Applications of Context Free Grammar in Natural Language Processing

Vishal Kakkar
&
Harit Vishwakarma

(Based on slides of Michael Collins, Dan Jurafsky, Dan Klein, Chris Manning, Ray Mooney, Luke Zettlemoyer)
Why Parse?

- Part of speech information
- Phrase information
- Useful relationships

⇒ “the burglar” is the subject of “robbed”
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]
Example Application: Machine Translation

- The boy put the tortoise on the rug
- लड़के ने रखा कछुआ ऊपर कालीन
- SVO vs. SOV; preposition vs. post-position
Example Application: Machine Translation

- The boy put the tortoise on the rug
- लड़के ने रखा कछुआ ऊपर कालीन
- SVO vs. SOV; preposition vs. post-position
Example Application: Machine Translation

- The boy put the tortoise on the rug
  - लड़के ने रखा कछुआ ऊपर कालीन
- SVO vs. SOV; preposition vs. post-position
Example Application: Machine Translation

- The boy put the tortoise on the rug

- लड़के ने रखा कछुआ ऊपर कालीन

- SVO vs. SOV; preposition vs. post-position
Example Application: Machine Translation

- The boy put the tortoise on the rug
- लड़के ने रखा कछुआ ऊपर कालीन
- SVO vs. SOV; preposition vs. post-position
Context Free Grammars and Ambiguities
Context-Free Grammars in NLP

- A context free grammar G in NLP = (N, C, Σ, S, L, R)
  - Σ 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 → x
    - X ∈ C and x ∈ Σ
  - R is the grammar, a set of items of the form X → γ
    - X ∈ N and γ ∈ (N ∪ 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 Context Free Grammar of English

\[ N = \{S, \ NP, \ VP, \ PP, \ DT, \ Vi, \ Vt, \ NN, \ IN\} \]
\[ S = S \]
\[ \Sigma = \{\text{sleeps, saw, man, woman, telescope, the, with, in}\} \]

\[ R = \]

<table>
<thead>
<tr>
<th></th>
<th>S → NP VP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VP → Vi</td>
</tr>
<tr>
<td></td>
<td>VP → Vt NP</td>
</tr>
<tr>
<td></td>
<td>VP → VP PP</td>
</tr>
<tr>
<td></td>
<td>NP → DT NN</td>
</tr>
<tr>
<td></td>
<td>NP → NP PP</td>
</tr>
<tr>
<td></td>
<td>PP → IN NP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Vi → sleeps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Vt → saw</td>
</tr>
<tr>
<td></td>
<td>NN → man</td>
</tr>
<tr>
<td></td>
<td>NN → woman</td>
</tr>
<tr>
<td></td>
<td>NN → telescope</td>
</tr>
<tr>
<td></td>
<td>DT → the</td>
</tr>
<tr>
<td></td>
<td>IN → with</td>
</tr>
<tr>
<td></td>
<td>IN → in</td>
</tr>
</tbody>
</table>

Note: \( S = \)sentence, \( VP = \)verb phrase, \( NP = \)noun phrase, \( PP = \)prepositional phrase, \( DT = \)determiner, \( Vi = \)intransitive verb, \( Vt = \)transitive verb, \( NN = \)noun, \( IN = \)preposition
Left-Most Derivations

A left-most derivation is a sequence of strings $s_1 \ldots s_n$, where

- $s_1 = S$, the start symbol
- $s_n \in \Sigma^*$, i.e. $s_n$ is made up of terminal symbols only
- Each $s_i$ for $i = 2 \ldots n$ is derived from $s_{i-1}$ by picking the left-most non-terminal $X$ in $s_{i-1}$ and replacing it by some $\beta$ where $X \rightarrow \beta$ is a rule in $R$

For example: [S], [NP VP], [D N VP], [the N VP], [the man VP], [the man Vi], [the man sleeps]

Representation of a derivation as a tree:
Properties of CFGs

- A CFG defines a set of possible derivations.
- A string $s \in \Sigma^*$ is in the *language* defined by the CFG if there is at least one derivation that yields $s$.
- Each string in the language generated by the CFG may have more than one derivation ("ambiguity").
Attachment ambiguities

• A key parsing decision is how we ‘attach’ various constituents
  • PPs, adverbial or participial phrases, infinitives, coordinations, etc.

