4.4 自顶向下的语法分析 →∪Σεαβ

语法分析: 给定某个文法, 为输入串(或称句子) 构造语法分析树的过程。

自顶向下: 它从语法树的根结点开始（即文法的起始符号）,按照先根次序（深度优先地）创建语法分析树的各个结点。

自顶向下语法分析也可以被看作寻找输入串的最左推导的过程.

例1

E → TE'

E' → +TE' | ε

T → F T'

T' → \* F T' | ε

F → (E) | **id**

为输入串 id + id \* id 自顶向下构造分析树

4.4.1 递归下降的语法分析

假设有产生式:

*type* → *simple*

| ↑ **id**

| **array** [*simple*] **of** *type*

*simple* → **integer**

| **char**

| **num dot num**

构造递归下降的预测分析程序：

* + - 1. match
      2. type
      3. simple

void match (terminal **t**) {

**if** (lookahead == **t**) lookahead = nextToken( );

**else** error( );

}

void type( ) {

**if** ( (lookahead == **integer**) || (lookahead == **char**) || (lookahead == num) )

simple( );

**else if** ( lookahead == ′↑′ ) { match(′↑′); match(**id**);}

**else if** (lookahead == **array**) {

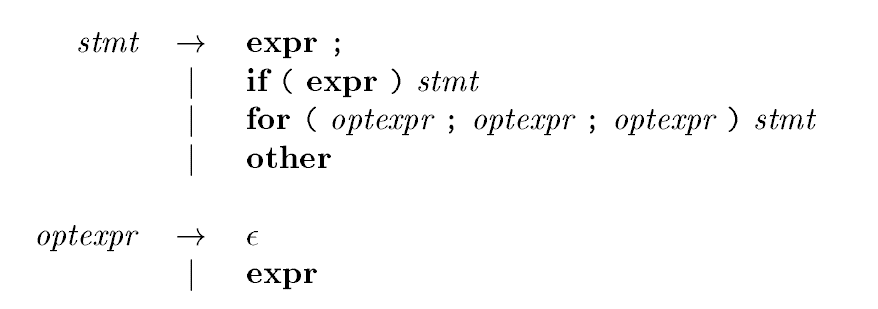
match(**array**); match( ′[′ ); simple( ); match( ′]′ ); match(**of** ); type( );

}

**else** error( );

}

假设有产生式:



针对某个非终结符号 A, 和当前输入符号a, 总是能够选择唯一的那个产生式; 要么出错.

构造递归下降的预测分析程序:

//为每个非终结符（例如A），定义一个函数

回溯法

bool A() {

while (true ) {

1) 选择一个A的产生式 A → X1 X2 … Xk;

2) for (i = 1 to k) {

3) if (Xi是一个非终结符号)

4) output = Xi();

If output == false 2break;

5) else if (Xi等于当前的输入符号a)

6) 读入下一个输入符号

7) else /\* 发生了一个错误 \*/ 2break;

}

}

}

While ( true ) {

}

假设文法存在左递归?

* + 例，下列文法：

*E* → *E* + *T* | *T*

*T* → *T* \* *F* | *F*

*F* → ( *E* ) | **id**

针对某个非终结符号 A, 和当前输入符号a, 总是能够选择唯一的那个产生式; 要么出错.

