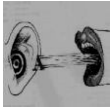


Lecture 8: Parsing tricks; beyond context-free grammars



Professor Robert C. Berwick
berwick@csail.mit.edu

The Menu Bar

- Administrivia: Lab 3...
- The Earley algorithm
- Some speedup tricks
- How do the actual ‘large scale systems’ do?
- Pluses and minuses of context-free grammars

6.863J/9.611J SP11

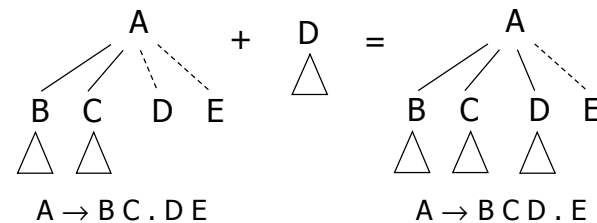
Earley parsing

- Top down instead of bottom-up
- Uses left-context and optionally right-context to constrain search so we waste less time building impossible things
- No restrictions of the form of the grammar
e.g., $NP \rightarrow \text{Det } N \text{ with } John \text{ PP}$ is an OK rule, thanks to a clever ‘on the fly’ transformation to binary branching rules (“dotted rules”)
- Again $O(n^3)$ time (but watch out for grammar size!)

6.863J/9.611J SP11

Overview of Earley’s Algorithm

- Finds constituents and partial constituents in input
 - $A \rightarrow B C . D E$ is partial: only the first half of the A



6.863J/9.611J SP11

Overview of Earley's Algorithm

- Proceeds incrementally, left-to-right
 - Before it reads word 5, it has already built all hypotheses that are consistent with first 4 words
 - Reads word 5 & attaches it to immediately preceding hypotheses. Might yield new constituents that are then attached to hypotheses immediately preceding *them* ...
- E.g., attaching D to $A \rightarrow BC$. DE gives $A \rightarrow BCD$.
- Attaching E to that gives $A \rightarrow BCDE$.
- Now we have a complete A that we can attach to hypotheses immediately preceding the A, etc.

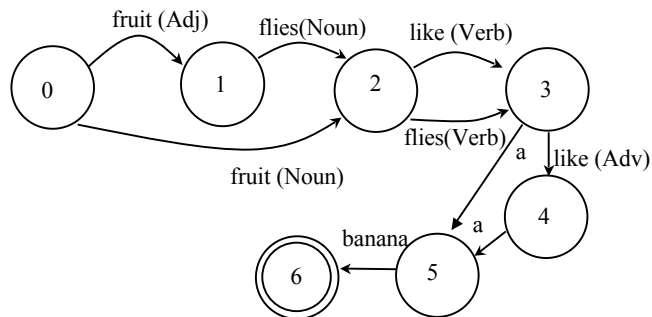
6.863J/9.611J SP11

Motivation

- We will carry out a *deterministic* simulation of a *nondeterministic* machine that can keep track of 'all the possible next states' S_i it can be in after reading each word
- So we imagine a sequence of state sets, S_0, S_1, \dots
- Representation as a 'chart' of 'columns'; Column i or $chart(i)$ will contain *all of the possible* machine states after reading the i^{th} word (NB. Indexing: we start at 0, *before* the first word of the sentence)
- Compare this to how we can solve 'fruit flies like a banana' in a nondeterministic finite-state automaton...

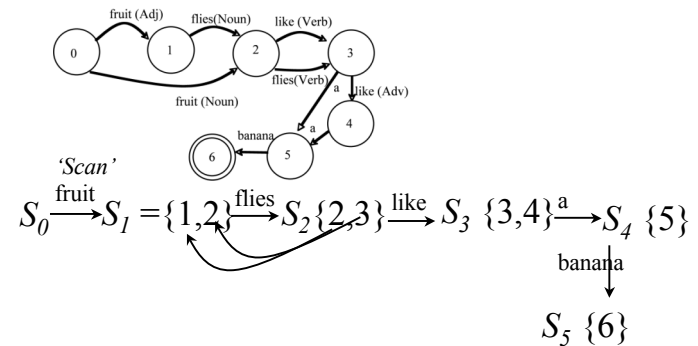
6.863J/9.611J SP11

FSA analog 'fruit flies like a banana'



