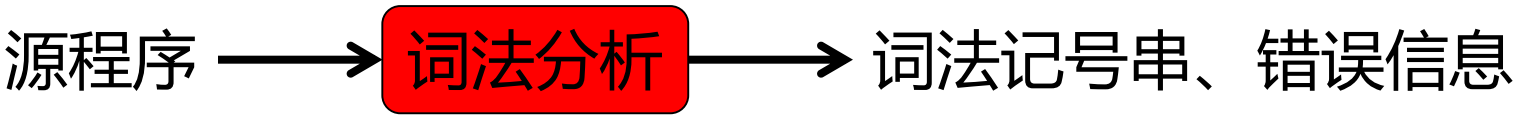
第四章 词法分析2025

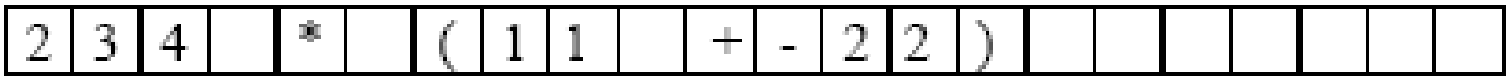
C2 词法分析



- 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ASICII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。



NUM(234) MUL LPA NUM(11) ADD NUM(-22) RPA

- 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，LPA，ADD，RPA

C语言的例子

```
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。

记号的二元组表示

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

```
while (i <= 100) {\n\ts += a[i]; /* form vector total */\n\ti++;\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，单元素语言可以用其成员指代它。

