

## 二、语法分析

### (5. 上下文无关文法)

魏恒峰

hfwei@nju.edu.cn

2024 年 03 月 29 日



```

20 functionDecl : type ID '(' formalParameters? ')' block ;
21
22 formalParameters : formalParameter (',' formalParameter)* ;
23
24 formalParameter : type ID ;
25
26 block : '{' stat* '}' ;
27
28 stat : block      # BlockStat
29      | varDecl    # VarDeclStat
30      | 'if' expr 'then' stat ('else' stat)? # IfStat
31      | 'return' expr? ';'      # ReturnStat
32      | expr '=' expr ';'      # AssignStat
33      | expr ';' # ExprStat
34 ;

```



## <Context-Free Grammar>

上下文无关文法 (CFG)

## Definition (Context-Free Grammar (CFG); 上下文无关文法)

上下文无关文法  $G$  是一个四元组  $G = (T, N, S, P)$ :

- ▶  $T$  是**终结符号** (Terminal) 集合, 对应于词法分析器产生的词法单元
- ▶  $N$  是**非终结符号** (Non-terminal) 集合
- ▶  $S$  是**开始** (Start) 符号 ( $S \in N$  且唯一)
- ▶  $P$  是**产生式** (Production) 集合

$$A \in N \longrightarrow \alpha \in (T \cup N)^*$$

头部/左部 (Head)  $A$ : **单个**非终结符

体部/右部 (Body)  $\alpha$ : 终结符与非终结符构成的串, 也可以是空串  $\epsilon$

$$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$$

$$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$$

$$E \rightarrow TE'$$

$$E' \rightarrow + TE' \mid \epsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow * FT' \mid \epsilon$$

$$F \rightarrow (E) \mid \mathbf{id}$$

$$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$$

$$E \rightarrow TE'$$

$$E' \rightarrow + TE' \mid \epsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow * FT' \mid \epsilon$$

$$F \rightarrow (E) \mid \mathbf{id}$$

$$S \rightarrow aSa$$

$$S \rightarrow bSb$$

$$S \rightarrow a$$

$$S \rightarrow b$$

$$S \rightarrow \epsilon$$

```
20  functionDecl : type ID '(' formalParameters? ')' block ;
21
22  formalParameters : formalParameter (',' formalParameter)* ;
23
24  formalParameter : type ID ;
25
26  block : '{' stat* '}' ;
27
28  stat : block      # BlockStat
29       | varDecl   # VarDeclStat
30       | 'if' expr 'then' stat ('else' stat)? # IfStat
31       | 'return' expr? ';'      # ReturnStat
32       | expr '=' expr ';'      # AssignStat
33       | expr ';' # ExprStat
34       ;
```



# [Extended] Backus-Naur form ([E]BNF)

+ \* ?



John Backus  
(1924 ~ 2007)



Peter Naur  
(1928 ~ 2016)



Niklaus Wirth  
(1934 ~ 2024)

## [Extended] Backus-Naur form ([E]BNF)

+ \* ?



John Backus  
(1924 ~ 2007)

1977 (FORTRAN)



Peter Naur  
(1928 ~ 2016)

2005 (ALGOL60)



Niklaus Wirth  
(1934 ~ 2024)

1984 (PLs; PASCAL)

## Context-Sensitive Grammar (CSG)

$$S \rightarrow aBC$$

$$S \rightarrow aSBC$$

$$CB \rightarrow CZ$$

$$CZ \rightarrow WZ$$

$$WZ \rightarrow WC$$

$$WC \rightarrow BC$$

$$aB \rightarrow ab$$

$$bB \rightarrow bb$$

$$bC \rightarrow bc$$

$$cC \rightarrow cc$$

$$\alpha A \beta \rightarrow \alpha \gamma \beta \quad A \in N \quad \alpha, \beta \in (N \cup \Sigma \setminus \{S\})^* \quad \gamma \in (N \cup \Sigma \setminus \{S\})^+$$

# Syntax

# Semantics

语义: 上下文无关文法  $G$  定义了一个**语言**  $L(G)$



Syntax

Semantics

语义: 上下文无关文法  $G$  定义了一个**语言**  $L(G)$

语言是**串**的集合

串从何来?

