



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

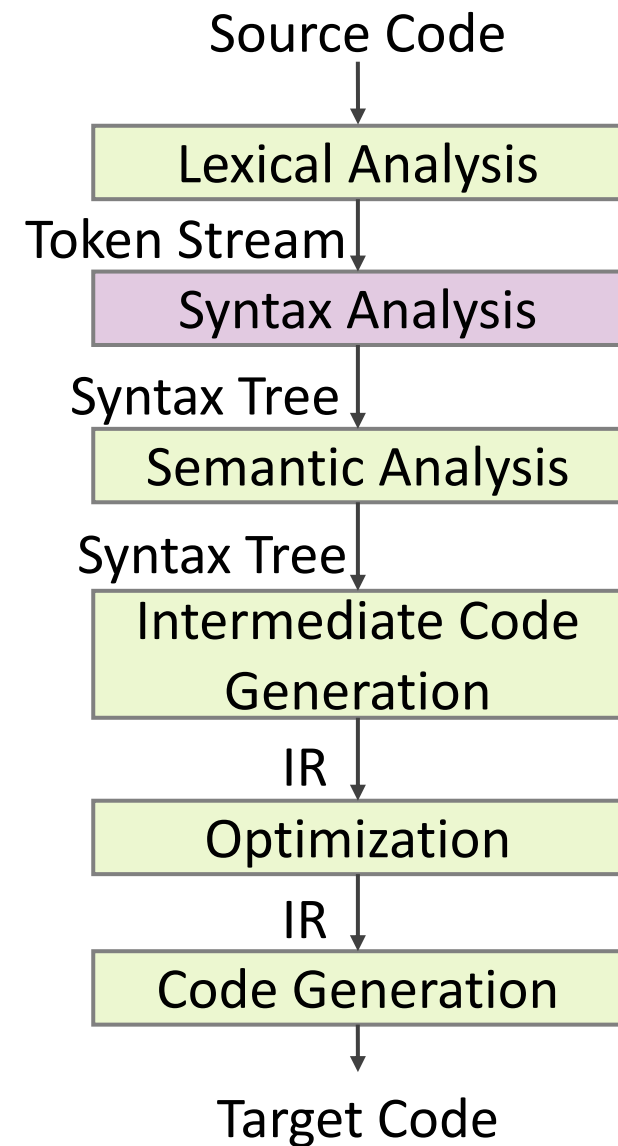
算符优先分析
Operator
Precedence Parsing

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)



