文章编号：2095-2163(2020)10-0115-06 中图分类号：TP311 文献标志码：A

基于路径关键状态变量的测试用例约简

高 杰，赵逢禹，刘 亚
（上海理工大学 光电信息与计算机工程学院，上海 200093）

摘 要：关联矩阵测试用例约简在软件测试中具有重要的作用，可以提高测试效率，降低测试成本。本文提出了基于路径关键状态变量的测试用例约简方法，该方法通过抽象语法树获取关键状态变量信息，并基于该信息构建测试用例关联矩阵，最后根据约简准则来判断该测试用例是否可以被约简。本文选取了西门子测试用例集中四个程序进行了实验，实验结果表明，基于路径的关键状态变量测试用例约简方法可以在保证原有测试用例集测试效率的情况下，有效地减少原测试用例集的数量。

关键词：软件测试；测试用例约简；抽象语法树；关联矩阵

Test case reduction based on path critical state variables

GAO Jie, ZHAO Fengyu, LIU Ya
(School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China)

【Abstract】Incidence matrix test case reduction plays an important role in software testing, which can improve testing efficiency and reduce testing cost. This paper proposes a test case reduction method based on path key state variables. This method obtains the key state variables information through abstract syntax tree, and constructs the test case incidence matrix based on the information. Finally, it judges whether the test case can be reduced according to the reduction criteria. In this paper, four programs of Siemens test case set are selected for experiments. The experimental results show that the key state variable test case reduction method based on path can effectively reduce the number of the original test case set while ensuring the test efficiency of the original test case set.

【Key words】Software Test; Test Case Reduction; Abstract Syntax Tree; Association Matrix

0 引 言

软件测试是指使用人工或自动的手段来运行或测定某个软件系统的过程，其目的在于检验它是否满足规定的需求或弄清预期结果与实际结果之间的差别[1]，科学应用测试方法可以尽可能地避免软件在运行过程中出现各种故障问题[2]。软件测试的核心任务之一就是构建测试用例集。但软件规模的扩大会导致测试用例集会变得复杂庞大，若对其分类和约简，会节省测试的时间和资源消耗。因此，测试用例集的约简旨在最大限度地减少执行的测试用例的数量。

测试用例约简的研究可以分为两类，一类是基于模型的测试用例约简[3]，如：通过构建多优化目标模型，提出了不同的基于多优化目标的测试用例集约简算法[4]。建模过程复杂，且该严重依赖于模型设计的好坏；另一类是贪心算法、启发式算法等基于人工智能技术的测试用例约简。由于这些算法过早收敛、优化效率低的局限性，不断有学者对其进行改进，如：针对传统贪婪算法存在过拟合的问题，提出了一种基于贝叶斯网络模型的黑匣子测试选择方法[5]，该方法可以确保仅将黑盒测试之间的强关系用于测试选择，达到减少测试用例冗余的目的，而且该方法对于过度拟合更为健壮。但这种算法的效果仍取决于初始的测试用例集；应用二分 K-means 聚类算法对回归测试的测试用例集约简，以白盒测试的路径覆盖为准则，对每个测试用例进行量化，使每个用例变成一个点[6]。以黑盒测试的功能需求数作为聚类数，在聚类结果的每一簇中，按照离中心点的距离排序，依次从每一簇中选择测试用例，直至满足所有测试需求，得到约简的测试用例集。

本文认为不论是采用模型的方法还是采用贪心、启发式算法等方法，基于路径覆盖的测试用例约

基金项目：国家自然科学基金(61803264)。

作者简介：高 杰(1994-)，男，硕士研究生，主要研究方向：软件测试、软件缺陷；赵逢禹(1963-)，男，博士，教授，CCF 会员，主要研究方向：计算机软件与软件系统安全、软件工程与软件质量控制；刘 亚(1983-)，女，博士，副教授，主要研究方向：信息安全、密码学、区块链等。

收稿日期：2020-04-24

简是基本的方法,为了获取测试用例的执行路径,必须计算关键变量的状态。此外,约简后的测试用例集能否与原测试用例集具有相同的错误检测率对测试用例约简至关重要[7]。本文提出一种基于谓词和关键状态变量分析的测试用例约简方法,旨在减少测试用例的冗余,提高测试效率,降低测试成本,并与原测试用例集具有相同的错误检测率。