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

- 值是子串的表示，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。

记号的设计

```
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, [)

常见记号的类别

类别	*类别名字*	*语言定义*
Identifiers	ID	[a-zA-Z_][a-zA-Z_0-9]*
Integer constants	NUM	[+-]?[0-9]+
Character constants	CHR	'([a-zA-Z0-9]|\[a-zA-Z]) '
Floating constants	FLO	[+-]?[0-9]*. [0-9][0-9]*|[+-]?[0-9][0-9]*[. [0-9]*]?
String constants	STR	"([a-zA-Z0-9]|\[a-zA-Z])*"
Operator tokens	ADD, MUL, …	+, *, <,<=,==, ∧, ∨, ¬, …
Assignments	ASG, AAS, …	=, +=, ++, …
括号	LPA, LBK, …	(,), [,], {, }, ..
Keywords	IF, INT, …	while, int, if, else, return, …

对源程序预处理

- 该忽略的忽略，该处理的处理，因为它们不是记号。

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

回顾C程序

<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>→(r_1 | … | r_n)$^+$

预处理结果

<源程序>→(<有效串><分隔符>)$^+$<EOF>

<有效串>→ $(r_1 \mid \ldots \mid r_n)^+$

<分隔符>可以是任意感兴趣的且不在U1≤i≤$n \cdot \Sigma_i$中的符号。

扫描一个<有效串>返回一个记号串，直到遇到<EOF>，依次将各记号串连接作为词法分析结果。

针对有效串的拆分与识别

逐一扫描<有效串>#直到EOF结束。

<有效串>→$(r_1 \mid \ldots \mid r_n)^+$

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)?×

+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)√

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。

- 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。

多语言联合自动机识别原理

- 假定有n个语言依次定义为$r_1, ..., r_n$
- <有效串>$\in L((r_1|...|r_n)^+)$

- 为每个r_i构建FA A_i，初始状态为q_i。每个接受状态都带有标记L_i。

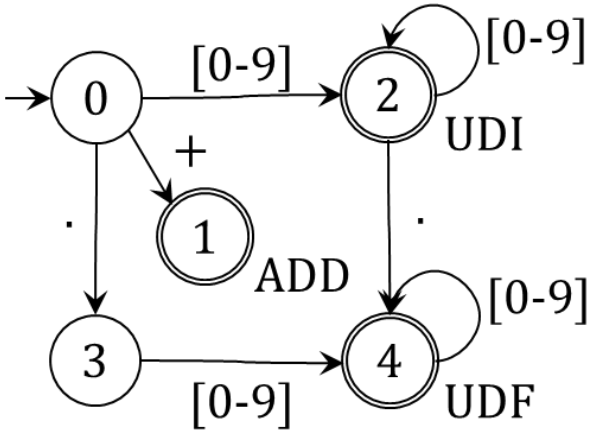
- 构建多语言联合NFA，从初始状态q_0都有ε弧射出到q_i。

