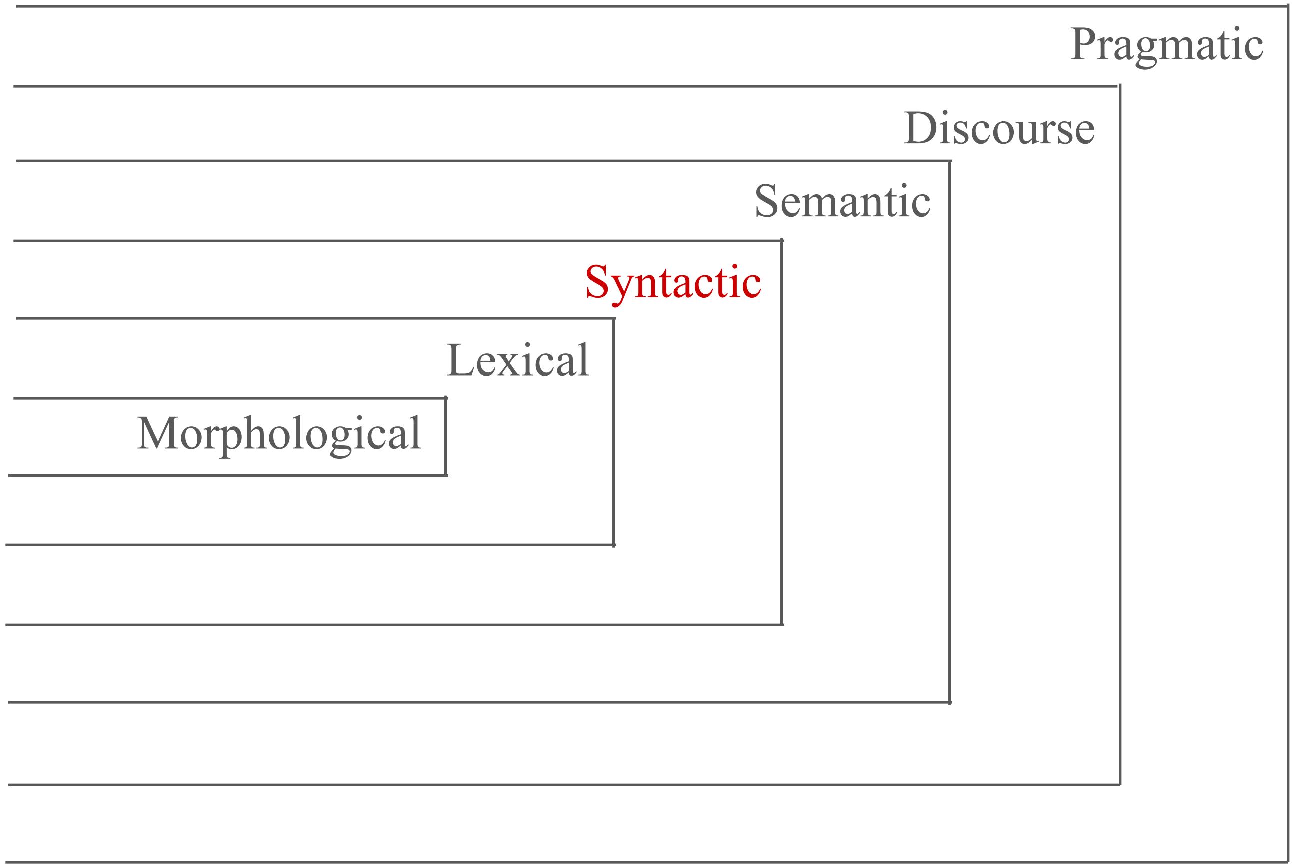
Introduction to Context-Free Grammars (CFGs)

School of Information Studies
Syracuse University

Levels of Language



School of Information Studies
Syracuse University

Syntactic Analysis

Syntax expresses the way in which words are arranged together.

It is 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 year

  Chomsky examples:
  - colorless green ideas sleep furiously
  - furiously sleep ideas green colorless

NLP uses syntax to produce a structural analysis of the input sentence.

Formal Grammar

Rules that embody generalizations that hold for the symbols and combinations of symbols in a language for constructing acceptable sentences

- Grammar is most closely identified with syntax but may contain elements of all levels of language

Theoretical linguists use grammar

- To indicate which are the well-formed sentences, defining that language
- Deep structure: an ideal speaker's internalized ability to create and understand all sentences

Applied uses in NLP

- To assign a structural description to pieces of text
- Typically not consciously modeled after any particular linguistic theory, but based on actual use of language

Context-Free Grammars

This type of formal grammar is the most widely used for natural language

- Later, we will also discuss dependency grammars

CFGs describe the structure of language by capturing **constituency and ordering**

- Ordering is:
  - The rules that govern the ordering of words and bigger units in the language
- Constituency is:
  - How words group into units and what we say about how the various kinds of units behave

Constituents in CFGs

A constituent is a sequence of words that behave as a unit, sometimes thought of as the "phrases" of the language

- Some units 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).
- Constituents can be expanded or substituted for:
  - I sat [on the box/right on top of the box/there].
- Not always obvious when a group of words is a constituent
  - Other properties include:
    - Coordination (can put "and" in-between constituents)
    - Regular internal structure

CFG Formal Definition

Non-Terminal Symbols

S, NP, VP, etc. representing the constituents or categories of phrases

Terminal Symbols

car, man, house, representing words in the lexicon

- The rewrite rules will include lexical rules (e.g., N -> car|man|house)

Rewrite Rules/Productions

S → NP VP | VP

showing possible constituents of the phrase (note the use of the | symbol to give the alternate right-hand side (RHS) of the rules)

A Designated Start Symbol S

A derivation is a sequence of rewrite rules applied to a string that exactly covers the items in that string

Generativity vs. Parsing

You can view these rules as either synthesis or analysis machines.

- Generate strings in the language.
- Impose structures (trees) on strings in the language.

The latter is the analysis task of parsing.

- Parsing is the process of finding a derivation (i.e., sequence of productions) leading from the **start** symbol to a **terminal** symbol (or **terminals** to **start** symbol).
- It shows how a particular sentence could be generated by the rules of the grammar.
- If a sentence is structurally ambiguous, more than one possible derivation is produced.

Context-Free Grammars

Why use the term "context-free"?

