基于 BISON 的 ANSI C 编译器设计与实现

王知远，王树义
大连理工大学软件学院，辽宁 大连 (116024)
E-mail：ggyydlut@yahoo.com.cn

摘　要：随着编译技术的日益成熟，bison 已经替代 yacc 的成为主流的语法分析程序生成器。本文针对 ANSI C 程序循环终止证明，设计了以该语言为处理对象的编译器分析部分。设计构造出一种新的基于状态机的数据流词法分析，同时用 bison 实现了该语言编译程序的语法分析阶段。语义分析、符号表构造及中间代码生成部分使用编译原理的方法构造，并对其进行了详细地阐述。该编译器可以清晰有效的将 ANSI C 程序的结构展现出来，对于后期的工作如自动构造程序流程图，寻找循环不变式奠定了基础。

关键词：编译器；ANSI C；Bison
中图分类号：TP314

1 引言

编译器的主要功能是将高级语言编写的源程序翻译为等价的目标程序。这种翻译不是结构上的变换，而是基于语言语义的等价变换，因此编译程序的设计与实现的难度和复杂性是极高的。以往对于关于 C 语言的编译原理方面的工作都是基于 Nicklaus Wirth 先生实现的 PL/0 语言或 mini C 语言。PL/0 语言的嵌套式程序结构和当今主流编程语言已有很大区别，而 mini C 语言的编译器仅仅包含 C 语言的小部分基本特点而不能满足实际工作需要。ANSI C 编译器为程序验证，自动寻找循环不变式等难题提供了一种方向。本文使用 bison 来实现编译器的语法分析部分。Bison 是属于 GNU 项目的一个语法分析器生成器，使用它可以生成解释器，编译器，协议实现等多种程序。它不但与 Yacc(Yet Another Compiler Compiler) 兼容还具有许多 Yacc 不具备的特性，已经成为目前使用最广泛的 Yacc-like 分析器生成器。

2 ANSI C 编译程序

编译器分为两个阶段，分别是分析部分和综合部分[1]。其中分析部分被称作为编译器的前端，其主要功能如下：把源程序分解成为多个组成要素，并且在这些要素之上加上语法结构，然后使用这个结构来创建该院程序的一个中间表示。如果分析部分检查出源程序没有按照正确的语法构成，或者语义上不一致，它就必须提供有用的信息，使得用户可以按此进行改正。分析部分还会收集有关源程序的信息，并把信息存放在符号表中。图 1 说明了 ANSI C 编译器的功能结构。

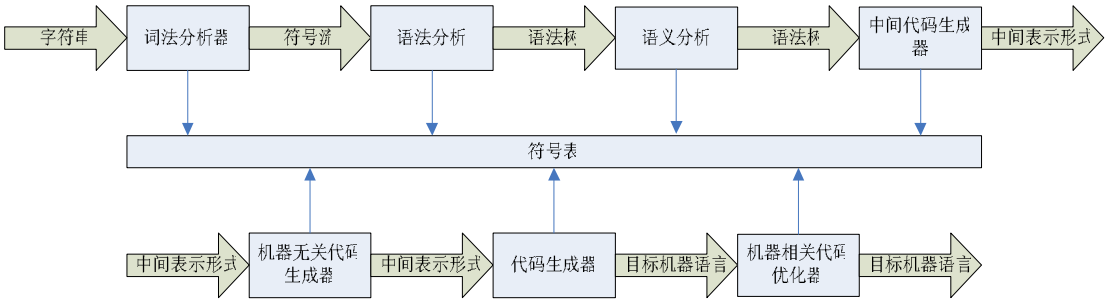

图 1　ANSI C 编译器的功能结构
Fig1 The functional structure of ANSI C compiler

2.1 词法分析

词法分析器的功能是，按照语言文法，从输入中逐个的读入源程序的符号，识别出符号串，即单词符号，然后分析单词符号的属性，并把单词符号及其属性填写在符号表中，同时把源程序改造成等价的计算机内部表示词法单元。词法分析器所输出的词法单元采用形式为<单词种别,单词属性值>的二元组。单词的种别是语法分析需要的信息,而单词属性值则是编译的语义分析和代码生成等阶段需要的信息。主要函数 init_rst_list()伪代码如下：

FILE *fp;//读取文件指针

struct result { char ch[kwlen]; int type; result *next;};//存结果的二元组链表

char op[]="+-*/%=<>:;,{}[]()&|.!?^~";

void init_rst_list(result *token 列表头)

