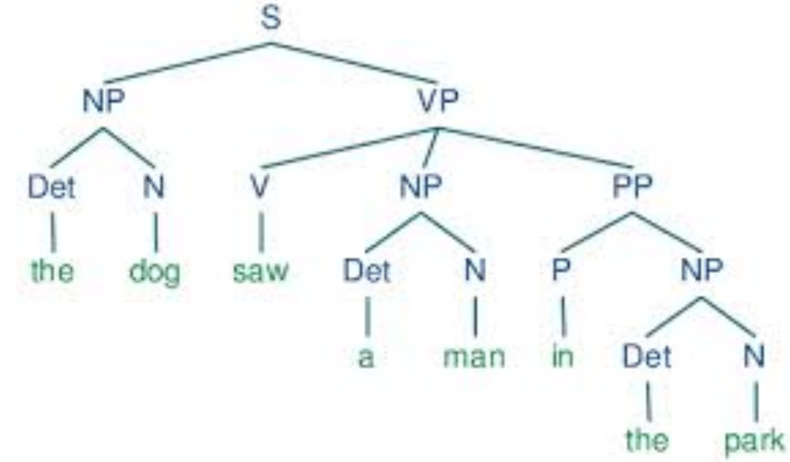
Context-free Grammars

COMP90042 LECTURE 17



Syntactic Constituents

- Sequential models like HMMs (regular grammars, etc) assume a linear structure
- But language clearly isn't like that

[*A man*] [*saw* [*a dog*] [*in* [*the park*]]]

- Words group together to form syntactic constituents
  - Can be replaced, or moved around *as a unit*
- Grammars allow us to formalize these intuitions
  - Symbols correspond to syntactic constituents

Testing for constituency

- Various tests for constituency, based on linguistic intuition, e.g.,
  - *Only constituents can answer a question*

Trevor gave a lecture on grammar

Who gave the lecture on grammar?	*Trevor*
Trevor did what with the lecture on grammar?	**gave (fails)*
What topic was Trevor's lecture on?	*on grammar*

  - *Only constituents can be coordinated with others (of same type)*

Trevor gave a lecture on grammar and on parsing

Trevor gave a lecture on grammar and parsing

Trevor gave a lecture on grammar and a treatise on parsing

Trevor gave a lecture on grammar and ate a tasty pie

#Trevor gave a lecture on grammar and away a tasty pie

#Trevor gave a lecture on and a treatise about grammar

Outline

- The context-free grammar formalism
- Parsing with CFGs
- Representing English with CFGs

Basics of Context-free grammars

- **Symbols**
  - **Terminal**: word such as *book*
  - **Non-terminal**: syntactic label such as NP or NN
  - Convention to use upper and lower-case to distinguish, or else “quotes” for terminals
- **Productions** (rules)

$$W \rightarrow X\ Y\ Z$$

  - Exactly one non-terminal on left-hand side (LHS)
  - An ordered list of symbols on right-hand side (RHS)
    — can be **Terminals** or **Non-terminals**

