第20卷第9期 控 制 与 决 策 2005年9月
Vol. 20 No. 9 Control and Decision Sept. 2005

文章编号: 1001-0920(2005)09-1065-04

一种基于路径的测试数据自动生成算法

陈继锋[1a], 朱 利[1b], 沈钧毅[1a], 陈 玲[2]
(1. 西安交通大学 a. 计算机软件研究所, b. 软件学院, 西安 710049; 2. 长安大学 信息工程学院, 西安 710064)

摘 要: 提出了一种新的基于路径测试数据自动生成的算法. 该算法将路径中的线性谓词函数直接作为线性算术表示来构造谓词函数关于输入变量的线性约束, 仅当谓词函数是输入变量的非线性函数时, 才计算其线性算术表示, 因而不必计算所有谓词函数的线性算术表示, 也不必计算谓词片和确定输入依赖集, 以及构造谓词函数关于输入变量的增量的线性约束. 理论分析和实例证明, 该算法具有简单、容易、有效且计算量小等特点.

关键词: 谓词函数; 线性约束; 线性算术表示; 输入变量
中图分类号: TP311 **文献标识码**: A

Path-Based Automatic Test Data Generation Approach

CHEN Ji-feng[1a], *ZHU Li*[1b], *SHEN Jun-yi*[1a], *CHEN Ling*[2]
(1a. Institute of Computer Software, 1b. School of Software, Xi'an Jiaotong University, Xi'an 710049, China. 2. School of Information Engineering, Chang'an University, Xi'an 710055, China. Correspondent: CHEN Ji-feng, E-mail: jfchen@mail.xjtu.edu.cn)

Abstract: A new approach to path-based automatic test data generation is presented. The linear predicate function on a given path is directly used as linear arithmetic representation to construct linear constrains of predicate functions for input variables. Only if the predicate function is nonlinear, the linear arithmetic representation needs to be computed. The constructions of predicate slice, input dependency set and the linear constraint of predicate function on the increments for the input are no longer needed. Theoretical analysis and practical test show that this approach is simple effective, and takes less computation.

Key words: Predicate function; Linear constrain; Linear arithmetic representation; Input variable

1 引 言

面向路径的测试数据自动生成方法是软件白盒测试中一种重要的方法. 文献[1]将其分为静态法[2~6]、动态法[7~11]和试探法[12]等, 并进行了分析和比较. 在这些方法中, 文献[2]描述了使用符号执行针对线性约束路径自动产生测试数据, 但它仅能探测到线性路径约束条件中不可达的路径, 并且处理数组的功能有限. 文献[3]获得的约束是不精确的, 其测试数据的获得并未经过迭代提炼, 所以当求出的近似解不能使给定的路径被经过时, 该方法是失败的. 文献[7, 8]是使用回溯的方法, 每一次仅考虑一个分支谓词和一个输入变量, 因而即使路径中所有的分支约束条件都是输入变量的线性函数, 也需进行大量的迭代. 为解决以上问题, 文献[10]提出了用线性算术表示对谓词函数进行线性化, 并建立输入变量增量的线性约束系统, 通过迭代提炼求得测试数据, 较好地解决了[2, 3]中的问题. 另外, 该方法在每次迭代时程序的执行次数与路径的长度无关, 仅受限于输入变量的个数, 所以它能够避免文献[7, 8]中回溯所带来的资源浪费问题, 但它需要计算谓词片, 确定输入依赖集, 计算路径中所有谓词函数的线性算术表示和其谓词残量, 以及构造谓词函数关于输入变量增量的线性约束. 文献[11]对[10]的方法进行了改进, 可不计算谓词片和确定输入依赖

收稿日期: 2004-10-15; **修回日期**: 2005-03-16.
基金项目: 国家863高技术研究发展计划基金项目(2003AA1Z2610).
作者简介: 陈继锋(1966—), 男, 湖南浏阳人, 博士生, 从事软件测试自动化研究; 沈钧毅(1939—), 男, 江苏常熟人, 教授, 博士生导师, 从事软件工程、数据挖掘等研究.

