position of the leaf and also as a position of its symbol. Note that a symbol can have several positions; for instance, a has positions 1 and 3 in Fig. 3.56. The positions in the syntax tree correspond to the important states of the constructed NFA.

Example 3.32 : Figure 3.57 shows the NFA for the same regular expression as Fig. 3.56, with the important states numbered and other states represented by letters. The numbered states in the NFA and the positions in the syntax tree correspond in a way we shall soon see. □

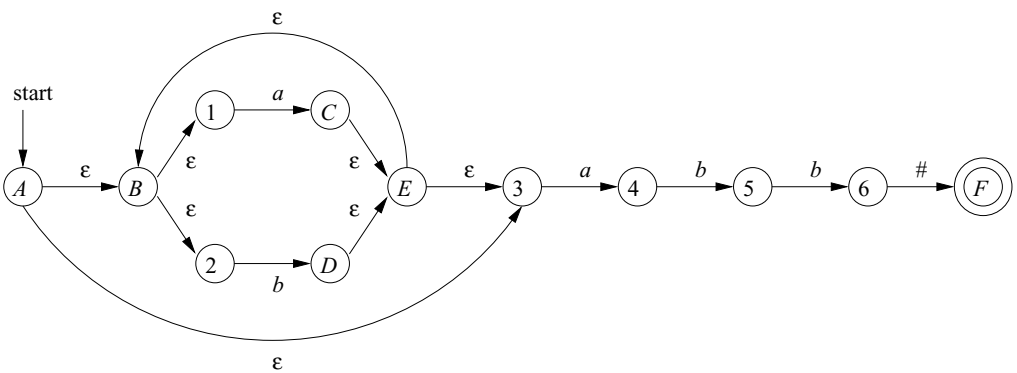

Figure 3.57: NFA constructed by Algorithm 3.23 for $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}\#$

3.9.2 Functions Computed From the Syntax Tree

To construct a DFA directly from a regular expression, we construct its syntax tree and then compute four functions: *nullable*, *firstpos*, *lastpos*, and *followpos*, defined as follows. Each definition refers to the syntax tree for a particular augmented regular expression $(r)\#$.

1. *nullable*(n) is true for a syntax-tree node n if and only if the subexpression represented by n has ϵ in its language. That is, the subexpression can be "made null" or the empty string, even though there may be other strings it can represent as well.

2. *firstpos*(n) is the set of positions in the subtree rooted at n that correspond to the first symbol of at least one string in the language of the subexpression rooted at n.

3. *lastpos*(n) is the set of positions in the subtree rooted at n that correspond to the last symbol of at least one string in the language of the subexpression rooted at n.

4. *followpos*(p), for a position p, is the set of positions q in the entire syntax tree such that there is some string $x = a_1 a_2 \cdots a_n$ in $L\big((r)\#\big)$ such that for some i, there is a way to explain the membership of x in $L\big((r)\#\big)$ by matching a_i to position p of the syntax tree and a_{i+1} to position q.

Example 3.33: Consider the cat-node n in Fig. 3.56 that corresponds to the expression $(\mathbf{a}|\mathbf{b})^*\mathbf{a}$. We claim *nullable*($n$) is false, since this node generates all strings of a's and b's ending in an a; it does not generate ϵ. On the other hand, the star-node below it is nullable; it generates ϵ along with all other strings of a's and b's.

firstpos(n) $= \{1, 2, 3\}$. In a typical generated string like aa, the first position of the string corresponds to position 1 of the tree, and in a string like ba, the first position of the string comes from position 2 of the tree. However, when the string generated by the expression of node n is just a, then this a comes from position 3.

lastpos(n) $= \{3\}$. That is, no matter what string is generated from the expression of node n, the last position is the a from position 3 of the tree.