Regular expressions as CFGs

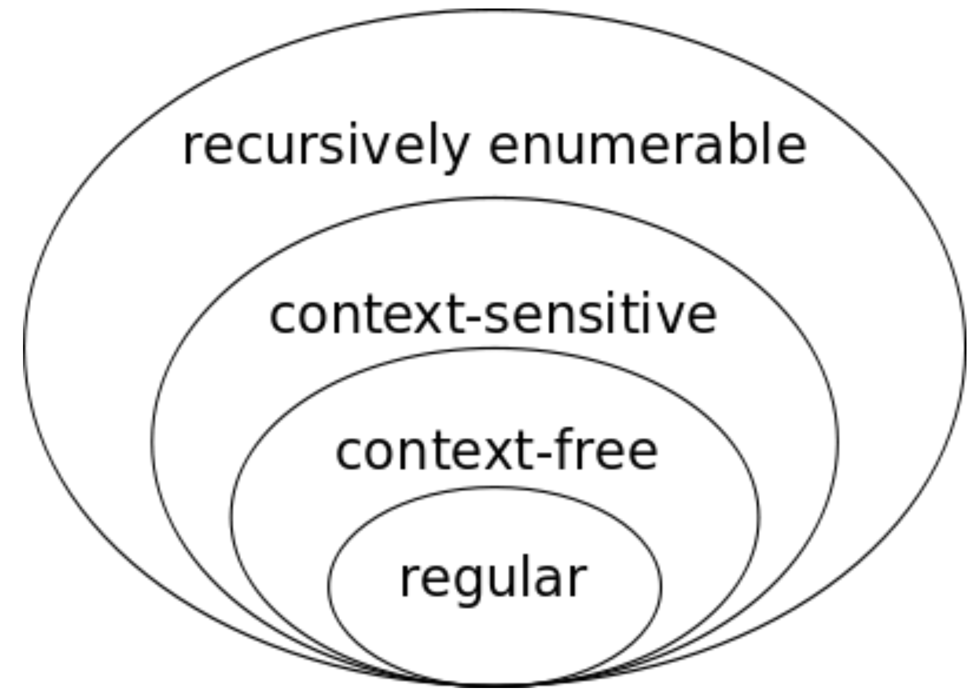
- Regular expressions match simple patterns
  - E.g., [A-Z][a-z]* words starting with a capital
- Can rewrite as a grammar (“regular grammar”)
  - S → U S → U LS
  - U → “A” U → “B” … U → “Z”
  - LS → L LS → L LS
  - L → “a” L → “b” … L → “z”
- The class of regular languages is a subset of the **context-free languages**, which are specified using a CFG

CFGs vs regular grammars

- CFGs (and regexs) used to describe a set of strings, aka a "*language*"
- Regular grammars
  - describe a smaller class of languages
  - can be parsed using finite state machines (FSA, FST)
- CFGs
  - can describe hierarchical groupings
  - requires more complex automata to parse (PDA)
- Context sensitive grammars are even more expressive (and intractable)

Chomsky hierarchy

- CF languages *more general* than RLs
  - Allows representation of recursive nesting



- Adequate for most constructions in natural language
  - but not e.g., cross-serial dependencies in Swiss-German

images: https://en.wikipedia.org/wiki/Chomsky_hierarchy https://en.wikipedia.org/wiki/Cross-serial_dependencies

A simple CF grammar

Terminal symbols: *rat, the, ate, cheese*

Non-terminal symbols: S, NP, VP, DT, VBD, NN

Productions:

S → NP VP

NP → DT NN

VP → VBD NP

DT → *the*

NN → *rat*

NN → *cheese*

VBD → *ate*

Generating sentences with CFGs

Always start with S (the sentence/start symbol)

S

Apply a rule with S on LHS (*S* → *NP VP*), i.e substitute RHS

NP VP

Apply a rule with NP on LHS (*NP* → *DT NN*)

DT NN VP

Apply rule with DT on LHS (DT → *the*)

***the* NN VP**

Apply rule with NN on LHS (NN → *rat*)

***the rat* VP**

In each step we rewrite the left-most non-terminal

Generating sentences with CFGs

Apply rule with VP on LHS (VP → VBD NP)

***the rat* VBD NP**

Apply rule with VBD on LHS (VBD → *ate*)

***the rat ate* NP**

Apply rule with NP on LHS (NP → DT NN)

***the rat ate* DT NN**

Apply rule with DT on LHS (DT → *the*)

***the rat ate the* NN**

Apply rule with NN on LHS (NN → *cheese*)

