第 36 卷 第 3 期　　东 北 农 业 大 学 学 报　　36(3) : 397~400
2005 年 6 月　　Journal of Northeast Agricultural University　　June 2005

文章编号　1005-9369(2005) 03-0397-04

动态程序切片研究及其应用

李英梅, 伦立军, 丁雪梅

(哈尔滨师范大学计算机科学系, 黑龙江　哈尔滨　150080)

摘　要: 动态程序切片根据程序的输入, 从源程序删除零条或多条语句, 得到对最终结果有潜在影响的源程序子集, 可用于程序调试、程序理解、软件测试和软件维护等方面。文章讨论了动态切片基本概念, 提出了一种基于动态流的动态切片方法, 并讨论了其在软件测试和程序调试中的应用。

关键词: 程序切片; 动态切片; 切片准则

中图分类号: TP31　　文献标识码: A

程序切片是一种代码分析技术, 它从语义角度对程序进行分解, 根据切片准则, 识别源代码中影响已知点变量值的所有语句。基本程序切片可分为静态切片和动态切片, 静态切片不需执行程序, 即可找到影响变量值的语句; 而动态切片依赖程序的特定输入, 保留对最终结果有影响的语句。通过运行动态程序切片, 有助于找出错误, 缩小错误的范围。由于动态切片可获得更简化的切片, 因此可应用于程序调试、软件测试、程序理解、软件维护和逆向工程等诸多方面。

程序切片技术最早由 Weiser 提出, 并通过遍历程序依赖图[1]得到过程内切片, 然后 Horwitz 通过将程序依赖图扩展为系统依赖图, 从而得到过程间切片; Korel 和 Laski 引入动态切片[2, 3], 主要寻找程序某次执行中, 影响变量在程序中某点状态的语句; Kamkar、Shahmehri 和 Fritzson 提出在执行过程中构造动态依赖图生成动态切片; A-grawal、DeMillo 和 Spaffford 提出依赖图算法[4]将定义基于存储单元的定义和引用上; 而 Duesterwald、Gupta 和 Soffa 提出依赖图算法用切片中决定信息陈述代替程序中非决定信息陈述。

本文首先讨论了动态切片基本概念, 提出一种基于动态流的动态程序切片方法。该方法简单, 易于实现, 仅需要一定的时空开销生成、存储动态流和执行状态迹。

收稿日期: 2004-08-31

基金项目: 黑龙江省教育厅科技项目(10541098)

作者简介: 李英梅 1970-), 女, 黑龙江人, 硕士研究生, 讲师, 主要从事软件工程、数据库技术方面的研究工作。

1　基本概念

定义1: 程序控制流图是一个有向图, 用 $G=(N, A, s, e)$ 表示, 其中 N 是结点集, 结点表示语句; A 是边集, 边表示语句间可能的控制流向; s 是唯一的源结点, 对应程序的开始语句; e 是唯一的汇结点, 对应程序的终止语句。从结点 s 到某结点 $k \in N$ 的路径是语句序列 $T=<n_1, n_2, \cdots, n_q>$, 其中 $n_1=s$, $n_q=k$ 且 $n(1 \leqslant i<q)$, $(n_i, n_i+1) \in A$。若存在输入数据使一条路径可执行, 则称该路径是可达的。一个程序的状态迹是对某特定输入数据而实际执行的可达路径。

语句 X 在状态迹中位置 p 处可表示为(X, p) , 记为 X^p。

定义 2: 设 T 是程序 P 对输入 x 的状态迹, P 的动态切片准则表示为一个三元组 $C=(x, I^q, V)$, 其中 x 是 P 的输入; I^q 是状态迹 T 中第 q 个元素的状态迹标号; V 是程序 P 的变量子集。

定义 3: 设 $C=(x, I^q, V)$ 是程序 P 的动态切片准则, T 是 P 对输入 x 的状态迹, C 上 P 的动态切片是满足如下条件的一个可执行程序:

①删除 P 中零条或若干条语句可得到 P;

②若 P 对输入 x 可终止执行且状态迹为 T, 则

P′对输入 x 也可终止执行且状态迹为 T′，且存在相应执行位置 q′，使 T 中 x^q 值与 T′中 $x^{q'}$值相等。

定义 4：一个变量在程序中的某处出现，使之与该变量相绑定的内容被引用，则称该出现是引用性出现，形式定义为：

$U(X^p)=\{v \mid$ 在 X^p 中引用变量 $v\}$。

一个变量在程序中的某处出现使数据与该变量相绑定，则称该出现是定义性出现，形式定义为：

$D(X^p)=\{v \mid$ 在 X^p 中定义变量 $v\}$。

定义 5：对给定的状态迹 T 和执行位置 r，若变量 v 满足如下条件，则称 X^p 是变量 v 的最后定义。

①$v \in D(X^p)$；

②$\forall k(p<k<r)$和 Y 满足 $T(k)=Y$，$v \notin D(Y^k)$。

定义 6：若变量 v 满足如下条件，则称 X^p 与 Y^t 具有定义–引用关系，记为 X^p DU $Y^t(1 \leqslant p<t)$。

①$v \in U(Y^t)$；

②X^p 是在 t 处变量 v 的最后定义。

定义 7：设 X 是程序 P 中的谓词，若语句 Y 出现在条件分支或循环中，则称语句 Y 在 X 影响范围内。

定义 8：设 X 是程序 P 中的谓词，Y 是一个语句，若 Y 在 X 影响范围内且 $\forall k(p<k<t)$，$T(k) \neq X$，则称 X^p 与 Y^t 具有测试控制关系，记为 X^p TC $Y^t(1 \leqslant p<t)$。

