3.2

3.2 Connection Between Regular Expressions and Regular Languages

Regular Expressions Denote Regular Languages

Theorem 3.1 Let r be a regular expression. Then there exists some nondeterministic finite accepter that accepts $L(r)$. Consequently, $L(r)$ is a regular language.

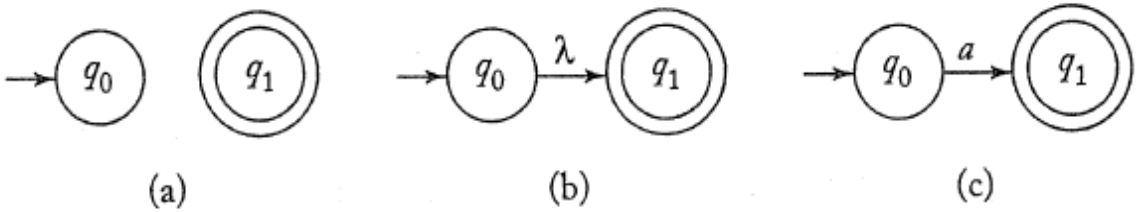


Figure 3.1
(a) nfa accepts $\varnothing$.
(b) nfa accepts $\{\lambda\}$.
(c) nfa accepts $\{a\}$.

Figure 3.2
Schematic representation of an nfa accepting $L(r)$.

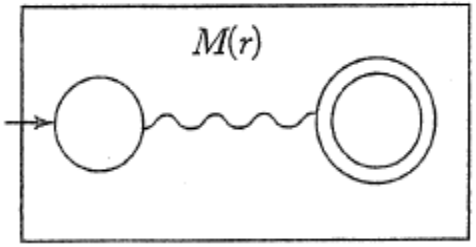


Figure 3.3
Automaton for $L(r_1 + r_2)$.

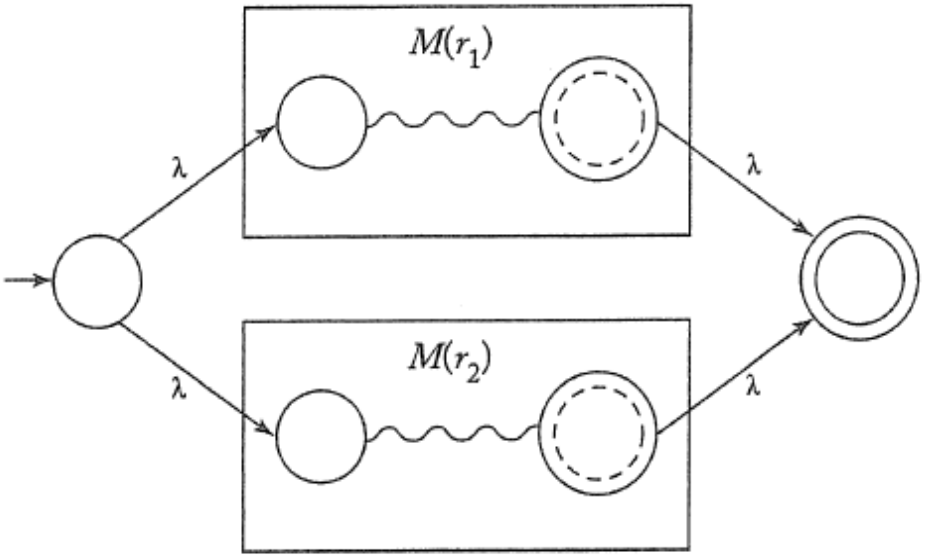


Figure 3.4
Automaton for $L(r_1 r_2)$.

