基于有限状态机的 C ++ 重载唯一性确定 *

牟永敏，刘梦婷
（北京信息科技大学 计算机开放系统实验室，北京 100101）

摘　要：重载是面向对象技术中的重要特性，由于它在程序中的普遍使用，在使用基于函数调用的路径覆盖方法对程序进行测试时，会出现很多冗余函数路径，浪费测试资源和时间。针对这一问题，提出了一种基于状态机的重载唯一性确定方法。依据面向对象特点，分析提取出函数所有调用信息，将提取到的函数原型与函数调用点在状态机上进行迭代比对，确定每个调用点唯一对应的函数原型。实验表明，该方法针对程序代码中的重载函数调用点，能确定与之唯一对应的函数，有效剔除重载导致的冗余函数路径，提高测试效率。

关键词：重载；函数路径；唯一性确定；有限状态机
中图分类号：TP311　　**文献标志码**：A　　**文章编号**：1001-3695(2014)04-1059-04
doi:10.3969/j.issn.1001-3695.2014.04.025

Determination of overload uniqueness in C ++ based on finite-state machine

MU Yong-min, LIU Meng-ting
(*Open Computer System Laboratory*, *Beijing Information Science & Technology University*, *Beijing* 100101, *China*)

Abstract: Overload is an important feature in the object-oriented technology. Due to its general use in the programs, there will be many redundant function calling paths. The redundant paths waste the test resources and time. This paper proposed an algorithm to determine the overload uniqueness based on the finite-state machine(FSM). The algorithm analysed and extracted the information of the function calling according to the features of the object-oriented technology. Then it matched the function calling points and the function prototypes on the FSM iteratively. And it determined the unique function corresponding to each function calling point. The experiment results show that the algorithm can determine the unique function of each overload calling point in the program. It rejects the redundant function paths generated by overload efficiently and improves the testing efficiency.

Key words: overload; function paths; uniqueness determination; FSM

0　引言

随着面向对象软件开发技术的广泛应用，以及软件测试在软件开发过程中地位的不断上升，软件项目开发过程中面向对象技术对传统软件测试产生了新的挑战，面向对象软件测试技术的研究越来越受人们的关注[1]。重载作为面向对象技术中的重要特性，给程序的执行带来了不确定性，使得传统测试实践中的静态分析方法遇到了不可逾越的障碍，也增加了系统运行中可能的执行路径，加大了测试用例选取的难度和数量[2]。基于函数调用的路径覆盖生成方法要求确切知道被测试程序的路径数目，以便安排测试计划、分配测试资源，并对实际测试所达到的覆盖率作出评估，以决定是否结束测试工作[3,4]。因此对 C ++ 的重载函数进行唯一性确定，可以使得系统中的可能执行路径条数有效降低，并对测试覆盖率作出更加准确的评估，节约测试成本和资源。

目前项目组已经实现了软件测试工具 Regression test for C/C ++ 1.0 版本，该工具可以根据程序源代码或者是程序信息获得带有控制流的函数调用关系图和函数调用路径[5~7]。本文提出的基于状态机的重载唯一性确定方法，将重载唯一性确定问题抽象为一种模型，并且根据静态分析时获得函数重载调用点信息，建立状态机的实例，并通过在对静态分析时提取到的函数信息上迭代状态，确定重载调用点唯一对应的函数。该方法属于项目组正在开发的 Regression test for C/C ++ 2.0 工具中的一部分。

1　问题分析

按照文献[3]中对函数调用关系图和函数调用路径的定义可知，路径中的节点仅为函数名，而重载却是一个函数名可对应多个函数，但是在实际调用时，只调用唯一的一个函数。因此，重载的存在使得以往仅根据函数名来获得的调用函数路径存在很多实际运行时不存在的调用路径。因此重新定义函数调用路径，如定义 1 所示。

定义 1　函数调用图 G 的一条路径是一个节点序列（$v_i = v_{i0}, v_{i1}, \cdots, v_{im}$），路径中的节点为类名::函数名(参数列表)，路径中相邻节点表示调用关系。

如下所示的一段包含重载的代码：

```
#include〈iostream〉
using namespace std;
class person
```

收稿日期：2013-06-19；**修回日期**：2013-07-29　　**基金项目**：北京市学科与研究生教育基金资助项目(PXM2013_014224_00004)；北京市教委计划科研资助项目(KM201110772016)

