WANG Hanfei

School of Computer Wuhan University

September 22, 2015



上下文无关文法

- 上下文无关文法
 - 锑归定义与产生式
 - 上下文无关文法
 - 上下文无关语言
- 语法树
 - 语法树
 - 语法树与椎导的关系
 - 二义性
- ③ 上下文无关文法的设计
 - 设计举例
 - 正则表达式转换为上下支自由支法
 - 自动机转换为上下文自由文法
 - CFG 的表达能力
- 4 二义性的再讨论

上下支充关支法

上下文无关文法

- 递归定义与产生式
- 上下文无关文法
- 上下文无关语言
- - 语法树
 - 语法树与维导的关系
 - ●二义性
- 上下文无关文法的设计
 - 设计举例
 - 正则表达式转换为上下支自由支法
 - 自动机转换为上下支自由支法
 - CFG 的表达能力
- 二义性的再讨论

XL 语言





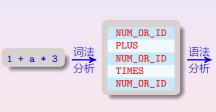
イロト (個) (を見) (達)

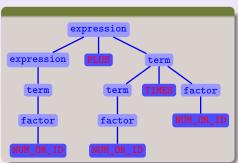




◆□ト→圖ト→重ト→重ト

XL 语言





上下文无关文法

- 形式语言的第二次抽象,对单词再次重组.
 - Ex: Σ = {NUM_OR_ID, PLUS, TIMES, LP, RP, SEMI};
- 描述问题:

- 识别问题: ∃PΣ* → Boolean, s → true or false.
- 语法树的建立,

- 形式语言的第二次抽象,对单词再次重组.
 - Ex: Σ = {NUM_OR_ID, PLUS, TIMES, LP, RP, SEMI};
- XL ⊆ Z**.
- 描述问题:

- 识别问题: ∃PΣ* → Boolean, s → true or false
- 语法树的建立,

- 形式语言的第二次抽象,对单词再次重组.
 - Ex: Σ = {NUM_OR_ID, PLUS, TIMES, LP, RP, SEMI};
 - \bullet XL $\subset \Sigma^*$.
- 描述问题

- 识别问题: ∃PΣ* → Boolean, s → true or false.
- 语法树的建立

- 形式语言的第二次抽象,对单词再次重组。
 - Ex: Σ = {NUM OR ID, PLUS, TIMES, LP, RP, SEMI};
 - $XL \subset \Sigma^*$.
- 描述问题:
 - 正则表达式:表达能力太弱,嵌套的括号对不能描述
 - 數学递归定义:不利于机器对文法的识别。
- 识别问题: ∃PΣ* → Boolean, s → true or false.
- 语法树的建立,

- 形式语言的第二次抽象,对单词再次重组。
 - Ex: Σ = {NUM OR ID, PLUS, TIMES, LP, RP, SEMI};
 - XL $\subset \Sigma^*$.
- 描述问题:
 - 正则表达式:表达能力太弱,嵌套的括号对不能描述。

- 形式语言的第二次抽象,对单词再次重组。
 - Ex: $\Sigma = \{\text{NUM OR ID, PLUS, TIMES, LP, RP, SEMI}\}$;
 - XL $\subset \Sigma^*$.
- 描述问题:
 - 正则表达式:表达能力太弱,嵌套的括号对不能描述。
 - 数学递归定义: 不利于机器对文法的识别。

- 形式语言的第二次抽象,对单词再次重组。
 - Ex: Σ = {NUM OR ID, PLUS, TIMES, LP, RP, SEMI};
 - $XL \subset \Sigma^*$.
- 描述问题:
 - 正则表达式:表达能力太弱,嵌套的括号对不能描述.
 - 数学递归定义:不利于机器对文法的识别.
- 识别问题: ∃PΣ* → Boolean, s → true or false.
- ◎ 语法树的建立

- 形式语言的第二次抽象,对单词再次重组。
 - Ex: $\Sigma = \{\text{NUM OR ID, PLUS, TIMES, LP, RP, SEMI}\}$;
 - XL $\subset \Sigma^*$.
- 描述问题:
 - 正则表达式:表达能力太弱,嵌套的括号对不能描述。
 - 数学递归定义: 不利于机器对文法的识别。
- 识别问题: ∃PΣ* → Boolean, s → true or false.
- 语法树的建立

```
设: \Sigma = \{NUM\_OR\_ID, PLUS, TIMES, LP, RP\}, factor, term, expression \subset \Sigma^* 递归定义如下:
```

- ① 归纳基础: NUM OR TD ← factor
 - ② 归纳条款: S₁ ∈ expression, then LP S₁ RP ∈ factor.
 if S₁ ∈ factor, S₂ ∈ term, then S₁, S₂ TIMES S₁ ∈ term.
 if S₁ ∈ term, S₂ ∈ expression, then S₁, S₂ PLUS S₁ ∈ expression
 其中: S₁和 S₂和为递归定义中的三变量 (metavariable)
 - 3 由以上规则在有限步生成的字符串构成factor、 term和expression集合。称为语言、每个语言中的云麦称为语句

XL 语言的语句 --- 归纳过程

设: $\Sigma = \{NUM_OR_ID, PLUS, TIMES, LP, RP\}$, factor, term, expression $\subset \Sigma *$ 遂归定义如下:

- 」 回纳基础: NUM OR ID∈ factor.
 - ② 归纳条款: $S_1 \in \text{expression}$, then LP $S_1 \text{ RP} \in \text{factor}$.

 if $S_1 \in \text{factor}$, $S_2 \in \text{term}$, then S_1 , $S_2 \text{ TIMES } S_1 \in \text{term}$.

 if $S_1 \in \text{term}$, $S_2 \in \text{expression}$, then S_1 , $S_2 \text{ PLUS } S_1 \in \text{expression}$ 其中、 $S_2 \oplus S_2 \oplus S_3 \oplus \mathbb{R}$ 英语中文中的一个景(metavariable)
- ③ 由以上规则在有限步生成的字符串构成factor、 term和expression集合,称为语言.每个语言中的元素称为语句.

XL 语言的语句 --- 归纳过程

```
设: Σ = {NUM_OR_ID, PLUS, TIMES, LP, RP}, factor, term, expression 

⊂ Σ*递归定义如下:
```

- ② 归纳条款: $S_1 \in \text{expression}$, then $\text{LP } S_1 \text{ RP} \in \text{factor}$. if $S_1 \in \text{factor}$, $S_2 \in \text{term}$, then S_1 , $S_2 \in \text{TIMES}$ $S_1 \in \text{term}$. if $S_1 \in \text{term}$, $S_2 \in \text{expression}$, then S_1 , $S_2 \in \text{PLUS}$ $S_1 \in \text{expression}$. 其中: $S_1 \neq S_2$ 称为递归定义中的元变量 (metavariable).
 - 3 由以上规则在有限涉住成的字符串构成factor、 term和expression集合、称为语言。每个语言中的元素称为语句

XL 语言的语句 --- 归纳过程

设: $\Sigma = \{NUM_OR_ID, PLUS, TIMES, LP, RP\}$, factor, term, expression $\subset \Sigma *$ 递归定义如下:

- ② 归纳条款: $S_1 \in \text{expression}$, then LP $S_1 \text{ RP} \in \text{factor}$. if $S_1 \in \text{factor}$, $S_2 \in \text{term}$, then S_1 , $S_2 \in \text{TIMES}$ $S_1 \in \text{term}$. if $S_1 \in \text{term}$, $S_2 \in \text{expression}$, then S_1 , $S_2 \in \text{PLUS}$ $S_1 \in \text{expression}$. 其中: $S_1 \in S_2$ 称为递归定义中的元变量 (metavariable).
- ③ 由以上规则在有限步生成的字符串构成factor、 term和expression集合,称为语言。每个语言中的元素称为语句。

XL 语言的语句 --- 归纳过程

设: Σ = {NUM_OR_ID, PLUS, TIMES, LP, RP}, factor, term, expression ⊂ Σ*递归定义如下:

- ② 归纳条款: $S_1 \in \text{expression}$, then $\text{LP } S_1 \text{ RP} \in \text{factor}$. if $S_1 \in \text{factor}$, $S_2 \in \text{term}$, then S_1 , $S_2 \text{ TIMES } S_1 \in \text{term}$. if $S_1 \in \text{term}$, $S_2 \in \text{expression}$, then S_1 , $S_2 \in \text{PLUS } S_1 \in \text{expression}$. 其中: $S_1 \approx S_2$ 称为递归定义中的元变量 (metavariable).
- ③ 由以上规则在有限步生成的字符串构成factor、 term和expression集合,称为语言:每个语言中的元素称为语句:

XL 语言的语句 --- 归纳过程

- NUM_OR_ID 是 factor 的语句,由①.
- NUM_OR_ID 代入到条款②中的 S₁ 和 S₂ 得: NUM_OR_ID PLUS NUM OR ID 世是 term 和 expression 的语句.
- 由②, NUM_OR_ID PLUS NUM_OR_ID 是 expression 语言的语句.

设: $\Sigma = \{NUM_OR_ID, PLUS, TIMES, LP, RP\}$, factor, term, expression $\subset \Sigma *$ 递归定义如下:

- ② 归纳条款: $S_1 \in \text{expression}$, then $\text{LP } S_1 \text{ RP} \in \text{factor}$. if $S_1 \in \text{factor}$, $S_2 \in \text{term}$, then S_1 , $S_2 \text{ TIMES } S_1 \in \text{term}$. if $S_1 \in \text{term}$, $S_2 \in \text{expression}$, then S_1 , $S_2 \in \text{PLUS } S_1 \in \text{expression}$. 其中: S_1 和 S_2 和为递归定义中的元变量 (metavariable).
- ③ 由以上规则在有限步生成的字符串构成factor、 term和expression集合,称为语言。每个语言中的元素称为语句。

XL 语言的语句 --- 归纳过程

- NUM_OR_ID 是 factor 的语句,由①.
- NUM_OR_ID 代入到条款②中的 S₁ 和 S₂ 得: NUM_OR_ID PLUS NUM OR ID 世是 term 和 expression 的语句.
- 由②, NUM OR ID PLUS NUM OR ID 是 expression 语言的语句

设: Σ = {NUM_OR_ID, PLUS, TIMES, LP, RP}, factor, term, expression ⊂ Σ*锑归定义如下:

- ② 归纳条款: $S_1 \in \text{expression}$, then $\text{LP } S_1 \text{ RP} \in \text{factor}$. if $S_1 \in \text{factor}$, $S_2 \in \text{term}$, then S_1 , $S_2 \in \text{TIMES}$ $S_1 \in \text{term}$. if $S_1 \in \text{term}$, $S_2 \in \text{expression}$, then S_1 , $S_2 \in \text{PLUS}$ $S_1 \in \text{expression}$. 其中: $S_1 \neq S_2$ 称为递归定义中的元变量 (metavariable).
- ③ 由以上规则在有限步生成的字符串构成factor、 term和expression集合,称为语言。每个语言中的元素称为语句。

XL 语言的语句 --- 归纳过程

- NUM_OR_ID 是 factor 的语句,由①.
- NUM_OR_ID 代入到条款②中的 S₁ 和 S₂ 得: NUM_OR_ID PLUS NUM_OR_ID 世是 term 和 expression 的语句.
- 由②, NUM_OR_ID PLUS NUM_OR_ID 是 expression 语言的语句.

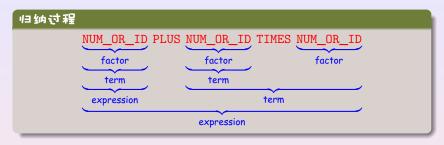
设: $\Sigma = \{NUM_OR_ID, PLUS, TIMES, LP, RP\}$, factor, term, expression $\subset \Sigma *$ 递归定义如下:

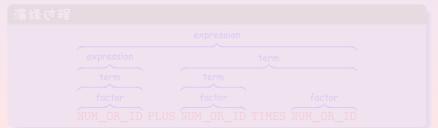
- ② 归纳条款: $S_1 \in \text{expression}$, then $\text{LP } S_1 \text{ RP} \in \text{factor}$. if $S_1 \in \text{factor}$, $S_2 \in \text{term}$, then S_1 , $S_2 \in \text{TIMES } S_1 \in \text{term}$. if $S_1 \in \text{term}$, $S_2 \in \text{expression}$, then S_1 , $S_2 \in \text{PLUS } S_1 \in \text{expression}$. 其中: $S_1 \in S_2$ 称为递归定义中的元变量 (metavariable).
- ③ 由以上规则在有限步生成的字符串构成factor、 term和expression集合,称为语言。每个语言中的元素称为语句。

XL 语言的语句 --- 归纳过程

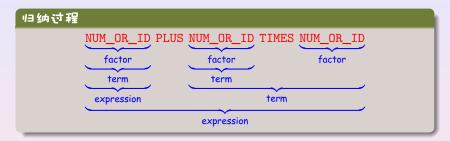
- NUM_OR_ID 是 factor 的语句,由①.
- NUM_OR_ID 代入到条款②中的 S₁ 和 S₂ 得: NUM_OR_ID PLUS NUM_OR_ID 世是 term 和 expression 的语句.
- 由②, NUM_OR_ID PLUS NUM_OR_ID 是 expression 语言的语句.

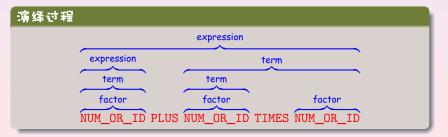
形式化语言的住成过程





形式化语言的生成过程





XL 语言的谦归定义

设: Σ = {NUM_OR_ID, PLUS, TIMES, LP, RP}, factor, term, expression ⊂ Σ* 递归定义如下:

- ① 归纳基础: NUM_OR_ID ∈ factor.
 - 2 归纳条款: $S_1 \in \text{expression}$, then LP $S_1 \text{ RP} \in \text{factor}$. if $S_1 \in \text{factor}$, $S_2 \in \text{term}$, then S_1 , $S_2 \text{ TIMES } S_1 \in \text{term}$. if $S_1 \in \text{term}$, $S_2 \in \text{expression}$, then S_1 , $S_2 \text{ PLUS } S_1 \in \text{expression}$
- 重写规则 --- 产住式

XL 语言的谦归定义

 $\mathfrak{P}: \Sigma = \{NUM_OR_ID, PLUS, TIMES, LP, RP\}, factor, term, expression$

⊆Σ* 锑归定义如下: ① 归纳基础: NUM OR_ID ∈ factor.

2 归纳条款: $S_1 \in \text{expression}$, then LP S_1 RP \in factor. if $S_1 \in \text{factor}$, $S_2 \in \text{term}$, then S_1 , S_2 TIMES $S_1 \in \text{term}$. if $S_1 \in \text{term}$, $S_2 \in \text{expression}$, then S_1 , S_2 PLUS $S_1 \in \text{expression}$

重复初则 ___ 产生式



XL 语言的谦归定义

设: Σ = {NUM_OR_ID, PLUS, TIMES, LP, RP}, factor, term, expression ⊂ Σ* 递归定义如下:

- ② 归纳条款: $S_1 \in \text{expression}$, then $\text{LP } S_1 \text{ RP} \in \text{factor}$. if $S_1 \in \text{factor}$, $S_2 \in \text{term}$, then S_1 , $S_2 \in \text{TIMES } S_1 \in \text{term}$. if $S_1 \in \text{term}$, $S_2 \in \text{expression}$, then S_1 , $S_2 \in \text{PLUS } S_1 \in \text{expression}$.

上下文无关文法的设计

Example: XL 语言的递归定义

XL 语言的谦归定义

 \mathfrak{P} : $\Sigma = \{NUM_OR_ID, PLUS, TIMES, LP, RP\}, factor, term, expression$

C ∑* 递归定义如下:

- ∑* 递归定义如下:
 □ 归纳基础: NUM_OR_ID ∈ factor.
 ② 归纳条款: S₁ ∈ expression, then LP S₁ RP ∈ factor.
 if S₁ ∈ factor, S₂ ∈ term, then S₁, S₂ TIMES S₁ ∈ term.
 if S₁ ∈ term, S₂ ∈ expression, then S₁, S₂ PLUS S₁ ∈ expression.

XL 语言的谦归定义

上下支无关支法

设: Σ = {NUM_OR_ID, PLUS, TIMES, LP, RP}, factor, term, expression C ∑* 递归定义如下:

- ∑* 递归定义如下:
 □ 归纳基础: NUM_OR_ID ∈ factor.
 ② 归纳条款: S₁ ∈ expression, then LP S₁ RP ∈ factor.
 if S₁ ∈ factor, S₂ ∈ term, then S₁, S₂ TIMES S₁ ∈ term.
 if S₁ ∈ term, S₂ ∈ expression, then S₁, S₂ PLUS S₁ ∈ expression.

- expression → expression PLUS term.

XL 语言的谦归定义

上下支充差支法

 \mathfrak{P} : $\Sigma = \{NUM_OR_ID, PLUS, TIMES, LP, RP\}, factor, term, expression$ C ∑* 递归定义如下:

- ∑* 递归定义如下:
 □ 归纳基础: NUM_OR_ID ∈ factor.
 ② 归纳条款: S₁ ∈ expression, then LP S₁ RP ∈ factor.
 if S₁ ∈ factor, S₂ ∈ term, then S₁, S₂ TIMES S₁ ∈ term.
 if S₁ ∈ term, S₂ ∈ expression, then S₁, S₂ PLUS S₁ ∈ expression.

- expression --> expression PLUS term.
- expression → term.

XL 语言的谦归定义

上下支充差支法

设: Σ = {NUM_OR_ID, PLUS, TIMES, LP, RP}, factor, term, expression

- C ∑* 递归定义如下:
- ∑* 递归定义如下:
 □ 妈納基础: NUM_OR_ID ∈ factor.
 ② 妈纳条款: S₁ ∈ expression, then LP S₁ RP ∈ factor.
 if S₁ ∈ factor, S₂ ∈ term, then S₁, S₂ TIMES S₁ ∈ term.
 if S₁ ∈ term, S₂ ∈ expression, then S₁, S₂ PLUS S₁ ∈ expression.

- expression --> expression PLUS ferm.
- \bullet expression \longleftrightarrow term.
- term → term TIMES factor.

