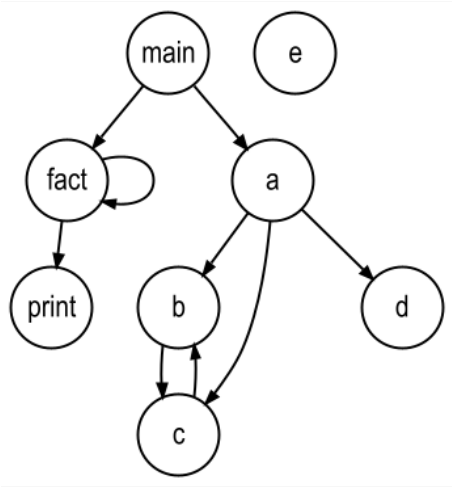
语法分析
(1. ANTLR 4 语法分析器)

魏恒峰

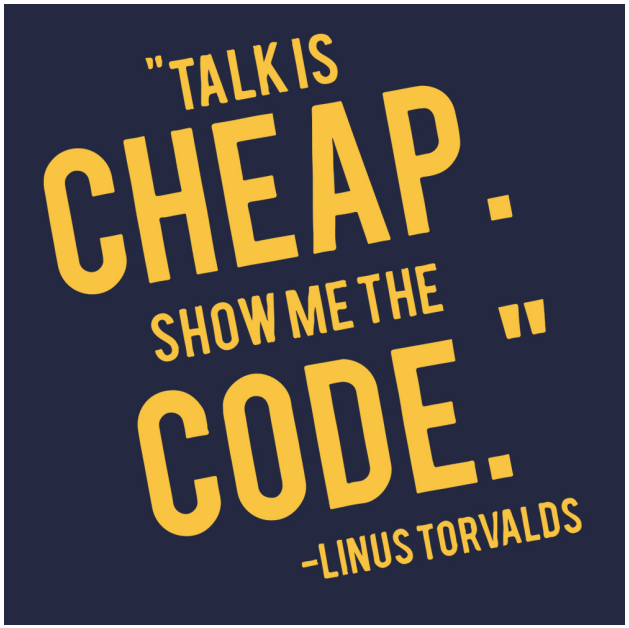
hfwei@nju.edu.cn

2022 年 11 月 21 日

Cymbol.g4



函数调用图



迷你计算器

IfStat.g4

```
stat : 'if' expr 'then' stat
     | 'if' expr 'then' stat 'else' stat
     | expr
     ;
```

二义性文法

IfStat.g4

二义性文法

```
stat : 'if' expr 'then' stat
     | 'if' expr 'then' stat 'else' stat
     | expr
     ;
```

```
stat : matched_stat | open_stat ;

matched_stat : 'if' expr 'then' matched_stat 'else' matched_stat
             | expr
             ;

open_stat: 'if' expr 'then' stat
         | 'if' expr 'then' matched_stat 'else' open_stat
         ;
```

Left-Factoring

```
stat : 'if' expr 'then' stat
     | 'if' expr 'then' stat 'else' stat
     | expr
     ;
```

```
stat : 'if' expr 'then' stat stat_prime ;
stat_prime : 'else' stat
           |
           ;

expr : ID ;
```

很明显, 提取左公因子无助于消除文法二义性

而且, ANTLR 4 可以处理有左公因子的文法

Expr.g4

二义性文法

```
expr :
    | expr '*' expr
    | expr '-' expr
    | DIGIT
    ;
```

结合性、优先级

Expr.g4

二义性文法

```
expr :
    | expr '*' expr
    | expr '-' expr
    | DIGIT
    ;
```

结合性、优先级 (−, ^)

```
expr :
    | expr '*' expr
    | expr '-' expr
    | DIGIT
    ;
```

ANTLR 4 可以处理
(直接) 左递归

```
expr : expr '-' term
     | term
     ;

term : term '*' factor
     | factor
     ;

factor : DIGIT ;
```

左递归 (左结合)

```
expr :
     | expr '*' expr
     | expr '-' expr
     | DIGIT
     ;
```

ANTLR 4 可以处理
(直接) 左递归

```
expr : expr '-' term
     | term
     ;

term : term '*' factor
     | factor
     ;

factor : DIGIT ;
```

左递归 (左结合)

