6.863J Natural Language Processing
Lecture 8: Going nonlinear - Marxist analysis

Instructor: Robert C. Berwick
berwick@ai.mit.edu

The Menu Bar

- Administrivia:
  - Lab 2a/2b due Friday

Agenda:

Going nonlinear: beyond finite-state machines

- Marxist analysis – simple & post-modern
- What: hierarchical representations; constituents, representation
- How: constituent or 'context-free' parsing (next time – how to do it *fast*)
- Why: to extract 'meaning'

Motivation

- What, How, and Why
- <u>What:</u> word *chunks* behave as units, like words or endings (morphemes), like *ing*
- <u>How:</u> we have to recover these from input
- <u>Why:</u> chunks used to discover *meaning*
- Parsing: mapping from *strings* to *structured representation*

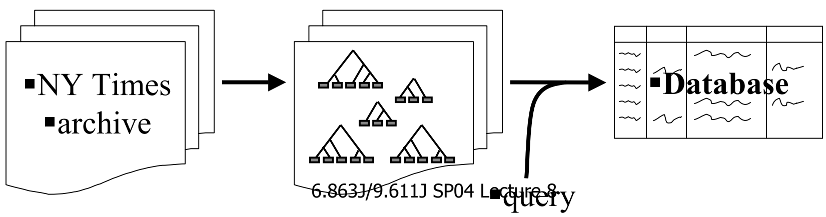
Why parsing?

- A (context-free) grammar tells us what (syntactic)structure(s) we can assign to a string
- It doesn't tell us is how we should go about assigning a string a structure

Applications of parsing

- Grammar checking (Microsoft)
- Indexing for information retrieval (Woods 72-1997)

  ... washing a car with a hose ... → vehicle maintenance
- Information extraction (Keyser, Chomsky '62 to Hobbs 1996)



Why: Q&A systems (lab 4)

```
(top-level)
Shall I clear the database? (y or n) y
>John saw Mary in the park
OK.
>Where did John see Mary
IN THE PARK.
>John gave Fido to Mary
OK.
>Who gave John Fido
I DON'T KNOW
>Who gave Mary Fido
JOHN
>John saw Fido
OK.
>Who did John see
FIDO AND MARY
```

Language & hierarchical structure

- Claim: Most, perhaps <u>all</u> properties in syntax are defined over hierarchical structure
- One needs to parse to see subtle distinctions

More examples: Marxist analysis

- This morning, I shot an elephant in my pajamas
- “How he got into my pajamas, I’ll never know” (G. Marx)

Examples (courtesy Dave Barry)

- National Park Service:
- Avoid the traffic by using a shuttle bus and view the elk rut with a park ranger
- PA *Patriot News:*
- "Smoking organ causes stir at nursing home"

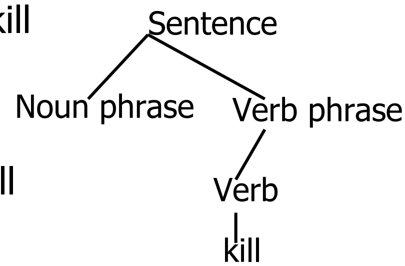
- Where do these come from??
- Visiting relatives can be dangerous/smoking organs can be dangerous

Why: linguistic properties defined over hierarchical structure

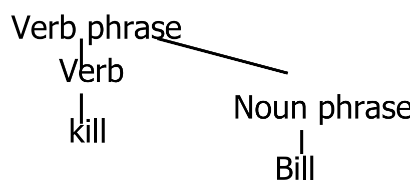
- What <u>are</u> the linguistic properties we need?
- Subject-of, object-of – to get predicate structure
- Scope
- Structural ambiguity (hence multiple meaning)
- All these from syntax