followpos is trickier to compute, but we shall see the rules for doing so shortly. Here is an example of the reasoning: *followpos*(1) $= \{1, 2, 3\}$. Consider a string $\cdots ac \cdots$, where the c is either a or b, and the a comes from position 1. That is, this a is one of those generated by the $\mathbf{a}$ in expression $(\mathbf{a}|\mathbf{b})^*$. This a could be followed by another a or b coming from the same subexpression, in which case c comes from position 1 or 2. It is also possible that this a is the last in the string generated by $(\mathbf{a}|\mathbf{b})^*$, in which case the symbol c must be the a that comes from position 3. Thus, 1, 2, and 3 are exactly the positions that can follow position 1. □

3.9.3 Computing *nullable*, *firstpos*, and *lastpos*

We can compute *nullable*, *firstpos*, and *lastpos* by a straightforward recursion on the height of the tree. The basis and inductive rules for *nullable* and *firstpos* are summarized in Fig. 3.58. The rules for *lastpos* are essentially the same as for *firstpos*, but the roles of children c_1 and c_2 must be swapped in the rule for a cat-node.

Example 3.34: Of all the nodes in Fig. 3.56 only the star-node is nullable. We note from the table of Fig. 3.58 that none of the leaves are nullable, because they each correspond to non-ϵ operands. The or-node is not nullable, because neither of its children is. The star-node is nullable, because every star-node is nullable. Finally, each of the cat-nodes, having at least one nonnullable child, is not nullable.

The computation of *firstpos* and *lastpos* for each of the nodes is shown in Fig. 3.59, with *firstpos*(n) to the left of node n, and *lastpos*(n) to its right. Each of the leaves has only itself for *firstpos* and *lastpos*, as required by the rule for non-ϵ leaves in Fig. 3.58. For the or-node, we take the union of *firstpos* at the

NODE n	*nullable*(n)	*firstpos*(n)
A leaf labeled ϵ	**true**	$\emptyset$
A leaf with position i	**false**	$\{i\}$
An or-node $n = c_1 \mid c_2$	*nullable*(c_1) **or** *nullable*(c_2)	*firstpos*(c_1) $\cup$ *firstpos*(c_2)
A cat-node $n = c_1 c_2$	*nullable*(c_1) **and** *nullable*(c_2)	**if** (*nullable*(c_1)) *firstpos*(c_1) $\cup$ *firstpos*(c_2) **else** *firstpos*(c_1)
A star-node $n = {c_1}^*$	**true**	*firstpos*(c_1)

Figure 3.58: Rules for computing *nullable* and *firstpos*

children and do the same for *lastpos*. The rule for the star-node says that we take the value of *firstpos* or *lastpos* at the one child of that node.

Now, consider the lowest cat-node, which we shall call n. To compute *firstpos*(n), we first consider whether the left operand is nullable, which it is in this case. Therefore, *firstpos* for n is the union of *firstpos* for each of its children, that is $\{1,2\} \cup \{3\} = \{1,2,3\}$. The rule for *lastpos* does not appear explicitly in Fig. 3.58, but as we mentioned, the rules are the same as for *firstpos*, with the children interchanged. That is, to compute *lastpos*(n) we must ask whether its right child (the leaf with position 3) is nullable, which it is not. Therefore, *lastpos*(n) is the same as *lastpos* of the right child, or $\{3\}$. □

3.9.4 Computing *followpos*

Finally, we need to see how to compute *followpos*. There are only two ways that a position of a regular expression can be made to follow another.

1. If n is a cat-node with left child c_1 and right child c_2, then for every position i in *lastpos*(c_1), all positions in *firstpos*(c_2) are in *followpos*(i).

2. If n is a star-node, and i is a position in *lastpos*(n), then all positions in *firstpos*(n) are in *followpos*(i).

Example 3.35 : Let us continue with our running example; recall that *firstpos* and *lastpos* were computed in Fig. 3.59. Rule 1 for *followpos* requires that we look at each cat-node, and put each position in *firstpos* of its right child in *followpos* for each position in *lastpos* of its left child. For the lowest cat-node in Fig. 3.59, that rule says position 3 is in *followpos*(1) and *followpos*(2). The next cat-node above says that 4 is in *followpos*(3), and the remaining two cat-nodes give us 5 in *followpos*(4) and 6 in *followpos*(5).