CS 3100 – Models of Computation – Fall 2010
Notes 8, Posted online: September 16, 2010

These problems will be helpful for Midterm-1. More solutions will be worked out. The midterm exam itself won't be this long! Many of the actual questions will be a subset of these questions with small variations.

Midterm-1 will be open book and notes. However, please do not depend on this material (except for quick reassuring peeks). It will not help shuffle through books/notes during the midterm.

Short Questions

Please say true or false, giving reasons in at least two good sentences

1. Alphabets can contain ε.

   False; alphabets never include ε.

2. All alphabets are non-empty.

   True. Alphabets must be non-empty.

3. The alphabet of a DFA can be infinite.

   False. DFAs by definition have a finite alphabet.

4. The alphabet of an NFA can contain ε.

   False. The alphabet of an NFA cannot contain ε (same for a DFA).

5. There are only a finite number of languages over a given alphabet.

   False. One can have infinitely many languages over any alphabet.

6. $\{\}^*$ and $\{\varepsilon\}^*$ are different languages.

   False. They are the same language.

7. $\{\}^*$ and $\{\varepsilon\}$ are different languages.

   False. They are the same language.

8. NFAs can always be complemented by exchanging final (accepting) and non-final (rejecting) states.

   False. Consider an NFA N over alphabet $\{a\}$ with one initial state I going to two different states A (accepting) and B (rejecting), with both transitions labeled by a. A and B have no outgoing transitions. This NFA accepts a, but the erroneous complementation procedure gives an NFA with a in its language as well.

9. All DFAs can be redrawn to have exactly one final state, resulting in a language-equivalent DFA.

   This is false. Consider the minimal DFA for the language $\{a, aa\}$ over $\{a\}^*$. Such a DFA must have two distinct final states.

10. NFA intersection can be performed using the same algorithm as used for DFA intersection (marching the machines together, labeling a next state as accepting if both next states are accepting), and the result will be a single NFA for the intersection of the languages.

    For NFAs without ε jumps, this works. We must adapt this algorithm for NFAs with ε jumps. The question is how to treat ε jumps in one NFA versus non-ε jumps in another. If we can do the ε-closure after every jump and pull in those states also, then it can be made to work. So the answer is "no, not without doing extra work concerning ε."

11. Every DFA is also an NFA in every respect (including how their next-state functions are defined).

    False. The next-state function of an NFA specifies sets of states as the target of every jump.

12. Given a DFA of M states and another of N states, there is a straightforward algorithm to obtain a DFA of no more than $M \times N$ states that represents the union of these two DFAs.

    True. The standard algorithm.

Please give short answers in one to two (sometimes three) sentences

1. Please give an example of an NFA N such that it cannot be starred without adding an extra initial state. I am referring to the standard NFA starring procedure here.

    Consider an NFA

    ```
    Initial state = I
    Final state = F
    Transitions:

    I -a-> F
    I -b-> I
    ```

    This NFA accepts strings ending in a with any number of bs before it. However a string of just bs is not accepted. If we star this NFA by turning the state I into a final state and putting an ε jump from F to I, we will accept strings of just bs, which cannot be in the star of the language of this NFA.

2. The reverse of a language is defined to be a set containing all the original strings, except each string appears reversed. What is in the reverse of $S = \{\varepsilon, a, aa, aaa, aab, aba\}$ that is not in S?

    baa is not present in the reverse of S.

3. There are languages L such that they are identical to their reverse. What are these languages called?

    Palindromic languages.

4. What is the alphabet of a GNFA (generalized NFA) that you obtain during DFA to RE conversion?

    Regular expressions.

5. Describe a way to obtain an NFA n_2 from a given NFA n_1 such that the language of n_2 is the reverse of the language of n_1. Hint: you can retain all the states of n_1 and do something simple with the edges, and also add one extra state.

    (i) Introduce a new initial state. (ii) Turn all final states into non-final states, introducing a jump from this initial state to the former final states through ε. (iii) Turn the initial state into a final state (it may have temporarily become non-final in step (ii). (iv) Reverse all transitions. This construction yields the reverse of a language because any string from the initial state to a final state can now be traced backwards from the new initial state to the original initial state.

6. Does this NFA reversal algorithm work for DFAs too (in the sense of yielding a DFA which represents the reverse of the language of the original DFA)? Why or why not?

