English is a STEM subject

tinyurl.com/SigmaCFG

Normative grammar is lame

- Normative grammar is concerned with the rules that make sentences “right” or “wrong”, developing a uniform “standard” language.
- These rules often don’t match spoken language varieties.
  - *Bob and me played Pokemon yesterday.
  - Bob and I played Pokemon yesterday.
  - *There’s a lot of people here.
  - There are a lot of people here.
  - The cat that the dog that the man fed chased meowed

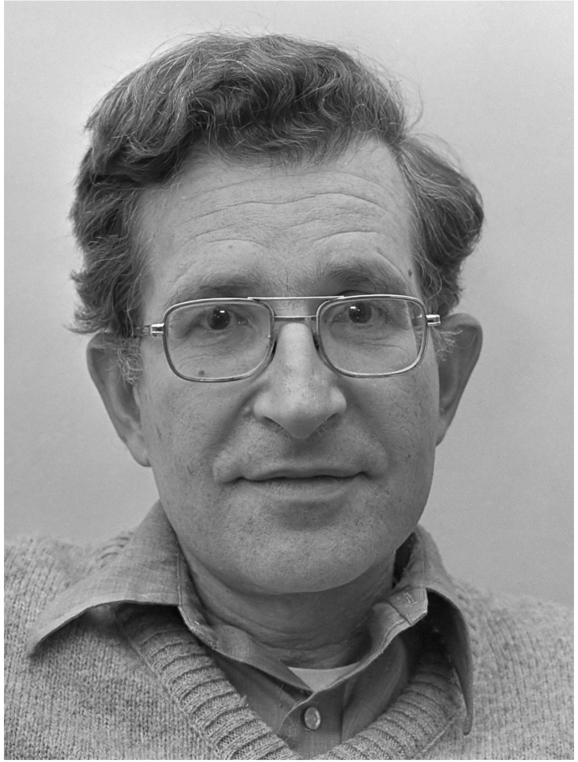
Descriptive grammar doesn’t make rules (kind of)

- Descriptive grammar aims at describing how a language actually behaves, and how speakers perceive sentences as ill-formed or not.
- Linguists are often concerned with descriptive grammars.
  - Yous are going to the party?
  - *Yous are went to the party?
  - There’s a lot of people here.
  - *A lot are of people here.
  - *The cat that the dog that the man fed chased meowed.

Rules seem to be natural to language

- Speakers still perceive some new sentences as ill-formed. Why?
    - *This sentence ill-formed.
- Speakers can say novel, well-formed sentences. How?
    - This sentence is well-formed.
- The number of well-formed sentences is seemingly infinite. How?
    - This well-formed sentence was made for Sigma Camp 2025.
    - This well-formed sentence was made for Sigma Camp 2026.
    - This well-formed sentence was made for Sigma Camp 2027.
- → Generative grammar rules the structure (syntax) of sentences

Grammar is seemingly natural and biological

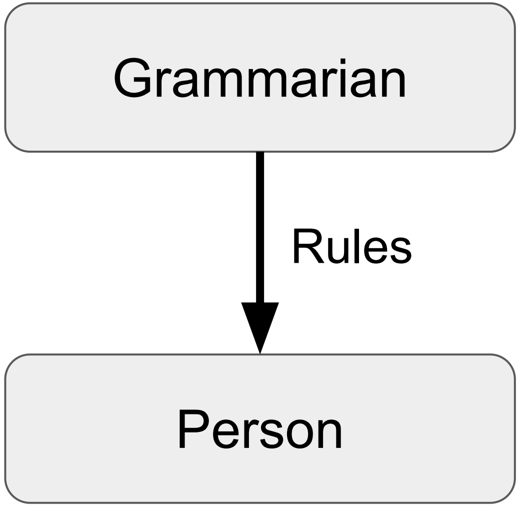


- Average Joe can learn a language, but his dog can't.
- Children learn languages despite a poverty of stimulus.
- Languages are seemingly too complex to be learned so quickly