- 确定化为一个多语言联合DFA（σ-DFA），它的每个接受状态的标记这样确定，如果该状态包含NFA的多个接受状态，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。

语言ADD、UDI和UDF联合DFA

消解冲突1：事实优先级

消解冲突2：前缀最大化原则



12+1.2#
(UDI, 12)(ADD, _)
(UDF, 1.2)

100#0.1#

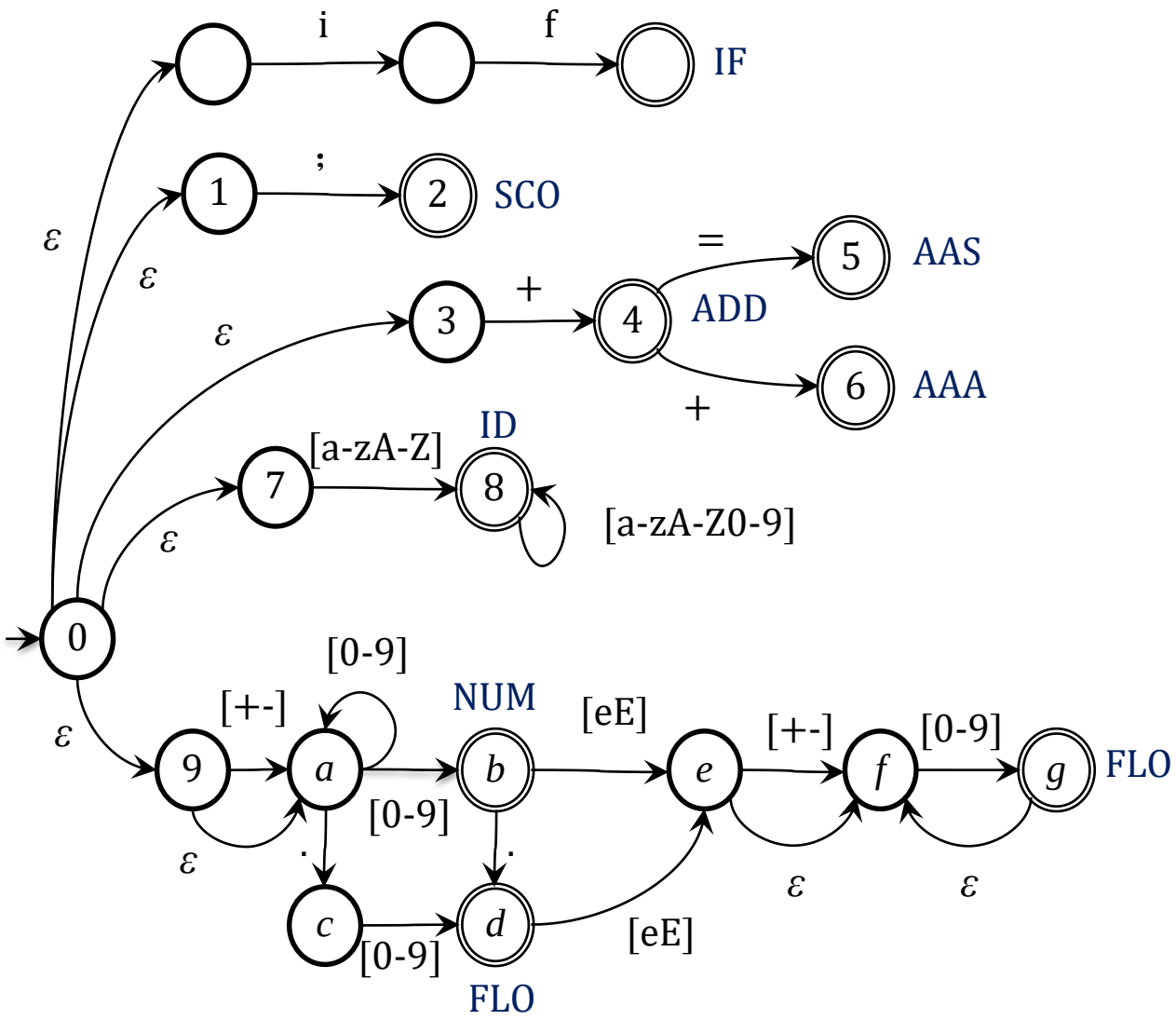
$\mathrm{ADD} = \{+\}$ $\mathrm{UDI} = L([0\text{-}9][0\text{-}9]^*)$
$\mathrm{UDF} = L([0\text{-}9][0\text{-}9]^*.[0\text{-}9]^* + .[0\text{-}9][0\text{-}9]^*)$

- ADD、UDI与UDF不相交
- UDF包含UDI　　状态2存在语言冲突

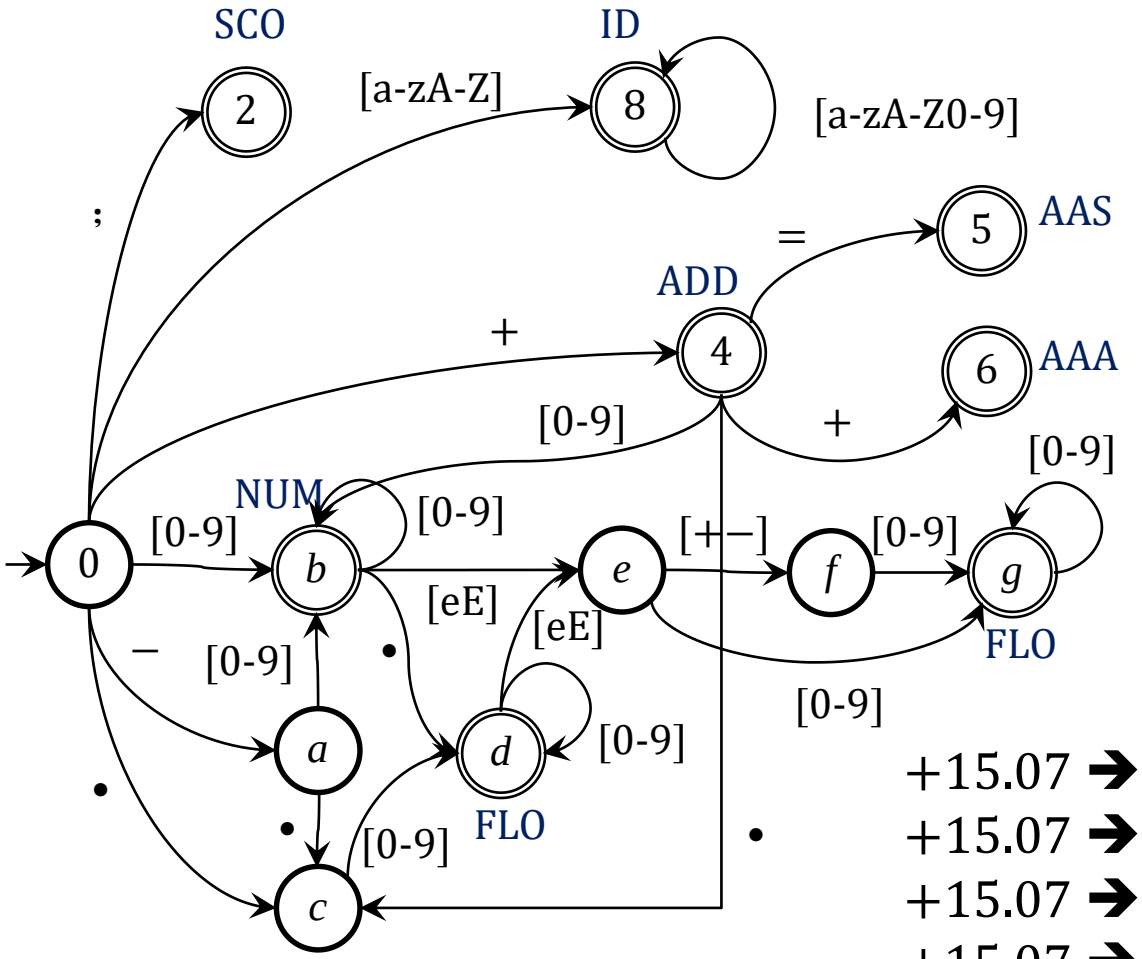
σ-DFA({ADD, UDI, UDF})的接受状态带有一个语言标记，表明该状态接受的语言是唯一的，这就将其语言划分为ADD、UDI和UDF了。

4.3 构建多语言联合DFA

例：语言簇{IF, SCO, ADD, AAS, AAA, ID, NUM, FLO}



多语言联合DFA



return21# ➔
(ID, r)(ID, eturn21)×
(ID, return)(NUM 21)×
(ID, return21)√
21return# ➔
(NUM, 21)(RETURN, _)×
(NUM, 21)(ID, return)√
2+3# ➔
(NUM, 2)(NUM, +3)√

+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)√

2+3# ➔
(NUM, 2)(ADD, _)(NUM, 3)?

多个自动机联合起来

- 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。（长度最长者）
- **原理：** 用n个语言$r_1,r_2,\ldots,r_n$同时判定一个串x，等价于用一个语言$r_1|r_2|\ldots|r_n$来判定。
- 若x被接受，那么x应该属于哪个语言呢？（按优先级决定）

- 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。

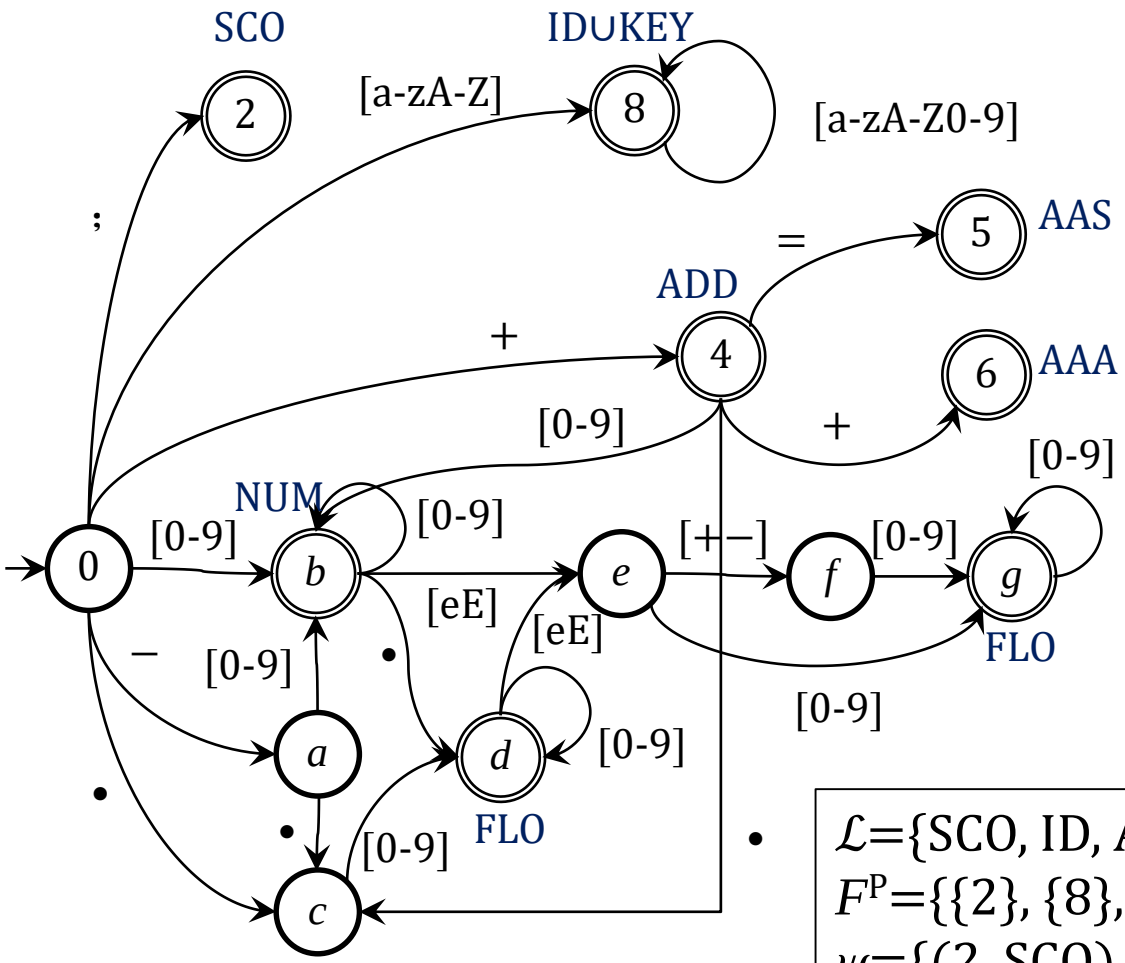
σ-DFA概念

- σ-DFA$(A, \mathcal{L})$是正则语言簇$\mathcal{L}$上的多语言联合DFA，其中DFA $A=(Q, \Sigma, \upsilon, q_0, F)$，$\mathcal{L}$元素两两不相交。
- 满足 $\forall q \in F \cdot \exists L \in \mathcal{L} \cdot L(q) \subseteq L$且$\forall L \in \mathcal{L} \cdot \exists q \in F \cdot L(q) \subseteq L$

