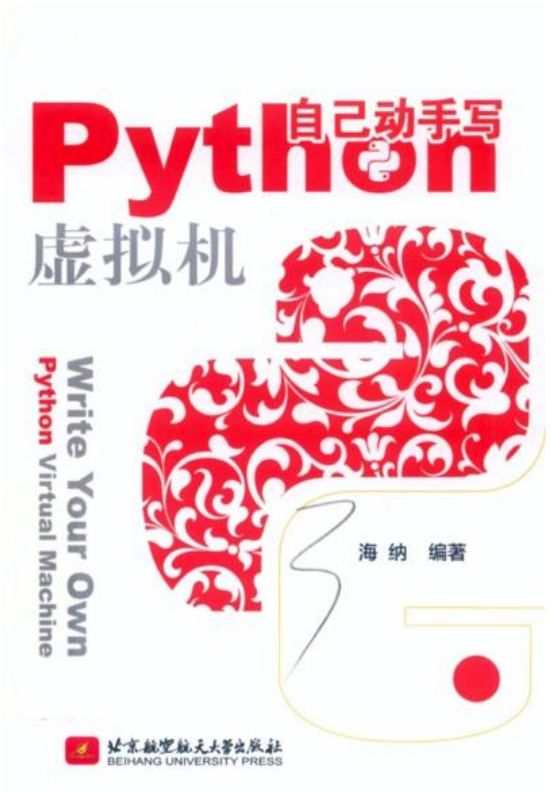
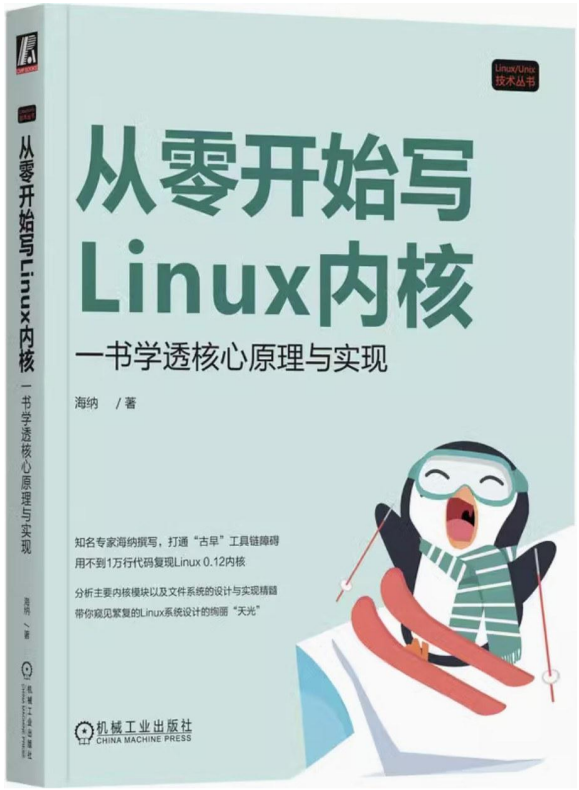
一段代码的神奇之旅

