

COMP130014 编译

## 第二讲：词法分析

徐辉

xuh@fudan.edu.cn



# 主要内容

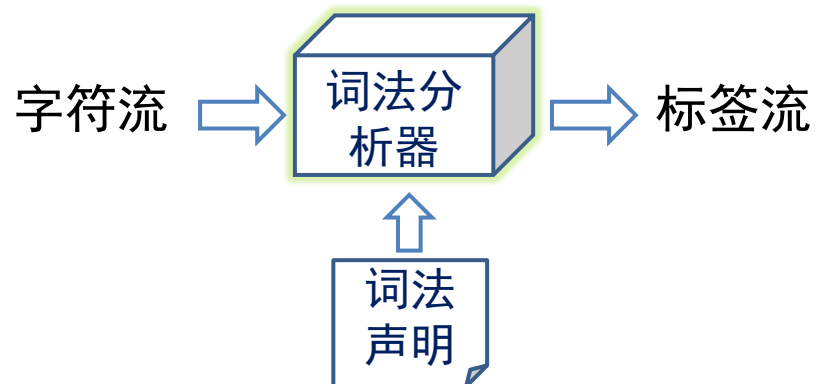
- 一、词法声明：正则表达式 (Regex)
- 二、词法声明：使用Regex声明TeaPL词法
- 三、词法解析：Regex转NFA
- 四、词法解析：NFA转DFA
- 五、正则语言及其等价性

# 一、词法声明：正则表达式

---

# 词法声明

- 词法声明定义了什么是有效的字符串输入及其对应标签类型



# 基本概念

- 模式（Pattern）：字符串模式描述，一般用正则表达式
- 词素（Lexeme）：符合某标签模式的字符串实例
- 标签（Token）：由标签类型和属性组成的二元组

类型	模式（文字描述）	词素举例	标签
二元运算符	任意加减乘除符号	+	<BINOP (+)>
操作数	任意数据常量	3.14	<NUM (3.14)>

# 正则表达式

- 正则表达式定义了字母表 $\Sigma$ 上的字符串集合
- 其单个字符元素的表述方式包括：

正则表示	含义
$a$	$x \in \{a\}$
$[ab]$	$x \in \{a, b\}$
$[a - z]$	$x \in \{a, \dots, z\}$
$[a - zA - Z]$	$x \in \{a, \dots, z, A, \dots, Z\}$
$[^a]$	$x \in \Sigma \setminus a$
$a?$	$x \in \{a, \epsilon\}$
$.$	$x \in \Sigma$
$\epsilon$	$x \in \emptyset$

# 正则表达式 (Regular Expression)

- 字符元素之间以及正则表达式之间（递归定义）的组合方法包括：

运算类型		正则表示	含义
选择 (union)		$R S$	$x \in R \cup S$
连接 (concatenation)		$RS$	$x \in \{rs \mid r \in R, s \in S\}$
闭包 (Kleene closure)		$R^*$	$x \in \{r_0 \dots r_n \mid r_i \in R, n \in \{0, 1, \dots, \infty\}\}$
	正闭包	$R^+$	$x \in \{r_0 \dots r_n \mid r_i \in R, n \in \{1, \dots, \infty\}\}$
	区间闭包	$R^{\{min, max\}}$	$x \in \{r_0 \dots r_n \mid r_i \in R, n \in \{min, \dots, max\}\}$

# 基本运算法则

- 运算符优先级：闭包 > 连接 > 选择

运算类型	交换律	结合律	分配律	幂等率
选择	$r s = s r$	$r (s t) = (r s) t$		
连接		$r(st) = (rs)t$	$r(s t) = rs rt$	
闭包				$r^* = r^{**}$



# 使用正则表达式声明词法

$\langle \text{UINT} \rangle := [0-9]^+$   
 $\langle \text{UNUM} \rangle := [0-9]^+(\cdot [0-9]^+ | \epsilon)$

## 利用中间变量简化词法声明

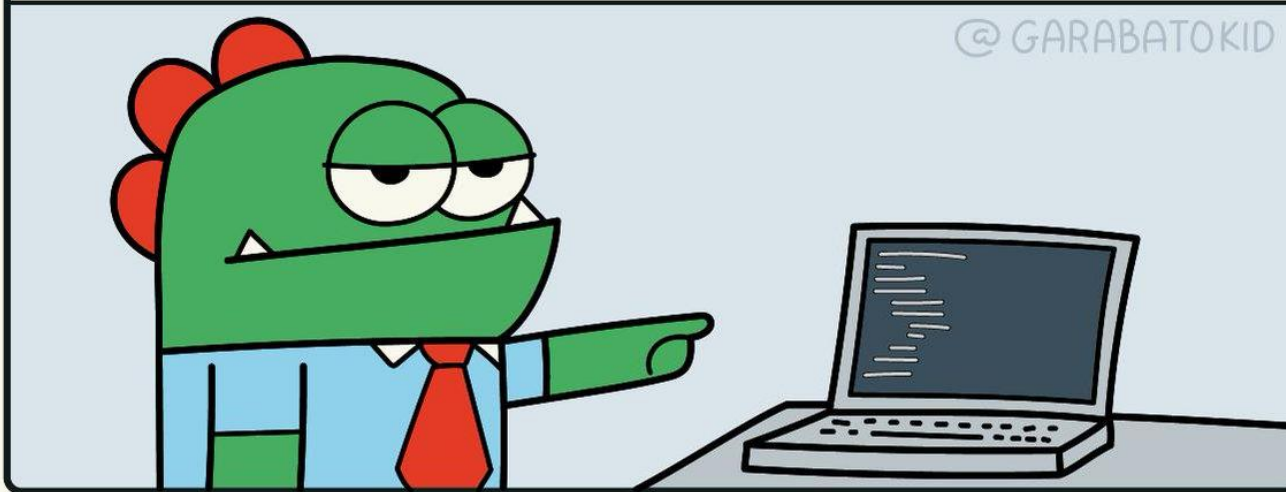
$\text{DIGIT} := [0-9]$   
 $\langle \text{UINT} \rangle := \{\text{DIGIT}\}^+$   
 $\langle \text{UNUM} \rangle := \{\text{DIGIT}\}^+(\cdot \{\text{DIGIT}\}^+ | \epsilon)$

# 练习

- 定义无符号数的正则表达式：
  - 支持浮点数和整数，如0.1、123
  - 支持科学计数法表示，如123e2、2.1e-3（指数不能为浮点数）

# HOW TO REGEX

STEP 1: OPEN YOUR FAVORITE EDITOR



STEP 2: LET YOUR CAT PLAY ON YOUR KEYBOARD





## 二、词法声明：使用Regex声明TeaPL词法

---

# 一门语言中需要定义的标签类型

- 数据：
  - 数字
  - 标识符
- 符号：
  - 运算符
  - 其它符号
- 保留字

# TeaPL中的数字和标识符

$\langle \text{UNUM} \rangle := [1-9][0-9]^* | 0$

$\langle \text{ID} \rangle := [a-zA-Z][a-zA-Z0-9]^*$

# TeaPL中的运算符

## 二元运算符

<ADD> := +  
<SUB> := -  
<MUL> := \*  
<DIV> := /

## 比较运算符

<CEQ> := ==  
<NEQ> := !=  
<GT> := >  
<GTE> := >=  
<LT> := <  
<LTE> := <=

## 逻辑运算符

<AND> := &&  
<OR> := ||  
<NOT> := !

