CSCI 3434: Theory of Computation

Lecture 01: Introduction

Ashutosh Trivedi (ashutosh.trivedi@colorado.edu)
Department of Computer Science, University of Colorado Boulder

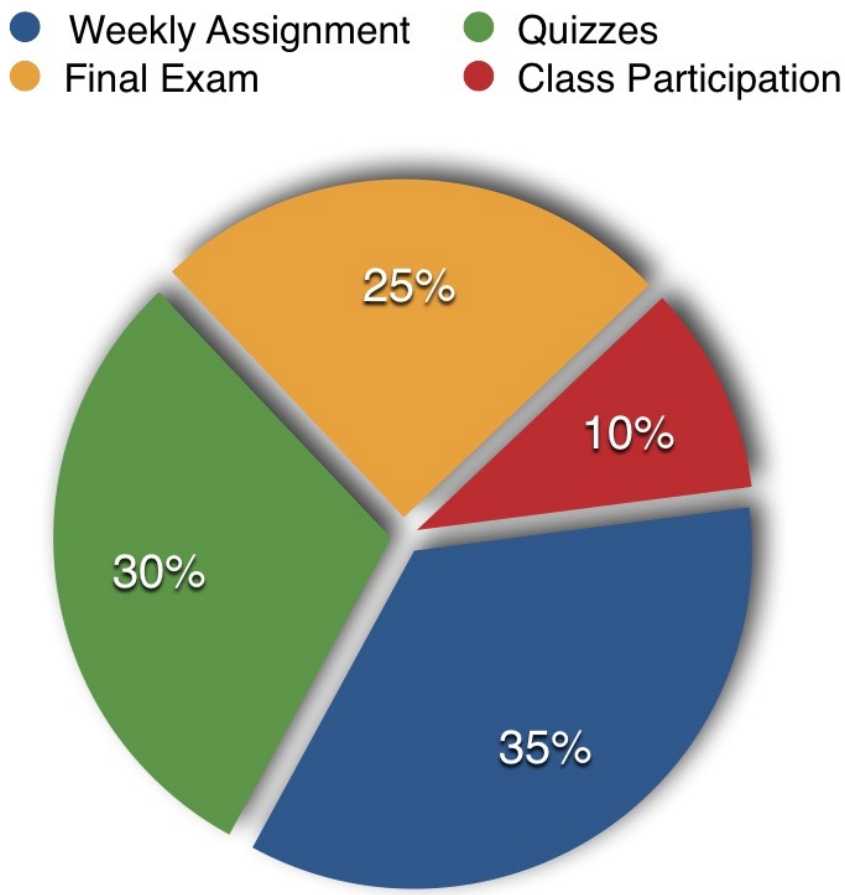
Logistics

- Web-page http://www.cs.colorado.edu/~astr3586/courses/csci3434.html
- Instructor and grading assistant
  - Ashutosh Trivedi (ashutosh.trivedi@colorado.edu)
  - Krithika Balan (krithika.balan@colorado.edu)
- Lectures
  - Tuesday (2:00pm – 3:15pm)
  - Thursday (2:00pm – 3:15pm)
- Office hours
  - Monday (10:00am – 11:00am) or by appointment
- Venue
  - Class meeting location: ECCR 155
  - Office location: ECCS 121C

Logistics (Contd.)

- Requisite
  - Discrete Structures (CSCI 2824) http://www.cs.colorado.edu/~yuvo9296/courses/csci2824/index.html
  - Algorithms (CSCI 3104)
- Textbook
  - *Michael Sipser*. Introduction to the Theory of Computation, PWS Publishing Company.
  - 2nd or 3rd edition
- Other supplemental materials
  - Online notes and readings distributed by instructor
- Moodle (CSCI3434-F16)

Logistics: Grading



Find this week's assignment on Moodle on prerequisites!

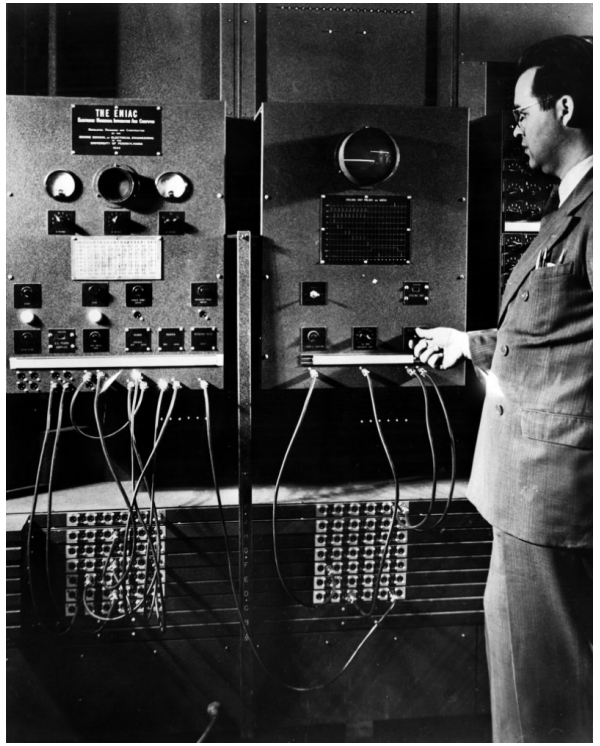
Theory of Computation

What are the fundamental capabilities and limitations of computers?

