Modern Cryptography Notes

1 Preliminaries

2 Sets

We start with one of the simplest notions in mathematics, *sets*, which is a collection of *distinct* objects. Our first example is the infinite set of natural numbers,

$$\mathbb{N} := \{1, 2, 3, \cdots\}.$$

This also gives us the opportunity to establish our first notation of ":=", which we will use as the assignment operator. This is to be contrasted with "=", which is the operator used to establish equality.

The most common set we shall encounter in the course is the finite set $\{0,1\}$. This will often be referred to as an *alphabet* as we shall use it to build strings

$$\{0,1\}^2 := \{00, 01, 10, 11\}$$
$$\{0,1\}^* := \{\epsilon, 0, 1, 00, 01, 10, 11, 000 \cdots\}$$

Size. We shall restrict our discussion of size to *finite sets.* We denote by $|S|$ the size of the set S, defined to be the number of elements in the set.

Example 1 *Let the set S be defined as follows:*

$$S := \{1, 01, 10, 100, 010, 001\}.$$

Then,

$$|S| = 6$$

Note that by definition, a set only consists of *distinct* elements, and thereby it suffices to define the size simply as the number of elements.

Membership. For an element x, we shall indicate by $x \in S$ if x is in the set S. If not, we shall indicate it by $x \notin S$.

2.1 Operations

Below we describe the common set operations:

Union. The union of two sets A and B, denoted by $A \cup B$ is defined as

$$A \cup B := \{x \mid x \in A \text{ OR } x \in B\},$$

to be the set of elements that are present in either A or B.

The above shorthand notation is called the *set-builder notation*, that describes the properties an element x must satisfy to be a part of the set $A \cup B$. We shall use the *set-builder notation* frequently in this course.

Intersection The intersection of two sets A and B, denoted by $A \cap B$ is defined as

$$A \cap B := \{x \mid x \in A \text{ AND } x \in B\},$$

to be the elements present only in both A and B.

Difference The set difference of A and B, denoted by $A \setminus B$, is defined as

$$A \setminus B := \{x \mid x \in A \text{ AND } x \notin B\},$$

to be the elements present in A but not in B.

Note that the for set difference, the order of the two sets matter, and $A \setminus B \neq B \setminus A$. Think of a simple example to illustrate this.

Complement The complement of a set A is defined with respect to the *universe*, denoted by U. The universe denotes all possible elements which can be used to construct the set A. The complement of A, denoted by $\overline{A}$, is defined as

$$\overline{A} := \{x \mid x \notin A\},$$

to be the elements not present in A.

These four common set operations are illustrated in Figures 1,2, 3 and 4.

For our context, it is easiest of think of the universe to be $\{0,1\}^*$, the set of all binary strings.

Cartesian product. The cartesian product of sets A and B, denoted by $A \times B$ is defined as

$$A \times B := \{(a, b) \mid a \in A \text{ AND } b \in B\}$$

is the set of all *ordered pairs* (a, b) when $a \in A$ and $b \in B$. To differentiate unordered sets from an ordered tuple, we will represented ordered tuples by brackets $(\cdot)$.

Since we are talking about ordered tuples, $A \times B$ and $B \times A$ give rise to different sets since $(a, b) \neq (b, a)$ when a and b are distinct.

Subsets. For sets A and B, A is a subset of B, denoted by $A \subseteq B$, if every element of A is also an element of B.

Two sets A and B are equal if *both* $A \subseteq B$ and $B \subseteq A$. When arguing two sets are equal, this is the go-to method for a proof.

