



第四章 词法分析2024

赵世瑜



- 通过扫描源程序逐一识别出每个词法单元。
- 源程序是一个符号串，字母表一般为ASCII字符集；
- 词法记号串也是一个符号串，字母表是？每个词法记号都是某个词法单元的符号表示。
- 从词法分析原理反映出如何拆分源程序符号串、如何识别为词法单元、如何表示为词法记号。
- 能够手写或自动生成词法分析器。

C2.1 词法分析原理

- 源程序是大字符集ASCII上的一个符号串w。
- 词法单元是w的子串x及其所属的语言。
- 子串x所属的语言都是已知的，即根据源语言而设计好的。

2	3	4		*		(1	1		+	-	2	2)							
---	---	---	--	---	--	---	---	---	--	---	---	---	---	---	--	--	--	--	--	--	--

NUM(234) MUL LPAR NUM(11) ADD NUM(-22) RPAR

- 词法分析任务：确定源程序w中每一个子串x及其所属语言。
- 示例确定出7个子串：234, *, (, 11, +, -22,)
- 所属语言5个：NUM, MUL, LPR, ADD, RPR



```
while (i <= 100) {  
    s += a[i]; /* form vector total */  
    i++;  
}
```

- while (reserved word), (, i (identifier), <=, 100 (integer constant),), {, s (identifier), +=, a (identifier), [, i (identifier),], ;; i (identifier), ++, ;; }
- 12个语言: reserved word; (; identifier; <=; integer constant;); { ; +=; [;]; ++; ;。
- 18个词法单元。
- 空白与注解起分隔记号的作用, 词法分析忽略它, 而不会识别为记号。
- 双字符记号<=、+=、++不要与单字符记号<、=、+混淆。
- 识别标识符的DFA也识别保留字, 但所有保留字是已知的, 故可区分。



```
while (i <= 100) {
    s += a[i]; /* form vector total */
    i++;
}
```

```
while (i <= 100) {\n\t s += a[i]; /* form vector total */\n\t i++;\n}
(WHILE,-) ((,-) (ID,i) (<=,-) (INT,100) (,,-) ({,-) (ID,s) (+=,-) (ID,a) ([,-)
(ID,i) (],,-) (;,-) (ID,i) (++,-) (;,-) (},-)
```

- 输入是ASCII字符串，输出是记号串（或称单词符号串）
- 记号是类别和值组成的二元组，类别是语言的名字，由于所有语言都是已知的，所以可命名。特别地，单元素语言可以用元素指代它。

```
(WHILE,-) (LPA,-) (ID,i) (LE,-) (NUM,100) (RPA,-) (LBR,-) (ID,s) (AAS,-)
(ID,a) (LBK,-) (ID,i) (RBK,-) (SCO,-) (ID,i) (AAA,-) (SCO,-) (RBR,-)
```

- 值是子串表示，可以是子串名、或子串值、或存储位置。
- 比如(NUM,100)中的100可以是数值100也可是字符串“100”
- 比如(ID,i)中i可以是“i”也可以是i在符号表中的登记项。



```
while (i <= 100) {
    s += a[i]; /* form vector total */
    i++;
}
```

```
(WHILE,-) (LPA,-) (ID,i) (LE,-) (NUM,100) (RPA,-) (LBR,-) (ID,s) (AAS,-)
(ID,a) (LBK,-) (ID,i) (RBK,-) (SCO,-) (ID,i) (AAA,-) (SCO,-) (RBR,-)
```

- 全体一种：把某个符号串集合命名为语言LNAME，它的每个元素是一个类别为LNAME的词法单元。如ID，INT
- 一符一种：单元素语言的词法单元只有类别，为语言名字，值省略。
- 只能设计为全体一种的语言：元素可有近无穷个。比如数、标识符等
- 对于有穷语言的记号设计，两种都可以，取决于效果。
 - 关键字 (WHILE,-) (KEY, while)
 - 运算符 (LE,-) (ROP,<=)
 - 括号 (LBR,-) (BRACE, [)

类别	类别名字	语言定义
Identifiers	ID	$[a-zA-Z_][a-zA-Z_0-9]^*$
Integer constants	INT	$[+-]?[0-9]^+$
Character constants	CHR	$'([a-zA-Z0-9] \backslash[a-zA-Z])'$
Floating constants	FLOAT	$[+-]?[0-9]^*\.[0-9][0-9]^* [-+]?[0-9][0-9]^*\.[0-9]^*?'$
String constants	STRING	$"([a-zA-Z0-9] \backslash[a-zA-Z])^*"'$
Operator tokens	ADD, MUL, ...	$+, *, <, <=, =, \wedge, \vee, \neg, \dots$
Assignments	ASG, AASG, ...	$=, +=, ++, \dots$
括号	LPAR, LBK, ...	$(), [], \{ \}, ..$
Keywords	IF, INT, ...	$\text{while, int, if, else, return, } \dots$



- 该忽略的忽略，该处理的处理，因为它们不是记号。

类别	Examples
comment	<code>/* ignored */</code>
preprocessor directive	<code>#include <foo.h></code>
	<code>#define NUMS 5, 6</code>
macro	<code>NUMS</code>
whitespace	<code>\t \n \b</code>



<源程序> → (<文本行>\n | <文本行><注释行>\n | <注释行>\n)+<EOF>

<文本行> → <预处理命令行>

<预处理命令行> → <宏定义> | <文件包含> | <编译指示>

<宏定义> → #define<不含\n的串>

<文件包含> → #include<不含\n的串>

<编译指示> → #pragma<不含\n的串>

<续行> → <文本行>\\

<文本行> → <续行>

<文本行> → (<分隔> <有效串>)+ | <有效串>(<分隔> <有效串>)*

<分隔> → (<注释块> | <空白>)+

<注释行> → //<不含\n串>

<注释块> → /*<不含*/串>*/

<空白> → (\t | \r | <空格>)+

<有效串> → (r₁ | ... | r_n)⁺



$\langle \text{源程序} \rangle \rightarrow (\langle \text{有效串} \rangle \langle \text{分隔符} \rangle)^+ \langle \text{EOF} \rangle$

$\langle \text{有效串} \rangle \rightarrow (r_1 \mid \dots \mid r_n)^+$

$\langle \text{分隔符} \rangle$ 可以是任意感兴趣的且不在 $\cup_{1 \leq i \leq n} \Sigma_i$ 中的符号。

扫描一个 $\langle \text{有效串} \rangle$ 返回一个记号串，直到遇到 $\langle \text{EOF} \rangle$ ，依次将各记号串连接作为词法分析结果。

