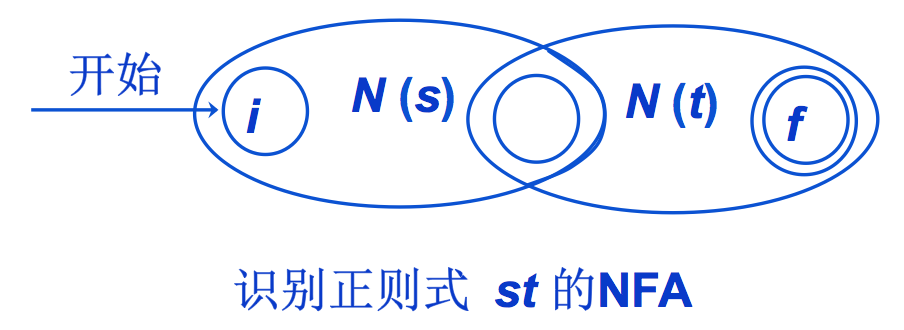
→∪Σε

正则表达式 🡪 NFA

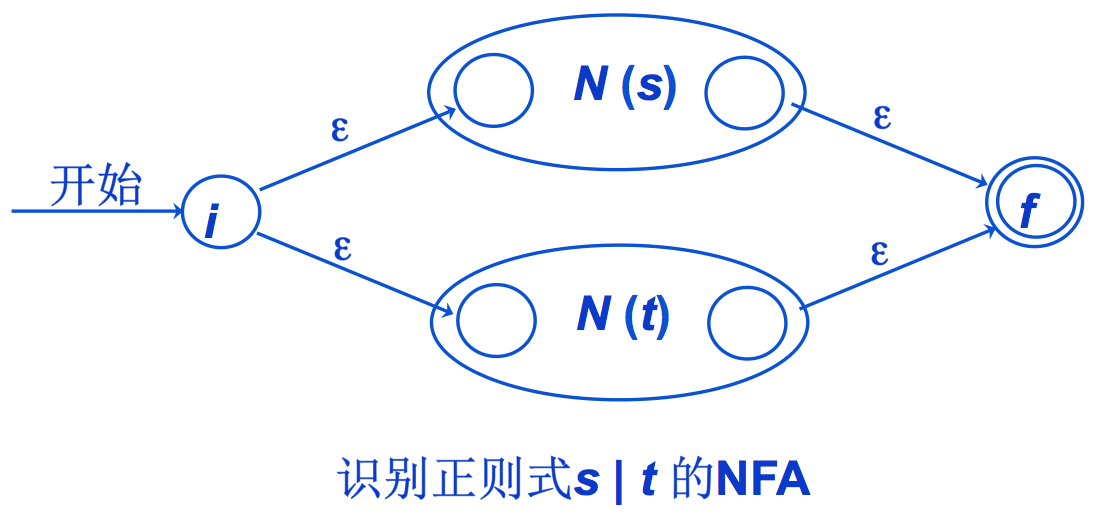
(1) 两个基本转换

(3) 三个操作转换

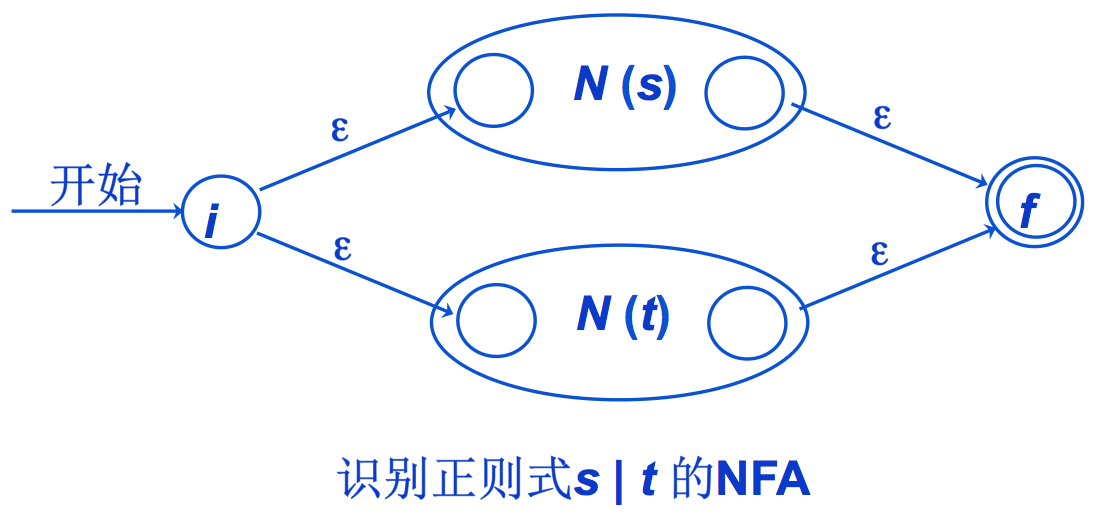