摩尔线程

海纳

个人简介





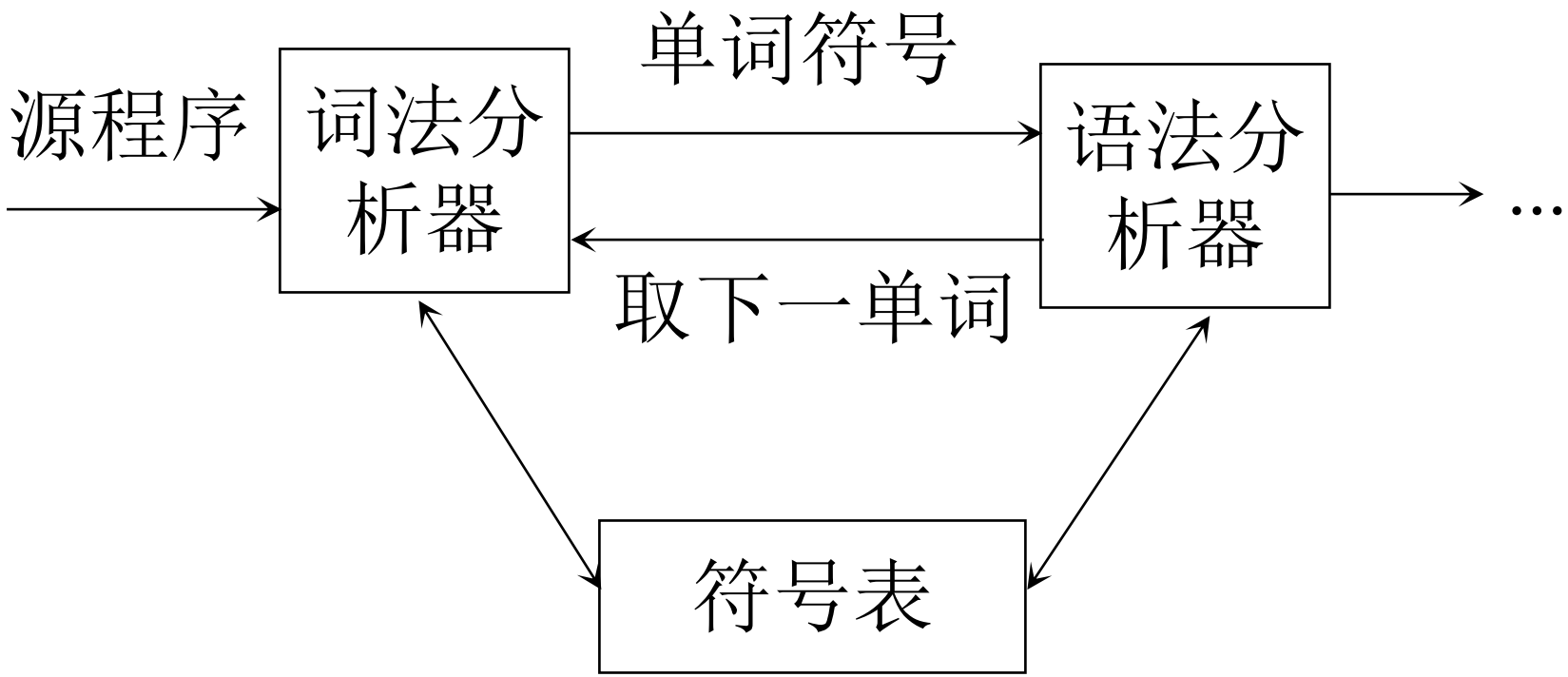
海纳
浙江 杭州

扫一扫上面的二维码图案，加我为朋友。

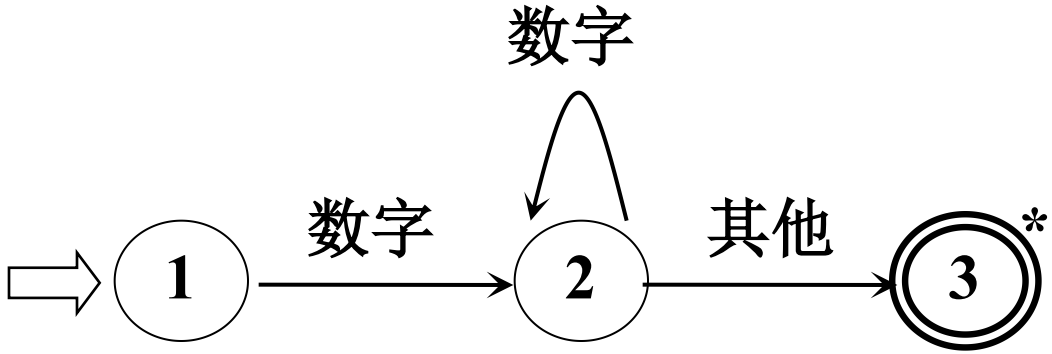
目录

- 词法分析
- 文法分析：从顶向下的分析方法
- 生成字节码
- 虚拟机执行

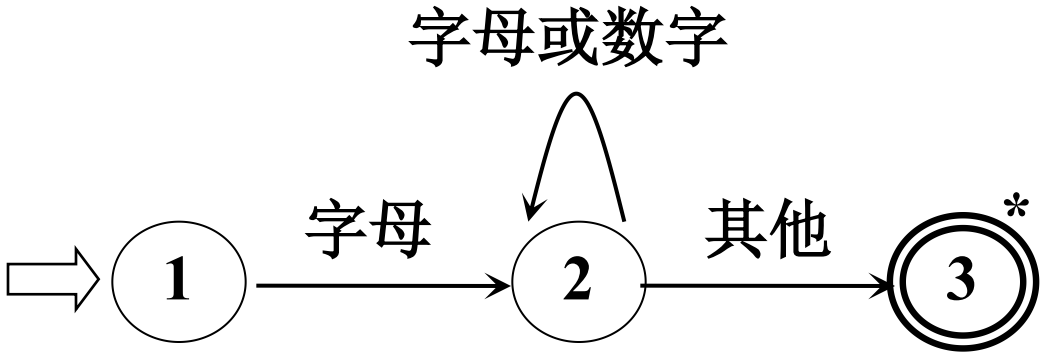
词法分析的基本结构



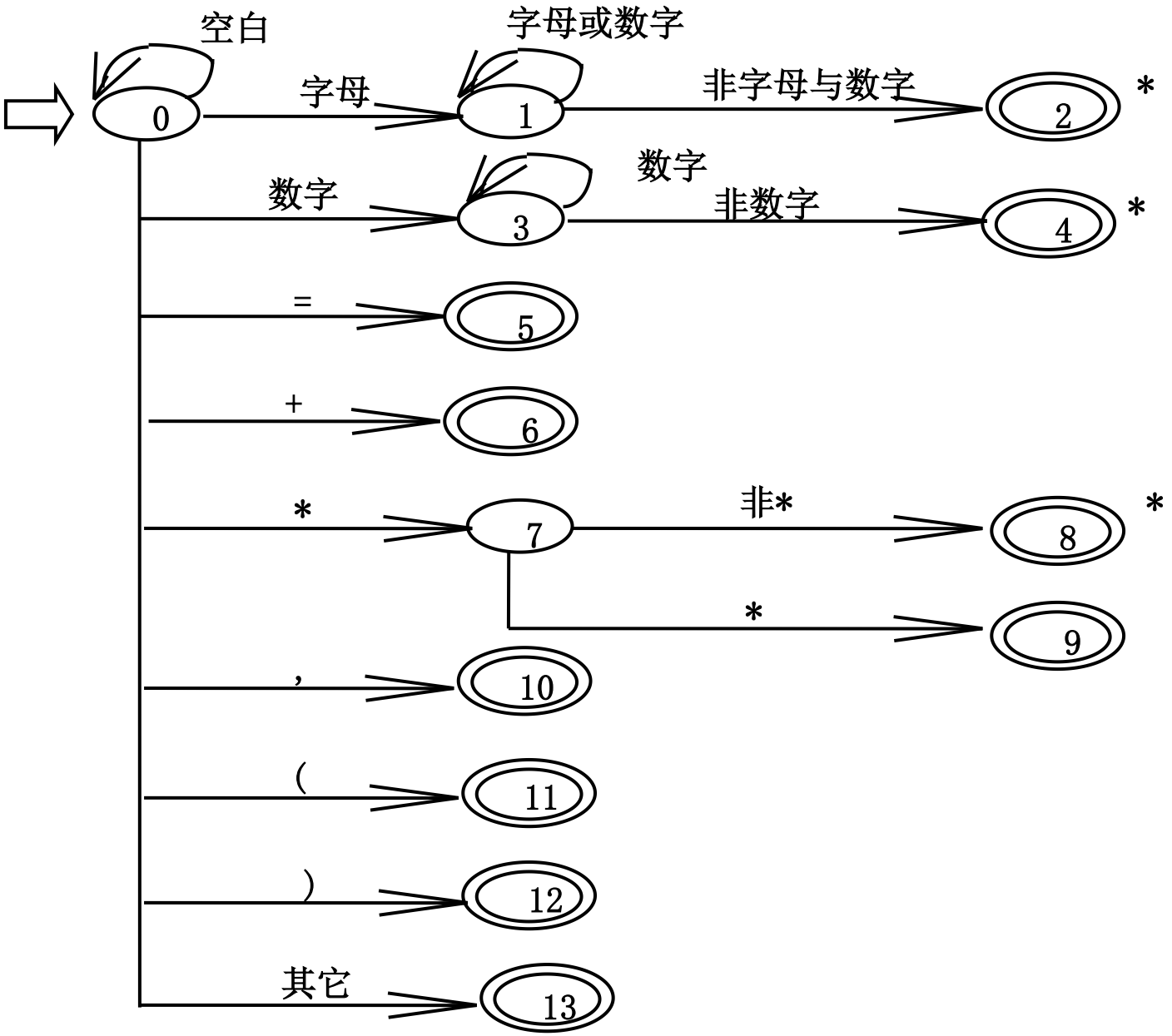
- 一个状态转换图可用于识别(或接受)一定的字符串。



识别整常数的状态转换图



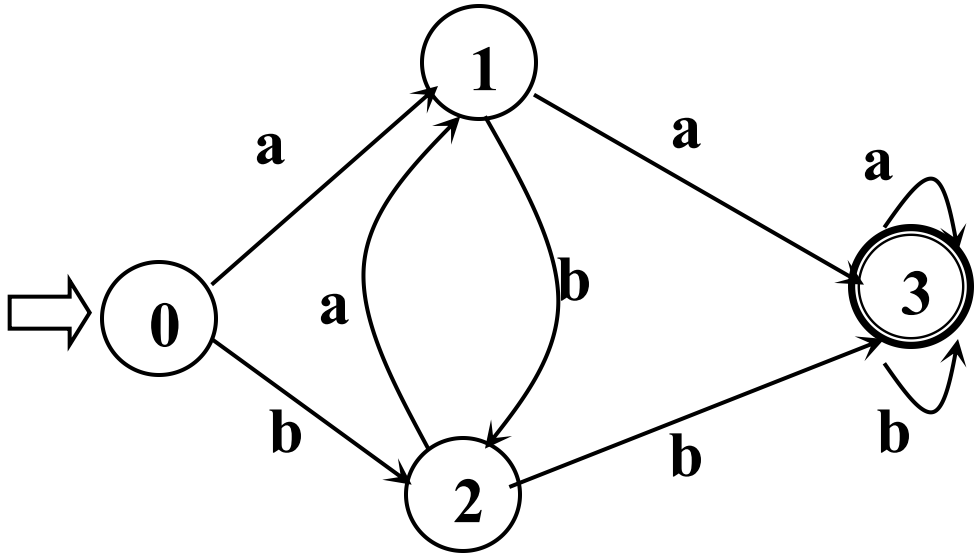
识别标识符的状态转换图



- 例如：DFA M={{0，1，2，3}，{a，b}，f，0，{3}}，其中：f定义如下：
  - f(0，a)=1　　f(0，b)=2
  - f(1，a)=3　　f(1，b)=2
  - f(2，a)=1　　f(2，b)=3
  - f(3，a)=3　　f(3，b)=3

	a	b
0	1	2
1	3	2
2	1	3
3	3	3

状态转换矩阵



状态转换图

文法分析

自顶向下的递归下降分析法

- E -> T (‘+’ | ’-’ T) *
- T -> F (‘*’ | ‘/’ T) *
- F -> num | var | ‘(‘ E ‘)’

- 基本规则：
  - 非终结符实现为函数
  - 或操作（丨）实现为 if 语句
  - 包含一个或者多个（＋＊）实现为 while 语句

思考

- 为什么不能有左递归？
- stmt -> if_stmt | while_stmt | for_stmt 如何决定应该使用哪个规则进行推导呢？
- assign -> var ‘=’ expr

  call_stmt -> var ‘(‘ args ‘)’

