Appendix ♡

altracts.org presents:

an ♡Ξ production

The Perfect Light of Reason:

A Guide to Reality

(because logic is elementary, my dear!)
(version 1.00)

A GoBrightly Booklet

Panglow Press, Planet Earth, Western Spiral Rim, Milky Way Galaxy

REASON

Introduction

Everyone can live better by learning how to think clearly. In English, this is called 'logical thinking' or 'correct reasoning'. In ☮T, it is called T!_. Symbolic logic is a kind of calculus that is easier than algebra and much more useful. In this age of growing computer technology full of information, misinformation, and disinformation, the need to teach symbolic logic early grows daily. Study of symbolic logic gives people the power to organize their thoughts and to see nonsense as nonsense. Computers use the same logic that is shown in this quire. Future problems in design and programming will only be solved by people who know logic.

Much of ⦸T→ is built on nonsense. People who learn to think correctly will be able to make the world better. They will also learn how to deal with people who are stuck in unclear thought. T! is necessary to remake the world into a ☮T world.

Unit 0 – Basic Ideas

What is logic?

Logic is a basic science. It helps people to think clearly. It also helps people to find good answers using organized thought. Logic is about organized thinking.

<u>*Logic* is the science of knowing how ideas are connected.</u>

What is symbolic logic?

<u>*Symbolic logic* is the science of showing how ideas are connected.</u>

Symbolic logic is like math. Math is about numbers. Symbolic logic is about words and ideas. This book is about symbolic logic of *propositions* (☐!CD).

<u>*Propositions* are sentences that are either true or not true.</u>

Calculations can be done with numbers. Calculations can also be done with propositions. This is called *deduction* (T!↓). This system will show you how to do deduction using propositions.

<u>*Deduction* is calculation done with ideas.</u>

Math uses symbols. Symbolic logic also uses symbols. Look at the sentence, "One plus one equals two". That is easy to read, but math uses symbols to make the ideas easier to read and easier to work with (1 + 1 = 2). Like math, symbolic logic changes propositions into symbols.

Propositions

Here are some propositions:

0. The pen is red.
1. Television is a fad.
2. Five quarters make a whole.
3. Nothing is certain.

Proposition 2 is clearly false. Proposition 3 is used as everyday wisdom, but after you study and understand T!, you will know that some important things are certain because you will be able to *prove* (CT!↓) them (to those who understand them).

Questions and commands are *not* propositions. Only propositions are good for doing symbolic logic. The symbolic logic in this quire is sometimes called *propositional calculus* or *sentential calculus*. Math is a way to find answers using numbers. Symbolic logic is a way to find truth using propositions.

Arguments

Logic uses *arguments* (CT!). An argument is made of two or more propositions. At least one of the propositions is a *conclusion* (!.). A conclusion is what the argument is meant to show is true. At least one of the propositions is a premise (→!). A premise helps to support a conclusion.

Here is a sample argument:

0. I love you, and you love me.	(premise)	0. (I_♡_U)_⋀_ (U_♡_ I)	(→!)
1. Therefore, I love you.	(conclusion)	1. T:_I_♡_U	(!.)

Do not confuse the meaning of 'argument' as we use it in logic with the meaning of 'argument' as it is often used in everyday speech. In everyday speech, 'argument' often means a talk where someone gets angry. That is not the meaning of 'argument' here.

There are good arguments, and there are bad arguments. Good arguments use true premises and T! to support their conclusions. Bad arguments fail to support their conclusions because they use premises that are not true or because they use bad reasoning.

In symbolic logic, we study (C⋀C) arguments with perfect forms. These are called *valid arguments* (☮T#CT!).

Valid and Sound Arguments

Logic is about the forms of arguments. Right now, we will not worry about true premises. We will study some perfect arguments forms. Arguments with perfect forms are *valid* (☮T#).

An argument is *valid* just in case, if the premises are true, then the conclusion *must* be true.

If you do not understand valid right away, do not worry. Sometimes the idea takes a while to understand.

Do not confuse the meaning of 'valid' used here with the meaning often used in everyday speech. In everyday speech, 'valid' often means 'good' as in 'good argument'. Bad arguments can be valid. We can make arguments with premises that are not true but still have valid logical forms. An argument with a premise that is not true is a bad argument, but it might still be valid. The idea of valid is about the forms of arguments and not about the truth of the premises.

The correct term to describe what people mean by 'valid' in everyday speech is *sound* (T⋀☮T#).

An argument is *sound* just in case it is valid *and* its premises are true.

The conclusion of a sound argument must be true.

To understand 'valid' better, let us look again at the sample argument:

0. I love you, and you love me.	(premise)
1. Therefore, I love you.	(conclusion)

This argument is valid. *If* the premise is true, then the conclusion must be true. The premise might not be true, but the truth of the premises does not matter for the logic. What matters with valid arguments are the conclusions that we can deduce *if* the premises are true. *If* the premise of the argument above is true, then the conclusion *must* be true. We can deduce the conclusion from the premises using logic.

It is easier to understand 'valid' if we ignore the meanings of the propositions. We can just look at the forms of the propositions to see the forms of arguments. "I love you, and you love me" has the form of "A and B". What A and B mean is not important. The form of the sentence is important. A and B each could be *any* proposition. To learn symbolic logic, we are going to ignore the meanings of the propositions and look only at the forms of the arguments.

