第三章：语法分析

LL(1)语法分析

1. LL(1)语法分析原理

- 基本思想

从左到右扫描，按最左推导的方式推出输入流

- 定义：

对于文法G中任一非终极符A，其任意两个产生式$A\rightarrow\alpha$和$A\rightarrow\beta$，都要满足下面条件：

$Predict(A\rightarrow\alpha)\cap Predict(A\rightarrow\beta)=\varnothing$

满足这一条件的文法称为LL(1)文法。

1. LL(1)语法分析原理

- LL(1)是LL(k)的特例, 其中的k则表示向前看k个符号.
- LL(1)方法和递归下降法属于同一级别的自顶向下分析法.

用“LL(1)”命名该分析方法的原因：

●第一个L表示：相应的语法分析将按自左至右的顺序扫描输入符号串；

●第二个L表示：在分析过程中产生一个句子的最左推导；

●括号中的“1”，则表示在分析过程中，每进行一步推导，只要查看一个输入符号便能确定当前所应选用的产生式.

1. LL(1)语法分析原理

- 语法分析的动作
  - 替换 当分析栈的第一个符号是V_N时，就要用规则进行推导，用规则右部将其替换
  - 匹配 分析栈第一个符号是V_T时，与输入流中的第一个符号进行匹配
  - 成功
  - 出错、失败

例 G[z] :

[1] Z →aBe	{a}	[2] Z →Bd	{b,c}
[3] B →bB	{b}	[4] B →cK	{c}
[5] D → d	{d}	[6] K → ε	{d,e}

对给定的终极符串ace，分析过程：

Z #	ace #	替换
aBe #	ace #	匹配
Be #	ce #	替换
cKe #	ce #	匹配
Ke #	e #	替换
e #	e #	匹配
#	#	成功

例 G[z]：

[1] Z →aBe	{a}	[2] Z →Bd	{b,c}
[3] B →bB	{b}	[4] B →cK	{c}
[5] D → d	{d}	[6] K → ε	{d,e}

对给定的终极符串acb，分析过程：

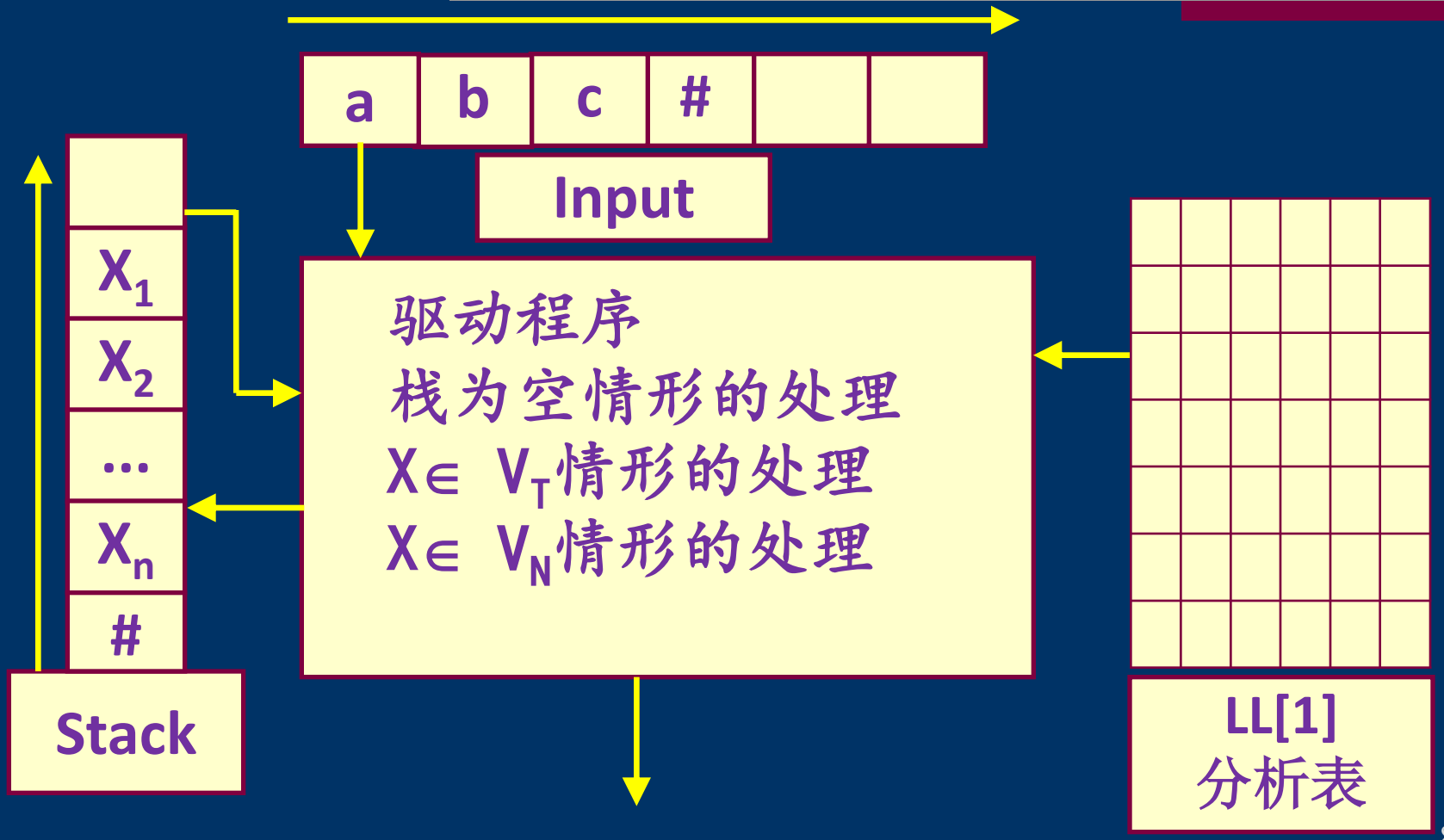
Z #	acb #	替换
aBe #	acb #	匹配
Be #	cb #	替换
cKe #	cb #	匹配
Ke #	b #	出错

