Sample Solution for TCS Exam — Summer 2016

July 31, 2017

Problem One: Automata and Languages

Consider the following language $L = \{w | w$ starts with arbitrary number (including zero) of a, followed by arbitrary number (including zero) of b, and then ends with at least one $a\}$. The alphabet for L is $\{a, b\}$. Answer the following questions:

(a) Give a regular expression that generates L.

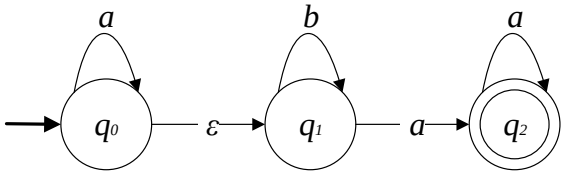
(b) Construct a *three state* NFA that recognizes L.

(c) Construct a DFA which recognizes L. (For example, you can covert the NFA from (b), if you believe it is correct.)

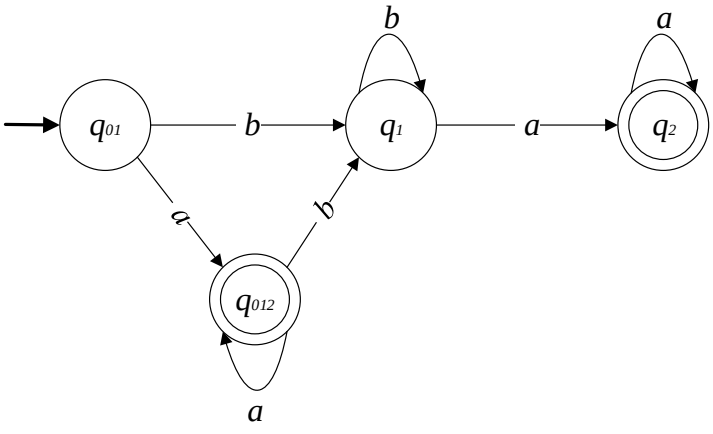
Sample Solution

(a) The regular expression is $a^*b^*a^+$.

(b) One possible NFA is shown in the following figure.



(c) One possible DFA (which is converted from the above NFA) is shown in the following figure. (Notice, some transitions are omitted.)



Problem Two: Automata and Languages

Let the alphabet $\Sigma = \{0, 1\}$. For each of the following language, answer if it is a CFL or not. You need to prove your answer as well.

(a) $L_1 = L^*$, where L is a CFL.

(b) $L_2 = \{w | w = w^{\mathcal{R}}$, which means w is a palindrome$\}$.

(c) $L_3 = \{w | w = w^{\mathcal{R}}$, and w contains equal number of 0s and 1s$\}$.

Sample Solution

(a) L_1 is a CFL. Since L is a CFL, w.l.o.g., assume $\mathcal{G}$ is a CFG that describes L. Assume S is the start symbol of $\mathcal{G}$. Consider a new CFG $\mathcal{G}'$ in which S' is the new start symbol, and contains all the rules in $\mathcal{G}$. We also add the following rule to $\mathcal{G}'$: $S' \to \epsilon | SS'$. It is easy to see that $\mathcal{G}'$ is a CFG which describes L_1. Hence, L_1 is a CFL.

(b) L_2 is a CFL. To show this, we give the following CFG which describes L_2.

$$S \to \epsilon | 0 | 1 | 0S0 | 1S1$$

(c) L_3 is not a CFL. We use pumping lemma to prove our claim. Consider string $0^p 1^{2p} 0^{2p} 1^{2p} 0^p$, where p is the pumping length. Split string into $uvxyz$ as in the pumping lemma. If vxy contains only 0 from the first 0^p or the last 0^p, then by pumping up, the string will no longer be a palindrome. Similarly, if vxy contains only 1 from the first 1^{2p} or the last 1^{2p}, then by pumping up, the string will no longer be a palindrome. Moreover, if vxy contains only 0 from 0^{2p} in the middle, then by pumping up, the string will contain more 0 than 1. Finally, if vxy contains both 0 and 1, similar contradictions can be obtained as well. Hence, L_3 is not a CFL.