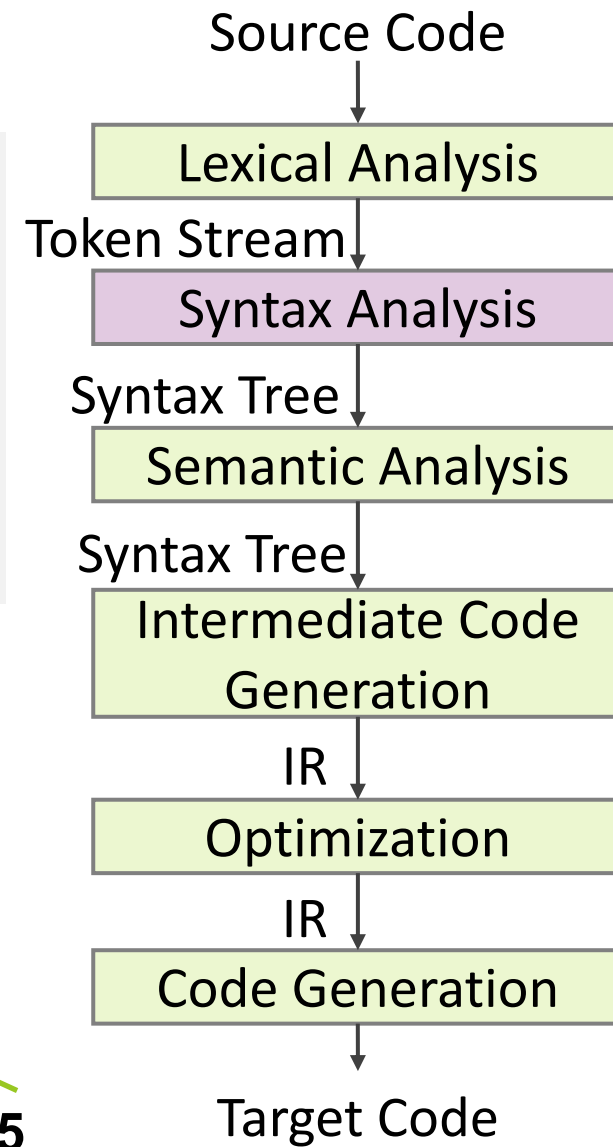
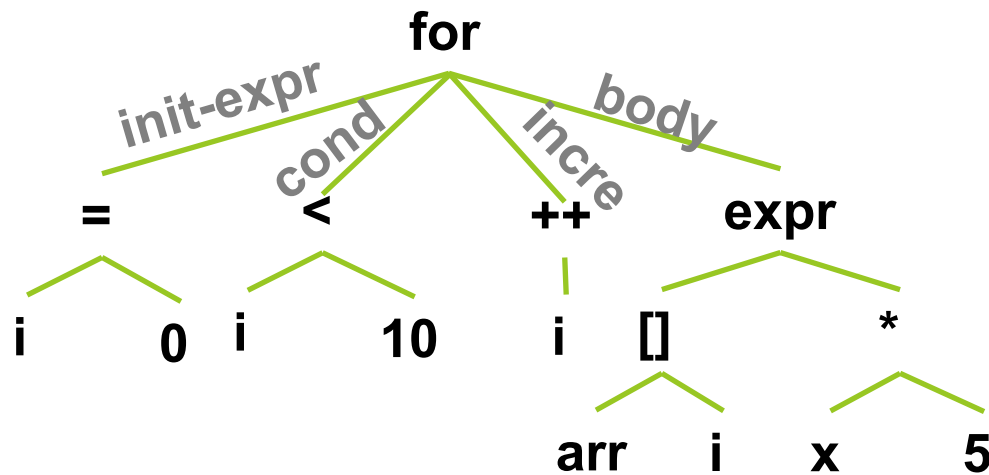
1. 语法分析[Syntax Analysis]

- 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;`

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

```
void main()
{
    int arr[10], i, x = 1;
    for (i = 0; i < 10; i++)
        arr[i] = x * 5;
}
```



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**?

