Handout 18: Feature grammars

1. Our grammars so far allow some ungrammatical sentences

- **a.** "the dogs barks"
- **b.** "these dog"
- **c.** "the dog chased," "the dog barked the cat"
- **d.** "has will chasing"
- **e.** "who did you see the dog"

2. Using features

- **a.** Consider this grammar

```
S -> A A
A -> b
A -> c
```

- **b.** Language: *bb*, *bc*, *cb*, *cc*
- **c.** Feature representing choice between *b* and *c*

```
A[w=b] -> b
A[w=c] -> c
```

- **d.** Require that we make the same choice in both places:

```
S -> A[w=?w] A[w=?w]
```

- **e.** The variables "link" the two positions—must have same value in both positions

3. Longer chains

- **a.** What language does this grammar generate?

```
S -> A A
A -> a A
A -> b
A -> c
```

- **b.** Add features to require the same choice (*b* versus *c*) both times

4. Agreement

- **a.** Grammar `fg1.fcfg`

```
% start S

S -> NP[pl=?p] VP[pl=?p]
NP[pl=?p] -> Det[pl=?p] N[pl=?p]
VP[pl=?p] -> Vt[pl=?p] NP
VP[pl=?p] -> Vi[pl=?p]
```

```
Det[-pl] -> 'a'
Det -> 'the'
N[-pl] -> 'cat' | 'dog'
N[+pl] -> 'cats' | 'dogs'

Vi[-pl] -> 'barks' | 'meows'
Vi[+pl] -> 'bark' | 'meow'
Vt[-pl] -> 'chases' | 'likes'
Vt[+pl] -> 'chase' | 'like'
```

b. `Vt[+pl]` is shorthand for `Vt[pl=+]` (except that the latter is bad syntax)

c. `Vt[+pl] -> 'barks'` requires the Vt node to have the attribute `pl`, and requires its value to be `+`

d. If the Vt node has additional attributes, `Vt[+pl] -> 'barks'` does not care

5. Draw the parse tree; show the chains

a. *a cat barks*

b. *a cats barks*

c. *the cat barks*

d. *the cats bark*

e. *the cat chases the dog*

6. How do we modify the grammar to handle

[NP [NP the cat] [PP in the park]] chases the dog

7. Python

a. Creating a parser for a feature grammar.

```
>>> from nltk import load_parser
>>> parser = load_parser('fg1.fcfg')
>>> print parser.grammar()
Grammar with 18 productions (start state = S[])
    Det[-pl] -> 'a'
    Det[] -> 'the'
    ...
```

b. For the function `load_parser`, the file suffix matters

c. A convenience function:

```
def pp (s):
    for t in parser.parse(s.split()):
        print(t)
```

d. Examples:

```
>>> pp('the cat barks')
(S[] (NP[-pl] (Det[] the) (N[-pl] cat)) (VP[-pl] (Vi[-pl] barks)))
>>> pp('the cats barks')
>>>
```

8. Handling subcategorization with features (`fg2.fcfg`)

```
% start S

###  Grammar
NP[f=?f] -> Det[f=?f] N[f=?f]
PP[f=?f] -> P[f=?f] NP
S -> NP[f=?f] VP[f=?f]
SC[f=?c] -> Comp[f=?c] S
VP[f=?f] -> V[f=?f, -t, s=0]
VP[f=?f] -> V[f=?f, +t, s=0] NP
VP[f=?f] -> V[f=?f, -t, s=?p] PP[f=?p]
VP[f=?f] -> V[f=?f, +t, s=?p] NP PP[f=?p]
VP[f=?f] -> V[f=?f, -t, s=?c] SC[f=?c]
VP[f=?f] -> V[f=?f, +t, s=?c] NP SC[f=?c]

###  Lexicon
Comp[f=that] -> 'that'
Comp[f=Q] -> 'if'
Comp[f=Q] -> 'whether'
Det[f=sg] -> 'a' | 'an'
Det[f=pl] -> 'these'
Det -> 'the'
N[f=sg] -> 'aardvark' | 'cat' | 'dog'
N[f=pl] -> 'aardvarks' | 'cats' | 'dogs'
P[f=loc] -> 'in'
P[f=loc] -> 'on'
P[f=of] -> 'of'
P[f=to] -> 'to'
V[f=sg, -t, s=0] -> 'barks'
V[f=pl, -t, s=0] -> 'bark'
V[f=sg, +t, s=0] -> 'chases'
V[f=pl, +t, s=0] -> 'chase'
V[f=sg, +t, s=loc] -> 'puts'
V[f=sg, -t, s=that] -> 'thinks'
V[f=sg, -t, s=Q] -> 'wonders'
```

