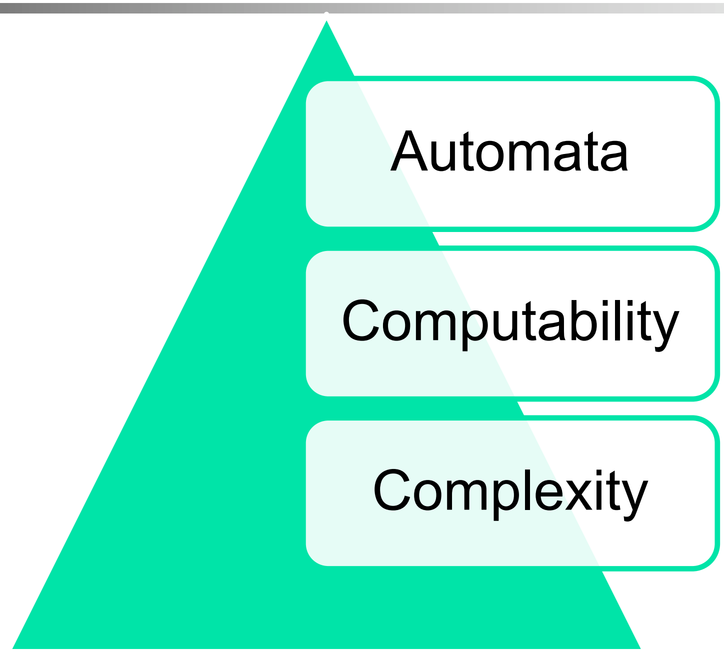
Getting started with the theory of computation

An overview + review of some math concepts we will need later

Three central areas of the theory of computation

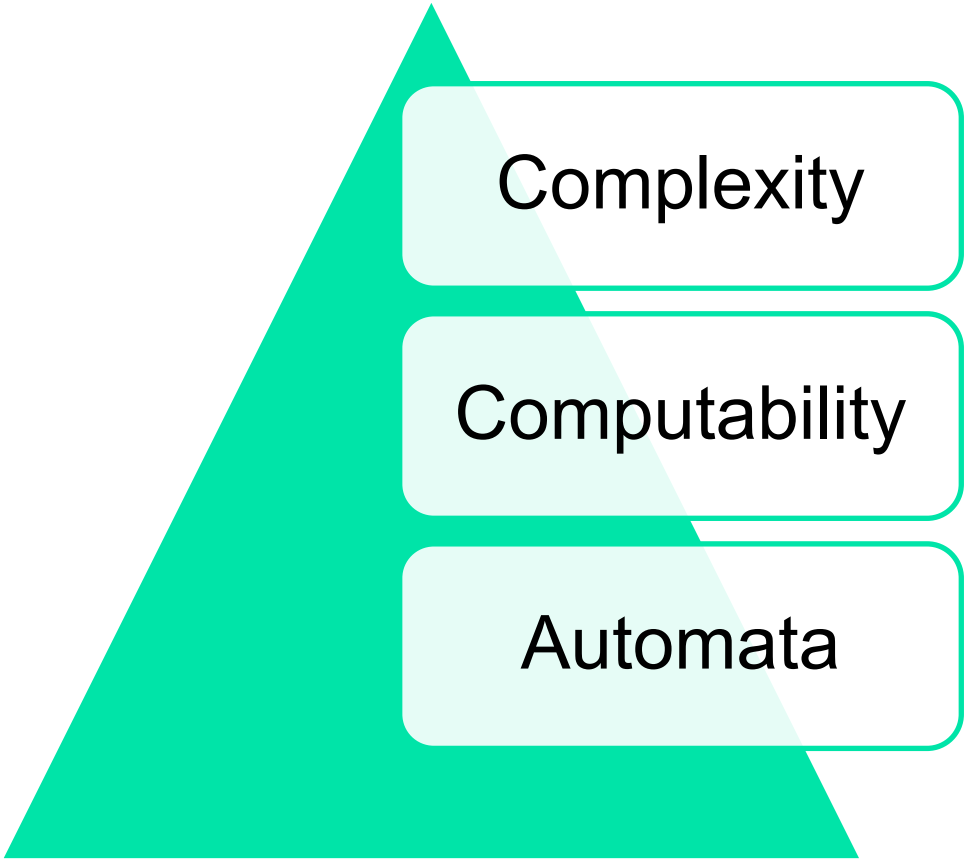


- Linked by this question:

What are the fundamental capabilities and limitations of computers?

