Theory of Computation

1. Finite Automata

1. Central Concepts of Automata Theory :
   - L(A): the set of strings accepted by an automaton A is the *language* of A.
   - $\sum$: an alphabet is any finite set of symbols.
   - $\sum^*$: set of all strings over alphabet $\sum$
   - L: A *language* is a subset of $\sum^*$ for some alphabet $\sum$.
2. DFA:
   - Five parts: Q, $\sum$, δ, q_0, F
   - Transition Function: $\delta(q, a)$ or $\delta^*(q, w)$
3. Proofs of Set Equivalence: L(A) and the set of strings satisfies N
   1. If w is accepted by the DFA then $w \in N$: an induction on length of w
   2. If $w \in N$ then w is accepted by the DFA: contrapositive
4. Prove a language L is regular:
   1. A language *L* is *regular* if it is the language accepted by some DFA.
   2. Construct the DFA: five parts.
5. NFA:
   - Five parts: Q, $\sum$, δ, q_0, F
   - For any NFA there is a DFA that accepts the same language: subset construction (lazy form)
   - $\delta_N(q, w)$
6. ϵ-NFA
   - CL(q): set of states you can reach from state q following only arcs labeled ε.
   - $\delta_E(q, w)$
   - Equivalence of NFA, ϵ-NFA

Exercise:

2.2.4: Give DFA's accepting the following languages over the alphabet {0,1}

a) The set of all strings ending in 00.

b) The set of all strings with three consecutive 0's (not necessarily at the end)

c) The set of strings with 011 as a substring.

States:

A: string seen so far has no 011, but ends in 111 or 00(q_0)

B: string seen so far has no 011, but ends in a single 0

C: string seen so far has no 011, but ends in 01

D: string seen so far has 011(F)

2.3.1: Convert to a DFA the following NFA:

$$
\begin{array}{r||l|l}
 & 0 & 1 \\
\hline
\rightarrow p & \{p,q\} & \{p\} \\
q & \{r\} & \{r\} \\
r & \{s\} & \emptyset \\
*s & \{s\} & \{s\}
\end{array}
$$

	0	1
→{p}	{p,q}	{p}
{p,q}	{p,q,r}	{p,r}
{p,q,r}	{p,q,r,s}	{p,r}
{p,r}	{p,q,s}	{p}
*{p,q,r,s}	{p,q,r,s}	{p,r,s}
*{p,q,s}	{p,q,r,s}	{p,r,s}
*{p,r,s}	{p,q,s}	{p,s}
*{p,s}	{p,q,s}	{p,s}

2.5.1 Consider the following ϵ-NFA. Convert the automaton to a DFA

$$
\begin{array}{r||l|l|l|l}
 & \epsilon & a & b & c \\
\hline
\rightarrow p & \emptyset & \{p\} & \{q\} & \{r\} \\
q & \{p\} & \{q\} & \{r\} & \emptyset \\
*r & \{q\} & \{r\} & \emptyset & \{p\}
\end{array}
$$

CL(p)={p},CL(q)={p,q},CL(r)={p,q,r}

	a	b	c
→{p}	{p}	{p,q}	{p,q,r}
{p,q}	{p,q}	{p,q,r}	{p,q,r}
*{p,q,r}	{p,q,r}	{p,q,r}	{p,q,r}

2. Regular Language

1. Regular Expressions:
    - Three operations: Union, Concatenation, Kleene star
    - Equivalence of RE and DFA:
        - RE to ϵ-NFA: an induction on the number of operators (+, concatenation, *) in the RE

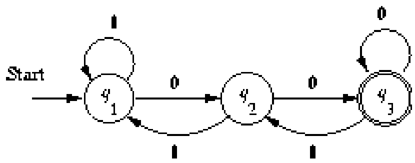
- DFA to RE: ans induction on k, the maximum state number we are allowed to traverse along a path

Exercise:

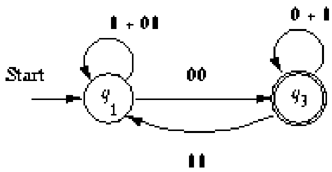
3.2.1: Here is a transition table for a DFA. Construct the transition diagram for the DFA and give a regular expression for its language by eliminating state q_2

	0	1
$\rightarrow q_1$	q_2	q_1
q_2	q_3	q_1
$*q_3$	q_3	q_2

Here is the transition diagram:



If we eliminate state q_2 we get:



So the expression for the ways to get from q_1 to q_3 is: **[1 + 01 + 00(0+10)*11]*00(0+10)***

3. Properties of Regular Languages

1. Pumping lemma:
   - Let L be a regular language. Then there exists a constant n (which depends on L) such that for every string w in L such that $|w| \geq n$, we can break w into three strings, $w = xyz$, such that:
     1. $y \neq \epsilon$.
     2. $|xy| \leq n$.
     3. For all $k \geq 0$, the string xy^kz is also in L.
   - Prove a language is not a RL: contrapositive
2. Product DFA:
   - Product DFA has set of states $Q \times R$. I.e., pairs [q, r] with q in Q, r in R.
   - We can construct the final state and let a property is true if and only if the product automaton's language is empty.
3. State Minimization:
   - Combining indistinguishable states: If state p is indistinguishable from q, and q is indistinguishable from r, then p is indistinguishable from r.
   - Eliminating Unreachable State: Remove states that are not reachable from the start state.

Exercise:

4.1.1: Prove that the following are not regular languages.

c) {$0^n10^n | n \geq 1$}

Let n be the pumping-lemma constant. Pick $w = 0^n10^n$. Then when we write $w = xyz$, we know that $|xy| \leq n$, and therefore y consists of only 0's. Thus, xz, which must be in L if L is regular, consists of fewer than n 0's, followed by a 1 and exactly n 0's. That string is not in L, so we contradict the assumption that L is regular.

4.3.1: Give an algorithm to tell whether a regular language L is infinite.

Let *n* be the pumping-lemma constant. Test all strings of length between *n* and $2n - 1$ for membership in *L*. If we find even one such string, then *L* is infinite.

4. Context-Free Grammar

1. CFG Formalism:
   - $G = (V, T, P, S)$, where V is the set of variables, T the terminals, P the set of productions, and S the start symbol.
   - Terminals: symbols of the alphabet of the language being defined.
   - Variables: a finite set of other symbols, each of which represents a language.
   - Start symbol: the variable whose language is the one being defined.
   - Production: variable (*head*) $\rightarrow$ string of variables and terminals (*body*).
   - Derivations: $\Rightarrow^*$ means "zero or more derivation steps."
   - L(G): G is a CFG, the language of G $L(G) = \{w|S \Rightarrow^* w\}$. A language that is defined by some CFG is called a *context-free language*(CFL).
2. Parse trees:
   - Yield: The concatenation of the labels of the leaves in left-to-right order.
   - Ambiguous: A CFG is *ambiguous* if there is a string in the language that is the yield of two or more parse trees. Or if there is a string in the language that has two different leftmost/rightmost derivations.
   - LL(1) Grammars: a grammar you can always figure out the production to use in a leftmost derivation by scanning the given string left-to-right and looking only at the next one symbol. LL(1) grammars are never ambiguous.

Exercise:

5.4.3: Consider the grammar: $S \rightarrow aS \mid aSbS \mid \epsilon$. This grammar is ambiguous. Please find an unambiguous grammar for the language.

$$S \rightarrow aS|aTbS|\epsilon$$
$$T \rightarrow aTbT|\epsilon$$

5. Pushdown Automata

1. PDA:
   - NPDA:
     - Defines all the CFL's.
     - $L_{wwr} = \{ww^R|w\ is\ in\ (0+1)^*\}$
   - DPDA:
     - $\delta(q, a, X)$ and $\delta(q, \varepsilon, X)$ cannot both be nonempty.
     - If L is a regular language, then $L = L(P)$ for some DPDA P.
     - If $L = N(P)/L(P)$ for some DPDA P, then L has an unambiguous context-free grammar.
     - $L_{wcwr} = \{wcw^R|w\ is\ in\ (0+1)^*\}$ is not regular, but accepted by DPDA.
   - Seven parts: $Q, \sum, \Gamma, \delta, q_0, Z_0, F$
   - $\delta(q, a, Z)$: a set of zero or more actions of the form (p, α).
2. Instantaneous description(ID):
   - a triple (q, w, α), q is the current state, w is the remaining input, α is the stack contents, top at the left.
   - $(q, aw, X\alpha) \vdash (p, w, \beta\alpha)$for any w and α, if $\delta(q, a, X)$ contains (p, β).