REASON

In a standard symbolic-logic textbook, the argument might look like this:

A ⋀ B

∴ A

⋀ means "or" (just like it does in ☮T), and ∴ means "therefore".

In ☮T, the argument will look like this:

⓪_⋀_①

T:_⓪

T: means "therefore" in ☮T _.

Unit 1 – Formulas

Symbolic logic uses symbolic writing systems. There are many systems. We are going to use the ☮T system. There are different kinds of symbols: *variables*, *modes and joints*, *boxes and a spacer*, and *T:_*.

Variables

A *variable* is a symbol that can stand for anything. It can be a word, a number, a letter, a sentence, or a part of a sentence. In math, variables stand for numbers. In this quire, variables stand for propositions.

The basic way to show variables in ☮T is with the ☮TCD for 'how much or 'how many', ʔ#. Often, more than one variable is needed, so each different one gets a different number. Three different variables can be shown as ʔ#0, ʔ#1, and ʔ#2. To make them look nicer and to make them easier to understand, we will show them as circled numbers, ⓪, ①, and ②, respectively. The T☮ sounds are the same. Ⓧ is a non-specific variable. It can be any variable number.

Logical Operators – Modes and Joints

The important logical operators are already ☮T LETs. They are:

Logical Operator	English Meaning	Type
⦸	'not'	mode
⋀	'and'	joint
⋁	'or' (inclusive)	joint
⊃	'if-then', 'implies'	joint

Modes change one variable. Joints join two variables. Four operators are enough for now.

Formulas

A variable stands for a proposition. Variables correctly put together with modes or joints also stand for propositions. These symbolic propositions are *formulas* (T!☐CD).

A formula is a proposition shown as logical symbols.

Using a formula in an argument is like saying that the formula is true.

Spacers and Boxes

_ is the spacer. Spacers connect joints to variables and also connect T: to a conclusion. Some formulas need boxes to make the logical connections clear. () is a box, and ~~()~~ is a box. Formulas go inside of boxes. ⓪_⋀_① is a formula. Boxed, it looks like this: (⓪_⋀_①).

One variable alone is a *basic formula* (T!☐0). A formula that is not basic but does not need a box is a *simple formula* (T!☐1). A formula that needs at least one box is a *complex formula* (T!☐2).

Here are some sample formulas:

Formula	English Meaning	Type
⓪	⓪ is true.	Basic
⦸⓪	⓪ is not true.	simple
⓪_∧_①	⓪ is true, *and* ① is true.	simple
⓪_∨_①	⓪ is true, *or* ① is true.	simple
⓪_⊃_①	*If* ⓪ is true, *then* ① is true.	simple
(⓪_∧_①)_∨_②	⓪ and ① are both true, *or* ② is true.	complex
⓪_∧_(①_∨_②)	⓪ is true, *and* ① or ② is true.	complex

You can see that the two complex formulas have very different meanings. The boxes make the meanings clear. Variables inside of boxes connect with each other before they connect with anything outside of their boxes. () can do all of the work of boxes, but to be very clear, there is another way. () is used. Then ~~()~~ is used. Next comes (()). Then ~~(())~~, and so on.

How to Make ☮TT! Formulas

Logical symbols next to each other in a line are *strings* (←CD→). Not just any string is a formula. There are rules for making ☮TT! formulas.

0. Formulas may have in them only:
   a. variables (⓪,①,②, and so on)
   b. modes and joints (⦸, ∧,∨,⊃, and more that will come later)
   c. spacers and boxes (_, (), ~~()~~, (()), ~~(())~~, and so on)
1. One variable by itself is a formula. ⓪ is a formula. ① is a formula. ② is a formula.
2. One mode in front of a formula is a formula. ⦸⓪ is a formula. ⦸① is a formula. Since ⦸⓪ is a formula, ⦸⦸⓪ is a formula.
3. A boxed formula is a formula. ⓪ is a formula, so (⓪) is also a formula.
4. One joint flanked on each side by _ and then flanked by a formula on each side is a formula. ⓪_∧_① is a formula. ⓪ _∨_⦸① is a formula. (⓪_∧_①)_⊃_① is a formula.
5. A formula with a joint must be boxed before is it connected to a mode or a joint. (⓪_∧_①)_∧_② is a formula. ⦸~~(~~⓪_⊃_(②_∧_①)~~)~~ is a formula. ⓪_∧_①_∧_② is *not* a formula; it is just a string.

Unit 2 – Truth Tables

In math and logic, a *table* (##) is a grid (#) of rectangles (☐). (LET # shows a table.) Each line of rectangles from left to right is called a *row* (←☐→). Each stack of rectangles from top to bottom is called a *column* (↑ ☐ ↓). The rectangles are called *cells* (#☐). Cells hold information (!T) called *values* (#☐!T).

A truth table (T#) is a way to show all of the possible truth values of formulas. A cell in a truth table can have only one value, true (T). If a formula is not true, the cell of the truth table is left blank.

Here is a very simple truth table for just one variable:

	#⃝	
0.	T	#⃝ is true
1.		#⃝ is not true

The truth table above shows all of the possible truth values for #⃝, true and not true. It has two rows, one column, and two cells. One cell holds a value.

