Parsing, PCFGs, and the CKY Algorithm

Yoav Goldberg

(with slides by Michael Collins, Julia Hockenmaier)
What is grammar?
Sentences in natural language have structure.
Linguists create Linguistic Theories for defining this structure.
The parsing problem is recovering that structure.
What is grammar?

Grammar formalisms
(= linguists’ programming languages)
   A precise way to define and describe
   the structure of sentences.
   (N.B.: There are many different formalisms out there, which each define their
   own data structures and operations)

Specific grammars
(= linguists’ programs)
   Implementations (in a particular formalism) for a particular
   language (English, Chinese,....)
Can we define a program that generates all English sentences?

The number of sentences is infinite. But we need our program to be finite.
John saw Mary.
I ate sushi with tuna.

Did you go there?
I want you to go there.

I ate the cake that John had made for me yesterday.
John made some cake.

Did you went there?
John Mary saw.
with tuna sushi ate I.
Basic sentence structure

I eat sushi.
A finite-state-automaton (FSA)
A Hidden Markov Model (HMM)

Noun (Subject) → Verb (Head) → Noun (Object)

I, you, .... → eat, drink → sushi, ...

(by Julia Hockenmaier)
Words take arguments

I eat sushi. ✓
I eat sushi you. ???
I sleep sushi ???
I give sushi ???
I drink sushi ?

Subcategorization:

Intransitive verbs (sleep) take only a subject.
Transitive verbs (eat) take also one (direct) object.
Ditransitive verbs (give) take also one (indirect) object.

Selectional preferences:
The object of eat should be edible.
A better FSA

Noun (Subject) → Transitive Verb (Head) → Noun (Object)

Intransitive Verb (Head)
Previously

- 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?
Language is recursive

The ball
the big ball
the big, red ball
the big, red, heavy ball

Adjectives can modify nouns. The number of modifiers/adjuncts a word can have is (in theory) unlimited.
Another FSA

Diagram:
- Determiner
- Adjective
- Noun

(by Julia Hockenmaier)
Recursion can be more complex

the ball
the ball in the garden
the ball in the garden behind the house
the ball in the garden behind the house next to the school
....
What is the structure of a sentence?

Sentence structure is hierarchical:
A sentence consists of **words** (*I, eat, sushi, with, tuna*)
..which form phrases or **constituents**: “*sushi with tuna*”

Sentence structure defines dependencies between words or phrases:

```
```
Strong vs. weak generative capacity

Formal language theory:
– defines language as string sets
– is only concerned with generating these strings
  (weak generative capacity)

Formal/Theoretical syntax (in linguistics):
– defines language as sets of strings with (hidden) structure
– is also concerned with generating the right structures
  (strong generative capacity)
Structure
Example 1: math

3*2 + 5*3
Structure

Example 1: math

$3 \times 2 + 5 \times 3$

\[
\begin{array}{c}
\text{ADD} \\
\text{MUL} & + & \text{MUL} \\
3 & \times & 2 & + & 5 & \times & 3
\end{array}
\]
Structure

Example 1: math

\[ 3 \times 2 + 5 \times 3 \]

\[
\begin{array}{c}
\text{ADD} \\
\text{MUL} + \\
3 \times 2 + 5 \times 3
\end{array}
\]

\[
\begin{array}{c}
+ \\
\times \\
3 \\
2 \\
5 \\
3
\end{array}
\]
Constituency (Phrase Structure) Trees

- Phrase structure organizes words into nested constituents

```
  N  V
 Fed raises N N
    interest rates
```
Constituency (Phrase Structure) Trees

- Phrase structure organizes words into nested constituents
- Linguists can, and do, argue about details
Constituency (Phrase Structure) Trees

- Phrase structure organizes words into nested constituents
- Linguists can, and do, argue about details

we will talk more about **constituents** soon.
Programming Languages?