   Consider two transitions in this DFA that are labeled by $a \in \Sigma$ entering one state s. After the steps described above, we will have s exiting through a to two possibly different states—*i.e.*,, we end up introducing non-determinism. So we may not obtain a DFA as a result; but we can obtain an NFA that describes the reverse of the language of the original DFA.

7. Are regular languages closed under reversal? Explain of course, relating to questions you have already answered.

   Yes, they are, because we can reverse NFAs, obtaining NFAs that represent the reversed language.

8. Is the language of palindromes over a singleton alphabet (alphabet with exactly one symbol) regular?

   Yes because any language that includes *all* palindromes over a singleton alphabet $\Sigma = \{a\}$ is nothing but a^*.

Long Answer Questions

1. Draw a DFA for the set of binary strings that are fed MSB-first and are evenly divisible by seven. Show the transitions out of states 3 and 4 fully. There is no need to show the remaining transitions.

   The state transition equations are as follows.

   $N\%7 \xrightarrow{0} (2.(N\%7))\%7$

   $N\%7 \xrightarrow{1} ((2.(N\%7))\%7 + 1)\%7$

   Thus, out of state 3, we will have

   $3 \xrightarrow{0} 6$

   $3 \xrightarrow{1} 0$

   Work out the moves out of 4 similarly.

2. Show using the Pumping lemma that the language of Palindromes L_{pal} over the alphabet $\{0, 1\}$ is not regular.

   Consider a DFA of k states being able to recognize this language. Consider the string 0^k10^k which is a palindrome. By "pumping up" within the first 0^k strings, we can obtain a string that is not in L_{pal}. (By pumping up, we are introducing extra 0s before the 1.)

3. Write down a regular expression for the language L_{epis1}—namely all strings over $\{0, 1\}$ such that every even position is a 1.

   $((0 + 1)1)^*(0 + 1 + \varepsilon)$

4. How would you change this for all strings over $\{0, 1\}$ such that every even position is a 1 for all strings that begin with a 1, and for strings that begin with a 0, there are no constraints? Argue why your answer is correct.

   $1(1(0 + 1))^*(1 + \varepsilon) \quad + \quad 0(0 + 1)^*$

   We provided for the constraints in the first part of the RE by insisting on there being a 1 in the right place for all strings—of odd or even lengths. The "start with 0" was handled simply.

5. Describe the complement of $L_{allEpsAre1s}$ in English. Write down four of the shortest strings in this language (in case of a tie for lengths, choose any string of the same length group). Call this language $L_{oneEpIs0}$.

   The complement will have one even position that is a 0. As discussed in class, it consists of strings like

   00, 10, 000, and 100.

6. Write down a regular expression for $L_{oneEpIs0}$.

   The first "broken" even position is after a few good even positions, if any.

   This is what the following RE says.

   Once broken, we don't care what happens:

   $((0+1)1)^*\ ((0+1)0)\ (0+1)^*$

7. Obtain a DFA $D_{allEpsAre1s}$ for $L_{allEpsAre1s}$. Use any method (short-cuts are OK so long as you are sure about your work).

   ANSWER.

   This will be a simple DFA.

   Let us denote initial states by Ixx (xx means anything; e.g. I23 etc)

   Initial and final states by IFxx

   Final states by Fxx

   Non-final states by any other name.

   Then we have the following DFA:

```
IF --0,1--> F0

F0 --1--> IF

F0 --0--> A

A  --0,1--> A
```

8. Obtain a DFA $D_{oneEpIs0}$ for $L_{oneEpIs0}$. Use any method (short-cuts are OK so long as you are sure about your work).

   ANSWER.

```
I --0,1--> B

B --1--> I

B --0--> F1

F1  --0,1--> F1
```

9. Show how to systematically arrive at $D_{allEpsAre1s} \cap \overline{D_{oneEpIs0}}$. Here, the "overline" means *complement*. *Before you begin your work,* what answer do you expect to obtain? Now finish your work and please explain whether you obtained the expected answer?

   We do the product construction, obtaining back

   $D_{allEpsAre1s}$

10. Show how to systematically arrive at $D_{oneEpIs0} \cap \overline{D_{allEpsAre1s}}$. Here, the "overline" means *complement*. *Before you begin your work,* what answer do you expect to obtain? Now finish your work and please explain whether you obtained the expected answer?

    We do the product construction, obtaining back

    $D_{oneEpIs0}$

11. Obtain an NFA N_{errc} that corrects all one-bit errors with respect to the language $(0+1)^*0101(0+1)^*$. Directly draw this NFA without any intermediate steps. This NFA is not the NFA for $RE_{orig} = (0+1)^*0101(0+1)^*$, but all strings that are different in *up to one position* with respect to R_{orig}. That is, error-free strings plus strings corresponding to one-bit errors are both included.

