Hand-axe

Longest-used tool in human history (1.5 million years)
Wheel

Idea behind transportation revolution
(Other uses: potter’s wheel, steering wheel, flywheel)
Simple machines

- Pulley
- Lever
- Screw
Machines
Computing
Counting cattle

1  2  3  4  5
Machine for computing

- How do humans compute or calculate or solve problems?
- Is it possible to build a computing machine that can mechanically (i.e., without thinking) simulate the computations performed by a human brain like that of Galileo or Newton or Einstein?
- If so, what problems can or cannot be solved by such a computing machine?
2000 BC: Abacus

- Not automatic
- Operations: Addition, subtraction, multiplication, and division
1643: Pascal’s calculator (Pascaline)

- Inventor: Blaise Pascal
- Operations: Addition and subtraction
- World’s first mechanical calculator

Source: Computer Museum History Center
1694: Leibniz’ calculator (Step reckoner)

- Inventor: Gottfried Wilhelm Leibniz
- Operations: Addition, subtraction, multiplication, and division
1820: Colmar's calculator (Arithmometer)

- Inventor: Thomas de Colmar
- Operations: Addition, subtraction, multiplication, division, square root, involution, resolution of triangles, etc
- Applications: Financial organizations

\[ \sqrt{35} \rightarrow 5.9161 \]
1822: Babbage’s calculator (Difference engine)

- Designer: Charles Babbage
- The system was never built due to conflicts and insufficient funding
- Operations: Addition, subtraction, multiplication, division, logarithmic, trigonometric functions, etc

Source: Science Museum London
1833: Babbage’s computer (Analytical engine)

- Designer: Charles Babbage
- The system was never built due to conflicts and insufficient funding
- World’s first general-purpose computer (Turing-complete)
- Components: arithmetic logic unit, control flow in the form of conditional branching and loops, and integrated memory

Source: Science Museum London
1843: Lovelace’s algorithm

- Designer: Ada Lovelace
- World’s first programmer
- Published the first algorithm to be implemented on a computer
- The algorithm was used to compute Bernoulli numbers
1931: Gödel’s proof

- Discoverer: Kurt Gödel
- Some mathematical truths cannot be proved
1931: Gödel’s proof

- Discoverer: Kurt Gödel
- Some mathematical truths cannot be proved
  (If you cannot prove a mathematical statement, then how do you know that the statement is true?)
1936: Turing machine

- Discoverer: Alan Mathison Turing
- Creator of computer science
- Turing machine – the simplest, the most intuitive, the most generic, and the most powerful mathematical model of a computing human brain and a computer
- Algorithm and computation
1936: Turing’s proof

- Discoverer: Alan Mathison Turing
- Some computational problems cannot have algorithms
1936: Turing’s proof

- Discoverer: Alan Mathison Turing
- Some computational problems cannot have algorithms
  (If you cannot mechanically compute a computational problem, then why is it called a computational problem?)
1941: Zuse’s Z3

- Designer: Konrad Zuse
- World’s first working programmable, fully automatic digital computer (Turing-complete)

Source: http://www.horst-zuse.homepage.t-online.de/
1943: McCulloch and Pitts’ finite automata

- Designers: Warren McCulloch and Walter Pitts
- Finite automata – simple model of computation
1945: Mauchly and Eckert’s ENIAC

- Designers: John Mauchly, J. Presper Eckert
- World’s first electronic general-purpose computer (Turing-complete)
1957: Chomsky’s grammars

- Designer: Noam Chomsky
- Context-free grammar and context-sensitive grammar – models of computation
1985: Deutsch’s quantum machine

- Discoverer: David Deutsch
- **Quantum model of computation**
- Model based on **quantum physics** and not classical physics
- **Exponentially faster** than classical computing for some problems

Source: twitter
1989: Lee’s world wide web

- **Designer:** Tim Berners Lee
- **World wide web** – led to Internet revolution
What is a computer/computation/algorithm?
What is a computer/computation/algorithm?

Input info → Diagram of laptop → Output info
What is an alphabet?

**Definition**

- An **alphabet**, denoted by \( \Sigma \), is a finite, non-empty set of symbols.

**Examples**

- \( \Sigma = \{a, b\} \)
- Unary alphabet \( \Sigma = \{1\} \)
- Binary alphabet \( \Sigma = \{0, 1\} \)
- English alphabet \( \Sigma = \{a, \ldots, z, A, \ldots, Z\} \)
- Alphanumeric alphabet \( \Sigma = \{a-z, A-Z, 0-9\} \)
- Morse code alphabet \( \Sigma = \{\text{dot, dash, pause}\} \)
- DNA alphabet \( \Sigma = \{A, C, G, T\} \)
- Java programming language alphabet
  \( \Sigma = \{a-z, A-Z, 0-9, (, ), {, }, \ldots, ;\} \)
- \( \{1, 2, 3, \ldots\} \) is not an alphabet as the set is not finite
Powers of an alphabet

