Alan Turing's legacy is even bigger than we realise

December 1 2014, by David Craven

Deep thoughts for deep problems. Credit: xcv, CC BY-NC

Alan Turing is one of the world's best-known mathematicians, and probably the best known in the past century. This is partly for his work on cracking German codes in World War II, and partly for his arrest, conviction and punishment for homosexuality in the 1950s. The mathematics that made him famous, however, rarely gets mentioned.

His early work, which is more theoretical in nature, set the foundations for the concept of a modern computer. He then went on to actually start building these computers, mostly during World War II, and this is the part of Turing's life that is most often talked about.

After the war, he had a brief spell working on real-life computers before thinking about artificial intelligence. This is when the famous Turing Test appeared, and it was just before the other part of his life that is normally discussed – his conviction for homosexuality, chemical castration and suicide.

If you have watched the recently released The Imitation Game, you will have heard about his exploits in World War II, and of his later downfall and death. What is rarely mentioned is this early work on the theory of computers, but this is as important for the development of the field as his practical work at Bletchley Park.

Turing's first contribution to mathematics was to define what he called an a-machine, but which later became known as a "Turing machine". To a generation used to computers, this makes a lot of intuitive sense, but in the 1930s it was a bold and innovative leap.

Roughly speaking, a Turing machine needs an input of some description, a set of states it can be in, like an internal thought process reminding it what it should be doing, and a program that tells the machine what to do with the input it has and the frame of mind it is in.

An example is a calculator: the input is you pressing the buttons, and the internal state is it being in an adding or subtracting mood, for example. The program tells the calculator, given the input 10 and 10 and the mood of addition, how to switch between the various states in its program in order to calculate the response.

Often, for even simple tasks, the set of states is very large in this theoretical concept. Its use is not so much as a practical design but as a method of discerning what can and can not be done by a computer. Turing took this one stage further by defining what became known as a Universal Turing Machine.

This machine is programmable, in the sense that the input into this machine includes a complete program as well, so that it can simulate any other Turing machine.

As an example of this, consider a standard computer. Opening the calculator application on your desktop produces a simulation of the earlier Turing machine I described. The computer here is a Universal Turing Machine, and the calculator program is on the hard drive, being read into the machine as an input. Thus, a Turing machine can be capable of only one task (like a light switch), a range of tasks (the calculator), or can be Universal, and then is capable of any task that can be done by any Turing machine one can consider.

But why did Turing invent these concepts? He wanted to attack David Hilbert's so-called "decision problem": is there some algorithm which can take any mathematical statement and tell you whether it is true or false?

Turing machines allowed one to formally define the concept of "algorithms", and allowed him to prove rigorously that no such algorithm can ever exist. He did this via another problem, the "halting problem": suppose you write some computer code, and you ask your computer to run it, will that computer program finish running after a minute, a year, a millennium, or will it continue running forever, and never end? The halting problem asks for an algorithm that will take any computer program and tell you whether, if you run it, it will eventually stop, or whether it will run forever. Like the decision problem, this has a

negative answer.

Hilbert's decision problem was also solved by Alonzo Church (Turing's PhD supervisor) at the same time, but whereas Church's methods are difficult, Turing machines are intuitive, and it is his formulation that is taught to mathematicians and computer scientists across the world to this day. Turing's radical ideas were a huge step in understanding, and led to people being able to ask deep questions about computers and the limits of computation. The formalism of Turing machines forms the fabric of computer science, and informs the ideas and concepts that have come since. This legacy will live on for centuries hence.

This story is published courtesy of The Conversation *(under Creative Commons-Attribution/No derivatives).*

Provided by The Conversation

Citation: Alan Turing's legacy is even bigger than we realise (2014, December 1) retrieved 20 September 2024 from https://phys.org/news/2014-12-alan-turing-legacy-bigger-realise.html