广西科学院学报
Journal of Guangxi Academy of Sciences

2009,25(1):26~29
Vol. 25,No. 1 February 2009

基于 JavaSDG 的 Java 程序参数依赖性分析
Program Parameter Dependence Analysis for Java Based on JavaSDG

蒋曹清
JIANG Cao-qing

(广西财经学院计算机与信息管理系,广西南宁 530003)
(Department of Computer and Information Management, Guangxi University of Financial and Economics, Nanning, Guangxi, 530003, China)

摘要:基于一种带有标签的 Java 程序描述(JavaSDG),给出 JavaSDG 中的过程依赖图(PDG)的形式化定义,然后分析 Java 程序方法调用引起的参数之间的依赖关系。并比较基于 SDG 和基于 JavaSDG 两种程序分析方法的异同,基于 JavaSDG 的 Java 程序分析方法构造的 PDG 具有结点少、可以重用、可以并发构造等优点。

关键词:程序分析 Java 程序 依赖分析
中图法分类号:TP311.5 **文献标识码**:A **文章编号**:1002-7378(2009)01-0026-04

Abstract: Based on a representation for Java program with tags JavaSDG, this paper gives out formal definition of PDG in JavaSDG and analyzes parameter dependence for Java program method calling. Moreover advantages differences of the two approaches between SDG-based and JavaSDG-based are discussed. The Java program analysis based on JavaSDG are fewer nodes of PDG, repeated use of PDG, etc.

Key words: program analysis, Java program, dependence analysis

程序分析广泛应用在编译器的优化和变换及软件工程的各个环节中。Java 语言是一种新的面向对象编程语言,由于它具有很强的可移植性、安全性和网络功能,在 Internet 和 Web 普及的信息时代,已成为互联网应用开发的主流语言,对它的分析研究具有重要的意义。然而现存的程序分析技术主要基于相互连接的系统依赖图(SDG)给出[1~4],这种 SDG 的描述非常复杂[5],特别是大程序,在构造 SDG 的过程中容易出现错误。

文献[6]介绍了一种由一组带有标签而且互不相连的过程依赖图(PDG)组成的 Java 程序描述方法(不妨称为 JavaSDG)。JavaSDG 是由方法和类的 PDG 组成的集合,一个类的 PDG 是由它的所有方法的 PDG 组成的集合。JavaSDG 在 Java 程序描述时,程序中的方法通过参数之间的依赖关系相互影响,内部数据和语句之间的依赖关系在外部不可见;每个 PDG 是一个独立的图,不与外部连接。本文基于这种简洁的 Java 程序描述,首先给出 JavaSDG 中 PDG 形式化定义,然后分析 JavaSDG 调用点参数之间的依赖关系,最后对比这种基于 JavaSDG 的 Java 程序分析方法与传统的 SDG 程序分析方法的优缺点。

1 JavaSDG 中 PDG 形式化定义

在 Java 中,方法可以由控制流图(CFG)表示,一个方法的 CFG 是一个有向图,CFG=(S,E,S_{entry},S_{exit}),这里 S 是结点集,E 是边集,结点表示语句或谓词,E={<s_1,s_2>| s_1,s_2∈S,并且 s_1 执行后可能执行 s_2},S_{entry}是方法的入口结点,S_{exit}是方法的退出结点。如果<s_1,s_2>∈E,那么 s_1 是 s_2 的前驱,s_2 是 s_1 的后继。

定义 1 设 s 是程序 CFG 中的任一结点,则:(1)定义集 Def(s)表示语句 s 中被定义(修改)值的变量;(2)引用集 Ref(s) 表示在语句 s 中引用但没

收稿日期:2008-07-22
作者简介:蒋曹清(1973-),男,硕士,讲师,主要从事软件测试与程序分析研究。

有修改值的变量;(3)Def(s,x)表示在语句 s 定义变量 x 所引用的变量;(4)数据定义依赖集 Dep-D(s,x)={(x,s',y)|x∈Def(s)∧y∈Def(s')∧y∈Def(s,x)∧存在一条 s'到 s 的路,并且在这条路上没有被重新定义};(5)数据引用依赖集 Dep-R(s)={(x,s',x)|如果 s 是一条谓词语句(例如 if,while 语句),x 是 s 处使用的条件变量,x∈Def(s')∧x∉Def(s)∧存在一条 s'到 s 的路,并且在这条路上没有被重新定义}。

