中山大學
SUN YAT-SEN UNIVERSITY

国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

DCS290

Compilation Principle

编 译 原 理

第四章 语法分析（1）

郑馥丹
zhengfd5@mail.sysu.edu.cn

CONTENTS

目 录

01 自顶向下分析 Top-Down Parsing

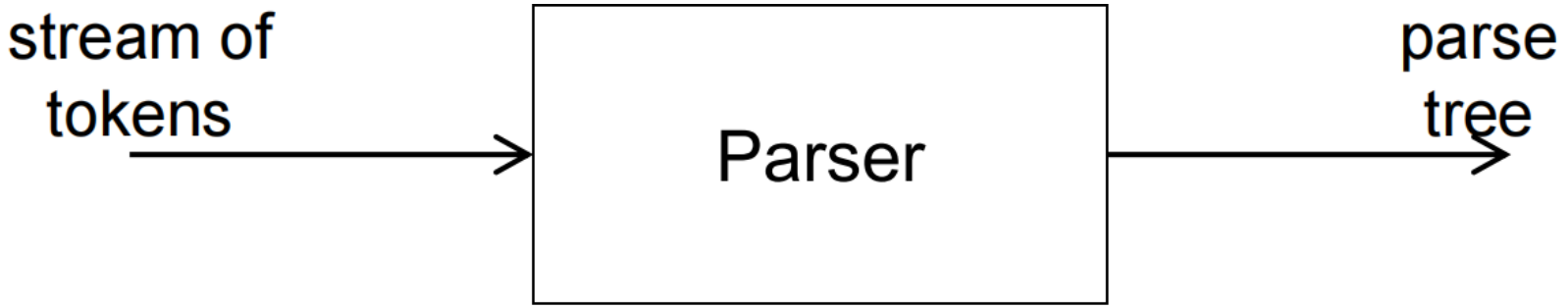
02 LL(1)分析 LL(1) Parsing

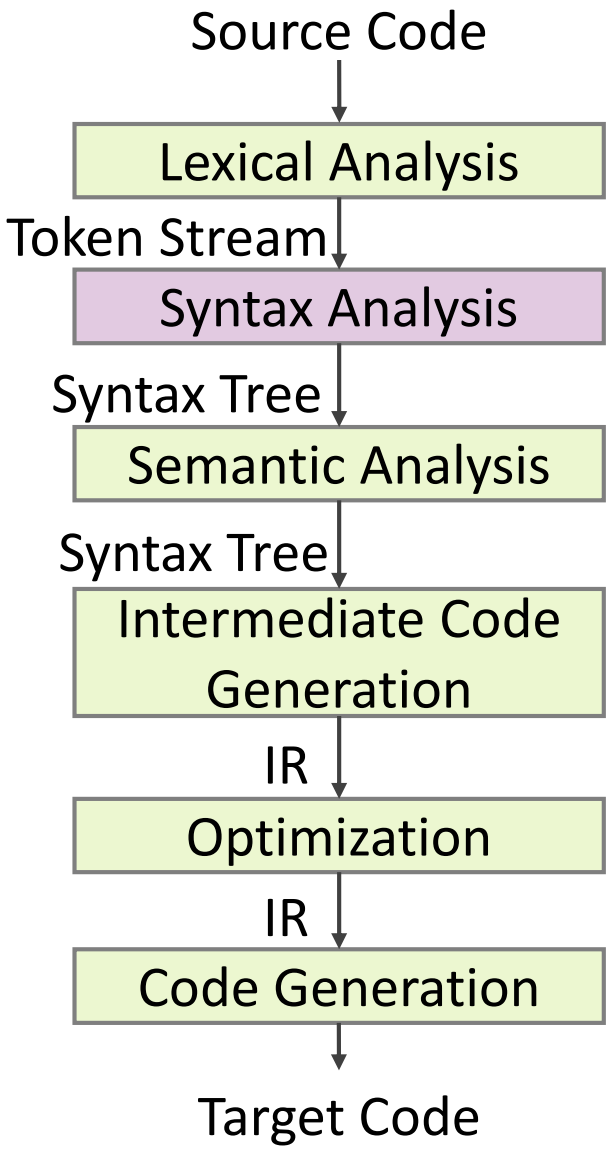
03 自底向上分析 Bottom-Up Parsing

04 LR分析 LR Parsing

1. 语法分析[Syntax Analysis]

