Parsing:
Top-Down vs. Bottom-Up Parsing Algorithms
Partial Parsing – Chunking
Treebanks
Statistical Parsing
Dependency Parsing

Parsing defined:

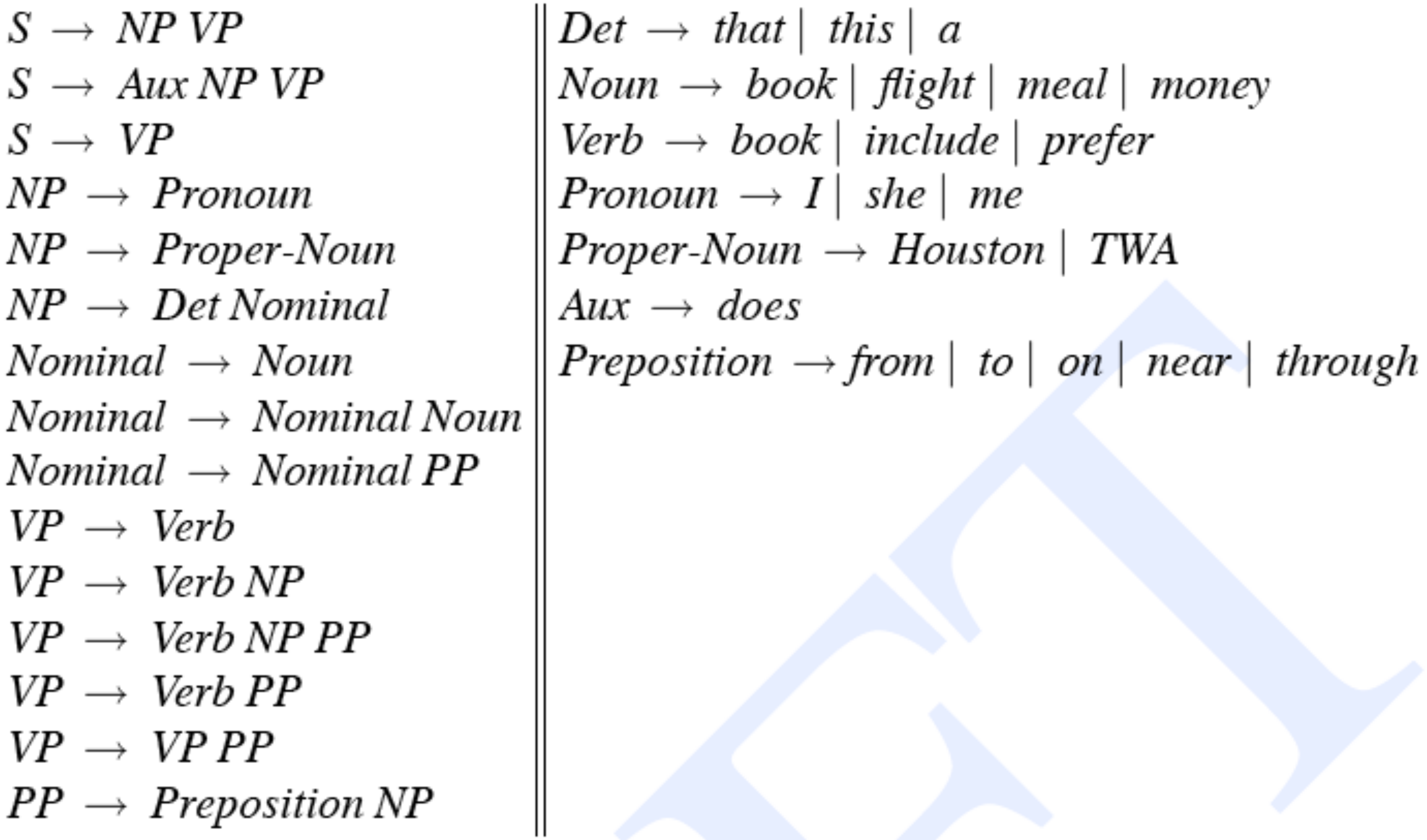
- The process of finding a derivation (i. e. sequence of productions) leading from the START symbol to a TERMINAL symbol
  - Shows how a particular sentence ***could be*** generated by the rules of the grammar
- If sentence is structurally ambiguous, more than one possible derivation is produced
- Can solve both the recognition and analysis problems
  - Is this sentence derived from this grammar?
  - Give the derivation(s) that can derive this sentence.
- 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

- Hypothesis-driven
  - At each stage, parser hypothesizes a structure, and tests whether data (next word in sentence) fits the hypothesis
- Looks at goal first (S) and then sees which rules can be applied
  - Typically progresses from top-to-bottom, left-to-right
  - Non-deterministic (can be rewritten in more than one way)
- When rules derive lexical elements (words), check with the input to see if the right sentence is being derived
- An algorithm may include a backtracking mechanism
  - When it is determined that the wrong rule has been used, it backs up and tries another rule

Example Grammar

- The flight grammar from the text:



S → *NP VP*
S → *Aux NP VP*
S → *VP*
NP → *Pronoun*
NP → *Proper-Noun*
NP → *Det Nominal*
Nominal → *Noun*
Nominal → *Nominal Noun*
Nominal → *Nominal PP*
VP → *Verb*
VP → *Verb NP*
VP → *Verb NP PP*
VP → *Verb PP*
VP → *VP PP*
PP → *Preposition NP*

