文章编号:1671-6833(2009)02-0084-04

一种简化依赖关系的动态程序切片算法

贾利敏,张忠林

(兰州交通大学 电子与信息工程学院,甘肃 兰州 730070)

摘 要:动态程序切片是由影响程序中某个兴趣点处变量值的所有语句和控制谓词组成的一个程序子集.笔者提出了一种简化依赖关系的动态程序切片算法,主要目的是解决H. Agrawal的动态切片算法计算切片不太精确的问题,该算法通过程序执行轨迹,确定数据依赖结点、控制依赖结点和结点可到达语句来计算变量切片.实例证明,该算法提高了动态切片的精确度,减少了计算动态程序切片的复杂度.

关键词:动态程序切片;切片准则;动态程序切片算法;依赖关系;程序依赖图

中图分类号:TP 311.5 文献标识码:A

0 引言

程序切片是一种分析和理解程序的技术,它通过程序中可能影响某处变量值的所有语句来分析理解程序.1979年,Mark Weiser首次提出程序切片的概念:程序P的切片slice是P的一个可执行部分,对于某个兴趣点v处的s变量而言,这个可执行部分相对于程序P,在功能上是等效的[1].这是一种静态程序切片,它考虑了程序中与感兴趣点处变量相关的所有语句,分析了程序所有可能的执行路径,容易包含不相关结点,工作量较大[2].

基于静态切片的不足,Korel等人提出了动态程序切片,动态切片是在某一具体输入下,程序P实际执行影响某个兴趣点s处变量v的所有语句构成[3-5].动态切片只关注程序在特定输入下,对最终结果有影响的语句,从而简化了切片.因此,吸引了很多人研究动态程序切片,并取得了一定的发展,具有代表性的是H. Agrawal的动态切片[6].

H. Agrawal的动态切片是扩展基于程序依赖图的静态切片方法得到的[7],其算法思想是程序在执行期间会产生相应的依赖关系,根据依赖关系画出程序依赖图,从程序依赖图中找到感兴趣的结点,沿着该结点的依赖边进行遍历,遍历经过的所有结点,由这些结点对应的语句所构成的集合就形成了切片.这种方法虽然简单高效,但产生的动态切片较大,且对于有循环的程序,产生的动态切片不太精确.笔者拟提出一种简化依赖关系的动态程序切片算法,以提高动态切片的精确度,减少生成动态切片的复杂度.

1 程序切片的基本概念与定义

动态切片是由影响程序中某个兴趣点处变量值的所有语句和控制谓词组成的一个程序子集.其表示方法:$C=\langle x,s^k,v\rangle$,x为输入数据子集,s为程序中的一条语句,s^k表示程序在第k步执行语句s,v是一个(组)程序变量.

定义1 程序控制流图$CFG=(N,E,s,e)$,其中N是结点集,E是边集,s是控制流图唯一入口,即程序的开始语句,e是控制流图唯一出口,即程序的终止语句.程序从入口结点s到结点n_k的路径是结点序列$\langle n_1,n_2,n_3,\cdots,n_k\rangle$,其中$n_1=s$,$\langle n_i,n_{i+1}\rangle\in E,1\leqslant i\leqslant k$.

定义2 定义变量.如果变量v在程序n处被赋值,则称变量v在n处被定义.$n\in N,v\in V$,以$Def(n)$表示n处定义变量的集合.

定义3 使用变量.如果变量v在程序中n处被引用,则称变量v在n处被使用.$n\in N,v\in V$,以$Use(n)$表示n处使用变量的集合.

定义4 数据依赖关系(DDR).如果存在一个变量v在n点被定义在m处被使用,并且在控制流

收稿日期:2008-11-15;修订日期:2009-01-20

基金项目:甘肃省科技支撑计划项目(0804JKCA040)

作者简介:贾利敏(1981-),女,河北邯郸人,甘肃交通大学硕士研究生,主要研究方向为软件工程、软件测试.通讯作者:张忠林(1965-),男,河北衡水人,甘肃交通大学教授,主要研究方向为数据挖掘、软件测试.

图中存在一条从结点 n 到 m 的路径 p，v 在此路径除 n 点处未被重新定义，则称 m 数据依赖于 n.

定义 5　控制依赖关系(CDR). 如果存在一条从 n 到 m 的路径 p，对路径 p 上除 n、m 之外的其它结点 t，m 是 t 的后必经结点，m 不是 n 的后必经结点，则称 m 控制依赖于 n.

数据依赖关系和控制依赖关系统称为依赖关系，如果 m 依赖于 n，记作 $n = dep(m)$.