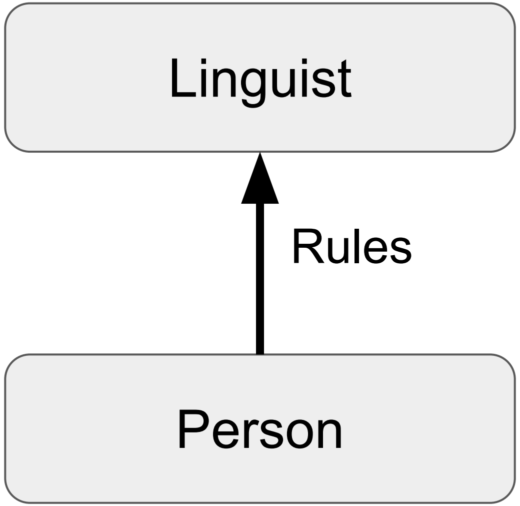
- → (simple) Universal grammar

Summary: normative vs generative grammar

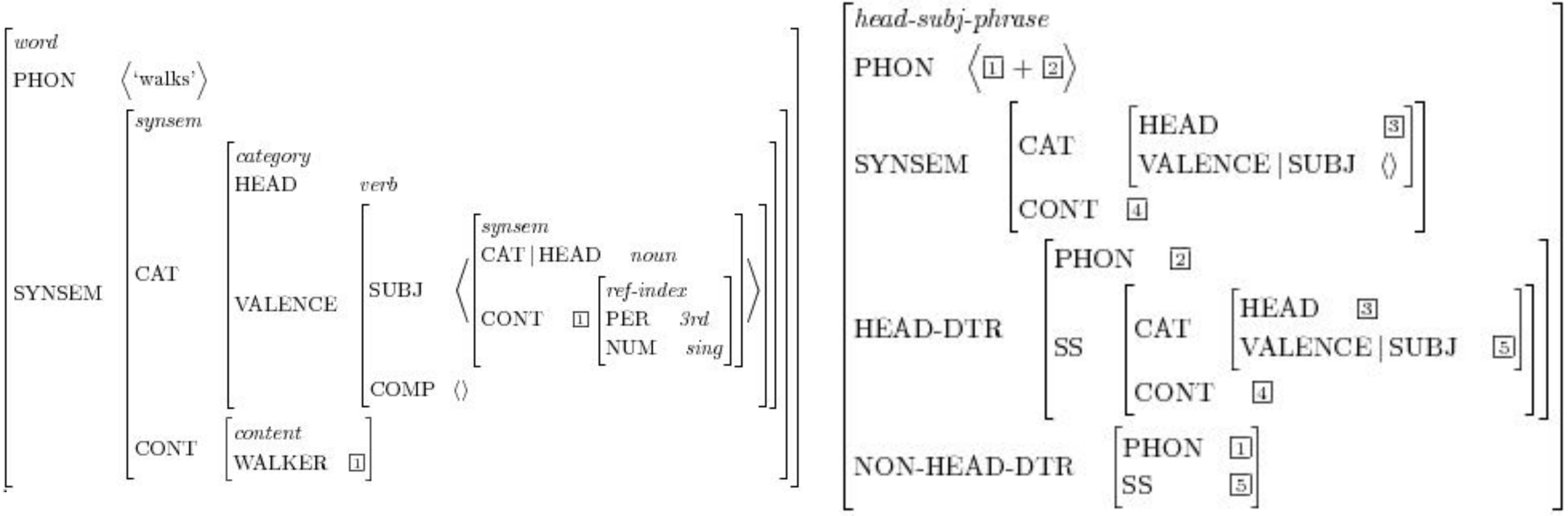
<u>Normative grammar</u>



<u>Generative grammar</u>



Comprehensive generative grammars can become weird...



