Week 5:

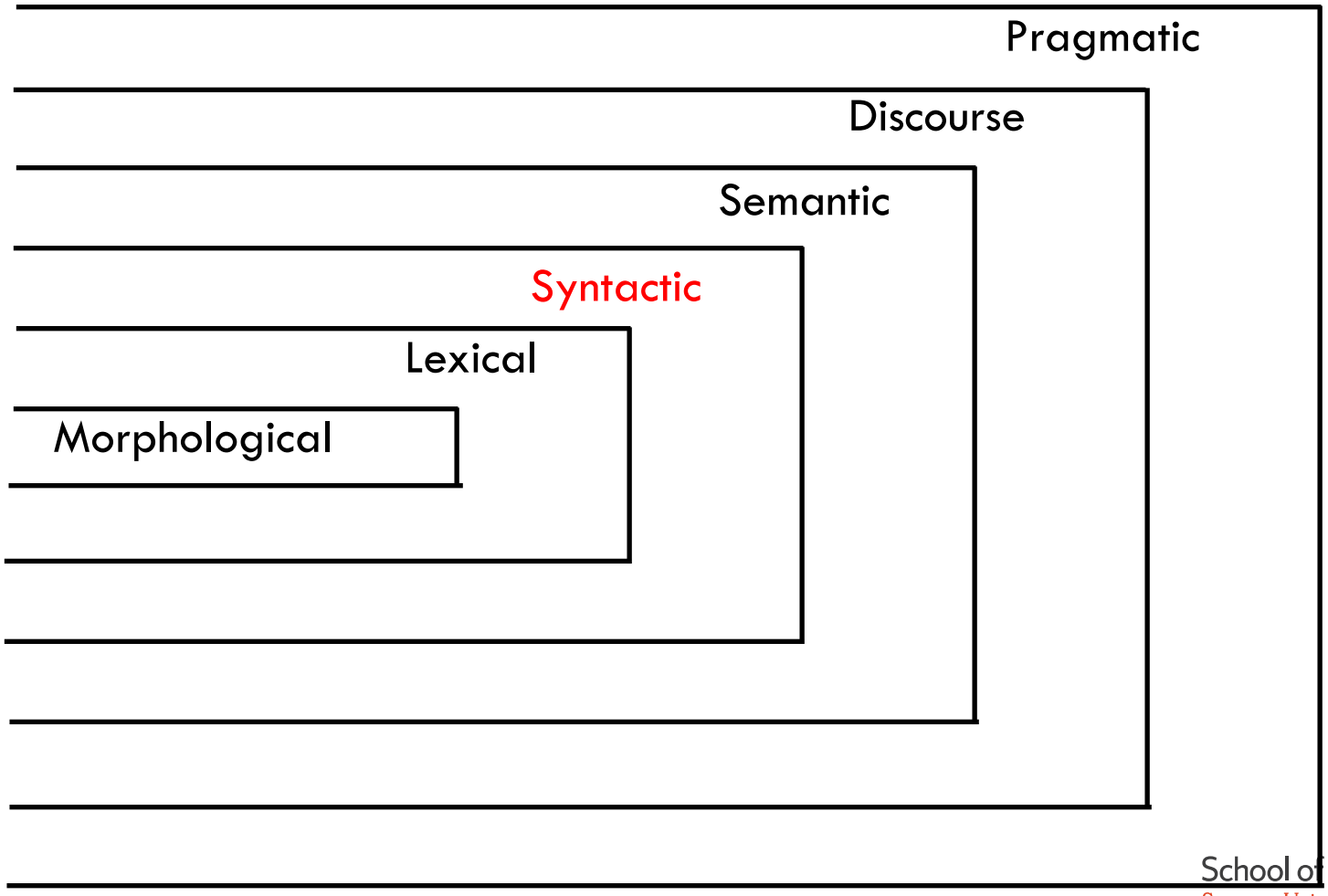
Language Analysis Foundations III

School of Information Studies
Syracuse University

Overview

- Reminder:
  - individual assignment #2 is available now (Due on 10/24)
  - Final group project: first step- Proposal (Due on 10/31)
- Mini-Talks
- Lecture: Context-Free Grammar, Parsing
- Lab

Levels of Language



Intro to Context-Free Grammars (CFG)

School of Information Studies
Syracuse University

Syntactic Analysis

- To analyze how words are put together to make sentences

- The kind of implicit knowledge of your native language that you had mastered by the time you were 3 or 4 years old without explicit instruction

  - Do these word sequences fit together?
    - I saw you yesterday
    - you yesterday I saw year

Why should we care?

