Introduction to the Theory of Computation

Set 1

Course Goals

Explore the capabilities and limitations of computers

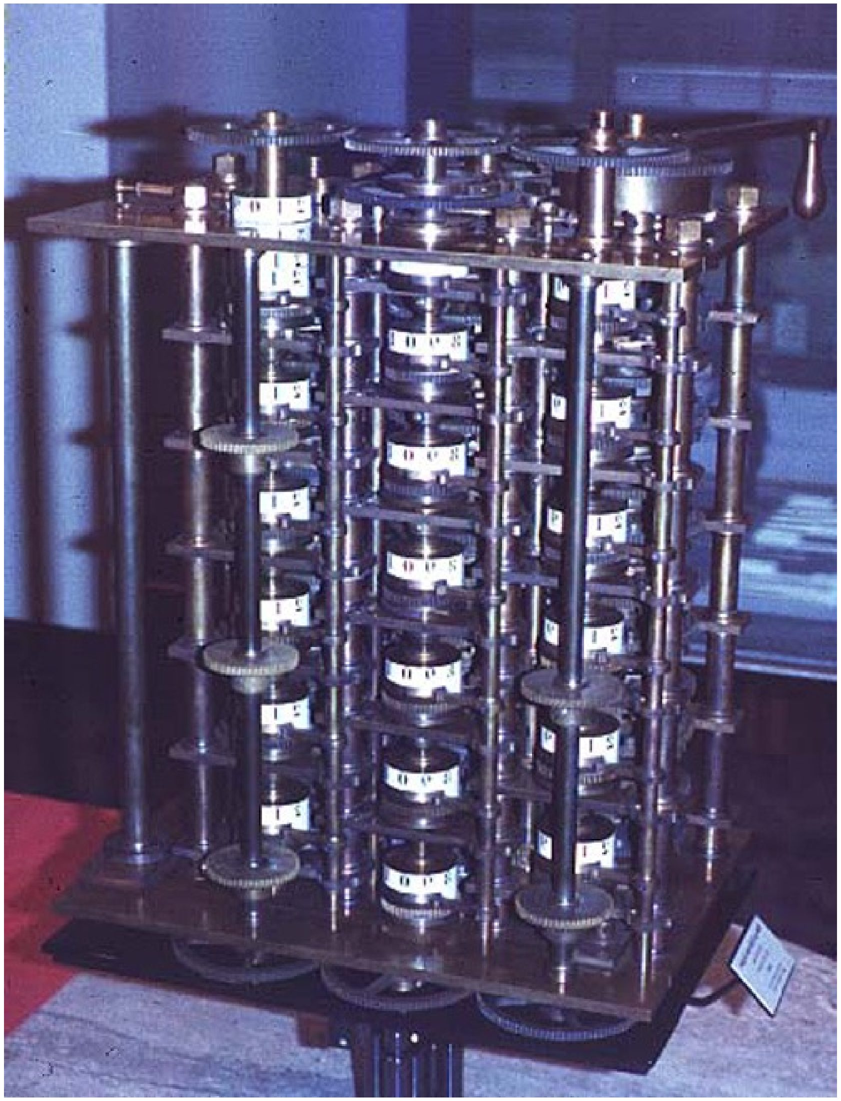
- Automata theory
  - How can we mathematically model computation?
- Computability theory
  - What problems can be solved by a computer?
- Complexity theory
  - What makes some problems computationally hard and others easy?

Introduction to the Theory of Computation

History of Computation

Devices to Aid Computation

- Abacus
  - aids memory
- Napier's Bones
  - dynamic logarithm
- Slide Rule
- Pascaline
- Jacquard Loom
- Difference Engine

