Algorithm 3.3. (*Thompson's construction.*) An NFA from a regular expression.

Input. A regular expression r over an alphabet Σ.

Output. An NFA N accepting $L(r)$.

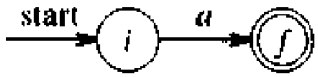
Method. We first parse r into its constituent subexpressions. Then, using rules (1) and (2) below, we construct NFA's for each of the basic symbols in r (those that are either ϵ or an alphabet symbol). The basic symbols correspond to parts (1) and (2) in the definition of a regular expression. It is important to understand that if a symbol a occurs several times in r, a separate NFA is constructed for each occurrence.

Then, guided by the syntactic structure of the regular expression r, we combine these NFA's inductively using rule (3) below until we obtain the NFA for the entire expression. Each intermediate NFA produced during the course of the construction corresponds to a subexpression of r and has several important properties: it has exactly one final state, no edge enters the start state, and no edge leaves the final state.

1. For ϵ, construct the NFA

Here i is a new start state and f a new accepting state. Clearly, this NFA recognizes $\{\epsilon\}$.

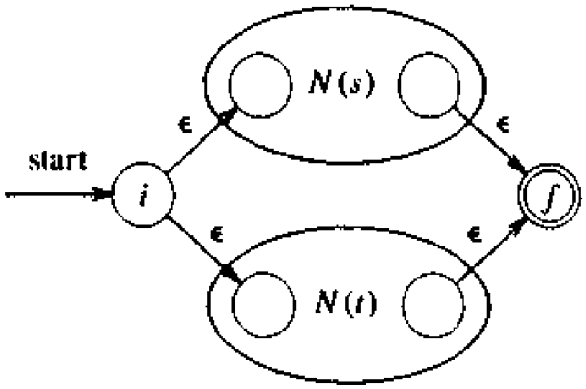
2. For a in Σ, construct the NFA



Again i is a new start state and f a new accepting state. This machine recognizes $\{a\}$.

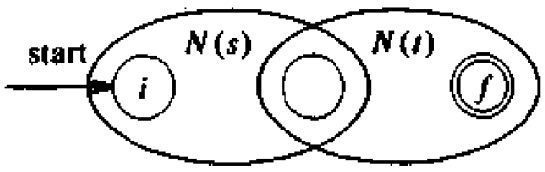
3. Suppose $N(s)$ and $N(t)$ are NFA's for regular expressions s and t.

   a) For the regular expression $s|t$, construct the following composite NFA $N(s|t)$:



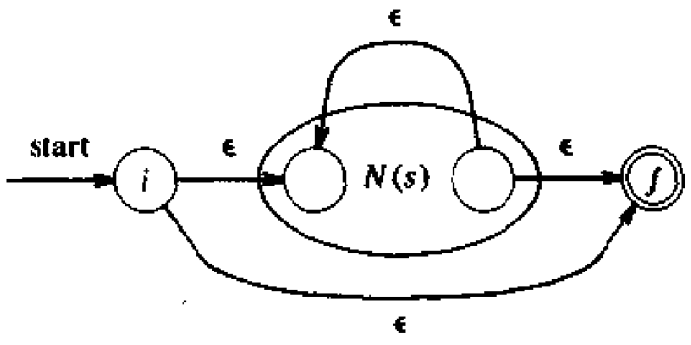
Here i is a new start state and f a new accepting state. There is a transition on ϵ from i to the start states of $N(s)$ and $N(t)$. There is a transition on ϵ from the accepting states of $N(s)$ and $N(t)$ to the new accepting state f. The start and accepting states of $N(s)$ and $N(t)$ are not start or accepting states of $N(s|t)$. Note that any path from i to f must pass through either $N(s)$ or $N(t)$ exclusively. Thus, we see that the composite NFA recognizes $L(s) \cup L(t)$.

b) For the regular expression st, construct the composite NFA $N(st)$:



The start state of $N(s)$ becomes the start state of the composite NFA and the accepting state of $N(t)$ becomes the accepting state of the composite NFA. The accepting state of $N(s)$ is merged with the start state of $N(t)$; that is, all transitions from the start state of $N(t)$ become transitions from the accepting state of $N(s)$. The new merged state loses its status as a start or accepting state in the composite NFA. A path from i to f must go first through $N(s)$ and then through $N(t)$, so the label of that path will be a string in $L(s)L(t)$. Since no edge enters the start state of $N(t)$ or leaves the accepting state of $N(s)$, there can be no path from i to f that travels from $N(t)$ back to $N(s)$. Thus, the composite NFA recognizes $L(s)L(t)$.

c) For the regular expression s^*, construct the composite NFA $N(s^*)$:



Here i is a new start state and f a new accepting state. In the composite NFA, we can go from i to f directly, along an edge labeled ϵ, representing the fact that ϵ is in $(L(s))^*$, or we can go from i to f passing through $N(s)$ one or more times. Clearly, the composite NFA recognizes $(L(s))^*$.

d) For the parenthesized regular expression (s), use $N(s)$ itself as the NFA.

Every time we construct a new state, we give it a distinct name. In this way, no two states of any component NFA can have the same name. Even if the same symbol appears several times in r, we create for each instance of that symbol a separate NFA with its own states. □

We can verify that each step of the construction of Algorithm 3.3 produces an NFA that recognizes the correct language. In addition, the construction produces an NFA $N(r)$ with the following properties.

1. $N(r)$ has at most twice as many states as the number of symbols and operators in r. This follows from the fact each step of the construction creates at most two new states.
2. $N(r)$ has exactly one start state and one accepting state. The accepting state has no outgoing transitions. This property holds for each of the constituent automata as well.
3. Each state of $N(r)$ has either one outgoing transition on a symbol in Σ or at most two outgoing ϵ-transitions.