- The notion of context in CFGs has nothing to do with the ordinary meaning of the word context in language
- All it really means is that the non-terminal on the left-hand side of a rule can be replaced, regardless of context
  - Context-sensitive grammars allow context to be placed on the left-hand side of the rewrite rule

In programming languages and other uses of CFGs in computer science, notably XML, CFGs are:

- Unambiguous
  - Assign, at most, one structural description to a string
- Parsable in time, linearly proportional to the length of the string

CFG Derivations

School of Information Studies
Syracuse University

Example for Derivation

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

The sentence:

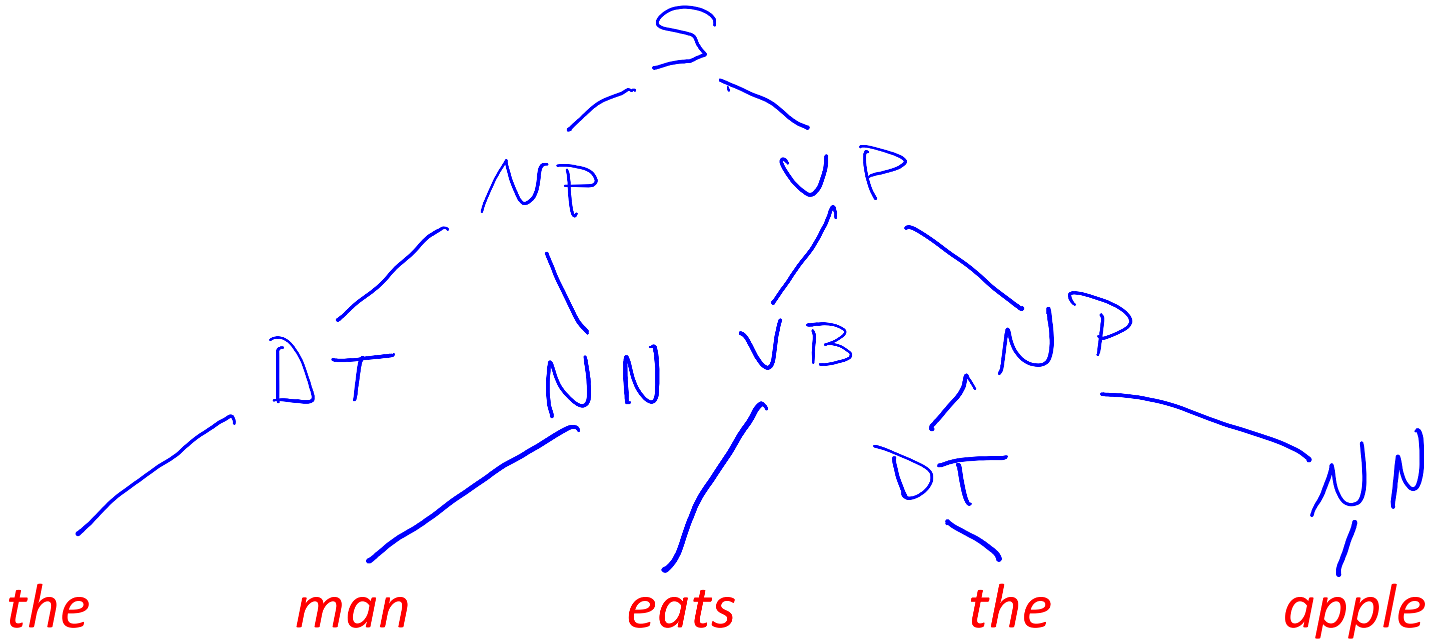
the *man* *eats* *the* *apple*

The (toy) grammar, with phrase structure rules on the left and lexical insertion rules on the right

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	

Top–Down Derivation

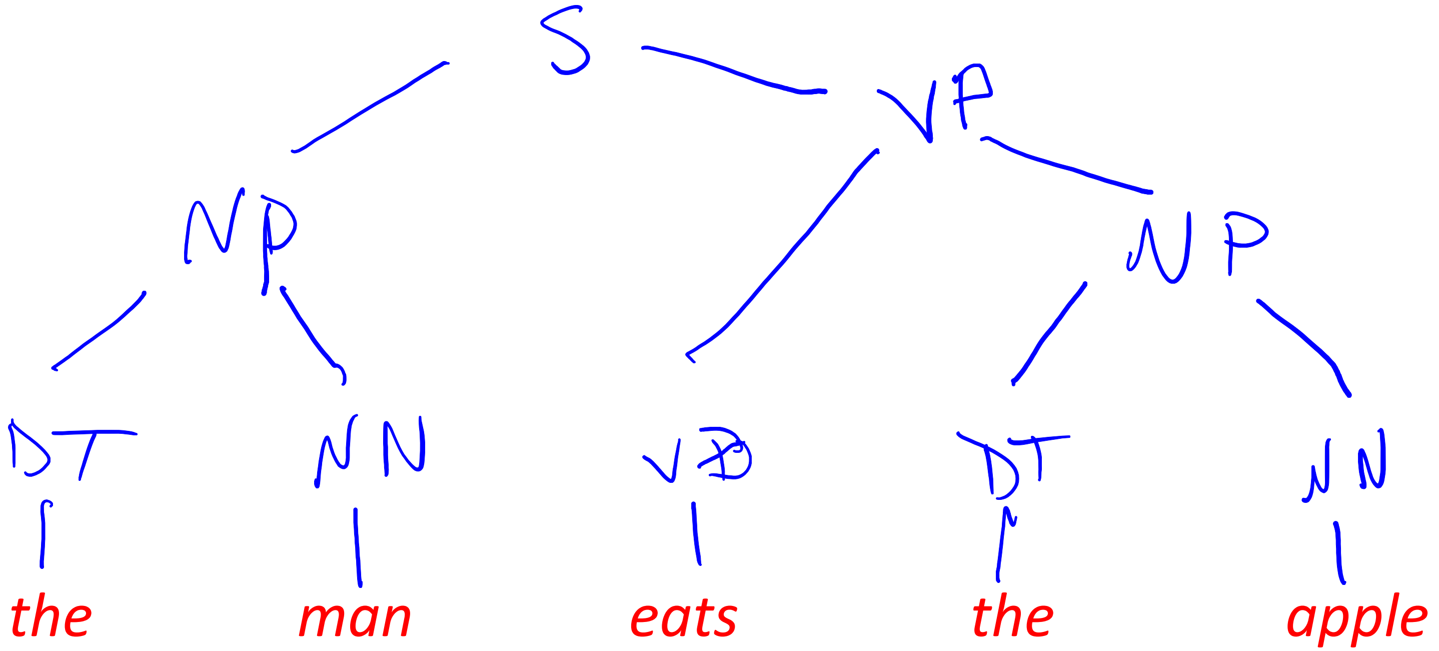


Context-free grammar rules (for this example):

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

School of Information Studies
Syracuse University

Bottom–Up Derivation



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	

CFG Phrases for English

School of Information Studies
Syracuse University

Key Constituents for English

English has headed phrase structure.

- X-bar theory: In natural languages, phrases are headed by particular kinds of words with modifiers and qualifiers around them (specifiers, adjuncts, and complements).