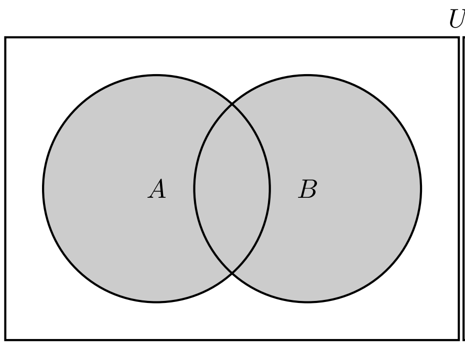

Figure 1: $A \cup B$

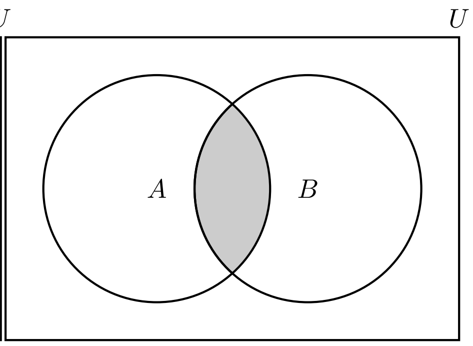

Figure 2: $A \cap B$

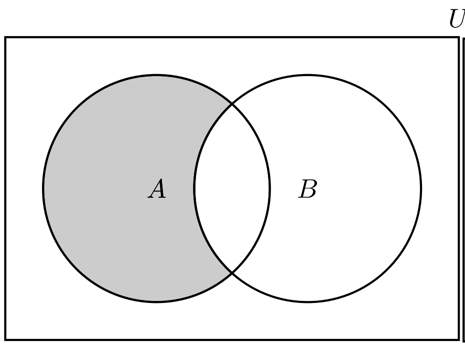

Figure 3: $A \setminus B$

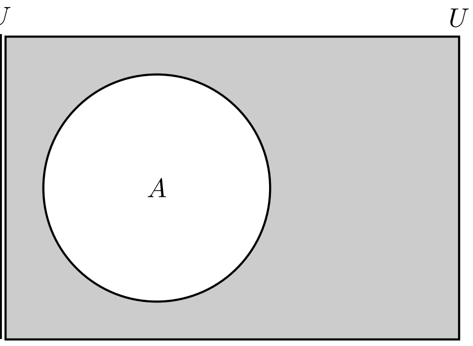

Figure 4: $\overline{A}$

3 Relations and Functions

3.1 Relation

A relation R from A to B is a subset of the Cartesian product $A \times B$. Consider the following relation R_1 from $A \coloneqq \{x_1, x_2, x_3, x_4\}$ to $B \coloneqq \{y_1, y_2, y_3, y_4\}$,

$$R_1 \coloneqq \{(x_1, y_2), (x_1, y_3), (x_2, y_4), (x_3, y_3), (x_4, y_1)\}$$

The *domain* of a relation is the subset of A that appear as the first component in a relation; and the *range* is the subset of B that appears as the second component in the relation. We shall typically consider relations where the domain is the entire set A. The set B is referred to as the co-domain of the relation.

3.2 Functions

A function $f : A \mapsto B$ is a relation from A to B with the additional restriction that each element in the domain A appears in exactly one ordered pair in the relation. The relation R_1 is not a function since x_1 appears in two ordered pairs. Consider R_2 over the same sets A and B.

$$R_s \coloneqq \{(x_1, y_2), (x_2, y_1), (x_3, y_3), (x_4, y_1)\}$$

The relations R_1 and R_2 are illustrated in Figures 5 and 6.

In function terminology, the elements from A are referred to as inputs, and the corresponding second element in the pair from B to the output. When we consider boolean functions, it is common to represent functions by a truth table, i.e. a row corresponding to each possible input and the corresponding output of the function.

Example 2 *What are the total number of boolean functions with n_1 inputs and n_2 outputs?*

Solution Let us consider first the simple case of a single output, and we shall see how to extend this to multiple outputs. As discussed above, every function can be represented by the corresponding truth table. For a function with n_1 inputs, any truth table will have 2^{n_1} rows.

For each row, since there is a single output, we can specify the output for that row to be either 0 or 1. Specifying such a value for each row determines the function. The total number of ways one can fill in this column then determines the number of functions. Hence the total number of functions with a single bit of output is $2^{2^{n_1}}$.

x_1	x_2	$\cdots$	x_{n_1}	y
0	0	$\cdots$	0	r_1
0	0	$\cdots$	1	r_2
		$\vdots$		
1	1	$\cdots$	0	r_{N-1}
1	1	$\cdots$	1	r_N

Now let us extend this to multiple output bits. The first important point to note is that the number of rows in the truth table remains unchanged.

