Parsing, PCFGs, and the CKY Algorithm

Yoav Goldberg
(some slides taken from Michael Collins)

November 18, 2013
Natural Language Parsing

- Sentences in natural language have structure.
- Linguists create Linguistic Theories for defining this structure.
- The parsing problem is recovering that structure.
Previous

- Structure is a sequence.
- Each item can be tagged.
- We can mark some spans.
Previously

- Structure is a sequence.
- Each item can be tagged.
- We can mark some spans.

Today

- Hierarchical Structure.
Hierarchical Structure?
Structure
Example 1: math

3*2+5*3
Structure
Example 1: math

```
3*2+5*3
```

```
3  *  2       +       5  *  3
```

```
ADD
```

```
MUL       +       MUL
```

```
3  *  2       +       5  *  3
```

```
5 / 48
```
Structure

Example 1: math

3 * 2 + 5 * 3

ADD

MUL + MUL

3 * 2 5 * 3

3 * 2 + 5 * 3

+  

* * 

3 2 5 3
Programming Languages?
Structure
Example 2: Language Data

Fruit flies like a banana
Structure
Example 2: Language Data

Fruit flies like a banana

Constituency Structure

Dependency Structure
Constituency Structure

- In this class we concentrate on Constituency Parsing: mapping from sentences to trees with labeled nodes and the sentence words at the leaves.
Why is Parsing Interesting?

- It’s a first step towards understanding a text.
- Many other language tasks use sentence structure as their input.
Some things that are done with parse trees

- Grammar Checking
- Word Clustering
- Information Extraction
- Question Answering
- Sentence Simplification
- Machine Translation
- ... and more
Some things that are done with parse trees

- Grammar Checking
- **Word Clustering**
- Information Extraction
- Question Answering
- Sentence Simplification
- Machine Translation
- ...and more

Words in similar grammatical relations share meanings.
Some things that are done with parse trees

- Grammar Checking
- Word Clustering
- Information Extraction
- Question Answering
- Sentence Simplification
- Machine Translation
- ... and more

Extract factual relations from text

Answer questions

Factz from Wikipedia: we found the following about CIA

<table>
<thead>
<tr>
<th>CIA</th>
<th>trained: exiles, agent, fighters, army, issues, Farm, facilities</th>
<th>used: buildings, dealers, seal, missile, methods, processes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>provided: proof, lists, leftists, communists, Factbook, train</td>
<td></td>
</tr>
</tbody>
</table>

Tokyo was hit by powerful earthquakes in 1703, 1782, 1812, 1855 and 1923.

2004 Indian Ocean earthquake: The European nation hardest hit may have been Sweden, whose death toll was 543. The deadliest earthquakes since 1900 were the Tangshan, China earthquake of 1976, in which at least 255,000 were killed; the earthquake of 1927 in Xining, Qinghai, China (200,000); the Great Kanto earthquake which struck Tokyo in 1923 (143,000); and the Gansu, China, earthquake of 1920 (200,000).
Some things that are done with parse trees

▶ Grammar Checking
▶ Word Clustering
▶ Information Extraction
▶ Question Answering
▶ Sentence Simplification
▶ Machine Translation
▶ ... and more

The first new product, ATF Protype, is a line of digital postscript typefaces that will be sold in packages of up to six fonts.

ATF Protype is a line of digital postscript typefaces will be sold in packages of up to six fonts.
Some things that are done with parse trees

- Grammar Checking
- Word Clustering
- Information Extraction
- Question Answering
- Sentence Simplification
- Machine Translation
- ...and more
Some things that are done with parse trees

- Grammar Checking
- Word Clustering
- Information Extraction
- Question Answering
- Sentence Simplification
- Machine Translation
- ...and more
Why is parsing hard?

Ambiguity

Fat people eat candy
Why is parsing hard?

Ambiguity

Fat people eat candy

S

NP   VP

Adj Nn Vb NP

Fat people eat Nn candy
Why is parsing hard?

Ambiguity

Fat people eat candy

Fat people eat candy accumulates
Why is parsing hard?

Ambiguity

Fat people eat candy

Fat people eat accumulates
“Former Beatle Paul McCartney today was ordered to pay nearly $50M to his estranged wife as their bitter divorce battle came to an end.”

