Harvard University
Computer Science 121

Problem Set 1

Due Friday, September 24, 2010 at 1:20 PM.
Submit a single PDF (lastname+ps1.pdf) of your solutions to cs121+ps1@seas.harvard.edu
Late problem sets may be turned in until Monday, September 27, 2010 at 1:20 PM with a 20% penalty.
See syllabus for collaboration policy.

Name

Problem set by !!! Your Name Here !!!

with collaborator !!! Collaborators' names here !!!!

Notes

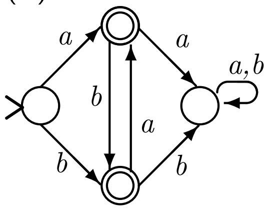
DFAs and NFAs are tough to draw in LaTeX, we recommend using an image editor like Paint, PowerPoint, drawing by hand and scanning in, etc., saving the image to a file and importing the file. The TEX file of this problem set contains the necessary code to import graphics where needed.

When not stated, assume $\Sigma = \{a, b\}$

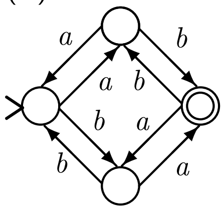
PROBLEM 1 (3+3 points)

Describe informally the language represented by each of the deterministic finite automata below.

(A)



(B)



Common mistake: ϵ was often not considered, especially as being excluded by the first language.

(A) Nonempty strings alternating $a's$ and $b's$.
(B) Strings with an odd number of $a's$ and an odd number of $b's$.

PROBLEM 2 (4+4 points)

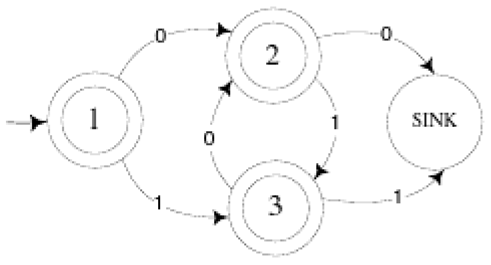
For the following two languages L draw a DFA that recognizes L and give its 5-tuple representation (from Sipser pg. 35) over the alphabet $\{0, 1\}$:

(A) L is the set of all strings containing no consecutive 1s and no consecutive 0s.
(B) L is the set of all strings starting with 00 and ending with 00.

Some common problems were failure to consider the empty string, which should be part of the first language (it does not have consecutive ones or zeros) and failure to realize that the strings 00 and 000 need to be accepted by the DFA for the second language.

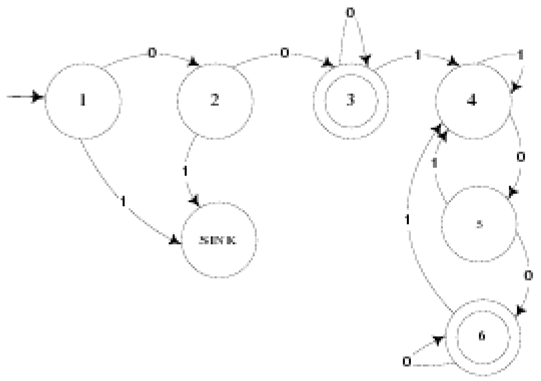
(A) $\{\{1, 2, 3, SINK\}, \{0, 1\}, \delta, 1, \{1, 2, 3\}\}$ where $\delta =$

q	0	1
1	2	3
2	SINK	3
3	2	SINK
SINK	SINK	SINK



(B) $\{\{1, 2, SINK, 3, 4, 5, 6\}, \{0, 1\}, \delta, 1, \{3, 6\}\}$ where $\delta =$

q	0	1
1	2	SINK
2	3	SINK
SINK	SINK	SINK
3	3	4
4	5	4
5	6	4
6	6	4



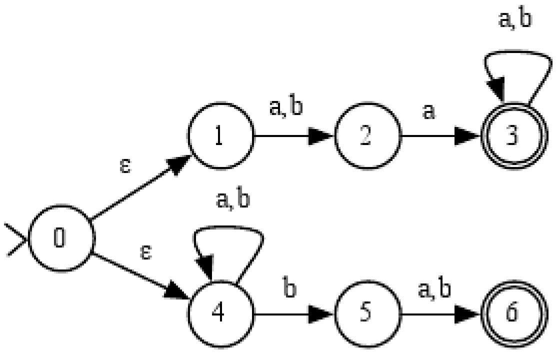
PROBLEM 3 (4+8 points)

(A) Draw an NFA that recognizes the language of all strings with the substring *aba*.

