Speech and Language Processing

SLP Chapter 13 (continued)

Parsing

Today

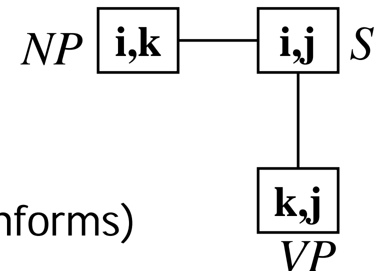
- Parsing with CFGs
  - CKY
  - Earley
- Nitin: CCG

Dynamic Programming

- DP search methods fill tables with partial results and thereby
  - Avoid doing avoidable repeated work
  - Solve exponential problems in polynomial time (well, no not really)
  - Efficiently store ambiguous structures with shared sub-parts.
- We'll cover two approaches that roughly correspond to top-down and bottom-up approaches.
  - CKY
  - Earley

Conversion to CNF

- Procedure:
  - Copy all conforming rules unchanged
  - Convert terminals w/in rules to dummy non-terminals
  - Convert unit productions (this is a flattening operation!)
  - Binarize rules
- Example:
  - S→ VP
  - VP → $Verb_{trans}$ NP // Copy this over (it already conforms)
  - VP → V NP PP
  - INF-VP → to VP
- Rule 4 becomes 4′ and 4″:
  - INF-VP → TO VP // Convert terminals to dummy non-terminals
  - TO → to
- Rule 1 becomes 1′ and 1″:
  - S → $Verb_{trans}$ NP // Convert unit productions
  - S → V NP PP
- Rules 3 and 1″ become: // Binarize
  - S → V Double-Comp
  - VP → V Double-Comp
  - Double-Comp → NP PP



CYK Algorithm

- Classic example of dynamic programming paradigm
- Fill in upper-triangular matrix (chart) in bottom-up fashion using CNF grammar, where chart(i,j) = $\{A \mid A \rightarrow^* w_{i+1} \ldots w_j\}$
- Note that cells include **sets** of non-terminals.

TO:

FROM:	1	2	3	4	5	6
0	0–1	0–2	0–3	0–4	0–5	0–6
1		1–2	1–3	1–4	1–5	1–6
2			2–3	2–4	2–5	2–6
3				3–4	3–5	3–6
4					4–5	4–6
5						5–6

CKY

- Build a table so that an A spanning from i to j in the input is placed in cell [i,j] in the table.
- So a non-terminal spanning an entire string will sit in cell [0, n]
  - Hopefully an *S*
- If we build the table bottom-up, we'll know that the parts of the A must go from i to k and from k to j, for some k.

CKY

- Meaning that for a rule like A → B C we should look for a B in [i,k] and a C in [k,j].
- In other words, if we think there might be an A spanning i,j in the input… AND

  A → B C is a rule in the grammar THEN

- There must be a B in [i,k] and a C in [k,j] for some i<k<j

CKY

- So to fill the table loop over the cell[i,j] values in some systematic way
  - What constraint should we put on that systematic search?
  - For each cell, loop over the appropriate k values to search for things to add.

CKY Algorithm

function CKY-PARSE(*words, grammar*) **returns** *table*

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**

$table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$

for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**

for $k \leftarrow i+1$ **to** $j-1$ **do**

$$table[i,j] \leftarrow table[i,j] \cup \{A \mid A \rightarrow BC \in grammar, B \in table[i,k], C \in table[k,j]\}$$