XL 语言的谦归定义

上下支充差支法

设: Σ = {NUM_OR_ID, PLUS, TIMES, LP, RP}, factor, term, expression C ∑* 递归定义如下:

- ∑* 递归定义如下:
 □ 妈纳基础: NUM_OR_ID ∈ factor.
 ② 妈纳条款: S₁ ∈ expression, then LP S₁ RP ∈ factor.
 if S₁ ∈ factor, S₂ ∈ term, then S₁, S₂ TIMES S₁ ∈ term.
 if S₁ ∈ term, S₂ ∈ expression, then S₁, S₂ PLUS S₁ ∈ expression.

- expression → expression PLUS term.
- expression → term.
- term → term TIMES factor.
- term → factor:

XL 语言的谦归定义

上下支充差支法

设: Σ = {NUM_OR_ID, PLUS, TIMES, LP, RP}, factor, term, expression C ∑* 递归定义如下:

- ① 归纳基础: NUM OR ID∈factor.
- 递归定义如下: 归纳基础: NUM_OR_ID \in factor. 归纳基歌: $S_1 \in$ expression, then LP S_1 RP \in factor. if $S_1 \in$ factor, $S_2 \in$ term, then S_1 , S_2 TIMES $S_1 \in$ term. if $S_1 \in$ term, $S_2 \in$ expression, then S_1 , S_2 PLUS $S_1 \in$ expression. ② 归纳条款:S₁∈expression,then LP S₁RP∈ factor.

- expression → expression PLUS term.
- expression → term.
- ◆ term → term TIMES factor.
- term → factor:
- factor → NUM_OR_ID.

XL 语言的谦归定义

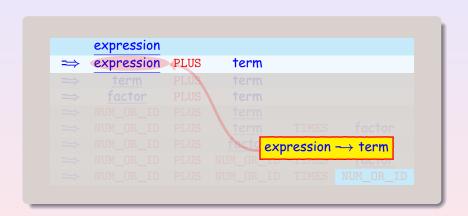
上下支充差支法

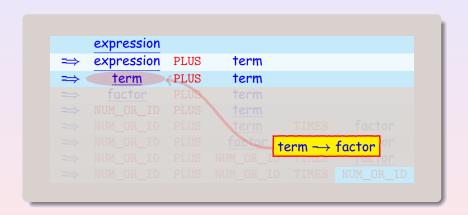
设: Σ = {NUM_OR_ID, PLUS, TIMES, LP, RP}, factor, term, expression C ∑* 递归定义如下:

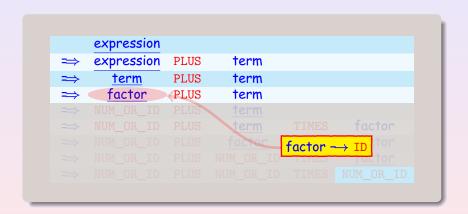
- ① 归纳基础: NUM_OR_ID∈factor.
- ② 归纳条款: S₁∈ expression, then LP S₁ RP∈ factor. if $S_1 \in \text{factor}$, $S_2 \in \text{term}$, then S_1 , $S_2 \text{ TIMES } S_1 \in \text{term}$. if $S_1 \in \text{term}$, $S_2 \in \text{expression}$, then S_1 , $S_2 \neq \text{PLUS}$ $S_1 \in \text{expression}$

- expression → expression PLUS term.
- expression → term.
- term → term TIMES factor.
- term → factor:
- factor \longrightarrow NUM_OR_ID.
- factor \rightarrow LP expression RP.

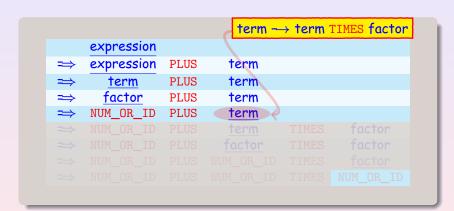


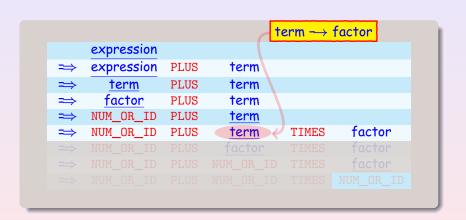


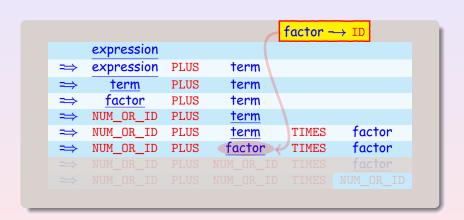


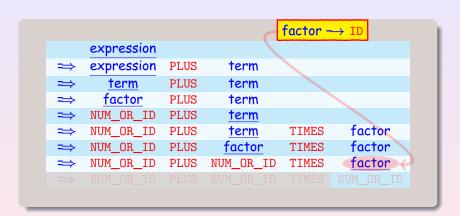


重写 (Rewriting)



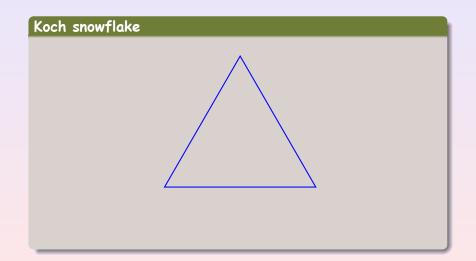


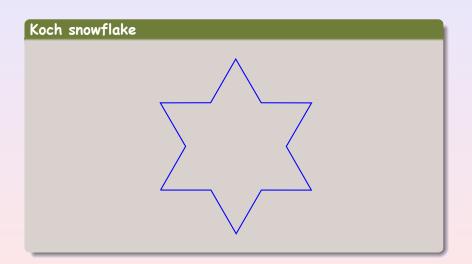


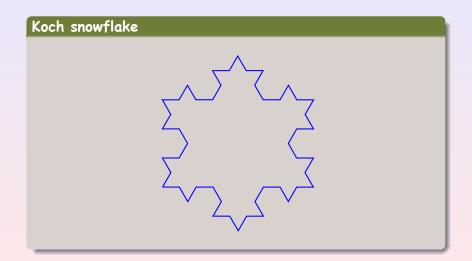


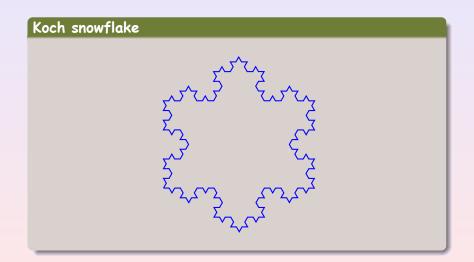
```
expression
     expression
                   PLUS
                              term
\Rightarrow
        term
                   PLUS
                             term
\Rightarrow
       factor
                  PLUS
                             term
\Rightarrow
     NUM OR ID PLUS
                              term
                                                   factor
     NUM OR ID
                   PLUS
\Rightarrow
                              term
                                        TIMES
\Rightarrow
     NUM OR ID PLUS
                             factor
                                        TIMES
                                                   factor
                                                   factor
\Rightarrow
     NUM OR ID PLUS
                          NUM_OR_ID TIMES
     NUM OR ID
                   PLUS
                          NUM OR ID
\Rightarrow
                                        TIMES
                                                 NUM OR ID
```

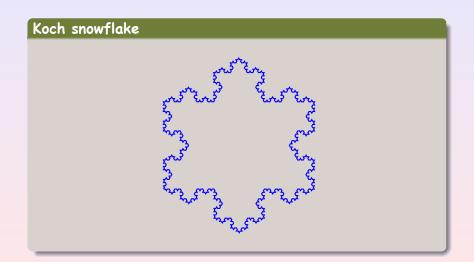
上下支充关支法的设计











上下文无关文法 (Context-Free Grammar, CFG)

CFG 定义

G = (T, N, S, P), 其中:

- T是有限字母表,其元素称 为终结符号(terminals);
- N是有限语法成分标记集合, N∩T=∅, 称为非终结符号 (non-terminals), 相当于递归定 义的元变量;
- S∈N称为文法开始符号;
- P = {S_i → w | S_i ∈ N, w ∈ (T∪N)*} 是一个有限的集合,其正麦称为产生式。

Example

注释

为了书写方便,在后续的讲义中,expression 简记为 exp, factor 简记为 fac, NUM_OR_ID 简记为ID.

- Backus-Naur Form (BNF. 源于 Aacol60).
- Terminal = 字母; Non-terminal = 递归定义中的 metavariable
- 产生式左边 (Left Hand Side, LHS)—定是唯一的一个 非终结符。
- 产生式右边(Right Hand Side, RHS)可以是空串,称为空产生式
- 一个非终结符可以有多个产生式对应, 可用``|'' 连接 其对应的 RHS.

- Backus-Naur Form (BNF, 源于 Agcol60).
- Terminal = 字母; Non-terminal = 递归定义中的 metavariable.
- 产生式左边 (Left Hand Side, LHS)—定是唯一的一个 非终结符。
- 产生式右边(Right Hand Side, RHS)可以是空串,称为空产生式。
- 一个非终结符可以有多个产生式对应, 可用``|'' 连接 其对应的 RHS

- Backus-Naur Form (BNF, 源于 Agcol60).
- Terminal = 字母; Non-terminal = 递归定义中的 metavariable.
- 产生式左边 (Left Hand Side, LHS)—定是唯一的一个 非终结符。
- 产生式右边(Right Hand Side, RHS)可以是空串,称为空产生式。
- 一个非终结符可以有多个产生式对应, 可用``|'' 连接 其对应的 RHS

- Backus-Naur Form (BNF, 源于 Agcol60).
- Terminal = 字母; Non-terminal = 递归定义中的 metavariable.
- 产生式左边 (Left Hand Side, LHS)—定是唯一的一个 非终结符.
- 产生式右边(Right Hand Side, RHS)可以是空串,称为空产生式。
- 一个非终结符可以有多个产生式对应, 可用``|''连接 其对应的 RHS.

- Backus-Naur Form (BNF, 源于 Agcol60).
- Terminal = 字母; Non-terminal = 递归定义中的 metavariable.
- 产生式左边 (Left Hand Side, LHS)—定是唯一的一个 非终结符.
- 产生式右边(Right Hand Side, RHS)可以是空串、称为空产生式。
- 一个非终结符可以有多个产生式对应, 可用 ``|'' 连接 其对应的 RHS.

- Backus-Naur Form (BNF, 源于 Agcol60).
- Terminal = 字母; Non-terminal = 递归定义中的 metavariable
- 产生式左边 (Left Hand Side, LHS)—定是唯一的一个 非终结符.
- 产生式右边(Right Hand Side, RHS)可以是空串、称为空产生式。
- 一个非终结符可以有多个产生式对应, 可用``|'' 连接 其对应的 RHS.

CFG 所描述的语言 --- 推导

```
Example of XL
T = \{ ID, PLUS, \}
       TIMES, LP, RP };
N = \{ \exp, term, fac \};
 S = \exp;
P:
 1/ exp -> exp PLUS term
 2/
        | term
 3/ term -> term TIMES fac
4/
         | fac
5/ fac -> ID
 6/
        | LP exp RP
```

```
推导过程
     exp
 \Rightarrow exp PLUS term
                               by 1/
                              by 2/
 ⇒ term PLUS term
                              by 4/
 ⇒ fac PLUS term
                              by 5/
 ⇒ ID PLUS term
 \Rightarrow ID PLUS <u>term</u> TIMES fac by 3/
                              by 4/
 ⇒ ID PLUS fac TIMES fac
 \Rightarrow ID PLUS ID TIMES fac
                              by 5/
                              by 5/
 ⇒ ID PLUS ID TIMES ID
```

- 设 $G \in CFG$, $\alpha A\beta \in (T \cup N)^*$, $A \in N$, $A \rightarrow \gamma \in P$, 则 $\alpha A\beta \Rightarrow \alpha \gamma \beta$ 是一 光 柱 星 (one step derivation) 。 $\alpha \in A \cap A$ な 発 な 発 量 の た な は 星 的 と
 - 下文环境。
 - ② ⇒⊆ (T∪N)*×(T∪N)* 是 (T∪N)* 上的二元关系.

推导

- □ 设 G 是 CFG, αAβ ∈ (T∪N)*, A ∈ N, A → y ∈ P, 则 $\alpha A\beta \Rightarrow \alpha \gamma \beta$ 是一步推导 (one step derivation), a和 B 称推导的上 下支环境.

- □ 设 G 是 CFG, αAβ ∈ (T∪N)*, A ∈ N, A → y ∈ P, 则 $aA\beta \Rightarrow ay\beta$ 是一步推导 (one step derivation), a和 B 称推导的上 下支环境.
- ② ⇒C (T∪N)* × (T∪N)* 是 (T∪N)* 上的二元关系.

- ① 设 $G \in CFG$, $\alpha A\beta \in (T \cup N)^*$, $A \in N$, $A \longrightarrow \gamma \in P$, 则 $\alpha A\beta \Rightarrow \alpha \gamma \beta$ 是一步推导 (one step derivation), $\alpha 和 \beta$ 称推导的上下文环境
- ② ⇒⊆ (T∪N)* × (T∪N)* 是 (T∪N)* 上的二元关系.

- 1 ID PLUS term TIMES fac \Rightarrow ID PLUS fac TIMES fac.
 - а в а
- 2 PLUS term TIMES fac \Rightarrow PLUS fac TIMES fac.

- ① 设 $G \in CFG$, $\alpha A\beta \in (T \cup N)^*$, $A \in N$, $A \longrightarrow \gamma \in P$, 则 $\alpha A\beta \Rightarrow \alpha \gamma \beta$ 是一步推导 (one step derivation), $\alpha 和 \beta$ 称推导的上下文环境.
- ② ⇒⊆ (T∪N)* × (T∪N)* 是 (T∪N)* 上的二元关系.

- 1 ID PLUS term TIMES fac \Rightarrow ID PLUS fac TIMES fac.
- 2 PLUS term TIMES fac \Rightarrow PLUS fac TIMES fac.

上下支充差支法

- □ 设 G 是 CFG, αAβ ∈ (T∪N)*, A ∈ N, A → y ∈ P, 则 $aA\beta \Rightarrow ay\beta$ 是一步推导 (one step derivation), a和 B 称推导的上 下支环境.
- ② ⇒C (T∪N)* × (T∪N)* 是 (T∪N)* 上的二元关系.

- 1 IDPLUS $\underline{\text{term}}$ TIMES fac \Rightarrow IDPLUS $\underline{\text{fac}}$ TIMES fac.
- 2 PLUS $\underline{\text{term}}$ TIMES $\underline{\text{fac}} \Rightarrow \underline{\text{PLUS}}$ $\underline{\underline{\text{fac}}}$ TIMES $\underline{\text{fac}}$.

$$aA\beta \stackrel{0}{\Rightarrow} aA\beta$$

$$\stackrel{*}{\Rightarrow} \triangleq \bigcup_{i=0}^{i} \stackrel{i}{\Rightarrow}$$

n 步推导

① 0 歩推导 (⇒):

 $\alpha A\beta \stackrel{0}{\Rightarrow} \alpha A\beta$

。 ⇒是 (T∪N)*上的恒等关系.

- ② n 步维导 (♣): 设 δ ♣ αAβ 是 n 步维导,则 δ ♣ αAβ ⇒ αγβ ≜ δ ➡ αγβ 为 n+1 步维导, ♣ 是关系 ⇒ 的 n次合成。
- **3** ⇒

 $\stackrel{*}{\Rightarrow} \triangleq \bigcup_{i=0}^{\infty} \stackrel{i}{\Rightarrow}$

Example

n步维导

① 0 歩推导 (⇒):

 $\alpha A\beta \stackrel{0}{\Rightarrow} \alpha A\beta$

⇒是 (T∪N)*上的恒等关系.

② n 步维导 ($\stackrel{\rightarrow}{\rightarrow}$): 设 $\delta \stackrel{\rightarrow}{\rightarrow} \alpha A \beta$ 是 n 步维导,则 $\delta \stackrel{\rightarrow}{\rightarrow} \alpha A \beta \Rightarrow \alpha \gamma \beta \triangleq \delta \stackrel{n+1}{\rightarrow} \alpha \gamma \beta$ 为 n+1 步维导, $\stackrel{\rightarrow}{\rightarrow}$ 是关系 \Rightarrow 的 n次合成。

Evampla

n 步维导

上下支充关支法

① 0 步维导(⇒):

 $aAB \stackrel{0}{\Rightarrow} aAB$

⇒是 (T∪N)*上的恒等关系.

- ② n 步椎导 (♣): 设 δ ♣ αAβ 是 n 步椎导, 则 $\delta \stackrel{\mathsf{n}}{\Rightarrow} aAb \Rightarrow avb \triangleq \delta \stackrel{\mathsf{n+1}}{\Rightarrow} avb$ 为 n+1 步维导. $\stackrel{\triangle}{\rightarrow}$ 是关系 \Rightarrow 的 n 次合成.
- **3** * ...

$$\stackrel{*}{\Rightarrow} \triangleq \bigcup_{i=0}^{i} \stackrel{i}{\Rightarrow}$$

n 步维导

① 0 步维导(⇒):

$$aA\beta \stackrel{0}{\Rightarrow} aA\beta$$

⇒是 (T∪N)*上的恒等关系.

- ② n 步维导 (♣): 设 δ ♣ αAβ 是 n 步维导, 则 $\delta \stackrel{\mathsf{n}}{\Rightarrow} aAb \Rightarrow avb \triangleq \delta \stackrel{\mathsf{n+1}}{\Rightarrow} avb$ 为 n+1 步维导. $\stackrel{\triangle}{\rightarrow}$ 是关系 \Rightarrow 的 n 次合成.
- **3** *

$$\stackrel{*}{\Rightarrow} \triangleq \bigcup_{i=0}^{i} \stackrel{i}{\Rightarrow}$$

n 步维导

① 0 步维导(⇒):

$$aA\beta \stackrel{0}{\Rightarrow} aA\beta$$

⇒是 (T∪N)*上的恒等关系.

- ② n 步椎导 (♣): 设 δ ♣ αAβ 是 n 步椎导, 则 $\delta \stackrel{\mathsf{n}}{\Rightarrow} aAb \Rightarrow avb \triangleq \delta \stackrel{\mathsf{n+1}}{\Rightarrow} avb$ 为 n+1 步维导. $\stackrel{\triangle}{\rightarrow}$ 是关系 \Rightarrow 的 n 次合成.
- **3** *:

$$\stackrel{*}{\Rightarrow} \triangleq \bigcup_{i=0}^{i} \stackrel{i}{\Rightarrow}$$

- 1 exp $\stackrel{*}{\Rightarrow}$ ID PLUS ID TIMES ID.

n 步推导

① 0 歩堆昇 (⇒):

$$\alpha A\beta \stackrel{0}{\Rightarrow} \alpha A\beta$$

⇒是 (T∪N)*上的恒等关系.

