1. (**30 points,** from Sipser, problem 2.6)
Give context free grammars generating the following languages (for parts (a) and (b), the alphabet is $\{\mathsf{a}, \mathsf{b}\}$; for part (c), the alphabet is $\{\mathsf{a}, \mathsf{b}, \#\}$):

   (a) **(10 points)** $\{w \mid \exists n \geq 0.\ (w = \mathsf{a}^n\mathsf{b}^{2n}) \vee (w = \mathsf{a}^{3n}\mathsf{b}^n)\}$.

   Solution:

$$\begin{array}{lcll} S & \rightarrow & X \mid Y, & S \text{ is the start symbol} \\ X & \rightarrow & \epsilon \mid \mathsf{a}\ X\mathsf{bb}, & X \text{ generates } \mathsf{a}^n\mathsf{b}^{2n} \\ Y & \rightarrow & \epsilon \mid \mathsf{aaa}\ Y\mathsf{b}, & Y \text{ generates } \mathsf{a}^{3n}\mathsf{b}^n \end{array}$$

   (b) **(10 points)** The complement of $\{w \mid \exists n \geq 0.\ w = \mathsf{a}^n\mathsf{b}^n\}$.

   Solution:

$$\begin{array}{lcll} S & \rightarrow & X \mid Y \mid Z, & S \text{ is the start symbol} \\ X & \rightarrow & \epsilon \mid W\mathsf{ba}W & X \text{ generates strings that don't match } a^*b^* \\ Y & \rightarrow & \mathsf{a}AC, & Y \text{ generates strings } a^ib^j \text{ with } i > j \\ Z & \rightarrow & CB\mathsf{b}, & Z \text{ generates strings } a^ib^j \text{ with } i < j \\ W & \rightarrow & \epsilon \mid \mathsf{a}W \mid W\mathsf{b}, & W \text{ generates all strings} \\ A & \rightarrow & \epsilon \mid \mathsf{a}A & A \text{ generates } a^* \\ B & \rightarrow & \epsilon \mid B\mathsf{b} & A \text{ generates } a^* \\ C & \rightarrow & \epsilon \mid \mathsf{a}C\mathsf{b}, & A \text{ generates } a^kb^k \end{array}$$

   (c) **(10 points)** $\{x_1\#x_2\#\cdots x_k \mid \text{each } x_i \in \{\mathsf{a}, \mathsf{b}\}^*, \text{ and for some } i \text{ and } j,\ x_i = x_j^{\mathcal{R}}\}$.

   Solution:

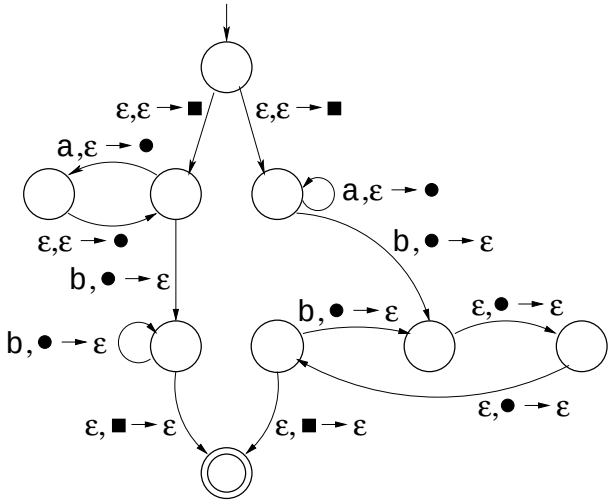
$$\begin{array}{lcll} S & \rightarrow & BWA, & S \text{ is the start symbol;} \\ B & \rightarrow & \epsilon \mid X\#B, & B \text{ generates the string before } x_i; \\ W & \rightarrow & \#M \mid \mathsf{a}W\mathsf{a} \mid \mathsf{b}W\mathsf{b}, & W \text{ generates } x_i\#Mx_j \text{ with } x_i = x_j^{\mathcal{R}}; \\ A & \rightarrow & \epsilon \mid A\#X, & B \text{ generates the string after } x_j; \\ X & \rightarrow & \epsilon \mid \mathsf{a}X \mid \mathsf{b}X, & X \text{ generates } (\mathsf{a} \cup \mathsf{b})^*; \\ M & \rightarrow & \epsilon \mid \mathsf{a}X\# & M \text{ generates string between } x_i \text{ and } x_j. \end{array}$$

   Note that M and A are identical. I wrote them as separate variables to have the idea of "before", "middle" and "after" strings.

