

Problem Set 1 Solutions

Instructor: Daniele Micciancio

Jan. 12, 2000

Problem 1

Consider the DFAs M_1 and M_2 shown in figure 1. Build a DFA M accepting the intersection of the languages $L(M_1)$ and $L(M_2)$ using the cartesian product construction described in the second lecture. Give both a state diagram and the formal description of the automaton M .

Figure 1: Automata M_1 and M_2 **Solution:**

Given the DFAs $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$, we construct a DFA $M = (Q, \Sigma, \delta, s, F)$ accepting the intersection of the language $L(M_1)$ and $L(M_2)$ as follows.

The set of states Q is a cartesian product of Q_1 and Q_2 .

$$Q = \{(q_1, p_1), (q_1, p_2), (q_2, p_1), (q_2, p_2)\}$$

$$\Sigma = \{a, b\}$$

$$s = (q_1, p_1)$$

$$F = \{(q_2, p_1)\}$$

δ	(q_1, p_1)	(q_1, p_2)	(q_2, p_1)	(q_2, p_2)
a	(q_1, p_1)	(q_1, p_1)	(q_2, p_1)	(q_2, p_1)
b	(q_2, p_2)	(q_2, p_2)	(q_2, p_2)	(q_2, p_2)

The state diagram for the above DFA is given in Figure 2.

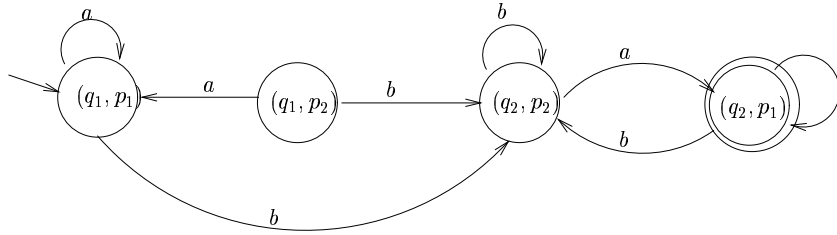


Figure 2: State diagram of the DFA M

Problem 2

In many programming languages (C,C++,Java,perl, etc.) strings are typically specified as sequences of characters enclosed in double quotes (e.g. "ntnt" stands for the string $ntnt$). In order to specify strings containing the double quote character, the symbol " can be quoted inserting a backslash character right before it. For example, "ttn\"ntt" denotes the 6-character string $ttn"ntt$. Other "escape sequences" with special meaning are $\backslash n$ (new line), $\backslash t$ (tab), etc. Backslash characters can also be inserted using the escape sequence $\backslash \backslash$. Give a DFA accepting the language of all words over the alphabet $\Sigma = \{n, t, \backslash, "\}$ corresponding to valid C-style string constants, as informally described above. (Examples of words that should be accepted by the automaton are $ttn\backslash t\backslash "$ or $t\backslash t$ but not $tnnt$ or $\backslash .$) Give both the state diagram of the DFA and the table corresponding to its transition function.

Solution:

In order to solve this problem, we need to carefully analyze what a valid string constant is.

- They must start (and end) with a double quote;
- After reading the first double quote, if a backslash (\backslash) is found, any following character must belong to the "escape sequence", which ends just after it;
- reading n 's and t 's does not change the state of "being inside of the double quotes" and does not implies anything about the following character.

The state diagram for the DFA M that recognizes this language is shown in Figure 3.

Notice that q_5 is a rejecting state (or "black hole"). The following table describes the transition function of M .

	"	\	n	t
q_1	q_2	q_5	q_5	q_5
q_2	q_4	q_3	q_2	q_2
q_3	q_2	q_2	q_2	q_2
q_4	q_5	q_5	q_5	q_5
q_5	q_5	q_5	q_5	q_5

