中国科学技术大学
University of Science and Technology of China

词法分析 III

《编译原理和技术(H)》、《编译原理(H)》

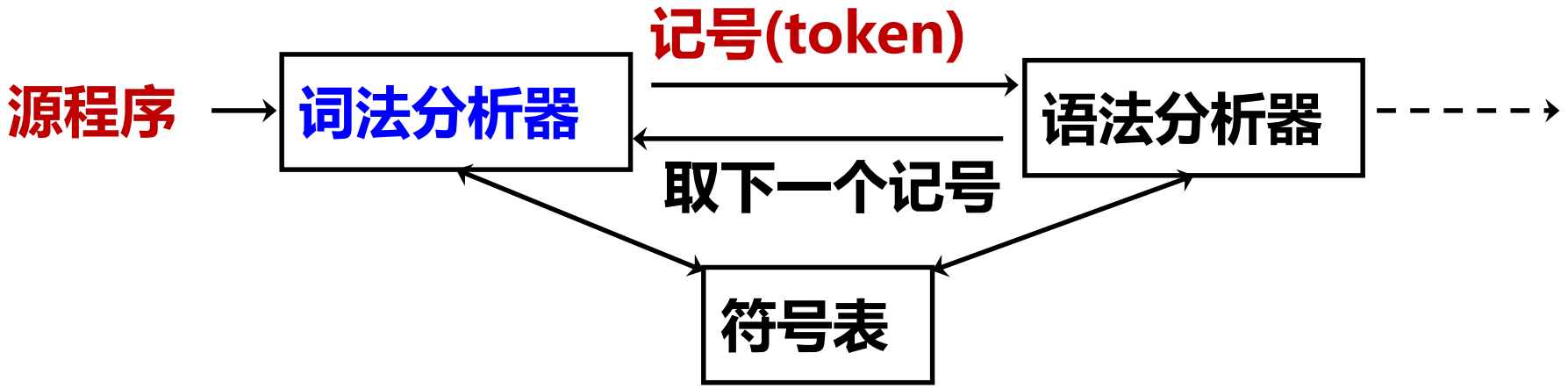
张昱

0551-63603804，yuzhang@ustc.edu.cn

中国科学技术大学

计算机科学与技术学院

本章内容



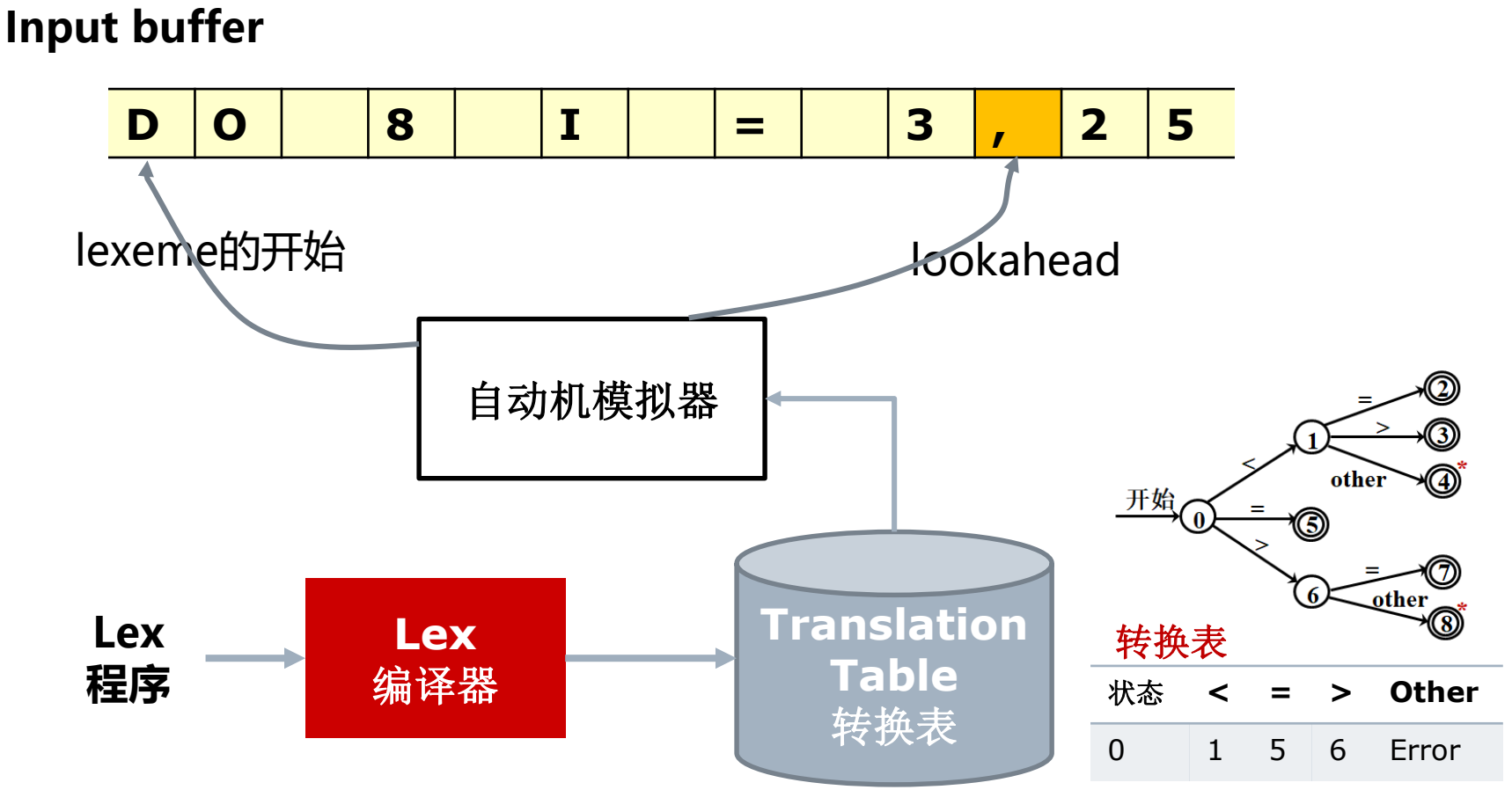
- 词法分析及要解决的问题
  - 向前看(Lookahead)、歧义(Ambiguities)
- 词法分析器的自动生成
  - 词法的描述：正规式；词法记号的识别：转换图
  - 有限自动机：NFA、DFA

中国科学技术大学
University of Science and Technology of China

2.5 词法分析器的生成器

- Lex: flex、jflex、antlr

词法分析器的生成器



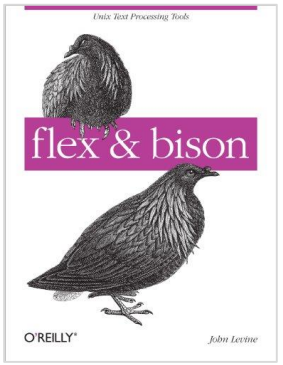
状态	<	=	>	Other
0	1	5	6	Error
1	4	2	3	4
2				

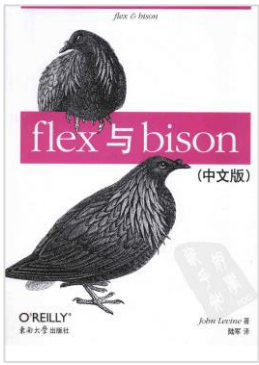