- ⊦*, meaning "zero or more moves"

3. L(P): P is a PDA, $L(P) = \{w|(q_0, w, Z_0) \vdash^* (f, \epsilon, \alpha)\}$ for final state f and any α.

   N(P): P is a PDA, $N(P) = \{w|(q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon)\}$ for any state q.

   These two definitions are equal.

4. Proof $L(P) = L(G)$:
   - $(q, wx, S) \vdash^* (q, x, \alpha)$ for any x if and only if $S \Rightarrow^*_{lm} w\alpha$.
   - Only if: an induction on the number of steps made by P
   - If: an induction on the number of steps in the leftmost derivation.

5. Equivalence of PDAs and CFGs:
   - CFG(L(G))⟶PDA(N(P)):
     - Input: $w = xy = xA\alpha$
     - ID$(q, y, A\alpha)$
     - Tail: $A\alpha$
     - Transition function δ:
       1. For each variable A: $\delta(q, \epsilon, A) = \{(q, B)|A \rightarrow \beta\, is\ a\ production\ of\ G\}$
       2. For each terminal a, $\delta(q, a, a) = \{(q, \epsilon)\}$
   - PDA(N(P))⟶CFG(L(G)):
     - [pXq]:
       1. A way of describing one variable.
       2. All those strings w that cause P to pop X from its stack while going from state p to state q.
       3. $(p, w, X) \vdash^* (q, \epsilon, \epsilon)$
     - $S \Rightarrow^* w$ if and only if $[q_0Z_0p] \Rightarrow^* w$, and $[q_0Z_0p] \Rightarrow^* w$ if and only if $(q_0, w, Z_0) \vdash^* (p, \epsilon, \epsilon)$
     - The productions of G:
       1. $S \rightarrow [q_0Z_0p]$
       2. $[qXr_k] \rightarrow a[rY_1r_1][r_1Y_2r_2]...[r_{k-1}Y_kr_k]$ if $\delta(q, a, X)$ contain the pair $(r, Y_1Y_2...Y_k)$

Exercise:

6.2.8: A PDA is called restricted if on any transition it can increase the height of the stack by at most one symbol. That is, for any rule $\delta(q, a, Z)$ contains (p, γ), it must be that $|\gamma| \leq 2$. Show that if P is a PDA, then there is a restricted PDA P_3 such that $L(P) = L(P_3)$.

Suppose that there is a rule that $(p, X_1X_2...X_k)$ is a choice in $\delta(q, a, Z)$. We create $k-2$ new states $r_1, r_2, ...r_{k-2}$ that simulate this rule but do so by adding one symbol at a time to the stack. That is, replacing $(p, X_1X_2...X_k)$ in the rule by $(r_{k-2}, X_{k-1}X_k)$. Then create new rules $\delta(r_{k-2}, \epsilon, X_{k-1}) = \{(r_{k-3}, X_{k-2}X_{k-1})\}$, and so on, down to $\delta(r_2, \epsilon, X_3) = \{(p, X_1X_2)\}$

6.3.1: Convert the grammar: $S \rightarrow 0S1|A, A \rightarrow 1A0|S|\epsilon$, to a PDA that accepts the same language by empty stack.

$(\{q\}, \{0, 1\}, \{0, 1, A, S\}, \delta, q, S, q)$ where δ is defined by:

1. $\delta(q, \epsilon, S) = \{(q, 0S1), (q, A)\}$
2. $\delta(q, \epsilon, A) = \{(q, 1A0), (q, S), (q, \epsilon)\}$
3. $\delta(q, 0, 0) = \{(q, \epsilon)\}$
4. $\delta(q, 1, 1) = \{(q, \epsilon)\}$

6.3.3: Convert the PDA P=({p,q},{0,1},{X,Z_0},δ,q,Z_0) to a CFG, if δ is given by:

$$1.\ \delta(q, 1, Z_0) = \{(q, XZ_0)\}.$$

$$2.\ \delta(q, 1, X) = \{(q, XX)\}.$$

$$3.\ \delta(q, 0, X) = \{(p, X)\}.$$

$$4.\ \delta(q, \epsilon, X) = \{(q, \epsilon)\}.$$

$$5.\ \delta(p, 1, X) = \{(p, \epsilon)\}.$$

$$6.\ \delta(p, 0, Z_0) = \{(q, Z_0)\}.$$

1. $S \rightarrow [qZ_0q]|[qZ_0p]$
2. The following four productions come from rule (1).