The board approved [its acquisition] [by Royal Trustco Ltd.]
  [of Toronto]
  [for $27 a share]
  [at its monthly meeting].
Attachment ambiguities

• A key parsing decision is how we ‘attach’ various constituents
  • PPs, adverbial or participial phrases, infinitives, coordinations, etc.

The board approved [its acquisition] [by Royal Trustco Ltd.]
  [of Toronto]
  [for $27 a share]
  [at its monthly meeting].

• Catalan numbers: \( C_n = \frac{(2n)!}{(n+1)! n!} \)
• An exponentially growing series, which arises in many tree-like contexts:
  • E.g., the number of possible triangulations of a polygon with \( n+2 \) sides
  • Turns up in triangulation of probabilistic graphical models….
Parsing: Two problems to solve:

1. Repeated work…
Two problems to solve:
1. Repeated work...
Parsing: Two problems to solve:

2. Choosing the correct parse

• How do we work out the correct attachment:
  • She saw the man with a telescope
• Is the problem ‘AI complete’? Yes, but ...
• 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.
Classical NLP Parsing: The problem and its solution

- Categorical constraints can be added to grammars to limit unlikely/weird parses for sentences
  - But the attempt make 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]

( (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))))))
  (. .)))))
**Penn Treebank Non-terminals**

*Table 1.2.* The Penn Treebank syntactic tagset

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJP</td>
<td>Adjective phrase</td>
</tr>
<tr>
<td>ADVP</td>
<td>Adverb phrase</td>
</tr>
<tr>
<td>NP</td>
<td>Noun phrase</td>
</tr>
<tr>
<td>PP</td>
<td>Prepositional phrase</td>
</tr>
<tr>
<td>S</td>
<td>Simple declarative clause</td>
</tr>
<tr>
<td>SBAR</td>
<td>Subordinate clause</td>
</tr>
<tr>
<td>SBARQ</td>
<td>Direct question introduced by <em>wh</em>-element</td>
</tr>
<tr>
<td>SINV</td>
<td>Declarative sentence with subject-aux inversion</td>
</tr>
<tr>
<td>SQ</td>
<td>Yes/no questions and subconstituent of SBARQ excluding <em>wh</em>-element</td>
</tr>
<tr>
<td>VP</td>
<td>Verb phrase</td>
</tr>
<tr>
<td>WHADVP</td>
<td>Wh-adverb phrase</td>
</tr>
<tr>
<td>WHNP</td>
<td>Wh-noun phrase</td>
</tr>
<tr>
<td>WHPP</td>
<td>Wh-prepositional phrase</td>
</tr>
<tr>
<td>X</td>
<td>Constituent of unknown or uncertain category</td>
</tr>
<tr>
<td>*</td>
<td>“Understood” subject of infinitive or imperative</td>
</tr>
<tr>
<td>0</td>
<td>Zero variant of <em>that</em> in subordinate clauses</td>
</tr>
<tr>
<td>T</td>
<td>Trace of <em>wh</em>-Constituent</td>
</tr>
</tbody>
</table>
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
Probabilistic Context Free Grammar
Probabilistic – or stochastic – context-free grammars (PCFGs)

- $G = (\Sigma, N, S, R, P)$
  - $T$ 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 $X \rightarrow \gamma$
  - $P$ 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 $L$. 
  \[ \sum_{\gamma \in L} P(\gamma) = 1 \]
### PCFG Example

