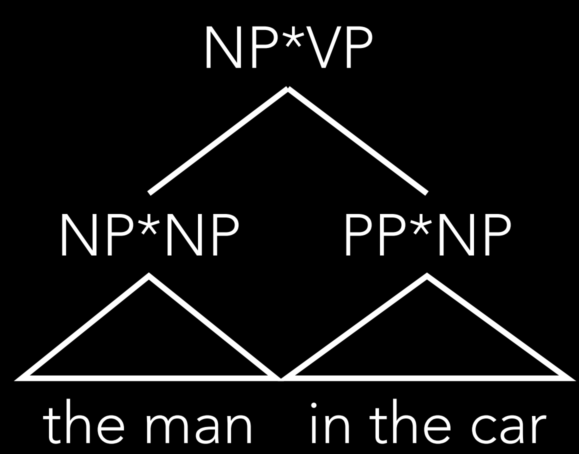
90 times

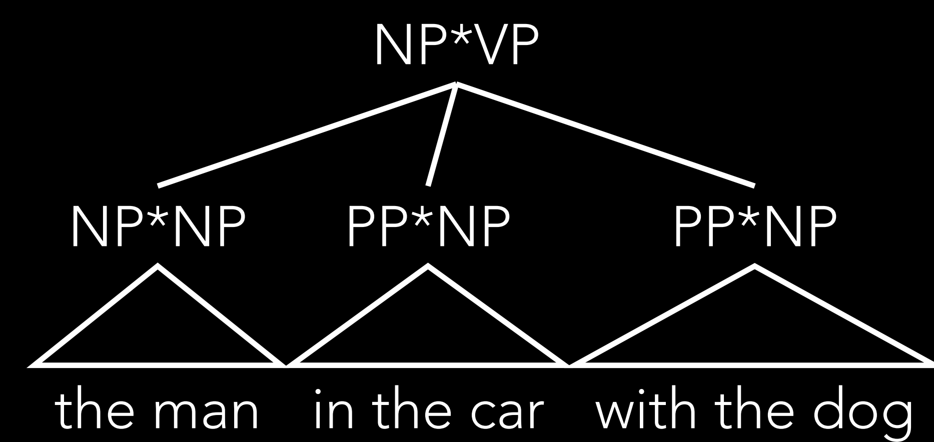


10 times

Not Specific Enough (I)

Add parent symbol annotation





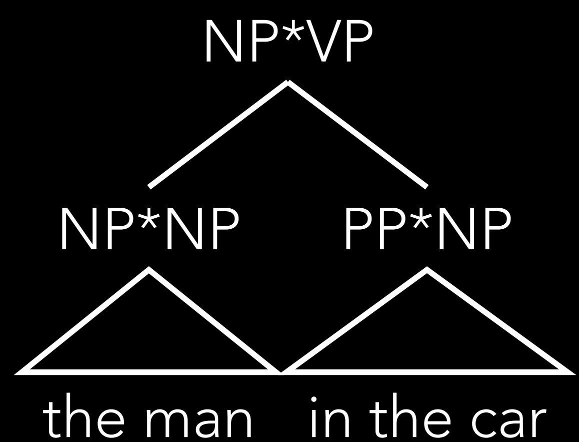
90 times

10 times

p(NP*VP -> NP*NP PP*NP)
= 90/200 = .45

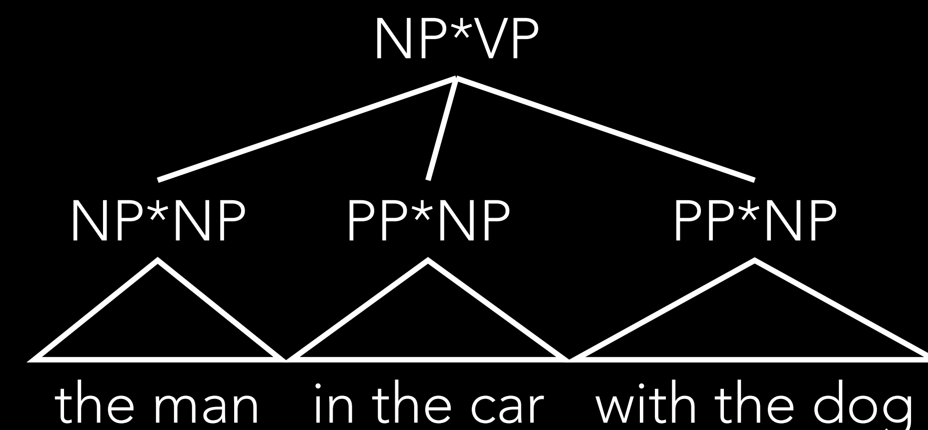
Not Specific Enough (I)

Add parent symbol annotation



90 times

p(NP*VP -> NP*NP PP*NP)
= 90/200 = .45

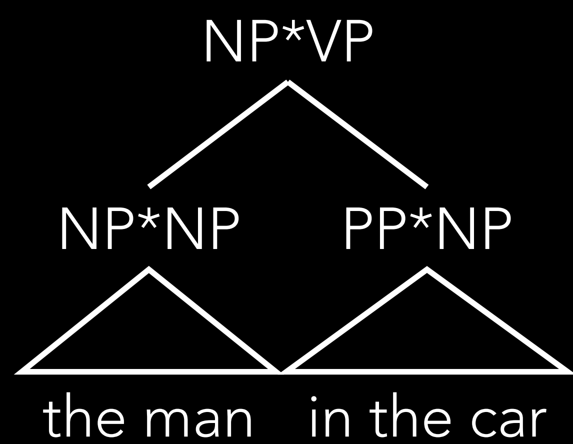


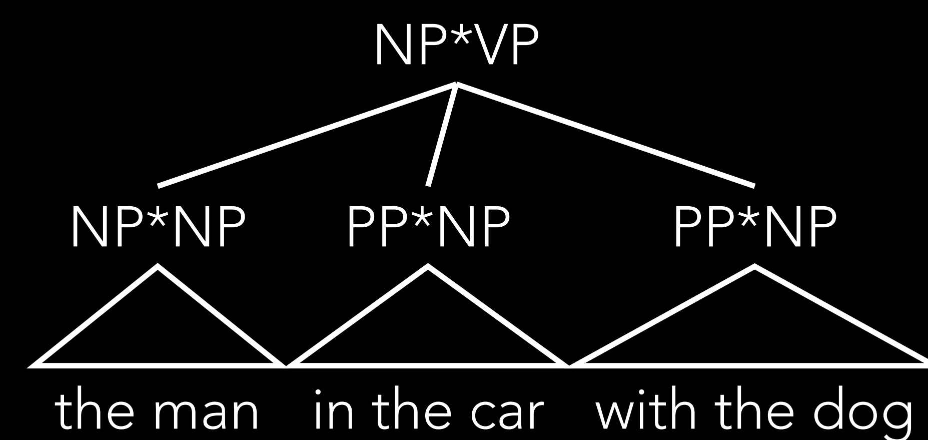
10 times

p(NP*VP -> NP*NP PP*NP PP*NP)
= 10/200 = .05

Not Specific Enough (I)

Add parent symbol annotation



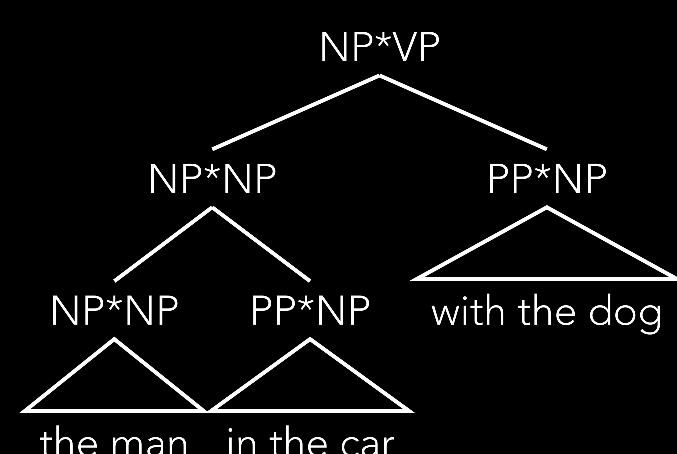


90 times

p(NP*VP -> NP*NP PP*NP)
= 90/200 = .45

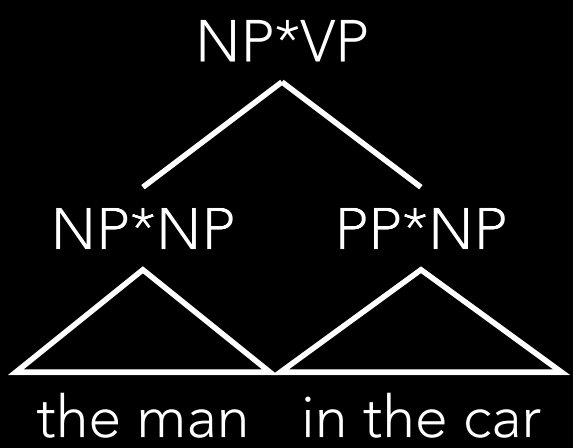
10 times

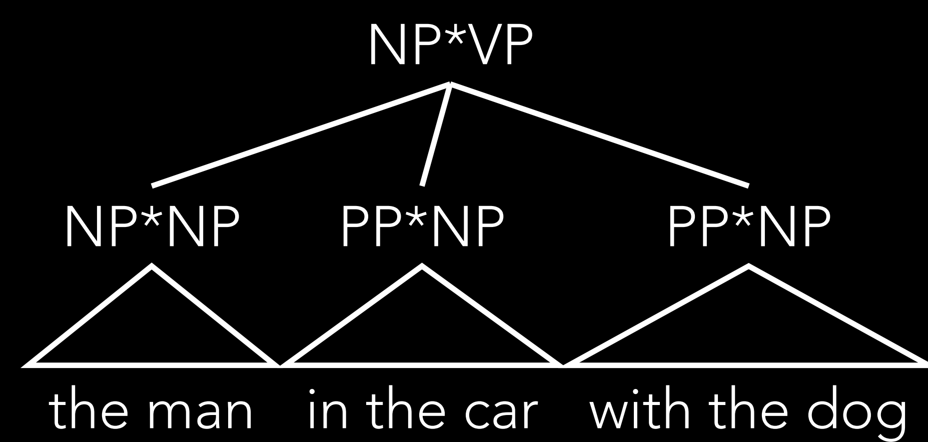
p(NP*VP -> NP*NP PP*NP PP*NP)
= 10/200 = .05



Not Specific Enough (I)

Add parent symbol annotation



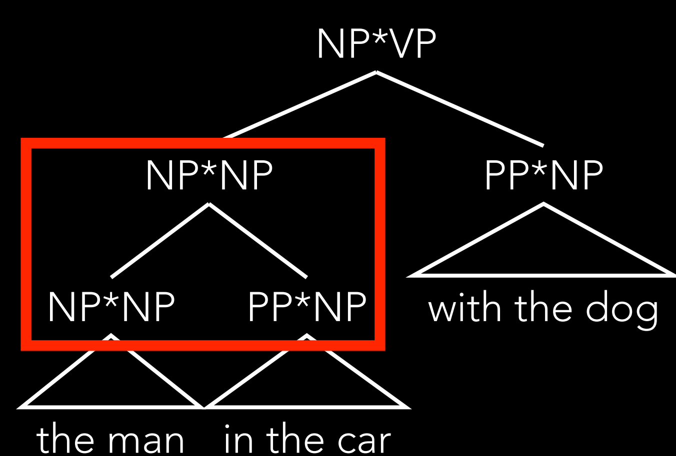


90 times

p(NP*VP -> NP*NP PP*NP)
= 90/200 = .45

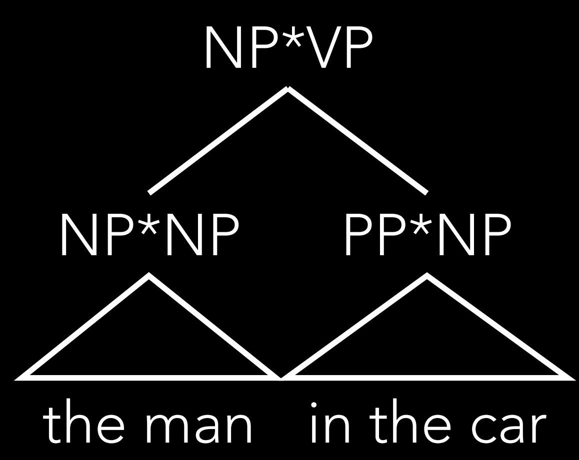
10 times

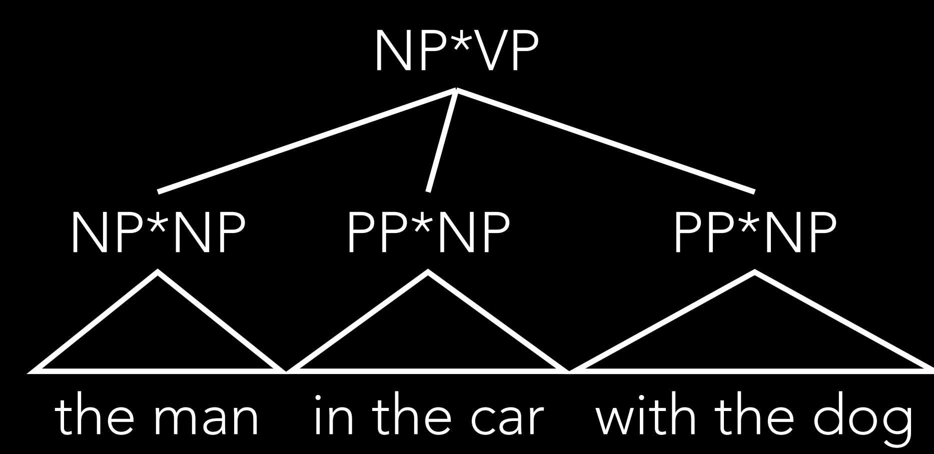
p(NP*VP -> NP*NP PP*NP PP*NP)
= 10/200 = .05



Not Specific Enough (I)

Add parent symbol annotation





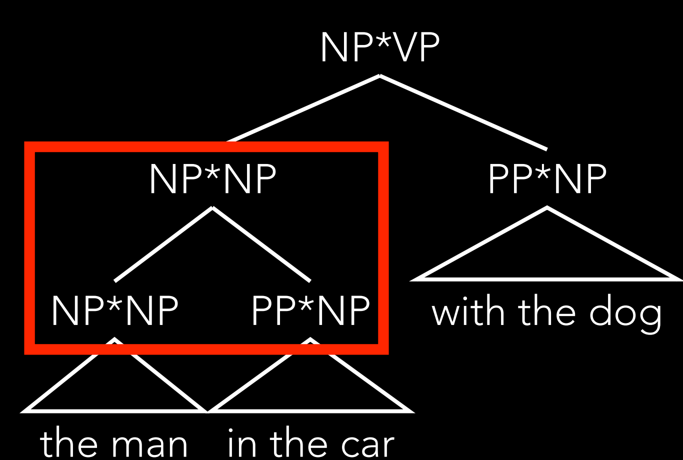
90 times

p(NP*VP -> NP*NP PP*NP)
= 90/200 = .45

10 times

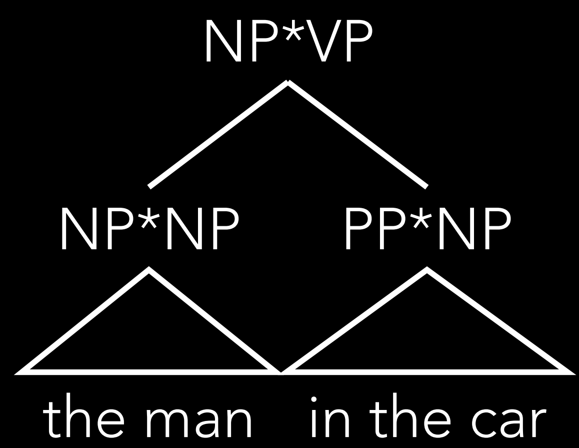
p(NP*VP -> NP*NP PP*NP PP*NP)
= 10/200 = .05

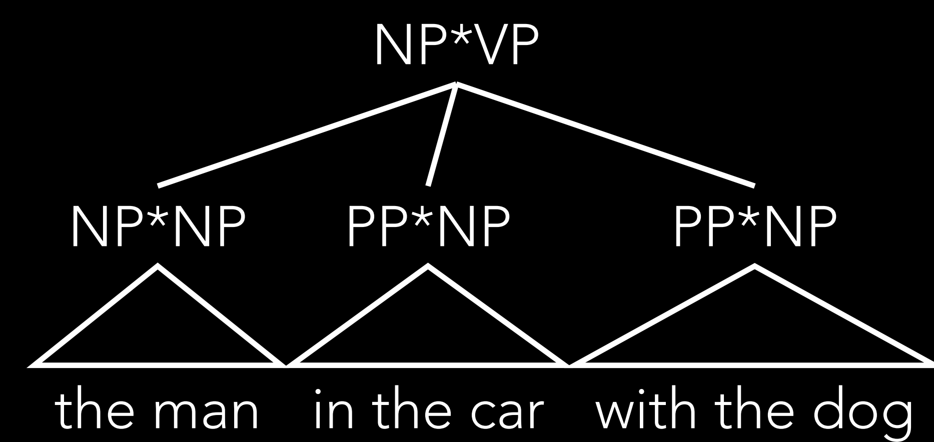
p(NP*NP -> NP*NP PP*NP) = 0



Not Specific Enough (I)

Add parent symbol annotation





90 times

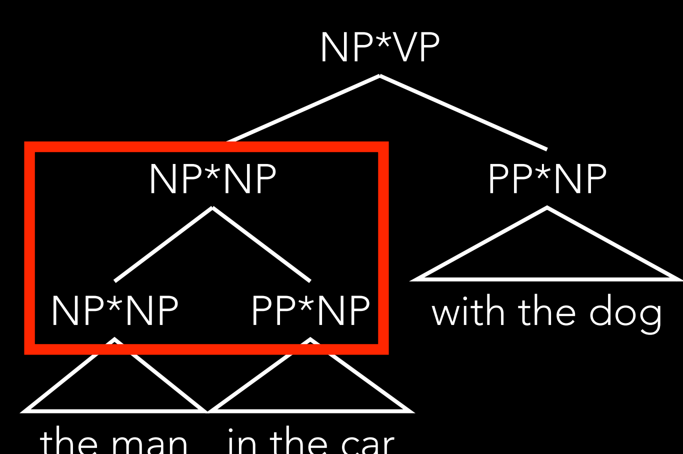
p(NP*VP -> NP*NP PP*NP)
= 90/200 = .45

10 times

p(NP*VP -> NP*NP PP*NP PP*NP)
= 10/200 = .05

p(NP*NP -> NP*NP PP*NP) = 0

Not possible!



Other Useful Category Splits

Other Useful Category Splits

- Mark base NPs (NPs that do not contain other NPs)

Other Useful Category Splits

- Mark base NPs (NPs that do not contain other NPs)
- Parent annotation on preterminals

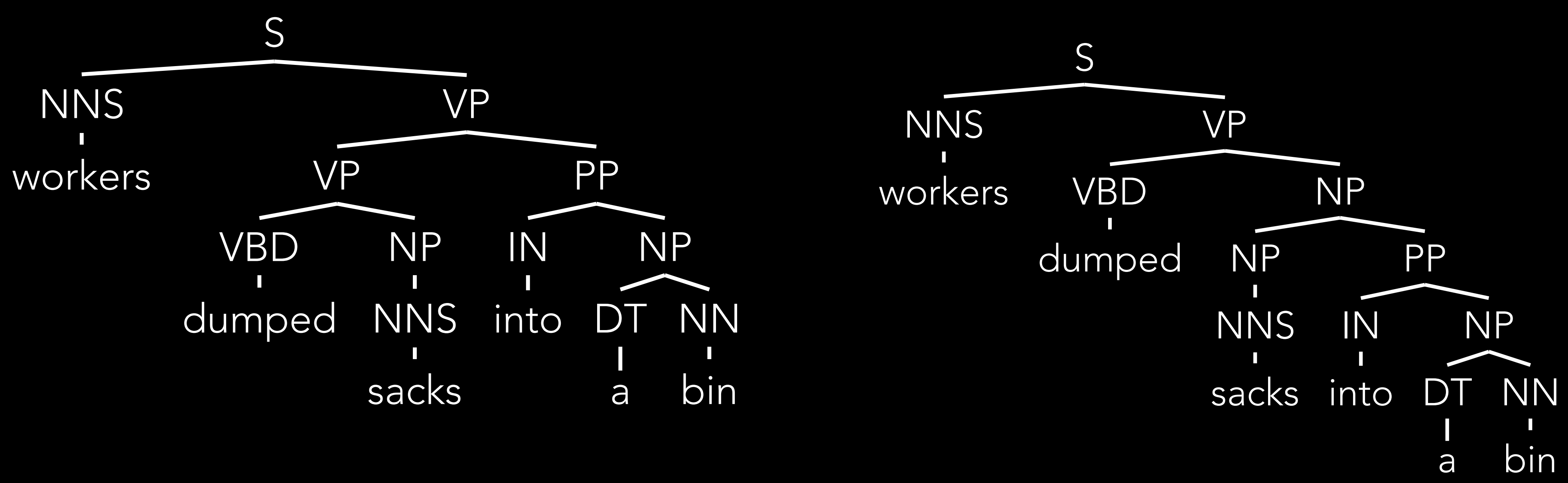
Other Useful Category Splits

- Mark base NPs (NPs that do not contain other NPs)
- Parent annotation on preterminals
- Subdivide IN, CC, “have” vs “be” auxiliary

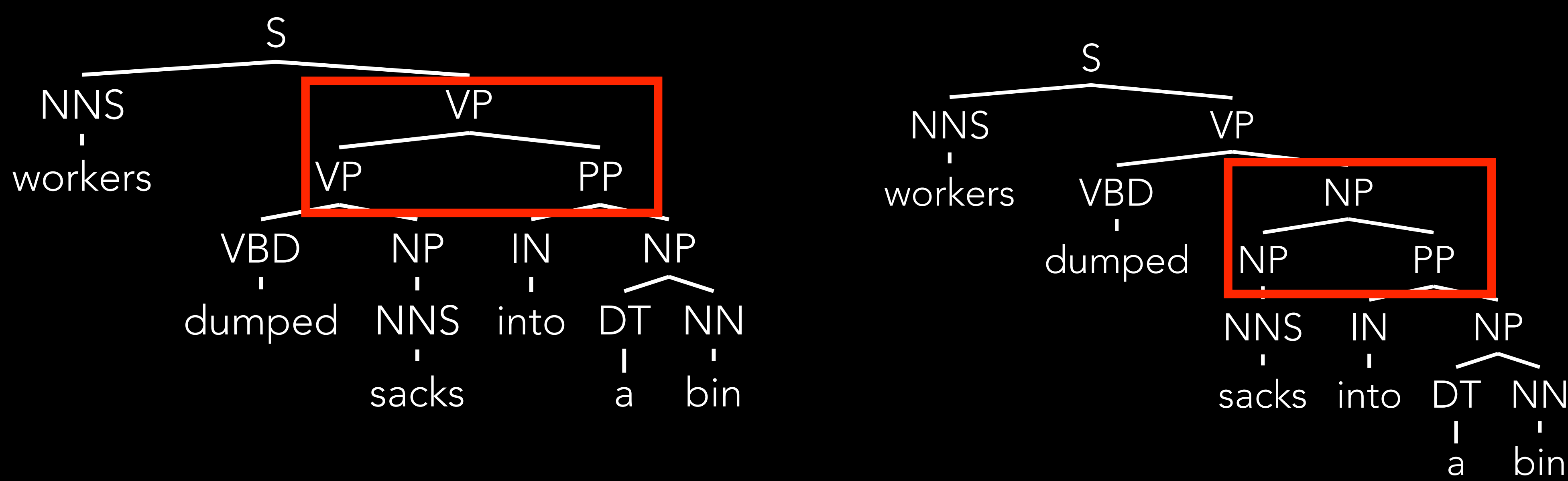
Other Useful Category Splits

- Mark base NPs (NPs that do not contain other NPs)
- Parent annotation on preterminals
- Subdivide IN, CC, “have” vs “be” auxiliary
- See “Accurate Unlexicalized Parsing” (Klein & Manning, 2003)

Not Specific Enough (II)

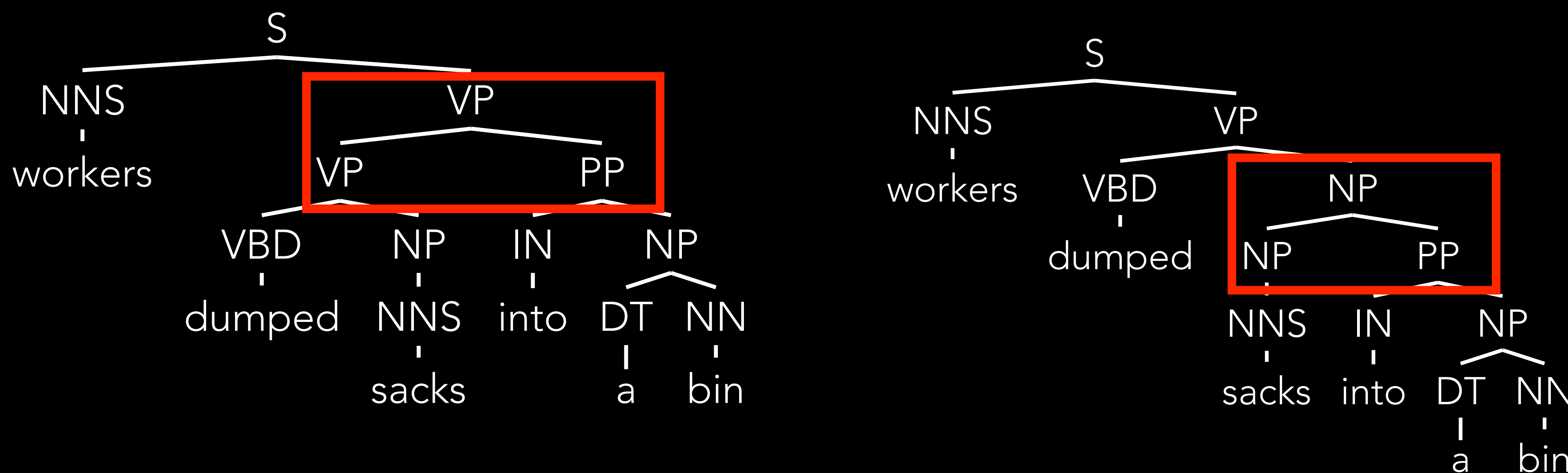


Not Specific Enough (II)



These derivations only differ by the highlighted rules

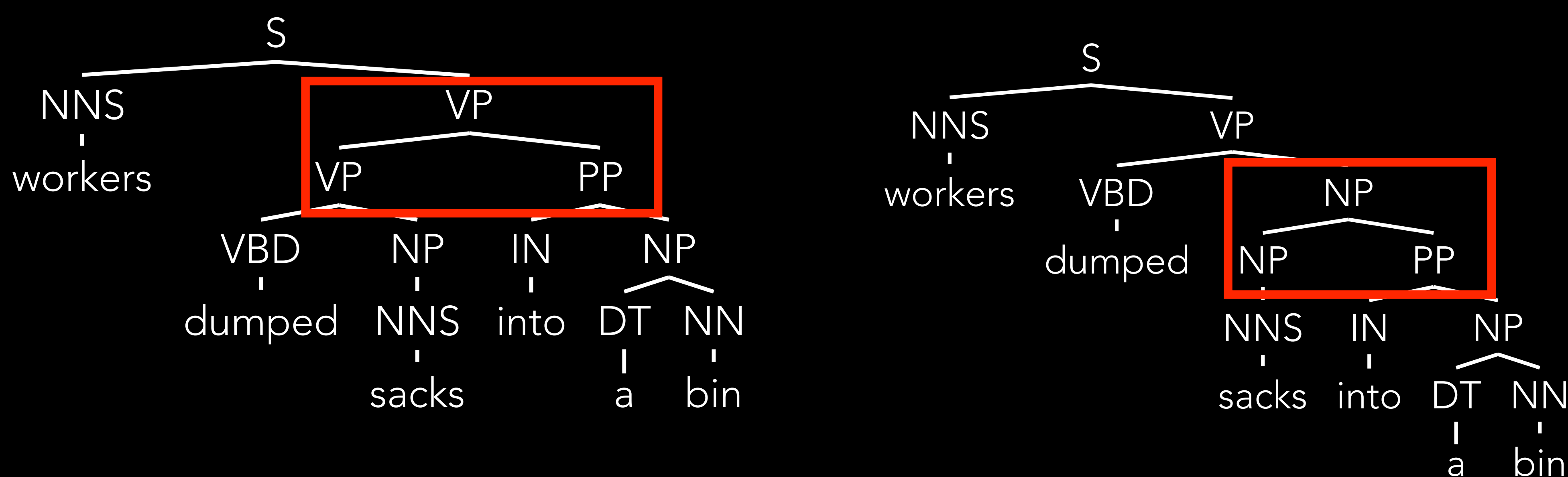
Not Specific Enough (II)



These derivations only differ by the highlighted rules

If p(VP -> VP PP) > p(NP -> NP PP) then
the left derivation wins. Else, the right derivation

Not Specific Enough (II)



These derivations only differ by the highlighted rules

If p(VP -> VP PP) > p(NP -> NP PP) then
the left derivation wins. Else, the right derivation

The words (particularly “into”) are not considered. PPs with “into” are much more likely to attach to VP than NP.

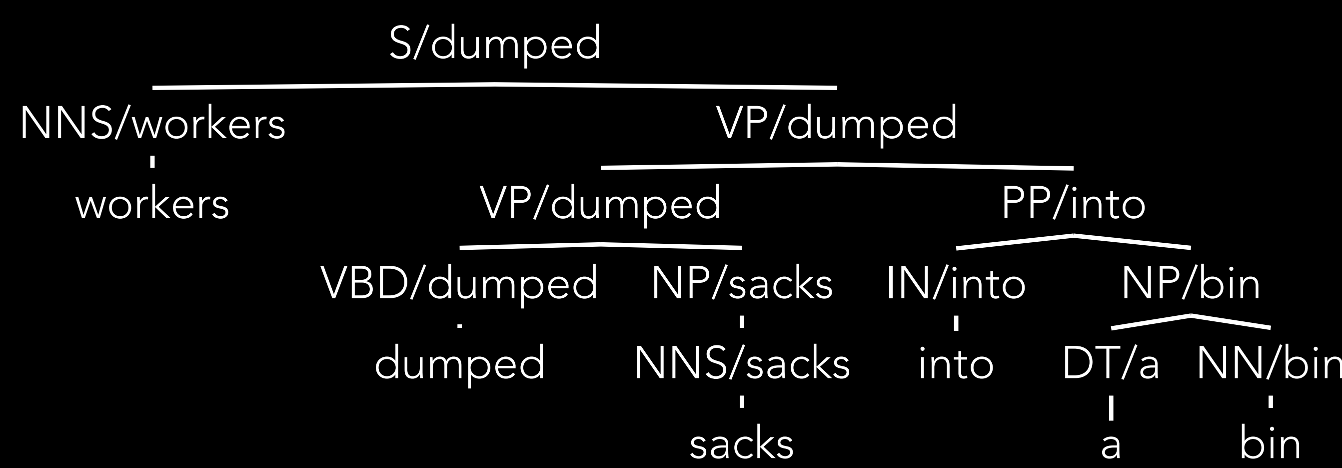
Head lexicalization

Head lexicalization

Annotate tree labels with their “head” word

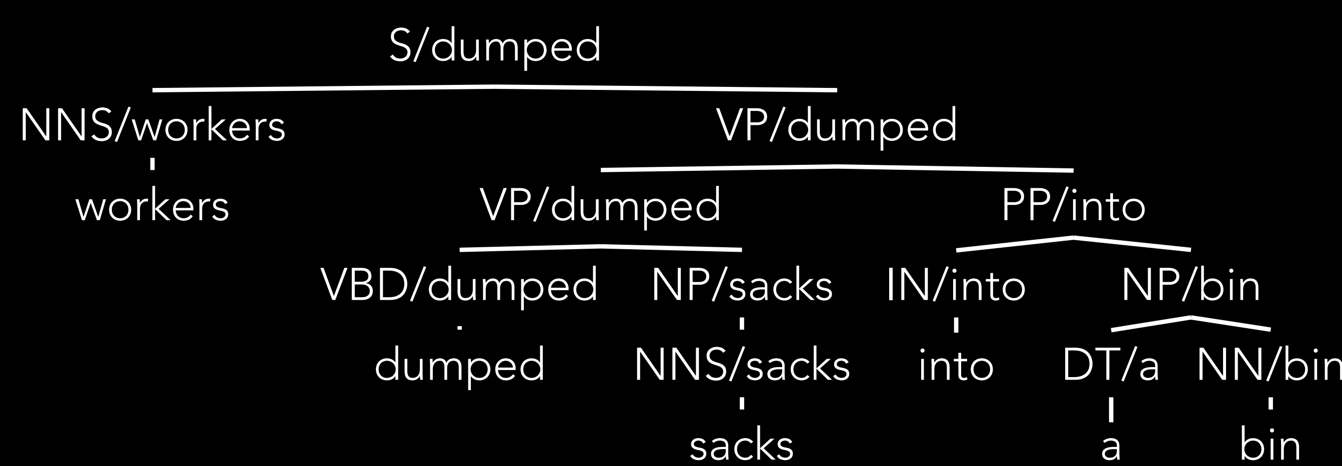
Head lexicalization

Annotate tree labels with their “head” word



Head lexicalization

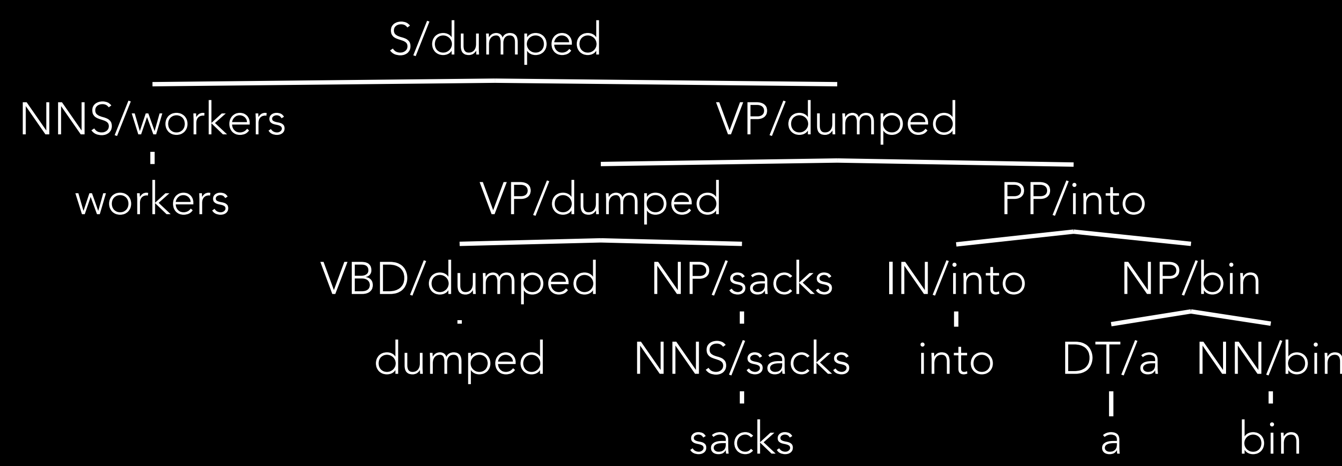
Annotate tree labels with their “head” word



Heads are the “important” word in a phrase

Head lexicalization

Annotate tree labels with their "head" word

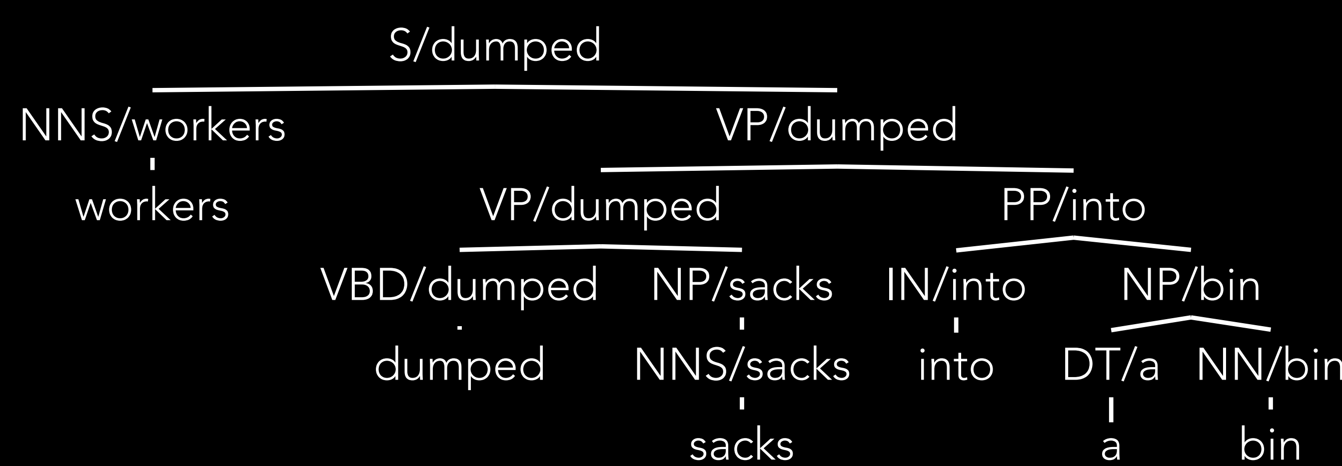


Heads are the "important" word in a phrase

They could be provided by annotators

Head lexicalization

Annotate tree labels with their “head” word



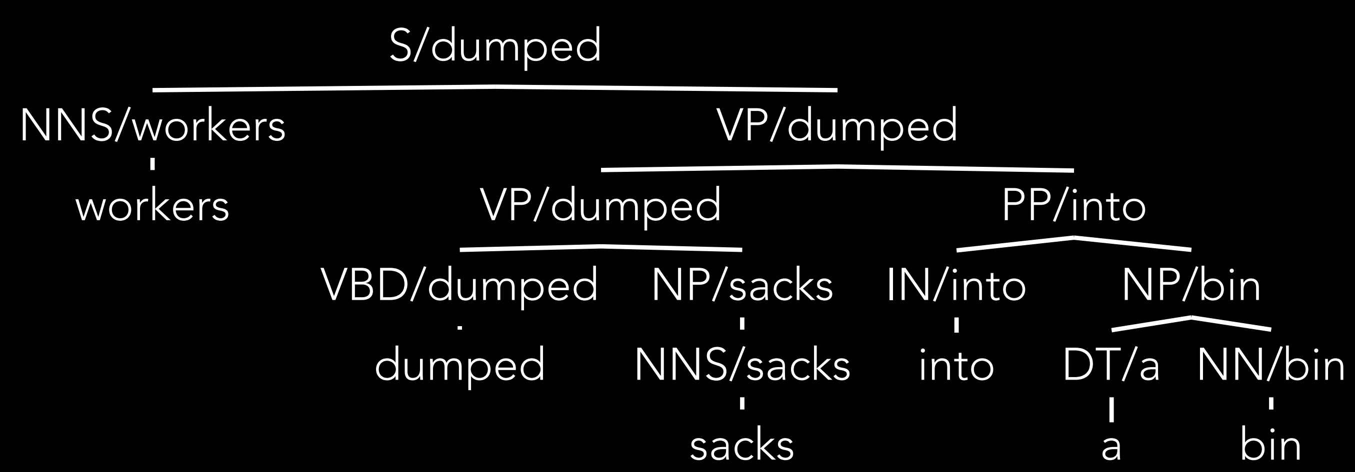
Heads are the “important” word in a phrase

They could be provided by annotators

In practice, for English, some heuristics work well

Head lexicalization

Annotate tree labels with their "head" word



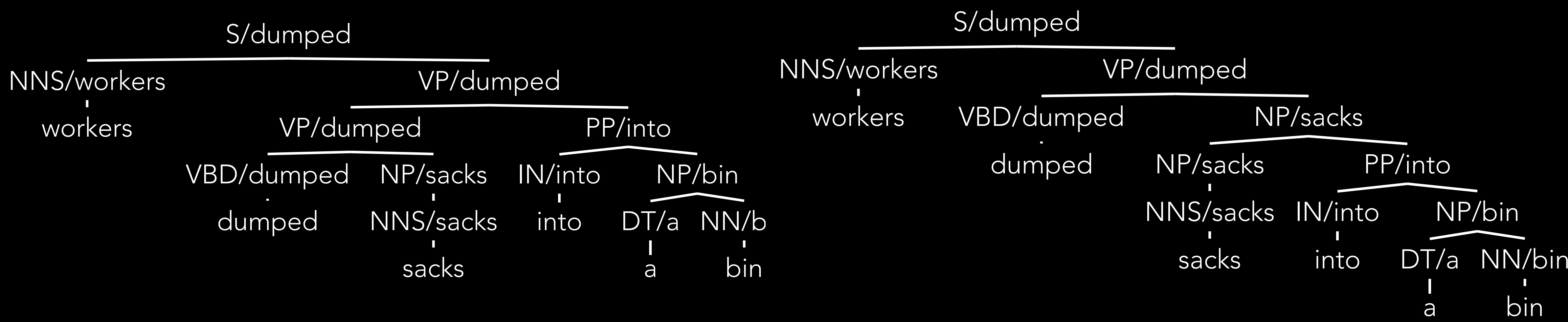
Heads are the "important" word in a phrase

They could be provided by annotators

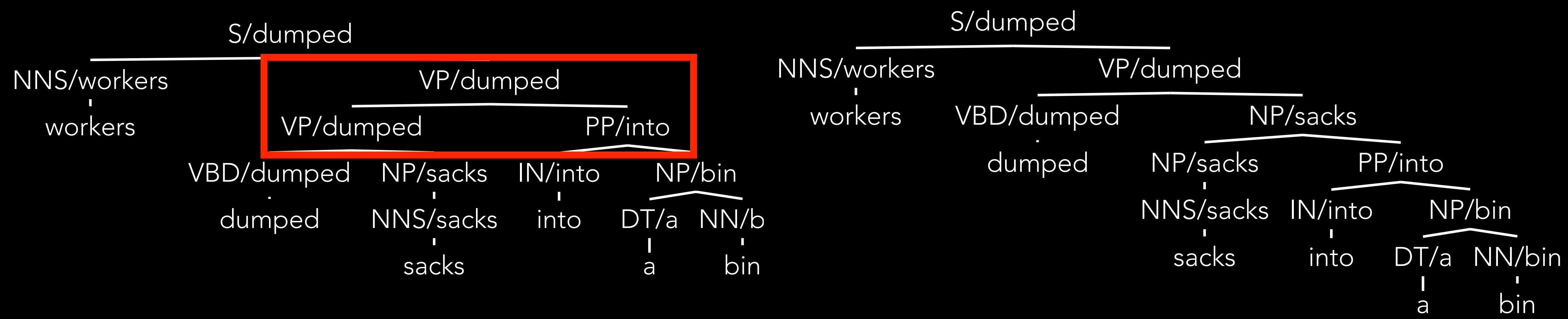
In practice, for English, some heuristics work well

http://www.cs.columbia.edu/~mcollins/papers/heads

Head lexicalization

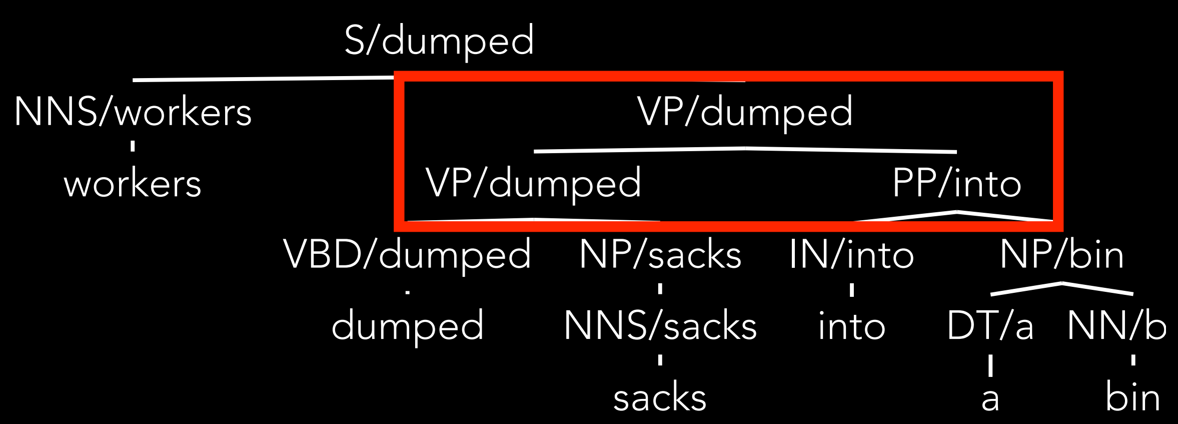


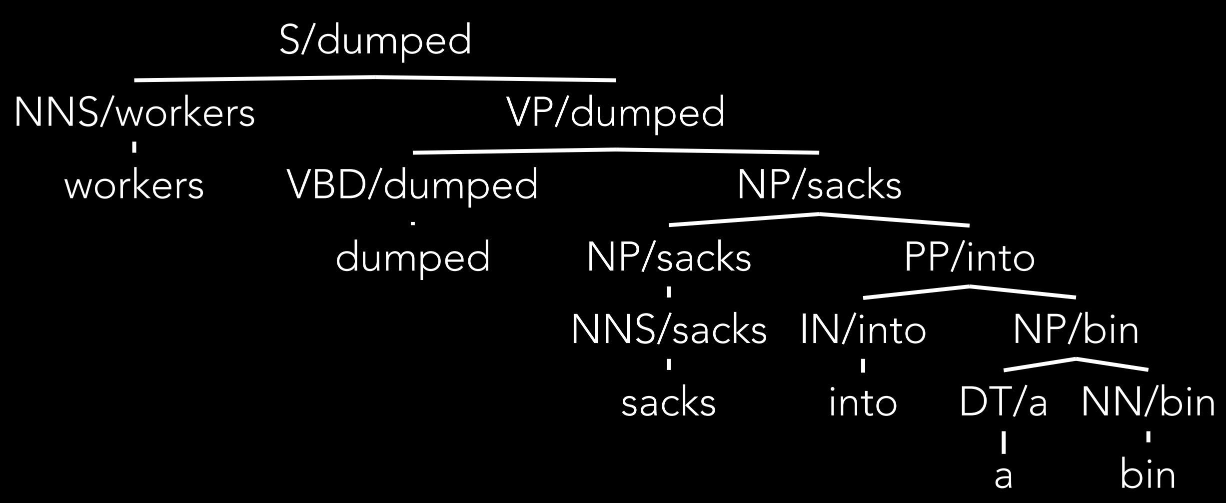
Head lexicalization



VP/dumped -> VP/dumped PP/into

Head lexicalization

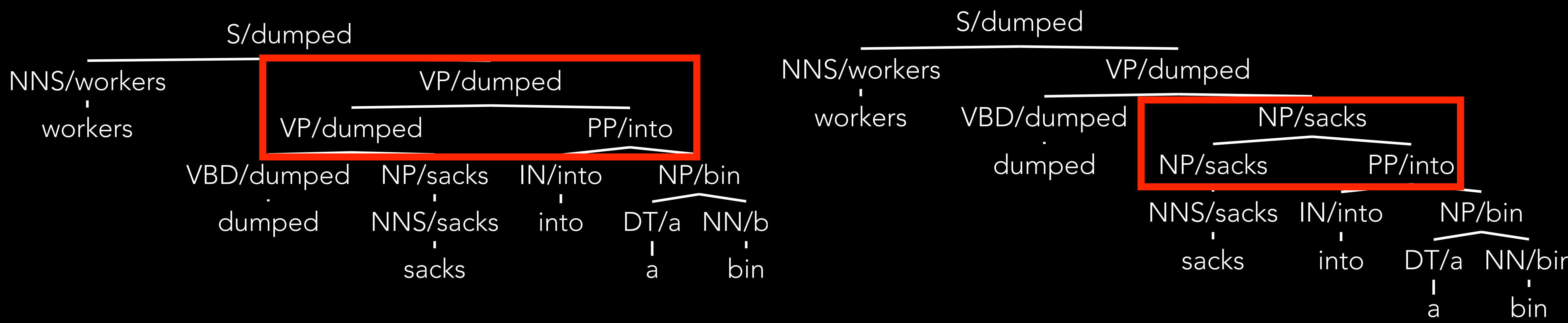




VP/dumped -> VP/dumped PP/into

more likely than

Head lexicalization



VP/dumped -> VP/dumped PP/into

more likely than

NP/sacks-> NP/sacks PP/into

Lexicalization and Binarization

Lexicalization and Binarization

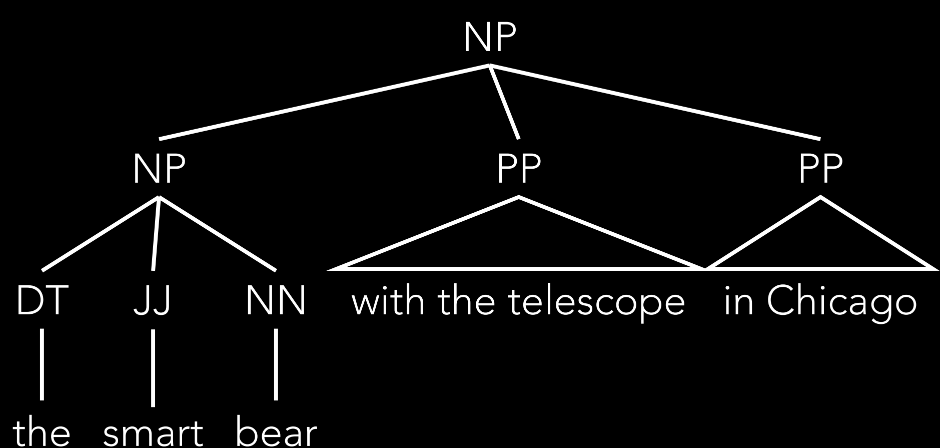
- Want to keep meaningful head at every point

Lexicalization and Binarization

- Want to keep meaningful head at every point
- Need to make binarization head-aware

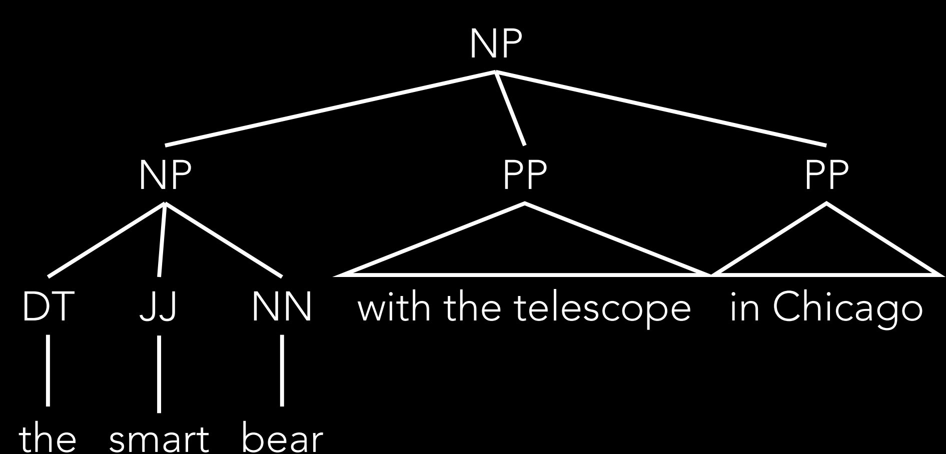
Lexicalization and Binarization

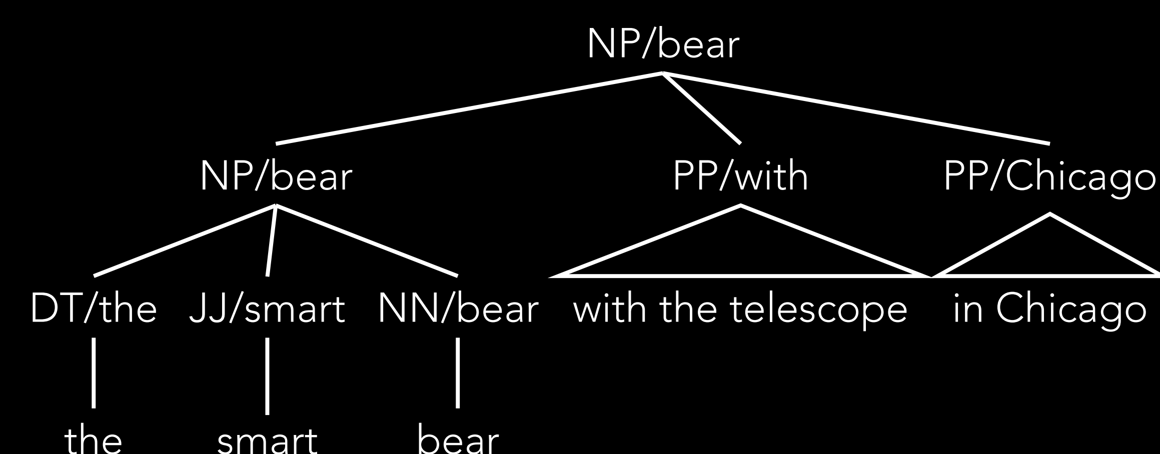
- Want to keep meaningful head at every point
- Need to make binarization head-aware



Lexicalization and Binarization

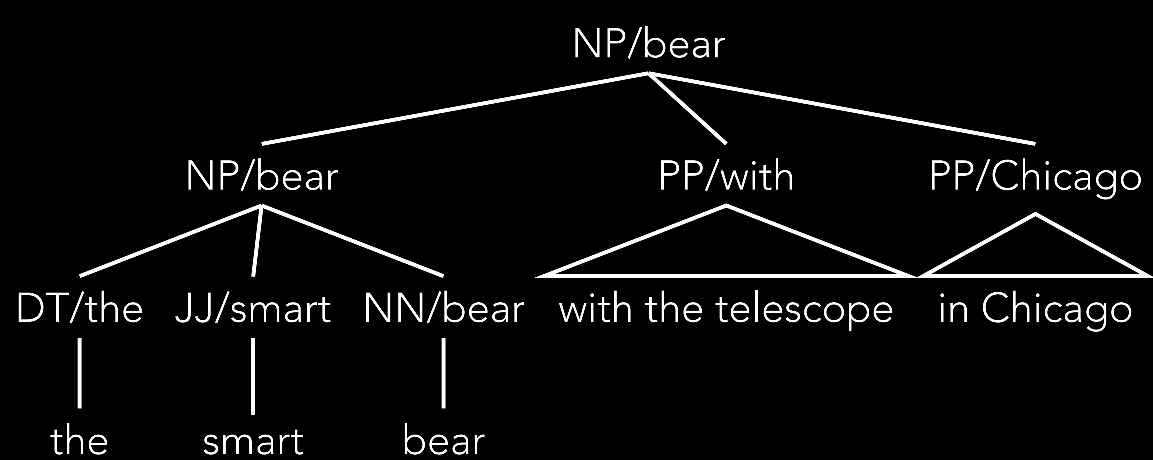
- Want to keep meaningful head at every point
- Need to make binarization head-aware





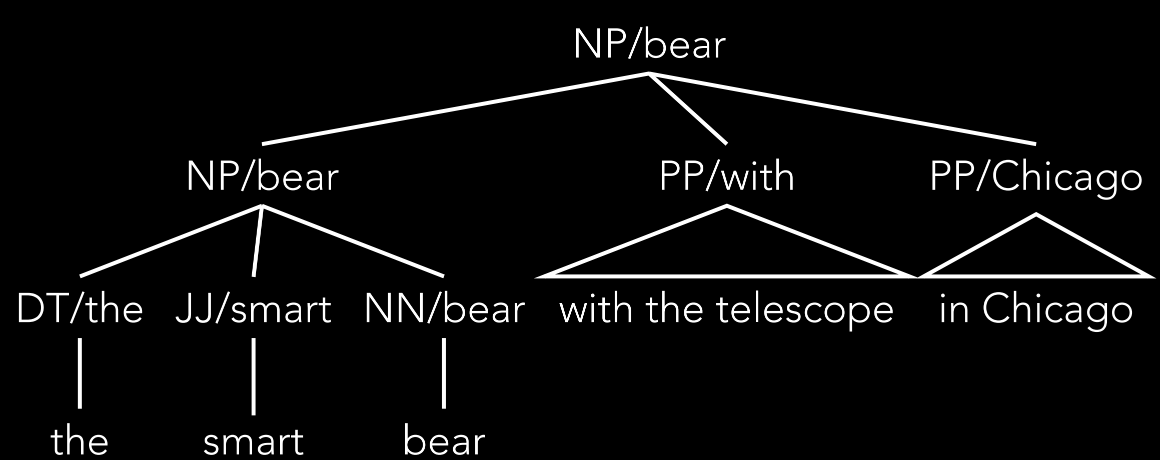
Lexicalization and Binarization

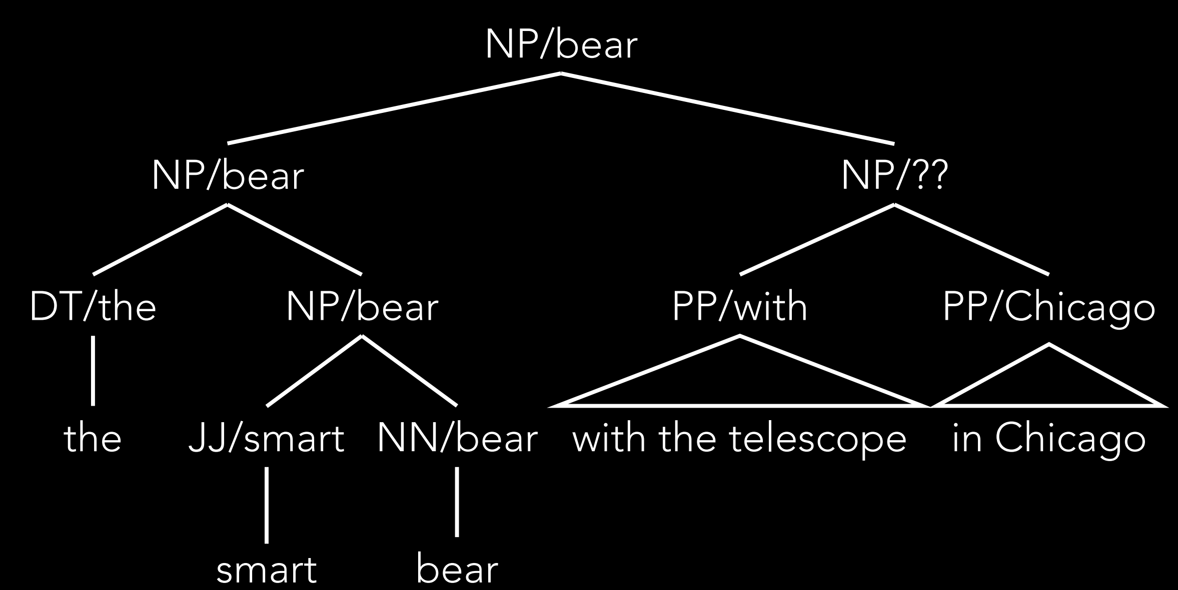
- Want to keep meaningful head at every point
- Need to make binarization head-aware



Lexicalization and Binarization

- Want to keep meaningful head at every point
- Need to make binarization head-aware

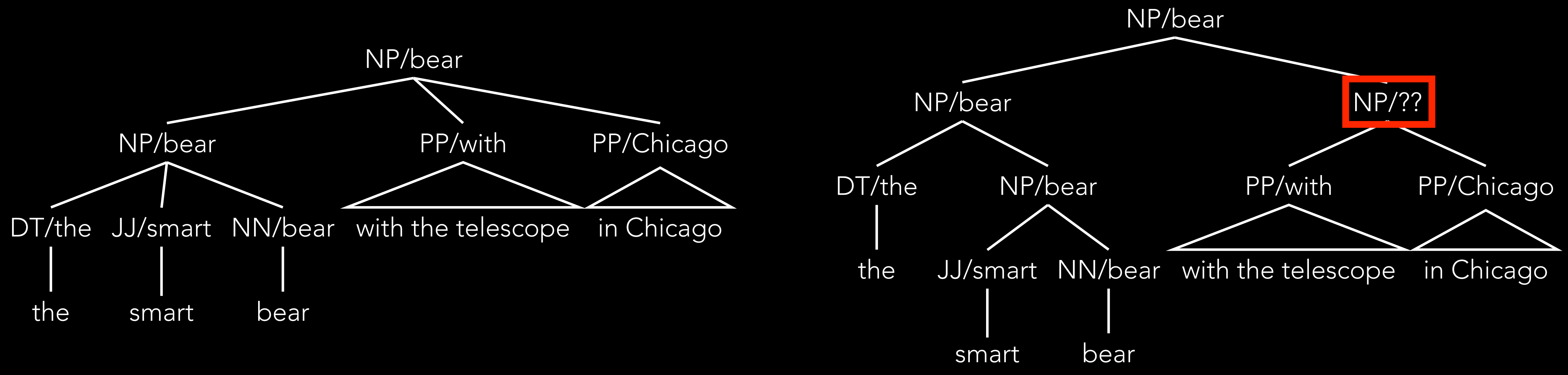




Right binarization

Lexicalization and Binarization

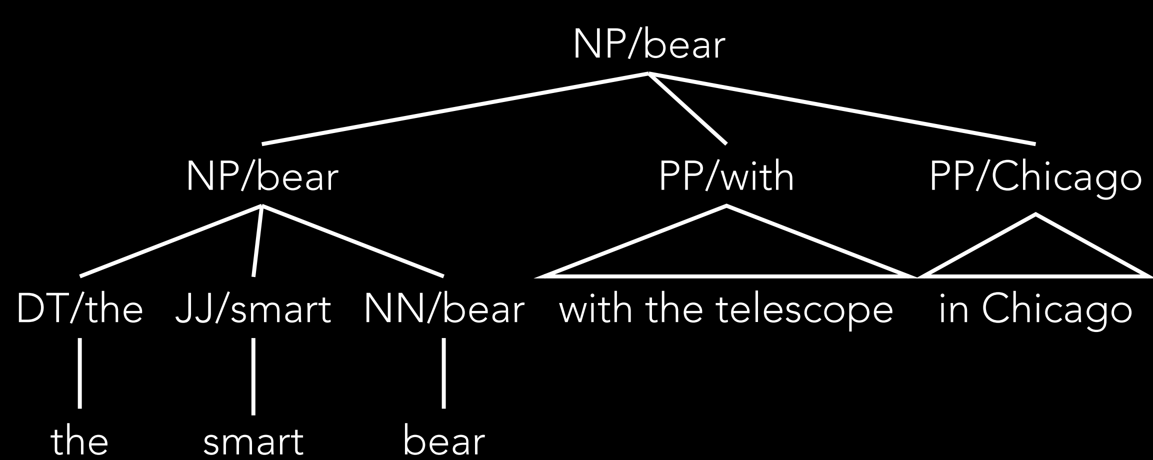
- Want to keep meaningful head at every point
- Need to make binarization head-aware



Right binarization

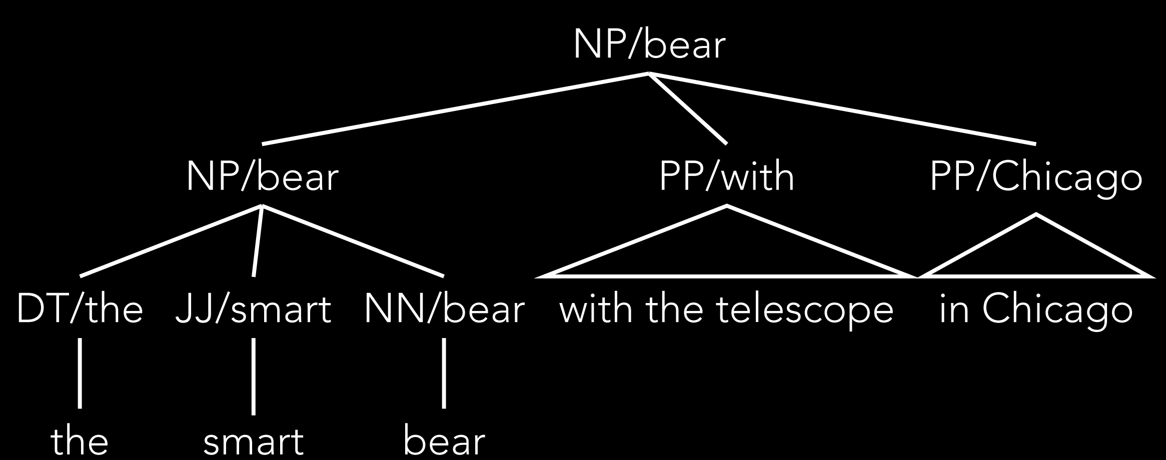
Lexicalization and Binarization

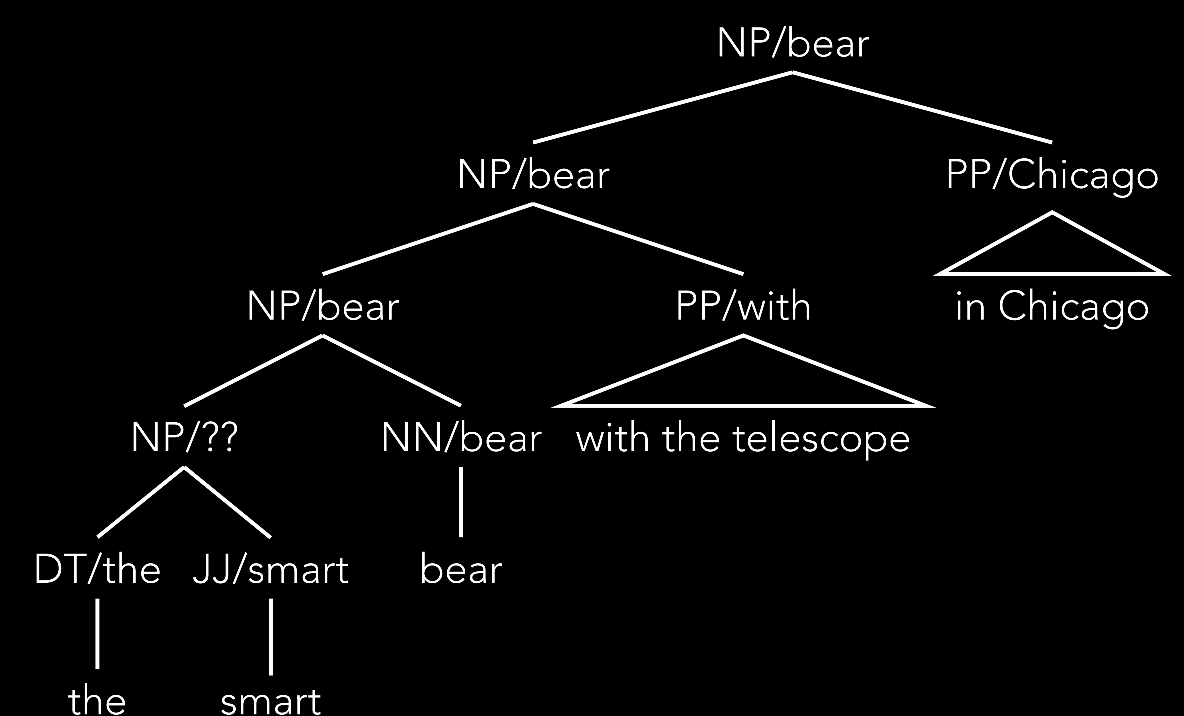
- Want to keep meaningful head at every point
- Need to make binarization head-aware



Lexicalization and Binarization

- Want to keep meaningful head at every point
- Need to make binarization head-aware