## 推导 (Derivation)

$$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$$

推导即是将某个产生式的左边**替换**成它的右边

每一步推导需要选择**替换哪个非终结符号**, 以及**使用哪个产生式**

## 推导 (Derivation)

$$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$$

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(\mathbf{id} + E) \Rightarrow -(\mathbf{id} + \mathbf{id})$$

## 推导 (Derivation)

$$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$$

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(\mathbf{id} + E) \Rightarrow -(\mathbf{id} + \mathbf{id})$$

$E \Rightarrow -E$  : 经过一步推导得出

$E \xRightarrow{+} -(\mathbf{id} + E)$  : 经过一步或多步推导得出

$E \xRightarrow{*} -(\mathbf{id} + E)$  : 经过零步或多步推导得出



## 推导 (Derivation)

$$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$$

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(\mathbf{id} + E) \Rightarrow -(\mathbf{id} + \mathbf{id})$$

$E \Rightarrow -E$  : 经过一步推导得出

$E \xRightarrow{+} -(\mathbf{id} + E)$  : 经过一步或多步推导得出

$E \xRightarrow{*} -(\mathbf{id} + E)$  : 经过零步或多步推导得出

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(E + \mathbf{id}) \Rightarrow -(\mathbf{id} + \mathbf{id})$$

## 推导 (Derivation)

$$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$$

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(\mathbf{id} + E) \Rightarrow -(\mathbf{id} + \mathbf{id})$$

$E \Rightarrow -E$  : 经过一步推导得出

$E \xRightarrow{+} -(\mathbf{id} + E)$  : 经过一步或多步推导得出

$E \xRightarrow{*} -(\mathbf{id} + E)$  : 经过零步或多步推导得出

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(E + \mathbf{id}) \Rightarrow -(\mathbf{id} + \mathbf{id})$$

Leftmost (最左) Derivation      Rightmost (最右) Derivation

## Definition (Sentential Form; 句型)

如果  $S \xRightarrow{*} \alpha$ , 且  $\alpha \in (T \cup N)^*$ , 则称  $\alpha$  是文法  $G$  的一个**句型**。

$$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$$

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id} + E) \Rightarrow -(\mathbf{id} + \mathbf{id})$$

## Definition (Sentential Form; 句型)

如果  $S \xRightarrow{*} \alpha$ , 且  $\alpha \in (T \cup N)^*$ , 则称  $\alpha$  是文法  $G$  的一个**句型**。

$$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$$

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id} + E) \Rightarrow -(\mathbf{id} + \mathbf{id})$$

## Definition (Sentence; 句子)

如果  $S \xRightarrow{*} w$ , 且  $w \in T^*$ , 则称  $w$  是文法  $G$  的一个**句子**。

Definition (文法  $G$  生成的语言  $L(G)$ )

文法  $G$  的**语言**  $L(G)$  是它能推导出的**所有句子**构成的集合。

$$L(G) = \{w \mid S \xRightarrow{*} w\}$$

关于文法  $G$  的**两个基本问题**:

- ▶ **Membership 问题**: 给定字符串  $x \in T^*$ ,  $x \in L(G)$ ?
- ▶  $L(G)$  究竟是什么?

给定字符串  $x \in T^*$ ,  $x \in L(G)$ ?  
(即, 检查  $x$  是否符合文法  $G$ )

给定字符串  $x \in T^*$ ,  $x \in L(G)$ ?

(即, 检查  $x$  是否符合文法  $G$ )

这就是**语法分析器**的任务:

为输入的词法单元流寻找推导、**构建语法分析树**, 或者报错



$L(G)$  是什么?

