Context Free Grammars

Introduction to Natural Language Processing

Andrew McCallum

Also includes material from Chris Manning.

Today's Main Points

- In-class hands-on exercise
- A brief introduction to a little syntax.
- Define context free grammars.
  Give some examples.
- Parsing as search
  Top-down, bottom up (shift-reduce), and the problems with each.

Language structure and meaning

We want to know how meaning is mapped onto what language structures. Commonly in English in ways like this:

[Thing The dog] is [Place in the garden]

[Thing The dog] is [Property fierce]

[Action [Thing The dog] is chasing [Thing the cat]]

[State [Thing The dog] was sitting [Place in the garden] [Time yesterday]]

[Action [Thing We] ran [Path out into the water]]

[Action [Thing The dog] barked [Property/Manner loudly]]

[Action [Thing The dog] barked [Property/Amount nonstop for five hours]]

Word categories: Traditional parts of speech

Noun	Names of things	boy, cat, truth
Verb	Action or state	become, hit
Pronoun	Used for noun	I, you, we
Adverb	Modifies V, Adj, Adv	sadly, very
Adjective	Modifies noun	happy, clever
Conjunction	Joins things	and, but, while
Preposition	Relation of N	to, from, into
Interjection	An outcry	ouch, oh, alas, psst

Part of speech “Substitution Test”

The $\{$sad, intelligent, green, fat, ...$\}$ one is in the corner.

Constituency

The idea: Groups of words may behave as a single unit or phrase, called a **consituent**.

E.g. Noun Phrase

Kermit the frog
they
December twenty-sixth
the reason he is running for president

Constituency

Sentences have parts, some of which appear to have subparts. These groupings of words that go together we will call constituents.

(How do we know they go together? Coming in a few slides...)

I hit the man with a cleaver
I hit [the man with a cleaver]
I hit [the man] with a cleaver

You could not go to her party
You [could not] go to her party
You could [not go] to her party

Constituent Phrases

For constituents, we usually name them as phrases based on the word that heads the constituent:

the man from Amherst	is a Noun Phrase (NP) because the head man is a noun
extremely clever	is an Adjective Phrase (AP) because the head clever is an adjective
down the river	is a Prepositional Phrase (PP) because the head down is a preposition
killed the rabbit	is a Verb Phrase (VP) because the head killed is a verb

Note that a word is a constituent (a little one). Sometimes words also act as phrases. In:

Joe grew potatoes.
Joe and *potatoes* are both nouns and noun phrases.

Compare with:

$\underline{\textit{The man from Amherst}}$ *grew* $\underline{\textit{beautiful russet potatoes}}$*.*

We say *Joe* counts as a noun phrase because it appears in a place that a larger noun phrase could have been.

Evidence constituency exists

1. They appear in similar environments (before a verb)
   $\textit{\underline{Kermit the frog} comes on stage}$
   $\textit{\underline{They} come to Massachusetts every summer}$
   $\textit{\underline{December twenty-sixth} comes after Christmas}$
   $\textit{\underline{The reason he is running for president} comes out only now.}$
   But not each individual word in the consituent
   $\textit{*\underline{The} comes our... *\underline{is} comes out... *\underline{for} comes out...}$

2. The constituent can be placed in a number of different locations
   Consituent = Prepositional phrase: $\textit{On December twenty-sixth}$
   $\textit{\underline{On December twenty-sixth} I'd like to fly to Florida.}$
   $\textit{I'd like to fly \underline{on December twenty-sixth} to Florida.}$
   $\textit{I'd like to fly to Florida \underline{on December twenty-sixth}.}$
   But not split apart
   $*\textit{\underline{On December} I'd like to fly \underline{twenty-sixth} to Florida.}$
   $\textit{*\underline{On} I'd like to fly \underline{December twenty-sixth} to Florida.}$

Context-free grammar

The most common way of modeling constituency.

CFG = Context-Free Grammar = Phrase Structure Grammar
= BNF = Backus-Naur Form

The idea of basing a grammar on constituent structure dates back to Wilhem Wundt (1890), but not formalized until Chomsky (1956), and, independently, by Backus (1959).

Context-free grammar

$G = \langle T, N, S, R \rangle$

- T is set of terminals (lexicon)
- N is set of non-terminals For NLP, we usually distinguish out a set $P \subset N$ of *preterminals* which always rewrite as terminals.
- S is start symbol (one of the nonterminals)
- R is rules/productions of the form $X \rightarrow \gamma$, where X is a nonterminal and γ is a sequence of terminals and nonterminals (may be empty).
- A grammar G generates a language L.