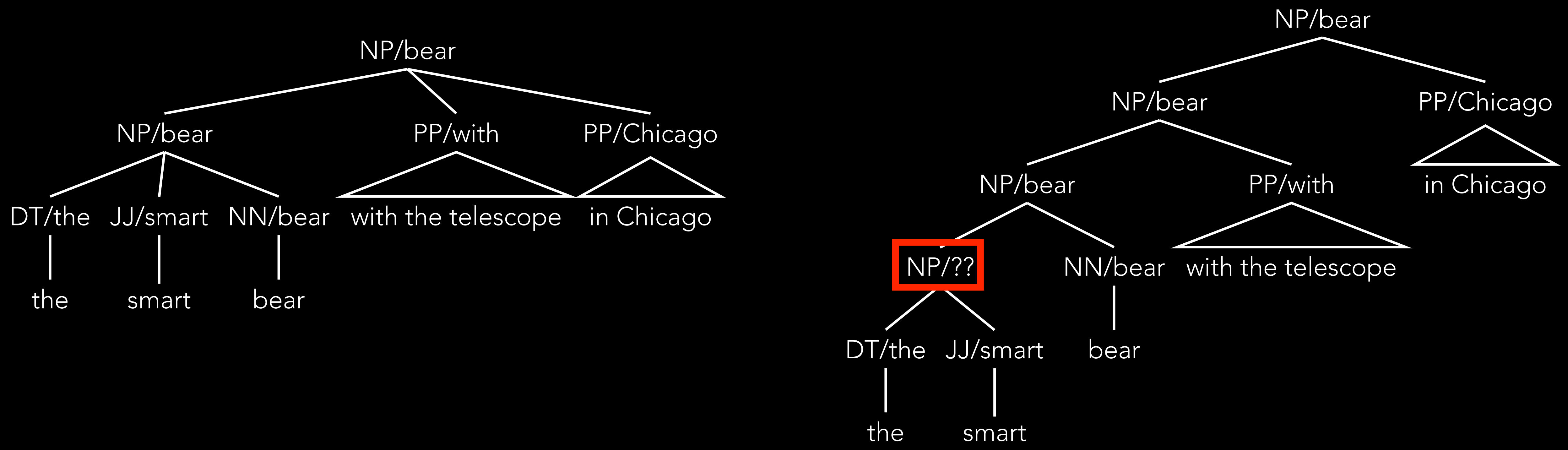




Left binarization

Lexicalization and Binarization

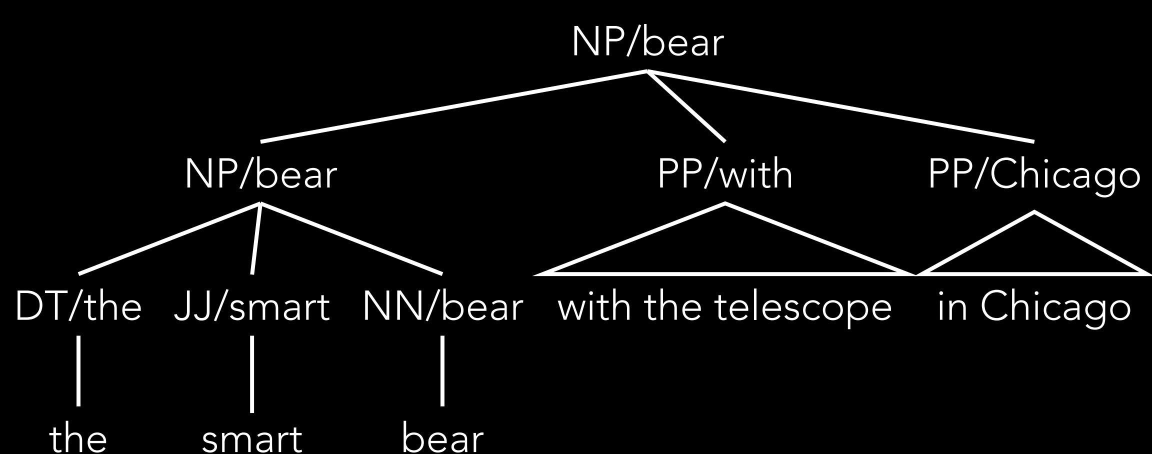
- Want to keep meaningful head at every point
- Need to make binarization head-aware



Left binarization

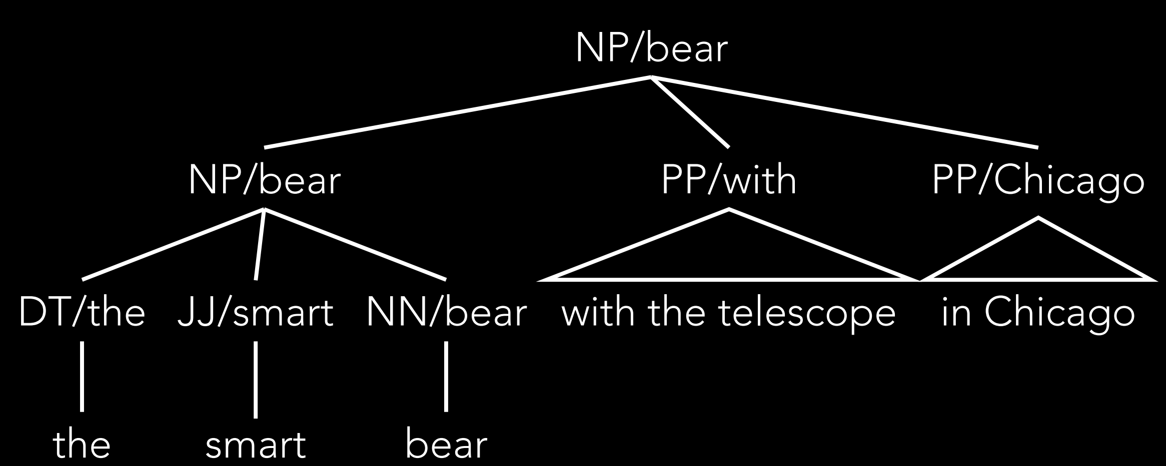
Lexicalization and Binarization

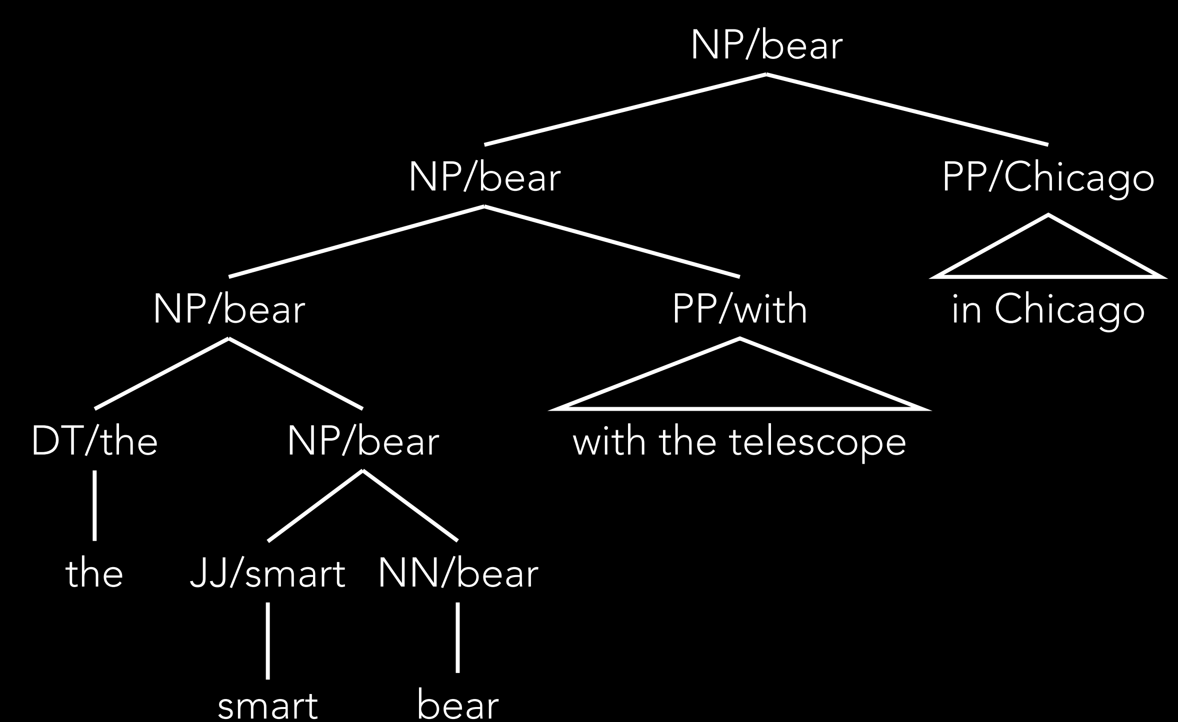
- Want to keep meaningful head at every point
- Need to make binarization head-aware



Lexicalization and Binarization

- Want to keep meaningful head at every point
- Need to make binarization head-aware

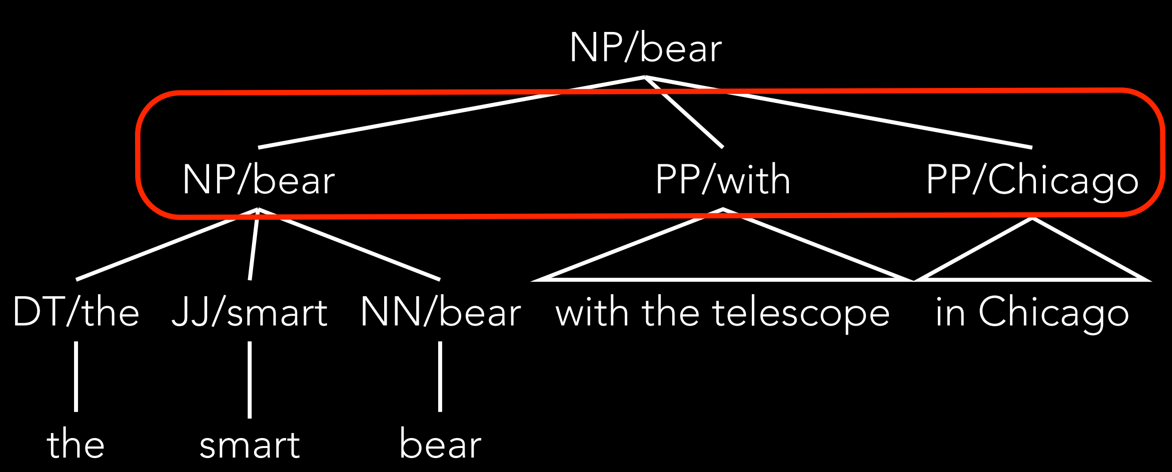




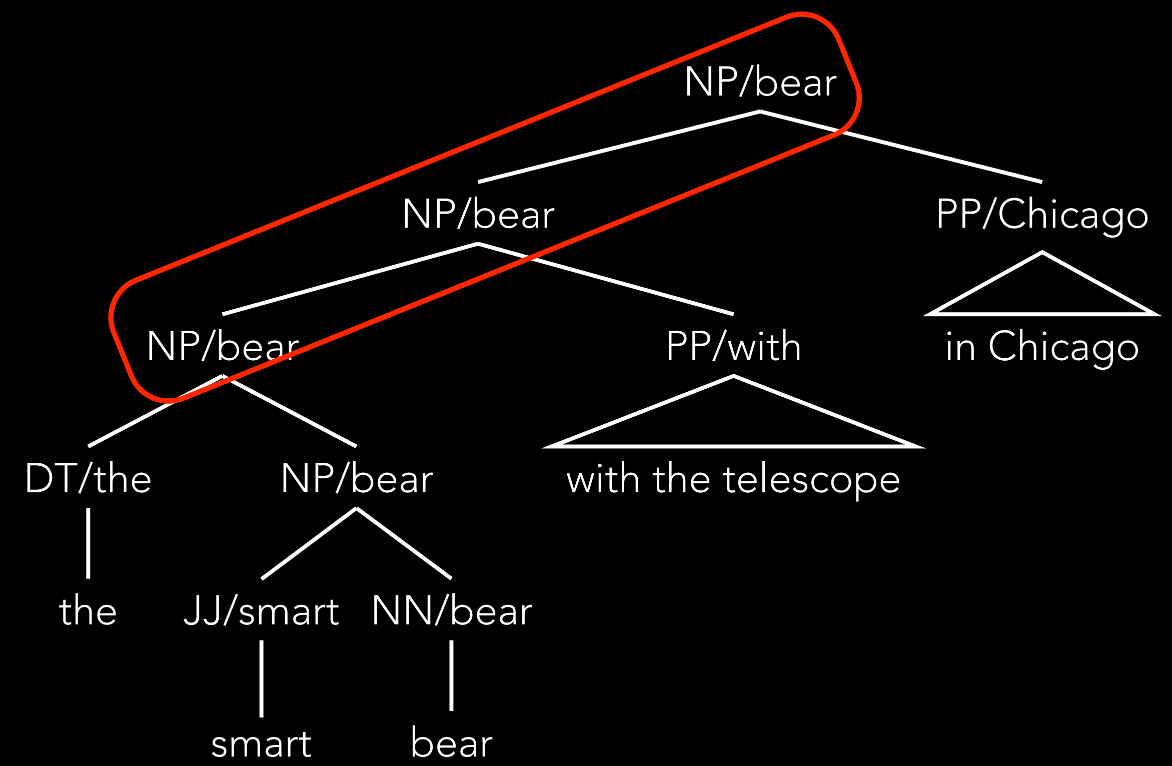
Head binarization!

Lexicalization and Binarization

- Want to keep meaningful head at every point
- Need to make binarization head-aware



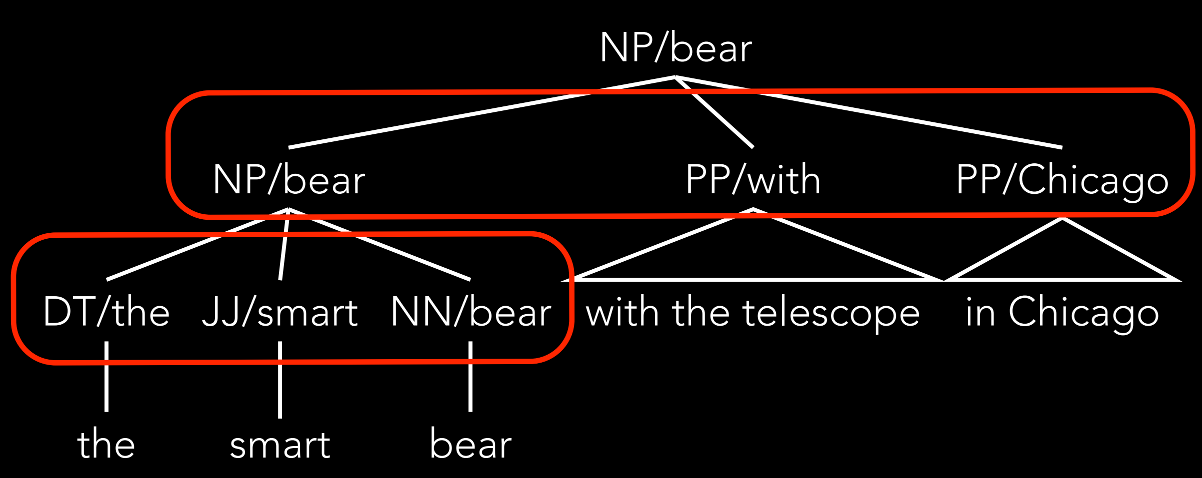
Left (for NP PP PP)



Head binarization!

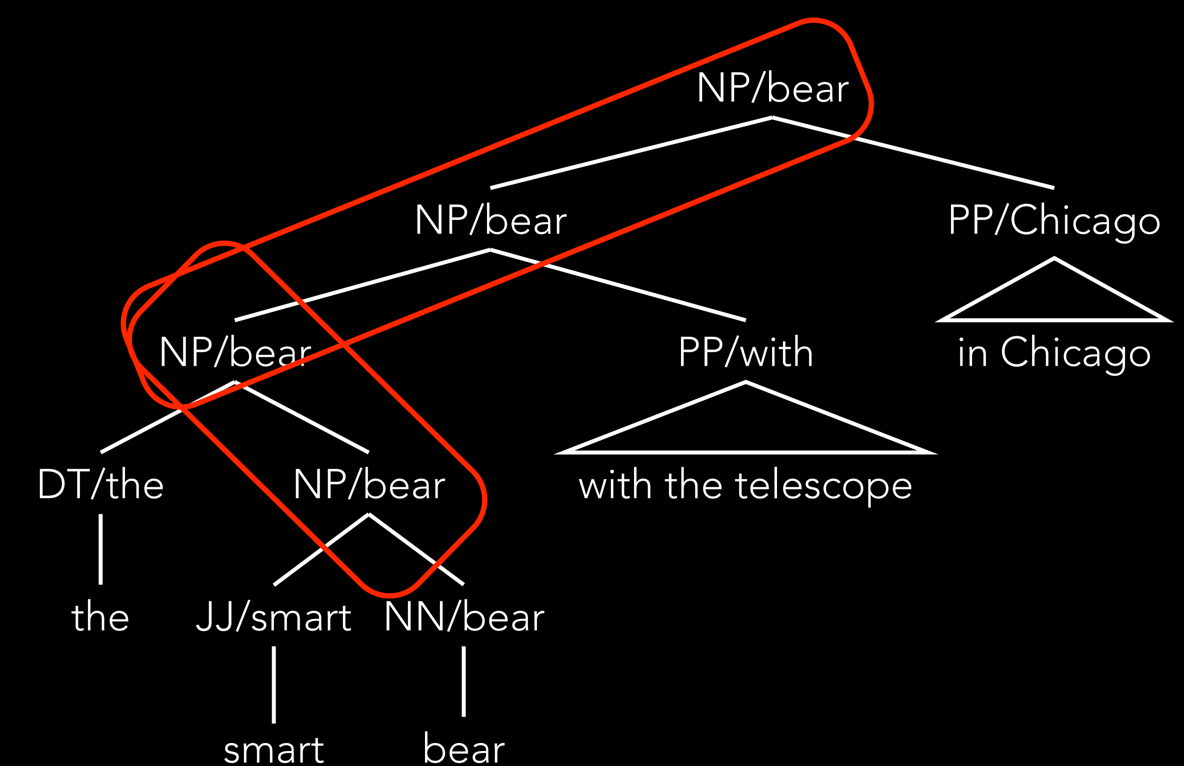
Lexicalization and Binarization

- Want to keep meaningful head at every point
- Need to make binarization head-aware



Left (for NP PP PP)

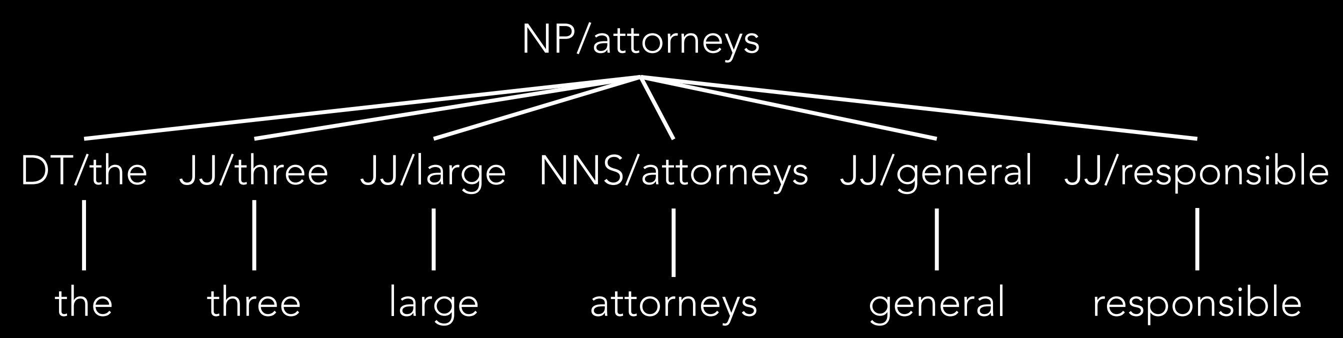
Right (for DT JJ NN)



Head binarization!

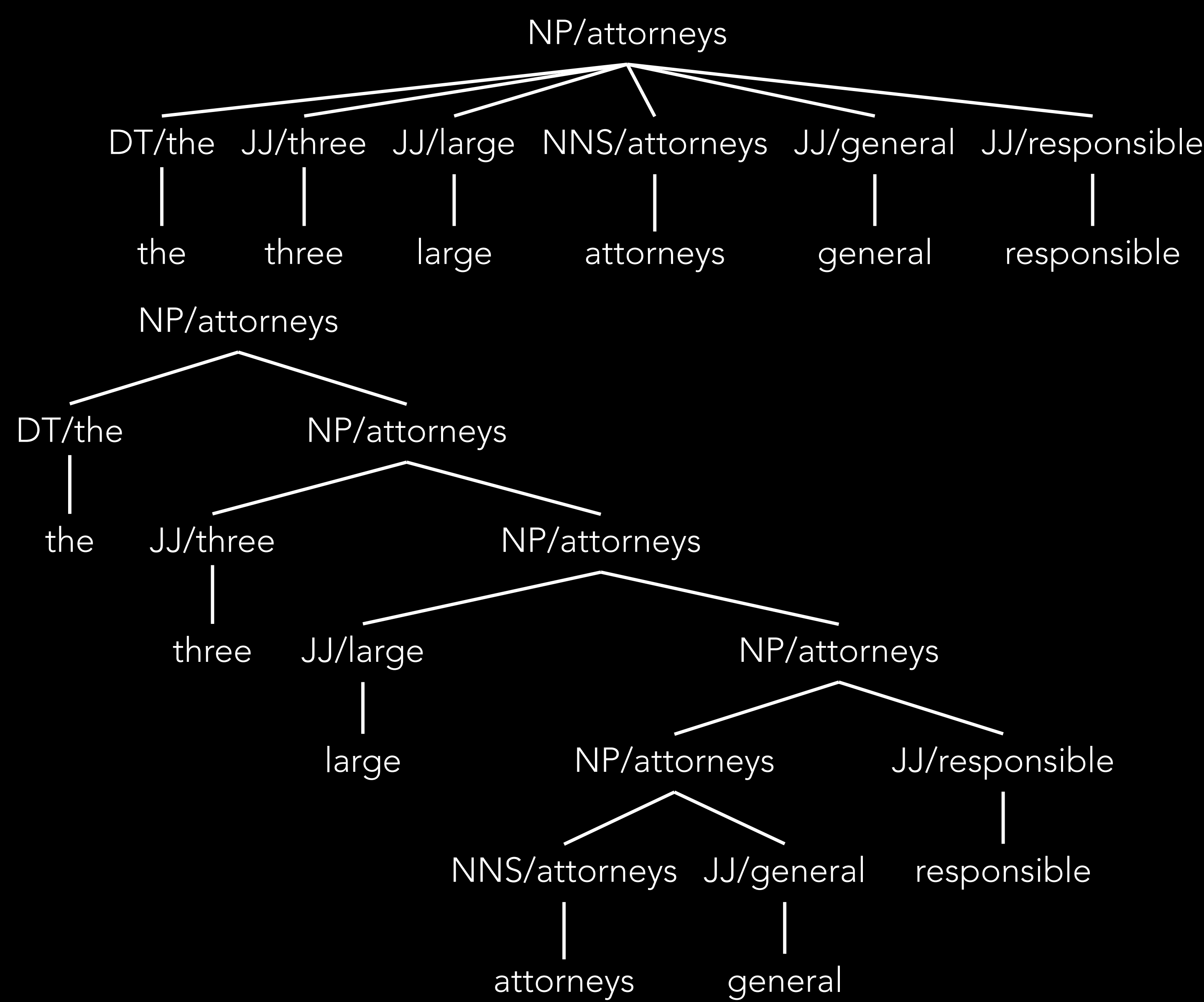
Lexicalization and Binarization

More complicated example



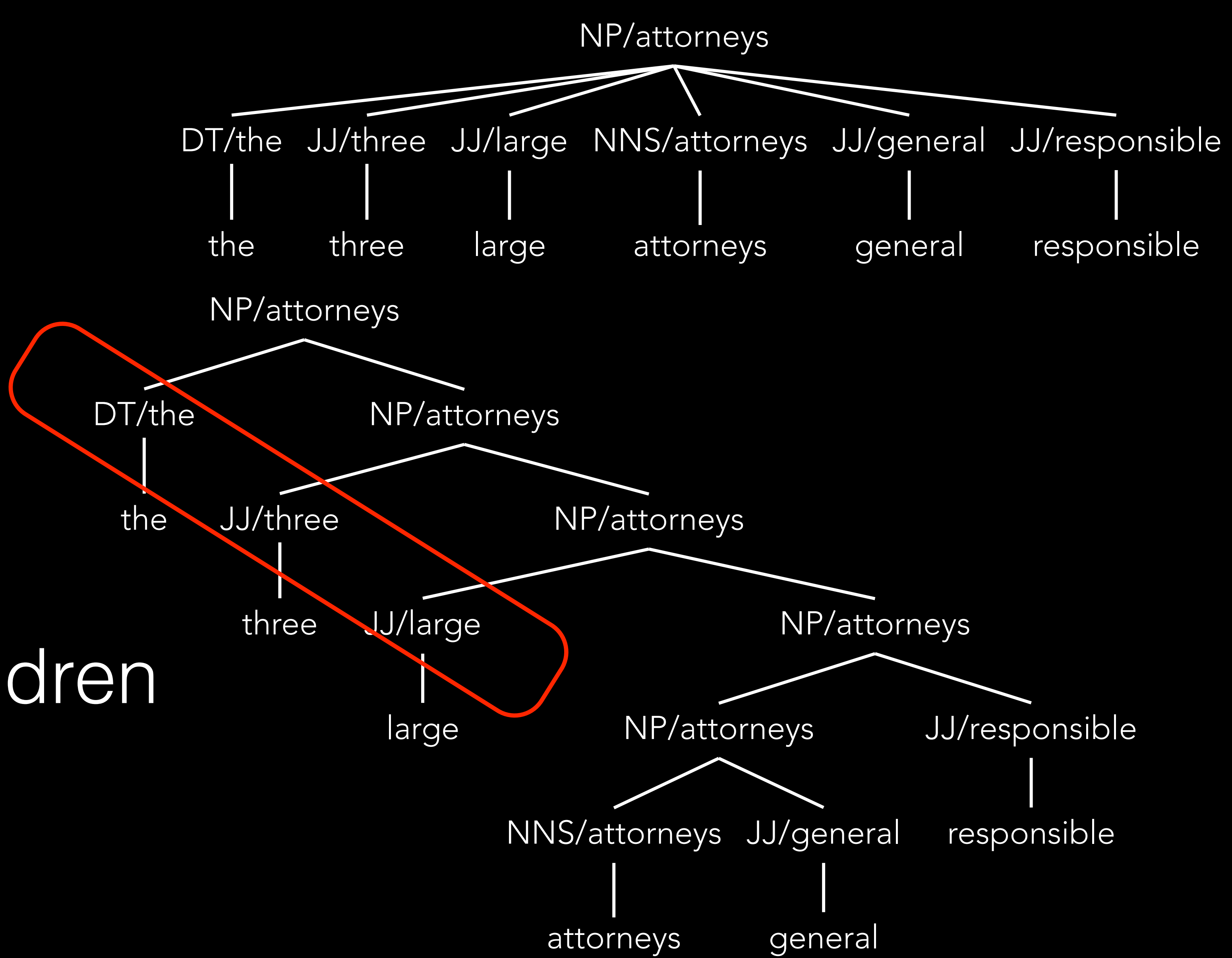
Lexicalization and Binarization

More complicated example



Lexicalization and Binarization

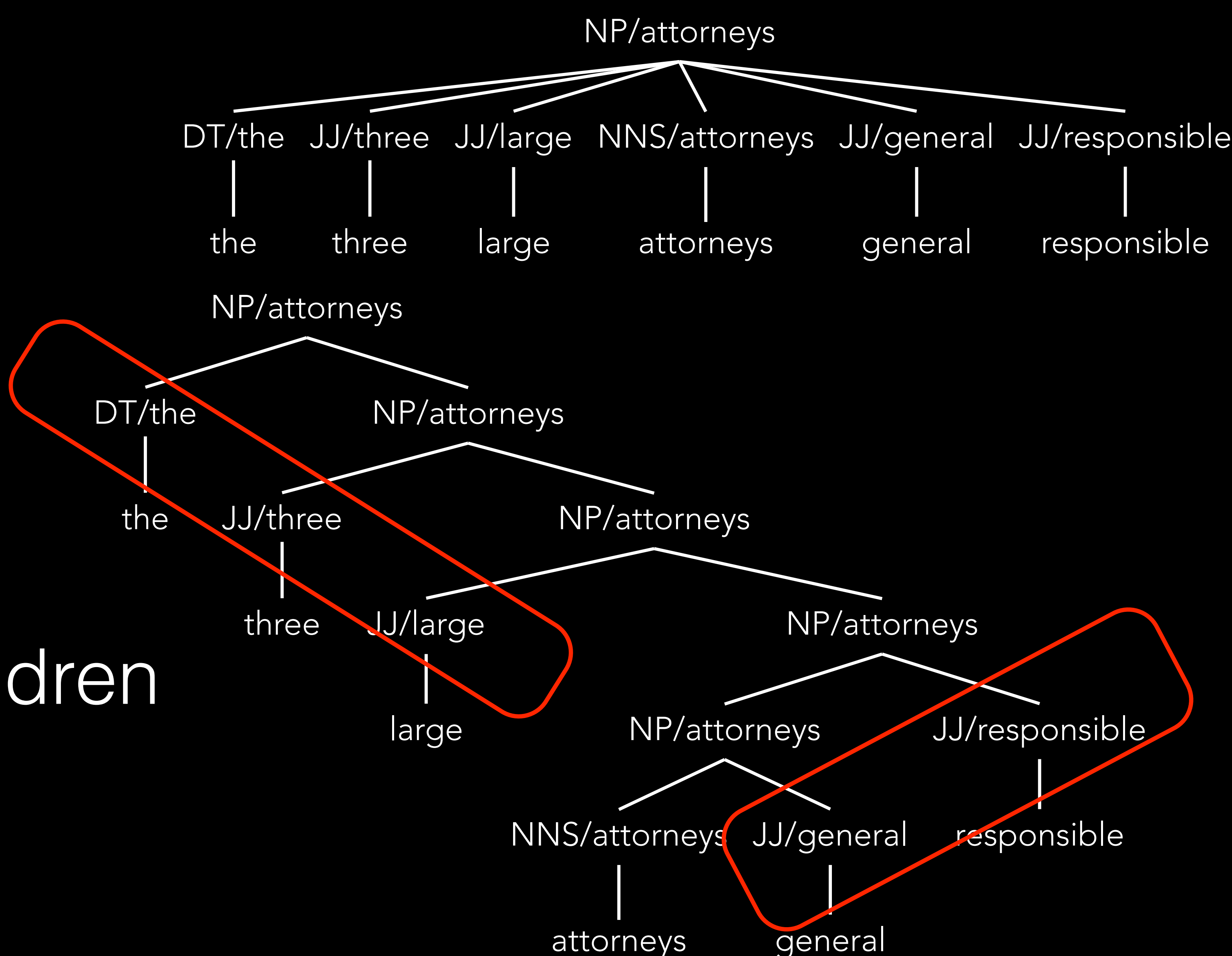
More complicated example



Left children

Lexicalization and Binarization

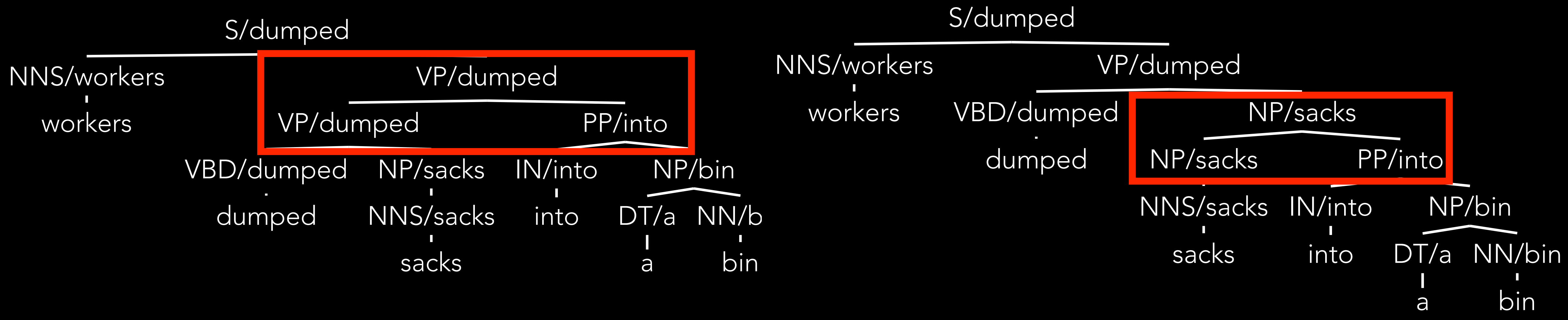
More complicated example



Left children

Right children

Head lexicalization

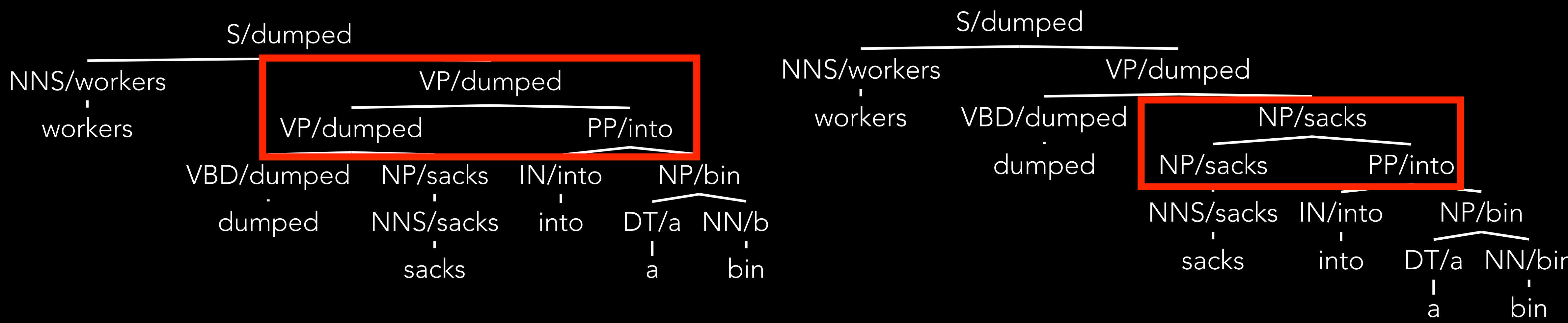


VP/dumped -> VP/dumped PP/into

more likely than

NP/sacks-> NP/sacks PP/into

Head lexicalization



VP/dumped -> VP/dumped PP/into

more likely than

NP/sacks-> NP/sacks PP/into

But now our parameter space is getting very large and our estimates are going to be very poor!

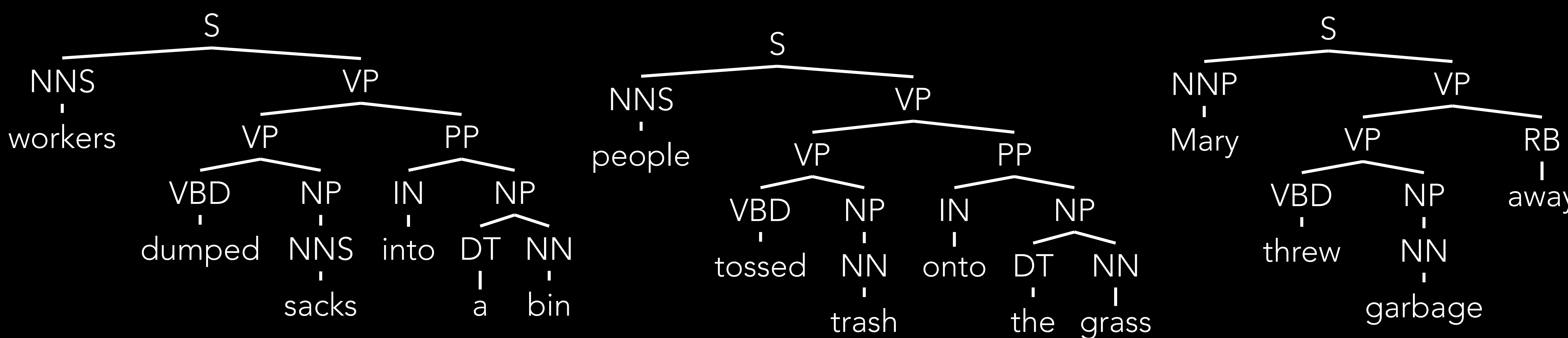
Head lexicalization

Head lexicalization

If your training data is

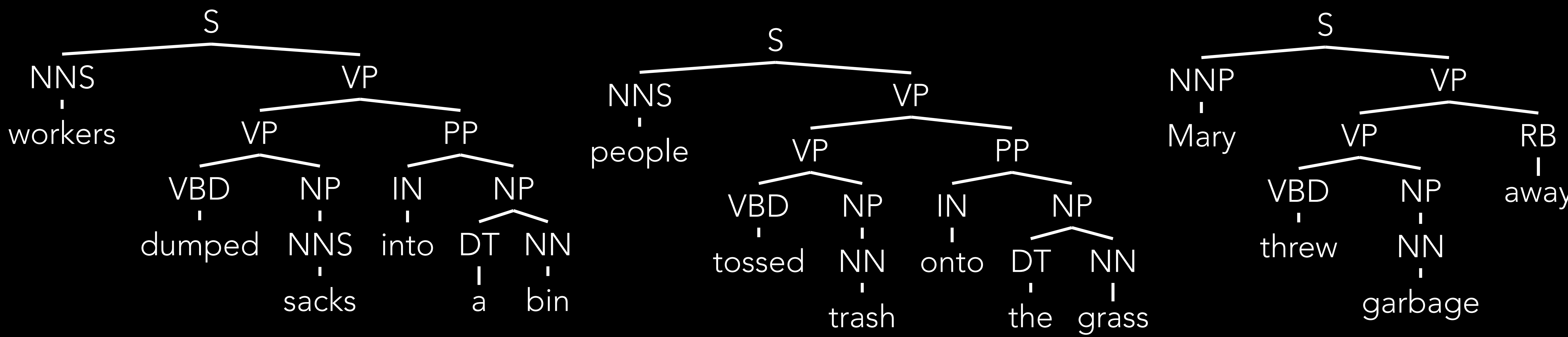
Head lexicalization

If your training data is



Head lexicalization

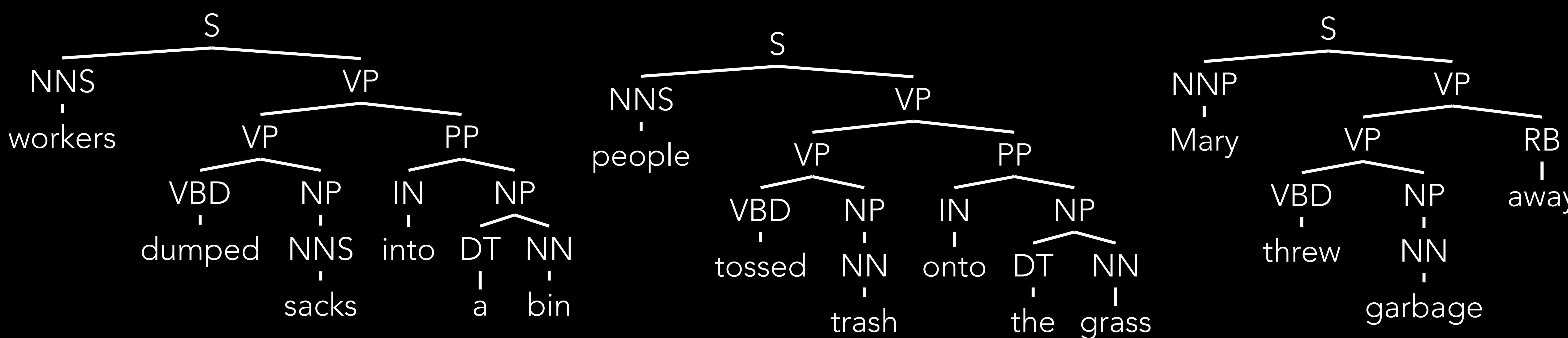
If your training data is



And you want to parse

Head lexicalization

If your training data is

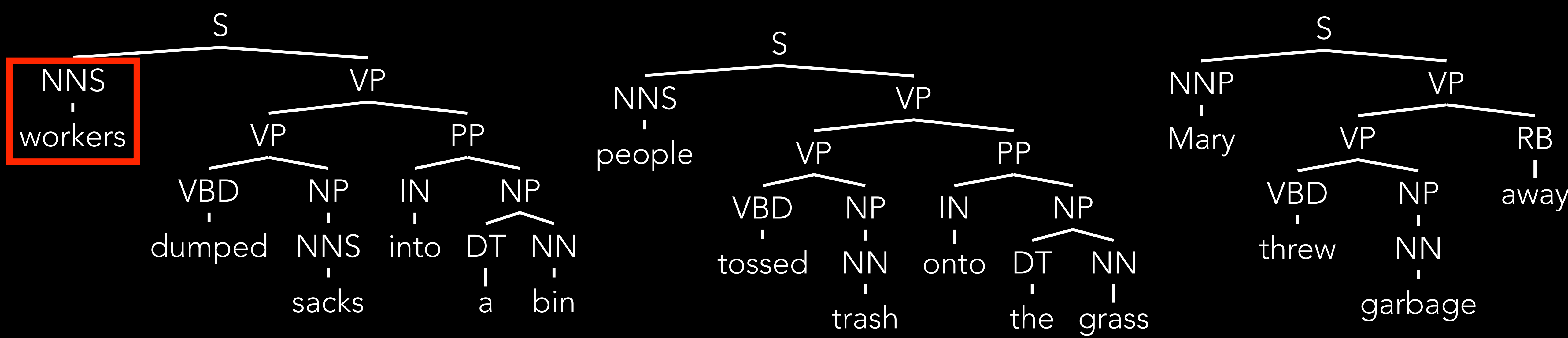


And you want to parse

workers threw trash
onto the grass

Head lexicalization

If your training data is

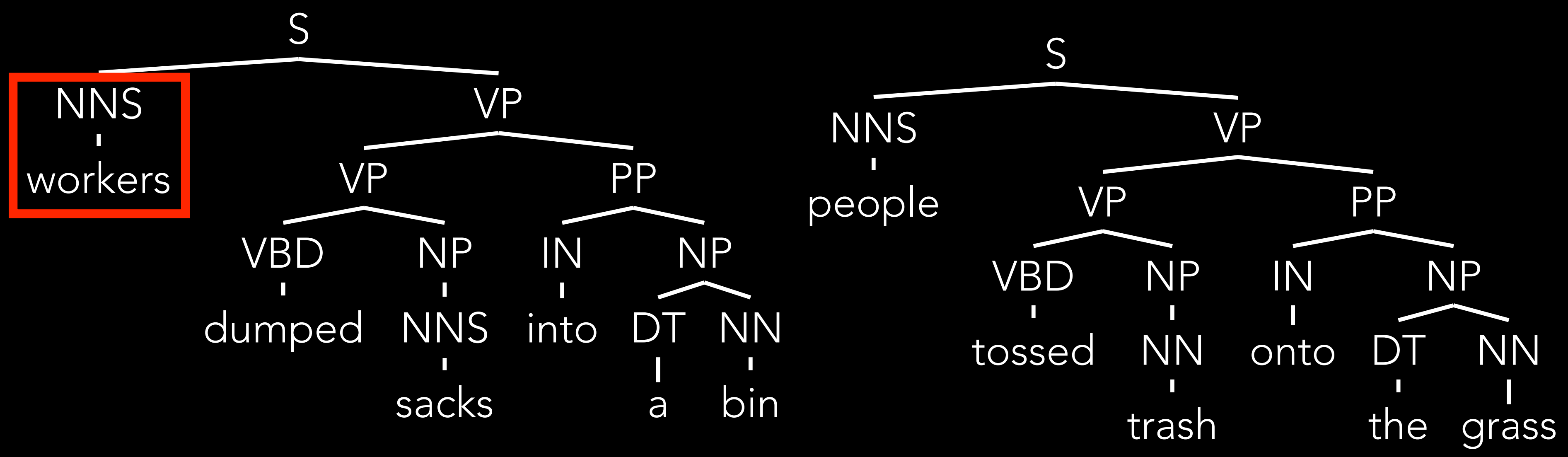


And you want to parse

workers threw trash
onto the grass

Head lexicalization

If your training data is

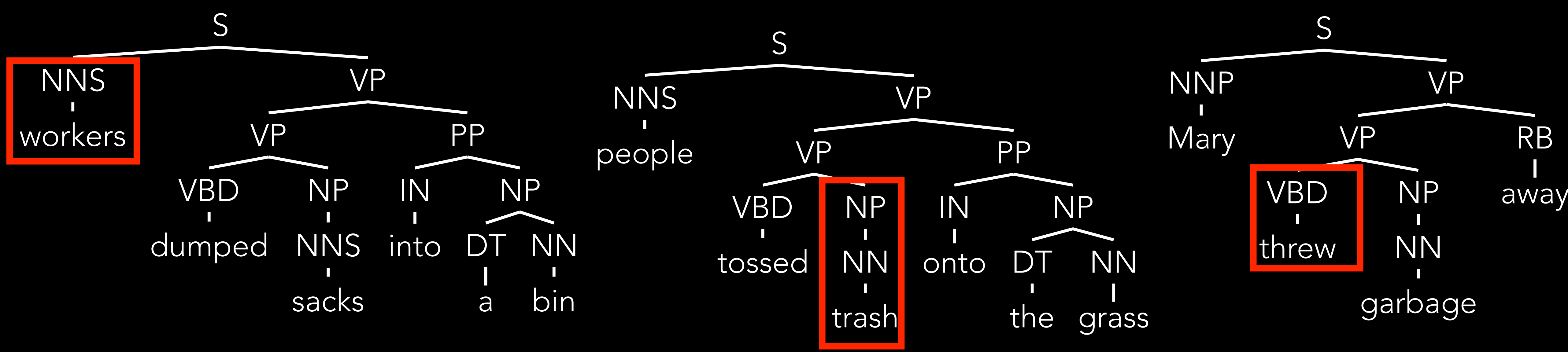


And you want to parse

workers threw trash
onto the grass

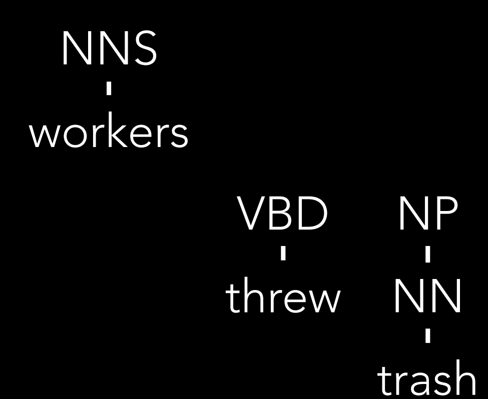
Head lexicalization

If your training data is



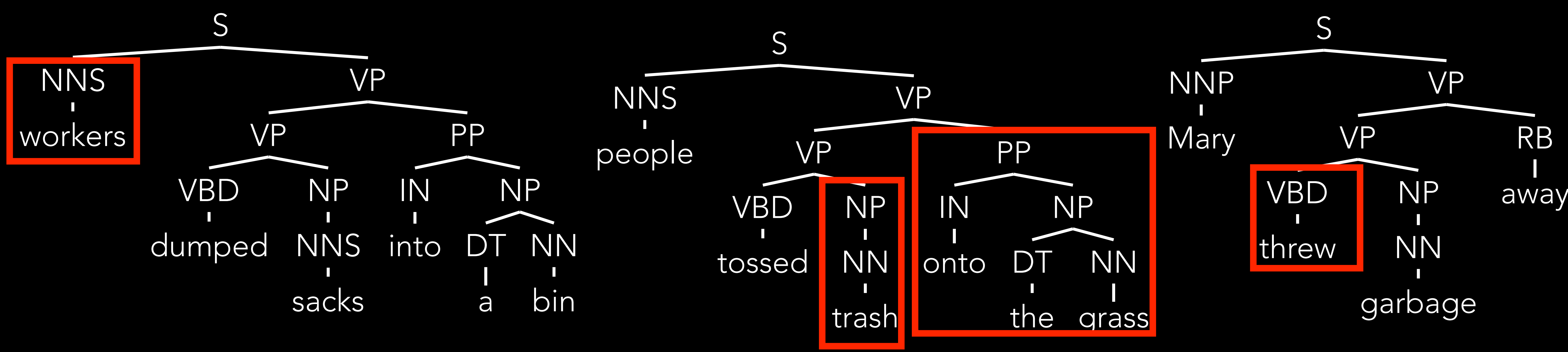
And you want to parse

workers threw trash onto the grass



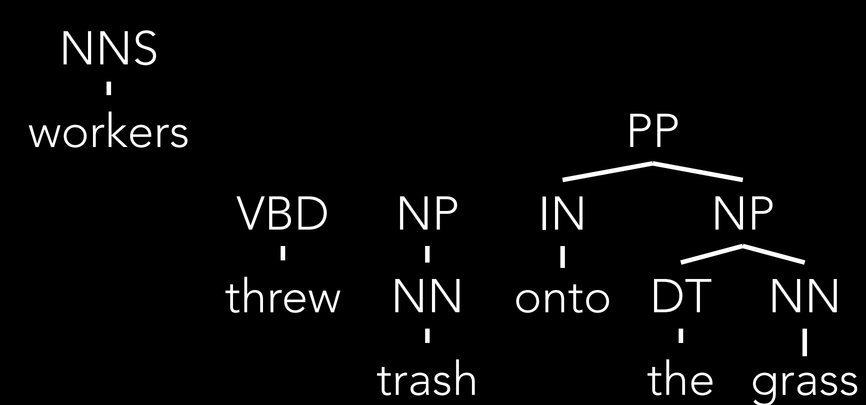
Head lexicalization

If your training data is



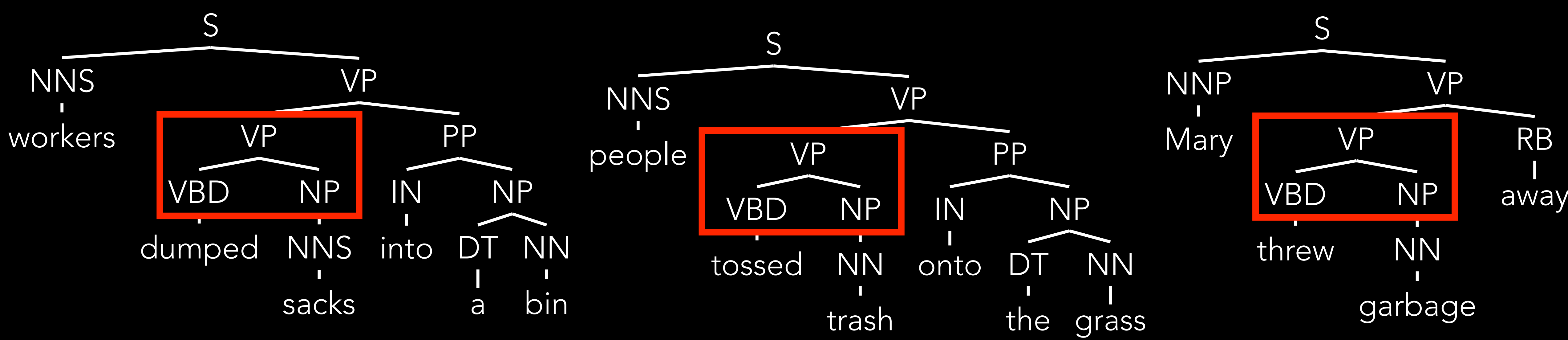
And you want to parse

workers threw trash
onto the grass



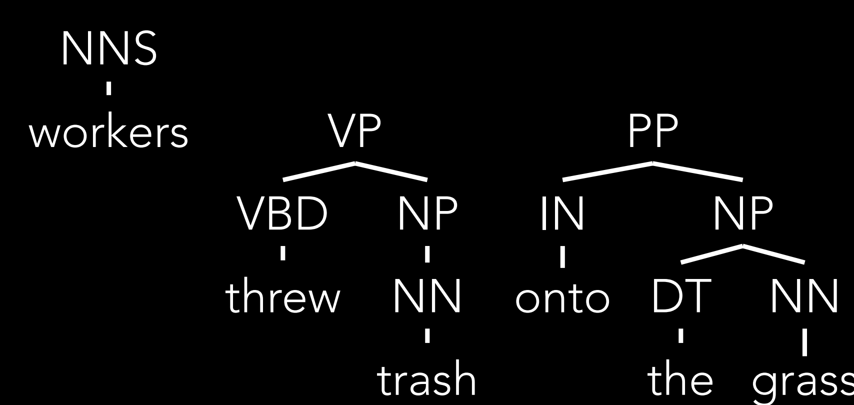
Head lexicalization

If your training data is



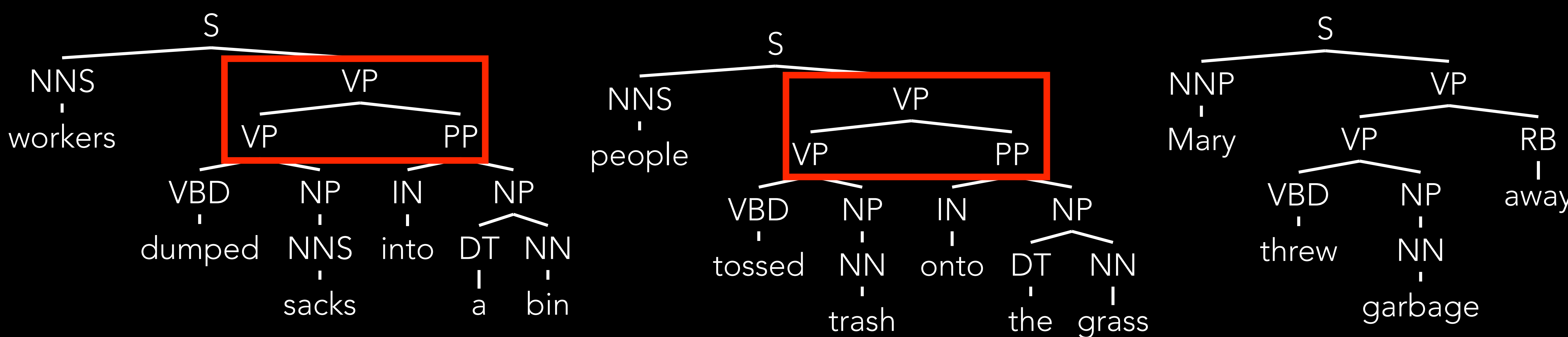
And you want to parse

workers threw trash
onto the grass



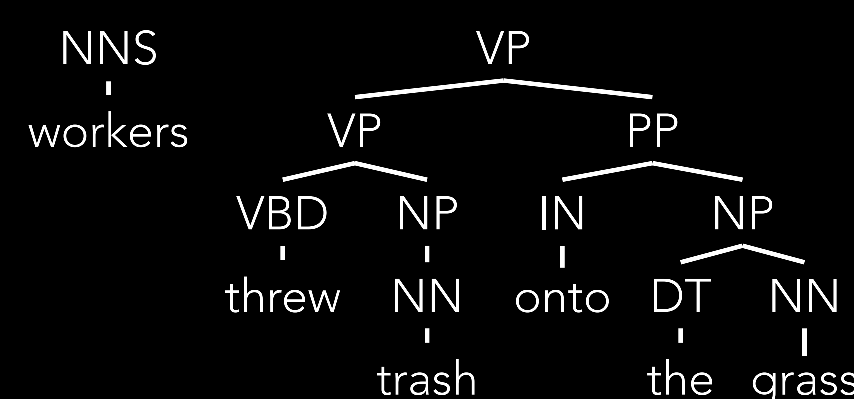
Head lexicalization

If your training data is



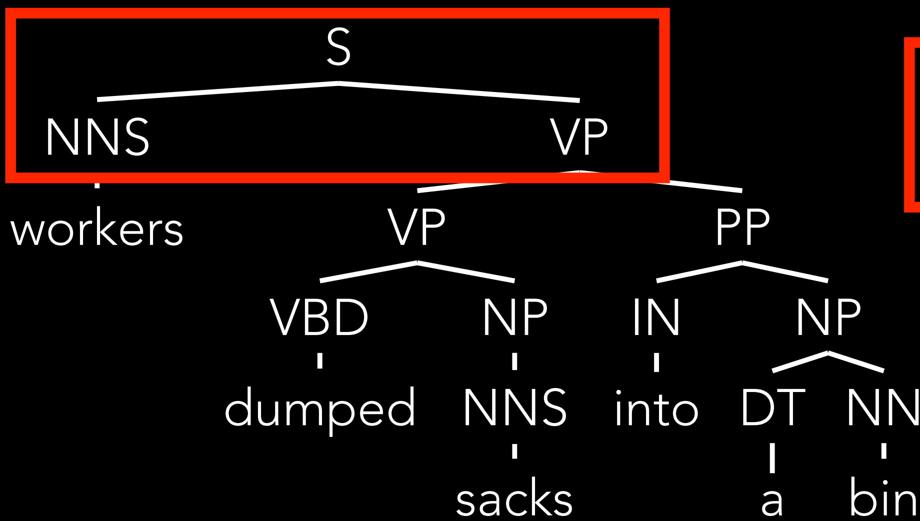
And you want to parse

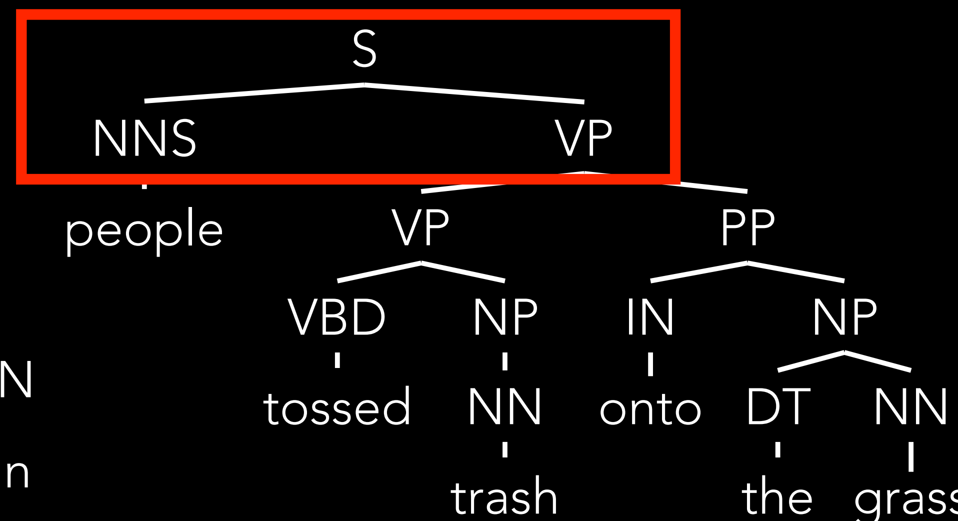
workers threw trash
onto the grass

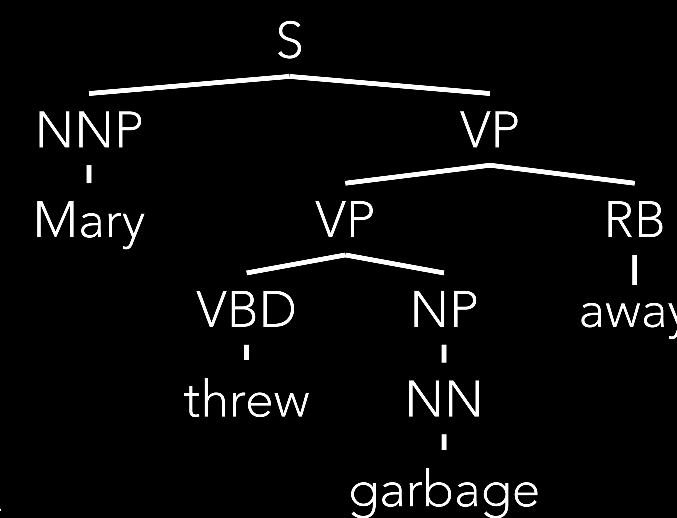


Head lexicalization

If your training data is

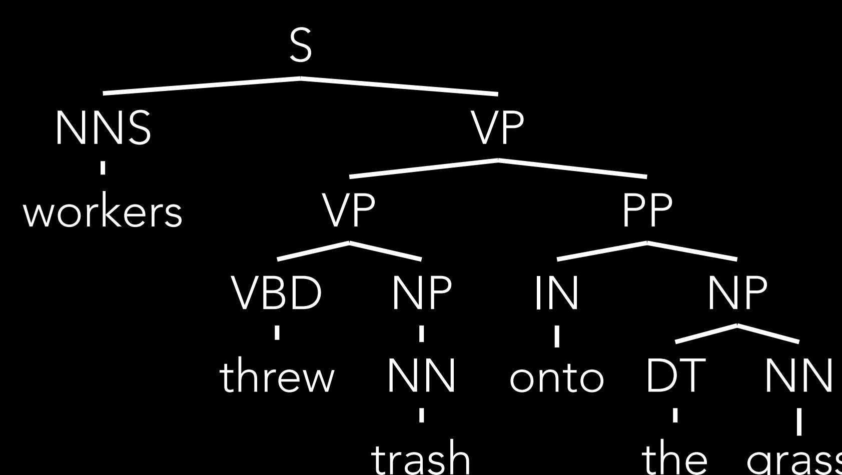






And you want to parse

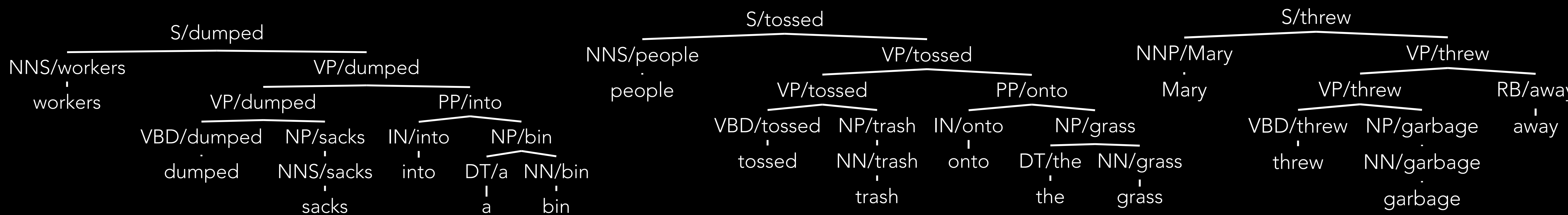
workers threw trash onto the grass



You can!