- ② n 步维导 ($\stackrel{\wedge}{\rightarrow}$): 设 $\delta \stackrel{\wedge}{\rightarrow} \alpha A \beta$ 是 n 步维导,则 $\delta \stackrel{\wedge}{\rightarrow} \alpha A \beta \Rightarrow \alpha \gamma \beta \triangleq \delta \stackrel{n+1}{\rightarrow} \alpha \gamma \beta$ 为 n+1 步维导, $\stackrel{\wedge}{\rightarrow}$ 是关系 \Rightarrow 的 n 次合成。
- 3 *;

$$\stackrel{*}{\Rightarrow} \triangleq \bigcup_{i=0}^{i} \stackrel{i}{\Rightarrow}$$

- exp

 * ID PLUS ID TIMES ID.
- 2 term exp ^{*} term ID PLUS ID TIMES ID.

注释

上下文无关的含义

- 设 A ⇒ u ៧ ∀α,β ∈ (T∪N)*,αAβ ⇒ αuβ.
- 拳 是 (TUN)* 上的关系 ⇒ 的自反传递闭包.

上下文无关的含义

上下文无关的含义

- 误 $A \stackrel{*}{\Rightarrow} u$ 则 $\forall \alpha, \beta \in (T \cup N)^*, \alpha A \beta \stackrel{*}{\Rightarrow} \alpha u \beta.$



上下文无关的含义

- 误 $A \stackrel{*}{\Rightarrow} u$ 则 $\forall \alpha, \beta \in (T \cup N)^*, \alpha A \beta \stackrel{*}{\Rightarrow} \alpha u \beta.$
- 拳 是 (T∪N)* 上的关系 ⇒ 的自反传递闭包.



上下文无关的含义

- 设 A ⇒ u ⋈ ∀α, β ∈ (T∪N)*, αAβ ⇒ αuβ.
- 🕇 是 (T∪N)* 上的关系 ⇒ 的自反传递闭包.

上下支充差支法

上下文无关的含义

- 设 $A \longrightarrow Y$, $\nabla^{1} \forall \alpha, \beta \in (T \cup N)^{*}$, $\alpha A \beta \stackrel{1}{\Rightarrow} \alpha Y \beta$.
- 译 $A \stackrel{*}{\Rightarrow} u$ 则 $\forall \alpha, \beta \in (T \cup N)^*, \alpha A \beta \stackrel{*}{\Rightarrow} \alpha u \beta.$
- * 是 (T∪N)*上的关系 ⇒ 的自反传递闭包.

- ID PLUS $\underline{\text{term}}$ TIMES $\underline{\text{fac}} \Rightarrow \text{ID PLUS fac TIMES fac}$ ID TIMES $\underline{\text{term}}$ TIMES $\underline{\text{fac}} \Rightarrow \text{ID TIMES fac TIMES fac}$.
- 因为 exp ⇒ ID PLUS ID TIMES ID, 则:
 term PLUS exp ⇒ term PLUS ID PLUS ID TIMES ID
 产生式如同公理 维星结果如同定理

上下支充差支法

上下文无关的含义

- 译 $A \longrightarrow Y$, ምነ $\forall \alpha, \beta \in (T \cup N)^*$, $\alpha A\beta \stackrel{1}{\Rightarrow} \alpha Y\beta$.
- 误 $A \stackrel{*}{\Rightarrow} u$ 则 $\forall \alpha, \beta \in (T \cup N)^*, \alpha A \beta \stackrel{*}{\Rightarrow} \alpha u \beta.$
- 🕇 是 (T∪N)* 上的关系 ⇒ 的自反传递闭包.

- ID PLUS term TIMES fac ⇒ ID PLUS fac TIMES fac ID TIMES term TIMES fac \Rightarrow ID TIMES fac TIMES fac.
- 因为 exp ⇒ ID PLUS ID TIMES ID, 则: term PLUS exp ⇒ term PLUS ID PLUS ID TIMES ID 产生式如同公理,推导结果如同定理.

上下文无关语言

定义



上下文无关语言

定义

- 设 G 是 CFG、G 所住成的语言 L(G) ⊆ T* 定义为: $L(G) = \{\alpha \mid \alpha \in T^* \land S \stackrel{*}{\Rightarrow} \alpha\}$
 - α 称为语言 L(G) 的语句.



上下支充关支法的设计

上下文无关语言

定义

- 设 G 是 CFG、G 所住成的语言 L(G) ⊂ T* 定义为: $L(G) = \{\alpha \mid \alpha \in T^* \land S \stackrel{*}{\Rightarrow} \alpha\}$ α 称为语言 L(G) 的语句.
- L C T* 是上下文无关语言 (CFL) iff ∃ CGF G and L = L(G).



上下文无关文法的设计

上下文无关语言

定义

- 设 G 是 CFG、G 所住成的语言 L(G) ⊂ T* 定义为: $L(G) = \{\alpha \mid \alpha \in T^* \land S \stackrel{*}{\Rightarrow} \alpha\}$ α 称为语言 L(G) 的语句.
- L C T* 是上下文无关语言 (CFL) iff ∃ CGF G and L = L(G).

上下文无关文法的设计

上下文无关语言

定义

- 设 G 是 CFG、G 所住成的语言 L(G) ⊂ T* 定义为: $L(G) = \{\alpha \mid \alpha \in T^* \land S \stackrel{*}{\Rightarrow} \alpha\}$ α 称为语言 L(G) 的语句.
- L C T* 是上下文无关语言 (CFL) iff ∃ CGF G and L = L(G).

- L(XL) = { 算术表达式集合}.

上下文无关语言

定义

- 设 G 是 CFG, G 所住成的语言 L(G) ⊆ T* 定义为:
 L(G) = {a | a ∈ T* ∧ S ⇒ a}
 a 称为语言 L(G) 的语句.
- L ⊆ T* 是上下文无关语言 (CFL) iff ∃ CGF G and L = L(G).

- L(XL) = { 算术表达式集合}.
- IDPLUS ID TIMES ID 是 XL 语言的语句;
- ID PLUS TIMES ID 不是 XL 语言的语句.

定义

上下文无关文法

- 设 G 是 CFG、G 所住成的语言 L(G) ⊂ T* 定义为: $L(G) = \{\alpha \mid \alpha \in T^* \land S \stackrel{*}{\Rightarrow} \alpha\}$ α 称为语言 L(G) 的语句.
- L C T* 是上下文无关语言 (CFL) iff ∃ CGF G and L = L(G).

- L(XL) = { 算术表达式集合}.
- ID PLUS ID TIMES ID 是 XL 语言的语句;
- ID PLUS TIMES ID 不是 XL 语言的语句.

- 设 $G \not\in CFG$, $a \in (T \cup N)^* \not\in G$ 的一个句型 (sentential form) iff $S \stackrel{\Rightarrow}{\Rightarrow} a$.
- 语句是特殊的句型.
- 句型中至少有个子串是某一产生式的右边的文法符号串。
- 如果句型不正确,则永远不可能推出语句.
- 句型分析是语法分析的核心.

- 译 $G \not\in CFG$, $a \in (T \cup N)^* \not\in G$ 的一个句型 (sentential form) iff $S \stackrel{*}{\Rightarrow} a$.
- 语句是特殊的句型
- 句型中至少有个子串是某一产生式的右边的文法符号串。
- 如果句型不正确,则永远不可能推出语句,
- 旬型分析是语法分析的核心.

- 译 $G \in CFG$, $a \in (T \cup N)^* \in G$ 的一个句型 (sentential form) iff $S \stackrel{*}{\Rightarrow} a$.
- 语句是特殊的句型.
- 句型中至少有个子串是某一产生式的右边的文法符号串。
- 如果句型不正确,则永远不可能推出语句.
- 旬型分析是语法分析的核心。

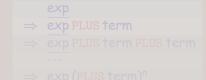
- 译 $G \not\in CFG$, $a \in (T \cup N)^* \not\in G$ 的一个句型 (sentential form) iff $S \stackrel{*}{\Rightarrow} a$.
- 语句是特殊的句型.
- 句型中至少有个子串是某一产生式的右边的文法符号串。
- 如果句型不正确,则永远不可能推出语句
- 句型分析是语法分析的核心。

上下支充差支法

定义及性质

- 设 G 是 CFG, a ∈ (T∪N)* 是 G 的一个句型 (sentential form) iff $S \stackrel{*}{\Rightarrow} a$.
- 语句是特殊的句型.
- 句型中至少有个子串是某一产生式的右边的文法符号 串
- 如果句型不正确,则永远不可能推出语句。

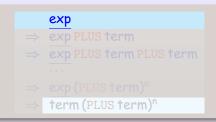
- 设 $G \in CFG$, $a \in (T \cup N)^* \in G$ 的一个句型 (sentential form) iff $S \stackrel{*}{\Rightarrow} a$.
- 语句是特殊的句型.
- 句型中至少有个子串是某一产生式的右边的文法符号串。
- 如果句型不正确,则永远不可能推出语句.
- 句型分析是语法分析的核心.



⇒ term (PLUS term)ⁿ

XI 语言每个非终结终对应的句形

- (exp|term)(PLUS term)* 是 exp 的句型.
- 即 exp 只能推出 term 或者多 term 中间用 PLUS 链接的串
 - 同样 (term|fac) (TIMES fac)* 是 term 的句型.
 - 即 term 只能是一个 fac 或者多 fac 中间用 TIMES 链接的 串.
 - fac 的句型是 LP exp RP ID



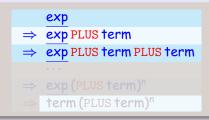
XL 语言每个非络结符对应的句型

- (exp|term)(PLUS term)* 是 exp 的句型.
- 即 exp 只能推出 term 或者多 term 中间用 PLUS 链接的串
- 同样 (term|fac)(TIMES fac)* 是 term 的句型。
- 即 term 只能是一个 fac 或者多 fac 中间用 TIMES 链接的 串.
- fac 的句型是 LP exp RP ID



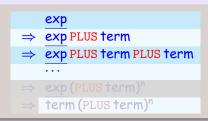
XL 语言每个非终结符对应的句型

- (exp|term)(PLUS term)* 是 exp 的句型.
- 即 exp 只能推出 term 或者多 term 中间用 PLUS 链接的串
- 同样 (term|fac) (TIMES fac)* 是 term 的句型
- 即 term 只能是一个 fac 或者多 fac 中间用 TIMES 链接的 串。
- fac 的句型是 LP exp RP ID.



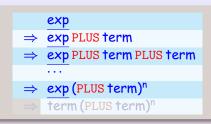
XI 语言每个非终结符对应的句型

- (explterm)(PLUS term)* 是 exp 的句型.
- 即 exp 只能推出 term 或者多 term 中间用 PLUS 链接的串
- 同样 (term|fac)(TIMES fac)* 是 term 的句型.
- EP term 只能是一个 fac 或者多 fac 中间用 TIMES 链接的 串
- fac 的句型是 LP exp RP ID.



XI 语言每个非终结符对应的句型

- (explterm)(PLUS term)* 是 exp 的句型.
- 即 exp 只能推出 term 或者多 term 中间用 PLUS 链接的串
- 同样 (term|fac) (TIMES fac)* 是 term 的句型.
- EP term 只能是一个 fac 或者多 fac 中间用 TIMES 链接的 串
- fac 的句型是 LP exp RP ID



XL 语言每个非终结符对应的句型

- (exp|term)(PLUS term)* 是 exp 的句型.
- 即 exp 只能推出 term 或者多 term 中间用 PLUS 链接的串
- 同样 (term|fac) (TIMES fac)* 是 term 的句型.
- 即 term 只能是一个 fac 或者多 fac 中间用 TIMES 链接的 串.
- fac 的句型是 LP exp RP ID



- \Rightarrow exp PLUS term
- $\Rightarrow \frac{\exp PLUS \text{ term PLUS term}}{\dots}$
- \Rightarrow exp (PLUS term)ⁿ
- \Rightarrow term (PLUS term)ⁿ

XL 语言每个非络结符对应的句型

- (exp|term)(PLUS term)* 是 exp 的句型.
- 即 exp 只能推出 term 或者多 term 中间用 PLUS 链接的串
- 同样 (term|fac) (TIMES fac)* 是 term 的句型.
- 即 term 只能是一个 fac 或者多 fac 中间用 TIMES 链接的 串.
- fac 的句型是 LP exp RP | ID

exp

- \Rightarrow exp PLUS term
- ⇒ exp PLUS term PLUS term
- \Rightarrow exp (PLUS term)ⁿ
- \Rightarrow term (PLUS term)ⁿ

XL 语言每个非终结符对应的句型

- (exp|term)(PLUS term)* 是 exp 的句型.
- 即 exp 只能推出 term 或者多 term 中间用 PLUS 链接的串.
- 同样 (term|fac)(TIMES fac)* 是 term 的句型.
- 即 term 只能是一个 fac 或者多 fac 中间用 TIMES 链接的 串.
- fac 的句型是 LP exp RP ID.

- exp PLUS exp 和 exp TIMES term 不是句型.
- term TIMES term 和 term PLUS exp 世不是句型
- ID TIMES ID PLUS ID
 term
 exp
 exp
 exp
- 语句 IDTIMES ID PLUS ID 只能是由句型 exp PLUS term 生成的句型 term TIMES fac PLUS term 生成,即 TIMES 两边的 ID 由同一个非终结符住成,PLUS 两边的 ID 不可能由同一个非终结符住成,因为生成 PLUS 的非终结符只有 exp,但 ID TIMES exp 不是句型。
- XL 语言的乘法运算先于加法结合,即乘法的优先级高 工加法

- exp PLUS exp 和 exp TIMES term 不是句型.

- 20/65 -

- exp PLUS exp 和 exp TIMES term 不是句型.
- term TIMES term 和 term PLUS exp 也不是句型.

- 语句 ID TIMES ID PLUS ID 只能是由句型 exp PLUS term 生成的句型 term TIMES fac PLUS term 生成. 即 TIMES 两边的 ID 由同一个非终结符生成. PLUS 两边的 ID 不可能由同一个非终结符生成. 因为生成 PLUS 的非终结符只有 exp,但 ID TIMES exp 不是句型.
- XL 语言的乘法运算先于加法结合,即乘法的优先级高工机法

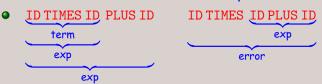
- exp PLUS exp 和 exp TIMES term 不是句型.
- term TIMES term 和 term PLUS exp 世不是句型.
- ID TIMES ID PLUS ID





- 语句 ID TIMES ID PLUS ID 只能是由句型 exp PLUS term 生成的句型 term TIMES fac PLUS term 生成,即 TIMES 两边的 ID 由同一个非终结符住成,PLUS 两边的 ID 不可能由同一个非终结符住成,因为生成 PLUS 的非终结符只有 exp,但 ID TIMES exp 不是句型。
- XL 语言的乘法运算先于加法结合,即乘法的优先级高工加法

- exp PLUS exp 和 exp TIMES term 不是句型.
- term TIMES term 和 term PLUS exp 世不是句型.



- 语句 ID TIMES ID PLUS ID 只能是由句型 exp PLUS term 生成的句型 term TIMES fac PLUS term 生成.即 TIMES 两边的 ID 由同一个非终结符住成. PLUS 两边的 ID 不可能由同一个非终结符住成. 因为住成 PLUS 的非终结符只有 exp,但 ID TIMES exp 不是句型.
- XL 语言的乘法运算先于加法结合,即乘法的优先级高平加法

- 20/65 -

上下支无关支法的设计

XL 语言的运算优先级

- exp PLUS exp 和 exp TIMES term 不是句型.
- term TIMES term 和 term PLUS exp 世不是句型.



- 语句 ID TIMES ID PLUS ID 只能是由句型 exp PLUS term 生 成的句型 term TIMES fac PLUS term 生成. 即 TIMES 两 边的 ID 由同一个非终结符住成, PLUS 两边的 ID 不可能 由同一个非终结符住成, 因为住成 PLUS 的非终结符只 有 exp, 但 ID TIMES exp 不是句型.
- XL 语言的乘法运算先子加法结合、即乘法的优先级高 干加法.

XL 语言的运算结合律

上下文无关文法

- exp error
- 原因:没有 PLUS exp 为了串的句型
- 加法运算左结合
- 同样乘法运算也是左结合的

上下支充关支法



- 原因:没有 PLUS exp 为了串的句型
- 加法运算左结合.
- 同样乘法运算也是左结合的,

XL 语言的运算结合律

上下支充关支法



- 原因:没有 PLUS exp 为了串的句型.

XL 语言的运算结合律

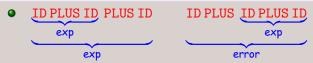
上下支充关支法



- 原因:没有 PLUS exp 为了串的句型.
- 加法运算左结合.

XL 语言的运算结合律

上下支充差支法



- 原因:没有 PLUS exp 为了串的句型.
- 加法运算左结合.
- 同样乘法运算也是左结合的.

文法等价

设 G₁, G₂ 是两个 CFG, G₁ 和 G₂ 等价 iff L(G₁) = L(G₂).

```
Grammar of XL'
T = \{ ID, PLUS, \}
      TIMES, LP, RP );
N = \{exp\};
S = \exp;
P:
1/ exp -> exp PLUS exp
2/ | exp TIMES exp
3/ LP exp RP
    | ID
4/
```

```
Grammar of XL"
T = \{ ID, PLUS, \}
      TIMES, LP, RP };
N = {exp, term, fac};
S = \exp;
P:
1/ exp -> term PLUS exp
2/ | term
3/ term -> fac TIMES term
4/ | fac
5/ fac -> ID
6/ LP exp RP
 /* 乘法优先且右结合 */
```

L(XL) = L(XL') = L(XL'').

