# 非定态路径测试问题的分析与一种转换算法

缪 力[1], 张大方[1], 季 洁[1], 宣恒农[2]
(1. 湖南大学计算机与通信学院, 湖南长沙 410082; 2. 南京财经大学信息工程学院, 江苏南京 210003)

摘 要: 测试数据生成中使用静态分析法的主要问题之一是难以处理程序变量的不确定性. 本文对软件测试数据生成中的变量/不确定0问题进行了分析, 认为该问题的实质是程序中变量本身的符号不确定性和程序的动态性, 根据这个思路提出非定态路径、变符号变量、程序状态变量等概念, 并将问题进行了形式化的描述, 证明了非定态路径约束解空间可进一步划分为多个子空间. 在此结论的基础上, 本文给出了一个将非定态路径测试转换为定态路径测试的算法, 对非定态路径测试问题的理论分析和解决途径进行了有益的尝试.

关键词: 软件测试; 非定态路径; 变符号变量; 数组和指针
中图分类号: TP311. 5 文献标识码: A 文章编号: 0372-2112 (2005) 02-0258-04

# Formal Analysis of Nondeterminism Path Test Problem and a Transform Algorithm

MIAO Li[1], ZHANG Da-fang[1], JI jie[1], XUAN Hen-nong[2]
(1. College of Computer and Communication, Hunan University, Changsha, Hunan 410082, China;
2. School of Information Engineering, Nanjing University of Economics, Nanjing, Jiangsu 210003, China)

Abstract: In the field of automatic test data generation, a difficult problem is the nondeterminism of program variable, e. g., array problem and pointer problem. Array variants and pointer variants are defined as a special kind of variants, variants with variable symbol, and a path with variants with variable symbol is defined as a nondeterministic path. Generating testing data of a path is finding an input that satisfies the path constraint condition. By proving the input space satisfying the constraint of a nondeterministic path can be further partitioned into many sub-spaces, the nondeterminism is avoided when test data is searched only in a sub-space in theory. An algorithm is presented to find a sub-space by program slicing.

Key words: software testing; nondeterministic path; variant with variable symbol; array and pointer

## 1 引言

在整个软件开发过程中, 软件测试通常要占到开发成本的50%甚至更多[1]. 为了减少如此巨大的测试费用, 人们希望能实现软件测试的自动化. 软件测试自动化的核心问题之一是测试数据自动生成.

静态分析方法, 采用直接分析程序的源代码而不执行程序的形式, 是测试数据生成技术中的重要方法, 然而实现上的一些困难使这种方法难以实用, 其中程序变量的不确定性问题, 有时又根据情况不同称为数组下标问题、指针问题等等, 是静态分析实现测试数据生成中的一个较难处理的困难问题[2~5], 也是静态分析难以实用的重要原因之一[2~4]. 为便于下面的讨论, 本文将程序变量的不确定性问题定义为非定态路径测试问题. 动态分析虽然利用程序的执行信息而避免了直接的变量不确定问题, 但该问题却在另一方面对动态分析的性能造成影响, 例如: 对于数组下标与输入相关且数组也是输入变量的情况, 假定通过动态分析确定了一组满足非定态路径前 k 个谓词约束(部分约束中包括该数组变量)的输入, 如果与数组下标相关的变量不满足第 k+1 个谓词约束, 此时对输入变量的调整使程序状态变量改变, 显然可能破坏已经满足的谓词约束, 从而导致大量的重复搜索, 调整搜索次序的方法[2]只能减少这个问题发生的次数.

本文引入变符号变量、程序状态变量、非定态路径的概念从另一个角度该问题进行描述, 通过理论分析得出非定态路径约束解空间可进一步划分为多个子空间的结论. 并基于分析结论给出一个将非定态路径的测试转换为定态路径测试的算法.

## 2 基本概念

我们以数组下标问题来简要说明这个程序不确定性问题(其实指针的问题也是一样). 如一程序中有如下语句:

```
, ,
a[i]:= p;
, ,
q:= a[j];
ifp= q then
, ,
```

显然, 若 i= j 则p= q, 否则 p= a[j], 对静态分析而言, 此时已经无法分辨这两个变量 a[i]和 a[j]的关系. 目前解决方法主要是采用为数组及相关变量设置虚拟变量的方法[6],

收稿日期: 2003-08-04; 修回日期: 2004-11-03
基金项目: 国家自然科学基金(No. 60473031, No. 60273070); 江苏省自然科学基金(No. BK2004119)

这正是看到了这样一个问题: 变量本身的符号在程序中也会变化. 但为数组相关变量设置虚拟变量使静态分析中的方程和变量的规模迅速扩大, 这种解决方案并不令人满意[2~5].

