CS/ECE 374 A (Spring 2022)
Past HW3 Problems with Solutions

Problem Old.3.1: For the following languages in (a)–(b), draw an NFA that accepts them. Your automata should have a small number of states. Provide a short explanation of your solution.

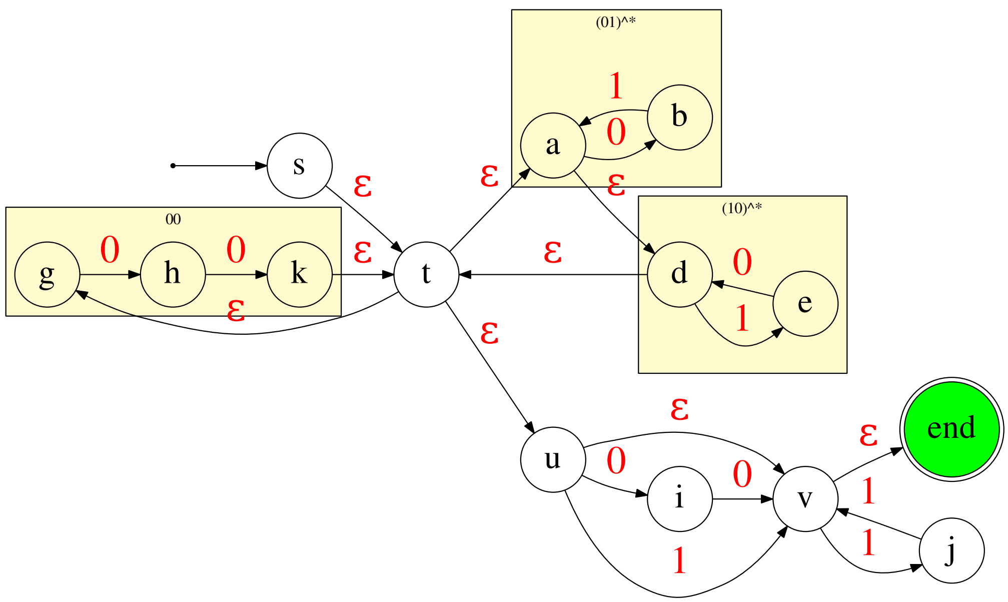
(a) $\big((01)^*(10)^* + 00\big)^* \cdot (1 + 00 + \varepsilon) \cdot (11)^*$.

(b) All strings in $\{0, 1\}^*$ such that the last symbol is the same as the third last symbol. (Example: 1100101 is in the language, since the last and the third last symbol are 1.)

(c) Use the subset (i.e., power set) construction to convert your NFA from (b) to a DFA. You may omit unreachable states.

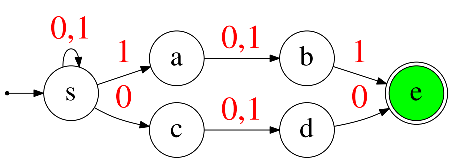
Solution:

(a)



