Probabilistic Parsing: Issues & Improvement

LING 571 — Deep Processing Techniques for NLP

Shane Steinert-Threlkeld

Announcements

- Shane traveling on Wednesday for a conference
  - Our wonderful TA Cassie will deliver the lecture (intro to dependency parsing)
  - **No office hours** on Wednesday as well.

Notes on HW #3

- Python's `range` has many use cases by manipulating start/end, and step
  - `range(n)` is equivalent to `range(0, n, 1)`
- Reminder: the `rhs=` argument in NLTK's `grammar.productions()` method only matches the *first* symbol, not an entire string
  - You'll want to implement an efficient look-up based on RHS
- HW3: compare your output to running HW1 parser on the same grammar/sentences
  - order of output in ambiguous sentences could differ
- We will provide grammars in CNF; don't need to use your HW2 for that

Indigenous Peoples' Day

- Seattle/Sealth/Si'ahl

Indigenous Peoples’ Day

- Seattle/Sealth/Si'ahl
- The Lushootseed spelling [IPA] of Chief Seattle/Sealth/Si'ahl:
  - siʔał [ˈsiʔaːɬ]
  - ɬ : voiceless alveolar lateral fricative

Indigenous Peoples’ Day

- Seattle/Sealth/Si'ahl
- The Lushootseed spelling [IPA] of Chief Seattle/Sealth/Si'ahl:
    - siʔaɬ [ˈsiʔaːɬ]
    - ɬ : voiceless alveolar lateral fricative
- Duwamish — Dxʷdəwʔabš [dxʷdɐwʔabʃ]

Indigenous Peoples' Day

- Seattle/Sealth/Si'ahl
- The Lushootseed spelling [IPA] of Chief Seattle/Sealth/Si'ahl:
  - siʔaɬ [ˈsiʔaːɬ]
  - ɬ : voiceless alveolar lateral fricative
- Duwamish — Dxʷdəwʔabš [dxʷdɐwʔabʃ]
- Vi Hilbert (Skagit elder, died in 2008; *last* native speaker) recording:
  - Source, with more Lushootseed: https://historylink.org/File/8156

Indigenous Peoples’ Day

- Seattle/Sealth/Si'ahl
- The Lushootseed spelling [IPA] of Chief Seattle/Sealth/Si'ahl:
  - siʔał [ˈsiʔaːɬ]
  - ɬ : voiceless alveolar lateral fricative
- Duwamish — Dxʷdəwʔabš [dxʷdɐwʔabʃ]
- Vi Hilbert (Skagit elder, died in 2008; *last* native speaker) recording:
  - Source, with more Lushootseed: https://historylink.org/File/8156

Indigenous Peoples’ Day

- Seattle/Sealth/Si'ahl
- The Lushootseed spelling [IPA] of Chief Seattle/Sealth/Si'ahl:
  - siʔał [ˈsiʔaːɬ]
  - ɬ : voiceless alveolar lateral fricative
- Duwamish — Dxʷdəwʔabš [dxʷdɐwʔabʃ]
- Vi Hilbert (Skagit elder, died in 2008; *last* native speaker) recording:
  - Source, with more Lushootseed: https://historylink.org/File/8156
- IPA resources:
  - https://en.wikipedia.org/wiki/International_Phonetic_Alphabet
  - http://web.mit.edu/6.mitx/www/24.900%20IPA/IPAapp.html

Indigenous Peoples' Day

- Studying non-English languages gives more holistic insight for NLP tasks
  - Many interesting phenomena in non-Indo-European languages

Indigenous Peoples' Day

- Studying non-English languages gives more holistic insight for NLP tasks
  - Many interesting phenomena in non-Indo-European languages
- Important that our technologies work for all languages of the world

Indigenous Peoples' Day

- Studying non-English languages gives more holistic insight for NLP tasks
  - Many interesting phenomena in non-Indo-European languages
- Important that our technologies work for all languages of the world
- Lushootseed exhibits debatable distinction between verbs and nouns [link to Glottolog page for more references]
  -
?ux̌ʷ	ti	**sbiaw**
goes	that-which	**is-a-coyote**

    "The/a coyote goes"
  -
sbiaw	ti	**?ux̌ʷ**
is-a-coyote	that-which	**goes**

    "The one who goes is a coyote"
  - (Translation distinction provided for clarity — semantically equivalent)

Indigenous Peoples’ Day

- Studying non-English languages gives more holistic insight for NLP tasks
  - Many interesting phenomena in non-Indo-European languages
- Important that our technologies work for all languages of the world
- Lushootseed exhibits debatable distinction between verbs and nouns [link to Glottolog page for more references]

?ux̌^{w}	ti	**sbiaw**
goes	that-which	**is-a-coyote**

“The/a coyote goes”

sbiaw	ti	**?ux̌^{w}**
is-a-coyote	that-which	**goes**

“The one who goes is a coyote”

- (Translation distinction provided for clarity — semantically equivalent)

via Beck, 2013

- Lillooet Salish quantification has repercussions for e.g. English (Matthewson 2001)

Title logo

Can you name any other languages indigenous to the Americas?

✓ 0

Nobody has responded yet.

Hang tight! Responses are coming in.

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Title logo

How many indigenous languages do you think there are in the US today?

✓ 0

< 10	0
11-100	0
101-200	0
201-300	0
> 300	0

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Title logo

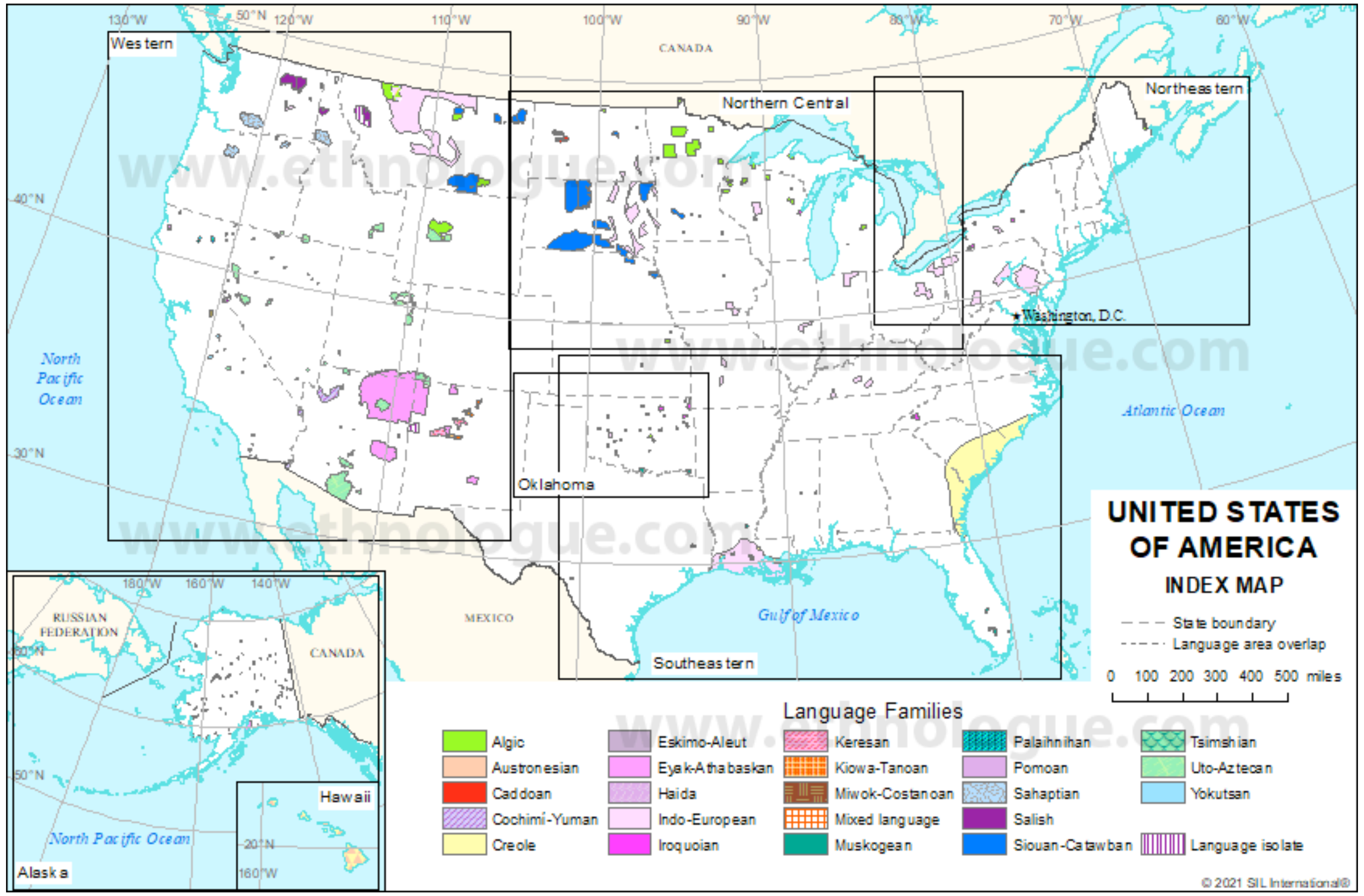
How many languages do you think were spoken in the now-US at the time Europeans arrived?

< 10	0
11-100	0
101-200	0
201-300	0
> 300	0

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Languages in the U.S.

- Current estimate (Ethnologue): 238
  - 226 living, 12 extinct
  - Lushootseed: “reawakening”
  - 195 indigenous
- Navajo: ~170,000 speakers
  - Not in U.S. top 25 by pop size
- Many, many endangered; increased need for revitalization efforts



https://www.ethnologue.com/map/US_x_

Indigenous Peoples' Day: Resources

- UW American Indian Studies Courses
  - (Sometimes including language courses, e.g. Southern Lushootseed, Salish, from Tami Hohn)
- Lushootseed resources: https://tulaliplushootseed.com/
- Computational Methods in the Study of Endangered Languages: https://computel-workshop.org/
- AmericasNLP: https://turing.iimas.unam.mx/americasnlp/
  - Workshop annually
  - Usually with a shared task (including great data resources!)

Unit Testing

Unit Testing

- Strategy of testing individual pieces of code in isolation
- Helps ensure:
  - Basic functionality in isolation
  - Complex functionality when individual components are combined
- In many industry jobs, you can't commit code without unit tests!
- Useful practice: write tests *before* implementing

Unit Testing in Python

- Many good tutorials on the web
  - https://diveinto.org/python3/unit-testing.html
- In a nutshell:

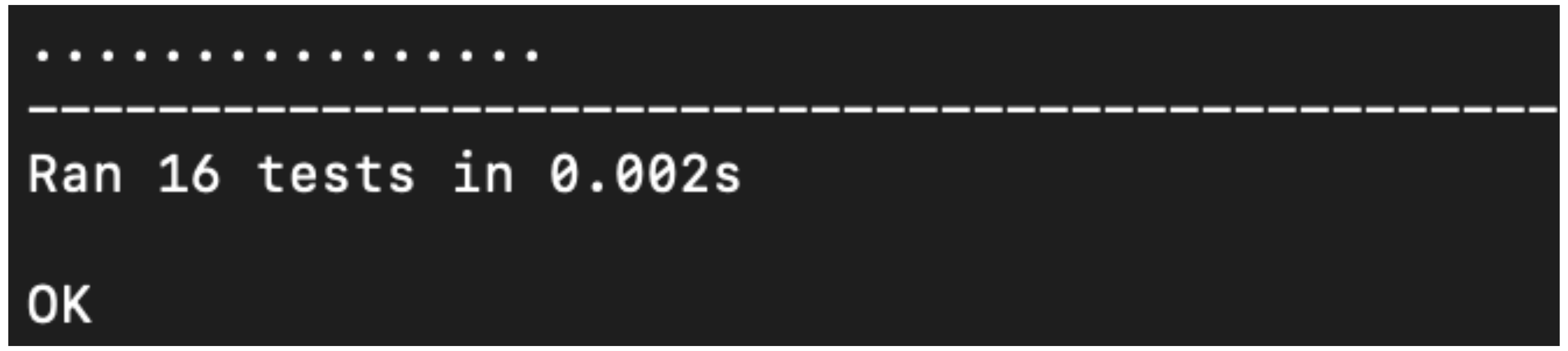
```
from unittest import TestCase

class longTests(TestCase):
 def test_three(self):
   length_3_rule = parse_productions('A -> B C D')
   target_rules = parse_productions('''A -> B _X0_
                                      _X0_ -> C D''')
   self.assertSetEqual(set(target_rules),
                       set(fix_long_rules(length_3_rule)))
```

Unit Testing in Python

- Built-in unittest module/library:

```
python -m unittest hw2.py
```



```
................
-----------------------------------------------
Ran 16 tests in 0.002s

OK
```

Unit Testing

- Good practice:
  - Save input that crashes your program for a unit test
- Other popular unit testing frameworks for python (e.g. in 574):
  - `pytest`: https://docs.pytest.org/
    - Nice auto-discovery of tests based on file, class, and method name
    - Works with native assert statements, not special ones
    - …
- NB: passing tests is necessary, not sufficient, for knowing your code is correct

Today's Plan

- PCFG Induction example
- Problems with PCFGs
  - Independence
  - Lack of lexical conditioning
- Improving PCFGs
  - Coverage (3 methods)
  - Efficiency

PCFG Induction

Learning Probabilities

- Simplest way:
  - Use treebank of parsed sentences

Learning Probabilities

- Simplest way:
    - Use treebank of parsed sentences
    - To compute probability of a rule, count:

Learning Probabilities

- Simplest way:
  - Use treebank of parsed sentences
  - To compute probability of a rule, count:
    - Number of times a nonterminal is expanded: $\Sigma_{\gamma}\text{Count}(\alpha \rightarrow \gamma)$

Learning Probabilities

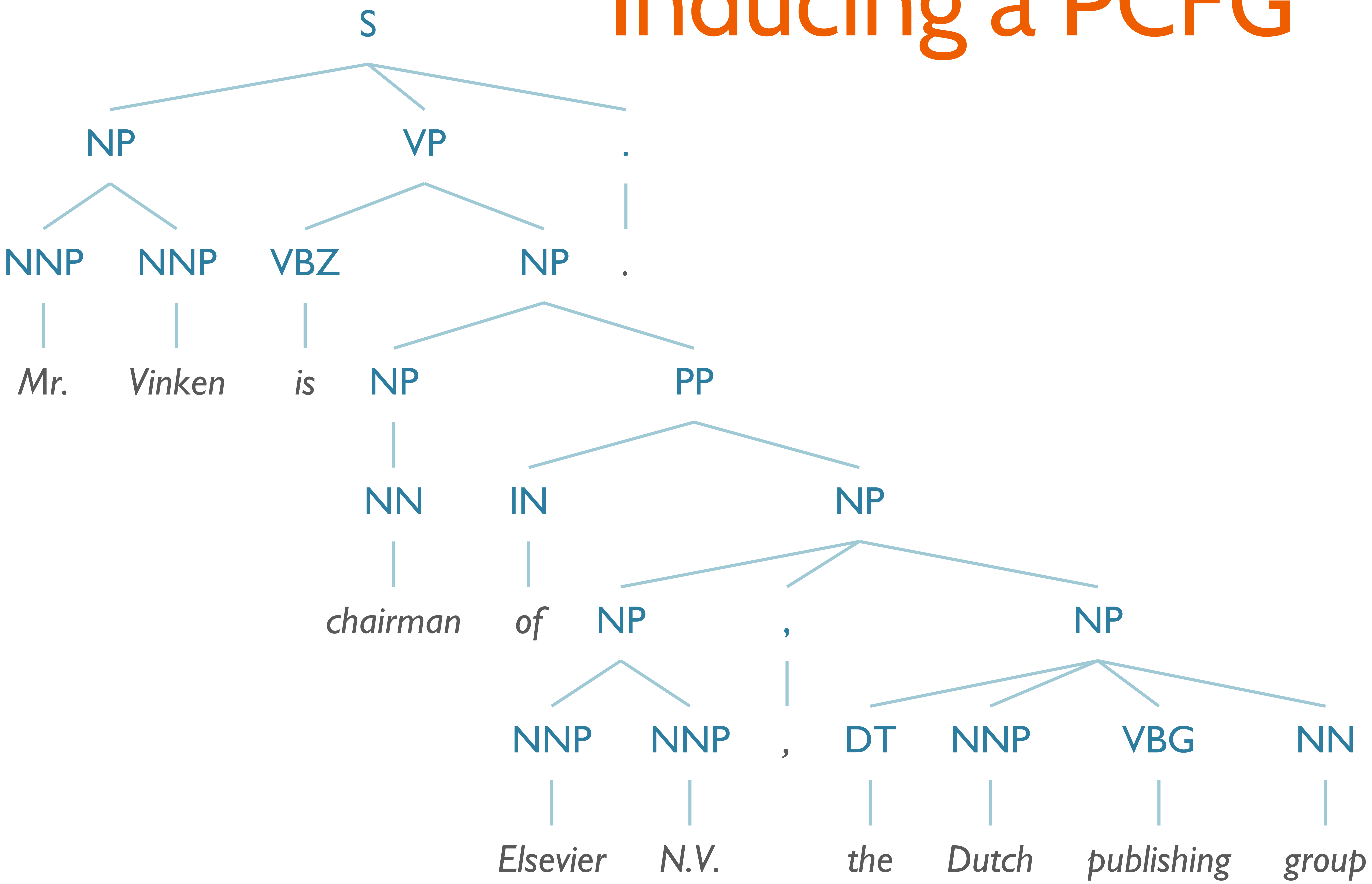
- Simplest way:
  - Use treebank of parsed sentences
  - To compute probability of a rule, count:
    - Number of times a nonterminal is expanded: $\Sigma_\gamma \text{Count}(\alpha \rightarrow \gamma)$
    - Number of times a nonterminal is expanded by a given rule: $\text{Count}(\alpha \rightarrow \beta)$

Learning Probabilities

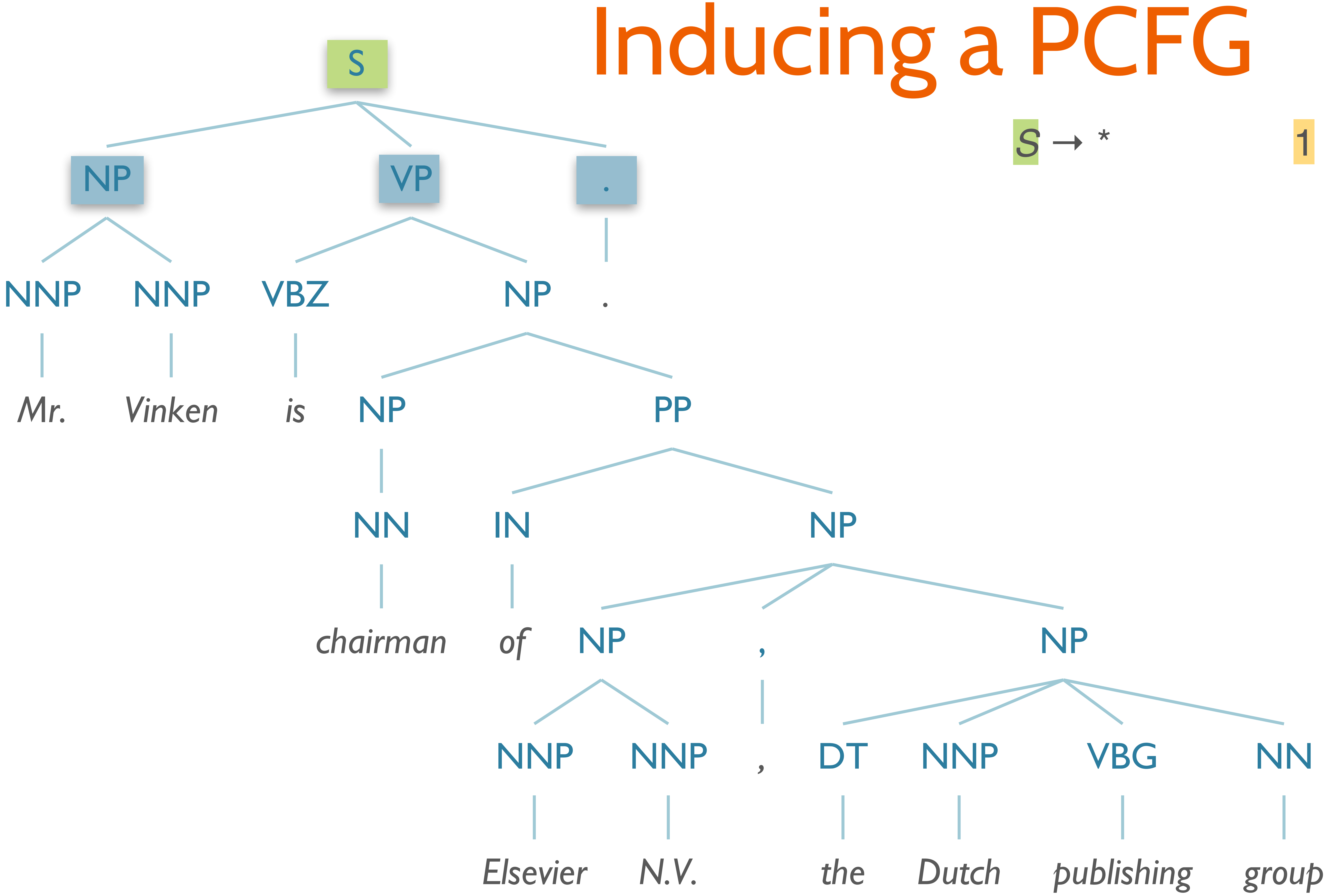
- Simplest way:
  - Use treebank of parsed sentences
  - To compute probability of a rule, count:
    - Number of times a nonterminal is expanded: Σ_γCount$(\alpha \rightarrow \gamma)$
    - Number of times a nonterminal is expanded by a given rule: Count$(\alpha \rightarrow \beta)$

$$P(\alpha \rightarrow \beta \mid \alpha) = \frac{Count(\alpha \rightarrow \beta)}{\sum_\gamma Count(\alpha \rightarrow \gamma)} = \frac{Count(\alpha \rightarrow \beta)}{Count(\alpha)}$$