Predication depends on configuration

- Subject-of: Bill-kill



- Object-of: kill-Bill



Configurational properties

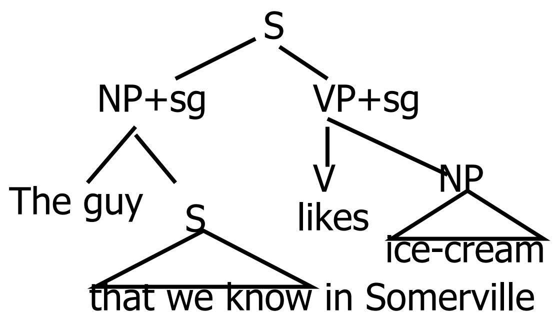
- More sophisticated configurational property

Sara likes her. *(her ≠ Sara)*
Sara thinks that someone likes her. *(her = or ≠ Sara)*
Sara dislikes anyone's criticism of her. *(her = Sara* or *her ≠ Sara)*
Who did John see? → For which x, x a person, likes(Bill, x)

Distinction here is based on *hierarchical structure* = scope in natural language

Why: express ‘long distance’ relationships via *adjacency*

- The guy that we know in Somerville likes ice-cream
- Who did the guy who lives in Somerville see __?

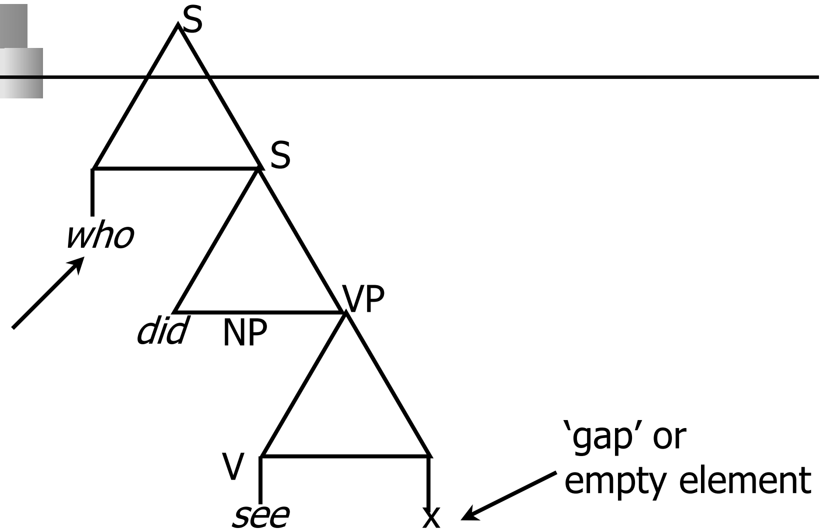


Why: recover meaning from structure

John ate ice-cream → ate(John, ice-cream)

-This must be done from *structure*
-Actually want something like $\lambda x \lambda y$ ate(x,y)
How?

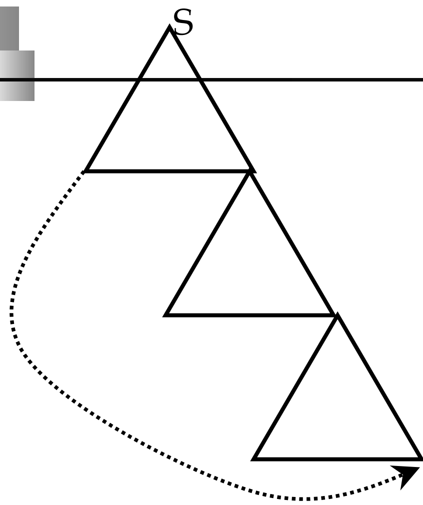
Structure *must* be recovered



But now we have a more complex Marxist analysis

- I shot an elephant in my pajamas
- This is *hierarchically* ambiguous – not just linear! (each possible hierarchical structure corresponds to a *distinct* meaning)
- A case of <u>structural ambiguity</u>

What is the structure that matters?



Turns out to be SCOPE for natural languages!

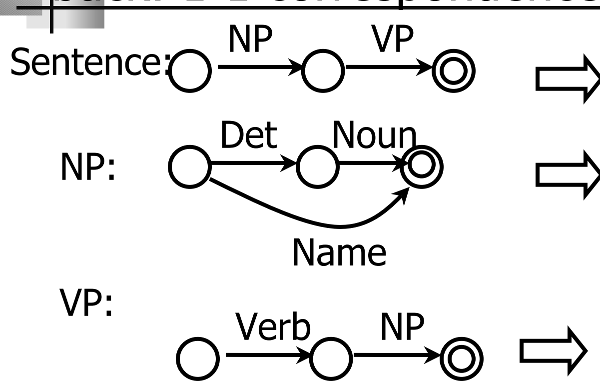
The language for hierarchical structure

- What are the basic elements
- How are they put together?

