1.1 Three areas of ToC (*Sipser 0.1*)
Theory of Computation is to study the fundamental capabilities and limitations of computers. It contains three areas.

- Automata theory: Models of computation. Seeking a precise but concise definition of a computer.

- Computability theory: What can and cannot a computer do? Computationally unsolvable versus computationally solvable problems. Determining whether a mathematical statement is true or false is unsolvable by any computer algorithms.

- Complexity theory: What can a computer do efficiently? Computationally hard versus computationally easy problems. For example, factorization and sorting. Cryptography needs hard problems such as factorization to ensure security.
1.2 Basic math (Sipser 0.2)
Sets, sequences, functions, relations, graphs, and logic. Here is a discussion on strings and languages, which are central concepts in automata theory.

▶ Alphabet $\Sigma$: Finite and nonempty, e.g., $\Sigma = \{0, 1\}$ and $\Sigma = \{a, b, \ldots, z\}$.

▶ String (or word), e.g., $w = 01110$, the empty string $\varepsilon$, the length of the string, $|w|$, the concatenation of two strings $w_1w_2$, the reverse of a string $w^R$, and the substring of a string.

▶ Language: A language is a set of strings, e.g., $\{\varepsilon\}$, $\emptyset$, $\Sigma$, $A = \{w | w$ has an equal number of 0s and 1s$\}$, and $B = \{0^n1^n | n \geq 1\}$. 
Regular operators: Let $A$ and $B$ be two languages.

- **Union:** $A \cup B = \{ x \mid x \in A \text{ or } x \in B \}$.
- **Concatenation:** $AB = \{ xy \mid x \in A \text{ and } y \in B \}$.
- **Star:** $A^* = \{ x_1 x_2 \cdots x_k \mid \text{all } k \geq 0 \text{ and each } x_i \in A \}$. $A^*$ is infinite unless $A = \emptyset$ or $A = \{ \varepsilon \}$.

Power of a language: Let $A$ be a language.

- $A^k = \{ x_1 x_2 \cdots x_k \mid x_i \in A \text{ for } i = 1, 2, \cdots, k \}$
- $A^0 = \{ \varepsilon \}$
- $A^* = A^0 \cup A^1 \cup A^2 \cup \cdots$
- $A^+ = A^1 \cup A^2 \cup \cdots$
- $A^* = A^+ \cup \{ \varepsilon \}$

Why languages, not problems:

- **Decision problems:** Given an input $x$, does $x$ satisfy property $P$, or is $x \in \{ y \mid y \text{ satisfies } P \}$?
- **Membership in a language:** Given a language $A$ and a string $w \in \Sigma^*$, is $w$ a member of $A$, or is $w \in A$?
1.3 Common forms of theorems (*Sipser 0.3*)

- If $H$ (hypothesis) then $C$ (conclusion). Also equivalently, $H$ implies $C$, $H$ only if $C$, $C$ if $H$, whenever $H$ holds, $C$ follows, or $H \rightarrow C$.

- $A$ if and only if $B$. Also equivalently, $A$ iff $B$, $A$ is equivalent to $B$, $A$ exactly when $B$, $A \equiv B$, or $A \leftrightarrow B$.

- $S$ (a statement without any hypothesis).

- For all $x$, $S(x)$ holds. Also equivalently, $\forall x S(x)$.

- There is $x$ such that $S(x)$ holds. Also equivalently, $\exists x S(x)$.
1.4 Proof techniques *(Sipser 0.4)*

- By definition: Convert terms in the hypothesis to their definitions.
- By construction: To prove the existence of $X$, construct it.
- By counterexample: Used to disprove something seemingly true but actually not true. **Example:** All primes are odd. **Example:** There is no integers $a$ and $b$ such that $a \mod b = b \mod a$.
- By contradiction: $H \rightarrow C$ is equivalent to $\neg C \rightarrow \neg H$ or $\neg C \land H \rightarrow \neg T$, where $T$ is an axiom, a known truth, or a proven fact. **Example:** The number of primes is infinite. **Example:** $\sqrt{2}$ is irrational.
By induction: Used to prove a statement $S(n)$ about an integer $n \geq c$. There are several forms of inductions:

- **Simple integer induction:** $S(n)$ for $n \geq c$ iff $S(c) \land \forall k(S(k) \rightarrow S(k + 1))$. (What are the basis step, inductive hypothesis, and inductive step?)
  **Example:** For $n \geq 1$, $\sum_{i=1}^{n} i^2 = \frac{1}{6} n(n + 1)(2n + 1)$.

- **General (strong) integer induction:** $S(n)$ for $n \geq c$ iff $S(c) \land \forall k(S(c) \land S(c + 1) \land \cdots \land S(k) \rightarrow S(k + 1))$.
  **Example:** For any $n \geq 8$, $n$ can be written as a sum of 3s and 5s.

- **Structural induction:** To prove a property $S(X)$ of a recursively defined structure $X$, it is suffice to prove that $S(X)$ is true for the basis structures $X$ and $S(Y_1) \land S(Y_2) \land \cdots \land S(Y_k) \rightarrow S(X)$, where $X$ is constructed from $Y_1, Y_2, \ldots, Y_k$.
  **Example:** Every tree has one more node than it has edges.
2.1 Example of FA *(Sipser 1.1 (pp. 31-34))*

- A farmer (F) with a cabbage (C), a dog (D), and a goat (G) wants to cross a river. There is a small boat which can only carry the farmer plus one of the three things. At any time, the cabbage and the goat cannot be left alone neither can the dog and the goat. How can the farmer cross the river? A finite automaton with start state FCDG-∅ and accept state 0-FCDG can solve the puzzle.
2.2 DFA (Sipser 1.1 (pp. 35-44))

- DFA $M = (Q, \Sigma, \delta, q_0, F)$, where
  - $Q$ is a finite set of states,
  - $\Sigma$ is an alphabet,
  - $q_0 \in Q$ is the start state,
  - $F \subseteq Q$ is a set of accept/final states, and
  - $\delta : Q \times \Sigma \rightarrow Q$ is a transition function, where $\delta(q, a) = p$ is the next state of $M$ if the current state is $q$ and the current symbol is $a$. 

Extending $\delta$ to $\hat{\delta}: Q \times \Sigma^* \to Q$: For any $q \in Q$ and any $w = xa \in \Sigma^*$, define $\hat{\delta}(q, x)$ recursively as below:

- $\hat{\delta}(q, \epsilon) = q$ and
- $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$

Language of a DFA $M$ (or language recognized by $M$) is $L(M) = \{ w | \hat{\delta}(q_0, w) \in F \}$. A language is called a regular language if some DFA recognizes it.

How does a DFA accept/reject a string?
Example: A DFA that accepts strings over alphabet \{0, 1\} that have even numbers of 0s and 1s. (Given a language \(A\), what is the DFA \(M\) such that \(L(M) = A\)?)
Example (Sipser p. 36): A DFA that accepts strings over the alphabet \{0, 1\} that have at least one 1 and an even number of 0s after the last 1. (Given a DFA $M$, what is $L(M)$?)
Remarks:

- State diagrams and transition tables are other representations of DFAs.
- Although rigorously, $\delta(q, a)$ should be defined $\forall q \in Q$ and $\forall a \in \Sigma$, all inaccessible and dead-end states with associated arcs may be removed to simplify the DFA.
2.3 NFA (Sipser 1.2 (pp. 47-54))

- NFA \( N = (Q, \Sigma, \delta, q_0, F) \), where

  - \( Q \) is a finite set of states,
  - \( \Sigma \) is an alphabet,
  - \( q_0 \in Q \) is the start state,
  - \( F \subseteq Q \) is a set of accept/final states, and
  - \( \delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q \) is a transition function, where \( \delta(q, a) = P \) is the set of states that \( N \) may enter if the current state is \( q \) and the current symbol is \( a \). In the case of \( \delta(q, \varepsilon) = P \), \( N \) ignores the current input symbol and transitions from the current state \( q \) to any state in \( P \).
- **ε-closure:** For any $P \subseteq Q$, $E(P)$ is the set of all states reachable from any state in $P$ via zero or more ε-transitions.

- **Extending δ to $\hat{\delta} : Q \times \Sigma^* \rightarrow 2^Q$:** For any $q \in Q$ and any $w = xa \in \Sigma^*$, define
  - $\hat{\delta}(q, \varepsilon) = E(\{q\})$ and
  - $\hat{\delta}(q, w) = E(\bigcup_{i=1}^{k} \delta(p_i, a))$ if $w = xa$ and $\hat{\delta}(q, x) = \{p_1, \ldots, p_k\}$.

