CS360 – Theory of Computing

August 19, 2019

All notes are typed by LaTeX

Important Notes:
· Computational problems and devices can be viewed as mathematical object
· Computation is very general

- Computer running program
- Networks of computers interacting
- People performing computation by hand
- Proofs of theorem
- Biological process

Mathematical fundation:
· Set theory

$$\begin{cases} Naive\ theory \\ Axiomatic\ theory \end{cases}$$

Some notation:
1. TFAE = **T**he **F**ollowing **A**re **E**quivalent

Lecture 1:

Defn: A set is countable if
· A is empty
· There exist an onto function f of this form:

$$f: \mathbb{N} \to A$$

TFAE:
1. A is countable.
2. There exist a one-to-one function g of the form:

$$g: A \to \mathbb{N}$$

3. Either A is finite or there exist a one-to-one onto function h of the form:

$$h: \mathbb{N} \to A$$

Terminology:

If A is a set, then the power set $P(A)$ is the set of all subsets of A.

Cantor Theorm:

The power set of natural number is not countable.

$$\textit{i.e. } P(\mathbb{N}) \textit{ is not countable.}$$

Prove by contradiction:
Assume towards contradiction that $P(N)$ is countable.
This implies that there exist an onto function f, such that

$$f: \mathbb{N} \to P(N)$$

Consider we define a set $S = \{n \in \mathbb{N} : n \notin f(n)\}$
Since f is onto function, there must exist some number $n \in N$, such that $S = f(n)$
We fix such a choice of n! (i.e. $S = f(n)$)
We get

$$n \in S \Leftrightarrow n \notin f(n) \Leftrightarrow n \notin S$$

Thus it is a contradiction. Therefore $P(\mathbb{N})$ is not countable.

Lecture 2:

Terminologies:
Alphabet – symbols for using computation

An alphabet is a finite and nonempty set.

Typical names: Σ, Γ
Examples:

$$\Sigma = \{0, 1\} \rightarrow \textit{binary alphabet}$$
$$\Sigma = \{0\} \rightarrow \textit{unary alphabet}$$
$$\Sigma = \{0, 1, 2, \cdots, n-1\} \textit{ for some possible interger } n$$

String – over some alphabet

Let Σ be an alphabet, the string over Σ is a finite sequence of symbols from Σ.

Typical variables names for symbols: a, b, c, d
Typical string names: u, v, w, x, y, z
Examples:

$$\textit{under } \Sigma = \{0, 1\} \textit{ we have } 0101,\ 0,\ 1,\ \epsilon$$
$$\textit{under } \Sigma = \{0, 1, 2\} \textit{ we have } 01,\ 1,\ 0$$

Language

Let Σ be an alphabet. A language over Σ is a set of strings over Σ. Σ^* language consisting of EVERY string over Σ.

Typical names: A, B, C, D
Example(assume $\Sigma = \{0, 1\}$):

$$B = \{w \in \Sigma^* : w \textit{ ends with } 0\}$$
$$C = \{w \in \Sigma^* : |w| \textit{ is a prime number}\}$$

Question:

Consider Σ^*. Is it countable?
Yes, it is!
Consider we can find an onto function f

$$f : \mathbb{N} \to \Sigma^*$$

We arrange the all the strings in Σ^* by its length.
i.e. ϵ, 0, 1, 00, 01, 10, 11, $\cdots$
The sequence above has lexicographic order
Then we set the function to be:

$$f(0) = \epsilon$$
$$f(1) = 0$$
$$f(2) = 1$$
$$f(3) = 00$$
$$\cdots$$

Thus we can form an onto function that is able to count all the strings.
Therefore Σ^* is countable.

Consider $\Sigma = \{0, 1\}$, how many languages over Σ are there?
TFAE:
1. A is language over Σ
2. $A \subseteq \Sigma^*$
3. $A \in P(\Sigma^*)$

The answer is No. We can prove this by using Cantor Theorem.
From the previous questions, we know that there is an one-to-one function f that can map from Σ^* to $\mathbb{N}$.

$$i.e.\ \ f : \Sigma^* \to \mathbb{N}$$

Assume towards contradiction that $P(\Sigma^*)$ is countable.
Therefore we can find an onto function, say g, that

$$g : \mathbb{N} \to P(\Sigma^*)$$

Since we can map $\mathbb{N}$ to Σ^*, thus we can find an onto function, say h, such that

$$h : \mathbb{N} \to P(\mathbb{N})$$

That contradicts Cantor's Theorem.
Therefore we get the languages over Σ is uncountable.

Deterministic Finite Automata (DFA)

Definition:
A DFA is a 5 tuple $m = (Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite nonempty set of states.
- Σ is an alphabet.
- δ is the transition function of the form

$$\delta : \ Q \times \Sigma \to Q$$

- q_0 is the starting state.
- F is the set of accepting states.

Important notes:
A DFA needs to accept all the strings in one language AND rejects all the strings that are not in that language.

Formal Definition of DFA
Let M be a DFA and let $w \in \Sigma^*$. The DFA m accepts w if one of these following holds:
1. $\omega = \epsilon$ and $q_0 \in F$
2. $\omega = a_1 a_2 \cdots a_n$ for $n \leq 1$ and $a_1, a_2, \cdots, a_n \in \Sigma$ and there exist a sequence of states $r_0, r_1, \cdots, r_n \in Q$ such that $r_0 = q_0$, $r_n \in F$ and $r_{k+1} = \delta(r_k, a_{k+1})$ for every $k \in \{0, 1, 2, \cdots, n-1\}$. If m does not accept, then it rejects w.

