More Fun With Automata
Homework 2, CS500, Fall 2014

Aaron Gonzales, Ahmad Darki, Manasa Navada
group 16

February 2, 2015

Ex 7

Show that if L is regular then L^* is regular. Why does it not suffice to use the fact that the regular languages are closed under concatenation and union?

Answer:

We know that $L* = \epsilon \cup L \cup LL \cup \ldots$, the enumerated set of all possible combinations of strings made up of the language L. We also know that for a language to be regular, there must be some DFA or NFA that can accept it.

If we let M_L be the machine that recognizes langague L, then let M_{L^*} be the machine that recognizes L^*. To construct M_{L^*}, add a new start state S' that is also an accepting state to M_L. Connect S' to M_L with an ϵ transition and add another ϵ edge out of M_L such that it connects to S'. This machine can accept any possible combinatins of valid instances of L and includes ϵ, showing that L^* is indeed regular.

Ex 8

Given a string w , let w^R denote w written in reverse. Given a language L, let $L^R = \{w^R | w \in L\}$. Prove that L is regular if and only if L^R is regular. Why is this harder to prove with DFAs?

Answer:

We know that a language is regular if there is a DFA that can accept it. This tells us that in order to prove W^R's regularity, there must be some DFA that can accept it. Generally put, we can map our current DFA to a DFA that can accept the new reversed language by modifying the start state, accepting states, and transition functions as follows, which will be formalized below.

Proof. Let $M = \{S, A, s^0, S^{yes}, \delta\}$ be the machine that recognizes L. Let $M' = S', A', S^{0'}, S^{yes'}, \delta*'$ be the NFA that recognizes L^R, constructed as below:

Let $S_{new} \in S$ be a new start state and connect it to all $s \in S^{yes}$ with an ϵ label. Make S^0 into an $S^{yes'}$. Make $s \in S^{yes} \to S'$ (make accepting states into normal states). Make $S^0 \to S^{yes'}$ (make the original intial state into the sole accepting state.

This new NFA will only accept langagues that begin in the first char of the w^R and end in the last char of w^r. As we have constructed an NFA that recognizes this L^R, we can deduce that that L^R is regular, meaning that $L \Leftrightarrow L^R$. □

The NFA epsilon ability allows us to more easily create new NFAs from exisisting automata.

Ex 9

A for-all NFA is one such that $L(M)$ is the set of strings where every computation path ends in an accepting state. Show how to simulate an for-all NFA with a DFA, and thus prove that a language is recognized by some for-all NFA if and only if it is regular.

Answer:

This is similar to converting any NFA to a DFA. let

$$A_n = \{Q, q_0 \in Q, F \subseteq Q, \delta : Q \times \Sigma \to P(Q)\}$$

be the $\forall$ NFA we wish to convert.

Let

$$A_d = \{Q' = \mathcal{P}(Q), \{q_0\}, \delta'(S, a) = \cup_{q \in S} \delta(q, a), F' = \{S \subseteq Q : S \cap F \neq \varnothing\}\}$$

be the DFA to which we are mapping A_n.

note that

$$Q' = \mathcal{P}(Q) \qquad Q' \text{ is power set of } Q$$
$$F' = \{S \subseteq Q : S \cap F \neq \}\} \qquad \text{DFA accepting states as subsets of Q with all elements accepting}$$

So all of the DFA's accepting states are on a computation path and that the All-paths NFA recognizes a regular language as we have build its DFA. [1]

Ex 10

A parity finite-state automaton, or *PFA* for short, is like an *NFA* except that it accepts a string w if and only if the number of accepting paths induced by reading w is odd. Show how to simulate a *PFA* with a *DFA*, and thus prove that a language is recognized by a *PFA* if and only if it is regular. Hint: this is a little trickier than our previous simulations, but the number of states of the *DFA* is the same.

Answer:

Assume an *NFA* being regular:

$$M_2 = (S_2, A, S_2^0, S_2^{yes}, \delta_2)$$

From this *NFA* we will have a *DFA* as:

$$M = (S, A, S^0, S^{yes}, \delta)$$

Where $S = 2^{S_2}$, $S^{yes} = S_2^{yes}$, $\delta(T, a) = \cup_{t \in T} \delta_2(t, a)$, and $S^{yes} = \{T \cap S_2^{yes}\}$ in which this *DFA* is regular as well. For this problem we need to prove that for an odd number of $\delta_2(S_2^0, w) = s_2^{yes} \in S_2^{yes}$ we will be able to make another *DFA*. From this we know that there will be an odd number of transitions of $\delta_2(S_2^0, w)$ which means $|w|$ is an odd number as in $w = w_1 w_2 ... w_(2n+1)$.