2. (**30 points**) Give a PDA for each language from question 1. You can just draw a transition diagram where edges are labeled as in Sipser.

   (a) **(10 points)** $\{w \mid \exists n \geq 0.\ (w = \mathsf{a}^n\mathsf{b}^{2n}) \vee (w = \mathsf{a}^{3n}\mathsf{b}^n)\}$.

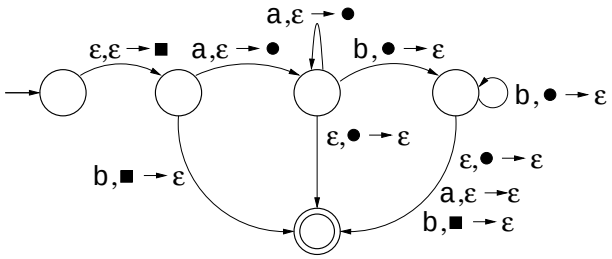
   Solution:



   The machine starts with a non-deterministic choice. The left branch matches $a^n b^{2n}$ and the right side matche $a^{3n}b^n$.

   (b) **(10 points)** The complement of $\{w \mid \exists n \geq 0.\ w = \mathsf{a}^n\mathsf{b}^n\}$.

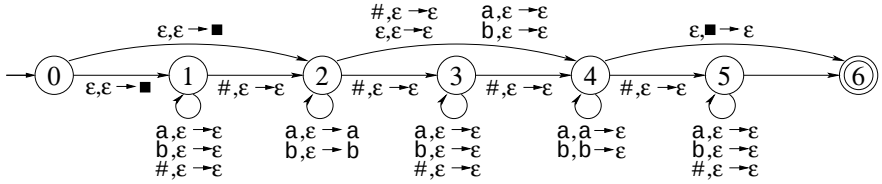
   Solution:



   My machine is based on a machine to recognize $a^n b^n$ with transitions to the accept state for everything where that machine would have rejected.

   (c) **(10 points)** $\{x_1\#x_2\#\cdots x_k \mid \text{each } x_i \in \{\mathsf{a},\mathsf{b}\}^*, \text{and for some } i \text{ and } j,\ x_i = x_j^{\mathcal{R}}\}$.

   Solution:



   My machine starts in state 0 from which it pushes a end-of-stack marker, ■, onto the stack and move to state 1. State 1 consumes input until x_i at which point the machine moves to state 2; the move from state 0 directly to state 2 is for the case that x_i is the first x string.
   In state 2, the machine reads a string of a's and b's, pushing them on the stack to compare later with x_j. Upon encountering a #, the machine moves to state 3 which consumes the x's between x_i and x_j. The move from state 2 directly to state 4 handles two special cases. Taking that arrow while reading a #handles the case that $j = i+1$. Taking the arrow while reading a a, bor ϵ handles the case that $i = j$

(and therefore that x_i is a palindrome). The problem as quoted from Sisper doesn't say whether or not the language should include the case that $i = j$. Either answer is acceptable. One point of extra-credit should be granted for stating that this is an issue and stating how their solution handles it.
In state 4, the machine pops a's and b's off the stack verifying that x_j indeed matches $x_i^{\mathcal{R}}$. State 5 discards the remaining input. The arrow from state 5 directly to state 6 handles the case that x_j is the last x string.

3. **(20 points)** Sipser, problem 1.13.
Let $G = (V, \Sigma, R, S)$ be the following grammar: $V = \{S, T, U\}$; $\Sigma = \{\mathtt{0}, \#\}$; and R is the set of rules:

$$\begin{array}{lcl} S & \to & TT \mid U \\ T & \to & \mathtt{0}T \mid T\mathtt{0} \mid \# \\ U & \to & \mathtt{0}U\mathtt{00} \mid \# \end{array}$$

(a) (**10 points**) Describe $L(G)$ in English.

Solution: $L(G) = \mathtt{0}^n\#\mathtt{0}^{2n} \cup \mathtt{0}^*\#\mathtt{0}^*\#\mathtt{0}^*$. In English, G generates all strings that are either have some number of 0's followed by a # followed by twice that many zeros, or that have two #'s. To see this, note that T generates $\mathtt{0}^*\#\mathtt{0}^*$.