Inducing a PCFG

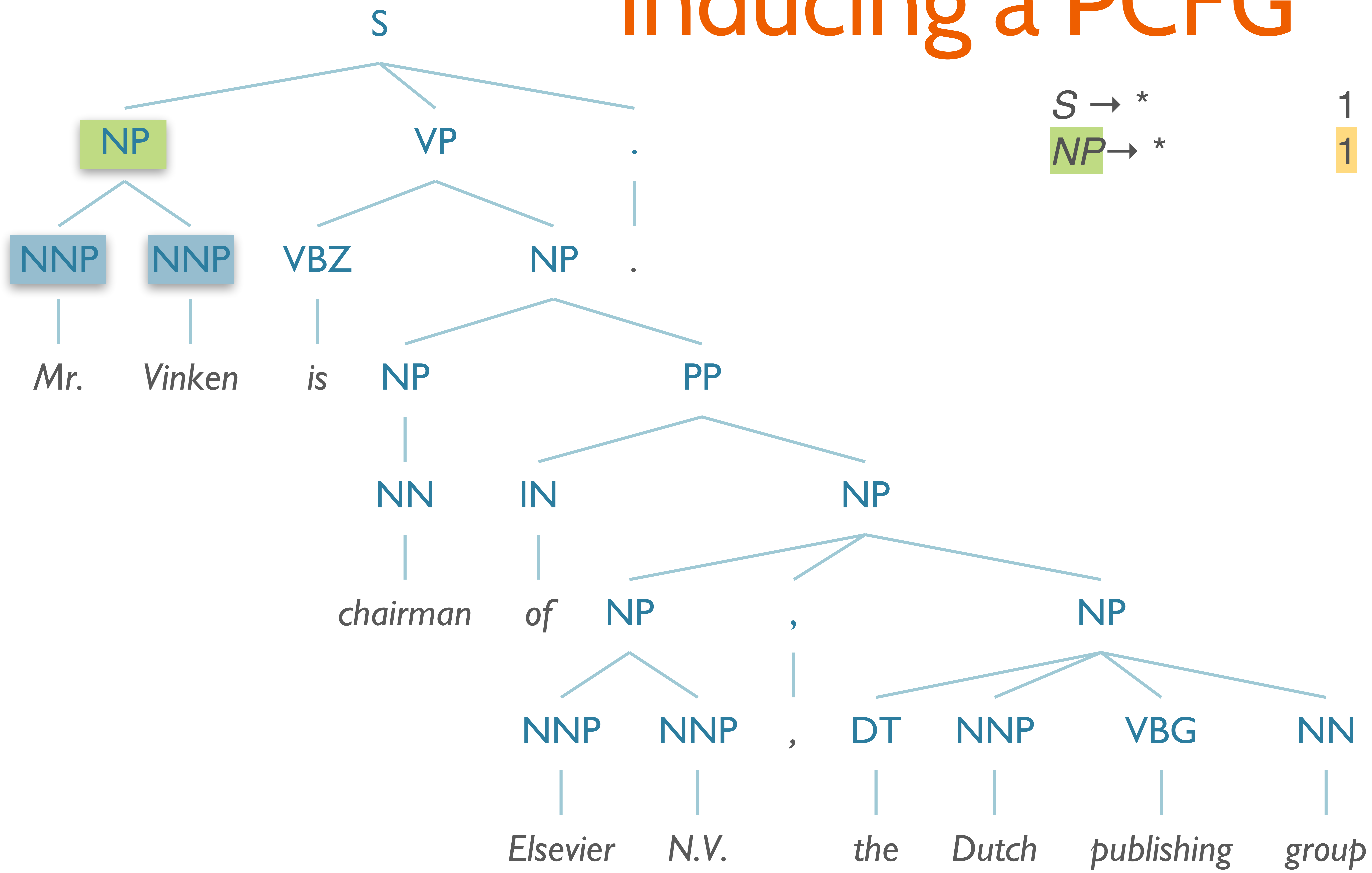


Inducing a PCFG



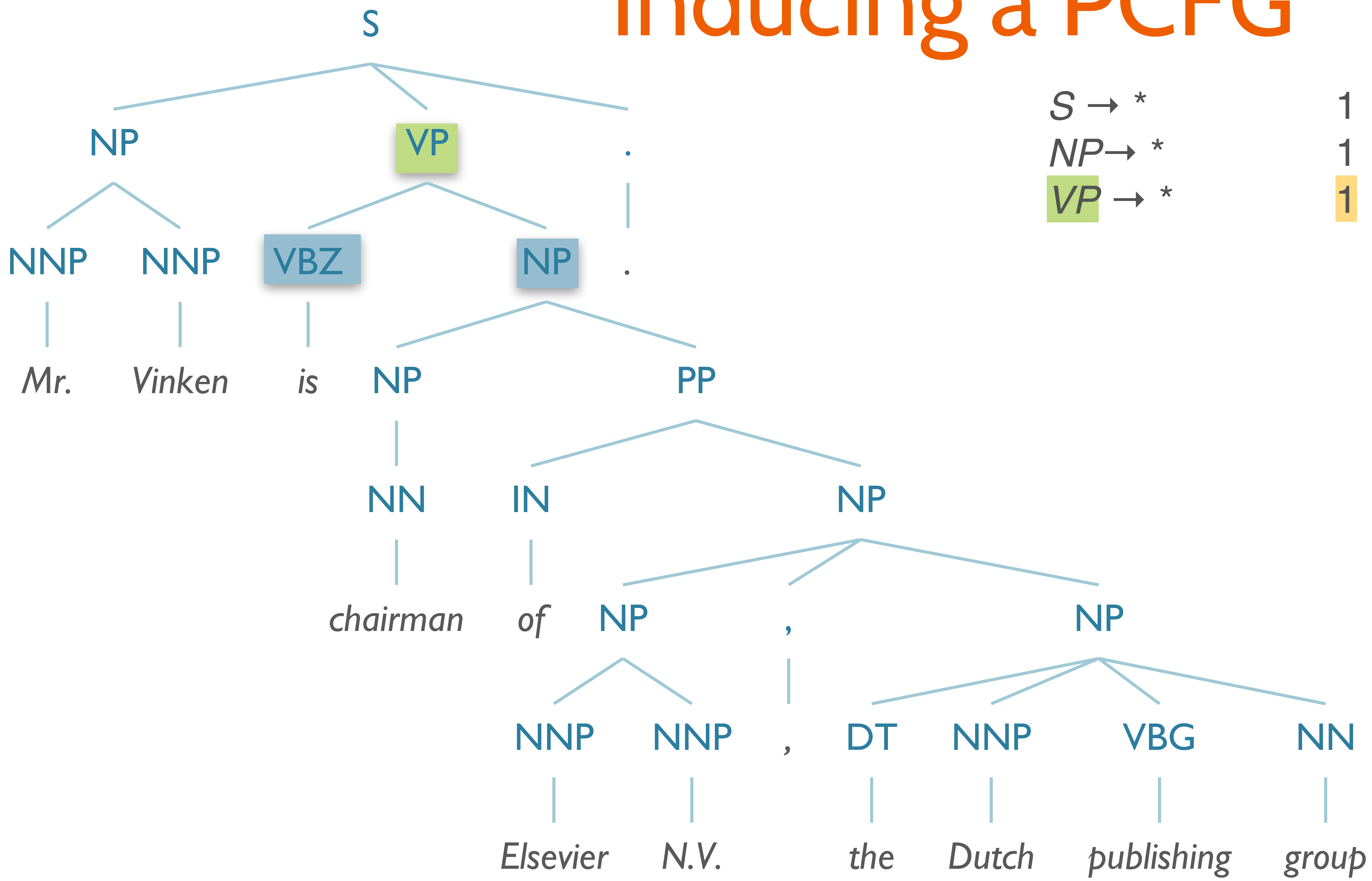
S → *	1	S → NPVP .	1

Inducing a PCFG



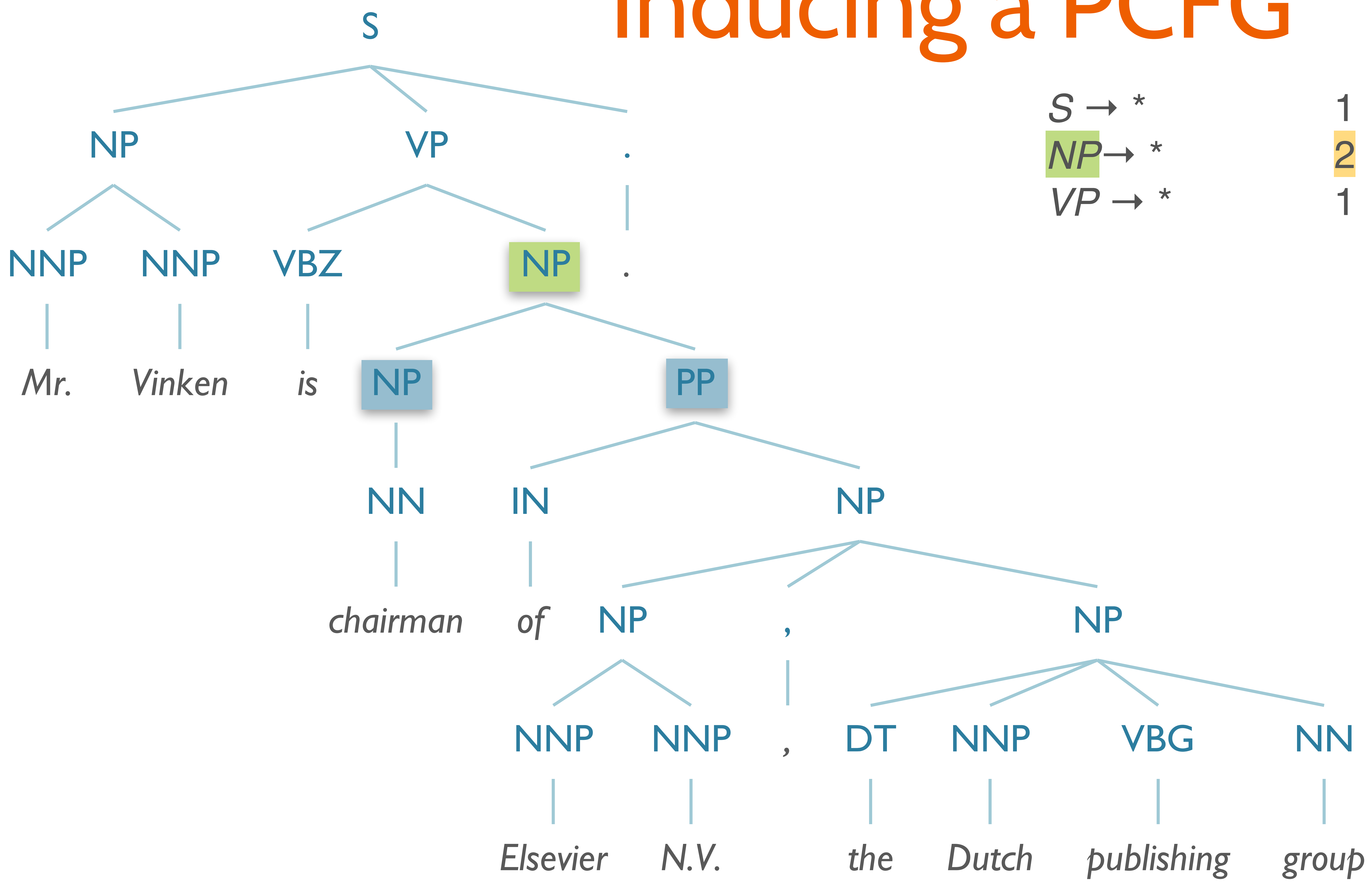
S → *	1	S → NP VP .	1
NP→ *	1	NP→ NNP NNP	1

Inducing a PCFG



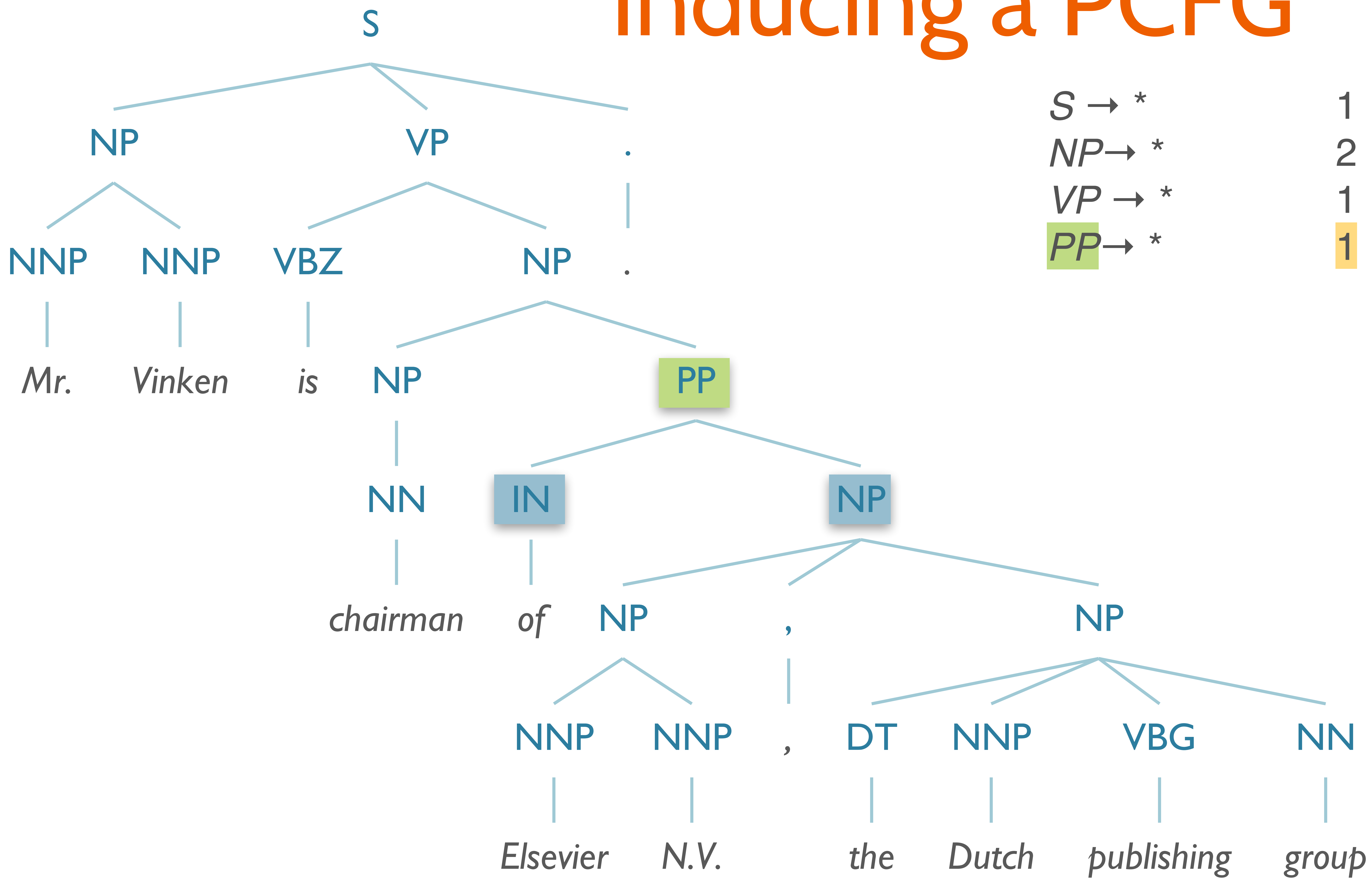
S → *	1	*S* → *NP VP .*	1
NP→ *	1	*NP* → *NNP NNP*	1
VP → *	1	*VP* → *VBZ NP*	1

Inducing a PCFG



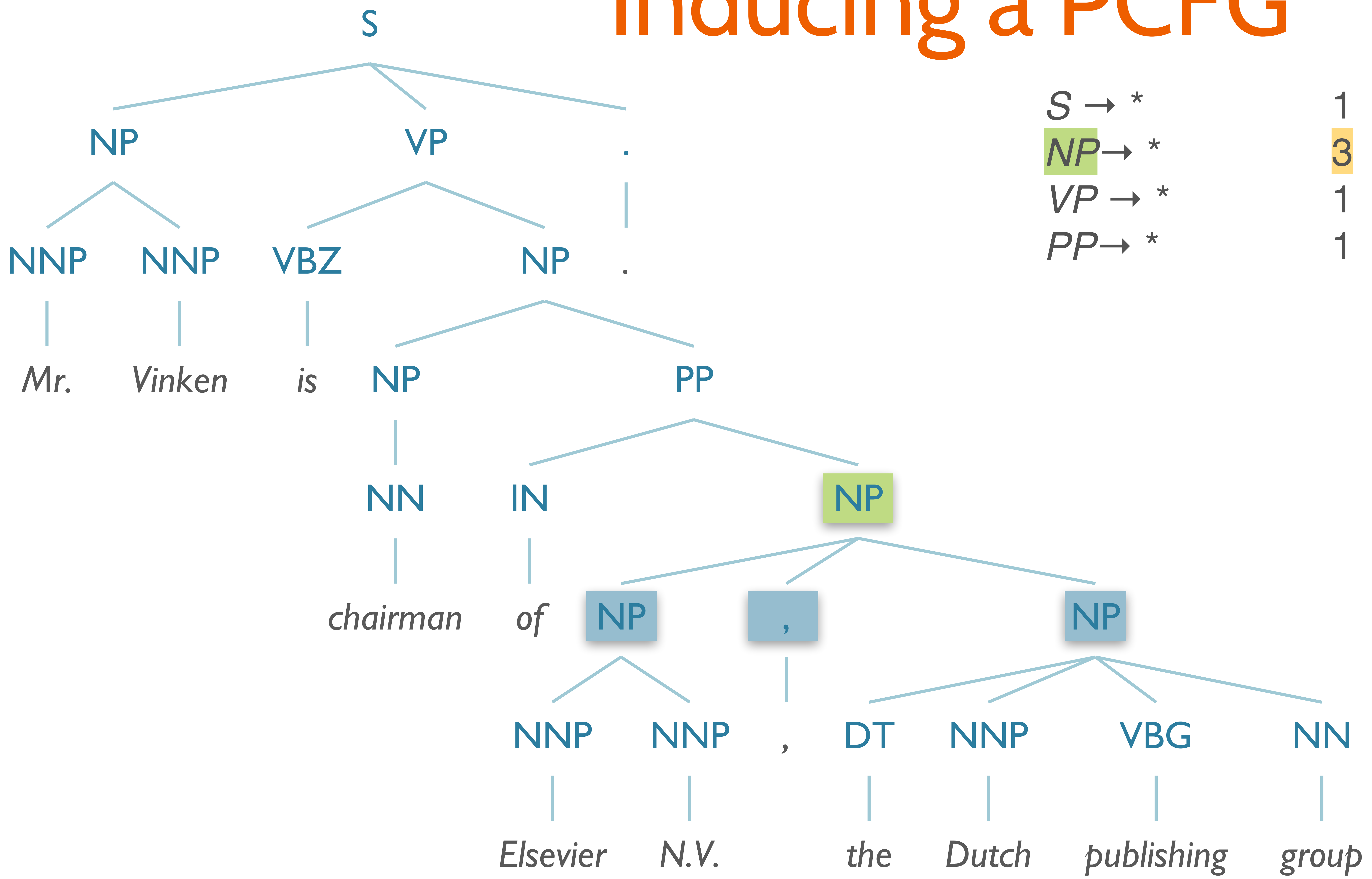
S → *	1	*S* → *NP VP .*	1
NP→ *	2	*NP* → *NNP NNP*	1
VP → *	1	*VP* → *VBZ NP*	1
		NP→ *NP PP*	1

Inducing a PCFG



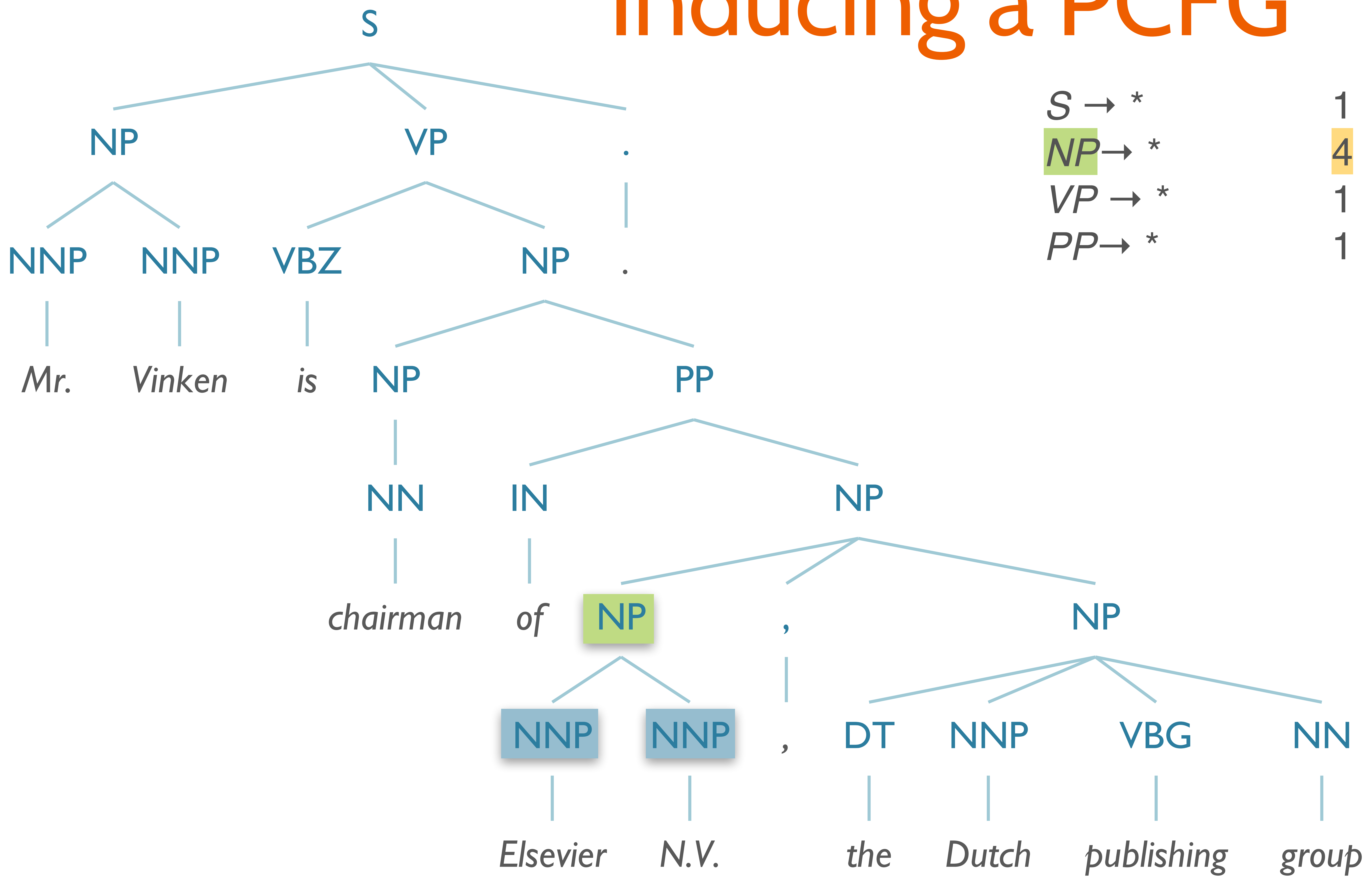
S → *	1	*S* → *NP VP .*	1
NP→ *	2	*NP* → *NNP NNP*	1
VP → *	1	*VP* → *VBZ NP*	1
PP→ *	1	*NP*→ *NP PP*	1
		PP→ *IN NP*	1

