Spring 2024

Logic Lessons

Elizabeth Skiffington

Follow this and additional works at: https://cedar.wwu.edu/wwu_honors

Part of the Computer Sciences Commons, and the Logic and Foundations of Mathematics Commons

Recommended Citation
Skiffington, Elizabeth, "Logic Lessons" (2024). WWU Honors College Senior Projects. 816.
https://cedar.wwu.edu/wwu_honors/816

This Project is brought to you for free and open access by the WWU Graduate and Undergraduate Scholarship at Western CEDAR. It has been accepted for inclusion in WWU Honors College Senior Projects by an authorized administrator of Western CEDAR. For more information, please contact westerncedar@wwu.edu.
Logic Lessons
A course showing the intersection and shared basis of computer science and philosophy by Ellie Skiffington

Introduction .................................................................................................................................................. 2

Unit 1: Foundations .................................................................................................................................. 2
  1.1 True and False ...................................................................................................................................... 3
    1.1.1 And, Or, Not ................................................................................................................................. 4
    1.1.2 Laws of Logic ............................................................................................................................... 6
    1.1.3 Set Logic ....................................................................................................................................... 8
  1.2 Beyond Binary ..................................................................................................................................... 10
    1.2.1 What does x>y look like in the real world .................................................................................... 10
    1.2.2 How to critique ideas ................................................................................................................... 11
    1.2.3 Proof ............................................................................................................................................ 13

Unit 2 Practical Applications .................................................................................................................... 14
  2.1 Optimization ...................................................................................................................................... 14
    Philosophy ............................................................................................................................................... 14
    Computer Science ............................................................................................................................. 14
    Intersection ......................................................................................................................................... 15
  2.1 Artificiality ....................................................................................................................................... 15
    Philosophy ............................................................................................................................................... 15
    Computer Science ............................................................................................................................. 16
    Intersection ......................................................................................................................................... 16
  2.3 Responsibility .................................................................................................................................... 16
    Philosophy ............................................................................................................................................... 17
    Computer Science ............................................................................................................................. 17
    Intersection ......................................................................................................................................... 17
  2.4 Unsolvable ....................................................................................................................................... 17
    Philosophy ............................................................................................................................................... 18
    Computer Science ............................................................................................................................. 18
Introduction

I was sitting in a political philosophy class discussing the rationality of decision-making when the professor started explaining the statistical concept of expected value. Expected value is the average value of something when you consider the probability and value of each outcome. In the math class, the word problems associated with it were often gambling or other games, but here they were talking about concepts like leadership and opportunity. I had taken a statistics class a few quarters before that was a requirement for the computer science major where this topic was covered. I was foolishly surprised when the other students acted like this was an entirely new concept to them. That was not the first time I had seen parallels between my computer science and philosophy classes, but it was what spawned the idea for this project. Throughout my time in these disciplines that most people see as incredibly far apart, I kept finding common threads that connected them.

The goals of computer science and philosophy as disciplines are different. Computer science seeks to make things, philosophy seeks to understand the world. But at their core, they have a common seed, logic. I would recommend taking this course with other people, not solely on your own. Many of the questions asked are best talked over with another person. However, this could be completed on your own. Regardless, we are going to explore logic, and then the disciplines it has been used to build. To begin, start with Unit 1.

Unit 1: Foundations

To understand or use something, we need to know how to define it. What is logic? The Merriam Webster dictionary defines it as “a science that deals with the principles and criteria of validity of inference and demonstration: the science of the formal principles of reasoning.” With this definition the common core of these two disciplines seems less far-fetched. Reasoning and inference are both commonly used skills in the construction of programs and theories. But dictionary definitions are not always the ones we carry in our heads. I would have said how I try to understand the world. Logic is a set of tools that gives us reproducible results when applied according to logic’s rules.

