Parser Generator 词法分析原理研究

于思江[1]　王小兵[2]
（1.西安邮电大学财务处　西安　710121；2.西安电子科技大学计算机学院　西安　710071）

摘　要：针对 Windows 环境下的词法分析工具相关文献较少，不便排除词法错误的不足，对 Parser Generator 中 ALEX 的词法分析原理进行了研究。在分析 ALEX 词法源文件结构的基础上，研究了输出 verbose 文件的内部结构，给出了相应自动机的图示说明，并概述了生成的 C++文件的内容。对转换得到的 C++文件进行编译执行，结果分析表明，对于不同的字符串输入，ALEX 生成的词法分析器均能正确的进行处理。最后，提出使用 ALEX 编写词法分析器，分析其 verbose 文件及相应的自动机有利于用户纠正词法错误。

关键词：Parser Generator；ALEX；词法分析
中图分类号：TN06　TP314　　**文献标识码**：A　　**国家标准学科分类代码**：520.4040

Principles study of lexical analysis in Parser Generator

Yu Sijiang[1]　Wang Xiaobing[2]
（1. Financial Section, Xi'an University of Posts and Telecommunications, Xi'an 710121, China;
2. School of Computer Science, Xidian University, Xi'an 710071, China）

Abstract: In order to overcome the shortages which the lexical analysis tools in Windows have such as lack of related documents and inconvenient error correction, the principles of lexical analysis in ALEX of Parser Generator are introduced. The structure of a lexical file in ALEX is given, and the output verbose file is presented and illustrated with a related automaton, then the details of the output C++ files are explained. The transferred C++ files can be compiled and executed, then the analysis of results shows that the lexical analyzer produced by ALEX can deal with various input strings correctly. At last, it shows that ALEX can be used to create the lexical analyzer while the analysis of verbose file and its related automaton can help users to correct the lexical errors.

Keywords: Parser Generator; ALEX; lexical analysis

1　引　言

Lex 与 Yacc[1] 是构造语言解释器与编译器的自动化工具，在波形编辑软件设计[2]、程控命令分析[3]、视频序列行为识别[4]等领域也得到了广泛应用。Lex 在 20 世纪 70 年代由贝尔实验室开发完成：Stephen C. Johnson 首先开发了 Yacc，其后为了与 Yacc 配合工作，Mike Lesk 与 Eric Schmidt 开发了 Lex。目前，Lex 已经成为标准的 UNIX 实用程序。40 年来已经形成了针对不同高级语言、运行于多种操作系统的 Lex 工具，如 FLex、JLex 等。Lex 是标准的 Unix、Linux 辅助工具，无需安装就可以直接使用。在 Windows 环境中，支持 Lex 有 Parser Generator[5]、JavaCC[6] 等工具，它们在实际当中都得到了广泛的应用。Parser Generator 的一大特点是能够很好的与 Visual C++进行集成，适合于面向对象系统的开发，是实现计算机语言翻译的有力工具。ALEX 是 Parser Generator 提供的词法分析工具，在进行词法分析的过程中，不可避免的会出现各种词法错误[7]，排除错误的关键在于程序员对 ALEX 内部词法分析原理的掌握程度。到目前为止，已有文献讨论了 Lex[1]、Flex[8] 等工具的使用方法，却鲜有文献对 ALEX 进行详尽的研究。为了更好的使用 Parser Generator 提供的词法分析工具 ALEX，开发出高效的词法分析器，并且快速分析用户的词法错误，对 Parser Generator 中 ALEX 的词法分析原理进行全面系统的研究。

收稿日期：2014-11

2 Parser Generator 概述

Parser Generator 是一个集成了 Lex 与 Yacc 的 Windows 集成开发环境，如图 1 所示。其中包含的 Lex 与 Yacc 被称为 ALEX 与 AYacc，包含的库文件被用于生成词法分析器与语法分析器。经过多年的发展，Parser Generator 的功能与性能得到了逐步的完善与提高。