逐一扫描<有效串>#直到EOF结束。

<有效串> $\rightarrow (r_1 | \dots | r_n)^+$

return21# \rightarrow (ID, 'r')(ID, 'eturn21) \times

return21# \rightarrow (return, _)(NUM 21) \times

return21# \rightarrow (ID, 'return21) \checkmark

21return# \rightarrow (NUM, 21)(RETURN, _) \checkmark

2+3# \rightarrow (NUM, 2)(ADD, _)(NUM, 3) $? \checkmark$

2+3# \rightarrow (NUM, 2)(NUM, +3) $? \times$

+15.07 \rightarrow (ADD, _)(NUM, 15)(FLO, .07) \times

+15.07 \rightarrow (NUM, +15)(FLO, .07) \times

+15.07 \rightarrow (ADD, _)(FLO, 15.07) $? \checkmark$

+15.07 \rightarrow (FLO, +15.07) $? \checkmark$

+N15.07 \rightarrow (ADD, _)(ID, 'N')(FLO, 15.07) \times

+N15.07 \rightarrow (ADD, _)(ID, 'N15')(FLO, .07) \checkmark

一些情形可能不在实际语言中出现；但实际语言服从此原理。这为学生打开了思维空间

➤ 要解决的问题：

➤ ① 剩余<有效串>的每一个前缀就是一个拆分方案，哪一个才是词法单元？

- 前缀最大化原则

➤ ② 拆分出的前缀可能属于多个语言，其类别选哪个？

- 事实优先级

➤ ③ 当前缀被识别为词法单元后，表示为一个词法记号，接着从剩余串中继续①和②直到剩余串为#结束。

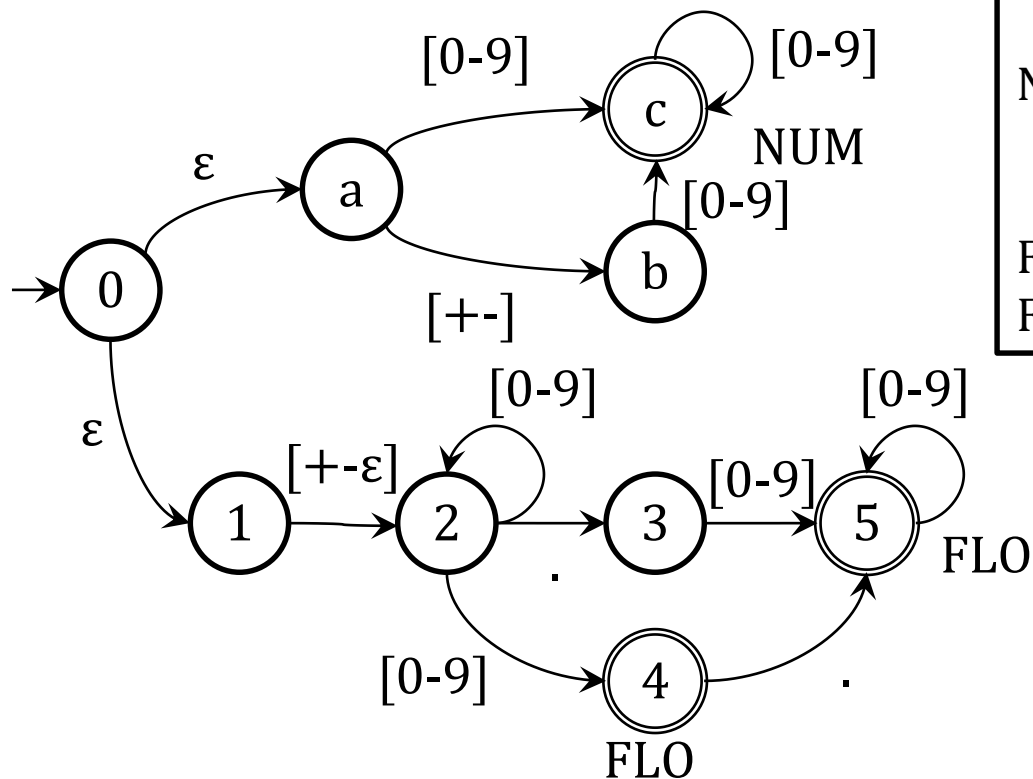
➤ ④ 输出的记号串与<有效串>保持次序一致。可并行加速。



- 假定有 n 个语言依次定义为 r_1, \dots, r_n
- $\langle \text{有效串} \rangle \in L((r_1 | \dots | r_n)^+)$
- 为每个 r_i 构建DFA A_i , 初始状态为 q_i 。每个接受状态都带有标记 L_i 。
- 构建多语言联合NFA, 从初始状态 q_0 都有 ϵ 弧射出到 q_i 。
- 确定化为一个多语言联合DFA, 是 σ_DFA 。 σ_DFA 的每个接受状态的标记这样确定, 如果它包含多个NFA接受状态, 那么选优先级最高的那个语言作为标记
- 采用事实优先级, 即下标较小者优先级较高。

例1

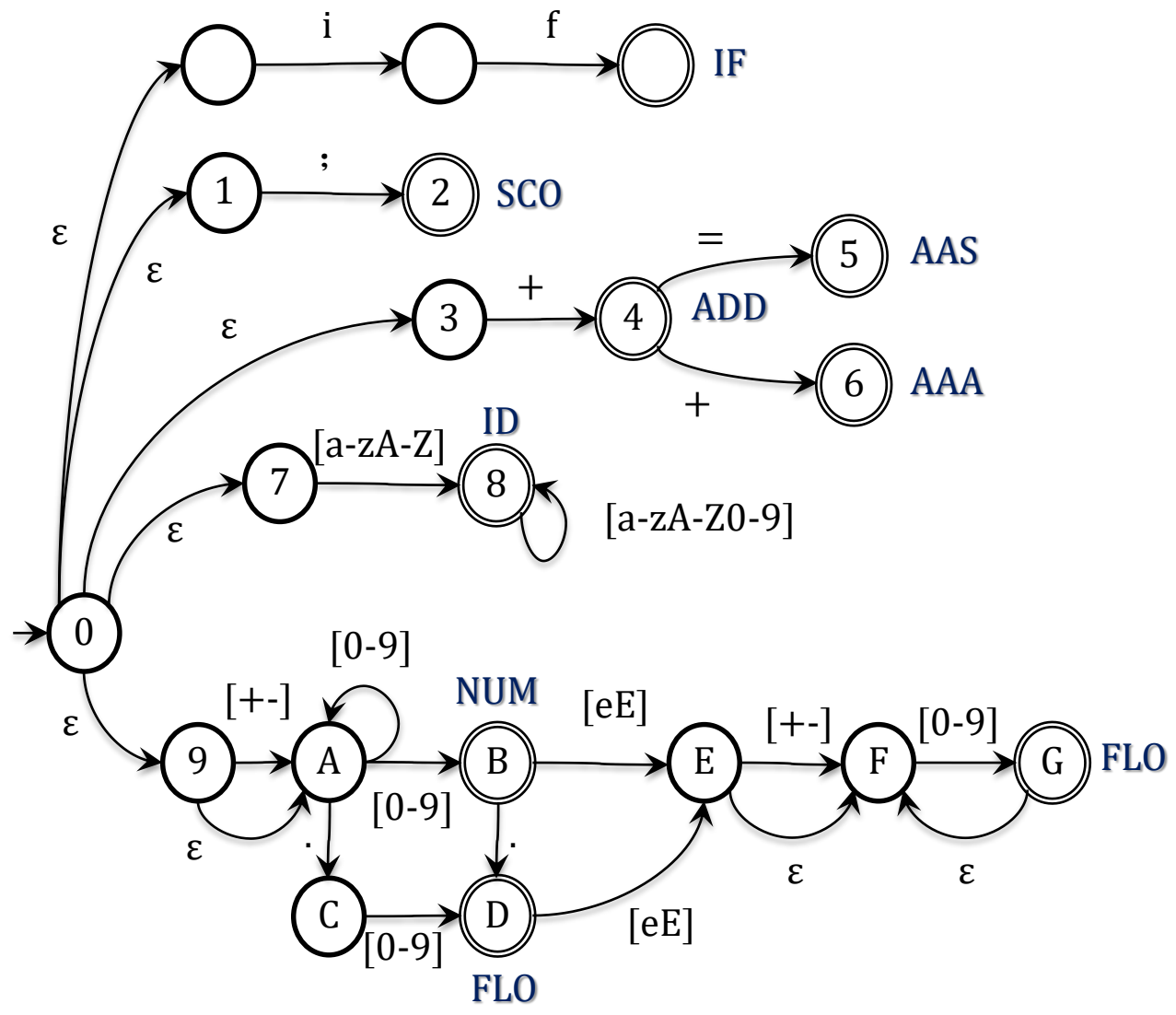
- $r_1 = [+ -]?[0-9]^+$
- $r_2 = [+ -]?[0-9]^* \cdot [0-9][0-9]^* \mid [+ -]?[0-9][0-9]^* \cdot [0-9]^*?$



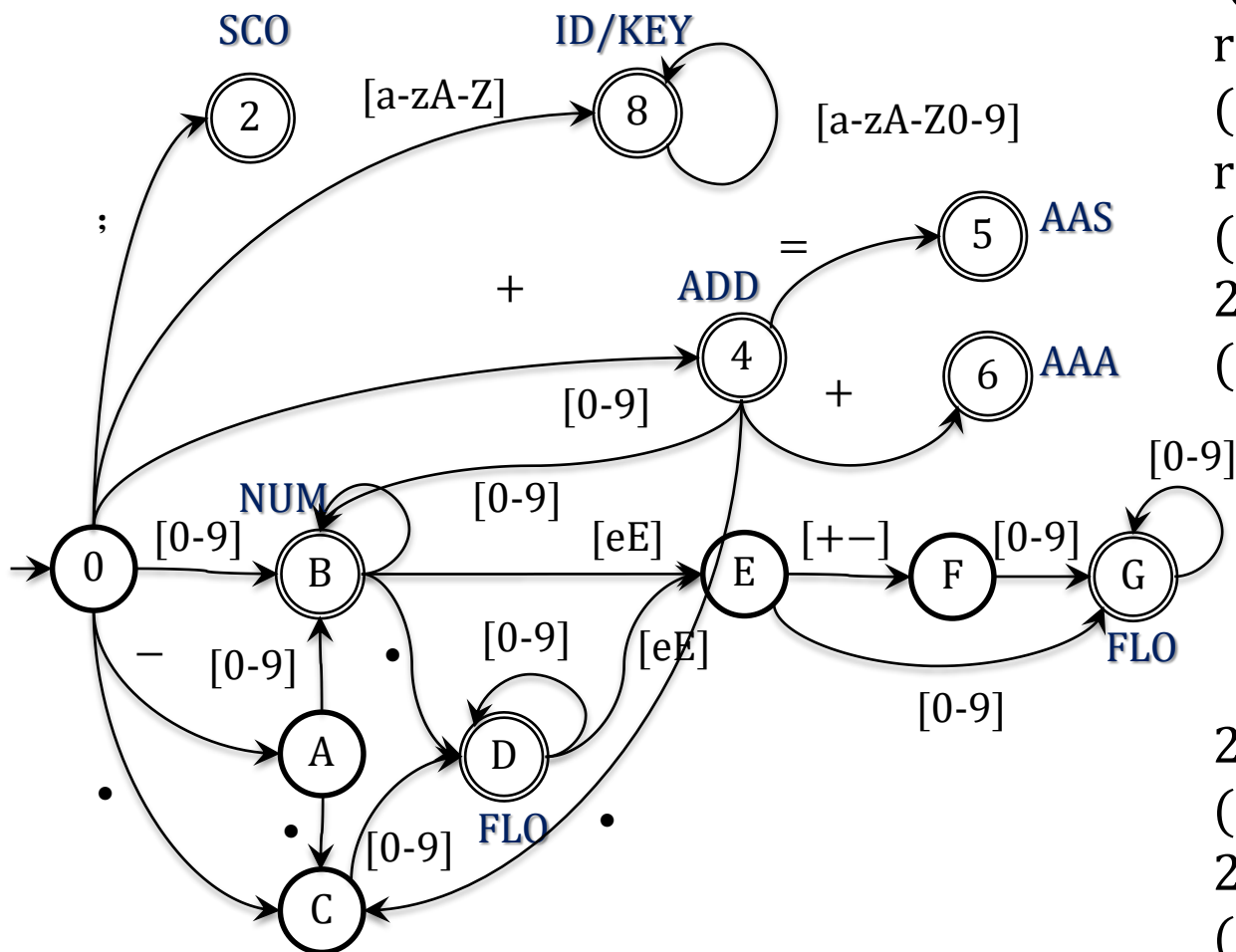
标记	状态	[0-9]	[+ -]	.
	$\rightarrow 012a$	c24	b2	3
NUM	*c24	c24	\varnothing	35
	b2	c24	\varnothing	3
	3	5	\varnothing	\varnothing
FLO	*35	5	\varnothing	\varnothing
FLO	*5	5	\varnothing	\varnothing