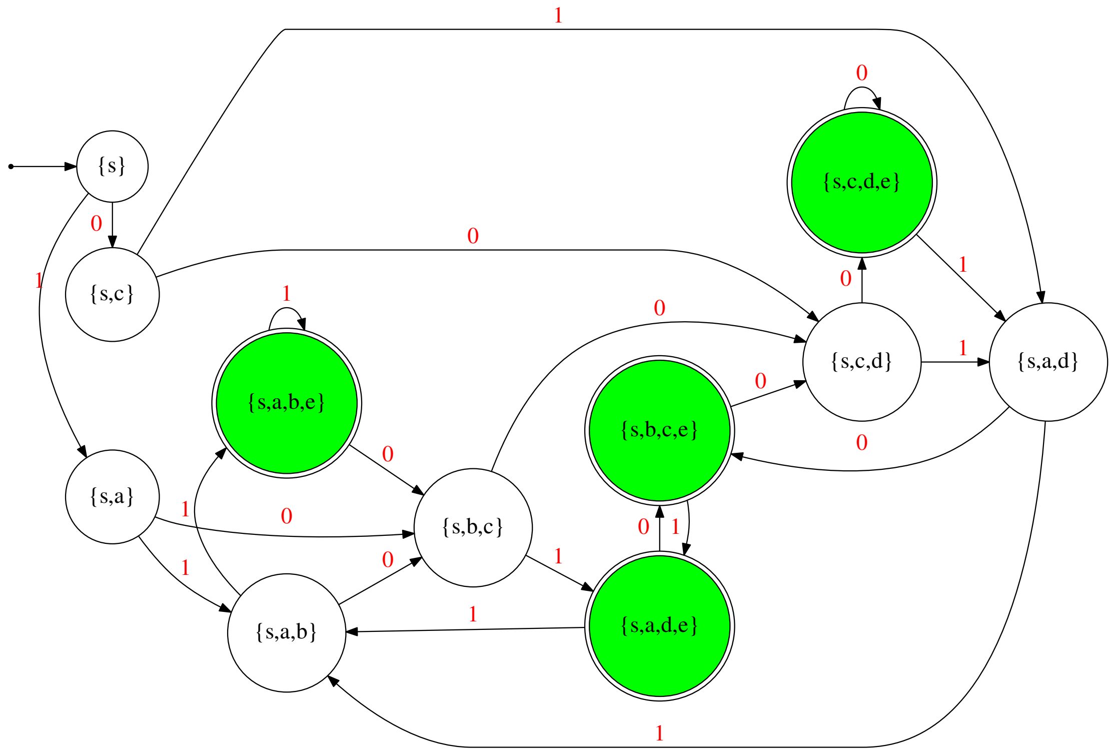
We apply the recursive algorithm from class (with some shortcuts taken, although further shortcuts could still be made). States a,b,c,d,g,h,k,t deal with $\big((01)^*(10)^* + 00\big)^*$. States u,i,v,j deal with $(1 + 00 + \varepsilon) \cdot (11)^*$.

(b)



We use nondeterminism to guess when we have reached the 3rd-to-last symbol. If it is a 1, we follow the path s,a,b,e to ensure that the last 3 symbols are in $1(0 + 1)1$. If it is a 0, we follow the path s,c,d,e to ensure that the last 3 symbols are in $0(0 + 1)0$.

(c)



Problem Old.3.2: Given $L \subseteq \{0, 1\}^*$, define $even_0(L)$ to be the set of all strings in $\{0, 1\}^*$ that can be obtained by taking a string in L and inserting an even number of 0's (anywhere in the string). Similarly, define $odd_0(L)$ to be the set of all strings x in $\{0, 1\}^*$ that can be obtained by taking a string in L and inserting an odd number of 0's.

(Example: if $01101 \in L$, then $01010000100 \in even_0(L)$.)

(Another example: if L is 1^*, then $even_0(L)$ can be described by the regular expression $(1^*01^*0)^*1^*$.)

Prove that if $L \subseteq \{0, 1\}^*$ is regular, then $even_0(L)$ and $odd_0(L)$ are regular. Specifically, given a regular expression r, describe a recursive algorithm to construct regular expressions for $even_0(L(r))$ and $odd_0(L(r))$.

Solution:

Algorithm $\text{EVEN}_0(r)$:

1. if $r = \emptyset$ then return $\emptyset$
2. if $r = \varepsilon$ then return $(00)^*$
3. if $r = 0$ then return $0(00)^*$
4. if $r = 1$ then return $(00)^*1(00)^* + 0(00)^*10(00)^*$
5. if $r = r_1 + r_2$ then return $\text{EVEN}_0(r_1) + \text{EVEN}_0(r_2)$