  该如何进行区分呢？

G(E)： $E \rightarrow i \mid E+E \mid E-E \mid E*E \mid E/E \mid (E)$

i*i+i

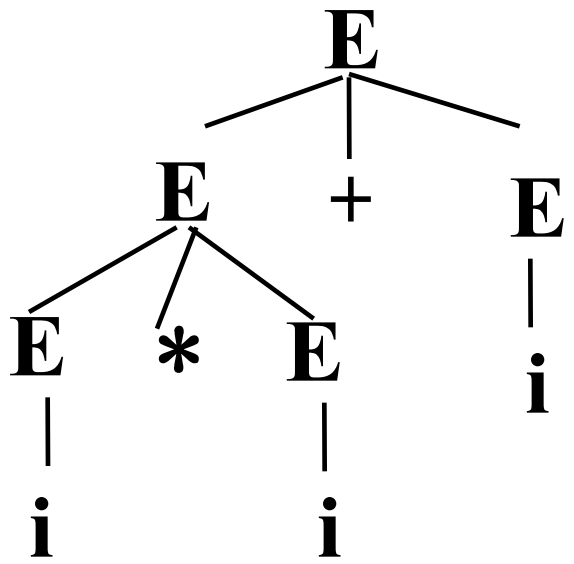
E*i+i

E*E+i

E+i

E+E

E

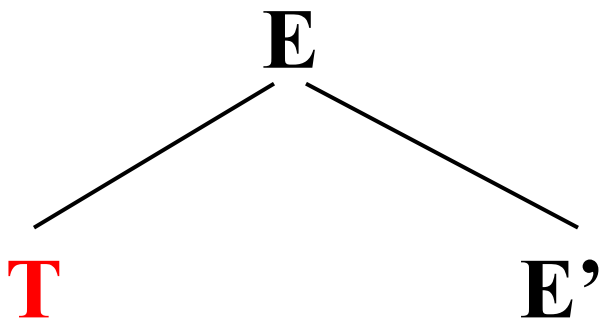


E

i + i

IP

- G(E):
  - E→TE′
  - E′→+TE′ | ε
  - T→FT′
  - T′→*FT′ | ε
  - F→(E) | i

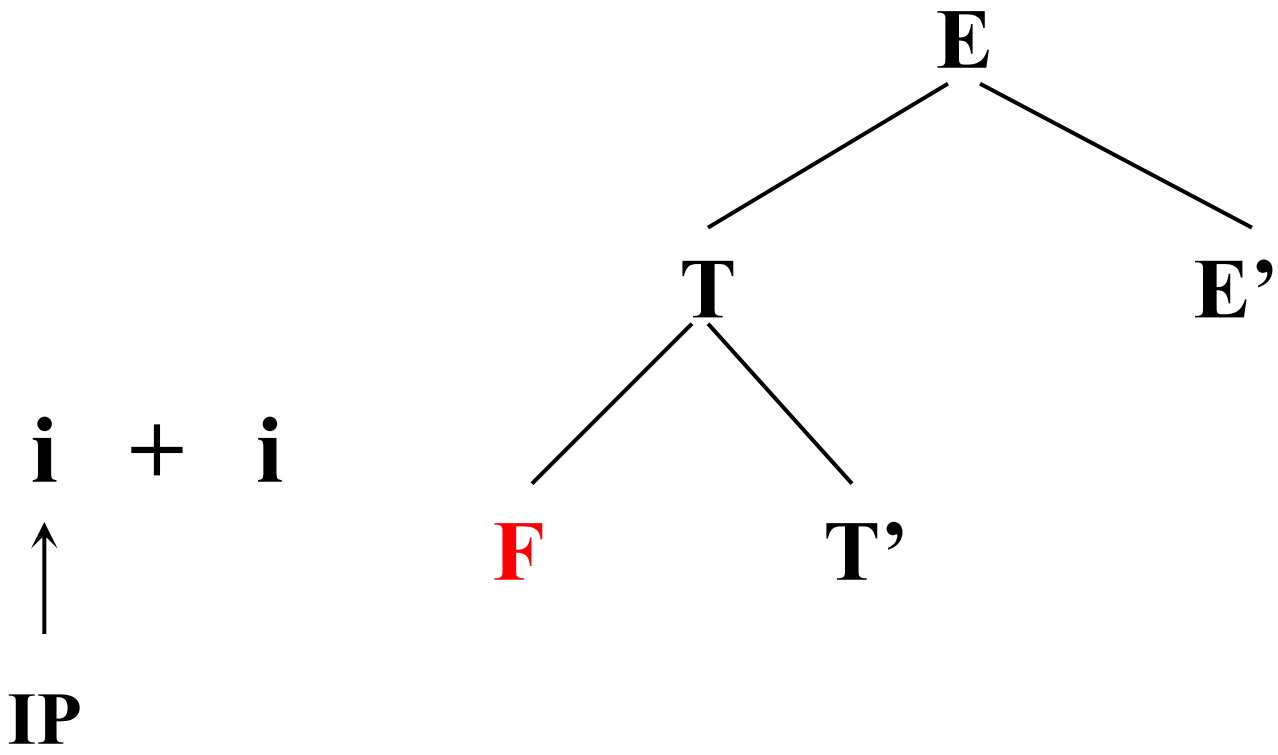


i + i

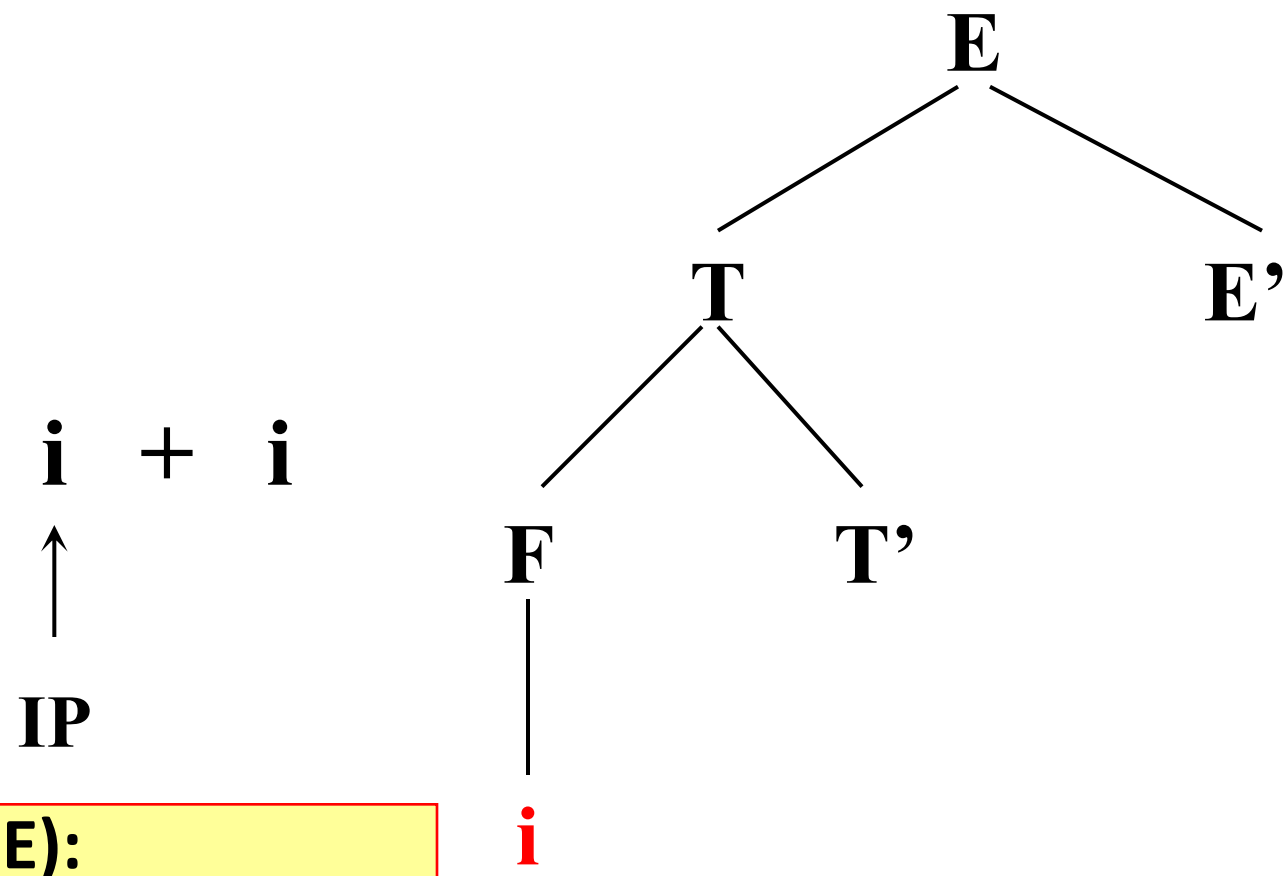
↑

IP

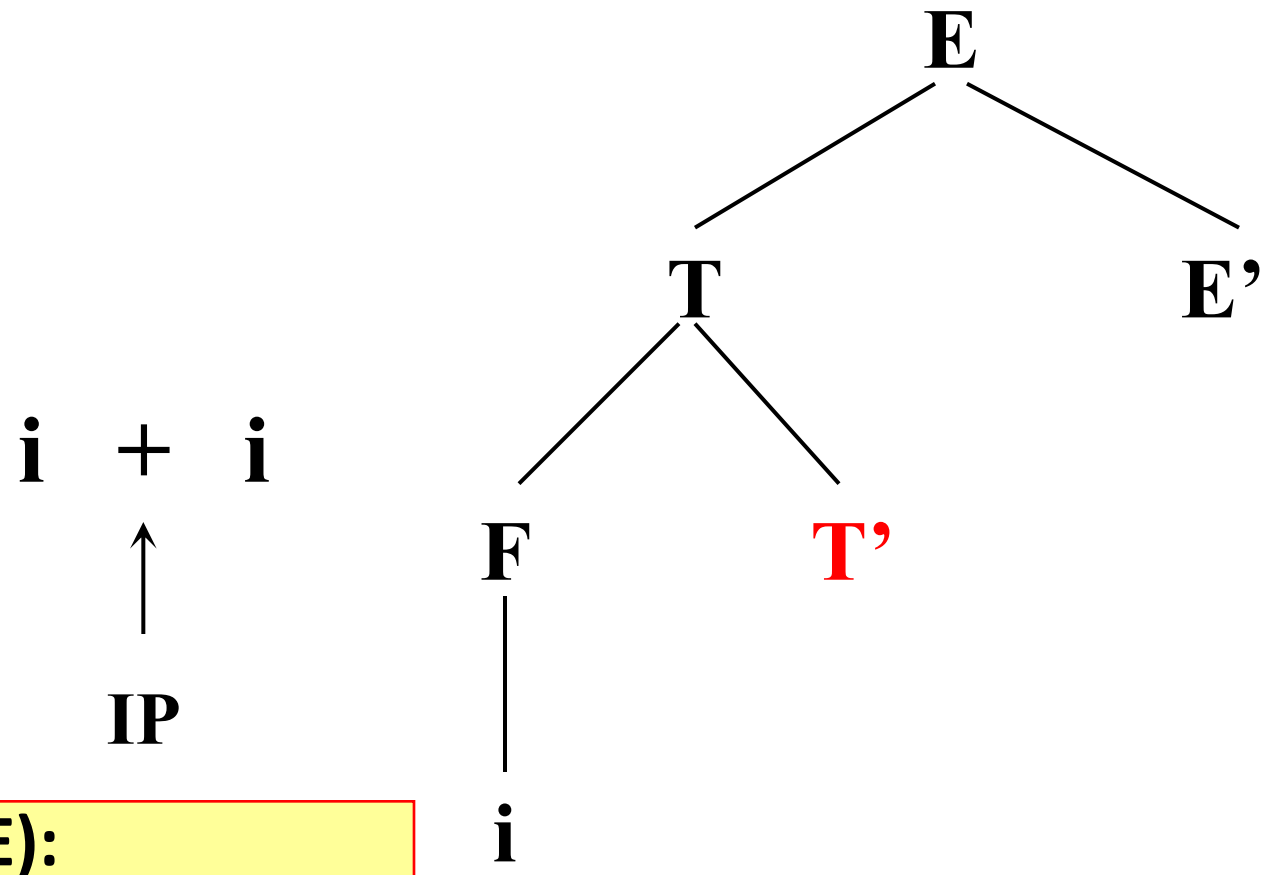
- G(E):
  - E→TE′
  - E′→+TE′ | ε
  - T→FT′
  - T′→*FT′ | ε
  - F→(E) | i



