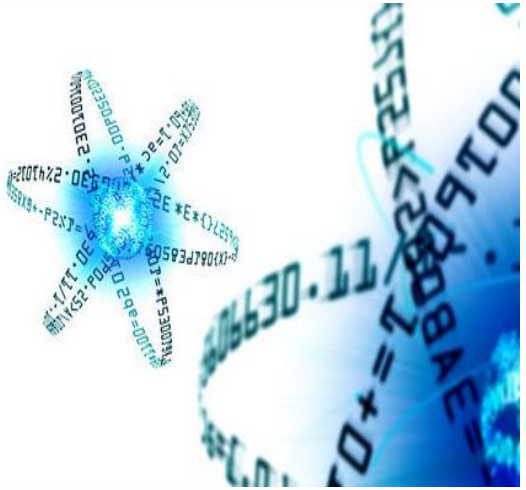
编译原理

Compilers – Principles, Techniques and Tools

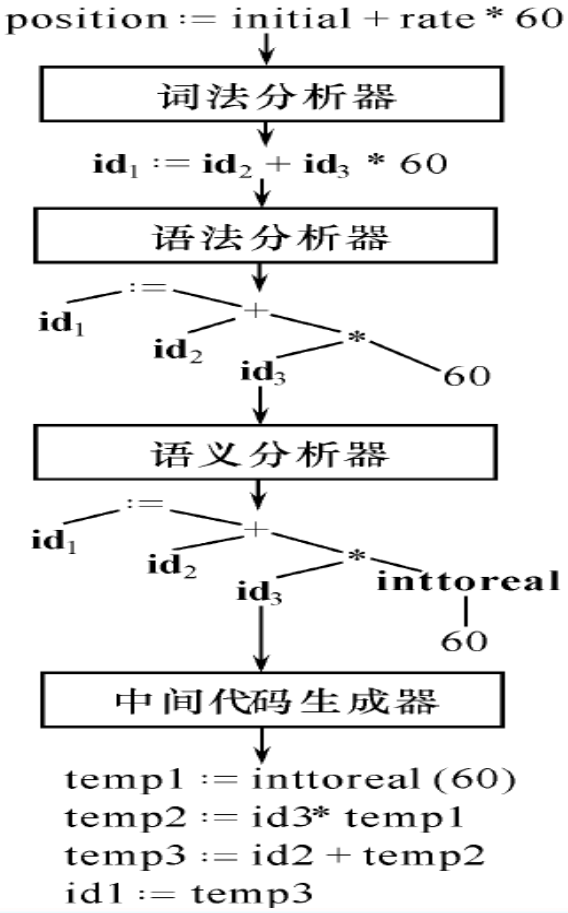
复习



第一章：编译简介

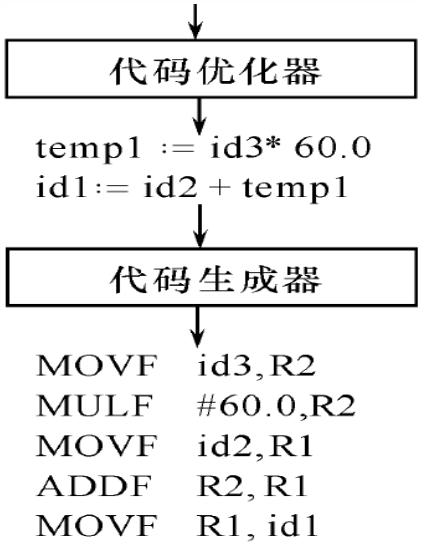
- 编译器的工作可以分成若干阶段，每个阶段把源程序从一种表示变换成另一种表示。

例：



符号表

1	position	. . .
2	initial	. . .
3	rate	. . .
	. . .	. . .



- 正则表达式

正则表达式用来表示简单的语言，叫做正则集

正则式	定义的语言	备注
ε	$\{\varepsilon\}$	
a	$\{a\}$	$a \in \Sigma$
$(r) \mid (s)$	$L(r) \cup L(s)$	r和s是正则式
$(r)(s)$	$L(r)L(s)$	r和s是正则式
$(r)^*$	$(L(r))^*$	r是正则式
(r)	$L(r)$	r是正则式

$((a)\ (b)^*) \mid (c)$可以写成$ab^* \mid c$

词法记号的描述与识别

- 正则定义
  - 对正则式命名，使表示简洁

$$d_1 \rightarrow r_1$$
$$d_2 \rightarrow r_2$$
$$\ldots$$
$$d_n \rightarrow r_n$$

