Handout 21: Predicate Calculus

1. Orientation

- **a.** The next step is **interpretation**: mapping sentences to representations of meaning (via parse trees)
- **b.** Meaning-representation language: first-order predicate calculus (FOPC)
- **c.** We will then add **semantic translations** to the grammar to translate the parse tree to FOPC

  “Fido chases every cat” ⇒
  `all x.( CAT(x) -> CHASES(FIDO, x) )`
- **d.** But what is the meaning of an FOPC formula? Our representation of the world is a **model**.

2. A little world

- **a.** “Everything in the world” is the **domain**
- **b.** The domain is a set, and its elements are **individuals**

```
D = set(['i1', 'i2', 'i3', 'i4'])
```

3. Suppose:

- **a.** i_1 and i_2 are dogs, i_3 is a cat, and i_4 is a seal.
- **b.** i_1 and i_3 are red, and i_2 is blue.
- **c.** i_1, i_2, and i_4 bark.
- **d.** i_1 is named “Fido,” i_2 is named “Spot,” and i_3 is named “Max.”
- **e.** i_1 chases i_3 and i_3 chases i_2

4. Logical constants

- **a.** First step in defining a **language** to talk about our world.
- **b.** **Constants** name individuals: `FIDO`, `SPOT`, `MAX`
- **c.** **Predicates** name “detector functions” that take an individual as input and return true or false: `RED`, `DOG`, `BARKER`
- **d.** A detector function is equivalent to a set. (It is the **characteristic function** of the set.)
- **e.** **Relation symbols** name relations, i.e., sets of tuples: `CHASES`

5. Models

a. A **valuation** associates constants with their meanings. File `m1.val`:

```
FIDO => i1
SPOT => i2
MAX => i3
DOG => {i1, i2}
CAT => {i3}
SEAL => {i4}
RED => {i1, i3}
BLUE => {i2}
BARKER => {i1, i2, i4}
CHASES => {(i1, i3), (i3, i2)}
```

b. Loading it:

```
>>> from nltk import Valuation
>>> F = Valuation.fromstring(open('m1.val').read())
```

c. Getting the value of a constant:

```
>>> F['FIDO']
'i1'
```

d. A **model** pairs a domain with a valuation.

```
>>> from nltk import Model
>>> M = Model(D,F)
```

6. Variables

a. The model contains meanings for *constants.* An `Assignment` contains meanings (values) for variables.

b. Let $x = i_1$ and $y = i_3$

```
>>> from nltk import Assignment
>>> g = Assignment(D)
>>> g['x'] = 'i1'
>>> g['y'] = 'i3'
```

c. Getting the value of a variable:

```
>>> g['y']
'i3'
```

7. Evaluating expressions

a. Function application: `RED(FIDO)`

b. Using the model and assignment to evaluate an expression:

```
>>> M.evaluate('RED(FIDO)', g)
True
>>> M.evaluate('RED(SPOT)', g)
```

```
False
>>> M.evaluate('RED(y)', g)
True
```

c. Can evaluate simple expressions, too:

```
>>> M.evaluate('MAX', g)
'i3'
```

d. And relations applied to arguments:

```
>>> M.evaluate('CHASES(FIDO, MAX)', g)
True
```

8. To save some typing:

```
>>> class Evaluator (object):
...     def __init__ (self, model, asst):
...         self.model = model
...         self.assignment = asst
...     def __call__ (self, s):
...         return self.model.evaluate(s, self.assignment)
...
>>> v = Evaluator(M,g)
```

9. Some more expression types

```
>>> v('RED(FIDO)')
True
>>> v('BLUE(MAX)')
False
>>> v('RED(FIDO) & BLUE(MAX)')
False
>>> v('RED(FIDO) | BLUE(MAX)')
True
>>> v('-( RED(FIDO) & BLUE(MAX) )')
True
```

10. Precedence

a. Negation binds tightly

```
>>> v('-BLUE(FIDO) & DOG(MAX)')
False
>>> v('-( BLUE(FIDO) & DOG(MAX) )')
True
```

b. Conjunction binds more tightly than disjunction

```
>>> v('DOG(SPOT) | BLUE(FIDO) & DOG(MAX)')
True
>>> v('( DOG(SPOT) | BLUE(FIDO) ) & DOG(MAX)')
False
```

11. The material conditional

a. If the condition is satisfied, we evaluate the body

```
>>> v('BARKER(FIDO) -> CAT(MAX)')
True
>>> v('BARKER(FIDO) -> DOG(MAX)')
False
```

b. If the condition is *not* satisfied, the statement is vacuously true

```
>>> v('BLUE(FIDO) -> CAT(MAX)')
True
>>> v('BLUE(FIDO) -> DOG(MAX)')
True
```

