Modal Logic

Checking the background

- In this course we assume basic knowledge of propositional and predicate logic.

  – Can you read *formulas*?

  – Do you know *truth tables*?

    Derivations? (but this course does not focus much on derivations)

- ***Refutation trees***?

  – What is the difference between *truth* and *validity*?

    What is *logical validity*?

Formal language

and = & , ∧

or = v

if…, then… = → ,

not = - , ⌐

if and only if = ↔

Formulas

Read the following formulas:

P & Q

P V R

-T

Q → -R

(-P & - -Q)

(P V -R) → P

(P V R) → (R V -P)

(P& -Q) → P

Q → - -R

Truth Tables

P	Q	¬ P	P ∧ Q	P ∨ Q	P → Q	P ↔ Q
False	False	True	False	False	True	True
False	True	True	False	True	True	False
True	False	False	False	True	False	False
True	True	False	True	True	True	True

Truth tables

1. Give the truth tables of:

   &, V, -, →

2. Give the truth tables of:

   - -P; -P & Q; P V -Q; P → -R; (P&Q) → R, ...

- Be sure that you know and remember the language and the **truth tables** for propositional logic.

    → Necessary!

- **Semantic trees** (also called *refutation trees* or *tableaux*) apply truth tables.

    - We use semantic trees in modal logic.

Logical validity

- What is *logical validity*?

- What is the difference between *truth* and *validity*?

- Logical validity is a **relation** between premises and conclusion of an argument.

→ It is not the **truth** of the premises or the **truth** of the conclusion.

- An argument is <u>logically valid</u> iff:

  in any case in which the premises are true,

  also the conclusion is true.

- Note!

  An argument can be **valid** even if its premises and conclusion are **false**!

  and an argument can be **not valid** even if its premises and conclusion are **true**!

- Is the following argument valid?

P & -P

therefore,

Q

Semantic trees for propositional logic

- Semantic trees are a way **to prove** validities.

  → Proof theory

  (like natural deduction, axiomatic systems, sequents, etc.)

- They are ‘mechanical’ procedures.

- Semantic trees for propositional logic are easy if one knows **truth tables**.

  → Knowledge of truth tables for &, v, -, →, ↔ is assumed.

- A semantic tree is the **search for a counter-example** to a formula.

→ If the search is successful, then there is a counter-example.

→ If there is a counter-example, the original sentence/argument is not valid.

- Suppose we want to know whether the formula **A** is valid.

Then, we look for a **counter-**example to **A**.

We check whether **A** could be **false**.

Namely, whether **not A** could be true.

→ If **not A** could be true, then **A** is not always true. **A** could be false. **A** has counter-examples. So **A** is not logically valid.

- If, instead, the search for a counter-example fails, and there is no counter-example,

  then **not A** is never true.

  Namely, **A** is never false.

- Which means that A is always true.

  → **A** is logically valid.

- The search for counter-example fails if all options lead to contradictions.

A **contradiction** is a pair of the form p, -p (with p atomic)

→ The “options” are represented by different paths/branches on the tree.

Rules

- Semantic trees use rules for each logical constant.

  (conjunction, disjunction, etc…)

- For each logical constant there are <u>two</u> rules:
  1. when the formula containing it is **true**.
  2. when the formula containing it is **false** (<u>negated</u>).

- The rules immediately follow from the truth tables.

  → One can study the rules, or just recover them from the truth tables.

  (I suggest to do the latter, and memorize the rules while practicing.)

- Note:

  we indicate **falsity** by **negation**.

  “p is false” is written: **-p**

Example.

- Consider a conjunction (A & B)

- There is a rule for when: (A & B) is **true**.

- And there is a rule for when: (A & B) is **false**,

Namely for its <u>negation</u>: -(A & B)

- Each rule can be easily obtained by the truth tables for conjunction.

1. (A & B) is **true**.

   When is a conjunction (A & B) true?

2. -(A & B) (A&B is false).

   When is a conjunction (A & B) **false**?

   - Note: when you have the negation you ask when the **original** formula (A&B) is **false**, not when the negated one -(A&B) is false.

1. (A & B) *True*

- When is a conjunction (A & B) **true**?

  When **both** conjuncts are **true**.

- So we write **both** conjuncts below the formula.

(A & B)

    A

    B

2. -(A & B) *False*

- When is a conjunction (A & B) **false**?

When **at least one** conjunct is **false**.

So we write the false conjuncts (**negated**) as **two** different cases.

```
     -(A&B)
    /     \
 -A        -B
```

- For the other connectives the procedure is similar.

- By similar reasoning on truth tables you can recover the other rules.

General notions

- Note that trees are drawn upside down.
- In trees we can distinguish the “root”, “the leafs” (terminal points), the branch.

- One branch (path) is a way to go from a leaf to the root.

  Note that branches (patches) do not cross.