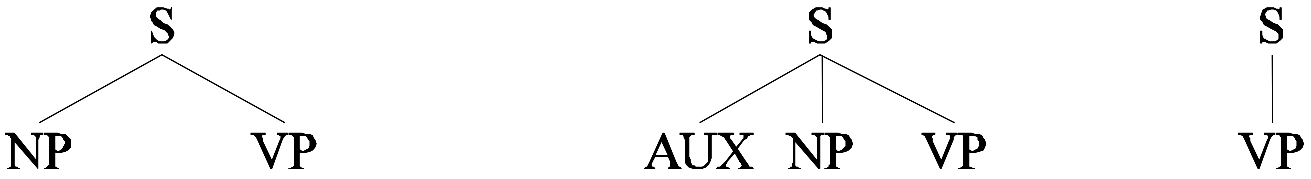
Det → *that* | *this* | *a*
Noun → *book* | *flight* | *meal* | *money*
Verb → *book* | *include* | *prefer*
Pronoun → *I* | *she* | *me*
Proper-Noun → *Houston* | *TWA*
Aux → *does*
Preposition → *from* | *to* | *on* | *near* | *through*

Example Derivation

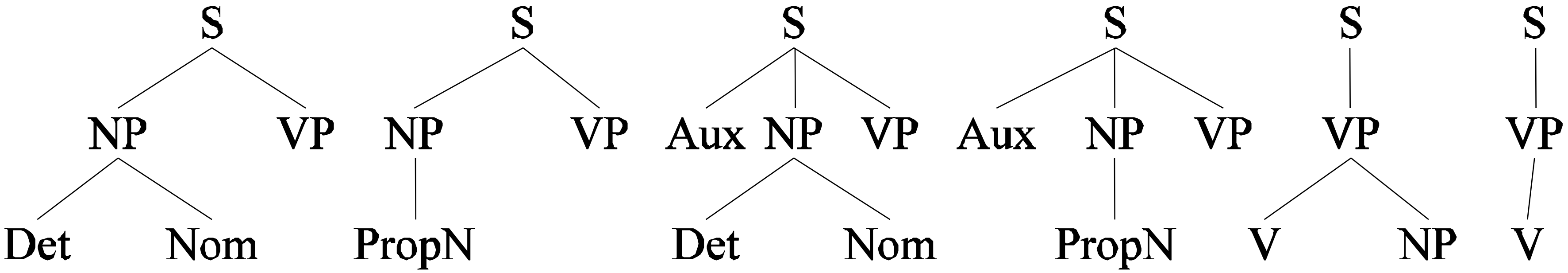
- Derivation for “Book that flight” (from the text)
  - The Start symbol

S

  - Can derive 3 rules as follows:



  - Each non-terminal can derive additional rules



  - Only the last two trees can derive the word “book” as first in the input

Bottom-up Parser

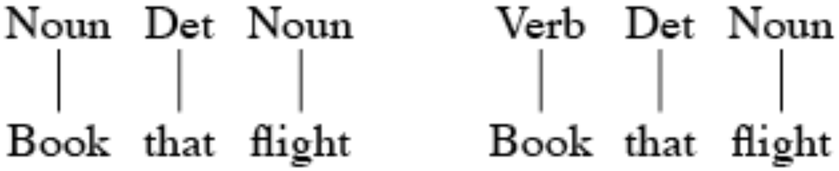
- Data-driven
- Looks at words in input string first, checks / assigns their category(ies), and tries to combine them into acceptable structures in the grammar
- Involves scanning the derivation so far for sub-strings which match the right-hand-side of grammar / production rules and using the rule that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