HSPG feature structure by Lizmarie, Public domain, via Wikimedia Commons

Context-free grammars

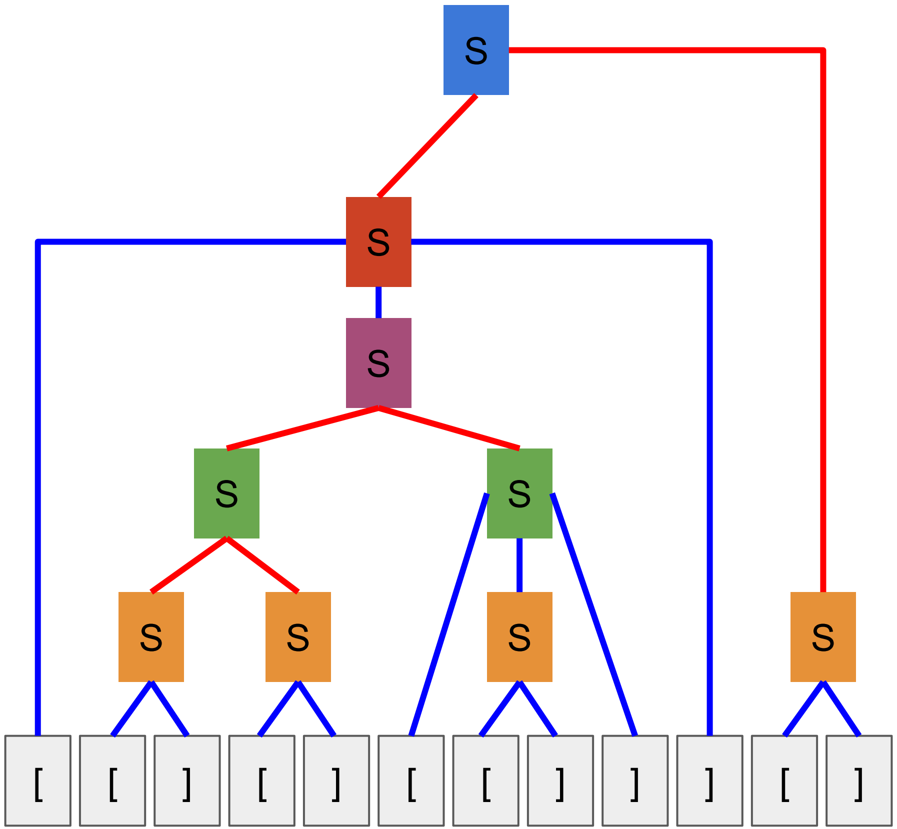
Lexicon is the dictionary, grammar is the rule

- A context-free grammar is composed of
  - A lexicon corresponding to a set of words with their classes
    - E.g. Noun → “dog”
  - A grammar consisting of rules of substitution, with start symbol S
    - E.g. Noun phrase → Article Adjective Noun
- Any sentence that can be generated (or parsed) by the grammar is well-formed with respect to it, and ill-formed otherwise.

(Parsing means to show how a sentence can be predicted by the rules)