等价文法 $G_1 = \begin{cases} S \rightarrow AB \mid CA \\ A \rightarrow a \\ B \rightarrow BC \mid AB \\ C \rightarrow aB \mid b \end{cases}$

等价文法

$$G_1 = \begin{cases} S \rightarrow AB \mid CA \\ A \rightarrow a \\ B \rightarrow BC \mid AB \\ C \rightarrow aB \mid b \end{cases}$$

$$G_2 = \begin{cases} S \rightarrow CA \\ A \rightarrow a \\ C \rightarrow b \end{cases}$$

- B 不可能推出终结符来, 是无用符号.
- 消除 B 即得等价支法 G₂

等价文法

$$G_1 = \begin{cases} S \rightarrow AB \mid CA \\ A \rightarrow a \\ B \rightarrow BC \mid AB \\ C \rightarrow aB \mid b \end{cases}$$

$$G_2 = \begin{cases} S \rightarrow CA \\ A \rightarrow a \\ C \rightarrow b \end{cases}$$

- B 不可能推出终结符来,是无用符号.
- 消除 B 即得等价文法 G₂.

上下支充差支法

推导的不确定性

两个不确定性

- 选择句型中的非终结符展升。
- 选择产生式.

<u>Gra</u>mmar of XL

Example

Ī		
		by 1/

推导的不确定性

两个不确定性

- 选择句型中的非终结 符展升.
- 选择产生式.

Grammar of XL

```
T = \{ ID, PLUS, TIMES, \}
     LP, RP };
N = \{ \exp, term, fac \};
S = \exp;
P:
1/ exp -> exp PLUS term
2/ | term
3/ term -> term TIMES fac
4/ | fac
5/ fac -> ID
6/ | LP exp RP
```

	exp	
\Rightarrow	exp PLUS term	by 1/
\Rightarrow	term PLUS term	by 2/
\Rightarrow	<u>fac</u> PLUS term	by 4/
	ID PLUS <u>term</u>	by 5/
	ID PLUS <u>term</u> TIMES fac	by 3/
	ID PLUS <u>fac</u> TIMES fac	by 4/
\Rightarrow	ID PLUS ID TIMES <u>fac</u>	by 5/
\Rightarrow	ID PLUS ID TIMES ID	by 5/
		by 4/
		by 5/

exp

 \Rightarrow exp PLUS term

⇒ term PLUS term

by 1/

by 2/

两个不确定性

- 选择句型中的非终结符展升。
- 选择产生式.

Grammar of XL

\Rightarrow	<u>fac</u> PLUS term	by 4/
\Rightarrow	ID PLUS <u>term</u>	by 5/
\Rightarrow	ID PLUS <u>term</u> TIMES fac	by 3/
\Rightarrow	ID PLUS <u>fac</u> TIMES fac	by 4/
\Rightarrow	ID PLUS ID TIMES fac	by 5/
\Rightarrow	ID PLUS ID TIMES ID	by 5/
	exp	
\Rightarrow	exp PLUS <u>term</u>	by 1/
\Rightarrow	exp PLUS term TIMES fac	by 3/
\Rightarrow	exp PLUS term TIMES ID	by 5/
\Rightarrow	exp PLUS fac TIMES ID	by 4/
\Rightarrow	exp PLUS ID TIMES ID	by 5/
\Rightarrow	term PLUS ID TIMES ID	by 2/
\Rightarrow	fac PLUS ID TIMES ID	by 4/
\Rightarrow	ID PLUS ID TIMES ID	by 5/

最左维导与最右维导

定义

上下支充关支法的设计

最左维导与最右维导

定义

- 每次推导都是对句型中的最左边的非终结符展开,称 为最左推导, 记为: ⇒ (leftmost);

上下文无关文法的设计

最左维导与最右维导

定义

上下支充差支法

- 每次推导都是对句型中的最左边的非终结符展开,称 为最左维导, 记为: ⇒ (leftmost);
- 每次推导都是对句型中的最右边的非终结符展开,称 为最右维导, 记为: 촺 (rightmost).



最左维导					
	exp				
⇒ Im	exp PLUS term	by 1/			
⇒ Im	term PLUS term	by 2/			
⇒ Im	fac PLUS term	by 4/			
⇒ Im	ID PLUS <u>term</u>	by 5/			
\Rightarrow Im	ID PLUS <u>term</u> TIMES fac	by 3/			
\Rightarrow	ID PLUS <u>fac</u> TIMES fac	by 4/			
⇒ Im	ID PLUS ID TIMES fac	by 5/			
\Rightarrow Im	ID PLUS ID TIMES ID	by 5/			

最右维导 exp exp PLUS term by 1/ \Rightarrow exp PLUS term TIMES \underline{fac} by 3/ by 5/ $\Rightarrow \exp PLUS \underline{term} TIMES ID$ $\Rightarrow_{rm} exp$ PLUS \underline{fac} TIMES ID by 4/ by 5/ exp PLUS ID TIMES ID by 2/ term PLUS ID TIMES ID by 4/ fac PLUS ID TIMES ID ⇒ ID PLUS ID TIMES ID by 5/

最左与最右句型

定义

上下支充关支法

设 G 是 CFG.

- a ∈ (T∪N)* 是 G 的一个最左句型 (leftmost sentential form) iff S 拳 a.
- $a \in (T \cup N)^*$ 是 G 的一个最右句型 (rightmost sentential form) iff $S \stackrel{\Rightarrow}{\Rightarrow} a$.

最左与最右句型

定义

设 G 是 CFG.

- $a \in (T \cup N)^*$ 是 G 的一个最左句型 (leftmost sentential form) iff $S \stackrel{\Rightarrow}{\Rightarrow} a$.
- $\alpha \in (T \cup N)^*$ 是 G 的一个最右句型 (rightmost sentential form) iff $S \stackrel{\Rightarrow}{\Rightarrow} \alpha$.

最左与最右句型

定义

设 G 是 CFG.

- $a \in (T \cup N)^*$ 是 G 的一个最左句型 (leftmost sentential form) iff $S \stackrel{*}{\Rightarrow} a$.
- α ∈ (T∪N)* 是 G 的一个最右句型 (rightmost sentential form) iff S ⇒ α.

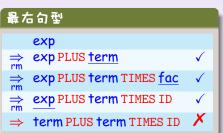


```
最右句型

exp

⇒ exp PLUS term
→ exp PLUS term TIMES fac ✓
rm
exp PLUS term TIMES ID ✓
rm
term PLUS term TIMES ID ✓
```





- ●最左维导是最左边的文法符号形成为终结符,最右维 具是最右边的文法符号形式为终结符
- 每条语句既是左句型又是右句型;
- 设 S ૐ αAβ, β ૐ β′, 则 αAβ′ 一定不是左句型.
- 设 S [⇒] αAβ, α [⇒] α', 则 α'Aβ 一定不是右句型。
- 语法分析的过程就是模拟该两种推导。即对该两种句型的识别

性质

- 最左推导是最左边的文法符号形成为终结符,最右推导是最右边的文法符号形成为终结符。
- 每条语句既是左句型又是右句型:
- 设 S ⇒ αAβ, β ⇒ β', 则 αAβ' 一定不是左句型。

语法树

- 设 $S \stackrel{*}{\Rightarrow} aA\beta, a \stackrel{*}{\Rightarrow} a', 则 a'A\beta 一定不是右句型$
- 语法分析的过程就是模拟该两种维导。即对该两种句型 的识别

- 最左推导是最左边的文法符号形成为终结符。 最右推 早是最右边的文法符号形成为终结符。
- 每条语句既是左句型又是右句型;

- 最左推导是最左边的文法符号形成为终结符。 最右推 早是最右边的文法符号形成为终结符.
- 每条语句既是左句型又是右句型;
- 设 S * αAβ, β * β', № αAβ' 定不是左句型.

- 最左推导是最左边的文法符号形成为终结符,最右推导是最右边的文法符号形成为终结符
- 每条语句既是左句型又是右句型;
- 设 S ^{*} αAβ,β [†] β',则αAβ' 定不是左句型.
- 设 S 🌦 αAβ,α 🛖 α′, 则 α′Aβ 一定不是右句型.
- 语法分析的过程就是模拟这两种推导,即对这两种句型的识别

- 最左推导是最左边的文法符号形成为终结符,最右推导是最右边的文法符号形成为终结符。
- 每条语句既是左句型又是右句型;
- 设 S ^{*} αAβ, β [†] β′, № αAβ′ 定不是左<mark>行</mark>型.
- 设 S * αAβ, α ÷ α', 则 α'Aβ 一定不是右句型.
- 语法分析的过程就是模拟这两种维导。即对这两种句型的识别。

上下支无关支法

- - 递归定义与产生式
 - 上下文无关文法
 - 上下文无关语言
- 语法树
 - 语法树
 - 语法树与椎导的关系
 - 二义性
- 上下文无关文法的设计
 - 设计举例
 - 正则表达式转换为上下支自由支法
 - 自动机转换为上下支自由支法
 - CFG 的表达能力
- 二义性的再讨论

最左维导

上下支无关支法

exp

 $\underset{\text{Im}}{\overset{1}{\Rightarrow}}$ exp PLUS term

 $\stackrel{?}{\Rightarrow}$ term PLUS term

 $\stackrel{3}{\Rightarrow}$ fac PLUS term

⁴→ID PLUS <u>term</u>

 \Rightarrow ID PLUS <u>term</u> TIMES fac

 $\stackrel{6}{\Longrightarrow}$ ID PLUS <u>fac</u> TIMES fac

 $\stackrel{7}{\Longrightarrow}$ ID PLUS ID TIMES \underline{fac}

⁸ ID PLUS ID TIMES ID



exp

最左维导

上下支无关支法

exp

语法树的建立



最左维导

上下支无关支法

exp

$\Rightarrow \frac{1}{m} \underbrace{\exp \text{PLUS term}}$

 $\stackrel{2}{\Rightarrow}$ term PLUS term

 $\Rightarrow \frac{3}{\text{Im}} \frac{\text{fac}}{\text{PLUS}} \text{ term}$

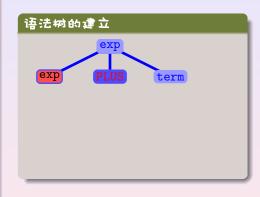
⇒ ID PLUS <u>term</u>

 $\stackrel{5}{\Rightarrow}$ ID PLUS <u>term</u> TIMES fac

 \Rightarrow ID PLUS <u>fac</u> TIMES fac

 $\stackrel{7}{\Rightarrow}$ ID PLUS ID TIMES <u>fac</u>

 $\stackrel{8}{\Rightarrow}$ ID PLUS ID TIMES ID



语法树 --- Example 1/2

最左维导

上下支无关支法

exp

 $\Rightarrow \frac{1}{m} = \exp PLUS term$

 $\Rightarrow \frac{2}{\text{lm}} \text{PLUS term}$

 $\stackrel{3}{\Rightarrow}$ fac PLUS term

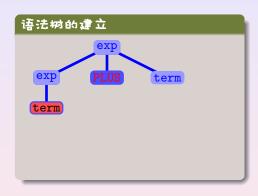
 $\stackrel{4}{\Rightarrow}$ ID PLUS <u>term</u>

 \Rightarrow ID PLUS <u>term</u> TIMES face

⇒ ID PLUS <u>fac</u> TIMES fac

→ ID PLUS ID TIMES fac

8 ⇒ID PLUS ID TIMES ID



语法树 --- Example 1/2

最左维导

上下文无关文法

exp

 $\Rightarrow \frac{1}{m} = \exp PLUS term$

 $\Rightarrow \frac{2}{\text{term}}$ PLUS term

 $\Rightarrow \frac{3}{\text{lm}} \underline{\text{fac}}$ PLUS term

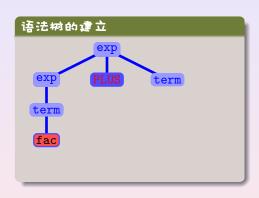
 $\stackrel{4}{\Rightarrow}$ ID PLUS <u>term</u>

5 ID PLUS term TIMES fac

 $\stackrel{6}{\Longrightarrow}$ ID PLUS <u>fac</u> TIMES fac

 $\stackrel{7}{\Longrightarrow}$ ID PLUS ID TIMES <u>fac</u>

8 ⇒ID PLUS ID TIMES ID



最左维导

上下支无关支法

exp

 $\Rightarrow \frac{1}{m} = \exp PLUS term$

≥ <u>term</u> PLUS term

 $\Rightarrow \frac{3}{\text{lm}} \underline{\text{fac}}$ PLUS term

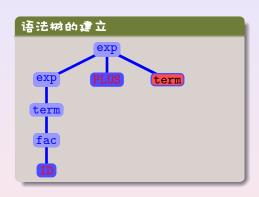
 $\underset{\text{lm}}{\overset{4}{\Rightarrow}}$ ID PLUS $\underline{\text{term}}$

 \Rightarrow ID PLUS <u>term</u> TIMES fac

 $\stackrel{6}{\Rightarrow}$ ID PLUS <u>fac</u> TIMES fac

 $\xrightarrow{7}$ ID PLUS ID TIMES <u>fac</u>

8 ⇒ ID PLUS ID TIMES ID



语法树 --- Example 1/2

最左维县

exp

 $\Rightarrow \frac{1}{m} \underbrace{\exp \text{PLUS term}}$

 $\stackrel{2}{\Rightarrow} \underline{\text{term}} \text{ PLUS term}$

 $\Rightarrow \frac{3}{\text{lm}} \underline{\text{fac}}$ PLUS term

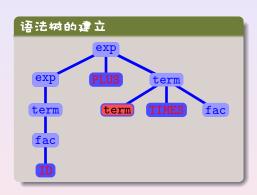
 $\stackrel{4}{\Rightarrow}$ ID PLUS <u>term</u>

 \Rightarrow ID PLUS <u>term</u> TIMES fac

 $\underset{\mathsf{Im}}{\overset{6}{\Rightarrow}}$ ID PLUS <u>fac</u> TIMES fac

 $\stackrel{7}{\Longrightarrow}$ ID PLUS ID TIMES <u>fac</u>

⁸ ID PLUS ID TIMES ID



语法树 --- Example 1/2

最左维导

exp

⇒exp PLUS term

≥ term PLUS term

 $\Rightarrow \frac{3}{\text{lm}} \underline{\text{fac}}$ PLUS term

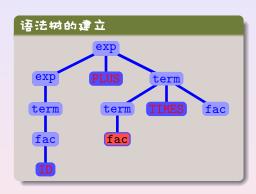
 $\stackrel{4}{\Rightarrow}$ ID PLUS <u>term</u>

 $\stackrel{5}{\Rightarrow} ID PLUS \underline{term} TIMES fac$

 $\Rightarrow \frac{6}{m}$ ID PLUS <u>fac</u> TIMES fac

→ ID PLUS ID TIMES fac

8 → ID PLUS ID TIMES ID



最左维导

exp

⇒exp PLUS term

≥ term PLUS term

 $\Rightarrow \frac{3}{\text{lm}} \underline{\text{fac}}$ PLUS term

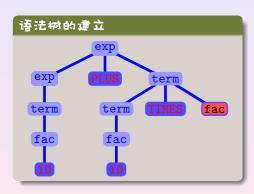
 $\stackrel{4}{\Rightarrow}$ ID PLUS <u>term</u>

 \Rightarrow ID PLUS <u>term</u> TIMES fac

 $\stackrel{6}{\Longrightarrow}$ ID PLUS <u>fac</u> TIMES fac

 $\overrightarrow{\Rightarrow}_{lm}$ ID PLUS ID TIMES <u>fac</u>

 $\overset{\$}{\Rightarrow}$ ID PLUS ID TIMES ID



最左维导

exp

 \Rightarrow exp PLUS term

⇒term PLUS term

 $\Rightarrow \frac{3}{\text{Im}} \frac{\text{fac}}{\text{PLUS term}}$

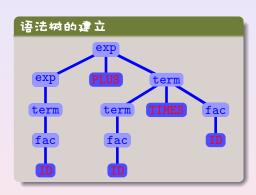
 $\stackrel{4}{\Rightarrow}$ ID PLUS term

 $\stackrel{5}{\Rightarrow}$ ID PLUS term TIMES fac

 $\stackrel{6}{\Rightarrow}$ ID PLUS fac TIMES fac

 $\stackrel{7}{\Rightarrow}$ ID PLUS ID TIMES fac

⇒ID PLUS ID TIMES ID



最右维导

 $\Rightarrow_{rm}^{1} exp PLUS \underline{term}$

 \Rightarrow exp PLUS term TIMES \underline{fac}

 $\Rightarrow_{rm}^{3} exp PLUS <u>term</u> TIMES ID$

 \Rightarrow exp PLUS <u>fac</u> TIMES ID

 \Rightarrow exp PLUS ID TIMES ID

 $\stackrel{6}{\Rightarrow} \underbrace{\text{term}}_{\text{rm}} \text{PLUS ID TIMES ID}$

 $\Rightarrow \underline{\mathsf{fac}}$ PLUS ID TIMES ID

 $\stackrel{8}{\Rightarrow}$ ID PLUS ID TIMES ID

语法树的建立

最右维导

上下支无关支法

exp

语法树的建立

exp

最右维导

exp

 $\Rightarrow_{\text{rm}}^{1} \exp \text{PLUS} \, \underline{\text{term}}$

语法树的建立 exp term

最右维导

上下文无关文法

exp

 $\underset{rm}{\overset{1}{\Rightarrow}} exp PLUS \underline{term}$

 \Rightarrow exp PLUS term TIMES <u>fac</u>

 \Rightarrow exp PLUS term TIMES ID

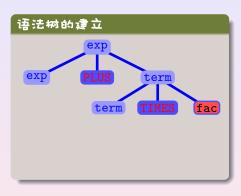
 \Rightarrow exp PLUS <u>fac</u> TIMES ID

5 exp PLUS ID TIMES ID

 $\stackrel{6}{\Rightarrow}$ term PLUS ID TIMES ID

 $\stackrel{7}{\Rightarrow}$ fac PLUS ID TIMES ID

 $\stackrel{8}{\Longrightarrow}$ ID PLUS ID TIMES ID



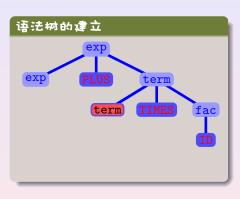
最右维导

exp

 \Rightarrow exp PLUS term

 \Rightarrow exp PLUS term TIMES \underline{fac}

 \Rightarrow exp PLUS <u>term</u> TIMES ID



最右堆县

exp

 $\Rightarrow_{rm}^{1} exp PLUS \underline{term}$

 \Rightarrow exp PLUS term TIMES <u>fac</u>

 $\Rightarrow_{rm}^{3} exp PLUS <u>term</u> TIMES ID$

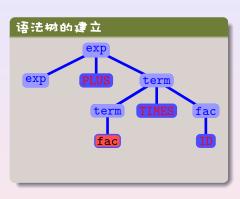
 $\underset{rm}{\overset{4}{\Rightarrow}}$ exp PLUS \underline{fac} TIMES ID