Convention Function:
Suppose $m = (Q, \Sigma, \delta, q_0, F)$ is a DFA.
Define $\delta^* : Q \times \Sigma^* \to Q$ as follows:

- $\delta^*(q, \epsilon) = q$
- $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$

Notation:
$L(M) = \{w \in \Sigma^* : M \text{ accepts } w\} \rightarrow$ "language recognized by M"

Regular Language Definition:
Let Σ be an alphabet and let A be a language over Σ. A is regular if there exist a DFA such that $L(M) = A$

Consider fixed $\Sigma = \{0, 1\}$, how many regular language over Σ?
The answer is countable many.
Consider the number of states.
Similar to the approach above.
We can list all the DFA by counting their states.
Then we can claim that there exist an onto function f that maps from $\mathbb{N}$ to all the DFA over Σ.
Thus we know that DFA is countable.
Therefore by the definition above, we know regular language is countable.

Lecture 3:

Non-Deterministic finite automatas(NFAs)

— verification whether computer can get the result by trying multiple ways.

Defn:
NFA is a 5-tuple:

$$N = (Q, \Sigma, \delta, q, F)$$

where

- Q is a finite and non-empty set of states
- Σ is a transition function of the form

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow P(Q)$$

- $q_0 \in Q$ is a start state
- $F \subseteq Q$ is a set of accept states

Formal Definition
Let $N = \{Q, \Sigma, \delta, q_0, F\}$ be an NFA and let $w \in \Sigma^*$. N accepts w if there exist $m \in \mathbb{N}$, a sequence of states, $r_0 \cdots r_m \in Q$ and a sequence of alphabet (move), $a_1, \cdots, a_m \in \Sigma \cup \{\epsilon\}$

- $r_0 = q_0$, $r_m \in F$
- $r_{k+1} \in \delta(r_k, a_{k+1}),\ \forall\ k \in \{0, 1, \cdots, m-1\}$
- $w = a_1 a_2 \cdots a_m$ If N does not accept w, then rejects w

Now we need to check the base case when $m = 0$.
Consider $m = 0$, then we have

$$\begin{cases} r_0 \cdots r_m & \to r_0 \in F \\ a_1 \cdots a_0 & \to \text{empty sequence} \end{cases}$$

From above, we get the starting state is in the set of accepting states.
Therefore, we know the condition satisfy all situations.

ϵ - closure
Let $R \subseteq Q$ and let $N = \{Q, \Sigma, \delta, q_0, F\}$ be an NFA. The ϵ – closure of R denoted $\epsilon(R)$ is the set of all states that can be reached from any states in R by following 0 or more ϵ transition.

$$\epsilon : P(Q) \to P(Q)$$

Define δ^* for an NFA

$$\delta^*(p, \epsilon) = \epsilon(\{p\})$$
$$\delta^*(p, wa) = \epsilon(\cup_{r \in \epsilon^*(p,w)} \epsilon(r, a))$$

Theorem:
Let Σ be an alphabet, A $\subseteq \Sigma^*$ be a language.
The language A is regular iff there exist an NFA, N $= \{Q, \Sigma, \delta, q_0, F\}$ such that $A = L(N)$
proof:
$\Rightarrow$ given that A is regular, then there exist a DFA $M = (Q, \Sigma, \delta, q_0, F)$ with A = L(N)
Define $N = (Q, \Sigma, \eta, q_0, F)$ where

$$\eta(q, a) = \{\delta(q, a)\}$$
$$\eta(q, \Sigma) = \emptyset \ \forall \ q \in Q, \ a \in \Sigma$$

In this case, L(N)=L(M)=A.

$\Leftarrow$ given that NFA, $N = (Q, \Sigma, \delta, q_0, F)$
idea: we will construct a DFA m, such that L(m) = L(N). We will take the state set of m to be P(Q).
define m as follows:

$$m = (P(Q), \Sigma, \gamma, \epsilon(\{q\}), G)$$

where γ and G are as follows.
$\gamma : P(Q) \times \Sigma \to P(Q)$ where $\gamma(R, a) = \epsilon(\cup_{r \in R} \delta(r, a))$ for $R \in Q, a \in \Sigma$
$G = \{R \in P(Q) : R \cap F \neq \emptyset\}$

Lecture 4:

Regular Operation:
Let $A, B \subseteq \Sigma^*$ be language over a alphabet Σ
The regular operation is:
1. Union: $A \cup B = \{w : w \in A \; or w \in B\}$
2. Concatenation: $AB = \{wx : w \in A \; and \; x \in B\}$
3. Kleene star: $A^* = \{\epsilon\} \cup A \cup AA \cup AAA \cdots$