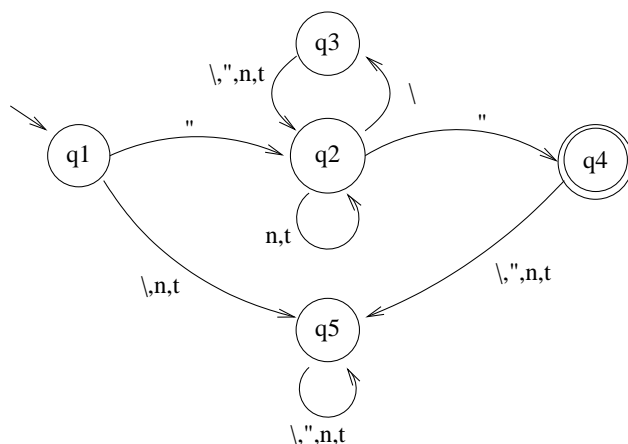


Figure 3: State diagram for M

For completeness' sake, we fully describe DFA M as $M = (Q, \Sigma, \delta, s, F)$, where $Q = \{q_1, q_2, q_3, q_4, q_5\}$, $\Sigma = \{\tau, n, \backslash, \text{"}\}$, δ as described in the previous table, $s = q_1$ and $F = \{q_4\}$.

Problem 3

Given a word $w = w_1w_2 \dots w_n$ (where $w_i \in \Sigma$ for all i), let the even and odd parts of w be the words $even(w) = w_2w_4w_6 \dots$ and $odd(w) = w_1w_3w_5 \dots$ obtained taking only the characters at even (resp. odd) positions. Notice: the even part of a single character word, and the even or odd part of the empty word ϵ are all equal to the empty word ϵ . [Informal explanation: you can think of the word w as a sequence of single character messages sent by two alternating parties (Jee Hea and Alejandro) during a conversation. Then $even(w)$ is the sequence of messages transmitted by Alejandro, while $odd(w)$ is the sequence of messages sent by Jee Hea.] These operations can be extended to languages taking the even (resp. odd) part of every word in the language: $even(L) = \{even(w) \mid w \in L\}$ and $odd(L) = \{odd(w) \mid w \in L\}$. For example, $even(\{askwr, eityqe\}) = \{sw, iye\}$ and $odd(\{askwr, eityqe\}) = \{akr, etq\}$.

Prove that if L is a regular language, then also $odd(L)$ and $even(L)$ are regular. (Hint: you can use non-deterministic automata.)

Optional (you don't have to write anything for this part): start thinking about the opposite problem, i.e., assuming that $even(L)$ and $odd(L)$ are both regular, can you conclude that L is also regular?

Solution:

We want to show that if L is a regular language, then $odd(L)$ and $even(L)$ are

regular. Since L is regular, we have a DFA $M = (Q, \Sigma, \delta, s, F)$ that recognizes the language L . In order to show that $odd(L)$ and $even(L)$ are regular, we need to show that there exists a finite state automaton (NFA) for each language $odd(L)$ and $even(L)$. (We construct NFAs here because it is easier and simpler in this case.) So, given the DFA M , we construct an NFA for each language $odd(L)$ and $even(L)$ using the given DFA M . Let $N_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $N_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ be the NFAs recognizing $odd(L)$ and $even(L)$, respectively. (Note that the set of alphabets remains the same in both automata). The hardest problem in constructing these NFAs is probably figuring out the transition functions using the transition functions of the original DFA.

Let us start with constructing the NFA N_1 for $odd(L)$. You are given a DFA M recognizing some language L . In order to build an automaton that recognizes every odd symbols in the strings accepted by the original DFA M , we need to make use of the transition function δ of M . For example, let M be a DFA that accepts a string $abaab \in L$. The NFA N_1 that recognizes $odd(L)$ should accept aab skipping the symbols b, a in the even positions. How do we skip the even position states? Suppose in processing the first symbol a , the original transition moves from the start state s to a state r_1 , and processing the second symbol b , it moves from r_1 to r_2 . In the new transition for N_1 , given the first input symbol a at the start state s , we can move to r_2 instead of r_1 . A critical thing to note is that it does not matter what the symbols at the even positions in the original input are. For example, even if both strings aba and aaa are in L , only aa will be in $odd(L)$. Hence, when we skip the second transition, we need to do so on any input symbol, not just on one particular symbol. We can describe the transition function more formally as follows:

For $q \in Q$ and $a \in \Sigma$,

$$\delta_1(q, a) \in \{p \mid p = \delta(\delta(q, a), b) \text{ for some } b \in \Sigma\}$$