Inducing a PCFG



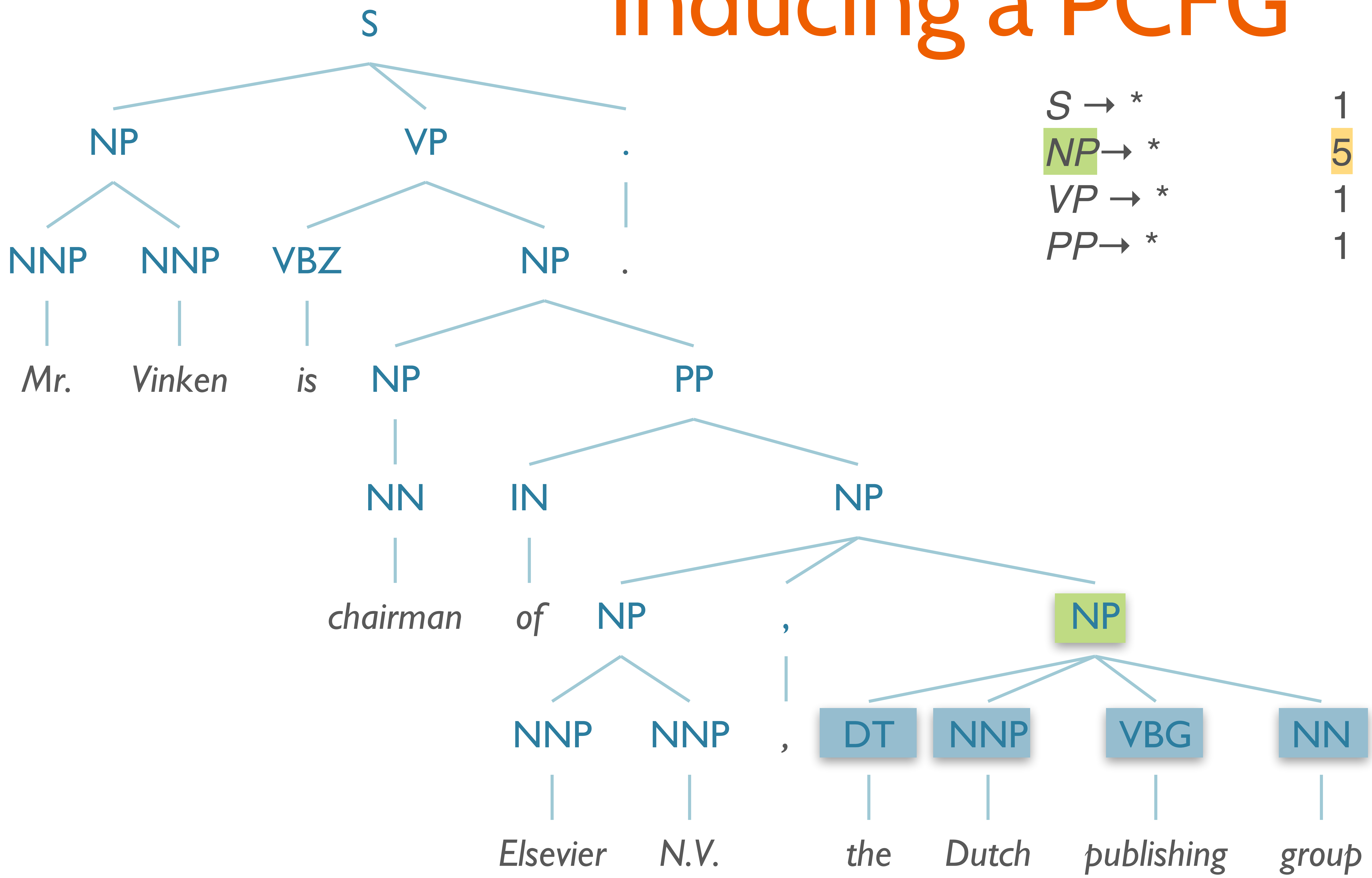
S → *	1	S → NP VP .	1
NP→ *	3	NP → NNP NNP	1
VP → *	1	VP → VBZ NP	1
PP→ *	1	NP→ NP PP	1
		PP→ IN NP	1
		NP→ NP , NP	1

Inducing a PCFG



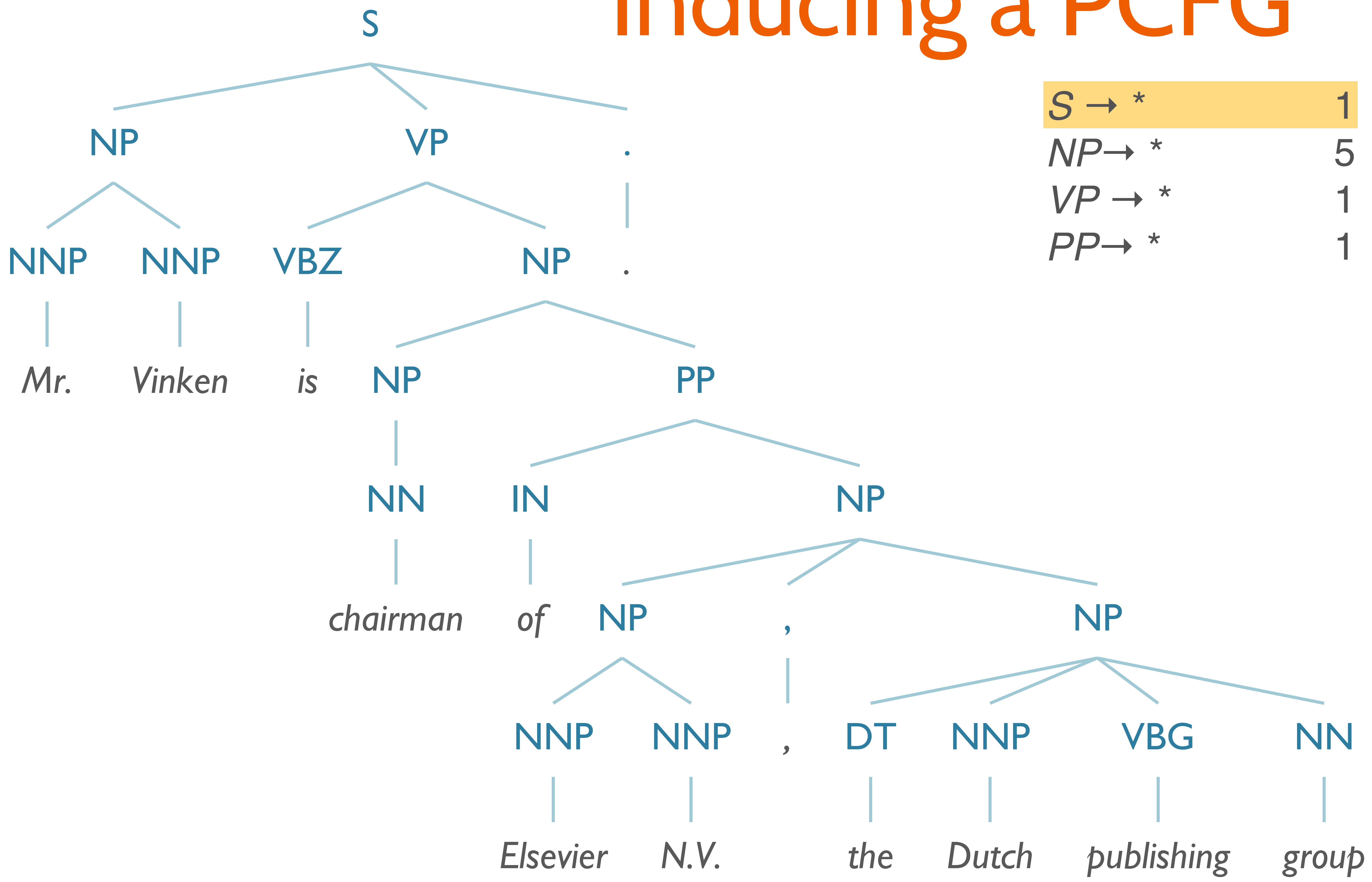
S → *	1	*S* → *NP VP .*	1
NP→ *	4	*NP*→ *NNP NNP*	2
VP → *	1	*VP* → *VBZ NP*	1
PP→ *	1	*NP*→ *NP PP*	1
		PP→ *IN NP*	1
		NP→ *NP , NP*	1

Inducing a PCFG



S → *	1	*S → NP VP .*	1
NP→ *	5	*NP→ NNP NNP*	2
VP → *	1	*VP → VBZ NP*	1
PP→ *	1	*NP→ NP PP*	1
		PP→ IN NP	1
		NP→ NP , NP	1
		NP→ DT NNP VBG NN	1

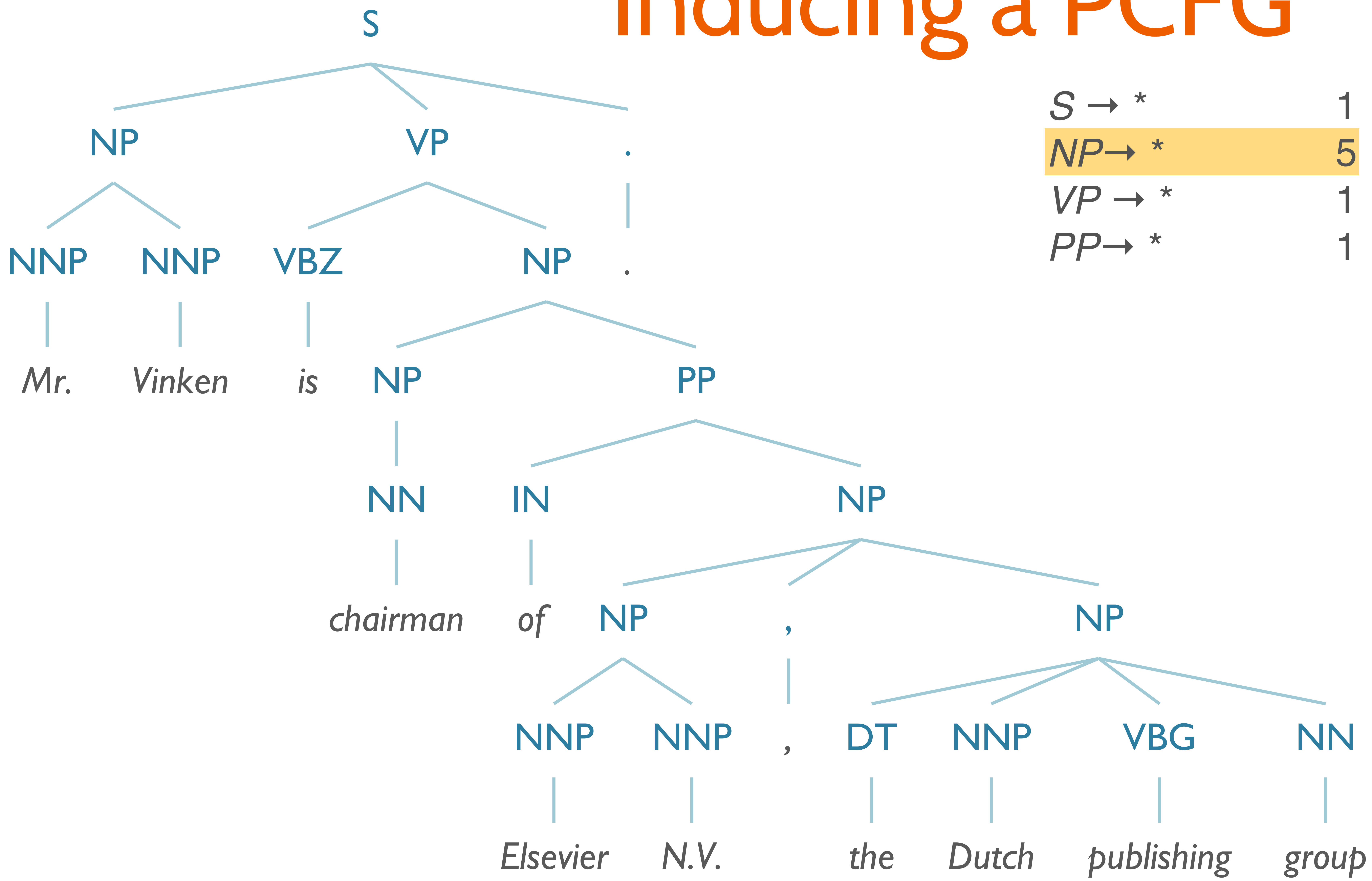
Inducing a PCFG



*S → **	1
*NP→ **	5
*VP → **	1
*PP→ **	1

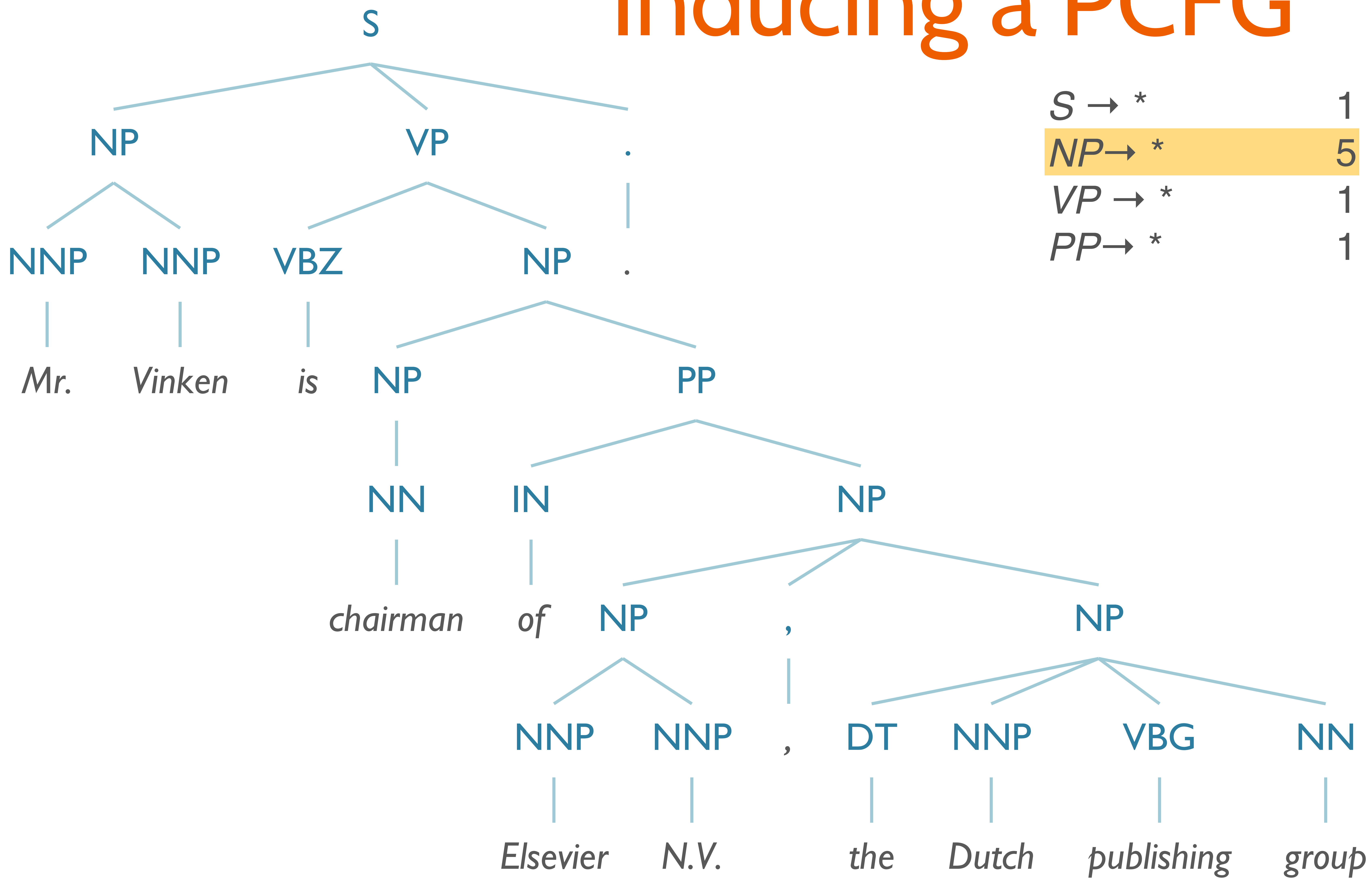
S → NP VP .	1
NP→ NNP NNP	2
VP → VBZ NP	1
NP→ NP PP	1
PP→ IN NP	1
NP→ NP , NP	1
NP→ DT NNP VBG NN	1

Inducing a PCFG



S → *	1	*S* → *NP VP .*	1
NP→ *	5	*NP*→ *NNP NNP*	2/5
VP → *	1	*VP* → *VBZ NP*	1
PP→ *	1	*NP*→ *NP PP*	1/5
		PP→ *IN NP*	1
		NP→ *NP , NP*	1/5
		NP→ *DT NNP VBG NN*	1/5

Inducing a PCFG



S → *	1	*S → NP VP .*	1
NP→ *	5	*NP→ NNP NNP*	0.4
VP → *	1	*VP → VBZ NP*	1
PP→ *	1	*NP→ NP PP*	0.2
		PP→ IN NP	1
		NP→ NP , NP	0.2
		NP→ DT NNP VBG NN	0.2

Problems with PCFGs

Problems with PCFGs

- Independence Assumption
  - Assume that rule probabilities are independent

Problems with PCFGs

- Independence Assumption
  - Assume that rule probabilities are independent

- Lack of Lexical Conditioning
  - Lexical items should influence the choice of analysis

Issues with PCFGs: Independence Assumption

- *Context Free ⇒ Independence Assumption*
  - Rule expansion is context-independent
  - Allows us to multiply probabilities

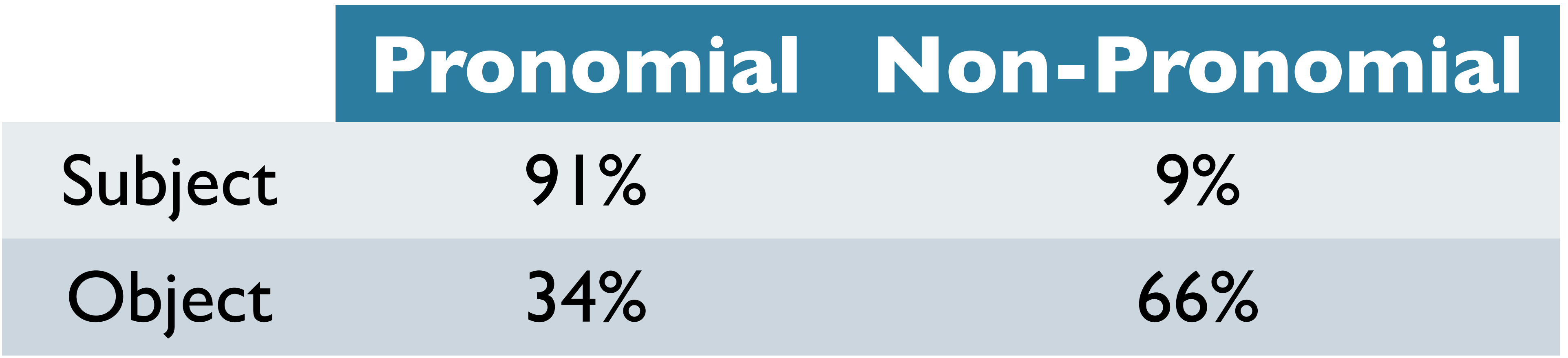
Issues with PCFGs: Independence Assumption

- *Context Free ⇒ Independence Assumption*
  - Rule expansion is context-independent
  - Allows us to multiply probabilities
- If we have two rules:
  - *NP* → *DT NN* [0.28]
  - *NP* → *PRP* [0.25]

Issues with PCFGs: Independence Assumption

- *Context Free ⇒ Independence Assumption*
  - Rule expansion is context-independent
  - Allows us to multiply probabilities
- If we have two rules:
  - *NP* → *DT NN* [0.28]
  - *NP* → *PRP* [0.25]

Semantic Role of **NPs** in Switchboard Corpus



	Pronomial	**Non-Pronomial**
Subject	91%	9%
Object	34%	66%

Issues with PCFGs: Independence Assumption

- *Context Free ⇒ Independence Assumption*
  - Rule expansion is context-independent
  - Allows us to multiply probabilities
- If we have two rules:
  - *NP* → *DT NN* [0.28]
  - *NP* → *PRP* [0.25]
- What does this new data tell us?