- Each branch is developed until:

  - a contradiction is reached (in a single path),

  or:

  - all formulas in the path/branch cannot be analyzed further.

- When a <u>branch/path</u> includes a **contradiction** we write an X below, and we say that the branch **closes**.

- If the <u>branch</u> cannot be analyzed further, and it does not have a contradiction, we write a vertical **arrow** below it and we say that the branch is **open**.

- If **all** branches **close** (have contradictions), then there is no way to make the counter-example true.

- If **all** branches close, we say that the **tree closes**.

→ If **at least one** branch remains open, **the tree is open**.

1		✓	$P \vee Q$		
2			$\sim P$		
3			$\sim Q$		
4	P	$1 \vee$		Q	$1 \vee$
5	X	$2, 4 \sim$		X	$3, 4 \sim$

Semantic trees

Examples

- Example:

  check whether the formula

  (p V -p) is valid.

- **First step:**

we NEGATE the formula.

- Because we are looking for a counter-example!
- We want to test the formula and see if the formula can be false.
- We want to see whether we can go against the formula.

So we write:

-(p V -p)

Then, we draw the tree.

So we apply the rules derived from truth tables, as presented above.

The formula that we have now is

-(p V -p)

Which is a **<u>negated</u> disjunction**.

- So we ask: when is a disjunction (p V -p) **false**?
  - → False, because (p V -p) is negated.

A disjunction is false if **both** disjuncts are **false**.

So we write **both negations** of the disjuncts below the formula.

-(p V -p)

-p

- -p

- Now we have two new formulas to consider: -p and - -p. We consider them in turn.

- Consider -p.

  What kind of formula is the negated formula p?

  p is just an atomic formula. Truth tables do not tell us anything about atomic formulas.

  Truth tables say that -p is true, if p is false. But to write that p is false, we use negation, -p, which is the initial formula. So we do not go anywhere.

  -p is then already completely analyzed. We stop here.

- Consider - -p.

  What kind of formula is -p?

  It is a **negation**. We want -p to be **false.**

  (Note we are considering - -p now!).

  Given truth tables, -p is **false** when p is true.

- So from - -p we get p.

  → Which is just double negation elimination!

- Concerning negations, in general if A is **atomic**:

i. If we have - -A, we write A.

(double negation elimination)

ii. If we have -A, we stop.

So we now have the following tree for (p V -p):

- (p V -p)

-p

- -p

p

- But there is a contradiction in the path! So we **close** it.

- Check (p V -p)

- (p V -p)

-p

- -p

p

X

- The contradiction shows that it is not possible to negate (p V -p)! There is no counter-example.

- So (p V -p) is logically valid.

  → It is just the excluded middle.

Other example.

- Check: (p & q)

- **First step:** we negate the formula.

-(p & q)

- Then we draw the tree following the rules.

First formula: -(p & q)

When is a conjunction false?

When **at least one** conjunct is **false**.

→ “At least one” not “both”, so we have two cases now.

- Check: (p & q)

```
      -(p&q)
      /       \
   -p          -q
```

- At this point there is nothing else we can do.

  -p and -q cannot be analyzed further. So we stop.

- There is no contradiction in the paths. Both branches are open.

  So -(p&q) could be true.

  There are counter-examples to the initial formula.

  (p & q) is not valid.

Semantic trees for arguments

- How do we check **arguments** instead of single formulas?

- First, we write, in column, all premises, and the negation of the conclusion.

  → This is a counterexample to the validity of an argument.

- Then we proceed as usual.

- For example:

  Check whether the following argument is valid:

  $p \& q, -p \vdash q$

- Argument: p&q, -p ├ q

- We write:

  p&q

  -p

  -q

  ...and then proceed with the tree as before.

Counter-examples

- From open trees you can build counter-examples to the initial formula

  by considering the atomic formulas appearing in the **open** branch.

- As usual:

  p = p is **true**

  -p = p is **false**.

For example, we know that the tree for (p & q) is open.

- Check: (p & q)

```
   - (p & q)
    /       \
  -p         -q
```

- There are **two open branches** here,
  so there are **two** counter-examples.

A first counter-example is given by the **left** path:

-p (so p must be **false**. p = 0)

Another counter-example is given by the **right** path:

-q (so q must be **false**. q=0)

- Consider the first (left) branch.

  The **left** path gives p as false, p = 0.

  But what is the value of q, in the left path?

  - It does not matter. It can be true or false. To have a counterexample to (p&q) it is enough that p is false.

    → Similarly for q.

- These undetermined values can be given no value, or an arbitrary value (for example false).