“Welcome to our Columbus hotels guide, where you’ll find honest, concise hotel reviews, all discounts, a lowest rate guarantee, and no booking fees.”
Let’s learn how to parse
Let's learn how to parse ... but first let's review some stuff we learned at Automata and Formal Languages.
Context Free Grammars

A context free grammar $G = (N, \Sigma, R, S)$ where:

- $N$ is a set of non-terminal symbols
- $\Sigma$ is a set of terminal symbols
- $R$ is a set of rules of the form $X \to Y_1 Y_2 \cdots Y_n$ for $n \geq 0$, $X \in N$, $Y_i \in (N \cup \Sigma)$
- $S \in N$ is a special start symbol
Context Free Grammars

a simple grammar

\[ N = \{ S, NP, VP, Adj, Det, Vb, Noun \} \]
\[ \Sigma = \{ fruit, flies, like, a, banana, tomato, angry \} \]
\[ S = 'S' \]
\[ R = \]

\[ S \rightarrow NP \ VP \]
\[ NP \rightarrow Adj \ Noun \]
\[ NP \rightarrow Det \ Noun \]
\[ VP \rightarrow Vb \ NP \]
\[ Adj \rightarrow fruit \]
\[ Noun \rightarrow flies \]
\[ Vb \rightarrow like \]
\[ Det \rightarrow a \]
\[ Noun \rightarrow banana \]
\[ Noun \rightarrow tomato \]
\[ Adj \rightarrow angry \]
Left-most derivations

Left-most derivation is a sequence of strings $s_1, \cdots, s_n$ where

- $s_1 = S$ the start symbol
- $s_n \in \Sigma^*$, meaning $s_n$ is only terminal symbols
- Each $s_i$ for $i = 2 \cdots n$ 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: $[S],[NP \ VP],[Adj \ Noun \ VP]$, $[fruit \ Noun \ VP]$, $[fruit \ flies \ VP]$, $[fruit \ flies \ Vb \ NP]$, $[fruit \ flies \ like \ NP]$, $[fruit \ flies \ like \ Det \ Noun]$, $[fruit \ flies \ like \ a]$, $[fruit \ flies \ like \ a \ banana]$
Left-most derivation example

S

The resulting derivation can be written as a tree. Many trees can be generated.
Left-most derivation example

S
NP VP

S → NP VP

The resulting derivation can be written as a tree.

Many trees can be generated.
Left-most derivation example

S
NP VP
Adj Noun VP

S
NP VP
Adj Noun VP

NP → Adj Noun

The resulting derivation can be written as a tree. Many trees can be generated.
Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP

Adj → fruit

The resulting derivation can be written as a tree.
Many trees can be generated.
Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP
fruit flies VP

The resulting derivation can be written as a tree.

Many trees can be generated.
Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP
fruit flies VP
fruit flies Vb NP

VP \rightarrow Vb \text{ NP}
Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP
fruit flies VP
fruit flies Vb NP
fruit flies like NP

Vb → like
Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP
fruit flies VP
fruit flies Vb NP
fruit flies like NP
fruit flies like Det Noun

NP → Det Noun
Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP
fruit flies VP
fruit flies Vb NP
fruit flies like NP
fruit flies like Det Noun
fruit flies like a Noun

Det → a
Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP
fruit flies VP
fruit flies Vb NP
fruit flies like NP
fruit flies like Det Noun
fruit flies like a Noun
fruit flies like a banana

Noun $\rightarrow$ banana
Left-most derivation example

S
  NP VP
  Adj Noun VP
  fruit Noun VP
  fruit flies VP
  fruit flies Vb NP
  fruit flies like NP
  fruit flies like Det Noun
  fruit flies like a Noun
  fruit flies like a banana

▶ The resulting derivation can be written as a tree.
Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP
fruit flies VP
fruit flies Vb NP
fruit flies like NP
fruit flies like Det Noun
fruit flies like a Noun
fruit flies like a banana

- The resulting derivation can be written as a tree.
- Many trees can be generated.
a simple grammar
S → NP VP
NP → Adj Noun
NP → Det Noun
VP → Vb NP
- 
Adj → fruit
Noun → flies
Vb → like
Det → a
Noun → banana
Noun → tomato
Adj → angry
...
a simple grammar

S → NP VP
NP → Adj Noun
NP → Det Noun
VP → Vb NP

- Adj → fruit
Noun → flies
Vb → like
Det → a
Noun → banana
Adj → angry

Example

```
S
  /\  \
 VP NP
  /\  \
 Vb Adj Noun
  /\  \
 fruit flies
  /\  \
 like Det Noun
  /\  \
 a banana
```
Context Free Grammars

a simple grammar
S → NP VP
NP → Adj Noun
NP → Det Noun
VP → Vb NP

- Adj → fruit
Noun → flies
Vb → like
Det → a
Noun → banana
Adj → angry

Example

```
S
  NP
    Adj
    Angry
    Flies
  VP
    Vb
    like
    Det
    a
    Noun
    banana
```
a simple grammar