6.863J/9.611J SP11

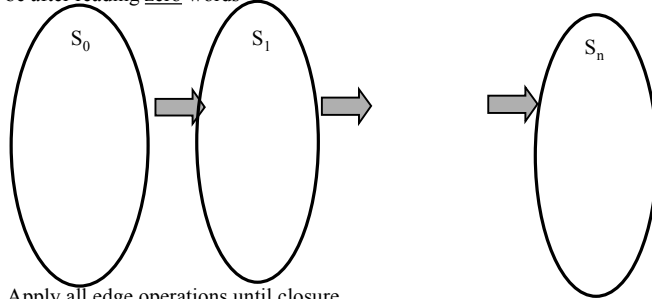
Parsing with State Sets 'fruit flies like a banana'



6.863J/9.611J SP11

State set construction as sequence of machine states, extending paths...

Initial: state set S_0 = where machine could be after reading zero words



6.863J/9.611J SP11

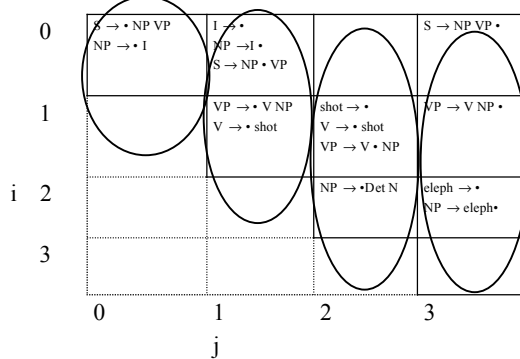
State Set machine

- Need to specify initial state
- Need to specify how to construct State Set S_i , given State Set S_{i-1}
- Need to specify final state (accept or reject)
- For getting from one state to next, need only calculate how to scan under closure, word by word
- For the linear, or finite-state case, this is all easy
- For the hierarchical, or context-free case, we need a more complex representation of 'state', and how to get from one state set to the next

6.863J/9.611J SP11

Table, or 'chart' as a sequence of State Sets
Column j = holds a *State Set*

What is a "state"? Now we need a bit more info...



6.863J/9.611J SP11

But not *much* more info...Earley's algorithm

- Use table as before; but 1 column *per* word as 'bag' (a 'chart')
- The column entries are called *states* or *items* and are represented with *dotted-rules*, plus information about *how much* of the rule has been found. A dotted rule is of 3 types:

- $S \rightarrow \bullet VP$ A VP is predicted
- $NP \rightarrow Det \bullet Noun$ An NP is in progress
- $VP \rightarrow V NP \bullet$ A VP has been found

To the dotted rule we add 2 indices that denote where the phrase on the left of the arrow *starts* (its left edge), and *how far to the right* the phrase has been constructed *so far*

These two elements constitute a 'state'

So, e.g., $[NP \rightarrow Det \bullet Noun, (1, 2)]$ means that we started building an NP at word 1 of the input, and we've found the first word, a Det.

A collection of states in a given column (e.g., col. 2), is called a *State Set*

6.863J/9.611J SP11

Dotted rules: run-time conversion to binary branching

- $S \rightarrow \bullet VP$ [0,0]
 - A VP is predicted at the start of the sentence
- $NP \rightarrow Det \bullet Noun$ [1,2]
 - An NP is in progress; the Det goes from 1 to 2
- $VP \rightarrow V NP \bullet$ [0,3]
 - A VP has been found starting at 0 and ending at 3

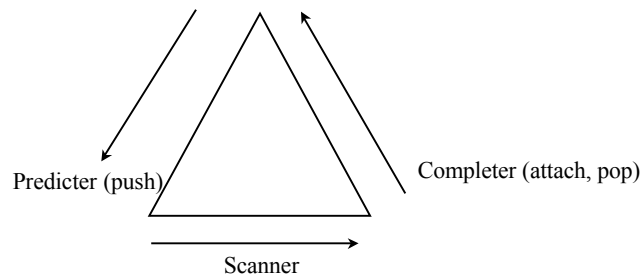
6.863J/9.611J SP11

What are...

