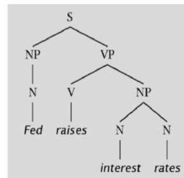
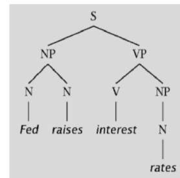
Statistical Natural Language Parsing

Gerald Penn
[based on slides by Christopher Manning]

Parsing examples

- Fed raises interest rates





Parsing in the early 1990s

- The parsers produced detailed, linguistically rich representations
- Parsers had uneven and usually rather poor coverage
  - E.g., 30% of sentences received no analysis
- Even quite simple sentences had many possible analyses
  - Parsers either had no method to choose between them or a very ad hoc treatment of parse preferences
- Parsers could not be learned from data
- Parser performance usually wasn't or couldn't be assessed quantitatively and the performance of different parsers was often incommensurable

Statistical parsing

- Many would ascribe the relative success of parsers since the early 90s to the advent of statistics, but there were several other important developments, e.g. the availability of part-of-speech taggers and parse-annotated corpora, as well as the awareness that getting all the parses quickly wasn't as important as getting the *right* parse.
- Statistical parsers do more than disambiguate – they are also robst: they assign some parse to literally every input string.
- Parsing is now highly commoditized, but parsers are still improving year-on-year.
  - Collins (C) or Bikel reimplementation (Java)
  - Charniak or Johnson-Charniak parser (C++)
  - Stanford Parser (Java)

Statistical parsing applications

- High precision question answering systems (Pasca and Harabagiu SIGIR 2001)
- Improving biological named entity extraction (Finkel et al. JNLPBA 2004):
- Syntactically based sentence compression (Lin and Wilbur *Inf. Retr.* 2007)
- Extracting people's opinions about products (Bloom et al. NAACL 2007)
- Improved interaction in computer games (Gorniak and Roy, AAAI 2005)
- Helping linguists find data (Resnik et al. BLS 2005)

Ambiguity: natural languages vs. programming languages

- Programming languages have only local ambiguities, which a parser can resolve with lookahead (and conventions)
- Natural languages have global ambiguities
  - *I saw that gasoline can explode*
  - "Construe an **else** statement with which **if** makes most sense."

Classical NLP Parsing

- Wrote symbolic grammar and lexicon
  - S → NP VP　　　NN → *interest*
  - NP → (DT) NN　　　NNS → *rates*
  - NP → NN NNS　　　NNS → *raises*
  - NP → NNP　　　VBP → *interest*
  - VP → V NP　　　VBZ → *rates*
- Was hamstrung by the 1980s *Zeitgeist* of encoding this as a deductive proof search.
- Looking for all parses scaled badly and didn't help coverage
  - Minimal grammar on "Fed raises" sentence: 36 parses
  - Simple 10 rule grammar: 592 parses
  - Real-size broad-coverage grammar: millions of parses

Classical NLP Parsing: The problem and its solution

- Very constrained grammars attempted to limit unlikely/weird parses for sentences
- But the underlying method made that both difficult and a trade-off relative to coverage, i.e., some sentences can wind up with no parses.
- Solution: There needs to be an explicit mechanism that allows us to rank how likely each of the parses is
  - Statistical parsing lets us work with very loose grammars that admit millions of parses for sentences but to still quickly find the best parse(s)

The rise of annotated data: The Penn Treebank

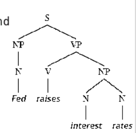
```
( (S
    (NP-SBJ (DT The) (NN move))
    (VP (VBD followed)
      (NP
        (NP (DT a) (NN round))
        (PP (IN of)
          (NP
            (NP (JJ similar) (NNS increases))
            (PP (IN by)
              (NP (JJ other) (NNS lenders)))
            (PP (IN against)
              (NP (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)))))))
    (. .)))
```


The rise of annotated data

- Starting off, building a treebank seems a lot slower and less useful than building a grammar
- But a treebank gives us many things
  - Reusability of the labor
  - Broad coverage (up to the corpus, at least)
  - "Analysis in context" - probably a better way to think about grammar anyway
  - Frequencies and distributional information
  - A way to evaluate systems

