HANDOUT #4 – PROPOSITIONAL LOGIC –TRUTH TREES

Truth tables provide us with a mechanical method for determining whether a proposition, set of propositions, or argument has a particular logical property. For example, we can show that an argument is deductively valid (or invalid) using the truth-table method. The problem with truth tables is that they are extremely cumbersome when dealing with formulas that have more than two propositional letters. While a truth table for a proposition with two propositional letters only has *four* rows, a truth table for a proposition with three propositional letters has *eight*, a proposition with four letters has *sixteen*, and so on. The advantage of **truth trees** is that it is a *decision procedure* whose complexity is not a function of the number of propositional letters in the formula being analyzed.

Truth Trees: The Setup

Truth trees first begin with an initial setup involving *three* columns:

(1) for numbering the propositions,
(2) writing (stacking) the propositions,
(3) justification of propositions.

To illustrate, consider the following two propositions: P∧R, M∧¬P.

1	P∧R	P
2	M∧¬P	P

Truth Trees: Some Basics in Decomposition (introducing ∧D)

Next, propositions are *decomposed* (broken apart) according to the conditions under which they are true. Decomposition occurs by applying specific decomposition rules. There are 9 decomposition rules, each applying to a specific *type* of decomposable proposition (see p.86). For example, conjunctions can only be decomposed using *conjunction decomposition* (∧D) and not any other decomposition rule.

When you decompose a proposition, you write the decomposed propositions under the stack, you number these propositions, and you cite the decomposition rule you used (along with the line number of the proposition you are decomposing). Also, you place a check mark by the proposition you decomposed to indicate it has been decomposed.

1	P∧R✓	P
2	M∧¬P	P
3	P	1∧D
4	R	1∧D

The procedure continues until all propositions that can be decomposed are decomposed. That is, until you have a *fully decomposed tree*:

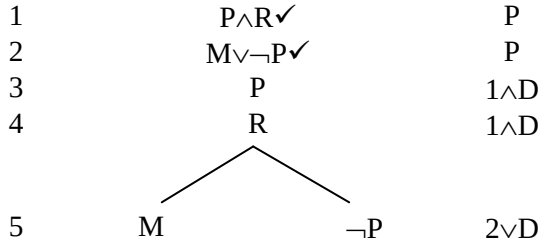
1	P∧R✓	P
2	M∧¬P✓	P
3	P	1∧D
4	R	1∧D
5	M	2∧D
6	¬P	2∧D

Truth Trees: Stacking vs. Branching (introducing ∨D)

There are three major kinds of decomposition rules: stacking, branching, and branching and stacking. When decomposing a proposition **P**, a *stacking rule* represents that there is one condition in which **P** is true, a *branching rule* represents that there are three conditions in which the **P** is true, and *a branching and stacking rule* represents that there are two conditions in which **P** is true. All of these rules *look* different. Consider the following propositions: P∧R, M∨¬P

1	P∧R✓	P
2	M∨¬P	P
3	P	1∧D
4	R	1∧D

Since P∧R (at line 1) is a conjunction, and 'P∧R' is true if 'P' is true and 'R' is true, we can represent this by stacking 'P' and 'R' under the existing set of propositions. In contrast, 'M∨¬P' is a disjunction and is true if either 'M' is true or '¬P' is true. This is represented by creating two branches (or paths) that break from the stack of propositions.

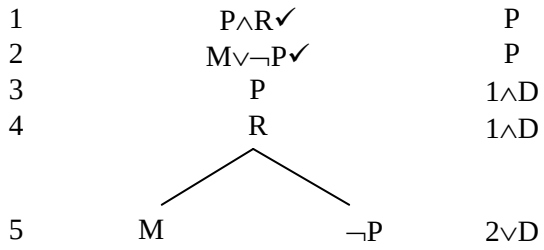


Truth Trees: Some Terminology

Branch	All the propositions obtained by starting from the bottom of the tree and reading *upward* through the tree.
Fully decomposed branch	A branch is *fully decomposed* when all propositions in the branch that can be decomposed have been decomposed.
Partially decomposed branch	A branch is *partially decomposed* when there is at least one proposition in the branch that has not been decomposed.
Closed Branch	A branch containing a proposition **P** and its literal negation **¬P**. A closed branch is represented by an **X**.
Open Branch	An *open branch* is a branch that is not closed. That is, a branch that *does not* contain a proposition **P** and its literal negation **¬P**.