S → NP VP
NP → Adj Noun
NP → Det Noun
VP → Vb NP
-
Adj → fruit
Noun → flies
Vb → like
Det → a
Noun → banana
Noun → tomato
Adj → angry

Example

S
NP
Adj
Angry
Noun
Flies
VP
Vb
like
Det
a
Noun
tomato
a simple grammar
S → NP VP
NP → Adj Noun
NP → Det Noun
VP → Vb NP
-  
Adj → fruit
Noun → flies
Vb → like
Det → a
Noun → banana
Noun → tomato
Adj → angry

Example
a simple grammar

S → NP VP
NP → Adj Noun
NP → Det Noun
VP → Vb NP

- Adj → fruit
Noun → flies
Vb → like
Det → a
Noun → banana
Noun → tomato
Adj → angry

Example

S
  / \   /  \
NP   VP
  /  \
Det Noun Vb NP
  /  \
  a banana like Det Noun
  / \
  a tomato
a simple grammar

S → NP VP
NP → Adj Noun
NP → Det Noun
VP → Vb NP

- Adj → fruit
Noun → flies
Vb → like
Det → a
Noun → banana
Noun → tomato
Adj → angry

Example
A Brief Introduction to English Syntax
Product Details (from Amazon)
Hardcover: 1779 pages
Publisher: Longman; 2nd Revised edition
Language: English
ISBN-10: 0582517346
Product Dimensions: 8.4 x 2.4 x 10 inches
Shipping Weight: 4.6 pounds
A Brief Overview of English Syntax

Parts of Speech (tags from the Brown corpus):

- **Nouns**
  - \text{NN} = \text{singular noun} \quad \text{e.g., man, dog, park}
  - \text{NNS} = \text{plural noun} \quad \text{e.g., telescopes, houses, buildings}
  - \text{NNP} = \text{proper noun} \quad \text{e.g., Smith, Gates, IBM}

- **Determiners**
  - \text{DT} = \text{determiner} \quad \text{e.g., the, a, some, every}

- **Adjectives**
  - \text{JJ} = \text{adjective} \quad \text{e.g., red, green, large, idealistic}
A Fragment of a Noun Phrase Grammar

\[
\begin{array}{|c|c|}
\hline
\text{N} & \Rightarrow \text{NN} \\
\hline
\text{N} & \Rightarrow \text{NN} \quad \text{N} \\
\hline
\text{N} & \Rightarrow \text{JJ} \quad \text{N} \\
\hline
\text{N} & \Rightarrow \text{N} \quad \text{N} \\
\hline
\text{NP} & \Rightarrow \text{DT} \quad \text{N} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{NN} & \Rightarrow \text{box} \\
\hline
\text{NN} & \Rightarrow \text{car} \\
\hline
\text{NN} & \Rightarrow \text{mechanic} \\
\hline
\text{NN} & \Rightarrow \text{pigeon} \\
\hline
\text{DT} & \Rightarrow \text{the} \\
\hline
\text{DT} & \Rightarrow \text{a} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{JJ} & \Rightarrow \text{fast} \\
\hline
\text{JJ} & \Rightarrow \text{metal} \\
\hline
\text{JJ} & \Rightarrow \text{idealistic} \\
\hline
\text{JJ} & \Rightarrow \text{clay} \\
\hline
\end{array}
\]
Prepositions, and Prepositional Phrases

- Prepositions
  IN = preposition
  e.g., of, in, out, beside, as
An Extended Grammar

| N  ⇒  NN               | JJ  ⇒  fast  |
| N  ⇒  NN N | NN  ⇒  box  |
| N  ⇒  JJ N | NN  ⇒  car   |
| N  ⇒  N N | NN  ⇒  mechanic |
| NP  ⇒  DT N | NN  ⇒  pigeon |
| PP  ⇒  IN NP | DT  ⇒  the |
| N  ⇒  N PP | DT  ⇒  a   |
| N  ⇒  N N | IN  ⇒  in   |
| PP  ⇒  IN NP | IN  ⇒  under |
| N  ⇒  N PP | IN  ⇒  of   |
| N  ⇒  N PP | IN  ⇒  on   |
| N  ⇒  N PP | IN  ⇒  with |
| N  ⇒  N PP | IN  ⇒  as   |

