6.863J Natural Language Processing
Lecture 9: Going nonlinear - Marxist analysis

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

The Menu Bar

- Administrivia:
  - Lab 2a/2b due <u>MONDAY</u>; 3a out Monday

Agenda:

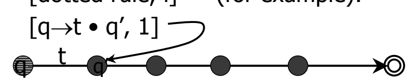
- Going nonlinear: beyond finite-state machines
- Parsing strategies: parsing as search; top-down; bottom-up methods
- Parsing strategies: chart parsing as all-purpose search data structure – algorithm & time complexity; CKY and Earley algorithm
- Preview of Lab 3

Three senses of rules

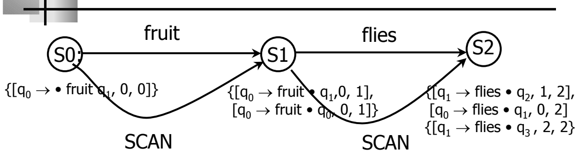
- generation (production): S → NP VP
- parsing (comprehension): S ← NP VP
- verification (checking): S = NP VP
- CFGs are <u>declarative</u> – tell us *what* the well-formed structures & strings are
- Parsers are <u>procedural</u> – tell us *how* to compute the structure(s) for a given string

Where are we at? Marxist analysis: simple version for linear parsing

- Suppose just *linear* relations to recover
- Need complete description of parse state = [state name, 'you are here' index]= [dotted rule, i] = (for example): [q→t • q', 1]



Parsing = building State sets via operation of 'scanning' from word to word



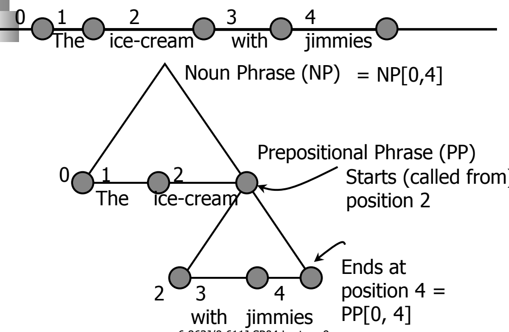
State set parsing = compute machine state after i words

- <u>Given</u> grammar G, <u>input</u> string $w=w_1\ w_2 \ldots w_n$
  Note: we mark interword positions $_0w_1\ w_2 \ldots w_n$
- <u>Initialize:</u> write down what can be in "start state set" S_0
- <u>Loop:</u> for each word w_i , compute S_i from S_{i-1}
- <u>Final:</u> see if final state is in last state set S_n

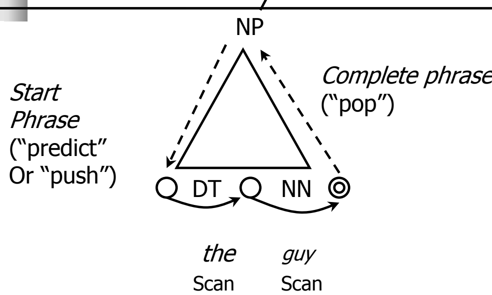
What information do we need for nonlinear (hierarchical) parsing?

- All that we need in linear case (state name AKA dotted rule, where we are in input) plus
- One more additional piece of information
- Who dominates me? (Who called me?)
- This is necessary for hierarchical description
- This plus precedence (as we saw) is also sufficient

Picture: what we need



Another way to view it



Three operations suffice

- One to scan (needed for linear relation) plus:
- One to push for new phrases
- One to complete or pop new phrases
- These 3 show up under different stage names, but always there as ‘abstract’ ops for context-free parsing

So the extra info we need

- In addition to the phrase type
- The start and stop position (spanning) of a phrase
- The start position tells us ‘who called’ us (= caller address, also return address)
- This is all we need to describe hierarchical info
- If we update the State set algorithm with this, we can extend it to parse context-free grammars (we will simulate a nondeterministic machine on-the-fly, as before. Q: why can’t we convert it offline?)

