第36卷 第19期 计 算 机 工 程 2010年10月
Vol.36 No.19 Computer Engineering October 2010
·软件技术与数据库· 文章编号：1000—3428(2010)19—0070—02 文献标识码：A 中图分类号：TP311

基于全局支配图算法的覆盖测试方法

于炳霞，谷青范
(南京航空航天大学信息科学与技术学院，南京 210016)

摘 要：目前的覆盖测试存在着大量的冗余测试用例，严重影响测试的效率。基于此，介绍一种基于全局支配图算法的覆盖测试工具，通过在局部支配图中加入辅助循环树算法及寻找临近节点，提出一种全局支配图改进算法，利用该算法能够计算出覆盖源程序的最小测试用例集。实验结果表明，该算法能够减少覆盖分析时间，生成较少的测试用例和达到较高的覆盖率。

关键词：覆盖测试；全局支配图算法；测试用例；程序流图；代码插装

Coverage Test Method Based on Global Domination Graph Algorithm

YU Bing-xia, GU Qing-fan
(College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

【Abstract】Now many redundant test cases occur in coverage test. This paper introduces an improved coverage test tool based on global domination graph algorithm, the algorithm is an improved algorithm through adding the annotated looping tree and finding nearest common node. It can compute the minimum test case set which covers the source code. Experimental result shows that the algorithm can decrease the analysis time of coverage result file, generate small test case, and achieves high coverage rate.

【Key words】coverage test; global domination graph algorithm; test case; program flow graph; code instrumentation

1 概述

覆盖测试应用最广泛的是支配图算法，最早的支配关系公式于1959年由Prosser提出。而后， Lowry和Medlock提出了关于计算支配关系的算法。Allen在1970年提供了一种数据流的解决方案，而最有名的关于支配关系的算法是由文献[1]提出的，文献[2]发展了这种算法。

在目前的市场上，用的较多的工具有Numega的True Coverage、Rational的PureCoverage TeleLogic公司的Logi Scope以及Macabe公司的Macabe，这些工具都是收费工具，而且在测试后的报告调用关系复杂，从而加大了覆盖分析的难度，造成了大量的测试用例的冗余。

本文介绍一种全局支配图算法的覆盖测试工具，通过对程序控制流程图的简化加工，使得通过这种算法能够在不降低覆盖率的基础上，减少需要生成的测试用例数量。

2 全局支配图算法及其改进

2.1 程序流程图的基本概念

基本块[3]：是指程序中这样的语句序列，它只有一个入口语句和一个出口语句，从入口语句到出口语句的语句序列是顺序执行的。

分支：是指基本块之间的控制流。分支表示了基本块之间结构关系，通过分支可以知道程序在执行完一个基本块的所有语句后将会转移到哪一个基本块继续执行。

程序流图：是程序结构的图形表示。一个基本块是流图的一个节点，一个程序的所有基本块组成了程序流图的节点集，程序第1个语句所在的基本块称为首节点，把基本块之间的控制流向作为节点之间的有向边集合。

下面引入超块的定义，一个程序P的超块S是p中使得每一基本块都执行的最大子集，当且仅当S中每一个基本块都被测试用例调用到且程序正常终止。如果一个超块Sj的执行能导致另一个超块Si的执行，但Si不是Sj的一个子程序，那么称Sj能够支配Si。

超块之间的支配关系可以用有向图来表示，超块Si支配Sj，当且仅当Si是Sj的祖先节点。如果能找到一个测试子集使得一个超块支配图的所有叶子节点被执行，那么就能确定所有超块可以被执行，进一步程序的所有基本块也可以被执行。

2.2 全局支配图改进算法

Havlak提出的算法不利于解决程序之间的调用问题，也存在着二次迭代的问题，下面在此算法的基础上，提供一个线性时间算法来改进该算法。

对于给定的一个程序P，包含P的前驱后继支配树，合并其中的前驱后继支配树，消除其中的强连通分量，得到了局部支配图。通过程序的调用关系图来合并它们，如果程序的调用关系图是非循坏图，则可以对调用关系图进行拓扑排序，通过连续的合并排好序独立的局部支配图来建立全局支配图。如果调用关系图是循环的或者不可简化的，那么处理这种图，通常采用Havlak的算法来建立嵌套循环树，对所有有出入弧的循环头做一次深度搜索，然后，以相反的顺序检验相对应的循环来确定是否含有外部调用节点，同样自下而上地建立嵌套循环树。