Bottom-up Derivation

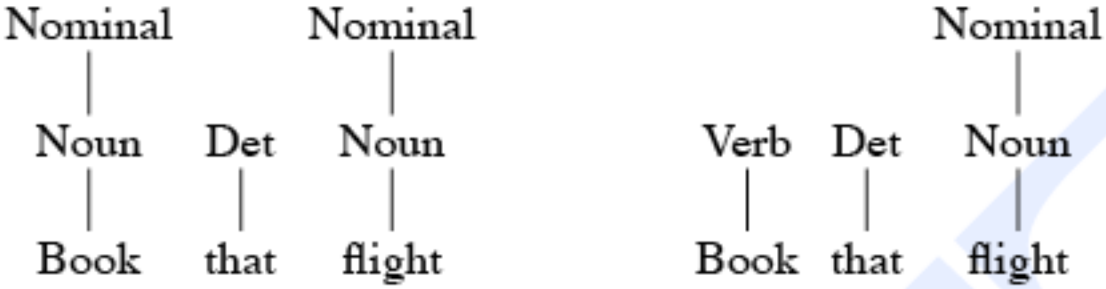
- Starts with input text

Book that flight

- derive the text from rules, in this case, two possible lexical rules

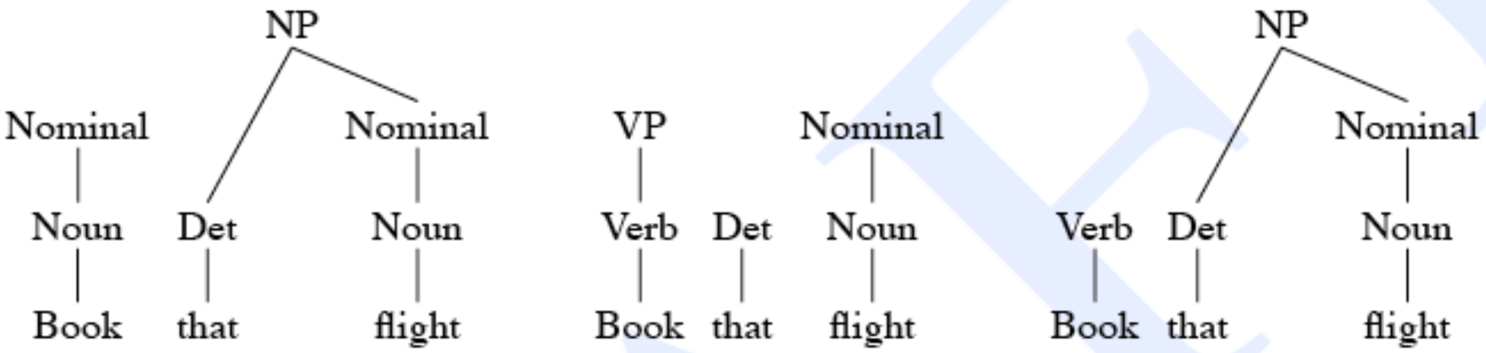


- Each of those can be derived from nonterminals

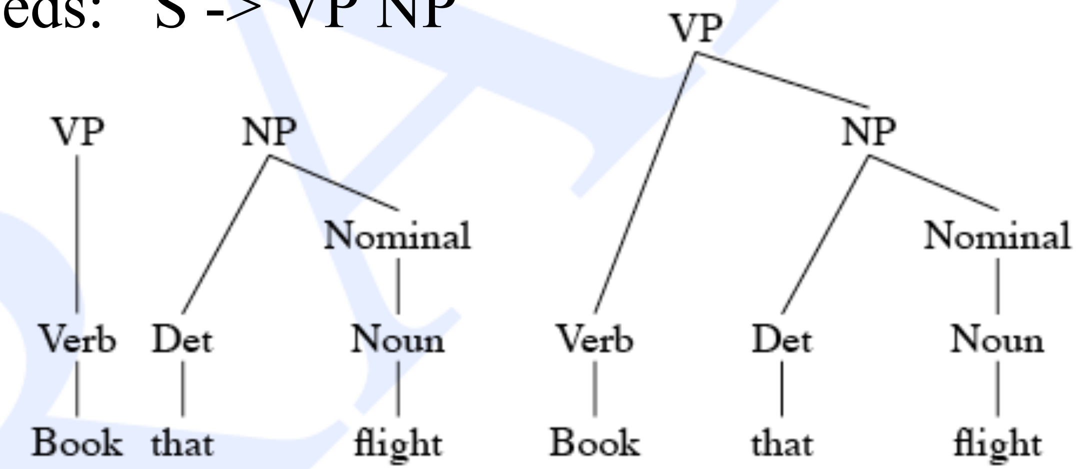


Bottom-Up Derivation

- Only the rightmost tree can continue the derivation here:



- And only one succeeds: S -> VP NP



Bottom-up Parsing

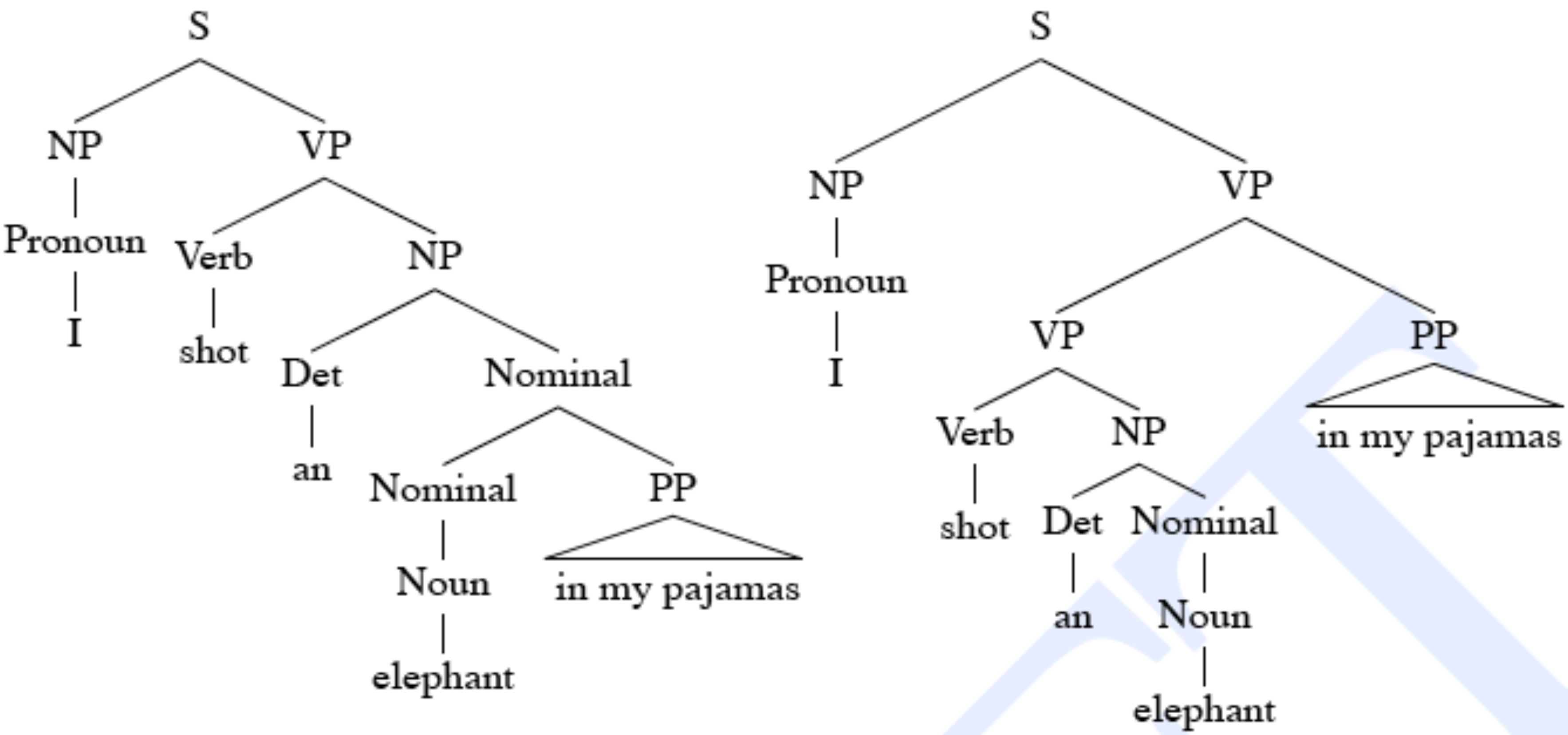
- Algorithm called shift/reduce parsing
  - Scans the input from left to right and keeps a “stack” of the partial parse tree so far
  - The shift operation looks at the next input and shifts it onto the stack
  - The reduce operation looks at N symbols on the stack and if they match the RHS of a grammar rule, reduces the stack by replacing those symbols with the nonterminal
- Also must either incorporate back-tracking or must keep multiple possible parses