3.1 连接操作



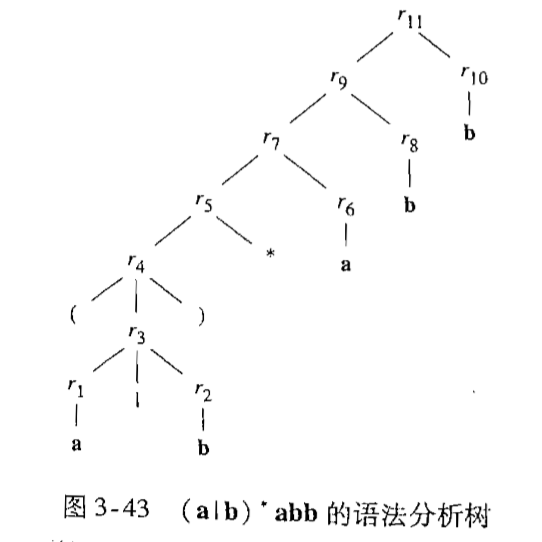
3.2 选择操作



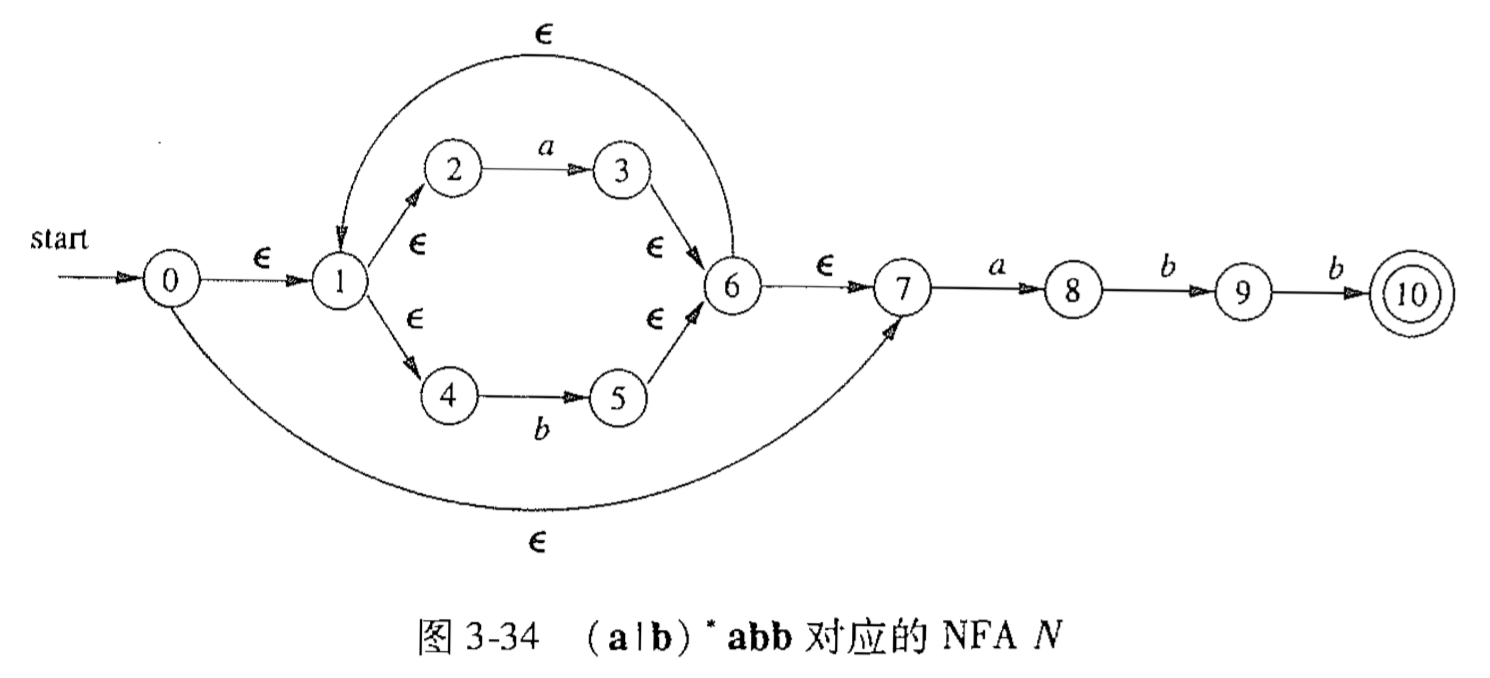
3.3 闭包操作



给定正则表达式 r = (a|b)\*abb构造一个NFA.



构造NFA是一个自底向上的过程.