Generates:
in a box, under the box, the fast car mechanic under the pigeon in the box, . . .
An Extended Grammar

\[
\begin{align*}
\tilde{N} & \Rightarrow \text{NN} \\
\tilde{N} & \Rightarrow \text{NN} \; \tilde{N} \\
\tilde{N} & \Rightarrow \text{JJ} \; \tilde{N} \\
\tilde{N} & \Rightarrow \tilde{N} \; \tilde{N} \\
\text{NP} & \Rightarrow \text{DT} \; \tilde{N} \\
\text{PP} & \Rightarrow \text{IN} \; \text{NP} \\
\tilde{N} & \Rightarrow \tilde{N} \; \text{PP}
\end{align*}
\]
Verbs, Verb Phrases, and Sentences

- **Basic Verb Types**
  - Vi = Intransitive verb e.g., sleeps, walks, laughs
  - Vt = Transitive verb e.g., sees, saw, likes
  - Vd = Ditransitive verb e.g., gave

- **Basic VP Rules**
  - VP \rightarrow Vi
  - VP \rightarrow Vt \ NP
  - VP \rightarrow Vd \ NP \ NP

- **Basic S Rule**
  - S \rightarrow NP \ VP

**Examples of VP:**
sleeps, walks, likes the mechanic, gave the mechanic the fast car

**Examples of S:**
the man sleeps, the dog walks, the dog gave the mechanic the fast car
A new rule: $\text{VP} \rightarrow \text{VP PP}$

New examples of VP:
sleeps in the car, walks like the mechanic, gave the mechanic the fast car on Tuesday, ...
Complementizers, and SBARs

- Complementizers
  \[ \text{COMP} = \text{complementizer} \quad \text{e.g., that} \]

- SBAR
  \[ \text{SBAR} \rightarrow \text{COMP} \quad \text{S} \]

**Examples:**
that the man sleeps, that the mechanic saw the dog …
More Verbs

- **New Verb Types**
  - V[5] e.g., said, reported
  - V[6] e.g., told, informed
  - V[7] e.g., bet

- **New VP Rules**
  - VP → V[5] SBAR
  - VP → V[6] NP SBAR
  - VP → V[7] NP NP SBAR

**Examples of New VPs:**
said that the man sleeps
told the dog that the mechanic likes the pigeon
bet the pigeon $50 that the mechanic owns a fast car
Coordination

- A New Part-of-Speech:
  \[ CC = \text{Coordinator} \quad \text{e.g., and, or, but} \]

- New Rules
  \[
  \begin{align*}
  \text{NP} & \rightarrow \text{NP} \quad CC \quad \text{NP} \\
  \text{N} & \rightarrow \text{N} \quad CC \quad \text{N} \\
  \text{VP} & \rightarrow \text{VP} \quad CC \quad \text{VP} \\
  \text{S} & \rightarrow \text{S} \quad CC \quad \text{S} \\
  \text{SBAR} & \rightarrow \text{SBAR} \quad CC \quad \text{SBAR}
  \end{align*}
  \]
We’ve Only Scratched the Surface...

- Agreement
  
  The dogs laugh vs. The dog laughs

- Wh-movement
  
  The dog that the cat liked ___

- Active vs. passive
  
  The dog saw the cat vs.
  The cat was seen by the dog

- If you’re interested in reading more:

Parsing with (P)CFGs
Parsing with CFGs

Let’s assume...

- Let’s assume natural language is generated by a CFG.
- ... and let’s assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from $S$ that generates it.
Parsing with CFGs

Let’s assume...

- Let’s assume natural language is generated by a CFG.
- ...and let’s assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from $S$ that generates it.

Problem

- Natural Language is NOT generated by a CFG.
Let’s assume…

- Let’s assume natural language is generated by a CFG.
- … and let’s assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from $S$ that generates it.

Problem

- Natural Language is NOT generated by a CFG.

Solution

- We assume really hard that it is.
Parsing with CFGs

Let’s assume...

- Let’s assume natural language is generated by a CFG.
- ...and let’s assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from S that generates it.

Problem

- We don’t have the grammar.
Let’s assume...  
- Let’s assume natural language is generated by a CFG.
- . . . and let’s assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from $S$ that generates it.

Problem
- We don’t have the grammar.