Semantic Role of **NPs** in Switchboard Corpus

	Pronomial	Non-Pronomial
Subject	91%	9%
Object	34%	66%

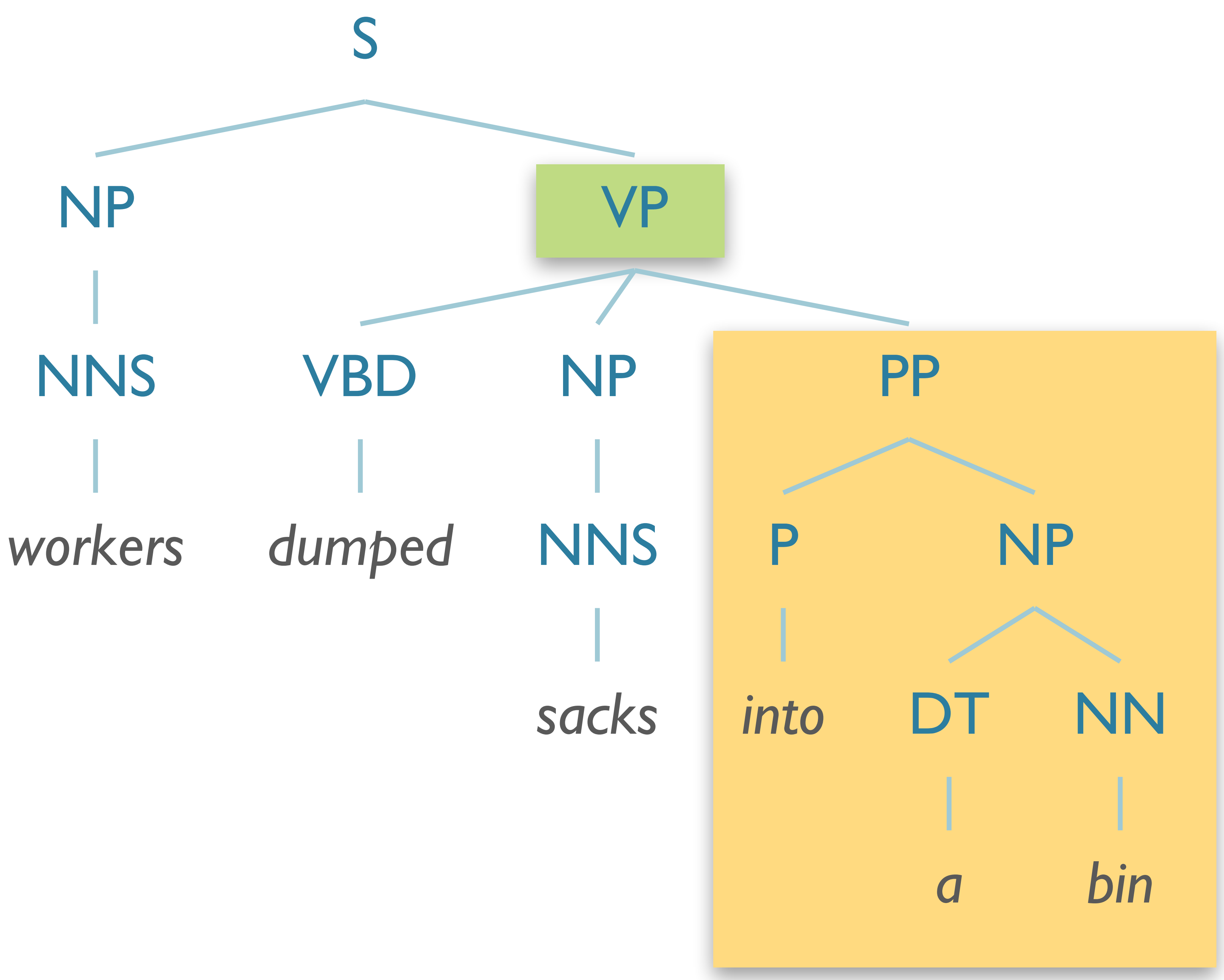
Issues with PCFGs: Independence Assumption

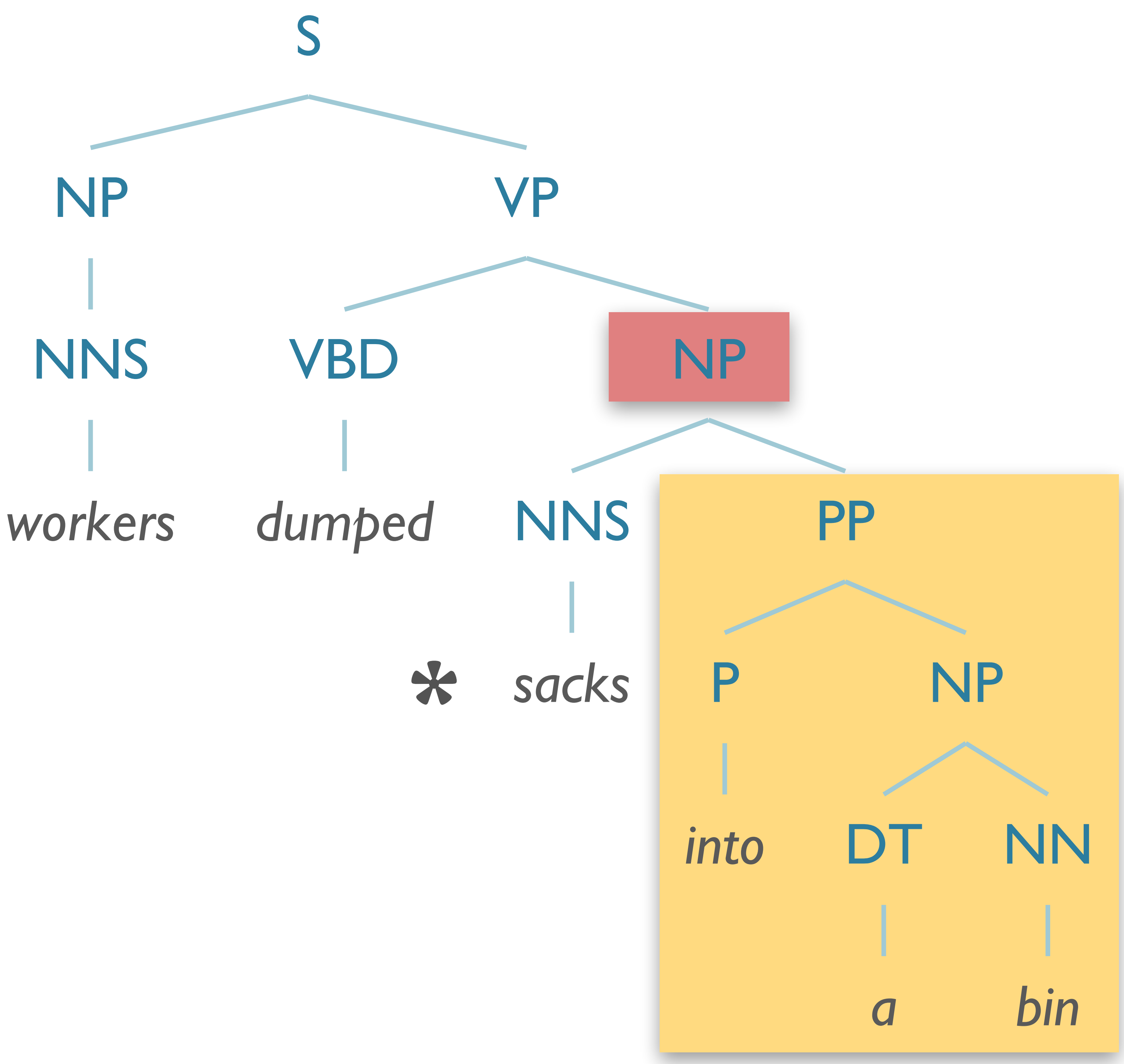
- *Context Free ⇒ Independence Assumption*
  - Rule expansion is context-independent
  - Allows us to multiply probabilities
- If we have two rules:
  - *NP* → *DT NN* [0.28]
  - *NP* → *PRP* [0.25]
- What does this new data tell us?
  - *NP* → *DT NN* [0.09 **if** $NP_{\Theta=subject}$ **else** 0.66]
  - *NP* → *PRP* [0.91 **if** $NP_{\Theta=subject}$ **else** 0.34]

Semantic Role of **NPs** in Switchboard Corpus

	Pronomial	**Non-Pronomial**
Subject	91%	9%
Object	34%	66%

Issues with PCFGs: Lexical Conditioning

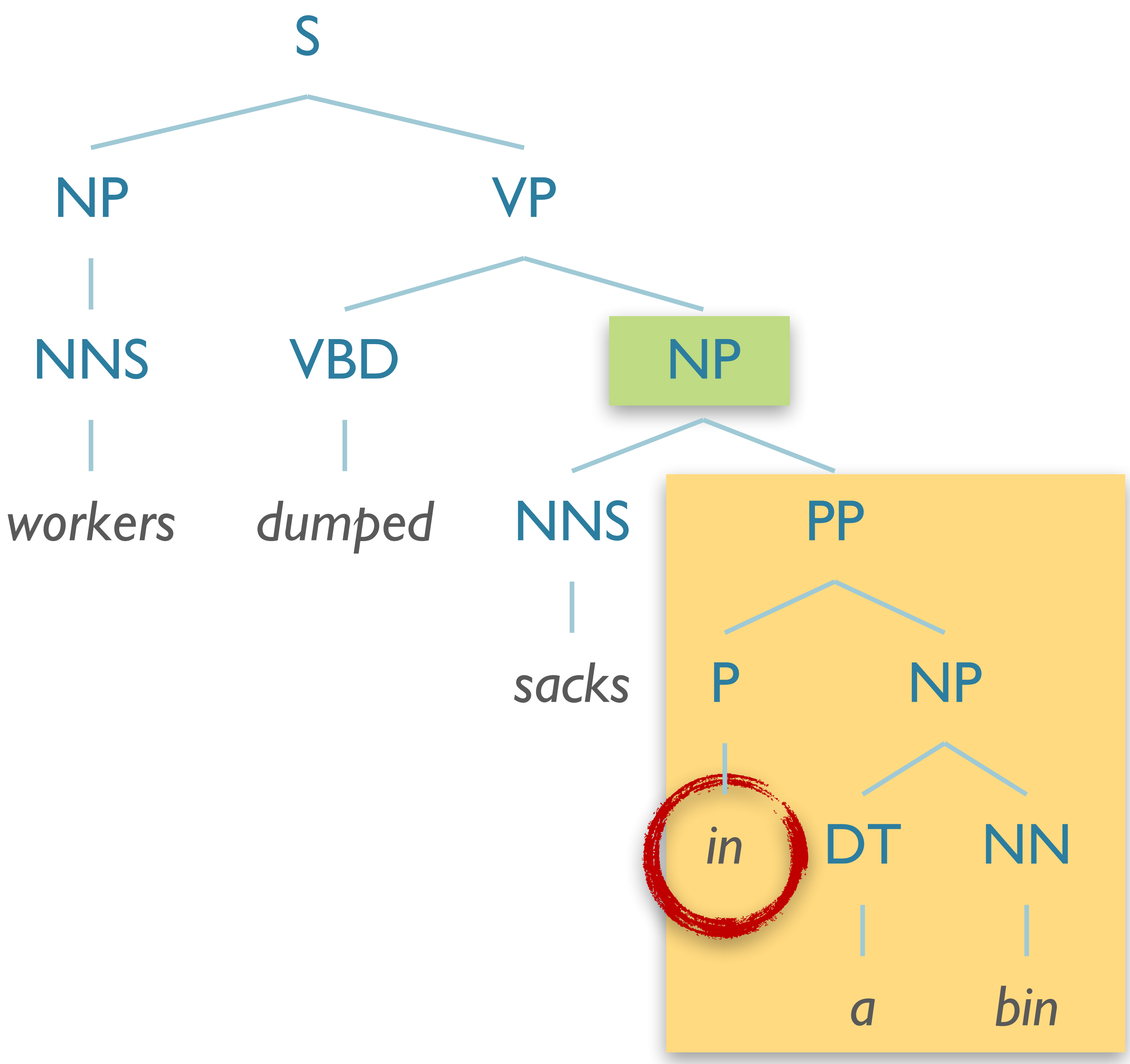


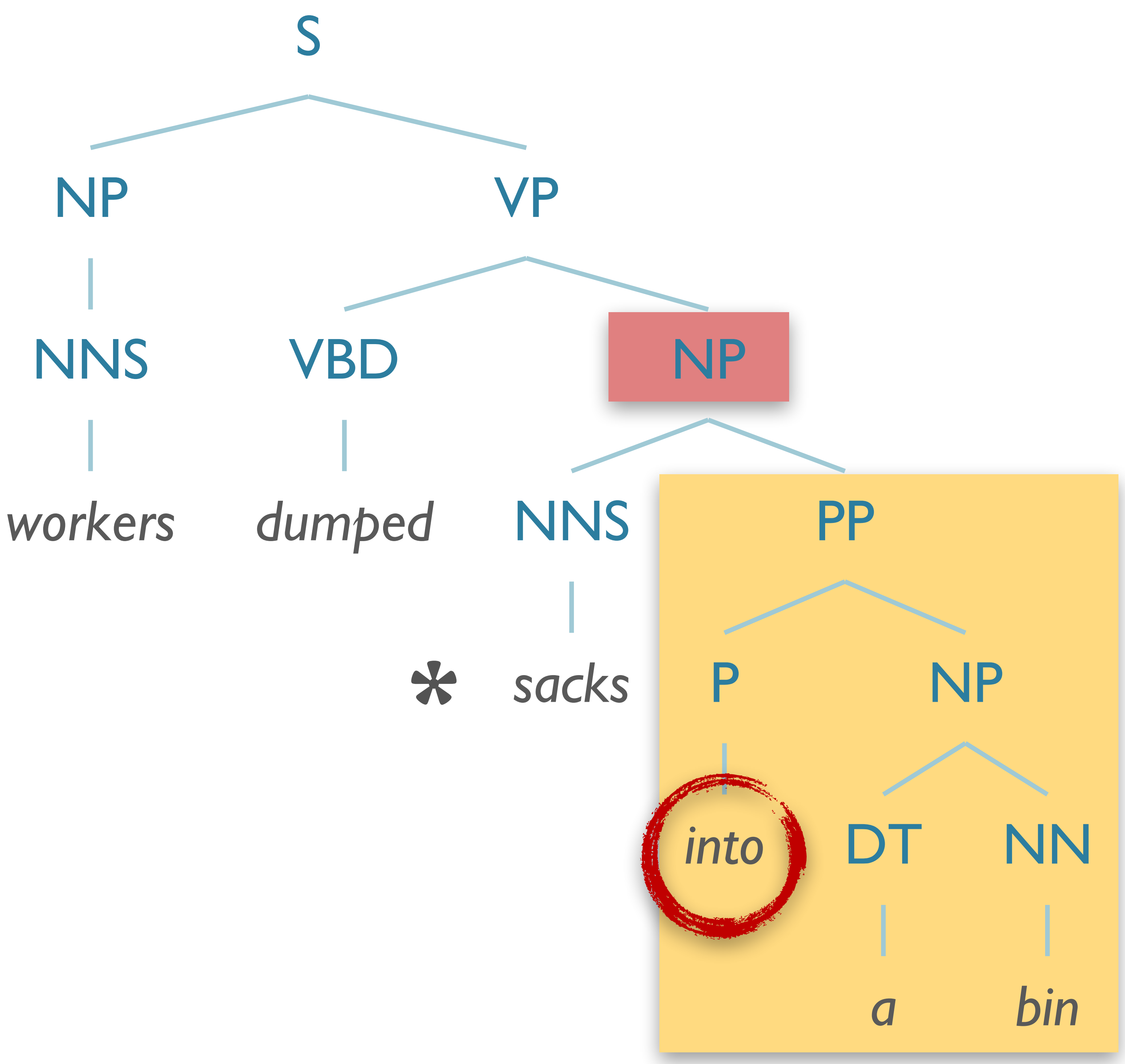


("into a bin" = location of sacks after dumping)
OK!

("into a bin" = *the sacks which were located ***in PP***)
not OK

Issues with PCFGs: Lexical Conditioning





(“***in*** a bin” = location of sacks ***before*** dumping)
OK!

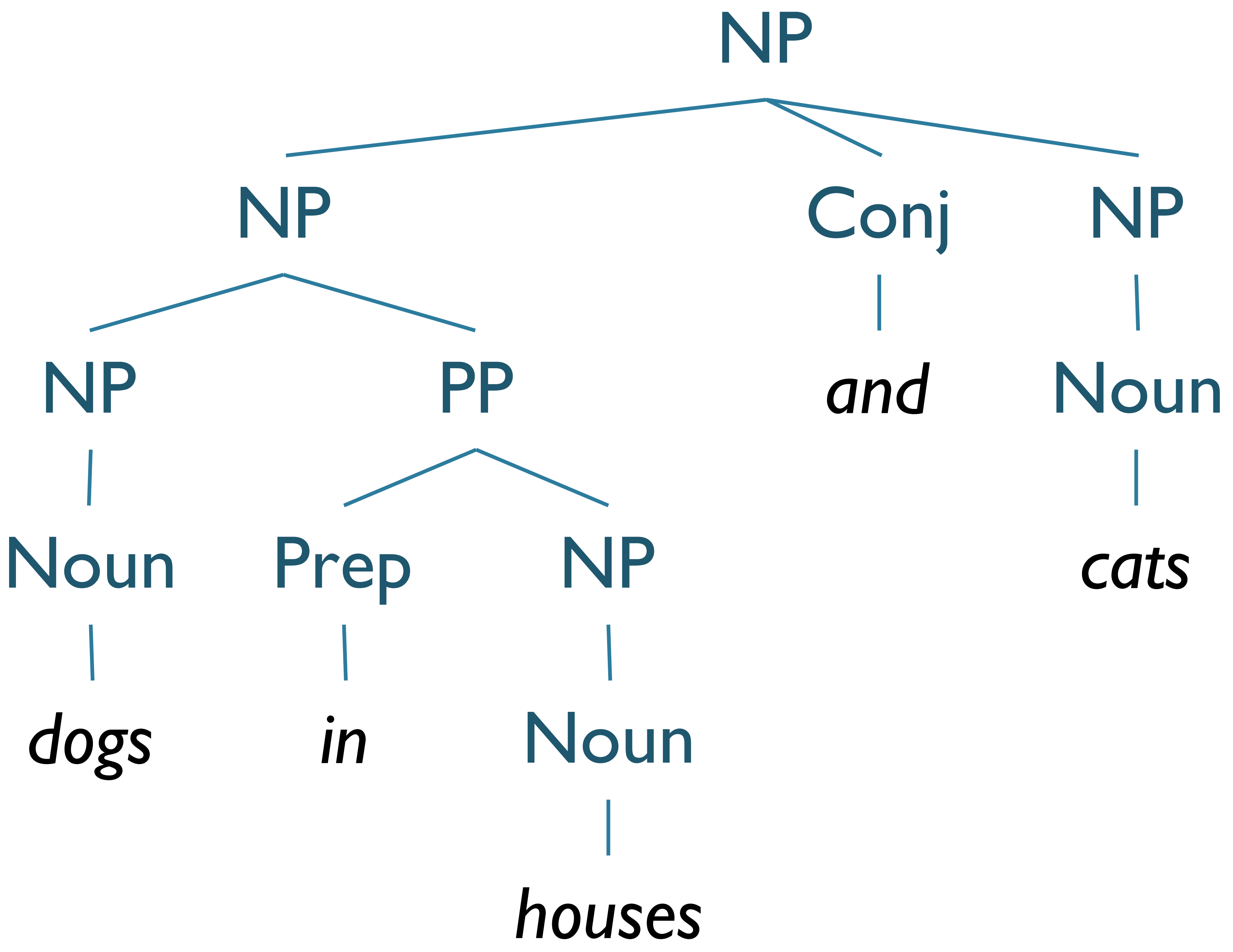
(“***into*** a bin” = *the sacks which were located ***in PP***)
not OK

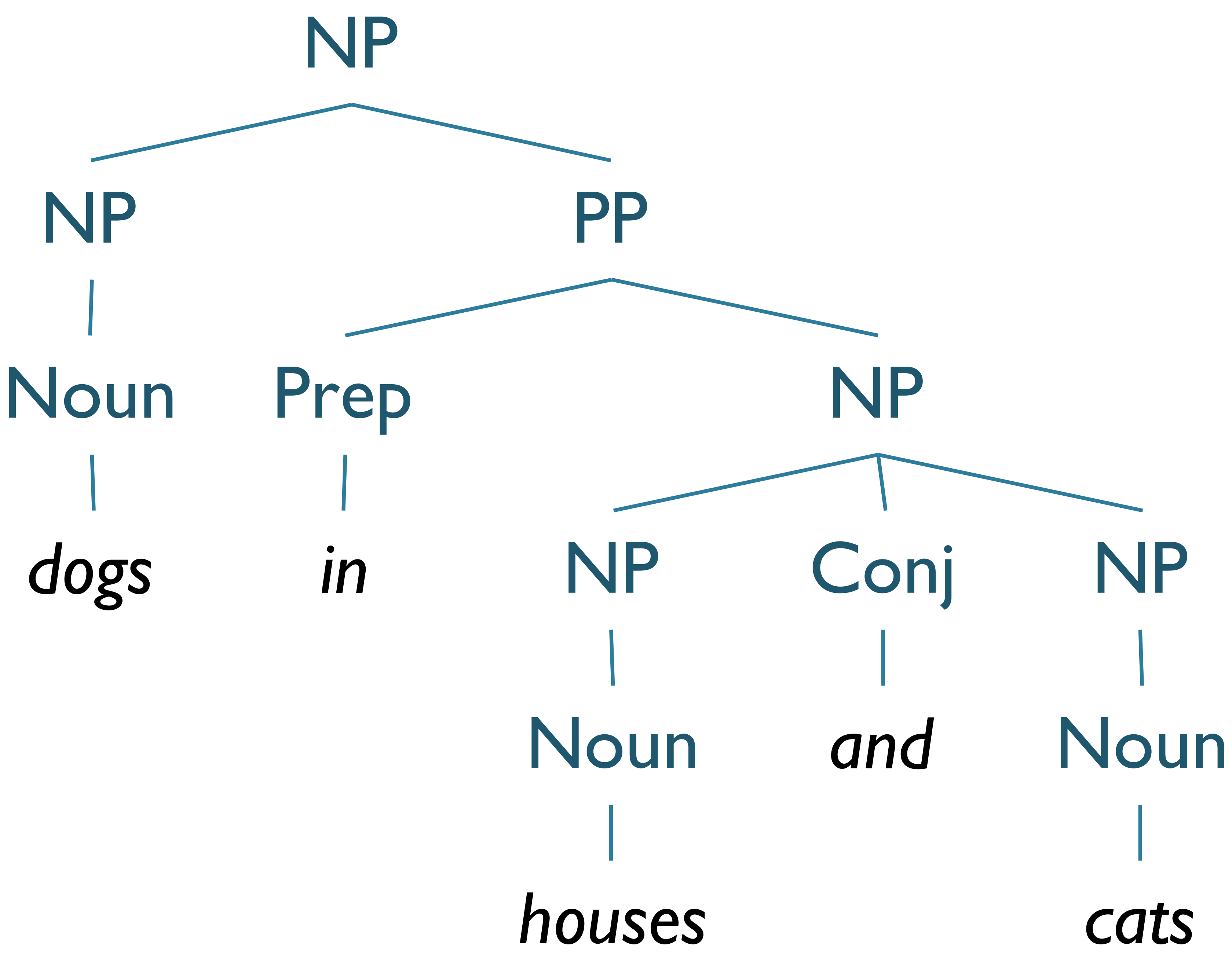
Issues with PCFGs: Lexical Conditioning

- *workers dumped sacks into a bin*
  - ***into*** should **prefer** modifying ***dumped***
  - ***into*** should **disprefer** modifying ***sacks***

