Theory of Computing

Xuzhe Xia

January 1, 2023

Contents

1 Automaton

Definition 1.

- Define the **alphabet** Σ as the set of symbols, usually take $\Sigma = \{0, 1\}$.
- Define a **string** as a finite sequence of symbols from Σ.
- Define $\Sigma^* = \bigcup_{i\geq 0} \Sigma^i$, where Σ^i denotes the set of strings with length i. Σ^* is the set of all string over the **alphabet** Σ.
- Define a **language** L as a subset of Σ^*.

Example.

- $\{0, 1\}^2 = \{00, 01, 10, 11\}$.
- $\{0, 1\}^0 = \{\epsilon\}$ with ϵ as an empty string.

1.1 Regular languages

1.1.1 Deterministic finite automaton

Definition 2 (Deterministic finite automaton).
A **deterministic finite automaton (DFA)** is a 5-tuple $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$,

1. Q is a non-empty finite set, whose elements are called **states**.
2. Σ is **alphabet**.
3. $\delta : Q \times \Sigma \to Q$ is a **transition function**.
4. $q_0 \in Q$ is a **start state**.
5. $F \subseteq Q$ is a set of **accept states**.

Example.

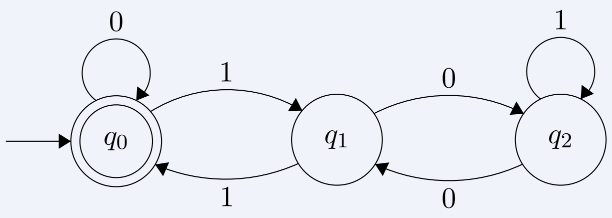

Figure 1.1: Example DFA

1. $Q = \{q_0, q_1, q_2\}$.
2. $\Sigma = \{0, 1\}$.
3. δ is described as:

	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

4. q_0 is the start state.
5. $F = \{q_0\}$.

Definition 3 (Extended transition function)**.** For convenience, we extend δ to $\bar{\delta} : Q \times \Sigma^* \to Q$ inductively as follows:

$$\begin{cases} \bar{\delta}(q, \epsilon) = q \\ \bar{\delta}(q, aw) = \bar{\delta}(\delta(q, a), w) \quad (a \in \Sigma, w \in \Sigma^*) \end{cases}$$

Definition 4.

- If $\bar{\delta}(q_0, w) \in F$, we say that w is **accepted** by $\mathcal{M}$.
- **The language accepted by** $\mathcal{M}$ is $L(\mathcal{M}) = \{w \in \Sigma^* : \bar{\delta}(q_0, w) \in F\}$, or we could say that the language accepted by $\mathcal{M}$ is the set of strings accepted by $\mathcal{M}$.
- If there is a DFA $\mathcal{M}$ accepts language L, we call L is **DFA-recognizable**.

1.1.2 Regular languages

Definition 5 (Regular language)**.** A language L is regular if L is DFA-recognizable.

Example.

- $L_1 = \{x \in \Sigma^* : x$ is the binary representation of multiplier of 3 or $\epsilon\}$ is regular.
- $L_2 = \{0^n 1^n : n \in \mathbb{N}\}$ is not regular.

Proof.

- L_1 is the language accepted by the DFA $\mathcal{M}$ described in Figure 1.1.
- Before proving L_2 is not regular, we introduce Pumping Lemma.

Lemma 1 (Pumping lemma for regular languages)**.** Let L be a regular language. Then there exists an integer $p \geq 1$ such that every string $w \in L$ of length at least p can be written as $w = xyz$ satisfying the following conditions:

- $|y| \geq 1$
- $|xy| \leq p$
- $\forall n \geq 0$, $xy^n z \in L$

Proof. For every regular language L there is a DFA accepts it. Let $p = |Q|$, the number of the states. For any string $w \in L$ of length at least p, then w can be written as xyz such that $\bar{\delta}(q_0, x) = \bar{\delta}(q_0, xy)$, where $|y| \geq 1$ and $|xy| \leq p$, as a consequence of pigeonhole principle. (Intuitively, it means that there are $|w|$ states plus one start state visited after transition function finishes, $|w| + 1 > p$, so there must be a repeat of states visited during the reading of the first p symbols).
Therefore, by induction we have $\forall n \geq 0$, $\bar{\delta}(q_0, xy^n) = \bar{\delta}(q_0, xy) \implies \bar{\delta}(q_0, xy^n z) = \bar{\delta}(q_0, xyz) \in F \implies xy^n z \in L$. $\square$

Corollary. $L_2 = \{0^n 1^n : n \in \mathbb{N}\}$ is not regular.

Proof. For any integer $p \geq 1$, there exists a string $w = 0^p 1^p$ such that for any x, y, z that satisfying $w = xyz$, $|y| \geq 1$, $|xy| \leq p$, both x and y has to be a sequence of 0, then let $n = p + 1$, $xy^n z$ contains $|xy^{p+1}| > p$ 0s in the front which is more than the number of 1s, so $xy^n z \notin L$, hence L_2 is not regular. $\square$

Remark. Note that we proved L_2 is not regular by using the contrapositive of pumping lemma in the form:
If $\forall p \in \mathbb{N}$, $\exists s \in A$ with $|s| \geq p$ s.t

$$\forall x, y, z \in \Sigma^*, (s = xyz \wedge |y| > 0 \wedge |xy| \leq p) \implies (\exists i \geq 0, \text{ s.t. } xy^i z \notin L)$$

, then A is not regular.

1.1.3 Nondeterministic finite automata

Definition 6 (Nondeterministic finite automata).
A **Nondeterministic finite automata (NFA)** is a 5-tuple $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$,

1. Q, Σ, q_0 and F are defined same as in DFA.
2. $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ is a transition function, where $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$

Remark.

- $\mathcal{P}(Q)$ is the power set of Q.
- We extended Σ with an empty string so that it allows NFA to move from one state to another without reading any symbols.

Definition 7 (Extended transition function). We extend δ to $\bar{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$ as follows:

$$\bar{\delta}(q, aw) = \bigcup_{p \in \delta(q,a)} \bar{\delta}(p, w) \qquad (a \in \Sigma, w \in \Sigma^*)$$

Similar to DFA, we also have the following definitions:

Definition 8.

- If $\bar{\delta}(q_0, w) \cap F \neq \varnothing$, we say that w is **accepted** by $\mathcal{M}$.
- **The language accepted by** $\mathcal{M}$ is $L(\mathcal{M}) = \{w \in \Sigma^* : \bar{\delta}(q_0, w) \cap F \neq \varnothing\}$.
- If there is a NFA $\mathcal{M}$ accepts language L, we call L is **NFA-recognizable**.

Theorem 1 (Equivalence of NFAs and DFAs). A language L is regular if and only if L is NFA-recognizable. In another word, every NFA has an equivalent DFA, and vise versa.

Proof. $\implies$: Any DFA is a NFA by definition.
$\impliedby$: For a NFA $\mathcal{M} = (Q, \Sigma_\epsilon, \delta, q_0, F)$, construct a DFA $M' = (Q', \Sigma, \delta', q_0', F')$ as follows:

1. $Q' = \mathcal{P}(Q)$.
2. $\delta(A, a) = \bigcup_{q \in A} \delta(q, a)$ with $A \in Q'$.
3. $q_0' = \{q_0\}$.
4. $F' = \{A \in Q' : A \cap F \neq \varnothing\}$.