⇒exp PLUS ID TIMES ID

 $\stackrel{6}{\Rightarrow} \underbrace{\mathsf{term}}_{\mathsf{rm}} \mathsf{PLUS} \mathsf{ID} \mathsf{TIMES} \mathsf{ID}$

 \Rightarrow fac PLUS ID TIMES ID

 $\stackrel{8}{\Rightarrow}$ ID PLUS ID TIMES ID



最右维导

exp

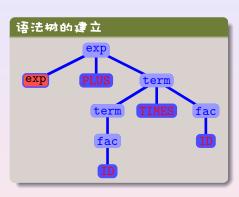
 \Rightarrow exp PLUS term

 $\Rightarrow_{rm}^{2} exp PLUS term TIMES <u>fac</u>$

 \Rightarrow exp PLUS <u>term</u> TIMES ID

 $\Rightarrow \exp PLUS \underline{fac} TIMES ID$

 $\Rightarrow \frac{5}{\text{rm}} \text{ PLUS ID TIMES ID}$



◆□ → ◆□ → ◆豆 → ◆豆 → □ □

最右维导

exp

 $\underset{rm}{\overset{1}{\Rightarrow}}$ exp PLUS $\underline{\text{term}}$

≥ exp PLUS term TIMES fac

 $\Rightarrow_{rm}^{3} exp PLUS <u>term</u> TIMES ID$

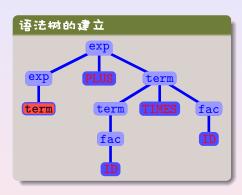
 \Rightarrow exp PLUS \underline{fac} TIMES ID

 $\Rightarrow \frac{5}{\text{rm}} \text{exp}$ PLUS ID TIMES ID

 $\stackrel{6}{\Longrightarrow}$ term PLUS ID TIMES ID

 $\Rightarrow \frac{fac}{rm}$ PLUS ID TIMES ID

*B ID PLUS ID TIMES ID



4 日 × 4 团 × 4 豆 × 4 豆 × 豆

最右维导

exp

 $\Rightarrow_{rm}^{1} exp PLUS \underline{term}$

≥ exp PLUS term TIMES fac

 $\Rightarrow_{rm}^{3} exp PLUS <u>term</u> TIMES ID$

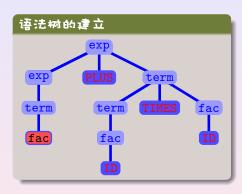
 $\underset{rm}{\overset{4}{\Rightarrow}}$ exp PLUS \underline{fac} TIMES ID

 $\Rightarrow \frac{5}{\text{cm}} \text{ PLUS ID TIMES ID}$

 $\stackrel{6}{\Longrightarrow} \underbrace{\text{term}}_{\text{rm}} \text{PLUS ID TIMES ID}$

 $\Rightarrow \frac{7}{\text{rm}} \text{fac}$ PLUS ID TIMES ID

 $\stackrel{8}{\Longrightarrow}$ ID PLUS ID TIMES ID



4 日 × 4 团 × 4 豆 × 4 豆 × 豆

最右维导

exp

 $\underset{rm}{\overset{1}{\Rightarrow}} exp PLUS \underline{term}$

≥ exp PLUS term TIMES fac

 $\Rightarrow_{rm}^{3} exp PLUS <u>term</u> TIMES ID$

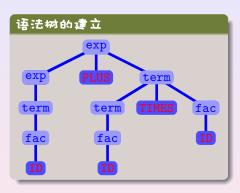
 $\Rightarrow \exp PLUS \underline{fac} TIMES ID$

 $\Rightarrow \frac{5}{\text{cm}} \text{ PLUS ID TIMES ID}$

 $\stackrel{6}{\Longrightarrow} \underbrace{\text{term}}_{\text{rm}} \text{PLUS ID TIMES ID}$

 $\Rightarrow \frac{7}{\text{rm}} \text{fac}$ PLUS ID TIMES ID

 $\stackrel{8}{\Longrightarrow}$ ID PLUS ID TIMES ID



递归定义



谦归定义

- 母 S 文法开始符号,则 S 是语法树 (语法分析树, Parse Tree).



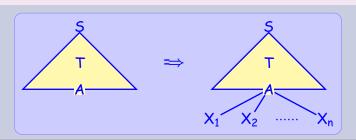
谦归定义

- 设 S 文法开始符号,则 S 是语法树 (语法分析树, Parse Tree).
- 设 T 是语法树、A∈N 是 T 的树叶、
 设 A → X₁X₂ ··· X_m ∈ P, 则在 A 下从左到右加上标号
 分别为 X₁, X₂, ··· , X_m 的 m 个儿子所得到树是语法树。



谦归定义

- 设 S 文法开始符号,则 S 是语法树 (语法分析树, Parse Tree).
- 设 T 是语法树、A∈N 是 T 的树叶、
 设 A → X₁X₂ ··· X_m ∈ P,则在 A 下从左到右加上标号
 分别为 X₁, X₂, ··· , X_m 的 m 个儿子所得到树是语法树。



语法树与推导的关系

- 语法树的内部节点一定是非终结符
- 语法树过滤掉了推导过程对非终结符选择策略.
- 一个维导过程等价子对语法树的语法树的调历过程,即:S ⇒ a 是一个维导 iff 日语法树下,其叶结点从左到右循序排列为 a.
- 最左椎导是前序端布 (树根 → 左ル子 → 右ル子). 最右椎导
 是 NRL: 树根 → 右ル子 → 左ル子.
- a 是最左句型 (S ♣ a) iff 存在一颗语法树,其前序遍历得 到的叶结点序列是 a.
- α 是最右句型 (S ♣ α) iff 存在一颗语法树,其 NRL 遍历境 到的叶结点序列是 α.
- 一颗语法树对应唯一的一个最左维导和唯一的一个最右维导。
- 语法分析也可以看成建立语法树的过程!

语法树与推导的关系

- 语法树的内部节点一定是非终结符.
- 语法树过滤掉了推导过程对非终结符选择策略,
- 一个维导过程等价于对语法树的语法树的遍历过程,即:
 S ⇒ α是一个维导 iff 日语法树 T,其叶结点从左到右循序 排列为 α.
- 最左維导是前序端牙 (树根 → 左ル子 → 右ル子). 最右維导
 是 NRL: 树根 → 右ル子 → 左ル子.
- α 是最左句型 (S ♣ α) iff 存在一颗语法树,其前序遍历得 到的叶结点序列是 α
- α 是最右句型 (S ♣ α) iff 存在一颗语法树,其 NRL 遍历得 到的叶结点序列是 α.
- 一颗语法树对应唯一的一个最左维导和唯一的一个最右维导。
- 语法分析也可以看成建立语法树的过程

语法树与维导的关系

- 语法树的内部节点一定是非终结符。
- 语法树过滤掉了推导过程对非终结符选择策略。

- 语法树的内部节点一定是非终结符。
- 语法树过滤掉了推导过程对非终结符选择策略.
- 一个椎导过程等价于对语法树的语法树的遍历过程,即:
 S ⇒ α是一个椎导 iff ∃语法树 T,其叶结点从左到右循序排列为 α.
- 最左维导是前序遍历 (树根 → 左ル子 → 右ル子). 最右维导
 是 NRL: 树根 → 右ル子 → 左ル子.
- α 是最左句型 (S 拳 α) iff 存在一颗语法树,其前序遍历得到的叶结点序列是 α
- a 是最右句型 (S ♣ a) iff 存在一颗语法树,其 NRL 遍历得 到的叶结点序列是 a.
- 一颗语法树对应唯一的一个最左维导和唯一的一个最右维导
- 语法分析也可以看成建立语法树的过程

语法树与椎导的关系

- 语法树的内部节点一定是非终结符.
- 语法树过滤掉了推导过程对非终结符选择策略。
- 一个推导过程等价于对语法树的语法树的遍历过程,即:
 S ⇒ α是一个推导 iff ∃语法树 T,其叶结点从左到右循序排列为 α.
- 最左椎导是前序端牙 (树根 → 左ル子 → 右ル子). 最右椎导
 足 NRL: 树根 → 右ル子 → 左ル子.
- a 是最左句型 (S 拳 a) iff 存在一颗语法树,其前序遍历得到的叶结点序列是 a
- α 是最右句型 (S ≱ α) iff 存在一颗语法树,其 NRL 遍历得 到的叶结点序列是 α.
- 一颗语法树对应唯一的一个最左椎界和唯一的一个最右椎
- 语法分析也而以看成建立语法树的过程

- 34/65 -

- 语法树的内部节点一定是非终结符.
- 语法树过滤掉了推导过程对非终结符选择策略.
- 一个推导过程等价于对语法树的语法树的遍历过程,即:
 S → α是一个推导 iff ∃语法树 T,其叶结点从左到右循序排列为 α.
- 最左椎导是前序端牙 (树根 → 左ル子 → 右ル子). 最右椎导
 足 NRL: 树根 → 右ル子 → 左ル子.
- α 是最左句型 (S 촺 α) iff 存在一颗语法树,其前序遍历得 到的叶结点序列是 α.
- α 是最右句型 (S ≱ α) iff 存在一颗语法树,其 NRL 遍历得 到的叶结点序列是 α.
- 一颗语法树对应唯一的一个最左维导和唯一的一个最右维导。
 - 语法分析也可以看成建立语法树的过程

- 语法树的内部节点一定是非终结符。
- 语法树过滤掉了推导过程对非终结符选择策略。
- 一个推导过程等价子对语法树的语法树的偏历过程。即: S ⇒ α是一个推导 iff ∃语法树 T, 其叶结点从左到右循序 排列为口
- 最左推导是前序遍历 (树根 → 左ル子 → 右ル子). 最右推导 是 NRI: 树根 → 右ルユ → 左ルユ
- α 是最左句型 (S ⇒ α) iff 存在一颗语法树,其前序遍历得 到的叶结点序列是 (1
- a 是最右句型 (S ⇒ a) iff 存在一颗语法树,其 NRL 遍历得 到的叶结点序列是 (1

语法树与椎界的关系

- 语法树的内部节点一定是非终结符.
- 语法树过滤掉了推导过程对非终结符选择策略。
- 一个推导过程等价于对语法树的语法树的遍历过程,即:
 S → α是一个推导 iff ∃语法树 T,其叶结点从左到右循序排列为 α.
- 最左椎导是前序端牙 (树根 → 左ル子 → 右ル子). 最右椎导
 足 NRL: 树根 → 右ル子 → 左ル子.
- a 是最左句型 (S 촺 a) iff 存在一颗语法树,其前序遍历得 到的叶结点序列是 a.
- α 是最右句型 (S 촺 α) iff 存在一颗语法树,其 NRL 遍历得 到的叶结点序列是 α.
- 一颗语法树对应唯一的一个最左推导和唯一的一个最右推导。
- 语法分析也可以看成建立语法树的过程

- 语法树的内部节点一定是非终结符。
- 语法树过滤掉了推导过程对非终结符选择策略。
- 一个推导过程等价于对语法树的语法树的遍历过程,即:
 S → α是一个推导 iff ∃语法树 T,其叶结点从左到右循序排列为 α.
- 最左維导是前序端牙 (树根 → 左ル子 → 右ル子). 最右維导
 足 NRL: 树根 → 右ル子 → 左ル子.
- α 是最左句型 (S 촺 α) iff 存在一颗语法树,其前序遍历得 到的叶结点序列是 α.
- α 是最右句型 (S 촺 α) iff 存在一颗语法树,其 NRL 遍历得 到的叶结点序列是 α.
- 一颗语法树对应唯一的一个最左推导和唯一的一个最右推导。
- 语法分析也可以看成建立语法树的过程。

Example

两个不同的最左维导

Example

```
    西介不同的最左维导

    exp ⇒ exp PLUS exp ⇒ exp TIMES exp ⇒ exp PLUS exp TIMES exp ⇒ im exp PLUS exp TIMES exp ⇒ in D PLUS exp TIMES exp ⇒ in D PLUS exp TIMES exp ⇒ in D PLUS in TIMES in D plus in TIMES in D in D PLUS in D In D PLUS in TIMES in D in D PLUS in D In
```

exp PLUS TIMES exp TIMES exp PLUS exp ID ID (a) 乘法运算先结合 (b) 加法运算先结合

上下支无关支法

二义性 (Ambiguity)

二义性的定义

称 CFGG 是二义性文法 iff 存在一语句有两颗不同的语法树 iff 存在一语句有两个不同的最左 (右) 推导.

: 丰野

称 CFG G 是二义性文法 iff 存在一语句有两颗不同的语法树 iff 存在一语句有两个不同的最左 (右) 推导.

注释

上下支无关支法

称 CFG G 是二义性文法 iff 存在一语句有两颗不同的语法树 iff 存在一语句有两个不同的最左 (右) 推导.

注释

- □ 二义性导致了语法分析的不确定性,同样的语句有不同的理解,在文法设计时要避免:
 - Ex: C 语言数组元素的引用 ``A[]'' 和函数调用 ``A
 - Ex: C 语言的 ``A (B)'': 函数调用或者变量说明?
- 二义性文法一定没有确定的语法分析 (无回溯).
- 可以通过特定的分析规则在语法分析过程中解消二义性,

称 CFGG 是二义性文法 iff 存在一语句有两颗不同的语法树 iff 存在一语句有两个不同的最左 (右) 推导.

注释

- 二义性导致了语法分析的不确定性,同样的语句有不同的理解,在文法设计时要避免:
 - Ex: C 语言数组元素的引用 ``A[]'' 和函数调用 ``A()''
- EX: C 语言的 A(B): 函数调用或有变重说明!
- 二义性文法一定没有确定的语法分析 (无回溯).
- 可以通过特定的分析规则在语法分析过程中解消二义性

称 CFG G 是二义性文法 iff 存在一语句有两颗不同的语法树 iff 存在一语句有两个不同的最左 (右) 推导.

注释

- 二义性导致了语法分析的不确定性,同样的语句有不同的理解,在文法设计时要避免:
 - Ex: C 语言数组元素的引用 ``A[]'' 和函数调用 ``A()''
 - Ex: C 语言的 ``A (B)'': 函数调用或者变量说明?
- 二义性文法一定没有确定的语法分析(无回溯)。
- 可以通过特定的分析规则在语法分析过程中解消二义性

● 其世领域也有二义性,如面向对象的 diamond problem.

称 CFGG 是二义性文法 iff 存在一语句有两颗不同的语法树 iff 存在一语句有两个不同的最左 (右) 推导.

- 二义性导致了语法分析的不确定性,同样的语句有不同的理解,在文法设计时要避免:
 - Ex: C 语言数组元素的引用 ``A[]'' 和函数调用 ``A()''
 - Ex: C 语言的 ``A (B)'': 函数调用或者变量说明?
- 二义性文法一定没有确定的语法分析 (无回溯)。
- 可以通过特定的分析规则在语法分析过程中解消二义性

二义性 (Ambiguity)

二义性的定义

称 CFGG 是二义性文法 iff 存在一语句有两颗不同的语法树 iff 存在一语句有两个不同的最左 (右) 推导.

- 二义性导致了语法分析的不确定性,同样的语句有不同的理解,在文法设计时要避免:
 - Ex: C 语言数组元素的引用 ``A[]'' 和函数调用 ``A()''
 - Ex: C 语言的 ``A (B)'': 函数调用或者变量说明?
- 二义性文法一定没有确定的语法分析 (无回溯)。
- 可以通过特定的分析规则在语法分析过程中解消二义性.
 - 二义性在 XL' 上表现为运算的优先级别和结合次序。
- 其世领域也有二义性,如面向对象的 diamond problem.

称 CFGG 是二义性文法 iff 存在一语句有两颗不同的语法树 iff 存在一语句有两个不同的最左 (右) 推导.

- 二义性导致了语法分析的不确定性,同样的语句有不同的理解,在文法设计时要避免:
 - Ex: C 语言数组元素的引用 ``A[]'' 和函数调用 ``A(
 - Ex: C 语言的 ``A(B)'': 函数调用或者变量说明?
- 二义性文法一定没有确定的语法分析 (无回溯).
- 可以通过特定的分析规则在语法分析过程中解消二义性.
 - Ex: XL 无二义性文法,而 XL' 是二义文法.
 - 二义性在 XL'上表现为运算的优先级别和结合次序。 ● 二义性之法要比工二义性之法要然法
- 其世领域也有二义性,如面向对象的 diamond problem.

称 CFG G 是二义性文法 iff 存在一语句有两颗不同的语法树 iff 存在一语句有两个不同的最左 (右) 推导.

- 二义性导致了语法分析的不确定性,同样的语句有不同的理解。在文法设计时要避免:
 - Ex: C 语言数组元素的引用 ``A[]'' 和函数调用 ``A()''
 - Ex: C 语言的 ``A (B)'': 函数调用或者变量说明?
- 二义性文法一定没有确定的语法分析 (无回溯)。
- 可以通过特定的分析规则在语法分析过程中解消二义性.
 - Ex: XL 无二义性文法,而 XL' 是二义文法.
 - 二义性在 XL' 上表现为运算的优先级别和结合次序.
 - 二×性交法要吐尤二×性交法要间治.

称 CFG G 是二义性文法 iff 存在一语句有两颗不同的语法树 iff 存在一语句有两个不同的最左 (右) 维导.

- 二义性导致了语法分析的不确定性,同样的语句有不同的理解。在文法设计时要避免:
 - Ex: C 语言数组元素的引用 ``A[]'' 和函数调用 ``A(
 - Ex: C 语言的 ``A (B)'': 函数调用或者变量说明?
- 二义性文法一定没有确定的语法分析 (无回溯)。
- 可以通过特定的分析规则在语法分析过程中解消二义性.
 - Ex: XL 无二义性文法,而 XL' 是二义文法.
 - 二义性在 XL' 上表现为运算的优先级别和结合次序.
 - 二义性文法要比无二义性文法要简洁。
 - 其世领域也有二义性,如面向对象的 diamond problem.

