LING/C SC 581:

Advanced Computational Linguistics

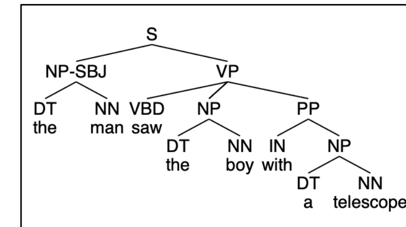
Lecture 25

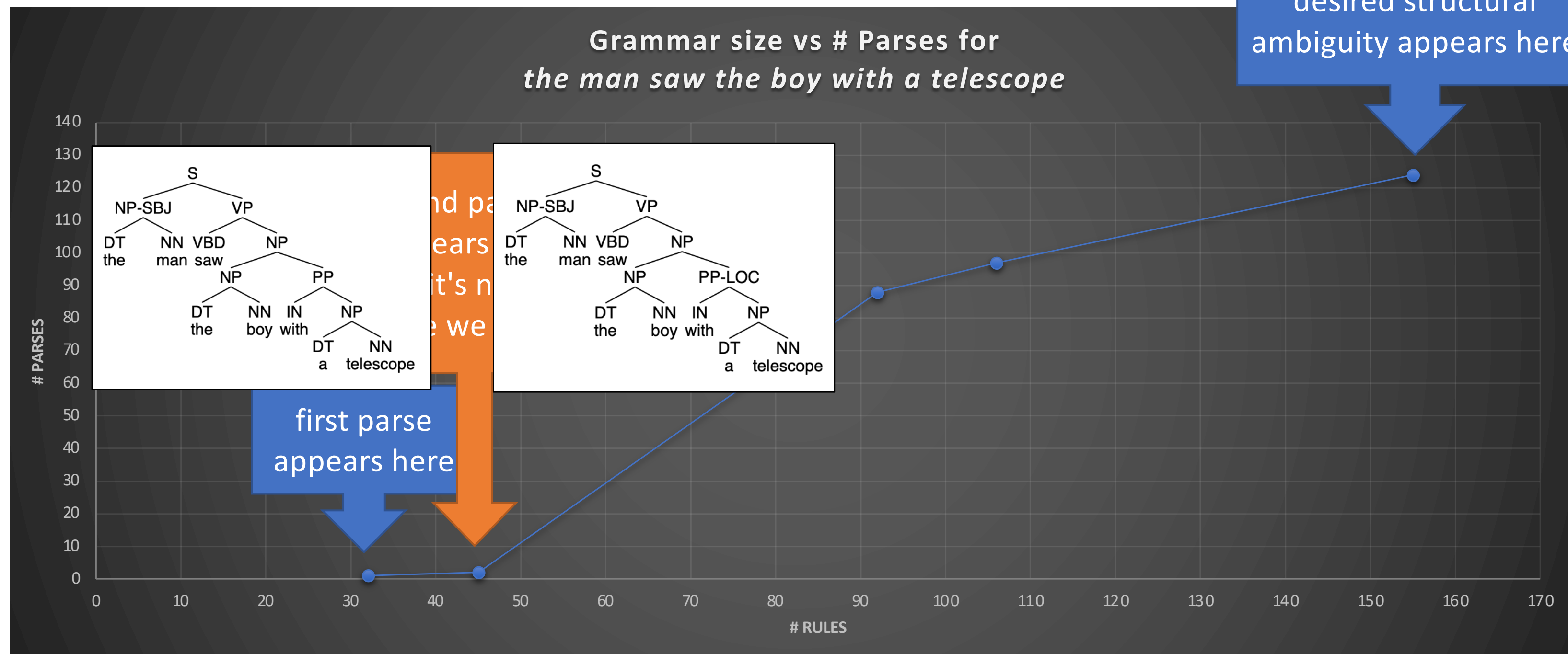
Today's Topic

- Explaining the code used last time for the experiment:
  - *Let’s look inside the file*
  - `find_scfg.py`

- Homework 11

Smallest grammar for a sentence





find_scfg.py

- Setup:

```
ps = [p for t in ptb.parsed_sents() for p in t.productions()]
print(f"Productions: {len(ps):,}")
```