They also have structure.
How does it differ from human's language structure?
Fruit flies like a banana
Structure
Example 2: Language Data

Fruit flies like a banana

Constituency Structure

Dependency Structure

S
   NP
      Adj
          Fruit
      Noun
          Flies
   VP
      Vb
          like
      NP
          Det
              a
          Noun
              banana
Dependency Representation

The lawyer questioned the witness.
I heard that the lawyer questioned the witness under oath yesterday.
Dependency Representation

I heard that the lawyer questioned the witness under oath yesterday.
Dependency representation is very common.
We will return to it in the future.
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
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
Constituency (Phrase Structure) Trees

- Phrase structure organizes words into nested constituents

**What is a constituent?**
**How do we know they exist?**
Context-free grammars (CFGs) capture recursion

Language has complex constituents
("the garden behind the house")

Syntactically, these constituents behave just like simple ones.
("behind the house" can always be omitted)

CFGs define nonterminal categories to capture equivalent constituents.
The black dog sat

is "dog" a constituent?

is "The black dog" a constituent?

is "the black" a constituent?

is "dog sat" a constituent?

why?
Substitution Test

Every word is a constituent.

If we can substitute a sequence of words by a single word, it is likely to be a constituent.

In particular, pronouns.

(based on book chapter by Beatrice Santorini)
Substitution Test

(1) a. The little boy fed the cat. → He fed her.
b. Black cats detest green peas. → They detest them.

(based on book chapter by Beatrice Santorini)
Substitution Test

(2) a. The little boy from next door fed the cat without a tail.
    b. These black cats detest those green peas.

(based on book chapter by Beatrice Santorini)
Substitution Test

(2) a. The little boy from next door fed the cat without a tail.

b. These black cats detest those green peas.

→ * He from next door fed her without a tail.
→ * These they detest those them.

(based on book chapter by Beatrice Santorini)
Substitution Test

(3) a. The little boy from next door fed the cat without a tail.

b. These black cats detest those green peas.

→ He fed her.

→ They detest them.

(based on book chapter by Beatrice Santorini)
Substitution Test

adverbs (here/there) can also be used.

(4) a. Put it on the table. → Put it there.
b. Put it over on the table. → Put it over there.
c. Put it over on the table. → Put it there.

(5) a. Put it on the table that's by the door. → * Put it there that's by the door.
b. Put it over on the table that's by the door. → * Put it over there that's by the door.
c. Put it over on the table that's by the door. → * Put it there that's by the door.

(based on book chapter by Beatrice Santorini)
Substitution Test

"so",
In a comparison.

(6) a. I am very happy, and Linda is so, too.
    b. I am very fond of Lukas, and Linda is so, too.
    c. I am very fond of my nephew, * and Linda is so of her niece.

(based on book chapter by Beatrice Santorini)
Substitution Test

'so', 'it', instead of a subordinate clause.

(7) a. I {know, suspect} that they're invited. \(\rightarrow\) I {know, suspect} it.
   b. I {imagine, think} that they're invited. \(\rightarrow\) I {imagine, think} so.

(based on book chapter by Beatrice Santorini)
Substitution Test

Substitution tests are a good indicator of constituency.

But they may fail, also for real constituents.

There are other tests: movement, short answers, it clefts

We will briefly cover movement.
See more in the Santorini's chapter.
Movement Test

Can we move the sequence to a different position in the sentence?

(based on book chapter by Beatrice Santorini)
Movement Test

(8) a. I fed the cats.
   
   b. I fed the cats with long, fluffy tails.

→ The cats, I fed ___. (The dogs, I didn't.)

→ The cats with long, fluffy tails, I fed ___. (The other cats, I didn't.)

(based on book chapter by Beatrice Santorinini)
Movement Test

(9) a. Prepositional phrase: The cat strolled across the porch with a confident air.

b. Adjective phrase: Ali Baba returned from his travels wiser than before.

c. Adverb phrase: They arrived at the concert hall more quickly than they had expected.

→ With a confident air, the cat strolled across the porch ___.
→ Wiser than before, Ali Baba returned from his travels ___.
→ More quickly than they had expected, they arrived at the concert hall ___.

(based on book chapter by Beatrice Santorini)
Movement Test

(10) a. I fed the cats with long, fluffy tails.
   b. The cat strolled across the porch with a confident air.
   c. Ali Baba returned from his travels wiser than before.
   d. They arrived at the concert hall more quickly than they had expected.

(based on book chapter by Beatrice Santorini)
Movement Test

(10) a. I fed the cats with long, fluffy tails.
   b. The cat strolled across the porch with a confident air.
   c. Ali Baba returned from his travels wiser than before.
   d. They arrived at the concert hall more quickly than they had expected.

→ * The cats, I fed ___ with long, fluffy tails.¹
→ * With a, the cat strolled across the porch ___ confident air.
→ * Wiser than, Ali Baba returned from his travels ___ before.
→ * More quickly than they, they arrived at the concert hall ___ had expected.