Theorem:
Let Σ be an alphabet and let $A, B \subseteq \Sigma^*$ be regular language. The language $A \cup B, AB, A^*$ are regular.
Proof:
Consider let $M_a = (P, \Sigma, \delta, p_0, F)$ and let $M_b = (Q, \Sigma, \gamma, q_0, G)$ be DFAs with $A = L(M_a), B = L(M_b)$. WLOG, let $P \cap Q = \emptyset$
Consider $A \cup B$, we define a NFA as follows:

$$N = (P \cup Q \cup \{r_0\}, \Sigma, \mu, r_0, F \cup G)$$

where,

$$\begin{aligned}
\mu(p, a) &= \{\delta(p, a)\} && \text{for } p \in P, \; a \in \Sigma \\
\mu(p, \epsilon) &= \emptyset && \text{for } p \in P, \; a \in \Sigma \\
\mu(q, a) &= \{\gamma(q, a)\} && \text{for } q \in Q, \; a \in \Sigma \\
\mu(q, \epsilon) &= \emptyset && \text{for } q \in Q, \; a \in \Sigma \\
\mu(r_0, \epsilon) &= \{q_0, p_0\} && \text{for } a \in \Sigma \\
\mu(r_0, a) &= \emptyset && \text{for } a \in \Sigma
\end{aligned}$$

The proof for AB, A^* is similar.

Questions:
$A \subseteq \Sigma^*$, define $\bar{A} = \Sigma^* \setminus A = \{x \in \Sigma^* : x \notin A\}$
If A is regular is $\bar{A}$ regular?
Yes, by swaping all accepting state with non-accepting states.
Then we can form a new DFA/NFA, so we know $\bar{A}$ is regular.

If $A, B \subseteq \Sigma^*$ are regular is $A \cap B$ regular?
$m = (P \times Q, \Sigma, \mu, (p_0, q_0), F \times G)$
$\mu((p, q), a) = (\delta(p, a), \gamma(q, a))$

Another way is using DeMorgan's Laws.
$A \cap B = \overline{\bar{A} \cup \bar{B}}$

Regular Expression:

Defn: Let Σ be an alphabet. R is a regular expression over Σ if one of these following is true:
1. $R = \emptyset$
2. $R = \epsilon$
3. $R = a \ for \ a \ \in \Sigma$
4. $R = (R_1, R_2)$ for $a \in \Sigma$
5. $R = (R_1 R_2)$ for R_1, R_2 regular expressions
6. $R = (R_1^*)$ for R_1 regular expressions

Let R be regular expression over Σ. The language recognized or method by R is defined as follows:
1. If $R = \emptyset$, then $L(R) = \emptyset$
2. If $R = \epsilon$, then $L(R) = \{\epsilon\}$
3. If $R = a$ for $a \in \Sigma$, then $L(R) = \{a\}$
4. If $R = (R_1 \cup R_2)$, then $L(R) = L(R_1) \cup L(R_2)$
5. If $R = (R_1 R_2)$, then $L(R) = L(R_1)L(R_2)$
6. If $R = (R_1^*)$, then $L(R) = L(R_1)^*$

Order of precedure:
1. Kleene star
2. Concatenation
3. Union

Theorem: Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a language over Σ. A is regular if and only if there exist a regular expression R with $L(R) = A$.

Leture 5

Lemma: Pumping Lemma

Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a regular language. There exist a positive integer, n (called a pumping length of A) that possesses the folloing property. For every string $W \in A$ with $|w| > n$. It is possible to write $w = xyz$ for some choice of strings $x, y, z \in \Sigma^*$ such that

1. $y \neq \epsilon$
2. $|xy| \leq n$
3. $xy^iz \in A$, for every $i \in \mathbb{N}$

Proof:

Let $m = (Q, \Sigma, \delta, q_0, F)$ be a DFA with $A = L(m)$, and let $n = |Q|$

If there is no strings $w \in A$, then the statement we need to prove is vacuously true.

Suppose $w \in A$ and $|w| \geq n$

Then we have $w = a_1a_2 \cdots a_m$ where $m \geq n$. Therefore they must exist sates $r_0, r_1, \cdots, r_m \in Q$ with $r_0 = q_0, r_m \in F$,and $r_{k+1} = \delta(r_k + a_{k+1})$

Consider $r_0, r_1, \cdots, r_n$

By the pigeon hole principle, there must exist $s, t \in \{0, 1, \cdots, n\}$, with $s < t$ and $r_s = r_t$

Let $x = a_1a_2 \cdots a_s$, $y = a_{s+1} \cdots a_t$ and $z = a_{t+1} \cdots a_m$

We have $y \neq \epsilon$, since $s < t$, then item one is proved.

We have $|xy| = t \leq n \Rightarrow$, since the definition is claimed above.

We have xy^iz for any $i \in \mathbb{N}$ must be accepted.

Consider the sequence, $r_0, r_1, \cdots, r_s, r_{s+1}, \cdots, r_t, r_{t+1} \cdots, r_m$, we can repeat i times,since $r_s = r_t$.

Proposition: Let $\Sigma = \{0, 1\}$ and let $A = \{0^m1^m : m \in \mathbb{N}\}$, we want to show that A is non-regular.

Assume towards contradiction A is regular. By the pumping lemma, there exist a pumping length $n \geq 1$ for A, such that the statements in that lemma are all true.

Fix such a choice of n.

Define $w = 0^n1^n$

It is the case that $w \in A$ and $|w| = 2n \geq n$. Therefore, by pumpling lemma, there must exist x, y, z such that it satisfies 3 items.

we know that $w = xyz$ and $w = 0^n1^n$, where $|xy| \leq n$

This implies that $y = 0^k$ for some $k \in \mathbb{N}$

Since $y \neq \epsilon$, then $k > 0$

Consider $i = 2$,in third item, then we have $xy^2z = xyyz = 0^{n+k}1^n \in A$

However, $0^{k+n}1^n \notin A$, contradiction! Therefore, we have A is non-regular.