The elements

1. What: hierarchical representations (anything with recursion) using *phrases* AKA "constituents"
2. Constituents are <u>equivalence classes</u> of words
3. How: context-free parsing (plus…)
4. Why: (meaning)

Recursive Transition Networks to context-free grammars (CFGs) and back: 1-1 correspondence



$S \rightarrow NP\ VP$

$NP \rightarrow Name$
$NP \rightarrow Det\ Noun$

$VP \rightarrow Verb\ NP$

+ *terminal expansion rules*

Added information

- FSA represents pure *linear* relation: what can *precede* or (*follow*) what
- CFG/RTN adds a single new predicate: *dominate*
- Claim: The dominance and precedence relations amongst the words exhaustively describe its *syntactic* structure
- When we parse, we are recovering these predicates

Dominance & precedence define context-free grammars completely

- Definition of context-free grammar (CFG)
- Definition of derives : determines hierarchy
- We'll get to that soon…but first, from linear machines to hierarchical ones…

The deepest lesson

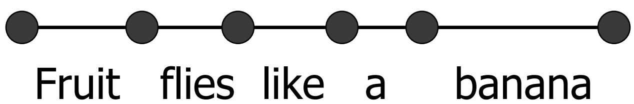
- Claim: *all* apparently nonadjacent relationships in languge can be reduced to *adjacent* ones via projection to a new level of representation
- (In one sense, vacuous; in another, deep)
- Example: Subject-Verb agreement (agreement generally)
- Example: so-called *wh*-movement

OK: start with finite-state machines

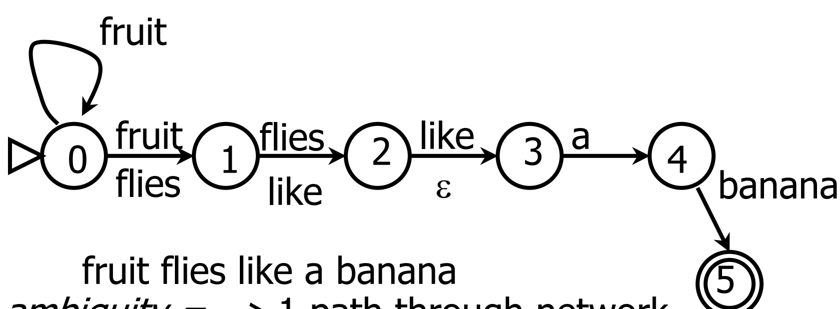
- Marxist analysis, step 1
- Then historical revisionism…

Marxist analysis: simple version

- Suppose just *linear* relations to recover
- Still can be ambiguity – multiple paths
- Consider:



Parsing for fsa's: keep track of what 'next state' we could be in at each step



fruit flies like a banana

NB: *ambiguity =* > 1 path through network
= > 1 sequence of states ('*parses*')
= > 1 'syntactic rep' = >1 'meaning'

Methods for parsing

- How do we handle ambiguity?
- Methods:
  1. Backtrack
  2. Convert to deterministic machine (ndfsa $\rightarrow$ dfsa): *offline* compilation
  3. Pursue all paths in parallel: *online* computation ("state set" method)
  4. Use lookahead
  - We will use <u>all</u> these methods for more complex machines/language representations

FSA terminology

- Input alphabet, Σ; transition mapping, δ; finite set of states, Q; start state q_0; set of final states, q_f
- $\delta(q, s) \rightarrow q'$
- Transition function: next state unique = deterministic fsa
- Transition relation: > 1 next state = nondeterministic fsa

