The Church-Turing Thesis and Turing-completeness

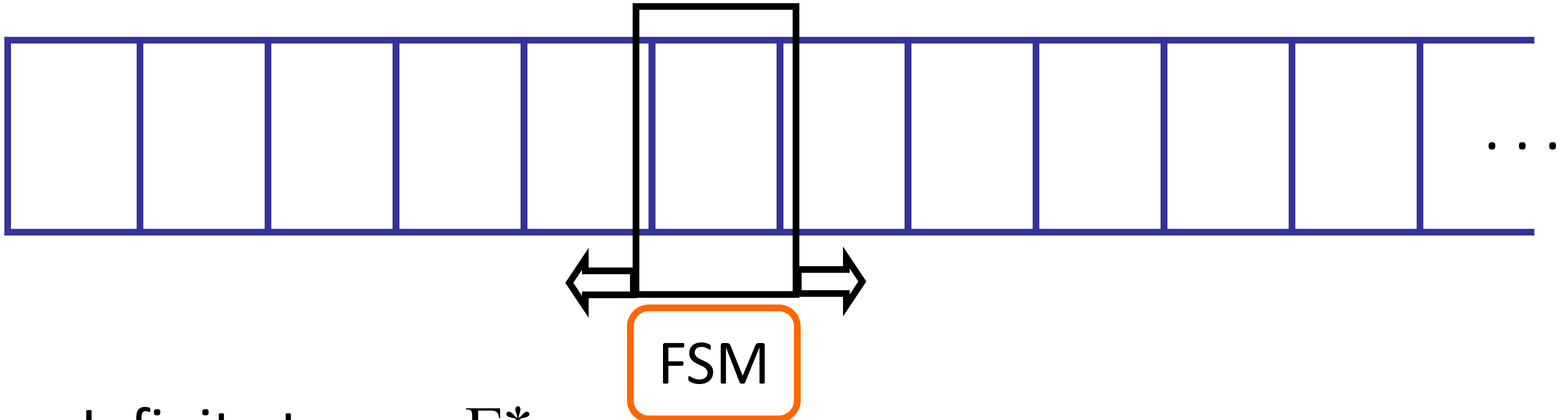
Michael T. Goodrich
Univ. of California, Irvine

Alonzo Church (1903-1995)

Alan Turing (1912-1954)

Some slides adapted from David Evans, Univ. of Virginia, and Antony Galton, Univ. of Exeter

Recall: Turing Machine Definition



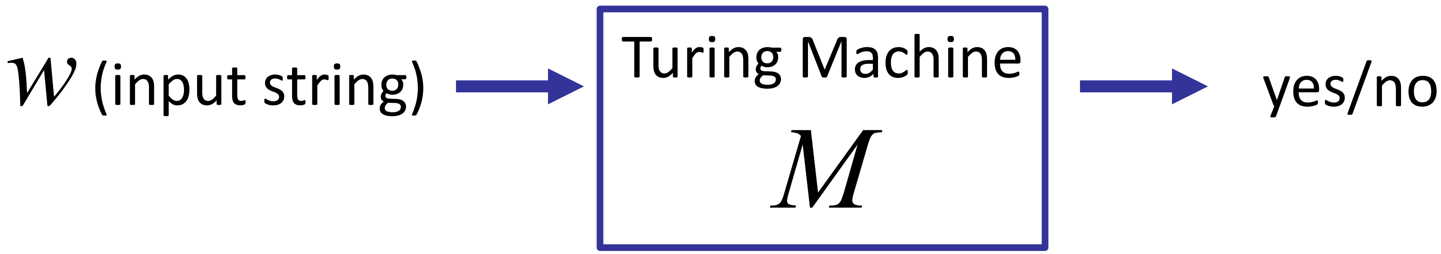
Infinite tape: Γ^*

Tape head: read current square on tape,
write into current square,
move one square **left** or **right**

