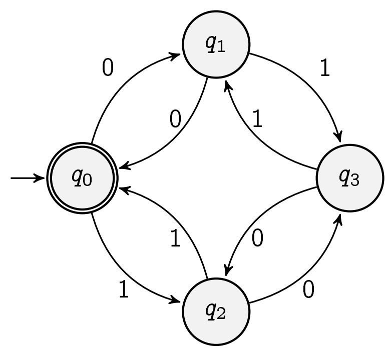
CSC236 tutorial exercises, Week #12
sample solutions

1. Consider the language EVENEVEN $= \{x \in \{0,1\}^* \mid x$ contains an even number of 0's and 1's$\}$. Below is a 4-state DFSA that accepts EVENEVEN:



Prove that it is impossible to construct a DFSA for this language with fewer states than this.

Solution:
Consider the four prefixes $x_1 = \varepsilon$, $x_2 = 1$, $x_3 = 0$, and $x_4 = 01$. We also define an identical sequence of suffixes, $y_1 = x_1, y_2 = x_2$, etc.

By inspection, we can see that the concatenation $x_j y_k$ is in EVENEVEN if and only if $j = k$.

Suppose for sake of contradiction that M is a DFSA with less than 4 states which accepts EVENEVEN. Then by the pigeonhole principle, there exists a state q such that $\delta^*(s, x_j) = \delta^*(s, x_k) = q$, where $j \neq k$.

Because $x_j y_j \in$ EVENEVEN, $\delta^*(q, y_j)$ must be accepting. But this means we also accept $x_k y_j$ which is known not to be in the language, a contradiction. So it is impossible for M to have fewer than 4 states.

2. Consider the language SPLIT consisting of strings of the form $x\#y$ where $x, y \in \{0,1\}^*$ and $|x| = |y|$. Prove that SPLIT is not regular. You may use the pumping lemma, or directly apply the pigeonhole principle.

Solution:
Suppose M is a DFSA that accepts SPLIT. By the pigeonhole principle, there exist distinct $n, m \in \mathbb{N}$ such that $\delta^*(s, 0^n) = \delta^*(s, 0^m) = q$ for some state q. Since $0^n\#0^n \in$ SPLIT, $\delta*(q, \#0^n)$ is an accepting

state. But this means we also accept $0^m\#0^n$ which is not in the language, a contradiction. So SPLIT is not accepted by any DFSA, and is therefore non-regular.

Pumping lemma alternative: Assume SPLIT is regular, and let n be the pumping length. Consider the string $x = 0^n\#0^n$. The pumping lemma says that there is some segment of the first n characters (which must be of the form 0^k for some $0 < k \leq n$) which we can repeat any number of times. However, $0^{n+k}\#0^n$ is not in SPLIT, for $k > 0$, a contradiction.

3. Which of the following languages are regular? (You don't need to provide proofs, though you should think about how you *would* prove each answer if you had to.)

   (a) DOUBLEZEROS: strings in $\{0, 1\}^*$ having twice as many zeros as ones

   Solution:
   Non-regular. 1^n needs to go to a different state for every value of n, because each such string has a different suffix (namely 0^{2n}) that puts it in the language.

   (b) PHONES: the language of 7-digit telephone numbers, e.g. '555-5555'.

   Solution:
   Regular. We could show this by explicitly constructing a DFSA or RE that matches this language. Here's an RE that does the job: $ddd - dddd$, where d expands to the expression $(0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9)$. (We might need to tweak this slightly to match the nuances of North American phone numbers. For example, 0 and 1 aren't allowed as the first digit.)

   We could also observe that PHONES is clearly finite (it's a subset of the finite language $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -\}^8$), and use the fact that all finite languages are regular.

   (c) PAN: the language of 'pangrams', i.e. strings that contain at least one of every letter from a-z. e.g. 'the quick brown fox jumps over the lazy dog'.

   Solution:
   Regular. Constructing a full RE or FSA for this language would be tedious, but it's easy to do for a small alphabet like $\{a, b\}$, and it's clear the same technique could be extended to arbitrarily large alphabets.

   (d) PYTHON: the language of valid Python programs. e.g. 'print(1+1)' is in the language, but 'print(1+)' is not, because it raises a SyntaxError.

   Solution:
   Non-regular. This may seem intuitively reasonable given the complexity of the syntax of Python, but *proving* non-regularity is difficult for the same reason. A helpful trick here is to focus on a narrow subset of Python expressions. For example, ((())) is a valid Python expression, but if we remove any one parenthesis, it becomes invalid. We can use a pumping lemma/pigeonhole principle argument to show that no FSA can handle this class of expressions.

   (e) SMALLPRIMES: strings of the form 1^n where n is a prime number less than 1000.

   Solution:
   Regular. Without the size restriction, this would not be a regular language, but because of the limit of 1000, this language is finite and therefore regular. We can represent this language by an RE like $(11 + 111 + 11111 + 1111111 + \ldots)$ (and it's unlikely we can get much more compact than this).