# Context Free Grammars

Introduction to Natural Language Processing
CS 585
Andrew McCallum

March 23, 2004

# Administration

- Handing back Midterm today.
  Answer questions in class.

- According to syllabus, “One paragraph progress report due today.” Everyone, please email it to me and Aron <culotta@cs.umass.edu>. Anytime before midnight tonight will be fine.

- Project proposal presentations: Five minutes in front of class explaining: (a) the problem, (b) why it is interesting from an NLP point of view, (c) your approach for a solution, (d) evaluation plan, (e) intended milestones with dates.

  Slide transparencies would be good.
  Date changed from April 1 to April 6.
  These points should also be covered in your 1-page proposals, due April 1st.

# 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.

# 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*

# Evidence constituency exists

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

2. The constituent can be placed in a number of different locations
   Consituent = Prepositional phrase: *On December twenty-sixth*
   *On December twenty-sixth I'd like to fly to Florida.*
   *I'd like to fly on December twenty-sixth to Florida.*
   *I'd like to fly to Florida on December twenty-sixth.*
   But not split apart
   **On December I'd like to fly twenty-sixth to Florida.*
   **On I'd like to fly 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 (1990), 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 $\gamma$ is a sequence of terminals and nonterminals (may be empty).
- A grammar $G$ generates a language $L$.

# An example context-free grammar

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

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

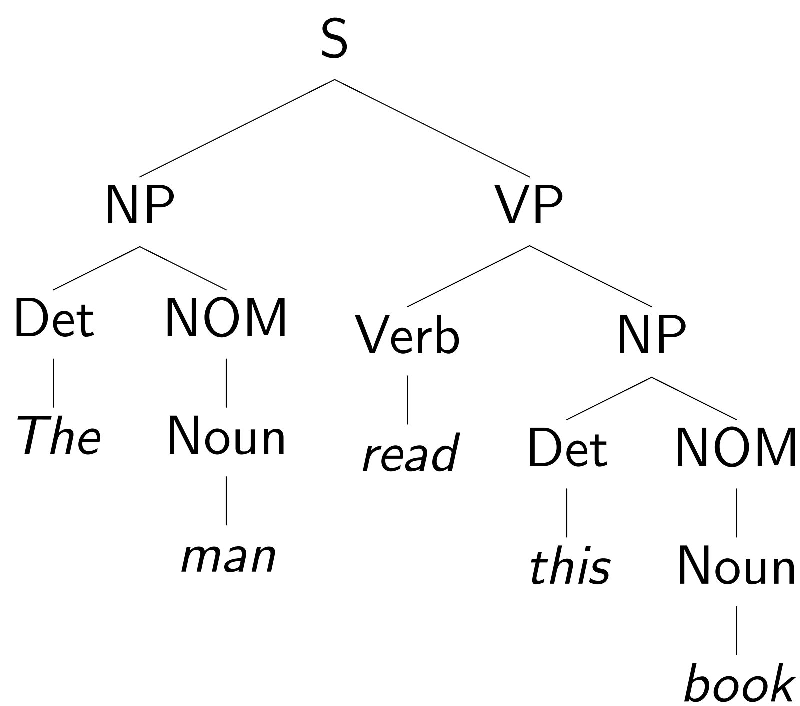
$\}$

# Application of grammar rewrite rules

| | |
|---|---|
| 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 | |

S $\longrightarrow$ NP VP
$\longrightarrow$ Det NOM VP
$\longrightarrow$ *The* NOM VP
$\longrightarrow$ *The* Noun VP
$\longrightarrow$ *The man* VP
$\longrightarrow$ *The man* Verb NP
$\longrightarrow$ *The man read* NP
$\longrightarrow$ *The man read* Det NOM
$\longrightarrow$ *The man read this* NOM
$\longrightarrow$ *The man read this* Noun
$\longrightarrow$ *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 shepherds ate their [$_{NP}$ sandwiches [$_{PP}$ of cheese [$_{PP}$ from the cows [$_{PP}$ of the farm [$_{PP}$ by the fork [$_{PP}$ in the river]]]]]]].

(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 $\alpha \to \beta$
  where $\alpha$ and $\beta$ are any string of terminals and nonterminals

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

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

- Regular Languages / Grammars
  Rewrite rules $X \to \alpha Y$
  where $X, Y$ are single nonterminals, and $\alpha$ 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.)
ab, aaabbb, *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.

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

<u>John and Mary</u>, always early to rise, even after a sleepless night filled with the cries of the neighbor's baby, <u>go</u> 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 to $a^n b^n$, but cannot do $a^n b^n c^n$.

## Chomsky Normal Form

All rules of the form $X \rightarrow YZ$ or $X \rightarrow a$.
Any CFG can be converted into this form.

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

## Parsing context-sensitive grammars

(Languages that can be recognized by a non-deterministic Turing machine whose tape is bounded by a constant times the length of the input.)

Natural languages are really not context-free: e.g. pronouns more likely in Object rather than Subject of a sentence.

But parsing is PSPACE-complete! (Recognized by a Turing machine using a polynomial amount of memory, and unlimited time.)

Often work with *mildly* context-sensitive grammars. More on this next week. E.g. Tree-adjoining grammars. Time needed to parse, e.g. $O(n^6)$ or $O(n^5)$...