- Grammars (and parsing) are key components in many applications:
  - Grammar checkers
  - Dialogue management
  - Question answering
  - Information extraction
  - Machine translation

Context-Free Grammars

- **Definition:** a set of recursive rewriting rules (or productions) used to generate patterns of strings.

- CFGs describe the structure of language by capturing **constituency and ordering**
  - **Constituency:** How do words group into units and what we say about how the various kinds of units behave
  - **Ordering:** Rules that govern the ordering of words and bigger units in the language

Constituents in CFGs

- A constituent is a sequence of words that behave as a unit, sometimes thought of as the "phrases" of the language
  - Common ways to see if things are a unit:
    - *Movement test*: some constituents can be reordered
      - John talked [to the children] [about drugs].
      - John talked [about drugs] [to the children].
      - But not: John talked drugs to the children about (random reorder)
    - *Replacement test*: constituents can be expanded or substituted for:
      - I sat [on the box / right on top of the box / there]

CFG Format

- A CFG consists of the following components:
  - **Non-terminal** symbols: symbols that represent the phrases, the categories of phrases, or the constituents (words grouped into units)
    - NP, VP, etc. representing the constituents or categories of phrases
  - **Terminals** symbols are the words
    - E.g.: car, man, house. These usually come from words in a lexicon
  - **Rewrite rules / productions –** rules for replacing nonterminal symbols (on the left side) in a string with other nonterminal or terminal symbols (on the right side)
  - **Start symbol:** a special nonterminal symbol that appears in the initial string generated by the grammar
  - S → [NP VP] | VP

    (note use of | symbol to give alternate set of rules)

CFG Derivation

- Shows how a particular sentence could be generated by the rules of the grammar
- If sentence is structurally ambiguous, more than one possible derivation is produced

Example for CFG Derivation

- Given the grammar and a sentence, show how grammar rules can give (one or more) derivations.

The sentence:

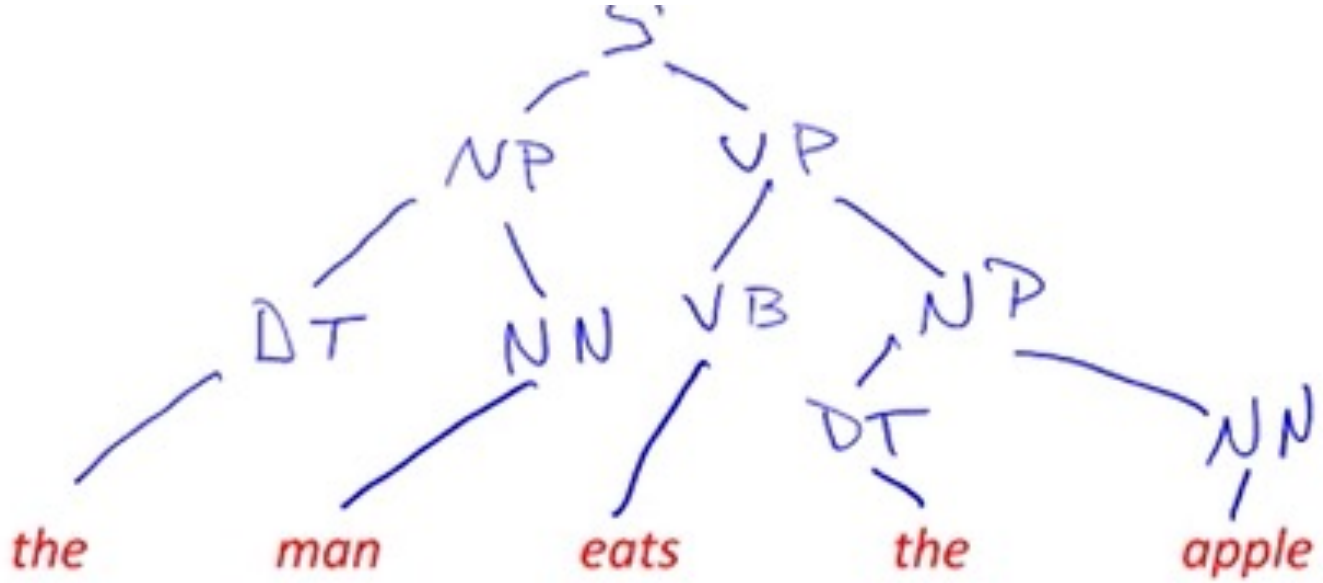
the *man* *eats* *the* *apple*

Context Free Grammar Rules (for this example):

S → NP VP
NP → DT NN
VP → VB NP
VP → VB

DT → *the* | ...
NN → *man* | apple | … (add words)
VB → *eats* | ...

Bottom UP or Top-Down Derivation



Context Free Grammar Rules (for this example):

S → NP VP
NP → DT NN
VP → VB NP
VP → VB

DT → *the* | ...
NN → *man* | apple | … (add words)
VB → *eats* | ...

CFG Phrases for English

School of Information Studies
Syracuse University

Key Constituents for English

English has **headed phrase** structure

- in natural languages, phrases are headed by particular kinds of words with modifiers and qualifiers around them

- Verb Phrases VP → … VB* …
  - You ***may have played*** this game before.
- Noun Phrases NP → … NN* …
  - I like ***the red hat***.
- Adjective Phrases ADJP → … JJ* …
  - Today is ***pretty cold***.
- Adverb Phrases ADVP → … RB* …
  - ***Luckily for us****, the cost was not so high.*

Sentences

- Types of Sentences
  - Declaratives: A plane left

    S -> NP VP
  - Imperatives: Leave!

    S -> VP
  - Yes-No Questions: Did the plane leave?

    S -> Aux NP VP
  - WH (Who,What,When,Why) Questions: When did the plane leave?

    S -> WH Aux NP VP

- More general structure for sentences and clauses:

  SBAR $\rightarrow$ S | SINV | SQ …
  - Sentences, inverted sentences, direct questions, … can also appear in larger clause structure SBAR where sentence is preceded by that
    - e.g. I know that at no time will Jessica say that.

Recursive Rules

- One type of noun phrase is a *noun phrase* followed by a *prepositional phrase*, but a *prepositional phrase* includes a *noun phrase*

  * NP -> NP PP
    PP -> Prep NP

- Of course, this is what makes syntax interesting
  - flights from Denver
  - flights from Denver to Miami
  - flights from Denver to Miami in February
  - flights from Denver to Miami in February on a Friday
  - flights from Denver to Miami in February on a Friday under $300
  - flights from Denver to Miami in February on a Friday under $300 with lunch

Problems for CFGs

School of Information Studies
Syracuse University

Challenges for CFG

- Context-Free Grammars can represent many parts of natural language adequately

- Here are some of the problems that are difficult to represent in a CFG:
  - Agreement
  - Subcategorization
  - Movement

Agreement

This dog	*This dogs
Those dogs	*Those dog
This dog eats	*This dog eat
Those dogs eat	*Those dogs eats

- In English,
  - subjects and verbs have to agree in person and number
  - Determiners and nouns have to agree in number

Subcategorization

- Subcategorization expresses the constraints that a particular verb (predicate) places on the number and syntactic types of arguments it wants to take (occur with).

  - Sneeze: *John sneezed*
  - Find: *Please find* [a flight to NY]NP
  - Give: *Give* [me]NP[a cheaper fare]NP
  - Prefer: I *prefer* [to leave earlier]TO-VP
  - Told: *I was told* [United has a flight]S

Should these be correct?