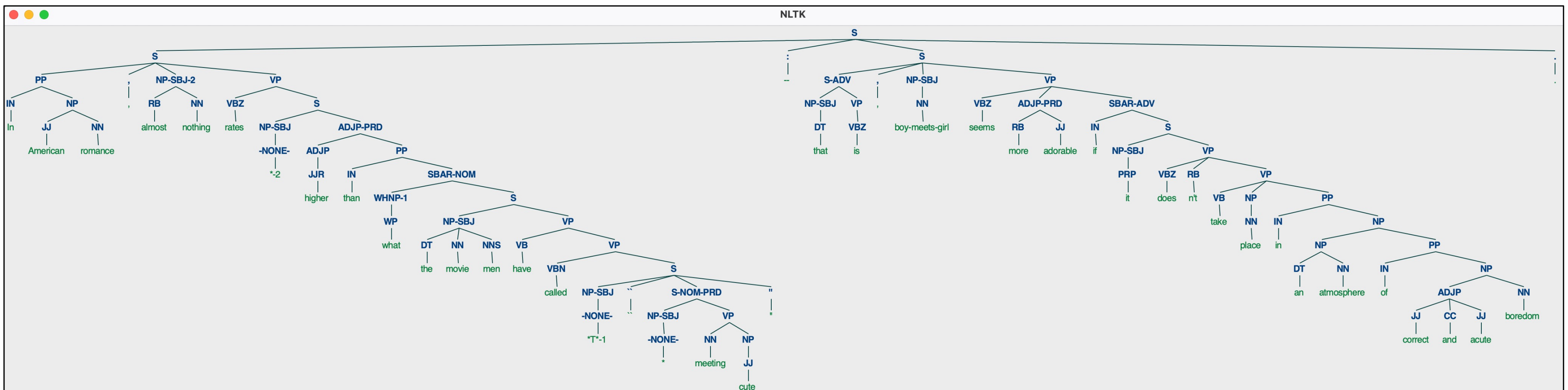
```
Productions: 3,131,242
```

```
>>> ps[:30]
```

```
[S -> S : S ., S -> PP , NP-SBJ-2 VP, PP -> IN NP, IN -> 'In', NP -
 > JJ NN, JJ -> 'American', NN -> 'romance', , -> ',', NP-SBJ-2 ->
 RB NN, RB -> 'almost', NN -> 'nothing', VP -> VBZ S, VBZ ->
 'rates', S -> NP-SBJ ADJP-PRD, NP-SBJ -> -NONE-, -NONE- -> '*-2',
 ADJP-PRD -> ADJP PP, ADJP -> JJR, JJR -> 'higher', PP -> IN SBAR-
 NOM, IN -> 'than', SBAR-NOM -> WHNP-1 S, WHNP-1 -> WP, WP ->
 'what', S -> NP-SBJ VP, NP-SBJ -> DT NN NNS, DT -> 'the', NN ->
 'movie', NNS -> 'men', VP -> VB VP]
```