集, 在一定程序上简化了算法, 但仍需计算所有谓词函数的线性算术表示及其谓词残量

基于以上情况, 本文提出了直接用线性谓词函数构造线性约束系统的方法, 并且只需构造谓词函数关于输入变量的线性约束, 而非输入变量增量的线性约束. 该方法不必计算谓词片和确定输入依赖集, 另外在一般的程序中, 谓词函数大部分都是线性的, 所以采用本文算法, 较其他方法可有效地减少算法的计算量, 使求解过程更加直观简单, 容易理解.

2 算法的提出

2.1 基本概念

为便于说明, 以如下程序 1 为例, 定义算法中的几个基本概念:

```
程序 1:  0:  read (X, Y, Z)
         1:  U = (X - Y) * 2
         P1: if (X > Y) then
         2:  W = U
         3:  else W = Y endif
         P2: if (W + Z) > 100 then
         4:  X = X - 2
         5:  Y = Y + W
         6:  write ("Linear")
         P3: else if (X^2 + Z^2 > 100) then
         7:  Y = X * Z + 1
         8:  write ("Nonlinear: Quadratic")
         endif
         P4: if (U > 0) then
         9:  write (U)
         P5: else if (Y - Sin(Z)) > 0 then
         10: write ("Nonlinear: Sine")
         endif
```

定义 1 路径上具有明确取值要求的判断语句中的条件表达式称为分支谓词. 如程序语句 P1 中的 $X > Y$.

定义 2 分支谓词所在的判断语句的表达式 $E1$ op $E2$, 总可以转换成 F op 0 的形式, 其中 op $\in (<, \leq, >, \geq, =, \neq)$, 那么 F 则称为谓词函数. 如分支谓词 $X > Y$, 可转换成 $X - Y > 0$, 则 $F = X - Y$ 为谓词函数.

定义 3 根据路径 P 上的谓词结点 n_i 关于给定输入 I_k 的输入变量, 写出一个通用线性函数 $L(n_i, I_k, P)$, 然后计算 $L(n_i, I_k, P)$, 使得 $L(n_i, I_k, P) = 0$ 代表谓词函数 F 在 I_k 处的切平面, 则称 $L(n_i, I_k, P)$ 为 n_i 的谓词函数 F 关于 I_k 的线性算术表示.

定义 4 若谓词函数 F 为线性函数, 则称表达式 F op 0 为线性约束; 否则, 称 F 的线性算术表示 $L(n_i, I_k, P)$ op 0 为线性约束.

2.2 理论基础

本文以迭代松弛法[10,11] 为基础, 给出以下两个定理.

定理 1 如果路径 P 中的谓词函数是线性的, 则其线性算术表示就是谓词函数本身.

证明 因已知谓词函数是线性的, 故可设谓词函数为

$$F(X) = a_1X_1 + a_2X_2 + \cdots + a_iX_i + \cdots + a_mX_m + a_0$$

其中: X_i 为输入变量, a_i 为系数, m 为输入变量个数, $i \in (1, 2, \cdots, m)$, a_0 为常数项.

设 k 为迭代次数, $k \in (0, 1, 2, \cdots, T)$, 则第 k 次迭代的 $F(X)$ 关于 $X_{i,k}$ 的均差为

$$F[X_{i,k+1}, X_{i,k}] = \frac{F(X_{i,k+1}) - F(X_{i,k})}{X_{i,k+1} - X_{i,k}} = \frac{a_{i,k}(X_{i,k+1} - X_{i,k})}{X_{i,k+1} - X_{i,k}} = a_{i,k};$$

$F(X)$ 关于 $X_{i,k}$ 的导数为: $F'_{X_i} = a_{i,k}$.