- **Language of an NFA $N$ (or language recognized by $N$) is $L(N) = \{w | \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$.**

- How does an NFA accept/reject a string? The meaning of nondeterminism.
Example: An NFA that accepts decimal numbers (a number that may have + or – preceding it, but must have a decimal point).
2.4 DFAs ⇔ NFAs (*Sipser 1.2 (pp.54-58)*)

Subset construction method: Given NFA $N = (Q, \Sigma, \delta, q_0, F)$, construct a DFA $M = (Q', \Sigma, \delta', q'_0, F')$ such that $L(M) = L(N)$.

- $Q' = 2^Q$ (power set), i.e., $Q'$ contains all subsets of $Q$. Note that if $|Q| = n$ then $|Q'| = 2^n$. This is just the worst case. Since many states in $M$ are inaccessible or dead-end states and thus may be thrown away, so in practice, $|Q'|$ may be much less than $2^n$.
- $q'_0 = E(\{q_0\})$.
- $F' = \{R \in Q' | R \cap F \neq \emptyset\}$.
- For each $R \in Q'$ and each $a \in \Sigma$, $\delta'(R, a) = E(\cup_{p \in R} \delta(p, a))$.

**Theorem**: The equivalence of DFAs, NFAs, and RLs.
**Example**: A DFA that accepts decimal numbers (converted from the previous NFA).

**Example** (Sipser p. 51): A four-state NFA and an eight-state DFA that recognize the language consisting of strings over \{0, 1\} with a 1 in the third position from the end.

**Example**: A bad case for the subset construction: \(|Q_N| = n + 1\) and \(|Q_D| = 2^n\).
2.5 Closure Properties of RL’s (Sisper 1.1 (pp. 44-47) and 1.2 (pp. 58-63))

▶ Union: If $A$ and $B$ are regular, so is $A \cup B$.
▶ Concatenation: If $A$ and $B$ are regular, so is $AB$.
▶ Star: If $A$ is regular, so is $A^*$. (Need a new start state.)
▶ Complementation: If $A$ is regular, so is $\overline{A}$ (which is $\Sigma^* - A$).
▶ Intersection: If $A$ and $B$ are regular, so is $A \cap B$.
▶ Difference: If $A$ and $B$ are regular, so is $A - B$.
▶ Reverse: If $A$ is regular, so is $A^R$.
▶ Homomorphism: If $A$ is regular, so is $h(A)$ (which is $\{h(w) | w \in A\}$ for a homomorphism $h : \Sigma \rightarrow (\Sigma')^*$).
▶ Inverse homomorphism: If $A$ is regular, so is $h^{-1}(A)$ (where $h^{-1}(A) = \{w | h(w) \in A\}$).
Example: Prove that $A = \{ w \in \{a, b\}^* | w \text{ is of odd length and contains an even number of } a\text{'s} \}$ is regular. ($A$ is the intersection of two regular languages.)
3.1 Definition of REs *(Sipser 1.3 (pp. 63-66))*

Regular expressions (REs) are to represent regular languages. Let $L(R)$ be the language that regular expression $R$ represents. A recursive definition is given below:

- **Basis**: $\varepsilon$ and $\emptyset$ are REs, and $L(\varepsilon) = \{\varepsilon\}$ and $L(\emptyset) = \emptyset$. For any $a \in \Sigma$, $a$ is an RE and $L(a) = \{a\}$.

- **Induction**: If $R_1$ and $R_2$ are REs, then
  - $R_1 \cup R_2$ is an RE, with $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$,
  - $R_1 R_2$ is an RE, with $L(R_1 R_2) = L(R_1)L(R_2)$,
  - $R_1^*$ is an RE, with $L(R_1^*) = (L(R_1))^*$, and
  - $(R_1)$ is an RE, with $L((R_1)) = L(R_1)$. 
Remark:

- Precedence order for regular-expression operators: Star, concatenation, and finally union. () may override this order.
- Use of $R^+$ and $R^k$.
- Algebraic laws:
  - $R_1 \cup R_2 = R_2 \cup R_1$, $(R_1 \cup R_2) \cup R_3 = R_1 \cup (R_2 \cup R_3)$, and $(R_1 R_2) R_3 = R_1 (R_2 R_3)$.
  - $\emptyset \cup R = R \cup \emptyset = R$, $\varepsilon R = R \varepsilon = R$, $\emptyset R = R \emptyset = \emptyset$, and $R \cup R = R$.
  - $R_1 (R_2 \cup R_3) = R_1 R_2 \cup R_1 R_3$ and $(R_1 \cup R_2) R_3 = R_1 R_3 \cup R_2 R_3$.
  - $R^* = R^*$, $\emptyset^* = \varepsilon$, $R^+ = RR^* = R^* R$, and $R^* = R^+ \cup \varepsilon$. 
Examples:
1. A regular expression for the language of strings that consist of alternating 0s and 1s: \((01)^* \cup (10)^* \cup 0(10)^* \cup 1(01)^*\).
2. A regular expression for the language of strings with a 1 in the third position from the end: \((0+1)^*1(0+1)(0+1)\).
3.2 RE \Rightarrow NFA (Sipser 1.3 (pp. 67-69))
Since REs are defined recursively, it is suitable to construct the equivalent NFAs recursively.

- **Basis**: The finite automata for regular expressions \( \varepsilon, \emptyset, \) and \( a \) for \( a \in \Sigma \).
- **Induction**: Given the NFAs for REs \( R_1 \) and \( R_2 \), what are the NFAs for \( R_1 \cup R_2, R_1 R_2, \) and \( R_1^* \)?

**Example**: Convert regular expression \((0 \cup 1)^* 1 (0 \cup 1)\) to a finite automaton.
3.3 DFA \Rightarrow RE (Sipser 1.3 (pp. 69-76))
A generalized NFA (GNFA) is an NFA with REs (not symbols) on its transition arcs.
Assume that the given finite automaton is a DFA \( M = (Q, \Sigma, \delta, q_0, F) \). We first convert the DFA to a GNFA by
1. adding a new start state \( s \) that goes to the old start state \( q_0 \) via an \( \varepsilon \)-transition,
2. adding a new accept state \( a \) to which there is an \( \varepsilon \)-transition from each old accept state in \( F \), and
3. converting symbols to regular expressions on all arcs.
Then this GNFA with \( |Q| + 2 \) states will be converted to an equivalent GNFA with \( |Q| + 1 \) states by eliminating a state that is neither \( s \) nor \( a \). This state elimination step will be applied a total of \( |Q| \) times until there are only states \( s \) and \( a \) left in the resulting GNFA. The regular expression on the arc from \( s \) to \( a \) is the regular expression for the original DFA.
Given a GNFA with \( k \) states, how can one convert it to an equivalent GNFA with \( k - 1 \) states by eliminating a state that neither \( s \) nor \( a \)? (Sipser Figure 1.63 on p.72)
**Example** (Siper p. 76): A three-state DFA to be converted to a regular expression.

**Example**: A language of all strings over \( \{0, 1\} \) with one 1 either two or three positions from the end.

**Theorem**: The equivalence of RLs, REs, and FAs.
4.1 Regular versus nonregular languages

- \( A = \{0^*1^*\} \)
- \( B = \{0^n1^n | n \geq 0\} \)
- \( C = \{w \in \{0, 1\}^* | w \text{ has an equal number of 0s and 1s}\} \)
- \( D = \{w \in \{0, 1\}^* | w \text{ has an equal \# of substrings 01 and 10}\} \)
4.2 Proving nonregularity by pumping lemma (Sisper 1.4 (pp. 77-82))

**Theorem** (The pumping lemma for regular languages): Let $A$ be a regular language. Then there exists a constant $p$ (the pumping length which depends on $A$) such that $\forall s \in A$ with $|s| \geq p$, we can break $s$ into three substrings $s = xyz$ such that

1. $|y| > 0$;
2. $|xy| \leq p$; and
3. $\forall i \geq 0$, string $xy^i z \in A$. 
Proof: Let $A = L(M)$ for some DFA $M$ with $p$ states. Consider any $s \in A$ with $s = s_1 s_2 \cdots s_n$, where $n \geq p$ and $s_i \in \Sigma$ for $i = 1, 2, \ldots, n$. Assume that $\hat{\delta}(q_0, s_1 \cdots s_i) = q_i$. On the path $q_0 \to q_1 \to \cdots \to q_n$ that accepts $s$, there are $n + 1 \geq p + 1$ states. By the pigeonhole principle, there are at least two identical states on the path. Let $q_i = q_j$ for some $0 \leq i < j \leq n$ be the first such pair on the path. Now we can break $s = xyz$ as follows:

- $x = s_1 \cdots s_i$
- $y = s_{i+1} \cdots s_j$; and
- $z = s_{j+1} \cdots s_n$.