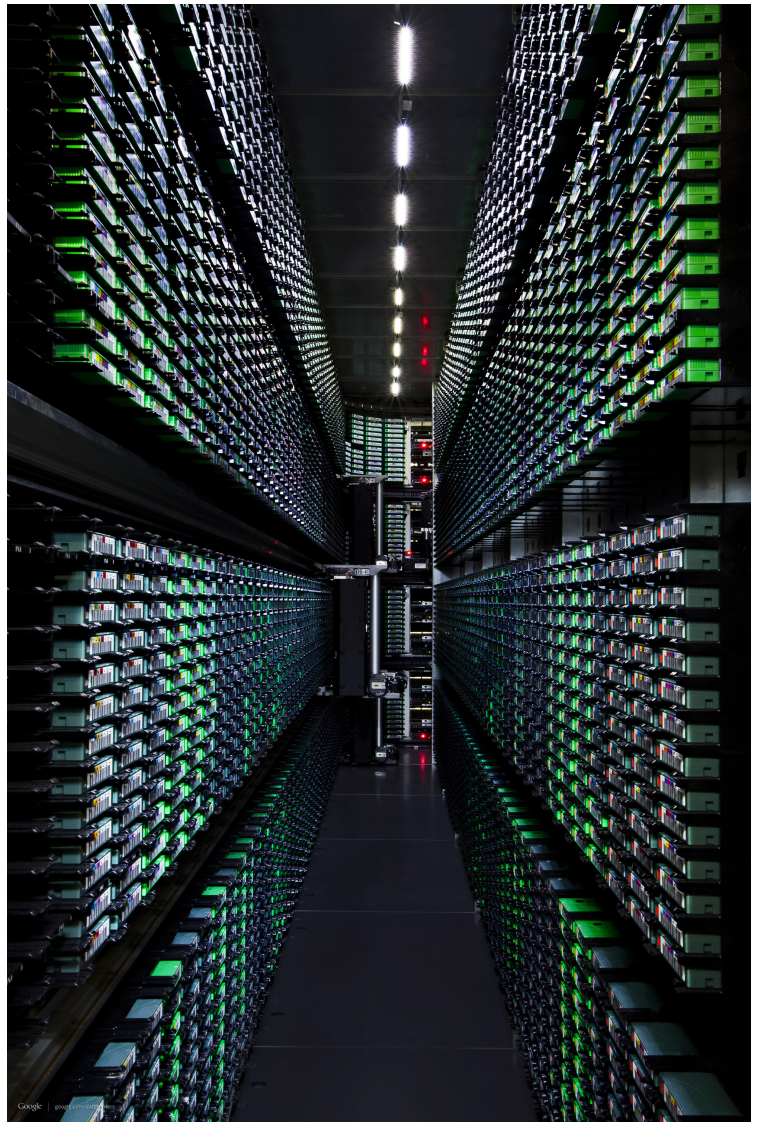
Theory of Computation

Mechanical computer
"*Antikythera*" (2 BC)

Electro-mechanical Computer
"*Bombe*" (1938)

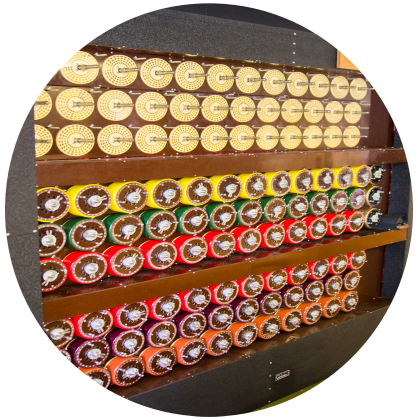
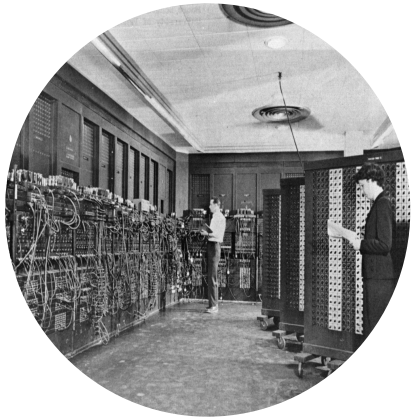