State-set method: simulate a nondeterministic fsa

- Compute all the possible next states the machine can be in at a step = *state-set*
- Denote this by S_i = set of states machine can be in after analyzing *i* tokens
- Algorithm has 3 parts: (1) *Initialize;* (2) *Loop;* (3) *Final state?*
- *Initialize:* S_0 denotes initial set of states we're in, before we start parsing, that is, q_0
- *Loop:* We must compute S_i, given S_{i-1}
- *Final?:* S_f = set of states machine is in after reading all tokens; we want to test if there is a final state in there

State-set parsing

Initialize: Compute initial state set, S_0

1. $S_0 \leftarrow q_0$
2. $S_0 \leftarrow \varepsilon$–closure($S_0$)

Loop: Compute S_i from S_{i-1}

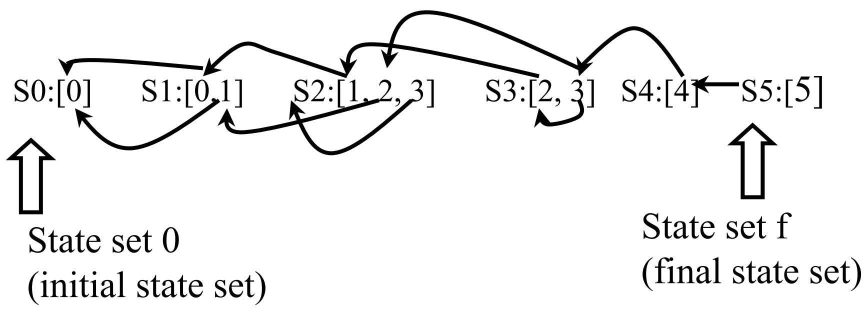
1. For each word w_i , i=1,2,…,n
2.
3. $S_i \leftarrow \varepsilon$–closure($S_i$)
4. if $S_i = \varnothing$ then halt & reject else continue

Final: Accept/reject

1. If $q_f \in S_n$ then accept else reject

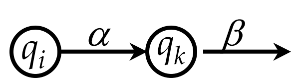
States in sequence dictate parse path:

States: {0} →{0,1} →{1,2,3} →{2,3} →{4} →{5} (final)



What's the minimal data structure we need for this?

- [*S, i*] where *S* = denotes *set of* states we could be in; *i* denotes current point we're at in sentence
- As we'll see, we can use this *same* representation for parsing w/ more complex networks (grammars) - we just need to add *one* new piece of information for *state names*
- In network form



- In rule form:

$q_i \rightarrow t \bullet q_k$ *where* τ= some token of the input,

State to state jumps…

- Progress (& ultimately parse) recorded by what state machine is in
- Consider each transition as rule:

  $q_0 \rightarrow$ fruit q_1 , also loop: $q_0 \rightarrow$ fruit q_0

  $q_1 \rightarrow$ flies q_2 ; $q_0 \rightarrow$ flies q_1 also epsilon transition: $q_1 \rightarrow q_3$

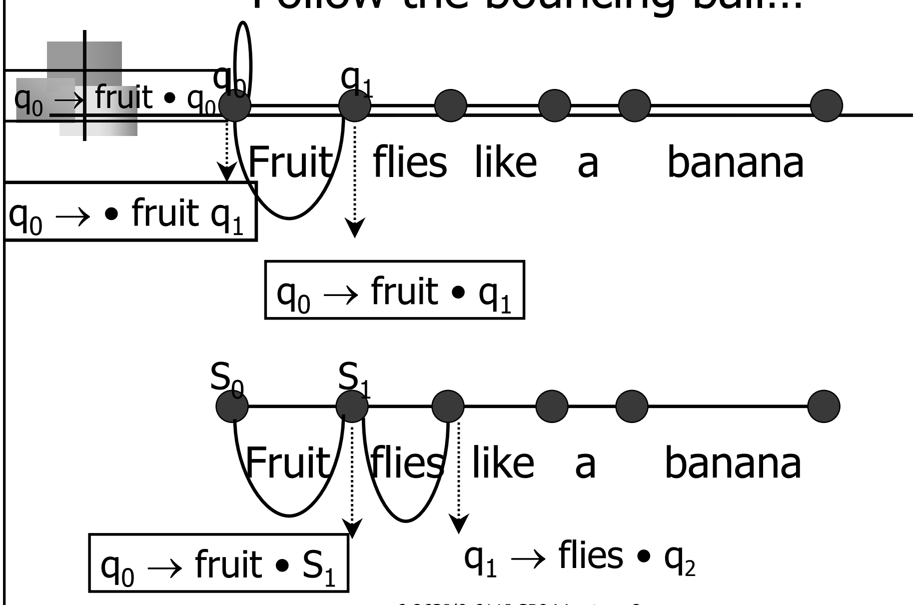
  $q_2 \rightarrow$ like q_3 also epsilon transition: $q_2 \rightarrow q_3$