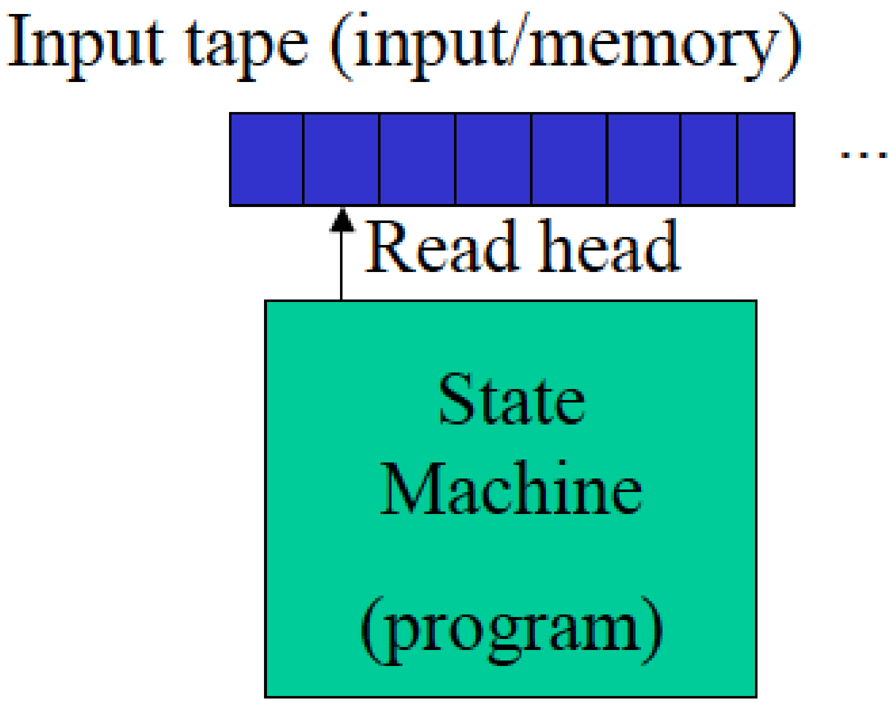


Devices to Aid Computation

- Abacus
  - aids memory
- Napier's Bones
  - dynamic loga
- Slide Rule
- Pascaline
- Jacquard Loom
- Difference Engine
- Hollerith Desk

Automated Computation

- Analytic Engine (1820s)
- Turing Machine (1936)



- Tape that holds character string
- Tape head that reads and writes character
- Machine that changes state based on what is read in

Automated Computation

- Analytic Engine (1820s)
- Turing Machine (1936)

  "Can there exist, at least in principle, a definite method by which all mathematical problems can be decided"

- Z1 Computer (1938)
- ENIAC 1 (1946)
- UNIVAC (1951)
- IBM 701 (1953)
- IBM 704 (1954)

Computation
Basic Questions in Computer Science

What problems can and cannot be computed?

- *Computability*

If a problem can be solved, how long will it take?

- *Complexity*

Approach:

– **Develop a formal model for a "computer"**

– **"Run" the problem using the model to determine computability and efficiency**

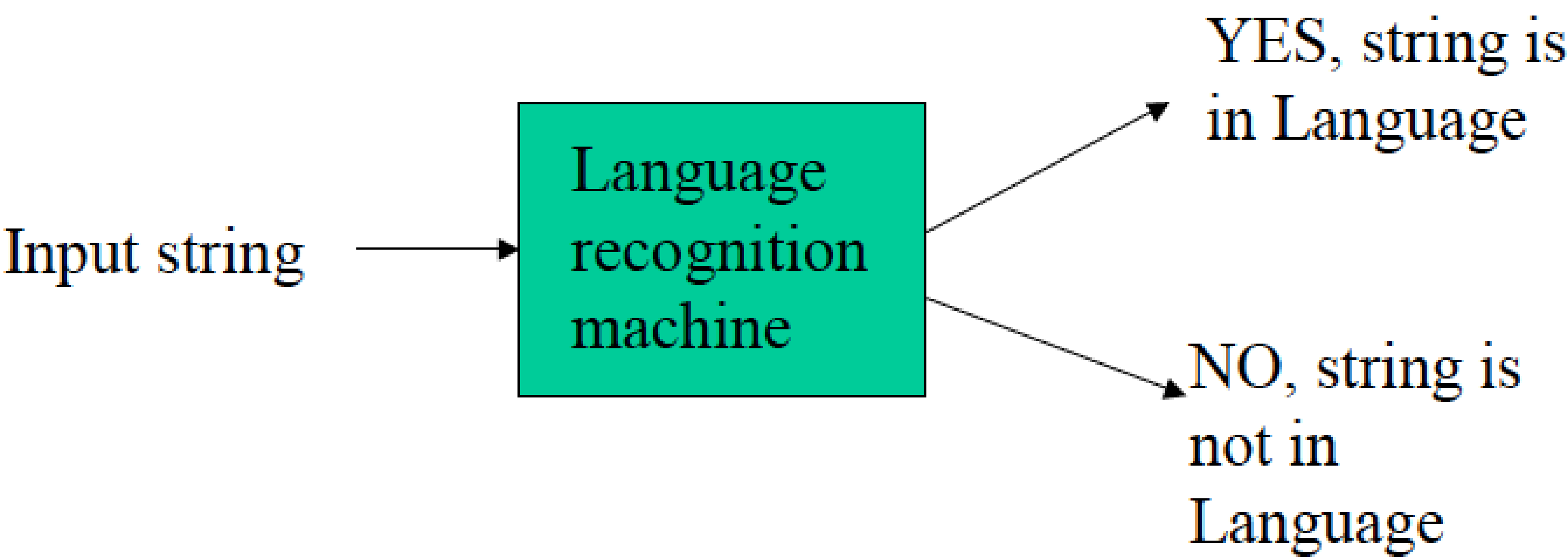
Introduction to the Theory of Computation

The theory can be described using mathematics.