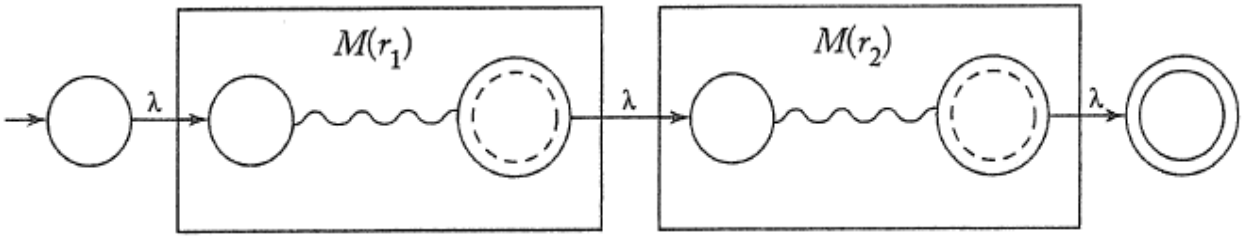
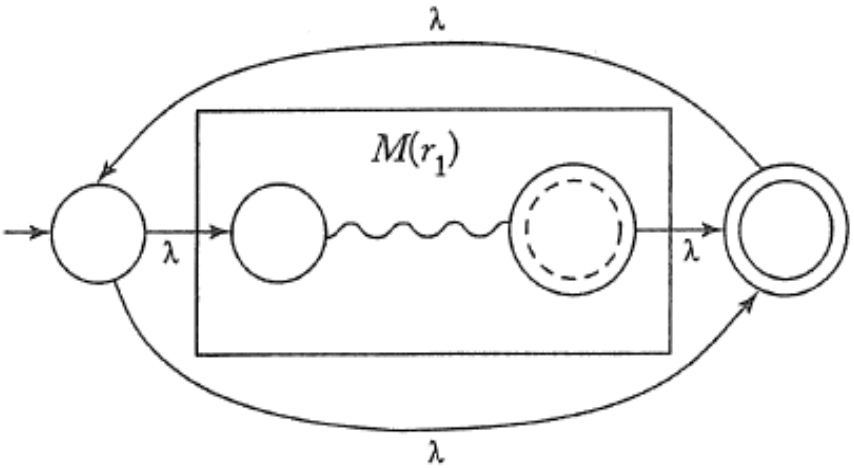


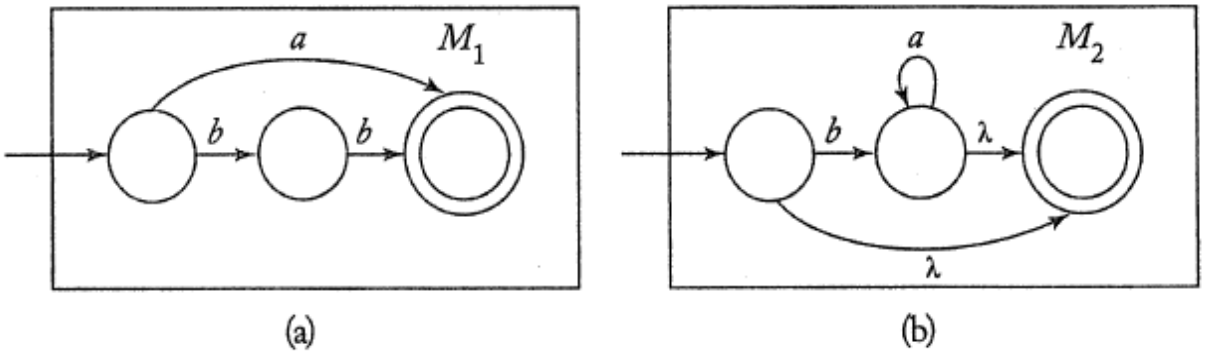
Figure 3.5
Automaton for $L(r_1^*)$.