可见, $F(X)$ 的均差等于 $F(X)$ 的一阶导数, 均为线性谓词函数的变量系数, 而与输入变量无关, 因此其线性表示即为谓词函数 $F(X)$ 本身, 定理 1 得证. □

定理 2 谓词函数关于输入变量的线性约束与谓词函数关于输入变量增量的线性约束等价.

证明 设谓词函数为(其中各参数意义同定理 1)

$$F(X) = a_1X_1 + a_2X_2 + \cdots + a_iX_i + \cdots + a_mX_m + a_0 = \sum_{i=1}^{m} a_iX_i + a_0.$$

由迭代松弛法知, $F(X)$ 第 k 次迭代的谓词残量为

$$R(X_{i,k}) = \sum_{i=1}^{m} a_{i,k}X_{i,k} + a_{0,k};$$

$F(X)$ 第 k 次迭代的关于输入变量增量的线性约束表达式为

$$\sum_{i=1}^{m} a_{i,k}\Delta X_{i,k} + R(X_{i,k}) \text{ op } 0, \tag{1}$$

其中 op $\in (<, \leq, >, \geq, =, \neq)$.

因为

$$\Delta X_{i,k} = X_{i,k+1} - X_{i,k},$$

将 $\Delta X_{i,k}$ 和 $R(X_{i,k})$ 代入式(1) 可得

$$\sum_{i=1}^{m} a_{i,k}X_{i,k+1} + a_{0,k} \text{ op } 0. \tag{2}$$

式(2) 即为谓词函数 $F(X)$ 关于输入变量的线性约束表达式, 其求得的 $X_{i,k+1}$ 即为第 $k+1$ 次迭代的输入变量值 I_{k+1}. 因为式(2) 是由式(1) 推出的,

所以式(1) 与(2) 等价, 故定理 2 得证. □

2.3 主要思想

依据定理 1 和定理 2, 新算法的主要思想是: 从输入变量的值域中, 任选一组输入, 考察给定路径上各分支谓词, 当谓词函数是线性表达式时, 直接将其作为线性算术表示来构造线性约束; 当谓词函数是非线性表达式时, 则计算其关于当前输入的线性算术表示, 然后将得到的线性算术表示与路径上的线性谓词函数一起构造输入变量的线性约束系统, 进而建立输入变量的线性方程系统, 求解出输入变量值, 从而获得一组新的输入. 若新的输入仍不能使给定的路径被经过, 则重复以上过程, 直至求出所期望的输入, 或达到规定的迭代次数上限, 算法结束.

2.4 算法描述

根据第 2.3 节, 可将算法具体描述如下:

输入: 路径 $P = \{n_1, n_2, \cdots, n_k\}$, 程序的初始输入 I_0 以及迭代次数上限 T;

输出: 使路径 P 被经过的程序输入 I_f;

```
procedure TESTGEN (P, I0)
  if I0 能让 P 被经过 then If = I0 return
    k = 0; Done = false
  while (not Done) and (k ≤ T) do
    for P 上每个谓词结点 ni do
      if 路径 P 上 ni 的谓词函数为非线性函数
      then
      Step1: 计算 ni 的谓词函数的线性算术表
             示 L (ni, Ik, P)
      endif
      endfor
      Step2: 用 P 中的线性谓词函数和 L (ni, Ik,
             P) 构造输入变量的线性约束系统
      Step3: 求解线性约束系统, 得到新的输入
             Ik+1
      if Ik+1 能让 P 被经过 then If = Ik+1;
             Done = true
      else k + +
      endif
  endwhile
endprocedure
```

3 实例验证

3.1 算法应用实例

利用程序 1, 使用新算法进行迭代求解. 分别考虑路径 P 中是否含有非线性谓词函数的情况.

例 1 谓词函数为线性函数

选取路径 $P = \{0, 1, P_1, 2, P_2, 4, 5, 6, P_4, 9\}$, $I_0 = (X_0, Y_0, Z_0) = (1, 2, 3)$.

因 I_0 不能使 P 通过, 故继续执行算法后面的步骤.