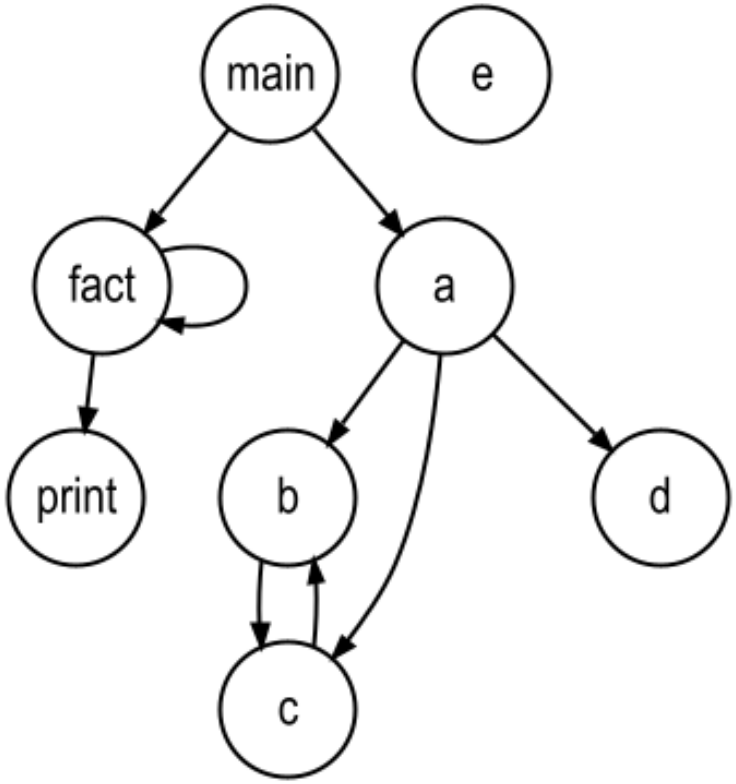
```
expr :
     | expr '*' expr
     | expr '-' expr
     | DIGIT
     ;
```

ANTLR 4 可以处理
(直接) 左递归

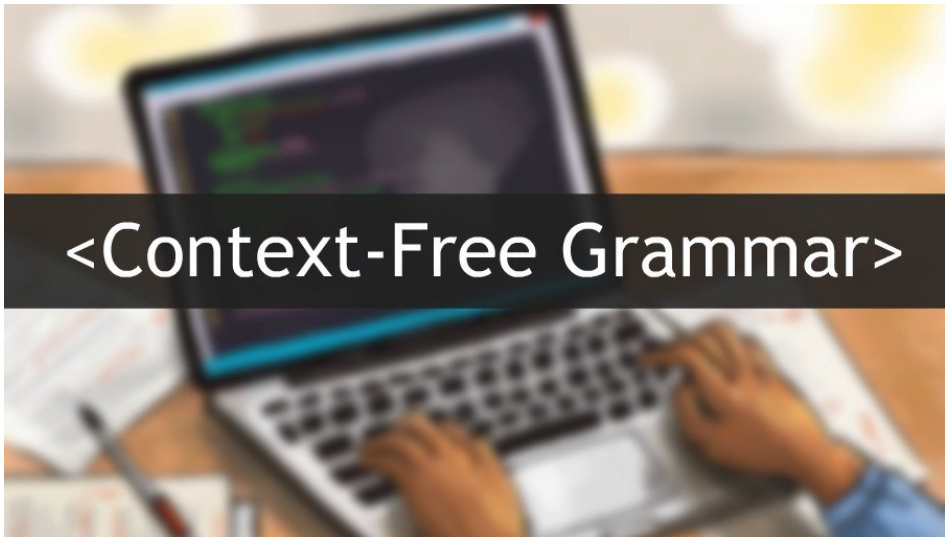
```
expr : term expr_prime ;
expr_prime : '-' term expr_prime
     |
     ;

term : factor term_prime ;
term_prime : '*' factor term_prime
     |
     ;

factor : DIGIT ;
```

右递归 (右结合)

Call Graphs



Calculator



上下文无关文法

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 $G = (T, N, P, S)$:

- 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;
- P 是**产生式** (Production) 集合;