- John sneezed the book
- I prefer United has a flight
- Give with a flight

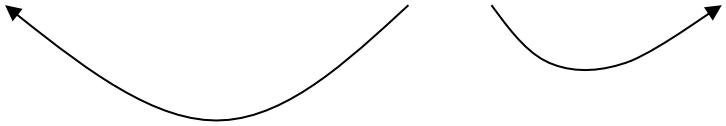
The various rules for VPs **overgenerate**.

- They permit the presence of strings containing verbs and arguments that don't go together

Movement

Consider the verb “booked” in the following example:

- [[My travel agent]NP [booked [the flight]NP]VP]S



i.e. “book” is a straightforward transitive verb. It expects a single NP arg within the VP as an argument, and a single NP arg as the subject.

Movement

But what about?

- Which flight do you want me to have the travel agent **book**?

The direct object argument to “book” isn’t appearing right after the verb. It is in fact a long way from where it’s supposed to appear.

And note that it’s separated from its verb by 2 other verbs.

The Point about CFGs

- CFGs capture basic hierarchical structures of language.
- But there are problems: agreement, subcategorization, and movement
- There are simpler, more elegant solutions that take us out of the CFG framework

Dependency Grammars

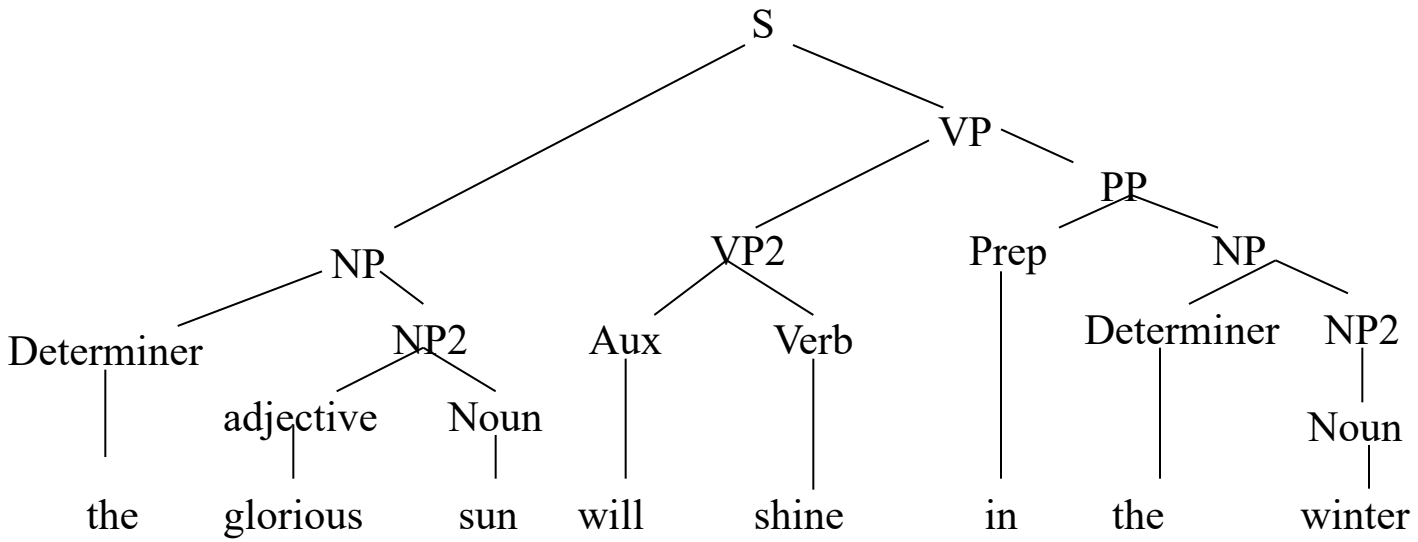
School of Information Studies
Syracuse University