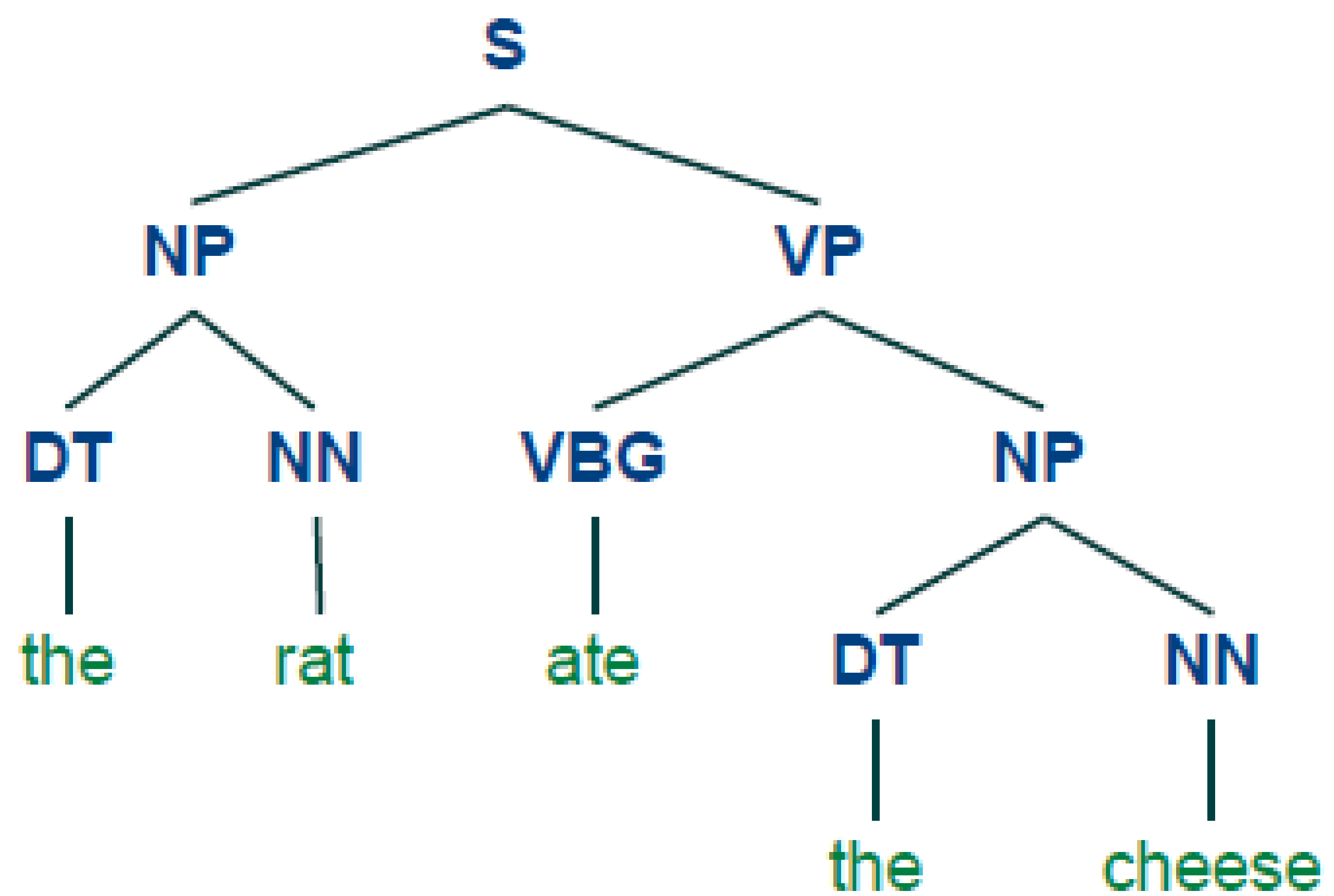
the rat ate the cheese

No non-terminals left, we're done!

CFG trees

- Generation corresponds to a syntactic tree
- Non-terminals are internal nodes
- Terminals are leaves