作者简介：牟永敏(1961-)，男，山东烟台人，教授，博士，主要研究方向为软件理论与应用(yongminmu@163.com)；刘梦婷(1990-)，女，硕士研究生，主要研究方向为软件理论与应用.

```
}
private:
  int age;
  char * name;
  void print(int i)
  {
    cout << "print a integer: " << i << endl;
  }
  void print(char * s)
  {
    cout << "print a string: " << s << endl;
  }
public:
  person()
  {
    age = 23;
    name = "Liu Mengting";
  }
  void showAge()
  {
    print(age);//重载调用点 1
  }
  void showName()
  {
    print(name);//重载调用点 2
  }
};
int main()
{
  person p1;
  int flag;
  cin >> flag;
  if(flag == 0)
    p1.showAge();
  else
    p1.showName();
}
```

调用 Regression test for C/C ++ 1.0,画出的函数调用图 G 如图 1 所示。

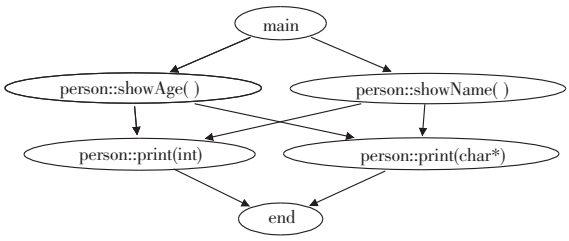

图 1 重载例子的函数调用图

按照定义 1,函数调用路径为

v_1 = (main, person::showAge(), person::print(int));

v_2 = (main, person::showAge(), person::print(char *));

v_3 = (main, person::showName(), person::print(int));

v_4 = (main, person::showName(), person::print(char *))。

而程序在实际运行时,只会执行其中的 v_1 或 v_2 这两条路径。v_3 和 v_4 这两条路径的存在是因为按照传统的方法,在重载调用点 1 时,该函数调用语句 print(age),根据函数名可对应 print(int) 和 print(char *) 两个函数;在重载调用点 2 时,该函数调用语句 print(name),根据函数名可对应 print(int) 和 print(char *) 两个函数,这就导致了冗余路径的存在。本文的重载唯一性确定算法就是找出程序中的重载调用点,并根据函数名和参数类型对可匹配的函数进行唯一性确定,从而获得重载调用点唯一对应的函数,消除因重载导致的冗余函数路径。定义 2 给出了重载唯一性确定的定义。

定义 2 如果定义重载调用点 OCP 是一条函数调用语句 OCP = 〈CN, FN, AP〉。其中,CN 是类名,FN 是调用函数名,AP 是函数的实参类型序列(AP = $ap_1, ap_2, \cdots, ap_m$)。定义程序中的函数为 F = 〈CN, FN, FP〉。其中,CN 是类名,FN 是函数名,FP 是函数的形参类型序列(FP = $fp_1, fp_2, \cdots, fp_k$)。定义 C ++ 中的类型转换规则为 CR:{R = 〈T_P, T_A〉}。其中 T 是 C ++ 中的基本类型(int, double, char…)以及程序中自定义的类型集合,T_P 是转换前类型,T_A 是转换后类型,R 是转换级别($R \in Z$, $R \geqslant 0$,转换级别越高,R 越小)。那么重载唯一性确定可表示为对于一个给定的 ocp,寻找一个 $f(f \in F)$,使得 f. CN = ocp. CN, f. FN = ocp. FN, |f. FP| = |ocp. AP|,并且对于 $fp_i \in f$. FP, $ap_i \in$ ocp. AP,满足 R_i = 〈ap_i, fp_i〉存在且 $\sum R_i$ 最小。|A| 表示序列 A 的长度。

2 唯一性确定状态机设计

定义 3 唯一性确定状态机是用于唯一性确定的有限状态机(finite state machine, FSM),包括状态集合 D、状态转移集合 T 及转移集合 conditions。其中 D = {START, ERROR, END} $\cup D_{other}$, T: $D \times$ condition $\rightarrow D$[9]。START 和 ERROR 分别表示起始状态和错误状态,D_{other} 表示其他状态的集合。

