1 Prerequisite Definitions

Alphabets Σ, and Γ are finite nonempty sets of symbols.

A *string* is a finite sequence of zero or more symbols from an alphabet.

$\Sigma^\star$ is the set of all strings over alphabet Σ.

ε is the empty string and cannot be in Σ.

A *problem* is a mapping from strings to strings.

A *decision problem* is a problem whose output is yes/no (or often accept/reject).

A decision problem be thought of as the set of all strings for which the function outputs "accept".

A *language* is a set of strings, so any set $S \subseteq \Sigma^\star$ is a language, even $\emptyset$. Thus, decision problems are equivalent to languages.

2 Regular Languages

$L(M)$ is the language accepted by machine M.

A deterministic finite automaton is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set of states,
- Σ is an alphabet,
- $\delta : Q \times \Sigma \to Q$ is a transition function describing its transitions and labels,
- $q_0 \in Q$ is the starting state, and
- $F \subseteq Q$ is a set of accepting states.

If δ is not fully specified, we assume an implicit transition to an *error state*.

A deterministic finite automaton M accepts input string $w = w_1 w_2 \ldots w_n$ ($w_i \in \Sigma$) if there exists a sequence of states $r_0, r_1, r_2, \ldots, r_n$ ($r_i \in Q$) such that

- $r_0 = q_0$,
- for all $i \in \{1, \ldots, n\}$, $r_i = \delta(r_{i-1}, w_i)$, and
- $r_n \in F$.

$r_0, r_1, r_2, \ldots, r_n$ are the sequence of states visited during the machine's computation.

A non-deterministic finite automaton is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where

- Q, Σ, q_0, F are the same as a deterministic finite automaton's, and
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$.

A non-deterministic finite automaton accepts the string $w = w_1 w_2 \ldots w_n$ ($w_i \in \Sigma$) if there exist a string $y = y_1 y_2 \ldots y_m$ ($y_i \in \Sigma \cup \{\varepsilon\}$) and a sequence $r = r_0, r_1, \ldots, r_n$ ($r_i \in Q$) such that

- $w = y_1 \circ y_2 \circ \cdots \circ y_m$ (i.e. y is w with some ε inserted),
- $r_0 = q_0$,
- for all $i = \{1, \ldots, m\}$, $r_i \in \delta(r_{i-1}, q_i)$, and
- $r_m \in F$.

The *ε-closure* for any set $S \subseteq Q$ is denoted $E(S)$, which is the set of all states in Q that can be reachable by following any number of ε-transition.

Theorem 1. *A non-deterministic finite automaton can be converted to an equivalent deterministic finite automaton.*

A *regular language* is any language accepted by some finite automaton. The set of all regular languages is called the *class of regular languages*.

Theorem 2. *Regular languages are closed under*

- *Concatenation* $L_1 \circ L_2 = \{x \circ y : x \in L_1 \text{ and } y \in L_2\}$. *Note:* $L_1 \not\subseteq L_1 \circ L_2$.
- *Union* $L_1 \cup L_2 = \{x : x \in L_1 \text{ or } x \in L_2\}$.
- *Intersection* $L_1 \cap L_2 = \{x : x \in L_1 \text{ and } x \in L_2\}$.
- *Complement* $\overline{L} = \Sigma^\star \setminus L = \{x : x \notin L\}$.
- *Star* $L^\star = \{x_1 \circ x_2 \circ \cdots \circ x_k : x_i \in L \text{ and } k \geq 0\}$.

R is a regular expression if R is

- $a \in \Sigma$,
- ε,
- $\emptyset$,
- $R_1 \cup R_2$, or $R_1 | R_2$,
- $R_1 \circ R_2$, or $R_1 R_2$,
- $R_1^\star$,
- Shorthand: $\Sigma = (a_1 | a_2 | \ldots | a_k)$, $a_i \in \Sigma$,

where R_i is a regular expression.

Identities of Regular Languages

- $\emptyset \cup R = R \cup \emptyset = R$
- $\emptyset \circ R = R \circ \emptyset = \emptyset$
- $\varepsilon \circ R = R \circ \varepsilon = R$
- $\varepsilon^\star = \varepsilon$
- $\emptyset^\star = \emptyset$
- $\emptyset \cup R \circ R^\star = R \circ R^\star \cup \varepsilon = R^\star$
- $(a|b)^\star = (a^\star | b^\star)^\star = (a^\star b^\star)^\star = (a^\star | b)^\star = (a | b^\star)^\star = a^\star (b a^\star)^\star = b^\star (a b^\star)^\star$

Theorem 3. *Languages accepted by DFAs = languages accepted by NFAs = regular languages*

Theorem 4. *If L is a finite language, L is regular.*

If a computation path of any finite automaton is longer than the number of states it has, there must be a cycle in that computation path.