Problem Three: Computability

Prove the following languages are decidable.

(a) $L_1 = \overline{L}$, where L is a decidable language. (That is, L_1 is the complement of L.)

(b) $L_2 = \{\langle R \rangle | R$ is a regular expression describing a language containing at least one string w that has 00 as a substring$\}$.

Sample Solution

(a) Assume M is a machine that recognizes L. Consider another machine M', which, upon input i, simulates M. Moreover, if M accepts i, then M' rejects it; otherwise, if M rejects i, then M' accepts it. It is easy to see that M' will always give the correct answer. Moreover, since L is decidable, M' will give definite answer in finite time. Hence, L_1 is decidable as well.

(b) Let M_1 be a DFA that accepts strings that contain 00 as a substring (w.r.t. the corresponding alphabet). Notice, it is always possible to construct such a DFA. Consider the following machine M. Upon input R, M constructs a DFA M_2 which accepts the regular language described by R. Let M_3 be $M_1 \cap M_2$, which is also a DFA. Then, M tests whether M_3 accepts anything at all. If the answer is yes, then M accepts the input R, otherwise it rejects the input. It is easy to see the correctness of M. Moreover, each step of M takes a finite time. Hence, L_2 is a decidable language.

Problem Four: Big-O Notation

Determine the relationship between the functions $f(n)$ and $g(n)$ for the following three examples. That is, state whether it holds that $f(n) = O(g(n))$, $g(n) = O(f(n))$, or if both these statements are true (which would imply $f(n) = \Theta(g(n))$).

(a) $f(n) = n^{100}$, and $g(n) = 2^n$

(b) $f(n) = \log_{10} n$, and $g(n) = 100 \log_2 n$

(c) $f(n) = (2n)!$, and $g(n) = n^n$

Sample Solution

(a) Exponential grows faster than polynomial, hence we know $f(n) = O(g(n))$. (In fact, $f(n) = o(g(n))$.)

(b) Notice $f(n) = \log_{10} n = (1/\log_2 10) \cdot \log_2 n$. Therefore, it is easy to see $f(n) = \Theta(g(n))$.

(c) Notice $f(n) = (2n)! > n^n \cdot n!$. Therefore, we know $f(n) = \Omega(g(n))$. (In fact, $f(n) = \omega(g(n))$.)

Problem Five: Complexity

(a) An instance of SET-COVER is given by a set U, a collection of m sets $S_1, S_2, \cdots, S_m$ such that each set S_i is a subset of U, and a positive integer k. The question is, can you find a collection $\mathcal{C}$ containing at most k of these subsets such that taken together, they "cover" all elements in U? In other words, is there a set $\mathcal{C}$ such that $|\mathcal{C}| \leq k$ and $\bigcup_{S_i \in \mathcal{C}} S_i = U$? Prove that SET-COVER is NP-complete.

(b) Let T-SAT $= \{\langle\phi\rangle | \phi$ has at least three satisfying assignments$\}$. Show that T-SAT is NP-complete.

Sample Solution

(a) To show SET-COVER is NP-complete, we first show it is NP. This is easy to prove, as a non-deterministic Turning machine can non-deterministically guess k sets and verify if the union of them is U. In particular, the machine can create a branch for each such guess, and accept the input if at least one branch is accepted. Since verifying a guess is easy, the machine can terminate in polynomial time. Hence, SET-COVER is NP.

We then prove SET-COVER is NP-hard—which, together with SET-COVER is NP, would imply SET-COVER is NP-complete—by showing a reduction from VERTEX-COVER. I.e., given an instance of VERTEX-COVER, we convert it to an instance of SET-COVER. More specifically, consider an instance of VERTEX-COVER in which we have a graph $G = (V, E)$ and an integer k'. We construct an instance of SET-COVER in the following way. Let $U = E$. For each node $v \in V$, construct S_v to contain all edges that are indecent to v. Finally, let $k = k'$.