1 关键状态变量

1.1 关键状态变量定义

本文将影响谓词表达式中判断条件的变量称为关键状态变量。在程序运行过程中,程序传递的参数的值会影响谓词表达式中的判断条件,而谓词表达式中的判断条件的结果会影响程序运行的路径。关键状态变量包括以下 3 种参数:

(1) 程序运行过程中传递给方法的参数;

(2)程序中影响谓词表达式中的判断条件的输入或读取的成员变量;

(3) (1)、(2)中经过一系运算之后所得到的新参数。

通过代码来说明本文所定义的关键状态变量。

代码 1 关键状态变量定义范例。

```
1.int compute( int x)
2.{
3.    int d = 7;
4. if (d > 3)
5.{
6.    x = 2 * x;
7. }else
8.{
9.    x = x * x;
10.}
11. int a, b, c;
12.    b = x / 2;
13.    a = b;
14.    c = a * 3;
15. if (a ! = 0)
16.{
17.   return 1;
18. }
19. return 0;
20.}
```

在代码 1 范例中,有 x, a, b, c, d 这 5 个变量,其中 x 是传入的参数,a, d 是谓词判断中的参数,而 a 的值是由 b 赋予的。因此,本文认为 x, a, b, d 是关键状态变量。

1.2 关键状态变量获取

本文利用抽象语法树(Abstract Syntax Tree, AST) 来获取关键状态变量。抽象语法树是程序源代码的抽象语法结构的树状表现形式。

不同的计算机高级语言都提供了抽象语法树的处理包。本文所做的研究以 C 代码为实验对象,C 语言中的 gcc 编译器可以构造抽象语法树。

通过在程序源代码的抽象语法树中搜索不同的节点类型得到关键状态变量。其具体步骤是:首先在抽象语法树中找到这些节点类型,针对不同的节点类型做不同的处理,最终将满足本文定义的所有关键状态变量添加到关键状态变量集合中。

2 测试用例约简方案

测试用例集约简的目的是去除测试用例中冗余的、无法发现程序缺陷的测试用例,从而减少测试成本,提高测试效率。

本文所提出的测试用例约简方案主要有 6 步:

(1) 测试用例聚类。本文所提出的测试用例约简方法与代码执行的路径和关键状态变量密切相关。因此,要将测试用例按其测试的功能以及输入参数的参数列表聚类,聚类成若干个不同的测试用例集。

(2)查找每类测试用例集对应的源代码。针对聚类后的每类测试用例集,根据测试的功能与输入参数的型构确定被测程序的程序源代码。

(3)代码预处理。程序源代码中的每个文件可能会有多个方法,每个方法中的局部变量可能会出现多次赋值和使用的情况。

例如代码 2 所示的代码中,当程序执行执行到第 11 行的条件判断语句时, x 的值并不能确定,可能会取第 5 行的 x, 也可能会取第 8 行的 x,即同一个变量在不同的情况下可能会被赋予不同的值。为了解决这种情况,本文对代码进行了预处理,保证每个变量对应一个值。

代码 2 代码预处理示例

```
1.int compute( int x)
2. {
3.    if (x > 3)
4.    {
5.       x = 2 * x;
6.    }else
7.    {
8.       x = x * x;
```

```
9.  }
10.     …
11.  if (x > 10)
12.  {
13.     int y = x;
14.  }
15. }
```

对代码中的变量进行预处理就是针对同一变量可能会被赋予不同值的变量换成不同的变量名。代码 2 中示例代码经预处理后的结果如代码 3 所示。

代码 3　代码预处理结果示例

```
1.int compute( int x)
2.  {
3.   if (x > 3)
4. {
5.      x1 = 2 * x;
6.     } else
7. {
8.   x2 = x * x;
9.  }
10.   …
11.  if (x1 > 10)
12.  {
13.  int y = x1;
14.   } else if
15.  {
16.       int y = x2;
17.   }
18.}
```

(4)基于抽象语法树的信息提取。利用高级语言中所提供的技术构建程序源代码的抽象语法树。获取抽象语法树中的每个节点信息，从而找到代码中的关键状态变量、关键状态变量的计算表达式以及谓词表达式中的判断条件，并将它们分别添加到相应的集合中。

