C源程序分析及理解的辅助工具—CAAT

林阿龙

(东南大学计算机科学与工程系，南京 210018)

CAAT—A TOOL FOR ANALYZING AND UNDERSTANDING C SOURCE PROGRAMS

Lin Along

(Department of Computer Science and Engineering, Southeast University, Hanjing 210018)

Abstract In this paper, We propose the idea of implementing CAAT, which is used for analyzing C source programs, and introduce the idea of generating C source program tree.

摘要 本文提出了一种辅助分析C源程序的工具—CAAT的实现思想，重点介绍了CAAT的核心内容——程序树的生成。

一、引 言

随着反编译技术的不断完善，许多软件被反编译出来。由此出现了二个有待解决的问题：(1)反编译是否违反了软件版权法？(2)得到的源程序既没有文档说明也没有注释，如何维护它们？事实上，许多没有文档或文档不全的软件都存在这个问题。对第一个问题，Byte杂志1990年专门作了讨论。如果反编译得到的软件作为研究人员进一步开发之用或利用其思想设计新软件，这在法律上都是被认可的。因此，只需要解决第二个问题就可以了。源程序中全局变量的使用，破坏了软件模块的封装性，给分析程序带来一定困难。此外，如果分析人员能了解软件中的各函数调用关系，显然有助于理解软件。通常，我们是通过对源程序的逐级抽象来理解的。因此，对程序加注释不可缺少。

本文提出一种C源程序分析的辅助工具—CAAT的实现思想及解决方案。CAAT的研制目的就是为了解决前述困难，其主要功能包括：

(1) 分层浏览C源程序，程序以格式化形式显示；

(2) 帮助用户对源程序进行抽象和加注释；

(3) 显示各类变量的使用(定值、引用)情况；

(4) 查询宏、变量、结构、联合等的定义、说明；

(5) 输出模块调用关系及源程序中的函数调用关系图。

CAAT是我们在80386微机上用C语言开发的可辅助分析2～4Mbytes C源程序的工

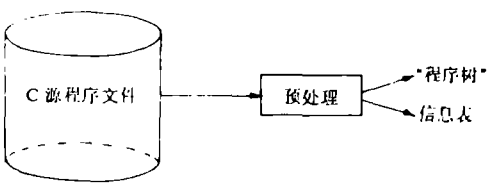
具．根据实测结果，CAAT 所需内存容量至少为它所分析的 C 源程序的 2～4 倍．当然，对于查询速度要求不高的场合可以不占用太多内存，而把中间结果存在外存．目前，CAAT 将中间结果存在内存，响应速度很高，各方面性能也较令人满意．

二、系 统 功 能

从功能上看，CAAT 由如下三个模块组成：预处理、浏览及抽象．CAAT 首先对需要分析的源程序文件(允许不全)进行预处理，然后再执行各种功能．

1．预处理

该模块的作用是将指定分析的 C 源程序转化为某种中间形式表示的语法树，同时生成各种有用的信息表．我们以后将源程序得到的语法树称为"程序树"．信息表是指源程序中的函数调用关系表、宏定义表、各种标识符表等．因此，预处理模块可描述为(见下图)：



其中，预处理模块调用了 CAAT 中的语法分析器对输入的 C 源程序进行语法分析，最后生成一棵"程序树"及有关的信息表．

2．源程序浏览及查询

该模块的功能是：在预处理基础上，对各种信息表中的内容及"程序树"进行浏览、检索、剪取，以获得所需的信息流并在屏幕或其它设备上输出．C 程序语言是一种支持单独编译的高级语言．通常，一个 C 程序由若干个 C 源程序文件组成，每个 C 文件可含有多个函数定义．因此，CAAT 将源程序浏览对象分为三级：整个 C 源程序、某一特定 C 文件以及某个指定的 C 函数．此外，在函数级 CAAT 还允许用户指定该函数内的特定层次．CAAT 以交互方式要求用户输入查询命令，然后输出．浏览功能包括：

(1) 显示系统中的函数调用关系及被调用关系．例如：对某一程序(由七个 C 文件组成)，其中的主函数 main 所调用的函数关系显示如下：

```
main (0)        (main.C / 16)
                 init_system(0)   (main.C / 571)
                 printf
                 signal
                 setjmp
                 ass_name(0)   (main.C / 789)
                 init_var(0)   (main.C / 438)
                 free_treenode (1)   (compiler.C / 228)
                 sub_rou(2)      (main.C / 319)
                 check_lex(0)   (check.C / 170)
                 lex (1)   (lex.C / 29)
                 lpt (3)   (check.C / 225)
                 struct treenode * statement(2)     (compiler.C / 233)
                 post_file(1)   (main.C / 592)
                 fill_call_fun(1)   (compiler.C / 1003)
                 menu_sub (1)    (main.C / 88)
                 free_func(0)    (main.C / 775)
                 exit
```