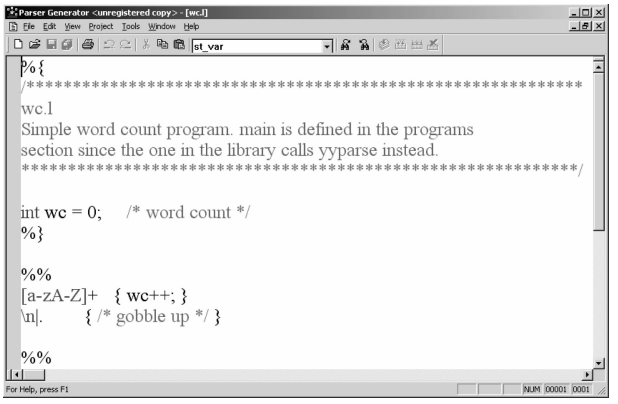

图 1 Parser Generator 界面

以 Parser Generator 2.0 为例，其主要特点包含：工程管理能够方便的创建工程、增加删除文件、设置生成选项；工程生成包含输出窗口，能够正确的将错误映射到源文件；ParserWizard 可以生成 ALEX 与 AYacc 文件框架，特别的，支持 C、C++与Java中的多个词法分析器与语法分析器，即一个程序中可以包含多个词法分析器与语法分析器；LibBuilder 可以生成 ALEX、AYacc 库文件，将来提供给词法分析器与语法分析器运行时使用；支持多字节字符集编码模式；编辑器支持 ALEX、AYacc、C/C++与 Java 的语法；参考实例包含详细的 ALEX 与 AYacc 语法参考手册，并且提供若干 ALEX 与 AYacc 源文件。

3 ALEX 及支持的正规式

Parser Generator 2.0 中 ALEX 的主要特点包含：能够生成 C、C++与 Java 3 种版本的词法分析器；生成的词法分析器支持 Unicode 与多字节字符集，对 Java 只支持 Unicode；生成的词法分析器分析速度快、并且分析表紧凑；C 版本的词法分析器支持单模型、多模型与多实例模型，即 C 版本的词法分析器可以支持单个词法规则模型，也可以支持多个词法规则模型，或者一个词法规则模型可以实例化出多个词法分析器；支持面向对象编程，C++或 Java 版本的词法分析器被实现为一个子类，用户根据需要可以添加属性与方法；在分析出现错误时，分析器能够执行相应的恢复动作，如当符号被弹出栈、记号被抛弃时可以执行自定义动作。

Parser Generator 2.0 中 ALEX/AYacc 的库文件包含了大部分的分析表驱动代码，用户根据需要可以调整优化；支持静态链接库文件与 DLL；通过优化的 IO 操作提高词法分析器与语法分析器的效率；直接支持 Microsoft Visual C++，Borland C++ Builder 与 Borland C++，使用 LibBuilder 可以生成其他编译器需要的库文件；支持多种语言、编译器与平台。

ALEX 中常用的正规表达式及其扩展的语法和语义主要包含：x 匹配字符 x，例如 a 匹配字符 a；x|y 匹配正规式 x 或 y，例如 ab|cd 匹配 ab、cd；xy 匹配正规式 xy，例如 abc 匹配 abc；x* 表示正规式 x 的闭包，例如 a* 匹配空、a、aa、aaa、…；(x)匹配正规式 x，括号用于改变运算优先级；x+表示正规式 x 的正闭包，例如 a+匹配 a、aa、aaa、…；x? 表示正规式 x 或空，例如 a? 匹配空、a；x{m}匹配 m 个正规式，例如 abc{2}匹配 abcabc；x{m,n}匹配 m 到 n 个正规式 abc{2, 3}匹配 abcabc、abcabcabc；. 匹配除换行符\n、以外任意单一字符，例如. 匹配 a、b、c 等；“…”：引号中的字符按照字面意义被解释，例如“.”匹配.；\x 匹配字符 x 自身，或转移序列，例如。匹配.，\t 匹配制表符，\n 匹配换行符，* 匹配 *；[xy]匹配方括号中的任意一个字符，例如[ab]匹配 a、b；[x—z]匹配 x 至 z 之间的任意一个字符，例如[0—9]匹配 0～9 一共 10 个数字，[a—z]匹配 a 至 z 一共 26 个字母，—在方括号中最左或最右表示其本身；[^x]匹配除 x 外的任意一个字符，例如[^\n\t]匹配除制表符、换行以外的任意一个字符；^x 匹配一行开始出的 x，例如^abc 匹配行首的 abc；x 匹配一行结束处的 x，例如 abc 与 abc\n 相同，均匹配行尾的 abc；x/y 匹配/前的正规式 x，要求其后是正规式 y，例如[1—9]+/KG 匹配 64KG 中的 64。

4 ALEX 词法源文件

4.1 源文件结构

利用 Parser Generator 中的 ALEX 设计词法分析器，重点需要了解 3 个方面的内容：ALEX 使用的正规式的语法与语义；使用 ALEX 正规式定义词法规则的方法；对于匹配的词法规则，ALEX 提供的语义动作支持机制。ALEX 源程序文件一般以.l 为后缀，下面以采用 C++语言的 mylexer.l 文件为例，经过分析研究得到其如下基本结构：

