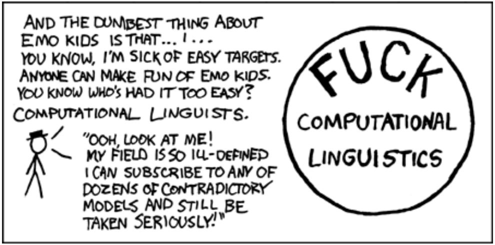
Parsing Natural Language With PCFGs an Overview

Yoav Goldberg



Graduate Seminar, Mar. 2008

Outline

Natural Language Parsing

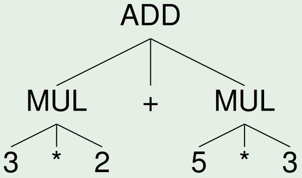
- Sentences in natural language have structure.
- Linguists create Linguistic Theories for defining this structure.

Structure
Example 1: math

3*2+5*3

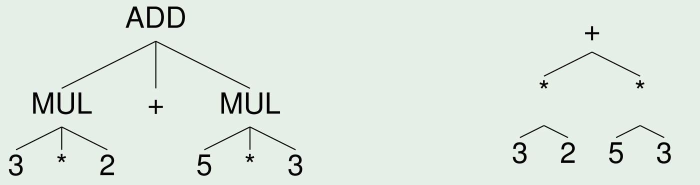
Structure
Example 1: math

3*2+5*3



Structure
Example 1: math

3*2+5*3



Structure

Example 2: Language Data

Fruit flies like a banana

Constituency Structure

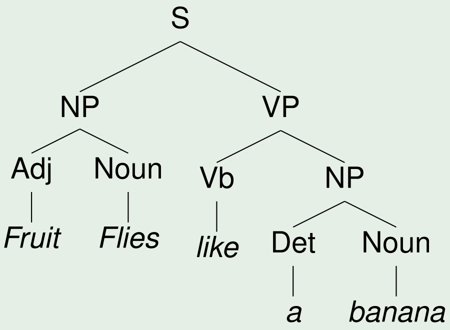
Dependency Structure

Structure

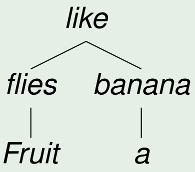
Example 2: Language Data

Fruit flies like a banana

Constituency Structure



Dependency Structure



Parsing

What is parsing?

- Parsing is the task of assigning structure to a sentence.
- You can think of it as a function from sentences to structures.

Constituency Parsing

- In this talk we concentrate on Constituency Parsing: mapping from sentences to trees with labeled nodes and the sentence words at the leaves.
- I discuss only binary-branching trees.
  - Not a big restriction: binarizing trees is easy.

Parsing

What is parsing?

- Parsing is the task of assigning structure to a sentence.
- You can think of it as a function from sentences to structures.

Constituency Parsing

- In this talk we concentrate on Constituency Parsing: mapping from sentences to trees with labeled nodes and the sentence words at the leaves.
- I discuss only binary-branching trees.
  - Not a big restriction: binarizing trees is easy.

Why is Parsing Interesting?

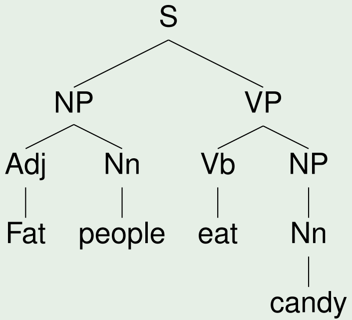
- It's a first step towards understanding a text.
- Many other language tasks use sentence structure as their input.