重载唯一性确定的过程可以用图 2 所示的唯一性确定状态机来描述。它以每个函数调用语句为单位,针对提取到的函数信息进行重载唯一性确定。

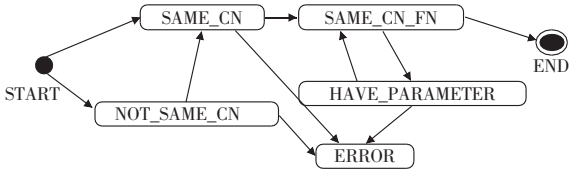

图 2 唯一性确定状态机

唯一性确定状态机的状态集合设计 D = {START, SAME_CN, SAME_CN_FN, HAVE_PARAMETER, ERROR, END}。其中,START 状态代表初始状态,表示在程序中存在自定义函数;SAME_CN 状态表示函数调用语句中的类名与进行匹配的函数类名相同; SAME_CN_FN 状态表示函数调用语句中的类名、函数名和参数个数与进行匹配的类名、函数名和参数个数相同;HAVE_PARAMETER 状态表示还有需要进行匹配的参数;ERROR 状态表示函数调用语句中的函数与进行匹配的函数不匹配;END 表示函数调用语句中的函数与进行匹配的函数匹配。

表 1 标志了唯一性确定状态机的状态转移集合 T 及转移条件集合 conditions。表 1 中转移条件中符号的具体含义可参看定义 2 中的相关定义。

表 1 唯一性确定状态机的状态转换

序号	状态转换	转移条件
0	START→SAME_CN	f. CN = ocp. CN
1	START→ERROR	f. CN! = ocp. CN
4	SAME_CN→ERROR	f. FN! = ocp. FN
5	SAME_CN→SAME_CN_FN	f. FN = ocp. FN
6	SAME_CN_FN→HAVE_PARAMETER	待匹配的参数个数大于 0
7	SAME_CN_FN→END	待匹配的参数个数等于 0
8	HAVE_PARAMETER→SAME_CN_FN	当前匹配的参数匹配成功
9	HAVE_PARAMETER→ERROR	当前匹配的参数匹配失败

3　算法设计与实现

3.1　函数调用点和函数信息提取

根据定义 2,在进行重载唯一性确定时,需要获得程序中的重载调用点、函数调用语句中实参的类型以及程序中的函数信息,因此需要对程序中的变量、类、函数等信息进行提取。各个部分需要获得的信息以及存储方式如下所示。

a)类:类名和父类名。所有类存储在一个哈希表中,以类名为 key。

b)函数:行号,函数名,所属类,形参类型和返回类型。如果函数为类外函数,则将函数的所属类置为空字符串。所有函数存储在一个哈希表中,以连接符"-"将类名、函数名和参数个数连接作为 key,如有冲突,按照哈希表的冲突处理方法进行处理。如果函数定义时带有 n 个默认参数,那么需要为该函数生成 n 个参数个数不同的重载函数,放入函数集合中。

c)变量:变量名,变量类型,所属类或所属函数。如果变量为全局变量,则将变量的所属类或函数置为空字符串。所有变量存储在一个哈希表中,以连接符"-"将变量名和所属类或函数连接作为 key。

d)自定义类型:类名,内部变量(变量名和变量类型),自定义类型的内部变量以变量名为 key,变量类型为 value。

e)函数调用点:行号,调用对象,所属类,函数名,所在函数,所在类,实参名,实参类型,所指函数。

对源程序代码进行扫描,建立待分析程序的符号表,存储类、变量和函数等信息。然后按照如下所示的类型判断算法,获得函数调用点的所属类和实参类型。

遍历函数调用点集合中的每一个函数调用点。

a)遍历函数调用点中的每一个实参名。

(a)如果实参名 name 中包含取地址符"&",计算"&"个数为 n,将 name 中"&"符去除。

(b)如果实参名 name 中包含直接选择符"."或者间接选择符"->",转入步骤(e)。

(c)根据 name 与所在函数查找变量表中信息,如果存在,则将变量类型放入 type 中,转入步骤(f)。

(d)根据 name 和所在类查找变量表中信息,如果存在,则将变量类型放入 type 中,转入步骤(f)。

(e)根据 name 查找变量表中信息,如果存在,则将变量类型放入 type 中,转入步骤(f)。

(f)将实参名按照直接选择符"."或者间接选择符"->"分割,存入 word 数组中。将 word[0]作为 name,$i=1$。转入步骤(b)。

(g)如果 type 是基本类型,则将 type 末尾加入 n 个"*",然后存入实参类型,转到步骤 a)。

(h)如果 type 是自定义类型或者类名,且 word[i]有效,则根据 type 查找自定义类型中名为 word[i]的类型放入 type 中,转到步骤(g)。

