PY1003: Introduction to Logic
Lecture Number 18
Revision (I)

(1) Truth trees and their uses

The basic idea: we take a sentence. According to some strict rules, we construct a 'tree' beneath it. A branch of the tree *closes* iff it contains both a sentence and its negation. Every open branch of a tree represents a way in which the starting sentence can be true.

The truth tree method can also be applied to sets of sentence. In this case, every open branch of the tree represents a way in which the sentences in the set can be true together.

<u>Proofs using truth trees:</u>

- A single sentence is *consistent* iff its truth tree has at least one open branch
- A single sentence is a *contradiction* iff its truth tree has no open branches
- A single sentence is a *tautology* iff the truth tree for its negation has no open branches.
- A set of sentences is *consistent* iff its truth tree has at least one open branch
- To demonstrate that an argument is *valid*, take the set of sentences consisting of the premisses, and the negation of the conclusion. The argument is valid iff the truth tree for this set of sentence has no open branch.

(2) Truth tree rules for sentential connectives

The basic rules:

$$\begin{array}{c} \neg\neg A \ \surd \\ | \\ A \end{array} \qquad \begin{array}{c} A \wedge B \ \surd \\ | \\ A \\ B \end{array} \qquad \begin{array}{cc} \multicolumn{2}{c}{A \vee B \ \surd} \\ / & \backslash \\ A & B \end{array} \qquad \begin{array}{cc} \multicolumn{2}{c}{A \rightarrow B \ \surd} \\ / & \backslash \\ \neg A & B \end{array} \qquad \begin{array}{cc} \multicolumn{2}{c}{A \leftrightarrow B \ \surd} \\ / & \backslash \\ A & \neg A \\ B & \neg B \end{array}$$

The rules for negated sentential connectives:

$$\begin{array}{cc} \multicolumn{2}{c}{\neg (A \wedge B) \ \surd} \\ / & \backslash \\ \neg A & \neg B \end{array} \qquad \begin{array}{c} \neg (A \vee B) \ \surd \\ | \\ \neg A \\ \neg B \end{array} \qquad \begin{array}{c} \neg (A \rightarrow B) \ \surd \\ | \\ A \\ \neg B \end{array} \qquad \begin{array}{cc} \multicolumn{2}{c}{\neg (A \leftrightarrow B) \ \surd} \\ / & \backslash \\ A & \neg A \\ \neg B & B \end{array}$$

Remember that:

- You must tick a sentence after you have applied a rule to it.
- You must always apply a rule to the main connective of a sentence.
- Each new step must be represented by a vertical or slanting line on the tree
- You should usually apply non-branching rules first.
- A tree must not fork into more than two branches.
- A branch of a tree closes only if that branch contains a sentence, and its negation as *isolated sentences.* For instance, a branch will not close merely because it contains the sentences '$\neg A \to B$' and 'A'.

(3) Interpretations, models and counterexamples (I)

Within sentential logic, an *interpretation* of a set of sentences is an assignment of truth-values to the atomic sentences that occur within that set.

A *model* of a set of sentences is an interpretation that makes all of the sentences in that set true.

A *counterexample* to an argument is an interpretation that makes the premisses true and the conclusion false. Thus a counterexample for an argument is a model for the set of sentences consisting of the premisses of the argument, and the negation of its conclusion.

In order to construct a model using a truth tree, we select any open branch of the tree. If an atomic sentence occurs on that branch, we assign the sentence the truth-value True. If negation of an atomic sentence occurs on that branch, we assign the sentence the truth-value False. If an atomic sentence occurs in the set of starting sentences, but not on the selected branch, we assign it an arbitrary truth-value.

Example:

```
      A → B √
     ¬ (C v B)  √
      (A v D) v ¬ B
          |
         ¬C
         ¬ B
         /   \
       ¬A    B
       /  \
  A v D   ¬B*
  /    \
 A      D
```

Model constructed using the branch*:

(I) $A = F$
(I) $B = F$
(I) $C = F$
(I) $D = F$

Notice that D is assigned an arbitrary truth-value.

(4) Tree rules for predicate logic

Universal instantiation:

$\forall x\ Fx\ \surd a$
|
Fa

(where a is any individual letter)

Existential instantiation:

$\exists x\ Fx\ \surd$
|
Fc

(where c is a NEW individual letter.)

The rules for negated quantifiers:

$\neg \forall x\ Fx\ \surd$
|
$\exists x\ \neg Fx$

$\neg \exists x\ Fx\ \surd$
|
$\forall x \neg Fx$

Remember that:

- A sentence does *not* become used up after you apply the rule for Universal Instantiation.
- When possible, use *old* letters to instantiate universal quantifiers
- Apply the rules for negated quantifiers first, followed by the rule for existential instantiation, followed by the rule for universal instantiation.

(5) Interpretations, models and counterexamples (II)

In predicate logic, an *interpretation* is a statement of a domain, together with an assignment of predicate letters to the individual letters that occur within the domain.

As in sentential logic, a *model* for a set of sentences is an interpretation that makes the sentences true together. And a *counterexample* for an argument is an interpretation that makes the premisses true and the conclusion false.

To construct a model for a set of sentences, select an open branch of its tree. As the domain of the model, write down every individual letter that occurs on the selected branch. For instance, our domain might look like this:

D = {a, b, c}

Then take look at the atomic sentences and negated atomic sentences that occur on the selected branch. (Remember that in predicate logic, an atomic sentence consists of single predicate letter following by an appropriate number of individual letters. e.g. Fa, Lab.) If an atomic sentence occurs on the branch, assign it the truth-value True. If the negation of the atomic sentence occurs on the branch, assign the sentence the truth-value False.

Finally, see whether any further atomic sentences can be formed using the letters in the domain, and the predicate letters that feature in the starting sentences. Assign any such atomic sentence an arbitrary truth-value.

Example:

```
           ∀x (Fx → Gx) √a √b
           ∃x Gx √
           ¬ ∀x Fx  √
              |
           ∃x¬Fx √
              |
             Ga
              |
            ¬Fb
              |
           Fa → Ga √
          /          \
       ¬Fa            Ga
         |             |
     Fb → Gb √    Fb → Gb √
     /    \        /      \
  ¬Fb     Gb    ¬Fb       Gb*
```

Model formed using branch*

D= {a, b}

I (Gb) = T
I (Ga) = T
I (Fb) = F
I (Fa) = T

Note that Fa is assigned an arbitrary truth-value.