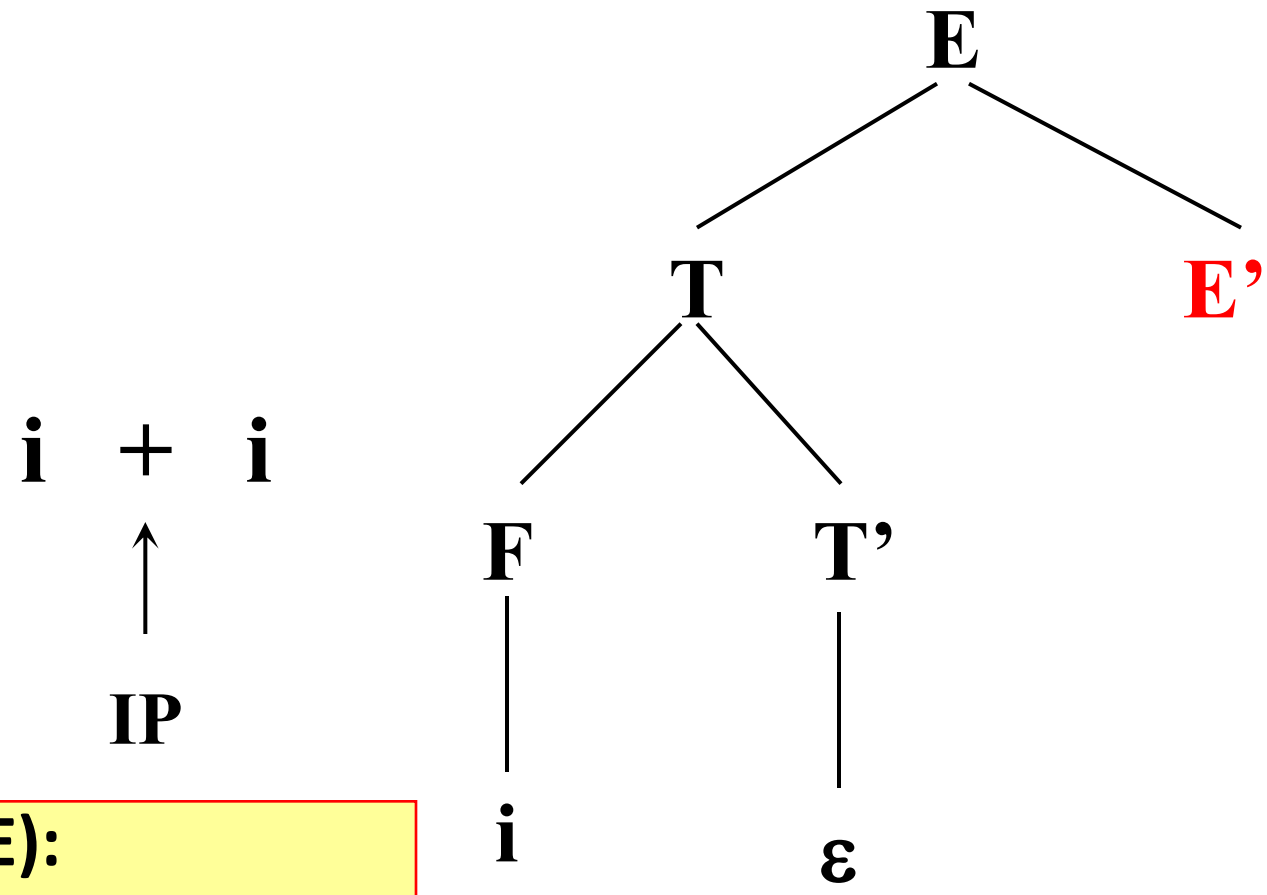
- G(E):
  - E→TE′
  - E′→+TE′ | ε
  - T→FT′
  - T′→*FT′ | ε
  - F→(E) | i



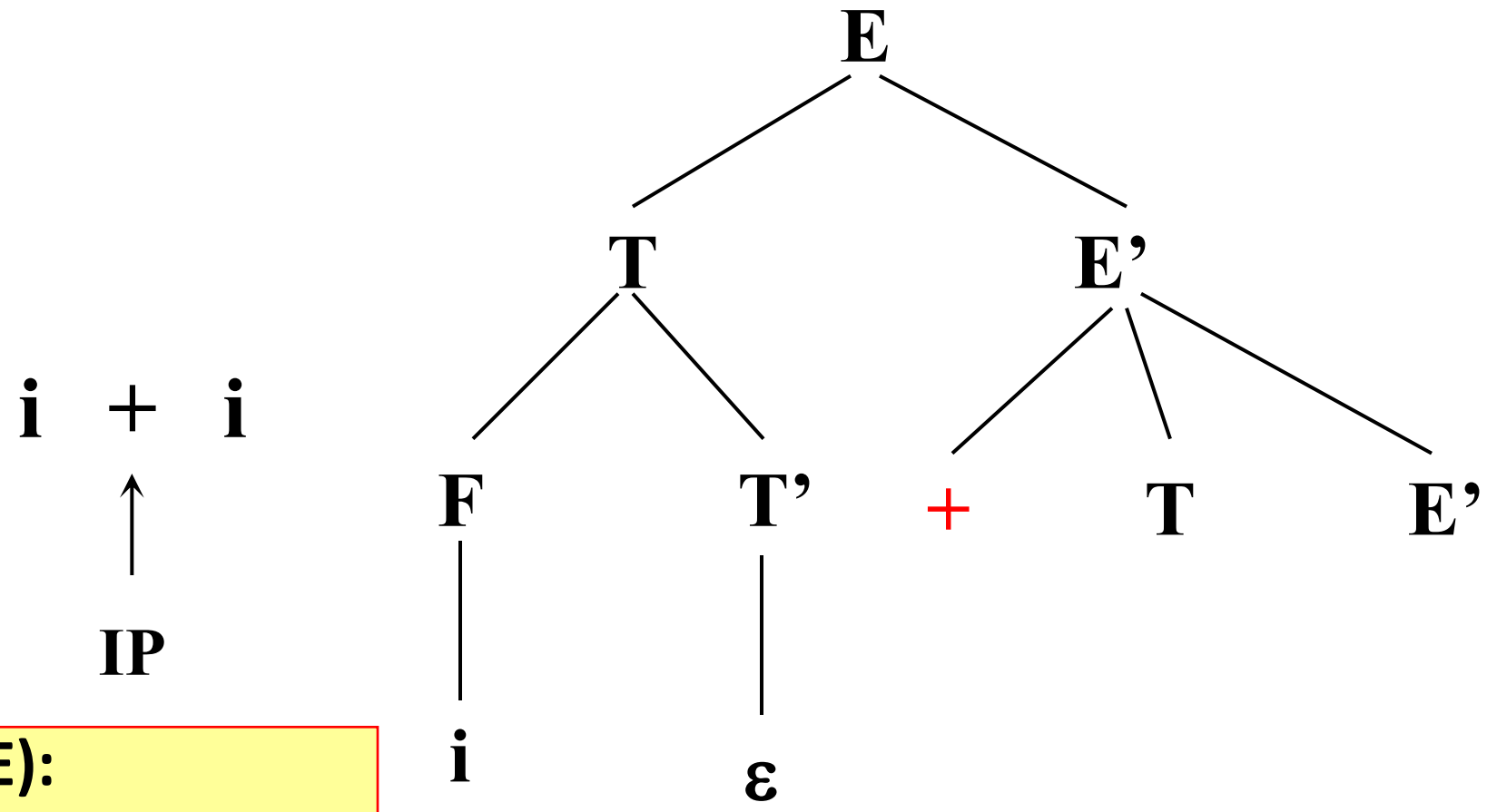
- G(E):
  - E→TE′
  - E′→+TE′ | ε
  - T→FT′
  - T′→*FT′ | ε
  - F→(E) | i



