420/520 Automata Theory Midterm

Undergrads can pick 5 of the questions to answer and graduate students need to do all 6. Undergrads are of course welcome to do all 6 if they like. You do not need to do proofs of correctness for constructions of finite automata or grammars. Midterm is due Tues November 5.

1. Design a deterministic finite automata for the set of strings over the alphabet \{a, b\} containing at least three occurrences of three consecutive b’s, overlapping permitted (e.g., the string \textit{bbbb} should be accepted.)

\textbf{Solution} 1 Here is a DFA for this language. Igor’s solution had the best layout for the graph and I’ve used it here.
2. Give a regular expression for the following DFA.

\[
(a^*b)^2(a \cup b^+a^+b)^*
\]

**Solution 2** There’s lots of correct expressions. Here is a short one.

3. Consider the following language:

\[
A = \{w \in \{a,b\}^* | w \text{ has both } aa \text{ and } bb \text{ as substrings } \}
\]

Construct a regular grammar for \( A \) and a finite automata for \( A \).

**Solution 3**

Here’s the regular grammar:

\[
\begin{align*}
S & \rightarrow aA \mid bB \\
A & \rightarrow aC \mid bB \\
B & \rightarrow aA \mid bD \\
C & \rightarrow aC \mid bE \\
D & \rightarrow aF \mid bD \\
E & \rightarrow aC \mid bG \\
F & \rightarrow aG \mid bD \\
G & \rightarrow aG \mid bG \mid \epsilon
\end{align*}
\]

Here’s a DFA.
4. Sipser 1.38

**Solution** 4 The conditions under which an *all*-NFA rejects a string are exactly the conditions under which an NFA accepts a string and vice versa. The definition of *all*-NFAs and NFAs are the same in all other aspects. In essence, an *all*-NFA $N$ is already an NFA for the strings that $N$ rejects. Similarly, the rejecting computations of NFA $M$ are an *all*-NFAs recognizing $L(M)$. Because regular languages are closed under complement we have that *all*-NFAs recognize the regular languages.\(^1\)

If we wanted to be more formal we can do something like this. Given an *all*-NFA $N = (Q, \Sigma, \delta, q_0, F)$, we can construct an NFA $M = (Q, \Sigma, \delta, q_0, Q - F)$. We’ve changed our definition of acceptance from all paths accepting to there exists a path accepting the input, and we’ve swapped the accept and reject states. Machines $M$ and $N$ share the same computations paths for any input $w$. If $N$ accepts $w$, then $M$ will reject it and vice versa so $M$ recognizes $\overline{L(N)}$. $\overline{L(N)}$ is regular since it’s recognized by an NFA and so is $L(N)$ by closure of regular languages under complement. So we’ve shown that the languages recognized by *all*-NFAs are regular.

Now we just need to show that every regular language is recognized by some *all*-NFA. Every regular language is recognized by a DFA and a DFA is an *all*-NFA. We can conclude that *all*-NFAs recognize the class of regular languages.

\(^1\)There are some subtle issues about when a computation halts in an NFA computation. In the Sipser book, an NFA computation halts if it enters an accept state after reading the entire input. However, if we’ve read all the input, we’re allowed to keep following epsilon transitions if an accept state hasn’t been encountered. Sipser fails to mention this explicitly. We solve this if we swap the accept states with the reject states ensuring halting for accepting computations (previously rejecting computations).
The other way to prove all-NFAs recognize the regular languages is to apply the construction for converting NFAs to DFAs to an all-NFA. This is the approach that most of you took. The only adjustment to the construction we need to make is for the set of acceptance states. In the original construction we required the final state to contain at least one element from the original set of accept states. Now we require a non-empty set containing only elements from our original set of accept states. Given \( Q \) as our original states and \( F \) our original accept states, we go from

\[
\{ R \in \mathcal{P}(Q) \mid R \cap F \neq \emptyset \}
\]

to

\[
\{ \mathcal{P}(F) - \emptyset \}
\]

This construction shows that the languages recognized by all-NFAs are regular. We can borrow the second half the previous proof to show that all regular languages are recognized by some all-NFA and we’re done. Again, we conclude that all-NFAs recognize the class of regular languages.

5. For the following languages, state whether the language is regular or not. If the language is regular give some sort of construction as proof (DFA, NFA, regular expression, regular grammar). If the language is not regular use the pumping lemma to prove it is not regular. Note: \( \equiv_5 \) denotes equivalence modulo 5.

\[
B = \{ a^i b^j \mid i, j \geq 0 \text{ and } i - j = 5 \}
\]

\[
C = \{ a^i b^j \mid i, j \geq 0 \text{ and } |i - j| \equiv_5 0 \}
\]

**Solution 5** The language \( B \) is not regular.

*Proof.* Proof by contradiction. Assume \( B \) is regular. Then there is \( p \in \mathbb{N} \) such that for all strings \( s \in B \) with \( |s| \geq p \) we can pump \( s \) using the pumping lemma. Let \( s = a^{p+5}b^p \). \( s \) can be written as \( xyz \) such that \( |y| > 0 \) and \( |xy| \leq p \). When we pump \( s \), the non-empty string \( y \) will contain only a’s because \( |xy| \leq p \). If the length of \( y \) is \( k \) then the pumped string \( xy^2z = a^{p+5+k}b^p \) is not in the language \( B \) since \( p + 5 + k - p > 5 \). This is a contradiction so we can conclude that \( B \) is not regular.