Verb phrases	VP	→	… VB* …
Noun phrases	NP	→	… NN* …
Adjective phrases	ADJP	→	… JJ* …
Adverb phrases	ADVP	→	… RB* …

For example, see the Penn Treebank Constituent tags

http://www.surdeanu.info/mihai/teaching/ista555-fall13/readings/PennTreebankConstituents.html

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- questions: When did the plane leave?

    S -> WH Aux NP VP

More general structure for sentences and clauses

SBAR → S | SINV | SQ …

- Sentences, inverted sentences, direct questions, etc. can also appear in the larger clause structure SBAR, where the sentence is preceded by *that*

Noun Phrases

The most common noun phrases are pronouns, proper nouns, and a determiner followed by a “nominal.”

NP -> Pronoun | Proper | Det Nominal

The nominals are noun phrases that have a head noun with pre- and post-modifiers.

- Pre-modifiers, including cardinals, ordinals, quantifiers, and adjective phrases, are all optional, indicated here with parentheses.

  Nominal -> (Card) (Ord) (Quan) (AP) Noun (Post-mods)

  Noun -> NN | NP | NPS | NNS (the four noun POS tags)

- Post-modifiers include prepositional phrases, gerundive phrases, and relative clauses.

  the man [from Moscow]

  any flights [arriving after 11p.m.] (gerundive)

  the spy [who came in from the cold] (relative clause)

Verb Phrases

Simple verb phrases

VP -> Verb	leave
| Verb NP	leave Boston
| Verb NP PP	leave Boston in the morning
| Verb PP	leave in the morning

Verbs may also be followed by a clause

VP -> Verb S

I think I would like to take a 9:30 flight.

The phrase or clause following a verb is sometimes called the complementizer

Other Phrases

Adjective and adverb phrases typically repeat adjective or adverb words:

Adjective phrase example: the long red dress

- But may also have an adverb: least expensive

Adverbs: quickly tomorrow

Prepositional phrase:

Typically PP -> Preposition NP *over the river*

Plus minor phrase types:

- QP quantifier phrase in NP
- CONJP (multi-word constructions: as well as)
- INTJ (interjections), etc.

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

(Syntax trees for these also need rules for NP -> Noun, etc.)

This grammar illustrates the recursion but may not give the best derivation for these phrases!

Conjunctive Constructions

S -> S and S

- John went to New York, and Mary followed him.

NP -> NP and NP

VP -> VP and VP

…

In fact, the right rule for English would be:

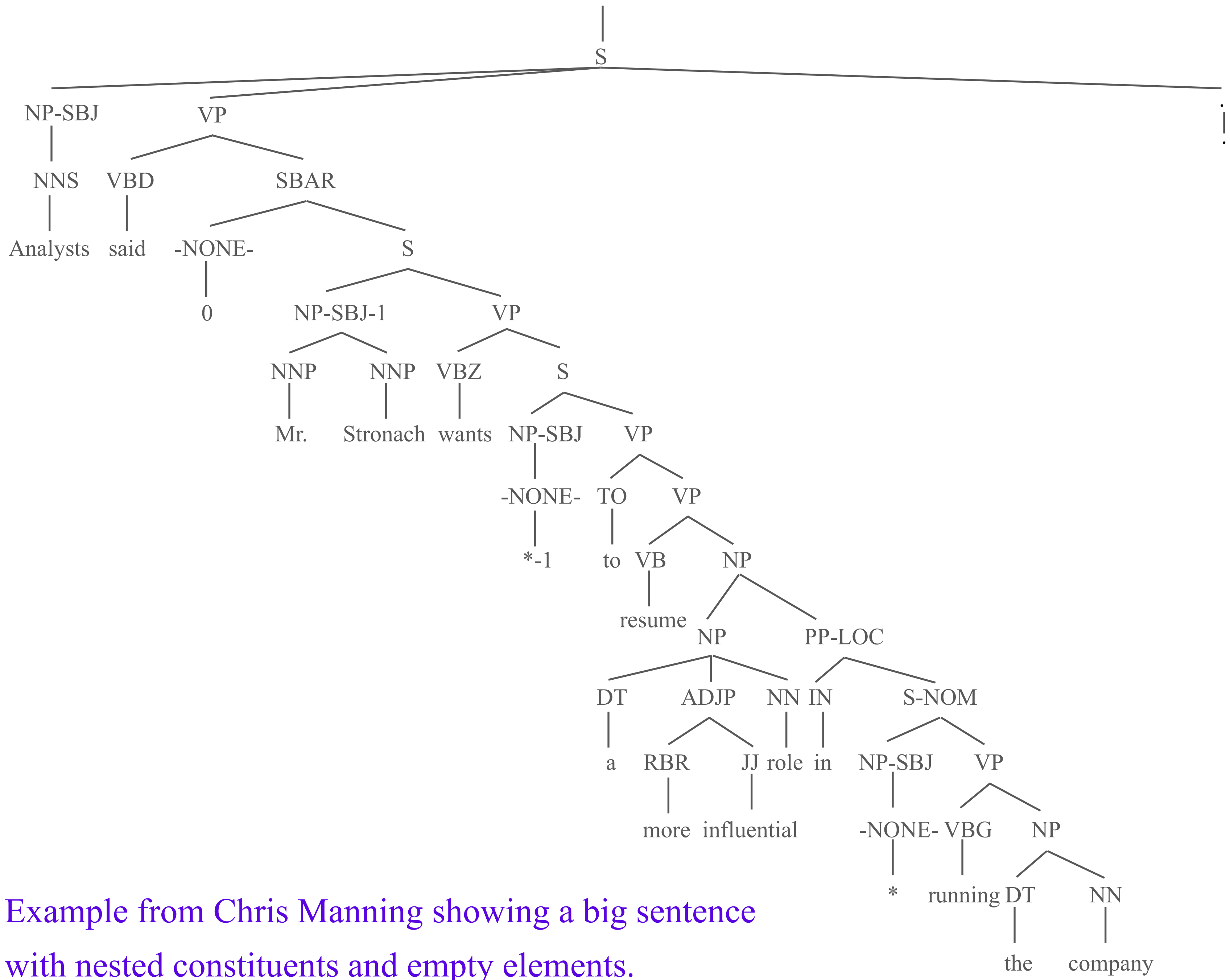
- X -> X and X
  for all constituents X, but this is not valid CFG

Example Grammar

Grammar Rules	Example
S → *NP VP*	I + want morning flight
NP → *Pronoun*	I
| *Proper Noun*	Los Angeles
| *Det Nominal*	a + flight
Nominal → *Nominal Noun*	morning + flight
| *Noun*	flights
VP → *Verb*	do
| *Verb NP*	Want + a flight
| *Verb NP PP*	leave + Boston + in the morning
| *Verb PP*	leaving + on Thursday
PP → *Preposition NP*	from + Los Angeles

Figure 11.3. The grammar for $\mathscr{L}_0$, with example phrases for each rule.