Parsing issues

- 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
- Note that in the previous example, there was local ambiguity between “book” being a verb or a noun that was resolved at the end of the parse
- But examples with structural ambiguity will not be resolved, resulting in more than one possible derivation

Structural Ambiguity

- *One morning I shot an elephant in my pajamas. How he got into my pajamas I don't know.* Groucho Marx, *Animal Crackers,* 1930.



Working with Parsing

- NLTK parsing demos (in lab next time)
  - Top-down parsing using a recursive descent algorithm
    - Top down parsing with back-tracking
    - Must not have left-recursion in the grammar rules
  - Bottom-up parsing using a shift-reduce algorithm
    - Instead of back-tracking or multiple parses, this NLTK implementation requires outside intervention to apply the correct rule when there is a choice

Parsing Algorithms

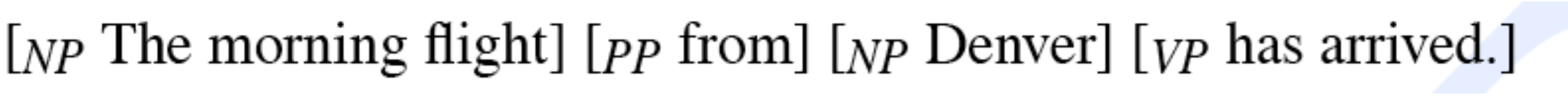
- Avoid back-tracking and re-doing subtrees
  - Recall that the backtracking recursive descent expanded some subtrees multiple times
- Use forms of dynamic programming to search for good parse trees
  - Attempt to perform exponential process in polynomial time
- CKY (Cocke-Kasami-Younger) algorithm
  - Bottom-up parser
  - Requires grammar to be in Chomsky Normal Form, with only two symbols on the righ-hand-side of each production
  - For input of length n, fills a parse table matrix of size (n+1, n+1) , where each element has the non-terminal production representing the span of text from position i to j.

Parsing Algorithms

- Earley algorithm (due to J. Earley, 1970)
  - Top-down parsing
  - Uses a chart of states to represent partial parse trees generated so far
  - For input of length n, scans the input and fills an array of length n+1 with the chart of states representing each item in the input.
- Chart parsing
  - Similar to CKY and Earley algorithms, but more flexible in the order that it searches through the states to fill in charts
- Note that all these techniques use the idea of a table or chart to hold partial parses in order to avoid duplication of effort in finding parses for sub-trees

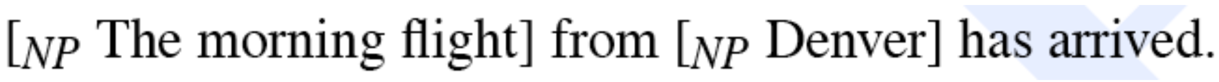
Partial Parsing

- For many applications you don’t really need a full-blown syntactic parse. You just need a good idea of where the base syntactic units are.
  - Often referred to as chunks.
- For example, if you’re interested in locating all the people, places and organizations in a text it might be useful to know where all the base NPs are.
- A full partial parse would have chunks for all the text, but with no hierarchical structure:



[$_{NP}$ The morning flight] [$_{PP}$ from] [$_{NP}$ Denver] [$_{VP}$ has arrived.]

- A partial parse for just base NPs would be:

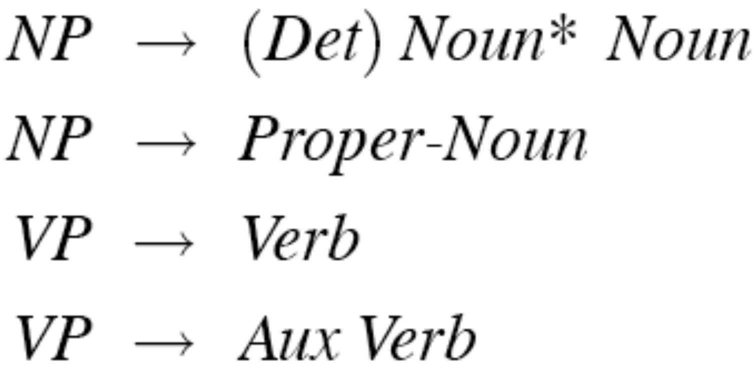


[$_{NP}$ The morning flight] from [$_{NP}$ Denver] has arrived.

Rule-Based Partial Parsing

- Restrict the form of rules to exclude recursion (make the rules flat).
- Group and order the rules so that the RHS of the rules can refer to non-terminals introduced in earlier rules but not later ones.
- Write regular expressions to recognize the right-hand-side of rules, starting from the later ones.
- For complete chunking, typical ordering:
  - Base syntactic phrases
  - Larger verb and noun groups
  - Sentential level rules

Partial Parsing



$$\begin{aligned}
NP &\rightarrow (Det)\ Noun^{*}\ \ Noun \\
NP &\rightarrow Proper\text{-}Noun \\
VP &\rightarrow Verb \\
VP &\rightarrow Aux\ Verb
\end{aligned}$$

- No direct or indirect recursion allowed in these rules.
- That is you can’t directly or indirectly reference the LHS of the rule on the RHS.

Evaluation

- For evaluation, we need a metric that works at the level of the chunks.
- Precision:
  - The fraction of chunks the system returned that were right
    - "Right" means the boundaries and the label are correct given some labeled test set.
- Recall:
  - The fraction of the chunks that system got from those that it should have gotten.
- F measure: Harmonic mean of those two numbers.