Why is parsing hard?
Ambiguity

Fat people eat candy

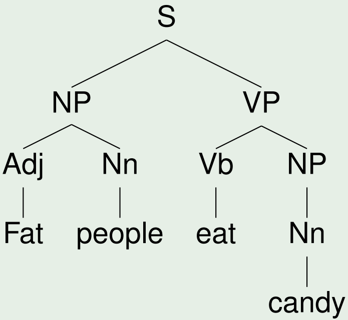
Why is parsing hard?
Ambiguity

Fat people eat candy



Why is parsing hard?
Ambiguity

Fat people eat candy

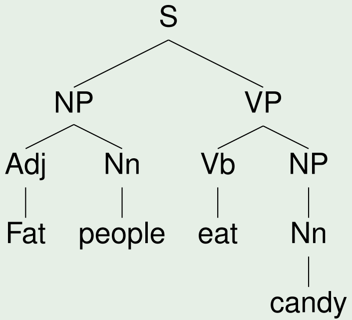


Fat people eat accumulates

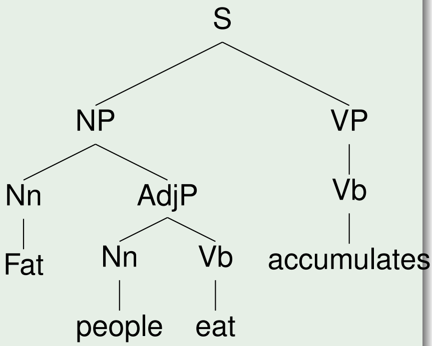
Why is parsing hard?

Ambiguity

Fat people eat candy



Fat people 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."

Context Free Grammars

a simple grammar

S $\longrightarrow$ NP VP
NP $\longrightarrow$ Adj Noun
NP $\longrightarrow$ Det Noun
VP $\longrightarrow$ Vb NP
-
Adj $\longrightarrow$ fruit
Noun $\longrightarrow$ flies
Vb $\longrightarrow$ like
Det $\longrightarrow$ a
Noun $\longrightarrow$ banana
Noun $\longrightarrow$ tomato
Adj $\longrightarrow$ angry
...

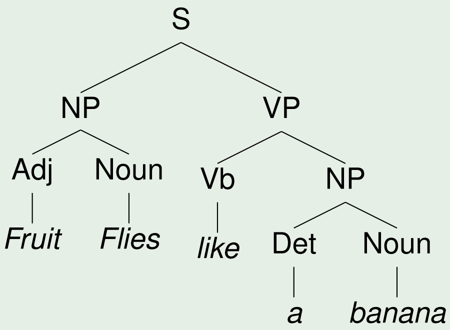
Example

Context Free Grammars

a simple grammar

S $\longrightarrow$ NP VP
NP $\longrightarrow$ Adj Noun
NP $\longrightarrow$ Det Noun
VP $\longrightarrow$ Vb NP
-
Adj $\longrightarrow$ fruit
Noun $\longrightarrow$ flies
Vb $\longrightarrow$ like
Det $\longrightarrow$ a
Noun $\longrightarrow$ banana
Noun $\longrightarrow$ tomato
Adj $\longrightarrow$ angry
...

Example

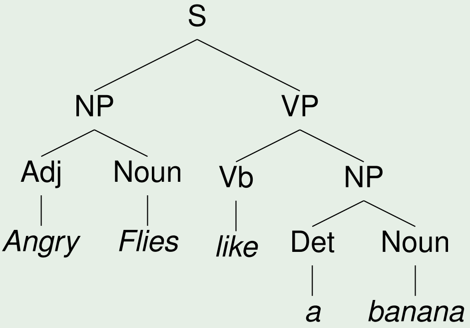


Context Free Grammars

a simple grammar

S $\longrightarrow$ NP VP
NP $\longrightarrow$ Adj Noun
NP $\longrightarrow$ Det Noun
VP $\longrightarrow$ Vb NP
-
Adj $\longrightarrow$ fruit
Noun $\longrightarrow$ flies
Vb $\longrightarrow$ like
Det $\longrightarrow$ a
Noun $\longrightarrow$ banana
Noun $\longrightarrow$ tomato
Adj $\longrightarrow$ angry
...

Example

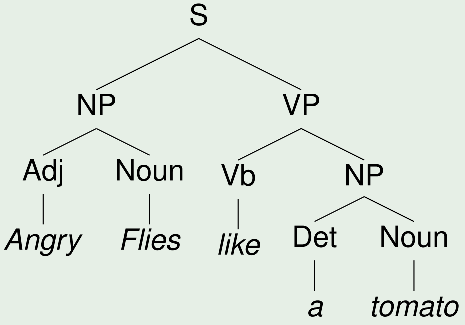


Context Free Grammars

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
...