x_1	x_2	$\cdots$	x_{n_1}	y_1	$\cdots$	y_{n_2}
0	0	$\cdots$	0	$r_{1,1}$	$\cdots$	$r_{n_2,1}$
0	0	$\cdots$	1	$r_{1,2}$	$\cdots$	$r_{n_2,2}$
			$\vdots$			
1	1	$\cdots$	0	$r_{1,N-1}$	$\cdots$	$r_{n_2,N-1}$
1	1	$\cdots$	1	$r_{1,N}$	$\cdots$	$r_{n_2,N}$

$$\underbrace{2^{2^{n_1}} \times \cdots \times 2^{2^{n_1}}}_{n_2 \text{times}} = 2^{n_2 2^{n_1}}$$

□

We now describe some special cases of functions below.

Injective Function. *Injective* functions or 1-1 functions are functions where each input has a distinct output, i.e. there does not exist $x, x' \in A$ such that $f(x) = f(x')$. This immediately requires $|B| \geq |A|$.

Surjective Function. A function is *surjective* if the co-domain B is the same as the range, i.e. every element of B is an output of the function for some element of A. This requires $|A| \geq |B|$.

Bijective Function. A function that is both *injective* and *surjective* is called a *bijective* function.

Permutation. A permutation on a set A is defined to be a *bijection* from A to itself. It is common to denote a permutation as $\Pi : A \mapsto A$.

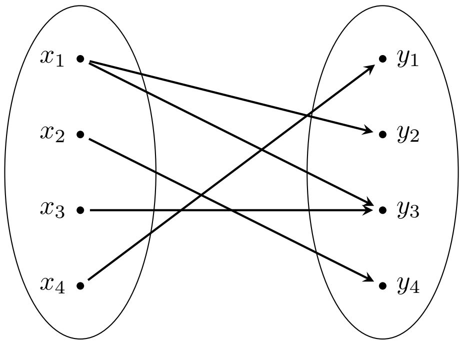

Figure 5: An example relation R_1

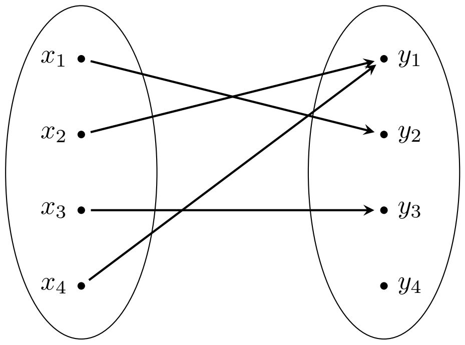

Figure 6: An example function R_2

3.3 Logical Operations

We describe below the four most common logical/boolean operations. While the AND, OR and XOR gate have been described with two inputs, they can easily be extended to multiple inputs.

AND. AND gate

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

OR. OR gate

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

XOR. XOR gate

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

NOT.

x	$\overline{x}$ (or $\neg x$)
0	1
1	0

3.4 Quantification.

We will often want to make a claim about a variable by either claiming a statement holds for all values that the variable takes, or there is at least one value satisfying the statement. The following are largely taken from the notes in [Lov], and you should take a look there for further details.

Universal Quantification.

$$\forall x \in A,\ P(x)$$

which indicates that for all x in the set A, the statement $P(x)$ is true.

Existential Quantification.

$$\exists x \in A,\ P(x)$$

which indicates that there is at least one x in A such that the statement $P(x)$ is true.

Nesting Quantifiers. We can nest the above quantifiers in an arbitrary manner. For instance, the following is a valid nesting

$$\forall x_1 \in A_1, \forall x_2 \in A_2, \exists x_3 \in A_3, \forall x_4 \in A_4\ P(x_1, x_2, x_3, x_4).$$

Given the fact that the nesting can be arbitrary, it is important to ask if there order of quantification matters. If the quantifiers are the same, then the order of nesting does not matter. Therefore, $\forall x_1 \in A_1, \forall x_2 \in A_2\ P(x_1, x_2)$ and $\forall x_2 \in A_2, \forall x_1 \in A_1\ P(x_1, x_2)$ are equivalent, and the same is true for the existential quantifiers. But things aren't as simple when the quantifiers are different.