Lemma 1 (Pumping Lemma)**.** *Every regular language satisfies the pumping condition.*

Pumping condition: There exists an integer p such that for every string $w \in L$, with $|w| \geq p$, there exist strings $x, y, z \in \Sigma^\star$ with $w = xyz, y \neq \varepsilon, |xy| \leq p$ such that for all $i \geq 0$, $xy^i z \in L$.

Negation of pumping condition: For all integers p, there exists a string $w \in L$, with $|w| \geq p$, for all $x, y, z \in \Sigma^\star$ with $w = xyz, y \neq \varepsilon, |xy| \leq p$, there exists $i \geq 0, i \neq 1$ such that $xy^i z \notin L$.

Limitations of finite automata:

- Only read input once, left to right.
- Only finite memory.

3 Context-Free Languages

A pushdown automaton is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where

- Q is a finite set of states,
- Σ is its input alphabet,
- Γ is its stack alphabet,
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \to 2^{Q \times (\Gamma \cup \{\varepsilon\})}$ is its transition function,
- $q_0 \in Q$ is its starting state, and
- $F \subseteq Q$ is a finite set of accepting states.

Labels: $a, b \to c$: if input symbol is a, and top of stack is b, pop it and push c. In other words, input symbol read, stack symbol popped $\to$ stack symbol pushed, e.g. $0, \varepsilon \to \$$.

Suppose u, v, w are strings of variables and terminals, and there is a rule $A \to w$. From the string uAv, we can obtain uwv. We write $uAv \to uwv$, and say uAv yields uwv.

If $u_1 \to u_2 \to \cdots \to u_k$, then $u_1 \to^\star u_k$, or u_1 derives u_k. There must be a finite number of arrows between u_1 and u_k.

Given a grammar G, the language derived by the grammar is $L(G) = \{w \in$

$\Sigma^\star : S \to^\star w$ and S is the start variable$\}$

Context-free grammar: the lhs of rules is a single variable, rhs is any string of variables and terminals. A *context-free language* is one that can be derived from a context-free grammar. An example context-free grammar is $G = (V, \Sigma, R, \langle \texttt{EXPR} \rangle)$, where $V = \{\langle \texttt{EXPR} \rangle, \langle \texttt{TERM} \rangle, \langle \texttt{FACTOR} \rangle\}$, $\Sigma = \{a, +, \times, (,)\}$, and $R = \{\langle \texttt{EXPR} \rangle \to \langle \texttt{EXPR} \rangle + \langle \texttt{TERM} \rangle | \langle \texttt{TERM} \rangle, \langle \texttt{TERM} \rangle \to \langle \texttt{TERM} \rangle \times \langle \texttt{FACTOR} \rangle | \langle \texttt{FACTOR} \rangle, \langle \texttt{FACTOR} \rangle \to (\langle \texttt{EXPR} \rangle)\}$.

A *left-most derivation* is a sequence $S \to u_1 \to u_2 \to \cdots \to u_k \to w$ where each step applies a rule to the left-most variable. A grammar is *ambiguous* when it has multiple left-most derivations for the same string.

Theorem 5. *A language L is recognized by a pushdown automaton iff L is described by a context-free grammar.*

Theorem 6. *Context-free languages are closed under union, concatenation, star.*

4 Recognizable Languages

Differences from previous models

- The input is written on tape.
- It can write to the tape.
- It can move left and right on tape.
- It halts immediately when it reaches an accepting or rejecting state. The rejecting state must exist but may not be shown.

A deterministic Turing machine is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where

- Q is its finite non-empty set of states,
- Σ is its input alphabet,
- Γ is its tape alphabet ($\Sigma \subset \Gamma$ and $\llcorner\!\lrcorner \in \Gamma \setminus \Sigma$),
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is its transition function,
- $q_0 \in Q$ is its starting state,
- $q_{accept} \in Q$ is its accepting state, and
- $q_{reject} \in Q$ is its rejecting state ($q_{reject} \neq q_{accept}$).

Labels: $a \to b, R$: if tape symbol is a, write b and move head right. $a \to R$: if tape symbol is a, move head right. $a, b, c \to R$: if tape symbol is a, b, or c, move head right.