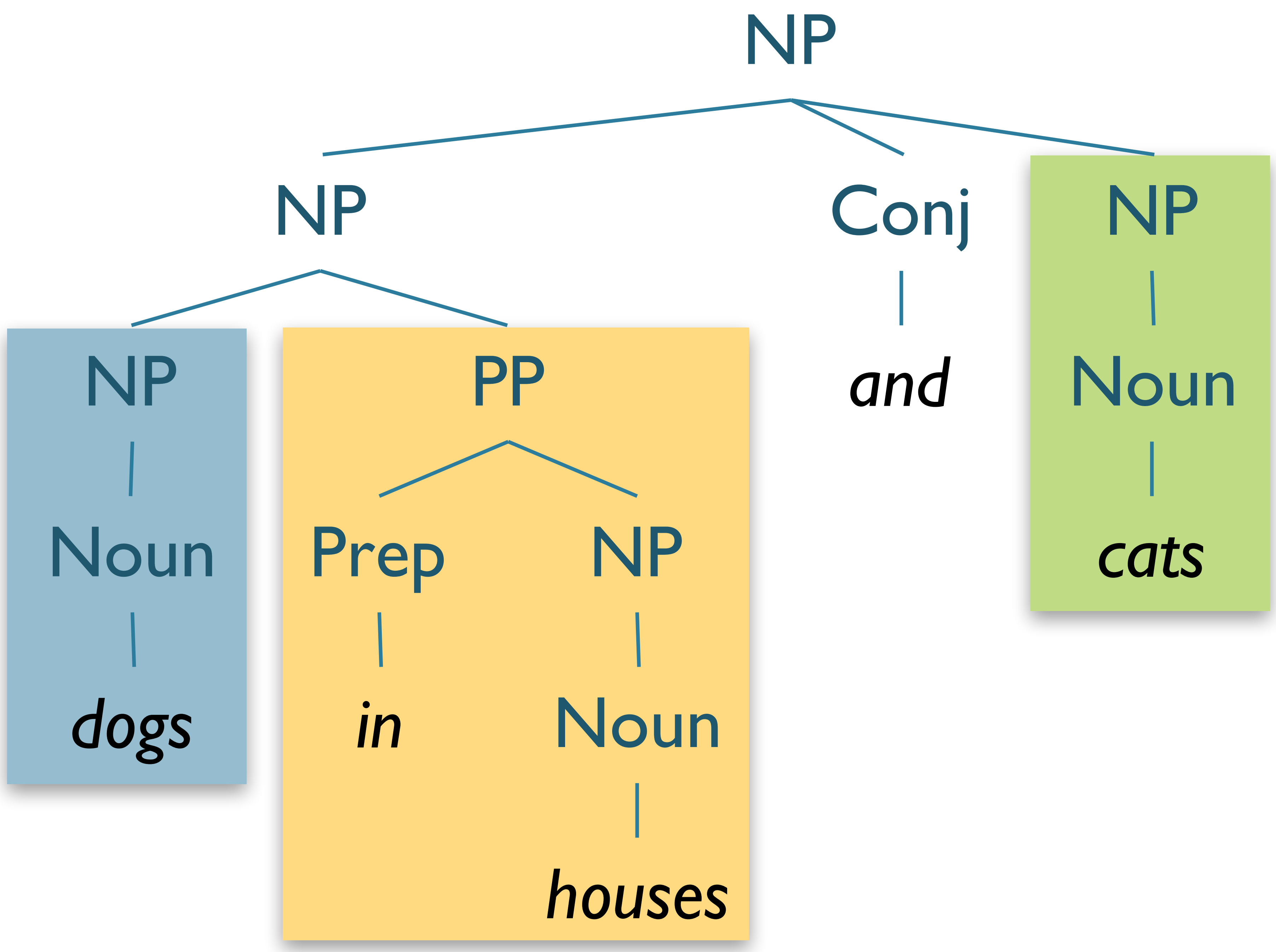
- *workers dumped sacks in a bin* (cf. also *fisherman caught tons of herring*)
  - ***in*** should **prefer** modifying ***sacks***
  - ***in*** should **disprefer** modifying ***dumped***

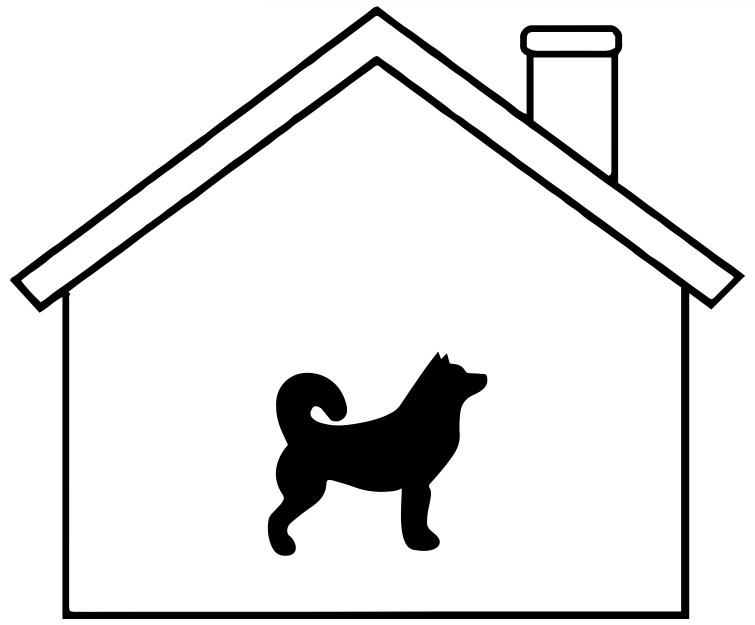
Issues with PCFGs: Coordination Ambiguity