else没有匹配的if
表达式缺少分号结尾

```
#include<iostream.h>
```

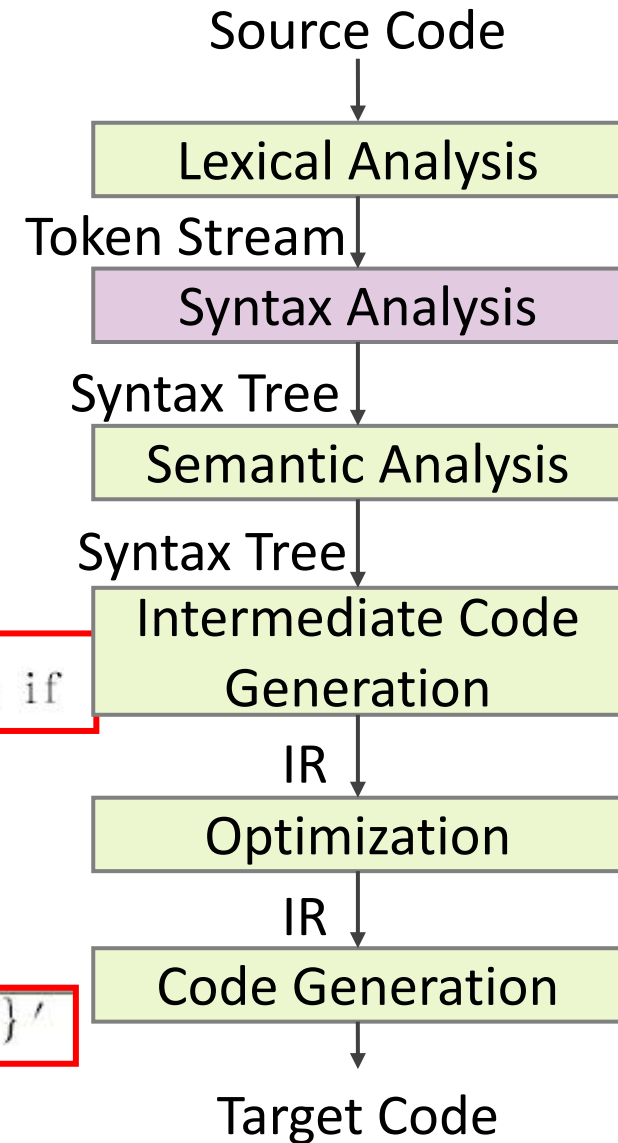
```
void main() {
    cout<<"Hello world!"<<endl;
    else cout<<"oh no!"<<endl;
}
```

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 $\alpha \rightarrow \beta$, 都有 $\alpha \in V_N$, $\beta \in (V_N \cup V_T)^*$

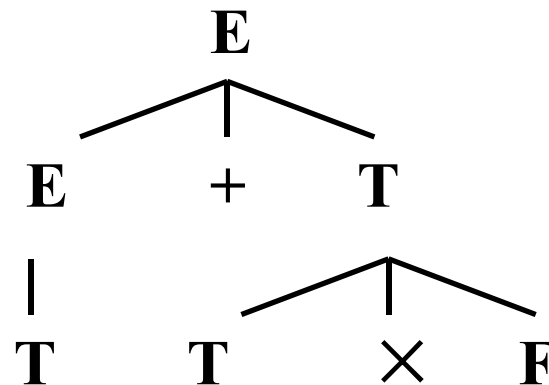
例 文法G[S]: $S \rightarrow AB$ $A \rightarrow BS|0$ $B \rightarrow SA|1$

– 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: $E \rightarrow E+T|T$
 $T \rightarrow T \times F|F$
 $F \rightarrow (E)|i$

$\underline{E} \Rightarrow \underline{E} + \underline{T} \Rightarrow \underline{E} + T \times F \Rightarrow T + T \times F$



1. 语法分析[Syntax Analysis]

