EECS 498-004: Introduction to Natural Language Processing

Instructor: Prof. Lu Wang
Computer Science and Engineering
University of Michigan
https://web.eecs.umich.edu/~wangluxy/

1

1

Two views of linguistic structure: 1. Constituency (phrase structure)

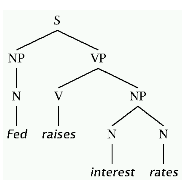
- Phrase structure organizes words into nested constituents.
  - Fed raises interest rates

2

2

Two views of linguistic structure: 1. Constituency (phrase structure)

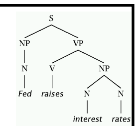
- Phrase structure organizes words into nested constituents.



3

3

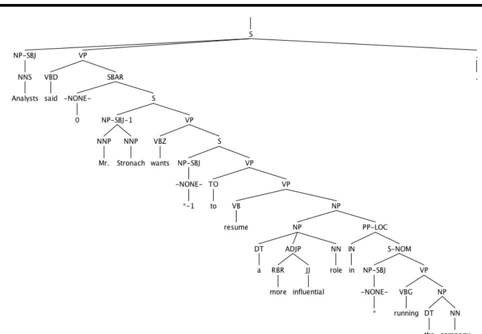
Two views of linguistic structure: 1. Constituency (phrase structure)



- Phrase structure organizes words into nested constituents.
- How do we know what is a constituent? (Not that linguists don't argue about some cases.)
  - Distribution: a constituent behaves as a unit that can appear in different places:
    - John talked [to the children] [about drugs].
    - John talked [about drugs] [to the children].
    - *John talked drugs to the children about
  - Substitution/expansion/pronoun:
    - I sat [on the box/right on top of the box/there].

4

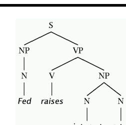
4



5

5

Headed phrase structure



- Context-free grammar
- VP → … VB* …
- NP → … NN* …
- ADJP → … JJ* …
- ADVP → … RB* …

- S → … NP VP …

- Plus minor phrase types:
  - QP (quantifier phrase in NP: *some people*), CONJP (multi word constructions: *as well as*), INTJ (interjections: *aha*), etc.

6

6

Two views of linguistic structure: 2. Dependency structure

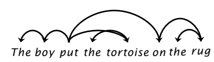
- Dependency structure shows which words depend on (modify or are arguments of) which other words.

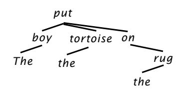
The boy put the tortoise on the rug

7

Two views of linguistic structure: 2. Dependency structure

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





8

Outline

- Phrase Chunking
- (Probabilistic) Context-Free Grammars
- Chomsky Normal Form
- CKY Parsing

9

Phrase Chunking

- Find all non-recursive noun phrases (NPs) and verb phrases (VPs) in a sentence.
  - [NP I] [VP ate] [NP the spaghetti] [PP with] [NP meatballs].
  - [NP He] [VP reckons] [NP the current account deficit] [VP will narrow] [PP to] [NP only 1.8 billion] [PP in] [NP September] .

10

Phrase Chunking as Sequence Labeling

- Tag individual words with one of 3 tags
  - B (Begin) word starts new target phrase
  - I (Inside) word is part of target phrase but not the first word
  - O (Other) word is not part of target phrase
- Sample for NP chunking
  - He reckons the current account deficit will narrow to only 1.8 billion in September.

Begin **Inside** **Other**

11

Evaluating Chunking

Per token accuracy does not evaluate finding correct full chunks. Instead use:

$$\text{Precision} = \frac{\text{Number of correct chunks found}}{\text{Total number of chunks found}}$$

$$\text{Recall} = \frac{\text{Number of correct chunks found}}{\text{Total number of actual chunks}}$$

$$\text{F measure:} \quad F_1 = \frac{1}{(\frac{1}{P} + \frac{1}{R})/2} = \frac{2PR}{P+R}$$