## 赋值符号

<EQ> := =



# TeaPL中的其它符号

## TeaPL代码样式

```
fn foo(a:int, b:int) -> int {  
    return a + b;  
}
```

## 用途

<COMMA>	:=	,	分隔多个元素
<SEMI>	:=	;	分隔多条语句
<RARROW>	:=	->	函数返回类型
<COLON>	:=	:	类型声明
<DOT>	:=	.	结构体域

## 域

<LPAR>	:=	(
<RPAR>	:=	)
<LSQ>	:=	[
<RSQ>	:=	]
<LBRA>	:=	{
<RBRA>	:=	}
<DQUT>	:=	"

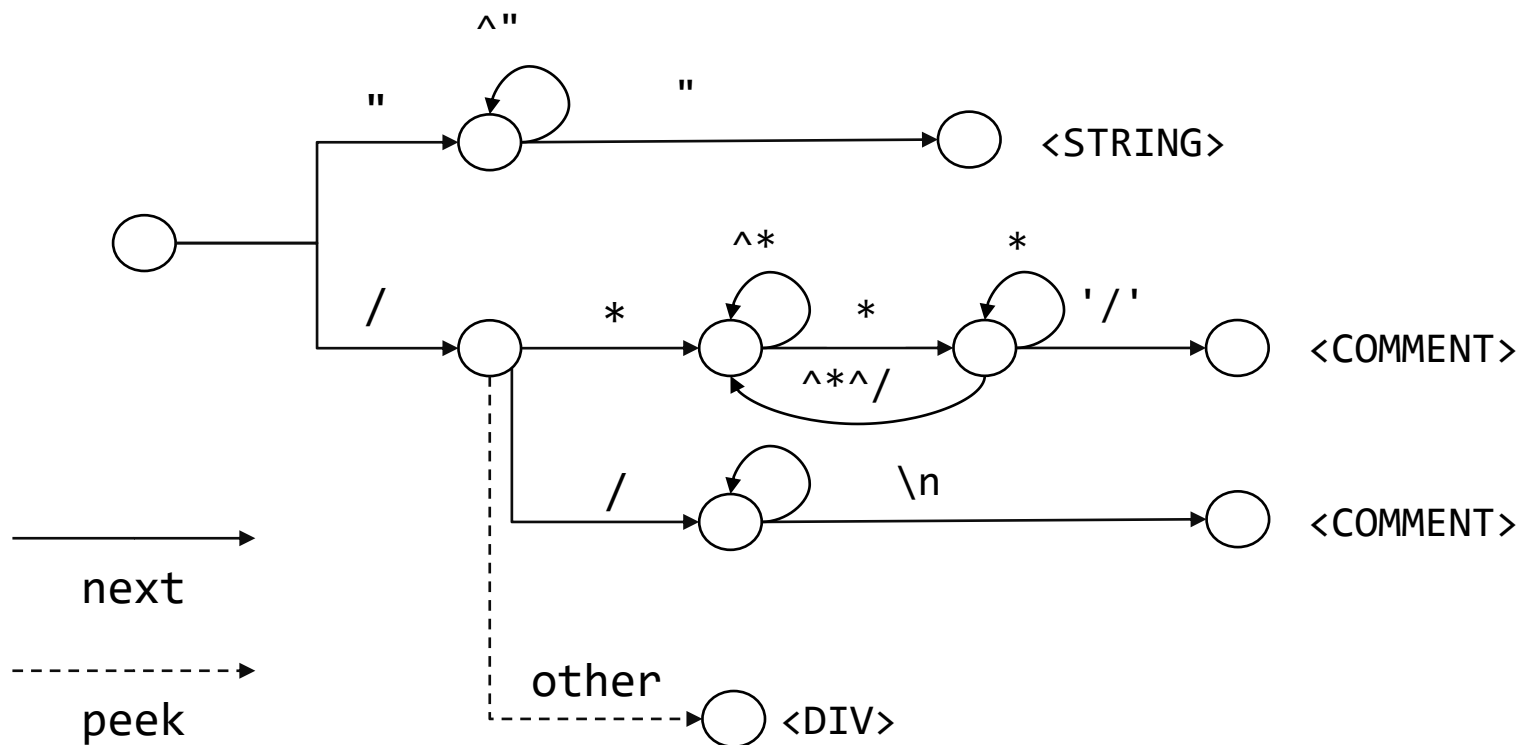
## 注释

<SLASHES>	:=	//
<LSTAR>	:=	/*
<RSTAR>	:=	*/

# 注译和引号

- 引号或注释内部的单词是否应识别为单独的标签？
- 否=>更新标签定义：

```
<STRING>  := "[^"]*"
<COMMENT>  := "//.*" | /\n.*\n/
```



# TeaPL中的保留字

## 函数、变量声明

```
<FN>   := fn  
<LET>  := let
```

## 类型

```
<INT>   := int  
<STRUCT> := struct
```

## 控制流

```
<IF>    := if  
<ELSE>   := else  
<WHILE>  := while
```

# 识别时的冲突处理

- 保留字 vs 标识符：保留字优先级高于标识符
  - 如字符串 “if” 应识别为<IF>，非<ID (if)>
- 存在多种匹配方案时，选择最长的匹配
  - 如 “<=” 应识别为<LTE>，不应识别为两个标签<LT><EQ>
  - “ifabc” 应识别为<ID (ifabc)>，不应识别为<IF>和<ID (abc)>

### 三、词法解析：Regex转NFA

---

# 有穷自动机 (Finite State Automaton)

- 识别无符号浮点数的FSA:

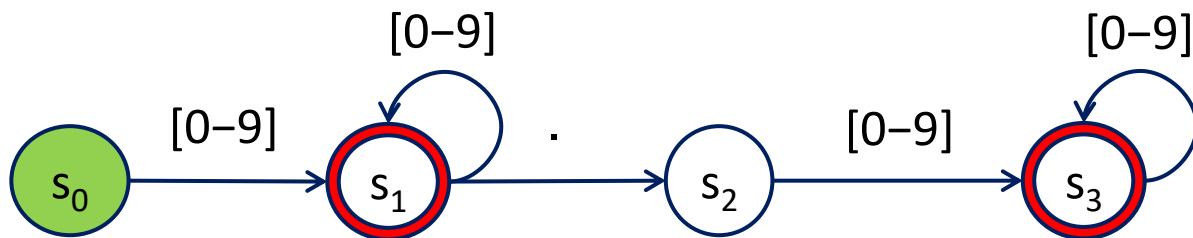
- 字符集:  $\Sigma = \{0,1,2,3,4,5,6,7,8,9,.\}$

- 状态集:  $S = \{s_0, s_1, s_2, s_3\}$

- 初始状态:  $s_0 = s_0$

- 接受状态:  $s_{acc} = \{s_1, s_3\}$

- 状态转移关系:  $\Delta = \left\{ \begin{array}{l} s_0 \xrightarrow{[0-9]} s_1, s_1 \xrightarrow{[0-9]} s_1, s_1 \xrightarrow{.} s_2 \\ s_2 \xrightarrow{[0-9]} s_3, s_3 \xrightarrow{[0-9]} s_3 \end{array} \right\}$

