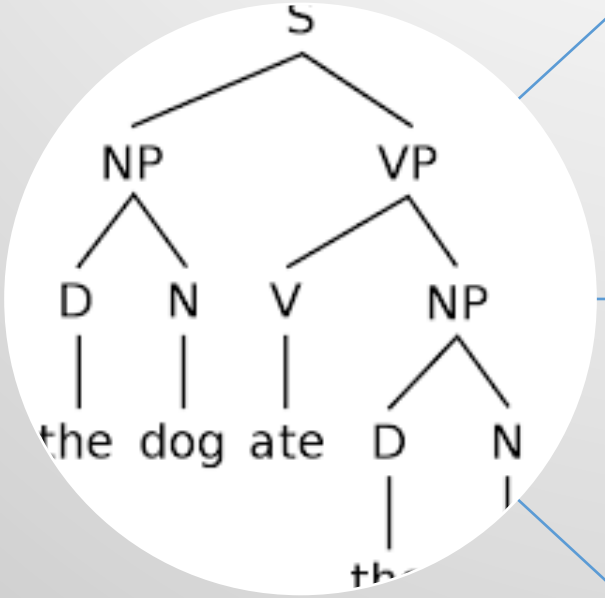
Natural Language Processing

Dr. Karen Mazidi

Part Three: Sentences



Topics

- CFG
- Context-free grammar

Quizzes

- No quiz on this material

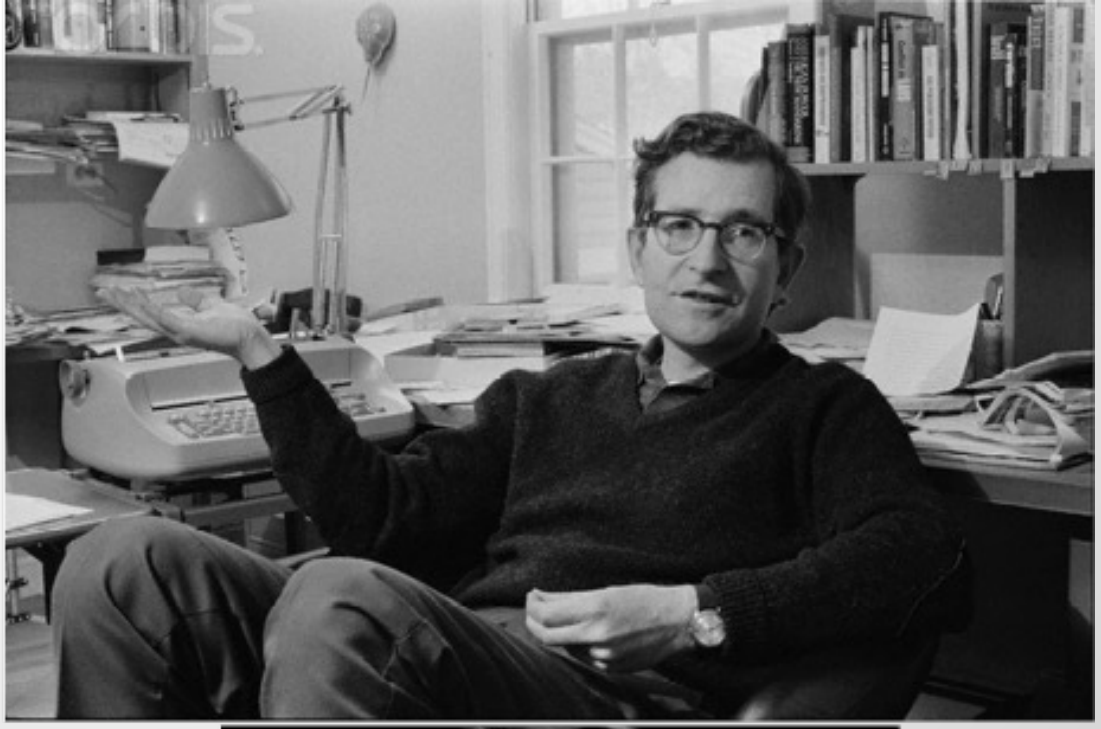
Homework

- No homework on this material

https://plato.stanford.edu/

Noam Chomsky

- American theoretical linguist
- 1950s developed theory that language learning is innate
  - Contrast to ‘tabula rasa’ view of children’s minds
  - Refutation of behaviorists
- 1956 MIT professor
  - Retired as professor emeritus 2002
- https://www.britannica.com/biography/Noam-Chomsky/Linguistics