(5)构建测试用例关联矩阵。测试用例集中的每个测试用例对应测试用例关联矩阵的一行信息，多个测试用例构成了测试用例关联矩阵。测试用例关联矩阵用 $\boldsymbol{M}$ 表示，$\boldsymbol{M}$ 矩阵中每一行对应一个测试用例的输入参数、关键状态变量、关键状态变量的计算表达式和谓词表达式中的判断条件，形式化表示为式(1)：

$$\boldsymbol{M}_i = (t_i, < V_{i1}, V_{i2}, \cdots, V_{ik} >, < E_{i1}, E_{i2}, \cdots, E_{im} >, < C_{i1}, C_{i2}, \cdots, C_{in} >). \quad (1)$$

其中，t_i 表示测试用例中第 i 个测试用例的输入参数，V 表示关键状态变量的集合，E 表示关键状态变量的计算表达式集合，C 表示谓词表达式中的条件判断的集合。

(6)测试用例约简。本文采用的测试用例约简准则是：任意二个测试用例 t_i，t_k，如果对应的谓词表达式中的条件判断的逻辑值完全相同，即 $< C_{i1}, C_{i2}, \cdots, C_{in} > = < C_{k1}, C_{k2}, \cdots, C_{kn} >$，那么这二个或多个测试用例属于同一等价类的测试用例，具有相同或相似的测试效果，可以选取其中的一个作为约简后测试用例。

该约简方案的整体流程框架，如图 1 所示。

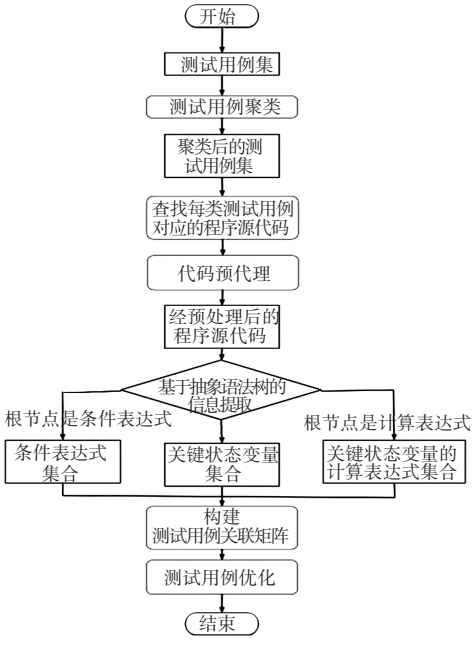

图 1　约简方法流程框架

Fig. 1　Reductionprocess framework

3　主要算法实现

在图 1 所示的流程图中，主要有二个重要的算法：

(1)在抽象语法树中获取关键状态变量；

(2)在测试用例关联矩阵中查找并移除等效的测试用例，实现测试用例约简。

3.1　基于抽象语法树的信息提取

任何一个 C 程序文件都可以转换为抽象语法树的树状结构，任意一个包括一个变量和一个方法的 C 程序文件的树形结构，如图 2 所示。利用抽象语法树中提供的方法可以获取树中的各个节点。

为获取关键状态变量、关键状态变量的计算表达式以及谓词表达式中的判断条件，需要从抽象语法树中找到与其相关的 3 类节点：params 节点、

IfStatement 节点以及 ExpressionStatement 节点。针对这 3 类不同的节点分别找出关键状态变量、关键状态变量的计算表达式和谓词表达式中的判断条件集合,并分别将其添加到相应的集合。算法 1 给出了获取关键状态变量集合、关键状态变量的计算表达式集合以及谓词表达式中的判断条件集合的算法。

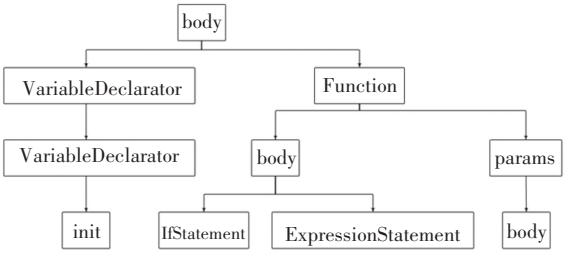

图 2 抽象语法树

Fig. 2 Abstract syntax tree

算法 1 基于抽象语法树的信息提取算法

输入 预处理后的程序源代码的抽象语法树。

输出 关键状态变量集合 keyVariable,关键状态变量的计算表达式集合 comExpression,谓词表达式中的判断条件集合 conExpression。

(1)算法以 root 为根节点;