- 定理4.1 σ-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。
- 其中，F的划分集F^{P}满足$|F^{\mathrm{P}}|=|\mathcal{L}|$且$\forall P \in F^{\mathrm{P}} \cdot (\cup q \in P \cdot L(q)) \in \mathcal{L}$。

- 也就是有函数$\psi: F \to \mathcal{L}$
- $\psi(q)=L$，当且仅当$\exists P \in F^{\mathrm{P}} \cdot (q \in P \wedge L(q) \subseteq L)$

σ-DFA的构成示例

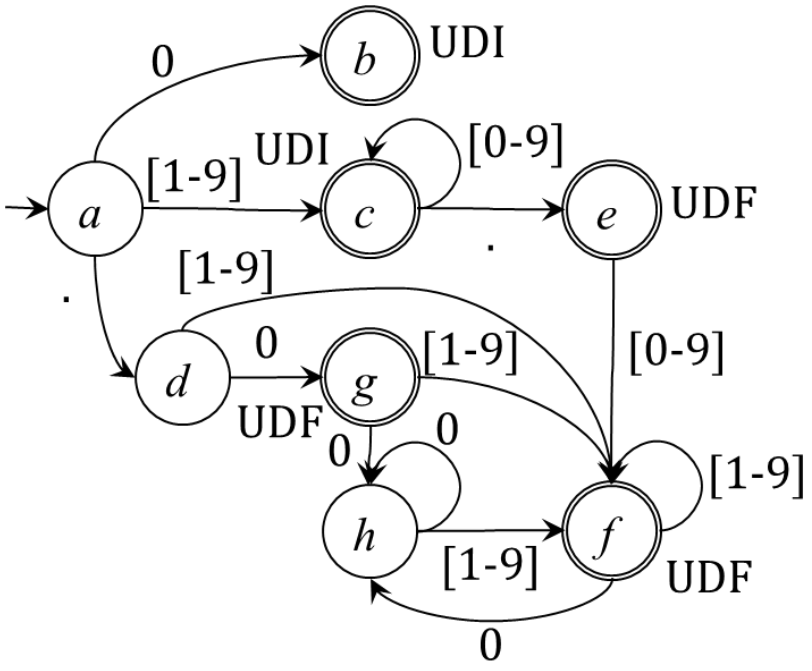


$\mathcal{L}$={SCO, ID, AAS, ADD, AAA, NUM, FLO}
F^P={{2}, {8}, {4}, {5} {6}, {B}, {D, G}}
ψ={(2, SCO), (8, ID), (4, ADD),(5, AAS), (6, AAA), (B, NUM), (D, FLO), (G, FLO)}

σ-DFA({UDI, UDF})

<udi>⟶0 | [1-9][0-9]*
<tail>⟶[1-9] | [0-9]*[1-9]
<udf>⟶<udi>\.<tail> | <udi>\. | <udi>\.0 | \.<tail> | 0\.<tail>

语言UDI = L(<udi>)和UDF = L(<udf>)都是正则语言，存在σ-DFA。



$F^{P} = \{\{b, c\}, \{e, f, g\}\}$

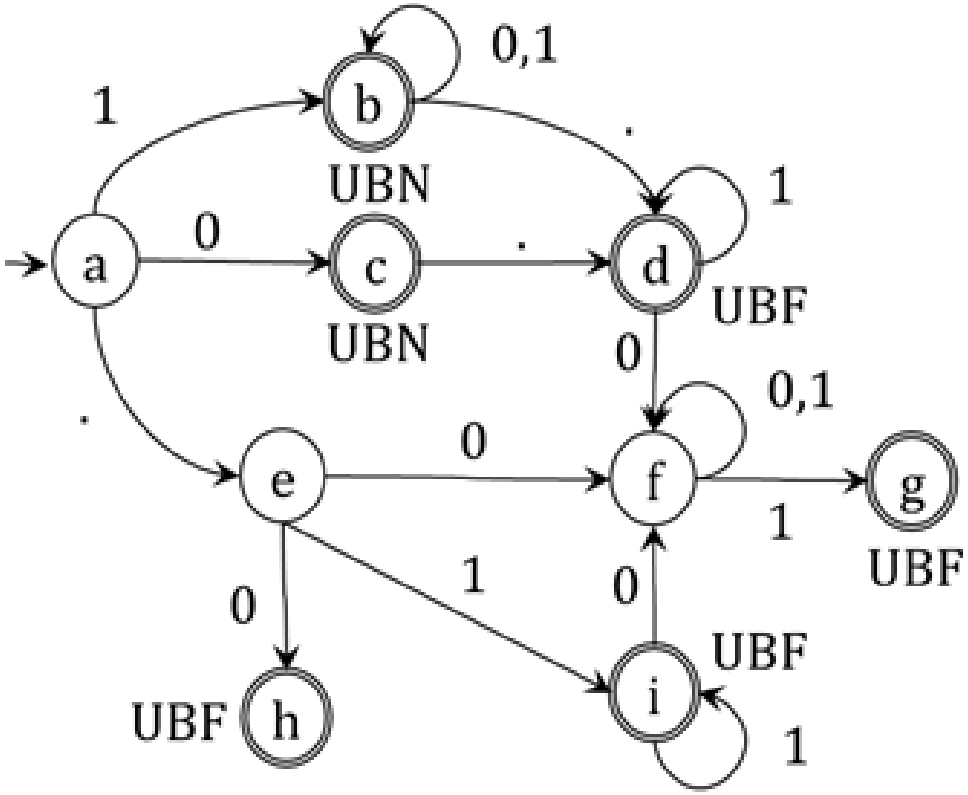
$\text{UDI} = \{w \in \Sigma \mid \tilde{v}(a, w) \in \{b, c\}\}$
$\text{UDF} = \{w \in \Sigma \mid \tilde{v}(a, w) \in \{e, f, g\}\}$

$\psi = \{(b, \text{UDI}), (e, \text{UDF}), (c, \text{UDI}), (f, \text{UDF}), (g, \text{UDF})\}$

$\text{UDI} \cap \text{UDF} = \emptyset$

σ-DFA$(A, \mathcal{L})$举例

- $\mathcal{L}$={UBN, UBF}
- F^{P}={{b, c}, {d, g, h, i}}
- ψ={(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，其中$\mathcal{L}$的元素两两不相交。
- 用到一个松弛的有序语言簇$\mathbb{L}$，二者元素个数相同，满足，$\mathcal{L} = \{L_1, \ldots, L_n\}, L_i = \mathbb{L}[i] \setminus \mathbb{L}[i\text{-}1] \setminus \ldots \setminus \mathbb{L}[1]$，可见$L_i$可以被放大至$\mathbb{L}[i]$，从而获得构建上的便利。