定义 9：设 X 是程序 P 中的谓词，Y 是一个语句，若 X = Y，则称 X^p 与 Y^t 具有恒等关系，记为 X^p IR Y^t。

2 动态流算法

动态流算法步骤如下：

①给定一个程序 P 和切片准则(x, I^q, v)，生成其状态迹 T；

②确定 DU、TC、IR 关系；

③定义 $A^0=LD(q, V) \cup LT(I^q)$，$LD(q, v)$是在执行位置 q，V 中变量最后定义集，$LT(I^q)$是对 I^q 有 TC 关系的测试控制集；

④定义 $S^0=A^0$，$S^{i+1}=S^i \cup A^{i+1}$，其中 $A^{i+1}=\{X^p \mid (X^p, Y^t) \in (DU \cup TC \cup IR)$，对 $Y^t \in S^i$ 项中 $X^p \notin S^i$，$p<q\}$，将最后得到的 S^{i+1} 作为切片 S_c；

⑤通过反复计算产生 S_c 作为序列 $S^0, S^1, \cdots, S^n$ 的极限 $(0 \leqslant n<q)$；

⑥对于 S_c 中所有的 X^p，切片包括语句 X 和切片准则 I^q 中的语句 I。

至此得到所需切片。见简单程序①：

```
1   read(n)
2   i=1
3   while (i<=n) do
      begin
4       if (i mod 2=0) then
5         x=17
        else
6         x=18
        endif
7       i=i+1
    end
8   write(x)
```

简单程序①

以上程序对输入 n = 2 的执行过程状态迹 T =< 1, 2, 3, 4, 6, 7, 3, 4, 5, 7, 3, 8>，见执行语句②：

```
1[1]   read(n)
2[2]   i=1
3[3]   i<=n
4[4]   i mod 2=0
6[5]   x=18
7[6]   i=i+1
3[7]   i<=n
4[8]   i mod 2=0
5[9]   x=17
7[10]  i=i+1
3[11]  i<=n
8[12]  write(x)
```

执行语句②

下面构造 DU、TC 和 IR 关系：

①DU 关系描述一个行为为一个数据项分配一个值，其它行为引用该值，即将一个变量的引用和其最后定义联系起来，在执行语句②中：

$DU=\{(1^1, 3^3), (1^1, 3^7), (1^1, 3^{11}), (2^2, 3^3), (2^2, 4^4), (2^2, 7^6), (7^6, 3^7), (7^6, 4^8), (7^6, 7^{10}), (5^9, 8^{12}), (7^{10}, 3^{11})\}$

②TC 关系描述测试行为及其影响执行的行为间的关系，即将一个逻辑表达式的最新出现和控制依赖于其上的状态迹中出现的语句联系起来。如执行语句②中 3^3 影响 $4^4 6^5 7^6$ 的执行，但不影响 $4^8 5^9 7^{10}$ 的执行，影响它们的是 3^7。在执行语句②中：

TC={(3^3, 4^4), (3^3, 6^5), (3^3, 7^6), (4^4, 6^5), (3^7, 4^8), (3^7, 5^9), (3^7, 7^{10}), (4^8, 5^9)}

③IR 关系描述出现的相同语句。在执行语句②中：

IR={(3^3, 3^7), (3^3, 3^{11}), (3^7, 3^3),(3^7, 3^{11}),(3^{11}, 3^3),(3^{11}, 3^7), (4^4, 4^8), (4^8, 4^4), (7^6, 7^{10}), (7^{10}, 7^6)}

在程序①中切片准则 $(2, 8^{12}, x)$ 为计算切片 S_c，首先找到对 I^q 有直接影响的所有行为集 A^0，因 x=17 对 write(x)有直接影响，所以 $LD(8, \{x\})=\{5^9\}$，$LT(8^{12})=\varnothing$，从而 $A^0=\{5^9\}$，再通过 $S^0=A^0$，$S^{i+1}=S^i\cup A^{i+1}$，$A^{i+1}=\{$DU、TC、IR 三者中所有项的后项$\in S^i$，此项中前项$\notin S^i$ 的前项$\}$，反复计算产生 S_c 作为序列 $S^0, S^1,\cdots, S^n$ 的极限 $(0\leqslant n<q)$。在语句②中：