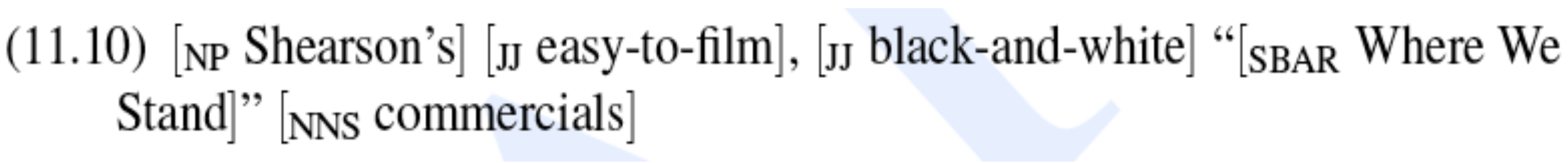
Need for Treebanks

- 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, but representing phrases of text in actual use).

Section on Treebanks and probabilistic parsing from Jim Martin's online slides.

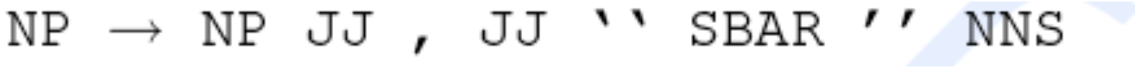
Example

- Given an annotated sentence,



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

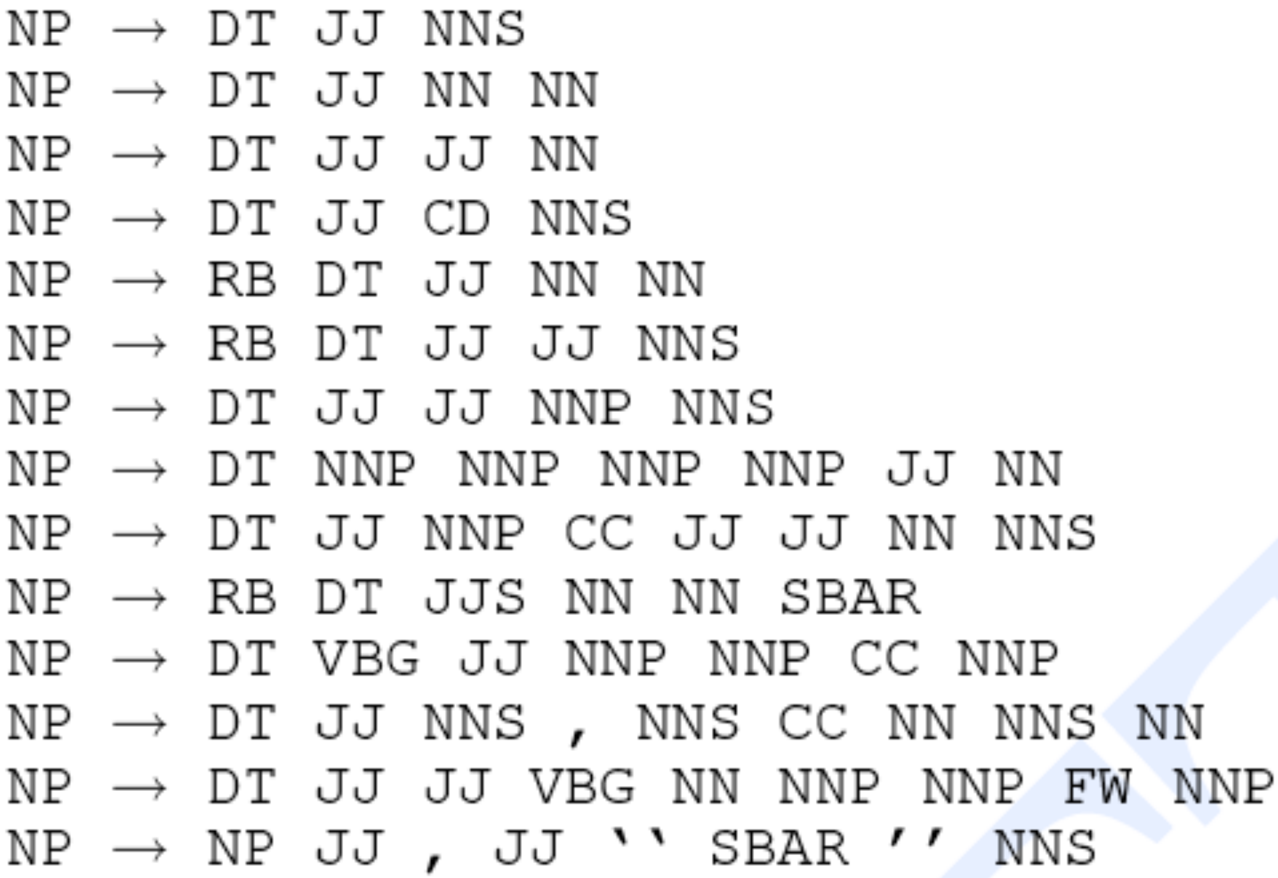
- We can make a grammar rule:



```
NP → NP JJ , JJ `` SBAR '' NNS
```

- And we'll make rules for sub-trees as well

Sample Rules for Noun Phrases



```
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
```

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.
- But 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.
- The grammar hovers behind the Treebank; it is in the minds of the human annotators (and in the annotation manual!)