An example context-free grammar

$G = \langle T, N, S, R \rangle$

$T = \{\textit{that, this, a, the, man, book, flight, meal, include, read, does}\}$

$N = \{$S, NP, NOM, VP, Det, Noun, Verb, Aux$\}$

$S = S$

$R = \{$

S $\rightarrow$ NP VP
S $\rightarrow$ Aux NP VP
S $\rightarrow$ VP
NP $\rightarrow$ Det NOM
NOM $\rightarrow$ Noun
NOM $\rightarrow$ Noun NOM
VP $\rightarrow$ Verb
VP $\rightarrow$ Verb NP

Det $\rightarrow$ *that* | *this* | *a* | *the*
Noun $\rightarrow$ *book* | *flight* | *meal* | *man*
Verb $\rightarrow$ *book* | *include* | *read*
Aux $\rightarrow$ *does*

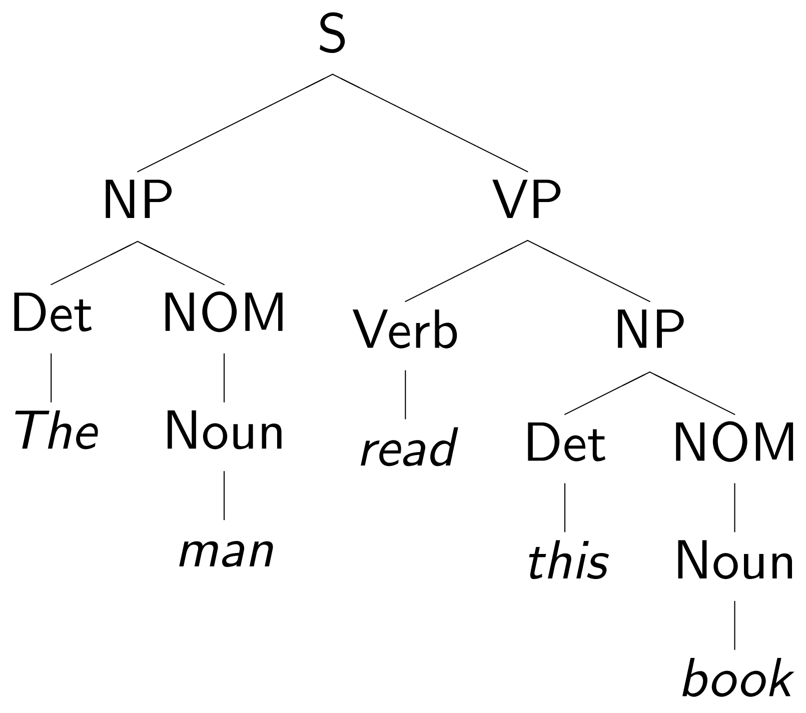
$\}$

Application of grammar rewrite rules

S → NP VP	Det → *that* | *this* | *a* | *the*
S → Aux NP VP	Noun → *book* | *flight* | *meal* | *man*
S → VP	Verb → *book* | *include* | *read*
NP → Det NOM	Aux → *does*
NOM → Noun	
NOM → Noun NOM	
VP → Verb	
VP → Verb NP	

S → NP VP
→ Det NOM VP
→ *The* NOM VP
→ *The* Noun VP
→ *The man* VP
→ *The man* Verb NP
→ *The man read* NP
→ *The man read* Det NOM
→ *The man read this* NOM
→ *The man read this* Noun
→ *The man read this book*

Parse tree



CFGs can capture recursion

Example of seemingly endless recursion of embedded prepositional phrases:
PP $\rightarrow$ Prep NP
NP $\rightarrow$ Noun PP

$[_S$ The mailman ate his $[_{NP}$ lunch $[_{PP}$ with his friend $[_{PP}$ from the cleaning staff $[_{PP}$ of the building $[_{PP}$ at the intersection $[_{PP}$ on the north end $[_{PP}$ of town]]]]]]].

(Bracket notation)

Grammaticality

A CFG defines a formal language = the set of all sentences (strings of words) that can be derived by the grammar.

Sentences in this set said to be **grammatical**.

Sentences outside this set said to be **ungrammatical**.