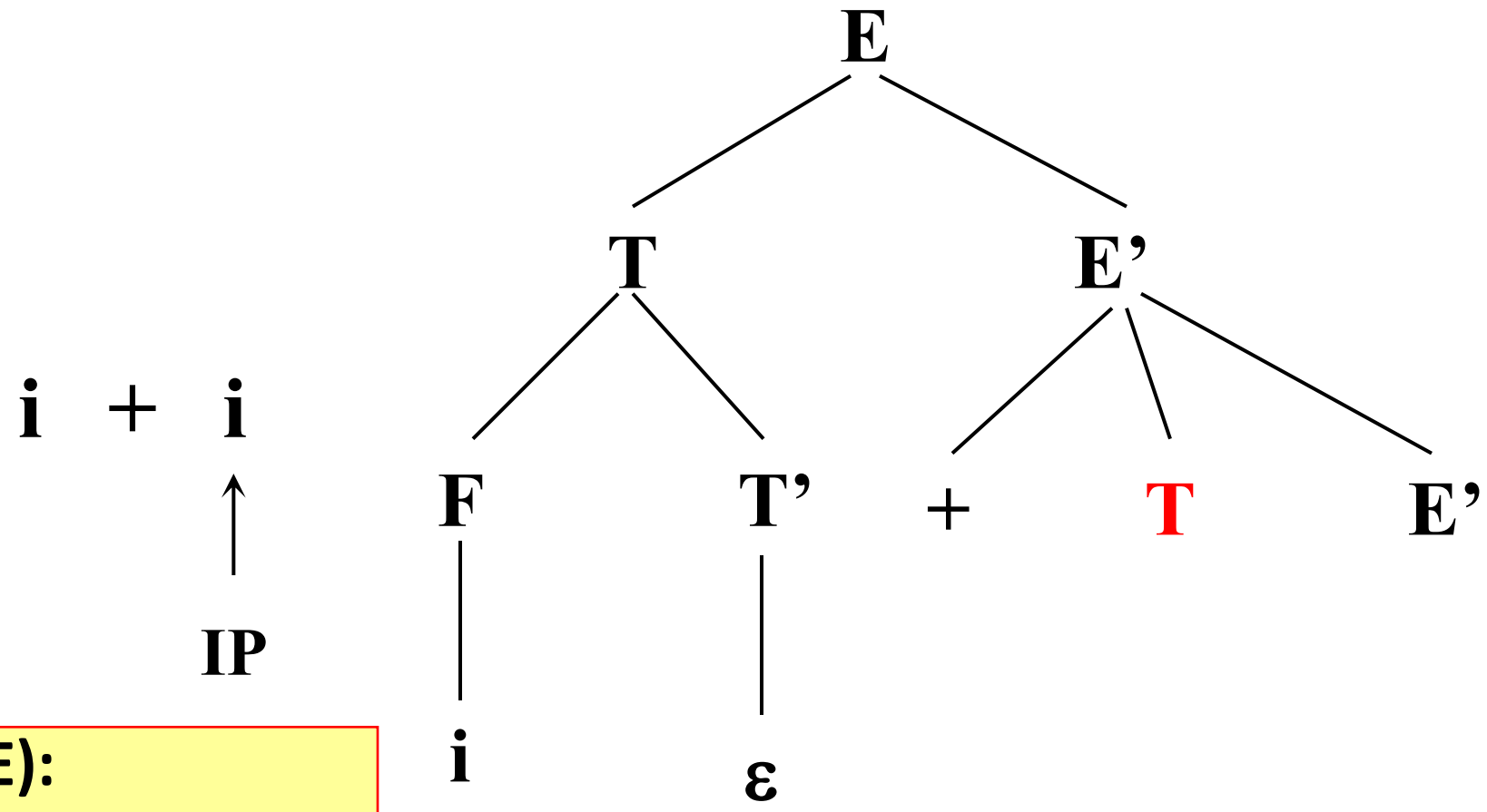
- G(E):
  - E→TE′
  - E′→+TE′ | ε
  - T→FT′
  - T′→*FT′ | ε
  - F→(E) | i



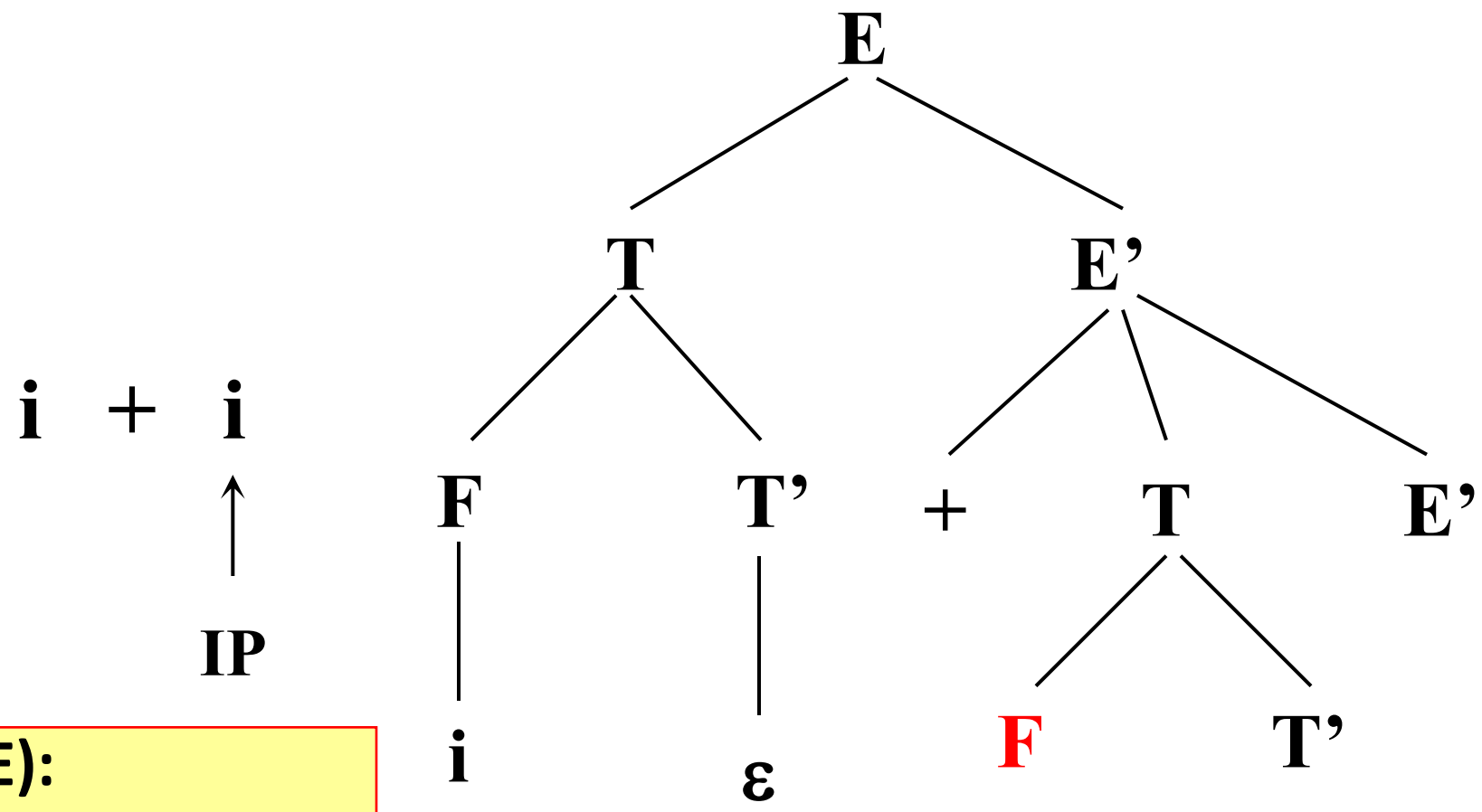
- G(E):
  - E→TE′
  - E′→+TE′ | ε
  - T→FT′
  - T′→*FT′ | ε
  - F→(E) | i



