Problem Set 5

Spring 10: CS373

Due: Problems 1-4 due on Friday, March 12, by 5pm, in Elaine Wilson's office, 3229 SC, by 4pm.

Extra credit Problem 5 due on Thursday, March 18th, in class before class begins, at 2pm.

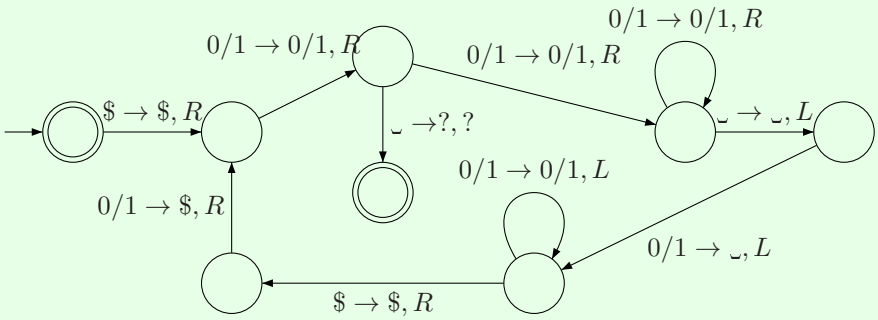
Please follow the homework format guidelines posted on the class web page:

`http://www.cs.uiuc.edu/class/sp10/cs373/`

1. Turing Machines [**Category**: Construction, **Points**: 20]

   We want to design a Turing machine that given a binary string $\$x$, where $x \in \{0,1\}^*$ is of odd length, computes the middle character of x. To do this our TM overwrites the first and last character of the string x with $\sqcup$, and repeats this procedure till only one character survives on the tape. Then the machine stops and accepts. Carefully draw the diagram of such a TM with input alphabet $\{0,1,\$\}$ and tape alphabet $\{0,1,\$,\sqcup\}$.

Solution:



2. Non-Regularity [**Category**: Proof, **Points**: 20]

   Prove that the following language is not regular:

   $$L = \{0^{n^3} : n \geq 0\}$$

   Hint: If you are using the Myhill-Nerode Theorem, you may want to choose S to be L.

Solution:

We will prove this using MNT. We show that all the strings of the form 0^{i^3} are distinguishable and therefore L has an infinite number of suffix languages and cannot be regular. Let S be L. Consider two arbitrary strings from S: 0^{i^3} and 0^{j^3} with $0 \leq i < j$.

Let $z = 0^{(i+1)^3 - i^3}$. Obviously $0^{i^3}z = 0^{(i+1)^3} \in L$. We need to show that $0^{j^3}z \notin L$. We know that $|0^{j^3}z| = j^3 + (i+1)^3 - i^3$ and

$$0 \leq i < j \implies 3i^2 + 3i + 1 < 3j^2 + 3j + 1 \implies (i+1)^3 - i^3 < (j+1)^3 - j^3 < (j+3)^3$$

$$\implies |0^{j^3}| < |0^{j^3}z| < |0^{(j+1)^3}|$$

Since L has no element with length more than j^3 and less than $(j+1)^3$, we have that $0^{j^3}z \notin L$.

3. Non-determinism [**Category**: NFA design, **Points**: 20]

Let P be a regular language over $\{0, 1\}$. Let L_0 and L_1 be two regular languages over $\{a, b\}$.

Let L be the set of words w over $\{a, b\}$ such that w can be split into n words (for some n), $w = w_1 w_2 \dots w_n$, and there exists a word of length n, $x = x_1, x_2 \dots x_n \in P$ such that each $w_i \in L_{x_i}$. In other words,

$$L = \{w \in \{a, b\}^* \mid \exists x \in P, x = x_1 \dots x_n, x_i \in \{0, 1\}, w_i \in L_{x_i}, w = w_1 \dots w_n\}$$

Intuitively, L consists of words formed by concatenating words in L_0 and L_1, using a pattern described in P.

Show L is regular by exhibiting an NFA for it. Give the formal description of your NFA. You do not need to prove formally that your NFA accepts this language, but you do need to give a description of how and why it works.

Solution:

Let $A_p = (Q_p, \{0, 1\}, \delta_p, q_p^0, F_p)$, $A_0 = (Q_0, \{a, b\}, \delta_0, q_0^0, F_0)$ and $A_1 = (Q_1, \{a, b\}, \delta_1, q_1^0, F_1)$ be the DFAs accepting P, L_0 and L_1 correspondingly. We will construct an NFA $A = (Q, \{a, b\}, \delta, q_0', F)$ that accepts L.