We can then easily verify that this partition of $s$ satisfies all three requirements stated in the theorem, i.e., $|y| > 0$, $|xy| \leq p$, and $xy^iz \in A$ for any $i \geq 0$. This completes the proof.
How to use the pumping lemma to prove that a language is not regular:

- Assume that $A$ was regular by contradiction. Then the pumping lemma applies to $A$. Let $p$ be the constant in the pumping lemma.
- Select $s \in A$ with $|s| = f(p) \geq p$.
- By the pumping lemma, $\exists x, y, z$ such that $s = xyz$ with $|y| > 0$, $|xy| \leq p$ and $xy^i z \in A$ for any $i \geq 0$.
- For any $x, y, z$ such that $s = xyz$, $|y| > 0$, and $|xy| \leq p$, find $i \geq 0$ such that $xy^i z \not\in A$. A contradiction!
Example (Sipser p. 80): Prove that $B = \{0^n1^n| n \geq 0\}$ is not regular.

Example: Prove that $A = \{1^r | r \text{ is a prime}\}$ is not regular.

Example (Sipser p. 81): Prove that $A = \{ww | w \in \{0, 1\}^*\}$ is not regular.

Example: Prove that $L = \{1^{n^2} | n \geq 1\}$ is not regular.

Example: Prove that $A = \{10^n1^n | n \geq 0\}$ is not regular.

Example: Prove that $A = \{(01)^a0^b | a > b \geq 0\}$ is not regular.

Example: Prove that $A = \{0^m1^n | m \neq n\}$ is not regular.
4.3 Prove nonregularity by closure properties

To prove that $A$ is not regular, assume it was. Find a regular language $B$ and a language operator that preserves regularity, and then apply the operator on $A$ and $B$ to get a regular language $C$. If $C$ is known to be nonregular, a contradiction is found.

**Example:** Prove that $C = \{ w \in \{0, 1\}^* | w \text{ has an equal } \# \text{ of 0s and 1s} \}$ is not regular.

**Example:** Prove that $A = \{0^m1^n | m \neq n\}$ is not regular.

**Example:** Prove that $A = \{a^mb^nc^{m+n} | m, n \geq 0\}$ is not regular.
5.1 Context-free grammars *(Sipser 2.1 (pp. 100-105))*

- CFG $G = (V, \Sigma, R, S)$, where $V$ is the set of variables, $\Sigma$ is the set of terminals (alphabet), $R$ is the set of rules in the form of $V \rightarrow (V \cup \Sigma)^*$ (head→body), and $S \in V$ is the start variable.
- The CFG that generates all palindromes (strings that read the same forward and backward) over $\{0, 1\}$ is $G = (\{S\}, \{0, 1\}, R, S)$, where $R$ contains $S \rightarrow \varepsilon | 0 | 1 | 0S0 | 1S1$. 
Let $u, v, w$ be strings in $(V \cup \Sigma)^*$. If $A \to w$ is a rule, then $uAv$ yields $uwv$, written $uAv \Rightarrow uwv$. We say $u$ derives $v$, written $u \Rightarrow^* v$, if $\exists u_1, \ldots, u_k \in (V \cup \Sigma)^*$ such that $u \Rightarrow u_1 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v$. Here, $\Rightarrow$ means one step and $\Rightarrow^*$ means zero or more steps.

- Leftmost and rightmost derivations: $\Rightarrow_{lm}, \Rightarrow^*_{lm}, \Rightarrow_{rm}, \Rightarrow^*_{rm}$.

- The language of a CFG $G$, $L(G) = \{ w \in \Sigma^* | S \Rightarrow^* w \}$. $L(G)$ is said to be a CFL.

- Why called “context-free”? 
Example: Is $L = \{0^n1^n | n \geq 0\}$ a context-free language? Yes since it is the language of the context-free grammar $S \rightarrow 0S1|\varepsilon$.

Example: What is the language for grammar $G$ with $S \rightarrow AA$ and $A \rightarrow AAA|bA|Ab|a$? $L(G) = \{w \in \{a, b\}^* | w$ has an even (nonzero) number of $a$s$\}$.

Example: A CFG for simple expressions in programming languages: $E \rightarrow E + E | E * E | (E) | I$ and $I \rightarrow Ia | Ib | I0 | I1 | a | b$. 
5.2 Parse trees

- A parse tree is a tree representation for a derivation, in which each interior node is a variable, each leaf node is either a variable, a terminal, or $\epsilon$, and if an interior node is a variable $A$ and its children are $X_1, \ldots, X_k$, then there must be a rule $A \rightarrow X_1 \cdots X_k$.

- Yield of a parse tree: Concatenation of the leaf nodes in a parse tree rooted at the start variable.

- Four equivalent notions:
  1. $A \Rightarrow^* w$;
  2. $A \Rightarrow^* \overset{lm}{w}$;
  3. $A \Rightarrow^* \overset{rm}{w}$; and
  4. A parse tree with root $A$ and yield $w$. 
5.3 Regular grammars

- A regular grammar (RG) is a special CFG where the body of each rule contains at most one variable which, if present, must be the last symbol in the body.
- For each regular language $L$ there is a RG $G$ such that $L = L(G)$.
- Given RG $G = (V, \Sigma, R, S)$, construct an equivalent NFA $N = (Q, \Sigma, \delta, q_0, F)$, where $Q = V \cup \{f\}$, $q_0 = S$, $F = \{f\}$, and for each rule $A \rightarrow wB$, $\delta(A, w) = B$ and for each rule $A \rightarrow w$, $\delta(A, w) = f$.

**Example:** Given RG: $S \rightarrow aS|bA|aB$, $A \rightarrow aS|\varepsilon$, and $B \rightarrow bS|\varepsilon$, what is its NFA?
Given DFA $A = (Q, \Sigma, \delta, q_0, F)$, define an equivalent RG $G = (V, \Sigma, R, S)$, where $V = Q$, $S = q_0$, and for each transition $\delta(q, a) = p$, $q \rightarrow ap$ and for each final state $q \in F$, $q \rightarrow \varepsilon$.

**Example**: Given a four-state DFA for language $aa^* bb^* a$, what is its regular grammar?
5.4 Ambiguity in grammars and languages (Sipser 2.1 (pp. 105-106))

- A CFG $G = (V, \Sigma, R, S)$ is ambiguous if there is $w \in \Sigma^*$ for which there are at least two parse trees (or leftmost derivations).

- Grammar $G$: $E \rightarrow E + E | E \cdot E | (E) | I$ and $I \rightarrow Ia | Ib | l0 | l1 | a | b$ is ambiguous since $a + b \cdot a$ has two parse trees.

- Some ambiguous grammars have an equivalent unambiguous grammar. For example, an unambiguous grammar for the simple expressions is $G': E \rightarrow E + T | T$, $T \rightarrow T \cdot F | F$, $F \rightarrow (E) | I$, and $I \rightarrow Ia | Ib | l0 | l1 | a | b$. 
A context-free language is said to be inherently ambiguous if all its grammars are ambiguous. For example, 
\[ \{a^n b^m c^m d^m | m, n \geq 1\} \cup \{a^n b^m c^m d^n | m, n \geq 1\} \]. Its grammar is 
\[ S \rightarrow S_1 | S_2, \quad S_1 \rightarrow AB, \quad A \rightarrow aAb | ab, \quad B \rightarrow cBd | cd, \]
\[ S_2 \rightarrow aS_2 d | aCd, \quad \text{and} \quad C \rightarrow bCc | bc. \]
The grammar is ambiguous (considering \(abcd\)). There is a not so easy proof that all grammars for the language are ambiguous, thus it is inherently ambiguous.

There is no algorithm to determine whether a given CFG is ambiguous. There is no algorithm to remove ambiguity from an ambiguous CFG. There is no algorithm to determine whether a given CFL is inherently ambiguous.
5.5 Chomsky normal form *(Sipser 2.1 (pp. 106-109))*
For a CFL, different people may come up with different but equivalent CFGs. There are several steps to simplify a CFG.

