The Turing Computational model and decision problem

The theory of computability

Joseph Chang

Goal

- Introduce finite state machine and Turing machine briefly
- Introduce the idea of Turing complete computability
- Prove the decision problem
- Note: This topic is taught as a senior level computer science course in college (Theoretical computer science).

Brief history of Alan Turing

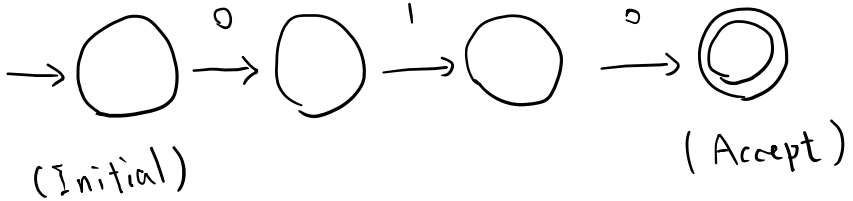
- The father of theoretical computer science
- Invented the **universal** computer model: Turing machine
- Proof of decision problem (Entscheidungsproblem) i.e., a universal algorithm answers yes or no to a question. It answers whether an assumption leads to a conclusion.
- Turing reducibility i.e., If machine A reduces to machine B, then machine B can be a solution of machine A.

Jacquard loom machine

- Input: punch cards
- Output: textile fabric

Finite state machine (deterministic)

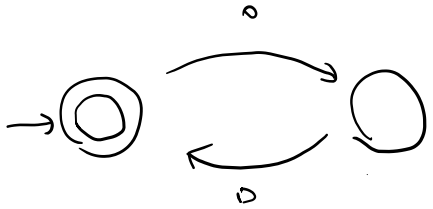
A machine that reads a binary sequence (string) and decides if the machine recognizes (accepts) the sequence.



$L = 010$

$S \rightarrow 010$

Accepts string 010 only



Finite state machine

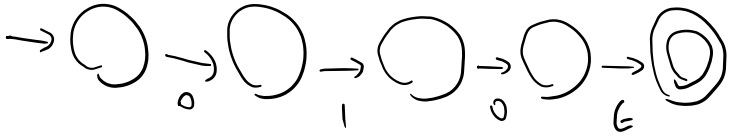
$L = (00)^*$ Regular language

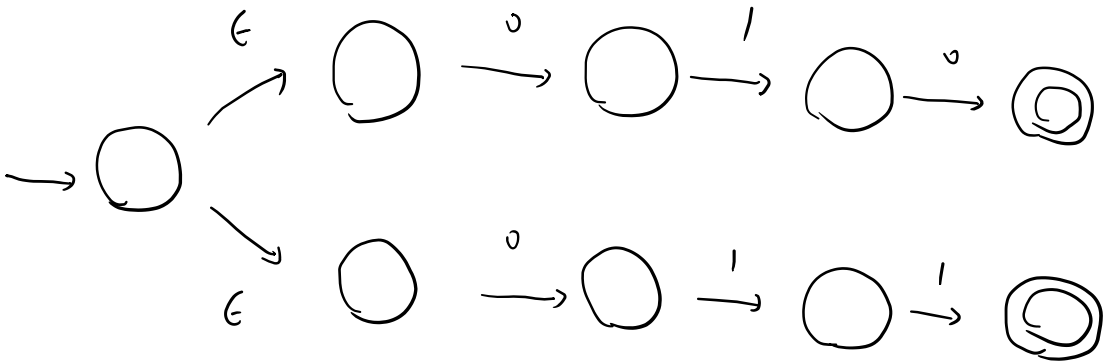
$S \rightarrow SS \mid 00 \mid \epsilon$ Regular grammar

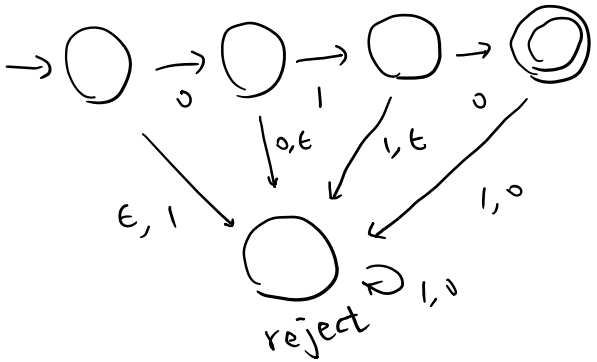
Accepts even numbers of 0s

Variants of finite state machine

Epsilon character, rejection state, non-deterministic finite state machine.



