General Context-Free Grammar Parsing: A phrase structure grammar

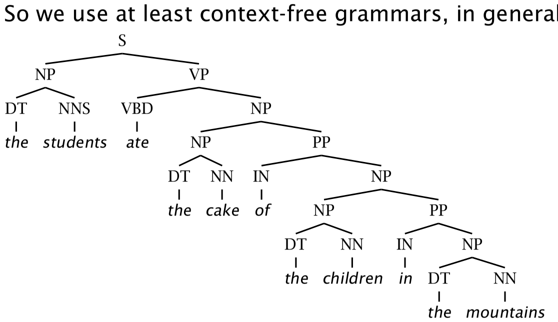
- Also known as a context-free grammar (CFG)
- $$\begin{array}{lll@{\qquad}lll} S & \to & NP\ VP & DT & \to & \textit{the} \\ NP & \to & \left\{\begin{array}{l} DT\ NNS \\ DT\ NN \\ NP\ PP \end{array}\right\} & NNS & \to & \left\{\begin{array}{l} \textit{children} \\ \textit{students} \\ \textit{mountains} \end{array}\right\} \\ VP & \to & \left\{\begin{array}{l} VP\ PP \\ VBD \\ VBD\ NP \end{array}\right\} & VBD & \to & \left\{\begin{array}{l} \textit{slept} \\ \textit{ate} \\ \textit{saw} \end{array}\right\} \\ PP & \to & IN\ NP & IN & \to & \left\{\begin{array}{l} \textit{in} \\ \textit{of} \end{array}\right\} \\ & & & NN & \to & \textit{cake} \end{array}$$

Application of grammar rewrite rules

- S
  → NP VP
  → DT NNS VBD
  → *The children slept*
- S
  → NP VP
  → DT NNS VBD NP
  → DT NNS VBD DT NN
  → *The children ate the cake*

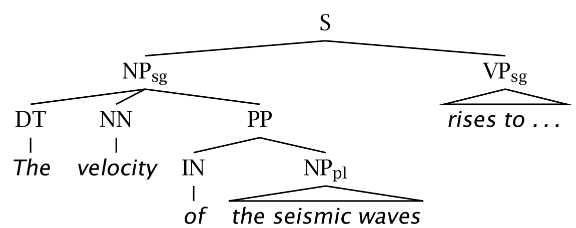
Phrase structure is recursive

So we use at least context-free grammars, in general



Why we need recursive phrase structure

- Kupiec (1992): Sometimes HMM tagger goes awry: *waves* → verb
- *The velocity of the seismic waves rises to . . .*



- Language model: There are similar problems.
  The captain of the ship yelled out.

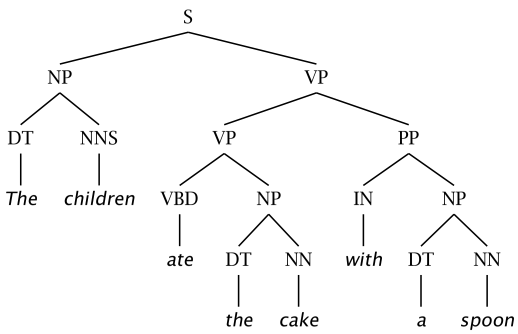
Why we need phrase structure (2)

- Syntax gives important clues in information extraction tasks and some cases of named entity recognition
- *We have recently demonstrated that stimulation of [*$_{\text{CELLTYPE}}$*human T and natural killer cells] with [*$_{\text{PROTEIN}}$*IL-12] induces tyrosine* ***phosphorylation*** *of the [*$_{\text{PROTEIN}}$*Janus family tyrosine kinase] [*$_{\text{PROTEIN}}$*JAK2] and [*$_{\text{PROTEIN}}$*Tyk2].*
- Things that are the object of *phosphorylate* are likely proteins.

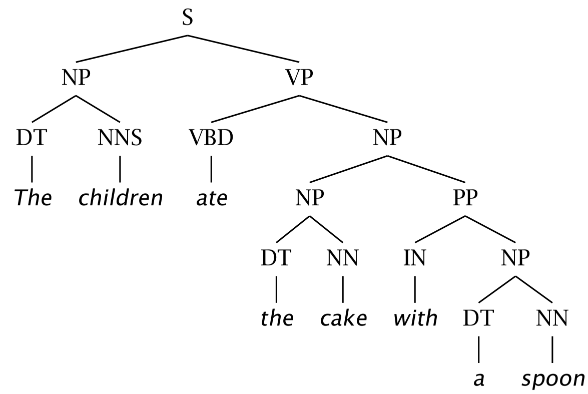
Constituency

- Phrase structure organizes words into nested *constituents*.
- How do we know what is a constituent? (Not that linguists don't argue about some cases.)
  - Distribution: behaves as a unit that appears in different places:
    - *John talked [to the children] [about drugs].*
    - *John talked [about drugs] [to the children].*
    - **John talked drugs to the children about*
  - Substitution/expansion/pro-forms:
    - *I sat [on the box/right on top of the box/there].*
  - Coordination, no intrusion, fragments, semantics, . . .