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 NP	.05

Getting the probabilities

- From a treebank of annotated data, get the probabilities that any non-terminal symbol is rewritten with a particular rule
  - So for example, 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)
- 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

- 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

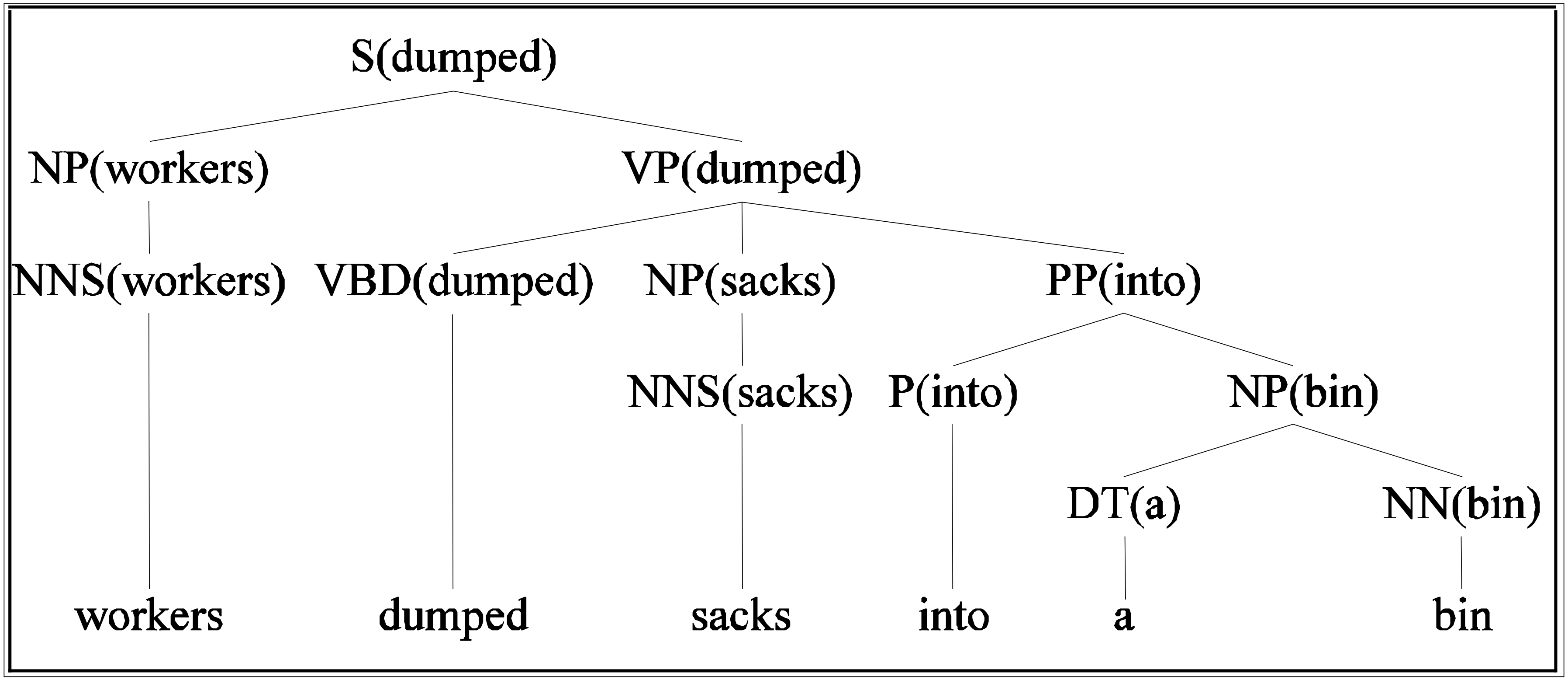
- But this typical approach always just picks the most likely rule in the derivation
  - For example, if it is more likely that a prepositional phrases attaches to the noun phrase that it follows instead of the verb, then the probabilistic parser will always attach prepositional phrases to the closest noun
- The probability model we're using is only based on the rules in the derivation…
  - Doesn't use the words in any real way
  - Doesn't take into account where in the derivation a rule is used
    - E.g. the parent of the non-terminal of the derivation
  - Doesn't really work
    - Most probable parse isn't usually the right one (the one in the treebank test set).