Over this course we are going to be learning more about the tools logic can provide and how to use them in a variety of contexts. The contexts we will focus on are two fields that put logic on
a pedestal, philosophy, and computer science. Philosophy uses it to construct arguments and theories of how the world is or should be. Computer science uses it to build programs to accomplish a variety of tasks.

The following two questions are to get you thinking about logic in your life. Answer them to the best of your ability. Throughout this course I am going to pose more questions in this format. There are no correct answers to these questions, they are to spark your thinking about this material.

Where did you first encounter logic?

When did you first use logic?

1.1 True and False

To understand logic, we need a place to start. Logic is built on the assumption that there are true things in the world. These true facts are the building blocks of logical statements.

Logical statements take the form of If, Then statements. If then statements in normal text look like “If ___, Then ___. “ It can also take the form of “___, so ____” In natural language. In computer science, it is one of the most common and powerful tools used in programming. An example of an if statement in C is in this program.

```c
#include<stdio.h>
#include<string.h>

int main()
    { /*Getting input*/
    int lines;
    scanf("%d",&lines);
    get_t
    char text[lines][100];
    
    for(int i=0;i<lines;i++)
        { fgets(text[i],100,stdin);
         }

    /*Finding the longest line*/
    int maxl
    for (int i=1;i<lines;i++)
        if (strlen(text[i])>maxlen)
            { maxlen=strlen(text[i]);
            }
    }
```

It can be helpful to think of them as the = in a math equation to start but be careful, it is not entirely true. If/then statements are not equivalent, because if they were if a then b would also mean if b then a. That is not always true.
These statements are considered binary logic, as statements are either true or false. In the following sections, we will be using binary logic as our jumping off point.

Examples

If I am alive, then I must have been born

If the + symbol means we add the two values on either side, then 1 + 1 is equivalent to 2

1.1.1 And, Or, Not

Now that we have the concept of logical statements we can start demonstrating more complicated concepts. That’s where the big three operators come in; And, Or and Not. You should have a general idea from the English versions of the word, but because these concepts are essential to the course we will go over them in detail. Operators are a term from computer science but when it comes to this kind of logic, I find it helpful to think of these like +, -, *, and /. These are what you use to find what a statement is really expressing.

And

Symbols: &, &&, ∧, ·

In an and expression both sides of the operator must be true for the whole to be true. Computer Science generally likes && for comparisons, and & for bit operations (comparisons with very small amounts of data). Philosophy likes ∧ or · as the and.

Throughout this, I will be using & for simplicity’s sake. Here is an example of what & looks like when and is used with T for a true statement and F for a false statement.

T & T => T
T & F => F
F & T => F
F & F => F

Or

Symbols: |, ||, v
In an or expression at least one side of the operator must be true for the whole to be true. Computer Science generally likes || for comparisons, and | for bit operations. Philosophy likes V as the or.

\[
\begin{array}{c|c}
T | T & T \\
T | F & T \\
F | T & T \\
F | F & F \\
\end{array}
\]

Not

Symbols !, ¬

A Not operator switches the truth value of what’s inside it. Essentially just Flip it!

\[
\begin{array}{c|c}
!T & F \\
!F & T \\
\end{array}
\]

Practice

Evaluate the following statements given this information:

A = True
B = True
C = True
D = False
E = False

A&D
!D
B | E
!(E | (A&C))
Given the image above what is the output of the following?

\((\text{there is water in the fountain} \& \text{there are at least 8 benches}) \mid !\text{(there are lampposts)})\)

### 1.1.2 Laws of Logic

So far we have two building blocks, the logical statement (if _, then __), and the operators (and, or, not). Let’s start building.

There are several common laws that extend those operators, and we are going to look at two of them, double negation and De Morgans.

**Double negation**

Let's start with the simplest and the one you are most likely to have used in your life. Have you ever said that you have not not done something? Well, that’s essentially what double negation is.

