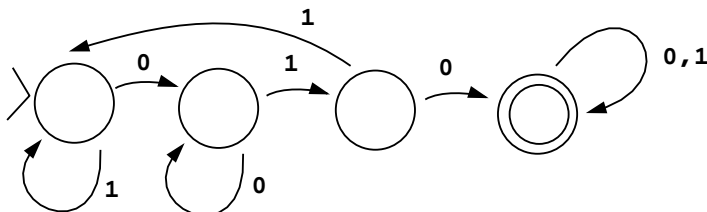


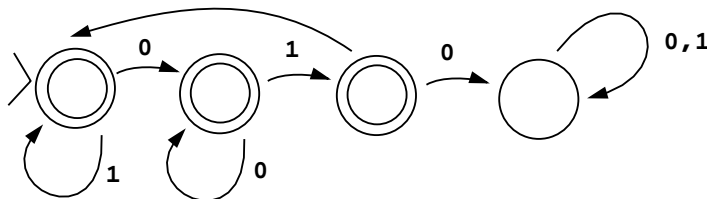
Problem Set 2 Solutions

Problem 1. Give DFAs for the following languages. Assume an alphabet of $\Sigma = \{0, 1\}$.

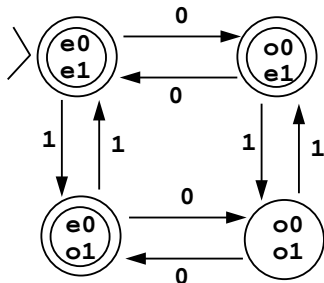
(a) The set of all strings with 010 as a substring



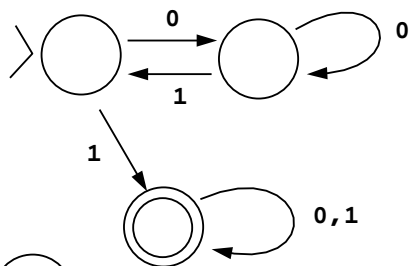
(b) The set of all strings which do not have 010 as a substring



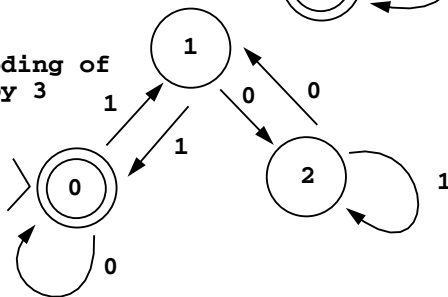
(c) The set of all strings which have an even number of 0's or an even number of 1's



(d) The complement of $\{0, 01\}^*$



(e) The binary encoding of numbers divisible by 3



Problem 2 State whether the following propositions are true or false, proving each answer.

Part A. Every DFA-acceptable language can be accepted by a DFA with an even number of states.

True. The idea is to add a “dummy state” in the case that the machine has an odd number of states. Formally, given a DFA $M = (Q, \Sigma, \delta, q_0, F)$, if $|Q|$ is even set $M' = M$ and if $|Q|$ is odd then let $M' = (Q \cup \{s\}, \Sigma, \delta', q_0, F)$ (where $s \notin Q$) where $\delta'(q, \sigma) = \delta(q, \sigma)$ for $q \in Q, \sigma \in \Sigma$ and $\delta'(s, \sigma) = s$ (say) for $\sigma \in \Sigma$. Clearly $L(M) = L(M')$ and M' has an even number of states.

Part B. Every DFA-acceptable language can be accepted by a DFA whose start state is never visited twice.

True. Add a new start state and connect it up to all the states that the old start state was connected to, in the same way. Formally, given a DFA $M = (Q, \Sigma, \delta, q_0, F)$ construct a DFA $M' = (Q \cup \{s\}, \Sigma, \delta', s, F')$ (for $s \notin Q$) by saying $\delta'(q, \sigma) = \delta(q, \sigma)$ for $q \in Q, \sigma \in \Sigma$, and $\delta'(s, \sigma) = \delta(q_0, \sigma)$ for $\sigma \in \Sigma$, and $F' = F$ if $q_0 \notin F$ and $F' = F \cup \{s\}$ if $q_0 \in F$.

Part C. Every DFA-acceptable language can be accepted by a DFA no state of which is ever visited more than once.

False. Only finite languages can be accepted by such a machine, and some DFA-acceptable languages are infinite.

Part D. Every infinite DFA-acceptable language can be accepted by a DFA that, for some string $x \in L$, visits the start state twice on input x .

False. First make sure you understand the question! The (false) assertion means that for any infinite DFA-acceptable language $L = L(M_0)$ there exists a DFA $M = (Q, \Sigma, \delta, q_0, F)$ and a string $x \in \Sigma^+$ such that $L(M) = L$ and $\delta^*(q_0, x) = q_0$. This isn't going to be possible because, intuitively, some languages need you to remember the first character of the string they are fed. To disprove this assertion you must exhibit a DFA-acceptable language L for which no such M and x exists. Consider $L = \{ab^i a : i \geq 0\}$. Then assume for contradiction that there exists a machine $M = (Q, \{a, b\}, \delta, q_0, F)$ that accepts L and a string $x \in \Sigma^+$ such that $\delta^*(q_0, x) = q_0$. Then for any $y \in \{0, 1\}^*$ we must have that $\delta^*(q_0, xy) = \delta^*(q_0, y)$, and so $y \in L$ iff $xy \in L$. The first character of the nonempty string x is either a or b . But if $x = ax'$ then it is not the case that $y \in L$ iff $ax'y \in L$ (for example, $y = bb \in L$ but $ax'bb \notin L$); and if $x = bx'$ for some x' then it is also not the case that $y \in L$ iff $ax'y \in L$ (for example, $y = aa \in L$ but $bx'aa \notin L$).