称 CFGG 是二义性文法 iff 存在一语句有两颗不同的语法树 iff 存在一语句有两个不同的最左 (右) 推导.

- □ 二义性导致了语法分析的不确定性,同样的语句有不同的理解,在文法设计时要避免:
 - Ex: C 语言数组元素的引用 ``A[]'' 和函数调用 ``A()''.
 - Ex: C 语言的 ``A (B)'': 函数调用或者变量说明?
- 二义性文法一定没有确定的语法分析(无回溯)。
- 可以通过特定的分析规则在语法分析过程中解消二义性.
 - Ex: XL 五二义性文法。而 XL¹ 是二义文法。
 - 二义性在 XL' 上表现为运算的优先级别和结合次序.
 - 二义性文法要比无二义性文法要简洁.
- 其世领域也有二义性,如面向对象的 diamond problem.

上下支充关支法的设计

- - 递归定义与产生式
 - 上下文无关文法
 - 上下文无关语言
- - 语法树
 - 语法树与维导的关系
 - ●二义性
- 上下文无关文法的设计
 - 设计举例
 - 正则表达式转换为上下文自由文法
 - 自动机转换为上下文自由文法
 - CFG 的表达能力
- 二义性的再讨论

上下支无关支法

L(G) = {以字母 a 和 b 字符串,该串中 a 和 b 的出现次数相同} 分析 L(G) 语句的增长规律:

上下支无关支法

L(G)={以字母 a 和 b 字符串,该串中 a 和 b 的出现次数相同} 分析 L(G) 语句的增长规律:

上下支充关支法的设计

•000000000000000000

- $\mathbf{0}$ $\epsilon \in \mathsf{L}$.

上下支无关支法

L(G)={以字母 a 和 b 字符串,该串中 a 和 b 的出现次数相同} 分析 L(G) 语句的增长规律:

- $\mathbf{0}$ $\epsilon \in \mathsf{L}$.
- 2 if $w \in L$, then awb, abw, baw, bwa, wab, wba $\in L$.



```
L(G) = {以字母 a 和 b 字符串,该串中 a 和 b 的出现次数相同}分析 L(G) 语句的增长规律:
```

- $\mathbf{0}$ $\epsilon \in \hat{\mathsf{L}}$.
- 2 if $w \in L$, then awb, abw, baw, bwa, wab, wba $\in L$.
- 3 支法 6:

 $S \rightarrow aSb \mid abS \mid baS \mid bSa \mid Sab \mid Sba \mid \epsilon$

上下支充关支法

验证

支法 6:

 $S \rightarrow aSb \mid abS \mid baS \mid bSa \mid Sab \mid Sba \mid \epsilon$ 则 aⁿb²ⁿaⁿ ∉ L(G)(n ≥ 2), 但是该串中 a 和 b 均出现 2n 次. 句型分析:

验证

支法 6:

 $S \rightarrow aSb \mid abS \mid baS \mid bSa \mid Sab \mid Sba \mid \epsilon$ 则 aⁿb²ⁿaⁿ ∉ L(G) (n ≥ 2), 但是该串中 a 和 b 均出现 2n 次. 句型分析:

- ① 设第一步推导用 S → abS.

验证

支法 6:

 $S \rightarrow aSb \mid abS \mid baS \mid bSa \mid Sab \mid Sba \mid \epsilon$ 则 aⁿb²ⁿaⁿ ∉ L(G)(n ≥ 2), 但是该串中 a 和 b 均出现 2n 次. 句型分析:

- ① 设第一步推导用 S → abS.
- ② 则 bS ⇒ aⁿ⁻¹b²ⁿaⁿ. 不可能.

上下文无关文法的设计

上下支无关支法

验证

支法 6:

 $S \rightarrow aSb \mid abS \mid baS \mid bSa \mid Sab \mid Sba \mid \epsilon$ 则 aⁿb²ⁿaⁿ ∉ L(G)(n ≥ 2), 但是该串中 a 和 b 均出现 2n 次. 白型分析:

- ① 设第一步推导用 S → abS.
- ② 则 bS ⇒ aⁿ⁻¹b²ⁿaⁿ. 不可能.
- ③ 同理对其世产生式可以得出相同的结论.

设 $x_1x_2...x_n$ ∈ {a, b}*, 定义函数 $f(i) = |x_1x_2...x_i|_a - |x_1x_2...x_i|_b$, 即子串 X1X2...Xi 中 a 出现的次数减去 b 的次数,则f 按步长 1 递 增或递减

ax₂x₃····x_{2n-1}a ∈ L 具有如下性质:

 X_2 X_3 ... X_i ... X_{2n-1}



上下支无关支法

设 $x_1x_2...x_n$ ∈ {a, b}*, 定义函数 $f(i) = |x_1x_2...x_i|_a - |x_1x_2...x_i|_b$, 即子串 X1X2...Xi 中 a 出现的次数减去 b 的次数,则f 按步长 1 递 增或递减

ax₂x₃···x_{2n-1}a ∈ L 具有如下性质:

4 D > 4 D > 4 D > 4 D > -

设 $x_1x_2...x_n$ ∈ {a, b}*, 定义函数 $f(i) = |x_1x_2...x_i|_a - |x_1x_2...x_i|_b$, 即子串 X1X2...Xi 中 a 出现的次数减去 b 的次数,则f 按步长 1 递 增或递减.

ax₂x₃···x_{2n-1}a ∈ L 具有如下性质:



上下支无关支法

设 $x_1x_2...x_n$ ∈ {a, b}*, 定义函数 $f(i) = |x_1x_2...x_i|_a - |x_1x_2...x_i|_b$, 即子串 X1X2...Xi 中 a 出现的次数减去 b 的次数,则f 按步长 1 递 增或递减.

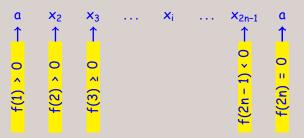
ax₂x₃···x_{2n-1}a ∈ L 具有如下性质:



4 日 × 4 周 × 4 国 × 4 国 ×

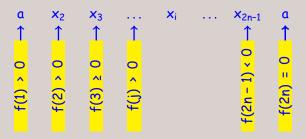
设 $x_1x_2...x_n$ ∈ {a, b}*, 定义函数 $f(i) = |x_1x_2...x_i|_a - |x_1x_2...x_i|_b$, 即子串 X1X2...Xi 中 a 出现的次数减去 b 的次数,则f 按步长 1 递 增或递减.

ax₂x₃···x_{2n-1}a ∈ L 具有如下性质:



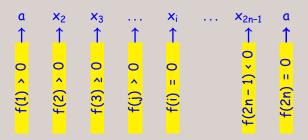
设 $x_1x_2...x_n$ ∈ {a, b}*, 定义函数 $f(i) = |x_1x_2...x_i|_a - |x_1x_2...x_i|_b$, 即子串 X1X2...Xi 中 a 出现的次数减去 b 的次数, 则 f 按步长 1 递 增或递减.

ax₂x₃···x_{2n-1}a ∈ L 具有如下性质:



设 $x_1x_2...x_n$ ∈ {a, b}*, 定义函数 $f(i) = |x_1x_2...x_i|_a - |x_1x_2...x_i|_b$, 即子串 X1X2...Xi 中 a 出现的次数减去 b 的次数,则f 按步长 1 递 增或递减.

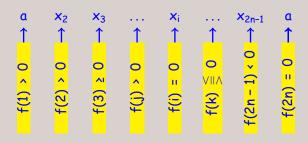
ax₂x₃···x_{2n-1}a ∈ L 具有如下性质:



4 D > 4 D > 4 D > 4 D > -

设 $x_1x_2...x_n$ ∈ {a, b}*, 定义函数 $f(i) = |x_1x_2...x_i|_a - |x_1x_2...x_i|_b$, 即子串 X1X2...Xi 中 a 出现的次数减去 b 的次数, 则 f 按步长 1 递 增或递减.

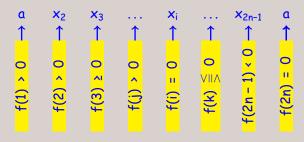
ax₂x₃···x_{2n-1}a ∈ L 具有如下性质:



4 D > 4 D > 4 D > 4 D > -

设 $x_1x_2...x_n \in \{a,b\}^*$, 定义函数 $f(i) = |x_1x_2...x_i|_a - |x_1x_2...x_i|_b$, 即了串 $x_1x_2...x_i$ 中 a 出现的次数减去 b 的次数, 则 f 按步长 1 递增或递减.

ax₂x₃···x_{2n-1}a ∈ L 具有如下性质:



 \boxtimes Lt, $\exists s_1, s_2 \in L$, and, $ax_1x_2 \cdots x_{2n}a = s_1s_2$.

4□ > 4□ > 4 ≡ > 4 ≡ > 3 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0 € < 0

新的文法 G

 $S \rightarrow aSb \mid bSa \mid SS \mid \epsilon$

新的文法 G

 $S \rightarrow \alpha Sb \mid bS\alpha \mid SS \mid \epsilon$

注释

产生式的独立性: 原文法 G 的产生式 S → abS 是新文法 G 的定理:

$$5 \Rightarrow 55 \Rightarrow a5b5 \Rightarrow ab5$$

○ 文法是二义文法:

 $S \Rightarrow a\underline{S}b \Rightarrow ab\underline{S}ab \Rightarrow abab$

 $S \Rightarrow \underline{S}S \Rightarrow \underline{a}\underline{S}bS \Rightarrow \underline{a}\underline{b}\underline{S} \Rightarrow \underline{a}\underline{b}\underline{S}b \Rightarrow \underline{a}\underline{b}\underline{b}$

● 等价支法:

 $S \rightarrow aSbS \mid bSaS \mid \epsilon$

上下文无关文法

新的文法 G

 $S \rightarrow aSb \mid bSa \mid SS \mid \epsilon$

注释

产生式的独立性: 原文法 6 的产生式 5 → abS 是新文 法6的定理:

$$S \Rightarrow SS \Rightarrow aSbS \Rightarrow abS$$

新的文法 G

$$S \rightarrow aSb \mid bSa \mid SS \mid \epsilon$$

注释

产生式的独立性: 原文法 6 的产生式 5 → abS 是新文 法6的定理:

$$S \Rightarrow \underline{S}S \Rightarrow \underline{a}\underline{S}bS \Rightarrow \underline{a}bS$$

文法是二义文法:

$$\begin{array}{lll} S & \underset{lm}{\Rightarrow} & a\underline{S}b \underset{lm}{\Rightarrow} ab\underline{S}ab \underset{lm}{\Rightarrow} abab \\ \\ S & \underset{lm}{\Rightarrow} & \underline{S}S \underset{lm}{\Rightarrow} a\underline{S}bS \underset{lm}{\Rightarrow} ab\underline{S} \underset{lm}{\Rightarrow} aba\underline{S}b \underset{lm}{\Rightarrow} abab \end{array}$$

$$S \rightarrow aSbS \mid bSaS \mid \epsilon$$

新的文法 G

$$S \rightarrow aSb \mid bSa \mid SS \mid \epsilon$$

注释

产生式的独立性: 原文法 6 的产生式 5 → abS 是新文 法6的定理:

$$S \Rightarrow \underline{S}S \Rightarrow a\underline{S}bS \Rightarrow abS$$

● 文法是二义文法:

$$\begin{array}{lll} S & \underset{lm}{\Rightarrow} & a\underline{S}b \underset{lm}{\Rightarrow} ab\underline{S}ab \underset{lm}{\Rightarrow} abab \\ \\ S & \underset{lm}{\Rightarrow} & \underline{S}S \underset{lm}{\Rightarrow} a\underline{S}bS \underset{lm}{\Rightarrow} ab\underline{S} \underset{lm}{\Rightarrow} aba\underline{S}b \underset{lm}{\Rightarrow} abab \end{array}$$

等价支法:

$$S \rightarrow aSbS \mid bSaS \mid \epsilon$$

 $B \rightarrow aBB \mid bS$

解法 3: The unambiguous grammar

```
设
            S={a和b出现的次数相同}
           A={a的次数比b的次数多1,
                且A的任一真前缀a不ttb含}
            B={b的次数比 a的次数多 1,
                且B的任一真前缀b不比a多}
因此
                    S = aBS \cup bAS \cup \{\epsilon\}
                    A = \{a\} \cup bAA
                    B = aBB \cup \{b\}
故文法 G"
                     S \rightarrow aBS \mid bAS \mid \varepsilon
                     A \rightarrow a \mid bAA
                     B \rightarrow aBB \mid b
```

$$L(G) = \{a^nb^n \mid n \in \mathbb{N}\}$$

分析 L(G) 中的串如何锑归增长:

$$S \rightarrow aSb \mid \epsilon$$

$$L(G) = \{a^nb^n \mid n \in \mathbb{N}\}$$

分析 L(G) 中的串如何递归增长:

- $\mathbf{0}$ $\epsilon \in \mathsf{L}$.

$$S \rightarrow aSb \mid \epsilon$$

$$L(G) = \{a^n b^n \mid n \in \mathbf{N}\}$$

分析 L(G) 中的串如何递归增长:

- $\mathbf{0}$ $\epsilon \in \mathsf{L}$.
- 2 if $w \in L$, then $awb \in L$.

$$S \rightarrow aSb \mid \epsilon$$

上下支充关支法的设计

Example 2

上下支无关支法

$$L(G) = \{a^nb^n \mid n \in \mathbb{N}\}$$

分析 L(G) 中的串如何锑归增长:

- \bullet $\epsilon \in L$.
- 2 if $w \in L$, then $awb \in L$.
- 3 支法 6:

$$S \rightarrow aSb \mid \epsilon$$





上下支无关支法

$$L(G) = \{a^nb^n \mid n \in \mathbb{N}\}$$

分析 L(G) 中的串如何锑归增长:

- $\mathbf{0}$ $\epsilon \in \mathsf{L}$.
- 2 if $w \in L$, then $awb \in L$.
- 3 支法 6:

$$S \rightarrow aSb \mid \epsilon$$

4 元二义性.

S = {a^mbⁿ | m, n ∈ N ∧ m ≠ n} 对集合 S 分拆,设 A = {a^mbⁿ | m > n}, B = {a^mbⁿ | m < n}, 则 S = A ∪ B

- ② 支法 A → aAb | aA | a (即 a 在 a 和 b 平衡之后还有 8 余) 生成集合 A.
- ② 支法 B → aBb | Bb | b (即 b 在 a 和 b 平衡文后还有多余) 生成集合 B.
- ③ 支法 G:

$$S \rightarrow A \mid B$$

 $A \rightarrow aAb \mid aA \mid a$
 $B \rightarrow aBb \mid Bb \mid b$

4 二义性?

上下支无关支法

S = {a^mbⁿ | m, n ∈ N ∧ m ≠ n} 对集合 S 分拆,设 A = {a^mbⁿ | m > n}, B = {a^mbⁿ | m < n}, 则 S = A ∪ B

- → aAb | aA | a (即 a 在 a 和 b 平衡之后还有多余) 生成集合 A.
- ② 文法 B → aBb | Bb | b (即 b 在 a 和 b 平衡之后还有多余) 生成集合 B.
- ③ 支法 G:

$$S \rightarrow A \mid B$$

 $A \rightarrow aAb \mid aA \mid a$
 $B \rightarrow aBb \mid Bb \mid b$

▲ 二义性?

S = {a^mbⁿ | m, n ∈ N ∧ m ≠ n} 对集合 S 分拆,设 A = {a^mbⁿ | m > n}, B = {a^mbⁿ | m < n}, 则 S = A ∪ B

- ② 文法 B → aBb | Bb | b (即 b 在 a 和 b 平衡之后还有多余) 生成集合 B.
- ③ 文法 G:

 $S \rightarrow A \mid B$ $A \rightarrow aAb \mid aA \mid a$ $B \rightarrow aBb \mid Bb \mid b$

④ 二义性?

- 文法 A → aAb | aA | a (即 a 在 a 和 b 平衡之后还有多余) 生成集合 A.
- ② 文法 B → aBb | Bb | b (即 b 在 a 和 b 平衡之后还有多余) 生成集合 B.
- ③ 文法 6:

$$S \rightarrow A \mid B$$

 $A \rightarrow aAb \mid aA \mid a$
 $B \rightarrow aBb \mid Bb \mid b$

4 二义性?

- ⇒ 法 A → aAb | aA | a (即 a 在 a 和 b 平衡之后还有多余) 生成集合 A.
- ② 文法 B → aBb | Bb | b (即 b 在 a 和 b 平衡之后还有多余) 生成集合 B.
- ③ 文法 6:

$$S \rightarrow A \mid B$$

 $A \rightarrow aAb \mid aA \mid a$
 $B \rightarrow aBb \mid Bb \mid b$

◎ 二义性?

$S = \{a^m b^n \mid m, n \in \mathbb{N} \land 2m \ge n\}$

- ① 文法 $A \rightarrow aAbb \mid \epsilon$ 住成集合 $\{a^nb^{2n}\}$, 且 $A \subseteq S$.
- ③ 集合 S 的串在两个 b 平衡一个 a (b 的次数是 a 的两倍) 之后,还可能有多余的 a.
- ④ 支法 6:

$$5 \rightarrow a5 \mid A$$

 $A \rightarrow aAbb \mid ab$

- $A \rightarrow dADD \mid dD \mid$
- ⑤ 删除非终结符 A 后
 - $S \rightarrow aS \mid aSbb \mid ab \mid \epsilon$

6 二 义性?

- $S = \{a^m b^n \mid m, n \in \mathbb{N} \land 2m \ge n\}$
- ① 文法 $A \rightarrow aAbb \mid \epsilon$ 住成集合 $\{a^nb^{2n}\}$, 且 $A \subseteq S$.
- ③ 集合 S 的串在两个 b 平衡一个 a (b 的次数是 a 的两倍) 之后, 还可能有多余的 a.
 - ④ 支法 6:

$$5 \rightarrow a5 \mid A$$

 $A \rightarrow aAbb \mid ab \mid \varepsilon$

- ③ 删除非终结符 A 后
 - $S \rightarrow aS \mid aSbb \mid ab \mid \epsilon$

6 二 义性?