  - 各个d_i的名字都不同
  - 每个r_i都是$\Sigma \cup \{d_1, d_2, \ldots, d_{i-1}\}$上的正则式

词法记号的描述与识别

- 正则定义的例子
  - C语言的标识符是字母、数字和下划线组成的串

letter_ → *A* | *B* | … | *Z* | *a* | *b* | … | *z* | _

digit → 0 | 1 | … | 9

id → letter_(letter_ | digit)*

第三章 词法分析：有限自动机

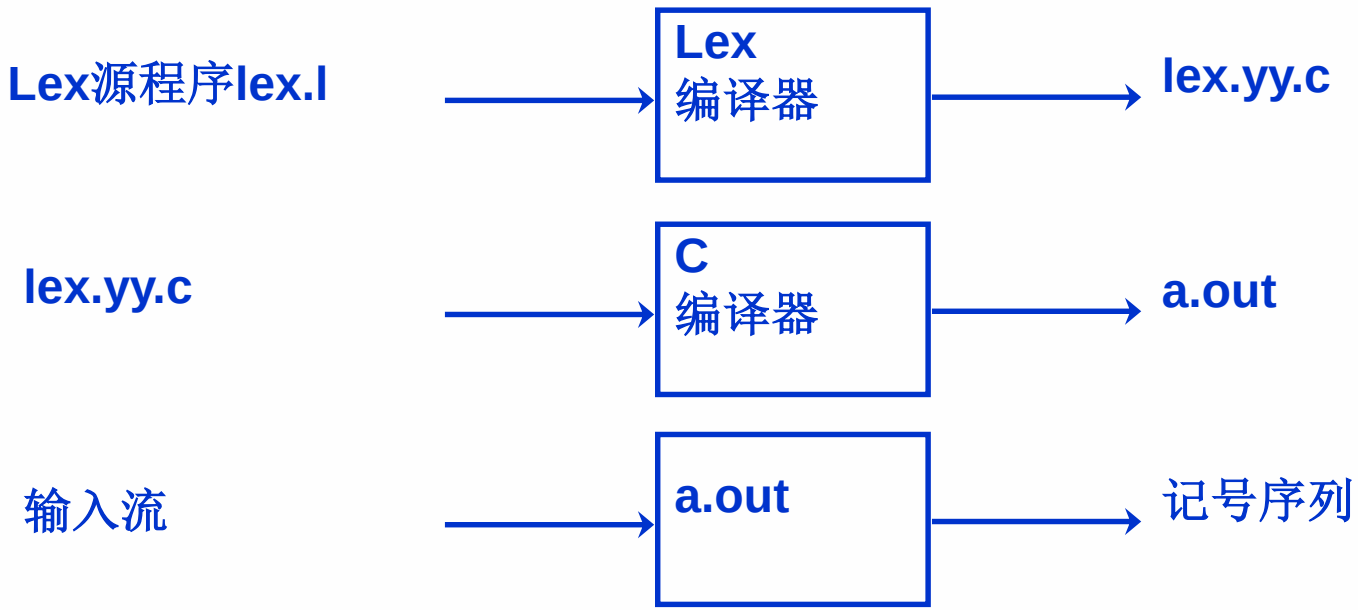
- 不确定的有限自动机（简称NFA）
- 确定的有限自动机（简称DFA）

- NFA → DFA
- DFA化简

- 正则表达式 → NFA → DFA → 最简DFA

第三章 词法分析：词法分析器的生成器

- 用Lex建立词法分析器的步骤



词法分析器的生成器

- **Lex程序包括三个部分**

  声明
  %%
  翻译规则
  %%
  辅助过程

- **Lex程序的翻译规则**

p_1	{动作1}
p_2	{动作2}
…	…
p_n	{动作n}

词法分析器的生成器

- 例——声明部分

```
%{
/* 常量LT, LE, EQ, NE, GT, GE,
      WHILE, DO, ID, NUMBER, RELOP的定义*/
%}
/* 正则定义 */
delim              [  \t \n ]
ws          {delim}+
letter      [A –Za – z]
digit       [0–9]
id          {letter}({letter}|{digit})*
number      {digit}+(\ .{digit}+)?(E[+\–]?{digit}+)?
```

词法分析器的生成器

- 例——翻译规则部分

```
{ws}        {/* 没有动作，也不返回 */}
while       {return (WHILE);}
do          {return (DO);}
{id}        {yylval = install_id ( ); return (ID);}
{number}    {yylval = install_num( );
                                  return (NUMBER);}
“ < ”       {yylval = LT; return (RELOP);}
 “ <= ”     {yylval = LE; return (RELOP);}
“ = ”       {yylval = EQ; return (RELOP);}
“ <> ”      {yylval = NE; return (RELOP);}
“ > ”       {yylval = GT; return (RELOP);}
“ >= ”      {yylval = GE; return (RELOP);}
```

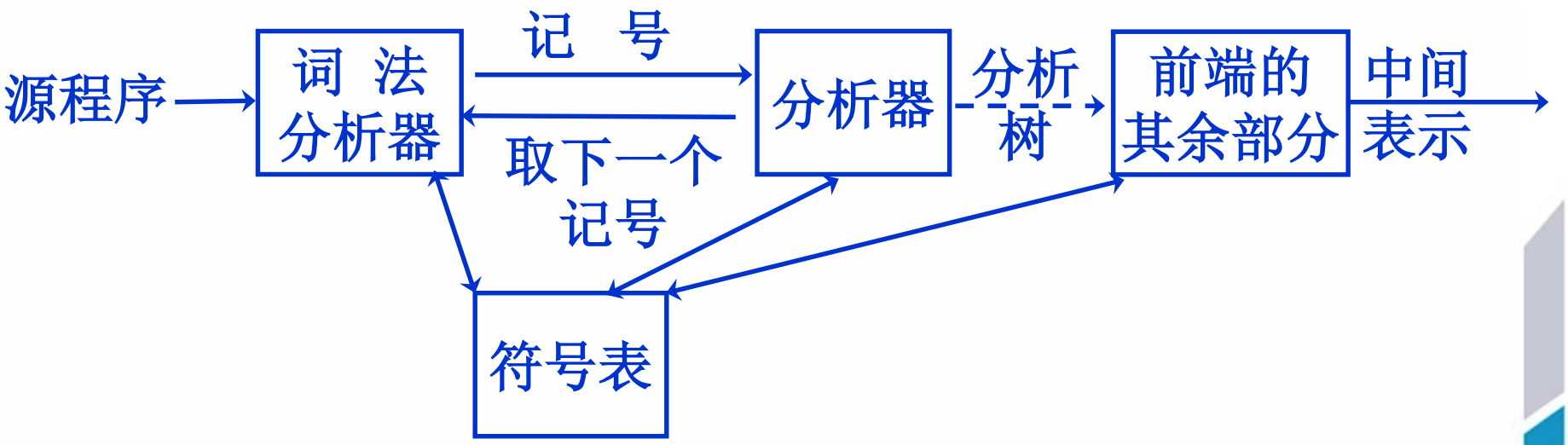
词法分析器的生成器

- 例——辅助过程部分

```
installId ( ) {
        /*  把词法单元装入符号表并返回指针。
        yytext指向该词法单元的第一个字符，
        yyleng给出的它的长度          */
}

installNum ( ) {
        /*  类似上面的过程，但词法单元不是标识符而
    是数 */
}
```