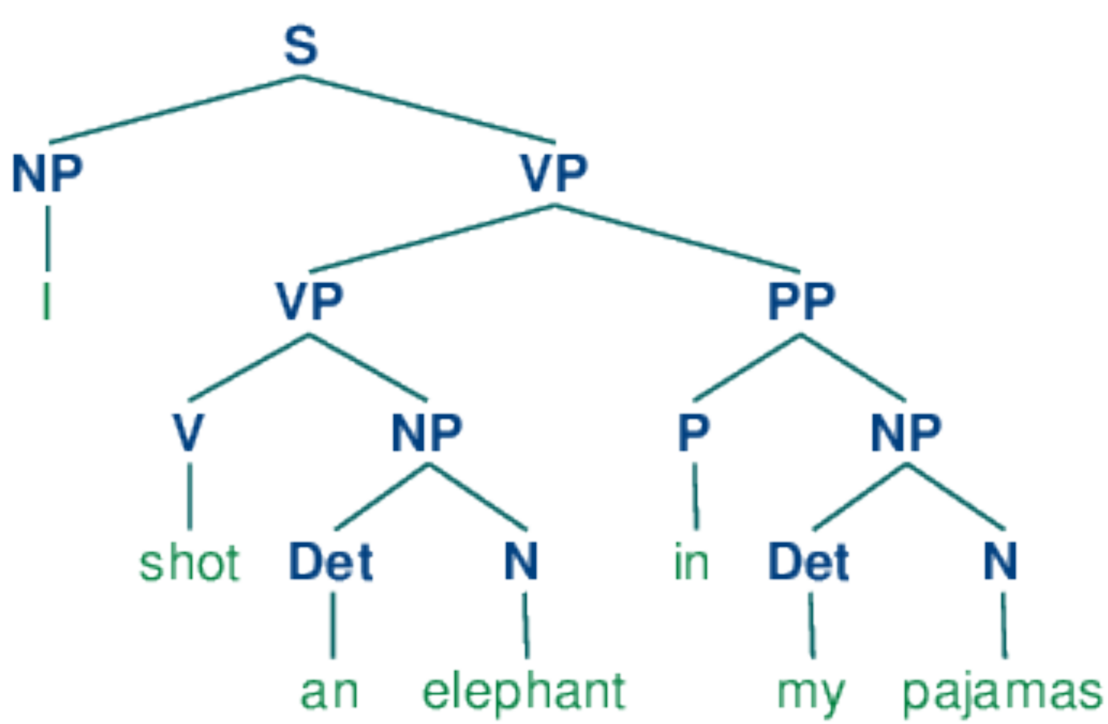
```
(S (NP (DT the)
       (NN rat))
   (VP (VBG ate)
       (NP (DT the)
           (NN cheese))))
```