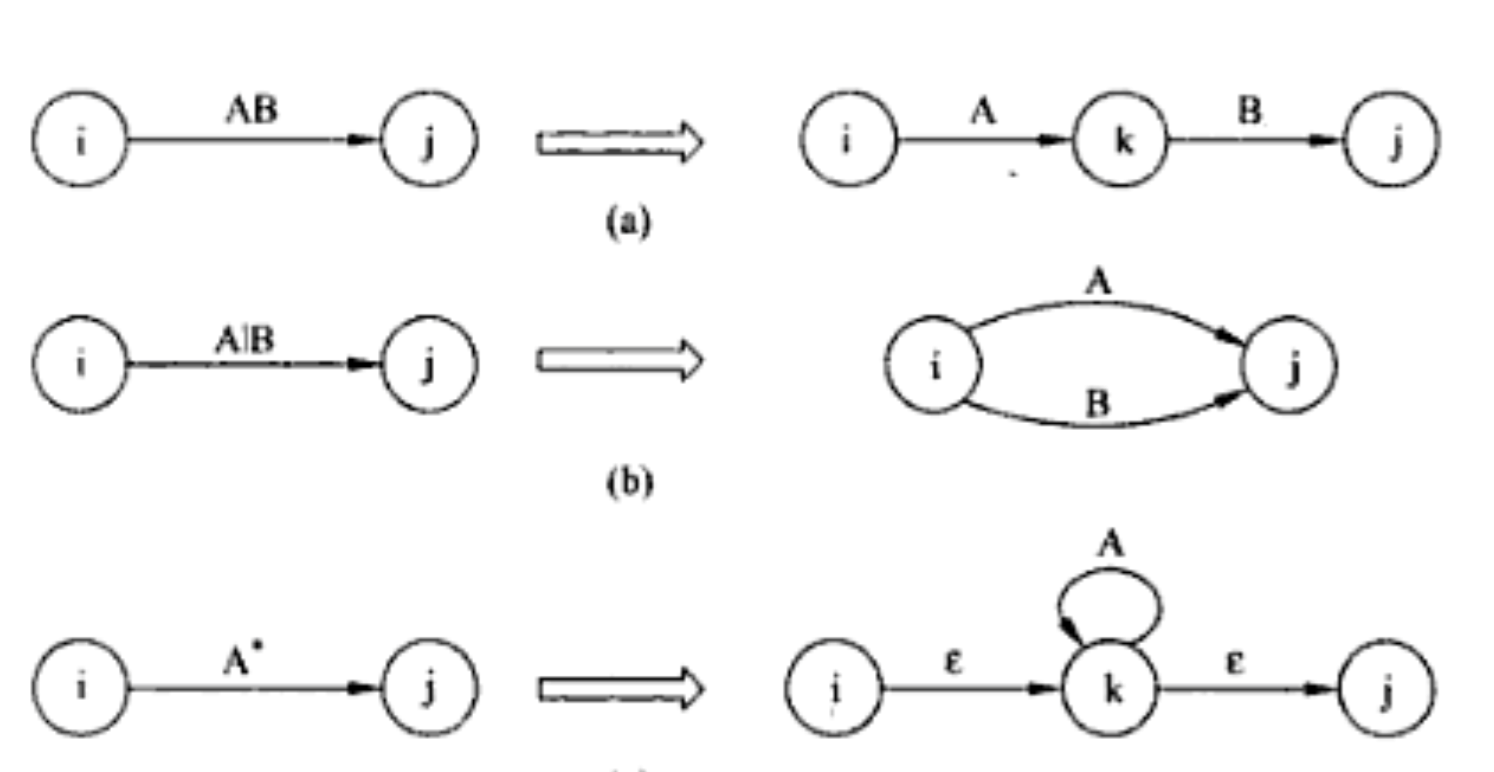


如何自顶向下构建NFA?

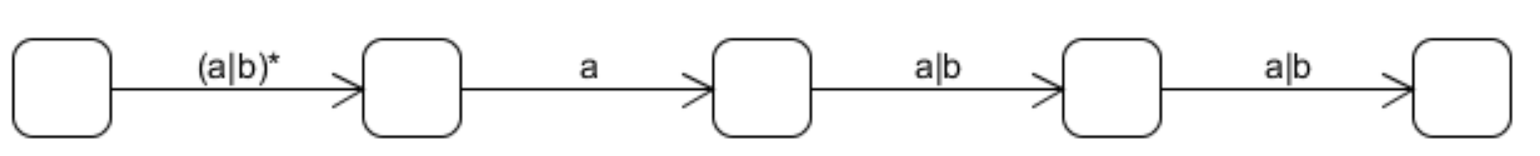
(a|b)\*abb总共6结点，7条边（2空串边）

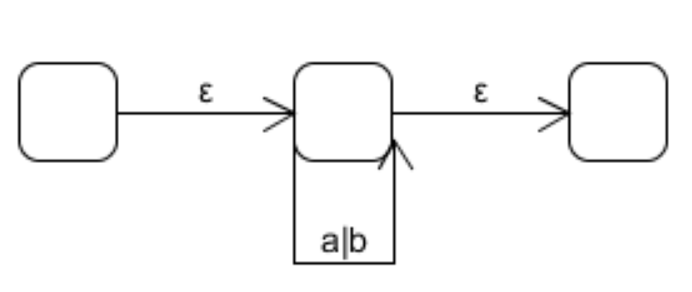
(ab)\*(a\*|b\*)(ba)\*

总共10结点，14条边（8空串边）



r = (a|b)\* a (a|b) (a|b) , 构造其NFA.





r = (a|b)\*abb, 构造其NFA.

r= (ab)\*(a\*|b\*)(ba)\*，构造其NFA.

3.7.2 NFA的模拟

检验某个字符串是否能够被某个NFA接受

输入: 一个以文件结束符eof结尾的输入串x;

一个NFA N(开始状态s0, 接受状态集合F, 转换函数move);

输出: 如果N接受x则返回"yes", 否则返回"no".

