词法器生成工具 FLEX

WANG Hanfei

School of Computer
Wuhan University

September 15, 2015

flex 简介

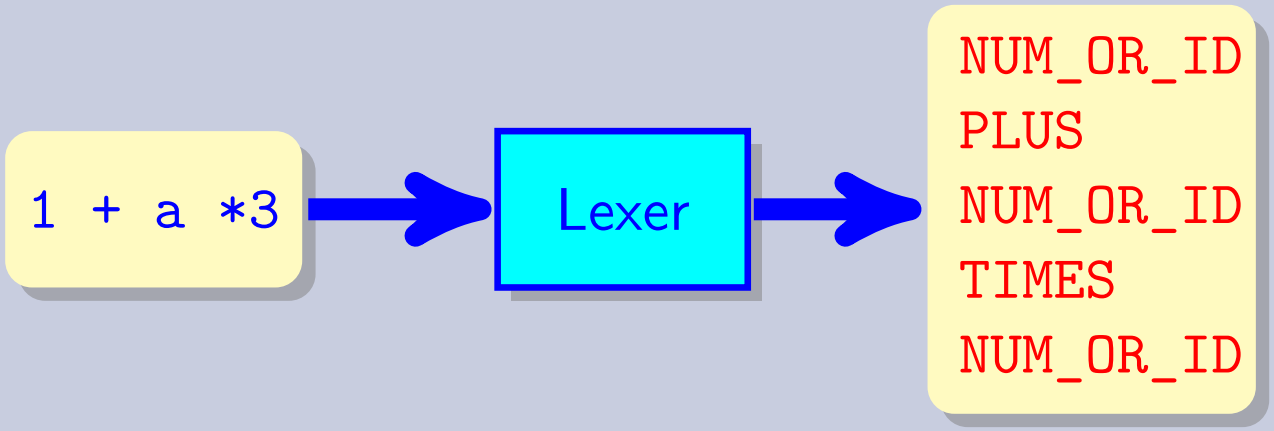
1. Fast LEXical analyser generator;
2. flex is fully compliant with the POSIX lex specification with little exception;
3. 开源, BSD Licence, Vern Paxson 维护, Last version 2.5.38 (2014, 02, 12): http://sourceforge.net/projects/flex;
4. Linux distribution standard utility;
5. Windows 版随 gnuwin32 包发行: http://gnuwin32.sourceforge.net/packages/flex.htm;
6. DOS 版：见 My CD-ROM;
7. 类似工具: JLex (for Java) (http://www.cs.princeton.edu/~appel/modern/java/JLex/).

