Formal Languages and Automata Theory

Siu On CHAN

Fall 2019

Chinese University of Hong Kong

Welcome to 3130

https://www.cse.cuhk.edu.hk/~siuon/csci3130

Tentative syllabus and schedule

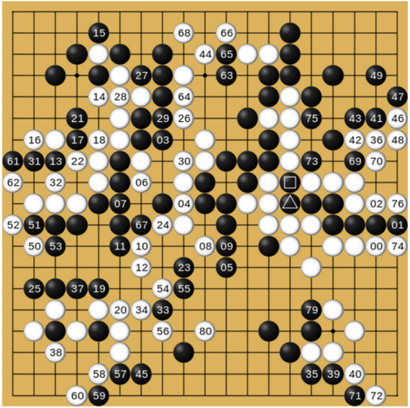
Reference book

Introduction to the Theory of Computation, Michael Sipser

Please sign up on piazza.com and ask questions

Or come to our office hours

Computers can beat experts at Go



Source: Wikipedia on AlphaGo versus Lee Sedol

Computers can compose essays

https://openai.com/blog/better-language-models/

Topic: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

> The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.
> Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.
> Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.
> Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.
> Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them — they were so close they could touch their horns. ...

Is there anything that a computer cannot do?

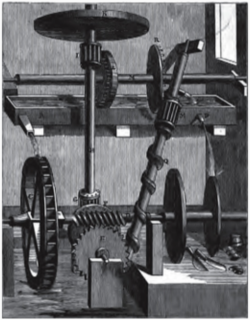
Why care about the impossible?

Example from Physics:

Since the Middle Ages, people tried to design machines that use no energy

Later physical discoveries forbid creating energy out of nothing

Perpetual motion is impossible



“water screw” perpetual motion machine

Understanding the impossible helps us to focus on the possible

Laws of computation

Just like laws of physics tell us what are (im)possible in nature...

$$\Delta U = Q + W \qquad \mathrm{d}S = \frac{\delta Q}{T} \qquad S - S_0 = k_B \ln \Omega$$

Laws of computation tell us what are (im)possible to do with computers

Part of computer science

To some extent, laws of computation are studied in automata theory

Exploiting impossibilities

Certain tasks are believed impossible to solve quickly on current computers

Given $n = pq$ that is the product of two unknown primes, find p and q

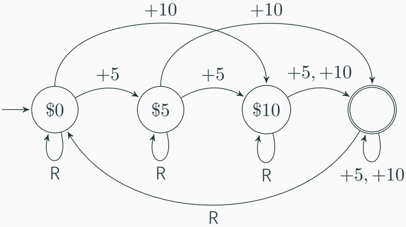
Building block of cryptosystems

Candy machine

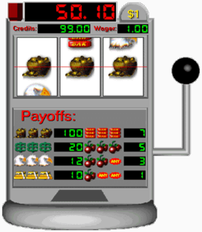
Machine takes $\$5$ and $\$10$ coins

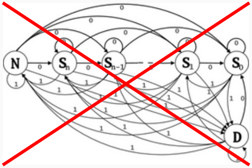
A gumball costs $\$15$

Actions: $+5$, $+10$, Release



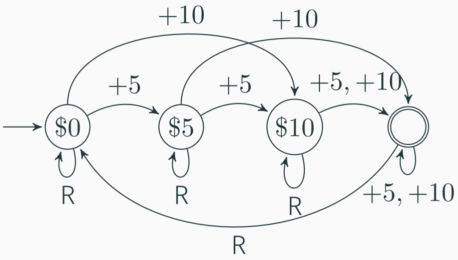
Slot machine





Why?

Different kinds of machines



Only one example of a machine

We will look at different kinds of machines and ask

- what kind of problems can this kind of machines solve?
- What are impossible for this kind of machines?
- Is machine A more powerful than machine B?