Dependency Grammars

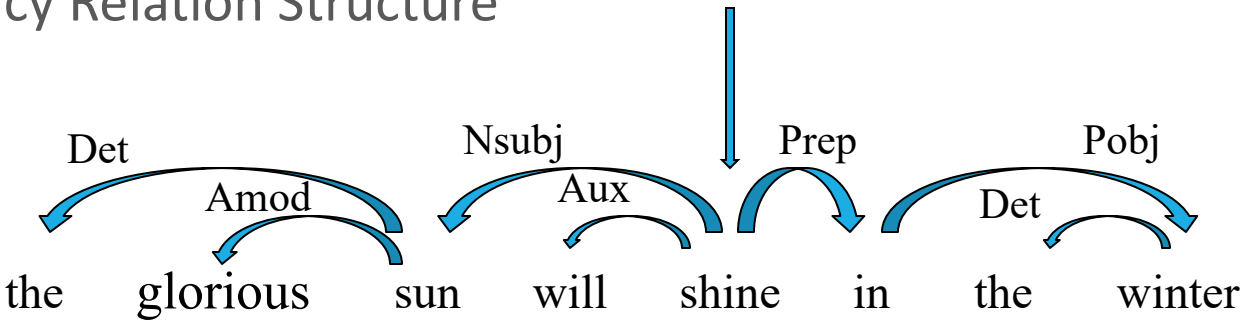
- Dependency grammars offer a different way to represent syntactic structure
  - CFGs represent constituents in a parse tree that can derive the words of a sentence
  - Dependency grammars represent syntactic dependency relations between words that show the syntactic structure
- Syntactic structure is the set of relations between a word (aka the head word) and its dependents.

Examples

Context Free Grammar Tree Structure



Dependency Relation Structure



Note that the head word of a sentence is the verb.

Dependency Relations

The set of Grammar Relations has varied in number

- 48 in the Stanford dependency parser
- 59 in Minipar, a dependency parser from Dekang Lin
- 106 in Link, a related link grammar parser from CMU

Argument Dependencies	Description
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition
Modifier Dependencies	**Description**
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier

Introduction to Parsing

School of Information Studies
Syracuse University

Parsing algorithms defined:

- The process of finding a derivation (i. e. sequence of productions) leading from the START symbol to the TERMINAL symbols
  - Shows how a particular sentence could be generated by the rules of the grammar

- If sentence is more than one possible parse (structurally ambiguous), more than one possible derivation is produced

- Parsing algorithms give a strategy for finding a derivation by making choices among the derivation rules and deciding when the derivation is complete or not.

Top-down Parser

- Goal-driven
- At each stage, the parser looks at goal of a non-terminal symbol (starting with S) and then sees which rules can be applied
- Typically progresses from top-to-bottom, left-to-right

Bottom-up Parser

- Data-driven
- Looks at words in input string first, checks / assigns their category(ies), and tries to combine them into acceptable structures in the grammar
- Involves scanning the derivation so far for sub-strings which match the right-hand-side of grammar / production rules and using the rule that would show their derivation from the non-terminal symbol of that rule

Example: Bottom UP or Top-Down Derivation

The man eats the apple.

Context Free Grammar Rules (for this example):

S → NP VP	DT → *the* | ...
NP → DT NN	NN → *man* | apple | … (add words)
VP → VB NP	VB → *eats* | ...
VP → VB	

In-class activity: Bottom UP or Top-Down Derivation

John saw the man in the park.

Context Free Grammar Rules (for this example):

S -> NP VP
VP -> V NP | V NP PP
PP -> P NP
V -> "saw"
NP -> Prop | Det N | Det N PP

Prop -> "John"
Det -> "a" | "the" |
N -> "man" | "park"
P -> "in"

Parsing issues

- Top-down
  - Only searches for trees that can be answers
  - 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
- Ambiguity:
  - Structural ambiguity will result in more than one possible derivation
- Performance of backtracking is exponential in time:
  - Backtracking may result in smaller sub-trees being parsed many times

Parsing Algorithms

School of Information Studies
Syracuse University

Solutions to parsing problems

Modern parsing algorithms solve 3 problems :