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

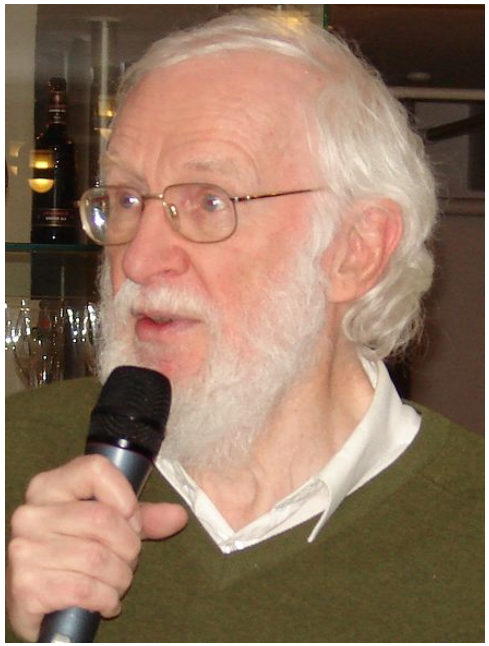
头部/左部 (Head) A: **单个**非终结符

体部/右部 (Body) α: 终结符与非终结符构成的串, 也可以是空串 ϵ

- S 为**开始** (Start) 符号。要求 $S \in N$ 且唯一。

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

John Backus
(1924 ∼ 2007)



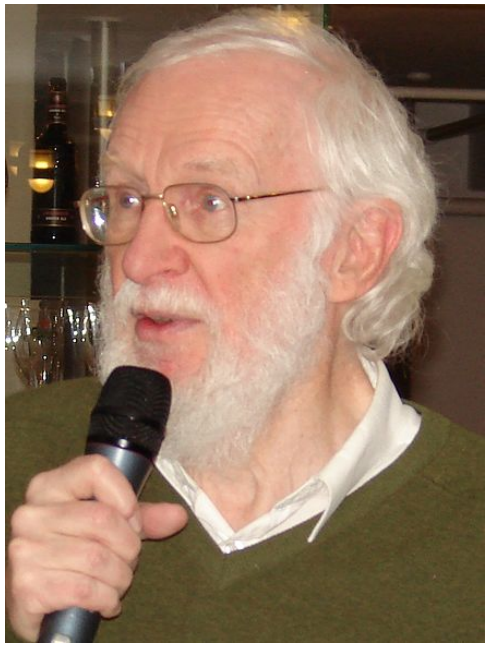
Peter Naur
(1928 ∼ 2016)

Niklaus Wirth (1934 ∼)

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

John Backus
(1924 ∼ 2007)

1977 (FORTRAN)



Peter Naur
(1928 ∼ 2016)

2005 (ALGOL60)

Niklaus Wirth (1934 ∼)

1984 (PLs; PASCAL)

语义: 上下文无关文法 G 定义了一个**语言** $L(G)$

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)

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

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)

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

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

推导 (Derivation)

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

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

$E \implies -E$: 经过一步推导得出

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

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

推导 (Derivation)

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

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

$E \implies -E$: 经过一步推导得出

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

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

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

Definition (Sentential Form; 句型)

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

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

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

Definition (Sentential Form; 句型)

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

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

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

Definition (Sentence; 句子)

如果 $S \overset{*}{\Rightarrow} w$, 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。

