

一种包含异常处理的粗粒度切片方法

郝 杰

(徐州医学院 医学信息学院, 江苏 徐州 221004)

摘 要: 在构造面向对象带有异常处理的系统依赖图的基础上, 分析了异常处理结构对程序依赖性的影响, 并且为了得到粗粒度程序切片, 提出了一种把异常处理结构转化为一个类内部私有方法的有效方法来简化带有异常处理结构的系统依赖图. 然后在简化的带有异常处理结构的系统依赖图上利用图可达性算法实现了包含异常处理结构的粗粒度切片. 最后, 通过一个切片标准对粗粒度切片和细粒度切片进行了比较. 结果证明粗粒度切片完全包含细粒度切片, 这样就降低了计算切片的复杂度.

关键词: 程序切片; 粗粒度切片; 系统依赖图; 简化的系统依赖图; 异常处理

中图分类号: TP311

文献标识码: A

文章编号: 1000-7180(2012)07-0116-05

An Approach to Calculating Coarse-Grained Slice with Exception Handling

HAO Jie

(College of Medical Informatics, Xuzhou Medical College, Xuzhou 221004, China)

Abstract: Based on constructing Object-Oriented system dependence graph with exception handling, this paper analyzes the effects of exception handling constructs on program dependence. In order to get coarse-grained slice, this paper presents a practical approach that convert exception structure to private methods inside a class to simplify the system dependence graph with exception handling constructs. And using graph reachability algorithm on the simplified system dependence graph with exception handling constructs can get coarse-grained slice with exception structure. Finally, we compared coarse-grained slice with fine-grained slice through a slice standard. From the comparison, we can see that coarse-grained slice completely contains the fine-grained slice. Thus we succeed in reducing the complexity of calculating slice.

Key words: program slicing; coarse-grained slice; system dependence graph; simplified system dependence graph; exception handling

1 引言

程序切片可以用于程序理解、程序分析、逆向工程、测试等多个方面. 自从 Mark Weiser^[1] 提出程序切片的概念以来, 面向过程语言的切片技术得到了广泛地研究. 面向对象的程序切片由于具有很多特点, 如: 类, 对象, 继承, 多态以及动态联接, 许多学者也相继提出了对面向对象程序切片的各种算法. 郝

学娇等人^[2]提出了针对面向对象并发程序的一种基于依赖标识的动态切片方法, 但这些方法构造系统依赖图(SDG)的过程非常复杂, 因此人工构造 SDG 和分析各种依赖关系(尤其是数据依赖)几乎不可能完成. 李必信^[3]等人提出了简化系统依赖图的静态粗粒度切片方法, 戚晓芳等人提出了并发程序的粗粒度切片方法^[4], 但是并没有考虑异常结构对切片的影响. 由于异常处理的特殊性, throw 语句可能会

收稿日期: 2011-11-11; 修回日期: 2011-12-12

基金项目: 国家自然科学基金(60970032); 教育部科学技术研究重点项目(108063); 江苏省自然科学基金(BK2008124); 江苏省“青蓝工程”

影响程序执行的控制流,所以在切片中就应该考虑异常处理所带来的数据和控制依赖关系。

本文提出了一种包含异常结构的计算面向对象的粗粒度切片方法.以一个包含异常处理结构的C++程序为例,首先构造了包含异常处理结构的系统依赖图(ESDG),然后提出把异常处理结构从语句级向方法级转换的方法,并在此基础上完成对包含异常处理的简化系统依赖图(SES DG)的构造.最后,根据李必信^[3]的方法在 SES DG 上得到了包含异常处理结构的粗粒度切片,并把语句级的细粒度切片和方法级的粗粒度切片进行了比较。

2 包含异常处理的系统依赖图(ESDG)

虽然 Sinha^[5]提出了异常结构的表示方式,姜淑娟^[6]等人对此进行了改进,但他们的研究都局限于单个类内部方法间的异常传播,张艳梅^[7]通过分析类簇级测试中的异常传播对程序数据依赖的影响,提出一种包括异常结构在内的类间C++程序数据依赖分析方法,并完成了系统依赖图的构造.根据上述提出的方法,可以完成对图1所示程序的系统依赖图的构造.构造的系统依赖图如图2所示。