Example 3.7 Find an nfa that accepts $L(r)$, where

$$r = (a + bb)^* (ba^* + \lambda).$$

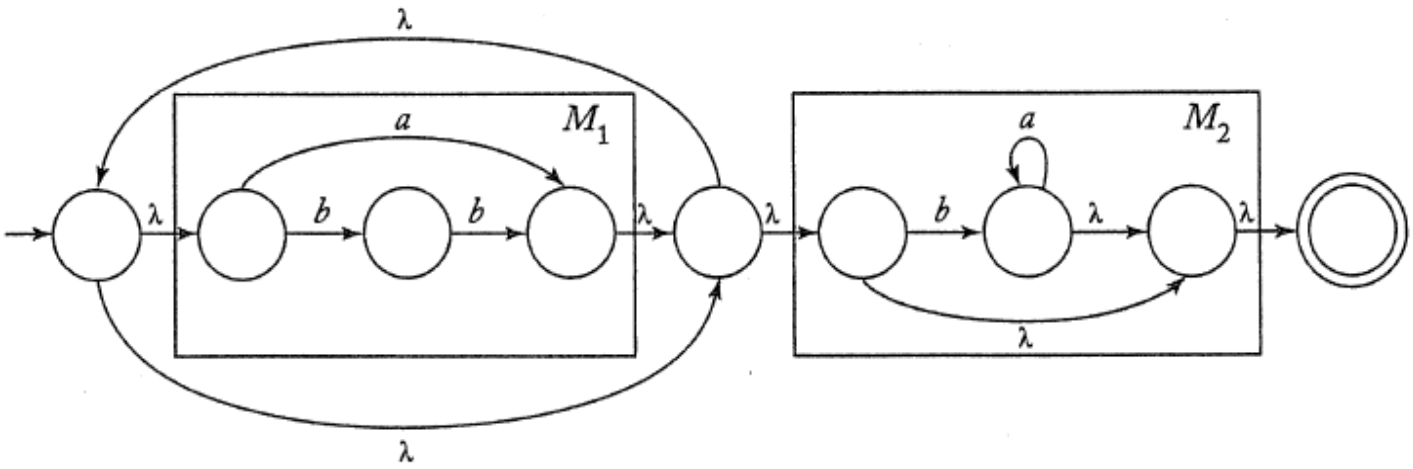
Automata for $(a + bb)$ and $(ba^* + \lambda)$, constructed directly from first principles, are given in Figure 3.6. Putting these together using the construction in Theorem 3.1, we get the solution in Figure 3.7.



(a) (b)

Figure 3.6
(a) M_1 accepts $L(a + bb)$.
(b) M_2 accepts $L(ba^* + \lambda)$.

Figure 3.7
Automaton accepts $L((a+bb)^*(ba^*+\lambda))$.



Regular Expressions for Regular Languages

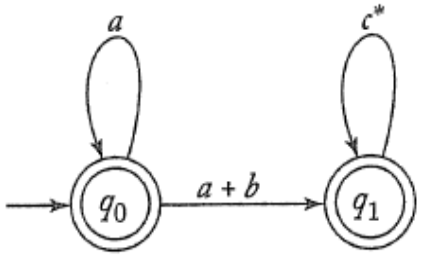
generalized transition graphs (GTG)

A generalized transition graph is a transition graph whose edges are labeled with regular expressions; otherwise it is the same as the usual transition graph. The label of any walk from the initial state to a final state is the concatenation of several regular expressions, and hence itself a regular expression. The strings denoted by such regular expressions are a subset of the language accepted by the generalized transition graph, with the full language being the union of all such generated subsets.

Example 3.8

Figure 3.8 represents a generalized transition graph. The language accepted by it is $L(a^* + a^*(a+b)c^*)$, as should be clear from an inspection of the graph. The edge (q_0, q_0) labeled a is a cycle that can generate any number of a's, that is, it represents $L(a^*)$. We could have labeled this edge a^* without changing the language accepted by the graph.