定义 1　输入变量 X

输入变量 X= (x1, x2, , , xn) 是矢量, 对应于程序输入参数表.

定义 2　程序定义域空间 D

程序输入变量各维分量取值范围的笛卡尔乘积, 构成其定义域空间 D.

即 D= Dx1@Dx2@, @Dxn= {(x1, x2, , , xn) | P x1 I Dx1, P x2 I Dx2, , , P xn I Dxn}.

其中 xi( i= 1, 2, n) 是程序的输入变量第 i 维分量, Dxi 是xi 所有可能的取值.

定义3　程序控制流图CFG 是一种有向图, 记为 G= 3N, E, s, e4, 其中N 是节点集, 表示程序中所有语句, 每个语句对应图中的一个节点; E 是边集, E= {( ni, nj) | ni, nj I N 且语句ni 执行后, 可能立即执行 nj; s 与 e 是N 中两个特殊节点, 称为初始节点与终止节点, 从 s 有指向程序第一个语句节点的边, 程序的每个出口节点有指向 e 的边. 对于无循环的情况, 本文中的程序路径 p 是一条从s 到e 的无环路径, 存在循环的情况下, 程序路径 p 通过将 CFG 中对应路径的循环展开, 并将每次循环体内的节点加以表示区别得到.

在本文中, 需要区分变量的名称( 变量符号) 和变量的取值, 对程序中任意变量 x, 我们用 S( x) 表示程序中代表变量的变量符号的集合. 对 CFG 中任意节点 ni, S( ni) 表示节点 ni 中代表语句中所有变量的变量符号的集合, ni| x 表示当输入为x 时节点ni 各变量取相应的赋值, vni| x 表示当输入为x 时节点ni 的变量v 取相应的赋值.

定义 4　变符号变量

程序中某个变量 v, 若有

(1) v x v xc v niBni I px C ni I pic C S( v) A S( ni) ] S( vni| x) X S( vni| xc) 或

(2) v x v ni v njBni I p C ni I p C S( v) A S( ni) C S( v) A S( nj) ] S( vni| x) X S( vnj| x) 有一个成立, 则称变量 v 为变符号变量, 其中 x, xc 为程序输入, px, pcx 为对应程序执行路径.

定义 5　符号控制变量

对程序中任意某个变量 v, 我们知道, 对变量 v 的赋值语句并不能改变变量v 的符号S( v), 变符号变量之所以能改变符号是因为程序中存在某个( 或某一组) 变量 c, 使得在程序中的变符号变量的符号发生变化, 称变量 c 为符号控制变量.

即: v cv cc: cX cc ] S( v( c) ) X S( v( cc) ) .

例如: 一个数组变量是变符号变量, 其下标是对应的符号控制变量; 一个指针所指的对象是变符号变量, 其指针变量是对应的符号控制变量;

定义 6　程序状态变量 Y

程序状态变量 Yx= (y1, y2, , , ym) 是矢量, 当程序输入为 x 时, 若对应程序路径为 px, 程序状态变量 Yx 对应于程序运行中用到的所有不同变量. 对于变符号变量, S( Yx) 包括了当程序输入为 x 时其所有可能的变量符号. 我们可用公式将 S( Yx) 表示为:　　S( Yx) = G S( i| x)

状态变量的引入, 是为了强调分析运行中的程序与分析静态程序源代码的区别与联系, 通过程序状态变量, 我们将静态的程序中定义的变量与确定输入运行中的程序联系起来.

定义 7　可控路径约束 F ( 以下简称路径约束)

对于确定的程序路径 p, 通过符号执行技术, 如某个路径分支谓词可相应地转换为以输入变量为表达式的等式或不等式方程约束: f, 我们称该约束为可控谓词约束( 以下简称谓词约束) . p 上所有谓词约束形成p 的路径约束 Fp= {f1, f2, , , fp}. 不失一般性, 本文中仅讨论简单谓词约束, 但结果不难扩展到复杂谓词约束的情况.

定义 8　路径约束解空间

对于确定的程序路径 p, 满足路径约束 Fp 的所有输入X 构成程序定义域空间D 上的一个子空间, 称为路径 p 的路径约束解空间 Dp.

定义 9　非定态路径

对于程序中确定的路径 p, 如果 x1, x2 为程序输入I Dp, Y1, Y2 为 x1, x2 对应的路径 p 程序状态变量, S( Y1), S( Y2) 为 Y1, Y2 对应的状态变量符号集, 若

v x1v x2(x1 X x2) ] (S( Y1) X S( Y2))

则称路径 p 为一条非定态路径.

