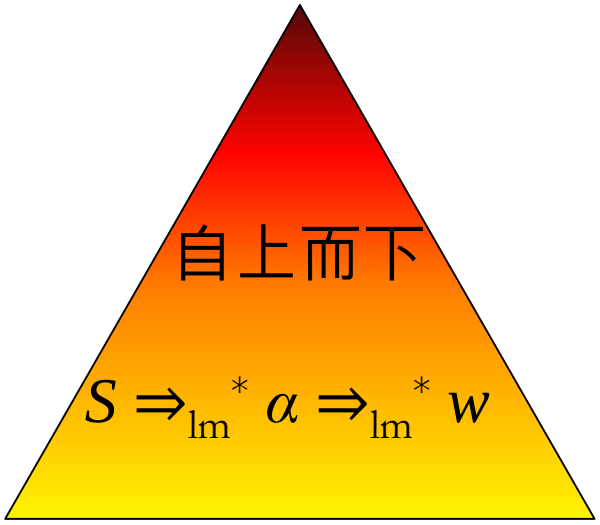
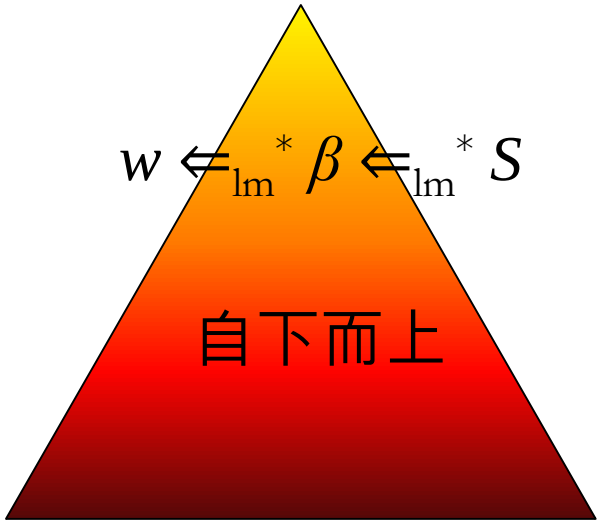
第六章 自上而下语法分析

— 2025

语法分析概貌

- 语法分析本质上是基于文法识别终结符串w是否为句子的一种确定性计算过程，其中，每个句子的语法树都唯一。
  - 对合法句子w有一个确定的最左推导过程$S \Rightarrow_{lm}^{*} \alpha \Rightarrow_{lm}^{*} w$;
  - 确定性地从S为根开始逐步扩展为一颗产物为w的语法树;
  - 对合法句子w有一个确定的最左推导过程$w \Leftarrow_{lm}^{*} \beta \Leftarrow_{lm}^{*} S$;
  - 确定性地以w中每个符号为一个叶子，逐步，构建为一颗产物为w根为S的语法树。





自上而下的语法分析

- 输入串w是记号串，是对某个源程序进行词法分析的结果。
  - 寻找到一个最左推导$S \Rightarrow_{lm}^{*} \alpha \Rightarrow_{lm}^{*} w$；或者，
  - 以S为根，从根开始自上而下地为w扩展出一颗语法树。
- 语法分析中逐次读入输入符号（可以是调用词法分析器返回单词），然后选择合适的产生式或者用于形成一次直接推导，或者用于扩展当前语法树的一个叶子节点。
- 这个过程一直进行到输入串w被读完（消耗完），同时w的推导完成成功，或者w的语法树扩展成功，则说w是合法句子，语法分析结束。
- 否则报出语法错误，作为拒绝的理由。

- 思考：如何寻找到一个最左推导$S \Rightarrow_{lm}^{*} \alpha \Rightarrow_{lm}^{*} w$？

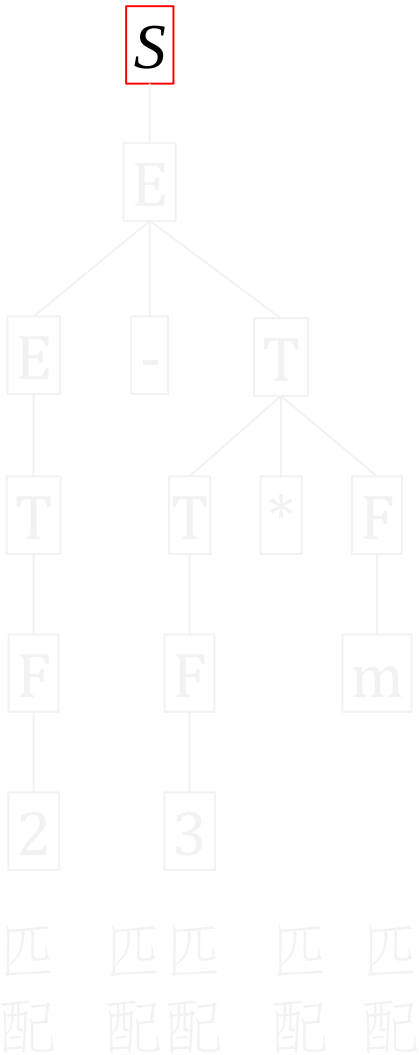
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

$S \longrightarrow E$

$E \longrightarrow E + T$

$E \longrightarrow E - T$

$E \longrightarrow T$

$T \longrightarrow T * F$

$T \longrightarrow T / F$

$T \longrightarrow F$

$F \longrightarrow (E)$

$F \longrightarrow i$

$F \longrightarrow d$

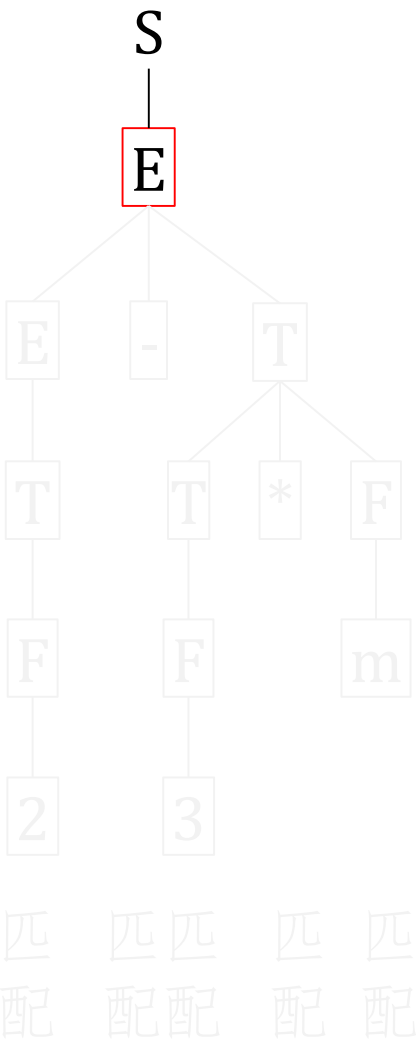
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

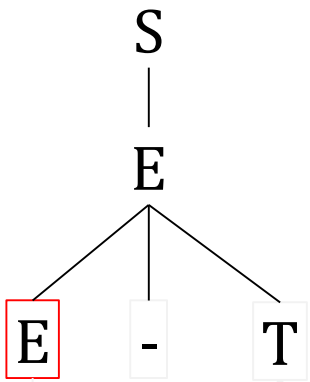
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

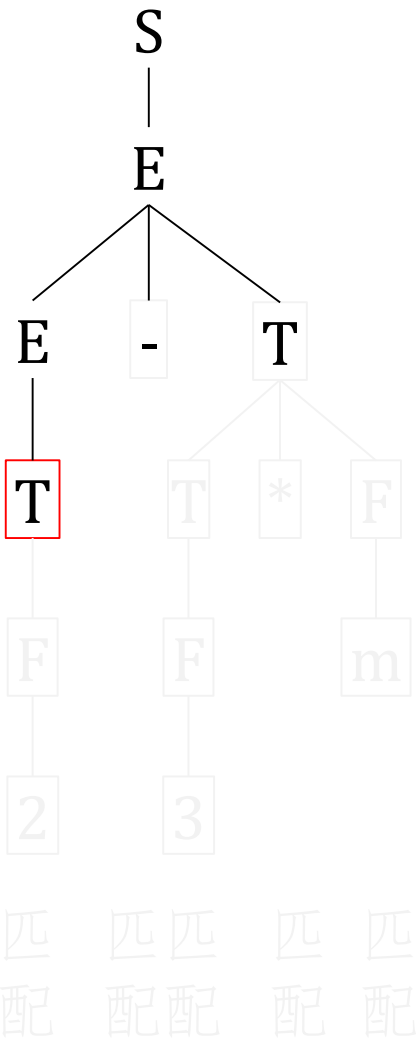
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

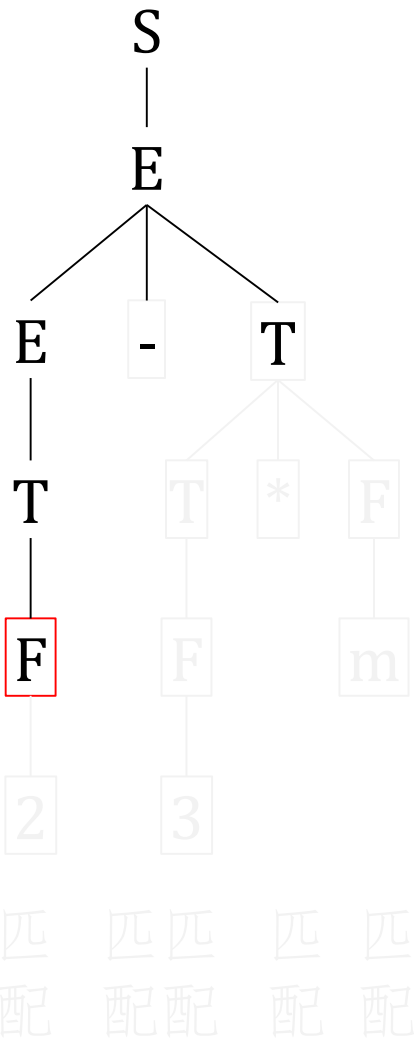
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

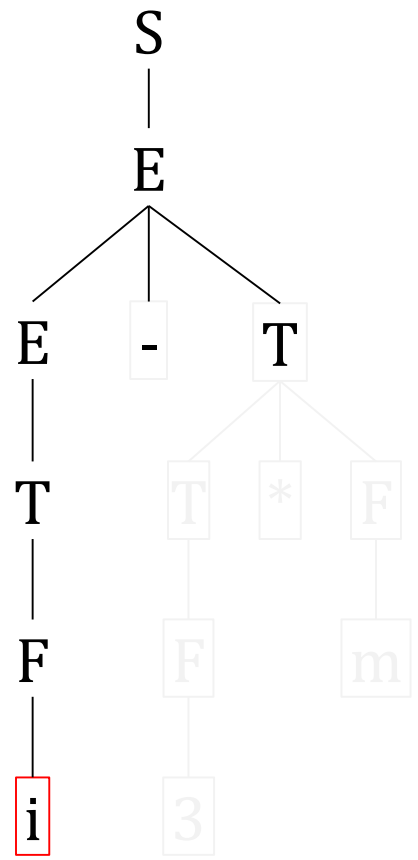
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

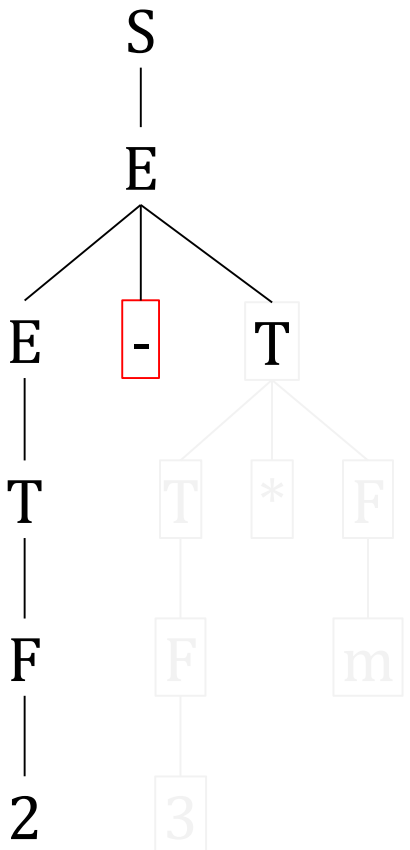
The Parsing Process

剩余输入串：

-3*m

^

当前输入符号



匹配

$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

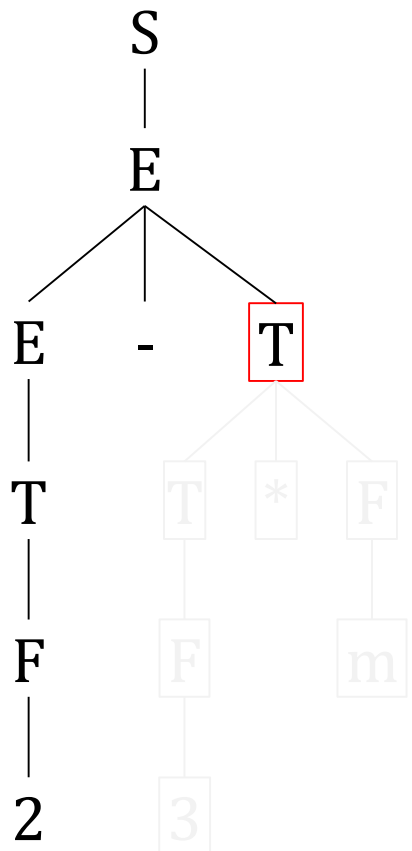
The Parsing Process

剩余输入串：

3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E – T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

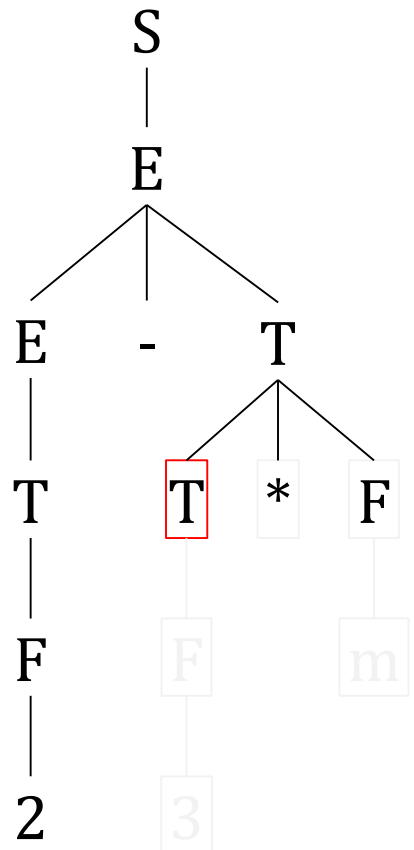
The Parsing Process

剩余输入串：

3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E – T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

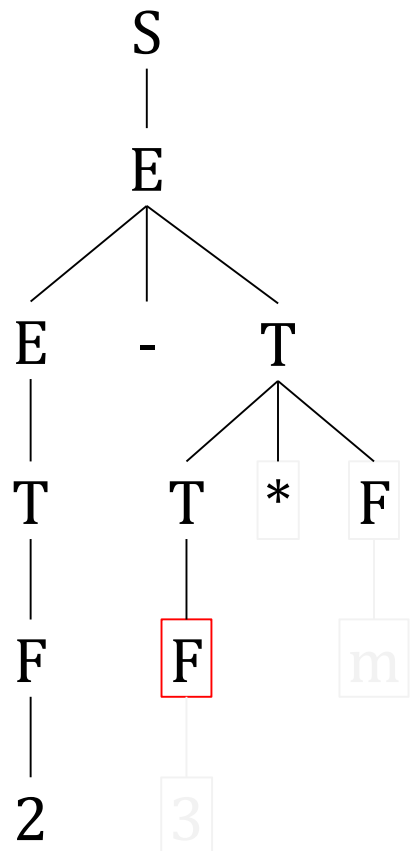
The Parsing Process

剩余输入串：

3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

匹配 匹配 匹配 匹配

文法：

S⟶E

E⟶E + T

E⟶E – T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

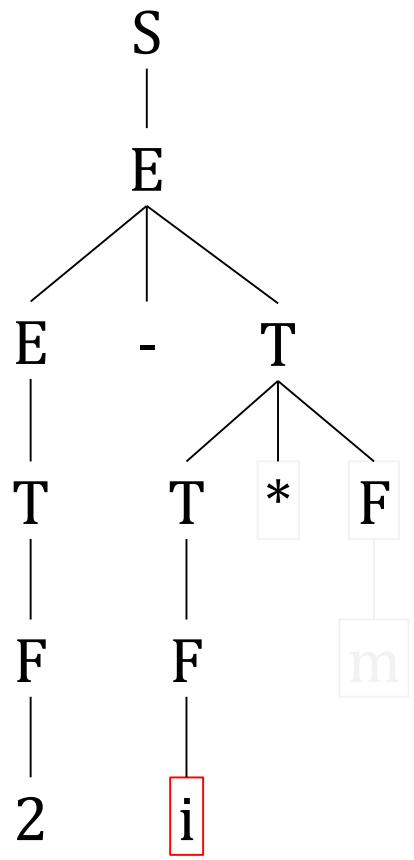
The Parsing Process

剩余输入串：

3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E – T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

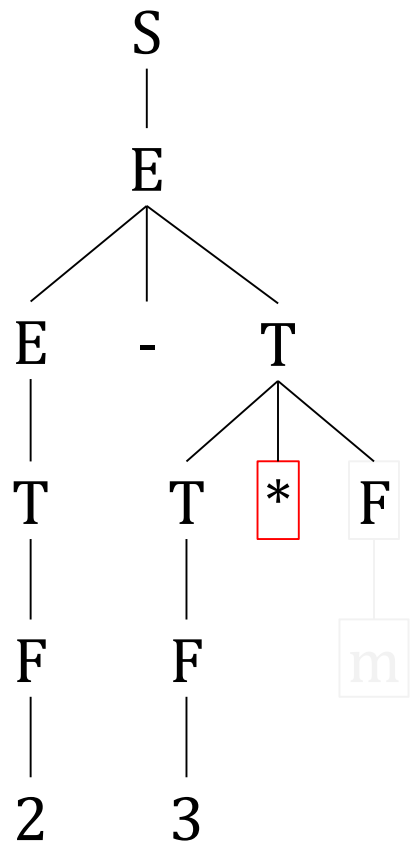
The Parsing Process

剩余输入串：

*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E – T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