We will start by describing simpler machines that answer simpler problems… such as the *string recognition problem*.

String Recognition Problem

Given a string and a definition of a language as a set of strings, is the string a member of the language?



Formal Language Study

Three Elements:

- The language itself (set of strings)
- Mechanism for defining/generating language
- Mathematically-formal machine used to test if a string is in the language

Formal Languages

- **Alphabet**
  - **Finite collection of objects (denoted Σ)**
- **String**
  - **Concatenation of 0 or more elements of an alphabet**
- **Language**
  - **Collection of strings**

Σ^* is the set of all strings over Σ (including ε)

$\varepsilon \triangleq$ *the empty string*

`ε.length()==0`

Alphabets, Strings, Languages

- **Alphabet: any finite set** (elements called *symbols*)

  $\Sigma_1 = \{1,2,3\}$

  $\Sigma_2 = \{\alpha,\beta,\gamma\}$

- **String: a sequence of symbols from a given alphabet**

  1212123

  $\alpha\beta\beta\beta\alpha\beta$

  - **Empty string ε contains no symbols of the alphabet**

- **Language: a set of strings**

  $A = \{1,3,13,233,323\}$

  $B = \{\varepsilon,\beta\beta,\beta\gamma\gamma\}$

Languages

We will look at several classes of languages:

- Each class will have its own means for language generation
- Each class will have its own machine model for string recognition
- We will progress from simpler to more complex languages and machines

Theory of Computation

- **Languages**
  - Parsers
  - Compilers
  - Regular Expressions
  - Programming Languages
  - ...
- **Computation**
  - Computability
  - Complexity

Introduction to the Theory of Computation

Review of Prerequisite Concepts

Set 1a

Sets, Multisets and Sequences

- **Set**
  - Order and repetition don’t matter
    - {7,4,7,3} = {3,4,7}
- **Multiset**
  - Order doesn’t matter, repetition does
    - {7,4,7,3} = {3,4,7,7} ≠ {3,4,7}
- **Sequence**
  - Order and repetition matter
    - (7,4,7,3) ≠ (3,4,7,7)
    - Finite sequence of k elements may be called a k-tuple

Examples

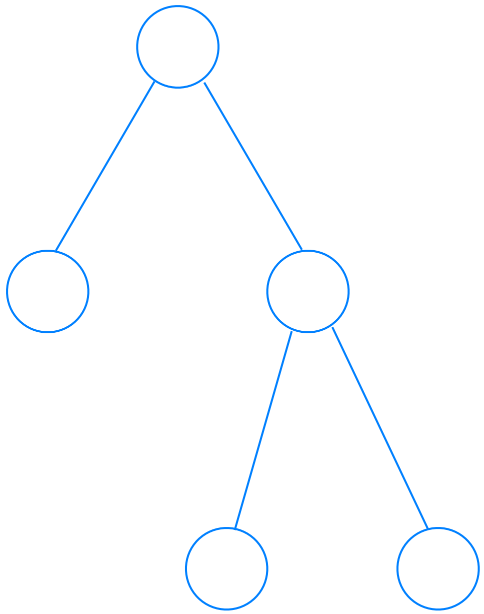
$A=\{1,2\}$, $B=\{2,3\}$, $\Sigma=\{x \in \mathbb{N} \mid x < 6\}$

- $A \cup B = \{1,2,3\}$
- $A \cap B = \{2\}$
- $\bar{A} = \{3,4,5\}$
- $A \times B = \{(1,2), (1,3), (2,2), (2,3)\}$
- $\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$

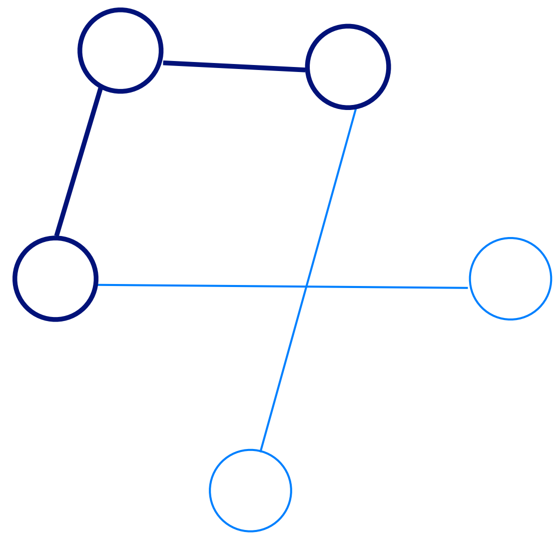
- **Union: A∪B**
- **Intersection: A∩B**
- **Complement: Ā**
- **Cartesian Product: A×B**
  - **Also called cross product**
- **Power set: $\mathcal{P}$(A)**

