CS 365 Models of Computation (Advanced)

Keven Qiu
Instructor: Eric Blais
Winter 2024

Contents

Chapter 1

Introduction

1.1 Two Simplifying Restrictions

Definition: Computational Problem

A task where for each possible input to the problem, there is one or more valid outputs that is to be produced.

This however is too broad, so we impose two simplifying restrictions.

Simplifying Restriction 1

We only consider problems whose inputs are binary strings.

The binary alphabet is $\{0,1\}$ and the set of strings of length n is denoted $\{0,1\}^n$. The unique string in $\{0,1\}^0$ is the empty string, denoted ε.

We write $\{0,1\}^* = \bigcup_{n \geq 0} \{0,1\}^n$ to denote the set of all possible binary strings.

Proposition

For every finite set $\mathcal{X}$ with k elements, there is a one-to-one encoding function $h : \mathcal{X} \to \{0,1\}^{\lceil \log k \rceil}$.

Proof. Fix any ordering $a_1, \ldots, a_k$ of the elements of $\mathcal{X}$. Then define the encoding function h that maps a_i to the string that gives the binary representation of i. ■

Simplifying Restriction 2

We only consider decision problems, where there is exactly one valid output for each input, and this output is in $\{0,1\}$.

1.2 Functions and Languages

A decision problem where all inputs are binary strings of length n can be described a Boolean function

$$f : \{0,1\}^n \to \{0,1\}$$

where for each $x \in \{0,1\}^n$, the value $f(x)$ represents the valid output for input x.

We do not want to restrict to just length n binary strings. So problems can be represented by a family of Boolean functions $\{f_n\}_{n\geq 0}$.

Definition: Language

A language is $L \subseteq \{0,1\}^*$.

A language L is equivalent to the family of functions $\{f_n\}$ if for every $x \in \{0,1\}^*$ of length n,

$$x \in L \iff f_n(x) = 1$$

1.3 Cardinality of Languages

Definition: Finite Set

A set S is finite if there is a one-to-one mapping between the elements of S and the elements in the set $\{1, 2, \ldots, n\}$ for some $n \geq 0$.

Definition: Infinite Set

A set not finite.

Definition: Countable Set

A set S is countable if there is a one-to-one mapping between the elements of S and the set of natural numbers $\mathbb{N}$.

Definition: Uncountable Set

A set not countable.

The set of binary strings $\{0,1\}^n$ is finite. The set $\{0,1\}^*$ is infinite.

Proposition

The set $\{0,1\}^*$ is countable.

Proof. Consider the mapping $h : \{0,1\}^* \to \mathbb{N}$ where for each $x \in \{0,1\}^*$, we define $h(x)$ to be the natural number with binary representation $1x$, where we use $1x$ to denote string concatenation. The mapping h is one-to-one. ∎

Theorem

The set of all languages is uncountable.

Proof. This proof is an example of a diagonalization argument.

Assume for contradiction that the set of all languages is countable. Then we can list the set of languages in some order $L_1, L_2, \ldots$.

We can build a table whose columns are labelled by the strings in $\{0,1\}^*$ in lexicographical order and rows labelled by the languages $L_1, L_2, \ldots$ in the order we defined. For each cell (L_k, x) in the table, enter a 1 in the cell if $x \in L_k$ and 0 otherwise.

Consider now the language D that we obtain by look at the diagonal entries of this table and using their negation to determine if the corresponding string is in D. Namely, if x is the kth string in the lexicographical ordering of $\{0,1\}^*$, then $x \in D$ if and only if $x \notin L_k$.

D is a language so by our assumption, there is a value $n \in \mathbb{N}$ such that $D = L_n$ is the nth language in our list. Let x denote the nth string in the lexicographical order of $\{0,1\}^*$. But then $x \in D$ holds if and only if $x \notin L_n$, so $D \neq L_n$. We have arrived at our contradiction, so the set of all languages must be uncountable. ■

1.4 First Uncomputability Result

Proposition

There exist a language L for which there is no program that accepts each input $x \in \{0,1\}^*$ if and only if $x \in L$.

Proof. Assume for contradiction that for every language, there is a program that accepts exactly the set of strings in that language. Then there is a map from the set of all languages to the set of all programs. But every program can be represented as a binary string. So there is a mapping from the set of all languages to $\{0,1\}^*$. But since $\{0,1\}^*$ is countable, there is a mapping from the set of all languages to $\mathbb{N}$, contradicting the previous theorem. ■

Chapter 2

Turing Machines

We want a definition of a computer than can capture any computer, no matter how complicated. Our goal is to identify an explicit language that cannot be computed by algorithms over any machine model.

2.1 Definition

Consider an infinite tape split into squares. It has a finite number of states. Each square contains exactly one symbol. There is a tape head that points to over one of the squares. The head is allowed to move left or right and each state has a set of rules.

Definition: Deterministic 1-Tape Turing Machine

An abstract machine described by the triple

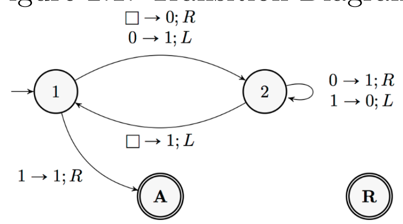
$$M = (m, k, \delta)$$

with $m, k \geq 1$ where

- $Q = \{1, 2, \ldots, m\}$ is the set of internal states,
- $\Gamma = \{\square, 0, 1, 2, \ldots, k\}$ is the tape alphabet, and
- $\delta \colon \mathbf{Q} \times \Gamma \to (\mathbf{Q} \cup \{\mathbf{A}, \mathbf{R}\}) \times \Gamma \times \{L, R\}$ is the transition function.

The state $\mathbf{1}$ is the initial state of the Turing machine M. $\mathbf{A}$ and $\mathbf{R}$ are the accept and reject states, respectively.

Figure 2.1: Transition Diagram



Definition: Configuration

A string $w\mathbf{q}y$ where

- $\mathbf{q} \in \mathbf{Q} \cup \{\mathbf{A}, \mathbf{R}\}$ represents the current state of the machine,
- $wy \in \Gamma^*$ is the current string on the tape, and
- the position of the tape head is on the first symbol of y.

Two configurations are equivalent when they are identical up to blank symbols at the beginning of w or at the end of y. In other words,

$$w\mathbf{q}y = \square w\mathbf{q}y = w\mathbf{q}y\square$$

Definition: Yields

For any strings $w, y \in \Gamma^*$, symbols $a, b, c \in \Gamma$, and states $\mathbf{q} \in \Sigma$ and $\mathbf{r} \in \Sigma \cup \{\mathbf{A}, \mathbf{R}\}$, the configuration $wa\mathbf{q}by$ of the Turing machine M yields the configuration $w\mathbf{r}acy$, denoted

$$wa\mathbf{q}by \vdash w\mathbf{r}acy$$

when $\delta(\mathbf{q}, b) = (\mathbf{r}, c, L)$. Similarly,

$$wa\mathbf{q}by \vdash wac\mathbf{r}y$$

when $\delta(\mathbf{q}, b) = (\mathbf{r}, c, R)$.

A configuration can *derive* another configuration in 0, 1, or more steps.

Definition: Accepts

A Turing machine M accepts $x \in \{0, 1\}^*$ if $\mathbf{1}x$ derives an accepting configuration $w\mathbf{A}y$.

Definition: Rejects

A Turing machine M rejects $x \in \{0, 1\}^*$ if $\mathbf{1}x$ derives a rejecting configuration $w\mathbf{R}y$.

Definition: Halts

A Turing machine M halts on x if it accepts or rejects x.

Definition: Decides

A Turing machine M decides the language $L \subseteq \{0,1\}^*$ if it accepts every $x \in L$ and rejects every $x \notin L$.

Definition: Recognize

A Turing machine M recognizes L if M accepts every $x \in L$ and M rejects or does not halt on $x \notin L$.

Definition: Decidable

A language $L \subseteq \{0,1\}^*$ if and only if there is a Turing machine that decides L.

2.2 Universal Turing Machine

Proposition

There is an encoding that maps each Turing machine M to a binary string $\langle M \rangle \in \{0,1\}^*$.

Proof. Consider the Turing machine $M = (m, k, \delta)$. We find a mapping $\{0, 1, +\}$ to the string $\langle m \rangle + \langle k \rangle + \langle \delta(1,0) \rangle + \cdots$. The positive integers m and k can be encoded by taking their binary representation. The transition function δ can be represented as a table of $m \cdot (k+2)$ entries (one for each internal state-tape symbol pair). Each of these entries can be encoded as a binary string. We can combine all these elements into a single binary representation to obtain the encoding of M. ■

Theorem

There is a Universal Turing Machine U such that for every Turing machine M and every input $x \in \{0,1\}^n$, when the input to U is the string $\langle M \rangle x$, then U simulates the execution of M on input x.

Proof. First, U turns x into the initial configuration of M by having the string $\mathbf{1}x$. Then

- Read the current state $\mathbf{q}$ and the symbol a at the tap head in M's current configuration.
- Go back to the encoding $\langle M \rangle$ of M to read the entry $\delta(\mathbf{q}, a)$ of its transition table.
- Update the configuration appropriately by overwriting the symbol a at the tap head position, moving the tape head left or right, and updating the current state of the machine.

■

2.3 Church-Turing Thesis

Church-Turing Thesis

Any decision problem that can be solved by any computer that respects the laws of physics corresponds to a language that can be decided by a Turing machine.

Proposition

Every language $L \subseteq \{0,1\}^*$ that can be computed using counter machines can also be decided by a Turing machine.

Chapter 3

Recursion Theorem

3.1 Building Blocks

Proposition

For every string $s \in \{0,1\}^*$, there exists a Turing machine P_s that on input $x \in \{0,1\}^*$ writes the string sx on the tape and then accepts.

Proof. When $s = a_1 \cdots a_n$ we can simply let P_s be the simple Turing machine that repeatedly moves left and overwrites then n blank symbols to the left of x with $a_n, \ldots, a_1$ in that order. ■

Proposition

There is a Turing machine F that on input $s \in \{0,1\}^*$, replaces s with $\langle P_s \rangle$ on the tape tape then accepts.

Proof. Given s as input, it is straightforward to determine the transition function for the corresponding Turing machine P_s as defined above. We can then write its encoding on the tape. ■

Definition: Concatenation of Turing Machines

The concatenation of Turing machines A and B is the Turing machine AB that on every input, first runs A on that input, then runs B on what is on the tape when A halts.

$\langle A \rangle \langle B \rangle \neq \langle AB \rangle$.

Proposition

There is a Turing machine C that on input $\langle A \rangle \langle B \rangle$ for any two Turing machines A and B, replaces that input with $\langle AB \rangle$ on the tape and then accepts.

Proof. The Turing machine AB has the same initial state as A. When A halts, the machine AB instead transitions to the initial state of B. When B halts, AB does and accepts if and only if B did. Therefore, the transition function for AB is easy to determine when we have the transition functions for both A and B and we can easily encode it to generate the desired output. ■

3.2 The Recursion Theorem

Theorem (Recursion Theorem)

For every Turing machine M, there exists a Turing machine Q_M that on every input $x \in \{0,1\}^*$ simulates M on the input $\langle Q_M \rangle\, x$.