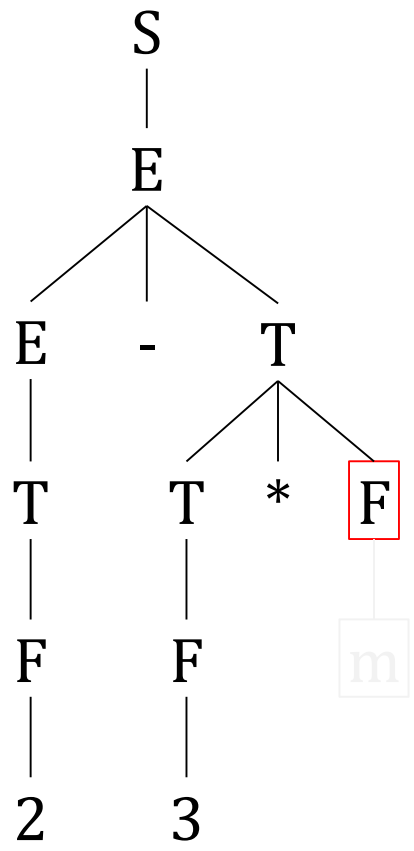
The Parsing Process

剩余输入串：

m

^

当前输入符号



匹配 匹配 匹配 匹配

$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E – T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

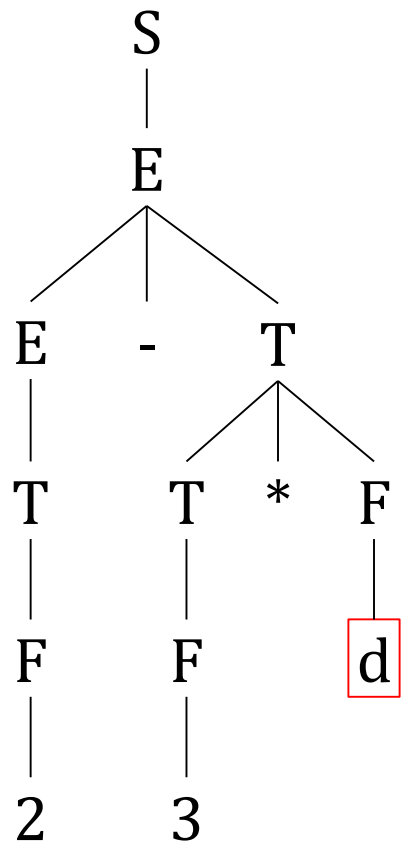
The Parsing Process

剩余输入串：

m

^

当前输入符号



匹配 匹配 匹配 匹配

$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E – T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

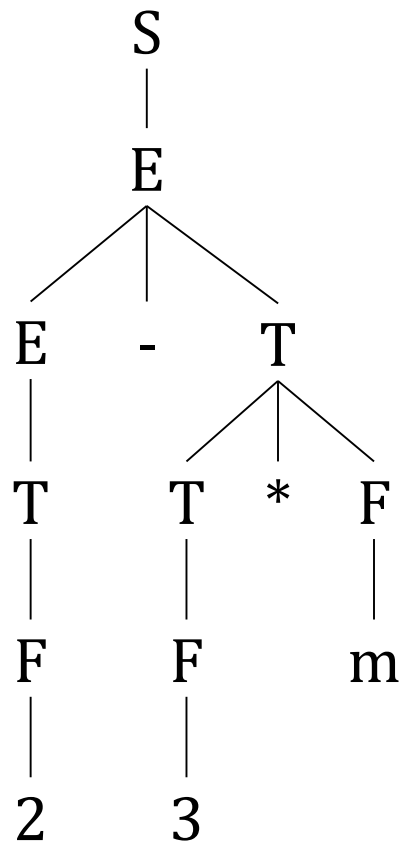
F⟶d

The Parsing Process

剩余输入串：

^

当前输入符号



匹 匹匹 匹 匹

配 配配 配 配

$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E – T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

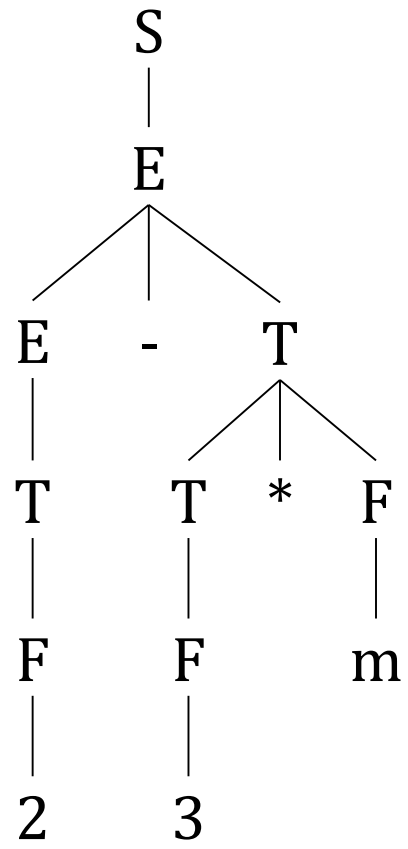
The Parsing Process

剩余输入串：

^

当前输入符号

成功！



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E – T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

The Parsing Process

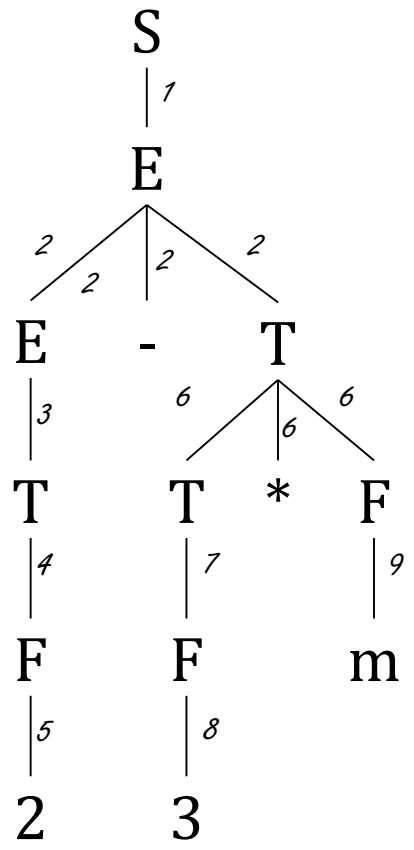
剩余输入串：

^

当前输入符号

成功！

1



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E – T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

观察

- 分析过程对应于扩展语法树的过程，并限定为与最左推导过程有对应关系。
- 在分析过程的每个步骤都知道当前唯一待扩展节点。
- 分析过程有三类基本步骤（也称分析动作）：
  - 当待扩展节点为变元时，应用产生式，扩展出该结节点的孩子节点，依次为它的候选式中文法符号所示；
  - 当待扩展节点为终结符时，匹配掉当前输入符号，除非匹配失败。匹配失败时简单地转错误处理，表示分析失败；
  - 当剩余串为空时分析成功。
- 事后看到，分析过程是先根遍历有序树（语法树）的过程。

6.1 自上而下分析面临的问题

- 在最左推导策略中，仍然存在不确定性，即待扩展变元有多个候选式可选其一。

- 由多候选式带来的不确定性导致分析过程可能出现两个问题。
  - 分析过程中的“死循环”问题；
  - 分析过程中的回溯问题。

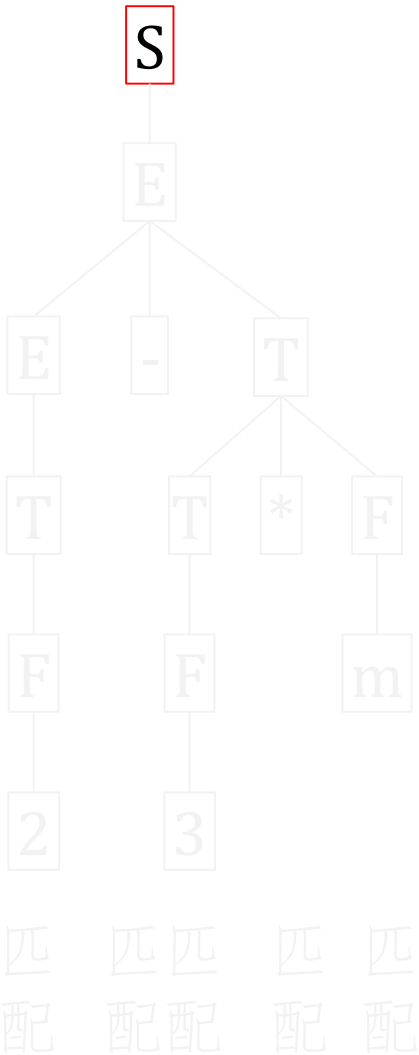
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

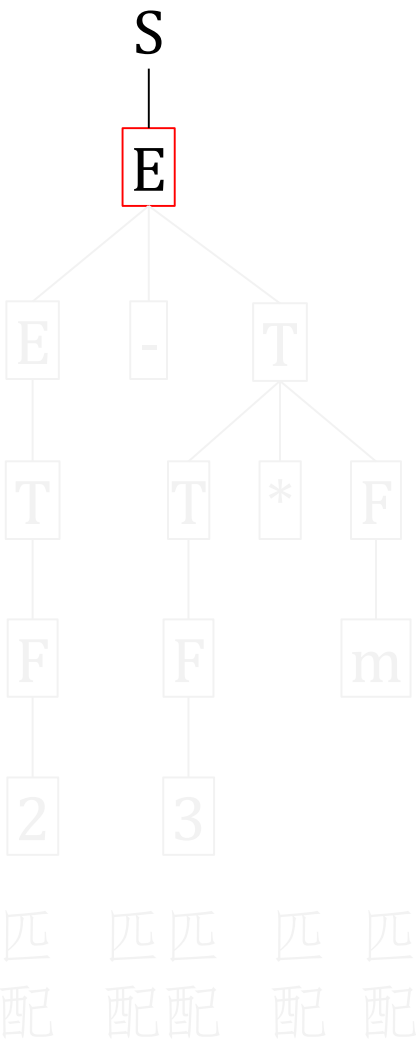
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

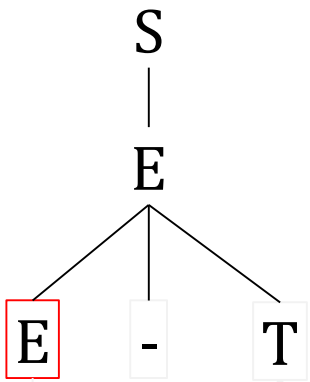
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

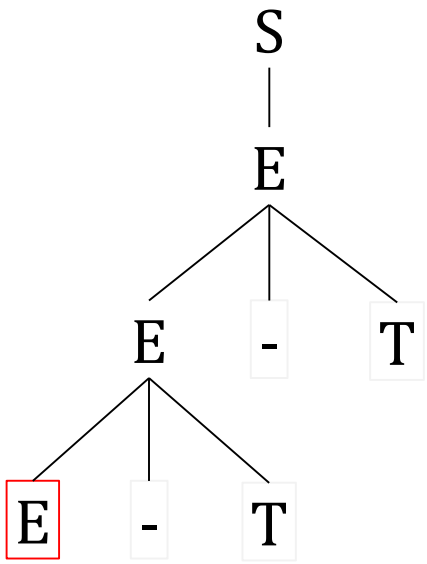
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

E-T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

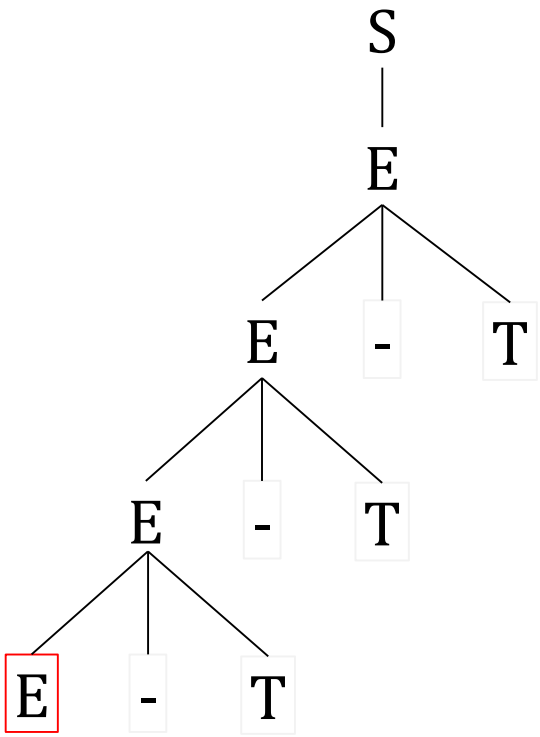
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

E-T-T

E-T-T-T

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

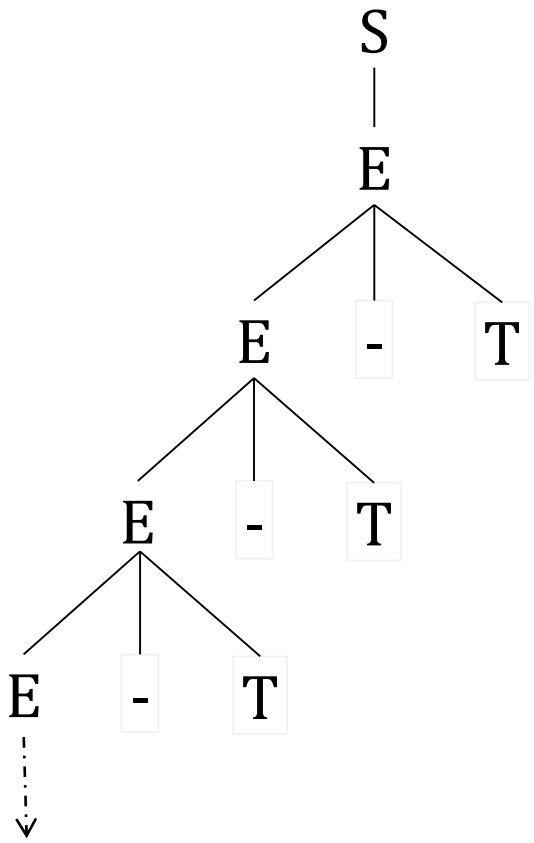
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



“死循环”了！

$\Rightarrow_{lm}^{*}$：

S

E

E-T

E-T-T

E-T-T-T

…

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

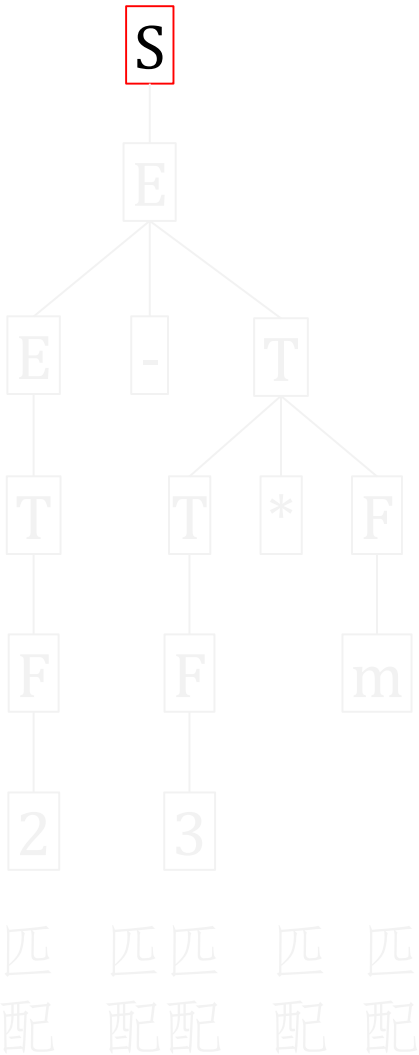
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

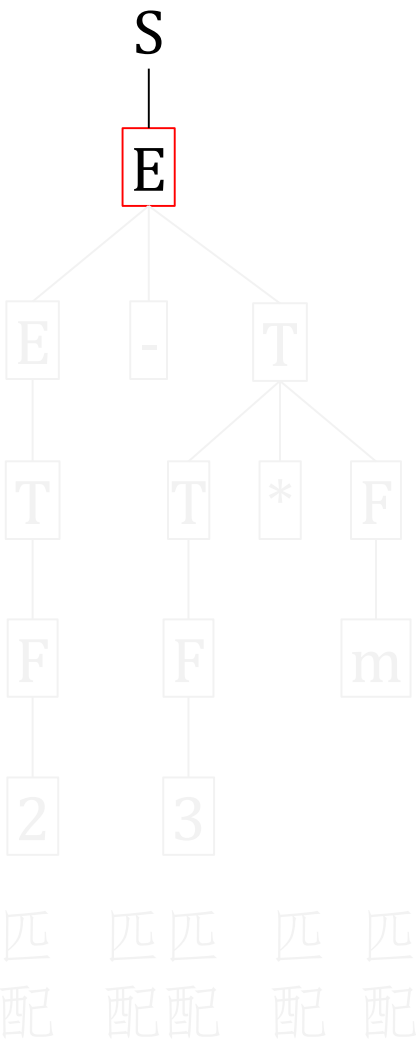
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

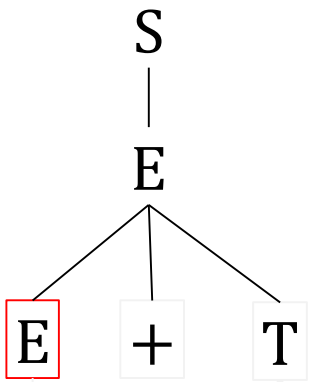
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E+T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