基本概念

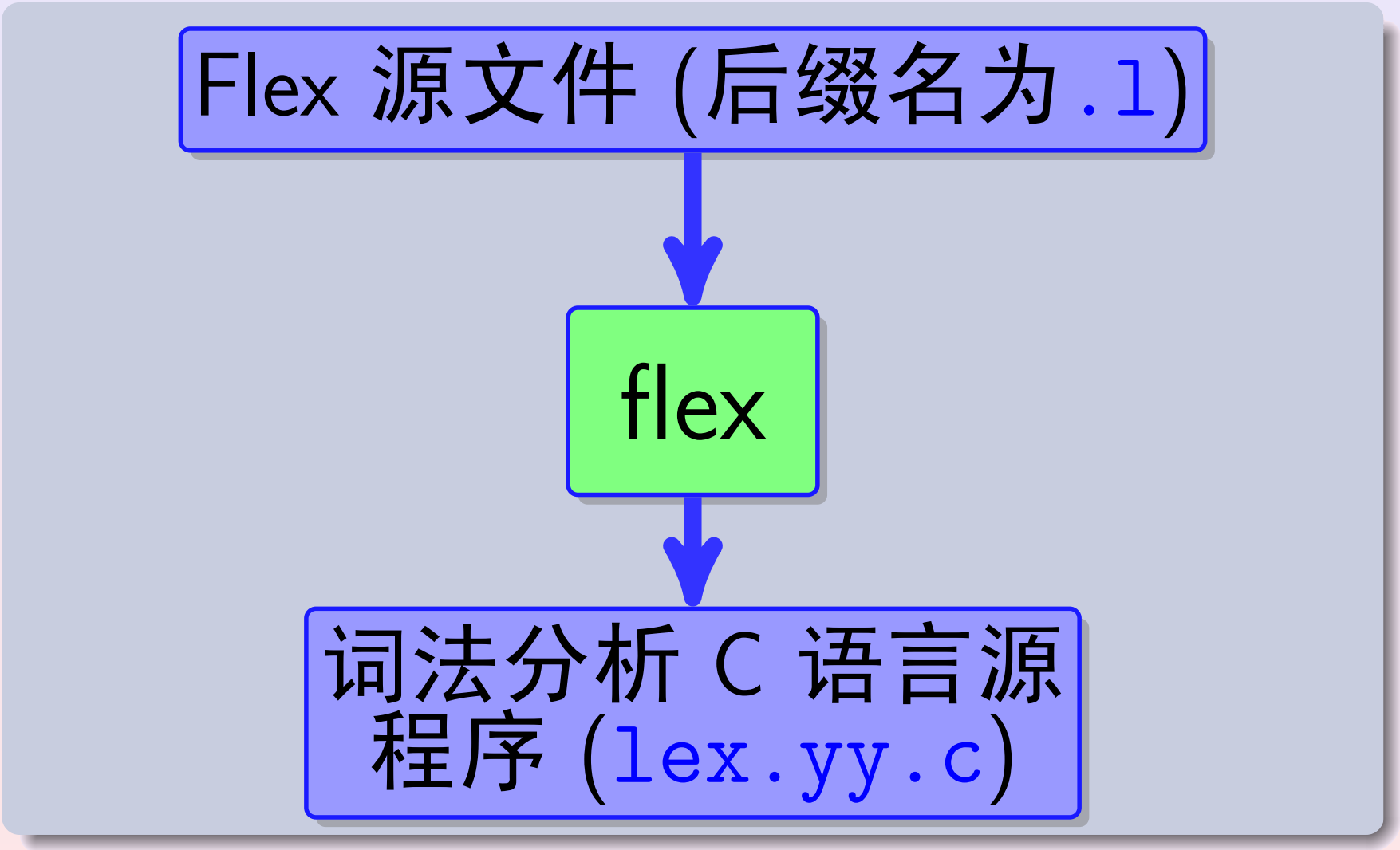
1. 单词(Token): Σ^*上的子集合，对字符串的分类，如: `identifier`;
2. 模式(Pattern): Σ^*上的子集合的描述，正规表达式 (RE), 如：`identifier`的模式为：`[_a-zA-Z][_a-zA-Z0-9]*`;
3. 词形(Lexeme): 单词集合上的一个具体元素，如: `my_variable` $\in$ `identifier`.

词法分析

将输入文件重组为 token 序列，如：XL 语言：



Flex



Example: Word Counter 1/2

```
$ cat count.l
  int num_lines = 0, num_chars = 0;
%%
\n        ++num_lines; ++num_chars;
.         ++num_chars;
%%
main()
{
  yylex();
  printf( "# of lines = %d, # of chars = %d\n",
    num_lines, num_chars );
}

int yywrap()
{
  return 1;
}
```

Example: Word Counter 2/2

```
$ flex count.l
$ ls -l
-rw-r--r--   1 hfwang   teacher       221 Mar 13 11:41 count.l
-rw-r--r--   1 hfwang   teacher     35769 Mar 13 11:46 lex.yy.c
$ cat lex.yy.c
/* A lexical scanner generated by flex */
/* Scanner skeleton version:
 * $Header: /home/daffy/u0/vern/flex/RCS/flex.skl,
   v 2.91 96/09/10 16:58:48 vern Exp $
 */

#define FLEX_SCANNER
#define YY_FLEX_MAJOR_VERSION 2
......
$ gcc lex.yy.c
$ ./a.out < count.l
# of lines = 16, # of chars = 221
```

Input File Format

```
DEFINITION PART

%%

RULE PART

%%

USER C Source Code
```

格式要求

- 顶行(Unindent, 无缩进):

  XXXXXX␣␣

- 非顶行(Indent, 有缩进):

  ␣␣XXXXXX

Patterns (Regular Expression) 1/4

`x` Match the character 'x';

`.` any character (byte) except newline;

`[xyz]` a "character class" ; in this case, the pattern matches either an 'x', a 'y', or a 'z';

`[abj-oZ]` a "character class" with a range in it; matches an 'a', a 'b', any letter from 'j' through 'o', or a 'Z';

`[^A-Z]` a "negated character class" , i.e., any character but those in the class. In this case, any character EXCEPT an uppercase letter;