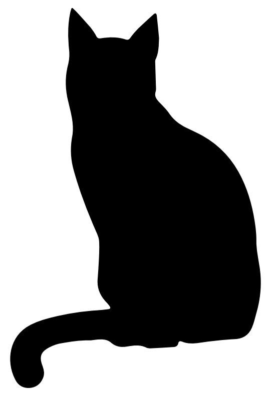


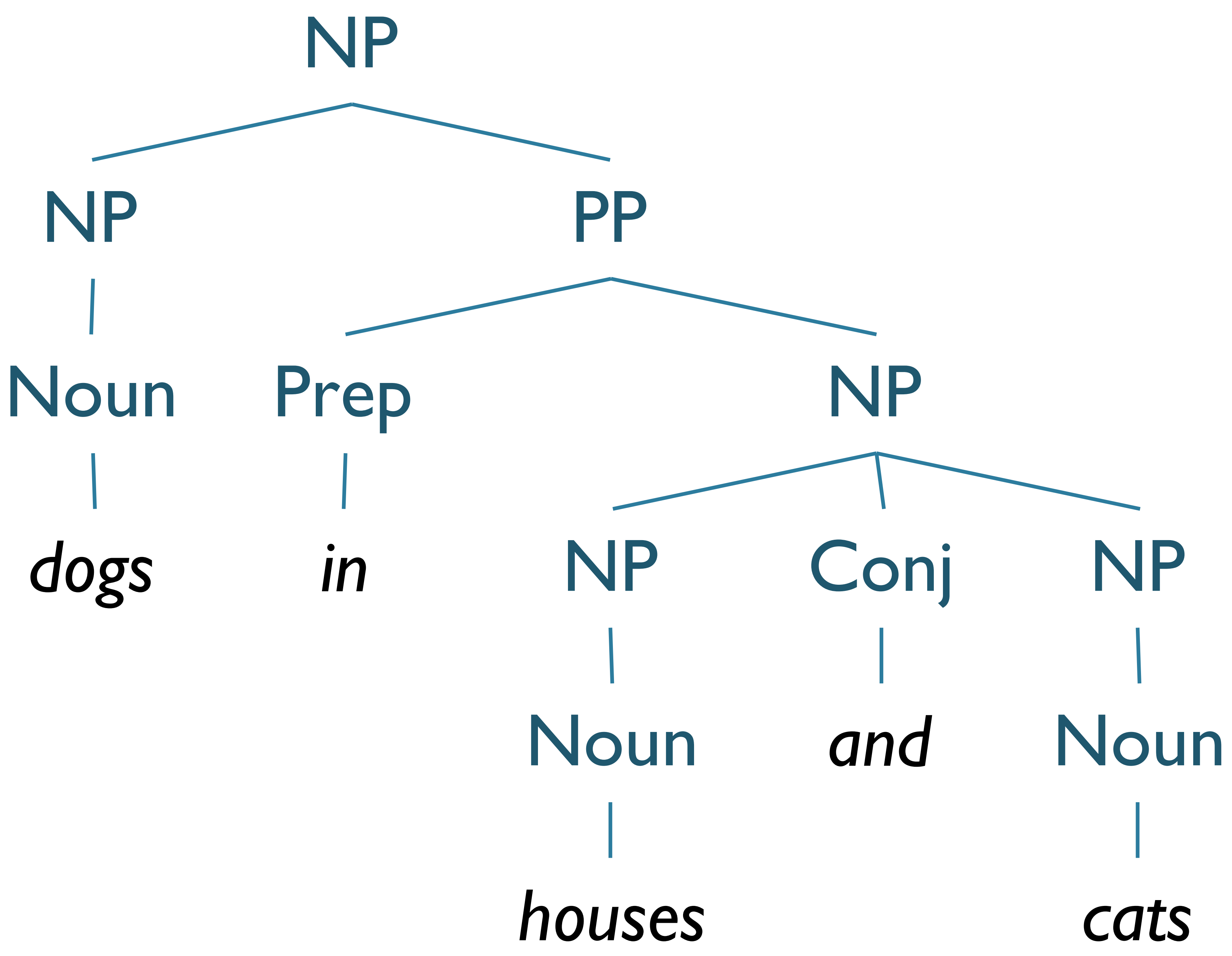


Issues with PCFGs: Coordination Ambiguity

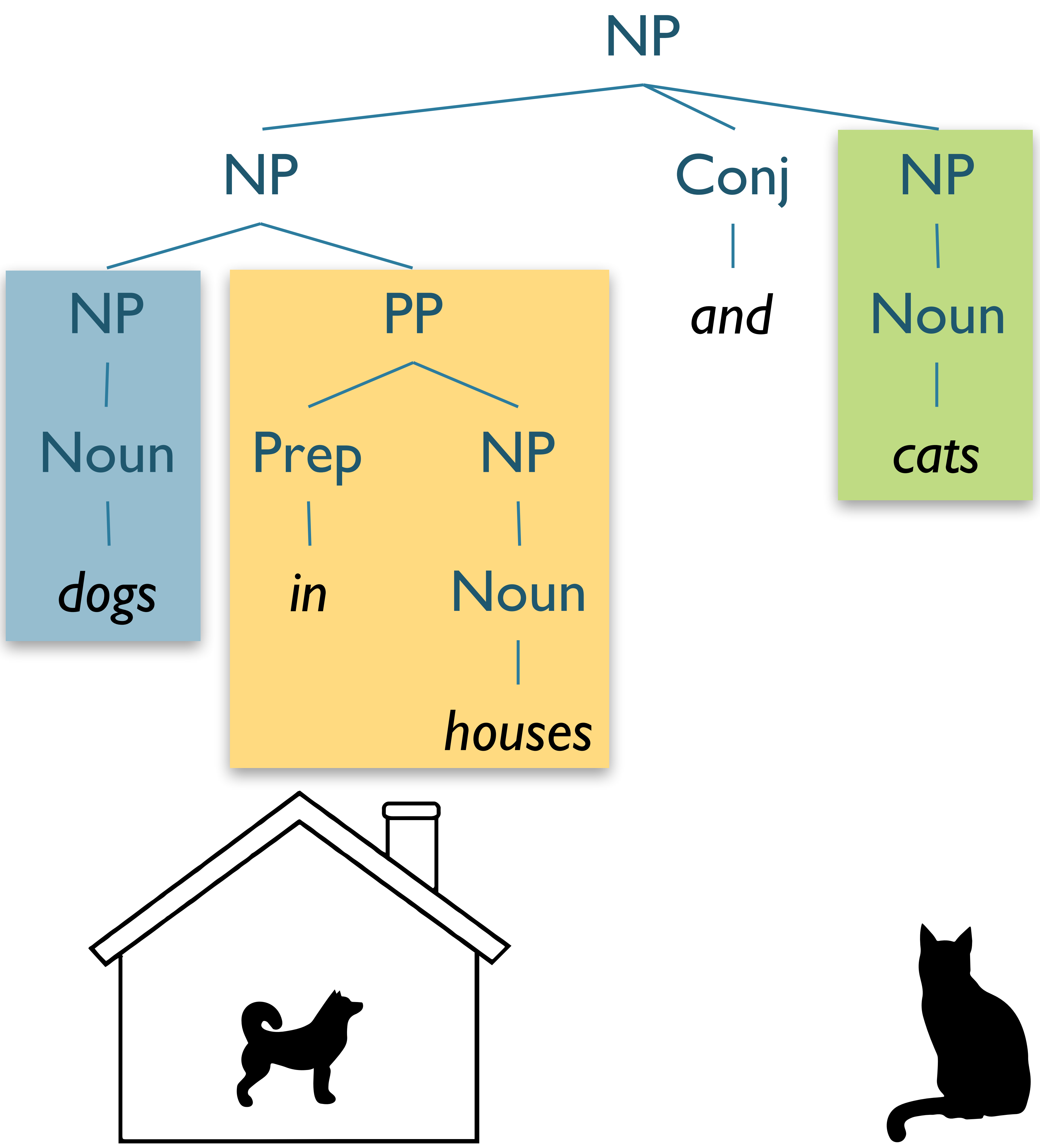


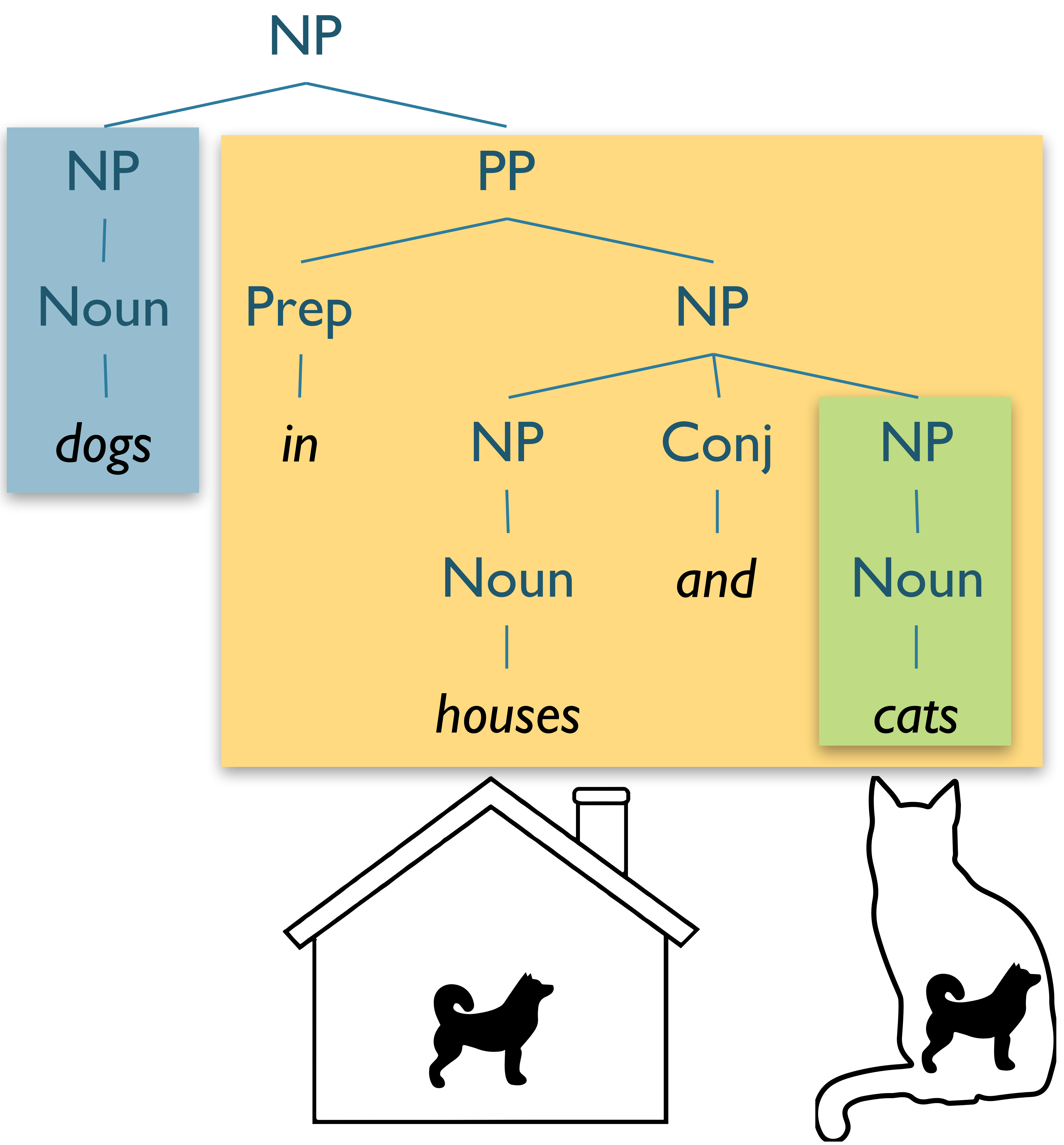




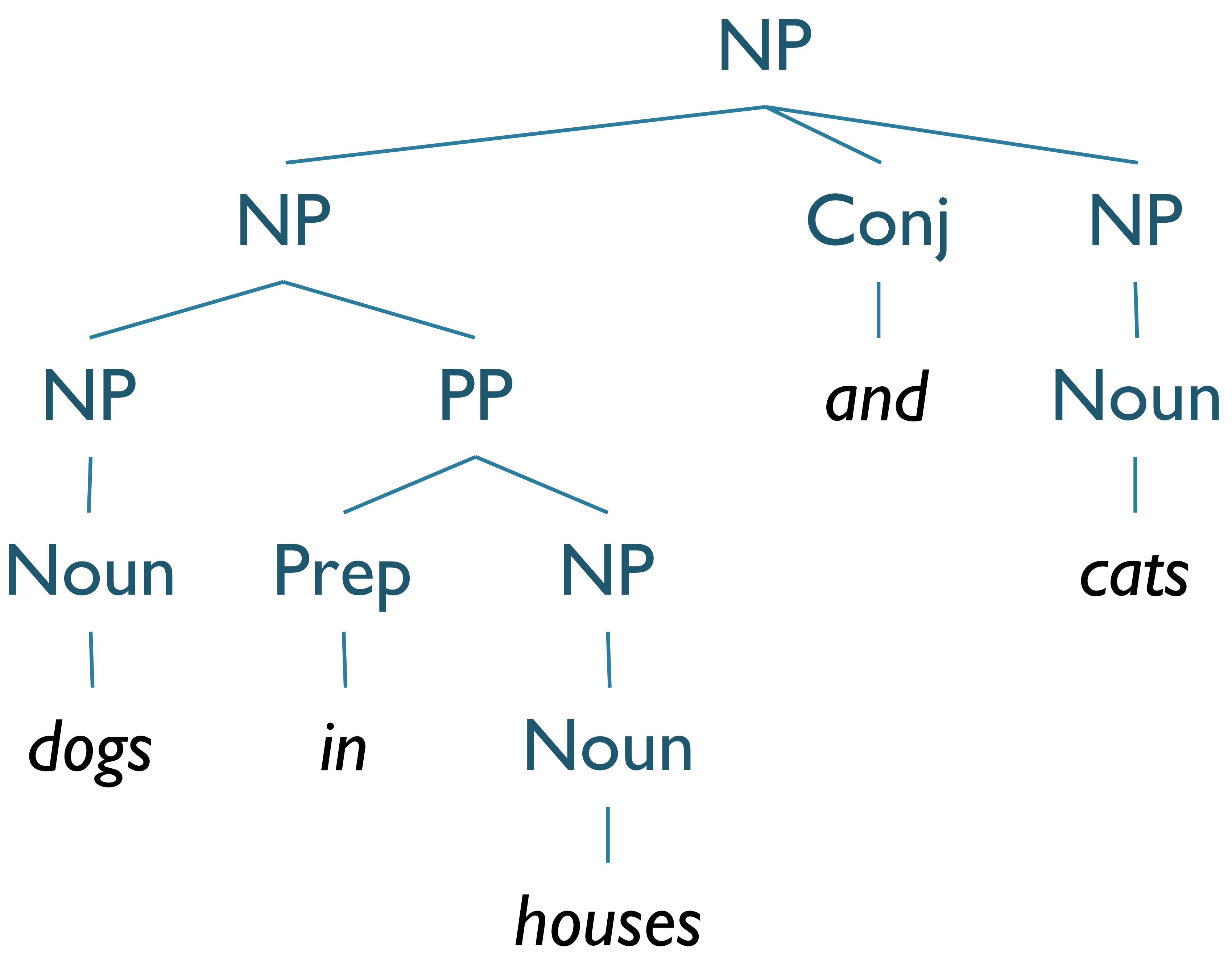


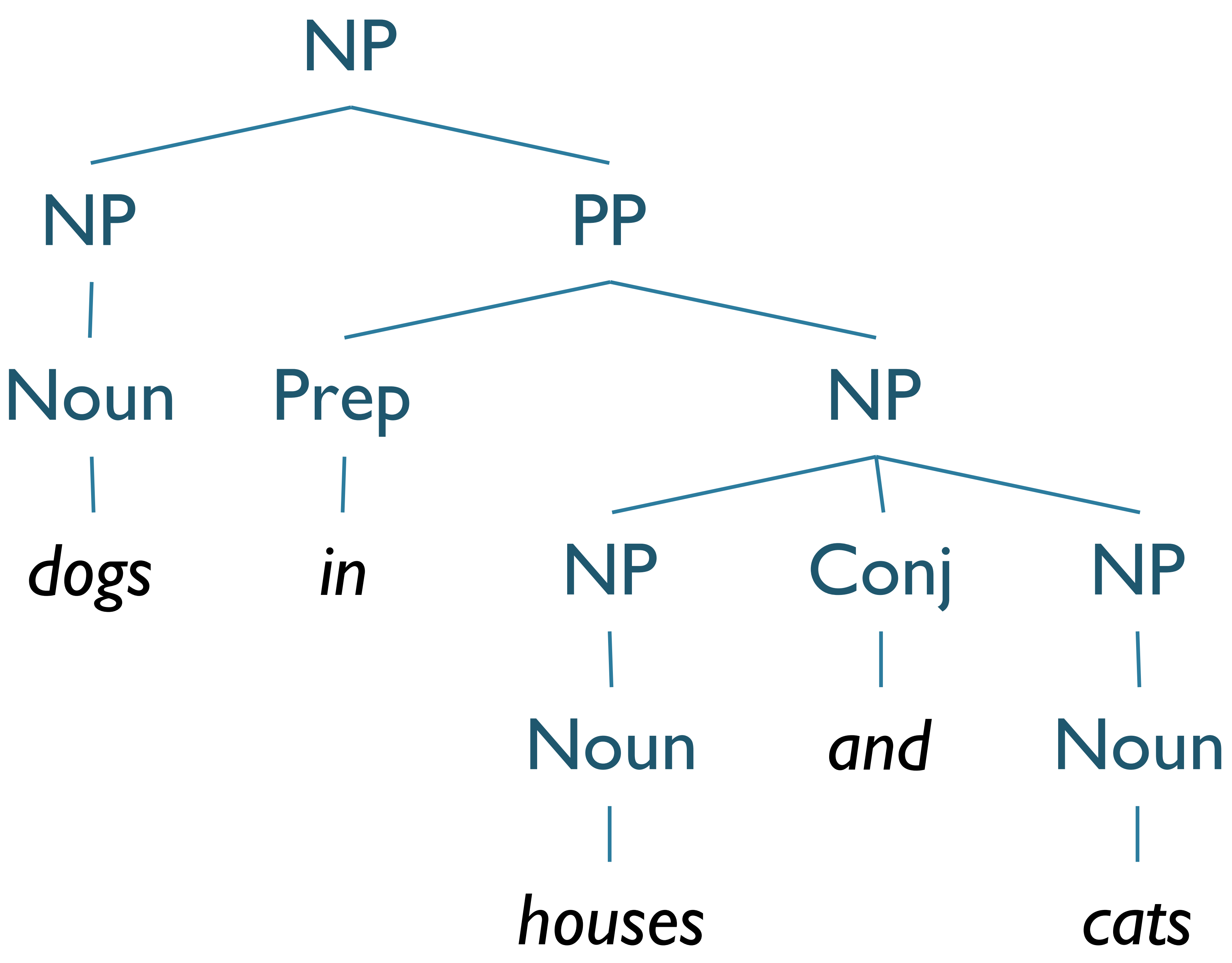
Issues with PCFGs: Coordination Ambiguity





Issues with PCFGs: Coordination Ambiguity



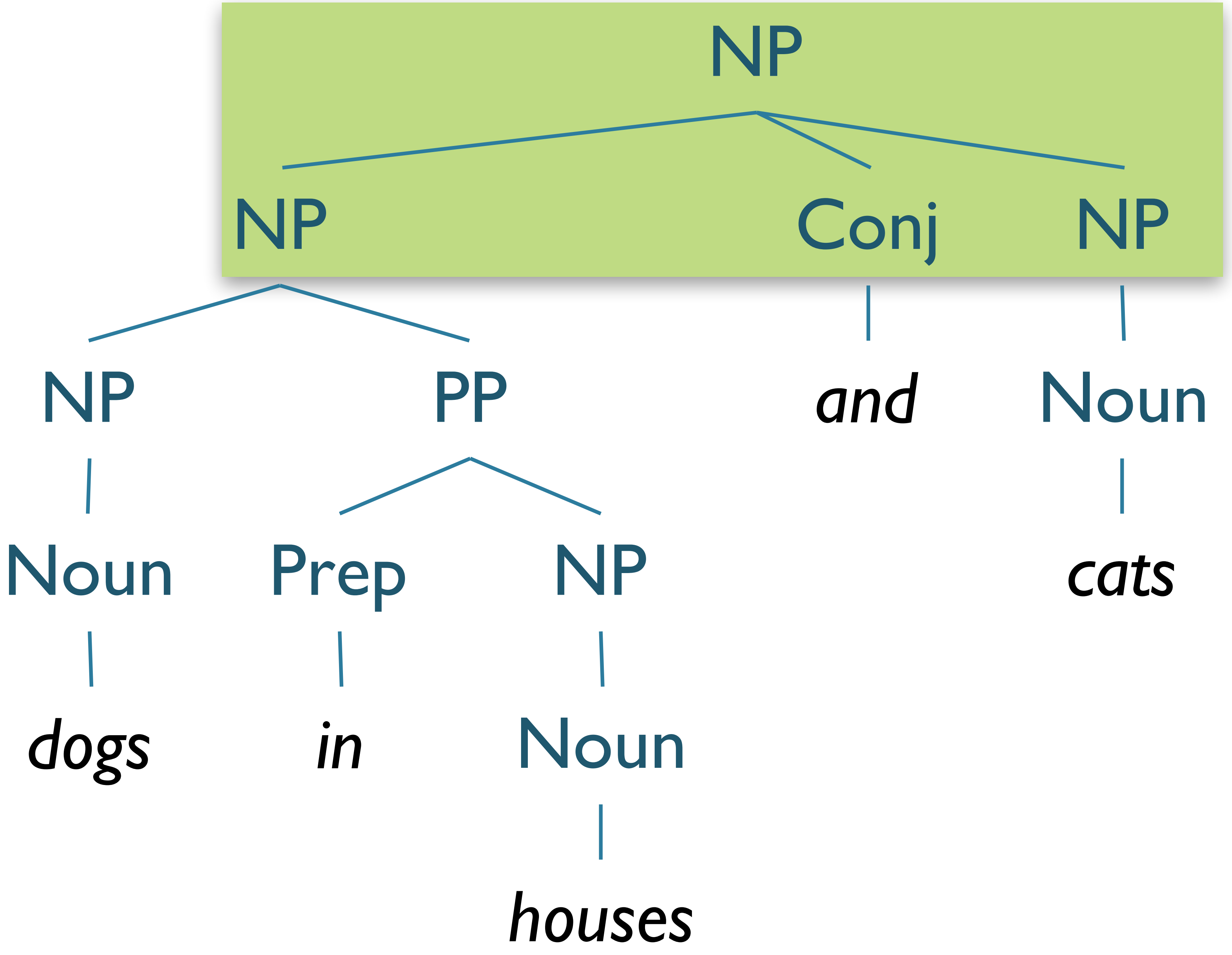


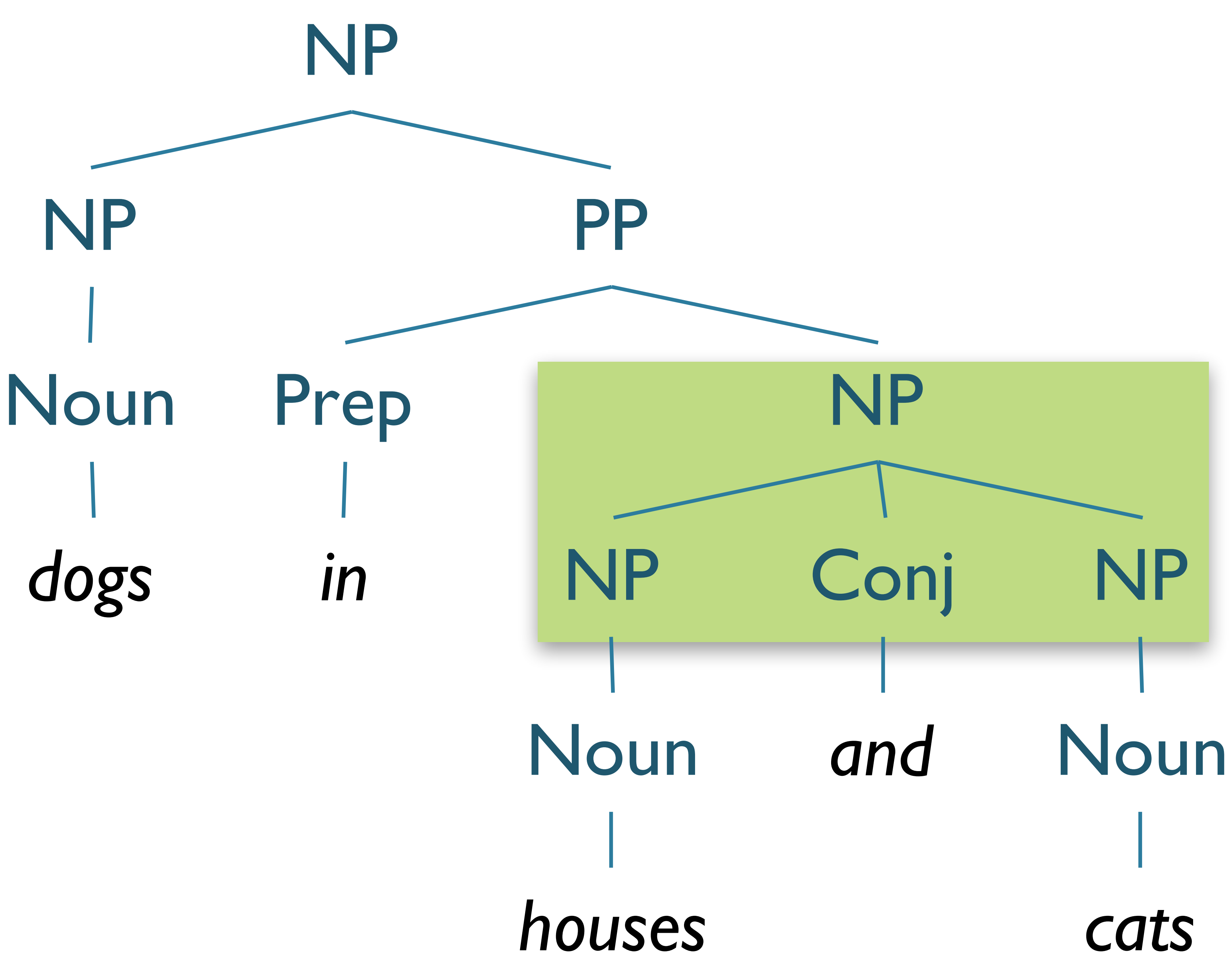
NP → NP Conj NP
NP → NP PP
Noun → "dogs"
PP → Prep NP
Prep → "in"
NP → Noun
Noun → "houses"
Conj → "and"
NP → Noun
Noun → "cats"

Same Rules!

NP → NP PP
Noun → "dogs"
PP → Prep NP
Prep → "in"
NP → NP Conj NP
NP → Noun
Noun → "houses"
Conj → "and"
NP → Noun
Noun → "cats"

Issues with PCFGs: Coordination Ambiguity



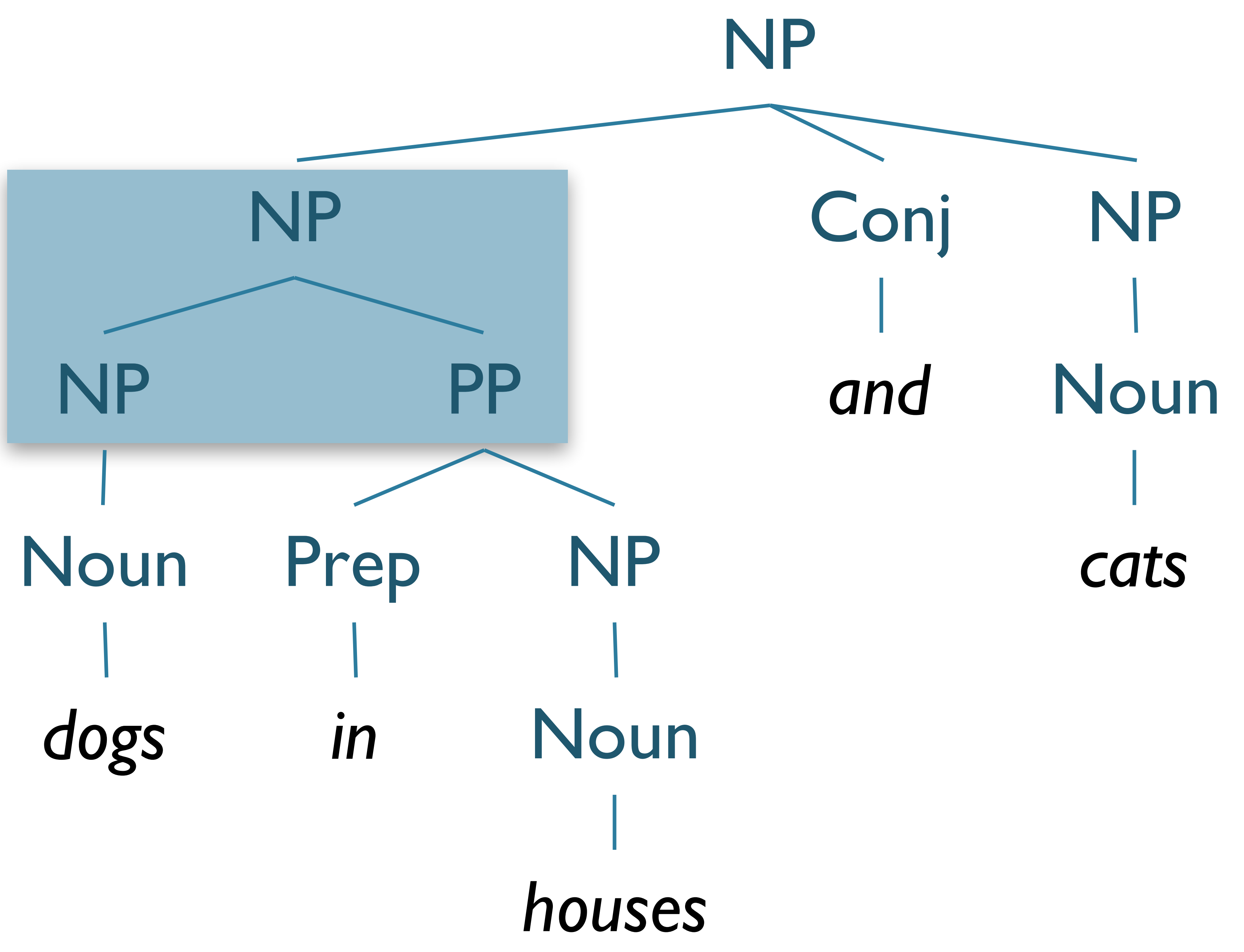


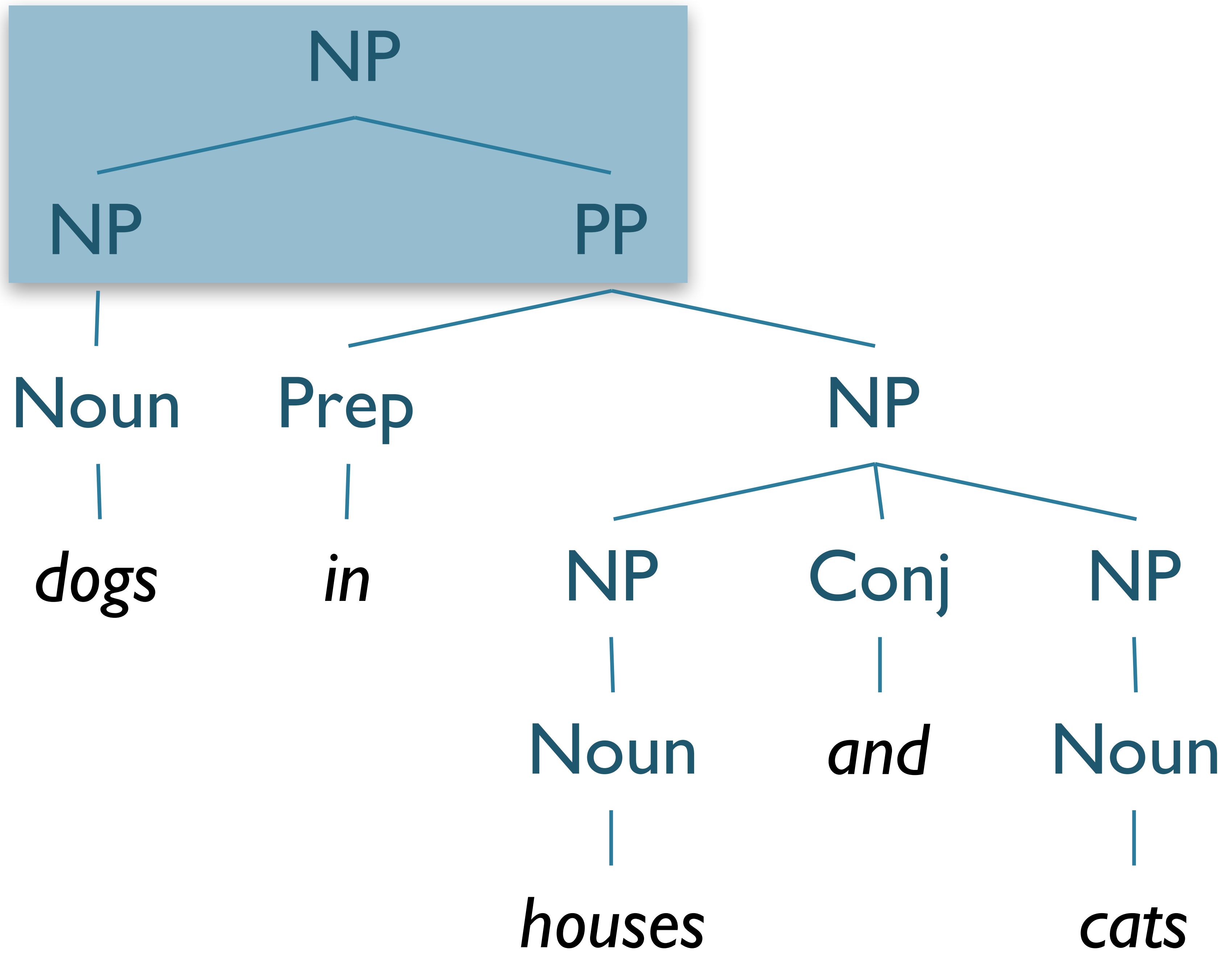
NP → NP Conj NP
NP → NP PP
Noun → "dogs"
PP → Prep NP
Prep → "in"
NP → Noun
Noun → "houses"
Conj → "and"
NP → Noun
Noun → "cats"

Same Rules!

NP → NP PP
Noun → "dogs"
PP → Prep NP
Prep → "in"
NP → NP Conj NP
NP → Noun
Noun → "houses"
Conj → "and"
NP → Noun
Noun → "cats"

Issues with PCFGs: Coordination Ambiguity





NP → NP Conj NP
NP → NP PP
Noun → "dogs"
PP → Prep NP
Prep → "in"
NP → Noun
Noun → "houses"
Conj → "and"
NP → Noun
Noun → "cats"

Same Rules!