Normally, we can use Pumping lemma to prove a language is non-regular. However, there is also another way to prove as well.
Firstly, we need to introduce some concepts.

Definition:
w^R is the reverse of w, $(aw)^R = w^R a$ for $a \in \Sigma$ and $w \in \Sigma^*$

We can simply prove the reverse of w is non-regular by using $0^n 1 0^n$
But when comes to $E = \{w \in \Sigma^* : w \neq w^R\}$, we can easy to prove by using contradiction.
Then we assume towards contradiction E is regular, then we know that $\bar{E}$ is regular (proved before), however we know that $\bar{E} \equiv w = w^R$, which is non-regular contradiction!

On the other hand, we can prove the above proposition in a easy way.
Consider $F = \{w \in \Sigma^* : |w|_0 = |w|_1\}$
Assume F is regular, $F \cap L(0^*1^*) = A$
We know that $L(0^*1^*)$ is regular and A is non-regular. If F is regular, then we know that $F \cap L(0^*1^*)$ is regular, which is not.
Therefore, we get F is non-regular.

Lecture 6:

Proposition: Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a regular language.
The language $A^R = \{w^R : w \in A\}$ is regular.
Proof:
Consider there exist a DFA $M = (Q, \Sigma, \delta, q_0, F)$, such that $L(M) = A$
We can construct an NFA N, such that $L(N) = A^R$
Consider $N = (Q \cup \{r_0\}, \Sigma, \eta, r_0, \{q_0\})$, where

$$\begin{cases} \eta(r_0, \epsilon) & = F \\ \eta(r_0, q) & = \emptyset \\ \eta(q, \epsilon) & = \emptyset \\ \eta(q, a) & = \{p \in Q : \delta(p, a) = q\} \end{cases}$$

A is a regular language, there exist a regular expression S, $L(R) = A$
Define the reverse of a regular expression in the natural way S

$$L(S^R) = L(S)^R = A^R$$

The reverse of a regular expression S can be defined like this:

$$\begin{cases} 1.\emptyset^R = \emptyset \\ 2.\epsilon^R = \epsilon \\ 3.a^R = a \\ 4.S = (S_1 \cup S_2) \Rightarrow S^R = (S_1 \cup S_2)^R = (S_1^R \cup S_2^R) \\ 5.S = (S_1 S_2) \Rightarrow S^R = (S_1 S_2)^R = (S_2^R S_1^R) \\ 6.S = S_1^* \Rightarrow S^R = ((S_1^R)^*) \end{cases}$$

Synmetric Difference
Consider $A, B \subseteq \Sigma^*$ be language. Define $A \triangle B = (A \cap \bar{B}) \cup (\bar{A} \cap B)$
1. If $A, B \subseteq \Sigma^*$ are regular, then $A \triangle B$ is regular?
YES
Proof: we can prove this by using closure property and cartesion product.
2. If $A, B \subseteq \Sigma^*$ are non-regular, then $A \triangle B$ is non-regular?
NO
Proof: we can prove this by providing conter-example.
Consider $A = B = \{0^n 1^n : n \in \mathbb{N}\}$
Then we have $A \triangle B = \emptyset$ that is regular.
3. If A is regular, B is non-regular, then $A \triangle B$ is non-regular.
YES
Proof: we can assume $A \triangle B$ is regular
Then $(A \triangle B) \triangle A = B$ is regular that is contradiction.

Prefix, Suffix and Substring
Consider $A \subseteq \Sigma^*$ be any language. We define:
$Prefix(A) = \{x \in \Sigma^*, \exists v \in \Sigma^* : xv \in A\}$
$Suffix(A) = \{x \in \Sigma^*, \exists v \in \Sigma^* : vx \in A\}$
$Substring(A) = \{x \in \Sigma^*, \exists u, v \in \Sigma^* : uxv \in A\}$
Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA with $L(M) = A$
Prefix:
Define $P \subseteq Q$ to be the set of all states from which we reach to accepting states.

$$P = (q \in Q, \exists v \in \Sigma^* \text{ such that } \delta^*(q, v) \in F)$$

We define $K_1 = (Q, \Sigma, \delta, q_0, P)$ then $L(K_1) = prefix(A)$

Suffix:
Define $N = (Q \cup \{r_0, \}, \Sigma, \eta, r_0, F)$, where

$$\begin{cases} \eta(q, a) & = \{\delta(q, a)\} \\ \eta(q, \epsilon) & = \emptyset \\ \eta(r_0, a) & = \emptyset \\ \eta(r_0, \epsilon) & = \{q \in Q : \exists\, a \in \Sigma^*, \delta^*(q_0, a) = q\} \end{cases}$$

Substring:
We can use closure property say that $substring(A) = prefex(suffix(A))$ or $substring(A) = suffix(prefix(A))$