   [qZ_0q] $\rightarrow$ 1[qXq][qZ_0q]

   [qZ_0q] $\rightarrow$ 1[qXp][pZ_0q]

   [qZ_0p] $\rightarrow$ 1[qXq][qZ_0p]

   [qZ_0p] $\rightarrow$ 1[qXp][pZ_0p]
3. The following four productions come from rule (2).

   [qXq] $\rightarrow$ 1[qXq][qXq]

   [qXq] $\rightarrow$ 1[qXp][pXq]

   [qXp] $\rightarrow$ 1[qXq][qXp]

   [qXp] $\rightarrow$ 1[qXp][pXp]
4. The following two productions come from rule (3).

   [qXq] $\rightarrow$ 0[pXq]

   [qXp] $\rightarrow$ 0[pXp]
5. The following production come from rule (4).

   [qXq]$\rightarrow$ ϵ
6. The following production come from rule (5).

   [pXp]$\rightarrow$1
7. The following two productions come from rule (6).

   [pZ_0q] $\rightarrow$ 0[qZ_0q]

   [pZ_0p] $\rightarrow$ 0[qZ_0p]

6. Properties of CFL:

1. Chomsky Normal Form(safe order $O(n^2)$):
   1. Eliminate ϵ-productions
   2. Eliminate unit productions
   3. Eliminate useless symbols: variables that derive nothing and unreachable symbols
   4. CNF: all productions are in one of two simple forms: $A \rightarrow BC$ or $A \rightarrow a$

   If G is a CFG whose language contains at least one string other than ϵ, then there is a grammar G_1 in Chomsky Normal Form, such that $L(G_1) = L(G) - \{\epsilon\}$

2. Pumping lemma for CFL:
   - Let L be a CFL. Then there exists a constant n such that if z is any string in L such that $|z|$ is at least n, then we can write $z = uvwxy$, subject to the following conditions:
     1. $|vwx| \leq n$. That is, the middle portion is not too long.
     2. $vx \neq \epsilon$. Since v and x are the pieces to be pumped, this condition says that at least one of the strings we pump must not be empty.
     3. For all $i \geq 0$, uv^iwx^iy is in L. That is, the two strings v and x may be "pumped" any number of times, including 0, and the resulting string will still be a member of L.
   - Prove a language is not a CFL: contrapositive
3. Decision Properties: membership(CYK:$O(n^3)$), empty($O(n)$), infinite
4. Non-Decision Properties: equivalence, disjoint
5. Closure Properties: union, concatenation, closure(*), positive closure(+), homomorphism, reversal, inverse homomorphism, L/a
6. Non-closure properties:
   - intersection, complement, subtraction, min/max
   - Give a counter example
   - If L is a CFL and R is a regular language, then
     1. $L \bigcap R$ is a CFL
     2. $L - R$ is a CFL

Exercise:

7.1.2: Begin with the grammar. Put the resulting grammar into Chomsky Normal Form.

$$S \rightarrow ASB|\epsilon$$
$$A \rightarrow aAS|a$$
$$B \rightarrow SbS|A|bb$$

1. Eliminate ϵ-productions

$$S \rightarrow ASB|AB$$
$$A \rightarrow aAS|aA|a$$
$$B \rightarrow SbS|bS|Sb|b|A|bb$$

2. Eliminate unit productions

$$S \rightarrow ASB|AB$$
$$A \rightarrow aAS|aA|a$$
$$B \rightarrow SbS|bS|Sb|b|aAS|aA|a|bb$$

3. Eliminate useless symbols

   no useless symbols

4. Construct CNF

$$\begin{array}{rc} S \rightarrow & AE|AB \\ A \rightarrow & CF|CA|a \\ B \rightarrow & SG|DS|SD|b|CF|CA|a|DD \\ C \rightarrow & a \\ D \rightarrow & b \\ E \rightarrow & SB \\ F \rightarrow & AS \\ G \rightarrow & DS \end{array}$$

7.2.1: Use the CFL pumping lemma to show each of these languages not to be context-free

a) $\{a^ib^jc^k|i < j < k\}$

Let n be the pumping-lemma constant and consider $z = a^n b^{n+1} c^{n+2}$. We may write $z = uvwxy$, $|vwx| \leq n$.

1. If vwx doesn't have c's, then uv^3wx^3y has at least n+2 a's or b's, and thus could not be in L.
2. If vwx has a c, then it could not have an a, because its length is limited to n. Thus, uwy has n a's, but no more than 2n+2 b's and c's in total. Thus uwy is not in L.

