CSC265 Fall 2020 Homework Assignment 9

Solutions

The list of people with whom I discussed this homework assignment: None

Suppose that $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ are two deterministic finite automata with disjoint state sets. Let $Q = Q_1 \cup Q_2$, let $F = F_1 \cup F_2$, and let $\delta : Q \times \Sigma \to Q$ denote the function

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{if } q \in Q_1 \\ \delta_2(q, a) & \text{if } q \in Q_2. \end{cases}$$

Two states $q, q' \in Q$ are *equivalent* if, for all $x \in \Sigma^*$,

$$\delta(q, x) \in F \text{ if and only if } \delta(q', x) \in F.$$

Lemma q and q' are equivalent states if and only if

- for all $a \in \Sigma$, the states $\delta(q, a)$ and $\delta(q', a)$ are equivalent, and
- either $q, q' \in F$ or $q, q' \in Q - F$.

You may use this lemma without proof. It can be proved by induction, but that is not part of the assignment.

1. Explain how to use the DISJOINT SETS abstract data type to test whether two deterministic finite automata accept exactly the same set of strings. Justify the correctness of your algorithm.

   Solution:
   Algorithm: First, for every state s ∈ Q, MAKESET(s).

   Then, for each $q \in Q_1$, for each $q' \in Q_2$, if FINDSET(q) $\neq$ FINDSET(q') and if for every $x \in \Sigma^*$, [either (FINDSET($\delta(q, x)$) = $FINDSET(\delta(q', x))$ and (either q and q' both in F or in Q-F)) or ($\delta(q, x) \in F$ and $\delta(q', x) \in F$)], then q and q' are equivalent and LINK(q,q'), and start iterating both Q_1 and Q_2 from the beginning again. Denote the outermost loop l1.

   After one complete iteration of the loop above, iterate Q_1 again. Using a similar loop structure as the last loop, if for every state q in Q_1, there exists (which is done using a for loop with a break when found an instance) a state q' in Q_2 such that for every $x \in \Sigma^*, FINDSET(\delta(q, x)) = FINDSET(\delta(q', x))$ and either q and q' both in F or in Q-F, then the two sets accept exactly the same strings. If this check fails for any state q in Q_1, then they do not. Denote the outermost loop l2.

   Justification:
   Assume there are n elements in Q, m elements in the alphabet set Σ.
   Lemma 1: the innermost loop of l1 iterates at most mn^3 times. After loop l1 terminates, all equivalent states are in the same node of the disjoint set.

Consider the states in F, they are all equivalent to each other. At the beginning of the loop, each node contains exactly one element. all combinations of states in Q_1 and Q_2 are iterated, and checked with all potential alphabets, such that the first equivalent pair is found using the original definition. This exhaustive search takes at most mn^2 iterations, with at most 2 FINDSET for each iteration. Then, all the combinations are checked again such that if any pair satisfies either the definition or the lemma, they are linked together.

Since for every link operation, the candidates satisfies either the definition of equivalence or the lemma, all linked elements are equivalent. Therefore, during the loop iteration, all states in the same node of the disjoint set are equivalent.

Using induction, we know that when the loop terminates, all elements in the same node are equivalent. Every search for a potential equivalence candidate pairs take mn^2 iterations, and since if they are already linked together, they will not be checked again since they will have the same representative and that there are at most n elements in Q, the iteration runs at most mn^3 times. Since the iteration is exhaustive, if any state in Q1 is equivalent with any state in Q2, then they will be linked together.

Theorem: after loop l2 terminates, if all checks succeed then two sets accept exactly the same strings, if any check fails for any state q in Q_1 then they don't.
First justify that succeeding the check means having the same accepted string set. If two sets accept exactly the same strings, then for each string, M1 and M2 will reach a state in F with the string as input. this means that the state before inputting the final alphabet is equivalent, and so the state before inputting the second last alphabet is equivalent... Therefore, for every possible input, if M1 and M2 are at q1 and q2 or any equivalent state, the state of M1 and M2 after the input must also be equivalent. This means that every equivalent state must be grouped together, which is done by lemma 1.

Now suppose there is a failed check. This means that there is a state, without loss of generality assume in Q1, with no equivalent state in Q2. consider these states as q and q'. By lemma, either only one of them is in F(which means that their accepted input set is different), or there exists an input which produces different non-equivalent states. Repeatedly consider the two new different non-equivalent states again, until one of them is in F. Then, since they aren't equivalent, only one of them is in F.

2. What data structure should you use to implement the DISJOINT SETS abstract data type for this application? Justify your answer.

   Solution: The best data structure is a tree with path compression and union by weight (or rank). Consider the algorithm in question 1. we called n times MAKESET, and by our proof in Lemma 1, there are at most $4mn^3$ times of FIND-SET operations (there are 2 times of FINDSET for each of the mn^3 iterations for both l1 and l2 since their loop structure is identical except that l2 has an early terminating break condition, and FINDSET is called nowhere else). Assume that there are a total of k operations in the algorithm, which will not change no matter we use which implementation. LINK is called at most once in every iteration in l1 only, so there are at most mn^3 calls. Since the number of calls to FINDSET and MAKESET are asymptotically the same and may be very large, we want a tree with path compression and union by weight which may provide log(n) time and have the least asymptotic sequence complexity.