Computational Linguistics 1

CMSC/LING 723, LBSC 744

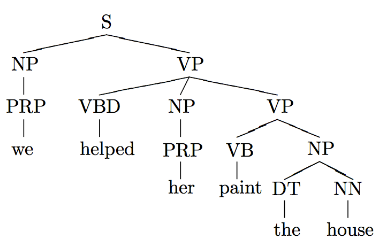
Kristy Hollingshead Seitz
Institute for Advanced Computer Studies
University of Maryland

Lecture 17: 1 November 2011

Agenda

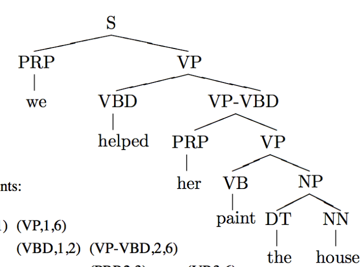
- HW4, due Thursday
- Questions, comments, concerns?
- Parsing algorithms
  - Left-corner grammar transformation
  - Earley parsing
- Context-sensitive grammar formalisms?

Parse Tree, Derivation



leftmost derivation
S → NP VP
NP → PRP
PRP → we
VP → VBD NP VP
VBD → helped
NP → PRP
PRP → her
VP → VB NP
VB → paint
NP → DT NN
DT → the
NN → house

Parse Tree, CNF



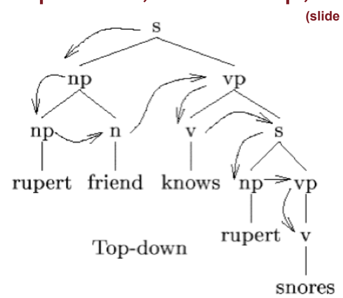
constituents:
(S,0,6)
(PRP,0,1) (VP,1,6)
(VBD,1,2) (VP-VBD,2,6)
(PRP,2,3) (VP,3,6)
(VB,3,4) (NP,4,6)
(DT,4,5) (NN,5,6)

CYK Chart, span 4, midpoint 3

Span				
4	(NP, 0.015, 1, NN, NP) (NP, 0.025, 2, NP, NP) (NP, 0.045, 3, NP, NN)			
3	(NP, 0.15, 2, NP, NN)	(NP, 0.15, 3, NP, NN)		
2	(NP, 0.5, 1, NN, NN)	(NP, 0.5, 2, NN, NN)	(NP, 0.5, 3, NN, NN)	
1	NN	NN	NN	NN

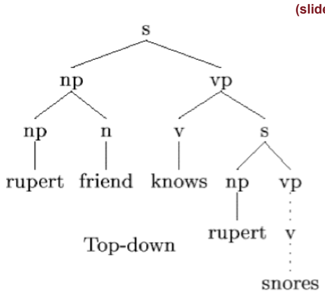
Top-down, Bottom-up, Left-corner

(slide adapted from Mark Johnson)



Top-down, Bottom-up, Left-corner

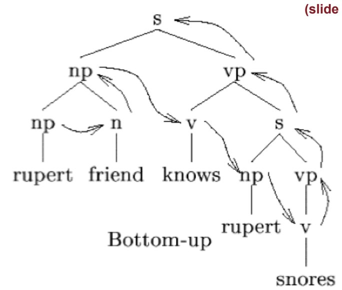
(slide adapted from Mark Johnson)



Intuitive?

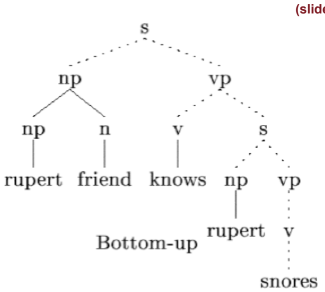
Top-down, Bottom-up, Left-corner

(slide adapted from Mark Johnson)



Top-down, Bottom-up, Left-corner

(slide adapted from Mark Johnson)



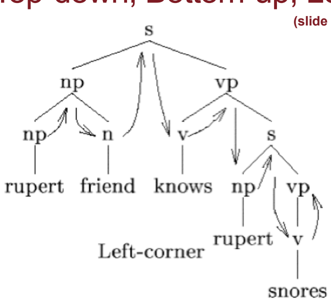
Intuitive?