定义 10　定态路径

对于程序中确定的路径 p, 如果任意 x1, x2 为程序输入 I Dp, Y1, Y2 为 x1, x2 对应的程序状态变量, S( Y1) , S( Y2) 为 Y1, Y2 对应的状态变量符号集, 若有　S( Y1)= S( Y2)

则称路径 p 为一条定态路径.

定理 1　非定态路径必定存在有使用变符号变量指令.

证明: ( 略) .

## 3　非定态路径测试问题的研究

利用符号执行技术将程序的所有路径约束转化为有关输入的符号的等式或不等式方程组, 则所谓测试数据自动生成的问题其实也只是一个约束满足问题. 满足不同路径的输入将程序定义域空间划分为许多不同的子空间, 也就可以认为测试数据自动生成的过程就是一个基于不同路径约束划分程序定义域空间的过程. 然而, 对于非定态路径, 这种划分过程遇到困难.

因为非定态路径的路径约束解空间当然是程序定义域空间的一个子空间, 但其本身还可以进一步划分为多个同构的子空间. 确定非定态路径的路径约束解空间的过程隐含了一个对非定态路径的程序状态变量符号集的选取过程.

定理 2　对一条非定态路径, 若所有满足该路径约束的输入可得到个 k 不同程序状态变量符号集, 则这 k 个程序状态变量符号集可以将该路径约束解空间进一步划分为k 个子空间. 证明: ( 略) .

由定理 2, 可以看出, 一组约束得到解空间的只是一个解空间, 非定态路径的路径约束解空间却有多个子空间, 也就是

说, 非定态路径的路径约束应该是多个约束方程组. 同时根据定理 2, 我们得到一个解决非定态路径问题的想法: 尽管非定态路径的路径约束解空间的输入会产生不同的状态变量符号集, 有多个约束方程组和多个同样满足路径约束的子空间, 但一个子空间只能对应一个程序状态变量符号集. 这样的一个子空间性质与一个定态路径的路径约束解空间相同.

根据以上对非定态路径的分析, 可以想到, 要将对非定态路径的测试数据生成转换为对定态路径的测试数据生成, 关键在于把测试数据生成的范围缩小到只能对应一个程序状态变量符号集的路径约束解空间的子空间. 问题转变成怎样在还不知道路径约束解的情况下找到路径约束解空间子空间.

由定理 1, 我们知道, 使用变符号变量的指令的存在是产生非定态路径的必要条件. 但是, 使用变符号变量的指令仍然只能改变变符号变量的赋值, 并不能改变变符号变量的符号本身. 变符号变量的符号能改变, 除了其本身具备可变的条件外, 引起其产生变化的外因是符号控制变量的变化. 这样, 我们只要使符号控制变量取一确定值就能确定变符号变量. 但符号控制变量的取值也必须满足路径约束.

## 4 转换算法

本节主要讨论非定态路径测试数据生成转换为对定态路径测试数据生成的算法步骤. 算法中采用了数据流分析和程序切片技术, 详细技术细节见文献[7~ 9]. 但有一点要注意的是: 静态分析与程序的动态特性的矛盾同样存在于程序切片技术中, 如文献[7]在定义变量的引用变量集和涉及变量集时, 强制定义了数组 a[ i]和 i 的引用关系(指针、引用类型的情况类似), 又如文献[9]定义数组 a[ i]和 i 是数据依赖关系; 但从本文的分析中看到: a[ i]是一个变符号变量, i 是其相应的符号控制变量, 所以, a[ i]和 i 之间并不是直接有一般意义上的变量相关关系, 而是符号变化的控制关系.

当我们考虑程序的动态特性时, 对一个有确定输入的程序来说, 程序的执行路径就已经确定了, 因此, (谓词)控制依赖的意义在这里也要重新考虑.

假定我们基于变量集 A 进行程序切片, 所有与 A 有数据依赖关系的节点构成集合$N_A$,

PC 关系: 谓词( predicate) ) ) ) 控制( control) 关系[7]

K PC J: 结点 J 在谓词 K 的直接影响范围内, 即 K 对 J 有最直接的控制行为.

主要针对: If K then B1 else B2 (1) (J I B1D J I B2)

While K do B (2) (J I B)

对情况(1), 若有(K| $N_A$) C (J I $N_A$), 静态切片技术认为, 只有当 K 确定, 程序才能执行到 J, 所以, K 也应包含在程序切片的结果中. 但对一个有确定输入 X 的程序来说, 程序的执行路径就已经确定了, 若 B1 在该路径中, 其隐含的意义是, 对于 X, K 已经成立. 因此, 即使不考虑 K, 对于 X, 程序切片结果依然成立.