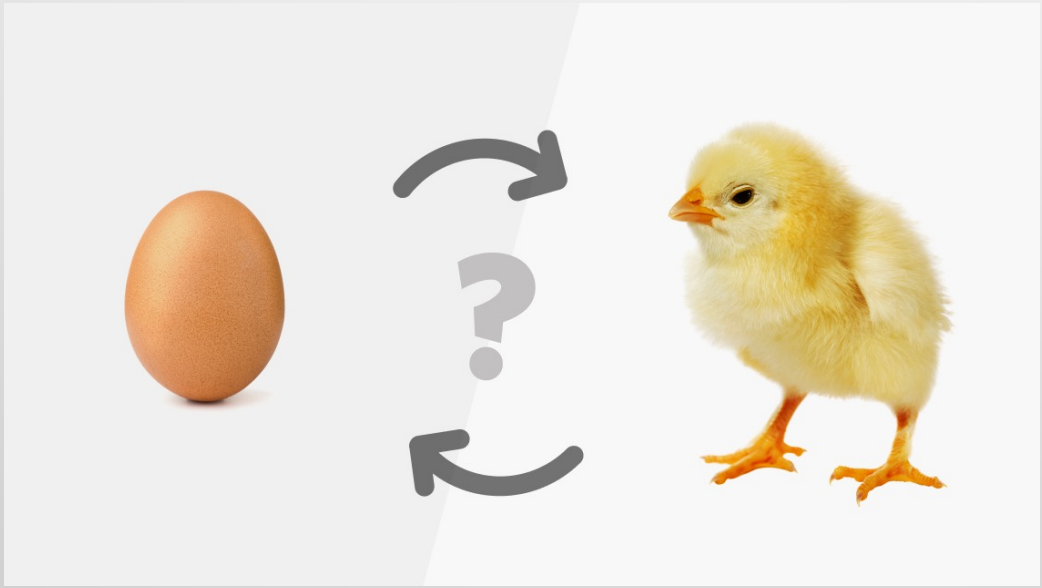


Children and language

- Language is caught not taught
- Evidence for innate structures: Poverty of the stimulus
- What is innate?
  - Principles
  - Parameters
  - Ability to construct a grammar

Universal grammar

- Genetic component of language abilities
- Guides language acquisition in any language



Kinds of rules

- Prescriptive – the way that grammar teachers tell you to create well-formed language; explicitly taught
  - Ex: don't split infinitives; don't end sentence with prep
  - The 'correct' way is often the older way, ex: who/whom
- Descriptive – observations of the way that people use a given language; studied by linguists
  - Induced from observations, 'rules'
  - By observing corpora:
  - What is grammatical? Not?

Example of inducing a rule

- Coordinate structure constraint:
  - Anna read a book.
  - Which book did Anna read?
  - Anna read a book and a newspaper.
  - * Which book did Anna read and a newspaper?
- This rule was never taught explicitly but every native speaker will recognize the problem in the * sentence
- A linguist would have to formalize this into a rule

Children as linguists

- They form these rules by hearing language around them, but the rules remain unconscious
  - No one teaches these rules
  - They all learn in the same path, order
  - Children tend to ignore explicit instruction anyway
  - A list of grammatical sentences is impossible

Sentence components

- A sentence has one or more clauses
  - A clause has a verb and a noun
  - Mary said that John is smart.
- A clause consists of one or more phrases, like NP, VP, PP
- A phrase may contain other components

Constituents

- Sentences consist of constituents which in turn may consist of constituents
- What is the evidence that given words form a constituent:
  - Can the sequence be replaced by a single word?
    - The sad girl looked out the window.
    - She looked out the window.
  - Can the sequence be moved elsewhere in the sentence?
    - John broke the window.
    - The window was broken by John.

Phrases

- Noun phrases (NP): the little dog
- Adjective phrases (JJP): fairly recent
- Prepositional phrases (PP): on the table
- Verb phrases (VP): ran amuck
- Phrases:
  - Have head words that determine type of phrase; a word ‘projects’ a phrase
  - Phrase has no meaning without the head word
  - Head word determines singular/plural
  - Other languages, head word determines: gender

Noun Phrase ambiguity example

- Visiting relatives can be boring.
  - Problem: ‘can’ doesn’t reveal agreement
  - Replace with ‘be’ to clarify ambiguity
- Visiting relatives is boring.
  - ‘Visiting’ is head
- Visiting relatives are boring.
  - ‘relatives’ is head

Noun Phrase substitution

- Can be replaced by a pronoun
- John saw the boy who fed the cats.
- John saw him.