该算法所需要的主要函数及其函数调用关系，如图1所示。

作者简介：于炳霞(1983－)，女，硕士研究生，主研方向：覆盖测试，软件工程；谷青范，副教授

收稿日期：2010-02-23 **E-mail：**yubingxia@163.com

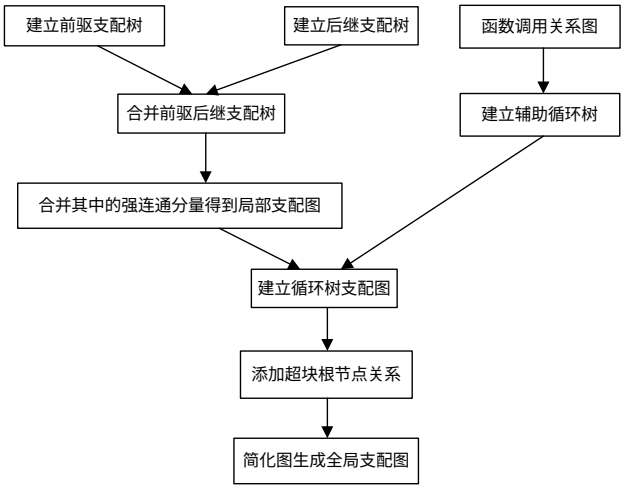

图 1　主要函数及其函数调用关系

在建立初始全局支配图的过程中，采用了合并支配图算法，避免了在寻找 U 节点在 GR 图中最邻近节点时重复遍历图，伪代码如下：

```
Merge_local_graph{ Create a null set S；
  for each function in Gu.Entryfunctions .do
      for each E in F.callsites do
          if (E in GR) add E to S;
  for each node B in S, do
      if ( B.pathlength==U.pathlength-1)
        { if(isAncestor(U,B))
          add edge(B,root(GU) to GR; break; }}
```

在这个过程中，需要对调用关系图建立循环辅助树，其伪代码如下：

```
build_assi_loop_tree (C, START) {
Node[N]= first_depth_search(C);
for (i=1 toN) Initial noChildAnce[i] and childAnce[i];
head[i]=1; type[i]= nonheader;
for each edge (v, i) entering i in C do
if(isAncestor(i,v)) then add v to childAnce[i];
else add v to noChildAnce[i]; header[1] = 0;
for j= N to1 do P = null;
for each node v in childAnce [j] do
if (v!=j)    then add Find(v) to P;
else type[j] = self; listnow = P;
if (P != null) then type[j] = reducible; Loop [j] = {j};
while (listnow !=null) do
select a node m from listnow and delete it ;
for each node y in noChildAnce [w] do y1= Find(y);
if (not isAncestor(j, y1)) then type[j]= irreducible;
add y1 to noChildAnce [j]and add m to Loop [j];
else if (y1 not in P) and (y1!=j) then
add y1to P and add y1 to listnow;
for each node m in P do
header[m] = j; Connect(m, j);
for each edge (m, y) in the loop tree do y1= Find(y);
if (y1 != j) then
replace (m, y) with a new assistant edge (j, y);
for each non-back edge (wj, y) in C, do y1= Find(y);
if (y1 != j) then
add an assistant call edge (j, y1) to the loop tree;
for each irreducible loop header, w, do
for each m in Loop[j], m !=w, do
delete all assistant edges upon m;
merge the node containing m with w;
return the assistant loop tree;}
```

3　全局支配图算法在 GCC 中的实现

GCC 的插桩[4]和覆盖率分析操作是在基本块、分支和程序流图这几种数据结构的基础上进行的，它首先分析程序源代码，进行词法和语法分析后[5]，将程序划分为若干个基本块，再按照各基本块之间的分支信息构造出程序流图[6]，然后对其进行分析，找到需要收集测试数据的关键位置(桩点)，GCC 为每一个桩点分配一个整型变量，数组长度就是桩点的数目。在这些位置处插入相应的插桩代码，最后生成目标代码。程序执行完后，数组就记录下了所有桩点的执行情况，GCC 将这个数组按照*.da 文件的格式输出，交给 GCOV 作分析和处理。图 2 是 GCOV 与 GCC 配合进行覆盖测试的工作流程。

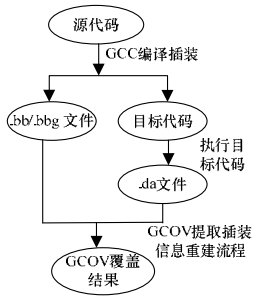

图 2　覆盖测试工作流程

GCOV 根据.bb/.bbg 文件重建流程，根据.da 文件获取插装信息，在重建流程图的过程中采用上述算法，根据覆盖报告可以看出，以最少的测试用例达到了覆盖效果，减少了分析时间。

例如，对于如下的程序：

```
void A(int i){1 if(i>0) 2 B(i);else 3 C(i);}
void B(int i ){4 D(i); 5 E(i);}
void C(int i ){6 printf("Now it is C!");}
void D(int i ){7 if(i<10)
8 printf("This program end in D!"); 9 return;}
10 else printf("Now it is D");}
void E(int i ){11 if(i<5){12 C(i);}
13 printf("Now it is E");}
```

经过用全局支配算法分析得出的程序流程，只要覆盖掉其中的 3、8、9 和 12 基本块，用如下测试用例来覆盖该程序：

```
int main(){int i=1;A(i); i=10;A(i); return 0;}
```

将整个程序命名为 excode.c

通过在 GCC 编译得到如下的结果：

```
$ gcc -fprofile-arcs -ftest-coverage -o excode excode.c
$ ./excode
This program end in D!Now it is C!Now it is ENow it is C!Now it is DNow it is ENow it i gcov excode.c
```

excode.c：正在创建‘excode.c.gcov’File‘excode.c’

已执行的行数：100% (共 28 行)

上述结果说明了所有块均被覆盖，并生成了 excode.c.gov 文件，说明了程序函数的调用次数，该实验只用了 2 个测试用例就覆盖了 ABCDE 5 个函数达到了 100%的覆盖率，而不采用此算法，可能要依次覆盖 ABCDE，不包含其中的分支就要至少 5 个测试用例，可以看出，本文算法明显减少了测试用例的数量。

（下转第 74 页）