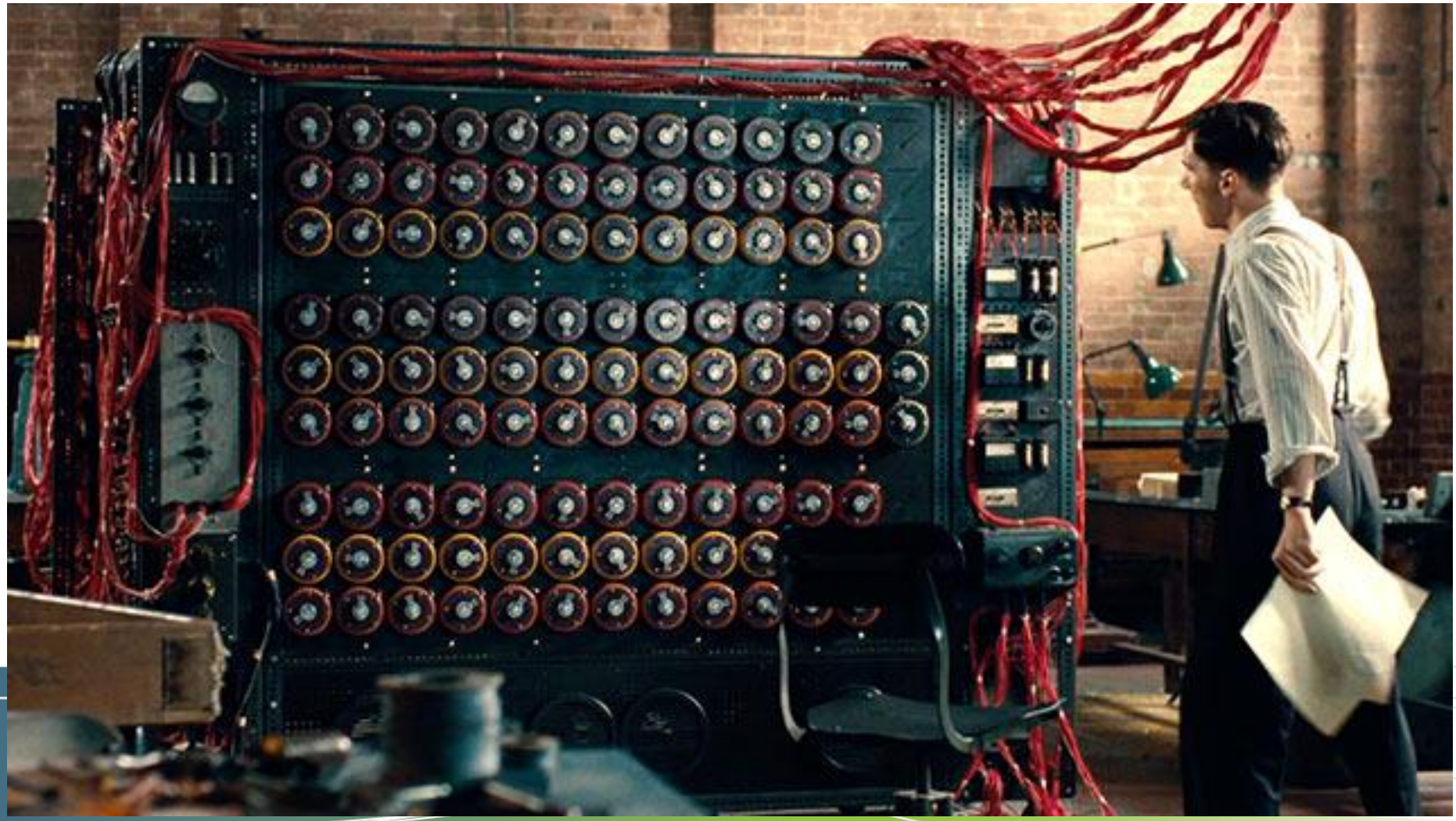
Readings…

- Sections 3.1 – 3.3 from Sipser
- Wikipedia article:

  https://en.wikipedia.org/wiki/Turing_machine
- Program your own Turning Machine!

  https://turingmachinesimulator.com/

What is a Turing Machine?



A Bit of History

- 1928 – David Hilbert's Entscheidungsproblem (*Decision Problem*):

"Is there an algorithm that can prove whether a statement in first-order logic is true or false?"