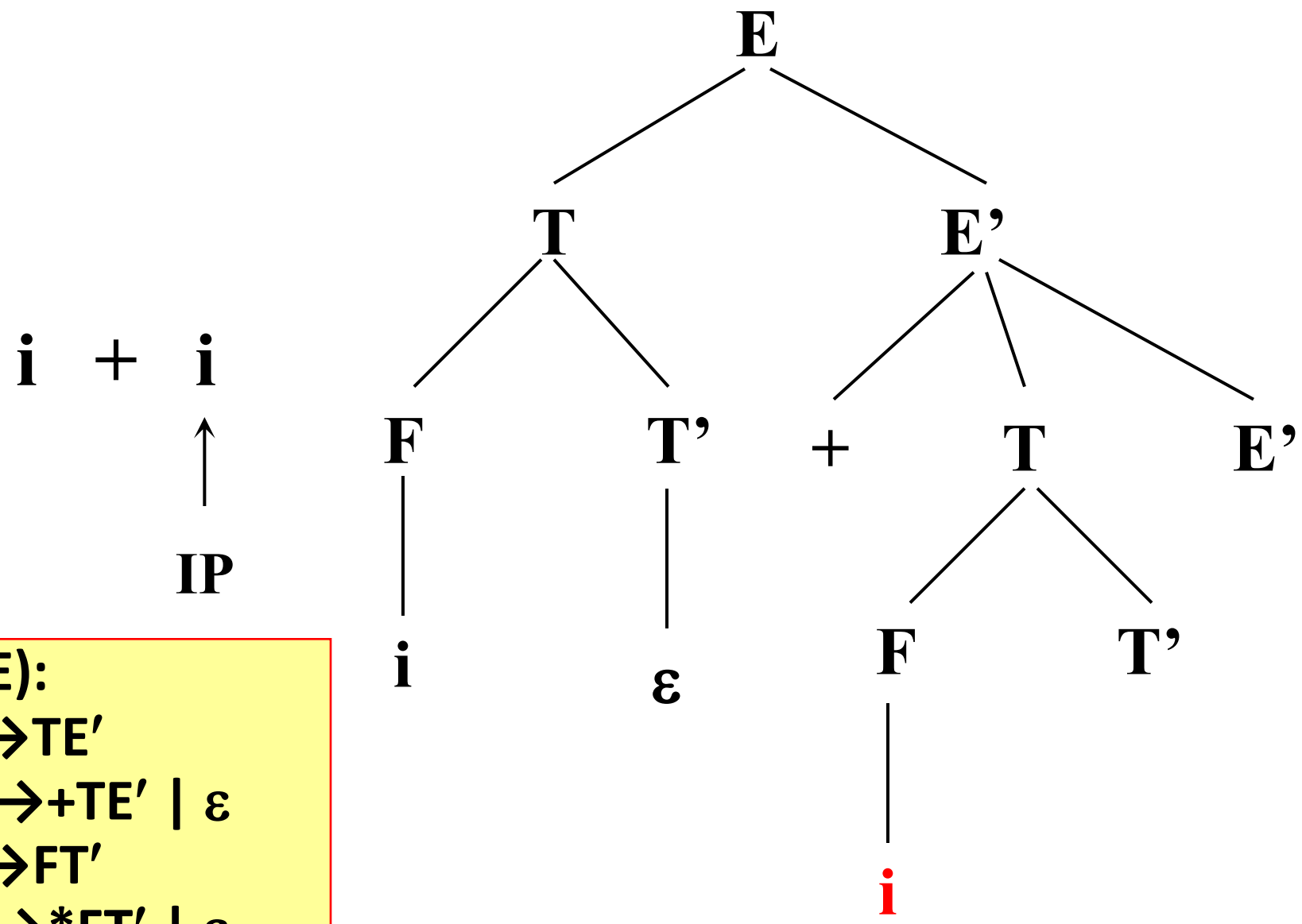
- G(E):
  - E→TE′
  - E′→+TE′ | ε
  - T→FT′
  - T′→*FT′ | ε
  - F→(E) | i



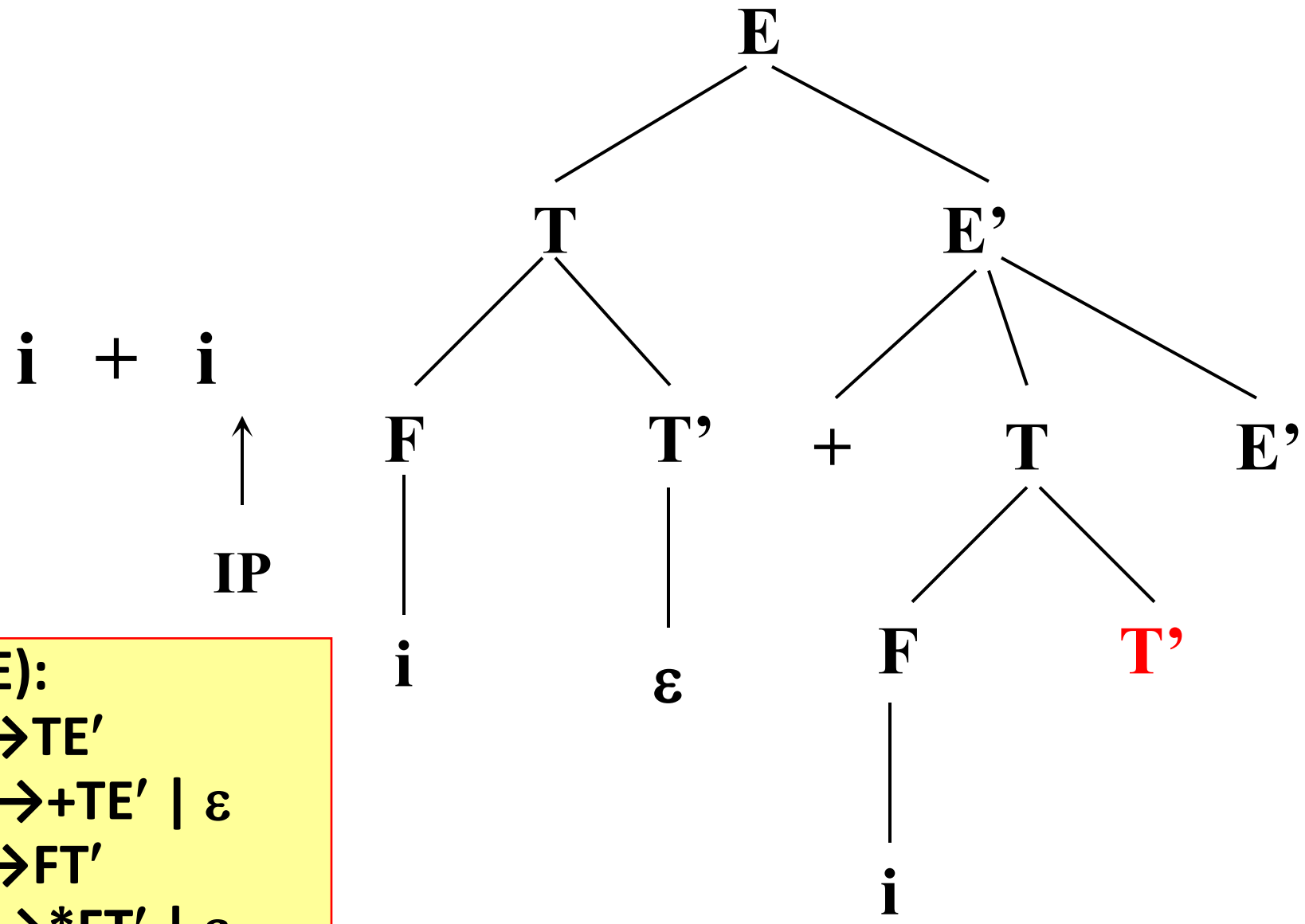
- G(E):
  - E→TE′
  - E′→+TE′ | ε
  - T→FT′
  - T′→*FT′ | ε
  - F→(E) | i



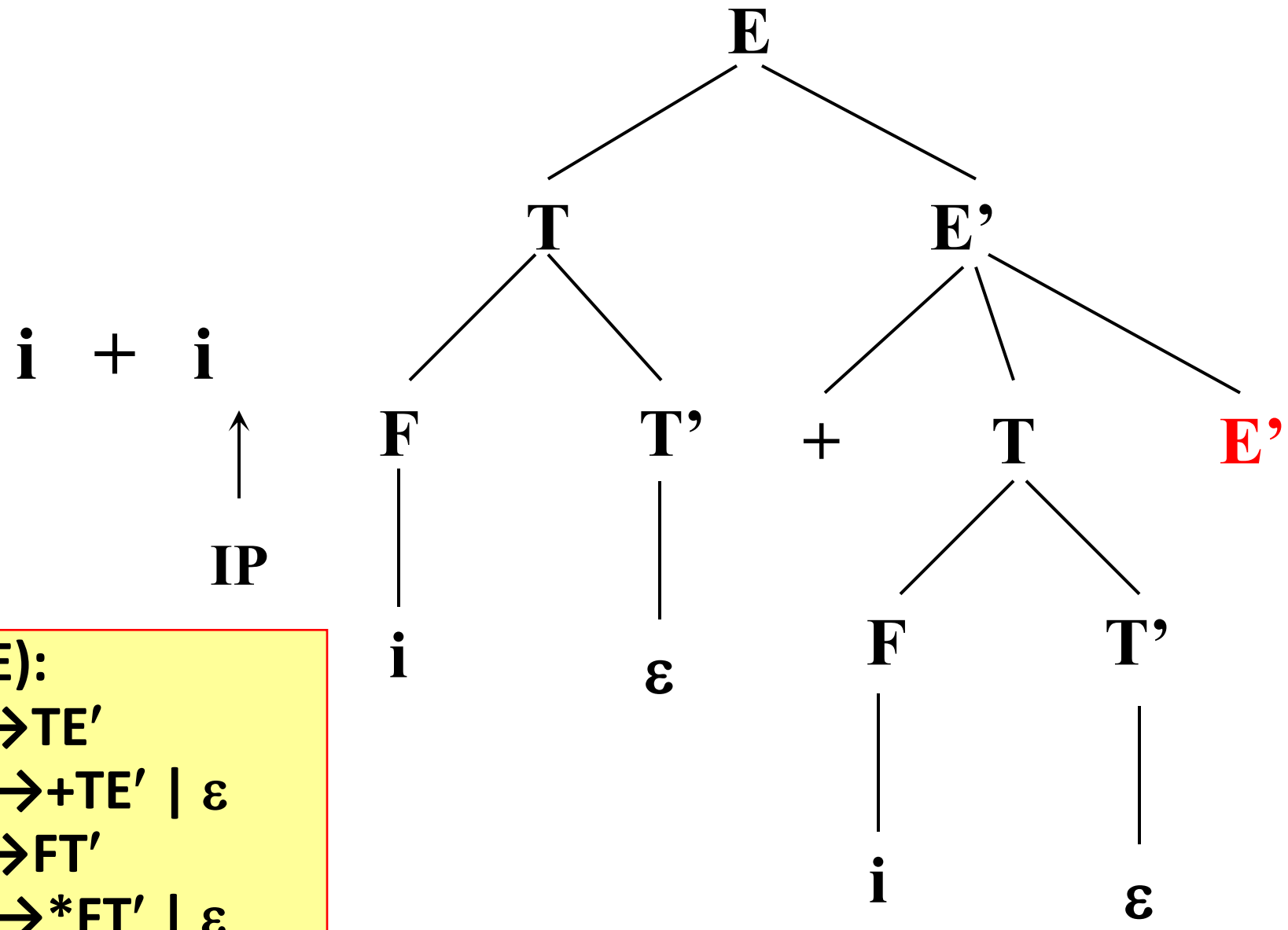
- G(E):
  - E→TE′
  - E′→+TE′ | ε
  - T→FT′
  - T′→*FT′ | ε
  - F→(E) | i