Solution
- We’ll ask a genius linguist to write it!
Let’s assume…

- Let’s assume natural language is generated by a CFG.
- …and let’s assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from S that generates it.

Problem

- How do we find the chain of derivations?
Parsing with CFGs

Let’s assume...

- Let’s assume natural language is generated by a CFG.
- ...and let’s assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from S that generates it.

Problem

- How do we find the chain of derivations?

Solution

- With dynamic programming! (soon)
Parsing with CFGs

Let’s assume...

- Let’s assume natural language is generated by a CFG.
- ... and let’s assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from $S$ that generates it.

Problem

- Real grammar: hundreds of possible derivations per sentence.
Parsing with CFGs

Let’s assume...

- Let’s assume natural language is generated by a CFG.
- ...and let’s assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from $S$ that generates it.

Problem

- Real grammar: hundreds of possible derivations per sentence.

Solution

- No problem! We’ll choose the best one. (sooner)
Obtaining a Grammar

Let a genius linguist write it

- Hard. Many rules, many complex interactions.
- Genius linguists don’t grow on trees!
Obtaining a Grammar

Let a genius linguist write it

- Hard. Many rules, many complex interactions.
- Genius linguists don’t grow on trees!

An easier way - ask a linguist to grow trees

- Ask a linguist to annotate sentences with tree structure.
- (This need not be a genius – Smart is enough.)
- Then extract the rules from the annotated trees.
Obtaining a Grammar

Let a genius linguist write it

- Hard. Many rules, many complex interactions.
- Genius linguists don’t grow on trees!

An easier way - ask a linguist to grow trees

- Ask a linguist to annotate sentences with tree structure.
- (This need not be a genius – Smart is enough.)
- Then extract the rules from the annotated trees.

Treebanks

- **English Treebank**: 40k sentences, manually annotated with tree structure.
- **Hebrew Treebank**: about 5k sentences
( (S
   (NP-SBJ
      (NP (NNP Pierre) (NNP Vinken) )
      (, ,)
   (ADJP
      (NP (CD 61) (NNS years) )
      (JJ old) )
      (, ,) )
   (VP (MD will)
   (VP (VB join)
      (NP (DT the) (NN board) )
      (PP-CLR (IN as)
         (NP (DT a) (JJ nonexecutive) (NN director)
           (NP-TMP (NNP Nov.) (CD 29) )))
      (, .) ))
)
Supervised Learning from a Treebank
Extracting CFG from Trees

- The leafs of the trees define $\Sigma$
- The internal nodes of the trees define $N$
- Add a special $S$ symbol on top of all trees
- Each node an its children is a rule in $R$
Extracting CFG from Trees

- The leafs of the trees define $\Sigma$
- The internal nodes of the trees define $N$
- Add a special $S$ symbol on top of all trees
- Each node an its children is a rule in $R$

Extracting Rules

```
S
  NP       VP
  Adj  Noun  Vb  NP
  Fruit  Flies like  Det  Noun
  a  banana
```
Extracting CFG from Trees

- The leafs of the trees define $\Sigma$
- The internal nodes of the trees define $N$
- Add a special $S$ symbol on top of all trees
- Each node an its children is a rule in $R$

Extracting Rules

```
S
  /   \
 NP   VP
  / \  /  \
 Adj Noun Vb NP
    /  \
   Fruit Flies like  a banana
```

$S \rightarrow NP \ VP$
Extracting CFG from Trees

- The leafs of the trees define $\Sigma$
- The internal nodes of the trees define $N$
- Add a special $S$ symbol on top of all trees
- Each node an its children is a rule in $R$

Extracting Rules

\[
S \rightarrow NP \ VP \\
NP \rightarrow \text{Adj Noun} \\
VP \rightarrow \text{Vb NP} \\
\text{NP} \rightarrow \text{Adj Noun} \\
\text{Adj} \rightarrow \text{fruit} \\
\text{Noun} \rightarrow \text{Fruit Flies} \\
\text{Vb} \rightarrow \text{like} \\
\text{Det Noun} \rightarrow \text{a banana}
\]
Extracting CFG from Trees

- The leaves of the trees define $\Sigma$
- The internal nodes of the trees define $N$
- Add a special $S$ symbol on top of all trees
- Each node and its children is a rule in $R$