- Eliminating $\varepsilon$ rules for $L(G) - \{\varepsilon\}$ by finding nullable variables ($A \xrightarrow{*} \varepsilon$). For example, $S \rightarrow AB$, $A \rightarrow aAA|\varepsilon$, and $B \rightarrow bBB|\varepsilon$ can be changed to $S \rightarrow AB|A|B$, $A \rightarrow aAA|aA|a$, and $B \rightarrow bBB|bB|b$.

- Eliminating unit rules. For example, $E \rightarrow T|E + T$, $T \rightarrow F|T * F$, $F \rightarrow I|(E)$, and $I \rightarrow la|lb|l0|l1|a|b$ can be changed to $E \rightarrow E + T|T * F|(E)|la|lb|l0|l1|a|b$, $T \rightarrow T * F|(E)|la|lb|l0|l1|a|b$, $F \rightarrow (E)|la|lb|l0|l1|a|b$, and $I \rightarrow la|lb|l0|l1|a|b$.

- Eliminating useless variables (and thus associated rules) by finding nongenerating and unreachable variables. For example, $S \rightarrow AB|a$ and $A \rightarrow b$ can be simplified to $S \rightarrow a$. 
The Chomsky Normal Form (CNF): Any nonempty CFL without \(\varepsilon\) has a CFG \(G\) in which all rules are in one of the following two forms: \(A \rightarrow BC\) and \(A \rightarrow a\), where \(A, B, C\) are variables, and \(a\) is a terminal. Note that one of the uses of CNF is to turn parse trees into binary trees.

To convert a CFG to a grammar in CNF:

- Add a new start variable \(S_0\) in the case when the old start variable \(S\) appears in the body of some rules.
- Simplify the grammar by removing \(\varepsilon\) rules, unit rules, and useless variables.
- Convert the rules in the simplified grammar into the proper forms of CNF by adding additional variables and rules.

**Example** (Sipser p. 108): Given a CFG \(G\), construct a CNF \(G'\) such that \(L(G) - \{\varepsilon\} = L(G')\).
6.1 PDAs (*Sipser 2.2 (pp. 102-114)*)

- PDA = NFA + Stack (still with limited memory but more than that in FAs)
- PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where
  - $Q$: A finite set of states
  - $\Sigma$: A finite set of input symbols (input alphabet)
  - $\Gamma$: A finite set of stack symbols (stack alphabet)
  - $\delta$: The transition function from $Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\})$ to $2^{Q \times (\Gamma \cup \{\varepsilon\})}$
  - $q_0$: The start state
  - $F$: The set of final states
What does $\delta(q, a, X) = \{(p, Y)\}$ mean? If the current state is $q$, the current input symbol is $a$, and the stack symbol at the top of the stack is $X$, then the automaton changes to state $p$ and replace $X$ by $Y$. What if $\varepsilon$ replaces $a$, or $X$, or $Y$?

The state diagram of PDAs: For transition $\delta(q, a, X) = \{(p, Y)\}$, draw an arc from state $q$ to state $p$ labeled with $a, X \rightarrow Y$. 
Instantaneous description (ID) of a PDA: \((q, w, \gamma)\) represents the configuration of a PDA in the state of \(q\) with the remaining input of \(w\) yet to be read and the stack content of \(\gamma\). (The convention is that the leftmost symbol in \(\gamma\) is at the top of the stack.)

Binary relation \(\vdash\) on ID’s: \((q, aw, X\beta) \vdash (p, w, Y\beta)\) if \(\delta(q, a, X)\) contains \((p, Y)\). \(\vdash\) represents one move of the PDA, and \(\vdash^*\) represents zero or more moves of the PDA.

Language of a PDA \(M\) (or language recognized by \(M\)) is \(L(M) = \{w | (q_0, w, \varepsilon) \vdash^*(f, \varepsilon, \gamma) \text{ for } f \in F\}\).

How does a PDA check the stack is empty? At the beginning any computation, it always pushes a special symbol $ to the initially empty stack by having transition \(\delta(q_0, \varepsilon, \varepsilon) = \{(q,\$)\}\).
Example (Sipser p. 112): A PDF that recognizes \( \{0^n1^n| n \geq 0\} \).

Example (Sipser p. 114): A PDA that recognized for \( \{a^ib^j c^k | i, j, k \geq 0, i = j \text{ or } i = k\} \).

Example: How the PDA in the above example accepts input \( aabbcc \).
6.2 Equivalence of PDAs and CFGs (Sipser 2.2 (pp. 115-112))

- From CFG to PDA: Let $G$ be the CFG for CFL $A$. Construct a PDA $M$ such that $L(M) = L(G) = A$.

Given a CFG $G = (V, \Sigma, R, S)$. Define a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ to simulate leftmost derivations in CFGs, where

- $Q = \{q_0, q_1, q_2, q_3\} \cup Q'$, where $Q'$ is a set of additional states that may be added later on.
- $F = \{q_3\}$
- $\Gamma = V \cup \Sigma \cup \{\$\} \cup \{\varepsilon\}$

The transition function is defined as follows:

- $\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, \$)\}$
- $\delta(q_1, \varepsilon, \varepsilon) = \{(q_2, S)\}$
- For each rule $X \rightarrow X_k \cdots X_1$ in $R$ with $k \geq 1$, $\delta(q_2, \varepsilon, X) = \{(q_2^1, X_1)\}$, $\delta(q_2^1, \varepsilon, \varepsilon) = \{(q_2^2, X_2)\}$, ..., $\delta(q_2^{k-1}, \varepsilon, \varepsilon) = \{(q_2^k, X_k)\}$
- For each rule $X \rightarrow \varepsilon$ in $R$, $\delta(q_2, \varepsilon, X) = \{(q_2, \varepsilon)\}$
- For each $a \in \Sigma$, $\delta(q_2, a, a) = \{(q_2, \varepsilon)\}$.
- $\delta(q_2, \varepsilon, \$) = \{(q_3, \varepsilon)\}$

**Example** (Sipser p. 118): $S \rightarrow aTb|b$, $T \rightarrow Ta|\varepsilon$. 
From PDA to CFG: Let $M$ be a PDA, where there is only one final state, the stack is emptied upon acceptance, and any transition either pushes or pops (but not both or neither). Construct a CFG $G$ such that $L(M) = L(G)$.

Given a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, \{q_f\})$. Define CFG $G = (V, \Sigma, R, S)$, where $V = \{A_{pq} \mid p, q \in Q\}$, $S = A_{q_0q_f}$, and $R$ has the following rules:

- For each $p \in Q$, $A_{pp} \rightarrow \epsilon$. (other rules for $A_{pp}$ later)
- For each $p, q, r \in Q$, $A_{pq} \rightarrow A_{pr}A_{rq}$. (Stack becomes empty at least once between states $p$ and $q$.)
- For each $p, q, r, s \in Q$, $t \in \Gamma$, and $a, b \in \Sigma \cup \{\epsilon\}$, if $(r, t) \in \delta(p, a, \epsilon)$ and $(q, \epsilon) \in \delta(s, b, t)$, then $A_{pq} \rightarrow aA_{rs}b$. (Stack never becomes empty between states $p$ and $q$.)

Note: Variable $A_{pq}$ is intended to generate all strings that can take $M$ from state $p$ with an empty stack to state $q$ with an empty stack. This is why we set $S = A_{q_0q_f}$.

**Theorem**: The equivalence of PDAs, CFGs, and CFLs.
7.1 Proving non-CFLs by pumping lemma (Sipser 2.3 (pp. 123-127))

**Lemma**: Consider a parse tree according to a CNF grammar with a yield of $w \in \Sigma^*$. If the height of the tree is $h$ then $|w| \leq 2^{h-1}$.

