PY1003
Lecture 19: Revision Lecture II

An important point to remember about all trees:

When applying a tree rule, you must apply it to every open branch on which the sentence appears.

1. Many-place predicates

All predicates have a certain number of *places*. This is equal to the number of individual constants by which they would need to be followed in order to create a grammatical atomic sentence. The order of the constants or variables following a many-place predicate is important.

Dave hates Arnold
Hab (a: Dave, b: Arnold)

Arnold hates Dave
Hba

Everyone hates Arnold
∀xHxb

Arnold hates everyone
∀xHbx

Dave prefers Christine to Arnold
Pacb (c: Christine)

Dave prefers Arnold to Christine
Pabc

Remember: a predicate letter always has a *fixed* number of places.

Mary is an aunt and Emma is John's aunt

X Aa ∧ Abc
(a: Mary
b: Emma
c: John
A??)

√ ∃xAax ∧ Abc
(a: Mary
b: Emma
c: John
Axy: x is an aunt of y)

2. Multiple generality

When two or more quantifiers have overlapping scope, we must use different variable letters with each quantifier:

X $\exists x \forall x Lxx$

√ $\exists x \forall y Lxy$

The order of the quantifiers is also important:

$\exists x \forall y Hxy$

There is someone who hates everyone

$\forall x \exists y Hxy$

Everyone is hated by someone or other

Using multiple generality we can translate some quite complex phrases, e.g.:

Some student is taller than some professor

$\exists x \exists y[(Sx \wedge Py) \wedge Txy]$

(Sx: x is a student
Px: x is a professor
Txy: x is taller than y)

Some boy hates every girl

$\exists x[Bx \wedge \forall y(Gy \rightarrow Hxy)]$

(Bx: x is a boy
Gx: x is a girl
Hxy: x hates y)

Remember: only apply the rule for a quantifier (or negated quantifier) if it has the widest scope.

Examples:

1. $\exists x \forall y Rxy$
2. $\forall x \forall y(Sx \wedge Py)$
3. $\neg \forall x(Fx \vee \exists y Lxy)$
4. $\neg(\forall x Fx \vee \exists y Gy)$
5. $\neg \exists x \neg Px$

And never apply two quantifier rules at once.

Infinite trees can occur in Predicate Logic. If an argument is valid or a set of sentences is inconsistent, your tree will close (provided you apply the right rules). But if the argument is invalid or the set is consistent, it may have infinitely long open branches.

Sometimes it may look like you have an infinite branch, but in fact the branch can be closed if you apply the correct rule. You must apply the rule that will close the branch in these cases.

Models for infinite trees become difficult, because many sentences will appear on an infinite branch (and hence cannot be assigned arbitrary truth-values) although you have not actually written them down.

We have covered this briefly but don't worry about it: you will not be assessed on models for infinite trees.

<u>3. Choosing which letter to instantiate with</u>

Are these sentences consistent?

> {Brian is frivolous; Everyone is liked by someone or other; If a person is frivolous everyone dislikes them}

An inefficient tree:

```
                          Fa
                       ∀x∃yLyx                √
                   ∀x(Fx→∀y¬Lyx)              √^b√^a
                          |
                        ∃yLya                 √
                          |
                         Lba
                          |
                     Fb→∀y¬Lyb                √
                    /              \
               ¬Fb                  ∀y¬Lyb
                |                     |
Fa→∀y¬Lya       √              Fa→∀y¬Lya          √
  /             \                /       \
¬Fa            ∀y¬Lya  √^a√^b   ¬Fa      ∀y¬Lya        √^a√^b
                 |                          |
               ¬Laa                       ¬Laa
                 |                          |
               ¬Lba                       ¬Lba
```

An efficient tree that proves exactly the same thing:

```
          Fa
       ∀x∃yLyx             √
   ∀x(Fx→¬∃yLyx)           √^a
          |
        ∃yLya              √
          |
         Lba
          |
     Fa→∀y¬Lya             √
     /         \
  ¬Fa        ∀y¬Lya        √^b
                |
              ¬Lba
```

3. Identity

Qualitative identity vs numerical identity.
Some things we may want to write:

a=b
¬a=b
a=b v Fa

Never write any of the following:

a=¬b
a=F
a=bvc
a=b∧c

Identity Rule 1: if a=b appears on a branch then you can uniformly substitute a for b (or b for a) in any sentence that appears on the branch.
This rule is sometimes also known as *Leibniz's Law*.