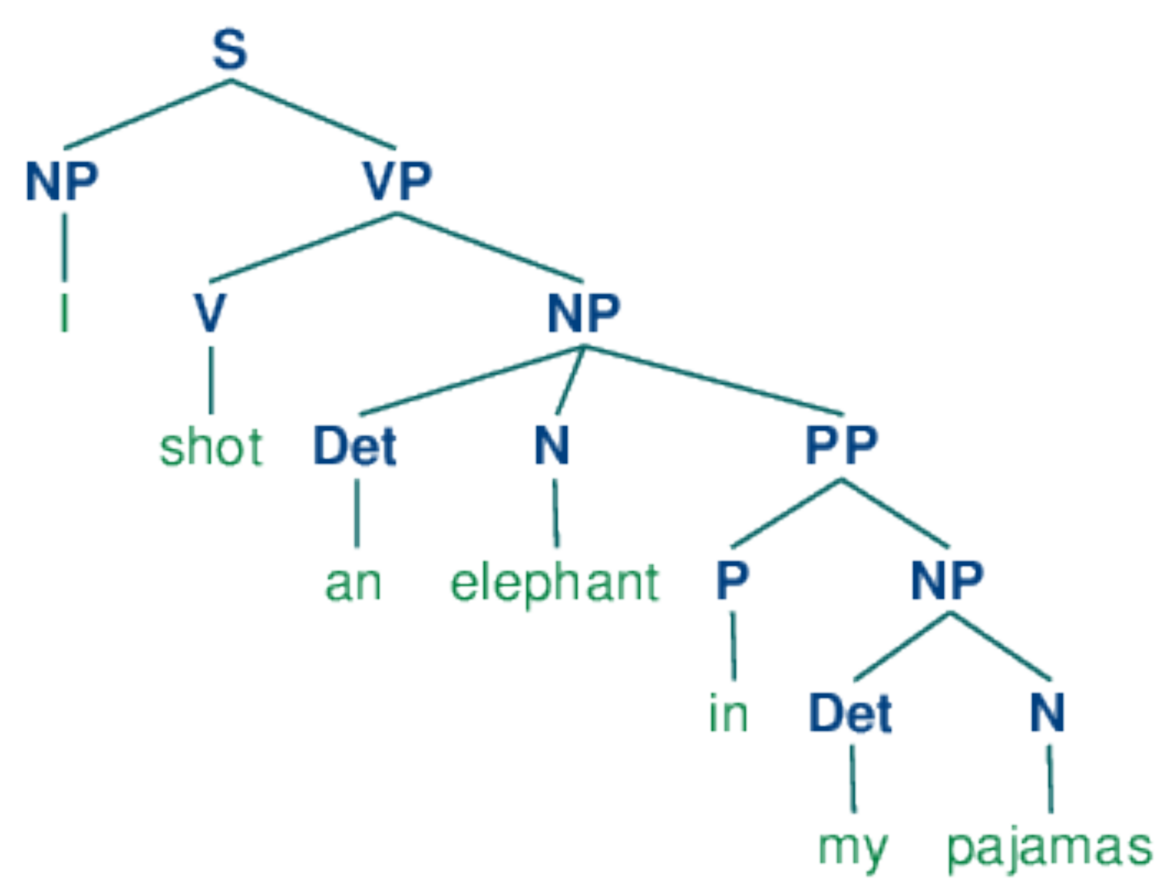


- Parsing is the *reverse* process

Parse Ambiguity

- Often more than one tree can describe a string
- *"While hunting in Africa, I shot an elephant in my pajamas. How he got into my pajamas, I don't know."*
  Animal Crackers (1930)





Example & figures: http://www.nltk.org/book/ch08.html

Parsing CFGs

- Parsing: given string, identify possible structures
- Brute force search is intractable for non-trivial grammars
  - Good solutions use dynamic programming
- Two general strategies
  - Bottom-up
    - Start with words, work up towards S
    - CYK parsing
  - Top-down
    - Start with S, work down towards words
    - Earley parsing (not covered)

The CYK parsing algorithm

- Convert grammar to Chomsky Normal Form (CNF)
- Fill in a parse table
- Use table to derive parse
- Convert result back to original grammar

Convert to Chomsky Normal Form

- Change grammar so all rules of form
  A → B C or A → a
- Step 1: Convert rules of form
  A → B c into pair of rules A → B X, X → c
  - Not usually necessary in POS-based grammars
- Step 2: Convert rules A → B C D into A → B Y, Y → C D
  - Usually necessary, but not for our toy grammar
  - E.g., VP → VP NP NP
    for ditransitive cases, "*sold [her] [the book]*"
- X, Y are new symbols we have introduced

CNF (cont)

- CNF disallows unary rules, A → B. Why?
- Imagine NP → S; and S → NP … leads to infinitely many trees with same yield.
- If no cycles, can transform grammar, e.g.,
  - if A → B and B → c and B → d then make new non-terminal Z, with rules Z → c and Z → d; all instances of A in RHS of other rules now also support Z.
  - common occurrence in formal grammars, e.g., NP → NN, VP → VB, where NN and VB are pre-terminals (POS tags), and only rewrite as strings

CYK algorithm

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

  for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
    for all $\{A \mid A \rightarrow words[j] \in grammar\}$
      $table[j-1, j] \leftarrow table[j-1, j] \cup A$
    for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
      for $k \leftarrow i+1$ **to** $j-1$ **do**
        for all $\{A \mid A \rightarrow BC \in grammar$ **and** $B \in table[i,k]$ **and** $C \in table[k,j]\}$
          $table[i,j] \leftarrow table[i,j] \cup A$

Figure 12.5 The CKY algorithm.

- What role do i, j and k play?
- Why does this need CNF grammar?
- How to use table for checking acceptance? Finding tree?