Example

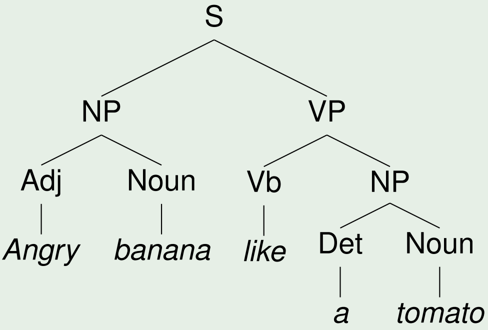


Context Free Grammars

a simple grammar

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
. . .

Example

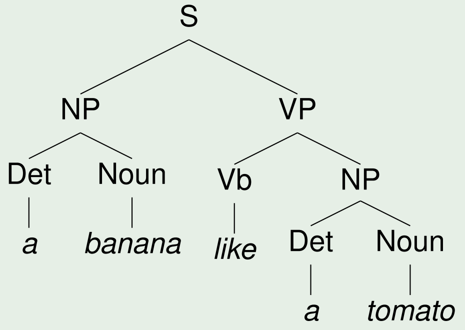


Context Free Grammars

a simple grammar

S $\longrightarrow$ NP VP
NP $\longrightarrow$ Adj Noun
NP $\longrightarrow$ Det Noun
VP $\longrightarrow$ Vb NP
-
Adj $\longrightarrow$ fruit
Noun $\longrightarrow$ flies
Vb $\longrightarrow$ like
Det $\longrightarrow$ a
Noun $\longrightarrow$ banana
Noun $\longrightarrow$ tomato
Adj $\longrightarrow$ angry
. . .

Example

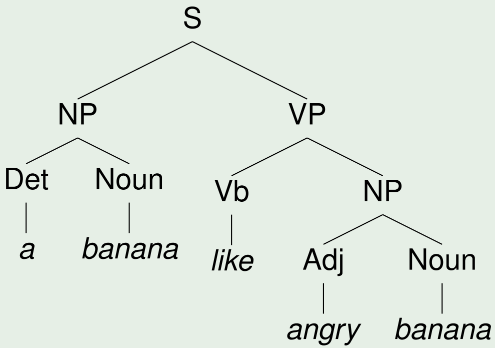


Context Free Grammars

a simple grammar

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
...

Example



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.

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

- 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.

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

- 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

- Real grammar: hundreds of possible derivations per sentence.

Solution

- No problem! We'll choose the best one. (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.

Solution

- No problem! We'll choose the best one. (soon)

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

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

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

Treebank Sentence Example

```
( (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) )))
```

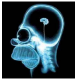
Supervised Learning from a Treebank

((fruit/ADJ flies/NN) (like/VB

(time/NN (flies/VB (like/IN

. . . .

. . . .



From CFG to PCFG

Choosing the best tree

- 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.

From CFG to PCFG

Choosing the best tree

- 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.

From CFG to PCFG

Choosing the best tree

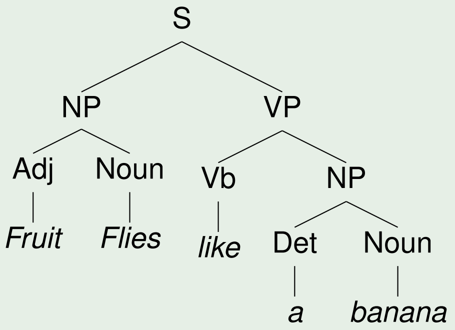
- English is NOT generated from CFG $\Rightarrow$ It's generated by a PCFG!
- PCFG: probabilistic context free grammar. Just like a CFG, but each rule has an associated probability.
- All probabilities for the same LHS sum to 1.
- Multiplying all the rule probs in a derivation gives the probability of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