On input x, a Turing machine can (1) accept, (2) reject, or (3) run in an infinite loop.

The language *recognized* by a Turing machine M is $L(M) = \{x :$ on input x, M halts in $q_{accept}\}$. A language is *recognizable* if there exists a Turing machine which recognizes it.

Regular languages $\subseteq$ context-free languages $\subseteq$ decidable languages $\subseteq$ recognizable languages

A *configuration* is a way to describe the entire state of the Turing machine. It is a string aqb where $a \in \Gamma^\star, q \in Q, b \in \Gamma^\star$, which indicates that q is the current state of the Turing machine, the tape content currently is ab and its head is currently pointing at the first symbol of b. Any Turing machine halts if its configuration is of the form $aq_{accept}b$, or $aq_{reject}b$ for any ab. Config(i) uniquely determines Config($i+1$).

Theorem 7. *Every k-tape Turing machine has an equivalent single tape Turing machine.*

If the alphabet of the multitape Turing machine is Γ, we can make the single tape Turing machine's alphabet $(\Gamma \cup \{\#\}) \times \{\texttt{normal}, \texttt{bold}\}$.

A non-deterministic Turing machine is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where the only difference from a deterministic Turing machine is the transition function $delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$.

A non-deterministic Turing machine accepts its input iff some node in the configuration tree has q_{accept}. It does not accept its input iff the configuration tree grows forever (infinite loop) or no node in the tree has q_{accept}.

Acceptance of a non-deterministic Turing machine: input w is accepted if there exist configurations $c_0, c_1, \ldots, c_k$ where

- $c_0 = q_{start}w$, and
- $c_i \Rightarrow c_{i+1}$ (c_{i+1} is a possible configuration from c_i, following the transition function δ).

The outcomes could be

- w is accepted, i.e. there exists a node in the tree which is an accepting configuration,
- w is explicitly rejected, i.e. the tree is finite but no node is an accepting configuration (all leaves are rejecting configurations), or
- the non-deterministic Turing machine runs forever on w, i.e. the tree is infinite but no node is an accepting configuration (there might be finite branches terminating in a rejecting configuration in the tree).

A Turing machien is a *decider* if it halts on all inputs, i.e. it either rejects or accepts all inputs.

Theorem 8. *Every non-deterministic Turing machine has an equivalent deterministic Turing machine. If that non-deterministic Turing machine is a decider, there is an equivalent deterministic Turing machine decider.*

Theorem 9. *Recognizable languages are closed under union, intersection, concatenation, star.*

Implementation level description of a multitape Turing machine for $L = \{x\#x : x \in \{0,1\}^\star\}$:

- Scan the first head to the right until it reads a #. Move right. The second head is still at the start of the second tape.
- Repeatedly read symbol from the first tape (reject if the symbol is not 0 or 1), write it to the second tape, and move both heads right, until seeing a blank on the first tape.
- Move the first head left until a # is under it. Replace the symbol with a blank ($\llcorner\!\lrcorner$).
- Move both heads left until they reach the start of their respective tapes (using the $ sign hack to mark the start of the tape).
- Repeat until seeing a blank on both tapes.
  - If the symbols on the two tapes differ, reject.
  - Otherwise, move both head right.

$\langle O \rangle$ is a string encoding for the object O.

Cardinality of Sets: two sets A and B have the same *cardinality* if there exists a bijection $f : A \to B$.

$\mathbb{N} = \{1, 2, 3, \ldots\}$ is the set of all natural numbers. A set is *finite* if it has a bijection to $\{1..n\}$ for some natural number n. A set is *countably infinite* if it has the same cardinality as $\mathbb{N}$. A set is *countable* or *at most countable* if it is finite or countably infinite.

Lemma 2. *Any language L is countable.*

Lemma 3. *The set of all Turing machines is countable.*

Lemma 4. *The set $\mathscr{B}$ of all infinite bit-sequences is not countable.*

Lemma 5. *$2^{\Sigma^\star}$ is uncountable.*

5 Reductions

$A_{TM} = \{\langle M,w\rangle : M \text{ accepts } w\}$ and $HALT_{TM} = \{\langle M,w\rangle : M \text{ halts on input } w\}$ are recognizable but not decidable.