Prepositional phrases

- On the table
- In English the preposition is first, other languages it is a postposition; often called adposition
- Categories of prepositions can be replaced:
  - Locative: at the movies → there
  - Temporal: in 2008 → then

Adjective phrases

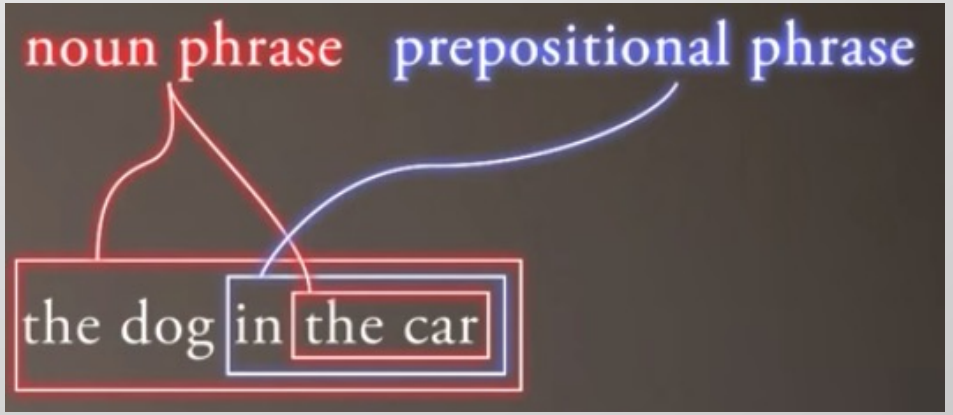
- Mary is extremely ill.
- The noise is too loud to be tolerable.
- Adjective phrases can be used:
  - Predicatively: Mary is ill.
  - Attributively: A very sad Mary cried relentlessly.

Verb phrases

- To insult your mother is disgraceful.
- Jenny will attend the conference.
- “do so” can often be a substitute

Constituent recursion

- A finite number of words and a finite way to combine them form a practically infinite number of sentences
- John regrets that Mary was not invited.
- The sentence contains a sentence.
- The constraints on recursion are not part of language but on practical considerations
- Noun phrase example:



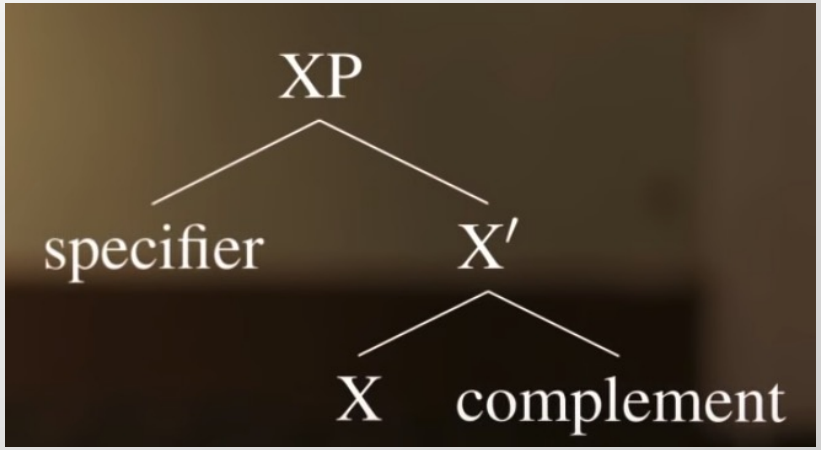
Verbs and complements

- Verbs can be transitive or intransitive
  - John wants ice cream.
  - John left.
  - John left his wallet.