Machines with different resources in this course

finite automata	Devices with a small amount of memory These are very simple machines
push-down automata	Devices with unbounded memory that can be accessed in a restricted way Used to parse grammars
Turing machines	Devices with unbounded memory These are actual computers
time-bounded Turing Machines	Devices with unbounded memory but bounded running time These are computers that run fast

Course highlights

- Finite automata
  Closely related to pattern searching in text

  Find (ab)*(ab) in abracadabra
- Grammars
  - Grammars describe the meaning of sentences in English, and the meaning of programs in Java (or any language)
  - Useful for natural language processing and compilers

Course highlights

Turing machines

- General model of computers, capturing anything we could ever hope to compute
- But there are many things that computers cannot do

Given the code of a program, tell if the program prints the string "3130"

Does the program

```
#include <stdio.h>
main(t,_,a)char *a;{return!0<t?t<3?main(-79,-13,a+main(-87,1-_,
main(-86,0,a+1)+a)):1,t<_?main(t+1,_,a):3,main(-94,-27+t,a)&&t==2?_<13?
main(2,_+1,"%s %d %d\n"):9:16:t<0?t<-72?main(_,t,
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+,/+#n+,/#\
;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# \
){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \
iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n' ')# \
}'+}##(!!/")
:t<-50?_==*a?putchar(31[a]):main(-65,_,a+1):main((*a=='/')+t,_,a+1)
  :0<t?main(2,2,"%s"):*a=='/'||main(0,main(-61,*a,
"!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);}
```

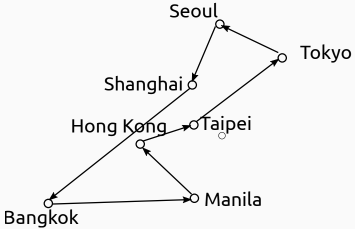
print "3130"?

Formal verification of software must fail on corner cases

Course highlights

Time-bounded Turing machines

- Many problems can be solved on a computer in principle, but takes too much time in practice
- Traveling salesperson: Given a list of cities, find the shortest way to visit them all and return home



- For 100 cities, takes 100+ years to solve even on the fastest computer!

Problems we will look at

Can machine A solve problem B?

- Examples of problems we will consider
  - Given a word s, does it contain “to” as a subword?
  - Given a number n, is it divisible by 7?
  - Given two words s and t, are they the same?
- All of these have “yes/no” answers (decision problems)
- There are other types of problems, like “Find this” or “How many of that” but we won’t look at them

Alphabets and Strings

- Strings are a common way to talk about words, numbers, pairs of numbers
  Which symbols can appear in a string? As specified by an alphabet

An alphabet is a finite set of symbols

- Examples
  $\Sigma_1 = \{a, b, c, d, \ldots, z\}$: the set of English letters
  $\Sigma_2 = \{0, 1, 2, \ldots, 9\}$: the set of digits (base 10)
  $\Sigma_3 = \{a, b, c, \ldots, z, \#\}$: the set of letters plus special symbol #

Strings

An input to a problem can be represented as a string

A string over alphabet Σ is a finite sequence of symbols in Σ

axyzzy is a string over $\Sigma_1 = \{a, b, c, \ldots, z\}$

3130 is a string over $\Sigma_2 = \{0, 1, \ldots, 9\}$

ab#bc is a string over $\Sigma_3 = \{a, b, \ldots, z, \#\}$

- The empty string will be denoted by ε
  (What you get using "" in C, Java, Python)
- Σ^* denotes the set of all strings over Σ
  All possible inputs using symbols from Σ only

Languages

A language is a set of strings (over the same alphabet)

Languages describe problems with "yes/no" answers:

$L_1 =$ All strings containing the substring "to" $\qquad \Sigma_1 = \{\text{a}, \ldots, \text{z}\}$

stop, to, toe are in L_1

ε, oyster are not in L_1

$$L_1 = \{x \in \Sigma_1^* \mid x \text{ contains the substring "to"}\}$$

Examples of languages

$L_2 = \{x \in \Sigma_2^* \mid x \text{ is divisible by } 7\}$ $\Sigma_2 = \{0, 1, \ldots, 9\}$

L_2 contains 0, 7, 14, 21, …

Examples of languages

$L_2 = \{x \in \Sigma_2^* \mid x \text{ is divisible by } 7\}$ $\Sigma_2 = \{0, 1, \ldots, 9\}$

L_2 contains 0, 7, 14, 21, ...

$L_3 = \{s\#s \mid s \in \{a, \ldots, z\}^*\}$ $\Sigma_3 = \{a, b, \ldots, z, \#\}$

Which of the following are in L_3?

ab#ab ab#ba a##a#

Examples of languages

$L_2 = \{x \in \Sigma_2^* \mid x \text{ is divisible by } 7\}$ $\Sigma_2 = \{0, 1, \ldots, 9\}$

L_2 contains 0, 7, 14, 21, ...