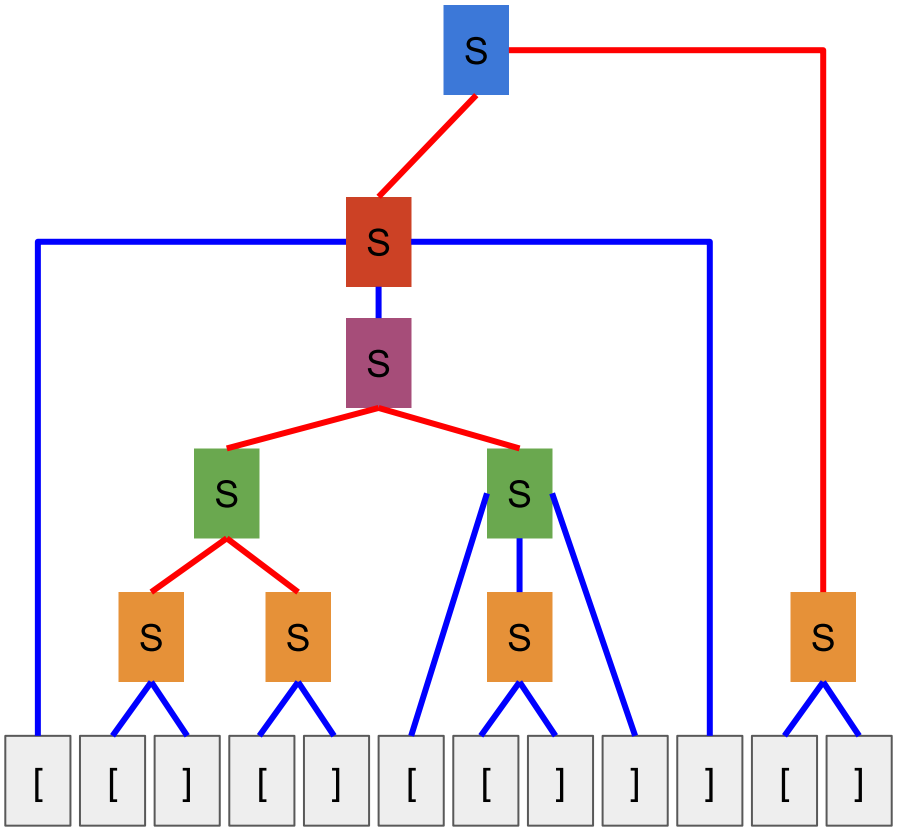
Example: brackets

- Lexicon: { [,] }
- Grammar:
  - S → S⁺ S
  - S → [(S)]
- Example sentences:
  - [ ]
  - [ ] [ ]
  - [ ] [ ] [ ] [ ]
  - *[ ] [ ] [ ] [ ] [
  - [[ ] [ ] [[ ]]] [ ]



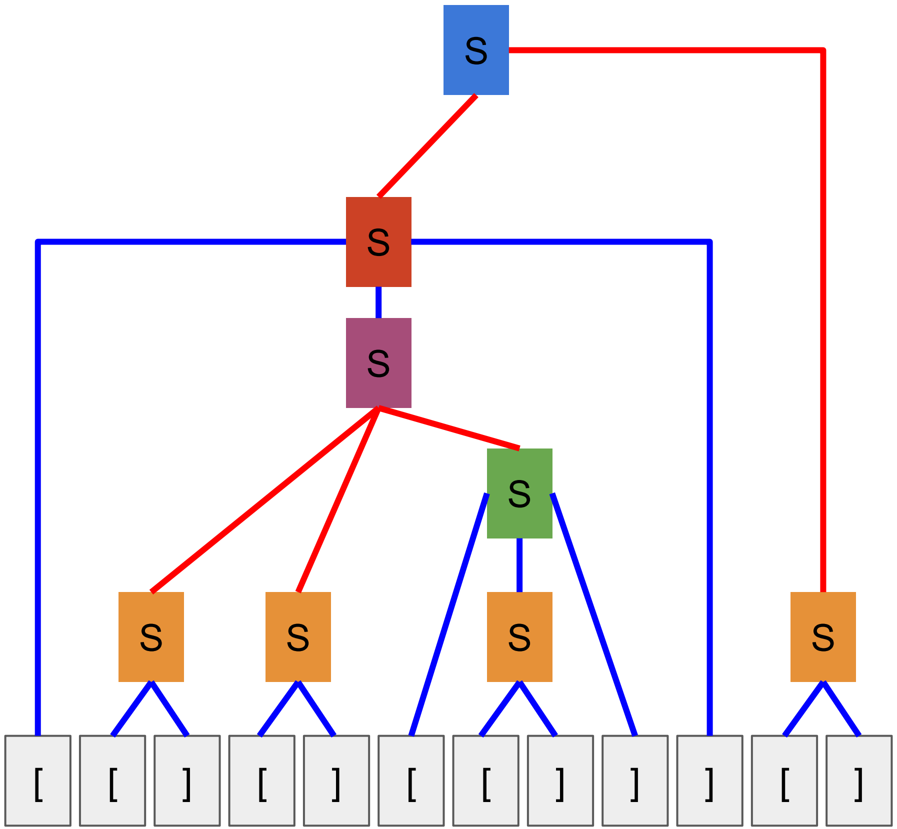
Sisters side-by-side, parent dominates

- If a symbol X is directly above Y, then:
  - Y is the *daughter* of X
  - X *dominates* Y
  - X is the *parent* of Y
- Two symbols with the same parent are *sisters*



Example: brackets // ambiguity

- CFGs can be complex enough to convey ambiguity!
- Ambiguity shows up as different tree structures parsing the same sentence.
- Real languages are ambiguous, so this really is an advantage.