CKY Parsing

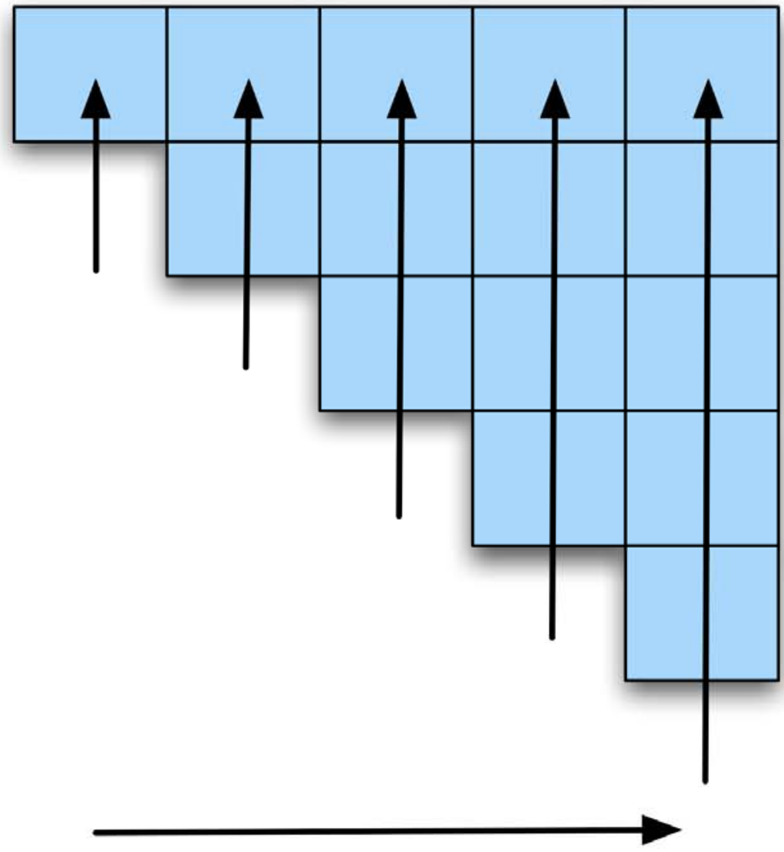
- Is that really a parser?

Note

- We arranged the loops to fill the table a column at a time, from left to right, bottom to top.
  - This assures us that whenever we're filling a cell, the parts needed to fill it are already in the table (to the left and below)
  - It's somewhat natural in that it processes the input a left to right a word at a time
    - Known as online

Example

Book	the	flight	through	Houston
S, VP, Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	S,VP,X2 [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper-Noun [4,5]



Example

Book	*the*	*flight*	*through*	*Houston*
S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3]	[1,4]	[1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	[3,5]
				NP, Proper-Noun [4,5]

Filling column 5

Example

	Book	*the*	*flight*	*through*	*Houston*
	S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
		Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
			Nominal, Noun [2,3]	[2,4]	[2,5]
				Prep [3,4]	PP [3,5]
					NP, Proper-Noun [4,5]