第四章 语法分析



- 本章内容
  - 上下文无关文法
  - 自上而下分析和自下而上分析
  - 围绕分析器的自动生成展开

- 上下文无关文法是四元组（V_T, V_N, S, P）

  V_T：　终结符集合

  V_N：　非终结符集合

  S：　开始符号，非终结符中的一个

  P：　产生式集合，　产生式形式：$A \to \alpha$

- 例 ({id, +, *, –, (,)}, {*expr*, *op*}, *expr*, *P*)

  expr → *expr op expr*　　*expr* → (*expr*)

  expr → – *expr*　　*expr* → id

  op → +　　*op* → *

消除左递归

- 文法左递归 $A \Rightarrow^{+} A\alpha$
- 直接左递归 $A \rightarrow A\alpha \mid \beta$
  - 串的特点 $\beta\alpha \ldots \alpha$
- 消除直接左递归

$$A \rightarrow \beta A'$$
$$A' \rightarrow \alpha A' \mid \varepsilon$$

语言和文法

- 例 算术表达文法

$E \to E + T \mid T$　　（$T + T \ldots + T$）

$T \to T * F \mid F$　　（$F * F \ldots * F$）

$F \to (E) \mid \mathrm{id}$

消除左递归后文法

$E \to TE'$

$E' \to + TE' \mid \varepsilon$

$T \to FT'$

$T' \to * F T' \mid \varepsilon$

$F \to (E) \mid \mathrm{id}$

第四章 语法分析：分析方法

- 自上而下分析方法（预测分析方法）
  - 递归的方法
  - 非递归的方法——**LL方法（Left-to-right Leftmost）**
- 自下而上分析方法
  - **LR方法（Left-to-right Rightmost）**

- **LL(1)文法**
  任何两个产生式$A \to \alpha \mid \beta$都满足下列条件：
  - $\mathbf{FIRST}(\alpha) \cap \mathbf{FIRST}(\beta) = \varnothing$
  - 若$\beta \Rightarrow^* \varepsilon$，那么$\mathbf{FIRST}(\alpha) \cap \mathbf{FOLLOW}(A) = \varnothing$

- **LL(1)文法有一些明显的性质**
  - 没有公共左因子
  - 不是二义的
  - 不含左递归

自上而下分析

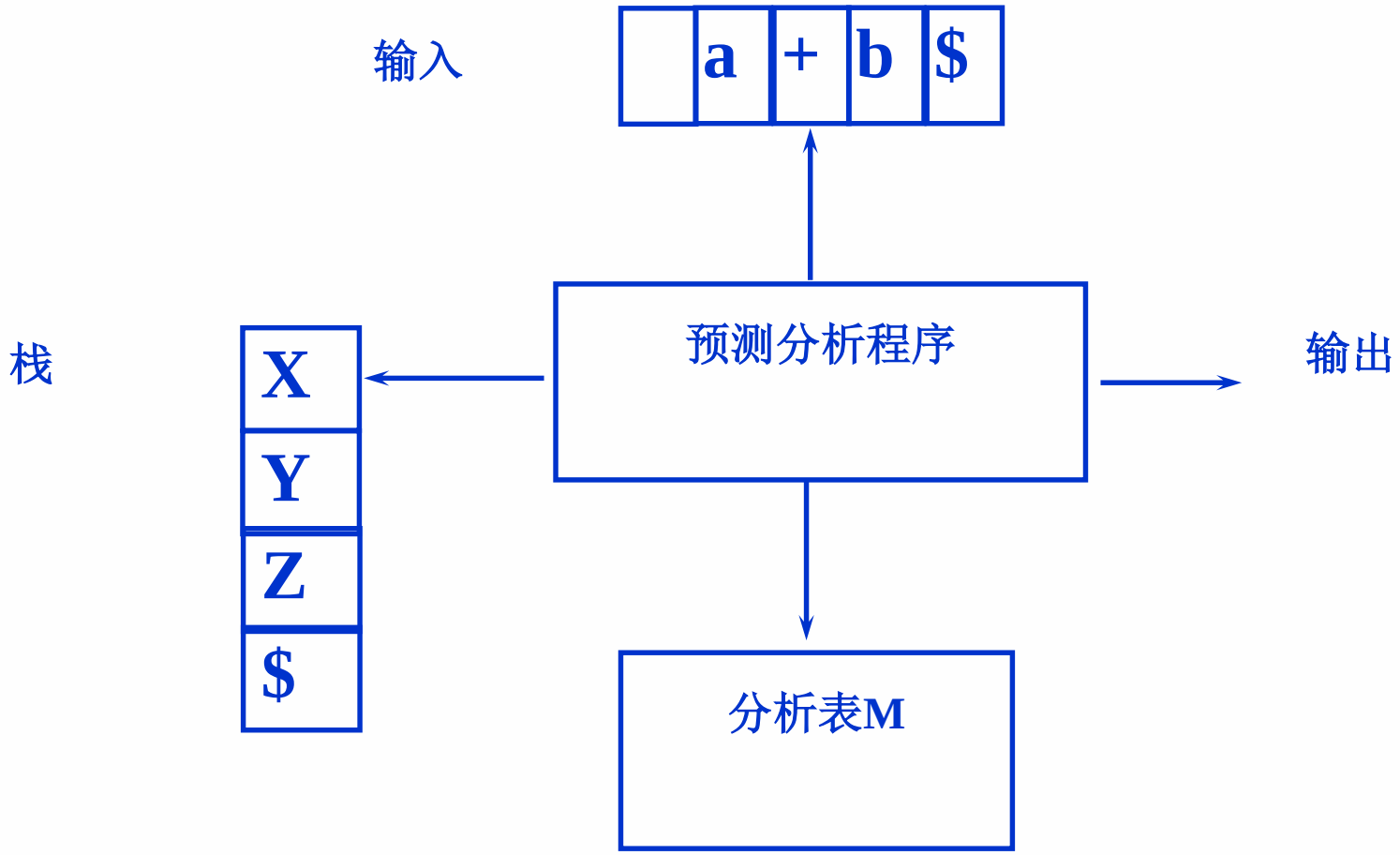
- 递归下降的预测分析
  - 为每一个非终结符写一个分析过程
  - 这些过程可能是递归的
- 例

type → ***simple***
| ↑ id
| array [***simple***] of ***type***

simple → integer
| char
| num dotdot num

自上而下分析

- 非递归的预测分析



自上而下分析

分析表的一部分

非终结符	输入符号			
	id	+	*	…
E	$E \to TE'$			
E'		$E' \to +TE'$		
T	$T \to FT'$			
T'		$T' \to \varepsilon$	$T' \to *FT'$	
F	$F \to \mathrm{id}$			

自上而下分析

预测分析器接受输入id * id + id的前一部分动作

栈	输　入	输　出
$E	id * id + id$	
$E ′T	id * id + id$	$E \rightarrow TE'$
$E ′T ′F	id * id + id$	$T \rightarrow FT'$
$E ′T ′ id	id * id + id$	$F \rightarrow$ id
$E ′T ′	* id + id$	
$E ′T ′F *	* id + id$	$T' \rightarrow *FT'$
$E ′T ′F	id + id$	
$E ′T ′ id	id + id$	$F \rightarrow$ id