The Chomsky hierarchy

- Type 0 Languages / Grammars
  Rewrite rules $\alpha \rightarrow \beta$
  where α and β are any string of terminals and nonterminals

- Context-sensitive Languages / Grammars
  Rewrite rules $\alpha X \beta \rightarrow \alpha \gamma \beta$
  where X is a non-terminal, and α, β, γ are any string of terminals and nonterminals, (γ must be non-empty).

- Context-free Languages / Grammars
  Rewrite rules $X \rightarrow \gamma$
  where X is a nonterminal and γ is any string of terminals and nonterminals

- Regular Languages / Grammars
  Rewrite rules $X \rightarrow \alpha Y$
  where X, Y are single nonterminals, and α is a string of terminals; Y might be missing.

Parsing regular grammars

(Languages that can be generated by finite-state automata.)
Finite state automaton $\leftrightarrow$ regular expression $\leftrightarrow$ regular grammar

Space needed to parse: constant

Time needed to parse: linear (in the length of the input string)

Cannot do embedded recursion, e.g. $a^n b^n$. (Context-free grammars can.)
In the language: ab, aaabbb; not in the language: aabbb

The cat likes tuna fish.
The cat the dog chased likes tuna fish
The cat the dog the boy loves chased likes tuna fish.

John, always early to rise, even after a sleepless night filled with the cries of the neighbor's baby, goes running every morning.

John and Mary, always early to rise, even after a sleepless night filled with the cries of the neighbor's baby, go running every morning.

Parsing context-free grammars

(Languages that can be generated by pushdown automata.)

Widely used for surface syntax description (correct word order specification) in natural languages.

Space needed to parse: stack (sometimes a stack of stacks)
In general, proportional to the number of levels of recursion in the data.

Time needed to parse: in general $O(n^3)$.

Can do $a^n b^n$, but cannot do $a^n b^n c^n$.

Chomsky Normal Form

All rules of the form $X \rightarrow YZ$ or $X \rightarrow a$ or $S \rightarrow \epsilon$.
(S is the only non-terminal that can go to ϵ.)
Any CFG can be converted into this form.

How would you convert the rule $W \rightarrow XYaZ$ to Chomsky Normal Form?

Chomsky Normal Form Conversion

These steps are used in the conversion:

1. Make S non-recursive

2. Eliminate all epsilon except the one in S (if there is one)

3. Eliminate all chain rules

4. Remove useless symbols (the ones not used in any production).

How would you convert the following grammar?
$S \rightarrow ABS$
$S \rightarrow \epsilon$
$A \rightarrow \epsilon$
$A \rightarrow xyz$
$B \rightarrow wB$
$B \rightarrow v$

Bottom-up versus Top-down science

- **empiricist**
  Britain: Francis Bacon, John Locke
  Knowledge is induced and reasoning proceeds based on data from the real world.

- **rationalist**
  Continental Europe: Descartes
  Learning and reasoning is guided by prior knowledge and innate ideas.

What is parsing?

We want to run the grammar backwards to find the structure.

Parsing can be viewed as a search problem.

We search through the legal rewritings of the grammar.
We want to find *all* structures matching an input string of words (for the moment)

We can do this bottom-up or top-down
This distinction is independent of depth-first versus breadth-first; we can do either both ways.

Recognizers and parsers

- A **recognizer** is a program for which a given grammar and a given sentence returns YES if the sentence is accepted by the grammar (i.e., the sentence is in the language), and NO otherwise.

- A **parser** in addition to doing the work of a recognizer also returns the set of parse trees for the string.

Soundness and completeness

- A parser is **sound** if every parse it returns is valid/correct.
- A parser **terminates** if it is guaranteed not to go off into an infinite loop.
- A parser is **complete** if for any given grammar and sentence it is sound, produces every valid parse for that sentence, and terminates.
- (For many cases, we settle for sound but incomplete parsers: e.g. probabilistic parsers that return a k-best list.)

Top-down parsing

Top-down parsing is goal-directed.

- A top-down parser starts with a list of constituents to be built.
- It rewrites the goals in the goal list by matching one against the LHS of the grammar rules,
- and expanding it with the RHS,
- ...attempting to match the sentence to be derived.

If a goal can be rewritten in several ways, then there is a choice of which rule to apply (search problem)

Can use depth-first or breadth-first search, and goal ordering.

