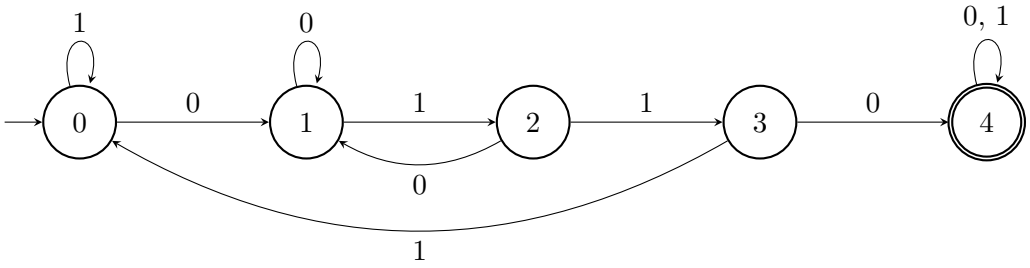
CS 181 Homework 3

Einar Balan

1. As we mentioned in class the critical part of KMP string matching algorithm is the following: Given a pattern $p \in \{0,1\}^*$, we have to design a DFA that accepts only those strings that contain p. Let us explore this idea for a bit.

   Suppose the pattern p is 0110, can you design a DFA that only accepts strings that contain this pattern?

Answer



The DFA above works by only progressing to the next state when the correct bit is seen. If it is not seen, then the state is reset to the state which would have been progressed to if the bit was read frrom the initial state. Once we encounter p, we enter a persistent accepting state.

2. Converting a regular expression to an NFA. In class, we saw a procedure to build a NFA for every regular expression r. Let us understand the 'size' of the NFA created by this procedure as a function of the length of the regular expression r. Here, length of the regular expression refers to the number of characters in the regular expression.

   Suppose the regular expression r had length m. How many states would be present in the final NFA built by the procedure in class as a function of m (use big-Oh notation and avoid computing constants)? Explain your reasoning in a few sentences [.5 points]

   Here, by size of a regex you can take it to mean the length of the string that corresponds to the regex. For instance, the regex $(01)^*$ would have size 5. Alternately, more intuitively, you can take size to mean the number of recursive subpieces and operations that you'd see within the regular expression. Any way you interpret the size, the answer should be same up to big-Oh notation.

   Do not worry about figuring out the edge-cases and/or the exact constants.

   [Hint: Look at each of the steps in the compound case, and see how the number of states can increase.]

Answer $O(m)$ states.

Considering base cases:

- "0" regex has an NFA with 2 states
- "1" regex has an NFA with 2 states

Considering each compound case:

- concatenation simply adds epsilon transitions from accepting states of the first NFA to the initial state of the second, so performing c concatenations in the regex will add no states beyond the base cases
- union can be computed by adding a new inital state with epsilon transitions to both original initial states, so if there are u unions in the regex this translates to adding u states in the NFA
- kleen* is computed by adding a new inital state and epsilon transitions, so if there are k kleen* operations in the regex this translates to adding k states in the NFA

So if there are n 0s or 1s in the regex, we will have at most $2n$ states in the NFA representing them. We know that each of $n, c, u, k < m \therefore O(2n + 0c + u + k) = O(m)$ states in the NFA.

3. Design a regular expression for the following language:

$$L = \{x : \text{ every 1 in } x \text{ is followed by at least two zeroes }\}.$$

For instance, $010010000, 1000, 0010010000$ are in the language but $10101, 011001000$ are not.You don't have to prove your expression works but write a few sentences describing why your expression could/does work. [.75 point]

Answer $r = (0|100)^*$

At every point, we only want to accept a string that is either a 0 or a 1 followed by two 0s. We use the union operation to capture 0 or 100 and we use kleene* to extend this logic to every point in the string.