**Definition**

- $\Sigma = $ Some alphabet
- $\Sigma^k = $ Set of all strings of length $k$ over $\Sigma$
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots = $ Set of all strings over $\Sigma$
  - $\Sigma^*$ is the universal set of all strings
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots = $ Set of nonempty strings over $\Sigma$

**Examples**

- Let $\Sigma = \{a, b\}$
- $\Sigma^0 = \{\epsilon\}$
  - $\Sigma^1 = \{a, b\}$
  - $\Sigma^2 = \{aa, ab, ba, bb\}$
- $\Sigma^* = \{\epsilon, a, b, aa, ab, ba, bb, \ldots\}$
  - This ordering is called **canonical ordering**, which is different from lexicographic ordering
- $\Sigma^+ = \{a, b, aa, ab, ba, bb, \ldots\}$
**What is a string?**

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>• A <strong>string</strong> or word is a finite sequence of symbols chosen from $\Sigma$. A string $x \in \Sigma^*$. An empty string is denoted by $\epsilon$.</td>
</tr>
<tr>
<td>• $</td>
</tr>
<tr>
<td>• $n_\sigma(x)$ = #occurrences of symbol $\sigma \in \Sigma$ in the string $x$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>• $x = abaaabb$ from $\Sigma = {a, b}$</td>
</tr>
<tr>
<td>• $x = 111$ from $\Sigma = {0, 1}$</td>
</tr>
<tr>
<td>• $x = \epsilon$ from $\Sigma = {a, \ldots, z, A, \ldots, Z}$</td>
</tr>
<tr>
<td>• $x = Bond007$ from $\Sigma = {a - z, A - Z, 0 - 9}$</td>
</tr>
<tr>
<td>• $x = CGGTCCGC$ from $\Sigma = {A, C, G, T}$</td>
</tr>
<tr>
<td>• $x = a$ simple hello world C program from $\Sigma = {if, main, return, for, (,), {}, \ldots, ;}$</td>
</tr>
</tbody>
</table>
What is a language?

**Definition**
- A language over $\Sigma$ is a subset of $\Sigma^*$.

**Examples**
- The empty language $\emptyset$.
- $\{\epsilon, a, aab\}$ - a finite language.
- Language of palindromes over $\{a, b\}$
- $\{x \in \{a, b\}^* \mid n_a(x) > n_b(x)\}$.
- $\{x \in \{a, b\}^* \mid |x| \geq 2$ and $x$ begins and ends with $b\}$
What is a language?

Examples (continued)

- Language of your favorite quotations
- Language of legal Java identifiers
- Language of legal algebraic expressions involving the identifier $a$, the binary operations $+$ and $\ast$, and parentheses
  (strings: $a, a + a \ast a$, and $(a + a \ast (a + a))$)
- Language of balanced strings of parentheses.
  (strings: $\epsilon, ()(())$, and $((((())))$))
- Language of numeric “literals” in Java (e.g: $-41, 0.03, 5.0E-3$).
- Language of legal Java programs.
- Language of theorems (true statements) in arithmetic
- Language of theorems (true statements) in geometry
How can we represent information?

Strings can be used to represent all types of information. Strings can encode information about names, numbers, dates, text documents, images, videos, and literally any type of data. Binary strings are the simplest type of strings that can encode any information. Binary strings can also be viewed as numbers. Hence, numbers can also be used to represent all types of information.
<table>
<thead>
<tr>
<th>Concept</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model of computation</td>
<td>An abstract but physically realistic machine that does computation</td>
</tr>
<tr>
<td>Language</td>
<td>Set of all strings that the computational model accepts</td>
</tr>
<tr>
<td>Grammar</td>
<td>Set of rules to derive any string from the language</td>
</tr>
</tbody>
</table>
## Core idea of Theory of Computation

<table>
<thead>
<tr>
<th>Computation model</th>
<th>Language</th>
<th>Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finite automaton</td>
<td>Regular language</td>
<td>Regular grammar</td>
</tr>
<tr>
<td>Pushdown automaton</td>
<td>Context-free language</td>
<td>Context-free grammar</td>
</tr>
<tr>
<td>Linear-bounded automaton</td>
<td>Context-sensitive language</td>
<td>Context-sensitive grammar</td>
</tr>
<tr>
<td>Turing machine</td>
<td>Recursively enumerable language</td>
<td>Unrestricted grammar</td>
</tr>
<tr>
<td>No computer or no algorithm</td>
<td>Undecidable language</td>
<td>?</td>
</tr>
</tbody>
</table>

- We will spend an entire semester for this course trying to understand this table.
### Three major topics of Theory of Computation

<table>
<thead>
<tr>
<th>Covered topic</th>
<th>Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automata theory</td>
<td>What can be computed with extremely limited space?</td>
</tr>
<tr>
<td>Computability theory</td>
<td>What can be computed? Can a computer solve all computational problems, given enough (finite) time and space?</td>
</tr>
<tr>
<td>Complexity theory</td>
<td>How fast can we solve a problem? How small space can we use to solve a problem?</td>
</tr>
<tr>
<td>Not covered topic</td>
<td>Questions</td>
</tr>
<tr>
<td>Algorithms</td>
<td>How can a given computational problem be solved efficiently (less time and space)?</td>
</tr>
</tbody>
</table>
## What can be computed?