- $S = \{a^m b^n \mid m, n \in \mathbb{N} \land 2m \ge n\}$
- ① 文法 A → αAbb | ε 生成集合 {aⁿb²ⁿ}, 且 A ⊆ S.
- ② 但是 b 在 S 中可为奇数, 在 S 中最少 b 的出现为语 句 ab. 因此加产生式 $A \rightarrow ab$, 即 $L(A) = \{a^m b^n \mid |2m-n| \le 1\}$.
- ③ 集合 S 的串在两个 b 平衡一个 a (b 的次数是 a 的两倍) 之后, 还可能有多余的 a.
- ④ 支法 6:

$$S \rightarrow aS \mid A$$

 $A \rightarrow aAbb \mid ab \mid \epsilon$

- 5 删除非终结符 A 后
 - $S \rightarrow aS \mid aSbb \mid ab \mid \epsilon$

(6) 二义性?

- $S = \{a^m b^n \mid m, n \in \mathbb{N} \land 2m \ge n\}$
- ② 但是 b 在 S 中可为奇数, 在 S 中最少 b 的出现为语 句 ab. 因此加产生式 $A \rightarrow ab$, 即 $L(A) = \{a^m b^n \mid |2m n| \le 1\}$.
- ③ 集合 S 的串在两个 b 平衡一个 a (b 的次数是 a 的两倍) 之后, 还可能有多余的 a.
 - ④ 束 法 G:

 $S \rightarrow aS \mid A$ $A \rightarrow aAbb \mid ab \mid \epsilon$

- ⑤ 删除非终结符 A 后
 - $S \rightarrow aS \mid aSbb \mid ab \mid \epsilon$

⑥ 二义性?

$$S = \{a^m b^n \mid m, n \in \mathbb{N} \land 2m \ge n\}$$

- ② 但是 b 在 S 中可为奇数, 在 S 中最少 b 的出现为语 句 ab. 因此加产生式 $A \rightarrow ab$, 即 $L(A) = \{a^m b^n \mid |2m n| \le 1\}$.
- ③ 集合 S 的串在两个 b 平衡一个 a (b 的次数是 a 的两倍) 之后, 还可能有多余的 a.
- ◎ 文法 6:

$$S \rightarrow aS \mid A$$

 $A \rightarrow aAbb \mid ab \mid \epsilon$

⑤ 删除非终结符 A 后

 $S \rightarrow aS \mid aSbb \mid ab \mid \epsilon$

⑥ 二义性?

$$S = \{a^m b^n \mid m, n \in \mathbb{N} \land 2m \ge n\}$$

- ② 但是 b 在 S 中可为奇数, 在 S 中最少 b 的出现为语 句 ab. 因此加产生式 $A \rightarrow ab$, 即 $L(A) = \{a^m b^n \mid |2m n| \le 1\}$.
- ③ 集合 S 的串在两个 b 平衡一个 a (b 的次数是 a 的两倍) 之后, 还可能有多余的 a.
- ◎ 文法 6:

$$S \rightarrow aS \mid A$$

 $A \rightarrow aAbb \mid ab \mid \epsilon$

5 删除非终结符 A 后

$$S \rightarrow aS \mid aSbb \mid ab \mid \epsilon$$

6 二义性?

$$S = \{a^m b^n \mid m, n \in \mathbb{N} \land 2m \ge n\}$$

- ③ 集合 S 的串在两个 b 平衡一个 a (b 的次数是 a 的两倍) 之后, 还可能有多余的 a.
- 4 → 法 6:

$$S \rightarrow aS \mid A$$

 $A \rightarrow aAbb \mid ab \mid \epsilon$

● 删除非终结符 A 后

$$S \rightarrow aS \mid aSbb \mid ab \mid \epsilon$$

⑥ 二义性?

上下支无关支法

L(G) = {由〈和〉组成的嵌套括号对}.

上下支无关支法

L(G) = {由〈和〉组成的嵌套括号对}.

上下文无关文法的设计

- $\mathbf{0}$ $\epsilon \in L$:

上下支无关支法

L(G) = {由〈和〉组成的嵌套括号对}.

- $\mathbf{0}$ $\epsilon \in L$;
- 2 if $w \in L$, then $\langle w \rangle$, $ww \in L$.

上下支无关支法

- $\mathbf{0}$ $\epsilon \in L$;
- 2 if $w \in L$, then $\langle w \rangle$, $ww \in L$.
- 3 二义性支法 6₁:

$$S \rightarrow \langle S \rangle | SS | \epsilon$$

$$S \rightarrow \langle S \rangle S | \varepsilon$$

上下支无关支法

分析 L(G) 的语句如何递归增长:

- \bullet $\epsilon \in L$:
- 2 if $w \in L$, then $\langle w \rangle$, $ww \in L$.
- 3 二义性文法 6₁:

$$S \rightarrow \langle S \rangle | SS | \epsilon$$

④ 无二义文法 G₂:

$$S \rightarrow \langle S \rangle S | \epsilon$$

- ⑤ 上述两文法等价等价.
- ⑥ 将 < 看成 start tag, > 看成 end tag, 即是 XML 语言的文法

上下支无关支法

L(G) = {由〈和〉组成的嵌套括号对}.

分析 L(G) 的语句如何递归增长:

- $\mathbf{0}$ $\epsilon \in L$:
- 2 if $w \in L$, then $\langle w \rangle$, $ww \in L$.
- 3 二义性支法 6₁:

$$S \rightarrow \langle S \rangle | SS | \epsilon$$

4 无二义文法 6₂:

$$S \rightarrow \langle S \rangle S | \epsilon$$

- ⑤ 上述两文法等价等价。

分析 L(G) 的语句如何递归增长:

- $\mathbf{0}$ $\epsilon \in L$;
- 2 if $w \in L$, then $\langle w \rangle$, $ww \in L$.
- 3 二义性支法 6₁:

$$S \rightarrow \langle S \rangle | SS | \epsilon$$

4 无二义文法 6₂:

$$S \rightarrow \langle S \rangle S | \epsilon$$

- ⑤ 上述两文法等价等价。
- 🗿 将〈看成 start tag,〉看成 end tag, 即是 XML 语言的文 法.

Example

$$S \rightarrow aS \mid Sb \mid a \mid b$$

- ② 由此看出:在句型中 a 只能出现在 S的左边, b 只能 在 S 的右边
- \odot a, b \in L(G), $\varepsilon \notin$ L(G)
- ① $L(G) = \{a^m b^n | m, n \in \mathbb{N}, m + n > 0\}$

Example

上下支无关支法

$$S \rightarrow aS \mid Sb \mid a \mid b$$

- ② 由此看出:在句型中 a 只能出现在 S的左边,b 只能在 S 的右边。
- ⓐ $a, b \in L(G), \epsilon \notin L(G)$
- **Q** $L(G) = \{a^m b^n | m, n \in \mathbb{N}, m + n > 0\}$

Example

上下支无关支法

$$S \rightarrow aS \mid Sb \mid a \mid b$$

- ② 由此看出:在句型中 a 只能出现在 S的左边,b 只能在 S 的右边
- \bigcirc a, b \in L(G), $\epsilon \notin$ L(G)
- 4 L(G) = $\{a^mb^n|m, n \in \mathbb{N}, m + n > 0\}$

Example

上下支无关支法

 $S \rightarrow aS \mid Sb \mid a \mid b$

- ② 由此看出:在句型中 a 只能出现在 S的左边,b 只能在 S 的右边.
- \odot a, b \in L(G), $\varepsilon \notin$ L(G)
- 4 L(G) = $\{a^mb^n|m, n \in \mathbb{N}, m+n > 0\}$

Example

上下支无关支法

 $S \rightarrow aS \mid Sb \mid a \mid b$

- ② 由此看出:在句型中 a 只能出现在 S的左边, b 只能在 S 的右边.
- 3 a, b \in L(G), $\epsilon \notin$ L(G)
- **1** $\mathbf{U}(G) = \{a^m b^n | m, n \in \mathbb{N}, m + n > 0\}$

L(G) = {由 a 和 b 组成的正则表达式}

分析 L(G)中的串如何递归增长:

- lefta ϵ , \emptyset , a, $b \in L$.
- ② if $r, s \in L$, then $r \mid s$, $r \mid s$, $r \mid s$, $(r) \in L$
 -) 二义性支法 G1:
 - $S \rightarrow S|S|SS|S*|(S)|a|b|\epsilon|($



上下支无关支法

Example 7 --- 正则表达式的文法设计 (1/2).

```
L(G) = {由 a 和 b 组成的正则表达式}
```

分析 L(G)中的串如何递归增长:

- \bullet ϵ , \emptyset , α , $b \in L$.
- \bigcirc if $r, s \in L$, then $r \mid s, rs, r^*, (r) \in L$
- ③ 二义性文法 G1:

 $S \rightarrow S|S|SS|S*|(S)|a|b|\epsilon|\emptyset$

上下支无关支法

Example 7 --- 正则表达式的文法设计 (1/2)

L(G) = {由 a 和 b 组成的正则表达式}

上下文无关文法的设计

分析 L(G)中的串如何递归增长:

- \bullet \bullet , \bullet , \bullet , \bullet
- 2 if $r, s \in L$, then $r \mid s, rs, r^*, (r) \in L$



Example 7 --- 正则表达式的文法设计 (1/2)

```
L(G) = {由 a 和 b 组成的正则表达式}
```

分析 L(G)中的串如何递归增长:

- \bullet \bullet , \bullet , \bullet , \bullet
- 2 if $r, s \in L$, then $r \mid s, rs, r^*, (r) \in L$
- 3 二义性支法 6₁:

 $S \rightarrow S|S|SS|S*|(S)|a|b|\epsilon|\emptyset$

Example 7 --- 正则表达式的文法设计 (2/2)

对语法成分进行分类: F记录字母或括号内容, K记 录 Kleene Closure. C记录连接运算. S记录并运算:



```
对语法成分进行分类: F 记录字母或括号内容. K 记录 Kleene Closure. C 记录连接运算. S 记录并运算: \mathfrak{o}, \mathfrak{o}, \mathfrak{o}, \mathfrak{o}, \mathfrak{o}, \mathfrak{o}, \mathfrak{o}
```

- $\forall f \in L(F), f \in L(K); if k \in L(K) then k* \in L(K);$
- ③ \forall k ∈ L(K), k ∈ L(C); if c ∈ L(C), k ∈ L(K) then ck ∈ L(C):
- $\forall c \in L(C), c \in L(S); \text{ if } s \in L(S), c \in L(C) \text{ then}$
- ⑤ 无二义运算左结合的文法 **G**₂:

Example 7 --- 正则表达式的文法设计 (2/2)

对语法成分进行分类: F记录字母或括号内容, K记 录 Kleene Closure. C记录连接运算. S记录并运算:

- \bullet ϵ , \emptyset , α , β , $(S) \in F$;
- $2 \forall f \in L(F), f \in L(K); if k \in L(K) then k* \in L(K);$

对语法成分进行分类: F记录字母或括号内容, K记 录 Kleene Closure. C记录连接运算. S记录并运算:

- \bullet ϵ , \emptyset , α , β , $(S) \in F$;
- $2 \forall f \in L(F), f \in L(K); if k \in L(K) then k* \in L(K);$
- $ck \in L(C)$;

$$S \rightarrow S |C| C$$

$$C \rightarrow CK \mid K$$

$$\mathsf{K} o \mathsf{K}^* \mid \mathsf{F}$$

Example 7 --- 正则表达式的文法设计 (2/2)

对语法成分进行分类: F记录字母或括号内容. K记录 Kleene Closure. C记录连接运算. S记录并运算:

- \bullet ϵ , \emptyset , α , β , $(S) \in F$;
- $\forall f \in L(F), f \in L(K); if k \in L(K) then k^* \in L(K);$
- \forall k \in L(K), k \in L(C); if c \in L(C), k \in L(K) then ck \in L(C);
- $\forall c \in L(C), c \in L(S)$; if $s \in L(S), c \in L(C)$ then $s|c \in L(S)$;
- ⑤ 无二义运算左结合的文法 G₂:

$$S \to S |C| C$$

$$C \to CK |K$$

$$K \to K^* |F$$

Example 7 --- 正则表达式的文法设计 (2/2)

对语法成分进行分类: F记录字母或括号内容. K记录 Kleene Closure. C记录连接运算. S记录并运算:

运法树

- \bullet ϵ , \emptyset , α , β , $(S) \in F$;
- $\forall f \in L(F), f \in L(K); if k \in L(K) then k^* \in L(K);$
- \forall k \in L(K), k \in L(C); if c \in L(C), k \in L(K) then ck \in L(C);
- $\forall c \in L(C), c \in L(S)$; if $s \in L(S), c \in L(C)$ then $s|c \in L(S)$;
- ⑤ 无二义运算左结合的文法 G₂:

$$S \to S |C| \overline{C}$$

$$C \to CK |K$$

$$K \to K^* |F$$

$$F \to a |b| \epsilon |\emptyset| (S)$$

正则表达式转换为上下文自由文法

对应关系

$$\mathsf{S} o \mathsf{aS} |\mathsf{bS}| \mathsf{abb} \quad \Longleftrightarrow \quad$$

$$\begin{array}{ccc} S & \rightarrow & aS|bS|aA \\ A & \rightarrow & bB \\ B & \rightarrow & bC \\ C & \rightarrow & \epsilon \end{array}$$

<ロト <回り < 回り < 重り < 重り 。

正则表达式转换为上下文自由文法

对应关系

上下支无关支法

$$S \rightarrow aS|bS|abb \iff$$

$$\begin{array}{ccc}
S & \to & aS|bS|aA \\
A & \to & bB \\
B & \to & bC \\
C & \to & \varepsilon
\end{array}$$

对应关系

- 2 $r|s \iff S \rightarrow S_r|S_s$;

$$S \rightarrow aS|bS|abb \iff$$

$$\begin{array}{ccc} S & \rightarrow & aS|bS|aA \\ A & \rightarrow & bB \\ B & \rightarrow & bC \\ C & \rightarrow & \epsilon \end{array}$$

对应关系

上下支无关支法

- 1 $r^* \iff S \rightarrow S_r S | \epsilon$;
- 2 $r|s \iff S \rightarrow S_r|S_s$:

(alb)*abb

$$S o aS|bS|abb \iff \left\{ egin{array}{lll} S & o & aS|bS|aA \ A & o & bB \ B & o & bC \ C & o & \epsilon \end{array}
ight.$$

◎ 形如后者的文法称为正则文法或右线性文法。

对应关系

上下支无关支法

- 2 $r|s \iff S \rightarrow S_r|S_s$;
- 3 rs \iff 5 \rightarrow 5_r5_s;

(a|b)*abb

$$S o aS|bS|abb \iff \left\{ egin{array}{lll} S & o & aS|bS|aA \ A & o & bB \ B & o & bC \ C & o & \epsilon \end{array}
ight.$$

对应关系

上下支无关支法

- 2 $r|s \iff S \rightarrow S_r|S_s$;
- 3 rs \iff 5 \rightarrow 5_r5_s;

(a|b)*abb

$$S \rightarrow aS|bS|abb \iff \begin{cases} S \rightarrow aS|bS|aA \\ A \rightarrow bB \\ B \rightarrow bC \\ C \rightarrow \epsilon \end{cases}$$

- 形如后者的文法称为正则文法或右线性文法。

对应关系

- 2 $r|s \iff S \rightarrow S_r|S_s$;
- 3 rs \iff 5 \rightarrow S_rS_s;

(a|b)*abb

$$S \rightarrow aS|bS|abb \iff \begin{cases} S \rightarrow aS|bS|aA \\ A \rightarrow bB \\ B \rightarrow bC \\ C \rightarrow \epsilon \end{cases}$$

- 形如后者的文法称为正则文法或右线性文法。
- 正则语言都能用 CFG 描述,所以 CFG 的表达能力强于前者.

Example --- 基于字母的表达式文法

```
\exp \rightarrow \exp + \exp | \exp \times \exp | id

id \rightarrow letter rest

letter \rightarrow a | b | \dots | z

rest \rightarrow letter rest | digit rest | \epsilon

digit \rightarrow 0 | 1 | \dots | 9
```

Example --- 基于字母的表达式文法

```
\exp \rightarrow \exp + \exp | \exp \times \exp | id
       id \rightarrow letter rest
letter \rightarrow a | b | \dots | z
   rest \rightarrow letter rest | digit rest | \epsilon
  digit \rightarrow 0 \mid 1 \mid \dots \mid 9
```

上下文无关文法的设计

Example --- 基于字母的表达式文法

```
\exp \rightarrow \exp + \exp | \exp \times \exp | id
       id \rightarrow letter rest
letter \rightarrow a | b | \dots | z
   rest \rightarrow letter rest | digit rest | \epsilon
  digit \rightarrow 0 \mid 1 \mid \dots \mid 9
```

- 上下文无关文法的表达能力比正则表达式强,因此形式语言的识 别宗全可以一级抽象"字母→ 语法",不需要"字母→ 单 词→语法"二级抽象。

Example --- 基于字母的表达式文法

```
\exp \rightarrow \exp + \exp | \exp \times \exp | id
id \rightarrow letter rest
letter \rightarrow a | b | \dots | z
rest \rightarrow letter rest | digit rest | \epsilon
digit \rightarrow 0 | 1 | \dots | 9
```

- 上下文无关文法的表达能力比正则表达式强,因此形式语言的识别完全可以一级抽象"字母→语法",不需要"字母→单词→语法"二级抽象。
- 文法的识别证比正则表达式复杂.
- 采用二级抽象可以显著降低语法分析的负担
- Extended Backus-Naur Form(EBNF) 会许产生式右边为正则表达式 exp → term(+term)*

这样只用一级抽象就可以同时描述词法和语法,如 XML 规范,语法自动生成工具 ANTLR(http://www.antlr.org/) 等采用 EBNF,但在识别时还是分解式两级结构

Example --- 基于字母的表达式文法

```
\exp \rightarrow \exp + \exp | \exp \times \exp | id
id \rightarrow letter rest
letter \rightarrow a | b | \dots | z
rest \rightarrow letter rest | digit rest | \epsilon
digit \rightarrow 0 | 1 | \dots | 9
```

- 上下文无关文法的表达能力比正则表达式强,因此形式语言的识别完全可以一级抽象"字母→语法",不需要"字母→单词→语法"二级抽象。
- 文法的识别证比正则表达式复杂.
- 采用二级抽象可以显著降低语法分析的负担.
- Extended Backus-Naur Form(EBNF)允许产生式右边为正则表达式 exp → term(+term)*

这样只用一级抽象就可以同时描述词法和语法,如 XML 规范,语法自动生成工具 ANTLR(http://www.antlr.org/) 等采用 EBNF,但在识别时还是分解式两级结构

Example --- 基于字母的表达式文法

```
\exp \rightarrow \exp + \exp | \exp \times \exp | id
       id \rightarrow letter rest
letter \rightarrow a | b | \dots | z
   rest \rightarrow letter rest | digit rest | \epsilon
  digit \rightarrow 0 \mid 1 \mid \dots \mid 9
```

- 上下文无关文法的表达能力比正则表达式强,因此形式语言的识 别完全可以一级抽象"字母→ 语法"。不需要"字母→ 单 词 → 连法"二级抽象
- 文法的识别证比正则表达式复杂。
- 采用二级抽象可以显著降低语法分析的负担.
- Extended Backus-Naur Form(EBNF)允许产生式右边为正则表达式: $exp \rightarrow term (+ term)^*$

这样只用一级抽象就可以同时描述词法和语法,如 XML 规范,语 法自动生成工具 ANTLR(http://www.antlr.org/) 等采用 EBNF, 但在识别时还是分解成两级结构

对应关系

对应关系

每个状态对应一个非终结符,开始状态对应文法开始符号。

上下支充关支法的设计

对应关系

① 每个状态对应一个非终结符,开始状态对应文法开始符号。

上下支充关支法的设计

- 2 if $\stackrel{a}{\longrightarrow}$ $\stackrel{\bullet}{\longrightarrow}$ $\stackrel{\bullet}{\longrightarrow}$, then: $S_i \rightarrow aS_i$.