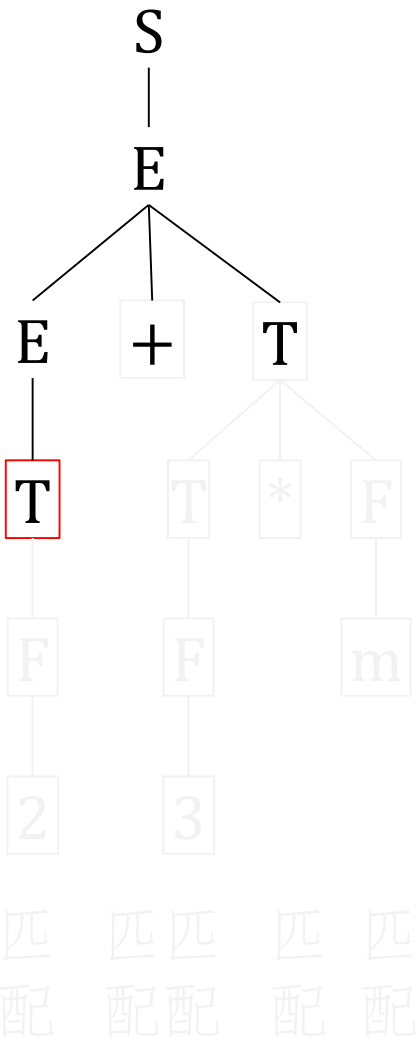
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E+T

T+T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

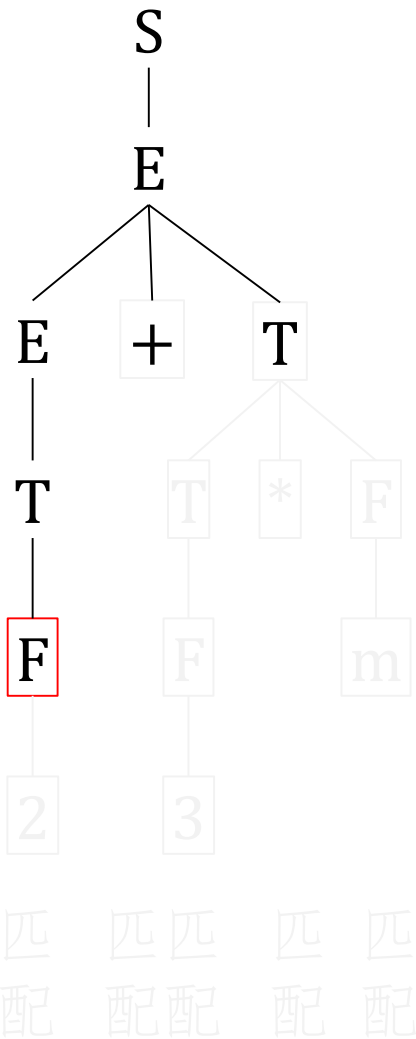
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E+T

T+T

F+T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

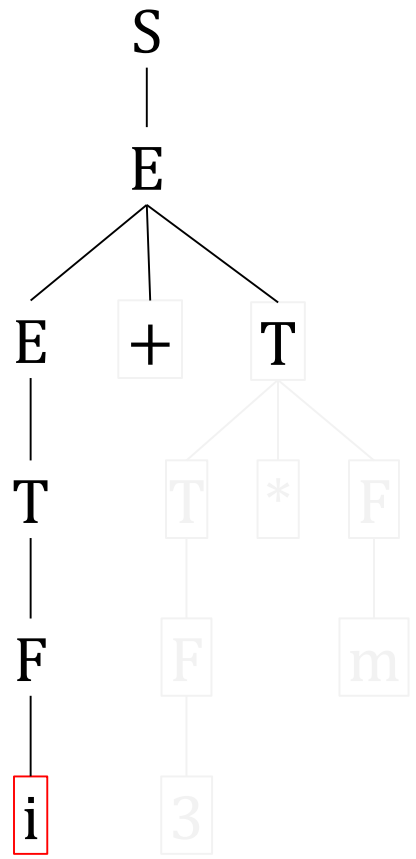
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E+T

T+T

F+T

2+T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

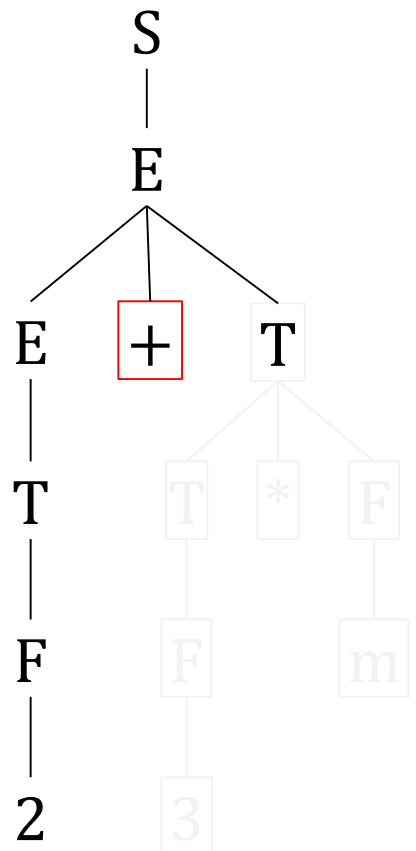
The Parsing Process

剩余输入串：

-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E+T

T+T

F+T

2+T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

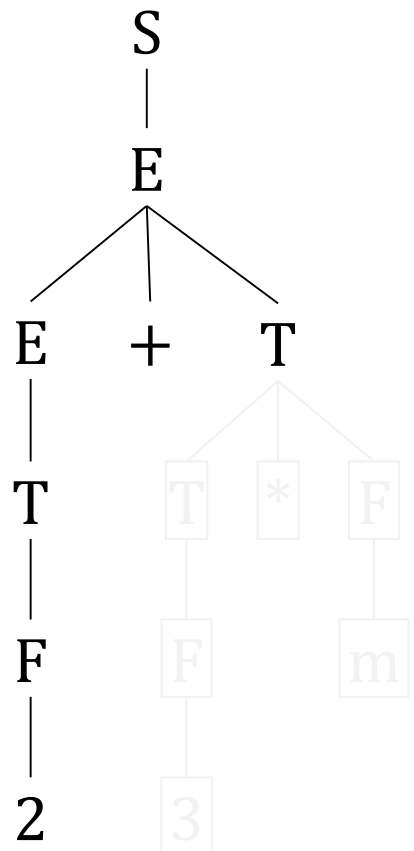
The Parsing Process

剩余输入串：

-3*m

^

当前输入符号



匹配失败！

$\Rightarrow_{lm}^{*}$：

S

E

E+T

T+T

F+T

2+T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

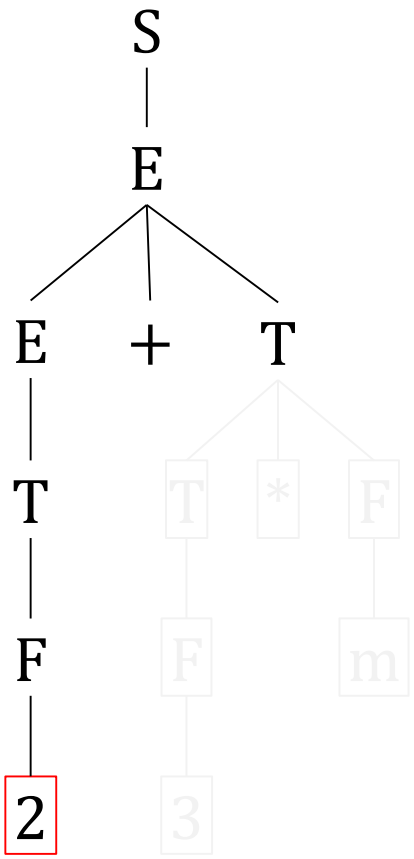
The Parsing Process

剩余输入串：

-3*m

^

当前输入符号



回溯！

$\Rightarrow_{lm}^{*}$：

S

E

E+T

T+T

F+T

2+T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

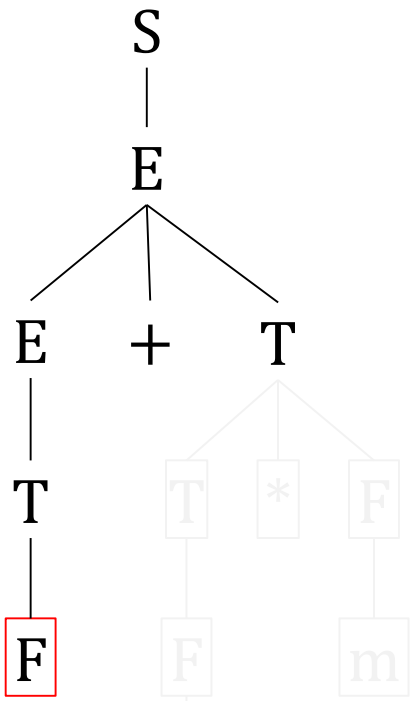
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



回溯！

$\Rightarrow_{lm}^{*}$：

S

E

E+T

T+T

F+T

2+T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

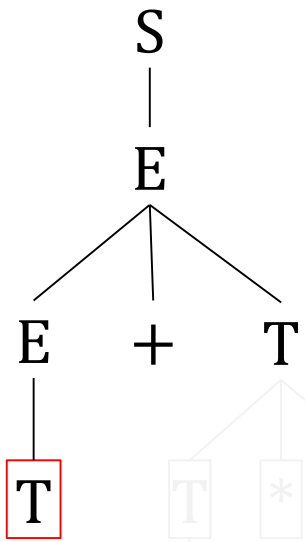
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



回溯！

$\Rightarrow_{lm}^{*}$：

S

E

E+T

T+T

F+T

2+T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

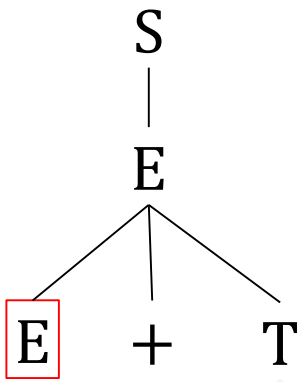
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



匹配 匹配 匹配 匹配

回溯！

$\Rightarrow_{lm}^{*}$：

S

E

E+T

T+T

F+T

2+T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

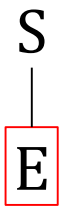
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



回溯！

$\Rightarrow_{lm}^{*}$：

S

E

E+T

T+T

F+T

2+T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

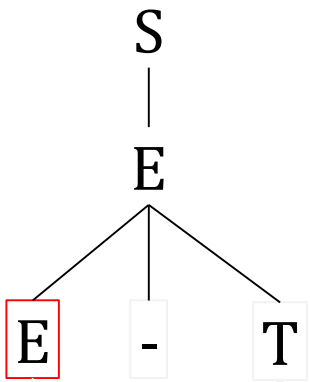
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

继续扩展下去

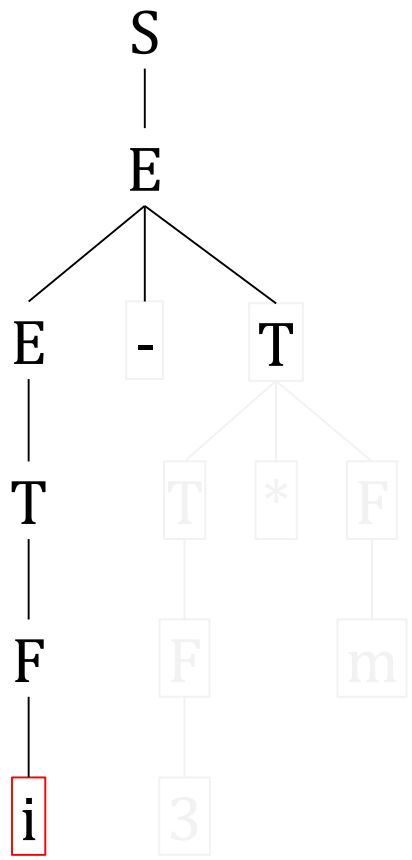
The Parsing Process

剩余输入串：

2-3*m

^

当前输入符号



继续扩展

$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

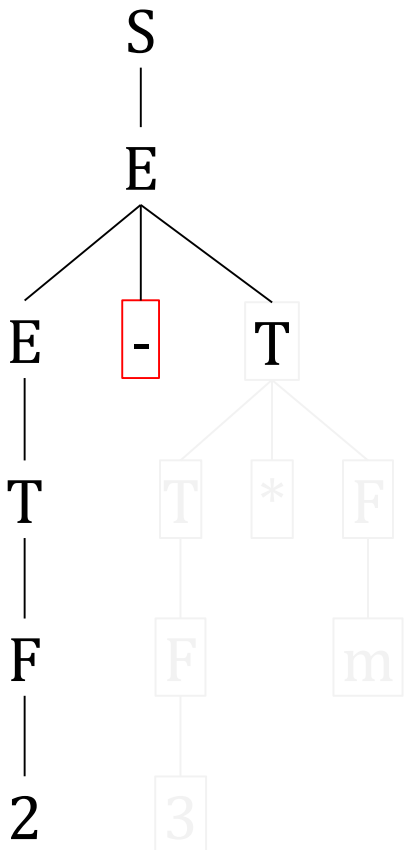
The Parsing Process

剩余输入串：

-3*m

^

当前输入符号



匹配

$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E - T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

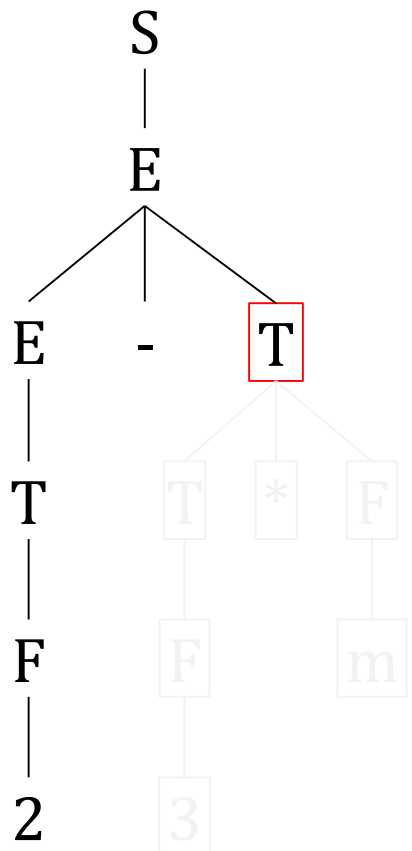
The Parsing Process

剩余输入串：

3*m

^

当前输入符号



$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E – T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

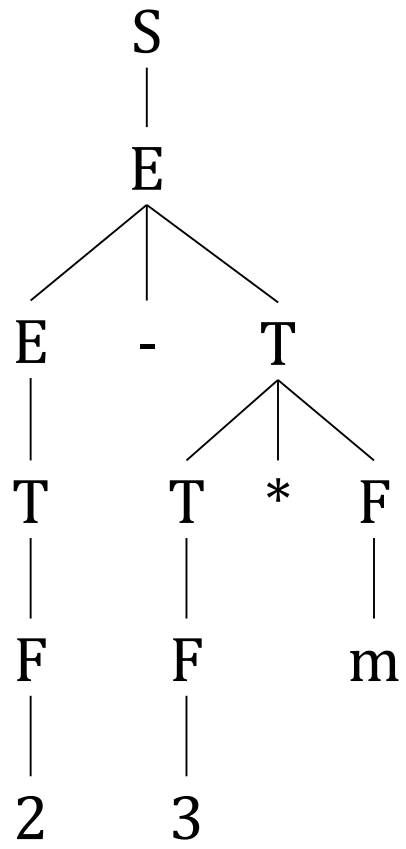
The Parsing Process

剩余输入串：

^

当前输入符号

成功！



过程曲折

$\Rightarrow_{lm}^{*}$：

S

E

E-T

T-T

F-T

2-T

2-T*F

2-F*F

2-3*F

2-3*m

文法：

S⟶E

E⟶E + T

E⟶E – T

E⟶T

T⟶T * F

T⟶T / F

T⟶F

F⟶(E)

F⟶i

F⟶d

归纳“死循环”与回溯问题

- 分析过程中出现“死循环”的原因：
  - 左递归文法，即导致如$P \Rightarrow^{+} P\alpha$这样的文法。
  - 扩展P节点或替换P时可能发生不消耗输入而反复扩展P节点的现象。
- 分析过程中的回溯问题
  - 把分析过程看作一个搜索过程，在每个选择点处试探下个选择，如果已经明确当前的试探失败，则退回到前一个选择点处并试探不同的选项。
  - 产生回溯的原因是在扩展变元节点时存在多个可用候选式。
- 对这些有不确定性的问题的解决对分析过程而言是必要的。
  - 分析过程显然不允许出现“死循环”；
  - 回溯问题让分析过程变得复杂（错误处理、效率），事实上可以确定化，这是本章的任务。

解决思路

- 对于分析过程面临的“死循环”和回溯问题，通过修剪文法得到解决。

- 对文法G进行复合p修剪
  - g（变元都是产出的）
  - a（文法符号都是可达的）
  - $\bar{e}$（不含ε产生式）
  - $\bar{u}$（不含单位产生式）
  - $\bar{r}$（不含左递归产生式）
  - $\bar{c}$（无回溯）
  - $\bar{a}$（无歧义性）
- 得到LL (1)文法，可表示为LL(1)·G

本章内容

- 6.2 LL(1)分析方法
  - 修剪文法以消除左递归和引起回溯的产生式规则得到LL(1)文法。
  - 6.2.1 消除文法中的左递归r̄
  - 6.2.2 消除回溯c̄，无回溯的分析步骤：
  - 6.2.3计算FIRST集和FOLLOW集
  - 6.2.4 LL(1)分析：条件与框架
- 6.3 分析程序的构造
  - 6.3.1 递归下降分析程序
  - 6.3.2 预测分析程序与预测分析表