Left-corner Parsing

- The left corner of a context-free rule is the first symbol on the right hand side:
  S → NP VP: left corner is NP.
- The left-corner of each production is recognized bottom-up, and everything else is predicted top-down

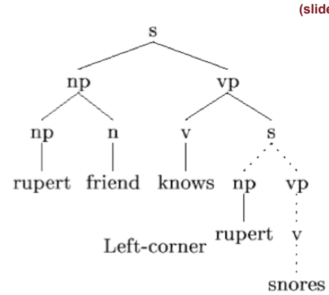
Top-down, Bottom-up, Left-corner

(slide adapted from Mark Johnson)



Top-down, Bottom-up, Left-corner

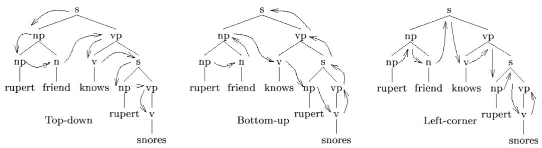
(slide adapted from Mark Johnson)



Intuitive?

Top-down, Bottom-up, Left-corner

(slide adapted from Mark Johnson)



- Top-down:
  - Right-recursive grammars require finite state size
  - But left-recursive grammars require *unbounded* state size
- Left-corner
  - Finite-state size for both left-recursive and right-recursive grammars
  - Only center-embedded structures require unbounded stacks

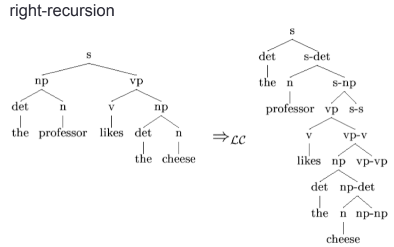
Top-down, Bottom-up, Left-corner

- Top-down:
  - Right-recursive grammars require finite state size
  - But left-recursive grammars require *unbounded* state size
- Left-corner
  - Finite-state size for both left-recursive and right-recursive grammars
  - Only center-embedded structures require unbounded stacks
  - ...which emulates human behavior!
- From [Resnik, 1992]:

Strategy	Space required		
	Left	*Center*	*Right*
Top-down	$O(n)$	$O(n)$	$O(1)$
Bottom-up	$O(1)$	$O(n)$	$O(n)$
Left-corner†	$O(1)$	$O(n)$	$O(1)$
What people do	$O(1)$	$O(n)$	$O(1)$

Building a Left-corner Parser?

- Perform a left-corner transform on grammar G, then can use a top-down parser
  - because the LC-transform converts left-recursion into right-recursion



Left-corner Grammar Transform

- A → aA-a for all A∈V , a ∈ T
- A → A-C for all A∈V , C → ε ∈ P
- A-X→βA-B for all A∈V , B→Xβ∈P
- A-A → ε for all A∈V

- After transforming the grammar, do ... what?

CYK!

Agenda

- HW4, due Thursday
- Parsing algorithms
  - Left-corner grammar transform
  - Earley parsing
- Context-sensitive grammar formalisms

CKY: Analysis

- Since it's bottom up, CKY populates the table with a lot of "phantom constituents"
  - Spans that are constituents, but cannot really occur in the context in which they are suggested
- Conversion of grammar to CNF adds additional non-terminal nodes
  - Leads to weak equivalence wrt original grammar
  - Additional terminal nodes not (linguistically) meaningful: but can be cleaned up with post processing
- Is there a parsing algorithm for arbitrary CFGs that combines dynamic programming and top-down control?

Earley Parsing Algorithm

- One advantage of top-down over bottom-up is that one never builds constituents that cannot be rooted
- Earley parsing motivation
  - Only want to build categories that can be rooted
  - Use a top-down *filter*
  - Use a chart parsing approach
- Dynamic programming algorithm (surprise)
- Allows arbitrary CFGs
- Fills a chart in a single sweep over the input

Earley Parsing: Chart, States

- Chart is an array of length $N + 1$, where N = number of words
- Chart entries represent states:
  - Completed constituents and their locations
  - In-progress constituents
  - Predicted constituents
- Each state contains three items of information:
  - A grammar rule
  - Information about progress made in completing the sub-tree represented by the rule
  - Span of the sub-tree

Chart Entries: State Examples

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

Earley in a nutshell

- Start by predicting S
- Step through chart:
  - New predicted states are created from current states
  - New incomplete states are created by advancing existing states as new constituents are discovered
  - States are completed when rules are satisfied
- Termination: look for S → α • [0, *N*]

Earley Algorithm

```
function EARLEY-PARSE(words, grammar) returns chart

  ENQUEUE((γ → • 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)
```