例2



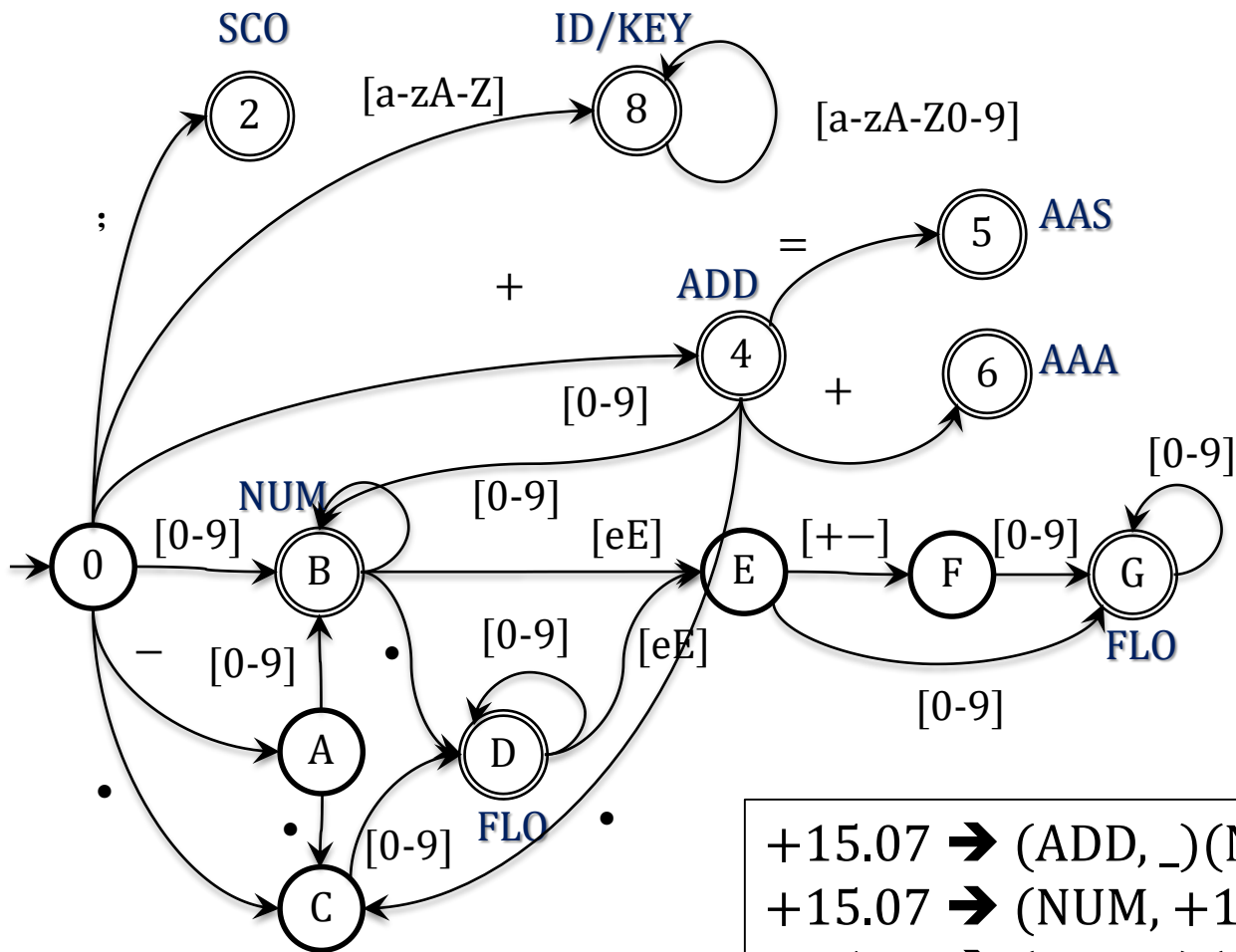
σ _DFA的识别举例



return21# →
 (ID, 'r)(ID, 'eturn21) ×
 return21# →
 (return, _)(NUM 21) ×
 return21# →
 (ID, 'return21) ✓
 21return# →
 (NUM, 21)(RETURN, _) ✓

2+3# →
 (NUM, 2)(ADD, _)(NUM, 3) ? ✓
 2+3# →
 (NUM, 2)(NUM, +3) ? ×

σ _DFA的识别举例



- +15.07 → (ADD, _)(NUM, 15)(FLO, .07) ✗
- +15.07 → (NUM, +15)(FLO, .07) ✗
- +15.07 → (ADD, _)(FLO, 15.07) ?√
- +15.07 → (FLO, +15.07) ?√
- +N15.07 → (ADD, _)(ID, 'N')(FLO, 15.07) ✗
- +N15.07 → (ADD, _)(ID, 'N15')(FLO, .07) √



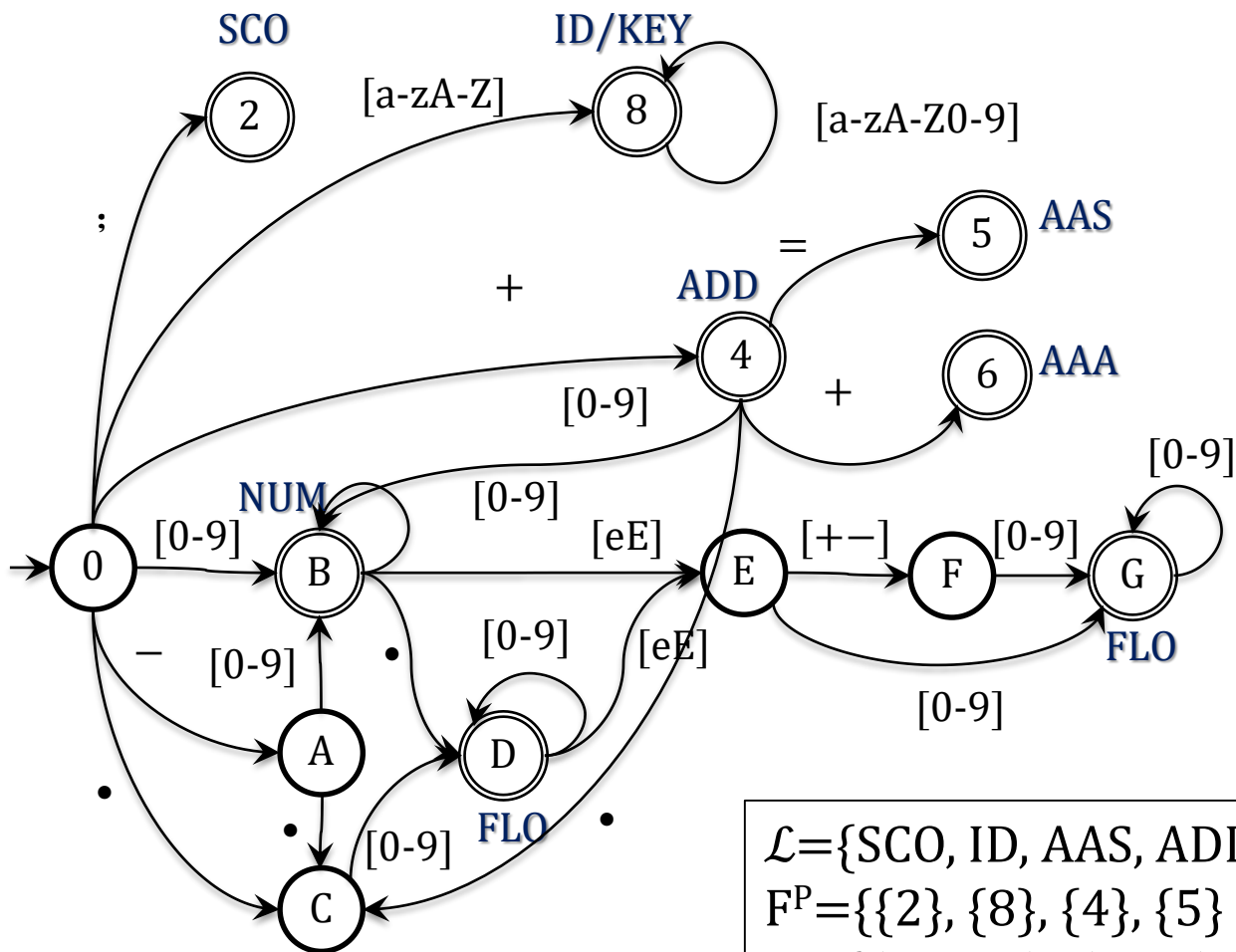
- 构建联合DFA，它按照规则拆分输入串为各子串并识别之。
- 给词法分析器指定要识别的记号类别，即所有的语言；
- 词法分析器合理切分源程序符号串，切分出的子串属于这些语言中的一个。
- 如果有多个切分方案，按规则决定应该是哪一个方案。
- 原理：用 n 个语言 r_1, r_2, \dots, r_n 同时判定一个串 x ，等价于用一个语言 $r_1|r_2|\dots|r_n$ 来判定。
- 若 x 被接受，那么 x 应该属于哪个语言呢？（长度最长的）
- 构建联合DFA步骤：已知 r_i 的NFA；构建集成NFA；转换为DFA；对接受状态做语言标记。