12

Current Chunking Results

- Best system for NP chunking: F_1=96%
- Typical results for finding range of chunk types (CoNLL 2000 shared task: NP, VP, PP, ADV, SBAR, ADJP) is F_1=92−94%

13

13

Outline

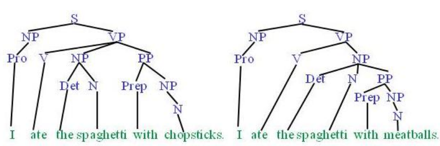
- Phrase Chunking
- ➡ (Probabilistic) Context-Free Grammars
- Chomsky Normal Form
- CKY Parsing

14

14

Syntactic Parsing

- Produce the correct syntactic parse tree for a sentence.



15

15

Annotated data: The Penn Treebank

[Marcus et al. 1993, *Computational Linguistics*]

```
( (S
   (NP-SBJ (DT The) (NN move))
   (VP (VBD followed)
    (NP
     (NP (DT a) (NN round))
     (PP (IN of)
      (NP
       (NP (JJ similar) (NNS increases))
       (PP (IN by)
        (NP (JJ other) (NNS lenders)))
       (PP (IN against)
        (NP (NNP Arizona) (JJ real) (NN estate) (NNS loans))))))
    (, ,)
    (S-ADV
     (NP-SBJ (-NONE- *))
     (VP (VBG reflecting)
      (NP
       (NP (DT a) (VBG continuing) (NN decline))
       (PP-LOC (IN in)
        (NP (DT that) (NN market)))))))
   (. .)))
```

16

16

The rise of annotated data

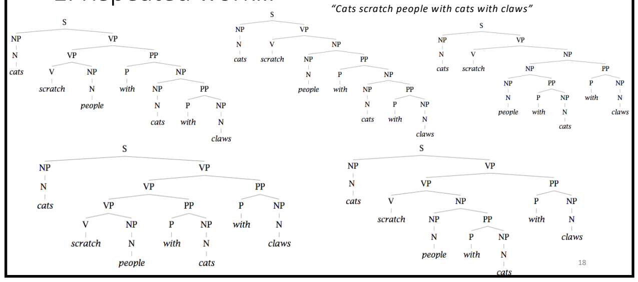
- Starting off, building a treebank seems a lot slower and less useful than building a grammar
- But a treebank gives us many things
  - Reusability of the labor
    - Many parsers, POS taggers, etc.
    - Valuable resource for linguistics
  - Broad coverage
  - Frequencies and distributional information
  - A way to evaluate systems

17

17

Two problems to solve for parsing: 1. Repeated work…

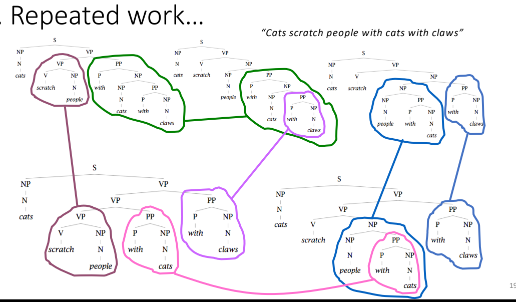
"Cats scratch people with cats with claws"



18

18

Two problems to solve for parsing: 1. Repeated work…



19

Two problems to solve for parsing: 2. Choosing the correct parse

- How do we work out the correct attachment:
  - She saw the man with a telescope
- Words are good predictors of attachment, even absent full understanding
  - Moscow **sent** more than 100,000 soldiers **into** Afghanistan …
  - Sydney Water breached an **agreement with** NSW Health …
- Our statistical parsers will try to exploit such statistics.