find_scfg.py

```
>>> ptb.parsed_sents()[0].draw()
```



find_scfg.py

```
>>> ptb.parsed_sents()[0]
Tree('S', [Tree('S', [Tree('PP', [Tree('IN', ['In']), Tree('NP', [Tree('JJ', ['American']), Tree('NN',
  ['romance'])])]), Tree(',', [',']), Tree('NP-SBJ-2', [Tree('RB', ['almost']), Tree('NN',
  ['nothing'])]), Tree('VP', [Tree('VBZ', ['rates']), Tree('S', [Tree('NP-SBJ', [Tree('-NONE-', ['*-
  2'])]), Tree('ADJP-PRD', [Tree('ADJP', [Tree('JJR', ['higher'])]), Tree('PP', [Tree('IN', ['than']),
  Tree('SBAR-NOM', [Tree('WHNP-1', [Tree('WP', ['what'])]), Tree('S', [Tree('NP-SBJ', [Tree('DT',
  ['the']), Tree('NN', ['movie']), Tree('NNS', ['men'])]), Tree('VP', [Tree('VB', ['have']), Tree('VP',
  [Tree('VBN', ['called']), Tree('S', [Tree('NP-SBJ', [Tree('-NONE-', ['*T*-1'])]), Tree('``', ['``']),
  Tree('S-NOM-PRD', [Tree('NP-SBJ', [Tree('-NONE-', ['*'])]), Tree('VP', [Tree('NN', ['meeting']),
  Tree('NP', [Tree('JJ', ['cute'])])])])]), Tree("''", ["''"])])])])])])])])])])])]), Tree(':', ['--']),
  Tree('S', [Tree('S-ADV', [Tree('NP-SBJ', [Tree('DT', ['that'])]), Tree('VP', [Tree('VBZ',
  ['is'])])]), Tree(',', [',']), Tree('NP-SBJ', [Tree('NN', ['boy-meets-girl'])]), Tree('VP',
  [Tree('VBZ', ['seems']), Tree('ADJP-PRD', [Tree('RB', ['more']), Tree('JJ', ['adorable'])]),
  Tree('SBAR-ADV', [Tree('IN', ['if']), Tree('S', [Tree('NP-SBJ', [Tree('PRP', ['it'])]), Tree('VP',
  [Tree('VBZ', ['does']), Tree('RB', ["n't"]), Tree('VP', [Tree('VB', ['take']), Tree('NP', [Tree('NN',
  ['place'])]), Tree('PP', [Tree('IN', ['in']), Tree('NP', [Tree('NP', [Tree('DT', ['an']), Tree('NN',
  ['atmosphere'])]), Tree('PP', [Tree('IN', ['of']), Tree('NP', [Tree('ADJP', [Tree('JJ', ['correct']),
  Tree('CC', ['and']), Tree('JJ', ['acute'])]), Tree('NN', ['boredom'])])])])])])])])])])])]), Tree('.',
  ['.'])])
```

find_scfg.py

- Divides the productions into lexical rules and syntax rules
- Organizes them by frequency of occurrence

```
with concurrent.futures.ThreadPoolExecutor(max_workers=4) as executor:
    executor.submit(make_lexicalrules, ps)
    executor.submit(make_syntaxrules, ps)
```

- make_lexicalrules

```
["NNP -> 'Mary'", "VBD -> 'saw'", "PRP -> 'I'", "DT -> 'the'", "NN -> 'boy'"]
```

- make_syntaxrules

```
"S -> NP-SBJ VP", "PP -> IN NP", "NP-SBJ -> NONE", "NP -> DT NN", "NP-SBJ ->
PRP", "NP -> NP PP", "VP -> TO VP", "NP -> NN", "NP -> NONE", "PP-LOC -> IN
NP", …]
```

make_lexicalrules

```
189 def pos_rule(p):
190     """POS -> word"""
191     return(len(p.rhs()) == 1 and isinstance(p.rhs()[0], str))
192 
193 def make_lexicalrules(ps):
194     global wts
195     global mc_wts
196     wts = [wordtag(p) for p in ps if pos_rule(p)]
197     mc_wts = nltk.FreqDist(wts).most_common()
198     print(f"Lexical rules: {len(wts):,}")
```

```
>>> wts[:30]
[('In', 'IN'), ('American', 'JJ'), ('romance', 'NN'), (',', ','), ('almost', 'RB'), ('nothing', 'NN'), ('rates',
'VBZ'), ('*-2', 'NONE'), ('higher', 'JJR'), ('than', 'IN'), ('what', 'WP'), ('the', 'DT'), ('movie', 'NN'), ('men',
'NNS'), ('have', 'VB'), ('called', 'VBN'), ('*T*-1', 'NONE'), ('``', '``'), ('*', 'NONE'), ('meeting', 'NN'), ('cute',
'JJ'), ("''", "''"), ('--', ':'), ('that', 'DT'), …]

>>> mc_wts[:30]
[((',', ','), 84260), (('the', 'DT'), 73202), (('.', '.'), 70385), (('of', 'IN'), 39294), (('to', 'TO'), 37024),
(('a', 'DT'), 32606), (('and', 'CC'), 30415), (('in', 'IN'), 24902), (('*-1', 'NONE'), 20268), (('*', 'NONE'),
17363), (('*T*-1', 'NONE'), 17346), (('0', 'NONE'), 15504), (('``', '``'), 13750), (("''", "''"), 13487), (('for',
'IN'), 12669), (("'s", 'POS'), 12006), (('is', 'VBZ'), 10919), (('The', 'DT'), 10653), (('was', 'VBD'), 10394),
(('*U*', 'NONE'), 9255), (('it', 'PRP'), 9079), (('$', '$'), 8875), (('on', 'IN'), 8497), (('that', 'IN'), 8353),
(('said', 'VBD'), 8248), …]
```