[声明部分：
1)%{ C++声明% }
2)词法分析类声明
3)辅助定义正规式
]
%%
规则部分：
词法规则正规式 { 语义动作(C++代码)}
[%%
子程序部分(C++代码)
]

ALEX 源文件由声明部分、规则部分与子程序部分 3

部分组成，其中声明部分与子程序部分可以省略。下面以mylexer.l文件为例说明ALEX源程序的结构。

```
01 %{
02 %}
03 %name mylexer
04 // class definition
05 {
06 // place any extra class members here
07 }
08 // constructor
09 {
10 // place any extra initialisation code here
11 }
12 // destructor
13 {
14 // place any extra cleanup code here
15 }
16 string_ab (a|b)*
17 %%
18 {string_ab}a {
19 cout<<"识别a结束的ab串：长度为"<<yyleng<<"的"<<yytext<<endl;
20     return 1;
21   }
22 ^01 {
23 cout<<"识别在行首的01!"<<endl;
24 return 2;
25   }
26 a/b {
27 cout<<"识别b前面的a!"<<endl;
28     return 3;
29 }
30 %%
31 int main(void)
32 {
33 int n = 1;
34 mylexer lexer;
35 if (lexer.yycreate()) {
36   n = lexer.yylex();
37 }
38 return n;
39 }
```

mylexer.l文件的第01至16行为声明部分，分为C++声明、词法分析类声明与辅助定义正规式：

1)C/C++声明：位于第01～02行，包围在%{与%}之间，使用C++语法，可以不包含任何源代码，也可以包括C++的预处理语句、类/类型、变量说明语句以及子程序说明等，此处说明的一切在ALEX源程序的其他部分均可被引用。

2)词法分析类声明：位于第03～15行，使用C++语法，声明了一个特殊的专用于词法分析的类。默认的%name mylexer说明词法分析类的名称为mylexer，该名称可以根据需要修改，随后的3个{}之间的内容分别声明该类的成员、建构函数与析构函数，可以不包含任何源代码。在子程序部分中，可以实例化出mylexer的一个对象，进而调用其中的方法实现词法分析。

3)辅助定义正规式：位于第16行，辅助正规式的名字为string_ab，定义为正规式(a|b)*，这样在ALEX源程序的规则部分使用{string_ab}代替(a|b)*。如果一条词法规则的正规式很长，可以将它分成若干部分，每一部分取一个简化的名字并定义为一个辅助正规式，通过组合这些简化的名字来定义原有的词法规则。类似的，如果若干条词法规则都出现相同的正规式，那么可以把它提取出来，定义一个辅助正规式，从而简化词法规则的描述。通俗地讲，辅助定义正规式的作用是为复杂的或重复出现的正规式命名，并在以后的使用中用名字代替该正规式。辅助定义的正规式仅供内部使用，只是词法规则的一部分，因此不能说明记号，不能用于识别一个完整的单词。

mylexer.l文件的第18～29行为规则部分，它由一组词法规则组成，每条规则的形式如下：

正规式 { 语义动作(C++代码) }

正规式中使用的辅助正规式需要使用{}括起来，当词法分析器使用该正规式匹配一个输入序列时，就执行其后{}中的语义动作。语义动作代码只有一行时，直接放在正规式后，否则加{}。

第1条词法规则的正规式为{string_ab}a，string_ab为辅助定义正规式，如果不加{}，则string_ab就表示本身。辅助正规式展开后，该条词法规则为(a|b)*a，表示识别a结束的ab串。当识别成功后，执行相应的语义动作，使用全局变量yyleng输出匹配成功的单词的长度，使用全局变量yytext输出匹配成功的单词的字面值，return 1表示匹配成功后词法分析器退出执行，1代表匹配成功的单词属于标记为1的记号。

第2条词法规则的正规式为^01，表示识别行首的01，语义动作中的return 2表示匹配成功的单词属于标记为2的记号。

第3条词法规则的正规式为a/b，表示识别b前面的a，语义动作中的return 3表示匹配成功的单词属于标记为3的记号。

mylexer.l文件的第31～39行为子程序部分，包含规则部分的语义动作设计的函数调用以及其他需要的程序定义，如果词法分析器单独运行，一般定义一个main函数。如果子程序部分很长，可以单独组织为一个源文件，在声明部分用 include语句将其包含进来。lexer是词法分析类mylexer的一个实例，在调用yycreate()初始化后，执行yylex()进行词法分析，返回结果即是前面词法规则

