

# 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		
	<del>1 V . drank</del>		
	<del>1 V . snorted</del>		
	<del>1 P . with</del>		

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	<del>0 NP NP . PP</del>		
0 NP . NP PP	1 VP . V NP		
0 NP . Papa	1 VP . VP PP		
0 Det . the	<del>1 PP . P NP</del>		
0 Det . a	1 V . ate		
	<del>1 V . drank</del>		
	<del>1 V . snorted</del>		
	<del>1 P . with</del>		

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	<del>0 NP NP . PP</del>		
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 ...

0	Papa	1	ate
0 ROOT . S	0 NP Papa .		
0 S . NP VP	0 S NP . VP		
0 NP . Det N	<del>0 NP NP . PP</del>		
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	<del>0 NP NP . PP</del>		
0 NP . NP PP	1 VP . V NP		
0 NP . Papa	1 VP . VP PP		
0 Det . the	1 V . ate		
0 Det . a	<del>1 V . drank</del>		
	<del>1 V . snorted</del>		

predict

0	Papa	1	ate
0 ROOT . S	0 NP Papa .		
0 S . NP VP	0 S NP . VP		
0 NP . Det N	<del>0 NP NP . PP</del>		
0 NP . NP PP	1 VP . V NP		
0 NP . Papa	1 VP . VP PP		
0 Det . the	1 V . ate		
0 Det . a	<del>1 V . drank</del>		
	<del>1 V . snorted</del>		

predict

## Merging Right-Hand Sides

- Grammar might have rules
  - $X \rightarrow A G H P$
  - $X \rightarrow B G H P$
- Could end up with both of these in chart:
  - (2,  $X \rightarrow A . G H P$ ) in column 5
  - (2,  $X \rightarrow B . G H P$ ) 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:  $X \rightarrow \dots G H P$

## Merging Right-Hand Sides

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

## Merging Right-Hand Sides

- Combining the two previous cases:
  - $X \rightarrow A G H P$
  - $X \rightarrow A G H Q$
  - $X \rightarrow B G H P$
  - $X \rightarrow B G H Q$

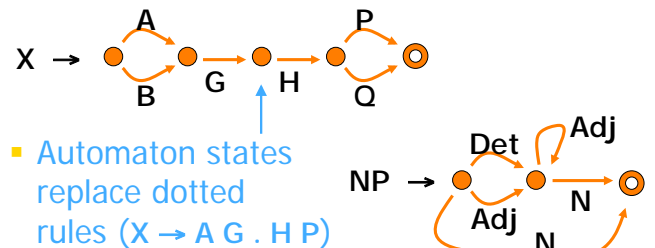
becomes

- $X \rightarrow (A | B) G H (P | Q)$
- And often nice to write stuff like
  - $NP \rightarrow (Det | \epsilon) Adj^* N$

## Merging Right-Hand Sides

$X \rightarrow (A | B) G H (P | Q)$   
 $NP \rightarrow (Det | \epsilon) Adj^* N$

- These are regular expressions!
- Build their minimal DFAs:



- Automaton states replace dotted rules ( $X \rightarrow A G . H P$ )

# 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

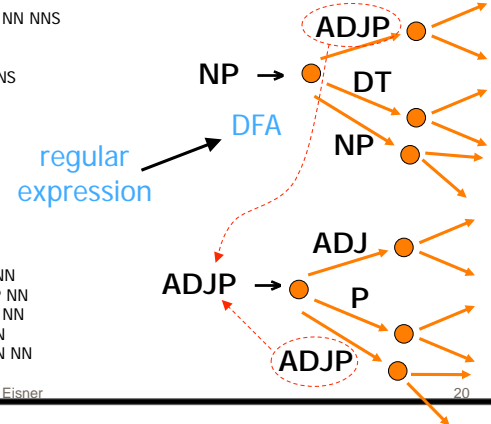
etc.

# Merging Right-Hand Sides

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

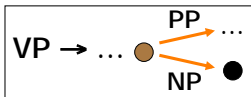
- NP → ADJP ADJP JJ JJ NN NNS
- ADJP DT NN
- ADJP JJ NN
- ADJP JJ NN NNS
- ADJP JJ NNS
- ADJP NN
- ADJP NN NN
- ADJP NN NNS
- ADJP NNS
- ADJP NPR
- ADJP NPRS
- DT
- DT ADJP
- DT ADJP , JJ NN
- DT ADJP ADJP NN
- DT ADJP JJ JJ NN
- DT ADJP JJ NN
- DT ADJP JJ NN NN

etc.