20

Statistical parsing applications

Statistical parsers are now robust and widely used in larger NLP applications:

- High precision question answering [Pasca and Harabagiu SIGIR 2001]
- Improving biological named entity finding [Finkel et al. JNLPBA 2004]
- Syntactically based sentence compression [Lin and Wilbur 2007]
- Extracting opinions about products [Bloom et al. NAACL 2007]
- Improved interaction in computer games [Gorniak and Roy 2005]
- Helping linguists find data [Resnik et al. BLS 2005]
- Source sentence analysis for machine translation [Xu et al. 2009]
- Relation extraction systems [Fundel et al. Bioinformatics 2006]

21

(Probabilistic) Context-Free Grammars

- CFG
- PCFG

22

Phrase structure grammars = context-free grammars (CFGs)

- G = (T, N, S, R)
  - T is a set of terminal symbols
  - N is a set of nonterminal symbols
  - S is the start symbol (S ∈ N)
  - R is a set of rules/productions of the form X → γ
    - X ∈ N and γ ∈ (N ∪ T)*

23

A phrase structure grammar

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → *e*
PP → P NP

people fish tanks
people fish with rods

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
V → *fish*
V → *tanks*
P → *with*

24

Phrase structure grammars = context-free grammars (CFGs)

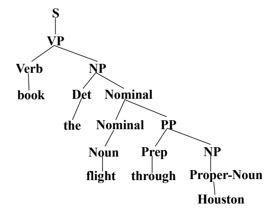
- G = (T, N, S, R)
  - T is a set of terminal symbols
  - N is a set of nonterminal symbols
  - S is the start symbol (S ∈ N)
  - R is a set of rules/productions of the form $X \rightarrow \gamma$
    - $X \in N$ and $\gamma \in (N \cup T)^*$
- A grammar G generates a language L.

25

25

Sentence Generation

- Sentences are generated by recursively rewriting the start symbol using the productions until only terminals symbols remain.



26

26

Phrase structure grammars in NLP

- G = (T, C, N, S, L, R)
  - T is a set of terminal symbols
  - C is a set of preterminal symbols
  - N is a set of nonterminal symbols
  - S is the start symbol (S ∈ N)
  - L is the lexicon, a set of items of the form $X \rightarrow x$
    - $X \in C$ and $x \in T$
  - R is the grammar, a set of items of the form $X \rightarrow \gamma$
    - $X \in N$ and $\gamma \in (N \cup C)^*$
- By usual convention, S is the start symbol, but in statistical NLP, we usually have an extra node at the top (ROOT, TOP)
- We usually write *e* for an empty sequence, rather than nothing

27

27

A phrase structure grammar

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → *e*
PP → P NP

people fish tanks
people fish with rods

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
V → *fish*
V → *tanks*
P → *with*

28

28

Probabilistic – or stochastic – context-free grammars (PCFGs)

- G = (T, N, S, R, P)
  - T is a set of terminal symbols
  - N is a set of nonterminal symbols
  - S is the start symbol (S ∈ N)
  - R is a set of rules/productions of the form $X \rightarrow \gamma$
  - P is a probability function
    - $P: R \rightarrow [0,1]$
    - $\forall X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1$
- A grammar G generates a language model L.

29

29

A PCFG

S → NP VP	1.0	N → *people*	0.5
VP → V NP	0.6	N → *fish*	0.2
VP → V NP PP	0.4	N → *tanks*	0.2
NP → NP NP	0.1	N → *rods*	0.1
NP → NP PP	0.2	V → *people*	0.1
NP → N	0.7	V → *fish*	0.6
PP → P NP	1.0	V → *tanks*	0.3
		P → *with*	1.0

[With empty NP removed so less ambiguous]

30

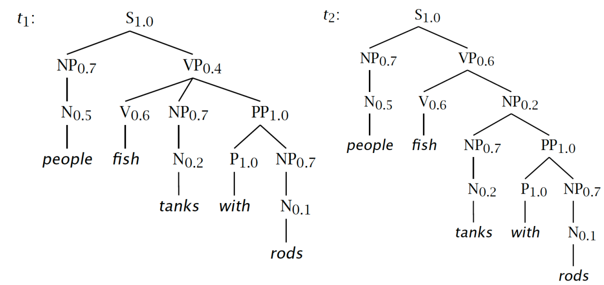
30

The probability of trees and strings

- P(*t*) – The probability of a tree *t* is the product of the probabilities of the rules used to generate it.
- P(*s*) – The probability of the string *s* is the sum of the probabilities of the trees which have that string as their yield