Now, given an instance of VERTEX-COVER $\langle G, k'\rangle$. If $\langle G, k'\rangle$ belongs to VERTEX-COVER, then it is easy to see that $\langle U, \{S_1, S_2, \cdots, S_m\}, k\rangle$ belongs to SET-COVER. Similarly, if $\langle G, k'\rangle$ does not belong to VERTEX-COVER, then it is easy to see that $\langle U, \{S_1, S_2, \cdots, S_m\}, k\rangle$ does not belong to SET-COVER as well.

(b) To show T-SAT is NP-complete, we first show it is NP. This is easy to prove, as a non-deterministic Turning machine can non-deterministically guess three assignments and verify if all of them are satisfying assignments. In particular, the machine can create a branch for each such guess, and accept the input if at least one branch is accepted. Since verifying a guess is easy, the machine can terminate in polynomial time. Hence, T-SAT is NP.

We then prove T-SAT is NP-hard—which, together with T-SAT is NP, would imply T-SAT is NP-complete—by showing a reduction from SAT. I.e., given an instance of SAT, we convert it to an instance of T-SAT. Assume ϕ is a formula. Assume x and y are two new variables that never appear in ϕ. Now, consider formula $\phi \wedge (x \vee \overline{x}) \wedge (y \vee \overline{y})$. If ϕ belong to SAT, then it is easy to see $\phi \wedge (x \vee \overline{x}) \wedge (y \vee \overline{y})$ belongs to T-SAT. (In fact, in such case, $\phi \wedge (x \vee \overline{x}) \wedge (y \vee \overline{y})$ has at least four satisfying assignments.) On the other hand, if ϕ does not belong to SAT, then it is easy to see $\phi \wedge (x \vee \overline{x}) \wedge (y \vee \overline{y})$ does not belong to T-SAT as well. (In such case, $\phi \wedge (x \vee \overline{x}) \wedge (y \vee \overline{y})$ has no satisfying assignments at all.)

Problem Six: Logic

(a) Given knowledge base $KB = \{\neg p \vee q \rightarrow r, s \vee \neg q, \neg t, p \rightarrow t, \neg p \wedge r \rightarrow \neg s\}$, derive $\neg q$ from KB.

(b) A *predicate* is a sentence that contains a finite number of variables and becomes a statement when specific values are substituted for the variables. (For example, "$x^2 > x$" is a predicate.) The domain of a predicate variable is the set of all values that may be substituted in place of the variable.

Let $P(x)$ and $Q(x)$ be predicates and suppose D is the domain of x. For each of the following pair of statements, determine whether the two statements are equivalent. If the two statements are equivalent, no explanation is needed, if they are not equivalent, you need to give a counterexample.

- $\forall x \in D, (P(x) \wedge Q(x))$, and $(\forall x \in D, P(x)) \wedge (\forall x \in D, Q(x))$.
- $\exists x \in D, (P(x) \wedge Q(x))$, and $(\exists x \in D, P(x)) \wedge (\exists x \in D, Q(x))$.
- $\forall x \in D, (P(x) \vee Q(x))$, and $(\forall x \in D, P(x)) \vee (\forall x \in D, Q(x))$.
- $\exists x \in D, (P(x) \vee Q(x))$, and $(\exists x \in D, P(x)) \vee (\exists x \in D, Q(x))$.

Sample Solution

(a) Number the statements in KB in the following way: (1) $\neg p \vee q \rightarrow r$; (2) $s \vee \neg q$; (3) $\neg t$; (4) $p \rightarrow t$; (5) $\neg p \wedge r \rightarrow \neg s$. We now derive $\neg q$.

(3) and (4)	$\neg p$	(6)
(6) and (1)	r	(7)
(6) and (7) and (5)	$\neg s$	(8)
(8) and (2)	$\neg q$	(9)

(b) See the following:

- The first pair is equivalent.
- The second pair is not equivalent. For example, D is the set of integers, $P(x)$ is "x is an even number", and $Q(x)$ is "x is an odd number".
- The third pair is not equivalent. For example, D is the set of integers, $P(x)$ is "x is an even number", and $Q(x)$ is "x is an odd number".
- The last pair is equivalent.