Proof. Define a Turing machine R that on input $\langle N \rangle\, x$ for any Turing machine N and any binary string x replaces that input with the string

$$\left\langle P_{\langle N \rangle} N \right\rangle x$$

on the tape and halts. R exists because

1. Call F on $\langle N \rangle$ to get $\left\langle P_{\langle N \rangle} \right\rangle$.
2. Call C on $\left\langle P_{\langle N \rangle} \right\rangle \langle N \rangle$ to get $\left\langle P_{\langle N \rangle} N \right\rangle$.
3. Keep x to the right of the string.

Now, we define Q_M to be the Turing machine

$$Q_M = P_{\langle RM \rangle} RM$$

When we run Q_M on the input x, we obtain

$$x \overset{P_{\langle RM \rangle}}{\to} \langle RM \rangle\, x \overset{R}{\to} \left\langle P_{\langle RM \rangle} RM \right\rangle x = \langle Q_M \rangle\, x \overset{M}{\to} M(\langle Q_M \rangle\, x)$$

where we write $M(\langle Q_M \rangle\, x)$ to denote the output of M on input $\langle Q_M \rangle\, x$. ■

Corollary

There is a Turing machine Q that on input ε prints out $\langle Q \rangle$ on the tape and then halts.

Proof. Take M that does nothing. Q_M will now run M on $\langle Q_M \rangle\, x$. When $x = \varepsilon$, Q_M ends up writing its own description on the tape and halts. ■

3.3 Application To Undecidability

Corollary

Without loss of generality, we can always assume that a Turing machine has access to its encoding as well as its usual input x on the tape.

Proof. Design a Turing machine M that assumes the input is the form $\langle N \rangle\, x$ for any Turing machine N and is correct when $\langle N \rangle$ is its own description.

Then Q_M runs M on $\langle Q_M \rangle\, x$. ■

Theorem

The language

$$A_{TM} = \{\langle M \rangle\, x : M \text{ accepts } x\}$$

is undecidable.

Proof. Assume T decides A_{TM}. Let D be the Turing machine that

1. Obtains its own description $\langle D \rangle$ using the Recursion Theorem.
2. Run T on input $\langle D \rangle\, x$.
3. Do the opposite of T; reject if T accepts, and accept if T rejects.

By construction, D accepts x if and only if T does not accept $\langle D \rangle\, x$. This contradicts that T decides A_{TM}. ■

Chapter 4

Undecidability

4.1 More Undecidable Languages

Theorem

The language

$$\mathsf{Halt}_{TM} = \{\langle M \rangle\, x : M \text{ halts on input } x\}$$

is undecidable.

Proof. Assume on the contrary that Halt_{TM} is decidable by Turing machine T. Consider the machine M that on input x does the following:

1. Obtain its own encoding $\langle M \rangle$ using the Recursion Theorem.
2. Run T on input $\langle M \rangle\, x$.
3. If T accepts, run forever in an infinite loop; otherwise, halt and accept.

By construction, M halts on x if and only if T does not accept $\langle M \rangle\, x$. This contradicts that T decides Halt_{TM}. ■

Definition: $L(M)$

The language recognized by M.

$$L(M) = \{x \in \{0,1\}^* : M \text{ accepts } x\}$$

Theorem

The language

$$\mathsf{Empty}_{TM} = \{\langle M \rangle : L(M) = \emptyset\}$$

is undecidable.

Proof. Assume on the contrary that Empty_{TM} is decidable by Turing machine T. Consider the machine M that on input x does the following:

1. Obtain its own encoding $\langle M \rangle$ using the Recursion Theorem.
2. Run T on input $\langle M \rangle$.
3. Accept if T accepts; otherwise reject.

By this construction, $L(M) = \{0,1\}^*$ when T accepts $\langle M \rangle$ and $L(M) = \emptyset$ when T rejects. This contradicts the claim that T decides Empty_{TM}. ■

We can extend this theorem to show that it is impossible to decide any non-trivial property of languages of Turing machines.

Theorem (Rice's Theorem)

Let P be a subset of all languages over $\{0,1\}^n$ such that

1. There exists a Turing machine M_1 for which $L(M_1) \in P$, and
2. There exists a Turing machine M_2 for which $L(M_2) \notin P$.

Then the language
$$L_P = \{\langle M \rangle : L(M) \in P\}$$
is undecidable.

Proof. Assume on the contrary that L_P is decidable by Turing machine T. Consider the machine M that on input x does the following:

1. Obtain its own encoding $\langle M \rangle$ using the Recursion Theorem.
2. Run T on input $\langle M \rangle$.
3. If T accepts, simulate M_2 on x; otherwise simulate M_1 on x.

By this construction, $L(M) = L(M_2) \notin P$ when T accepts $\langle M \rangle$ and $L(M) = L(M_1) \in P$ when T rejects. This contradicts the claim that T decides L_P. ■

4.2 Reductions

Theorem

The language
$$A_{TM}^{\varepsilon} = \{\langle M \rangle : M \text{ accepts } \varepsilon\}$$
is undecidable.

Proof. Assume on the contrary that Turing machine T decides A^{ε}_{TM}.

Define a Turing machine A that takes input $\langle M \rangle\, x$. Let M' be a Turing machine that first writes x on the tape and then copies the behaviour of M. From the encoding of M and the string x, A can determine the encoding of M'. So it can call T on $\langle M' \rangle$ to determine whether M' accepts ε or not.

Since M' accepts ε if and only if M accepts x, then A decides the language A_{TM}, a contradiction. ■

Definition: (m, k)th Busy Beaver Number

Maximum number $BB_{m,k}$ of steps that a Turing machine with m states and tape alphabet $\Gamma_k = \{0, 1, \dots, k, \square\}$ can complete before halting on a tape that is initially empty.

Theorem

The language

$$B = \{\langle m \rangle \langle k \rangle \langle n \rangle : BB_{m,k} \leq n\}$$

is undecidable.

Proof. Assume on the contrary that there is a Turing machine T that decides B.

Define a Turing machine A that takes input $\langle M \rangle\, x$. As a first step, A uses the description $\langle M \rangle$ to determine the values of m and k for machine M. Then by calling T with input $\langle m \rangle \langle k \rangle \langle n \rangle$ for $n = 1, 2, \dots$ until T accepts, A can determine the value of $BB_{m,k}$. (Note that since there are finitely many distinct Turing machines with m states and tape alphabet Γ_k, the value of $BB_{m,k}$ is finite and so T will accept after a finite number of calls.)

Now A can simulate up to $BB_{m,k}$ steps of computation of M on input ε. Specifically, it can do that by copying the behaviour of the Universal Turing Machine with an additional twist: a counter that is incremented after each simulation step and that interrupts the simulation when it reaches the value $BB_{m,k}$. If M accepts or rejects during the simulation, A does the same. Otherwise, at the end of $BB_{m,k}$ steps of simulation, A halts and rejects.

A decides the language A^{ε}_{TM}. That is because if M accepts or rejects ε, then A does the same. And if M runs for more than $BB_{m,k}$ steps, then by definition of the Busy beaver numbers, it must run forever, which means that it does not accept ε. ■

4.3 Recognizability

Every language that is decidable is also recognizable. The converse statement is false.

Proposition

The undecidable language A_{TM} is recognizable.

Proof. Consider the Universal Turing Machine U. On input $\langle M \rangle\, x$, it simulates M on x and accepts if and only if M accepts x. Therefore, U recognizes A_{TM}. ■

Theorem

If a language L and its complement $\overline{L} = \{0,1\}^* \setminus L$ are both recognizable, then L and $\overline{L}$ are both decidable.

Proof. Assume that T_1 and T_2 recognize L and $\overline{L}$, respectively. Let M be a Turing machine that simulates both T_1 and T_2 in parallel. Specifically, it dedicates separate portions of the tape for the simulation of both T_1 and T_2 and interleaves their simulations by performing one step of computation of each of them at a time. The simulation is completed when either T_1 or T_2 accepts. If T_1 is the machine that accepts, M also accepts. Otherwise, if T_2 accepts, then M rejects.

When $x \in L$, then T_1 is guaranteed to accept x after a finite number of steps and T_2 is guaranteed to not accept, so M correctly accepts x. Similarly, when $x \in \overline{L}$, then T_2 is guaranteed to accept after a finite number of steps and T_1 will not accept so M correctly rejects x. Therefore, M decides L and the machine M' obtained by switching the accept and reject labels in M decides $\overline{L}$. ■

Corollary

The language $\overline{A_{TM}}$ is unrecognizable.

Proof. We have see that A_{TM} is recognizable. If $\overline{A_{TM}}$ was also recognizable, then by previous theorem, A_{TM} would be decidable, which is a contradiction. ■

Chapter 5

Time Complexity

5.1 Time Complexity Classes

Definition: Time Cost on an Input

The time cost of a Turing machine M on input $x \in \{0,1\}^*$ is the number of computational steps M performs before it halts.

Definition: Time Cost

The (worst-case) time cost of the Turing machine M is the function $t_M : \mathbb{N} \to \mathbb{N}$ where $t_M(n)$ is the maximum time cost of M on any input $x \in \{0,1\}^n$ of length n.

Definition: TIME(f)

For every function $f : \mathbb{N} \to \mathbb{N}$, the time complexity class $\textbf{TIME}(f)$ is the set of all languages that can be decided by a multi-tape Turing machine M with worst-case time cost $t_M \leq O(f)$.

Time complexity classes can also be defined in terms of other models of computation. They are often defined in terms of the minimum number of operations that multi-tape Turing machines execute before they halt.

5.2 Time Hierarchy Theorem

Theorem

$$\textbf{TIME}(n) \subsetneq \textbf{TIME}(n^3)$$

Proof. Idea is to use the fact that Turing machines can simulate any Turing machine given

Time Class	Definition
Constant time	$\mathbf{TIME}(1)$
Linear time	$\mathbf{LIN} = \mathbf{TIME}(n)$
Quasi-linear time	$\bigcup_{k\geq 0} \mathbf{TIME}(n \log^k(n))$
Quadratic time	$\mathbf{TIME}(n^2)$
Cubic time	$\mathbf{TIME}(n^3)$
Polynomial time	$\mathbf{P} = \bigcup_{k\geq 0} \mathbf{TIME}(n^k)$
Linear-exponential time	$\mathbf{E} = \mathbf{TIME}(2^{O(n)})$
Exponential time	$\mathbf{EXP} = \bigcup_{k\geq 0} \mathbf{TIME}(2^{O(n^k)})$
Double exponential time	$\mathbf{EEXP} = \bigcup_{k\geq 0} \mathbf{TIME}(2^{2^{O(n^k)}})$
Tower-type	$\mathbf{ELEMENTARY} = \bigcup_{k\geq 0} \mathbf{TIME}(\underbrace{2^{2^{\cdots^{2^n}}}}_{k})$

its description, but they cannot predict what M will do without simulating it.

Consider the language we can decide by simulating a Turing machine for $O(n^2)$ steps:

$$A_{TM}^{n^2} = \{x = \langle M \rangle\, 0^k : M \text{ accepts } x \text{ in} \leq |x|^2 \text{ steps}\}$$

We first show $A_{TM}^{n^2}$ is in $\mathbf{TIME}(n^3)$. We can decide $A_{TM}^{n^2}$ with a variant of the Universal Turing machine that has a timer that starts with $|x|^2$ and rejects when the timer hits 0. Each step in the simulation can be completed in $O(n)$ time. The simulation has worst-case time cost $O(n^3)$.