Some Symbol Filtering: `nltk`

```
1.  'SBAR-PRP -> IN , S'
2.  'WHNP-1 -> WP$ NNP NNP NN'
3.  'SINV -> PP-LOC-TPC-1 VBD VP NP-SBJ'
4.  'NX -> JJ NN'
5.  'S -> NP-SBJ , PP-TMP VP'
6.  'VP -> S-ADV , VBD ADJP-PRD'
7.  'S -> S CC S ADVP'
8.  'VP -> ADVP VBD SBAR-NOM'
9.  'S-TPC-1 -> ADVP-TMP , S , CC S'
10. 'S-IMP -> INTJ , NP-SBJ VP .'
```

- Eliminate:
  - numeric suffixes (indexing),
    - e.g. `-1`
  - punctuation (terminals)*
    - e.g. `,` `.`
- Replace:
  - $ in POS tag names, signifies possessive form, by S*
    - e.g. `WP$` *whose* `PRP$` *his*
  - -NONE- by NONE*
    - also `-LRB-` `-RRB-`*

due to nltk CFG limitations

```
ValueError: Unable to parse line 2: WHNP -> WP$ NNP NNP NN
Expected a nonterminal, found: $ NNP NNP NN
ValueError: Unable to parse line 8: NP-SBJ -> -NONE-
Expected a nonterminal, found: -NONE-
```

make_lexicalrules

```
33# Overcome miscellaneous nltk grammar problems:
34def misc(s):
35    m = re.match(r"(.+)\$$", s)
36    if m:
37        return m.group(1) + 'S'                 # e.g. WP$ maps to WPS
38    else:
39        m = re.match(r"-(.+)-", s)
40        if m:
41            return m.group(1)              # e.g. -NONE- maps to NONE
42        else:
43            return s
44
45def wordtag(p):
46    """tuple (word, tag) with cleaned-up pos tag"""
47    return (p.rhs()[0], misc(trim_num(p.lhs().symbol())))
```

class nltk.grammar.Nonterminal **[source]**

Bases: `object`

A non-terminal symbol for a context free grammar. `Nonterminal` is a wrapper class for node values; it is used by `Production` objects to distinguish node values from leaf values. The node value that is wrapped by a `Nonterminal` is known as its "symbol". Symbols are typically strings representing phrasal categories (such as `"NP"` or `"VP"`). However, more complex symbol types are sometimes used (e.g., for lexicalized grammars). Since symbols are node values, they must be immutable and hashable. Two `Nonterminals` are considered equal if their symbols are equal.

```
16## GRAMMAR SYMBOL PROCESSING
17
18def trim_num(s):
19    """removes index etc. =1-4 =3 -1"""
20    m = re.match(r'(.+?)([=-][0-9]+)+', s)
21    if m:
22        return m.group(1)
23    else:
24        return s
25
26def trim_punc(s):
27    """delete non-words"""
28    if re.match(r'\W+$', s):
29        return ''
30    else:
31        return s
```

symbol()

Return the node value corresponding to this `Nonterminal`.

make_syntaxrules

```
200 def make_syntaxrules(ps):
201     global rules
202     global mc_rules
203     rules = [p_str2(p) for p in ps if not pos_rule(p)]
204     mc_rules = [r for (r, c) in nltk.FreqDist(rules).most_common() if c > 1]
205     print(f"Syntax rules: {len(rules):,}")
```

```
209 blockedrules = ['S -> NONE', 'NP -> NP NP', 'NP-SBJ -> NP NP', 'NP -> DT', 'NP -> NP NP-ADV', 'PP -> NONE', 'SBAR -> NONE']
```

```
>>> rules[:30]
['S -> S S', 'S -> PP NP-SBJ VP', 'PP -> IN NP', 'NP -> JJ NN', 'NP-SBJ -> RB NN', 'VP -> VBZ S', 'S -> NP-SBJ ADJP-PRD', 'NP-SBJ -> NONE', 'ADJP-PRD -> ADJP PP', 'ADJP -> JJR', 'PP -> IN SBAR-NOM', 'SBAR-NOM -> WHNP S', 'WHNP -> WP', 'S -> NP-SBJ VP', 'NP-SBJ -> DT NN NNS', 'VP -> VB VP', 'VP -> VBN S', 'S -> NP-SBJ S-NOM-PRD', 'NP-SBJ -> NONE', 'S-NOM-PRD -> NP-SBJ VP', 'NP-SBJ -> NONE', 'VP -> NN NP', 'NP -> JJ', 'S -> S-ADV NP-SBJ VP', 'S-ADV -> NP-SBJ VP', 'NP-SBJ -> DT', 'VP -> VBZ', 'NP-SBJ -> NN', 'VP -> VBZ ADJP-PRD SBAR-ADV', 'ADJP-PRD -> RB JJ']
```