So we have a contradiction no matter how z is broken into uvwxy.

7. Turing Machines

1. TM:
    - Seven parts: $Q, \sum, \Gamma, \delta, q_0, B, F$
    - $\delta(q, Z)$ is either undefined or a triple of the form (p, Y, D), p is a state, Y is the new tape symbol, and D is a direction, L or R.
    - ID:
        - $\alpha q \beta$
        - $\vdash^*$ will be used to indicate zero, one, or more moves of the TM.
        - If $\delta(q, Z) = (p, Y, R)$, then $\alpha q Z \beta \vdash \alpha Y p \beta$
        - If $\delta(q, Z) = (p, Y, L)$, then $\alpha X q Z \beta \vdash \alpha p X Y \beta$
2. $L(M) = \{w | q_0 w \vdash^* I, where\ I\ is\ an\ ID\ with\ a\ final\ state\}$

    $H(M) = \{w | q_0 w \vdash^* I, and\ there\ is\ no\ move\ possible\ from\ ID\ I\}$
    - Equivalence of L(M) and H(M)
3. Multiple Tracks:
    - Each track can hold one symbol, and the tape alphabet of the TM consists of tuples, with one component for each "track".
4. Subroutines:
    - A Turing machine subroutine is a set of states that perform some useful process.
5. Multitape Turing Machines:
    - The control enters a new state which could be the same as the previous state.
    - On each tape, a new tape symbol is written on the cell scanned.
    - Each of the tape heads makes a move, which can be either left, right, or stationary. The heads move independently, so different heads may move in different directions, and some may not move at all.
6. NTM:
    - $\delta(q, X) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots, (q_k, Y_k, D_k)\}$
    - If M_N is a nondeterministic Turing machine, then there is a deterministic Turing machine M_D such that $L(M_N) = L(M_D)$
7. Semi-infinite Tape:
    - Every language accepted by a TM M_2 is also accepted by a TM M_1 with the following restrictions:
        - M_1's head never moves left of its initial position.
        - M_1 never writes a blank.
8. Multistack Machines:
    - We can restrict the tapes of a multitape TM to behave like a stack.
    - A one-stack machine is really a DPDA, while a machine with two stacks can accept any RE language.
9. Recursively Enumerable Languages and Recursive Languages
    - Every language accepted by a TM that always halts is recursive language. Every CFL is a recursive language.
    - Every language accepted by a multitape TM is recursively enumerable
    - If L is a recursive language, so is $\overline{L}$.
    - If both a language L and its complement are RE, then L and its complement is recursive.

Exercise:

8.2.5: Consider the Turing machine $M = (\{q_0, q_1, q_2, q_f\}, \{0,1\}, \{0,1,B\}, \delta, q_0, B, \{q_f\})$.

Informally but clearly describe the language L(M) if δ consists of the following sets of rules:

$$\delta(q_0, 0) = (q_1, 1, R); \delta(q_1, 1) = (q_0, 0, R); \delta(q_1, B) = (q_f, B, R)$$

Regular expression: $(01)^*0$

8. Undecidability

1. L_d:
   - The "diagonalization language", which consists of all those string w such that the TM M whose code is w does not accept when given w as input.
   - L_d is not a recursively enumerable language.
   - L_d has no Turing machine at all that accepts it.
2. L_u:
   - L_u is the set of strings representing a TM and an input accepted by that TM.
   - U simulates M on w. U accepts the coded pair (M, w) if and only if M accepts w.
   - L_u is RE but not decidable/recursive.
3. Encode TM:
   - Transition rule: $\delta(q_i, X_j) = (q_k, X_l, D_m)$. We can code this rule by the string $0^i10^j10^k10^l10^m$
   - A code for the entire TM M consists of all the codes for the transitions, in some order, separated by pairs of 1's: $C_1 11 C_2 11 \cdots C_{n-1} 11 C_n$
   - A TM and a string (M, w): For this pair we use the code for M followed by 111, followed by w
4. Reductions:
   - If we have an algorithm to convert instances of a problem P_1 to instances of a problem P_2 that have the same answer, then we say that P_1 reduces to P_2. P_2 is at least as hard as P_1.
5. Rice's Theorem: Every nontrivial property of the RE languages is undecidable.
   - $L_e = \{M | L(M) = \emptyset\}$ is non-RE, while $L_{ne} = \{M | L(M) \neq \emptyset\}$ is RE.
6. Post's Correspondence Problem(PCP):
   - L_u reduces to MPCP, and MPCP reduces to PCP, so PCP and MPCP are undecidable.
   - PCP reduces to the question of whether a given CFG is ambiguous, so it is undecidable whether a CFG is ambiguous.
   - List languages:
     - $A \to x_i A a_i, B \to x_i B a_i$
     - If L_A is the language for list A, then L_A and $\overline{L_A}$ is a context-free language.
     - Let G_1 and G_2 be context-free grammars, and let R be a regular expression. Then the following are undecidable:

       a) Is $L(G_1) \cap L(G_2) = \emptyset$?