Example

0 I 1 shot 2 an 3 elephant 4 in 5 my 6 pajamas 7 #

S(entence)[0, 7];	NP[0, 1];
Verb Phrase VP[1, 7];	NP[2, 3];
PP[4,7];	NP[5, 6]

What else?

Parsing as search

- "All" we need to do is find the right elements S[0,7], etc. – these are 'points' in a search space of possibilities
- How?
- Q: What is the size of the search space?
- Q: well, consider the # of possible elts of the form X[i,j]
- Question 2: How do we search from point to point?

What does the search look like?

Parsing as a Search Problem (II)

- Search space: The set of phrasal extents
  - PhraseType[start:end]
  - E.g.: NP[0:2]
- Goal:
  - Find a set of paths through the search space…
    - That don't overlap…
      - And that connect S[0:n] to each word.
- Size of search space: $|G|n^2$ (G=grammar; n=words)
- Time to search the space: ?
  - If we look at each phrasal extent once, Gn^2
  - otherwise, it might be more (exponential)

How should we explore the 'phrase space' most efficiently?

- Depth first search = top-down parsing
- Parallel - Breadth-first search
- Bottom-up parsing
- ?Best first (we'll get to it later)
- Let's take a quick look

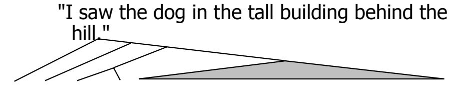
Top-Down Parsing

- Two basic operations:
  1. Expand LHS of rule into RHS elements
  2. Match against against input
- When good?
- When bad?
- When does it do useless work?
- What is its complexity?

Top Down Parsing Issues

- "Left-recursive" rules can cause infinite loops
  - NP → NP and NP
- Explores trees that are inconsistent with the input
- Redundant parsing of phrases.

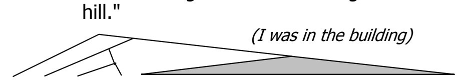
"I saw the dog in the tall building behind the hill."



(the dog was in the building)

"I saw the dog in the tall building behind the hill."

(I was in the building)



Bottom-up parsing

- Two basic operations
  1. Shift words onto stack
  2. Reduce stack elts and replace with LHS of rule

Parsing Issues: Solutions

- Re-use the sub-parses we've already computed
- Combine top-down and bottom-up approaches
  - Get the "best of both worlds"
  - We need some *common representation* for the information from top-down and bottom-up approaches.
  - Use heuristics to decide when to use bottom-up or top-down approaches.

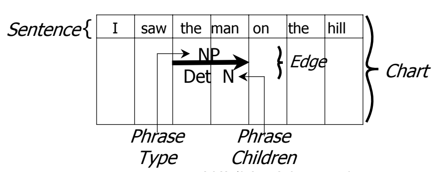
Chart Parsing

Use can use a *chart* to record hypotheses about possible syntactic constituents.

A chart contains a set of edges

Each edge represents a possible phrase.

Edges provide a common representation for parse information.



General method: Chart Parsing

- Note: parses *share* common constituents
- Build chart = graph data structure for storing partial & complete parses (AKA well-formed substring table)
- 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*

Chart parsing

- Example of chart

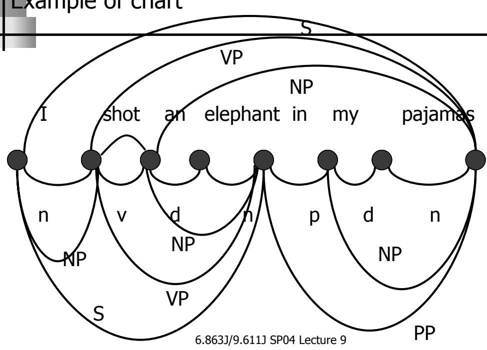


Chart parsing

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

Chart as a Matrix

- We can represent a chart as an upper triangular matrix.
- chart[i,j] is the set of dotted rules that span [i:j]

i \ j	0	1	2	3
0	S → • NP VP NP → • John	John → • NP → John • S → NP • VP		S -> NP VP •
1		VP → • V NP V → • saw	saw → • V → • saw VP → V • NP	VP → V NP •
2			NP → • Mary	Mary → • NP → Mary•
3				