<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 ⇒ Vi</td>
<td>0.4</td>
</tr>
<tr>
<td>VP ⇒ Vt NP</td>
<td>0.4</td>
</tr>
<tr>
<td>VP ⇒ VP PP</td>
<td>0.2</td>
</tr>
<tr>
<td>NP ⇒ DT NN</td>
<td>0.3</td>
</tr>
<tr>
<td>NP ⇒ NP PP</td>
<td>0.7</td>
</tr>
<tr>
<td>PP ⇒ P NP</td>
<td>1.0</td>
</tr>
<tr>
<td>Vi ⇒ sleeps</td>
<td>1.0</td>
</tr>
<tr>
<td>Vt ⇒ saw</td>
<td>1.0</td>
</tr>
<tr>
<td>NN ⇒ man</td>
<td>0.7</td>
</tr>
<tr>
<td>NN ⇒ woman</td>
<td>0.2</td>
</tr>
<tr>
<td>NN ⇒ telescope</td>
<td>0.1</td>
</tr>
<tr>
<td>DT ⇒ the</td>
<td>1.0</td>
</tr>
<tr>
<td>IN ⇒ with</td>
<td>0.5</td>
</tr>
<tr>
<td>IN ⇒ in</td>
<td>0.5</td>
</tr>
</tbody>
</table>

- Probability of a tree \( t \) with rules

\[
\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \ldots, \alpha_n \rightarrow \beta_n
\]

is

\[
p(t) = \prod_{i=1}^{n} q(\alpha_i \rightarrow \beta_i)
\]

where \( q(\alpha \rightarrow \beta) \) is the probability for rule \( \alpha \rightarrow \beta \).
## Probability of a Parse

### Parse Rules

<table>
<thead>
<tr>
<th>S</th>
<th>NP</th>
<th>VP</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP</td>
<td>Vi</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>VP</td>
<td>Vt</td>
<td>NP</td>
<td>0.4</td>
</tr>
<tr>
<td>VP</td>
<td>VP</td>
<td>PP</td>
<td>0.2</td>
</tr>
<tr>
<td>NP</td>
<td>DT</td>
<td>NN</td>
<td>0.3</td>
</tr>
<tr>
<td>NP</td>
<td>NP</td>
<td>PP</td>
<td>0.7</td>
</tr>
<tr>
<td>PP</td>
<td>P</td>
<td>NP</td>
<td>1.0</td>
</tr>
</tbody>
</table>

### Probability Calculations

\[ t_1 = S \xrightarrow{1.0} NP \xrightarrow{0.3} VP \xrightarrow{0.4} DT \xrightarrow{1.0} NN \xrightarrow{0.7} Vi \xrightarrow{1.0} \text{The man sleeps} \]

\[ p(t_1) = 1.0 \times 0.3 \times 1.0 \times 0.7 \times 1.0 \times 0.4 = 1.0 \]

\[ t_2 = S \xrightarrow{1.0} VP \xrightarrow{0.2} NP \xrightarrow{0.3} Vt \xrightarrow{1.0} NP \xrightarrow{0.3} IN \xrightarrow{0.5} PP \xrightarrow{0.4} NP \xrightarrow{0.3} DT \xrightarrow{1.0} NN \xrightarrow{0.7} \text{The man saw the woman with the telescope} \]

\[ p(t_2) = 1.0 \times 0.3 \times 1.0 \times 0.7 \times 0.2 \times 1.0 \times 0.3 \times 1.0 \times 0.2 \times 0.5 \times 0.3 \times 1.0 \times 0.1 \]

### Sample Sentences

- The man sleeps
- The man saw the woman with the telescope
PCFGs: Learning and Inference

- **Model**
  - The probability of a tree $t$ with $n$ rules $\alpha_i \rightarrow \beta_i$, $i = 1..n$
  
  $$p(t) = \prod_{i=1}^{n} q(\alpha_i \rightarrow \beta_i)$$

- **Learning**
  - Read the rules off of labeled sentences, use ML estimates for probabilities
  
  $$q_{ML}(\alpha \rightarrow \beta) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

  - and use all of our standard smoothing tricks!