  $q_3 \rightarrow$ a q_4

  $q_4 \rightarrow$ banana q_5

- We can record progress path via 'bouncing ball' <u>dot</u> telling us how to sing the song…

Follow the bouncing ball…



To be picky about the 'dotted rules'

- We can write it this way:
- [$q_i \rightarrow t \bullet q_j$, k] where *k*= index of where we are at in the parse (i=0, 1, 2, …, n for a string n words long)
- Let us also call this an item
- A collection of items in a state set is an item set

Reviewing this representation

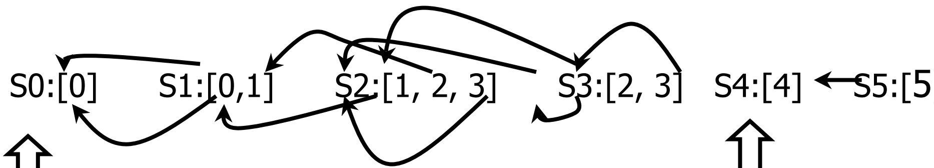
- Dotted rules indicate 'progress so far'
- They also denote traversal between categories (aka 'states')
- The collection of dotted rules at any step in a parse denotes the set of possible states the parser could be in at that step (a set union), more precisely, it is State Set i, that denotes the set of all possible states the parsing could be in after processing *i* words
- We could also add a 'return pointer' that tells us how we got to the current state
- So now an item looks like:

  [dotted rule, return ptr, current word so far] e.g

  [$q_0 \rightarrow$ fruit $\bullet$ q_1 , 0, 1]

States in sequence dictate parse path – from this:

States: {0} →{0,1} →{1,2,3} →{2,3} →{4} →{5} (final)



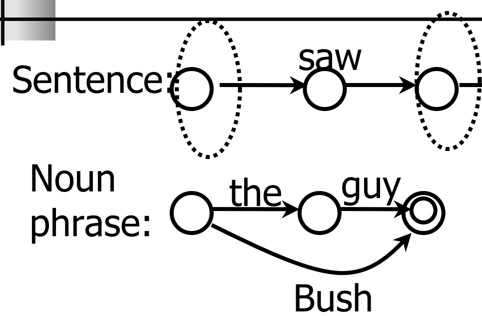
State set 0
(initial state set)

State set f
(final state set)

To this:

S0:	fruit →	S1	flies →	S2
{[q_0 → • fruit q_1, 0, 0]}		{[q_0 → fruit • q_1,0, 1], [q_0 → fruit • q_0, 0, 1]}		{[q_1 → flies • q_2, 1, 2], [q_0 → flies • q_1, 0, 2] {[q_1 → flies • q_3 , 2, 2}

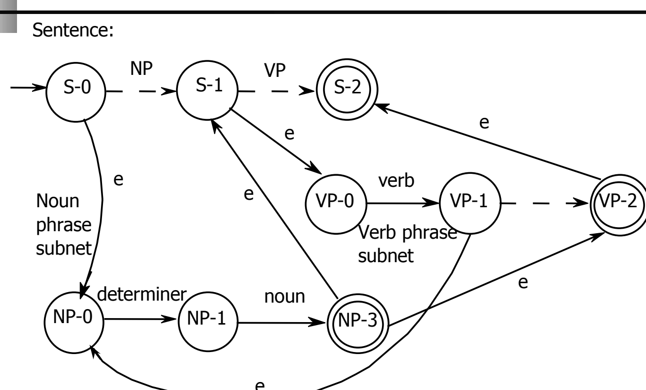
How do we move from linear to hierarchical?