**Theorem** (The pumping lemma for context-free languages): Let $A$ be a CFL. Then there exists a constant $p$ (the pumping length) such that $\forall s \in A$ with $|s| \geq p$, we can write $s = uvxyz$ such that

1. $|vy| > 0$;
2. $|vxy| \leq p$; and
3. $\forall i \geq 0$, string $uv^i xy^i z \in A$. 
**Proof:** Given a CFL $A$, there is a CNF $G = (V, \Sigma, R, S)$ such that $L(G) = A - \{\epsilon\}$. Let $m = |V|$ and choose $p = 2^m$. Suppose $s \in L(G)$ with $|s| \geq p$. Any parse tree for $s$ must have a height of at least $m + 1$, otherwise $|s| \leq 2^{m-1} = p/2$ by Lemma. Therefore, there must be a path $A_0(S), A_1, \ldots, A_l, a$ with at least $m + 1$ variables, i.e., $l \geq m$, in the parse tree for $s$. Since there are only $m$ different variables in the grammar, by the pigeonhole principle, there must be at least two identical variables on the path. Choose the identical pair closest to the leaf, i.e., $A_i = A_j$ with $l - m \leq i < j \leq l$. 
Then it is possible to divide the parse tree such that $x$ is the yield of the subtree rooted at $A_j$, $vxy$ is the yield of the subtree rooted at $A_i$, and $s = uvxyz$ is the yield of the entire parse tree. Next we examine whether this partition satisfies all three conditions stated in the pumping lemma. First, since there is no unit production in CNF, $v$ and $y$ can not both be $\varepsilon$. So $|vy| > 0$. Second, since the height of the subtree rooted at $A_i$ is at most $m + 1$, its yield $vxy$ has length at most $2^m = p$. So $|vwx| \leq p$. Third, since $A_i = A_j = T$, we have $T \Rightarrow vTy$ and $T \Rightarrow x$ from the parse tree. So for any $i \geq 0$, $T \Rightarrow^* v^i xy^i$, thus, $S \Rightarrow^* uTz \Rightarrow^* uv^i xy^i z$. So $uv^i xy^i z \in A$ for $i \geq 0$. This completes the proof of the pumping lemma.
How to use the pumping lemma to prove that a language is not context-free:

- Assume that $A$ was context-free by contradiction. Then the pumping lemma applies to $A$. Let $p$ be the constant in the pumping lemma.
- Select $s \in A$ with $|s| = f(p) \geq p$.
- By the pumping lemma, $s = uvxyz$ with $|vy| > 0$, $|vxy| \leq p$, and $uv^i xy^i z \in A \forall i \geq 0$.
- For any $u, v, x, y, z$ such that $s = uvxyz$, $|vy| > 0$, and $|vxy| \leq p$, find $i \geq 0$ such that $uv^i xy^i z \notin A$. A contradiction!
Example (Sipser p. 126): Prove that $B = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free.
Example (Sipser p. 127): Prove that $D = \{ww \mid w \in \{0, 1\}^*\}$ is not context-free.
Example: Prove that $\{0^i 1^j \mid j = i^2\}$ is not context free.
7.2 Proving non-CFLs by closure properties

- Closed under union: If $A$ and $B$ are context-free, so is $A\cup B$.
- Closed under concatenation: If $A$ and $B$ are context-free, so is $AB$.
- Closed under star: If $A$ is context-free, so is $A^*$.
- Closed under reverse: If $A$ is context-free, so is $A^R$.
- Not closed under intersection: Consider $A = \{a^n b^n c^m\}$ and $B = \{a^m b^n c^n\}$.
- Not closed under complementation: Note that $A \cap B = \overline{A} \cup \overline{B}$.
- Not closed under difference: Note that $\overline{A} = \Sigma^* - A$.
- Intersect with a regular language: If $A$ is context-free and $B$ is regular, then $A \cap B$ is context-free.
- Difference from a regular language: If $A$ is context-free and $B$ is regular, then $A - B$ is context-free. Note that $A - B = A \cap \overline{B}$.
The closure properties of CFLs may be used to prove or disprove that a given language is context free.

**Example:**

\[ A = \{ w \in \{a, b, c\}^* | w \text{ has equal numbers of } as, bs \text{ and } cs \} \] is not a CFL.
Computability Theory:
The question of what computers can do/solve, or equivalently, what languages can be defined/recognized by any computational device whatsoever.
8.1 Unsolvable problems

- Two types of problems: “Solve this” and “decide this”.
- Decision problems (the “decide this” type) have a yes/no solution. They are just as hard as their “solve this” version in the sense of dealing with important questions in complexity theory.
- A problem is said to be unsolvable/undecidable if it cannot be solved/decided using a computer.
- Recall that a decision problem is really membership of a string in a language. For example, the problem of primality testing is actually the language of all prime numbers in binary representation.
The number of problems/languages over an alphabet with more than one symbol is uncountably infinite. However, the number of programs that a computer may use to solve problems is countably infinite. Therefore, there are more problems than there are programs. Thus, there must be some undecidable problems.

An undecidable problem (the halting problem):
- Input: Any program $P$ and any input $I$ to the program;
- Output: “Yes” if $P$ terminates on $I$ and “No” otherwise.
8.2 Turing machine *(Sipser 3.1 (pp. 137-147))*

- A Turing machine includes a control unit, a read-write head, and a one-way infinite tape.
- TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where
  - $Q$: The finite set of states for the control unit.
  - $\Sigma$: An alphabet of input symbols, not containing the “blank symbol” $\sqcup$ (or $B$).
  - $\Gamma$: The complete set of tape symbols. $\Sigma \cup \{\sqcup\} \subset \Gamma$.
  - $\delta$: The transition function from $Q \times \Gamma$ to $Q \times \Gamma \times D$, where $D = \{L, R\}$.
  - $q_0$: The start state.
  - $q_{accept}$: The accept state.
  - $q_{reject}$: The reject state.
Configuration: $X_1 \cdots X_{i-1}qX_i \cdots X_n$ is a configuration (snapshot) of the TM in which $q$ is the current state, the tape content is $X_1 \cdots X_n$, and the head is scanning $X_i$.

- If $\delta(q, X_i) = (p, Y, L)$, then
  $X_1 \cdots X_{i-1}qX_i \cdots X_n \vdash X_1 \cdots X_{i-2}pX_{i-1}YYX_{i+1} \cdots X_n$.

- If $\delta(q, X_i) = (p, Y, R)$, then
  $X_1 \cdots X_{i-1}qX_i \cdots X_n \vdash X_1 \cdots X_{i-1}YpX_{i+1} \cdots X_n$.

Starting configuration $q_0w$, accepting configuration $uq_{accept}v$, and rejecting configuration $uq_{reject}v$, where the latter two are called the halting configurations.

Language of a Turing machine $M$ (or language recognized by $M$) is $L(M) = \{ w \in \Sigma^* | q_0w \vdash \alpha q_{accept} \beta \text{ for } \alpha, \beta \in \Gamma^* \}$. 
For any given input, a TM has three possible outcomes: accept, reject, and loop. Accept and reject mean that the TM halts on the given input, but loop means that the TM does not halt on the input.
A language $A$ is Turing-recognizable (or recursively enumerable) if there is a TM $M$ such that $A = L(M)$. In other words, $\forall w \in A$, $M$ accepts $w$ by entering $q_{\text{accept}}$. However, $\forall w \notin A$, $M$ may reject or loop.

A language $A$ is Turing-decidable (or decidable, or recursive) if there is a TM $M$ such $A = L(M)$ and $M$ halts on all inputs. In other words, $\forall w \in A$, $M$ accepts $w$ and $\forall w \notin A$, $M$ rejects $w$. Such TM’s are a good model for algorithms.
Example: Give a TM $M$ with $L(M) = \{0^n1^n | n \geq 0\}$.

Example (Sipser p. 143): Give a TM $M$ that decides $A = \{0^{2^n} | n \geq 0\}$. 
8.3 Properties of TDLs and TRLs

**Theorem:** A Turing-decidable language is also Turing-recognizable, but not vice versa.

**Theorem:** $A$ and $\overline{A}$:

- If $A$ is Turing-decidable, so is $\overline{A}$.
- If $A$ and $\overline{A}$ are both Turing-recognizable, then $A$ is Turing-decidable.
- For any $A$ and $\overline{A}$, we have one of the following possibilities: (1) Both are Turing-decidable; (2) Neither is Turing-recognizable; (3) One is Turing-recognizable but not decidable, the other is not Turing-recognizable.
Some additional closure properties: TRLs and TDLs both are closed under

- Union
- Intersection
- Concatenation
- Star
8.4 Variations of TMs \((\text{Sipser 3.2 (pp. 148-159)})\)

- TM with multi-tapes (and multi-cursors) \((\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k)\).
- TM with multi-strings (and multi-cursors).
- TM with multi-cursors.
- TM with multi-tracks.
- TM with two-way infinite tape.
- TM with multi-dimensional tape.
- Nondeterministic TM’s \((\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times D})\).
**Theorem**: The equivalent computing power of the above TM’s: For any language $L$, if $L = L(M_1)$ for some TM $M_1$ with multi-tapes, multi-strings, multi-cursors, multi-tracks, two-way infinite tape, multi-dimensional tape, or nondeterminism, then $L = L(M_2)$ for some basic TM $M_2$. 
**Theorem:** The equivalent computing speed of the above TM’s except for nondeterministic TM’s: For any language $L$, if $L = L(M_1)$ for some TM $M_1$ with multi-tapes, multi-strings, multi-cursors, multi-tracks, two-way infinite tape, or multi-dimensional tape in a polynomial number of steps, then $L = L(M_2)$ for some basic TM $M_2$ in a polynomial number of steps (with a higher degree). Or in other words, all reasonable models of computation can simulate each other with only a polynomial loss of efficiency.