- **Inference**
  - For input sentence $s$, define $T(s)$ to be the set of trees whose *yield* is $s$ (whole leaves, read left to right, match the words in $s$)
  
  $$t^*(s) = \arg \max_{t \in T(s)} p(t)$$
Grammar Transforms
Chomsky Normal Form

- All rules are of the form $X \rightarrow YZ$ or $X \rightarrow w$
  - $X, Y, Z \in N$ and $w \in \Sigma$
- A transformation to this form doesn’t change the weak 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

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
Chomsky Normal Form steps

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

S → NP VP
VP → V NP
S → V NP
VP → V
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
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
V → fish
V → tanks
S → people
S → fish
S → tanks
P → with
Chomsky Normal Form steps

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

\[
\begin{align*}
S & \rightarrow NP \ VP \\
VP & \rightarrow V \ NP \\
S & \rightarrow V \ NP \\
VP & \rightarrow V \ VP_{V} \\
VP_{V} & \rightarrow NP \ PP \\
S & \rightarrow V \ S_{V} \\
S_{V} & \rightarrow NP \ PP \\
VP & \rightarrow V \ PP \\
S & \rightarrow V \ PP \\
NP & \rightarrow NP \ PP \\
NP & \rightarrow NP \ NP \\
NP & \rightarrow P \ NP \\
PP & \rightarrow P \ NP \\
\end{align*}
\]

\[
\begin{align*}
NP & \rightarrow people \\
NP & \rightarrow fish \\
NP & \rightarrow tanks \\
NP & \rightarrow rods \\
V & \rightarrow people \\
S & \rightarrow people \\
VP & \rightarrow people \\
V & \rightarrow fish \\
S & \rightarrow fish \\
VP & \rightarrow fish \\
V & \rightarrow tanks \\
S & \rightarrow tanks \\
VP & \rightarrow tanks \\
P & \rightarrow with \\
PP & \rightarrow with \\
\end{align*}
\]
A phrase structure grammar

\[
S \rightarrow \text{NP} \ \text{VP} \\
\text{VP} \rightarrow \text{V} \ \text{NP} \\
\text{VP} \rightarrow \text{V} \ \text{NP} \ \text{PP} \\
\text{NP} \rightarrow \text{NP} \ \text{NP} \\
\text{NP} \rightarrow \text{NP} \ \text{PP} \\
\text{NP} \rightarrow \text{N} \\
\text{NP} \rightarrow e \\
\text{PP} \rightarrow \text{P} \ \text{NP} \\
\text{N} \rightarrow \text{people} \\
\text{N} \rightarrow \text{fish} \\
\text{N} \rightarrow \text{tanks} \\
\text{N} \rightarrow \text{rods} \\
\text{V} \rightarrow \text{people} \\
\text{V} \rightarrow \text{fish} \\
\text{V} \rightarrow \text{tanks} \\
\text{P} \rightarrow \text{with}
\]
Chomsky Normal Form steps

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

- You should think of this as a transformation for efficient parsing
- With some extra book-keeping in symbol names, you can even reconstruct the same trees with a detransform
- In practice full Chomsky Normal Form is a pain
  - Reconstructing n-aries is easy
  - Reconstructing unaries/empties is trickier

- **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…
After binarization…
Treebank: empties and unaries

PTB Tree
NoFuncTags
NoEmpties
High
Low
NoUnaries
Parsing
Constituency Parsing

PCFG

Rule Prob $\theta_i$

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

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

$\ldots$

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

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

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

$\ldots$
Cocke-Kasami-Younger (CKY) Constituency Parsing

fish people fish tanks
Viterbi (Max) Scores

S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
VP → V 0.1
VP → V @VP_V 0.3
VP → V PP 0.1
@VP_V → NP PP 1.0
NP → NP NP 0.1
NP → NP PP 0.2
NP → N 0.7
PP → P NP 1.0
Extended CKY parsing

• Unaries can be incorporated into the algorithm
  • Messy, but doesn’t increase algorithmic complexity

• Empties can be incorporated
  • Use fenceposts
  • 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
    • Binarization may be an explicit transformation or implicit in how the parser works (Early-style dotted rules), but it’s always there.
A Recursive Parser

\[
\text{bestScore}(X,i,j,s) \quad \text{if } (j == i) \quad \text{return } q(X->s[i]) \quad \text{else} \quad \text{return max } q(X->YZ) * \\
\quad \quad \eta_{k,X->YZ} \quad \text{bestScore}(Y,i,k,s) * \\
\quad \quad \text{bestScore}(Z,k+1,j,s)
\]
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