6.2.1 消除文法左递归

- 直接左递归（消除）
  - $P \longrightarrow P\alpha \mid \beta$，其中$P$不是$\beta$的前缀。
- 间接左递归（消除）
  - $N \longrightarrow A\alpha$
  - $A \longrightarrow N\beta \mid \gamma$
- 间接左递归（回避）（通过限制文法为$G = \bar{e}G$可以做到）
  - $N \longrightarrow \alpha N$
  - $\alpha \Rightarrow^{*} \varepsilon$

消除直接左递归

- 直接左递归
  - $P \longrightarrow P\alpha \mid \beta$，其中$P$不是$\beta$的前缀。
  - $L(P)=L(\beta\alpha^*)$
- 修剪为：
  - $P \longrightarrow \beta P'$
  - $P' \longrightarrow \alpha P' \mid \varepsilon$
- 解释为：
  - $P \Rightarrow^* \beta\alpha^*$的另一种产生式表示
  - 不允许文法中有循环的情况$P \longrightarrow P$和$P \Rightarrow^+ P$

推广到一般情形

- 一般地，文法G中以P为左部的产生式形如：
  - $P \longrightarrow P\alpha_1 \mid \ldots \mid P\alpha_m \mid \beta_1 \mid \ldots \mid \beta_n$，
    其中每个α都不等于ε，而且每个β都不以P开头。

- 通过消除直接左递归得到：
  - $P \longrightarrow \beta_1 P' \mid \ldots \mid \beta_n P'$
  - $P' \longrightarrow \alpha_1 P' \mid \ldots \mid \alpha_m P' \mid \varepsilon$

- 例：$T \longrightarrow T*F \mid T/F \mid F$
- 结果文法：
  - $T \longrightarrow FT'$
  - $T' \longrightarrow *FT' \mid /FT' \mid \varepsilon$

消除间接左递归

- 间接左递归G：
  - $N \longrightarrow A\alpha$
  - $A \longrightarrow N\beta \mid \gamma$
- 应用代入法形成一个产生式之后，即为直接左递归消除之。

- 代入情形把A的代入N的：
  - $N \longrightarrow N\beta\alpha \mid \gamma\alpha$
  - $A \longrightarrow N\beta \mid \gamma$

结果文法$a\bar{r}G$：

$N \longrightarrow \gamma\alpha N'$

$N' \longrightarrow \beta\alpha N' \mid \varepsilon$

- 代入情形把N的代如A的：
  - $N \longrightarrow A\alpha$
  - $A \longrightarrow A\alpha\beta \mid \gamma$

结果文法$a\bar{r}G$：

$N \longrightarrow A\alpha$

$A \longrightarrow \gamma A'$

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

消除文法左递归的通用方法

- 输入文法满足$G = \bar{u}\bar{e}G$，即G不含单位产生式和ε产生式
- 对G的所有非终结符进行排序，按次序迭代：
  - 被代入的产生式是排在它前面的所有产生式。
  - 接着对被代入的产生式完成代入后立即消除其直接左递归，若有的话。
- 性质：不同排序得到的结果文法可能不同，但都是等价的。
- $\bar{r}$修剪算法：$L(G) = L(\bar{r} \cdot G) = L(\bar{r}G)$
- 算法有效的条件：输入文法不含单位产生式和ε产生式。
- 注意：结果文法中可能含有ε产生式。

消除文法左递归算法

输入：$G = \bar{u}\bar{e} \cdot G = (V, T, P, S)$。

输出：$\bar{r}G = (V_{\bar{r}}, T, P_{\bar{r}}, S)$不含左递归产生式。

令$P_{\bar{r}} = P$

对V的元素建立索引$V_i, 0 \leqslant i < |V|$（排序）

for$(i = 0; i < |V| ; i++)$ {

  for$(j = 0; j < i; j++)$ {

  将V_j的产生式代入V_i的产生式中，若可代入的话更新$P_{\bar{r}}$;

  }

消除V_i的直接左递归，若有的话更新$P_{\bar{r}}$

}

$P_{\bar{r}}$中所有变元构成集合$V_{\bar{r}}$

举例：消除下述文法的左递归

▻ 例1：

$S \longrightarrow Qc \mid c$

$Q \longrightarrow Rb \mid b$

$R \longrightarrow Sa \mid a$

▻ 排序为：R, Q, S

▻ $S \longrightarrow Sabc \mid abc \mid bc \mid c$

▻ $Q \longrightarrow Sab \mid ab \mid b$

▻ $R \longrightarrow Sa \mid a$

▻ 结果文法：

▻ $S \longrightarrow abcS' \mid bcS' \mid cS'$

▻ $S \longrightarrow abcS' \mid \varepsilon$

i	j	$P_{\bar{r}}$: $S \longrightarrow Qc \mid c$ $Q \longrightarrow Rb \mid b$ $R \longrightarrow Sa \mid a$
1	0	更新Q产生式为 $Q \longrightarrow Sab \mid ab \mid b$
2	1	更新S产生式为 $S \longrightarrow Sabc \mid abc \mid bc \mid c$
2	_	更新S产生式为 $S \longrightarrow abcS' \mid bcS' \mid cS'$ $S' \longrightarrow abcS' \mid \varepsilon$

▻ 排序为：S, Q, R

i	j	$P_{\bar{r}}$: $S \longrightarrow Qc \mid c$ $Q \longrightarrow Rb \mid b$ $R \longrightarrow Sa \mid a$
2	0	更新R产生式为 $R \longrightarrow Qca \mid ca \mid a$
2	1	更新R产生式为 $R \longrightarrow Rbca \mid bca \mid ca \mid a$
2	_	更新R产生式为 $R \longrightarrow bcaR' \mid caR' \mid aR'$ $R' \longrightarrow bcaR' \mid \varepsilon$

例2续：

- $A \longrightarrow Bb \mid Ac \mid c$
- $B \longrightarrow Ba \mid Ad \mid d$

排序A，B

i	j	P_{er}：$A \longrightarrow c \mid Ac \mid Bb$ $B \longrightarrow d \mid Ad \mid Ba$
0	-	更新A产生式为 $A \longrightarrow BbA' \mid cA'$ $A' \longrightarrow cA' \mid \varepsilon$
1	0	更新B产生式为 $B \longrightarrow Ba \mid BbA'd \mid cA'd \mid d$
1	-	更新B产生式为 $B \longrightarrow cA'dB' \mid dB'$ $B' \longrightarrow aB' \mid bA'dB' \mid \varepsilon$

- i=0, , j=-（第7行）
- 消除A直接左递归
- $A \longrightarrow BbA' \mid cA'$
- $A' \longrightarrow cA' \mid \varepsilon$

- i=1, j=0
- A代入B中
- $B \longrightarrow Ba \mid BbA'd \mid cA'd \mid d$

- i=1, j=- 第7行）
- 消除B 直接左递归
- $B \longrightarrow cA'dB' \mid dB'$
- $B' \longrightarrow aB' \mid bA'dB' \mid \varepsilon$

- 结果文法
- $A \longrightarrow BbA' \mid cA'$
- $A' \longrightarrow cA' \mid \varepsilon$
- $B \longrightarrow cA'dB' \mid dB'$
- $B' \longrightarrow aB' \mid bA'dB' \mid \varepsilon$

for(i=1; i< $|V|$; i++)
  for(j=1； j< i-1； j++)
    将P_j代入P_i的每个候选中；
  消除关于P_i的直接左递归

6.2.2 消除回溯

- 产生回溯的原因分析：
  - 尽管遵守最左推导策略，但多候选式变元在被用于形成直接推导时存在不确定性，从而可能引起分析过程发生回溯。
- 产生回溯的原因：
  - 当一个变元的多个候选式都有公共前缀时，无法根据当前输入符号决定用哪个候选式扩展这个变元。
  - 当一个变元的多个候选式推导出的串之间有一些有公共终结符前缀时，无法根据当前输入符号决定用哪个候选式扩展这个变元。
  - 对于当前输入符号，有ε候选式的待扩展变元让分析过程不确定是应用该ε产生式呢还是用其它可应用候选式。
- 考虑消除回溯的方法：
  - 分别针对这三个原因构建消除回溯的方法。

例：产生回溯的原因分析

- 假设 T' 是语法树中的当前待扩展节点
- 现有三个产生式可用
  - $T' \longrightarrow * i T'$
  - $T' \longrightarrow / i T'$
  - $T' \longrightarrow \varepsilon$

（1）原文法：

$T \longrightarrow T*F \mid T/F \mid F$

$F \longrightarrow i$

（2）消除左递归得：

$T \longrightarrow FT'$

$T' \longrightarrow *FT' \mid /FT' \mid \varepsilon$

（3）没有“死循环”

- 结合向前查看一个输入符号，那么，
  - 若当前输入符号是 *，则应用 $T' \longrightarrow * i T'$ 实现推导。
  - 若当前输入符号是 /，则应用 $T' \longrightarrow / i T'$ 实现推导。
  - 若是其它情况，那么是否应用 $T' \longrightarrow \varepsilon$ 实现推导？
  - 若候选式有相同的终结符前缀，怎么办？
  - 进一步，候选式所能推导出的串的情况怎么考虑？

原因1：多个候选式有公共前缀

- 例如文法：
  - $N \longrightarrow$ if then
  - $N \longrightarrow$ if then else
- 假定 N 是待扩展变元，if 是当前输入符号，两个候选式都可用。

- 解决办法：让公共前缀只出现在单个产生式中
  - $N \longrightarrow$ if then N'
  - $N' \longrightarrow$ else $\mid \varepsilon$
- 这时，当前输入符号为if时，分析过程只有唯一选择。
- 结论：同一个变元的有公共前缀的候选式们合并为一个最大公共前缀与一个新的变元连接，并将余下的部分都作为新变元的候选式。

问题1的解决

- 若有$A \longrightarrow x\beta \mid x\gamma \mid x\delta \mid \eta \mid \ldots \mid \mu$，其中$x \neq \varepsilon$，$\eta,\ldots,\mu$都没有前缀$x$，
- 那么将其修剪为：
  - $A \longrightarrow xA' \mid \eta \mid \ldots \mid \mu$
  - $A' \longrightarrow \beta \mid \gamma \mid \delta$
- 如法继续迭代处理余下的候选式$xA',\eta,\ldots,\mu$，直到A的候选式彼此都没有公共前缀为止。
- 同时还要如法处理新引入的变元A'等。

原因2：候选式推导出的串有终结符公共前缀

- 例：候选式第一个符号为非终结符时，如：

  $N \longrightarrow N_1\ \alpha_1$

  $N \longrightarrow N_2\ \alpha_2$

- 选择哪一个取决于N_1和N_2推导出来的串的第一个终结符跟当前输入符号的匹配情况。
- 若N_1或N_2能推导出ε，还需要考虑到α_1和α_2进行判断
- 所以统一观察候选式推导终结符前缀的情况，再进行处理，但仍未考虑到候选式推导不出来终结符前缀的情况（这属于原因3）。

问题2的解决

- 问题描述：
  - 当前输入符号为a，待扩展变元为A时，
  - 若有$A \longrightarrow \alpha \mid \ldots \mid \beta \mid \eta \mid \ldots \mid \mu$，$\alpha \Rightarrow^* a\gamma, \ldots, \beta \Rightarrow^* a\delta\eta$，其中$a \in T$，且$\eta,\ldots,\mu$都不能推导出前缀为$a$的串，
  - 那么，选择哪个候选式进行扩展？
- 解决办法：
  - 定义α的首符集，FIRST(α)为α能推导出来的所有串的终结符前缀的集合。若该串为ε那么首符集中包含ε。
  - 修剪文法让每一个变元都满足它的候选式的首符集都两两不相交。

问题3：ε候选式带来的不确定性

- 对于当前输入符号，有ε候选式的待扩展变元让分析过程应用该ε产生式还是其它可应用候选式变得不确定。
- 推广到一般：
  - 对于 $A \longrightarrow \alpha_1 \mid \ldots \mid \alpha_n$，其中$n>0$，存在$k, 1 \leqslant k \leqslant n$，有$\varepsilon \in \mathrm{FIRST}(\alpha_k)$。
  - A已满足它的任意两个候选式的首符集都不相交，即 $\mathrm{FIRST}(\alpha_i) \cap \mathrm{FIRST}(\alpha_j),\ i \neq j$
- 讨论：$n=1$；$n>1$；k唯一；k不唯一。
- 问题出在当前输入符号不在任意候选式的首符集里时，是否可以使用α_k扩展A?

问题3的解决

- 若A的候选式的首符集里找不到当前输入符号a时，再看A的FOLLOW集中有没有，若有就用α_k扩展A，否则按出错对待。
- FOLLOW(A)为所有句型中能紧跟在A后面的那些终结符组成的集合。
- 修剪文法对于每个变元，它的候选式首符集两两不相交，同时，若有首符集含ε的话还要求所有候选式的首符集都与左部变元的FOLLOW集不相交。
- 分析过程示范：当a和A分别为当前输入符号和当前待扩展变元时：①若a属于A的候选式α的首符集FIRST(α)，那么应用α扩展A；②若A的候选式α的首符集FIRST(α)包含ε，并且a属于FOLLOW(A)，那么用α扩展A；③其它情况为语法错误。

小结：回溯问题的解决

对于文法G的产生式形如：$A \longrightarrow \alpha_1 | \ldots | \alpha_n$

可以计算出每个候选式的首符集。如果任意两个候选式的首符集都两两不相交的话，则意味着由A的每个候选式所能够推导出来的符号串的首符号均不相同。

于是，对语法树中当前节点A，并且当前输入符号为a时只能够唯一地选择A的候选α_k，即$a \in \text{FIRST}(\alpha_k)$。

如果还不能选出候选式，则看A的首符集有没有ε，若有的话，再看a是否属于A的FOLLOW集（这里要求A的FOLLOW集与A的首符集不相交），若是，则对A节点唯一地选择A的可空候选式进行扩展。

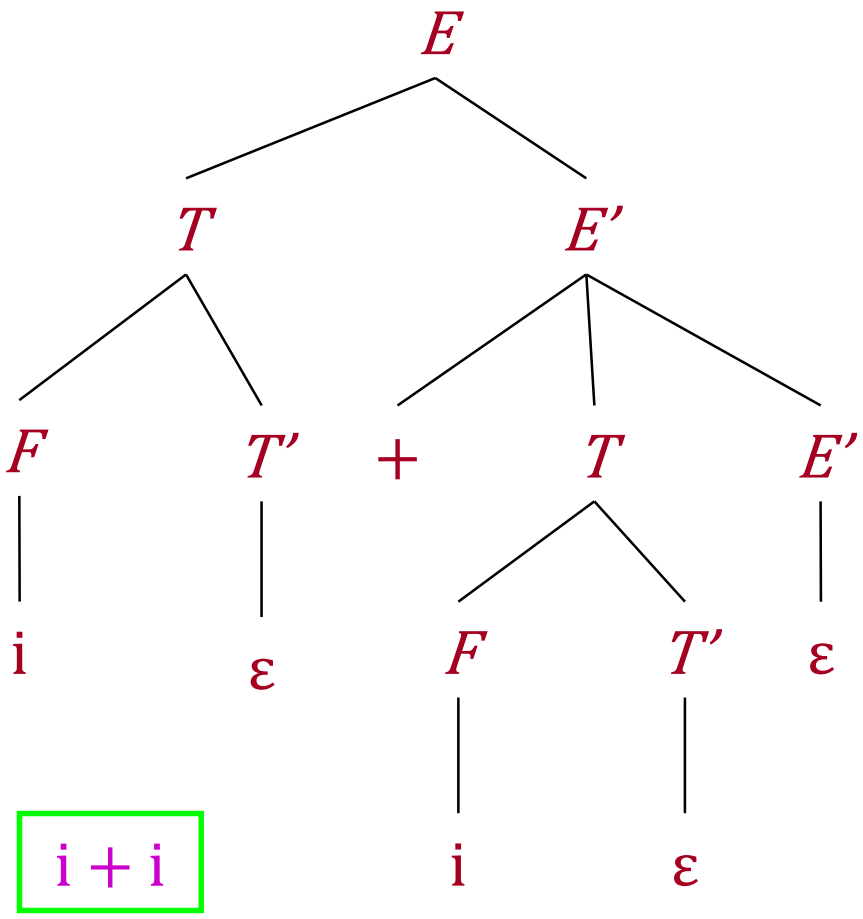
其它情况属于语法错误。

把确定性分析过程提炼为

- 设当前输入符号为a，语法树当前待扩展节点为A，且有规则$A \longrightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$，
  - 若 $a \in \text{FIRST}(\alpha_k)$，则采用规则$A \longrightarrow \alpha_k$扩展；
  - 若 $a \notin \text{FIRST}(A)$，但是有某个k有$\varepsilon \in \text{FIRST}(\alpha_k)$且$a \in \text{FOLLOW}(A)$，则采用规则$A \longrightarrow \alpha_k$扩展 。
- 若不满足上述情况，则a的出现是一种语法错误。