Note: The speed-up of a nondeterministic TM vs. a basic TM is exponential.
The Church-Turing Thesis: Any reasonable attempt to model mathematically algorithms and their time performance is bound to end up with a model of computation and associated time cost that is equivalent to Turing machines within a polynomial. (The power of TM.)
9.1 Hilbert’s tenth problem

The Hilbert’s tenth problem (proposed in 1900 among a list of 23 open problems for the new century): Devise a process with a finite number of operations that tests whether a polynomial has an integral root. What Hilbert meant by “a process with a finite number of operations” is an algorithm.

Formulating Hilbert’s problem with today’s terminology: Is there an algorithm to test whether a polynomial has an integral root? (If yes, give the algorithm.) Or, define a language \( D = \{ p \mid p \text{ is a polynomial with integral root} \} \). Is there a Turing machine to decide \( D \)? Here \( p \), although a polynomial, is treated as a string.

Note: In 1970, it was proved that \( D \) is not Turing-decidable (or undecidable).
9.2 A binary encoding scheme for TMs

- TM $\leftrightarrow$ binary number.
  
  $Q = \{ q_1, q_2, \ldots, q_{|Q|} \}$ with $q_1$ to be the start state, $q_2$ to be
  
  the accept state, and $q_3$ to be the reject state.
  
  $\Gamma = \{ X_1, X_2, \ldots, X_{|\Gamma|} \}$.
  
  $D = \{ D_1, D_2 \}$ with $D_1$ to be $L$ and $D_2$ to be $R$.
  
  A transition $\delta(q_i, X_j) = (q_k, X_l, D_m)$ is coded as
  
  $0^i10^j10^k10^l10^m$. A TM is coded as $C_111C_211\cdots11C_n$, where each $C$ is the code for a transition.

- TM $M$ with input $w$ is represented by $< M, w >$ and coded
  
  as $M111w$.

- Using similar schemes, we can encode DFA, NFA, PDA, RE, and CFG.
9.3 Decidable languages (Sipser 4.1 (pp. 166-173))
The following languages are decidable by TMs.

- $A_{DFA} = \{ < B, w > \mid B$ is a DFA that accepts string $w \}.$
- $A_{NFA} = \{ < B, w > \mid B$ is an NFA that accepts string $w \}.$
- $A_{REX} = \{ < R, w > \mid R$ is a regular expression that generates string $w \}.$
- $E_{DFA} = \{ < B > \mid B$ is a DFA and $L(B) = \emptyset \}.$
- $EQ_{DFA} = \{ < B_1, B_2 > \mid B_1$ and $B_2$ are DFAs and $L(B_1) = L(B_2) \}.$
- $A_{CFG} = \{ < G, w > \mid G$ is a CFG that generates string $w \}.$
- $E_{CFG} = \{ < G > \mid G$ is a CFG and $L(G) = \emptyset \}.$

Every CFL is decidable.
9.4 Diagonalization (*Sipser 4.2 (pp. 174-179)*)

- The size of an infinite set: Countably infinite and uncountably infinite.

- Diagonalization to prove a set to be uncountably infinite.
  
  **Example** (*Sipser* p. 175): $Q$, the set of positive rational numbers, is countably infinite.

  **Example** (*Sipser* p. 177): $R$, the set of real numbers, is uncountably infinite.

- Some languages are not Turing-recognizable. (Or equivalently, there are more languages than Turing machines. Since the number of Turing machines is countable, we wish to prove that the number of languages over an alphabet is uncountable.)
9.4 A non-TRL

- Enumerating binary strings: $\varepsilon, 0, 1, 00, 01, 10, 11, \ldots$. The $i$th string, $w_i$, is the $i$th string in the above lexicographic ordering.

- Let the $i$th TM, $M_i$, be the TM whose code is $w_i$, the $i$th binary string. If $w_i$ is not a valid TM code, then let $M_i$ be the TM that immediately rejects any input, i.e., $L(M_i) = \emptyset$.

- Define the diagonalization language $A_D = \{ w_i | w_i \not\in L(M_i) \}$. A boolean table where the $(i, j)$ entry indicates whether TM $M_i$ accepts string $w_j$. Language $A_D$ is made by complementing the diagonal.

- $A_D$ is not Turing-recognizable.
  Proof: Suppose, by contradiction, there is a TM $M$ such that $A_D = L(M)$. Then $M = M_i$ with code $w_i$ for some $i$. $w_i \in A_D$ iff $w_i \not\in L(M_i)$ by definition of $A_D$. $w_i \in A_D$ iff $w_i \in L(M_i)$ by $A_D = L(M_i)$. A contradiction.
9.5 A TRL but non-TDL (*Sipser 4.2 (pp. 173-174 and 179-182))

▶ A universal TM:
- Each TM (among those discussed) can only solve a single problem, however, a computer can run arbitrary algorithms. Can we design a general-purposed TM that can solve a wide variety of problems?
- Theorem: There is a universal TM $U$ which simulates an arbitrary TM $M$ with input $w$ and produces the same output.
- TM $U$ is an abstract model for computers just as TM $M$ is a formal notion for algorithms.
Let $A_{TM} = \{ <M, w> | M \text{ is a TM and } M \text{ accepts string } w \}$

$A_{TM}$ is TR since it can be recognized by TM $U$. $A_{TM}$ is called the universal language.

$A_{TM}$ is non-TD.

Proof: Assume that $A_{TM}$ is decided by TM $T$, Then on input $<M, w>$, $T$ accepts iff $M$ accepts $w$.

Define TM $D$, which on input $<M>$, runs $T$ on input $<M, <M>>$ and accepts iff $T$ rejects $<M, <M>>$.

Feed $<D>$ to $D$. We see that $D$ accepts $<D>$ iff $T$ rejects $<D, <D>>$ iff $D$ does not accept $<D>$. A contradiction.

Diagonalization is used in this proof. Why?
10.1 Reducibility (Sipser 5 (pp. 187-188))
We say that problem $A$ reduces (or is reducible) to problem $B$, if we can use a solution to $B$ to solve $A$ (i.e., if $B$ is decidable/solvable, so is $A$).

We may use reducibility to prove undecidability as follows: Assume we wish to prove problem $B$ to be undecidable and we know a problem $A$ that has already been proved undecidable. We use contradiction. Assume $B$ is decidable. Then there exists a TM $M_B$ that decides $B$. If we can use $M_B$ as a sub-routine to construct a TM $M_A$ that decides $A$, we have a contradiction. The construction of TM $M_A$ using TM $M_B$ establishes that $A$ is reducible to $B$. 
10.2 Another proof of the non-TD $A_{TM}$
Recall $A_{TM} = \{ < M, w > | w \in L(M) \}$. Thus
$A_{\overline{TM}} = \{ < M, w > | w \notin L(M) \}$. Recall $A_D = \{ w_i | w_i \notin L(M_i) \}$.

Proof: Assume that $A_{TM}$ is decidable. Then $\overline{A_{TM}}$ must be
decidable. Let $\overline{M}$ be the TM that decides $\overline{A_{TM}}$. We will construct
a TM $M_D$ that would decide $A_D$, an undecidable language. $M_D$
works as follows: For input $w_i$ (the $i$th binary string in the
lexicographic sequence of all binary strings), it first makes a
string $w_i111w_i$ and then feed it to $\overline{M}$. We notice that
$w_i111w_i \in L(\overline{M})$ iff $w_i111w_i \in \overline{A_{TM}}$ iff $w_i111w_i \notin A_{TM}$ iff
$w_i \notin L(M_i)$ (recall that $M_i$ is the TM with code $w_i$) iff $w_i \in L(M_D)$.
So $M_D$ accepts $w_i$ iff $\overline{M}$ accepts $w_i111w_i$.
We just proved that $A_D$ is reducible to $\overline{A_{TM}}$. 
10.3 The halting problem (*Sipser 5.1 (pp. 188-189)*)

Let $HALT_{TM} = \{< M, w > | M \text{ is a TM and } M \text{ halts on string } w \}$. $HALT_{TM}$ is TR since it can be recognized by TM $U$. $HALT_{TM}$ is non-TD.