- 若$w \in \mathbb{L}[i] \setminus \ldots \setminus \mathbb{L}[1]$且$w \notin \mathbb{L}[k], k<i$，那么$w$的词法类别为$L = \mathbb{L}[i] \setminus \ldots \setminus \mathbb{L}[1]$，$L \in \mathcal{L}$，但对任意$j>i$，$w$也可能属于$\mathbb{L}[j] \setminus \ldots \setminus \mathbb{L}[1]$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。

算法4.1构造σ-DFA $(A, \mathcal{L})$

输入：n个正则语言的有序集合$\mathbb{L}$, $n=|\mathcal{L}|$,
满足$\{\mathbb{L}[i]\backslash\mathbb{L}[i\text{-}1]\backslash\ldots\backslash\mathbb{L}[1] \mid i=1,\ldots,n\}=\mathcal{L}$

输出：σ-DFA $((\mathbb{Q}, \Sigma, v_D, \mathbb{q}_0, \mathbb{F}), \mathcal{L})$且有满射映射
$\{(\mathbb{q}, L) \mid \mathbb{q}\in\mathbb{F}, L=\mathbb{L}[\text{mark}(\mathbb{q})]\backslash\mathbb{L}[\text{mark}(\mathbb{q})\text{-}1]\backslash\ldots\backslash\mathbb{L}[1]\}$

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

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

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

（4）采用子集法将NFA N转换为DFA $A=(\mathbb{Q}, \Sigma, v_D, \mathbb{q}_0, \mathbb{F})$;
令$\forall\mathbb{q}\in\mathbb{F} \cdot \text{mark}[\mathbb{q}]=\min q\in\mathbb{q} \cdot \text{mark}(q)$;

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

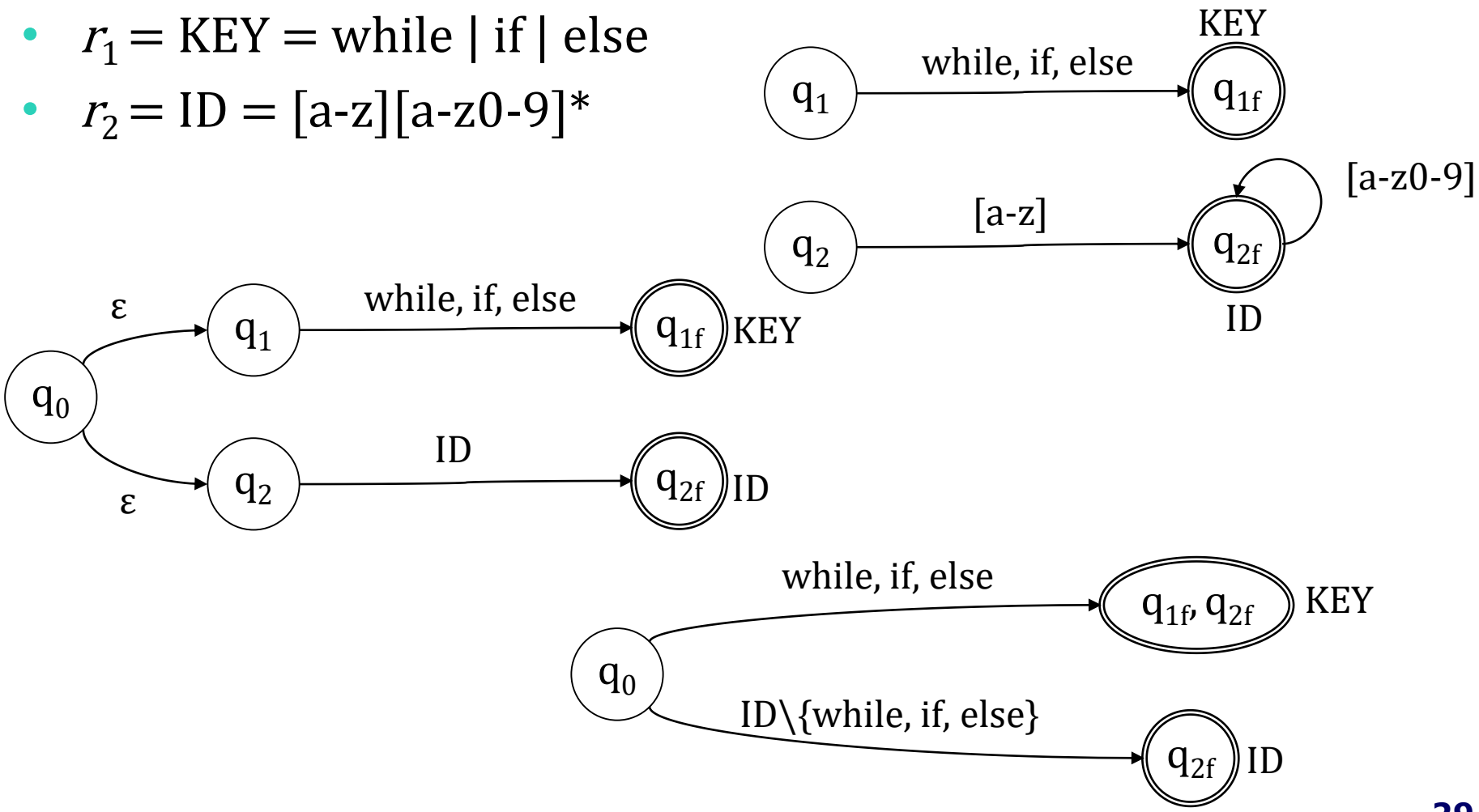
词法分析器设计要点：σ-DFA

- 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

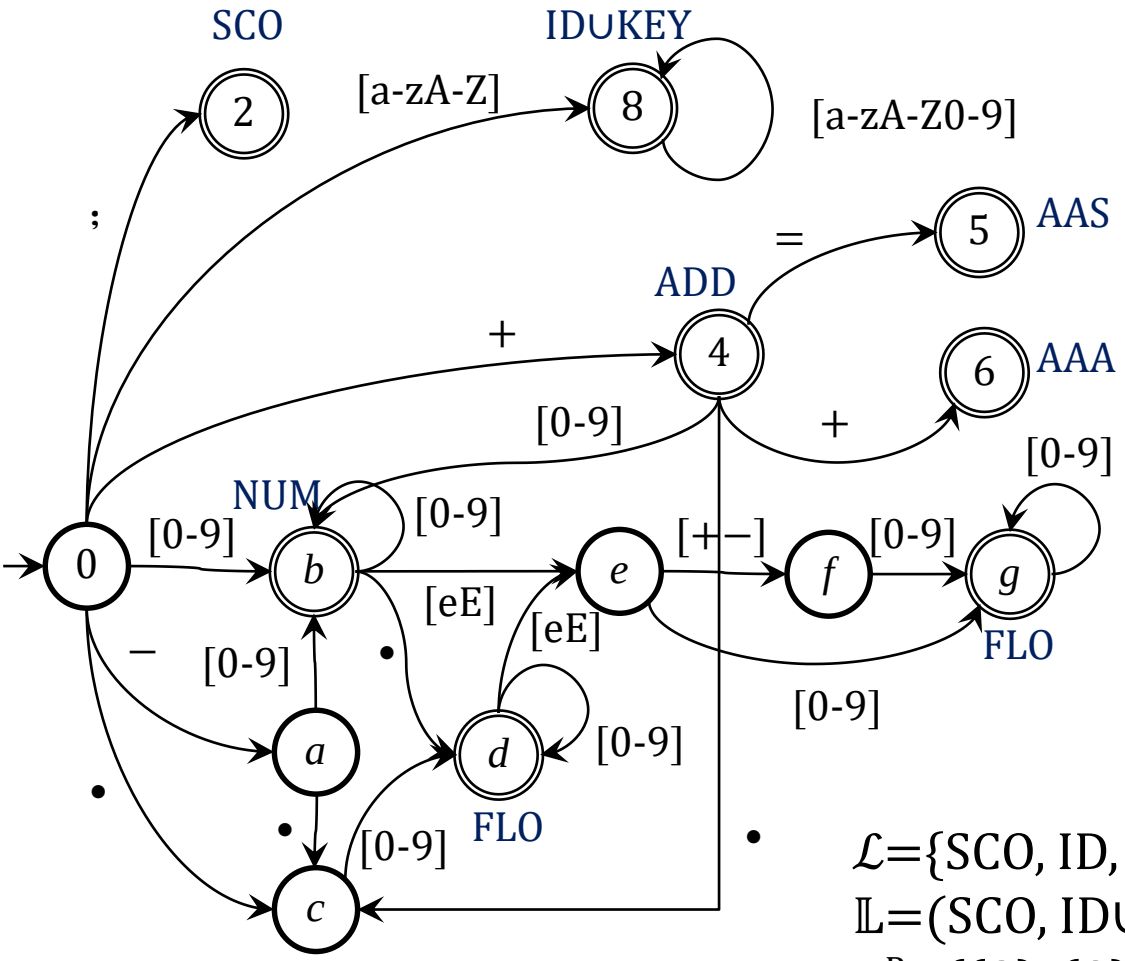
- 构造算法的输入中下标较小者的优先级较高

- r_1 = KEY = while | if | else
- r_2 = ID = [a-z][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)
}
DO99K=1.10
```