Σ = alphabet

Graphs

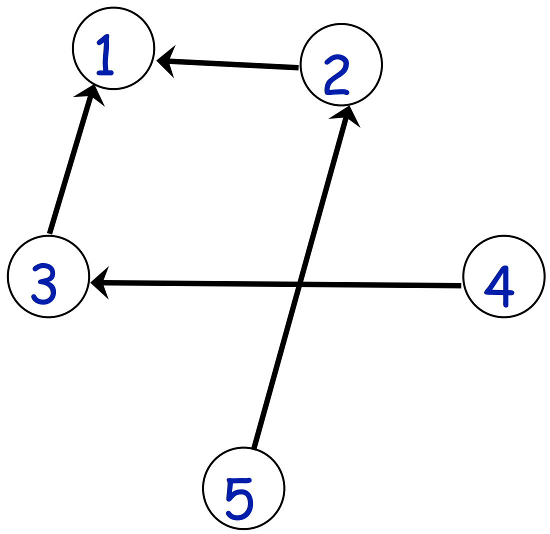


Binary tree



Subgraph

Directed Graphs



{(2,1),(3,1),(4,3),(5,2)}

Function

Mechanism associating each input value with exactly one output value

- Domain: set of all possible input values
- Range: set containing all possible output values

$$f : D \to R$$

n	f(n)
1	2
2	4
3	2
4	4

$$f : \{1, 2, 3, 4\} \to \{2, 4\}$$

$$f : \{1, 2, 3, 4\} \to \{1, 2, 3, 4\}$$

Relation

- **Predicate:** **function whose output value is always either** TRUE **or** FALSE
- **Relation:** **predicate whose domain is the set A×A×…×A**
  - **If domain is all *k*-tuples of A, the relation is a *k-ary relation on A***

Relation on A

- Function $R: A \times A \times \ldots \times A \rightarrow \{\text{TRUE, FALSE}\}$
  - Often described in terms of the set of elements for which the relation is TRUE
- Example
  - $A=\{1,2,3,4,5\}$
  - $R: A \times A \times A \rightarrow \{\text{TRUE, FALSE}\}$
    - R is TRUE if the three-tuple is increasing
  - $\{(1,2,3),(1,2,4),(2,3,4),(3,4,5)\} \subset R$
  - $(1,1,5) \notin R$

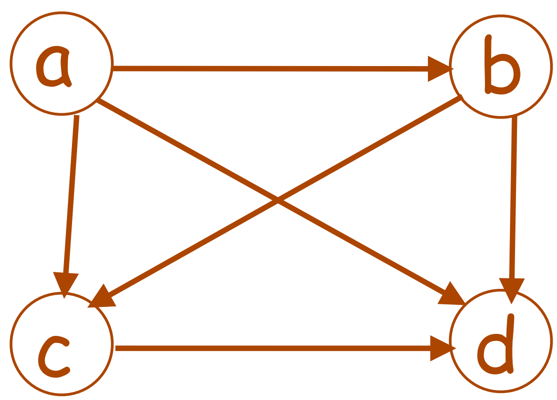
Graphical Representation
(Binary Relations Only)

Directed graph with edge (a,b) if (a,b)∈R

Example:

A={a,b,c,d}, R="earlier in alphabet"

R={(a,b),(a,c),(a,d),(b,c),(b,d),(c,d)}



Equivalence Relation

- **Reflexive**
  - $\{(a,a) \mid a \in A\} \subseteq R$
- **Symmetric**
  - $(a,b) \in R \Rightarrow (b,a) \in R$
- **Transitive**
  - $(a,b) \in R \wedge (b,c) \in R \Rightarrow (a,c) \in R$
- **Examples**
  - Equality
  - "Has the same eye color"

Boolean Logic

- Conjunction (AND) $\wedge$
- Disjunction (OR) $\vee$
- Negation (NOT) $\neg$
- Exclusive or (XOR) $\otimes$
- Equality $\leftrightarrow$
- Implication $\rightarrow$

Proof Techniques

- **Construction (Direct)**
  - Prove a “there exists” statement by finding an object that exists
- **Contradiction**
  - Assume the opposite and find a contradiction
- **Induction**
  - Show true for a base case and show that if the property holds for the value k, then it must also hold for the value k + 1