CS 6120/CS4120: Natural Language Processing

Instructor: Prof. Lu Wang

College of Computer and Information Science

Northeastern University

Webpage: www.ccs.neu.edu/home/luwang
Logistics

• Project proposal is due on Feb 6.

• If you haven’t found a group yet, make a private post on piazza today and let me know.

• Assignment 2 is released, due on March 20\textsuperscript{th}, 11:59pm.
Two views of linguistic structure:
1. Constituency (phrase structure)
   • Phrase structure organizes words into nested constituents.
     • Fed raises interest rates
Two views of linguistic structure:
1. Constituency (phrase structure)

- Phrase structure organizes words into nested constituents.
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].

Diagram:

```
S
  NP
    N  V
      Fed raises
  VP
    N
      interest
  NP
    N
      rates
```
Analysts said -NONE- to resume a more influential role in the company.
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), CONJP (multi word constructions: as well as), INTJ (interjections), etc.
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*
Two views of linguistic structure:
2. Dependency structure

• Dependency structure shows which words depend on (modify or are arguments of) which other words.
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]
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
Evaluating Chunking

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

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

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

F measure: \quad F_1 = \frac{1}{\frac{1}{P} + \frac{1}{R}}/2 = \frac{2PR}{P + R}
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 $F_1=92–94\%$
Syntactic Parsing

• Produce the correct syntactic parse tree for a sentence.
Classical NLP Parsing: The problem and its solution

- Adding constraints to grammars to limit unlikely/weird parses for sentences
  - But the attempt makes the grammars not robust
    - In traditional systems, commonly 30% of sentences in even an edited text would have no parse.
- A less constrained grammar can parse more sentences
  - But simple sentences end up with ever more parses with no way to choose between them
- We need mechanisms that allow us to find the most likely parse(s) for a sentence
  - Statistical parsing lets us work with very loose grammars that admit millions of parses for sentences but still quickly find the best parse(s)
The rise of annotated data: The Penn Treebank

[Marcus et al. 1993, *Computational Linguistics*]
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
Two problems to solve for parsing:
1. Repeated work...

“Cats scratch people with cats with claws”
Two problems to solve for parsing:
1. Repeated work...

“Cats scratch people with cats with claws”
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.
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]
(Probabilistic) Context-Free Grammars

• CFG
• PCFG
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 ($S \in N$)
  • $R$ is a set of rules/productions of the form $X \rightarrow \gamma$
    • $X \in N$ and $\gamma \in (N \cup T)^*$
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

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

people fish tanks
people fish with rods
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 ($S \in N$)
  - $R$ 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 $L$. 
Sentence Generation

- Sentences are generated by recursively rewriting the start symbol using the productions until only terminals symbols remain.
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 ($S \in N$)
  - $L$ 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
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

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

people fish tanks
people fish with rods
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 → γ
  - P is a probability function
    - P: R → [0,1]
    - ∀X ∈ N, \( \sum_{X \rightarrow γ \in R} P(X \rightarrow γ) = 1 \)

- A grammar G generates a language model L.
### A PCFG

<table>
<thead>
<tr>
<th>Production</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>1.0</td>
</tr>
<tr>
<td>VP → V NP</td>
<td>0.6</td>
</tr>
<tr>
<td>VP → V NP PP</td>
<td>0.4</td>
</tr>
<tr>
<td>NP → NP NP</td>
<td>0.1</td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>0.2</td>
</tr>
<tr>
<td>NP → N</td>
<td>0.7</td>
</tr>
<tr>
<td>PP → P NP</td>
<td>1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-terminal</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>N → people</td>
<td>0.5</td>
</tr>
<tr>
<td>N → fish</td>
<td>0.2</td>
</tr>
<tr>
<td>N → tanks</td>
<td>0.2</td>
</tr>
<tr>
<td>N → rods</td>
<td>0.1</td>
</tr>
<tr>
<td>V → people</td>
<td>0.1</td>
</tr>
<tr>
<td>V → fish</td>
<td>0.6</td>
</tr>
<tr>
<td>V → tanks</td>
<td>0.3</td>
</tr>
<tr>
<td>P → with</td>
<td>1.0</td>
</tr>
</tbody>
</table>