Jurafsky and Martin, Chapter 11



Example from Chris Manning showing a big sentence with nested constituents and empty elements.

Problems for CFGs

School of Information Studies
Syracuse University

Problems

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 (for want of a better term)

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

Many languages have agreement systems that are far more complex than this.

Solution can be either to add rules with agreement or to have a layer on the grammar called the features.

Subcategorization

Subcategorization expresses the constraints that a particular verb (sometimes called the 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
- Help: *Can you help* [me]NP [with a flight]PP
- Prefer: I *prefer* [to leave earlier]TO-VP
- Told: *I was told* [United has a flight]S
- …

Subcategorization

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
- For example: VP -> V NP
- Therefore, *sneezed the book* is a VP since "sneeze" is a verb and "the book" is a valid NP

Now overgeneration is a problem for a generative approach.

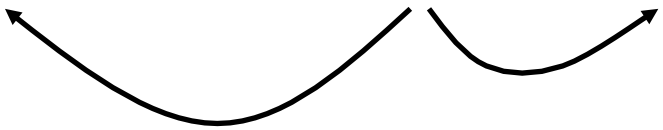
- The grammar should represent all and only the strings in a language

From a parsing point of view, it's not so clear that there's a problem.

Movement

Consider the verb “booked” in the following example:

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



That is, “book” is a straight-forward transitive verb. It expects a single NP argument within the VP as an argument, and a single NP argument 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 in the right place. It is in fact a long way from where it’s supposed to appear.

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

In Penn Treebank, these types of movement are annotated by having an empty Trace constituent appear in the right place.

The Point About CFGs

CFGs appear to be just about what we need to account for a lot of basic syntactic structure in English.

But there are problems that:

- Can be dealt with adequately, although not elegantly, by staying within the CFG framework

There are simpler, more elegant, solutions that take us out of the CFG framework (beyond its formal power).

- For example, feature structures for CFGs place constraints on how the rules can be applied

Dependency Grammars

School of Information Studies
Syracuse University

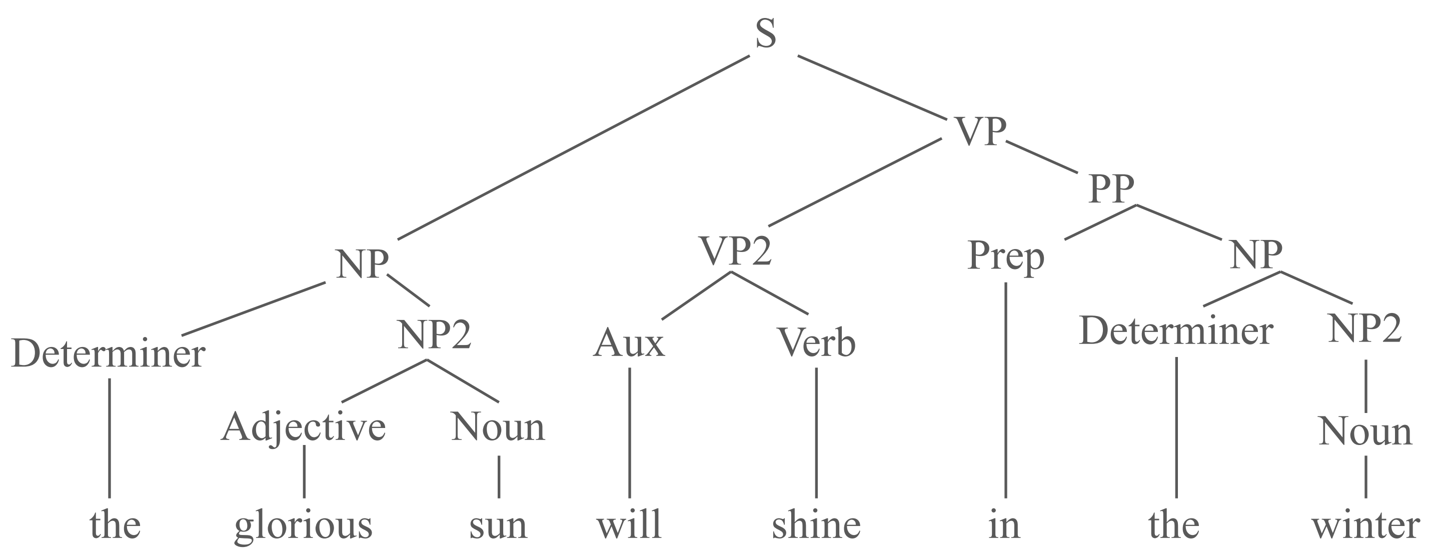
Dependency Grammars

Dependency grammars offer a different way to represent syntactic structure.

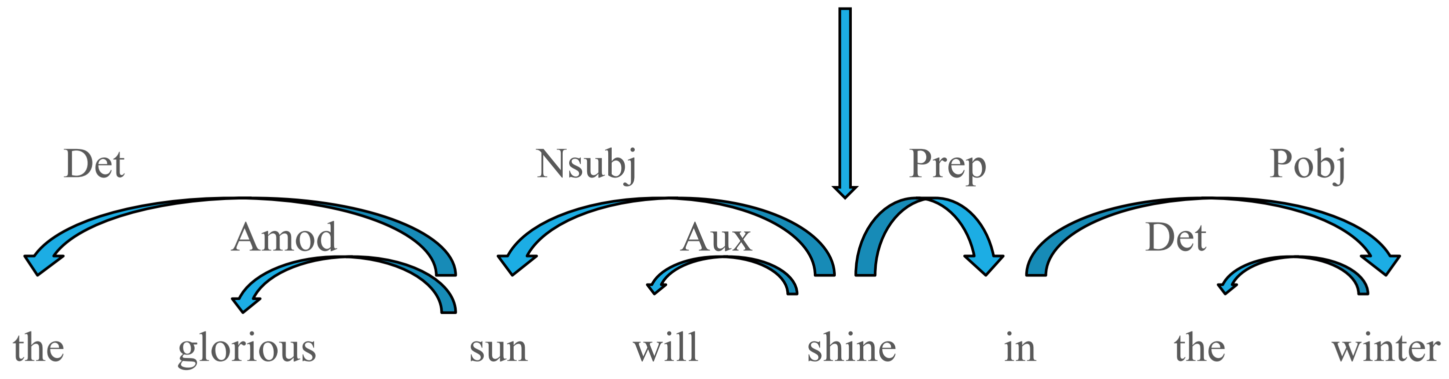
- CFGs represent constituents in a parse tree that can derive the words of a sentence.
- Dependency grammars represent syntactic dependency relations between words that show the syntactic structure.
- Typed dependency grammars label those relations as to what the syntactic structure is.

Syntactic structure is the set of relations between a word (a.k.a. the head word) and its dependents.

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

The examples on the previous slide used those from the Stanford dependency parser.