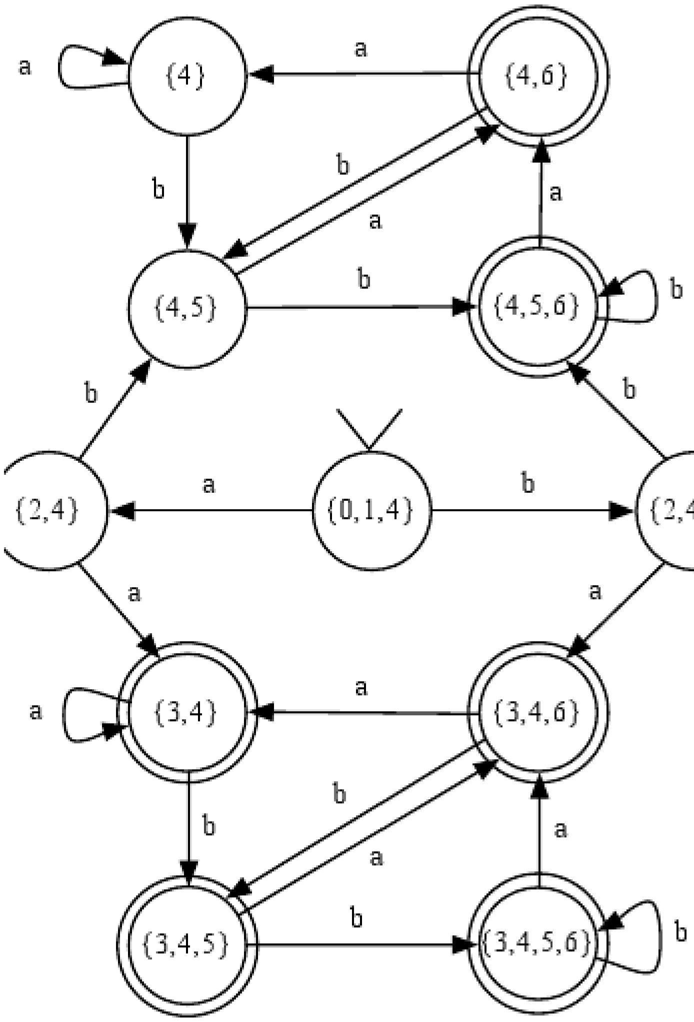
(B) Convert your NFA from part (A) to a DFA using the subset construction.

A particularly elegant solution to this problem is drawing the DFA first, the conversion is then trivial. Here is a sample DFA and the corresponding construction:

(A)



(B)



PROBLEM 4 (4+4+4 points)

Are the following statements true or false? Justify your answers with a proof or counterexample.

(A) For any languages L_1 and L_2, $(L_1 \cap L_2)^* = L_1^* \cap L_2^*$.

(B) For any languages L_1 and L_2, $(L_1 \cup L_2)^* = L_1^* \cup L_2^*$.

(C) If L is a regular language, then the language of all the strings in L which do not contain ab is regular. (Hint: Regular languages are closed under intersection.)

(A) *False.* Let $L_1 = \{a\}$ and $L_2 = \{aa\}.(L_1 \cap L_2)^* = \emptyset^* = \{\epsilon\}$. But $aa \in L_1^* \cap L_2*$.
(B) *False.* Let $L_1 = \{a\}$ and $L_2 = \{b\}$. Then $(L_1 \cup L_2)^* = \{a, b\}^*$, or Σ^*, but $L_1^* \cup L_2^* = \{\epsilon, a, aa, aaa \ldots\} \cup \{\epsilon, b, bb, bbb \ldots\}$, which defines strings consistint of any number of $a's$ or any number of $b's$, but not both. The latter clearly does not include every possible string.
(C) *True.* The language of strings which do not contain ab is regular and the intersection of two regular languages is regular. We need to demonstrate that the former language is regular, however, since it's not given. We can see this by considering the regular expression b^*a^* which generates this language. Since the language is recognized by a regular expression it is regular. Alternatively a DFA or NFA could be easily constructed.

PROBLEM 5 (5+5 points)

An NFA M contains a *cycle* if there is a state q and a string x such that if M is in state q and reads string x, M can return to state q. Prove or disprove the following statements:

(A) If M recognizes an infinite language, then M has a cycle.
(B) If M has a cycle, then M recognizes an infinite language.

(A) Proof by contradiction. Suppose the NFA M has n states and recognizes an infinite language but does not contain a cycle. Since $L(M)$ is infinte there exists a string $w \in L(M)$ of length greater than n (infinite languages do not have maximum length strings, so for any fixed value there must be a string of greater length in the language). Now consider an accepting computation path of w in M. By the pigeonhole principle at least one state q must appear at least twice in the computation path, since $|w| > n$. However, this would require a cycle, a contradiction.