定义 6　可到达语句. 设 s^i, s^j 是程序 P 中的两个语句，$s^i, s^j \in N$，如果语句 s^i 依赖于 s^j，则称语句 s^i 可到达语句 s^j，记作 Reachstates(s^i) = $\{s^j\}$，如果 Reachstates(s^i) = $\{s^j\}$，Reachstates(s^j) = $\{s^k\}$，$s^i, s^j, s^k \in N$，则 Reachstates(s^i) = $\{s^j, s^k\}$.

定义 7　循环依赖关系. 设程序 P 是一个带有循环的程序，s 是程序 P 中的一个语句，如果语句 s 在程序 P 的 n 次循环执行中分别为 $s^1, s^2, \cdots, s^{n-1}, s^n$，则称语句 s^n 依赖于语句 s^{n-1}，记作 s^{n-1} = cirdepend(s^n)，$s^1, s^2, \cdots, s^{n-1}, s^n \in N$.

定义 8　程序执行轨迹. 在某特定输入下，程序实际执行时所经过的结点组成程序执行轨迹.

定义 9　直接前驱. 设 s^n, s^m 是程序 P 执行轨迹中的两个语句，$s^n, s^m \in N$，如果 s^m 是 s^n 的前一个被执行的语句，则称语句 s^m 是语句 s^n 的直接前驱语句，记作 s^m = front(s^n).

2　动态程序切片生成算法

笔者改进 H. Agrawal 的扩展基于程序依赖图的静态切片方法得到的动态切片，通过程序执行轨迹，确定数据依赖结点、控制依赖结点和结点可到达语句，使得程序语句关系明确，提高了动态程序切片的精度.

确定切片准则 C 并指定一条程序路径，根据程序执行轨迹上的数据依赖结点、控制依赖结点和结点可到达语句计算变量 v 的切片. 以 Reachstates(s^k) 记录当前执行位置 k 处语句 s 的可到达语句. 分两种情况进行讨论，当 s 无循环依赖关系时，计算 Reachstates(s^k)，如果变量 v 在语句 s 被使用，则 slice(v) = Reachstates(DDNode(s^k))；当 s 有循环依赖关系时，计算 Reachstates(s^k)，如果 s^i 循环依赖于 s^j，并且 s^i 的 Reachstates 与 s^j 的 Reachstates 相等，那么合并结点 s^i, s^j，合并它们的 Reachstates；如果 s 前驱结点 Reachstates 与 s 的 Reachstates 相等，则合并结点 s 和 s 的前驱结点，合并它们的 Reachstates. 变量 v 在语句 s 被使用，则 slice(v) = Reachstates(DDNode(s^k)).

动态程序切片算法描述：

```
Input:数据依赖结点(DDNode)、控制依赖结点
(CDNode)和切片准则 C = ⟨x, s^q, v⟩
Output:在执行位置 q 处的变量 v 的动态切片
初始化:For all v ∈ V do slice(v) = Φ
For all s^k ∈ P do Reachstates(s^k) = {s}
执行每个 s^k 后(k < = q)
Reachstates(s^k) = Reachstates(s^k) ∪
DDNode(s^k) ∪ CDNode(s^k)
If s 无循环依赖关系 then
{
Reachstates(s^k) = Reachstates(s^k) ∪
Reachstates(Dep(s))
For v ∈ V do
If v ∈ Use(s) then
slice(v) = Reachstates(DDNode(s^k))
endfor
}
else
Reachstates(s^k) = Reachstates(s^k) ∪ Reachstates
(Dep(s))
If s^i = cirdepend(s^j) and
Reachstates(s^i) = = Reachstates(cirdepend(s^j))
then s^i = s^i ∪ cirdepend(s^j);//修正
else
If Reachstates(front(s^i)) = = Reachstates(s^i)
then front(s^i) = front(s^i) ∪ s^i;//修正
For v ∈ V do
If v ∈ Use(s) then
slice(v) = Reachstates(DDNode(s^k))
endfor
```

k = q 时，算法停止.

3　实例分析

下面以程序 P 为例，说明采用上述动态切片算法计算动态切片的过程.

```
程序 P
1 read (n);
2 i = 1;
3 while (i < = n) {
4 read (x);
5 if (x < 0) then
6 y = f1(x);
7 else y = f2(x)
endif
```

```
8 z=y;
9 write (z);
10 i=i+1;
}
```

假设切片准则 $C=\langle x,9^{22},v\rangle$,其中输入为:$n=3,x=\{-1,2,-3\}$,$V=\{z\}$,在程序 P 输入 $n=3,x=-1,2,-3$ 时,程序实际执行轨迹为 $Tath=<<1^1,2^2,3^3,4^4,5^5,6^6,8^7,9^8,10^9,3^{10},4^{11},5^{12},7^{13},8^{14},9^{15},10^{16},3^{17},4^{18},5^{19},6^{20},8^{21},9^{22},10^{23},3^{24},>>$

根据程序 P 的切片准则 C 和执行路径,得到程序 P 的数据依赖结点、控制依赖结点,如表 1 所示.