- De Marneffe, MacCartney, & Manning, "Generating Typed Dependency Parses From Phrase Structure Parses," LREC (Language Resources and Evaluation Conference), 2006.

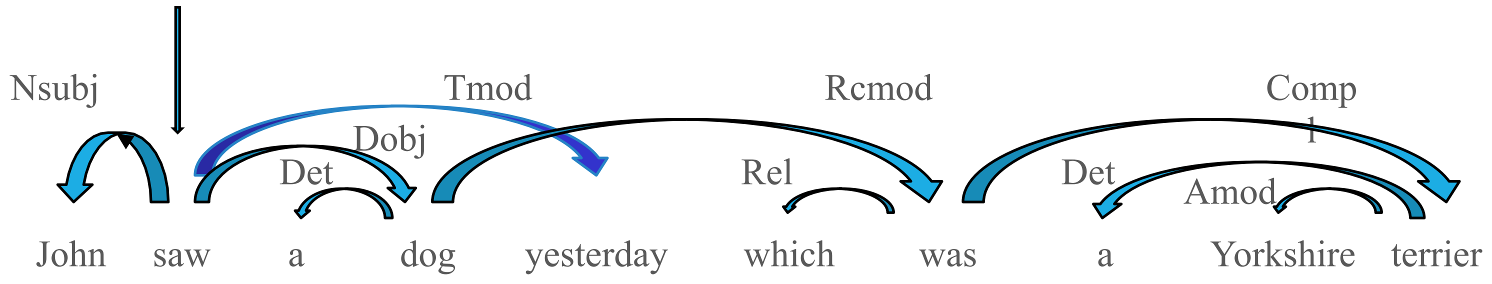
Examples of Dependency Relations

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

Projective vs. Non-Projective

In the dependency graph as depicted in the previous example, with the words in sentence order, if no arcs cross, then it is a projective tree.

If there are crossing arcs, then it is a non-projective tree.



One study found that, of the languages that had non-projective sentences in the treebanks, from 0.5–5% of the sentences were non-projective.

Non-projective trees are a problem for parsing, not for expressive power of the grammar.

Dependency Grammar vs. CFG

Dependency grammars and CFGs are strongly equivalent

- Generate the same sentences and make the same structural analysis

Provided that the CFGs are restricted in that one word or phrase can be designated as its "head"

- This restriction is also accepted by linguists in X-bar theory
- Note that the head of a noun phrase is a noun, the head of a verb phrase is a verb, etc.

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 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

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
- 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

Recursive Descent parser does this strategy

- Includes a backtracking mechanism
  - When it is determined that the wrong rule has been used, it backs up and tries another rule
- Must not have rules of the form NT -> NT X X
  - Tries to apply leftmost Non-Terminal and gets into infinite loop

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 that 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

Also requires back-tracking if the wrong action is taken that does not lead to a derivation

Bottom–Up Parsing Algorithm

Algorithm called shift/reduce parsing

- Scans the input from left to right and keeps a “stack” of the partial parse tree so far
- Chooses shift or reduce operations
  - 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 incorporate back-tracking and 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

Ambiguity:

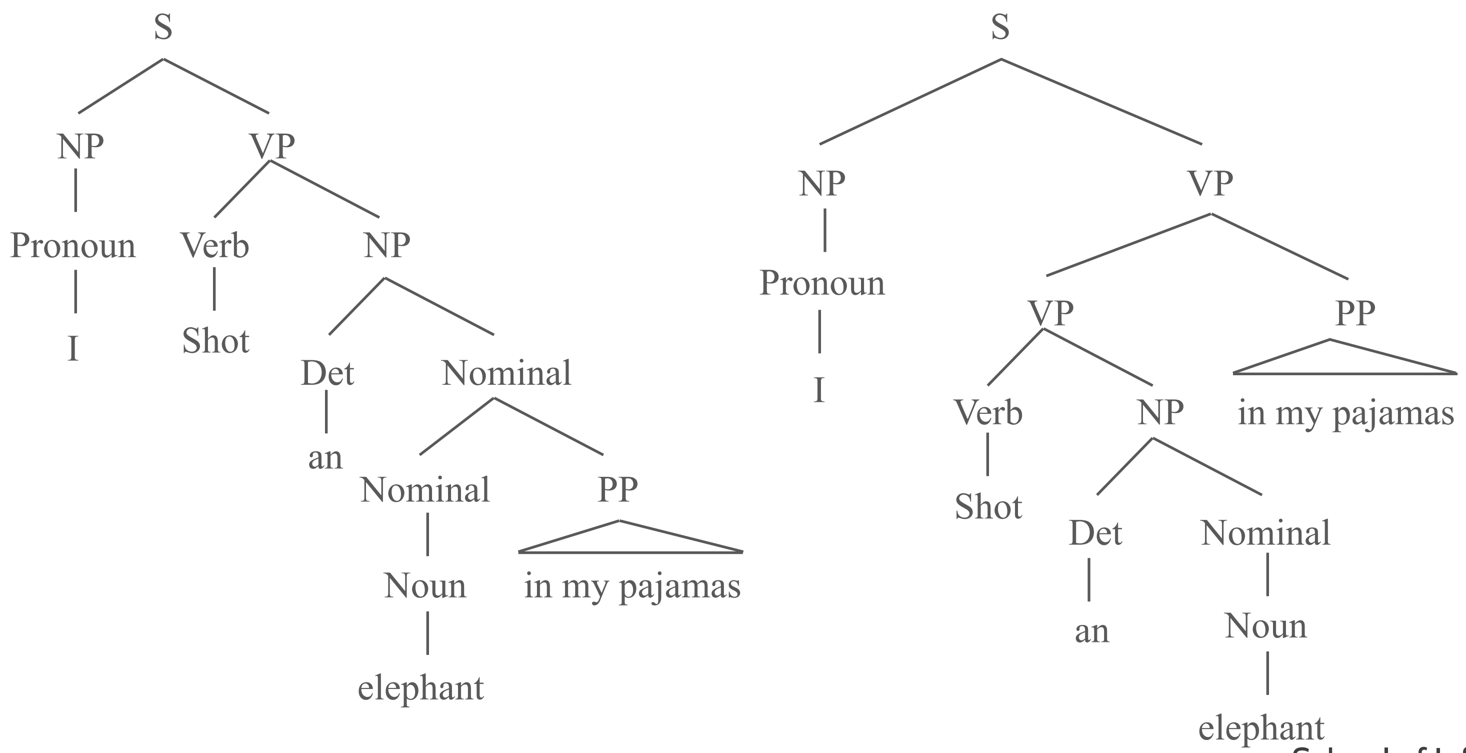
- Some word-level ambiguities may be resolved
- But examples with structural ambiguity will not be resolved, resulting in more than one possible derivation

Performance of backtracking is exponential in time:

- Backtracking may result in smaller sub-trees being parsed many times

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



School of Information Studies
Syracuse University

Parsing Demo With NLTK

School of Information Studies
Syracuse University

Top–Down Parsing Demo

NLTK parsing demos

- Top-down parsing using a recursive descent algorithm
  - Top down parsing with back-tracking
  - Must not have left-recursion in the grammar rules

```
nltk.app.rdparser()
```

Demo operation: can go one step at a time or automatic

- Although uses back-tracking to apply all rules when it gets stuck, it does not continue to find all parses
- Note the use of highlighting as it tries rules.

