Chapter 0: Introduction

The theory of computation focuses on three traditionally central areas:

1. Automata
2. Computability
3. Complexity

They are all linked by the questions: What is a computer and what are the fundamental capabilities and limitations of computers?

Automata theory

- Deals with the definitions and properties of mathematical models of computation

Computability theory

- Deals with classifying problems as solvable or not solvable

Complexity theory

- The central question of complexity theory: what makes some problems computationally hard and others easy?
    - Scientists do not know the answer yet
    - However, they are able to classify them according to their computational difficulty
- When confronted with a problem that appears to be computationally hard:
    1. By understanding which aspect of the problem is at the root of the difficulty, you may be able to alter it so that the problem is more easily solvable
    2. You may be able to settle for less than a perfect solution to the problem
    3. Some problems are hard only in the worst case situation
    4. You may consider alternative types of computation
- Complexity theory affects cryptography directly

Part 1: Automata and Languages

Chapter 1: Regular Languages

Some different computational models

1. Deterministic finite automata (DFA)
2. Nondeterministic finite automata (NFA)
3. Generalized nondeterministic finite automata (GNFA)
4. Markov chain == probabilistic finite automata

Regular language: a language recognized / accepted by a finite automaton

- M recognizes / accepts A
  - M == machine / automaton
  - A == language, a set of strings from a certain alphabet
- Regular operations (in order of priority)
  - Star (repetition)
  - Concatenation (often implicit, like multiplication)
  - Union
- Regular operations are used to construct a regular expression
- Regular expressions are generators for a regular language (whereby an automaton is a recognizer of such a language)
- The pumping lemma can be used to figure out if a language is regular

Deterministic finite automata (DFA)

- Has an extremely limited amount of memory (i.e., no stack or other memory devices)
- Components
  - Finite set of states
  - Finite input alphabet
  - Transition function
  - Start state
  - Accept states

Nondeterministic finite automate (NFA)

- Properties
  - A generalization of determinism
  - Can have the empty string (i.e., ε) as a transition
  - In a DFA, the transition function takes a state and an input symbol and produces the next state. In an NFA, the transition function takes a state and an input symbol or the empty string and produces the set of possible next states (think of it like forking a process)
- DFA's and NFA's recognize the same class of languages (i.e., they're equivalent in power)

Generalized nondeterministic finite automata (GNFA)

- An NFA's where the transition function can have a regular expression as a label
- Can be converted into a regular expression (since a regular language can be recognized by a DFA, NFA, and a GNFA)

Part 1: Automata and Languages

Chapter 2: Context Free Languages

Grammar

- Used to describe a language
- Can generate the language it describes (which later can be recognized by a machine)
- Generates strings through derivation / parse tree

Context free grammar (CFG)

- Were first used in the study of human languages
- Is applied in the specification and compilation of programming languages
- Generates a context free language (CFL)
  - A pumping lemma exists for context free languages
- Is comprised of a set of substitution rules, for example
  - $A \rightarrow BC$
  - $A \rightarrow a$
- Can be ambiguous if a string has multiple parse trees
- Can be in Chomsky normal form
  - Chomsky normal form == simplified CFG
- Can describe features that have recursive structure

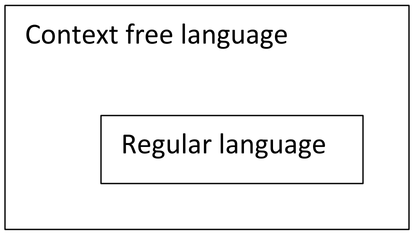
Pushdown automata

- Similar to a DFA but has a stack (LIFO stack, unlimited size)
- DPDA ≅ DCFG
  - DPDA == deterministic pushdown automata
  - DCFG == deterministic context free grammar
- NPDA == CFG
  - NPDA == nondeterministic pushdown automata
  - CFG == context free grammar (nondeterministic)
- DPDA ≠ NPDA (unlike DFA = NFA)

Generators and recognizers / automatons

- Regular expressions are equivalent to
  - Deterministic finite automata
  - Nondeterministic finite automata
  - Generalized nondeterministic finite automata
- Deterministic context free grammar is equivalent to
  - Deterministic pushdown automata
- Nondeterministic context free grammar is equivalent to
  - Nondeterministic pushdown automata

- A computation in automata == derivations in grammar



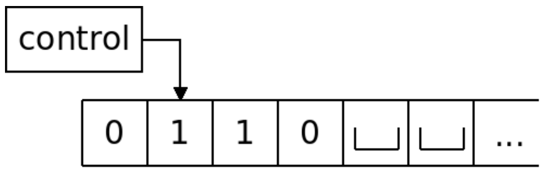
Part 2: Computability Theory

Chapter 3: The Church-Turing Thesis

Brief history of the theory of computation

1. الخوارزمي
2. David Hilbert, in the 1900's, proposed a set of problems. One of which required to be solved by an algorithm.
3. Alonzo Church developed a notational system called Lambda Calculus (λ-Calculus) to define algorithms.
4. Independently, Turing used machines to define algorithms.
5. The Church-Turing thesis provides the definition of algorithms necessary to solve Hilbert's 10th problem.
6. In 1970, the problem was finally resolved when it was discovered that no algorithm exists that can solve it.