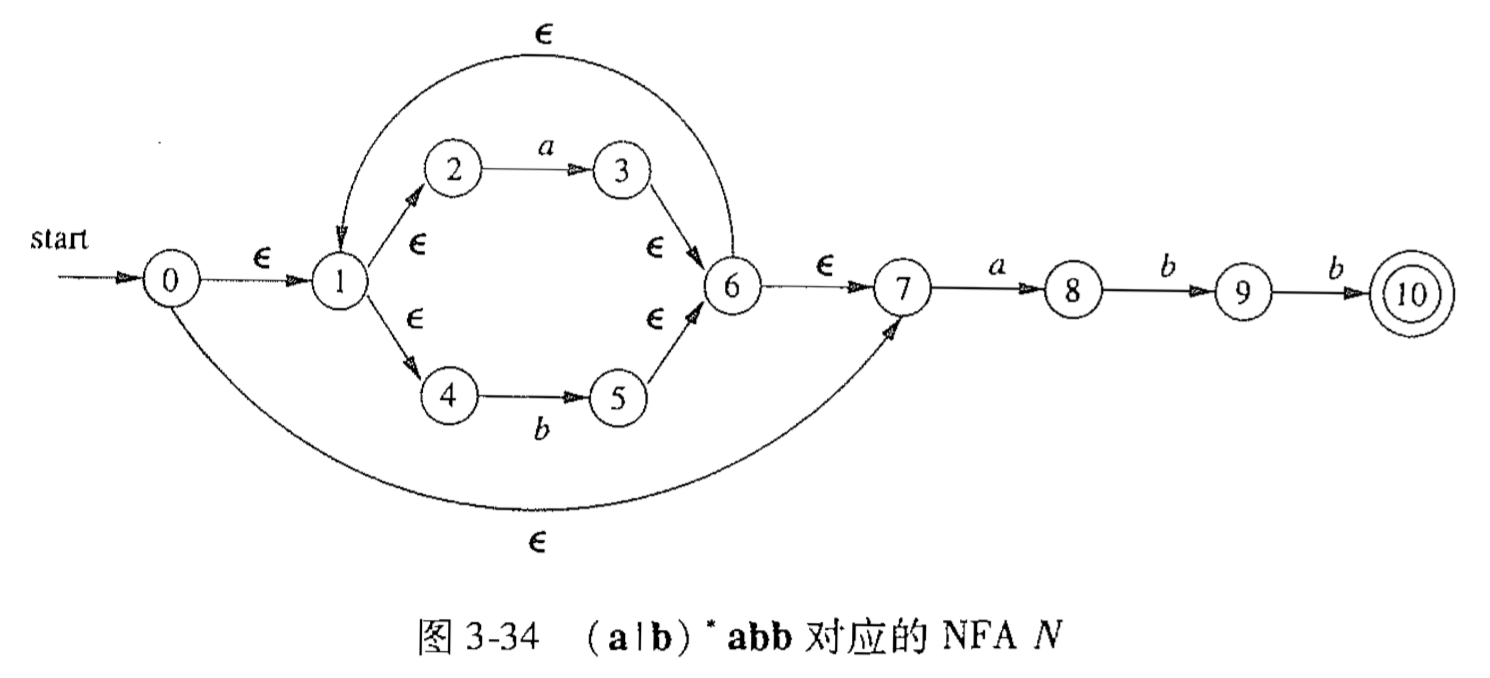
aaabb

当前指针: 用于记录待遍历的字符

当前到达的状态集合:

(1)当前指针指向非eof,并且当前状态集合为空，返回NO

(2)当前指针指向eof,



aaabb

方法： 保存一个当前状态集合S, 即那些可以从s0开始沿着标号为当前已读入输入部分的路径到达的状态的集合。如果c是函数nextChar()读到的下一个输入字符, 那么我们首先计算move(S, c), 然后使用ε-closure求出这个集合的闭包。T = move(S, c); ε-closure(T) ε-closure(move(S, c))

初始化S = ??

下个字符nextChar 状态集合S

1) S = ε-closure({s0})

2) c = nextChar()

3) while (c != eof) {

4) S = ε-closure(move(S,c));

if ( |S| = 0) return “no”

5) c = nextChar();

6) }

7) if (S交F!=空) return "yes";

8) else return "no";

算法3.22 模拟一个NFA

如何快速求 ε-closure(X) 的结果, 假设X是一个状态集合

即求X中的每个状态，经过ε可以到达的下一个状态集合

数据结构:

1) X表示一个状态集合, 用栈来表示

2) Y表示其求解结果集合, 用栈来表示

3) 布尔数组alreadyOn, 用于表示某个状态是否已经在Y中, 即alreadyOn[s] =1或0

加入某个状态s至Y中

addState(s) {

将s压入栈Y中;

alreadyOn[s] = true;

for (t on move(s, ε))

if (!alreadyOn[t])

addState(t)

}

求ε-closure(move(X,c));

alreadyOn = [false]

Y = {}

for (X中的每个s) {

for (move(s, c)中的每个t)

if (!alreadyOn[t] == false)

addState(t);

将s弹出X栈

}