       b) Is $L(G_1) = L(G_2)$?

       c) Is $L(G_1) = L(R)$?

       d) Is $L(G_1) = T^*$ for some alphabet T?

       e) Is $L(G_1) \subseteq L(G_2)$?

       f) Is $L(R) \subseteq L(G_1)$

       ---

       a) Let $L(G_1) = L_A, L(G_2) = L_B$. Then $L(G_1) \cap L(G_2)$ is the set of solutions to this instance of PCP. The intersection is empty if and only if there is no solution.

b), c), d) Let $L(G_1) = \overline{L_A} \cup \overline{L_B} = \overline{L_A \cap L_B}, L(G_2) = (\sum \bigcup I)^*$. Their languages are equal if and only if the PCP instance has no solution.

e), f) Let $L(G_1) = (\sum \bigcup I)^*, L(G_2) = \overline{L_A} \cup \overline{L_B} = \overline{L_A \cap L_B}$. Then $L(G_1) \subseteq L(G_2)$ if and if PCP has no solution.

Exercise:

9.1.3: Show that the language is not accepted by a Turing machine, using a diagonalization-type argument.

a) The set of all w_i such that w_i is not accepted by M_{2i}

Suppose this language were accepted by some TM M. We need to find an i such that $M = M_{2i}$. Since all the codes for TM's end in a 0, that is not a problem.

If w_i is accepted by M_{2i}, then w_i is not accepted by M, and therefore not accepted by M_{2i}, which is the same TM. Similarly, if w_i is not accepted by M_{2i}, then w_i is accepted by M, and therefore by M_{2i}. Either way, we reach a contradiction, and conclude that M does not exist.

9.2.4: Let $L_1, L_2, \cdots, L_k$ be a collection of languages over alphabet $\sum$ such that:

1. For all $i \neq j, L_i \cap L_j = \emptyset$; i.e., no string is in two of the languages.
2. $L_1 \cup L_2 \cup \cdots \cup L_k = \sum^*$, i.e., every string is in one of the languages.
3. Each of the languages L_i for $i = 1, 2, \cdots, k$ is recursively enumerable.

   Prove that each of the languages is therefore recursive.

By symmetry, if we can prove L_1 is recursive, we can prove any of the languages to be recursive. Take TM's $M_1, M_2, \cdots, M_k$ for each of the languages $L_1, L_2, \cdots, L_k$, respectively. Design a TM M with k tapes that accepts L_1 and always halts. M copies its input to all the tapes and simulates M_i on the i-th tape. If M_1 accepts, then M accepts. if any of the other TM's accepts, M halts without accepting. Since exactly one of the M_i's will accept, M is sure to halt.

9.3.8: Tell whether each of the following are recursive, RE-but-not-recursive, or non-RE

d) The set of all TM codes for TM's that fail to halt on at least one input.

We'll show it's non-RE by a reduction from the nonhalting problem, which is non-RE. Given (M, w), construct M' as follows:

1. M' ignores its own input and simulates M on w.
2. If M halts, M' halts on its own input. However, if M never halts on w, then M' will never halt on its own input.

As a result, M' fails to halt on at least one input if M fails to halt on w. If M halts on w, then M' halts on all input.