例：无回溯分析

$E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow iT'E'$



i + i

$E \longrightarrow TE'$

$E' \longrightarrow +TE' \mid -TE' \mid \varepsilon$

$T \longrightarrow FT'$

$T' \longrightarrow *FT' \mid /FT' \mid \varepsilon$

$F \longrightarrow (E) \mid i$

$i \in FIRST(TE')$

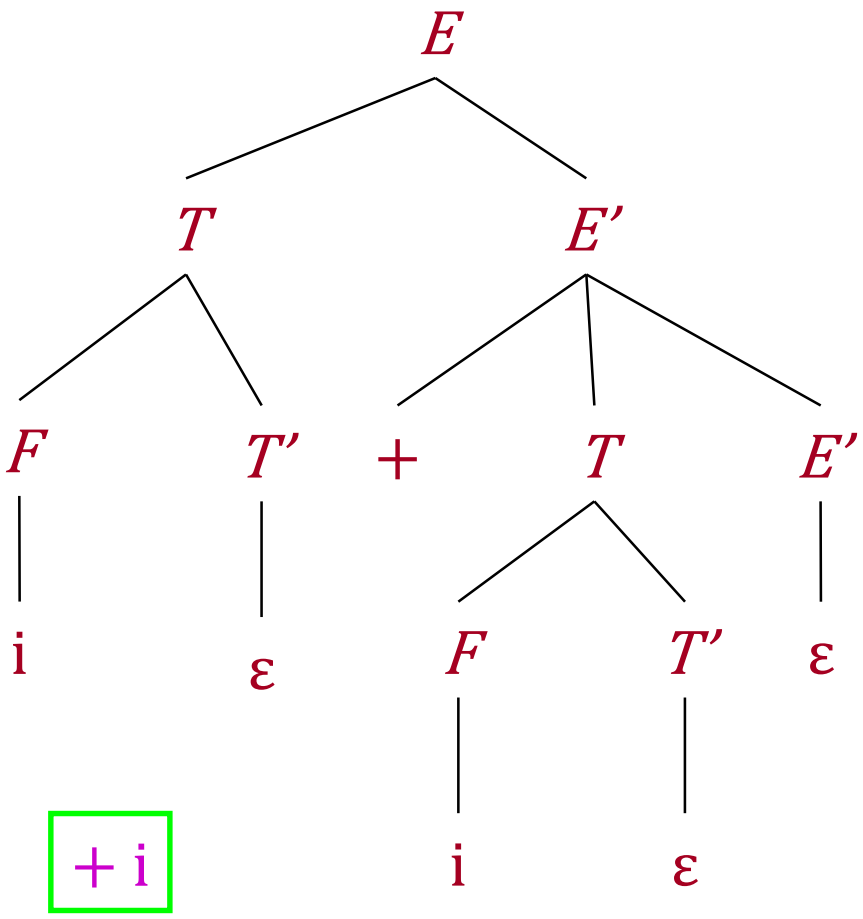
$i \in FIRST(FT')$

$i \in FIRST(i)$

例：无回溯分析

E ⇒ TE' ⇒ FT'E' ⇒ iT'E'

⇒iE'⇒i+TE'



+ i

E⟶TE'

E'⟶+TE' | -TE' | ε

T⟶FT'

T'⟶*FT' | /FT' | ε

F⟶(E) | i

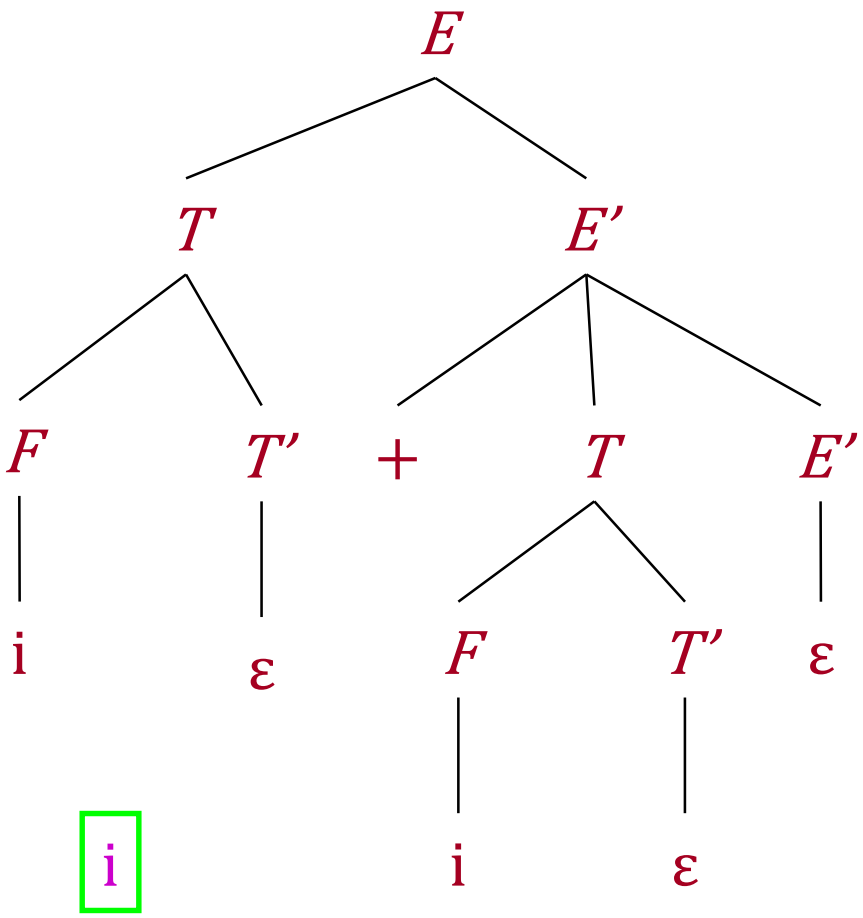
ε ∈FIRST(T')；+∈FOLLOW(T')

+∈FIRST(+TE')

例：无回溯分析

$E \Rightarrow TE' \Rightarrow FT'E' \Rightarrow iT'E'$

$\Rightarrow iE' \Rightarrow i+TE' \Rightarrow i+FT'E' \Rightarrow i+iT'E'$



$E \longrightarrow TE'$

$E' \longrightarrow +TE' \mid -TE' \mid \varepsilon$

$T \longrightarrow FT'$

$T' \longrightarrow *FT' \mid /FT' \mid \varepsilon$

$F \longrightarrow (E) \mid i$

$i \in FIRST(FT')$

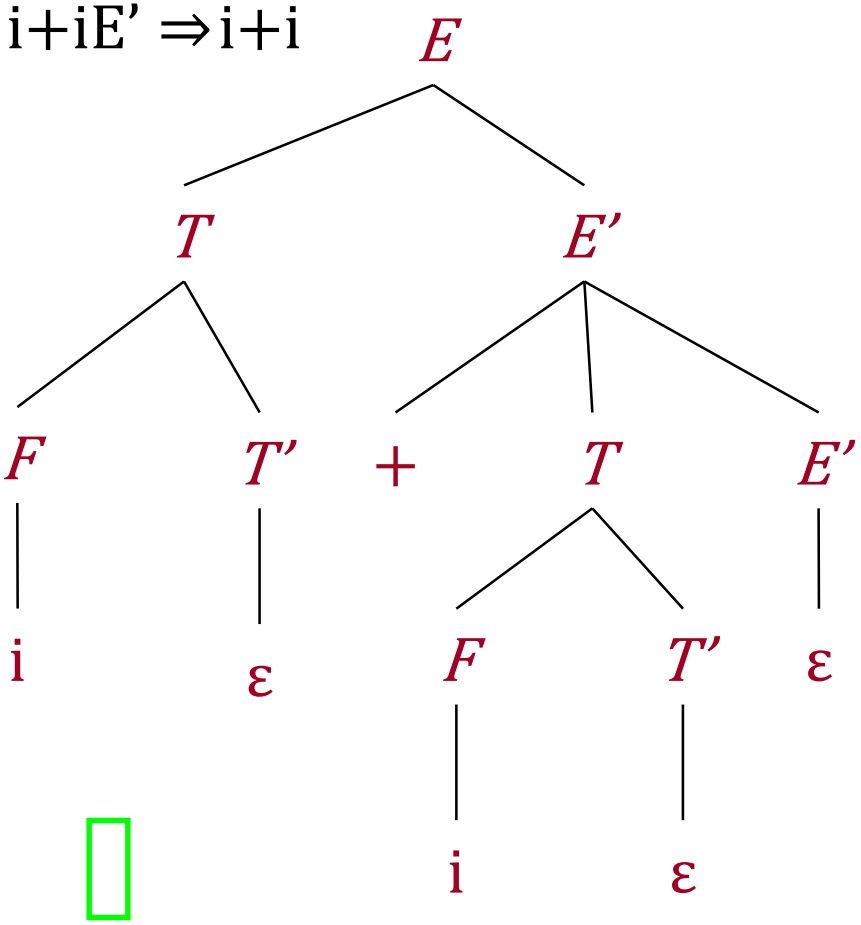
$i \in FIRST(i)$

例：无回溯分析

E ⇒ TE' ⇒ FT'E' ⇒ iT'E'

⇒iE'⇒i+TE' ⇒i+FT'E' ⇒i+iT'E'

⇒i+iE' ⇒i+i



E⟶TE'

E'⟶+TE' | -TE' | ε

T⟶FT'

T'⟶*FT' | /FT' | ε

F⟶(E) | i

ε ∈FIRST(T')； #∈FOLLOW(T')

ε ∈FIRST(E')； #∈FOLLOW(E')

6.2.3 无回溯的分析步骤

- 计算首符集和FOLLOW集；
- 修剪文法为LL(1)文法；
- 构建LL(1)分析框架。

计算首符集和FOLLOW集

- FIRST(α)为α能推导出来的所有串的终结符前缀的集合。若该串为ε那么首符集中包含ε。
  - FIRST(α)=$\{a \mid \alpha \Rightarrow^* a\beta,\ \ a\in T, \beta\in(V\cup T)^*\}$;
  - 若$\alpha \Rightarrow^*\varepsilon$，则$\varepsilon\in$FIRST($\alpha$)。
- FOLLOW(A)为所有句型中能紧跟在A后面的那些终结符组成的集合。
  - FOLLOW(B) = $\{a\in T\cup\{\#\} \mid S\Rightarrow^*\alpha B\gamma, a\in\text{FIRST}(\gamma\#)\}$
  - 为了一致期间，给输入串加一个后缀#，该符号被解释为终结符，但不属于T。
  - 初始符号的FOLLOW集中必然有#。
- 计算方法的显著特点是反复扫描产生式规则最终得到结果。

计算文法符号的首符集

输入：CFG(V,T,P,S)；

输出：文法符号的首符集

for all $a \in T$ do FIRST(a):={a}；

for all $A \in V$ do FIRST(A):={}；

While there are changes to any FIRST(A) do

    for each production $A \longrightarrow \alpha$ do

    add FIRST(α) to FIRST(A)；

计算符号串的首符集

输入：CFG(V,T,P,S)，$\alpha=X_1X_2\ldots X_n$；　$\alpha\in(V\cup T)^*$，　$n\geq 0$

输出：　FIRST$(X_1X_2\ldots X_n)$

$k:=1$; Continue := true;

while (Continue = true && $k\leqslant n$) {

　　add FIRST$(X_k)\setminus\{\varepsilon\}$ to FIRST(α);

　　if (ε is not in FIRST(X_k)) Continue := false;

　　$k:=k+1$;}

if (Continue = true) add $\{\varepsilon\}$ to FIRST(α);

对于候选式α:
$\alpha=\varepsilon$，ε加入FIRST(α)
$\alpha=a$，FIRST$(\alpha)=\{a\}$
$\alpha=a\ldots$，a加入FIRST(α)

α为其他，找出α的最大前缀ρ且$\rho\Rightarrow^*\varepsilon$，让$\rho$的每个元素的首符集\{$\varepsilon$\}都加入FIRST$(\alpha)$
如果$\rho=\alpha$那么ε加入FIRST(α)

计算文法符号的首符集

输入： CFG (V, T, P, S)。

输出： 首符集索引表FIRST[]，其中以文法符号和候选式作索引。

for($a\in T$) FIRST[a] = list(a);　　for($A\in V$) FIRST[A] = NIL;

for((A, α)$\in P$) FIRST[α] = NIL;

do { tmp = FIRST[];

　for((A, α)$\in P$) { $n = |\alpha|$; $k = 1$; cont = TRUE;

　　while(cont==TRUE&&$k\leqslant n$) { X = nth(k, α);

　　FIRST[α] = FIRST[α] ∪ (FIRST[X]\(ε));

　　if($\varepsilon\notin$FIRST[X])cont = FALSE;

　　k++ }

　　if(cont==TRUE) FIRST[α] = cons(ε, FIRST[α]);

　　FIRST[A] = FIRST[A] ∪ FIRST[α]　}

}while(tmp!=FIRST[])。

对于候选式α：

$\alpha=\varepsilon$，ε加入FIRST(α)

$\alpha=c$，FIRST(α)={c}

$\alpha=c\beta$，c加入FIRST(α)

α为其他，找出α的最大前缀ρ且$\rho\Rightarrow^{*}\varepsilon$，让$\rho$的每个元素的首符集\{$\varepsilon$}都加入FIRST($\alpha$)

如果$\rho=\alpha$那么ε加入FIRST(α)

$A\longrightarrow\alpha$，FIRST(α)加入FIRST(A)

例：计算文法符号的FIRST集

- P⟶Ď Š
- Ď⟶ε | D; Ď
- Š⟶S | Š; S
- D⟶T d
- T⟶int
- S⟶d(e)

;	{;}		
d	{d}		
int	{int}		
(	{(}		
e	{e}		
)	{)}		
P	{}	{}	{int,d}
Ď	{ε}	{ε,int}	
Š	{}	{d}	
D	{}	{int}	
T	{int}		
S	{d}		

对于候选式α：
$\alpha=\varepsilon$，ε加入FIRST(α)
$\alpha=c$，FIRST(α)={c}
$\alpha=c\beta$，c加入FIRST(α)

α为其他，找出α的最大前缀ρ且$\rho\Rightarrow^{*}\varepsilon$，让$\rho$的每个元素的首符集\{$\varepsilon$}都加入FIRST($\alpha$)
如果$\rho=\alpha$那么ε加入FIRST(α)

$A\longrightarrow\alpha$，FIRST(α)加入FIRST(A)

例：计算文法符号的FIRST集

- E⟶TE'
- E'⟶+TE' | -TE' | ε
- T⟶FT'
- T'⟶*FT' | /FT' | ε
- F⟶(E) | i

- \+ {+}
- \- {-}
- * {*}
- / {/}
- ({(}
-) {)}
- i {i}
- E {} {(,i}
- E' {+,-,ε}
- T {} {(,i}
- T' {*,/,ε}
- F {(,i}

计算FOLLOW集的约束规则

- 对于任一非终结符A，FOLLOW(A)是一个由终结符组成的集合，可能含#，

  - 若A为文法初始符号，则#属于FOLLOW(A)　　（规则1）

  - 若有产生式 $B \longrightarrow \alpha A \beta$

    则　　$\text{FIRST}(\beta) \setminus \{\varepsilon\} \subseteq \text{FOLLOW}(A)$　　（规则2）

  - 若有产生式 $B \longrightarrow \alpha A \beta$ 且$\varepsilon \in \text{FIRST}(\beta)$

    则　　$\text{FOLLOW}(B) \subseteq \text{FOLLOW}(A)$　　（规则3）

计算FOLLOW集的算法

输入：CFG(V,T,P,S)；输出：每个非终结符的FOLLOW集

FOLLOW(S) := {#}; //规则1

for all A∈V do if A≠S then FOLLOW (A):={}; //规则1

while there are changes to any FOLLOW(A) do

for each production $A \longrightarrow X_1X_2\ldots X_n$ do

for i := 1 to n do

if $X_i \in V$ then

add $FIRST(X_{i+1}X_{i+2}\ldots X_n) \setminus \{\varepsilon\}$ to $FOLLOW(X_i)$//规则2

if $\varepsilon \in FIRST(X_{i+1}X_{i+2}\ldots X_n)$ then

add FOLLOW(A) to $FOLLOW(X_i)$ //规则3

注：当i=n时$X_{i+1}X_{i+2}\ldots X_n=\varepsilon$

例：计算FOLLOW集

- ; {;}
- d {d}
- int {int}
- ({(}
- e {e}
-) {)}
- P {int,d}
- Ď {ε,int}
- Š {d}
- D {int}
- T {int}
- S {d}

- 规则1：FOLLOW(S)={#}，其它空；
- 规则2：若A⟶αBβ是一个产生式，则将FIRST(β)\{ε}加入FOLLOW(B)中；
- 规则3：若A⟶αB是一个产生式，或者A⟶αBβ是产生式且ε∈FIRST(β)则将FOLLOW(A)加入FOLLOW(B)中。

P⟶Ď Š
Ď⟶ε | D; Ď
Š⟶S | Š; S
D⟶T d
T⟶int
S⟶d(e)

	规则1	规则2	规则3
P	{#}		
Ď	{}	{d}	
Š	{}	{;}	{#,;}
D	{}	{;}	
T	{}	{d}	
S	{}		{#,;}

计算文法各变元FOLLOW集

输入：CFG (V, T, P, S)。

输出：V的元素A的FOLLOW集为FOLLOW[A]。

FOLLOW[S] = list(#); //规则1

for($A \in V$) if($A \neq S$) FOLLOW[A] = NIL; //初始化

do {tmp = FOLLOW[];

for($(A, \alpha) \in P$) { $n = |\alpha|$; $k = 1$;

while($k \leqslant n$) {

$X = \text{nth}(k, \alpha)$;

$\eta = \tau(\alpha, n-k)$;

if($X \in V$) FOLLOW[X] = FOLLOW[X] ∪ (FIRST[η]\(ε)); //规则2

if($\varepsilon \in$FIRST[η]) FOLLOW[X] = FOLLOW[X] ∪ FOLLOW[A]; //规则3

k++ } }

}while(tmp!=FOLLOW[])。

例：计算FOLLOW集

- + {+}
- - {-}
- * {*}
- / {/}
- ({(}
-) {)}
- i {i}
- E {(,i}
- E' {+,-,ε}
- T {(,i}
- T' {*,/,ε}
- F {(,i}

E⟶TE'
E'⟶+TE' | -TE' | ε
T⟶FT'
T'⟶*FT' | /FT' | ε
F⟶(E) | i

	规则1	规则2	规则3
E	{#}	{#,)}	
E'	{}	{}	{#,)}
T	{}	{+,-}	{+,-,#,)}
T'	{}	{}	{+,-,#,)}
F	{}	{*,/}	{*,/,+,-,#,)}

- 规则1：FOLLOW(S)={#}，其它空；
- 规则2：若A⟶αBβ是一个产生式，则将FIRST(β)\{ε}加入FOLLOW(B)中；
- 规则3：若A⟶αB是一个产生式，或者A⟶αBβ是产生式且ε∈FIRST(β)则将FOLLOW(A)加入FOLLOW(B)中。