If A => True

\(!A \Rightarrow False\)

\(!!A \Rightarrow True\)

**De Morgan’s Law**
De Morgan’s law can require more mental juggling, but you have probably also heard it in conversation, “You can’t be doing under water basket weaving and sky knitting”. It comes into play when a not operator is applied to the outside of an & or | statement.

Let’s remind ourselves of the And (\&) and Or (\|) operators.

For an expression with &, both sides must be true for the whole to be true, whereas for |, only one must be true. The law has two versions, Not And, and Not Or.

In formal logic they look like

\[ !(A \& B) \Rightarrow (! A) \| (! B) \]
\[ !(A \| B) \Rightarrow (! A) \& (! B) \]

Let’s start with Not And. It says saying that if both A and B are true, the result is false. If they are anything other than that, the result will be true. In the symbolic version that looks like \(!!(A \& B)\). From this we can logically deduce that \(!!(A \& B)\) is equivalent to \((! A) \| (! B)\). How you may ask?

Given that \(!!(A \& B)\) \Rightarrow (! A) \| (! B), we can test all cases of A and B.

If A and B are both true, \(!!(A \& B)\) = false, and \((! A) \| (! B)\) = false.

If A and B are both false, \(!!(A \& B)\) = true, and \((! A) \| (! B)\) = true.

If A is true and B is false, \(!!(A \& B)\) = true, and \((! A) \| (! B)\) = true.

If A is false and B is true, \(!!(A \& B)\) = true, and \((! A) \| (! B)\) = true.

Each time, they give an equivalent result.

Here is a more detailed example breaking down the individual steps, of one case. In this example A is true and B is false, and red = false and green = true.

\[ !(\text{red} \& \text{green}) \Rightarrow (! \text{red}) \| (! \text{green}) \]

A&B evaluate to false

\[ !\text{red} \Rightarrow (! \text{red}) \| (! \text{green}) \]
Not A = false and Not B = true

!(A) => I I

Not false = true

I => I I

True or false = true

I =>

An example of this in language might be the I can’t take 2 classes at the same time. That is the same thing as saying I have to not take one class or not take the other class. It feels awkward to say it like that, but they are equivalent statements.

Conversely, there is the other side of De Morgan’s law. !(A | B) => (! A) & (! B).

Given that !(A | B) => (! A) & (! B), we can test all cases of A and B.

If A and B are both true, !(A | B) = false, and (! A) & (! B) = false.

If A and B are both false, !(A | B) = true, and (! A) & (! B) = true.

If A is true and B is false, !(A | B) = false, and (! A) & (! B) = false.

If A is false and B is true, !(A | B) = false, and (! A) & (! B) = false.

Each time, they give an equivalent result.

To practice, assign an arbitrary true or false to A or B and see if, using the logical rules you have previously learned, you can boil each of the sides down to true or false.

1.1.3 Set Logic

Set Logic is going to be our bridge to larger concepts. So far we have stayed within the binary logic but the world is far broader than binary logic. Sets allow us to organize groups of things.

First a definition of sets: A set is a collection of things. Most often in mathematics, a set is a collection of numbers, though it doesn’t have to be. The set is made up of a number of elements. A good way to think about sets is they are some kind of container.

For example:
Set $S = \{1,2,3,4,5\}$ is a set of numbers, a family with a mom, son and grandmother is a set of people. 5 is an element of the set of numbers, the mom is an element of the family. We can describe this with symbols $5 \in S$, or the mom $\in$ the family.

Practice: Come up with a set of at least 5 numbers, and a set of something other than numbers with at least 5 elements. We will use these throughout the lesson.

Sets can also have super sets and subsets. Subsets are sets that have some elements from an existing set. An example of this would be considering a silverware drawer a set, with subsets being what is contained within the dividers. An example with numbers would be given the set $T = \{10, 30, 50, 70, 90\}$, a subset could be $\{10, 30\}$. Supersets are sets that contain all the elements of a given set. Following our current examples, the entire kitchen would be a superset of the silverware drawer and $\{10,20,30,40,50,60,70,80,90\}$ would be a superset of $\{10, 30, 50, 70, 90\}$. We can draw this with symbols, for example, $\{10, 30\} \subseteq \{10, 30, 50, 70, 90\} \subseteq \{10,20,30,40,50,60,70,80,90\}$.