σ -DFA概念

- $\sigma_DFA(A, \mathcal{L})$ 是正则语言簇 \mathcal{L} 上的多语言联合DFA, 其中DFA $A=(Q, \Sigma, \nu, q_0, F)$, \mathcal{L} 元素两两不相交。
- 满足 $\forall q \in F \cdot \exists L \in \mathcal{L} \cdot L(q) \subseteq L$ 与 $\forall L \in \mathcal{L} \cdot \exists q \in F \cdot L(q) \subseteq L$
- 定理4.1 $\sigma_DFA(A, \mathcal{L})$ 是正则语言簇 \mathcal{L} 上的多语言联合DFA当且仅当 F 有一个与 \mathcal{L} 一一对应的划分集 F^P , F 为 A 的接受状态集。
- 也写为 F 的划分集 F^P 满足 $|F^P|=|\mathcal{L}|$ 且 $\forall P \in F^P \cdot (\cup_{q \in P} \cdot L(q)) \in \mathcal{L}$ 。
- 也就是有函数 $\psi: F \rightarrow \mathcal{L}$
- $\psi(q)=L$, 当且仅当 $\exists P \in F^P \cdot (q \in P \wedge L(q) \in L \wedge L \in \mathcal{L})$

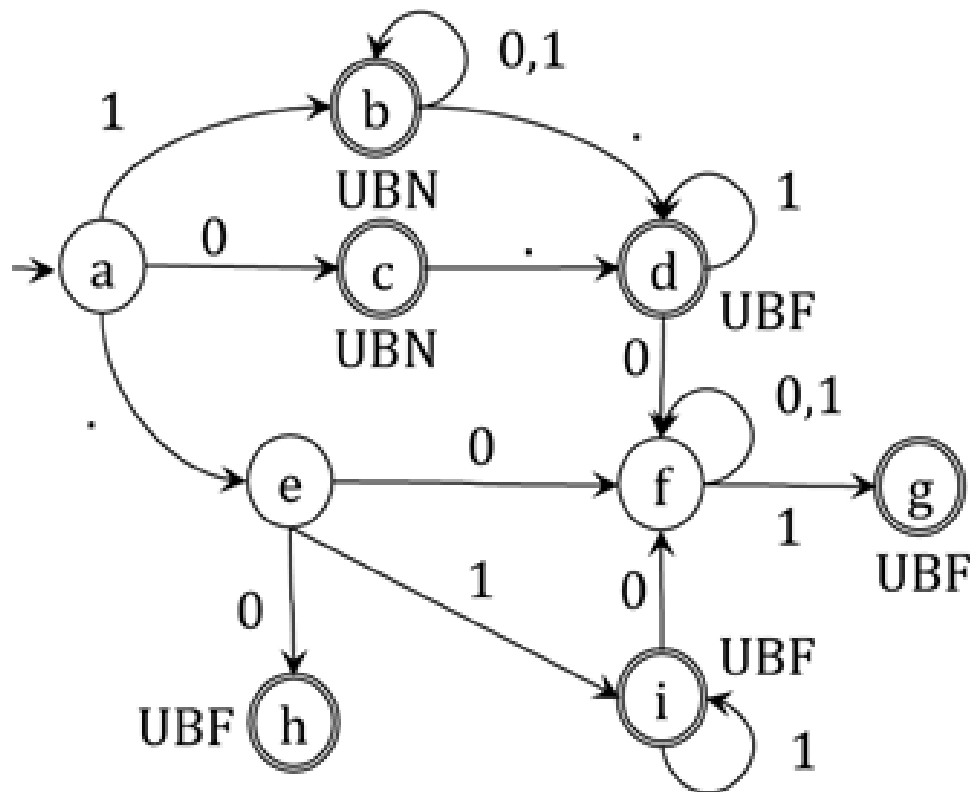
σ _DFA的识别举例



$\mathcal{L} = \{SCO, ID, AAS, ADD, AAA, NUM, FLO\}$
 $F^P = \{\{2\}, \{8\}, \{4\}, \{5\}, \{6\}, \{B\}, \{D, G\}\}$
 $\psi = \{(2, SCO), (8, ID), (4, ADD), (5, AAS), (6, AAA), (B, NUM), (D, FLO), (G, FLO)\}$

σ _DFA(A, \mathcal{L}) 举例

- $\mathcal{L} = \{UBN, UBF\}$
- $F^P = \{\{b, c\}, \{d, g, h, i\}\}$
- $\psi = \{(b, UBN), (d, UBF), (c, UBN), (g, UBF), (h, UBF), (i, UBF)\}$





构建 σ _DFA(A, \mathcal{L})

- 构建语言簇 \mathcal{L} 上的联合DFA。用到一个松弛的有序语言簇 \mathbb{L} ，二者元素个数相同，满足， $\mathcal{L} = \{L_1, \dots, L_n\}$, $L_i = \mathbb{L}[i] \setminus \mathbb{L}[i-1] \setminus \dots \setminus \mathbb{L}[1]$ ，可见 L_i 可以被放大至 $\mathbb{L}[i]$ ，从而获得构建上的便利。
- 若 $w \in \mathbb{L}[i] \setminus \dots \setminus \mathbb{L}[1]$ 且 $w \notin \mathbb{L}[k]$, $k < i$ ，那么 w 的词法类别为 $L = \mathbb{L}[i] \setminus \dots \setminus \mathbb{L}[1]$, $L \in \mathcal{L}$ ，但对任意 $j > i$ ， w 也可能属于 $\mathbb{L}[j] \setminus \dots \setminus \mathbb{L}[1]$ 但是不能改变它的词法类别 L 。这里使用了事实优先级原则，即有序集合 \mathbb{L} 下标较小者的优先级较高。



输入： n 个正则语言的有序集合 $\mathbb{L}=(L_1 L_2 \dots L_n)$ ，
满足 $\{\mathbb{L}[i]\setminus\mathbb{L}[i-1]\setminus\dots\setminus\mathbb{L}[1] \mid i=1,\dots,n\}=\mathcal{L}$

输出： σ -DFA $((Q, \Sigma, v_D, q_0, F), \mathcal{L})$ 且有满射映射
 $\{(q, L) \mid q \in F, L=\mathbb{L}[\text{mark}(q)]\setminus\dots\setminus\mathbb{L}[1]\}$

(1) 为每个 $\mathbb{L}[i]$, $1 \leq i \leq n$, 构造有穷自动机FA $A_i=(Q_i, \Sigma_i, v_i, q_i, F_i)$, 且
 $Q_i \cap Q_j = \varnothing, i \neq j$;

(2) 对于 $i=1, \dots, n$, 令 $\forall q \in F_i \cdot \text{mark}[q]=i$ 且
 $\forall q \in Q_i \setminus F_i \cdot \text{mark}[q]=\infty$;

(3) 构建集成NFA $N=(Q_N, \Sigma, v_N, q_0, F_N)$, 其中
 $Q_N=\{q_0\} \cup \cup_i Q_i$; $q_0 \notin \cup_i Q_i$; $\Sigma=\cup_i \Sigma_i$;
 $v_N=\{(q_0, q_i, \varepsilon) \mid 1 \leq i \leq n\} \cup \cup_i v_i$; $F_N=\cup_i F_i$; $\text{mark}[q_0]=\infty$;

(4) 采用子集法将NFA N 转换为DFA $A=(Q, \Sigma, v_D, q_0, F)$;
令 $\forall q \in F \cdot \text{mark}[q]=\min_{q \in q} \text{mark}(q)$;

(5) 返回 σ -DFA (A, \mathcal{L}) 且有满射映射
 $\{(q, L) \mid q \in F, L=\mathbb{L}[\text{mark}(q)]\setminus\dots\setminus\mathbb{L}[1]\}$ 。



