Lecture #7: Simulations and Turing Machine Variants
The Church-Turing Thesis

The Early, Early Days

While the ideas of "computability" and "algorithms" certainly existed before the twentieth century, they had not been made precise enough for people to be confident that they were talking or writing about the same things, when they were using these terms. Work in the 1930's included (at least) three significant efforts to formalize these:

- In 1933, Kurt Gödel and Jacques Herbrand formalized the definition of ***general recursive functions*** — the smallest set of functions with (arbitrarily many) non-negative integers as inputs, computing natural numbers that include a set of "base functions" and that are closed under a set of operations on functions (so that the definition of this set of functions has the same general structure as the definitions of "regular expressions over an alphabet", as given in this course, even though the details are quite different).

  "General Recursive Functions" are still discussed today. Indeed, they are often featured more prominently in (more specialized) introductions to computability than either of the other formalizations being mentioned here.

- In 1936, Alonzo Church introduced another method of defining (some) functions — again, with (arbitrarily many) non-negative integers as inputs, and returning natural numbers as outputs, called the ***λ-calculus***.

  The λ-calculus is also still used today. Indeed, the development and use of ***functional programming languages*** and ***functional programming*** makes considerable use of this.

- In 1937 — but independently of Church's work — Alan Turing introduced a version of the kind of ***Turing machine*** that is being studied in this course.

It was not immediately clear how these formalisms were related. Indeed, they do not even work with the same types of objects — so that it is necessary to describe a way to "encode"

natural numbers as strings over an alphabet, before asking a function that is general recursive, or that can be specified using the λ-calculus. It is also necessary to "encode" strings over an alphabet as natural numbers in order to ask whether a function, that can be computed using a Turing machine, is also general recursive or can be specified using the λ-calculus.

There was a period of time when it was *not* known the formalisms were equivalent or even related, in any useful sense — and each was considered to be the "right" formalism by various people. In time, though, it was proved that — up to "encodings" — all three of the above formalisms are equivalent. So are several others that were proposed at approximately that time and — for one reason or another — are still useful, or interesting, today.

What is the "Church-Turing Thesis"?

The ***Church-Turing Thesis*** is a widely accepted belief that the researchers who introduced these formalisms "got it right". One way to state this is that "any function of the natural numbers can be calculate by an effective method ***if and only if*** if is computable by a Turing machine".

Evidence in support of this *belief* includes the body of the results, establishing equivalence of proposed models of computation, like the ones mentioned above.

More Models

As time passed, and computers were developed, additional models — including models that more closely resembled components of computers in use today — were proposed. One model (or, perhaps, "family of models") was the ***random access machine***.

- Instead of a set Q of *states*, a random access machine generally includes a ***program*** — consisting of a sequence of instructions written using an extremely simple programming language.
- Instead of one or more storage tapes, a random access machine has an infinite sequence of ***registers*** $r_0, r_1, r_2, \ldots$ that can each store a non-negative integer. A computation begins with the input(s) stored in as many initial registers as are needed, with the values of all other registers set to zero.
- The programming language's instructions allow the values of specified registers to be incremented or decremented. Values can be copied from one register. There are also simple instructions that provide simple tests and branching. Furthermore, the value of a register can be used as the *address* of another register whose information can be accessed — providing a form of ***indirect addressing***.

Several variants of a "random access machine", with somewhat different statements in the programming language, have been proposed in computing research — additional details can be found in the Wikipedia page for this topic [2]. For a sketch of a proof that a random access machine can be simulated by a Turing machine, See Section 7.6 of the text of Hopcroft and Ullman [3].

For a variety of other abstract models of computation that have been introduced (up until the end of the twentieth century) — and sketches of additional simulations that support the "Church-Turing thesis", see the text of Savage [6]. Still more information like this — and quite a bit more of the *history* (and development) of this thesis — can be found in the text of Robič [5].

Additional Details

For additional details — including a brief overview of more recent developments, see the Wikipedia page for the "Church-Turing thesis" [1]. For even more information — including evidence that there has been some dispute over what the "Church-Turing thesis" should be taken to *mean* about the computability — see the collection of Olszewski, Wolenski and Janusz [4].

References

[1] Wikipedia Foundation. Church-Turing Thesis. `https://en.wikipedia.org/wiki/Church-Turing_thesis`. Accessed on October 9, 2025.

[2] Wikipedia Foundation. Random Access Machine. `https://en.wikipedia.org/wiki/Random-access_machine`. Accessed on October 9, 2025.

[3] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, first edition, 1979.

[4] Adam Olszewski, Jan Wolenski, and Robert Janusz, editors. *Church's Thesis After 70 Years*. Ontos Verlag, 2007.

[5] Borut Robič. *The Foundations of Computability Theory*. Springer-Verlag, second edition, 2020.

[6] John E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley, 1998.