$$w \in L(G) \iff S \overset{*}{\Rightarrow} w$$

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

$$S \to SS$$

$$S \to (S)$$

$$S \to ()$$

$$S \to \epsilon$$

$L(G) =$

$$S \to SS$$

$$S \to (S)$$

$$S \to ()$$

$$S \to \epsilon$$

$L(G) = \{$良匹配括号串$\}$

$$S \to SS$$

$$S \to (S)$$

$$S \to ()$$

$$S \to \epsilon$$

$L(G) = \{$良匹配括号串$\}$

$$S \to aSb$$

$$S \to \epsilon$$

$L(G) =$

$$S \to SS$$

$$S \to (S)$$

$$S \to ()$$

$$S \to \epsilon$$

$L(G) = \{\text{良匹配括号串}\}$

$$S \to aSb$$

$$S \to \epsilon$$

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

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

$$S \to aSa$$

$$S \to bSb$$

$$S \to a$$

$$S \to b$$

$$S \to \epsilon$$

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

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

$$S \to bSbb \mid A$$

$$A \to aA \mid \epsilon$$

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

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

$$\boxed{V \to aVbV \mid bVaV \mid \epsilon}$$

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

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

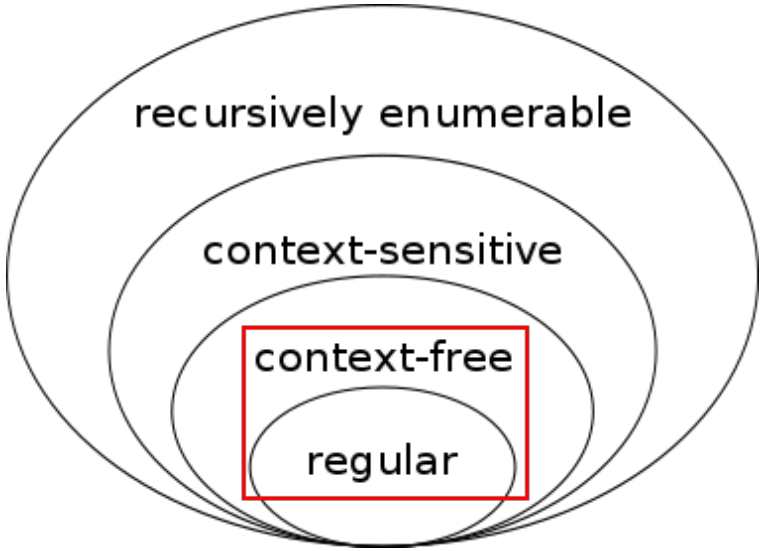
$$\begin{aligned}
S &\to T \mid U \\
T &\to VaT \mid VaV \\
U &\to VbU \mid VbV \\
V &\to aVbV \mid bVaV \mid \epsilon
\end{aligned}$$

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

$$\begin{aligned} S &\to T \mid U \\ T &\to VaT \mid VaV \\ U &\to VbU \mid VbV \\ V &\to aVbV \mid bVaV \mid \epsilon \end{aligned}$$

Let me know if you have a proof.

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?