A common confusion on this problem was the length of the strings in an infinite language. While the language contains an infinite number of strings no string is of infinite length (all strings have finite length) and strings of a particular length are not guaranteed to be in the language (it's easy to consider a language with no strings of length 10 in it, for example). However, there is always a string of length greater than some fixed integer.

(B) This is false. Consider an NFA consisting of only a single state which is its start state but not an accepting state. It accepts no strings, not even the empty string. Now construct a transition from this state to itself for any element of the alphabet. This is a cycle but the language is unchanged and certainly not infinite.

PROBLEM 6 (5+5 points)

(A) For every $n \geq 6$ divisible by 3, prove that there is an undirected graph with exactly n nodes, each of which has degree 4.

(B) Prove that there is no undirected graph with any *odd* number of nodes with the property that every node has degree 3. (Hint: Every edge connects to two nodes.)

(A) The easiest way to construct this is to consider placing all vertices in a ring and connect each vertex with its four nearest neighbors, defined naturally by adjacency in the ring. This gives every vertex degree 4 because this connection is symmetric–a node is neighbor to another only if that node is also its neighbor. An alternative scheme is to form a ring and connect adjacent nodes as before, but instead of forming additional connections to slightly more distant neighbors a labelling around the ring is constructed where each node forms a triangle with two other vertices. The ring connection gives degree two and the triangle gives an additional degree two. In particular, starting at an arbitrary point assign natural numbers to vertices in the ring and connect vertex i to $i + \frac{n}{3}$, well defined since the number of vertices is a multiple of three.
(B) Proof by contradiction. Suppose there is an undirected graph with an odd number of nodes where every node has degree three. The sum of the degrees of the graph is $3n$, an odd number. However, we know the number of edges in the graph must be the sum of the degrees over two. Dividing an odd number by two doesn't produce a natural number, and since we can't have a half edge we have a contradiction.

PROBLEM 7 (Challenge!!! 1 points)

Consider a regular language L. Prove that the language consisting of all strings which are the first half of a string in L is also regular. More formally, prove that the language HALF(L) = $\{w \mid \exists w' \in \Sigma^* \text{ s.t. } |w| = |w'| \text{ and } ww' \in L\}$ is regular. You may ignore odd length strings.

(A) The intuition is that we'll parse the half string w as if it were a complete string in L while simultaneously nondeterministically "guessing" a string of equal length, w', and determining if they together form a string in L. Importantly, there only need exist such a w' for w to be a string in HALF(L). We can't begin parsing w' from the correct point, however—we don't know where that would be—so instead we parse it from right to left, going from the final states of the NFA backwards.

Elaborating with some mathematics. Consider an NFA for L, M. We also consider the powerset of states of M, $P(Q)$. Since M has finite states $P(Q)$ is a finite set. We construct a new NFA to recognize HALF(L), M', which has states Q' representing a tuple consisting of two elements. The first is the state from M the half string would be in as parsed normally, $q* \in Q$, the second is the set of states (element of the power set of states of $M, p^* \in P(Q)$) for which a string of equal length might have reached a final state in M. Accepting states in M' are states where $q^* \in p^*$.

First, let's note the number of states described is finite. The power set is finite and the number of states in the original NFA is certainly finite. The maximum number of states is then $|Q|^3$. These are the states M' will consider. Additionally, note that the accepting states are previously defined. This leaves us to define the transition function and start state (the alpahbet is given in the note at the top of the problem set).

As mentioned states are labelled $(q \in Q, p \in P(Q))$. We can state the transition function pairwise, too, in a slight abuse of notation for clarity. First, we have δ_M, the original transition function for the NFA M, and we use that to update the q value of the tuple. Second, we define a transition function on the element of the tuple from the powerset to output, for any character in the alphabet, the set of all states in M which have transitions (arrows) into any $p^* \in p$; we call this set INTO(p). The total transition function is then $\delta((q,p),\Sigma) \to (\delta_M(q,\Sigma)$,INTO$(p))$.

Finally we need to pick a start state. Following our intuition it should have $q = q_0$, but which element of the powerset should we start at? The answer is the set of states which are accepting in M, F. So we start at (q_0, F).

We now have a well defined NFA, M'. Following our argument, we argue for correctness by noting that the set of states of the second element of the tuple contains all states for which a string of the same length as the string already seen may reach an accepting state from. This is because our transition function counts back from accepting states one character at a time as we parse one character from the left. The first element of the tuple obviously is the state M would be when parsing that particular string. When parsing is finished, then, we accept if the state we'd be at in M is also one which might reach an accepting state in M with additional characters of the same length—exactly HALF(L).