通过图2可以清楚地分析出两个包含异常结构并存在继承关系的两个类之间的各种依赖关系.在类B中的函数m()调用了类A中的函数f(),因此在图2中对于函数的调用有两条返回边:一条是正常返回边,一条是异常返回边.如果在f()中没有抛出异常,那么就沿正常返回边返回(从 norm_exit 结点到 norm_return 结点),并且返回到函数的调用点;如果在f()中抛出异常,那么将沿异常返回边返回(从异常退出结点到相应的 Catch 结点),并且在m()中寻找匹配的处理程序.在图2中把 throw 语句和 catch 语句作为一个断言来处理,有“T”边和“F”边,这是因为 throw 语句和 catch 语句能够引起控制流的变化.通过程序依赖图不仅可以分析异常结构引发的控制流的变化,同样可以分析出异常结构对数据流的影响.在图2中把全局变量s作为函数的参数来处理,如果在f()中没有抛出异常,那么S29行的s值数据依赖于S13行对s的定义,输出s=1,因此在图2中增加了数据依赖边:从函数f()的 norm_exit 结点的后继结点 s_out=s 到函数 m()的 norm_return 结点的后继结点 s=s_out.如果在f()中抛出了异常,那么S29行的s值数据依赖于S22行对s的定义,输出s=0,因此在图2中增加了数据依赖边:从函数f()中异常退出结点的后继结点 s_

out=s 到函数 m()中 catch 结点的后继结点 s=s_out.

<pre># include<iostream, h> class Ex1{}; class Ex2:public Ex1{}; int s; CE1 class A{ public:int a,b; public: ME2 A(){ S3 a=0; S4 b=0;} ME5 virtual vm{ S6 a=a+b; } ME7 int f(int a,int b){ int c; S8 if(b==0) S9 throw Ex1 (); S10 else if(b<0) S11 throw Ex2(); S12 c=a/b; S13 s=1; S14 return c; } }</pre>	<pre>CE15 class B:public class A{ public:int c; public: ME16 B(){ S17 c=0;} ME18 virtual vm(){ S19 c=c-b;} ME20 void m(int x,int y){ S21 int z=1; S22 s=0; C23 try{ C24 z=f(x,y); S25 cout<<z<<endl; } C26' catch (Ex1 & e1){ S27 cout<<s<<endl; } C26' catch(Ex2 & e2){ S28 cout<<s<<endl; } S29 cout<<s<<endl; } } ME30 void main(){ S31 class B b=new b(); C32 b.m(2,3);}</pre>
---	--

图1 程序示例

在图2中,把异常处理结构引入到语句层次的系统依赖图中,得到了准确的控制依赖和数据依赖信息,但是增加了构造系统依赖图的复杂度.下面在李必信^[3]提出的简化系统依赖图的基础上完成对包含异常处理的系统依赖图的简化。

3 简化包含异常处理的系统依赖图的方法

为了得到包含异常处理的粗粒度切片,需要对ESDG进行简化.李必信等人^[3]曾提出简化SDG的原则,他把对语句级描述结点和边的细节问题上升到了方法级的表示,这样使两个方法的参数结点之间的数据依赖关系用方法之间的数据依赖来表示.因此,简化的SDG就只需用过程入口结点表示即可,而不必展开表示.但是对异常处理结构的分析,