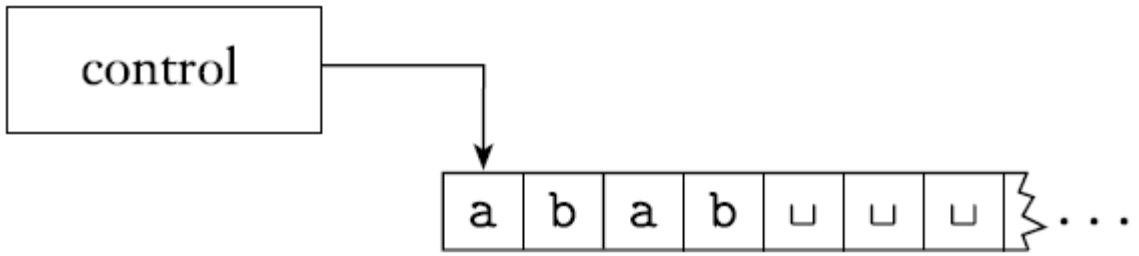
- Technical Problem: What exactly do we mean by "algorithm" here?

Church's Characterization

- In the 1930s, Alonzo Church characterized "effective calculability" as being anything that could be computed using his λ-calculus
- The functional language LISP was influenced by the λ-calculus
- Very "mathy", and doesn't immediately resemble our sequential notions of algorithm

Turing's Characterization

- Alan Turing saw algorithms as a "mechanical process", and so used a machine-based characterization
  - These became known as Turing Machines
- Turing Machines can be in one of a finite number of states, and are deterministic, very much like a DFA
- However, TMs also have an infinite tape for memory on which it can read and write characters



Church-Turing Thesis

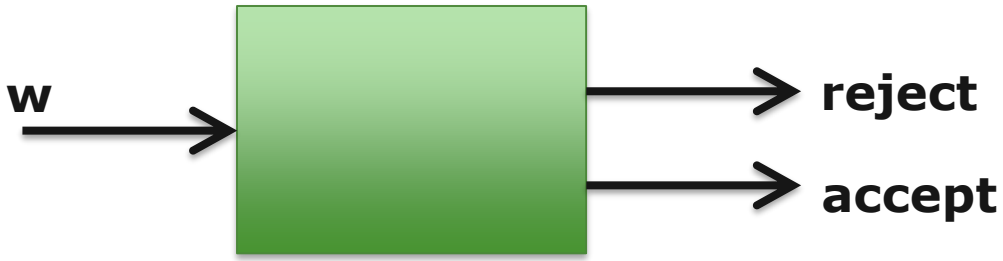
- Church-Turing Thesis: Both of these notions are equivalent in terms of the ability to compute things, and can be used to describe any algorithm!

Ramifications of Church-Turing

- Any computer program you can write in your favorite language (e.g., C/C++, Java, Lisp) can be converted into an equivalent Turing Machine!
- Thus, if someone asks you "Is this function computable?" or "Can a Turing Machine do this?" you can simply ask yourself if you can write a program in any other language to do it
- For this reason, we will not spend time on how to write a formal Turing Machine definition this semester

Turing Machines as Language Recognizers

- Turing Machines are meant to act as language recognizers like DFAs – they attempt to either accept or reject a given input string, though they might loop forever!



Keep in mind that TMs have a single "hard coded" algorithm that they follow - they are more like DFAs or programs than general purpose computers

Recognizers vs. Deciders

- We say that a Turing Machine M recognizes a language L, if M accepts exactly those strings in L
- We say that M decides a language L, if M accepts all strings in L, and rejects all other strings
  - In other words, deciders always give an accept or reject answer, and will never loop forever

Recognizable vs. Decidable

- We say that a language L is recognizable if there exists some Turning Machine that recognizes it
- We say that L is decidable if there exists some Turning Machine that decides it
- Most languages that you can generally think of are decidable
  - All regular languages are decidable, as are many non-regular languages

Transducer Turing Machines

- A Transducer Turing Machine is a Turing Machine that is used to compute functions, instead of recognizing or deciding languages
- TTMs operate in the same way as standard TMs, except that they produce an output string for a given input string, instead of accepting or rejecting it

Transducer Turing Machines

- TTMs must always terminate with an answer, they are not allowed to loop forever
- We say that a function f is computable if there is a TTM that outputs the correct value for f(w) for every possible input w
- All arithmetic operations on integers are computable, as are most functions that we commonly use (given that the input/output can be represented using finite strings)