基于变量作用域的数据流分析

姜淑娟 赵雪峰
(中国矿业大学计算机科学与技术学院 徐州 221116)

摘 要 数据流分析作为程序分析的一种重要手段,已广泛应用于各种软件工程任务中。传统的数据流迭代分析法没有考虑变量因作用域问题而被隐藏和覆盖的现象,导致数据流信息不准确。在传统数据流迭代分析法的基础上提出一种基于变量作用域的数据流分析方法,它解决了变量被隐藏和覆盖的问题。最后将改进的方法和传统分析方法分别应用于程序切片中,实验证实了改进的方法更加准确。

关键词 数据流,迭代,作用域,程序切片
中图法分类号 TP311 **文献标识码** A

Data Flow Analysis Based on the Variable Scope

JIANG Shu-juan ZHAO Xue-feng
(School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China)

Abstract As an important technology of programming analysis, data flow analysis is widely applied into kinds of software projects. Problem of variables being hidden and covered for their scope has not been taken into accout in traditional dataflow iteration analysis, leading to inaccurate data flow information. An improved dataflow analysis method based on variables scope and traditional dataflow iteration analysis method was proposed to solve the problem of varibles being hidden and covered. At last, both of these methods were applied into program slice, which proves the efficiency of the improved one by comparing.

Keywords Data flow, Iteration, Socpe, Program slicing

1 引言

1977年,L. J. Osterweil 和 L. D. Fosdick 首次提出数据流分析的概念[1]。数据流分析指的是一组用来获取有关数据如何沿着程序执行路径流动的相关信息的技术[2]。

数据流分析是编译领域的一个重要问题,主要包括"定值—引用"分析、"活跃变量"分析以及公共子表达式删除等。数据流分析已广泛应用到各类软件工程任务中,如程序理解、程序优化、软件维护、软件测试以及软件调试等。

目前,有两种主要的数据流分析方法,分别是用于结构化程序的语法制导求解法和用于任意控制流程序的迭代数据流方程求解法。迭代数据流方程求解法应用更为广泛,在该方法中首先建立控制流图,然后在控制流图的基础上为每个结点建立数据流方程,最后迭代求解这些数据流方程,以计算出相关的数据流信息。

然而,在传统数据流迭代分析方法中没有考虑变量因作用域变化而被隐藏和覆盖的问题,使得数据流分析结果不够准确。本文在传统迭代法的基础上对数据流方程进行了改进,提出一种基于变量作用域的数据流分析方法,并给出相应的分析算法。

最后,将本文提出的数据流分析方法和传统迭代分析法分别应用于程序切片中,实验表明通过本文提出的方法计算出的切片结果更加准确。

2 传统数据流迭代分析法

2.1 传统迭代数据流方程及其算法

定义1(基本块) 一组连贯的语句,其中的控制流在模块开始时进入,在模块结束时离开,而没有在除模块结束之外的地方停止或出现分支的可能性[3]。

定义2(控制流图) 是一个有向图,其中节点代表基本语句块,边代表控制流路径,即若控制流能直接从基本块 X 流入基本块 Y,则在 X 与 Y 之间存在一条有向边[4]。

目前在数据流分析中使用最多的是编译领域的迭代分析法。在迭代分析法中常用到如下的数据流集合(假设 B 为基本块):

Def(B):在 B 中定义(或赋值)的变量集;

Use(B):在 B 中引用的变量集;

In(B):到达 B 入口处的数据流信息;

Out(B):在 B 的出口处流出的数据流信息;

Gen(B):在 B 中新生成的定值点,且能到达 B 的出口处

到稿日期:2011-04-28 返修日期:2011-07-15 本文受江苏省自然科学基金(BK2008124),国家自然科学基金(60970032),中国矿业大学科学研究基金(OD080310)资助。

姜淑娟(1966—),女,博士,教授,博士生导师,CCF会员,主要研究方向为软件工程、软件测试、程序设计语言等,E-mail:shjjiang@cumt.edu.cn;
赵雪峰(1986—),男,硕士生,主要研究方向为软件工程、软件测试、程序切片等。

