6.863J Natural Language Processing Lecture 7: parsing with hierarchical structures – context-free parsing

Robert C. Berwick berwick@ai.mit.edu

## The Menu Bar

- Administrivia:
  - Schedule alert: Lab2 due Weds; Lab 3 out Monday (chunk parsing to 'real' parsing)
  - Lab time today, tomorrow
  - Please read notes3.pdf, englishgrammar.pdf (on web)
- Agenda:
- Marxist analysis simple & post-modern
- <u>What:</u> hierarchical representations; constituents, representation
- <u>How</u>: constituent or 'context-free' parsing (next time – how to do it *fast*)
- Why: to extract 'meaning'

6.863J/9.611J Lecture 7 Sp03

## 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*

## Programming languages

```
printf ("/charset [%s",
       (re_opcode_t) *(p - 1) == charset_not ? "^" : "");
assert (p + *p < pend);
for (c = 0; c < 256; c++)
  if (c / 8 < *p && (p[1 + (c/8)] & (1 << (c % 8)))) {
     /* Are we starting a range? */
     if (last + 1 == c & \& ! inrange) {
       putchar ('-');
       inrange = 1;
     }
/* Have we broken a range? */
    else if (last + 1 != c \&\& inrange) {
       putchar (last);
       inrange = 0;
     if (! inrange)
       putchar (c);
                                  Easy to parse.
     last = c;
  }
                                  Designed that way!
```

## Natural languages

printf "/charset %s", re\_opcode\_t \*p - 1 == charset\_not ? "^"
: ""; assert p + \*p < pend; for c = 0; c < 256; c++ if c / 8 <
\*p && p1 + c/8 & 1 << c % 8 Are we starting a range? if last +
1 == c && ! inrange putchar '-'; inrange = 1; Have we broken
a range? else if last + 1 != c && inrange putchar last;
inrange = 0; if ! inrange putchar c; last = c;</pre>

- No {} () [] to indicate scope & precedence
- Lots of overloading (arity varies)
- Grammar isn't known in advance!
- Context-free grammar not best formalism

## How: The parsing problem



## Syntactic Parsing

- Declarative formalisms like CFGs define the legal strings of a language but don't specify how to recognize or assign structure to them
- Parsing algorithms specify how to recognize the strings of a language and assign each string one or more syntactic structures
- Parse trees useful for grammar checking, semantic analysis, MT, QA, information extraction, speech recognition...and almost every task in NLP

## Applications of parsing (1/2)

Machine translation (Alshawi 1996, Wu 1997, ...)

English dependence operations dependence Chinese

 Speech synthesis from parses (Prevost 1996) The government plans to raise income tax. The government plans to raise income tax the imagination.

Speech recognition using parsing (Chelba et al 1998)
 Put the file in the folder.
 Put the file and the folder.