- In each of the three areas the question is interpreted differently, and the answers vary according to the interpretation.

We will look at the three areas in reverse order, starting from the end

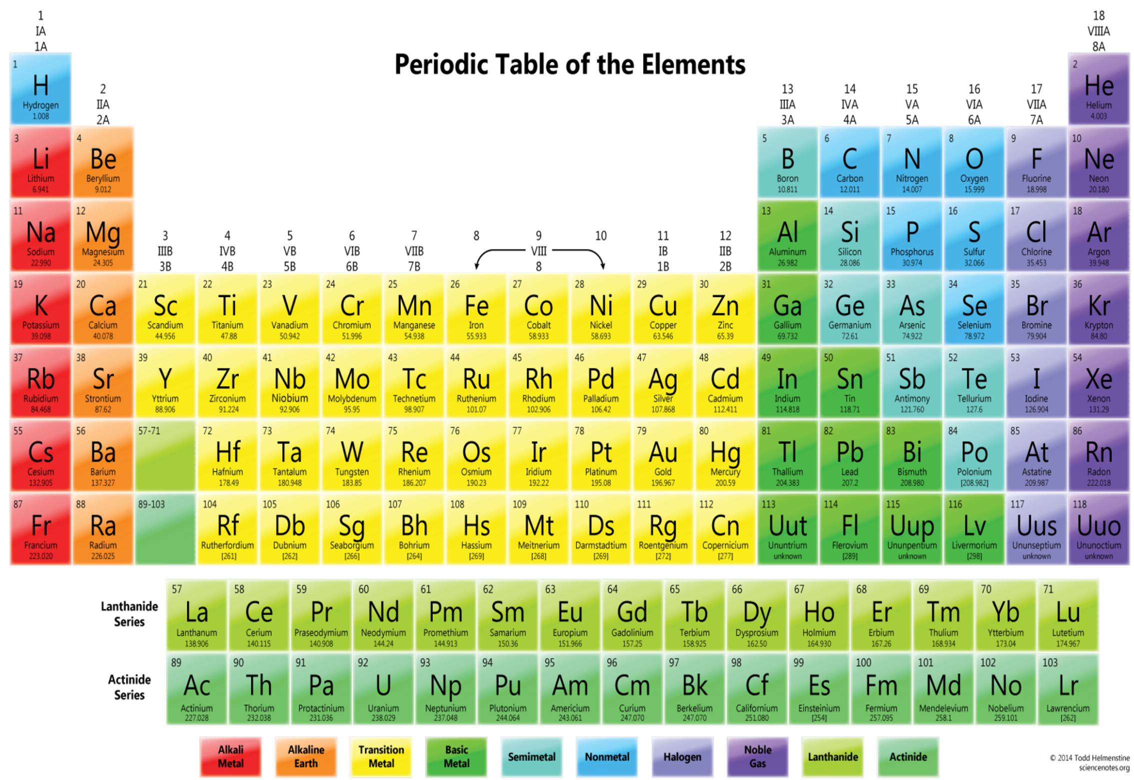


Complexity theory

- Computer problems are not created equal; some are easy, others are hard
- Example:
  - Sorting (easy)
  - Scheduling (much harder)
- Central question of complexity theory:
  - *What makes some computational problems hard and others easy?*
- *Bad news: we don't know the answer to this question yet (despite intensive research since the 1970s)*
- *Good news: researchers have come up with an elegant scheme for classifying problems according to their computational difficulty*

Complexity theory: Classification of problems

Much like the periodic table is used to classify elements according to their chemical properties, complexity theory classifies problems according to their computational difficulty.