- 解析源程序对应的token序列，生成语法分析结构（syntax tree, 语法分析树）





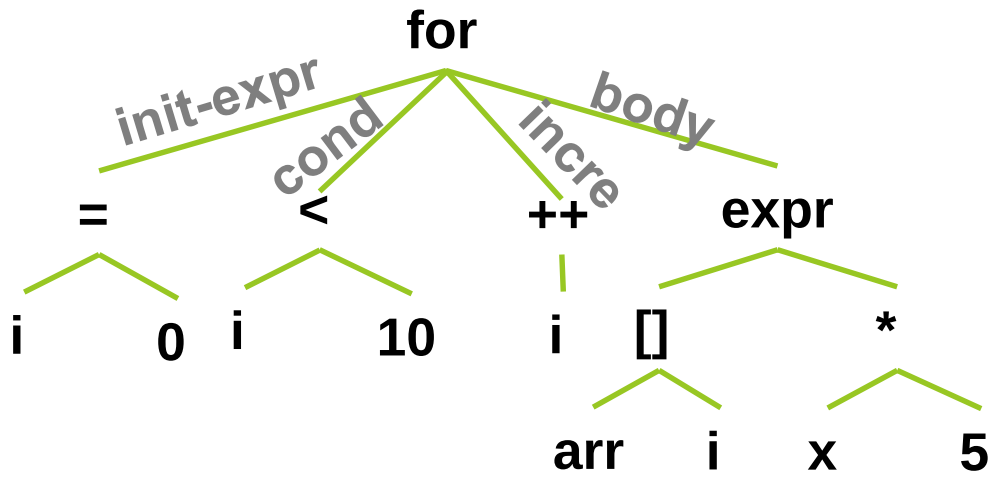
1. 语法分析[Syntax Analysis]

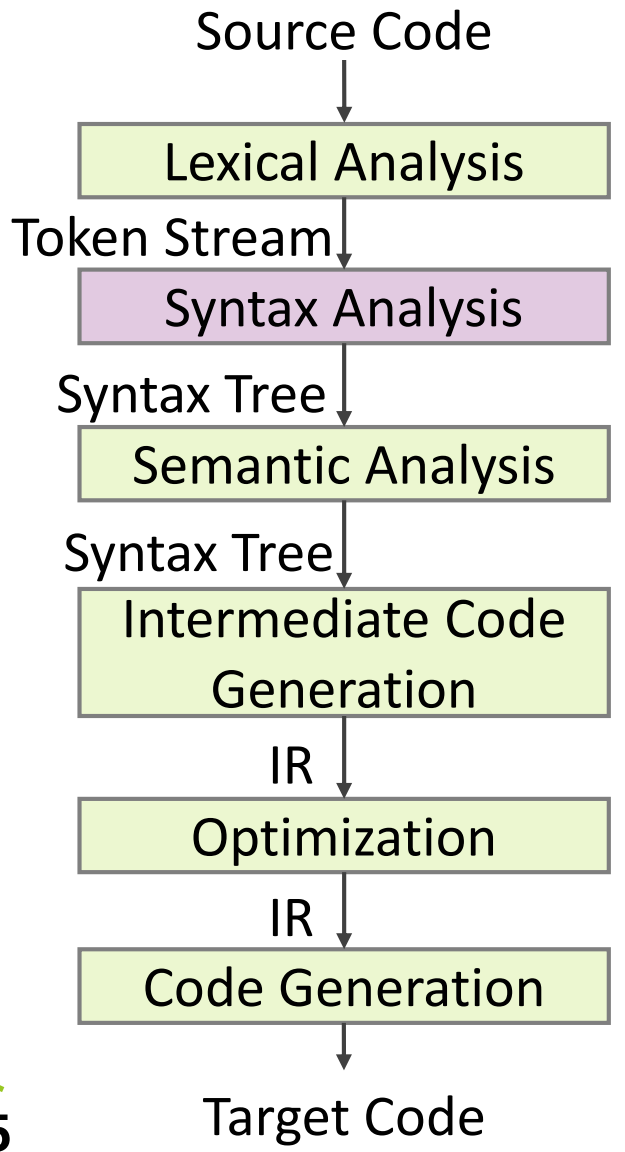
- 解析源程序对应的token序列，生成语法分析结构（syntax tree, 语法分析树）
  - 输入：单词流，输出：语法树
  - 输入程序是否符合语法规则？
    - ✓ x*+
    - ✓ a += 5;

```
void main()
{
    int arr[10], i, x = 1;

    for (i = 0; i < 10; i++)
        arr[i] = x * 5;
}
```

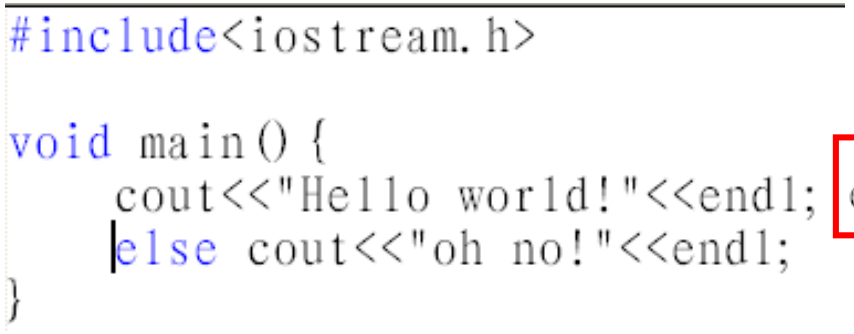
```
keyword(for)
symbol(()
id(i)
symbol(=)
num(0)
symbol(;)
id(i)
symbol(<)
num(10)
symbol(;)
id(i)
symbol(++)
symbol())
id(arr)
symbol([)
id(i)
symbol(])
symbol(=)
id(x)
symbol(*)
num(5)
symbol(;)
```





