More Fun With Automata
Homework 2, CS500, Fall 2014

Aaron Gonzales (group 16), Ahmad Darki (group 11), Manasa Navada (group 30)

February 3, 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.

Since we are adding an ϵ transition, concatenation and union do not cover ϵ closure and we need to account for that.

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

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

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.

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:

Same as the previous question, assume an NFA:

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

And the PFA:

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

in which $Q' = \mathcal{P}(Q)$ is the power set of Q, and $F' = \{S \subseteq Q : S \cap F \neq \varnothing\}\}$ is the DFA accepting states as subsets of Q with all elements accepting.

We will consider a new accepting state called f where $f \subseteq F'$ and all the accepting states will have an ϵ transition to it. This state will assist us by considering only one final state in order to audit the number of paths. The idea of getting the number of paths is to do a symmetric difference between the sets of all visited states. This will help us eliminate the even number of paths which visited the f state. At the end we will do a summation over the cardinality of each set, and we will consider the remainder of this value divided by 2 to show if the number of paths is even or odd.

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:

The machine for L_1 can be defined as:

$$M_1 = \{S_1, A, S_1^0, S_1^{yes}, \delta_1\}$$

and the machine for L_2 as:

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

The product construction of the two languages allows us to run them in parallel. Let $L' = L_1 \times L_2$ to be the *Cartesian product* of the two languages L_1 to L_2, with its machine defined as:

$$M' = \{S' = \{S_1 \times S_2\}, A, S'^0 = \{S_1^0 \cup S_2^0\}, S'^{yes} = \{S_1^{yes} \cup S_2^{yes}\}, \delta' = \{\delta_1 \times \delta_2\}\}$$

Now, in order to handle the potential for various combinations of valid interweaves, we need to use the power set of L', which would result in this final machine, M, that recognizes $L_1 \wr L_2$.

$$\begin{aligned} M =& (S, A, S_0, S^{yes}, \delta) \\ S =& \mathcal{P}(S') \\ S_0 =& S_{L_1}^0 \cup S_{L_2}^0 \\ S^{yes} =& s \in S'^{yes} \\ \delta =& \{\{\delta_1, S_n, a \in A\}, \{\delta_2, S_n, a \in A\}\} \end{aligned}$$

This allows us to accept words that are valid in and order of M_1 and M_2, such as the word “cat” $\in L_1$ and “cat” $\in L_2$ allows us to accept the first word in any valid accpeting state of M_1 and continue weaving it with the rest of the possible states of M_2, e.g., “cacatt”.

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:

$$\begin{aligned} M =& \{Q, \Sigma, \delta, S_0, S^{yes}\} && (1) \\ M^R =& \{Q^R, \Sigma, \delta^R, S^{yes^R}\} && (2) \\ Q^R =& Q \cup \{S_0^R\} && (3) \end{aligned}$$

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.

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

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

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

, 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 [1] 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.

t

Part 1

Answer:

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.

[1] Automata notes

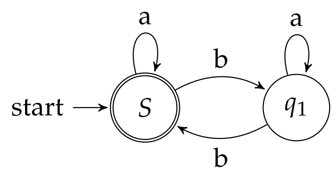


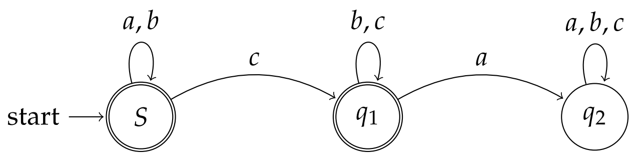
Figure 1: DFA for Exercise $2(a)$, the set of words in $\{a, b\}*$ with an even number of b's.

Part 2

> **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.**

Exercise 2.2: The set of strings in $\{a, b, c\}^*$ where there is no c anywhere to the left of an a.