用Lex建立词法分析器

词法分析器 — Lexical analyzer, scanner

生成器 — generator

Flex https://github.com/westes/flex ; **JFlex** http://jflex.de/



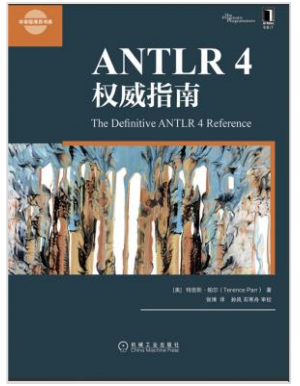


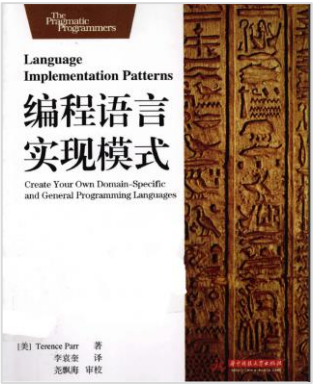
ANTLR https://www.antlr.org/ **v4.8 Jan 16, 2020** v4.9.2 Mar 11, 2021

v4.11.1 Sept 4, 2022

v4.13.2 Aug. 2024

http://lab.antlr.org/





Terence Parr is a tech lead at Google and until 2022 was a professor of data science / computer science at **Univ. of San Francisco**. He is the maniac behind ANTLR and has been working on language tools since 1989.