1. Solve the problem of performance with chart parsers

2. Solve the problems of pre-defining CFG or other grammars by using Treebanks and statistical parsing

3. Partially solve the problems of correctly choosing the best parse trees by using lexicalization (information about words from the Treebank)

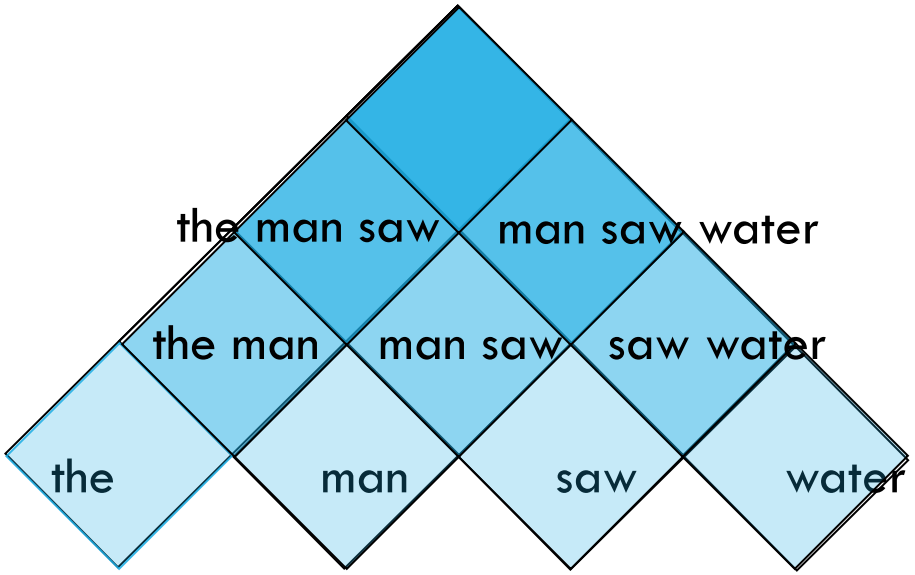
1. Chart Parsers

- CKY (Cocke-Kasami-Younger) algorithm is an example
  - Bottom-up parser (but can also have top-down chart parsers)
  - Requires grammar to be in Chomsky Normal Form
    - CNF: only allow two types of **right-hand sides**:
      - Two non-terminal symbols: NP -> DT NN
      - One terminal symbol: VP→ look
  - Fills in a data structure called a chart or a parse triangle

CKY (Chart) Parsing

- For input of length n, fills a parse table triangle of size (n, n) , where each element has the non-terminal production representing the span of text from position i to j.
  - Cells in first (bottom) layer describe trees of single words
  - Cells in second layer describes how rewrite rules can be used to combine trees in first layer for trees with two words
  - Etc.
- No back-tracking

Each cell has rules for:



2. Need for Treebanks

- Before you can parse you need a grammar.
- So where do grammars come from?
  - Grammar Engineering
    - Hand-crafted decades-long efforts by humans to write grammars
  - TreeBanks
    - Semi-automatically generated sets of parse trees for the sentences in some corpus (manually corrected by human annotators).

Penn Treebank example: manually annotated sentence

```
( (S
    (NP-SBJ (DT The) (NN move))
    (VP (VBD followed)
      (NP
        (NP (DT a) (NN round))
        (PP (IN of)
          (NP
            (NP (JJ similar) (NNS increases))
            (PP (IN by)
              (NP (JJ other) (NNS lenders)))
            (PP (IN against)
              (NP (NNP Arizona) (JJ real) (NN estate) (NNS loans))))))
      (, ,)
      (S-ADV
        (NP-SBJ (-NONE- *))
        (VP (VBG reflecting)
          (NP
            (NP (DT a) (VBG continuing) (NN decline))
            (PP-LOC (IN in)
              (NP (DT that) (NN market)))))))
    (. .)))
```

[Marcus et al. 1993, *Computational Linguistics*]

TreeBank Grammars

- We could read off the grammar from the Treebank
  - The grammar is the set of rules (local subtrees) that occur in the annotated corpus
  - Penn TreeBank has about 17500 grammar rules under this definition.