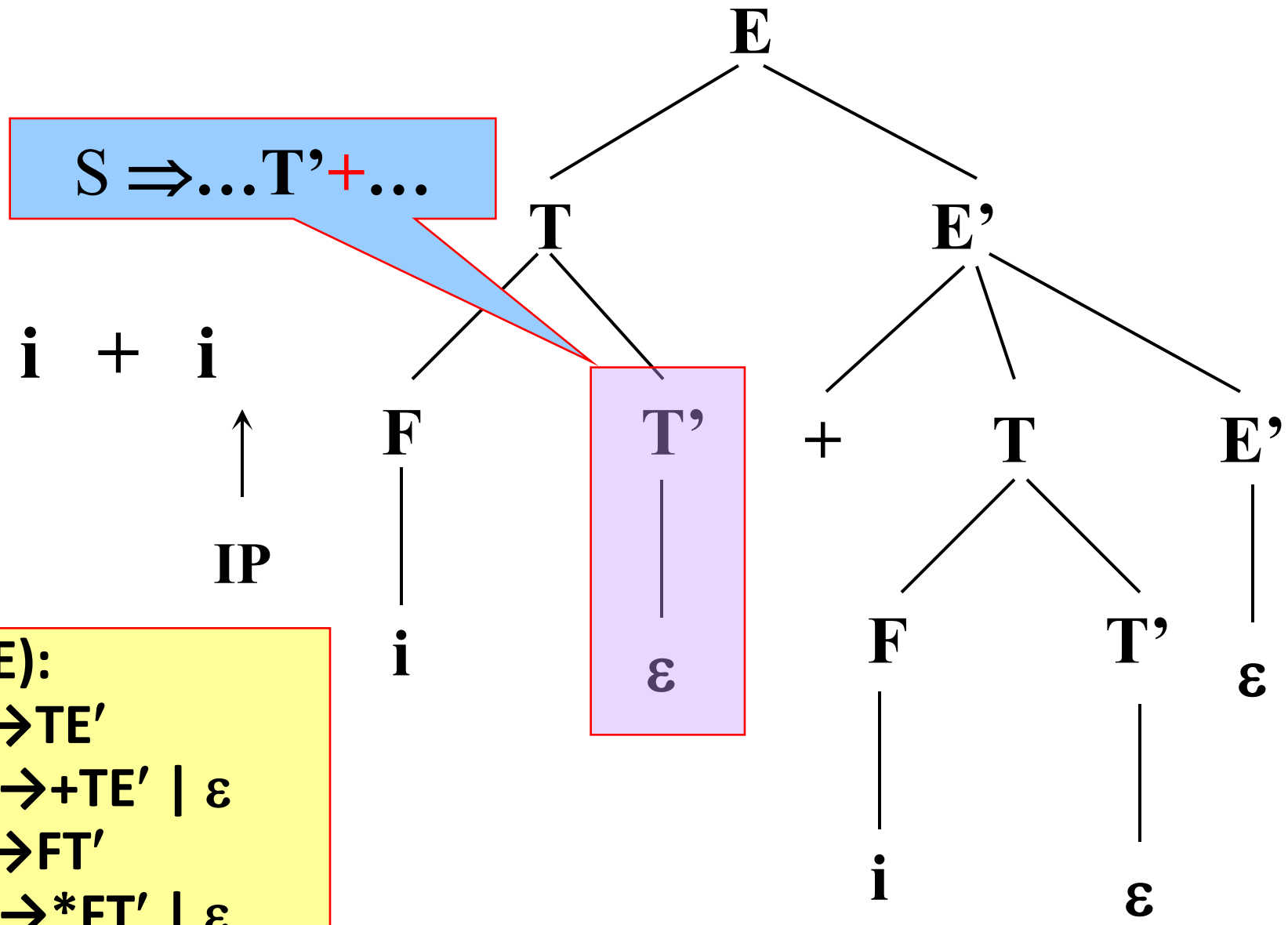
- G(E):
  - E→TE′
  - E′→+TE′ | ε
  - T→FT′
  - T′→*FT′ | ε
  - F→(E) | i



- G(E):
  - E→TE′
  - E′→+TE′ | ε
  - T→FT′
  - T′→*FT′ | ε
  - F→(E) | i



- G(E):
  - E→TE′
  - E′→+TE′ | ε
  - T→FT′
  - T′→*FT′ | ε
  - F→(E) | i



- G(E):
  - E→TE′
  - E′→+TE′ | ε
  - T→FT′
  - T′→*FT′ | ε
  - F→(E) | i

Follow集合的意义

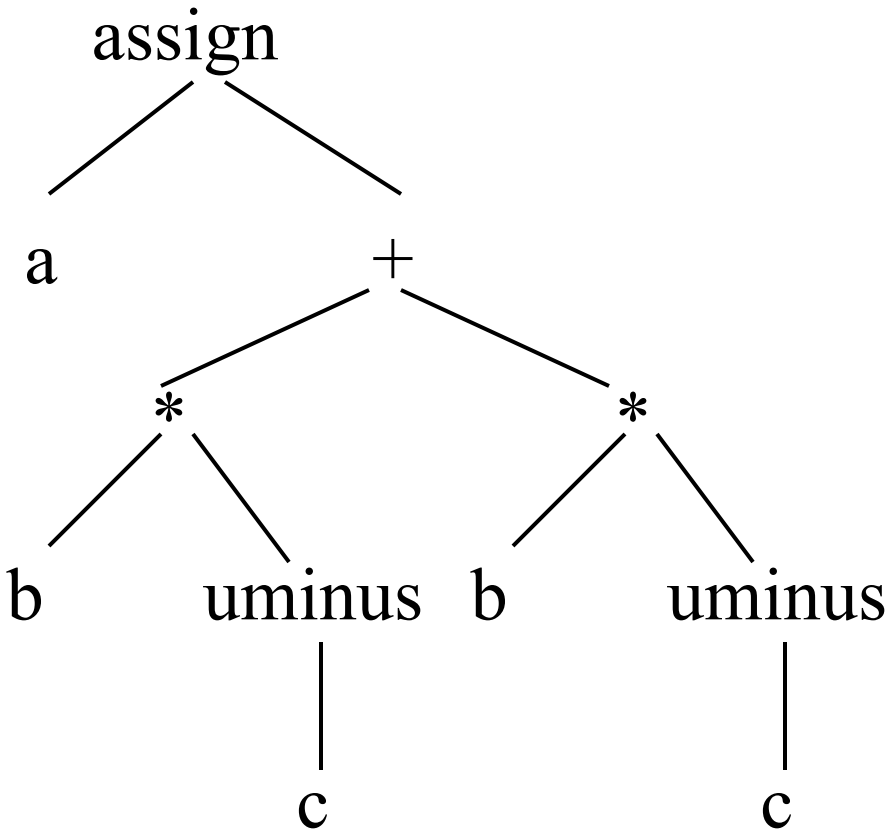
- 假定S是文法G的开始符号，对于G的任何非终结符A，我们定义

$$FOLLOW(A) = \{a \mid S \overset{*}{\Rightarrow} \ldots Aa \ldots, a \in V_T\}$$

