Handout 22: Semantic interpretation

1. Lambda simplification. Simplify:

- **a.** `(\x.BARKER(x)) (Max)`
- **b.** `(\y.RED(y)) (x)`
- **c.** `(\y.(RED(y) & DOG(y))) (SPOT)`
- **d.** `(\P.(DOG(MAX) -> P(MAX))) (BARKER)`
- **e.** `(\P. all x.(DOG(x) -> P(x))) (BARKER)`

2. Complexify:

- **a.** `BARKER(MAX)` = `?(MAX)`
- **b.** `DOG(SPOT) | CAT(SPOT)` = `?(SPOT)`
- **c.** `DOG(SPOT) <-> DOG(MAX)` = `?(DOG)`
- **d.** `all x.(BLUE(x) -> DOG(x))` = `?(DOG)`
- **e.** `\P.all x.(CAT(x) -> -P(x))` = `?(CAT)`

3. A grammar with semantics

a. Use angle brackets to mark logical expressions. `sem1.fcfg`:

```
S[v=<?vp(?np)>] -> NP[v=?np] VP[v=?vp]
NP[v=<FIDO>] -> 'Fido'
VP[v=<BARKER>] -> 'barks'
```

b. Note: `load_parser()` doesn't reload the parser if you change the grammar file.

```
>>> from nltk.grammar import FeatureGrammar
>>> from nltk import FeatureChartParser
>>> def reload (fn):
...     s = open(fn).read()
...     g = FeatureGrammar.fromstring(s)
...     return FeatureChartParser(g)
...
```

c. Load and parse

```
>>> p = reload('sem1.fcfg')
>>> ts = list(p.parse('Fido barks'.split()))
>>> print ts[0]
(S[v=<BARKER(FIDO)>] (NP[v=<FIDO>] Fido) (VP[v=<BARKER>] barks))
```

d. Getting just the semantic value

```
>>> s = ts[0].label()
>>> s
S[v=<BARKER(FIDO)>]
>>> e = s['v']
>>> e
<ApplicationExpression BARKER(FIDO)>
>>> print e
BARKER(FIDO)
```

e. Let `M` be the model from Handout 27 and let `g` be the assignment.

```
>>> M.satisfy(e,g)
True
```

(Remember that `M.evaluate` takes a string, not an expression object.)

4. Transitive verbs

a. "Max chases Fido" $\Rightarrow$ `CHASES(MAX,FIDO)`

b. [$_{\text{NP}}$ Max] [$_{\text{VP}}$ chases Fido]

c. How to pull "`Max`" out of the semantic form?

`CHASES(x,FIDO)` with $x =$ `MAX`
`\x.CHASES(x,FIDO)` applied to `MAX`
`(\x.CHASES(x,FIDO)) (MAX)` $=$ `CHASES(MAX,FIDO)`

d. Update grammar:

```
% start S

###  Grammar
S[v=<?vp(?np)>] -> NP[v=?np] VP[v=?vp]
NP[v=?n] -> Name[v=?n]
VP[v=?v] -> V[-t, v=?v]
VP[v=<\x.?v(x,?np)>] -> V[+t, v=?v] NP[v=?np]

###  Lexicon
Name[v=<FIDO>] -> 'Fido'
Name[v=<SPOT>] -> 'Spot'
Name[v=<MAX>] -> 'Max'
V[-t, v=<BARKER>] -> 'barks'
V[+t, v=<CHASES>] -> 'chases'
```

e. Reload and parse

```
>>> p = reload('sem1.fcfg')
>>> ts = list(p.parse('Max chases Fido'.split()))
>>> print ts[0]
(S[v=<CHASES(MAX,FIDO)>]
```

```
    (NP[v=<MAX>] (Name[v=<MAX>] Max))
    (VP[v=<\x.CHASES(x,FIDO)>]
      (V[+t, v=<CHASES>] chases)
      (NP[v=<FIDO>] (Name[v=<FIDO>] Fido))))
```

f. Translation

```
>>> s = ts[0].label()
>>> e = s['v']
>>> e
<ApplicationExpression CHASES(MAX,FIDO)>
```

g. Value

```
>>> M.satisfy(e,g)
False
```

5. Quantifiers

a. "every dog barks," "a dog barks"

b. `all x.(DOG(x) -> BARKER(x))`

c. `exists x.(DOG(x) & BARKER(x))`

d. First approximation

```
S[v=<all x.(?n(x) -> ?vp(x))>] -> 'every' N[v=?n] VP[v=?vp]
S[v=<exists x.(?n(x) & ?vp(x))>] -> 'a' N[v=?n] VP[v=?vp]
```

e. How to group quantifier with N? Pull out VP meaning:

`all x.(DOG(x) -> P(x))` with $P =$ `BARKER`
`exists x.(DOG(x) & P(x))` with $P =$ `BARKER`

f. That is:

```
         ?qp                       (?vp)
(\P. all x.(DOG(x) -> P(x)))     (BARKER)
(\P. exists x.(DOG(x) & P(x)))   (BARKER)
```

g. Works for $P =$ `\x.LIKES(x,MAX)`, too

```
(\P. all x.(DOG(x) -> P(x)))  (\y.LIKES(y,MAX))
all x.(DOG(x) -> (\y.LIKES(y,MAX))(x))
all x.(DOG(x) -> LIKES(x,MAX))
```

6. Putting it together

a. Additions to grammar:

```
S[v=<?qp(?vp)>] -> QP[v=?qp] VP[v=?vp]
QP[v=<\P.all x.(?n(x) -> P(x))>] -> 'every' N[v=?n]
QP[v=<\P.exists x.(?n(x) & P(x))>] -> 'a' N[v=?n]

N[v=<CAT>] -> 'cat'
N[v=<DOG>] -> 'dog'
```

b. `every dog chases Max`

```
(S[v=<all x.(DOG(x) -> CHASES(x,MAX))>]
  (QP[v=<\P.all x.(DOG(x) -> P(x))>] every (N[v=<DOG>] dog))
  (VP[v=<\x.CHASES(x,MAX)>]
    (V[+t, v=<CHASES>] likes)
    (NP[v=<MAX>] (Name[v=<MAX>] Max))))
```

c. `a cat chases Spot`

```
>>> ts = list(p.parse('a cat chases Spot'.split()))
>>> print ts[0]
(S[v=<exists x.(CAT(x) & CHASES(x,SPOT))>]
  (QP[v=<\P.exists x.(CAT(x) & P(x))>] a (N[v=<CAT>] cat))
  (VP[v=<\x.CHASES(x,SPOT)>]
    (V[+t, v=<CHASES>] chases)
    (NP[v=<SPOT>] (Name[v=<SPOT>] Spot))))
```

d. Evaluate:

```
>>> s = ts[0].label()
>>> e = s['v']
>>> print e
exists x.(CAT(x) & CHASES(x,SPOT))
>>> M.satisfy(e,g)
True
```

7. Noun modification

a. Grammar additions/modifications

```
QP[v=<\P.all x.(?n(x) -> P(x))>] -> 'every' N1[v=?n]
QP[v=<\P.exists x.(?n(x) & P(x))>] -> 'a' N1[v=?n]
N1[v=?n] -> N[v=?n]
N1[v=<\x.(?a(x) & ?n(x))>] -> A[v=?a] N1[v=?n]
A[v=<RED>] -> 'red'
A[v=<BLUE>] -> 'blue'
```

b. `ts = list(p.parse('a blue dog barks'.split()))`

```
>>> print ts[0]
(S[v=<exists x.(BLUE(x) & DOG(x) & BARKER(x))>]
  (QP[v=<\P.exists x.(BLUE(x) & DOG(x) & P(x))>]
    a
    (N1[v=<\x.(BLUE(x) & DOG(x))>]
      (A[v=<BLUE>] blue)
      (N1[v=<DOG>] (N[v=<DOG>] dog))))
  (VP[v=<BARKER>] (V[-t, v=<BARKER>] barks)))
```

c. Translation and evaluation

```
>>> s = ts[0].label()
>>> e = s['v']
```

```
>>> print e
exists x.(BLUE(x) & DOG(x) & BARKER(x))
>>> M.satisfy(e,g)
True
```

8. Refinement: breaking QP into Det and N

a. QP meaning is `\P. all x.(DOG(x) -> P(x))`

```
        ?d                         (?n)
(\R.\P. all x.(R(x) -> P(x)))     (DOG)
(\R.\P. exists x.(R(x) & P(x)))  (DOG)
```

b. Rules:

```
QP[v=<?d(?n)>] -> Q[v=?d] N[v=?n]
Q[v=<\R.\P.all x.(R(x) -> P(x))>] -> ’every’
Q[v=<\R.\P.exists x.(R(x) & P(x))>] -> ’a’
```