`[^A-Z\n]` any character EXCEPT an uppercase letter or a newline;

Patterns (Regular Expression) 2/4

*r** zero or more *r*'s, where *r* is any regular expression;
r+ one or more *r*'s, where *r* is any regular expression (=*rr**);
r? zero or one *r*'s (that is, "an optional *r*") (=("" | *r*));
`r{2,5}` anywhere from two to five *r*'s;
`r{2,}` two or more *r*'s;
`r{4}` exactly four *r*'s ;
`{NAME}` the expansion of the "*NAME*" definition (see below);
`"[xyz]\"foo"` the literal string: '`[xyz]"foo`';
`\X` if `X` is an 'a', 'b', 'f', 'n', 'r', 't', or 'v', then the ANSI-C interpretation of `\X`. Otherwise, a literal 'X' (used to escape operators such as '*');

Patterns (Regular Expression) 3/4

\123 the character with octal value 123;
\x2a the character with hexadecimal value 2a;
(*r*) match an *r*; parentheses are used to override precedence (see below);
rs the regular expression *r* followed by the regular expression *s*; called "concatenation";
r|*s* eihter an *r* or *s*;
r/*s* an *r* but only if it is followed by an *s*. The text matched by *s* is included when determining whether this rule is the "longest match", but is then returned to the input before the action is executed. So the action only sees the text matched by *r*. This type of pattern is called "trailing context".

Patterns (Regular Expression) 4/4

^*r* an *r*, but only at the beginning of a line (i.e., which just starting to scan, or right after a newline has been scanned);

r$ an *r*, but only at the end of a line (i.e., just before a newline). Equivalent to "*r*/\n" ;

<*S*>*r* an *r*, but only in start condition *S* (see below for discussion of start conditions);

<<EOF>> the end-of-file.

注意事项

- 运算优先级别按上述列表的先后由高到低排列，如：
  `foo|bar* = ((foo)|(ba(r*)))`
- 空格只能出现在`" "`和`[ ]`中：
  `an error` flex 将之分析为`an`为 pattern，而`error`是 action，这样在编译输出文件是出错;
- `" "`和`[ ]`支持转义字符, 如：`"\\x"` will match `\x`;
- 支持汉字：
  `[\x81-\xfe][\x40-\xfe]`匹配一个汉字 GBK 码。
  `[\xb0-\xf7][\xa0-\xfe]`匹配一个汉字 GB 码。
  `"土豪"`：如果本 flex 源程序是用的 GBK 码，则该模式匹配 GBK 码汉字“土豪”...

Definition Part

```
%{
C语言说明部分（全局变量、包含文件、宏定义和引用说明）
%}

␣␣C语句

  /* pattern definition */
NAME pattern

  /* inclusive start condition list */
%s  S1 S2
  /* exclusive start condition list */
%x  X1 X2
```

说明

- 定义部分的%{}和非顶行的开始直到该行结束所有的文字将直接拷贝到输出文件 (去掉%{}), flex不进行任何处理;
- %{}一定要顶行书写;
- Pattern Definition 所定义的*NAME*, 在 Rule Part 的 Pattern 中可以用{*NAME*}引用，flex将其替换为对应的正规表达式，如：

```
ID [_a-zA-Z][_0-9a-zA-Z]*
```

在 Rule Part 中可以：

```
{ID}    printf ("id: %s\n", yytext);
```

等价于

```
[_a-zA-Z][_0-9a-zA-Z]*    printf ("id: %s\n", yytext);
```

- flex不识别任何 C 语句，C 代码只能出现在其格式规定的位置，但是为了能正确地将 C 语句拷贝到输出文件指定的地方，flex能识别 C 的{ }块结构, 注释和字符串常量。

Rule Part

pattern1 *action1*

.....

patternN *actionN*

- *patterN*是正规表达式;
- *patterN*一定要顶行书写;
- *actionN*是 C 语句;

Actions

```
program    printf("keyword\n"); /* one line */

{ID}       {
              printf("Identifier\n");
              /* multiple lines */
           }
```

注意

- action中出现的{}一定要平衡，如果少了一个}，flex 将把以下的输入都认为是Action的代码，由于直到文件结束都找不到结束action的}而报错如下：EOF encountered inside an action。
- action可以为空或“;”，表示对所识别的词形不做任何处理，即过滤当前识别的串。

Default rule

- Simply echo unmatched text to output, ex: input file just two %%:

```
%%

%%
```

this will copy input to output — a copy program.