Example: brackets // the code

- As you can imagine, CFG sentence generation can easily be implemented by code
- Parsing is a slightly harder problem, and may need the rules to be re-written in simpler terms at the expense of conciseness
- Python code for generating well-formed bracket sentences:

```
def bracket(depth):
    S = ""
    if depth == 0:
        return "[ ]"
    if randint(0,1) % 2 == 0:
        #first rule
        length = randint(2, 3)
        for i in range(length):
            S += bracket(depth - 1)
    else:
        #second rule
        S = " [ " + bracket(depth - 1) + " ] "
    return S
```

Example: English

- Before jumping into English CFGs, we need to decide which types of words and phrases exist.
- It is useful to separate things into nonterminal and terminal symbols

Abstract classes:
NP → Article + Noun

Lexical entries:
“dog”, “see”, “an”

Example: English // terminal symbols

- Terminal symbols can be grouped into standard grammatical classes
  - Noun = N → “cat” | “woman” | “telescope” | “house”
  - Adjective = Adj → “quick” | “slow”
  - Article = Art → “the” | “a”
  - Preposition = Pr → “with” | “in”
  - Verb = V → “loves” | “sees” | “runs” | “walks”
  - Adverb = Adv → “quickly” | “slowly”
  - Conjunction = Conj → “and”

Example: English // nonterminal symbols

- Nonterminal symbols are less trivial. One can think of them as more complex, compound versions of the terminal symbols.
  - Noun phrase = NP → Art Adj* N
  - Verb phrase = VP → V (NP) (Adv)
  - Prepositional phrase = PP → Pr NP
- They can also refer to each other
  - VP → V PP
  - NP → NP PP
  - NP → NP Conj NP
  - S → NP VP

a slow woman runs a quick house slowly

- N → “cat” | “woman” | “telescope” | “house”
- Adj → “quick” | “slow”
- Art → “the” | “a”
- Pr → “with” | “in”
- V → “loves” | “sees” | “runs” | “walks”
- Adv → “quickly” | “slowly”
- Conj → “and”

- NP → Art Adj* N
- NP → NP PP
- NP → NP Conj NP
- VP → V (NP) (PP) (Adv)
- PP → Pr NP

Example: English // code

- The code for our grammar is a bit long, but can also be easily implemented to generate “valid” sentences!
- Python code

```python
from random import randint
lexicon = {"N":("cat ", "woman ", "telescope ", "house "), "Adj":("quick ", "slow "), "Art":("the ", "a
def sel(c):
    return lexicon[c][randint(0,len(lexicon[c])-1)]
def PP(depth):
    if depth == 0:
        return ""
    return sel("Pr") + NP(depth - 1)
def NP(depth):
    rule = randint(0, 2)
    if rule == 0 or depth == 0:
        return sel("Art") + randint(0,1)*sel("Adj") + randint(0, 1)*sel("Adj") + sel("N")
    elif rule == 1:
        return NP(depth - 1) + PP(depth-1)
    elif rule == 2:
        return NP(depth - 1) + sel("Conj") + NP(depth - 1)
def VP(depth):
    return sel("V") + randint(0,1)*NP(depth-1) + randint(0,1)*PP(depth - 1) + randint(0, 1)*sel("Adv")
def S(depth):
    return NP(depth - 1) + VP(depth - 1)
print(S(3))
```