的数据流信息；

Kill(B)：在 B 中被注销的数据流信息；

Pred(B)：控制流图中 B 的前趋节点集合。

根据沿控制流正向或逆向分析，数据流分析可以分为正向数据流分析和逆向数据流分析。在本文中主要考虑正向数据流分析。数据流迭代分析法将问题抽象为基本数据流方程的求解。正向数据流分析的基本方程如式(1)所示[2]。

$$\begin{cases} \mathrm{Out}(B)=\mathrm{Gen}(B)+(\mathrm{In}(B)-\mathrm{Kill}(B)) \\ \mathrm{In}(B)=\bigcup\limits_{P\in\mathrm{Pred}(B)}\mathrm{Out}[P] \end{cases} \tag{1}$$

与式(1)中数据流方程相应的迭代分析算法如图 1 所示。

```
input : gens----the set of variable-definition of all blocks
        kills---- the set of variable-kill of all blocks
output: ins----input-data flows
        outs----output-data flows
begin
1 for each block B
2     out[B] = gen[B];
3 end for
4 change = true;
5 while change
6     change = false;
7     for each block B

8         in[B] = ∪ out[p] ;
                  p is precursor of B

9         oldout = out[B];
10        out[B] = gen[B] + (in[B] - kill[B]);
11        if out[B] ≠ oldout then
12            change = true;
13        end if
13    end for
14 end while
end
```

图 1　传统数据流迭代分析算法

2.2　传统数据流迭代分析法中存在的问题

传统的数据流分析方法没有考虑变量因作用域的变化而被隐藏和覆盖的问题，造成数据流信息丢失，导致数据流分析结果不够准确。

如图 2 所示，在函数 $fun()$ 中，第 3 行定义的变量 c 因第 6 行的定义语句在 6－15 行的范围内被隐藏，同理第 6 行定义的变量 c 也因第 9 行的定义语句在 9－11 行的范围内被隐藏，但因第 9 行定义的变量 c 的作用域为 9－11 行，因此第 6 行定义的变量 c 从 12 行起再次可见。若采用传统迭代分析法，第 6 行定义的变量 c 会在第 9 行被注销(KILL)，因此它不可能到达第 12 行，很明显这样的结果是不准确的，其余变量的分析同理可得。

```
1  void fun()
2  {
3      int a, b, c;
4      {
5          int d = 0;
6          int c = 1;
7          a = 9;
8          {
9              int c;
10             c = 0;
11         }
12         int a;
13         int b;
14         a = c;
15     }
16     b = a;
17 }
```

图 2　例子程序

3　基于变量作用域的数据流分析方法

3.1　基于变量作用域的数据流方程及其算法

正如 2.2 节所述，在不考虑变量作用域的情况下得到的数据流分析结果不准确。为此，本文引入了两个新的集合用于表示变量的作用域对数据流分析带来的影响。

Show(B)：在基本块 B 的入口处重新可见的数据流信息；

Hide(B)：在基本块 B 内被隐藏的数据流信息。其中：

$$\begin{cases} \mathrm{Show}(B)=\{\mathrm{val}\mid \mathrm{val}\in\mathrm{Hide}(P)\ \&\&\ P\in\mathrm{Pred}(B) \\ \qquad \&\&\ \mathrm{val\ shows\ again\ in\ the\ entry\ point\ of}\ B\} \\ \mathrm{Hide}(B)=\{\mathrm{val}\mid((\mathrm{val}\in\mathrm{Hide}(P)\ \&\&\ \mathrm{val\ continue\ to} \\ \qquad \mathrm{be\ hided\ in\ the\ entry\ point\ of}\ B)\ ||\ (\mathrm{val}\notin \\ \qquad \mathrm{Hide}(P)\ \&\&\ \mathrm{val\ is\ hided\ in}\ B))\ \&\& \\ \qquad P\in\mathrm{Pred}(B)\} \end{cases} \tag{2}$$

改进后的正向数据流分析方程如下：

$$\begin{cases} \mathrm{Out}(B)=\mathrm{Gen}(B)+(\mathrm{In}(B)-\mathrm{Kill}(B)-\mathrm{Hide}(B)) \\ \mathrm{In}(B)=\bigcup\limits_{\substack{P\in\mathrm{Pred}(B)\ \&\&\ B\ \mathrm{in} \\ \mathrm{the\ scope\ of\ the\ Out}[P]}}\mathrm{Out}(P)+\mathrm{Show}(B) \end{cases} \tag{3}$$