Natural language grammars are ambiguous: Prepositional phrase attaching to verb



Prepositional phrase attaching to noun



Attachment ambiguities in a real sentence

The board approved [its acquisition] [by Royal Trustco Ltd.]
[of Toronto]
[for $27 a share]
[at its monthly meeting].

Penn Treebank Sentences: an example

```
( (S
    (NP-SBJ (DT The) (NN move))
    (VP (VBD followed)
      (NP
        (NP (DT a) (NN round))
        (PP (IN of)
          (NP
            (NP (JJ similar) (NNS increases))
            (PP (IN by)
              (NP (JJ other) (NNS lenders)))
            (PP (IN against)
              (NP (NNP Arizona) (JJ real) (NN estate) (NNS loans))))))
      (, ,)
      (S-ADV
        (NP-SBJ (-NONE- *))
        (VP (VBG reflecting)
          (NP
            (NP (DT a) (VBG continuing) (NN decline))
            (PP-LOC (IN in)
              (NP (DT that) (NN market)))))))
    (. .)))
```


Ambiguity

- Programming language parsers resolve local ambiguities with lookahead
- Natural languages have global ambiguities:
  - *I saw that gasoline can explode*
- What is the size of embedded NP?

What is parsing?

- We want to run the grammar backwards to find the structures
- Parsing can be viewed as a search problem
- Parsing is a hidden data problem
- We search through the legal rewritings of the grammar
- We want to examine *all* structures for a string of words (for the moment)
- We can do this bottom-up or top-down
  - This distinction is independent of depth-first/bread-first etc. – we can do either both ways
  - Doing this we build a *search tree* which is different from the *parse tree*

Human parsing

- Humans often do ambiguity maintenance
  - *Have the police . . . eaten their supper?*
  - *come in and look around.*
  - *taken out and shot.*
- But humans also commit early and are “garden pathed”:
  - *The man who hunts ducks out on weekends.*
  - *The cotton shirts are made from grows in Mississippi.*
  - *The horse raced past the barn fell.*

State space search

- States:
- Operators:
- Start state:
- Goal test:
- *Algorithm*

```
stack = { startState }
solutions = {}
loop
      if stack is empty, return solutions
      state = remove-front(stack)
      if goal(state) push(state, solutions)
      stack = pushAll(expand(state, operators), stack)
end
```


Another phrase structure grammar

S	→	NP VP	N	→	*cats*
VP	→	V NP	N	→	*claws*
VP	→	V NP PP	N	→	*people*
NP	→	NP PP	N	→	*scratch*
NP	→	N	V	→	*scratch*
NP	→	e	P	→	*with*
NP	→	N N	PP	→	P NP

(By linguistic convention, S is the start symbol, but in the PTB, we use the unlabeled node at the top, which can rewrite various ways.)

cats scratch people with claws

S					
NP	VP				
NP	PP	VP			*3 choices*
NP	PP	PP	VP		
oops!					
N	VP				
cats	VP				
cats	V	NP			*2 choices*
cats	scratch	NP			
cats	scratch	N			*3 choices – showing 2nd*
cats	scratch	people			*oops!*
cats	scratch	NP	PP		
cats	scratch	N	PP		*3 choices – showing 2nd . . .*
cats	scratch	people	with	claws	

Phrase Structure (CF) Grammars

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

- T is set of terminals
- N is set of nonterminals
  - 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

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 to not 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 purposes, 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. The top-down parser 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.

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 category.
- 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.

Problems with top-down parsing

- Left recursive rules
- A top-down parser 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 V, and the sentence starts with V.
- Useless work: expands things that are possible top-down but not there
- 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

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 TD (LL) and BU (LR) parsers can (and frequently do) do work exponential in the sentence length on NLP problems.

Principles for success: what one needs to do

- If you are going to do parsing-as-search with a grammar as is:
  - Left recursive structures must be found, not predicted
  - Empty categories must be predicted, not found
- Doing these things doesn't fix the repeated work problem.

An alternative way to fix things

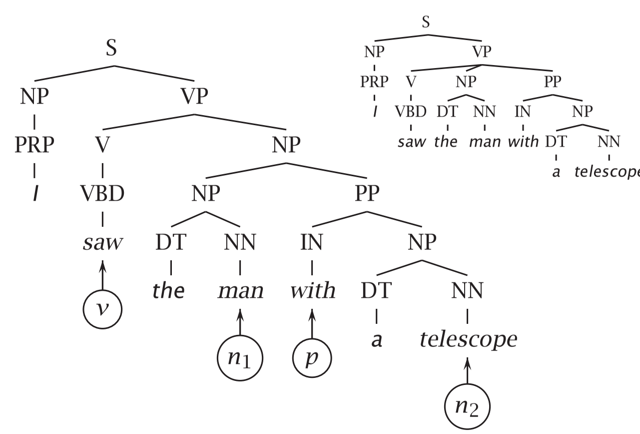
- Grammar transformations can fix both left-recursion and epsilon productions
- Then you parse the same language but with different trees
- Linguists tend to hate you
  - ◻ But this is a misconception: they shouldn't
  - ◻ You can fix the trees post hoc