对情况(2), 若程序的执行路径已经确定, 则我们可以将循环展开成多个(1)的组合. 所以, 若有(K| $N_A$) C (J I $N_A$), 由(1)的讨论可知, 即使不考虑 K, 对于 X, 程序切片结果依然成立, 当然还需要进行循环展开的预处理.

根据上面的讨论, 我们对下面算法用到的程序切片技术作如下改动: (1) 不考虑控制依赖关系; (2) 不强制定义控制符号变量和变符号变量的关系

同时, 我们还对程序有如下限制:

若对于路径 p, 控制符号变量的引用变量集和涉及变量集为 $U_C$, 变符号变量的引用变量集和涉及变量集为 $U_E$, S( $U_C$) H S( $U_V$)= ª. 从编程经验就可以知道, 这个条件对大部分程序都是满足的.

算法实现步骤:

第一步: 找到路径 p 中的所有变符号变量 E 以及对应的控制符号变量 C.

第二步: 利用数据流分析技术确定 $U_C$, 确定与 $U_C$ 对应的输入变量 X 的各分维变量组成的新输入变量 $X_C$, 不失一般性, 我们设 $X_C$= ( $x_i$, , , $x_j$) 对应原程序输入变量的第 i 维分量到第 j 维分量.

第三步: 基于路径 p 和 $U_C$ 将程序展开. 即除了使用 $U_C$ 变量的指令外, 我们假定其他变量都已经满足路径 p 的路径约束, 从而解开循环, 消除其他路径指令, 这样得到新程序 +.

第四步: 基于得到的新输入变量 $X_C$ 对程序 + 采用程序切片技术, 得到新程序 $.

第五步: 采用静态或动态分析的测试技术, 凭经验可知, 基于控制符号变量得到的新程序不会复杂, 得到满足约束的一组解( $a_i$, , $a_j$).

第六步: 原程序的输入变量定为( $x_1$, , , $a_i$, , , $a_j$, , , $x_n$), 即程序定义域空间 D 变为: $D_{new}$= $Dx_1$@$Dx_2$@, @$Dx_{i-1}$ @$a_i$@, @$a_j$ @$Dx_2$@, @$Dx_{j+1}$= {( $x_1$, $x_2$, , , $x_n$)| P $x_1$ I $Dx_1$, P $x_2$ I $Dx_2$, , , P $x_i$= $a_i$, , , P $x_j$= $a_j$, , , P $x_n$ I $Dx_n$}. 其中 $x_i$( i= 1, 2, n) 是程序的输入变量第 i 维分量, $Dx_i$ 是 $x_i$ 所有可能的取值. 此时基于新程序定义域空间 $D_{new}$的非定态路径的测试已转换为定态路径测试.

算法的主要分析耗时在与程序切片, 算法的最坏时间复杂度为 O( $n^2$), 其中 n 为该路径的节点数.

## 5 举例

文献[2]中的程序是一个典型的例子, 尽管很简单, 而且实现的是一个常见的功能, 却造成了静态分析的困难, 我们同样也用这个程序为例来解释我们的算法思想, 同时也便于通过比较指出我们的算法即使是对于动态分析也有优化搜索次数的效果. 程序如下:

设我们要测试的程序路径 p 为3 1, 2, 3, 4, 5, 6, 8, 10, 5, 6, 8, 9, 10, 5, 114, 显然 p 是一条非定态路径.

第一步, 确定变符号变量, 显然, 程序中的变符号变量是 a[ ]; 确定对应的控制符号变量, 此程序中对应的变量为 i.

第二步, 找到 $U_C$, 此程序中为 low, high, step, i; 找到 $X_C$, 此程序中为 low, high, step.

第三步, 基于路径 p 和 $U_C$ 将程序展开为新程序 +.

第四步: 基于得到的新输入变量 $X_C$ 对程序 + 采用程序切片技术, 得到新程序 $ 如下:

第五步: 采用静态或动态分析的测试技术, 得到满足 p 路径约束的一组 low, high, step 输入解. 一般情况下, 会有很多组, 我们可以任取一组, 如 low = 17, high= 45, step = 12.