make_syntaxrules

```
200 def make_syntaxrules(ps):
201     global rules
202     global mc_rules
203     rules = [p_str2(p) for p in ps if not pos_rule(p)]
204     mc_rules = [r for (r, c) in nltk.FreqDist(rules).most_common() if c > 1]
205     print(f"Syntax rules: {len(rules):,}")
```

```
>>> mc_rules[:40]
['S -> NP-SBJ VP', 'PP -> IN NP', 'NP-SBJ -> NONE', 'NP -> DT NN', 'NP-SBJ -> PRP', 'NP ->
  NP PP', 'VP -> TO VP', 'NP -> NN', 'NP -> NONE', 'PP-LOC -> IN NP', 'ADVP -> RB', 'NP ->
  DT JJ NN', 'NP -> NNS', 'VP -> MD VP', 'SBAR -> WHNP S', 'NP -> NNP', 'VP -> VB NP', 'NP
  -> PRP', 'PP-TMP -> IN NP', 'SBAR -> NONE S', 'PP-CLR -> IN NP', 'NP -> NNP NNP', 'NP-
  SBJ -> DT NN', 'NP -> JJ NNS', 'NP -> NP SBAR', 'SBAR -> IN S', 'NP-SBJ -> NP PP', 'VP -
  > VBD VP', 'NP-SBJ -> NNP', 'VP -> VBD SBAR', 'ADVP-TMP -> RB', 'VP -> VBD NP', 'PP ->
  TO NP', 'QP -> CD CD', 'S -> NONE', 'WHNP -> WDT', 'NP -> DT NNS', 'VP -> VBZ VP', 'VP -
  > VBG NP', 'NP-SBJ -> NNP NNP']
```

find_scfg.py

```
150 def find_scfg(s, parses=1, ec=False, printparses=True, oneline=True, printgrammar=False, lex=None, stop=None):
151     """finds the smallest cfg that can parse sentence s (string)"""
152     words = s.split()
153     if not lex:
154         lex = lexrules(words)
155     else:
156         lex = lexrules2(words, lex)
157         
158     if ec:
159         lex.append('NONE ->')        # EC rule
160     print(lex)
```

```
>>> find_scfg("Mary persuaded John to leave")
["TO -> 'to'", "NNP -> 'Mary'", "NNP -> 'John'", "VB -> 'leave'", "VBN -> 'persuaded'"]
```

```
lex: list of lexical rules to use (overrides default)
     example lexical rule: "DT -> 'the'"
```

find_scfg.py

```
165     blocked = 0
166     syntax = []
167     for l in range(len(mc_rules)):
168         found = 0
169         if rule_blocked(mc_rules[l]):
170             l += 1
171             blocked += 1
172             continue
173         syntax.append(mc_rules[l])
174         g = syntax + lex
175         cfg = nltk.CFG.fromstring(g)
176         p = nltk.ChartParser(cfg)
```

```
>>> len(mc_rules)
13485
```

classmethod fromstring(*input, encoding=None*)

Return the grammar instance corresponding to the input string(s).

