Parsing Tricks

Left-Corner Parsing

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

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 in Homework 3?

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 ...

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~~		

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 PP . ate
either!

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

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			
0 Det . a			

predict

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 V . ate
0 Det . a	~~1 V . drank~~
	~~1 V . snorted~~

predict

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 V . ate
0 Det . a	~~1 V . drank~~
	~~1 V . snorted~~

predict

Merging Right-Hand Sides

- Grammar might have rules
  - **X ? A G H P**
  - **X ? B G H P**
- Could end up with both of these in chart:
  - **(2, X ? A . G H P)** in column 5
  - **(2, X ? B . G H P)** in column 5
- But these are now interchangeable: if one produces X then so will the other
- To avoid this redundancy, can always use dotted rules of this form: **X ? ... G H P**

Merging Right-Hand Sides

- Similarly, grammar might have rules
  - **X ? A G H P**
  - **X ? A G H Q**
- Could end up with both of these in chart:
  - **(2, X ? A . G H P)** in column 5
  - **(2, X ? A . G H Q)** in column 5
- Not interchangeable, but we'll be processing them in parallel for a while ...
- Solution: write grammar as **X ? A G H (P | Q)**

Merging Right-Hand Sides

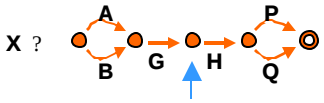
- Combining the two previous cases:
  - **X ? A G H P**
  - **X ? A G H Q**
  - **X ? B G H P**
  - **X ? B G H Q**

becomes

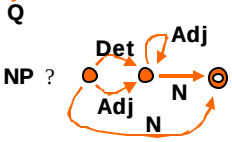
- **X ? (A | B) G H (P | Q)**
- And often nice to write stuff like
  - **NP ? (Det | ?) Adj* N**

Merging Right-Hand Sides

- **X ? (A | B) G H (P | Q)**
- **NP ? (Det | ?) Adj* N**
- These are regular expressions!
- Build their minimal DFAs:



- Automaton states replace dotted rules (**X ? A G . H P**)



Merging Right-Hand Sides

Indeed, *all* **NP** ? rules can be unioned into a single DFA!

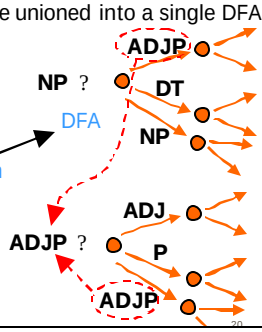
NP ? ADJP ADJP JJ JJ NN NNS
NP ? ADJP DT NN
NP ? ADJP JJ NN
NP ? ADJP JJ NN NNS
NP ? ADJP JJ NNS
NP ? ADJP NN
NP ? ADJP NN NN
NP ? ADJP NN NNS
NP ? ADJP NNS
NP ? ADJP NPR
NP ? ADJP NPRS
NP ? DT
NP ? DT ADJP
NP ? DT ADJP , JJ NN
NP ? DT ADJP ADJP NN
NP ? DT ADJP JJ JJ NN
NP ? DT ADJP JJ NN
NP ? DT ADJP JJ NN NN
etc.

Merging Right-Hand Sides

Indeed, *all* **NP** ? rules can be unioned into a single DFA!

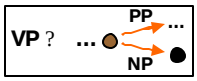
NP ? ADJP ADJP JJ JJ NN NNS
| ADJP DT NN
| ADJP JJ NN
| ADJP JJ NN NNS
| ADJP JJ NNS
| ADJP NN
| ADJP NN NN
| ADJP NN NNS
| ADJP NNS
| ADJP NPR
| ADJP NPRS
| DT
| DT ADJP
| DT ADJP , JJ NN
| DT ADJP ADJP NN
| DT ADJP JJ JJ NN
| DT ADJP JJ NN
| DT ADJP JJ NN NN
etc.

regular expression → DFA



Earley's Algorithm on DFAs

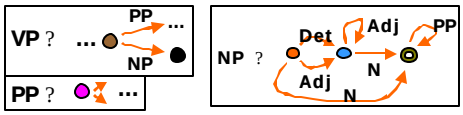
- What does Earley's algorithm now look like?



Column 4
...
(2, ●) predict

Earley's Algorithm on DFAs

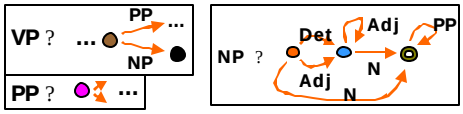
- What does Earley's algorithm now look like?