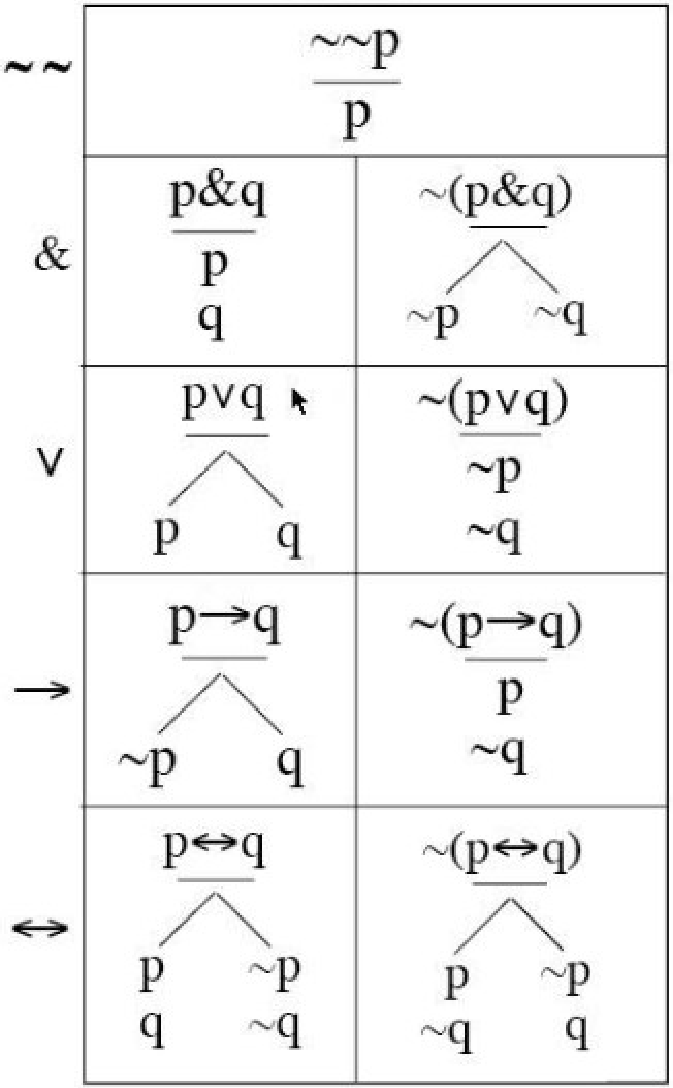
- So, when is (p & q), the initial formula, false?

  When p = 0   or   when q=0.

  As expected.

All propositional rules

Rules



- To use the propositional trees in modal propositional logic, it is enough that we add a specification of the world in which the formula is true or false.

→ As we know, worlds are inert and basically useless for propositional logic.

- For example,
  test (A&-B)

- First: we negate the formula in one world, say w:

  -(A&B)   (w)

- Then we just proceed taking trace of the world in which we are supposed to be.

- (A & -B)   (w)

|

-A   (w)

|

--B   (w)

|

B   (w)

Negation (~): If an open path contains both a formula and its negation, place an '*X*' at the bottom of the path.

Negated Negation (~ ~): If an open path contains an unchecked wff of the form $\sim\sim\phi$, check it and write ϕ at the bottom of every open path that contains this newly checked wff.

Conjunction (&): If an open path contains an unchecked wff of the form $\phi \,\&\, \psi$, check it and write ϕ and ψ at the bottom of every open path that contains this newly checked wff.

Negated Conjunction (~&): If an open path contains an unchecked wff of the form $\sim(\phi \,\&\, \psi)$, check it and split the bottom of each open path containing this newly checked wff into two branches, at the end of the first of which write $\sim\phi$ and at the end of the second of which write $\sim\psi$.

Disjunction (∨): If an open path contains an unchecked wff of the form $\phi \vee \psi$, check it and split the bottom of each open path containing this newly checked wff into two branches, at the end of the first of which write ϕ and at the end of the second of which write ψ.

Negated Disjunction (~∨): If an open path contains an unchecked wff of the form $\sim(\phi \vee \psi)$, check it and write both $\sim\phi$ and $\sim\psi$ at the bottom of every open path that contains this newly checked wff.

Conditional (→): If an open path contains an unchecked wff of the form $\phi \rightarrow \psi$, check it and split the bottom of each open path containing this newly checked wff into two branches, at the end of the first of which write $\sim\phi$ and at the end of the second of which write ψ.

Negated Conditional (~→): If an open path contains an unchecked wff of the form $\sim(\phi \rightarrow \psi)$, check it and write both ϕ and $\sim\psi$ at the bottom of every open path that contains this newly checked wff.

Biconditional (↔): If an open path contains an unchecked wff of the form $\phi \leftrightarrow \psi$, check it and split the bottom of each open path containing this newly checked wff into two branches, at the end of the first of which write both ϕ and ψ, and at the end of the second of which write both $\sim\phi$ and $\sim\psi$.

Negated Biconditional (~↔): If an open path contains an unchecked wff of the form $\sim(\phi \leftrightarrow \psi)$, check it and split the bottom of each open path containing this newly checked wff into two branches, at the end of the first of which write both ϕ and $\sim\psi$, and at the end of the second of which write both $\sim\phi$ and ψ.