Head Words

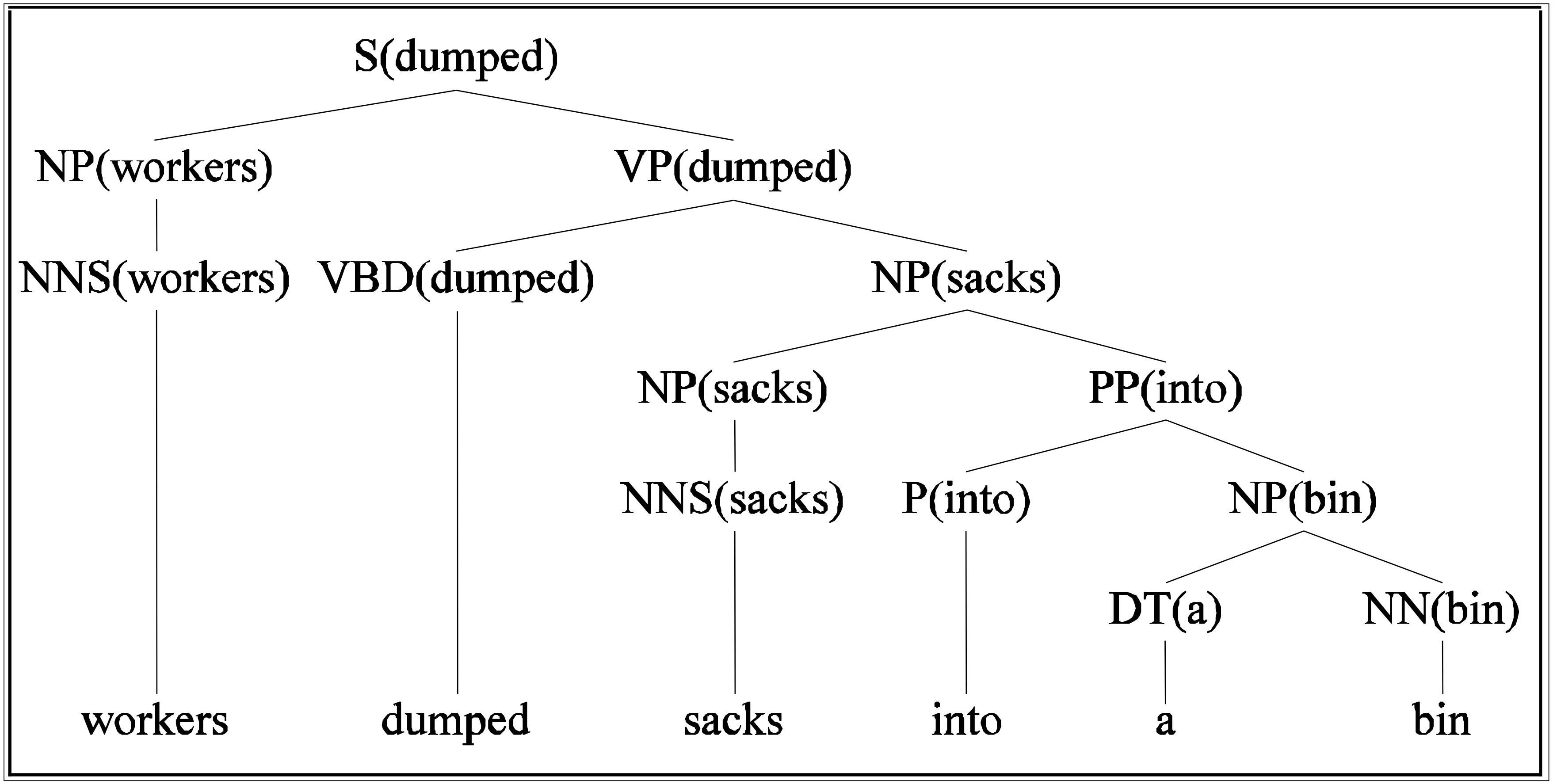
- Add lexical dependencies to the scheme…
  - Integrate the preferences of particular words into the probabilities in the derivation
  - i.e. Condition the rule probabilities on the actual words
- To do that we’re going to make use of the notion of the head of a phrase
  - The head of an NP is its noun
  - 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.)

Example (right)



Example (wrong)



Preferences

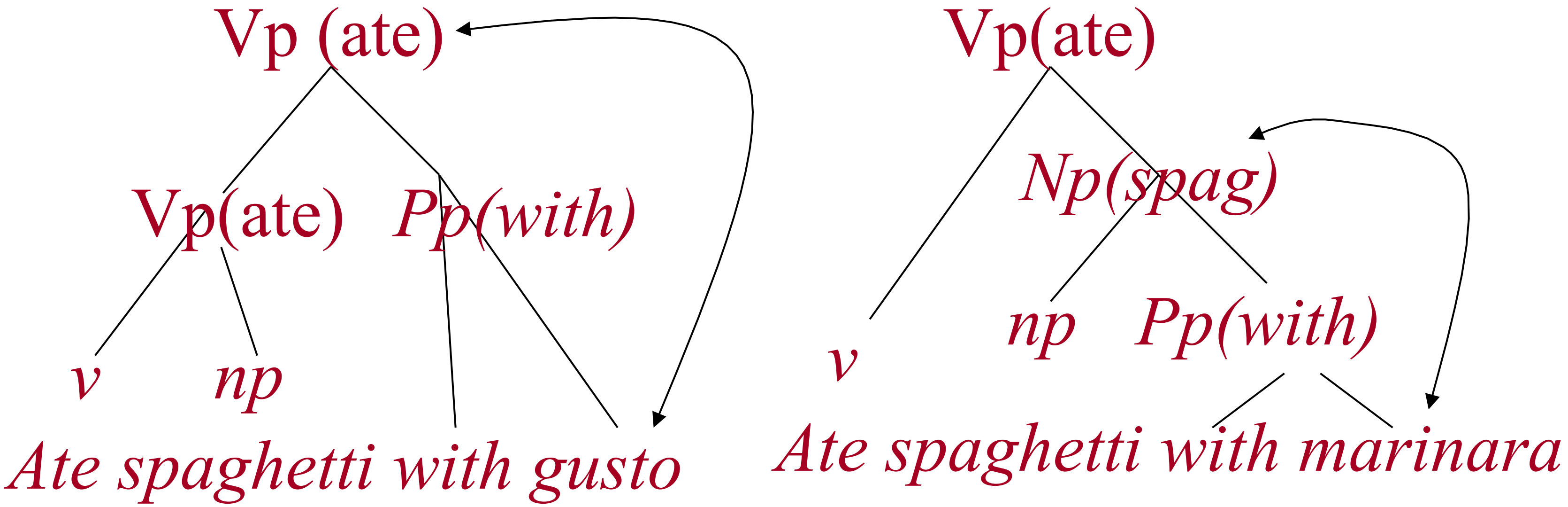
- The issue here is the attachment of the PP. So the affinities we care about are the ones between dumped and into vs. sacks and into.
  - So count the places where dumped is the head of a constituent that has a PP daughter with into as its head and normalize
  - Vs. the situation where sacks is a constituent with into as the head of a PP daughter.
- In general, collect statistics on preferences (aka affinities)
  - Use verb subcategorization
    - Particular verbs have affinities for particular VPs
  - Objects affinities for their predicates (mostly their mothers and grandmothers)
    - Some objects fit better with some predicates than others

Preference example

- Consider the VPs
  - Ate spaghetti with gusto
  - Ate spaghetti with marinara
- The affinity of gusto for eat is much larger than its affinity for spaghetti
- On the other hand, the affinity of marinara for spaghetti is much higher than its affinity for ate

Preference Example (2)

- Note the relationship here is more distant and doesn't involve a headword since gusto and marinara aren't the heads of the PPs.



