2.4 从正规式到词法分析器

构造词法分析器的一般方法和步骤：

1. 用正规式对模式进行描述；
2. 为每个正规式构造一个NFA，它识别正规式所表示的正规集；
3. 将构造出的NFA转换成等价的DFA，这一过程也被称为**确定化**；
4. 优化DFA，使其状态数最少，这一过程也被称为**最小化**；
5. 从优化后的DFA构造词法分析器。

问题：

我们是从DFA构造词法分析器，为何不直接从正规式构造DFA，而要先构造NFA，然后转换为DFA?

原因：希望使用机器（工具）构造词法分析器，而机器构造需要规范的算法。

〈1〉正规式→NFA：有规范的一对一的构造算法

〈2〉NFA确定化、DFA最小化都有统一的算法

〈3〉DFA→分析器：有便于记号识别的算法

2.4.1 从正规式到NFA

算法2.2 Thompson 算法

输入　字母表∑上的正规式r

输出　接受L(r)的NFA N

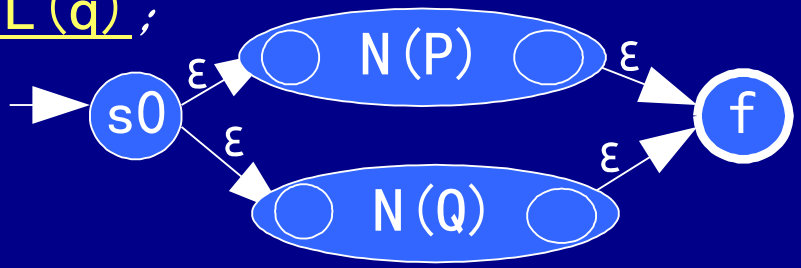
方法　首先分解r，然后根据下述步骤构造NFA:

<1> 对ε，构造NFA N(ε)如下。其中，s0为初态，f为终态，此NFA接受{ε}；

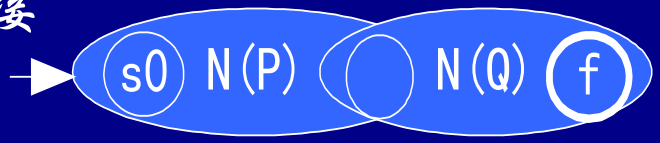
<2> 对∑上的每个字符a，构造NFA N(a)如右上，它接受{a}；

<3> 若N(p)和N(q)是正规式p和q的NFA，则

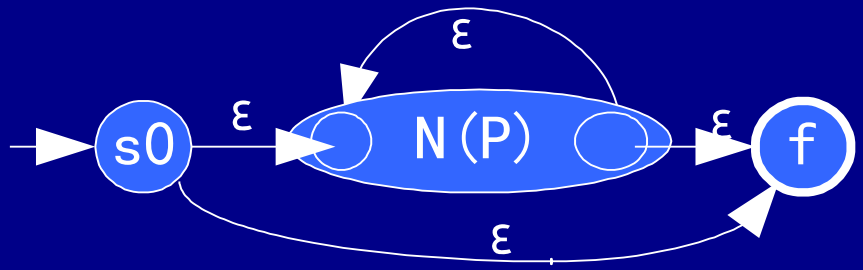
（a） 对正规式p|q，构造NFA N(p|q)如下。其中，s0为初态，f为终态，此NFA接受L(p)∪L(q)；