1. LL(1)语法分析原理

在逻辑上，一个LL(1)分析器由输入流、LL(1)分析表、符号栈和驱动程序组成：

- 输入流:待分析的符号串 .
- 符号栈:存放分析过程中的文法符号串.
- LL(1)分析表：用来表示相应文法的全部信息 的一个矩阵（或二维数组）.
- 驱动程序：语法分析程序.

1. LL(1)语法分析原理


a
b
c
#
Input
X_1
X_2
...
X_n
#
Stack
驱动程序
栈为空情形的处理
$X \in V_T$情形的处理
$X \in V_N$情形的处理
LL[1]
分析表

2.1 LL(1)分析表的构造

- **LL(1)分析表的定义：**

$$T: V_N \times V_T \rightarrow P \cup \{\text{Error}\}$$

$$\begin{cases} T(A,t)=A\rightarrow\alpha \text{ 若} t\in Predict(A\rightarrow\alpha) \\ T(A,t)=Error \text{ 否则} \end{cases}$$

其中P表示所有产生式的集合.

2.1 LL(1)分析表的构造

- **LL(1)分析表的构造方法：**
  - 对文法的每一个产生式求其predict集；
  - 对文法的每一个产生式$A\rightarrow\alpha$进行如下处理：

    若：$Predict(A\rightarrow\alpha)=(a_1, a_2, \cdots, a_n)$；

    则令：$T(A, a_i)=A\rightarrow\alpha$

    其中：$i=1, 2, 3\cdots, n$
  - LL(1)分析表的其它元素为error.

2.2 求LL(1)分析表的实例

- 文法G:

$$E \rightarrow T\ E'\ [1]$$

$$E' \rightarrow +\ T\ E'\ [2] \mid \varepsilon\ [3]$$

$$T \rightarrow F\ T'\ [4]$$

$$T' \rightarrow *\ F\ T'\ [5] \mid \varepsilon\ [6]$$

$$F \rightarrow i\ [7] \mid (\ E\ )\ [8]$$

Predict([1]) = first(TE’) = { i , (}

Predict([2]) = first(+TE’) = { + }

Predict([3]) = follow(E’) = {) , # }

Predict([4]) = first(FT’) = { i , (}

Predict([5]) = first(*FT’) = { * }

Predict([6]) = follow(T’) = { + ,) , # }

Predict([7]) = first(i) = { i }

Predict([8]) = first((E)) = { (}

	i	+	*	(	)	#
E	1			1		
E’		2			3	3
T	4			4		
T’		6	5		6	6
F	7			8		

$E \rightarrow T\ E'$[1]

$E' \rightarrow +\ T\ E'$[2]

$\quad | \ \varepsilon$[3]

$T \rightarrow F\ T'$[4]

$T' \rightarrow *\ F\ T'$[5]

$\quad | \ \varepsilon$[6]

$F \rightarrow i$[7]

$\quad | \ (\ E\ )$[8]

Predict([1]) = { i , (}

Predict([2])= { + }

Predict([3]) = {) , # }

Predict([4]) = { i , (}

Predict([5]) = { * }

Predict([6] = { + ,) , # }

Predict([7]) = { i }

Predict([8]) = { (}

分析示例1

分析示例2

3.1驱动程序的设计

- 分析的初始格局（S#，$a_1 \ldots a_n$#）
- 一般格局（$X_1 \ldots X_m$#，$a_i \ldots a_n$#）

设存在格局为（$X_1 \ldots X_m$#，$a_1 \ldots a_n$#）

- 若 $X_1 \in V_T$ & $X_1 = a_1$ 则有（$X_2 \ldots X_m$#，$a_2 \ldots a_n$#）
- 若 $X_1 \in V_N$则查表，若$T(X_1, a_1) = X_1 \rightarrow \alpha$，格局为（$\alpha X_2 \ldots X_m$#，$a_1 \ldots a_n$#），若$T(X_1, a_1)$=error，则报错。
- 若格局为（#，#），则分析成功
- 其他情况 报错