1. 语法分析[Syntax Analysis]

- 解析源程序对应的token序列，生成语法分析结构（syntax tree, 语法分析树）
  - 输入：单词流，输出：语法树
  - 输入程序是否符合语法规则？



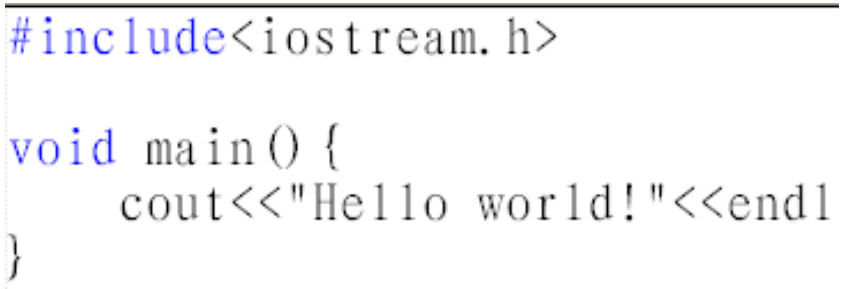
```
#include<iostream.h>

void main(){
    cout<<"Hello world!"<<endl;
    else cout<<"oh no!"<<endl;
}
```

else没有匹配的if

表达式缺少分号结尾

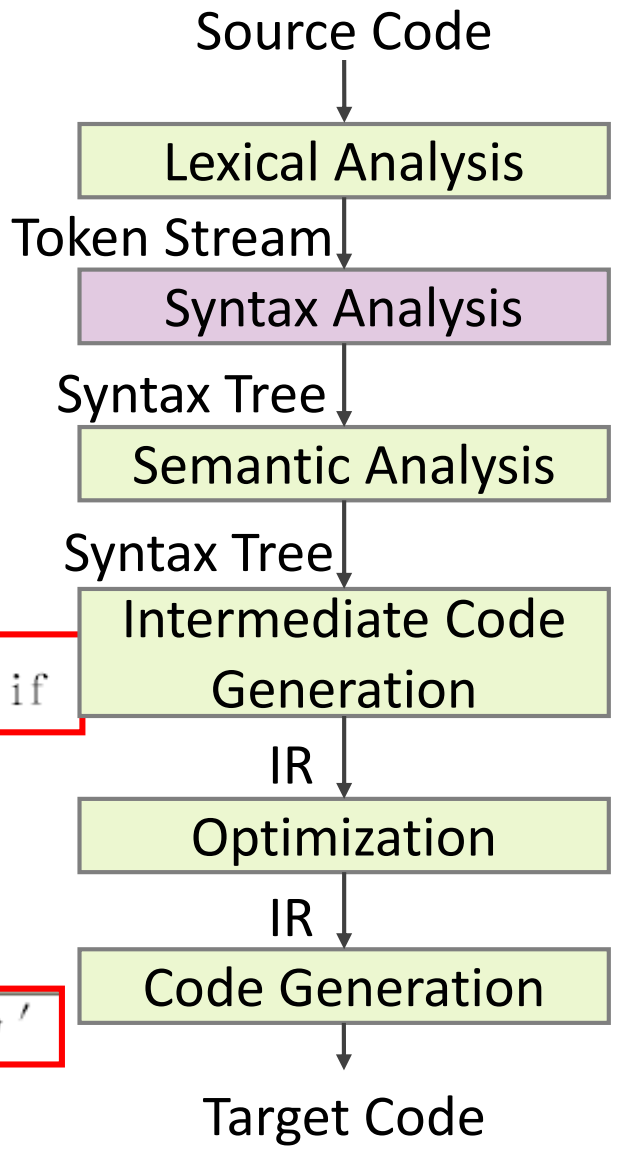
```
error C2181: illegal else without matching if
```



```
#include<iostream.h>

void main(){
    cout<<"Hello world!"<<endl
}
```

```
syntax error : missing ';' before '}'
```



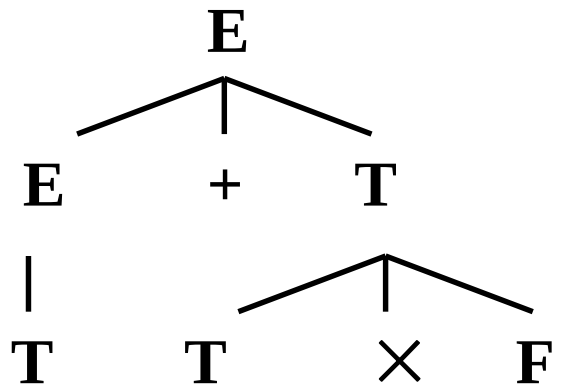
1. 语法分析[Syntax Analysis]

- 复习——准备工作
  - 上下文无关文法**[Context-Free Grammar，CFG]**:
    - ✓ 对任一产生式α→β，都有α∈V_N， β∈(V_N∪V_T)*