JM3, Ch 11

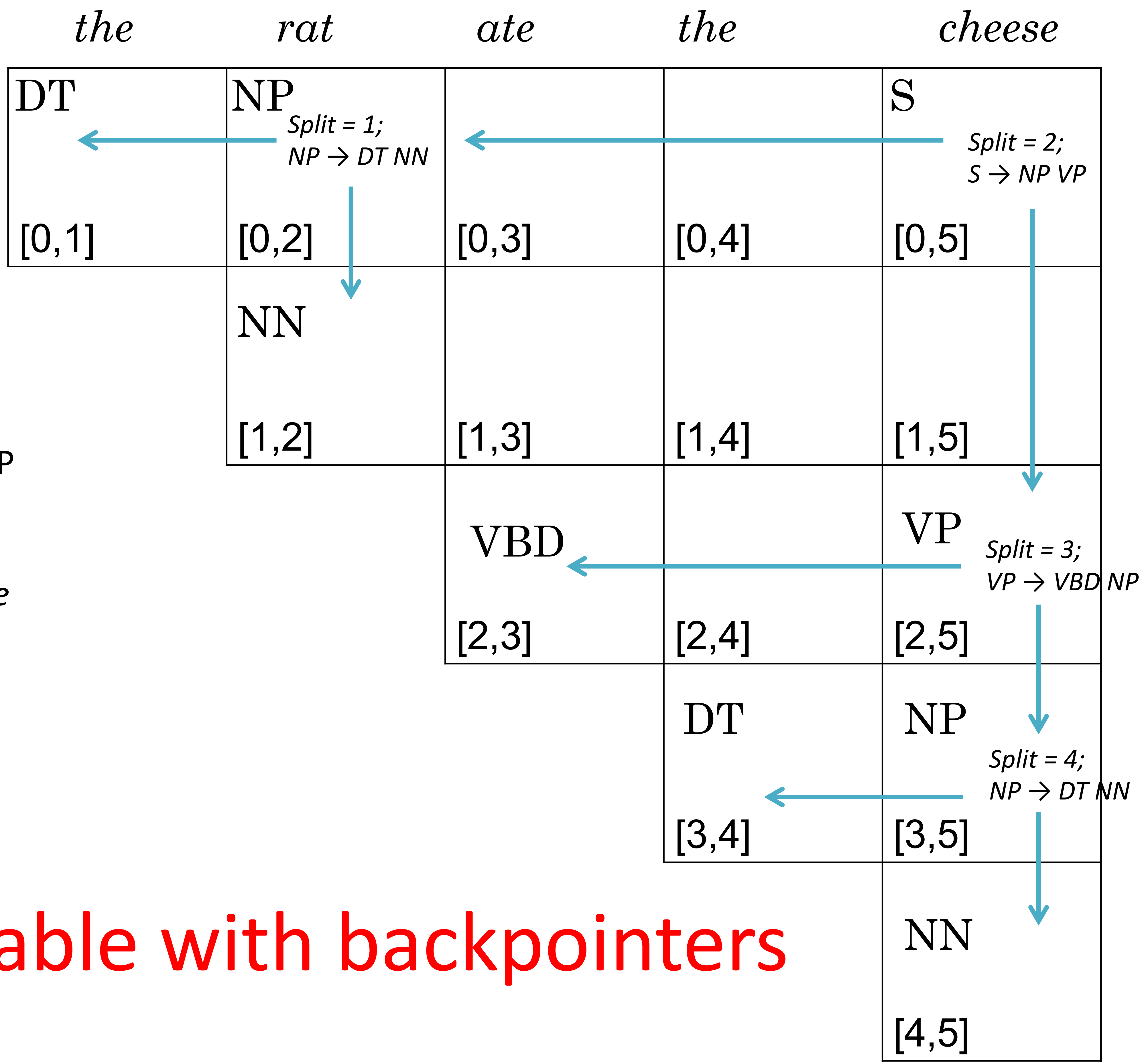
CYK by example

the	*rat*	*ate*	*the*	*cheese*
DT [0,1]	NP [0,2]	[0,3]	[0,4]	S [0,5]
	NN [1,2]	[1,3]	[1,4]	[1,5]
		VBD [2,3]	[2,4]	VP [2,5]
			DT [3,4]	NP [3,5]
				NN [4,5]

S → NP VP
NP → DT NN
VP → VBD NP
DT → *the*
NN → *rat*
NN → *cheese*
VBD → *ate*

CYK: Retrieving The parses

- S in the top-left corner of parse table indicates success
- To get parse(s), follow pointers back for each match
- Convert back from CNF by transforming new non-terminals back to their original values
  - E.g., if VP → VP NP NP was changed to
    VP → VP NP+NP; NP+NP → NP NP
  - If we have the latter two productions in tree, transform tree back to top production