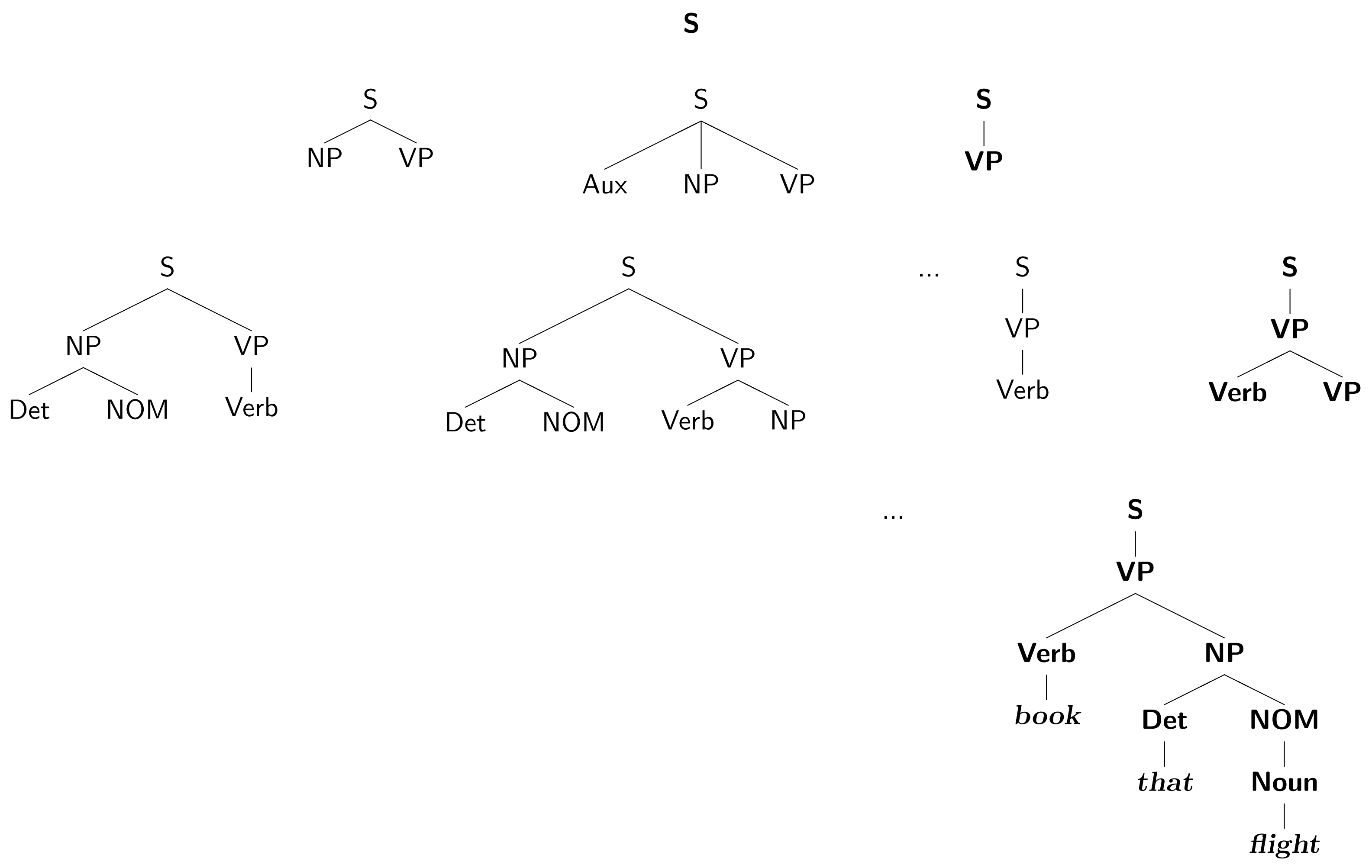
Top-down parsing example (Breadth-first)

S $\longrightarrow$ NP VP	Det $\longrightarrow$ *that* | *this* | *a* | *the*
S $\longrightarrow$ Aux NP VP	Noun $\longrightarrow$ *book* | *flight* | *meal* | *man*
S $\longrightarrow$ VP	Verb $\longrightarrow$ *book* | *include* | *read*
NP $\longrightarrow$ Det NOM	Aux $\longrightarrow$ *does*
NOM $\longrightarrow$ Noun	
NOM $\longrightarrow$ Noun NOM	
VP $\longrightarrow$ Verb	
VP $\longrightarrow$ Verb NP	

Book that flight.

(Work out top-down, breadth-first search on the board...)