      例 文法G[S]：S→AB　A→BS|0　B→SA|1
  - 推导与语法树
    - ✓ 给定文G=(V_N,V_T,P,S)，对于G的任何句型都能构造与之关联的语法树

文法G：　E→E+T|T
T→T×F|F
F→(E)|i

E=>E+T　=>E+T×F　=>T+T×F

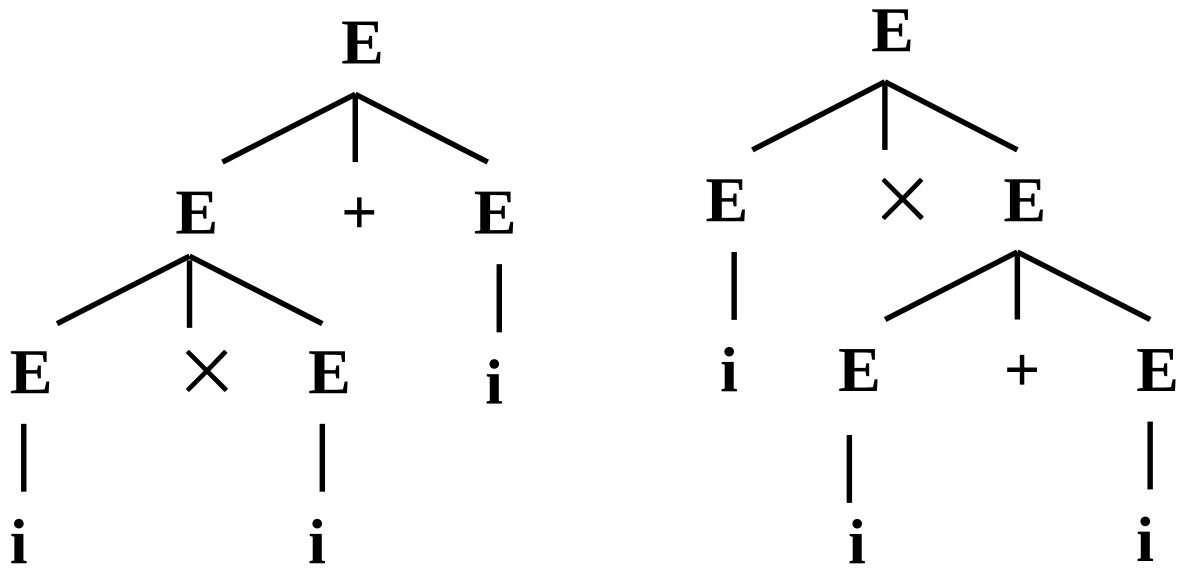


1. 语法分析[Syntax Analysis]

- 复习——准备工作
  - 二义性
    - ✓ 如果一个文法存在某个句子对应**两棵不同的语法树**，则说这个文法是二义的
    - ✓ 二义性文法存在某个句子，它有**两个不同的最左（右）推导**

文法G：E→E+E|E×E|（E）|i

句子 i×i+i 对应的语法树



存在两个不同的最左推导：

推导1：$\underline{E} \Rightarrow \underline{E}+E \Rightarrow \underline{E}\times E+E \Rightarrow i\times\underline{E}+E \Rightarrow i\times i+\underline{E} \Rightarrow i\times i+i$

推导2：$\underline{E} \Rightarrow \underline{E}\times E \Rightarrow i\times\underline{E} \Rightarrow i\times\underline{E}+E \Rightarrow i\times i+\underline{E} \Rightarrow i\times i+i$

1. 语法分析[Syntax Analysis]

- 复习——准备工作
  - 句型分析
    - ✓ **自顶向下分析法 (Top-Down parsing)**
      - **从文法的开始符号出发**，**反复使用文法的产生式**，寻找与输入符号串匹配的推导。
      - 将文法的**开始符号作为语法树的根**，**向下**逐步建立语法树，使语法树的末端结点符号串正好是输入符号串。
    - ✓ **自底向上分析法 (Bottom-Up parsing)**
      - **从输入符号串开始**，逐步进行**归约**，直至归约到文法的开始符号。
      - 从输入符号串开始，以它作为**语法树的末端结点符号串**，**自底向上**的构造语法树。

2. 自顶向下语法分析[Top-Down parsing]

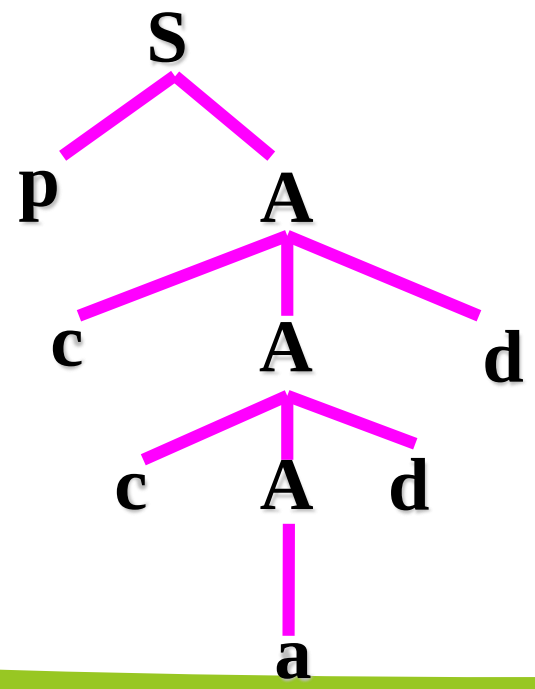
- 例：设有文法G[S]:

  S→pA|qB

  A→cAd|a

  B→dB|b

若输入串W=pccadd。自顶向下的推导过程为:



该推导过程是确定的！

原因:

(1) 每个产生式的右部由终结符开头;

(2) 同一个非终结符的不同产生式的右部由不同的终结符开头。

因此，在推导过程中可以根据当前的输入符号**唯一确定选哪个产生式**往下推导，分析过程是确定的。

S =>pA =>pcAd =>pccAdd =>pccadd

2. 自顶向下语法分析[Top-Down parsing]

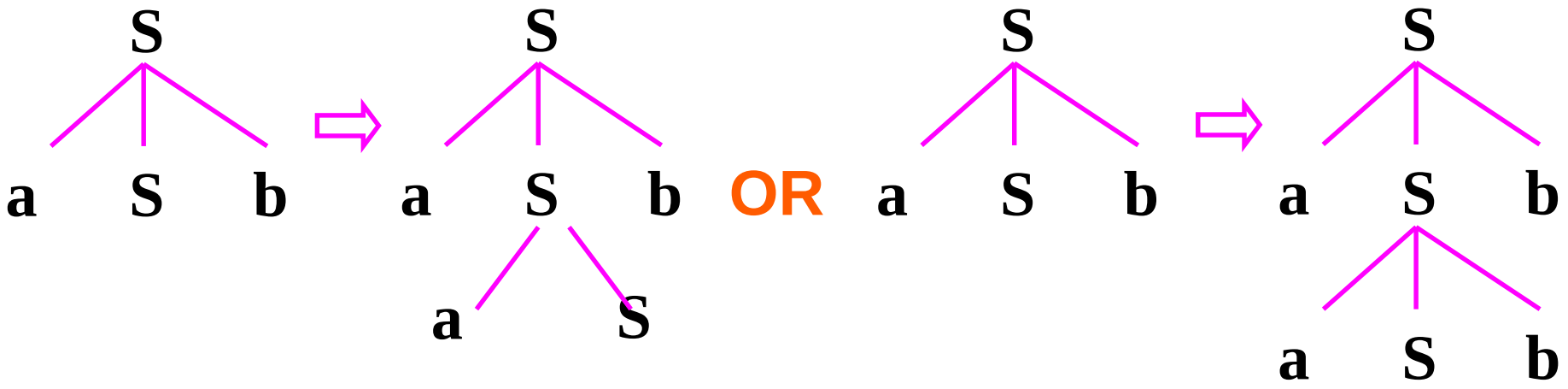
- 例：设有文法G[S]:

  S→aSb

  S→aS

  S→ε

若输入串W=aab。自顶向下的推导过程为:



该推导过程是不确定的！

原因：

同一个非终结符的不同产生式的右部由相同的终结符开头——**存在左公因子**。

因此，要设法**消除左公因子**。

2. 自顶向下语法分析[Top-Down parsing]

- 例：设有文法G[E]:

  E→E+T|T

  T→T*F|F

  F→(E)|id

若输入串W=id+id。自顶向下的推导过程为:

E=>E+T=>E+T+T=>E+T+T+T

该推导过程进入无限循环！

原因:

文法中存在**左递归**的产生式，

导致对该产生式的无限调用。

因此，要设法**消除左递归**。★

为什么不选择产生式E→T，而是一直递归调用E→E+T?

只有当输入串得到匹配时，向前看符号才会发生改变，而调用第一次E→E+T后，输入串id+id并没有得到任何匹配，因此，向前看符号未发生改变，第2次调用E时与第1次调用时采用相同的动作，故而陷入无限循环。

2. 自顶向下语法分析[Top-Down parsing]

- 例：设有文法G[S]:

  S→Sa

  S→b

若输入串W=baaaa。

则自顶向下分析时：

若直接采用S→b：S=>b，无法匹配输入串；

若采用S→Sa：S=>Sa=>Saa...，无法确定何时停止调用S→Sa，使用S→b。

该推导过程是不确定的！

原因：

文法中存在**左递归**的产生式，

导致对该产生式的无限调用。

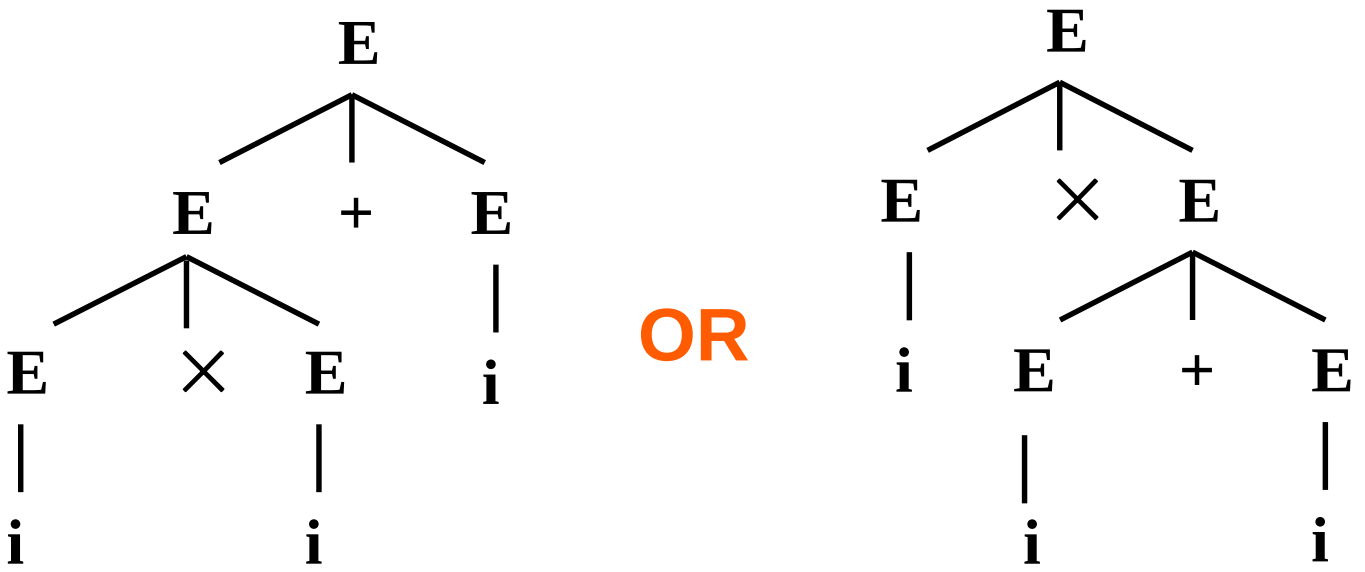
因此，要设法**消除左递归**。★

2. 自顶向下语法分析[Top-Down parsing]

- 例：设有文法G[E]:

$$E \rightarrow E+E|E\times E|(E)|i$$

若输入串W=i×i+i。自顶向下的推导过程为:



该推导过程是不确定的!

原因:
文法**存在二义性**，推导过程本身不唯一。

因此，要设法**消除二义性**。★

存在两个不同的最左推导:

推导1：$\underline{E} \Rightarrow \underline{E}+E \Rightarrow \underline{E}\times E+E \Rightarrow i\times\underline{E}+E \Rightarrow i\times i+\underline{E} \Rightarrow i\times i+i$

推导2：$\underline{E} \Rightarrow \underline{E}\times E \Rightarrow i\times\underline{E} \Rightarrow i\times\underline{E}+E \Rightarrow i\times i+\underline{E} \Rightarrow i\times i+i$

3. 改写文法——消除不确定性/无限循环

(1) 提取左公因子[Left Factoring]

– 规则：

✓ **提取左公因子**，将产生式A→**α**β|**α**r等价变换为：A→**α**(β|r)，

✓ 将括号内用一**新引入的非终结符**A’表示，得：A→αA’，A’→β|r

将做出决定的时间往后延！

– 一般形式：

✓ 若A→**α**β_1|**α**β_2|…|**α**β_n，提取左公共因子后变为：A→**α**(β_1|β_2|…|β_n)，

✓ **引进新的非终结符A’**，得：A→αA’，A’→ β_1|β_2|…|β_n

✓ 若在β_i中仍含有左公共因子，可再次提取。

例：文法G[S]: S→**aS**b|**aS**|ε

✓ **提取左公因子**得：S→aS(b|ε)|ε

✓ **引进新的非终结符S’**得：S→aSS’|ε，S’ →b|ε

随堂练习（1）

- 提取左公因子
  - 对文法G[S]：S→if E then S | if E then S else S | other，E → bool提取左公因子

✓ **提取左公因子**得：S→if E then S(ε|else S) | other

✓ **引进新的非终结符S'**得：S→if E then SS' | other

S'→ε|else S

E → bool

3. 改写文法——消除不确定性/无限循环

(2) 消除左递归[Eliminating Left Recursions]

– 直接左递归

✓ 文法G[E]：**E**→**E**+T|T，**T**→**T***F|F，F→(E)|n

– 间接左递归

✓ 文法G[S]：**S**→**Q**c|c，**Q**→**R**b|b，**R**→**S**a|a

3. 改写文法——消除不确定性/无限循环

(2) 消除左递归

① 消除直接左递归

- 规则：
  - ✓ **引入新的非终结符A'**，将产生式**A**→**A**α|β改写成：A→βA'，A'→αA'|ε
- 为什么能确保等价性？
  - ✓ 原产生式**A**→**A**α|β的语言：**β,βα,βαα,...**，即β后跟零个或多个α，隐式终止递归(不再调用A→Aα时)
  - ✓ **现产生式A→βA'，A'→αA'|ε能产生同样的语言**，变换成右递归，且利用ε产生式显式终止递归
- 一般形式：
  - ✓ 若**A**→**A**α_1|**A**α_2|…|**A**α_m|β_1|β_2|…|β_n，则消除左递归后改写成：

