4CS4-6:Theory of Computation(Closure on Reg. Langs., regex to NDFA, DFA to regex)

Lecture 08: Feb. 13 & 14, 2020

Prof. K.R. Chowdhary *: Professor of CS*

8.1 Closure properties of Regular languages

We know that regular languages are expressible as Sets. The set operations of union and concatenation correspond to language operations. Hence regular languages have closure properties under these operations. However, there is no operation on sets corresponding to *Kleene Star* operation on Regular Sets. In fact, this can be considered as repeated concatenation operation.

This section explores the closure properties of regular languages corresponding to other operations in sets, like *intersections*, *complementation*, *difference*, and *symmetric difference*.

Theorem 8.1 *The class of languages accepted by Finite Automata are closed under the operation of (i) Complementation, and (ii) Intersection.*

Proof: *I. Complementation:* Let the $M = (Q, \Sigma, \delta, s_0, F)$ be the finite automaton and $L = L(M)$ be the language accepted by it. The complement of M is $\overline{M}$. Let $\overline{M}$ be the FA that accepts the language $\overline{L}$. Obviously $\overline{M}$ would accept languages that are not accepted by M and vice-versa. Therefore, the final states in $\overline{M}$ are those that are non-final states in M and, what are non-final states in $\overline{M}$ are final states in M. In other words, the final states and non-final states are interchanged in M and $\overline{M}$. However, the total states remain the same in both the automata. Thus, $\overline{L} = L(\overline{M}) = \Sigma^* - L(M)$, and $\overline{M} = (Q, \Sigma, \delta', s_0, Q - F)$.

Since $\overline{M}$ is a finite automaton, the $\overline{L}$ language accepted by this is regular. This shows that complement of a regular language is also regular, hence the regular languages are closed under the operation of complementation.

II. Intersection: Let us assume that intersection of regular languages L_1 and L_2 is language $L = L_1 \cap L_2$. We want to determine whether L is regular. Using De Morgan's theorem we rewrite the expression for intersection of L_1 and L_2 as,

$$L_1 \cap L_2 = \overline{(\overline{L_1} \cup \overline{L_2})} \tag{8.1}$$

This means that, the language $L = L_1 \cap L_2$ consists all words that are in $\overline{(\overline{L_1} \cup \overline{L_2})}$. In other words L consists those words that are not in $(\overline{L_1} \cup \overline{L_2})$, that is they are neither in $\overline{L_1}$

nor in $\overline{L_2}$. Since L_1 and L_2 are regular, hence $\overline{L_1}$ and $\overline{L_2}$ are also regular. In addition, their union $(\overline{L_2} \cup \overline{L_2})$ is regular and its complement $(\overline{\overline{L_1} \cup \overline{L_2}})$ too. Hence, the intersection of two regular languages results to a regular language. Therefore, the regular languages are closed under the operation of intersection.

Regarding the states of automation recognizing the language $(\overline{\overline{L_1} \cup \overline{L_2}})$, the following proof clarifies it: Let $L_1 = L(M_1)$, and $L_2 = L(M_2)$, where $M_1 = (Q_1, \Sigma_1, \delta_1, s_1, F_1)$, and $M_2 = (Q_2, \Sigma_2, \delta_2, s_2, F_2)$. A combined automation out of M_1 and M_2 can be constructed as $M = (Q, \Sigma, \delta, (s_1, s_2), F)$, where $Q = Q_1 \times Q_2 = \{(q_i, q_j) \mid q_i \in Q_1, q_j \in Q_2\}$. The transition function δ is defined in such a way that M is in state (q_i, q_j) whenever the M_1 is in state q_i and M_2 is in state q_j. Thus, the state transitions in M are defined as,

$$\delta((q_i, qj_), a) = (q_k, q_l)$$

If and only if,

$$\begin{aligned} \delta_1(q_i, a) &= q_k, \text{ and,} \\ \delta_2(q_j, a) &= q_l. \end{aligned}$$

Similarly F is defined as the set of all the pairs (q_i, q_j), such that $q_i \in F_1$, and $q_j \in F_2$.