Prove regularity by constructing a regular language:
Define $M_{p,q} = (Q, \Sigma, \delta, q_0, F)$ be any DFA.
$A_{p,q} = L(M_{p,q}) \rightarrow$ any string starts from symbol p and ends with symbol q
Consider $B = \{uv : uv \in \{0,1\}^*, uav \in A \text{ for some } a \in \{0,1\}\}$

$$\bigcup_{r \in Q, a \in \{0,1\}, q \in F} A_{q_0,r} A_{\delta(r,a),q} = B$$

Consider $C = \{uav : uv \in \{0,1\}^*, uav \in A \text{ for some } a \in \{0,1\}\}$

$$\bigcup_{r \in Q, a \in \{0,1\}, q \in F} A_{q_0,r} \{a\} A_{r,q} = C$$

Lecture 7 – Context Free language

Definition:
A context-free gramma has four tuple. $G = (V, \Sigma, R, S)$ where
1. V is a finite and non-empty set of variables
2. Σ is an alphabet
3. R is finite and non-empty set of rules of the form:

$$A \to w \quad \text{where } A \in V \text{ and } w \in (V \cup \Sigma)^*$$

4. $S \in V$ is the starting variable
e.g. $G = (V, \Sigma, R, S)$ where $V = \{S\}, \Sigma = \{0, 1\}$ and $R = \{S \to 0S1, S \to \epsilon\}$
$S \Rightarrow 0S1 \Rightarrow 01$
For this grammar, the set of all possible strings that be generated is $L(G) = \{0^n 1^n : n \in \mathbb{N}\}$

Yields relation
Given CFG, $G = (V, \Sigma, R, S)$ we define the yields relation corrsponding to G as follows:
Given $u, v, w \in (V \cup \Sigma)^*$ and $(A \to w) \in R$ we have $uAv \Rightarrow uwv$
The yields relation " $\Rightarrow$ " consists of all possible pairs obtained in this way. We are also interested in the reflexive, transitive, closure of $\Rightarrow$, $\Rightarrow^*$ means 0 or more times.

Another way of defining $\Rightarrow^*$ is as follows (for $x, y \in (V \cup \Sigma)^*$)
If there exist $m \geq 1$ and $z_1 \cdots z_m \in (V \cup \Sigma)^*$ such that

$$x = z \quad y = Zm \text{ and } Z_k \Rightarrow Z_{k+1} \quad \forall\, k \in \{1, \cdots, m-1\}$$

Definition:
Let $G = (V, \Sigma, R, S)$ be a CFG. The language genereated by G is $L(G) = \{w \in \Sigma^* : S \Rightarrow^* w\}$

Definition:
If A language $A \in \Sigma^*$ is context-free if $A = L(G)$ for some CFG, G.
e.g. $PAL = \{w \in \{0, 1\}^*, w^R = w\}$

$$\begin{cases} S \to \epsilon \\ S \to 0S0 \\ S \to 1S1 \\ S \to 1 \\ S \to 0 \end{cases}$$

We can write this in a simplier way $S \to 0S0|1S1|1|0|\epsilon$

Consider $\Sigma = \{0,1\}, w \in \Sigma^*$, write $|w|_0$ to mean the number of 0's appearing in w likewise for $|w|_1$
e.g. $A = \{w \in \Sigma^*, |w|_0 = |w|_1\}$
$S \to 0S1S|1S0S|\epsilon$
$w \in A \quad w = a_1a_2a_3\cdots a_n$ for $a_1 \cdots a_n \in \Sigma$
$|a_1 \cdots a_n|_0 = |a_1 \cdots a_n|_1$
$N_k = |a_1 \cdots a_k|_0 - |a_1 \cdots a_k|_1$ for each $k \in (0, \cdots, n)$

e.g. $\Sigma = \{(,)\}$
$BAL = \{w \in \Sigma^* :$ parenthese are balanced in w$\}$
$w \in \{(,)\}^*$ has balanced parentheses if we can repeatly remove the substring () from w to get ϵ
$S \to \epsilon|(S)|SS$

e.g. $\Sigma = \{0,1\}$, $\Gamma = \{0,1,\#\}$
$A \subseteq \Gamma^*$, $A = \{u\#v, uv \in \Sigma^*, u \neq v\}$ $u \neq v$ means:
1. exist an index k such that $u_k \neq v_k$
2. $|u| \neq |v|$

$$\underbrace{\square\square\cdots\square}_{\text{k-1}}0\underbrace{\square\square\cdots\square}_{\text{don't care}}\#\underbrace{\square\square\cdots\square}_{\text{k-1}}1\underbrace{\square\square\cdots\square}_{\text{don't care}}$$

Consider we have $X = 0|1$
To represent the don't care part we have $Y = XY|\epsilon$
In order to make sure that both parts with number of $k-1$ elements to be the same, and separated by 0 or 1, don't care part and #.
We can construct $Z_0 = XZ_0X|0Y\#$ and $Z_1 = XZ_1X|1Y\#$ Put into together we have $S \to Z_01Y|Z_10Y$ and we resolve both situations.

Lecture 8

Left-most Derivation and Parse tree
Consider $G : S \to 0S1S|1S0S|\epsilon$, $L(G) = \{w \in \{0,1\}^*, |w|_0 = |w|_1\}$
Let do the left-most Derivation for 0101 by using the grammar shown above:
$S \Rightarrow 0S1S \Rightarrow 01S0S1S \Rightarrow 010S1S \Rightarrow 0101S \Rightarrow 0101$
However, we can use another way to represent string 0101.
Definition:
G is an ambiguous grammar. If CFG G generates at least one string w with 2 different pares tree, then G is ambiguous.

