Theory of Computation

Instructor: Kai Cai

Period: 2019.10-2020.02

Introduction

In this course you will learn

1. Computation models [Imitation Game]
   - Finite automaton
   - Push-down automaton
   - Turing machine Alan Turing

2. Computability
   - Decidable
   - Undecidable

3. Complexity: P and NP

In this course you will learn

1. Computation models

- Finite automaton

e.g. text, numbers, variable names: x=0.1

- Push-down automaton

e.g. grammar of programming languages: begin…end

- Turing machine

= real computer

In this course you will learn

2. Computability

What problems can be solved, or cannot be solved?

- Decidable

e.g. Given a map of JR routes in Osaka, determine if one can go from Sugimotocho to Morinomiya.

- Undecidable

e.g. Given a program, determine if it always terminates.

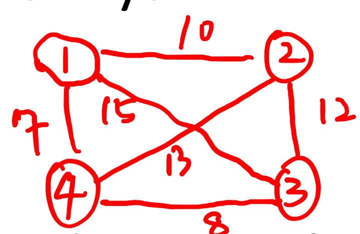
In this course you will learn

3. Complexity

What problems can be solved fast, or slow?

- P

e.g. Given a map of JR routes in Osaka, determine if one can go from Sugimotocho to Morinomiya.



- NP

$\boxed{2^n}$

e.g. Given a list of cities and the distances between each pair of cities, find the *shortest* route that visits each city once and returns to the origin city (travelling salesman problem).

In this course you will learn

3. Complexity

$1M "Millennium Problem":

Is P = NP?

Math training

In this course you will exercise many maths:

- Set
- Logic
- Proof

Course website

https://www.control.eng.osaka-cu.ac.jp/teaching/compute2019

Cellphone

Finite Automaton

“AUTOMATON” = “SELF-MOVER”

Homer’s *Iliad* - 18, lines 373-377

Twenty tripods [Hephaistos] crafted, to stand around … his house. At the base of each he placed golden wheels, so these self-movers [*hoi automatoi*] might enter the divine assembly, and return back to the house, a wonder to behold!

- Computation theory begins with this question:

  What is a good math model for a computer?

- We will introduce several computational models, with different features

- We begin with the simplest and important one:

  finite automaton

Example: automatic door

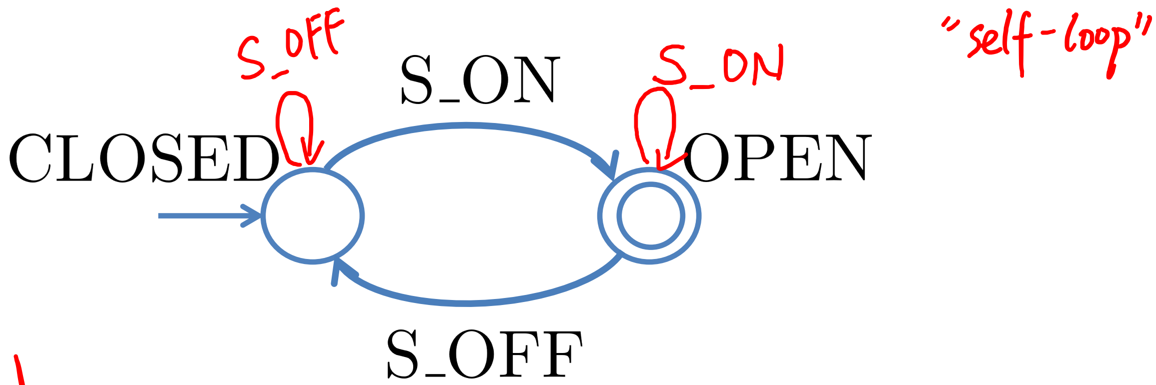
Door has 2 states: CLOSED, OPEN

There are 2 conditions (or events):

1) S_ON: sensor detects a person
2) S_OFF: no person is detected

Example: automatic door

Design a state transition diagram for the door:



Initial

Initial State: CLOSED

initial & accept state

Accept

Accept (Marker) State: OPEN

1-step state

1-step transition: e.g. (CLOSED, S_ON)=OPEN

2-step

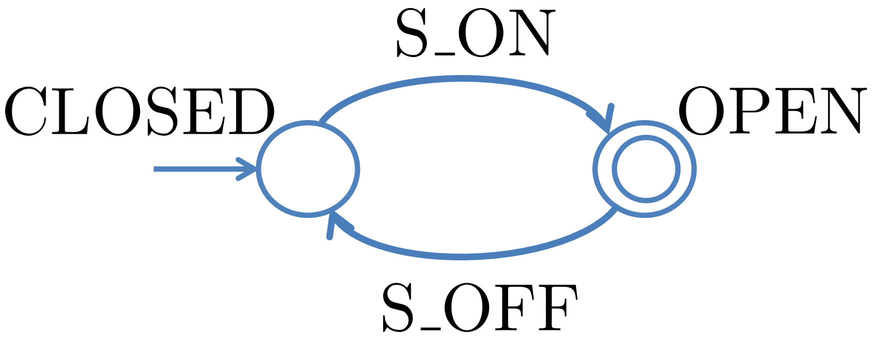
2-step transition: e.g. (OPEN, S_OFF.S_ON)=OPEN

k-step

k-step transition: e.g. (CLOSED, S_ON. S_OFF. S_ON) = OPEN

Example: automatic door

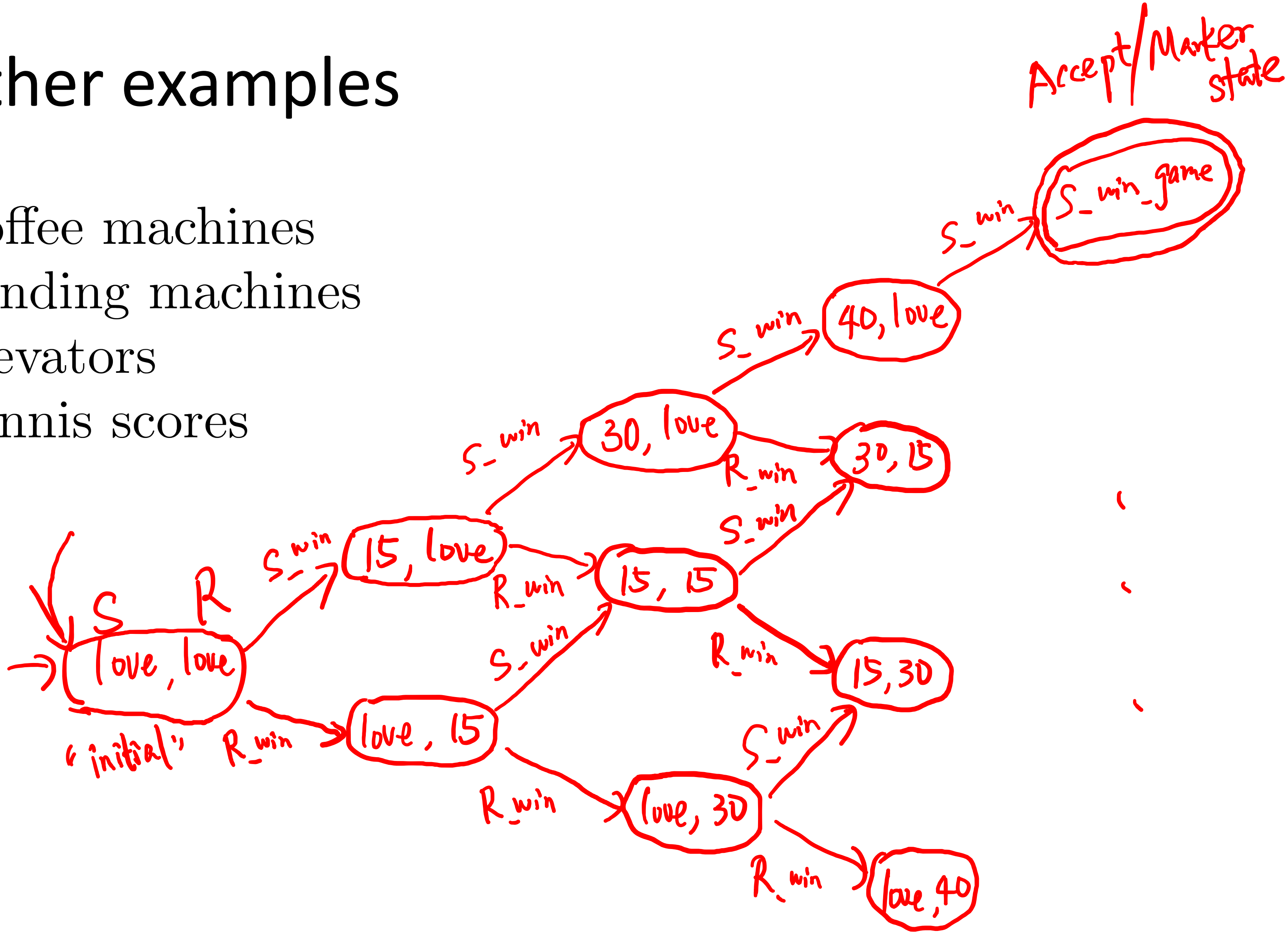
Design a state transition diagram for the door:



This is a simple computer, with just 1-bit memory.
This is called a finite automaton.