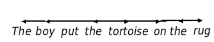
Two views of linguistic structure: 1. Constituency (phrase structure)

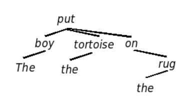
- Phrase structure organizes words into nested *constituents*.
- How do we know what is a constituent?
  - Good question: it's a goofy admixture of observed constraints on word order and semantic interpretability – we often have to ask linguists, and even they don't always agree
  - Distribution: a constituent behaves as a unit that can appear 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, regular internal structure, no intrusion, fragments, semantics, …



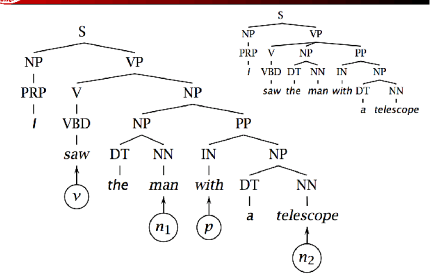
Two views of linguistic structure: 2. Dependency structure

- Dependency structure shows which words depend on (modify or are arguments of) which other words.
- This is often goofy in its own way, in that it ofte fails to bridge the gap between "who did what to whom" slot-filling and a structure that can guide us towards the composition of the logical forms that philosophers of language have traditionally assigned to these sentences as "meanings."





Attachment ambiguities: Two possible PP attachments



Attachment ambiguities

- The key parsing decision: How do we 'attach' various kinds of constituents – PPs, adverbial or participial phrases, coordinations, etc.
- 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

- Proposed simple structural factors
  - Right association (Kimball 1973) = 'low' or 'near' attachment = 'early closure' (of NP)
  - Minimal attachment (Frazier 1978). Effects depend on grammar, but gave 'high' or 'distant' attachment = 'late closure' (of NP) under the assumed model
- Which is right?
- Such simple structural factors dominated in early psycholinguistics (and are still widely invoked).
- In the V NP PP context, right attachment usually gets right 55–67% of cases.
- But that means it gets wrong 33–45% of cases.

Attachment ambiguities

- Words are good predictors 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 …

The importance of lexical factors

- Ford, Bresnan, and Kaplan (1982) [promoting 'lexicalist' linguistic theories] argued:
  - Order of grammatical rule processing [by a person] determines closure effects
  - Ordering is jointly determined by strengths of alternative lexical forms, strengths of alternative syntactic rewrite rules, and the sequences of hypotheses in the parsing process.
  - "It is quite evident, then, that the closure effects in these sentences are induced in some way by the choice of the lexical items." (Psycholinguistic studies show that this is true *independent of discourse context.*)

A simple prediction

- Use a likelihood ratio:
  - E.g., $$LR(v,n,p)=\frac{P(p|v)}{P(p|n)}$$

  - P(*with*|*agreement*) = 0.15
  - P(*with*|*breach*) = 0.02

  - LR(*breach, agreement, with*) = 0.13
    → Choose noun attachment

A problematic example

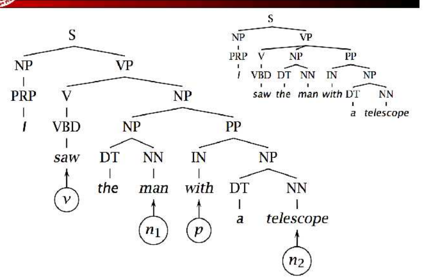
- *Chrysler confirmed that it would end its troubled venture with Maserati.*
- Should be a noun attachment but get wrong answer:
  - *w* *C(w)* *C(w, with)*
  - *end* 5156 607
  - *venture* 1442 155

$$P(with|v)=\frac{607}{5156}\text{»}0.118>P(with|n)=\frac{155}{1442}\text{»}0.107$$

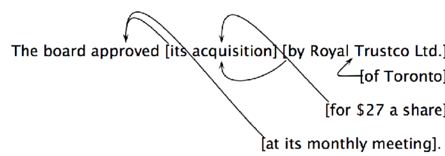
A problematic example

- What might be wrong here?
  - If you see a V NP PP sequence, then for the PP to attach to the V, then it must also be the case that the NP doesn't have a PP (or other postmodifier)
    - Since, except in extraposition cases, such dependencies can't cross
- Parsing allows us to factor in and integrate such constraints.