X-bar schema

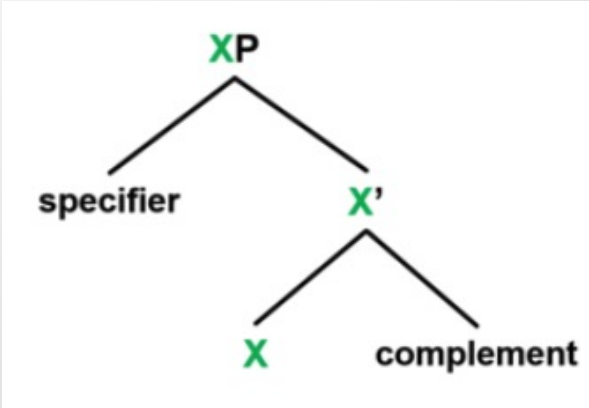
(https://www.youtube.com/watch?v=jgRMBykXg4Q&t=2s)

- Asserts that all phrases have a similar structure
- X is head
- Specifier
- X’ (X bar) contains the complement
- The collection of bicycles
  - Head: collection
  - Complement: bicycles
  - Specifier: the
- The noun projects to the other components
- Specifier and complement can be omitted



X-bar theory

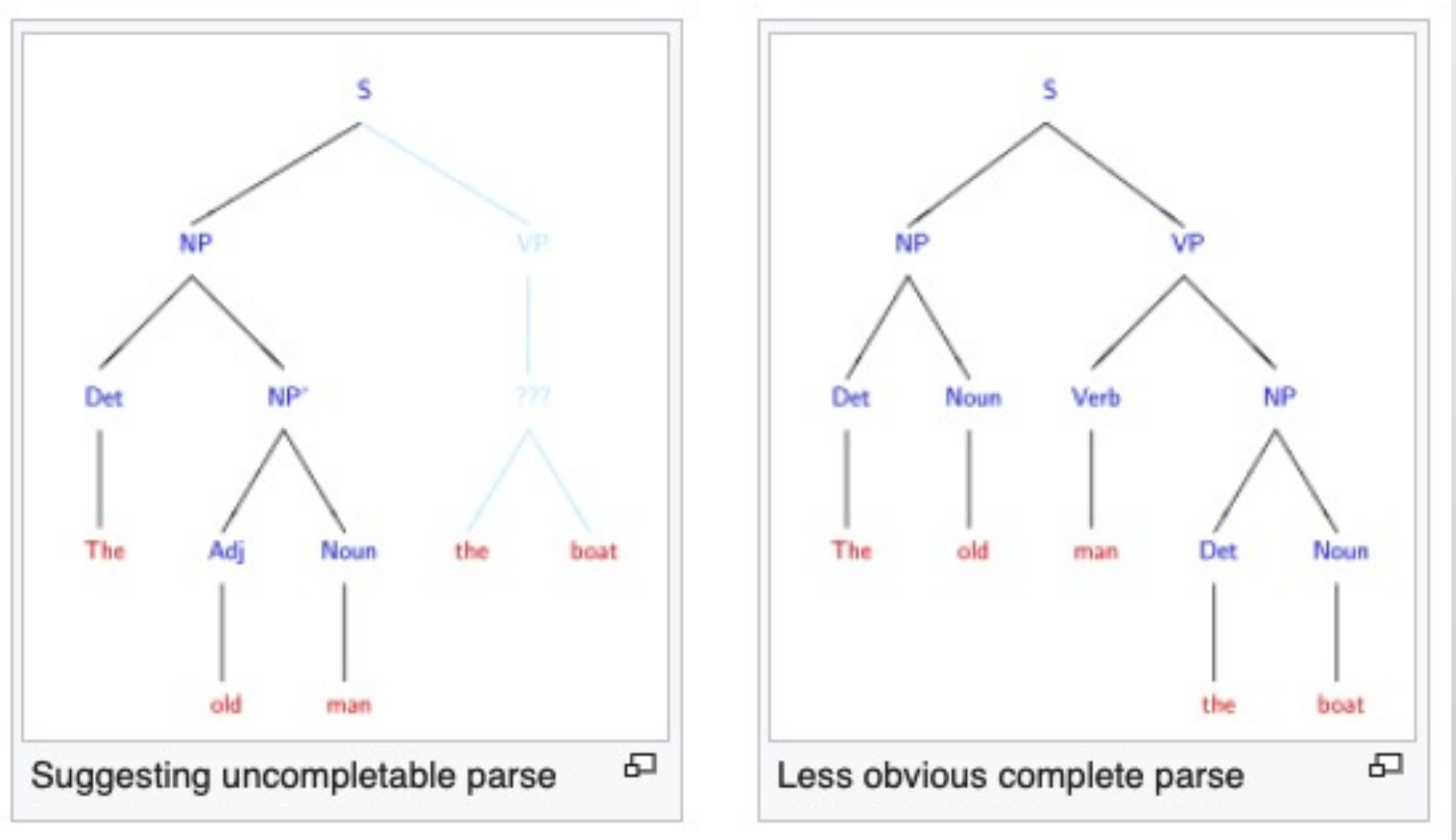
- Lexical items tend to have three levels of structure
  - X - head
  - X’ - complement
  - XP – specifier
- Example: NP under the table
  - Under: X
  - The: specifier
  - Table: complement
- The X’ allows for complex sentences