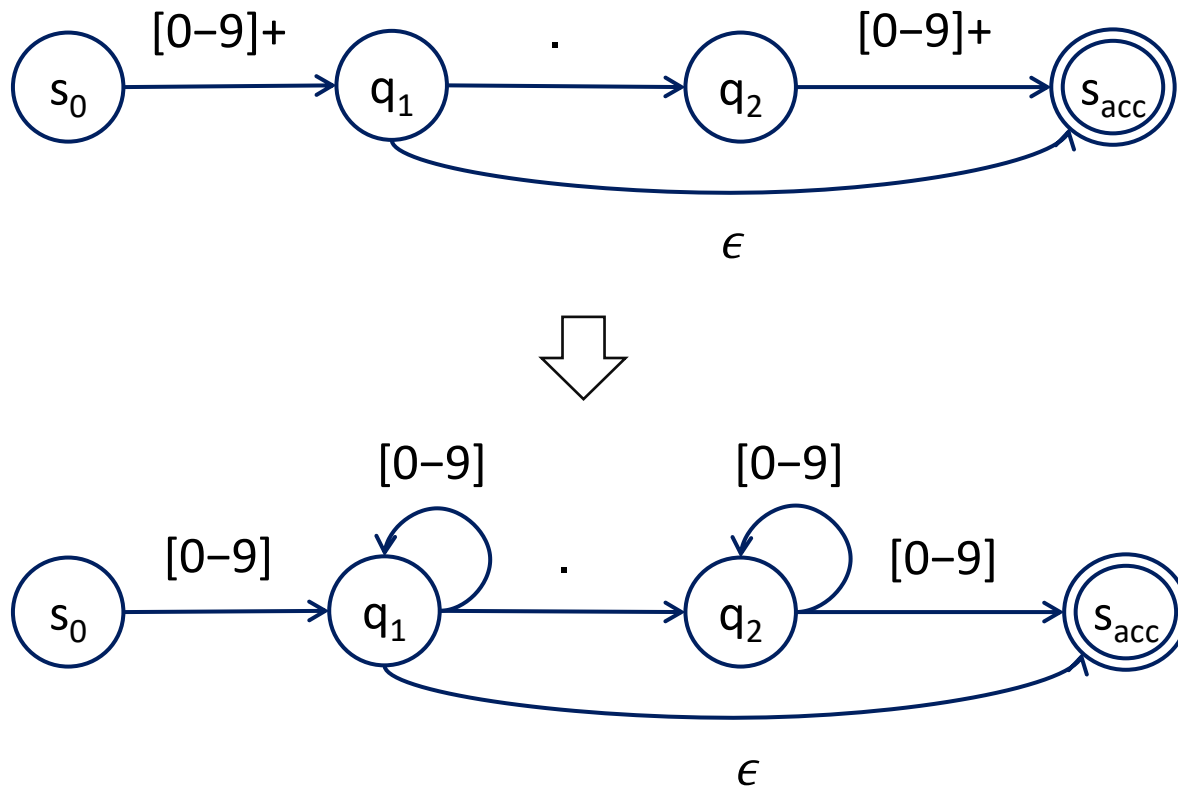


# FSA接受字符串的条件

- FSA接受字符串  $x_1 x_2 \dots x_k \mid \forall i \in \{1..k\}, x_i \in \Sigma$  的充要条件是:
  - 存在序列  $s_{t_0} s_{t_1} \dots s_{t_n} \in S$ , 其中  $s_{t_0}$  是初始状态,  $s_{t_n} \in S_{acc}$
  - 并且  $\forall s_{t_{i-1}}, x_i, s_{t_i}, (s_{t_{i-1}}, x_i, s_{t_i}) \in \Delta$
  - 即  $\delta(\dots \delta(\delta(s_{t_0}, x_1), x_2) \dots, x_n) \in S_{acc}$
- 反之, 则转移至拒绝状态  $s_{rej}$

# 如何将正则表达式转换为FSA?

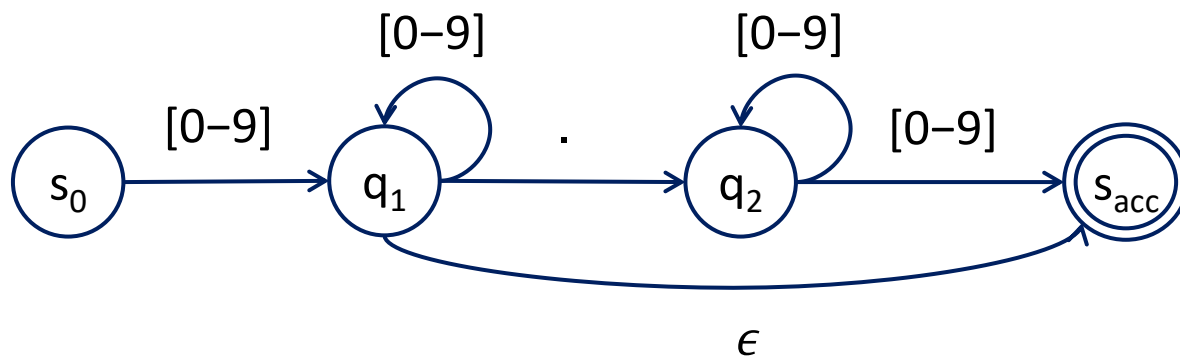
- 如何构造正则表达式  $[0-9]^+((.[0-9]^+)|\epsilon)$  对应的FSA?





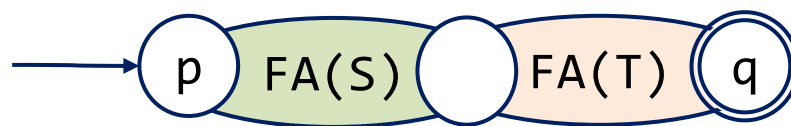
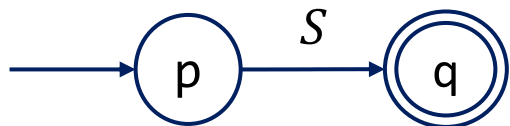
# DFA和NFA

- 确定型有穷自动机 (Deterministic FSA)
  - 对于FSA的任意一个状态和输入字符，最多只有一条状态转移边
- 非确定型有穷自动机 (Nondeterministic FSA)
  - 对于FSA的任意一个状态和输入字符，可能存在多条状态转移边