Remark 1 (Order of Quantification) *When the nested quantifiers are different, the order of quantifiers matters. We illustrate this with an example below.*

Example 3 *Consider the two following statements*

1. $\forall x \in \mathbb{Z}, \exists y \in \mathbb{Z}$ *s.t.* $x + y = 4$
2. $\exists x \in \mathbb{Z},$ *s.t.* $\forall y \in \mathbb{Z}\ x + y = 4$

The first statement is true, since for every fixed x, we can set y to be $4 - x$. Thus existence of such a $y \in \mathbb{Z}$ is guaranteed. On the other hand, the second statement says that there must be a single x such that for every value of $y \in \mathbb{Z}$, $x + y = 4$. It is clear to see that this statement cannot be true.

In fact, the second ordering is a stronger claim than the first.

Negation of Quantifiers. When negating a quantified statement, negate all the quantifiers first, from left to right (keeping the same order), then negate the statement. By negating quantifiers we mean swapping $\forall$ and $\exists$. Below are few examples of negation

1. $\neg\left(\forall x \in A,\ P(x)\right) \iff \exists x \in A,\ \neg P(x)$
2. $\neg\left(\exists x \in A,\ P(x)\right) \iff \forall x \in A,\ \neg P(x)$
3. $\neg\left(\exists x \in A, \forall y \in B,\ P(x,y)\right) \iff \forall x \in A, \exists y \in B,\ \neg P(x,y)$
4. $\neg\left(\forall x \in A, \exists y \in B,\ P(x,y)\right) \iff \exists x \in A, \forall y \in B,\ \neg P(x,y)$

Here, $\iff$ is the notation for *if and only if* that connects two statements, where either both statements are true or both are false.

4 Reductio ad Absurdum

Reduction to absurdity, or proof by contradiction is a common proof technique that we shall employ throughout this course. We start with the statement we want to prove, and then assume that the statement is false, and then derive as a consequence a statement we believe to be false.

The following is an example taken from Boaz Barak's lecture notes:

Theorem 1 *There are infinitely many primes.*

Proof Let us assume for that the above statement is indeed false and there are a finite number of primes. Let us denote the primes by $p_1, \cdots, p_N$ with N indicating the total number of primes.

Consider the following number,

$$P = p_1 \cdot p_2 \cdots p_N + 1,$$

i.e. P is one greater than the product of all the primes. It is clear that none of the primes, $p_1, \cdots, p_N$ divide P, since the remainder with respect to all of them is 1.

Therefore, P has only two factors 1 and P, establishing it as a prime. But P is clearly not in the set of finite primes, therefore a contradiction to our assumption of a finite set of primes.

Since the same argument can be applied to any set of finite primes, it must be the case that the number of primes are infinite. $\square$

Another common example of this type of proof technique is to prove that $\sqrt{2}$ cannot be expressed as an irreducible fraction, thereby establish that it is irrational.

5 Probability

Throughout this course, we shall see the importance of randomness, in that it permeates every aspect of cryptography. In this section, we shall review some of the relevant material for discrete probability.

Sample Space: Sample space S of a probabilistic experiment is the set of all possible outcomes of the experiment. A couple points of note:

- We will only consider finite sample spaces
- In most cases, the sample space will be the set $\{0,1\}^k$ of size 2^k

Probability Distribution: With the sample space, we now define a distribution which assigns probabilities to this sample space.

Definition 1 (Distribution) *X is a* ***distribution*** *over a sample space $\mathcal{S}$ if it assigns a probability p_s to the element $s \in \mathcal{S}$ such that*

$$\sum_{s \in \mathcal{S}} p_s = 1.$$

A common distribution is a *uniform distribution* where each element in the sample space is assigned the same probability $\frac{1}{|S|}$. For example, when the sample space is the set $\{0,1\}^k$, this distribution is denoted by U_k with each k-bit string assigned probability $\frac{1}{2^k}$.