We can modify G to be unambiguous by introducing more variables.
Consider we have

$$S \to 0X1S|1Y0S|\epsilon$$
$$X \to 0X1X|\epsilon$$
$$Y \to 1Y0Y|\epsilon$$

Then we have $H = L(G) = L(H)$ and H is unambiguous.

There are some context-free language do not have an unambiguous grammar. We called this type of language as *inherently ambiguous language*.
Consider we have $A = \{0^n1^m2^k : n = m \text{ or } m = k\}$. The context-free gramma is constructed as followed:

$$S \Rightarrow UX_2|X_0V$$
$$U \Rightarrow 0U1|\epsilon$$
$$V \Rightarrow 1V2|\epsilon$$
$$X_0 \Rightarrow 0X_0|\epsilon$$
$$X_1 \Rightarrow 1X_1|\epsilon$$
$$X_2 \Rightarrow 2X_2|\epsilon$$

Chomsky Normal Form
Definition: A CFG G is in Chomsky normal form if every rule in G has one of these forms:
1. $X \to YZ$ for variables X, Y, Z neither Y nor Z is the start variable
2. $X \to a$ for X is a variable and a is a symbol
3. $S \to \epsilon$ for S is the starting variable.

Theorem: If $w \in L(w)$ for some context-free grammar G in Chomsky normal form and $w \neq \epsilon$ has $2|w| - 1$ variable nodes and $|w|$ leaves.

Theorem: Let A be a context-free language. There exist a CFG G in Chomsky normal form with $L(G) = A$.
The following steps show how to convert to CNF, however it is tedious and extremely easy to make a mistake.
Nowadays, this conversion can be done easily by computer.
1. Introduce a new start variable S_0 along with rule.
2. Introduce a new variable X_a for every $a \in \Sigma$, along with rule.
3. Add auxiliary variables to split up rules like $X_1 \to Y_1 \cdots Y_m$ to becomes

$$X \to Y_1Z_2$$
$$Z_2 \to Y_2Z_3$$
$$\cdots$$
$$Z_{m-2} \to Y_{m-2}Z_{m-1}$$

$$Z_{m-1} \to Y_{m-1}Y_m$$

4. Eliminate ϵ rules.
5. Eliminate the unit rules.

Lecture 9

Theorem:
Let Σ be an alphabet and let $A, B \subseteq \Sigma^*$ be context-free languages. These languages are also context-free $A \cup B, AB, A^*$.
Proof: Let $G_n = (V_A, \Sigma, R_A, S_A)$, $G_B = (V_B, \Sigma, R_A, S_B)$ be CFGs with $A = L(G_A), B = L(G_B)$.
assume WLOG that $V_A \cap V_B = \emptyset$
$S \to S_A | S_B$
$G = (V_A \cup V_B \cup S, \Sigma, R_A \cup R_B \cup \{S \to S_A, S \to S_B\}, S)$
Similar proof to AB and A^*
$AB = S \to S_A S_B$, $A^* = S_A S | \epsilon$

Theorem:
Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be regular, then A is context-free.
First proof: convert a regular expression to a CFG in a straightforward way.
Second proof: Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA with $L(M) = A$, we will define a CFG $G = (V, \Sigma, R, S)$ for A from M.
We will have one variable X_q for each state $q \in Q$
Also let $S = X_{q_0}$, rules of G.
For each $q \in Q$ and $a \in \Sigma$ include $X_q \to aX_{\delta(q,a)}$.
Also include $X_q \to \epsilon$ for all $q \in F$.
$Xq_0 \Rightarrow aX_{r_1} \Rightarrow a_1 a_2 X_{r_2} \Rightarrow a_1 a_2 \cdots a_n$ if $(r_n \in F)$.
$L(G') = A$ so A is context-free.

Theorem: Let Σ be an alphebat, let $A \subseteq \Sigma^*$ be a context-free language and let $B \subseteq \Sigma^*$ be a regular language. The language $A \cap B$ is context-free.
Let $G = (V, \Sigma, R, S)$ be a context-free gramma in chomsky normal form with $L(G) = A$ and $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA for B.
idea: for each variable X of G, we will have $|Q|^2$ variables, also include a new state variable S_0
$X_{p,q}$ for $p, q \in Q$ in H –¿ new CFG for $A \cap B$
$X_{p,q}$ will generate strings that:
1. could be generated by X in G and
2. take M from state p to state q.
Include rules in H as follows: 1. for each rule $X \to a$ in G, include $X_{p,\delta(q,a)} \to a$ in H for every state $p \in Q$
2. for each rule $X \to YZ$ includes $X_{p,q} \to Y_{p,r} Z_{r,q}$ for all $p, q, r \in Q$

3. $S \to \epsilon$ is a rule in G and $\epsilon \in B$ include rule $S_0 \to \epsilon$
4. Finally, include $S_0 \to S_{q_0,p}$ for every $p \in F$
claim: $L(H) = A \cap B$

Question:
Suppose $A \subseteq \Sigma^*$ is context-free, prefix(A) $= \{u \in \Sigma : ux \in A$ for some $X \in \Sigma^*\}$ is context-free.
Let G be a CFG for and assume that G is in chomsky normal form.
Assume G has no useless variables (i.e. variables that generate no strings).
Assume $A \neq \emptyset \cdots$ otherwise $Prefix(A) = \emptyset$ that is context-free.
We will define a new CFG H that generates $Prefix(A)$
idea: for every variable X appearing in G, we will have 2 variables in H.
1. X will generate exactly the same strings in H that did in G.
2. X_0 will generate all prefixes of strings generated by X.
Include rules in H as follows:
1. for each rule $X \to a$ in G, include

$$X \to a$$

$$X_0 \to a|\epsilon$$

2. for each rule $X \to YZ$ in G, include

$$X \to YZ$$

$$X_0 \to YZ_0|Y_0$$

Take S_0 as starts.