这是**程序设计语言设计者**需要考虑的问题

$$S \rightarrow SS$$

$$S \rightarrow (S)$$

$$S \rightarrow \epsilon$$

$$L(G) =$$

$$S \rightarrow SS$$

$$S \rightarrow (S)$$

$$S \rightarrow \epsilon$$

$$L(G) = \{\text{良匹配括号串}\}$$

$$S \rightarrow SS$$

$$S \rightarrow (S)$$

$$S \rightarrow \epsilon$$

$$L(G) = \{\text{良匹配括号串}\}$$

$$S \rightarrow aSb$$

$$S \rightarrow \epsilon$$

$$L(G) =$$

$$S \rightarrow SS$$

$$S \rightarrow (S)$$

$$S \rightarrow \epsilon$$

$$L(G) = \{\text{良匹配括号串}\}$$

$$S \rightarrow aSb$$

$$S \rightarrow \epsilon$$

$$L(G) = \{a^n b^n \mid n \geq 0\}$$

字母表  $\Sigma = \{a, b\}$  上的所有回文串 (Palindrome) 构成的语言

字母表  $\Sigma = \{a, b\}$  上的所有**回文串** (Palindrome) 构成的语言

$$S \rightarrow aSa$$

$$S \rightarrow bSb$$

$$S \rightarrow a$$

$$S \rightarrow b$$

$$S \rightarrow \epsilon$$

$$\{b^n a^m b^{2n} \mid n \geq 0, m \geq 0\}$$



$$\{b^n a^m b^{2n} \mid n \geq 0, m \geq 0\}$$

$$S \rightarrow bSbb \mid A$$

$$A \rightarrow aA \mid \epsilon$$

$\{x \in \{a, b\}^* \mid x \text{ 中 } a, b \text{ 个数相同}\}$

$\{x \in \{a, b\}^* \mid x \text{ 中 } a, b \text{ 个数相同}\}$

$$V \rightarrow aVb \mid bVa \mid \epsilon$$

$\{x \in \{a, b\}^* \mid x \text{ 中 } a, b \text{ 个数相同}\}$

$$V \rightarrow aVb \mid bVa \mid \epsilon$$

$$V \rightarrow aVbV \mid bVaV \mid \epsilon$$

$\{x \in \{a, b\}^* \mid x \text{ 中 } a, b \text{ 个数相同}\}$

$$V \rightarrow aVb \mid bVa \mid \epsilon$$

$$V \rightarrow aVbV \mid bVaV \mid \epsilon$$

$$V \rightarrow VV \mid aVb \mid bVa \mid \epsilon$$

$\{x \in \{a, b\}^* \mid x \text{ 中 } a, b \text{ 个数相同}\}$

$$V \rightarrow aVb \mid bVa \mid \epsilon$$

$$V \rightarrow aVbV \mid bVaV \mid \epsilon$$

$$V \rightarrow VV \mid aVb \mid bVa \mid \epsilon$$

(By 香港中文大学 (深圳) 华同学)

$$\{x \in \{a, b\}^* \mid x \text{ 中 } a, b \text{ 个数不同}\}$$

$\{x \in \{a, b\}^* \mid x \text{ 中 } a, b \text{ 个数不同}\}$

$$S \rightarrow T \mid U$$

$$T \rightarrow VaT \mid VaV$$

$$U \rightarrow VbU \mid VbV$$

$$V \rightarrow aVbV \mid bVaV \mid \epsilon$$



$$S \rightarrow aBC$$

$$S \rightarrow aSBC$$

$$CB \rightarrow CZ$$

$$CZ \rightarrow WZ$$

$$WZ \rightarrow WC$$

$$WC \rightarrow BC$$

$$aB \rightarrow ab$$

$$bB \rightarrow bb$$

$$bC \rightarrow bc$$

$$cC \rightarrow cc$$

$$S \rightarrow aBC$$

$$S \rightarrow aSBC$$

$$CB \rightarrow CZ$$

$$CZ \rightarrow WZ$$

$$WZ \rightarrow WC$$

$$WC \rightarrow BC$$

$$aB \rightarrow ab$$

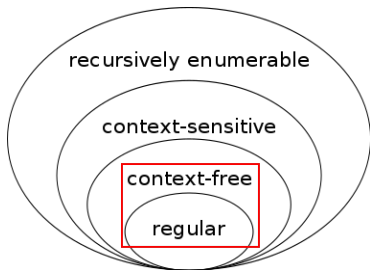
$$bB \rightarrow bb$$

$$bC \rightarrow bc$$

$$cC \rightarrow cc$$

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

为什么不使用优雅、强大的**正则表达式**描述程序设计语言的语法？



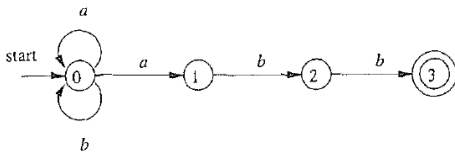
正则表达式的表达能力**严格弱于**上下文无关文法

每个正则表达式  $r$  对应的语言  $L(r)$  都可以使用上下文无关文法来描述

$$r = (a|b)^*abb$$

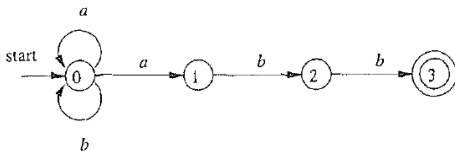
每个正则表达式  $r$  对应的语言  $L(r)$  都可以使用上下文无关文法来描述