Completed Open Branch	A *completed open branch* is a fully decomposed branch that is not closed. That is, it is a fully-decomposed branch that *does not* contain a proposition and its literal negation. An open branch is denoted by writing an **0** at the bottom of the tree.
Completed Open Tree	A tree is a *completed open tree* if and only if it has at least one completed open branch. That is, a tree is a completed open tree if and only if it contains at least one fully decomposed branch that is not closed. A completed open tree is a tree where there is at least one branch that has an **0** under it.
Closed Tree	A tree is *closed* when all of the tree's branches are closed. A closed tree will have an **X** under every branch.
Descending Decomposition Rule	When decomposing a proposition **P**, decompose **P** under *every open* branch that descends from **P**.

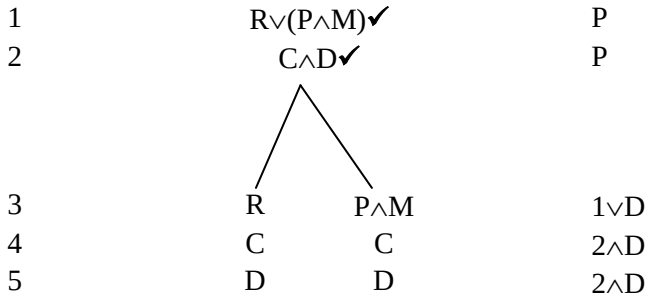
Illustration #1: Basic Terminology



Identify all of the branches. Determine whether the branches are fully decomposed or *not* fully decomposed (explain why). Are the branches open or closed (explain why)? If there are any open branches, are they completed open branches (explain why)? Is the tree a completed open tree or a closed tree?

Illustration #2: Of Descending Decomposition Rule

The descending rule states that when decomposing a proposition **P**, decompose **P** under *every open* branch that descends from **P** (see pp.100–103). This means that when you decompose **P** you decompose it under *every open branch* that descends from **P**. You do *not* decompose **P** under closed branches or under branches that do not descend from **P**. Consider the following partially decomposed tree:



Next, compare the following two trees, which are completions of the above tree.

Which one of these trees violates the descending decomposition rule (and why)?

Tree #1

1	R∨(P∧M)✓		P
2	C∧D✓		P
3	R	P∧M✓	1∨D
4	C	C	2∧D
5	D	D	2∧D
6	**0**	P	3∧D
7		M	3∧D
		0	

Tree #2

1	R∨(P∧M)✓		P
2	C∧D✓		P
3	R	P∧M✓	1∨D
4	C	C	2∧D
5	D	D	2∧D
6	P	P	3∧D
7	M	M	3∧D
	0	**0**	

Classroom Exercises

1. P∨(R∨D)
2. P∧(R∧D), Z∧M
3. W∧¬W, Z∨(D∨R)

Truth Trees: Remaining Decomposition Rules

There are 9 decomposition rules, each applying to a specific *type* of decomposable proposition (see p.86). Choosing which decomposition rule to apply to a proposition is *entirely determined by the type of proposition it is*. If **P** is a conjunction **P∧R**, you apply ∧D. If **P** is a disjunction **P∨R**, you apply ∨D. If **P** is a negated conditional **¬(P→R)**, you apply ¬→D.

9 Decomposable Proposition Types		Decomposition Rule
conjunction	**P∧R**	∧D
disjunction	**P∨R**	∨D
conditional	**P→R**	→D
biconditional	**P↔R**	↔D
negated conjunction	¬(**P∧R**)	¬∧D
negated disjunction	¬(**P∨R**)	¬∨D
negated conditional	¬(**P→R**)	¬→D
negated biconditional	¬(**P↔R**)	¬↔D
double-negation	¬¬**P**	¬¬D

Classroom Exercises

1. ¬(P∨L), P→Z
2. P↔R, P→R
3. ¬P→(Z∧M), ¬(P→Z)

Truth Trees: Decomposition Strategies

Using the decomposition rules *blindly* will ultimately lead to a completed open or a closed tree, but a *strategic* use of these rules will lead to the same result in a timelier manner. There are four such strategic rules:

Strategic Rule #1	Use no more rules than needed.
Strategic Rule #2	Use rules that *close* branches.
Strategic Rule #3	Use *stacking rules* before *branching rules*
Strategic Rule #4	Decompose more complex propositions before simpler propositions.