的语义动作返回记号的整数记号。

4.2 输出 verbose 文件结构

以上面的 mylexer.l 文件为例，Parser Generator 对其进行编译处理，会得到 mylexer.v、mylexer.h 与 mylexer.cpp 一共 3 个文件。

mylexer.v 是 ALEX 生成的 verbose 文件，描述了一个自动机，也是词法分析器的文本描述，用于分析词法分析器的具体动作，便于用户对错误进行定位。Verbose 文件分为 3 个部分：带编号的正规式，它们是从 ALEX 源程序中提取出来的词法规则，对于辅助定义正规式需要进行展开；自动机状态，由正规式词法规则根据算法转化得到，每个状态包含状态编号，开始条件，转移与匹配说明 4 个部分，其中开始条件与匹配说明可以为空；统计信息，包括开始状态个数、正规式个数、自动机状态数等。

mylexer.v 的具体内容如下：

```
01    #Expressions
02 1 (a|b)*a
03 2 ^01
04 3 a/b
05    #States
06 state 1
07 INITIAL
08    0x61 goto 3
09    0x62 goto 4
10 state 2
11    ^INITIAL
12    0x30 goto 7
13    0x61 goto 3
14    0x62 goto 4
15 state 3
16    0x61 goto 5
17    0x62 goto 6
18    match 1
19    backup match 3
20 state 4
21    0x61 goto 5
22    0x62 goto 4
23 state 5
24    0x61 goto 5
25    0x62 goto 4
26    match 1
27 state 6
28    0x61 goto 5
29    0x62 goto 4
30    match 3 and backup
31 state 7
32    0x31 goto 8
33 state 8
34    match 2
35    #Summary
36 1 start state(s)
37 3 expression(s), 8 state(s)
```

第 01～04 行是 3 条正规式，表示 3 条词法规则。其中，ALEX 源程序中的{string_ab}ab 由于使用了辅助定义正规式，最终被展开为(a|b)*a。

第 05～34 行是识别上述 3 条正规式的自动机文本描述，如图 2 所示。

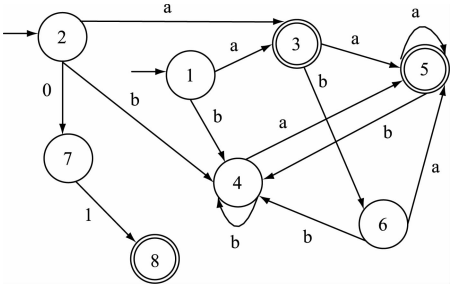

图 2 转化出的自动机

状态 1 模块中有 INITIAL 开始条件，状态 2 模块中有 ^INITIAL 开始条件，说明这 2 个状态均是开始状态。正规式中的 ^ 用于匹配行首，因此带 ^INITIAL 开始条件的状态可用于匹配带 ^ 的正规式(注意，也可匹配不带 ^ 的正规式)，而 INITIAL 开始条件标记的状态不能匹配带 ^ 的正规式。换言之，^INITIAL 状态可以匹配行首或非行首，INITIAL 状态不能匹配行首，并且前一种状态上的转移应该包含后一种状态的转移。状态 1 模块中的“0x61 goto 3”表示接受 a(16 进制 ascii 值为 0x61，如果建立工程时选择支持 Unicode，则为 0x0061)转移到状态 3，“0x62 goto 4”表示接受 b(16 进制 ascii 值为 0x62)转移到状态 4，INITIAL 开始条件表示其不能处理 ^，即不能接受第 2 个正规式 ^01 的首字母 0。状态 2 模块除了接受 a 与 b 外，“0x30 goto 4”表示接受 0(16 进制 ascii 值为 0x30)转移到状态 4。每个状态模块最后还可以包含匹配说明，分 3 种情况：match n，match n and backup，backup match n，其中 n 代表匹配第 n 个正规式。match n 表示当前状态可以为终止状态，匹配第 n 个正规式，但是如果后面还有字母，则还需要继续匹配，即尽可能多的匹配输入序列。match n and backup 表示匹配第 n 个正规式，但是同时需要回退字符，一直到遇见一个状态包含 backup and match n。

第 35～37 行是统计信息，表明有 1 个开始状态(实际按照是否接受行首分为 1 与 2 两个开始状态)，识别 3 个正规式，共计 8 个状态。

mylexer.h 与 mylexer.cpp 是 ALEX 生成的 C++源文件。mylexer.h 中声明了一个词法分析类 mylexer，与 ALEX 源程序 mylexer.l 文件第 11 行%name mylexer 相对应。mylexer.h 还包含了 mylexer.l 文件的第 05～07

