Formal Languages and Automata

Lets start with an easy question:

Q: What is theory of computation?

- Comprises fundamental mathematical properties of computer hardware / software
- What can and cannot be computed? How quickly? How much memory? And on what type of computational model?

Q: Why should I care about theory of computation?

- Designing a new programming language or compiler?

In this course you will learn about grammars

- Dealing with string searching and pattern matching?

  In this course you will learn about Finite automata and regular expressions

- Confronted with a problem that seems to require more time that you can afford?

  In this course you will learn about NP-Completness

- Things that we have to talk about before seeing any theory:

- Evaluation method?
  - P1 - 40% date?
  - P2 + 40% date?
  - Class Questions - 20%

- Bibliography?

  Sipser, Introduction to the theory of computation 2012

Chapter 0 - Introduction

Lets start by presenting some of the main concepts of the area

0.1 Automata, Computability and Complexity

The main question behind T.C. is:

> Q: What are the fundamental capabilities and limitations of computers?

This question is interpreted differently for each of these three areas: automata (3), computability (2) and complexity (1).

1 Complexity Theory

- Not all computational problems are the same:
  - Sorting: easy
  - Scheduling university classes: hard
  - Factoring numbers: hard

Q: But what makes some problems computationally hard and others easy?

- Remarkably, we do not know the answer (even after 50 years of research)

- It is possible to demonstrate a method for giving evidence that certain problems are computationally hard even if we are unable to prove that they are.

- Several possibilities for hard problems

Possibility 1: Understand the root cause of the difficulty. Is it possible to alter it?

Possibility 2: Are approximate solutions enough?

Possibility 3: Some problems are hard only on the worst cases but easy most of the time.

(2) Computability Theory

> Q: What problems can be solved by a computer?

- Not all problems can be solved by a computer

- Examples:
  - Is a mathematical statement true or false?
  - Given a sequence of instructions acting on an input will the program end? (a.k.a. halting problem)

(3) Automata Theory

- Deals with definitions and properties of mathematical models of computation

- Different models may have different capabilities

0.2 Mathematical notions and terminology

Sets

- Group of objects represented as a unit
- May contain any type of objects:
  - numbers
  - symbols
  - even other sets

- Objects in a set are called **elements** or **members**

- Example: $S = \{7, 21, 57\}$
  - $7 \in S$ (belongs)
  - $8 \notin S$ (does not belong to)

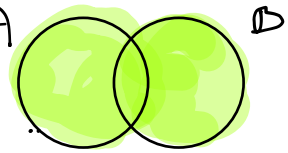
- Set A is a **subset** of B (written $A \subseteq B$) if every member of A is also a member of B

- Set A is a **proper subset** of B (written $A \subsetneq B$) if A is a subset of B and not equal to B.

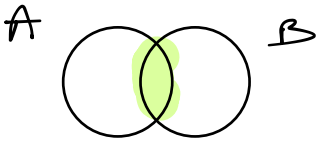
- Order and repetition do not matter

- The set with zero members is called the empty set (written $\emptyset$)

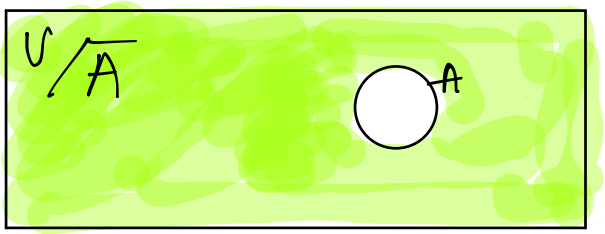
- $A \cup B$: combination of all the elements in set (union) A and B



- $A \cap B$: set of elements that are both in A and B (intersection)



- $\overline{A}$: set of all elements that are not A (complement)



Sequences and Tuples

- A sequence of objects is a list of these objects in some order: $(7, 21, 57) \neq (21, 7, 57)$

- Repetition matters in a sequence: $(7, 21, 57) \neq (7, 7, 21, 37)$

- A sequence with k elements is a k-tuple

  - A 2-tuple is called an ordered pair

- If A and B are two sets, the Cartesian product of A and B (written $A \times B$) is the set of all ordered pairs where in the first element is a member of A and the second element is a member of B

Cartesian Product Example:

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

$\{(1,x,1), (1,x,2),$
$(1,y,1), (1,y,2),$
$(1,z,1), (1,z,2)$
$(2,x,1), (2,x,2)$
$(2,y,1), (2,y,2)$
$(2,z,1), (2,z,2)\}$

Functions and Relations

- Function is an object that sets up an input-output 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 $f(a) = b$

- The set of possible inputs of a function is called its domain. The outputs of a function are its range set

- The notation for saying that f is a function with domain D and range R is:

$$f: D \to R$$

Example 1

- $f: \{0, 1, 2, 3, 4\} \to \{0, 1, 2, 3, 4\}$

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

- We could also represent this as $f: \mathbb{Z}_5 \to \mathbb{Z}_5$

Example 2

- $g: \mathbb{Z}_4 \times \mathbb{Z}_4 \to \mathbb{Z}_4$

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

- We could also represent this as $g: \mathbb{Z}_4^2 \to \mathbb{Z}_4$

Example 3

- $g: \mathbb{Z}_4 \times \mathbb{Z}_4 \to \mathbb{Z}_4 \times \mathbb{Z}_4$

g	0	1	2	3
0	(0,1)	(1,2)	(2,3)	(3,0)
1	(1,2)	(2,3)	(3,4)	(0,1)
2	(2,3)	(3,2)	(0,1)	(1,2)
3	(3,0)	(0,1)	(1,2)	(2,3)

- We could also represent this as $g: \mathbb{Z}_4^2 \to \mathbb{Z}_4^2$

Strings and Languages

- Strings of characters are fundamental building blocks in computer science

- The alphabet over which the strings are defined may vary with the application. An alphabet is any nonempty finite set.

- The members of the alphabet are the symbols / characters of the alphabet.

- Alphabet examples:

$\Sigma_1 = \{0, 1\}$

$\Sigma_2 = \{a, b, c, d, \ldots, x, y, z\}$

$\Gamma = \{0, 1, x, y, z\}$

- A string over an alphabet is a finite sequence of symbols from that alphabet:
  - String over Σ_1: 01001
  - String over Σ_2: abracadabra

- If w is a string over Σ, the length of w, written $|w|$ is the number of symbols it contains. $|w| = n \Rightarrow w = w_1 w_2 \dots w_n$

- String of length zero is called **empty string** and is written ε

- String z is a **substring** of w if z appears consecutively within w. Example: "cad" is a substring of "abracadabra"