There is one more thing to consider regarding the accepting state transitions. When the transition $\delta(q, a)$ happens to return an accepting state, we need to accept the string if the symbol “a” is the last input symbol in the input string. In this case, we should not just return the next state following an additional transition on any input symbol, and we should not return the same accepting state $\delta(q, a)$ because in case “a” is not the last symbol, we need to follow the normal transition function as shown above. So, we create an additional accepting state f for this case and return the new accepting state in addition to the state returned by the transition function shown above. Since we do not know whether the input symbol is the last one or there are more input symbols following it, we need to consider this as a separate case as follows:

For $q \in Q$ and $a \in \Sigma$, if $\delta(q, a)$ is an accept state (i.e. $\delta(q, a) \in F$), then

$$\delta_1(q, a) \in \{f\} \cup \{p \mid p = \delta(\delta(q, a), b) \text{ for some } b \in \Sigma\},$$

where f is an additional accepting state.

Since the transition function δ_1 maps a state and an input symbol into a set of states, we can see that the corresponding automaton is nondeterministic automaton (NFA). Note that there is no transition defined for the new accepting state f for any input symbol (i.e. the accepting state f does not have outgoing arrow from it on any symbol).

Combining the above transition functions and other definitions for the NFAs, we have the following formal definitions of the NFA $N_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ that recognizes $odd(L)$, given the DFA $M = (Q, \Sigma, \delta, s, F)$ that recognizes L .

$$\begin{aligned} Q_1 &= Q \cup \{f\} \\ s_1 &= s \\ F_1 &= F \cup \{f\} \end{aligned}$$

For each $q \in Q$ and $a \in \Sigma$,

$$\delta_1(q, a) \in \begin{cases} \{f\} \cup \{p \mid p = \delta(\delta(q, a), b) \text{ for some } b \in \Sigma\} & \text{if } \delta(q, a) \in F \\ \{p \mid p = \delta(\delta(q, a), b) \text{ for some } b \in \Sigma\} & \text{otherwise} \end{cases}$$

In order to build an NFA N_2 for $even(L)$, we can make use of the NFA N_1 that recognizes $odd(L)$ (especially its transition function). For example, suppose the original language accepts the string “ababa”. If we can modify the NFA N_1 built for $odd(L)$ such that it skips the first symbol “a”, we will be able to accept $bb \in even(L)$. (Note that $even(ababa) = odd(baba) = bb$.) We do so by adding a new start state s' and define a new ϵ transition function from it to be as follows:

For the new start state s' ,

$$\delta_2(s', \epsilon) \in \{p \mid p = \delta(s, a) \text{ for some } a \in \Sigma\},$$

where s is the start state of the original DFA M (which is the same the start state of N_1). For the other cases, we can define the transition function in the same way as the one δ_1 defined for N_1 .

One thing to be careful about adding a new start state is that it should be included in the set of accepting states if the original start state was an accepting state. If the empty string ϵ was in the language L , this will enable the new automaton to accept an empty string ϵ whose even string is also an empty string.

Given the DFA $M = (Q, \Sigma, \delta, s, F)$ that recognizes L , the formal definition for the NFA $N_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ that recognizes $even(L)$ is shown below:

$$\begin{aligned} Q_2 &= Q \cup \{s', f\} \\ s_2 &= s' \\ F_2 &= \begin{cases} F \cup \{f, s'\} & \text{if } s \in F \\ F \cup \{f\} & \text{otherwise} \end{cases} \end{aligned}$$

For the new start state s' ,

$$\delta_2(s', \epsilon) \in \{p \mid p = \delta(s, a) \text{ for some } a \in \Sigma\},$$

and for each $q \in Q$ and $a \in \Sigma$,

$$\delta_2(q, a) \in \begin{cases} \{f\} \cup \{p \mid p = \delta(\delta(q, a), b) \text{ for some } b \in \Sigma\} & \text{if } \delta(q, a) \in F \\ \{p \mid p = \delta(\delta(q, a), b) \text{ for some } b \in \Sigma\} & \text{otherwise} \end{cases}$$

Problem 4

Consider again the two deterministic automata from figure 1. Give the state diagram and formal description of an NFA accepting the concatenation of the two languages. Transform the NFA into an equivalent DFA using the transformation described in the book. (Just draw the state diagram of the DFA. You only need to draw the reachable states, but please label the states of the DFA with sets of states of the NFA so that the relation between the two automata is clear.)