Assume $A_{TM}^{n^2} \in \mathbf{TIME}(n)$. Let T decide $A_{TM}^{n^2}$ with time cost $O(n)$. Let D be the Turing machine that on input $\langle M \rangle\, 0^k$ that

- Calls T on that input.
- Accepts if and only if T rejects.

What does D do on the input for large enough k? D runs in time $O(n)$ (since it runs T). So on large enough k, T completes the simulation of $\langle D \rangle\, 0^k$. It outputs the wrong answer. So T does not decide $A_{TM}^{n^2}$, this is a contradiction. ■

Definition: Time-Constructible

The function $t : \mathbb{N} \to \mathbb{N}$ is time-constructible if there is a Turing machine that on every input of the form 0^n for some $n \geq 1$ writes $t(n)$ on the tape and halts in time $O(t(n))$.

Theorem (Time-Hierarchy Theorem)

For every pair of functions $f, g : \mathbb{N} \to \mathbb{N}$ where $f \log f = o(g)$ and g is time-constructible, $\mathbf{TIME}(f) \subsetneq \mathbf{TIME}(g)$.

5.3 The Class P

Complexity theorists study **P** because it is closed under subroutine calls and is robust (it is invariant under the choice of model of computation in the definition of **TIME** classes)./.

Cobham-Edmonds Thesis

Any decision problem that can be solved in polynomial time by an algorithm on any physically-realizable computer corresponds to a language that is in **P**.

Postulate 1

Every pseudocode deterministic algorithm can be implemented with a Turing machine. If the time complexity of the algorithm is $t(n) = \Omega(n)$, then the worst-case time cost of the corresponding Turing machine is $O(t(n)^k)$ for some constant k ($k = 3$).

Postulate 2

Every algorithm has access to a GetMyTMDescription() function. This function runs in time $O(1)$.

5.4 Mapping Reductions

We want a notion $A \leq B$ where A is no harder than B. For example, if A is undecidable, then B is undecidable. If B is decidable, then A is decidable.

If A is not in **P**, then B is not in **P**. If B is in **P**, then A is in **P**.

Definition: Mapping

The function $f : \{0,1\}^* \to \{0,1\}^*$ is a mapping from $A \subseteq \{0,1\}^*$ to the language $B \subseteq \{0,1\}^*$ if

1. For all $x \in A$, $f(x) \in B$.
2. For all $x \notin A$, $f(x) \notin B$.

Algo$A(x)$

```
y = f(x)
return AlgoB(y)
```

Definition: Mapping Reduction $A \leq_m B$

A has a mapping reduction to be if there is a mapping $f : \{0,1\}^* \to \{0,1\}^*$ from A to B for which there is a Turing machine that on input x outputs $f(x)$ and always halts.

Proposition

If $A \leq_m B$, then

1. If B is decidable, then A is decidable.
2. If A is undecidable, then B is undecidable.

Proof. If $A \leq_m B$, there is an algorithm AlgoF that on input x, outputs $f(x)$ and halts.

1. If B is decidable, there exists AlgoB that decides if $f(x) \in B \iff x \in A$ and halts, so AlgoA decides A.
2. The contrapositive.

Proposition

$$\mathsf{Empty}_{TM} \leq_m EQ_{TM}$$

Proof. $EQ_{TM} = \{\langle M \rangle \langle N \rangle : L(M) = L(N)\}$. Let $f(\langle M \rangle) = \langle M \rangle \left\langle M^{\emptyset} \right\rangle$ where $M^{\emptyset}$ that rejects everything. Then,

1. $\langle M \rangle \in \mathsf{Empty}_{TM} \implies f(\langle M \rangle) = \langle M \rangle \left\langle M^{\emptyset} \right\rangle \in EQ_{TM}$.
2. $\langle M \rangle \notin \mathsf{Empty}_{TM} \implies f(\langle M \rangle) \notin EQ_{TM}$.
3. AlgoF($\langle M \rangle$) returns $\langle M \rangle \left\langle M^{\emptyset} \right\rangle$.

■

Does $\mathsf{Empty}_{TM} \leq_m \overline{\mathsf{Empty}_{TM}}$? No.

Chapter 6

P vs. NP

6.1 The Class NP

Definition: Verifier

A verifier is a multi-tape Turing machine that in addition to its usual tapes has a special certificate tape. The input to a verifier is a pair (x, c) with the input string x written on the first normal tape and the certificate string c written on the certificate tape.

Definition: Recognized, Decides

The language $L(V)$ recognized by the verifier V is

$$L(V) = \{x \in \{0,1\}^* : \exists c \in \{0,1\}^* \text{ s.t. } V \text{ accepts } (x, c)\}$$

The verifier V decides $L(V)$ if it halts on all input pairs (x, c).

Definition: Polynomial-Time Verifier

A verifier that halts on each pair (x, c) in a number of steps polynomial in $|x|$.

The class **NP** is the class of languages decidable by a nondeterministic polynomial-time Turing machine. Another equivalent definition is defined below.

Definition: NP (Nondeterministic Polynomial-Time)

Set of all languages that can be decided by polynomial-time verifiers.

P vs. NP Problem

Can every language that is efficiently verifiable also be efficiently computable?

6.2 NP-Completeness

Definition: Polynomial-Time Reducible

The language $A \subseteq \{0,1\}^*$ is polynomial-time reducible to the language $B \subseteq \{0,1\}^*$, denoted

$$A \leq_{\mathbf{P}} B$$

if and only if there exists a function $f : \{0,1\}^* \to \{0,1\}^*$ such that

1. For every $x \in \{0,1\}^*$, we have $x \in A \Leftrightarrow f(x) \in B$, and
2. There is a polynomial-time Turing machine M that on any input $x \in \{0,1\}^*$, replaces it with $f(x)$ on the tape then halts.

There are also known as Karp reductions and polynomial-time many-to-one reductions.

Lemma

For every two languages $A, B \subseteq \{0,1\}^*$ that satisfy $A \leq_{\mathbf{P}} B$,

1. If $B \in \mathbf{P}$, then $A \in \mathbf{P}$. (If $A \notin \mathbf{P}$, then $B \notin \mathbf{P}$)
2. If $B \in \mathbf{NP}$, then $A \in \mathbf{NP}$. (If $A \notin \mathbf{NP}$, then $B \notin \mathbf{NP}$)

Proof. Let f be a function that shows $A \leq_{\mathbf{P}} B$ and let M_f be a polynomial-time Turing machine that replaces x with $f(x)$ on the tape in polynomial time.

Consider when $B \in \mathbf{P}$ and is decided by the polynomial-time Turing machine M_B. We can decide A with a simple algorithm. On input x, run M_f to replace it with $f(x)$. Simulate M_B. The resulting algorithm decides A since $x \in A \iff f(x) \in B$. It can be implemented with a polynomial-time Turing machine since both M_f and M_B have polynomial-time cost. So $A \in \mathbf{P}$.

We verify A by running M_f and then simulate the polynomial-time verifier V_B for B. Then for any $x \in \{0,1\}^*$, there is a certificate c that causes this verifier to accept (x, c) if and only if the certificate c causes V_B to accept $(f(x), c)$, so x is accepted if and only if $f(x) \in B \iff x \in A$. ■

Definition: NP-Hard

The language L is $\mathbf{NP}$-hard if every language $A \in \mathbf{NP}$ satisfies $A \leq_{\mathbf{P}} L$.

Definition: NP-Complete

The language L is $\mathbf{NP}$-complete if $L \in \mathbf{NP}$ and L is $\mathbf{NP}$-hard.

Proposition

If any **NP**-hard language L is in **P**, then $\mathbf{P} = \mathbf{NP}$.
Equivalently, if $\mathbf{P} \neq \mathbf{NP}$ and L is **NP**-hard, then $L \notin \mathbf{P}$.

Proof. Let L be an **NP**-hard language that is in **P**. Fix any $A \in \mathbf{NP}$. By definition of **NP**-hardness, $A \leq_{\mathbf{P}} L$. By the lemma, $L \in \mathbf{P}$ implies that $A \in \mathbf{P}$. ■

6.3 First NP-Complete Language

Theorem

There exists an **NP**-complete language.

Proof. Consider

$$A_{TM}^{cert} = \{\langle M \rangle \, x 0^m 1^t : \exists c \in \{0,1\}^*, |c| \leq m, M \text{ accepts } (x, c) \text{ in at most } t \text{ steps}\}$$

First we show that A_{TM}^{cert} is in **NP**. Let V be a verifier that simulates at most t transitions of M using its own certificate as c. The check that the certificate c has length at most m and the simulation of t steps of M can both be completed in polynomial-time. V accepts if and only if there is a certificate c of length $|c| \leq m$ that causes M to accept x within at most t steps, so it is a polynomial-time verifier for A_{TM}^{cert}.

We show A_{TM}^{cert} is **NP**-hard. Fix any language $L \in \mathbf{NP}$. This means that there exist polynomials p, q and a verifier V such that for every $x \in \{0,1\}^n$,

1. If $x \in L$, there exists $u \in \{0,1\}^{p(n)}$ that causes V to accept (x, u).
2. If $x \notin L$, every possible $u \in \{0,1\}^{p(n)}$ causes V to reject (x, u).
3. V has time-cost $q(n)$.

Define $f : x \mapsto \langle V \rangle \, x 0^{p(|x|)} 1^{q(|x|)}$. The function f can be computed by a polynomial-time Turing machine and satisfies the condition that $x \in L$ if and only if $f(x) \in A_{TM}^{cert}$, so $L \leq_{\mathbf{P}} A_{TM}^{cert}$. ■

Chapter 7

Polynomial Hierarchy

7.1 The Class coNP

Definition: ∀-Verifier

The language $L(V)$ decided by the ∀-verifier V that halts on all input pairs is

$$L(V) = \{x \in \{0,1\}^* : \forall c \in \{0,1\}^*, V \text{ accepts } (x,c)\}$$

Definition: Polynomial-Time ∀-Verifier

A ∀-verifier that decides its language and has time cost $O(n^k)$ for some constant $k \geq 0$.

Definition: coNP

The set of all languages that can be decided by polynomial-time ∀-verifiers.

Proposition

The language $L \subseteq \{0,1\}^*$ is in **NP** if and only if its complement $\overline{L}$ is in **coNP**.

Proof. For any verifier V, let V' be the ∀-verifier obtained by interchanging the Accept and Reject states of V. Then a string $x \in \{0,1\}^*$ is accepted by V if and only if the same string is rejected by V', because a certificate that causes V to accept (x,c) causes V' to reject the same input pair. So L has a polynomial-time verifier V if and only if there is a polynomial-time ∀-verifier V' that rejects exactly the set of L of strings, or V' decides L. ■

Proposition

If $\mathbf{P} = \mathbf{NP}$, then $\mathbf{P} = \mathbf{coNP}$.

Proof. For any language $L \in \mathbf{coNP}$, then $\overline{L} \in \mathbf{NP}$. If $\mathbf{P} = \mathbf{NP}$, $\overline{L} \in \mathbf{P}$. But then $L \in P$.
■

Proposition

If $\mathbf{NP} \neq \mathbf{coNP}$, then $\mathbf{P} \neq \mathbf{NP}$.