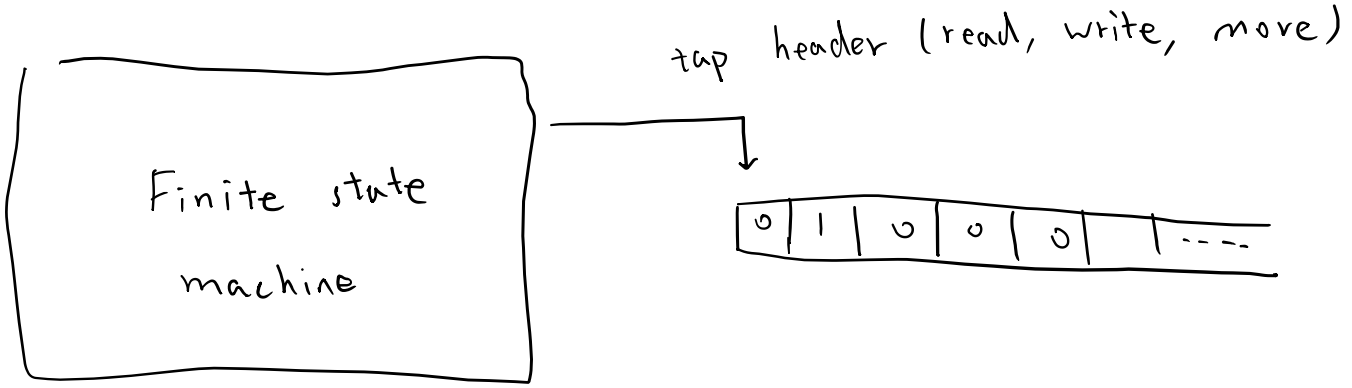




Any non-deterministic FSM can be expressed using a deterministic FSM using power sets. As a result, they have the same computability.