While the examples given for these supersets and subsets had patterns to them, they don’t necessarily need to. As long as they fit the definitions, it counts.

Practice: Find a superset and subset for each of the examples you came up with in your last practice.

When we want to compare sets, we use operators. They are similar to, but not the same as the logical operators we have talked about previously. To help visualize these concepts I’ll be referring to this:

![Venn Diagram](image)

Intersection: $\cap$

The intersection is finding the overlap in two sets: In the diagram above, the intersection is the middle of the Ven diagram. This is similar to the “and” operator that we have looked at, where we are looking for two things to be the same.
Union: \( \cup \)

Union is about combining two sets. The union is the entirety of the Ven diagram. It is similar to the “or” operator, where we are looking for something to be in at least one to be a part of the union.

Difference: -

The difference would be one side of the diagram. This is similar to the not-operator, where

Before we are done with sets we have two more terms to introduce. These will help us fully make the jump out into the real world.

There exists: \( \exists \)

There exists, \( \exists \) is a term for when some attribute is true in at least one element of a set. For example, in the set of words contained in this website, there exists some analogies. This is a common way we attach attributes to sets. Let's return to one of the mathematical examples we looked at earlier: \( T = \{10, 30, 50, 70, 90\} \). We can say there exists some elements under the number 20.

For all: \( \forall \)

For all is a term for when some attribute is true for every element in the set. For example in the set of words contained in this website, for all words there is a definition. This is another common way we attach attributes to sets. Let's return to one of the mathematical examples we looked at earlier: \( T = \{10, 30, 50, 70, 90\} \). We can say for all elements, they are some odd number multiplied by 10.

Practice: For the sets you made at the beginning of this lesson can you define an attribute using the there exists and for all operators?

1.2 Beyond Binary

The world is so much more complicated than, and or and not. This section will be a stepping stone to full discussions about real topics. We will learn about how to apply these tools in more ways and how to adapt and expand them as we encounter things that don’t fit into nice and neat boxes.

1.2.1 What does \( x \rightarrow y \) look like in the real world

Facts are hard to come by. Start by trying to come up with 3 statements that are completely true. Now try to disprove those statements.
If you had a true statement you probably had to provide context. If you said something on your desk was 11 inches long, then you might need to clarify that you meant at the time of measurement, or from a certain time interval, because things are made and it wasn’t always 11 inches and there is a future where you break it, and it stops being 11 inches. There is a potential future where the definition of an inch changes and you need to further clarify. True statements often need things around them to give context.

But there is something regarding the construction of an argument that is independent of the truth, validity. Validity is not concerned with the content of an argument, it is concerned with its structure. When multiple statements use the same facts, the larger arguments construction can introduce conflicts.

Example: Assume all of the following statements equate to true.

A & B
! B | C
! C

This is an example of an invalid argument structure where, regardless of the facts, the whole together doesn’t make sense. For the statement A & B to be true, both A and B must be true. For ! C to be true, C must be false. But then ! B | C would be false.

Given an abstract structure like this, you can track the truth of the components to see if the general structure works. This is an important skill because if an argument is not valid, it is an immediate sign to disregard the argument in its entirety.

This skill of determining validity is often used in philosophical debate, and in programming to determine blocks of code that might be redundant, as well as determining ranges of inputs and outputs.

1.2.2 How to critique ideas

In philosophy and computer science, one of the things we are always trying to do is come up with the right answer. To do that, we need to look at things critically.