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)
The grammar:
Binary, no epsilons,

\[
\begin{align*}
S & \rightarrow NP \ VP & 0.9 \\
S & \rightarrow VP & 0.1 \\
VP & \rightarrow V \ NP & 0.5 \\
VP & \rightarrow V & 0.1 \\
VP & \rightarrow V@VP_V & 0.3 \\
VP & \rightarrow V \ PP & 0.1 \\
@VP_V & \rightarrow NP \ PP & 1.0 \\
NP & \rightarrow NP \ NP & 0.1 \\
NP & \rightarrow NP \ PP & 0.2 \\
NP & \rightarrow N & 0.7 \\
PP & \rightarrow P \ NP & 1.0 \\
N & \rightarrow people & 0.5 \\
N & \rightarrow fish & 0.2 \\
N & \rightarrow tanks & 0.2 \\
N & \rightarrow rods & 0.1 \\
V & \rightarrow people & 0.1 \\
V & \rightarrow fish & 0.6 \\
V & \rightarrow tanks & 0.3 \\
P & \rightarrow with & 1.0
\end{align*}
\]
<table>
<thead>
<tr>
<th></th>
<th>fish</th>
<th>people</th>
<th>fish</th>
<th>tanks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
score[0][1]  score[0][2]  score[0][3]  score[0][4]
score[1][2]  score[1][3]  score[1][4]
score[2][3]  score[2][4]
score[3][4]
```
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]);
### Grammar Rules

- **S → NP VP**: 0.9
- **S → VP**: 0.1
- **VP → V NP**: 0.5
- **VP → V**: 0.1
- **VP → V @VP_V**: 0.3
- **VP → V PP**: 0.1
- **@VP_V → NP PP**: 1.0
- **NP → NP NP**: 0.1
- **NP → NP PP**: 0.2
- **NP → N**: 0.7
- **PP → P NP**: 1.0

### Word Transitions

- **N → fish**: 0.2
- **V → fish**: 0.6
- **N → people**: 0.5
- **V → people**: 0.1
- **N → fish**: 0.2
- **V → fish**: 0.6
- **N → tanks**: 0.2
- **V → tanks**: 0.1

### Code Snippet