A better predictor would use n_2 as well as v, n_1, p



Attachment ambiguities in a real sentence



- Catalan numbers
  - $C_n = (2n)!/[(n+1)!n!]$
- An exponentially growing series, which arises in many tree-like contexts:
  - E.g., the number of possible triangulations of a polygon with n+2 sides

What is parsing?

- We want to run a grammar backwards to find possible structures for a sentence
- Parsing can be viewed as a search problem
- Parsing is a hidden data problem
- For the moment, we want to examine *all* structures for a string of words
- We can do this bottom-up or top-down
  - This distinction is independent of depth-first or breadth-first search – we can do either both ways
  - We search by building a *search tree* which his distinct from the *parse tree*

A 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 convention, S is the start symbol, but in the PTB, we have an extra node at the top (ROOT, TOP)

Phrase structure grammars = context-free grammars

- G = (T, N, S, R)
  - T is set of terminals
  - N is set of nonterminals
    - For NLP, we usually distinguish out a set P ⊂ N of preterminals, which always rewrite as terminals
    - S is the start symbol (one of the nonterminals)
    - R is rules/productions of the form X → γ, where X is a nonterminal and γ is a sequence of terminals and nonterminals (possibly an empty sequence)
- A grammar G generates a language L.

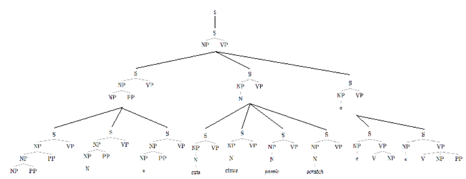
Soundness and completeness

- 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 produces every valid parse for that sentence (on its way to infinity? Can a sentence have infinitely many parses? Are there infinite-length sentences?)
- 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 *goal* 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.

Top-down parsing



Problems with top-down parsing

- Left recursive rules
- Will do badly if there are many 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.
- In general, not goal-oriented enough: expands things that are possible top-down but not there in the sentential input
- 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 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*.

Shift-reduce parsing: one path

	cats scratch people with claws	
cats	*scratch people with claws*	SHIFT
N	*scratch people with claws*	REDUCE
NP	*scratch people with claws*	REDUCE
NP *scratch*	*people with claws*	SHIFT
NP V	*people with claws*	REDUCE
NP V *people*	*with claws*	SHIFT
NP V N	*with claws*	REDUCE
NP V NP	*with claws*	REDUCE
NP V NP *with*	*claws*	SHIFT
NP V NP P	*claws*	REDUCE
NP V NP P *claws*		SHIFT
NP V NP P N		REDUCE
NP V NP P NP		REDUCE
NP V NP PP		REDUCE
NP VP		REDUCE
S		REDUCE

What other search paths are there for parsing this sentence?

Problems with bottom-up parsing

- Empty categories: non-terminals with no words (terminals) under them. This can cause termination problems, unless rewriting empties as constituents is somehow restricted (but then it's generally incomplete)
- Strictly local goal-orientation: locally possible, but globally impossible.
- Inefficient when there is a great deal of 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

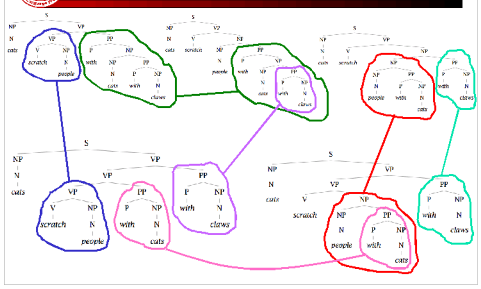
Quiz Question!

- How many distinct parses does the following sentence have due to PP attachment ambiguities?
  - A PP can attach to any preceding V or N within the verb phrase, subject only to the parse still being a tree.
    - (This is equivalent to there being no crossing dependencies, where if d_2 is a dependent of d_1 and d_3 is a dependent of d_2, then the line d_2–d_3 begins at d_2 under the line from d_1 to d_2.)