例如:对于下面一个 Java 程序段,

```
S1   public class SimpleCalc {
S2     int a,b;
E3     public SimpleCalc(int aIn, int bIn){
S4       a = aIn;
C5       b = multiply(bIn);
           }
E6     protected int multiply(int d){
S7       a=a+d;
S8       int i=0;
S9       While (i<a){
S10        d=d+d;
S11        i++;
         }
S12      return d;
           }
       }
```

可以得到 Def(S7)={a}, Ref(S7)={a,d}, Def(S7,a)={a,d}, Dep-D(S11,i)={i,S8,i}, Dep-R(S9)={i,S8,i}。

定义 2 一个方法 M 的 PDG 是一个带有标签的有向图,PDG=(S,E,T),这里结点集 S 由入口结点 entry、语句结点、谓词结点组成;边集 $E=E_1 \cup E_2$,这里 E_1 表示有向控制依赖边集,$E_2=\{<s_1,s_2> | (x, s_1, y) \in \text{Dep-D}(s_2,x) \vee (x, s_1, x) \in \text{Dep-R}(s_2)\}$ 表示有向数据的依赖边集;T 是一个标签集,边$<s_1,s_2>$上标签通过以下形式定义:如果$<s_1,s_2> \in E_1$,那么标签为(*,*);如果$<s_1,s_2> \in$ Dep-D(s_2,x),那么标签为(y,x);如果$<s_1,s_2> \in$ Dep-R(s_2),那么标签为(x,x)。

图 1 是上述 Java 程序的方法 multiply 的 PDG,在这个 PDG 中入口结点是 S6,d 是一个形参,a 是 multiply 的全局变量,JavaSDG 中默认在方法的入口结点处定义形参和全局变量,故 Def(S6)={a,d}。语句 S7 定义 a,使用了在 S6 定义的变量 a,d,所以边<S6, S7>上的标签有(a,a)、(d,a)。可以看到,这里的 PDG 不考虑参数结点,把参数和语句之间的数据依赖聚合在入口结点和语句之间的依赖,并只用一条边连接起来,在其上作标签记录依赖关系。两语句之间用一条边连起来,并在其上作标签记录依赖关系。

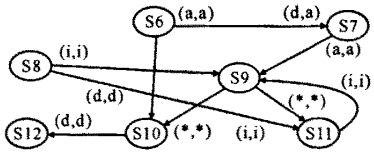

图 1 方法 multiply 的 PDG

2 Java 程序的依赖性分析

在 Java 中,参数能通过值或引用传递,值传递的参数叫做 in 参数;对于引用传递的参数,如果在被调用函数中仅被写,它是一个 out 参数,如果仅被读它是一个 in 参数,否则是 in-out 参数。非局部变量被看作一个形参与实参同名的参数,如果在执行过程中此变量的值可能被改变,则作为一个 in-out 参数,否则作为 in 参数。return 值被处理为一个 out 参数,名字与方法名相同。总之,有 3 种类型的形数:in 参数,out 参数,in-out 参数。in 参数仅能读,out 参数仅能写,in-out 参数能读能写,所以仅仅 in-out 参数和 out 参数可能对外部调用者有影响。这样,我们可以独立地分析每个方法来获得其参数(形参)之间的依赖关系。

JavaSDG 使用参数间的依赖集描述参数之间的依赖关系(在以前的 SDG 中 Summary 边表示这些依赖关系)。

定义 3 设 y 是方法 M 中 out、in-out 参数,它的参数依赖集可表示为 Dep-P(M,y)={x|x 是参数,且 y 依赖于 x};方法 M 的参数依赖集可表示为 Dep-P(M)={(y,Dep-P(M,y))|y 是方法 M 中 out、in-out 参数}。

例如:在前述的 Java 程序中,Dep_P(multiply,a)={a,d}, Dep_P(multiply,d)={a,d}, Dep_P(Move)={(a,{a,d}),(d,{a,d})}。

JavaSDG 方法对外只需要提供集合 Dep-P(M),它内部的数据依赖关系对调用程序是隐蔽的。这样处理的优点主要在于:(1) 如果改变了程序中某一方法,只需要重新分析它的 PDG 图,如果其参数间的依赖关系发生变化,则继续分析它的所有调用点;(2) 在分析整个程序 SDG 时,方法之间可以并行处理。