NP → NP PP
Noun → "dogs"
PP → Prep NP
Prep → "in"
NP → NP Conj NP
NP → Noun
Noun → "houses"
Conj → "and"
NP → Noun
Noun → "cats"

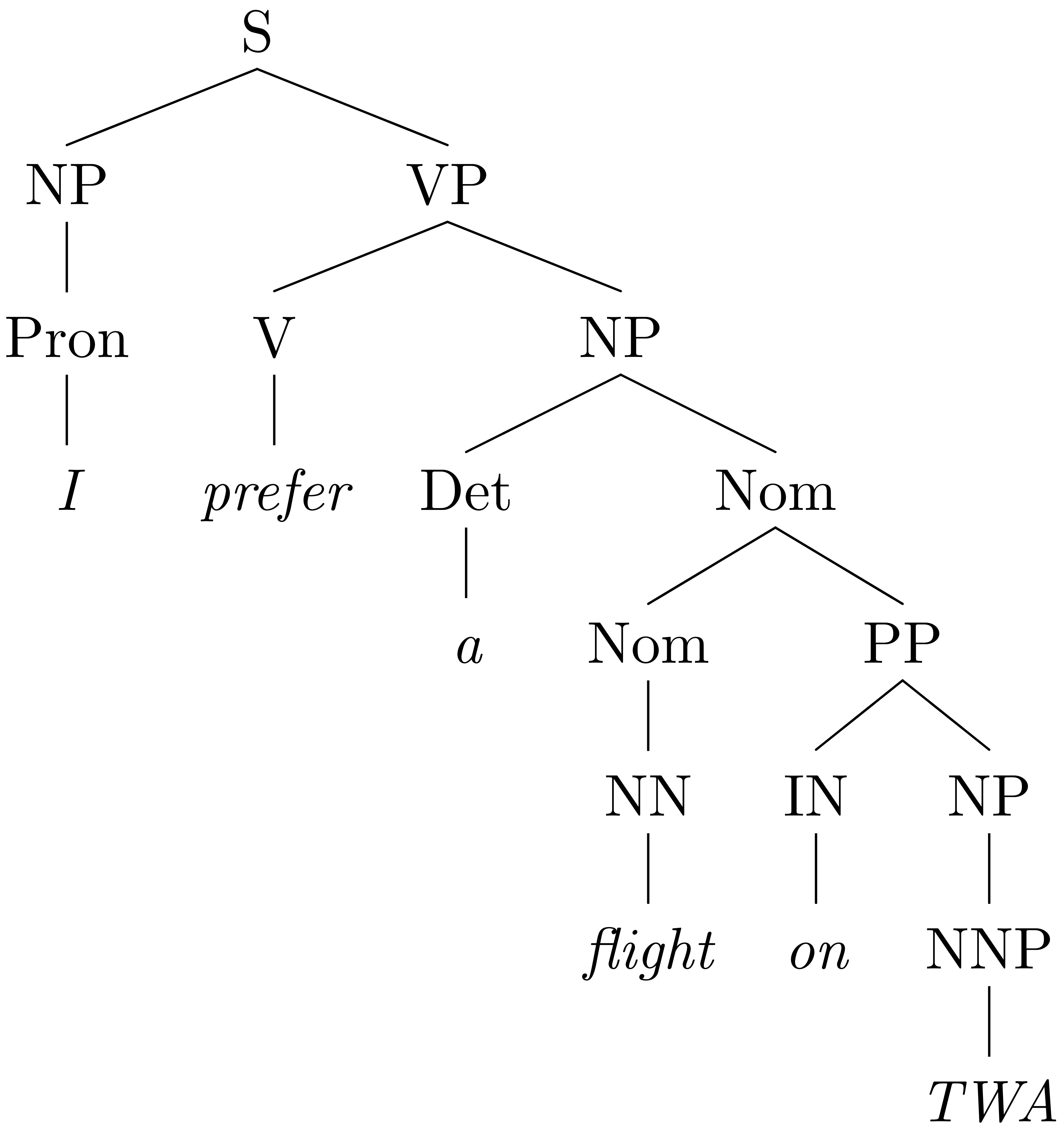
Improving PCFGs

Improving PCFGs

- **Parent Annotation**
- Lexicalization
- Reranking

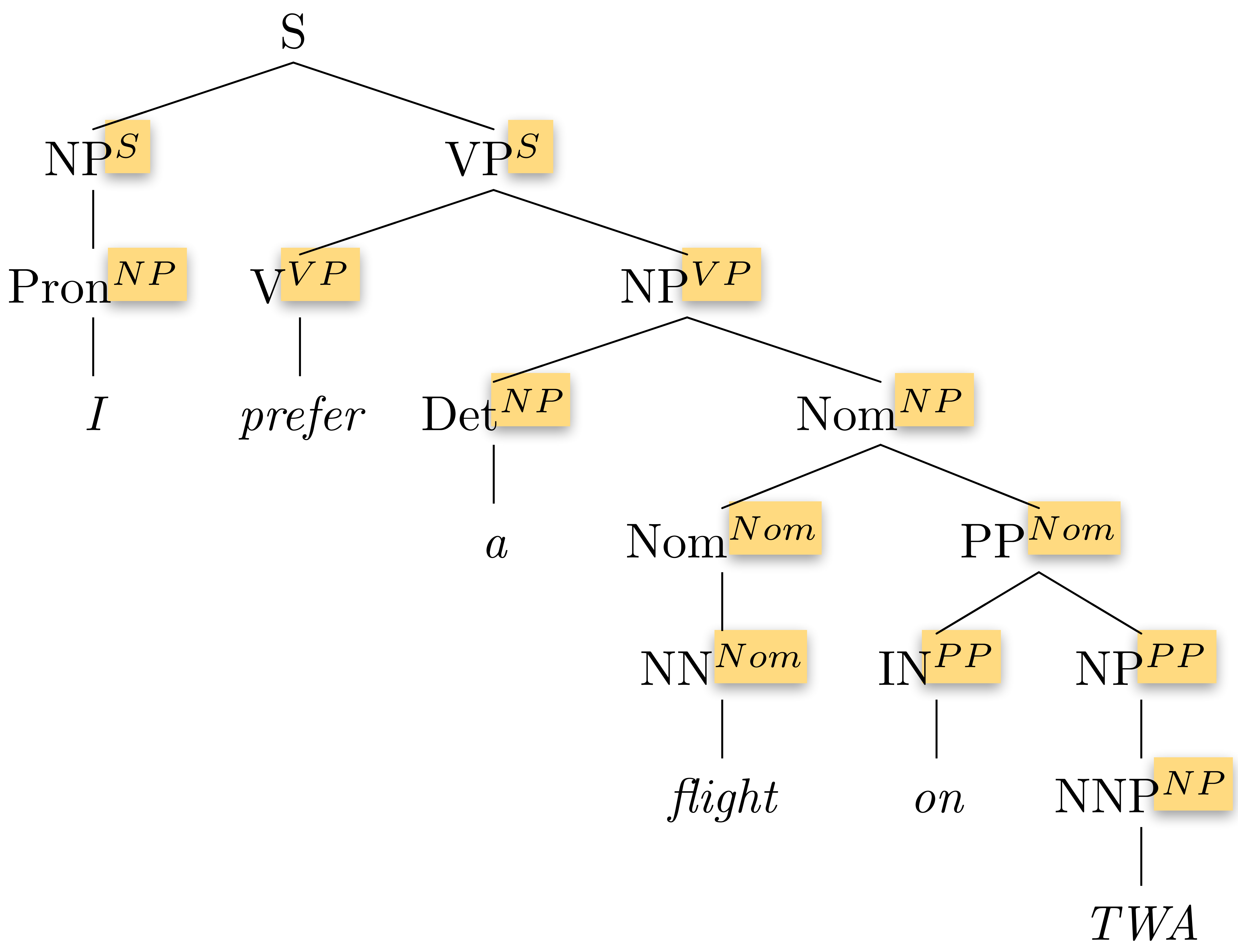
Improving PCFGs: Parent Annotation

- To handle the *NP* → *PRP* [0.91 **if** $NP_{\Theta=subject}$ **else** 0.34]



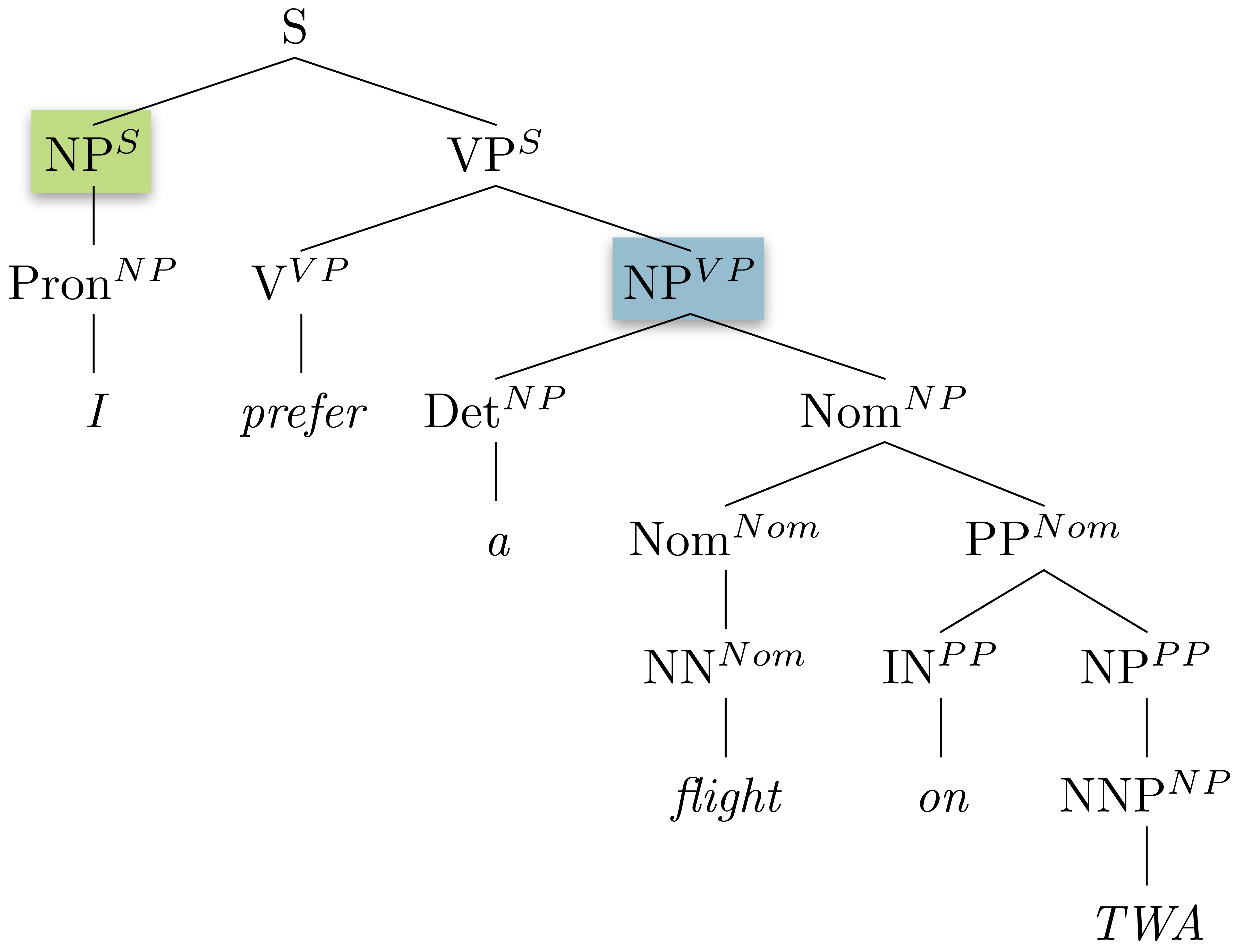
Improving PCFGs: Parent Annotation

- To handle the *NP* → *PRP* [0.91 **if** $NP_{\Theta=subject}$ **else** 0.34]



Improving PCFGs: Parent Annotation

- To handle the *NP* → *PRP* **[0.91 if** *NP*$_{\Theta=subject}$ **else** 0.34]



Improving PCFGs: Parent Annotation

- Advantages:
  - Captures structural dependencies in grammar

Improving PCFGs: Parent Annotation

- Advantages:
  - Captures structural dependencies in grammar
- Disadvantages:
  - Explodes number of rules in grammar
    - Same problem with subcategorization
  - Results in sparsity problems

Improving PCFGs: Parent Annotation

- Advantages:
  - Captures structural dependencies in grammar
- Disadvantages:
  - Explodes number of rules in grammar
    - Same problem with subcategorization
  - Results in sparsity problems
- Strategies to find an optimal number of splits
  - Petrov et al (2006)

Improving PCFGs

- Parent Annotation
- **Lexicalization**
- Reranking

Improving PCFGs: Lexical “Heads”

- Remember back to syntax intro (Lecture #1)
  - Phrases are “headed” by key words
    - **VP** are headed by **V**
    - **NP** by **NN, NNS, PRON**
    - **PP** by **PREP**

- We can take advantage of this in our grammar!

Improving PCFGs: Lexical Dependencies

- As we’ve seen, some rules should be conditioned on certain words

- **Proposal**: annotate nonterminals with lexical head

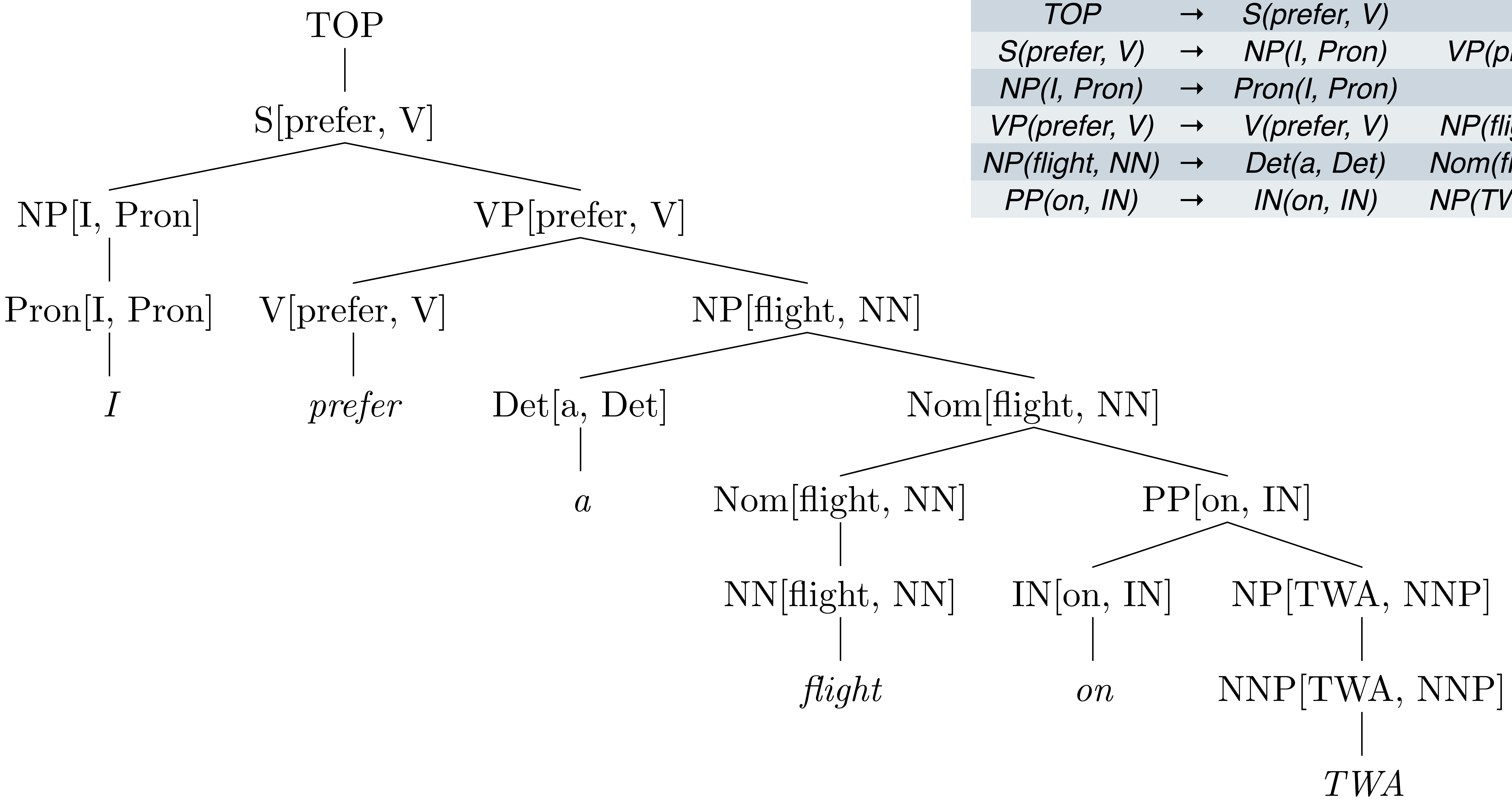
  VP → VBD NP PP

  *VP(**dumped**) → VBD(**dumped**) NP(**sacks**) PP(**into**)*

- **Additionally**: annotate with lexical head + POS

  *VP(dumped, **VBD**) → VBD(dumped, **VBD**) NP(sacks, **NNS**) PP(into, **IN**)*

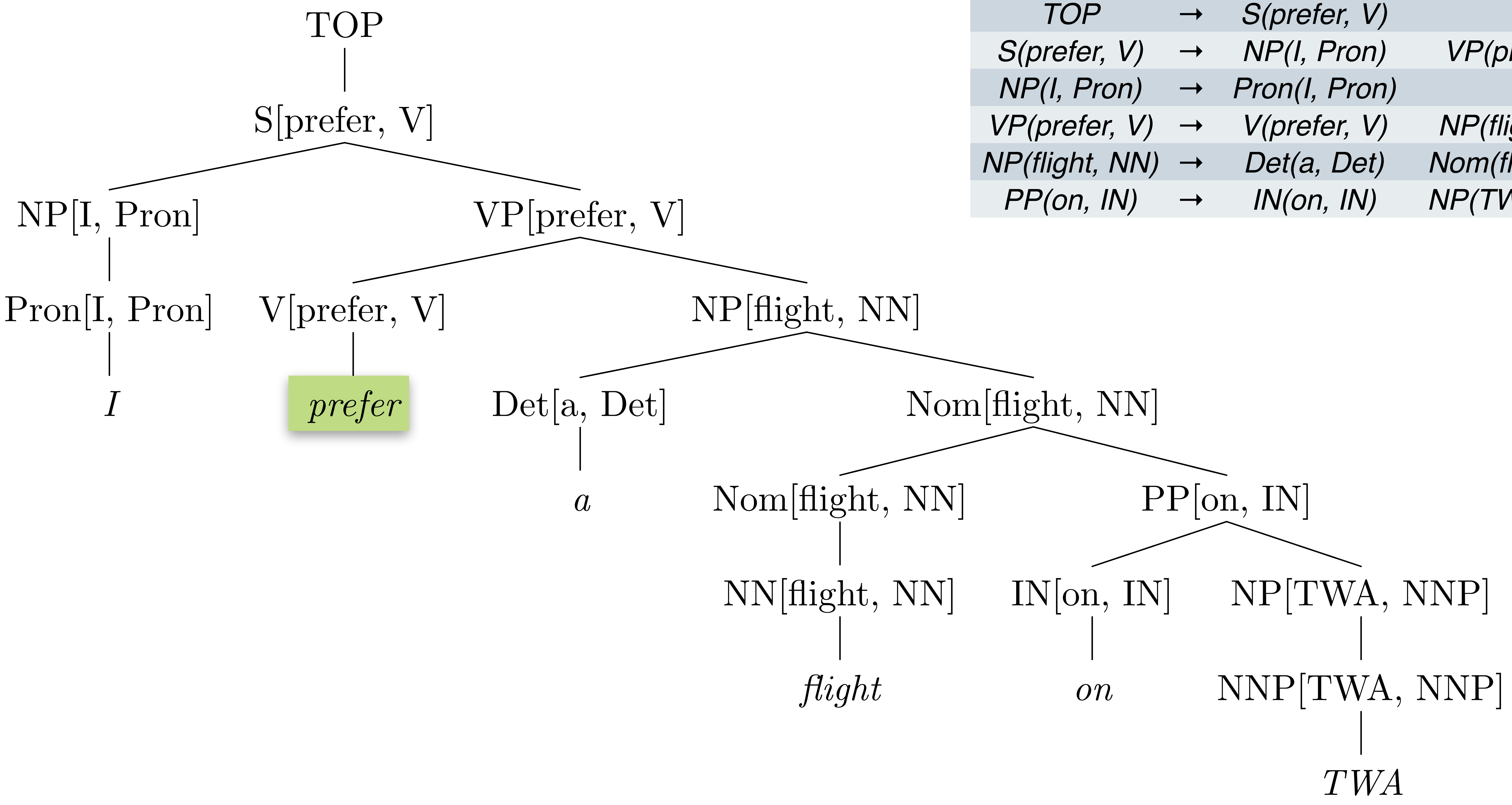
Lexicalized Parse Tree



Internal Rules			
TOP	→	*S(prefer, V)*	
S(prefer, V)	→	*NP(I, Pron)*	*VP(prefer, V)*
NP(I, Pron)	→	*Pron(I, Pron)*	
VP(prefer, V)	→	*V(prefer, V)*	*NP(flight, NN)*
NP(flight, NN)	→	*Det(a, Det)*	*Nom(flight, NN)*
PP(on, IN)	→	*IN(on, IN)*	*NP(TWA, NNP)*

Lexical Rules		
Pron(I, Pron)	→	I
V(prefer, V)	→	prefer
Det(a, Det)	→	a
NN(flight, NN)	→	flight
IN(on, IN)	→	on
NNP(NWA, NNP)	→	TWA

Lexicalized Parse Tree

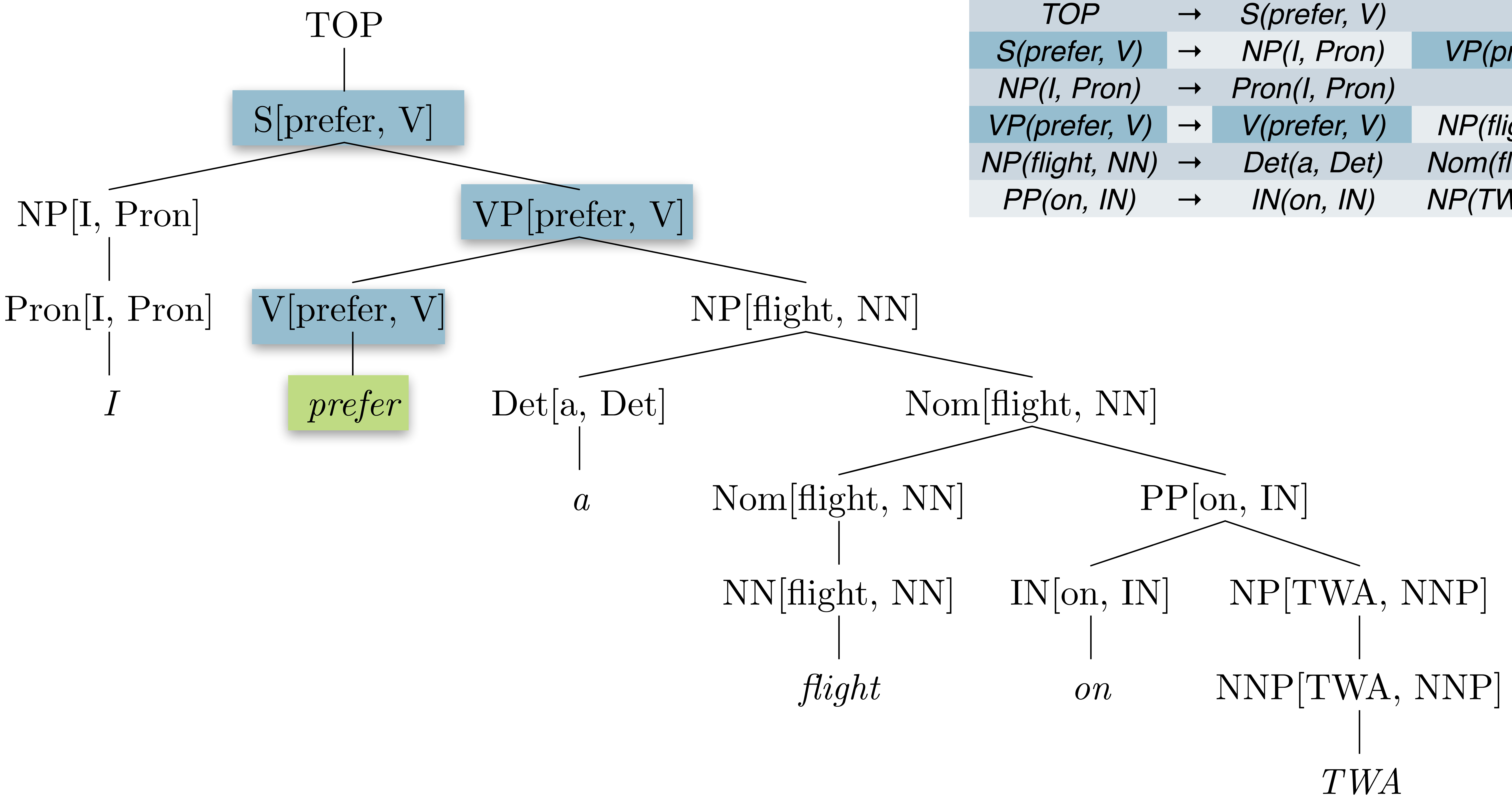


TOP
S[prefer, V]
NP[I, Pron]
VP[prefer, V]
Pron[I, Pron]
V[prefer, V]
NP[flight, NN]
I
prefer
Det[a, Det]
Nom[flight, NN]
a
Nom[flight, NN]
PP[on, IN]
NN[flight, NN]
IN[on, IN]
NP[TWA, NNP]
flight
on
NNP[TWA, NNP]
TWA