自上而下分析

- 构造预测分析表

（1）对文法的每个产生式$A \to \alpha$，执行(2)和(3)

（2）对FIRST(α)的每个终结符a，

把$A \to \alpha$加入$M[A, a]$

（3）如果ε在FIRST(α)中，对FOLLOW(A)的每个终结符b（包括$），把$A \to \alpha$加入$M[A, b]$

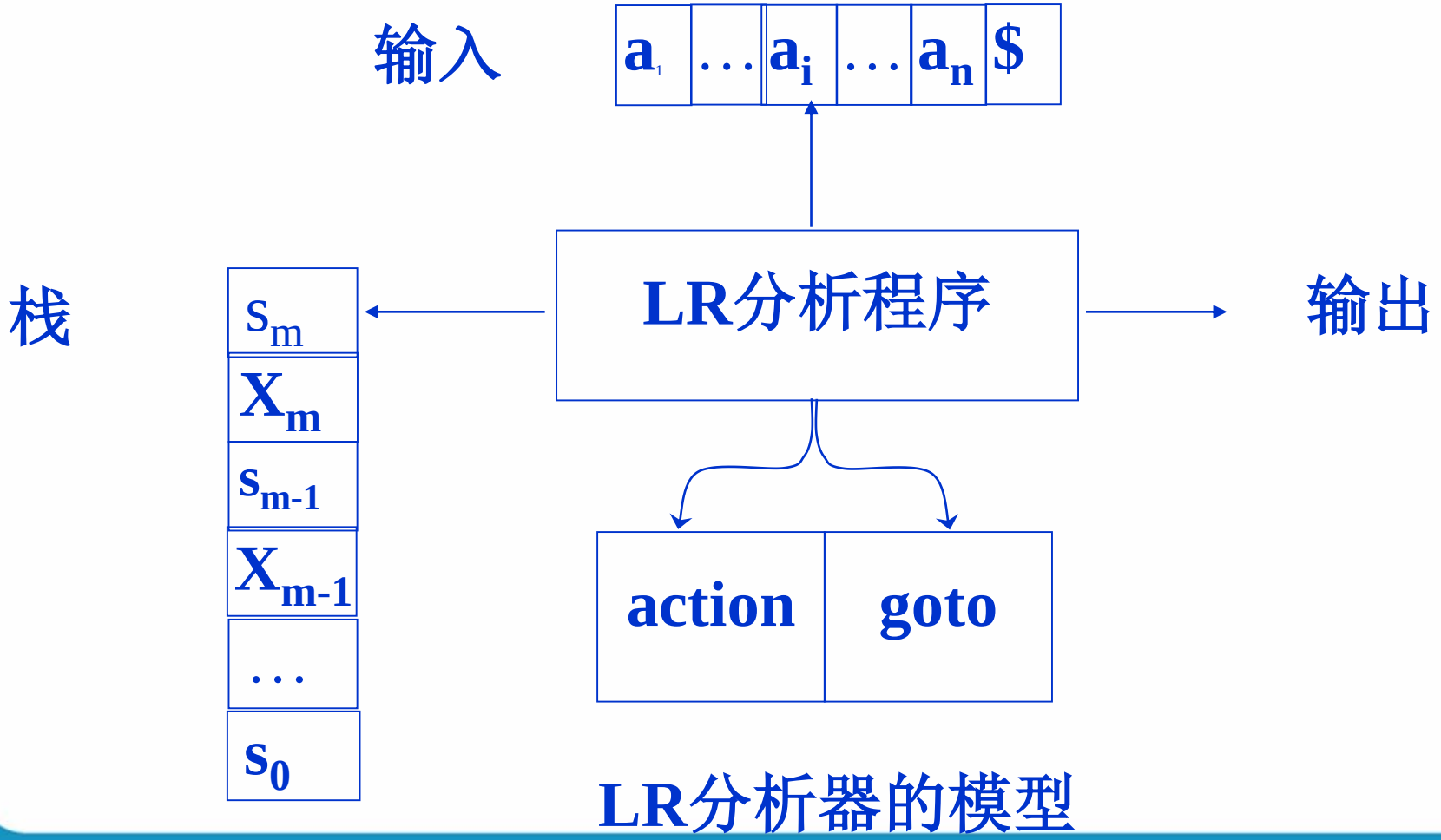
（4）M中其它没有定义的条目都是error

自下而上分析

例：$E \to E+E \mid E*E \mid (E) \mid \mathrm{id}$

栈	输　入	动　作
$	$\mathrm{id}_1 * \mathrm{id}_2 + \mathrm{id}_3$$	移进
$ id_1	$* \mathrm{id}_2 + \mathrm{id}_3$$	按$E \to$ id归约
$$E$	$* \mathrm{id}_2 + \mathrm{id}_3$$	移进
$$E*$	$\mathrm{id}_2 + \mathrm{id}_3$$	移进
$$E*\mathrm{id}_2$	$+ \mathrm{id}_3$$	按$E \to$ id归约
$$E*E$	$+ \mathrm{id}_3$$	移进
$$E*E+$	$\mathrm{id}_3$$	移进
$$E*E+\mathrm{id}_3$	$$	按$E \to$ id归约
$$E*E+E$	$$	按$E \to E+E$归约
$$E*E$	$$	按$E \to E*E$归约
$$E$	$$	接受

LR分析算法



LR分析器的模型

LR分析器

- 例 $E \to E + T \mid E \to T$
  $T \to T * F \mid T \to F$
  $F \to (E) \mid F \to \text{id}$

状态	动作						转移		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3

LR分析器

栈	输　入	动　作
0	id * id + id $	移进
0 id 5	* id + id $	按$F \rightarrow$ id归约
0 F 3	* id + id $	按$T \rightarrow F$归约
0 T 2	* id + id $	移进
0 T 2 * 7	id + id $	移进
0 T 2 * 7 id 5	+ id $	按$F \rightarrow$ id归约
0 T 2 * 7 F 10	+ id $	按$T \rightarrow T*F$归约
. . .	. . .	. . .
0 E 1	$	接受

LR分析器

构造SLR分析表

- 术语：LR(0)项目（简称项目）
  - 在右部的某个地方加点的产生式
  - 加点的目的是用来表示分析过程中的状态
- 例 $A \rightarrow XYZ$对应有四个项目