First Electronic general-purpose computer
"ENIAC" (1946)



Datacenters
Google (Now)

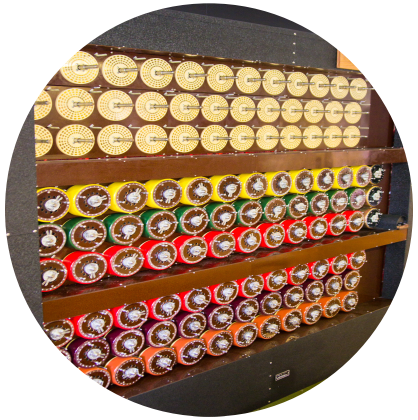
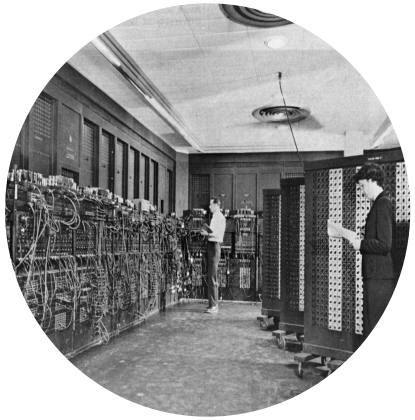
Theory of Computation


What are the fundamental capabilities and limitations of computation?

- What do we mean by computation?
- What is a problem?
- Are all problems computable?
- What is an “efficient” computation?
- Are some problems inherently more difficult than others?

Theory of Computation


What are the fundamental capabilities and limitations of computers?

- How do we model "computational machines"?
- Are all computational machines **equally powerful**?
- Why to study computationally less powerful models?
- Why a practically-oriented computer-programmer should learn theory of computation?

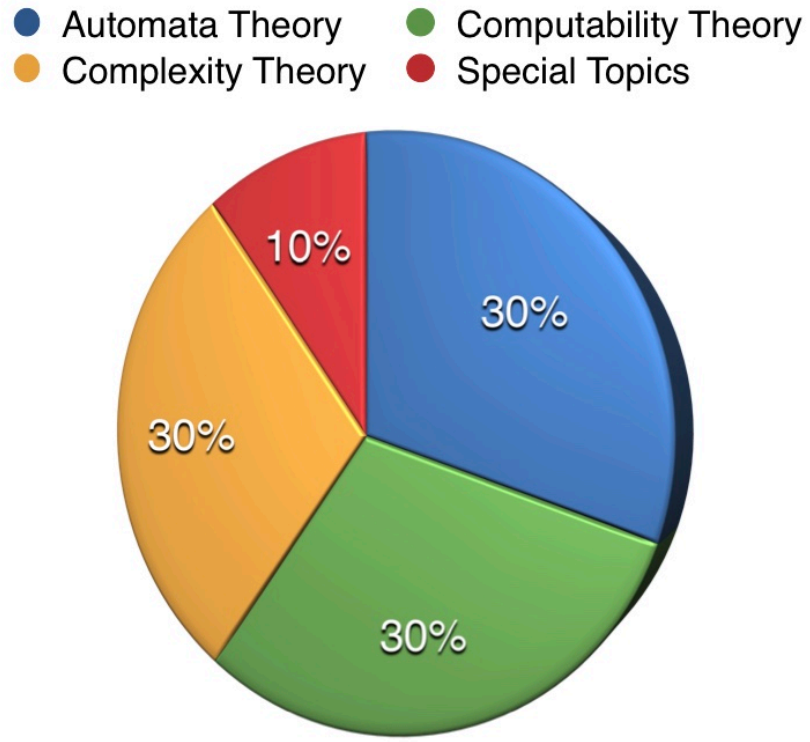
Theory of Computation

Automata Theory

- Formalization of the notion of problems via **formal languages**
- Formalization of the notion of computation using "abstract computing devices" called **automata**
- Understanding a hierarchy of classes of problems or formal languages (regular, context-free, context-sensitive, decidable, and undecidable)
- Understanding a hierarchy of classes of automata (finite automata, pushdown automata, and **Turing machines**)
- Understanding applications to pattern matching, parsing, and programming languages