$\mathcal{L}$={SCO, ID, KEY, AAS, ADD, AAA, NUM, FLO}

$\mathbb{L}$=(SCO, IDUKEY, ADD, AAS, AAA, NUM, FLO)

F^{P}={{2}, {8}, {4}, {5} {6}, {B}, {D, G}}

ψ={(2, SCO), (8, ID), (8, KEY), (4, ADD),(5, AAS), (6, AAA), (*b*, NUM), (*d*, FLO), (*g*, FLO)}

超前搜索

- 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兵乓缓冲：
  - 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。

词法错误

- 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

词法分析例子程序设计要点

- 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

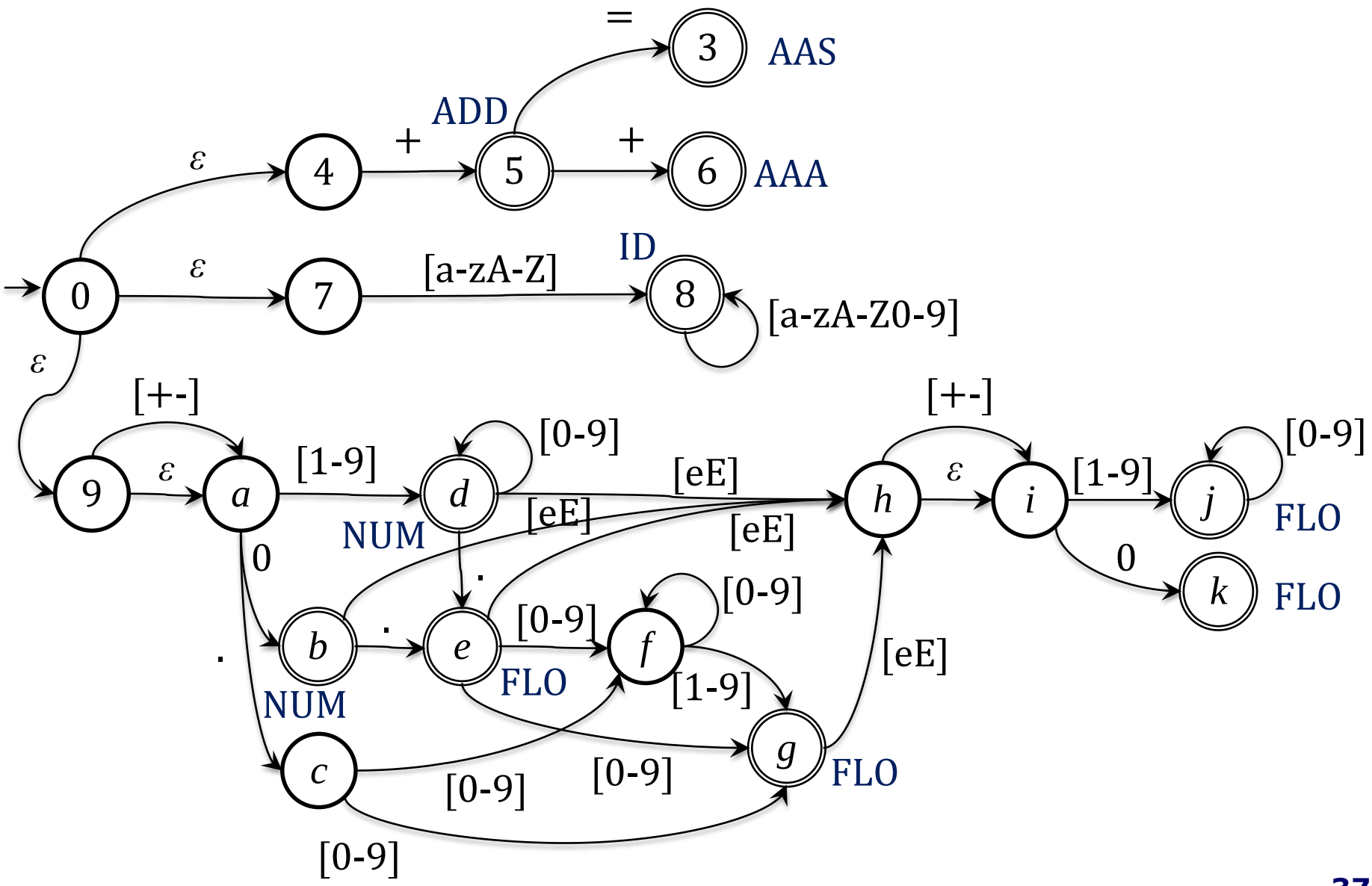
词法分析例子程序设计要点（续）

- 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。

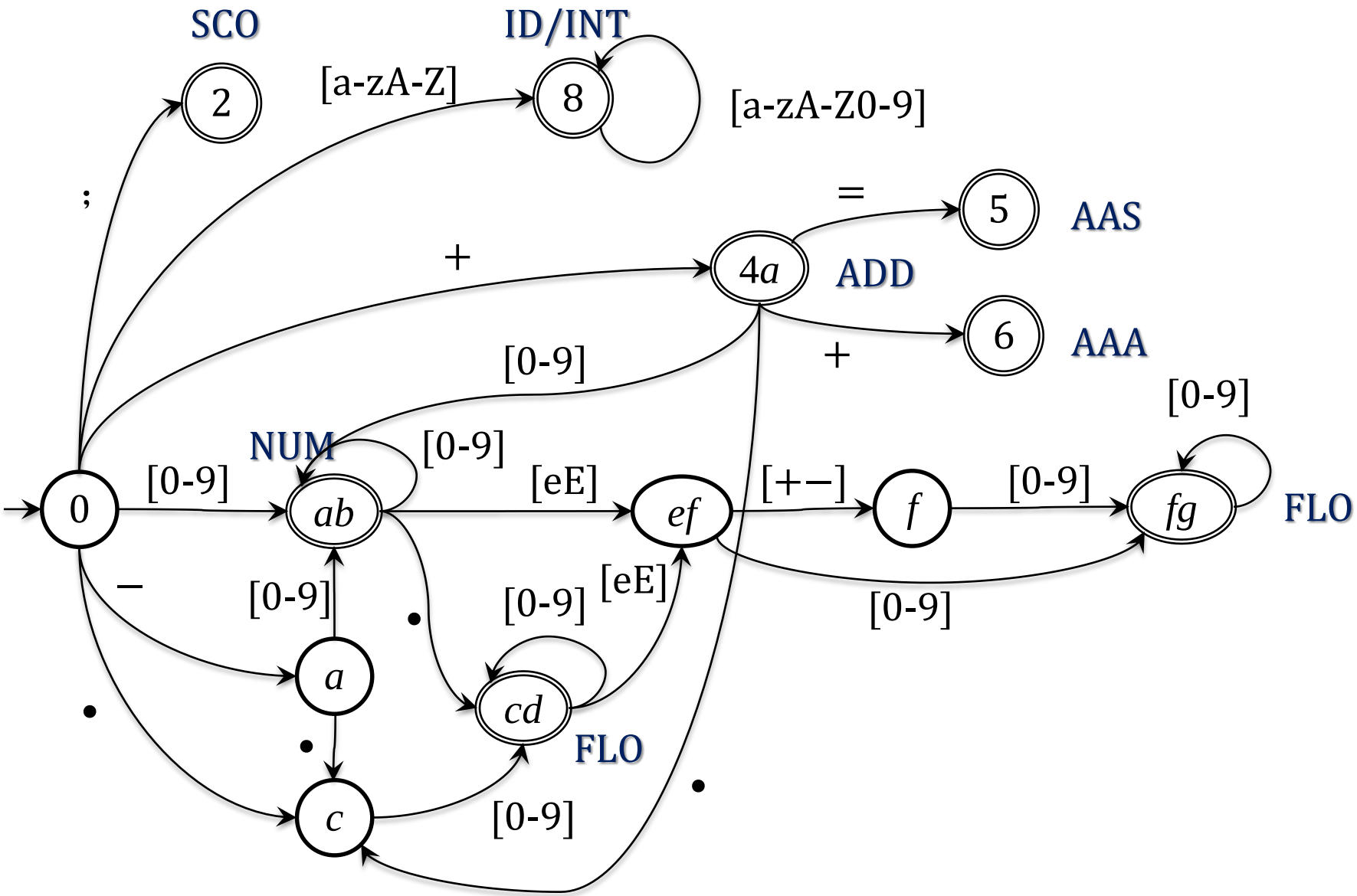
例：联合DFA构建

- 可索引语言簇𝕃=list(ADD AAA NUM FLO IF ID)：
- <usn> ⟶ [1-9][0-9]*|0
- <num> ⟶ [+-]?([1-9][0-9]*|0)
- <fpn> ⟶ [+-]?<usn>?. ([0-9]*[1-9]|0) | [+-]?<usn> .
- <flo> ⟶ <fpn>| <fpn>[e|E]<num> | <num>[e|E]<num>
- <str> ⟶ [A-Za-z0-9]+
- ADD = L(+)
- AAA = L(++)
- NUM = L(<num>)
- FLO = L(<flo>)
- IF = L(if)
- ID=L(<str>)\L(<num>)\L(<flo>)\L(<if>)