  $A \rightarrow \cdot XYZ$

  $A \rightarrow X \cdot YZ$

  $A \rightarrow XY \cdot Z$

  $A \rightarrow XYZ \cdot$
- 例 $A \rightarrow \varepsilon$只有一个项目和它对应

  $A \rightarrow \cdot$

LR分析器

构造SLR分析表的两大步骤

1、从文法构造识别可行前缀的DFA
2、从上述DFA构造分析表

LR分析器

1、从文法构造识别可行前缀的DFA

1）拓广文法

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid \mathrm{id}$

LR分析器

1、从文法构造识别可行前缀的DFA

1）拓广文法

$E' \rightarrow E$

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid \mathrm{id}$

LR分析器

1、从文法构造识别可行前缀的DFA

2）构造LR(0)项目集规范族

I_0:

$E' \rightarrow \cdot E$

LR分析器

1、从文法构造识别可行前缀的DFA

2）构造LR(0)项目集规范族

I_0:

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

LR分析器

1、从文法构造识别可行前缀的DFA

2）构造LR(0)项目集规范族

I_0:

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

LR分析器

1、从文法构造识别可行前缀的DFA

2）构造LR(0)项目集规范族

I_0:

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

$F \rightarrow \cdot \mathrm{id}$

LR分析器

1、从文法构造识别可行前缀的DFA

2）构造LR(0)项目集规范族

I_0:

$E' \rightarrow \cdot E$	（核心项目）
$E \rightarrow \cdot E + T$	
$E \rightarrow \cdot T$	（非核心项目，
$T \rightarrow \cdot T * F$	通过对核心项目求闭包
$T \rightarrow \cdot F$	而获得）
$F \rightarrow \cdot (E)$	
$F \rightarrow \cdot \mathrm{id}$	

LR分析器

1、从文法构造识别可行前缀的DFA

2）构造LR(0)项目集规范族

I_0:

$E' \to \cdot E$

$E \to \cdot E + T$

$E \to \cdot T$

$T \to \cdot T * F$

$T \to \cdot F$

$F \to \cdot (E)$

$F \to \cdot \mathrm{id}$

$\xrightarrow{E}$

I_1:

$E' \to E \cdot$

$E \to E \cdot + T$

LR分析器

1、从文法构造识别可行前缀的DFA

2）构造LR(0)项目集规范族

I_0:

$E' \rightarrow \cdot E$

$E \rightarrow \cdot E + T$

$E \rightarrow \cdot T$

$T \rightarrow \cdot T * F$

$T \rightarrow \cdot F$

$F \rightarrow \cdot (E)$

$F \rightarrow \cdot \mathrm{id}$

$\xrightarrow{E}$

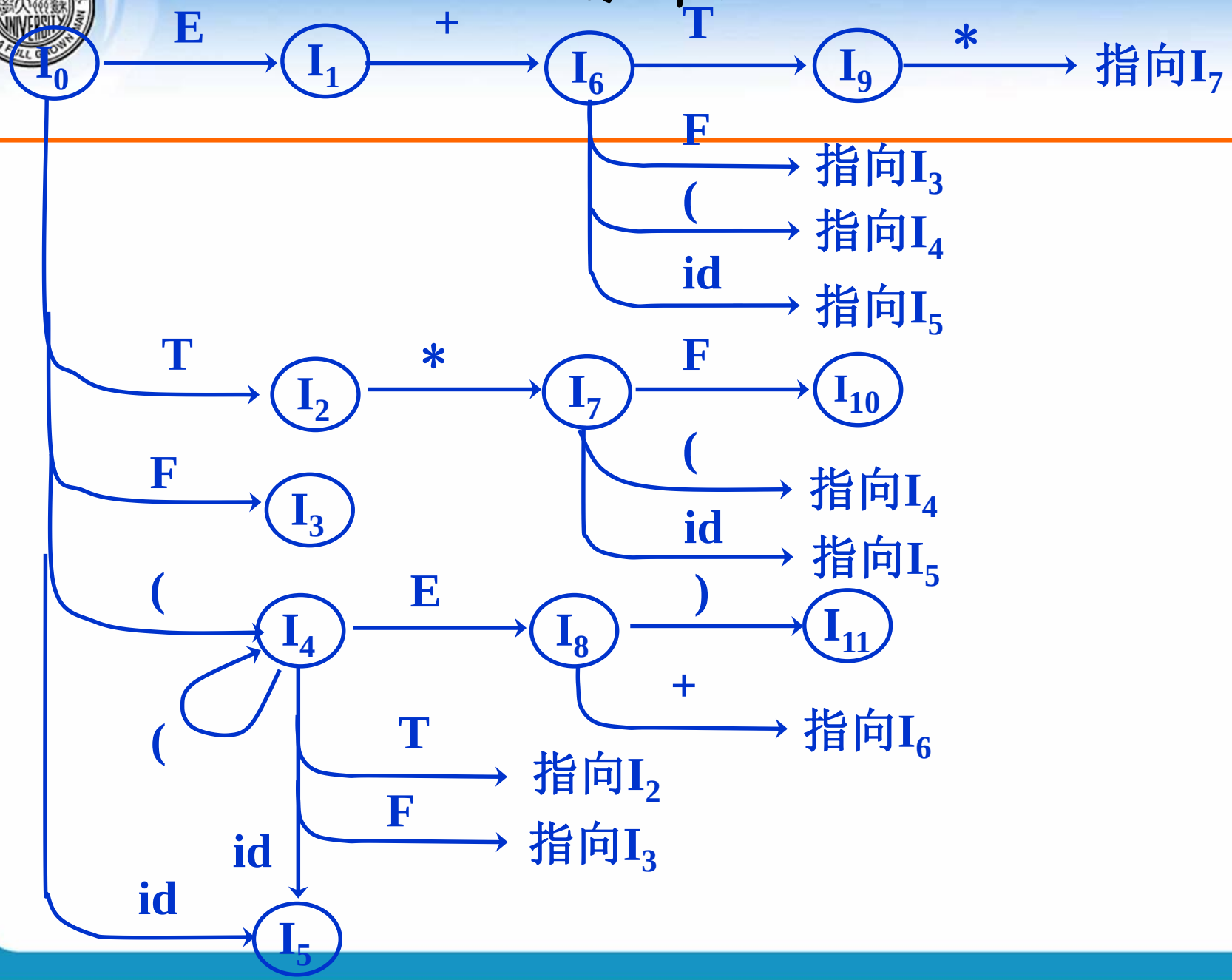
I_1:

$E' \rightarrow E \cdot$

$E \rightarrow E \cdot + T$

$I_1 := goto\ (I_0,\ E)$

LR分析器


E
+
T
*
指向I_7
F
指向I_3
(
指向I_4
id
指向I_5
T
*
F
I_{10}
F
(
指向I_4
id
指向I_5
(
E
)
I_{11}
+
指向I_6
(
T
指向I_2
F
指向I_3
id
id
I_0
I_1
I_6
I_9
I_2
I_7
I_3
I_4
I_8
I_5

LR分析器

构造SLR分析表的两大步骤

1、从文法构造识别可行前缀的DFA
2、从上述DFA构造分析表

LR分析器

2、从DFA构造SLR分析表

- 状态i从I_i构造，它的**action**函数如下确定：
  - 如果$[A\to\alpha\cdot a\beta]$在I_i中，并且$goto(I_i, a) = I_j$，那么置$action[i, a]$为sj
  - 如果$[A\to\alpha\cdot]$在I_i中，那么对FOLLOW(A)中的所有a，置$action[i, a]$为rj, j是产生式 $A\to\alpha$的编号
  - 如果$[S'\to S\cdot]$在I_i中，那么置$action[\ i,\ \$\ ]$为接受$acc$

LR分析器

2、从DFA构造SLR分析表

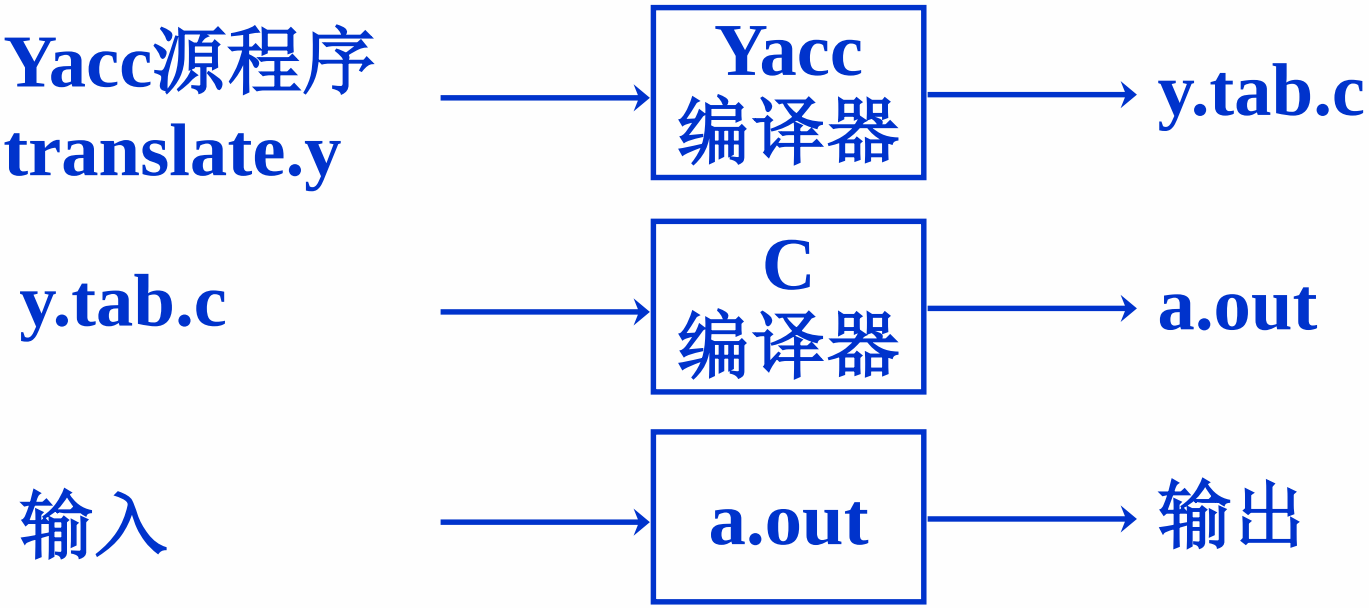
- 状态i从I_i构造，它的**action**函数如下确定：
  - . . .
- 使用下面规则构造状态i的**goto**函数：
  - 对所有的非终结符A，如果$goto(I_i, A) = I_j$，那么$goto[i, A] = j$

LR分析器

2、从DFA构造SLR分析表

- 状态*i*从I_i构造，它的*action*函数如下确定：
  - . . .
- 使用下面规则构造状态*i*的*goto*函数：
  - . . .
- 不能由上面两步定义的条目都置为error
- 分析器的初始状态是包含[$S' \rightarrow \cdot S$]的项目集对应的状态

- 分析器的生成器Yacc



分析器的生成器

```
%{
# include  <ctype .h>
# include  <stdio.h >
#  define  YYSTYPE  double  /*将栈定义为double类
   型 */
%}

%token  NUMBER
%left  '+'  '–'
%left  '*'  '/ '
%right  UMINUS
%%
```

分析器的生成器

```
lines     : lines expr '\n'    {printf ( "%g \n", $2 ) }
          | lines '\n'
          |  /* ε */
          ;
expr      : expr '+' expr      {$$ = $1 + $3; }
          | expr '–' expr      {$$ = $1 – $3; }
          | expr '*' expr      {$$ = $1 * $3; }
          | expr '/ ' expr     {$$ = $1 / $3; }
          | '(' expr ')'       {$$ = $2; }
          | '–' expr  %prec UMINUS    {$$ = –$2; }
          | NUMBER
          ;
%%
```

分析器的生成器

```
yylex ( ) {
        int c;
        while ( ( c = getchar ( ) ) == ‘  ’ );
        if ( ( c == ‘.’ ) | | (isdigit (c) ) ) {
                ungetc (c, stdin);
                scanf ( “% lf ”,  &yylval);
                return  NUMBER;
        }
        return c;
}
```

第五章　语法制导的翻译

- 本章内容

1、介绍语义描述的一种形式方法：语法制导的翻译，它包括两种具体形式

- 语法制导的定义
- 翻译方案

2、介绍语法制导翻译的实现方法

语法制导定义的形式

- 基础文法
- 每个文法符号有一组属性
- 每个文法产生式$A \rightarrow \alpha$有

  一组形式为$b=f(c_1, c_2, \ldots, c_k)$的语义规则，其中$b$和$c_1, c_2, \ldots, c_k$是该产生式文法符号的属性，$f$是函数
- 综合属性：如果b是A的属性，$c_1, c_2, \ldots, c_k$是产生式右部文法符号的属性或A的其它属性
- 继承属性：如果b是右部某文法符号X的属性