Solution:

Let $N = (Q, \Sigma, \delta, s, F)$ be the NFA describing the concatenation of the two languages accepted by the DFAs from figure 1. The state diagram for this NFA follows simply from “concatenating” the two DFAs using an ϵ -transition (and making state q_2 non-final). The diagram is shown in Figure 4.

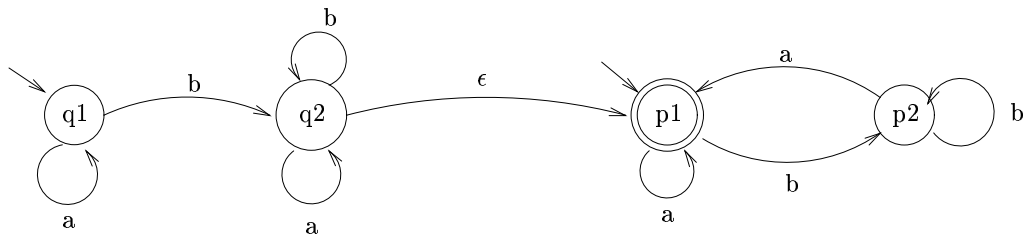


Figure 4: State diagram for NFA N

The formal description of $N = (Q, \Sigma, \delta, s, F)$ is given as follows:

$$Q = \{q_1, q_2, p_1, p_2\}$$

$$\Sigma = \{a, b\}$$

$$s = q_1$$

$$F = \{p_1\}$$

The transition function δ is given as

	a	b	ϵ
q_1	$\{q_1\}$	$\{q_2\}$	\emptyset
q_2	$\{q_2\}$	$\{q_2\}$	$\{p_1\}$
p_1	$\{p_1\}$	$\{p_2\}$	\emptyset
p_2	$\{p_1\}$	$\{p_2\}$	\emptyset

We now proceed to transform NFA N into a new DFA $M = (Q', \Sigma', \delta', s', F')$. The description of DFA M is as follows:

$$\begin{aligned}
 Q' &= \mathcal{P}(Q) = \{ \emptyset, \{q_1\}, \{q_2\}, \dots, \{q_2, p_1, p_2\}, Q \} \\
 \Sigma' &= \Sigma \\
 s' &= \{q_1\} \\
 F' &= \{ \{p_1\}, \{q_1, p_1\}, \{q_2, p_1\}, \{p_1, p_2\}, \{q_1, q_2, p_1\}, \{q_2, p_1, p_2\}, \{q_1, p_1, p_2\} \}
 \end{aligned}$$

The transition function δ' is given as

	a	b
$\{q_1\}$	$\{q_1\}$	$\{q_2, p_1\}$
$\{q_2\}$	$\{q_2, p_1\}$	$\{q_2, p_1\}$
$\{q_2, p_1\}$	$\{q_2, p_1\}$	$\{q_2, p_1, p_2\}$
$\{q_2, p_1, p_2\}$	$\{q_2, p_1\}$	$\{q_2, p_1, p_2\}$

Note that the value of the transition function δ' on all the other states in Q' is not important since they are unreachable from s' , and thus, they could be removed. Finally, the corresponding state diagram, after removing the unnecessary states, is shown in Figure 5.

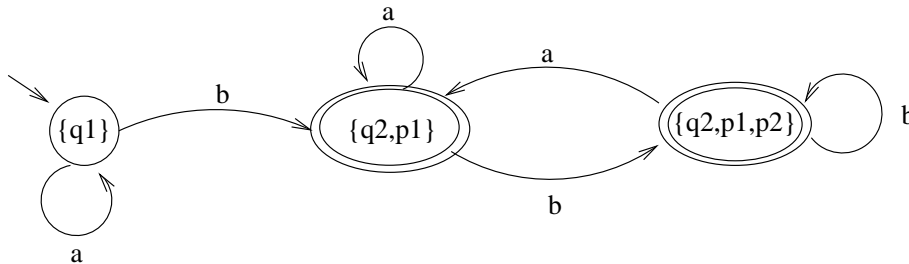


Figure 5: State diagram of DFA M

Problem 5

Transform the following regular expression into an equivalent (non-deterministic) finite state automaton using the procedure described in class:

$$(a \cup b)^* ab(bab \cup a^*)^* bb^*$$

Solution:

Figure 6 depicts the final NFA resulting from the transformation from the regular expression given in the problem. The diagram can be broken into four parts, showing the NFAs for $(a \cup b)^*$, ab , $(bab \cup a^*)^*$, and bb^* , separately. They are labeled in the diagram accordingly. For each of the four NFAs, the accepting states are q_0 , q_7 , q_8 , and q_{17} where the ϵ transition comes out of each of the four NFA. When combined (concatenated) together to get the final answer, the first three accepting states are no longer marked accepting, leaving only q_{17} as the accepting state for the final NFA transformed from the given regular expression. You can also simplify the diagram by removing some unnecessary ϵ transitions. The closure (“*”) operation is transformed in the way shown in class. Note that it is different from the one shown in the book. You can use either method because they are equivalent.