词法分析器设计要点： σ -DFA

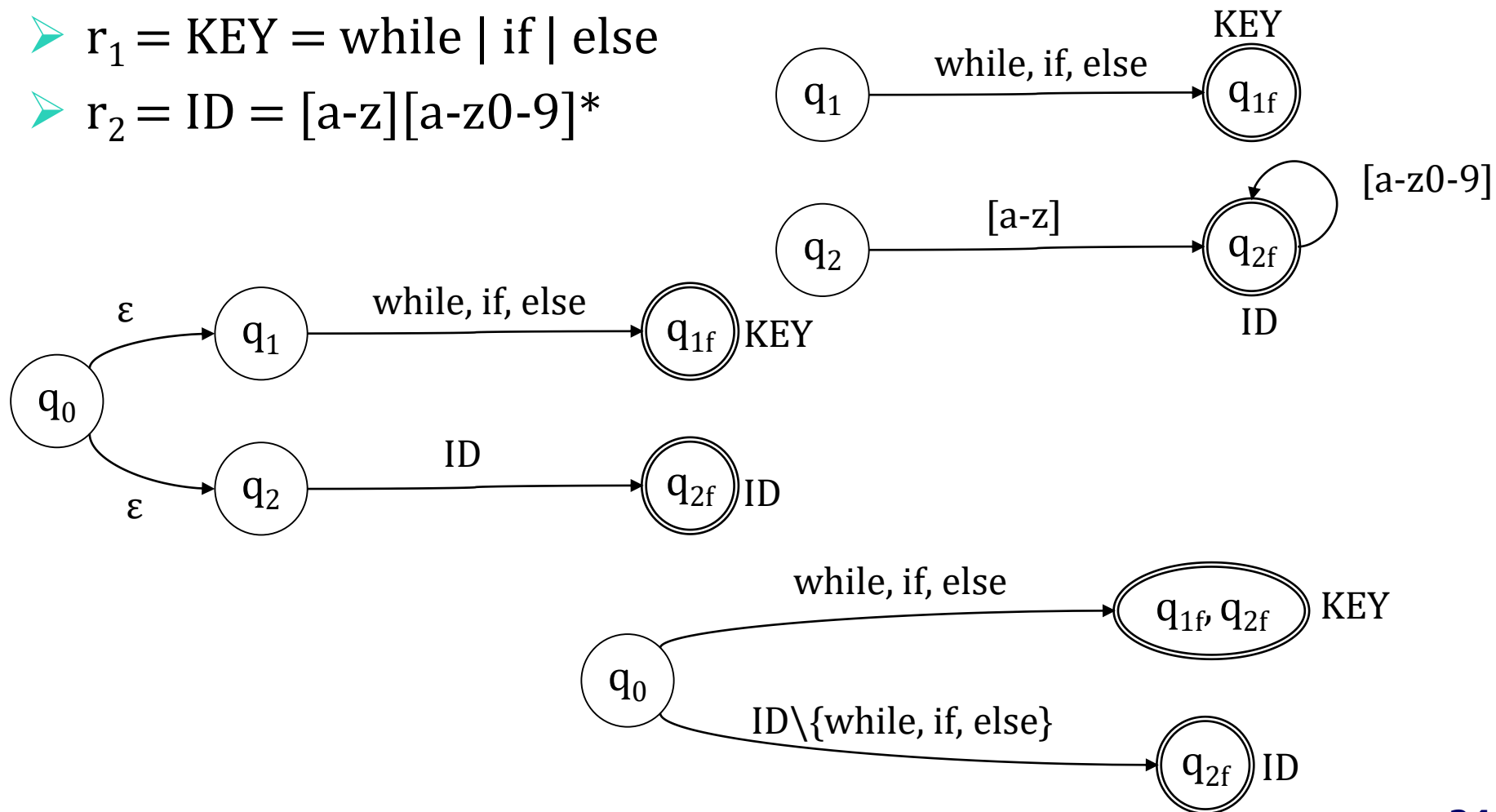
- 确定好各个词法单元种属，构建 σ -DFA
- 借用字符流控制功能编码实现该 σ -DFA
- σ -DFA基于前缀最大化匹配原则进行识别

例：标识符与关键字

➤ 构造算法的输入中下标较小者的优先级较高

➤ $r_1 = \text{KEY} = \text{while} \mid \text{if} \mid \text{else}$

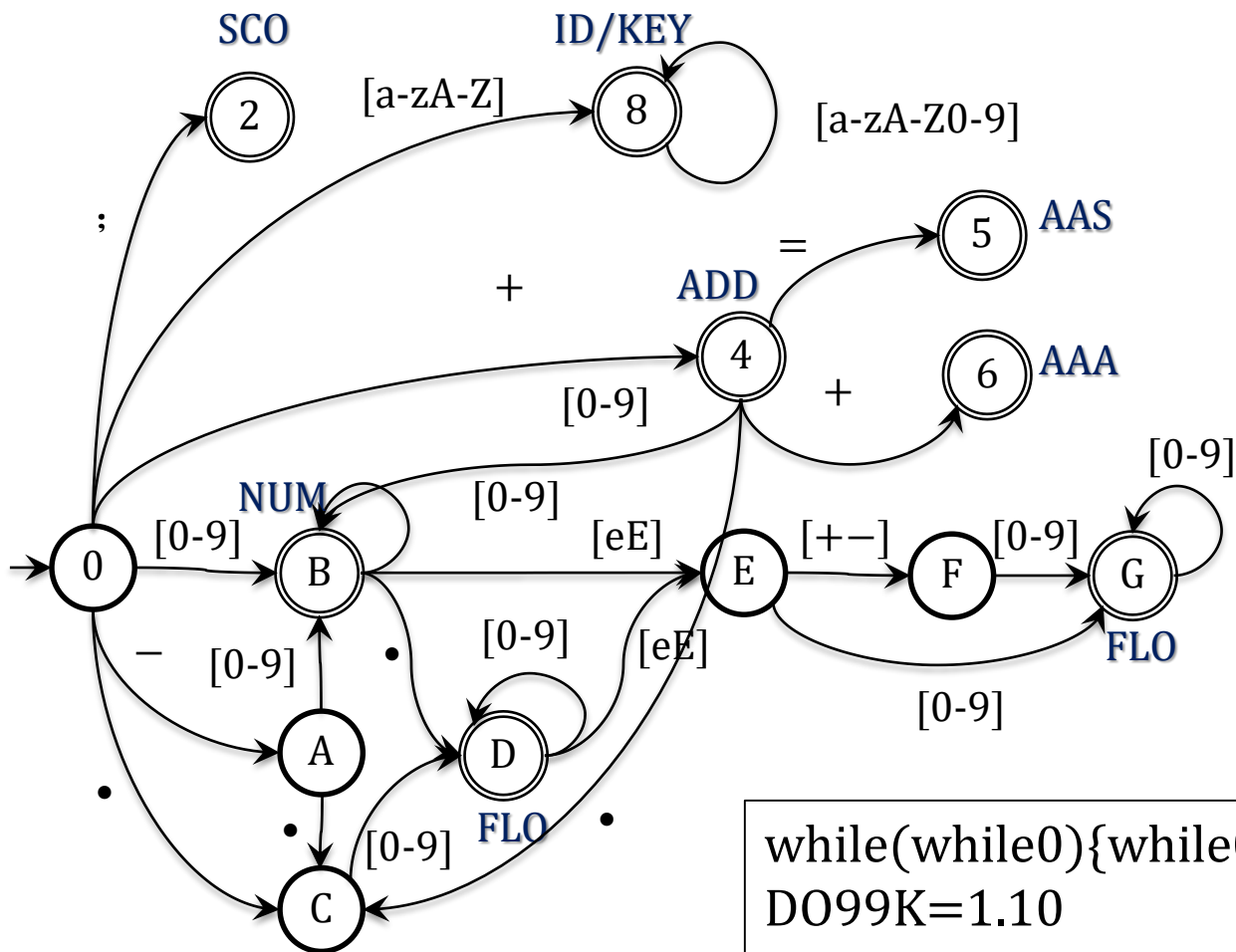
➤ $r_2 = \text{ID} = [\text{a-z}][\text{a-z0-9}]^*$





- UBN与UBF
- 整数与定点数
- 无符号整数与有符号整数
- 标识符与常数（各种进制）

σ _DFA的识别举例



```
while(while0){while0--;scan(while0)}
D099K=1.10
```



- 前缀最大化匹配原则能让 σ -DFA尽可能读入符号。
- 当 σ -DFA继续运行拒绝的时候，那么就需要回退到最近一次在接受状态的地方并返回结果。从那以后多读入的字符都要退回给输入流。
- 接下来如果输入流还有，则从初始状态开始，对剩余串再次运行 σ -DFA识别下一词法单元。
- 在存储受限情况下可采用乒乓缓冲：
 - 每个缓冲区大小不小于最大超前搜索长度。
 - 保持一个缓冲区既是另一个的前驱又是后继。



- 如果输入串的前缀都不是合法记号，则遇到一个词法错误
- 报错，真错误
- 跳过一些符号直到找到合法前缀，找到更多错误，不一定是真的



➤ C语言词法分析器

- 词法分析器作为子程序
- 事先构建好识别词法单元的联合自动机
- 编码实现为“当前状态+当前输入符号->下一状态”，若下一状态不存在则返回结果或错误，若存在则作为已读入串并做相应处理。
- 利用文件系统的字符流控制功能。