John wrote the book with a pen in the room.

a) 2 b) 4 c) 6 d) 8

Repeated work…



Principles for success: take 1

- If you are going to do parsing-as-search with a grammar as is:
  - Evidence for left recursive structures must be found, not predicted
  - Presence of empty categories must be predicted, not found
- Doing these things helps with termination but doesn't fix the repeated work problem:
  - Both TD (LL) and BU (LR) parsers can (and frequently do) do work exponentially in the sentence length on NLP problems.

Principles for success: take 2

- Grammar transformations can fix both left-recursion and epsilon productions
- Then you parse the same language but with different trees
- Your linguist-friends will hate you
  - But they don't need to know – you can fix up the trees later so that no one notices what you did
- But the big problem is the global ambiguities – this creates no problems with termination, but can lead to exponentially many parses.

Principles for success: take 3

- Rather than doing parsing-as-enumeration, we do parsing as dynamic programming
- This is the most standard way to do things
  - E.g., CKY parsing
- It solves the problem of doing repeated work
- But there are also other ways of solving the problem of doing repeated work
  - e.g., doing graph-search rather than tree-search.

Human parsing

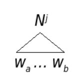
- Humans often do ambiguity maintenance
  - *Have the police … eaten their supper?*
  - *come in and look around.*
  - *taken out and shot.*
- But humans also commit early and are “garden pathed”:
  - *The man who hunts ducks out on weekends.*
  - *The cotton shirts are made from grows in Mississippi.*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

Probabilistic or stochastic context-free grammars (PCFGs)

- G = (T, N, S, R, P)
  - 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 the start symbol (one of the nonterminals)
    - R is rules/productions of the form X $\rightarrow$ γ, where X is a nonterminal and γ is a sequence of terminals and nonterminals (possibly an empty sequence)
    - P(R) gives the probability of each rule
    - For each non-terminal, X: $\sum_{X\rightarrow\gamma\in R} P(X\rightarrow\gamma)=1$
- A grammar G generates a language model L: $\sum_{\gamma\in T^*} P(\gamma)=1$

PCFGs – Notation

- $w_{1n} = w_1 \ldots w_n$ = the word sequence from 1 to n (sentence of length n)
- w_{ab} = the subsequence $w_a \ldots w_b$
- N^j_{ab} = the nonterminal N^j dominating $w_a \ldots w_b$



- We’ll write P($N^i \rightarrow \zeta^j$) to mean P($N^i \rightarrow \zeta^j$ | N^i)
- We’ll want to calculate $\max_t$ P($t \Rightarrow^* w_{ab}$)

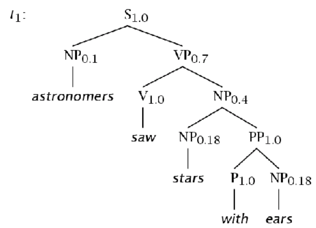
The probability of trees and strings

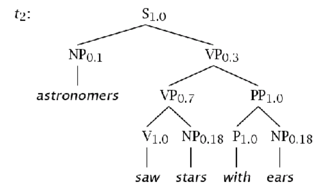
- P(t) -- The probability of tree is the product of the probabilities of the rules used to generate it.
- P(w_{1n}) -- The probability of the string is the sum of the probabilities of the trees which have that string as their yield

P(w_{1n}) = Σ_j P(w_{1n}, t_j) where t_j is a parse of w_{1n}
= Σ_j P(t_j)

A Simple PCFG (in CNF)

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





Tree and String Probabilities

- w_{15} = *astronomers saw stars with ears*
- $P(t_1)$ = 1.0 × 0.1 × 0.7 × 1.0 × 0.4 × 0.18
  × 1.0 × 1.0 × 0.18
  = 0.0009072
- $P(t_2)$ = 1.0 × 0.1 × 0.3 × 0.7 × 1.0 × 0.18
  × 1.0 × 1.0 × 0.18
  = 0.0006804
- $P(w_{15})$ = $P(t_1)$ + $P(t_2)$
  = 0.0009072 + 0.0006804
  = 0.0015876