Proof. If $\mathbf{P} = \mathbf{NP}$, then $\mathbf{P} = \mathbf{coNP}$, so $\mathbf{NP} = \mathbf{coNP}$. ■

If $\mathbf{P} \neq \mathbf{NP}$, then we do not know if $\mathbf{NP} \neq \mathbf{coNP}$.

Let $\mathsf{FACTORING} = \{\langle k \rangle \langle \ell \rangle \langle u \rangle : k \text{ has a prime factor } p \text{ in } \ell \leq p \leq u\}$. This is in **NP** by giving the actual prime p as a certificate. The complement is in **NP** by giving the complete list of prime factors as the certificate.

7.2 The Class PH

Let MaxClique be the language $\{\langle G \rangle \langle k \rangle : \text{largest clique in } G \text{ has size } k\}$.

Definition: ∃∀-Verifier

The language $L(V)$ decided by the ∃∀-verifier V that halts on all input tuples (x, c_1, c_2) is

$$L(V) = \{x \in \{0,1\}^* : \exists c_1 \forall c_2 \text{ such that } V \text{ accepts } (x, c_1, c_2)\}$$

Definition: $\mathbf{\Sigma_2}$

The set of all languages that can be decided by polynomial-time ∃∀-verifiers.

Definition: ∀∃-Verifier

The language $L(V)$ decided by the ∃∀-verifier V that halts on all input tuples (x, c_1, c_2) is

$$L(V) = \{x \in \{0,1\}^* : \forall c_1 \exists c_2 \text{ such that } V \text{ accepts } (x, c_1, c_2)\}$$

Definition: $\mathbf{\Pi_2}$

The set of all languages that can be decided by polynomial-time ∀∃-verifiers.

$\mathbf{\Sigma_0} = \mathbf{\Pi_0} = \mathbf{P}$, $\mathbf{\Sigma_1} = \mathbf{NP}$, $\mathbf{\Pi_1} = \mathbf{coNP}$.

Definition: Polynomial Hierarchy

The class of languages defined by

$$\mathbf{PH} = \bigcup_{k \geq 1} \mathbf{\Sigma_k} = \bigcup_{k \geq 1} \mathbf{\Pi_k}$$

7.3 Properties of PH

Proposition

$$\mathbf{PH} \subseteq \mathbf{EXP}$$

Proof. Exhaustive search over all certificates. ■

Definition: $\mathcal{C}$-Complete

For any class $\mathcal{C}$ of languages that includes $\mathbf{P}$, a language $L \subseteq \{0,1\}^*$ is $\mathcal{C}$-complete if $L \in \mathcal{C}$ and every $A \in \mathcal{C}$ satisfies $A \leq_{\mathbf{P}} L$.

Theorem

If the polynomial hierarchy does not collapse (i.e. if $\mathbf{\Sigma_k} \subsetneq \mathbf{\Sigma_{k+1}}$ for all $k \geq 1$), then there is $\mathbf{PH}$-complete language.

Chapter 8

Boolean Circuits

8.1 Introduction

Definition: Boolean Circuit

N input bits labelled $x_1, \ldots, x_n$ and a sequence of gates $g_1, \ldots, g_m$, with the three types of gates

- NOT gates ($\neg$): one input bit and negate it.
- AND gates ($\wedge$): any number of input bits and output 1 if and only if all the input bits have value 1.
- OR gates ($\vee$): any number of input bits and output 1 if and only if at least one of the input bits have value 1.

There are two complexity measures for Boolean circuits:

- The size of a circuit is the total number of AND and OR gates.
- The depth of the circuit is the number of AND and OR gates in the longest path from any input bit to the output of the circuit.

8.2 First Examples

Definition: Parity

The parity of a set of bits is the function $\oplus_n : \{0,1\}^n \to \{0,1\}$ defined by

$$\oplus_n(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i \pmod 2$$

that outputs 1 if and only if x contains an odd number of 1s.

8.3 Shannon's Theorem

Theorem

For every function $f : \{0,1\}^n \to \{0,1\}$, there exists a Boolean circuit C of size at most $2^n + 1$ that computes f.

Proof. Every Boolean function f can be represented as a table of truth values. The circuit can be obtained in the following way. We create one AND gate for each row with function value 1 with n inputs corresponding to the n literals in that row. We then connect all the AND gates with a single OR gate to produce the output of the circuit.

By construction, the circuit outputs 1 if and only if the inputs to the circuit belong to one of the 1-valued rows in the table, so it computes f. The circuit has a total of n NOT gates, at most 2^n AND gates, and a single OR gate. ■

Corollary

For every language $L \subseteq \{0,1\}^*$, there exists a family of circuits $\{C_n\}_{n \geq 0}$ that computes L.

Theorem (Shannon's Theorem)

For every $n \geq 4$, there exists a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by any Boolean circuit of size $< 2^{n/2}/2$.

Proof. There are 2^{2^n} distinct Boolean functions on n bits. There are 2^m ways to label each gate with AND or OR. Each gate has 2^{n+m} possible inputs plus $\leq 2^{m+n}$ ways to choose which of these inputs to negate. So there are $2^{2(m+n)}$. Therefore, the total number of circuits is

$$2^m \cdot (2^{2(m+n)})^m \leq 2^{4m^2}$$

When $m < 2^{n/2}/2$, then the number of circuits of size m is $\leq 2^{4m^2} < 2^{4(2^{n/2}/2)^2} = 2^{2^n}$. ■

Chapter 9

Non-Uniform Computation

9.1 Circuits Simulate Turing Machines

Theorem

For any single-tape Turing machine M that runs in time t and uses at most s squares of tape on any input of length n, there is a circuit C of size $O(st)$ that outputs 1 on input $x \in \{0, 1\}^n$ if and only if M accepts x.

Proof. We use the representation of the configurations of M as a tableau. We can build each row of the tableau for M given the previous row using a circuit. Specifically we create one circuit for each bit of the current row. The input to each of these circuits is a subset of the bits in the previous row. Each of these circuits only depend on a constant number of bits of the previous row since the state and symbol for the ith square in the tape only ever depend on the contents of the squares $i-1, i, i+1$ in the previous configuration. As a result, each of these circuits can be implemented with $O(1)$ gates, and the total size of the circuit is the size of the tableau which is $O(st)$. ■

Corollary

Every language in $\mathbf{P}$ can be computed with a family $\{C_n\}_{n \geq 1}$ of polynomial-size circuits.

Proposition

The exist undecidable languages that can be computed by constant-size circuits.

Proof. There exists an undecidable language $L \subseteq \{0\}^*$. For $0^n \notin L$, just output 0 and for $0^n \in L$, we can negate all bits and AND them. ■

Circuits are a non-uniform model of computation.

9.2 The Class P/poly

To give Turing machines non-uniformity, we give the Turing machine an advice tape that depends only on the length of the input.

Definition: Turing Machine with Advice

A Turing machine that in addition to its usual tape also has a special advice tape and is defined with a sequence of *advice strings* $a_0, a_1, \ldots$, one for each $n \in \mathbb{N}$.
When running a Turing machine with advice on input $x \in \{0,1\}^n$, the machine is initialized with the advice string a_n corresponding to the length of x on the advice tape.

Definition: P/poly

The set of languages that can be decided by polynomial-time Turing machines with advice strings that have length polynomial in n.

Theorem

The language class **P/poly** is exactly the set of languages that can be computed by families of polynomial-size Boolean circuits.

Proof. If $\{C_n\}_{n\geq 0}$ computes L and all the circuits have polynomial size, then

$$a_n = \langle C_n \rangle$$

We need 1 bit to encode AND and OR gates, $m+n$ bits to say which inputs go to the gate, and $m+n$ bits to say which inputs are negated. So the total is $O(m(m+n)) = O(m^2)$.

The Turing machine can simulate C_n in polynomial-time. ■

Theorem (Karp-Lipton)

If $\mathbf{NP} \subseteq \mathbf{P/poly}$, then $PH = \Sigma_2$.

Chapter 10

Formula Complexity

10.1 Boolean Formulas

We can bound certain parameters. A first parameter is the depth of the circuit. The class **AC0** is the class of circuits with constant depth. Another parameter is the bounded fan-in. If we set this to 2, we can represent circuits with depth $\log n$. This is the class **NC1**.

If we have fan-in 2 and fan-out 1, we have a Boolean formula with bounded fan-in. We will denote $L(F)$ as the number of leaves in the tree representation of the formula F. This is equal to the number of AND or OR gates plus 1.

10.2 First Examples

For the parity circuit, we had a tree of XOR. To change it to a Boolean formula, for each XOR gate, we have 4 different inputs (we are allowed to use the negation in the input in addition). We need $n \cdot 2^d = n \cdot 2^{\log n} = n^2$ leaves.

10.3 Restriction Method

Theorem (Suhbotovskaya)

$$L(\oplus_n) \geq n^{1.5}$$

Definition: Restricted Function

Given a function $f : \{0,1\}^n \to \{0,1\}$, an index $i \in \{1, \ldots, n\}$, and a value $b \in \{0,1\}$, it is the function that hard codes the ith bit to b in f.

Lemma

For every function $f : \{0,1\}^n \to \{0,1\}$ and every formula F that computes f, there is an index $i \in \{1, \dots, n\}$ and a value $b \in \{0,1\}$ such that the restricted function $f' : \{0,1\}^{n-1} \to \{0,1\}$ defined by

$$f'(x_1, \dots, x_{n-1}) = f(x_1, \dots, x_{i-1}, b, x_i, \dots, x_{n-1})$$

can be computed by a formula F' of leaf size

$$L(F') \leq \left(1 - \frac{1}{n}\right)^{3/2} L(F)$$

Proof. There must be an index i such that at least $\frac{L(F)}{n}$ leaves are labelled with x_i or $\overline{x}_i$. Without loss of generality, there are $\geq \frac{L(F)}{n}$ gates with x_i or $\overline{x}_i$ as one of their inputs.

For each gate w, x_i or $\overline{x}_i$ as one input, there is one assignment $x_i = b$ that eliminates the gate and so at least one other leaf. This implies that there exists $b \in \{0,1\}$ that eliminates $\geq \frac{L(F)}{2n}$ gates.

The formula F' obtained by removing x, leaves and other eliminates leaves has leaf size

$$L(F') \leq L(F) - \frac{l(F)}{n} - \frac{L(F)}{2n} = \left(1 - \frac{3}{2n}\right) L(F) \leq \left(1 - \frac{1}{n}\right)^{3/2} L(F)$$

■

Proof. (Suhbotovskaya) Let F compute $\oplus_n$. Apply the lemma to get $f^{(1)}$ computed by $F^{(1)}$,

$$L(F^{(1)}) \leq \left(1 - \frac{1}{n}\right)^{3/2} L(F)$$

Apply lemma again to $f^{(1)}$ to get $f^{(2)}$ computed by $F^{(2)}$,

$$L(F^{(2)}) \leq \left(1 - \frac{1}{n-1}\right)^{3/2} L(F^{(1)})$$

Apply lemma until we get $f^{(n-1)}$ computed by $F^{(n-1)}$. $f^{(n-1)}$ is x_j or $\overline{x}_j$ for some $j \in \{1, \dots, n\}$. Then

$$\begin{aligned} 1 \leq L(F^{(n-1)}) &\leq \left(1 - \frac{1}{n}\right)^{3/2} \left(1 - \frac{1}{n-1}\right)^{3/2} \cdots \left(1 - \frac{1}{2}\right)^{3/2} L(F) \\ &= \left(\left(1 - \frac{1}{n}\right)\left(1 - \frac{1}{n-1}\right) \cdots \left(1 - \frac{1}{2}\right)\right)^{3/2} L(F) \\ &= \left(\frac{n-1}{n} \cdot \frac{n-2}{n-1} \cdots \frac{2}{3} \cdot \frac{1}{2}\right)^{3/2} L(F) \\ &= \frac{1}{n^{3/2}} L(F) \end{aligned}$$

So we get $L(F) \geq n^{3/2}$. ■

Chapter 11

Satisfiability

11.1 Circuit Satisfiability

Definition: CircuitSAT

$$\mathsf{CircuitSAT} = \{\langle C \rangle : C \text{ is satisfiable}\}$$

Theorem

The language CircuitSAT is **NP**-complete.