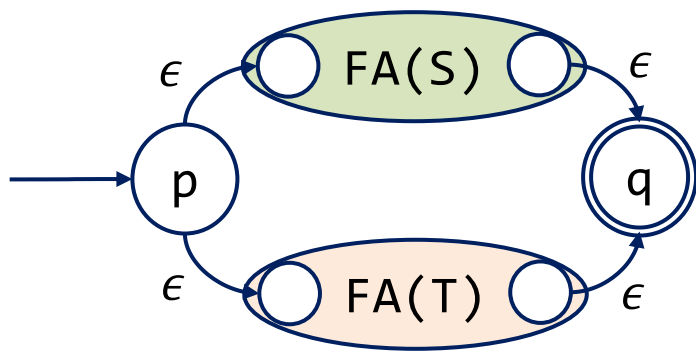


# Thompson构造法: McNaughton–Yamada–Thompson

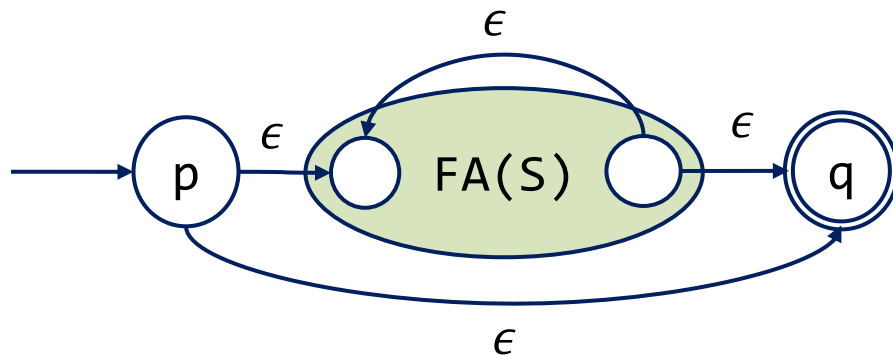
- 根据运算次序（逆序）将正则表达式递归展开
- 根据运算符匹配特定展开模式



连接:  $ST$



选择:  $S|T$

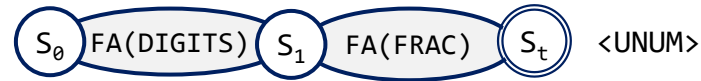


闭包:  $S^*$

# 展开过程



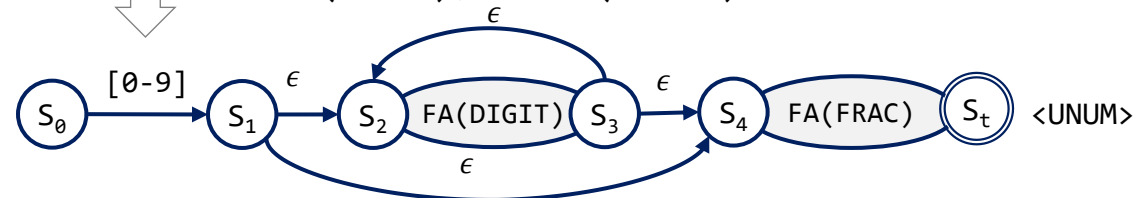
展开FA(UNUM): 连接



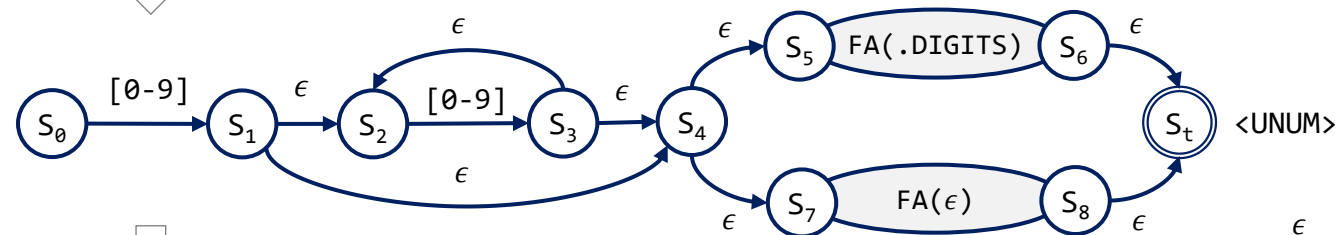
展开FA(DIGITS): 连接



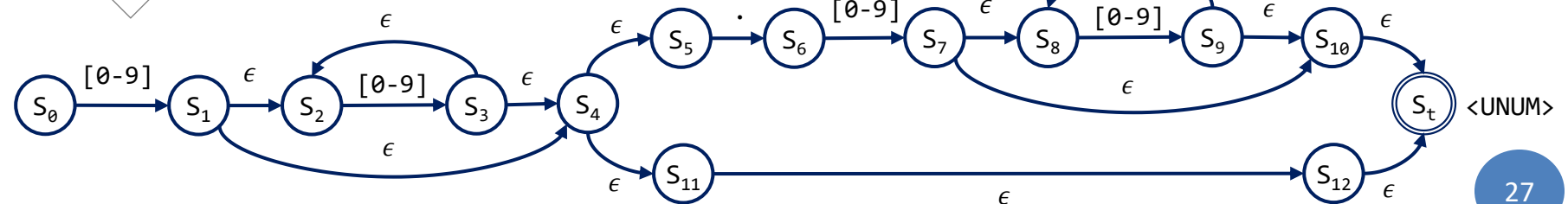
展开FA(DIGIT); 展开FA(DIGIT\*): 闭包



展开FA(DIGIT); 展开FA(FRAC): 选择



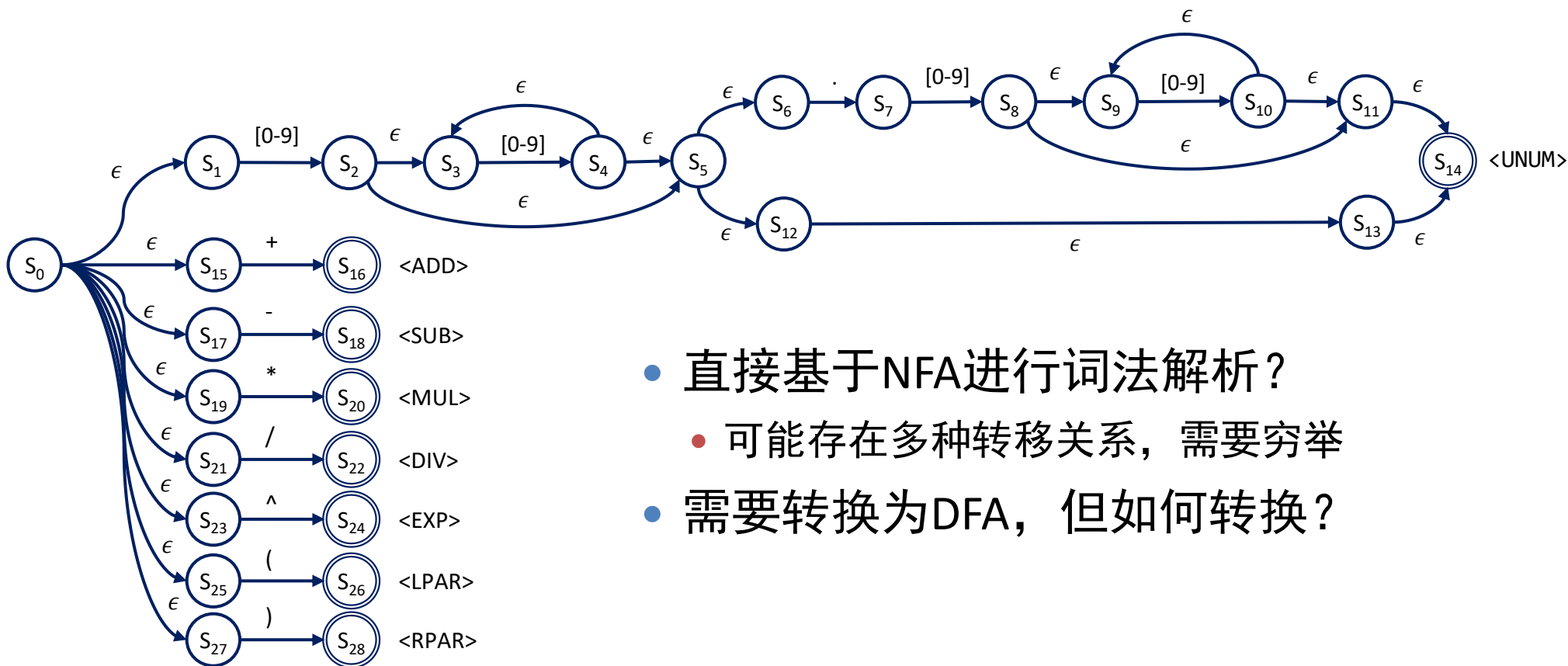
继续递归展开其余子FA



DIGIT  $\equiv [0-9]$   
 DIGITS  $\equiv \{DIGIT\} \{DIGIT\}^*$   
 FRAC  $\equiv \{DIGITS\} | \epsilon$   
 UNUM  $\equiv \{DIGITS\} \{FRAC\}$

## 如何使用一个NFA表示多个正则表达式？

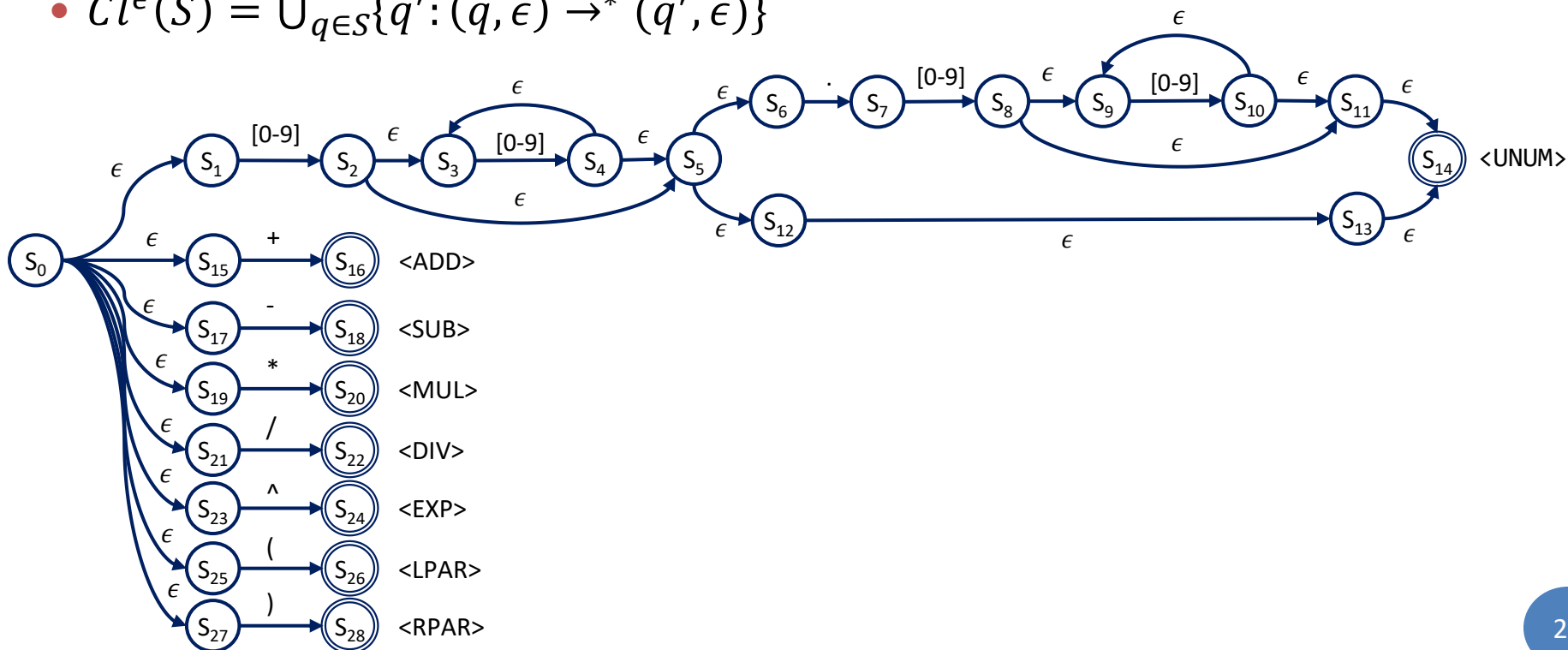
- 使用 $\epsilon$ 转移将多个正则表达式的NFA合并为一个NFA



- 直接基于NFA进行词法解析？
  - 可能存在多种转移关系，需要穷举
- 需要转换为DFA，但如何转换？

# $\epsilon$ 闭包 (closure)

- 状态 $s_i$ 的 $\epsilon$ 闭包指的是 $s_i$ 的 $\epsilon$ -transition的状态集合
  - $Cl^\epsilon(s_i) = \cup \{s_j : (s_i, \epsilon) \rightarrow^* (s_j, \epsilon)\}$
  - $Cl^\epsilon(s_0) = \{s_0, s_1, s_{15}, s_{17}, s_{19}, s_{21}, s_{23}, s_{25}, s_{27}\}$
- 状态集 $S$ 的 $\epsilon$ 闭包指的是 $S$ 中所有状态的 $\epsilon$ -transition的状态集合
  - $Cl^\epsilon(S) = \cup_{q \in S} \{q' : (q, \epsilon) \rightarrow^* (q', \epsilon)\}$