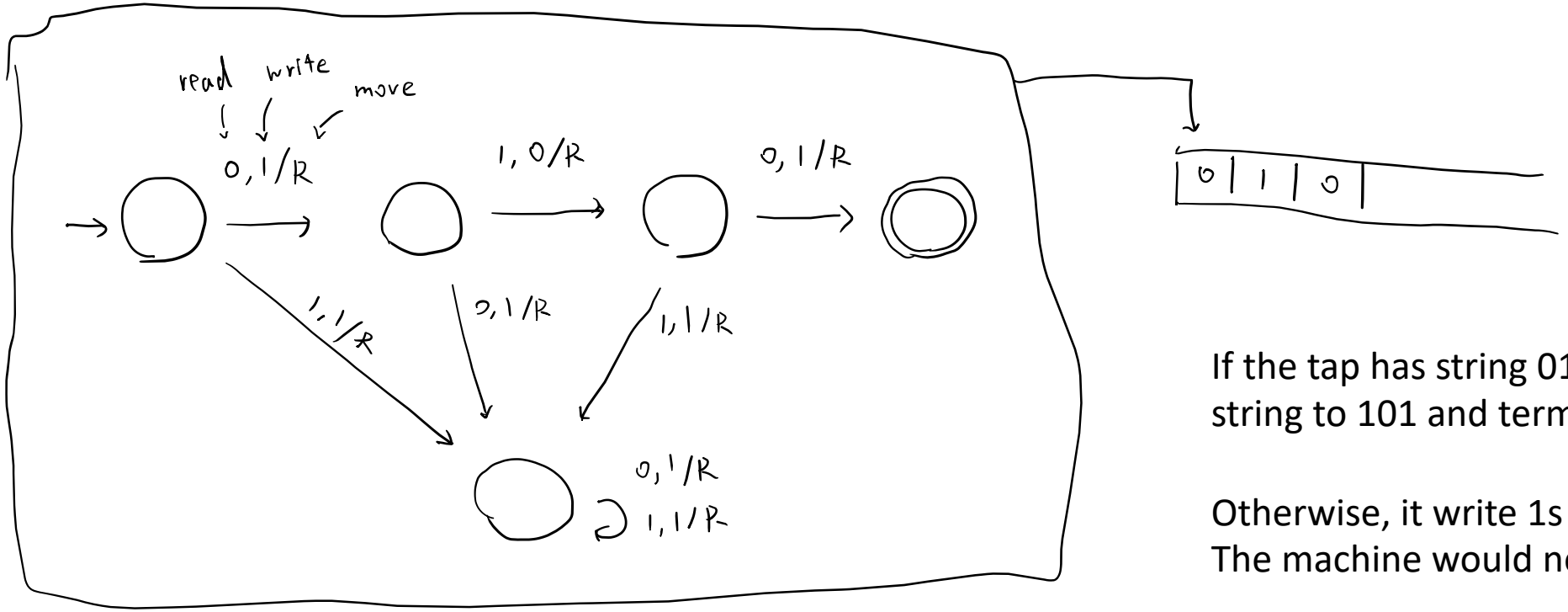
The Turing machine

Previously, the finite state only specifies read. Here, the finite state machine specifies read, write, and move.



The Turing machine

Previously, the finite state only specifies read. Here, the finite state machine specifies read, write, and move.



If the tap has string 010, it overwrites the string to 101 and terminates.

Otherwise, it write 1s to tap non-stop.
The machine would never halt in this case.

The Turing model

With the Turing model, you can design a machine that acts on input (from tap) and produces output (to tap) accordingly. Any mathematical logic can be implemented on the Turing model. However, the goal of Turing machine is not to design a machine that handles a specific problem but to use it to represent **logic**.

Turing completeness:

If a language, grammar, or machine can simulate the Turing machine and it can be simulated by the Turing machine, then it is Turing complete meaning it has the same computability as a Turing machine.

Facts:

Non-deterministic Turing machine, modern computer, and quantum computer are all Turing complete.

Question:

Is our brain Turing complete? (General AI problem)

The Turing model

The Turing machine is a universal computer model.

However, Turing proposed this model to solve the halting problem. He invented computers by accident.

Half time questions

The decision problem (Entscheidungsproblem)

Is there a universal algorithm answers yes or no to a question?

Can an algorithm answers yes or no to whether an assumption leads to a conclusion?

Is this problem decidable? Does the algorithm itself exists in the first place?

Tricky part: It is not possible work out all logical questions.

Reduce to The halting problem

The decision problem can be reduced to the halting problem.

Let’s use a different logic to approach the decision problem:

> Assume there is a black box machine that decides yes or no to any question.
>
> The problem becomes whether this black box machine ever gives an answer. Would it halt and answer yes or no? Or would it keep computing forever?

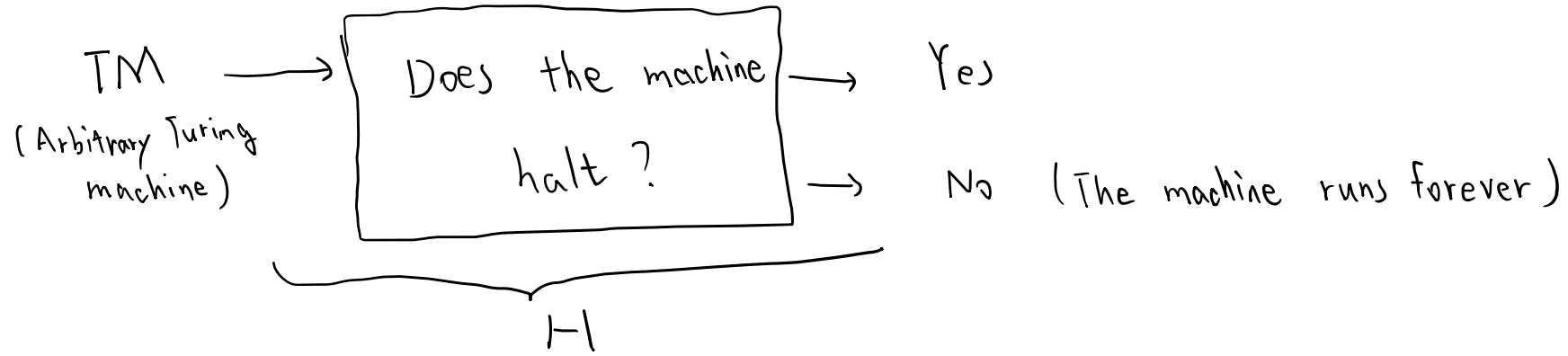
Would the machine give a decisive answer is the same as would the machine halt.