Head lexicalization

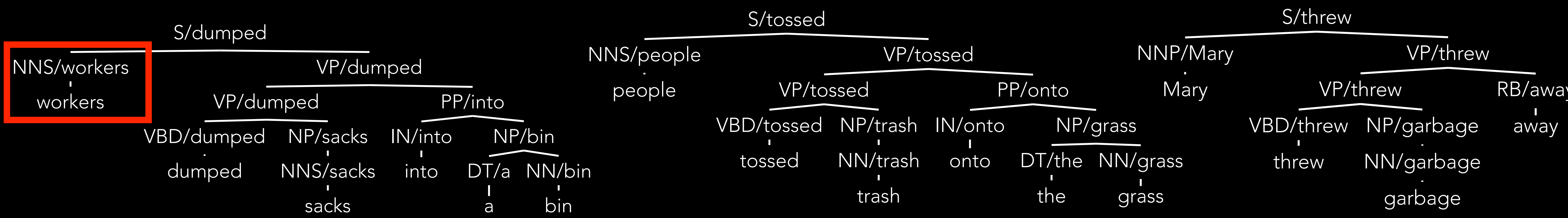
But if we lexicalize



workers threw trash
onto the grass

Head lexicalization

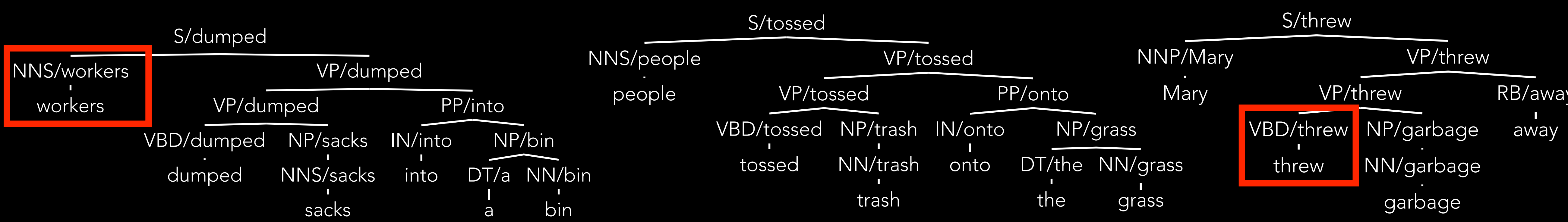
But if we lexicalize



workers threw trash
onto the grass

Head lexicalization

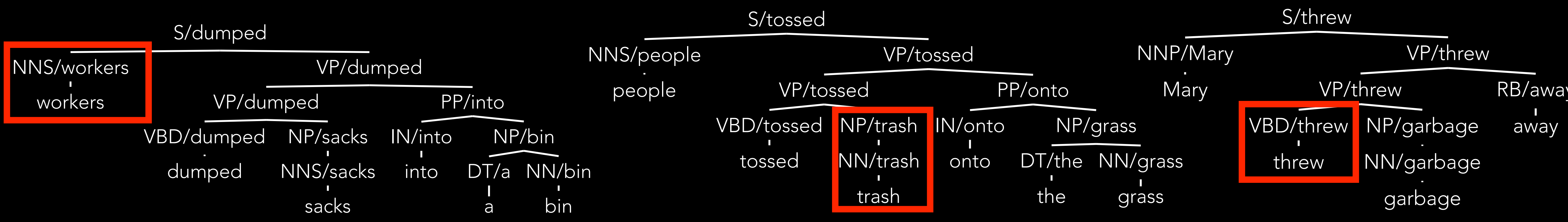
But if we lexicalize



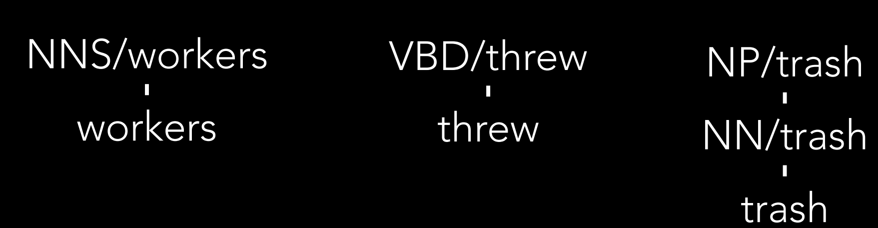
workers threw trash
onto the grass

Head lexicalization

But if we lexicalize

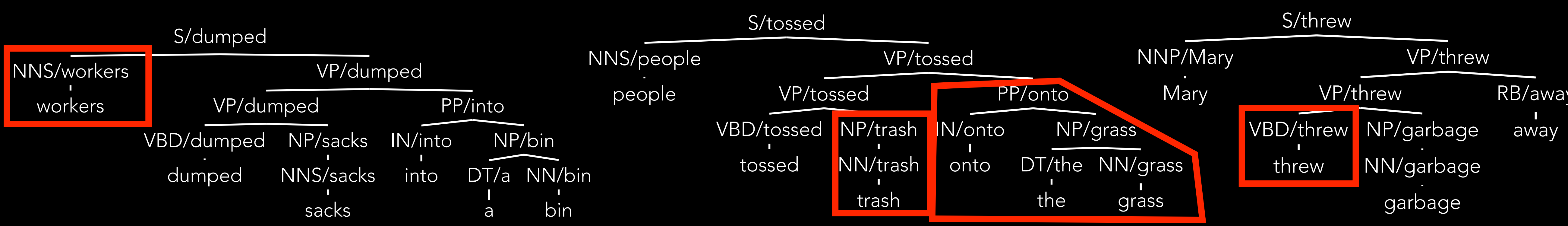


workers threw trash
onto the grass



Head lexicalization

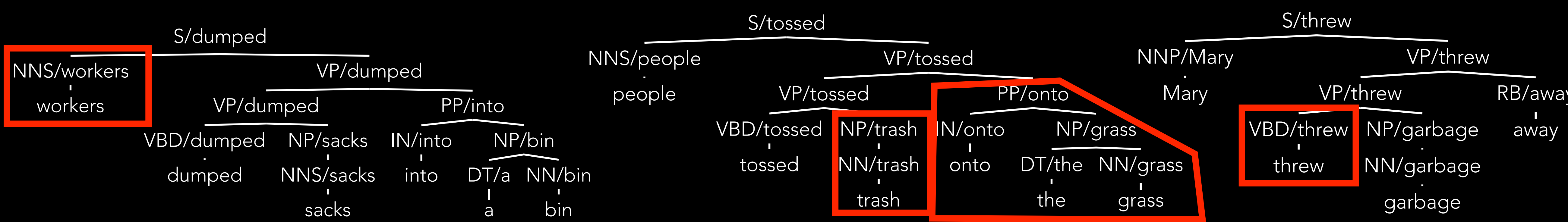
But if we lexicalize



workers threw trash
onto the grass

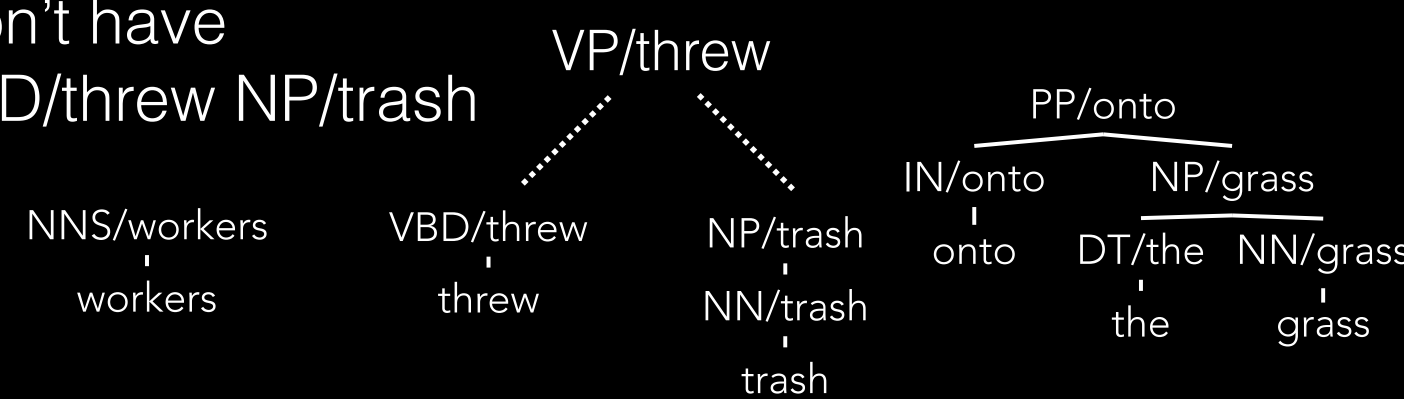
Head lexicalization

But if we lexicalize



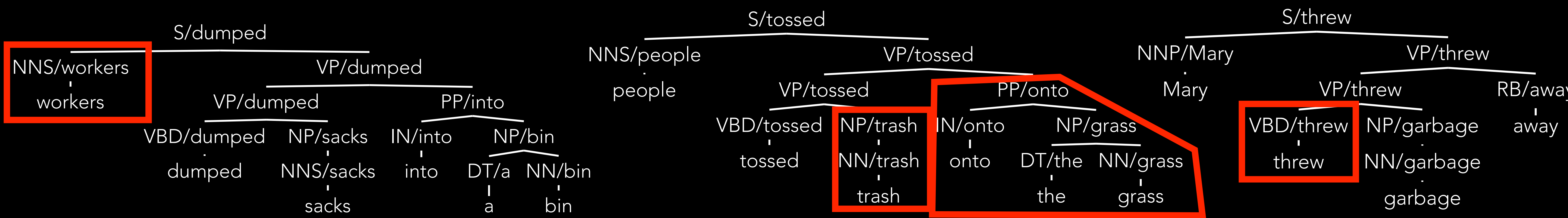
workers threw trash
onto the grass

We don’t have
VP/threw -> VBD/threw NP/trash



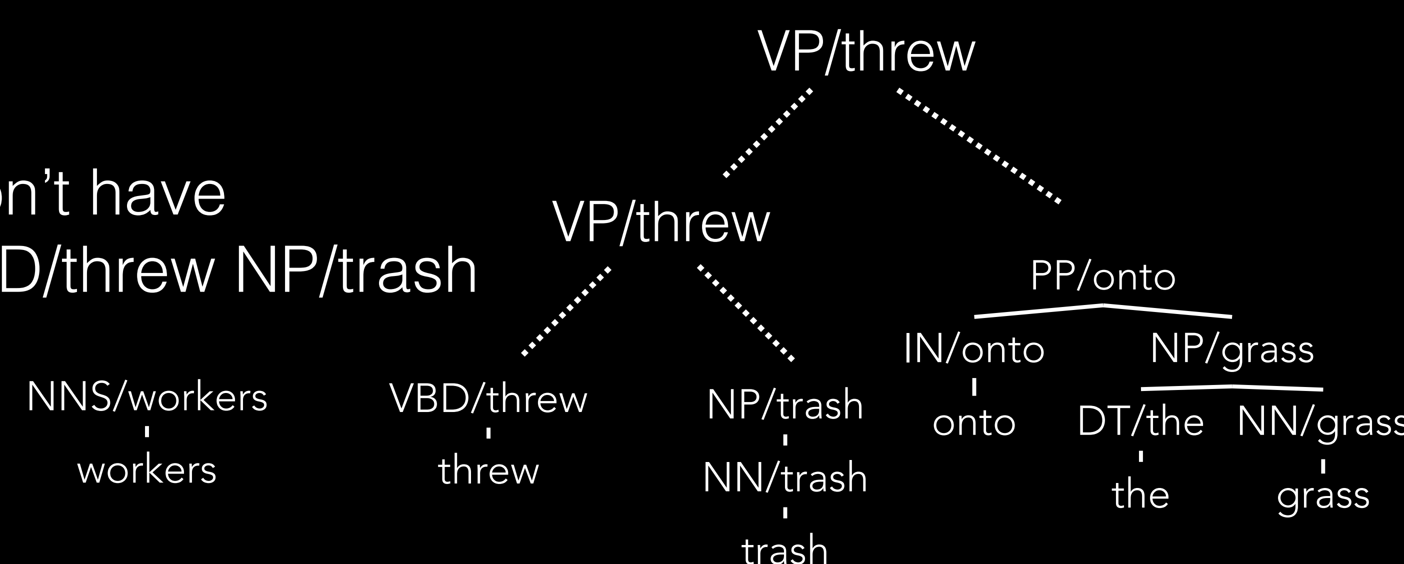
Head lexicalization

But if we lexicalize



workers threw trash
onto the grass

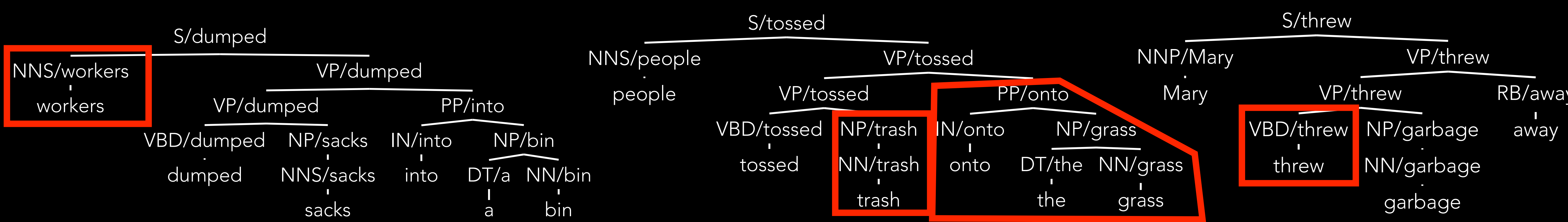
We don't have
VP/threw -> VP/threw PP/onto



We don't have
VP/threw -> VBD/threw NP/trash

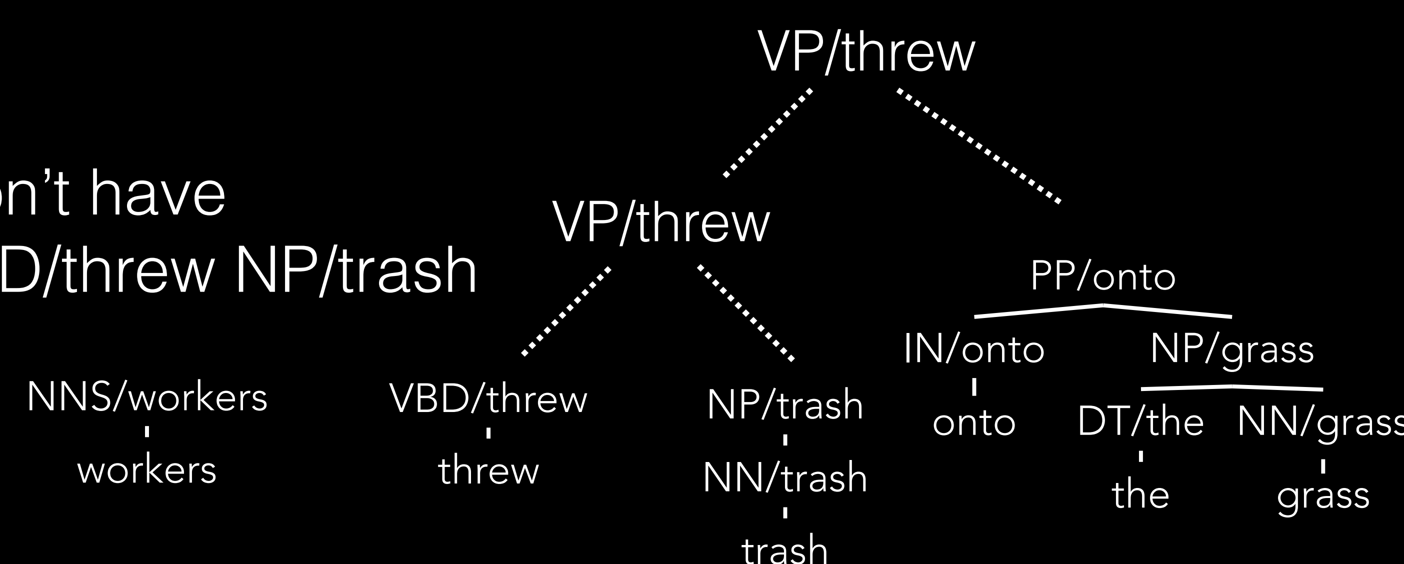
Head lexicalization

But if we lexicalize



workers threw trash
onto the grass

We don't have
VP/threw -> VP/threw PP/onto



We don't have
VP/threw -> VBD/threw NP/trash

Now we have a problem!

Smoothing Lexicalized PCFGs

We want to estimate

P(VP/threw -> VP/threw PP/onto | VP/threw)

Smoothing Lexicalized PCFGs

We want to estimate

P(VP/threw -> VP/threw PP/onto | VP/threw)

= P(VP -> <u>VP</u> PP, onto | VP, threw)

Smoothing Lexicalized PCFGs

We want to estimate

P(VP/threw -> VP/threw PP/onto | VP/threw)

= P(VP -> <u>VP</u> PP, onto | VP, threw)

or more generally

P(X -> YZ, m | X, h)

Smoothing Lexicalized PCFGs

We want to estimate

P(VP/threw -> VP/threw PP/onto | VP/threw)

= P(VP -> <u>VP</u> PP, onto | VP, threw)

or more generally

P(X -> YZ, m | X, h)

since X and h were already chosen, leaving the rule and the lexicalization of the non-head path to choose.

Smoothing Lexicalized PCFGs

First apply the chain rule

Smoothing Lexicalized PCFGs

First apply the chain rule

$$P(X \rightarrow YZ, m \mid X, h) = P(X \rightarrow YZ \mid X, h)\, P(m \mid X \rightarrow YZ, X, h)$$

Smoothing Lexicalized PCFGs

First apply the chain rule

$$P(X \text{ -> } YZ, m \mid X, h) = \underbrace{P(X \text{ -> } YZ \mid X, h)}_{\text{rule model}} P(m \mid X \text{ -> } YZ, X, h)$$

Smoothing Lexicalized PCFGs

First apply the chain rule

$$\text{P}(X \text{ -> } YZ, m \mid X, h) = \underbrace{\text{P}(X \text{ -> } YZ \mid X, h)}_{\text{rule model}} \; \underbrace{\text{P}(m \mid X \text{ -> } YZ, X, h)}_{\text{modifier model}}$$

Smoothing Lexicalized PCFGs

First apply the chain rule

$$P(X \text{ -> } YZ, m \mid X, h) = \underbrace{P(X \text{ -> } YZ \mid X, h)}_{\text{rule model}} \; \underbrace{P(m \mid X \text{ -> } YZ, X, h)}_{\text{modifier model}}$$

Now we consider a means for smoothing each of these components

Smoothing Lexicalized PCFGs

First apply the chain rule

$$\text{P(X -> YZ, m | X, h)} = \underbrace{\text{P(X -> YZ | X, h)}}_{\text{rule model}} \ \underbrace{\text{P(m | X -> YZ, X, h)}}_{\text{modifier model}}$$

Now we consider a means for smoothing each of these components

Interpolate P(X -> YZ | X, h) with P(X -> YZ | X)

Smoothing Lexicalized PCFGs

First apply the chain rule

$$P(X \text{ -> } YZ, m \mid X, h) = \underbrace{P(X \text{ -> } YZ \mid X, h)}_{\text{rule model}} \underbrace{P(m \mid X \text{ -> } YZ, X, h)}_{\text{modifier model}}$$

Now we consider a means for smoothing each of these components

Interpolate $P(X \text{ -> } YZ \mid X, h)$ with $P(X \text{ -> } YZ \mid X)$

Interpolate $P(m \mid X \text{ -> } YZ, X, h)$ with $P(m \mid X \text{ -> } YZ)$

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$P(r, m \mid X, h) = P(r \mid X, h)\, P(m \mid r, X, h)$$

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$\underbrace{P(r, m \mid X, h) = P(r \mid X, h)}_{\text{rule model}} P(m \mid r, X, h)$$

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$P(r, m \mid X, h) = \underbrace{P(r \mid X, h)}_{\text{rule model}} \underbrace{P(m \mid r, X, h)}_{\text{modifier model}}$$

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$P(r, m \mid X, h) = \underbrace{P(r \mid X, h)}_{\text{rule model}} \underbrace{P(m \mid r, X, h)}_{\text{modifier model}}$$

Rule Model

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$P(r, m \mid X, h) = \underbrace{P(r \mid X, h)}_{\text{rule model}} \underbrace{P(m \mid r, X, h)}_{\text{modifier model}}$$

Rule Model

$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$P(r, m \mid X, h) = \underbrace{P(r \mid X, h)}_{\text{rule model}} \underbrace{P(m \mid r, X, h)}_{\text{modifier model}}$$

Rule Model

$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

$$P_{ML}(r \mid X, h) = \frac{\text{count}(r, h)}{\text{count}(X, h)}$$

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$P(r, m \mid X, h) = \underbrace{P(r \mid X, h)}_{\text{rule model}} \underbrace{P(m \mid r, X, h)}_{\text{modifier model}}$$

Rule Model

$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

$$P_{ML}(r \mid X, h) = \frac{\text{count}(r, h)}{\text{count}(X, h)}$$

X/h ->Y/h Z/*

X/h

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$P(r, m \mid X, h) = \underbrace{P(r \mid X, h)}_{\text{rule model}} \underbrace{P(m \mid r, X, h)}_{\text{modifier model}}$$

Rule Model

$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

$$P_{ML}(r \mid X, h) = \frac{\text{count}(r, h)}{\text{count}(X, h)}$$

X/h ->Y/h Z/*

X/h

$$P_{ML}(r \mid X) = \frac{\text{count}(r)}{\text{count}(X)}$$

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$P(r, m \mid X, h) = \underbrace{P(r \mid X, h)}_{\text{rule model}} \underbrace{P(m \mid r, X, h)}_{\text{modifier model}}$$

Rule Model

$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

$$P_{ML}(r \mid X, h) = \frac{\text{count}(r, h)}{\text{count}(X, h)} \qquad \frac{X/h \to Y/h\ Z/*}{X/h}$$

$$P_{ML}(r \mid X) = \frac{\text{count}(r)}{\text{count}(X)} \qquad \frac{X/* \to Y/*\ Z/*}{X/*}$$

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$P(r, m \mid X, h) = \underbrace{P(r \mid X, h)}_{\text{rule model}}\ \underbrace{P(m \mid r, X, h)}_{\text{modifier model}}$$

Rule Model

$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

$$P_{ML}(r \mid X, h) = \frac{\text{count}(r, h)}{\text{count}(X, h)} \qquad \frac{X/h \rightarrow Y/h\ Z/*}{X/h}$$

$$P_{ML}(r \mid X) = \frac{\text{count}(r)}{\text{count}(X)} \qquad \frac{X/* \rightarrow Y/*\ Z/*}{X/*}$$

Vanilla PCFG!

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$P(r, m \mid X, h) = \underbrace{P(r \mid X, h)}_{\text{rule model}} \underbrace{P(m \mid r, X, h)}_{\text{modifier model}}$$

Modifier Model

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$P(r, m \mid X, h) = \underbrace{P(r \mid X, h)}_{\text{rule model}} \underbrace{P(m \mid r, X, h)}_{\text{modifier model}}$$

Modifier Model

$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$P(r, m \mid X, h) = \underbrace{P(r \mid X, h)}_{\text{rule model}} \underbrace{P(m \mid r, X, h)}_{\text{modifier model}}$$

Modifier Model

$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

$$P_{ML}(m \mid r, X, h) = \frac{\text{count}(m, r, X, h)}{\text{count}(r, X, h)}$$

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$P(r, m \mid X, h) = \underbrace{P(r \mid X, h)}_{\text{rule model}} \underbrace{P(m \mid r, X, h)}_{\text{modifier model}}$$

Modifier Model

$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

$$P_{ML}(m \mid r, X, h) = \frac{\text{count}(m, r, X, h)}{\text{count}(r, X, h)}$$

X/h ->Y/h Z/m

X/h ->Y/h Z/*

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$P(r, m \mid X, h) = \underbrace{P(r \mid X, h)}_{\text{rule model}} \underbrace{P(m \mid r, X, h)}_{\text{modifier model}}$$

Modifier Model

$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

$$P_{ML}(m \mid r, X, h) = \frac{\text{count}(m, r, X, h)}{\text{count}(r, X, h)}$$

X/h ->Y/h Z/m

X/h ->Y/h Z/*

$$P_{ML}(m \mid r) = \frac{\text{count}(m, r)}{\text{count}(r)}$$

Smoothing Lexicalized PCFGs

Let r be a vanilla rule (e.g. X -> YZ), X be its root symbol, h be its head, and m be its modifier

$$P(r, m \mid X, h) = \underbrace{P(r \mid X, h)}_{\text{rule model}} \underbrace{P(m \mid r, X, h)}_{\text{modifier model}}$$

Modifier Model

$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

$$P_{ML}(m \mid r, X, h) = \frac{\text{count}(m, r, X, h)}{\text{count}(r, X, h)}$$

X/h ->Y/h Z/m

X/h ->Y/h Z/*

$$P_{ML}(m \mid r) = \frac{\text{count}(m, r)}{\text{count}(r)}$$

X/* ->Y/* Z/m

X/* ->Y/* Z/*

Head lexicalization

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Head lexicalization

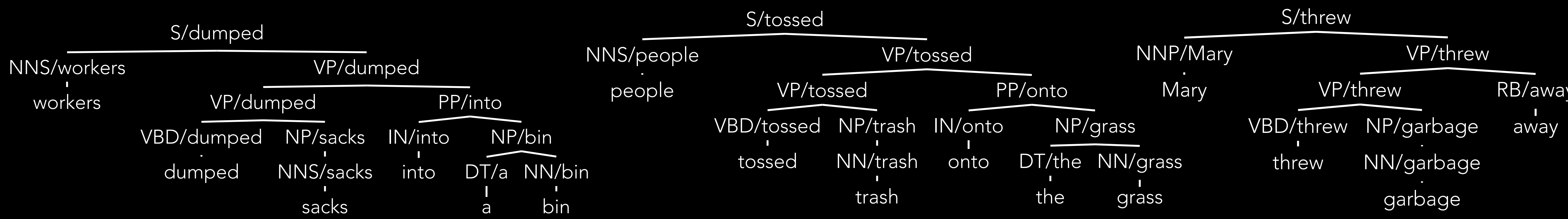
By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

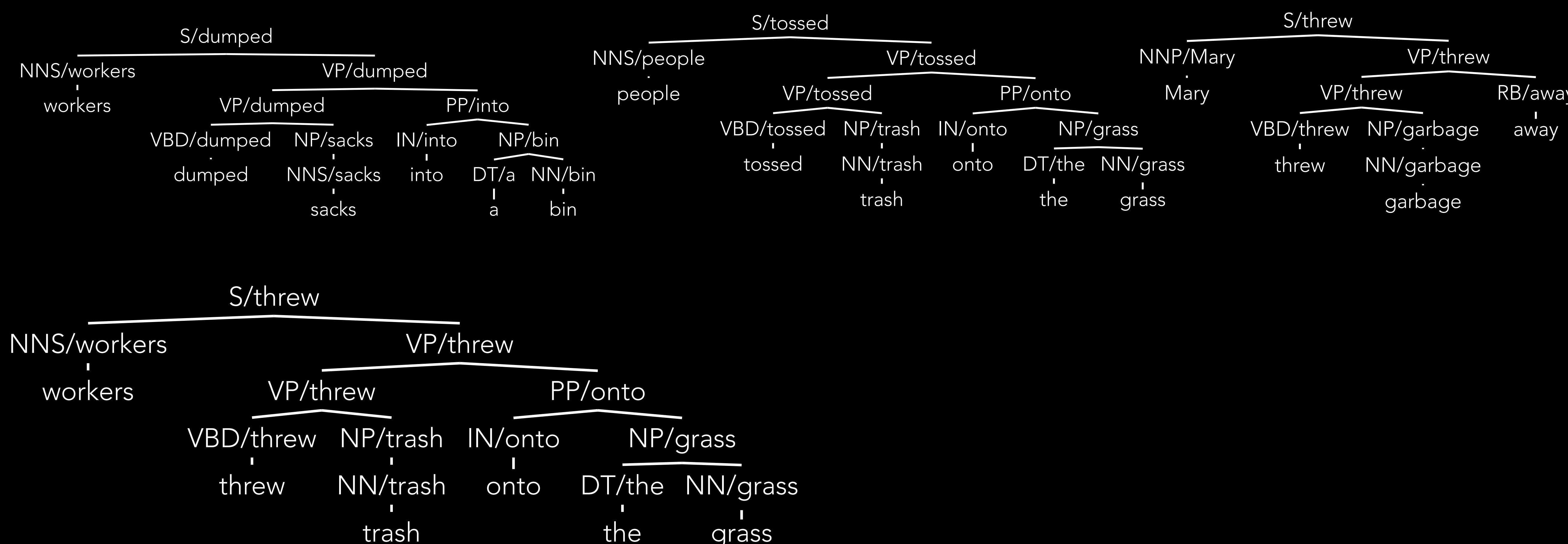
Assume we saw each training tree 20 times



Head lexicalization

By decomposing we can estimate unseen lexicalized rules

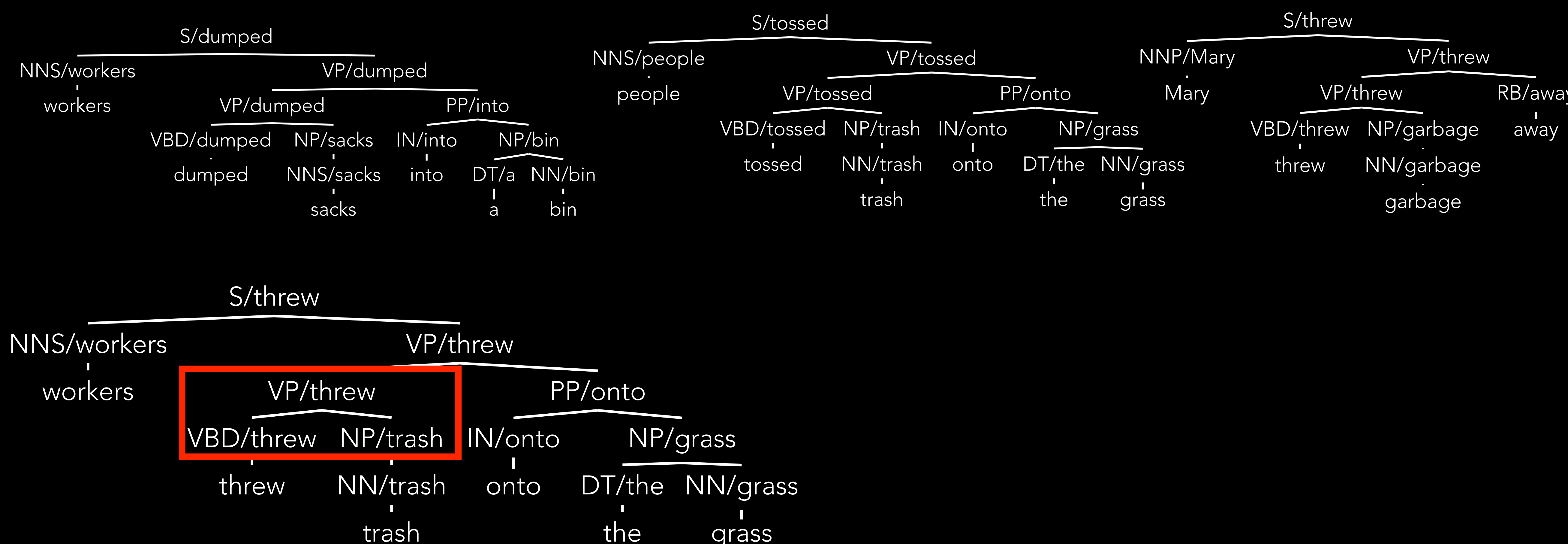
Assume we saw each training tree 20 times



Head lexicalization

By decomposing we can estimate unseen lexicalized rules

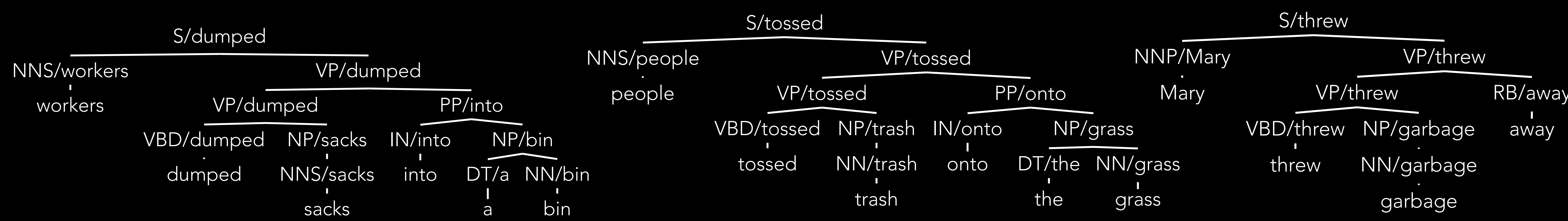
Assume we saw each training tree 20 times

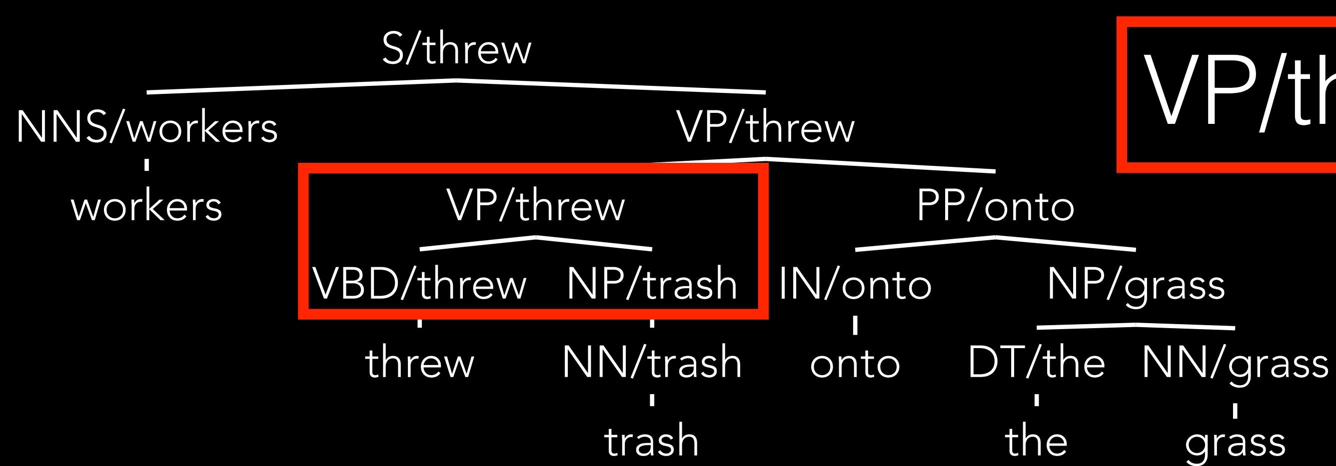


Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times



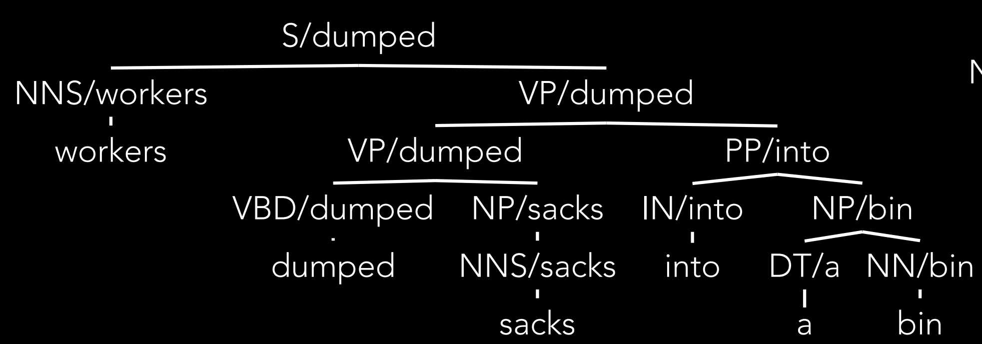


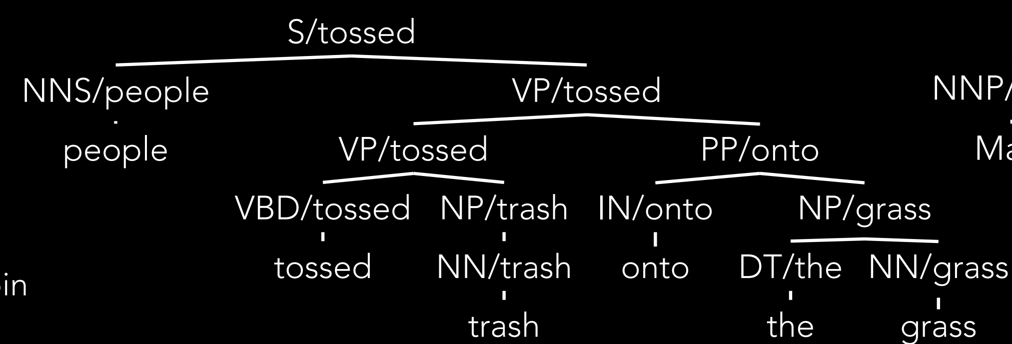
VP/threw -> VBD/threw NP/trash

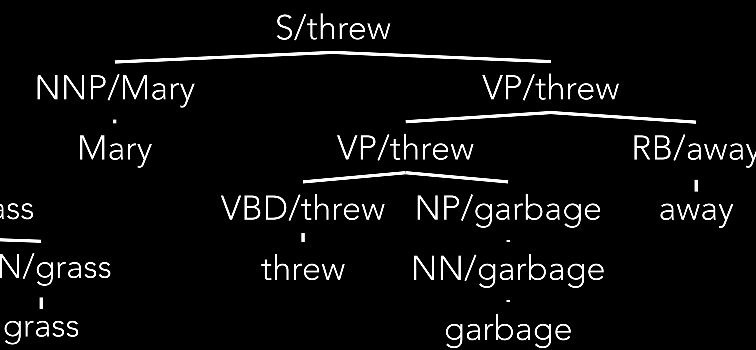
Head lexicalization

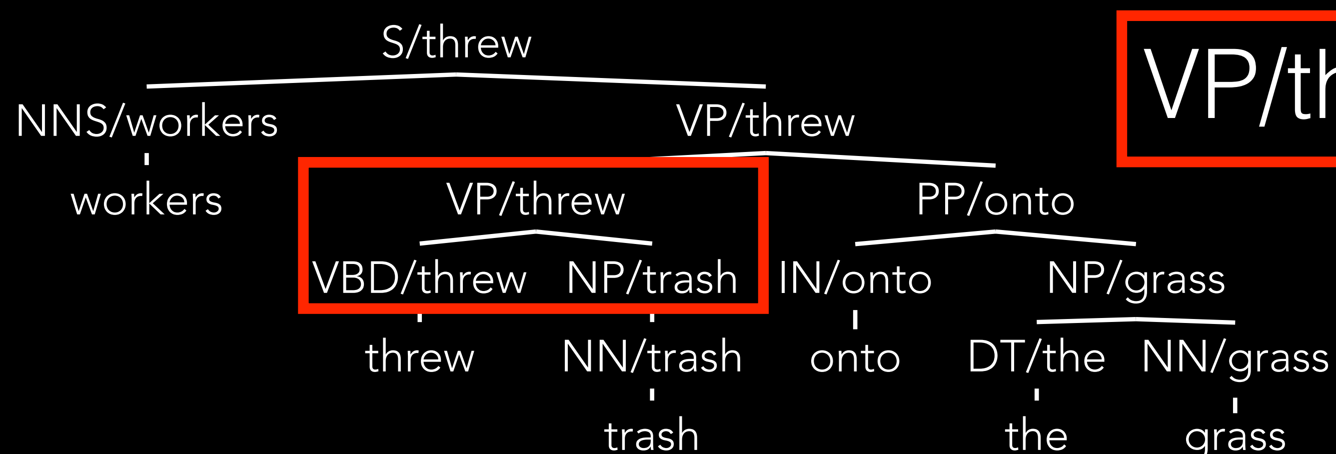
By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times









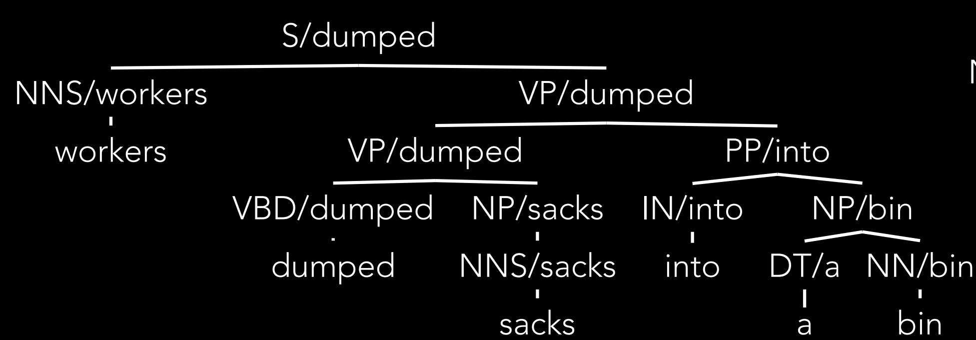
VP/threw -> VBD/threw NP/trash

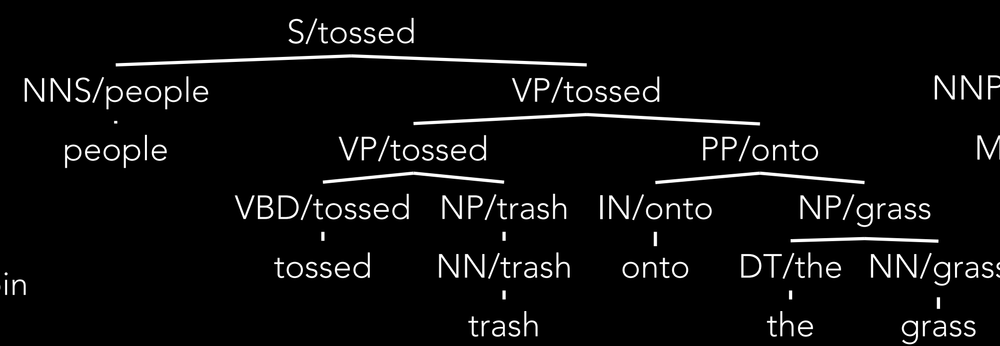
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

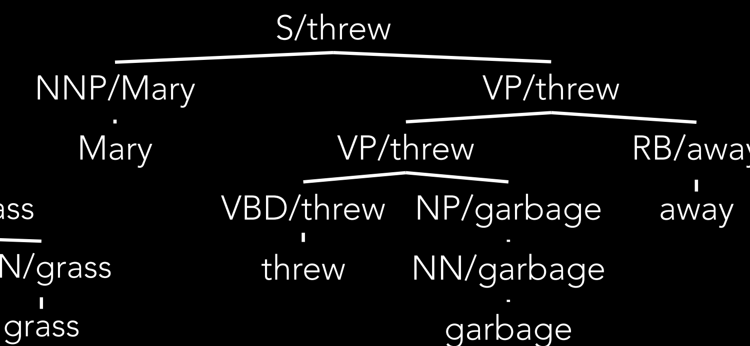
Head lexicalization

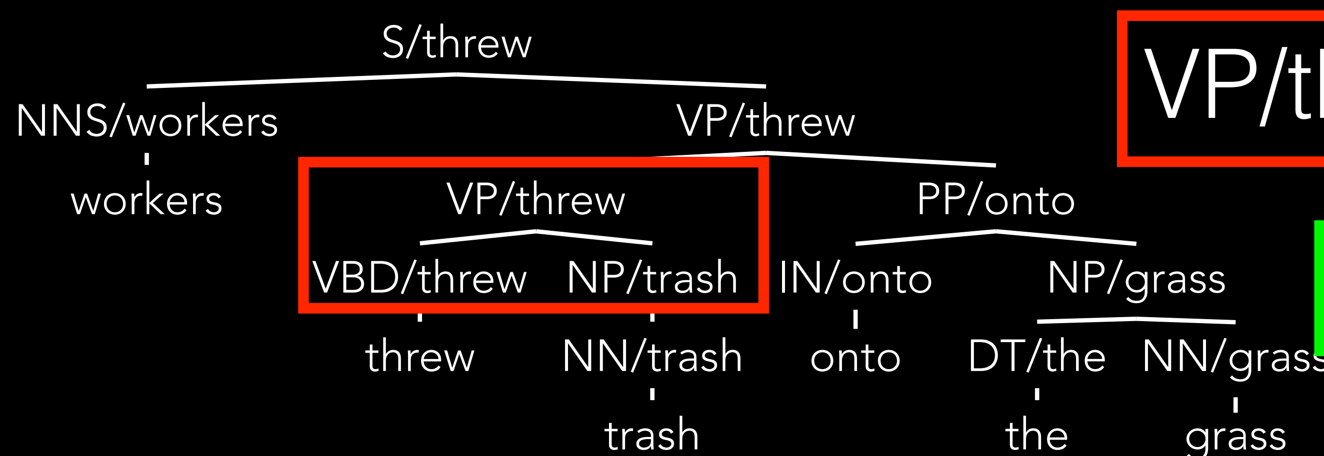
By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times









VP/threw -> VBD/threw NP/trash

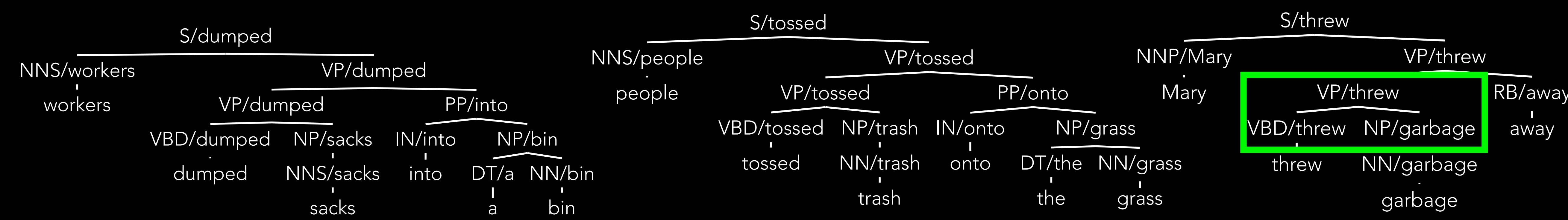
VP/threw -> VBD/threw NP

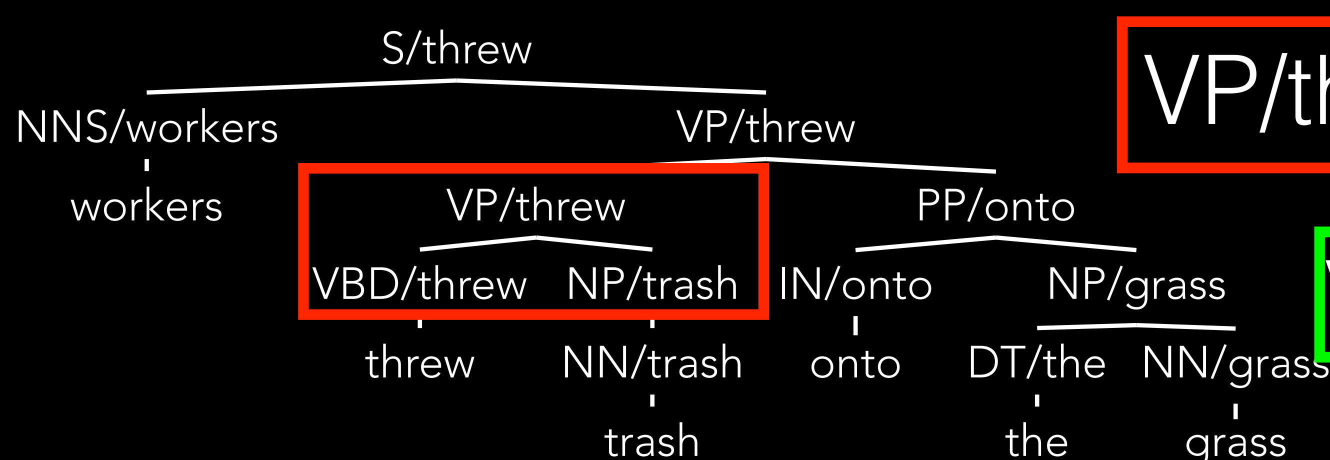
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

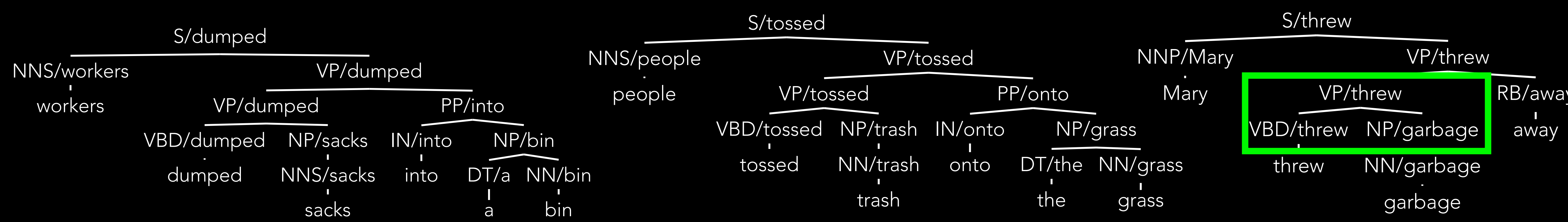
VP/threw -> VBD/threw NP

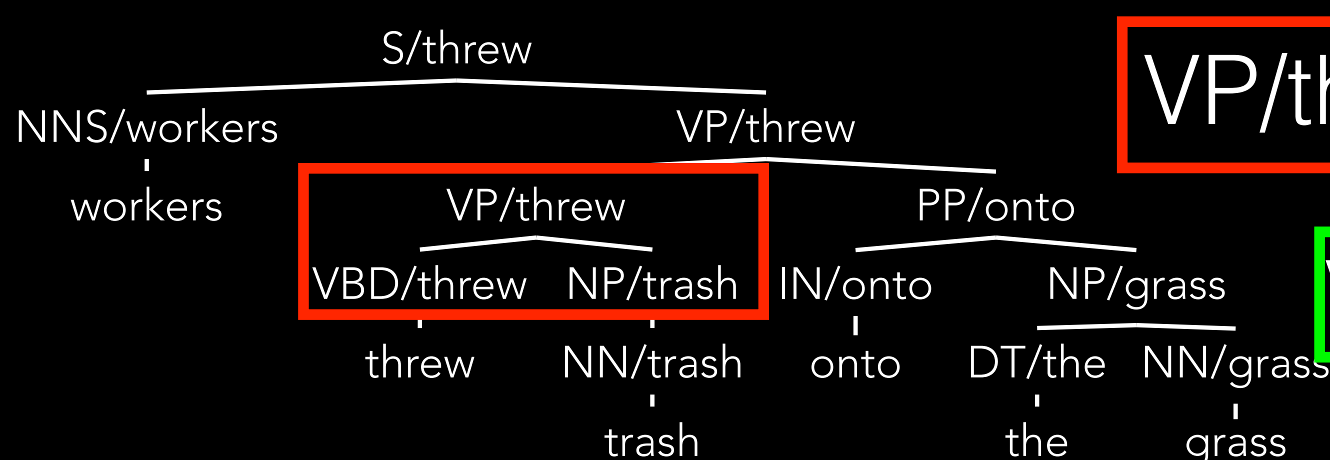
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP/threw -> VBD/threw NP

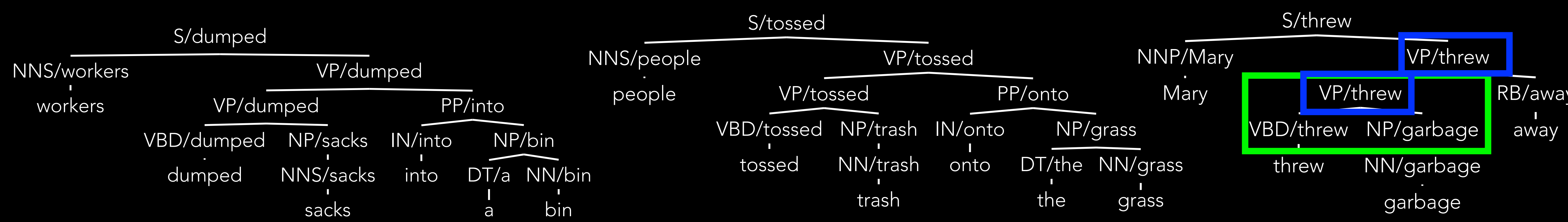
VP/threw -> …

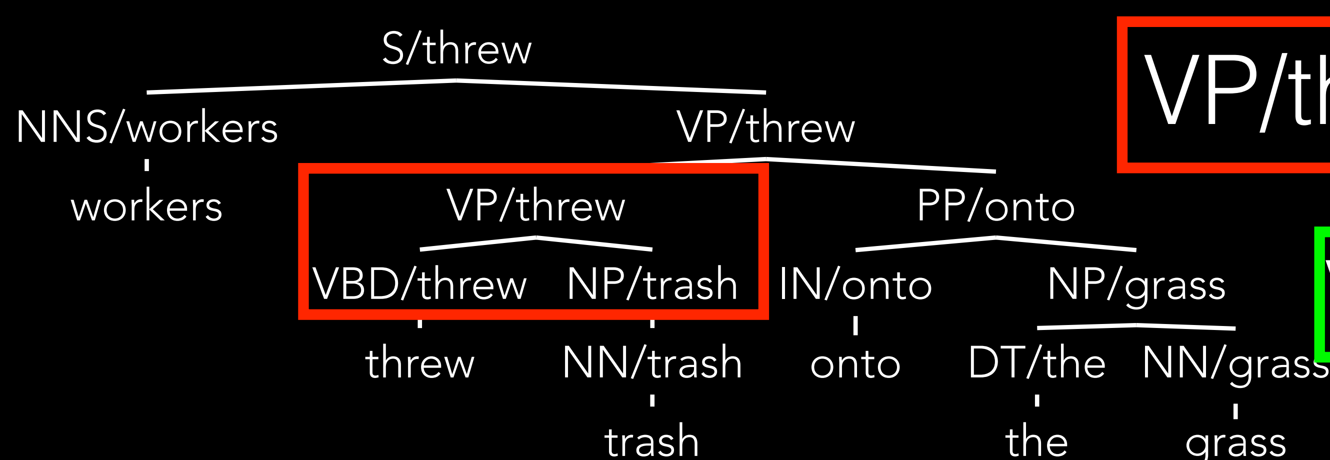
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP/threw -> VBD/threw NP

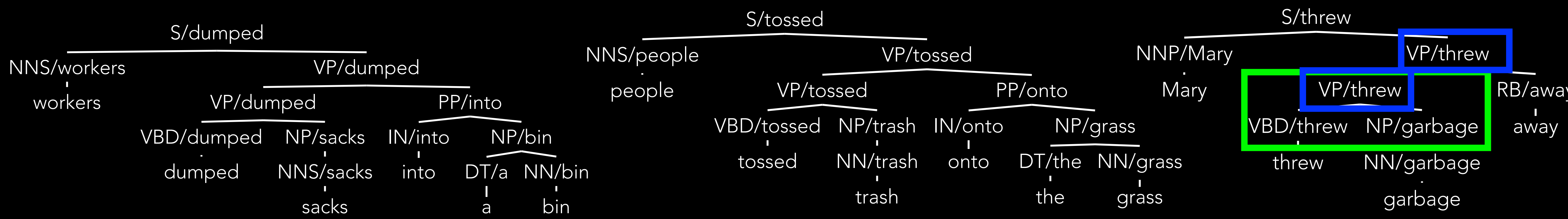
VP/threw -> …

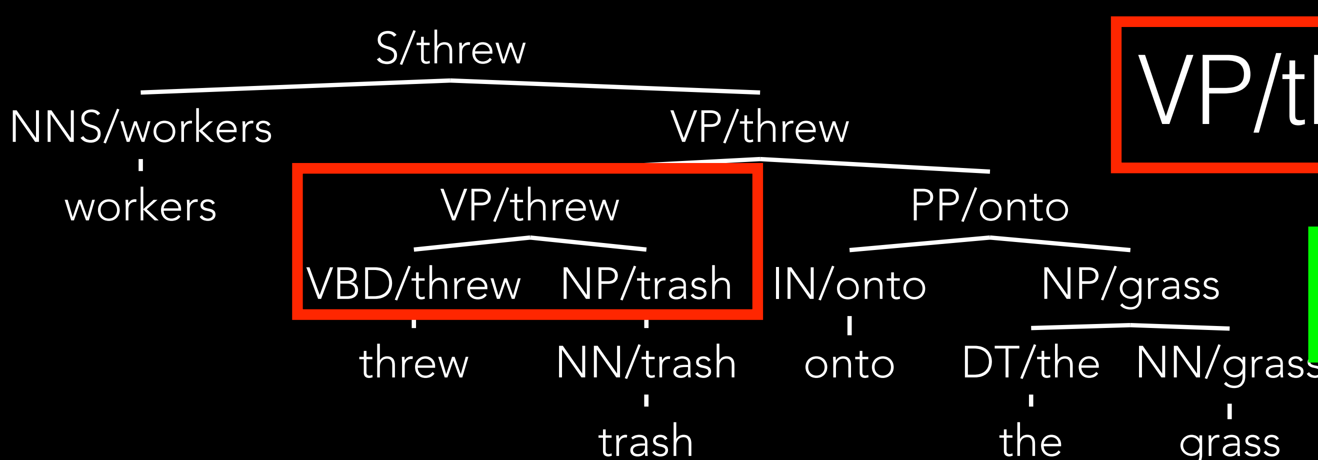
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP/threw -> VBD/threw NP

VP/threw -> …

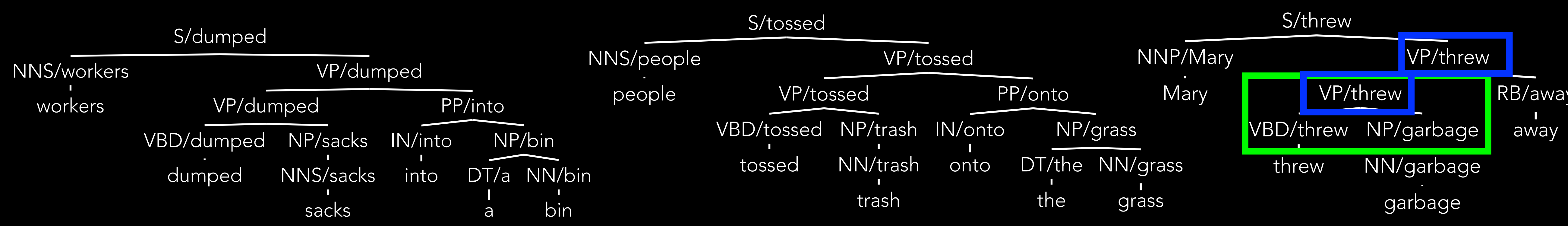
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

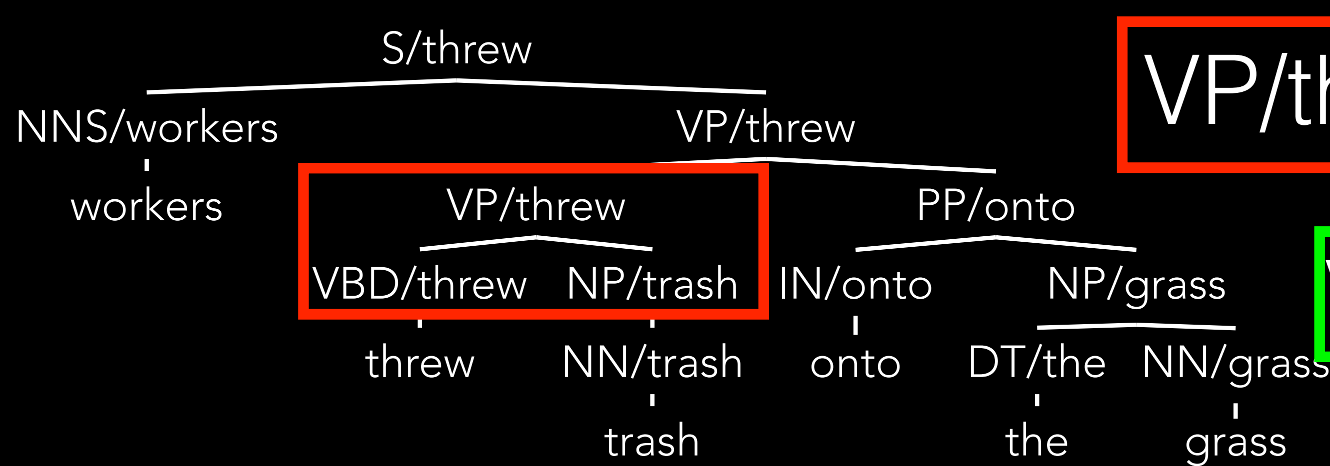
20/40

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP/threw -> VBD/threw NP

VP/threw -> …

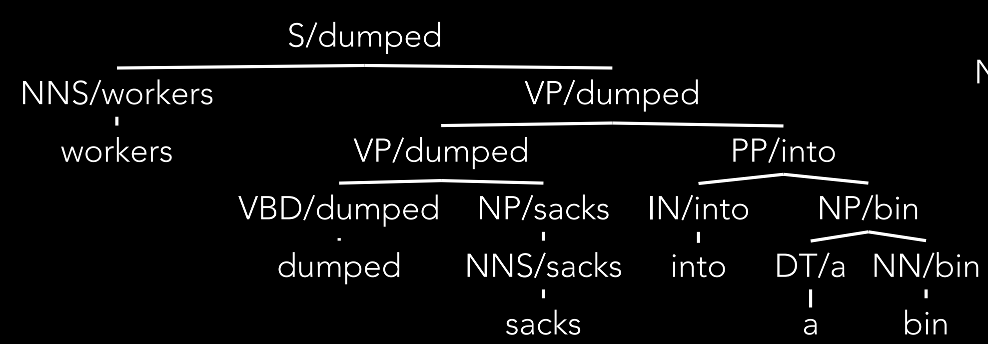
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

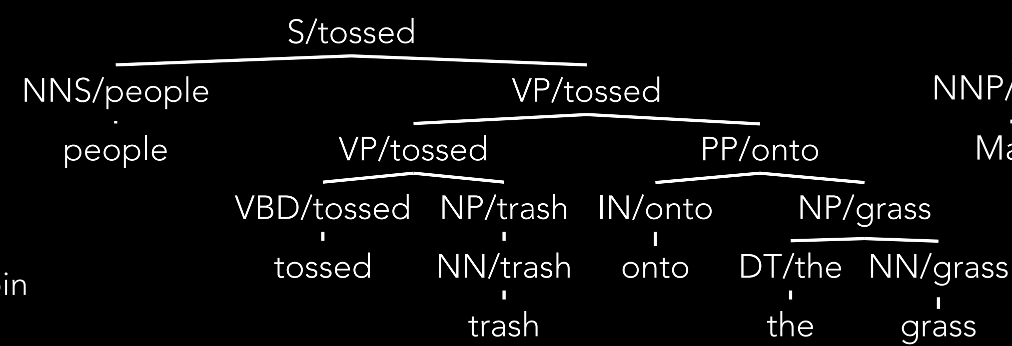
$$\boxed{20}/40$$

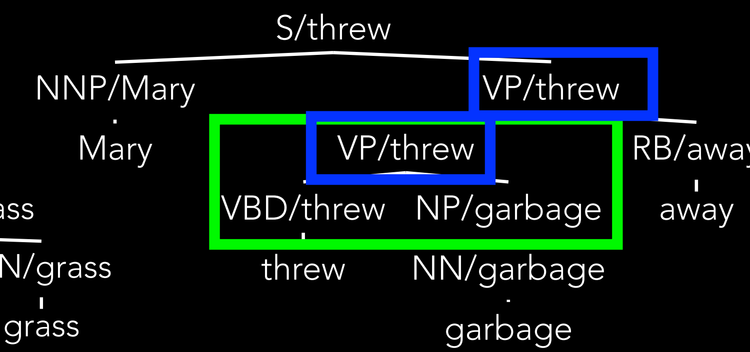
Head lexicalization

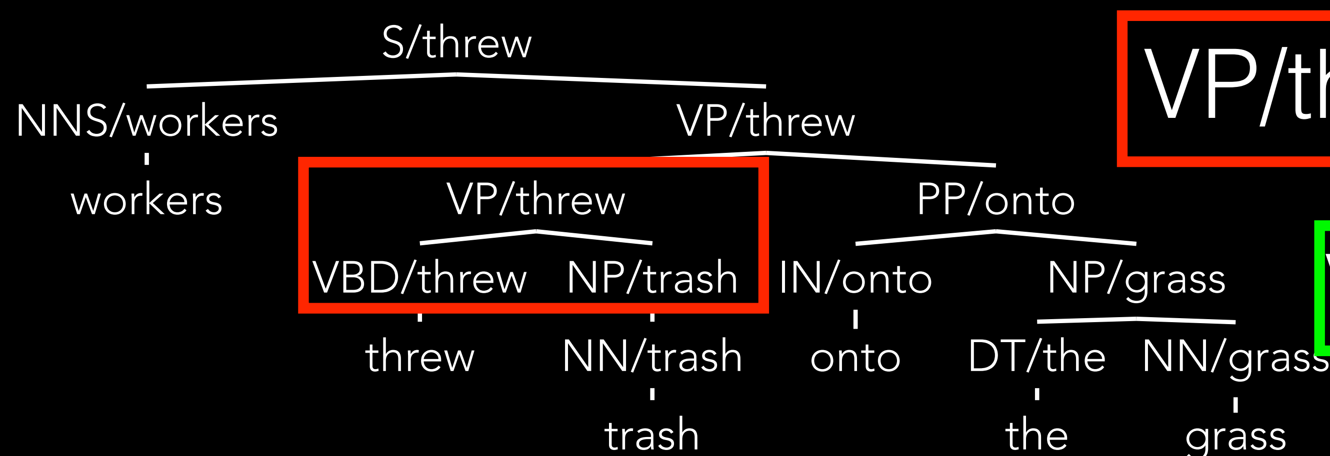
By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times









VP/threw -> VBD/threw NP/trash

VP/threw -> VBD/threw NP

VP/threw -> …

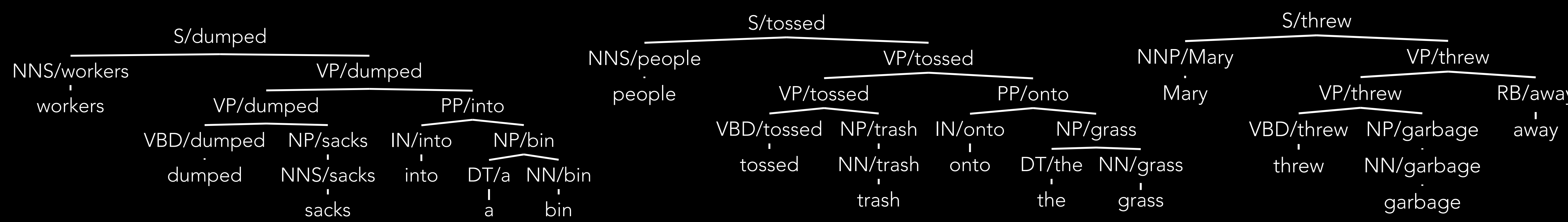
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

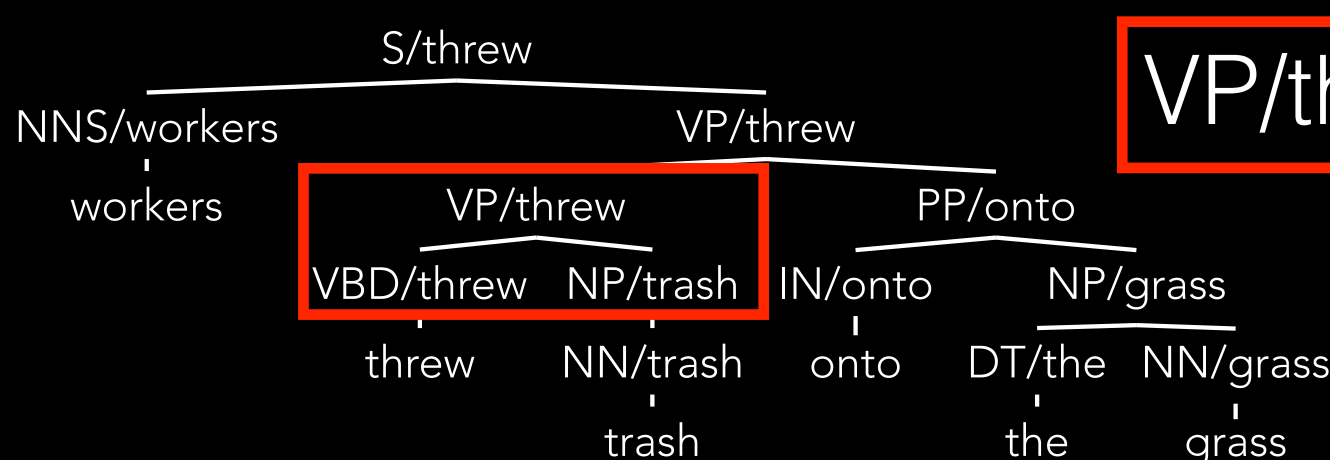
$$\frac{20}{40}$$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





$\boxed{\text{VP/threw -> VBD/threw NP/trash}}$

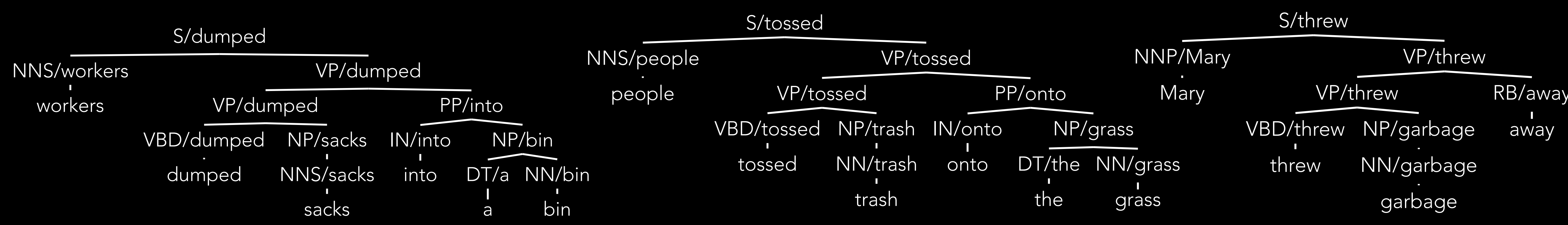
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

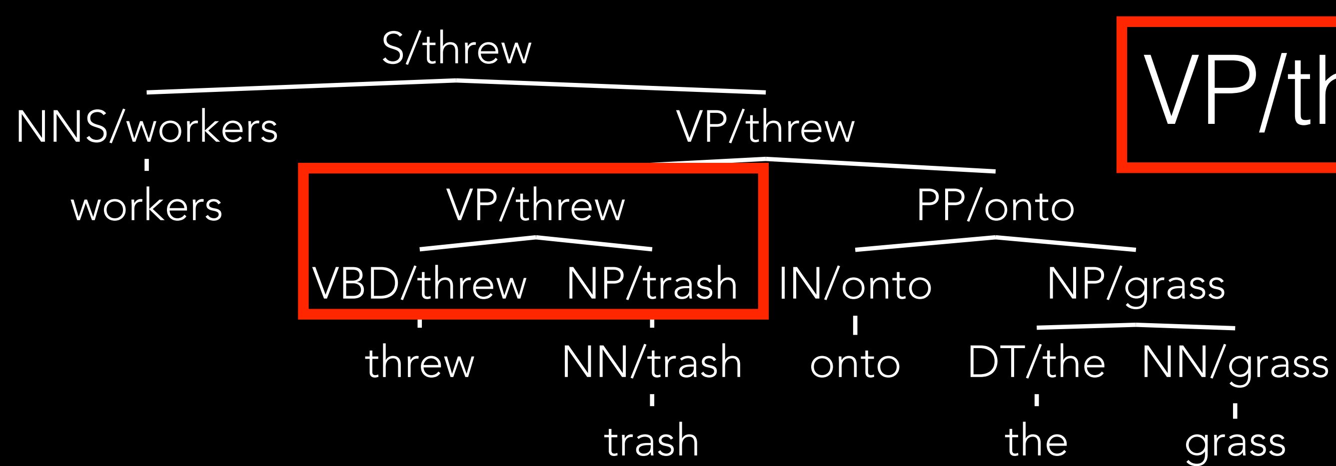
20/40

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP -> VBD NP

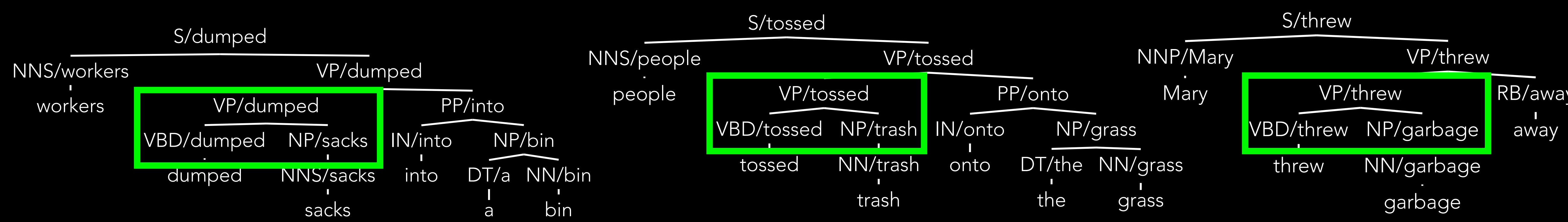
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

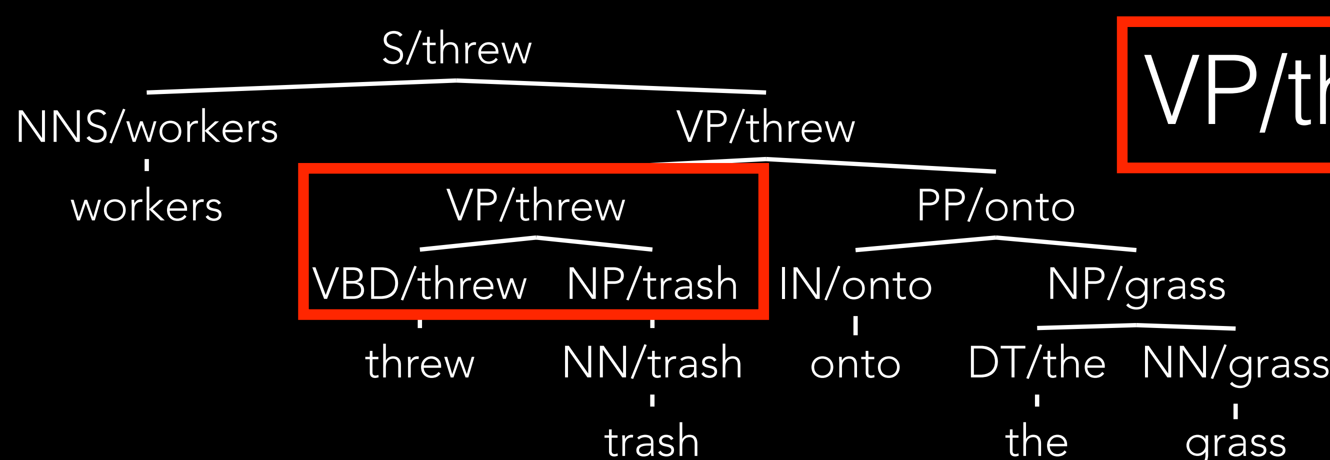
20/40

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP -> VBD NP

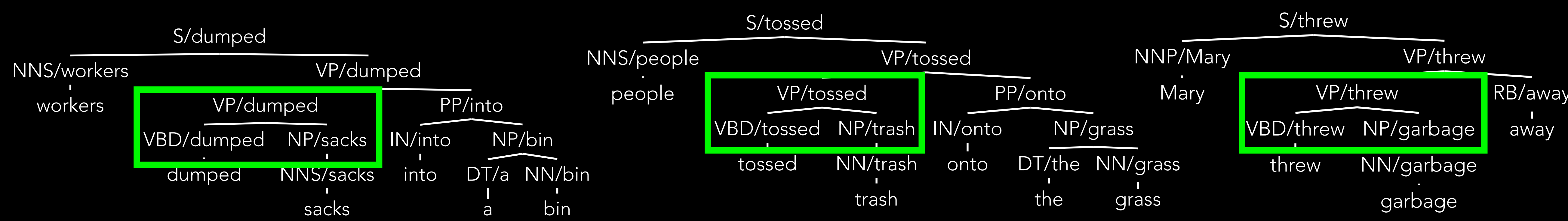
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

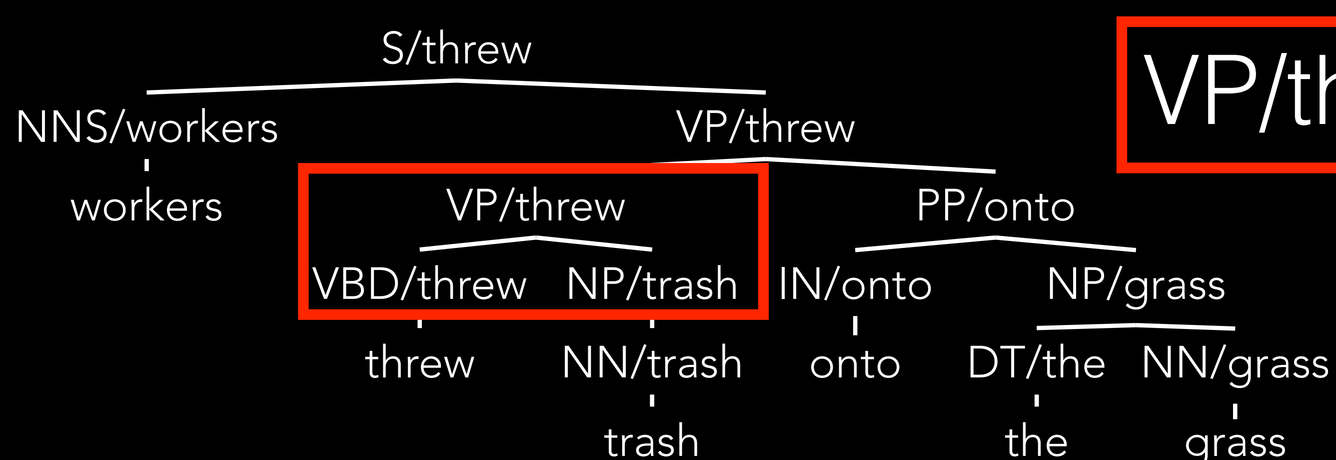
20/40

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP -> VBD NP

VP -> ...