Proof. CircuitSAT $\in$ **NP** because we can design a verifier V that on input $\langle C \rangle$ asks for a satisfying assignment y as the certificate and accepts if and only if $C(y) = 1$.

CircuitSAT is **NP**-hard. Fix any $A \in$ **NP**. Let V be a polynomial-time verifier for A. We can construct a polynomial-size circuit C that simulates V on any input (x, c).

Let C_x be C with the value of x hard coded. Its inputs are the bits of c only. Then C_x is satisfiable if and only if there exists c such that $C_x(c) = 1$ if and only if there exists c such that V accepts (x, c) if and only if $x \in A$. So $x \mapsto \langle C_x \rangle$ gives a polynomial-time reduction from A to CircuitSAT. ■

11.2 Cook-Levin Theorem

Definition: SAT

$$\mathsf{SAT} = \{\langle F \rangle : F \text{ is a satisfiable formula}\}$$

Theorem (Cook-Levin Theorem)

SAT is **NP**-complete.

Proof. SAT $\in$ **NP** because we can verify a given assignment x satisfies F given the encoding of F.

CircuitSAT $\leq_{\mathbf{P}}$ SAT. Given $\langle C \rangle$ with inputs $x_1, \ldots, x_n$, we construct $\langle F \rangle$ with inputs $x_1, \ldots, x_n$ and x_g for each gate g in C.

When x_g is associated with $\wedge$ gate that takes inputs $x_a, \ldots, x_z$, then the constraint

$$(x_g \vee \overline{x}_a \vee \cdots \vee \overline{x}_z) \wedge (\overline{x}_g \vee x_a) \wedge \cdots \wedge (\overline{x}_g \vee x_z)$$

is satisfied if and only if $x_g = x_a \wedge \cdots \wedge x_z$.

When x_g is associated with $\vee$ gate that takes inputs $x_a, \ldots, x_z$, then the constraint

$$(\overline{x}_g \vee x_a \vee \cdots \vee x_z) \wedge (x_g \vee \overline{x}_a) \wedge \cdots \wedge (x_g \vee \overline{x}_z)$$

is satisfied if and only if $x_g = x_a \vee \cdots \vee x_z$.

When x_g is $\neg$ gate with input x_a, then the constraint

$$(x_g \vee x_a) \wedge (\overline{x}_g \vee x_a)$$

is satisfied if and only if $x_g = \neg x_a$. Then $F = \bigwedge(\text{of all } x_g \text{ correctness conditions}) \wedge x_{g_m}$ is satisfiable if and only if C is satisfiable, $\langle F \rangle$ is computable in polynomial-time. ■

11.3 3SAT

Definition: Conjunctive Normal Form (CNF) Formula

A Boolean formula with unbounded fan-in of depth 2, where the bottom-most gate is a single $\wedge$ gate, the inputs to that output gate are $\vee$ gates, and the inputs to each $\vee$ gates are subsets of the input variables or their negations.

Definition: Clause

Each $\vee$ gate in a CNF formula.

Definition: Width of a Clause

The number of inputs of a $\vee$ gate.

Definition: Size

The number of clauses the formula contains.

Definition: Width

Maximum width of all clauses.

Definition: 3SAT

$$\mathsf{3SAT} = \{\langle F \rangle : F \text{ is a satisfiable CNF formula of width} \leq 3\}$$

Theorem

$\mathsf{3SAT}$ is **NP**-complete.

Proof. Same as SAT proof (already in CNF form) except

1. Correct C to bounded fan-in C' first, or
2. We can turn any clause $x_1 \vee x_2 \vee \cdots \vee x_k$ by changing it to

$$(x_1 \vee x_2 \vee y_1) \wedge (\overline{y}_1 \vee x_3 \vee y_2) \wedge (\overline{y}_2 \vee x_4 \vee y_3) \wedge \cdots \wedge (\overline{y}_{k-3} \vee x_{k-1} \vee x_k)$$

Chapter 12

Randomized Computation

12.1 Probabilistic Turing Machines

Definition: Probabilistic Turing Machine

A Turing machine that has a standard input tape as well as a read-only randomness tape. The definition of a Turing machine includes a function $\ell : \mathbb{N} \to \mathbb{N}$ that determines the length of the random string generated for inputs of all lengths. It has independent tape heads for the input and randomness tapes, and each transition is determined by the current symbol under both tape heads.

On input x of length n, a probabilistic Turing machine's randomness tape is initialized with a string $r \in \{0,1\}^{\ell(n)}$ drawn uniformly at random from the binary strings of length $\ell(n)$.

Definition: ϵ-Decides

Fix $0 < \epsilon < \frac{1}{2}$. The probabilistic Turing machine M ϵ-decides the language $L_\epsilon(M)$ when

- M always halts (on all inputs x and on all random strings r).
- For every input $x \in L_\epsilon(M)$,

$$\Pr_{r \in \{0,1\}^{\ell(|x|)}}[M \text{ accepts } (x, r)] \geq 1 - \epsilon.$$

- For every input $x \notin L_\epsilon(M)$,

$$\Pr_{r \in \{0,1\}^{\ell(|x|)}}[M \text{ accepts } (x, r)] \leq \epsilon.$$

Definition: BPP (Bounded-Error Probabilistic Polynomial-Time)

The set of all languages that can be $\frac{1}{3}$-decided by polynomial-time probabilistic Turing machines.

12.2 Primality Testing

Let $\mathsf{PRIME} = \{\langle N \rangle : N \text{ is a prime number}\}$, where $\langle N \rangle$ represents the binary encoding of a number and has length $n = \lceil \log N \rceil$.

Theorem

$\mathsf{PRIME} \in \mathbf{BPP}$.

Definition: Prime-Like Relative

A positive integer N is prime-like relative to the integer $a \in \{1, \ldots, N-1\}$ if

- $a^{N-1} \equiv 1 \pmod{N}$, and
- Let $N - 1 = d \cdot 2^s$ where d odd. For all $r \leq s$, if $a^{2^r d} \equiv 1 \pmod{N}$, then $a^{2^{r-1} d} \equiv \pm 1 \pmod{N}$.

Fact 1

When $N > 2$ is any odd prime number, then N is prime-like relative to every $a \in \{1, \ldots, N-1\}$.

Fact 2

When $N > 2$ is any odd composite number, then N is prime-like relative to at most $\frac{N-1}{4}$ of the integers $a \in \{1, \ldots, N-1\}$.

Algorithm 1 Miller-Rabin Primality Test(N)

```
if N = 2 then Accept
if N is even then Reject
Draw a ∈ {1, ..., N − 1} uniformly at random
if N is prime-like relative to a then
    Accept
else
    Reject
```

Miller-Rabin primality test has time cost $O(\log^3 N) = O(n^3)$.

Lemma

The Miller-Rabin primality test $\frac{1}{3}$-decides the PRIME language.

Proof. MRT always halts.

When N is prime, MRT always accepts.

$$\Pr_r[M \text{ accepts } (x,r)] = 1 \geq 1 - \frac{1}{3}$$

When N is composite, if N is even, MRT never accepts. If N is odd, by fact 2,

$$\Pr_r[\text{MRT accepts } (x,r)] \leq \frac{(N-1)/4}{N-1} = \frac{1}{3} \leq \frac{1}{3}$$

■

12.3 Success Amplification

Theorem

For every language $L \in \mathbf{BPP}$ and any two constants $c_0, c_1 \geq 0$, there is a polynomial-time probabilistic Turing machine that ϵ-decides L with $\epsilon = e^{-c_0 n^{c_1}}$.

Definition: Chernoff Bound/Hoeffding Inequality (Special Case)

Let $X_1, \ldots, X_k$ be independent random variables that each take the values 0 or 1. Define $T = \frac{1}{k} \sum_{i=1}^{k} X_k$ to be their average value. Then for any $t > 0$,

$$\Pr[T - \mathbb{E}[T] \geq t] \leq e^{-2t^2 k}$$

Proof. (Success Amplification) Let A $\frac{1}{3}$-decide L. Let B run $k =$ copies of A and return the most common output.

For $i = 1, \ldots, k$, let

$$X_i = \begin{cases} 1 & \text{if } i\text{th instance of } A \text{ is wrong} \\ 0 & \text{if } i\text{th instance of } A \text{ is right} \end{cases}$$

Then with $T = \frac{1}{k} \sum X_i$,

$$\mathbb{E}[T] = \mathbb{E}\left[\frac{1}{k} \sum X_i\right] = \frac{1}{k} \sum \mathbb{E}[X_i] =\leq \frac{1}{k} \cdot k \cdot \frac{1}{3} = \frac{1}{3}$$

Then

$$\Pr[B \text{ is wrong}] = \Pr\left[T \geq \frac{1}{2}\right] = \Pr\left[T - \mathbb{E}[T] \geq \frac{1}{2} - \mathbb{E}[T]\right] \leq \Pr\left[T - \mathbb{E}[T] \geq \frac{1}{6}\right]$$

Chapter 13

P vs. BPP

13.1 BPP and Polynomial-Size Circuits

Theorem (Adleman)

$\mathbf{BPP} \subseteq \mathbf{P/poly}$.

Proof. Fix any language $L \in \mathbf{BPP}$. By the success ampflication theorem, there exists a polynomial-time probabilistic Turing machine A that $2^{-(n+1)}$-decides the language L.

Fix any $n \in \mathbb{N}$. Let $m = \ell(n)$ be the length of the random strings used by A on inputs of length n. We will say that a random string $r \in \{0,1\}^m$ is bad for input $x \in \{0,1\}^n$ if A does not correctly accept or reject (x, r).

For any fixed $x \in \{0,1\}^n$, the randomness guarantee on A implies that there are at most $2^m/2^{n+1}$ random strings $r \in \{0,1\}^m$ that are bad for any input of length n. Therefore, for at least half of the random strings, A correctly accepts or rejects every x. We can then design a polynomial-time Turing machine with advice that correctly decides L by choosing on of the good random strings to be the advice string a_n then simulating A with this advice string in place of the random string. ■

13.2 BPP and the Polynomial Hierarchy

Definition: Shift of S

For a set $S \subseteq \{0,1\}^m$ and a vector $u \in \{0,1\}^m$, the shift of S by u is the set

$$S \oplus u := \{x \oplus u : x \in S\}$$

where $x \oplus y = (x_1 \oplus y_1, \ldots, x_m \oplus y_m)$.