Part E. Every DFA-acceptable language can be accepted by a DFA with only a single final state.

False. Consider $L = \{\varepsilon, 1\}$ over the unary alphabet $\Sigma = \{1\}$. Then L can be accepted by the obvious three-state DFA having two final states, but L can not be accepted by any DFA having only one final state. To see the latter claim, I claim that if $M = (Q, \Sigma, \delta, q_0, F)$ is a DFA that accepts L then $\delta^*(q_0, \varepsilon) = q_0 \in F$ and $\delta^*(q_0, 1) = \delta(q_0, 1) \in F$ must be distinct states. Indeed if these states were not distinct, $q_0 = \delta(q_0, 1) \in F$, then $\delta(q_0, 1) = \delta(\delta(q_0, 1), 1) = \delta^*(q_0, 11) \in F$, so the machine would incorrectly accept 11.

Problem 3. Let $\text{Extend}(L) = \{x \in L : \text{there exists a } y \in \Sigma^+ \text{ for which } xy \in L\}$.

Part A. What is $\text{Extend}(\{0, 1\}^*)$? What is $\text{Extend}(\{\varepsilon, 0, 1, 00, 01, 111, 1110, 1111\})$?

$\text{Extend}(\{0, 1\}^*) = \{0, 1\}^*$, while $\text{Extend}(\{\varepsilon, 0, 1, 00, 01, 111, 1110, 1111\}) = \{\varepsilon, 0, 1, 111\}$.

Part B. Prove that if L is DFA-acceptable then $\text{Extend}(L)$ is too.

Given a DFA $M = (Q, \Sigma, \delta, q_0, F)$ for L , a DFA $M' = (Q, \Sigma, \delta, q_0, F')$ is constructed for $\text{Extend}(L)$ by “pruning” the final state set; we define F' to be the set of all states $q \in F$ such that there exists some nontrivial path from q to some final state of M . Then $x \in L(M')$ iff $x \in L$ and there is some $y \in \Sigma^+$ such that $xy \in L(M)$.

Problem 4.

Part A. Recall the DIOPHANTINE EQUATION problem: given a multivariate polynomial P with integer coefficients (e.g., $P(x, y, z) = x^2 - 5xy + 3yz^2 + xyz$), decide whether or not P has an integer root. I claimed without proof that there is no algorithm to answer this question. But suppose I provide you with a “magic box” that answers the question. In a single computational step, it says yes or no according to whether or not P has a root. Given such a magic box, describe an algorithm that finds an integer root of any multivariate polynomial that has one (and the algorithm answers No Root if the polynomial provided doesn’t have an integer root).

The following magicbox-using algorithm finds a root or identifies that there is none:

```
Algorithm FindRootWithHelpOfMagicBox( $P(x_1, \dots, x_k)$ )
if MagicBox( $P$ )=no then return No Root
for  $m \leftarrow 1$  to  $\infty$  do
  for every  $c_1, \dots, c_k \in \{-m, \dots, m\}$  do
    if  $P(c_1, \dots, c_k) = 0$  then return  $(c_1, \dots, c_k)$ 
```

Notice that (despite the apparently infinite **for** loop) the pseudocode above always terminates: if P has a root then there is a smallest m such that P has a root in which every variable has magnitude is at most m , and the algorithm will terminate when it gets to that m .

The technique used above is called *dovetailing* (I am hoping that you “invented” the method for solving this problem). We will be using dovetailing quite a bit in the second half of this course.

Part B. Let $s(n)$ be the maximum number of computational steps that your algorithm takes to run (on some fixed, magicbox-containing computer) when the polynomial P contains at most n variables and each coefficient is between $-n$ and n . Explain why there is no algorithm to compute $S(n)$ for any function S such that $S(n) \geq s(n)$ for all n .

Suppose for contradiction that there exists an algorithm FindS that calculates an $S(n)$ as above. Then consider the following algorithm that uses FindS:

```
Algorithm FindRoot( $P(x_1, \dots, x_k)$ )
let  $n$  be the larger of  $k$  and the absolute value of the coefficient of  $P$  of largest magnitude
for  $m \leftarrow 1$  to FindS( $n$ ) do
  for every  $c_1, \dots, c_k \in \{-m, \dots, m\}$  do
    if  $P(c_1, \dots, c_k) = 0$  then return Has A Root return No Root
```

This algorithm correctly decides if P has a root under the assumption that FindS(n) bounds the running time of FindRootWithHelpOfMagicBox. That’s because for any P that has a root, we are certain to run with the m -value that will cause some root to be identified. Since we know (from my unproven claim) that it is impossible to make an algorithm that decides if P has a root, it must be the case that it is impossible to compute FindS as well.