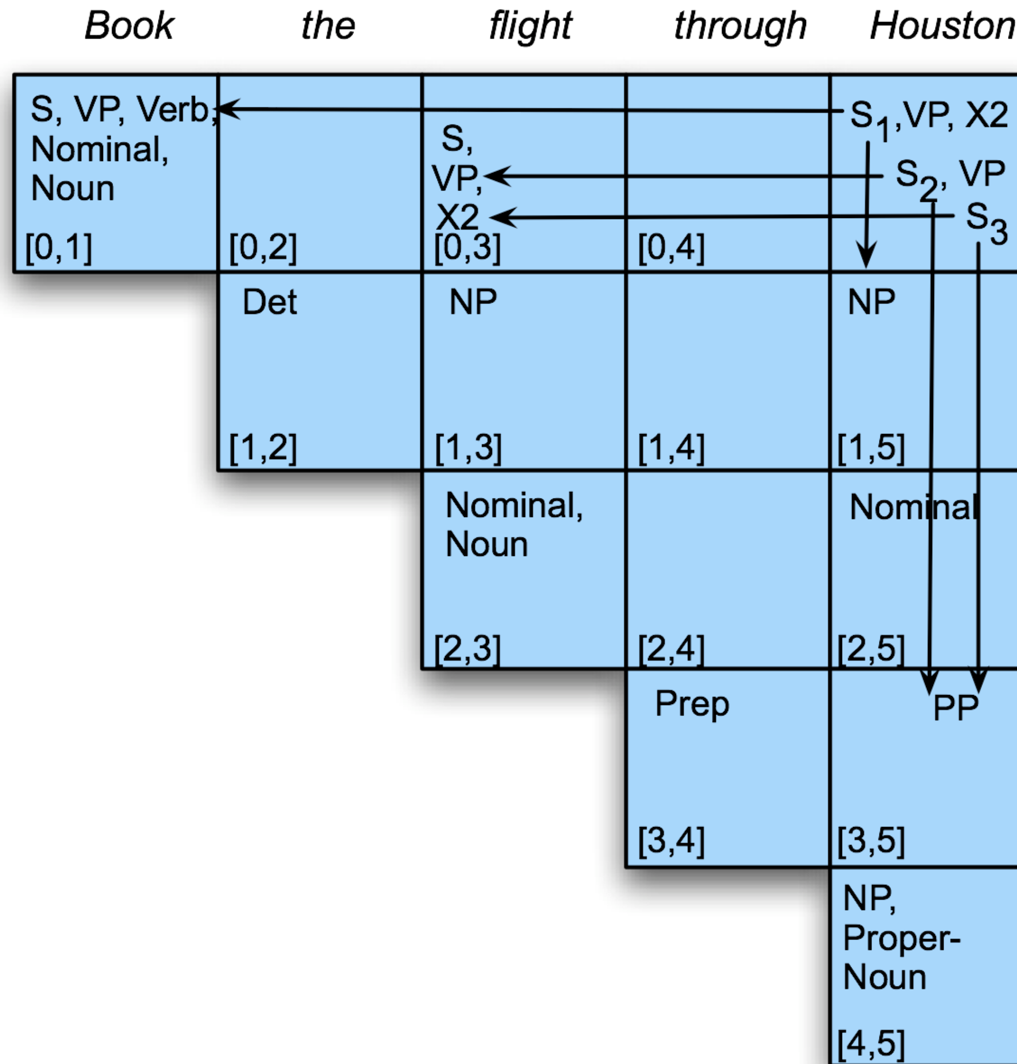
Example

Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper-Noun [4,5]

Example

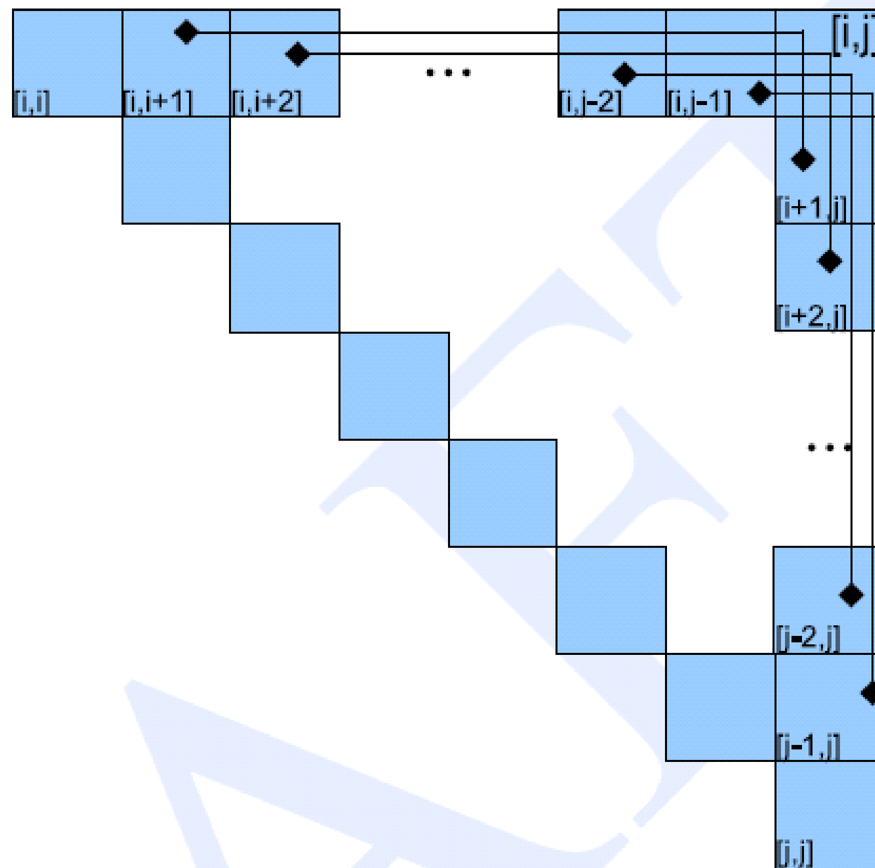
Book	the	flight	through	Houston
S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper-Noun [4,5]

Example



How does Dynamic Programming Provide Savings?

- Multiple ways to fill cells with the same non-terminal—but only **one** copy is included in the cell.



This shows all the ways to fill the [i,j] cell

Disadvantages of CYK

- Primary disadvantage: Conversion to CNF.
  - Leads to “weak equivalence” w.r.t. original grammar
  - Introduces non-linguistically relevant categories e.g., X2
  - Complicates semantic analysis
  - Adds a level of “hidden” complexity. Parse time is Gn^3 (where G is size of grammar) but conversion to CNF throws in another factor of G: G^2n^3 — same as Earley!
- Since it’s bottom up, CKY populates the table with a lot of phantom constituents.
  - Segments that by themselves are constituents but cannot really occur in the context in which they are being suggested.
  - To avoid this we can switch to a top-down control strategy
  - Or we can add some kind of filtering that blocks constituents where they can not happen in a final analysis.
- Is there a parsing approach that adopts dynamic programming and accepts arbitrary CFGs?