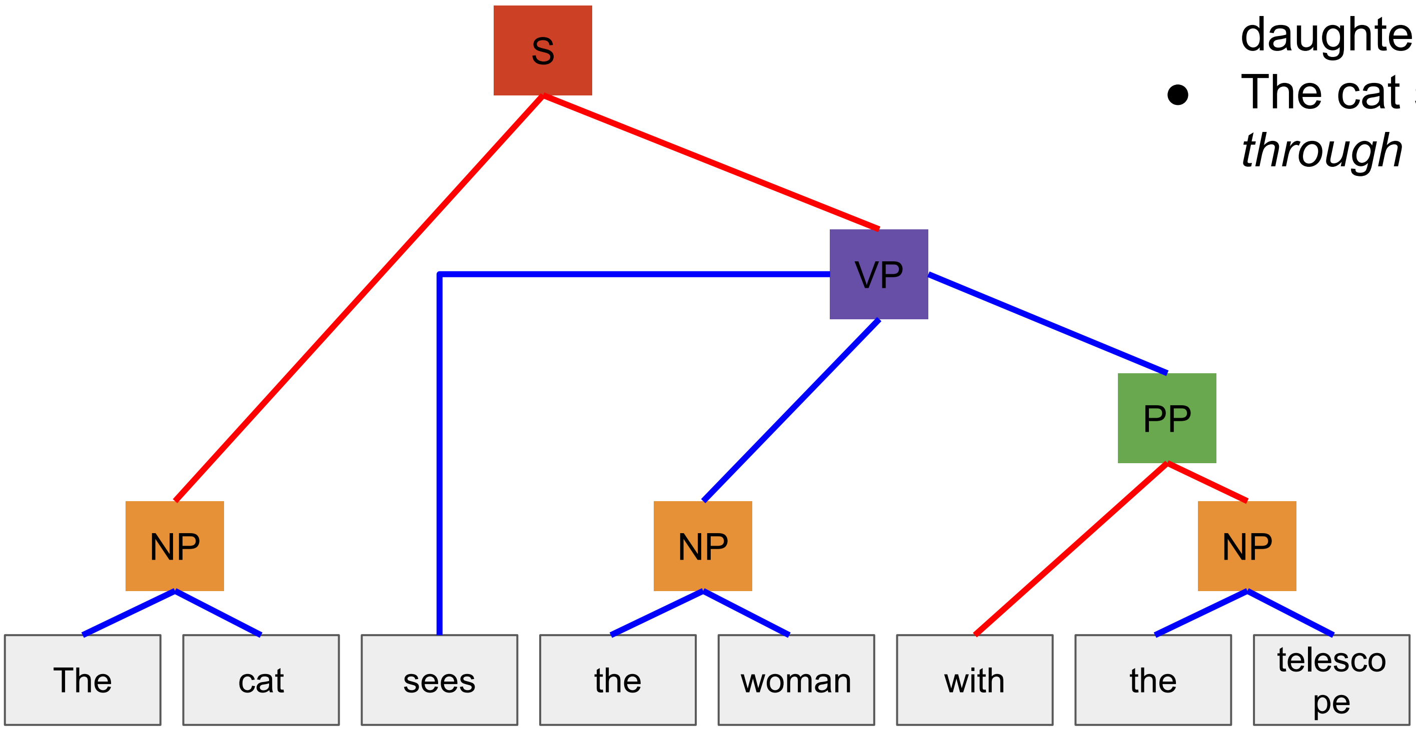
Example: English // sample nice phrases

- A cat walks quickly
- The slow cat runs quickly
- The quick cat runs quickly
- The woman with the telescope loves the cat
- The woman walks in the telescope
- The woman walks the quick cat and the slow cat in the house

- N → “cat” | “woman” | “telescope” | “house”
- Adj → “quick” | “slow”
- Art → “the” | “a”
- Pr → “with” | “in”
- V → “loves” | “sees” | “runs” | “walks”
- Adv → “quickly” | “slowly”
- Conj → “and”

- NP → Art Adj* N
- NP → NP PP
- NP → NP Conj NP
- VP → V (NP) (PP) (Adv)
- PP → Pr NP
- S → NP VP