Proof: We will reduce $A_{TM}$ to $HALT_{TM}$. Assume TM $R$ decides $HALT_{TM}$. We construct TM $S$ that decides $A_{TM}$ as follows: On input $< M, w >$ where $M$ is a TM and $w$ is a string, $S$ first run TM $R$ on $< M, w >$, if $R$ rejects, rejects. If $R$ accepts, simulate $M$ on $w$ until it halts. If $M$ accepts, accept; if $M$ rejects, reject.
10.4 Other non-TD problems (Sipser 5.1 (pp. 189-192))
The following problems about Turing machines are non-TD:

- Whether \( L(M) = \emptyset \) for any TM \( M \). (See proofs below.)
- Whether \( L(M_1) = L(M_2) \) for any two TMs \( M_1 \) and \( M_2 \).
- Whether \( L(M) \) is finite for any TM \( M \)
- Whether \( \varepsilon \in L(M) \) for any TM \( M \).
- Whether \( L(M) = \Sigma^* \) for any TM \( M \).
$E_{TM} = \{ <M> | M \text{ is a TM and } L(M) = \emptyset \}$ is non-TD.

Proof: Reduce $A_{TM}$ to $E_{TM}$. Assume that $E_{TM}$ is decidable. Let $R$ be the TM that decides $E_{TM}$. We use $R$ to construct TM $S$ that decides $A_{TM}$ as follows: On input $<M, w>$,

- Construct TM $M_1$ which on input $x$, rejects if $x \neq w$ and simulates $M$ on $w$ if $x = w$.
- Run $R$ on $<M_1>$.
- If $R$ accepts, reject and if $R$ rejects, accept.
$NE_{TM} = \{ <M> | M \text{ is a TM and } L(M) \neq \emptyset \}$ is TR but non-TD.

Proof: To prove that $NE_{TM}$ is TR, we design a TM $M_{NE}$ to recognize $NE_{TM}$. On input $<M>$,

- $M_{NE}$ systematically generates strings $w$: $\varepsilon, 0, 1, 00, 01, \ldots$ and use the universal TM $U$ to test whether $M$ accepts $w$. (What if $M$ never halts on $w$? Run $M$ on $w_1, \ldots, w_i$ for $i$ steps for $i = 1, \ldots$)

- If $M$ accepts some $w$, then $M_{NE}$ accepts its own input $M$. 
We next prove that $NE_{TM}$ is non-TD. Assume that there is a TM $M_{NE}$ that decides $NE_{TM}$, i.e., TM $M_{NE}$ determines whether $L(M) \neq \emptyset$ for any TM $M$. We will use $M_{NE}$ to construct a TM $M_u$ that would decides the undecidable $A_{TM}$. On input $<M, w>$,

- $M_u$ constructs a new TM $M'$, which rejects if its input is not $w$ and mimics $M$ if its input is $w$.
- $M'$ is then fed to $M_{NE}$.
- $M_{NE}$ accepts its input $M'$ iff $L(M') \neq \emptyset$ iff $M$ accepts $w$.

Note: $E_{TM}$ is non-TR.

Rice’s Theorem: Every nontrivial property of the TRLs is undecidable.
10.5 More non-TD problems (Sipser 5.2 (pp. 199-205))

Post’s correspondence problem (PCP) is non-TD.

INSTANCE: $P = \{ \frac{t_1}{b_1}, \frac{t_2}{b_2}, \ldots, \frac{t_k}{b_k} \}$, where $t_1, t_2, \ldots, t_k$ and $b_1, b_2, \ldots, b_k$ are strings over alphabet $\Sigma$. ($P$ can be regarded as a collection of dominos, each containing two strings, with one stacked on top of the other.)

QUESTION: Does $P$ contain a match, i.e., $i_1, i_2, \ldots, i_l \in \{1, 2, \ldots, k\}$ with $l \geq 1$ such that $t_{i_1} t_{i_2} \cdots t_{i_l} = b_{i_1} b_{i_2} \cdots b_{i_l}$?

Equivalently, defined as a language, we have $L_{PCP} = \{ <P> \mid P \text{ is an instance of PCP with a match} \}$.

For example, for $P_1 = \{ \frac{b}{ca}, \frac{a}{ab}, \frac{ca}{a}, \frac{abc}{c} \}$, sequence $2, 1, 3, 2, 4$ indicates a match. For $P_2 = \{ \frac{abc}{ab}, \frac{ca}{a}, \frac{acc}{ba} \}$, there is no match.
Any nontrivial property that involves what a program does is undecidable. For example, whether a program prints a certain message, whether it terminates, or whether it calls a certain function.

It is undecidable whether a CFG is ambiguous. Let $G_1$ and $G_2$ be CFG’s and let $R$ be a regular expression. It is undecidable whether

- $L(G_1) \cap L(G_2) = \emptyset$.
- $L(G_1) = L(G_2)$.
- $L(G_1) = L(R)$.
- $L(G_1) = \Sigma^*$.
- $L(G_1) \subseteq L(G_2)$.
- $L(R) \subseteq L(G_1)$. 
Complexity Theory:

- Theory of computability is the study of what can or cannot be computed by a TM/computer, among all problems.
- Theory of complexity is the study of what can or cannot be computed **efficiently** by a TM/computer, among all decidable problems.
11.1 The class of $P$ (Sipser 7.2)

- Definition: $P$ is the class of problems solvable in polynomial time (number of steps) by deterministic TMs. $O(n^c)$, where $c$ is a constant. Tractable (not so hard).

- Why use polynomial as the criterion?
  - If a problem is not in $P$, it will be extremely expensive and probably impossible to solve in practice for large sizes.
  - Polynomials have nice closure properties: $+, -, *,$ and composition.
  - $P$ is independent of models of computation: Basic TM, TMs with multi-tracks, multi-tapes, multi-cursors, multi-strings, multi-dimensional tape. (All variations of TM except nondeterministic TM.)

- Examples of problems in $P$: Sorting, Searching, Selecting, Minimum Sapnning Tree, Shortest Path, etc..
11.2 The class of NP *(Sipser 7.3)*

- An NTM is an unrealistic (unreasonable) model of computation which can be simulated by other models with an exponential loss of efficiency. It is a useful concept that has had great impact on the theory of computation.

- **NTM** $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where $\delta : Q \times \Gamma \rightarrow 2^P$ for $P = Q \times \Gamma \times \{L, R\}$.

- $\delta(q, X)$ is a set a moves. Which one to choose? This is nondeterminism. The computation can be illustrated by a tree, with each node representing a configuration.
Nondeterminism can be viewed as a kind of parallel computation wherein multiple independent processes or threads can be running concurrently. When a nondeterministic machine splits into several choices, that corresponds to a process forking into several children, each then proceeding separately. If at least one process accepts, then the entire computation accepts.

Time complexity of nondeterministic TMs (NTMs): Let $N$ be an NTM that is a decider (where all computation paths halt in the tree for any input). The time complexity of $N$, $f(n)$, is the maximum number of steps that $N$ uses on any computation path for any input of length $n$. In other words, $f(n)$ is the maximum height of all computation trees for all input of length $n$. 
An unreasonable model of computation:

**Theorem:** Every $T(n)$-time multi-tape TM has an equivalent $O(T^2(n))$-time single-tape TM.

**Theorem:** Every $T(n)$-time single-tape NTM has an equivalent $O(2^{O(T(n))})$-time single-tape DTM.

**Definition:** NP is the class of problems solvable in polynomial time by nondeterministic TMs.

**Another definition of nondeterministic TMs (algorithms):**

- **Guessing phase:** Guess a solution (always on target).
- **Verifying phase:** Verify the solution.
Example: TSP (DEC) is in **NP**.

**INSTANCE:** An edge-weighted, undirected, and complete graph $G(V, E, w)$ and a bound $B \geq 0$. (The weight on an edge is allowed to be $+\infty$.)

**QUESTION:** Is there a tour (a cycle that passes through each node exactly once) in $G$ with total weight no more than $B$?