因 P 中谓词函数 P_1, P_2, P_4 均为线性的, 所以直接执行 Step2, 构造谓词函数关于输入变量 I_0 的线性约束系统:

Step2:

$$\begin{cases} X - Y > 0, \\ 2X - 2Y + Z - 100 > 0, \\ 2X - 2Y > 0. \end{cases}$$

Step3: 求解线性约束系统.

根据文献[13, 14] 中求解线性约束系统的方法, 将线性约束不等式转变成线性约束方程, 再求最小二乘解, 可求得 $X = 2.6, Y = 2, Z = 99.8$. 则新的输入 $I_1 = (2.6, 2, 99.8)$.

因 I_1 可使路径 P 被经过, 故算法结束.

例 2 谓词函数中含有非线性函数

选取路径 $P = \{0, 1, P_1, 3, P_2, 4, 5, 6, P_4, P_5, 10\}$, $I_0 = (X_0, Y_0, Z_0) = (1, 2, 3)$, 迭代增量 $\Delta X = \Delta Y = \Delta Z = 1$.

可知, I_0 不能使 P 通过, 故继续执行算法后面的步骤.

因 P 中谓词函数 P_5 是非线性的, 故执行 Step1:

Step1: 求 P_5 的线性算术表示:

P_5 的谓词函数为 $F = Y - \sin(Z)$, 因而可令其线性算术表示为

$$L(BP_5, I_k, P) = bY + cZ + d$$

利用均差近似导数, 可求得 $b = 1, c = 0.89792$, 又

$$bY_0 + cZ_0 + d = Y_0 - \sin(Z_0),$$

故可求得 $d = -2.83488$, 所以谓词函数 $F = Y - \sin(Z)$ 关于 I_0 的线性算术表示为

$$L(BP5, I_0, P) = Y + 0.89792Z - 2.83488.$$

Step2: 用 P 中的线性谓词函数 P_1, P_2, P_4 和 $L(BP_5, I_0, P)$ 构造输入变量的线性约束系统

$$\begin{cases} X - Y \leq 0, \\ Y + Z - 100 > 0, \\ 2X - 2Y \leq 0, \\ Y + 0.89792Z - 2.83488 > 0. \end{cases}$$

Step3: 求解线性约束系统.

按照例 1 中的求解线性约束系统的方法, 可求得 $X = -80.15, Y = -79.15, Z = 180.5$. 则得到新的输入 $I_1 = (3.235, 3.835, 51.196)$.

因 I_1 仍不能使路径 P 被经过, 故重复 Step1～Step3, 直到第 3 次迭代, 求得 $I_3 = (101.125, 101.725, -0.398)$ 可使路径 P 被经过, 算法结束,

I_3 即为所求. 各次迭代所产生的测试数据及分支谓词的覆盖情况如表 1 所示. 其中: T 表示 True; F 表示 False.

表 1 测试数据及分支谓词覆盖情况一览表

迭代次数	X	Y	Z	BP_1	BP_2	BP_3	BP_4
0	1	2	3	F	F	F	T
1	3.235	3.835	51.196	F	F	F	T
2	-5.682	-5.082	122.563	F	T	F	F
3	101.125	101.725	-0.398	F	T	F	T

3.2 结果分析

从实例可看出, 用新的算法求解给定路径 P 的测试数据时, 当 P 中的谓词函数为线性表达式, 该方法十分简单、容易, 不必计算谓词片和确定输入依赖集, 也不必计算谓词函数的线性算术表示、谓词残量以及构造谓词函数关于输入变量的增量的线性约束, 迭代 1 次即可求得所需测试数据. 当 P 中谓词函数含有非线性表达式时, 只需计算非线性谓词函数的线性算术表示, 如例 2, 迭代 3 次即求得期望的输入, 而同样的路径和初始输入, 文献[10]中的算法却需要迭代 4 次, 所以新的算法要明显优于文献[10, 11]中的算法, 具有运行程序步骤少、计算量小, 求解速度快等特点.