Classroom Exercises

1. M∨(R∨V), P∧(R∧Z), ¬(P→Z),
2. Q→(Q→Q), ¬(P→P), ¬R→[L∨(F∨M)]
3. M↔(L↔Z), ¬(P∨L), ¬P→L

Truth Trees: Analysis

Truth trees can be used to determine various semantic properties about propositions, sets of propositions, and arguments. Using truth trees to do this requires that you (i) set up the tree in a specific way to test for a specific property (you can't just stack the propositions in every instance), (ii) know how a closed (or completed open) tree indicates a specific semantic property.

Sets of Propositions – Consistency

Consistent	A set of propositions {**P, Q, R, … Z**} is *consistent* if and only if there is at least one valuation where **P, Q, R, … Z** are true. A truth tree shows that {**P, Q, R, …, Z**} is *consistent* if and only if a complete tree of the stack of **P, Q, R, …, Z** determines a completed open tree. That is, if there is at least one completed open branch.
Inconsistent	A set of propositions {**P, Q, R, … Z**} is *logically inconsistent* if and only if there is no valuation where **P, Q, R, … Z** are jointly true. A truth tree shows that {**P, Q, R, …, Z**} is *inconsistent* if and only if a tree of the stack of **P, Q, R, …, Z** is a closed tree. That is, if all branches close.

Suppose you are testing **P, Q, Z** to see whether or not the set of propositions is consistent / inconsistent.

1	**P**	P
2	**Q**	P
3	**Z**	P
4	**Z**	
	0	

P, Q, Z is consistent

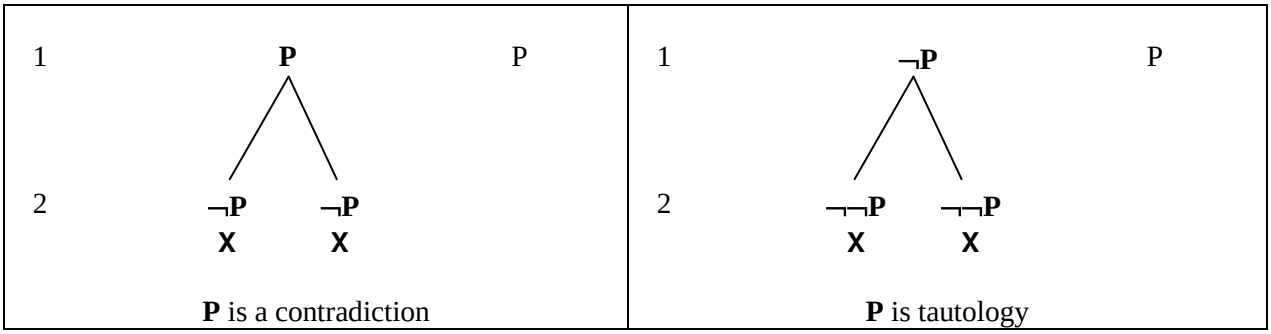
1	**P**	P
2	**Q**	P
3	**Z**	P
4	**¬Z**	
	X	
	¬Z	
	X	