{

if(文件打开失败){打印("open failed!"); exit(1);}

从文件取出来一个字符，存在变量 chr 中;

建立 rst 二元组列表;

while(未到达文件末端)

{

建立一个节点 rsttemp，类型为*result;

去掉注释;去空格及 tab 符;

if(chr=='\n'){rsttemp 节点类型为换行;从文件取出下一个字符;行数加 1;continue;}

if(chr>='0'&&chr<='9'){rsttemp 节点类型为 CONSTANT;将整个数字串部分读出，区分小数，科学计数法，八进制和十六进制，存入 rsttemp 节点中;从文件取出下一个字符;continue;}

if(chr=='\''){rsttemp 节点类型为 CONSTANT;将与下一个非以\'形式出现的的'之间的字符存入 rsttemp 节点;从文件取出下一个字符;continue;}

if((chr>='a'&&chr<='z')||(chr>='A'&&chr<='Z')||chr=='_')

{

while((chr>='a'&&chr<='z')||(chr>='A'&&chr<='Z')||chr=='_'||(chr>='0'&&chr<='9'))

{

将标识符或关键字写入 rsttemp 节点中，和关键字一一对比，如果存在，rsttemp 节点类型为该关键字，否则，rsttemp 节点类型为 IDENTIFIER;

}

从文件取出下一个字符;continue;

}

if(chr=='\"'){rsttemp 节点类型为 STRING_LITERAL;将与下一个非以\"形式出现的的"之间的字符存入 rsttemp 节点;从文件取出下一个字符;continue;}

for(int i=0;op[i]!=NULL;i++)if(chr==op[i])进入运算符及省略号状态;

if(运算符状态)

{

从文件取出下一个字符;

if(构成二位运算符)

{

从文件取出下一个字符;

if(构成>>=){将该三位运算符写入 rsttemp 节点中;rsttemp 节点类型为 RIGHT_OP;}
else if(构成<<=){将该三位运算符写入 rsttemp 节点中;rsttemp 节点类型为 LEFT_OP;}
else 将该二位运算符写入 rsttemp 节点中;rsttemp 节点类型为该二位运算符;
}
else{将该一位运算符写入 rsttemp 节点中;rsttemp 节点类型为该一位运算符;}
if(两个字符为..){从文件取出下一个字符;if(三个字符为...)将"..."写入 rsttemp 节点中;rsttemp 节点类型为 ELLIPSIS;}
continue;
}
if(chr 为其他非法字符)警告并返回行数;
}
关闭文件;
}

为了将词法分析的结果即符号流传递给 bison 生成的语法分析程序，提供一个接口函数 yylex()如下：

```
int yylex(YYSTYPE *yylval)
{
if(rst1->next!=NULL)
{
rst1=rst1->next;
yylval= rst1->ch;
return rst1->type;
}
else return YYEOF;
}
```

2.2 语法分析及语义分析

语法分析将从一个文法的开始符号推导出一个给定的终结符号串。推导方法是反复的将某个非终结符替换为它的某个产生式的体，最后产生语法分析树。语法分析树的根结点的标号为文法的开始符号，每个非叶子结点对应于一个产生式，每个叶子结点的标号为一个终结符号。文本中语法分析部分采用 bison 来实现。GNU bison 是一个自由软件，用于自动生成语法分析器程序，实际上可用于所有常见的操作系统。Bison 使用 BNF 范式(Backus-Naur Form)定义语法，把 LALR 形式的上下文无关文法描述转换为可做语法分析的 C 程序。

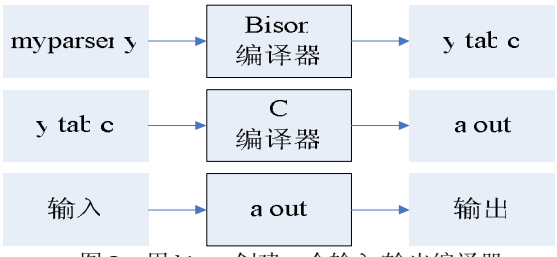

图 2 用 bison 创建一个输入/输出编译器
Fig2 Build a i/o compiler with bison

Bison 的源程序由三个部分组成[2]：

声明
%%
翻译规则
%%
辅助性 C 语言例程

声明部分包含由其他两个部分中的翻译规则及过程使用的临时变量。将 ANSI C 文法用 BNF 范式表示，并加入语义动作，作为翻译规则部分。写入 myparser.y 文件中。Bison 生成的分析器文件是定义了名为 yyparse()并且实现了那个语法的函数的 C 代码。这个函数并不能成为一个完成的 C 程序：我们需要额外的一些函数。其中之一是词法分析器。另外的一个是一个分析器报告错误时调用的错误报告函数 yyerror()。另外，一个完整的 C 程序必须以名为 main 的函数开头，并且安排它调用 yyparse()。

y.tab.c 中，主要的函数 yyparse()调用 yylex()来获得标志(token)，与标志对应的值由 lex 放在变量 yylval 中。yylval 的类型由 YYSTYPE 决定，定义 YYSTYPE 为 char *。如：

```
primary_expression
: CONSTANT    {yylval = ch;return INTEGER;}
| …
;
```