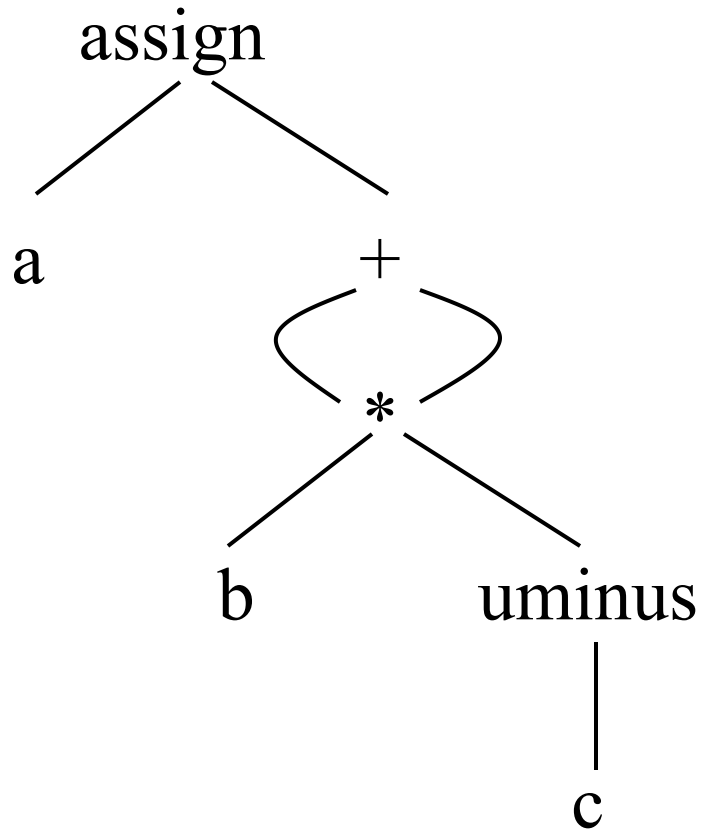
特别是，若$S \overset{*}{\Rightarrow} \cdots A$，则规定

#∈FOLLOW(A)

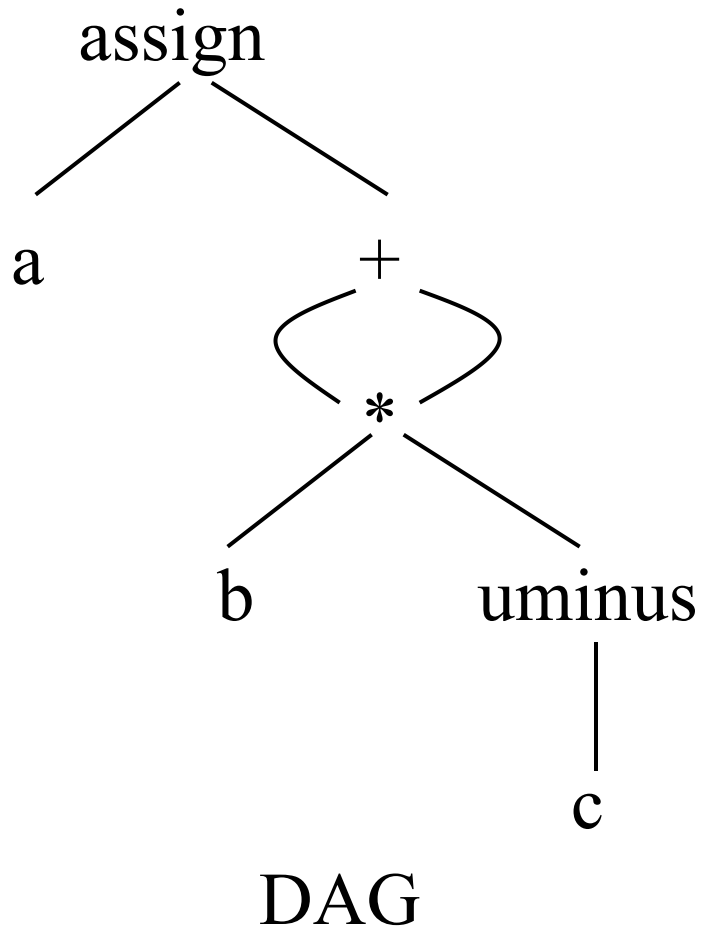
a=b*(-c)+b*(-c)的图表示法



抽象语法树



DAG



DAG

抽象语法树对应的代码：

T_1:=-c

T_2:=b*T_1

T_3:=-c

T_4:=b*T_3

T_5:=T_2+T_4

a:=T_5

DAG对应的代码：

T_1:=-c

T_2:=b*T_1

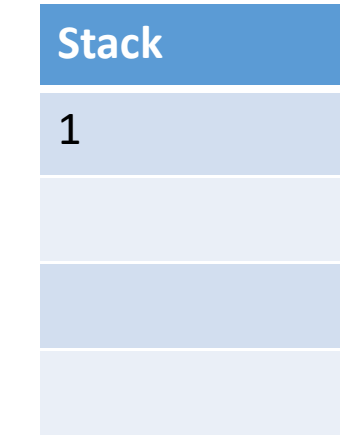
T_5:=T_2+T_2

a:=T_5

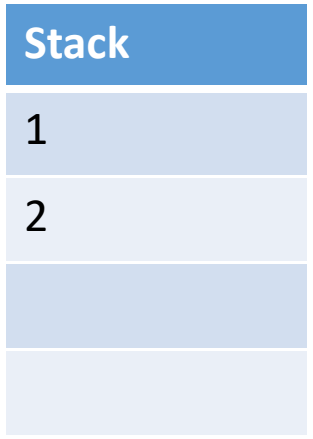
字节码文件

基于栈的虚拟机

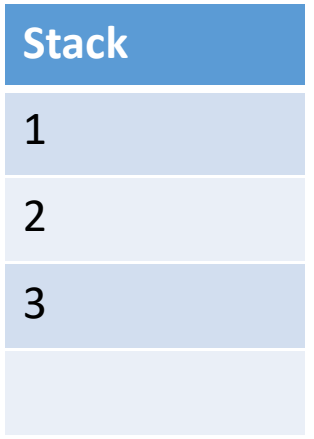
LOAD_CONST 1
LOAD_CONST 2
LOAD_CONST 3
BINARY_MULTIPLY
BINARY_ADD



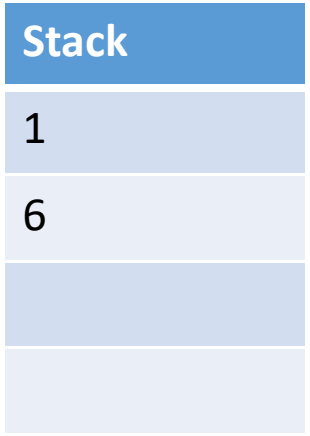
LOAD_CONST 1



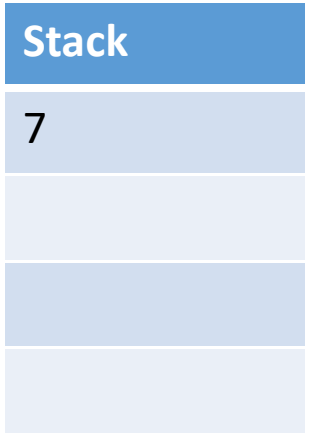
LOAD_CONST 2



LOAD_CONST 3

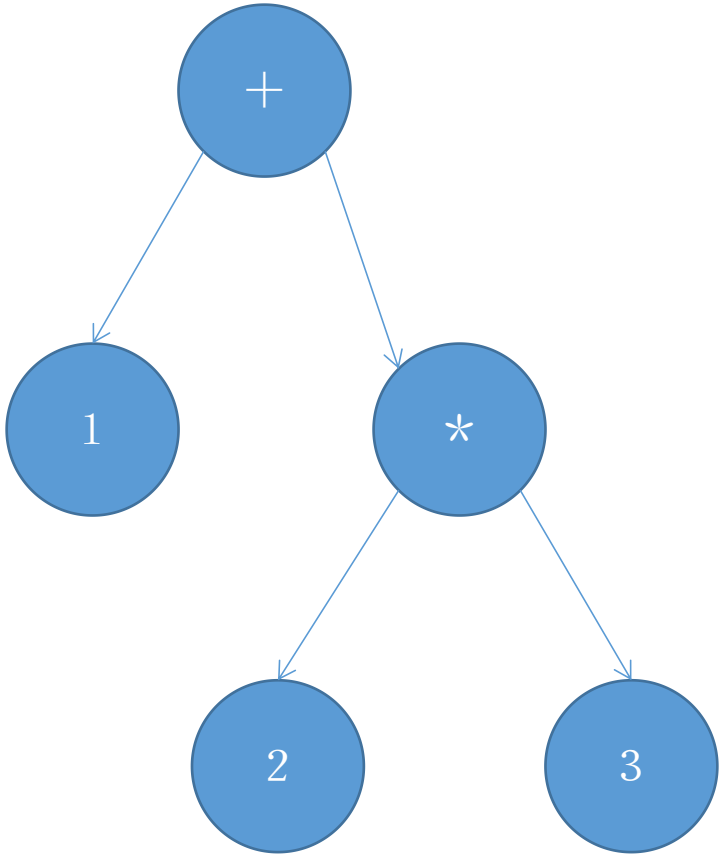


BINARY_MULTIPLY



BINARY_ADD

CodeGen：从 AST 到 ByteCode



后序遍历

```
LOAD_CONST 1
LOAD_CONST 2
LOAD_CONST 3
MUL
ADD
```

CodeGen : 从 AST 到 ByteCode