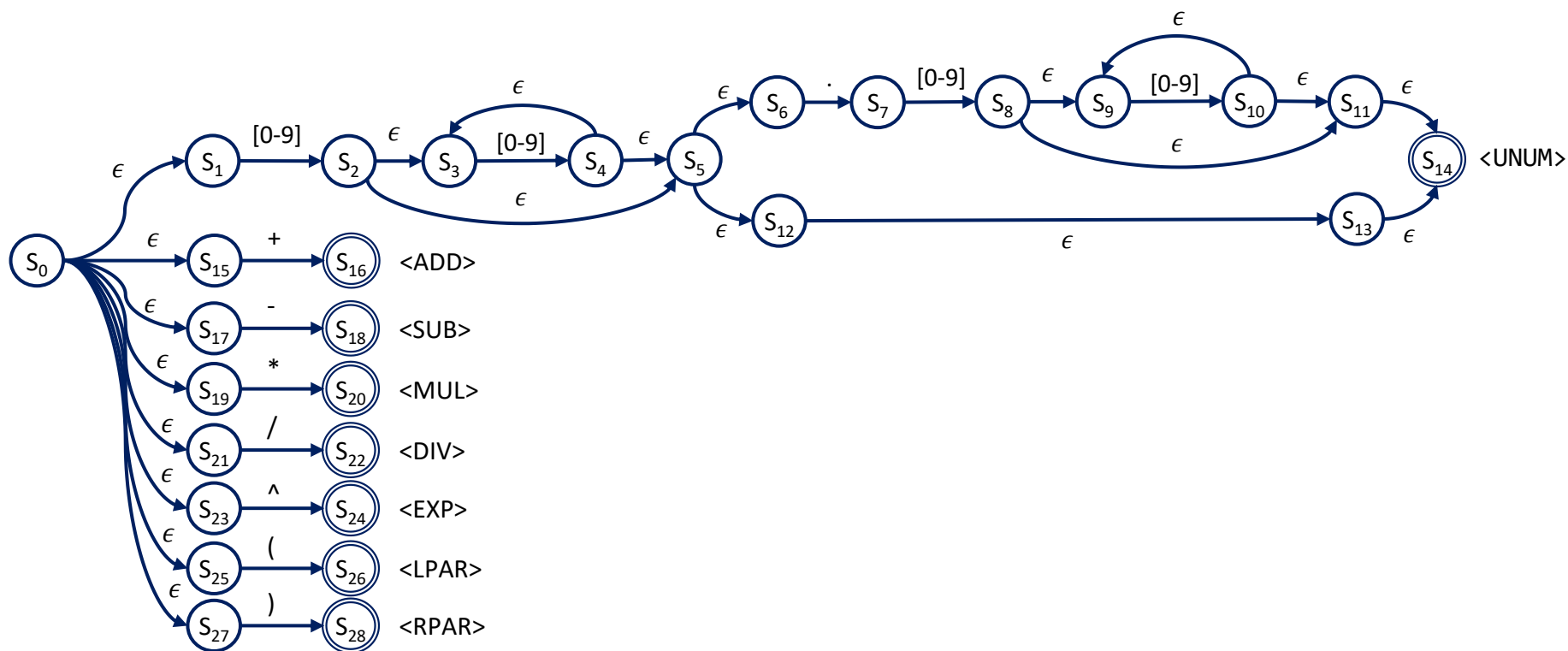


# $a$ -transition

- 状态集 $S$ 接受字符 $a$ 后状态集的 $\epsilon$ 闭包

- $\delta(S, a) = Cl^\epsilon(\cup_{q \in S} \{q' : (q, a) \rightarrow q'\})$

- $\delta(\{s_0, s_1, s_{15}, s_{17}, s_{19}, s_{21}, s_{23}, s_{25}, s_{27}\}, 0) = \{s_2, s_3, s_5, s_6, s_{12}, s_{13}, s_{14}\}$



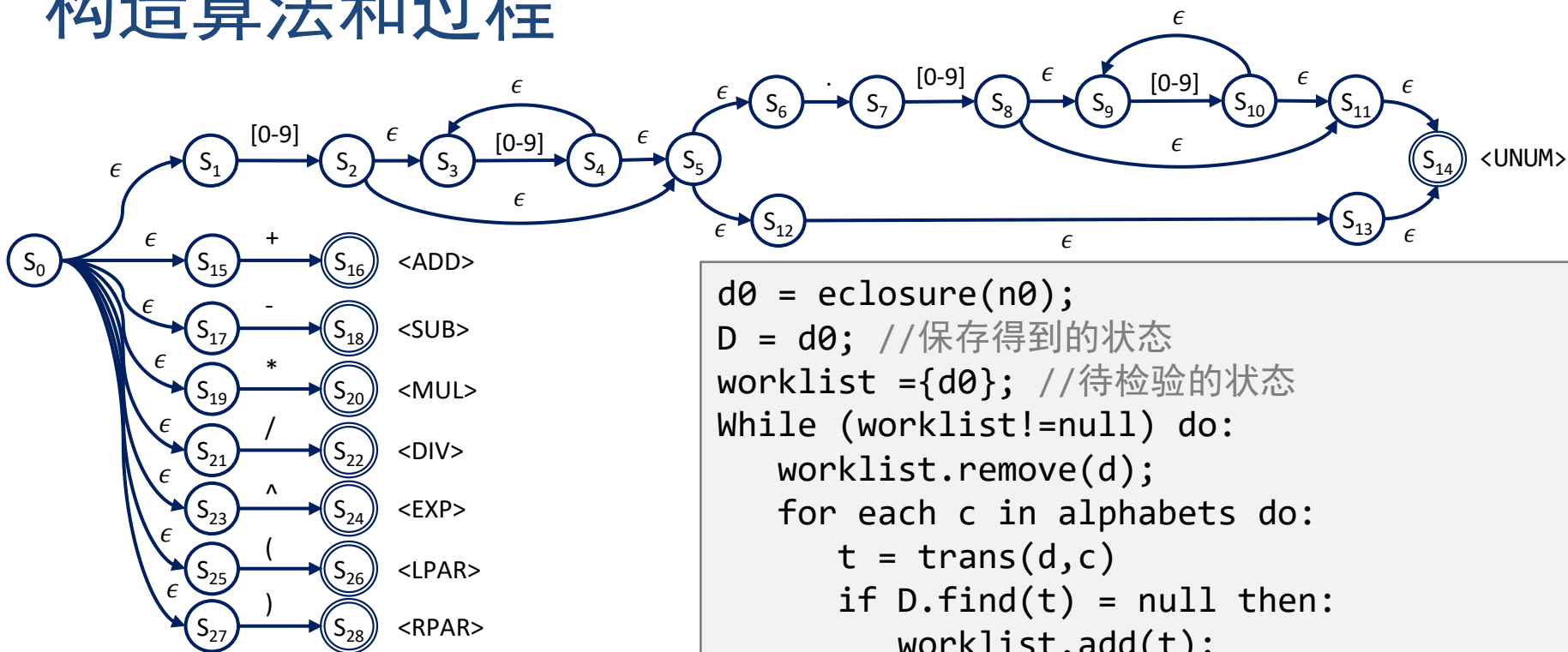
## 四、词法解析：NFA转DFA

---

- 给定一个字符集 $\Sigma$ 上的NFA  $(N, \Delta, n_0, N_{acc})$ ，它对应的可接受同一语言的DFA  $(D, \Delta', d_0, D_{acc})$ 定义如下：
  - $D$ 中的所有状态 $d_i$ 都是 $N$ 的一个子集， $D \subseteq 2^N$
  - $d_0 = Cl^\epsilon(n_0)$  //  $d_i$ 均为 $\epsilon$ 闭包
  - $\Delta' = \{d_i \times c \times d_j\}, \forall n_j \in d_j, \exists n_i \in d_i \text{ \& } c \in \Sigma, \text{ s.t. } (n_i, c, n_j) \in \Delta\}$
  - $D_{acc} = \{d_i \subseteq D \mid d_i \cap N_{acc} \neq \emptyset\}$