4. Show that the following functions/languages are not regular:

   (a) $L = \{x : x = 0^m 1^{3m}\}$. For instance, $0111, 00111111$ would be in L, but $0100, 0011$ would not be elements of L. [.75 points]

   (b) $F : \{0,1\}^* \to \{0,1\}$ defined by $F(x) = 1$ if and only if $x = 1^{i^3}$ for some $i > 0$. [.75 points]

   (c) $UNEQUAL = \{0^m 10^n : m \neq n\}$. [.75 points]

   Your proofs should be at the similar level of detail as the examples from lecture 11; that is, write down the different steps as we did in class.

Answer

(a) Suppose L was regular. Then there must exist p such that PL holds. Consider $x = 0^p 1^{3p}$, which is in the language and has length greater than p. If we split x into $a, b,$ and c subcomponents such that length$(ab) \leq p$, then $abbc$ is not in the language. This is because we know that the split between a and b must lie within the section of 0's due to the length restriction. So no matter what, pumping b will result in changing the number of 0's while leaving the number of 1's constant and the number of 1's will no longer be 3 times the number of 0's. This is a contradiction, so L must not be regular.

(b) Suppose F was regular. Then there must exist p such that PL holds. Consider $x = 1^{p^3}$, which is in the language and has length greater than p. If we split x into $a, b,$ and c subcomponents such that length$(ab) \leq p$, then ab^0c is not in the language. We know that $0 <$ length$(b) \leq p$ so the length l of ab^0c is $p^3 - p \leq l < p^3$. The previous cube is $(p-1)^3 = p^3 - 3p^2 + 3p - 1$ which is less than the range of lengths possible for ab^0c, therefore it must not be in the language. This is a contradiction, so L must not be regular.

(c) Consider $L = \{x :$ starting and ending bits are the same$\}$. We know L is regular because it is computed by the regex $r = ((0(0|1)^*0) \mid (1(0|1)^*)1)$. Suppose that $UNEQUAL$ is regular as well. Then there must exist p such that PL holds for $L \setminus UNEQUAL$ (by closure properties). Also consider $x = 0^p 10^p$, which is in $L \setminus UNEQUAL$. If we split x into $a, b,$ and c subcomponents such that length(ab) $\leq p$, then $abbc$ is not in the language. This is because we know that the split between a and b must lie within the first section of 0's due to the length restriction, so no matter what pumping b will only change the number of 0's in the first section, causing the string to be in $UNEQUAL$ and therefore not in $L \setminus UNEQUAL$. This is a contradiction, so $L \setminus UNEQUAL$ must not be regular. We showed L was regular, so it must be that $UNEQUAL$ is not regular.

Additional practice problems - Not to be graded

1. (COMMENTED OUT) Show that if a language L is regular, then so is $Reverse(L) = \{x : Reverse(x) \in L\}$.

2. Design a regular expression for the following languages:

   - $L = \{x :$ third bit from end of x is a 1$\}$.
   - $L = \{x :$ number of 1's in x is divisible by 5$\}$.
   - $L = \{x :$ x has an odd number of 1's$\}$.
   - $L = \{x :$ x has at least three 1's$\}$.
   - L = All strings except the empty string.
   - $L = \{x :$ x has an even number of 0's or contains exactly two 1's$\}$.

3. Continuing problem (2) above:

   3a How many transitions/edges are present in the final NFA built by the procdure in class as a function of m (use big-Oh notation and avoid computing constants)?

   [Hint: Look at each of the steps in the compund case, and see how the number of edges added changes.]

   3b (Advanced) Do you see a way to modify the procedure to only add at most $O(m)$ transitions in the NFA? Such an NFA is what gets used in grep and other regex matching tools.

   [Hint: The most expensive step in terms of number of edges added is the compound case corresponding to the (*) operation. What happens if you maintain the invariant that all the NFAs you build only have one accept state?Can we maintain this invariant?]

4. Show that the following languages are not regular.

   (a) $L = \{0^{|w|} \circ w : w \in \{0,1\}^*\}$

   (b) $L = \{ww : w \in \{0,1\}^*\}$