CS 4510 Automata and Complexity 8/29/2024

Homework 2: Nondeterminism

Prof. Jaeger *Due: 9/5/2024*

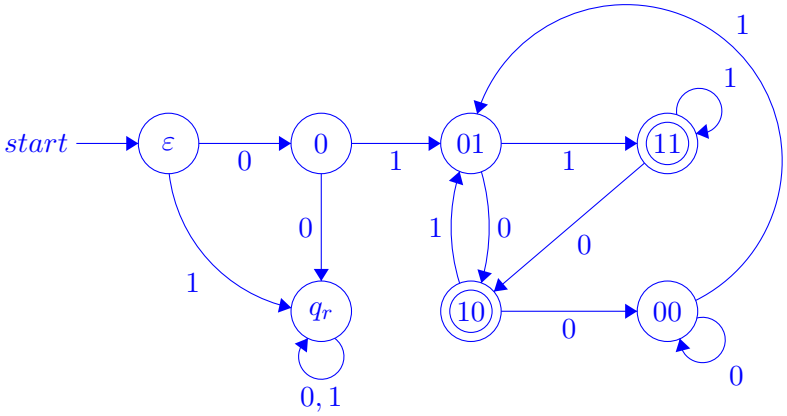
This assignment is due on **10:00 PM EST, Thursday, September 5, 2024**. You may turn it in one day late for no penalty or two days late for a 10% penalty. On-time submissions receive 3% extra credit. Note that a late submission means late feedback, which means less time to study before an exam.

You should submit a typeset or *neatly* written pdf on Gradescope. The grading TA should not have to struggle to read what you've written; if your handwriting is hard to decipher, you will be required to typeset your future assignments.

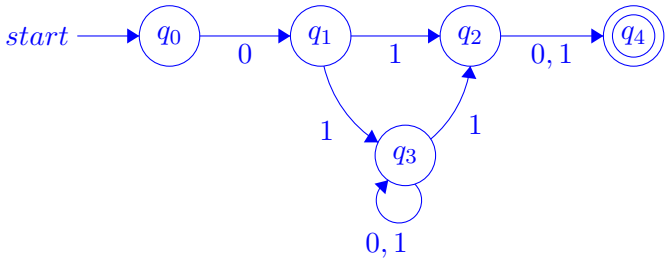
You may collaborate with other students, but any written work should be your own. Write the names of the students you work with on the top of your assignment.

1. **DFA/NFA/Regular Expression** (15 points)
   Let $L = \{x | x$ is a binary string which starts with a 01 and has a 1 as the 2^{nd} character from the end$\}$. Some examples of strings in L are 010, 0111, 0100111. Some strings not in L would be ε, 10, 0010, and 0100. A minimal finite automaton for a language is the finite automaton with the fewest number of states that recognizes that language.

   (a) (5 points) Give a minimal DFA for L.
   Solution Our DFA is as follows. It uses its first 3 states to restrict starting characters to 0,1 and the rest 4 to remember the last 2 characters.



   (b) (5 points) Give a minimal NFA for L.
   Solution Trying to make the minimal NFAs for the first (starting with 01) and the second language separately (1 second from the end) and then combine them is a reasonable approach. Another approach would be to make a regular expression for L and condense it as much as possible and then convert it to an NFA. We find that 2 states can be reused, the main tricky corner case is the need to accept 010 and 011.



(c) (5 points) Give a regular expression for L.

Solution L is very close to being a concatenation of two languages $01\Sigma^*$ (strings starting with 01) and $\Sigma^*1\Sigma$ (strings with a 1 second from the end). However, this would miss that possibility that the 1 in 01 is also the 1 second from the end.

It's probably easier to write a regular expression based on the above, but adding the second corner case. This gives

$$L = (01\Sigma^*1\Sigma) \cup 01\Sigma.$$

This could equivalently be written as $L = 01(\Sigma^*1\Sigma \cup \Sigma)$, or

$$L = 01(\Sigma^*1 \cup \varepsilon)\Sigma$$

from the distributive rule for $\circ$ over $\cup$. The final one can be used to construct an NFA for the same.

2. **Closure** (20 points) Let L be a language. Define $\text{INSERT}(L) = \{xay \in \Sigma^* \mid xy \in L, a \in \Sigma\}$. We add a single symbol anywhere to strings in L (including the beginning or end). Prove that if L is regular, then so is $\text{INSERT}(L)$ ...

(a) (10 points) ... using NFAs.

Solution

Since L is a regular language, there is a DFA D for L. The high-level idea is that we will create an NFA from two copies of D. From every state q in the copy of D we start in, we will create transitions to q in the second copy of D for every $a \in \Sigma$. There will not be any transitions back. We accept only in the accept states of the second copy.

To make this super formal, we now construct a new NFA N for our language $\text{INSERT}(L)$ as follows.