Lemma

Fix any $n \geq 1$ and m in the range $n < m < 2^n$. For any $S \subseteq \{0,1\}^m$ of cardinality $|S| \leq 2^{m-n}$ and any set of $k = \left\lceil \frac{m}{n} \right\rceil$ vectors $u_1, \ldots, u_k \in \{0,1\}^m$, we have

$$\bigcup_{i=1}^{k} (S \oplus u_i) \neq \{0,1\}^m$$

Proof. For every vector $u \in \{0,1\}^m$, $|S \oplus u| = |S| \leq 2^{m-n}$. So by the union bound,

$$\left| \bigcup_{i=1}^{k} (S \oplus u_i) \right| \leq k\,|S| \leq k \frac{2^m}{2^n} = \frac{k}{2^n} 2^m$$

This cardinality is strictly less than 2^m when $k < 2^n$.

Lemma

Fix any $n \geq 1$ and any $m \geq n$. For any $S \subseteq \{0,1\}^m$ of cardinality $|S| \geq (1 - 2^{-n})2^m$, there exist $k = \left\lceil \frac{m}{n} \right\rceil + 1$ vectors $u_1, \ldots, u_k \in \{0,1\}^m$ such that

$$\bigcup_{i=1}^{k} (S \oplus u_i) = \{0,1\}^m$$

Theorem

$\mathbf{BPP} \subseteq \mathbf{\Sigma_2} \cap \mathbf{\Pi_2}$.

Proof. We show $\mathbf{BPP} \subseteq \mathbf{\Sigma_2}$. Fix any language $L \in \mathbf{BPP}$. By the success amplification theorem, there exists a polynomial-time probabilistic Turing machine M that 2^{-n}-decides L. For any $x \in \{0,1\}^n$ and set $m = \ell(n)$. Let $S_x \subseteq \{0,1\}^m$ denote the set of random strings r for which M accepts (x, r). The correctness guarantees of M imply that

- If $x \in L$, then $|S_x| \geq (1 - 2^{-n})2^m$.
- If $x \notin L$, then $|S_x| \leq 2^{-n}2^m$.

By previous two lemmas, we get that $x \in L$ if and only if there exist $u_1, \ldots, u_k \in \{0,1\}^m$ for which every $r \in \{0,1\}^m$ satisfies $r \in \bigcup_{i=1}^{k}(S_x \oplus u_i)$. By definition, $r \in \bigcup_{i=1}^{k}(S_x \oplus u_i)$ if and only if M accepts $(x, r \oplus u_i)$ for at least 1 index $i \in 1, \ldots, k$.

Since the check that M accepts at least one pair $(x, r \oplus u_i)$ can be performed in polynomial-time, this means that we have obtained a polynomial-time $\mathbf{\Sigma_2}$-verifier for L.

The proof that $L \in \mathbf{\Pi_2}$ is obtained by symmetry of $\mathbf{BPP}$. When $L \in \mathbf{BPP}$, the so is $\overline{L} \in \mathbf{BPP}$. By the argument above, this means $\overline{L} \in \mathbf{\Sigma_2}$, so $L \in \mathbf{\Sigma_2}$. ■

13.3 Polynomial Identity Testing

Definition: PIT

$\mathsf{PIT} = \{\langle C \rangle : C \text{ is the 0 polynomial}\}$.

This can be computed in polynomial time with randomized algorithms.

Theorem

$\mathsf{PIT} \in \mathbf{BPP}$.

We do not know if PIT can be decided in polynomial-time deterministically.

Chapter 14

Randomized Verification

14.1 Merlin-Arthur

Definition: Probabilistic Verifier

A Turing machine with two additional tapes on top of its usual input tape: one certificate tape and a randomness tape.

Definition: ϵ-Decides

The probabilistic verifier V ϵ-decides the language L for some $\epsilon \geq 0$ if it always halts and

- $x \in L \Rightarrow \exists c \Pr_r[V \text{ accepts } (x, c, r)] \geq 1 - \epsilon$
- $x \notin L \Rightarrow \exists c \Pr_r[V \text{ accepts } (x, c, r)] \leq \epsilon$

Definition: MA

The set of languages that can be $\frac{1}{3}$-decided by polynomial-time probabilistic verifiers.

Claim

$\mathbf{NP} \cup \mathbf{BPP} \subseteq \mathbf{MA}$.

Definition: Succinct Minimum Arithmetic Circuit Size Problem

$\mathsf{SuccinctMACSP} = \{\langle C \rangle : \exists C', |\langle C' \rangle| < |\langle C \rangle| ; C, C' \text{ compute the same polynomial}\}$.

Proposition

$\mathsf{SuccinctMACSP} \in \mathbf{MA}$.

Proof. Consider the probabilistic verifier with certificate $|\langle C' \rangle|$. It checks that $|\langle C' \rangle| < |\langle C \rangle|$ and then uses the **BPP** algorithm for polynomial identity testing to check that C and C'

are the same polynomial. ■

14.2 Interactive Proofs

Definition: 2-Round Interactive Verifier

A probabilistic Turing machine with an additional read-write oracle tape. It has a special Query Oracle state that it can use once during its computation. The contents of the oracle is changed to a certificate (that depends on the input and query written on oracle tape). The verifier proceeds exactly as with standard probabilistic verifiers.

Definition: IP[2]

The set of languages that can be $\frac{1}{3}$-decided by polynomial-time 2-round interactive verifiers.

Arthur gets to use his random bits and the input to decide what query string q to send to Merlin. Merlin responds with a certificate c that depends on x and q, but not the random string r Arthur has. This is a private randomness model.

Theorem

$\mathsf{GraphNonIso} \in \mathbf{IP}[\mathbf{2}]$.

Proof. Arthur picks G or H at random, then uniformly at random permutes the vertices of the selected graph. He sends the result as a query to Merlin, and Merlin answers G or H. ■

14.3 Arthur-Merlin

Definition: $\epsilon - (\Pr\exists)$Decides

The probabilistic verifier V $\epsilon - (\Pr\exists)$-decides the language L for some $\epsilon \geq 0$ if it always halts and

- $\Pr_r[\exists c : V \text{ accepts } (x, c, r)] \geq 1 - \epsilon$
- $\Pr_r[\exists c : V \text{ accepts } (x, c, r)] \leq \epsilon$

Definition: AM

The set of all languages that can be $\frac{1}{3} - (\Pr\exists)$-decided by a polynomial-time probabilistic Turing machine.

Theorem

$\mathbf{MA} \subseteq \mathbf{AM}$.

Proof. Fix $L \in \mathbf{MA}$. Let V $\frac{1}{3}$-verify L. By success amplification, we get V' that

- $x \in L \implies \exists c \Pr_r[V' \text{ accepts } (x, c, r)] \geq 1 - 4^{-\ell(n)}$
- $x \notin L \implies \forall c \Pr_r[V' \text{ accepts } (x, c, r)] \leq 4^{-\ell(n)}$

where $\ell(n)$ is the length of certificates on inputs of length n.

Consider V' as a $(\Pr \exists)$-verifier.

- $x \in L$: When Merlin outputs the correct $\mathbf{MA}$ certificate,

$$\Pr_r[V' \text{ accepts } (x, c, r)] \geq 1 - 4^{-\ell(n)} \gg \frac{2}{3}$$

- $x \notin L$: By union bound,

$$\Pr_r[\exists c : V' \text{ accepts } (x, c, r)] \leq \sum_c \Pr[V' \text{ accepts } (x, c, r)] \leq 2^{\ell(n)} 4^{-\ell(n)} = \frac{1}{2^{\ell(n)}} \ll \frac{1}{3}$$

■

Theorem

For any constant $k \geq 2$, $\mathbf{AM} = \mathbf{IP}[\mathbf{k}]$.

Chapter 15

Space Complexity

15.1 Space Complexity Classes

Definition: Space Cost

The total number of distinct tape cells that are visited by M's tape head before M halts on input x.

Definition: Worst-Case Space Cost

The function $s : \mathbb{N} \to \mathbb{N}$ where $s(n)$ is the maximum space cost of M on any input x of length $|x| = n$.

Definition: SPACE(s)

For every function $s : \mathbb{N} \to \mathbb{N}$, **SPACE**($s$) is the set of all languages that can be decided by a Turing machine with worst-case space cost bounded above by $O(s)$.

Definition: NSPACE(s)

The set of languages that can be decided by verifiers with space cost $O(s)$.

Definition: PSPACE

$\bigcup_{k \geq 1} \mathbf{SPACE}(n^k)$.

Definition: NPSPACE

$\bigcup_{k \geq 1} \mathbf{NSPAPCE}(n^k)$.

Definition: BPPSPACE

All languages $\frac{1}{3}$-decidable by polynomial-space probabilistic Turing machines.

Definition: EXPSPACE

$\bigcup_{k \geq 0} \textbf{SPACE}(2^{n^k})$.

Proposition

$\mathsf{SAT} \in \textbf{PSPACE}$.

Proof. Given $\langle \phi \rangle$, we can iterate over all possible assignments to $x_1, \ldots, x_n$ and check if any satisfy ϕ, reusing the same space for each check. Start with $x = 0^n$, then if this does not satisfy, increment the assignment and repeat. ∎

Proposition

The Totally Quantified Boolean Formula language

$$\mathsf{TQBF} = \{\langle \phi \rangle : \exists x_1 \forall x_2 \cdots \exists/\forall x_n \text{ s.t. } \phi(x_1, \ldots, x_n) = 1\}$$

is in **PSPACE**.

Proof. (Sketch) Same as SAT. We iterate over all possible assignments of $x_1, \ldots, x_n$ to see if the condition on ϕ is satisfied, *reusing* the space. ∎

15.2 Time and Space

Theorem (Time-Space Hierarchy)

For every function $s : \mathbb{N} \to \mathbb{N}$,

$$\textbf{NTIME}(s) \subseteq \textbf{SPACE}(s) \subseteq \textbf{TIME}(2^{O(s)})$$

Proof. ($\textbf{SPACE}(s) \subseteq \textbf{TIME}(2^{O(s)})$) The number of configurations of M that decides a language $L \in \textbf{SPACE}(s)$ is

$$O(1) \cdot s \cdot |\Gamma|^s = m \cdot O(s) \cdot |\Gamma|^{O(s)} = 2^{O(s)}$$

Any M that always halts visits each configuration at most once. ∎

Corollary

$\textbf{P} \subseteq \textbf{NP} \subseteq \textbf{PSPACE} \subseteq \textbf{EXP}$.

15.3 Savitch's Theorem

Definition: STCON

$$\mathsf{STCON} = \{\langle G, s, t, k\rangle : \text{there is a path of length } \le k \text{ from } s \text{ to } t \text{ in } G\}$$

We use an exhaustive search through all possible (s,t)-paths of length k and reuse the space. Each vertex uses $O(\log n)$ bits to store, so we use a total of $O(k \log n)$ squares of the tape. We can do better using a divide-and-conquer algorithm.

Theorem

STCON can be decided by a Turing machine that uses $O(\log k \cdot \log n)$ additional space.

Proof. If $k = 1$, if $s = t$ or $(s,t) \in E(G)$, accept, otherwise, reject. Otherwise, for each $u \in V(G)$, if $\mathsf{STCON}(G, s, u, \lceil k/2 \rceil)$ and $\mathsf{STCON}(G, u, t, \lfloor k/2 \rfloor)$, accept, otherwise, reject. ■

Theorem (Savitch's Theorem)

For every $s : \mathbb{N} \to \mathbb{N}$,

$$\mathbf{NSPACE}(s(n)) \subseteq \mathbf{SPACE}(s(n)^2)$$

Proof. Fix any $L \in \mathbf{NSPACE}(s)$. Let V be a verifier that decides L and has space cost $O(s(n))$.