$LD(8, \{x\})=\{5^9\}$，$LT(8^{12})=\varnothing$

$A^0=\{5^9\}$，$S^0=\{5^9\}$

$A^1=\{3^7, 4^8\}$，$S^1=\{3^7, 4^8, 5^9\}$

$A^2=\{1^1, 3^3, 4^4, 7^6, 3^{11}\}$，$S^2=\{1^1, 3^3, 4^4, 7^6, 3^7, 4^8, 5^9, 3^{11}\}$

$A^3=\{2^2, 7^{10}\}$，$S^3=\{1^1, 2^2, 3^3, 4^4, 7^6, 3^7, 4^8, 5^9, 7^{10}, 3^{11}\}$

$A^4=\varnothing$

至此停止计算，得到的切片 Sc 为：

$S_c=S^3\cup\{8^{12}\}=\{1^1, 2^2, 3^3, 4^4, 7^6, 3^7, 4^8, 5^9, 7^{10}, 3^{11}, 8^{12}\}$

得到的动态切片为：

```
1   read(n)
2   i=1
3   while(i<=n) do
        begin
4       if (i mod 2=0) then
5           x=17
        endif
7       i=i+1
        end
8   write(x)
```

3　动态程序切片应用

动态切片可用于程序调试、软件测试、程序理解和软件维护等方面，下面主要讨论动态切片应用于软件测试和程序调试阶段。

3.1　软件测试

基于程序切片的软件测试是一种以程序或程序和需求相结合为基础的测试，它根据计算程序的不同切片缩小软件测试范围，提高测试效率，同时因程序切片考虑程序存在的各种依赖关系(而不仅是数据依赖和控制依赖)，所以使测试的准确性得到提高。此外任何程序可等价于一组程序切片的并集，测试每个切片实际就是测试整个程序，从而保证测试的充分性。

根据简单程序①对切片准则$(1, 8^{12}, x)$，得到动态切片为：

```
1   read(n)
2   i=1
3   while (i<=n) do
        begin
4       if (i mod 2=0) then
        else
6           x=18
        endif
7       i=i+1
        end
8   write(x)
```

源程序等价于对应切片准则，分别输入奇数和偶数得到两个动态切片的并集，因此测试这两个切片即等价于测试源程序。

3.2　程序调试

在程序调试中，最常见的工作是发现一个错误并找出所有与该错误有关的语句，动态程序切片工具可容易地做到这一点。如简单程序①，若执行结果错误，则应在所得到的动态切片中找错误，而不是被删除的语句，这就缩小了错误的范围，达到错误定位的目的。在大型软件项目的调试中，经常会遇到对于某些输入会产生正确结果，而对另一些输入则会产生错误结果的情况。此时若多次运行程序，逐条跟踪语句的执行，会耗费大量时间。而采用动态切片技术，可构造输入语句的前向切片和输出语句的动态切片，并取两者的交集，这样可极大缩小程序的考察范围[5~7]。

4　结　论

动态程序切片是根据程序的输入，保留对最终结果有影响的语句。通过运行动态程序切片，有助于找出错误，缩小错误范围。本文讨论了动态流算法，该方法得到的切片是可执行的，且易于理

解和实现。目前对程序切片技术的研究主要集中于过程型程序方面，如何解决面向对象技术引入的新问题，如何得到更准确、更快速和更有效的切片，有待于进一步研究。

[参 考 文 献]

[1] Weiser M. Program slicing [J]. IEEE Trans Software Eng, 1994, 10(4): 352–357.

[2] Korel B, Laski J. Dynamic slicing of computer programs [J]. J Systems and Software, 1990, 13(3): 187–195.

[3] Korel B, Rilling J. Dynamic program slicing methods [J]. Information and Software Technology, 1998, 40(11): 647–659.

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

[5] Agrawal H, Demillo R, Spafford E. Debugging with dynamic slicing and backtracking [J]. Softw Prac Exper, 1993, 23(6): 589–616.

[6] Gallagher K, Lyle J. Using program slicing in software maintenance [J]. IEEE Trans Software Eng, 1991, 17(8): 751–761.

[7] Lanubile F, Visaggio G. Extracting reusable functions by flow graph–based program slicing [J]. IEEE Trans Software Eng, 1997, 23(4): 246–259.

Research and application of dynamic program slicing

LI Ying–mei, LUN Li–jun, DING Xue–mei

(Department of Computer Science, Harbin Normal University, Harbin Heilongjiang 150080, PRC)

Abstract: Dynamic program slicing is computed regarding a program′s input, obtained the subset of original program which potentially affect the final results by removing zero or more statements from the original program. Dynamic program slicing is applied to program debugging, program comprehension, software testing and software maintenance. The paper discusses basic concepts in dynamic slicing at first, then presents a dynamic slicing method based–on dynamic flow, it′s application in software test and program debugging is discussed at last.

Key words: program slicing; dynamic slicing; slicing criterion