- The initial state? $S_0 = \{Start \rightarrow \bullet S, [0,0]\}$
- The final state? $S_n = \{Start \rightarrow S \bullet, [0,n]\}$
- How do we get from one State Set to the next?
 - We apply three operations on the items in a State Set until closure
 - Not just 'scan', but 2 more to *start* and *stop* the building of trees

6.863J/9.611J SP11

Earley's algorithm uses 3 basic operations, corresponding to how we construct 'trees'



6.863J/9.611J SP11

Earley's algorithm (from textbook)

```

function EARLEY-PARSE(words, grammar) returns chart
  ENQUEUE( $\gamma \rightarrow \bullet S, [0,0]$ , chart[0])
  for  $i$  ← from 0 to LENGTH(words) do
    for each state in chart[i] do
      if INCOMPLETE?(state) and
         NEXT-CAT(state) is not a part of speech then
        PREDICTOR(state)
      elseif INCOMPLETE?(state) and
         NEXT-CAT(state) is a part of speech then
        SCANNER(state)
      else
        COMPLETER(state)
    end
  end
  return(chart)
    
```

6.863J/9.611J SP11

The 3 operations: Predictor ('wishor'); Scanner, Completer

```

procedure PREDICTOR( $A \rightarrow \alpha \bullet B \beta$ ,  $[i, j]$ )
  for each ( $B \rightarrow \gamma$ ) in GRAMMAR-RULES-FOR( $B$ , grammar) do
    ENQUEUE( $B \rightarrow \bullet \gamma$ ,  $[j, j]$ , chart[ $j$ ])
  end
procedure SCANNER( $A \rightarrow \alpha \bullet B \beta$ ,  $[i, j]$ )
  if  $B \in$  PARTS-OF-SPEECH(word[ $j$ ]) then
    ENQUEUE( $B \rightarrow \text{word}[j]$ ,  $[j, j+1]$ , chart[ $j+1$ ])
  end
procedure COMPLETER( $B \rightarrow \gamma \bullet$ ,  $[j, k]$ )
  for each ( $A \rightarrow \alpha \bullet B \beta$ ,  $[i, j]$ ) in chart[ $j$ ] do
    ENQUEUE( $A \rightarrow \alpha B \bullet \beta$ ,  $[i, k]$ , chart[ $k$ ])
  end
procedure ENQUEUE(state, chart-entry)
  if state is not already in chart-entry then
    PUSH(state, chart-entry)
  end

```

6.863J/9.611J SP11

Algorithm is just a loop around this:

1. Add $\text{Start} \rightarrow \bullet S$ to column 0
2. For each j from 0 to n :
 - i. For each dotted rule in column j (including those we add as we go), look at what's after the dot:
 - a. If word, *Scan*
 - b. If nonterminal, *Predict* (until closure)
 - c. If nothing after dot, *Complete* (until closure)
 - ii. Output 'yes' if $\text{Start} \rightarrow S \bullet$ is in column n

Note: Don't add duplicates! (AWP; but this is *tricky* to do in practice!)

Question: where is the *stack* in all of this? (Since we need a *stack*...)