Sampling From Distribution: We say we sample an element x from the distribution X if each element in S is picked proportional to the probability defined by the distribution X. We denote this by $x \leftarrow\$ X$. For uniform distribution, we shall find it convenient to denote sampling from the uniform distribution by $x \leftarrow\$ \{0,1\}^k$ to indicate sampling uniformly from the set $\{0,1\}^k$.

Event: Event is a subset of the sample space. Let $E \subset S$ be an event, the probability of the event E, denoted by $\Pr[E]$, is defined as

$$\Pr[E] := \sum_{s \in E} p_s$$

Union Bound: If S is a sample space and $A, A' \subseteq S$, then the probability that either A or A' occurs is $\Pr[A \cup A'] \leq \Pr[A] + \Pr[A']$

Random Variables: A random variable is a function that maps elements of the sample space to another set. It assigns values to each of the experiment's outcomes.

Example 4 *In the uniform distribution* $\{0,1\}^3$*, Let Random Variable* N *denote the number of* $1s$ *in the chosen string, i.e., for every* $x \in \{0,1\}^3$*,* $N(x)$ *is the number of* $1s$ *in* x*.*

$$\Pr_{x \leftarrow_\$ \{0,1\}^3}[N(x) = 2] = \frac{3}{8}$$

Expectation: The expectation of a random variable is its weighted average, where the average is weighted according to the probability measure on the sample space. The expectation of random variable N is defined as:

$$\mathbb{E}[N] = \sum_{x \in S} N(x) \cdot \Pr_{y \leftarrow_\$ S}[y = x] \tag{1}$$

Here S is the sample space and $\Pr_{y \leftarrow_\$ S}[y = x]$ is the probability of obtaining x when sampling from S.

Example 5 *If* N *is defined as in the above example,*

$$\mathbb{E}[N] = 0 \cdot \frac{1}{8} + 1 \cdot \frac{3}{8} + 2 \cdot \frac{3}{8} + 3 \cdot \frac{1}{8} = 1.5$$

Expectation is a linear function, i.e.,

$$\mathbb{E}[N + M] = \mathbb{E}[N] + \mathbb{E}[M]$$

Variance: Variance of a random variable N is defined as the expectation of the square of the distance of N from its expectation.

$$\mathsf{Var}[N] = \mathbb{E}\left[(N - \mathbb{E}[N])^2\right] \tag{2}$$

If N is defined as in the first example,

$$\begin{aligned}\mathsf{Var}[N] &= (0-1.5)^2 \cdot \frac{1}{8} + (1-1.5)^2 \cdot \frac{3}{8} + (2-1.5)^2 \cdot \frac{3}{8} + (3-1.5)^2 \cdot \frac{1}{8} \\ &= 0.75\end{aligned}$$

Variance is a measure of how spread out the values in a distribution are. A low variance means the outcomes will usually be very close to one another.

Standard Deviation: Standard deviation of N is the square root of $\mathsf{Var}[N]$

Conditional Probability: The conditional probability of event B in relation to event A is the probability of event B occurring when we know that A has already occurred.

$$\boxed{\Pr[B \mid A] = \frac{\Pr[A \cap B]}{\Pr[A]} \qquad (3)}$$

Example 6 *Drawing Kings from a deck of cards. Event A is drawing a King first, and Event B is drawing a King second.*

$$\Pr[A] = \frac{4}{52}$$

$$\Pr[B|A] = \frac{3}{51}$$

Law of Total Probability: Throughout the course we will constantly use the law of total probability with regards to conditional probability.

Let $B_1, B_2, \cdots, B_n$ be a disjoint partition of a sample space S (i.e. $\forall i, j$ $B_i \cap B_j = \emptyset$). Then for any event E,

$$\boxed{\Pr[E] = \sum_{i=1}^{n} \Pr[E \cap B_i] = \sum_{i=1}^{n} \Pr[E \mid B_i] \Pr[B_i]}$$

Let's look at a very simple example.