A second way to fix things

- Rather than doing parsing-as-search, we do parsing as dynamic programming
- This is the most standard way to do things
- It solves the problem of doing repeated work
- But there are also other ways of solving the problem of doing repeated work
  - ◻ Memoization (remembering solved subproblems)
  - ◻ Doing graph-search rather than tree-search.

Attachment ambiguities: The key parsing decision

- The main problem in parsing is working out how to 'attach' various kinds of constituents – PPs, adverbial or participial phrases, coordinations, and so on
- Prepositional phrase attachment
  - ◻ *I saw the man with a telescope*
- What does *with a telescope* modify?
  - ◻ The verb *saw*?
  - ◻ The noun *man*?
- Is the problem 'AI-complete'? Yes, but . . .



Attachment ambiguities (2)

- Proposed simple structural factors
  - ◻ Right association (Kimball 1973) = 'low' or 'near' attachment = 'late closure' (of NP) [NP → NP PP]
  - ◻ Minimal attachment (Frazier 1978) [depends on grammar] = 'high' or 'distant' attachment = 'early closure' (of NP) [VP → V NP PP]
- Such simple structural factors dominated in early psycholinguistics, and are still widely invoked.
- In the V NP PP context, right attachment gets it right in 55–67% of cases.
- But that means it gets it wrong in 33–45% of cases

Importance of lexical factors

- Words are good predictors (or even inducers) of attachment (even absent understanding):
  - ◻ The children ate the cake with a spoon.
  - ◻ The children ate the cake with frosting.
  - ◻ Moscow sent more than 100,000 soldiers into Afghanistan
  - ◻ Sydney Water breached an agreement with NSW Health
- Ford et al. (1982):
  - ◻ Ordering is jointly determined by strengths of alternative lexical forms, alternative syntactic rewrite rules, and the sequence of hypotheses in parsing

PCFGs

A PCFG G consists of the usual parts of a CFG

- A set of terminals, $\{w^k\}, k = 1, \ldots, V$
- A set of nonterminals, $\{N^i\}, i = 1, \ldots, n$
- A designated start symbol, N^1
- A set of rules, $\{N^i \to \zeta^j\}$, (where ζ^j is a sequence of terminals and nonterminals)

and

- A corresponding set of probabilities on rules such that:

$$\forall i \quad \sum_j P(N^i \to \zeta^j) = 1$$

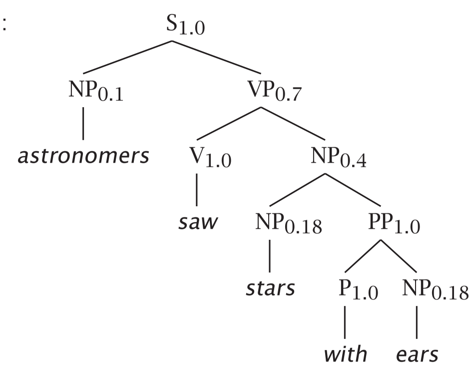
PCFG probability of a string

$$\begin{aligned} P(w_{1n}) &= \sum_t P(w_{1n}, t) \quad t \text{ a parse of } w_{1n} \\ &= \sum_{\{t:\text{yield}(t)=w_{1n}\}} P(t) \end{aligned}$$

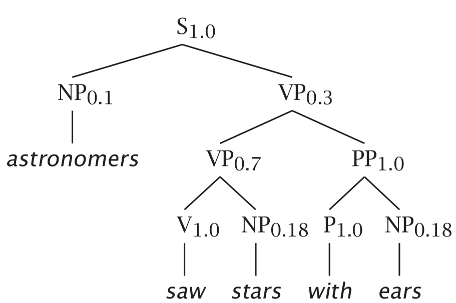
A simple PCFG (in CNF)

S → NP VP	1.0	NP → NP PP	0.4
PP → P NP	1.0	NP → *astronomers*	0.1
VP → V NP	0.7	NP → *ears*	0.18
VP → VP PP	0.3	NP → *saw*	0.04
P → *with*	1.0	NP → *stars*	0.18
V → *saw*	1.0	NP → *telescopes*	0.1

t_1:



t_2:



The two parse trees' probabilities and the sentence probability

$$\begin{aligned} P(t_1) &= 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \\ &\quad \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\ &= 0.0009072 \\ P(t_2) &= 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \\ &\quad \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\ &= 0.0006804 \\ P(w_{15}) &= P(t_1) + P(t_2) = 0.0015876 \end{aligned}$$