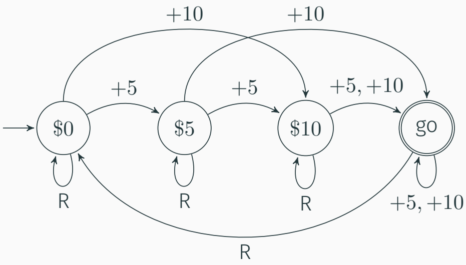
$L_3 = \{s\#s \mid s \in \{a, \ldots, z\}^*\}$ $\Sigma_3 = \{a, b, \ldots, z, \#\}$

Which of the following are in L_3?

ab#ab	ab#ba	a##a#
Yes	No	No

Finite Automata

Example of a finite automaton



- There are states \$0, \$5, \$10, go
- The start state is \$0
- Takes inputs from $\{+5, +10, \mathsf{R}\}$
- The state go is an accepting state
- There are transitions specifying where to go to for every state and every input symbol

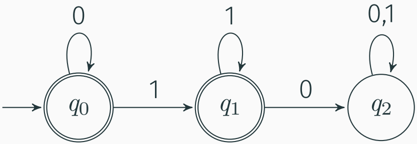
Deterministic finite automaton

A finite automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of states
- Σ is an alphabet
- $\delta : Q \times \Sigma \to Q$ is a transition function
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is the set of accepting states (or final states)

In diagrams, the accepting states will be denoted by double circles

Example



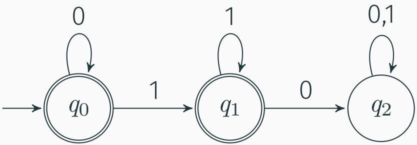
alphabet $\Sigma = \{0, 1\}$
states $Q = \{q_0, q_1, q_2\}$
initial state q_0
accepting states $F = \{q_0, q_1\}$

table of transition function δ

		inputs	
		0	1
states	q_0	q_0	q_1
	q_1	q_2	q_1
	q_2	q_2	q_2

Language of a DFA

A DFA accepts a string x if starting from the initial state and following the transition as x is read from left to right, the DFA ends at an accepting state



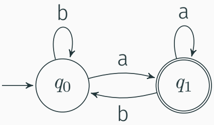
The DFA accepts 0 and 011 but not 10 and 0101

The language of a DFA is the set of all strings x accepted by the DFA

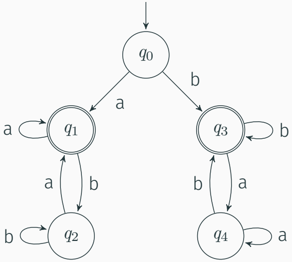
0 and 011 are in the language 10 and 0101 are not

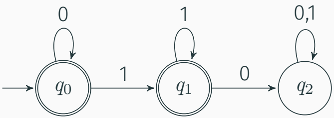
The languages of these DFAs?

$\Sigma = \{a, b\}$



$\Sigma = \{a, b\}$





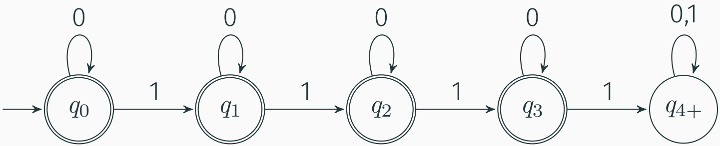
$\Sigma = \{0, 1\}$

Examples

Construct a DFA over $\{0, 1\}$ that accepts all strings with at most three 1s

Examples

Construct a DFA over $\{0, 1\}$ that accepts all strings with at most three 1s



Examples

Construct a DFA over $\{0, 1\}$ that accepts all strings ending in 01

Examples

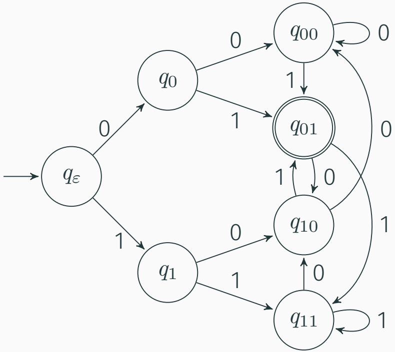
Construct a DFA over $\{0, 1\}$ that accepts all strings ending in 01

Hint: The DFA should "remember" the last 2 bits of the input string

Examples

Construct a DFA over $\{0, 1\}$ that accepts all strings ending in 01

Hint: The DFA should "remember" the last 2 bits of the input string



We will see a much simpler DFA in the next lecture

Construct a DFA over $\{0, 1\}$ that accepts all strings ending in 101