CSCI 5832 Natural Language Processing

Jim Martin

Lecture 13

2/28/08 1

Today 2/28

- Finish Grammars
  - Treebanks
- Parsing

2/28/08 2

Grammars

- Before you can parse you need a grammar.
- So where do grammars come from?
  - Grammar Engineering
    - Lovingly hand-crafted decades-long efforts by humans to write grammars (typically in some particular grammar formalism of interest to the linguists developing the grammar).
  - TreeBanks
    - Semi-automatically generated sets of parse trees for the sentences in some corpus. Typically in a generic lowest common denominator formalism (of no particular interest to any modern linguist).

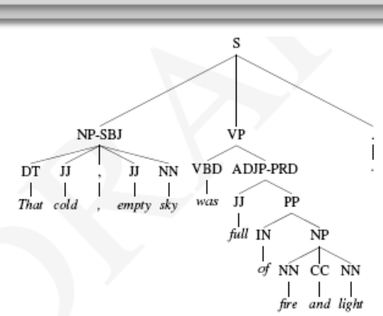
2/28/08 3

TreeBank Grammars

- Reading off the grammar…
- The grammar is the set of rules (local subtrees) that occur in the annotated corpus
- They tend to avoid recursion (and elegance and parsimony)
  - Ie. they tend to the flat and redundant
- Penn TreeBank (III) has about 17500 grammar rules under this definition.

2/28/08 4

TreeBanks



2/28/08 5

TreeBanks

```
((S
   (NP-SBJ (DT That)
     (JJ cold) (, ,)
     (JJ empty) (NN sky) )
   (VP (VBD was)
     (ADJP-PRD (JJ full)
       (PP (IN of)
         (NP (NN fire)
           (CC and)
           (NN light) ))))
   (. .) ))
```

2/28/08 6

Sample Rules

```
NP → DT JJ NNS
NP → DT JJ NN NN
NP → DT JJ JJ NN
NP → DT JJ CD NNS
NP → RB DT JJ NN NN
NP → RB DT JJ JJ NNS
NP → DT JJ JJ NNP NNS
NP → DT NNP NNP NNP NNP JJ NN
NP → DT JJ NNP CC JJ JJ NN NNS
NP → RB DT JJS NN NN SBAR
NP → DT VBG JJ NNP NNP CC NNP
NP → DT JJ NNS , NNS CC NN NNS NN
NP → DT JJ JJ VBG NN NNP NNP FW NNP
NP → NP JJ , JJ `` SBAR '' NNS
```

2/28/08 7

Example

NP → NP JJ , JJ `` SBAR '' NNS

(11.10) [$_{NP}$ Shearson's] [$_{JJ}$ easy-to-film], [$_{JJ}$ black-and-white] "[$_{SBAR}$ Where We Stand]" [$_{NNS}$ commercials]

2/28/08 8

TreeBanks

- TreeBanks provide a grammar (of a sort).
- As we'll see they also provide the training data for various ML approaches to parsing.
- But they can also provide useful data for more purely linguistic pursuits.
  - You might have a theory about whether or not something can happen in particular language.
  - Or a theory about the contexts in which something can happen.
  - TreeBanks can give you the means to explore those theories. If you can formulate the questions in the right way and get the data you need.

2/28/08 9

Tgrep

- You might for example like to grep through a file filled with trees.

```
NP < JJ . VP
```

```
(NP (NP (DT the) (JJ austere) (NN company) (NN dormitory))
  (VP (VBN run)
     (PP (IN by) (NP (DT a) (JJ prying) (NN caretaker)))))
```

2/28/08 10

TreeBanks

- Finally, you should have noted a bit of a circular argument here.
- Treebanks provide a grammar because we can read the rules of the grammar out of the treebank.
- But how did the trees get in there in the first place? There must have been a grammar theory in there someplace…

2/28/08 11

TreeBanks

- Typically, not all of the sentences are hand-annotated by humans.
- They're automatically parsed and then hand-corrected.