6. if $r = r_1 r_2$ then return $\textsc{even}_0(r_1) \cdot \textsc{even}_0(r_2) + \textsc{odd}_0(r_1) \cdot \textsc{odd}_0(r_2)$
7. if $r = (r_1)^*$ then return

$$(00)^* + \textsc{even}_0(r_1)^* \cdot (\textsc{odd}_0(r_1) \cdot \textsc{even}_0(r_1)^* \cdot \textsc{odd}_0(r_1) \cdot \textsc{even}_0(r_1)^*)^*$$

Algorithm $\textsc{odd}_0(r)$:

1. if $r = \emptyset$ then return $\emptyset$
2. if $r = \varepsilon$ then return $0(00)^*$
3. if $r = 0$ then return $00(00)^*$
4. if $r = 1$ then return $0(00)^*1(00)^* + (00)^*10(00)^*$
5. if $r = r_1 + r_2$ then return $\textsc{odd}_0(r_1) + \textsc{odd}_0(r_2)$
6. if $r = r_1 r_2$ then return $\textsc{odd}_0(r_1) \cdot \textsc{even}_0(r_2) + \textsc{even}_0(r_1) \cdot \textsc{odd}_0(r_2)$
7. if $r = (r_1)^*$ then return

$$0(00)^* + \textsc{even}_0(r_1)^* \cdot \textsc{odd}_0(r_1) \cdot \textsc{even}_0(r_1)^* \cdot (\textsc{odd}_0(r_1) \cdot \textsc{even}_0(r_1)^* \cdot \textsc{odd}_0(r_1) \cdot \textsc{even}_0(r_1)^*)^*$$

Justification of Algorithm $\textsc{even}_0(r)$:

- Lines 1–3 and 5 are self-explanatory.
- In line 4, for $r = 1$, we want all strings with one 1 and an even number of 0's. There are two cases: there are an even number of 0's before the 1 and even number of 0's after the 1, or there are an odd number of 0's before the 1 and odd number of 0's after the 1. This gives $(00)^*1(00)^* + 0(00)^*10(00)^*$,
- In line 6, for $r = r_1 r_2$, there are two cases for strings in $\mathit{even}_0(L(r_1)L(r_2))$: we can insert an even number of 0's to a string in $L(r_1)$ and an even number of 0's to a string in $L(r_2)$, or we can insert an odd number of 0's to a string in $L(r_1)$ and an odd number of 0's to a string in $L(r_2)$. This gives $\textsc{even}_0(r_1) \cdot \textsc{even}_0(r_2) + \textsc{odd}_0(r_1) \cdot \textsc{odd}_0(r_2)$.
- In line 7, for $r = (r_1)^*$, a string in $\mathit{even}_0(L(r_1)^*)$ can be divided into blocks, where each block is obtained by inserting either an even or an odd number of 0's to a string in $L(r_1)$, where the number of blocks of the latter "odd type" is even. This gives $\textsc{even}_0(r_1)^* \cdot (\textsc{odd}_0(r_1) \cdot \textsc{even}_0(r_1)^* \cdot \textsc{odd}_0(r_1) \cdot \textsc{even}_0(r_1)^*)^*$. The only remaining case is when we just insert an even number of 0's to the empty string; this is $(00)^*$.

Justification of Algorithm $\textsc{odd}_0(r)$ is similar.

Problem Old.3.3: Let L be an arbitrary regular language. Prove that the language $\mathit{half}(L) := \{w : ww \in L\}$ is also regular.

Solution: Let $M = (\Sigma, Q, s, A, \delta)$ be an arbitrary DFA that accepts L. We define a new NFA $M' = (\Sigma, Q', s', A', \delta')$ with ε-transitions that accepts $\mathit{half}(L)$, as follows:

$$\begin{aligned} Q' &= (Q \times Q \times Q) \cup \{s'\} \\ &s' \text{ is an explicit state in } Q' \\ A' &= \{(h, h, q) : h \in Q \text{ and } q \in A\} \\ \delta'(s', \varepsilon) &= \{(s, h, h) : h \in Q\} \\ \delta'((p, h, q), a) &= \{(\delta(p, a), h, \delta(q, a))\} \end{aligned}$$

Explanation: M' reads its input string w and simulates M reading the input string ww. Specifically, M' simultaneously simulates two copies of M, one reading the left half of ww starting at the usual start state s, and the other reading the right half of ww starting at some intermediate state h.