    $A\rightarrow\beta_1A'|\beta_2A'|\ldots|\beta_nA'$　　$A'\rightarrow\alpha_1A'|\alpha_2A'|\ldots|\alpha_mA'|\varepsilon$

3. 改写文法——消除不确定性/无限循环

(2) 消除左递归

① 消除直接左递归

例：文法G[E]：**E**→**E**+T|T，**T**→**T***F|F，F→(E)|n

- **引入E'**，E→E+T|T去除左递归得：E→TE'，E'→+TE'|ε
- **引入T'**，T→T*F|F去除左递归得：T→FT'，T'→*FT'|ε
- 最终：E→TE'，E'→+TE'|ε，T→FT'，T'→*FT'|ε，F→(E)|n

3. 改写文法——消除不确定性/无限循环

(2) 消除左递归

② 消除间接左递归

– 规则：把间接左递归变成直接左递归，再加以消除。

– 具体步骤：

✓ 把文法的所有非终结符**按任一顺序排列**，如：A_1，A_2，…，A_n

✓ 从A_1开始，按以下顺序处理A_i：

- 若左部为A_i的产生式的右部为非终结符A_j(**j<i**)开头，即$A_i \rightarrow A_j \ldots$，则**用左部为$A_j$的所有产生式的右部分别代替$A_i \rightarrow A_j \ldots$ 中的A_j**；
- 得到的左部为A_i的产生式若有直接左递归，则消除之。

✓ **去掉无用产生式**。

3. 改写文法——消除不确定性/无限循环

(2) 消除左递归

② 消除间接左递归

例：文法G[S]：(1)**S→Q**c|c　(2)**Q→R**b|b　(3)**R→S**a|a

- 将非终结符**排序 ： R，Q，S**
- **对R**：产生式(3)不含直接左递归，所以保持不变
- **对Q**：把(3)代入(2)得**(2')Q→Sab|ab|b**，无直接左递归
- **对S**：把(2')代入(1)得**(1')S→Sabc|abc|bc|c**，

  有直接左递归，消除直接左递归得： **S→abcS'|bcS'|cS'　S'→abcS'|ε**
- 处理结果为：R→Sa|a，Q→Sab|ab|b，S→abcS'|bcS'|cS'，S' →abcS'|ε
- 由于Q，R是不可到达的非终结符，其产生式应删除
- 最终得文法G'[S]：S→abcS'|bcS'|cS'，S'→abcS'|ε

若非终结符顺序为S, Q, R呢?

消除左递归的最终结果为：

S→Qc|c，Q→Rb|b，

R→bcaR'|caR'|aR'，

R'→bcaR'|ε。

当非终结符的排序不同时，结果的产生式形式不同，但它们是等价的。

随堂练习（2）

- 消除左递归
  - 对文法G[S]: S→Aa|b，A→Ac|Sd|ε消除左递归
    - ✓ 将非终结符**排序 ： S，A**
    - ✓ **对S**：无直接左递归，S→Aa|b保持不变
    - ✓ **对A**：用S的产生式替换A→Sd中的S：A→Ac|Aad|bd|ε
    - ✓ **引入A’，消除直接左递归**：A→bdA’|A’，A’→cA’|adA’|ε，S→Aa|b

3. 改写文法——消除不确定性/无限循环

(3) 消除二义性[Resolving Ambiguities]

例：文法G[stmt]:(**Dangling-else**)

stmt → **if** *expr* **then** *stmt*

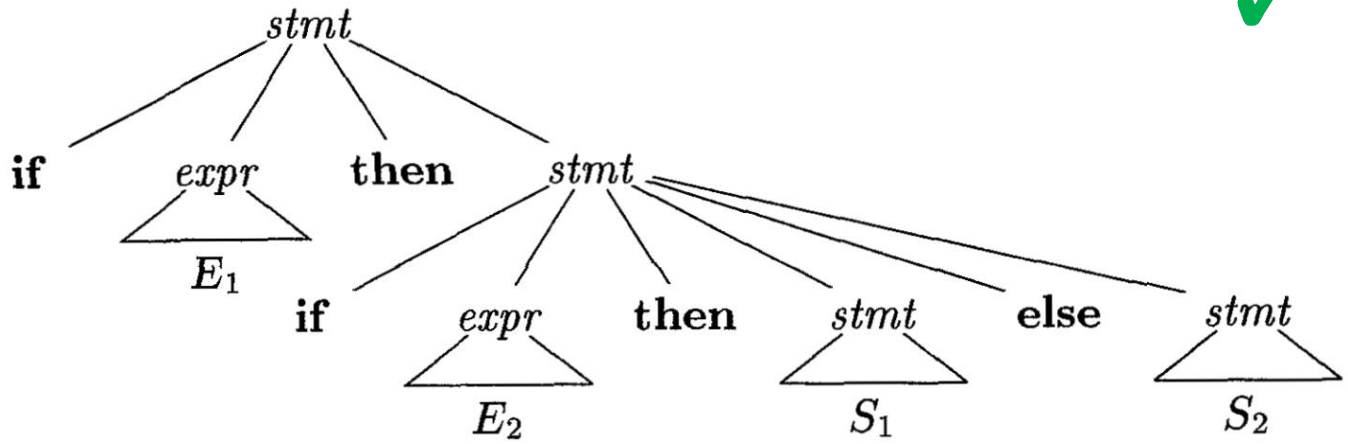
| **if** *expr* **then** *stmt* **else** *stmt*