Ex 11

Given finite words u and v, say that a word w is an interweave of u and v if I can get w by peeling off symbols of u and v, taking the next symbol of u or the next symbol of v at each step, until both are empty. (Note that w must have length $|w| = |u| + |v|$.) For instance, if $u = cat$ and $v = tapir$, then one interleave of u and v is $w = ctaapitr$. Note that, in this case, we don't know which a in w came from u and which came from v. Now given two languages L_1 and L_2, let $L_1 \wr L_2$ be the set of all interweaves w of u and v, for all $u \in L_1$ and $v \in L_2$. Prove that if L_1 and L_2 are regular, then so is $L_1 \wr L_2$.

Answer:

We assume $L_1 \wr L_2$ is a language that can be described by the following:

$$M_{L_inter} = (S, A, S^0, S^{yes}, \delta)$$

and L_1 as:

$$M_{L_1} = (S_{L_1}, A, S_{L_1}^0, S_{L_1}^{yes}, \delta_{L_1})$$

and L_2 as:

$$M_{L_2} = (S_{L_2}, A, S_{L_2}^0, S_{L_2}^{yes}, \delta_{L_2})$$

Let's take $L' = L_1 \times L_2$ to be the *Cartesian product* of the two languages L_1 to L_2. Where the language will be as:

[1] teammates, My notation here may be slightly wonky

$$M_{L cartesian 12} = (S_{L_1} \times S_{L_2}, A, (S^0_{L_1}, S^0_{L_2}), (S^{yes}_{L_1} \cup S^{yes}_{L_2}), \delta_{L_1} \times \delta_{L_2})$$

and also the L'' will be the *Cartesian product* of L_2 to L_1":

$$M_{L cartesian 21} = (S_{L_2} \times S_{L_1}, A, (S^0_{L_2}, S^0_{L_1}), (S^{yes}_{L_1} \cup S^{yes}_{L_2}), \delta_{L_2} \times \delta_{L_1})$$

as we know that *Cartesian product* of two regular language is also a regular language. Also the union of two regular languages will be a regular language itself. The idea is to take the interweave as a result of union over the two *Cartesian product*.

I can not prove what I'm trying to say here....!!!

Ex 12

Given a language L, let $L_{1/2}$ denote the set of words that can appear as first halves of words in L:

$$L_{1/2} = \{x | \exists y : |x| = |y| \text{ and } xy \in L\}$$

where $|w|$ denotes the length of a word w . Prove that if L is regular, then $L_{1/2}$ is regular. Generalize this to $L_{1/3}$, the set of words that can appear as middle thirds of words in L:

$$L_{1/3} = \{y | \exists x, z : |x| = |y| = |z| \text{ and } xyz \in L\}$$

Answer A:

We know that $|x| = |y|$ and must show that the set of words in $L_{1/2}$ can be represented by a DFA to be proved regular. FA's limit us in that we cannot go back in time or record with external memory, and we may not know how large a word w is before it is read. As such, we must (A) build a FA that allows us to give $|x| = |y|$ and allows us to verify that (B) $xy \in L$. This is complicated but not impossible.

Chris Moore's hint about River Song and the Doctor is a reference to Dr. Who, in which the characters are moving toward each other in time, one going forward and the other going backward. Given that hint, let us construct a FA that allows us to accomplish A and B, mostly given by the product construction of two FAs.