Theorem 10. *If L and $\overline{L}$ are recognizable, then L is decidable (and so is $\overline{L}$).*

Lemma 6. *$\overline{A_{TM}}$ is unrecognizable.*

Proof template for undecidability via Turing reduction: Reduce a problem known to be undecidable to that language L, usually A_{TM}, i.e. $A_{TM} \leq_T L$. Assume a Turing machine decider R for L. Construct S that decides A_{TM} using R.

Runtime of a deterministic Turing machine is a function $f:\mathbb{N}\to\mathbb{N}$ given by $f(n) = max_{x\in\Sigma^\star,|x|=n}$(no. of steps of M on input x).

$TIME(t(n)) = \{$language L : $\exists$deterministic Turing machine that decides L in time $O(t(n))\}$.

$P = \bigcup_{c\geq 0} TIME(n^c)$

$EXP = \bigcup_{k\geq 0} TIME(2^{n^k})$

Theorem 11 (Time hierarchy theorem)**.** *If $f:\mathbb{N}\to\mathbb{N}$ is reasonable and $f=\Omega(n\log n)$ then $TIME(f(n)) \subset TIME(f(n)^2)$.*

Lemma 7. *$P \subset EXP$*

Runtime of a non-deterministic Turing machine is the height of the configuration tree.

$NTIME(t(n)) = \{$language L : $\exists$ non-deterministic Turing machine that decides L in time $t(n)\}$

$NP = \bigcup_{c>0} NTIME(n^c)$, i.e. languages for which it is easy to verify membership.

Lemma 8. *$P \subseteq NP$*

Lemma 9. *$NP \subseteq EXP$*

Verifier-based definition for $L \in NP$: there exists a deterministic polytime Turing machine V and a constant c such that $L = \{x\in\Sigma^\star : \exists y \in \Sigma^\star, |y| \leq |x|^c, V \text{ accepts } (x,y)\}$.

A function is *polytime computable* if $f:\Sigma^\star\to\Sigma^\star$ if there exists a Turing machine M that has x as input, runs for time poly($|x|$) and halts with $f(x)$ written on the tape.

f is a *polytime reduction* from language A to language B, denoted $A\leq_P B$ if (1) $f(A)\subseteq B$, (2) $f(\overline{A})\subseteq\overline{B}$, and (3) f is a polytime computable function.

Theorem 12. *If $A \leq_P B$ and $B \in P$ then $A \in P$.*

A language L is *NP-hard* if $A \leq_P L$ for all $A \in NP$. A language L is *NP-complete* if L is NP-hard and $L \in NP$.

Theorem 13. *If $P \neq NP$, then there exists language L such that $L \notin NP$-complete, $L \notin P$, and $L \in NP$.*

Theorem 14. *If (1) B is NP-complete, (2) $C \in NP$, and (3) $B \leq_P C$, then C is NP-complete.*

CLIQUE is a language whose strings are of the form $\langle G,k\rangle$, where $G=(V,E)$ is a graph and $k\in\mathbb{N}$, for which there exists $U\subseteq V$ with $|U|\geq k$ such that $\{u,v\}\in E$ for all distinct vertices $u,v\in U$.

Theorem 15. *CLIQUE is NP-complete*

Theorem 16. *$3SAT \leq_P CLIQUE$*

Theorem 17. *$3SAT \leq_P MAXCLIQUE$*

Reductions from $3SAT$ often involves *gadgets*:

- *Clause gadgets:* for the assignemnt to pick a true literal in each clause (a clique must pick a vertex from each group)
- *Variable gadget:* force assignemnt to set each variable either to true or false but not both (a clique cannot pick both x_i and $\overline{x_i}$).

INDSET is a language whose strings are of the form $\langle H,k\rangle$, where $H=(V,E)$ is a graph and $k\in\mathbb{N}$, for which there exists $U\subseteq V$ with $|U|=k$ such that $\forall u,v\in U, \{u,v\}\notin E$.

VERTEX − COVER is a language whose strings are of the form $\langle H,t\rangle$, where $H=(V,E)$ is a graph and $t\in\mathbb{N}$, for which there exists a set $C\subseteq V$ with $|C|\leq t$ such that $\forall\{u,v\}\in E$, either u, v or both is in C.

Let $G=(V,E)$ be a graph. Then $\overline{G}=(V,\overline{E})$ where $\overline{E}=\{\{u,v\}:\{u,v\}\notin E\}$.