$P(s) = \Sigma_t P(s, t)$ where t is a parse of s

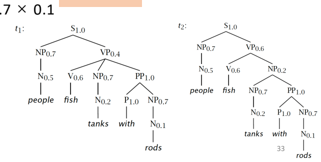
31



32

Tree and String Probabilities

- s = *people fish tanks with rods*
- $P(t_1)$ = 1.0 × 0.7 × 0.4 × 0.5 × 0.6 × 0.7 × 1.0 × 0.2 × 1.0 × 0.7 × 0.1 = 0.0008232 — Verb attach
- $P(t_2)$ = 1.0 × 0.7 × 0.6 × 0.5 × 0.6 × 0.2 × 0.7 × 1.0 × 0.2 × 1.0 × 0.7 × 0.1 = 0.00024696 — Noun attach
- $P(s) = P(t_1) + P(t_2)$ = 0.0008232 + 0.00024696 = 0.00107016



33

Outline

- Phrase Chunking
- (Probabilistic) Context-Free Grammars
- ➡ Chomsky Normal Form
- CKY Parsing

34

Chomsky Normal Form

- All rules are of the form X → Y Z or X → w
  - X, Y, Z ∈ N and w ∈ T
- A transformation to this form doesn't change the generative capacity of a CFG
  - That is, it recognizes the same language
    - But maybe with different trees
- Empties and unaries are removed recursively
- n-ary rules are divided by introducing new nonterminals (n > 2)

35

A phrase structure grammar

S → NP VP	N → *people*
VP → V NP	N → *fish*
VP → V NP PP	N → *tanks*
NP → NP NP	N → *rods*
NP → NP PP	V → *people*
NP → N	V → *fish*
NP → *e*	V → *tanks*
PP → P NP	P → *with*

36

Chomsky Normal Form steps

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

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
V → *fish*
V → *tanks*
P → *with*

37

37

Chomsky Normal Form steps

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

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
V → *fish*
V → *tanks*
P → *with*

38

38

Chomsky Normal Form steps

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

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
S → *people*
V → *fish*
S → *fish*
V → *tanks*
S → *tanks*
P → *with*

39

39

Chomsky Normal Form steps

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

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
S → *people*
VP → *people*
V → *fish*
S → *fish*
VP → *fish*
V → *tanks*
S → *tanks*
VP → *tanks*
P → *with*

40

40

Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

NP → *people*
NP → *fish*
NP → *tanks*
NP → *rods*
V → *people*
S → *people*
VP → *people*
V → *fish*
S → *fish*
VP → *fish*
V → *tanks*
S → *tanks*
VP → *tanks*
P → *with*
PP → *with*

41

41

Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V @VP_V
@VP_V → NP PP
S → V @S_V
@S_V → NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

NP → *people*
NP → *fish*
NP → *tanks*
NP → *rods*
V → *people*
S → *people*
VP → *people*
V → *fish*
S → *fish*
VP → *fish*
V → *tanks*
S → *tanks*
VP → *tanks*
P → *with*
PP → *with*

42

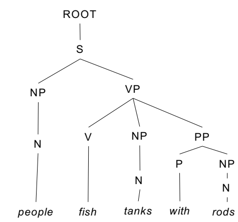
42

Chomsky Normal Form

- You should think of this as a transformation for efficient parsing
- **Binarization** is crucial for cubic time CFG parsing
- The rest isn't necessary; it just makes the algorithms cleaner and a bit quicker

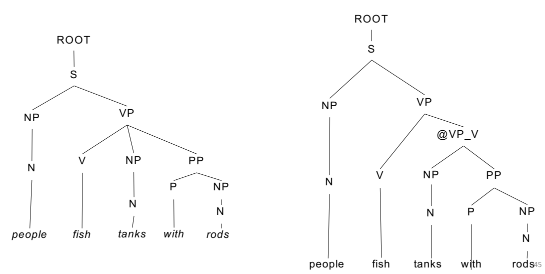
43

An example: before binarization…



44

Before and After binarization on VP



45

Outline

- Phrase Chunking
- (Probabilistic) Context-Free Grammars
- Chomsky Normal Form
- CKY Parsing