(2)初始化关键状态变量集合 keyVariable = null,关键状态变量的计算表达式集合 comExpression = null,谓词表达式中的判断条件集合 conExpression = null,根 root 入队 enqueue(root);

(3) 循环 queue 中的元素。取出队首元素,如果为空,转(8),算法结束,否则转(4);

(4)如果该节点的节点类型是 IfStatement,找到该节点下的表达式,并将该节点添加到谓词表达式中的判断条件集合 conExpression 中,并将该节点下的变量放入一个 set 集合中;

(5)如果该节点的节点类型是 VariableDeclaration 或者 params 或者 Identifier,并且该变量在(4)的 set 中出现过,将该节点添加到关键状态变量集合 keyVariable 中;

(6)如果该节点的节点类型是 BinaryExpression 或者 CallExpression 或者 UnaryExpression。继续向下判断,若该节点包括符号“<”,“>”,“<=”,“>=”,“!=”,“==”,找到该节点下的表达式,并将该节点添加到谓词表达式中的判断条件集合 conExpression 中;若该节点包括符号“=”,找到该节点下的表达式,并将该节点添加到关键状态变量的计算表达式集合 comExpression 中;

(7)如果该节点的节点类型不属于以上节点,则把该节点的子节点入队;

(8)算法结束。

通过算法 1 可以获取测试用例关联矩阵 M 中的关键状态变量集合、关键状态变量的计算表达式集合以及谓词表达式中的判断条件集合。

3.2 基于测试用例关联矩阵的约简

3.2.1 构建测试用例关联矩阵

测试用例关联矩阵的列由关键状态变量集合、关键状态变量的计算表达式集合以及谓词表达式中的判断条件集合中的每个元素组成,每个测试用例的输入值、关键状态变量集合、关键状态变量的计算表达式集合以及谓词表达式中的判断条件集合构成测试用例关联矩阵的每一行, n 个测试用例构成了测试用例关联矩阵。图 3 给出了一个测试用例关联矩阵示例。

$$M=\begin{array}{c|ccc|cc|ccc}
\text{测试用例集合} & \text{关键状态变量集合} & & & \text{关键状态变量计算表达式集合} & & \text{条件表达式集合} & & \\
 & a & b & c & a_1=a*2 & a_2=c*3 & a-b>4 & a_1>5 & a_2>2 \\
\hline
t_1 & 2 & 2 & 3 & 4 & 9 & \text{false} & \text{false} & \text{true} \\
t_2 & 5 & 10 & 0 & 10 & 0 & \text{false} & \text{false} & \text{false} \\
t_3 & -6 & -12 & 5 & -12 & 15 & \text{true} & \text{true} & \text{true} \\
t_4 & 11 & 0 & 0 & 22 & 0 & \text{false} & \text{true} & \text{false}
\end{array}$$

图 3 测试用例关联矩阵示例

Fig. 3 Example of test case correlation matrix

构造测试用例关联矩阵的输入可以概括为:测试用例集 $T=\{t_1,t_2,t_3,\cdots,t_m\}$,由算法 1 中得到的关键状态变量集合、关键状态变量的计算表达式集合以及谓词表达式中的判断条件集合,其数学表达式为式(1)。

在图 3 的示例中,t_1,t_2,t_3,t_4 表示 4 个测试用例,每个测试用例有 3 个输入参数:a、b、c;$a_1=a*2$ 和 $a_2=c*3$ 表示代码中的计算表达式;$a-b>4$,$a_1>5$,$a_2>2$ 则表示谓词表达式中的判断条件。

3.2.2 测试用例约简算法

根据本文所提出的测试用例约简准则对测试用例关联矩阵进行约简,算法 2 给出了测试用例关联矩阵的约简描述。

算法 2 测试用例关联矩阵中测试用例的约简

输入 测试用例关联矩阵 $\boldsymbol{M}$

输出 约简后的测试用例关联矩阵 $\boldsymbol{M'}$

(1)for each i to row // row 是测试用例关联矩阵的行数

(2) for each $j=i+1$ to row

(3) 如果第 i 行和第 j 行的测试用例对应的谓词表达式中的判断条件集合中的每一列的结果值都相同,表示第 i 行和第 j 行的测试用例具有相同或相

似的功能，保留第 i 行的测试用例

（4）　end for

（5）　end for

4　实验分析

4.1　实验数据及评价指标

本文选择了西门子测试用例集中 4 个程序作为实验对象来验证本文所提出的测试用例约简的方法。为了评估本文所提出的方法的有效性，将本文算法与传统 HGS 算法和贪心算法（Greedy，G 算法）从约简率作对比和分析。约简率的计算公式（2）。

$$\text{约简率} = \frac{|\,|OriginalTestSuite| - |T^*|\,|}{|OriginalTestSuite|}. \quad (2)$$