Garden path sentence

- The horse raced past the barn fell.

Syntax supports semantics



By Jochen Burghardt - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=92742400

Constituency

- A constituent is a group of words that belong together as a unit, examples: NP, VP, etc.
- Constituent structure is hierarchical: constituents within constituents
- Constituents have a head word (not relevant for CFG)

- The book is on the table.
- The carefree girl chased the dancing butterfly.
- She danced her cares away.

Constituents

- Make up a sentence and find all phrases that are constituents
- Discuss with a neighbor
- Any gray areas?

Main types of phrases:

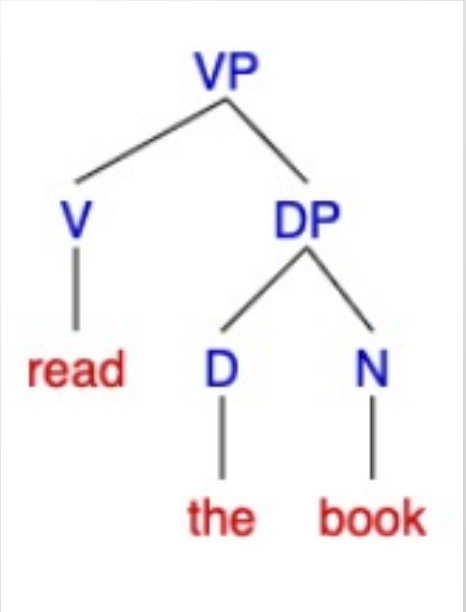
- S sentence
- NP noun phrase (Jack)
- VP verb phrase (jumped)
- PP prepositional phrase (over the candlestick)
- JJP (extremely energetic)
- RBP (very quickly)

CFG Context-Free Grammar

- A formal grammar is a set of rules for rewriting strings to form correct syntax
- A formal grammar consists of:
  - A set of rules
  - A start symbol
- Developed independently by Noam Chomsky 1956 (linguistics) and John Backus 1960s (BNF form computer languages), but also dates back to Panini's rules for Sanskrit (6th-4th C BCE)

CFG context free grammar

- Generative
- Recursive
- Merge – two syntactic structures combine
  - Sentence structure is generated bottom-up
  - Syntactic structure is binary
  - Constituency based
- https://www.youtube.com/watch?v=9JScy7ulDpE&t=6s



CFG – NLP exploration

- Context-free grammar example:
  - Set of rules
  - Start symbol ‘S’
  - Set of terminal symbols (happy, sad, ...)

```
'S'   ->  [['NP', 'VP']],
'NP'  ->  [['DT', 'NN'], ['DT', 'JJ', 'NN']],
'VP'  ->  [['VB'], ['VB', 'PP']],
'PP'  ->  [['P', 'DT', 'NN']],
'JJ'  ->  ['happy', 'sad', 'silly'],
'DT'  ->  ['a', 'the'],
'NN'  ->  ['cat', 'dog', 'clown'],
'VB'  ->  ['plays', 'runs'],
'P'   ->  ['with', 'near', 'beside']
```

CFG

- Can be used to:
  - Generate correct sentences
  - Parse a sentence to check the syntax

```
'S'   ->  [['NP', 'VP']],
'NP'  ->  [['DT', 'NN'], ['DT', 'JJ', 'NN']],
'VP'  ->  [['VB'], ['VB', 'PP']],
'PP'  ->  [['P', 'DT', 'NN']],
'JJ'  ->  ['happy', 'sad', 'silly'],
'DT'  ->  ['a', 'the'],
'NN'  ->  ['cat', 'dog', 'clown'],
'VB'  ->  ['plays', 'runs'],
'P'   ->  ['with', 'near', 'beside']
```