2.3 符号表

符号表是供编译器用于存放有关标识符的信息的数据结构，因为编译器分析源程序的过程会生成很多符号的属性，这些信息在编译器的分析阶段被逐步收集并放入符号表，综合阶段根据这些属性来生成合适的指令和代码的格式。当分析一个标识符的声明的时候，该标识符的信息被放入符号表中。当在后来使用这个标识符时，语义动作将从符号表中获取这些信息。本文中符号表的设计中使用了地址池和下推链域技术[3]，使得其存储效率得到了有效提升。符号表数据结构如下：

```
struct symbol
{
int gjz;//符号名在关键字池中的首位置
int src; //符号出现在文件中的行号
unsigned defined:1; //符号被定义了,避免声明多次
unsigned structarg:1; //结构参数标志
unsigned addressed:1; //地址访问的变量
unsigned computed:1; //地址树的标志
unsigned temporary:1; //生成的临时变量标志
unsigned generated:1; //生成的符号标志
int ref; //标号或变量的引用计数
int typ;//符号类型  -1 未定义
int  func_param;//函数-形参指针域，函数符号指向第一个形参位置，形参符号指向下一个形参位置，如果函数无参数或者为最后一个形参该值为-1
int **array_item;//数组内情向量指针域，指向一个 int 型数组，这个数组用来存数组的类型，维数，首地址，每一维的元素个数。还要另行计算。
```

int struct_member;//结构-成员指针域，结构标志符号存第一个成员的符号在符号表中的位置，成员是结构量的最后一个成员为-1

int scope;//作用域水平,初值 0

fuhao *fhxtyl;//下推链域

union

{

int label; //全局分配唯一的标号,这时 name 保存标号字符串

int value;

struct

{

Value v; //保存实际的常量值.

symbol *loc; //指向符号表的入口.

} c;

int seg; //全局变量或静态变量给出定义的段

}u;

};

符号表的操作有四种，分别是创建、插入、查找和删除分别由以下四个函数实现：

void init_symbol();

symbol *insert_symbol(char *name,int type,int storage_typ,int *arrayinfo,int *funcinfo);

symbol *search_ symbol (char *name);

int del_symbol(symbol* ds);

2.4 中间代码

语法分析及语义分析的结果是源代码的一种中间表示形式，称为中间代码。抽象语法树构造完成，我们就可以计算树中各结点的属性值并执行各结点中的代码片段，进行进一步的分析和综合。通过遍历语法树，为其中每个运算符结点生成一个生三地址代码，三地址代码规则如下：

1．形如 x=y op z 的赋值指令，其中 op 是一个双目算术或逻辑运算符。x、y、z 是地址。

2．形如 x=op y 的赋值指令，其中 op 是单目运算符。基本的单目运算符包括单目减，逻辑非，移位操作和转换操作。

3．形如 x=y 的复制指令，其将 y 的值赋给 x。

4．无条件转移指令 goto L，下一步要执行的指令是带有标号 L 的三地址指令。

5．形如 if x goto L 或 if False x goto L 的条件转移指令，分别当 x 为真或为假时，这两个指令的下一步将执行带有标号 L 的指令。否则下一步将照常执行序列中的后一条指令。

6．形如 if x relop y goto L 的条件转移指令。它对 x 和 y 应用一个关系运算符，如果 x 和 y 之间满足 relop 关系，那么下一步将执行带有标号 L 的指令。否则将执行指令序列中跟在这个指令之后的指令。

7．过程调用和返回通过下列指令来实现：param x 进行参数传递。call p,n 和 y=call p,n 分别进行过程调用和函数调用；return y 是返回指令，其中 y 表示可选的返回值。

8．带下标的复制指令 x=y[i]和 x[i]=y。x=y[i]指令将把距离位置 y 处 i 个内存单元的位置中存放的值赋给 x。指令 x[i]=y 将距离位置 x 处 i 个内存单元的位置中的内容设置为 y 的值。