9. Intractable Problems

1. - $\mathscr{P}$: problems can be solved in polynomial time by deterministic TM's.
   - $\mathscr{NP}$: problems can be solved in polynomial time by nondeterministic TM's. $\mathscr{P} \subseteq \mathscr{NP}$
   - NP-complete:
     - L is in $\mathscr{NP}$.
     - For every language L' in $\mathscr{NP}$, there is a polynomial-time reduction of L' to L.
     - If some $NP-$complete problem P is in $\mathscr{P}$, then $\mathscr{P} = \mathscr{NP}$
2. The Satisfiability Problem:
   - A boolean expression E is said to be satisable if there exists at least one truth assignment T that satises E.
   - The satisability problem is: given a boolean expression, is it satisable?
   - Cook's Theorem: SAT is NP-complete.
     - $E_{M,w} = U \wedge S \wedge N \wedge F$

        - The running time of this algorithm on a multitape, DTM is $O(p^2(n))$, and that multitape TM can be converted to a single-tape TM that runs in time $O(p^4(n))$.
        - We can reduce SAT to CSAT, then reduce CSAT to 3SAT.
    - The independent-set problem is NP-complete.(IS)
        - maximal if it is as large has as many nodes as no two nodes of I are connected by an edge of G
    - The node-cover problem is NP-complete.(NC)
        - minimal if it has as few nodes as any node cover for the given graph.
    - The Directed Hamilton-Circuit Problem and Hamilton-Circuit Problem are NP-complete.(DHC, HC)

Exercise:

10.3.1: Put the following boolean expressions into 3-CNF:

a) $xy + \overline{x}z$

1. Put it into CNF: $(x+u)(y+u)(\overline{x}+\overline{u})(y+\overline{u})$
2. Put it into 3-CNF: $(x+u+v_1)(x+u+\overline{v_1})(y+u+v_2)(y+u+\overline{v_2})(\overline{x}+\overline{u}+v_3)(\overline{x}+\overline{u}+\overline{v_3})(y+\overline{u}+v_4)(y+\overline{u}+\overline{v_4})$

d) $wx + \overline{x}y + u + v + w$

1. Put it into CNF: $(w+a+b+c+d)(x+a+b+c+d)(\overline{x}+\overline{a}+b+c+d)(y+\overline{a}+b+c+d)(u+\overline{b}+c+d)(v+\overline{c}+d)(w+\overline{d})$
2. Put it into 3-CNF:
$$(w+a+e_1)(b+\overline{e_1}+e_2)(c+d+\overline{e_2})(x+a+e_3)(b+\overline{e_3}+e_4)(c+d+\overline{e_4})(\overline{x}+\overline{a}+e_5)(b+\overline{e_5}+e_6)$$
$$(c+d+\overline{e_6})(y+\overline{a}+e_7)(b+\overline{e_7}+e_8)(c+d+\overline{e_8})(u+\overline{b}+e_9)(c+d+\overline{e_9})(v+\overline{c}+d)(w+\overline{d}+e_{10})(w+\overline{d}+\overline{e_{10}})$$

Hint:

Suppose $e_i = (x_1 + x_2 + \cdots + x_m)$ for some $m \geq 4$. Introduce new variables $y_1, y_2, \ldots, y_{m-3}$ and replace e_i by the product of clauses

$$\begin{aligned}&(x_1 + x_2 + y_1)(x_3 + \overline{y_1} + y_2)(x_4 + \overline{y_2} + y_3)\cdots \\ &(x_{m-2} + \overline{y_{m-4}} + y_{m-3})(x_{m-1} + x_m + \overline{y_{m-3}})\end{aligned} \tag{10.2}$$

10. Additional Classes of Problems

1. co-$\mathscr{NP}$:
    - The class of complements of $\mathscr{NP}$.
    - $\mathscr{NP}$=co-$\mathscr{NP}$ if and only if there is some NP-complete problem whose complement is in $\mathscr{NP}$
2. PS :
    - all the problems that can be solved by a Turing machine using an amount of tape that is polynomial in the length of its input.
    - Savitch's theorem: $PS = NPS$
    - PS-complete:
        - P is in PS
        - All languages L in PS are polynomial-time reducible to P.
    - Suppose P is a PS-complete problem. Then:

      a) If P is in $\mathscr{P}$, then $\mathscr{P} = PS$

      b) If P is in $\mathscr{NP}$, then $\mathscr{NP} = PS$
3. RP:
    - have an algorithm that runs in polynomial time□ using a random-number generator.
    - $RP \subseteq \mathscr{NP}$

- Both the primes and the complement of the language of primes—the composite numbers—are in NP. The composite numbers are in RP.

4. ZPP (zero-error, probabilistic, polynomial):
   - The class is based on a randomized TM that always halts, and has an expected time(rather than the worst-case running time) to halt that is some polynomial in the length of the input.
   - $ZPP = RP \bigcap$ co-RP
   - $\mathscr{P} \subseteq ZPP$