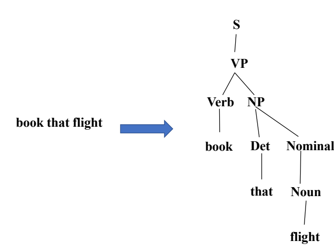
46

Parsing

- Given a string of terminals (e.g. sentences) and a CFG, determine if the string can be generated by the CFG.
  - Also return a parse tree for the string
  - Also return all possible parse trees for the string
- Must search space of derivations for one that derives the given string.
  - **Top-Down Parsing**: Start searching space of derivations for the start symbol.
  - **Bottom-up Parsing**: Start search space of reverse derivations from the terminal symbols in the string.

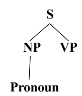
47

Parsing Example



48

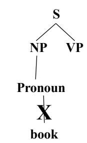
Top Down Parsing



49

49

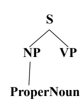
Top Down Parsing



50

50

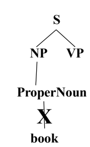
Top Down Parsing



51

51

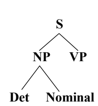
Top Down Parsing



52

52

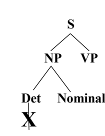
Top Down Parsing



53

53

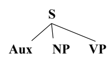
Top Down Parsing



54

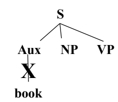
54

Top Down Parsing



55

Top Down Parsing

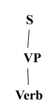


56

Top Down Parsing

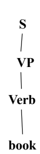
57

Top Down Parsing



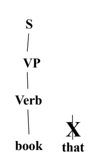
58

Top Down Parsing



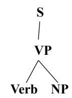
59

Top Down Parsing



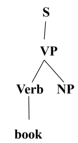
60

Top Down Parsing



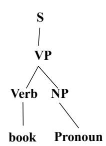
61

Top Down Parsing



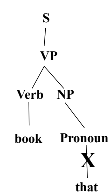
62

Top Down Parsing



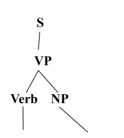
63

Top Down Parsing



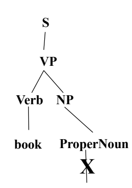
64

Top Down Parsing



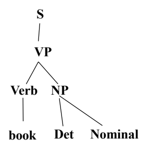
65

Top Down Parsing



66

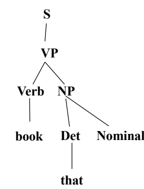
Top Down Parsing



67

67

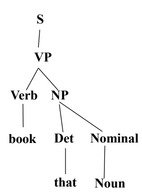
Top Down Parsing



68

68

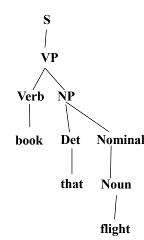
Top Down Parsing



69

69

Top Down Parsing



70

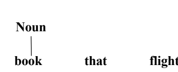
70

Bottom Up Parsing

71

71

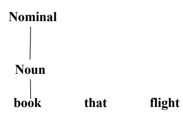