Then $\bar{\delta}(q_0, w) = \bar{\delta}'(q_0', w)$, and thus $L(M) = L(M')$. $\square$

Then we can use the above theorem to prove the closure of regular language under regular operations $\cup$, $\circ$, and $*$:

Lemma 2. If $A, B \subseteq \Sigma^*$ are regular, so is

1. $A \cup B$.
2. $A \circ B = \{vw : v \in A, w \in B\}$.
3. $A^* = \{v_1 \circ v_2 \circ \cdots \circ v_n : v_i \in A, n \in \mathbb{N} \cup \{0\}\}$

Formal proof omitted, instead we show the idea of the proof:

Proof Idea. For any given $A = L(N_1)$ and $B = L(N_2)$,
To prove 1, we construct N as follows:

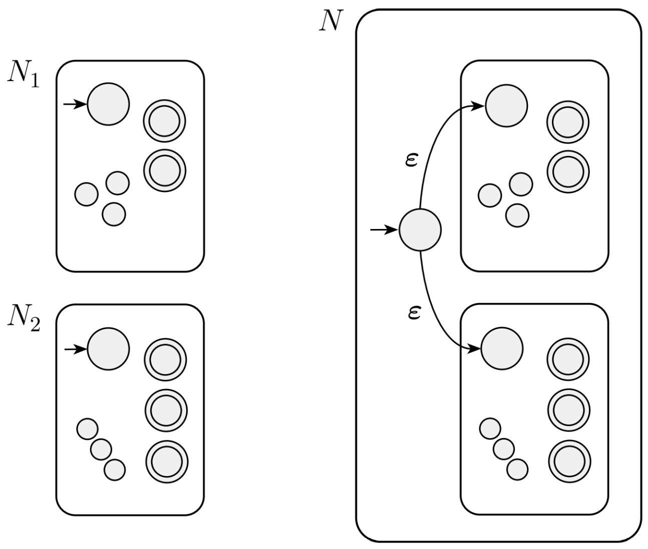

Figure 1.2: Construction of an NFA N that accepts $A \cup B$. (M. Sipser, 2012)

To prove 2, we construct N as follows:

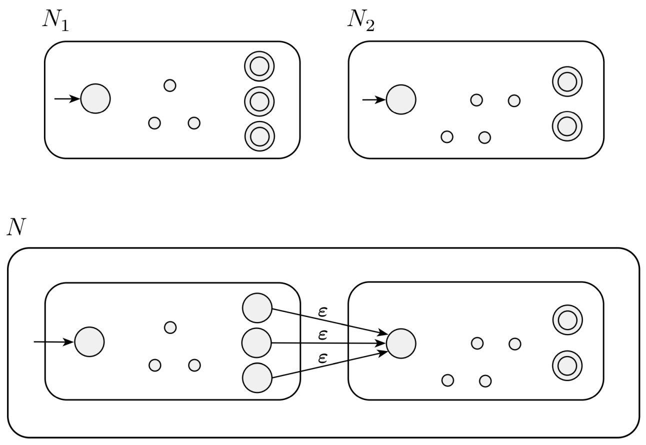

Figure 1.3: Construction of an NFA N that accepts $A \circ B$. (M. Sipser, 2012)

To prove 3, we construct N as follows:

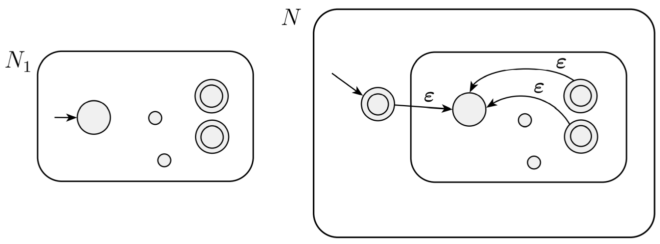

Figure 1.4: Construction of an NFA N that accepts A^*. (M. Sipser, 2012)

1.1.4 Regular expressions

We define regular expressions inductively:

Definition 9 (Regular expression). Say that R is a **regular expression** is R is

1. $\varnothing$,
2. $\{\epsilon\}$,
3. $\{a\} \subseteq \Sigma$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is regular expressions.

The set of NFA-recognizable languages and the set of DFA-recognizable languages are the same according to Theorem 1, but does the set of regular languages same as the set of NFA-recognizable languages? Before answering this question, we introduce the generalized nondeterministic finite automaton.