表 1 程序 P 的依赖结点

Tab.1 Dependence nodes of program P

结点	数据依赖结点	控制依赖结点	结点	数据依赖结点	控制依赖结点
1^1	Φ	Φ	7^{13}	4^{11}	5^{12}
2^2	Φ	Φ	8^{14}	7^{13}	3^{10}
3^3	$1^1,2^2$	Φ	9^{15}	8^{14}	3^{10}
4^4	Φ	3^3	10^{16}	10^9	3^{10}
5^5	4^4	3^3	3^{17}	$1^1,10^{16}$	Φ
6^6	4^4	5^5	4^{18}	Φ	3^{17}
8^7	6^6	3^3	5^{19}	4^{18}	3^{17}
9^8	8^7	3^3	6^{20}	4^{18}	5^{19}
10^9	2^2	3^3	8^{21}	6^{20}	3^{17}
3^{10}	$1^1,10^9$	Φ	9^{22}	8^{21}	3^{17}
4^{11}	Φ	3^{10}	10^{23}	10^{16}	3^{17}
5^{12}	4^{11}	3^{10}	3^{24}	$1^1,10^{23}$	Φ

程序 P 运行过程中可到达结点如表 2 所示.

在程序 P 的执行轨迹中存在循环依赖关系,根据语句 s 是否有循环依赖结点分别讨论如何计算切片:

(1)s 无循环依赖关系,即 while 中的谓词条件,在循环次数为 0 时,语句 1,……,语句 6,语句 8,语句 9,语句 10 是无循环依赖结点的.

这种情况下,得到变量 v 的切片相对比较容易.如果 v 在语句 s 处被使用,语句 s 数据依赖结点的可到达语句值就是变量 v 的切片,如 9^8 处 z 的切片 Slice(z)是语句 9^8 数据依赖结点 8^7 的可到达结点值.

(2)s 有循环依赖关系,即 while 语句体执行次数大于 1 时,语句 3,……,语句 10 则存在循环依赖结点.

在这种情况下,如果语句 s^n 循环依赖于程序前一次执行的 s^{n-1} 语句,且它们的可到达结点相等,则对语句做修正,例如语句 $4^{11},4^{18}$ 可以合并;如果语句 s^n 和其前驱语句 s^m 的可到达语句值相等,那么修正语句,例如合并语句 $10^9,3^{10},10^{16},3^{17},10^{23},3^{24}$.构建程序 P 的可到达结点表后,要获得变量 v 的动态切片,首先要得到使用变量 v 的语句 s,然后找到 s 的数据依赖结点,与该结点相对应的可达到语句就是变量 v 的切片.

程序 P 变量 z 的动态切片如表 3 所示.

表 2 程序 P 的可到达结点

Tab.2 Reachable node of program P

可到达结点	初始化	计算切片前可到达结点的值
Reachstates(1^1)	1	1
Reachstates(2^2)	2	2
Reachstates(3^3)	3	1,2,3
Reachstates(4^4)	4	1,2,3,4
Reachstates(5^5)	5	1,2,3,4,5
Reachstates(6^6)	6	1,2,3,4,5,6
Reachstates(8^7)	8	1,2,3,4,5,6,8
Reachstates(9^8)	9	1,2,3,4,5,6,8,9
Reachstates(10^9)	10	1,2,3,10
Reachstates(3^{10})	3	1,2,3,10
Reachstates(4^{11})	4	1,2,3,4,10
Reachstates(5^{12})	5	1,2,3,4,5,10
Reachstates(7^{13})	7	1,2,3,4,5,7,10
Reachstates(8^{14})	8	1,2,3,4,5,7,8,10
Reachstates(9^{15})	9	1,2,3,4,5,7,8,9,10
Reachstates(10^{16})	10	1,2,3,10
Reachstates(3^{17})	3	1,2,3,10
Reachstates(4^{18})	4	1,2,3,4,10
Reachstates(5^{19})	5	1,2,3,4,5,10
Reachstates(6^{20})	6	1,2,3,4,5,6,10
Reachstates(8^{21})	8	1,2,3,4,5,6,8,10
Reachstates(9^{22})	9	1,2,3,4,5,6,8,9,10
Reachstates(10^{23})	10	1,2,3,10
Reachstates(3^{24})	3	1,2,3,10