Earley Parsing

- Allows arbitrary CFGs
- Top-down control
- Fills a table in a single sweep over the input
  - Table is length N+1; N is number of words
  - Table entries represent
    - Completed constituents and their locations
    - In-progress constituents
    - Predicted constituents

States

- The table-entries are called states and are represented with dotted-rules.

S → · VP	A VP is predicted
NP → Det · Nominal	An NP is in progress
VP → V NP ·	A VP has been found

States/Locations

- S → • VP [0,0]
- NP → Det • Nominal [1,2]
- VP → V NP • [0,3]

- A VP is predicted at the start of the sentence
- An NP is in progress; the Det goes from 1 to 2
- A VP has been found starting at 0 and ending at 3

Earley

- As with most dynamic programming approaches, the answer is found by looking in the table in the right place.
- In this case, there should be an S state in the final column that spans from 0 to N and is complete. That is,
  - $S \rightarrow \alpha \bullet [0,N]$
- If that's the case you're done.

Earley

- So sweep through the table from 0 to N…
  - New predicted states are created by starting top-down from S
  - New incomplete states are created by advancing existing states as new constituents are discovered
  - New complete states are created in the same way.

Earley

- More specifically…
  1. *Predict* all the states you can upfront
  2. Read a word
     1. Extend states based on matches
     2. Generate new predictions
     3. Go to step 2
  3. When you're out of words, look at the chart to see if you have a winner

Core Earley Code

```
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 Code

```
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
```

Example

- Book that flight
- We should find… an S from 0 to 3 that is a completed state…

Chart[0]

S0	$\gamma \rightarrow \bullet S$	[0,0]	Dummy start state
S1	$S \rightarrow \bullet NP\ VP$	[0,0]	Predictor
S2	$S \rightarrow \bullet Aux\ NP\ VP$	[0,0]	Predictor
S3	$S \rightarrow \bullet VP$	[0,0]	Predictor
S4	$NP \rightarrow \bullet Pronoun$	[0,0]	Predictor
S5	$NP \rightarrow \bullet Proper\text{-}Noun$	[0,0]	Predictor
S6	$NP \rightarrow \bullet Det\ Nominal$	[0,0]	Predictor
S7	$VP \rightarrow \bullet Verb$	[0,0]	Predictor
S8	$VP \rightarrow \bullet Verb\ NP$	[0,0]	Predictor
S9	$VP \rightarrow \bullet Verb\ NP\ PP$	[0,0]	Predictor
S10	$VP \rightarrow \bullet Verb\ PP$	[0,0]	Predictor
S11	$VP \rightarrow \bullet VP\ PP$	[0,0]	Predictor

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

Chart[1]

S12	$Verb \rightarrow book \bullet$	[0,1]	Scanner
S13	$VP \rightarrow Verb \bullet$	[0,1]	Completer
S14	$VP \rightarrow Verb \bullet NP$	[0,1]	Completer
S15	$VP \rightarrow Verb \bullet NP\ PP$	[0,1]	Completer
S16	$VP \rightarrow Verb \bullet PP$	[0,1]	Completer
S17	$S \rightarrow VP \bullet$	[0,1]	Completer
S18	$VP \rightarrow VP \bullet PP$	[0,1]	Completer
S19	$NP \rightarrow \bullet Pronoun$	[1,1]	Predictor
S20	$NP \rightarrow \bullet Proper\text{-}Noun$	[1,1]	Predictor
S21	$NP \rightarrow \bullet Det\ Nominal$	[1,1]	Predictor
S22	$PP \rightarrow \bullet Prep\ NP$	[1,1]	Predictor

Charts[2] and [3]

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

Details

- As with CKY that isn't a parser until we add the backpointers so that each state knows where it came from.

Efficiency

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

  - It's predicting things that aren't consistent with the input
  - That's the flipside to the CKY problem.

Back to Ambiguity

- Did we solve it?

Ambiguity

- No...
  - Both CKY and Earley will result in multiple S structures for the [0,N] table entry.
  - They both efficiently store the sub-parts that are shared between multiple parses.
  - And they obviously avoid re-deriving those sub-parts.
  - But neither can tell us which one is right.

Ambiguity

- In most cases, humans don't notice incidental ambiguity (lexical or syntactic). It is resolved on the fly and never noticed.
- Can be modeled with probabilities—something that would be covered in an advanced NLP course.