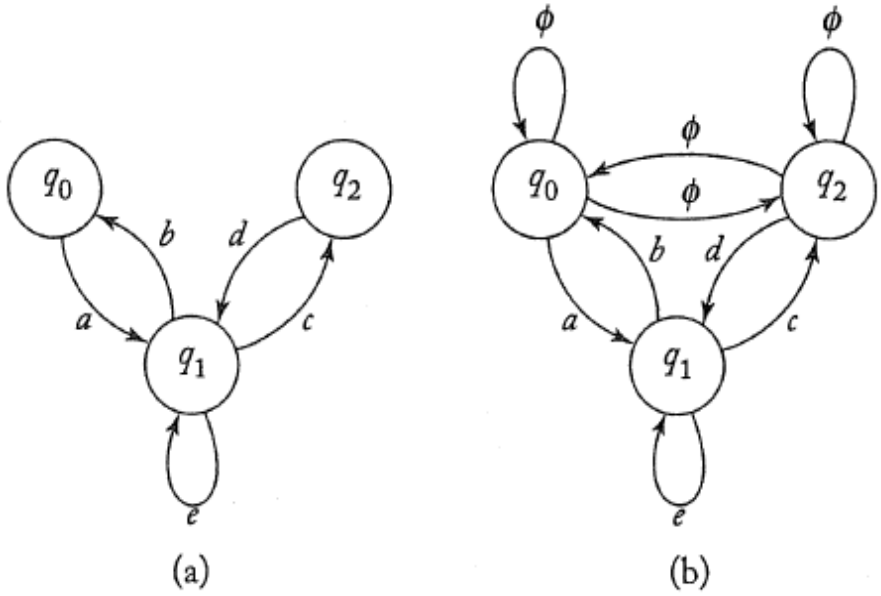
Figure 3.8



Equivalence for generalized transition graphs is defined in terms of the language accepted and the purpose of the next bit of discussion is to produce a sequence of increasingly simple GTGs. In this, we will find it convenient to work with complete GTGs. A complete GTG is a graph in which all edges are present. If a GTG, after conversion from an nfa, has some edges missing, we put them in and label them with $\varnothing$. A complete GTG with $|V|$ vertices has exactly $|V|^2$ edges.

Example 3.9 The GTG in Figure 3.9(a) is not complete. Figure 3.9(b) shows how it is completed.

Figure 3.9



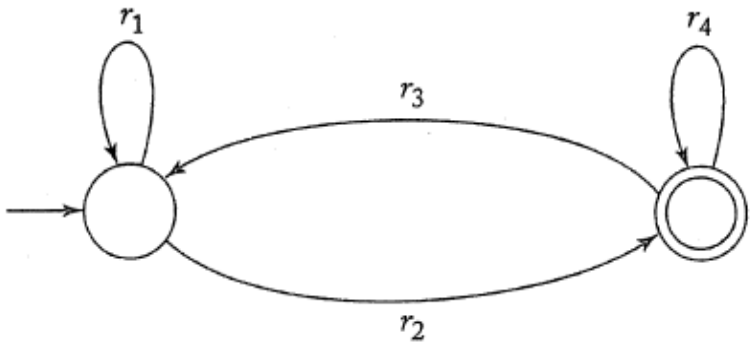
Suppose now that we have the simple two-state complete GTG shown in Figure 3.10. By mentally tracing through this GTG you can convince yourself that the regular expression

$$r = r_1^* r_2 (r_4 + r_3 r_1^* r_2)^* \tag{3.1}$$

covers all possible paths and so is the correct regular expression associated with the graph.

When a GTG has more than two states, we can find an equivalent graph by removing one state at a time. We will illustrate this with an example before going to the general method.

Figure 3.10



Example 3.10 Consider the complete GTG in Figure 3.11. To remove q_2, we first intoduce some new edges. We

create an edge from q_1 to q_1 and label it $e + af^*b$,
create an edge from q_1 to q_3 and label it $h + af^*c$,
create an edge from q_3 to q_1 and label it $i + df^*b$,
create an edge from q_3 to q_1 and label it $g + df^*c$.

When this is done, we remove q_2 and all associated edges. This gives the GTG in Figure 3.12. You can explore the equivalence of the two GTGs by seeing how regular expressions such as af^*c and e^*ab are generated.