行，即用户定义的类成员。mylexer.cpp 包含了词法分析类 mylexer 的方法定义，其中的建构函数与析构函数来自于 mylexer.l 的第 08～15 行。mylexer.l 中每条正规式词法规则后的语义动作(C++代码)被拷贝到 yyaction()方法中。注意，mylexer 类的核心方法是 yylex()，相当于词法分析的驱动算法，它与词法规则对应的自动机没有任何关系，因此 yylex()被定义在 mylexer 的父类中，用户根本不用关心，也不需要去修改。mylexer 类中存放的主要是自动机对应的状态转移矩阵，yylex()执行时会查表进行跳转，类似于在图 2 中所示的自动机上进行词法分析。

5 执行结果及分析

mylexer.h 与 mylexer.cpp 经过 Visual C++编译后生成词法分析器，执行时接收不同的输入字符串得到的词法分析结果如表 1 所示。

表 1 执行结果及分析

输入	输出	路径	备注
a	识别 a 结束的 ab 串：长度为 1 的 a	2-3	直接识别行首的 a
A	A	无	大小写敏感，无法识别大写 A
ca	c 识别 a 结束的 ab 串：长度为 1 的 a	1-3	c 无法识别，直接回显，再识别非行首的 a
a01	识别 a 结束的 ab 串：长度为 1 的 a	2-3	(a|b)*a 后的语义动作包含 return 1，直接返回，不再识别
01	识别在行首的 01！	2-7-8	直接识别行首的 01
c01	c01	无	c 无法识别，非行首的 01 也无法识别，全部直接回显
ab	识别 b 前面的 a！	2-3-6-3，其中 6-3 为回退	读入 b 进入状态 6，match 3 and backup 表示匹配第 3 个正规式，同时回退字符 b，到达状态 3 时遇到 backup and match 3 停止，执行 a/b 后的语义动作 return 3 直接返回
aba	识别 a 结束的 ab 串：长度为 3 的 aba	2-3-6-5	识别 ab 后没有结束，继续识别 a，遵守最长匹配原则

6 结 论

Parser Generator 使用 ALEX 工具编写 l 文件，能够自动编译生成词法分析器。ALEX 源程序转换得到 verbose 文件，它是词法分析自动机的文本描述，用于分析词法分析器的具体动作，便于用户对错误进行定位。ALEX 源程序转换得到 h 与 cpp 文件，其中定义了词法分析类 mylexer，ALEX 源程序中的 C++代码被拷贝到其中，正规式词法规则在其中表示为自动机的状态转移矩阵。正确的使用 ALEX 编写词法分析器，能够合理组成程序，提高系统性能，基于 verbose 文件及其相应的自动机也便于用户纠正词法错误。

参 考 文 献

[1] BROWN D, LEVINE J, MASON T. Lex 与 Yacc[M]. 2 版. 北京：机械工业出版社，2003.

[2] 杨朋辉，储飞黄，汪海兵. AWG 公式编辑软件的设计与实现[J]. 国外电子测量技术，2012，31(1)：92-97.

[3] 李树彪，韩敬伟. 基于多任务的智能测量仪器嵌入式软件设计[J]. 仪器仪表学报，2013，34(12)：1-7.

[4] 徐勤军，吴镇扬. 视频序列中的行为识别研究进展[J]. 电子测量与仪器学报，2014，28(4)：343-351.

[5] Bumble-Bee Software Ltd. The parser generator homepage[EB/OL]. (2010-06-13)[2015-01-09]. http://www.bumblebeesoftware.com/.

[6] Sun Microsystems, Inc. JavaCC Home[EB/OL]. (2009-09-30)[2015-01-09]. https://javacc.java.net/.

[7] RUIZ-VANOYE J A, PÉREZ-ORTEGA J, PAZOS RANGEL R A, et al. Application of formal languages in polynomial transformations of instances between NP-complete problems[J]. Journal of Zhejiang University-Science C (Computers & Electronics), 2013, 14(8): 623-633.

[8] LEVINE J. Flex 与 Bison[M]. 南京：东南大学出版社，2011.

作 者 简 介

于思江，1981 年出生，硕士，工程师，主要研究方向为编译技术。

E-mail：ysj@xupt.edu.cn

王小兵，1979 年出生，工学博士，副教授，主要研究方向为形式化方法。

E-mail：xbwang@mail.xidian.edu.cn