Extracting Rules

$$\begin{align*}
S & \rightarrow NP \ VP \\
NP & \rightarrow Adj \ Noun \\
Adj & \rightarrow fruit \\
S & \rightarrow NP \ VP \\
NP & \rightarrow Adj \ Noun \\
Adj & \rightarrow fruit
\end{align*}$$
From CFG to PCFG

- English is NOT generated from CFG ⇒ It’s generated by a PCFG!
From CFG to PCFG

- English is NOT generated from CFG ⇒ It’s generated by a PCFG!
- PCFG: probabilistic context free grammar. Just like a CFG, but each rule has an associated probability.
- All probabilities for the same LHS sum to 1.
From CFG to PCFG

- English is NOT generated from CFG ⇒ It’s generated by a PCFG!
- PCFG: probabilistic context free grammar. Just like a CFG, but each rule has an associated probability.
- All probabilities for the same LHS sum to 1.
- Multiplying all the rule probs in a derivation gives the probability of the derivation.
- We want the tree with maximum probability.
From CFG to PCFG

- English is NOT generated from CFG ⇒ It’s generated by a PCFG!
- PCFG: probabilistic context free grammar. Just like a CFG, but each rule has an associated probability.
- All probabilities for the same LHS sum to 1.
- Multiplying all the rule probs in a derivation gives the probability of the derivation.
- We want the tree with maximum probability.

More Formally

\[
P(tree, sent) = \prod_{l \rightarrow r \in deriv(tree)} p(l \rightarrow r)
\]
From CFG to PCFG

- English is NOT generated from CFG ⇒ It’s generated by a PCFG!
- PCFG: probabilistic context free grammar. Just like a CFG, but each rule has an associated probability.
- All probabilities for the same LHS sum to 1.
- Multiplying all the rule probs in a derivation gives the probability of the derivation.
- We want the tree with maximum probability.

More Formally

\[
P(tree, sent) = \prod_{l \rightarrow r \in deriv(tree)} p(l \rightarrow r)
\]

\[
tree = \arg \max_{tree \in trees(sent)} P(tree|sent) = \arg \max_{tree \in trees(sent)} P(tree, sent)
\]
**PCFG Example**

**a simple PCFG**

1.0 $S \rightarrow NP \ VP$
0.3 $NP \rightarrow Adj \ Noun$
0.7 $NP \rightarrow Det \ Noun$
1.0 $VP \rightarrow Vb \ NP$

- 
0.2 $Adj \rightarrow fruit$
0.2 $Noun \rightarrow flies$
1.0 $Vb \rightarrow like$
1.0 $Det \rightarrow a$
0.4 $Noun \rightarrow banana$
0.4 $Noun \rightarrow tomato$
0.8 $Adj \rightarrow angry$

**Example**

```
1 * 0.3 * 0.2 * 0.7 * 1.0 * 0.2 * 1 * 1 * 1 * 0.4 = 0.0033
```
PCFG Example

a simple PCFG
1.0 \( S \rightarrow NP \ VP \)
0.3 \( NP \rightarrow Adj \ Noun \)
0.7 \( NP \rightarrow Det \ Noun \)
1.0 \( VP \rightarrow Vb \ NP \)

- 
0.2 \( Adj \rightarrow fruit \)
0.2 \( Noun \rightarrow flies \)
1.0 \( Vb \rightarrow like \)
1.0 \( Det \rightarrow a \)
0.4 \( Noun \rightarrow banana \)
0.4 \( Noun \rightarrow tomato \)
0.8 \( Adj \rightarrow angry \)

Example

\[
S \\
\downarrow \ NP \\
\downarrow \ Adj \\
\downarrow Fruit \\
\downarrow \ Noun \\
\downarrow Flies \\
\downarrow \ VP \\
\downarrow \ Vb \\
\downarrow like \\
\downarrow \ Det \\
\downarrow a \\
\downarrow \ Noun \\
\downarrow banana
\]

\[1 \times 0.3 \times 0.2 \times 0.7 \times 1.0 \times 0.2 \times 1 \times 1 \times 0.4 = 0.0033\]
PCFG Example

a simple PCFG
1.0 $S \rightarrow NP \ VP$
0.3 $NP \rightarrow Adj \ Noun$
0.7 $NP \rightarrow Det \ Noun$
1.0 $VP \rightarrow Vb \ NP$

0.2 $Adj \rightarrow fruit$
0.2 $Noun \rightarrow flies$
1.0 $Vb \rightarrow like$
1.0 $Det \rightarrow a$
0.4 $Noun \rightarrow banana$
0.4 $Noun \rightarrow tomato$
0.8 $Adj \rightarrow angry$