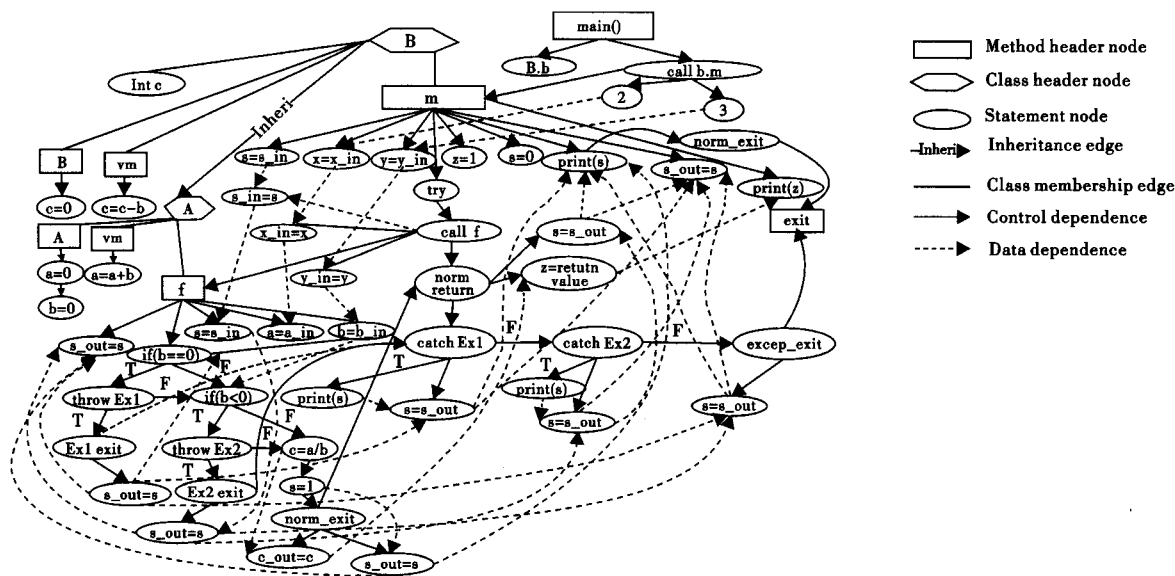


图2 程序的系统依赖图(ESDG)

以前的研究都是基于语句级的,无法进行简化,因此,需要把异常处理结构转化到方法级上。

3.1 异常处理结构向方法级的转换

为了表示 try/catch/throw 程序结构这种语句复杂的跳转关系,并把这种语句级的关系转换到方法级的关系上,我们需要通过以下的原则进行转换:

(1) 把每个 try/catch 看成一个过程,过程体由原有块中的语句构成;

(2) 除了在自己相应块中定义的变量外,其它在每个块中起作用的变量,都视为全局变量;

(3) 对每个可能抛出异常的方法,增加两个(或多个)形式输出参数:一个是布尔类型,表示是否抛出了异常;另一个是异常类型(如果抛出了多个异常,则增加相应的参数表示被抛出的异常);

(4) 添加一个 return 结点,把可能抛出异常方法的各种退出结点与此结点相连,表示各种可能的返回. 并把这个结点表示为断言结点,如果抛出了异常,则标号为 False 的边连接到相应的 catch 调用函数;如果没有抛出异常,则标号为 True 的边连接到 norm_return 结点. 并且在这个结点上建立相应的实际输出参数结点表示抛出的异常;

(5) 当有多个抛出的异常和多个异常处理的 catch 块时,这些 catch 块要被看作可重载的 catch 方法. 具体调用哪个 catch 方法根据抛出的异常类型而定。

根据以上转换原则,把图 2 所示的包含异常处理的方法 m() 和 f() 的 SDG 转换为图 3 的表示。在

图 3 中,将 try/catch 块看作和方法 m() 属于同一个类的私有方法,并且在方法 m 中调用了这几个方法. 变量 x, y, s 对于 try/catch 块都视为全局变量. 由于方法 f 可能会抛出异常,所以需要增加三个形式输出参数,其中 E1_out=E1, E2_out=E2 表示抛出异常的类型, F_out=F 表示是否会抛出异常. 在图 3 中,增加了 return f 结点与方法 f 中的 Ex1_exit, Ex2_exit 和 norm_exit 结点相连,来表示方法 f 各种可能的返回. 并且在此结点上,建立两个参数结点表示异常的抛出,其中 E=E_out 为异常类型, F=F_out 为布尔类型,当值为 True 表示没有异常抛出,否则表示有异常抛出. 根据转换原则,把此结点表示为断言结点,标号为 True 的边(表示没有异常抛出)连接到 norm_return 结点;相应地,标号为 False 的边(表示抛出了异常)连接到 call catch 结点. 在程序中有多个处理异常的 catch 块,按照转换原则就把它们看作可重载的 catch 方法,具体调用 catch(Ex1 & e1) 还是 catch(Ex2 & e2) 就依据抛出的异常类型而定。