The halting problem

Is it possible to predict if a machine would ever halt?

Proof:

Assume there is a Turing machine H solves the halting problem. H takes an arbitrary machine as its input. H answers yes or no to whether the arbitrary machine would halt (or run forever)?

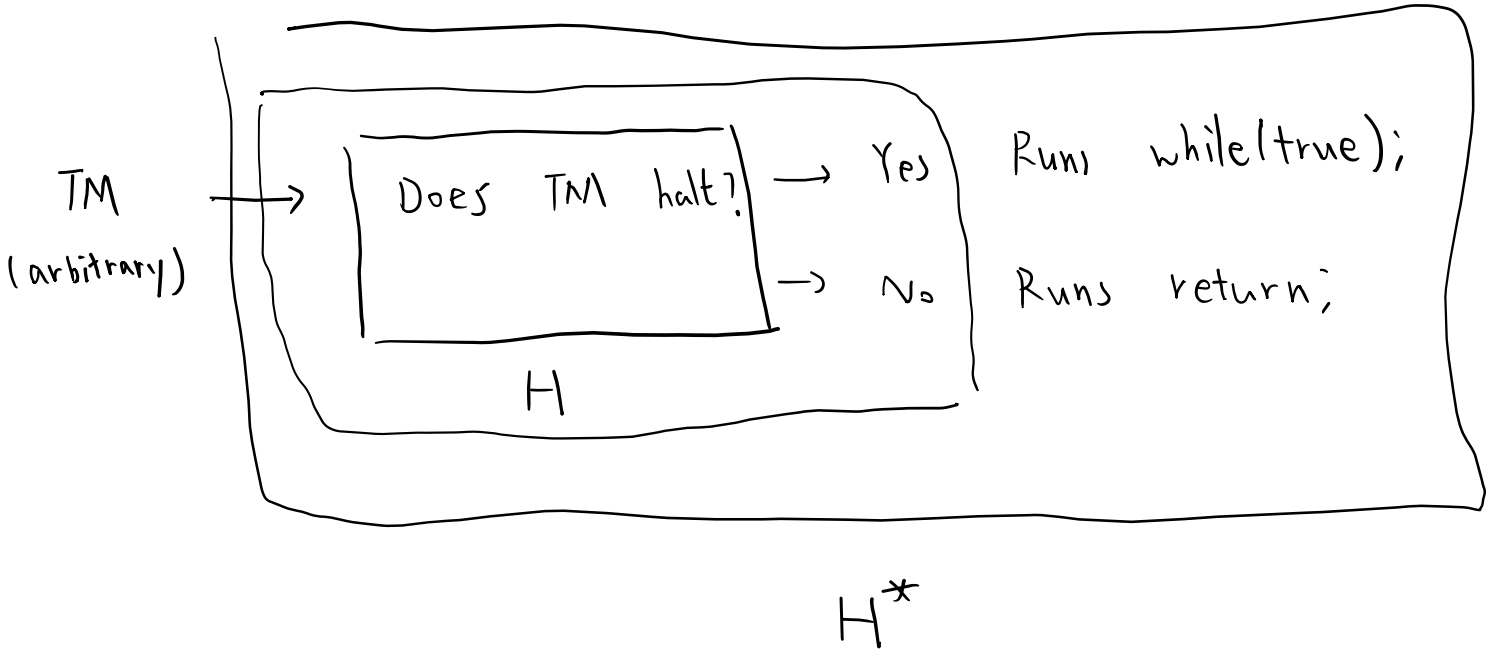


Construct a new machine H* using the machine H, and H* works as follows:

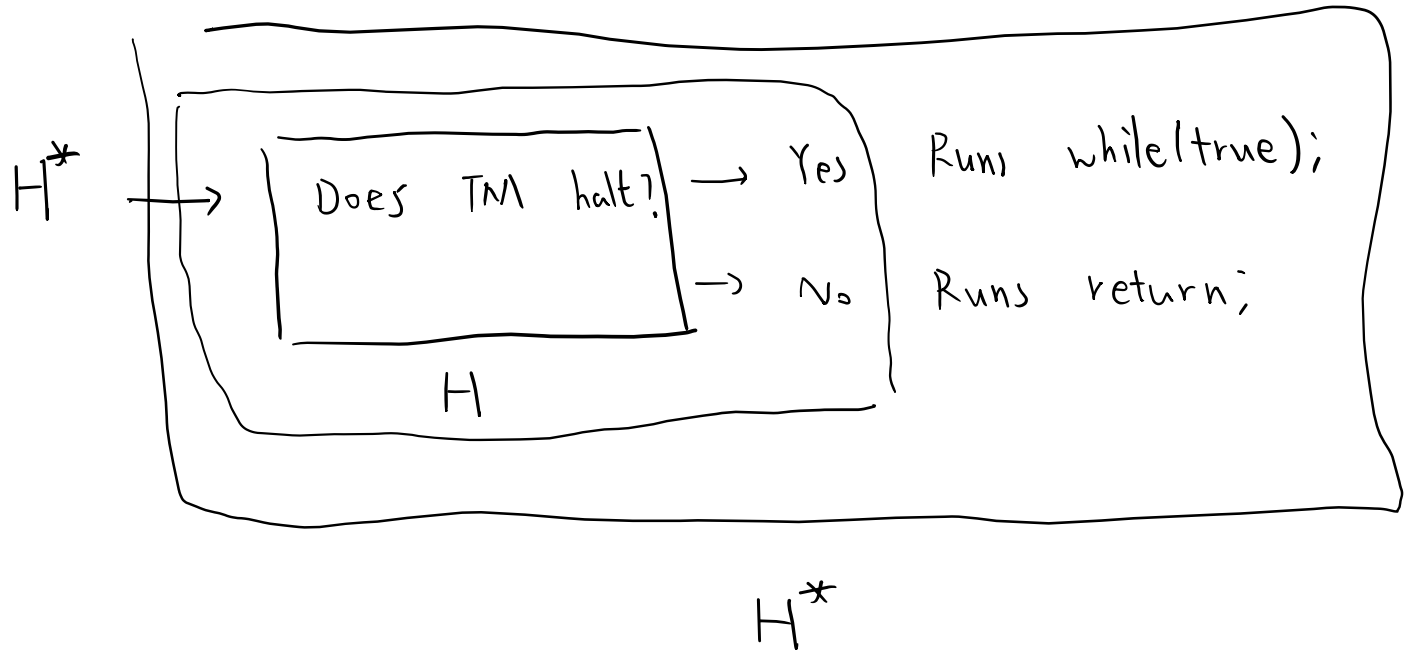
H* takes an arbitrary machine as its input. H* instantly feed the input to machine H.

If H answers Yes, H* loops forever. E.g., while(true);

If H answers No, H* halts. E.g., return;



Now, let's feed H* as the arbitrary input to machine H*.



If H decides that H* halts, then H* loops forever which does not halt. Contradiction!

If H decides that H* does not halt, then H* halt. Contradiction!

} Paradox !

Therefore, H does not exist in the first place. The halting problem is not decidable. The decision problem is not decidable.

What is yet to be solved in theoretical CS?

P vs. NP (It's one of seven Millennium Prize Problems)

Goal: Prove either P = NP or P ≠ NP

P: a set of problem can be solved in polynomial time by a **deterministic** Turing machine. E.g., Sorting

NP: a set of problem can be solved in polynomial time using a **non-deterministic** Turing machine. E.g., Chinese Postman Problem
https://en.wikipedia.org/wiki/List_of_NP-complete_problems

Directions: Prove (one of NP) = P. Then prove if the entire NP set reduce to the one problem in NP set.