Example

```
S
   └── NP
       └── Adj
            └── Fruit
        └── Noun
            └── Flies

   └── VP
        └── Vb
            └── like
        └── Det
            └── a
        └── Noun
            └── banana
```

$1 \times 0.3 \times 0.2 \times 0.7 \times 1.0 \times 0.2 \times 1 \times 1 \times 0.4 = 0.0033$
a simple PCFG
1.0 S → NP VP
0.3 NP → Adj Noun
0.7 NP → Det Noun
1.0 VP → Vb NP

0.2 Adj → fruit
0.2 Noun → flies
1.0 Vb → like
1.0 Det → a
0.4 Noun → banana
0.4 Noun → tomato
0.8 Adj → angry

Example

\[
\begin{align*}
S & \rightarrow NP \quad VP \\
NP & \rightarrow Adj \quad Noun\quad Flies \\
\quad Adj & \rightarrow Fruit \\
\quad Noun & \rightarrow Flies \\
\quad Vb & \rightarrow like \\
\quad Det & \rightarrow a \\
NP & \rightarrow banana
\end{align*}
\]

1 * 0.3 * 0.2 * 0.7 * 1.0 * 0.2 * 1 * 1 * 0.4 = 0.0033
Parsing with a PCFG is finding the most probable derivation for a given sentence. This can be done quite efficiently with dynamic programming (the CKY algorithm).
Parsing with PCFG

- Parsing with a PCFG is finding the most probable derivation for a given sentence.
- This can be done quite efficiently with dynamic programming (the CKY algorithm)

Obtaining the probabilities

- We estimate them from the Treebank.
- \[ P(LHS \rightarrow RHS) = \frac{\text{count}(LHS \rightarrow RHS)}{\text{count}(LHS \rightarrow \diamond)} \]
- We can also add smoothing and backoff, as before.
- Dealing with unknown words - like in the HMM
The CKY algorithm
The Problem

Input

- Sentence (a list of words)
  - \( n \) – sentence length
- CFG Grammar (with weights on rules)
  - \( g \) – number of non-terminal symbols

Output

- A parse tree / the best parse tree

But . . .

- Exponentially many possible parse trees!

Solution

- Dynamic Programming!
Cocke  Kasami  Younger
Cocke  Kasami  Younger
196?
CKY

Cocke    Kasami    Younger
196?     1965
<table>
<thead>
<tr>
<th>Cocke</th>
<th>Kasami</th>
<th>Younger</th>
</tr>
</thead>
<tbody>
<tr>
<td>196?</td>
<td>1965</td>
<td>1967</td>
</tr>
</tbody>
</table>
3 Interesting Problems

- Recognition
- Parsing
- Disambiguation
3 Interesting Problems

- Recognition
  - Can this string be generated by the grammar?
- Parsing
- Disambiguation

CKY can do all of these in polynomial time
3 Interesting Problems

- Recognition
  - Can this string be generated by the grammar?
- Parsing
  - Show me a possible derivation...
- Disambiguation
3 Interesting Problems

- Recognition
  - Can this string be generated by the grammar?
- Parsing
  - Show me a possible derivation...
- Disambiguation
  - Show me THE BEST derivation
3 Interesting Problems

- Recognition
  - Can this string be generated by the grammar?
- Parsing
  - Show me a possible derivation...
- Disambiguation
  - Show me THE BEST derivation

CKY can do all of these in polynomial time
3 Interesting Problems

- Recognition
  - Can this string be generated by the grammar?
- Parsing
  - Show me a possible derivation…
- Disambiguation
  - Show me THE BEST derivation

CKY can do all of these in polynomial time

- For any **CNF** grammar
Definition
A CFG is in CNF form if it only has rules like:

- $A \rightarrow B \ C$
- $A \rightarrow \alpha$

$A, B, C$ are non terminal symbols
$\alpha$ is a terminal symbol (a word...)