12. Tables for the operators:

Inputs			Output
T	&	T	T
T	&	F	F
F	&	T	F
F	&	F	F

Inputs			Output
T	|	T	T
T	|	F	T
F	|	T	T
F	|	F	F

Inputs		Output
-	T	F
-	F	T

Inputs			Output
T	->	T	T
T	->	F	F
F	->	T	T
F	->	F	T

Inputs			Output
T	<->	T	T
T	<->	F	F
F	<->	T	F
F	<->	F	T

13. Variables

a. Recall that x is i_1 (Fido) and y is i_3 (Max).

```
>>> v('CHASES(FIDO,y)')
True
>>> v('CHASES(y,FIDO)')
False
>>> v('CHASES(x,y)')
True
```

b. Unbound variables evaluate as `Undefined`

```
>>> v('BARKER(c)')
'Undefined'
```

14. How do we say "all dogs are barkers"?

a.
```
>>> v('all x.( DOG(x) -> BARKER(x) )')
True
>>> v('all x.( DOG(x) -> RED(x) )')
False
```

b. Don't omit the parentheses, or things go wrong in mysterious ways!

c. Computation:

x	DOG(x)	BARKER(x)	DOG(x) -> BARKER(x)
i_1	T	T	T
i_2	F	F	T
i_3	T	T	T
i_4	F	T	T

d. E.g., for the last line:

```
>>> v.assignment['x'] = 'i4'
>>> v('DOG(x)')
False
>>> v('BARKER(x)')
True
>>> v('DOG(x) -> BARKER(x)')
True
```

15. Versus “all barkers are dogs”

a.

```
>>> v('all x.( BARKER(x) -> DOG(x) )')
False
```

b. Why is it false?

```
>>> from nltk import Expression
>>> E = Expression.fromstring
>>> M.satisfiers(E('BARKER(x) -> DOG(x)'), 'x', g)
{'i1', 'i3', 'i2'}
```

c. Computation:

x	BARKER(x)	DOG(x)	BARKER(x) -> DOG(x)
i_1	T	T	T
i_2	F	F	T
i_3	T	T	T
i_4	T	F	F

16. Existential

a. “there is a red dog”

```
>>> v('exists x.( RED(x) & DOG(x) )')
True
```

b. Computation

x	RED(x)	DOG(x)	RED(x) & DOG(x)
i_1	T	T	T
i_2	F	T	F
i_3	T	F	F
i_4	F	F	F

c. Contrast with

```
>>> v('exists x.( BLUE(x) & CAT(x) )')
False
```

d. Computation

x	BLUE(x)	CAT(x)	BLUE(x) & CAT(x)
f	F	F	F
s	T	F	F
m	F	T	F
c	F	F	F

17. CAUTION

a. Not this way!

```
>>> v('exists x.( BLUE(x) -> CAT(x) )')
True
```

b. Computation

x	BLUE(x)	CAT(x)	BLUE(x) -> CAT(x)
f	F	F	T
s	T	F	F
m	F	T	T
c	F	F	T

18. Set abstraction (actually, function abstraction)

a. How do we say "`RED_DOG(FIDO)`"?

b. "`RED_DOG`" = `\x.( RED(x) & DOG(x) )`

c. It is a function that maps individuals to true or false

```
>>> v('\\x.( RED(x) & DOG(x) )')
{'i1': True, 'i3': False, 'i2': False, 'i4': False}
```

d. Apply it to Fido:

```
>>> v('\\x.( RED(x) & DOG(x) ) (FIDO)')
True
>>> v('\\x.( RED(x) & DOG(x) ) (SPOT)')
False
```

e. Same as replacing `x` with `FIDO`:

```
>>> v('RED(FIDO) & DOG(FIDO)')
True
```

19. Write lambda expressions:

a. "blue cat"

b. "something that is blue or a cat"

c. "non-barking dog"

d. "cat that is neither a barker nor red"

e. "individuals that like Spot"

f. "dogs that like Spot"

g. "dogs that do not like Max"

h. "individuals that like every cat"

i. "non-self-liking individuals"

20. Write predicate calculus expressions for each of the following. If the sentence is ambiguous, write an expression for each reading.

a. every red dog barks

b. Fido chases a big cat

c. there's a dog that chases every orange cat

d. an orange cat chases each dog

e. a dog in every household likes Lassie

f. every dog that is in some household likes Lassie

g. there is an orange cat such that every dog that chases it gets scratched