Church-Turing Thesis

What is computable? In the 1930s, two researchers from two countries – Alan Turing from England and Alonzo Church from the US – started analyzing the question of what can be computed. To answer this question, they used two different approaches:

- Turing, with his analysis of Turing machines, concentrated on elementary computational steps – what we would now consider computer engineering and computer architecture;
- on the other hand, Church focused on what can be described – i.e., on what we would now consider programming languages.

At first, they came up with two different answers to this question:

- Turing argued that a function is computable if and only if it can be computed on a Turing machine, while
- Church argued that a function is computable if and only it be described by a program in his programming language (which is similar to LISP and Scheme).

At first, it looked like these are two different answers – until Church proved that these definitions are actually equivalent:

- if a function can be computed on a Turing machine, then it can also be described by a program in Church's programming language, and
- if a function can be described by a program in Church's programming language, then it can also be computed on a Turing machine.

So, their two statements merged into one, which we now call *Church-Turing thesis*.

Comment. Sometimes, this thesis is simply called *Church thesis* – but this is an ambiguous name, since it sounds as a thesis provided by the church.

Formulation of the Church-Turing thesis. *Anything that can be computed on any computational device can also be computed on a Turing machine.*

Comment. Now that we have more modern programming languages, it was proven that computability in these language is also equivalent to computability

by a Turing machine. So, we can present the following equivalent formulation of the Church-Turing thesis:

Equivalent formulation of the Church-Turing thesis. *Anything that can be computed on any computational device can also be computed by a Java program.*

Comment. Instead of Java, we can use C, Python, Matlab, any universal programming language – some of these languages may be more efficient and/or more convenient, but they all compute the exact class of functions.

But is this thesis true? It depends on physics. For computational devices built on Newtonian physics, this has been proven by Turing's colleague Robin Gandy. For devices using quantum effects, it is still an open problem – although most computer scientists think that it is true.

This does not mean that this statement is true for all possible physical models. Let us give a simple example. We have shown that no Java program is possible that, given a program p and data d, checks whether p halts on d. But suppose that time machines are possible. Then, we can then check whether p halts on d very easily:

- let the program p run on data d, and
- set it up so that if p does halt at some future moment of time, the computer will send us a message from that future time back into our present time.

In this case:

- if we get such a message at the present moment of time, this means that the program halted;
- if we do not get such a message at the present moment of time, this means that the program did not halt.

So, by using time machines, we can solve programs that cannot be solved by Java programs!

Conclusions.

- *The Church-Turing thesis is a statement about the physical world.*
- Whether this statement is true or not depends on physics. So, *the Church-Turing thesis is not a mathematical theorem* – since mathematical theorems do not depend on physics.