# 构造算法和过程



```

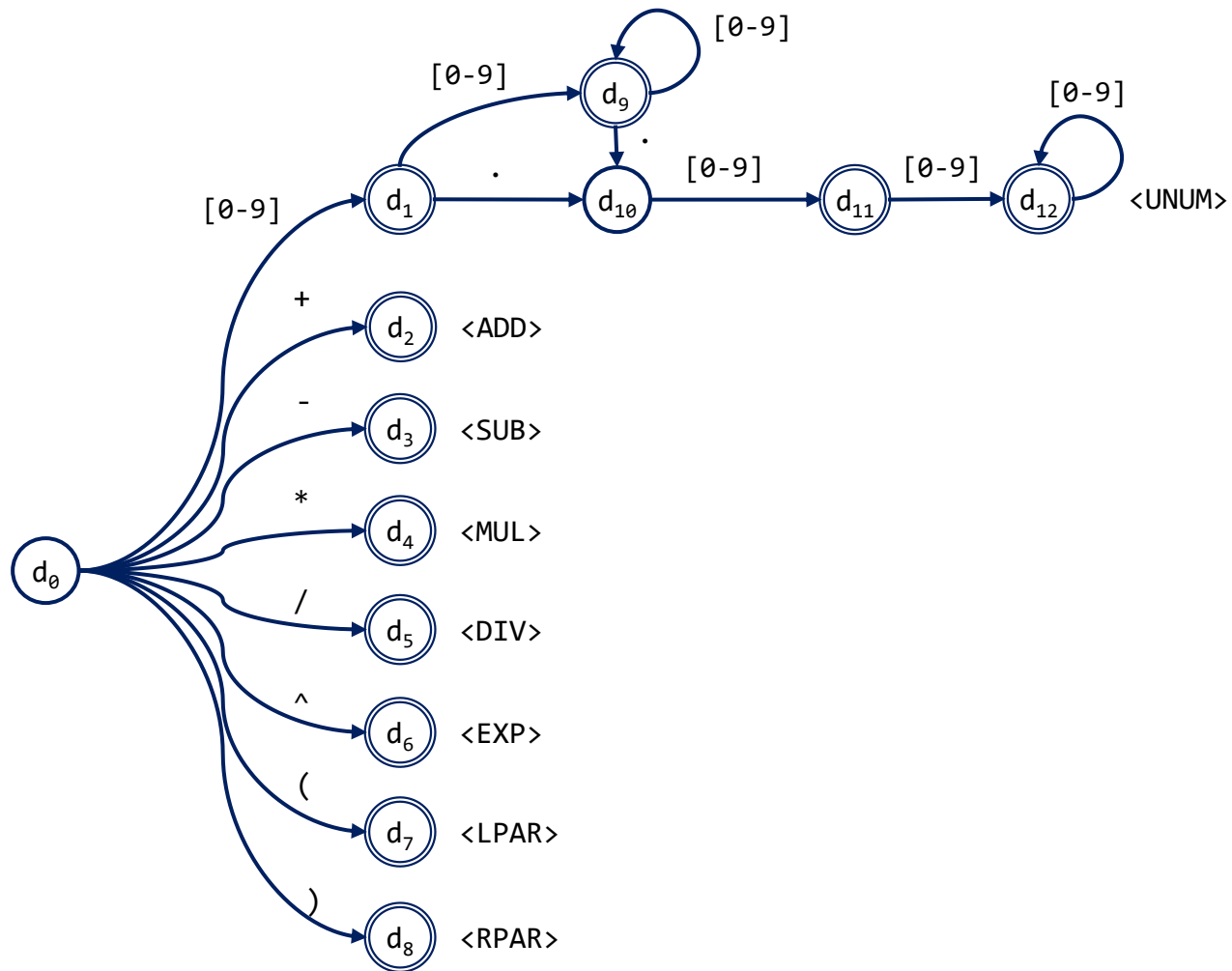
d0 = eclosure(n0);
D = d0; //保存得到的状态
worklist = {d0}; //待检验的状态
While (worklist != null) do:
    worklist.remove(d);
    for each c in alphabets do:
        t = trans(d, c)
        if D.find(t) = null then:
            worklist.add(t);
            D.add(t);
    
```

DFA状态	NFA状态集	0-9	.	+	-	*	/	^	(	)
$d_0$	$\{s_0, s_1, s_{15}, s_{17}, s_{19}, s_{21}, s_{23}, s_{25}, s_{27}\}$	$d_1: \{s_2, s_3, s_5, s_6, s_{21}, s_{23}, s_{25}, s_{27}\}$	-	$d_2: \{s_{16}\}$	$d_3: \{s_{18}\}$	$d_4: \{s_{20}\}$	$d_5: \{s_{22}\}$	$d_6: \{s_{24}\}$	$d_7: \{s_{26}\}$	$d_8: \{s_{28}\}$
$d_1$	$\{s_2, s_3, s_5, s_6, s_{21}, s_{23}, s_{25}, s_{27}\}$	$d_9: \{s_3, s_4, s_5, s_6, s_{12}, s_{13}, s_{14}\}$	$d_{10}: \{s_7\}$	-	-	-	-	-	-	-
$d_2$										