"splice out" common subnets

We already have the machinery for this…

Use of epsilon transitions ('jump' arcs) – they consume no input



…note that *no* input is consumed during jump

This will work… with one catch

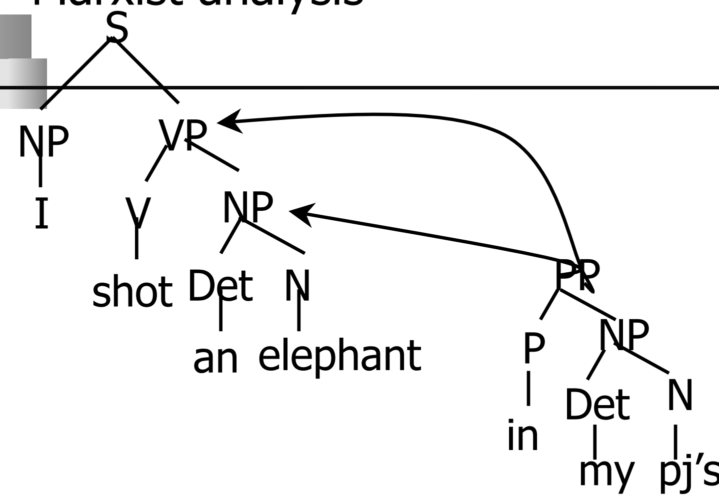
- Consider tracing through "the guy ate the ice-cream"
- What happens when we get to the second noun phrase????
- Where do we *return* to?
- Epsilon transition takes us back to <u>different</u> points

But now we have a more complex Marxist analysis

- I shot an elephant in my pajamas

- This is *hierarchically* ambiguous – not just linear! (each possible hierarchical structure corresponds to a *distinct* meaning)

Marxist analysis



What: Context-free grammars (CFG)

S(entence)→NP VP
VP→V NP
NP→Det N
N → *pizza,* N → *guy,* Det → *the* } pre-terminals, lexical entries
V → *ate*

A context-free grammar (CFG):
Sets of <u>terminals</u> (either lexical items or parts of speech)
Sets of <u>nonterminals</u> (the constituents of the language)
Sets of <u>rules</u> of the form $A \rightarrow \alpha$ where α is a string of zero or more terminals and nonterminals

More precisely

- A context-free grammar (CFG) is a 4-tuple (N, Σ, P, S) where:
  - N is a finite set of <u>nonterminal symbols</u> (phrase names, categories);
  - Σ is a finite set of <u>terminal symbols</u> (words);
  - P is a set of <u>production rules</u> $\langle A \in N, \alpha \rangle$, where α is a sequence of terminal or nonterminals; and
  - $S \in N$ is a designated <u>start symbol</u>.
- We write the productions as $A \rightarrow \alpha$ ('is-a')

Definitions for CFGs

- The derive relation $\Rightarrow$
- Define wrt grammar $G = (N, \Sigma, P, S)$ as follows

  $\alpha \Rightarrow \beta$ iff $\exists\ \alpha_1, \alpha_2$ s.t. $\alpha = \alpha_1\, A\, \alpha_2$; $\beta = \alpha_1\, \gamma\, \alpha_2$; and $A \rightarrow \gamma \in P$. (Some rule rewrites α as β)
- Reflexive, transitive closure of $\Rightarrow$ is $\Rightarrow^*$

If α, β is in $\Rightarrow^*$ then we say that α <u>derives</u> β (by 0 or more steps)

Derivation by a context-free grammar:rewrite line by line

generation

1. <u>S</u>	
2. NP <u>VP</u>	(via S→NP VP)
3. NP V <u>NP</u>	(via VP→V NP)
4. NP V Det <u>N</u>	(via NP→Det N)
5. NP V <u>Det</u> *pizza*	(via N → *pizza*)
6. NP <u>V</u> *the pizza*	(via Det → *the*)
7. <u>NP</u> *ate the pizza*	(via V → *ate*)
8. Det <u>N</u> *ate the pizza*	(via NP→Det N)
9. <u>Det</u> *guy ate the pizza*	(via N → *guy*)
10. *the guy ate the pizza*	(via Det → *the*)

Derives relation

- Relates <u>all</u> elts by either dominance or precedence
- Induces a (derivation) tree (Q: do we lose any information in this tree?)

