CSC265 Fall 2020 Homework Assignment 9

List of People I Talked to: Kunal Chawla, Raymond Liu, Jenny Zi Yi Xu.

1. We first present our algorithm. We assume every state has an additional field indicating whether it has been added to our disjoint-set DS.

```
1   push $(q_1, q_2)$ onto stack $ST$
2   while $ST$ is nonempty:
3       $(q, q') = ST.\textsc{Pop}()$
4       for $a \in \Sigma$:
5           if $\delta(q, a)$ or $\delta(q', a)$ are not in $DS$, use $DS.\textsc{Make-Set}$ to add them
6           if $DS.\textsc{Find-Set}(\delta(q, a)) \neq DS.\textsc{Find-Set}(\delta(q', a))$:
7               $DS.\textsc{Union}(\delta(q, a), \delta(q', a)))$
8               $ST.\textsc{Push}((\delta(q, a), \delta(q', a)))$
9   if for each set $S \in DS$, $\forall x \in S.x \in F$ or $\forall x \in S.x \notin F$:
10      return true
11  return false
```

We first show the termination of this algorithm.

Lemma 1. *The algorithm terminates.*

Proof. It suffices to show the loop from line 2 to 8 terminates. Notice the number of elements in ST decreases by 1 on line 3 every iteration. Also, there are finite number of pairs in $Q \times Q$, and each pair will be added at most once to ST on line 8 because line 7 unions the sets the two states belong to and so the condition on line 6 cannot be satisfied again. Combining these two facts, we know ST will eventually be empty, and thus the loop terminates. $\square$

We now show the correctness of output.

First, some terminology: $x \in Q_1, y \in Q_2$, if there exists a string $s \in \Sigma^*$ such that $\delta^*(q_1, s) = x$ and $\delta^*(q_2, s) = y$, we call this situation x is equivalent to y via s. Consider the following equivalence relation E on Q', the set of states in Q that are reachable from q_1 or q_2: for states $z, z' \in Q'$, $z \sim_E w$ iff $z = z'$ or there exists a chain of length k

$$\begin{aligned} z = z_0 \sim_E z_1 &\quad \text{via some string } s_0 \\ \dots& \\ \sim_E z_k = z' &\quad \text{via some string } s_{k-1}. \end{aligned}$$

Notice that by the end of our algorithm, DS is a partition of Q' since if a node x is reachable, then x will be q or q' on line 3 for some iteration[1],

[1]It can be proven by induction on the minimum path length from q_1 or q_2 to x that any reachable state x will be one of q or q'.

and line 5 adds x to DS. Since line 5 checks if x is already in DS, x will not be added twice. Observe furthermore that no nonreachable state are added to DS[2]. Thus, every state in Q' is in some set in DS and no other states are in DS, making DS a partition of Q'.

We need to show that by the end of our algorithm above, DS is a partition of Q' by E.

We first show that the partition by E is at least as fine as DS. Let $z, z' \in Q'$, we show $z \sim_E z'$ implies z and z' are in the same set in DS. The case where $z = z'$ is trivial (z is in DS by lemma above). Suppose $z \sim_E z'$ via some string s_0. We will need the following lemma.

Lemma 2. *If $z, z' \in Q'$ are such that $z \sim_E z'$ via some string s_0, then z and z' are in the same set in DS.*

Proof. We induct on the length of s_0 with a stronger hypothesis that if z and z' are in the same set, then z, z' will be q, q' during some iteration. For the base case, suppose $|s_0| = 1$. Then, we know $z = \delta(q_1, s_0)$ and $z' = \delta(q_2, s_0)$. But since (q_1, q_2) is pushed onto ST on line 1, we know on the first iteration of the following loop and on some iteration of the inner loop, $a = s_0$. And by line 7 and 8, we are done. Now suppose the lemma holds for $|s_0'| < k$, where $k \geq 2$, assume $|s_0| = k$. In this case we can write $s_0 = wa$ where $a \in \Sigma$. We know $\delta(q_1, w) \sim_E \delta(q_2, w)$ via w. By induction hypothesis, we have that $\delta(q_1, w), \delta(q_2, w)$ will be q, q' during some iteration. Applying the same argument as above, we know that z and z' are in the same set in DS. $\square$

Using this lemma, we see that z and z' are in the same set in DS. Suppose $z = z_0 \sim_E \ldots \sim_E z' = z_k$ via $s_0, \ldots, s_{k-1}$ for some $k > 1$, we can assume by induction hypothesis that z, z_{k-1} are in the same set in DS. By base case, we know z_{k-1}, z_k are in the same set. Thus, $z, z_k = z'$ are in the same set.

We now show DS is at least as fine as the partition by E. Let $z, z' \in Q'$, we show z and z' are in the same set in DS implies $z \sim_E z'$. This can be done by induction on the size m of the set in DS containing z, z'[3]. It is obvious that $m \geq 2$ by line 6 and 7 (no state in DS is left alone). For $m = 2$, this implies the set z, z' belong in is formed by taking the union of singletons (which happens on line 7), and we will need the following lemma.