A NTM $N$ that decides TSP in polynomial time can be defined as follows: Assume that a coded instance of TSP is shown on the tape as input. $N$ first generates an arbitrary permutation of the $n$ nodes and then checks whether the permutation represents a tour with total weight no larger than the bound $B$. If so, $N$ accepts the input. If not, $N$ rejects the input, perhaps hoping that another choice of permutations may end up accepting. Note that the permutation generated can be any of the $n!$ possible permutations. The nondeterminism of the machine guarantees that if the instance is a yes-instance, the right permutation will always be selected.
Note: The acceptance of an input by a nondeterministic machine is determined by whether there is an accepting computation among all, possibly exponentially many, computations. In the above proof, if there is a solution, it will always be generated by the Turing machine. This is like that a nondeterministic machine has a guessing power. A tour can only be found by a deterministic machine in exponential time, however, it can be found by a nondeterministic machine in just one step. So any nondeterministic proof should always contain two stages: Guessing and verifying.
Example: Graph Coloring is in \textbf{NP}. Consider the decision problem: For a given graph, is there a coloring scheme of the nodes that uses no more than $B$ colors such that no two nodes connected by an edge are given the same color? A nondeterministic TM first guesses a coloring scheme (in polynomial time) and then verifies if the coloring uses no more than $B$ colors and if any two adjacent nodes have different colors (in polynomial time).
Theorem: $P \subseteq NP$. Any deterministic TM is a special case of a nondeterministic TM.

Theorem: Any $\Pi \in NP$ can be solved by a deterministic TM in time $O(c^{p(n)})$ for some $c > 0$ and polynomial $p(n)$.

Open problem: $NP \subseteq P$? or $P = NP$?
11.3 Polynomial reduction (Sipser 7.4)

- Definition: Let $\Pi_1$ and $\Pi_2$ be two decision problems, and $
\{I_1\}$ and $\{I_2\}$ be sets of instances for $\Pi_1$ and $\Pi_2$, respectively. We say there is a polynomial reduction from $\Pi_1$ to $\Pi_2$, or $\Pi_1 \propto_p \Pi_2$, if there is $f : \{I_1\} \rightarrow \text{subset of } \{I_2\}$ such that (1) $f$ can be computed in polynomial time and (2) $I_1$ has a “yes” solution if and only if $f(I_1)$ has a “yes” solution.

- Theorem: If $\Pi_1 \propto_p \Pi_2$, then $\Pi_2 \in \mathbf{P}$ implies $\Pi_1 \in \mathbf{P}$.

- Theorem: If $\Pi_1 \propto_p \Pi_2$ and $\Pi_2 \propto_p \Pi_3$, then $\Pi_1 \propto_p \Pi_3$.

- Remark: $\propto_p$ means “no harder than”.
11.4 The class of NPC (Sipser 7.4)

- **Definition:** NPC (NP-complete) is the class of the hardest problems in NP, or equivalently, \( \Pi \in \text{NPC} \) if \( \Pi \in \text{NP} \) and \( \forall \Pi' \in \text{NP}, \Pi' \propto_p \Pi \).

- **Theorem:** If \( \Pi_1 \in \text{NPC}, \Pi_2 \in \text{NP} \), and \( \Pi_1 \propto_p \Pi_2 \), then \( \Pi_2 \in \text{NPC} \).

- **Theorem:** If \( \exists \Pi \in \text{NPC} \) such that \( \Pi \in \text{P} \), then \( \text{P} = \text{NP} \).

- **Theorem:** If \( \exists \Pi \in \text{NPC} \) such that \( \Pi \not\in \text{P} \), then \( \text{P} \neq \text{NP} \).
Satisfiability (SAT):
INSTANCE: A boolean formula $\alpha$ in CNF with variables $x_1, \ldots, x_n$ (e.g., $\alpha = (x_1 \lor x_2) \land (\neg x_2 \lor x_3)$).

QUESTION: Is $\alpha$ satisfiable? (Is there a truth assignment to $x_1, \ldots, x_n$ such that $\alpha$ is true?)

Cook’s Theorem: SAT $\in$ NPC. (Need to prove (1) SAT $\in$ NP and (2) $\forall \Pi \in$ NP, $\Pi \propto^p$ SAT.)
11.5 NP-complete problems
How to prove $\Pi$ is $\textbf{NP}$-complete:

- Show that $\Pi \in \textbf{NP}$.
- Choose a known $\textbf{NP}$-complete $\Pi'$.
- Construct a reduction $f$ from $\Pi'$ to $\Pi$.
- Prove that $f$ is a polynomial reduction by showing that (1) $f$ can be computed in polynomial time and (2) $\forall I'$ for $\Pi'$, $I'$ is a yes-instance for $\Pi'$ if and only if $f(I')$ is a yes-instance for $\Pi$. 
Seven basic **NP-complete problems**.

- **3-Satisfiability (3SAT):**
  INSTANCE: A formula $\alpha$ in CNF with each clause having three literals.
  QUESTION: Is $\alpha$ satisfiable?

- **3-Dimensional Matching (3DM):**
  INSTANCE: $M \subseteq X \times Y \times Z$, where $X, Y, Z$ are disjoint and of the same size.
  QUESTION: Does $M$ contain a matching, which is $M' \subseteq M$ with $|M'| = |X|$ such that no two triples in $M'$ agree in any coordinate?

- **PARTITION:**
  INSTANCE: A finite set $A$ of numbers.
  QUESTION: Is there $A' \subseteq A$ such that $\sum_{a \in A'} a = \sum_{a \in A - A'} a$?
Vertex Cover (VC):
INSTANCE: A graph $G = (V, E)$ and $0 \leq k \leq |V|$.
QUESTION: Is there a vertex cover of size $\leq k$, where a vertex cover is $V' \subseteq V$ such that $\forall (u, v) \in E$, either $u \in V'$ or $v \in V'$?

Hamiltonian Circuit (HC):
INSTANCE: A graph $G = (V, E)$.
QUESTION: Does $G$ have a Hamiltonian circuit, i.e., a tour that passes through each vertex exactly once?

CLIQUE:
INSTANCE: A graph $G = (V, E)$ and $0 \leq k \leq |V|$.
QUESTION: Does $G$ contain a clique (complete subgraph) of size $\geq k$?
A proof that 3SAT is NP-complete:
First, 3SAT is obvious in \textbf{NP}.
Next, we show that \text{SAT} \propto_p 3\text{SAT}.
Given any instance of SAT, \( f(x_1, \ldots, x_n) = c_1 \land \cdots \land c_m \), where \( c_i \) is a disjunction of literals. To construct an instance for 3SAT, we need to convert any \( c_i \) to an equivalent \( c'_i \), a conjunction of clauses with exactly 3 literals.

Case 1. If \( c_i = z_1 \) (one literal), define \( y^1_i \) and \( y^2_i \). Let
\[
\begin{align*}
c'_i &= (z_1 \lor y^1_i \lor y^2_i) \land (z_1 \lor y^1_i \lor \neg y^2_i) \land (z_1 \lor \neg y^1_i \lor y^2_i) \land (z_1 \lor \neg y^1_i \lor \neg y^2_i).
\end{align*}
\]

Case 2. If \( c_i = z_1 \lor z_2 \) (two literals), define \( y^1_i \). Let
\[
\begin{align*}
c'_i &= (z_1 \lor z_2 \lor y^1_i) \land (z_1 \lor z_2 \lor \neg y^1_i).
\end{align*}
\]

Case 3. If \( c_i = z_1 \lor z_2 \lor z_3 \) (three literals), let \( c'_i = c_i \).
Case 4. If $c_i = z_1 \lor z_2 \lor \cdots \lor z_k$ ($k > 3$), define $y_i^1, y_i^2, \ldots, y_i^{k-3}$. Let $c'_i = (z_1 \lor z_2 \lor y_i^1) \land (\neg y_i^1 \lor z_3 \lor y_i^2) \land (\neg y_i^2 \lor z_4 \lor y_i^3) \land \cdots \land (\neg y_i^{k-3} \lor z_{k-1} \lor z_k)$.

If $c_i$ is satisfiable, then there is a literal $z_l = T$ in $c_i$. If $l = 1, 2$, let $y_i^1, \ldots, y_i^{k-3} = F$. If $l = k - 1, k$, let $y_i^1, \ldots, y_i^{k-3} = T$. If $3 \leq l \leq k - 2$, let $y_i^1, \ldots, y_i^{l-2} = T$ and $y_i^{l-1}, \ldots, y_i^{k-3} = F$. So $c'_i$ is satisfiable.

If $c'_i$ is satisfiable, assume $z_l = F$ for all $l = 1, \ldots, k$. Then $y_i^1, \ldots, y_i^{k-3} = T$. So the last clause $(\neg y_i^{k-3} \lor z_{k-1} \lor z_k) = F$.

Therefore, $c'_i$ is not satisfiable. Contradiction.

The instance of 3SAT is therefore $f'(x_1, \ldots, x_n, \ldots) = c'_1 \land \cdots \land c'_m$, and $f$ is satisfiable if and only if $f'$ is satisfiable.