其中，函数名括号中的数字为该函数的形参个数．其后的"文件名／行号"指出了该函数在源程序中的定义位置．对于非用户定义的函数即系统库函数只给出函数名，那些不是系统库函

数而又没有分析定义该函数的文件的情况，在显示该函数名后指明"(undefined)"特性。函数的被调用关系显示形式上与上述类似。上述关系揭示了函数 main(定义在 main.C 文件的第16行开始)调用了 init__system，printf 等用户及系统函数。

(2) 分层浏览指定层次、函数、文件甚至整个源程序。输出时为调整后的格式。例如，对如下程序片段:

```
While ((C=getchar C))!=EOF) {i=(int) C & 0x7f;
    j=i-(i<58?48:97); printf("%d\n", j); }
```

经格式调整后的输出形式变为:

```
While ((C=getchar ( ))!=EOF)
    { i=(int) C & 0x7f;
    j=i=-(i<58? 48: 97);
    printf ("% d \n",  j);
  }
```

(3) 对指定的全局或局部变量，分层输出它在给定浏览对象内的赋值及引用情况。此时，该浏览对象内的所有与该变量无关的语句及程序片段都将被略去，有些地方用省略符"…"表示。使用户对所查询的变量的使用情况一目了然。

(4) 查询指定符号名、变量名、结构或联合名在程序中的定义、说明等情况。

(5) 对标号的查询，CAAT 既显示所查标号在源程序中的具体定义位置，还显示出源程序中查询对象内该标号的使用(转向)情况。

需要指出的是，如果用户在查询过程中忘了所要检索的符号名、变量名、结构或联合名时，CAAT 会自动匹配，将所有可能查询的该符号名的使用或定义情况显示出来。

3. 程序抽象或加注释

该模块的主要作用为: 当用户浏览程序时，如果理解了某一程序片段，希望将它抽象为某种功能(用注释加以说明)，CAAT 便以友善的接口方式辅助用户完成此项目的。根据需要，CAAT 还可在一定程度上编辑、修改源程序。因为，此时源程序已经预处理转化为"程序树"，编辑过程实际上是程序树中的某一子树的加工过程。

三、几个实现中的问题及解决方案

1. 预处理语句

我们知道，C 编译器在预处理时对 define 宏定义语句及文件蕴含语句 include 是通过扩展的方式完成的。这种解决办法对生成目标代码或可执行程序的编译器来说是必然的。但如果我们在预处理时也采用这种方案，势必造成生成的"程序树"中信息的冗余及不直观。因为程序设计人员使用这些语句的目的本来是为了增加程序的可读性和清晰性。CAAT 采用一种启发式方法，即: 多数情况下，不展开 define 及 include 语句不会影响预处理进行，只在"程序树"中生成相应的结点。只有当不展开宏或蕴含语句会使语法分析无法进行时，才采用宏扩展方案。

2. 注释语句

C 程序中的注释可以出现在任何一个地方，它以"/ * "及" * / "括起来。因此，有时出

现在一个语句中间。C 对注释是不预考虑的，全部略去。但 CAAT 不能这样处理，相反要把注释收集起来放在一个语句的合适地方。CAAT 实现中采用了一个注释栈，随着复合语句的进入或退出而压栈或退栈，栈中的内容是一个指向注释串首址的地址指针。并且规定，出现在语句中的注释一律提前到该语句之前。

3. 函数的作用域规则

在 C 中，有些函数的使用可以先引用后定义但不说明，这给函数的引用关系分析带来一定困难。我们在 CAAT 中采取了二趟扫描的方法，先处理完全部涉及到的函数定义，再对"程序树"中的函数调用关系分析，根据函数作用域原则填写调用与被调用函数的有关信息。当一个函数调用出现时，我们先在其当前文件中搜索该调用函数的定义。如果当前文件中未找到，则继续在其它预处理过的文件中寻找以非静态方式定义的函数。这一函数定义的搜索方法基于下列准则：(1) 在同一 C 程序文件中，不能同时定义两个或两上以上的同名函数；(2) 同名函数不能以外部非静态方式定义在多个文件中。