Lemma 10. *U is a clique in G iff $\overline{U}=V\setminus U$ is a vertex cover in $\overline{G}$. This implies G has a clique of size $\geq k$ iff $\overline{G}$ has a vertex cover of size $\leq n-k$, where $|V|=n$.*

Lemma 11. *$CLIQUE \leq_P VERTEX-COVER$*

Lemma 12. *$CLIQUE \leq_P INDSET$*

Theorem 18. *SAT is NP-complete via a where*

$$C = Q\cup\{\#\}\cup\Gamma$$
$$x_{i,j,s} = true \Leftrightarrow cell[i,j]=s$$
$$\Phi_{start} = x_{1,1,\#}\wedge x_{1,2,q_{start}}\wedge x_{1,3,w_1}\wedge\ldots$$
$$\wedge x_{1,n^k-1,\sqcup}\wedge x_{1,n^k,\#}$$
$$\Phi_{cell} = \bigwedge_{i,j=1}^{n^k}\left(\bigvee_{s\in C} x_{i,j,s}\wedge \bigwedge_{s,t\in C,1\leq i,j\leq n^k}\neg(x_{i,j,s}\wedge x_{i,j,t})\right)$$
$$\Phi_{moves} = \bigwedge_{i,j\geq n^k}(window[i,j] \text{ is valid})$$
$$\Phi_{accept} = \bigvee_{1\leq i,j\leq n^k} x_{i,j,q_{accept}}$$

$coNP = \{$language L : $\overline{L}\in NP\}$, i.e. languages for which it is easy to verify non-membership. Machine model for $L\in coNP$ is when $x\in L$, all leaves are accepting configurations; otherwise, when $x\notin L$, there exists one leaf which is a rejecting configuration.

$coNP$-complete $= \{$language B : $B\in coNP, \forall A\in coNP, A\leq_P B\}$.

Theorem 19. *NOSAT is coNP-complete.*

Lemma 13. *$L\in NP$-complete iff $\overline{L}\in coNP$-complete.*

6 Probabilistic Turing Machines

RP, or randomized polynomial time, are the languages L for which there is a *probabilistic* Turing machine that, on input x, runs in poly($|x|$) and when $x \in L$, Pr[reaching accept] $\geq \frac{1}{2}$; otherwise, when $x \notin L$, Pr[reaching reject] $= 1$.

Second definition for RP: it contains languages L for which there exists a deterministic polytime Turing machine V such that when $x \in L$, for at least half of all y with $|y| \leq \text{poly}(|x|), V$ accepts (x,y); when $x \notin L$, for all y with $|y| \leq \text{poly}(|x|), V$ rejects (x,y).

Contrast with *NP*, where $\forall x \in L$, Pr[reaching accept] > 0, $\forall x \notin L$, Pr[reaching reject] $= 1$.

Theorem 20. *$RP \subseteq NP$*

coRP: $\forall x \in L$, Pr[reaching accept] $= 1$, $\forall x \notin L$, Pr[reaching reject] $\geq \frac{1}{2}$.

coNP: $\forall x \in L$, Pr[reaching accept] $= 1$, $\forall x \notin L$, Pr[reaching reject] > 0.

BPP, or bounded error probabilistic polynomial time: $\forall x \in L$, Pr[reaching accept] $\geq \frac{2}{3}$, $\forall x \notin L$, Pr[reaching reject] $\geq \frac{2}{3}$.

Lemma 14. *$RP \subseteq BPP$*

Lemma 15. *$coRP \subseteq BPP$*

Lemma 16. *$RP(\frac{1}{2}) = RP(\frac{3}{4})$ (proof via amplification)*

Lemma 17. *RP is closed under composition.*

7 Communication Complexity

Model:

- Finite sets X, Y, Z
- Function $f : X \times Y \to Z$
- Two player, Alice and Bob
- Decide on a communication protocol beforehand
- Alice has $x \in X$, Bob has $y \in Y$
- Goal: collaboratively compute $f(x,y)$ by sending bits back and forth (must end with both side knowing $f(x,y)$)

The *trivial prototol:*

- Alice sends x to Bob ($\log|X|$)
- Bob computes and sends $z = f(x,y)$ to Alice ($\log|Z|$)

Total: $\log|X| + \log|Z|$ or $\log|Y| + \log|Z|$

A *communication protocol* is a binary tree where each node is labelled by either $a_v : X \to \{L,R\}$ or $b_v : Y \to \{L,R\}$ and each leaf is labelled by an element of Z. The depth of the protocol tree is the maximum number of bits sent by the protocol.