# Earley's Algorithm on DFAs

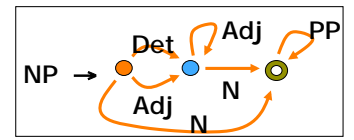
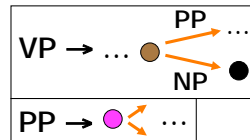
What does Earley's algorithm now look like?



Column 4
...
(2, ●) predict

# Earley's Algorithm on DFAs

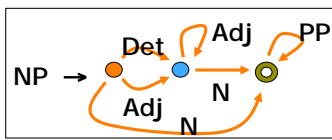
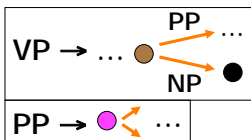
What does Earley's algorithm now look like?



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

# Earley's Algorithm on DFAs

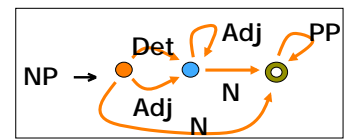
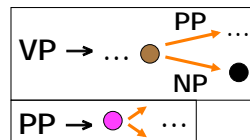
What does Earley's algorithm now look like?



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

# 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, ●) Both!
(4, ○)			(2, ●)

# Pruning and Prioritization

- 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
  - Iterative deepening: Throw x away if  $p(x) < 10^{-200}$  (and lower this threshold if we don't get a parse)
  - Heuristic pruning: Throw x away if  $p(x) < 100 * p(y)$  for some y that spans the same set of words (for example)
  - Prioritization: If  $p(x)$  is low, don't throw x away; just postpone using it until you need it (hopefully you won't).

# Prioritization continued: Agenda-Based Parsing

- Prioritization: If  $p(x)$  is low, don't throw x away; just postpone using it until you need it.
  - In other words, explore best options first.
  - Should get some good parses early on; then stop!

time	1	flies	2	like	3	an	4	arrow	5
0	NP 3		NP 10					NP 24	
	Vst 3		S 8					S 22	
1			NP 4					NP 18	
			VP 4					S 21	
								VP 18	
2					P 2			PP 12	
					V 5			VP 16	
3						Det 1		NP 10	
4								N 8	

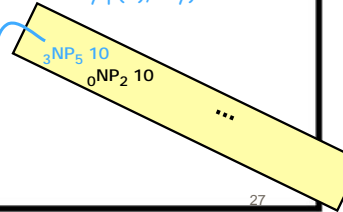
# Prioritization continued: Agenda-Based Parsing

- until we pop a parse  ${}_0S_5$  or fail with empty agenda
  - pop top element  ${}_iY_j$  from agenda into chart
  - for each right neighbor  ${}_jZ_k$ 
    - for each rule  $X \rightarrow YZ$  in grammar
    - put  ${}_iX_k$  onto the agenda
  - for each left neighbor  ${}_hZ_i$ 
    - for each rule  $X \rightarrow ZY$
    - put  ${}_hX_j$  onto the agenda

chart of good constituents

prioritized agenda of pending constituents (ordered by  $p(x)$ , say)

time	1	flies	2	like	3	an	4	arrow	5
0	NP 3								
	Vst 3		S 8						
1			NP 4						
			VP 4						
2					P 2				
					V 5				
3						Det 1			
4								N 8	



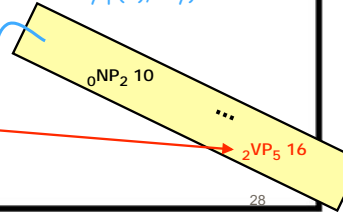
# Prioritization continued: Agenda-Based Parsing

- until we pop a parse  ${}_0S_5$  or fail with empty agenda
  - pop top element  ${}_iY_j$  from agenda into chart
  - for each right neighbor  ${}_jZ_k$ 
    - for each rule  $X \rightarrow YZ$  in grammar
    - put  ${}_iX_k$  onto the agenda
  - for each left neighbor  ${}_hZ_i$ 
    - for each rule  $X \rightarrow ZY$
    - put  ${}_hX_j$  onto the agenda

chart of good constituents

prioritized agenda of pending constituents (ordered by  $p(x)$ , say)

time	1	flies	2	like	3	an	4	arrow	5
0	NP 3								
	Vst 3		S 8						
1			NP 4						
			VP 4						
2					P 2				
					V 5				
3						Det 1		NP 10	
4								N 8	