Example 8.2 *Consider a regular expression $r = (ab + aba)^*$, and construct a finite automation which accepts the language $L(r)$.*

Let us first consider only part of the regular expression, $ab + aba$, and let $r_1 = ab$ and $r_2 = aba$. Therefore, $r_1 = a \circ b$. Applying the rule of concatenation from Theorem **??**, the FA which can recognize $L(r_1)$ is given as shown in Figure 8.1(a).

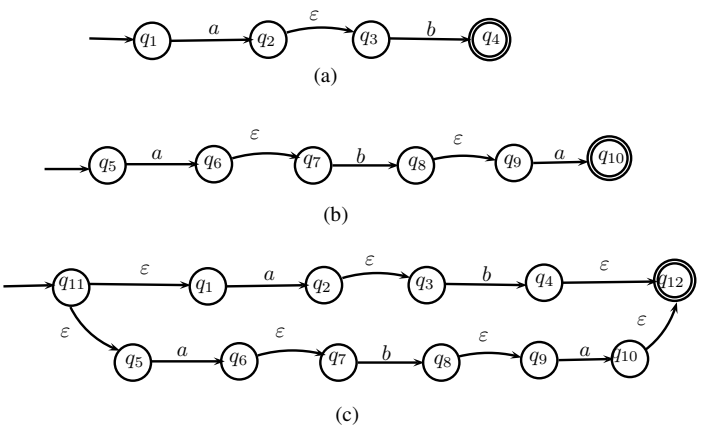

Figure 8.1: (a) FA for $r_1 = ab$, (b) FA for $r_2 = aba$, (c) FA for $r_3 = r_1 + r_2$

Similarly, the FA which can recognize $L(r_2)$ is shown in figure 8.1(b). Let $r_3 = r_1 + r_2$. By applying the union operation on the set of languages $L(r_1)$ and $L(r_2)$, the new language obtained is $L(r_3) = L(r_1)L(r_2)$. The finite automata for $L(r_3)$ is as shown in figure 8.1(c).

Finally, applying the operation of Kleene Star on the language $L(r_3)$ we get $(L(r_3))^* = L(r_3^*)$. The finite automaton for this is shown in figure 8.2. The start state for the NFA M is q_0 and final state is q_{12}.

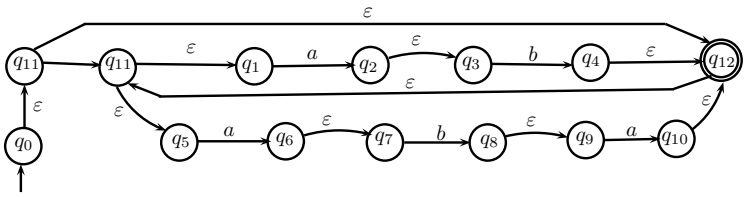

Figure 8.2: FA for regular expression $(ab + aba)^*$

We have noted that, the main part in deriving a FA for a given regular expression is to find subexpressions in the regular expressions, in the form of $r + s$, $r \circ q$ and r^*. Therefore, parentheses are essential for a complex expression to make precedence explicit. In this way, we can construct an NFA for any given regular expression. This NFA can be converted into DFA as discussed in the previous classes.

8.1.1 Generalized NFA

Unlike the NFA, which reads one symbol at a time, a generalized NFA (GNFA) reads block of symbols from the input, and moves along the transition arrows connecting two states when a block of symbols is read. This block of symbols is nothing but a regular expression. The GNFA accepts the input string corresponding to the label if it leads to the final state when the string is completely read.

A GNFA is formally defined as,

$$G = (Q, \Sigma, \rho, q_0, q_f), \tag{8.2}$$

where,

Q is finite set of states,

Σ is alphabet set,

$q_0 \in Q$ is initial state,

$q_f \in F$ is single final state,

$\rho : (Q - \{q_f\}) \times (Q - \{q_0\}) \to R_\Sigma$.

In the above R_Σ is set of all regular expressions over the alphabet set Σ.

An input string is accepted if there is a way to read the input string such that the GNFA is in the accepting (final) state after processing of the entire string. A GNFA has one start state (q_{start}) with no incoming arrows, and one accepting state (q_{final}) with no outgoing arrows.