Internal Rules			
TOP	→	*S(prefer, V)*	
S(prefer, V)	→	*NP(I, Pron)*	*VP(prefer, V)*
NP(I, Pron)	→	*Pron(I, Pron)*	
VP(prefer, V)	→	*V(prefer, V)*	*NP(flight, NN)*
NP(flight, NN)	→	*Det(a, Det)*	*Nom(flight, NN)*
PP(on, IN)	→	*IN(on, IN)*	*NP(TWA, NNP)*

Lexical Rules		
Pron(I, Pron)	→	I
V(prefer, V)	→	prefer
Det(a, Det)	→	a
NN(flight, NN)	→	flight
IN(on, IN)	→	on
NNP(NWA, NNP)	→	TWA

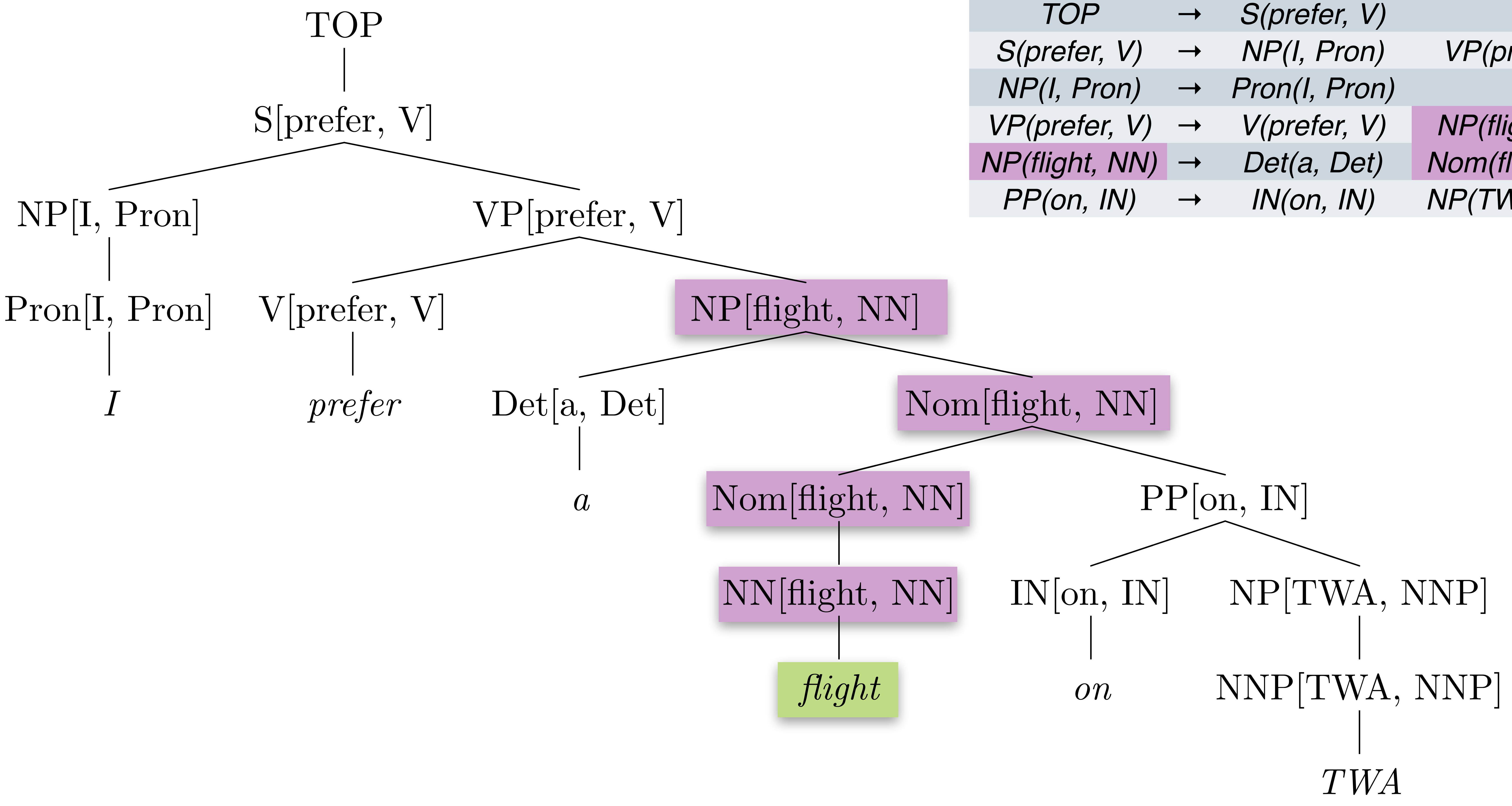
Lexicalized Parse Tree



Internal Rules			
TOP	→	*S(prefer, V)*	
S(prefer, V)	→	*NP(I, Pron)*	*VP(prefer, V)*
NP(I, Pron)	→	*Pron(I, Pron)*	
VP(prefer, V)	→	*V(prefer, V)*	*NP(flight, NN)*
NP(flight, NN)	→	*Det(a, Det)*	*Nom(flight, NN)*
PP(on, IN)	→	*IN(on, IN)*	*NP(TWA, NNP)*

Lexical Rules		
Pron(I, Pron)	→	I
V(prefer, V)	→	prefer
Det(a, Det)	→	a
NN(flight, NN)	→	flight
IN(on, IN)	→	on
NNP(NWA, NNP)	→	TWA

Lexicalized Parse Tree



Internal Rules			
TOP	→	S(prefer, V)	
S(prefer, V)	→	NP(I, Pron)	VP(prefer, V)
NP(I, Pron)	→	Pron(I, Pron)	
VP(prefer, V)	→	V(prefer, V)	NP(flight, NN)
NP(flight, NN)	→	Det(a, Det)	Nom(flight, NN)
PP(on, IN)	→	IN(on, IN)	NP(TWA, NNP)

Lexical Rules		
Pron(I, Pron)	→	I
V(prefer, V)	→	prefer
Det(a, Det)	→	a
NN(flight, NN)	→	flight
IN(on, IN)	→	on
NNP(NWA, NNP)	→	TWA

Improving PCFGs: Lexical Dependencies

- Upshot: heads propagate up tree:

Improving PCFGs: Lexical Dependencies

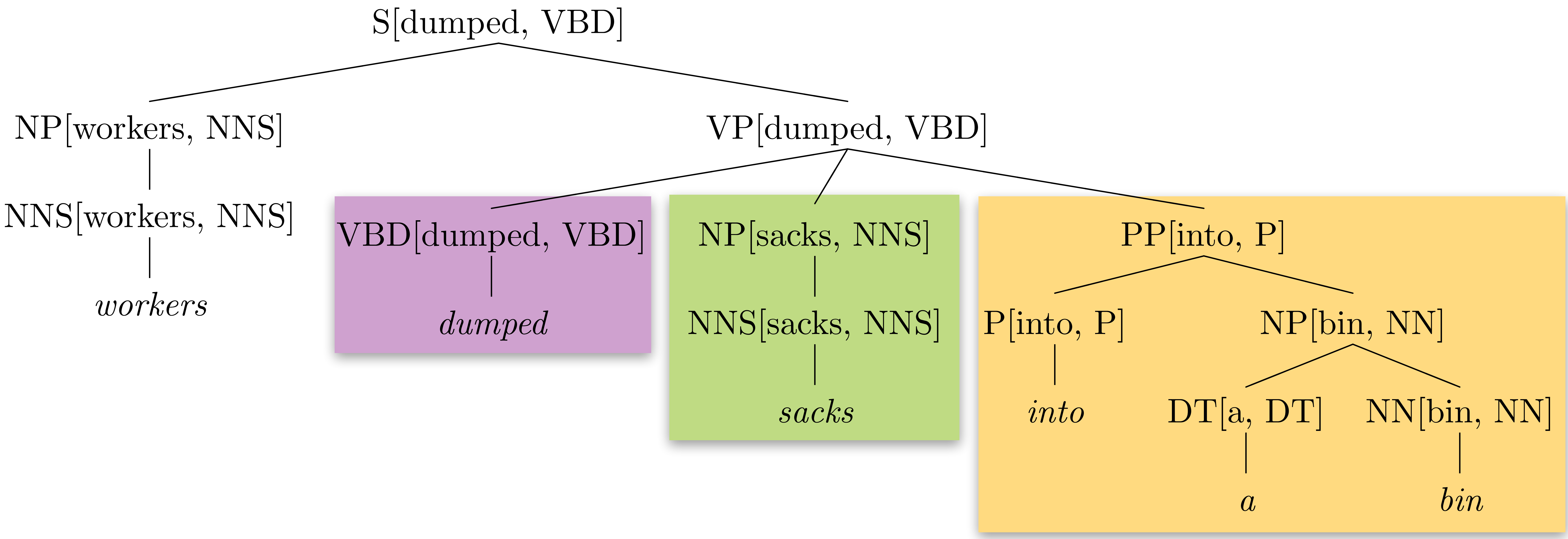
- Upshot: heads propagate up tree:
  - *VP → VBD(dumped, VBD) NP(sacks, NNS) PP(into, P)*
  - *NP → NNS(sacks, NNS) PP(into, P)*

Improving PCFGs: Lexical Dependencies

- Upshot: heads propagate up tree:
  - *VP → VBD(dumped, VBD) NP(sacks, NNS) PP(into, P)* ✔
  - *NP → NNS(sacks, NNS) PP(into, P)* ✘

Improving PCFGs: Lexical Dependencies

- Upshot: heads propagate up tree:
  - *VP → VBD(dumped, VBD) NP(sacks, NNS) PP(into, P)* ✓
  - *NP → NNS(sacks, NNS) PP(into, P)* ✗



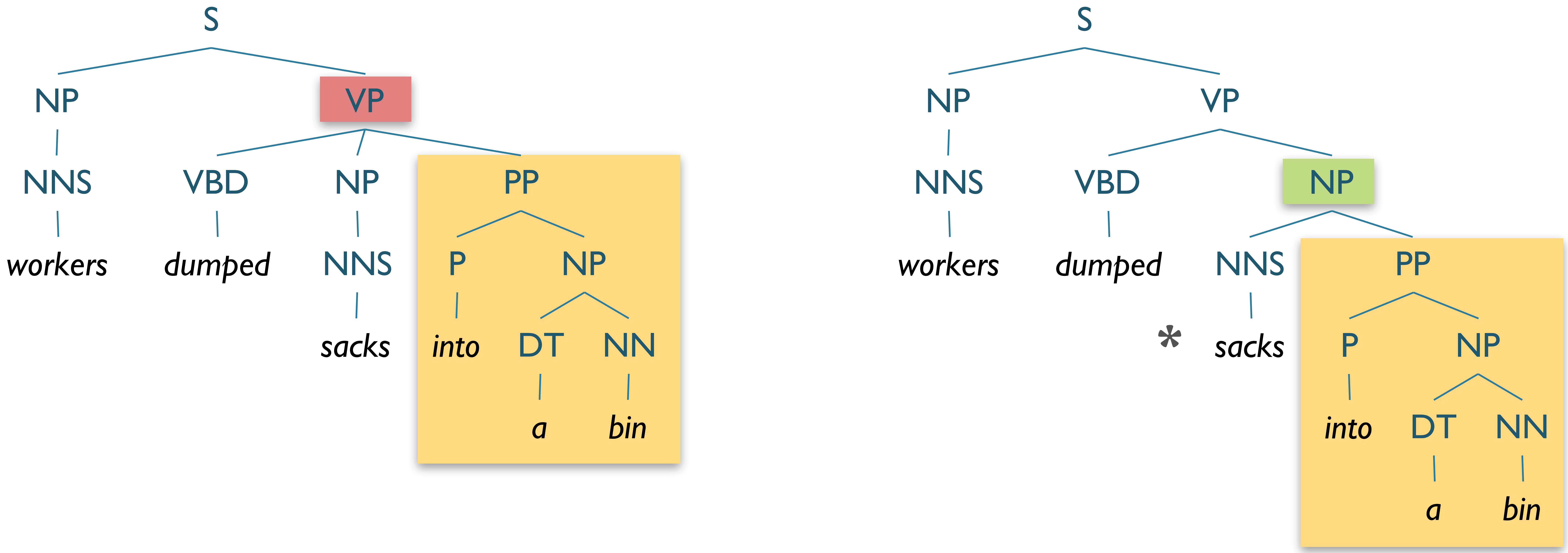
Improving PCFGs: Lexical Dependencies

- Downside:
  - Rules far too specialized — will be sparse
- Solution:
  - Assume ***conditional*** independence
  - Create more rules

Improving PCFGs: Collins Parser

- Proposal:
  - ***LHS*** → ***LeftOfHead*** … ***Head*** … ***RightOfHead***
  - Instead of calculating *P*(*EntireRule*), which is sparse:
  - Calculate:
    - Probability that ***LHS*** has nonterminal phrase ***H*** given head-word ***hw***…
    - × Probability of modifiers to the **left** given head-word ***hw***…
    - × Probability of modifiers to the **right** given head-word ***hw***…

Collins Parser Example



Collins Parser Example

$$P(VP \rightarrow VBD\ NP\ PP \mid VP, dumped)$$

Collins Parser Example

$$P(VP \rightarrow VBD\ NP\ PP \mid VP,\ dumped)$$

$$= \frac{Count\left(VP\left(dumped\right) \rightarrow VBD\ NP\ PP\right)}{\sum_{\beta} Count\left(VP\left(dumped\right) \rightarrow \beta\right)}$$

Collins Parser Example

$$P(VP \rightarrow VBD\ NP\ PP \mid VP,\ dumped)$$

$$= \frac{Count\left(VP\left(dumped\right) \rightarrow VBD\ NP\ PP\right)}{\sum_{\beta} Count\left(VP\left(dumped\right) \rightarrow \beta\right)}$$

$$= \frac{6}{9} = 0.67$$

Collins Parser Example

$$P(VP \rightarrow VBD\ NP\ PP \mid VP,\ dumped)$$

$$= \frac{Count\left(VP\left(dumped\right) \rightarrow VBD\ NP\ PP\right)}{\sum_{\beta} Count\left(VP\left(dumped\right) \rightarrow \beta\right)}$$

$$= \frac{6}{9} = 0.67$$

$$P_R(into \mid PP, dumped)$$

Collins Parser Example

$$P(VP \rightarrow VBD\ NP\ PP \,|\, VP,\ dumped)$$

$$= \frac{Count\left(VP\left(dumped\right) \rightarrow VBD\ NP\ PP\right)}{\sum_{\beta} Count\left(VP\left(dumped\right) \rightarrow \beta\right)}$$

$$= \frac{6}{9} = 0.67$$

$$P_R(into \,|\, PP, dumped)$$

$$= \frac{Count\left(X\left(dumped\right) \rightarrow \ldots\ PP\left(into\right)\ \ldots\right)}{\sum_{\beta} Count\left(X\left(dumped\right) \rightarrow \ldots\ PP\ \ldots\right)}$$

Collins Parser Example

$$P(VP \rightarrow VBD\ NP\ PP \mid VP,\ dumped)$$

$$= \frac{Count\left(VP\left(dumped\right) \rightarrow VBD\ NP\ PP\right)}{\sum_{\beta} Count\left(VP\left(dumped\right) \rightarrow \beta\right)}$$

$$= \frac{6}{9} = 0.67$$

$$P_R(into \mid PP, dumped)$$

$$= \frac{Count\left(X\left(dumped\right) \rightarrow \ldots\ PP\left(into\right)\ \ldots\right)}{\sum_{\beta} Count\left(X\left(dumped\right) \rightarrow \ldots\ PP\ \ldots\right)}$$

$$= \frac{2}{9} = 0.22$$

Collins Parser Example

$$P(VP \rightarrow VBD\ NP\ PP \mid VP, dumped)$$

$$= \frac{Count\left(VP\left(dumped\right) \rightarrow VBD\ NP\ PP\right)}{\sum_{\beta} Count\left(VP\left(dumped\right) \rightarrow \beta\right)}$$

$$= \frac{6}{9} = 0.67$$

$$P(VP \rightarrow VBD\ NP \mid VP, dumped)$$

$$= \frac{Count\left(VP\left(dumped\right) \rightarrow VBD\ NP\right)}{\sum_{\beta} Count\left(VP\left(dumped\right) \rightarrow \beta\right)}$$

$$= \frac{1}{9} = 0.11$$

$$P_R(into \mid PP, dumped)$$

$$= \frac{Count\left(X\left(dumped\right) \rightarrow \ldots\ PP\left(into\right)\ \ldots\right)}{\sum_{\beta} Count\left(X\left(dumped\right) \rightarrow \ldots\ PP\ \ldots\right)}$$

$$= \frac{2}{9} = 0.22$$

Collins Parser Example

$$P(VP \rightarrow VBD\ NP\ PP \,|\, VP, dumped)$$

$$= \frac{Count\left(VP\left(dumped\right) \rightarrow VBD\ NP\ PP\right)}{\sum_{\beta} Count\left(VP\left(dumped\right) \rightarrow \beta\right)}$$

$$= \frac{6}{9} = 0.67$$

$$P(VP \rightarrow VBD\ NP \,|\, VP, dumped)$$

$$= \frac{Count\left(VP\left(dumped\right) \rightarrow VBD\ NP\right)}{\sum_{\beta} Count\left(VP\left(dumped\right) \rightarrow \beta\right)}$$

$$= \frac{1}{9} = 0.11$$

$$P_R(into \,|\, PP, dumped)$$

$$= \frac{Count\left(X\left(dumped\right) \rightarrow \ldots\ PP\left(into\right)\ \ldots\right)}{\sum_{\beta} Count\left(X\left(dumped\right) \rightarrow \ldots\ PP\ \ldots\right)}$$

$$= \frac{2}{9} = 0.22$$

$$P_R(into \,|\, PP, sacks)$$

$$= \frac{Count\left(X\left(sacks\right) \rightarrow \ldots\ PP\left(into\right)\ \ldots\right)}{\sum_{\beta} Count\left(X\left(sacks\right) \rightarrow \ldots\ PP\ \ldots\right)}$$

$$= \frac{0}{0}$$

Improving PCFGs

- Parent Annotation
- Lexicalization
- **Reranking**

Reranking

- Issue: Locality
  - PCFG probabilities associated with rewrite rules
  - Context-free grammars are, well, context-free
  - Previous approaches create new rules to incorporate context
- Need approach that incorporates broader, global info

Discriminative Parse Reranking

- General approach:
  - Parse using (L)PCFG
  - Obtain top-N parses
  - Re-rank top-N using better features
- Use discriminative model (e.g. MaxEnt, NN) to rerank with features:
  - right-branching vs. left-branching
  - speaker identity
  - conjunctive parallelism
  - fragment frequency
  - …

Reranking Effectiveness

- How can reranking improve?
- Results from Collins and Koo (2005), with 50-best

System	Accuracy
Baseline	0.897
Oracle	0.968
Discriminative	0.917

- “Oracle” is to automatically choose the correct parse if in N-best

Improving PCFGs: Tradeoffs

- **Pros:**
  - Increased accuracy/specificity
  - e.g. Lexicalization, Parent annotation, Reranking
- **Cons**:
  - Explode grammar size
  - Increased processing time
  - Increased data requirements
- *How can we balance?*

Improving PCFGs: Efficiency

- **Beam thresholding**
- Heuristic Filtering

Efficiency

- PCKY is $|G| \cdot n^3$
  - Grammar can be huge
  - Grammar can be extremely ambiguous
  - Hundreds of analyses not unusual
- ...but only care about best parses
- Can we use this to improve efficiency?

Beam Thresholding

- Inspired by Beam Search
- Assume low probability parses unlikely to yield high probability overall
  - Keep only top k most probable partial parses
  - Retain only k choices per cell
    - For large grammars, maybe 50-100
    - For small grammars, 5 or 10

Heuristic Filtering

- **Intuition**: Some rules/partial parses unlikely to create best parse
- **Proposal**: Don’t store these in table.
- Exclude:
  - Low frequency: e.g. singletons
  - Low probability: constituents $\boldsymbol{X}$ s.t. $P(\boldsymbol{X}) < 10^{-200}$
  - Low relative probability:
    - Exclude $\boldsymbol{X}$ if there exists $\boldsymbol{Y}$ s.t. $P(\boldsymbol{Y}) > 100 \times P(\boldsymbol{X})$