Let G be a configuration graph for V, where partial configurations include

- current state
- position and string on input tape
- position on certificate tape

There is an edge (c_1, c_2) if there exists a certificate that causes the transition to happen in one step. So $x \in L$ if and only if there is a certificate that causes V to accept if and only if there exists a path in G from the initial configuration c_x to accept configuration c_A. We can solve this, by using STCON, in

$$O(\log k \cdot \log N) = O(\log^2 N) = O(s^2)$$

We do not need additional space for storing G in the input, since it suffices to examine if $(u,v) \in E(G)$. So the space cost is $O(s(n)^2)$. ■

Corollary

$\mathbf{PSPACE} = \mathbf{NPSPACE}$.

Chapter 16

Logarithmic Space

16.1 L and NL

Definition: Input/Work Tape Turing Machine

A two-tape Turing machine whose input is on the first read-only tape and where the other tape is a standard read/write tape that is initially empty.

For a verifier, we also have a read-only certificate tape, but we only allow reading from left to right.

Definition: L

$\mathbf{SPACE}(\log n)$.

Definition: NL

$\mathbf{NSPACE}(\log n)$.

Definition: polyL

$\bigcup_{k\geq 1} \mathbf{SPACE}(\log^k n)$.

Proposition

$\mathsf{PAL} = \{x : x = x^R\}$ is in L.

Proof. Check index i and $n+1-i$ of x; increment i. ■

Proposition

$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}$.

16.2 NL-Completeness

Definition: Log-Space Reducible

Given two languages $A, B \subseteq \{0,1\}^*$, the language A is log-space reducible to B, denoted
$$A \leq_{\mathbf{L}} B$$
if there is a function $f : \{0,1\}^* \to \{0,1\}^*$ such that

1. For every $x \in \{0,1\}^*$, we have $x \in A \Leftrightarrow f(x) \in B$.
2. There is a logarithmic-space Turing machine M that on input $x \in \{0,1\}^*$, index $i \in \mathbb{N}$, and symbol $\sigma \in \{0,1\}$, accepts if and only if the ith symbol of $f(x)$ is σ.

Lemma

For every two languages A, B that satisfy $A \leq_{\mathbf{L}} B$,

- If $B \in \mathbf{L}$, then $A \in \mathbf{L}$.
- If $B \in \mathbf{NL}$, then $A \in \mathbf{NL}$.

Proof. Let M_f be a log-space machine for the log-space reduction and M_B be the log-space machine that decides B. We simulate access to the input tape for M_B. Use a counter to keep track of the tape head position of M_B on the input tape. Use M_f to determine what symbol of $f(x)$ is at that position to simulate each transition of M_B. ■

Definition: NL-Complete

The language L is **NL**-complete if

1. $L \in \mathbf{NL}$.
2. Every language $A \in \mathbf{NL}$ satisfies $A \leq_{\mathbf{L}} L$, and

16.3 Connectivity

Proposition

`STCON` is in **NL**.

Proof. The certificate is the path. ■

Theorem

`STCON` is **NL**-complete.

Proof. $\mathsf{STCON} \in \mathbf{NL}$.

Fix $A \in \mathbf{NL}$. Let V be a log-space verifier for A. Let G be the configuration graph for V for x, polynomial-size in $|x|$. G is log-space computable because every bit in its encoding is the presence/absence of an edge between two vertices $v, w \Leftrightarrow$ transition between two configurations. ∎

Theorem

$\mathbf{NL} = \mathbf{coNL}$.

Chapter 17

Sublogarithmic Space

17.1 Loglog Space Complexity

Counting is a basic problem that can be accomplished with logarithmic space. Consider the language

$$\mathsf{Inc} = \{x01^k\#x10^k : x \in \{0,1\}^*, k \geq 0\}$$

Proposition

$\mathsf{Inc} \in \mathbf{L}$.

Proof. Check that the string is of the form $x\#y$, $x, y \in \{0,1\}^*$ with $O(1)$ space. Then keep pointers to check symbols of x and y from right to left to check they satisfy required format. ■

We can obtain one with smaller space complexity. Consider

$$\mathsf{Count} = \{0^k\#0^{k-1}1\#0^{k-2}10\#0^{k-2}11\#\cdots\#1^k : k \geq 1\}$$

Proposition

$\mathsf{Count} \in \mathbf{SPACE}(\log\log n)$.

Proof. Consider the first algorithm: Check that the binary string before the first $\#$ is 0^k for some $k \geq 1$. Use the Inc algorithm to check $x\#y$ satisfies $num(y) = num(x) + 1$ for all adjacent pairs of strings. Then we check that the last string is 1^k.

The first step takes $O(\log k)$ space. Using the increment algorithm is $O(\log k)$ space. Checking last string is 1^k is $O(\log k)$ bits. For any $x \in \mathsf{Count}$, $n = |x| \geq 2^k \cdot k > 2^k$. So $O(\log k) < O(\log\log n)$.

There is a problem. If $x = 0^k$, we have already used logarithmic space. The solution: First check that the string is of the form

$$z_0 0^\ell\#z_1 0^{\ell-1}1\#z_2 0^{\ell-2}10\#\cdots\#z_{2^\ell-1}1^\ell\#z_{2^\ell}$$

for $\ell = 1, 2, \ldots, k-1$. Then run the first algorithm. ■

17.2 Sub-loglog Space Complexity

Theorem

For every function $s : \mathbb{N} \to \mathbb{N}$ that satisfies $s = o(\log \log n)$,

$$\mathbf{SPACE}(s) = \mathbf{SPACE}(1)$$

Proof. Let M be a Turing machine with space cost $s = \omega(1)$. Fix $m \in \mathbb{N}$. Let x be a *shortest* string for which M has space cost m.

Let $n = |x|$. Consider any position $i = 1, 2, \ldots, n$ on the input tape. Every time the input tape head of M crosses between the ith and $(i+1)$th positions on the input tape, its configuration is defined by the internal state of M, the position of the work head tape, and the contents of the work tape. In total, there are

$$t = O(1) \cdot m \cdot |\Gamma|^m = 2^{O(m)}$$

such possible configurations. The same configuration can never be repeated more than twice (once left, once right) in any crossing sequence, otherwise M would be in an infinite loop.

We can list the configurations of M as it crosses between the ith and $(i+1)$th position on the input tape to obtain the crossing sequence at i of M on input x. The number of distinct crossing sequences is

$$\leq t^{2t} \leq 2^{m \cdot 2^{O(m)}} = 2^{2^{O(m)}}$$

Assume $n > 3t^{2t}$. Then there exists $i < j$ where M has the same crossing sequence on x at i and j and $x_i = x_j$. M behaves the same way on $y = x_1, \ldots, x_i, x_{j+1}, \ldots, x_n$. This implies that it has space cost m on y, contradicting the fact that x is the shortest string with cost m. ■

17.3 Constant Space Complexity

Definition: Two-Way Deterministic Finite Automaton (2DFA)

A Turing machine that has a read-only input tape but no work tape.

Lemma

Every language $L \in \mathbf{SPACE}(1)$ can be decided with a 2DFA.

Proof. Let M decide L with $\leq m$ space cost. There are at most $2^{O(m)}$ possible configurations of the work tape. By blowing up the finite automaton of M to have each state to become $2^{O(m)}$ states, a 2DFA can store and simulate M's work tape. ■

A DFA is a 2DFA where we add the restriction that the input tape head only moves right.

Lemma

Every language that can be decided by a 2DFA can also be decided by a DFA.

Definition: REGULAR

The set of languages that can be decided by DFAs.

Theorem

$\mathbf{REGULAR} = \mathbf{SPACE}(1)$.

Chapter 18

Nonregular Languages

18.1 Pumping Lemma

Definition: Deterministic Finite Automata (DFA)

A DFA M is defined by:

- a set $Q = \{1, \ldots, m\}$ of states
- a subset $F \subseteq Q$ of these states that are the accepting states
- a transition function $\delta : Q \times \{0, 1\} \to Q$ that indicate which state is reached given a state and the next symbol of input is read by M.

The transition diagram view of DFAs: every string s with $|s| > m$ (more than the number of states), then the walk on the transition diagram must include a cycle. We can write $s = xyz$ where y is the part of the string that causes this cycle. We can have many copies of y as well. Removing y, we can still get a string that gets to the accepting state.

Lemma (Pumping Lemma)

For every regular language L, there is a number p (pumping length) such that for any string $s \in L$ of length $|s| \geq p$, we can write $s = xyz$ where

- $|y| > 0$,
- $|xy| \leq p$,
- for each $k \geq 0$, $xy^k z \in L$.

Proof. Let M be a DFA that recognizes L. We choose $p = m$, the number of states in M.

Fix any $s \in L$ of length $|s| \geq p$. Let $q_0, \ldots, q_p \in Q$ be the states of M that we reach after reading the first i symbols of s for $i = 0, 1, \ldots, p$.

By the pigeonhole principle, there exists $i < j$ such that $q_i = q_j$ (p states but $p+1$ visited states). Define $x = s_1 s_2 \cdots s_i$ and $y = s_{i+1} \cdots s_j$ and $z = s_{j+1} \cdots s_{|s|}$.

Then the properties are satisfied below:

1. $|y| \geq 1$ because $i < j \implies s_{i+1}$ is in y.
2. $xy = s_1 \cdots s_j$, $j \leq p$.
3. For any k, $xy^k z$ ends in same state as $xyz = s$. Since $s \in L$, this is an accepting state.

■

18.2 Using the Pumping Lemma

Lemma (Pumping Lemma – Contrapositive)

Let L be a language that satisfies the following property: For every $p \geq 1$, there exists a string $s \in L$ of length $|s| \geq p$ such that for every decomposition $s = xyz$ with $|y| > 0$ and $|xy| \leq p$, there exists a value $k \geq 0$ where $xy^k z \notin L$. Then L is not a regular language.

One of the 3 conditions of the pumping conditions must fail.

Proposition

The language $L = \{0^n 1^n : n \geq 0\}$ is not regular.

Proof. Fix any $p \geq 1$. Consider $s = 0^p 1^p$. Fix any decomposition $s = xyz$ where $|xy| \leq p$, $|y| \geq 1$. Then $y = 0^\ell$ for some $\ell \geq 1$. By then $xy^2 z = 0^{p+\ell} 1^p \notin L$. L is not regular. ■

Proposition

The language $L_< = \{0^m 1^n : n > m \geq 0\}$ is not regular.

Proof. Fix any $p \geq 1$. Consider $s = 0^p 1^{p+1}$. Then $s = xyz$, $|xy| \leq p$, $|y| \geq 1$, we know $y = 0^\ell, \ell \geq 1$. Then $xy^2 z = 0^{p+\ell} 1^{p+1} \notin L$. L is not regular. ■

Proposition

The language $L_> = \{0^m 1^n : m > n \geq 0\}$ is not regular.

Proof. Fix any $p \geq 1$ and consider $s = 0^p 1^{p-1}$. Take $y = 0^\ell, \ell \geq 1$. If we choose $k = 0$, then $xy^0 z = xz = 0^{p-\ell} 1^{p-1} \notin L$. L is not regular. ■

Proposition

The language $L = \{0^{2^n} : n \geq 0\}$ is not regular.