We break this procedure into two parts, using a GNFA. First we show how to convert DFAs into GNFAs, and then GNFAs into regular expressions. The method is called *state removal method* (see figure 8.3).

Figure 8.3: Conversion of DFA to Regular Expression

The process of constructing regular expression requires the finite automaton with only one initial and one final state. To achieve this, we eliminate one state at a time from the initial transition-diagram. The resulting successive transition graphs are with r arrows, where r is a regular expression.

Converting a DFA into GNFA:

1. Add a new start state (q_{start}), with ε-transition from this to q_0).
2. Add a new accept state (q_{accept}), with ε-transition from each of the final state to q_{accept}.
3. If an edge in original DFA has multiple labels, replace the labels with a regular expression formed by the union of these labels.
4. If originally there is no arrow between two states, add a new edge with label ϕ.

Converting a GNFA into Regular Expression:

1. We interactively remove states one by one in GNFA such that, after each state is removed, the new GNFA obtained will recognize the same language as the previous one.
2. When states in the GNFA are two, we have achieved the regular expression for the given DFA, represented by the label of the edge connecting these states.

Removing a state q (not q_{start} or q_{accept}) in the process of construction of GNFA is demonstrated in Figure 8.3. The new label on the arrow from q_i to q_j becomes Regular Expression that describes all strings that take GNFA from q_i to q_j either directly or via q (the removed state). The Figure 8.4 shows an example of conversion of DFA to an equivalent GNFA.

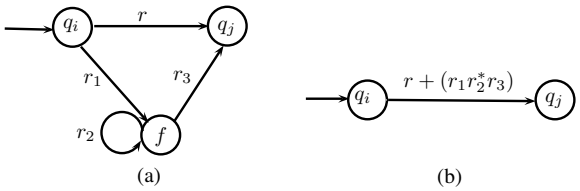

Figure 8.4: Conversion of DFA to GNFA: (a) DFA, (b) Equivalent GNFA.

Example 8.3 *Obtain Regular expressions for given finite Automata using the approach of GNFA construction.*

Various possible cases are shown for obtaining regular expressions for given finite automata. Each case shows an equivalent automaton for a given regular expression.

Case-I. A finite automaton accepts the expression $r_1 + r_2 + \cdots + r_n$ (figure 8.5(a)). Replace it by the one shown in figure 8.5(b).

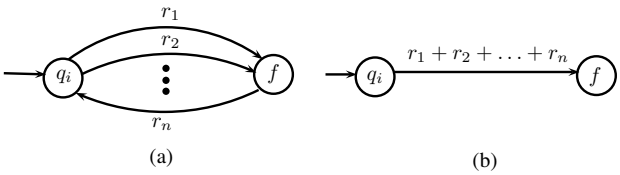

Figure 8.5: FA to Regular expression(case-I)

Case-II). If the transition diagram is like the one shown in figure 8.6(a), replace it by that in figure 8.6(b).

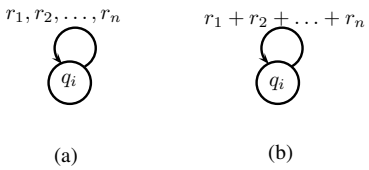

Figure 8.6: FA to Regular expression(case-II)

Case-III. If there are more than one final states, say, f_1, f_2, f_3 (figure 8.7(a), then replace them with single final state (figure 8.7(b)).

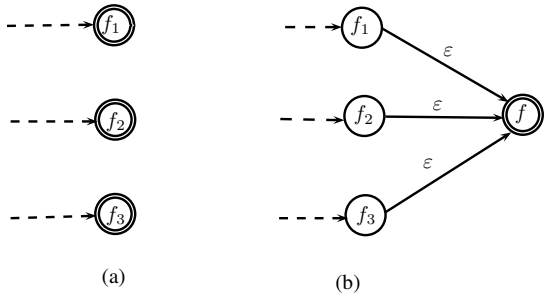

Figure 8.7: FA to Regular expression(case-III)

Case-IV. Simplify the transition diagram of three states as in figure 8.8(a) and replace it by figure 8.8(b).