Described in NLTK book, Chapter 8, Analyzing Sentence Structure

Bottom–Up Parsing Demo

NLTK parsing demos

- 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

```
nltk.app.srparser()
```

Demo operation: Best to use step as there is no backtracking

- It does highlight rules when the rhs is eligible for reduce

Described in NLTK book, Chapter 8, Analyzing Sentence Structure

Parsing Algorithms: Charts

School of Information Studies
Syracuse University

Solutions to Parsing Problems

Modern parsing algorithms have three key ideas:

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).

Parsing Algorithms

The simple parsers that we have seen are exponential in time (recursive descent with back-tracking) and (shift reduce with back-tracking) with respect to the length of the input.

Avoid back-tracking and re-doing sub-trees.

- Recall that the back-tracking recursive descent expanded some sub-trees multiple times.
- Use charts to record sub-trees to avoid redundant computation.

Use forms of dynamic programming to search for good parse trees.

- Attempt to perform an exponential process in polynomial time by saving partial computations.

Binarization: Chomsky Normal Form

All CFG grammars have a Chomsky Normal Form, where every rule has no more than two symbols on the RHS.

- Example grammar rule **with three RHS symbols**:

  VP -> Verb NP PP

- Transformed to equivalent grammar **with only two RHS symbols**:

  VP -> Verb NPtemp
  NPtemp -> NP PP

Using rules with only two RHS symbols allows the chart algorithm to “binarize” its operation.

- Only requires two-dimensional charts

Chart Parsers

CKY (Cocke–Kasami–Younger) algorithm is an example

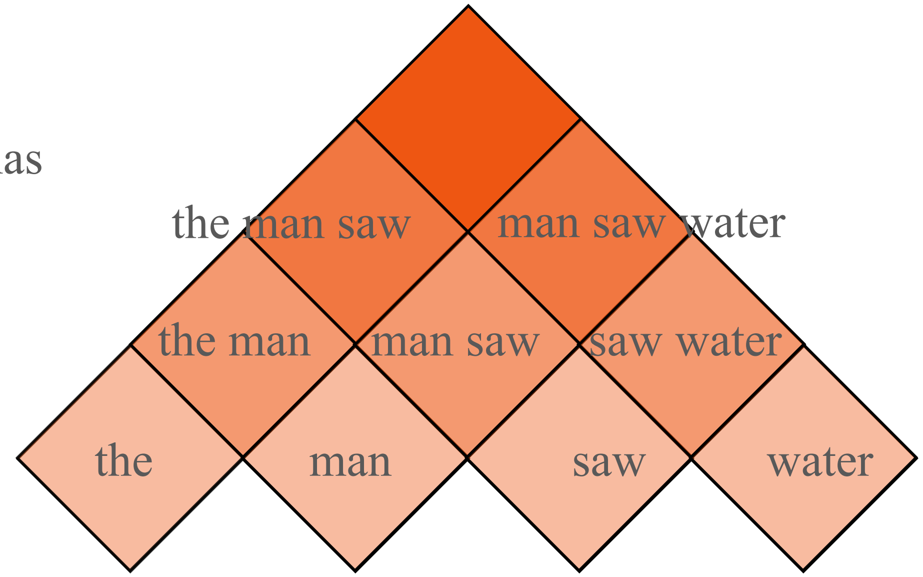
- Bottom–up parser (but can also have top–down chart parsers)
- Requires grammar to be in Chomsky Normal Form, with only two symbols on the right-hand side of each production
- Fills in a data structure called a chart or a parse triangle
  - Other parsers, such as Earley’s algorithm, use similar chart ideas to work on two sub-trees at a time

Key idea in parser development from 1970–1990

CKY 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 the first (bottom) layer describe trees of single words
  - Cells in second layer describes how rewrite rules can be used to combine trees in the first layer for trees with two words
  - Etc.

Each cell has rules for:



School of Information Studies
Syracuse University

S → NP VP	0.9
S → VP	0.1
VP → V NP	0.5
VP → V	0.1
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
NP → N	0.7
PP → P NP	1.0
N → *people*	0.5
N → *fish*	0.2
N → *tanks*	0.2
N → *rods*	0.1
V → *people*	0.1
V → *fish*	0.6

0	fish 1	people 2	fish 3	tanks 4
0	N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.0049 VP → V NP 0.105 S → VP 0.0105	NP → NP NP 0.0000686 VP → V NP 0.00147 S → NP VP 0.000882	NP → NP NP 0.0000009604 VP → V NP 0.00002058 S → NP VP 0.00018522
1		N → people 0.5 V → people 0.1 NP → N 0.35 VP → V 0.01 S → VP 0.001	NP → NP NP 0.0049 VP → V NP 0.007 S → NP VP 0.0189	NP → NP NP 0.0000686 VP → V NP 0.000098 S → NP VP 0.01323
2			N → fish 0.2 V → fish 0.6 NP → N 0.14 VP → V 0.06 S → VP 0.006	NP → NP NP 0.00196 VP → V NP 0.042 S → VP 0.0042
3				N → tanks 0.2 V → tanks 0.1 NP → N 0.14 VP → V 0.03 S → VP 0.003
4				

Example showing filled-in CKY chart for a PCFG for sentence "fish people fish tanks" from Chris Manning

School of Information Studies
Syracuse University

Parsing: Treebanks

School of Information Studies
Syracuse University

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 (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 (manually corrected by human annotators); 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.

Classical Parsing

Another problem with grammar engineering

In addition to correctly finding multiple parses due to structural ambiguity, typical grammars built before the 1990s would overgeneralize

- Real broad-coverage language grammar could give rise to millions of parses on a single sentence

Structuring grammar to restrict parses would leave up to 30% of the sentence without parses

The Rise of Annotated Data

Starting off, building a treebank seems a lot slower and less useful than building a grammar.

- Using human effort to annotate large amounts of text

But a treebank gives us many things.

- Reusability of the labor
  - Many parsers, POS taggers, etc.
  - Valuable resource for linguistics
- Broad coverage
- Frequencies and distributional information
- A way to evaluate systems on the same text

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*)

Getting Grammar From a Treebank

What grammar is represented in rules from the treebank?

Given an annotated sentence,

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

we can make a grammar rule:

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

and we'll make rules for sub-trees as well.