Fig 3.11

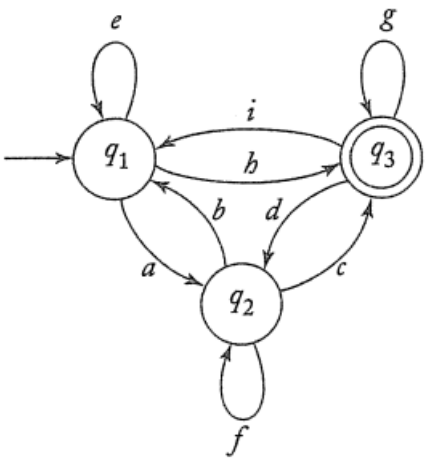
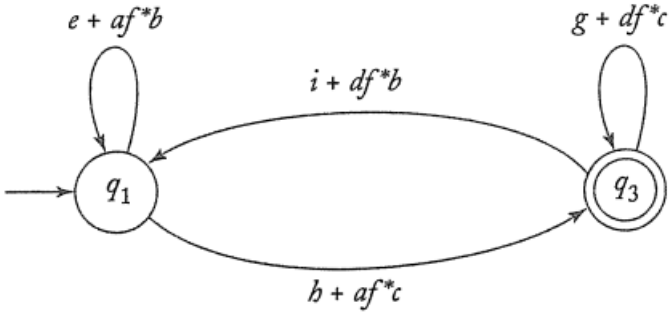


Fig 3.12



For arbitrary GTGs we remove one state at a time until only two states are left. Then we apply Equation (3.1) to get the final regular expression. This tends to be a lengthy process, but it is staightforward as the following procedure shows.

procedure: nfa-to-rex

1. Start with an nfa with states $q_0, q_1, ..., q_n$, and a single final state, distinct from its initial state.
2. Convert the nfa into a complete generalized transition graph. Let r_{ij} stand for the label of the edge from q_i to q_j.
3. If the GTG has only two states, with q_i as its initial state and q_j its final state, its associated regular expression is

$$r = r_{ii}^* r_{ij} (r_{jj} + r_{ji} r_{ii}^* r_{ij})^*. \tag{3.2}$$

4. If the GTG has three states, with initial state q_i, final state q_j, and third state q_k, introduce new edges, labeled

$$r_{pq} + r_{pk} r_{kk}^{*} r_{kq} \tag{3.3}$$

for $p = i, j$, $q = i, j$. When this is done, remove vertex q_k and its associated edges.

5. If the GTG has four or more states, pick a state q_k to be removed. Apply rule 4 for all pairs of states $(q_i, q_j), i \neq k, j \neq k$. At each step apply the simplifying rules

$$r + \varnothing = r,$$

$$r\varnothing = \varnothing,$$

$$\varnothing^{*} = \lambda,$$

wherever possible. When this is done, remove state q_k.

6. Repeat Steps 3 to 5 until the correct regular expression is obtained.

Example 3.11 Find a regular expression for the language

$$L = \{w \in \{a,b\}^* : n_a(w) \text{ is even and } n_b(w) \text{ is odd}\}.$$

An attempt to construct a regular expression directly from this description leads to all kinds of difficulties. On the other hand, finding an nfa for it is easy as long as we use vertex labeling effectively. We label the vertices with EE to denote an even number of a's and b's, with OE to denote an odd number of a's and an even number of b's, and so on. With this we easily get the solution which, after conversion into a complete generalized transition graph, is in Figure 3.13.

We now apply the conversion to a regular expression, using procedure *nfa-to-rex*. To remove the state OE, we apply Equation (3.3). The edge between EE and itself will have the label

$$\begin{aligned} r_{EE} &= \varnothing + a\varnothing^* a \\ &= aa. \end{aligned}$$

We continue in this manner until we get the GTG in Figure 3.14. Next, the state OO is removed, which gives Figure 3.15. Finally, we get the correct regular expression from Equation (3.2).