PCFG Example

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



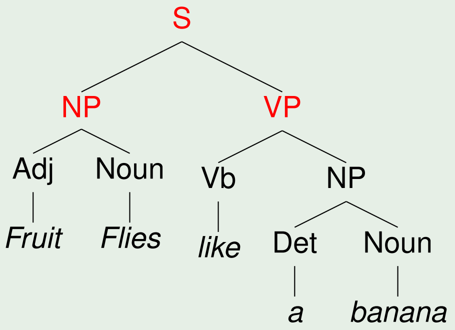
$1 * 0.3 * 0.2 * 0.7 * 1.0 * 0.2 * 1 * 1 * 0.4 = 0.0033$

PCFG Example

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



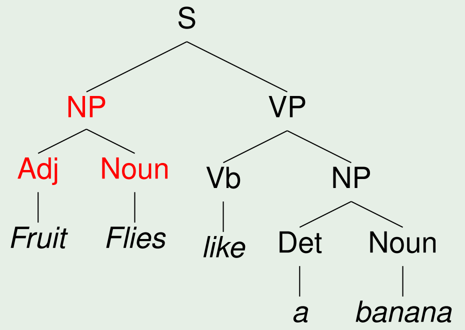
$1 * 0.3 * 0.2 * 0.7 * 1.0 * 0.2 * 1 * 1 * 0.4 = 0.0033$

PCFG Example

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



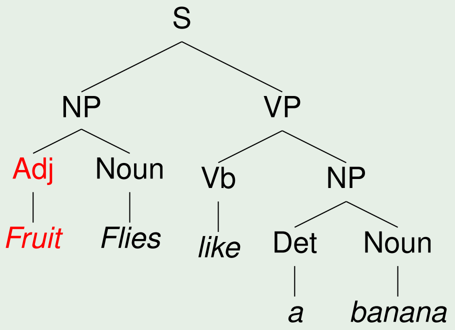
$1 * 0.3 * 0.2 * 0.7 * 1.0 * 0.2 * 1 * 1 * 0.4 = 0.0033$

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



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

Parsing with PCFG

So parsing is...

- 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 probabilitles

- The same way we obtained the rules: we estimate them from the Treebank.
- $P(LHS \rightarrow RHS) = \frac{count(LHS \rightarrow RHS)}{count(LHS \rightarrow \Diamond)}$
- This is called "Relative Frequency".
- (we probably need to add smoothing to get better estimations, but let's ignore it for this talk)

Parsing with PCFG

So parsing is. . .

- 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 probabilitles

- The same way we obtained the rules: we estimate them from the Treebank.
- $P(LHS \rightarrow RHS) = \frac{count(LHS \rightarrow RHS)}{count(LHS \rightarrow \Diamond)}$
- This is called "Relative Frequency".
- (we probably need to add smoothing to get better estimations, but let's ignore it for this talk)

So we know how to parse

So we know how to parse

Too bad this doesn't really work

- Parsing this way yield pretty bad results (around 60-70% of the common measure)
- Main reasons:
  - language is not really generated by PCFGs.
  - This is probably not how humans process language.
- Secondary reason: Treebank derived grammars are not very good.

So we know how to parse

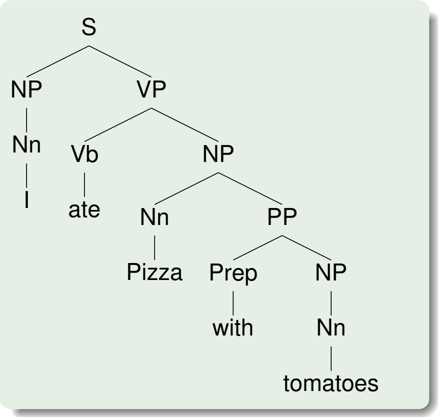
Too bad this doesn't really work

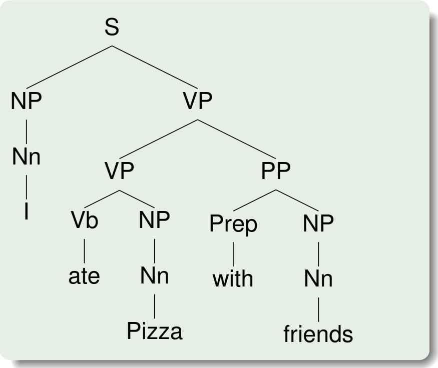
- Parsing this way yield pretty bad results (around 60-70% of the common measure)
- Main reasons:
  - language is not really generated by PCFGs.
  - This is probably not how humans process language.
- Secondary reason: Treebank derived grammars are not very good.