```java
// 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
```

---

---

---

---

---

---
N → fish 0.2  
V → fish 0.6  
NP → N 0.14  
VP → V 0.06  
S → VP 0.006

N → people 0.5  
V → people 0.1  
NP → N 0.35  
VP → V 0.01  
S → VP 0.001

N → fish 0.2  
V → fish 0.6  
NP → N 0.14  
VP → V 0.06  
S → VP 0.006

N → tanks 0.2  
V → tanks 0.1  
NP → N 0.14  
VP → V 0.03  
S → VP 0.003

0  
fish  
1  
people  
2  
fish  
3  
tanks  
4

S → NP VP 0.9  
S → VP 0.1  
VP → V NP 0.5  
VP → V 0.1  
VP → V @VP_V 0.3  
VP → V PP 0.1  
@VP_V → NP PP 1.0  
NP → NP NP 0.1  
NP → NP PP 0.2  
NP → N 0.7  
PP → P NP 1.0

N → people 0.5  
N → fish 0.2  
N → tanks 0.2  
N → rods 0.1  
V → people 0.1  
V → fish 0.6  
V → tanks 0.3  
P → with 1.0
N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006

N → people 0.5
V → people 0.1
NP → N 0.35
VP → V 0.01
S → VP 0.001

N → fish 0.2
V → fish 0.6
NP → N 0.14
VP → V 0.06
S → VP 0.006

N → tanks 0.2
V → tanks 0.1
NP → N 0.14
VP → V 0.03
S → VP 0.003

//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
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)
S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
VP → V 0.1
VP → V @VP_V 0.3
VP → V PP 0.1
@VP_V → NP PP 1.0
NP → NP NP 0.1
NP → NP PP 0.2
NP → N 0.7
PP → P NP 1.0

N → people 0.5
N → fish 0.2
N → tanks 0.2
N → rods 0.1
V → people 0.1
V → fish 0.6
V → tanks 0.3
P → with 1.0

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)
<table>
<thead>
<tr>
<th></th>
<th>fish</th>
<th>people</th>
<th>fish</th>
<th>tanks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N → fish 0.2</td>
<td>NP → NP NP 0.0049</td>
<td>NP → NP NP 0.0000686</td>
<td>N → tanks 0.2</td>
</tr>
<tr>
<td></td>
<td>V → fish 0.6</td>
<td>VP → V NP 0.105</td>
<td>VP → V NP 0.00147</td>
<td>V → tanks 0.1</td>
</tr>
<tr>
<td></td>
<td>NP → N 0.14</td>
<td>S → VP 0.0105</td>
<td>S → NP VP 0.000882</td>
<td>VP → V 0.014</td>
</tr>
<tr>
<td></td>
<td>VP → V 0.06</td>
<td>S → VP 0.0006</td>
<td>S → NP VP 0.00147</td>
<td>S → VP 0.003</td>
</tr>
<tr>
<td></td>
<td>S → VP 0.006</td>
<td>N → people 0.5</td>
<td>NP → NP NP 0.0000686</td>
<td>P → with 1.0</td>
</tr>
<tr>
<td></td>
<td>N → people 0.1</td>
<td>NP → N 0.35</td>
<td>V → people 0.1</td>
<td>N → rods 0.1</td>
</tr>
<tr>
<td></td>
<td>VP → V 0.01</td>
<td>S → VP 0.0189</td>
<td>VP → V 0.007</td>
<td>N → fish 0.6</td>
</tr>
<tr>
<td></td>
<td>S → VP 0.001</td>
<td>N → fish 0.2</td>
<td>VP → V 0.006</td>
<td>V → fish 0.6</td>
</tr>
<tr>
<td></td>
<td>N → tanks 0.2</td>
<td>VP → V 0.006</td>
<td>S → VP 0.000686</td>
<td>V → tanks 0.1</td>
</tr>
<tr>
<td></td>
<td>N → rods 0.1</td>
<td>VP → V 0.042</td>
<td>S → VP 0.003</td>
<td>VP → V 0.03</td>
</tr>
<tr>
<td></td>
<td>N → fish 0.2</td>
<td>S → VP 0.01</td>
<td>S → NP VP 0.01323</td>
<td>S → VP 0.003</td>
</tr>
<tr>
<td></td>
<td>NP → N 0.14</td>
<td>VP → V 0.006</td>
<td>S → VP 0.000686</td>
<td>VP → V 0.003</td>
</tr>
<tr>
<td></td>
<td>VP → V 0.06</td>
<td>S → VP 0.001</td>
<td>S → NP VP 0.00147</td>
<td>S → VP 0.003</td>
</tr>
</tbody>
</table>

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)
Call buildTree(score, back) to get the best parse
Evaluating constituency parsing

Gold standard brackets:

```
S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), NP-(4:6), PP-(6:9), NP-(7,9), NP-(9:10)
```

```
S
  NP
    NNS    NNS
    Sales  executives
  VP
    VBD    VBG
    were  examining
  VP
    DT    NNS    IN
    the  figures with
  NP
    JJ    NN
    great care
  PP
    yesterday
```

Candidate brackets:

```
S-(0:11), NP-(0:2), VP-(2:10), VP-(3:10), NP-(4:6), PP-(6-10), NP-(7,10)
```

```
S
  NP
    NNS    NNS
    Sales  executives
  VP
    VBD    VBG
    were  examining
  VP
    DT    NNS    IN
    the  figures with
  NP
    JJ    NN
    great care
  PP
    yesterday
```
Evaluating constituency parsing

Gold standard brackets:
S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), NP-(4:6), PP-(6-9), NP-(7,9), NP-(9:10)

Candidate brackets:
S-(0:11), NP-(0:2), VP-(2:10), VP-(3:10), NP-(4:6), PP-(6-10), NP-(7,10)

Labeled Precision 3/7 = 42.9%
Labeled Recall 3/8 = 37.5%
LP/LR F1 40.0%
Tagging Accuracy 11/11 = 100.0%