Question

- Generate a random sentence
- Is this sentence valid: A clown plays

```
'S'   ->  [['NP', 'VP']],
'NP'  ->  [['DT', 'NN'], ['DT', 'JJ', 'NN']],
'VP'  ->  [['VB'], ['VB', 'PP']],
'PP'  ->  [['P', 'DT', 'NN']],
'JJ'  ->  ['happy', 'sad', 'silly'],
'DT'  ->  ['a', 'the'],
'NN'  ->  ['cat', 'dog', 'clown'],
'VB'  ->  ['plays', 'runs'],
'P'   ->  ['with', 'near', 'beside']
```

Code Example CFG_1

Key points:

- how to set up production rules
- generating sentences from the rules

Generate

- Notebook CFG_1

```
rules = {'S' : [['NP', 'VP']],
         'NP': [['DT', 'NN'], ['DT', 'JJ', 'NN']],
         'VP': [['VB'], ['VB', 'PP']],
         'PP': [['P', 'DT', 'NN']],
         'JJ' : ['happy', 'sad', 'silly'],
         'DT': ['a', 'the'],
         'NN': ['cat', 'dog', 'clown'],
         'VB': ['plays', 'runs'],
         'P' : ['with', 'near', 'beside']
        }
```

Code 9.1.1 — CFG. Function to generate sentences.

```
# expand function - recursive version
def expand(expanded, expand_me, rules):
    if isinstance(expand_me, list):
        for token in expand_me:
            r = random.randint(0, len(rules[token])-1)
            replacement = rules[token][r]
            expand(expanded, replacement, rules)
    else:  # append the terminal
        expanded.append(expand_me)

    return expanded
```

New rules

- Add production rules:

```
rules['VP'].append(['CP', 'JJ'])
rules['CP'] = ['is']
```

- New sentence types:

```
['the', 'cat', 'is', 'silly']
['the', 'silly', 'clown', 'is', 'sad']
```

A better example

Key points:

- More complex rules
- https://www.youtube.com/watch?v=R_OVyFrBhiU

Code Example CFG_2

Key points:

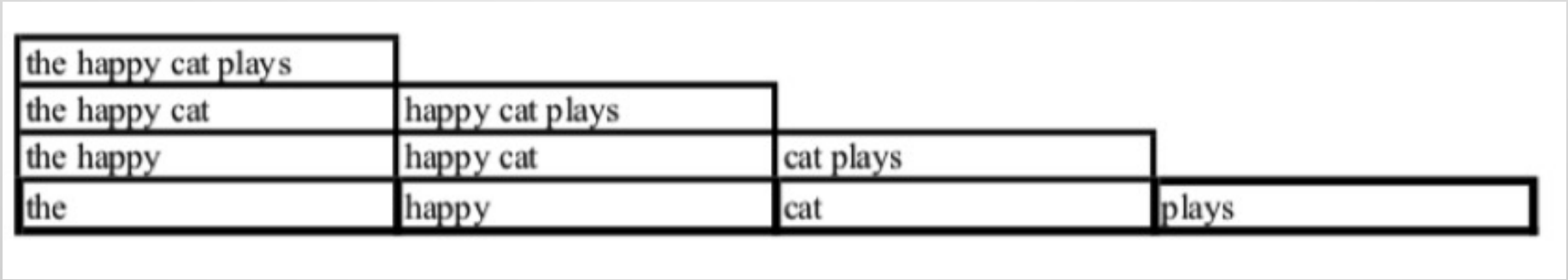
- how to use production rules to check syntax

Parsing

- See online notebook CFG2
- The same production rules used for generation can be used for syntax checking
- CYK algorithm is one approach
  - Bottom-up parsing using dynamic programming
  - Efficient in worst-case scenario: O(n3 * |G|) where G is the size of the grammar

Parsing example

- Parse ‘the happy cat plays’
- Bottom level: tokens
- Moving up: 2, 3, n words at a time



Parsing Round 1

- Assign pos to tokens by looking at production rules
- Possible pos in the list
- the happy cat plays
- [[’DT’], [’JJ’], [’NN’], [’VB’]]

```
'S'   ->  [['NP', 'VP']],
'NP'  ->  [['DT', 'NN'], ['DT', 'JJ', 'NN']],
'VP'  ->  [['VB'], ['VB', 'PP']],
'PP'  ->  [['P', 'DT', 'NN']],
'JJ'  ->  ['happy', 'sad', 'silly'],
'DT'  ->  ['a', 'the'],
'NN'  ->  ['cat', 'dog', 'clown'],
'VB'  ->  ['plays', 'runs'],
'P'   ->  ['with', 'near', 'beside']
```