Here is a simple truth table for two variables:

	⓪	①	
0.	T	T	⓪ is true. ① is true.
1.	T		⓪ is true. ① is not true.
2.		T	⓪ is not true. ① is true.
3.			⓪ is not true. ① is not true.

The truth table above has four rows, two columns, and eight cells. Four of the cells hold values.

Truth tables are good for showing when formulas are T_. They are also good for showing the meanings of modes and joints. We have to show the meanings of our modes and joints: ⦸, ⋀, ⋁, and ⊃. Truth tables will show what the logical operators mean. The English and ☮T meanings and English explanations are there just to help you to understand. The actual meaning of each mode or joint is its truth table. (Actually, we *cannot* use English or ☮T for the meanings without making the meanings circular.)

Meaning of ⦸: ⦸#⃝ is true just in case #⃝ is not true.

(⦸#⃝_T)_⊃=_(#⃝_⦸T)

⦸T#

	#⃝	⦸#⃝	
0.	T		#⃝ is true so ⦸#⃝ is not true.
1.		T	#⃝ is not true so ⦸#⃝ is true.

We can see that ⦸#⃝ is only true when #⃝ is ⦸T and *vice versa*.

Meaning of ⋀: ⓪_⋀_① is true just in case ⓪ is true, *and* ① is true.

(̵(⓪_⋀_①)_T)̵_⊃=_(̵(⓪_T)_⋀_(①_T))̵

⋀T#

	⓪	①	⓪_⋀_①	
0.	T	T	T	⓪ is true, and ① is true, so ⓪_⋀_① is true.
1.	T			⓪ is true, and ① is not true, so ⓪_⋀_① is not true.
2.		T		⓪ is not true, and ① is true, so ⓪_⋀_① is not true.
3.				⓪ is not true, and ① is not true, so ⓪_⋀_① is not true.

Meaning of ⋁: ⓪_⋁_① is true just in case ⓪ is true, *or* ① is true.

(̵(⓪_⋁_①)_T)̵_⊃=_(̵(⓪_T)_⋁_(①_T))̵

∨T#

	⓪	①	⓪_∨_①
0.	T	T	T
1.	T		T
2.		T	T
3.			

⓪ is true, and ① is true, so ⓪_∨_① is true.
⓪ is true, and ① is not true, so ⓪_∨_① is true.
⓪ is not true, and ① is true, so ⓪_∨_① is true.
⓪ is not true, and ① is not true, so ⓪_∨_① is not true.

Meaning of ⊃: ⓪_⊃_① is true just in case ⓪ is not true, *or* ① is true.

(̵(⓪_⊃_①)_T)̵_⊃=_(̵(⓪_⃠T)_∨_(①_T))̵

⊃T#

	⓪	①	⓪_⊃_①
0.	T	T	T
1.	T		
2.		T	T
3.			T

⓪ is true, and ① is true, so ⓪_⊃_① is true.
⓪ is true, and ① is not true, so ⓪_⊃_① is not true.
⓪ is not true, and ① is true, so ⓪_⊃_① is true.
⓪ is not true, and ① is not true, so ⓪_⊃_① is true.

This last meaning may seem like a very strange one, but it works very well in symbolic logic.

How to make T#

0. Count the variables to find ⊕. 2^⊕ is how many rows the T# will need.
1. Add the number of variables to the number of all simple and complex formulas that will be shown in the T#. This is how many columns the T# will need.
2. Write the variables in order above the columns, one formula for each column. Then write the other formulas above the rest of the columns from simplest to most complex, one formula for each column.
3. Write T in each cell or leave it blank. Here is how to know when to write T in a cell:
   a. For the beginning variable, write T in the top 1/2 of the cells in its column. Leave the bottom 1/2 of the cells blank.
   b. For the next variable, write T in the top 1/4 of the cells in its column. Leave the next 1/4 of the cells blank. Write T in the next 1/4 of the cells. Leave the bottom 1/4 of the cells blank.
   c. For the next variable, write T in the top 1/8 of the cells in its column. Leave the next 1/8 of the cells blank. Write T in the next 1/8 of the cells. Leave the next 1/8 of the cells blank, and so on. (Do you see the pattern?)
   d. For each simple and complex formula, use the meanings of the mode and joints to find the truth value for each cell.

Example: Let us make an T# for (⓪_∨_①)_∧_⃠②.

4. There are three variables, ⓪, ①, and ②. 2^3 = 8, so our T# will need 8 rows.
5. Now we calculate the number of columns:
   a. three variables, ⓪, ①, and ②,
   b. two simple formulas, ⓪_∨_① and ⃠②,
   c. and one complex formula, (⓪_∨_①)_∧_⃠②, for a total of six columns.

REASON

1. Write the variables and other formulas in order above the columns, one formula for each column.

	⓪	①	②	⦸②	⓪_∨_①	(⓪_∨_⓪)_∧_⦸②
0.						
1.						
2.						
3.						
4.						
5.						
6.						
7.						

2. a. Write the values of the beginning variable. Write T in the top 1/2 of the cells in its column. Leave the bottom 1/2 of the cells blank.

	⓪	①	②	⦸②	⓪_∨_①	(⓪_∨_①)_∧_⦸②
0.	T					
1.	T					
2.	T					
3.	T					
4.						
5.						
6.						
7.						