Note

- Jim Martin: "In case someone hasn't pointed this out yet, this lexicalization stuff is a thinly veiled attempt to incorporate semantics into the syntactic parsing process…
  - Duhh..,. Picking the right parse requires the use of semantics."

Statistical Lexicalized Parser Sketch

- Brief description of Collins parser (Model 1) 1999
- Although the parser operates in a bottom-up fashion, loosely based on CKY (or shift/reduce), the probabilities in the model are generated in a top-down fashion
- Think of the RHS of every CFG grammar rule, derived from a treebank, as a head non-terminal, with other non-terminals that occur to the left and right of the head. So each rule is:

  LHS -> L_n … L_1 H R_1 … R_m
- Given the LHS, we first generate the head of the rule and then generate the dependents of the head, inside-out with both Left and Right. A special Stop symbol is added so that the model can know when to stop generating symbols.
- Given a parse tree in the treebank, each constituent is analyzed as a generating rule from the non-terminal of its head, and contributes to the probabilities of the model.
- The parser later uses these probabilities to decide when to reduce a set of symbols as the RHS of a grammar rule.

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 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.

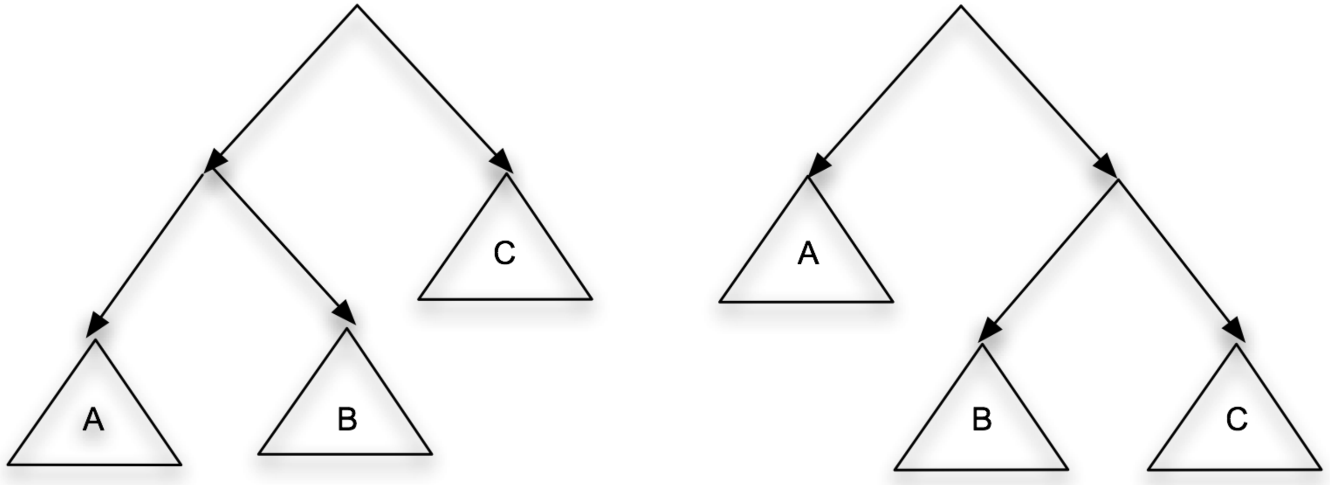
- There are also grammar-driven parsers that are competitive with the statistical parsers, notably the CCG (Combinatory Categorial Grammar) parsers

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?

Evaluation

- An additional evaluation measure that is often reported is that of Crossing Brackets errors, in which the subtrees are equal, but they are put together in a different order.



Parser hypothesis

((A B) C)

Reference answer

(A (B C))

Available Parsers

- Among the family of lexicalized statistical parsers are the well-known Collins parser (Michael Collins 1996, 1999) and the Charniak parser (1997)
  - both are publicly available and widely used in NLP, for non-commercial purposes.
- The Charniak series of parsers is still under development, by Eugene Charniak and his group; it produces N-best parse trees.
  - Its evaluation is (as of last year) currently the best on the Penn Treebank at about 91% F measure.
- Another 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://josie.stanford.edu:8080/parser/

Available Parsers

- The CCG parsers are available from their open source page
  - http://groups.inf.ed.ac.uk/ccg/software.html

- Parsers are also available through the OpenNLP project, with the OpenNLP API:
  - http://opennlp.sourceforge.net/

Dependency Parsing

- Dependency parsing has some resemblance to lexicalized parsing because of the importance of the lexical entities (words) to capturing the syntactic structure
- But dependency parsing produces a simpler representation of the structure.
  - Can be easier to use in some semantic applications

Transition-based parsers

- A typical parser of this type is that of Nivre 2004, which is a bottom-up “span” parser
- Operation of parser:
  - State: stack of partially processed items and a queue of remaining tokens
  - Transitions: add dependency arcs; stack or queue operations
    - Operations are
      - Build left arc
      - Build right arc
      - Shift
      - Reduce

Training the Parser

- How does the parser know which operation, and arc label, to apply?
- It learns these from the annotated corpus.
- For English, dependency grammar relations are derived from Penn Treebank
- Then a collection of examples is extracted from the text to train the parser
  - Each example consists of a set of features representing the state of the parser, including the next word, previous word, POS tags, etc.
  - A machine learning algorithm is applied to learn a classifier, which can assign a parsing operation to every parsing state

Non-Projective Parsing

- The bottom-up span parser is for projective parsing; but there is also a technique to add non-projective relations after the parse
- Alternate technique is due to Ryan McDonald, 2005, which converts the dependency parsing problem to that of finding a maximal spanning tree in the dependency graph.
  - Again, the dependencies in the graph are learned from the annotated corpus of Penn Treebank