Union of Rule

Example

```
program     |
procedure     printf("keyword\n");
```

$\Longleftrightarrow$

```
program|procedure  {
          printf("keyword\n");
        }
```

The generated scanner

- flex的输出lex.yy.c包含了一个原型为: int yylex()的词法分析函数、DFA 状态转移矩阵和一些辅助函数和宏定义;
- yylex()对yyin(缺省为stdin) 扫描, 匹配某一模式后执行该模式对应的 C 语言action, 如果action中没有return 等结束函数的语句, yylex()继续上述操作, 否则将返回调用它的函数; yylex()由于是用全局变量保存分析现场，因此下一次再调用它时，它将继续对输入文件中还没有扫描的部分继续分析;
- 用户可通单词编码 (TOKEN_NUM)，在单词 pattern 的action中加上 “return TOKEN_NUM;” 返回调用函数，调用函数通过yylex()的返回值知道当前识别的单词, 词法分析与语法分析就是通过该方法实现互动机制。

How input matched

- 最长匹配原则;
- 最先匹配原则。

Example

```
program     printf("keyword\n");
{ID}        printf("identifier\n");
prog        printf("prog\n");
```

如果输入是："programming program" 则经过flex生成的扫描程序扫描后将输出："identifier keyword"，由于模式{ID}包含模式prog，并且在后者之前，因此将模式prog永远不会匹配，用flex编译该文件时，将提示警告："warning, rule cannot be matched".

Inside lex.yy.c 1/4

状态转换矩阵 缺省时用压缩方式表示，加-f选项时直接用二维数组表示，这样输出的文件体积较大，但是执行速度快；

File * yyin; yylex()所扫描的文件，缺省值为stdin;

File * yyout; yylex()扫描输出文件，缺省值为stdout;

int * yyleng; yylex()当前识别的词形的长度;

char * yytext; yylex()当前识别的词形;

ECHO; (宏) 打印当前识别的词形到yyout;

REJECT; (宏) 选择下一个最佳 pattern, ex:

```
a       |
ab      |
abc     ECHO; REJECT;
```

如果输入是: abc, 则输出为：abcaba. 注意：REJECT之后的代码将不执行;

Inside lex.yy.c 2/4

`yymore();` 模式对应的action完成后，yytext不清空，将下一个模式匹配后的词形直接追加到yytext中，如：

```
mega-      ECHO; yymore();
kludge     ECHO;
```

如果输入是: mega-kludge, 则输出为：mega-mega-kludge.

`yyless(n);` 模式对应的action完成后，将缓冲区当前指针定位到当前词形的第n个字符之后的第一字符，下次扫描从该字符开始，而不是当前词形后的第一字符，如：

```
foobar     ECHO; yyless(3);
[a-z]+     ECHO;
```

如果输入是: foobar, 则输出为：foobarbar.

Inside lex.yy.c 3/4

unput(c); 回退字符c到输入流, 即追加c到缓冲区当前扫描字符之前，设置下次扫描的字符为在缓冲区的c, unput()将破坏yytext, 如下段程序将为识别的词形加上()后重新扫描：

```
{
  int i;
  /* Copy yytext because unput()
     trashes yytext */
  char *yycopy = strdup( yytext );
  unput(')');
  for ( i = yyleng - 1; i >= 0; --i )
    /* in reverse order */
    unput( yycopy[i] );
  unput('(');
  free(yycopy);
}
```

Inside lex.yy.c 4/4

`int input();` 从缓冲区读取一个字符，并把当前扫描字符指针下移一个;

`BEGIN (S);` (宏) 激活条件模式, 其中*S*是定义部分的定义的 (`%s` or `%x`) 条件模式名, 在以后的扫描中，以`<S>`开始的模式将被激活; 注意: 用户代码中不要定义与`BEGIN`等系统已有的宏重名的标识符;

`int yywrap();` (用户提供) 当扫描程序`yylex()`读到`yyin`的`EOF`时, 将调用该函数, 如果该函数返回`0`, 扫描程序`yylex()`认为用户已经重新设置了`yyin`, 将继续扫描; 如果返回非零, 描程序`yylex()`将正常结束, 返回`0`到调用它的函数。用户可以在该函数中对`yyin`重新设置，使得`yylex()`不间断地扫描多个文件，对单个文件; 用户提供下述函数定义即可：`int yywrap()`
`{ return 1; }`

注意 不需要`int yywrap()`时，可在定义部分加选项`%option noyywrap`或`gcc`编译链接时加选项`-fl`(加载`int yywrap()`库函数).