School of Information Studies
Syracuse University

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

We could read off the grammar from the treebank

- The grammar is the set of rules (local sub-trees) that occur in the annotated corpus.
- They tend to avoid recursion (and elegance and parsimony).
  - That is, they tend to the flat and redundant.
- Penn Treebank (III) has about 17,500 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!).

- **If we have a treebank, we don’t need to write down a grammar.**

Parsing: Probabilistic CFG

School of Information Studies
Syracuse University

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.

- 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).

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 maximum 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 phrase 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
  - Example: the parent of the non-terminal of the derivation
  - Most probable parse isn't usually the right one (the one in the treebank test set)

Collect statistics and use them in a better way.

Parsing: Lexicalized Statistical

School of Information Studies
Syracuse University

Lexicalized Statistical Parsing

Add lexical dependencies to the scheme of probabilities

- Integrate the preferences of particular words into the probabilities in the derivation.
- That is, condition the rule probabilities on the actual words.

To do that, we use of the notion of the head of a phrase

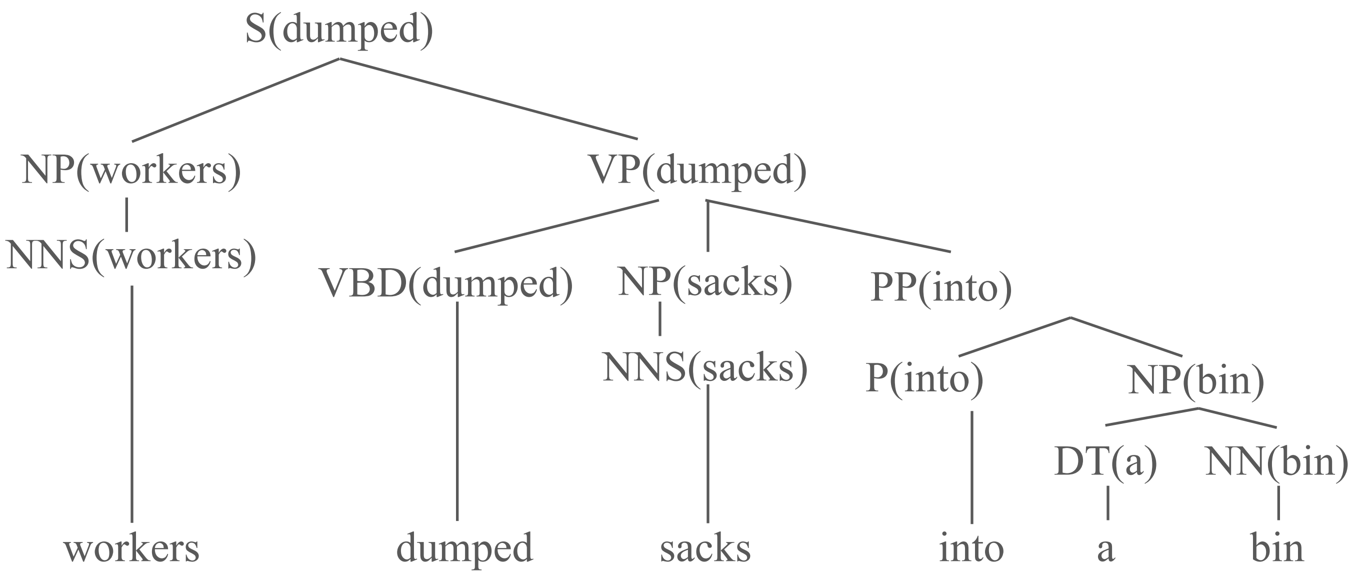
- The head of an NP is its noun.
- The head of a VP is its verb.
- The head of a PP is its preposition.
- (It’s really more complicated than that, but this will do.)

Main parsing breakthrough idea of the 1990s

Expand the set of phrase types with phrase type/word

Example (Right)

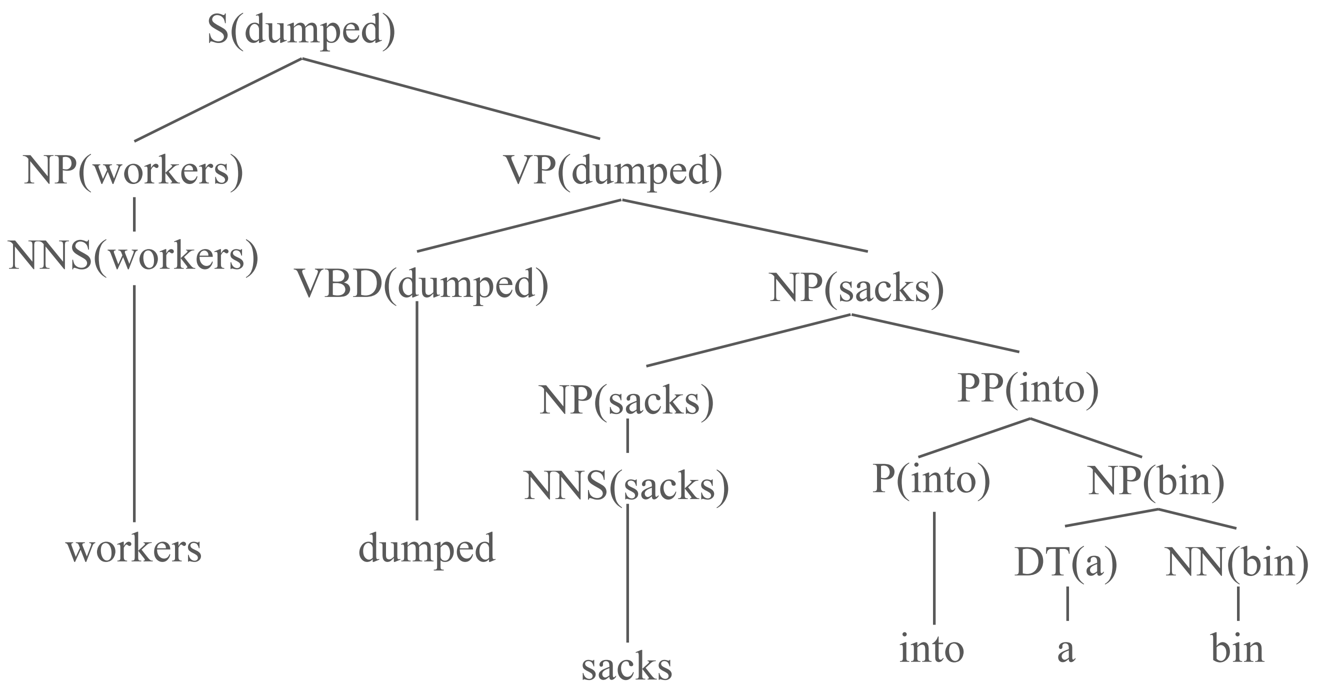
Should we attach the prepositional phrase with head “into” to the verb “dumped”?