对应关系

每个状态对应一个非终结符,开始状态对应文法开始符号。

上下支充关支法的设计

- 2 if $\stackrel{a}{\longrightarrow}$ $\stackrel{j}{\longrightarrow}$, then: $S_i \rightarrow aS_j$.
- 3 如果①是接受状态,则增加产生式: S_i → ε;

对应关系

每个状态对应一个非终结符,开始状态对应文法开始符号。

上下支充关支法的设计

- 2 if $\stackrel{a}{\longrightarrow}$ $\stackrel{\bullet}{\longrightarrow}$ $\stackrel{\bullet}{\longrightarrow}$, then: $S_i \rightarrow aS_i$.
- ③ 如果①是接受状态、则增加产生式: 5; → ε;

对应关系

▲ 每个状态对应一个非终结符,开始状态对应文法开始符号。

上下支充关支法的设计

- 2 if $\stackrel{a}{\longrightarrow}$ $\stackrel{\bullet}{\longrightarrow}$ $\stackrel{\bullet}{\longrightarrow}$, then: $S_i \rightarrow aS_i$.
- 3 如果①是接受状态,则增加产生式: 5;→ ε;



自动机转换为上下交自由交法

对应关系

上下支无关支法

- 每个状态对应一个非终结符,开始状态对应文法开始符号。
- 2 if $\stackrel{a}{\longrightarrow}$ $\stackrel{\bullet}{\longrightarrow}$ $\stackrel{\bullet}{\longrightarrow}$, then: $S_i \rightarrow aS_j$.
- ③ 如果①是接受状态,则增加产生式: $S_i \rightarrow \epsilon$;

自动机转换为上下交自由交法

对应关系

- ▲ 每个状态对应一个非终结符,开始状态对应文法开始符号。
- 2 if $\stackrel{a}{\longrightarrow}$ $\stackrel{\bullet}{\longrightarrow}$ $\stackrel{\bullet}{\longrightarrow}$, then: $S_i \rightarrow aS_j$.
- ③ 如果①是接受状态,则增加产生式: $S_i \rightarrow \epsilon$;

Syntactic sugar is extra syntax in a programming language that makes the code easier to understand or write.

上下文无关文法



Syntactic sugar is extra syntax in a programming language that makes the code easier to understand or write.

Example

- $C: E \to E[E]$ is an extra pruduction for simply *(E + E). so 2["abc"] is correct C expression.
- C#:
 public class Student {
 public string Name { get; set; }
 }
- Syntactic sugar is popular in functional programming languages. e.g.
 Haskell is much sweeter

 ``sweet or salty'' is always a debate in programming language designs

Syntactic sugar is extra syntax in a programming language that makes the code easier to understand or write.

- C: $E \to E[E]$ is an extra pruduction for simply *(E + E). so 2["abc"] is correct C expression.

Syntactic sugar is extra syntax in a programming language that makes the code easier to understand or write.

Example

- C: $E \to E[E]$ is an extra pruduction for simply *(E + E). so 2["abc"] is correct C expression.
- C#:

```
public class Student {
  public string Name { get; set; }
}
```

 Syntactic sugar is popular in functional programming languages. e.g. Haskell is much sweeter

 ``sweet or salty'' is always a debate in programming language designs.

Syntactic sugar is extra syntax in a programming language that makes the code easier to understand or write.

Example

- C: $E \to E[E]$ is an extra pruduction for simply *(E + E). so 2["abc"] is correct C expression.
- C#:

```
public class Student {
 public string Name { get; set; }
```

 Syntactic sugar is popular in functional programming languages. e.g. Haskell is much sweeter.

sweet	unsweet
a `mappend` b	mappend a b
(+ 2)	\x -> x + 2
(x, y)	(,) x y
[1, 2, 3]	(:) 1 ((:) 2 ((:) 3 []))

Syntactic sugar is extra syntax in a programming language that makes the code easier to understand or write.

Example

- C: $E \to E[E]$ is an extra pruduction for simply *(E + E). so 2["abc"] is correct C expression.
- C#:

```
public class Student {
 public string Name { get; set; }
```

 Syntactic sugar is popular in functional programming languages. e.g. Haskell is much sweeter.

sweet	unsweet
a `mappend` b	mappend a b
(+ 2)	\x -> x + 2
(x, y)	(,) x y
[1, 2, 3]	(:) 1 ((:) 2 ((:) 3 []))

``sweet or salty'' is always a debate in programming language designs.

Pumping lemma

Let L be a context-free language. There is $p \in N^+$ such that for any sentence z with the length $|z| \ge p$, there are $u, v, w, x, y \in \Sigma^*$ with the following properties:

```
\circ z = uvwxy.
```

```
    ∨X ≠ ε.
```

$$\mathsf{vx}^\mathsf{n}\mathsf{y} \in \mathsf{L}$$
 forall $\mathsf{n} \in \mathsf{N}$.

Pumping lemma

Let L be a context-free language. There is $p \in N^+$ such that for any sentence z with the length $|z| \ge p$, there are $u, v, w, x, y \in \Sigma^*$ with the following properties:

- z = uvwxy.
- VX ≠ ε.
- |vwx| < p.
- $uv^nwx^nv \in L$ for all $n \in N$.

Pumping lemma

Let L be a context-free language. There is $p \in \mathbb{N}^+$ such that for any sentence z with the length $|z| \ge p$, there are $u, v, w, x, y \in \Sigma^*$ with the following properties:

- z = uvwxy.
- vx ≠ ε.
- |vwx| < p.
- $uv^nwx^ny \in L$ for all $n \in N$.

Pumping lemma

Let L be a context-free language. There is $p \in \mathbb{N}^+$ such that for any sentence z with the length $|z| \ge p$, there are $u, v, w, x, y \in \Sigma^*$ with the following properties:

- z = uvwxy.
- vx ≠ ε.
- |vwx| < p.
- $uv^nwx^ny \in L$ for all $n \in N$.



上下文无关文法的设计

CFG 的表达能力

Pumping lemma

Let L be a context-free language. There is $p \in \mathbb{N}^+$ such that for any sentence z with the length $|z| \ge p$, there are $u, v, w, x, y \in \Sigma^*$ with the following properties:

- z = uvwxy.
- ∨x ≠ ε.
- | vwx | < p.</p>
- $uv^nwx^ny \in L$ for all $n \in N$.

◆□▶ ◆圖▶ ◆臺▶ ◆臺▶ · 臺

Pumping lemma

Let L be a context-free language. There is $p \in \mathbb{N}^+$ such that for any sentence z with the length $|z| \ge p$, there are $u, v, w, x, y \in \Sigma^*$ with the following properties:

- Z = UVWXY.
- ∨x ≠ ε.
- | vwx | < p.</p>
- $uv^nwx^ny \in L$ for all $n \in N$.

Pumping lemma

Let L be a context-free language. There is $p \in \mathbb{N}^+$ such that for any sentence z with the length $|z| \ge p$, there are $u, v, w, x, y \in \Sigma^*$ with the following properties:

- z = uvwxy.
- vx ≠ ε.
- | vwx | < p.
 </p>
- $uv^nwx^ny \in L$ for all $n \in N$.

- {aⁿbⁿcⁿ | n ∈ N} 不是 CFL,但 {a^mbⁿc^p | m, n, p ∈ N} 是 CFL.
- {ww | w ∈ {a, b}*} 不是 CFL, 但 {ww^R | w ∈ {a, b}*} 是 CFL.

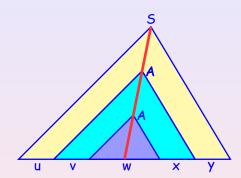
Pumping lemma

Let L be a context-free language. There is $p \in \mathbb{N}^+$ such that for any sentence z with the length $|z| \ge p$, there are $u, v, w, x, y \in \Sigma^*$ with the following properties:

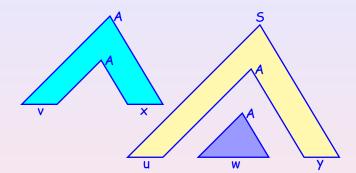
- Z = UVWXY.
- ∨x ≠ ε.
- | vwx| < p.</p>
- $uv^nwx^ny \in L$ for all $n \in N$.

- {aⁿbⁿcⁿ | n ∈ N} 不是 CFL,但 {a^mbⁿc^p | m, n, p ∈ N} 是 CFL.
- {ww | w ∈ {a,b}*} 不是 CFL, 但 {ww^R | w ∈ {a,b}*} 是 CFL.

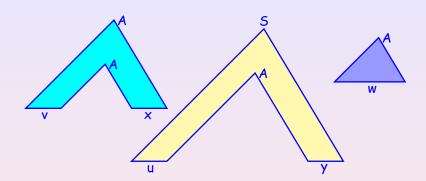
Illustration

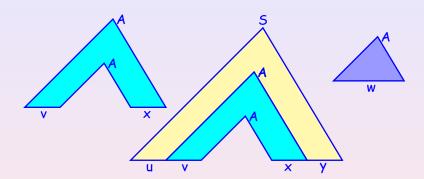


上下文无关文法

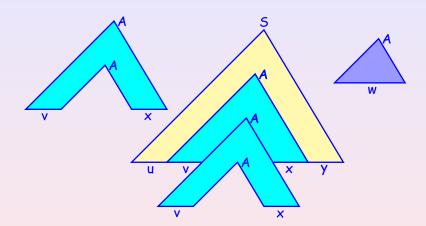


Illustration

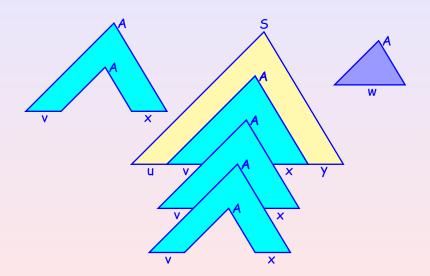




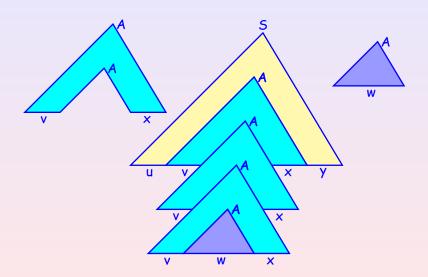




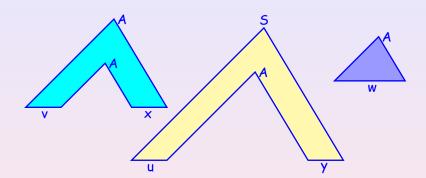




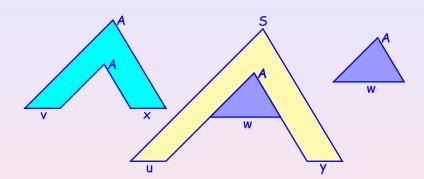














- - 递归定义与产生式
 - 上下文无关文法
 - 上下文无关语言
- - 语法树
 - 语法树与维导的关系
 - ●二义性
- 上下文无关文法的设计
 - 设计举例
 - 正则表达式转换为上下支自由支法
 - 自动机转换为上下支自由支法
 - CFG 的表达能力
- 二义性的再讨论

上下文无关文法

最早出现在 Algol60 的设计中.

if-then-else 运句

stmt → if expr then stmt
| if expr then stmt else stmt
| other

dangling-else

当 else 与 if 不平衡出现时,else 如何挂靠 if 产生二义性

最早出现在 Algol60 的设计中.

if-then-else 语句

stmt → if expr then stmt
| if expr then stmt else stmt
| other

dangling-else

当 else 与 if 不平衡出现时,else 如何挂靠 if 产生二义性.



最早出现在 Algol60 的设计中.

if-then-else 语句

```
stmt → if expr then stmt
| if expr then stmt else stmt
| other
```

dangling-else

当 else 与 if 不平衡出现时,else 如何挂靠 if 产生二义性.

- 平衡出现:
 - if E_1 then $\mathsf{S1}$ else if E_2 then S_2 else S_3
- 不平衡出现:
 - if E₁ then if E₂ then S₁ else S₂

最早出现在 Algol60 的设计中.

if-then-else 语句

```
stmt → if expr then stmt
| if expr then stmt else stmt
| other
```

dangling-else

当 else 与 if 不平衡出现时,else 如何挂靠 if 产生二义性.

● 平衡出现:

```
if E_1 then S1 else if E_2 then S_2 else S_3
```

● 不平衡出现:

if E_1 then if E_2 then S_1 else S_2 else 可以转靠第一个 if. 也可以是第二个 i

最早出现在 Algol60 的设计中.

if-then-else 语句

```
stmt → if expr then stmt
| if expr then stmt else stmt
| other
```

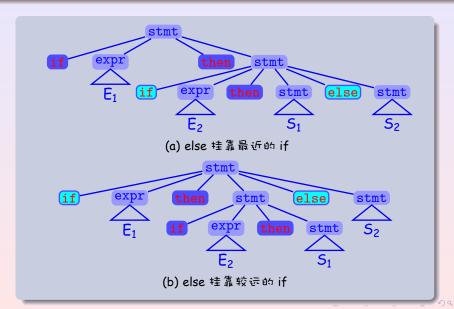
dangling-else

当 else 与 if 不平衡出现时,else 如何挂靠 if 产生二义性.

- 平衡出现:
 - if E_1 then S1 else if E_2 then S_2 else S_3
- 不平衡出现:

if E₁ then if E₂ then S₁ else S₂ else 可以挂靠第一个if, 也可以是第二个if.

两颗不同的分析树



解消二义性

让 else 挂靠最近的 if.

当这二义唯后的意法

Example

```
\underbrace{\text{if } E_1 \text{ then } \underbrace{\text{if } E_2 \text{ then } S_1 \text{ else } S_2}_{\text{m\_stmt}}}_{\text{u\_stmt}}
```

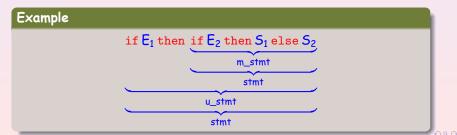
解消二义性

让 else 挂靠最近的 if.

```
消除二义性后的文法
       \rightarrow m_stmt | u_stmt
       m_stmt → if expr then m_stmt else m_stmt
                    other
       u\_stmt \rightarrow if expr then stmt
                    if expr then m_stmt else u_stmt
```

解消二义性

让 else 挂靠最近的 if.



```
解消二义性
```

```
不能解消二义性的文法
        stmt
                    if expr then stmt
                    m stmt
        m_stmt
                → if expr then m_stmt else stmt
                    other
```

不能解消二义性的文法

```
stmt → if expr then stmt

| m_stmt

m_stmt → if expr then m_stmt else stmt

| other
```

m_stmt 的 else 部分可能会导致不平衡 if E_1 then if E_2 then S_1 else if E_3 then S_2 else S_3 stmt m_stmt if E_1 then if E_2 then S_1 else if E_3 then S_2 else S_3 m_stmt m_stmt

stmt

- 程序设计语言一般是 CFL.

CFG 和自动机的关系

- 程序设计语言一般是 CFL.
- DFA 不能识别 CFL.
- CFG 与不确定的下堆自动机等价.
- 不确定的下推自动机与确定的下推自动机不等价。
- 无回溯的语法分析器仅能识别部分的 CFL

CFG 和自动机的关系

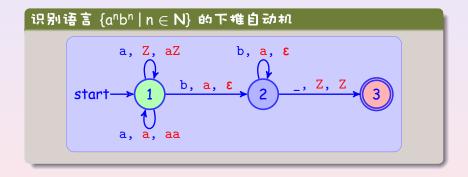
- 程序设计语言一般是 CFL.
- DFA 不能识别 CFL.
- CFG 与不确定的下推自动机等价.
- 不确定的下推自动机与确定的下推自动机不等价.
- 无回溯的语法分析器仅能识别部分的 CFL

CFG 和自动机的关系

- 程序设计语言一般是 CFL.
- DFA 不能识别 CFL.
- CFG 与不确定的下推自动机等价.
- 不确定的下推自动机与确定的下推自动机不等价.
- 无回溯的语法分析器仅能识别部分的 CFL

- 程序设计语言一般是 CFL.
- DFA 不能识别 CFL.
- CFG 与不确定的下推自动机等价。
- 不确定的下推自动机与确定的下推自动机不等价。
- 无回溯的语法分析器仅能识别部分的 CFL.

Example



本章小节

- 1 上下文无关文法
 - 递归定义与产生式
 - ◎ 上下文无关文法
 - 上下文无关语言
- 2 语法树
 - 语法树
 - 语法树与椎具的关系
 - 二义性
- ③ 上下文无关文法的设计
 - 设计举例
 - 正则表达式转换为上下文自由文法
 - 自动机转换为上下文自由文法
 - CFG 的表达能力
- 4 二义性的再讨论