式中，从基本块 B 流出的数据流(Out(B))包括在 B 中生成的数据流加上流入 B 中的数据流，同时要减去在 B 中注销和隐藏的数据流；流入基本块 B 的数据流(In(B))包括从 B 的所有前趋节点流出的且 B 在其作用域范围内的数据流加上在 B 中再次可见的数据流信息。与式(3)相应的改进后的数据流分析迭代算法如图 3 所示。

```
input : gens----the set of variable-definition of all blocks
        kills---- the set of variable-kill of all blocks
        hides----the set of variable which is hided in sub scope
        shows----the set of variable which shows again after hided
output: ins----input-data flows
        outs----output-data flows
begin
1 for each block B
2     out[B] = gen[B];
3 end for
4 change = true;
5 while change
6     change = false;
7     for each block B

8         in[B] = ∪ out[p] + show[B];
                  p is precursor of B &&
                  B in the scope of the out[p]
9         oldout = out[B];
10        out[B] = gen[B] + (in[B] - kill[B] - hide[B]);
11        if out[B] ≠ oldout then
12            change = true;
13        end if
13    end for
14 end while
end
```

图 3　改进的数据流迭代分析算法

第 1－3 行，初始化每个语句块的 Out 集合，使其等于该语句块的 Gen 集合；5－14 行迭代计算每个语句块的 In、Out 集合，其中，第 8 行计算语句块的 In 集合，使其等于该语句块所有前趋的 Out 集合的并集，但要求该语句块必须在其作用域内，同时将在该语句块处再次可见的变量定义集也加入 In 集合中；第 10 行计算语句块的 Out 集合；11－13 行判断是否还需要一次迭代，当所有语句块的 Out 集合不变时，停止迭代。

3.2　分析结果比较

对图 2 所示程序分别应用传统迭代分析法和本文提出的改进算法所得分析结果如表 1 所列，其中有灰色背景的为传统迭代法分析所得结果，两种方法所得结果的差别在表中用加粗表示。