## 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
              6.863J/9.611J Lecture 7 Sp03
```

## 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 \_\_?



6.863J/9.611J Lecture 7 Sp03

## Why: recover meaning from structure

John ate ice-cream  $\rightarrow$  ate(John, ice-cream)

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

## Why: recover meaning from structure



## Why: Parsing for the Turing Test

- Most linguistic properties are defined over hierarchical structure
- One needs to parse to see subtle distinctions

Sara likes her. $(her \ {}^{1}Sara)$ Sara thinks that someone likes her. $(her = or \ {}^{1}Sara)$ Sara dislikes anyone's criticism of her. $(her = Sara \text{ or } her \ {}^{1}Sara)$ Who did John see?  $\rightarrow$  For which x, x a person, likes(Bill, x)

Distinction here is based on *hierarchical structure* =  $\underline{\text{scope}}$  in natural language

### Structure *must* be recovered



### What is the structure that matters?



#### Turns out to be SCOPE for natural languages!

6.863J/9.611J Lecture 7 Sp03

## The elements

- What: hierarchical representations (anything with recursion) using *phrases* AKA "constituents"
- 2. How: context-free parsing (plus...)
- 3. Why: (meaning)



6.863J/9.611J Lecture 7 Sp03

## Added information

- FSA represents pure *linear* relation: what can *precede* or (*follow*) what
- CFG/RTN adds a 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

## How do we move from linear to hierarchical?



#### We already have the machinery for this...

## Use of epsilon transitions ('jump' arcs) – they consume <u>no</u> input

Sentence:



## 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

# What: Context-free grammars (CFG)

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

 $V \rightarrow ate$ 

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  $\alpha$  is a string of zero or more terminals and nonterminals

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

#### generation

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

(via  $S \rightarrow NP VP$ ) (via VP $\rightarrow$ V NP) (via NP $\rightarrow$ Det N) (via N  $\rightarrow$  *pizza*) (via Det  $\rightarrow$  *the*) (via V  $\rightarrow$  ate) (via NP $\rightarrow$ Det N) (via N  $\rightarrow$  guy) (via Det  $\rightarrow$  *the*)

### 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



- 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)

## 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

## Marxist analysis: simple version

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



## FSA, or linear Example



### State-set parsing for fsa Initialize: Compute initial state set, S<sub>0</sub>

1.  $S_0 \leftarrow q_0$ 

Loop:

- 2.  $S_0 \leftarrow \varepsilon$ -closure( $S_0$ ) Compute  $S_i$  from  $S_{i-1}$ 1. For each word  $w_i$ , i=1,2,...,n2.  $S_i \leftarrow \bigcup_{q \in S_{i-1}} \boldsymbol{d}(q,w_i)$ 3.  $S_i \leftarrow \varepsilon$ -closure( $S_i$ )
- 4. if  $S_i = \emptyset$  then halt & reject else continue

Final:

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

6.863J/9.611J Lecture 7 Sp03

# States in sequence dictate parse path:

States:  $\{0\} \rightarrow \{0,1\} \rightarrow \{1,2,3\} \rightarrow \{2,3\} \rightarrow \{4\} \rightarrow \{5\}$  (final)



## 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:  $q_0 \rightarrow fruit q_0$
  - $q_1 \rightarrow flies q_2$
  - $q_2 \rightarrow \text{like } q_3$  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' telling us how to sing the song...

6.863J/9.611J Lecture 7 Sp03



<sup>6.863</sup>J/9.611J Lecture 7 Sp03

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



# How can we extend this bouncing ball?

- Can't just be linear...
- How do we *pack* these possibilities together?
- We will augment... let's see how

#### From this...





6.863J/9.611J Lecture 7 Sp03

#### To this... what is called a Chart



#### Three senses of rules

- generation (production):
- parsing (comprehension): S ← NP VP

 $S \rightarrow NP VP$ 

- verification (checking):
  S = NP VP
- CFGs are <u>declarative</u> tell us *what* the well-formed structures & strings are
- Parsers are <u>procedural</u> tell us *how* to compute the structure(s) for a given string

# CFG minigrammar

$S \rightarrow NP VP$	$VP \rightarrow V$
$S \rightarrow Aux NP VP$	Det → that   this   a
$S \rightarrow VP$	$N \rightarrow book \mid flight \mid meal \mid$
	money
NP $\rightarrow$ Det Nom	V → book   include   prefer
NP →PropN	Aux → does
Nom $\rightarrow$ N Nom	Prep →from   to   on
Nom $\rightarrow$ N	PropN → Boston   United
Nom → Nom PP	
$VP \rightarrow V NP$ 6.863J/9.611J Lea	ture 7 Sp03

#### Parse Tree for 'Book that flight'



# Strategy 1: Top-down parsing

- Goal or expectation driven find tree rooted at S that derives input
- Trees built from root to leaves
- Assuming we build all trees in parallel:
  - Find <u>all trees with root S (or all rules w/lhs S</u>)