Lemma 3. *If $(q, q') \in ST$ after the first iteration of the loop on lines 2-8, then there exists a string s such that $q \sim_E q'$ via s.*

Proof. This is by induction on the iteration of the loop on lines 2-8. After the first iteration, this is true since if $(q, q') \in ST$, then $q = \delta(q_1, a)$ and

[2] Again, this is by induction on the minimum path length from the initial state to x

[3] The intuition is formed by viewing the set as a connected graph.

$q' = \delta(q_2, a)$ for some $a \in \Sigma$. So $q \sim_E q'$ via a. Suppose the loop runs for at least $k > 1$ iterations and (q, q') is newly added to ST during the kth iteration. Then we know during the kth iteration $q = \delta(p, a), q' = \delta(p', a)$ for some $a \in \Sigma$, where p, p' (by induction hypothesis) are states such that $p \sim_E p'$ via some string s. But then we have trivially that $q \sim_E q'$ via sa. □

Using the lemma, we can conclude that $z \sim_E z'$ via some string s_0. Now suppose $m > 2$. If $z \sim_E z'$ via some string directly, then we are done. Denote the set containing z, z' by S. We know that since $z' \in S$, some $z'' \in S$ has $z' \sim_E z''$ via some string s. By induction hypothesis on the set $S \setminus \{z'\}$, we are done.

Lemma 4. *M_1 and M_2 accept the same language if and only if for every equivalence class of E, all states in the equivalence class are final states or all are nonfinal states.*

Proof. For the forward direction: Suppose that M_1 and M_2 accept the same language, and let $z, z' \in Q'$ be two reachable states such that $z \sim_E z'$, we show either both of them are final states or both are nonfinal states. Notice that $z \sim_E z'$ means there exists a chain:

$$\begin{aligned} z = z_0 &\sim_E z_1 \quad \text{via some string } s_0 \\ &\cdots \\ &\sim_E z_k \quad \text{via some string } s_{k-1}. \end{aligned}$$

But since M_1 and M_2 accept the same language, we know that z and z_1 must have the same final/nonfinal status. Similarly for z_1 and z_2, and so on... This means the final/nonfinal status of z and z' are the same.

The backward direction is clearly true: if M_1 and M_2 accept different languages, then there exists a string s such that the final/nonfinal status of $\delta^*(q_1, s)$ and $\delta^*(q_2, s)$ are different. However, $\delta^*(q_1, s) \sim_E \delta^*(q_2, s)$ via s. □

Theorem 1. *The algorithm is correct.*

Proof. Termination is proved in lemma 1. We have also shown that DS is a partition of Q' that is equal to E. By the last lemma, we know that M_1 and M_2 accept the same language if and only if for every equivalence class of DS, all states in the equivalence class are final states or all are nonfinal states. This is exactly the condition we are checking on line 9 of our algorithm. Thus, our algorithm is correct. □

2. We shall use trees with rank-by-union and path compression for this application. We will go through a list of candidates and briefly explain why they are not chosen.

As in lecture, let m be the total number of DS calls during our algorithm, n of which are $DS.$MAKE-SET. Suppose $|Q_1| = k_1, |Q_2| = k_2$. Let the size of our alphabet be σ.

Since each state is added to DS at most once, MAKE-SET is called at most $k_1 + k_2$ times. So $n \leq k_1 + k_2$. We also know that UNION (which is simply two FIND-SET followed by LINK) is called at most $\binom{k_1k_2}{2}$ times. Furthermore, the loop in from lines 2-8 runs at most k_1k_2 times since $(q, q') \in ST$ has $q \in Q_1, q' \in Q_2$, and each pair is added to ST at most once. This means FIND-SET runs for at most $2\sigma k_1k_2$ times. With this information, we can bound

$$\begin{aligned}\text{the number of calls to } \textsc{Link}, \textsc{Make-Set}, \textsc{Find-Set} &= m \\ &\leq (k_1 + k_2) + 3\binom{k_1k_2}{2} + 2\sigma k_1k_2) \\ &\in O(\sigma k_1k_2 + (k_1k_2)^2).\end{aligned}$$

(a) Circular linked list is not going to work well since we do FIND-SET often and it costs too much (linear in the size of the equivalence class). This is problematic when there are many indistinguishable states in the automaton.

(b) Linked list with back pointers costs too much to LINK. This is problematic when when the equivalence classes are large.

(c) Linked list with back pointers using union by weight is better than the above but unioning two large equivalence classes is still costly.

(d) Trees are not going to work well because it is possible to create a tree that is a linked list, so they suffer from the same problems as list based data structures.

(e) Trees with union by weight performs poorly since the tree can be tall and accessing the deepest state in the set can be problematic.

(f) Trees with path compression are trees with union-by-weight & path compression are efficient but they are outperformed by trees with union-by-rank & path compression (we know this from lecture). If we use the bounds we obtained above, we see that our algorithm costs in worst case $O(\sigma k_1k_2 + (k_1k_2)^2\alpha(n))$. For reasonable large n (i.e reasonably large $k_1 + k_2$), this is the algorithm costs $O(\sigma k_1k_2 + (k_1k_2)^2)$.