第六步, 重新确定输入向量为 (17, 42, 12, a), 基于此输入变量对 p 的路径测试已被转换为定态路径测试.

```
var
a: array[1...000] of integer;
low, high, step: integer;
min, max: integer;
i: integer;
        begin
1       input(int low, int high, int step, int a[])
2        max:= a[low];
3        min:= a[low];
4        i= low+ step;
5        while i< high do
           begin
6, 7         if max< a[i] then max:= a[i]
8, 9         if min> a[i] then min:= a[i]
10           i+ i+ step
             end;
11         output(min, max);
        end;
```

此时, 对于静态分析而言, 采用混合符号执行技术可以得到 a 在路径的各个节点中确定的意义, 以下可以直接使用符号执行或区间算术的方法得到测试数据. 不仅如此, 对动态分析而言, 我们的转换算法也很有意义, 例如: 对于符号控制变量与输入相关且变符号变量是输入变量的情况(类似上例), 假定通过动态分析确定了一组满足非定态路径前 k 个谓词约束(部分约束中包括变符号变量) 的输入, 如果与符号控制变量相关的变量不满足第 k+ 1 个谓词约束, 此时对输入变量的调整使程序状态变量改变, 显然可能破坏已经满足的谓词约束, 从而导致大量的重复搜索, 调整搜索次序的方法[3] 只能减轻这个问题, 而用上面的转换算法预处理后, 可以看到因改变程序状态变量导致路径约束冲突的情况已经避免.

```
var
low, high, step: integer;
i: integer;
        begin
          input(int low, int high, int step)
            i= low+ step;
            while i< high do
          begin
            i= i+ step
          end;
        end;
新程序 $
```

## 6　结论

本文对软件测试数据生成中的变量/ 不确定0 问题进行了分析, 针对该问题提出非定态路径、变符号变量、程序状态变量等概念将该问题进行了形式化的描述, 在此基础上证明了非定态路径约束解空间实际上还可进一步划分为多个同样满足路径约束的子空间, 基于证明的结论我们提出把测试数据生成的范围缩小到其中一个子空间以解决非定态路径测试问题的思想, 通过对程序切片技术的改进给出一个将非定态路径测试转换为定态路径测试的算法. 目前大部分数据自动生成方法都是基于单元测试, 我们的思想方法也只是针对单元测试而言, 我们下一步的工作将考虑如何将该方法用于对有多个过程的结构化程序以及基于面向对象的程序自动生成测试数据.

参考文献:


[1] B Beizer. Software Testing Techniques[M]. 2nd edition, Van Nostrand Reinhold, 1990.

[2] Bogdan Korel. Automated software test data generation[J]. IEEE Trans on Software Eng, 1990, 16(8): 870- 879.

[3] N Gupta, A P Mathur, M L Soa. Automated test data generation using an iterative relaxation method[A]. Proceedings of the ACM SIGSOFT Sixth International Symposium on Foundations of Software Engineering [C]. Orlando, USA: ACM Press, November 1998. 231- 244.

[4] 奚红宇, 徐红, 高仲仪. Ada 软件测试用例生成工具[J], 软件学报, 1997, 8(4): 297- 302.
XI Hong2yu, XU Hong, GAO Zhong2yi. ADA software test case generation tool[J]. Journal of Software, 1997, 8(4): 297- 302(in Chinese).

[5] J Edvardsson. A survey on automatic test data generation[A]. the Second Conference on Computer Science and Engineering in Link Ping [C]. CCSSE. 99, ProNova, Norrkoping: ECSEL Press, October, 1999. 21- 28.

[6] C V Ramamoorthy, S F Ho, W T Chen. Automated generation of program test data[J]. IEEE Transactions on Software Engineering, 1976, SE22(4): 293- 300.

[7] 赵瑞莲, 闵应骅. 一种基于数据流分析的程序定义域自动确定方法[J]. 计算机辅助设计与图形学学报, 2001, 13(8): 7622768.
Ruilian Zhao, Yinghua Min. A method of program domain automated determination based on data flow analysis[J]. Journal of Computer2Aided Design & Computer Graphics, 2001, 13(8): 7622768(in Chinese).

[8] Sandra Rapps, Elaine J Weyuker. Selecting software teat data using data flow information[J]. IEEE Trans on Software Eng, 1985, SE211(4): 367- 374.

[9] Tip F. A survey of program slicing techniques[J]. Journal of Programming Languages, 1995, 3(3): 121- 189.


作者简介:

缪　力　男, 1972 年出生于湖南长沙, 博士研究生, 感兴趣的领域为软件测试与程序验证. E2mail: miaoli_ 2000@163. com.

张大方　男, 1959 年出生于上海, 教授, 博士, 博士生导师, 感兴趣的研究领域为容错计算, 软件测试与软件可靠性, 网络测试. E2mail: dfzhang@hnu. cn.