Parsing Round 2

```
'S'   ->  [['NP', 'VP']],
'NP'  ->  [['DT', 'NN'], ['DT', 'JJ', 'NN']],
'VP'  ->  [['VB'], ['VB', 'PP']],
'PP'  ->  [['P', 'DT', 'NN']],
'JJ'  ->  ['happy', 'sad', 'silly'],
'DT'  ->  ['a', 'the'],
'NN'  ->  ['cat', 'dog', 'clown'],
'VB'  ->  ['plays', 'runs'],
'P'   ->  ['with', 'near', 'beside']
```

- Pos list taken 2 at a time
- No RHS pattern for ‘DT JJ’ (and others) so these are X’d out

the happy cat plays			
the happy cat	happy cat plays		
DT JJ = X	JJ NN = X	NN VB = X	
the : DT	happy : JJ	cat : NN	plays : VB

Figure 9.2: Search for Patterns

Parsing Round 3

- 3 at a time search
- DT JJ NN is replaced with NP

```
'S'   ->  [['NP', 'VP']],
'NP'  ->  [['DT', 'NN'], ['DT', 'JJ', 'NN']],
'VP'  ->  [['VB'], ['VB', 'PP']],
'PP'  ->  [['P', 'DT', 'NN']],
'JJ'  ->  ['happy', 'sad', 'silly'],
'DT'  ->  ['a', 'the'],
'NN'  ->  ['cat', 'dog', 'clown'],
'VB'  ->  ['plays', 'runs'],
'P'   ->  ['with', 'near', 'beside']
```

the happy cat plays			
DT JJ NN = NP	JJ NN VB = X		
DT JJ = X	JJ NN = X	NN VB = X	
the : DT	happy : JJ	cat : NN	plays : VB

Parsing Round 4

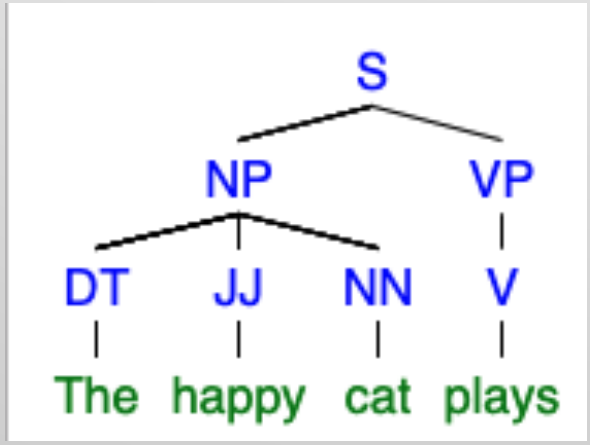
- S → NP VP
- We've reached the start symbol, so we are done

```
'S'   ->  [['NP', 'VP']],
'NP'  ->  [['DT', 'NN'], ['DT', 'JJ', 'NN']],
'VP'  ->  [['VB'], ['VB', 'PP']],
'PP'  ->  [['P', 'DT', 'NN']],
'JJ'  ->  ['happy', 'sad', 'silly'],
'DT'  ->  ['a', 'the'],
'NN'  ->  ['cat', 'dog', 'clown'],
'VB'  ->  ['plays', 'runs'],
'P'   ->  ['with', 'near', 'beside']
```

Bracket Notation

- See: http://mshang.ca/syntree/
- Two forms of same information:
- Bracket notation
- Tree visualization

[S [NP [DT The] [JJ happy] [NN cat]] [VP [V plays]]]



PCFG

- Probabilistic CFG
  - Considers the most likely choice at each step
  - Assigns a probability to each production rule using a corpus like a treebank

Treebanks

- Penn Treebank – expert annotated sentences from the Brown Corpus, Wall Street Journal, etc.
- Contains about 1 million words
- Grammar rules can be extracted from a Treebank:
- Penn has 17K distinct rule types

```
VP -> VB PP                 # go to the store
VP -> VB PP PP             # go to the store in my car
VP -> VB PP PP PP        # go to the store in my car on the tollway
and more . . .
```