S → NP VP
NP → DT NN
VP → VBD NP
DT → *the*
NN → *rat*
NN → *cheese*
VBD → *ate*

Parse table with backpointers

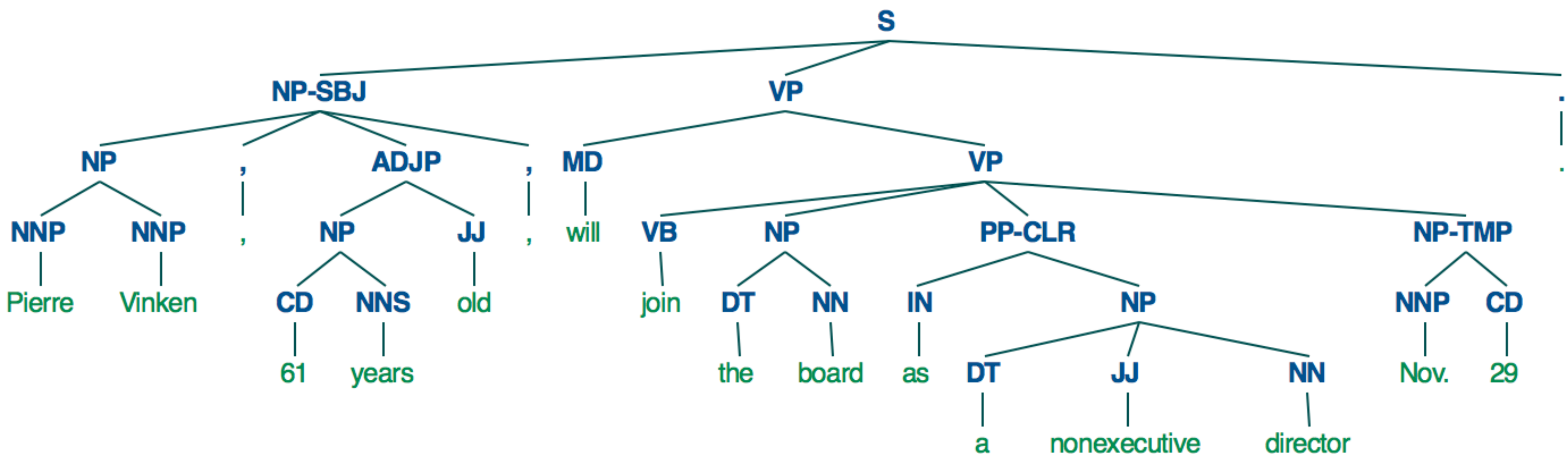
From Toy Grammars to Real Grammars

- Toy grammars with handful of productions good for demonstration or extremely limited domains
- For real texts, we need real grammars
- Many thousands of production rules

Key Constituents in Penn Treebank

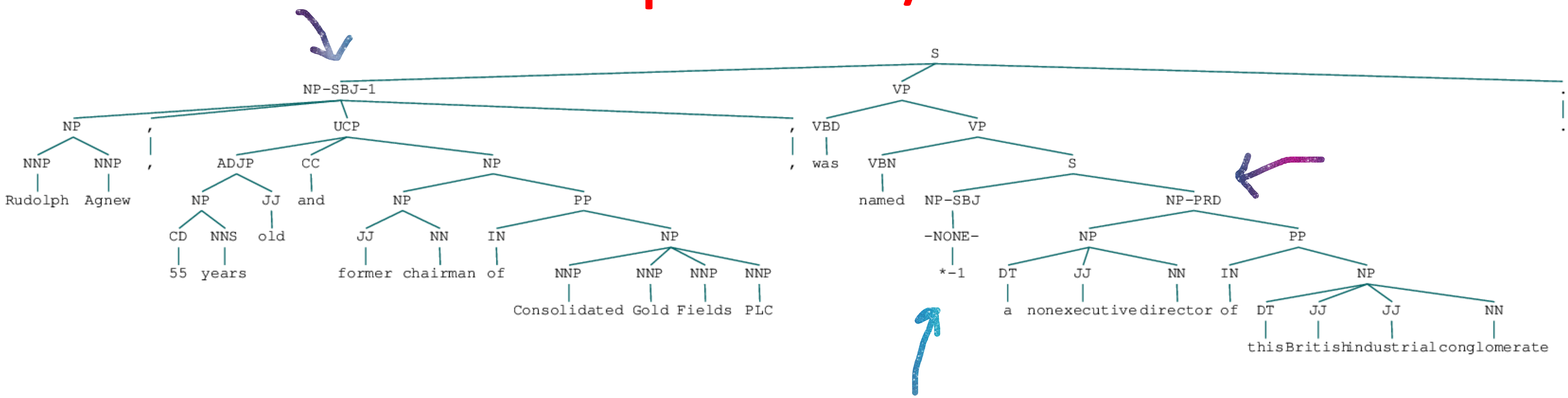
- Sentence (S)
- Noun phrase (NP)
- Verb phrase (VP)
- Prepositional phrase (PP)
- Adjective phrase (AdjP)
- Adverbial phrase (AdvP)
- Subordinate clause (SBAR)