Lets say that $P = \{w | w$ has an even number of 0s$\}$, $L_0 = aa*$ and $L_1 = \{w|$ last symbol of w is $a\}$. Consider the word $w = abaab$. Is it in L? To prove that w is in L we need to find some words from L_0 and L_1 such that when we concatenate them we get w and moreover the pattern that we use is in P. For example, we can take ab (from L_1), a (from L_0), a (from L_0), b (from L_1), the concatenation is w and 1001 is in P. Ot we can take a, b, aa, b, because 0101 is in P.

So given w, how can we find this partition of w into several words? We will try to guess the pattern, symbol by symbol. We begin in state q_p^0 of P. First we try to "guess" the first symbol in the pattern. Lets say it's 0. So currently we need to remember that we are trying to find the word from the 0-th language, we have moved to the state $\delta_p(q_p^0, 0$ and now we need to start simulating A_0 (from its initial state). After reading an a we are in the final state of A_0. What our choices are? Well, we can either continue

reading an input (a is in L_0, but so are aa, aaa and so on, so we must not stop here), or we can assume that this is an end of the current word. If this is an end of the word, then we need to guess the next symbol in the pattern, move to the corresponding state in P and start simulating the corresponding automator (A_0 or A_1). When do we need to accept? We must guess the right pattern, so we have to be in the final state of P. And moreover the current automaton (A_0 or A_1) must be in the final state, because we can not end in the middle of the word.

So formally, our NFA A will is the following. The state space of A is $Q = \{0, 1\} \times Q_p \times Q_0 \times Q_1$, a 4-tuple storing the index (0 or 1) of the language we are currently in and the current states in all of the 3 DFAs P, L_0 and L_1.

The δ-function is:

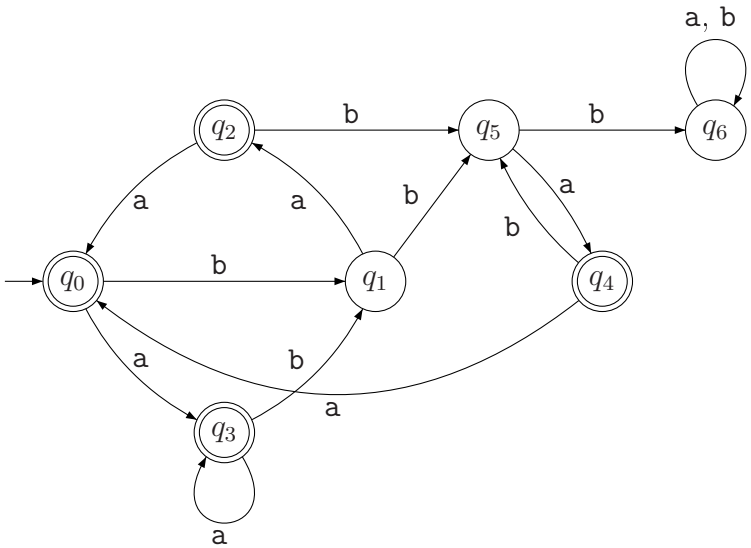
$$\delta(q_0', \epsilon) = \{(0, \delta_p(q_p^0, 0), q_0^0, q_1^0), (1, \delta_p(q_p^0, 1), q_0^0, q_1^0)\}$$

$$\delta((i, q_p, q_0, q_1), c) = \begin{cases} \{(0, q_p, \delta_0(q_0, c), q_1)\} & \text{if } i = 0 \text{ and } c \in \{a, b\} \\ \{(1, q_p, q_0, \delta_1(q_1, c))\} & \text{if } i = 1 \text{ and } c \in \{a, b\} \\ \{(0, \delta_p(q_p, 0), q_0^0, q_1^0), (1, \delta_p(q_p, 1), q_0^0, q_1^0)\} & \text{if } q_i \in F_i \text{ and } c = \epsilon \end{cases}$$

The final states are: $F = \{(i, q_p, q_0, q_1) \mid q_p \in F_p \text{ and } q_i \in F_i\}$

4. Minimization [**Category**: Construction., **Points**: 20]

Recall the minimization algorithm by partition refinment(Lecture Note #11). Use this algorithm to minimize the following DFA and draw the resulting minimal DFA. Show the partitions of the states at every iteration clearly.