# 结果

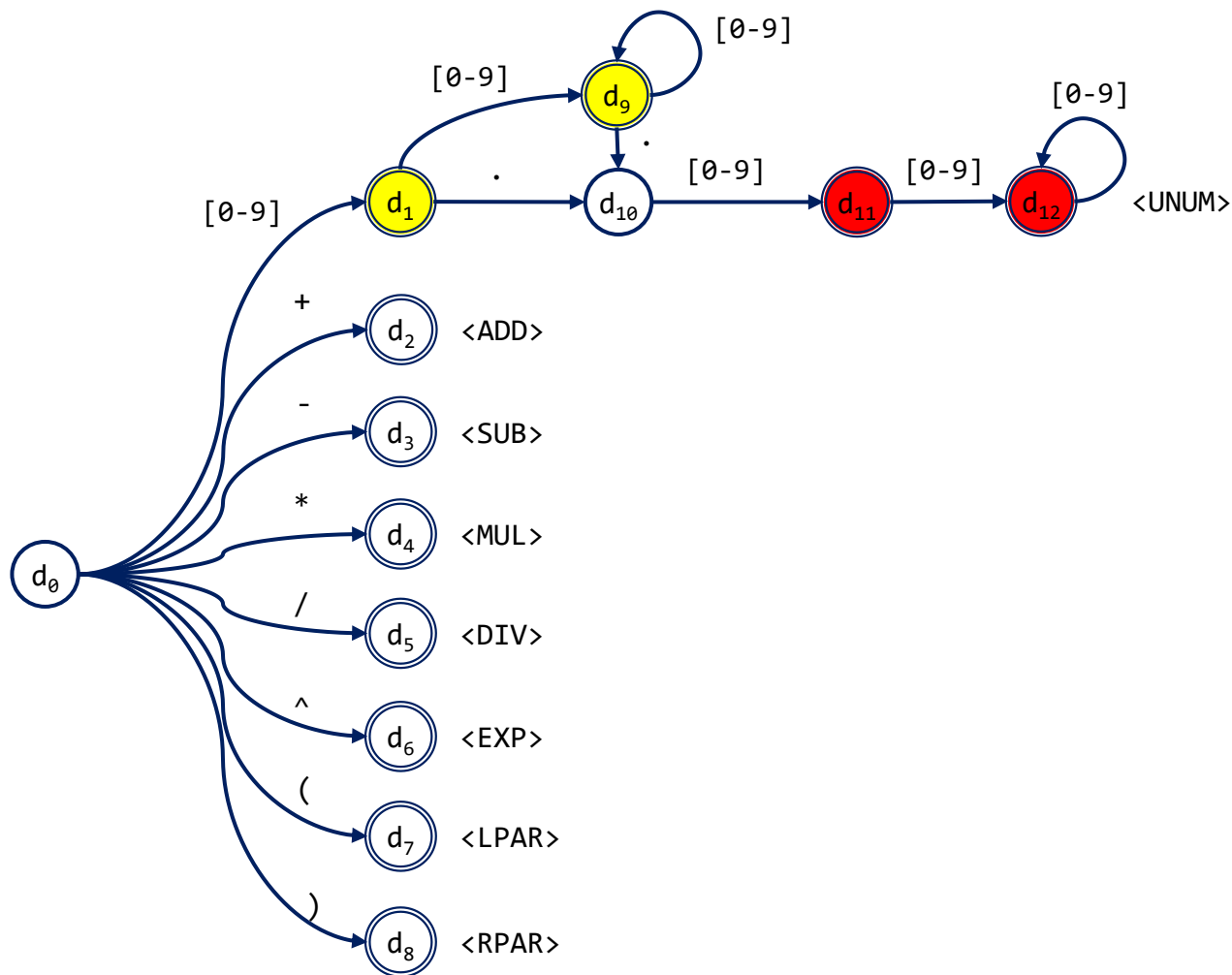
DFA状态	NFA状态集	0-9	.	+	-	*	/	^	(	)
$d_0$	$\{s_0, s_1, s_{15}, s_{17}, s_{19}, s_{21}, s_{23}, s_{25}, s_{27}\}$	$d_1$	-	$d_2$	$d_3$	$d_4$	$d_5$	$d_6$	$d_7$	$d_8$
$d_1$	$\{s_2, s_3, s_5, s_6, s_{21}, s_{23}, s_{25}, s_{27}\}$	$d_9$	$d_{10}$	-	-	-	-	-	-	-
$d_2$	$\{s_{16}\}$	-	-	-	-	-	-	-	-	-
$d_3$	$\{s_{18}\}$	-	-	-	-	-	-	-	-	-
$d_4$	$\{s_{20}\}$	-	-	-	-	-	-	-	-	-
$d_5$	$\{s_{22}\}$	-	-	-	-	-	-	-	-	-
$d_6$	$\{s_{24}\}$	-	-	-	-	-	-	-	-	-
$d_7$	$\{s_{26}\}$	-	-	-	-	-	-	-	-	-
$d_8$	$\{s_{28}\}$	-	-	-	-	-	-	-	-	-
$d_9$	$\{s_3, s_4, s_5, s_6, s_{12}, s_{13}, s_{14}\}$	$d_9$	$d_{10}$	-	-	-	-	-	-	-
$d_{10}$	$\{s_7\}$	$d_{11}$	-	-	-	-	-	-	-	-
$d_{11}$	$\{s_8, s_9, s_{11}, s_{14}\}$	$d_{12}$	-	-	-	-	-	-	-	-
$d_{12}$	$\{s_9, s_{10}, s_{11}, s_{14}\}$	$d_{12}$	-	-	-	-	-	-	-	-

# 转换后的DFA

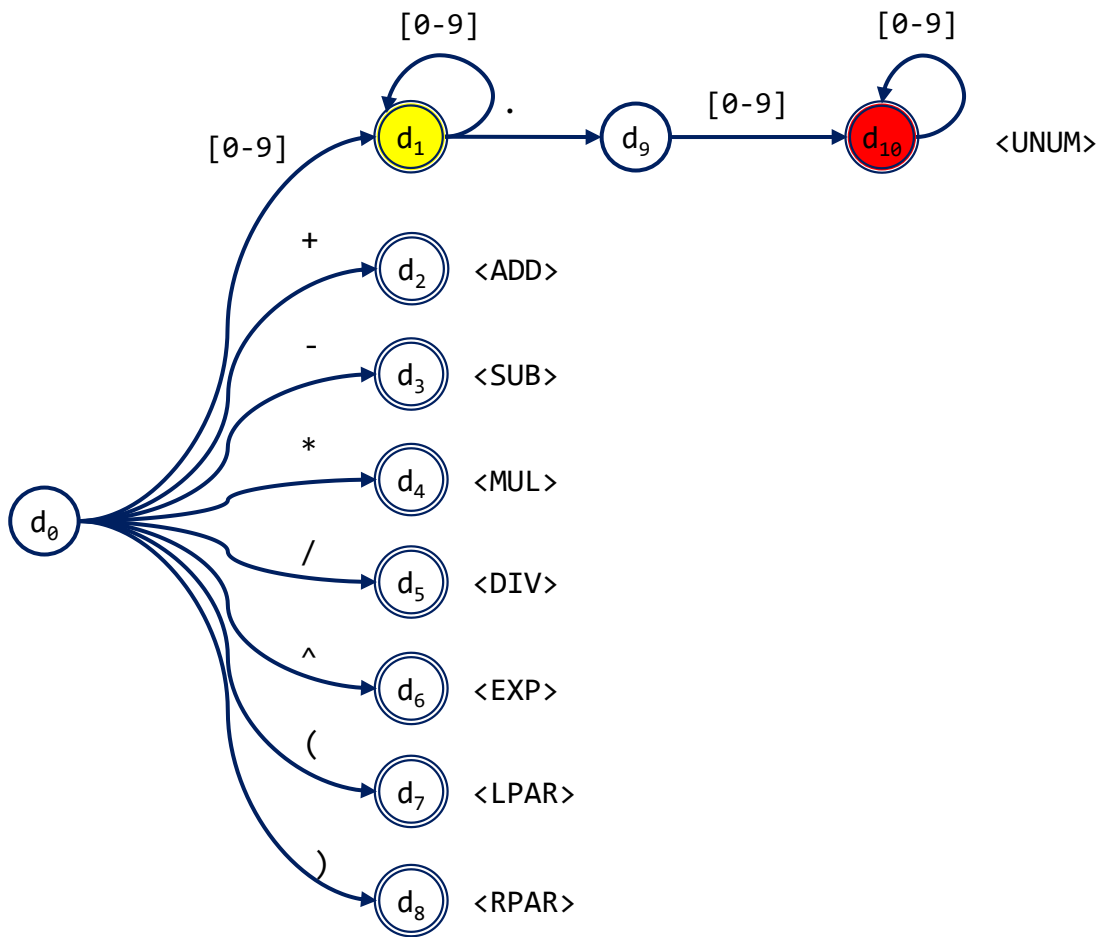


# DFA优化思路：合并同类项

- 对于两个同类型节点 $d_i$ 和 $d_j$ ，可以合并的条件是：
  - $\forall c \in \Sigma, \delta(d_i, c) = \delta(d_j, c)$