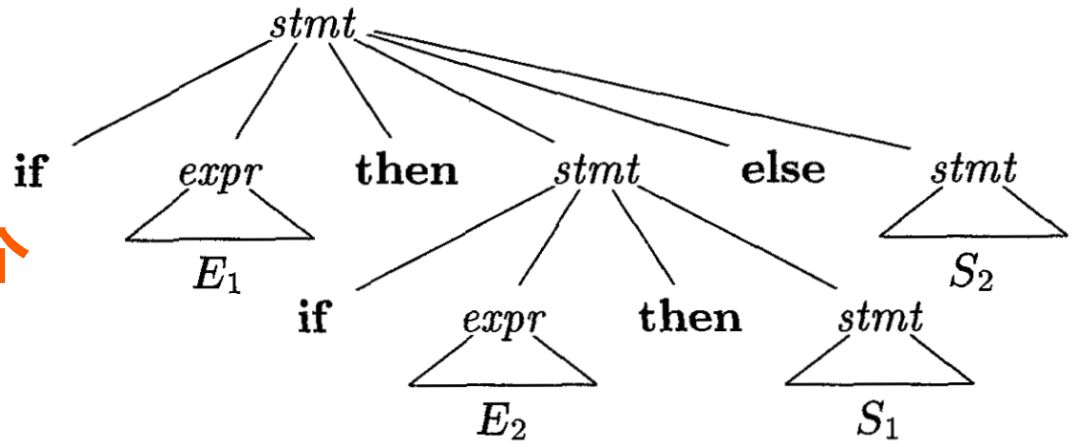
| **other**

– 存在二义性：

- ✓ if E1 then **if E2 then S1 else S2**
- ✓ if E1 then **if E2 then S1** else S2
- ✓ 可通过施加额外的规则来消除二义性，如“**每个else和最近的尚未匹配的then匹配**”
- ✓ 更好的做法是**改写成等价的无二义性的文法**

if E1 then S1 else **if E2 then S2 else S3** ✓





3. 改写文法——消除不确定性/无限循环

(3) 消除二义性

例：文法G[stmt]:(**Dangling-else**)

stmt → **if** *expr* **then** *stmt*

| **if** *expr* **then** *stmt* **else** *stmt*

| **other**

stmt → *matched_stmt* | *open_stmt*

matched_stmt → **if** *expr* **then** ***matched_stmt*** **else** *matched_stmt* **(完整)**

| other

open_stmt → **if** *expr* **then** *stmt* **(开放)**

| **if** *expr* **then** ***matched_stmt*** **else** *open_stmt*

改写原则：

- ✓ 在一个then和一个else**之间出现的语句必须是“已匹配的”**，不能以一个尚未匹配的(开放的)then结尾(没有对应的else)
- ✓ 已匹配的语句：要么是一个不包含开放语句的**完整的if-then-else语句**，要么是一个非条件语句
- ✓ 开放语句：以if-then结尾，没有对应的else

if E1 then if E2 then S1 else S2

3. 改写文法——消除不确定性/无限循环

(4) 消除ε产生式[Eliminating ε-Productions]

– 规则：

- ✓ **将每个产生式右部的非终结符均替换成ε，如果该非终结符能推出ε的话**
- ✓ **把所有可能性都替换出来**
- ✓ 对每个产生式$A \rightarrow X_1X_2...X_n$，其中$X_i \in \Sigma \cup N$，$1 \le i \le n$
- ✓ 增加新产生式$A \rightarrow a_1a_2...a_n$，其中：
  - **若$X_i \Rightarrow^* \varepsilon$，则$a_i=X_i|a_i=\varepsilon$；**
  - **若$X_i \not\Rightarrow^* \varepsilon$，则$a_i=X_i$**
  - $1 \le i \le n$，$a_i \neq \varepsilon$

例：文法G[S]：S→Aa|b，A→Ac|Sd|**ε**

改写为：S→Aa|**a**|b，A→Ac|**c**|Sd

随堂练习（4）

- 消除ε产生式
  - 对文法G[S]：S→aSbS|bSaS|ε消除ε产生式

改写为：

S'→S|**ε**

S→aSbS|abS|aSb|ab|bSaS|baS|bSa|ba，

注意：若由开始符号S→ε，则应补充S'→S|ε