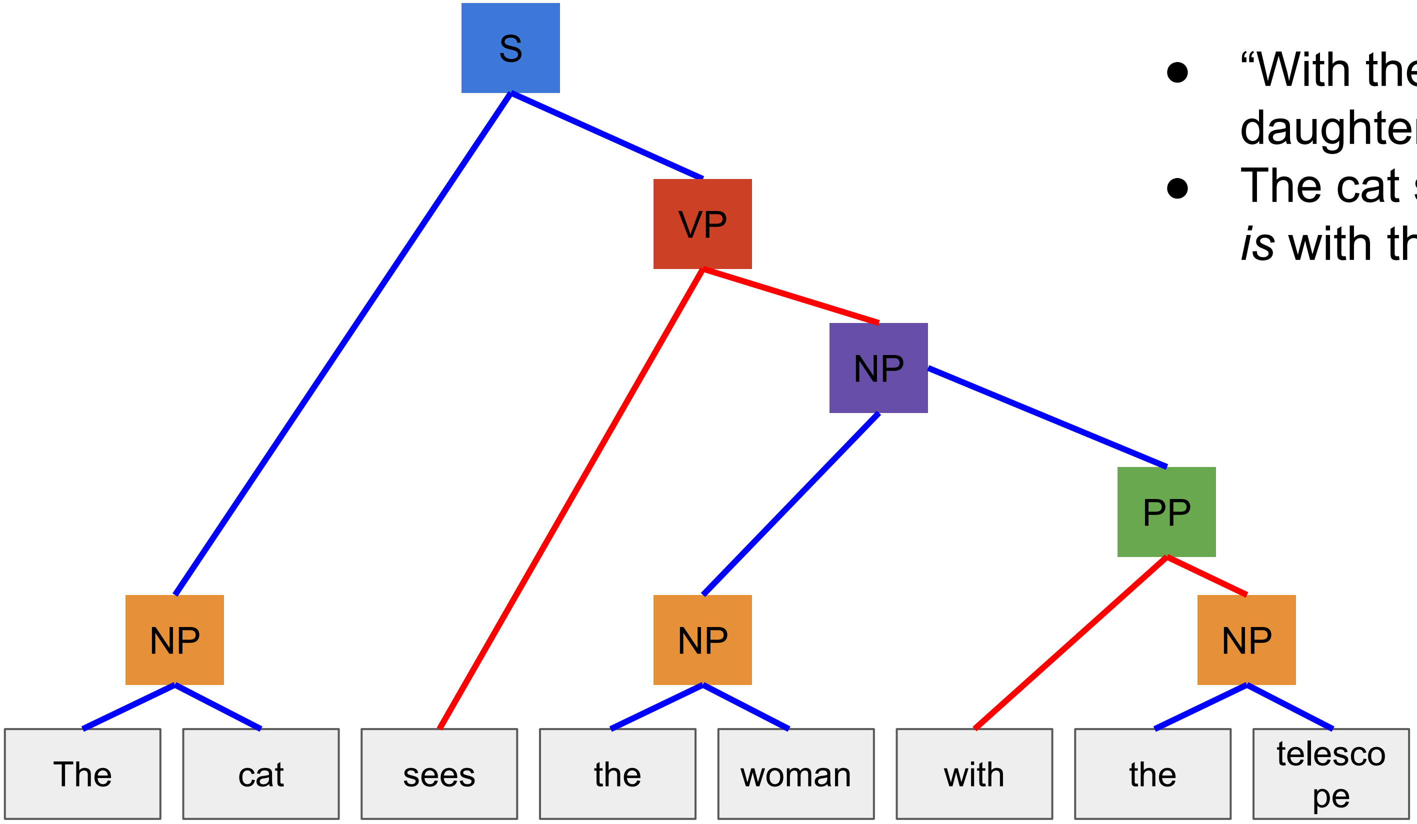
Example: English // ambiguity



- "With the telescope" is a daughter of VP
- The cat sees the woman *through* the telescope

- NP → Art Adj* N
- NP → NP PP
- NP → NP Conj NP
- VP → V (NP) (PP) (Adv)
- PP → Pr NP
- S → NP VP

Example: English // ambiguity



- “With the telescope” is a daughter of NP
- The cat sees the woman, *who is* with the telescope