由表 1 可知，在考虑由变量作用域引起的变量注销、隐藏和重新可用的情况后，所得分析结果更加准确，并且程序规模越大，效果越明显。

表 1　分析结果对比

语句＼集合	GEN	KILL	HIDE	SHOW	IN	OUT
3	⟨3,a⟩⟨3,b⟩⟨3,c⟩	Ø			Ø	⟨3,a⟩⟨3,b⟩⟨3,c⟩
	⟨3,a⟩⟨3,b⟩⟨3,c⟩	Ø	Ø	Ø	Ø	⟨3,a⟩⟨3,b⟩⟨3,c⟩
5	⟨5,d⟩	Ø			⟨3,a⟩⟨3,b⟩⟨3,c⟩	⟨3,a⟩⟨3,b⟩⟨3,c⟩⟨5,d⟩
	⟨5,d⟩	Ø	Ø	Ø	⟨3,a⟩⟨3,b⟩⟨3,c⟩	⟨3,a⟩⟨3,b⟩⟨3,c⟩⟨5,d⟩
6	⟨6,c⟩	⟨3,c⟩			⟨3,a⟩⟨3,b⟩⟨3,c⟩⟨5,d⟩	⟨3,a⟩⟨3,b⟩⟨5,d⟩⟨6,c⟩
	⟨6,c⟩	Ø	⟨3,c⟩	Ø	⟨3,a⟩⟨3,b⟩⟨3,c⟩⟨5,d⟩	⟨3,a⟩⟨3,b⟩⟨5,d⟩⟨6,c⟩
7	⟨7,a⟩	⟨3,a⟩			⟨3,a⟩⟨3,b⟩⟨5,d⟩⟨6,c⟩	⟨3,b⟩⟨5,d⟩⟨6,c⟩⟨7,a⟩
	⟨7,a⟩	⟨3,a⟩	⟨3,c⟩	Ø	⟨3,a⟩⟨3,b⟩⟨5,d⟩⟨6,c⟩	⟨3,b⟩⟨5,d⟩⟨6,c⟩⟨7,a⟩
9	⟨9,c⟩	⟨6,c⟩			⟨3,b⟩⟨5,d⟩⟨6,c⟩⟨7,a⟩	⟨3,b⟩⟨5,d⟩⟨7,a⟩⟨9,c⟩
	⟨9,c⟩	Ø	⟨3,c⟩⟨6,c⟩	Ø	⟨3,b⟩⟨5,d⟩⟨6,c⟩⟨7,a⟩	⟨3,b⟩⟨5,d⟩⟨7,a⟩⟨9,c⟩
10	⟨10,c⟩	⟨9,c⟩			⟨3,b⟩⟨5,d⟩⟨7,a⟩⟨9,c⟩	⟨3,b⟩⟨5,d⟩⟨7,a⟩⟨10,c⟩
	⟨10,c⟩	⟨9,c⟩	⟨3,c⟩⟨6,c⟩	Ø	⟨3,b⟩⟨5,d⟩⟨7,a⟩⟨9,c⟩	⟨3,b⟩⟨5,d⟩⟨7,a⟩⟨10,c⟩
12	⟨12,a⟩	⟨7,a⟩⟨10,c⟩			⟨3,b⟩⟨5,d⟩⟨7,a⟩⟨10,c⟩	⟨3,b⟩⟨5,d⟩⟨12,a⟩
	⟨12,a⟩	⟨10,c⟩	⟨3,c⟩⟨6,c⟩⟨7,a⟩	⟨6,c⟩	⟨3,b⟩⟨5,d⟩⟨7,a⟩⟨10,c⟩	⟨3,b⟩⟨5,d⟩⟨6,c⟩⟨12,a⟩
13	⟨13,b⟩	⟨3,b⟩			⟨3,b⟩⟨5,d⟩⟨12,a⟩	⟨5,d⟩⟨12,a⟩⟨13,b⟩
	⟨13,b⟩	Ø	⟨3,b⟩⟨3,c⟩⟨7,a⟩	Ø	⟨3,b⟩⟨5,d⟩⟨6,c⟩⟨12,a⟩	⟨5,d⟩⟨6,c⟩⟨12,a⟩⟨13,b⟩
14	⟨14,a⟩	⟨12,a⟩			⟨5,d⟩⟨12,a⟩⟨13,b⟩	⟨5,d⟩⟨13,b⟩⟨14,a⟩
	⟨14,a⟩	⟨12,a⟩	⟨3,b⟩⟨3,c⟩⟨7,a⟩	Ø	⟨5,d⟩⟨6,c⟩⟨12,a⟩⟨13,b⟩	⟨5,d⟩⟨6,c⟩⟨13,b⟩⟨14,a⟩
16	⟨16,b⟩	⟨5,d⟩⟨13,b⟩⟨14,a⟩			Ø	⟨16,b⟩
	⟨16,b⟩	⟨3,b⟩⟨5,d⟩⟨6,c⟩⟨13,b⟩⟨14,a⟩	⟨3,b⟩⟨3,c⟩⟨7,a⟩	⟨3,b⟩⟨3,c⟩⟨7,a⟩	**⟨5,d⟩⟨6,c⟩⟨13,b⟩⟨14,a⟩**	**⟨3,c⟩**⟨7,a⟩⟨16,b⟩

4　在程序切片中的应用

自从 1979 年，M. Weiser 博士首次提出程序切片的概念以来，程序切片技术得到极大的发展和应用[5]。研究人员先后提出前向与后向切片[6]、静态与动态切片[7]以及过程内与过程间切片等。

在计算程序切片时必须先进行数据流分析以获得各语句间的数据依赖关系，因此若用传统的数据流迭代分析方法计算数据依赖关系并由此计算程序切片，那么切片结果可能不够准确。

为了进一步通过实验验证本文的方法与传统方法的差异，在由 GCC 编译源程序生成的中间结果 AST（Abstract Syntax Tree，抽象语法树）的基础上开发出一个程序分析工具（OOAnalyser），该工具主要用于分析 C、C++源程序，其框架图如图 4 所示。