  - Next expand all constituents in these trees/rules
  - Continue until leaves are parts of speech (pos)
  - Candidate trees failing to match pos of input string are rejected (e.g. Book that flight can only match subtree 5)

#### Example: book the flight



# Top-down strategy

#### • Depth-first search:

- Agenda of search states: expand search space incrementally, exploring most recently generated state (tree) each time
- When you reach a state (tree) inconsistent with input, backtrack to most recent unexplored state (tree)
- Which node to expand?
  - Leftmost or rightmost
- Which grammar rule to use?
  - Order in the grammar

# Top-down, left-to-right, depth-first

- Initialize agenda with 'S' tree and ptr to first word and make this current search state (cur)
- Loop until successful parse or empty agenda
  - Apply all applicable <u>grammar rules</u> to leftmost unexpanded node of cur
    - If this node is a POS category and matches that of the current input, push this onto agenda
    - O.w. push new trees onto agenda
  - Pop new cur from agenda
- Does this flight include a meal?

# Strategy 2: Bottom-up

- Parser begins with words of input and builds up trees, applying grammar rules w/rhs that match
  - Book that flight
    - N Det N V Det N
    - Book that flight Book that flight
  - 'Book' ambiguous
  - Parse continues until an S root node reached or no further node expansion possible

#### Bottom-up search space



### Comparing t-d vs. b-u

- <u>Top-Down parsers</u> never explore illegal parses (e.g. can't form an S) -- but waste time on trees that can never match the input
- <u>Bottom-Up parsers</u> never explore trees inconsistent with input -- but waste time exploring illegal parses (no S root)
- For both: how to explore the search space?
  - Pursuing all parses in parallel or ...?
  - Which rule to apply next?
  - Which node to expand next?

#### Problems...

- Left-recursion
- Ambiguity: multiple parses
- Principle AWP

#### Left-recursion

- Rules of form:  $X \rightarrow X \alpha$
- Example: NP  $\rightarrow$  NP 's NP | Name



John's brother's book 6.863J/9.611J Lecture 7 Sp03

# Structural ambiguity

- Multiple legal structures
  - Attachment (e.g. I saw a man on a hill with a telescope)
  - Coordination (e.g. younger cats and dogs)
  - NP bracketing (e.g. Spanish language teachers)

# How to fix?

- Principle AWP! Dynamic programming...
- Create table of solutions to sub-problems (e.g. subtrees) as parse proceeds
- Look up subtrees for each constituent rather than re-parsing
- Since all parses implicitly stored, all available for later disambiguation
- Examples: Cocke-Younger-Kasami (CYK) (1960), Graham-Harrison-Ruzzo (GHR) (1980) and Earley (1970) algorithms

## General method: Chart Parsing

- Note: parses share common constituents
- Build <u>chart</u> = graph data structure for storing partial & complete parses (AKA <u>well-formed</u> <u>substring table</u>)
- Graph:
  - Vertices: used to delimit subsequences of the input
  - Edges (active, inactive)
    - Active = denote incompletely parsed (or found) phrase
    - Inactive = completely found phrase
    - Labels = name of phrase
- Note: chart *sufficient* to attain polynomial time parsability =  $O(n^3 |G|)$ , |G| = 'size' of grammar, *no matter what strategy we use*

### How do we build the chart?

- Idea: as parts of the input are successfully parsed, they are entered into chart
- Like memoization
- Can use any combo strategy of t-d, b-u, or in between to build the edges
- Annotate edges as they are built w/ the corresponding dotted rule
- Parser is a combination of chart + strategy

# Chart parsing



## Chart parsing

- Think of chart entries as sitting between words in the input string keeping track of states of the parse at these positions
- For each word position, chart contains the set of states representing all partial parse trees generated to date

# Chart parsing

- Chart entries represent three type of constituents (phrases):
  - predicted constituents
  - in-progress constituents
  - completed constituents

# Representing complete (inactive) vs. incomplete (active) edges

- Complete: full phrase found, e.g., NP, VP
- So: corresponding rule something like
  - NP $\rightarrow$ NP PP ("an elephant in my pajamas")
  - $S \rightarrow NP VP$  ("I saw an elephant")
  - NP  $\rightarrow$  Det N ("an elephant")
- Representation: use "dot" in rule to denote progress in discovering LHS of the rule:
  NP→• Det NP = I've just started to find an NP ("predict")
  NP → Det NP = Found a Det in input, now find NP
  NP → Det NP = Completed phrase (dot at end)

# Chart we displayed has only *inactive* (completed) edges



# Complete (Inactive) vs. Inprogress (active) edges

- Completed edges correspond to "having found a phrase" so really should be labeled with info like NP  $\rightarrow$  Det NP •
- We should go back & annotate our chart like this