Top-down parsing example (Breadth-first)



Problems with top-down parsing

- Left recursive rules... e.g. NP $\rightarrow$ NP PP... lead to infinite recursion
- Will do badly if there are many different rules for the same LHS. Consider if there are 600 rules for S, 599 of which start with NP, but one of which starts with a V, and the sentence starts with a V.
- Useless work: expands things that are possible top-down but not there (no bottom-up evidence for them).
- Top-down parsers do well if there is useful grammar-driven control: search is directed by the grammar.
- Top-down is hopeless for rewriting parts of speech (preterminals) with words (terminals). In practice that is always done bottom-up as lexical lookup.
- Repeated work: anywhere there is common substructure.

Bottom-up parsing

Bottom-up parsing is data-directed.

- The initial goal list of a bottom-up parser is the string to be parsed.
- If a sequence in the goal list matches the RHS of a rule, then this sequence may be replaced by the LHS of the rule.
- Parsing is finished when the goal list contains just the start symbol.

If the RHS of several rules match the goal list, then there is a choice of which rule to apply (search problem)

Can use depth-first or breadth-first search, and goal ordering.

The standard presentation is as **shift-reduce** parsing.

Bottom-up parsing example

S $\longrightarrow$ NP VP	Det $\longrightarrow$ *that* | *this* | *a* | *the*
S $\longrightarrow$ Aux NP VP	Noun $\longrightarrow$ *book* | *flight* | *meal* | *man*
S $\longrightarrow$ VP	Verb $\longrightarrow$ *book* | *include* | *read*
NP $\longrightarrow$ Det NOM	Aux $\longrightarrow$ *does*
NOM $\longrightarrow$ Noun	
NOM $\longrightarrow$ Noun NOM	
VP $\longrightarrow$ Verb	
VP $\longrightarrow$ Verb NP	

Book that flight.

(Work out bottom-up search on the board...)

Shift-reduce parsing

Stack	Input remaining	Action
()	Book that flight	shift
(Book)	that flight	reduce, Verb $\rightarrow$ book, (Choice #1 of 2)
(Verb)	that flight	shift
(Verb that)	flight	reduce, Det $\rightarrow$ that
(Verb Det)	flight	shift
(Verb Det flight)		reduce, Noun $\rightarrow$ flight
(Verb Det Noun)		reduce, NOM $\rightarrow$ Noun
(Verb Det NOM)		reduce, NP $\rightarrow$ Det NOM
(Verb NP)		reduce, VP $\rightarrow$ Verb NP
(Verb)		reduce, S $\rightarrow$ V
(S)		SUCCESS!

Ambiguity may lead to the need for backtracking.

Shift Reduce Parser

Start with the sentence to be parsed in an input buffer.

- a "shift" action correponds to pushing the next input symbol from the buffer onto the stack
- a "reduce" action occurrs when we have a rule's RHS on top of the stack. To perform the reduction, we pop the rule's RHS off the stack and replace it with the terminal on the LHS of the corresponding rule.

(When either "shift" or "reduce" is possible, choose one arbitrarily.)

If you end up with only the START symbol on the stack, then success!
If you don't, and you cannot and no "shift" or "reduce" actions are possible, backtrack.

Shift Reduce Parser

In a top-down parser, the main decision was which production rule to pick.
In a bottom-up shift-reduce parser there are two decisions:

1. Should we shift another symbol, or reduce by some rule?
2. If reduce, then reduce by which rule?

both of which can lead to the need to backtrack

Problems with bottom-up parsing

- Unable to deal with empty categories: termination problem, unless rewriting empties as constituents is somehow restricted (but then it's generally incomplete)

- Useless work: locally possible, but globally impossible.

- Inefficient when there is great lexical ambiguity (grammar-driven control might help here). Conversely, it is data-directed: it attempts to parse the words that are there.

- Repeated work: anywhere there is common substructure.

- Both Top-down (LL) and Bottom-up (LR) parsers can (and frequently do) do work exponential in the sentence length on NLP problems.

Principles for success

- Left recursive structures must be found, not predicted.
- Empty categories must be predicted, not found.
- Don't waste effort re-working what was previously parsed before backtracking.

An alternative way to fix things:

- Grammar transformations can fix both left-recursion and epsilon productions.
- Then you parse the same language but with different trees.
- BUT linguists tend to hate you, because the structure of the re-written grammar isn't what they wanted.

Coming next...

A dynamic programming solution for parsing: CYK (and maybe also Earley's Algorithm).

(Then later in the semester.)
Probabilistic version of these models. Find the most *likely* parse when several are possible.