Figure 3.13

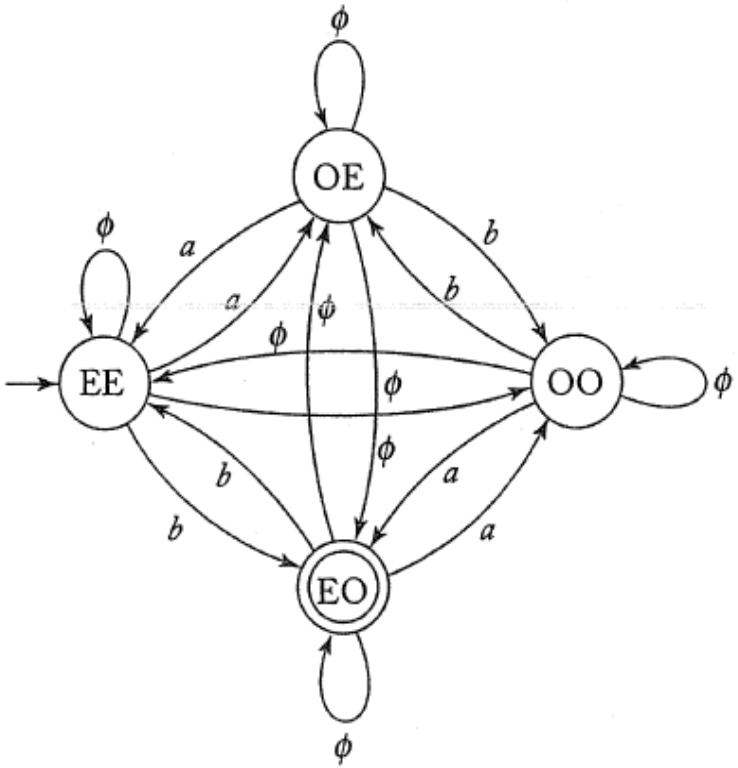


Figure 3.14

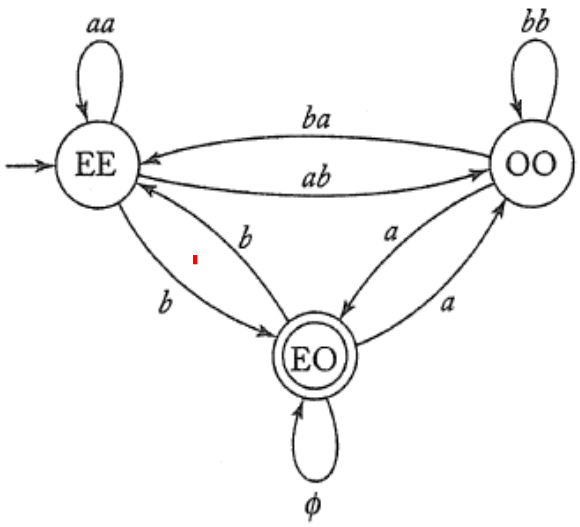
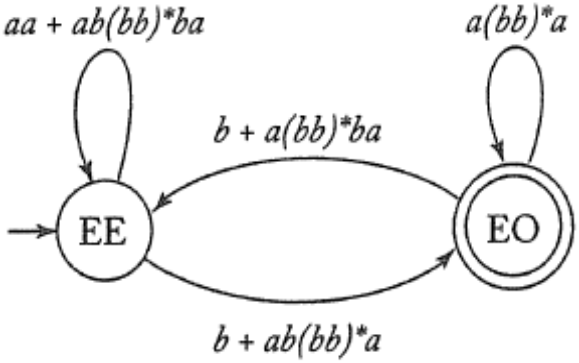


Figure 3.15



Theorem 3.2 Let L be a regular language. Then there exists a regular expression r such that $L = L(r)$.

Proof: If L is regular, there exists an nfa for it. We can assume without loss of generality, that this nfa has a single final state, distinct from its initial state. We convert this nfa to a complete generalized transition graph and apply the procedure *nfa-to-rex* to it. This yields the required regular expression r.

While this can make the result plausible, a rigorous proof requires that we show that each step in the process generates an equivalent GTG. This is a technical matter we leave to the reader. ■

Regular Expressions for Describing Simple Patterns (skip)