(i)将 type 末尾加入 n 个"*"存入实参类型,转到步骤 a)。

b)如果调用对象名为 this,则将所属类赋值为所在类。算法结束。

c)根据调用对象名与所在函数查找变量表中信息,如果存在,则将变量类型赋给所属类。算法结束。

d)根据调用对象名与所在类查找变量表中信息,如果存在,则将变量类型赋给所属类。算法结束。

e)根据调用对象名查找变量表中信息,如果存在,则将变量类型赋给所属类。算法结束。

3.2　类型转换规则

根据定义 2 可知,在获得了函数相关信息之后,需要首先定义 C ++ 的类型转换规则 CR:$\{R=\langle T_P,T_A\rangle\}$。根据《C ++ primer》一书的描述以及实验(Visual Studio 2010)验证,C ++ 中的基本类型有如下 17 种:BT = {bool, char, short, int, long, float, double, wchar_t, long double, unsigned int, signed int, unsigned char, signed char, unsigned short, signed short, unsigned long, signed long}。除此之外,还会有指针类型 PT(在程序信息提取时,遇到数组会自动保存为指针类型)、const 修饰的类型 COT、字面值常量 LC 和自定义类型 CUT 等。在进行重载匹配时这些类型之间的转换规则如下:

a) $CR_1:\{T_i=\text{signed}+T_i \mid T_i\in\{\text{int},\text{short},\text{long}\}\}$。

b) $CR_2:\{R=\langle T_P,T_A\rangle=0 \mid T_P=T_A\}$。

c) $CR_3:\{R=\langle T_P,T_A\rangle=0 \mid T_P\in\{\text{signed int},\text{signed short},\text{signed long}\}\}$。

d) $CR_4:\{R=\langle T_P,T_A\rangle=1 \mid T_P\in \text{BT}-\{\text{float, double, long double, unsigned long, signed int, int}\}, T_A=\{\text{int, signed int}\}\}$。

e) $CR_5:\{R=\langle T_P,T_A\rangle=1 \mid T_P=\text{float},T_A=\text{double}\}$。

f) $CR_6:\{R=\langle T_P,T_A\rangle=2 \mid T_P\in \text{BT}-\{\text{int, signed int, float}\},T_A\in \text{BT}-\{\text{int, signed int}\},T_P\neq T_A\}$。

g) $CR_7:\{R=\langle T_P,T_A\rangle=2 \mid T_P=\text{float},T_A\in \text{BT}-\{\text{double, float}\}\}$。

h) $CR_8:\{R=\langle T_P,T_A\rangle=1 \mid T_P=\text{void}*,T_A\in\text{PT}\}$。

i) $CR_9:\{T_i=\text{const}+T_i \mid T_i\in(\text{BT}\cup\text{CUT})-\text{PT}\}$。

j) $CR_{10}:\{R=\langle T_P,T_A\rangle=1 \mid T_P=\text{char}*,T_A=\text{string}\}$。

k) $CR_{11}:\{R=\langle T_P,T_A\rangle=\infty \mid$ 上述 10 条转换规则外的转换$\}$。

3.3　重载唯一性确定

根据定义 3,建立唯一性确定状态机。对所有获得的函数调用点进行分析,与得到的函数集合 F 进行匹配。具体算法如下:

LineN:行号。

CNAME:字符串集合。

CallO:调用对象。

OwnedC:所属类。

IncludedC:所在类。

FName:函数名。

ActualP:实参名。

ActualPT:实参类型。

Function:所指函数。

sumR:一次匹配中所有类型转换级别的和。

SUMR:所有转换级别和的集合。

fun:类型为 FUNCTION 的变量,存储函数信息。

FUN:所有符合类型匹配的函数集合,与 sumR 对应。

FormalPT:形参类型。

connect(a,b):以连接符"-"连接 a,b 两个字符串,并返回结果。

findFunction(a):找到以 a 为 key 的函数,并返回结果。

count(a):计算 a 中元素的个数。

CR(type1,type2):计算由 type1 转换到 type2 的转换级别。

selectMinIndex(sumR):找到 sumR 中值最小的所有索引,并返回。

输入:函数调用点 FCP,函数集合 F。

输出:函数调用点 FCP。

算法:

```
for each fcp in FCP
className = fcp. OwnedC
if className = NULL then className = fcp. IncludedC
CNAME←—className∪""
FUN←—∅
SUMR←—∅
for j = 0 to j = count( CNAME[ className] )
cName←—CNAME[ j]
tempF ← findFunction ( connect ( connect ( cName, fcp. FName ), count( fcp. ActualP) ) )
for each fun in tempF
if fun≠NULL then
sumR←—0
for i = 0 to i = count( fcp. ActualP)
sumR←—sumR + CR( fcp. ActualP[ i] ,fun. FormalPT[ i] )
if sumR = ∞ then exit for
end for
if sumR≠∞ then FUN←—fun, SUMR←—sumR
end if
end for
end for
if count( selectMinIndex( SUMR) ) = 0 then
fcp. Function←—system function
else if count( selectMinIndex( SUMR) ) = 1 then
fcp. Function←—FUN[ selectMinIndex( SUMR) ]
else
fcp. Function←—ERROR
end if
end for
```