Complexity theory: practical utility

Informing us how to deal with a problem that is computationally hard

- Identify aspect of the problem at the “root” of the difficulty, and *alter* it to make the problem easier to solve
- Settle for *less than perfect solution*
  - Approximation/heuristics
- Hardness may only be a *worst-case phenomenon*
- Consider *alternative* types of computation
  - Randomization

Use in cryptography

Computability theory

- Mathematicians such as Gödel, Turing, and Church discovered that certain basic problems cannot be solved by computers.
- One example is the problem of determining whether a mathematical statement is true or false.
- Among the consequences of this profound result was the development of ideas concerning theoretical models of computers.

Kurt Gödel
(1906--1978)

Alan Turing
(1912--1954)

Alonzo Church
(1903--1995)

Computability vs Complexity Theory

The theories of computability and complexity are closely related

Complexity theory

Objective is to classify problems as easy ones and hard ones

Computability theory

Objective is to classify problems as solvable or not solvable

Computability theory introduces several of the concepts used in complexity theory

Automata theory

- Deals with the definitions and properties of mathematical models of computation
- These models play a role in several applied areas of computer science
- One model, called finite automaton, is used in
  - Text processing
  - Compilers
  - Hardware design
- Another model, called context-free grammar, is used in
  - Programming languages
  - Artificial intelligence

Automata theory

Excellent place to begin the study of the theory of computation

The theories of computability and complexity require a precise definition of a **computer**

Allows practice with formal definitions of computation as it introduces concepts relevant to other non-theoretical areas of computer science

Mathematical notations and terminology

Math toolbox

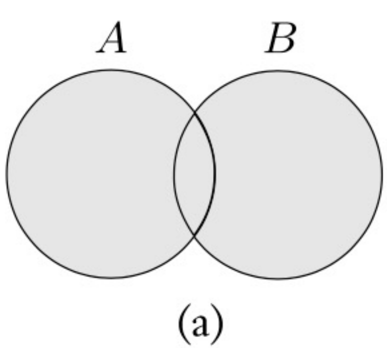
- Sets
- Sequences and Tuples
- Functions and Relations
- Graphs
- Strings and Languages
- Boolean Logic

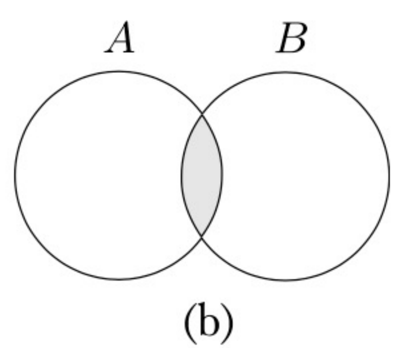
Sets

- A set is a group of objects represented as a unit
- Sets may contain any type of object, including numbers, symbols, and even other sets
- The objects in a set are called its elements or members
- For two sets A and B, we say A is a subset of B, written as $A \subseteq B$, if every member of A also is a member of B
- We say A is a proper subset of B, written $A \subsetneq B$, if A is a subset of B and not equal to B
- An infinite set contains infinitely many elements
- The set with zero members is called the empty set and is written $\emptyset$

Sets

- If we have two sets A and B, the union of A and B, written A ∪ B, is the set we get by combining all the elements in A and B into a single set
- The intersection of A and B, written A ∩ B, is the set of elements that are in both A and B
- The complement of A, written A', is the set of all elements under consideration that are not in A





Sequences and Tuples

- A sequence of objects is a list of these objects in some order
- We usually designate a sequence by writing the list within parentheses. For example, the sequence 5, 8, 12 would be written (5, 8, 12)
- The order doesn't matter in a set, but in sequence it does. Hence, (5, 8, 12) is not the same as (5, 12, 8).
- Similarly, repetition doesn't matter in a set, but in a sequence it does. Thus (5, 5, 8, 12) is different from both of the other sequences, whereas the set {5, 8, 12} is identical to the set {5, 5, 8, 12}.