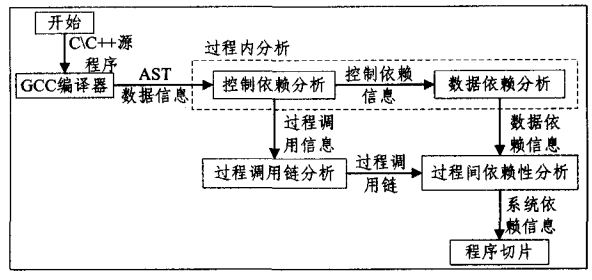

图 4　OOAnalyser 框架图

为了生成源代码的中间表示 AST，需在编译源代码时为 GCC 命令添加“-fdump-translation-unit”选项。AST 包含了源程序全部的静态信息，因此适合用于做源码分析。此外，AST 是 GCC 经过词法分析和语法分析后得到的结果，从而可以省去对源程序进行词法和语法分析的过程，提高了分析效率和准确度。

用本文设计的 OOAnalyser 工具对 6 个程序分别进行实验并与传统的方法进行了对比，实验结果如图 5 所示，其中纵坐标表示切片结果的差异，横坐标表示程序编号。由图 5 可知，利用本文提出的方法得到的切片结果比传统方法更加精确，因为在传统方法中变量一旦被隐藏就不再可用将导致部分数据流信息的丢失。

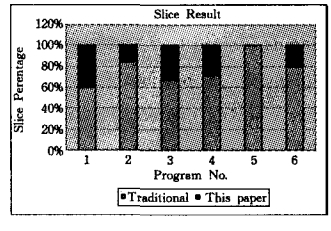

图 5　实验对比结果

5　相关工作

William E. Weihl[8] 提出在过程间进行数据流分析时解决指针、过程变量和标量变量等问题的方法，使得数据流分析结果更加准确；Gleb Naumovich[9] 将观察者设计模式应用于数据流分析中，它与传统的基于控制流图的数据流分析方法相比效率更高，更加适合于面向对象的程序分析。然而，他们的方法都没有很好地解决变量作用域的问题，使得最终的分析结果不够准确。

L. Larsen 和 M. J. Harrold[10] 提出一种面向对象程序的切片方法，它将之前的过程内和过程间的切片技术扩展到面向对象程序中。S. Sinha，M. J. Harrold 和 G. Rothermel[11] 在系统依赖图的基础上解决了含有任意控制流的程序切片问

题，如含有异常处理结构的程序切片问题。但他们的方法都基于传统的数据流迭代分析法，使得最终的切片结果会因变量作用域的问题变得不够准确。

结束语 目前主要的编程语言，如C、C++、Java、VB等都存在变量因作用域问题而被注销、隐藏和覆盖的问题，在进行数据流分析时若采用编译领域中的传统迭代分析法，会造成部分数据流信息的丢失。如果将这样的数据流分析结果应用于各种软件工程任务中，如程序切片、程序测试等，则最终获得的结果可能也是不准确的。本文在传统迭代分析方法基础上提出基于变量作用域的数据流分析方法，在数据流方程中引入两个新的集合分别用于记录变量因作用域的改变而被隐藏和重新可见的信息，解决了数据流信息因作用域变化而丢失的问题。最后利用我们开发的程序分析工具OOAnalyser，分别将传统的和本文提出的数据流分析方法应用于程序切片中。实验结果表明，我们的方法得到的切片结果更加准确。今后，将进一步研究将所得的程序切片结果应用于各种软件工程任务中以便解决实际的问题，如程序切片在测试数据生成中的应用、在错误定位中的应用等。

参 考 文 献

[1] Fosdick L D, Osterweil L J. Data Flow Analysis in Software Reliability[J]. ACM Computing Surveys, 1976, 8(3): 305-330

[2] Alfred V A, Monica S L, Ravi S, et al. Compilers Principles, Techniques and Tools(第2版)[M]. 赵建华，郑滔，戴新宇，译. 北京：机械工业出版社，2009：382-402

[3] 李必信. 程序切片技术及其应用[M]. 北京：科学出版社，2006：17-32

[4] Allen F E. Control flow analysis[J]. Proceedings of a symposium on Compiler optimization, 1970, 5(7): 1-19