（b） 对正规式pq，构造NFA N(pq)如右。其中s0为初态，f为终态，此NFA接受L(p)L(q)；



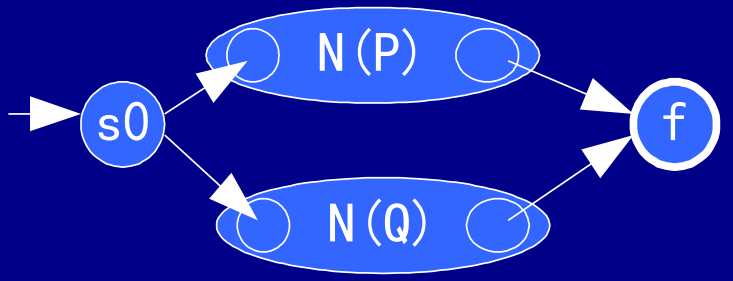
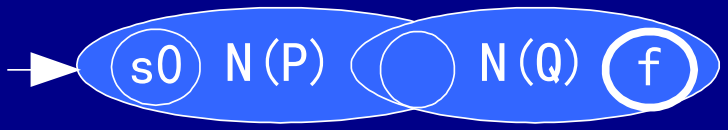
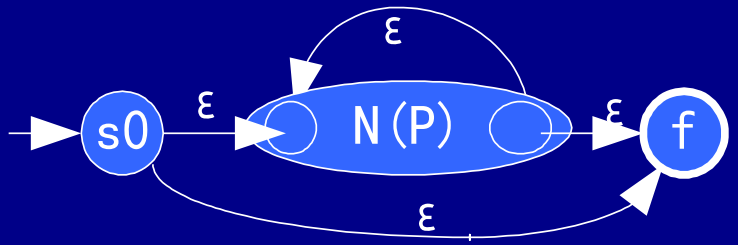
（c） 对正规式p*，构造NFA N(p*)如右。其中，s0为初态，f为终态，此NFA接受L(p*)；

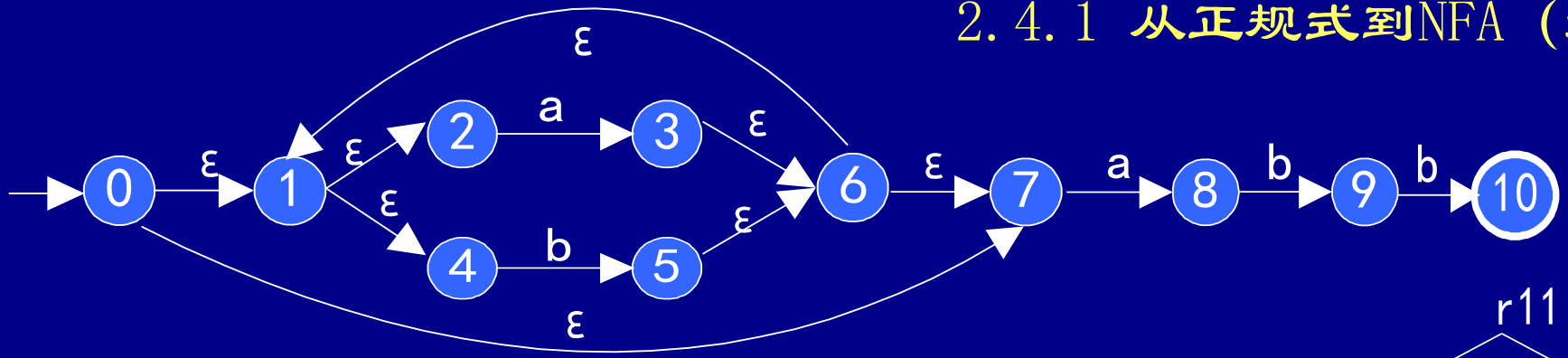


<4> 对于正规式(p)，使用p本身的NFA，不再构造新的NFA。

■

正规式与NFA的对应关系：

正规式	NFA
1. ε 表示集合L(ε)={ε}	
2. a表示集合L(a)={a}	
3. P|Q表示集合L(P) ∪ L(Q)	
4. PQ表示集合L(P)L(Q)	
5. P*表示集合(L(P))*	
6. (r) 仍然表示集合L(r)	