Computability Theory

Complexity Theory



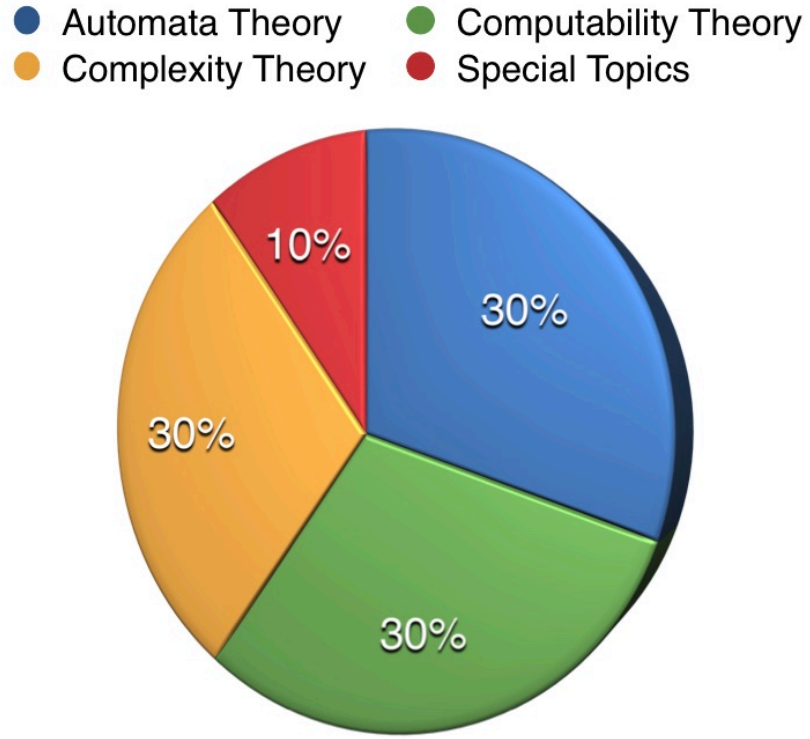
Theory of Computation

Automata Theory

Computability Theory

- Understanding **Church-Turing thesis** (Turing machines as a notion of "general-purpose computers")
- Understanding the concept of **reduction** , i.e., solving a problem using a solution (abstract device) for a different problem
- Understanding the concept of **undecidability** , i.e., when a problem can not be solved using computers

Complexity Theory



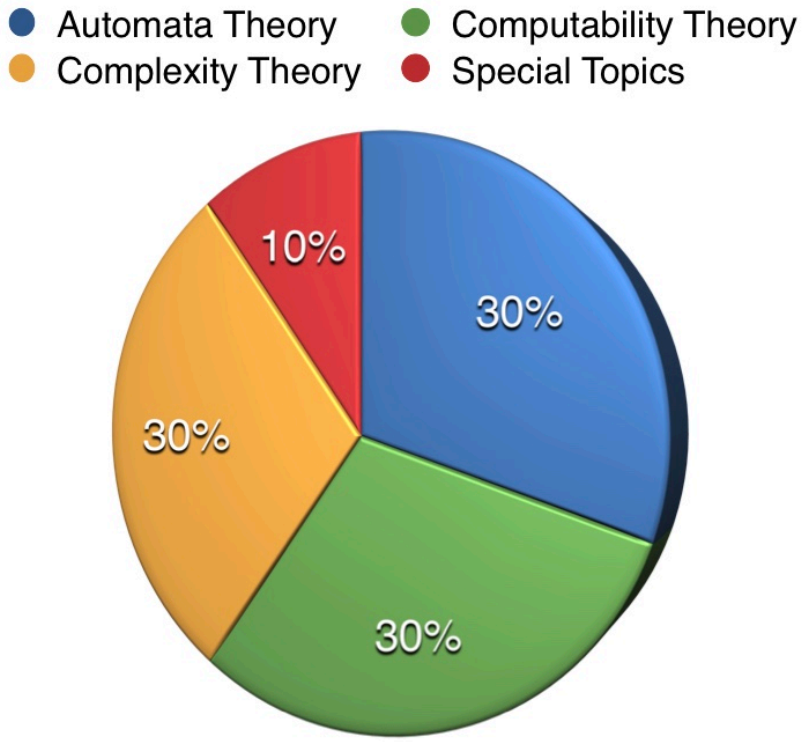
Theory of Computation

Automata Theory

Computability Theory

Complexity Theory

- **Complexity classes** : how to classify decidable problems based on their time and space requirements
- Complexity classes P and NP
- When a problem is called **intractable** (NP-completeness)
- Using reductions to prove problems intractable
- Space-complexity classes L and NL, PSPACE, and so on



Theory of Computation: Schedule

- Week 1 – Week 7 : Automata Theory (In-Class Quiz I)
- Week 7 – Week 11: Computability Theory (In-Class Quiz II)
- Week 11 – Week 16: Complexity Theory (In-Class Quiz III)
- Week 16: Special Topic