[5] Weiser M. Program slicing[C]// Proceedings of the 5th international conference on Software engineering, 1981. New York: IEEE Press, 1981: 439-449

[6] Zhang Xiang-yu, Gupta R, Zhang You-tao. Efficient forward computation of dynamic slices using reduced ordered binary decision diagrams[C]// Proceedings of the 26th International Conference on Software Engineering, 2004. Washington: IEEE Computer Society, 2004: 502-511

[7] Dhamdhere D M, Gururaja K, Ganu P G. A compact execution history for dynamic slicing[J]. Information Processing Letters, 2003, 85(3): 145-152

[8] Weihl W E. Interprocedural data flow analysis in the presence of pointers, procedure variables, and label variables[C]// Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 1980. New York: ACM, 1980: 83-94

[9] Naumovich G. Using the Observer Design Pattern for Implementation of Data Flow Analyses[C]// Proceedings of the 2002 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, 2003. New York: ACM, 2003: 61-68

[10] Larsen L, Harrold M J. Slicing Object-Oriented Software[C]// Proceedings of the 18th international conference on Software Engineering, 1996. Washington: IEEE Press, 1996: 95-505

[11] Sinha S, Harrold M J, Rothermel G. System Dependence Graph Based Slicing of Programs with Arbitrary Interprocedural Control Flow[C]// Proceedings of the 21st International Conference on Software Engineering, 1999. New York: ACM Press, 1999: 432- 441

(上接第117页)

[3] Sarbazi-Azad H, Khonsari A, Ould-Khaoua M. Analysis of k-ary n-cubes with dimension-ordered routing[J]. Future Generation Computer Systems, 2003, 19: 493-502

[4] Yang Y L, Funahashi A, Jouraku A, et al. Recursive Diagonal Torus: an interconnection network for massively parallel computers[J]. IEEE Transactions on Parallel and Distributed Systems, 2001, 12(7): 701-714

[5] Karim F, Nguyen A, Dey S. An Interconnect Architecture for Networking Systems on Chips[J]. IEEE Micro, 2002, 22(5): 36-45

[6] Shih J-D. Fault-tolerant wormhole routing in torus networks with overlapped block faults[J]. IEE Proc. Comput. Digit. Tech., 2003, 150(1)

[7] Dally W J, Seitz C. Deadlock-free message routing in multiprocessor interconnection networks [J]. IEEE Transactions on Computers, 1987, C-36: 547-553

[8] Glass C J, Ni L M. The turn model for adaptive routing[C]// Proceedings of the 19th International Symposium on Computer Architecture. May 1992: 278-287

[9] Chiu G-M. The odd-even turn model for adaptive routing[J]. IEEE Trans. on Parallel and Distributed Systems, 2000, 11: 729-738

[10] Wu J. A fault-tolerant and deadlock-free routing protocol in 2D meshes based on odd-even turn model[J]. IEEE Transactions on Computers, 2003, 52: 1154-1169

[11] Lin S-Y, Huang C-H, Chao C-H, et al. Traffic-balanced Routing Algorithm for Irregular Mesh-based on-Chip Networks [J]. IEEE Transactions on Computers, 2008, 57(9)

[12] Zou Y, Pascricha S. NARCO: Neighbor Aware Turn Model-based Fault Tolerant Routing for NoCs[J]. IEEE Embedded Systems Letters, 2010, 2: 85-89

[13] Li Y H, Gu H G. Fault tolerant routing algorithm based on the artificial potential field model in Network-on-Chip[J]. Applied Mathmatics and Computation, 2010, 217: 3226-3235

[14] Boppana R V, Chalasani S. Fault-tolerant wormhole routing algorithms for mesh networks[J]. IEEE Transactions on Computers, 1995, 44(7): 848-864

[15] Duan X M, Zhang D K, Sun X M. Routing Schemes of An Irregular Mesh-based NoC [C]// 2009 International Conference on NSWCTC. vol. 2, Apr, 2009: 572-575

[16] Linder D H, Harden J C. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes[J]. IEEE Trans. Comput, 1991, 40: 2-12