3. 2驱动程序的实现

[1]初始化： Stack := empty ; Push(#); Push(S);
即由符号栈和输入流构成的初始格局为：

(#S, $a_1a_2...a_n$#)

[2] 读第一个输入符： Read(a);

[3] 若当前格局是(#, #)，则成功结束；否则转下一步；

[4] 设当前格局为（..... X, a.....)，则：

- 若$X \in V_T$ & X= a ，则：
  { Pop(1)； Read(a)；goto [3] }
- 若$X \in V_T$ & $X \neq a$ ，则 Error;
- 若$X \in V_N$ ，则：
  if $T(X, a)=X \rightarrow Y_1 Y_2 ..... Y_n$
  then { Pop(1)； $Push(Y_n,....,Y_1)$；goto[3] }
  else Error

- 例:文法G:

$$E \rightarrow T\ E'^{[1]}$$

$$E' \rightarrow +\ T\ E'^{[2]} \mid \varepsilon^{[3]}$$

$$T \rightarrow F\ T'^{[4]}$$

$$T' \rightarrow *\ F\ T'^{[5]} \mid \varepsilon^{[6]}$$

$$F \rightarrow i^{[7]} \mid (\ E\ )^{[8]}$$

符号串 i + i * i # 的LL[1]分析过程:

分析栈	输入流	矩阵元素
# E	i + i * i #	LL[E ,i] = [1]
# E' T	i + i * i #	LL [T ,i] = [4]
# E' T' F	i + i * i #	LL [F ,i] = [7]
# E' T' i	i + i * i #	Match
# E' T'	+ i * i #	LL [T',+] = [6]
# E'	+i * i #	LL [E',+] = [2]
# E' T+	+i * i #	Match
# E' T	i * i #	LL [T,i] =[4]
# E' T' F	i* i #	LL [F,i] = [7]
# E' T' i	i * i #	Match
# E' T'	* i #	LL [T',*] = [5]
# E'T'F*	* i #	Match
# E'T'F	i #	LL[F,i] = [7]
# E'T'i	i #	Match
# E'T'	#	LL[T',#] = [6]
# E'	#	LL[E', #] = [3]
#	#	ok

$E \rightarrow T\ E'^{[1]}$

$E' \rightarrow +\ T\ E'^{[2]}$

$\quad | \ \varepsilon^{[3]}$

$T \rightarrow F\ T'^{[4]}$

$T' \rightarrow *\ F\ T'^{[5]}$

$\quad | \ \varepsilon^{[6]}$

$F \rightarrow i^{[7]}$

$\quad | \ (\ E\ )^{[8]}$

分析表

分析栈	输入流	矩阵元素
# E	i + i *+ i #	LL[E ,i] = [1]
# E' T	i + i * +i #	LL [T ,i] = [4]
# E' T' F	i + i * +i #	LL [F ,i] = [7]
# E' T' i	i + i * +i #	Match
# E' T'	+ i * +i #	LL [T',+] = [6]
# E'	+i * +i #	LL [E',+] = [2]
# E' T+	+i * +i #	Match
# E' T	i *+i #	LL [T,i] =[4]
# E' T' F	i* + i #	LL [F,i] = [7]
# E' T' i	i * + i #	Match
# E' T'	*+ i #	LL [T',*] = [5]
# E'T'F*	*+ i #	Match
# E'T'F	+i #	LL[F,+] =Error

$E \rightarrow T\ E'$[1]

$E' \rightarrow +\ T\ E'$[2]

$\quad | \ \varepsilon$[3]

$T \rightarrow F\ T'$[4]

$T' \rightarrow *\ F\ T'$[5]

$\quad | \ \varepsilon$[6]

$F \rightarrow i$[7]

$\quad | \ (\ E\ )$[8]

分析表

3.3 注意的一些问题

- 错误的处理
- LL(1)方法对文法的限制

假如不满足限定：

- 进行文法等价变换，使其满足我们的限定
- 不是所有的文法都适用于该方法，有时进行文法等价变换会破坏其可读性

- 通常来说，对于高级语言的语法分析也足够用了。

判断是否为LL(1)文法

1. $S \rightarrow (SS' \mid \varepsilon$
   $S' \rightarrow) \mid \varepsilon$

Predict(S→(SS')={(}
Predict(S→ε)={),#}
Predict(S'→))={)}
Predict(S'→ε)={),#}

2. $S \rightarrow (S \mid S'$
   $S' \rightarrow (S') \mid \varepsilon$

Predict(S→(S)={(}
Predict(S→S')={(,#}
Predict(S'→(S'))={(}
Predict(S'→ε)={),#}

构造LL(1)分析表

S→aBc[1]

S→bAB[2]

A→aAb[3]

A→b[4]

B→b[5]

B→ε[6]