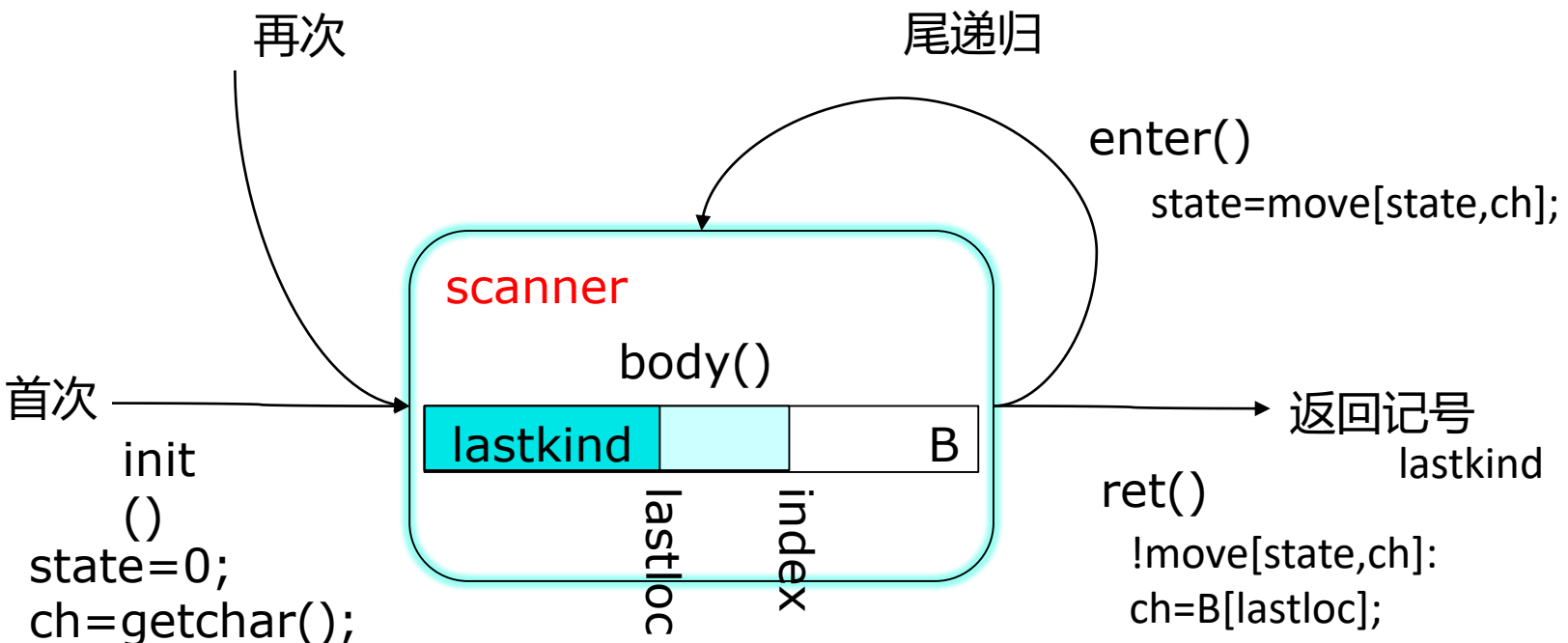
多语言集成NFA


=
3
AAS
ADD
+
+
ε
4
5
6
AAA
ε
ID
0
7
[a-zA-Z]
8
[a-zA-Z0-9]
ε
[+-]
9
ε
a
[1-9]
d
[0-9]
NUM
[eE]
[eE]
[eE]
h
ε
i
[1-9]
j
[0-9]
FLO
0
k
FLO
0
.
.
b
.
e
[0-9]
f
[0-9]
FLO
[1-9]
NUM
c
g
FLO
[eE]
[0-9]
[0-9]
[0-9]

σ-DFA


SCO
ID/INT
2
[a-zA-Z]
8
[a-zA-Z0-9]
;
=
5
AAS
+
4a
ADD
6
AAA
+
[0-9]
[0-9]
NUM
[0-9]
0
[0-9]
ab
[eE]
ef
[+−]
f
[0-9]
fg
FLO
−
[0-9]
[0-9]
[eE]
[0-9]
a
cd
FLO
c
[0-9]

词法分析器概貌


再次
尾递归
enter()
state=move[state,ch];
scanner
body()
首次
lastkind
B
lastloc
index
返回记号
lastkind
init()
state=0;
ch=getchar();
B[0]=ch;
index=0;
lastoc=0;
lastkind=ERR;
state∈F: lastloc=index;
lastkind=state标记;
ch=getchar();
自环a():
while(a(ch=getchar()))ch加入B;
lastloc=index;
其它:
ch=getchar()加入B;
ret()
!move[state,ch]:
ch=B[lastloc];
B[lastloc]='\0';
回退B[index..lastloc];
…;
cleanup()
state=0;
B[0]=ch;
lastoc=index=0;
lastkind=ERR;

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/IF	*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]	-	.

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。

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/IF	*8				8	8			
FLO	*9					9			10
	10		11			12	11		
	11					12			
FLO	*12					12			

词法分析过程示例

- +123.45e-7#

state	ch	textval	index	lastloc	lastkind	ret/en
0	+	+	0	0	ERR	enter
4	1	+1	1	0	ADD	enter
1	2	+12	2	1	NUM	enter
1	3	+123	3	2	NUM	enter
1	.	+123.	4	3	NUM	enter
9	4	+123.4	5	4	FLO	enter
9	5	+123.45	6	5	FLO	enter
9	e	+123.45e	7	6	FLO	enter
10	-	+123.45e-	8	6	FLO	enter
11	7	+123.45e-7	9	6	FLO	enter
12	#	+123.45e-7#	10	9	FLO	enter
⊥	#	+123.45e-7$_{\backslash 0}$	10	9	FLO	ret