2/28/08 12

Parsing

- Parsing with CFGs refers to the task of assigning correct trees to input strings
- Correct here means a tree that covers all and only the elements of the input and has an S at the top
- It doesn't actually mean that the system can select the correct tree from among all the possible trees

2/28/08 13

Parsing

- As with everything of interest, parsing involves a search which involves the making of choices
- We'll start with some basic (meaning bad) methods before moving on to the one or two that you need to know

2/28/08 14

For Now

- Assume…
  - You have all the words already in some buffer
  - The input isn't POS tagged
  - We won't worry about morphological analysis
  - All the words are known

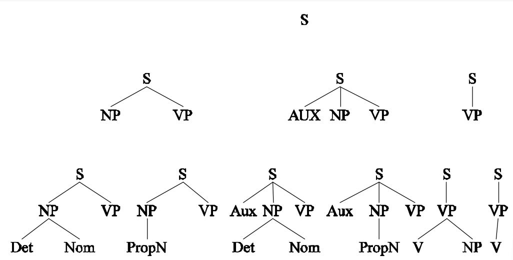
2/28/08 15

Top-Down Parsing

- Since we're trying to find trees rooted with an S (Sentences) start with the rules that give us an S.
- Then work your way down from there to the words.

2/28/08 16

Top Down Space



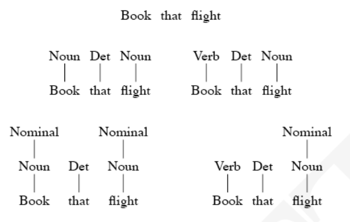
2/28/08 17

Bottom-Up Parsing

- Of course, we also want trees that cover the input words. So start with trees that link up with the words in the right way.
- Then work your way up from there.

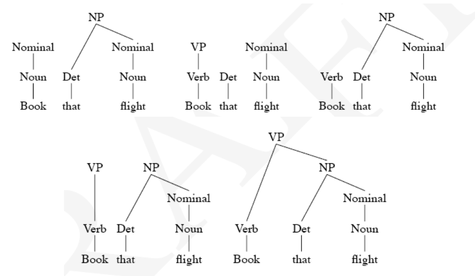
2/28/08 18

Bottom-Up Space



2/28/08 19

Bottom Up Space



2/28/08 20

Control

- Of course, in both cases we left out how to keep track of the search space and how to make choices
  - Which node to try to expand next
  - Which grammar rule to use to expand a node

2/28/08 21

Top-Down and Bottom-Up

- Top-down
  - Only searches for trees that can be answers (i.e. S's)
  - But also suggests trees that are not consistent with any of the words
- Bottom-up
  - Only forms trees consistent with the words
  - But suggest trees that make no sense globally

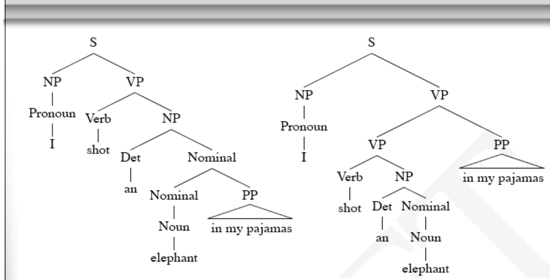
2/28/08 22

Problems

- Even with the best filtering, backtracking methods are doomed if they don't address certain problems
  - Ambiguity
  - Shared subproblems

2/28/08 23

Ambiguity



2/28/08 24

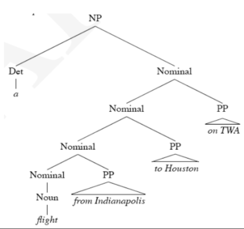
Shared Sub-Problems

- No matter what kind of search (top-down or bottom-up or mixed) that we choose.
  - We don't want to unnecessarily redo work we've already done.

2/28/08 25

Shared Sub-Problems

- Consider
  - A flight from Indianapolis to Houston on TWA



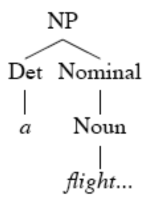
2/28/08 26

Shared Sub-Problems

- Assume a top-down parse making bad initial choices on the Nominal rule.
- In particular…
  - Nominal -> Nominal Noun
  - Nominal -> Nominal PP

