CSC 320 - Lecture 01

#intro #classical-computing #languages #countability

What's That? (Foundations of Computer Science)

- Study **fundamental nature** of (classical) **computation**.
    - What **problems** are **solvable** using a computer?
    - **Note**. In our context, solving a problem means: answer the question posed in the problem exactly.

Note. We are are saying classical in this case because quantum computing is becoming stronger.

- We **need to know** answers to...
    - What exactly is a **problem**?
    - What exactly is a **solution to a problem**?
    - How do we **model** a **computer**?

Course Topics

- Formal definitions of computations, languages and computability.
- Models of computation:
    - Finite Automata, Pushdown Automata, Turing Machines.
    - Grammars.
    - Deterministic and Non-Deterministic Machines.
- The Halting Problem:
    - Reductions.
- Intractability:
    - NP-hardness, Completeness.
    - Polynomial-Time Reductions.
- Dealing with Intractability or Quantum Computing. (Dr. Ulrike's field of research and interest).

Learning Outcomes

- An understanding of the following...
  - What are the fundamental capabilities and limitations of computers?
  - What makes some problems computationally hard and others easy?
- Topics discussed are in the following areas...
  - Automata Theory
  - Computability Theory (Computational models from three perspectives).
  - Complexity Theory

Goals

- Are there any problems that --no matter how powerful the computer/no matter how much time you have-- your computer would not not be able to solve?
- Is it decidable (i.e., is there a computation that allows us to decide), in **Conway's Game of Life**, whether given an initial pattern and another pattern, the latter pattern can ever appear when starting from the initial one?
- Is it decidable, given a finite set of tile types, to determine whether there is an arrangement of them with the adjacent sides matching that tiles the plane? (**The Tiling Problem**).
- Is it possible to design an algorithm that is the perfect antivirus software, that is a software that decides for any current or future software whether or not the software can act like a virus?
- Is it possible to design a perfect debugger (an algorithm that that, among other bugs, catches infinite loops)?
- Is there an algorithm that schedules exams over a limited time period, such that no student is scheduled to take two or more exams at the same time? (Solvable, BUT not easy).

Note. If the answer is yes to one or more such questions, we ask whether there exists an algorithm that is tractable, that is it returns the solution in an acceptable amount of time.

There are big but unachievable goals such as universal debuggers, universal interpreters, and universal malware detectors.

Questions

- If at all, what type of problems can humans solve of the ones that are not solvable by (classical) computers?
- Can computers (eg., robots, conversational agents) act as humans?
- What problems can and what problems cannot be solved? What problems can be solved but are hard? What problems are easy?

Automata Theory: Examples of Automata

- Finite Automata: Used in text processing, compilers, hardware design, appliances, candy machines, and ...
- Pushdown automate (or context-free grammars): Used in programming languages, and artificial intelligence.
- Turing Machines: Model of our "conventional" computer (Smart Phone, Computer, and ...).

Terminology

- **Unsolvable** or **undecidable**: problems not solvable using our (standard) computing model (independent of resource limitations).
- **Easy** to solve: problems solvable in polynomial time (using our standard computing model).
- **Hard** to solve: problems solvable (in theory, using our standard computing model) but (likely) not solvable in polynomial time (using our standard computing model).

Conway's Game of Life

Conway's Game of Life is an example of a cellular automaton. Played on an infinite two-dimensional rectangular grid of cells each cell can either be alive or dead. The initial pattern is considered the first generation.

The status of each cell c changes each turn of the game (generation) depending on the statuses of c's (eight) neighbours.

The next generation evolves from applying given rules simultaneously to every cell on the game board.

You can try it out here https://playgameoflife.com.

Conways's Game of Life Rules

- If the cell is alive, then it stays alive if it has either exactly 2 or exactly 3 live neighbours.
- If the cell is dead, then it is "born" only if it has exactly 3 live neighbours.