In a discussion, definitions are something that can be easily misconstrued. If you are presenting a logical argument, make clear what important terms mean in this context. This can easily be seen in a variety of plots in movies and TV where someone says something, and with other context or prior knowledge the meaning of the statement changes, (Shakespeare loved it!).

If you are critiquing a definition, it can come in several forms:
• Finding something to add to the definition
• Finding something that fits the general structure but is missing from the definition
• Finding something that is excluded from a definition

These are tangential to another important concept: oversimplification. Oversimplification includes when a definition is too broad.

Here is an example of oversimplification:

A car will not run without gas.
My car has no gas in it.
My car will not run.

Removing the oversimplifications looks like this:
My car will not run without gas.
My car currently has no gas in it.
My car will not currently run.

Adding the time and specifying which car in all lines we removed ambiguity from the argument.

Another common flaw in arguments is seeing correlation as causation. Correlation is when two affects have proportional changes over the same timeframe, whereas causation is the proof that those proportional changes are connected.

Here is a common example of correlation not equaling causation:

Ice cream sales rise in the summer.
Crime rates increase in the summer.
Therefore, ice cream sales cause increased crime rates.

Make sure you know the relationship of your facts if you are presenting them in an argument and investigate them in others.

The last way to critique ideas is identifying underlying assumptions. To operate in this world efficiently, we need to make assumptions. Assumptions are not bad, but they need to be recognized for what they are. When you are presenting a logical augment, explain what you based that on. This might be an ethical or economic framework or another theory. This is a common mistake in programming, as library functions can require a different state than you assumed. Take a second and try to remember the last time you found an assumption you made incorrect.
1.2.3 Proof

We’ve talked about ideas and what makes a clear argument, let's talk about proofs.

Firstly, what is a proof?

A proof is an explanation of a statement as to why it must be true. We use proofs in our everyday lives as justifications for our own ideas, actions, and decisions. These proofs however aren’t always good. They are often hastily constructed and don’t always hold up. When talking about computer science and philosophy proofs are more likely than not to be labored over for hours.

Definitions from Book of Proof:

Proofs often use the following terms:

A Theorem is a “mathematical statement that is true and can be (and has been) verified as true”.

A Definition “is an exact, unambiguous explanation of the meaning of a mathematical word or phrase”.

A Lemma is “a theorem whose main purpose is to help prove another theorem”.

A Corollary “is a result that is an immediate consequence of a theorem”.

Both computer science, math, and philosophy use these, though philosophy tends to have far more theories than theorems.

There are a lot of different kinds of proofs and they tend to look different in different disciplines. Some ones to be familiar with are direct, counter-example, contrapositive, and indicative.

Direct Proofs are the hardest to construct. This is a valid structure of lemmas that proves whatever point you are making. Counterexamples are generally the easiest type of proof to construct. It's far easier to prove something wrong than to prove something right, as you only need a single example of why it's incorrect. Hence the name, counter-example proof. Contrapositive proofs are say if something other than the expected outcome of what you are trying to prove happens, you can disprove the basis of that proof. Indicative proofs identify some pattern and then prove that pattern. They do this by proving a base case correct, then proving all larger are at their core the same as the base case.

If these are not all of the proofs in the world and if you have any more interest in proofs, I would highly recommend reading the Book of Proof.
Unit 2 Practical Applications

We have looked at how logic functions and practiced using it in simple examples. Let's take a bigger leap and look at the real world. We will be introducing a philosophical concept, a technology, and then looking at the intersection of the two. Throughout there will be questions asked that you should mull over. Consider them your practice problems.

2.1 Optimization

This is one of the first areas that I noticed concurrent ideals within philosophy and computer science. We will start by looking at an ethical framework, and then how programs are made more useful. Through this lesson, we will see how they run in parallel.

Philosophy

Utilitarianism going to be the only ethical framework for decision-making I am going to introduce in this course. There are a great deal of ethical frameworks out there, so I will start with one most are probably familiar with.

