CS 21 Decidability and Tractability **Winter 2023**

Solution Set 1

Posted: January 18

If you have not yet turned in the Problem Set, you should not consult these solutions.

1. We need to define a language that has the same computational power as the function f; that is, given f one should be able to "easily" compute L_f, and vice versa. There are many ways to do this; here is one example:

   Define

   $$L_f = \{(x, k, b) : x \in \Sigma^*, b \in \Sigma, k \text{ an integer and the } k\text{-th symbol of } f(x) \text{ is } b\}.$$

   The alphabet Γ thus includes Σ plus extra symbols to encode "(", ")", ",", and integers (which can be written in binary, and thus encoded using two additional symbols).

   To determine if a string z is in L_f using f, we first determine if z is of the form (x, k, b). If it is, we evaluate $f(x)$, and accept iff the k-th symbol of $f(x)$ is b.

   To compute $f(x)$ using L_f, we do the following: for $k = 1, 2, \ldots$, we determine whether $(x, k, b) \in L_f$ for each $b \in \Sigma$, and we add the b for which we get a positive answer to the output. We stop when we reach a value of k for which $\forall b\ (x, k, b) \notin L_f$ (this marks the end of the string $f(x)$).

2. (a) This is an"if and only if" proof so it must be proven in both directions. That is, (1) given a regular language, show that it can be recognized by an all-paths-NFA, and (2) given an all-paths-NFA show that the language it recognizes is regular.

   (1) If a language is regular, then by the equivalence theorems there is a FA that recognizes the language. Conveniently, a FA is an instance of an all-paths-NFA: since there is only one computation path for any given input, then strings are accepted only if all paths (the only path) ends in an accept state. So every regular language is recognized by an all-paths-NFA.

   (2) To show that a language recognized by an all-paths-NFA is regular, we can construct a FA that recognizes the exact same language as a given all-paths-NFA. This can be done in the same way as the NFA-FA equivalence proof in the lecture slides (or Sipser, Theorem 1.39). That is, if the all-paths-NFA is given by $M = (Q, \Sigma, F, q_0, \delta)$, then the FA M' is given by $M' = (Q' = \mathcal{P}(Q), \Sigma, F', q_0', \delta')$, where

   - For each $R \in Q'$ we have $\delta'(R, a) = \cup_{r \in R}(E(\delta(r, a)))$, as in the lecture (recall that $E(S)$ is defined to be the set of states reachable in M from some state in S via zero or more ϵ-transitions).
   - $q_0' = E(\{q_0\})$ as in the lecture.
   - $F' = \{R \in Q' |$ *every* state in R is an accept state of $M\}$. This corresponds to the fact that for the all-paths-NFA to accept a string, every computation on that string must end in an accept state.

As in the construction in lecture, we can argue that after reading each the first i symbols of the input, machine M' is in a state R which contains exactly those $r \in Q$ that are reachable from the start state in M after processing the first i symbols of the input, plus zero or more ϵ-transitions. This is true even when i is zero, corresponding to the states reachable in M after reading no symbols. Note that if all of these states are accept states in M, then q_0' will be an accept state of M', as required (and both machines will accept the empty string).

(b) If A and B are regular languages, then by part (a) there are all-paths-NFAs, M_a and M_b, that recognize each respectively. Given this fact, we can construct a third all-paths-NFA, M_c, that recognizes $C = (A \cap B)$. Simply create a start state (which is also an accept state) with epsilon-transitions that point to the start states of M_a and M_b. A string is accepted by M_c if and only if all computations end in accept states, which by the construction means all computations end in accept states in both M_a and M_b. Therefore M_c recognizes the intersection of A and B. By part (a) since C is recognized by an all-paths-NFA, it is a regular language. The new start state of M_c should be an accept state so that if ϵ is in both A and B, it will be accepted by all-paths-NFS M_c.

(c) It is useful to define a *valid computation of machine M on input x* as a sequence of states visited while processing x, starting from the start state, and reaching the end of x. Nondeterministic machines may have several valid computations on input x. With this notion in mind, we can see that L_{flip} is the complement of L. To see this, note that if x is a string in L, then some valid computation of M on x ends up in an accept state. Since accept states of M are not accept states in M_{flip}, and M_{flip} requires *all* valid computations to end in accept states, x cannot be a member of L_{flip}. On the other hand, if x is not a string in L, then all valid computations of M on x must end in non-accept states. Since non-accept states in M are accept states in M_{flip}, then all valid computations of x on M_{flip} end in accept states, and x is a member of L_{flip}.
There is an annoying edge case: if x is a string for which there are *no* valid computations of M on x, then x is not accepted by M (i.e. it is not in L), and for M_{flip} to accept x, we have to declare that an all-paths NFA accepts strings x for which there are *no* valid computations of M on x (so, vacuously, "every valid computation of M on x leads to an accept state"). You won't lose points if you missed this detail (which should have been made clearer in the problem statement!).

3. Let L be the language consisting of all palindromes. Suppose L is a regular language. Then the Pumping Lemma must hold. Let p be the pumping length for L. Consider $w = \underbrace{aa\ldots a}_{p} b \underbrace{aa\ldots a}_{p} \in L$. Since $|w| \geq p$, w can be written as $w = xyz$ such that

   (a) for every $i \geq 0, xy^iz \in L$, and

   (b) $|y| > 0$, and

   (c) $|xy| \leq p$.

   By (c), we see that $xy = aa\ldots a$. Thus

$$xy^2z = \underbrace{aa\ldots a}_{p+|y|} b \underbrace{aa\ldots a}_{p}.$$

Since $|y| > 0$, xy^2z is not a palindrome, and thus it is not in L. Contradiction. We conclude that the language consisting of all palindromes is not regular.

4. (a) The easiest way to show that L_n is regular is to show a Finite Automaton A that accepts it. A will have n states $S_0, S_1, \ldots S_{n-1}$ where the accept states are all the states except S_0, which is the start state. The transition function is $\delta(S_j, 0) = S_{(j+1) \pmod n}$. In other words, the machine is in state s_j iff the number of symbols it has read so far, modulo n, is j.

   (b) Suppose PRIMES is a regular language. Then the Pumping Lemma must hold, so there is a pumping length p for this language. Let $q > p$ be a prime, and consider the string $w = \underbrace{00\ldots0}_{q}$, which is in PRIMES.

   By the Pumping Lemma, w can be written $w = xyz$, with $|y| = r > 0$, and for all $i \geq 0$, the string

   $$xy^iz = \underbrace{00\ldots0}_{q+(i-1)r}$$

   is in PRIMES. This means that all the numbers $q + (i-1)r$ must be primes for all $i \geq 0$, but for $i = q + 1$ we have $q + (i-1)r = q(1+r)$ with both $q > 1$ and $r + 1 > 1$, which is a composite. Contradiction. We conclude that PRIMES is not a regular language.