- $N.\Sigma = D.\Sigma$
- $N.Q = D.Q \times \{0, 1\}$
- $N.q_0 = (D.q_0, 0)$
- $N.F = D.F \times \{1\}$
- We define $N.\delta$ by $N.\delta((q, 0), a) = \{(D.\delta(q, a), 0), (q, 1)\}$ and $N.\delta((q, 1), a) = \{(D.\delta(q, a), 1)\}$ for all $q \in D.Q, a \in D.\Sigma$ and $N.\delta((q, b), \varepsilon) = \emptyset$.

Our first copy of D is captured by the states $(q, 0)$ and our second copy by $(q, 1)$. Since we have constructed an NFA for INSERT(L), it must be regular.
Starting with D being an NFA would work analogously, with

$$N.\delta((q,b),a) = \begin{cases} (D.\delta(q,a) \times \{0\}) \cup \{(q,1)\} & b = 0, a \neq \varepsilon \\ D.\delta(q,a) \times \{b\} & \text{otherwise} \end{cases}$$

(b) (10 points) . . . using regular expressions.

Solution

Let RE denote the set of regular expressions. We will recursively define a function $f : RE \to RE$ such that $L(f(R)) = \text{INSERT}(L(R))$ as shown below.[1] Formally, we should be doing an induction over the number of rules used to produce R so the first three bullets below are our base case and the latter three are the induction step in which we assume $f(R_1)$, $f(R_2)$ give the correct regular expression for inserting into R_1, R_2 as these regular expressions must have been created with fewer rule applications than R.

i. If $R = \emptyset$, then $f(R) = \emptyset$.
Explanation: This is trivially correct (as there are no strings that need to be inserted into).

ii. If $R = \varepsilon$, then $f(R) = \Sigma$.
Explanation: This is trivially correct (as inserting into the empty string just gives the character we inserted).

iii. If $R = b$ for some $b \in \Sigma$, then $f(R) = \Sigma b \cup b\Sigma$.
Explanation: This is trivially correct (as we can insert the character before or after b).

iv. If $R = R_1 \cup R_2$, then $f(R) = f(R_1) \cup f(R_2)$.
Explanation: Intuitively, strings in INSERT$(R_1 \cup R_2)$ are created by picking a string from either R_1 or R_2, then inserting a single character into it.
In more detail, let $R_1, R_2 \in RE$ and $R = R_1 \cup R_2$. Then

$$\begin{aligned} w \in \text{INSERT}(R) &\iff w = xay, xy \in R_1 \cup R_2, a \in \Sigma \\ &\iff w = xay, (xy \in R_1 \vee xy \in R_2), a \in \Sigma \\ &\iff (w = xay, xy \in R_1, a \in \Sigma) \vee (w = xay, xy \in R_2, a \in \Sigma) \\ &\iff w \in \text{INSERT}(R_1) \vee w \in \text{INSERT}(R_2) \\ &\iff w \in \text{INSERT}(R_1) \cup \text{INSERT}(R_2). \end{aligned}$$

So INSERT$(R) = \text{INSERT}(R_1) \cup \text{INSERT}(R_2)$, as desired.

v. If $R = R_1 \circ R_2$, then $f(R) = (f(R_1) \circ R_2) \cup (R_1 \circ f(R_2))$.[2]

[1] The right-hand side is applying INSERT to the language expressed by R. The left-hand side is the language expressed by the regular expression obtain by applying f to R. We often (including later in this proof) don't distinguish between a regular language R and the language $L(R)$ it describes. In that case we could have written this equation as $f(R) = \text{INSERT}(R)$.

[2] Recall typically we omit $\circ$ in regular expressions. Here we write it explicitly for clarity.

Explanation: Intuitively, strings in INSERT($R_1 \circ R_2$) are created by concatenating strings from R_1 and R_2, then inserting a single character into one of the two.
In more detail, let $R_1, R_2 \in RE$ and $R = R_1 \circ R_2$. If $w \in$ INSERT(R), then $w = xay$ for some $a \in \Sigma$ and $xy \in R_1 \circ R_2$. The last of these means $xy = vz$ for some $v \in R_1$ and $z \in R_2$.
If v contains x, then the a in w was inserted in the v part of vz so we can write $v = x'y'$ such that $w = xay = x'ay'z$. Then $x'ay' \in$ INSERT(R_1) and $z \in R_2$, so $w = x'ay'z \in$ INSERT(R_1) $\circ R_2$.
Otherwise z contains y, meaning the a in w was inserted in the z part of vz so we can write $z = x'y'$ such that $w = xay = vx'ay'$. Then $v \in R_1$ and $x'ay' \in$ INSERT(R_2), so $w = vx'ay' \in R_1 \circ$ INSERT(R_2).
Putting the cases together gives $w \in$ INSERT(R_1) $\circ R_2 \cup R_1 \circ$ INSERT(R_2). This gives INSERT(R) $\subseteq$ INSERT(R_1) $\circ R_2 \cup R_1 \circ$ INSERT(R_2). The other direction, $\supseteq$, is not too hard and basically follows the above logic backwards.