(b) (**10 points**) Prove that $L(G)$ is not regular.

Solution:

Let $p$be a proposed pumping lemma constant for $L(G)$.

Let $w = \mathtt{0}^p\#\mathtt{0}^{2p}$.

Let x, y and z be strings such that $w = xyz$, $|y| > 1$ and $|xy| \leq p$.

$xy \in 0^*$. Thus, pumping the string changes the number of zeros to the left of the # without changing the number of zeros on the right side or the number of #'s. This creates a string that is not in $L(G)$.

4. (**25 points**, from Sipser, problem 2.19)
Let G be the CFG

$$\begin{array}{lcl} S & \to & \mathtt{a}S\mathtt{b} \mid \mathtt{b}Y \mid Y\mathtt{a} \\ Y & \to & \mathtt{b}Y \mid \mathtt{a}Y \mid \epsilon \end{array}$$

(a) (**15 points**) Give a simple description of $L(G)$ in English. Give a short explanation of your description.

Solution: $L(G)$ is the complement of a^nb^n.
Consider a string w that is generated by G. Let k be the largest integer such that there is a v such that $w = \mathtt{a}^k v \mathtt{b}^k$. This means that the derivation of w starts with k applications of the rule $S \to \mathtt{a}S\mathtt{b}$, and S must derive v. v must either start with a bor end with an a. In the first case, v is generated by $S \to \mathtt{b}Y$ because the variable Y generates any string. Likewise, in the second case, v is generated by $Y\mathtt{a}$. This shows that any string that is not in a^nb^n is generated by G.
If $w \in a^nb^n$, then w is not generated by G because neither $S \to \mathtt{b}Y$ nor $S \to Y\mathtt{a}$ can ever be applied.

(b) (**10 points**) Use your answer to part (a) to give a CFG for $\overline{L(G)}$, the complement of $L(G)$.

Solution: Let H be the grammar with start variable S_H and rules:

$$S_H \to \epsilon \mid \mathtt{a}S_H\mathtt{b}.$$

$L(H) = \overline{L(G)}$ as required.

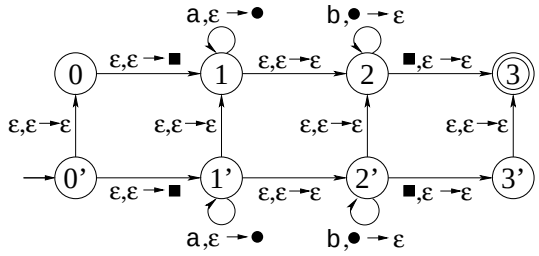
5. (**20 points**), Sipser, problem 2.25
For any language A, let $SUFFIX(A) = \{v \mid \exists u.\ uv \in A\}$. Show that the class of context-free languages is closed under the $SUFFIX$ operation.

Solution: Let A be a CFG and let $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA that recognizes A. We will create a new PDA, P_{SUFFIX} that recognizes $SUFFIX(A)$. The main idea is that P' introduces a new set of states, Q', that have mirror the states of P. The complete set of states for P' is $Q \cup Q'$. If P has a transition from state q_i to q_j when reading symbol c, then P' has that transition as well and a transition from q'_i to q'_j on input ϵ — both transitions do the same thing to the stack. Finally, P' has transitions from q'_i to q_i on input ϵ that don't modify the stack. The effect of the Q' states and transitions is that they allow P' to simulate what could happen on any input (i.e., the prefix, u). P' must transition to a state in Q to read any real input. The states in Q are used to check the suffix.

The paragraph above is a sufficient answer to the question. Here's an elaboration for those who want to see more details: Assume that $Q = \{q_1, q_2, \ldots, q_k\}$, and let $Q' = \{q'_1, q'_2, \ldots, q'_k\}$. Let $P' = (Q \cup Q', \Sigma, \Gamma, \delta', q'_0, F)$ with

$$
\begin{array}{rcll}
\delta'(q'_i, \epsilon, d) & = & \{(q'_j, e) \mid \exists c \in \Sigma \cup \{\epsilon\}.\ (q_j, e) \in \delta(q_i, c, d)\}, & \\
\quad \text{if } d \neq \epsilon & & & \\
\delta'(q'_i, c, d) & = & \emptyset, & \text{if } c \neq \epsilon \\
\delta'(q'_i, \epsilon, \epsilon) & = & \{(q'_j, e) \mid (\exists c \in \Sigma \cup \{\epsilon\}.\ (q_j, e) \in \delta(q_i, c, \epsilon))\} \cup \{(q_i, \epsilon)\} & \\
\delta'(p, c, d) & = & \delta(p, c, d), & \text{if } p \in Q
\end{array}
$$