- The new start state s' non-deterministically guesses the "halfway" state $h = \delta^*(s, w)$ without reading any input; this is the only non-determinism in M'.
- State (p, h, q) means the following:
  - The left copy of M (which started at state s) is now in state p.
  - The initial guess for the halfway state is h.
  - The right copy of M (which started at state h) is now in state q.
- M' accepts if and only if the left copy of M ends at state h (so the initial non-deterministic guess $h = \delta^*(s, w)$ was correct) and the right copy of M ends in an accepting state.

Problem Old.3.4: For a string $x \in \{0,1\}^*$, let x^F denote the string obtained by changing all 0's to 1's and all 1's to 0's in x.

Given a language L over the alphabet $\{0, 1\}$, define

$$\text{FLIP-SUBSTR}(L) = \{uv^F w : \; uvw \in L, \; u, v, w \in \{0,1\}^*\}.$$

Prove that if L is regular, then FLIP-SUBSTR(L) is regular.

(For example, $(1011)^F = 0100$. If $1011011 \in L$, then $1000111 = 10(110)^F 11 \in$ FLIP-SUBSTR(L). For another example, FLIP-SUBSTR(0^*1^*) $= 0^*1^*0^*1^*$.)

[*Hint*: given an NFA (or DFA) for L, construct an NFA for FLIP-SUBSTR(L). Give a formal description of your construction. Provide an explanation of how your NFA works, including the meaning of each state. A formal proof of correctness of your NFA is not required.]

Solution: Let L be a regular language over $\Sigma = \{0, 1\}$. By Kleene's theorem, L is accepted by some DFA $M = (\Sigma, Q, s, A, \delta)$. We construct an NFA $M' = (\Sigma, Q', s', A', \delta')$ accepting

FLIP-SUBSTR(L) (which would imply that FLIP-SUBSTR(L) is regular by Kleene's theorem). The construction is as follows:

$$
\begin{array}{rcll}
Q' & = & Q \times \{\mathit{before}, \mathit{middle}, \mathit{after}\} & \\
s' & = & (s, \mathit{before}) & \\
A' & = & \{(q, \mathit{after}) : q \in A\} & \\
\delta'((q, \mathit{before}), a) & = & (\delta(q, a), \mathit{before}) & \forall q \in Q,\ a \in \Sigma \\
\delta'((q, \mathit{before}), \varepsilon) & = & (q, \mathit{middle}) & \forall q \in Q \\
\delta'((q, \mathit{middle}), a) & = & (\delta(q, a^F), \mathit{middle}) & \forall q \in Q,\ a \in \Sigma \\
\delta'((q, \mathit{middle}), \varepsilon) & = & (q, \mathit{after}) & \forall q \in Q \\
\delta'((q, \mathit{after}), a) & = & (\delta(q, a), \mathit{after}) & \forall q \in Q,\ a \in \Sigma
\end{array}
$$

(All other unspecified entries of δ' are $\emptyset$.)

Explanation: The idea is to divide the process into three phases: *before* (reading the prefix u), *middle* (reading the substring v that needs to be flipped), and *after* (reading the suffix w). We use nondeterminism (ε-transitions) to guess when to switch from the *before* phase to the *middle* phase, and when to switch from the *middle* phase to the *after* phase. At the same time, we simulate M on the string uv^Fw. (Note that the definition of FLIP-SUBSTR(L) is equivalent to $\{uvw : uv^Fw \in L\}$.)

Meaning of states in M':

- M' may be in state (q, before) after reading input x iff M may be in state q after reading input x.
- M' may be in state (q, middle) after reading input x iff M may be in state q after reading input uv^F for some strings u and v with $x = uv$.
- M' may be in state (q, after) after reading input x iff M may be in state q after reading input uv^Fw for some strings u, v, w with $x = uvw$.