# Prioritization continued: Agenda-Based Parsing

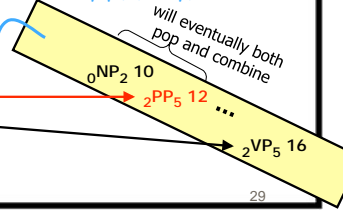
always finds best parse! analogous to Dijkstra's shortest-path algorithm

- until we pop a parse  ${}_0S_5$  or fail with empty agenda
  - pop top element  ${}_iY_j$  from agenda into chart
  - for each right neighbor  ${}_jZ_k$ 
    - for each rule  $X \rightarrow YZ$  in grammar
    - put  ${}_iX_k$  onto the agenda
  - for each left neighbor  ${}_hZ_i$ 
    - for each rule  $X \rightarrow ZY$
    - put  ${}_hX_j$  onto the agenda

chart of good constituents

prioritized agenda of pending constituents (ordered by  $p(x)$ , say)

time	1	flies	2	like	3	an	4	arrow	5
0	NP 3								
	Vst 3		S 8						
1			NP 4						
			VP 4						
2					P 2				
					V 5				
3						Det 1		NP 10	
4								N 8	



# Outside Estimates for better Pruning and Prioritization

- Iterative deepening: Throw x away if  $p(x) * q(x) < 10^{-200}$  (lower this threshold if we don't get a parse)
- Heuristic pruning: Throw x away if  $p(x) * q(x) < 100 * p(y) * q(y)$  for some y that spans the same set of words
- Prioritized agenda: Priority of x on agenda is  $p(x) * q(x)$ ; stop at first parse
- In general, the "inside prob"  $p(x)$  will be higher for smaller constituents
  - Not many rule probabilities inside them
- The "outside prob"  $q(x)$  is intended to correct for this
  - Estimates the prob of all the rest of the rules needed to build x into full parse
  - So  $p(x) * q(x)$  estimates prob of the best parse that contains x
  - If we take  $q(x)$  to be the best estimate we can get
    - Methods may no longer be safe (but may be fast!)
    - Prioritized agenda is then called a "best-first algorithm"
  - But if we take  $q(x) = 1$ , that's just the methods from previous slides
    - And iterative deepening and prioritization were safe there
  - If we take  $q(x)$  to be an "optimistic estimate" (always  $\geq$  true prob)
    - Still safe! Prioritized agenda is then an example of an "A\* algorithm"

## Outside Estimates for better Pruning and Prioritization

- **Iterative deepening:** Throw  $x$  away if  $p(x) * q(x) < 10^{-200}$  (lower this threshold if we don't get a parse)
- **Heuristic pruning:** Throw  $x$  away if  $p(x) * q(x) < 100 * p(y) * q(y)$  for some  $y$  that spans the same set of words
- **Prioritized agenda:** Priority of  $x$  on agenda is  $p(x) * q(x)$ ; stop at first parse

- In general, the "inside prob"  $p(x)$  will be higher for smaller constituents
  - Not many rule probabilities inside them
- The "outside prob"  $q(x)$  is intended to correct for this
  - Estimates the prob of all the rest of the rules needed to build  $x$  into full parse
  - So  $p(x) * q(x)$  estimates prob of the best parse that contains  $x$
  - If we take  $q(x)$  to be the best estimate we can get

**Terminology warning:** Here "inside" and "outside" mean probability of the best partial parse inside or outside  $x$

But traditionally, they mean total prob of all such partial parses (as in the "inside algorithm" that we saw in the previous lecture)

## 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  $\rightarrow$  if EXPR then STATEMENT endif

STATEMENT  $\rightarrow$  if EXPR then STATEMENT else STATEMENT endif

But not:

```

STATEMENT  $\rightarrow$  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]?

**Take that, English teachers!**

## 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] on] with] with] for]?

**"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.,  $S[S\_DEPTH=n+1] \rightarrow A S[S\_DEPTH=n] B$
- 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.
  - And compiling a **small** grammar into a much **larger** FSA may be a net loss – structure sharing in the parse chart is expanded out to duplicate structure all over the FSA.

## 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, e.g.,
  - for TAG (where the grammar specifies not just rules but larger tree fragments, which can be combined by "substitution" and "adjunction" operations)
  - for CCG (where the grammar only specifies preterminal rules, and there are generic operations to combine slashed nonterminals like  $X/Y$  or  $(X/Z)/(Y\backslash W)$ )