    We discussed this in class.

12. Obtain a regular expression RE_{errc} that captures the correction of all one-bit errors with respect to RE_{orig}. That is, this is an RE for all strings that are different in *up to one position* with respect to R_{orig}. Directly write this RE.

13. Show that N_{errc} and RE_{errc} have the same minimal DFA.

14. Review RE to NFA, NFA to DFA, and DFA to RE algorithms. Know how to apply them for simple problems.

15. Review the DFA minimization procedure. Know how to apply it for simple problems.

    Start with all final states in one equivalence class and non-final ones in another equivalence class. These are already distinguished states. Move the equivalence classes through each symbol. If the target states under the same symbol are distinguished, split the equivalence classes.

    We will work out a simple problem.

16. Is the language

$$L_{sqr} = \{a^{n^2} \mid n \geq 0\}$$

    regular? This language has strings of as of lengths that are perfect squares.

    No. There are two ways to prove it.

    (a) *Using the Pumping Lemma:* If we assume that there is a k-state DFA for L_{sqr}, pick any string z in L_{sqr} of length greater than k. This means that we should be able to split y into uvw with $\mid uv \mid \leq k$ (length of uv is $\leq k$) and $\mid v \mid > 0$. Furthermore, we should be able to generate strings in L_{sqr} by iterating v. That is, for all $i \geq 0$, $uv^iw \in L_{sqr}$. This is impossible for a language that only has perfect squares as the lengths of its strings. Perfect squares cannot be generated by adding successive $\mid v \mid$ lengths. In other words, uv^iw describe a family of lengths that lie on a straight-line slope which cannot be on a square curve.

(b) *Distinguishable Pairs of Strings Approach:* The creative part is to find sets of pairs of distinguishable strings. Let L_{sqr} itself be the set. Again, knowing how squares are, we can assert this: Suppose we take two successive strings s_1 and s_2 of lengths m and n in L_{sqr} (that is, $m = p^2$ and $n = (p+1)^2$ for some natural number p). Now consider a string z of length $(n - m)$. We know that $a^{m+z} \in L_{sqr}$. We can prove that $a^(r + z)$ is not in L_{sqr} for all r. That is, if we have a set of perfect squares, and if we take the difference between two adjacent perfect squares m and n, we cannot generate any perfect square by adding this difference to m. For example, in the set $\{1, 4, 9, 16, 25, 36, 49, 64, \ldots\}$, $36 - 25 = 11$, but adding 11 to any of the numbers except to 25 cannot yield anything in this set.

17. Suppose language A is unknown to be regular or not. Suppose B is known regular. Suppose we obtain $AandB = A \cap B$ and find $AandB$ to be non-regular. Now what can we say about A (regular or not)? (Hint: Consider A to be the set of strings over $\{0, 1\}$ with an equal number of 0s and 1s, and let B be 0^*1^*.)

    We can say that A is non-regular. If it were regular, then $AandB$ must be regular. It is not. For the given examples, $AandB$ is $\{0^n1^n \mid n \geq 0\}$.

18. Suppose language A is unknown to be regular or not. Suppose B is known regular. Suppose we obtain $AandB = A \cap B$ and find $AandB$ to be regular. Now what can we say about A (regular or not)? (Hint: Consider A to be $L0^n1^n$ and B be $\emptyset$.)

    We cannot say anything about A. Whether it is regular or not, the intersection of A yields the empty set (empty language). Empty languages are regular because they can be accepted by an NFA with a single start state with no transitions, and with no final state.

19. Show using the Pumping Lemma that

$$L_{a^m b^n} = \{a^m\, b^n \mid m > n\}$$

    is non-regular. You are required to use these facts/methods:

    (a) That $L_{a^m b^n} = \{a^m\, b^n \mid m < n\}$ is non-regular.

    (b) Regular languages are closed under reversal.

    If $L_{a^m b^n}$ is regular, so is its reverse. Take the reverse of $L_{a^m b^n}$. It is a similar language except $m < n$. Let it be regular with a k-state DFA. Now consider a string $a^k b^r$ for $r > k$ in the reversed language. We can pump the a^k portion to come outside of this language. Hence the reversed language, and hence $L_{a^m b^n}$ itself are non-regular.