Definition of derivation tree

- Binary Relation *D,* dominance:
  A D v iff $\exists\ \alpha_1, \alpha_2$ ($\alpha \Rightarrow \beta$ via $A \rightarrow \alpha_1 v\, \alpha_2$)

- Binary relation < precedence:
  $v <\ w$ iff $\exists\ \alpha_1, \alpha_2$ ($\alpha = \alpha_1 vw\, \alpha_2$ or $\beta = \alpha_1 vw\, \alpha_2$ & $\alpha \Rightarrow \beta$)

Confirm that our derivation steps previously induce such a tree… note that all elts are related by < or D. (Suppose not…?)

The yield of a nonterminal (category) *A* consists of all strings derivable from *A*

Context-free representation

- Is this representation adequate – Not really…why?
- We'll start here, though & illustrate parsing methods – how to make parsing efficient (in length of sentence, size of grammar)
- Obvious methods are exponential; we want polynomial time (or, even linear time, or, even, real time…)
- Challenges: recursion, ambiguity, nondeterminism

How: context-free parsing

- Parsing: assigning a correct hierarchical structure (or its derivation) to a string, given some grammar
  - The leaves of the hierarchical structure cover all and only the input;
  - The hierarchical structure ('tree') corresponds to a valid derivation wrt the grammar
- Note: 'correct' here means consistent w/ the input & grammar – NOT the "right" tree or "proper" way to represent (English) in any more global sense

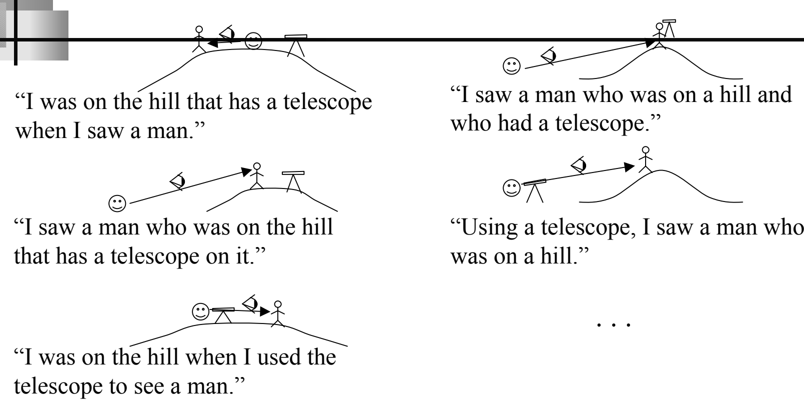
Parsing

- What kinds of constraints can be used to connect the grammar and the example sentence when searching for the parse tree?
- Top-down (goal-directed) strategy
  - Tree should have one rot (grammar constraint)
- Bottom-up (data-driven) strategy
  - Tree should have, e.g., 3 leaves (input sentence constraint)

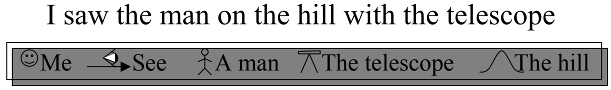
The input

- For now, assume:
  - Input is not tagged (we can do this…)
  - The input consists of unanalyzed word tokens
  - All the words are known
  - All the words in the input are available simultaneously (ie, buffered)

Ambiguity Can Yield Exponential # of Parses



I saw the man on the hill with the telescope



How do we do this?

- Searching FSAs
  - Finding the right path through the automaton
  - Search space defined by structure of FSA
- Searching CFGs
  - Finding the right parse tree among all possible parse trees
  - Search space defined by the grammar
- Constraints provided by the input sentence and the automaton or grammar