Example PTB/0001




```
( (S
    (NP-SBJ
      (NP (NNP Pierre) (NNP Vinken) )
      (, ,)
      (ADJP
        (NP (CD 61) (NNS years) )
        (JJ old) )
      (, ,) )
    (VP (MD will)
      (VP (VB join)
        (NP (DT the) (NN board) )
        (PP-CLR (IN as)
          (NP (DT a) (JJ nonexecutive) (NN director) ))
        (NP-TMP (NNP Nov.) (CD 29) )))
    (. .) ))
```

Example PTB/0001



- Some parts of PTB trees are often discarded
  - **grammatical roles:** SBJ = subject, PRD = predicate
  - **traces**: In NP-SBJ-1, the “1” is an index, referenced from the *-1 terminal — i.e., the “naming” refers to Rudolf Agnew
- And some structure is added to NPs, which are flat

Basic English Sentence structures

- Declarative sentences (S → NP VP)
  - *The rat ate the cheese*
- Imperative sentences (S → VP)
  - *Eat the cheese!*
- Yes/no questions (S → VB NP VP)
  - *Did the rat eat the cheese?*
- *Wh*-subject-questions (S → WH VP)
  - *Who ate the cheese?*
- *Wh*-object-questions (S → WH VB NP VP)
  - *What did the rat eat?*

English Noun phrases

- Pre-modifiers
  - DT, CD, ADJP, NNP, NN
  - E.g. *the two very best Philly cheese steaks*
- Post-modifiers
  - PP, VP, SBAR
  - A delivery from Bob coming today that I don't want to miss

NP → DT? CD? ADJP? (NN|NNP)+ PP* VP? SBAR?

NP → PRP

Verb Phrases

- Auxiliaries
  - MD, AdvP, VB, TO
  - E.g *should really have tried to wait*

VP → (MD|VB|TO) AdvP? VP

- Arguments and adjuncts
  - NP, PP, SBAR, VP, AdvP
  - E.g *told him yesterday that I was ready*
  - E.g. *gave John a gift for his birthday to make amends*

VP → VB NP? NP? PP* AdvP* VP? SBAR?

Other Constituents

- Prepositional phrase
  - PP → IN NP — *in the house*
- Adjective phrase
  - AdjP → (AdvP) JJ — *really nice*
- Adverb phrase
  - AdvP → (AdvP) RB — *not too well*
- Subordinate clause
  - SBAR → (IN) S — *since I came here*
- Coordination
  - NP → NP CC NP; VP → VP CC VP; etc. — *Jack and Jill*
- Complex sentences
  - S → S SBAR; S → SBAR , S; etc. — *if he goes, I'll go*

A final word

- Context-free grammars can represent linguistic structure
- There are relatively fast dynamic programming algorithms to retrieve this structure
- But what about ambiguity?
  - Extreme ambiguity will slow down parsing
  - If multiple possible parses, which is best?

Required Reading

- J&M3 Ch. 10.1-10.3, 10.5, Ch. 11.1-11.2