Utilitarianism is a way to detriment the “right” or “good” course of action by asking the question:

What would do the most good for the greatest amount of people? Many philosophers have proposed more specific definitions, as well as altering it slightly to be more useful or in response to critiques. There are a great deal of critiques of utilitarianism, including the following:

- It has no sense of deserving or justice when it comes to who the good is applied to.
- It relies on a variable definition of what is good.
- There are differing types of goods, and utilitarianism doesn’t care about type, only about quantity.

Question: Do you think utilitarianism is an ethical framework for making everyday life decisions?

Computer Science

Often in computer science, it is prudent to optimize a program for one of two things: efficiency or memory usage. In layman’s terms, we want the program to go as fast as possible or we want the program to use the smallest amount of the systems resources possible. Optimizing for efficiency typically means executing as few instructions as possible. Whereas, optimizing for memory on the other hand tries to use the least amount of dynamically allocated memory.
When optimizing for one specific aspect of a program, oftentimes other aspects may be worsened. An example of this is a type of programming problem called a dynamic programming problem. These problems calculate information based on previously calculated information which gives the programmer a choice. You can save the information, by using memory, or recalculate the information every time you need it, using more time. Either approach solves the problem, but the overall performance of the program is going to depend on what feature you optimized.

Intersection

This is an example of computer science and philosophy using the same underlying logic for vastly different purposes. It is the process of making decisions with a singular goal. In both cases, other important things can slide to the side when you only have one focus.

Question: What purpose does optimization serve? Where else is the process of optimization used?

2.1 Artificiality

Let’s begin with some big questions, what is real? What is natural?

Artificial has several definitions and connotations, something unnatural, something manmade, or something fake. Computer science and philosophy both have relationships with each of both definitions.

Philosophy

Because of the breadth of artificiality let's begin with a thought experiment, the experience machine.

The experience machine is a philosophical question posed by Robert Nozik in the book Anarchy State and Utopia. Nozik uses it to answer the question of what makes life meaningful, but we will be using it for a slightly different purpose.

Here is a summary of the question, If you could be plugged into a machine that gave you your perfect life would you enter? The machine operates for years at a time, so there is no option to try it out for a day. In the experience machine, you would have no knowledge that you are in the machine, and believe your experiences therein to be real, but know they are a result of the machine when you get out.

Would you enter, why or why not?

What does this tell you about what you consider to be real?
Computer Science

The conundrum we just looked at is what it means for an experience to be real. Computers don’t experience things, but we can look at how they remember them.

In modern-day computers, memory is stored in a couple of different places. The most basic is registers. Registers are the very base units of memory and are used when executing instructions. There is a small number of them, and many have designated purposes. When writing code, you don’t assign memory to registers unless you are writing in assembly (the coding language that is only a step up from pure binary). The next most used form is caches. Caches are places to store memory that is used more often to increase the performance of a computer. This is similar to making a summary sheet for an open-book test, allowing you to access information you know you might need with more efficiency than looking through a great deal of irrelevant information. Modern computers often have multiple sets of caches to make this even more efficient. Next up is the main memory, as the name suggests is the primary place things are stored. Main memory is used by two systems to access what is stored, logical/virtual memory and physical memory. When programs are run, each program assigns memory to its own set of addresses. Those placeholder addresses (the logical/virtual memory) are then assigned a place in the actual memory (physical memory). This allows programs to manage space more effectively. The last place memory is stored on a computer is the disk. This is the place where memory that is used very infrequently and can’t fit into main memory is stored. There is also cloud storage. This allows some set of data to be stored not on your computer, but on someone else's. With this, you can fetch, edit, and return that data, so that it doesn’t take up space on your personal machine.

We know that computer memory hardware is man-made, but is it real? The way we perceive memory on the screens we use (web pages, applications) is very different from the tiny electrical signals they actually are. Which is the real version?

Intersection

Memory and experience are difficult concepts to categorize even without trying to figure our what is real or not. Think about the following things:

How is memory in computers and people similar and different?