4 结 论

本文通过对基于路径测试数据自动生成的几种方法进行分析, 指出了各种方法的特点及存在的不足, 进而在此基础上提出了一种新的测试数据自动生成的方法. 文中从理论上对新算法的主要思想进行了详细的分析, 提出了理论依据, 给出了相应的证明, 并用实例进行了验证. 实例表明, 该方法不必用均差去近似导数计算线性谓词函数的线性算术表示, 也不需构造谓词函数关于输入变量增量的线性约束, 对于线性约束路径, 通过一次迭代即可求出测试数据, 或确保路径不可达; 对于含有非线性约束的路径, 通过多次迭代亦可求出测试数据, 或在相当程度上能保证路径不可达.

参考文献(References)

[1] 单锦辉, 王戟, 齐治昌. 面向路径的测试数据自动生成方法述评[J]. *电子学报*, 2004, 39(1): 109-113.
(Shan J H, Wang J, Qi Z C. Survey on Path-wise Automatic Generation of Test Data [J]. *Acta Electronica Sinica*, 2004, 39(1): 109-113.)

[2] Clarke L A. A System to Generate Test Data and Symbolically Execute Programs [J]. *IEEE Trans on Software Engineering*, 1976, 16(8): 870-879.

[3] DeMillo R, Offutt J. Constraint-based Automatic Test Data Generation [J]. *IEEE Trans on Software Engineering*, 1991, 17(9): 900-910.

[4] Alberto Coen-Porisini, Giovanni Denaro, Carlo Ghezzi, et al. Using Symbolic Execution for Verifying Safety-critical Systems [J]. *ACM SIGSOFT Software Engineering Notes*, 2001, 26(5): 142-151.

[5] Chen T Y, Tse T H, Zhou Z Q. Semi-proving: An Integrated Method Based on Global Symbolic Evaluation and Metamorphic Testing [J]. *ACM SIGSOFT Software Engineering Notes*, 2002, 27(4): 191-195.

[6] Willem Visser, Corina S Păsăreanu, Sarfraz Khurshid. Test Input Generation with Java PathFinder[J]. *ACM SIGSOFT Software Engineering Notes*, 2004, 29(4): 97-107.

[7] Gallagher M J, Narasimhan VL. ADTEST: A Test Data Generation Suite for Ada Software Systems [J]. *IEEE Trans on Software Engineering*, 1997, 23(8): 473-484.

[8] Korel B. Automated Software Test Data Generation [J]. *IEEE Trans on Software Engineering*, 1990, 16(8): 870-879.

[9] Nguyen Tran Sy, Yves Deville. Consistency Techniques for Interprocedural Test Data Generation [J]. *ACM SIGSOFT Software Engineering Notes*, 2003, 28(5): 108-117.

[10] Gupta N, Mathur A P, Soffa M L. Automated Test Data Generation Using an Iterative Relaxation Method [A]. *Proc of the 6th ACM SIGSOFT Int Symposium on Foundations of Software Engineering* [C]. Florida, 1998: 231-244.

[11] 单锦辉. *面向路径的测试数据自动生成方法研究*[D]. 长沙: 国防科学技术大学研究生院, 2002.
(Shan J H. *On Path-wise Automated Test Data Generation* [D]. Changsha: National University of Defense Technology, 2002.)

[12] Wegener J, Baresel A, Sthamer H. Evolutionary Test Environment for Automatic Structural Testing [J]. *Information and Software Technology*, 2001, 43(14): 843-854.

[13] Edvardsson J, Kamkar M. Analysis of the Constraint Solver in UNA Based Test Data Generation [J]. *ACM SIGSOFT Software Engineering Notes*, 2001, 26(5): 237-250.

[14] Gupta N, Cho Y J, Hossain M Z. Experiments with UNA for Solving Linear Constraints in Real Variables [A]. *Proc of the 2004 ACM Symposium on Applied Computing* [C]. Nicosia, 2004: 1013-1020.