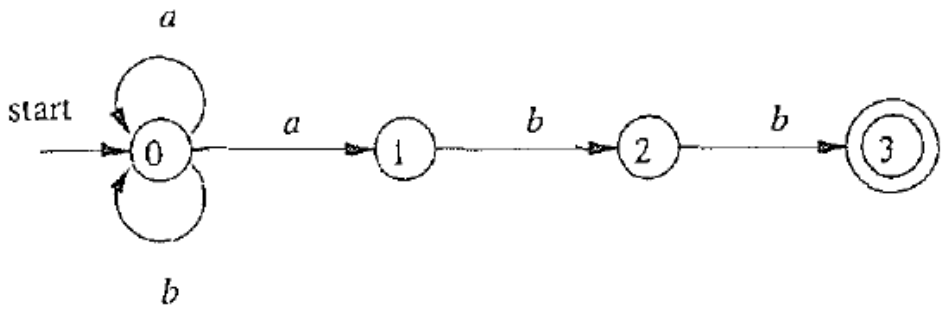
正则表达式的表达能力**严格弱于**上下文无关文法

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

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

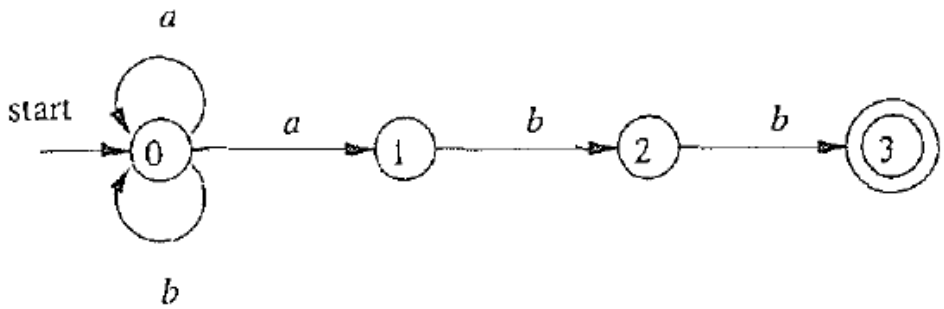
每个正则表达式 r 对应的语言 $L(r)$ 都可以使用上下文无关文法来描述

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



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

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



$$A_0 \to aA_0 \mid bA_0 \mid aA_1$$

$$A_1 \to bA_2$$

$$A_2 \to bA_3$$

$$A_3 \to \epsilon$$

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

$$S \to aSb$$

$$S \to \epsilon$$

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

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: $L(r) = L$

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: $L(r) = L$

则存在**有限**状态自动机 $D(r)$: $L(D(r)) = L$; 设其状态数为 k

Theorem

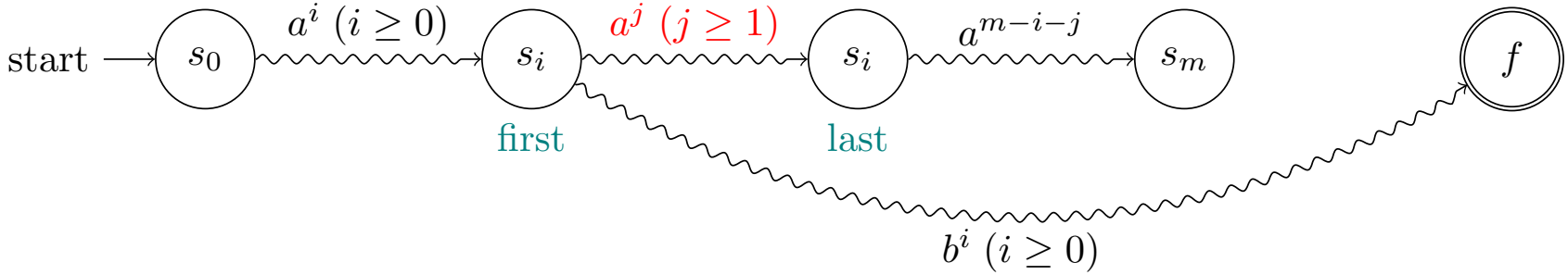
$L = \{a^n b^n \mid n \geq 0\}$ 无法使用正则表达式描述。

反证法

假设存在正则表达式 r: $L(r) = L$

则存在**有限**状态自动机 $D(r)$: $L(D(r)) = L$; 设其状态数为 k

考虑输入 $a^m (m > k)$



Theorem

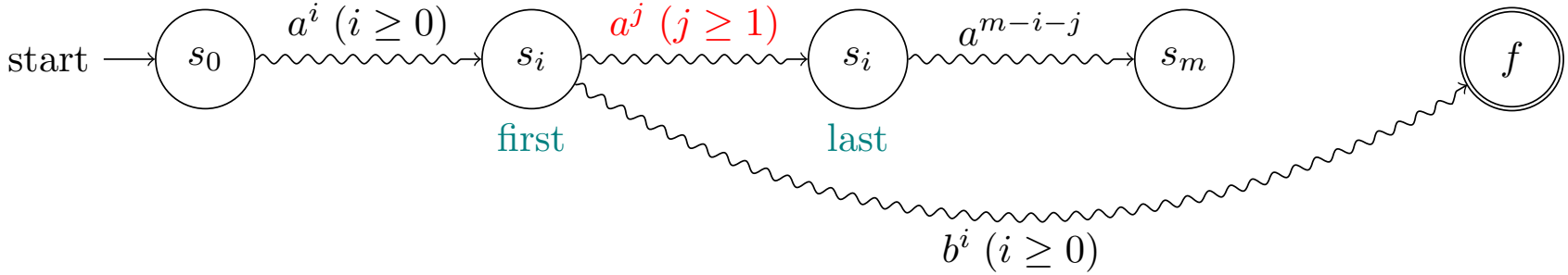
$L = \{a^n b^n \mid n \geq 0\}$ 无法使用正则表达式描述。

反证法

假设存在正则表达式 r: $L(r) = L$

则存在**有限**状态自动机 $D(r)$: $L(D(r)) = L$; 设其状态数为 k

考虑输入 $a^m (m > k)$



$D(r)$ 也能接受 $a^{i+j}b^i$; **矛盾!**

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

Pumping Lemma for Regular Languages

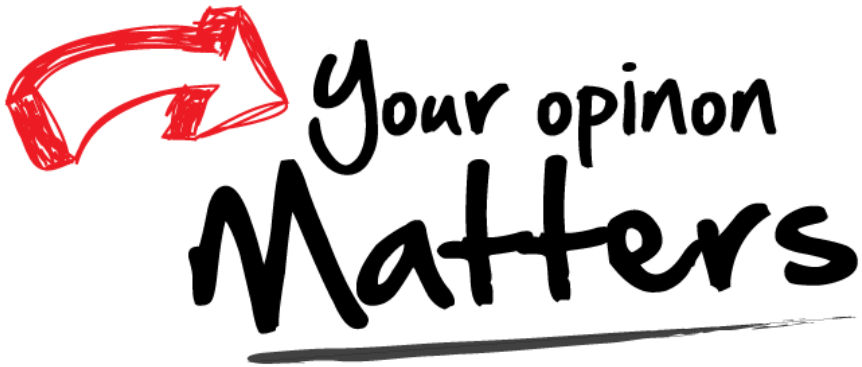
$$L = \{a^n b^n \mid n \geq 0\}$$

Pumping Lemma for Regular Languages

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

Pumping Lemma for Context-free Languages


Thank
You!



Office 926

hfwei@nju.edu.cn