Solutions

- We need to get better grammars.
- We do it by encoding some context into the grammar.

1 Problem with context-freeness

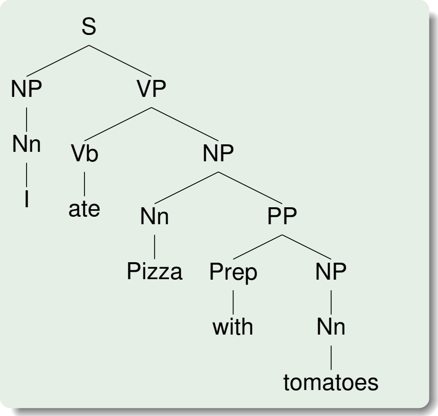


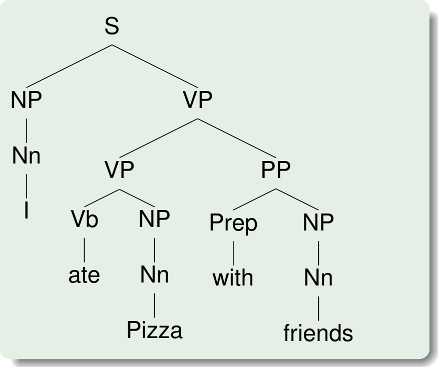


A PCFG Can't choose between these two structures!

- After the word level, the sentences look the same..

1 Problem with context-freeness

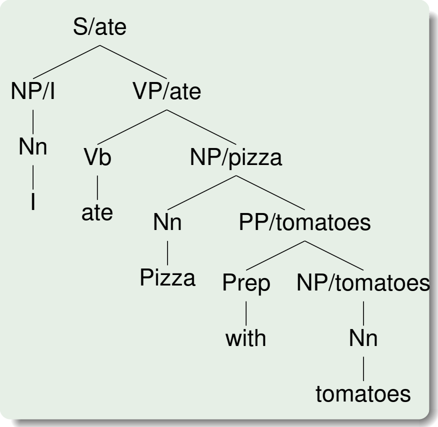


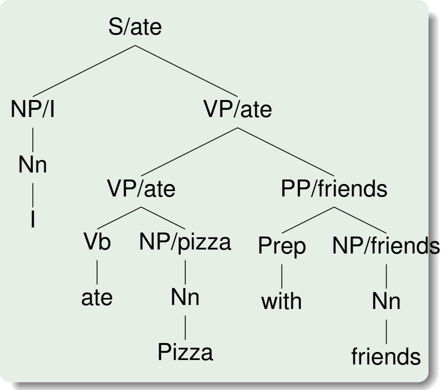


A PCFG Can't choose between these two structures!

- After the word level, the sentences look the same..

Better Grammars 1: Lexicalization (adding words)





Ah! Much better

Better Grammars 1: Lexicalization (adding words)

Lexicalized Grammar

VP/ate	$\rightarrow$	VP/ate PP/friends
VP/ate	$\rightarrow$	VP/ate PP/tomatoes
NP/pizza	$\rightarrow$	NP/pizza PP/friends
NP/pizza	$\rightarrow$	NP/pizza PP/tomatoes

. . . but

- Grammar is HUGE
- Hard to estimate parameters (many rare or unseen events)

Fortunately. . .

- Collins (1998), Charniak (1999) managed to do it.
- Lexicalized Treebank grammars achieve accuracy of 88 parsing-measure

Better Grammars 1: Lexicalization (adding words)

Lexicalized Grammar

VP/ate	$\rightarrow$	VP/ate PP/friends	GOOD
VP/ate	$\rightarrow$	VP/ate PP/tomatoes	
NP/pizza	$\rightarrow$	NP/pizza PP/friends	
NP/pizza	$\rightarrow$	NP/pizza PP/tomatoes	

. . . but

- Grammar is HUGE
- Hard to estimate parameters (many rare or unseen events)

Fortunately. . .