➤ 全局变量作为接口：

- `int ch;` 当前输入符号
- `int intval, floatval;` 返回时为记号的值（若有）
- `char textval[255]; int index;` 已读串及尾字符索引值
- `int lastloc, lastkind;` 最大化原则的实现



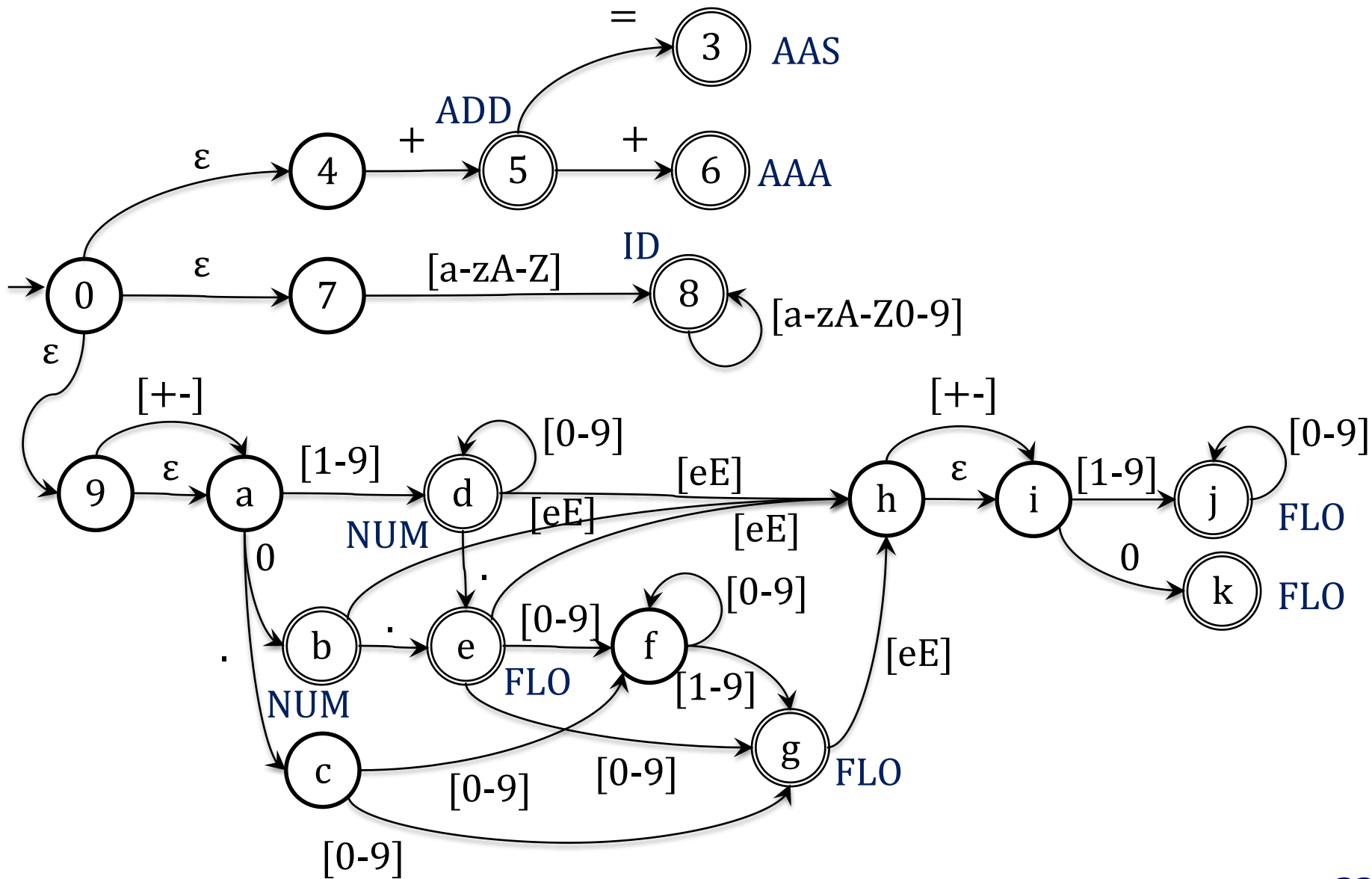
➤ 其他

- 关键字、分界符、运算符均采用一符一种
- 错误处理：无；
- 符号表、关键字表：都有
- 作为子程序方式，调用一次返回输入流中第一个token。

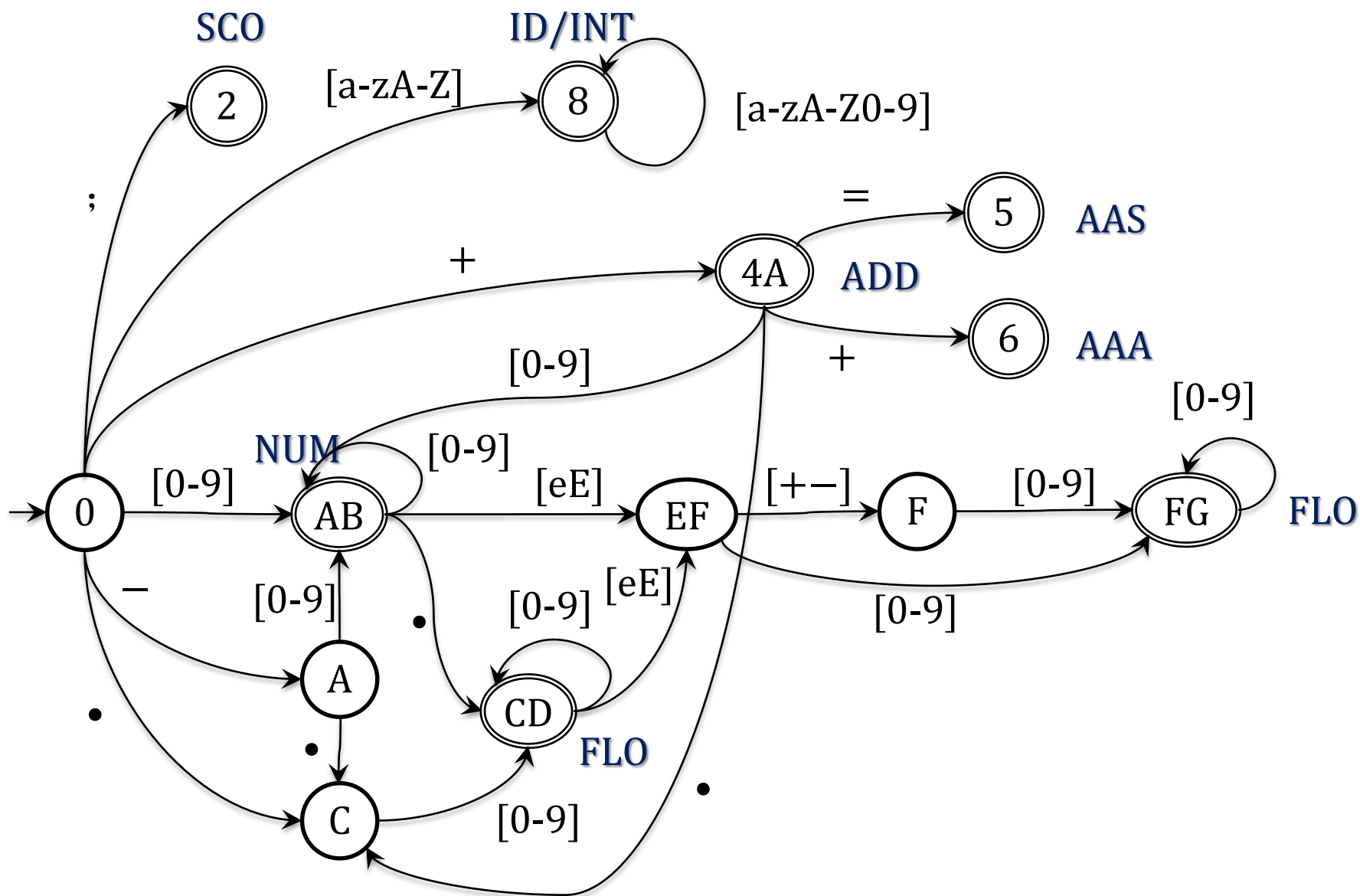


例：联合DFA构建

- 可索引语言簇 $L = \text{list}(\text{ADD AAA NUM FLO IF ID})$:
- $\langle \text{usn} \rangle \rightarrow [1-9][0-9]^*|0$
- $\langle \text{num} \rangle \rightarrow [+ -]^?([1-9][0-9]^*|0)$
- $\langle \text{fpn} \rangle \rightarrow [+ -]^? \langle \text{usn} \rangle ?. ([0-9]^*[1-9]|0) | [+ -]^? \langle \text{usn} \rangle .$
- $\langle \text{flo} \rangle \rightarrow \langle \text{fpn} \rangle | \langle \text{fpn} \rangle [e|E] \langle \text{num} \rangle | \langle \text{num} \rangle [e|E] \langle \text{num} \rangle$
- $\langle \text{str} \rangle \rightarrow [A-Za-z0-9]^+$
- $\text{ADD} = L(+)$
- $\text{AAA} = L(++)$
- $\text{NUM} = L(\langle \text{num} \rangle)$
- $\text{FLO} = L(\langle \text{flo} \rangle)$
- $\text{IF} = L(\text{if})$
- $\text{ID} = L(\langle \text{str} \rangle) \setminus L(\langle \text{num} \rangle) \setminus L(\langle \text{flo} \rangle) \setminus L(\langle \text{if} \rangle)$