Column 4
...
(2, ●) predict
(4, ●)
(4, ●)

Earley's Algorithm on DFAs

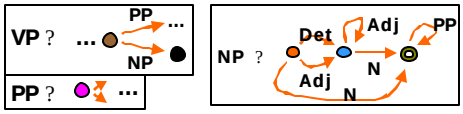
- What does Earley's algorithm now look like?



Column 4	Column 5	...	Column 7
...	...		
(2, ●)			(4, ●) predict or attach?
(4, ●)			
(4, ●) → (4, ●)			

Earley's Algorithm on DFAs

- What does Earley's algorithm now look like?



Column 4	Column 5	...	Column 7
...	...		
(2, ●)			(4, ●) predict or attach?
(4, ●)			(7, ●)
(4, ●) → (4, ●)			(2, ●) Both!

Pruning for Speed

- Heuristically throw away constituents that probably won't make it into best complete parse.
- Use probabilities to decide which ones.
  - So probs are useful for speed as well as accuracy!
- Both safe and unsafe methods exist
  - Throw x away if $p(x) < 10^{-200}$ (and lower this threshold if we don't get a parse)
  - Throw x away if $p(x) < 100 * p(y)$ for some y that spans the same set of words
  - Throw x away if $p(x) * q(x)$ is small, where $q(x)$ is an estimate of probability of all rules needed to combine x with the other words in the sentence

Agenda ("Best-First") Parsing

- Explore best options first
  - Should get some good parses early on – grab one & go!
- Prioritize constits (and dotted constits)
  - Whenever we build something, give it a priority
    - How likely do we think it is to make it into the highest-prob parse?
  - usually related to log prob. of that constit
  - might also hack in the constit's context, length, etc.
  - if priorities are defined carefully, obtain an A* algorithm
- Put each constit on a priority queue (heap)
- Repeatedly pop and process best constituent.
  - CKY style: combine w/ previously popped neighbors.
  - Earley style: scan/predict/attach as usual. What else?

Preprocessing

- First "tag" the input with parts of speech:
  - Guess the correct preterminal for each word, using faster methods we'll learn later
  - Now only allow one part of speech per word
  - This eliminates a lot of crazy constituents!
  - But if you tagged wrong you could be hosed
- Raise the stakes:
  - What if tag says not just "verb" but "transitive verb"? Or "verb with a direct object and 2 PPs attached"? ("supertagging")
- Safer to allow a few possible tags per word, not just one ...

Center-Embedding

```
if x
then
   if y
   then
      if a
      then b
      endif
   else b
   endif
else b
endif
```

STATEMENT → if EXPR then STATEMENT endif

STATEMENT → if EXPR then STATEMENT else STATEMENT endif

But not:

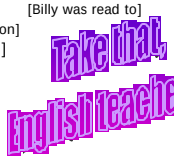
STATEMENT → if EXPR then STATEMENT

Center-Embedding

- This is the rat that ate the malt.
- This is the malt that the rat ate.
- This is the cat that bit the rat that ate the malt.
- This is the malt that the rat that the cat bit ate.
- This is the dog that chased the cat that bit the rat that ate the malt.
- This is the malt that [the rat that [the cat that [the dog chased] bit] ate].

More Center-Embedding

[What did you disguise
[those handshakes that you greeted
[the people we bought
[the bench
[Billy was read to]
on]
with]
with]
for]?



[Which mantelpiece did you put
[the idol I sacrificed
[the fellow we sold
[the bridge you threw
[the bench
[Billy was read to]
on]
off]
to]
on]?

Center Recursion vs. Tail Recursion

[What did you disguise
[those handshakes that
you greeted
[the people we bought
[the bench
[Billy was read to]
on]
with]
with]
for]?

[For what did you disguise
[those handshakes with which
you greeted
[the people with which we bought
[the bench on which
[Billy was read to]?

"pied piping" –
NP moves leftward,
preposition follows along

Disallow Center-Embedding?

- Center-embedding seems to be in the grammar, but people have trouble processing more than 1 level of it.
- You can limit # levels of center-embedding via features: e.g., S[S_DEPTH=n+1] → A S[S_DEPTH=n] B
- If a CFG limits # levels of embedding, then it can be compiled into a finite-state machine – we don't need a stack at all!
  - Finite-state recognizers run in linear time.
  - However, it's tricky to turn them into parsers for the original CFG from which the recognizer was compiled.

Parsing Algs for non-CFG

- If you're going to make up a new kind of grammar, you should also describe how to parse it.
- Such algorithms exist!
- For example, there are parsing algorithms for TAG (where larger tree fragments can be combined by substitution & adjunction)