6.863J/9.611J SP11

An Example...

```

ROOT  $\rightarrow$  S
S  $\rightarrow$  NP VP   NP  $\rightarrow$  Papa
NP  $\rightarrow$  Det N   N  $\rightarrow$  caviar
NP  $\rightarrow$  NP PP   N  $\rightarrow$  spoon
VP  $\rightarrow$  VP PP   V  $\rightarrow$  ate
VP  $\rightarrow$  V NP    P  $\rightarrow$  with
PP  $\rightarrow$  P NP    Det  $\rightarrow$  the
                   Det  $\rightarrow$  a
Papa ate the caviar with a spoon

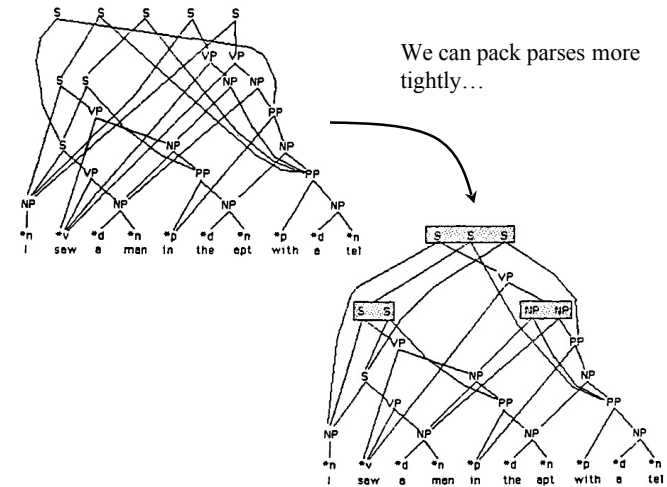
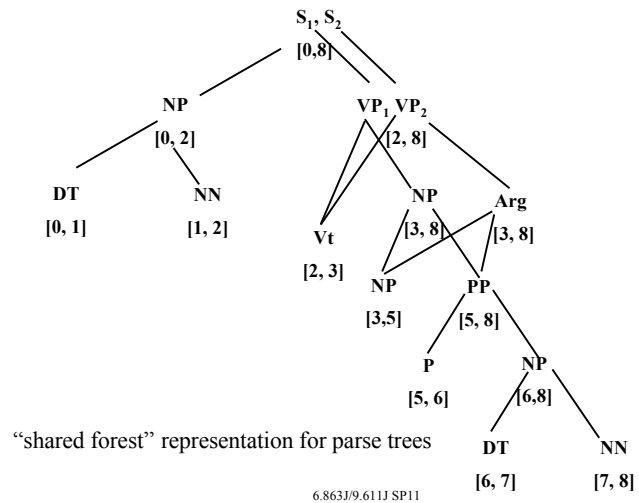
```

6.863J/9.611J SP11

Time complexity

- Algorithm iterates for every word of the input (n iterations)
- How many items can be created and processed in S_i ?
 - Each item in S_i has the form $[A \rightarrow X_1 \dots \bullet C \dots X_m \mathcal{I}]$, $0 \leq j \leq i$
 - Thus $O(n)$ items; better:
- The *scanner* and *predictor* operations on an item take at worst constant time (Why?)
- The *completer* operations on an item adds items of the form $[B \rightarrow X_1 \dots \bullet A \dots X_m \mathcal{I}]$ to S_i with $0 \leq l \leq i$, so it may require up to $O(n)$ time for each processed item
- Time required for each iteration S_i is thus $O(n^2)$
- Time bound on entire algorithm is therefore $O(n^3)$

6.863J/9.611J SP11



Implementation Details: we could have a whole algorithms course on this...

- Adding entry to parse take must check in $O(1)$ time whether another copy is there
- You have only $O(1)$ time to add entry if is new, so must be able to find bottom of the right column quickly
- (For probabilistic version): must keep track of that entry's current best parse, and total weight
- Rule $A \rightarrow WXYZ$ represented as list (W, X, Z, A)
- The dotted rule $A \rightarrow WX \cdot YZ$ represented as pair $(2, R)$ where R is a pointer to the rule (alternatively: list only elts that have not yet been matched, throw away W and X after matching; this keeps parse table smaller)
- Each column in parse table as extensible list or linked list with a tail pointer
- Duplicate check: use hash table

6.863J/9.611J SP11

Speed techniques

- Keep track of which categories have already been PREDICTed for the current column
- Build a trie that allows you to represent everything of the form $(3, A \rightarrow B \cdot C \dots)$ as a single entry in the parse table (there's a better idea, in a bit...)

6.863J/9.611J SP11

Left-Corner Parsing (“Left ancestor parsing”)

- Technique for 1 word of lookahead in algorithms like Earley’s
- (can also do multi-word lookahead but it’s harder)