Check out Terence impersonating a machine learning droid: explained.ai

- Java
- C# (and an alternate C# target)
- Python (2 and 3)
- JavaScript
- TypeScript
- Go
- C++
- Swift
- PHP
- DART

用Lex建立词法分析器

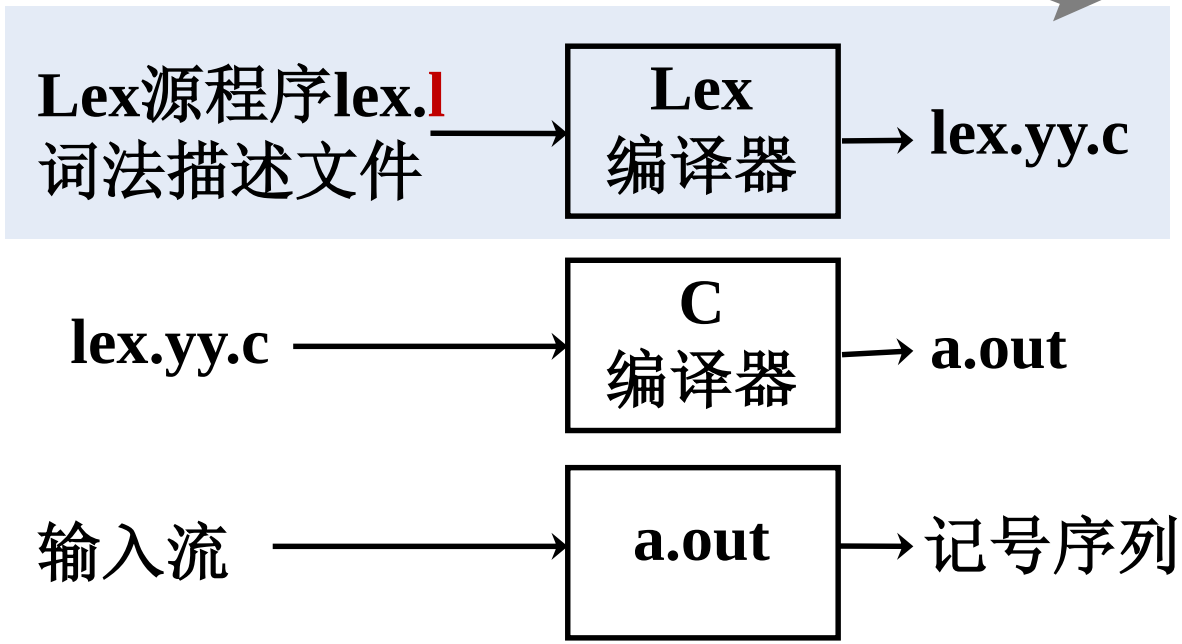
词法分析器 — Lexical analyzer, Scanner

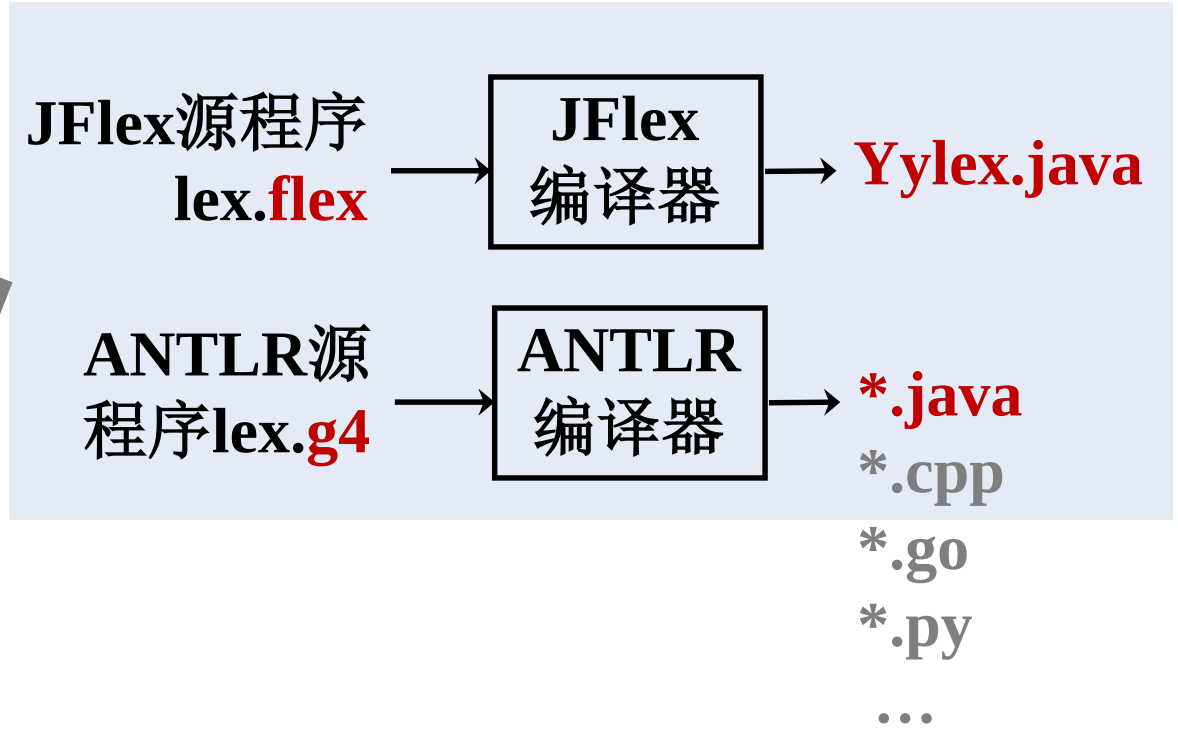
生成器 — Generator

Flex https://github.com/westes/flex

JFlex http://jflex.de/

ANTLR https://www.antlr.org/





Lex程序

- Lex程序包括三个部分

  声明

  %%

  翻译规则

  %%

  辅助过程

- Lex程序的翻译规则

  （p_i是模式）

  p_1 {动作1}

  p_2 {动作2}

  ……

  p_n {动作n}

```
%{
/* tokens.h包含常量LT, LE, EQ, NE, GT, GE, WHILE, DO, ID,
      NUMBER, RELOP的定义，它们用C的#define方式来写。*/
#include "tokens.h"
int addSymbol();
int addNumber();
%}
/* 正规定义 */
delim       [ \t \n ]
ws          {delim}+
letter      [A –Za – z]
digit       [0–9]
id          {letter}({letter}|{digit})*
number      {digit}+(\.{digit}+)?(E[+\–]?{digit}+)?
```

%{和%}包含的代码将直接复制到生成的分析器源程序文件中

对名字delim的引用

\.是转义序列，表示点本身；因为点是正规式中的元符号，表示匹配换行符以外的任意单一字符

Lex程序

- Lex程序包括三个部分

声明

%%

翻译规则

%%

辅助过程

5个字母连接成的正规式

对正规定义中名字ws的引用

```
{ws}        {/* 没有动作，也不返回 */}
while       {return (WHILE);}
do          {return (DO);}
{id}        {yylval = installId ( ); return (ID);}
{number}    {yylval = installNum( );
              return (NUMBER);}
" < "       {yylval = LT; return (RELOP);}
" <= "      {yylval = LE; return (RELOP);}
" = "       {yylval = EQ; return (RELOP);}
" <> "      {yylval = NE; return (RELOP);}
" > "       {yylval = GT; return (RELOP);}
" >= "      {yylval = GE; return (RELOP);}
```