Turing Machine

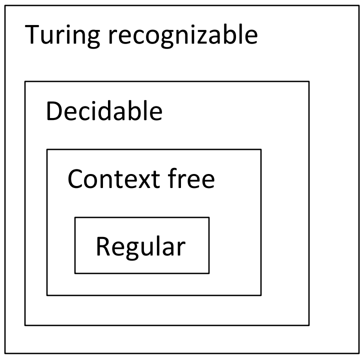


- Can do everything that a real computer can do
- Many variants exist, however they're equivalent in power (i.e., it is robust)
  - Deterministic Turing machine == multi-tape Turing machine == nondeterministic Turing machine
  - These variants (and many others) are equivalent in power
  - A machine is equivalent in power to another machine if they recognize the same languages
    - Two different programming languages (such as Pascal vs. LISP) describe exactly the same class of algorithms. For example, we can compile LISP into Pascal and Pascal into LISP
  - However, recognizable ≠ decidable ≠intractable
- Input is always a string (i.e., an object needs to be encoded as a string first)
- Has a head that can both read and write to an infinite memory tape
- States of operation
  - Accept
  - Reject
  - Loop
- They merely serve as a precise model for the definition of an algorithm

Part 2: Computability Theory

Chapter 4: Decidability

- Universal Turing Machine: capable of simulating any other Turing machine from the description of that machine
- A program can be run with itself as input
  - A compiler is a program that translates other programs
  - A compiler for Python may itself be written in Python
- If both a language and its compliment are Turing recognizable, the language is decidable
  - Decidable == solvable (i.e., a Turing machine halts with either an accept or reject state)
- The general problem of software verification is not solvable



Part 2: Computability Theory

Chapter 5: Reducibility

Reduction: a way of converting one problem to another problem in such a way that a solution to the second problem can be used to solve the first problem.

- If A is reducible to B and B is decidable, A is also decidable
- If A is undecidable and reducible to B, B is undecidable

Mapping Reducibility: a computable function that converts instances of problem A to instances of problem B. If we have such a conversion, called a reduction, we can solve A with a solver for B.

- Note: mapping reducibility is transitive

Part 2: Computability Theory

Chapter 6: Advanced Topics in Computability Theory

The Recursion Theorem

- Provides the ability to implement the self-referential ‘this’ into any programming language
- A program can obtain its own description and then go on to compute with it
- To make a Turing machine that can obtain its own description and then compute with it, we need only make a machine that receives the description of the machine as an extra input
- Example of an application: computer viruses

Mathematical Logic: the branch of mathematics that investigates mathematics itself

Th(N, +, *) is an undecidable theory. No algorithm exists for deciding the truth or falsity of mathematical statements, even when restricted to the language of (N, +, *). This theorem has great importance philosophically because it demonstrates that mathematics cannot be mechanized.

Kurt Gödel’s “incompleteness theorem”: any reasonable system of formalizing the notion of provability in number theory has some statements that are unprovable

- Some statements in (N, +, *) are not provable

Turing Reducibility: more general than mapping reducibility

Although a comprehensive definition of an algorithm exists, a similar one for information does not

- The quantity of information contained in an object is the size of that objects smallest representation or description
- The amount of information contained by a string cannot be substantially more than its length
- Incompressible strings of every length exist
- No algorithm can decide in general whether a string is incompressible

Part 3: Complexity Theory

Chapter 7: Time Complexity

We compute the running time for an algorithm purely as a function of the length of the string representing the input

Complexity analysis is done using asymptotic analysis

- Big O (i.e., asymptotically <=)
- Small o (i.e., asymptotically <)

In complexity theory, the choice of model affects the time complexity of languages

- Fortunately, time requirements don't differ greatly for typical deterministic models
- Single tape Turing machine == $O(t(n))$
- Multi tape Turing machine == $O(t^2(n))$
- Nondeterministic Turing machine == $2^{O(t(n))}$

P vs. NP

- P
  - The class of languages that are decidable in polynomial time on a deterministic single tape Turing machine
- NP
  - The class of languages that have polynomial time verifiers
  - The class of languages that can be decided by some nondeterministic Turing machine in polynomial time
  - Examples
    - Clique
    - Subset-sum
- NP-Complete: language B is NP-Complete if it satisfies two conditions
  - B is in NP
  - Every A in NP is polynomial time reducible to B
  - Examples
    - Satisfiability problem
    - Clique
    - Subset-sum
    - Hamiltonian path (directed or undirected graph)

Part 3: Complexity Theory

Chapter 8: Space Complexity

Space appears to be more powerful than time because space can be reused, whereas time cannot

PSPACE == NPSPACE

Savitch's theorem: any nondeterministic TM that uses $f(n)$ space can be converted to a deterministic TM that uses only $f^2(n)$ space

Current understanding of the different complexity classes

$$P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME$$

Part 3: Complexity Theory

Chapter 9: Intractability

Complexity classes for time and space form a hierarchy

$$Tractable \subseteq Decidable \subseteq Recognizable$$

$$P \subsetneq EXPTIME$$

Part 3: Complexity Theory

Chapter 10: Advanced Topics in Complexity Theory

Approximation algorithms

- Nearly max result needed
- Nearly min result needed

Probabilistic algorithms

- BPP: the class of languages that are decided by probabilistic (preferably not using a PRNG) time Turing machines with an error probability of 1/3
- RP: the class of languages that are decided by probabilistic (preferably not using a PRNG) time Turing machines where inputs in the language are accepted with a probability of at least ½, and inputs not in the language are rejected with a probability of 1