Definition 10 (generalized nondeterministic finite automaton).
A **generalized nondeterministic finite automaton (GNDF)** is a 5-tuple $\mathcal{M} = (Q, \Sigma, \delta, q_0, q_{\text{accept}})$, where

1. Q, Σ and q_0 are defined same as in DFA and NFA.
2. $\delta : (Q \setminus \{q_0\}) \times (Q \setminus \{q_{\text{accept}}\}) \to \mathcal{R}$ is the transition function, where $\mathcal{R}$ is the collection of all regular expressions over the alphabet Σ.
3. q_{accept} is the accept state.

Remark. The transition function takes as its argument a pair of two states and outputs a regular expression (the label of the transition).

Finally, we have a summarized theorem:

Theorem 2 (Kleene's theorem). The set of regular languages, the set of NFA-recognizable languages, and the set of DFA-recognizable languages are all the same.

Again, we omit the formal proof but provide the idea of the proof.

Proof Idea. Consider the following example reduction from a NFA to a GNFA and then a regular expression:

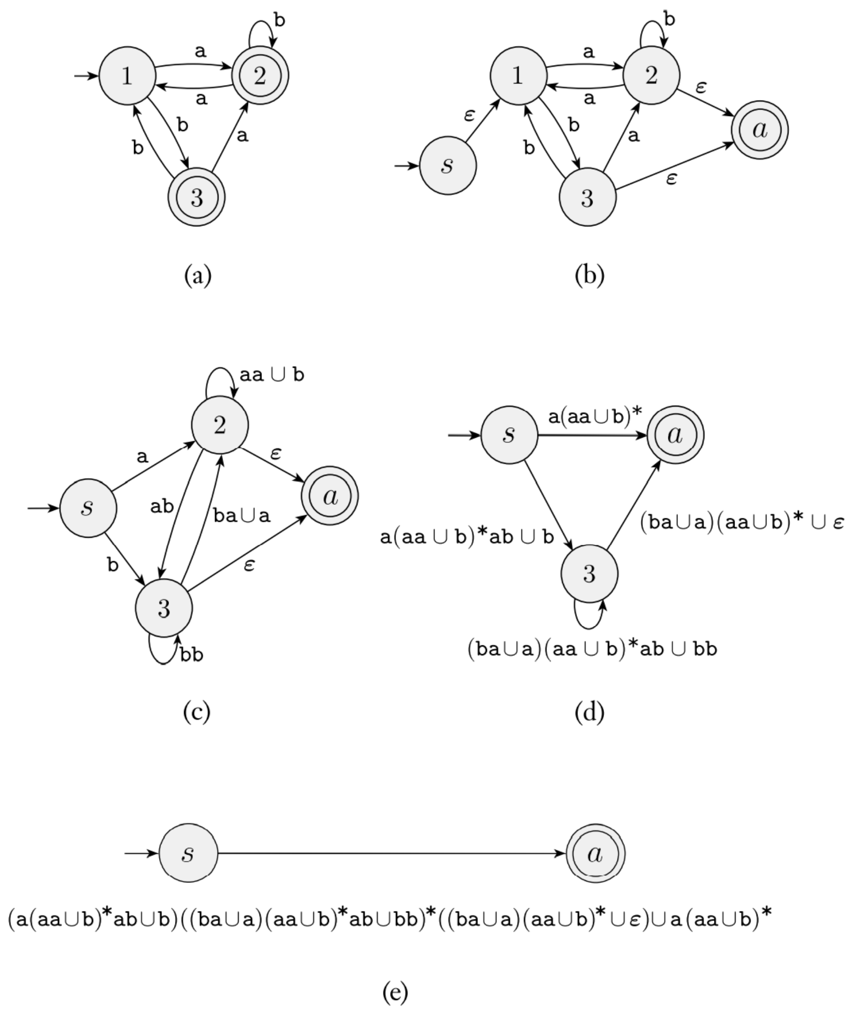

Figure 1.5: Converting a three-state DFA to an equivalent regular expression
(M. Sipser, 2012)