9. Examples

a. Load the parser

```
>>> parser = load_parser('fg2.fcfg')
```

b. `pp('the cat barks')`

```
(S[]
  (NP[f='sg'] (Det[] the) (N[f='sg'] cat))
  (VP[f='sg'] (V[f='sg', s=0, -t] barks)))
```

c. `pp('the cat chases the dog')`

```
(S[]
  (NP[f='sg'] (Det[] the) (N[f='sg'] cat))
  (VP[f='sg']
    (V[f='sg', s=0, +t] chases)
    (NP[f='sg'] (Det[] the) (N[f='sg'] dog))))
```

d. `pp('the cat barks the dog')` – no output

e. Getting more details

```
>>> parser.chart_parse('the cat barks the dog'.split(), trace=2)
|.th.ca.ba.th.do.|
Leaf Init Rule:
|[--]  .  .  .  .| [0:1] 'the'
|.  [--]  .  .  .| [1:2] 'cat'
|.  .  [--]  .  .| [2:3] 'barks'
|.  .  .  [--]  .| [3:4] 'the'
|.  .  .  .  [--]| [4:5] 'dog'
...
Feature Bottom Up Predict Combine Rule:
|.  .  [--]  .  .| [2:3] VP[f='sg'] -> V[f='sg', s=0, -t] *
|.  .  [-->  .  .| [2:3] VP[f=?f] -> V[f=?f, s=?p, -t] * PP[f=?p] {?f: u'sg', 
|.  .  [-->  .  .| [2:3] VP[f=?f] -> V[f=?f, s=?c, -t] * SC[f=?c] {?c: 0, ?f: u
...
```

f. `pp('the cat thinks that the aardvark chases the dog')`

```
(S[]
  (NP[f='sg'] (Det[] the) (N[f='sg'] cat))
  (VP[f='sg']
    (V[f='sg', s='that', -t] thinks)
    (SC[f='that']
      (Comp[f='that'] that)
      (S[]
        (NP[f='sg'] (Det[] the) (N[f='sg'] aardvark))
        (VP[f='sg']
          (V[f='sg', s=0, +t] chases)
          (NP[f='sg'] (Det[] the) (N[f='sg'] dog)))))))
```

g. `pp('the cat wonders that the aardvark chases the dog')` – no output

h. `pp('the cat wonders if the aardvark chases the dog')`

```
(S[]
  (NP[f='sg'] (Det[] the) (N[f='sg'] cat))
  (VP[f='sg']
    (V[f='sg', s='Q', -t] wonders)
    (SC[f='Q']
      (Comp[f='Q'] if)
      (S[]
        (NP[f='sg'] (Det[] the) (N[f='sg'] aardvark))
        (VP[f='sg']
          (V[f='sg', s=0, +t] chases)
          (NP[f='sg'] (Det[] the) (N[f='sg'] dog)))))))
```

10. Handling the auxiliary sequence: treat auxiliaries as verbs that select VPs (`fg3.fcfg`)

```
% start S

###  Grammar
NP[f=?f] -> Det[f=?f] N[f=?f]
S -> NP[f=?f] VP[f=?f]
VP[f=?f] -> V[f=?f, -t, s=0]
VP[f=?f] -> V[f=?f, +t, s=0] NP
VP[f=?f] -> V[f=?f, s=?g] VP[f=?g]

###  Lexicon
Det[f=sg] -> 'a'
N[f=sg] -> 'cat' | 'dog'
V[f=sg, s=ing] -> 'is' | 'was'
V[f=sg, -t, s=0] -> 'barks'
V[f=ing, -t, s=0] -> 'barking'
V[f=sg, +t, s=0] -> 'chases'
V[f=ing, +t, s=0] -> 'chasing'
```

11. Examples

a. `pp('a dog chases a cat')`

```
(S[]
  (NP[f='sg'] (Det[f='sg'] a) (N[f='sg'] dog))
  (VP[f='sg']
    (V[f='sg', s=0, +t] chases)
    (NP[f='sg'] (Det[f='sg'] a) (N[f='sg'] cat))))
```

b. `pp('a dog is chasing a cat')`

```
(S[]
  (NP[f='sg'] (Det[f='sg'] a) (N[f='sg'] dog))
  (VP[f='sg']
    (V[f='sg', s='ing'] is)
    (VP[f='ing']
      (V[f='ing', s=0, +t] chasing)
      (NP[f='sg'] (Det[f='sg'] a) (N[f='sg'] cat)))))
```

c. `pp('a dog is chases a cat')` – no output

d. `pp('a dog chasing a cat')` – no output

12. How do we modify the grammar to handle the following?

a. the cat has been chasing the dog

b. *the cat is having chased the dog

c. the cat will chase the dog

d. *the cat will chased the dog

e. the cat was chased

f. *the cat was barked

g. the cat wanted to bark

h. *the cat wanted that the dog barks

i. the aardvark persuaded the cat to bark

j. the aardvark persuaded the cat that the dog barks

k. the aardvark knows the cat

l. the aardvark knows that the cat barks

m. the aardvark knows whether the cat barks

n. the cat gave the dog a present

o. the cat is happy

p. you are happy

q. the black cats bark

r. *the black cats barks

s. the cat and the dog bark

t. *the cat and the dog barks

u. cats bark