- Collins (1998), Charniak (1999) managed to do it.
- Lexicalized Treebank grammars achieve accuracy of 88 parsing-measure

Better Grammars 1: Lexicalization (adding words)

Lexicalized Grammar

VP/ate	$\rightarrow$	VP/ate PP/friends	GOOD
VP/ate	$\rightarrow$	VP/ate PP/tomatoes	BAD
NP/pizza	$\rightarrow$	NP/pizza PP/friends	
NP/pizza	$\rightarrow$	NP/pizza PP/tomatoes	

. . . but

- Grammar is HUGE
- Hard to estimate parameters (many rare or unseen events)

Fortunately. . .

- Collins (1998), Charniak (1999) managed to do it.
- Lexicalized Treebank grammars achieve accuracy of 88 parsing-measure

Better Grammars 1: Lexicalization (adding words)

Lexicalized Grammar

VP/ate	$\rightarrow$	VP/ate PP/friends	GOOD
VP/ate	$\rightarrow$	VP/ate PP/tomatoes	BAD
NP/pizza	$\rightarrow$	NP/pizza PP/friends	BAD
NP/pizza	$\rightarrow$	NP/pizza PP/tomatoes	GOOD

. . . but

- Grammar is HUGE
- Hard to estimate parameters (many rare or unseen events)

Fortunately. . .

- Collins (1998), Charniak (1999) managed to do it.
- Lexicalized Treebank grammars achieve accuracy of 88 parsing-measure

Better Grammars 1: Lexicalization (adding words)

Lexicalized Grammar

VP/ate	$\rightarrow$	VP/ate PP/friends	GOOD
VP/ate	$\rightarrow$	VP/ate PP/tomatoes	BAD
NP/pizza	$\rightarrow$	NP/pizza PP/friends	BAD
NP/pizza	$\rightarrow$	NP/pizza PP/tomatoes	GOOD

. . . but

- Grammar is HUGE
- Hard to estimate parameters (many rare or unseen events)

Fortunately. . .

- Collins (1998), Charniak (1999) managed to do it.
- Lexicalized Treebank grammars achieve accuracy of 88 parsing-measure

Better Grammars 1: Lexicalization (adding words)

Lexicalized Grammar

VP/ate	$\rightarrow$	VP/ate PP/friends	GOOD
VP/ate	$\rightarrow$	VP/ate PP/tomatoes	BAD
NP/pizza	$\rightarrow$	NP/pizza PP/friends	BAD
NP/pizza	$\rightarrow$	NP/pizza PP/tomatoes	GOOD

...but

- Grammar is HUGE
- Hard to estimate parameters (many rare or unseen events)

Fortunately...

- Collins (1998), Charniak (1999) managed to do it.
- Lexicalized Treebank grammars achieve accuracy of 88 parsing-measure

Better Grammars 2: non-lexical context

Apparently, we can do quite good without the words

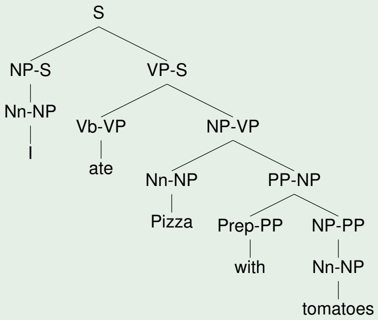
- Klein and Manning, standing on shoulders of Johnson, Collins and others.

Better Grammars 2: non-lexical context

Apparently, we can do quite good without the words

- Klein and Manning, standing on shoulders of Johnson, Collins and others.

1) Parent Annotation



Better Grammars 2: non-lexical context

Apparently, we can do quite good without the words

- Klein and Manning, standing on shoulders of Johnson, Collins and others.