(based on book chapter by Beatrice Santorini)
See more tests in the notes.

I hope that you are convinced that constituents are "real".

You need to know how to identify them.
Parsing: recovering the constituents of a sentence.
Why is parsing hard?

Ambiguity

Fat people eat candy
Why is parsing hard?

Ambiguity

Fat people eat candy

```
S
  NP
    Adj
      Fat
    Nn
      people
  VP
    Vb
      eat
    NP
      Nn
        candy
```
Why is parsing hard?

Ambiguity

Fat people eat candy

Fat people eat accumulates

- **S**: Sentence
  - **NP**: Noun Phrase
    - **Adj**: Adjective
      - Fat
    - **Nn**: Noun
      - people
  - **VP**: Verb Phrase
    - **Vb**: Verb
      - eat
    - **NP**: Noun Phrase
      - candy
Why is parsing hard?

Ambiguity

Fat people eat candy

Fat people eat accumulates

S

NP

Adj

Fat

Nn

people

Vb

eat

NP

candy

S

NP

Nn

Fat

AdjP

people

Vb

eat

accumulates
Why is parsing hard?
Real Sentences are long... 

“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.
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 $X \rightarrow Y_1 Y_2 \cdots Y_n$ for $n \geq 0$, $X, Y_i \in (N \cup \Sigma)$
- $S \in N$ 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 $s_1, \ldots, s_n$ 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
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

NP → Adj Noun

I
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
Left-most derivation example

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

Noun → flies
Left-most derivation example

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

VP → Vb 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 → 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.
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
Noun → tomato
Adj → angry
...

...
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
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
Noun → tomato
Adj → angry

...
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
Noun → tomato
Adj → angry

Example

```
S
  /  
NP VP
  /  
Adj Noun Vb NP
   /  
   Adj Flies Vb Det Noun
      /  
     Angry likes a tomato

```
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
Noun → tomato
Adj → angry

Example
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
  Noun → tomato
  Adj → angry
  ...
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
Noun → tomato
Adj → angry

...
A Brief Introduction to English Syntax
A COMPREHENSIVE GRAMMAR OF THE ENGLISH LANGUAGE

Randolph Quirk
Sidney Greenbaum
Geoffrey Leech
Jan Svartvik

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

(by Mike Collins)
A Brief Overview of English Syntax

Parts of Speech (tags from the Brown corpus):

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

- **Determiners**
  - DT = determiner  e.g., the, a, some, every

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

<table>
<thead>
<tr>
<th>N</th>
<th>⇒</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ĖN</td>
<td>⇒</td>
<td>NN ĖN</td>
</tr>
<tr>
<td>ĖN</td>
<td>⇒</td>
<td>JJ ĖN</td>
</tr>
<tr>
<td>ĖN</td>
<td>⇒</td>
<td>ĖN ĖN</td>
</tr>
<tr>
<td>NP</td>
<td>⇒</td>
<td>DT ĖN</td>
</tr>
<tr>
<td>NN</td>
<td>⇒</td>
<td>box</td>
</tr>
<tr>
<td>NN</td>
<td>⇒</td>
<td>car</td>
</tr>
<tr>
<td>NN</td>
<td>⇒</td>
<td>mechanic</td>
</tr>
<tr>
<td>NN</td>
<td>⇒</td>
<td>pigeon</td>
</tr>
<tr>
<td>DT</td>
<td>⇒</td>
<td>the</td>
</tr>
<tr>
<td>DT</td>
<td>⇒</td>
<td>a</td>
</tr>
<tr>
<td>JJ</td>
<td>⇒</td>
<td>fast</td>
</tr>
<tr>
<td>JJ</td>
<td>⇒</td>
<td>metal</td>
</tr>
<tr>
<td>JJ</td>
<td>⇒</td>
<td>idealistic</td>
</tr>
<tr>
<td>JJ</td>
<td>⇒</td>
<td>clay</td>
</tr>
</tbody>
</table>

The box
A car
The pigeon car
The fast metal pigeon

(by Mike Collins)
Prepositions, and Prepositional Phrases

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

| Ń   | ⇒ | NN |
| Ń   | ⇒ | NN Ń |
| Ń   | ⇒ | JJ Ń |
| Ń   | ⇒ | Ń Ń |
| NP  | ⇒ | DT Ń |
| PP  | ⇒ | IN NP |
| Ń   | ⇒ | Ń PP |

