Turing Script:

To review, a Turing machine is a theoretical construct. It has an infinite tape divided into cells with symbols (usually 1, 0 or blank), and a head. The head can move along the tape, read and write to a cell and contains finite instructions to determine what to change, where to move and when to stop. Since software didn’t exist at the time, the ‘instructions’ would be thought of as part of the machine. This means a new Turing machine needed to be made for each function.

As we know, these instructions/states are equivalent to an algorithm running on the input data.

The possible states of a Turing machine can be represented as a state diagram, with each state being a possible configuration. This example uses 0, 1, A, X, and # as its alphabet.

The head scans the tape one cell at a time and depending on the input changes it. L and R indicate for the machine to move left and right respectively.

When it reaches the halt state the program will stop and produce the output. Sometimes they have multiple exit states such as a True or False for decision problems, but this depends on the design.

The 2 letters (separated by “:”) indicates if the machine is in state R and reads the input “A”, it should replace it with “X” and move to the next state, L.

For this one requires only A’s as an input.

If we follow this logic: (PowerPoint) …

If we skip forward to when all the A’s are overwritten, the machine overwrites all the X’s with #’s till it halts.

In this example, it has turned 6 A’s into 110. Does anyone know what it does?

It turns unary (a system with one symbol) into binary.

Some other stuff to remember:

* Turing machines work on natural numbers
* The functions expressed on a Turing Machine are computable functions, since there is an algorithm that can do the job of the function.
* The output numbers are considered computable numbers since there is a function that can calculate it and it satisfies this inequality.
* Each individual Turing machine has a Number that identifies it, called a description number. In other words, the instructions/states that a Turing machine acts out are denoted a number. This can be equivalent to something like this.

The naming process for each Turing machine depends on the input and output but it is not usually done. The point is that a natural number could be interpreted as a description number for a Turing Machine even and that a number always exists for a corresponding Turing machine.

Remember how a new Turing Machine needed to be created for each function, The Universal Turing Machine solves this problem.

The UTM is a Turing machine that can simulate any other Turing machine. It takes the description number of the machine it is simulating and the input tape. Since the numbers are hard to usually identify a state diagram is put in place of it. The UTM can be thought of as a computer and each Turing machine it simulates as a program.

This brings us to Turing Completeness. For something to be Turing Complete it must be able to do anything a Turing machine can or can simulate a Universal Turing machine. This implies that it can manipulate data using conditional branching (run any algorithm) and have the memory required to solve the problem (a long enough tape). Basically, something that can compute anything that can be computed, no matter how long it takes, is Turing complete. Turing completeness can apply to hardware, like a computer, or software, a Virtual Machine.

For example, A calculator is an example of a Turing incomplete machine because it can only perform a small pre-defined subset of calculations.

However, a Mac or a PC is a Turing complete because it can do any calculation that a Turing machine can do if we give it enough memory and time.

On the software side, SQL is not Turing Complete while Python is.

The applications of a UTM are like that of Turing Machines. They can:

Check Decidability

If UTM cannot compute a problem, then there is no algorithm that can solve that problem. Meaning it is undecidable.

For a decision problem if the UTM can halt for all finite length inputs then we can say that the problem could be solved by an algorithm.

Classify Problem

UTM helps to classify decidable problems into classes of Polynomial Hierarchy.

To find the time complexity of a problem we can just look at the relationship of the input to the output.

Design and Implement Algorithm for Practical Machines

As we know, Turing Machines are the most basic form of a computer. If we know it can be solved by a Turing machine and what class the problem is, then we know it can be solved on a computer and the appropriate algorithm to use.

If we are being picky, nothing we have created today is Turing Complete. To be Turing Complete it requires the ability to access an arbitrary amount of memory. Every computer today has a limited amount of RAM and can’t access any more than what is available. Think of Turing Completeness as the ability to compute all computable problems in theory.

The limits of computation apply as well. The halting problem for example can’t be computed reliably for every input. This applies to every computer or computational device.

This also relates to how powerful tools we can create can be. If any Turing Complete programming language can in theory run any function a UTM can then all Turing complete languages are as powerful as each other. In the sense that they can all in theory solve the same problems and not solve the same problems. This links back to the computation point. Unless something paradigm shifting is discovered, any improvements in technology will speed up the time it takes to solve computable problems, and perhaps make some problems more feasible, but doesn’t solve problems previously incomputable.

All of this assumes that the Church-Turing thesis is indeed correct. For instance, the controversial field of Hypercomputation. To tackle the concept of incomputable numbers it requires the Church-Turing thesis to be wrong is either 3 ways:

1. There is a process that humans can do to calculate these numbers that computers cannot.
2. There are functions that machines can compute which humans cannot (given infinite time and memory)
3. There are instances in which incomputable numbers can be measured in nature rather than computed.

Many claim this field baseless, but it is still an interesting idea that the Thesis could be incorrect. If the thesis is false, it would say that the Universe doesn’t work like a Turing machine and would change our current understanding.