Treebank in NLTK

- Examine treebank

```
from nltk.corpus import treebank
treebank.fileids()[:10]
['wsj_0001.mrg',
 'wsj_0002.mrg',
 'wsj_0003.mrg',
. . .
```

Code 9.6.1 — Load the treebank. Look at words and POS

```
print(treebank.words('wsj_0003.mrg'))
 ['A', 'form', 'of', 'asbestos', 'once', 'used', '*', ...]

print(treebank.tagged_words('wsj_0003.mrg'))
[('A', 'DT'), ('form', 'NN'), ('of', 'IN'), ...]
```

Examine sentence

- Code:

```
print(treebank.parsed_sents('wsj_0003mrg')[0])
```

```
(S
  (S-TPC-1
    (NP-SBJ
      (NP (NP (DT A) (NN form)) (PP (IN of) (NP (NN asbestos))))
      (RRC
        (ADVP-TMP (RB once))
        (VP
          (VBN used)
          (NP (-NONE- *))
          (S-CLR
            (NP-SBJ (-NONE- *))
            (VP
              (TO to)
              (VP
                (VB make)
                (NP (NNP Kent) (NN cigarette) (NNS filters))))))))))
    (VP
      (VBZ has)
      (VP
        (VBN caused)
        (NP
          (NP (DT a) (JJ high) (NN percentage))
          (PP (IN of) (NP (NN cancer) (NNS deaths)))
          (PP-LOC
            (IN among)
            (NP
              (NP (DT a) (NN group))
              (PP
                (IN of)
                (NP
                  (NP (NNS workers))
                  (RRC
                    (VP
                      (VBN exposed)
                      (NP (-NONE- *))
                      (PP-CLR (TO to) (NP (PRP it)))
                      (ADVP-TMP
                        (NP
                          (QP (RBR more) (IN than) (CD 30))
                          (NNS years))
                        (IN ago)))))))))))))
    (, ,)
    (NP-SBJ (NNS researchers))
    (VP (VBD reported) (SBAR (-NONE- 0) (S (-NONE- *T*-1))))
    (. .))
```

Recursion in language

```
Rules with recursion:
NP -> NN
NP -> Noun PP
PP -> Prep NP
```

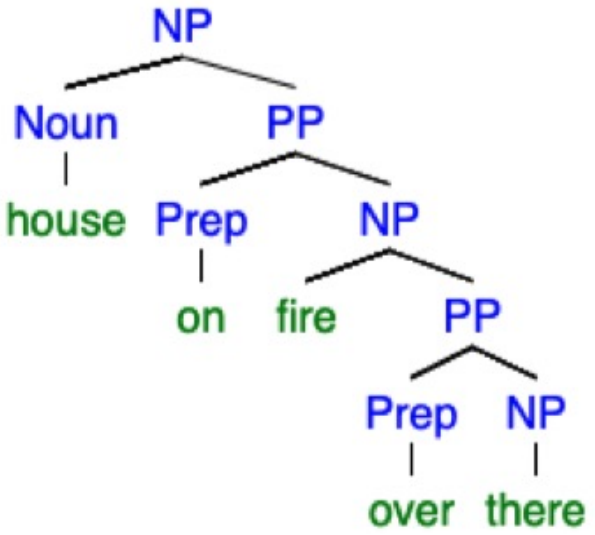


Figure 9.5: Recursive NP

Chunking

- The task of chunking divides a sentence into phrases
- NLTK conll2000 corpus (Conference on Computational Natural Language Learning)

```
from nltk.corpus import conll2000
for tree in conll2000.chunked_sents()[:2]:
    print(tree)
```

```
(S
  Chancellor/NNP
  (PP of/IN)
  (NP the/DT Exchequer/NNP)
  (NP Nigel/NNP Lawson/NNP)
  (NP 's/POS restated/VBN commitment/NN)
  (PP to/TO)
  (NP a/DT firm/NN monetary/JJ policy/NN)
  (VP has/VBZ helped/VBN to/TO prevent/VB)
  (NP a/DT freefall/NN)
  (PP in/IN)
  (NP sterling/NN)
  (PP over/IN)
  (NP the/DT past/JJ week/NN)
  ./.)
```

Essential points to note

- CFG exists in the space between linguistics and NLP
- NLP has moved on to neural network methods for learning about sentence patterns
- CFG still useful for defining/compiling programming languages

Next topic

syntax parsers