In this tree, each phrase type, such as NP or VP, is also shown with its attached head word.

Example (Wrong)

Or should we attach the prepositional phrase with head “into” to the noun “sacks”?



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 child with “into” as its head and normalize
- Versus the situation where “sacks” is a constituent with “into” as the head of a PP child

In general, collect statistics on preferences (a.k.a. affinities).

- Use verb sub-categorization
  - Particular verbs have affinities for particular VPs
- Objects affinities for their verbs, mostly their parents and grandparents
  - Some objects fit better with some verbs than others

Preference Example (1)

Consider the VPs.

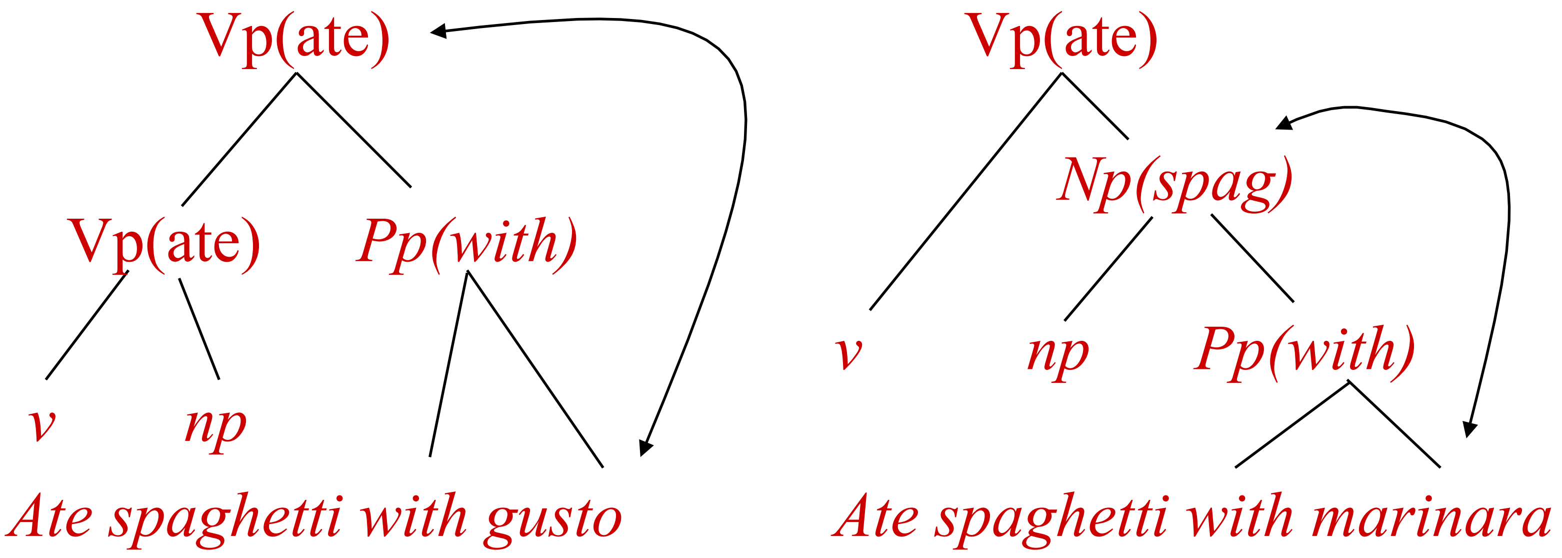
- *Ate spaghetti with gusto*
- *Ate spaghetti with marinara*

The affinity of “gusto” for “eat” is much larger than its affinity for “spaghetti.”

And, the affinity of “marinara” for “spaghetti” is much higher than its affinity for “ate.”

Preference Example (2)

Note that 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.”

Dependency Parsing

Dependency parsing has some resemblance to lexicalized statistical 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

Parsing algorithms are similar to constituent parses.

- Statistics for dependency relations learned from Penn Treebank
- Used bottom–up parser to find best parse(s)
- Some additional mechanism used to find non-projective parses

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.

There is a lot 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.

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 the content of the trees are correctly produced

- Evaluation measures based on the correct number of constituents (or sub-trees) in the system compared with 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?
  - Reduce the number of false positives

Evaluation

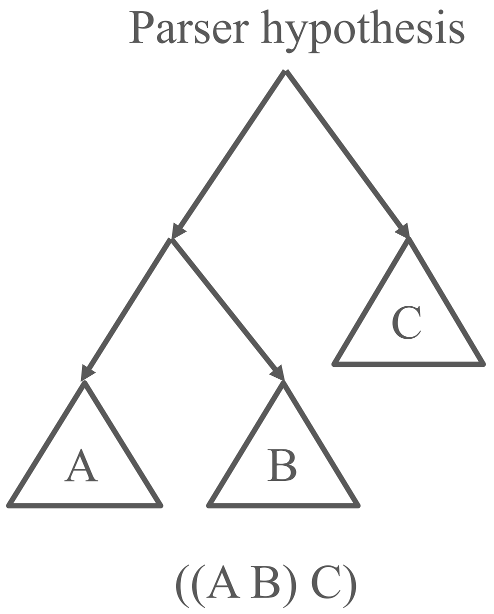
Recall

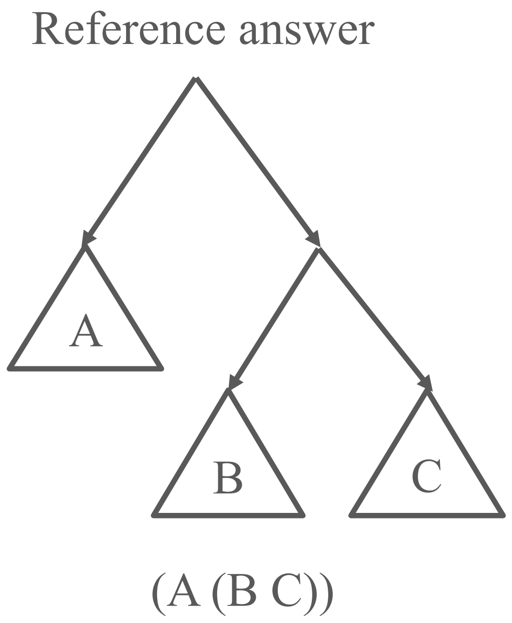
- What fraction of the sub-trees in the reference answer did we actually get?
  - How much of what we should have gotten did we get?
  - Reduce the number of false negatives

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

Evaluation

An additional evaluation measure that is often reported is that of crossing-brackets errors, in which the sub-trees are equal, but they are put together in a different order.





Performance of Parsers

The Charniak series of parsers is still under development 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.

- It combines “separate PCFG phrase structure and lexical dependency experts.”
- A demo can be found at http://nlp.stanford.edu:8080/parser/.