Other examples

Coffee machines
Vending machines
Elevators
Tennis scores
...



Aside: Set

A *set* is a collection of objects.

e.g. $S = \{a, b, c\}$

a is called an *element* of S: $a \in S$

the *size* of S is the number of its elements: $|S| = 3$

S is a *finite* set if $|S|$ is finite

e.g. $T = \{x \mid x > 0 \text{ \& } x \text{ is even}\}$

$24 \in T$, but $25 \notin T$

T is an *infinite* set if $|T| = \infty$

Aside: Set

Let S be a set.

A *subset* of S is a subcollection of elements of S.

e.g. $S = \{a, b, c\}$

$\{a, b\} \subseteq S$, $\{a\} \subseteq S$

e.g. $T = \{x \mid x > 0 \text{ \& } x \text{ is even}\}$

$\{2, 222, 22222\} \subseteq T$

Two special subsets: $\emptyset \subseteq S$, $S \subseteq S$ (always)

Aside: Cartesian Product

S, T

Let S, T be two sets.

The *Cartesian product* of S, T is a set of pairs of elements:

$$S \times T = \{(s, t) \mid s \in S \ \& \ t \in T\}$$

e.g. $S = \{a, b, c\}$

$T = \{x \mid x > 0 \ \& \ x \text{ is even}\}$

$S \times T = \{(a,2), (b,6), (a,8), \ldots\}$

Aside: Cartesian Product

Let S, T be two sets.

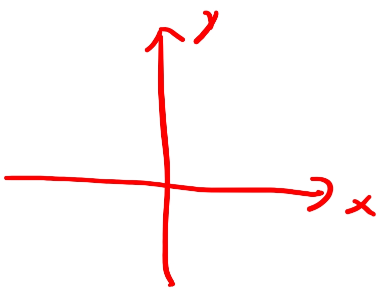
The *Cartesian product* of S, T is a set of pairs of elements:

$$S \times T = \{(s,t) \mid s \in S \;\&\; t \in T\}$$

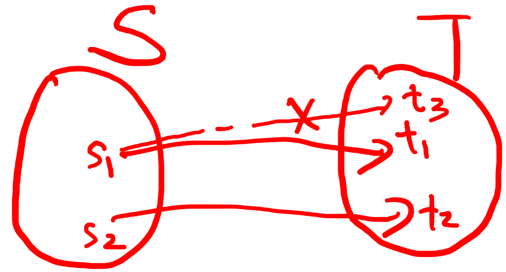
e.g. $S = \{x \mid x \in \mathbb{R}\}$

$T = \{y \mid y \in \mathbb{R}\}$

$S \times T = \mathbb{R}^2 = \{(x,y) \mid x, y \in \mathbb{R}\}$



Aside: Function



function

A *function* $f : S \to T$ is a mapping that assigns each element $s \in S$ a unique element $t \in T$

i.e. $\cancel{f(s) = t}$ $f(s) = t$

Call S *domain*, and T *codomain* of function f

e.g. $S = \{a, b, c\}$

$T = \{x \mid x > 0 \text{ \& } x \text{ is even}\}$

$f : S \to T$

$f(a) = 2$

$f(b) = 22$

$f(c) = 222$

Aside: Function

A *function* $f : S \to T$ is a mapping that assigns each element $s \in S$ a unique element $t \in T$ i.e. $f(s) = t$

Call S *domain*, and T *codomain* of function f

e.g. $S = \{a, b, c\}$

$T = \{x \mid x > 0 \text{ \& } x \text{ is even}\}$

$$f : \underbrace{S \times T}_{\text{domain}} \to \underbrace{\{0, 1\}}_{\text{codomain}}$$

$\cancel{f(a,2)}$

$f(a,2) = 0$

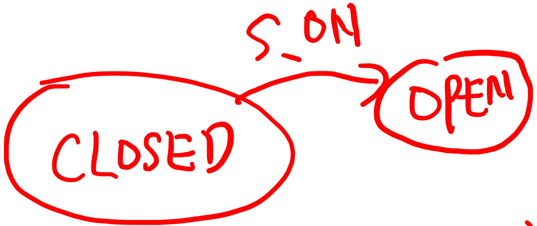
$f(b,20) = 1$

Finite automaton

finite

A ~~finite~~ automaton **G** is a five tuple

$\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_a)$, where



Q :