Lecture 10

Pumpling Lemma

Let Σ be an alphabet and $A \subseteq \Sigma^*$ be a context-free language.
There exist a pumpling length $n \geq 1$ for A that satisfies the following properties.
For everty string $w \in A$ with $|w| \geq n$, it is possible to write $w = uvxyz$, for $u, v, x, y, z \in \Sigma^*$, such that

$$v, y \neq \epsilon$$
$$|vxy| \leq n$$
$$uv^i xy^i z \in A \text{ for any i} \in \mathbb{N}$$

Proof:
Let G be a CFG in CNF with $L(G) = A$.
Let $m = |v|$, we will prove that the lemma holds for $n = 2^m$.
Suppose $w \in A$ satisfies $|w| \geq n$.
Pick any parse tree T for w, we know that T must have $2|w| - 1$ variable nodes with $|w|$ leaf nodes.
Must be at least one path from the root to a leaf with $\geq m + 1$ variable nodes.
There are only m different variables so some variable X must appear at least twice on this path.

Proposition:

Let $\Sigma = \{0, 1, 2\}$ and let $A = \{0^m 1^m 2^m : m \in \mathbb{N}\}$. We want to show that A is not context-free.
Proof:
Assume towards contradiction that A is context-free. Then the pumping lemma for context-free language says that there exist a pumping length $n \geq 1$ for A.
Let $w = 0^n 1^n 2^n$
We have $w \in A$ and $|w| = 3n \geq n$, therefore the properties of pumping lemma applies.
Say that we can write $w = uvxyz$, s.t.

$$1. v, y \neq \epsilon$$
$$2. |vxy| \leq n$$
$$3. uv^i xy^i z \in A \text{ for any i} \in \mathbb{N}$$

Since vxy has length at most n, it cannot contains both symbol 0 and the symbol 2.
Consider $i = 0$, $uv^0 xy^0 z = uxz \in A$.
Because $vy \neq \epsilon$, uxz has fewer than n $0s$, $1s$, or $2s$.
Bust we did not remove either 0 or 2.
Thus $uxz \notin A$, which contradicts the claim.
Therefore, A is not context-free.

Lecture 11

Pushdown Automata
A PDA is essentially just an NFA with a stack.

Definition:
A PDA is a 6-tuple, $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$:

1.Q is a finite and non-empty set of states
2.Σ is the input alphabet
3.Γ is the stack alphabet
4.δ is the transition function
$\delta : QX(\Sigma \cup stack(\Gamma) \cup \{\epsilon\} \to P(Q))$
where $stack\Gamma = \{\uparrow, \downarrow\} \times \Gamma$
5.$q_0 \in Q$ is the starting state
6.$F \subseteq Q$ is the set of accepting states

Example:
Let Γ be an alphabet consider strings over the alphabet, $stack(\Gamma) = \{\downarrow, \uparrow\} \times \Gamma$
$\Gamma = \{0, 1\}$
The transition can be as follows:

$(\downarrow, 0)$ means push 0 to the stack
$(\downarrow, 1)$ menas push 1 to the stack
$(\uparrow, 0)$ means pop 0 from the stack
$(\uparrow, 1)$ means pop 1 from the stack

A full PDA needs to add $(\downarrow, \diamond)$ at the very beginning to show that the stack has been added and $(\uparrow, \diamond)$ at the very end to show that the stack has been fully deleted.
Consider the language $A \subseteq stack(\Gamma)^*$ of all valid stack strings. This is a context-free language.

$$S \to \epsilon$$
$$S \to (\downarrow, a)S(\uparrow, a)S$$

Note that this CFG ensure a stack string has an empty stack at the end.
Remember a valid stack string may leave something in the end.

Definition:
$P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ $w \in \Sigma^*$
P accepts w if there exists $m \in \mathbb{N}$ a sequence of states, $r_0, r_1, \cdots, r_m \in Q$ and a sequence $a_1, a_2, \cdots, a_m \in \Sigma \cup stack(\Gamma) \cup \{\epsilon\}$, such that

1. $r_0 \in q_0$
2. $r_m \in F$
3. $r_{k+1} \in \delta(r_k, a_{k+1})$ for $k \in \{0, 1, \cdots m-1\}$
4. If we remove all symbols from stack(Γ) from $a_1, a_2, \cdots, a_m$ we obtain w
5. If we remove every symbol from Σ from $a_1, a_2, \cdots, a_m$ we obtain a valid stack string.

$L(P)$ = language of all strings w accepted by P

Lecture 12

Stack Machine
Idea: a stack machine is just like a PDA, but with multiple stacks.
We will focus on a Deterministic version of this model.

Non-deterministic stack machines(NSMs)

- Number of stack: r, index them as stack 0, stack 1, $\cdots$, stack $r-1$. Typically we will stack names like X, Y, Z, W.