E.g.:

```
 Fd
Rab
d=e
 |
 Fe
```

Motivation: if d=e and Fd, there is only one way for this to be true, namely if Fe. So a non-branching move takes you from these two sentences to Fe.

Identity Rule 2: for any individual constant a, you can write a=a on any branch at any time.

E.g.

```
      Fa
   Fa→¬a=a
    /     \
  ¬Fa     ¬a=a
           a=a
```

Motivation: a=a is always true.

Note: when using the identity rules we do not tick anything off.

Identity introduces a new kind of complexity into the construction of branch models and branch counterexamples. Unlike before, there may be sentences such that neither they nor their negations appear on your open branch, and yet they cannot be assigned arbitrary truth-values.

You will not be assessed on models or counterexamples for sentences or arguments involving identity.

But for those who are interested, consider this tree:

```
          a=b
        Fa v Fb          √
        /       \
    Fa           Fb
```

Now consider trying to construct a model for the starting sentences using the left-hand branch. We must assign:
I(Fa) = T
I(a=b) = T

But no other atomic sentences or negated atomic sentences appear on the branch. So it looks like we can arbitrarily assign values to the other combinations, for instance like this:
I(Fb) = F
I(b=a) = F

But in fact we cannot assign either of these sentences arbitrary values.
- Because of the symmetry of identity, if a=b is true then b=a must be true as well.
- And because of Leibniz's law, if Fa is true and so is a=b, then Fb must also be true.

<u>4. Only, at least n, at most n, exactly n</u>

You're the only one I love
$Lab \wedge \forall y(Lay \rightarrow y=b)$
(a: me
b: you
Lxy: x loves y)

Only Lister and Frankenstein survived
$[(Sa \wedge Sb) \wedge \forall y(Sy \rightarrow (y=a \vee y=b))]$
(a: Lister
b: Frankenstein
Sx: x survived)

There was at least one survivor
$\exists xSx$

There were at least two survivors
$\exists x \exists y[(Sx \wedge Sy) \wedge \neg x=y]$

There was at most one survivor
$\forall x \forall y[(Sx \wedge Sy) \rightarrow x=y]$

There were at most two survivors
$\forall x \forall y \forall z[((Sx \wedge Sy) \wedge Sz) \rightarrow (x=y \vee (x=z \vee y=z))]$

There was exactly one survivor
= There was at least one survivor and any survivor is identical to him/her
$\exists x[Sx \wedge \forall y(Sy \rightarrow y=x)]$

There were exactly two survivors
= There were at least two survivors and any survivor is identical to one of those two
$\exists x \exists y[((Sx \wedge Sy) \wedge \neg x=y) \wedge \forall z(Sz \rightarrow (z=x \vee z=y))]$

<u>5. Definite Descriptions</u>

The drive plate was inefficiently repaired
$\exists x[(Dx \wedge \forall y(Dy \rightarrow y=x)) \wedge Ix]$
(Dx: x is a drive plate
Ix: x was inefficiently repaired)

Simon's only sister is psychic
$\exists x[(Sxa \wedge \forall y(Sya \rightarrow y=x)) \wedge Px]$
(a: Simon
Sxy: x is a sister of y
Px: x is psychic)

The F is the G
$\exists x[(Fx \wedge \forall y(Fy \rightarrow x=y)) \wedge$
$\exists z((Gz \wedge \forall x'(Gx' \rightarrow x'=z)) \wedge x=z)]$