Start Condition

flex提供有条件地激活一组规则及对应action的设施，称之为条件模式 (Start Condition):

- 在定义部分：“%s 模式标签列表”定义相容模式标签; “%x 模式标签列表”定义互斥模式标签;
- 在模式部分加上模式标签:
  < 模式标签 *1*, 模式标签 *2*,...>*pattern* *action*
- 没有标签的模式系统为之加上缺省标签INITIAL;
- 扫描程序开始工作时有效的模式有INITIAL标签的模式;
- 如果在一个action执行了 “BEGIN(模式标签 *x*);”，这时如果模式标签 *x*是相容模式标签，系统将激活所有有该标签的模式，同时INITIAL模式也是有效模式；如果是互斥模式标签，系统将激活仅有该模式标签标记的模式；这些被激活的模式将始终处于激活状态直到下个 “BEGIN();” 语句;
- 利用条件模式可针对有不同词法规则的一段文字定义一个mini-scanners.

Example: C Comments

```
/* scanner which discards C comments while
 * maintaining a count of the current input line.
 */

%x comment

%%
  int line_num = 1;

"/*"         BEGIN(comment);

<comment>[^*\n]*        /* eat anything that's not a '*' */
<comment>"*"+[^*/\n]*   /* eat up '*'s not followed by '/'s */
<comment>\n             ++line_num;
<comment>"*"+"/"        BEGIN(INITIAL);
```

命令选项

- `-b` 输出回溯(back up) 到`lex.backup`文件中;
- `-C` 对输出的状态转移矩阵进行不同强度的压缩, 强度的强弱次序为: `-Cem`(缺省), `-Cm`, `-C`, `-C{f,F}e`, `-C{f,F}`和`-C{f,F}a`;
- `-d` 在每个模式匹配后输出调试信息: "`--accepting rule at line nnn ("matched text")`" 到`stderr`;
- `-f` fast scanner, 状态转换矩阵没有压缩;
- `-i` instructs flex to generate a case-insensitive scanner;
- `-s` suppress default rule;
- `-t` 不生成`lex.yy.c`文件，将该文件直接打印到`stdout`;
- `-v` verbose mode, 输出生成的扫描程序 DFA 等状态信息;
- `-T` trace mode, 跟踪扫描程序的生成的每个过程。

效率相关

- flex的目标是生成高效的实用扫描程序，为了能处理大量的模式，其性能已全面优化，所生成的扫描程序具有很高的效率，但是有些操作和模式对最后的扫描程序的执行效率有一定的影响；
- -C选项压缩 DFA 的状态转换矩阵，影响访问速度；
- REJECT 将回退当前识别的词形到输入；
- %option yylineno 打开记录被扫描文件的行号；
- 模式中有一个模式是另一个模式的前缀，或一个模式是另一个的子串，使得扫描程序不得不回溯 (back up)；
- “trailing context” 和^r也导致回溯；
- yymore(), yyless(n)对效率也有较小的影响。

Example: Bad Scanner 1/2

```
$ cat bad.l
%%
[^~]+"~"   ; /* String end with `~', Very Large String */
.|\n       ; /* single character is substring of the above */
%%

main()
{
  yylex();
  return ;
}

int yywrap()
{  return 1; }
$ flex -b bad.l
$ gcc lex.yy.c
```

Example: Bad Scanner 2/2

```
$ cat lex.backup
State #6 is non-accepting -
 associated rule line numbers:
       2
 out-transitions: [ \000-\377 ]
 jam-transitions: EOF []

Compressed tables always back up.
$ ls -l lex.yy.c
-rw-r--r--    1 hfwang   teacher     35766 Mar 14 14:12 lex.yy.c
$ time ./a.out <lex.yy.c
real    1m32.984s
user    1m30.350s
sys     0m2.630s
```

For Further Reading

V. Paxson.
Flex, version 2.5: A fast scanner generator.
http://www.gnu.org/software/flex/manual/, 1998.

B.R. Levine, T. Mason, and D. Brown.
lex & yacc, Second Edition.
O'Reilly & Associates, Inc., 1992.

M.E. Lesk and E. Schmidt.
LEX - Lexical Analyzer Generator.
Comp. Sci. Tech. Rep. No. 39. Bell Laboratories, 1975.

本章小节