4. 类型及其变量的作用域

C 允许程序设计人员自己定义数据类型，而且是一种分程序结构语言，故存在变量及其类型的作用域问题。当我们要显示某个特定变量的使用及定义情况时，我们首先必须正确判定变量的作用域。只有在其作用域内的该变量的赋值，引用语句才被显示出来。CAAT 在预处理时为每个分层次复合语句产生一个分程序语法树结点，该结点中有两个域分别指向其外层分程序语法树根结点及该分程序内的类型定义链首结点。因此，严格说来，生成的程序树是一个图。为统一起见，以后我们仍然称之为"程序树"或"程序语法树"，并且不加区别地使用这些术语。

四、中间形式的程序树表示

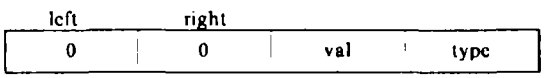
在 C 语言中，表达式语法既包括了赋值语句及函数调用语句，又可与赋值语句等结合在逻辑表达式中。由于它是 C 程序的基本构成成份，下面就从表达式的中间表示形式开始介绍。

1. 表达式的中间表示

我们将表达式划分为三类：基本量、函数调用及数组元素、算符表达式。

1) 常量及简单变量的表示

常量包括：整数、实数、字符及字符串。它们及简单变量均可用如下结构的"树结点"来描述：



其中，left 域和 right 域均置为 0，val 域存放具体的数值(包括数及字符)或者指向符号串(含变量名)的首字符地址。type 域指出了该树结点所代表的语法类型，它可以是整型数、实型数、字符、字符串或简单变量。在 CAAT 中，树结点结构定义为：

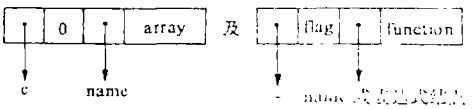
```
struct treenode {struct treenode * left;
                 struct treenode * right;
```

```
    int val;
    char type;
}
```

以后，"树结点"便是指上述描述的结构。

2) 下标变量及函数调用

CAAT 中允许下标维数超过 3 维。这二类表达式描述为如下树结点：



其中，树结点的 type 域指出了其表达式类型；val 域对数组元素来说是数组名，对函数调用来说有如下二种情况：对简单函数调用时为函数名符号串首址，对函数指针调用时为表达式(该表达式结果作为函数名)树结点地址。上述二种函数调用形式由 right 域的 flag 标志区分，当 flag 非 0 时为隐式调用；否则为显式调用。这二种结点的 left 域是一个指向如下结构的指针：

```
struct chain__node {struct treenode * expr;
                    struct chain__node * next;
                   }
```

该链上的每个结点表示了下标表达式分量或函数调用实参(形参)树结点。

3) 算符表达式

算符表达式是将上述二类表达式利用前缀、后缀或中缀算符联接起来的表达式。条件表达式、结构成员等都属于这种表达式。算符表达式的树结点中，left 和 right 域分别指向其第一操作数和第二操作数表达式所表示的树根结点指针(以后，表达式和表示该表达式的语法树不加区分)。而 val 域则为指向如下算符结构的指针：

```
struct optab { int name;
               char prece;
               char assoc;
               }
```

其中，name 域为算符名字符串首址指针，prece 为该算符优先数，assoc 为该算符的结合律。此外，对于前缀和后缀表达式，树结点的 right 域为 0。强制类型被视作前缀算符处理。

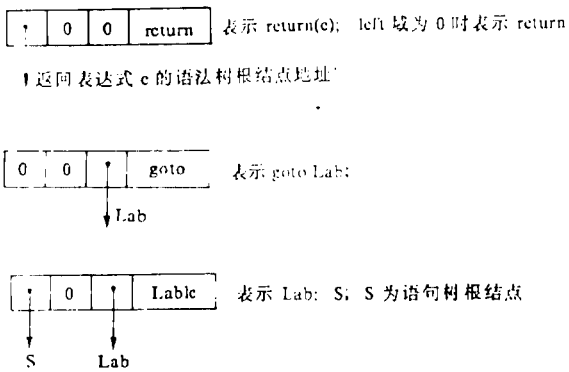
实际上，CAAT 在预处理生成程序树过程中，广泛使用了 treenode 结构类型的语法树结点。不仅用它来表示复杂表达式而且还用来描述语句、函数甚至源程序文件。因此，在下面描述语句时，我们将语句 S 和描述该语句的语法树等同起来，称为 S(表达式也同样处理)。此外，还将语法树用其树根结点来称呼。在树结点中用 S 表示。

2. 语　句

语句划分为简单句、复合句，说明语句三类(define、include 称为预处理语句)。下面依次介绍这些语句的中间表示形式。

1) 简单句

简单句包括 break、continue、goto、return 以及标号语句。其中，break 和 continue 语句单独在系统初始化时为它们生成树结点，以后在程序中遇到它们时不再生成新的语句结点。其它简单句可分别表示为：



