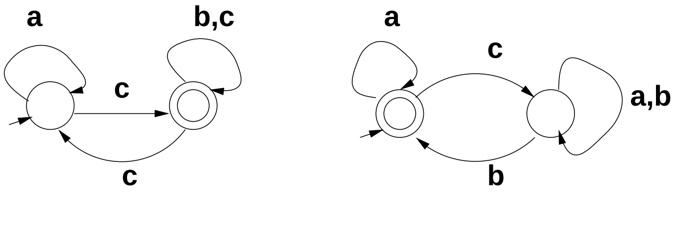
CS615 Spring 2004 Prerequisites Quiz

Time: 90 mins Total marks:

- ***Unnecessarily lengthy solutions will be penalized.***
- ***If you need to make any assumptions, state them clearly.***
- ***Do not copy solutions from others or indulge in unfair means.***

1. Let A_1 and A_2 be non-deterministic finite automata (NFA) over the alphabet $\Sigma = \{a, b, c\}$. Let L_1 and L_2 denote the languages of finite words accepted by A_1 and A_2 respectively. We wish to construct an automaton A (either deterministic or non-deterministic) that accepts all finite words that are present in L_1 or L_2 but not in both.

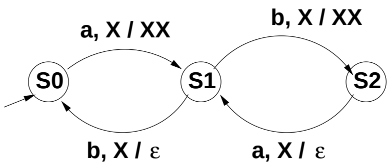


A1 **A2**

   (a) If A_1 and A_2 are as shown in Fig. 1, construct the automaton A. You must clearly explain your reasoning in the construction. Solutions without justification will be deemed to have been copied or randomly arrived at, and will fetch no credit.

   (b) Generalizing the above problem, if NFA A_i contains n_i states, give as small an upper bound as possible on the number of states of A (may be non-deterministic).

2. Consider the push-down automaton (PDA) shown in Fig. 2. The notation $S_i \overset{a,X/YZ}{\longrightarrow} S_j$ used in this figure signifies that when the PDA is in state S_i with X on top of the stack, if the input symbol read is a, then the PDA pops X from its stack, pushes Z and then Y into the stack and transitions to state S_j. If the transition is $S_i \overset{a,X/\epsilon}{\rightarrow} S_j$, this signifies that the PDA pops X but does not push back any other symbol in the stack. We assume that the PDA starts in state S_0 with a single X in its stack. Let L_1 denote the language accepted by the PDA by emptying its stack. We wish to know if L_1 is regular, i.e., if L_1 can be accepted by a deterministic finite automaton (DFA).



   - If you think that L_1 is regular, you must give a (DFA) for L_1 and provide justification for your construction.
   - If you think that L_1 is not regular, you must provide justification for your conclusion.

3. Let $\mathcal{D}$ be the set of all days (an infinite set) and let x, y, z be variables that assume values from $\mathcal{D}$. Consider the following predicates defined on $\mathcal{D}$.

   - $\mathsf{Shine}(x)$: This predicate evaluates to True if and only if (iff) the sun shines on day x.
   - $\mathsf{Flood}(x)$: This predicate evaluates to True iff there is a flood on day x.
   - $\mathsf{Later}(x, y)$: This predicate evaluates to True iff day y comes after (not necessarily immediately after) day x.
   - $\mathsf{Cons}(x, y)$: This predicate evaluates to True iff day y is the next day after day x.

   Using the above predicates and no other predicates, express the following English language statements as first-order logic *sentences* (i.e., no free variables):

   (a) There are days when the sun shone for the last three consecutive days and yet there is a flood.

   (b) Whenever the sun shines for four consecutive days, there is no flood for the next two days.

   (c) If there is a flood on two different days separated by at least one day in between, the sun does not shine on at least one day after the first day of flood and before the second day of flood.

4. Consider the first-order logic sentences $\phi = \exists x.\forall y.\, ((P(x, y) \wedge P(y, z)) \rightarrow P(x, z))$ and $\psi = \forall x.\exists y.\, ((P(x, y) \wedge P(y, z)) \rightarrow P(x, z))$.

   Give a model M, i.e., a domain and definition of predicate P, such that $M \models \phi$ but $M \not\models \psi$.

5. You are given a directed graph $G = (V, E)$ with an unknown (potentially very large) number of vertices. You are however told that G has the following properties:

   - The maximum outdegree (i.e., maximum number of edges going out) from any vertex $v \in V$ is k.
   - There is a designated start vertex $v_0 \in V$.
   - There is a cycle C of length at most c in G such that the shortest path from v_0 to any vertex in C is of length at most d. Note that the shortest path from vertex v to u is the minimum number of edges that need to be traversed in reaching u from v.

   Give C-like pseudo-code for an algorithm that starts from vertex v_0 and finds the cycle C (i.e., lists the vertices in C) described above. Your algorithm must always terminate for any graph G with the above properties. What is the asymptotic worst-case complexity of your algorithm?