Proof. Fix any $p \geq 1$. Consider $s = 0^{2^p}$. We know that if $s = xyz$, $|xy| \leq p$, $|y| \geq 1$ and $y = 0^\ell$ for some $1 \leq \ell \leq p$. Then $xy^2z = 0^{2^p+\ell}$ is a string of length between $2^p + 1 > 2^p$ and $2^p + p < 2^p + 2^p = 2^{p+1}$, so $xy^2z \notin L$. L is not regular. ■

18.3 Myhill-Nerode Theorem

Definition: L-Equivalent

The strings $x, y \in \{0,1\}^*$ are L-equivalent for some language $L \subseteq \{0,1\}^*$, denoted $x \equiv_L y$, if for every string $z \in \{0,1\}^*$, we have that $xz \in L \Leftrightarrow yz \in L$.

Definition: Index of a Language

The minimum cardinality of a set $X \subseteq \{0,1\}^*$ such that every string $y \in \{0,1\}^*$ is L-equivalent to some string $x \in X$.

Theorem (Myhill-Nerode Theorem)

The language $L \subseteq \{0,1\}^*$ is regular if and only if it has finite index.

When L is regular, then its index is exactly the minimum number of states of any DFA that decides L.

Chapter 19

Communication Complexity

19.1 Definitions

Communication complexity is a powerful tool to prove lower bounds or impossibility results in a variety of computational models.

Definition: Two-Player Communication Complexity

Two players, Alice and Bob, where Alice receives $x \in \{0,1\}^n$ and Bob receives $y \in \{0,1\}^n$, and work together to compute $f(x, y)$.

We can write f as a matrix, with Alice's input as rows and possible inputs to Bob as columns.

Definition: Communication Protocol

A protocol to determine who sends the next bit, and for which of their inputs the speaker will send the bit 1 (the other inputs 0). It also determines when to stop exchanging bits and to output $f(x, y)$.

We represent the communication protocol as a rooted binary tree. Each internal node of the tree is labelled with A or B, according to who sends the next bit, and the subset of $\{0,1\}^n$ for which the player sends the bit 1.

Leaves are labelled with values 0 and 1, representing the output of the protocol. A protocol computes the function f if and only if the protocol leads Alice and Bob to a leaf labelled with $f(x, y)$ for all possible inputs x and y.

Definition: Cost

The cost of a communication protocol is the depth of the tree, the maximum number of bits that Alice and Bob exchange before outputting $f(x, y)$.

Definition: Communication Complexity

The minimum cost of a communication protocol that computes $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, denoted $D^{cc}(f)$.

Proposition

Every function f has communication complexity bounded above by $D^{cc}(f) \leq n+1$.

Proof. We can design it by letting Alice send all n bits of the input x and let Bob compute $f(x,y)$ using the information he has. ■

Proposition

The parity function $\oplus_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ defined by

$$\oplus_n(x,y) = |\{i \in [n] : x_i \neq y_i\}| \pmod 2$$

has communication complexity $D^{cc}(\oplus_n) = 2$.

Proof. Note that $x_i + y_i$ is odd if and only if $x_i \neq y_i$, so we can rewrite $\oplus_n$ as

$$\oplus_n(x,y) = \sum_{i=1}^{n}(x_i + y_i) \pmod 2 = \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i \pmod 2$$

So here is a simple protocol to compute $\oplus_n$: Alice sends a bit corresponding to the parity of its input, then Bob does the same. The label of the leaf is the parity of the sum of the bits that they sent, and it will always equal $\oplus_n(x,y)$. ■

19.2 Fooling Set Method

Definition: Fooling Set

A set $\mathcal{S} \subseteq \{0,1\}^n \times \{0,1\}^n$ is a fooling set for $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ if

- Every $(x,y) \in \mathcal{S}$ satisfies $f(x,y) = 1$, and
- For every $(x,y) \neq (x',y') \in \mathcal{S}$, we have $f(x,y') = 0$ or $f(x',y) = 0$.

Lemma (Fooling Set Lemma)

If there is a fooling set $\mathcal{S}$ of cardinality $|S| = m$ for the function f, then $D^{cc}(f) \geq \log m$.

Proof. Assume for contradiction that there is a protocol of depth $d < \log m$ that computes f. Then the tree that represents the protocol has at most $2^d < m$ leaves. So by the pigeonhole principle, there must exist two distinct elements $(x,y) \neq (x',y') \in \mathcal{S}$ that both end up in the same leaf of this protocol tree.

Then the inputs (x, y') and (x', y) also must lead to the same leaf of the protocol because Alice and Bob follow the same path when Alice has either x or x' and Bob has either y or y'. But then there is no label that we can to that leaf so that it is consistent with all four of those inputs, so this is a contradiction the claim that the protocol computes f. ■

Theorem

The equality function $\mathsf{EQ} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ has communication complexity $D^{cc}(\mathsf{EQ}) \geq n$.

Proof. Consider the set

$$\mathcal{S} = \{(x, x) : x \in \{0,1\}^*\}$$

The set $\mathcal{S}$ is a fooling set because for any two distinct strings $x \neq x' \in \{0,1\}^n$, we have $\mathsf{EQ}(x, x) = \mathsf{EQ}(x', x') = 1$ but $\mathsf{EQ}(x, x') = \mathsf{EQ}(x', x) = 0$. So by Fooling Set lemma,

$$D^{cc}(\mathsf{EQ}) \geq \log|\mathcal{S}| = \log(2^n) = n$$

■

19.3 Application

Theorem

For every $c < 2$, $L_{pal} = \{x \in \{0,1\}^* : x = x^R\} \notin \mathbf{TIME}(n^c)$.

Proof. We use a reduction proof. Let T be a Turing machine that runs in time $O(n^c)$ and decides L_{pal}. T can be used to obtain a protocol that computes EQ with communication cost $O(n^c)$.

Alice and Bob defines a protocol in which they jointly simulate T. First, Alice simulates the machine on input $x0^nx^R$. Let $i \in [n]$ be the index for which T visits the cell $n + i$ the least number of times during its execution. The time complexity of T guarantees that this cell $n + i$ is visited $O(n^{c-1})$ times during T's execution on this input.

Bob similarly runs T on input $y0^ny^R$ to obtain the index $j \in [n]$ for which cell $n+j$ is visited the least number of times during T's execution.

Alice and Bob exchange the values of i and j. If these values are different, then it must be that $x \neq y$ and so the protocol ends with value 0. Otherwise, Alice and Bob jointly simulate T in the following way.

Alice begins the simulation, keeping track of the symbols on the tape all the way to cell $n + i$. She continues the simulation of T until it goes right to cell $n + i + 1$. At this point, she sends Bob the current state that T is in. Then Bob continues the simulation, keeping track of the contents of the tape from cell $n + i + 1$ and to the right. When T next goes left to cell $n + i$, he sends Alice the current state of T and she continues the simulation.

In the end, when T halts, the communication protocol terminates with the value 1 if T accepted and 0 otherwise.

This protocol does compute EQ correctly, since it outputs 1 if and only if T accepts if and only if $x = y$.

The protocol sends $O(\log n)$ bits to check that the indices i and j matched. Then each simulation transfer requires only $O(1)$ bits of communication and there are at most $O(n^{c-1})$ such transfers. So the overall communication cost of the protocol is $O(\log n) + O(n^{c-1}) = O(n^{c-1})$. From the lower bound, we already established the communication complexity of EQ, we have

$$n + 1 \leq D^{cc}(\mathsf{EQ}) \leq O(n^{c-1})$$

so $c \geq 2$. ■

Theorem

For any $c < 2$, $\mathsf{PAL} \notin \mathbf{TIME}(n^c)$.

Proof. Let M be a Turing machine that decides PAL in time $O(n^c)$. Alice and Bob can use it to compute EQ. They can simulate M on $x0^n y^r$. This is a palindrome if $x = y$.

Alice and Bob each have half of the tape and simulate on their sides. When the Turing machine cross into the other half, Alice sends a message to Bob with the state of the Turing machine is in when it crosses over. Each message is a constant number of bits. Therefore, the cost of the protocol is $O(1)\times$ number of crossings.

We also know that

$$n + 1 \leq D^{cc}(\mathsf{EQ}) \leq O(n^c)$$

Weak claim: When M has time cost $O(n^c)$ for any $i \in [n]$, it transitions between i and $i+1$ at most $O(n^c)$ times.

Stronger claim: For every input $x0^n y^r$, there is an index $i^* \in [n]$ where M crosses between i^* and $i^* + 1$ between 0-cells at most $O(n^{c-1})$ times.

Protocol:

1. Alice simulates M on $x0^n x^r$ and identifies i_a^* that crosses least often.
2. Bob simulates M on $y0^n y^r$ and identifies i_b^*.
3. Alice and Bob exchanges i_a^*, i_b^* which takes $O(\log n)$ bits.
4. If $i_a^* \neq i_b^*$, then terminate since x, y cannot have been the same.
5. If $i_a^* = i_b^* = i^*$, then simulate with i^* as the index of splitting.
   Alice and Bob simulate M with i^* as the hand off point. If M terminates with $O(n^{c-1})$ transitions across hand-off, output what M outputs, else output 0.

Therefore,

$$n + 1 \leq D^{cc}(\mathsf{EQ}) \leq O(n^{c-1})$$

which gives $c \geq 1$. ■

Chapter 20

Probabilistic Proof Systems

Theorem

$\mathbf{IP} = \mathbf{PSPACE}$.

Proof. (Idea) $\mathsf{TQBF} \in \mathbf{IP}$.

$$\exists x_1 \forall x_2 \exists x_3 \cdots \phi(x_1, \ldots, x_n) = 1$$

We can rewrite ϕ as a polynomial where $\wedge$ is $\otimes$, $\vee$ is $\oplus$, and $\neg$ is $1\ominus$. ∎

20.1 Multiple Interactive Proof

This is similar to having multiple Merlins.

Definition: $r(n), q(n)$-Probabilistic Verifier

A probabilistic verifier that uses $r(n)$ bits of randomness and queries $q(n)$ bits of the certificate and runs in polynomial time.

Definition: $\mathbf{PCP}(r(n), q(n))$

The class of languages that have $r(n), q(n)$-probabilistic verifiers.

Theorem

$$\mathbf{MIP} = \mathbf{PCP}(\mathit{poly}(n), \mathit{poly}(n)) = \mathbf{NEXP}$$

Theorem PCP Theorem

$$\mathbf{NP} = \mathbf{PCP}(\log n, 3)$$

Chapter 21

Approximation Algorithms

With optimization problems, you do not always have to find the optimal solution, but a very good solution.

Theorem

There is a polynomial-time algorithm that gives you a $\frac{7}{8}$-approximation for $\mathsf{Max3SAT}$.

Proof. Pick a random assignment. ■

Theorem

There is a $\frac{1}{2}$-approximation to $\mathsf{MinVertexCover}$ can be done in polynomial-time.

Proof. Start with $C = \emptyset$. For each $(v, w) \in E(G)$, if $v \notin C$ and $w \notin C$, then $C = C \cup \{v, w\}$. ■

Theorem

$\mathsf{MinKnapsack}$ has a $(1 - \epsilon)$-approximation for any $\epsilon > 0$.

This is a polynomial-time approximation scheme (PTAS).

Theorem

$\mathsf{MaxClique}$ has no known approximation.