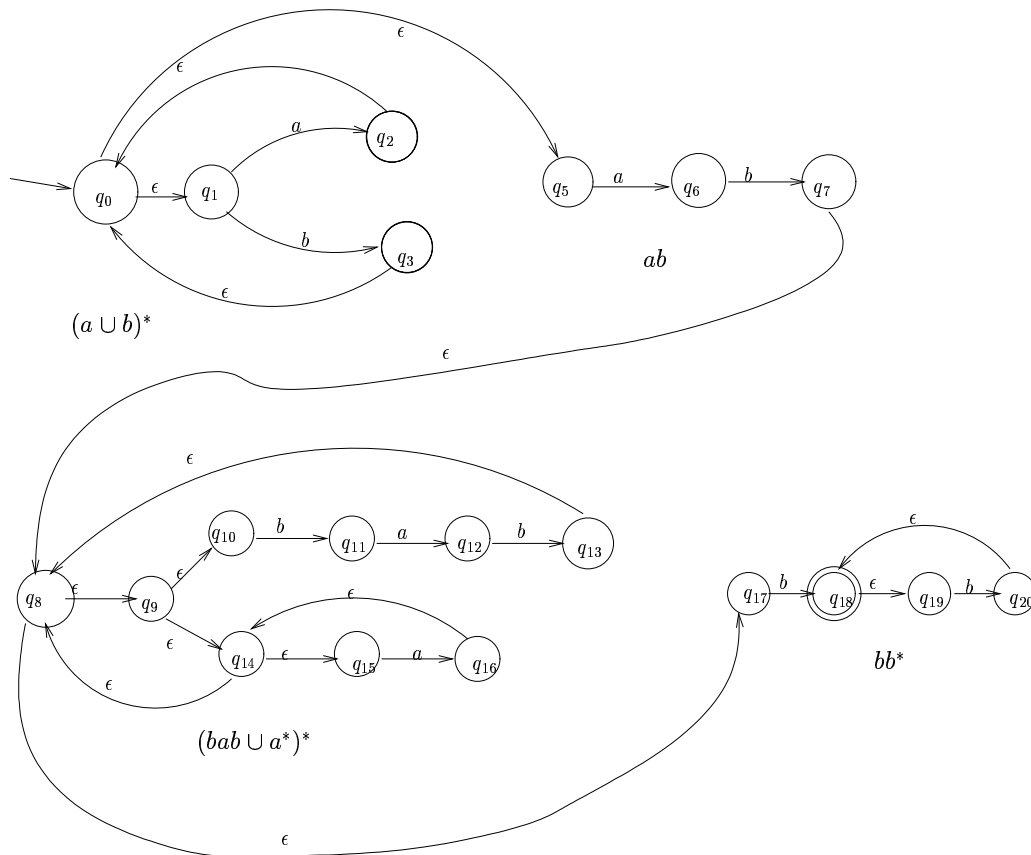


Figure 6: An NFA equivalent to $(a \cup b)^* ab(bab \cup a^*)^* bb^*$

Problem 6

Transform the following automaton into an equivalent regular expression using the procedure studied in class:

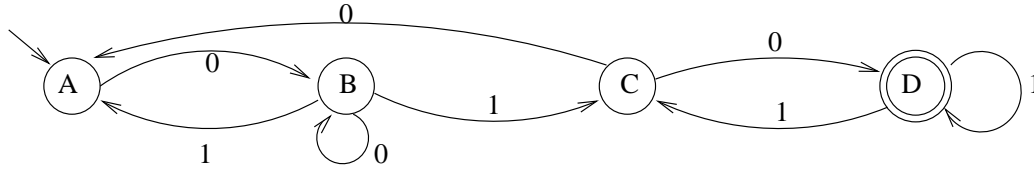


Figure 7: A non-deterministic finite state automaton

Solution:

We first transform the given automaton (say M) into a GNFA G . In order to do this, we add a new start state s and a new accept state f (accordingly, states A and B are no longer start and accept states). Thus, we get a GNFA like the one shown in Figure 8.