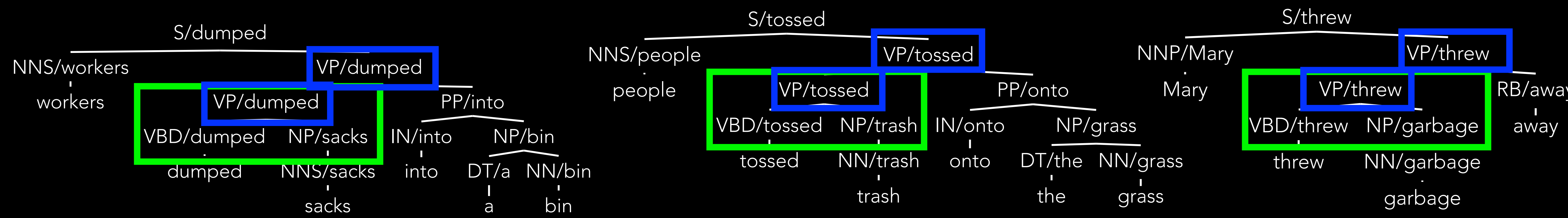
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

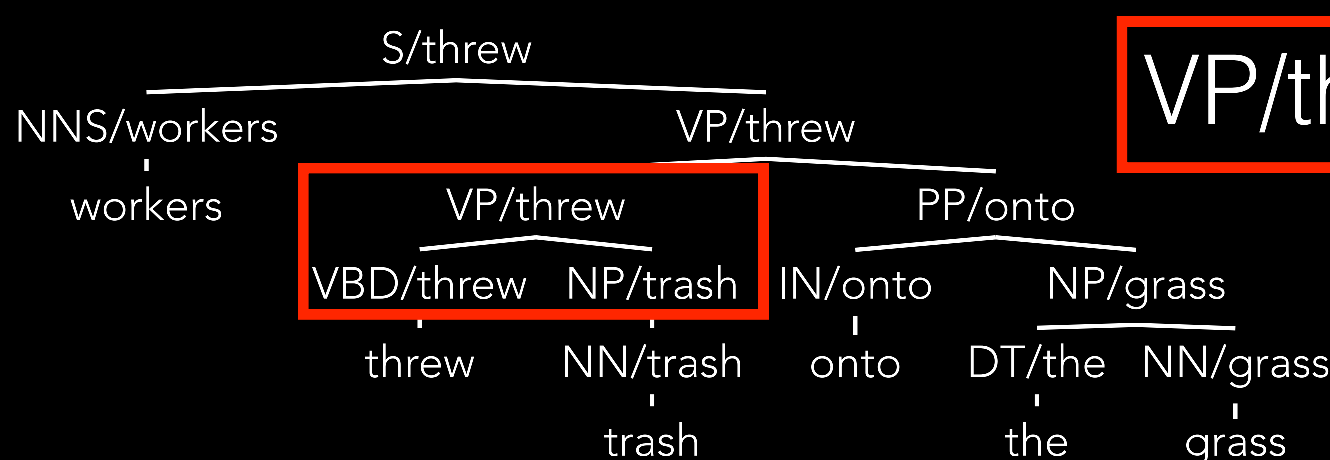
20/40

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP -> VBD NP

VP -> …

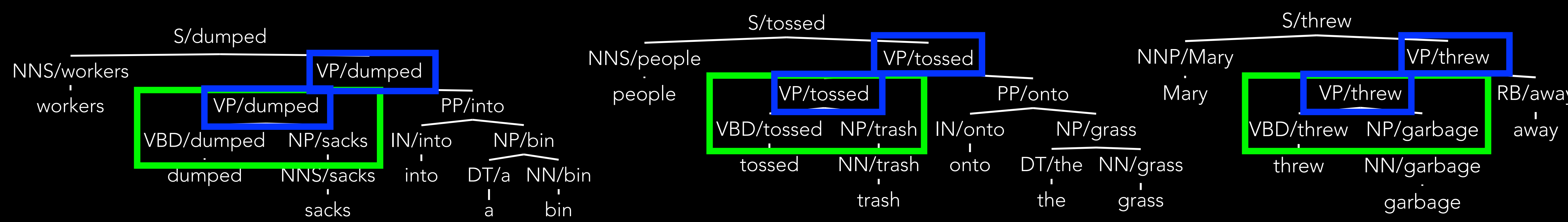
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

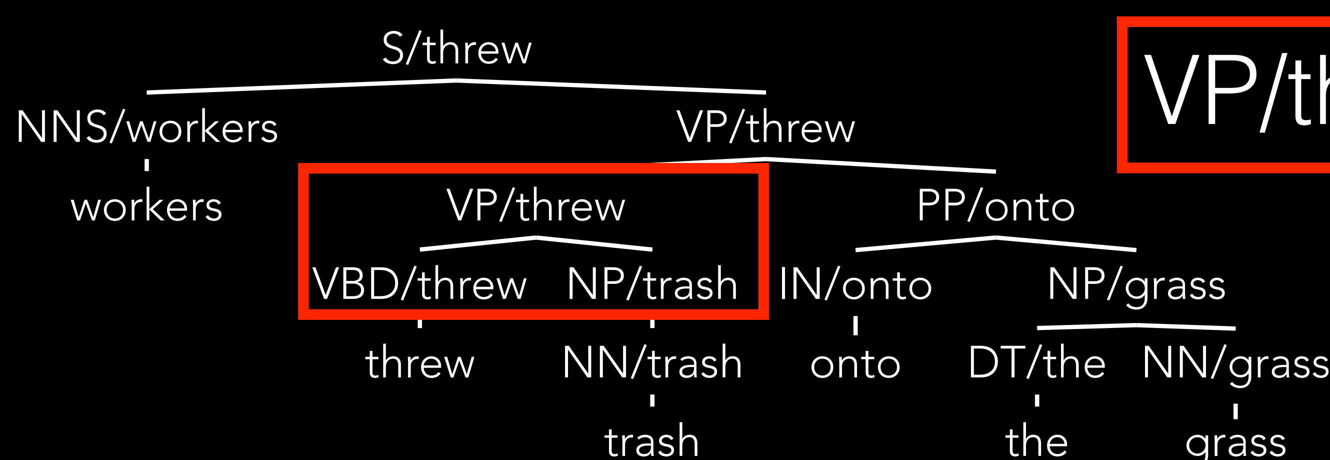
20/40

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP -> VBD NP

VP -> ...

$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

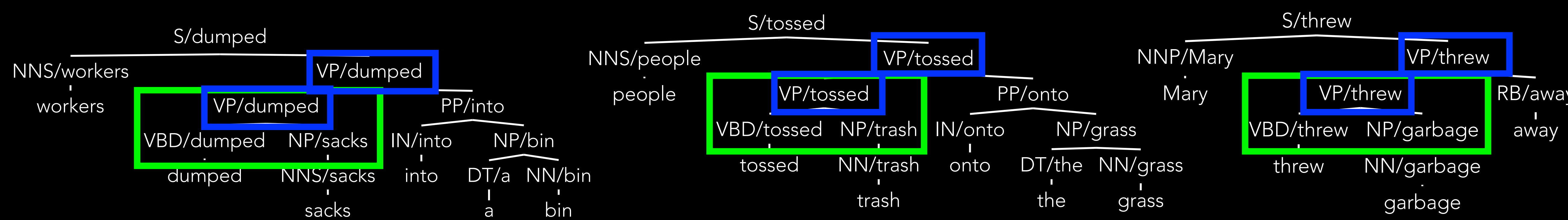
20/40

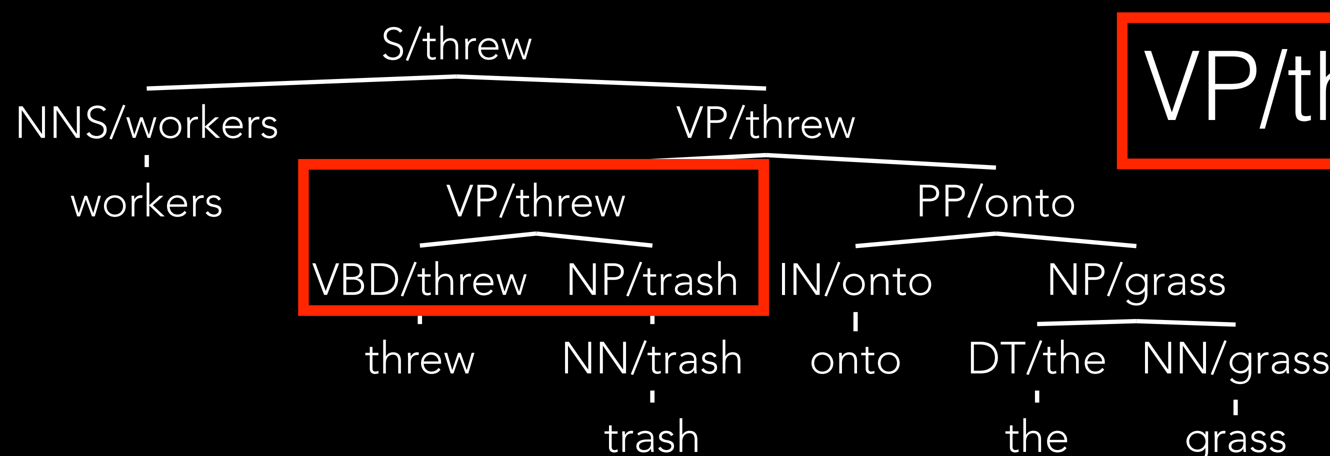
60/120

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP -> VBD NP

VP -> ...

$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

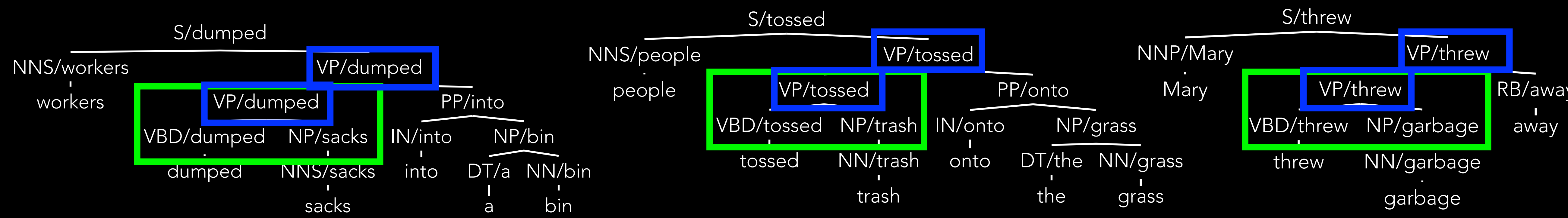
20/40

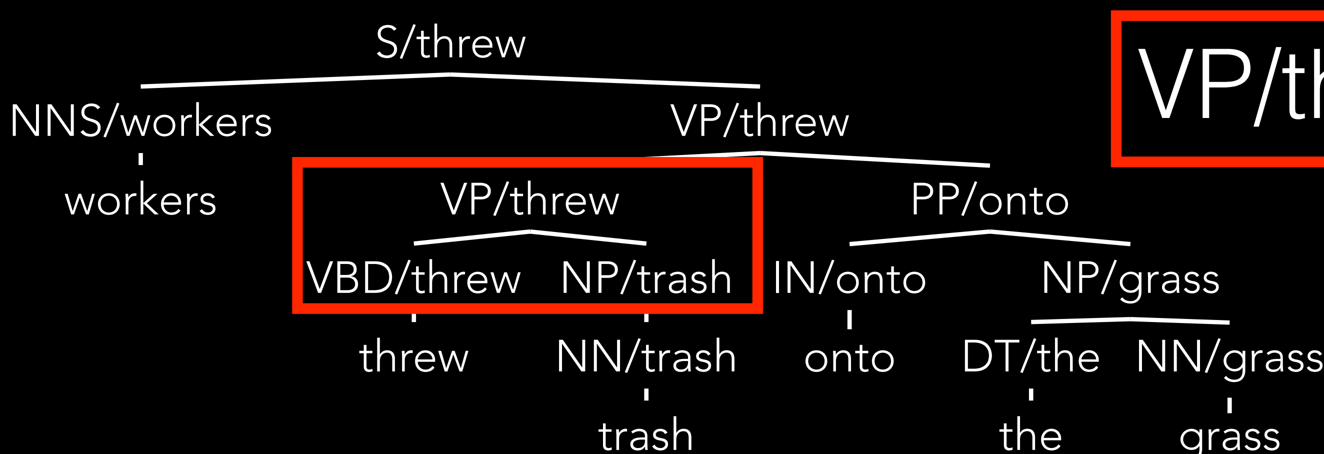
60/120

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





$\boxed{\text{VP/threw -> VBD/threw NP/trash}}$

$\boxed{\text{VP -> VBD NP}}$ $\boxed{\text{VP -> ...}}$

$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

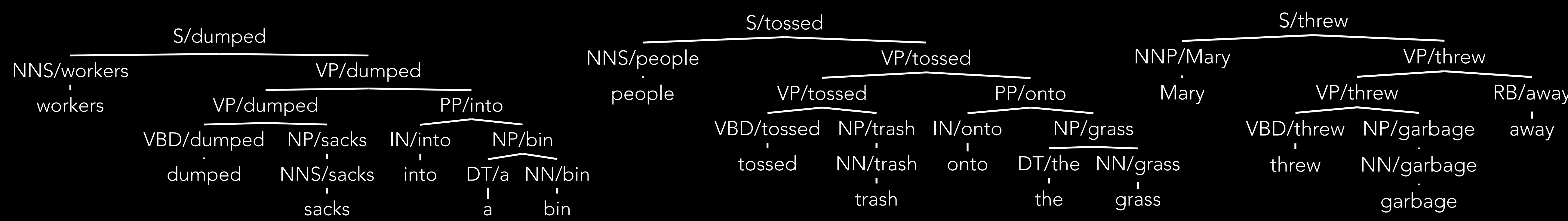
20/40

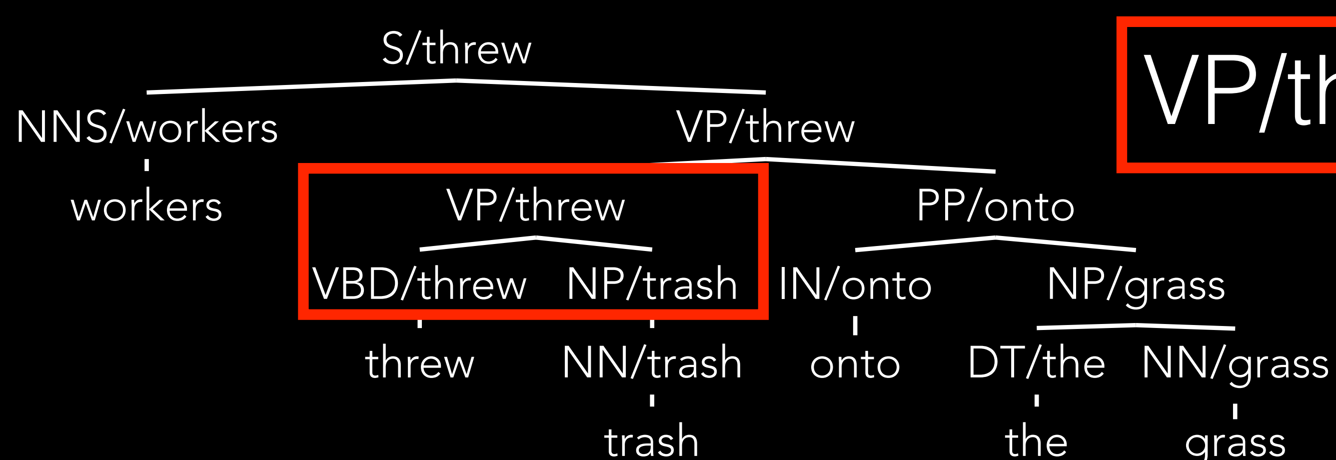
$\boxed{60}/\boxed{120}$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





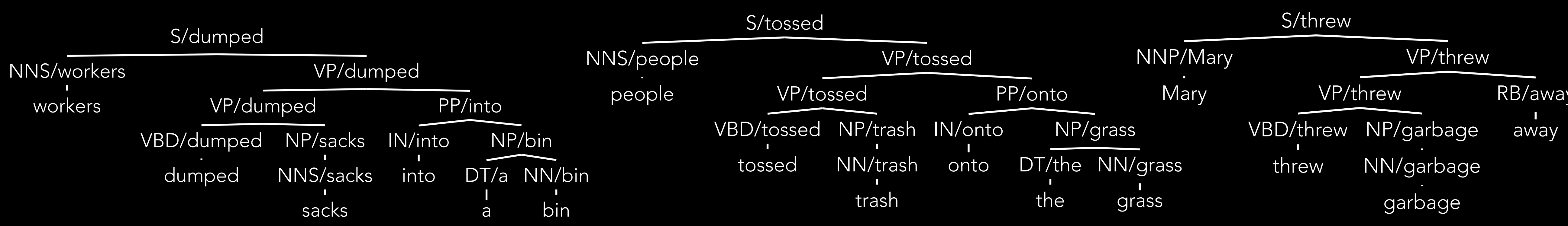
VP/threw -> VBD/threw NP/trash

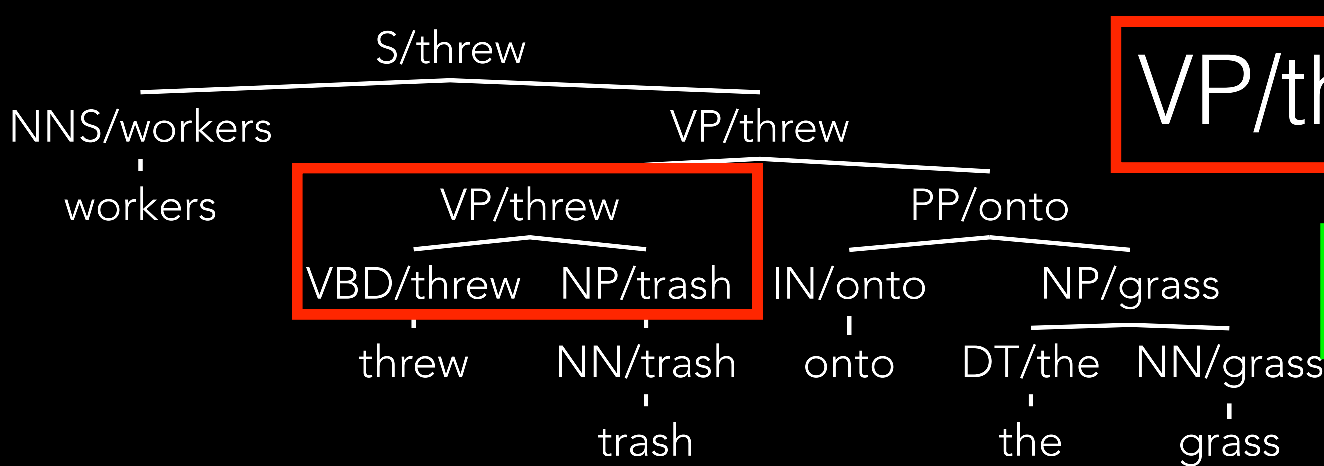
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

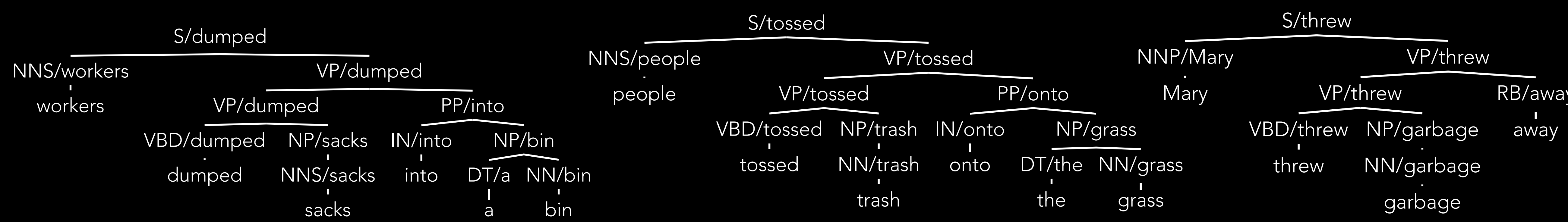
VP/threw -> VBD/threw NP/trash

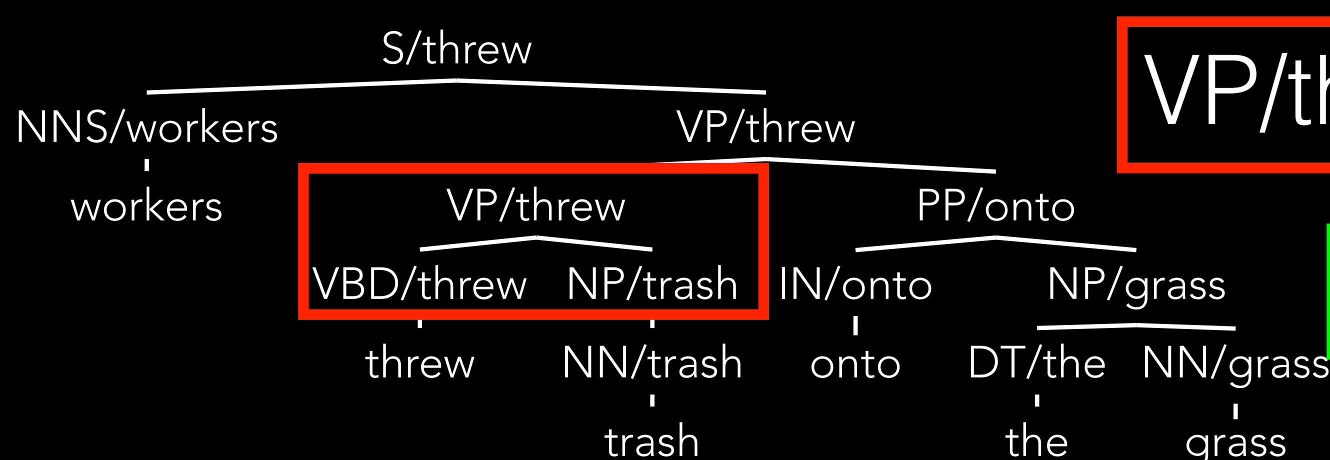
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP/threw -> VBD/threw NP/trash

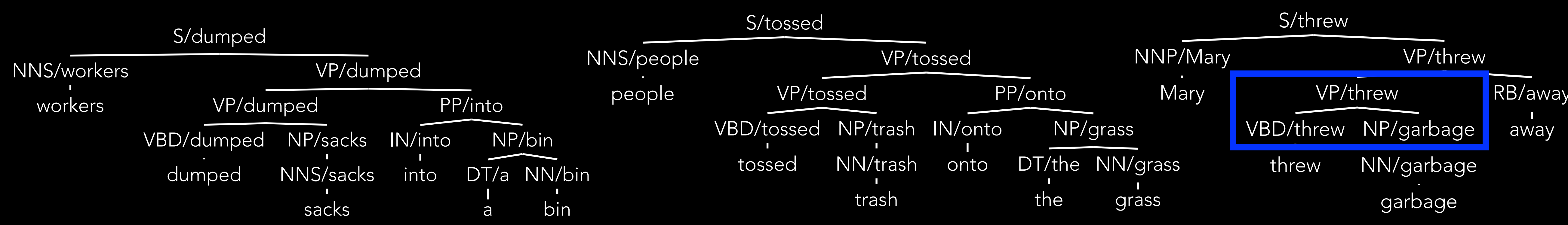
VP/threw -> VBD/threw NP

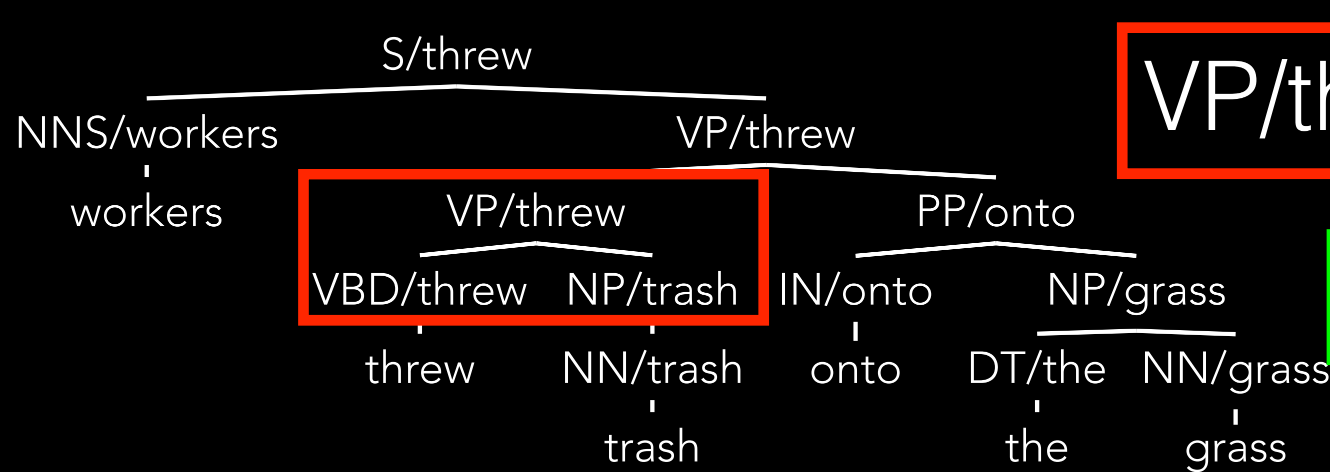
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP/threw -> VBD/threw NP/trash

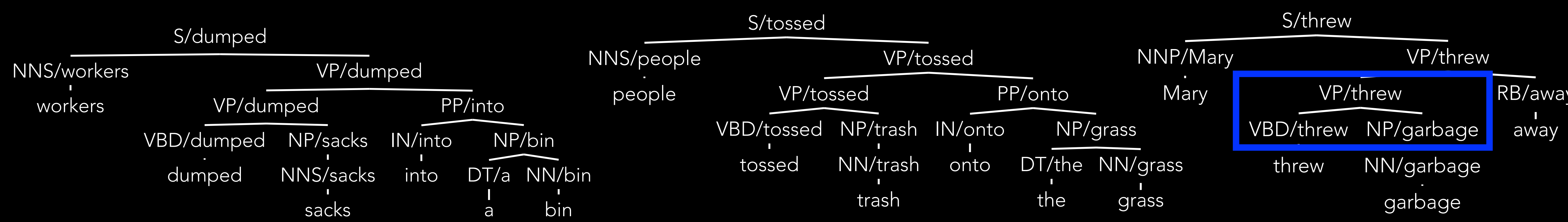
VP/threw -> VBD/threw NP

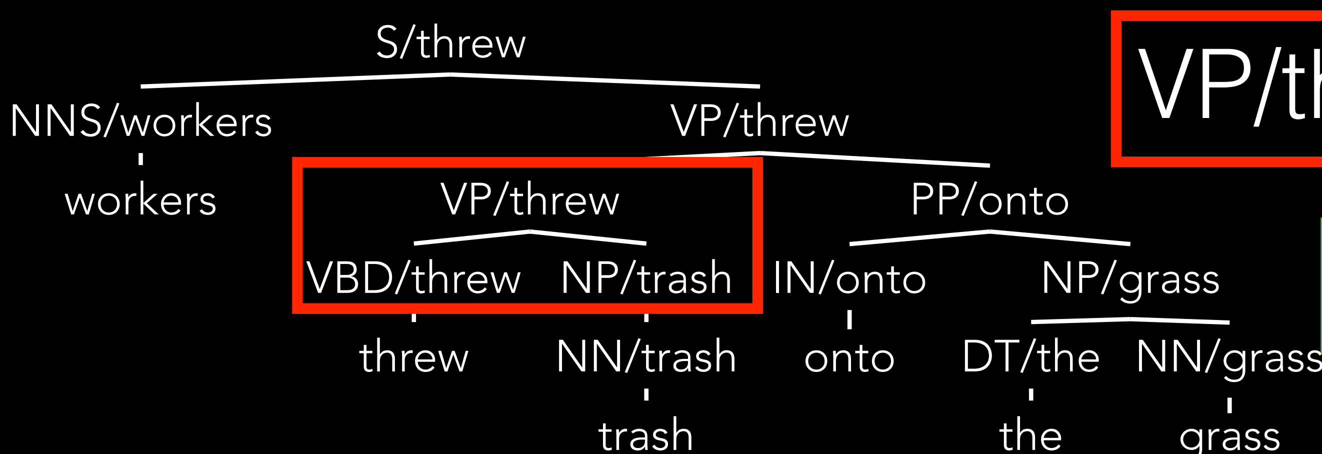
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP/threw -> VBD/threw NP/trash

VP/threw -> VBD/threw NP

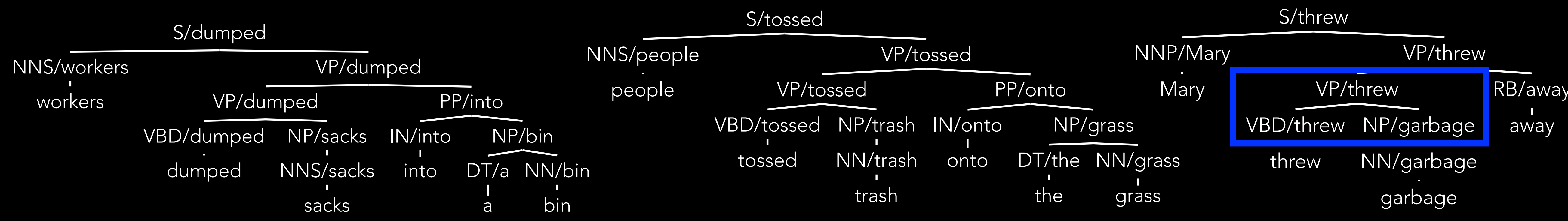
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

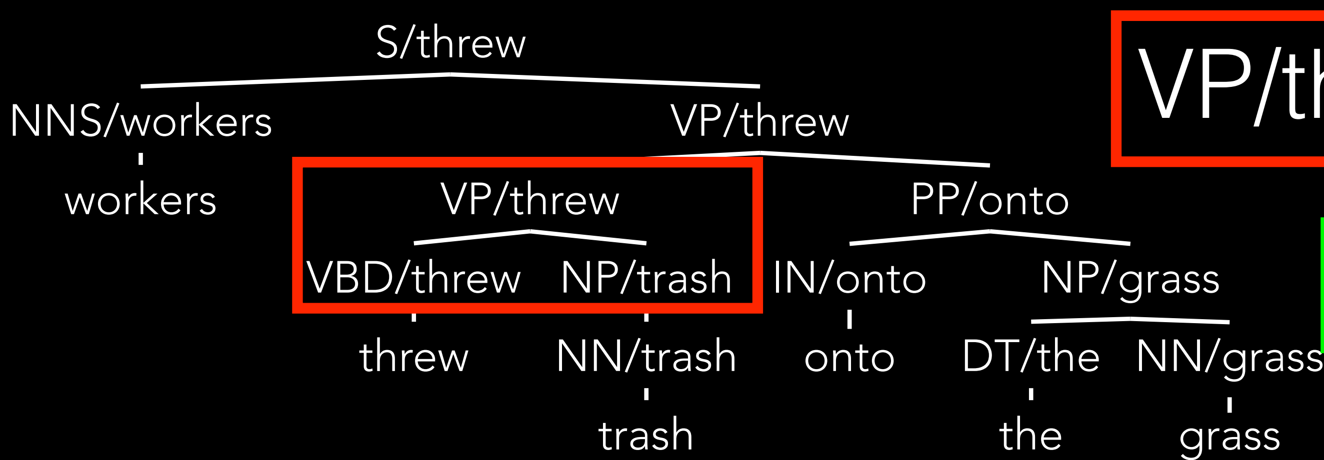
0/20

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP/threw -> VBD/threw NP/trash

VP/threw -> VBD/threw NP

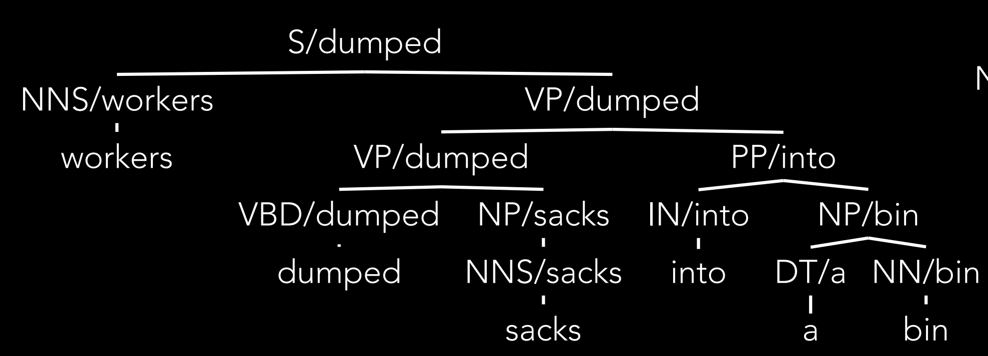
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

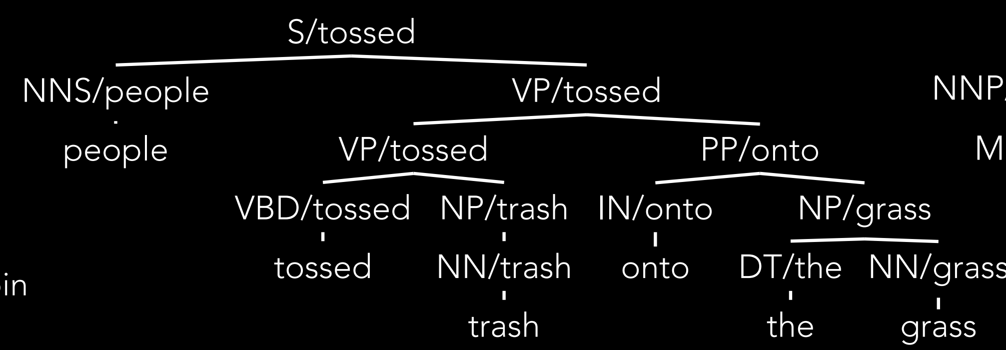
0/20

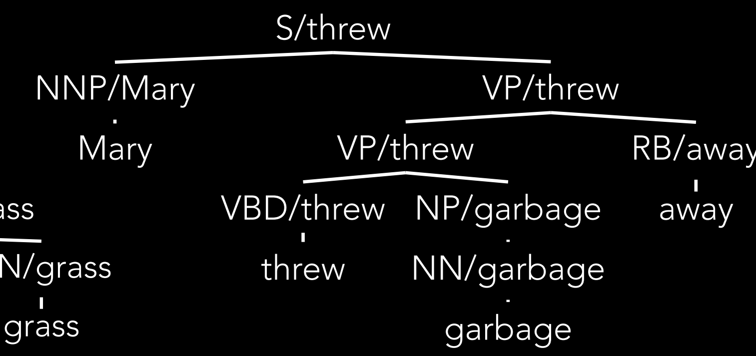
Head lexicalization

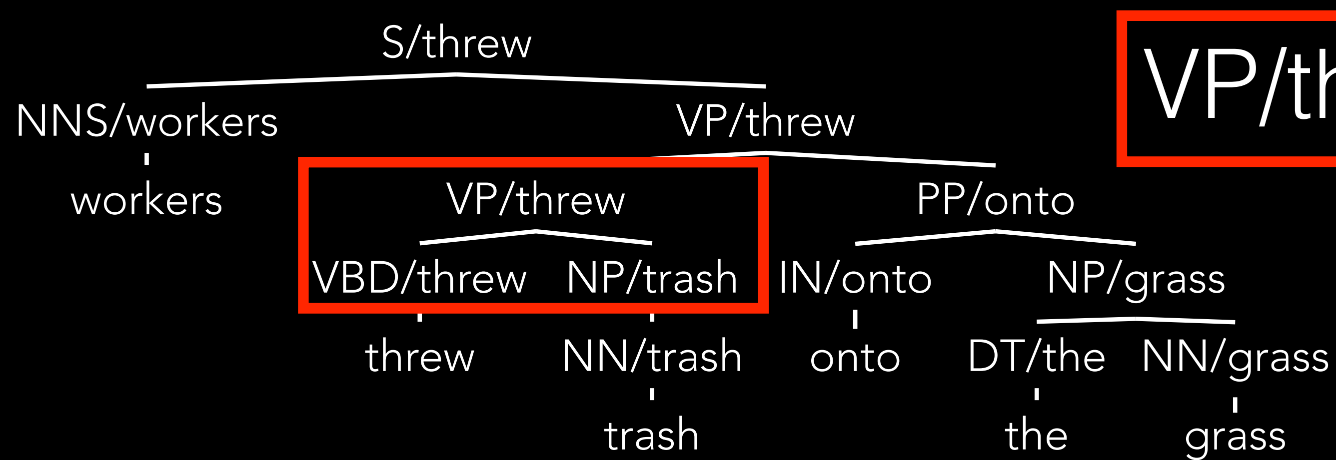
By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times









VP/threw -> VBD/threw NP/trash

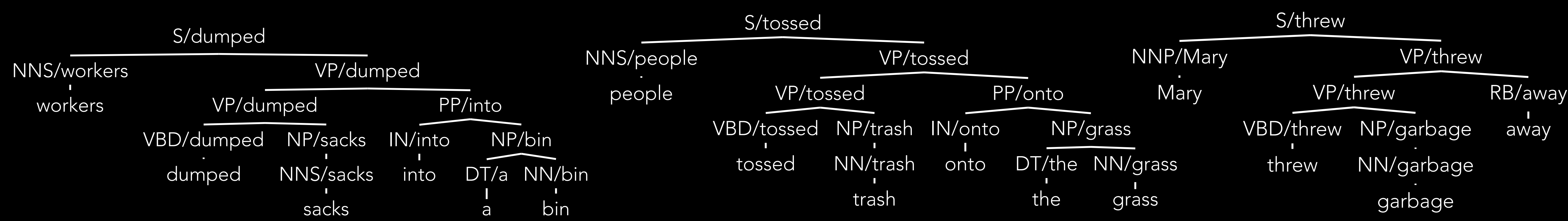
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

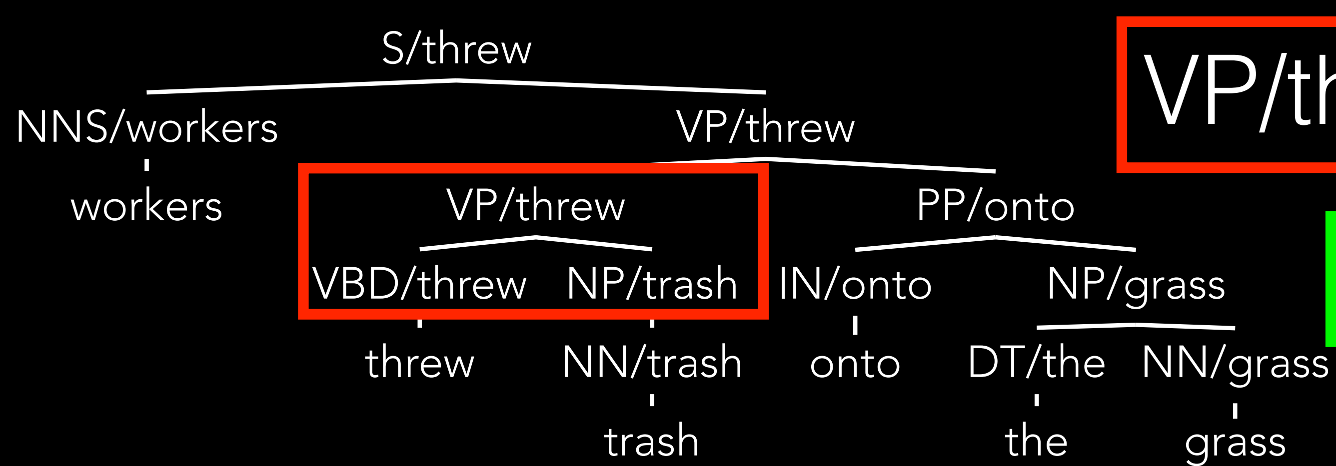
0/20

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP -> VBD NP/trash

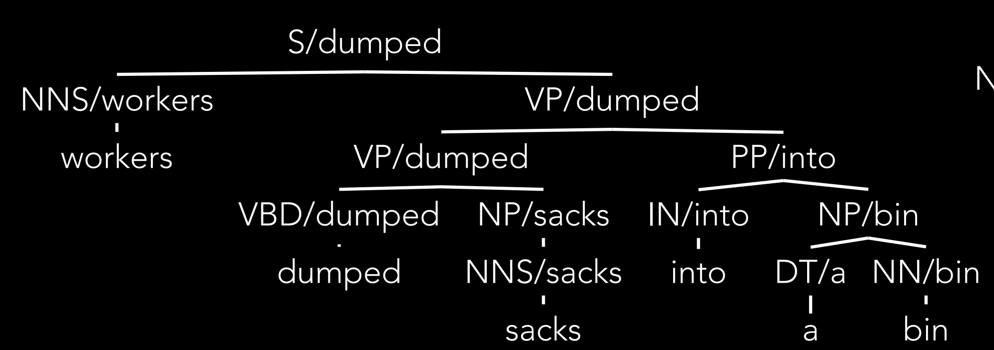
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

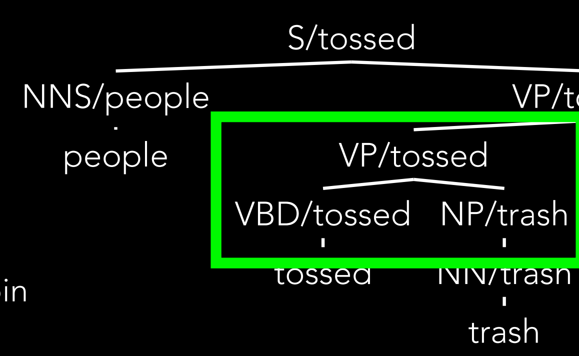
0/20

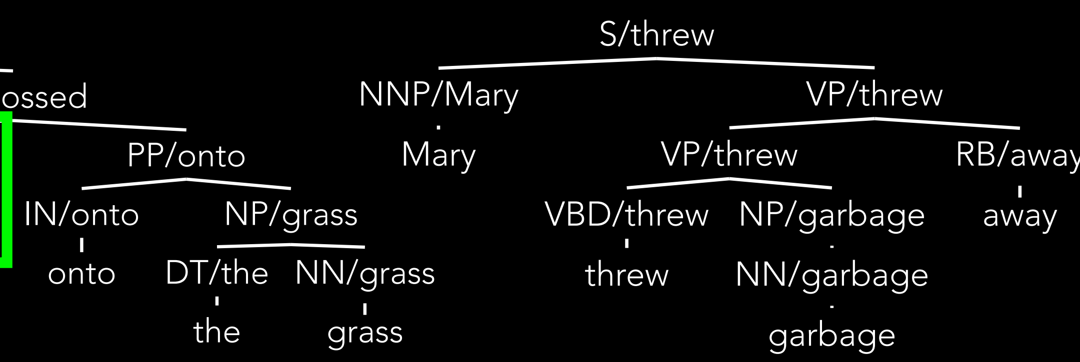
Head lexicalization

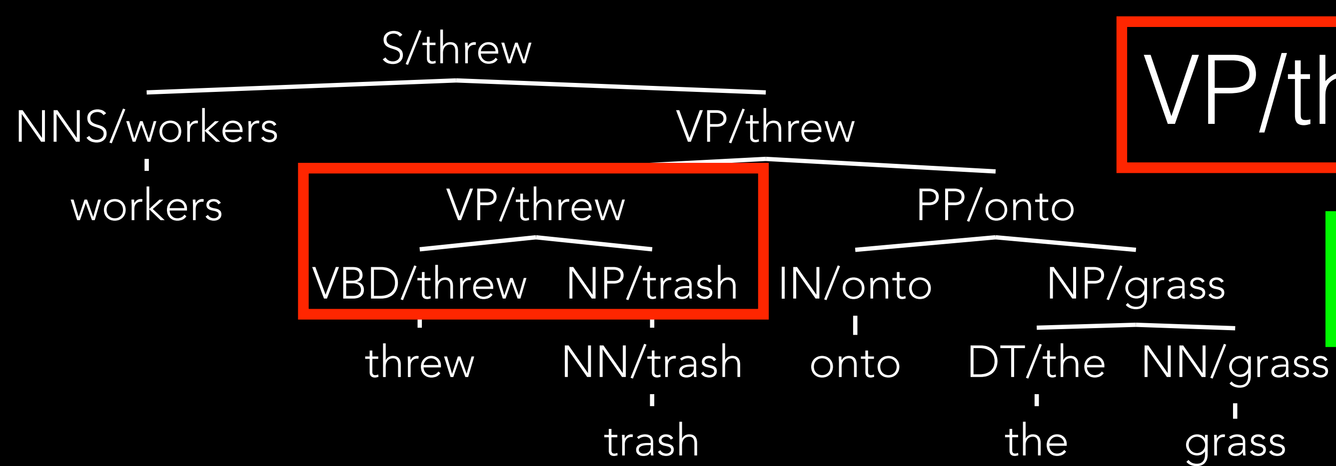
By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times









VP/threw -> VBD/threw NP/trash

VP -> VBD NP/trash

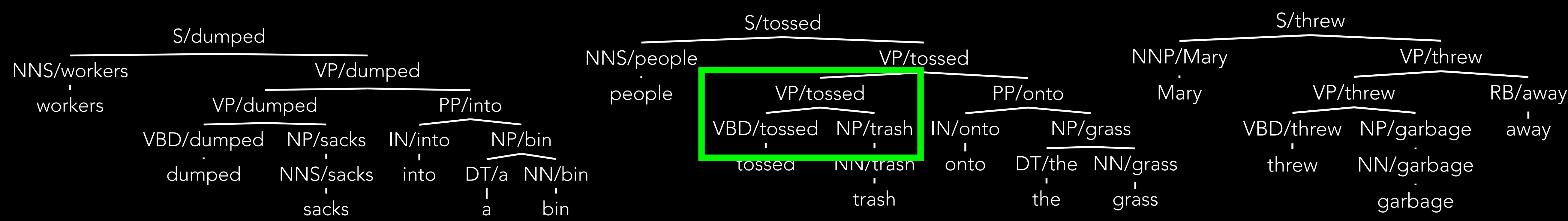
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

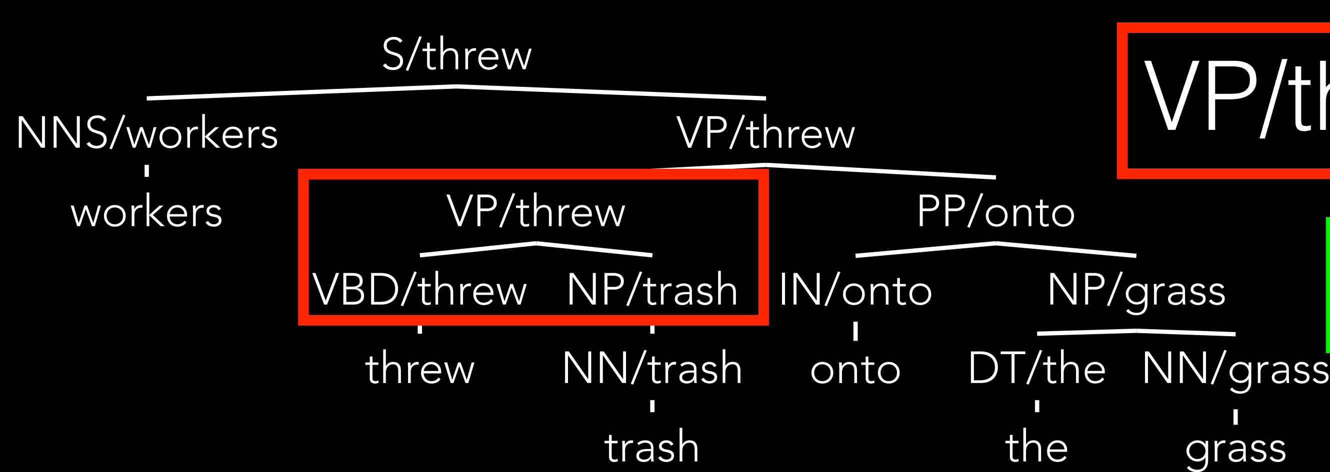
0/20

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP -> VBD NP/trash

VP -> VBD NP

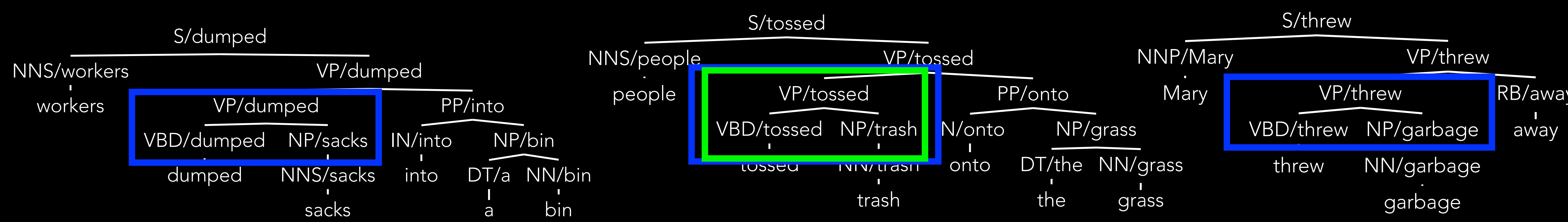
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

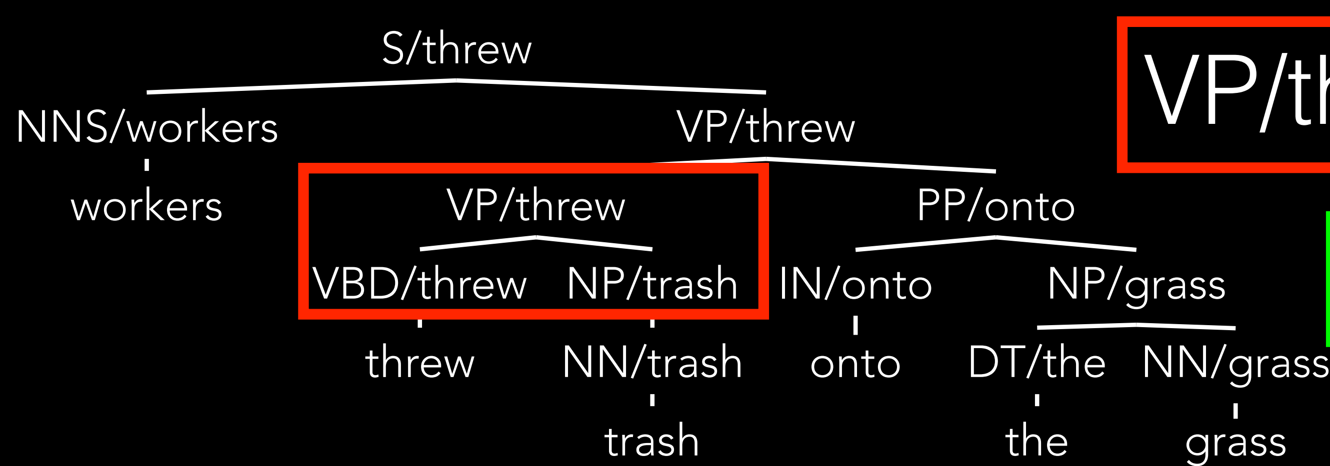
0/20

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP -> VBD NP/trash

VP -> VBD NP

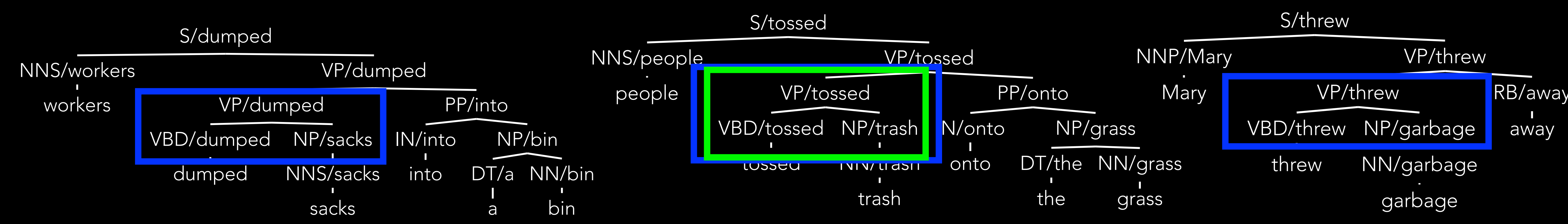
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

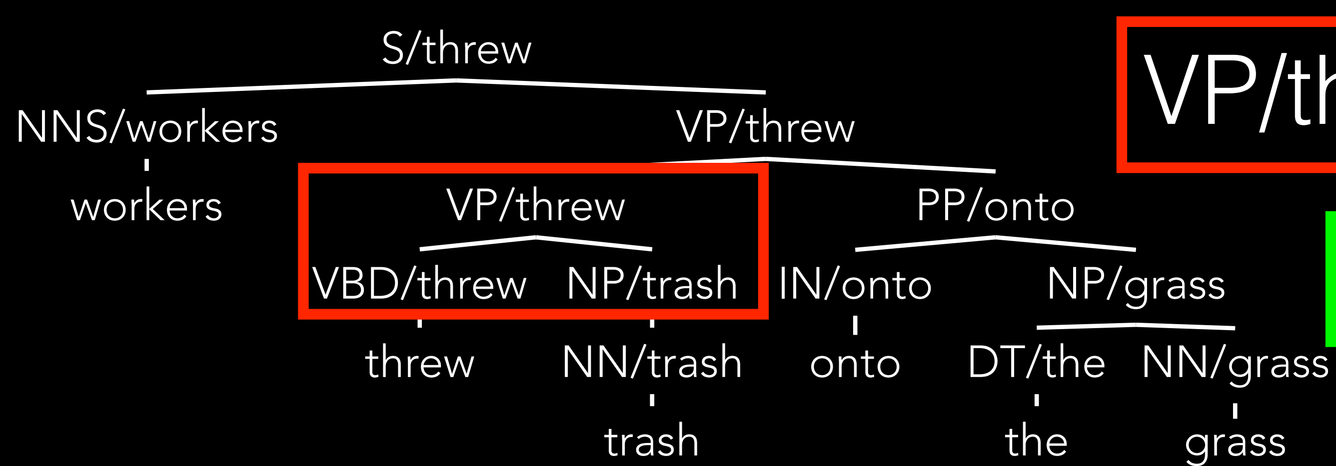
0/20

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP -> VBD NP/trash

VP -> VBD NP

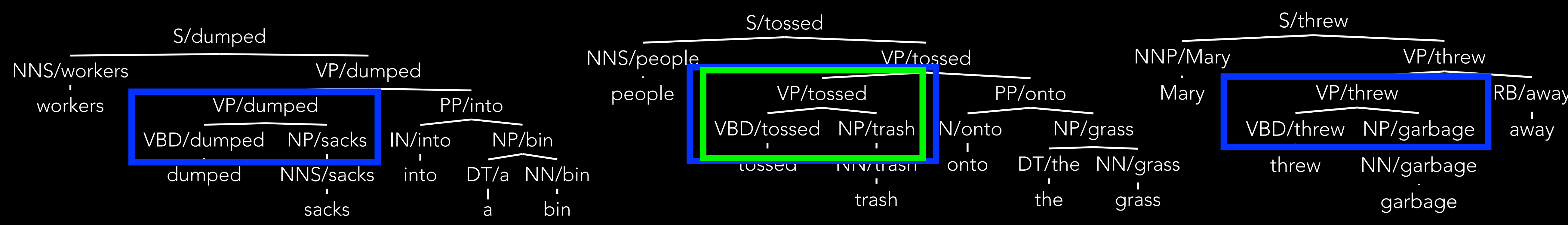
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

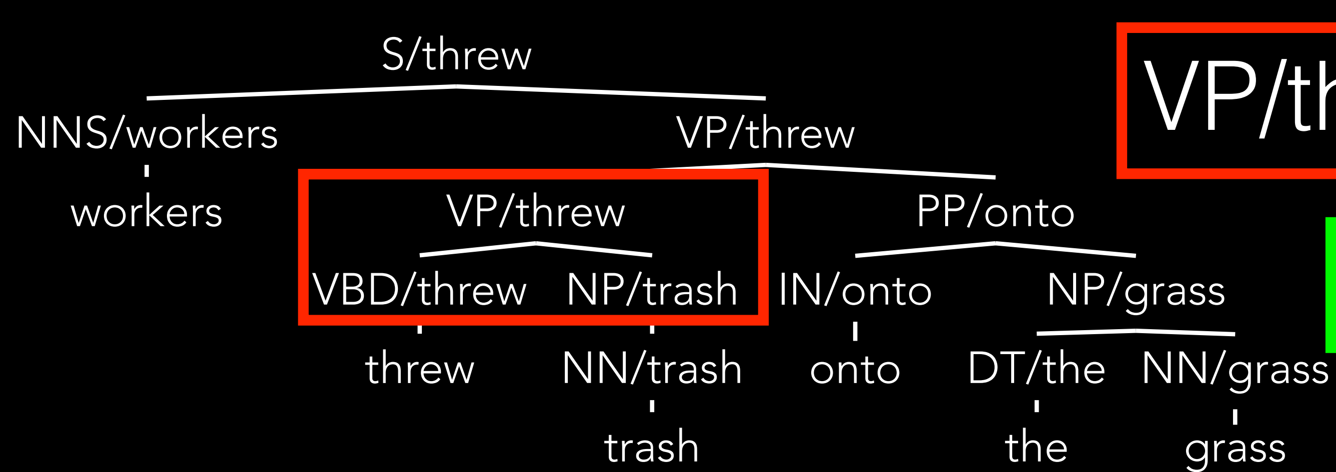
0/20 20/60

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP -> VBD NP/trash

VP -> VBD NP

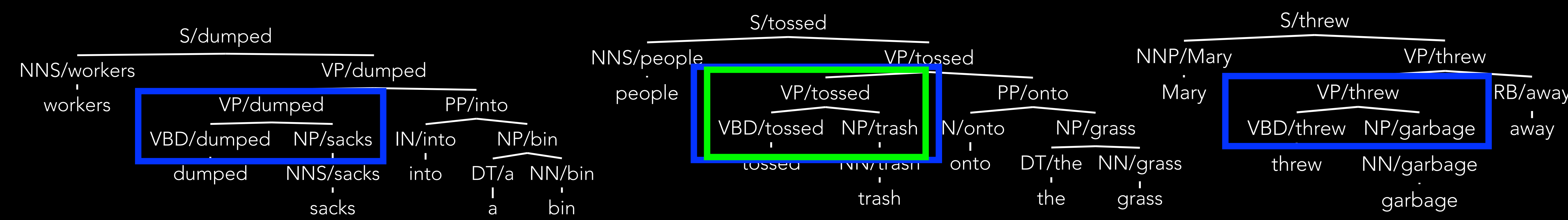
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

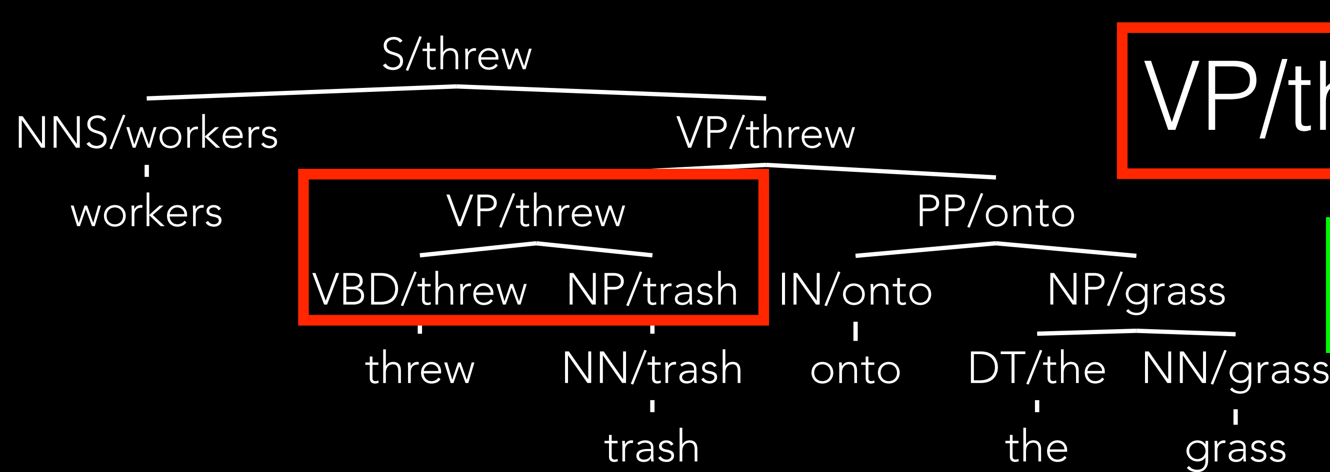
0/20 $\qquad$ $\boxed{20}$/60

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

VP -> VBD NP/trash

VP -> VBD NP

$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

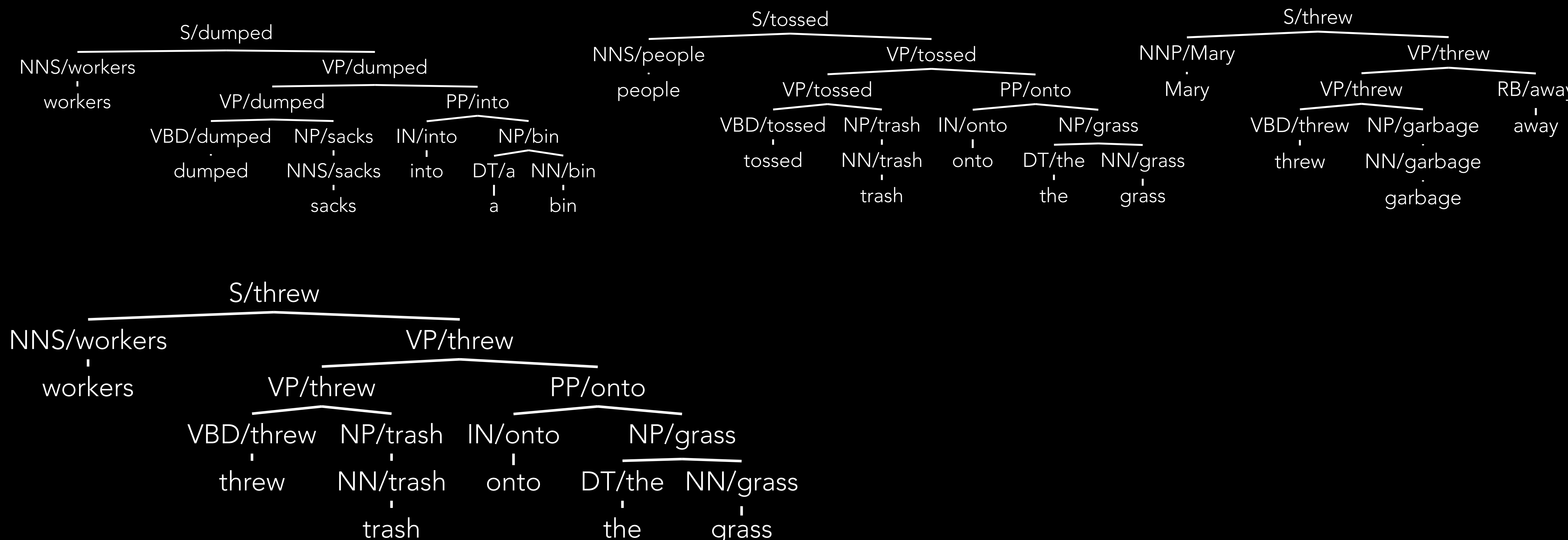
0/20

$\boxed{20}/\boxed{60}$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

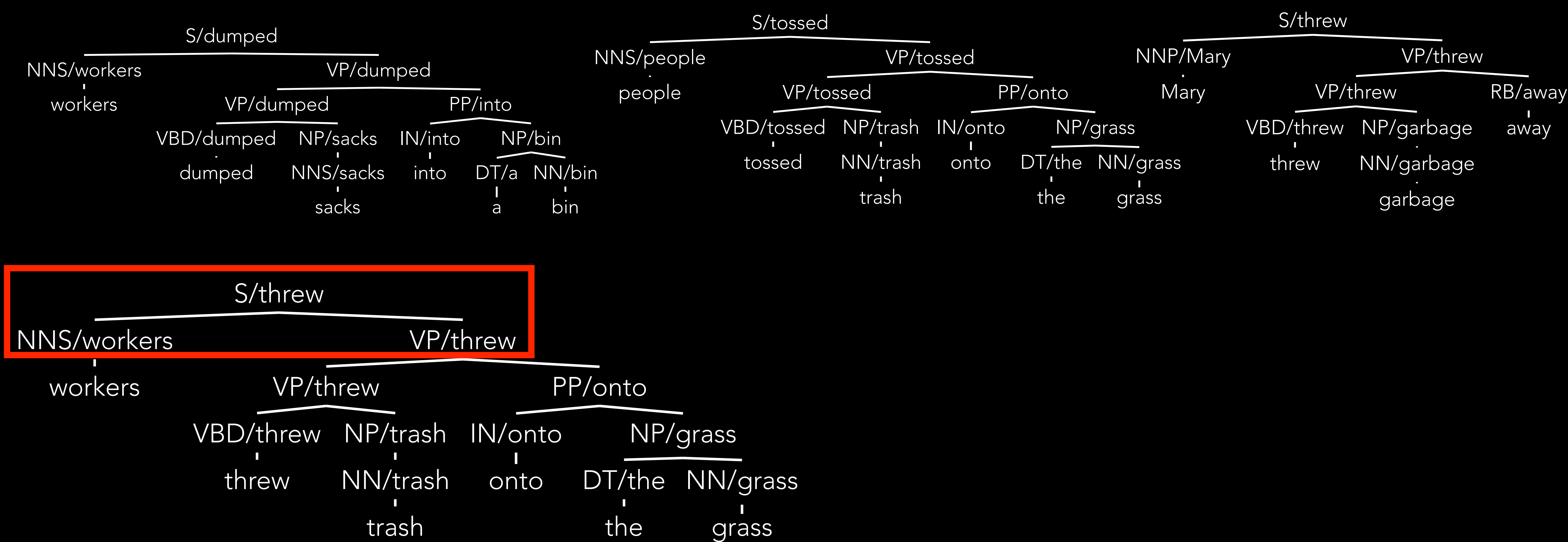
Assume we saw each training tree 20 times