Parameters

input – a grammar, either in the form of a string or as a list of strings.

class nltk.grammar.CFG **[source]**

Bases: `object`

A context-free grammar. A grammar consists of a start state and a set of productions. The set of terminals and nonterminals is implicitly specified by the productions.

find_scfg.py

```
209 blockedrules = ['S -> NONE', 'NP -> NP NP', 'NP-SBJ -> NP NP', 'NP -> DT', 'NP -> NP NP-ADV', 'PP -> NONE', 'SBAR -> NONE']
```

```
102 ### SYNTAX RULES
103
104 def rule_blocked(r):
105     return r in blockedrules
106
107 def find_rule(rule):
108     return mc_rules.index(rule)
```

```
165     blocked = 0
166     syntax = []
167     for l in range(len(mc_rules)):
168         found = 0
169         if rule_blocked(mc_rules[l]):
170             l += 1
171             blocked += 1
172             continue
173         syntax.append(mc_rules[l])
174         g = syntax + lex
175         cfg = nltk.CFG.fromstring(g)
176         p = nltk.ChartParser(cfg)
```

find_scfg.py

```
165     blocked = 0
166     syntax = []
167     for l in range(len(mc_rules)):
168         found = 0
169         if rule_blocked(mc_rules[l]):
170             l += 1
171             blocked += 1
172             continue
173         syntax.append(mc_rules[l])
174         g = syntax + lex
175         cfg = nltk.CFG.fromstring(g)
176         p = nltk.ChartParser(cfg)
```

class nltk.parse.chart.ChartParser **[source]**

Bases: `ParserI`

A generic chart parser. A “strategy”, or list of `ChartRuleI` instances, is used to decide what edges to add to the chart. In particular, `ChartParser` uses the following algorithm to parse texts:

Until no new edges are added:
For each *rule* in *strategy*:
Apply *rule* to any applicable edges in the chart.
Return any complete parses in the chart

find_scfg.py

```
177         trees = []¶
178         for tree in p.parse(words):¶
179             trees.append(tree)¶
180             found += 1¶
181         if found:¶
182             if found >= parses:¶
183                 print(f"Parses: {found:,}, # syntax rules: {l+1:,} (minus {blocked:,} blocked), # lexical rules: {len(
  lex):,}")¶
184                 parses = found + 1¶
185                 if stopping(stop, trees, printparses, oneline, g, printgrammar):¶
186                     break¶
```

parse(*tokens, tree_class=<class 'nltk.tree.tree.Tree'>*) **[source]**

Returns:

An iterator that generates parse trees for the sentence. When possible this list is sorted from most likely to least likely.

Parameters:

sent (*list(str)*) – The sentence to be parsed

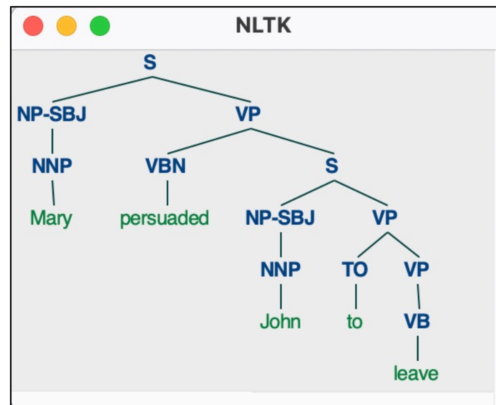
find_scfg.py

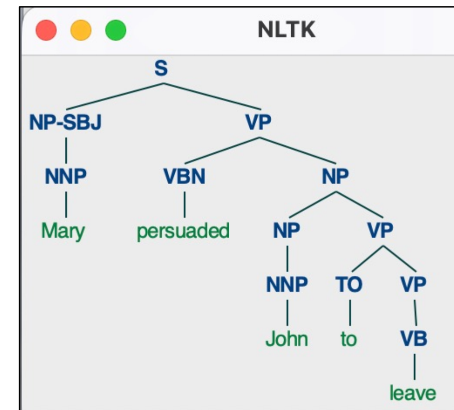
```
110# stop can be a rule or a list of rules
111def stopping(stop, trees, printparses, oneline, g, printgrammar):
112    if stop != None:
113        if type(stop) == 'str':
114            if stop in g:
115                print("FOUND {}!".format(stop))
116                print_parses(trees, printparses, oneline)
117                print_grammar(g, printgrammar)
118                return True
119        else:
120            notfound = 0
121            for r in stop:
122                if r not in g:
123                    notfound += 1
124            if notfound == 0:
125                print("FOUND ALL {}!".format(stop))
126                print_parses(trees, printparses, oneline)
127                print_grammar(g, printgrammar)
128                return True
129    else:
130        print_parses(trees, printparses, oneline)
131        print_grammar(g, printgrammar)
132        return True
133    return False
```

```
135### PRINTING
136
137def print_grammar(g, flag):
138    if flag:
139        for r in g:
140            print(r)
141
142def print_parses(trees, flag, oneline):
143    if flag:
144        for tree in trees:
145            if oneline:
146                tree.pprint(margin=1000)
147            else:
148                print(tree)
```