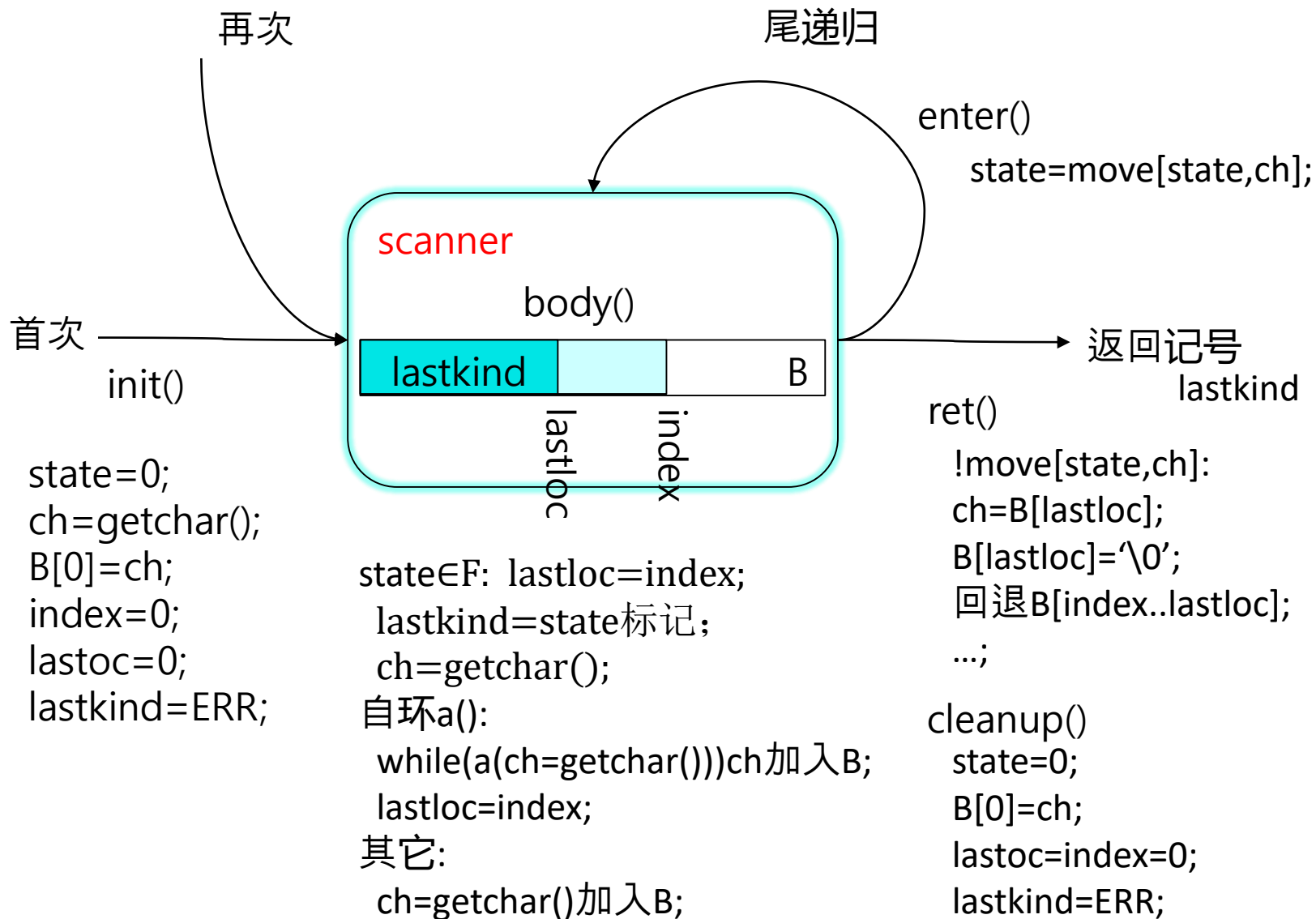


σ _DFA





词法分析器概貌



move[]

mark	state	;	+	=	[a-zA-Z]	[0-9]	-	.	[eE]
	->0	2	4		8	1	3	7	
NUM	*1					1		9	10
SCO	*2								
	3					1		7	
ADD	*4		6	5		1		7	
AAS	*5								
AAA	*6								
	7					9			
ID/INT	*8				8	8			
FLO	*9					9			10
	10		11			12	11		
	11					12			
FLO	*12					12			

0~01379A, 1~AB, 2~2, 3~A, 4~4A, 5~5,
6~6, 7~C, 8~8, 9~CD, 10~EF, 11~F, 12~FG

0	1	2	3	4	5	6	7
;	+	=	[a-zA-Z]	[0-9]	-	.	[eE]



σ _DFA

L	Q	i	f	\+	0	[1-9]	a-zA-Z	[eE]	.	\-
	>01479a	28	8	5a	b	d	8	8	c	a
ID	*28	8	38		8	8	8	8		
ID	*8	8	8		8	8	8	8		
ADD	*5a			6	b	d			c	
NUM	*b							hi	e	
NUM	*d				d	d		hi	e	
	c				fg	fg				
	a				b	d			c	
IF	*38	8	8		8	8	8	8		
AAA	*6									
	hi				k	j				i
FLO	*e				fg	fg		hi		
FLO	*fg				f	fg		hi		
	i				k	j				
	f				f	fg				
FLO	*k									
FLO	*j				j	j				



词法分析器的实现

- 词法单元: INT, ID, NUM, FLO, ADD, AAS, AAA, SCO
- RE的次序: 关键字int在ID之前; +在AAS和AAA之前。
- 超前搜索: 典型是+, 到+=或到++或到+整数或到+浮点数, 同时从+先到整数再到浮点数。用lastkind和lastloc记住回退位置 (前缀最大化原则的实现)
- 全局变量ch、index表示当前输入符号、缓冲内容的最末位置
- 返回结果的值在intval、floatval或textval中, 种属做为scanner()的返回值。
- scanner()调用前给出当前状态, 当前输入符号, 该符号也已放到缓冲textval中, 同时lastloc和lastkind保持记号最大化。
- 如果转移去的状态是接受状态则记住种属标记到lastkind中, 同时记下结尾处lastloc。
- 如果DFA死亡则返回lastkind, 若不是一符一种则textval中至lastloc为值, 可能还要做转换, 查表建立表项等。



词法分析器的实现

- 调用`scanner()`有两种时机：开始识别一个词法单元时调用；在识别的过程中（状态转移时）调用。
- 第一种调用发生在识别第一个词法单元时需要先调用`init()`；然后返回时（已经识别出该词法单元）需要调用`cleanup()`；
- 第二种调用发生在状态转移过程中，调用前已获得要转移去的状态和输入符号，并且`lastloc`和`lastkind`保持记号最大化。
- `move[state,row(ch)]`，即据此更新`state`。继续下面步骤。
- 如果`state`为接受状态，则置`lastloc`为`index`，置`lastkind`为该状态标记，读入下一符号，并返回`scanner()`调用。
- 如果`state`不是接受状态，则置`lastloc`为`index`，读入下一符号，并返回`scanner()`调用。
- 如果不能迁移则返回`token`，即`lastkind`作为`token`种属，并将已读串中至`lastloc`部分的进行相应转换，形成`token`值，当然但对于一符一种的`token`则值无定义。



➤ +123.45e-7#

state	ch	textval	index	lastloc	lastkind	ret/en
0	+	+	0	0	ADD	enter
4	#	+#	1	2	ADD	ret
4	1	+1	1	1	NUM	enter
1	#	+1#	2	2	NUM	ret
1	2	+12	2	2	NUM	enter
1	3	+123	3	3	NUM	enter
1	.	+123.	4	4	FLO	enter
1	#	+123#	4	4	NUM	ret
9	4	+123.4	5	5	FLO	enter
9	5	+123.45	6	6	FLO	enter