C2.2 词法分析器的设计与实现

字符流

<源程序>串

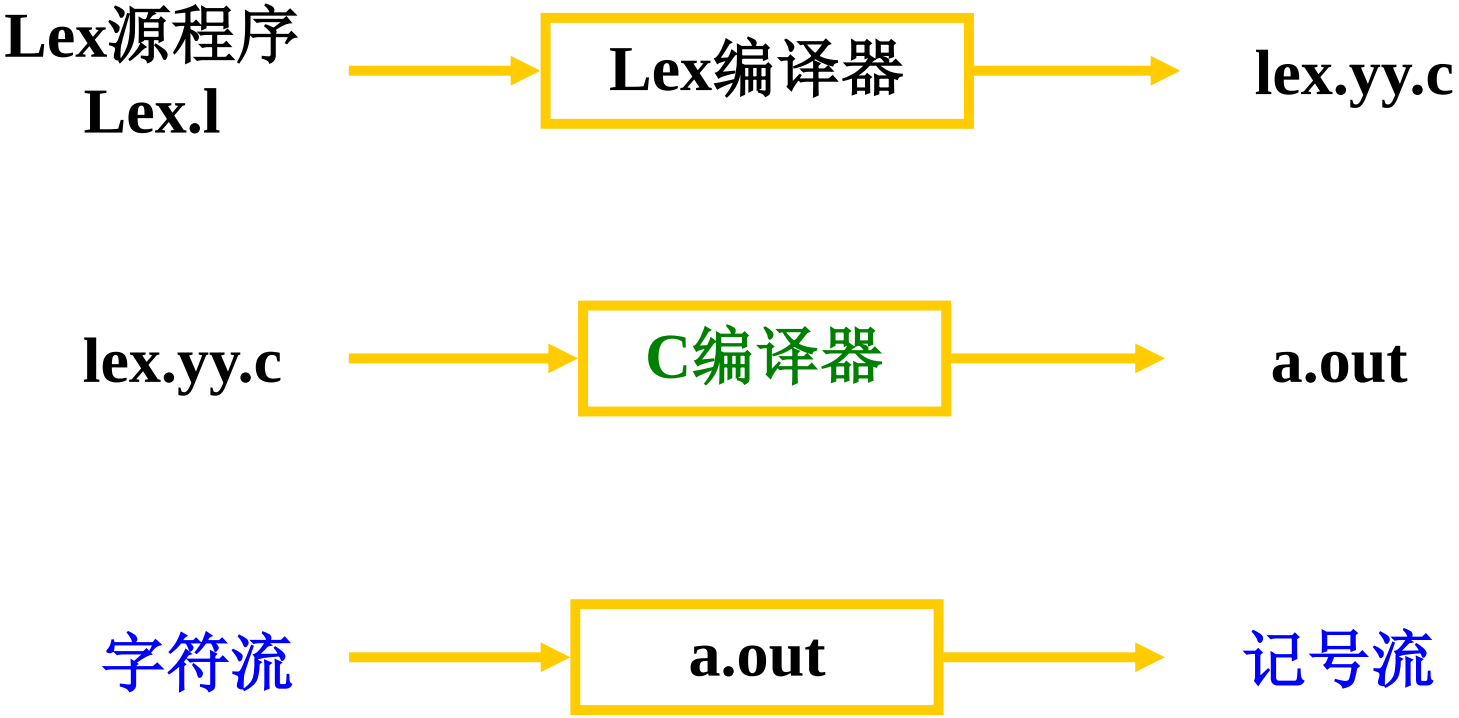
- ASCII字符集上符号串
- 预处理
- 有效串的序列

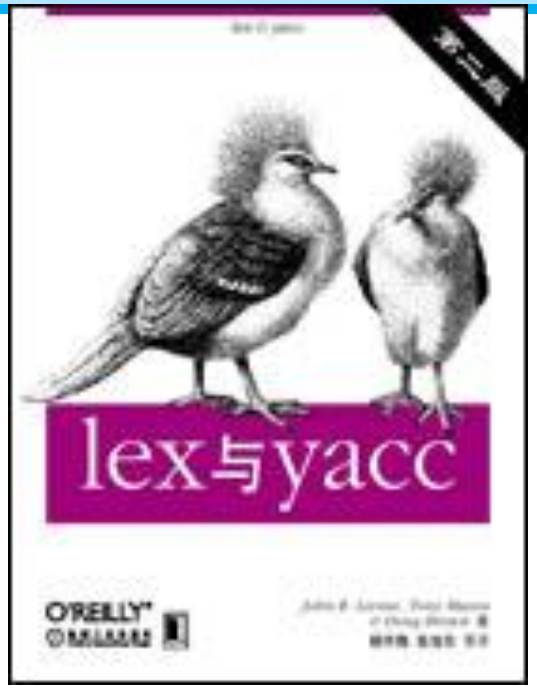
- σ-DFA
- 前缀最大化原则
- 词法单元设计
- 符号表设计

记号流

- (<类别>, <值>)
- 记号类别：
  - 一符一种
  - 全体一种
- 记号值：
  - 符号串
  - 内部表示

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

Lex程序结构

definitions	*变量、常量和正规定义*
%%	r_1 {Actions$_1$}
rules	…
%%	r_n {Actions$_n$}
user code	*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；}
...
```

本章小结

- 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 p121 习题4.2、
- 大作业的词法分析，对给定的$\mathcal{L}$设计$\mathbb{L}$构建σ-DFA($\mathcal{L}$)并给出ψ，写出scanner()程序或流程图，可参考图4-7，最后通过运行测试或手工模拟测试，用于测试的源程序将随后给出。

QLlarge语言文法

- 多维数组；多参数函数；布尔表达式作条件；函数、数组作参数，分支、循环、赋值、复合、过程调用等语句，单个变量、函数类型声明

$P \longrightarrow \check{D}\ \check{S}$
$\check{D} \longrightarrow \varepsilon \mid \check{D}\ D\ ;$
$D \longrightarrow T\ d \mid T\ d[\check{I}] \mid T\ d(\check{A})\ \{\check{D}\ \check{S}\}$
$T \longrightarrow \text{int} \mid \text{void}$
$\check{I} \longrightarrow i \mid \check{I}, i$
$\check{A} \longrightarrow \varepsilon \mid \check{A}\ A\ ;$
$A \longrightarrow T\ d \mid T\ d[] \mid T\ d()$
$\check{S} \longrightarrow S \mid \check{S}\ ;\ S$
$S \longrightarrow d = E \mid \text{if}\ (B)\ S \mid \text{if}\ (B)\ S\ \text{else}\ S \mid \text{while}\ (B)$
$S \mid \text{return}\ E \mid \{\check{S}\} \mid d(\check{R})$
$E \longrightarrow i \mid d \mid d[\check{E}] \mid d(\check{R}) \mid E\ \text{o}\ E \mid (E)$
$\check{E} \longrightarrow E \mid \check{E}, E$
$B \longrightarrow B \wedge B \mid B \vee B \mid E\ r\ E \mid E$
$\check{R} \longrightarrow \varepsilon \mid \check{R}\ R\ ,$
$R \longrightarrow E \mid d[] \mid d()$

词法单元
SCO，ID，LBK，NUM，RBK，LPA，RPA，LBR，RBR，ASG，CMA，INT，VOID，IF，ELSE，WHILE，RETURN，AOP={+，*}，AND，OR，NOT，ROP={<，==}
AOP={ADD，MUL}