Chart parsing

- Think of chart entries - edges 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

The chart

- A cell in the chart can contain more than one phrase (e.g., n & np)
- With each constituent is frequently stored information about which parsing rule was used to generate it and what smaller constituents make it up (to recover the parse)
- Used to prevent redundant work if 2 or more possible internal structures for a single phrase ("blue socks and shoes")

Chart parsing teminology

- A dotted rule is a CFG rule with a dot on the right hand side. This denotes a state in the nondet machine simulation
  - A dotted rule is complete if its dot is at the end (= phrase it is building is finished)
  - Otherwise, a dotted rule is incomplete
- An edge is a dotted rule at a location (start+end)
  - An edge is complete if its dotted rule is complete
- A chart is a set of edges

Chart parsing

- A chart parser has three data structures:
  - an input stack, which holds the words of the input sentence (in order)
  - a chart, which holds completed constituents organized by starting position and length (the edges)
  - a set of edges, organized by ending position
  - As we parse, edges are always added to the chart; never deleted from the chart

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 Strategies

- Chart parser rules define the basic operations.
- A strategy defines what rules are applied when.
- The chart parser keeps applying every rule until no more edges are added.
- But we can avoid redundant work with better strategies. E.g.:
  - Process edges in a fixed order
  - Use a queue, and examine each edge once
  - Use a more general data structure (aka "Agenda") for this

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

- Complete: full phrase found, e.g., NP, VP
- So: corresponding rule something like
  - NP→NP PP ("an elephant in my pajamas")
  - S → NP VP ("I saw an elephant")
  - NP → 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)

Complete (Inactive) vs. In-progress (active) edges

- Completed edges correspond to "having found a phrase" so really should be labeled with info like

  NP → 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

The input

- Positions in the input sentence will be numbered starting with zero and will be the positions between successive words. For example:

  The vine climbed the trellis

  0 1 2 3 4 5

  I saw an elephant in my pajamas

  0 1 2 3 4 5 6 7

Words annotated w/ pos – eg

The vine climbed the trellis

DT NNs Vbed DT NNs

Input sentence stack

- The input
- Positions in the input sentence will be numbered starting with zero and will be the positions between successive words. For example:

  $_0$ I $_1$ shot $_2$ an $_3$ elephant $_4$ in $_5$ my $_6$ pajamas $_7$

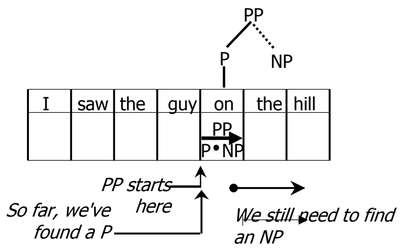
For now, assume POS already assigned, words consumed l-to-r

The Edges

- Each edge consists of a (dotted) grammatical rule, plus information about how it matches up against the input
- The edge contains:
  - A grammar rule, e.g, Verb Phrase (VP) → Verb NP
  - The position up to which we have matched the rule to the input, usually indicated by a dot in the middle of the rule (e.g. VP → Verb • NP)
  - Its <u>starting</u> position, i.e. first input word matched
  - The <u>number</u> of input words matched (so far)

Edges

Edges can represent *partial* phrases



Edges (continued)

- An edge consists of:
  - S: A start index (1...n)
  - E: An end index (1...n)
  - Type: A phrase type (NP, PP, etc.)
  - Found: What we've found so far (list of phrase types)
  - Need: What we still need (list of phrase types)

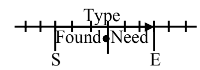


Chart Parser Rules: only 3!