语法制导的定义

综合属性

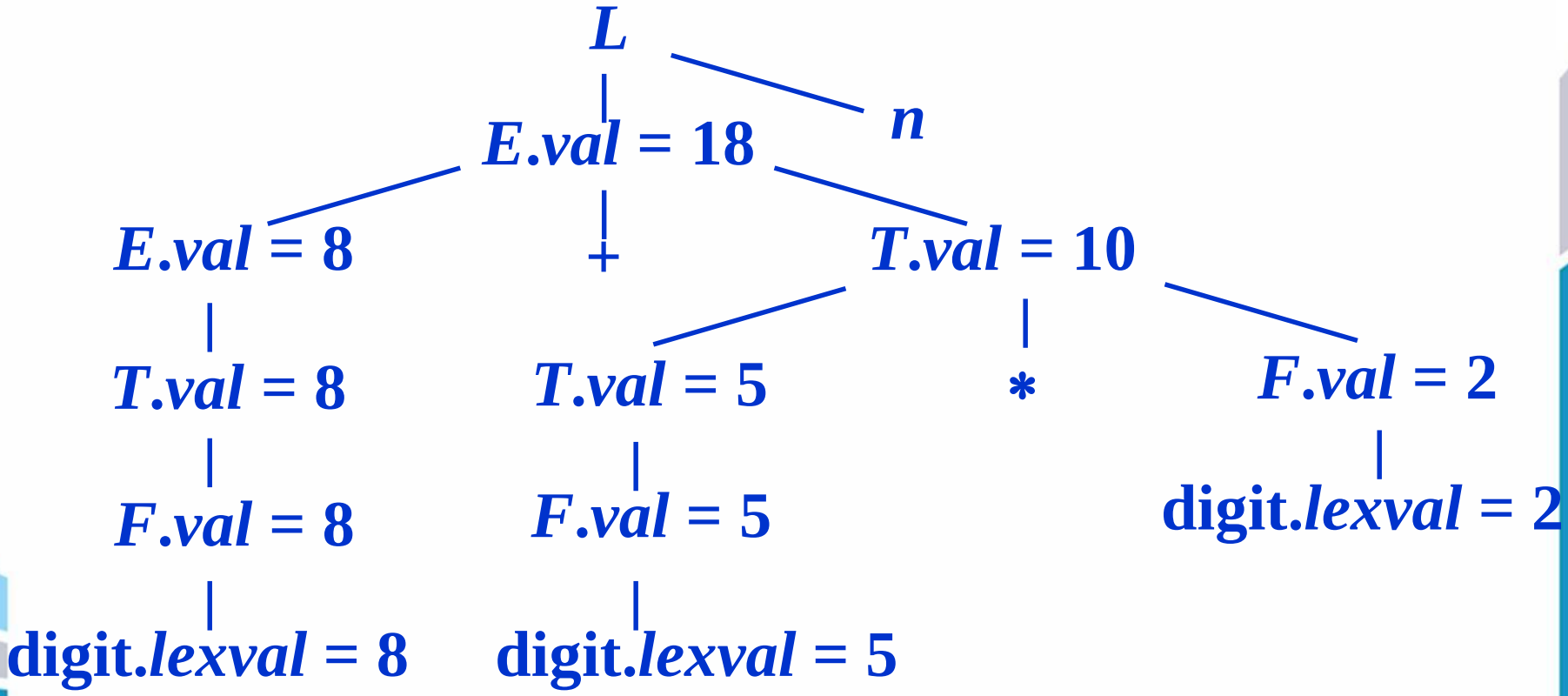
S属性定义：仅使用综合属性的语法制导定义

产 生 式	语 义 规 则
$L \rightarrow E$ n	*print* (*E.val*)
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow$ digit	$F.val =$ digit.*lexval*

语法制导的定义

注释分析树:结点的属性值都标注出来的分析树

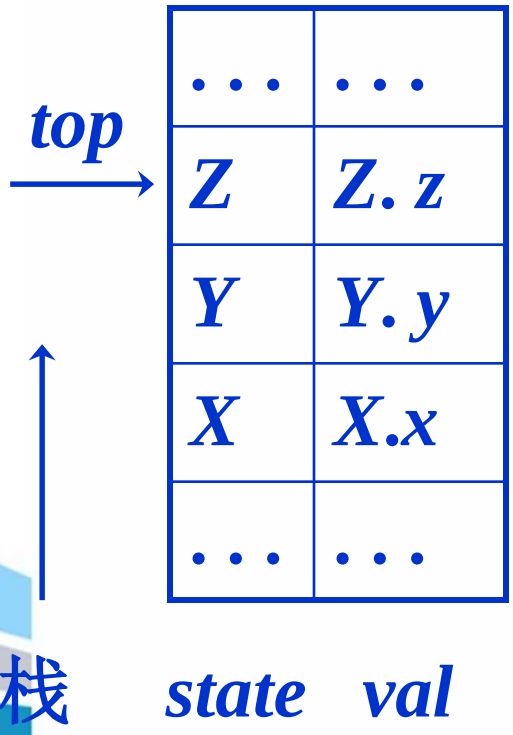
8+5*2 n的注释分析树



第五章 语法制导的翻译:
S属性定义的自下而上计算

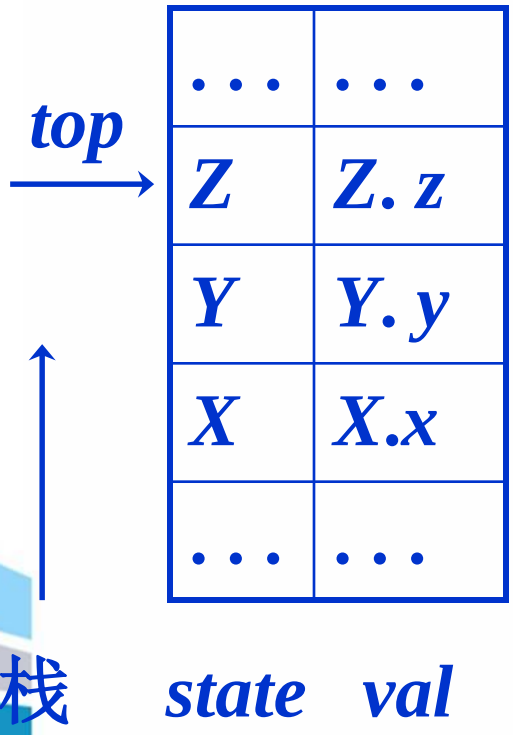
简单计算器的语法制导定义改成栈操作代码

产 生 式	语 义 规 则
$L \rightarrow E$ n	*print* (*E.val*)
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	*E.val* = *T.val*
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	*T.val* = *F.val*
$F \rightarrow (E)$	*F.val* = *E.val*
$F \rightarrow$ digit	*F.val* = digit.*lexval*