Q : state set; a finite set of states

Σ :

Σ : alphabet; a finite set of symbols

(CLOSED, S_ON) = OPEN

δ :

$\delta : \underbrace{Q \times \Sigma}_{\text{domain}} \to \underbrace{Q}_{\text{codomain}}$: state transition function

δ(CLOSED , S_ON) = OPEN

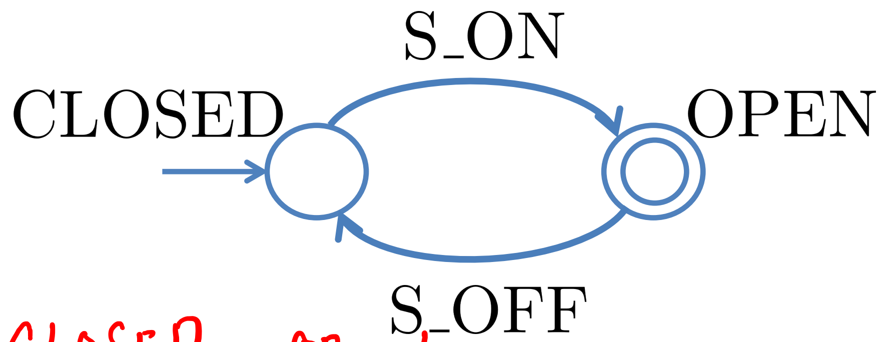
$q_0 \in Q$: initial state

q_0 $q \in Q$ $\xrightarrow{\sigma \in \Sigma}$ $q' \in Q$; $\delta(q, \sigma) = q'$

$Q_a \subseteq Q$: subset of accept states

Q_a

Example: automatic door



1 $Q = \{CLOSED, OPEN\}$

2 $\Sigma = \{S\_ON, S\_OFF\}$

3 $\delta(CLOSED, S\_ON) = OPEN$; $\delta(OPEN, S\_OFF) = CLOSED$

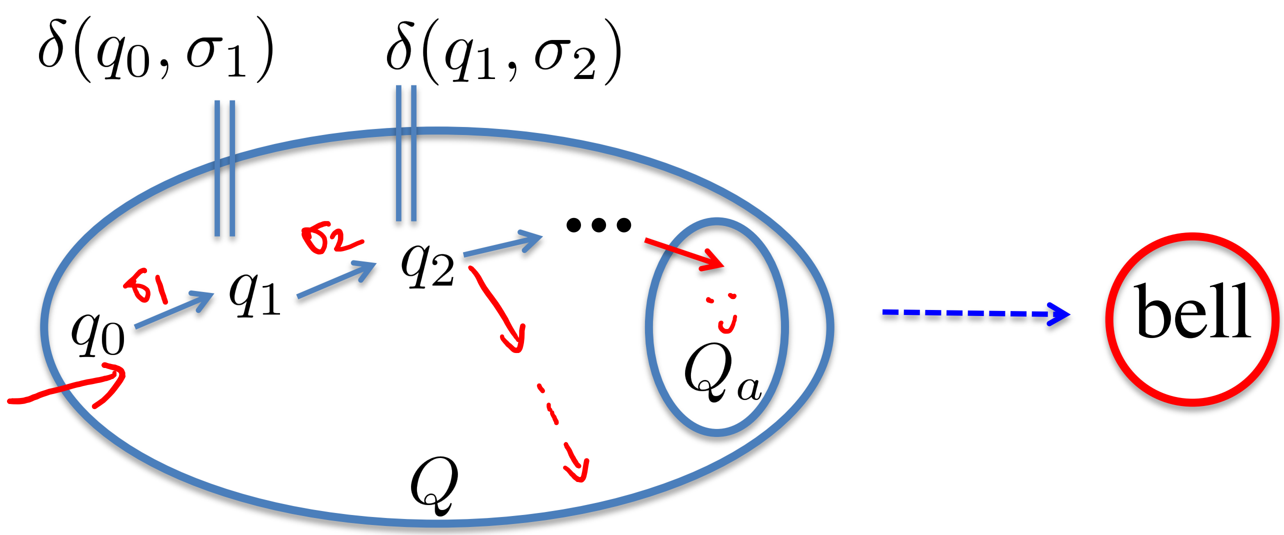
$\delta : Q \times \Sigma \to Q$

4 $q_0 = CLOSED$

5 $Q_a = \{OPEN\}$

How does a finite automaton work

A finite automaton $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_a)$



Internal state transitions

Beeps when a transition enters an accept state