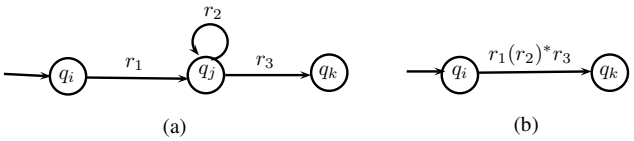

Figure 8.8: FA to Regular expression(case-IV)

Case-V. Replace the old transition diagram in figure 8.9(a) by shown in figure 8.9(b).

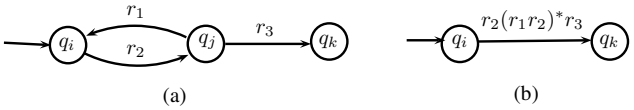

Figure 8.9: FA to Regular expression(case-V).

Case VI. Replace the old transition diagram in figure 8.10(a) by that in figure 8.10(b).

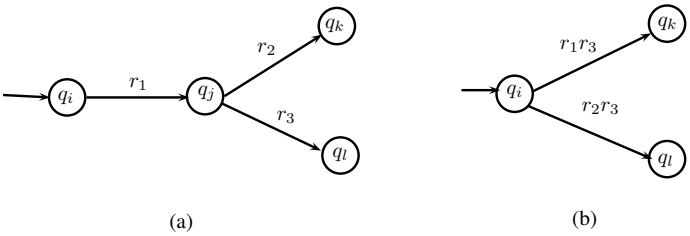

Figure 8.10: FA to Regular expression(case-VI)

In the similar way, using these primitives, any complex automaton can be reduced to GNFA, which ultimately produce the regular expression.

Example 8.4 *Find the regular expression for the automaton shown in the Figure 8.11.*

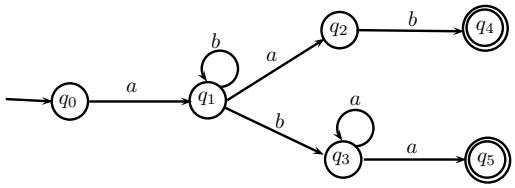

Figure 8.11: Finite automaton

The reduction steps discussed above make use of rules discussed we had discussed in the beginning of these notes. The resultant regular expression is $ab^*(ab + ba^*a)$.

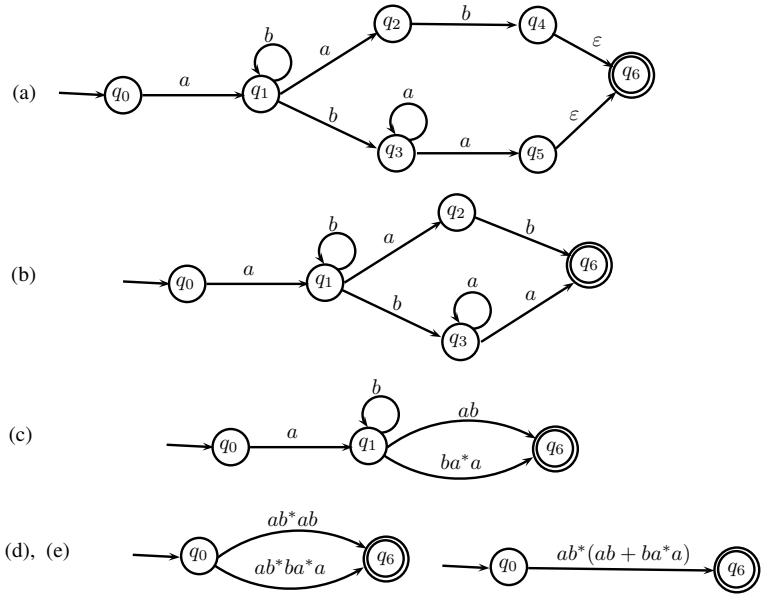

Figure 8.12: Steps (a)-(e) are for construction of regular expression from a Finite automaton.

The reduction steps for the finite automation, to obtain the regular expression, after conversion of FA into GNFA, are shown in Figure 8.12.

Using the primitives discussed above in most cases, a DFA can be converted into GNFA, and then to regular expression. However, some times the above method is too cumbersome, and even may not be a universal solution some times.