[With empty NP removed so less ambiguous]
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

\[
P(s) = \sum_t P(s, t) \quad \text{where } t \text{ is a parse of } s
\]
\[
= \sum_t P(t)
\]
$t_1$: S_1.0
   NP_0.7
      N_0.5  V_0.6  NP_0.7  PP_1.0
         people  fish  tanks  with  N_0.1

$t_2$: S_1.0
   NP_0.7
      N_0.5  V_0.6  NP_0.7  PP_1.0
         people  fish  tanks  with  N_0.1

Tree and String Probabilities

• \( s = \text{people fish tanks with rods} \)

\[
P(t_1) = 1.0 \times 0.7 \times 0.4 \times 0.5 \times 0.6 \times 0.7 \\
\times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1 \\
= 0.0008232
\]

\[
P(t_2) = 1.0 \times 0.7 \times 0.6 \times 0.5 \times 0.6 \times 0.2 \\
\times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1 \\
= 0.00024696
\]

\[
P(s) = P(t_1) + P(t_2) \\
= 0.0008232 + 0.00024696 \\
= 0.00107016
\]
Chomsky Normal Form

• All rules are of the form $X \rightarrow Y Z$ or $X \rightarrow w$
  • $X, Y, Z \in N$ and $w \in T$

• 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 ($n > 2$)
A phrase structure grammar

\[
\begin{align*}
S & \rightarrow NP \ VP \\
VP & \rightarrow V \ NP \\
VP & \rightarrow V \ NP \ PP \\
NP & \rightarrow NP \ NP \\
NP & \rightarrow NP \ PP \\
NP & \rightarrow N \\
NP & \rightarrow e \\
PP & \rightarrow P \ NP \\
N & \rightarrow people \\
N & \rightarrow fish \\
N & \rightarrow tanks \\
N & \rightarrow rods \\
V & \rightarrow people \\
V & \rightarrow fish \\
V & \rightarrow tanks \\
P & \rightarrow with
\end{align*}
\]
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
Chomsky Normal Form steps

\[
\begin{align*}
S &\rightarrow NP \ VP \\
VP &\rightarrow V \ NP \\
S &\rightarrow V \ NP \\
VP &\rightarrow V \\
S &\rightarrow V \\
VP &\rightarrow V \ NP \ PP \\
S &\rightarrow V \ NP \ PP \\
VP &\rightarrow V \ PP \\
S &\rightarrow V \ PP \\
NP &\rightarrow NP \ NP \\
NP &\rightarrow NP \\
NP &\rightarrow NP \ PP \\
NP &\rightarrow PP \\
NP &\rightarrow N \\
PP &\rightarrow P \ NP \\
PP &\rightarrow P \\
N &\rightarrow people \\
N &\rightarrow fish \\
N &\rightarrow tanks \\
N &\rightarrow rods \\
V &\rightarrow people \\
V &\rightarrow fish \\
V &\rightarrow tanks \\
P &\rightarrow with
\end{align*}
\]
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
Chomsky Normal Form steps

S $\rightarrow$ NP VP
VP $\rightarrow$ V NP
S $\rightarrow$ V NP
VP $\rightarrow$ V NP PP
S $\rightarrow$ V NP PP
VP $\rightarrow$ V PP
S $\rightarrow$ V PP
NP $\rightarrow$ NP NP
NP $\rightarrow$ NP
NP $\rightarrow$ NP PP
NP $\rightarrow$ PP
NP $\rightarrow$ N
PP $\rightarrow$ P NP
PP $\rightarrow$ P