2) 复合句

复合句包括条件语句，循环语句及 switch 开关语句。条件语句可用如下树结点表示：

S_1	S_2	e	if

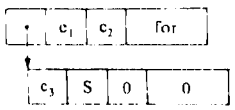
其中，S_2 为 0 时表示语句"if (e) S_1;"；否则，表示"if (e) S_1; else S_2"语句。

do__while 与 while 循环语句可表示为树结点；

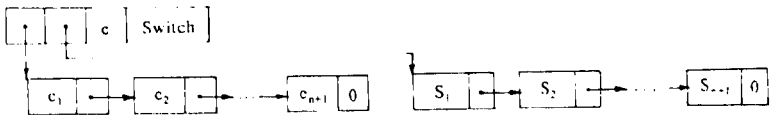
S	e	0	type

其中，type 为 do__while 或 while，分别描述语句："do {S} while (e);"及"while (e) s;"。

对 for 语句的描述略微复杂些，"for (e_1; e_2; e_3)S"；可描述为如下结构：



对开关语句 switch，采用结点域扩充的方式，表示为：



其中，left 域和 right 域指向 chain__node 结点链首结点。该树结点代表了如下语句：

```
Switch(e)
    { case c1: S1;
      case c2: S2;
               ⋮
      case cn: Sn;
      default : Sn+1;
    }
```

c_i 为 case 常量中，S_i 为对应于 c_i 的 case 执行体。

3. 说明语句

对于简单类型的说明语句，可用如下形式表示：

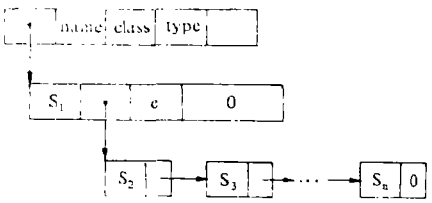
e	0	class	type

其中，class 为说明的变量的存贮方式，它可以是 register、auto、static 或 extern。type 域

指明了说明类型，如：int、short、char、float、double、long int、unsigned short、unsigned char 等简单类型。如果是结构或联合的说明语句，则 right 域为结构或联合名，type 为 struct 或 union。c 是指向说明语句中各变量构成的逗号表达式语法树根结点指针。right 域存放结构名或联合名字符串首地址。

有了上述变量说明语句，我们可将结构或联合的定义描述为如下结构：

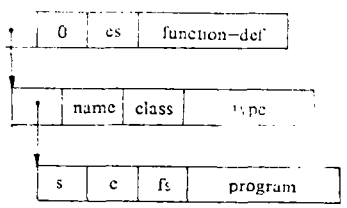
其中，name 为结构名或联合名串首址，type 为类型 struct 或 union，s_1、s_2、…、s_n 为该结构或联合定义中的成员说明语句；c 为说明语句中的变量名表达式。上述结构即定义了如下结构或联合：




```
class type name {s1;
                 s2;
                 :
                 sn;
                } c;
```

4. 函数程序树的构造

在 C 语言中，一个函数由头部说明及体部定义组成。体部定义实际上是一个分程序语句，包括数据说明及执行语句，它们可用一个语句链来表示。而头部说明则包括外部说明、函数返回类型、存贮类别、形参及形参说明语句。假设：函数定义表达式为 c，存贮类别为 class，返回类型为 type，定义体为 s，则该函数的存贮结构(或函数树结点)可描述为(见左下图)：



其中，cs 与 fs 分别指向该函数的外部说明语句链首结点和形参说明链首结点。即它们定义为：

```
struct chain__node * cs, * fs;
```

利用函数结点的 right 域我们可将一个文件中的全部 C 函数联接起来，构成一棵 C 程序文件的语法树。

因此，通过对文件语法树的串接，预处理就得到了源程序树的结果。浏览和辅助抽象均在这棵上进行信息查询及加工。由此可以看出：程序树是 CAAT 的核心和关键。

五、结 束 语

本文提出了一种实现 C 源程序辅助分析、抽象的工具—CAAT 的思想及技术。CAAT 可以帮助软件人员分析函数间的引用关系、全局变量的赋值及引用情况和辅助程序注释等功能。我们认为，对 CAAT 还可作如下改进：

(1) 完善窗口系统，使用户与系统之间的接口更为友善；

(2) 扩充图形功能，使用户能利用绘图设备直接画出函数的调用关系图及源程序流程框图；

(3) 完善抽象功能，使系统能作一定程度的自动抽象，帮助用户进行高级抽象，达到理解软件的目的。

致谢：感谢邢汉承教授对笔者多年来的指导及本文成稿过程中的指教。

参考文献(略)