- These edges are "inactive" because there is no more processing to be done to them
- Incomplete or "active" edges: work in progress,
  i.e., NP→• Det NP or NP → Det NP
- We build up the chart by extending active edges, gluing them together – let's see how

# Note correspondence between "dotted rules" & states in corresponding fsa - isomorphic

6.863J/9.611J Lecture 7 Sp03

#### Dotted rule – fsa correspondence



 $NP \rightarrow \bullet Det N = being in State 1$ 

 $NP \rightarrow Det \cdot N = being in State 2$ 

 $NP \rightarrow Det N \bullet = being in State 3$ 

6.863J/9.611J Lecture 7 Sp03

#### Correspondence



#### Correspondence



#### Correspondence



6.863J/9.611J Lecture 7 Sp03

### Representing the edges

- 0 Book 1 that 2 flight 3
  - $S \rightarrow \bullet VP$ , [0,0] (predicting VP)
  - $NP \rightarrow Det \cdot Nom$ , [1,2] (finding NP)

 $VP \rightarrow V NP \bullet$ , [0,3] (found VP)

- [x,y] tells us where a phrase begins (x) and where the dot lies (y) wrt the input – how much of the phrase is built *so far*
- So, a FULL description of a chart edge is:
  Edge Label, [start node, current progress dot pos] .e.g.,

 $NP \rightarrow Det \cdot Nom, [1,2]$ 

# Set of dotted rules encodes state of parse

- all states parser could be in after processing *i* tokens
- We now have almost all the ingredients...

### FSA, or linear Example



#### State-set parsing for fsa Initialize: Compute initial state set, S<sub>0</sub>

Loop:

1.  $S_0 \leftarrow q_0$ 2.  $S_0 \leftarrow \varepsilon$ -closure( $S_0$ ) Compute  $S_i$  from  $S_{i-1}$ 1. For each word  $w_i$ , i=1,2,...,n2.  $S_i \leftarrow \bigcup_{q \in S_{i-1}} d(q,w_i)$ 3.  $S_i \leftarrow \varepsilon$ -closure( $S_i$ ) 4. if  $S_i = 0$  then halt & reject of

4. if  $S_i = \emptyset$  then halt & reject else continue

Final:

Accept/reject

1. If  $q_f \in S_n$  then accept else reject 6.863J/9.611J Lecture 7 Sp03

# Use backpointers to keep track of the different paths (parses):


# Chart parsing is the same, except...

 Notion of 'state set' is just more complicated – not just the state #, but also the # of the state we started building the phrase at = the return ptr

Note this is what the chart graph structure encodes

### State set = chart after i words

<u>Given</u> grammar G, <u>input</u> string w=w<sub>1</sub> w<sub>2</sub>
 ...W<sub>n</sub>

Note: we mark interword positions  $_0W_1 W_2 \dots W_n$ 

- Initialize: write down what can be in "start state set"  $S_0$
- Loop: for each word  $w_i$ , compute  $S_i$  from  $S_{i-1}$
- Final: see if final state is in last state set  $S_n$



## Parsing procedure w/ chart

- Move through each <u>set of states</u> in order, applying one of three operators to each state:
  - predictor: add new active edges, predictions, to the chart
  - scanner: read input and advance dot, add corresponding active edge to chart
  - completer: if dot at the right end of a rule, then see if we can glue two edges together to form a larger one

- Results (new edges) added to current or next set of states in chart
- No backtracking and no edges removed: keep complete history of parse
- When we get to the end, there ought to be an edge labeled S, extending from 0 to n (n= length of sentence)

As in



## Predictor ('wishor')

- Intuition: new states represent top-down expectations
- Applied when non part-of-speech non-terminals are to the right of a dot – until closure
   S → • VP [i,i]
- Adds new states to *current* chart
  - One new state for each expansion of the nonterminal in the grammar

 $VP \rightarrow \bullet V [i,i]$  $VP \rightarrow \bullet V NP [i,i]$ 

## Scanner (as in fsa)

- New states for predicted part of speech
- Applicable when part of speech is to the right of a dot
  VP → V NP [0,0] 'Book...'
- Looks at current word in input
- If match, adds dotted rule edge starting at next point over, e.g.,
   VP → V • NP [0,1]

Just as with fsa's – jump to next point



- Intuition: parser has discovered a complete constituent, so must see if this completed edge can be pasted together with any preceding active edge to make a bigger one...
- E.g., NP[0, 2] & VP[2, 7] yields S[0,7]
- "Glue together" two edges
- Must do this until closure...

### Examples – will use v, v simple G

- $S \rightarrow NP VP$
- $VP \rightarrow V NP$
- VP  $\rightarrow$  V NP PP
- NP  $\rightarrow$  D N
- NP  $\rightarrow$  N
- NP  $\rightarrow$  NP PP
- $PP \rightarrow PNP$

## Strategies w/ Chart

- Top-down
- Bottom-up
- Left-corner (what's that??)

### Example: Top-down w/ chart



State set S<sub>0</sub> - nothing more can be added, so scan next word

Note how top-down strategy can introduce rules unconnected to the input.. 6.863J/9.611J Lecture 7 Sp03

## Scan to next word...follow the bouncing dot...



### Dot at end...so we 'complete' NP



## And now predict...expand VP (t-d)



#### Scan Verb



## NP Predictions added



### Process NP object



### Enough...no more! Demo easier!