9．形如 x=&y，x=*y 或*x=y 的地址及指针赋值指令。指令 x=&y 将 x 的右值设置为 y 的地址(左值)。这个 y 通常是一个名字，也可能是一个临时变量。它表示一个诸如 A[i][j]这样，具有左值的表达式。x 是一个指针名字或临时变量。指令 x=*y 中，假定 y 是一个指针，或是一个其右值表示内存位置的临时变量。这个指令使得 x 的右值等于存储在这个地址中的值。指令*x=y 把 y 的右值赋给由 x 指向的目标的右值。

3 翻译示例

通过以上部分的构造，文本已经实现了一个完整的 ANSI C 编译器前端，具体功能步骤如图 3 所示。

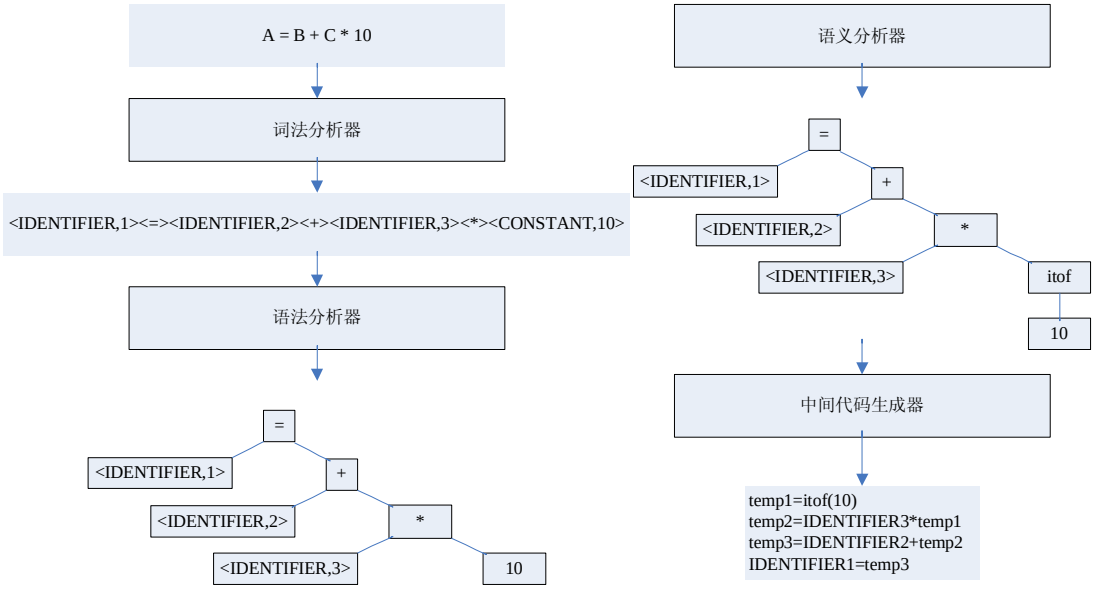

图 3　一个赋值语句的翻译
Fig3 The translation of assignment

4 结论

由于 bison 等 Yacc-like 分析器生成器的出现，很多具有研究价值或实用价值的工具的实现不再因为编译器相关工具的复杂而变得遥不可及，它为有特殊用途的编译工具的实现提供了一个优秀的选择。本文对 ANSI C 编译器的结构进行了剖析，并给出工作原理及实现。

参考文献

[1] Aho A V, Sethi R, Ullman J D. Compilers: Principles, Techniques, and Tools[M]. Addison Wesley, 1986.
[2] Tony Mason, Doug Brown. Lex & yacc[M]. O'Reilly & Associates, Inc.1996.
[3] 吕映芝,张素琴,蒋维杜. 编译原理[M]. 北京:清华大学出版社, 1998.

Design and Implementation of ANSI C Compiler Based on Bison

Wang Zhiyuan, Wang Shuyi
Dept. of Software Engineering of Dalian University of Technology, LiaoNing DaLian (116024)

Abstract

In the wake of increasing development of compiler technology, Bison has already become the most popular parser generator instead of Yacc. A compiler front-end with ANSI C was proposed for the proof of loop termination. In this paper, the syntax analysis is implemented by using Bison and data streaming lexical analysis is implemented by a new algorithm based on Finite State Machine. Semantics analyses, symbol table, intermediate code's generation, which are designed by principles of compiler, are detailed described. This compiler which can clearly and effectively reveal the structure of ANSI C program provide the foundation for the expansion application later on.

Keywords: Compiler;ANSI C;Bison