Head lexicalization

By decomposing we can estimate unseen lexicalized rules

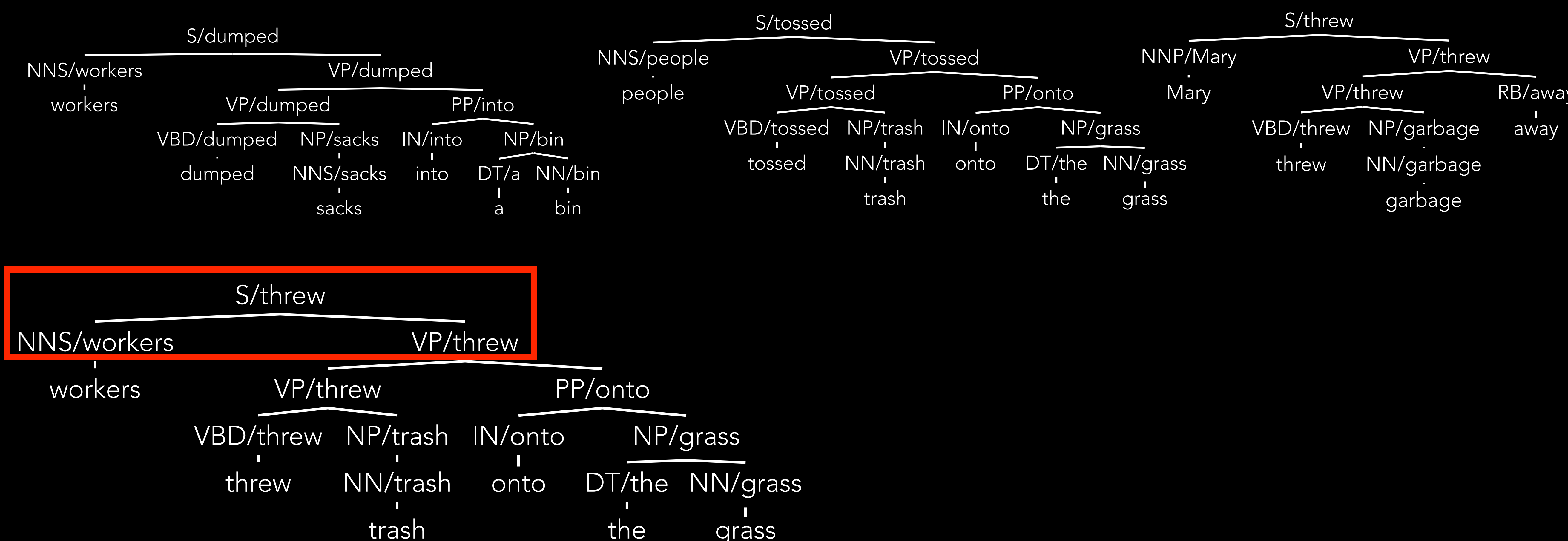
Assume we saw each training tree 20 times



Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times

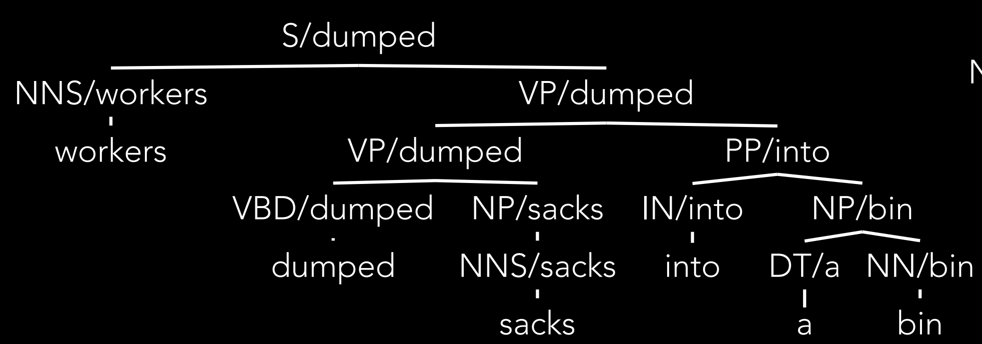


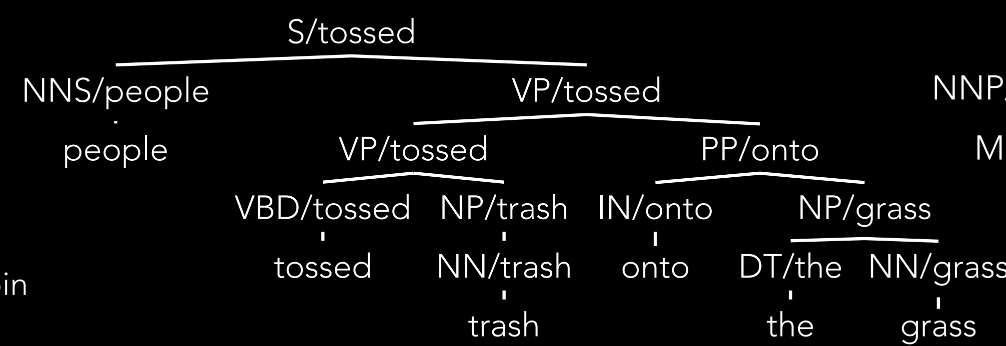
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

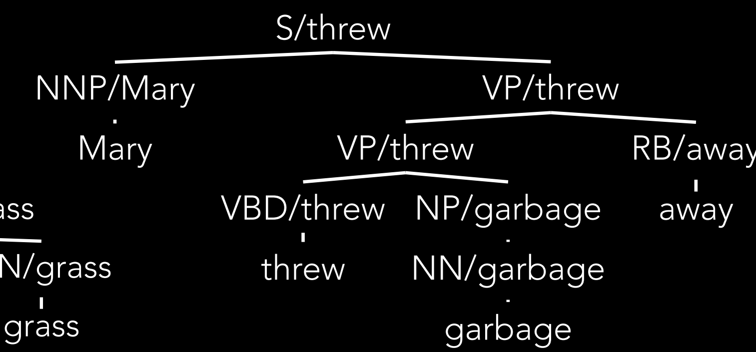
Head lexicalization

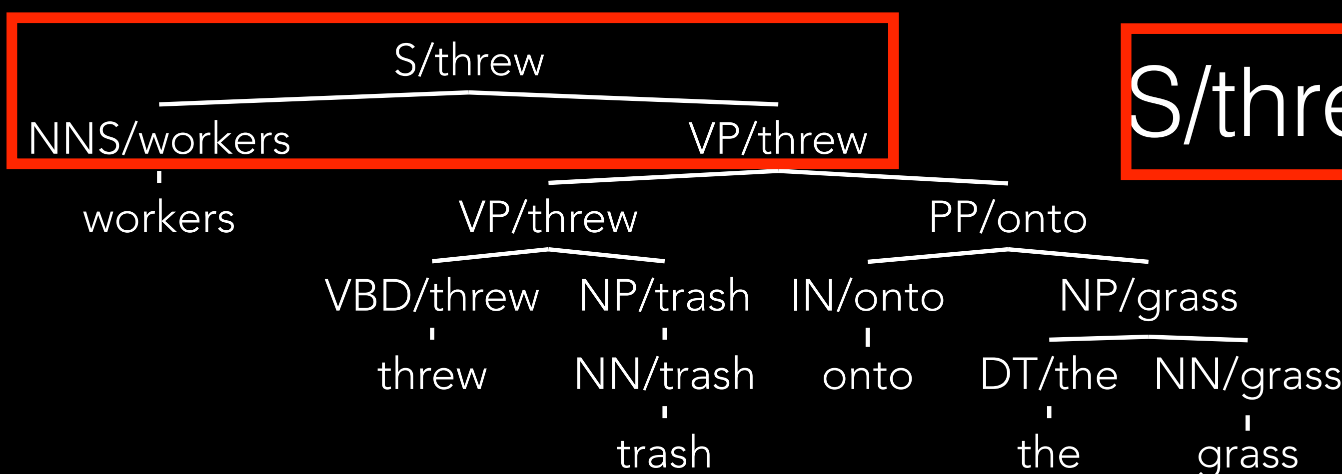
By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times









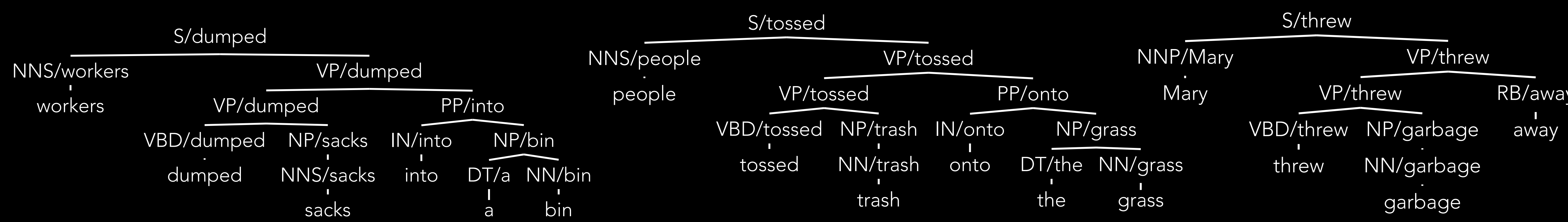
S/threw -> NNS/workers VP/threw

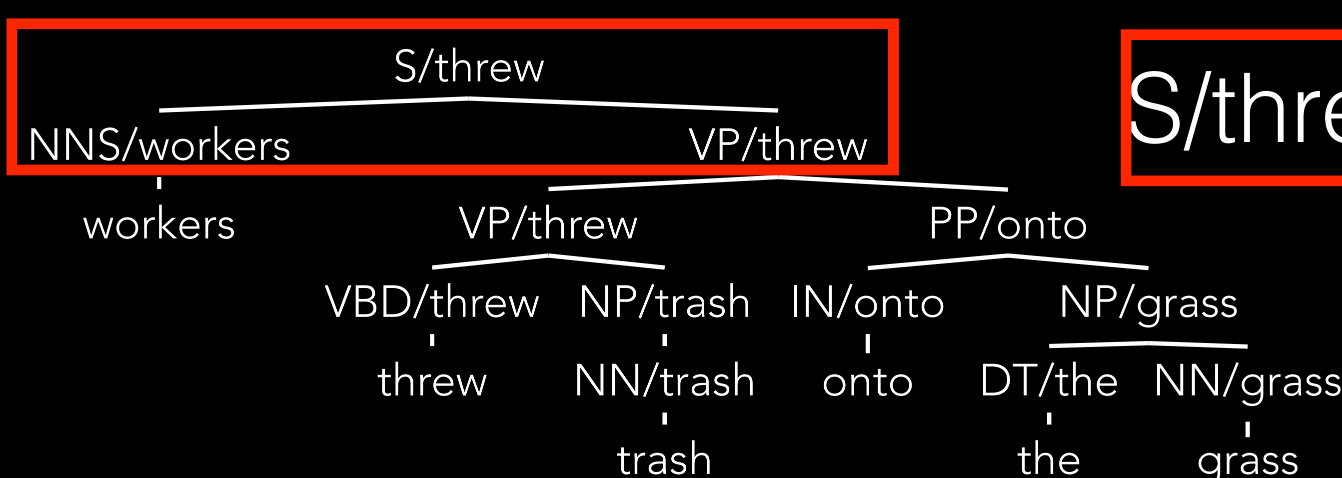
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

S/threw -> NNS VP/threw

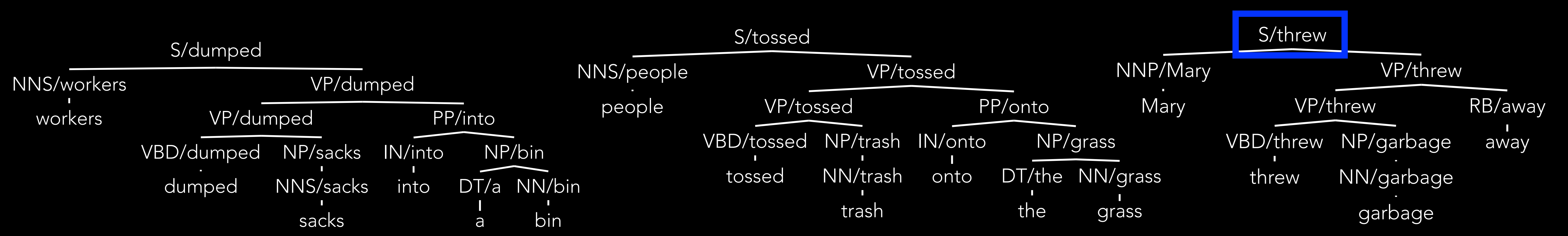
S/threw -> …

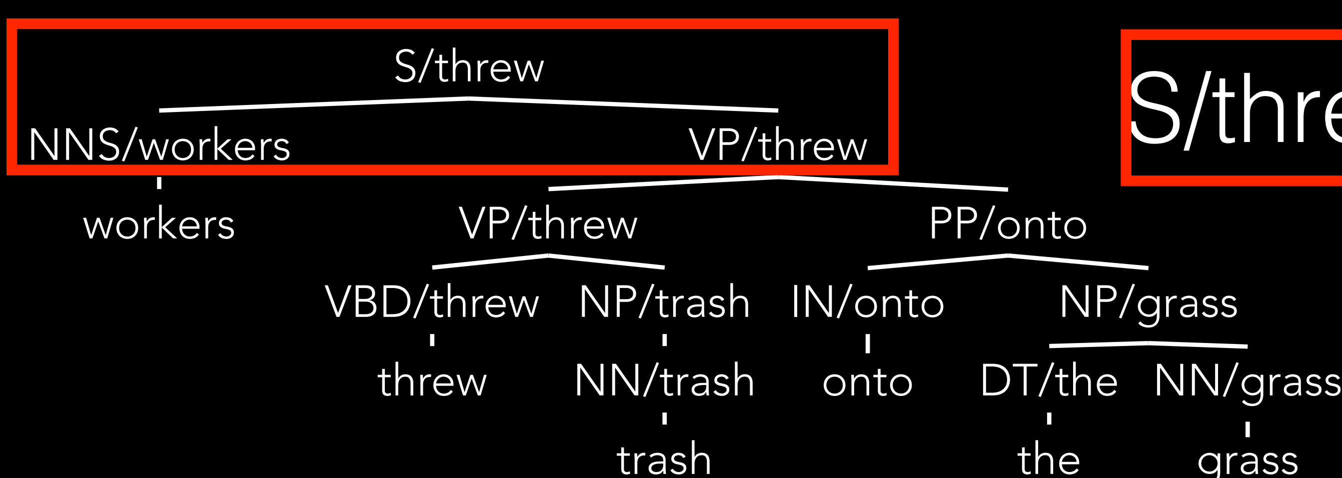
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

S/threw -> NNS VP/threw

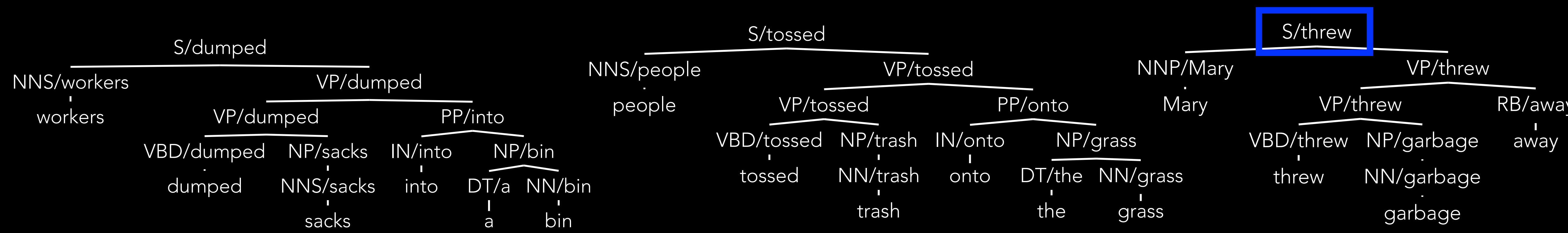
S/threw -> …

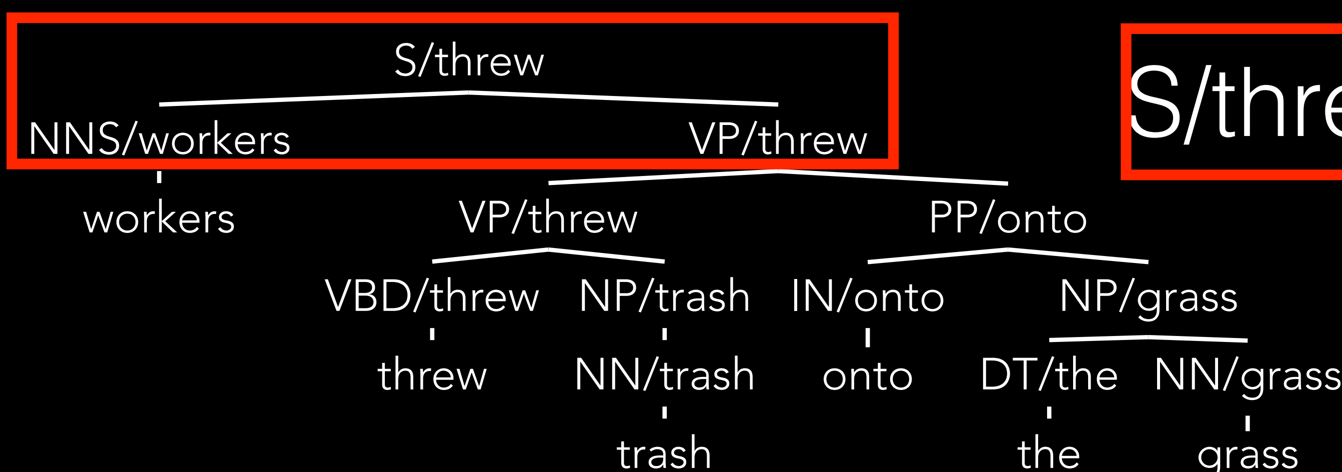
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

S/threw -> NNS VP/threw

S/threw -> …

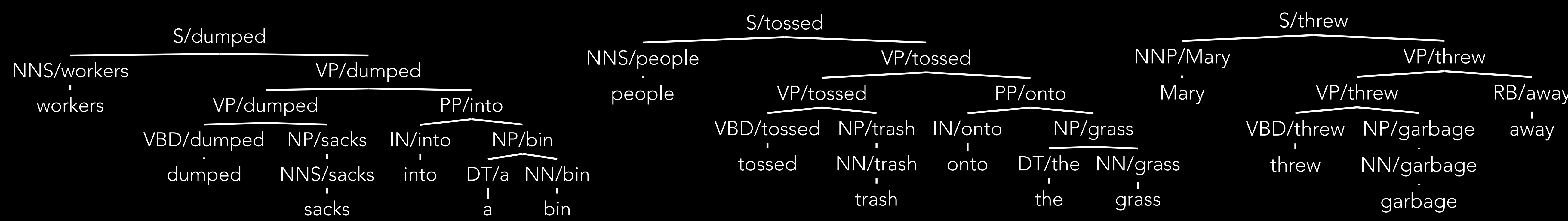
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

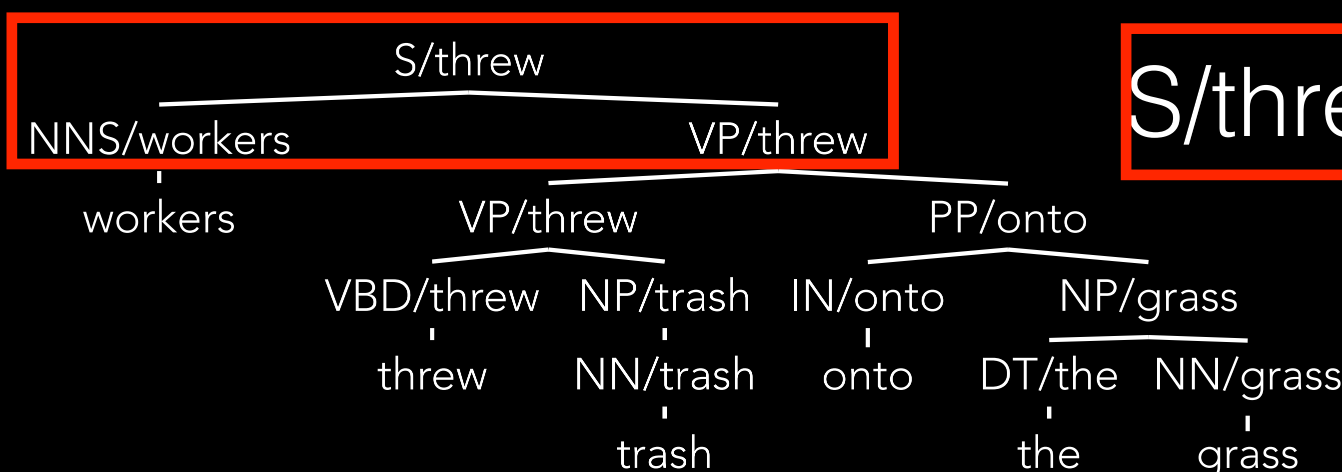
0/20

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

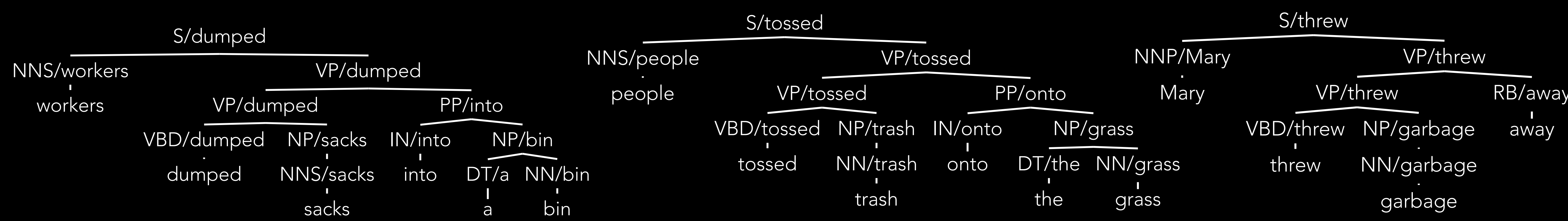
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

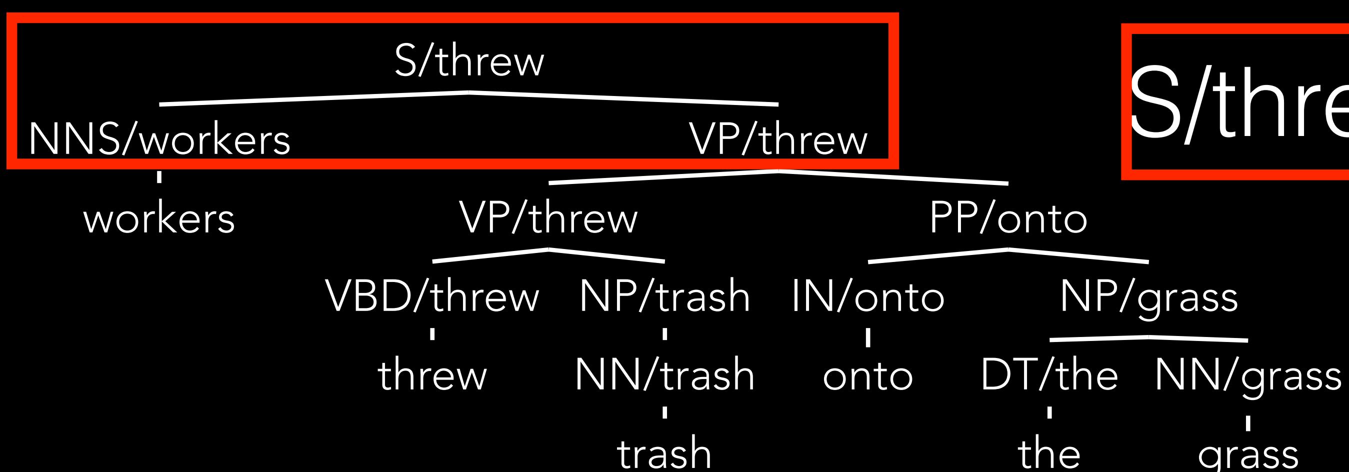
0/20

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

S -> NNS VP

S -> …

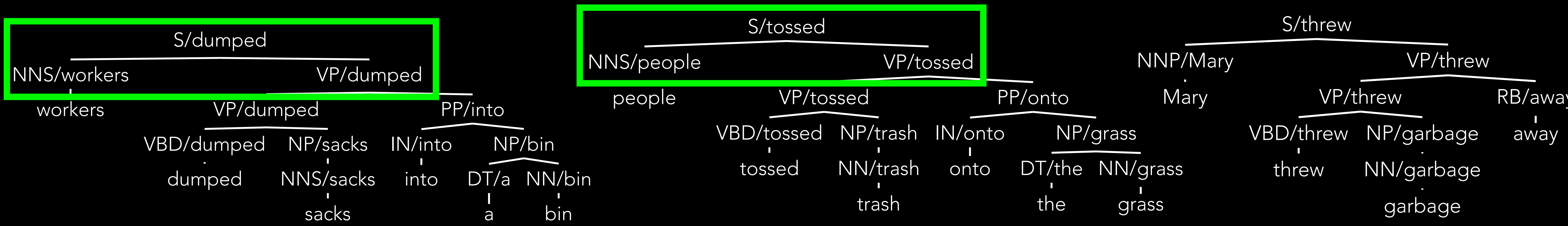
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

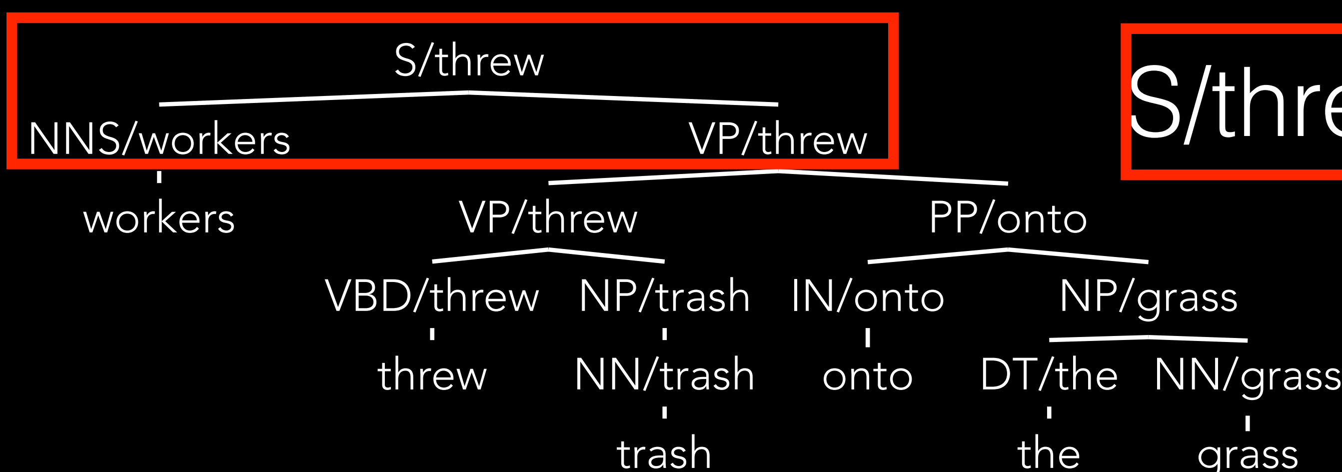
0/20

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

S -> NNS VP

S -> …

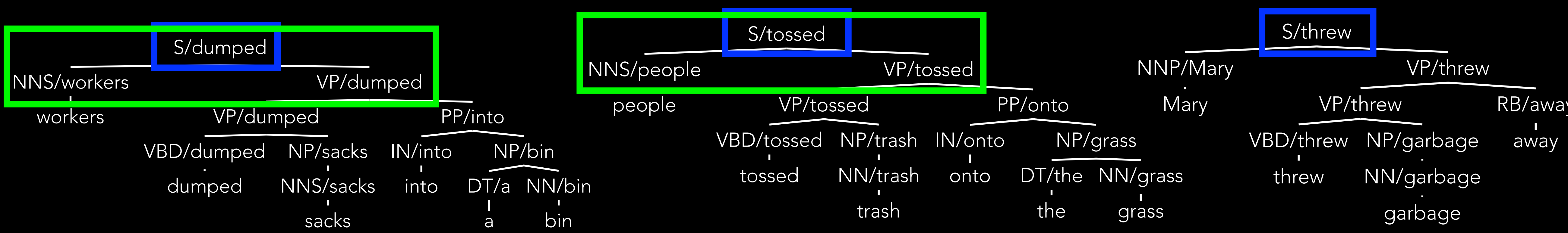
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

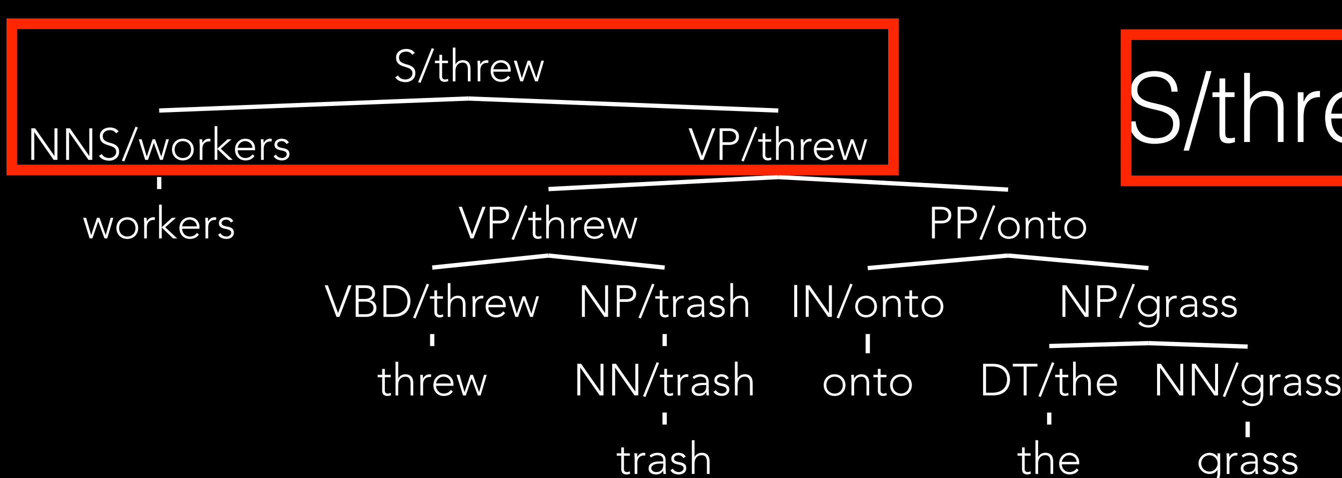
0/20

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

S -> NNS VP

S -> …

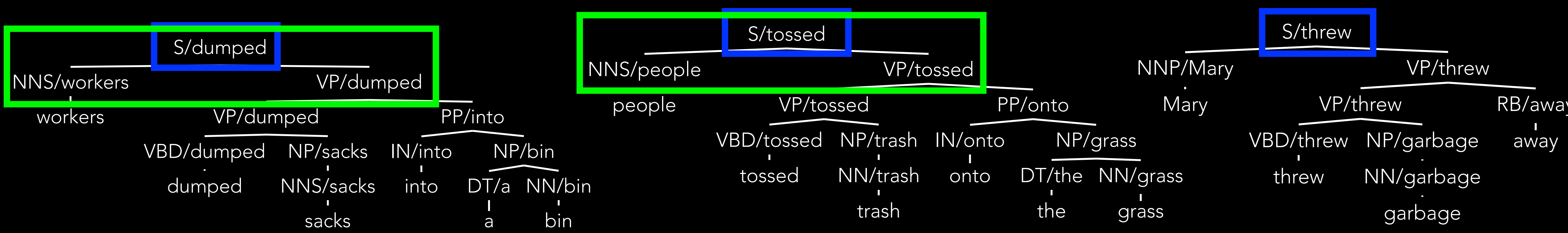
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

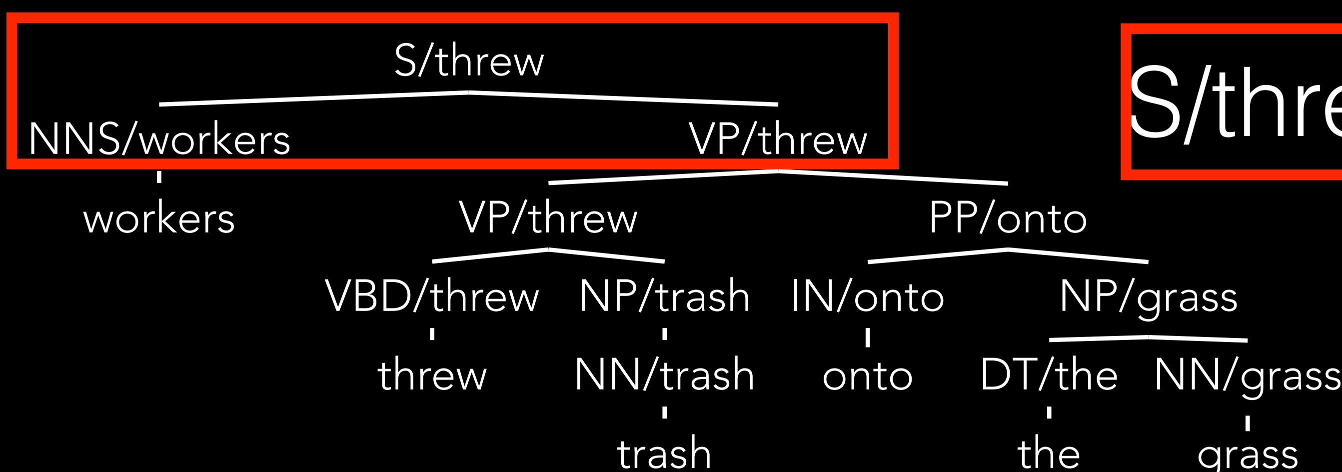
0/20

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

S -> NNS VP

S -> …

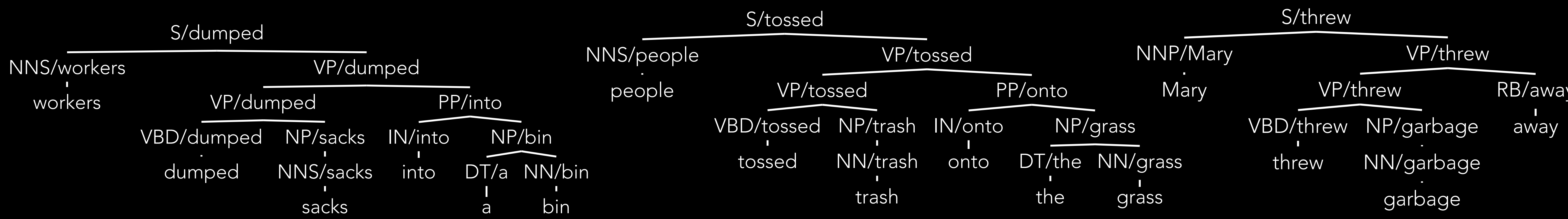
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

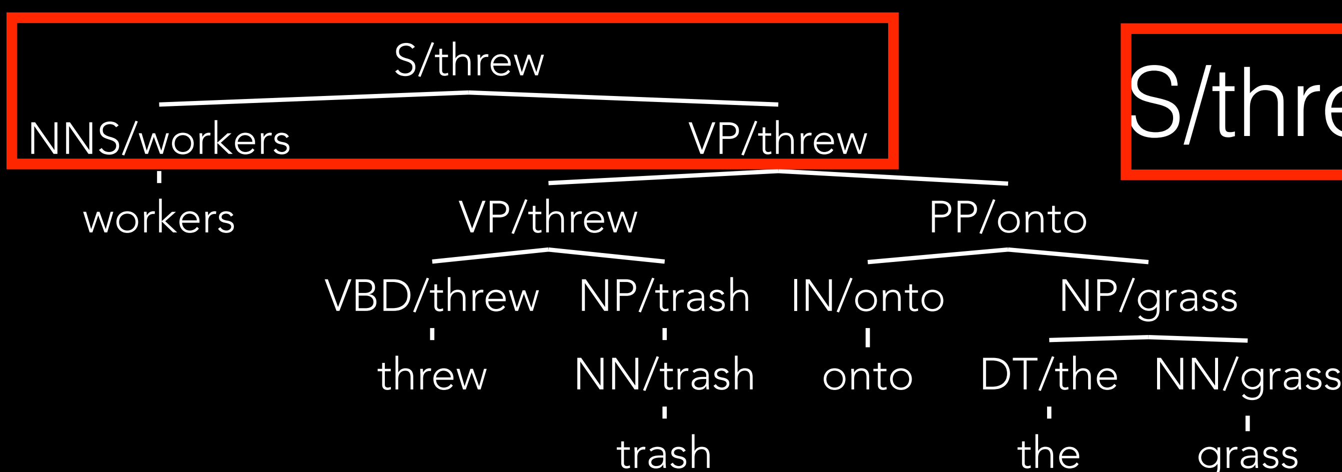
0/20 40/60

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





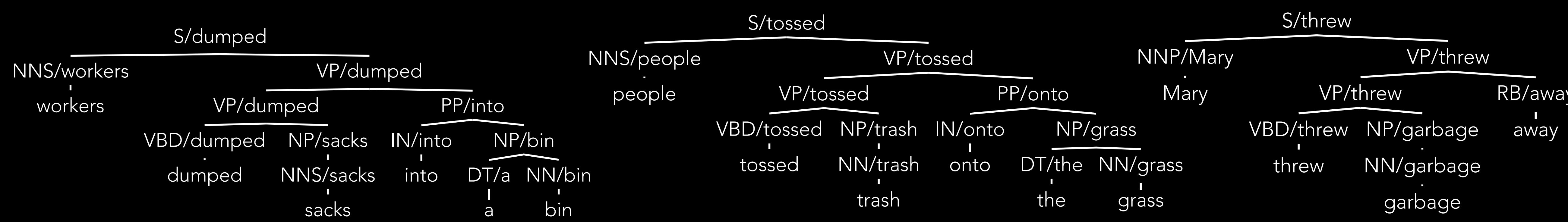
S/threw -> NNS/workers VP/threw

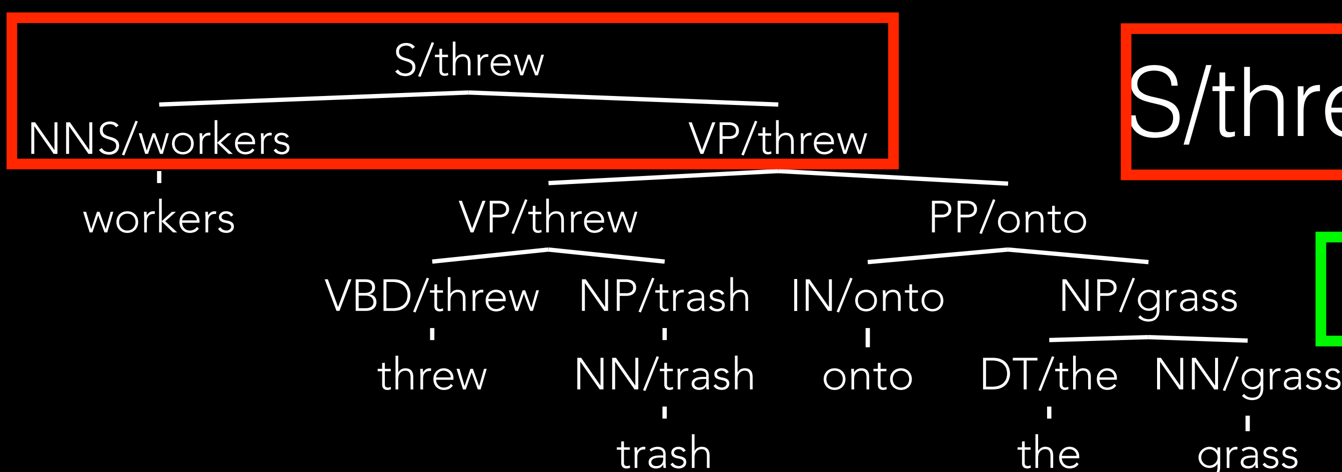
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

S/threw ->NNS/workers VP/threw

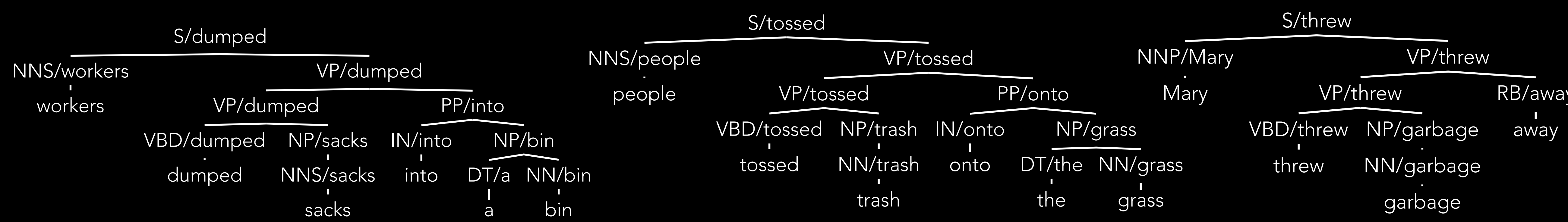
S/threw -> NNS VP/threw

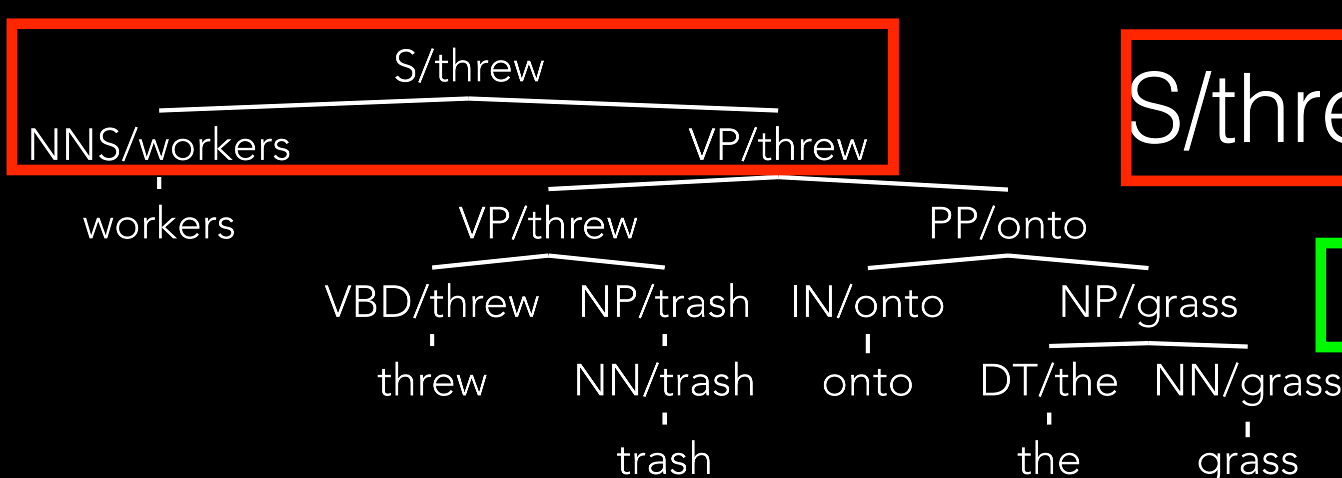
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

S/threw ->NNS/workers VP/threw

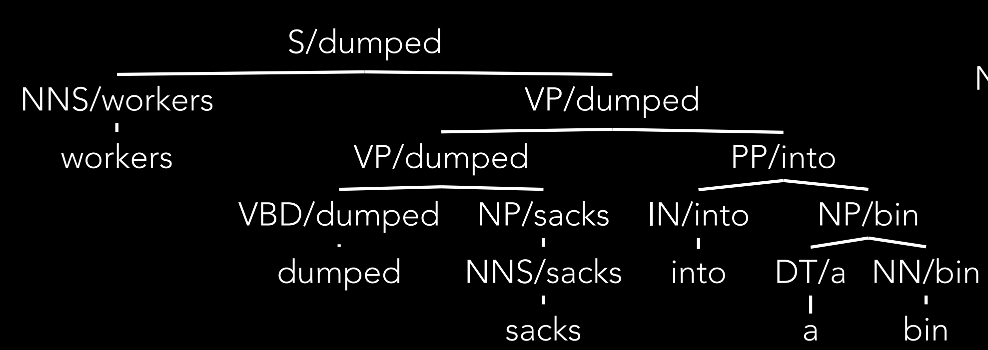
S/threw -> NNS VP/threw

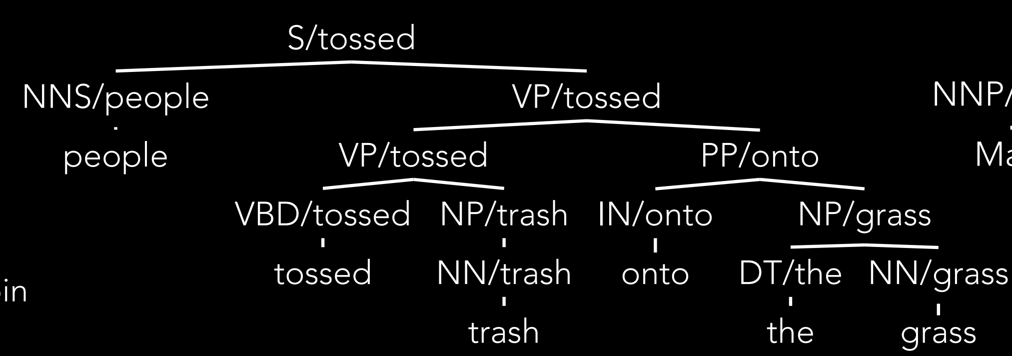
$$P(m \mid r, X, h) = \underbrace{\lambda_2 P_{ML}(m \mid r, X, h)}_{0/0} + (1-\lambda_2) P_{ML}(m \mid r)$$

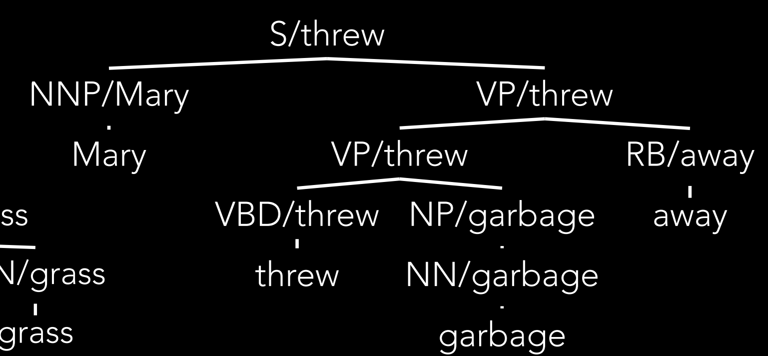
Head lexicalization

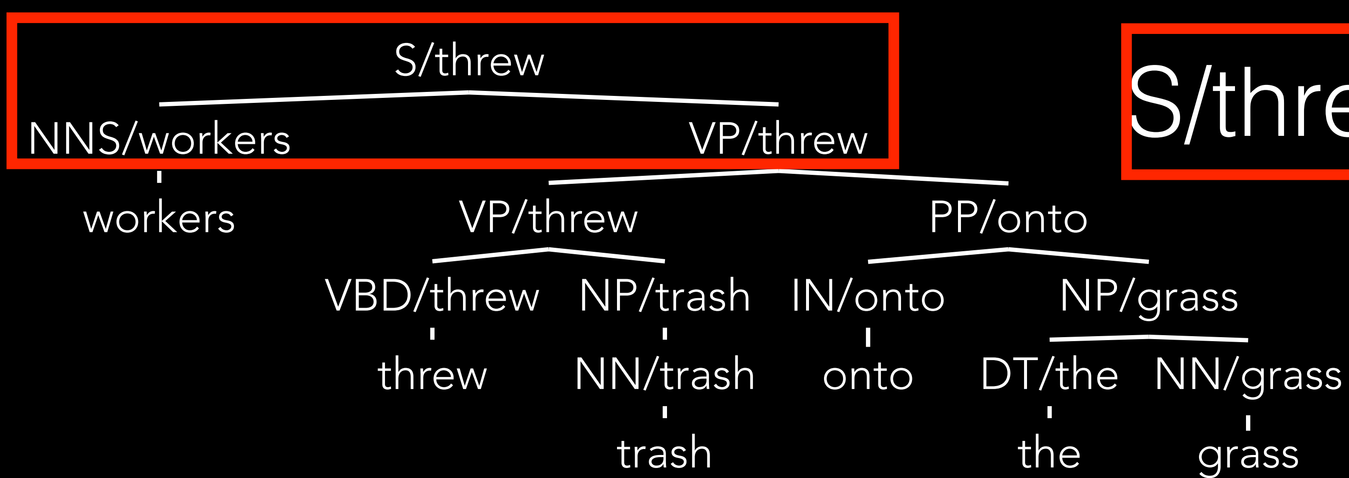
By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times









S/threw -> NNS/workers VP/threw

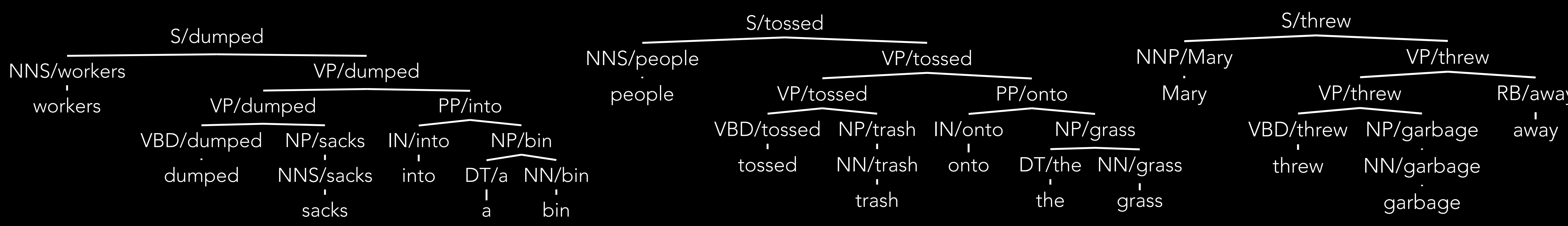
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

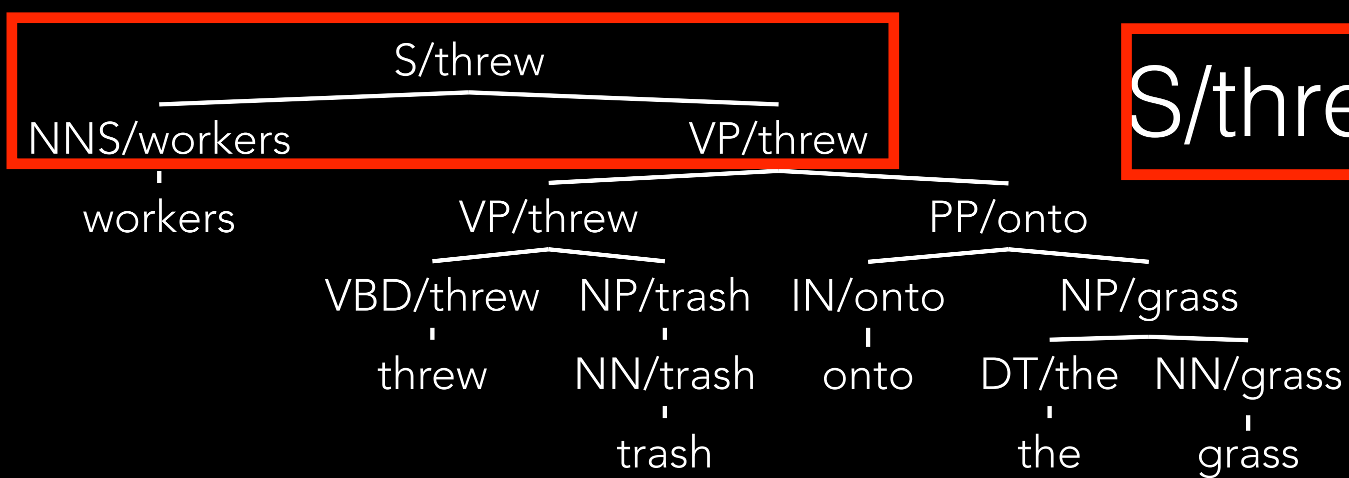
0/0

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

S -> NNS/workers VP

S -> NNS VP

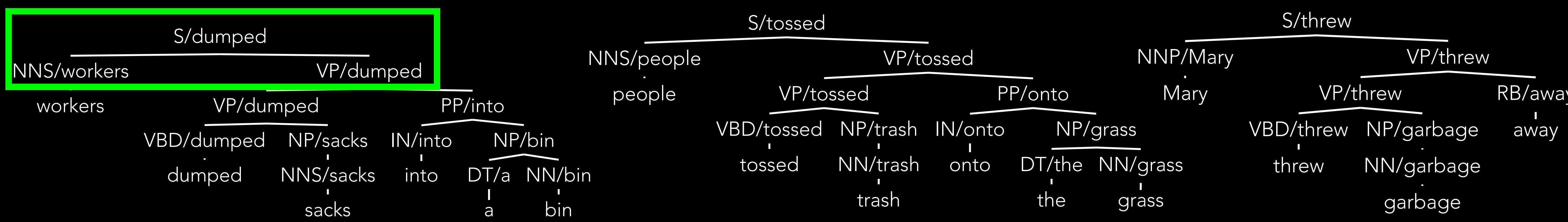
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

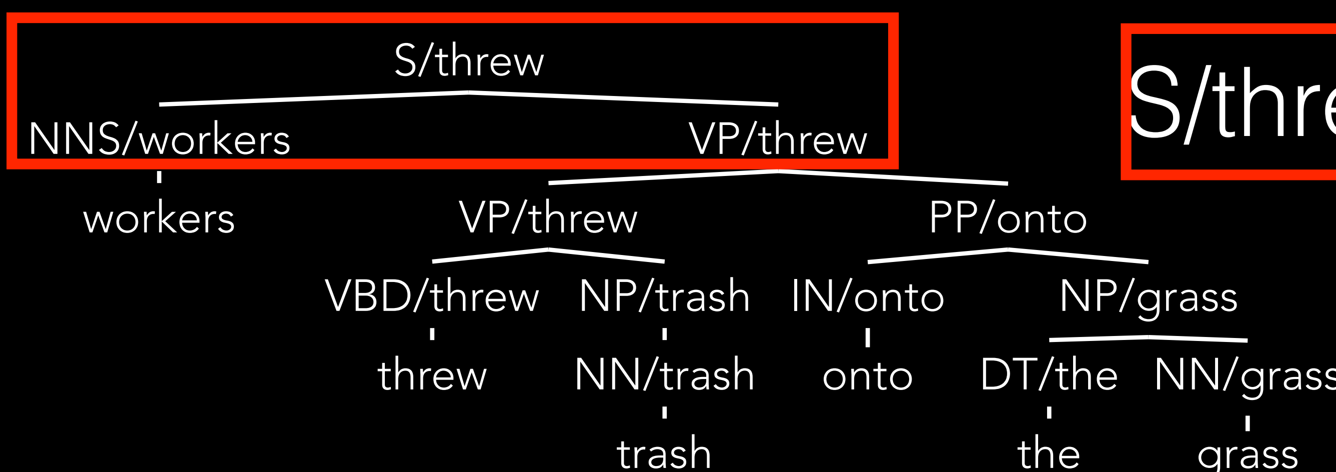
0/0

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

S -> NNS/workers VP

S -> NNS VP

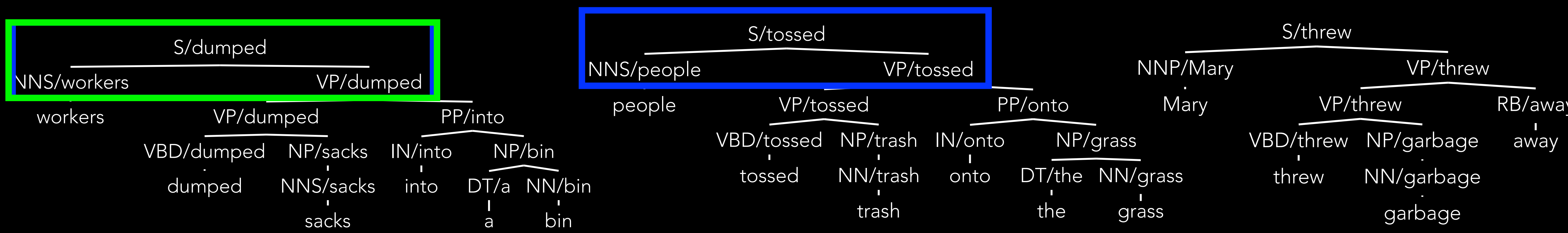
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

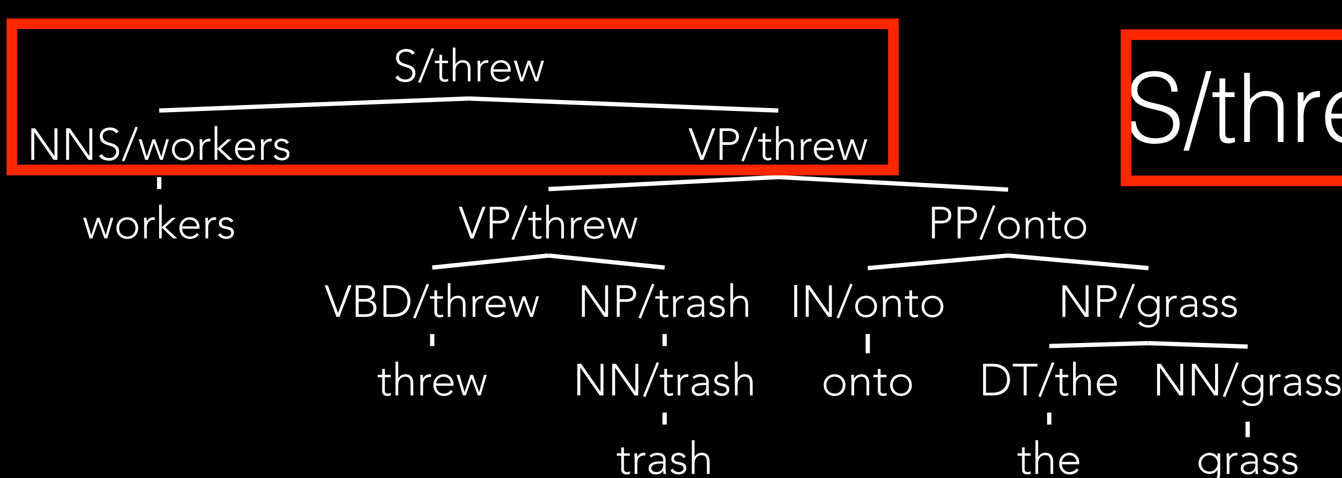
0/0

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





$\boxed{\text{S/threw -> NNS/workers VP/threw}}$

$\boxed{\text{S -> NNS/workers VP}}$ $\boxed{\text{S -> NNS VP}}$

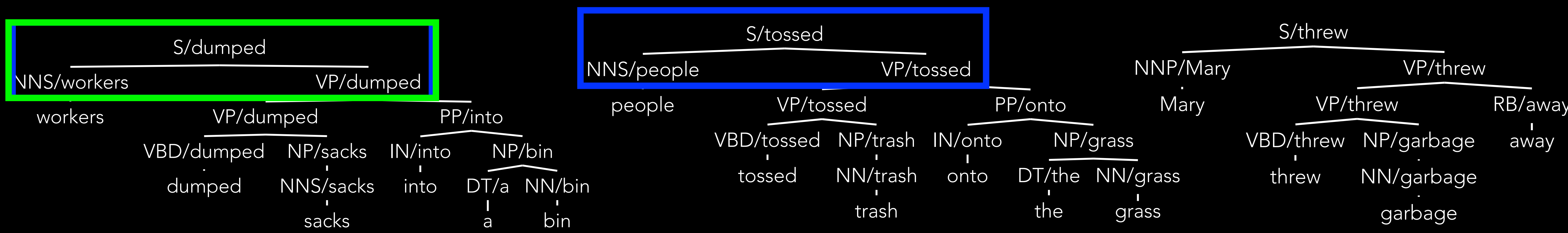
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

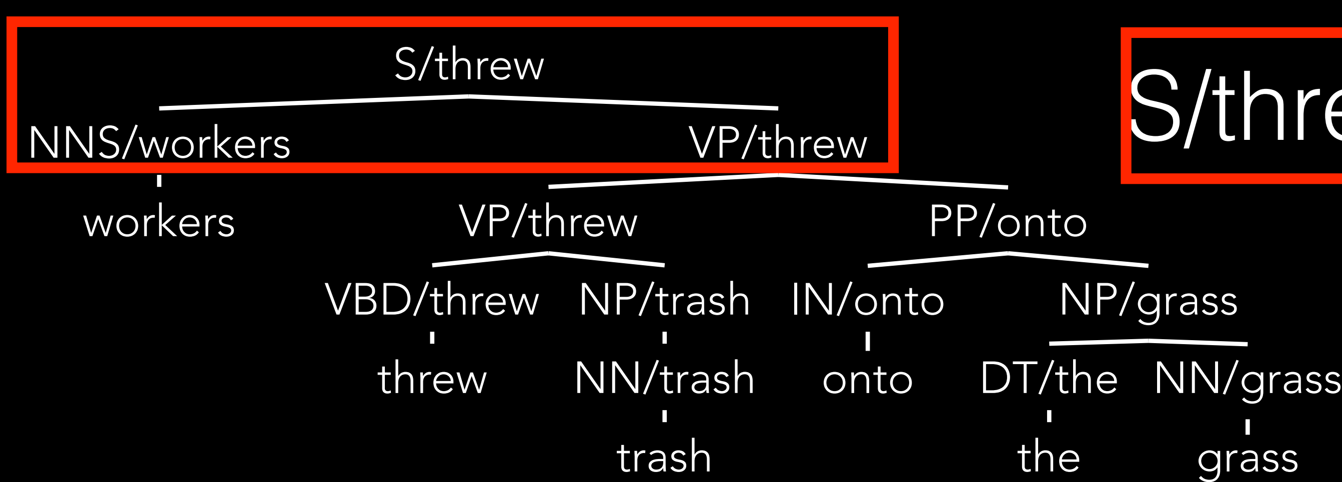
0/0

Head lexicalization

By decomposing we can estimate unseen lexicalized rules

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

S -> NNS/workers VP

S -> NNS VP

$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

0/0

20/40

Smoothing Lexicalized PCFGs

Smoothing Lexicalized PCFGs

- What about $\lambda 1$ and $\lambda 2$?