对于被语句 s 调用的方法，JavaSDG 根据形参和实参之间的关系能够获得集合 Def(s)和 Def(s, x)[6]。经过以上处理后，消息驱动的调用语句能像其它语句一样容易处理。

当分析一个方法 M 时，必须知道被 M 调用的方法的参数之间的依赖关系，所以这些被调用方法应该用给定的顺序进行分析，这可以通过使用调用图做到[7,8]。

Java 是一种静态类型的面向对象语言，对象的可能类型能静态的决定。JavaSDG 通过参数间依赖集表示多态，每个集合和一个对象类型相对应。当一个对象调用一个方法时，根据对象类型是否已决定，分两种情况计算参数间依赖集。

情况 1 对象类型已确定：如果一个方法不是多态方法，那么不需要更多的分析，否则，如果有必要的话，就把多态方法转换到相应的方法，重新分析多态方法的依赖关系。

情况 2 对象类型没有确定：如果一个方法不是一个多态方法，那么参数间依赖关系是所有可能方法的依赖关系的并集。否则对于每一个可能类型，把多态方法转换为相应的方法，并重新分析依赖关系，参数间依赖关系是可能方法依赖关系的并集。

3 两种程序分析方法的对比

在分析本文方法构造的 JavaSDG 与用传统方法构造的 SDG 的不同之处之前，我们首先列出与构造依赖图有关系的变量(见表 1)，这些变量摘录于文献[5]。

表 1 与 SDG 有关的变量[5]

变量名称	变量含义
Vertices	在一个方法中谓词和赋值语句结点的最大数目
Edges	在一个方法中边的最大数目
Params	在任意方法中形参的最大数目
Globals	在系统中全局变量的数目
InstanceVars	在一个类中实例变量(数据域)的最大数目
CallSites	在一个方法中调用点的最大数目
TreeDepth	继承树的深度，即确定间接调用的可能数目
Methods	在一个系统中所有方法的数目

对 ParamVertices 给出一个值，ParamVertices 值的定义为

$$ParamVertices = Params + Globals + InstanceVars, \quad (1)$$

可以基于 ParamVertices 值计算出方法依赖图顶点数目的上界：

$$Size(m) = O(Vertices + CallSites * (1 + TreeDepth * (2 * ParamVertices(m))) + 2 * ParamVertices(m)). \quad (2)$$

如果已知系统中的方法的数目，那么可以得到包含很多类 Java 程序的 SDG 的顶点数的可能最大数目[5]：

$$Size(SDG) = O(Size(m) * Methods). \quad (3)$$

前述 Java 程序段的方法 multiply 及调用点 S5 与方法入口 S6 之间的参考传递(没有分析 summary 边)的 SDG 见图 2。

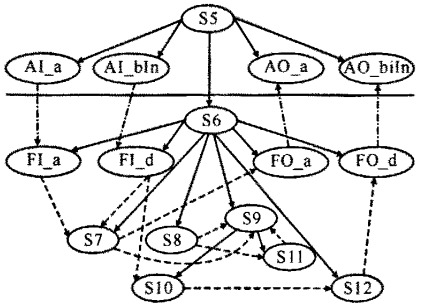

图 2 方法 multiply 及有关参数传递的部分 SDG

JavaSDG 如采用表 1 相同变量来考虑构造问题，其原理与构造 SDG 的原理一样[5]，因而公式同(1)式、(2)式和(3)式，只是公式中除了变量 Vertices、CallSites、TreeDepth、Methods 的值不变外，变量 Params、Globals、InstanceVars 在 JavaSDG 中这些值小了很多，故构造 JavaSDG 的结点要少很多(详见图 1)。

基于 SDG 和基于 JavaSDG 这两种程序分析方法的具体区别如表 2 所示。

表 2 两种程序分析方法的区别

程序分析方法	构造依赖图的区别	并发构造的区别
基于 SDG 的方法	必须建造复杂的 SDG，容易出错，结点多	每个方法 PDG 不是独立的，必须按照一定的顺序构造，因而不能并发构造
基于 JavaSDG 的方法	不必建造复杂的 SDG，减少了出错的可能性，结点少	还能通过参数之间的依赖关系，把方法间的依赖分析转换到方法内分析而且每个 PDG 是独立的，因此能够并发构造，从而提高了分析的效率