- NP → Art Adj* N
- NP → NP PP
- NP → NP Conj NP
- VP → V (NP) (PP) (Adv)
- PP → Pr NP
- S → NP VP

Example: English // sample wrong phrases

- *The slow cat
- *Cat walks quickly
- *The woman sees the cat the telescope
- *The cat slowly loves the woman and the telescope
- *The woman walks quickly in the house

- N → “cat” | “woman” | “telescope” | “house”
- Adj → “quick” | “slow”
- Art → “the” | “a”
- Pr → “with” | “in”
- V → “loves” | “sees” | “runs” | “walks”
- Adv → “quickly” | “slowly”
- Conj → “and”

- NP → Art Adj* N
- NP → NP PP
- NP → NP Conj NP
- VP → V (NP) (PP) (Adv)
- PP → Pr NP
- S → NP VP

Example: English // our grammar is imperfect :(

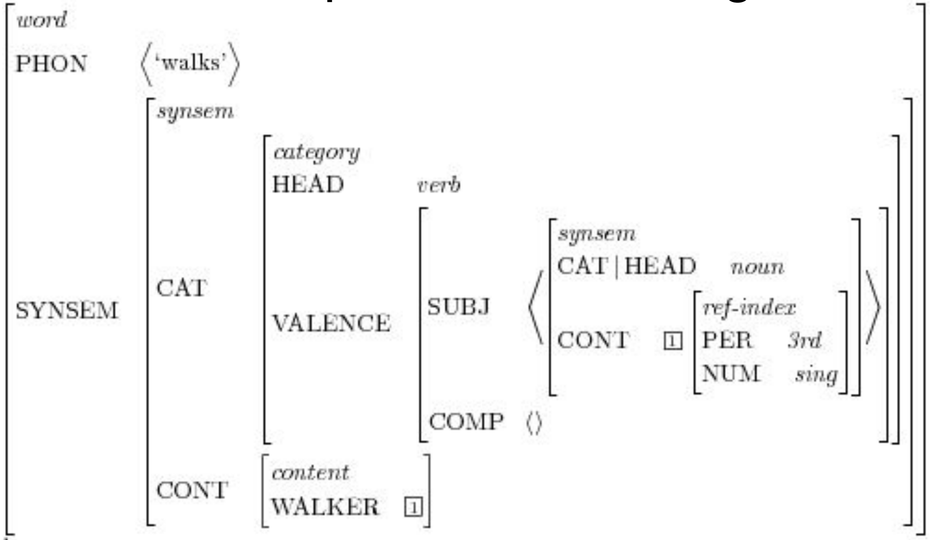
- *Cat walks quickly
- The woman loves
- The woman and the cat walks slowly
- *The slow cat walks slowly and the quick cat runs quickly
- *The cats love the telescope

- N → “cat” | “woman” | “telescope” | “house”
- Adj → “quick” | “slow”
- Art → “the” | “a”
- Pr → “with” | “in”
- V → “loves” | “sees” | “runs” | “walks”
- Adv → “quickly” | “slowly”
- Conj → “and”

- NP → Art Adj* N
- NP → NP PP
- NP → NP Conj NP
- VP → V (NP) (PP) (Adv)
- PP → Pr NP
- S → NP VP

CFGs are insufficient for linguistics

- An attempt to make CFGs match natural languages soon makes them exceedingly convoluted and redundant
- More advanced theories are needed to model them well
  - Example: Head-driven phrase structure grammars (HSPGs)



Computer science can be helped by linguistics

- Linguistic insight can help with natural language processing (both generation and parsing)
- Generative grammars are also mathematical objects that can be studied on their own right, without reference to human languages
  - Often, the design of a programming language and of its compiler goes through many linguistic considerations.
  - For example, it is useful to design a language that avoids ambiguity.
  - It is also useful to design a compiler that “solves” the ambiguity!

Your turn!

Write a CFG capable of generating valid haikus

A valid haiku has three lines, with syllables following the 5-7-5 pattern.

Example:

An old silent pond . . .

A frog jumps into the pond,

splash! Silence again.