Bottom Up Parsing



72

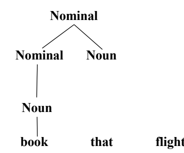
72

Bottom Up Parsing



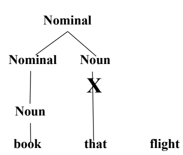
73

Bottom Up Parsing



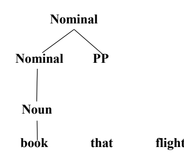
74

Bottom Up Parsing



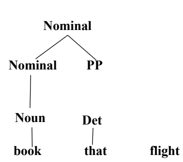
75

Bottom Up Parsing



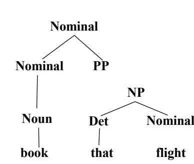
76

Bottom Up Parsing



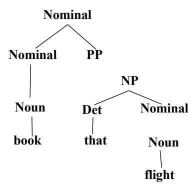
77

Bottom Up Parsing



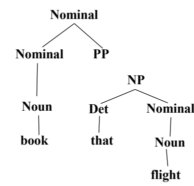
78

Bottom Up Parsing



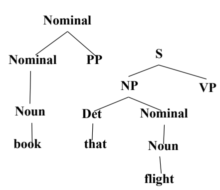
79

Bottom Up Parsing



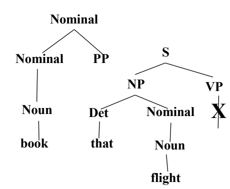
80

Bottom Up Parsing



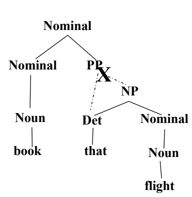
81

Bottom Up Parsing



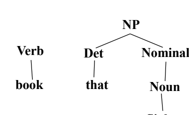
82

Bottom Up Parsing



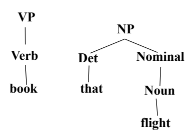
83

Bottom Up Parsing



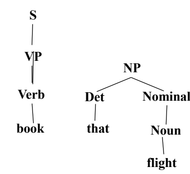
84

Bottom Up Parsing



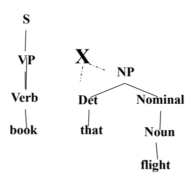
85

Bottom Up Parsing



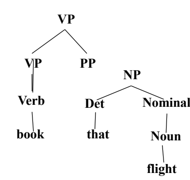
86

Bottom Up Parsing



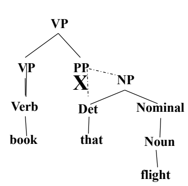
87

Bottom Up Parsing



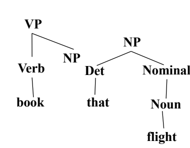
88

Bottom Up Parsing



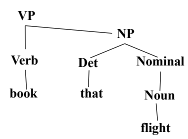
89

Bottom Up Parsing



90

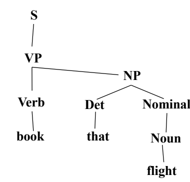
Bottom Up Parsing



91

91

Bottom Up Parsing



92

92

Top Down vs. Bottom Up

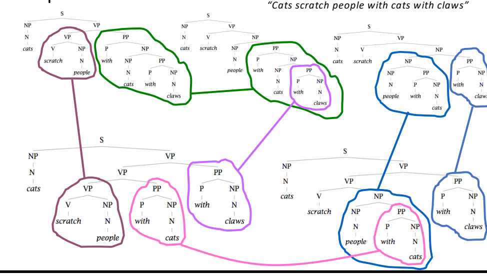
- Top down never explores options that will not lead to a full parse, but can explore many options that never connect to the actual sentence.
- Bottom up never explores options that do not connect to the actual sentence but can explore options that can never lead to a full parse.
- Relative amounts of wasted search depend on how much the grammar branches in each direction.

93

93

Two problems to solve for parsing: 1. Repeated work

"Cats scratch people with cats with claws"



94

94

Dynamic Programming Parsing

- To avoid extensive repeated work, must cache intermediate results, i.e. completed phrases.
- Caching (memorizing) is critical to obtaining a polynomial time parsing (recognition) algorithm for CFGs.