2) Linguistically Motivated Tag Splits

AUX $\rightarrow$ am | is | are | was | were | have | had | has	AUX-HAVE $\rightarrow$ have | had | has AUX-BE $\rightarrow$ am | is | are | was | were
IN $\rightarrow$ while | as | if | that | for | of | in | from	IN-CC $\rightarrow$ while | as | if IN-CM $\rightarrow$ that | for IN-PP $\rightarrow$ of | in | from
CC $\rightarrow$ and | but | &	CC-1 $\rightarrow$ and CC-2 $\rightarrow$ but CC-3 $\rightarrow$ &

Better Grammars 2: non-lexical context

Apparently, we can do quite good without the words

- Klein and Manning, standing on shoulders of Johnson, Collins and others.

3) Some other annotations

- Mark any node dominating a verb.
- Separate non-recursive NPs from regular NPs
- Separate temporal (time) NPs from other NPs
- etc, . . .

Better Grammars 2: non-lexical context

Apparently, we can do quite good without the words

- Klein and Manning, standing on shoulders of Johnson, Collins and others.

3) Some other annotations

- Mark any node dominating a verb.
- Separate non-recursive NPs from regular NPs
- Separate temporal (time) NPs from other NPs
- etc, . . .

- . . . and get an accuracy of 86.9 parsing-measure!

Better Grammars 3: Automatic Grammar Refinement

Humans are quite good at refining the grammar. Computers... are even better!

- Petrov *et al.* 2006 (following Matsuzaki, 2005) – Automatic grammar refinement.
- Start with a grammar extracted from the Treebank.
  - Tiny: 98 non-terminal symbols. 4076 rules.
  - about 62 parsing-accuracy
- Iteratively:
  - Split each symbol in 2 (e.g. NP $\Rightarrow$ NP1, NP2). Make splits that maximize the likelihood of the Treebank.
  - Merge back "useless" splits to keep the grammar size reasonable.

Better Grammars 3: Automatic Grammar Refinement

Humans are quite good at refining the grammar. Computers. . . are even better!

- Petrov *et al.* 2006 (following Matsuzaki, 2005) – Automatic grammar refinement.
- Start with a grammar extracted from the Treebank.
  - Tiny: 98 non-terminal symbols. 4076 rules.
  - about 62 parsing-accuracy
- Iteratively:
  - Split each symbol in 2 (e.g. NP $\Rightarrow$ NP1, NP2). Make splits that maximize the likelihood of the Treebank.
  - Merge back “useless” splits to keep the grammar size reasonable.

Better Grammars 3: Automatic Grammar Refinement

Humans are quite good at refining the grammar. Computers. . . are even better!

- Petrov *et al.* 2006 (following Matsuzaki, 2005) – Automatic grammar refinement.
- Start with a grammar extracted from the Treebank.
  - Tiny: 98 non-terminal symbols. 4076 rules.
  - about 62 parsing-accuracy
- Iteratively:
  - Split each symbol in 2 (e.g. NP $\Rightarrow$ NP1, NP2). Make splits that maximize the likelihood of the Treebank.
  - Merge back "useless" splits to keep the grammar size reasonable.

Better Grammars 3: Automatic Grammar Refinement

. . . After 6 iterations (and a few days)

- Big, but not huge: 1043 non-terminal symbols.

Better Grammars 3: Automatic Grammar Refinement

. . . After 6 iterations (and a few days)

- Big, but not huge: 1043 non-terminal symbols.
- 90.2 parsing-accuracy!

Better Grammars 3: Automatic Grammar Refinement

Is it lexicalized?

- Yes, but not much.
- Every POS category can be split into at most 64 sub-categories.
- So there may be 64 kinds of Nouns, 64 kinds of Verbs, etc.
- This catches the distinctions between
  `comes,goes,drives (somewhere)`
  and
  `gives,takes,sells (something, someone)`
  but does not solve the Pizaa case.

