15-453 Formal Languages, Automata, and Computation

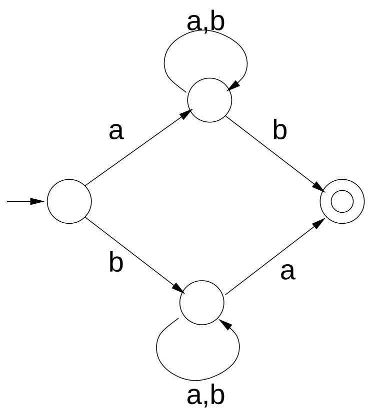
Model Solutions of the Practice Midterm

Note that the several problems have many correct solutions; the answers below give only one.

Problem 1

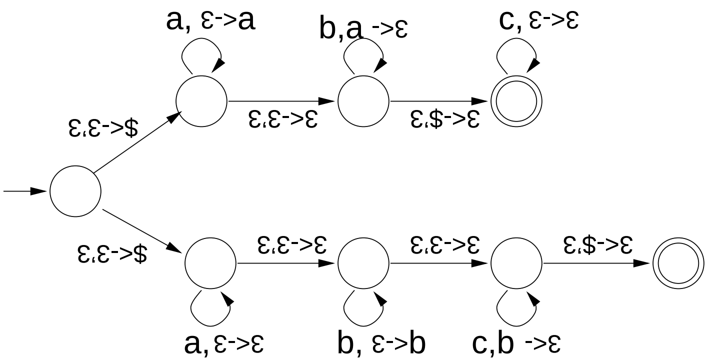
a) By inspection, we notice that the upper branch recognizes all words which begin in a and end in b. The lower branch is symmetric. A regular expression which describes the whole is $(a(a \cup b)^*b) \cup (b(a \cup b)^*a)$.

b)



Problem 2

a) We notice that we can write L as the union of two simpler languages: $L = \{a^n b^n c^k \mid n, k \geq 0\} \cup \{a^k b^n c^n \mid n, k \geq 0\}$. It is easy to write a PDA for each of these. The resulting PDA should be the union of the two PDAs:



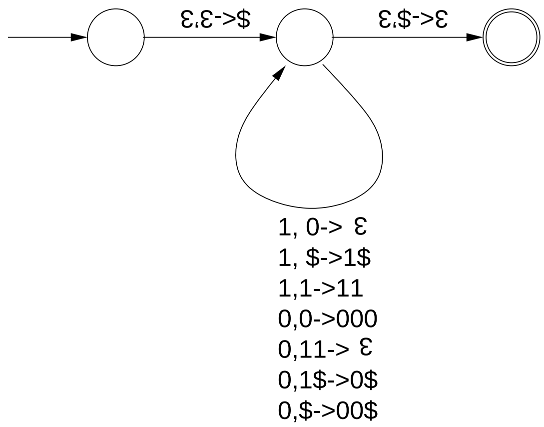
The first transition marks the bottom of the stack with a \$. The upper branch is nondeterministically selected to check the pairing between a's and b's, irrespective of the number of c's, while the lower branch discards all a's and next matches b's and c's.

b) Let us assume L is regular. Then let p be the pumping length. Consider the word $w = a^p b^p c^{p+1} \in L$. The pumping lemma says there are x, y, z such that $|xy| \leq p$ and $|y| \geq 1$ such that $w = xyz$, and $xy^i z \in L$ for any $i \geq 0$.

The conditions imply that $y = a^k$ for some $k > 0$. But taking $i = 0$ we obtain the word $a^{p-k} b^p c^{p+1}$ which obviously is not in L, as the three exponents are all different. This is a contradiction, so L couldn't have been regular in the first place.

Problem 3

The language is context-free. Here is a Big-step PDA which accepts it:



The idea is to keep track on the stack of the unmatched symbols seen so far; we are always matching a 0 with two ones. At each point the stack only contains either only zeros, or only ones. When reading a zero, it is transformed into a pair of zeros, which are pushed on the stack. If the stack has ones, they are popped, otherwise the zeros are pushed. The same goes for ones.

We can prove by induction on the length of a word that the state of the stack counts the number of ones seen minus twice the number of zeros seen.