4 实验与评测

将第 1 章问题分析中的代码作为输入数据,得到的输出结果如图 3 所示。

```
Function call: line 24,print(age),print(int)
Function: line 8,person::void print(int)

Function call: line 28,print(name),print(char*)
Function: line 12,person::void print(char*)

Function call: line 37,showAge(),person::showAge()
Function: line 22,person::showAge()

Function call: line 39,showName(),person::showName()
Function: line 26,person::showName()
```

图3 输出结果

从图 3 中可以看到,算法准确提取出了程序中的四个函数调用点,并且准确地找出了其对应调用的函数。其中第一个和第二个函数调用点为重载调用点,但是算法仍然准确识别出了实际运行时该调用点调用的函数。

调用 Regression test for C/C ++ 2.0,画出经过重载唯一性确定之后的函数调用关系图,如图 4 所示。

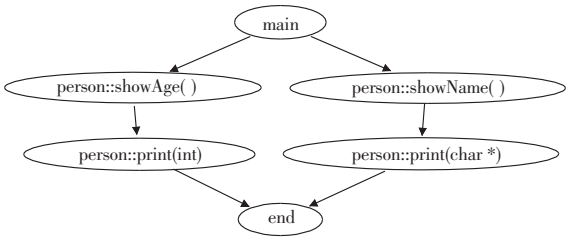

图 4 重载唯一性确定之后的函数调用关系图

为了更好地说明算法的效果,使用 Regression test for C/C ++1.0和使用了重载唯一性确定算法的 Regression test for C/C ++ 2.0 版本对几个 C ++ 源代码进行了测试。对比结果如表 2 所示。

表 2 实验结果比对

代码行数	Regression test for C/C ++1.0	Regression test for C/C ++2.0	剔除重载冗余函数路径百分比/%
40	4	2	50
96	5	3	40
218	196	4	97.96
629	121	5	95.87

从表 2 中可以看到,对于重载函数调用点,算法可以准确识别出其唯一对应的函数,并有效地减少了路径测试中的路径数目,提高了测试效率。

5 结束语

C ++ 中重载、多态和模板的存在,使得系统的冗余路径数目大大增加,增加了路径测试时测试用例的选取难度,并造成资源的浪费。本文提出一种基于有限状态机的重载唯一性确定算法,能够有效确定 C ++ 重载调用点所唯一对应的函数,有效减少了系统的冗余路径,提高了测试效率。在以后的研究中可以尝试改进算法,进一步提高算法的效率,同时可以增加对多态和模板唯一性的研究,尽可能地减少程序中的冗余路径。

参考文献:

[1] 韩欣欣. 面向对象的软件测试方法研究[J]. 信息化研究, 2010, 36(10):83.

[2] 郁莲. 软件测试方法与实践[M]. 北京:清华大学出版社,2008:184-185.

[3] 张志华,牟永敏. 基于函数调用的路径覆盖生成技术研究[J]. 电子学报,2010,38(8):1808-1811.

[4] MU Yong-min,ZHENG Yu-hui,ZHANG Zhi-hua,*et al*. The algorithm of infeasible paths extraction oriented the function calling relationship [J]. Chinese Journal of Electronics,2012,21(2):236-240.

[5] MU Yong-min, WANG Rui, ZHANG Zhi-hua, *et al*. Automatic test method research on the word part of document format translator[J]. Chinses Journal of Electronics,2013,22(1):55-60.

[6] QI Sheng, MU Yong-min, ZHANG Zhi-hua, *et al*. Research on determination of overloaded function uniqueness for regression testing oriented [C]//Proc of IEEE International Conference on Information Management and Engineering. 2010.

[7] ZHENG Yu-hui, MU Yong-min, ZHANG Zhi-hua. Research on the static function call path generating automatically[C]//Proc of the 2nd IEEE International Conference on Information Management and Engineering. 2010.

[8] STANLEY B, JOSEE L, BARBARA E. C ++ primer 中文版[M]. 4 版. 李师贤,等译. 北京:人民邮电出版社,2006:154-157.

[9] 杨睿,金大海,宫云战,等. Java 中空指针引用故障的静态检测方法[J]. 清华大学学报:自然科学版, 2011,51(S1):1509-1514.