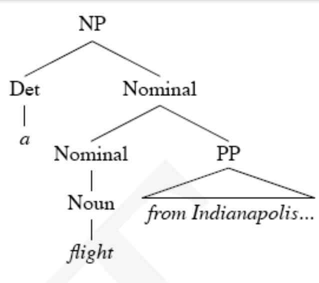
2/28/08 27

Shared Sub-Problems



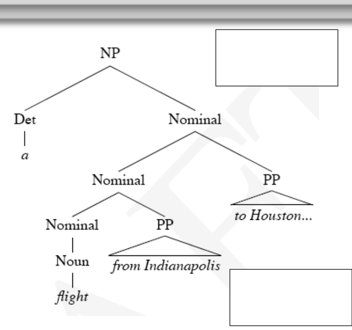
2/28/08 28

Shared Sub-Problems



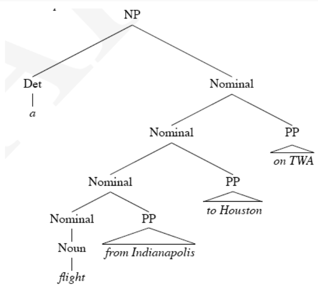
2/28/08 29

Shared Sub-Problems



2/28/08 30

Shared Sub-Problems



2/28/08 31

Break

- Next quiz will be pushed back...
- Readings for this section will be from
  - Chapters 12, 13, 14

2/28/08 32

Parsing

- We're going to cover from Chapter 13
  - CKY (today)
  - Earley (Thursday)
- Both are dynamic programming solutions that run in O(n**3) time.
  - CKY is bottom-up
  - Earley is top-down

2/28/08 33

Sample Grammar

$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid TWA$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal\ Noun$	
$Nominal \rightarrow Nominal\ PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	
$VP \rightarrow Verb\ NP\ PP$	
$VP \rightarrow Verb\ PP$	
$VP \rightarrow VP\ PP$	
$PP \rightarrow Preposition\ NP$	

2/28 34

Dynamic Programming

- DP methods fill tables with partial results and
  - Do not do too much avoidable repeated work
  - Solve exponential problems in polynomial time (sort of)
  - Efficiently store ambiguous structures with shared sub-parts.

2/28/08 35

CKY Parsing

- First we'll limit our grammar to epsilon-free, binary rules (more later)
- Consider the rule A -> BC
  - If there is an A in the input then there must be a B followed by a C in the input.
  - If the A spans from i to j in the input then there must be some k st. i<k<j
    - Ie. The B splits from the C someplace.

2/28/08 36

CKY

- So let's 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]
- 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