例2.11 用Thompson算法构造正规式

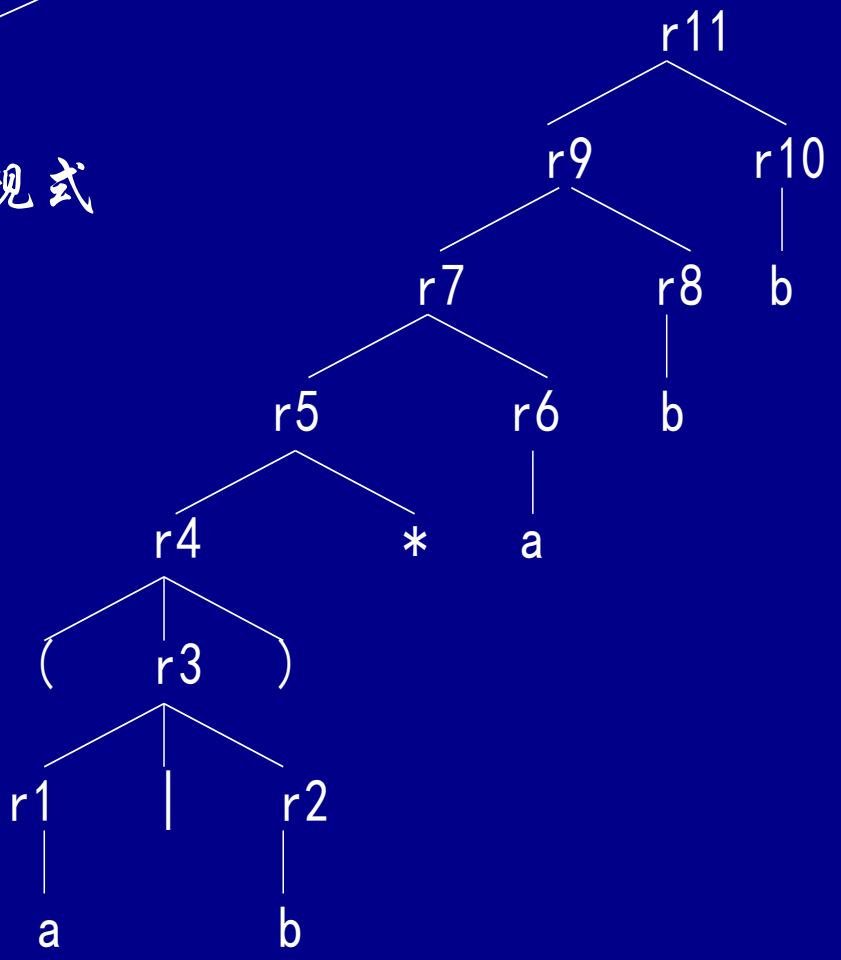
r=(a|b)*abb的NFA N(r)

<1> 分解正规式

<2> 自下而上构造NFA

强调：

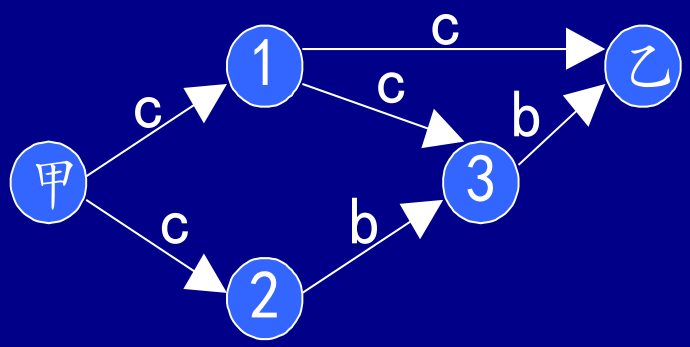
- 算法的构造与正规式一一对应
- 构造一个新的NFA最多增加两个状态



2.4.2 从NFA到DFA

<1> NFA识别记号的“并行”方法

例2.12 从甲地到乙地，可以乘小轿车也可以骑自行车，具体路线如右图。其中c表示乘车，b表示骑自行车。现在要求从甲地到乙地，只许乘车而不许骑自行车，可行吗?