<table>
<thead>
<tr>
<th>Problem</th>
<th>DFA</th>
<th>PDA</th>
<th>TM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw money from ATM</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Check if a string is present in another string</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Linux regular expressions</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Parse if-else blocks and for loops in C/C++/Java programs</td>
<td>X</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Parse nested arithmetic expressions</td>
<td>X</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Parse markup languages such as HTML</td>
<td>X</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Multiply two integers</td>
<td>X</td>
<td>X</td>
<td>✔</td>
</tr>
<tr>
<td>Factorize an integer into two integers</td>
<td>X</td>
<td>X</td>
<td>✔</td>
</tr>
<tr>
<td>Find a shortest path between two cities</td>
<td>X</td>
<td>X</td>
<td>✔</td>
</tr>
<tr>
<td>Check if a computer program halts or terminates</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Check if a computer program crashes</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Check if a computer program is correct</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

- DFA: Deterministic Finite Automaton
- PDA: Pushdown Automaton
- TM: Turing Machine
<table>
<thead>
<tr>
<th>Topic</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finite automaton</td>
<td>Regular expressions, Traffic signals, Vending machines, ATMs, String matching, Lexical analysis in a compiler, Combination/sequential digital logic circuits, Spell checkers</td>
</tr>
<tr>
<td>Pushdown automaton</td>
<td>Stack applications, Balanced parentheses, Syntax analysis in a compiler, Evaluating arithmetic expressions</td>
</tr>
<tr>
<td>Linear-bounded automaton</td>
<td>Variable declaration and definition in a compiler, Genetic programming</td>
</tr>
<tr>
<td>Turing machine</td>
<td>Understanding computation, Mother of classical computers and algorithms, Grandmother of quantum computers</td>
</tr>
<tr>
<td>Complexity theory</td>
<td>Cryptography</td>
</tr>
</tbody>
</table>
## Turing-complete systems

<table>
<thead>
<tr>
<th>Time</th>
<th>Turing-complete system</th>
<th>Designer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1830s</td>
<td>Analytical engine</td>
<td>Charles Babbage</td>
</tr>
<tr>
<td>1930s</td>
<td>Recursive functions</td>
<td>Stephen Kleene</td>
</tr>
<tr>
<td></td>
<td>λ-calculus</td>
<td>Alonzo Church</td>
</tr>
<tr>
<td></td>
<td>Turing machine</td>
<td>Alan Turing</td>
</tr>
<tr>
<td></td>
<td>Unrestricted grammar</td>
<td>—</td>
</tr>
<tr>
<td>1940s</td>
<td>Z3</td>
<td>Konrad Zuse</td>
</tr>
<tr>
<td></td>
<td>Tag systems</td>
<td>Emil Leon Post</td>
</tr>
<tr>
<td>1960s</td>
<td>Markov’s algorithms</td>
<td>Andrey Markov, Jr.</td>
</tr>
<tr>
<td></td>
<td>Unlimited register machines</td>
<td>John Shepherdson, Howard Sturgis</td>
</tr>
<tr>
<td>1970s</td>
<td>C</td>
<td>Dennis Ritchie</td>
</tr>
<tr>
<td></td>
<td>Game of life</td>
<td>John Conway</td>
</tr>
<tr>
<td>1980s</td>
<td>Rule 110</td>
<td>Stephen Wolfram</td>
</tr>
<tr>
<td></td>
<td>Quantum computers</td>
<td>David Deutsch</td>
</tr>
</tbody>
</table>
### Problems

- **[Halting program]**
  Write a computer program that takes a computer program $P$ as input and outputs whether $P$ halts (i.e., terminates) or not.

- **[Correctness program]**
  Write a computer program that takes a computer program $P$ and a specification $s$ for $P$ as input and outputs whether $P$ is correct or not (i.e., if $P$ follows the input-output specification $s$ or not).

- **[Equivalence program]**
  Write a computer program that takes two computer programs $P_1$ and $P_2$ as input and outputs whether $P_1$ is functionally equivalent to $P_2$ or not.

- **[Self-replicating program]**
  Write a computer program that does not take any input and outputs its own source code.
What can be computed?

<table>
<thead>
<tr>
<th>Problems</th>
</tr>
</thead>
</table>
| • [Halting program]  ▶ Impossible  
  Write a computer program that takes a computer program $P$ as input and outputs whether $P$ halts (i.e., terminates) or not. |
| • [Correctness program]  ▶ Impossible  
  Write a computer program that takes a computer program $P$ and a specification $s$ for $P$ as input and outputs whether $P$ is correct or not (i.e., if $P$ follows the input-output specification $s$ or not). |
| • [Equivalence program]  ▶ Impossible  
  Write a computer program that takes two computer programs $P_1$ and $P_2$ as input and outputs whether $P_1$ is functionally equivalent to $P_2$ or not. |
| • [Self-replicating program]  ▶ Possible  
  Write a computer program that does not take any input and outputs its own source code. |