其中：$|OriginalTestSuite|$ 表示原测试用例集中测试用例数量，$|T^*|$ 表示约简后测试用例集中测试用例的数量，约简率越大则约简程度越高。

西门子测试用例集中 4 个程序的程序集信息如见表 1。schedule2 是优先级调度器；tcas 是防止航空器空中相撞系统[8]；print_tokens 和 print_tokens2 主要用于词法分析。

表 1　西门子程序集信息

Tab. 1　Siemens assembly information

程序	版本数	代码行数	测试用例数量
schedule2	10	307	2 680
tcas	41	173	1 578
print_tokens	7	563	4 056
print_tokens2	10	508	4 071

4.2　实验方案

实验的具体方案：

（1）按不同的功能及输入参数的参数列表分别将四个开源程序中的测试用例集进行聚类，确定每一类测试用例集所对应的程序源代码；

（2）将对应的程序源代码预处理，将预处理后的源代码转换成抽象语法树，再使用本文所提出的算法 1 获取关键状态变量集合、关键状态变量的计算表达式集合以及谓词表达式中的判断条件集合；

（3）构建测试用例与关键状态变量集合、关键状态变量的计算表达式集合以及谓词表达式中的判断条件集合的测试用例关联矩阵，再计算测试用例关联矩阵中每一项所对应的值，最后根据测试用例约简准则对其进行约简。

4.3　实验结果和分析

实验结果如图 4 所示，可以看出本文所提出的算法可以有效的减少原测试用例集数量。

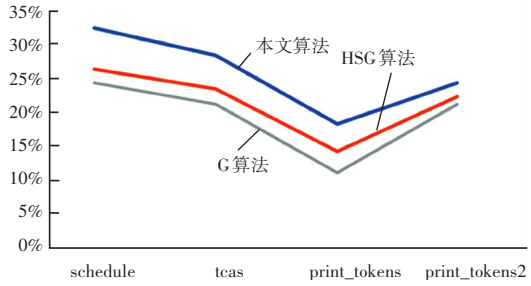

图 4　3 种算法的测试用例约简率

Fig. 4　Test case reduction rate of three algorithms

3 种算法约简后的测试用例数量见表 2。

表 2　约简后测试用例的数量

Tab. 2　Number of test cases after reduction

应用名称	初始测试用例数量	约简后的测试用例数量		
		本文算法	HGS 算法	G 算法
schedule2	2 680	1 822	1 983	2 037
tcas	1 578	1 136	1 215	1 247
print_tokens	4 056	3 326	3 488	3 610
print_tokens2	4 071	3 094	3 135	3 216

通过对西门子测试用例集中 4 个开源程序自带的测试用例集进行实验，本文所提算法对每个程序中的原测试用例数量都有不同程度程度的减少，测试用例数量的约简率还与原测试用例集中冗余测试用例数量的多少相关。

本文所提出的方法在保证测试用例集完整性的基础上减少了测试用例的数量。实验中针对约简掉的每个测试用例都经过了人工检测，通过对比测试用例关联矩阵中的测试用例，判断测试用例关联矩阵中存在与被约简的测试用例的每一列对应的值都是相同的，运行被约简的测试用例，进一步证实了这些测试用例集确实不能检测到程序中存在的缺陷。但本文算法涉及到的步骤相对较多，会增加计算方面的开销。

5　结束语

测试用例的约简是软件测试中一项非常重要的工作，约简测试用例可以极大地减少测试成本，因此对测试用例约简的研究具有十分重要的意义。

本文提出一种基于谓词分析和关键状态变量的测试用例约简方法，主要考虑了变量对测试路径的影响，提出一种基于谓词分析和关键状态变量的测试用例约简方法，主要考虑了变量对测试路径的影响。该方法通过捕获源代码中与关键状态变量相关语句和谓词表达式中的判断条件之间的关系来构造测试用例和关键状态变量、关键状态变量计算表达

（下转第 126 页）