问题抽象：识别是否有从甲到乙标记为全c的路径

试探（串行）：

甲 c 2　　无路可走，退回

甲 c 1 c 3　　无路可走，退回

甲 c 1 c 乙　　到达乙地，成功

假设有足够多的小汽车，每次均到达小汽车可能到达的全体

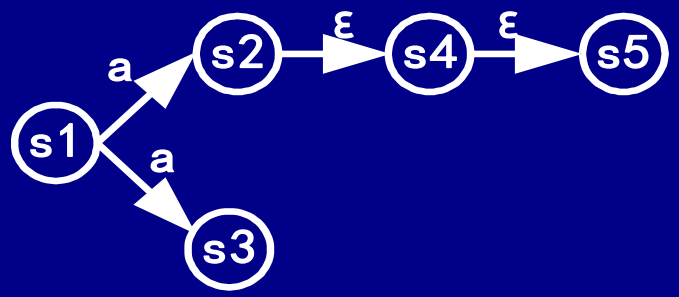
并行：

甲 c {1, 2} c {3, 乙}　　到达乙地，成功

问题：这种并行方法为什么不需要回溯?

由于并行的方法在每试探一步时，考虑了所有的下一状态转移，因此所走的每一步都是确定的。

用NFA识别记号，并不采用串行的方法（算法不易构造，复杂度高且回溯），而是采用并行的方法，核心思想是将不确定的下一状态确定化。



NFA上识别记号的确定化方法

确定化的两个步骤（回顾DFA定义）

计算下一状态转移时：

<1> 消除ε状态转移：ε-闭包(T)，状态集T的ε闭包

<2> 消除多于一个的下一状态转移：smove(S, a)，S是一个状态集，a≠ε

- smove(S, a)：从状态集S中的每个状态出发，经过标记为a的边直接到达的下一状态全体。与move(s, a)的唯一区别：用状态集取代状态
- ε-闭包(T)：从状态T出发，经过若干次ε转移到达的状态全体。

定义2.6 状态集T的ε-闭包(T)是一个状态集，且满足：

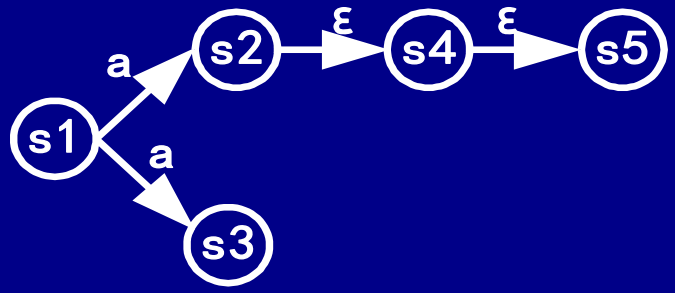
(1)T中所有状态属于ε-闭包(T)；

(2)如果t属于ε-闭包(T)且move(t, ε)=u，则u属于ε-闭包(T)；

(3)再无其他状态属于ε-闭包(T)。 ■

根据定义，ε-闭包({s2})应包括：

1.	s2自身	{s2}	(1)
2.	s4	{s2, s4}	(2)
3.	s5	{s2, s4, s5}	(2)



算法

算法2.4 求ε-闭包

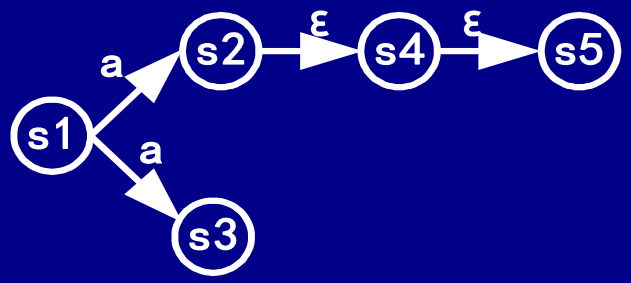
输入　状态集T。
输出　状态集T的ε-闭包
方法　用下边的函数计算ε-闭包

两个数据结构：
闭包U和模拟递归的stack

```
function ε-闭包(T) is
begin
    for T中每个状态t
    loop 加入t到U; push(t);
    end loop;
    while 栈不空
    loop pop(t);
         for 每个u=move(t, ε)
         loop if u不在U中 then 加入u到U; push(u); end if;
         end loop;
    end loop;
    return U;
end ε-闭包;
```