- A chart parser *rule* adds new edges to the chart.
- Each chart parsing *strategy* defines a set of rules and <u>how</u> they are applied
  - Top down:
    - top-down initialization rule
    - top-down rule
    - fundamental rule
  - Bottom-up:
    - bottom-up rule
    - fundamental rule

Other strategies possible -

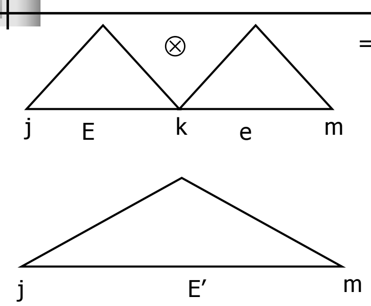
Generality of the Chart

- Chart lets us use either Top-down or BU strategy
- In fact – lets us mix strategies – depending on their value
- Extensible to features, probabilities

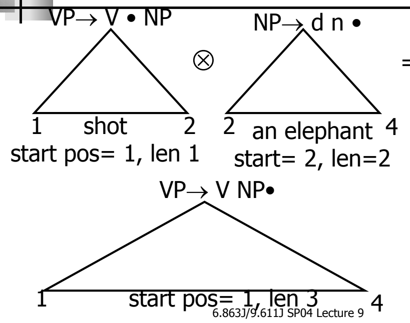
The Fundamental Rule

- Glues two subpieces into a larger one
- One rule to ring them all, one rule to bind them…

Picture of this – 'pasting' X+Y together (denoted ⊗)

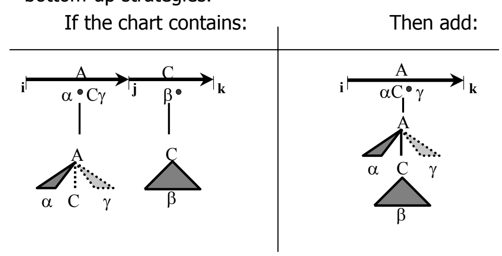


“The fundamental rule”: glues smaller trees into larger ones



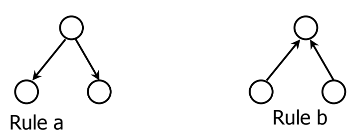
The Fundamental Rule (AKA “paste”)

- The fundamental rule is used by both top-down and bottom-up strategies.



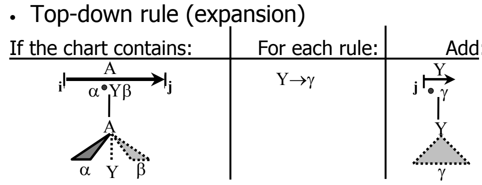
How can we make trees in the first place? (= make active edges)

- Only two ways – these exhaust the possibilities – we don’t need anything else to search the phrase space
- There is a rule for each one of these possibilities



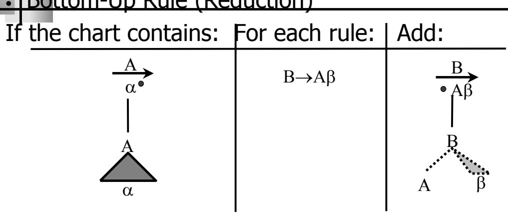
Rule a: Top-Down Rule (“Blow up”)

- Top-down initialization:
  For any rule $S \rightarrow \alpha$:
  - Add $S \rightarrow \bullet\ \alpha$ to the left side of the chart (start = end = 0).
- Top-down rule (expansion)



Rule b: Bottom-Up Rule (“Boil Down”)

- Bottom-Up Rule (Reduction)



Summary

- Chart: Set of edges (arcs), each characterizing a completed or partial constituent spanning a group of words
- Active edge: edge which still has words to be found
- Inactive edge: completed