6.863J/9.611J SP11

Basic Earley’s Algorithm

0	Papa	1
0 ROOT . S	0 NP Papa .	
0 S . NP VP	0 S NP . VP	
0 NP . Det N	0 NP NP . PP	
0 NP . NP PP		
0 NP . Papa		
0 Det . the		
0 Det . a		

attach

0	Papa	1
0 ROOT . S	0 NP Papa .	
0 S . NP VP	0 S NP . VP	
0 NP . Det N	0 NP NP . PP	
0 NP . NP PP	1 VP . V NP	
0 NP . Papa	1 VP . VP PP	
0 Det . the		
0 Det . a		

predict

0	Papa	1
0 ROOT . S	0 NP Papa .	
0 S . NP VP	0 S NP . VP	
0 NP . Det N	0 NP NP . PP	
0 NP . NP PP	1 VP . V NP	
0 NP . Papa	1 VP . VP PP	
0 Det . the	1 PP . P NP	
0 Det . a		

predict

0	Papa	1
0 ROOT . S	0 NP Papa .	
0 S . NP VP	0 S NP . VP	
0 NP . Det N	0 NP NP . PP	
0 NP . NP PP	1 VP . V NP	
0 NP . Papa	1 VP . VP PP	
0 Det . the	1 PP . P NP	
0 Det . a	1 V . ate	
	1 V . drank	
	1 V . snorted	

predict

- **V** makes us add all the verbs in the vocabulary!
- **Slow** – we'd like a shortcut.

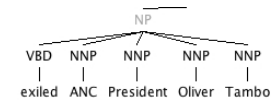
0	Papa	1
0 ROOT . S	0 NP Papa .	
0 S . NP VP	0 S NP . VP	
0 NP . Det N	0 NP NP . PP	
0 NP . NP PP	1 VP . V NP	
0 NP . Papa	1 VP . VP PP	
0 Det . the	1 PP . P NP	
0 Det . a	1 V . ate	
	1 V . drank	
	1 V . snorted	

predict

- **Every** •VP adds all VP → ... rules again.
- Before adding a rule, check it's not a duplicate.
- Slow if there are > 700 VP → ... rules, so what will you do? (A BIG constant ow!)
- Keep track of which categories have already been Predicted for a current column
- Add a table indexed on nonT's w/ 1 bit to flag 'added'; clear this going to next column
- Or, a prefix table $R(A,B)$ the set of all A s.t. there is at least one grammar rule $A \rightarrow B$.

There can be a lot of such rules

- Consider extracting rules from the 39,000 sentence Wall Street Journal corpus
 - There are 694 rules that expand NPs, from NP → PUNC: NN NN NN PUNC. to... NP → VBN NNS NNS NNS NNS
- This is a BIG constant...



1-word lookahead would help

0	Papa	1
0 ROOT . S	0 NP Papa .	
0 S . NP VP	0 S NP . VP	
0 NP . Det N	0 NP NP . PP	
0 NP . NP PP	1 VP . V NP	
0 NP . Papa	1 VP . VP PP	
0 Det . the	1 PP . P NP	
0 Det . a	1 V . ate	
	1 V . drank	
	1 V . snorted	
	1 P . with	

predict

•P makes us add all the prepositions ...

0	Papa	1	ate
0 ROOT . S	0 NP Papa .		
0 S . NP VP	0 S NP . VP		
0 NP . Det N	0 NP NP . PP		
0 NP . NP PP	1 VP . V NP		
0 NP . Papa	1 VP . VP PP		
0 Det . the	1 PP . P NP		
0 Det . a	1 V . ate		
	1 V . drank		
	1 V . snorted		
	1 P . with		

No point in adding words other than ate

1-word lookahead would help

0	Papa	1	ate
0 ROOT . S	0 NP Papa .		
0 S . NP VP	0 S NP . VP		
0 NP . Det N	0 NP NP . PP		
0 NP . NP PP	1 VP . V NP		
0 NP . Papa	1 VP . VP PP		
0 Det . the	1 PP . P NP		
0 Det . a	1 V . ate		
	1 V . drank		
	1 V . snorted		
	1 P . with		

In fact, no point in adding any constituent that can't start with ate
Don't bother adding PP, P, etc.

No point in adding words other than ate

With Left-Corner Filter

0	Papa	1	ate
0 ROOT . S	0 NP Papa .		
0 S . NP VP	0 S NP . VP		
0 NP . Det N	0 NP NP . PP		
0 NP . NP PP			
0 NP . Papa			
0 Det . the			
0 Det . a			

attach

PP can't start with ate

Birth control – now we won't predict
1 PP . P NP
1 P . with
either!

Need to know that ate can't start PP
Take closure of all categories that it does start ...