用算法计算ε-闭包({s2})：

	U	stack
1.	{s2}	s2
2.	{s2, s4}	s4
3.	{s2, s4, s5}	s5
4.	{s2, s4, s5}	



算法2.3 模拟NFA

输入　NFA N，x（eof），　s0（NFA初态），　F（NFA终态集）

输出　若N接受x，回答“yes”，否则“no”

方法　用下边的过程对x进行识别。S是一个状态的集合

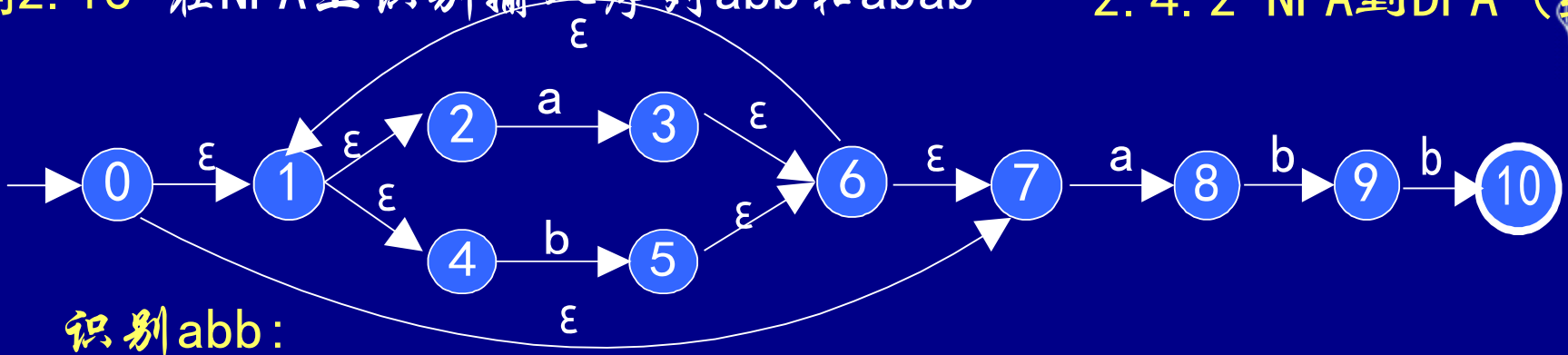
```
S := ε-闭包({s0});          -- 所有可能初态的集合
a := nextchar;
while a ≠ eof loop
    S:=ε-闭包(smove(S, a));  -- 所有下一状态的集合
    a := nextchar;
end loop;
if S∩F≠Φ then return “yes”; else return “no”;
end if;
```

■

与算法2.1的三点区别：

	模拟DFA	模拟NFA
1. 开始	初态(s0)	初态集(S)
2. 下一状态转移	下一状态	下一状态集
3. 结束	s is in F	S∩F≠Φ

例2.13 在NFA上识别输入序列abb和abab



识别abb:

1 计算初态集： ε-闭包({0}) = {0, 1, 2, 4, 7}， A

2 从A出发经a到达： ε-闭包(smove(A, a)) = {3, 8, 6, 7, 1, 2, 4}， B

3 从B出发经b到达： ε-闭包(smove(B, b)) = {5, 9, 6, 7, 1, 2, 4}， C

4 从C出发经b到达： ε-闭包(smove(C, b)) = {5, 10, 6, 7, 1, 2, 4}， D

5 结束且D∩{10}={10}，接受。识别的路径为：A a B b C b D

0 ε* A a ε* B b ε* C b ε* D 路径上的标记：ε*aε*bε*bε*=abb

识别abab:

1 初态集： ε-闭包(s0)={0, 1, 2, 4, 7}， A

2 从A出发经a到达： ε-闭包(smove(A, a)) = {3, 8, 6, 7, 1, 2, 4}， B

3 从B出发经b到达： ε-闭包(smove(B, b)) = {5, 9, 6, 7, 1, 2, 4}， C

4 从C出发经a到达： ε-闭包(smove(C, a)) = {3, 8, 6, 7, 1, 2, 4}， B

5 从B出发经b到达： ε-闭包(smove(B, b)) = {5, 9, 6, 7, 1, 2, 4}， C

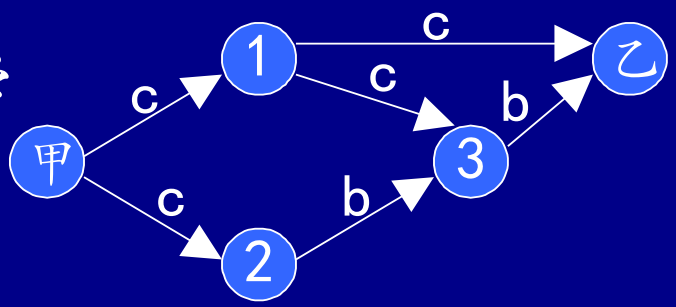
识别路径为：A a B b C a B b C。由于C∩{10}=Φ，所以不接受

“并行”模拟NFA的弱点：每次动态计算下一状态转移的集合，效率低。

改进方法：将NFA上的全部路径均确定化并且记录下来，得到与NFA等价的DFA。

回顾从甲地到乙地的路径，它的数学模型实质上是一个NFA （右上）。

可以找到一个等价的DFA（右下）。



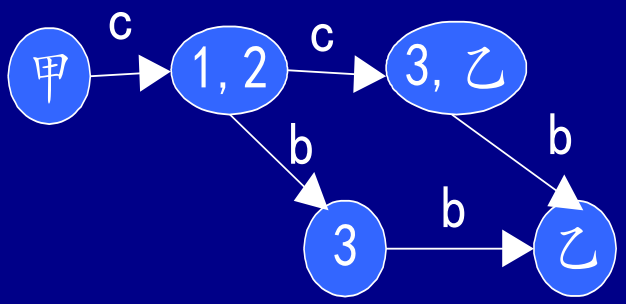
它们识别的路径均是：

cc　　ccb　　cbb

例2.14 用DFA识别cc和cbc：

甲 c {1, 2} c {3, 乙}，接受

甲 c {1, 2} b {3} c ?，不接受



优点：

1. 消除了不确定性（将NFA的下一状态集合并为一个状态）
2. 无需动态计算状态集合（针对模拟NFA的算法）

算法2.5 从NFA构造DFA（子集法）

输入　NFA N

输出　等价的DFA D。初态ε-闭包({s0})，终态是含有NFA终态的状态集合　　两个数据结构：Dstates(状态)，Dtran(状态转移)

方法　用下述过程构造DFA：

```
Dstates={ε-闭包({s0})}; //Dstates中仅有一个状态且未标记
while Dstates有尚未标记的状态T
loop   标记T;
       for   每一个输入字符a
       loop  U := ε-闭包(smove(T, a));
             if U非空
             then Dtran[T, a] := U;
                  if   U不在Dstates中
                  then U作为尚未标记的状态加入Dstates;
                  end if;
             end if;
       end loop;
end loop;
```

与算法2.3比较：记录了所有状态与状态转移

例2.15 用算法2.5构造(a|b)*abb的DFA

ε-闭包({0})={0, 1, 2, 4, 7}*		A
ε-闭包(smove(A, a))={3, 8, 6, 7, 1, 2, 4}*		B
ε-闭包(smove(A, b))={5, 6, 7, 1, 2, 4}*		C
ε-闭包(smove(B, a))={3, 8, 6, 7, 1, 2, 4}		B
ε-闭包(smove(B, b))={5, 9, 6, 7, 1, 2, 4}*		D
ε-闭包(smove(C, a))={3, 8, 6, 7, 1, 2, 4}		B
ε-闭包(smove(C, b))={5, 6, 7, 1, 2, 4}		C
ε-闭包(smove(D, a))={3, 8, 6, 7, 1, 2, 4}		B
ε-闭包(smove(D, b))={5, 10, 6, 7, 1, 2, 4}*		E
ε-闭包(smove(E, a))={3, 8, 6, 7, 1, 2, 4}		B
ε-闭包(smove(E, b))={5, 6, 7, 1, 2, 4}		C

识别abb和abab：

A a B b D b E　　接受

A a B b D a B b D　　不接受

问题：用哪个DFA识别输入序列？