1. We first add a new q_0 connects to the old q_0 and a new q_{accept} connects to all the states in F.
2. Then we delete every internal state one by one and append the labels to the corresponding transitions. (After deleting the state q, if it has a self-loop labeled with a, then we add a^* to every path pass through q by squeezing it with the labels at front and back)

1.2 Context-Free languages

1.2.1 Context-Free Grammar

Recall that $L = \{0^n 1^n : n \in \mathbb{N}\}$ is not regular, equivalent of saying that no regular expression describe it (Kleene's theorem), now we define a more powerful method that can describe this language:

Definition 11 (Context-free grammars)**.**
A **context-free grammars** is a 4-tuple $G = (V, \Sigma, R, S)$, where

1. V is a finite set called the **variables**,
2. Σ is a finite set, disjoint from V, called the **terminals**,
3. R is a finite set of **rules**, with each rule being a variable and a string of variables and terminals,
4. $S \in V$ is the start variable.

Definition 12 (Rule)**.**

-

Example. Consider grammar $G = (\{S\}, \{0, 1\}, R, S)$. The set of rules, R, is

$$S \to 0S1|\epsilon$$

2 | Turing Machine

2.1 Definition of Turing Machine

Definition 13 (Turing Machine).
A *Turing Machine* is a 7-tuple, $(Q, \Sigma, \Gamma, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and:

1. Q is the set of **states**
2. Σ is the **input alphabet** not containing the blank symbol $\sqcup$
3. Γ is the **tape alphabet**, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the **transition function**
5. $q_0 \in Q$ is the start state
6. $q_{\text{accept}} \in Q$ is the accept state
7. q_{reject} is the reject state, where $q_{\text{accept}} \neq q_{\text{reject}}$

2.1.1 Multi-tape Turing Machine

2.1.2 Nondeterministic Turing Machine

Definition 14 (Configuration).
A *configuration* of the Turing machine is uqv, where:

- $q \in Q$ is the current state.
- u and v are strings over the tape alphabet Γ.
- The first symbol of v is the current head location.

Definition 15 (Configuration Tree). Given a Turing machine M and an input string s.
A *configuration tree* is a tree with:

- The root node is $q_0 s$
- Given the current node uqv, the child $u'q'v' \in \delta(q, v[1])$

2 The Class NP

2.1 Definition

There are two equivalent definitions of **NP**, the first one is through Nondeterministic Turing machine.

Definition 16 (NTIME).

$$\begin{aligned}\mathbf{NTIME}(t(n)) = \{L : L \text{ is a language decided by an } O(t(n))\\ \text{time non-deterministic Turing Machine}\}\end{aligned}$$

Definition 17 (NP).

$$\mathbf{NP} = \bigcup_k \mathbf{NTIME}(n^k)$$

Note. **NP** stands for nondeterministic polynomial time.

The second one relies on the definition of a **verifier**.

Definition 18 (Verifier). A **verifier** for a language A is an algorithm V, where

$$A = \{w : V \text{ accepts } \langle w, c\rangle \text{ for some string } c\}$$

(c is called a **certificate** or **proof**)

Observe.

- If V accepts $\langle w, c\rangle$ then $w \in L$.
- If V rejects $\langle w, c\rangle$ then either
  - $w \in L$, but the certificate c is wrong, or
  - $w \notin L$, then no possible c works.

Intuition. A certificate c can be understood as a solution to the problem w, and the verifier V verifies whether the given input is a certificate of w.

Definition 19 (NP). **NP** is the class of languages that have polynomial time verifiers.

Intuition. Under the second definition, **NP** can be understood as the set of problems that when given a string, we can verify whether this string is a certificate in polynomial time.

Theorem 3. Definition 5 and 7 of **NP** are equivalent.

Proof Idea. We show how to convert a polynomial time verifier to an equivalent polynomial time NTM and vice versa. The NTM simulates the verifier by guessing the certificate. The verifier simulates the NTM by using the accepting branch as the certificate.