N $\rightarrow$ people
N $\rightarrow$ fish
N $\rightarrow$ tanks
N $\rightarrow$ rods
V $\rightarrow$ people
S $\rightarrow$ people
VP $\rightarrow$ people
V $\rightarrow$ fish
S $\rightarrow$ fish
VP $\rightarrow$ fish
V $\rightarrow$ tanks
S $\rightarrow$ tanks
VP $\rightarrow$ tanks
P $\rightarrow$ with
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
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
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
An example: before binarization...
Before and After binarization on VP

Before binarization:

```
NP
  V
   NP
     PP
       N
people fish tanks with rods
```

After binarization:

```
NP
  V
   NP
     PP
       N
people fish tanks with rods
```
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.
Parsing Example

book that flight

S
   VP
      Verb NP
         book Det Nominal
                that Noun
                        flight
Top Down Parsing

S

NP VP

Pronoun
Top Down Parsing

\[
\begin{array}{c}
S \\
NP \\
Pronoun \\
X \\
book \\
\end{array}
\]
Top Down Parsing

```
S
 /   \
NP   VP
   /   \
ProperNoun
```
Top Down Parsing

```
S
  /\  
 NP VP
   /   
 ProperNoun
    /    
 X
 book
```
Top Down Parsing

S

NP      VP

Det      Nominal
Top Down Parsing

```
S
  /
NP VP
  /
Det Nominal
    /
book
```
Top Down Parsing

```
S
 /   \
|    |
Aux NP VP
```
Top Down Parsing

```
S
   /\  
  /   \ 
Aux NP VP
   \   \ 
    X   book
```
Top Down Parsing

```
S  
|  
VP
```
Top Down Parsing

S
/ 
VP
/ 
Verb
Top Down Parsing

S
  /   
VP
 /    
Verb
 /     
book
Top Down Parsing
Top Down Parsing

```
S
 /\  
VP
   /\  
Verb NP
```
Top Down Parsing

S
  /\  \\  
VP
  /\  \\  
Verb NP
    /\  
bok
Top Down Parsing

S

/  

VP

|   |
Verb NP

|   |
book Pronoun
Top Down Parsing

```
S
  /
 VP
    /
   Verb NP
       /
      book Pronoun
           /
          X
             /
              that
```
Top Down Parsing

```
  S
  /|
 /  |
VP
  /|
 Verb  NP
     /|
  book  ProperNoun
```
Top Down Parsing

```
S
 /  
VP
   /
Verb NP
   /  
book ProperNoun
      /  
     X
   that
```
Top Down Parsing

```
S
  /
VP
  /
Verb NP
    /
book Det Nominal
```
Top Down Parsing

S
  /  
VP
  /  
Verb NP
    /  
book Det Nominal
      /  
that
Top Down Parsing

```
S
 |   
| VP 
| Verb NP 
| book Det Nominal 
| that Noun 
```
Top Down Parsing

S
  /
VP
  /
Verb NP
  /
  book Det Nominal
  / that Noun
   / flight
Bottom Up Parsing
Bottom Up Parsing

Noun

book that flight
Bottom Up Parsing

Nominal
  
  Noun

  book that flight
Bottom Up Parsing

```
Nominal
  /       \\     
Nominal Noun
    /      \
  Noun
  /  \
book that flight
```
Bottom Up Parsing

Nominal
  /     \
 Nominal  Noun
  |      /  \
 Noun  X
  |     / \
book that flight
Bottom Up Parsing
Bottom Up Parsing

```
Nominal
  / \     /
Nominal PP
  /       /
Noun Det
  /       /
book that flight
```
Bottom Up Parsing

Nominal

Nominal  PP

Noun  Det  Nominal

book  that  flight
Bottom Up Parsing

Nominal
  └── Nominal
     └── PP
         ├── Noun
         │    └── book
         └── PP
             └── NP
                 └── Det
                     └── Nominal
                         └── Noun
                             └── flight
Bottom Up Parsing
Bottom Up Parsing
Bottom Up Parsing
Bottom Up Parsing

Nominal
   /\          PP
  /   \        X
Nominal NP
   /\     /\  
  /   \   /   \  
Noun Det Nominal
  /\     /\     /\  
 book that Noun

flight
Bottom Up Parsing

```
Verb
  book

Det
  that

Nominal
  Noun
    flight
```