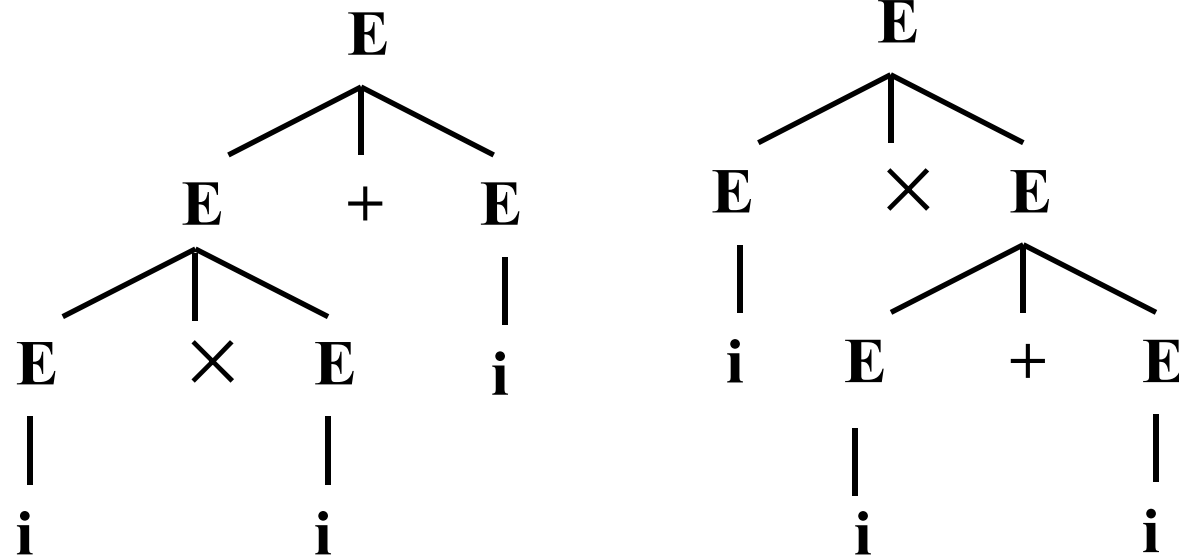
• 复习——准备工作

- 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: $E \rightarrow E+E \mid E \times E \mid (E) \mid i$

句子 $i \times i + i$ 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]

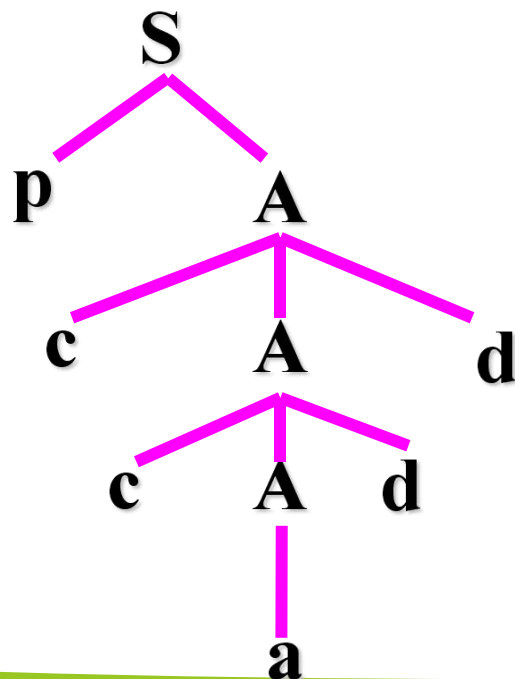
• 例：设有文法G[S]:

$$S \rightarrow pA|qB$$

$$A \rightarrow cAd|a$$

$$B \rightarrow dB|b$$

若输入串W=pccadd。自顶向下的推导过程为：



$$\underline{S} \Rightarrow p\underline{A} \Rightarrow pc\underline{A}d \Rightarrow pcc\underline{A}dd \Rightarrow pccadd$$

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。

2. 自顶向下语法分析[Top-Down parsing]

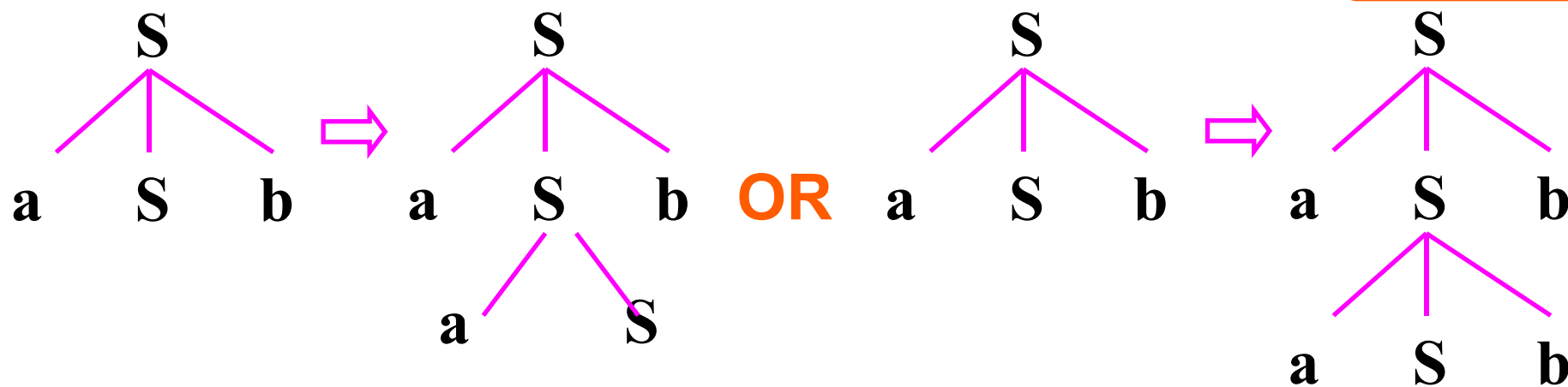
- 例：设有文法G[S]:

$$S \rightarrow aSb$$

$$S \rightarrow aS$$

$$S \rightarrow \varepsilon$$

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]:

$$E \rightarrow E+T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid id$$

若输入串W=id+id。自顶向下的推导过程为:

$$E \Rightarrow E+T \Rightarrow E+T+T \Rightarrow E+T+T+T$$

为什么不选择产生式 $E \rightarrow T$ ，而是一直递归调用 $E \rightarrow E+T$ ？

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 $E \rightarrow E+T$ 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。

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**。★

2. 自顶向下语法分析[Top-Down parsing]

- 例：设有文法G[S]:

$$S \rightarrow Sa$$
$$S \rightarrow b$$

若输入串 $W=baaaa$ 。

则自顶向下分析时：

若直接采用 $S \rightarrow b$ ： $S \Rightarrow b$ ，无法匹配输入串；

若采用 $S \rightarrow Sa$ ： $S \Rightarrow Sa \Rightarrow Saa \dots$ ，无法确定何时停止调用 $S \rightarrow Sa$ ，使用 $S \rightarrow b$ 。

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。

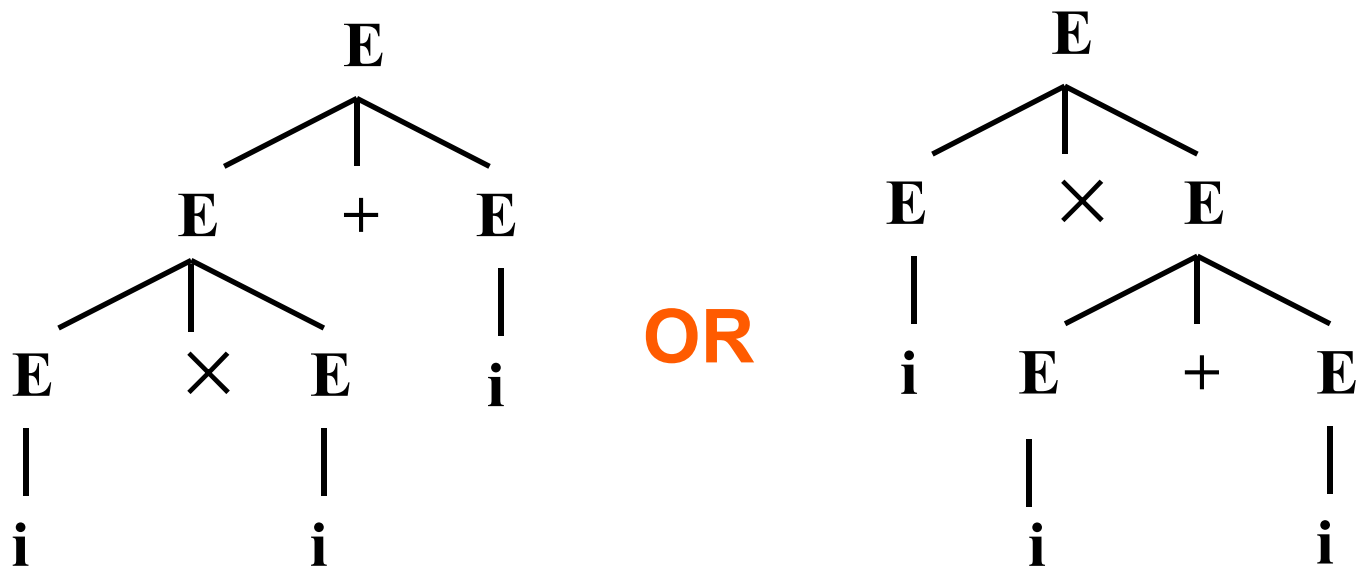
因此，要设法**消除左递归**。★

2. 自顶向下语法分析[Top-Down parsing]

- 例：设有文法G[E]:

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

若输入串W=i×i+i。自顶向下的推导过程为:



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

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**。★

3. 改写文法——消除不确定性/无限循环

(1) 提取左公因子

– 规则：

✓ **提取左公因子**，将产生式 $A \rightarrow \alpha\beta | \alpha r$ 等价变换为： $A \rightarrow \alpha(\beta | r)$ ，

✓ 将括号内用一**新引入的非终结符** A' 表示，得： $A \rightarrow \alpha A'$ ， $A' \rightarrow \beta | r$

– 一般形式：

将做出决定的时间往后延！

✓ 若 $A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \dots | \alpha\beta_n$ ，提取左公共因子后变为： $A \rightarrow \alpha(\beta_1 | \beta_2 | \dots | \beta_n)$ ，