Here's a transition diagram for a PDA built according to this construction that accepts $SUFFIX(\mathsf{a}^n\mathsf{a}^n)$:



6. (**25 points**), Sipser, problem 2.27
Let $G = (V, \Sigma, R, STMT)$ be the following grammar:

$$\begin{array}{rcl} STMT & \rightarrow & ASSIGN \mid IfThen \mid IfThenElse \\ IfThen & \rightarrow & \texttt{if condition then } STMT \\ IfThenElse & \rightarrow & \texttt{if condition then } STMT \texttt{ else } Stmt \\ ASSIGN & \rightarrow & \texttt{a:=1} \\ \\ \Sigma & = & \{\texttt{if}, \texttt{condition then}, \texttt{else}, \texttt{a:=1}\} \\ V & = & \{STMT,\ IfThen,\ IfThenElse,\ ASSIGN\} \end{array}$$

G is a natural-looking grammar for a fragment of a programming language, but G is ambiguous.

(a) Show that G is ambiguous.

Solution: The string

```
if condition then if condition then a:=1 else a:=1
```

Has two derivations:

First derivation:

$STMT$
$\rightarrow$ $IfThen$
$\rightarrow$ `if condition then` $STMT$
$\rightarrow$ `if condition then` $IfThenElse$
$\rightarrow$ `if condition then if condition then` $STMT$ `else` $STMT$
$\rightarrow$ `if condition then if condition then` $ASSIGN$ `else` $STMT$
$\rightarrow$ `if condition then if condition then a:=1 else` $STMT$
$\rightarrow$ `if condition then if condition then a:=1 else` $ASSIGN$
$\rightarrow$ `if condition then if condition then a:=1 else a:=1`

Second derivation:

$STMT$
$\rightarrow$ $IfThenElse$
$\rightarrow$ `if condition then` $STMT$ `else` $STMT$
$\rightarrow$ `if condition then` $IfThen$ `else` $STMT$
$\rightarrow$ `if condition then if condition then` $STMT$ `else` $STMT$
$\rightarrow$ `if condition then if condition then` $ASSIGN$ `else` $STMT$
$\rightarrow$ `if condition then if condition then a:=1 else` $STMT$
$\rightarrow$ `if condition then if condition then a:=1 else` $ASSIGN$
$\rightarrow$ `if condition then if condition then a:=1 else a:=1`

The first derivation corresponds to

```
if condition then
  { if condition then a:=1 else a:=1 }
```

and the second derivation corresponds to:

```
if condition then
  { if condition then a:=1 }
else a:=1
```

where I've added "{" and "}" to each to make the grouping clear. This is known as the "dangling else" problem (see http://en.wikipedia.org/wiki/Dangling_else).

(b) Give a new, unambiguous grammar for the same language.

Solution: Let $G' = (V, \Sigma, R, STMT)$ be the following grammar:

$$
\begin{array}{rcl}
STMT & \rightarrow & ASSIGN \mid IfThen \mid IfThenElse \\
IfThen & \rightarrow & \texttt{if condition then } STMT \\
IfThenElse & \rightarrow & \texttt{if condition then } STMT \texttt{ else } STMT2 \\
ASSIGN & \rightarrow & \texttt{a:=1} \\
STMT2 & \rightarrow & ASSIGN \mid IfThenElse \\
\\
\Sigma & = & \{\texttt{if}, \texttt{condition then}, \texttt{else}, \texttt{a:=1}\} \\
V & = & \{STMT,\ IfThen,\ IfThenElse,\ ASSIGN\}
\end{array}
$$

Grammar G' generates the same language as grammar G. Variable $STMT2$ generates the same strings as $STMT$ except that it can't generate an $IfThen$. This forces the "`else`" clause of an $IfThenElse$ statement to be bound the nearest `if`. This corresponds to the way that this is handled in programming languages that don't have an explicit `end` or `endif` for `if`–statements.