Smoothing Lexicalized PCFGs

- What about λ1 and λ2?
- Can do grid search on dev data (all 121 settings from 0.0 to 1.0 for each)

Smoothing Lexicalized PCFGs

- What about λ1 and λ2?
- Can do grid search on dev data (all 121 settings from 0.0 to 1.0 for each)
- Can do Witten-Bell smoothing

Witten-Bell Smoothing

Witten-Bell Smoothing

- I want to interpolate between a prediction of an outcome with a rich context and a prediction with less context

Witten-Bell Smoothing

- I want to interpolate between a prediction of an outcome with a rich context and a prediction with less context
- Key idea: if you haven't seen a lot of consistent examples of the rich context, trust its model less

Witten-Bell Smoothing

- I want to interpolate between a prediction of an outcome with a rich context and a prediction with less context

- Key idea: if you haven't seen a lot of consistent examples of the rich context, trust its model less

- Traditionally used in language modeling, e.g.: $p(w_i \mid w_{i-1}, w_{i-2})$ and $p(w_i \mid w_{i-1})$

Witten-Bell Smoothing

- I want to interpolate between a prediction of an outcome with a rich context and a prediction with less context

- Key idea: if you haven't seen a lot of consistent examples of the rich context, trust its model less

- Traditionally used in language modeling, e.g.: $p(w_i \mid w_{i-1}, w_{i-2})$ and $p(w_i \mid w_{i-1})$

- Here we'll apply it to lexicalization, e.g.: $P_{ML}(r \mid X, h)$ and $P_{ML}(r \mid X)$

Witten-Bell Smoothing

Witten-Bell Smoothing

- Let our prediction of the rich model is p(O | C)

Witten-Bell Smoothing

- Let our prediction of the rich model is p(O | C)
- types(C) = number of different outcomes seen with C

Witten-Bell Smoothing

- Let our prediction of the rich model is p(O | C)
- types(C) = number of different outcomes seen with C
- toks(C) = number of times C is seen

Witten-Bell Smoothing

- Let our prediction of the rich model is p(O | C)
- types(C) = number of different outcomes seen with C
- toks(C) = number of times C is seen
- λ_C = toks(C)/(types(C) + toks(C))

Witten-Bell Smoothing

- Let our prediction of the rich model is p(O | C)
- types(C) = number of different outcomes seen with C
- toks(C) = number of times C is seen
- λ_C = toks(C)/(types(C) + toks(C))
- count(C, O_1) = 40
  count(C, O_2) = 10
  types(C) = 2; toks(C) = 50; λ_C= 50/52

Witten-Bell Smoothing

- Let our prediction of the rich model is p(O | C)
- types(C) = number of different outcomes seen with C
- toks(C) = number of times C is seen
- λ_C = toks(C)/(types(C) + toks(C))
- count(C, O_1) = 40
  count(C, O_2) = 10
  types(C) = 2; toks(C) = 50; λ_C= 50/52
- least confident: saw n things 1 time each; λ = n/2n = 0.5

Witten-Bell Smoothing

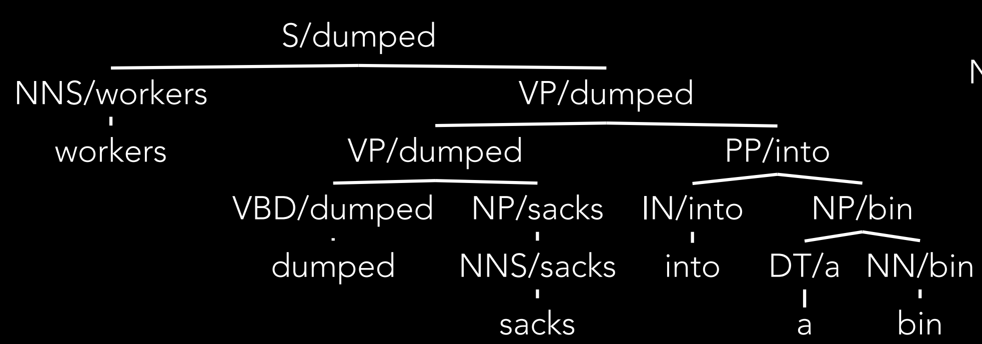
- Let our prediction of the rich model is p(O | C)
- types(C) = number of different outcomes seen with C
- toks(C) = number of times C is seen
- λ_C = toks(C)/(types(C) + toks(C))
- count(C, O_1) = 40
  count(C, O_2) = 10
  types(C) = 2; toks(C) = 50; λ_C= 50/52
- least confident: saw n things 1 time each; λ = n/2n = 0.5
- most confident: saw 1 thing n times; λ = n/(n+1) (more n = more confident)

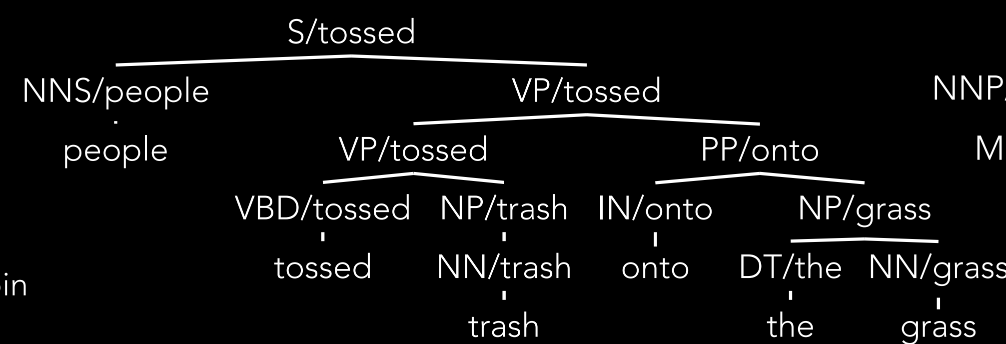
Witten-Bell Smoothing

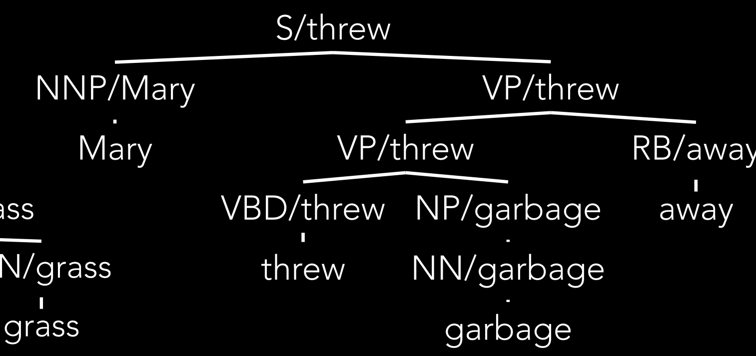
- Let our prediction of the rich model is p(O | C)
- types(C) = number of different outcomes seen with C
- toks(C) = number of times C is seen
- λ_C = toks(C)/(types(C) + toks(C))
- count(C, O_1) = 40
  count(C, O_2) = 10
  types(C) = 2; toks(C) = 50; λ_C= 50/52
- least confident: saw n things 1 time each; λ = n/2n = 0.5
- most confident: saw 1 thing n times; λ = n/(n+1) (more n = more confident)
- λ_C=0 if C never seen

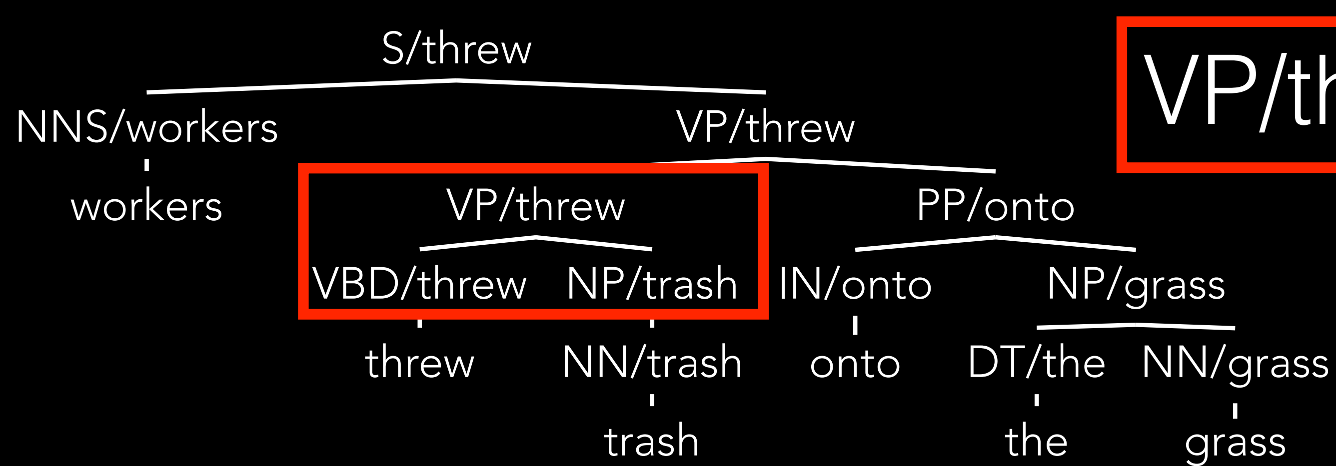
Incorporating Smoothing

Assume we saw each training tree 20 times









VP/threw -> VBD/threw NP/trash

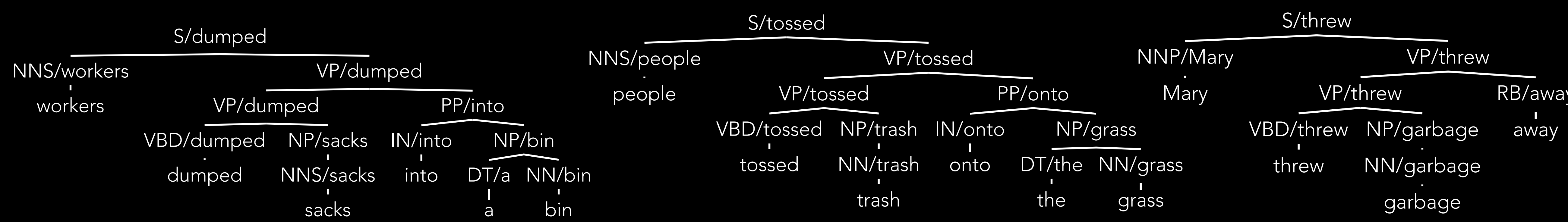
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

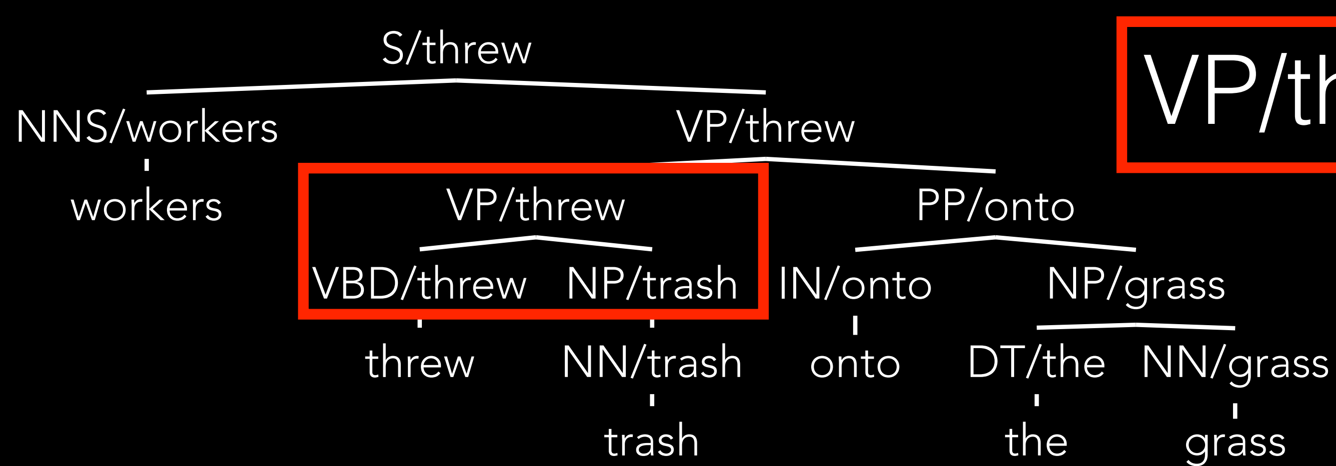
20/40

60/120

Incorporating Smoothing

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

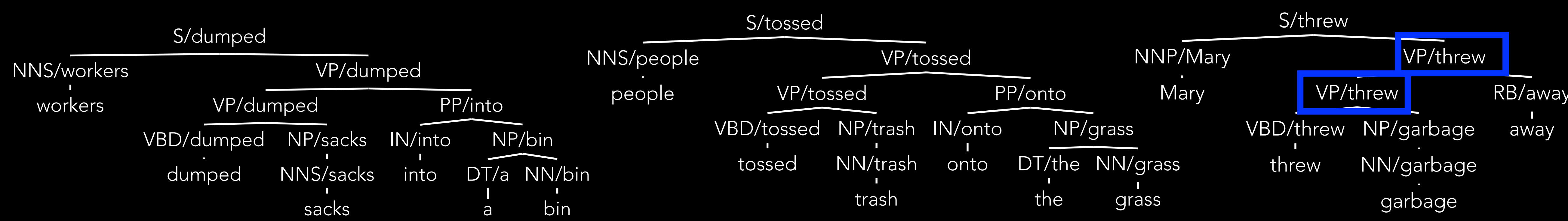
$$P(r \mid \boxed{X, h}) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

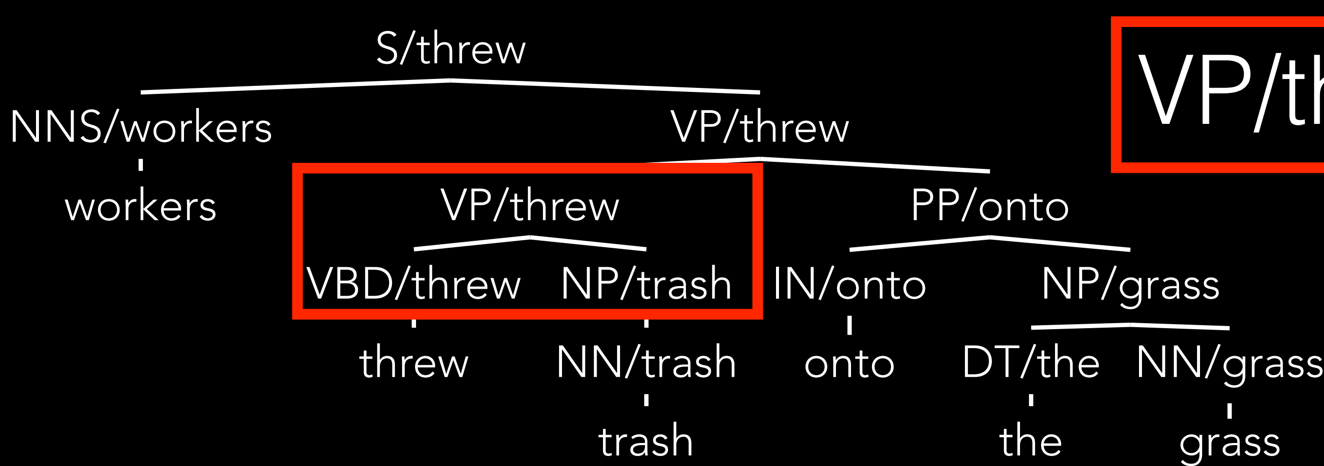
20/40

60/120

Incorporating Smoothing

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

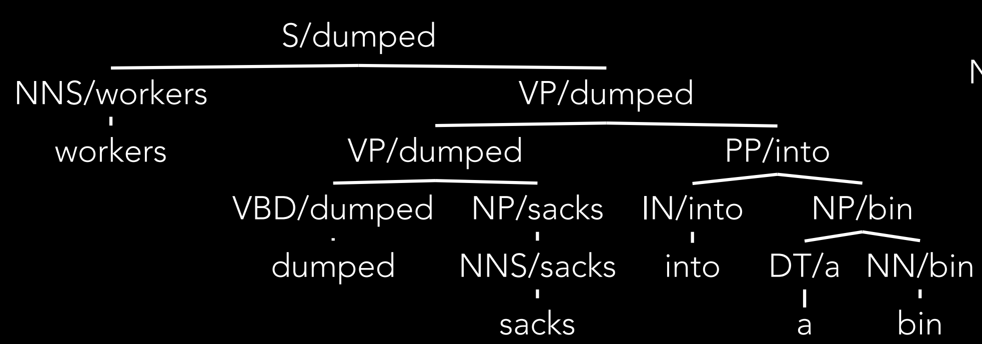
$$P(r \mid \boxed{X, h}) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

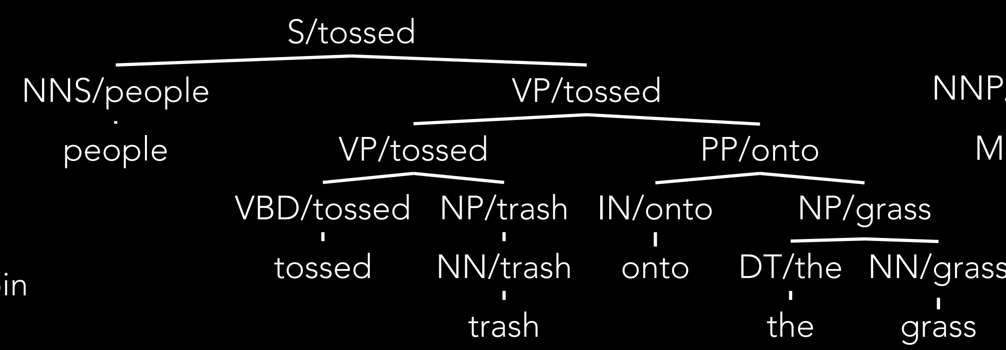
20/40

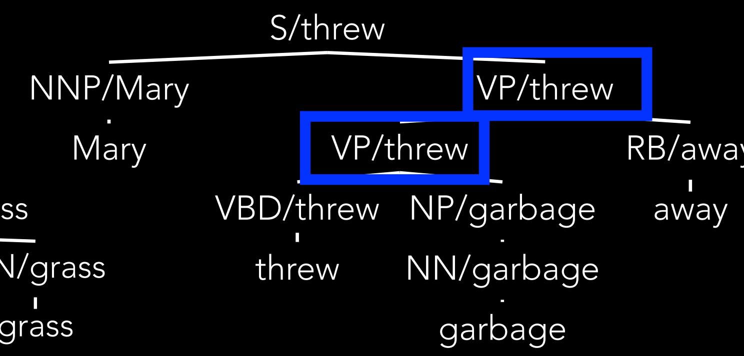
60/120

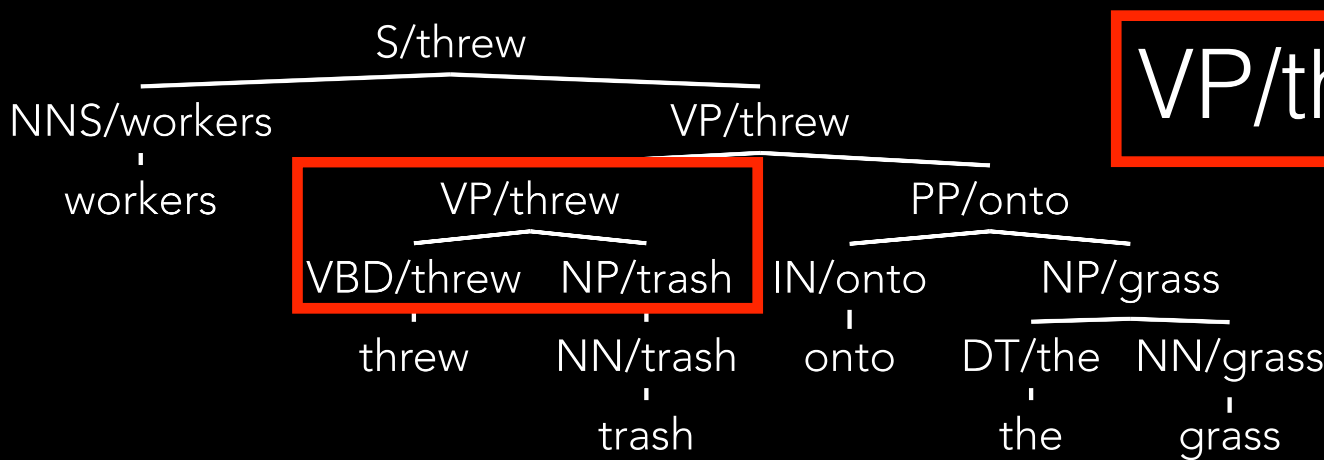
Incorporating Smoothing

Assume we saw each training tree 20 times









VP/threw -> VBD/threw NP/trash

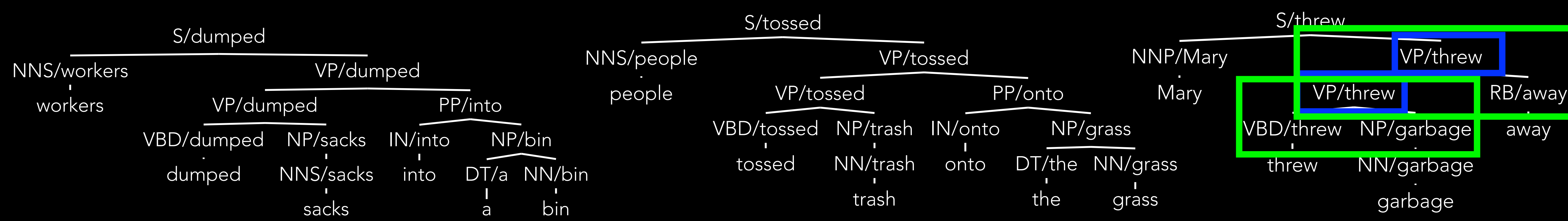
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

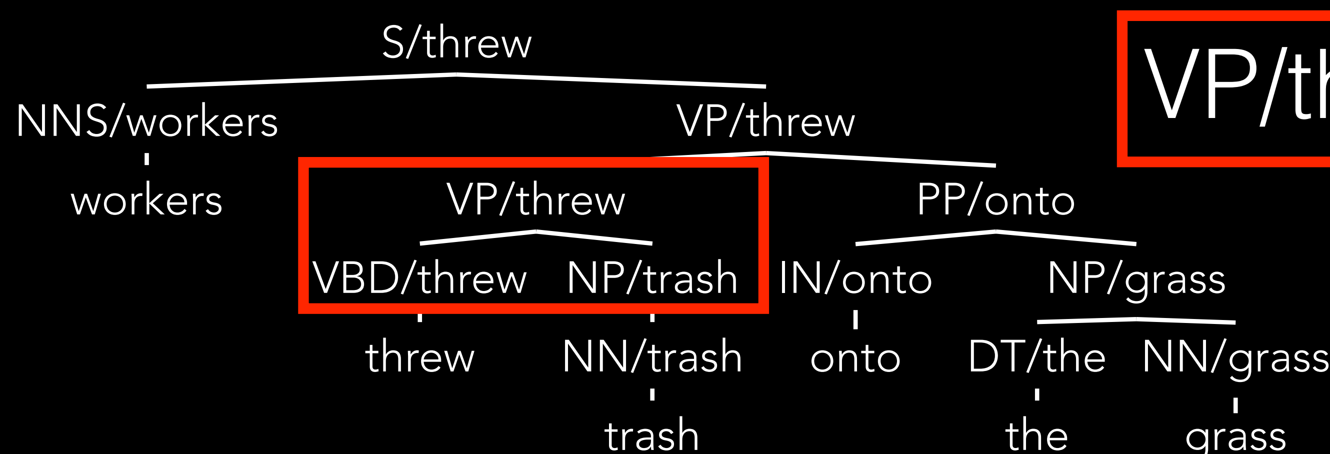
20/40

60/120

Incorporating Smoothing

Assume we saw each training tree 20 times





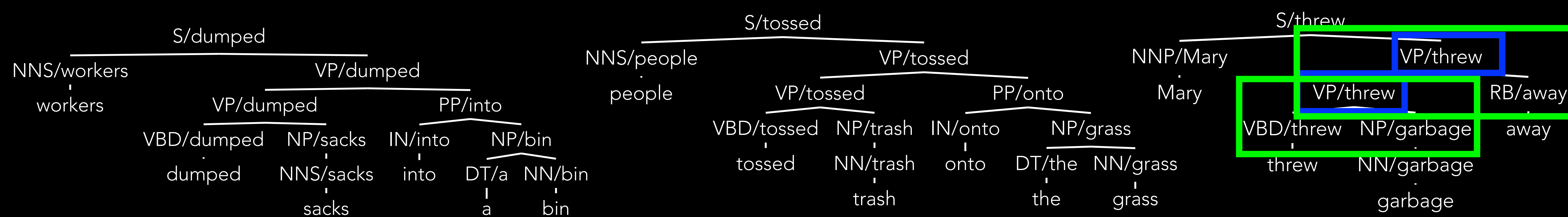
VP/threw -> VBD/threw NP/trash

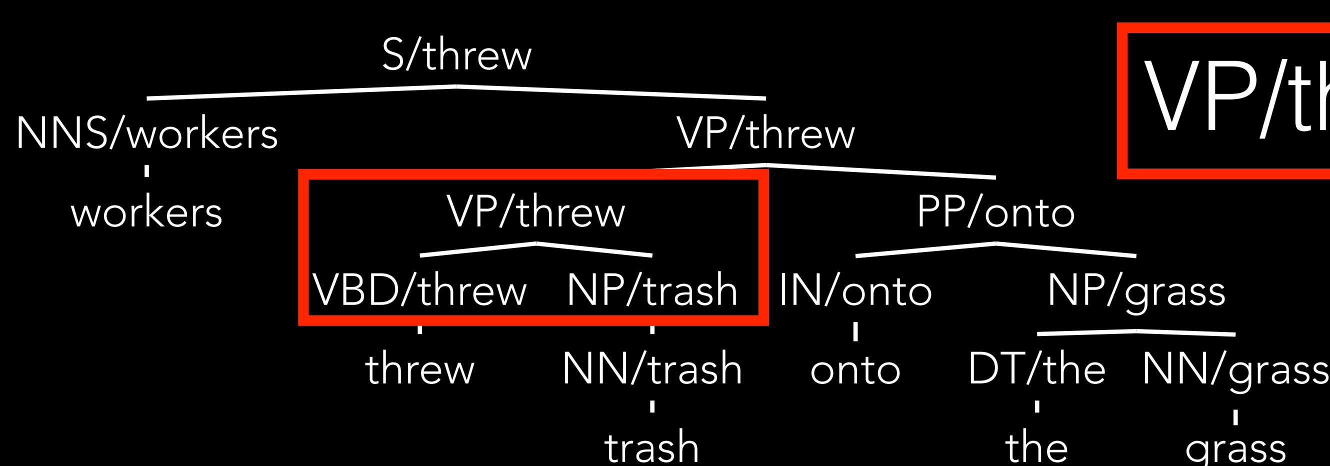
$$P(\boxed{r} \mid \boxed{X, h}) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

20/40 60/120

Incorporating Smoothing

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

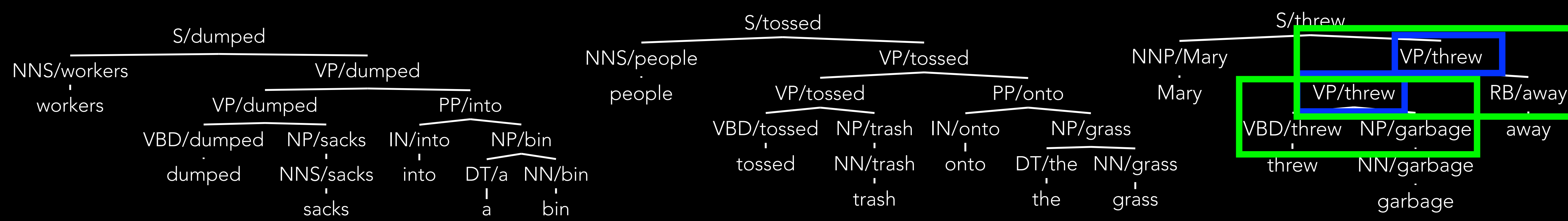
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

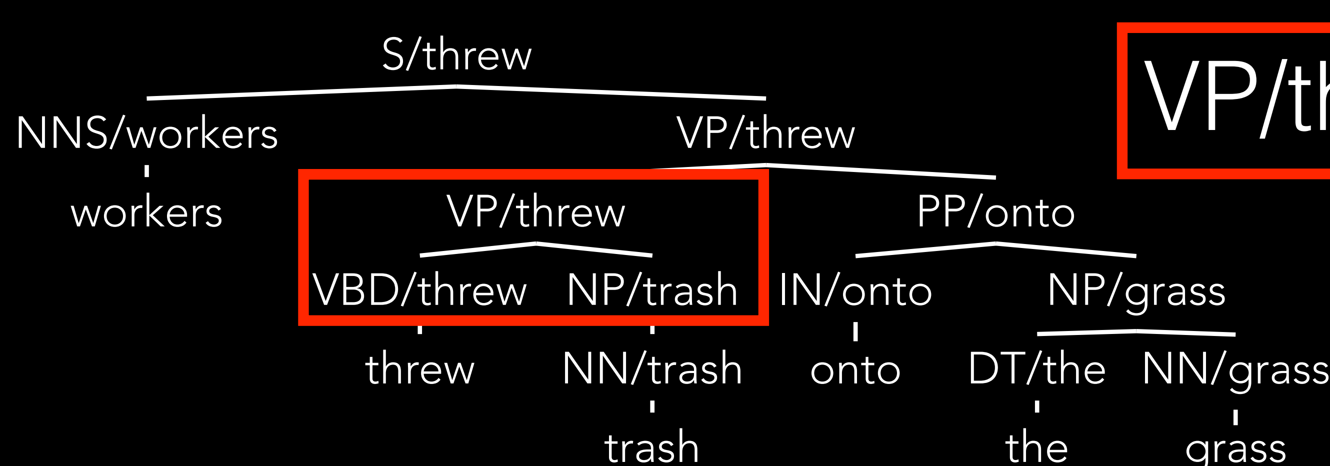
40/42 x 20/40

60/120

Incorporating Smoothing

Assume we saw each training tree 20 times





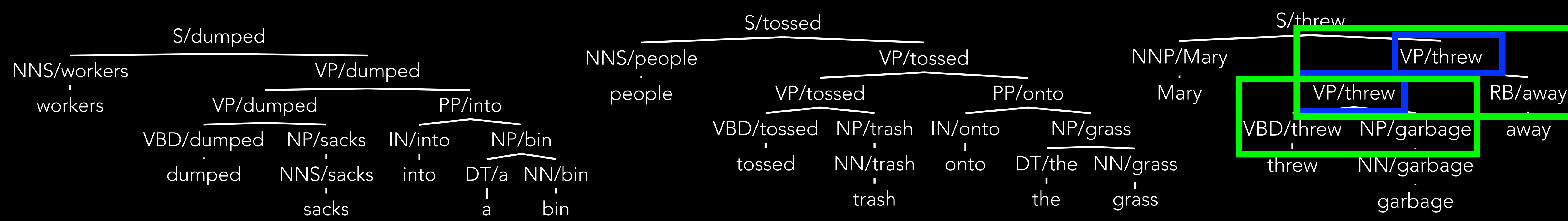
VP/threw -> VBD/threw NP/trash

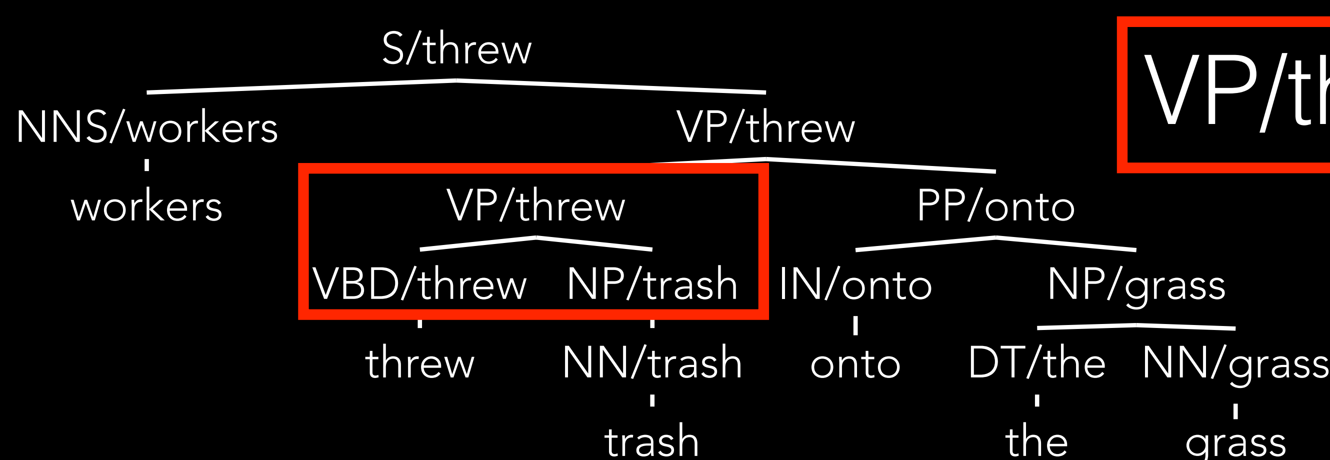
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

40/42 x 20/40   2/42 x 60/120

Incorporating Smoothing

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

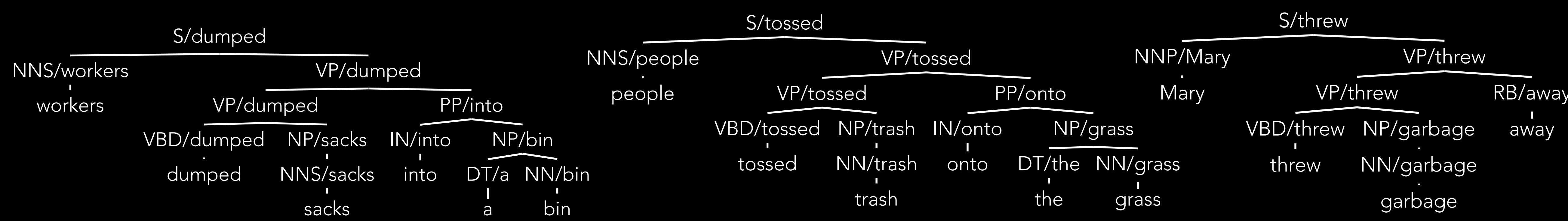
$$P(\boxed{r} \mid \boxed{X, h}) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

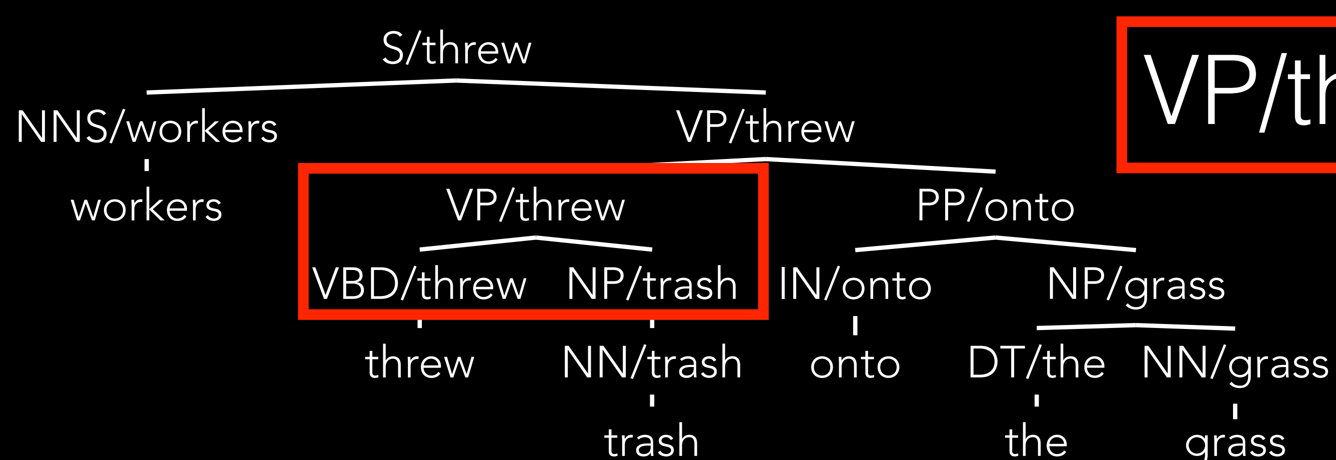
40/42 x 20/40   2/42 x 60/120

=0.5

Incorporating Smoothing

Assume we saw each training tree 20 times





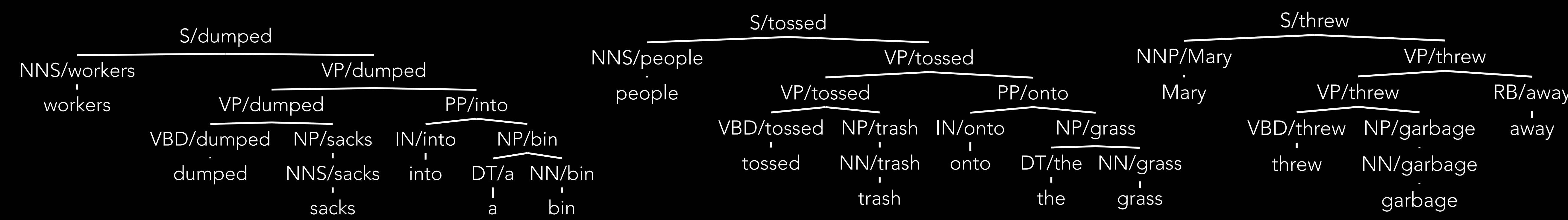
$\boxed{\text{VP/threw -> VBD/threw NP/trash}}$

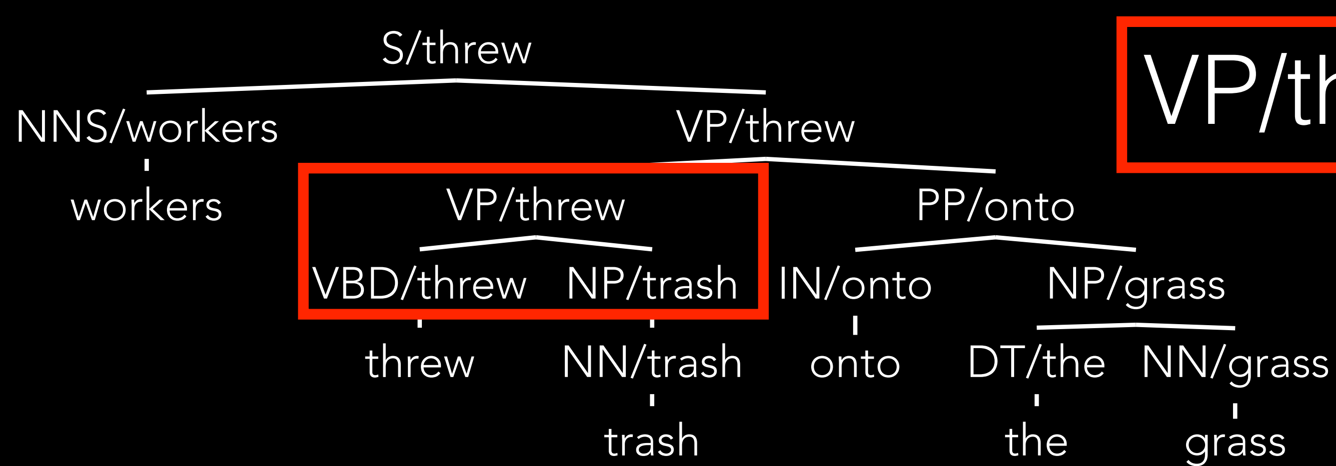
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

0/20 20/60

Incorporating Smoothing

Assume we saw each training tree 20 times





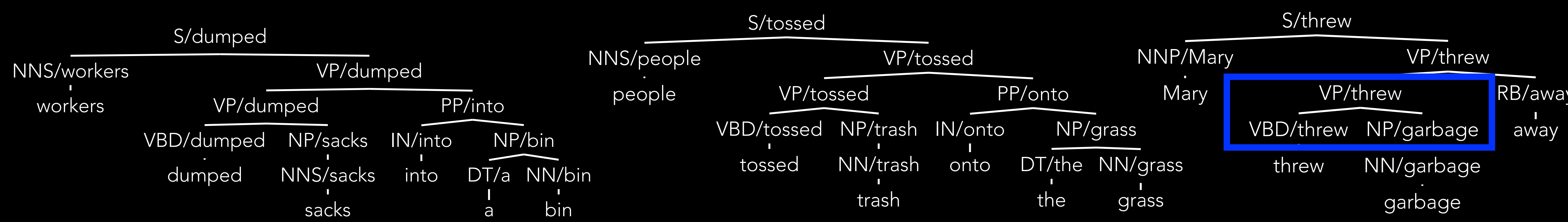
VP/threw -> VBD/threw NP/trash

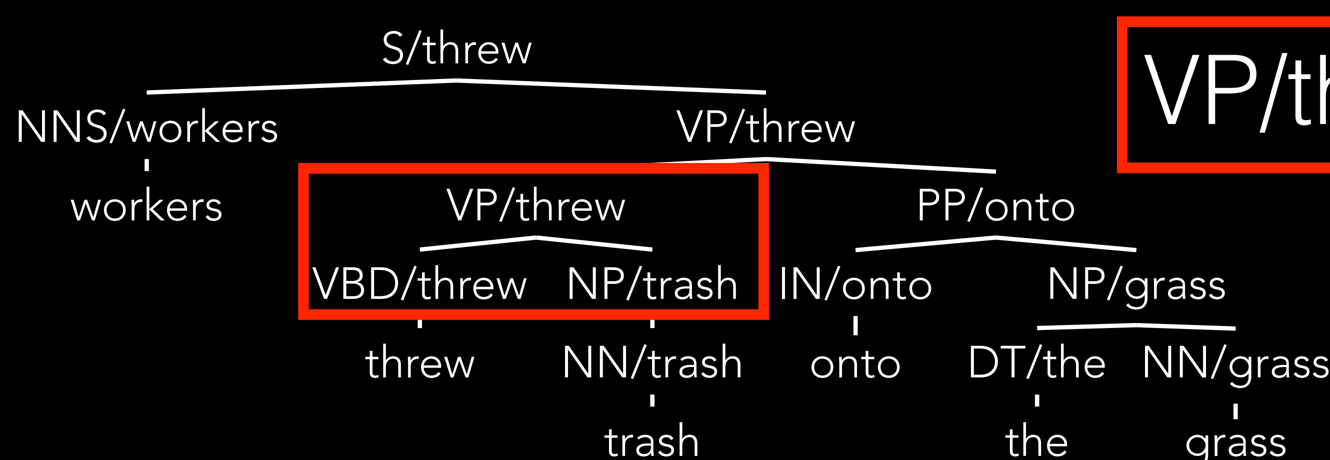
$$P(m \mid \boxed{r, X, h}) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

0/20 20/60

Incorporating Smoothing

Assume we saw each training tree 20 times





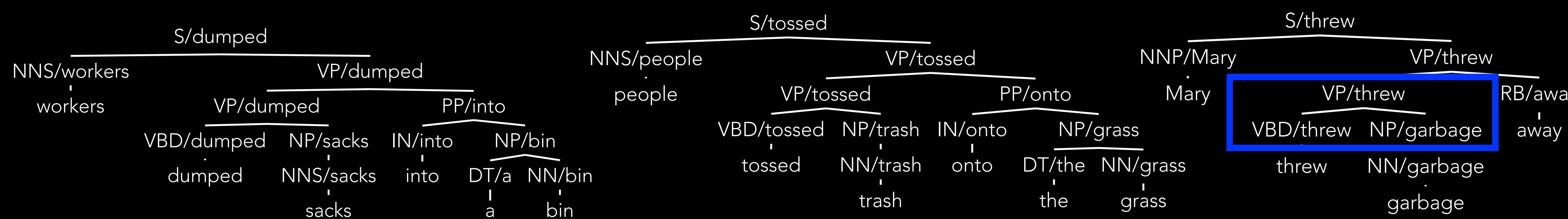
VP/threw -> VBD/threw NP/trash

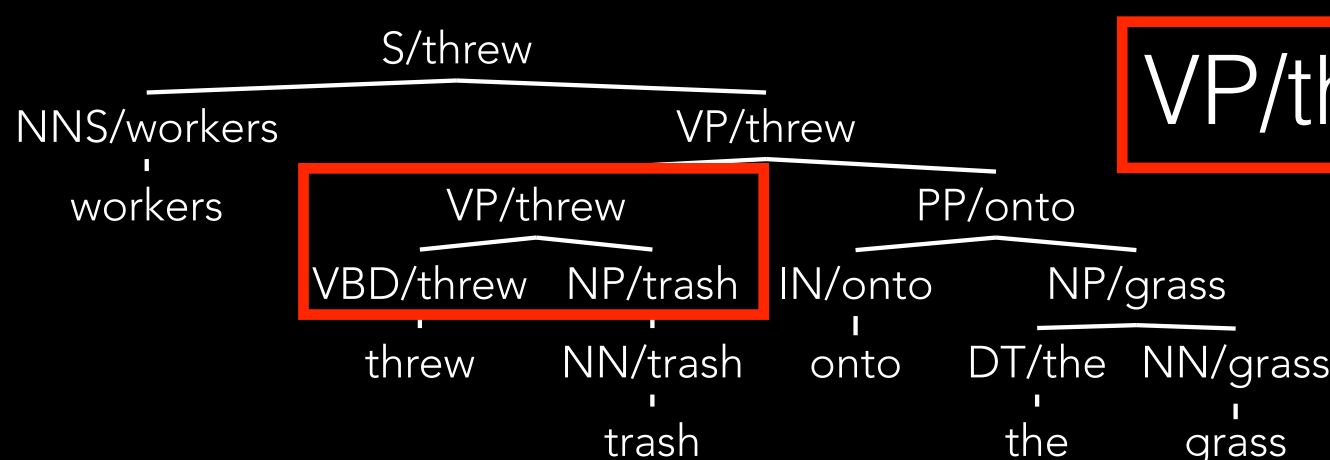
$$P(m \mid \boxed{r, X, h}) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

0/20 20/60

Incorporating Smoothing

Assume we saw each training tree 20 times





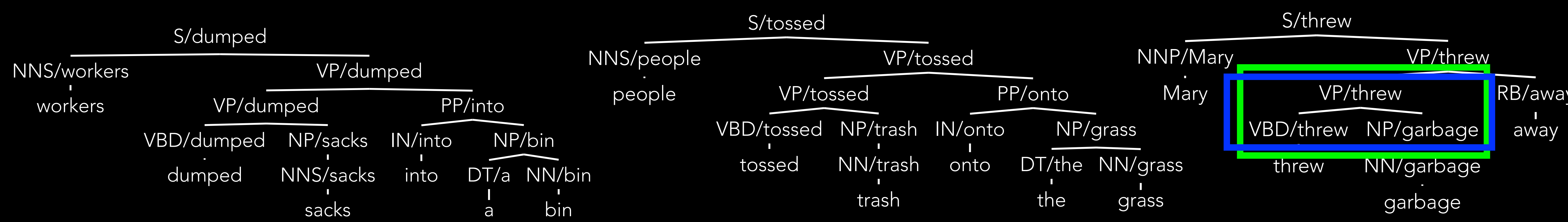
VP/threw -> VBD/threw NP/trash

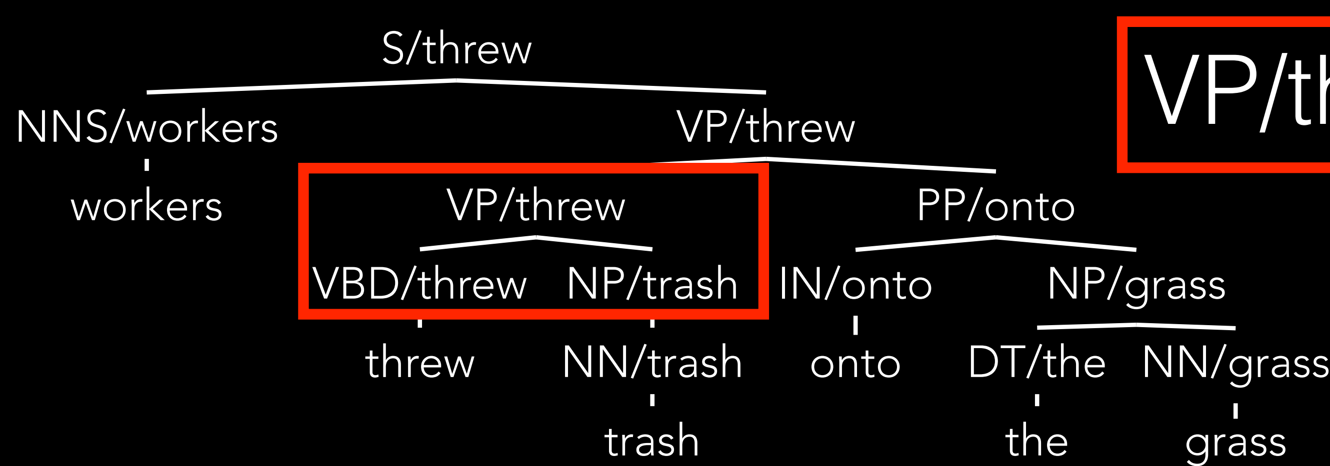
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

0/20 20/60

Incorporating Smoothing

Assume we saw each training tree 20 times





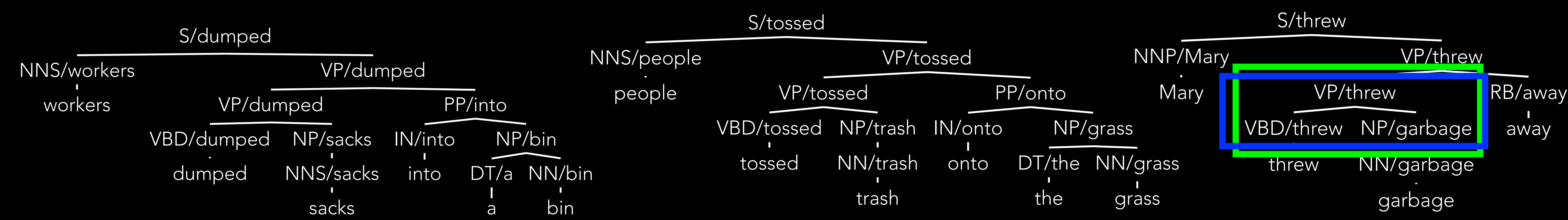
VP/threw -> VBD/threw NP/trash

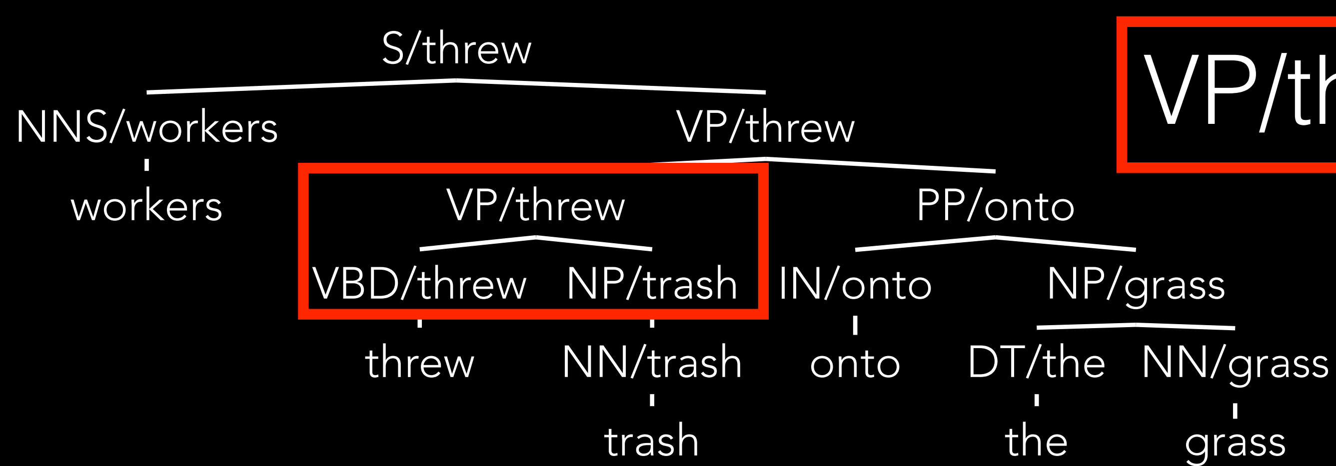
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

0/20 20/60

Incorporating Smoothing

Assume we saw each training tree 20 times





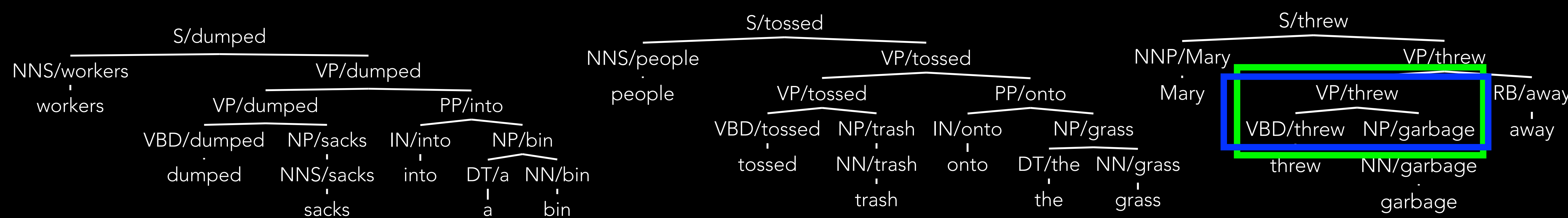
VP/threw -> VBD/threw NP/trash

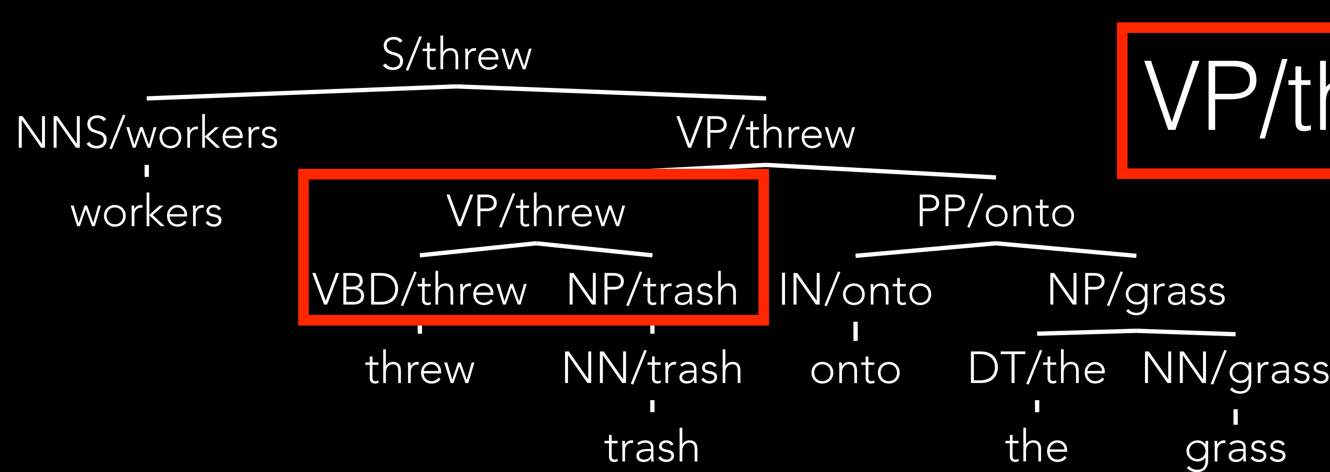
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

20/21 x 0/20 20/60

Incorporating Smoothing

Assume we saw each training tree 20 times





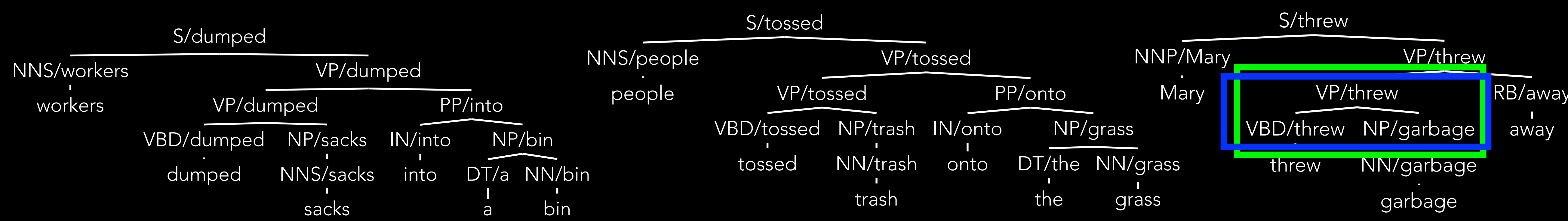
VP/threw -> VBD/threw NP/trash

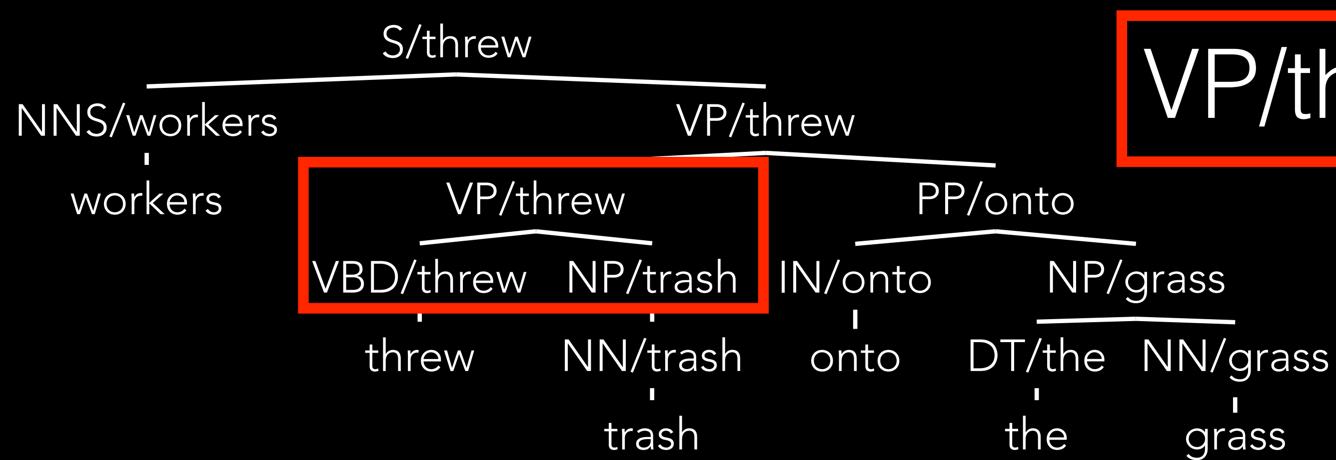
$$P(\boxed{m} \mid \boxed{r, X, h}) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

$$20/21 \times 0/20 \qquad 1/21 \times 20/60$$

Incorporating Smoothing

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

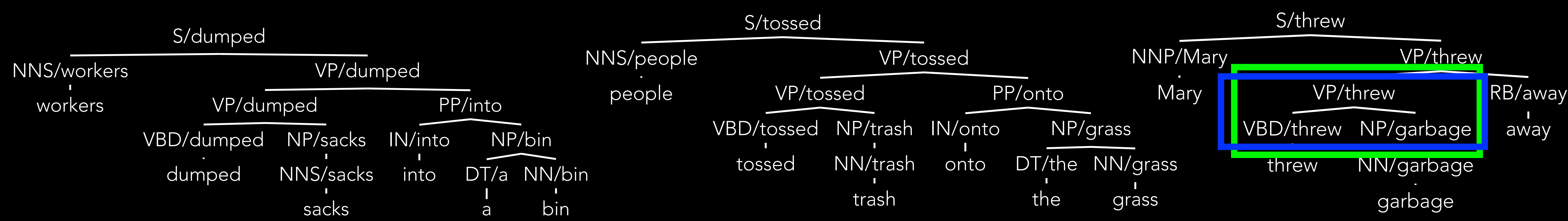
$$P(\boxed{m} \mid \boxed{r, X, h}) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

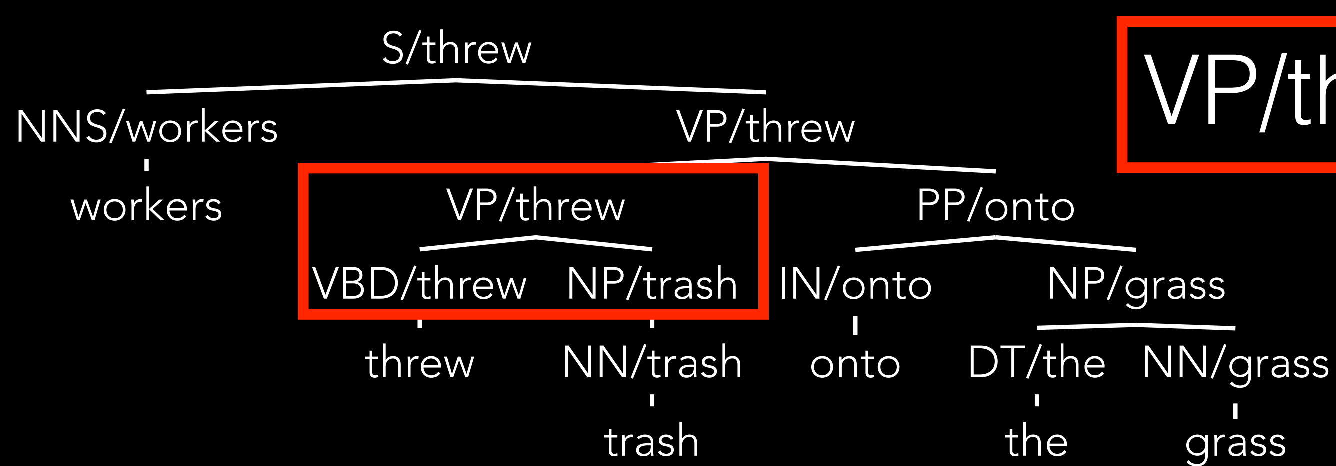
$$20/21 \times 0/20 \qquad 1/21 \times 20/60$$

$= .016$

Incorporating Smoothing

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

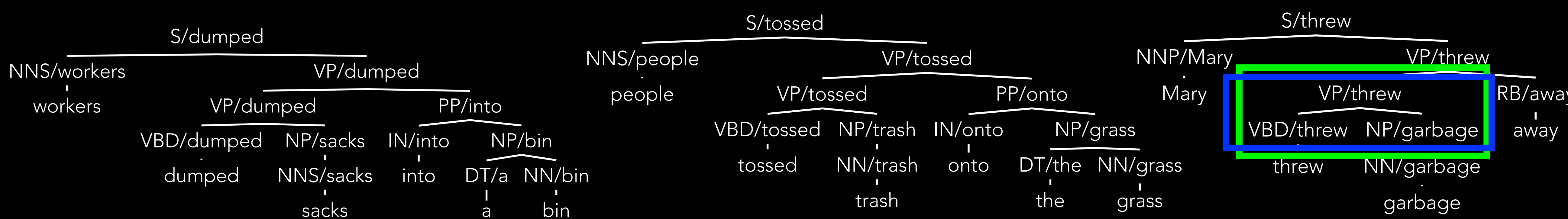
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

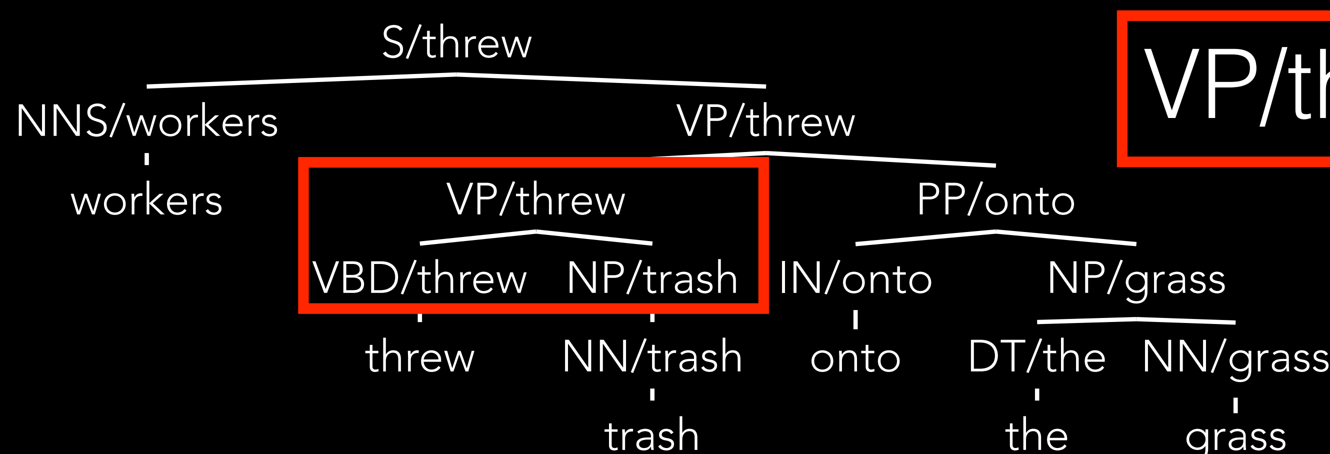
20/21 x 0/20 1/21 x 20/60

=.016 VP/threw -> VBD/threw NP/trash

Incorporating Smoothing

Assume we saw each training tree 20 times





VP/threw -> VBD/threw NP/trash

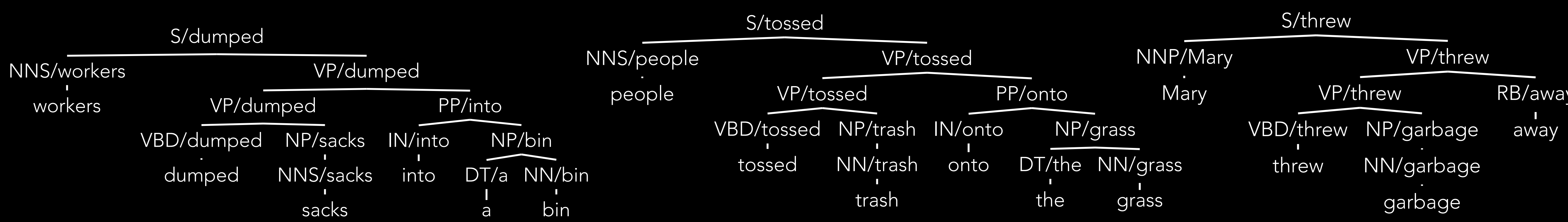
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

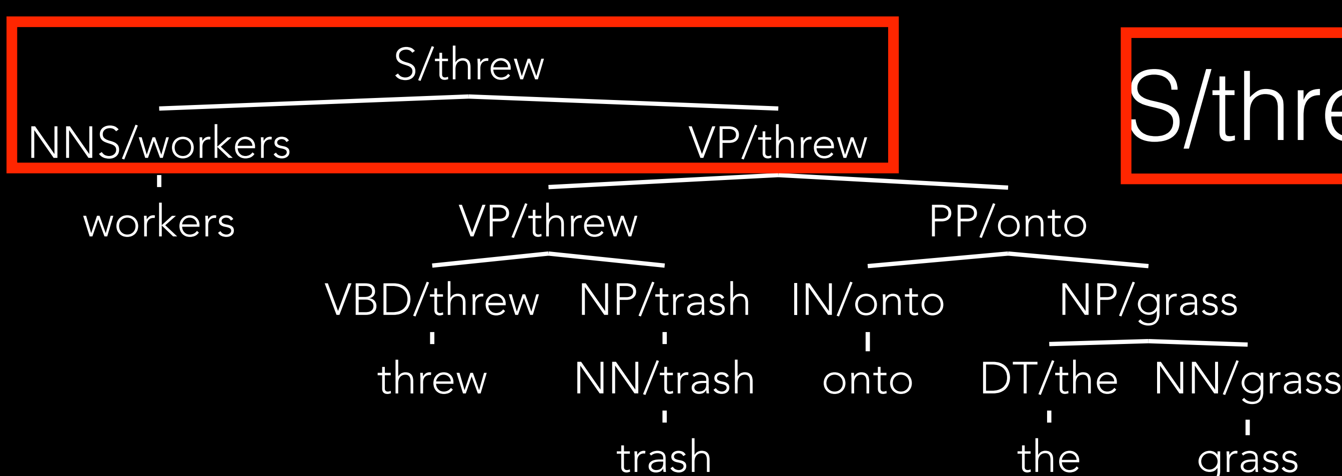
$20/21 \times 0/20$ $\quad$ $1/21 \times 20/60$