✓ **引进新的非终结符 A'** ，得： $A \rightarrow \alpha A'$ ， $A' \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$

✓ 若在 β_i 中仍含有左公共因子，可再次提取。

例：文法 $G[S]$: $S \rightarrow aSb | aS | \epsilon$

✓ **提取左公因子**得： $S \rightarrow aS(b | \epsilon) | \epsilon$

✓ **引进新的非终结符 S'** 得： $S \rightarrow aSS' | \epsilon$ ， $S' \rightarrow b | \epsilon$

随堂练习 (1)

• 提取左公因子

– 对文法 $G[S]$: $S \rightarrow \text{if } E \text{ then } S \mid \text{if } E \text{ then } S \text{ else } S \mid \text{other}$, $E \rightarrow \text{bool}$ 提取左公因子

✓ **提取左公因子**得: $S \rightarrow \text{if } E \text{ then } S(\epsilon \mid \text{else } S) \mid \text{other}$

✓ **引进新的非终结符 S'** 得: $S \rightarrow \text{if } E \text{ then } SS' \mid \text{other}$

$S' \rightarrow \epsilon \mid \text{else } S$

$E \rightarrow \text{bool}$

3. 改写文法——消除不确定性/无限循环

(2) 消除左递归

– 直接左递归

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

– 间接左递归

✓ 文法 $G[S]$: $S \rightarrow Qc|c$, $Q \rightarrow Rb|b$, $R \rightarrow Sa|a$

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 $A \rightarrow A\alpha | \beta$ 改写成: $A \rightarrow \beta A'$, $A' \rightarrow \alpha A' | \epsilon$

– 为什么能确保等价性?

✓ 原产生式 $A \rightarrow A\alpha | \beta$ 的语言: $\beta, \beta\alpha, \beta\alpha\alpha, \dots$, 即 β 后跟零个或多个 α , 隐式终止递归(不再调用 $A \rightarrow A\alpha$ 时)

✓ **现产生式 $A \rightarrow \beta A'$, $A' \rightarrow \alpha A' | \epsilon$ 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 $A \rightarrow A\alpha_1 | A\alpha_2 | \dots | A\alpha_m | \beta_1 | \beta_2 | \dots | \beta_n$, 则消除左递归后改写成:

$$A \rightarrow \beta_1 A' | \beta_2 A' | \dots | \beta_n A' \quad A' \rightarrow \alpha_1 A' | \alpha_2 A' | \dots | \alpha_m A' | \epsilon$$