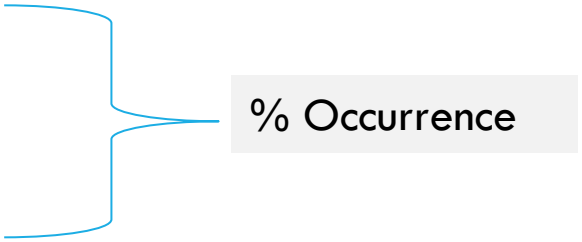
- The main use of the Treebank is to provide the probabilities to inform the statistical parsers, and the grammar does not actually have to be generated.
  - **If we have a Treebank, we don't need to write down a grammar**

Probabilistic Context-Free Grammars

- By way of introduction to statistical parsers, we first introduce the idea of associating probabilities with grammar rewrite rules.

  - Attach probabilities to grammar rules
  - The expansions for a given non-terminal sum to 1

VP -> Verb	.55
VP -> Verb NP	.40
VP -> Verb NP PP	.05



Getting the probabilities

- From a treebank of annotated data, get the probabilities that any non-terminal symbol is rewritten with a particular rule
  - E.g., to get the probability for a particular VP rule just count all the times the rule is used and divide by the number of VPs overall.

- The parsing task is to generate the parse tree with the highest probability (or the top n parse trees)

- For a PCFG parser, the probability of a parse tree is the product of the probabilities of the rules used in the derivation

$$P(T,S) = \prod_{node \in T} P(rule(n))$$

Typical Approach to PCFG parser

- Use CKY as the backbone of the algorithm
- Assign probabilities to constituents as they are completed and placed in the table
- Use the max probability for each constituent going up

Problems with PCFG Parsing

- This typical approach always just picks the most likely rule in the derivation

- The probability model we're using is only based on the rules in the derivation…
  - Doesn't use the words in any real way
  - Most probable parse is not usually the right one (the one in the treebank test set).

3. Lexicalized Statistical Parsing

- Add lexical dependencies to the scheme of probabilities
  - Integrate the preferences of particular words into the probabilities in the derivation
  - i.e. Condition the rule probabilities on the actual words
- To do that we're going to make use of the notion of the head of a phrase
  - The head of an NP is its noun
  - The head of a VP is its verb
  - The head of a PP is its preposition
  - (It's really more complicated than that but this will do.)
- Expand the set of phrase types with phrase type/word
  - In practice, we learn probabilities to automatically detect head words

Incorporating Semantics into the process

Last Points

- Statistical parsers are getting quite good, but it's still quite challenging to expect them to come up with the correct parse given only statistics from syntactic and lexical information.

- But if our statistical parser comes up with the top-N parses, then it is quite likely that the correct parse is among them.

- Lots of current work on
  - Re-ranking to make the top-N list even better.

Parsing: Evaluation

School of Information Studies
Syracuse University

Evaluation

- Given that it is difficult/ambiguous to produce the entire correct tree, look at how much of content of the trees are correctly produced
  - Evaluation measures based on the correct number of constituents (or sub-trees) in the system compared to the reference (gold standard)
- Precision
  - What fraction of the sub-trees in our parse matched corresponding sub-trees in the reference answer
    - How much of what we're producing is right?
- Recall
  - What fraction of the sub-trees in the reference answer did we actually get?
    - How much of what we should have gotten did we get?

F-measure combines precision and recall to give an overall score.

Performance of Parsers

- The Charniak series of parsers was developed by Eugene Charniak and his group; it produces N-best parse trees.
  - Its evaluation is on the Penn Treebank at about 92% F measure.
- Another top performing parser, originally by Dan Klein and Christopher Manning, is available from the Stanford NLP group
  - combines "separate PCFG phrase structure and lexical dependency experts".
  - Demo at: http://nlp.stanford.edu:8080/parser/

Lab

School of Information Studies
Syracuse University

Tasks

1. Create CFG grammar and its parser
   - Test on different sentences
2. Create PCFG grammar and its parser
   - Test on different sentences
3. Create Dependency grammar and its parser
   - Test on different sentences