Part I: Automata Theory

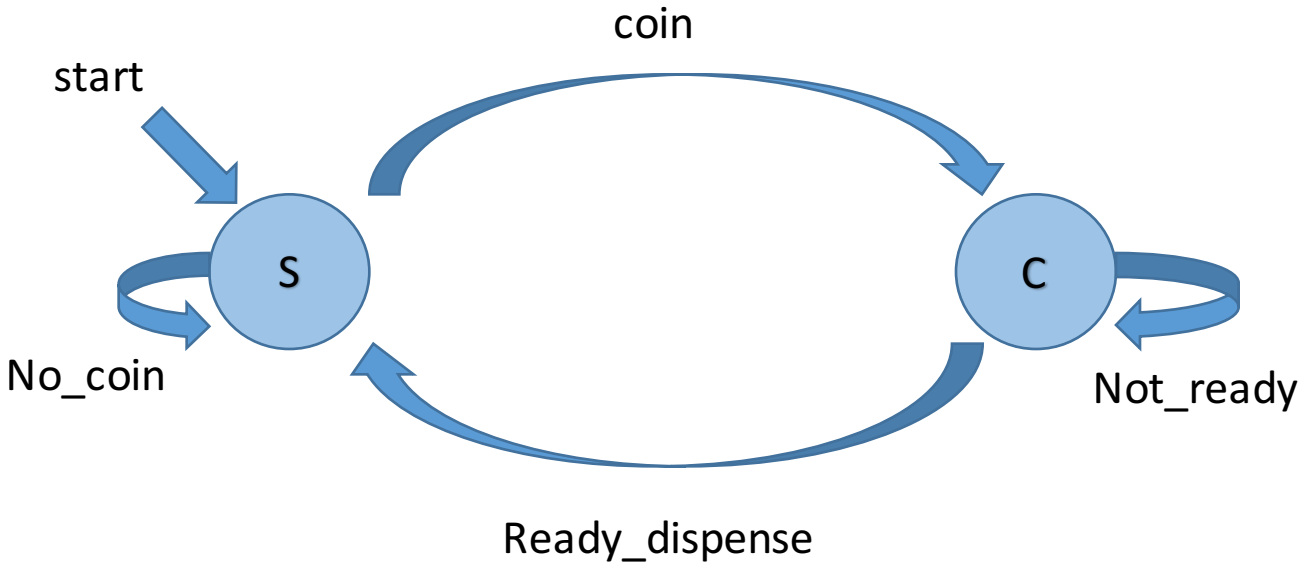
Introduction to Automata Theory

What's an automaton?

1. A moving mechanical device made in imitation of a human being.
2. A **machine** that performs a **function** according to a **predetermined set** of coded **instructions**.