S属性定义的自下而上计算

简单计算器的语法制导定义改成栈操作代码



产 生 式	代 码 段
$L \rightarrow E$ n	*print* (*val* [*top*–1])
$E \rightarrow E_1 + T$	*val* [*top* –2] = *val* [*top* –2]+*val* [*top*]
$E \rightarrow T$	
$T \rightarrow T_1 * F$	*val* [*top* –2] = *val* [*top* –2]×*val* [*top*]
$T \rightarrow F$	
$F \rightarrow (E)$	*val* [top –2] = *val* [top –1]
$F \rightarrow$ digit	

继承属性

int id, id, id

产 生 式	语 义 规 则
$D \rightarrow TL$	$L.in = T.type$
$T \rightarrow$ int	$T.type = integer$
$T \rightarrow$ real	$T.type = real$
$L \rightarrow L_1$, id	$L_1.in = L.in$; $addType(\text{id}.entry, L.in)$
$L \rightarrow$ id	$addType(\text{id}.entry, L.in)$

L属性定义的自上而下计算

L属性定义

- 如果每个产生式$A \rightarrow X_1 \ldots X_{j-1} X_j \ldots X_n$的每条语义规则计算的属性是$A$的综合属性；或者是$X_j$的继承属性，但它仅依赖：
  - 该产生式中X_j左边符号$X_1, X_2, \ldots, X_{j-1}$的属性；
  - A的继承属性
- S属性定义属于L属性定义

L属性定义的自上而下计算

- 预测翻译器的设计
  - 为每个非终结符A构造一个函数，A的每个继承属性作为形参，A的综合属性作为返回值
  - 产生式$R \rightarrow +TR \mid \varepsilon$ 的分析过程

```
syntaxTreeNode* R (syntaxTreeNode* i) {
    syntaxTreeNode *nptr, *i1, *s1, *s;
    char addoplexeme;

    if (lookahead == '+' ) { /* 产生式 R → +T R */
      addoplexeme = lexval;
      match('+' ); nptr = T();
      i1 = mkNode(addoplexeme, i , nptr);
      s1 = R (i1); s = s1;
    }
    else  s = i;  /* 产生式 R → ε */
    return s;
}
```

$R : i, s$

$T : nptr$

$+ : addoplexeme$

L属性的自下而上计算

分析栈上的继承属性

1、属性位置能预测

例 int p, q, r

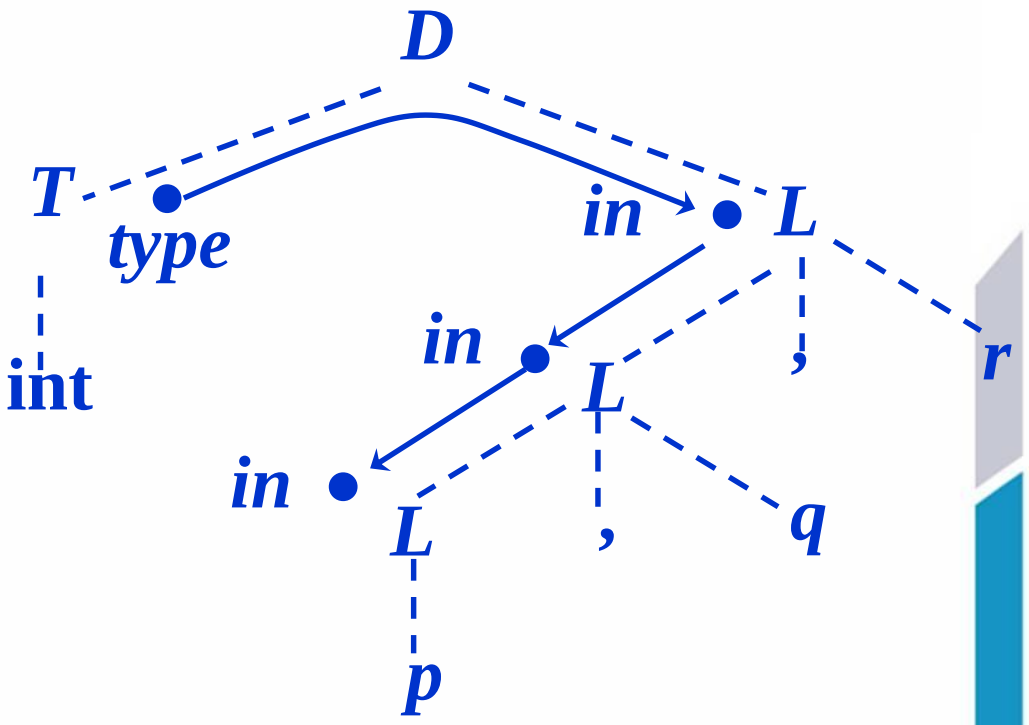
$D \rightarrow T$ {*L.in = T.type*}
　　L

$T \rightarrow$ int {*T. type = integer*}

$T \rightarrow$ real {*T. type = real*}

$L \rightarrow$ {L_1*.in = L.in* }
　L_1, id {*addtype* (id.*entry*, *L.in*)}

$L \rightarrow$ id {*addtype* (id.*entry*, *L.in*)}



继承属性的计算可以略去，引用继承属性的地方改成引用其他符号的综合属性

L属性的自下而上计算

产 生 式	代 码 段
$D \rightarrow TL$	
$T \rightarrow$ int	$val[top] = integer$
$T \rightarrow$ real	$val[top] = real$
$L \rightarrow L_1$, id	$addType(val[top], val[top-3])$
$L \rightarrow$ id	$addType(val[top], val[top-1])$