2.2 Examples

2.2.1 CLIQUE

$$\textbf{CLIQUE} = \{\langle G, k\rangle : G \text{ is an undirected graph with a } k\text{-clique}\} \tag{2.1}$$

2.2.2 SUBSET-SUM

$$\begin{aligned}\textbf{SUBSET-SUM} = \{\langle S, t\rangle : S = \{x_1, \cdots, x_k\},\ \text{and for some}\\ \{y_1, \cdots, y_l\} \subseteq \{x_1, \cdots, x_k\},\ \text{we have} \sum y_i = t\}\end{aligned} \tag{2.2}$$

2.2.3 SAT

$$\textbf{SAT} = \{\langle \phi\rangle : \phi \text{ is a satisfiable Boolean formula}\} \tag{2.3}$$

2.3 coNP

Definition 20 (coNP)**.** **coNP** is the class of language that are complements of languages in **NP**.

2 NP-Completeness

2.1 Polynomial time reduction

Definition 21 (Polynomial time computable function)**.**
A function $f : \Sigma^* \to \Sigma^*$ is a **polynomial time computable function** if there exists a polynomial time Turing machine M that halts with just $f(w)$ on its tape, when started on any input w.

Definition 22 (Polynomial time reducible)**.**
Language A is **polynomial time reducible** to language B, denoted as $A \leq_P B$, if there exists a polynomial time computable function $f : \Sigma^* \to \Sigma^*$, such that for every w,

$$w \in A \iff f(w) \in B \tag{2.1}$$

such a function f is called the **polynomial time reduction** of A to B.

Proposition 1. If $A \leq_P B$ and $B \in P$, then $A \in P$

2.2 NP-Completeness

Definition 23 (NP-Complete)**.** A language B is **NP-Complete** if it satisfies two conditions:

1. B is in **NP**.
2. every A in **NP** is polynomial time reducible to B.

Proposition 2. If B is **NP-Complete** and $B \in P$, then **P=NP**.

Proposition 3. If B is **NP-Complete** and $B \leq_P C$ in **NP**, then C is **NP-Complete**.

2.3 Examples

2 Communication complexity

Definition 24 (Deterministic Communication Complexity). For $f : X \times Y \to Z$, the *Deterministic Communication Complexity* is

$$D(f) = \min_{\text{protocal computing} f} \{ \max_{x \in X, y \in Y} \{\text{number of bits on input (x,y)}\}\}$$
$$= \min_{\text{protocal tree } T \text{ for } f} \text{depth}(T)$$

Observe. For any arbitrary tree, number of leaves $\leq 2^{\text{depth}}$.

Definition 25 (Rectangle). A *rectangle* in $X \times Y$ is a set of the form $S = A \times B$ where $A \subseteq X$ and $B \subseteq Y$.

Observe. X is a rectangle if and only if

$$\big((x,y) \in S \land (x',y') \in S\big) \implies \big((x,y') \in S \land (x',y) \in S\big)$$

Remark. Let v be a node in a protocol tree. Let R_v be inputs that arrives at v, then R_v is a rectangle.

Definition 26 (f-monochromatic). A rectangle R, is *f-monochromatic* if $\exists z \in Z$ s.t.t $\forall (x,y) \in R$, $f(x,y) = z$.

Remark. If v is a leaf, then R_v is an f-monochromatic rectangle.

Remark. Every input (x,y) must arrive at exactly one leaf v. So $(x,y) \in R_v$.

Proposition 4. The set $\{R_v : v \text{ is a leaf of } T\}$ is a partition of $X \times Y$ into f-monochromatic rectangles.

Definition 27 (C-partition). $C^{\text{partition}}(f) = \min\{|\mathcal{R}| : \mathcal{R} \text{ is a f-monochromatic partition }\}$.

$C^{\text{partition}}(f)$ is the minimum number of rectangles needed in any f-monochromatic partition.

Corollary. For any protocol tree T, $C^{\text{partition}}(f) \leq$ number of leaves in T.