Turing Machines, the Turing Test, and Conversational Agents

- The Turing Test developed by A. Turing (1950).
- First Conversation Agent ELIZA by J. Weizenbaum (1966).
  - IBM Watson Jeopardy, Alexa, Siri, Google Assistant, Cortana, Chatbots, and ...

Today's Theory of (Classical) Computation

Today it is based on the **Church-Turing Thesis**. (Any real-world computation can be translated into an equivalent computation involving a Turing Machine). We ignore resource constraints (time, space). (Questions of 2018).

Turing Machine: Abstract Model of Computer.

Note. Does Quantum Computing add computational power? (Questions of 2022). (There is no serious problem in theoretical computer ability between a quantum computer and a classical computer. Definitely everything you can solve in a classical computer can theoretically be solved in the quantum computer).

We assume that any problem that can be computed can be translated into a Turing Machine Computation. Then... there must be a Turing Machine Computation for every problem that we can compute.

What if there are **more problems** that we would like to compute than there are Turing Machine Computations? Then, there would be **problems** that are **not computable** by a Turing Machine. (ex., Halting Problem).

To know how many different problems or Turing Machine Computations there are, we need to know how to count sets, and how to determine sets that are uncountable.

Countable and Uncountable

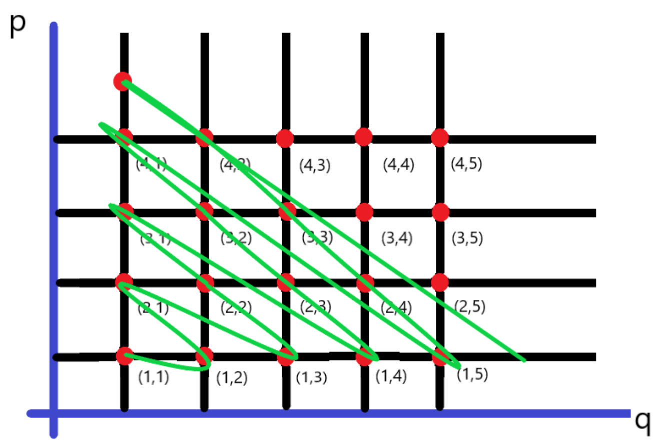
- A set is **countable** if it is **finite** or **countably infinite**: the elements of a countable set can always be counted one at a time; every element of the set is associated with a unique natural number.
  - There exists a **bijection** between any **countably infinite** set and the **set of natural number** $\mathbb{N}$.
  - There exists a **bijection** between any **finite set** and a **finite subset of** $\mathbb{N}$.
  - A set that is neither finite nor countably infinite is **uncountable**.

Counting the Set of All Integers $\mathbb{Z}$

$\mathbb{N}$	$\mathbb{Z}$
0	0
1	-1
2	1
3	-2
...	...

The idea behind this is that even numbers are mapped to positive numbers and odd numbers are mapped to negative numbers.

Counting the Set of All Integers $\mathbb{Q}^+ \setminus \{0\}$



We can enumerate all rational numbers without getting stuck in infinity. The enumeration algorithm above defines bijection with $\mathbb{N}$. Counting row by row (or alternatively column by column) would never reach all numbers - since we never leave the first row (column).

Note. No number is left behind ;)

The Set of Real $\mathbb{R}$ Numbers is Uncountable

We show that it is impossible for the set of real numbers $\mathbb{R}$ to be countable.

- **Idea**. Pick a particular infinite (sub)set $L \subseteq \mathbb{R}$ can be enumerated.
- **Show**. There exists a real number that should be in L but it not.
- **Consequence**. There is no list that enumerates **all** real numbers (since we already cannot enumerate L) and therefore we cannot enumerate $\mathbb{R}$.

Cantor's Diagonalisation Argument

Using the technique **proof by contradiction**, we assume: $\mathbb{R}$ is countable.