vi. If $R = R_1^*$, then $f(R) = \Sigma \cup R_1^* \circ f(R_1) \circ R_1^*$

Explanation: Intuitively, a string in INSERT(R) was either picked by inserting into ε[3] or by having a concatenation of one (or more) strings in R_1 and then inserting into one of them.
In more detail, let $R_1 \in RE$ and $R = R_1^*$. Recall $R_1^* = \cup_{i \geq 0} R_1^n$. If $w \in$ INSERT(R_1^*), then $w = xay$ for some $a \in \Sigma$ and $xy \in R_1^*$. If $xy = \varepsilon$, then $w \in \Sigma$. Otherwise, $xy \in R_1^n$ for some $n \geq 1$ and xy can be written as $r_1 r_2 \dots r_n$ where each r_i is in R_1. Then when viewing w as $w = xay$, we can view the a has having been inserted into of these r_i. We can write that one as $r_i = x'y'$ such that $w = r_1 \dots r_{i-1} x'ay' r_{i+1} \dots r_n$. Thus, $w \in R_1^* \circ$ INSERT(R_1) $\circ R_1^*$.
Putting the cases together gives $w \in \Sigma \cup R_1^* \circ$ INSERT(R_1) $\circ R_1^*$ and so This gives INSERT(R) $\subseteq \Sigma \cup R_1^* \circ$ INSERT(R_1) $\circ R_1^*$. The other direction, $\supseteq$, is not too hard and basically follows the above logic backwards.
Alternatively, note that from our concatenation analysis we have INSERT(R_1^n) $= \bigcup_{i=0}^{n-1} R_1^i \circ$ INSERT(R_1) $\circ R_1^{n-i-1}$ for any $n \geq 1$. Then we get

$$R = \bigcup_{n=0}^{\infty} R_1^n \implies \text{INSERT}(R) = \bigcup_{n \geq 0} \text{INSERT}(R_1^n) \qquad \text{by union analysis}$$

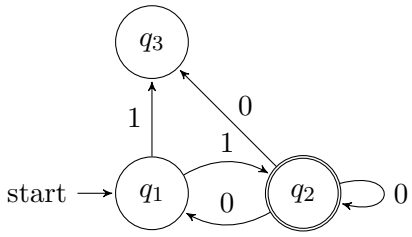
$$\implies \text{INSERT}(R) = \text{INSERT}(\varepsilon) \cup \bigcup_{n \geq 1} \bigcup_{i=0}^{n-1} R_1^i \circ \text{INSERT}(R_1) \circ R_1^{n-i-1}$$

$$\implies \text{INSERT}(R) = \Sigma \cup R_1^* \circ \text{INSERT}(R_1) \circ R_1^*$$

as desired.

3. **Regular Expression** (5 points) Give a regular expression for the language of the following NFA.

[3] When working with Kleene start *always* check if ε needs to be its own special case.



Solution

$$L = 1((01) \cup 0)^*$$

4. **Non-determinism** (10 points) Prove that if L is regular, then there exists an NFA N for L that only has one accept state.
Solution Let L be a regular language. Then there must be an DFA D that decides L. We create an NFA N with a single accept from D by adding a new state q_f which is the only accepting state and adding ε transitions from every previously accepting state to it.

   To make this super formal, N is defined as follows.

   - $N.\Sigma = D.\Sigma$
   - $N.Q = D.Q \cup \{q_f\}$
   - $N.q_0 = D.q_0$
   - $N.F = \{q_f\}$
   - $N.\delta : N.Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(N.Q)$ can be defined as an extension of $D.\delta$ by adding

     $$\forall q \in D.F, \quad N.\delta((q, \varepsilon)) = \{q_f\}$$

     That is, all the transitions of D remain and every state that was a final state in D will now have an epsilon transition to q_f. In full detail,

     $$N.\delta(q, x) = \begin{cases} \{D.\delta(q, x)\} & (q, x) \in D.Q \times \Sigma \\ \{q_f\} & (q, x) \in D.F \times \{\varepsilon\} \\ \{\} & \text{otherwise} \end{cases}$$

   N now has only 1 accepting state and decides the same language as D because:

   - If a string x is accepted by D, then x ended at a $q \in D.F$ in D. Then x can reach on $q \in D.F$ in N which has a ε transition to $q_f \in N.F$.
   - If a string x was not accepted by D, it ends at some $q \notin D.F$. The only complete path for x in N will bring it to the same q which means there is no transition to the only accept state. (If the path visited a state in $D.F$ earlier in x and we took one of the new ε transitions to q_f, we could not continue reading from x because q_f has no outgoing transitions.)