3. b. Write the values of the next variable. Write T in the top 1/4 of the cells in its column. Leave the next 1/4 of the cells blank. Write T in the next 1/4 of the cells. Leave the bottom 1/4 of the cells blank.

	⓪	①	②	⦸②	⓪_∨_①	(⓪_∨_①)_∧_⦸②
0.	T	T				
1.	T	T				
2.	T					
3.	T					
4.		T				
5.		T				
6.						
7.						

3. c. Write the values of the last variable.

	⓪	①	②	⦸②	⓪_∨_①	(⓪_∨_①)_∧_⦸②
0.	T	T	T			
1.	T	T				
2.	T		T			
3.	T					
4.		T	T			
5.		T				
6.			T			
7.						

All of the cells for the variables on the beginning row have the value T.
All of the cells for the variables on the last row are blank.

3. d0. Use the meaning of ⦸ to write the values of ⦸②.

	⓪	①	②	⦸②	⓪_∨_①	(⓪_∨_①)_∧_⦸②
0.	T	T	T			
1.	T	T		T		
2.	T		T			
3.	T			T		
4.		T	T			
5.		T		T		
6.			T			
7.				T		

3. d1. Use the meaning of ∨ to write the values of ⓪_∨_①.

	⓪	①	②	⦸②	⓪_∨_①	(⓪_∨_①)_∧_⦸②
0.	T	T	T		T	
1.	T	T		T	T	
2.	T		T		T	
3.	T			T	T	
4.		T	T		T	
5.		T		T	T	
6.			T			
7.				T		

3. d2. Use the meaning of ∧ to write the values of (⓪_∨_①)_∧_⦸②.

	⓪	①	②	⦸②	⓪_∨_①	(⓪_∨_①)_∧_⦸②
0.	T	T	T		T	
1.	T	T		T	T	T
2.	T		T		T	
3.	T			T	T	T
4.		T	T		T	
5.		T		T	T	T
6.			T			
7.				T		

Finished!

Unit 3 – Our First Proof

Valid argument forms are rules. After we prove that an argument form is valid, we can always use it to deduce conclusions from premises. We will look at ten very special valid argument forms, and we will see how to prove that they are valid using truth tables. These forms are so special that they have their very own extended ☮T syllables.

The☮T rules (valid argument forms) are:

☮TT!>T#

#	English (or Latin)	LET	SOUND	PIC	Meaning
0.	*modus ponens*	M	더ᄀ (BON)	中	result, consequence
1.	*modus tollens*	Q	거ㅣ (DOL)	卄	vowel, toy, DOL Elder
2.	double negation	⊗	거ᄀ (NON)	巾	redundancy
3.	disjunctive syllogism	ε	ᄀㅣㅣ (DIL)	丰	choice
4.	commutation	z	머ᄀ (GON)	巾	consonant, clothing, accessory, GON Elder
5.	addition	ℓ	ᄀㅏᄀ (DAD)	吉	extra, abundant, common
6.	simplification	s	ᄀㅣᄀ (DIN)	巾	simplicity, ease
7.	hypothetical syllogism	h	ㅣㅓㅣ (LOL)	車	hypothesis
8.	constructive dilemma	₳	ᄀㅕㅣ (DUL)	卄	problem
9.	transformation	Ξ	ㅣㅏᄀ (LAN)	巾	change, transformation, segue, purpose

Let us look at rule 0, M (*modus ponens*).

M (*modus ponens*)

0.	⓪_⊃_①	→!	0.	If ⓪, then ①.	premise
1.	⓪	→!	1.	⓪.	premise
2.	T:_①	0, 1, M	2.	Therefore, ①.	0, 1, *modus ponens*

M is a very powerful rule. We can prove that M is valid by understanding what its truth table shows.

MT#

	⓪	①	⓪_⊃_①	
0.	T	T	T	Premise 0 is true. Premise 1 is true.
1.	T			Premise 0 is not true. Premise 1 is true.
2.		T	T	Premise 0 is true. Premise 1 is not true.
3.			T	Premise 0 is true. Premise 1 is not true.

Look at the rows where premise 0 (⓪_⊃_①) is true. These are rows 0, 2, and 3. Now look at the rows where premise 1 (⓪) is true. These are rows 0 and 1. The only row where both premises are true is row 0. On row 0, the conclusion of M (①) is also true. Using the definition of valid, MT# proves that M is valid.

We have certainty! We can be certain that if all of the premises of any M argument are true, then the conclusion of that argument is also true. (It *has* to be!) Something really is certain! Now we can use M as a rule.

It is easier to understand why M is valid when we look at M arguments.

0. If Bob went on vacation, then he had a good time.
1. Bob went on vacation.
2. Therefore, he had a good time.

You might wonder if Bob could go on vacation and have a bad time. Maybe he could, but all it would mean here is that premise 0 is false. The form of the argument is still valid even if the argument is not sound. Remember that the premises do not have to be true for an argument to be valid (but they do have to be true for the argument to be sound).

Here is another example:

0. If she enjoys school, then she will learn well.
1. She enjoys school.
2. Therefore, ____________________________________.

You can fill in the blank.