The language \( C \) is regular. Some people gave the regular expression

\[
\bigcup_{k=0}^{4} a^k (a^5)^* b^k (b^5)^*
\]

Others gave a DFA construction.
6. Give a grammar for the following language.

\[ \{ w_1 w_1^R w_2 w_2^R \ldots w_n w_n^R \mid w_i \in \{0, 1\}^* \text{ and } i \geq 0 \} \]

**Solution 6**

\[
S \rightarrow SP \mid S \mid \epsilon \\
P \rightarrow 0P0 \mid 1P1 \mid \epsilon
\]

7. (Extra Credit) Let \( A/B = \{ w \mid wx \in A \text{ for some } x \in B \} \). Show that if \( A \) is context free and \( B \) is regular, then \( A/B \) is context-free.

**Solution 7**

*Proof*. We can show that \( A/B \) is context-free by constructing a PDA that recognizes it. Let \( M_A \) be a PDA for \( A \) and \( M_B \) be a DFA for \( B \). Our new PDA \( M \) will scan the input \( w \) and when it reaches the end of \( w \) it nondeterministically jumps to a part of the machine that uses nondeterminism to run all possible suffix strings in parallel through \( M_A \) and \( M_B \). This simulation will pick up in \( M_A \) wherever it left off after reading \( w \) using what ever was left on the stack when we reached the end of \( w \). If this nondeterministic simulation accepts, then there exists a suffix \( x \) such that \( wx \in A \) and \( x \) is in \( B \). It follows that \( w \) is in \( A/B \).

So how does a machine know when it reaches the end of an input? The Sipser book only mentions this briefly. One way to do this is to stick a special symbol at the end of all our inputs. This is the same trick we used to recognize an empty stack. Instead of recognizing the language \( A/B \), we will recognize the language \( A/B\$. 

5
If $A/B\$ is context free then we can use the cute closure under homomorphism trick to get rid of the $\$ at the end (see below in the next version of this proof). Let $h$ be a homomorphism that maps $h(a) = a$ if $a \in \Sigma$ and $h(\$) = \$. $A/B = h(A/B\$) so $A/B$ is context-free.

Here is a description of the machine $M$ built from $M_A = (Q_A, \Sigma, \Gamma, \delta_A, q_{A,0}, F_A)$ and $M_B = (Q_B, \Sigma, \delta_B, q_{B,0}, F_B)$.

(a) $Q = Q_A \cup (Q_A \times Q_B)$
(b) $\Sigma' = \Sigma \cup \{}$ augmented alphabet
(c) $\Gamma' = \Gamma \cup \{}$ augmented tape alphabet
(d) if $q \in Q_A$ then we are still reading the input. There are two options. If we haven’t hit the end of the input, we proceed as usual using the transition function $\delta_A$. If we read the end of the input symbol $\$, then we jump to our simulation, starting in the state $(q, q_{B,0})$. When we jump we make sure not to push or pop anything from the stack.

$$\delta(q, a, b) = \delta_A(q, a, b) \text{ whenever } a \in \Sigma$$

$$((q, q_{B,0}), \epsilon) \in \delta(q, \$, \epsilon) \text{ for all } q \in Q_A$$

(e) if $q = (r, s) \in Q_A \times Q_B$, then we are simulating all the possible suffixes for our input in parallel on $M_A$ and $M_B$. All of these transitions will be $\epsilon$ transitions since there is no input left. At each transition we will try every possible alphabet symbol.

$$\delta((r, s), \epsilon, b) = \bigcup_{c \in \Sigma} \{((r', s'), d) \mid (r', d) \in \delta_A(r, c, b) \text{ and } s' \in \delta_B(s, c)\}$$

The PDA $M_A$ is allowed to take $\epsilon$ transitions during the simulation that may change the stack contents. If the value of $c$ in the above expression is $\epsilon$ we can assume that the DFA $M_B$ just stays in the same state by defining $\delta_B(s, \epsilon) = s$ for all $s \in Q_B$.

(f) $q_0 = q_{A,0}$

(g) $F = \{(q_a, q_b) \mid q_a \in F_A \text{ and } q_b \in F_B\}$

Here’s a very cute proof for this problem that I found in a book. It makes use of closure under homomorphisms for context-free languages. Probably a good tool to add to our bag of tricks.

Proof. Make a marked copy of the alphabet. For example, if $\Sigma = \{a, b\}$ construct the alphabet \{a', b'\}. Let $h$ be the homomorphism that erases marks; that is $h(a) = h(a') = a$ and $h(b) = h(b') = b$. Let $g$ be the homomorphism that erases the marks on the marked symbols; that is $g(a) = g(b) = \epsilon$, $g(a') = a$, $g(b') = b$. Then

\footnote{Regular languages are also closed under homomorphism}
\[ A/B = g(h^{-1}(A) \cap \{a', b'\}^*B) \]

CFL are closed under homomorphic preimage, intersection with regular set, and homomorphic image. The set \( h^{-1}(A) \) is the set of strings that look like strings in \( A \) with some of the letters marked. \( \{a', b'\}^*B \) is a regular expression for all possible marked prefixes concatenated with unmarked suffix strings in \( B \). When we take the intersection we get strings \( w'x \) where the symbols in \( w' \) are marked and the symbols in \( x \) are not. Let \( w \) be the unmarked version of \( w' \). We have that \( wx \in A \) and \( x \in B \). Applying the homomorphism \( g \) removes all the letters in \( x \) (since they are unmarked) and turns the marked symbols in \( w' \) to the unmarked version \( w \).