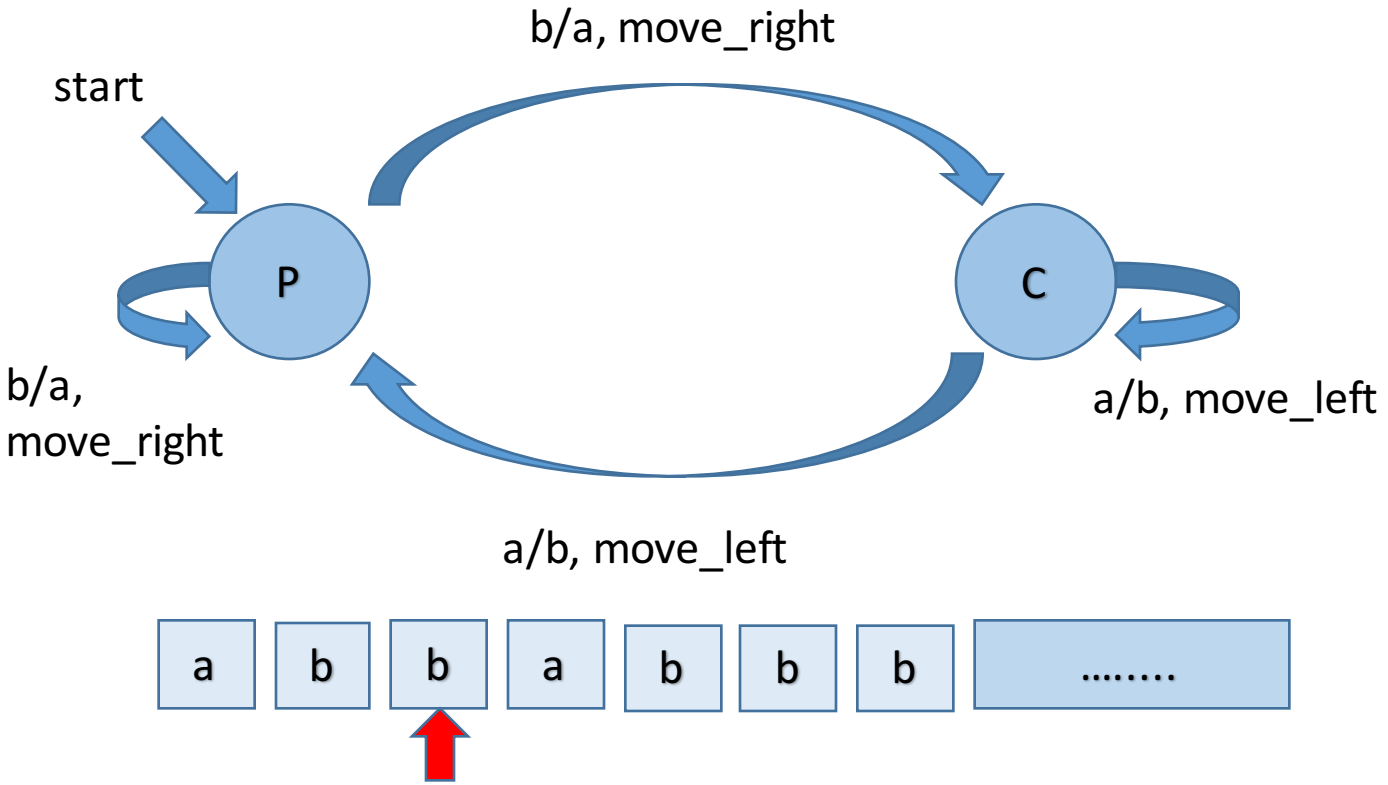
Introduction to Automata Theory

- Finite instruction machine with finite memory (*Finite State Automata*)

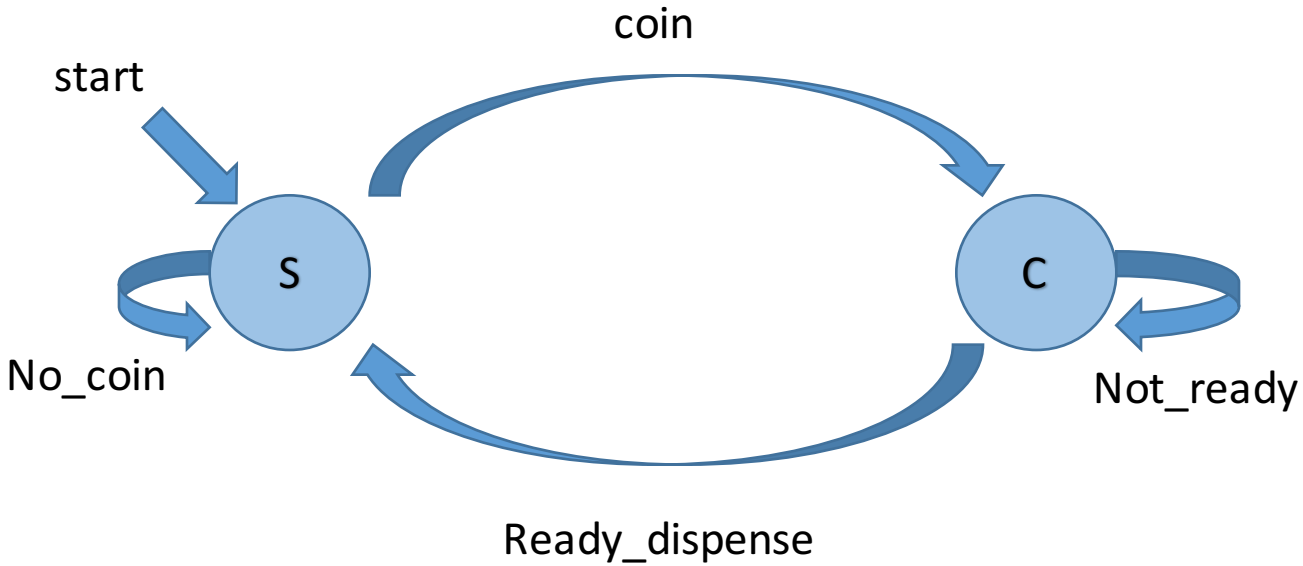


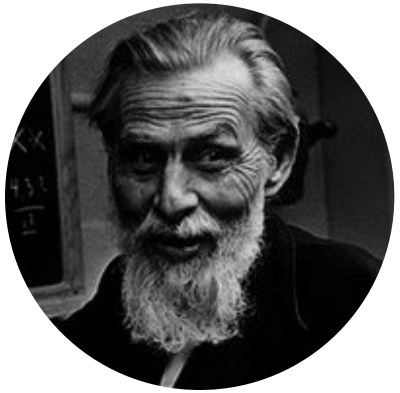
Introduction to Automata Theory

- Finite instruction machine with unbounded memory (*Turing machine*)