Unit 4 – Proofs Using Rules

We can use T# to prove that all of the argument forms listed in ☮TT!>T# are valid. Rule 1 is Q (*modus tollens*).

Q (*modus tollens*)

0. ⓪_⊃_① →!
1. ⃠① →!
2. T:_⃠ ⓪ 0, 1, Q

QT# is a little more complex than the MT# because we also need truth values for ⃠⓪ and for ⃠①.

QT#

	⓪	①	⃠⓪	⃠①	⓪_⊃_①	
0.	T	T			T	Premise 0 is true. Premise 1 is not true.
1.	T			T		Premise 0 is not true. Premise 1 is true.
2.		T	T		T	Premise 0 is true. Premise 1 is not true.
3.			T	T	T	Premise 0 is true. Premise 1 is true.

Look at the rows where premise 0 (⓪_⊃_①) is true. These are rows 0, 2, and 3. Now look at the rows where premise 1 (⃠①) is true. These are rows 1 and 3. The only row where both premises are true is row 3. You can see that on row 3, the conclusion of Q (⃠⓪) is also true. On every row of QT# where the premises of Q are all true, the conclusion is also true. We have another proof! Q is valid.

Now we can use our two rules to do a more difficult proof.

0. If Bob went on vacation, then he had a good time.
1. If Bob is not happy, then he did not have a good time.
2. Bob is not happy.
3. Therefore, Bob did not go on vacation.

We need to write the premises as logical symbols. First, we find the most basic propositions and give them variables.

⓪ is 'Bob went on vacation'. ① is 'He had a good time'. ② is 'Bob is happy'.

Next, we write the premises as formulas using the variables and logical operators.

0. ⓪_⊃_①	→!	0. If ⓪, then ①.	Premise
1. ⃠②_⊃_⃠①	→!	1. If not ②, then not ①.	Premise
2. ⃠②	→!	2. Not ②.	premise

Now that we have correctly set up the premises, we can add lines to the proof until we get to the conclusion that we are looking for. Every new line must begin with T: to show that it was deduced. After every T: line, we need to list the lines used and the rule used to deduce the conclusion.

Here we go!

3. T:_⃠ ①	1, 2, M	3. Not ①.	1, 2, *modus ponens*

We can deduce that ⃠① because lines 1 and 2 have the form of the premises of M. The premises of M let us deduce the conclusion of M, in this case ⃠①.

Now, we can finish the proof.

4. T:_⃠ ⓪	0, 3, Q THC	4. Therefore, not ⓪.	0, 3, *modus tollens* QED

Lines 0 and 3 have the form of the premises of Q. From the premises of Q, we can deduce the conclusion of Q, in this case that ⃠⓪. We flourish the proof with THC, the ☮T word for QED.

Unit 5 – Logical Equivalence

Let us look at an argument that we cannot yet prove but which is clearly valid.

0. If Bob did not go on vacation, then he did not have a good time.
1. He had a good time.
2. Therefore, Bob went on vacation.

At first look, it seems like we can deduce the conclusion using Q. However, Q alone does not quite do the trick. Let us try the proof to see what the problem is.

⓪ is 'Bob went on vacation'. ① is 'He had a good time'.

0. ⃠⓪_⊃_⃠① →!
1. ① →!
2. T:_

Now we are stuck. The premises do not have the form of M or the form of Q. We cannot deduce anything from the premises using M or Q. If premise 1 was 'He did *not not* have a good time', we could write it as ⃠⃠①, and deducing the conclusion using Q would be an instant slam dunk. However, the form of Q does not allow us to do that.

It is clear that if ① is true, then ⃠⃠① is true (and *vice versa*), but we do not yet have a rule that lets us deduce that. We need a new rule, ⊗ (double negation).

⊗ (double negation)

0. ㊉	→!	*and*	0. ⃠⃠㊉	→!
1. T:_⃠⃠㊉	0, ⊗		1. T:_㊉	0, ⊗

⊗ is a little bit special because it has two cases, but ⊗T# proves both cases.

⊗T#

	㊉	⃠㊉	⃠⃠㊉
0.	T		T
1.		T	

The truth values for ㊉ and ⃠⃠㊉ are the same on every line of ⊗T#. ㊉ and ⃠⃠㊉ are *logically equivalent* (T!=). This means that we can deduce ㊉ from ⃠⃠㊉ and ⃠⃠㊉ from ㊉.

Logical Equivalence

When two formulas have the same truth values on every row of their T#, the formulas are logically equivalent. Logically equivalent formulas can be deduced from one another.

<u>Two formulas are *logically equivalent* just in case both formulas have the same truth values on every row of the same truth table.</u>

Now we can finish the proof that we started this unit with.

0. ⃠⓪_⊃_⃠① →!
1. ① →!
2. T:_⃠⃠① 1, ⊗
3. T:_⃠⃠⓪ 0, 2, Q
4. T:_⓪ 3, ⊗ THC

Logical equivalence gives us much more than just ⊗_. In our system of logic, the meanings of the modes and joints come from T#. If two formulas have the same T#, they have the same meaning.

Minimal Systems

We have three joints so far, but we only need one of them. It could be ⋀, ⋁, or ⊃ (your choice). (We need ⦸ no matter what!)

That is a big claim. It will need to be proved.