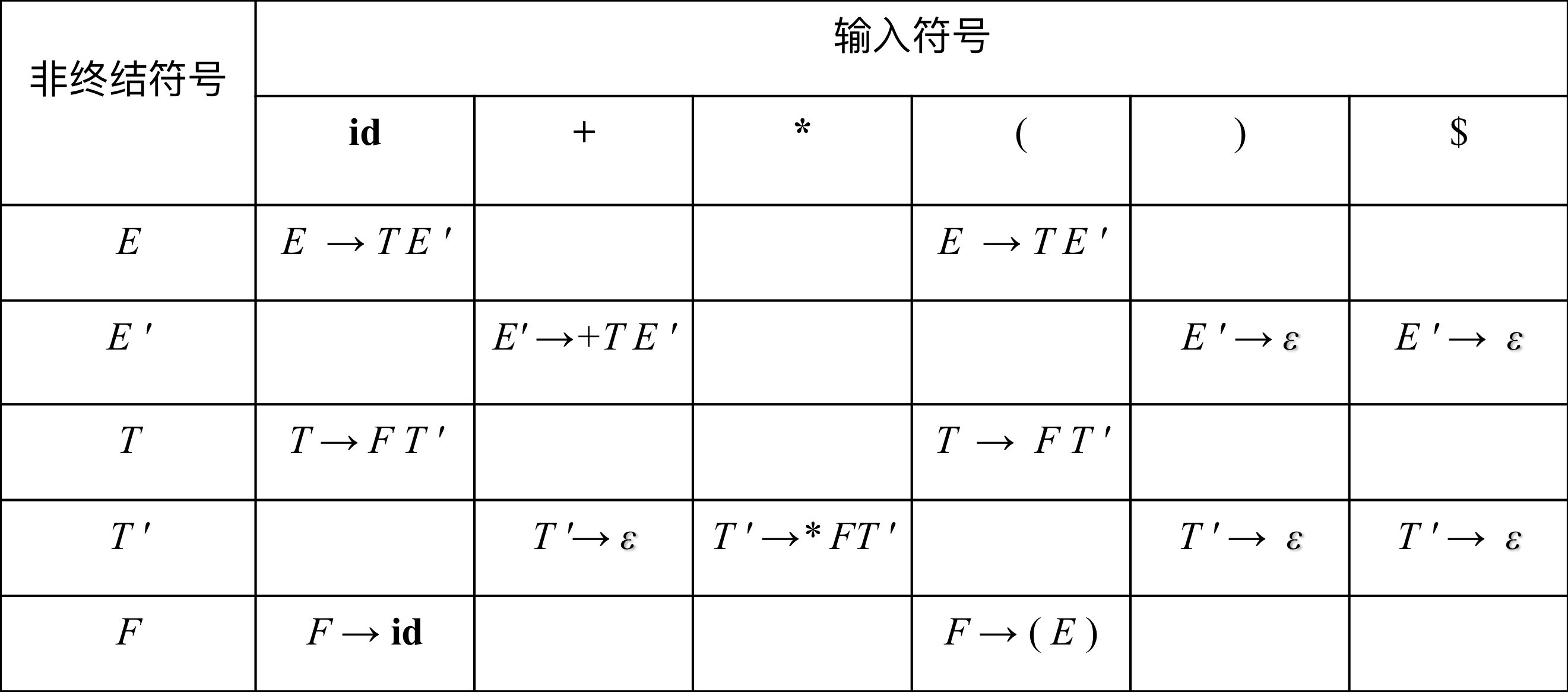
*E → T E'*

*E' → + T E'* | *ε*

*T → F T'*

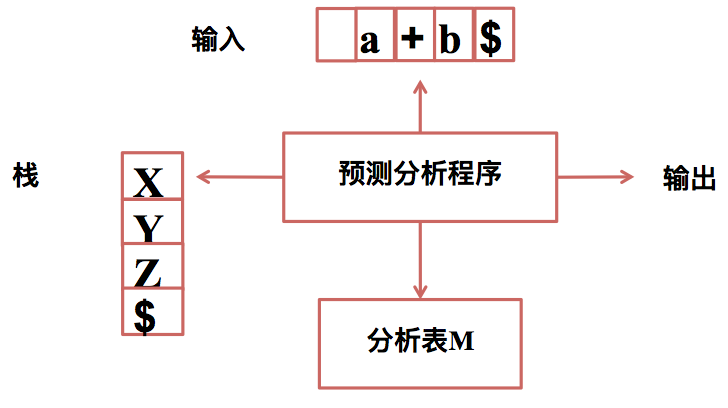
*T' → \* F T'* | *ε*

*F* → (*E* ) | **id**



针对某个非终结符号 A, 和当前输入符号a, 总是能够选择唯一的那个产生式

非递归下降法:



两个数据结构: 栈 和 输入串

栈用来保留？？

输入串: 用一个当前指针用来指示当前待匹配的字符

E → TE'

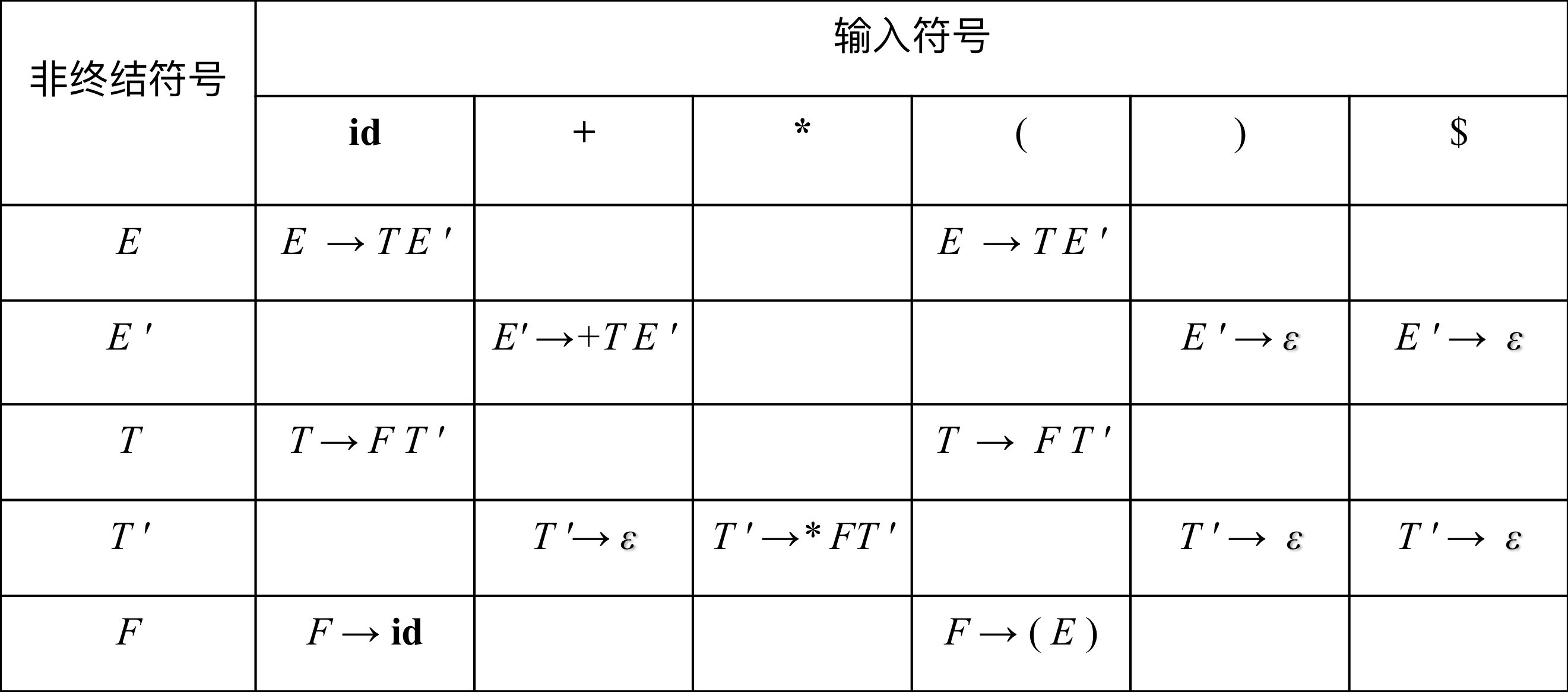
E' → +TE' | ε

T → F T'