计划进度

- 已有知识：
  - 消除文法左递归
  - 消除文法回溯
  - 计算串的首符集
  - 计算变元的FOLLOW集
  - 知道LL(1)文法的条件
- 本次课：
- 通过归纳、思考自上而下语法分析的深层问题取得思维进阶
- 熟练运用复合修剪，计算优化的LL(1)文法（寻求创新）
- 构建递归下降分析程序（对应于课程目标3中的设计语法分析框架之能力）
- 了解递归下降分析程序生成器（有挑战性）

6.2.4 LL(1)分析框架

- 确定性地模拟最左推导过程：
  - 若 $c\in\mathrm{FIRST}(\gamma)\vee(\varepsilon\in\mathrm{FIRST}(\gamma)\wedge c\in\mathrm{FOLLOW}(A))$则$xA\beta\Rightarrow_{\mathrm{lm}} x\gamma\beta$
  - 否则error()
- 性质6.1　CFG $G=(\mathbb{V},T,\mathcal{P},S)$是LL(1)文法，那么：
  1. $G=\bar{\mathrm{r}}\cdot G$;
  2. $\forall(A,\gamma),(A,\eta)\in\mathcal{P}\cdot\gamma=\eta\vee\mathrm{FIRST}(\gamma)\cap\mathrm{FIRST}(\eta)=\emptyset$;
  3. $\forall(A,\gamma)\in\mathcal{P}\cdot\varepsilon\in\mathrm{FIRST}(\gamma)\rightarrow\mathrm{FIRST}(A)\cap\mathrm{FOLLOW}(A)=\emptyset$。

? 不含左递归产生式，即$\forall(A,\alpha)\in\mathcal{P}\cdot\alpha\neq A\beta,\beta\in(\mathbb{V}\cup T)^*$

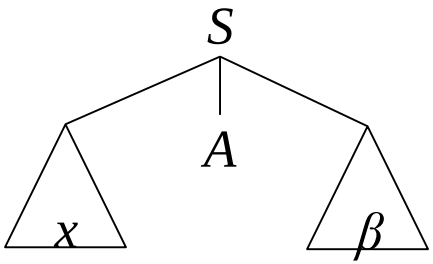
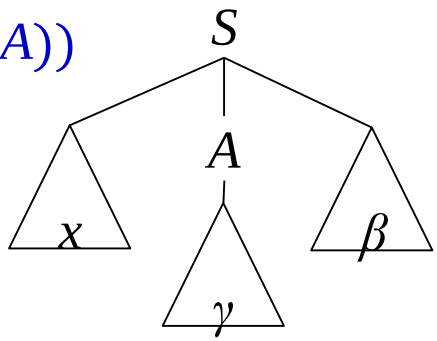
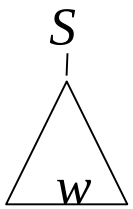
? 对于$A\longrightarrow\alpha_1\mid\alpha_2\mid\cdots\mid\alpha_n,n\geqslant1$，有$\mathrm{FIRST}(A)=\mathrm{U}1\leqslant i\leqslant\mathrm{n}\cdot\mathrm{FIRST}(\alpha_i)$

? $\varepsilon\in\mathrm{FIRST}(A)$当且仅当恰有一个$k$使得$\varepsilon\in\mathrm{FIRST}(\alpha_k)$

? $c\in\mathrm{FOLLOW}(A)$当且仅当$c\in T\wedge S\Rightarrow^*\gamma Ac\eta$

? $c\in\mathrm{FOLLOW}(A)$当且仅当$c\in T\wedge S\Rightarrow_{\mathrm{lm}}{}^*\,xAc\eta$

自上而下的语法分析过程示意

步骤	最左推导	语法树	待扩展变元	当前输入符号
初始化	S	S	S	$w[1]$
第i步	$S \Rightarrow_{lm}^{*} xA\beta$		A	$y[1]$, $w=xy$
第$i+1$步	$xA\beta \Rightarrow_{lm} x\gamma\beta$		$\gamma\beta$的最左变元	$y[1]$, $w=xy$
成功步	$S \Rightarrow_{lm}^{*} w$		无	#

若 $y[1]\in FIRST(\gamma)\vee(\varepsilon\in FIRST(\gamma)\wedge y[1]\in FOLLOW(A))$ 则$xA\beta\Rightarrow_{lm} x\gamma\beta$否则error()

课程思政点滴

- 输入串：一次成功的购买过程；一次成功的购物过程；一个合法的程序；个人一段成功的人生阶段。
- 初始符号：过程起点；我的起点
- 自上而下分析：基于全局视野的过程历程
- 两层不确定性决策：凡事都有不确定性、十字路口
- 从句型中多个变元出现选择一个：环境带给机遇
- 从该变元候选式中选择一个：自身的能力条件、机会是留给有准备的人
- 如何向成功迈进一步（直接推导）：比如靠正确的政策（党的政策）、有效的模型、方法、路线。
- 消除左递归、消除回溯：避免三心二意、决策错误、迷失在十字路口等
- 接受：最终，坚定不移地走出一条成功的路径。

消除文法中的直接左递归产生式

左递归产生式	消除左递归后	结果
$P \longrightarrow P\alpha \mid \beta$，其中$P$不是$\beta$的前缀	$P \longrightarrow \beta P' \quad P' \longrightarrow \alpha P' \mid \varepsilon$	√
$P \longrightarrow P \mid \beta$，其中$P$不是$\beta$的前缀	$P \longrightarrow \beta P' \quad P' \longrightarrow P' \mid \varepsilon$	×
$P \longrightarrow P\alpha \mid \varepsilon$	$P \longrightarrow P' \quad P' \longrightarrow \alpha P' \mid \varepsilon$	?
$P \longrightarrow P\alpha \mid \gamma P\beta$，其中$P$不是$\gamma$前缀但$\gamma$可空	$P \longrightarrow \gamma P \beta P' \quad P' \longrightarrow \alpha P' \mid \varepsilon$	×

? 可空串γ推导ε的每个直接推导有什么特点

$\gamma \Rightarrow \gamma_1 \Rightarrow \ldots \Rightarrow \gamma_i \Rightarrow \gamma_{i+1} \Rightarrow \ldots \Rightarrow \gamma_n = \varepsilon$

? 为了避免计算$\bar{\mathrm{r}} \cdot G$出现错误，要求文法G不含ε产生式和单位产生式，即满足$\bar{\mathrm{u}}\bar{\mathrm{e}}\, G = G$

? 如果文法G不是左递归文法，那么有$G = \bar{\mathrm{r}} \cdot G$

? 间接左递归如何消除

消除有回溯的产生式

- 若有$A \longrightarrow \alpha\beta_1 \mid \ldots \mid \alpha\beta_m \mid \eta_1 \mid \ldots \mid \eta_n$，其中$\alpha \neq \varepsilon$，$\eta_1, \ldots, \eta_n$都没有前缀$\alpha$，
- 那么将其修剪为：

$$A \longrightarrow \alpha A' \mid \eta_1 \mid \ldots \mid \eta_n$$

$$A' \longrightarrow \beta_1 \mid \ldots \mid \beta_m$$

- 一个变元的产生式（包含它的所有候选式）或者是回溯产生式，或者不是。
- 当$m \neq 0$时A的产生式是回溯产生式。
- 一般情况下，α是相对于前m个候选式的最大公共前缀，如果$\beta_1, \ldots, \beta_m$没有公共前缀
- α的一个非空前缀也是一个公共前缀。
- 当$n=0$时α是A的最大公共前缀。
- 对一个回溯产生式的修剪过程是迭代到任意两个候选式都没有公共前缀为止。
- 算法优化目标是迭代步数和新变元数量都最少。

回顾：修剪为LL(1)文法的过程

- 已知文法$G=(\mathbb{V}, T, \boldsymbol{\mathcal{P}}, S)$修剪为LL(1)文法的典型过程：
- **输入：G；输出：** $\mathrm{LL}(1)\cdot G$

  去除ε产生式，$L(\bar{\mathrm{e}}\cdot G)=L(G)$

  接着去除单位产生式，$L(\bar{\mathrm{u}}\cdot\bar{\mathrm{e}}\,G)=L(G)$

  接着去除无用符号，$L(ag\cdot\bar{\mathrm{u}}\bar{\mathrm{e}}\,G)=L(G)$

  接着消除文法左递归，$L(\bar{\mathrm{r}}\cdot ag\bar{\mathrm{u}}\bar{\mathrm{e}}\,G)=L(G)$

  接着去除回溯产生式，$L(\bar{\mathrm{c}}\cdot\bar{\mathrm{r}}ag\bar{\mathrm{u}}\bar{\mathrm{e}}\,G)=L(G)$

  消除文法歧义性，

  判断得到了LL(1)文法，或继续迭代 。

? $\mathrm{LL}(1)\cdot G$是采用复合p修剪完成，如何复合？

g（变元都是产出的）	$\bar{\mathrm{r}}$（不含左递归产生式）
a（文法符号都是可达的）	$\bar{\mathrm{c}}$（无回溯）
$\bar{\mathrm{e}}$（不含ε产生式）	$\bar{\mathrm{a}}$（无歧义性）
$\bar{\mathrm{u}}$（不含单位产生式）	

LL(1)分析框架

输入： LL(1)文法G。**输出：** 分析成功或失败。

初始化文法初始符号为当前待扩展节点；

初始化输入串的1前缀符号为当前输入符号；

重复以下步骤直至无可扩展变元时返回分析成功或者出错返回失败：

设当前输入符号为a，语法树当前待扩展节点为A，且有产生式规则$A \longrightarrow \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$，

(1) 如果 $a \in \text{FIRST}(\alpha_k)$则采用产生式$A \longrightarrow \alpha_k$扩展；

(2) 如果$a \notin \text{FIRST}(A)$，但是有$\varepsilon \in \text{FIRST}(\alpha_k)$且 $a \in \text{FOLLOW}(A)$，则采用规则$A \longrightarrow \alpha_k$扩展；

(3) 若是除上述以外情况，则a的出现是语法错误。

示例：消除给定文法的左递归和回溯的思路

$$P \longrightarrow \check{D}\ \check{S}$$
$$\check{D} \longrightarrow \varepsilon \mid D\ ;\ \check{D}$$
$$D \longrightarrow \text{int}\ d$$
$$\check{S} \longrightarrow S \mid \check{S}\ ;\ S$$
$$S \longrightarrow d = E \mid \text{print}\ d$$
$$E \longrightarrow i \mid d \mid E + i \mid E * d$$

- 2个左递归变元$\check{S}$和E
- E的2个候选式有公共前缀
- 消除左递归（先去除ε产生式？ 和单位产生式各1个）
- 提取同一变元的候选式的公共前缀
- 计算首符集和FOLLOW集
- 判断是否满足LL(1)文法的条件。

6.3 语法分析程序的构造

- LL(1)文法$(\mathbb{V}, T, \boldsymbol{\mathcal{P}}, S)$的分析程序是对LL(1)分析框架的一种程序实现，实现成每个变元对应一个递归子程序的形式。

- 下面通过例子展示如何进行程序实现。

已知文法CFG $G=G_t$:

$P \longrightarrow \check{D}\ \check{S}$

$\check{D} \longrightarrow \varepsilon \mid D\ ;\ \check{D}$

$D \longrightarrow \text{int}\ d$

$\check{S} \longrightarrow S \mid \check{S}\ ;\ S$

$S \longrightarrow d = E \mid \text{print}\ d$

$E \longrightarrow i \mid d \mid E + i \mid E * d$

已知一个句子:

```
int x;
int y;
x=1;
y=2*x+1;
print x;
print y
```

修剪为LL(1)文法的过程(1)

- 去除单位产生式：
- $\check{S} \longrightarrow d = E \mid \text{print}\, d \mid \check{S}\,; d = E \mid \check{S}\,; \text{print}\, d$
- 消除直接左递归：
- $\check{S} \longrightarrow d = E\, \check{S}' \mid \text{print}\, d\, \check{S}'$
- $\check{S}' \longrightarrow ; d = E\, \check{S}' \mid ; \text{print}\, d\, \check{S}' \mid \varepsilon$
- 消除回溯：
- $\check{S}' \longrightarrow ; R \mid \varepsilon$
- $R \longrightarrow d = E\, \check{S}' \mid \text{print}\, d\, \check{S}'$
- 消除左递归：
- $E \longrightarrow i\, E' \mid d\, E'$
- $E' \longrightarrow + i\, E' \mid * d\, E' \mid \varepsilon$
- 消除歧义性：（2和5候选式相同，放弃R，修改4为8）
- $\check{S}' \longrightarrow ; \check{S} \mid \varepsilon$

已知文法：

- $P \longrightarrow \check{D}\, \check{S}$
- $\check{D} \longrightarrow \varepsilon \mid D\,; \check{D}$
- $D \longrightarrow \text{int}\, d$
- $\check{S} \longrightarrow S \mid \check{S}\,; S$
- $S \longrightarrow d = E \mid \text{print}\, d$
- $E \longrightarrow i \mid d \mid E + i \mid E * d$

检查结果文法是否为LL(1)文法

- 结果文法：
- $P \longrightarrow \check{D}\ \check{S}$
- $\check{D} \longrightarrow \varepsilon \mid D\ ;\ \check{D}$
- $D \longrightarrow \text{int}\ d$
- $\check{S} \longrightarrow d = E\ \check{S}' \mid \text{print}\ d\ \check{S}'$
- $\check{S}' \longrightarrow ;\ \check{S} \mid \varepsilon$
- $E \longrightarrow i\ E' \mid d\ E'$
- $E' \longrightarrow +iE' \mid *dE' \mid \varepsilon$

- 首符集

P	{}	{int}	{int,*d*,print}
$\check{D}$	{ε}	{ε,int}	
D	{int}		
$\check{S}$	{*d*,print}		
$\check{S}'$	{;,ε}		
E	{*i*,*d*}		
E'	{+,*,ε}		

结果文法是LL(1)文法

- LL(1)文法：

$P \longrightarrow \check{D}\ \check{S}$

$\check{D} \longrightarrow \varepsilon \mid D\ ;\ \check{D}$

$D \longrightarrow \text{int}\ d$

$\check{S} \longrightarrow d = E\ \check{S}' \mid \text{print}\ d\ \check{S}'$

$\check{S}' \longrightarrow ;\ \check{S} \mid \varepsilon$

$E \longrightarrow i\ E' \mid d\ E'$

$E' \longrightarrow +\ i\ E' \mid *\ d\ E' \mid \varepsilon$

- 首符集

P	{int,d,print}
$\check{D}$	{ε,int}
D	{int}
$\check{S}$	{d,print}
$\check{S}'$	{;,ε}
E	{i,d}
E'	{+,*,ε}

- FOLLOW集

P	{#}	
$\check{D}$	{d,print}	
D	{;}	
$\check{S}$	{}	{#}
$\check{S}'$	{}	{#}
E	{;}	{;,#}
E'	{}	{;,#}

讨论：在解决具体问题时存在灵活性

- $\bar{\mathrm{e}}\cdot G$　　（考虑产生式分组独立处理，如ε产生式与左递归产生式不在一个组的情形）
- $\bar{\mathrm{u}}\cdot G$　　（单位产生式不会引起$P\Rightarrow^{+}P$）
- $g\cdot G$　　（）
- $a\cdot G$　　（考虑到消除左递归后出现不可达符号的情况）
- $\bar{\bar{\mathrm{r}}}\cdot G$　　（考虑到再消除ε产生式又导致左递归的情况，所以消除左递归给文法带来新的ε产生式）
- $\bar{\mathrm{c}}\cdot G$　　（对每个变元，通过提取候选式公共前缀并引入新变元的办法合并它们，这些候选式除去公共前缀余下的部分都作为新变元的候选式。这次修剪是在G_t上进行，直到达到LL(1)文法条件。注意到一些新变元是琐碎的。）
- 文法修剪过程中也可能出现重复的候选式，可以去除重复。
- 总之，建议以求出LL(1)文法为目标灵活运用。

将文法G_t划分为3个子文法

$$G_t = G = G^0 \cup G^1 \cup G^2$$

$$
\begin{aligned}
&P \longrightarrow \check{D}\ \check{S} \\
&\check{D} \longrightarrow \varepsilon \mid D\ ;\ \check{D} \\
&D \longrightarrow \text{int}\ d \\
&\check{S} \longrightarrow S \mid \check{S}\ ;\ S \\
&S \longrightarrow d = E \mid \text{print}\ d \\
&E \longrightarrow i \mid d \mid E + i \mid E * d
\end{aligned}
$$

G^0：$P \longrightarrow \check{D}\check{S}$　$\check{D} \longrightarrow \varepsilon \mid D;\ \check{D}$　$D \longrightarrow \text{int}\ d$

G^1：$\check{S} \longrightarrow S \mid \check{S};\ S$　$S \longrightarrow d = E \mid \text{print}\ d$

G^2：$E \longrightarrow i \mid d \mid E+i \mid E*d$

修剪方案（0）

G^0：$P \longrightarrow \check{D}\check{S}$　$\check{D} \longrightarrow \varepsilon \mid D;\ \check{D}$　$D \longrightarrow \text{int}\ d$

G^1：$\check{S} \longrightarrow S \mid \check{S};\ S$　$S \longrightarrow d = E \mid \text{print}\ d$

G^2：$E \longrightarrow i \mid d \mid E+i \mid E*d$

▻ 对G^1去除单位产生式$\check{S} \longrightarrow S$，用$S$代入，得到

$a\bar{u} \cdot G^1$：$\check{S} \longrightarrow d = E \mid \text{print}\ d \mid \check{S};\ S$

▻ 接着消除$\check{S}$的直接左递归得到

$\bar{r}a\bar{u} \cdot G^1$：$\check{S} \longrightarrow d = E\check{S}' \mid \text{print}\ d\check{S}'$　$\check{S}' \longrightarrow;\ S\check{S}' \mid \varepsilon$

▻ 得到结果子文法

$\check{S} \longrightarrow d = E\check{S}' \mid \text{print}\ d\check{S}'$　$\check{S}' \longrightarrow;\ S\check{S}' \mid \varepsilon$　$S \longrightarrow d = E \mid \text{print}\ d$