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 $G[E]$: $E \rightarrow E+T|T$, $T \rightarrow T*F|F$, $F \rightarrow (E)|n$

– 引入 E' , $E \rightarrow E+T|T$ 去除左递归得: $E \rightarrow TE'$, $E' \rightarrow +TE'|\epsilon$

– 引入 T' , $T \rightarrow T*F|F$ 去除左递归得: $T \rightarrow FT'$, $T' \rightarrow *FT'|\epsilon$

– 最终: $E \rightarrow TE'$, $E' \rightarrow +TE'|\epsilon$, $T \rightarrow FT'$, $T' \rightarrow *FT'|\epsilon$, $F \rightarrow (E)|n$

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, \dots, 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 $A_i \rightarrow A_j \dots$ ，则**用左部为**

A_j 的所有产生式的右部分别代替 $A_i \rightarrow A_j \dots$ 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) $S \rightarrow Qc|c$ (2) $Q \rightarrow Rb|b$ (3) $R \rightarrow Sa|a$

– 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') $Q \rightarrow Sab|ab|b$ ，无直接左递归

– **对S**：把(2')代入(1)得(1') $S \rightarrow Sabc|abc|bc|c$ ，

有直接左递归，消除直接左递归得： $S \rightarrow abcS'|bcS'|cS'$ $S' \rightarrow abcS'|\epsilon$

– 处理结果为： $R \rightarrow Sa|a$ ， $Q \rightarrow Sab|ab|b$ ， $S \rightarrow abcS'|bcS'|cS'$ ， $S' \rightarrow abcS'|\epsilon$

– 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 $G'[S]$ ： $S \rightarrow abcS'|bcS'|cS'$ ， $S' \rightarrow abcS'|\epsilon$

若非终结符顺序为 S, Q, R 呢？

消除左递归的最终结果为：

$S \rightarrow Qc|c$ ， $Q \rightarrow Rb|b$ ，

$R \rightarrow bcaR'|caR'|aR'$ ，

$R' \rightarrow bcaR'|\epsilon$ 。

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 $G[S]: S \rightarrow Aa|b, A \rightarrow Ac|Sd|\varepsilon$ 消除左递归

✓ 将非终结符**排序**：**S, A**

✓ **对S**：无直接左递归， $S \rightarrow Aa|b$ 保持不变

✓ **对A**：用S的产生式替换 $A \rightarrow Sd$ 中的S： $A \rightarrow Ac|Aad|bd|\varepsilon$

✓ **引入A'**，**消除直接左递归**： $A \rightarrow bdA'|A', A' \rightarrow cA'|adA'|\varepsilon, S \rightarrow Aa|b$

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**)