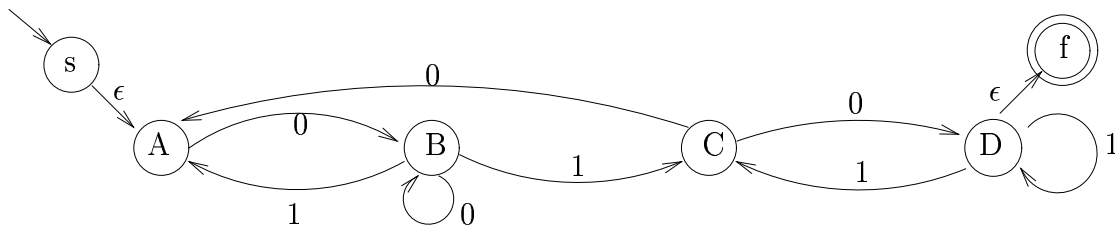


Figure 8: State diagram of GNFA G equivalent to M

We proceed to remove state B, C, A, D (in that order). The sequence of resulting GNFA is shown in Figures 9, 10, 11, and 12.

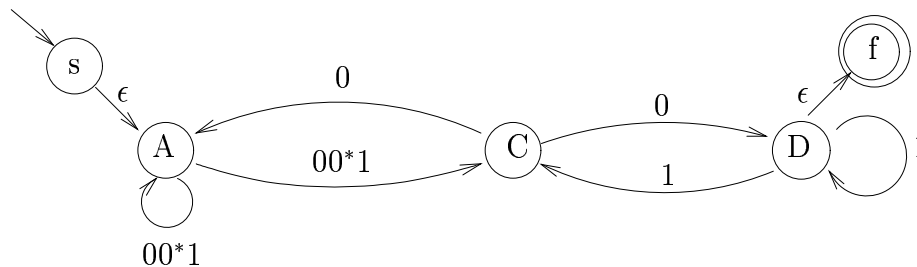


Figure 9: State diagram of GNFA G after removing B

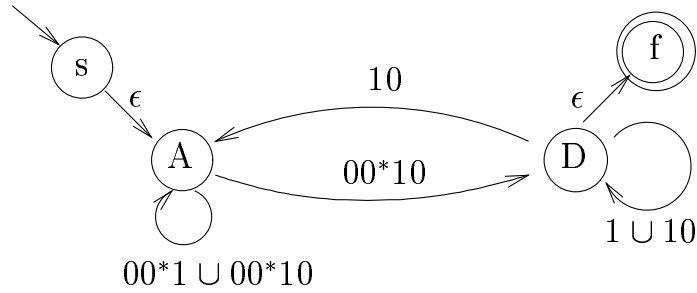


Figure 10: State diagram of GNFA G after removing C

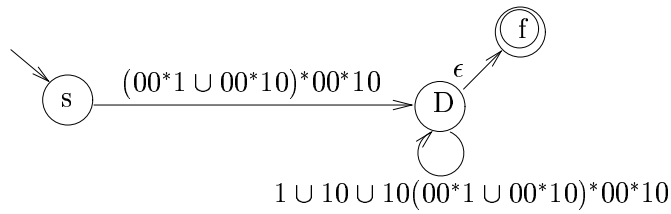


Figure 11: State diagram of GNFA G after removing A

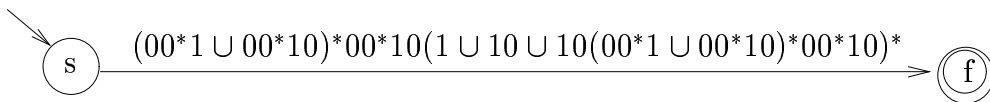


Figure 12: State diagram of GNFA G after removing D

Hence, the equivalent regular expression equivalent to the given NFA is

$$(00^*1 \cup 00^*10)^*00^*10(1 \cup 10 \cup 10(00^*1 \cup 00^*10)^*00^*10)^*$$