VBZ

VBZ-0	gives	sells	takes
VBZ-1	comes	goes	works
VBZ-2	includes	owns	is
VBZ-3	puts	provides	takes
VBZ-4	says	adds	Says
VBZ-5	believes	means	thinks
VBZ-6	expects	makes	calls
VBZ-7	plans	expects	wants
VBZ-8	is	’s	gets
VBZ-9	’s	is	remains
VBZ-10	has	’s	is
VBZ-11	does	Is	Does

NNP

NNP-0	Jr.	Goldman	INC.
NNP-1	Bush	Noriega	Peters
NNP-2	J.	E.	L.
NNP-3	York	Francisco	Street
NNP-4	Inc	Exchange	Co
NNP-5	Inc.	Corp.	Co.
NNP-6	Stock	Exchange	York
NNP-7	Corp.	Inc.	Group
NNP-8	Congress	Japan	IBM
NNP-9	Friday	September	August
NNP-10	Shearson	D.	Ford
NNP-11	U.S.	Treasury	Senate
NNP-12	John	Robert	James
NNP-13	Mr.	Ms.	President
NNP-14	Oct.	Nov.	Sept.
NNP-15	New	San	Wall

JJS

JJS-0	largest	latest	biggest
JJS-1	least	best	worst
JJS-2	most	Most	least

DT

DT-0	the	The	a
DT-1	A	An	Another
DT-2	The	No	This
DT-3	The	Some	These
DT-4	all	those	some
DT-5	some	these	both
DT-6	That	This	each
DT-7	this	that	each
DT-8	the	The	a
DT-9	no	any	some
DT-10	an	a	the
DT-11	a	this	the

CD

CD-0	1	50	100
CD-1	8.50	15	1.2
CD-2	8	10	20
CD-3	1	30	31
CD-4	1989	1990	1988
CD-5	1988	1987	1990
CD-6	two	three	five
CD-7	one	One	Three
CD-8	12	34	14
CD-9	78	58	34
CD-10	one	two	three
CD-11	million	billion	trillion

PRP

PRP-0	It	He	I
PRP-1	it	he	they
PRP-2	it	them	him

RBR

RBR-0	further	lower	higher
RBR-1	more	less	More
RBR-2	earlier	Earlier	later

IN

IN-0	In	With	After
IN-1	In	For	At
IN-2	in	for	on
IN-3	of	for	on
IN-4	from	on	with
IN-5	at	for	by
IN-6	by	in	with
IN-7	for	with	on
IN-8	If	While	As
IN-9	because	if	while
IN-10	whether	if	That
IN-11	that	like	whether
IN-12	about	over	between
IN-13	as	de	Up
IN-14	than	ago	until
IN-15	out	up	down

RB

RB-0	recently	previously	still
RB-1	here	back	now
RB-2	very	highly	relatively
RB-3	so	too	as
RB-4	also	now	still
RB-5	however	Now	However
RB-6	much	far	enough
RB-7	even	well	then
RB-8	as	about	nearly
RB-9	only	just	almost
RB-10	ago	earlier	later
RB-11	rather	instead	because
RB-12	back	close	ahead
RB-13	up	down	off
RB-14	not	Not	maybe
RB-15	n’t	not	also

Squeezing it a bit more

Now what?

- Automatically refined grammars are almost the best we can do
- How do we improve upon that?

- Observation: all decisions are quite localized. The process does not look at complete trees...
- Output k-best parses. Rank them based on tree-global features (usually using machine learning).

This is the current state-of-the-art in parsing technology

Squeezing it a bit more

Now what?

- Automatically refined grammars are almost the best we can do
- How do we improve upon that?

- Observation: all decisions are quite localized. The process does not look at complete trees...
- Output k-best parses. Rank them based on tree-global features (usually using machine learning).

This is the current state-of-the-art in parsing technology

Squeezing it a bit more

Now what?

- Automatically refined grammars are almost the best we can do
- How do we improve upon that?

- Observation: all decisions are quite localized. The process does not look at complete trees. . .
- Output k-best parses. Rank them based on tree-global features (usually using machine learning).

This is the current state-of-the-art in parsing technology

So now you know how to parse (sort-of)

Discussion

- World's best parsers work while hardly relying on words!
- Either words are not very important. . . or we are not using them correctly
- $\Rightarrow$ lot's of room for improvement. . . :)

Thanks

Thanks