Proof. Let us define two FAs: [2]

$$M = \{Q, \Sigma, \delta, S_0, S^{yes}\} \tag{1}$$
$$M^R = \{Q^R, \Sigma, \delta^R, S^{yes^R}\} \tag{2}$$
$$Q^R = Q \cup \{S^R_0\} \tag{3}$$

M is the DFA which recognizes L and M^R is the NFA that recognizes L^R. We have seen that the reverse operator is closed under regularity. Q^R is the union of states of M and acepting states of L^R. Now introduce $M_{1/2}$, the machine that recognizes $L_{1/2}$, which is the product of $M \times M^R$ and we define all of $M_{1/2}$ member's below.

$$M_{1/2} = M \times M^R \tag{4}$$
$$= \{Q_{1/2}, \Sigma, \delta_{1/2}, S_{o_{1/2}}, S^{yes}_{1/2}\} \tag{5}$$
$$Q_{1/2} = Q \times Q^R \qquad \text{States} \tag{6}$$
$$\delta_{1/2} = \{(q_1, q_2), a\} = \cup_{c \in \Sigma} \{\delta(q_1, a), \delta^R(q_2, c)\} \qquad \text{transition function} \tag{7}$$
$$S_{0_{1/2}} = \{S_0, S^R_0\} \qquad \text{initial states} \tag{8}$$
$$S^{yes}_{1/2} = \{(q1, q) | q \in Q\} \qquad \text{accepting states} \tag{9}$$

[2]teammates, My notation here may be slightly wonky (the NFA/DFA notation needs cleaning and I will gladly put in a figure to help explain this in better detail regarding the overlap of the boundary states for the words on both parts a and b

Our new machine, $M_{1/2}$ will read the string $w \in L$ and only accept it if the strings x, y are equal in length and are a part of this new $L_{1/2}$ language. □

(a) Answer B

Proving this for $L_{1/3}$ would involve repeating the process from above and getting a triple product construction for the three FA for which we have interest, though the construction of this is laborious. A possible proof by induction for an $L_{1/n}$ for which any word in the language can be chopped into n discreet and equal chunks could follow.

Proof. Via induction over the size of a word

$$|x_1|, |x_2|, \dots |x_n|| \sum_{i=1}^{n} |x_i| = w \in L$$

, where L is regular and $|w|\%n = 0$.

□

Ex 13

Show that if $u \sim_L v$, then $ua \sim_L va$ for any $a \in A$.

Answer:

Proof. L-equivalence is defined by Definition 3 [3] as given some language $L \subseteq A^*$, $u, v \in A^*$ are $u \sim_L v$ if $\forall w \in A^*$ $uw \in L$ iff $vw \in L$. Since this is a forall notion, we can say that this would include words $w = ax$ or a word comprised with a prefix a on x. $\forall x, uax \in L$ iff $vax \in L \implies ua \sim_L va$. □

Ex 14

Describe the equivalence classes of the three languages from Exercise 2. Use them to give the minimal DFA for each language, or prove that the DFA you designed before is minimal.

Answer:

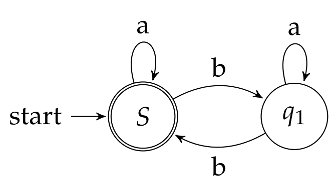

Figure 1: DFA for Exercise $2(a)$, the set of words in $\{a, b\}*$ with an even number of b's.

We know that there are only two equivalence classes, thanks to this DFA being in its minimal state.

$$\{z | \forall w \in L \Leftrightarrow zw \in L\}$$

Note that the above describes the class with z having even b's and w having even b's.

we have a class $\{bb\}$, the class where we have an even number of b's and a class $\{b\}$ in which we have an odd number of b's. All other combinations in the language do not matter.

[3] Automata notes

Ex 20

Consider the language

$$L_{a=b,c=d} = \{w \in \{a,b,c,d\}^* \,|\#_a(w) = \#_b(w) \text{ and } \#_c(w) = \#_d(w)$$

What are its equivalence classes? What does its minimal infinite-state machine look like?

Answer:

Clearly, we can see that we would require an infinite number of states to count the number of each letter in A. Formally, we can see the equivalence classes of $L_{a=b,c=d}$ as such:

Consider the set of words

$$\{a^i | i \geq 0\} = \{\epsilon, a, aa, aaa, \dots\}$$

. If $i \neq j \implies a^i \not\sim a^j$, as

$$a^i b^i \in L_{a=b}, \text{ but } a^j b^i \notin L_{a=b}$$

We need a similar set of words for c:

$$\{c^n | n \geq 0\} = \{\epsilon, c, cc, ccc, \dots\}$$

. If $i \neq j \implies c^i \not\sim c^j$, as

$$c^n d^n \in L_{c=d}, \text{ but } c^m d^n \notin L_{c=d}$$

Each possible i, n gives us an equivalence class. Note that this logic followed directly from the automata notes example for $L_{a=b}$.