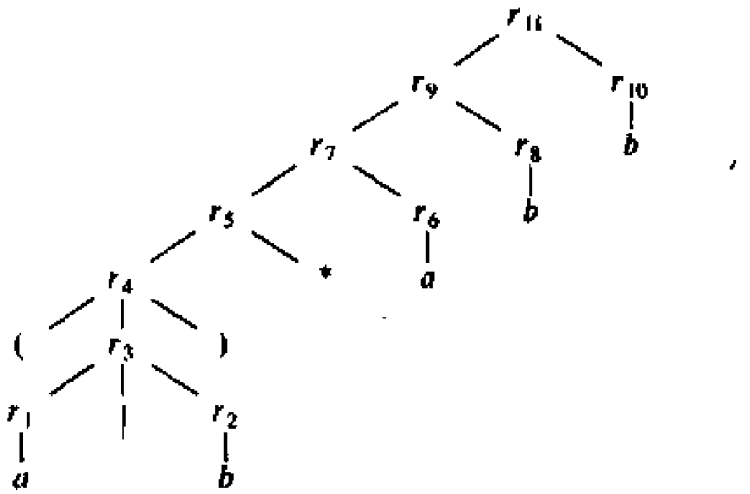
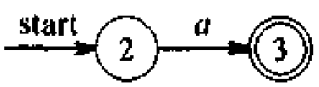


Fig. 3.30. Decomposition of $(a|b)^*abb$.

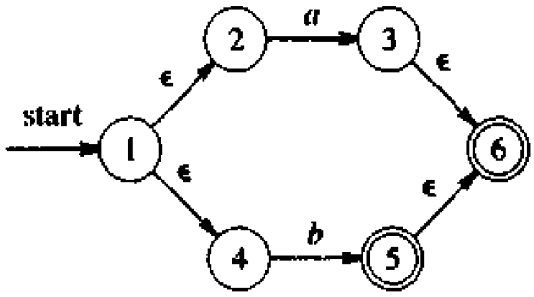
Example 3.16. Let us use Algorithm 3.3 to construct $N(r)$ for the regular expression $r = (a|b)^*abb$. Figure 3.30 shows a parse tree for r that is analogous to the parse trees constructed for arithmetic expressions in Section 2.2. For the constituent r_1, the first a, we construct the NFA



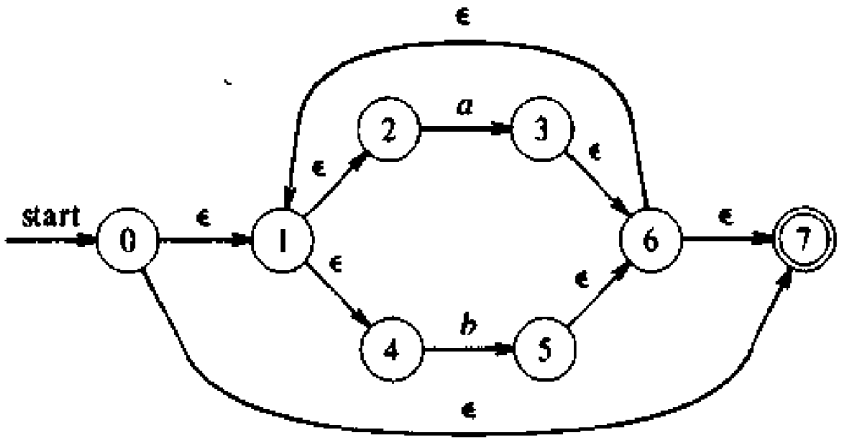
For r_2 we construct

We can now combine $N(r_1)$ and $N(r_2)$ using the union rule to obtain the

NFA for $r_3 = r_1|r_2$

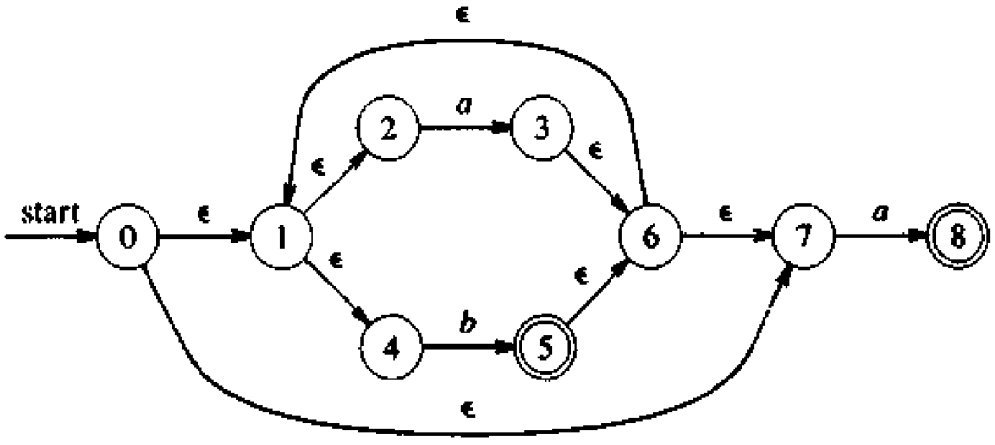


The NFA for (r_3) is the same as that for r_3. The NFA for $(r_3)^*$ is then:



The NFA for $r_6 = a$ is

To obtain the automaton for $r_5 r_6$, we merge states 7 and 7', calling the resulting state 7, to obtain



Continuing in this fashion we obtain the NFA for $r_{11} = (a|b)^*abb$ that was first exhibited in Fig. 3.27. □