$=.016$ $\quad$ VP/threw -> VBD/threw NP/trash $=.5*.016 = .008$

Incorporating Smoothing

Assume we saw each training tree 20 times





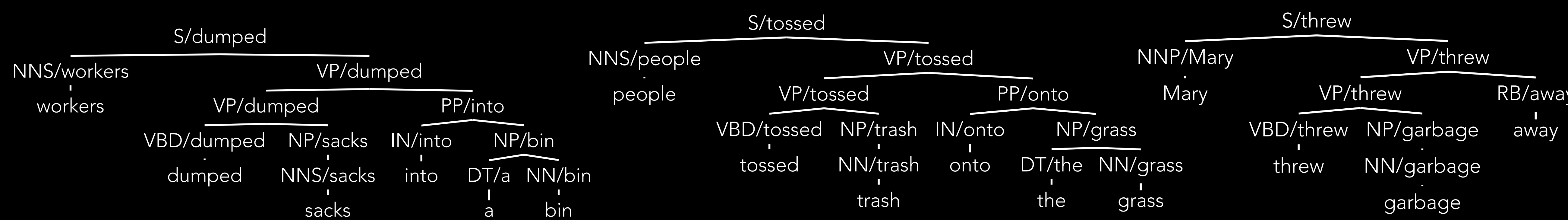
S/threw -> NNS/workers VP/threw

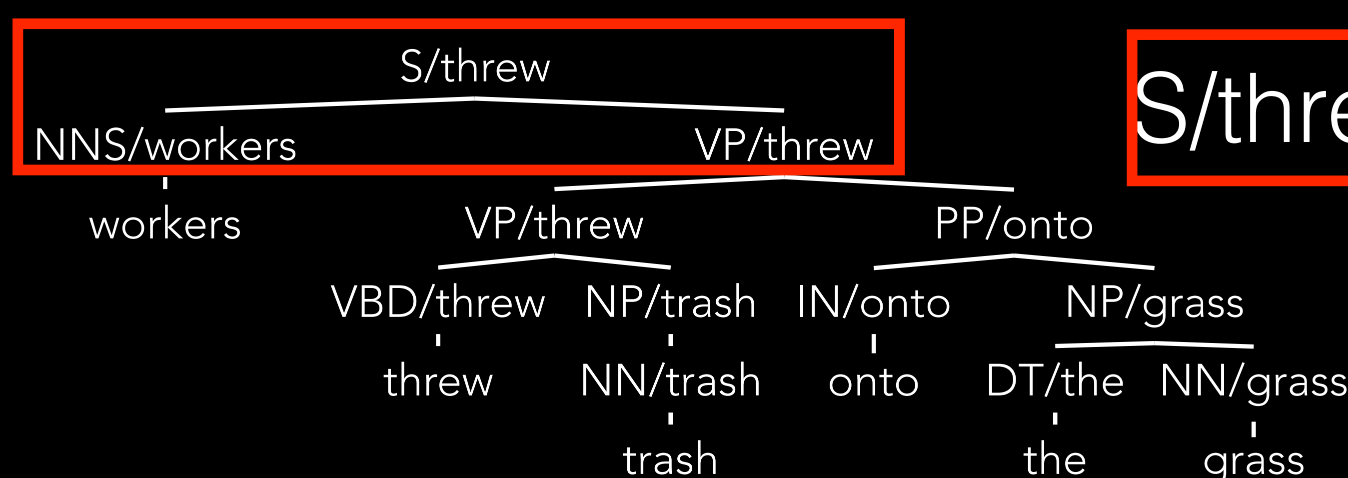
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

0/20 40/60

Incorporating Smoothing

Assume we saw each training tree 20 times





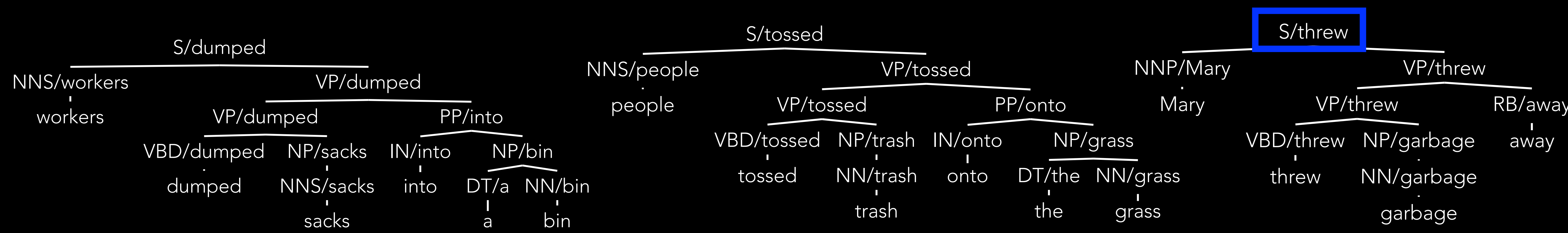
S/threw -> NNS/workers VP/threw

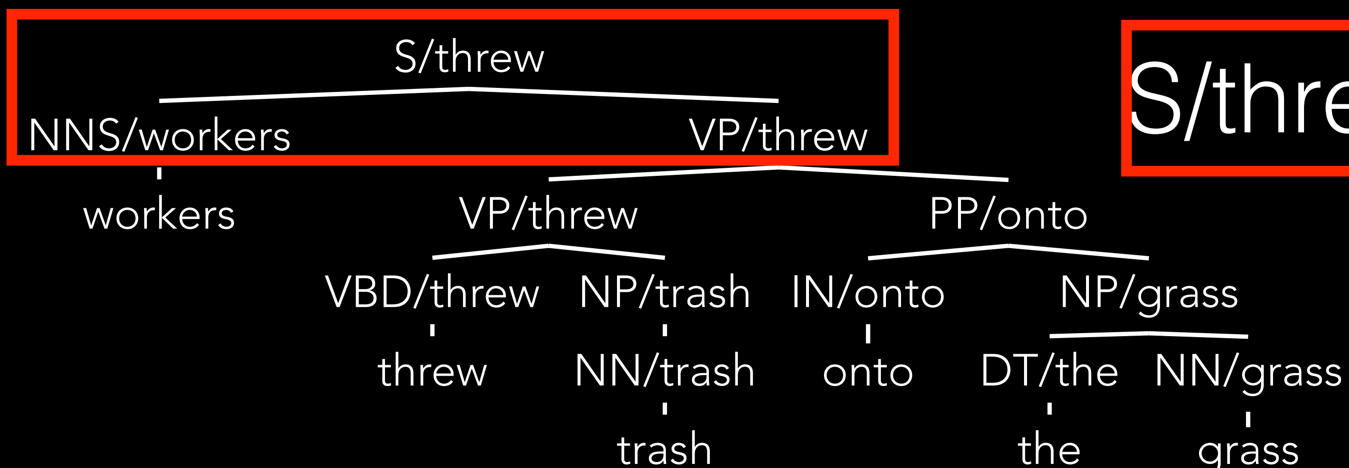
$$P(r \mid \boxed{X, h}) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

0/20 40/60

Incorporating Smoothing

Assume we saw each training tree 20 times





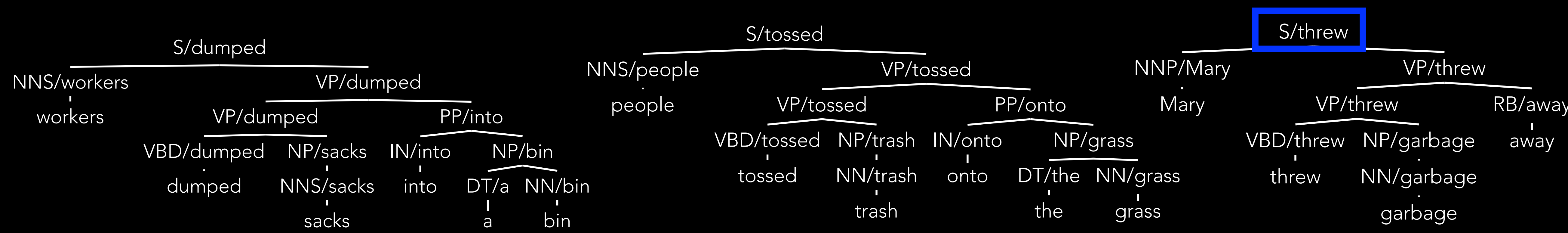
S/threw -> NNS/workers VP/threw

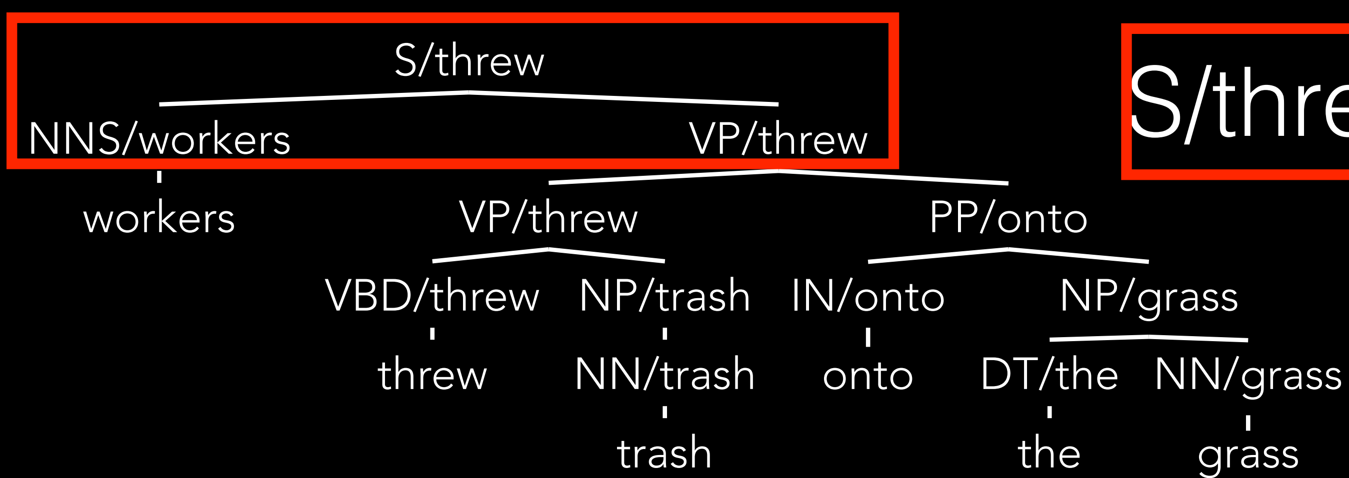
$$P(r \mid \boxed{X, h}) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

0/20 40/60

Incorporating Smoothing

Assume we saw each training tree 20 times





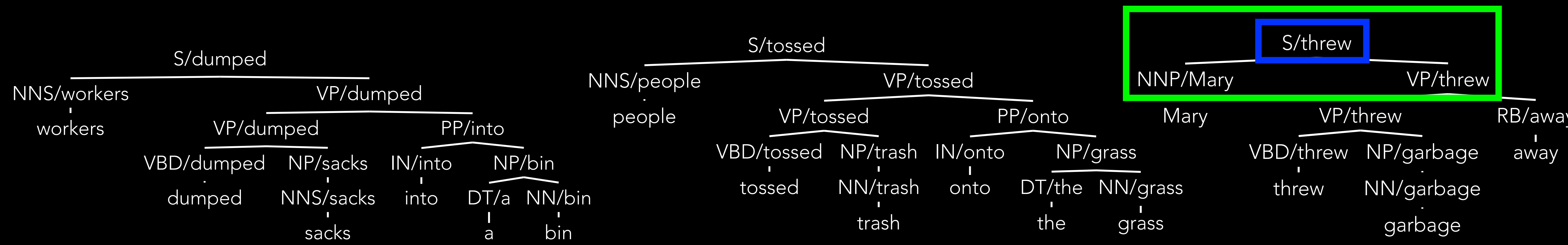
S/threw -> NNS/workers VP/threw

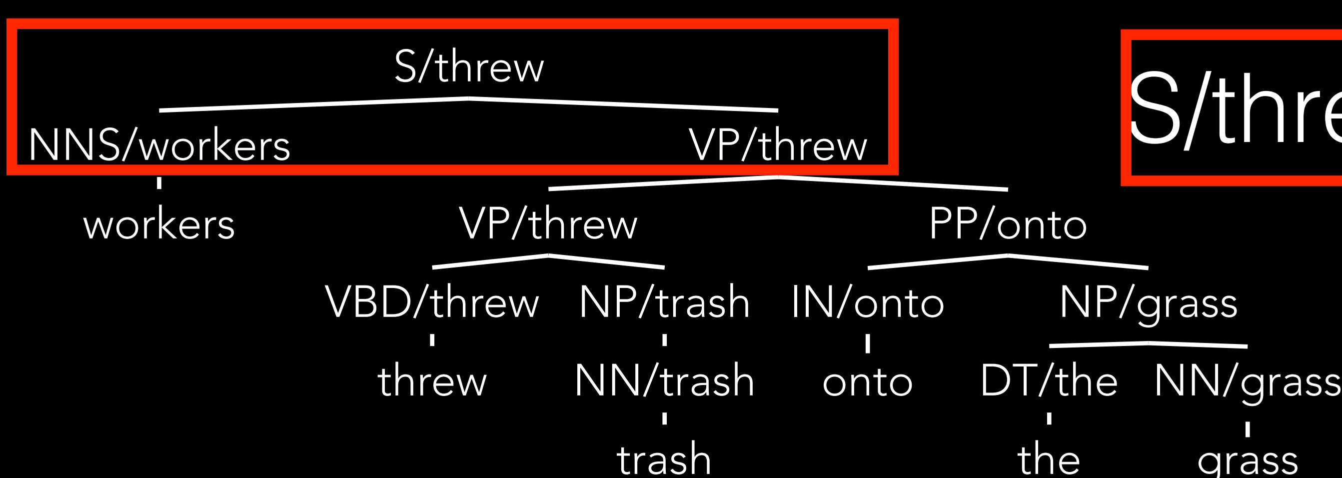
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

0/20 40/60

Incorporating Smoothing

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

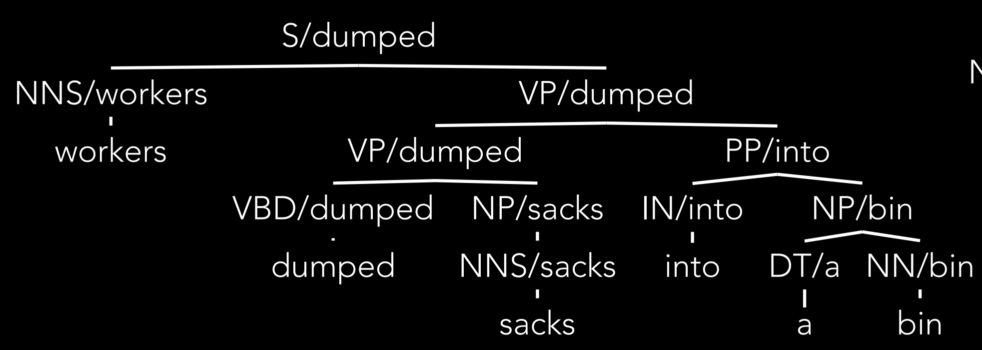
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

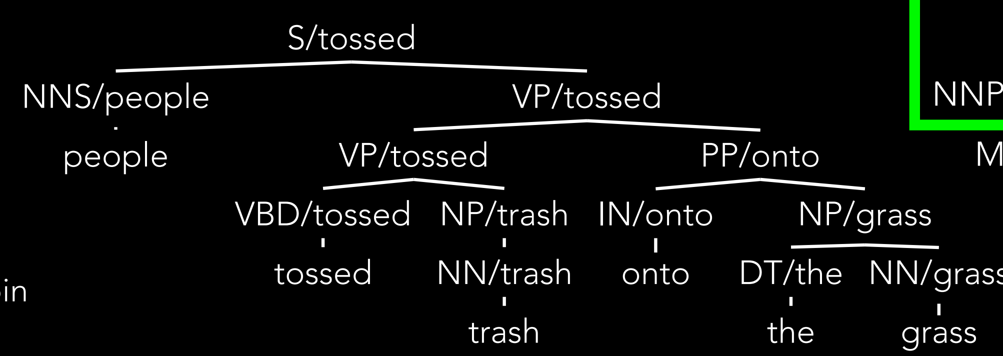
0/20

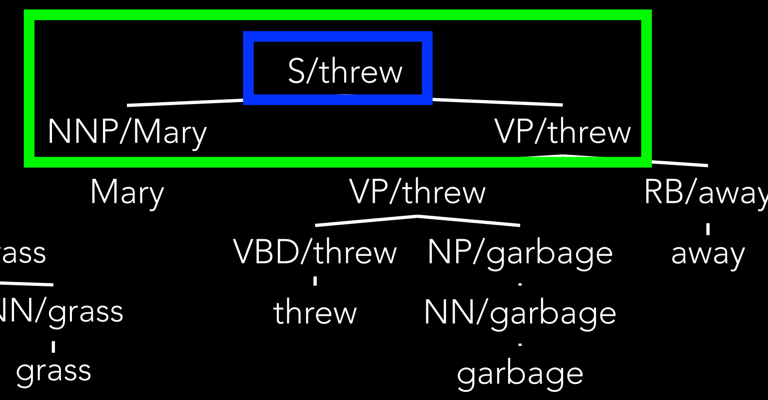
40/60

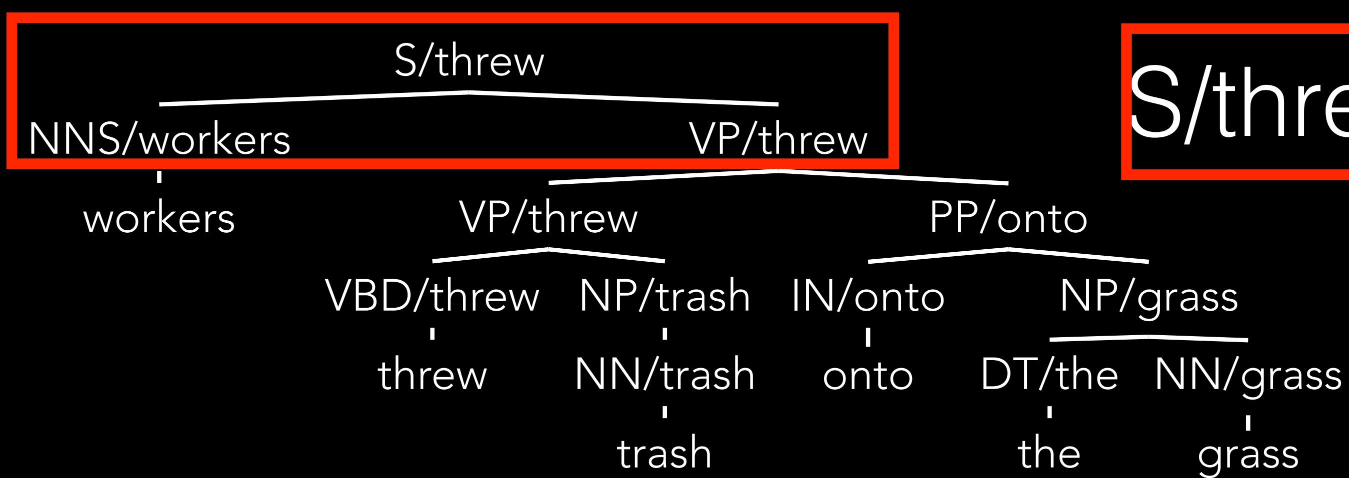
Incorporating Smoothing

Assume we saw each training tree 20 times









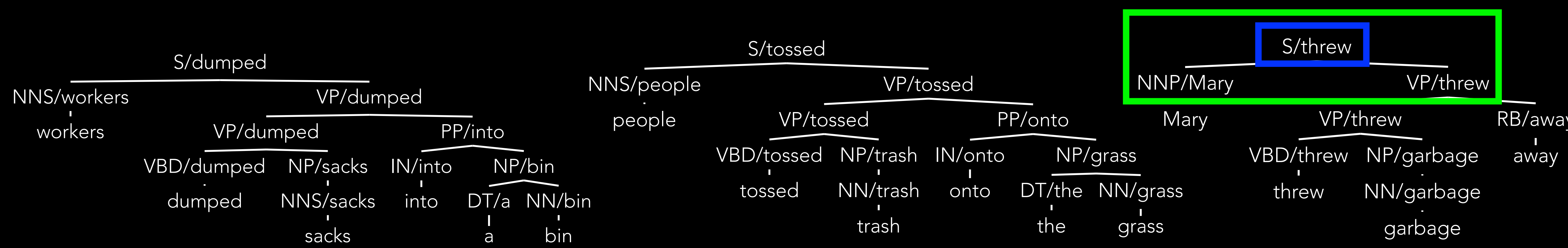
S/threw -> NNS/workers VP/threw

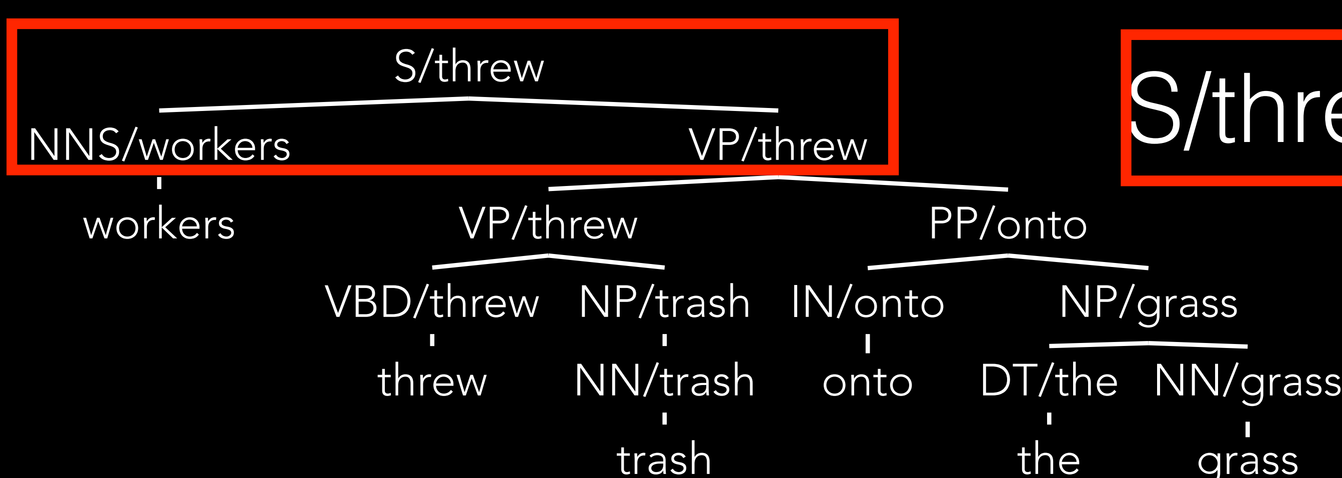
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

20/21 x 0/20 40/60

Incorporating Smoothing

Assume we saw each training tree 20 times





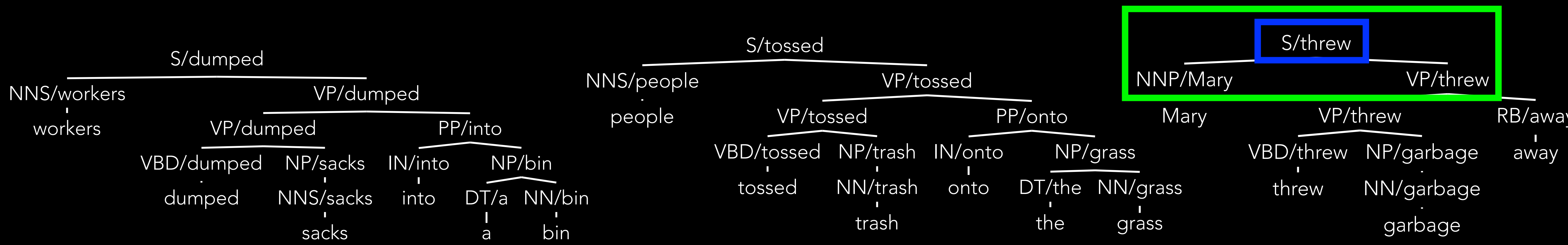
S/threw -> NNS/workers VP/threw

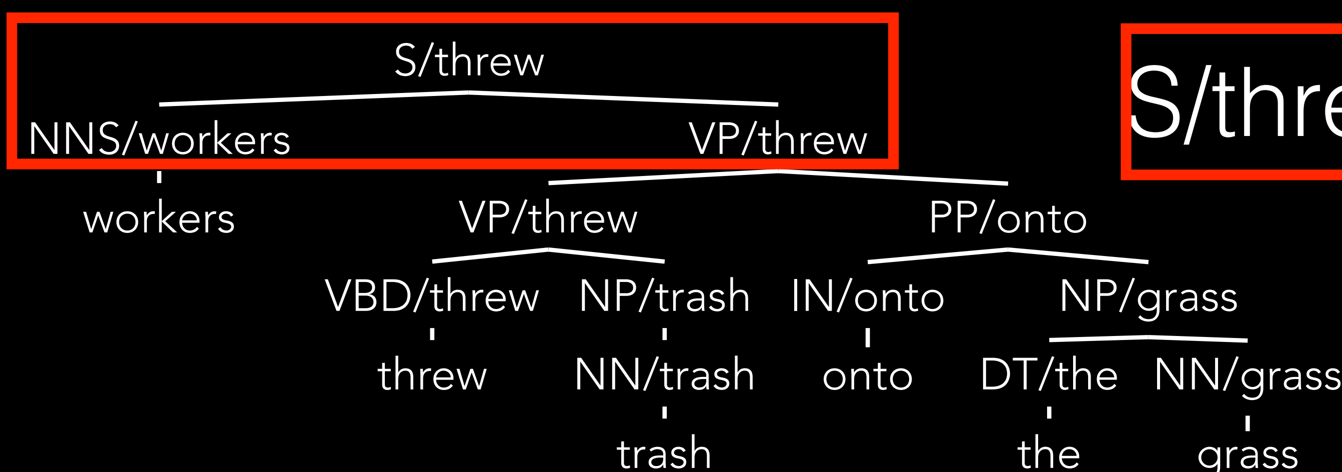
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

20/21 x 0/20   1/21 x 40/60

Incorporating Smoothing

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

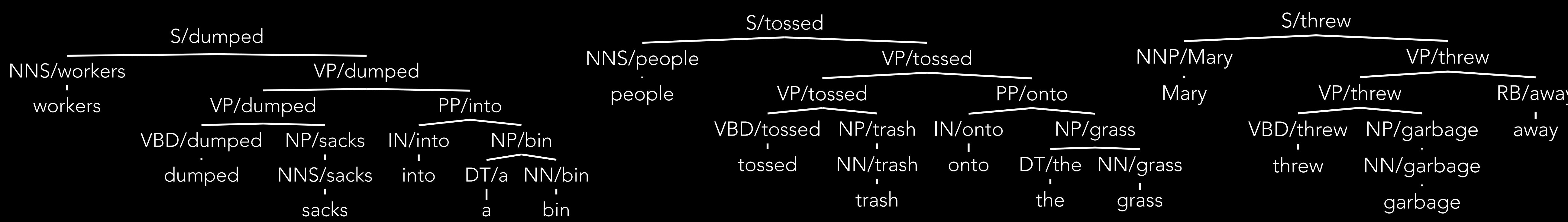
$$P(r \mid X, h) = \lambda_1 P_{ML}(r \mid X, h) + (1-\lambda_1) P_{ML}(r \mid X)$$

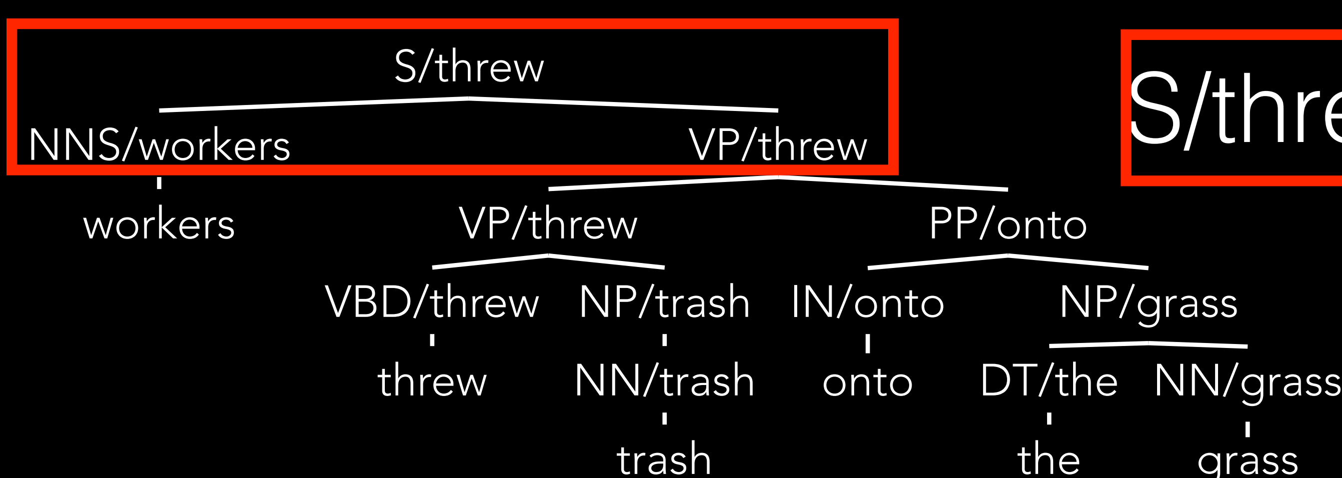
$20/21 \times 0/20 \qquad 1/21 \times 40/60$

$=.032$

Incorporating Smoothing

Assume we saw each training tree 20 times



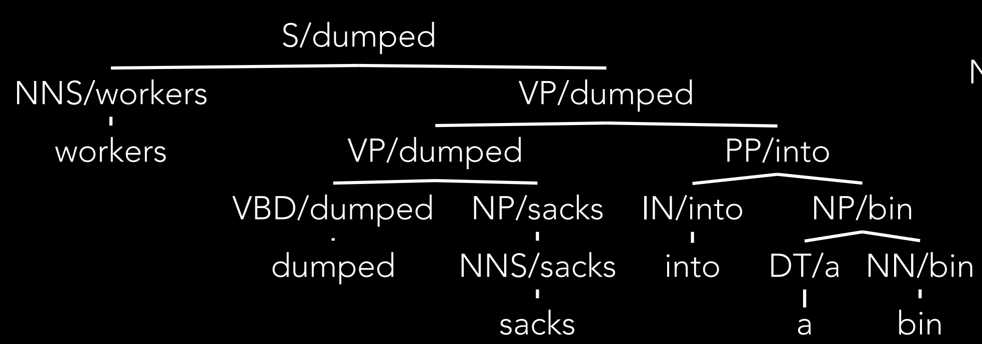


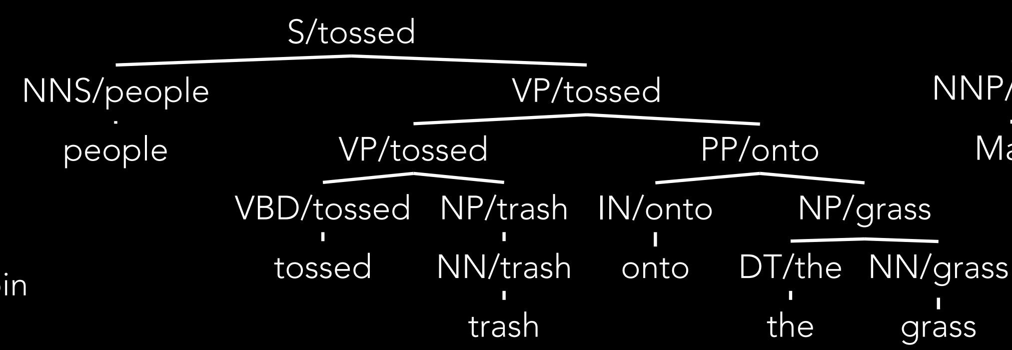
S/threw -> NNS/workers VP/threw

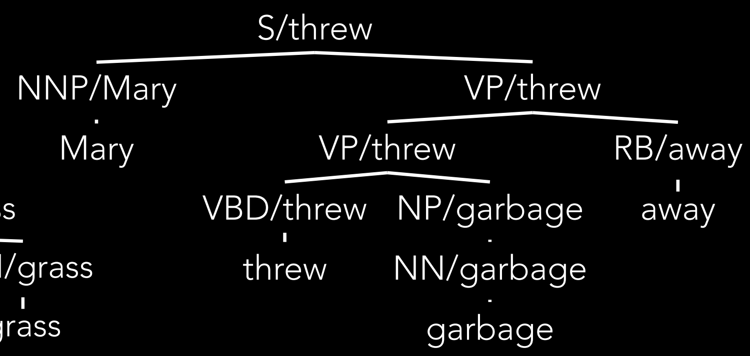
$$P(m \mid r, X, h) = \underbrace{\lambda_2 P_{ML}(m \mid r, X, h)}_{0/0} + \underbrace{(1-\lambda_2) P_{ML}(m \mid r)}_{20/40}$$

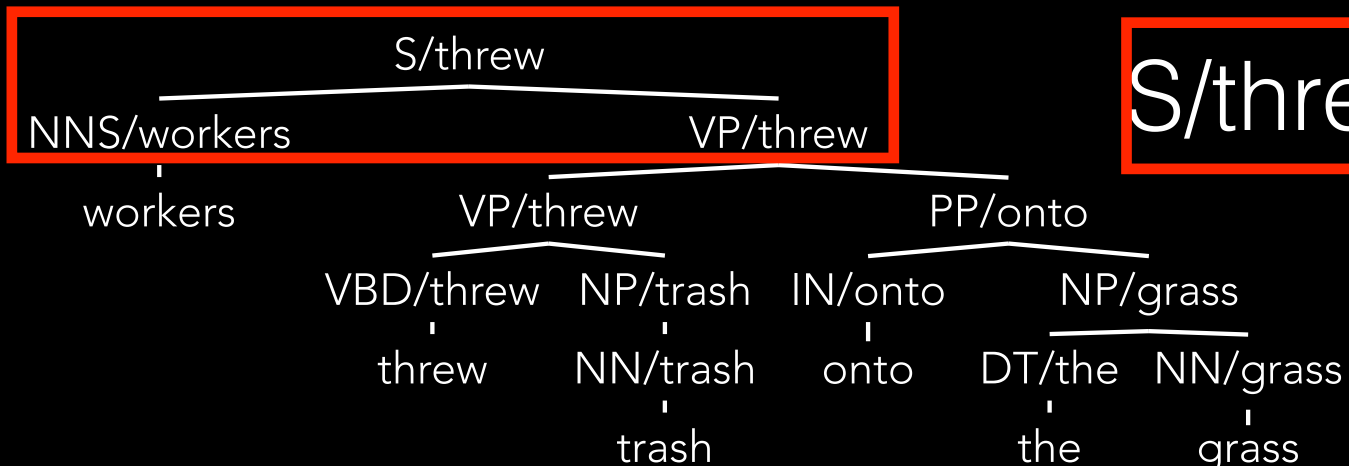
Incorporating Smoothing

Assume we saw each training tree 20 times







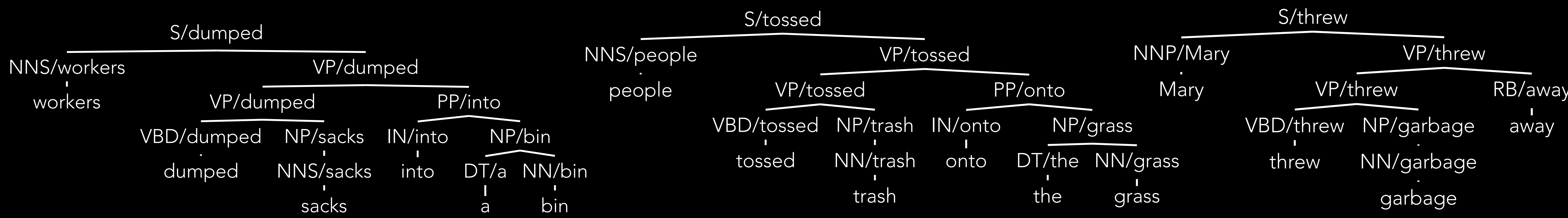


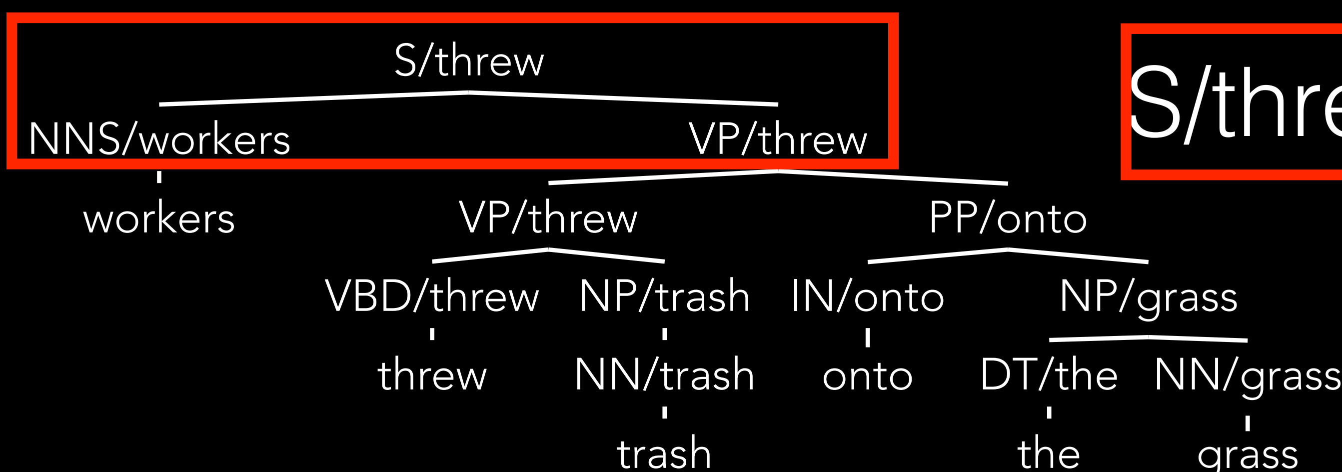
S/threw -> NNS/workers VP/threw

$$P(m \mid r, X, h) = \underbrace{\lambda_2}_{0} \times \underbrace{P_{ML}(m \mid r, X, h)}_{0/0} + (1-\lambda_2)\underbrace{P_{ML}(m \mid r)}_{20/40}$$

Incorporating Smoothing

Assume we saw each training tree 20 times





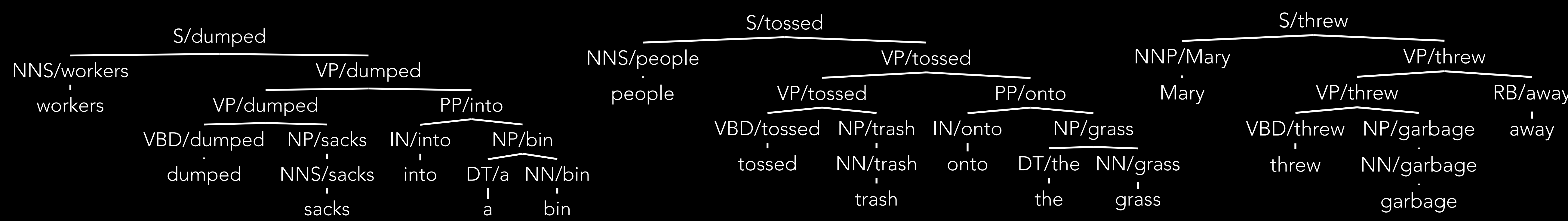
$\boxed{\text{S/threw -> NNS/workers VP/threw}}$

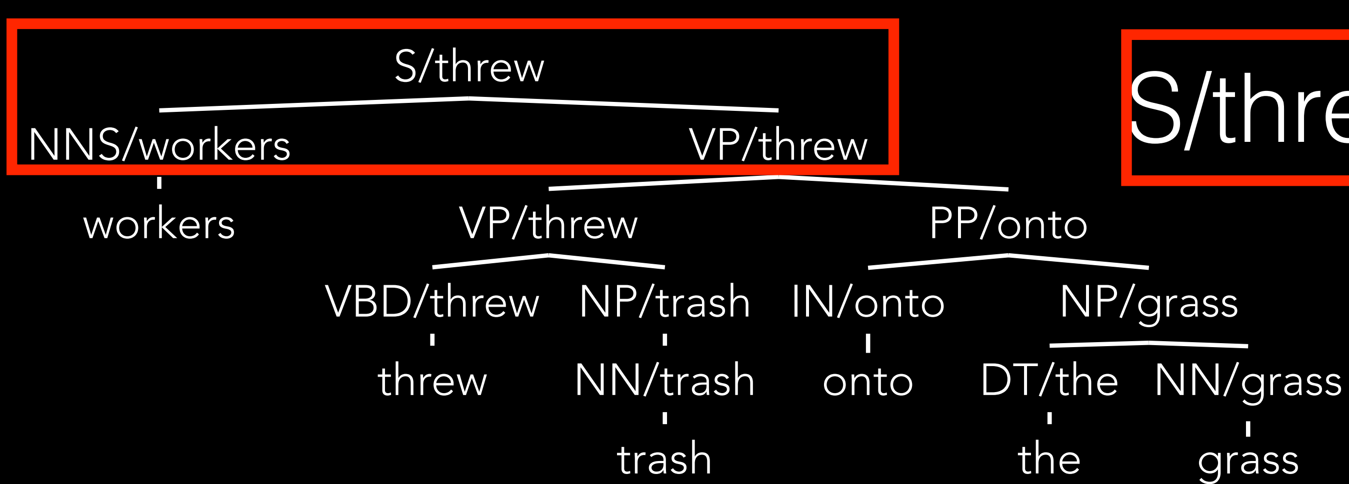
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

0 x 0/0   1 x 20/40

Incorporating Smoothing

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

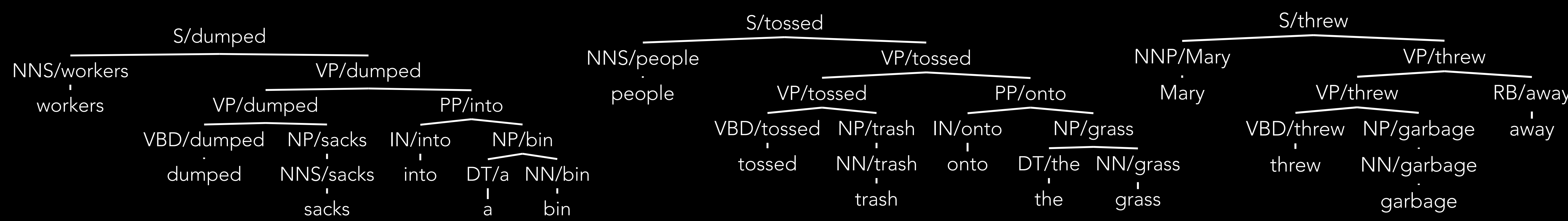
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

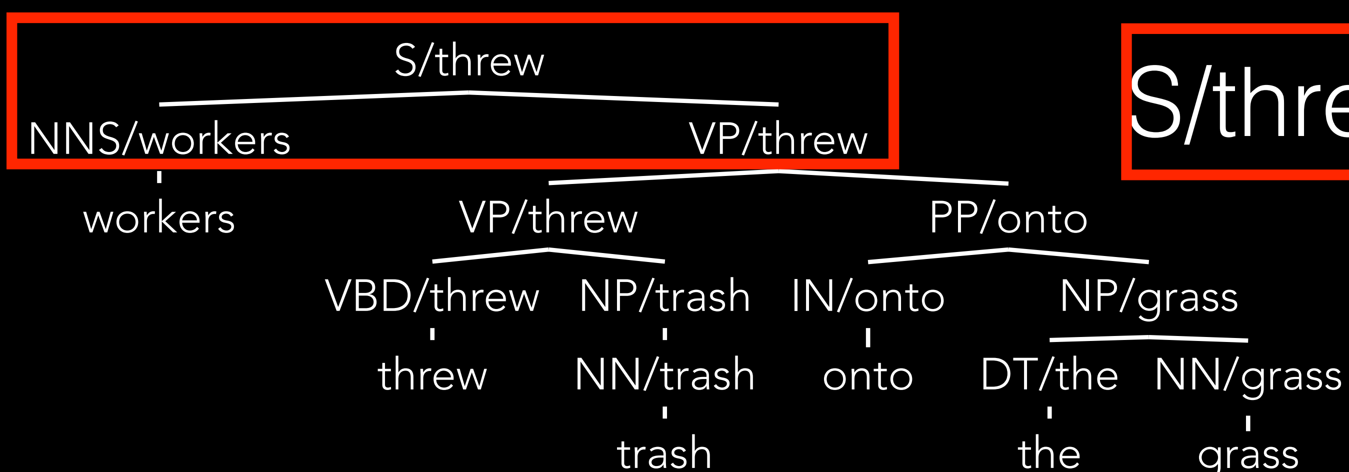
$$0 \times 0/0 \qquad\qquad 1 \times 20/40$$

$=0.5$

Incorporating Smoothing

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

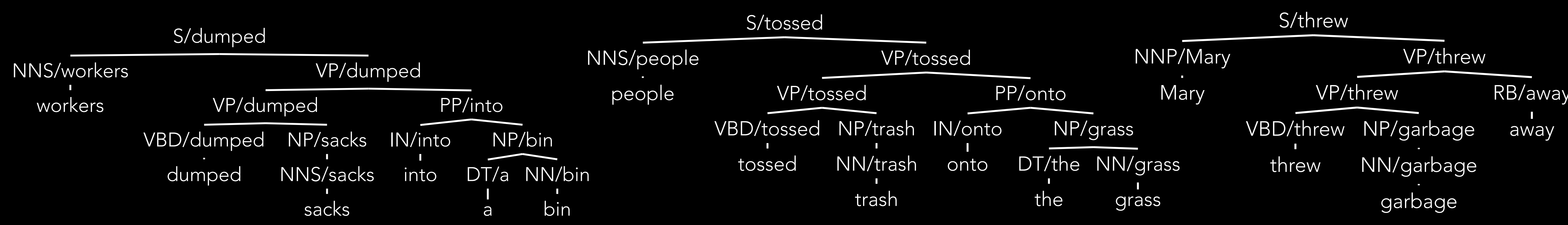
$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

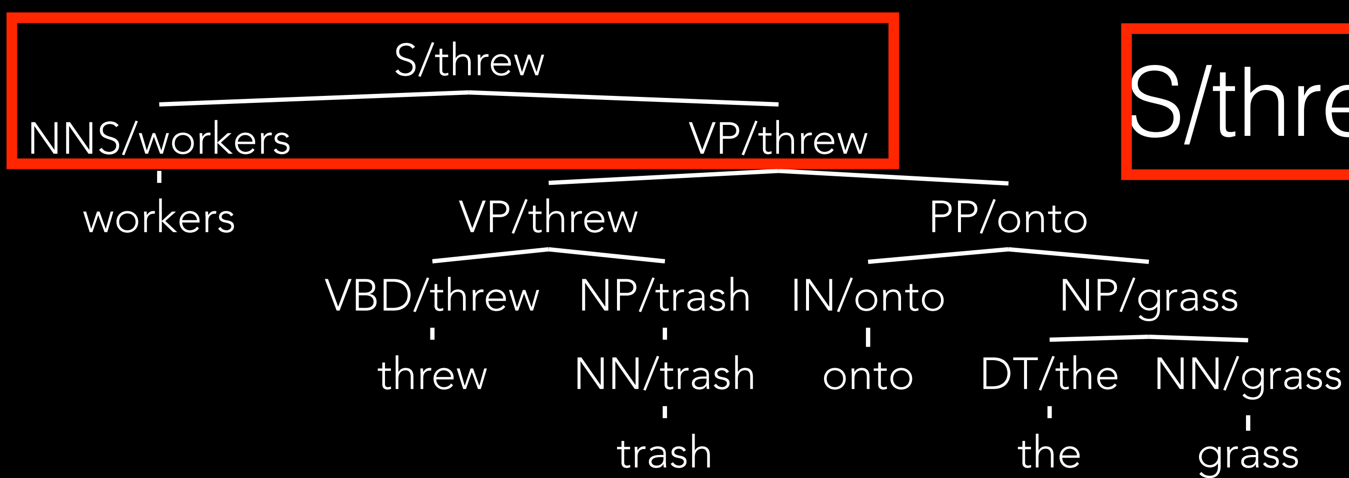
$0 \times 0/0$ $\quad$ $1 \times 20/40$

$=0.5$ S/threw -> NNS/workers VP/threw

Incorporating Smoothing

Assume we saw each training tree 20 times





S/threw -> NNS/workers VP/threw

$$P(m \mid r, X, h) = \lambda_2 P_{ML}(m \mid r, X, h) + (1-\lambda_2) P_{ML}(m \mid r)$$

$$0 \times 0/0 \qquad\qquad 1 \times 20/40$$

$=0.5$

S/threw -> NNS/workers VP/threw $= .032 * .5 = .016$

Other Ways To Smooth

Other Ways To Smooth

- Replace infrequently seen words with <unk> in training; map unseen words to <unk> when parsing

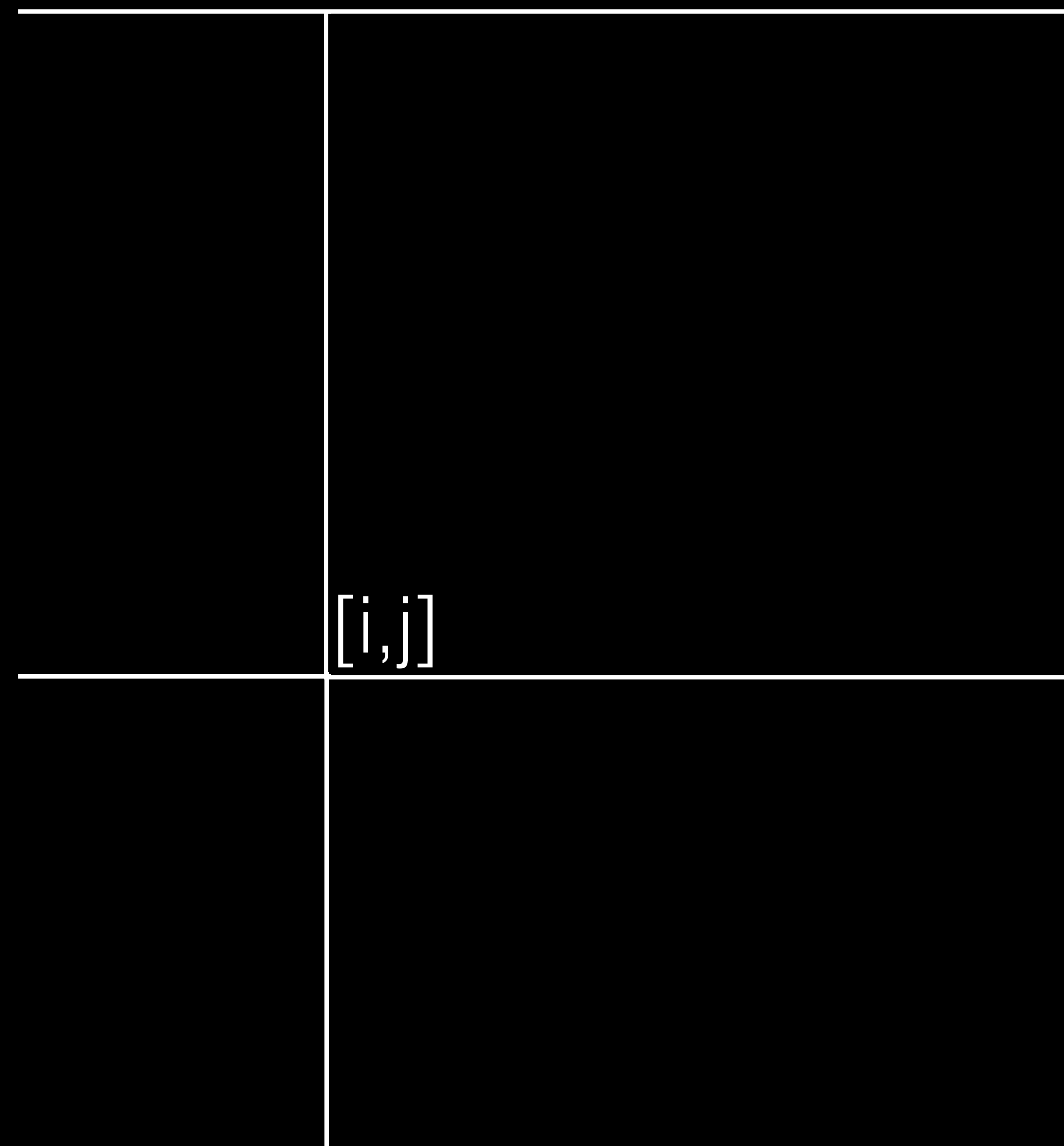
Other Ways To Smooth

- Replace infrequently seen words with <unk> in training; map unseen words to <unk> when parsing
- For interesting suffixes, stem infrequent words instead of replacing them; “working” becomes <unk>-ing

Other Ways To Smooth

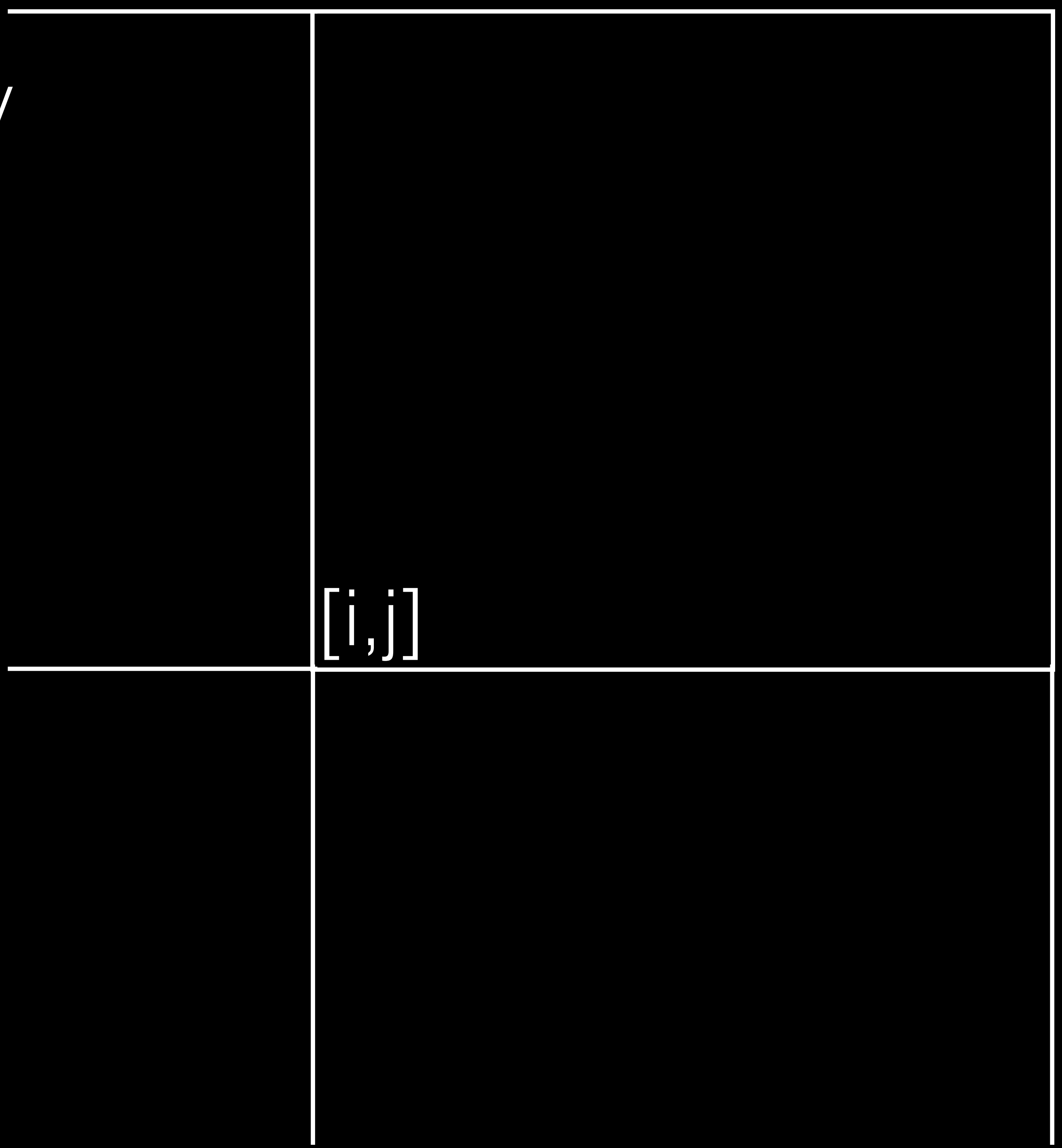
- Replace infrequently seen words with <unk> in training; map unseen words to <unk> when parsing
- For interesting suffixes, stem infrequent words instead of replacing them; “working” becomes <unk>-ing
- POS-tag ‘lexicalization’ instead of/in addition to true lexicalization

Beaming



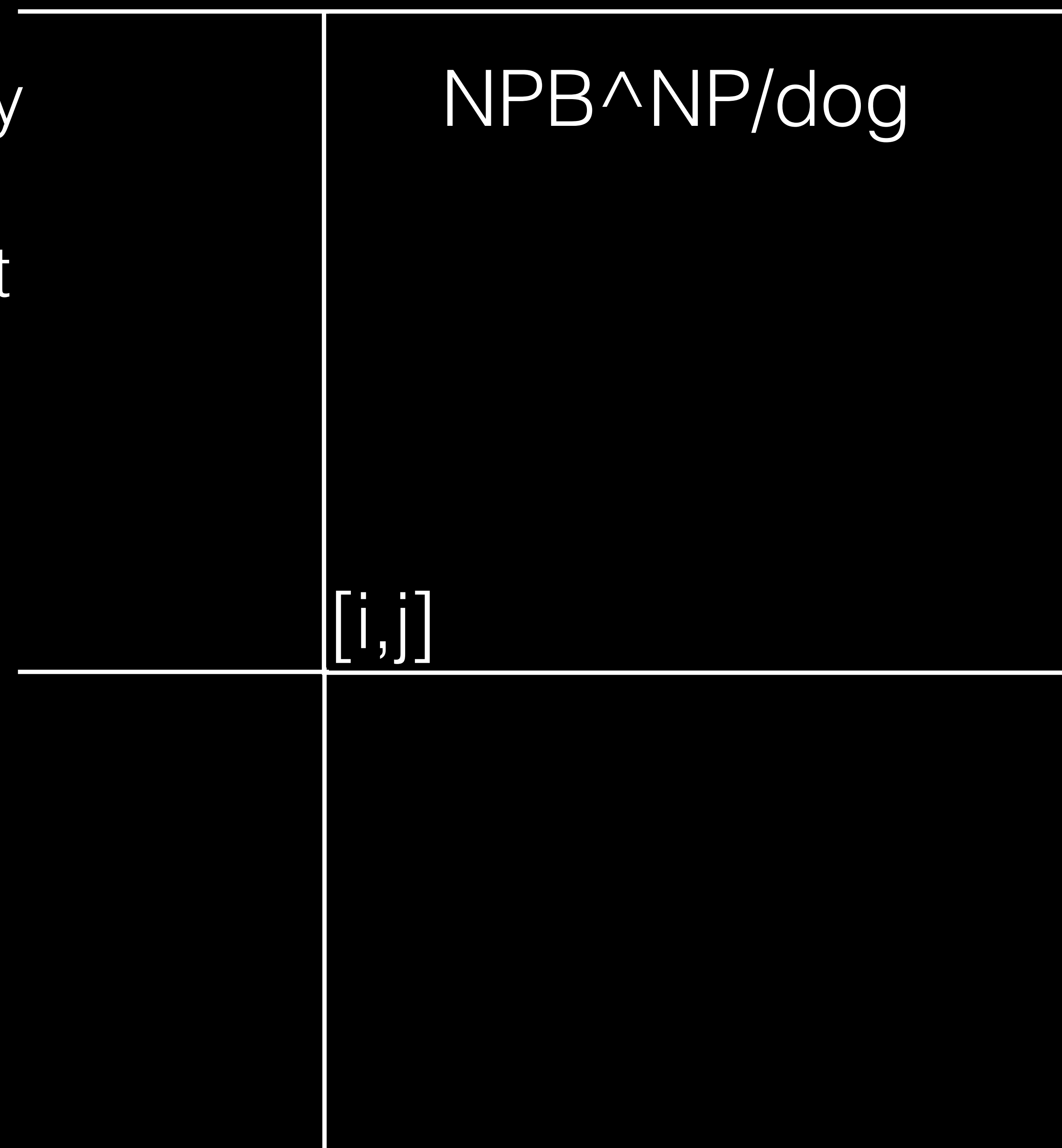
Beaming

- Every state that gets created can be potentially combined to form new states higher up the chart



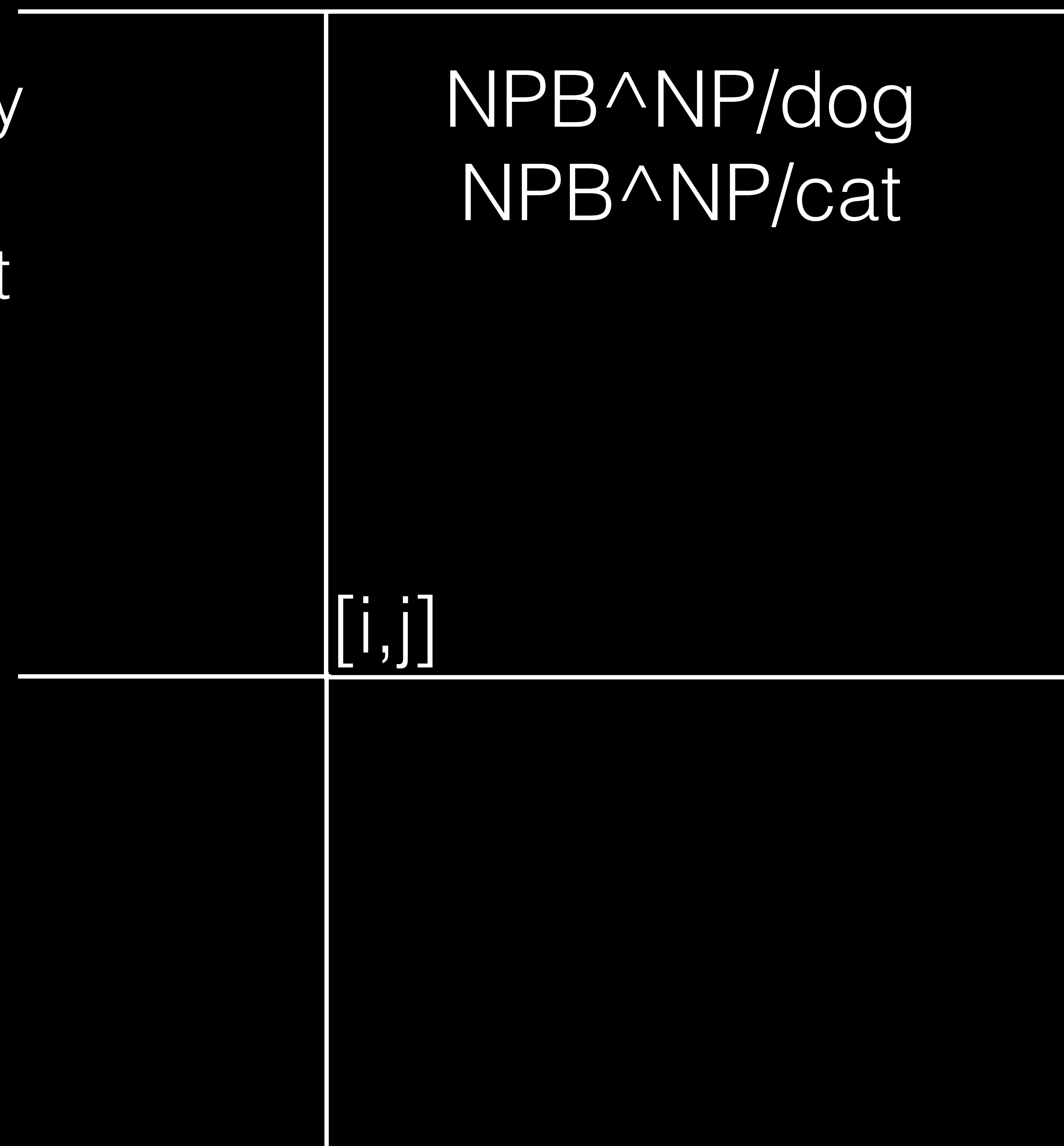
Beaming

- Every state that gets created can be potentially combined to form new states higher up the chart



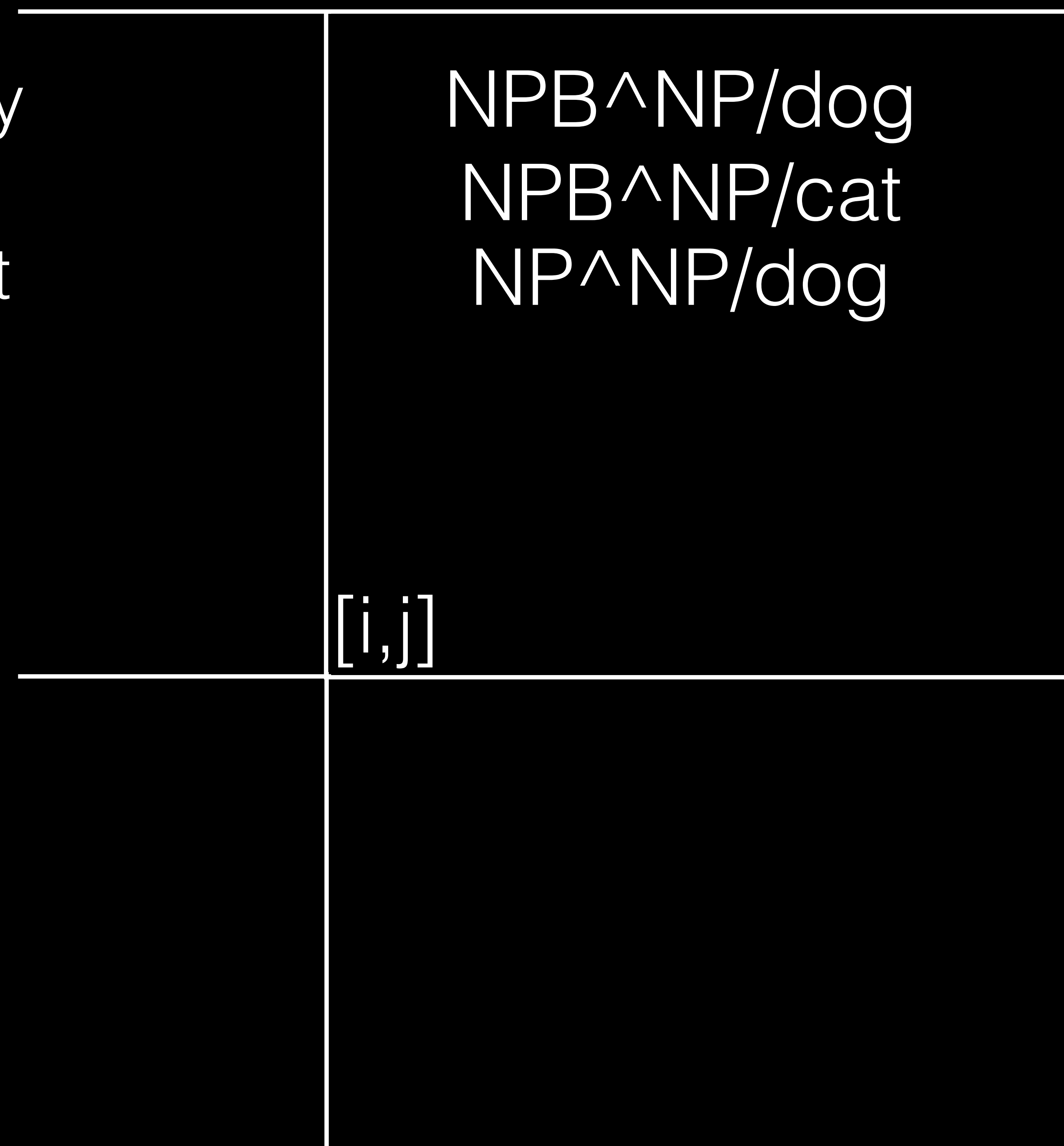
Beaming

- Every state that gets created can be potentially combined to form new states higher up the chart