# 优化结果

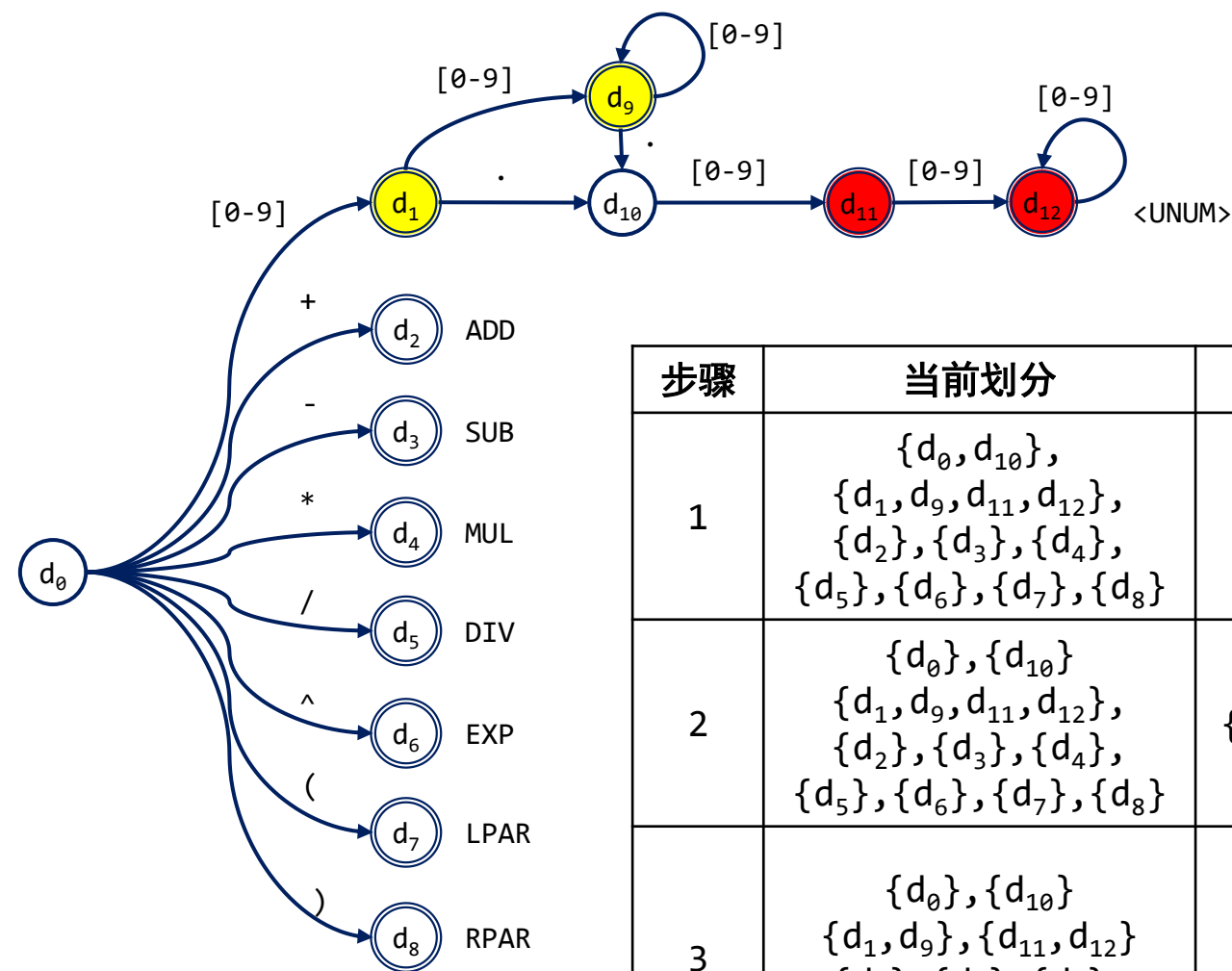


# DFA优化思路：Hopcroft分割算法

```
//将DFA的状态集合D划分为两个子集：接受状态 $D_{ac}$ 和普通状态 $D \setminus D_{ac}$ 。
D = { $D_{ac}$ ,  $D \setminus D_{ac}$ };
S = {}
While (S != D) do:
    S = D;
    D = {};
    foreach  $s_i \in S$  do:
        D = D  $\cup$  Split ( $s_i$ )
//两个节点 $d_i$ 和 $d_j$ 无需split的条件是:  $\forall c \in \Sigma, \delta(d_i, c) = \delta(d_j, c)$ 
Split(s) {
    foreach c in  $\Sigma$ 
        if c splits s into { $s_1$ ,  $s_2$ }
            return { $s_1$ ,  $s_2$ }
    return s
}
```

如果不同的接受状态分别对应不同标签应如何改进算法？

# Hopcroft分割算法应用示例



步骤	当前划分	集合	字符	Split
1	$\{d_0, d_{10}\},$ $\{d_1, d_9, d_{11}, d_{12}\},$ $\{d_2\}, \{d_3\}, \{d_4\},$ $\{d_5\}, \{d_6\}, \{d_7\}, \{d_8\}$	$\{d_0, d_{10}\}$	+	$\{d_0\}, \{d_{10}\}$
2	$\{d_0\}, \{d_{10}\}$ $\{d_1, d_9, d_{11}, d_{12}\},$ $\{d_2\}, \{d_3\}, \{d_4\},$ $\{d_5\}, \{d_6\}, \{d_7\}, \{d_8\}$	$\{d_1, d_9, d_{11}, d_{12}\}$	.	$\{d_1, d_9\},$ $\{d_{11}, d_{12}\}$
3	$\{d_0\}, \{d_{10}\}$ $\{d_1, d_9\}, \{d_{11}, d_{12}\}$ $\{d_2\}, \{d_3\}, \{d_4\},$ $\{d_5\}, \{d_6\}, \{d_7\}, \{d_8\}$			

# NFA/DFA复杂度分析

- 对于正则表达式 $r$ 来说，如采用Thompson构造法：
  - NFA状态数 $\leq |2r|$ ，边数 $\leq |4r|$
  - 解析单个词素 $x$ 的时间复杂度为 $O(|x| \times |r|)$
- 如果转化为DFA：
  - 对应DFA的状态数 $\leq |2^{|2r|}|$ 个
  - 解析单个词素的时间复杂度为 $O(|x|)$
- 结论：
  - NFA构造较快，但运行效率低
  - DFA构造耗时，但运行效率高



# 练习

- 1) 使用Thompson构造法将下列正则表达式转化为NFA
- 2) 应用子集构造法将NFA转化为DFA
- 3) 化简上一步得到的DFA

$\langle \text{UNUM} \rangle := [1-9][0-9]^* | 0$

$\langle \text{ID} \rangle := [a-zA-Z][a-zA-Z0-9]^*$

## 五、正则语言及其等价性

---

# 正则集

- 假设  $\Sigma = \{a, b\}$ , 则
  - $a|b$ 表示的语言为:  $\{a, b\}$  (称为正则集)
  - $(a|b)(a|b)$ 表示的语言为:  $\{aa, ab, bb, ba\}$
  - $a^*$ 表示的语言为:  $\{\epsilon, a, aa, aaa, \dots\}$
  - $(a|b)^*$ 表示的语言为:  $\{\epsilon, a, b, aa, ab, ba, \dots\}$
  - $a|a^*b$ 表示的语言为:  $\{a, aab, aaab, \dots\}$

# 正则语言及其等价性

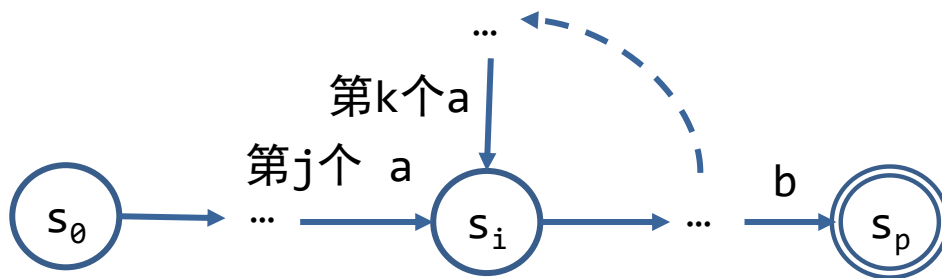
- 正则表达式是一种（表达能力有限的）语言描述方法
- 可用正则表达式描述的语言称为正则语言
- 正则集相等的两个正则表达式等价，如：
  - $a|b = b|a$
  - $(a|b)^* = (a^*|b^*)^*$

# 练习

- 分析下列正则表达式是否等价？
  - $a^*(a|b)^*a$
  - $((\epsilon|a)b^*)^*$
  - $b^*(abb^*)^*(a|\epsilon)$

# 非正则语言

- 不能用正则表达式或有穷自动机表示的语言
- $L = \{a^n b^n, n > 0\}$  不是正则语言；反证法：
  - 假设DFA可识别该语言，其包含 $p$ 个状态
  - 假设某词素为 $a^q b^q, q > p$
  - 识别该词素需要经过某状态 $s_i$ 至少两次，分别对应第 $j$ 和第 $k$ 个 $a$
  - 该DFA可同时接受 $a^q b^q$ 和 $a^{q-k+j} b^q$ ，推出矛盾
- 结论：正则语言不能计数；不能处理括号匹配问题： $(^*)^*$



# 正则语言的泵引理 (Pumping Lemma)

- 词素数量有限的语言一定是正则语言
- 词素数量无穷多的语言是否为正则语言？
- 某语言 $L(r)$ 是正则语言的必要条件：
  - 任意长度超过 $p$ （泵长）的句子都可以被分解为 $xyz$ 的形式
  - 其中 $x$ 和 $z$ 可为空
  - 子句 $y$ 被重复任意次（如 $xyyz$ ）后得到的句子仍属于该语言