Earley Algorithm

```
procedure PREDICTOR((A → α • B β, [i, j]))
   for each (B → γ) in GRAMMAR-RULES-FOR(B, grammar) do
        ENQUEUE((B → • γ, [j, j]), chart[j])
   end

procedure SCANNER((A → α • B β, [i, j]))
   if B ⊂ PARTS-OF-SPEECH(word[j]) then
       ENQUEUE((B → word[j], [j, j+1]), chart[j+1])

procedure COMPLETER((B → γ •, [j,k]))
   for each (A → α • B β, [i, j]) in chart[j] do
        ENQUEUE((A → α B • β, [i,k]), chart[k])
   end
```

Earley Example

- Input: Book that flight
- Desired end state: S → α • [0,3]
  - Meaning: S spanning from 0 to 3, completed rule

Earley: Chart[0]

S0	*γ* → • *S*	[0,0]	Dummy start state
S1	*S* → • *NP VP*	[0,0]	Predictor
S2	*S* → • *Aux NP VP*	[0,0]	Predictor
S3	*S* → • *VP*	[0,0]	Predictor
S4	*NP* → • *Pronoun*	[0,0]	Predictor
S5	*NP* → • *Proper-Noun*	[0,0]	Predictor
S6	*NP* → • *Det Nominal*	[0,0]	Predictor
S7	*VP* → • *Verb*	[0,0]	Predictor
S8	*VP* → • *Verb NP*	[0,0]	Predictor
S9	*VP* → • *Verb NP PP*	[0,0]	Predictor
S10	*VP* → • *Verb PP*	[0,0]	Predictor
S11	*VP* → • *VP PP*	[0,0]	Predictor

Note that given a grammar, these entries are the same for all inputs; they can be pre-loaded…

Earley: Chart[1]

S12	*Verb* → *book* •	[0,1]	Scanner
S13	*VP* → *Verb* •	[0,1]	Completer
S14	*VP* → *Verb* • *NP*	[0,1]	Completer
S15	*VP* → *Verb* • *NP PP*	[0,1]	Completer
S16	*VP* → *Verb* • *PP*	[0,1]	Completer
S17	*S* → *VP* •	[0,1]	Completer
S18	*VP* → *VP* • *PP*	[0,1]	Completer
S19	*NP* → • *Pronoun*	[1,1]	Predictor
S20	*NP* → • *Proper-Noun*	[1,1]	Predictor
S21	*NP* → • *Det Nominal*	[1,1]	Predictor
S22	*PP* → • *Prep NP*	[1,1]	Predictor

Earley: Chart[2] and Chart[3]

S23	*Det* → *that* •	[1,2]	Scanner
S24	*NP* → *Det* • *Nominal*	[1,2]	Completer
S25	*Nominal* → • *Noun*	[2,2]	Predictor
S26	*Nominal* → • *Nominal Noun*	[2,2]	Predictor
S27	*Nominal* → • *Nominal PP*	[2,2]	Predictor
S28	*Noun* → *flight* •	[2,3]	Scanner
S29	*Nominal* → *Noun* •	[2,3]	Completer
S30	*NP* → *Det Nominal* •	[1,3]	Completer
S31	*Nominal* → *Nominal* • *Noun*	[2,3]	Completer
S32	*Nominal* → *Nominal* • *PP*	[2,3]	Completer
S33	*VP* → *Verb NP* •	[0,3]	Completer
S34	*VP* → *Verb NP* • *PP*	[0,3]	Completer
S35	*PP* → • *Prep NP*	[3,3]	Predictor
S36	*S* → *VP* •	[0,3]	Completer
S37	*VP* → *VP* • *PP*	[0,3]	Completer

Earley: Recovering the Parse

As with CKY, add backpointers…

Chart[1]	S12	*Verb* → *book* •	[0,1]	Scanner
Chart[2]	S23	*Det* → *that* •	[1,2]	Scanner
Chart[3]	S28	*Noun* → *flight* •	[2,3]	Scanner
	S29	*Nominal* → *Noun* •	[2,3]	(S28)
	S30	*NP* → *Det Nominal* •	[1,3]	(S23, S29)
	S33	*VP* → *Verb NP* •	[0,3]	(S12, S30)
	S36	*S* → *VP* •	[0,3]	(S33)

Earley: Efficiency

- For such a simple example, there seems to be a lot of useless stuff…
- Why?

Back to Ambiguity

- Did we solve it?
- No: both CKY and Earley return multiple parse trees…
  - Plus: compact encoding with shared sub-trees
  - Plus: work deriving shared sub-trees is reused
  - Minus: neither algorithm tells us which parse is correct

Ambiguity

- Why don't humans usually encounter ambiguity?
- How can we improve our models?

Agenda: Summary

- HW4, due Thursday
- Parsing algorithms
  - Earley parsing
  - Left-corner grammar transform
- Next time: context-sensitive grammar formalisms