Example 7 *Consider any event E. Let us sample a single bit b uniformly at random. B_1 is the event that $b = 0$, and B_2 the event that $b = 1$. We can write, the probability of event E as*

$$\begin{aligned}\Pr[E] &= \Pr[E \mid b=0] \Pr_{b \leftarrow \$ \{0,1\}}[b=0] + \Pr[E \mid b=1] \Pr_{b \leftarrow \$ \{0,1\}}[b=1] \\ &= \Pr[E \mid b=0] \cdot \frac{1}{2} + \Pr[E \mid b=1] \cdot \frac{1}{2} \\ &= \frac{1}{2} \cdot (\Pr[E \mid b=0] + \Pr[E \mid b=1])\end{aligned}$$

Independent Events: We say that B is independent from A if $\Pr[B \mid A] = \Pr[B]$, i.e.,

$$\Pr[A \cap B] = \Pr[A] \cdot \Pr[B]$$

Example 8 *Tossing a coin. The probability that heads shows up on two consecutive coin tosses,*

$$\Pr[HH] = \Pr[H]. \Pr[H] = \frac{1}{2}.\frac{1}{2} = 0.25$$

Each toss of a coin is an Independent Event.

Now let's look at example where the individual samples are random, but the joint distributions are not.

Example 9 *Consider the following random variables X and Y both taking values in $\{0,1\}$ with the following joint distribution:*

$$\Pr[X = 0 \wedge Y = 0] = \frac{3}{16}, \quad \Pr[X = 0 \wedge Y = 1] = \frac{5}{16}$$
$$\Pr[X = 1 \wedge Y = 0] = \frac{5}{16}, \quad \Pr[X = 1 \wedge Y = 1] = \frac{3}{16}$$

Using the law of total probability, we have

$$\Pr[X = 0] = \Pr[X = 1] = \frac{1}{2}$$
$$\Pr[Y = 0] = \Pr[Y = 1] = \frac{1}{2}$$

but for each $x, y \in \{0,1\}$ we have

$$\Pr[X = a \wedge Y = b] \neq \Pr[X = a] \cdot \Pr[X = b].$$

Note that we're using the symbol $\wedge$ indicating 'AND' to indicate independence while we've previously spoken about the set intersection $\cap$ when describing independence. These two symbols will be used to convey the same meaning; while $\cap$ will be used exclusively for sets, $\wedge$ will be used when we talk about random variables.

Pairwise Independent Random Variables: Let $X_1, X_2......X_n$ be random variables. We say that the X_i's are pairwise-independent if for every $i \neq j$ and all a and b, it holds that:

$$\Pr[X_i = a \wedge X_j = b] = \Pr[X_i = a] \cdot \Pr[X_j = a] \tag{4}$$

Example 10 *We throw two dice. Let (i) A be the event "the sum of the points is 7"; (ii) B the event "die #1 came up 3"; and (iii) C the event "die #2 came up 4".*

$$\Pr[A] = \Pr[B] = \Pr[C] = \frac{1}{6}$$

$$\Pr[A \cap B] = \Pr[B \cap C] = \Pr[A \cap C] = \frac{1}{36}$$

But,

$$\Pr[A \cap B \cap C] = \frac{1}{36} \neq \Pr[A] \cdot \Pr[B] \cdot \Pr[C]$$

A, B and C are pairwise independent but not independent as a triplet.

6 Model of Computation - Turing Machines

Throughout this course, it is important to understand the model of computation that we shall work with. This will be useful when we want to model various participants in the primitives and protocols we develop.

In this course, we shall limit our model of computation to that of Turing Machines, which we describe informally below. While it is not important to understand the workings of a Turing Machine, it is presented here for completeness. The below description is taken verbatim from [Kat].

A Turing Machine is defined by an integer $k \geq 1$, a finite set of states Q, an alphabet Γ, a transition function $\delta : Q \times \Gamma^k \mapsto Q \times \Gamma^{k-1} \times \{\mathsf{L}, \mathsf{S}, \mathsf{R}\}^k$ where:

- k is the number of infinite, one-dimensional tapes used by the machine. We typically have $k \geq 3$, where the first tape is denoted as the the *input tape*, and the last tape the *output tape*. The position of the tape currently being read is specified by a separate tape head for each tape.
- the set of states Q contains two special states: (a) the start state q_{start}; and (b) the halt state q_{halt}.
- Γ contains $\{0, 1\}$, a special "blank symbol", and a special "start symbol".