Claim: ⓪_⋀_① is logically equivalent to ⦸(⦸⓪_⋁_⦸①)

	⓪	①	⦸⓪	⦸①	⦸⓪_⋁_⦸①	⦸(⦸⓪_⋁_⦸①)	⓪_⋀_①	
0.	T	T				T	T	
1.	T			T	T			
2.		T	T		T			
3.			T	T	T			THC

So far, so good. The truth values for ⓪_⋀_① and ⦸(⦸⓪_⋁_⦸①) are the same. The meaning of ⋀ can be used to show the meaning of ⋁ (and *vice versa*). We can throw out either one of those if we really want to (but we will keep them both because they are both very useful).

Claim: ⓪_⋀_① is logically equivalent to ⦸(⓪_⊃_⦸①)

	⓪	①	⦸①	⓪_⊃_⦸①	⦸(⓪_⊃_⦸①)	⓪_⋀_①	
0.	T	T			T	T	
1.	T		T	T			
2.		T		T			
3.			T	T			THC

It looks like we do not need ⊃ either (but we will keep it). Since ⓪_⋀_① is logically equivalent to both ⦸(⦸⓪_⋁_⦸①) and ⦸(⓪_⊃_⦸①), it is clear that ⦸(⦸⓪_⋁_⦸①) is logically equivalent to ⦸(⓪_⊃_⦸①). The truth tables prove it.

What this shows is that any one of the three joints can do the work of the other two. If we want a minimal system, we only need ⦸ and one joint. ⦸ with ⋀, ⦸ with ⋁, or ⦸ with ⊃, each as pairs can be used to prove all of the rules of our system.

Unit 6 – More Rules

We will now look at more rules. Rule 3 of ☮TT!>T# is ε (disjunctive syllogism).

ε (disjunctive syllogism)

0. ⓪_⋁_① →!
1. ⦸⓪ →!
2. T:_① 0, 1, ε

εT#

	⓪	①	⃠⓪	⓪_⋁_①
0.	T	T		T
1.	T			T
2.		T	T	T
3.			T	

Someone might look at ε and wonder if the form below is valid.

0. ⓪_⋁_① →!

1. ⃠① →!

2. T:_⓪

The form is valid. εT# *almost* proves that it is valid. We could name this form and make it a rule, but there is something better that we can do instead. We can prove rule 4, z (commutation), and use it to solve the problem.

z (commutation)

0. ⓪_⋀_① →!

1. T:_①_⋀_ ⓪ 0, z

and

0. ⓪_⋁_① →!

1. T:_①_⋁_ ⓪ 0, z

z is a little bit special because, like ⊗, there are two cases, one for ⋀ and one for ⋁. We can make two different rules, but there is no good reason for that. All the rule says is that for any ⋀ or ⋁ formula, you can swap the formulas on both sides of the joint. z is better as just one rule instead of two. zT# proves that both cases are valid.

zT#

	⓪	①	⓪_⋀_①	①_⋀_⓪	⓪_⋁_①	①_⋁_⓪
0.	T	T	T	T	T	T
1.	T				T	T
2.		T			T	T
3.						

Now we can deal with that argument that looks a lot like ε.

0. ⓪_⋁_① →!

1. ⃠① →!

2. T:_①_⋁_ ⓪ 0, z

3. T:_⓪ 2, 1, ε THC

The next rule is ℓ (addition).

☮TT! Games: ☮T is meant to be fun. Part of having fun is being creative. You already know enough about ☮TT! to make some games. Get a lot of cubes (like dice) and mark the sides with variables, modes, joints flanked by spacers, and sides of boxes. You can do this with stickers or just tape, too. Roll the dice and make formulas. Make your own rules for games. You can also write the symbols on tiles and play games like dominos. Cards are also fun to play with and learn from. Make a lot of cards with the symbols and make your own games.

ℓ (addition)

0. ⓪ →!
1. T:_⓪_∨_① 0, ℓ

ℓT#

	⓪	①	⓪_∨_①
0.	T	T	T
1.	T		T
2.		T	T
3.			

We are going to use ℓ and the next rule, s (simplification), to prove a deep truth about reality.

> **Cool Fact:** ☮TCD have basic meanings, but they do not have set meanings. After you learn about modes in Unit 9, you can set your own ☮T modes. For example, '=' basically means 'is', but in math, '=' means 'equals'. The two words work differently in English. 'Is' works only one way. 'Today is beautiful' is different from 'Beautiful is today' (unless you are Yoda). 'Equals' works both ways. '1 + 2 = 3' is the same as '3 = 1 + 2'. Mode is set by context. 'Context' means 'the situation'. When you talk about math, '=' means 'equals'. When you talk about the weather, '=' means 'is'. Each ☮T LET can have at least as many meanings as there are modes, and the modes are limited only by your imagination. That means that ☮T is an unlimited language. Cool, huh?

s (simplification)

0. ⓪_∧_① →!
1. T:_⓪ 0, s

sT#

	⓪	①	⓪_∧_①
0.	T	T	T
1.	T		
2.		T	
3.			

Now we are going to prove something amazing. First of all, we have to understand what a *contradiction* (⦸<T) is.

<u>*A contradiction* is a proposition that is true and not true at the same time.</u>

A contradiction has the logical form of ⓪_∧_⦸⓪.