（0）

G^0：$P \longrightarrow \check{D}\check{S}$ $\check{D} \longrightarrow \varepsilon \mid D;\ \check{D}$ $D \longrightarrow \text{int}\ d$

$\bar{r}a\bar{u}G^1$：$\check{S} \longrightarrow d = E\check{S}' \mid \text{print}\ d\check{S}'$ $\check{S}' \longrightarrow ;\ S\check{S}' \mid \varepsilon$ $S \longrightarrow d = E \mid \text{print}\ d$

G^2：$E \longrightarrow i \mid d \mid E{+}i \mid E{*}d$

- 接着对G^2消除左递归得到

  $\bar{r} \cdot G^2$：$E \longrightarrow iE' \mid dE'$ $E' \longrightarrow {+}iE' \mid {*}dE' \mid \varepsilon$

- 由性质6.1知道$G^0 \cup \bar{r}a\bar{u}G^1 \cup \bar{r}G^2$是LL(1)文法

$P \longrightarrow \check{D}\check{S}$ $\check{D} \longrightarrow \varepsilon \mid D;\ \check{D}$ $D \longrightarrow \text{int}\ d$

$\check{S} \longrightarrow d = E\check{S}' \mid \text{print}\ d\check{S}'$ $\check{S}' \longrightarrow ;\ S\check{S}' \mid \varepsilon$ $S \longrightarrow d = E \mid \text{print}\ d$

$E \longrightarrow iE' \mid dE'$ $E' \longrightarrow {+}iE' \mid {*}dE' \mid \varepsilon$

（0）是LL(1)文法

$P \longrightarrow \check{D}\check{S}$	P	{int,d,print}	{#}
$\check{D} \longrightarrow \varepsilon \mid D;\ \check{D}$	$\check{D}$	{int,ε}	{d,print}
$D \longrightarrow \text{int}\ d$	D	{int}	{;}
$\check{S} \longrightarrow d = E\check{S}' \mid \text{print}\ d\check{S}'$	$\check{S}$	{d,print}	{#}
$\check{S}' \longrightarrow\ ;\ S\check{S}' \mid \varepsilon$	$\check{S}'$	{;,ε}	{#}
$S \longrightarrow d = E \mid \text{print}\ d$	S	{d,print}	{;,#}
$E \longrightarrow iE' \mid dE'$	E	{i,d}	{;,#}
$E' \longrightarrow +iE' \mid *dE' \mid \varepsilon$	E'	{+,*,ε}	{;,#}

修剪方案（2）

G：$P\longrightarrow \check{D}\check{S}$ $\check{D}\longrightarrow \varepsilon \mid D;\check{D}$ $D\longrightarrow \text{int}\,d$ $\check{S}\longrightarrow S \mid \check{S};S$ $S\longrightarrow d=E \mid \text{print}\,d$

$E\longrightarrow i \mid d \mid E+i \mid E*d$

- $\bar{e}G$：$P\longrightarrow \check{D}\check{S} \mid \check{S}$ $\check{D}\longrightarrow D;\check{D} \mid D;$
- 接着去除单位产生式$P\longrightarrow \check{S}$和$\check{S}\longrightarrow S$，并消除$\check{S}$和$E$的左递归产生式得到$\bar{r}\bar{u}\bar{e}G$：

$P\longrightarrow \check{D}\check{S} \mid d=E \mid \text{print}\,d \mid \check{S};S$ $\check{D}\longrightarrow D;\check{D} \mid D;$ $D\longrightarrow \text{int}\,d$

$\check{S}\longrightarrow d=E\check{S}' \mid \text{print}\,d\check{S}'$ $\check{S}'\longrightarrow ;S\check{S}' \mid \varepsilon$ $S\longrightarrow d=E \mid \text{print}\,d$

$E\longrightarrow iE' \mid dE'$ $E'\longrightarrow +iE' \mid *dE' \mid \varepsilon$

- 接着消除歧义性，即S与P有公共候选式，对P的公共候选式用S替代得到$P\longrightarrow \check{D}\check{S}\mid S\mid \check{S};S$从而得到$\bar{a}\bar{r}\bar{u}\bar{e}G$。至此检查之发现不是LL(1)文法，那么，这条线路期待更多p修剪。

（2）检查不满足LL(1)条件

$P \longrightarrow \check{D}\,\check{S} \mid S \mid \check{S}\,;\,S$	{d,print,int} {#}
$\check{D} \longrightarrow D\,;\,\check{D} \mid D\,;$	{int} {d,print}
$D \longrightarrow \text{int}\,d$	{int} {;}
$\check{S} \longrightarrow d = E\,\check{S}' \mid \text{print}\,d\,\check{S}'$	{d,print} {;,#}
$\check{S}' \longrightarrow ;\,S\,\check{S}' \mid \varepsilon$	{;,ε} {;,#}
$S \longrightarrow d = E \mid \text{print}\,d$	{d,print} {;,#}
$E \longrightarrow i\,E' \mid d\,E'$	{i,d} {;,#}
$E' \longrightarrow + i\,E' \mid * d\,E' \mid \varepsilon$	{+,*,ε} {;,#}

修剪方案（3）

G^0：$P \longrightarrow \check{D}\check{S}\ \ \check{D} \longrightarrow \varepsilon \mid D;\ \check{D}\ \ D \longrightarrow \text{int}\ d$

G^1：$\check{S} \longrightarrow S \mid \check{S};\ S\ \ S \longrightarrow d = E \mid \text{print}\ d$

G^2：$E \longrightarrow i \mid d \mid E{+}i \mid E{*}d$

- 对G^0去除ε产生式得到$\bar{\text{e}} \cdot G^0$：$P \longrightarrow \check{D}\check{S} \mid \check{S}\ \ \check{D} \longrightarrow D;\ \check{D} \mid D;\ \ D \longrightarrow \text{int}\ d$
- 对G^1和G^2分别消除左递归得

$\bar{\text{r}} \cdot G^1$：$\check{S} \longrightarrow S\check{S}'\ \ \check{S}' \longrightarrow;\ S\check{S}' \mid \varepsilon\ \ S \longrightarrow d = E \mid \text{print}\ d$

$\bar{\text{r}} \cdot G^2$：$E \longrightarrow iE' \mid dE'\ \ E' \longrightarrow +iE' \mid *dE' \mid \varepsilon$

- 对$\bar{\text{e}}G^0$消除回溯得到$\bar{\text{c}} \cdot \bar{\text{e}}G^0$：

$P \longrightarrow \check{D}\check{S} \mid \check{S}\ \ \check{D} \longrightarrow D;\ D'\ \ D' \longrightarrow \check{D} \mid \varepsilon\ \ D \longrightarrow \text{int}\ d$

- 至此得到满足LL(1)文法性质的$\bar{\text{c}}\bar{\text{e}}G^0 \cup \bar{\text{r}}G^1 \cup \bar{\text{r}}G^2$。这条路线是常见的规范的复合修剪。（未消除单位产生式是因为S不可空）

（3）

G^0：$P \longrightarrow \check{D}\check{S}\ \ \check{D} \longrightarrow \varepsilon \mid D;\ \check{D}\ \ D \longrightarrow \text{int}\ d$

G^1：$\check{S} \longrightarrow S \mid \check{S};\ S\ \ S \longrightarrow d = E \mid \text{print}\ d$

G^2：$E \longrightarrow i \mid d \mid E+i \mid E*d$

- $\bar{c}\bar{e}G^0 \cup \bar{r}G^1 \cup \bar{r}G^2$

$P \longrightarrow \check{D}\check{S} \mid \check{S}\ \ \check{D} \longrightarrow D;\ D'\ \ D' \longrightarrow \check{D} \mid \varepsilon\ \ D \longrightarrow \text{int}\ d$

$\check{S} \longrightarrow S\check{S}'\ \ \check{S}' \longrightarrow ;\ S\check{S}' \mid \varepsilon\ \ S \longrightarrow d = E \mid \text{print}\ d$

$E \longrightarrow iE' \mid dE'\ \ E' \longrightarrow +iE' \mid *dE' \mid \varepsilon$

（3）检查是LL(1)文法

$P \longrightarrow \check{D}\ \check{S} \mid \check{S}$	{int,*d*,print}	{#}
$\check{D} \longrightarrow D\ ;\ D'$	{int}	{*d*,print}
$D' \longrightarrow \check{D} \mid \varepsilon$	{int,ε}	{*d*,print}
$D \longrightarrow \text{int}\ d$	{int}	{;}
$\check{S} \longrightarrow S\ \check{S}'$	{*d*,print}	{#}
$\check{S}' \longrightarrow ;\ S\ \check{S}' \mid \varepsilon$	{;,ε}	{#}
$S \longrightarrow d = E \mid \text{print}\ d$	{*d*,print}	{;,#}
$E \longrightarrow i\ E' \mid d\ E'$	{*i*,*d*}	{;,#}
$E' \longrightarrow +\ i\ E' \mid *\ d\ E' \mid \varepsilon$	{+,*,ε}	{;,#}

修剪方案（4）容忍ε产生式因与$\bar{r}G$无关

G^0：$P \longrightarrow \check{D}\check{S}\ \ \check{D} \longrightarrow \varepsilon \mid D;\ \check{D}\ \ D \longrightarrow \text{int}\ d$

G^1：$\check{S} \longrightarrow S \mid \check{S};\ S\ \ S \longrightarrow d = E \mid \text{print}\ d$

G^2：$E \longrightarrow i \mid d \mid E+i \mid E*d$

▻ 消除$\check{S}$和E的左递归得$\bar{r}G$，并且是LL(1)文法（未消除单位产生式是因为S不可空）：

$P \longrightarrow \check{D}\check{S}\ \ \check{D} \longrightarrow \varepsilon \mid D;\ \check{D}\ \ D \longrightarrow \text{int}\ d$

$\check{S} \longrightarrow S\check{S}'\ \ \check{S}' \longrightarrow ;\ S\check{S}' \mid \varepsilon\ \ S \longrightarrow d = E \mid \text{print}\ d$

$E \longrightarrow iE' \mid dE'\ \ E' \longrightarrow +iE' \mid *dE' \mid \varepsilon$

（4）是LL(1)文法

$P \longrightarrow \check{D}\ \check{S}$
$\check{D} \longrightarrow \varepsilon \mid D\ ;\ \check{D}$
$D \longrightarrow \text{int}\ d$
$\check{S} \longrightarrow S\ \check{S}'$
$\check{S}' \longrightarrow ;\ S\ \check{S}' \mid \varepsilon$
$S \longrightarrow d = E \mid \text{print}\ d$
$E \longrightarrow i\ E' \mid d\ E'$
$E' \longrightarrow +\ i\ E' \mid *\ d\ E' \mid \varepsilon$

P	{int,d,print}	{#}
$\check{D}$	{int,ε}	{d,print}
D	{int}	{;}
$\check{S}$	{d,print}	{#}
$\check{S}'$	{;,ε}	{#}
S	{d,print}	{;,#}
E	{i,d}	{;,#}
E'	{+,*,ε}	{;,#}

评价复合p修剪（创新性）

(1) $G^0 \cup \bar{a}\bar{c}\bar{r}a\bar{u}G^1 \cup \bar{r}G^2$或$\bar{a}\bar{c}\bar{r}a\bar{u}G$

(0) $G^0 \cup \bar{r}a\bar{u}G^1 \cup \bar{r}G^2$

(3) $\bar{c}\bar{e}G^0 \cup \bar{r}G^1 \cup \bar{r}G^2$

(4) $\bar{r}G$

文法修剪方案	（1）	（0）	（3）	（4）
变元数目	7	8	9	8
候选式数目	13	15	16	14
候选式平均长度	2.23	4.63	2	2.14
候选式总长度	29	37	32	30
候选式最大长度	4	4	3	3
计算复杂度	高	中	中	低

例：递归下降分析程序构造

- scan()返回词法单位类别：
- NUM、ID、SCO、MUL、ADD、ASG、INT、PRINT
- int tok;　　//当前token

```
int scan(){
    tok=scanner();
}

void main(){
    scan();
    Prog();    // P();
}
```

LL(1)文法：

$P \longrightarrow \check{D}\check{S}$

$\check{D} \longrightarrow \varepsilon \mid D;\check{D}$

$D \longrightarrow \text{int}\ d$

$\check{S} \longrightarrow d = E\check{S}' \mid \text{print}\ d\check{S}'$

$\check{S}' \longrightarrow ;\check{S} \mid \varepsilon$

$E \longrightarrow iE' \mid dE'$

$E' \longrightarrow +iE' \mid *dE' \mid \varepsilon$

例：递归下降分析程序构造

- $P \longrightarrow \check{D}\ \check{S}$ {int,*d*,print} {#}

```
int P() {if(tok∈(INT ID PRINT)){Dlist();Slist();}else error();}
int Prog(){
        if(tok==INT || tok==ID || tok==PRINT){Dlist();Slist();}
        else error();}
```

- $\check{D} \longrightarrow \varepsilon \mid D\ ;\ \check{D}$ {ε,int} {*d*,print}

```
int Dlist()      {
    if(tok∈(INT)){D();match(SCO);Dlist();}
        else if(tok∉(ID PRINT))error();}
int Dlist(){
        if(tok==INT){
                Decl();if(tok==SCO){scan();Dlist();}else error()
        }else if(tok!=ID&&tok!=PRINT)error();}
```

例：递归下降分析程序构造

- $D \longrightarrow \text{int}\ d$ {int} {;}

```
int D() {if(tok ∈(INT)){match(INT);match(ID);}else error();}
int Decl(){if(tok==INT){scan();
        if(tok==ID)scan()else error();}else error();}
```

- $\check{S} \longrightarrow d = E\ \check{S}' \mid \text{print}\ d\ \check{S}'$ {d,print} {#}

```
int Slist()      {
        if(tok∈(ID)){match(ID);match(ASG);E();Slp();}
        else if(tok∈(PRINT)){match(PRINT);match(ID);Slp();}
        else error();}
int Slist(){
        if(tok==ID){scan();if(tok==ASG){scan();Exp();Slp();}}
        else if(tok==PRINT){scan();if(tok==ID){scan();Slp();}}
        else error();}
```

例：递归下降分析程序构造

- $\check{S}' \longrightarrow ; \check{S} \mid \varepsilon$ {;,ε} {#}

```
int Slp()       {
        if(tok∈(SCO)){match(SCO);Slist();}
        else if(tok ∉(EOF))error();}
int Slp(){if(tok==SCO){scan();Slist();}
                else if(tok!=EOF)error();}}
```

- $E \longrightarrow i\ E' \mid d\ E'$ {i,d} {;,#}

```
int E()         {
        if(tok∈(NUM)){match(NUM);Ep();}
        else if(tok∈(ID)){match(ID);Ep();}else  error();}
int Exp(){if(tok==NUM){scan();Ep();}
        else if(tok==ID){scan();Ep();}else error();}
```

例：递归下降分析程序构造

- $E' \longrightarrow + i E' \mid * d E' \mid \varepsilon$ $\quad\quad$ {+,*,ε} $\quad\quad$ {;,#}

```
int Ep()        {
        if(tok ∈(ADD)){match(ADD);match(NUM);Ep();}
        else if(tok ∈(MUL)){match(MUL);match(ID);Ep();}
        else if(tok∉(SCO EOF))error();}
```

```
int Ep(){
        if(tok==ADD){scan();
                if(tok==NUM){scan();Ep();}else error();}
        else if(tok==MUL){scan();
                if(tok==ID){scan();Ep();}else error();}
        else if(tok!=SCO&&tok!=EOF)error();}
```

递归下降分析程序生成器（挑战度H）

▻ 输入：LL(1)文法(𝕍, T, $\mathcal{P}$, S)；输出：code，即分析程序

```
code=[]; tailc=[];
for(A∈𝕍){code++=gen[int ?A () {];
 for((A, α)∈𝒫){if(α!=NIL){code++=gen[if(tok∈?FIRST(α)){];
  C=α; while(C!=NIL){X=car(C); C= cdr(C);
   if(X∈T)code++=gen[match(?X);]else code++=gen[?X ();];}
  code++=gen[}else]}else if(ε∈FIRST(α)){
   if(α!=NIL){tailc++=gen[if(tok∈?FOLLOW(A)){];
    C=α; while(C!=NIL){X=car(C); C=cdr(C);
     if(X∈T)tailc++=gen[match(?X);]else tailc++=gen[?X ();];}
   }else tailc=gen[if(tok∉?FOLLOW(A))error();];
   }else tailc=gen[error()];}          //end of 2st for
 code++=tailc++gen[}];}                //end of 1st for
```