(by Mike Collins)
An Extended Grammar

<table>
<thead>
<tr>
<th>NP</th>
<th>DT</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN</td>
<td>box</td>
<td></td>
</tr>
<tr>
<td>NN</td>
<td>car</td>
<td></td>
</tr>
<tr>
<td>NN</td>
<td>mechanic</td>
<td></td>
</tr>
<tr>
<td>NN</td>
<td>pigeon</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>in</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>under</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>of</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>on</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>with</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>as</td>
<td></td>
</tr>
</tbody>
</table>

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

(by Mike Collins)
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

(by Mike Collins)
PPs Modifying Verb Phrases

A new rule: \[ VP \rightarrow VP \ PP \]

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

(by Mike Collins)
Complementizers, and SBARs

- Complementizers
  \( \text{COMP} = \text{complementizer} \quad \text{e.g., that} \)

- SBAR
  \( \text{SBAR} \rightarrow \text{COMP} \quad S \)

**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 $\rightarrow$ V[5] SBAR
  - VP $\rightarrow$ V[6] NP SBAR
  - VP $\rightarrow$ 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

(by Mike Collins)
Coordination

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

- New Rules
  \[
  \begin{align*}
  \text{NP} & \rightarrow \text{NP} \quad \text{CC} \quad \text{NP} \\
  \tilde{\text{N}} & \rightarrow \tilde{\text{N}} \quad \text{CC} \quad \tilde{\text{N}} \\
  \text{VP} & \rightarrow \text{VP} \quad \text{CC} \quad \text{VP} \\
  \text{S} & \rightarrow \text{S} \quad \text{CC} \quad \text{S} \\
  \text{SBAR} & \rightarrow \text{SBAR} \quad \text{CC} \quad \text{SBAR}
  \end{align*}
  \]

(by Mike Collins)
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:

(by Mike Collins)
CFGs and center embedding

The mouse ate the corn.
The mouse that the snake ate ate the corn.
The mouse that the snake that the hawk ate ate ate the corn.

These sentences are all grammatical. They can be generated by a CFG:

\[
S \rightarrow NP \ VP \\
NP \rightarrow NP \ RelClause \\
RelClause \rightarrow \text{that } NP \text{ ate}
\]

Linguists distinguish between a speaker’s
- competence (grammatical knowledge) and
- performance (processing and memory limitations)
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.
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.
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.

Solution

- We’ll ask a genius linguist to write 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

- 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

((fruit/ADJ flies/NN) (like/VB (time/NN (flies/VB (like/IN )))))
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 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
     └── NP
         │   │
         Adj Noun Vb NP
         └── a banana
             └── Det Noun
                 └── like Flies Fruit
```
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 an its children is a rule in $R$

Extracting Rules

```
S
  NP  VP
    Adj Noun  Vb NP
      Fruit  Flies like  Det Noun
      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 and its children is a rule in $R$

Extracting Rules

$$S \rightarrow NP \ VP$$

$$NP \rightarrow Adj \ Noun$$

$$VP \rightarrow Vb \ NP$$

$$S$$

$$NP$$

- Adj
  - Fruit
- Noun
  - Flies

$$VP$$

- Vb
  - like
- Det
  - a
- Noun
  - 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
Adj
Fruit
Noun
Flies
VP
Vb
like
Det
a
Noun
banana

S $\rightarrow$ NP VP
NP $\rightarrow$ Adj Noun
Adj $\rightarrow$ fruit
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 $\Rightarrow$ 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