The computation of a Turing machine M on input $x \in \{0, 1\}^*$ proceeds as follows: All tapes of the Turing machine contain the start symbol followed by the blank symbols, with the exception of the input tape which contains the input x. The machine starts in state $q = q_{\mathsf{start}}$ with its k heads at the leftmost position of each tape. Then, until q is the halt state, repeat the following:

1. Let the current contents of the cells being scanned by the k heads be $\gamma_1, \cdots, \gamma_k \in \Gamma$.
2. Compute $\delta(q, \gamma_1, \cdots, \gamma_k) = (q', \gamma'_2, \cdots, \gamma'_k, D_1, \cdots, D_k)$ where $q' \in Q$, $\gamma'_2, \cdots, \gamma'_k \in \Gamma$ and $D_i \in \{\mathsf{L}, \mathsf{S}, \mathsf{R}\}$.
3. Overwrite the contents of the currently scanned cell on tape i to γ'_i for $2 \leq i \leq k$; move head i to the left, to the same position, or to the right depending on whether $D_i = \mathsf{L}$, S, or R, respectively; and then set the current state to $q = q'$.

The output of M on input x, denoted $\mathsf{M}(x)$, is the binary string contained on the output tape when the machine halts. It is possible that M never halts for certain inputs x.

The above description is that of a **Deterministic Turing Machine**. We contrast this with a **Non-Deterministic Turing Machine** (NDTM). An NDTM instead of having a single transition function δ, has *two* transition function δ_0, δ_1 and at each step decides in a non-deterministic/arbitrary manner. Therefore, after n steps the machine can be in at most 2^n configurations, where a configuration is the joint description of all the tapes of the Turing Machine.

We fill often find it more convenient to talk about algorithms, defined below.

Definition 2 (Algorithm) *An algorithm is a Turing Machine whose input and output are strings over the binary alphabet* $\Sigma = \{0, 1\}$.

Note, that the two terms *Turing Machine* and *Algorithm* will be used interchangeably from now on.

Definition 3 (Running Time) *An algorithm A is said to run in time $T(n)$ if for all strings of length n over the input alphabet ($x \in \{0, 1\}^n$), $A(x)$ halts within $T(|x|)$ steps.*

In this course, we will primarily focus on algorithms that have a *polynomial* running time.

Definition 4 (Polynomial Running Time) *An algorithm A is said to run in polynomial time if there exists a constant c such that A runs in time $T(n) = n^c$.*

We say an algorithm is *efficient* if it runs in polynomial time. Even for efficient algorithms, the constant c can be a large value. For example, consider c=100. In practice, n^{100} may actually be considered "inefficient". For our purposes, however, we will stick with this definition of efficiency.

If an algorithm runs exponential or super-polynomial time, i.e., $T(n) = 2^n$ or $T(n) = n^{(\log n)}$, then we will say it is *inefficient*.

One might consider other notions of efficiency, and there is nothing pristine about using polynomial time as the measure of efficiency. But throughout this course, we shall see that this is often a convenient measure since the composition of polynomials remain a polynomial.

So far, we have only considered deterministic algorithms. In computer science, and specifically cryptography, randomness plays a central role. Therefore, throughout the course, we will be interested in randomized (a.k.a. probabilistic) algorithms.

Definition 5 (Randomized Algorithm) *A randomized algorithm, also called a probabilistic polynomial time Turing machine (*PPT*) is a Turing machine that runs in polynomial time and is equipped with an* *extra randomness tape. Each bit of randomness tape is uniformly and independently chosen.The output of a randomized algorithm is a distribution.*

Remark 2 *Note that we do not place any limits on the length of the ranom tape, once the randmness has been fixed, the computation is completely determinisitic.*

When we talk about randomized algorithms, we will often make explicit the randomness used. So an algorithm $\mathsf{M}(x;r)$ indicates that M runs on input x using randomness r that is sampled as $r \leftarrow\!\!\$ \{0,1\}^m$ for some m. In fact, one can think of the output of $\mathsf{M}(x;r)$ as a random variable with randomness from r.