[73x447]Bottom Up Parsing

[362x103]book             that

[364x155]Verb

[466x151]Det

[525x194]NP

[554x152]Nominal

flight
Bottom Up Parsing
Bottom Up Parsing

S
  
VP
  
Verb
  book

NP
  Det
  that

Nominal
  Noun
  flight
Bottom Up Parsing
Bottom Up Parsing

```
[VP
  [VP Verb book]
  [PP Det that]
  [NP Nominal flight]]
```
Bottom Up Parsing

```
VP -> VP
   -> Verb
      book
   -> PP
      Det
      that
   -> NP
      Nominal
      flight
```
Bottom Up Parsing
Bottom Up Parsing

[Diagram showing a parse tree for the sentence: book that flight]

- VP
  - NP
    - Det: that
    - Nominal: flight
  - Verb: book
Bottom Up Parsing
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.
Two problems to solve for parsing:

1. Repeated work

"Cats scratch people with cats with claws"
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.
(Probabilistic) CKY Parsing
Constituency Parsing

Input: a PCFG, and a sentence

PCFG

Rule Prob $\theta_i$

$S \rightarrow NP \ VP \quad \theta_0$

$NP \rightarrow NP \ NP \quad \theta_1$

$N \rightarrow fish \quad \theta_{42}$

$N \rightarrow people \quad \theta_{43}$

$V \rightarrow fish \quad \theta_{44}$
Constituency Parsing

Output: a parsing tree

PCFG

<table>
<thead>
<tr>
<th>Rule</th>
<th>Prob $\theta_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP\ VP$</td>
<td>$\theta_0$</td>
</tr>
<tr>
<td>$NP \rightarrow NP\ NP$</td>
<td>$\theta_1$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$N \rightarrow fish$</td>
<td>$\theta_{42}$</td>
</tr>
<tr>
<td>$N \rightarrow people$</td>
<td>$\theta_{43}$</td>
</tr>
<tr>
<td>$V \rightarrow fish$</td>
<td>$\theta_{44}$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Cocke-Kasami-Younger (CKY) Constituency Parsing

fish people fish tanks

S
  VP
    NP
      N
      N
      V
      N
    NP
      N
      N
      fish
      people
      fish
      tanks
Viterbi (Max) Scores

<table>
<thead>
<tr>
<th></th>
<th>people</th>
<th>fish</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>0.35</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>VP</td>
<td></td>
<td>0.06</td>
</tr>
<tr>
<td>NP</td>
<td></td>
<td>0.14</td>
</tr>
<tr>
<td>V</td>
<td></td>
<td>0.6</td>
</tr>
<tr>
<td>N</td>
<td></td>
<td>0.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>0.9</td>
</tr>
<tr>
<td>S → VP</td>
<td>0.1</td>
</tr>
<tr>
<td>VP → V NP</td>
<td>0.5</td>
</tr>
<tr>
<td>VP → V</td>
<td>0.1</td>
</tr>
<tr>
<td>VP → V @VP_V</td>
<td>0.3</td>
</tr>
<tr>
<td>VP → V PP</td>
<td>0.1</td>
</tr>
<tr>
<td>@VP_V → NP PP</td>
<td>1.0</td>
</tr>
<tr>
<td>NP → NP NP</td>
<td>0.1</td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>0.2</td>
</tr>
<tr>
<td>NP → N</td>
<td>0.7</td>
</tr>
<tr>
<td>PP → P NP</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Extended CKY parsing

- Unaries can be incorporated into the algorithm
  - Messy, but doesn’t increase algorithmic complexity
- Empties can be incorporated
  - Doesn’t increase complexity; essentially like unaries
- Binarization is *vital*
  - Without binarization, you don’t get parsing cubic in the length of the sentence and in the number of nonterminals in the grammar
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
The CKY algorithm (1960/1965) … extended to unaries

```java
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)
```