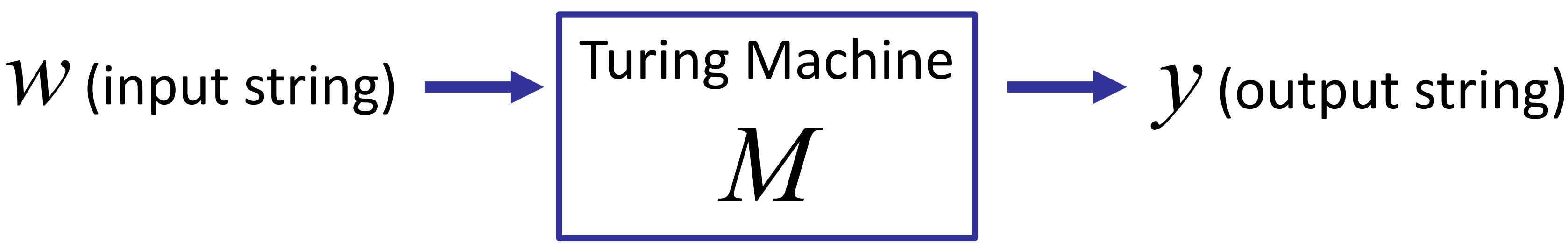
FSM: Finite-state machine:
transitions also include direction (**left**/**right**)
final accepting and rejecting states

Types of Turing Machines

- **Decider/acceptor**:



- **Transducer** (more general, computes a function):



Church

- Alonzo Church, 1936, *An unsolvable problem of elementary number theory*.
- Introduced **recursive functions** and **λ-definable functions** and proved these classes equivalent.

"We … define the notion … of an *effectively calculable* function of positive integers by identifying it with the notion of a recursive function of positive integers."

Turing

- Alan Turing, 1936, *On computable numbers, with an application to the Entscheidungs-problem*.
- Introduced the idea of a **Turing machine computable number**

"The [Turing machine] computable numbers include all numbers which could naturally be regarded as computable."

The Church-Turing Thesis

"Every effectively calculable function can be computed by a Turing-machine transducer."

"Since a precise mathematical definition of the term effectively calculable (effectively decidable) has been wanting, we can take this **thesis** ... as a definition of it..." – Kleene, 1943.

That is, for every definition of "effectively computable" functions that people have come up with so far, a Turing machine can compute all such such functions.

Equivalent Statements of the Church-Turing Thesis

- "Intuitive notion of algorithms equals Turing machine algorithms." Sipser, p. 182.
- Any mechanical computation can be performed by a Turing Machine
- There is a TM-*n* corresponding to every computable problem
- We can model any mechanical computer with a TM
- The set of languages that can be decided by a TM is identical to the set of languages that can be decided by any mechanical computing machine
- If there is no TM that decides problem P, there is no algorithm that solves problem P.

All of these statements are equivalent to the Church-Turing thesis

Examples of the Church-Turing Thesis

- With respect to computational power (i.e., what can be computed):
  - Making the tape infinite in both directions adds no power
  - [Soon] Adding more tapes adds no power
  - [Church] Lambda Calculus is equivalent to TM
  - [Chomsky] Unrestricted replacement grammars are equivalent to TM
  - Random-Access Machine (RAM) model is equivalent to a TM

"Some of these models are very much like Turing machines, but others are quite different."

Turing-Complete Systems

- A computer system, C, is **Turing-complete** if it can simulate a universal Turing machine.
- Thus, by the Chuch-Turing Thesis, the computer system, C, can compute any computable function.
- So… if you want to show that a computer system can compute anything, you just need to show that it can simulate a Turing machine.

Turing-complete Systems by Design

- Programming languages C, C++, Java, Python, Go, Visual Basic, Ruby, Pascal, Fortran, COBOL, …
  - These are Turing-complete by design.

Accidental Turing-complete Systems

- Excel
- C++ templates
- Java generics
- Border Gateway Protocol (BGP)
- Magic: The Gathering
- Minecraft
- Minesweeper
- Conway’s Game of Life