经过这样的转换,就已经成功地把异常处理结构转换到了方法级上. 下一步就是如何简化带有异常结构的系统依赖图。

3.2 简化包含异常处理的系统依赖图

以李必信^[3]提出的简化系统依赖图的方法为基础,省略那些表示输入参数和输出参数的结点并扩大每个方法入口结点的含义,用过程入口结点表示过程依赖图,用方法间的数据依赖关系表示两个方法的参数结点间的数据依赖关系。

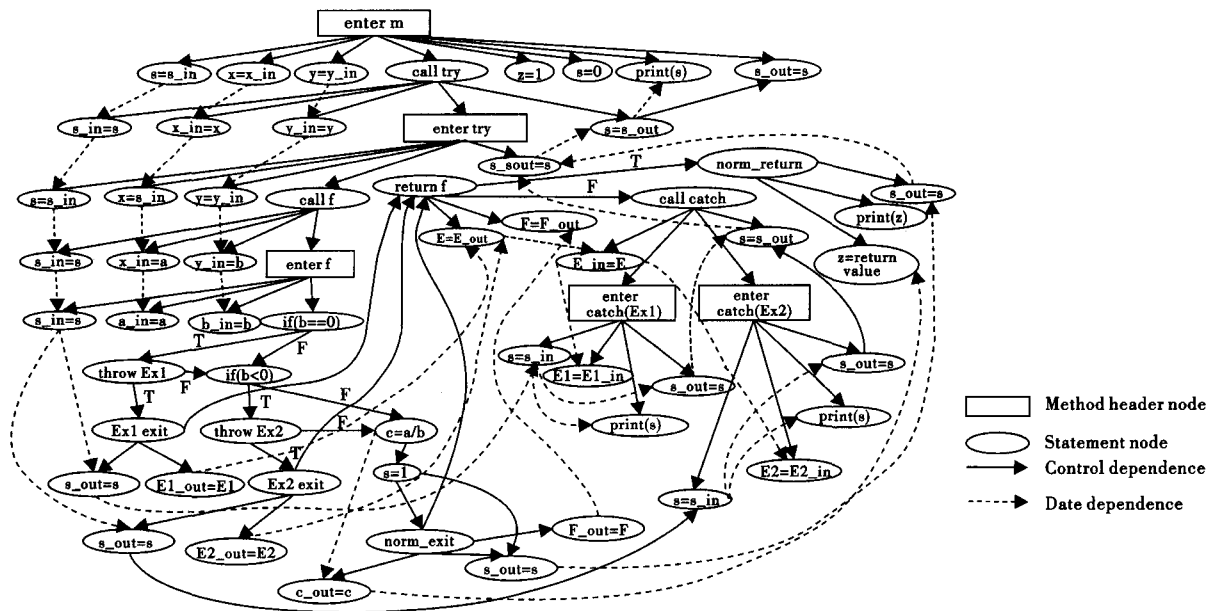


图 3 方法 m() 和 f() 转换后的 ESDG

为了得到 SESDG, 把图 1 程序中标识为 C23 和 C26 结点都一分为二, 分别为: C23/26 (表示一个调用语句), ME23/26 (表示一个方法). 这是因为我们把 try/catch 语句转换为了 try/catch 方法, 并把 try/catch 方法看作和 m() 方法同属于一个类的私

有方法, 因此在 SESDG 中把 C23 和 C26 作为 ME20 方法入口结点下的一个调用 try 方法和调用 catch 方法的语句结点, 同时把 ME23 和 ME26 作为 try 和 catch 的方法入口结点. 图 4 即为图 1 程序的 SESDG.