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

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

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

```
S
  /\    /
 NP   VP
    /\   /
   Adj Noun     Vb NP
     /\   /
    Fruit Flies like Det Noun
       /
      a 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 $NP \rightarrow \text{Adj} \ Noun$
0.7 $NP \rightarrow \text{Det} \ Noun$
1.0 $VP \rightarrow \text{Vb} \ NP$

- 

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

Example

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

1 * 0.3 * 0.2 * 0.7 * 1.0 * 0.2 * 1 * 1 * 0.4 = 0.0033
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

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

$1 \times 0.3 \times 0.2 \times 0.7 \times 1.0 \times 0.2 \times 1 \times 1 \times 1 \times 0.4 = 0.0033$
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)

\[
P(\text{LHS} \rightarrow \text{RHS}) = \frac{\text{count}(\text{LHS} \rightarrow \text{RHS})}{\text{count}(\text{LHS} \rightarrow \epsilon)}
\]

We can also add smoothing and backoff, as before.
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!
CKY

Cocke  Kasami  Younger
CKY

Cocke  Kasami  Younger
196?
Cocke    Kasami    Younger
196?     1965
CKY

Cocke   Kasami   Younger
196?    1965     1967
3 Interesting Problems

- Recognition
- Parsing
- Disambiguation

CKY can do all of these in polynomial time for any CNF grammar.
3 Interesting Problems

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

- Recognition
  - Can this string be generated by the grammar?
- Parsing
  - Show me a possible derivation…
- 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
  - 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**
CNF
Chomsky Normal Form

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
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 than 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

Sue saw her girl with a telescope
Filling the table

<table>
<thead>
<tr>
<th>Sue</th>
<th>saw</th>
<th>her</th>
<th>girl</th>
<th>with</th>
<th>a</th>
<th>telescope</th>
</tr>
</thead>
</table>
Handling Unary rules?

Sue saw her girl with a telescope.
Which Order?

Sue saw her boy with a telescope
Complexity?

$\text{In} 2 \text{g cells to fill}$

$\text{In} 2 \text{g ways to fill each one}$

$O(2^{3n^3})$
Complexity?

- $n^2g$ 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 $O(g^3 n^3)$, 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.
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
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.
- \( q(LHS \rightarrow RHS) = \frac{\text{count}(LHS \rightarrow RHS)}{\text{count}(LHS \rightarrow \diamond)} \)
- We can also add smoothing and backoff, as before.
- Dealing with unknown words - like in the HMM
The big question

Does this work?
Evaluation
Let’s assume we have a parser, how do we know how good it is?

Compare output trees to gold trees.

- Represent each tree as a set of labeled spans.
  - NP from word 1 to word 5.
  - VP from word 3 to word 4.
  - S from word 1 to word 23.

Measure Precision, Recall and F1 over these spans, as in
the segmentation case.
Let’s assume we have a parser, how do we know how good it is?

- Compare output trees to gold trees.
  - But how do we compare trees?
  - Credit of 1 if tree is correct and 0 otherwise, is too harsh.
Let’s assume we have a parser, how do we know how good it is?

Compare output trees to gold trees.
- But how do we compare trees?
  - Credit of 1 if tree is correct and 0 otherwise, is too harsh.

Represent each tree as a set of labeled spans.
- NP from word 1 to word 5.
- VP from word 3 to word 4.
- S from word 1 to word 23.
- ...

Measure Precision, Recall and F₁ over these spans, as in the segmentation case.
Evaluation: Representing Trees as Constituents

(by Mike Collins)
Precision and Recall

<table>
<thead>
<tr>
<th>Label</th>
<th>Start Point</th>
<th>End Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>NP</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>NP</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>PP</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>NP</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>VP</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Label</th>
<th>Start Point</th>
<th>End Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>NP</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>PP</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>NP</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>VP</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>

- $G =$ number of constituents in gold standard $= 7$
- $P =$ number in parse output $= 6$
- $C =$ number correct $= 6$

Recall $= 100\% \times \frac{C}{G} = 100\% \times \frac{6}{7}$

Precision $= 100\% \times \frac{C}{P} = 100\% \times \frac{6}{6}$

(by Mike Collins)
Parsing Evaluation

- Is this a good measure?
  - Why? Why not?
How well does the PCFG parser we learned do?

Not very well: about 73% $F_1$ score.
Problems with PCFGs
Weaknesses of Probabilistic Context-Free Grammars

Michael Collins, Columbia University

(by Mike Collins)
Weaknesses of PCFGs

- Lack of sensitivity to lexical information
- Lack of sensitivity to structural frequencies

(by Mike Collins)
p(t) =  \[ q(S \rightarrow NP \ VP) \times q(NNP \rightarrow IBM) \times q(VP \rightarrow V \ NP) \times q(Vt \rightarrow bought) \times q(NP \rightarrow NNP) \times q(NNP \rightarrow Lotus) \]
Another Case of PP Attachment Ambiguity

(by Mike Collins)
If $q(NP \rightarrow NP \text{ PP}) > q(VP \rightarrow VP \text{ PP})$ then (b) is more probable, else (a) is more probable. 

**Attachment decision is completely independent of the words**

(by Mike Collins)
A Case of Coordination Ambiguity

(by Mike Collins)
Here the two parses have identical rules, and therefore have identical probability under any assignment of PCFG rule probabilities.