# 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.
Doing this we build a *search tree* which is different from the *parse tree*.

# Recognizers and parsers

- A **recognizer** is a program for which a given grammar and a given sentence returns YES if teh 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.

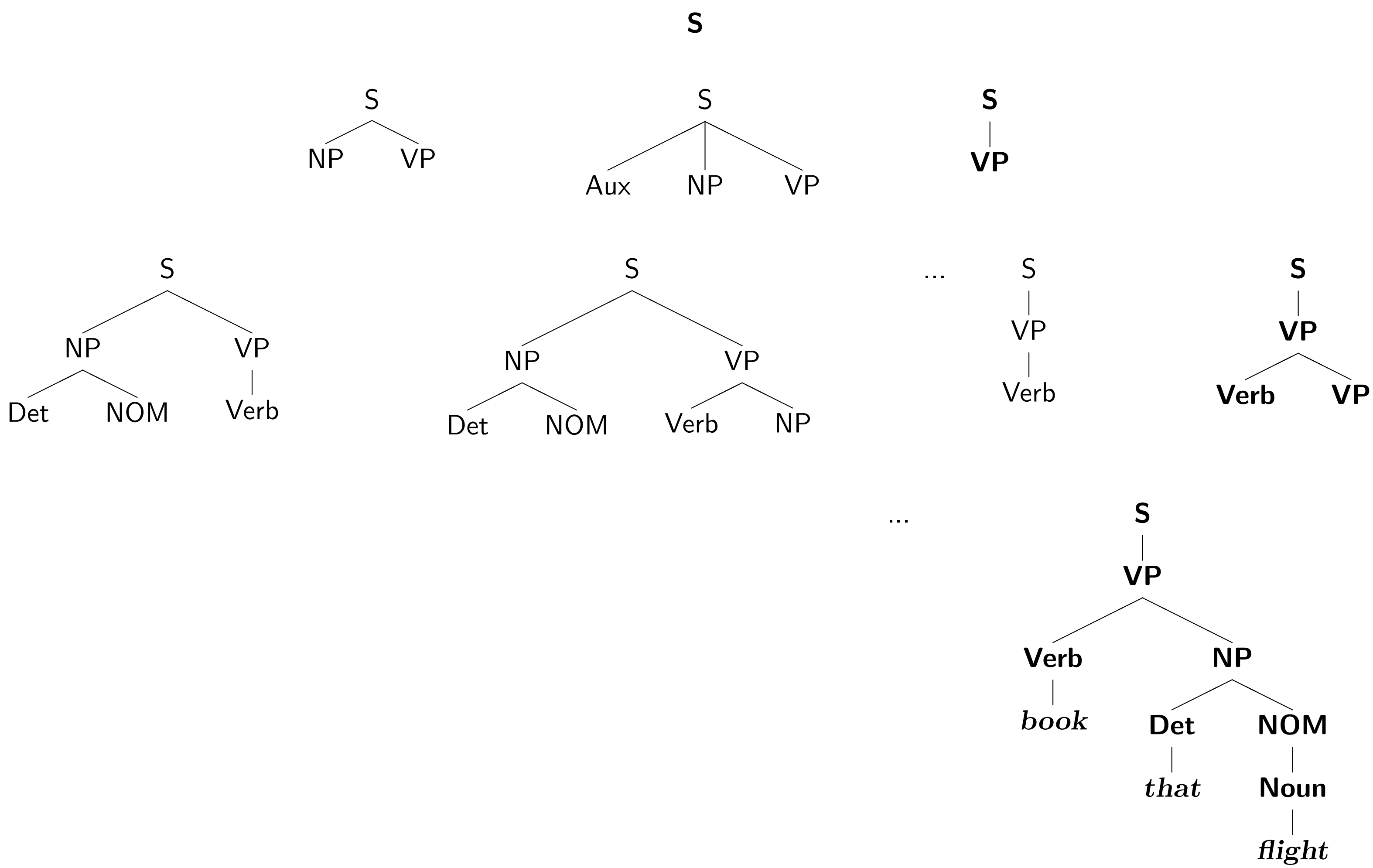
# Top-down parsing example (Breadth-first)

| | |
|---|---|
| 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

Top-down 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 |
| (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.

# 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: Earley's Algorithm.

Hey, where are all those probabilities we love to much? :-)
Probabilistic version of these models. Find the most $likely$ parse.

The definition of context free grammars (CFGs) allows us to develop a wide variety of grammars. Most of the time, some of the productions of CFGs are not useful and are redundant. This happens because the definition of CFGs does not restrict us from making these redundant productions. By simplifying CFGs we remove all these redundant productions from a grammar , while keeping the transformed grammar equivalent to the original grammar. Context-Free Grammars. A grammar is a set of rules for putting strings together and so corresponds to a language. Grammars. A grammar consists of: â€¢ a set of variables (also called nonterminals)Â  We focus on a special version of grammars called a context-free grammar (CFG). A language is context-free if it is generated by a CFG. Goddard 6a: 4. Example Continued. Context-Free Grammars for English. From: Chapter 12 of An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, by Daniel Jurafsky and James H. Martin. Overview.