As it has been proven that the following DFA is a minimal, with respect to the number of states the *equivalence class* are 3 as followed:



1. $[\epsilon] = [a] = [ab] = [aba]$, The set of strings in w without any c.
2. $[c] = [cb] = [cbc]$, The set of strings in w starting with c.
3. $[c] = [cb] = [cbc] = [cbcb] = [cbcba]$, The set of strings in w where there is c to the left of a

Part 3

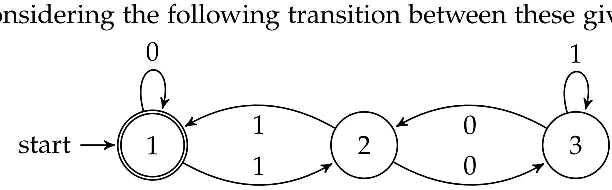
> **The set of strings in $\{0, 1\}^*$ that encode, in binary, an integer w that is a multiple of 3. Interpret the empty string ϵ as the number zero.**

Answer:

A minimal DFA for this language has three states. Each state represents the reminder of the number when it is divided by 3. The equivalence classes for this language are:

1. w containing the reminder 0. (Also the start state and accepting state)
2. w containing the reminder 1.
3. w containing the reminder 2.

Considering the following transition between these gives the following DFA.



Hence it is proved that the DFA we designed in HW1 is minimal.

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

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

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

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

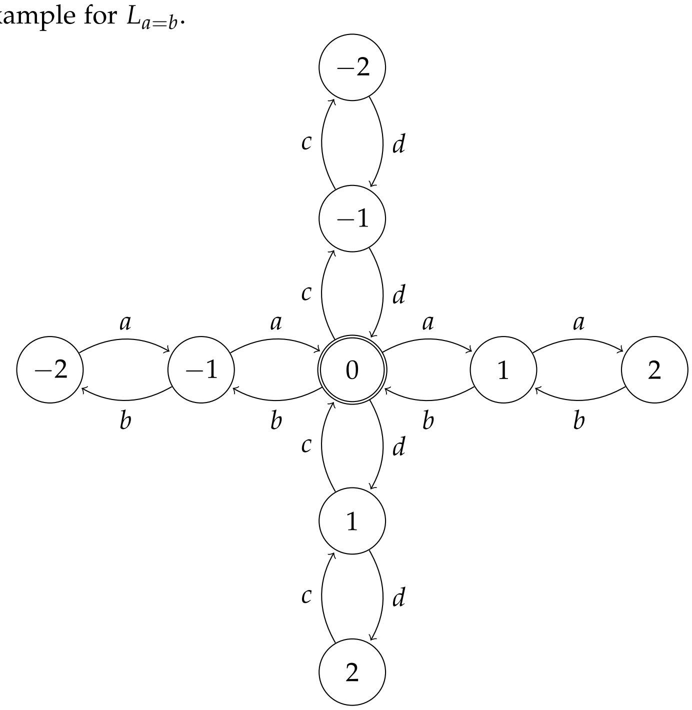
We need a similar set of words for c:

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

. 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 $L_{a=b}$.



Combined Group Critiques

These are sythesized critiques for each member's problems that were turned in by the initial due date.

(b) Aaron Gonzales

- Ex 7: Forgot to explain 2nd part of the question; otherwise correct.
- Ex: 8: Mentioned lack of both sides of "iff".(This was shown succicntly.) Didn't explicitly mention difficulty of using DFAs.
- 9: complete and no critiques
- 12: Good approach; lacking slightly in clarity
- 13: No comments
- 14: Correct
- 20: Correct

(c) Ahmad Darki

- Ex. 7: Explanation is correct but somewhat unclear.
- Ex. 8: Difficult to follow the explaination.
- Ex. 11: not attempted
- Ex. 10: Incomplete and incorrect solution.
- Ex. 13,14: Not attempted by intial date; added in (correctly) after
- Ex. 20: Correct

(d) Manasa Navada

- Ex7: Regular expressions were not appropriate to use at that point in the assignment, though it was a correct answer.
- Ex8: difficult to follow explaination for part two and lack of proving both directions for the "iff".
- Ex 9: No issues.
- Ex 12: Incorrect approach as the automata cannot keep track of a previous state. Difficult to follow.
- Ex 13: Correct; no issues.
- Ex 14: Correct; no issues.
- Ex 20: Poor explaination and lack of evidence as to why there are infinite equivalence classes.

Group contributions

(e) Aaron Gonzales

- Used explainations from his problems, typeset and compiled final report, extra work done on odd problems that were not initially completed or attempted.

(f) Ahmad Darki

- worked on remaining odd problems, contributed to final report, used some of his solutions in writeup.

(g) Manasa Navada

- Critiqued group member's solutions; used parts of a few answers.