Solution:

We start with only two sets: all final states and all non-final states. So

$$P_0 = \{\{q_0, q_2, q_3, q_4\}, \{q_1, q_5, q_6\}\}$$

We say $A = \{q_0, q_2, q_3, q_4\}$ and $B = \{q_1, q_5, q_6\}$. By reading a each state in A goes to A and by reading b goes to B. So we say that each state from A goes to (A, B). But for states in B, q_1, q_5 go to (A, B) and q_6 goes to (B, B). Hence q_6 should be in a different set, and we get a new partition of states which is a refinement of P_0:

$$P_1 = \{\{q_0, q_2, q_3, q_4\}, \{q_1, q_5\}, \{q_6\}\}$$

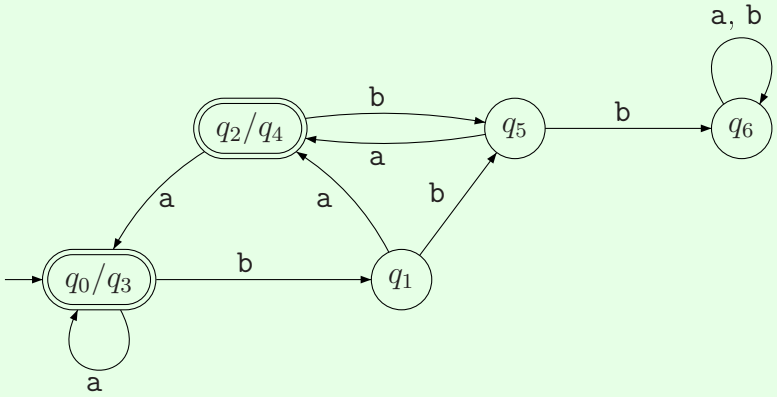
Similarly, examining the states in A again, all states still remain in the same set. But we can tell the difference between q_1 and q_5 by reading b. So the next partition is

$$P_2 = \{\{q_0, q_2, q_3, q_4\}, \{q_1\}, \{q_5\}, \{q_6\}\}$$

Now examine the states in A again, q_0 and q_3 remain in the same set, while q_2 and q_4 should ne in the other set. Then we form a new partition

$$P_3 = \{\{q_0, q_3\}, \{q_2, q_4\}, \{q_1\}, \{q_5\}, \{q_6\}\}$$

Now we are done since iterating to refine P_3 gives the same partition. So the partition P_3 gives the minimal automaton as follows:



5. Extra Credit/Honors: Clause Finite Automata [**Category**: Construction. Due only on Thu, Mar 18th, 2pm., **Points**: 20]

   A Clause Finite Automaton(CFA) is a generalization of an NFA (without ϵ transitions) such that the value of the transition function is no longer a set of states, but a combination of states by disjunctions/conjunctions. For example, if $\delta(q, \mathsf{a}) = q_1 \vee (q_2 \wedge q_3)$

in a CFA M, then at state q, reading $\mathtt{a}$, M nondeterministically choose to switch to q_1, or simulate the behavior of M from *both* q_2 and q_3 on the rest of the word. Intuitively, this transition says that M accepts the word aw from q if M accepts the word w from q_1, or, M accepts the word w from both q_2 and q_3.

Note that every NFA can be viewed as a CFA, since each transition $\delta(q, \mathtt{a}) = P$, where P is a subset of states, can be viewed as $\delta(q, \mathtt{a}) = \bigvee_{p \in P} p$.

Formally, a CFA is a tuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is a finite alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of final or accepting states, and $\delta : Q \times \Sigma \to B(Q)$ where $B(Q)$ is the set of all Boolean formulas over Q formed using conjunction and disjunction.

The notion of when M accepts a word is as follows:

- M accepts ϵ from state $q \in Q$ iff $q \in F$
- M accepts aw from state q (where $w \in \Sigma^*$ and $a \in \Sigma$) iff there is a set of states $Q' \subseteq Q$ such that the valuation that sets Q' to true and $Q \setminus Q'$ to false satisfies the Boolean formula $\delta(q, a)$, and M accepts w from each of the states $q' \in Q'$.

(a) Consider a CFA $M = (Q, \Sigma, \delta, q_0, F)$ where

$Q = \{q_0, q_1, q_2, q_3, q_4\}$;
$\Sigma = \{\mathtt{a}\}$;
δ is defined as follows:

δ	$\mathtt{a}$
q_0	$(q_1 \vee q_2) \wedge (q_3 \vee q_4)$
q_1	q_3
q_2	q_4
q_3	q_1
q_4	q_2

$F = \{q_1, q_3\}$.

Convert M to an equivalent NFA. (10 Points)

Solution:

Notice that each formula in the transition function of M can be converted to an equivalent Disjunctive Normal Form(DNF, "*or*s of *and*s"): $\delta(q_0, \mathtt{a})$ is equivalent to $(q_1 \wedge q_3) \vee (q_1 \wedge q_4) \vee (q_2 \wedge q_3) \vee (q_2 \wedge q_4)$, and other transitions are already in DNF. Moreover, let us extend the domain of δ to $\mathcal{P}(Q)$. For each $S \subseteq Q$ and each symbol x we define $\delta(S, X)$ to be $\delta(S, x) = \bigwedge_{p \in S} \delta(p, x)$, which can also be converted to a DNF.