Corollary. $C^{\text{partition}}(f) \leq \min_{\text{protocal tree}}\{2^{\text{depth}}(T)\}$

Theorem 4.

$$\lceil \log_2 C^{\text{partition}}(f) \rceil \leq \min_{\text{protocal tree}} \{\text{depth}(T)\} = D(T)$$

Definition 28 (Fooling set). TODO

Proposition 5.

$$C^{\text{partition}}(f) \geq |S| + 1$$

unless f is constant

Corollary.

$$D(f) \geq \lceil \log_2(|S| + 1) \rceil$$

Example. Let $EQ_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$

$$EQ_n(x,y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{o.w.} \end{cases}$$

The upper bound: $D(EQ_n) \leq n + 1$
The fooling set is $S = \{(x,x) : x \in \{0,1\}^n\}$, since:

1. $EQ_n(x,y) = 1, \forall x \in \{0,1\}^n$.
2. Consider (x,x) and $(x',x') \in S$ where $x \neq x'$, we have $EQ_n(x,x') = EQ_n(x',x) = 0$.

Conclude: $D(EQ_n) \geq \lceil \log_2(2^n + 2) \rceil = n + 1$

Definition 29 (Nondeterministic communication complexity)**.** *Nondeterministic communication complexity* of f is $N(f)$

$$= \min_{\text{nondet. protocal}} \{ \max_{x,y,\text{ s.t. } f(x,y)=1} \{\text{number of bits on a certificate for input } x, y\}\}$$

Example. For $f = EQ_n$:
Protocol 1:

1. Prover lets $z = (x,y)$
2. Alice accepts if $x = z[1] = z[2]$
3. Bob accepts if $x = z[2] = z[1]$

In this case, z is of length $2n$.
Protocol 2:

1. Prover lets $z = x$
2. Alice accepts if $z = y$
3. Bob accepts if $z = y$

(Explanation: If $x = y$, Prover sets $z = x$, both accepts; otherwise, there is no string z that will make both accepts.)
In this case, z is of length n.

Remark.

$$N(EQ_n) \leq n$$

Example. For $f = \neg EQ_n$:
Protocol 1: silly protocol, in this case, z is of length $2n$.
Protocol 2:

1. Prover lets $z = (i, b)$
2. Alice accepts if $x_i \neq b$
3. Bob accepts if $y_i = b$

(Explanation: If $x \neq y$, Prover finds i s.t. $x_i \neq y_i$, lets $b = y_i$, both will accepts; otherwise if $x = y$, one of them must reject regardless of z.)
In this case, z is of length $\lceil \log(n) \rceil + 1$.

Remark.

$$coN(EQ_n) = N(\neg EQ_n) \leq \lceil \log(n) \rceil + 1$$

2.0.1 Cover

Definition 30 (Cover)**.** The set $R = \{R_1, \cdots, R_n\}$ is a 1-cover of the 1-entries if:

1. Each R_i is a rectangle where all values are all equal to 1.
2. Each $(x, y) \in X \times Y$ with $f(x, y) = 1$ is contained in at least one R_i (Needn't be disjoint)

Definition 31.

$$C^{\text{1-cover}}(f) = \min\{|R| : R \text{ is a cover of the 1-entries}\}$$

$$C^{\text{0-cover}}(f) = \min\{|R| : R \text{ is a cover of the 0-entries}\} = C^{\text{1-cover}}(\neg f)$$

Observe.

$$C^{\text{partition}}(f) \geq C^{\text{1-cover}}(f) + C^{\text{0-cover}}(f)$$

Remark.

$$C^{\text{0-cover}}(DIST_n) \leq n$$

$$C^{\text{0-cover}}(EQ) \leq 2n$$

As previously seen.

$$D(f) \geq \lceil \log_2(C^{\text{partition}}(f)) \rceil$$

Theorem 5.

$$N(f) = \lceil \log_2(C^{\text{1-cover}}(f)) \rceil$$

$$coN(f) = \lceil \log_2(C^{\text{0-cover}}(f)) \rceil$$

Corollary.

$$D(f) \geq N(f)$$
$$D(f) \geq coN(f)$$

Observe. Deterministic communication complexity is closed under complement, but Nondeterministic communication complexity does not.