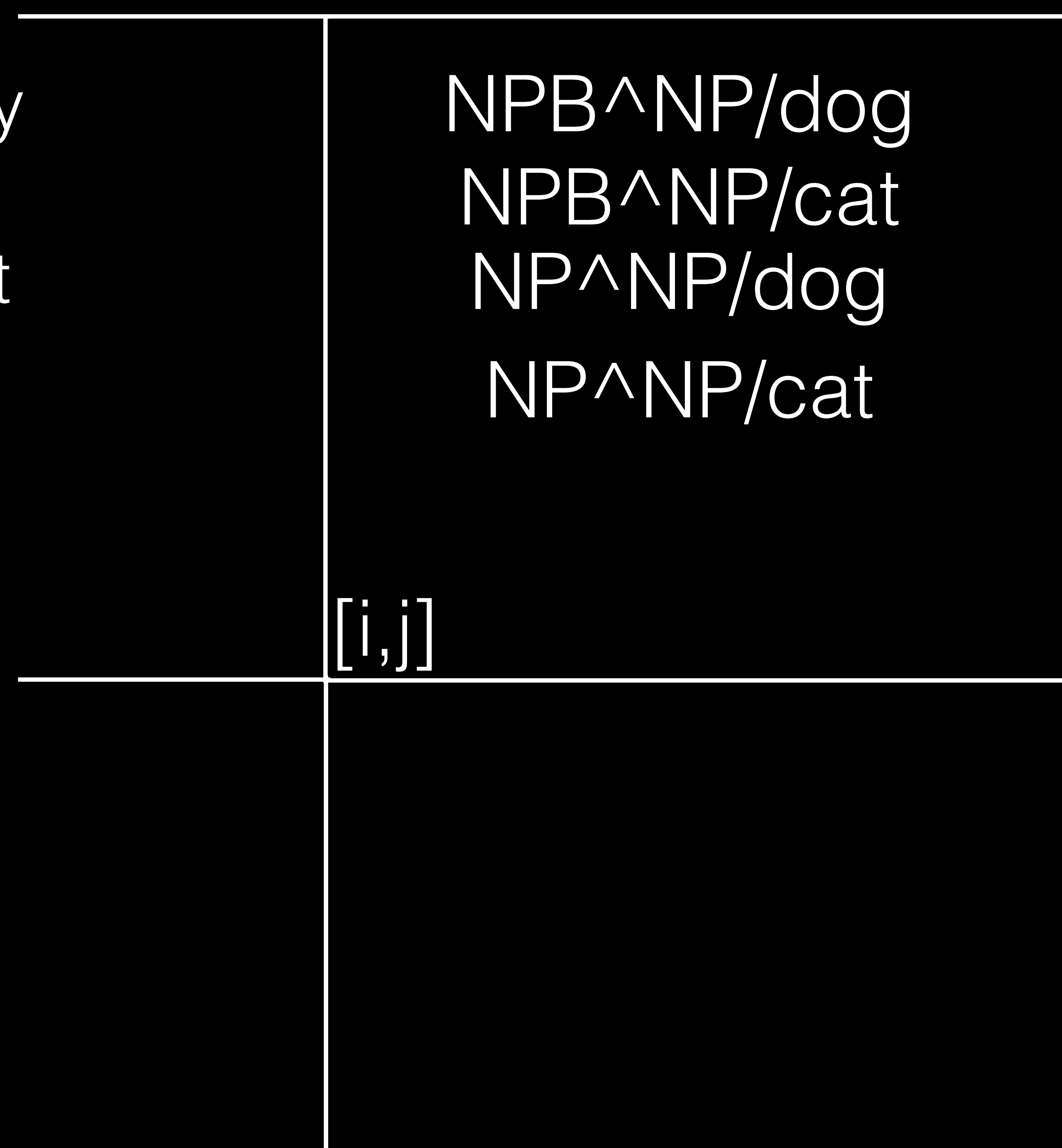
Beaming

- Every state that gets created can be potentially combined to form new states higher up the chart



Beaming

- Every state that gets created can be potentially combined to form new states higher up the chart



Beaming

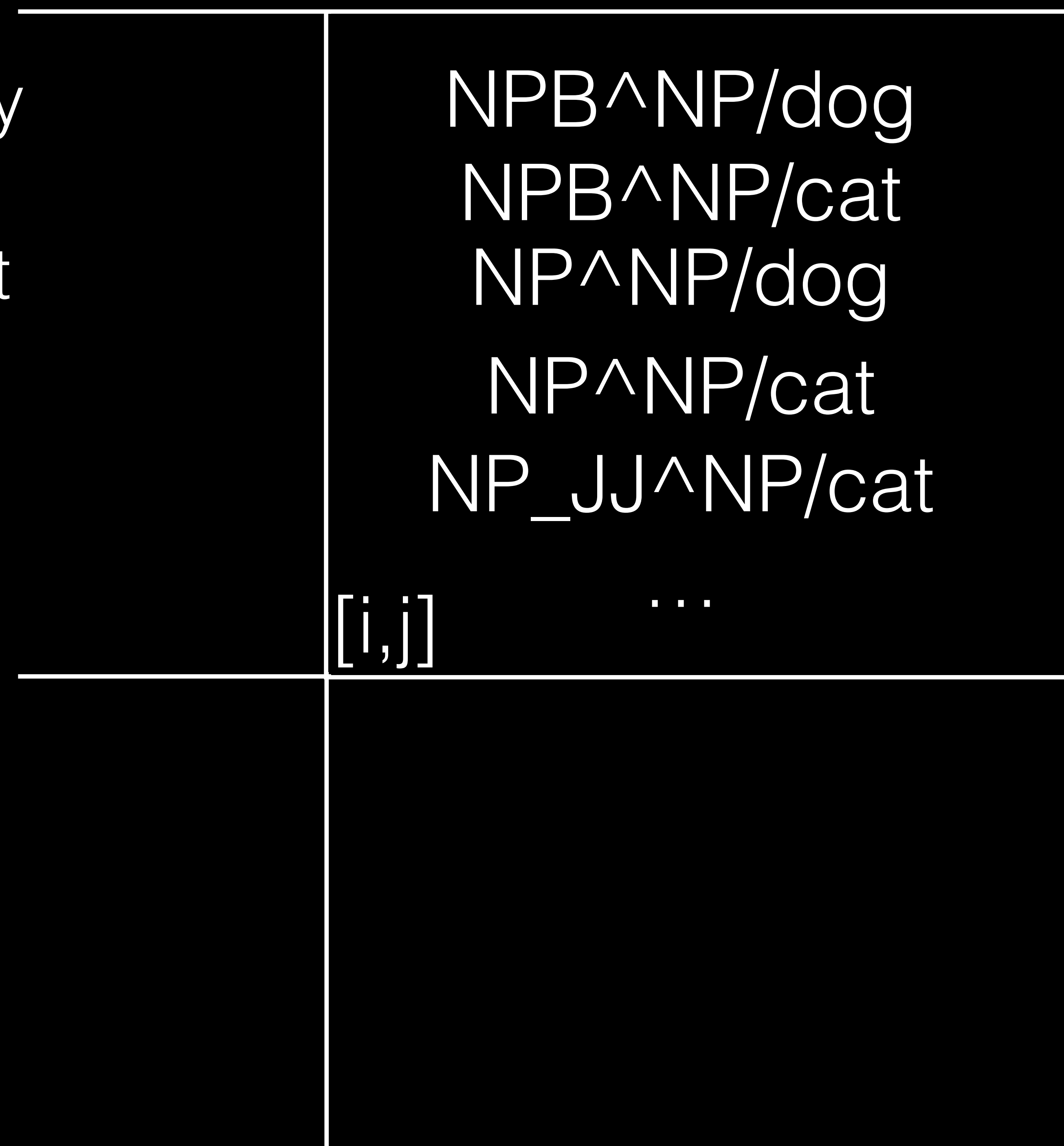
- Every state that gets created can be potentially combined to form new states higher up the chart

NPB^NP/dog
NPB^NP/cat
NP^NP/dog
NP^NP/cat
NP_JJ^NP/cat

[i,j]

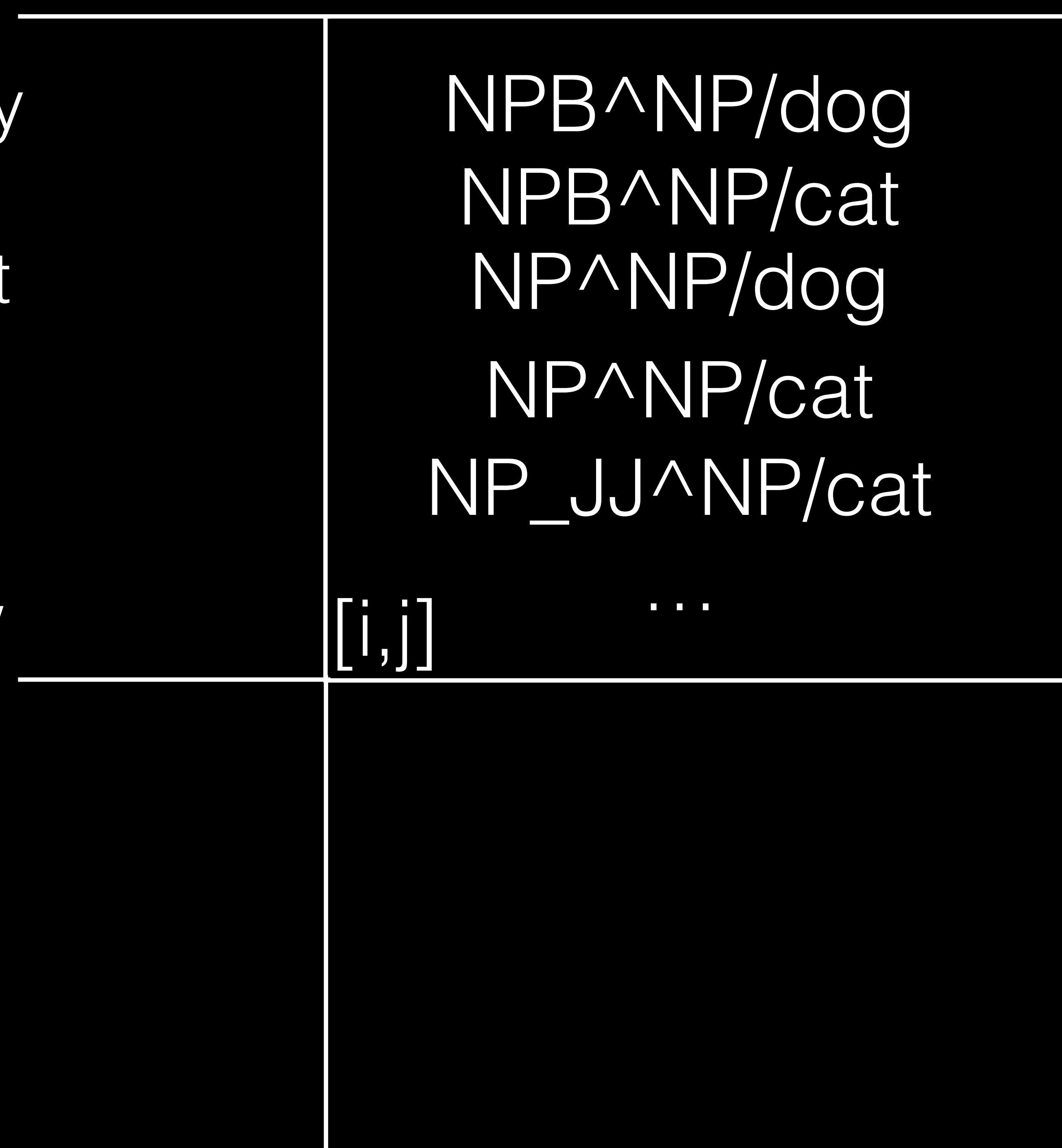
Beaming

- Every state that gets created can be potentially combined to form new states higher up the chart



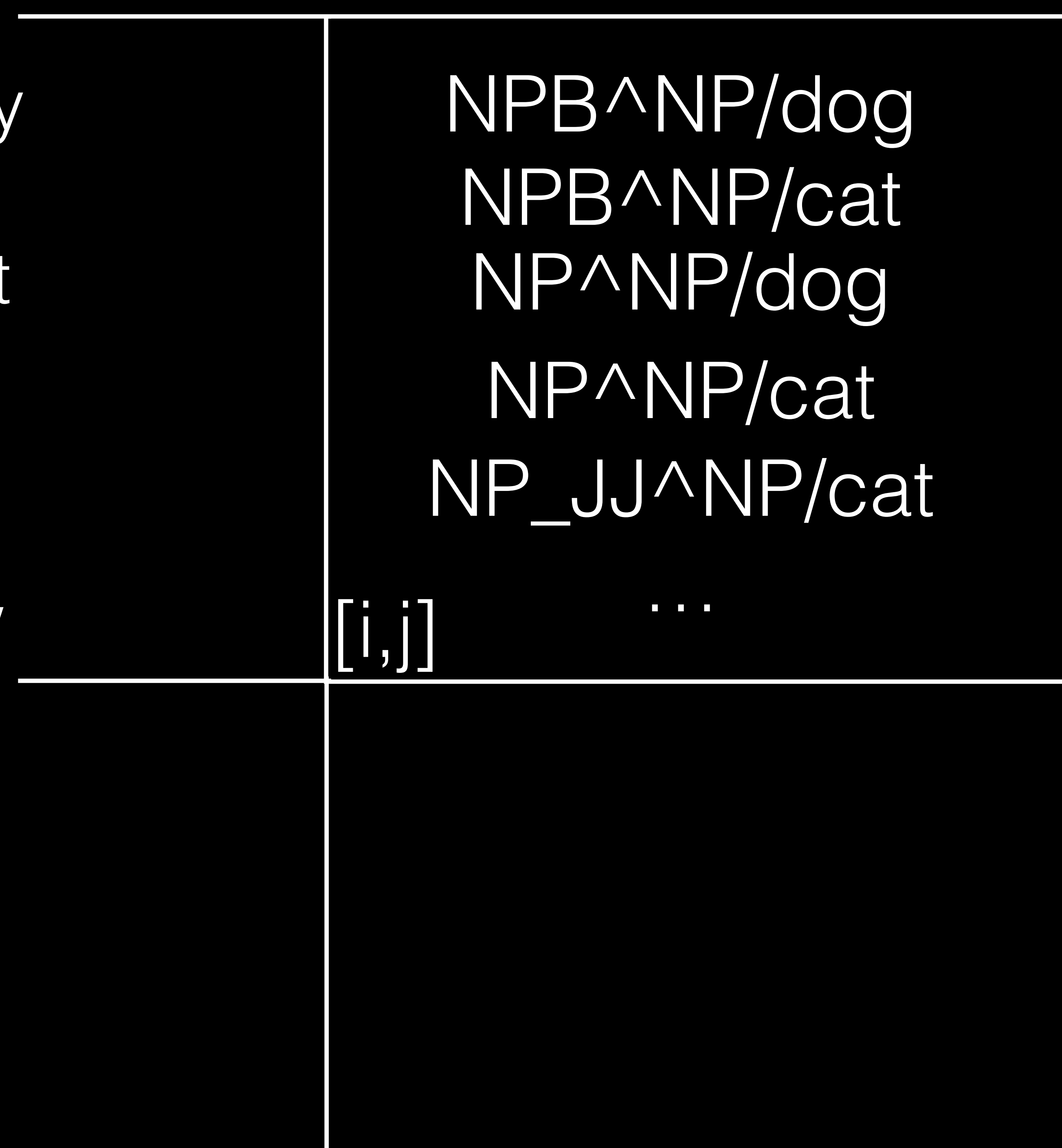
Beaming

- Every state that gets created can be potentially combined to form new states higher up the chart
- As grammars get more complex, number of states per span can grow



Beaming

- Every state that gets created can be potentially combined to form new states higher up the chart
- As grammars get more complex, number of states per span can grow
- Parsing will start to get slow unless something is done



Beaming

Beaming

- Beam search eliminates part of the chart as it is constructed

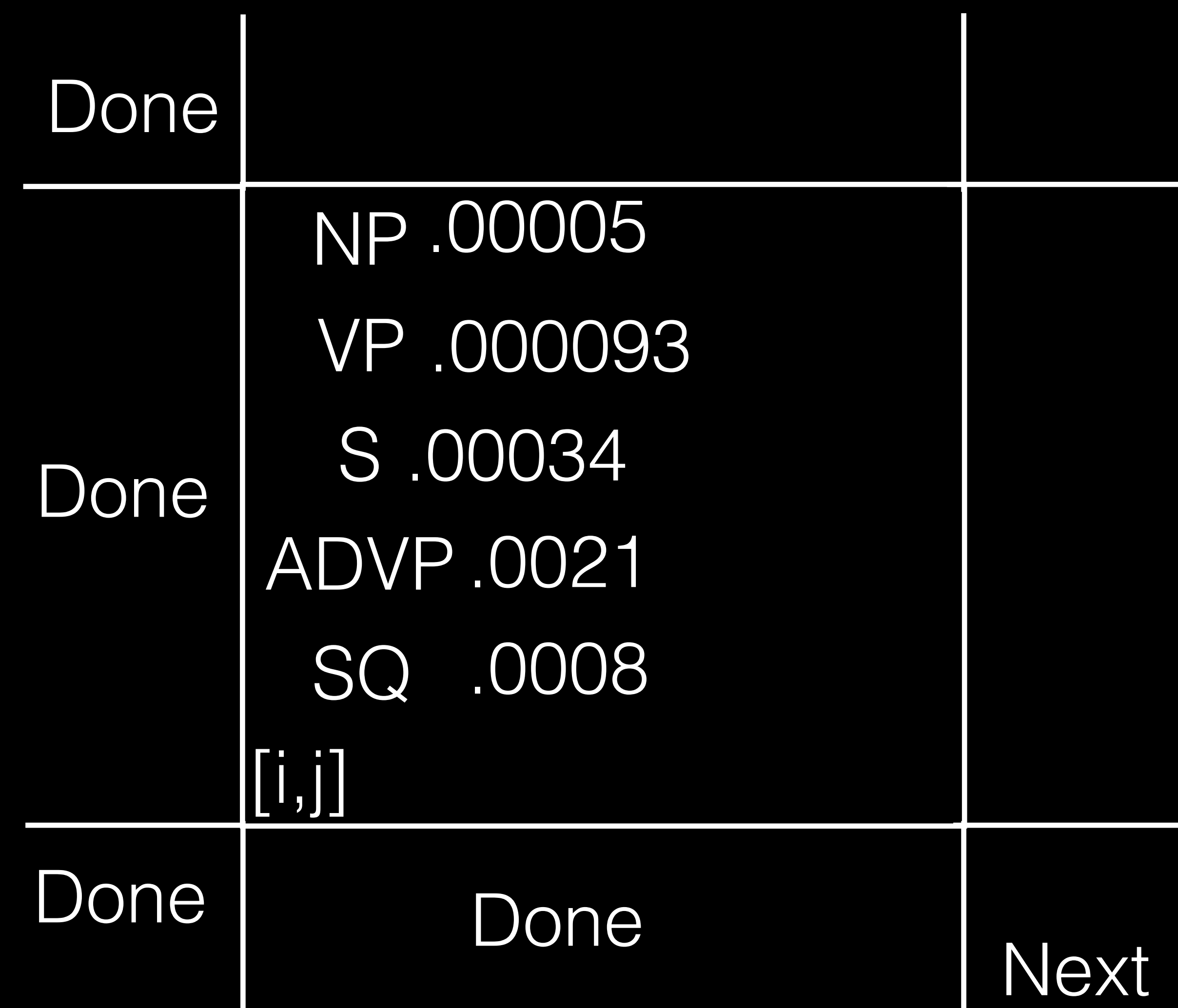
Beaming

- Beam search eliminates part of the chart as it is constructed
- Can cause search errors = best parse found is suboptimal

Beaming

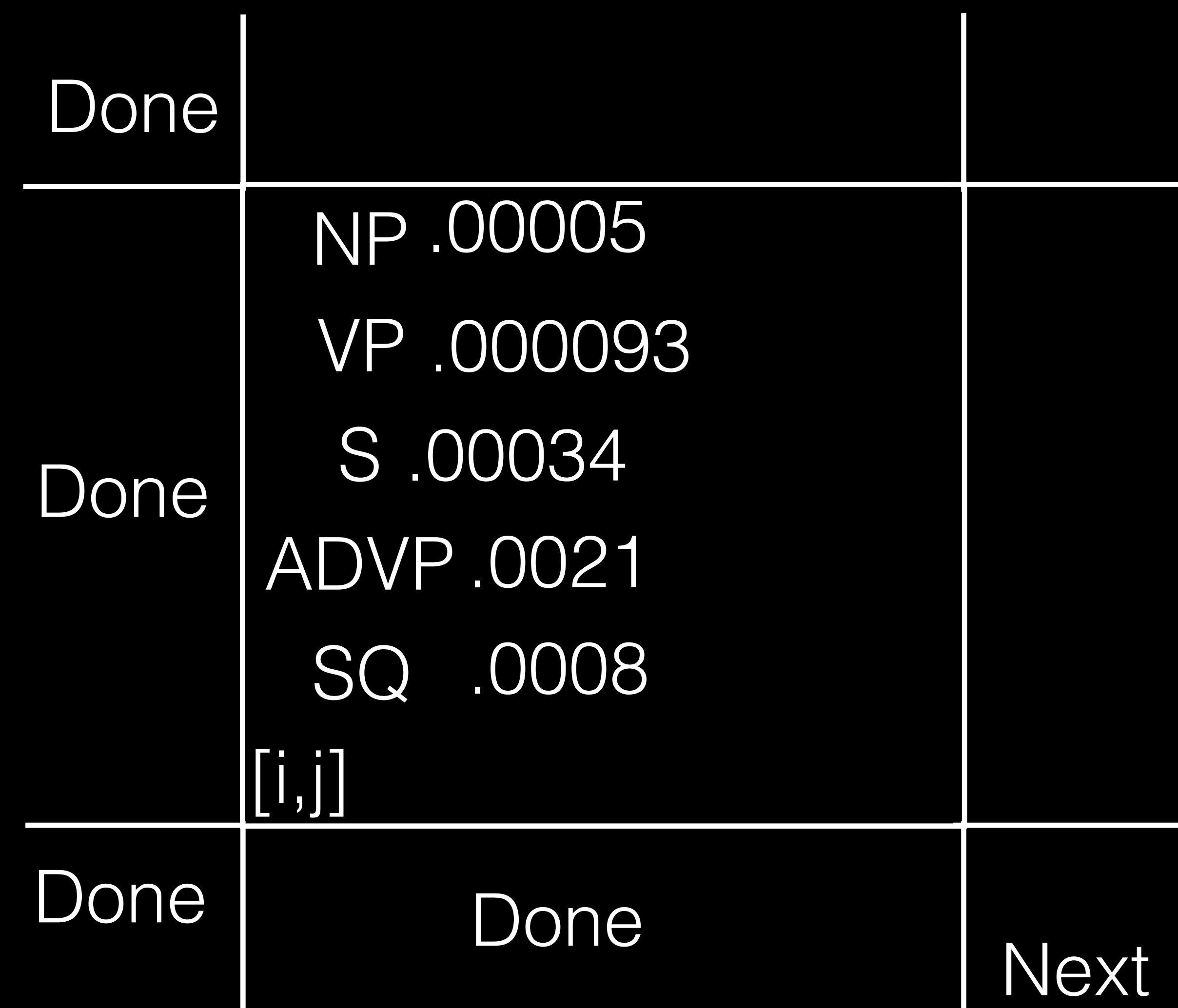
- Beam search eliminates part of the chart as it is constructed
- Can cause search errors = best parse found is suboptimal
- Basic idea: Using heuristic score of partial parse, eliminate states

Beaming


Done
Done
NP .00005
VP .000093
S .00034
ADVP .0021
SQ .0008
[i,j]
Done
Done
Next

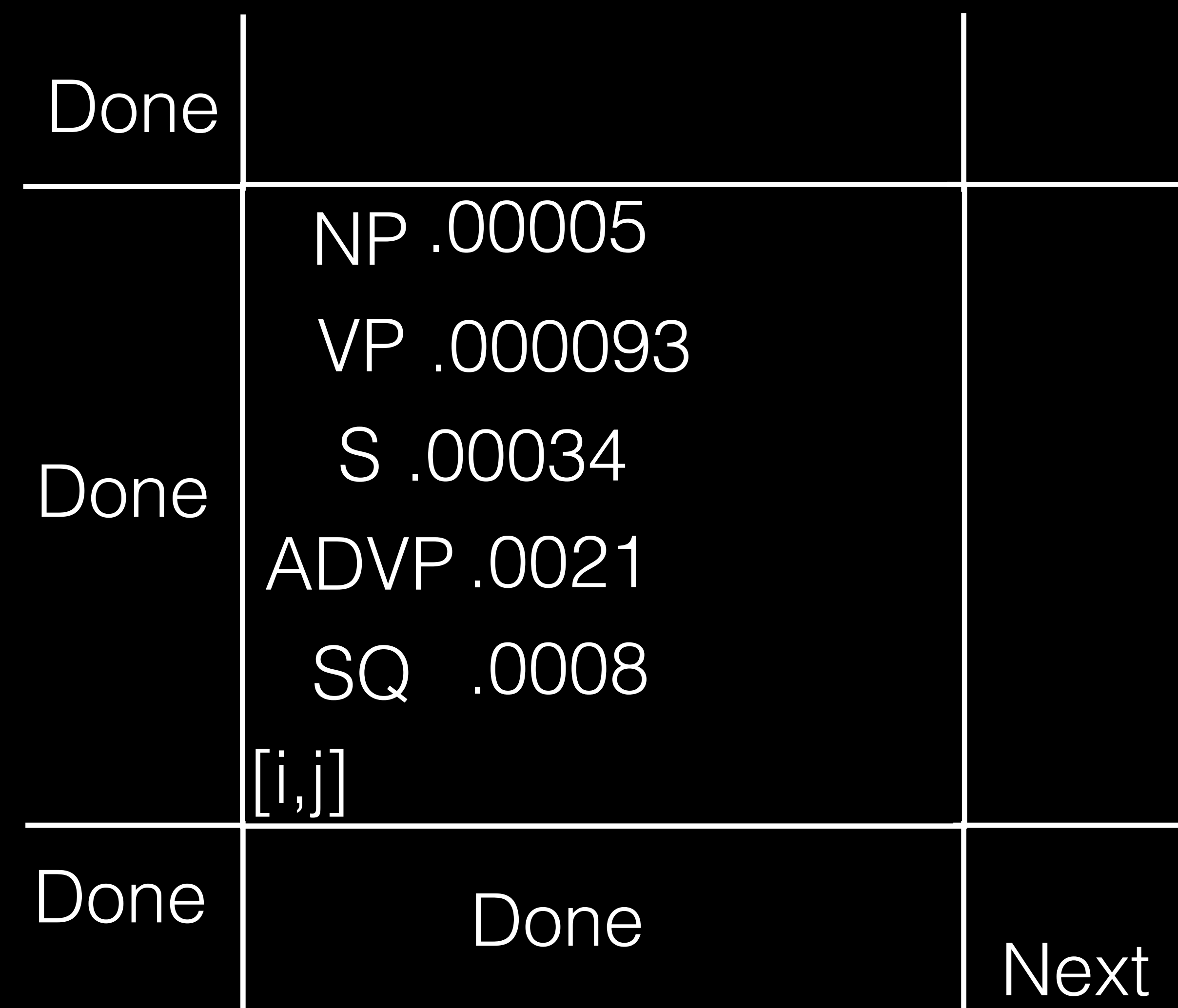
Beaming

- Before moving to [i+1, j+1], throw out some states



Beaming

- Before moving to [i+1, j+1], throw out some states
- Scores are *inside costs* = probability of subtree rooted in X



Beaming

- Before moving to [i+1, j+1], throw out some states

- Scores are *inside costs* = probability of subtree rooted in X

- We want the state most likely to be used in a *successful* parse

Done		
Done	NP .00005 VP .000093 S .00034 ADVP .0021 SQ .0008 [i,j]	
Done	Done	Next

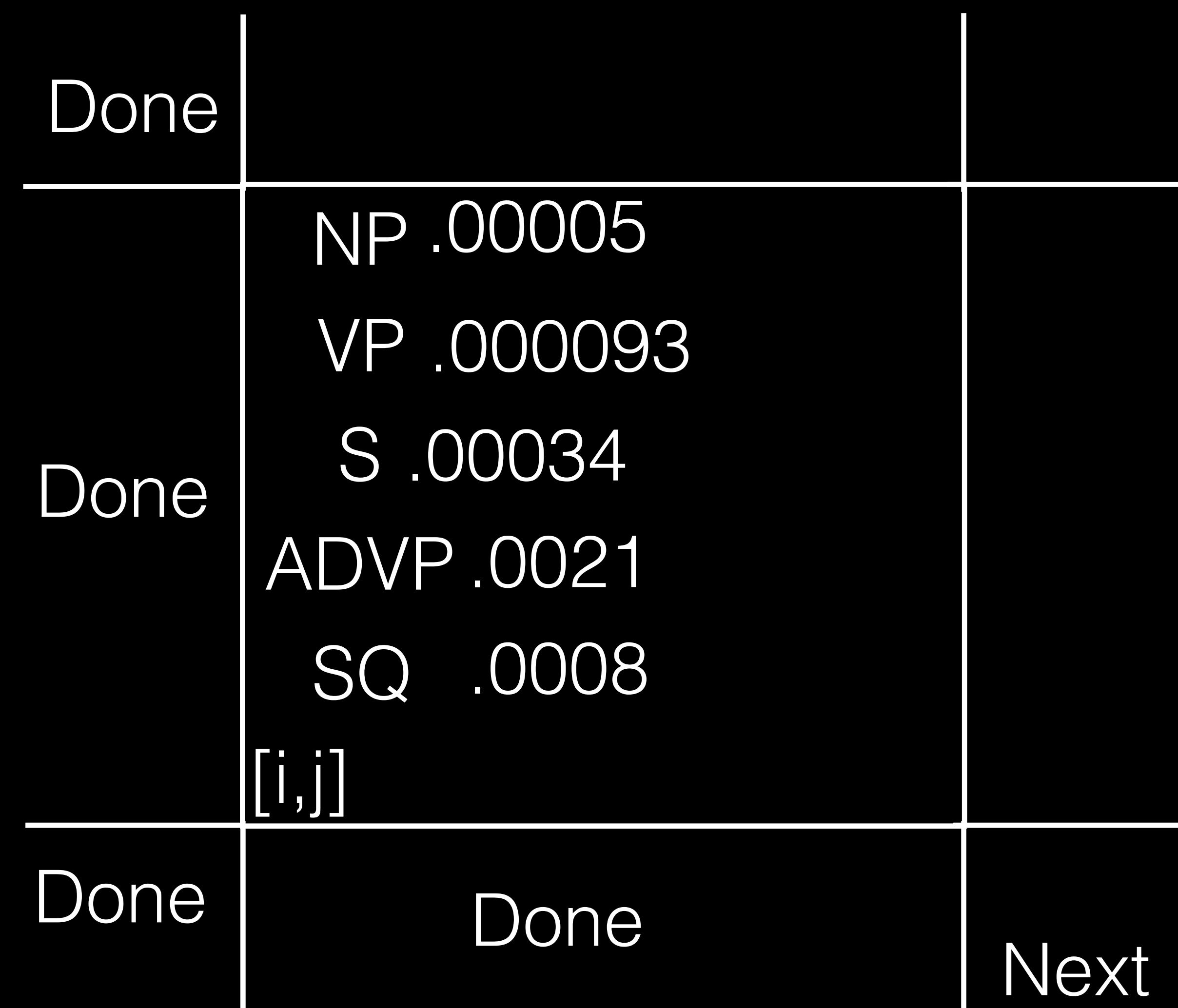
Beaming

- Before moving to [i+1, j+1], throw out some states

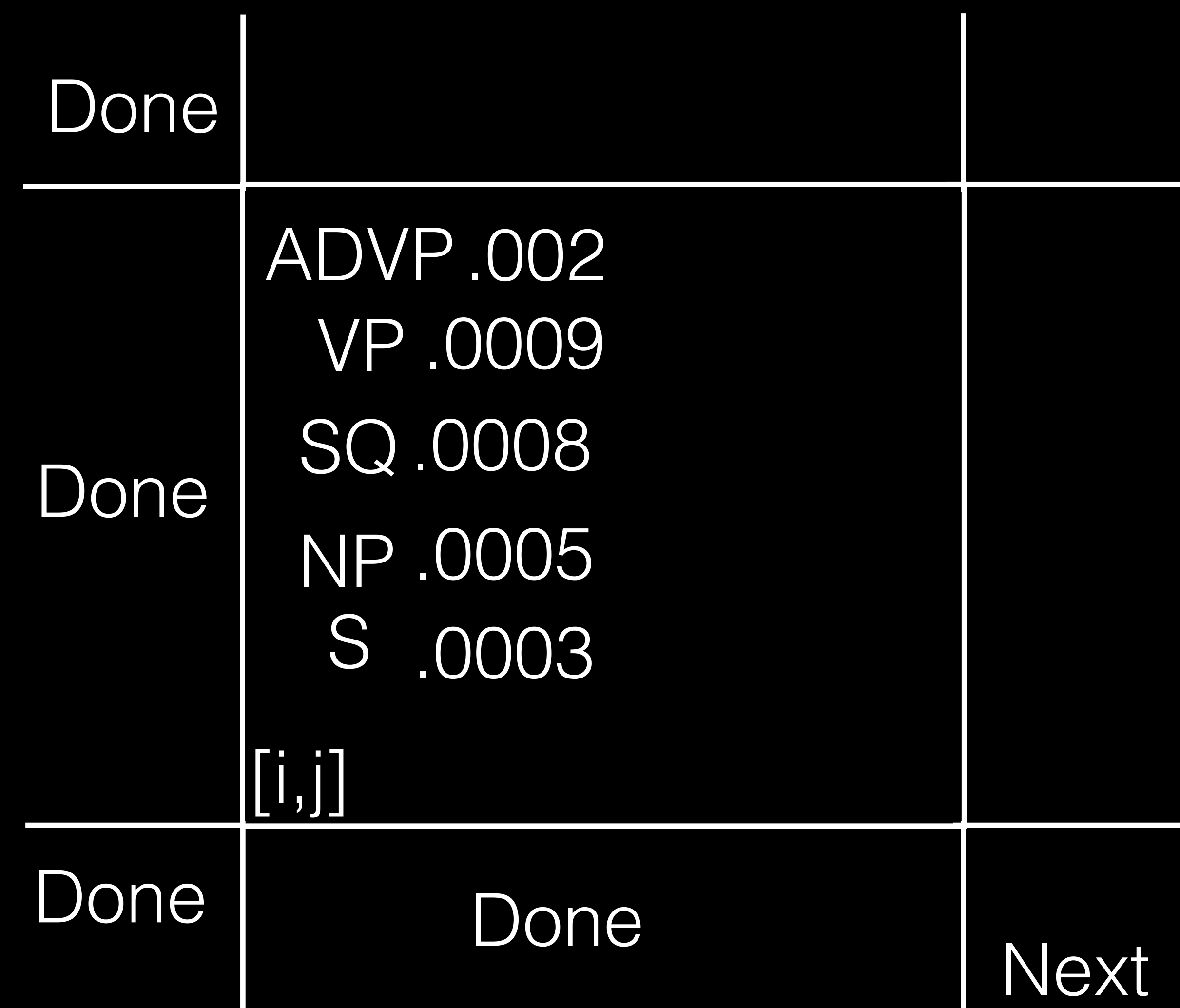
- Scores are *inside costs* = probability of subtree rooted in X

- We want the state most likely to be used in a *successful* parse

- What makes the state likely?

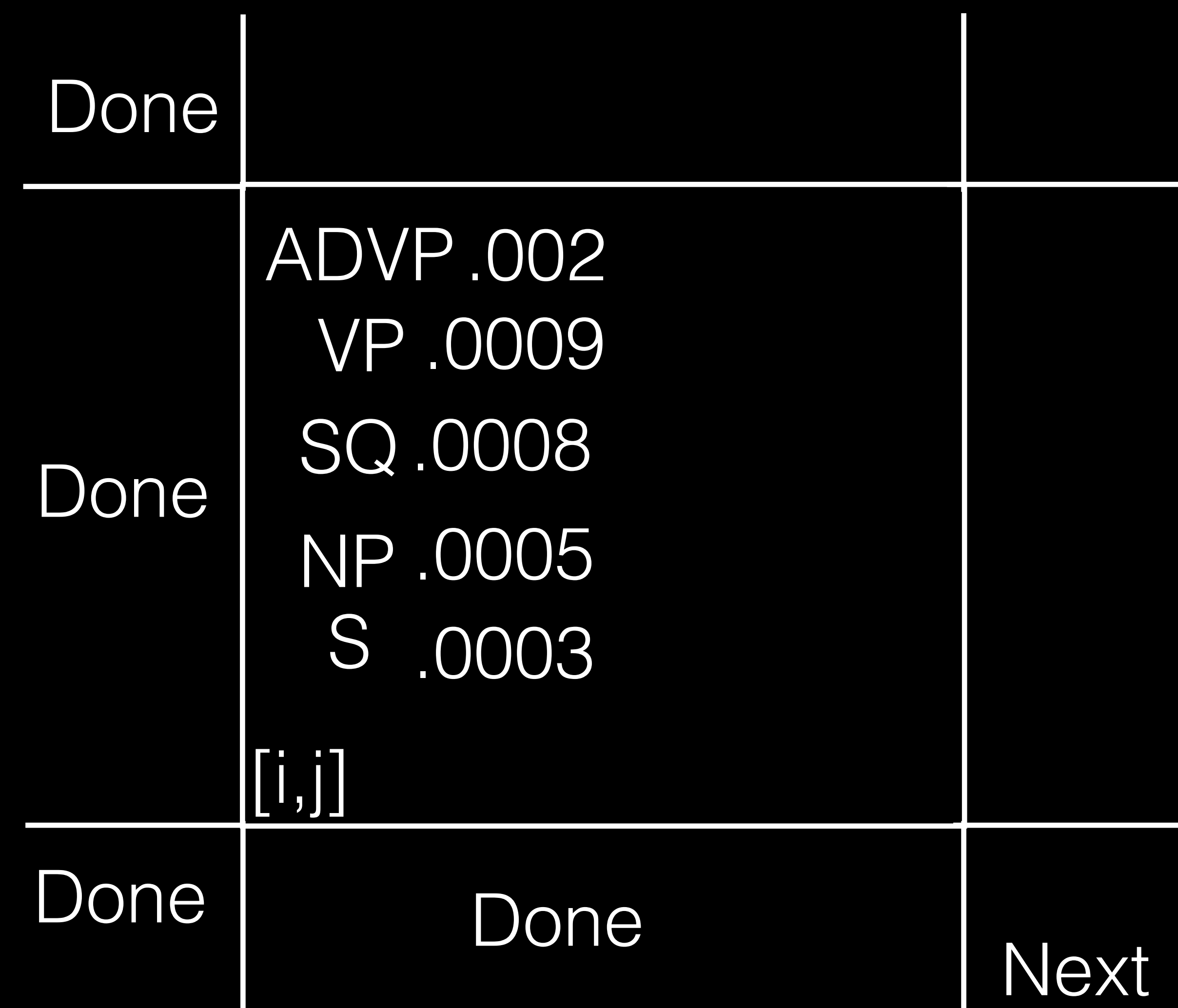


Beaming


Done
ADVP .002
VP .0009
SQ .0008
Done
NP .0005
S .0003
[i,j]
Done
Done
Next

Beaming

- Incorporate *heuristic* of completion (note: A* search!)



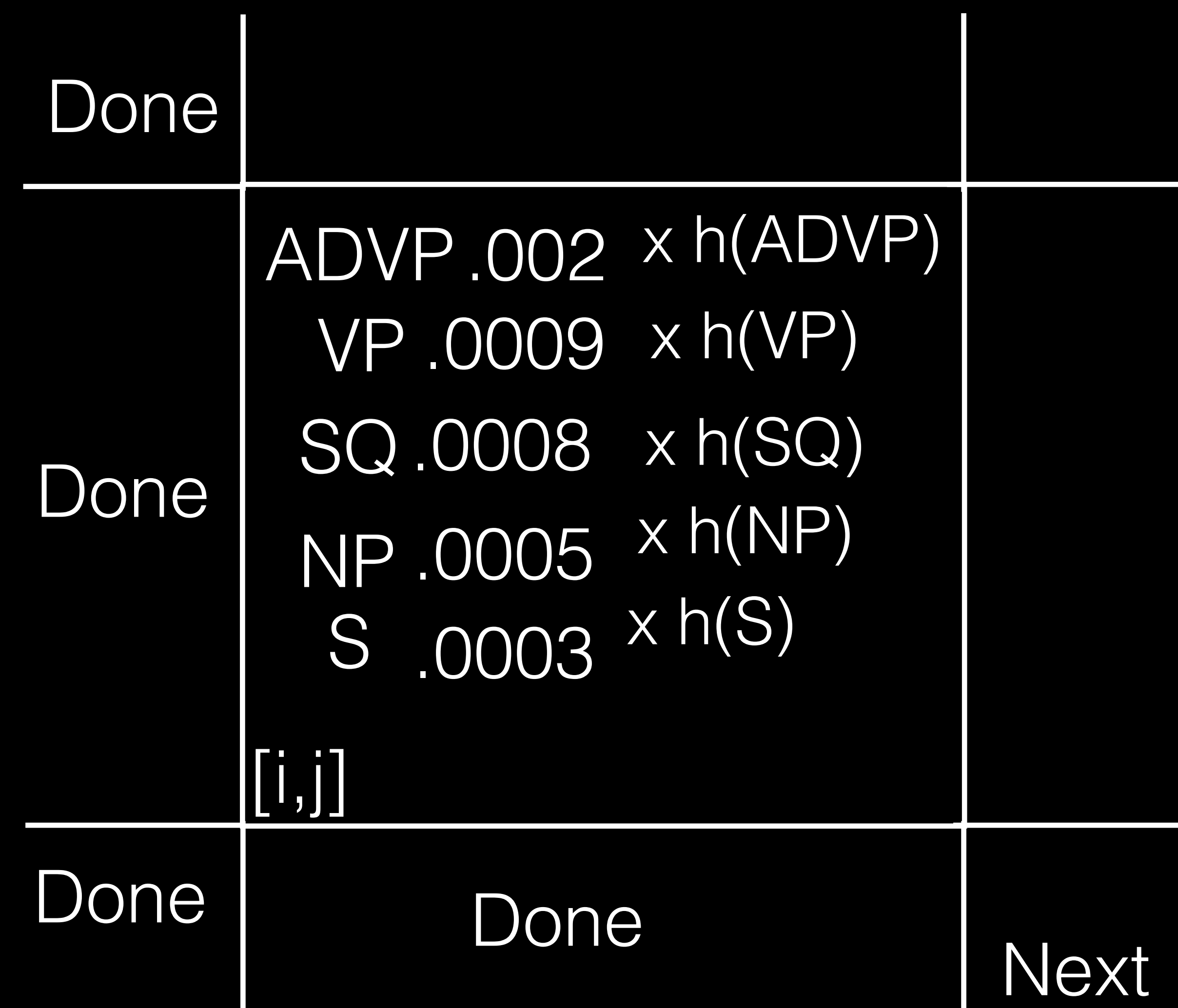
Beaming

- Incorporate *heuristic* of completion (note: A* search!)

Done		
Done	ADVP .002 x h(ADVP) VP .0009 x h(VP) SQ .0008 x h(SQ) NP .0005 x h(NP) S .0003 x h(S) [i,j]	
Done	Done	Next

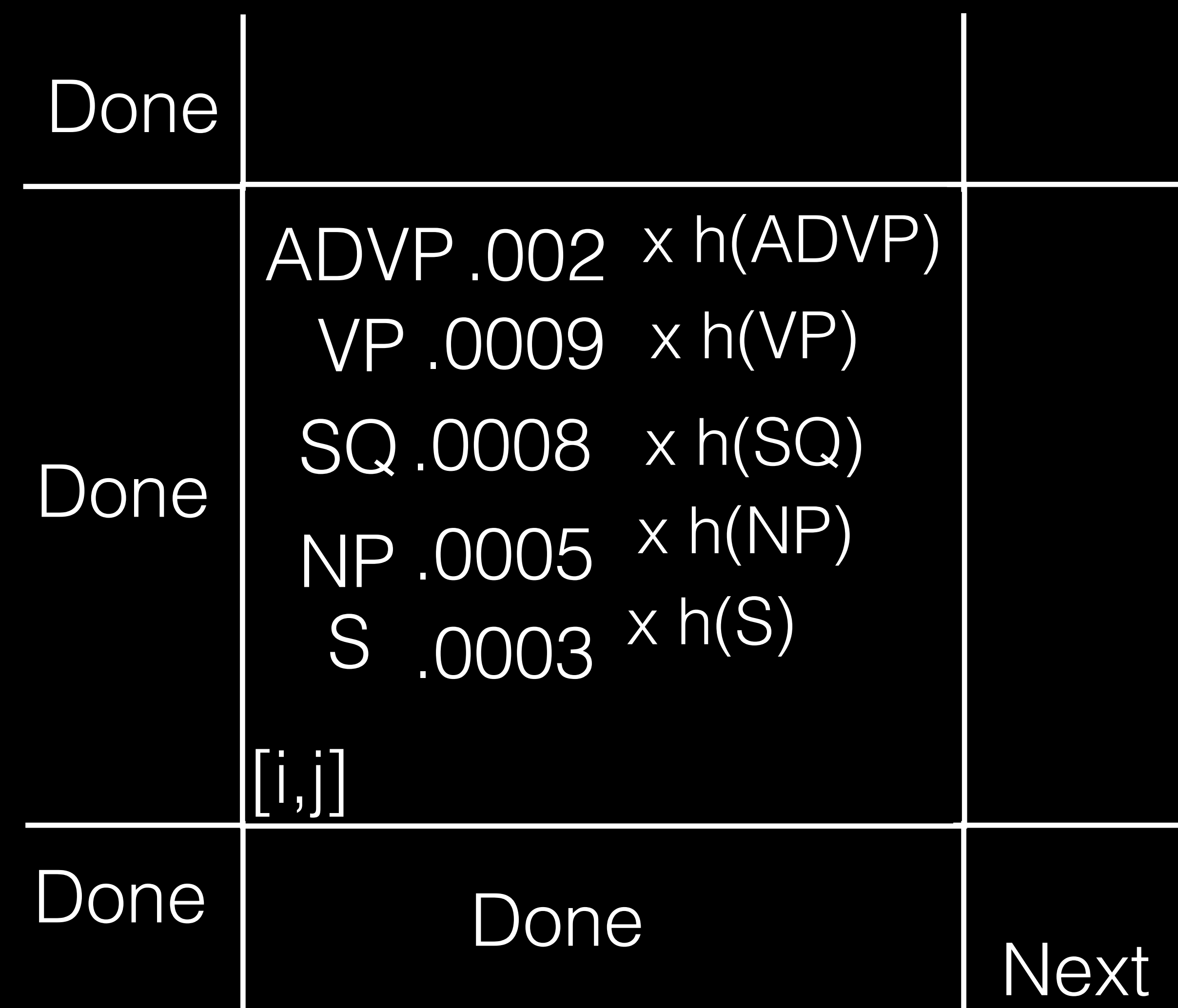
Beaming

- Incorporate *heuristic* of completion (note: A* search!)
- p(X) = prior probability of seeing X (count nonterminals in corpus)



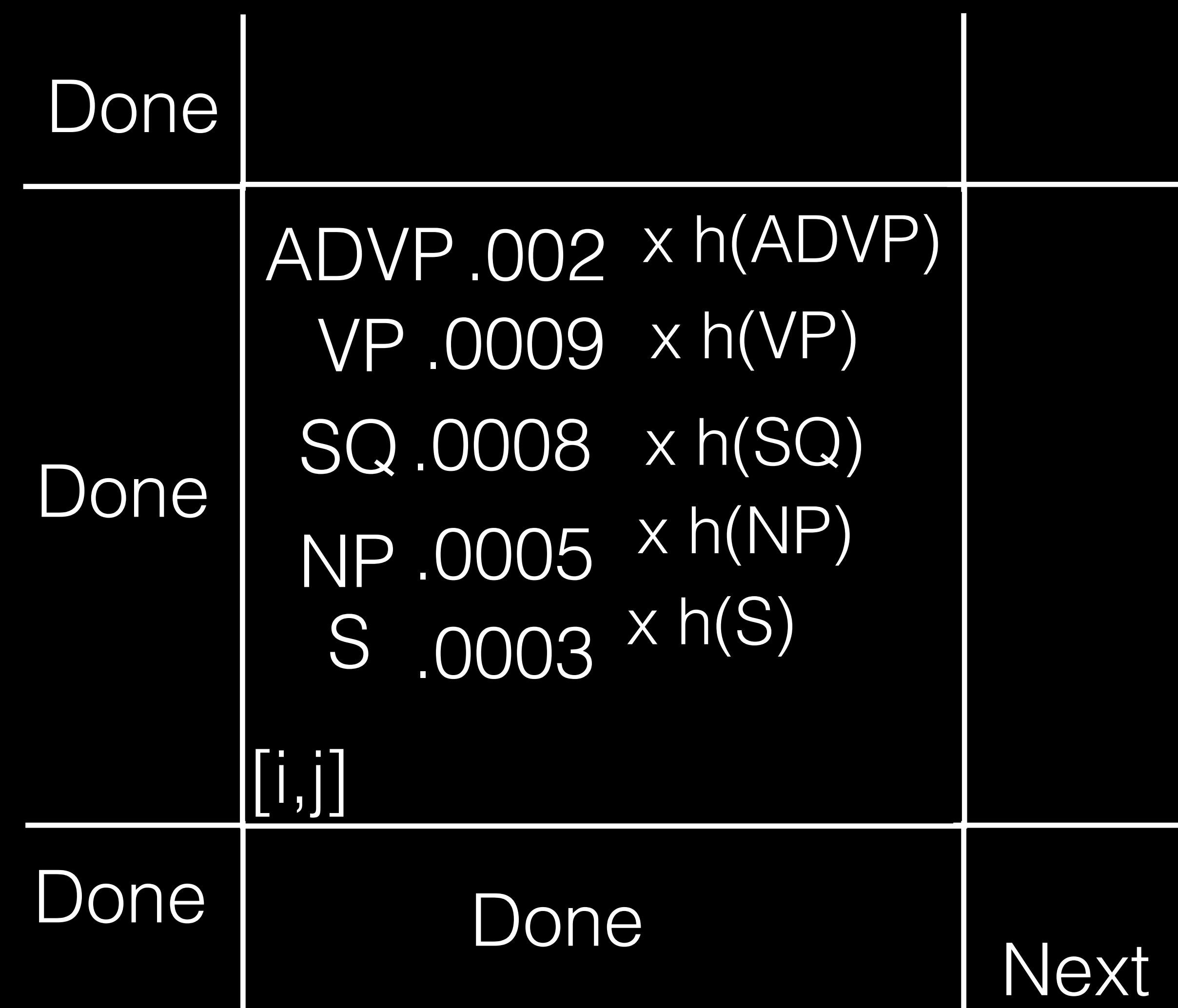
Beaming

- Incorporate *heuristic* of completion (note: A* search!)
- p(X) = prior probability of seeing X (count nonterminals in corpus)
- could subdivide by span size (p(X|j-i))



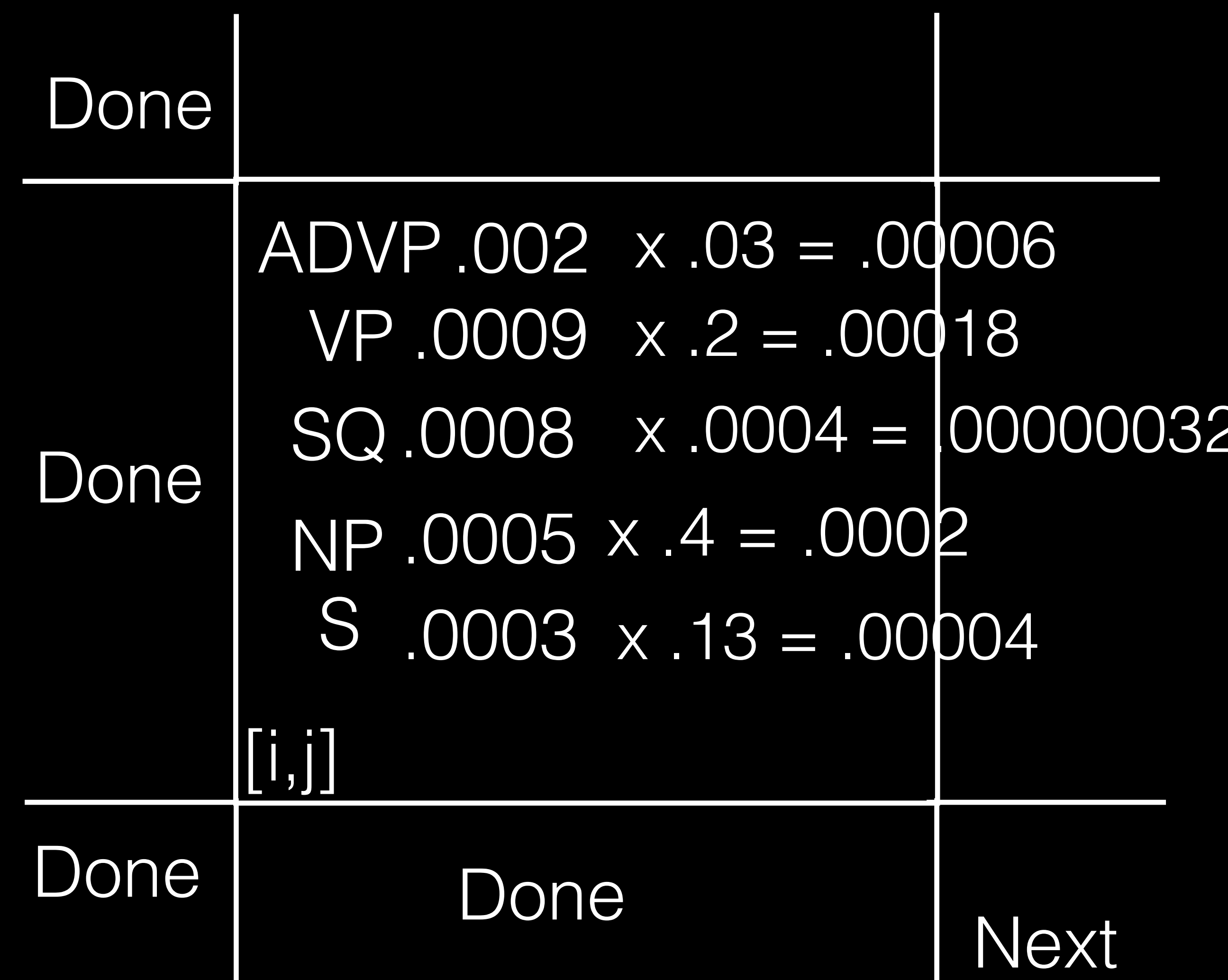
Beaming

- Incorporate *heuristic* of completion (note: A* search!)
- $p(X)$ = prior probability of seeing X (count nonterminals in corpus)
- could subdivide by span size ($p(X|j-i)$)
- could incorporate edge words ($p(X|w_{i-1}, w_j)$)



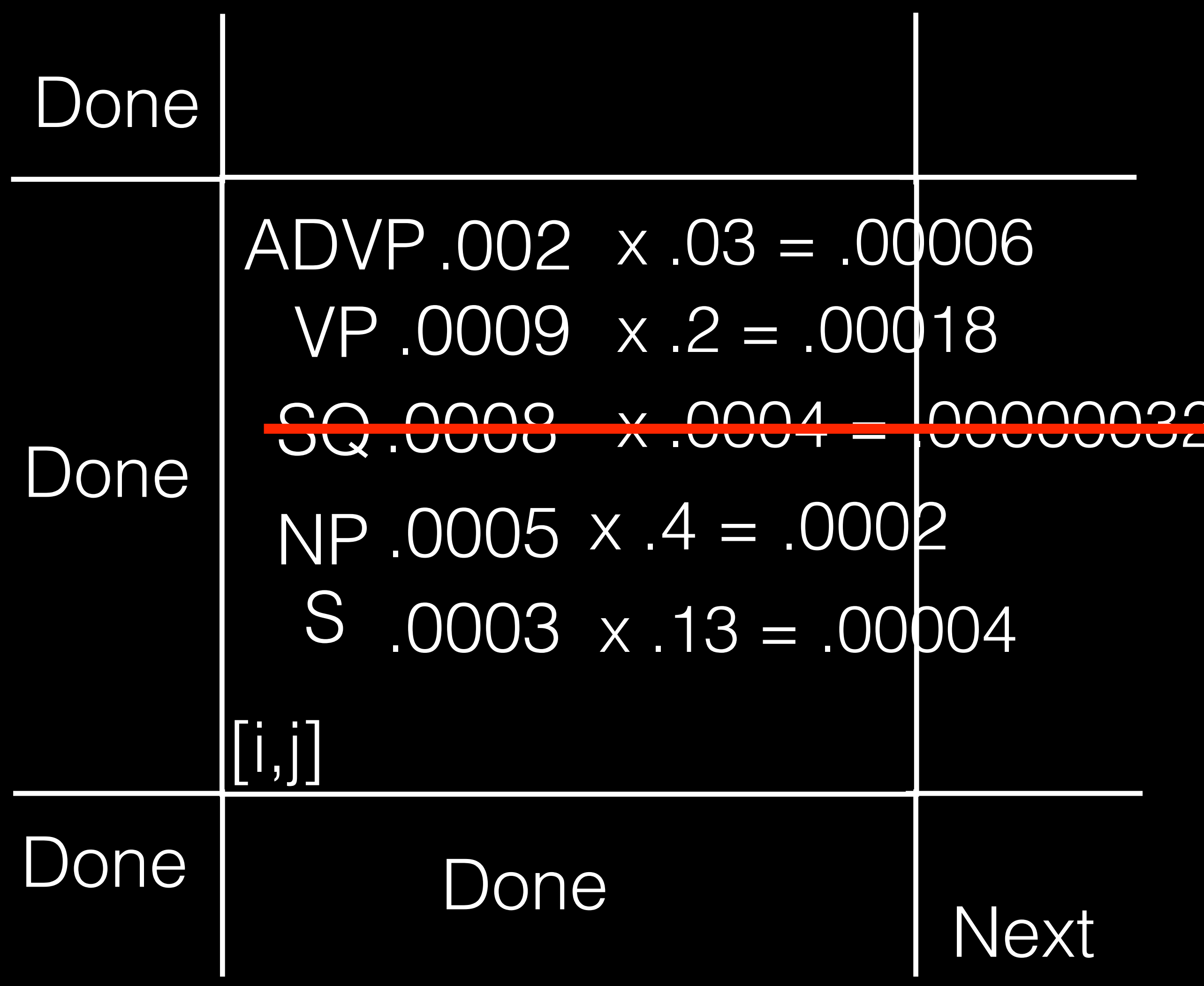
Beaming

- Incorporate *heuristic* of completion (note: A* search!)
- p(X) = prior probability of seeing X (count nonterminals in corpus)
- could subdivide by span size (p(X|j-i))
- could incorporate edge words ($p(X|w_{i-1}, w_j)$)



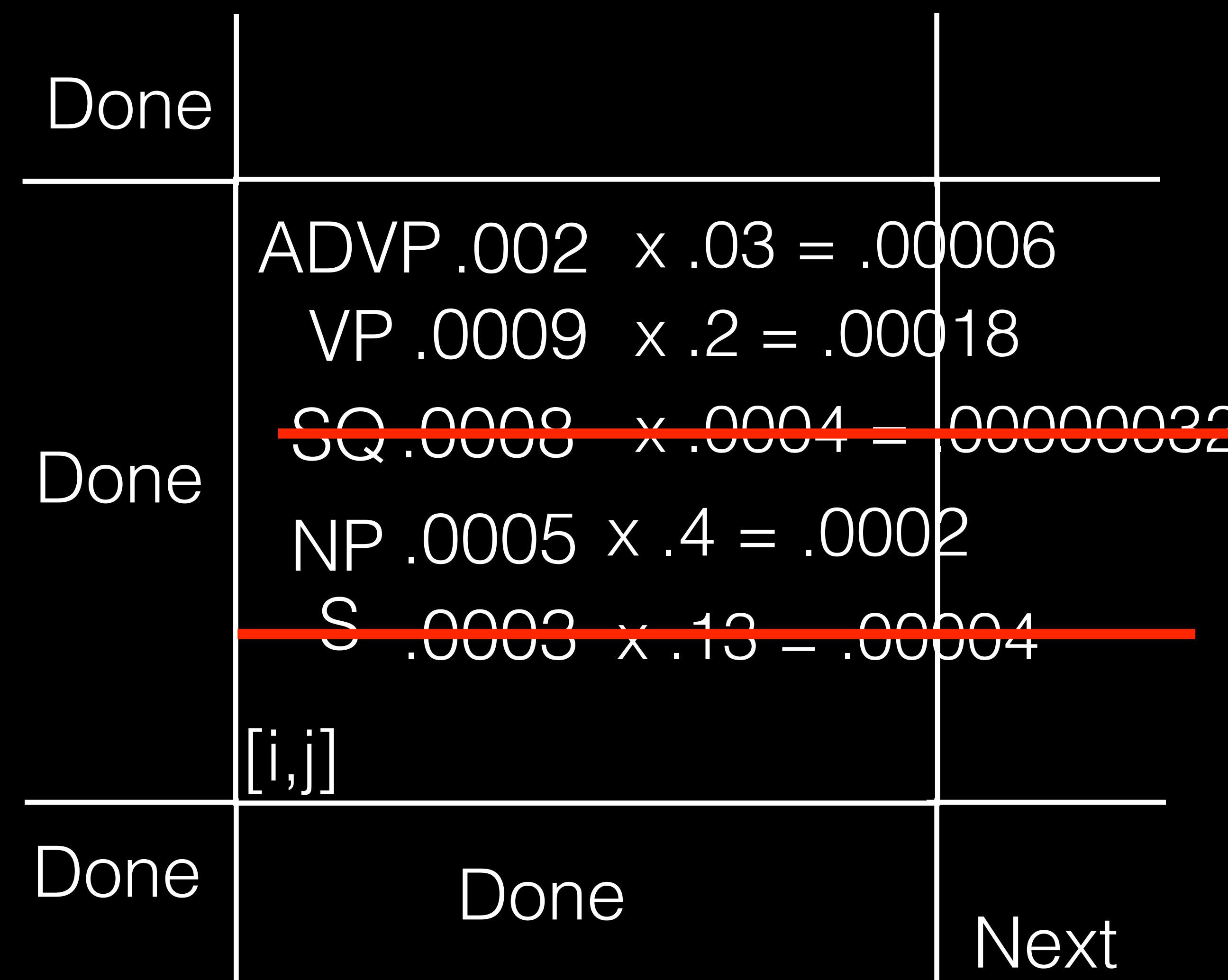
Beaming

- Incorporate *heuristic* of completion (note: A* search!)
- p(X) = prior probability of seeing X (count nonterminals in corpus)
- could subdivide by span size (p(X|j-i))
- could incorporate edge words ($p(X|w_{i-1}, w_j)$)



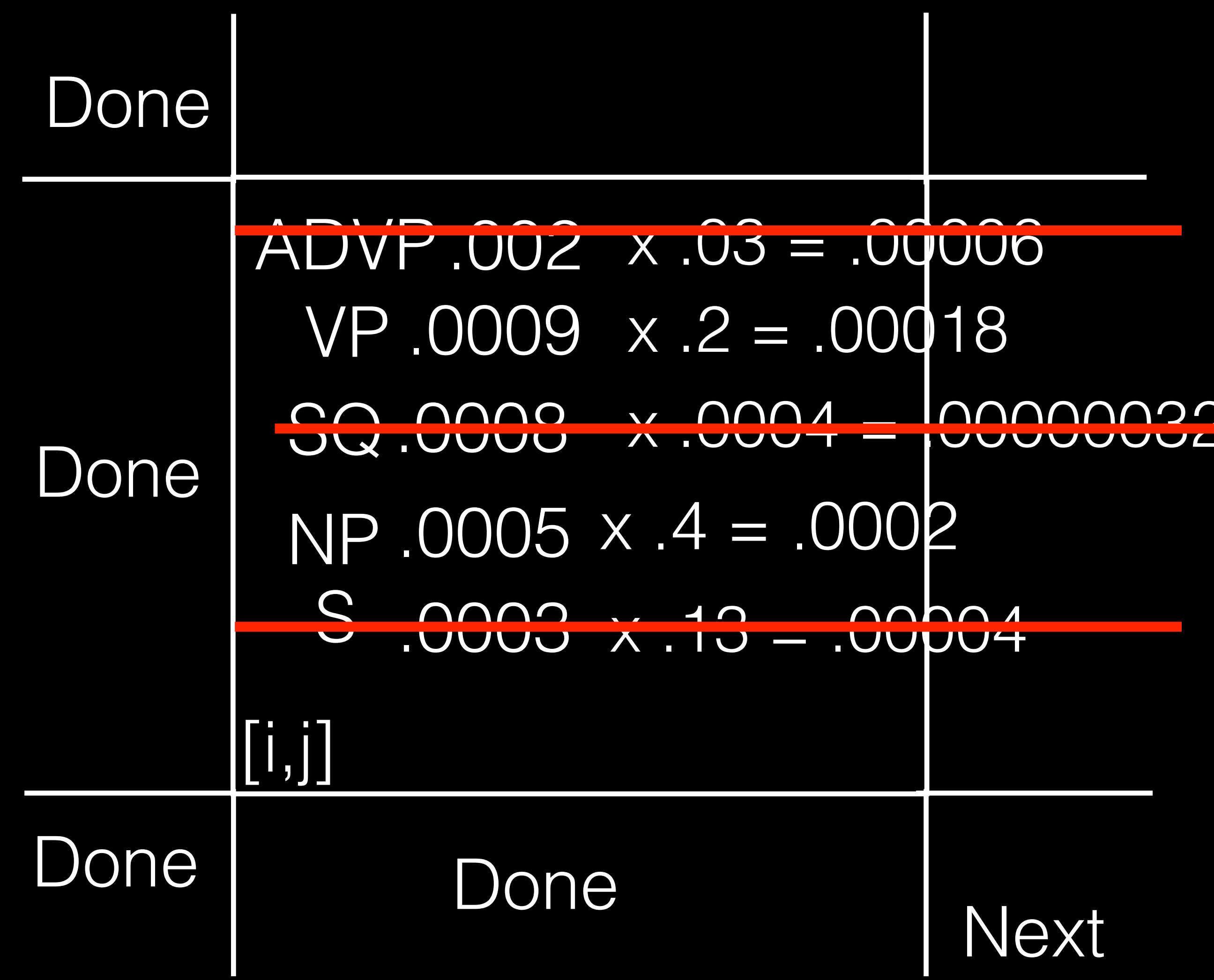
Beaming

- Incorporate *heuristic* of completion (note: A* search!)
- $p(X)$ = prior probability of seeing X (count nonterminals in corpus)
- could subdivide by span size ($p(X|j-i)$)
- could incorporate edge words ($p(X|w_{i-1}, w_j)$)



Beaming

- Incorporate *heuristic* of completion (note: A* search!)
- p(X) = prior probability of seeing X (count nonterminals in corpus)
- could subdivide by span size (p(X|j-i))
- could incorporate edge words (p(X|w_{i-1}, w_j))



Summary

Summary

- PCFG parsing

Summary

- PCFG parsing
- Estimating Vanilla PCFGs

Summary

- PCFG parsing
- Estimating Vanilla PCFGs
- Markovization

Summary

- PCFG parsing
- Estimating Vanilla PCFGs
- Markovization
- Parent Annotation

Summary

- PCFG parsing
- Estimating Vanilla PCFGs
- Markovization
- Parent Annotation
- Lexicalization

Summary

- PCFG parsing
- Estimating Vanilla PCFGs
- Markovization
- Parent Annotation
- Lexicalization
- Head binarization

Summary

- PCFG parsing
- Estimating Vanilla PCFGs
- Markovization
- Parent Annotation
- Lexicalization
- Head binarization
- Smoothing rich models

Summary

- PCFG parsing
- Estimating Vanilla PCFGs
- Markovization
- Parent Annotation
- Lexicalization
- Head binarization
- Smoothing rich models
- Beaming the search