95

95

(Probabilistic) CKY Parsing

96

96

Constituency Parsing

Input: a PCFG, and a sentence

fish people fish tanks

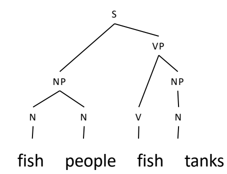
PCFG

Rule	Prob θ_i
S → NP VP	θ_0
NP → NP NP	θ_1
…	
N → fish	θ_{42}
N → people	θ_{43}
V → fish	θ_{44}
…	

97

Constituency Parsing

Output: a parsing tree

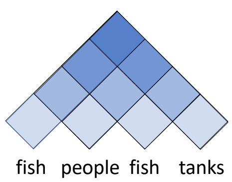


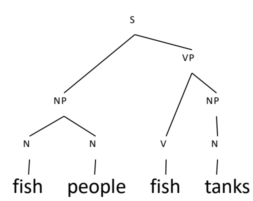
PCFG

Rule	Prob θ_i
S → NP VP	θ_0
NP → NP NP	θ_1
…	
N → fish	θ_{42}
N → people	θ_{43}
V → fish	θ_{44}
…	

98

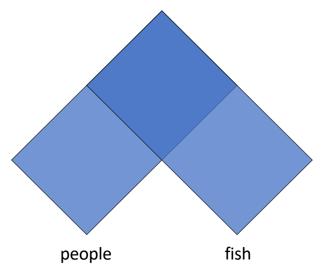
Cocke-Kasami-Younger (CKY) Constituency Parsing





99

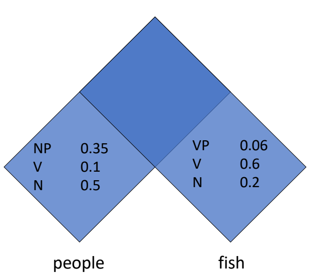
Reusing local decisions



NP → people	0.35
V → people	0.1
N → people	0.5
VP → fish	0.06
V → fish	0.6
N → fish	0.2
S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
NP → NP NP	0.1
NP → NP PP	0.2
PP → P NP	1.0

100

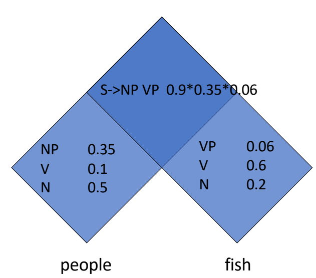
Reusing local decisions



NP → people	0.35
V → people	0.1
N → people	0.5
VP → fish	0.06
V → fish	0.6
N → fish	0.2
S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
NP → NP NP	0.1
NP → NP PP	0.2
PP → P NP	1.0

101

Reusing local decisions



NP → people	0.35
V → people	0.1
N → people	0.5
VP → fish	0.06
V → fish	0.6
N → fish	0.2
S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
NP → NP NP	0.1
NP → NP PP	0.2
PP → P NP	1.0

102

The CKY algorithm (1960/1965)
... extended to unaries

```
function CKY(words, grammar) returns [most_probable_parse,prob]
  score = new double[#(words)+1][#(words)+1][#(nonterms)]
  back = new Pair[#(words)+1][#(words)+1][#nonterms]]
  for i=0; i<#(words); i++
    for A in nonterms
      if A -> words[i] in grammar
        score[i][i+1][A] = P(A -> words[i])
    //handle unaries
    boolean added = true
    while added
      added = false
      for A, B in nonterms
        if score[i][i+1][B] > 0 && A->B in grammar
          prob = P(A->B)*score[i][i+1][B]
          if prob > score[i][i+1][A]
            score[i][i+1][A] = prob
            back[i][i+1][A] = B
            added = true
```

103

103

The CKY algorithm (1960/1965)
... extended to unaries

```
for span = 2 to #(words)
  for begin = 0 to #(words)- span
    end = begin + span
    for split = begin+1 to end-1
      for A,B,C in nonterms
        prob=score[begin][split][B]*score[split][end][C]*P(A->BC)
        if prob > score[begin][end][A]
          score[begin]end][A] = prob
          back[begin][end][A] = new Triple(split,B,C)
    //handle unaries
    boolean added = true
    while added
      added = false
      for A, B in nonterms
        prob = P(A->B)*score[begin][end][B];
        if prob > score[begin][end][A]
          score[begin][end][A] = prob
          back[begin][end][A] = B
          added = true
return buildTree(score, back)
```

104

104