- Lex程序的翻译规则（p_i是模式）

p_1 {动作1}

p_2 {动作2}

……

p_n {动作n}

- yylval:定义在lex.yy.c中，它保存所返回记号的属性值
- yytext:指向当前识别出的词法单元的开始字符
- yyleng:词法单元的长度

Lex程序

- Lex程序包括三个部分

声明

%%

翻译规则

%%

辅助过程

- Lex程序的翻译规则

（p_i是模式）

p_1 {动作1}

p_2 {动作2}

……

p_n {动作n}

```
installId ( ) {
    /* 把词法单元装入符号表并返回指针。
    yytext指向该词法单元的第一个字符，
    yyleng给出的它的长度                */
    return addSymbol(yytext, yyleng);
}

installNum ( ) {
    /* 类似上面的过程，但词法单元不是标识符而是数 */
    return addNumber(atof(yytext));
}
```

利用Flex生成词法分析器

- **关联的实验资源：** https://www.educoder.net/shixuns/k7p6t9sb/challenges

- **生成词法分析源码，xxx.l是词法描述文件**

  flex -i -I xxx.l

- **编译得到词法分析器可执行文件，-ll表示链接lex运行时库libl.a**

  gcc –g lex.yy.c –o xxx –ll

如果遇到找不到libl.a，可以将-ll改为-lfl，它代表去链接libfl.a 或者libfl.so
现代flex为-lfl

ANTLR 的文法文件 .g4的格式

- 格式

```
grammar MyG;
options { … }
import XXX;
tokens { … }
@actionName { … }
ruleName : <stuff> ;
……
```

- 纯词法

```
lexer grammar MyG;
```

- ruleName
  - 词法：大写字母开头
  - 语法：小写字母开头
- 词法规则

```
INT : DIGIT+ ;
fragment DIGIT : [0-9] ;
LQUOTE : '"' -> more, mode(STR) ;
mode STR;
STRING : '"' -> mode(DEFAULT_MODE);
TEXT: . -> more
```

匹配此规则但继续寻找记号，接下来匹配的记号规则包含本规则匹配的文本

正规定义，DIGIT不是记号

模式定义，词法状态

点通配任一字符

模式调用

利用ANTLR生成词法分析器

exprLexer.g4

```
lexer grammar exprLexer;

tokens {
    LeftParen,
    RightParen,
    Plus,
    Minus,
    Multiply,
    Divide,
    IntConst,
}
LeftParen: '(';
RightParen: ')';

Plus: '+' ;
Minus: '-' ;
Multiply: '*' ;
Divide: '/' ;
IntConst : [0-9]+ ;
ID: [a-zA-Z_][a-zA-Z_0-9]* ;
WS: [ \t\r\n]+ -> skip;
```

```
$ antlr4 exprLexer.g4
$ javac *.java
$ grun exprLexer tokens -tokens <expr 程序 >
```

exprLexer.java — 词法分析器的类定义

exprLexer.tokens — 词法记号名及其数值

exprLexer.interp — 包含允许运行内建解释器的数据，用于支撑IDEs对文法的调试

使用说明：https://github.com/antlr/antlr4/blob/master/doc/index.md

支持ANTLR4文法的VS Code扩展：
https://marketplace.visualstudio.com/items?itemName=mike-lischke.vscode-antlr4

ANTLR：Lexer规则中的命令

命令格式

TokenName : 选项1|…|选项N [-> 命令名 [(参数)]];

命令

- skip：不返回记号给parser，如识别出空白符或注释
- more：取另一个记号但不抛出当前的文本
- type(T)：设置当前记号的类型
- channel(C)：设置当前记号的通道，缺省为Token.DEFAULT_CHANNEL(值为0)；Token.HIDDEN_CHANNEL(值为1)
- mode(M)：匹配当前记号后，切换到模式M
- pushMode(M)：与mode(M)类似，但将当前模式入栈
- popMode：从模式栈弹出模式，使之成为当前模式

本章要点

- 词法分析器的作用和接口，用高级语言编写词法分析器等
- 掌握下面的相关概念，它们之间转换的技巧、方法或算法
  - 非形式描述的语言 ↔ 正规式
  - 正规式 → NFA
  - 非形式描述的语言 ↔ NFA
  - NFA → DFA
  - DFA → 最简DFA
  - 非形式描述的语言 ↔ DFA（或最简DFA）