- All terminal symbols are RHS of unary rules
- All non terminal symbols are RHS of **binary** rules

CKY can be easily extended to handle also unary rules: $A \rightarrow B$
Binarization

Fact

- Any CFG grammar can be converted to CNF form
Binarization

Fact
- Any CFG grammar can be converted to CNF form

Specifically for Natural Language grammars
- We already have $A \rightarrow \alpha$
Binarization

Fact

- Any CFG grammar can be converted to CNF form

Specifically for Natural Language grammars

- We already have $A \rightarrow \alpha$
  - $(A \rightarrow \alpha \beta$ is also easy to handle)
Fact

- Any CFG grammar can be converted to CNF form

Specifically for Natural Language grammars

- We already have $A \rightarrow \alpha$
  - $(A \rightarrow \alpha \beta$ is also easy to handle)
- Unary rules $(A \rightarrow B)$ are OK
Binarization

Fact

- Any CFG grammar can be converted to CNF form

Specifically for Natural Language grammars

- We already have $A \rightarrow \alpha$
  - $(A \rightarrow \alpha \beta$ is also easy to handle)
- Unary rules ($A \rightarrow B$) are OK
- Only problem: $S \rightarrow NP \ PP \ VP \ PP$
Binarization

Fact

- Any CFG grammar can be converted to CNF form

Specifically for Natural Language grammars

- We already have $A \rightarrow \alpha$
  - $(A \rightarrow \alpha \beta$ is also easy to handle)
- Unary rules $(A \rightarrow B)$ are OK
- Only problem: $S \rightarrow NP \ PP \ VP \ PP$

Binarization

\[
\begin{align*}
S & \rightarrow NP \ NP|PP.VP.PP \\
NP|PP.VP.PP & \rightarrow PP \ NP.PP|VP.PP \\
NP.PP|VP.PP & \rightarrow VP \ NP.PP.VP|PP
\end{align*}
\]
Finally, CKY

Recognition

- Main idea:
  - Build parse tree from bottom up
  - Combine built trees to form bigger trees using grammar rules
  - When left with a single tree, verify root is $S$

- Exponentially many possible trees...
  - Search over all of them in polynomial time using DP
  - Shared structure – smaller trees
Main Idea

If we know:

- $w_i \ldots w_j$ is an NP
- $w_{j+1} \ldots w_k$ is a VP

and grammar has rule:

- $S \rightarrow NP \ VP$

Then we know:

- $S$ can derive $w_i \ldots w_k$
Data Structure

(Half a) two dimensional array \((n \times n)\)
Data Structure

On its side
Data Structure

Each cell: all nonterminals that can derive word $i$ to word $j$
Data Structure

Each cell: all nonterminals than can derive word $i$ to word $j$

imagine each cell as a $g$ dimensional array
Filling the table

Sue saw her girl with a telescope
Handling Unary rules?

Sue saw her girl with a telescope.
Sue saw her boy with a telescope
Complexity?

$n^2g$ cells to fill
$g^2n$ ways to fill each one

$O(g^3n^3)$
Complexity?

- $n^2 g$ cells to fill
Complexity?

- $n^2 g$ cells to fill
- $g^2 n$ ways to fill each one
Complexity?

- $n^2 g$ cells to fill
- $g^2 n$ ways to fill each one

$O(g^3 n^3)$
A Note on Implementation

Smart implementation can reduce the runtime:

- Worst case is still $O(g^3n^3)$, but it helps in practice
- No need to check all grammar rules $A \rightarrow BC$ at each location:
  - only those compatible with $B$ or $C$ of current split
  - prune binarized symbols which are too long for current position
  - once you found 1 way to derive $A$ can break out of loop
  - order grammar rules from frequent to infrequent
- Need both efficient random access and iteration over possible symbols
  - Keep both hash and list, implemented as arrays
Finding a parse

Parsing – we want to actually find a parse tree

Easy: also keep a possible split point for each NT
PCFG Parsing and Disambiguation

**Disambiguation** – we want THE BEST parse tree

Easy: for each NT, keep best split point, and score.
Implementation Tricks

#1: sum instead of product

As in the HMM - Multiplying probabilities is evil

- keeping the product of many floating point numbers is dangerous, because product get really small
  - either grow in runtime
  - or loose precision (overflowing to 0)
  - either way, multiplying floats is expensive

Solution: use sum of logs instead

- remember: \( \log(p_1 \times p_2) = \log(p_1) + \log(p_2) \)

⇒ Use log probabilities instead of probabilities
⇒ add instead of multiply
Some recognition speedup tricks are no longer possible

- need the best way to derive a symbol, not just one way
  - can’t abort of loops early...

Solution: two passes

- pass 1: just recognition
- pass 2: disambiguation, with many pruned symbols at each cell
Summary

- Hierarchical Structure
- Constituent / Phrase-based Parsing
- CFGs, Derivations
- (Some) English Syntax
- PCFG
- CKY