If $\mathbb{R}$ is countable then we can enumerate, eg, the set of all real numbers between 0 and 1:

$$\begin{aligned} x_1 &= 0.d_{11}d_{12}d_{13}d_{14}\ldots \\ x_2 &= 0.d_{21}d_{22}d_{23}d_{24}\ldots \\ x_3 &= 0.d_{31}d_{32}d_{23}d_{34}\ldots \\ x_4 &= 0.d_{41}d_{42}d_{43}d_{44}\ldots \\ \ldots \\ x_n &= 0.d_{n1}d_{n2}d_{n3}d_{n4}\ldots \end{aligned}$$

Where d is a digit.

Now consider the number $c = 0.c_1c_2c_3c_4\ldots$ with $c_i \neq d_{ii}$ for each i.

Clearly, $0 \leq c \leq 1$. Is c in above list?

So... $c = x_i$? No: $c_1 \neq d_{11}$. Thus, $c \neq x_1$, $c \neq x_2$, $c \neq x_3$, $c \neq x_4$, $c \neq x_n$. And, $c = 0.c_1c_2c_3c_4$.... So, $c_i \neq d_{ii}$.

Answer. No! $c = 0.c_1c_2c_3c_4\ldots$ is not in the list as it is different from each list element! Therefore, since $0 \leq c \leq 1$, the list **does not** contain all real numbers between 0 and 1.

Is we cannot even enumerate this subset of $\mathbb{R}$, then it is also impossible to enumerate $\mathbb{R}$.

Note. Hilberts Hotel an Ingenious Explanation of Infinity.

Terminology Review

A big part of this course is about Automata and Turing Machines, and their **Languages** as well as their computational power and limitations.

Sets

- **Set**: $\mathrm{S} = \{3, l, 20, green, \alpha\}$
    - Objects in a set: **elements/members**. Each element is unique!
- **Memberships/Non-Membership**: $\alpha \in \mathrm{S}; \beta \notin \mathrm{S}$
- **Empty Set**: $\emptyset$, or $\{\}$
    - Set with zero members/elements.
- **Singleton Set**: Set with exactly one member.
- **Unordered Par**: Set with exactly two members.

Set Operations

- **Union** of sets A and B: $A \cup B = \{x | x \in A \text{ or } x \in B\}$
- **Intersection** of sets A and B: $A \cap B = \{x | x \in A \text{ and } x \in B\}$
- **Complement** of set A: $\bar{A} = \{x | x \notin A\}$ (or $A \subseteq U = \delta \in U | x \notin A\}$)
- **Set Difference** of sets A and B: $A - B = \{x \in A : x \notin B\}$

Powerset

- **Powerset** $\mathcal{P}(A)$ of set A is the set of all subsets of A. $\mathcal{P}(A) = \{S | S \subseteq A\}$. We remember that $\emptyset \in \mathcal{P}(A)$.

Alphabets, Languages, Strings, Symbols

Terminology to describe and work with finite automata…

- An **alphabet** Σ is a **finite** set of **symbols** (eg., Binary alphabet {0, 1} or the Roman/Latin alphabet). (An alphabet will never be infinite).
- A **string** over an alphabet Σ is a finite sequence of symbols from Σ (ex., 001 is a string over alphabet {0, 1} = Σ). (A string is always finite).
- The **empty string** ε is the string with no symbols (we note that it is a string not an empty set)!
- The **set of all strings** over an alphabet Σ in denoted Σ^* (remember that $\varepsilon \in \Sigma^*$ and Σ^* is infinite).

Example

Let $\Sigma = \{a, b\}$. Then $\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, bab, bbb, \dots\}$ (i.e., infinite size).

- The **length** $|w|$ of a string $w \in \Sigma^*$ is the number of symbols of w when considered as sequence (eg., length of the empty string $\varepsilon : |\varepsilon| = 0$ and $w = ab : |w| = 2$).
- For a string $w \in \Sigma^*$, the symbol in the i^{th} position of w is denoted w_i. We say that w_i **occurs** in position i of w (note that a symbol may occur more than once in the same string (eg., $w = aba$; w_2 = b)).

Next Lecture

Lecture02