The deterministic communication complexity of a function f is

$$D(f) = \min_{\text{tree for } f} \left(\max_{(x,y)} (\text{number of bits}) \right) = \min_{\text{tree for } f} (\text{depth of tree})$$

Lemma 18. *$D(EQ_n) \leq n+1$*

A *rectangle* in $X \times Y$ is a set of the form $R = A \times B$ where $A \subseteq X$ and $B \subseteq Y$. R is a rectangle iff $(x,y) \in R \wedge (x',y') \in R \Leftrightarrow (x,y') \in R \wedge (x',y) \in R$

Lemma 19. *Let T be a protocol tree, R_v be the set of inputs that causes the protocol to arrive at node v. Then R_v is a rectangle.*

A rectangle is called *f-monochromatic* if $f(x,y)$ is the same for all $(x,y) \in R$.

Let $R_i \subset X \times Y$ be a rectangle for $i = 1, \ldots, k$. The set $\mathscr{R} = \{R_1, \ldots, R_k\}$ is called an *f-monochromatic partition (into rectangles)* if each R_i is f-monochromatic, and each $(x,y) \in X \times Y$ is contained in exactly one R_i.

$C^{\text{partition}}(f) = \min\{|\mathscr{R}| : \mathscr{R}$ is an f-monochromatic partition$\}$

Lemma 20. *For any protocol tree T, the rectangles $\{R : v$ is a leaf in $T\}$ are an f-monochromatic partition.*

Lemma 21. *$C^{partition}(f) \leq \min_{\text{protocol tree } T} |\text{number of leaves in } T|$*

Lemma 22. *$D(f) \geq \lceil \log_2 C^{partition}(f) \rceil$*

A *fooling set* $S \subseteq X \times Y$ is a set where all points $(x,y) \in S$ have the same value $f(x,y) = z$, and for any distinct points (x,y) and (x',y') in S, either $f(x,y') \neq z$ or $f(x',y) \neq z$.

Lemma 23. *$C^{partition}(f) \geq |S| + 1$, where S is a fooling set for f*

Lemma 24. *$D(f) \geq \lceil \log_2(|S|+1) \rceil$, where S is a fooling set for f*

Lemma 25. *$D(EQ_n) = D(GTE_n) = D(DISJ_n) = n+1$*

Model for *non-deterministic communication complexity*:

- Function $f : X \times Y \to Z$ is known to all
- Bob does not know x, Alice does not know y
- Alice and Bob do not communicate
- Piere tries to force Alice and Bob to accept by sending certificate z. How short can z be?

$$N(f) = \min_{\text{nondet protocol}} (\text{length of cert}).$$

Or, $N(f) = \min\{k\}$ such that there exist A and B, for all $x \in X$, $y \in Y$, $f(x,y) = 1 \Rightarrow \exists z \in \{0,1\}^k, A(x,z) = 1 \wedge B(y,z) = 1$, $f(x,y) = 0 \Rightarrow \forall z \in \{0,1\}^k, A(x,z) = 0 \vee B(y,z) = 0$.

Lemma 26. *$N(\neg DISJ_n) \leq \log n$*

Lemma 27. *For all f, $D(f) = D(\neg f)$.*

Lemma 28. *$N(\neg EQ_n) \leq \log(n) + 1$*

Lemma 29. *Let S be a fooling set where $f(x,y) = 1$ for all $(x,y) \in S$. Then $N(f) \geq \lceil \log_2(|S|) \rceil$.*

Lemma 30. *$N(EQ_n) \geq n$*

The set $\mathscr{R} = \{R_1, \ldots, R_k\}$ is a cover of the 1-entries (by rectangles) if (1) each R_i is a rectangle containing only 1s, and (2) every $(x,y) \in X \times Y$ with $f(x,y) = 1$ is contained in at least one R_i.

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

$C^{\text{0-cover}}(f) = C^{\text{1-cover}}(\neg f)$.

Lemma 31. *$C^{partition}(f) = C^{1\text{-}cover}(f) + C^{0\text{-}cover}(f)$.*

Lemma 32. *$N(f) = \lceil \log_2(C^{1\text{-}cover}(f)) \rceil$*

Lemma 33. *$D(f) \geq N(f)$*

Lemma 34. *$D(\neg f) = N(f)$*

Theorem 21. *Let $f : X \times Y \to \{0,1\}$ be arbitrary, C_0 be a cover of the 0-entries, and C_1 be a cover of the 1-entries. Then $D(f) = O(\log C_0 * \log C_1)$.*

Lemma 35. *$D(f) = O(N(f) * N(\neg f))$*