find_scfg.py

```
>>> find_scfg("Mary persuaded John to leave")
["TO -> 'to'", "NNP -> 'Mary'", "NNP -> 'John'", "VB -> 'leave'", "VBN -> 'persuaded'"]
Begin CFG grow
Parses: 2, # syntax rules: 115 (minus 4 blocked), # lexical rules: 5
(S (NP-SBJ (NNP Mary)) (VP (VBN persuaded) (NP (NP (NNP John)) (VP (TO to) (VP (VB leave))))))
(S (NP-SBJ (NNP Mary)) (VP (VBN persuaded) (S (NP-SBJ (NNP John)) (VP (TO to) (VP (VB leave))))))
grow CFG time: 0.08 (s)
>>> s = "(S (NP-SBJ (NNP Mary)) (VP (VBN persuaded) (S (NP-SBJ (NNP John)) (VP (TO to) (VP (VB leave))))))"
>>> t = nltk.Tree.fromstring(s)
>>> t.draw()
```





Homework 11

- Run the Python code (`-i` *interactive mode*):

```
$ python3 -i find_scfg.py
Creates lexical and syntactic rules from ptb: stored in mc_wts, mc_rules
find_scfg(string) finds the smallest CFG that can parse string.
  rules are added to the grammar in order of frequency: high > low.
  optional parameters:
   parses=1, ec=False, printparses=True, printgrammar=False, lex=None, …
    parses: # of parses for string the smallest CFG must produce.
    ec: True, permits -NONE- -> ''
    printparses: True, prints the parses found.
    printgrammar: True, prints the CFG found.
    lex: list of lexical rules to use (overrides default)
         example lexical rule: "DT -> 'the'"
```

Homework 11

- Setup:

```
20:53:51
Productions: 3,131,242
Lexical rules: 1,740,895
Syntax rules: 1,390,347
Total time: 23 (s)
>>> mc_wts[:10]
[((',', ','), 84260), (('the', 'DT'), 73202), (('.', '.'), 70385), (('of',
  'IN'), 39294), (('to', 'TO'), 37024), (('a', 'DT'), 32606), (('and',
  'CC'), 30415), (('in', 'IN'), 24902), (('*-1', 'NONE'), 20268), (('*',
  'NONE'), 17363)]
>>> mc_rules[:10]
['S -> NP-SBJ VP', 'PP -> IN NP', 'NP-SBJ -> NONE', 'NP -> DT NN', 'NP-SBJ
  -> PRP', 'NP -> NP PP', 'VP -> TO VP', 'NP -> NN', 'NP -> NONE', 'PP-LOC
  -> IN NP']
```

top-10 most_common word tags

top-10 most_common syntax rules

Homework 11

- **T1**:
  - run `find_scfg(s)` on "*John saw Mary*" vs. "*John likes Mary*"
- **Q1**:
  - which grammar is smaller?
- **Q2**:
  - why (*is one smaller than the other*)?
  - (To see the CFG used, add the parameter `printgrammar=True`)

Homework 11

- **T2**:
  - run `find_scfg(s)` on "*Mary saw the boy who I saw* " vs. " *Mary saw the boy I saw* "
- **Q3**:
  - which grammar is smaller?
- **Q4**:
  - are the parses correct? Explain.
  - (you can 2D render the trees using `nltk.Tree.fromstring()` or `tregex`)

Homework 11

- **T3**:
  - run `find_scfg(s)` on "*Mary saw the boy who I saw* " vs. " *Mary saw the boy I saw* ", but adding the parameter `ec=True`
  - Note: parameter `ec: True, permits -NONE- -> ''`
- **Q5**:
  - which grammar is smaller? Why?
- **Q6**:
  - are the parses correct? Explain.

Homework 11

- Submit to sandiway@arizona.edu
- SUBJECT: 581 Homework 11 *YOUR NAME*
- One PDF file (for grading)
  - include your screenshots
  - include your Python code
- Usual Deadline:
  - midnight Saturday
  - to be grade Sunday
  - we will review the homework next Monday