$$r = (a|b)^*abb$$



每个正则表达式  $r$  对应的语言  $L(r)$  都可以使用上下文无关文法来描述

$$r = (a|b)^*abb$$



$$\begin{aligned} A_0 &\rightarrow aA_0 \mid bA_0 \mid aA_1 \\ A_1 &\rightarrow bA_2 \\ A_2 &\rightarrow bA_3 \\ A_3 &\rightarrow \epsilon \end{aligned}$$

此外, 若  $\delta(A_i, \epsilon) = A_j$ , 则添加  $A_i \rightarrow A_j$

$$S \rightarrow aSb$$

$$S \rightarrow \epsilon$$

$$L = \{a^n b^n \mid n \geq 0\}$$

该语言**无法**使用正则表达式来描述

## Theorem

$L = \{a^n b^n \mid n \geq 0\}$  无法使用正则表达式描述。



## Theorem

$L = \{a^n b^n \mid n \geq 0\}$  无法使用正则表达式描述。

## 反证法

## Theorem

$L = \{a^n b^n \mid n \geq 0\}$  无法使用正则表达式描述。

### 反证法

假设存在正则表达式  $r$ :  $L(r) = L = \{a^n b^n \mid n \geq 0\}$

## Theorem

$L = \{a^n b^n \mid n \geq 0\}$  无法使用正则表达式描述。

### 反证法

假设存在正则表达式  $r$ :  $L(r) = L = \{a^n b^n \mid n \geq 0\}$

则存在有限状态自动机  $D(r)$ :  $L(D(r)) = L$ ; 设其状态数为  $k \geq 1$

## Theorem

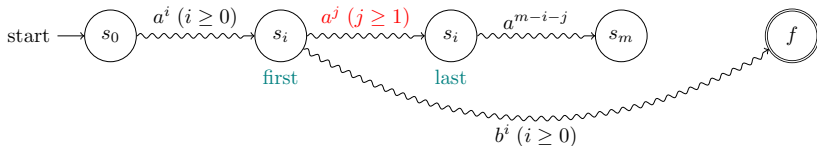
$L = \{a^n b^n \mid n \geq 0\}$  无法使用正则表达式描述。

### 反证法

假设存在正则表达式  $r: L(r) = L = \{a^n b^n \mid n \geq 0\}$

则存在有限状态自动机  $D(r): L(D(r)) = L$ ; 设其状态数为  $k \geq 1$

考虑输入  $a^m$  ( $m \geq k$ )



## Theorem

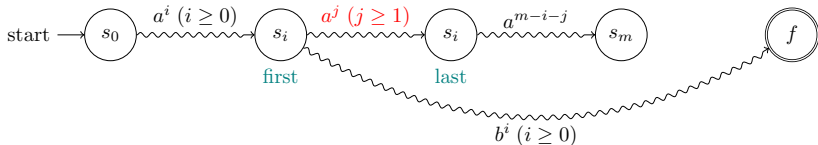
$L = \{a^n b^n \mid n \geq 0\}$  无法使用正则表达式描述。

### 反证法

假设存在正则表达式  $r: L(r) = L = \{a^n b^n \mid n \geq 0\}$

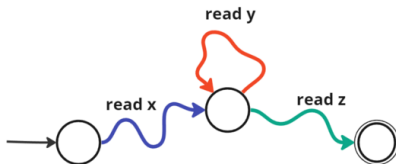
则存在有限状态自动机  $D(r): L(D(r)) = L$ ; 设其状态数为  $k \geq 1$

考虑输入  $a^m$  ( $m \geq k$ )

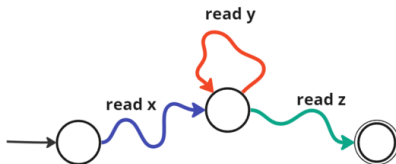


$D(r)$  也能接受  $a^{i+j}b^i$ ; 矛盾!