Now consider an NFA N with each state being a subset of Q. Then for each state S, when reading a symbol x, N nondeterministically switches to a disjunct of $\delta(S, x)$. For example, $q_1 \wedge q_3$ is a disjunct of $\delta(\{q_0\}, \mathtt{a})$, thus N can choose to switch to $\{q_1, q_3\}$ when reading $\mathtt{a}$. Same for $q_1 \wedge q_4$, $(q_2 \wedge q_3)$ and $(q_2 \wedge q_4)$. Moreover, a

state S is final iff each element in S is final in M. So the equivalent NFA is defined formally as $N = \{Q', \Sigma, \delta', q_0, F'\}$, where

$Q = \{q_0, q_{13}, q_{14}, q_{23}, q_{24}, q_\varnothing\}$;

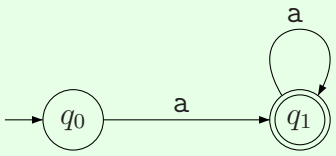
$\Sigma = \{\mathtt{a}\}$;

δ' is defined as follows:

δ	a	ϵ
q_0	$\{q_{13}, q_{14}, q_{23}, q_{24}\}$	$q_\varnothing$
q_{13}	$\{q_{24}\}$	$q_\varnothing$
q_{14}	$\{q_{23}\}$	$q_\varnothing$
q_{23}	$\{q_{14}\}$	$q_\varnothing$
q_{24}	$\{q_{13}\}$	$q_\varnothing$
$q_\varnothing$	$q_\varnothing$	$q_\varnothing$

$F = \{q_{13}\}$.

Which can be simplified to the following DFA:



(b) Given $\Sigma = \{0, 1, 2\}$ and $k > 0$, let $L_k = \{ww \mid |w| = k\}$. Construct a CFA for L_k with $O(k)$ states, for every k. (10 Points)

Solution:

A string w is accepted if $|w| = 2k$ and the i-th character matches the $k + i$-th character for each $1 \leq i \leq k$. The idea is to remember each i-th character and compare it with the $k + i$-th character, and to keep track of the length of the string in parallel. So we have states $q_0, \ldots, q_{2k}$ as the length counters, and a state q_{exceed} indicating the length exceeds $2k$ (in this case we do not accept). Moreover, for each character in Σ, say 0, there are states $q_{01}, \ldots, q_{0k}$ to skip the next $k - 1$ characters and then compare the next one with the one we have remembered, switching to q_{match} or $q_{non-match}$. At each $q_i (0 \leq i < k)$, N is about to read the $i+1$-th character, say 0, so it simulates q_{i+1} and q_{00} in parallel: q_{i+1} keep counting the length, while q_{00} after reading k symbols will end in an accepting state iff the $k + i + 1$-th character matches the $i + i$-th character.

More formally, L_k is accepted by the CFA $N = \{Q, \Sigma, \delta, q_0, F\}$, where

$Q = \{q_i \mid 0 \leq i \leq 2k\} \cup \{q_i j \mid 0 \leq i \leq 2, 1 \leq j \leq k\} \cup \{q_{exceed}, q_{match}, q_{non-match}\}$;

$\Sigma = \{0, 1, 2\}$;

δ is defined as follows:

δ	0	1	2
$q_i (0 \leq i < k)$	$q_{00} \wedge q_{i+1}$	$q_{10} \wedge q_{i+1}$	$q_{20} \wedge q_{i+1}$
$q_i (k \leq i < 2k)$	q_{i+1}	q_{i+1}	q_{i+1}
q_{2k}	q_{exceed}	q_{exceed}	q_{exceed}
q_{exceed}	q_{exceed}	q_{exceed}	q_{exceed}
$q_{ij} (1 \leq j < k)$	$q_{i,j+1}$	$q_{i,j+1}$	$q_{i,j+1}$
q_{0k}	q_{match}	$q_{non-match}$	$q_{non-match}$
q_{1k}	$q_{non-match}$	q_{match}	$q_{non-match}$
q_{2k}	$q_{non-match}$	$q_{non-match}$	q_{match}
q_{match}	q_{match}	q_{match}	q_{match}
$q_{non-match}$	$q_{non-match}$	$q_{non-match}$	$q_{non-match}$

$F = \{q_{2k}, q_{match}\}$.