Sequences and Tuples

- As with sets, sequences may be finite or infinite
- Finite sequences often are called tuples
- A sequence with k elements is a k-tuple
- Example: (5, 8, 12) is a 3-tuple
- A 2-tuple is also called an ordered pair

Sequences and Tuples

- Sets and sequences may appear as elements of other sets and sequences
- For example, the power set of of A is the set of all subsets of A
- If A is the set of {0,1}, the power set of A is the set {Ø, {0}, {1}, {0,1}}
- If A and B are two sets, the Cartesian product or cross product of A and B, written A × B, is the set of all ordered pairs wherein the first element is a member of A and the second element is a member of B

Example

EXAMPLE 0.5

If $A = \{1, 2\}$ and $B = \{x, y, z\}$,

$$A \times B = \{\, (1, x),\ (1, y),\ (1, z),\ (2, x),\ (2, y),\ (2, z)\,\}.$$

We can also take the Cartesian product of k sets, A_1, A_2, $\ldots$, A_k, written $A_1 \times A_2 \times \cdots \times A_k$. It is the set consisting of all k-tuples $(a_1, a_2, \ldots, a_k)$ where $a_i \in A_i$.

Example

EXAMPLE 0.6

If A and B are as in Example 0.5,

$$A \times B \times A = \{ (1,x,1), (1,x,2), (1,y,1), (1,y,2), (1,z,1), (1,z,2), (2,x,1), (2,x,2), (2,y,1), (2,y,2), (2,z,1), (2,z,2) \}.$$

If we have the Cartesian product of a set with itself, we use the shorthand

$$\overbrace{A \times A \times \cdots \times A}^{k} = A^k.$$

Functions and Relations

- A function is an object that sets up an input-out relationship
- A function takes an input and produces an output
- In every function, the same input always produces the same output
- If f is a function whose output value is b when the input value is a, we write $f(a) = b$
- A function also is called a mapping, and if $f(a) = b$, we say f maps a to b
- The set of possible inputs to the function is called its domain
- The outputs of a function come from a set called its range
- The notation for saying that f is a function with domain D and range R is $f: D \longrightarrow R$

Describing functions

- One way of describing functions is with a procedure for computing an output from a specified input.
- Another way is with a table that lists all possible inputs and gives the output for each each input.
- Examples (next slides)

Describing a function with a table

EXAMPLE 0.8

Consider the function $f\colon \{0,1,2,3,4\} \longrightarrow \{0,1,2,3,4\}$.

n	$f(n)$
0	1
1	2
2	3
3	4
4	0

This function adds 1 to its input and then outputs the result modulo 5. A number modulo m is the remainder after division by m. For example, the minute hand on a clock face counts modulo 60. When we do modular arithmetic, we define $\mathcal{Z}_m = \{0,1,2,\ldots,m-1\}$. With this notation, the aforementioned function f has the form $f\colon \mathcal{Z}_5 \longrightarrow \mathcal{Z}_5$.

Describing a function with a table

EXAMPLE 0.9

Sometimes a two-dimensional table is used if the domain of the function is the Cartesian product of two sets. Here is another function, $g\colon \mathcal{Z}_4 \times \mathcal{Z}_4 \longrightarrow \mathcal{Z}_4$. The entry at the row labeled i and the column labeled j in the table is the value of $g(i, j)$.

g	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

The function g is the addition function modulo 4. ■

More about functions

When the domain of a function f is $A_1 \times \cdots \times A_k$ for some sets $A_1, \ldots, A_k$, the input to f is a k-tuple $(a_1, a_2, \ldots, a_k)$ and we call the a_i the ***arguments*** to f. A function with k arguments is called a ***k-ary function***, and k is called the ***arity*** of the function. If k is 1, f has a single argument and f is called a ***unary function***. If k is 2, f is a ***binary function***. Certain familiar binary functions are written in a special ***infix notation***, with the symbol for the function placed between its two arguments, rather than in ***prefix notation***, with the symbol preceding. For example, the addition function add usually is written in infix notation with the $+$ symbol between its two arguments as in $a + b$ instead of in prefix notation $add(a, b)$.