下一步：输出给定程序的语法树

```
int x;
int y;
x=1;
y=2*x+1;
print x;
print y
```

$P \longrightarrow \check{D}\check{S}$

$\check{D} \longrightarrow \varepsilon \mid D;\ \check{D}$

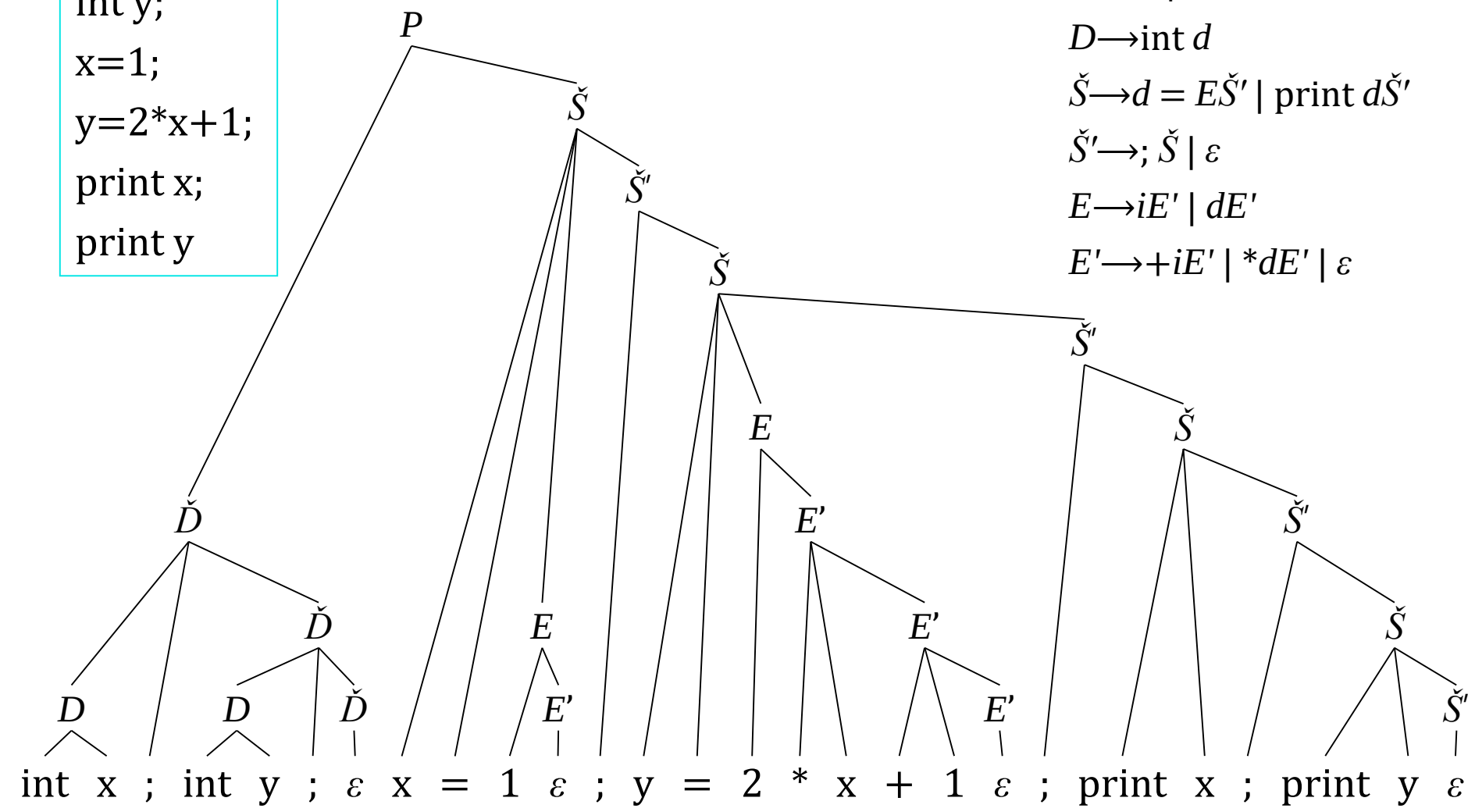
$D \longrightarrow \text{int}\ d$

$\check{S} \longrightarrow d = E\check{S}' \mid \text{print}\ d\check{S}'$

$\check{S}' \longrightarrow ;\ \check{S} \mid \varepsilon$

$E \longrightarrow iE' \mid dE'$

$E' \longrightarrow +iE' \mid *dE' \mid \varepsilon$



例：生成语法树

- struct node {int name; struct node *laoda; struct node *laoer; struct node *laosan; struct node *laosi; }
- #define P 1001; #define DLP 1010; #define DL 1002; #define D 1003; #define SL 1004; #define SLP 1005; #define E 1006; #define EP 1007;
- int tok; int scan(){tok=scanner();}
- node* tree;
- node *mknode(name){
  t=new(node); t.name=name; t.laoda=NULL; return t;}
- void main(){
- scan();
- tree=mknode(P);
- Prog(tree);
- output(tree);}　　//输出形式建议采用嵌套表示形式

例：生成语法树

- $P \longrightarrow \check{D}\ \check{S}$ {int,*d*,print} {#}

```
int Prog(tree){
   if(tok==INT || tok==ID || tok==PRINT){
        t=mknode(DL);
        tree.laoda=t;
        Dlist(t);
        t=mknode(SL);
        tree.laoer=t;
        Slist(t);
   }else error();
}
```

例：生成语法树

- $\check{D} \longrightarrow \varepsilon \mid D\,;\check{D}$ {ε,int} {d,print}

```
int Dlist(tree){
   if(tok==INT){
        t=mknode(D); tree.laoda=t; Decl(t);
        if(tok==SCO){
             tree.laoer=mknode(tok); scan();
             t=mknode(DL); tree.laosan=t; Dlist(t);
        }else if (tok ==ID || tok==PRINT){
             tree.laoda=mknode(EPSILON);
        }else error();
   }else error();
}
```

例：生成语法树

- $D \longrightarrow \text{int}\ d$ {int} {;}

```
int Decl(tree){
   if(token==INT){
        tree.laoda=mknode(INT);
        scan();
        if(token==ID){
             tree.laoer=mknode(ID);
             scan();
        }else error();
   }else error();
}
```

例：生成语法树

- $\check{S} \longrightarrow d = E\ \check{S}' \mid \text{print}\ d\ \check{S}'$ $\{d,\text{print}\}$ $\{\#\}$

```
int Slist(tree){
 if(tok==ID){tree.laoda=mknode(tok); scan();
        if(tok==ASG){tree.laoer=mknode(tok); scan();
        t=maknode(E); tree.laosan=t; Exp(t);
        t=mknode(SLP); tree.laosi=t; Slp(t);
        }else error();
 }else if(tok==PRINT){tree.laoda=mknode(tok); scan();
           if(tok==ID){tree.laoer=mknode(tok); scan();
                 t=maknode(SLP); tree.laosan=t; Slp(t);
                 }else error();
        }else error();
}
```

例：生成语法树

- $S' \longrightarrow ; S \mid \varepsilon$ {;,ε} {#}

```
int Slp(tree){
   if(tok==SCO){
        tree.laoda=mknode(SCO);
        scan();
        t=mknode(SL);
        tree.laoer=t; Slist(t);
   }else if(token==EOF){
        tree.laoda=mknode(EPSILON);
        }else error();
}
```

例：生成语法树

- $E \longrightarrow i\ E' \mid d\ E'$ $\{i,d\}$ $\{;,\#\}$

```
int Exp(tree){if(tok==NUM || tok==ID){
 tree.laoda=mknode(tok); scan(); t=mknode(EP);
 tree.laoer=t; Ep(t);}else error();}
```

- $E' \longrightarrow + i\ E' \mid * d\ E' \mid \varepsilon$ $\{+,*,\varepsilon\}$ $\{;,\#\}$

```
int Ep(tree){if(tok==ADD){tree.laoda=mknode(tok);scan();
if(tok==NUM){tree.laoer=mknode(tok);scan(); t=mknode(EP);
tree.laosan=t; Ep(t);}else error();}else
if(tok==MUL){tree.laoda=mknode(tok);scan();
if(tok==ID){tree.laoer=mknode(tok);scan();t=mknode(EP);
tree.laosan=t; Ep(t);}else if(tok==SCO || tok==EOF){
tree.laoda=mknode(EPSILON);}else error();}
```

小结：递归下降分析程序

- 是对LL(1)分析框架的一种实现。
- 在其中调用一次词法分析器来返回一个词法记号。
- 每一个变元（非终结符）对应于一个递归函数；
  - 在每个函数中，根据当前符号和变元的首符集、FOLLOW集决定所用候选式
  - 若当前符号属于所用候选式的首符集，那么对于候选式逐符号依次进行处理：遇到终结符则匹配掉（失败即出错）；遇变元则调用对应递归函数；处理完返回。
  - 若首符集有ε则当前符号为FOLLOW集元素时应用首符集含ε的候选式，随后同上一步处理。
  - 对于其它情况为出错。
- 如果没有错误就确定性地按照LL(1)分析过程进行下去，直到所有递归函数都返回就表示分析成功。
- 可在该框架上添加代码来完成语法分析引导的任务。

6.3.2 预测分析程序

- 是对LL(1)分析框架的另一种实现。
- 将LL(1)分析的条件表示为预测分析表
- 采用预测分析表和栈实现自上而下的语法分析程序
- 这种方法实现的语法分析程序又叫预测分析程序

预测分析表M[A, a]

- 当前待扩展变元为A，当前输入符号为a时，采用M[A, a]元素作为候选式进行扩展。因为M[A, a]满足：
  - 如果 a 在FIRST(α_k)中则M[A, a]=α_k;
  - 如果 a 不在FIRST(A)中，但是有某个k有ε∈FIRST(α_k)且a∈FOLLOW(A)，则M[A, a]=α_k。
  - 其它情况为语法错误。

- 在最左推导或最左扩展语法树过程中，根据当前待扩展变元A和当前输入符号a，查M[A, a]得出可应用的候选式。如此重复进行直到给定句子的推导或语法树形成为止，出错除外。

例：预测分析表构建

文法：

$E \longrightarrow TE'$

$E' \longrightarrow +TE' \mid -TE' \mid \varepsilon$

$T \longrightarrow FT'$

$T' \longrightarrow *FT' \mid /FT' \mid \varepsilon$

$F \longrightarrow (E) \mid i$

首符集：

E {(,i}

E' {+,-,ε}

T {(,i}

T' {*,/,ε}

F {(,i}

FOLLOW集：

E {#,)}

E' {#,)}

T {+,-,#,)}

T' {+,-,#,)}

F {*,/,+,-,#,)}

	+	-	*	/	(	)	i	#
E					TE'		TE'	
E'	$+TE'$	$-TE'$				ε		ε
T					FT'		FT'	
T'	ε	ε	$*FT'$	$/FT'$		ε		ε
F					(E)		i	

栈	输入串	动作
#E	2-2*2#	E⟶TE'
#E'T	2-2*2#	T⟶FT'
#E'T'F	2-2*2#	F⟶i
#E'T'i	2-2*2#	match
#E'T'	-2*2#	T'⟶ε
#E'	-2*2#	E'⟶-TE'
#E'T-	-2*2#	match
#E'T	2*2#	T⟶FT'
#E'T'F	2*2#	F⟶i
#E'T'i	2*2#	match
#E'T'	*2#	T'⟶*FT'
#E'T'F*	*2#	match
#E'T'F	2#	F⟶i
#E'T'i	2#	match
#E'T'	#	T'⟶ε
#E'	#	E'⟶ε
#	#	accept

	+	-	*	/	(	)	i	#
E					TE'		TE'	
E'	+TE'	-TE'				ε		ε
T					FT'		FT'	
T'	ε	ε	*FT'	/FT'		ε		ε
F					(E)		i	

```
push(#); //使用一个全局栈
push(E); scan();
while(1){  X=pop();
 if(X==#)if(X==tok)break;
                    else error();
 if(X∈T){if(X==tok)scan();
                    else error();
 }else{
   α=M[X,tok];
   if((X,α)∈P)      //是产生式
     pushlist(α);   //反序进栈
   else error();
 }else error();}   //end while分析成功
```

例：G_t的预测分析表

P⟶ĎŠ	{int,d,print}	{#}
Ď⟶ε | D;Ď	{int,ε}	{d,print}
D⟶int d	{int}	{;}
Š⟶d=EŠ' | print dŠ'	{d,print}	{#}
Š'⟶;Š | ε	{;,ε}	{#}
E⟶iE' | dE'	{i,d}	{;,#}
E'⟶+iE' | *dE' | ε	{+,*,ε}	{;,#}

	+	*	i	d	;	int	print	=	#
P				ĎŠ		ĎŠ	ĎŠ		
Ď				ε		D;Ď	ε		
D						int d			
Š				d=EŠ'			print dŠ'		
Š'					;Š				ε
E			iE'	dE'					
E'	+iE'	*dE'			ε				ε

例：语法分析过程

栈	输入串	动作
#P	int x; x=2*x+1; print x#	P⟶ĎŠ
#ŠĎ	int x; x=2*x+1; print x#	Ď⟶D;Ď
#ŠĎ;D	int x; x=2*x+1; print x#	D⟶int d
#ŠĎ;d int	int x; x=2*x+1; print x#	匹配
#ŠĎ;d	x; x=2*x+1; print x#	匹配
#ŠĎ;	; x=2*x+1; print x#	匹配
#ŠĎ	x=2*x+1; print x#	Ď⟶ε
#Š	x=2*x+1; print x#	Š⟶d=EŠ’
#Š’E=d	x=2*x+1; print x#	匹配

	+	*	i	d	;	int	print	=	#
P				ĎŠ		ĎŠ	ĎŠ		
Ď				ε		D;Ď	ε		
D						int d			
Š				d=EŠ’			print dŠ’		
Š’					;Š				ε
E			iE’	dE’					
E’	+iE’	*dE’			ε				ε

分析过程（续）

栈	输入串	动作
#Š'E=	=2*x+1; print x#	匹配
#Š'E	2*x+1; print x#	E⟶iE'
#Š'E'i	2*x+1; print x#	匹配
#Š'E'	*x+1; print x#	E' ⟶*dE'
#Š'E'd*	*x+1; print x#	匹配
#Š'E'd	x+1; print x#	匹配
#Š'E'	+1; print x#	E' ⟶+iE'
#Š'E'i+	+1; print x#	匹配
#Š'E'i	1; print x#	匹配
#Š'E'	; print x#	E' ⟶ε

	+	*	i	d	;	int	print	=	#
P				ĎŠ		ĎŠ	ĎŠ		
Ď				ε		D;Ď	ε		
D						int d			
Š				d=EŠ'			print dŠ'		
Š'					;Š				ε
E			iE'	dE'					
E'	+iE'	*dE'			ε				ε

分析过程（续）

栈	输入串	动作
#Š’	; print x#	Š’⟶;Š
#Š;	; print x#	匹配
#Š	print x#	Š⟶print d Š’
#Š’d print	print x#	匹配
#Š’d	x#	匹配
#Š’	#	Š’⟶ε
#	#	接受

	+	*	i	d	;	int	print	=	#
P				ĎŠ		ĎŠ	ĎŠ		
Ď				ε		D;Ď	ε		
D						int d			
Š				d=EŠ’			print dŠ’		
Š’					;Š				ε
E			iE’	dE’					
E’	+iE’	*dE’			ε				ε

例：对if语句的处理

S⟶I | other	{if,other}	{#,else}
I⟶if(E)SL	{if}	{#,else}
L⟶else S | ε	{else,ε}	{#,else}
E⟶0 | 1	{0,1}	{)}

	if	other	else	0	1	#
S	S⟶I	S⟶other				
I	I⟶if(E)SL					
L			L⟶else S L⟶ε			L⟶ε
E				E⟶0	E⟶1	

Parsing Stack	Input	Action
#S	if(0)if(1)other else other#	S⟶I
#I	if(0)if(1)other else other#	I⟶if(E)SL
#LS)E(if	if(0)if(1)other else other#	match
# LS)E(	(0)if(1)other else other#	match
# LS)E	0)if(1)other else other#	E⟶0
# LS)0	0)if(1)other else other#	match
# LS)	)if(1)other else other#	match
# LS	if(1)other else other#	S⟶I
# LI	if(1)other else other#	I⟶if(E)SL
#LLS)E(if	if(1)other else other#	match
#LLS)E(	(1)other else other#	match
#LLS)E	1)other else other#	E⟶1
#LLS)1	1)other else other#	match
#LLS)	)other else other#	match
#LLS	other else other#	S⟶other
#LLother	other else other#	match
#LL	else other#	L⟶else S
#LSelse	else other#	match
#LS	other#	S⟶other
#Lother	other#	match
#L	#	L⟶ε
#	#	accept

小结：预测分析程序

- 预测分析程序对应于最左推导，已消耗串与栈内容连接成左句型。栈内容符号串的前缀对应于栈顶方向。
- 初始时栈底是#，栈顶是初始符号；当前输入符号为a；
- 分析表：二维数组M[A, a],$a \in T$,$A \in V$，元素为可用候选式。
- 重复进行如下动作：
  - 栈顶为变元A：弹出并压入M[A, a]诸符号；
  - 栈顶为终结符：弹出并与a进行匹配；
  - 栈顶为#且a=#，则分析过程终止，分析成功。
- 出错的情形
  - 终结符不匹配；
  - 表项M[A, a]是产生式以外情况。

自上而下分析中的错误处理

- 尽快找出错误；
- 一旦发现错误后，还能够继续分析下去；
- 尽量减少一个错误所导致的错误（error cascade problem）；
- 避免在错误上死循环。

- 出现错误的情形：
- 栈顶终结符与当前输入Token不匹配；
- 栈顶为变元A，当前输入Token为a，但M[A,a]为空。

错误处理的思想

- 栈顶为终结符
  - 弹出
- 栈顶为变元
  - 跳过输入Token流直至遇到同步符号集元素出现
- 同步符号集
  - FOLLOW(*A*)
  - FIRST(*A*)

小结

- 掌握消除左递归、消除回溯、LL(1)文法
- 掌握计算首符集、FOLLOW集
- 掌握LL(1)分析框架
- 掌握构造预测分析表
- 自上而下分析程序

作业

- 习题6.2；
- 大作业（二）：对下列文法，分别计算每个变元的FIRST集和FOLLOW集，然后从该文法中找出不满足LL(1)文法条件的各个原因。

$P \longrightarrow \check{D}\ \check{S}$

$\check{D} \longrightarrow \varepsilon \mid \check{D}\ D\ ;$

$D \longrightarrow T\ d \mid T\ d[i] \mid T\ d(\check{A})\ \{\check{D}\ \check{S}\}$

$T \longrightarrow \text{int} \mid \text{void}$

$\check{A} \longrightarrow \varepsilon \mid \check{A}\ A\ ;$

$A \longrightarrow T\ d \mid d[] \mid T\ d()$

$\check{S} \longrightarrow S \mid \check{S}\ ;\ S$

$S \longrightarrow d = E \mid \text{if}\ (B)\ S \mid \text{if}\ (B)\ S\ \text{else}\ S \mid \text{while}\ (B)\ S \mid \text{return}\ E \mid \{\check{S}\} \mid d(\check{R})$

$B \longrightarrow B \wedge B \mid B \vee B \mid E\ r\ E \mid E$

$E \longrightarrow d = E \mid i \mid d \mid d(\check{R}) \mid E + E \mid E * E \mid (E)$

$\check{R} \longrightarrow \varepsilon \mid \check{R}\ R\ ,$

$R \longrightarrow E \mid d[] \mid d()$