## Pumping Lemma for Regular Languages (@ wiki)



## Pumping Lemma for Regular Languages (@ wiki)



### Theorem

If  $L$  is a regular language, then there exists a number  $p \geq 1$  (pumping length) such that any string  $s$  in  $L$  of length  $\geq p$  can be divided into three pieces,  $s = xyz$ , satisfying the following conditions:

- (i)  $|y| \geq 1$
- (ii)  $|xy| \leq p$
- (iii)  $\forall i \geq 0 : xy^i z \in L$

## Example

$D = \{1^{n^2} \mid n \geq 0\}$  is not regular.



## Example

$D = \{1^{n^2} \mid n \geq 0\}$  is not regular.

考虑  $s = 1^{p^2}$  ( $p$  is the pumping length)

## Example

$D = \{1^{n^2} \mid n \geq 0\}$  is not regular.

考虑  $s = 1^{p^2}$  ( $p$  is the pumping length)

$$|s| \geq p$$

## Example

$D = \{1^{n^2} \mid n \geq 0\}$  is not regular.

考虑  $s = 1^{p^2}$  ( $p$  is the pumping length)

$$|s| \geq p$$

$$s = xyz$$

## Example

$D = \{1^{n^2} \mid n \geq 0\}$  is not regular.

考虑  $s = 1^{p^2}$  ( $p$  is the pumping length)

$$|s| \geq p$$

$$s = xyz$$

$$xy^2z \in D$$

## Example

$D = \{1^{n^2} \mid n \geq 0\}$  is not regular.

考虑  $s = 1^{p^2}$  ( $p$  is the pumping length)

$$|s| \geq p$$

$$s = xyz$$

$$xy^2z \in D$$

$$p^2 < |xy^2z| = |xyz| + |y| \leq p^2 + p < p^2 + 2p + 1 = (p+1)^2$$

## Example

$D = \{1^{n^2} \mid n \geq 0\}$  is not regular.

考虑  $s = 1^{p^2}$  ( $p$  is the pumping length)

$$|s| \geq p$$

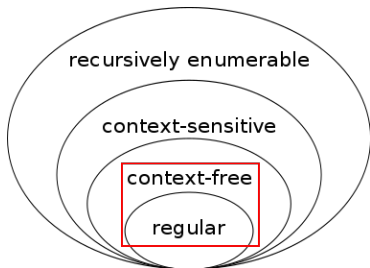
$$s = xyz$$

$$xy^2z \in D$$

$$p^2 < |xy^2z| = |xyz| + |y| \leq p^2 + p < p^2 + 2p + 1 = (p+1)^2$$

$$xy^2z \notin D$$

$L = \{a^n b^n \mid n \geq 0\}$  无法使用正则表达式描述



Q : 上下文无关文法描述能力的界限在那里?

$$S \rightarrow aBC$$

$$S \rightarrow aSBC$$

$$CB \rightarrow CZ$$

$$CZ \rightarrow WZ$$

$$WZ \rightarrow WC$$

$$WC \rightarrow BC$$

$$aB \rightarrow ab$$

$$bB \rightarrow bb$$

$$bC \rightarrow bc$$

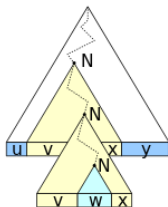
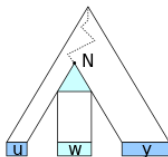
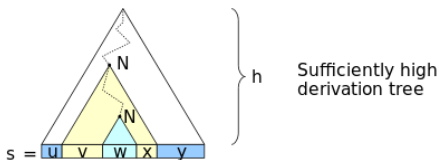
$$cC \rightarrow cc$$

$$L = \{a^n b^n c^n \mid n \geq 1\}$$



$$L = \{a^n b^n c^n \mid n \geq 0\}$$

## Pumping Lemma for Context-free Languages (@ wiki)



Generating  $uv^0wx^0y$

Generating  $uv^2wx^2y$

Thank  
You!



Office 926

hfwei@nju.edu.cn