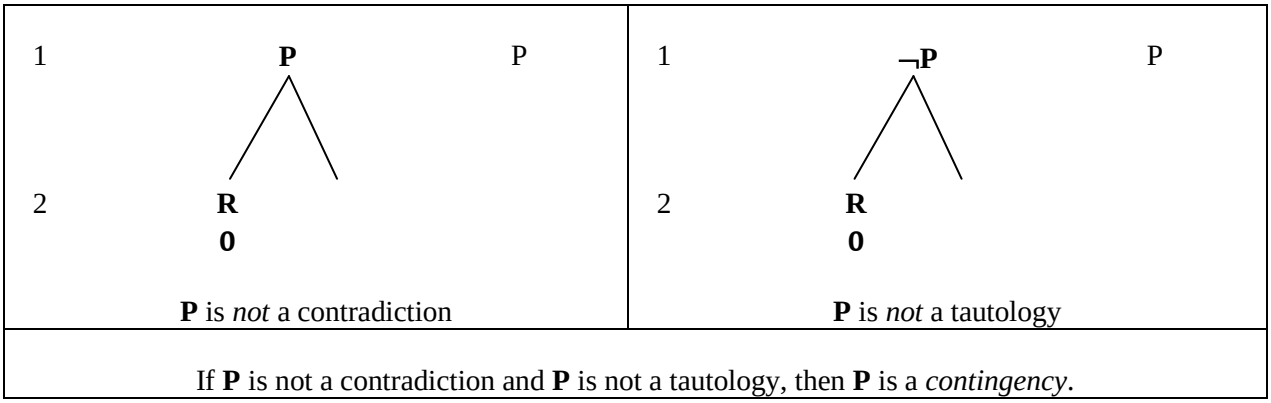
P, Q, Z is inconsistent

Propositions – Tautology, Contradiction, Contingency

Tautology	A proposition **P** is a *tautology* if and only if **P** is true under every valuation. A truth tree shows that **P** is a *tautology* if and only if a tree of the stack of **¬P** determines a closed tree.
Contradiction	A proposition **P** is a *contradiction* if and only if **P** is false under every valuation. A truth tree shows that **P** is a *contradiction* if and only if a tree of the stack of **P** determines a closed tree.
Contingency	A proposition **P** is a *contingency* if and only if **P** is neither always false under every valuation nor always true under every valuation. A truth tree shows that **P** is a *contingency* if and only if a tree of **P** does not determine a closed tree *and* a tree of **¬P** does not determine a closed tree.

Suppose you are testing **P** to see whether or not it is a contradiction, tautology, or contingency.





Pairs of Propositions – Equivalence

Equivalence	A pair of propositions **P, Q** is *equivalent* if and only if **P** and **Q** have identical truth values under every valuation. A truth tree shows that **P** and **Q** are *equivalent* if and only if a tree of **¬(P ↔ Q)** determines a closed tree.

Suppose you are testing **P** and **R** to see whether or not they are logically equivalent / not equivalent.

1 **¬(P↔R)** P 2 **P** **P** 3 **¬P** **¬P** **X** **X** **P, R** is an equivalence	1 **¬(P↔R)** P 2 **R** **0** **P, R** is a not an equivalence

Arguments – Validity

Validity	An argument **P, Q, …, Y ⊢ Z** is *valid* in PL if and only if it is impossible for the premises to be true and the conclusion false. A truth tree shows that an argument **P, Q, …, Y ⊢ Z** is *valid* in PL if and only if **P, Q, R, …, Y, ¬Z** determines a closed tree.

Suppose you are testing **P, Q, …, Y ⊢ Z** to see whether or not the argument is valid / invalid.

1 **P** P 2 **Q** P 3 **Y** P 4 **¬Z** P 4 **Z** **Z** **X** **X** **P, Q, …Y ⊢Z** is valid	1 **P** P 2 **Q** P 3 **Y** P 4 **¬Z** P 4 **Z** **0** **P, Q, …Y ⊢Z** is not valid

Classroom Exercises

Propositions – Contingency, Tautology, Contradiction
Construct a truth tree (or trees) for the following propositions and then state whether the tree shows the proposition to be a contingency, contradiction, or a tautology:
1. P∧¬¬P
2. P∨¬¬P
3. P∨(Q∨¬Q)

Sets of Propositions – Consistency

Construct a truth tree (or trees) for the following set of propositions and then state whether the tree shows the set of propositions to be consistent or inconsistent:
1. $T \rightarrow [(Q \vee R) \vee (\neg S \vee M)]$, $(P \vee Q) \vee F$, $(P \rightarrow L) \vee [W \leftrightarrow (\neg S \vee S)]$, $P \rightarrow Q$, $P \wedge \neg Q$
2. $L \vee \neg\neg Z$, $\neg(S \rightarrow P)$, $(T \vee M) \wedge A$, $Q \rightarrow \neg W$, $\neg(S \vee P)$

Pairs of Propositions – Equivalence
Construct a truth tree (or trees) for the following set of propositions and then state whether the tree shows the set of propositions to be consistent or inconsistent:
1. $P \rightarrow Q$, $\neg P \vee Q$
2. P, $\neg\neg P$

Arguments – Validity
Construct a truth tree (or trees) for the following set of propositions and then state whether the tree shows the set of propositions to be consistent or inconsistent:
1. $\neg(F \rightarrow Z) \vdash \neg(F \rightarrow Z)$
2. $P \rightarrow [R \rightarrow (S \wedge W)]$, $P \wedge \neg W \vdash \neg R$