Maybe you have heard that from a contradiction, all else follows. If you start with a contradiction, you can prove anything. This is true, and we can prove it!

Are you ready? Here we go!

0. ⓪_∧_⦸⓪ →!
1. T:_⓪ 0, s
2. ⦸⓪_∧_⓪ 0, z
3. T:_⦸ ⓪ 2, s
4. T:_⓪_∨_① 1, ℓ
5. T:_① 4, 3, ε

① could be *any* proposition. The premise could be *any* contradiction. If *any* proposition is true and not true at the same time, then *every* proposition is true (and not true) at that time.

Nonsense comes from nonsense. This tells us a lot about human society.

Do not panic. There is no problem with contradictions. Look again at the meaning of ⦸ (page 6). There is no truth value of any variable that would allow any proposition to be true and not true at the same time. A contradiction is not possible. Of course, it stands to reason that if something impossible is true, then everything is true (and not true) at the same time. However, 'impossible' *means* 'cannot be true'. A contradiction cannot be true.

Let us add ⊕_⋀_⃠⊕ and ⃠(⊕_⋀_⃠⊕) to ⃠T#.

	⊕	⃠⊕	⊕_⋀_⃠⊕	⃠(⊕_⋀_⃠⊕)
0.	T			T
1.		T		T

What we can see is interesting. ⊕_⋀_⃠⊕ has no truth value on every row (because it is impossible). Contradictions are never true. ⃠(⊕_⋀_⃠⊕) is true on every row. It is a very special kind of formula called a *tautology* (>T). A basic tautology is ⊕_⋁_⃠⊕. Tautologies are the opposites of contradictions. They are always true.

The last two rules from ☮TT!>T# are h (hypothetical syllogism) and ⩚ (constructive dilemma). You can make truth tables to prove that they are valid.

h (hypothetical syllogism)

0. ⓪_⊃_① →!
1. ①_⊃_② →!
2. T:_⓪_⊃_② 0, 1, h

⩚ (constructive dilemma)

0. ⓪_⋁_① →!
1. ⓪_⊃_② →!
2. ①_⊃_③ →!
3. T:_②_⋁_③ 0, 1, 2, ⩚

Unit 7 – More Joints

Maybe you saw back in Unit 2 that the ☮T meanings of the modes and joints used another joint, ⊃=_. There are many ways to say ⊃= in English such as, 'if and only if', 'only if', and 'just in case'. ⊃= is a *biconditional* and is also known as *conditional identity*. ⊃= is a joint that shows logical equivalence.

Meaning of ⊃=: ⓪_⊃=_① just in case (⓪_⊃_①)_⋀_(①_⊃_⓪).

(⓪_⊃=_①)_⊃=_~~(~~(⓪_⊃_①)_⋀_(①_⊃_⓪)~~)~~

⊃=T#

	⓪	①	⓪_⊃_①	①_⊃_⓪	⓪_⊃=_① (⓪_⊃_①)_⋀_(①_⊃_⓪)
0.	T	T	T	T	T
1.	T			T	
2.		T	T		
3.			T	T	T

⓪_⊃=_① is logically equivalent to (⓪_⊃_①)_⋀_(①_⊃_⓪) by its meaning. ⓪_⊃=_① is just a shorter way to write (⓪_⊃_①)_⋀_(①_⊃_⓪).

Maybe you also saw back in Unit 1 that ∨ had the word 'inclusive' next to it. Inclusive ∨ means 'at least one is true'. This leaves room for exclusive ∨, ∨∨, which means 'one and only one is true' or 'one is true but not both'. ∨∨ is a joint.

Meaning of ∨∨: ⓪_ ∨∨_① just in case (⓪_∨_①)_∧_⦸(⓪_∧_①)

(⓪_∨∨_①)_⊃=_{(⓪_∨_①)_∧_⦸(⓪_∧_①)}

∨∨T#

	⓪	①	⓪_∨_①	①_∧_⓪	⦸(①_∧_⓪)	⓪_∨∨_① (⓪_∨_①)_∧_⦸ (①_∧_⓪)
0.	T	T	T	T		
1.	T		T		T	T
2.		T	T		T	T
3.					T	

∨∨ shows what people sometimes really mean by 'or' or 'either'. Often, 'or' is used to show a choice where only one thing can be chosen. Logicians should keep in mind that natural language and the logic captured by ☮TT! are sometimes different. Logicians need to understand, appreciate, and respect those differences.

Someone might wonder if we could make a new joint for 'neither' to show the logic of 'Neither ⓪ nor ①'. We can, but we do not need a new joint. 'Neither' is easy to show in ☮TT!.

⦸(⓪_∨_①) Neither ⓪ nor ① is true.

We need to look at one last rule, Ξ (transformation). Ξ helps us with moving boxes around in complex formulas where the boxes either only affect ∧ joints or only affect ∨ joints.