4 结束语

本文基于 JavaSDG 的描述方法，首先给出 JavaSDG 中 PDG 的形式化定义，然后分析在 JavaSDG 中 Java 程序方法调用引起的参数之间的依赖关系。与传统构造 SDG 的方法相比，本文基于 JavaSDG 的 Java 程序分析的方法不必建造复杂的 SDG，减少了出错的可能性。而且本文的方法能通过参数之间的依赖关系，把方法间的依赖分析转换到

方法内分析，而且每个 PDG 是独立的，因此能够并发构造，从而提高了程序分析的效率。

参考文献：

[1] Horwitz S, Thomas Reps, David Binkley. Interprocedural slicing using dependence graphs [J]. ACM Transactions on Programming Languages and System, 1990,12(1): 26-60.

[2] Larsen L, Harrold M. Slicing object-oriented software: proceedings of the Internation Conference on Software Engineering (ICSE-18) [C]. Berlin: IEEE Computer Society Press, 1996: 495-505.

[3] Zhao J. Applying program dependence analysis to java software: proceedings of Workshop on Software Engineering and Database Systems[C]. Taiwan: IEEE Computer Society Press, 1998: 162-169.

[4] Liang D, Harrold M J. Slicing objects using system dependence graphs: proceedings of the 1998 International Conference on Software Maintenance[C]. Bethesda: IEEE Computer Society Press, 1998: 358-367.

[5] Walkinshaw N, Roper M, Wood M. The Java system dependence graph: proceedings of the Third IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'03)[C]. Washington: IEEE Computer Society Press, 2003:55-64.

[6] Chen Zhenqiang, Xu Baowen. Slicing object-oriented java programs[J]. ACM SIGPLAN Notices, 2001, 36 (4): 33-40.

[7] Frank Tip, Jens Palsberg. Scalable propagation-based call graph construction algorithms[J]. ACM SIGPLAN Notices, 2000,35(10): 281-293.

[8] Xu Baowen, Zhang Ting, Chen Zhenqiang. Dependence analysis of recursive subprograms and it's applications [J]. Chinese Journal of Computer, 2001, 24(11): 1278-1285.

（责任编辑：韦廷宗）

（上接第 18 页）

当 $m = 3$ 时，$P_3 \otimes P_n$ 的连通度为 3，顶点集 $S_i = \{V_{1,i}, V_{2,i}, V_{3,i}\}(2 \leq i \leq n-1)$ 构成 $P_3 \otimes P_n$ 的点割集，并且 $|S_i| = 3$，$G[S_i]$ 的补图为分支 K_1 与 $K_{1,1}$. 利用引理 1，分解得到 2 个 D_1，$n-3$ 个 D_2. 对于每个 D_1，分别以 $\{V_{1,2}, V_{2,1}, V_{2,2}\}, V_{1,n-1}, V_{2,n}, V_{2,n-1}$ 为完全点割集，分解得到 4 个 K_3 与 2 个 $K_1 + C_3$. 对于 D_2，分别以 $S_i = \{v_{1,i}, v_{3,i+1}, v_{2,i}, v_{2,i+1}\}(2 \leq i \leq n-1)$ 为点割集. 由于的 $G[S_i]$ 的补图为分支 K_1 与 K_2，再利用引理 1，对 D_2 分解得到 $2(n-3)$ 个 $K_1 + K_4$，故 $TW(P_3 \otimes P_n) = \max\{TW(K_3), TW(K_1 + C_3), TW(K_1 + K_4)\} = TW(K_1 + K_4) = 4$.

参考文献：

[1] Robertson N, Seymour P D. Graph minors Ⅰ: excluding a forest [J]. Comb Theory (B), 1983, 35: 39-61.

[2] Lin Yixun. Decomposition theorems for the tree width of graphs[J]. Journal of Mathematic Study, 2000, 33 (2): 113-120.

[3] 卜长江，高振滨，齐玉梅. 网图 $F(m; n_1, n_2, \cdots, n_m)$ 的 K-优美性[J]. 哈尔滨工程大学学报，1995(3): 102-105.

[4] 杨爱民，林诒勋. 图的 min-max 型最优消去顺序问题 [J]. 系统科学与数学，1997, 17(4): 354-361.

（责任编辑：尹 闯）