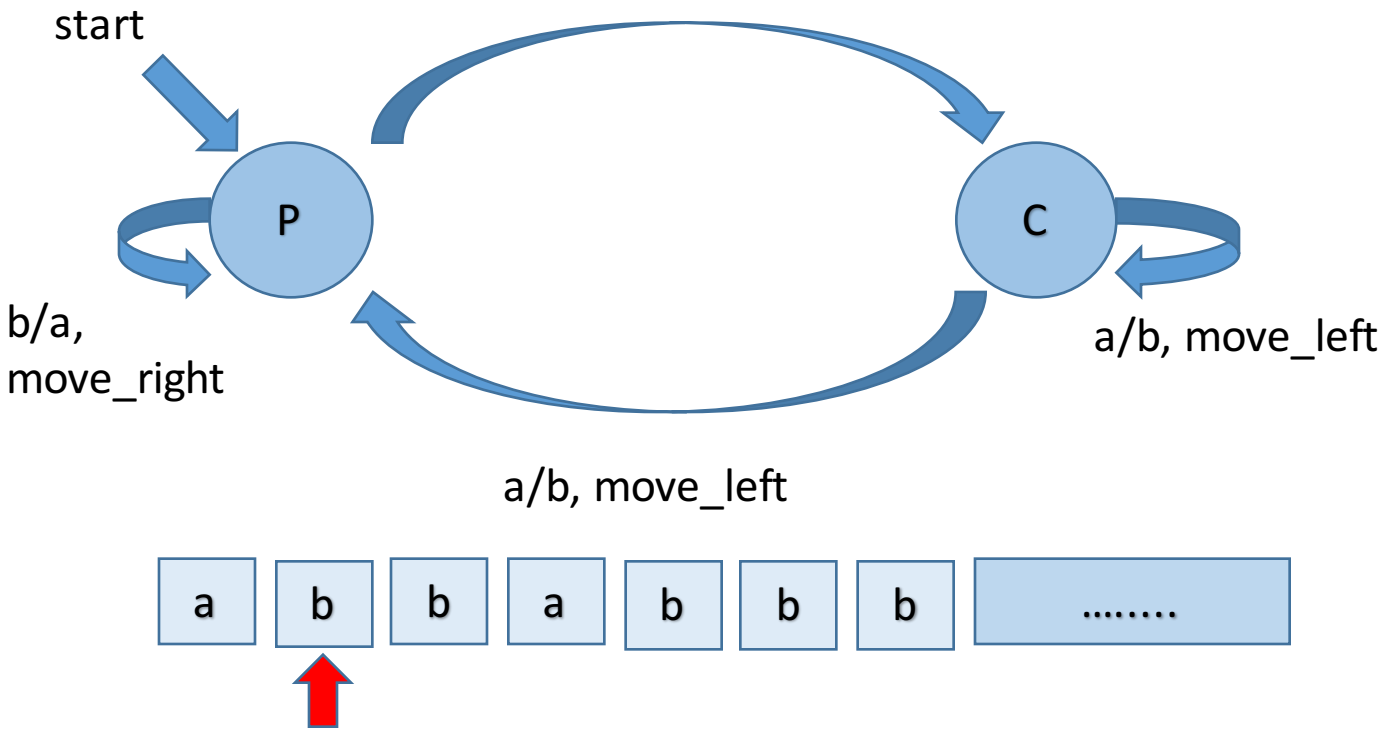
Finite State Automata





- Introduced first by two neuro-psychologists **Warren S. McCullough** and **Walter Pitts** in 1943 as a model for human brain!
- Finite automata can naturally model **microprocessors** and even **software programs** working on variables with bounded domain
- Capture so-called **regular sets** of sequences that occur in many different fields (logic, algebra, regular Expressions)
- Nice theoretical properties
- Applications in **digital circuit/protocol verification**, **compilers**, **pattern recognition**, and so on.

Turing Machines



- Introduced by **Alan Turing** as a simple model capable of expressing any imaginable computation
- Turing machines are widely accepted as a synonyms for algorithmic computability (Church-Turing thesis)
- Using these conceptual machines Turing showed that *first-order logic validity problem* is **non-computable**.
- I.e. there exists some problems for which you can never write a program no matter how hard you try!

Discrete Mathematics: Review

Discrete Mathematics: Review

- A set is a collection of objects, e.g.
  - $A = \{a, b, c, d\}$ and $B = \{b, d\}$
  - Empty set $\emptyset = \{\}$ (why it is not same as $\{\emptyset\}$)
  - $\mathbb{N} = \{0, 1, 2, 3, \dots\}$ and $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$
  - $\mathbb{Q}$ is the set of rational numbers.
  - $\mathbb{R}$ is the set of real numbers.
- $a \in A$: element of a set, belongs to, or contains
- Subset of $A \subseteq \mathbb{N}$, or proper subset of $A \subset \mathbb{N}$
- Notions of set union, intersection, difference, and disjoint
- Power set 2^A of a set A
- Partition of a set

Discrete Mathematics: Review (Contd.)