As mentioned earlier, in practice, everyone including the adversary has some bounded computational resources. These resources can be used in a variety of intelligent ways, but they are still limited. Turing machines are able to capture all the types of computations possible given these resources. Therefore, a adversary will be a computer program or algorithm modeled as a Turing machine.

This captures what we can do efficiently ourselves and can be described as a uniform PPT Turing machine. When it comes to adversaries, we will allow them to have some extra power. Instead of having only one algorithm that works for different input lengths, it can write down potentially a different algorithm for every input size. Each of them individually could be efficient. If that is the case, overall the adversary still runs in polynomial time.

Definition 6 (Non-Uniform PPT Machine) *A non-uniform probabilistic polynomial time Turing machine is a Turing machine A made up of a sequence of probabilistic machines* $\mathsf{M} = \{\mathsf{M}_1, \mathsf{M}_2, \cdots\}$ *for which there exists a polynomial* $\mathsf{p}(\cdot)$ *such that for every* $\mathsf{M}_i \in \mathsf{M}$*, the description size* $|\mathsf{M}_i|$ *and the running time of* M_i *are at most* $\mathsf{p}(i)$*. We write* $\mathsf{M}(x)$ *to denote the distribution obtained by running* $\mathsf{M}_{|x|}(x)$*.*

Our adversary will usually be a non-uniform probabilistic polynomial running time algorithm (n.u. PPT).

7 Asymptotic Notation

When describing the running time of algorithms, instead of describing the running as a complicated function in the size of the input, it is more convenient to describe the running time with the function that reflects its growth.

Big-O. The function $f(n)$ is $O(g(n))$ if $\exists$ constants c, n_0, such that $\forall n \geq n_0$

$$0 \leq f(n) \leq c \cdot g(n)$$

Big-Omega. The function $f(n)$ is $\Omega(g(n))$ if $\exists$ constants c, n_0, such that $\forall n \geq n_0$

$$0 \leq c \cdot g(n) \leq f(n)$$

Theta. The function $f(n)$ is $\theta(g(n))$ if $\exists$ constants c_1, c_2, n_0, such that $\forall n \geq n_0$

$$0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

We have illustrated these notions, and the importance of the constants c and n_0, in Figure 7.

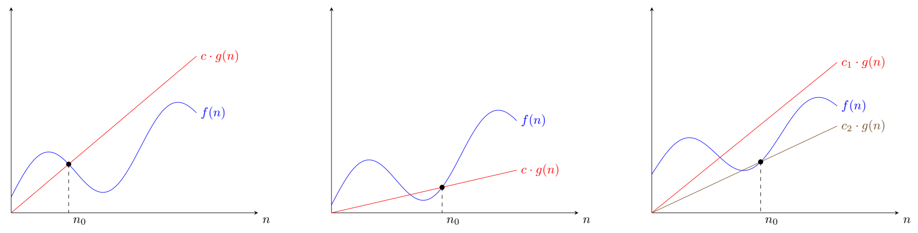

Figure 7: Here we give examples of the O, Ω and θ notations.

The above asymptotic notations of O and Ω may not be asymptotically tight, so below we describe their corresponding asymptotically tight notions.

Small-o. The function is $f(n)$ is $o(g(n))$ if $\forall$ constant c, $\exists$ constant n_0, such that $\forall$

$$0 \leq f(n) < c \cdot g(n)$$

Small-omega. The function is $f(n)$ is $\omega(g(n))$ if $\forall$ constant c, $\exists$ constant n_0, such that $\forall$

$$0 \leq c \cdot g(n) < f(n)$$

Note how the quantifiers, and their order, have changed in the above definition. Recall from our discussion of quantifiers how the order affects the definition.

Remark 3 *We have slightly abused notation above since $O(g(n))$ defines sets, but we shall find it convenient to use the above description. See Chapter 3 of [CLRS09] for a detailed discussion of these concepts.*

References

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009.

[Kat] Jonathan Katz. `http://www.cs.umd.edu/~jkatz/complexity/f11/all.pdf`.

[Lov] Andrew D Loveless. `https://sites.math.washington.edu/~aloveles/Math300Summer2011/m300Quantifiers.pdf`.