

Problem 4.5

Algorithm

1. Input a DFA A
2. Count the number of states in A . Call this number s .
3. Check if any string of length between $s + 1$ and $2s$ is accepted by A
 - if there is such an accepted string, accept; else reject

Notes

- Since the number of strings of lengths $s + 1$ to $2s$ is finite, the algorithm will halt.
- The computation path for any string found will have to repeat a state. So if we find such a string, $L(A)$ must be infinite (we can “repeat the loop” as often as we like). In other words, any string found in the algorithm will be pumpable.
- If $L(A)$ is infinite, it must contain a pumpable string. Let x be a shortest pumpable string. If $|x| < s$, then we can pump it up to get a string of the desired length. If $|x| > 2s$, then x is not the shortest pumpable string, since some state must occur 3 times along the computation path for x . If we delete the part of the path between the first two occurrences, we will get a path for a shorter pumpable string accepted by A . (It will be pumpable because there will still be a repeated state along the computation path.) This shows that if $L(A)$ is infinite, we will find a string of the desired length.

Problem 4.6

f not one-to-one because $f(1) = f(3)$. f not onto because 10 is not in the range of f . g is one-to-one since everything from X has exactly one partner in Y (no values in second column occur twice). g is onto since everything in Y has exactly one partner in X (every element from Y shows up in the second column). So f is not a correspondence but g is.

Problem 4.7

Let s_1, s_2, s_3, \dots be a listing of infinite sequences over $\{0, 1\}$. (That is $s : \mathbb{N} \rightarrow \{0, 1\}^\infty$.) We must show that the list is incomplete (that is, s is not onto) by finding a “missing element”. So we build a new infinite sequence s so that the i th bit of s is the opposite of the i th bit of s_i . That is,

$$s(i) = \begin{cases} 0 & \text{if } s_i(i) = 1 \\ 1 & \text{if } s_i(i) = 0 \end{cases}$$

Since s is different from each s_i in the list, s is missing from the list, hence this list is incomplete.

Since this argument could be applied to any such list, every such list must be incomplete. In other words, every function from the natural numbers to the set of infinite binary strings must fail to be onto.

Problem 4.8

For any $(i, j, k) \in T$, define $f(i, j, k) = 2^i \cdot 3^j \cdot 5^k$. $f : T \rightarrow \mathbb{N}$ and T is one-to-one, since each natural number n has a unique factorization, from which we can recover i , j , and k (provided n is of the correct form). This shows that $|T| \leq |\mathbb{N}|$.

On the other hand we can define $g : \mathbb{N} \rightarrow T$ by $g(n) = (n, n, n)$. Then g is one-to-one (if $(m, m, m) = (n, n, n)$ then m and n must be the same number), so $|\mathbb{N}| \leq |T|$. Putting these together we see that $|T| = |\mathbb{N}|$.

It is also possible to build a single function that is one-to-one and onto.

Problem 4.9

We must show that the “same size as” relation is reflexive, symmetric and transitive.

- reflexive: the identity function (everything maps to itself) is one-to-one and onto, so A is the same size as A for any A .
- symmetric: If $|A| = |B|$ then there is a function $f : A \rightarrow B$ that is one-to-one and onto. f^{-1} (the inverse function) is a function because f is one-to-one, it is onto because f , it is one-to-one because f is a function. So there is a one-to-one and onto function from B to A .
- transitive: If $f : A \rightarrow B$ is one-to-one and onto and $g : B \rightarrow C$ is one-to-one and onto, then $g \circ f : A \rightarrow C$ is one-to-one and onto.

Problem 4.11

Algorithm

1. Input regex's R and G
2. Build DFA's M_R and M_G that accept the languages generated by R and G .
3. From M_R and M_S build a DFA M that accepts the language $L(M_R) - L(M_S)$
4. Check if $L(M)$ is empty.
 - if it is empty accept; else reject

Notes

- We have shown that step 2 can be done algorithmically (i.e. with a Turing machine) in chapter 2.
- Step 3 can be done very much like the way we built machines for unions and intersections of regular languages: Each state in the new machine corresponds to two states – one from each of the machines M_R and M_S . The accepting states are those corresponding to an accepting state of M_R and a rejecting state in M_S .
Alternatively, we could notice that $A - B = A \cap \overline{B}$, and use the fact that we have already shown how to get machines to do intersections and complements.
- Step 4 is essentially the algorithm that decides E_{DFA} (see page 154).
- $A - B$ is empty exactly when $A \subseteq B$, so testing if $M \in E_{DFA}$ tells us if $L(R) \subseteq L(S)$.

Problem 4.14

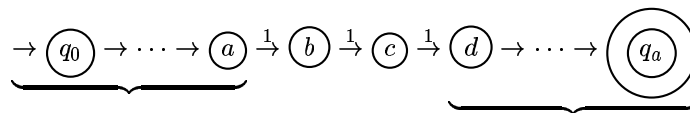
Algorithm

1. input a regex R .
2. Build a DFA M that accepts $L(R)$.
3. Explore all paths from the start state to the accept state such that no state occurs on the path more than 4 times.
4. Check if any of these paths has three consecutive edges labeled with a 1.
 - if there is such a path accept
 - if there is not such path, reject

Notes

1. Since the number of states in M is finite, the number of paths explored in step 3 is finite, so the algorithm will eventually halt.

2. If M does not accept any strings containing 111, then clearly this algorithm will reject.
3. If M does accept a string containing 111, we must argue that this algorithm will accept R . Let x be a shortest such string and consider the path through the state diagram for M on input x . It looks something like:



Note that in the two bracketed sections, no state can be repeated (else we could remove the loop without removing the 111 and find a shorter string than x). So in the worst case, $b = c$ and also occurs once in the beginning phase and once in the ending phase of the computation path.