- A ordered pair is a pair (a, b) of elements with natural order
- Similarly we define n-tuples, triplets, and so on
- Cartesian product $\mathrm{A}\times B$ of two sets is the set of orderd pairs

$$\mathrm{A}\times B = \{(a, b) : a \in A \; and \; b \in B\}$$

- Binary relation R on two sets $A \; and \; B \; is \; a \; subset \; of \; \mathrm{A}\times B$
- A function (or mapping) f from set A to B is a binary relation s.t.

  for all $a \in A$ we have that $(a, b) \in f$ and $(a, b') \in f$ implies that $b = b'$.
- We often write $f(a)$ for the unique element b such that $(a, b) \in f$.

Discrete Mathematics: Review (Contd.)

- Function $f: A \to B$ is one-to-one if for any two distinct elements $a, b \in A$ we have that $f(a) \neq f(b)$.
- Function $f: A \to B$ is onto if for every element $b \in B$ there is an element $a \in A$ such that $f(a) = b$.
- Function $f: A \to B$ is called bijection if it is both one-to-one and onto.
- Recall definitions of
  - Reflexive, Symmetric, and Transitive relations,
  - and Equivalence relation.

Cardinality of a Set

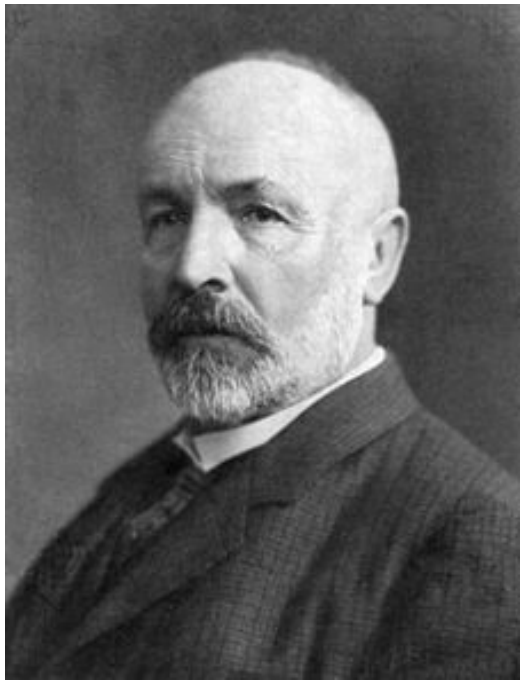
- Cardinality $|S|$ of a set S, e.g. $|A| = 4$ and $|\mathbb{N}|$ is an infinite number.
- Two sets have same cardinality if there is a bijection between them.
- A set is countably infinite (or denumerable) if it has same cardinality as $\mathbb{N}$.
- A set is countable if it is either finite or countably infinite.
- A transfinite number is a cardinality of some infinite set.

Theorem: Cardinality

Theorem

1. *The set of integers is countably infinite. (idea: interlacing)*
2. *The union of a finite number of countably infinite sets is countably infinite as well. (idea: dove-tailing)*
3. *The union of a countably infinite number of countably infinite sets is countably infinite.*
4. *The set of rational numbers is countably infinite.*
5. *The power set of the set of natural numbers has a greater cardinality than itself. (idea: contradiction, diagonalization)*

Cantor's Theorem



Theorem. *If a set S is of any infinite cardinality then its power set 2^S has a greater cardinality, i.e. $|2^S| > |S|$.*

(hint: happy, sad sets).

Corollary. *There is an infinite series of infinite cardinals.*

"Most admirable flower of mathematical intellect" --Hilbert

Existence of Problems with No Program

- An alphabet $\Sigma = \{a, b, c\}$ is a finite set of letters,
- A language is a set of strings over some alphabet
- Σ^* is the set of all strings over Σ, e.g. $aabbaa \in \Sigma^*$,
- A language L over Σ is then a subset of Σ^*, e.g.,
  - $L_{even} = \{w \in \Sigma^* \ : \ w \ is \ of \ even \ length\}$
  - $L_{\{a^n b^n\}} = \{w \in \Sigma^* \ : w \ is \ of \ the \ form \ a^n b^n \ for \ n \geq 0\}$
- We say that a language L is decidable if there exists a program P_L such that for every member of L program P_L returns "true", and for every non-member it returns "false".

Existence of Problems with No Program

Theorem. There exists some undecidable formal languages.

Proof.

1. The number of programs are countably infinite, why?
2. Consider the set of languages over alphabet $\{0, 1\}$.
3. Notice that the set of all strings over $\{0, 1\}$is countably infinite.
4. Set of all languages over $\{0, 1\}$ is the power-set of the set of all strings
5. From *Cantor's theorem*, it must be the case that for some languages there is no recognizing program.