Has your opinion of artificiality when it comes to experiences and memory changed?

2.3 Responsibility

Responsibility is not an easy topic. It has many corollaries, for example blame and duty, but rarely is it put into the context of creation.
**Philosophy**

Tech companies often refer to themselves as disruptors, changing the way we live our lives. But there are costs to changing things. The most common philosophical question when it comes to personal responsibility vs public good is the trolley problem.

The trolley problem is a common philosophical question most people already know. A trolley is running down a track at an unstoppable speed, there are 5 people on its current path, and one on an alternate path you could divert the trolley to. Do you divert the trolley, killing one, or do you let it go on its current path, killing 5. Different ethical frameworks have different answers to this question, and so do people.

This is a simple thought experiment but its variations are too numerous to name. Some have loved ones on the tracks, questioning what are you willing to give up personally to do good for others. Others make judgements on the type of people on the tracks, criminals versus saints. Others still ask about the diversion itself, how hard it is, or do you have to sacrifice yourself.

For practice, design a trolley problem that makes you unsure if what you would decide to do.

**Computer Science**

Responsibility covers a large spread of topics with the emergence of new technology. The broadest by far is effects it has on others. In the last decade, technology has radically shifted our lives. Many of us use an application, a website or a piece of physical technology daily that did not exist 2 years ago. These technologies do have impacts on us, though because of the rate of change, it can be hard to study. The impacts can also extend beyond the personal. Things like power consumption, security, materials, military forces, and economies are all affected by the emergence of new technology, to name a few.

Practice: Research your favorite technology for the impacts it’s had, both positive and negative.

**Intersection**

Compare the impacts of new technologies with our talk of responsibility previously.

How should tech advancements be made?

Where does the responsibility lie for the impacts of new technology?

**2.4 Unsolvable**

I love these both of these fields but I cannot leave you thinking they are everything. There are limits to philosophy and computer science and what they can do. To see that we are going to look at some paradoxes and some unsolvable problems.
Philosophy

Paradoxes are another common philosophical concepts that are more generally known. We are going to start with a classic written by a Greek philosopher Eubuildes called the Liar.

The paradox is as follows: A person says the statement, “This statement is a lie”

If the statement is true, that would be a contradiction, as the statement says it isn’t. If it was false, there would also be a contradiction, which would no longer be a lie. Essentially, there is no way for the statement to be accurately true or false. Paradoxes like this that use their own definitions as their source for conflict. There are many more like this.

Computer Science

The Halting problem is an example of the limitations of computer science. The Halting problem consists of the following question, is it possible to create a function, that for all inputs, will tell you if a program will halt (hence the name) or run forever?

Before I explain this in more detail, give an initial impression of what you think the answer might be if you have never heard of this problem before.

Now the answer is it’s impossible, with a short proof by contradiction. Assume there is a function that can compute if a program will halt, called T(R) where R is some input. It returns true if R halts, and false if not.

Then lets make another function called P

fun P
if T(P)
Infinite loop()
If T(P) = true, it will loop forever, and only terminate if T(P) = false.

This has many implications for the development of certain technologies. What else can’t computer science do, and can you prove it?

Intersection

In this section, we have discussed examples of computer science and philosophy reaching a certain endpoint. In this course, we have looked at how logic builds and supports these two amazing fields of study.

What are the limits of philosophy?
What are the limits of computer science?
Does logic play any role in those limits, why or why not?

**Unit 3 Conclusion**

After all of this learning, I would ask that you all take a step back and reflect. The goal of this course is to help make the connection between making and understanding. But as we know there are limits to both. There are limits to how much logic can do for us. Logic is a way of modeling the world and models are extremely helpful. But they are imperfect. At the start of this course, I asked you to write about your experiences of logic. I will now ask you to look for a time when logic was not solely enough.

Thank you for taking the time to do this course.