Ξ (transformation):

Conjunctive

0. (⓪_∧_①)_∧_②	→!	*and*	0. ⓪_∧_(①_∧_②)	→!
1. T:_⓪_∧_(①_∧_②)	0, Ξ		1. T:_(⓪_∧_①)_∧_②	0, Ξ

Disjunctive

0. (⓪_∨_①)_∨_②	→!	*and*	0. ⓪_∨_(①_∨_②)	→!
1. T:_⓪_∨_(①_∨_②)	0, Ξ		1. T:_(⓪_ ∨_①)_∨_②	0, Ξ

Ξ is like ⊗ and z. We can make it more than one rule, but there is no good reason to make more than one since there is no basic difference between the four cases. Ξ is simply about when you can move boxes around. You can write the truth table.

Unit 8 – Necessary Truths

We saw a *tautology* (>T) in Unit 6. It is now time to see just how wonderful and amazing >T are. >T are formulas that are always true just because of their internal logic. They are *necessary truths*. It is not possible for >T to not be true because they are true for every value of their variables.

All of the rules of deduction can be made into >T. There is a method to this.

0. Start by joining all of the premises of a rule using ∧ using the rules for making formulas from Unit 1.
1. Next, if the formula has a joint in it, box it.
2. Then, if the conclusion has a joint in it, box it.
3. Finally, join the premises to the conclusion using ⊃, premises in front and conclusion in back. The formula is a >T.

You will see that for every ☮T rule, its >T formula is true on every line of its T#.

Example: Let us make a >T formula for M.

0. We start by joining the premises with ∧_. (⓪_⊃_①)_∧_⓪
1. Next, we box the formula. ⦃(⓪_⊃_①)_∧_⓪⦄
2. Then, we check the conclusion to see if it needs a box.
   ⓪ is a basic formula, so it does not need a box.
3. Finally, we connect the premise formula to the conclusion formula of M using ⊃, premises in front and conclusion in back. ⦃(⓪_⊃_①)_∧_⓪⦄_⊃_①

Now, let us take a look at M>TT#.

M>TT#

	⓪	①	⓪_⊃_①	(⓪_⊃_①)_∧_⓪	⦃(⓪_⊃_①)_∧_⓪⦄_⊃_①
0.	T	T	T	T	T
1.	T				T
2.		T	T		T
3.			T		T

Take a good look at the last column of M>TT#. M>T is *always* true, no matter what! This is why M is valid! This shows us what >T truly means.

<u>A formula is a >T just in case it is true on every line of its truth table.</u>

Unit 9 – More Modes and a Mystery

You can make other logical systems. Maybe you noticed that ☮T lists < and > with the joints. Up to now, you probably have not seen them used in logic. < and > are not really joints. They are modes. They just fall between the joints and the maths (Quire 8, page 17) because they work well with both groups. They are *modal-logic operators* (<>T!CD). *Modal logic* (<>T!) is about being creative with logic. You can give your own meanings to < and > and work to find logical connections among formulas using those meanings.

One really great way to use < and > is for < to mean 'possible' and for > to mean 'necessary'. You have already seen them used to mean these things. The ☮TCD for 'tautology' is >T_. >T means 'necessary truth'. The ☮TCD for 'contradiction' is ⦸<T_. ⦸<T means 'not possibly true' or 'not even a little bit true'. This is the main form of <>T!_. This form gets a special index number. ☮T <>T! of probability and necessity is ☮T <>T!0_. (The ☮T name is prefixed with '☮T' because other logicians have their own systems. It would be wrong to decide that the system below is the main one for all <>T!_. Naming it <>T!0 would not be good, but since this is the first ☮T T! quire, naming it ☮T <>T!0 is no problem.)

<>T!0 has four *axioms* (!↑↓T). Axioms are rules that are not proved but just said to be true so that we can explore systems built around them. You can make your own <>T! axioms.

REASON

Axioms are made from what you choose < and > to mean and from what those meanings seem to lead to. The ☮T <>T!0 axioms are:

A0. >#⃝_⊃=_⦸<⦸#⃝

#⃝ is necessarily true just in case it is not possible for #⃝ to not be true.

A1. <#⃝_⊃=_⦸>⦸#⃝

#⃝ is possibly true just in case it is not necessary for #⃝ to not be true.

A2. (>#⃝_⊃_⦸>⦸#⃝)

If #⃝ is necessarily true, then it is not necessary that #⃝ is not true.

A3. (⦸<⦸#⃝_⊃_<#⃝)

If #⃝ is not possibly not true, then #⃝ is possibly true.

(Just so that you know, this all could be made into one axiom by joining all four with ∧, but then we would need to use s all of the time to prove anything, which would be bothersome.)

Challenge for students: Prove the claim below using ☮T <>T!0 axioms.

Claim: >#⃝_⊃_<#⃝ If #⃝ is necessarily true, then #⃝ is possibly true.

[Hint: You will need to use the definition of ⊃= and make a substitution. If two formulas have the same truth table, you can always substitute one for the other because their meanings are the same.]

A Mystery: Look at the argument below.

0. All ☮T kids are smart. 1. ♡Ξ is a ☮T kid. 2. Therefore, ♡Ξ is smart.

The form of the argument above is clearly valid, but our rules cannot be used to deduce the conclusion from the premises. Have fun solving that mystery.

Fadeout

Using < and >, you can explore modal logic in fun and creative ways. Decide on meanings for < and > and see what you can come up with for axioms that make sense. Have fun!

(By the way, there is also predicate logic, Aristotelian syllogistic logic, Venn diagrams, and more! Good luck finding the texts. Scour those university library stacks.)