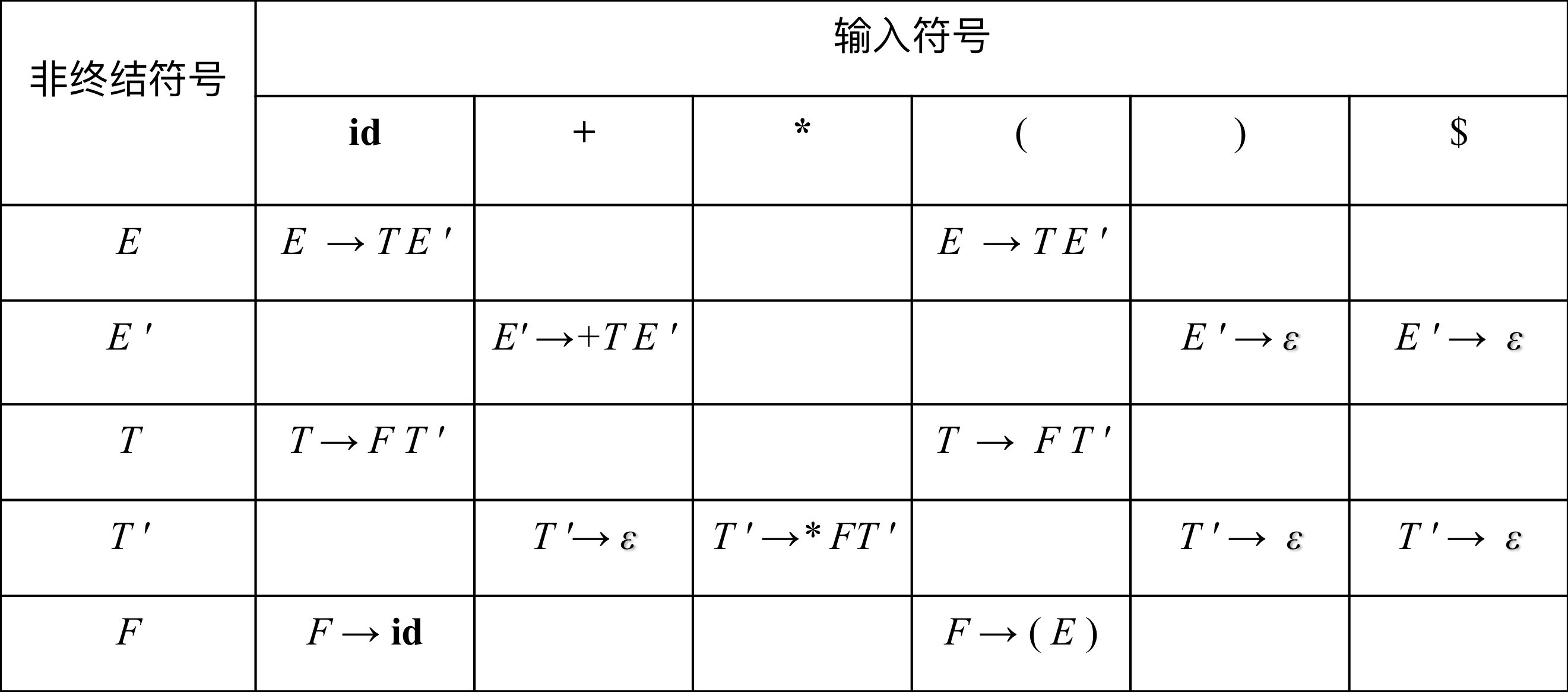
T' → \* F T' | ε

F → (E) | **id**

为输入串 id + id \* id $ 自顶向下构造分析树 FIRST FOLLOW



用栈来模拟最左推导的过程



|  |  |  |
| --- | --- | --- |
| 已匹配 | 栈 | 输入 |
|  | E $ | id + id \* id $ |
|  | TE' $ | id + id \* id $ |
|  | FT'E'$ | id + id \* id $ |
|  | idT'E'$ | id + id \* id $ |
| id | T'E'$ | + id \* id $ |
| id | E'$ | + id \* id $ |
| id | +TE'$ | + id \* id $ |
| id + | TE'$ | id \* id $ |
| id + | FT' E'$ | id \* id $ |
| id+ | id T' E' $ | id \* id $ |
| id + id | T' E' $ | \* id $ |