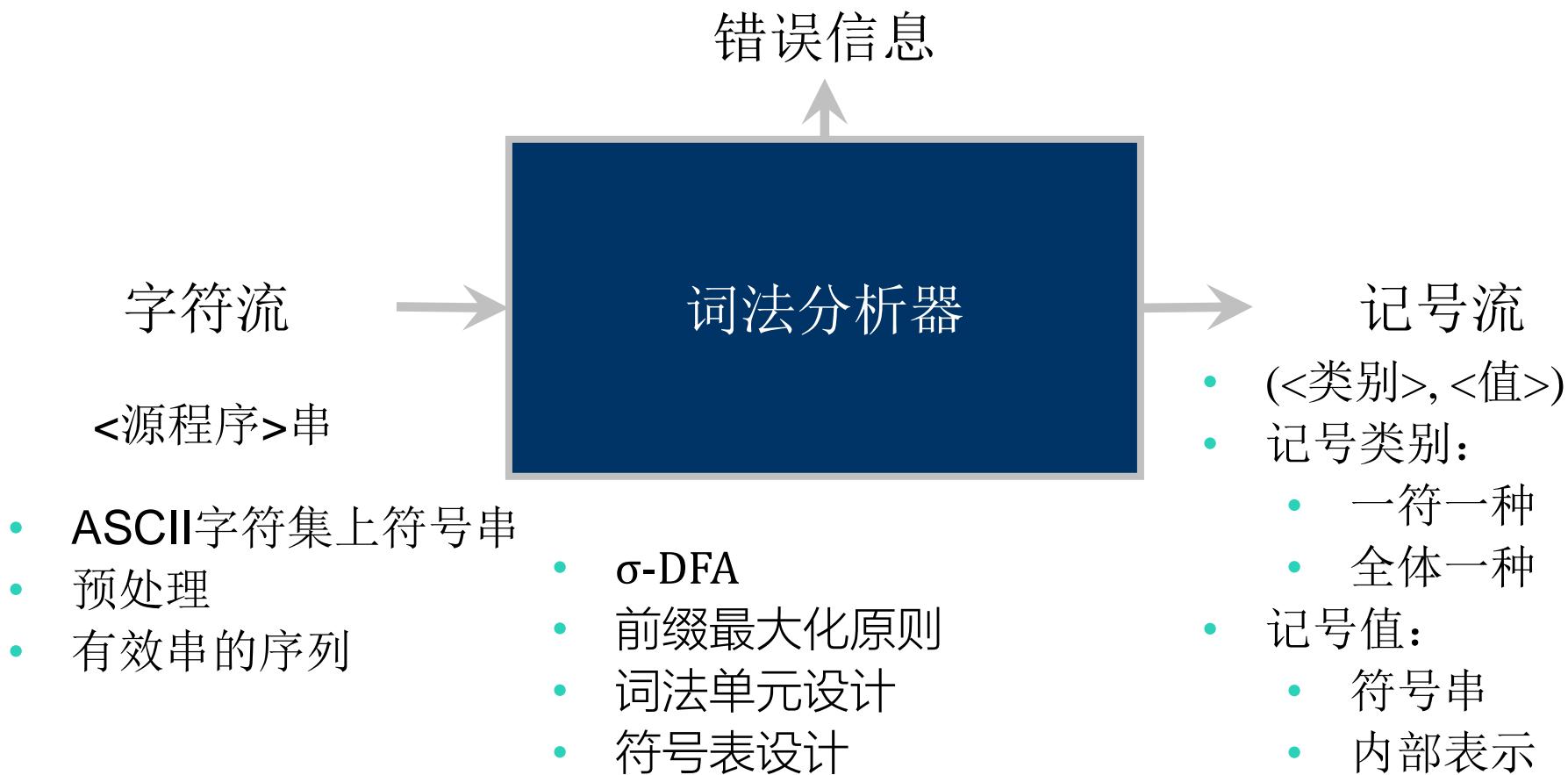
➤ +123.45e-7#

state	ch	textval	index	lastloc	lastkind	ret/call
9	5	+123.45	6	6	FLO	enter
9	e	+123.45e	7	7	FLO	enter
9	#	+123.45e#	8	7	FLO	ret
10	-	+123.45e-	8	7	FLO	enter
11	#	+123.45e-#	9	7	FLO	ret
11	7	+123.45e-7	9	9	FLO	enter
12	#	+123.45e-7#	10	9	FLO	ret

注：scanner()识别一个token即返回，输入流中以该token为前缀的部分已经去掉，随后再调它识别出下一token，依此下去。

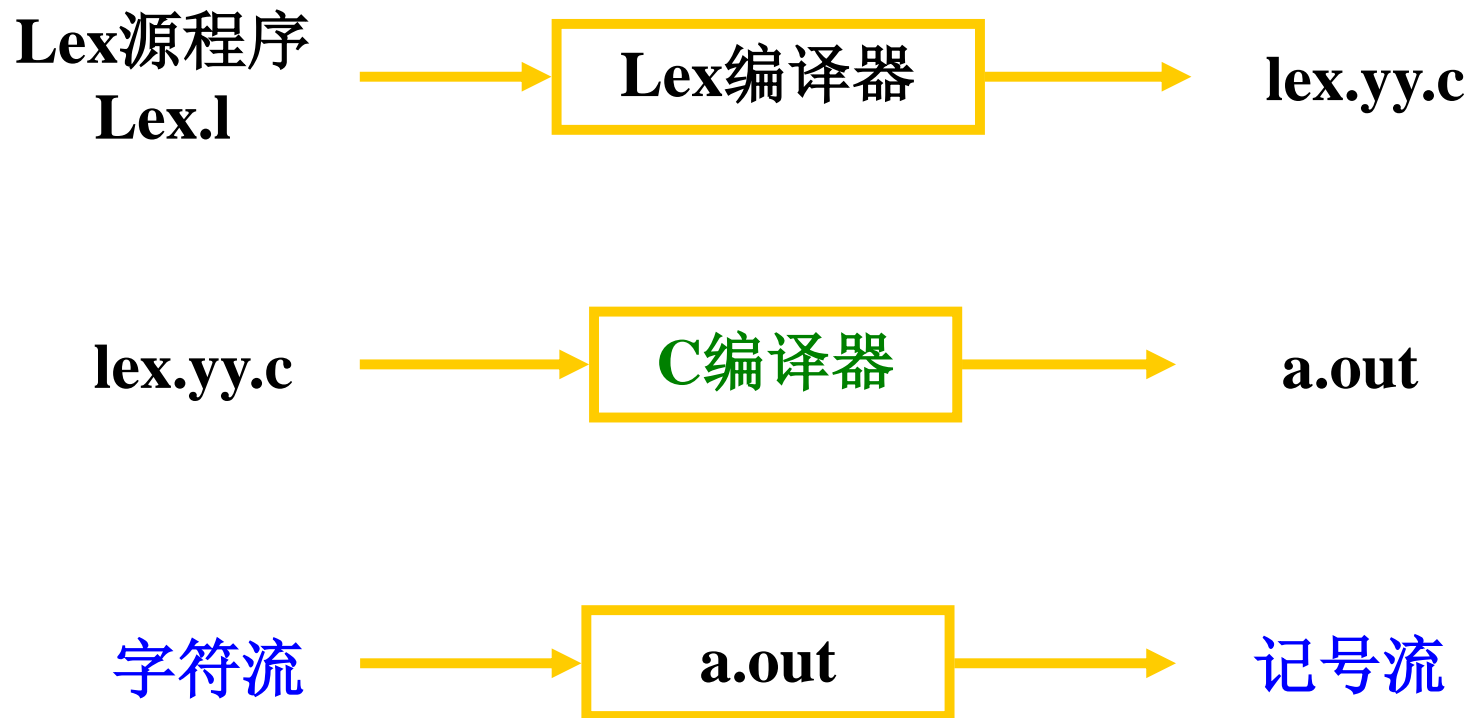


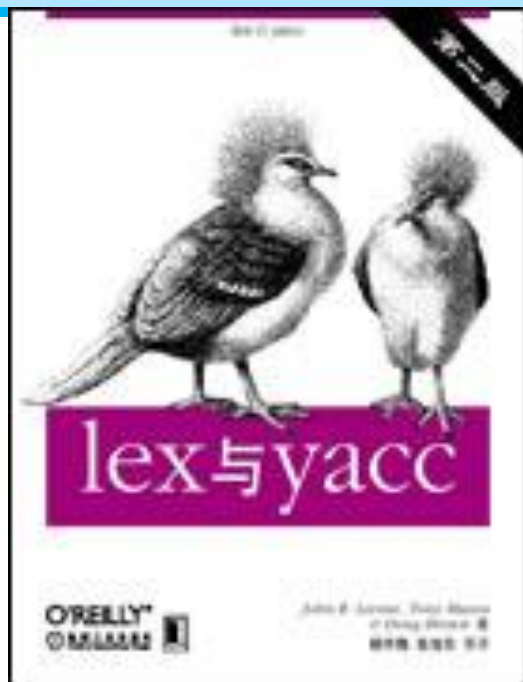
C2.2 词法分析器的设计与实现





C2.3 词法分析器的自动生成





《lex与yacc（第二版）》

作者: [John R. Levine](#), [Tony Mason](#), [Doug Brown](#) 著

杨作梅,张旭东,等 译

出版: 2003年1月

书号: 7-111-10721-7

页数: 392

定价: 45.00元



definitions

%%

rules

%%

user code

变量、常量和正规定义

R_1 $\{Actions\}$

...

R_n $\{Actions\}$

符号表代码、其它函数等



```

delim      [ \t\n]
ws         {delim}+
letter     [A-Za-z]
digit      [0-9]
id         {letter}({letter}|{digit})*
number     {digit}+(\.{digit}+)?(E[+\-]?{digit}+)?

%%
{ws} {}
if       {return(IF);}
then     {return(THEN);}
else     {return(ELSE);}
{id}     {yylval=install_id();return(ID);}
{number} {yylval=install_num();return(NUMBER);}
"<"     {yylval=LT;return(RELOP);}

...
%%
install_id() { 单词装入符号表并返回指针; }
...

```



- 知识点：多语言联合DFA、最大前缀匹配原理、一符一种记号、全体一种记号，编程语言常见记号类别
- 作业 p95 习题4.1-2
- 参考scanner(), 编写词法分析器 (专题实验)