表 3 程序 P 的切片结果

Tab.3 Slice result of program P

结点	Slice(z)	结点	Slice(z)
初始化	Φ	$10^9,3^{10},10^{16},3^{17},10^{23},3^{24}$	Φ
1^1	Φ	$4^{11},4^{18}$	Φ
2^2	Φ	$5^{12},5^{19}$	Φ
3^3	Φ	7^{13}	Φ
4^4	Φ	8^{14}	Φ
5^5	Φ	9^{15}	1,2,3,4,5,7,8,10
6^6	Φ	6^{20}	Φ
8^7	1,2,3,4,5,6,8	8^{21}	Φ
9^8	Φ	9^{22}	1,2,3,4,5,6,8,10

4 切片算法分析与评估

根据 H. Agrawal 计算动态切片的思想,从表 1 的所显示的依赖关系可画出程序 P 的依赖图,如图 1 所示:

(1)根据程序 P 的依赖图,计算切片准则为

$<x, 9^{15}, z>$ 的动态切片 Slice = {1,2,3,4,5,6,7,8,10}.

(2)根据程序 P 的依赖图,计算切片准则为 $<x, 9^{22}, z>$ 的动态切片 Slice = {1,2,3,4,5,6,7,8,10}.

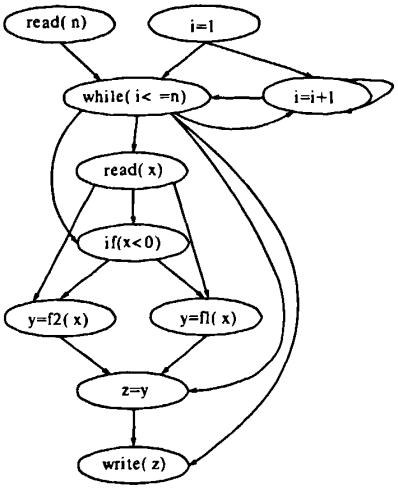

图 1　程序 P 的依赖图

Fig. 1　Dependence Graph of Program P

与 H. Agrawal 生成的动态切片相比,论文算法生成的动态程序切片更为精确,例如,9^{15} 处的 Slice(z) = {1,2,3,4,5,7,8,10},H. Agrawal 的计算结果 Slice(z) = {1,2,3,4,5,6,7,8,10};9^{22} 处的 Slice(z) = {1,2,3,4,5,6,8,10},而 H. Agrawal 的计算结果 Slice(z) = { 1,2,3,4,5,6,7,8,10}.

5　结论

笔者提出了一种简化依赖关系的动态程序切片算法,该算法通过程序执行轨迹,确定数据依赖结点、控制依赖结点和结点可到达语句,从而计算变量切片,避免了程序循环执行时重复遍历结点所得到的切片大且不精确的问题. 实例证明,该算法提高了动态程序切片的精确度,减少了计算动态程序切片的复杂度.

参考文献:

[1] WEISER M. Program slicing: formal psychological and practical investigations of an automatic program abstraction method [D]. Michigan: University of Michigan, 1979.

[2] 孙继荣,李志蜀,王　莉,等,程序切片技术在软件测试中的应用[J]. 计算机应用研究,2007,24(5):210 - 213.

[3] KOREL B,LASKL J. Dynamic program slicing[J]. Information Processing Letters, 1988, 29 (3): 155 - 163.

[4] KOREL B,LASKI J. Dynamic program slicing of computer program[J]. Journal of Systems and Software, 1990,13(3):155 - 163.

[5] KOREL B. Computation of dynamic slices for programs with arbitrary control - flow[C]// In 2nd International Wordshop on Automated and Algorithmic Debugging(ASDEBUG). Paris: Techniqueet Documentation Press,1996:71 - 86.

[6] 李必信. 程序切片技术及其应用[M]. 北京:科学出版社,2005.

[7] AGRAWAL H,DEMILLO R. Debugging with dynamic slicing and backtracking[J]. Software - Practice and Experience. 1993,23(6):589 - 616.

A Dynamic Program Slice Algorithm based on Simplified Dependence

JIA Li - min, ZHANG Zhong - lin

(School of Electronic and Information Engineering, Lanzhou Jiaotong University, Lanzhou 730070, China)

Abstract: Dynamic program slice is a program subset which contains variable values affecting some interesting point in program and some controlling predication. This paper presents a dynamic program slicing algorithm which simplifies dependence. The purpose of algorithm is to solve the problem that H. Agrawal's dynamic slicing is not accurate. By determining the data dependence node、control dependence node and reachable node, this algorithm calculates variable slice by program path. The experiment results show that it improves precision of dynamic program slicing and reduces complexity of dynamic slicing.

Key words: dynamic program slicing; slice criterion; dependence relationship; program dependence graph