Modelling Computations of Languages
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.

Interest in formalizing these ideas came in a somewhat surprising way. In the early 1920's, the German mathematician David Hilbert advanced a proposal to deal with the "foundational crisis in mathematics" by formalizing all of mathematics in axiomatic form — and providing a proof that this axiomatization of mathematics is consistent. This proposal came to be known as ***Hilbert's program***.

Interested in formalizing the notions of "algorithms" and "computation" came from results that, essentially, ended work on Hilbert's program — Gödel's ***Incompleteness Theorems***, which establish significant *limits* on provability on the types of formal axiomatic theories that Hilbert wished to develop. Indeed, interest in formalizing computation seems to have been motivated by an attempt to better understand, and extend, Gödel's results.

Other work in the 1930's included several significant efforts to formalize these:

- Jacques Herbrand was a French mathematician who made several significant contributions in different areas of mathematics — even though he died, in an accident while mountain-climbing, at the age of 23.

  In a private communication to Kurt Gödel, Herbrand presented an idea that Gödel completed in order to obtain the set of functions that we now call the ***primitive recursive functions***.[1]

  While Gödel had developed this by 1934, he was unconvinced that this was sufficiently general, or powerful, to accurately model computation.

[1] As early lectures suggest, significantly earlier work of Dedekind, and Skolem, included similar ideas.

- Alonzo Church had been been working on a system that we know of as the "λ-calculus" since, at least 1931. Church argued to Gödel, in 1934, that the λ-definability of functions should be "adopted" as the notion of their "computability".

  Gödel was not convinced — seemingly in part because the λ-calculus is quite abstract and is not clearly connected to how computations really get carried out. Nevertheless, Church published a version of this assertion — a "Church's Thesis" — in 1936.

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

- In 1936 Stephen Cole Kleene — a student of Church's — proposed an extension of the set of primitive recursive functions by including the "unbounded minimization" operator — giving us the ***μ-recursive functions*** introduced in Lecture #3.

- In 1936–37 — possibly slightly later, but independently of Church and his students — Alan Turing introduced a version of the kind of ***Turing machine*** that is being studied in this course. Turing advanced two claims — one about "computable" numbers, and another about the "operations" that a model of computation must support — that can each be called ***Turing's thesis***.

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 (as in an earlier lecture) to describe a way to "encode" natural numbers as strings over an alphabet, before μ-recursive functions, and Turing-computable functions can be compared.

A significant body of work, starting later in the 1930's — with significant contributions by Church, Turing, and (perhaps, especially) Kleene — established connections between these attempts to define computability — including versions of "equivalence" proofs. This lead to the realization that most of the early definitions of computability support each other, instead of contradicting each other.

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 on 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 11, 2023.

[2] Wikipedia Foundation. Random Access Machine. https://en.wikipedia.org/wiki/Random-access_machine. Accessed on October 11, 2023.

[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.