- We will assume the input is loaded into stack 0 (the input stack)
  - more precisely, if we have an input $X \in \Sigma^*$, stack 0 starts like $a_1 a_2 \cdots a_n \diamond$ or $x \diamond$

Definition:
An r-stack NSM is an 7-tuple, $m = (Q, \Sigma, \Delta, \delta, q_0, q_{acc}, q_{rej})$, where:

1.Q is a finite and non-empty set of states
2.Σ is the input alphabet (with $\diamond \notin \Sigma$)
3.Δ is the stack alphabet (with $\Sigma \cup \{\diamond\} \subseteq \Delta$)
4.δ is the transition function of the form
$\delta : (Q\{q_{acc}, q_{rej}\}) \times \{\uparrow_0, \downarrow_0, \cdots, \uparrow_{r-1}, \downarrow_{r-1}\}$
5.$q_0 \in Q$ is the initial state
6.q_{acc} and q_{rej} are the accept and reject states. $q_{acc} \neq q_{rej}$

Input starts on stack 0, above $\diamond$.
All other stack initially contains $\diamond$

DSM-deterministic stack machine
Intuitively, a DSM is an NSM with these restrictions.

1. Every state has one stack associated with it.
2. Every state is either a "push state" or a "pop state"
3. If a state q is a push state, there must be exactly one transition leading out of state q.
4. If a state q is a pop state, there must be exactly one transition out of q for each stack symbol.

Note: popping an empty stack immediately goes to reject.
Also note that a DSM can potentially run forever.

Lecture 13

Definition:
A configuration of an r-NSM is an element of the set $Q \times (\Delta^x)^r$
idea: m is in configuration $(q, x_0, x_1, x_2, \cdots, x_{r-1})$
If the stack is q and its stack contents are described by $x_0, \cdots, x_{r-1}$.
Yeild-Relation:
We define the yield relation $\mapsto_m$, on $Q \times (\Delta^x)^r$ as follows:

1. For every choice of state $P \in Q \setminus \{q_{acc}, q_{rej}\}$ $q \in Q$
   a stack symbol $a \in \Delta$, and a stack number km such that $q \in \delta(p, \downarrow_k, a)$ the relationship contains every pair of the form $(p, x_0, \cdots, x_{r-1}) \mapsto_M (q, x_0, x_1, \cdots, ax_k, x_{k+1}, \cdots, x_{r-1})$ for all $x_0, \cdots, x_{r-1} \in \Delta^*$.

2. For every choice of state $P \in Q \setminus \{q_{acc}, q_{rej}\} q \in Q$
   a stack symbol $a \in \Delta$, and a stack number k, such that $q \in \delta(p, \uparrow_k, a)$ the relationship contians every pair of the form $(p, x_0, \cdots, x_{k-1}, ax_k, x_{k+1}, x_{r-1}) \mapsto_M (q_{rej}, x_0, x_1, \cdots, x_{k-1}, x_k, x_{k+1}, \cdots$
   for all $q_0, x_1, \cdots, x_{r-1} \in \Delta^*$

3. For every choice of state $P \in Q \setminus \{q_{acc}, q_{rej}\} q \in Q$
   a stack symbol $a \in \Delta$, and a stack number k, such that $q \in \delta(p, \uparrow_k, a)$ the relationship contians every pair of the form $(p, x_0, \cdots, x_{k-1}, \epsilon, x_{k+1}, x_{r-1}) \mapsto_M (q_{rej}, x_0, x_1, \cdots, x_{k-1}, \epsilon, x_{k+1}, \cdots, x_i$
   for all $q_0, x_1, \cdots, x_{r-1} \in \Delta^*$

$\mapsto_M^*$ takes 0 or more configuration. **Definition**
Let $m = (Q, \Sigma, \Delta, \delta, q_0, q_{acc}, q_{rej})$ be an r-DSM and let $w \in \Sigma^*$ be an input string.

1. M accepts w if $(q_0, w\Delta, \underbrace{\Delta, \cdots, \Delta}_{r-1}) \mapsto_M^* (q_{acc}, x_0, \cdots, x_{r-1})$ for some $x_0, \cdots, x_{r-1} \in \Delta^*$
2. M rejects w if $(q_0, w\Delta, \underbrace{\Delta, \cdots, \Delta}_{r-1}) \mapsto_M^* (q_{rej}, x_0, \cdots, x_{r-1})$ for some $x_0, \cdots, x_{r-1} \in \Delta^*$
3. M runs forever on w if it neither accepts nor rejects.

Let M be a DSM, $L(M) = \{w \in \Sigma^* : macceptsw\}$.
Note: it does not tell the complete story. Strings not in $L(M)$ might be rejected or cause M to run forever.

Decidable and Semi-decidable
Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a language.
A is decidable if there exist a DSM M such that

- $w \in A \Rightarrow$ M accepts w
- $w \not\in A \Rightarrow$ M rejects w

A is semi-decidable if there exist a DSM M with $A = L(M)$ (shown above)

Computable function

Let Σ and Γ be alphabet and let $f : \Sigma^* \to \Gamma^*$, we say that f is computable if there exist an r-DSM M (any r) such that $(q_0, w\diamond, \diamond, \cdots, \diamond) \mapsto_m^* (q_{acc}, f(x)\diamond, \diamond, \cdots, \diamond)$ for all $w \in \Sigma^*$.
Can generate to functions of the form:

$$f : (\Sigma^*)^M \to (\Gamma^*)^M$$

modeify definition so that $(q_0, w\diamond)$