$stmt \rightarrow$ **if** *expr* **then** *stmt*

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

| **other**

– 存在二义性：

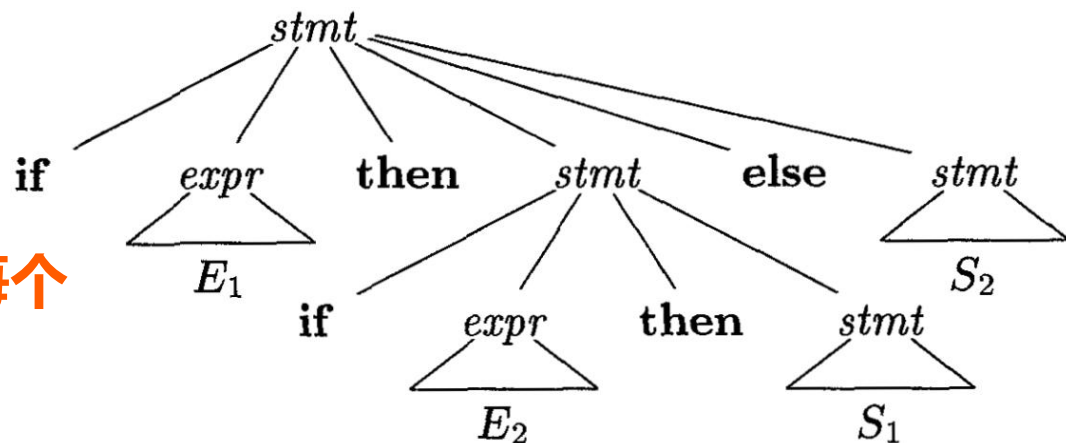
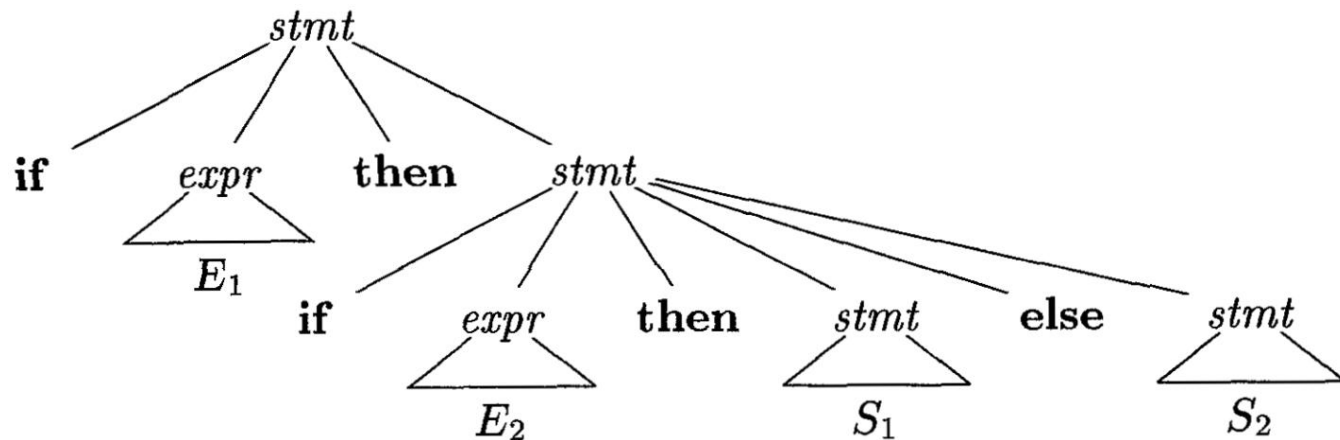
✓ if E1 then **if E2 then S1 else S2**

✓ 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

– 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 $A \rightarrow X_1X_2\dots X_n$ ，其中 $X_i \in \Sigma \cup N$ ， $1 \leq i \leq n$
- ✓ 增加新产生式 $A \rightarrow a_1a_2\dots a_n$ ，其中：
 - 若 $X_i \Rightarrow^* \varepsilon$ ，则 $a_i = X_i \mid a_i = \varepsilon$;
 - 若 $X_i \not\Rightarrow^* \varepsilon$ ，则 $a_i = X_i$
 - $1 \leq i \leq n$ ， $a_i \neq \varepsilon$

例：文法 $G[S]$: $S \rightarrow Aa \mid b$, $A \rightarrow Ac \mid Sd \mid \varepsilon$

改写为: $S \rightarrow Aa \mid a \mid b$, $A \rightarrow Ac \mid c \mid Sd$

随堂练习 (4)

• 消除 ϵ 产生式

– 对文法 $G[S]: S \rightarrow aSbS | bSaS | \epsilon$ 消除 ϵ 产生式

改写为:

$$S' \rightarrow S | \epsilon$$

$$S \rightarrow aSbS | abS | aSb | ab | bSaS | baS | bSa | ba,$$

注意: 若由开始符号 $S \rightarrow \epsilon$, 则应补充 $S' \rightarrow S | \epsilon$