2/28/08 37

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

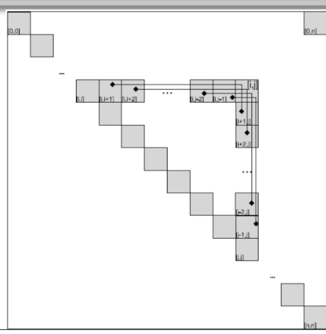
2/28/08 38

CKY

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

2/28/08 39

CKY Table



2/28/08 40

CKY Algorithm

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

  for j ← from 1 to LENGTH(words) do
      table[j − 1, j] ← {A | A → words[j] ∈ grammar }
      for i ← from j − 2 downto 0 do
          for k ← i + 1 to j − 1 do
              table[i,j] ← table[i,j] ∪
                          {A | A → BC ∈ grammar,
                               B ∈ table[i,k],
                               C ∈ table[k, j] }
```

2/28/08 41

CKY Parsing

- Is that really a parser?

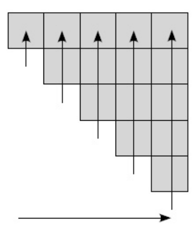
2/28/08 42

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)

Example

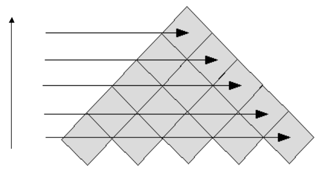
Book	the	flight	through	Houston
S,VP,Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	S, VP [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper-Noun [4,5]



Other Ways to Do It?

- Are there any other sensible ways to fill the table that still guarantee that the cells we need are already filled?

Other Ways to Do It?



2/28/08 46

Sample Grammar

S → *NP VP*	*Det* → *that* | *this* | *a*
S → *Aux NP VP*	*Noun* → *book* | *flight* | *meal* | *money*
S → *VP*	*Verb* → *book* | *include* | *prefer*
NP → *Pronoun*	*Pronoun* → *I* | *she* | *me*
NP → *Proper-Noun*	*Proper-Noun* → *Houston* | *TWA*
NP → *Det Nominal*	*Aux* → *does*
Nominal → *Noun*	*Preposition* → *from* | *to* | *on* | *near* | *through*
Nominal → *Nominal Noun*	
Nominal → *Nominal PP*	
VP → *Verb*	
VP → *Verb NP*	
VP → *Verb NP PP*	
VP → *Verb PP*	
VP → *VP PP*	
PP → *Preposition NP*	

2/28/08 47

Problem

- What if your grammar isn't binary?
  - As in the case of the TreeBank grammar?
- Convert it to binary… any arbitrary CFG can be rewritten into Chomsky-Normal Form automatically.
- What does this mean?
  - The resulting grammar accepts (and rejects) the same set of strings as the original grammar.
  - But the resulting derivations (trees) are different.

2/28/08 48

Problem

- More specifically, rules have to be of the form

  A -> B C

  Or

  A -> *w*

  That is rules can expand to either 2 non-terminals or to a single terminal.

2/28/08 49

Binarization Intuition

- Eliminate chains of unit productions.
- Introduce new intermediate non-terminals into the grammar that distribute rules with length > 2 over several rules. So…

  S -> A B C
  - Turns into

  S -> X C

  X - A B

  Where X is a symbol that doesn't occur anywhere else in the the grammar.

2/28/08 50

CNF Conversion

S → *NP VP*	*S* → *NP VP*
S → *Aux NP VP*	*S* → *X1 VP*
	X1 → *Aux NP*
S → *VP*	*S* → *book* | *include* | *prefer*
	S → *Verb NP*
	S → *X2 PP*
	S → *Verb PP*
	S → *VP PP*
NP → *Pronoun*	*NP* → *I* | *she* | *me*
NP → *Proper-Noun*	*NP* → *TWA* | *Houston*
NP → *Det Nominal*	*NP* → *Det Nominal*
Nominal → *Noun*	*Nominal* → *book* | *flight* | *meal* | *money*
Nominal → *Nominal Noun*	*Nominal* → *Nominal Noun*
Nominal → *Nominal PP*	*Nominal* → *Nominal PP*
VP → *Verb*	*VP* → *book* | *include* | *prefer*
VP → *Verb NP*	*VP* → *Verb NP*
VP → *Verb NP PP*	*VP* → *X2 PP*
	X2 → *Verb NP*
VP → *Verb PP*	*VP* → *Verb PP*
VP → *VP PP*	*VP* → *VP PP*
PP → *Preposition NP*	*PP* → *Preposition NP*

2/28/08 51

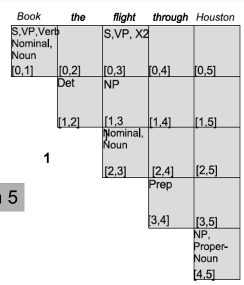
CKY Algorithm

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

  for j ← from 1 to LENGTH(words) do
     table[j − 1, j] ← {A | A → words[j] ∈ grammar }
     for i ← from j − 2 downto 0 do
        for k ← i + 1 to j − 1 do
           table[i,j] ← table[i,j] ∪
                 {A | A → BC ∈ grammar,
                    B ∈ table[i,k],
                    C ∈ table[k, j] }
```

2/28/08 52

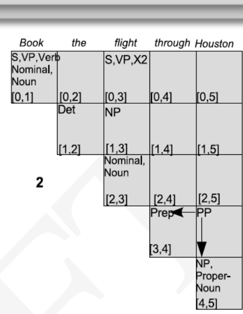
Example



Filling column 5

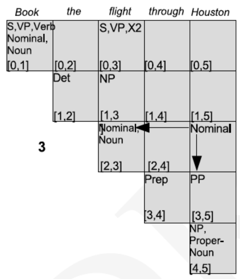
2/28/08 53

Example



2/28/08 54

Example

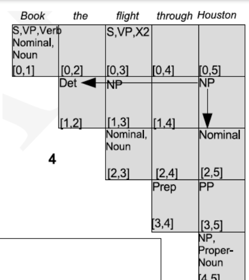


3

2/28/08

55

Example

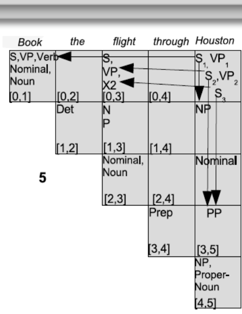


4

2/28/08

56

Example



5

2/28/08

57