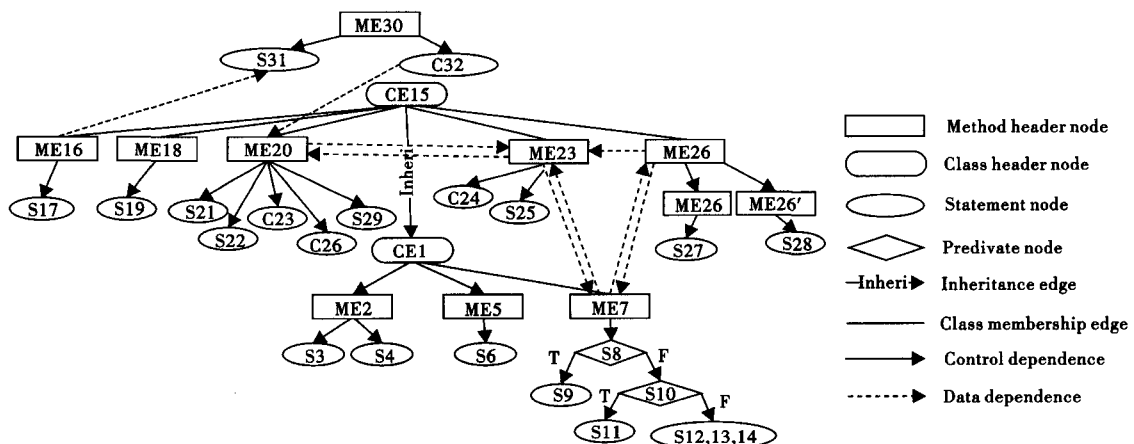


图 4 程序的 SESDG

从图 4 中可见 SESDG 中的结点分别由类头结点、方法/过程头结点、谓词结点和语句结点构成, 而在图 2 的 ESDG 中, 除了以上结点外, 在每个方法入口结点或调用/返回语句结点下会有大量的输入、输出参数结点和全局变量结点. 由于在 SESDG 中省略了参数结点, 因此在 ESDG 中各个参数结点之间的数据依赖关系就由方法间的数据依赖关系取代. 如图 4 所示的从 ME23 到 ME7 的方法入口结点

的数据依赖取代了图 2 中的 call f 结点下的实参结点 $x_{in}=x$ 和 $y_{in}=y$ 到 enter f 结点下的形参结点 $a=a_{in}$ 和 $b=b_{in}$ 的数据依赖. 从 ME7 到 ME23 的方法入口结点的数据依赖取代了图 2 中 norm_exit 结点下的形参结点 $c_{out}=c$ 和 $s_{out}=s$ 到 norm_return 结点下的实参结点 $s=s_{out}$ 和 $z=return\ value$ 的数据依赖. 这样大大降低了构造系统依赖图的复杂性.

4 切片结果

构造系统依赖图是实现程序切片的关键,从系统依赖图的某一个结点出发,沿边进行遍历就可以得到切片.因此,系统依赖图对程序表示的准确程度会直接影响到程序切片的准确程度.如果对简化的SDG进行切片计算,那么得出来的结果就应该是粗粒度的程序切片.依据传统的程序切片算法和李必信^[3]提出的粗粒度切片算法分别对图2的ESDG和图4的SESDG进行切片,切片标准为 $\langle S27, s \rangle$.切片结果如表1所示.

表1 切片结果比较

切片依据	切片结果	切片粒度
ESDG (图2)	ME7, S8, S9, S10, S11, ME20, S22, C23, C24, C26, S27, ME30, C32	细粒度
SESDG (图4)	ME7, S8, S9, S10, S11, S12, S13, S14, ME20, S21, S22, C23, ME23, C24, S25, C26, ME26, S27, S29, ME30, C32	粗粒度

把粗粒度切片结果表示为集合 U , 把细粒度切片结果表示为集合 V , 从表1可见 $V \subseteq U$. 因此, 从切片结果可见, 依据 SESDG 进行粗粒度切片计我们把粗粒度切片结果表示为集合 U , 把细粒度切片结果表示为集合 V , 从表1可见 $V \subseteq U$. 因此, 从切片结果可见, 依据 SESDG 进行粗粒度切片计算, 结果只是导致了不精确, 而不会导致错误. 这证实了李必信^[3]对粗粒度切片的描述.

5 结束语

程序切片是一种重要的程序分析技术, SDG 是计算切片的依据, 但是构造 SDG 特别是面向对象的

并且包含异常处理的大规模程序非常困难. 在李必信^[3]简化系统依赖图的基础上, 文中通过对异常结构的变形成功得到了带有异常结构的简化的系统依赖图 (SESDG), 并且利用 SESDG 得出了带有异常结构的粗粒度切片. 通过对细粒度切片和粗粒度切片的结果比较, 发现粗粒度切片完全包含细粒度切片, 成功地降低了利用 ESDG 计算切片的复杂度.

参考文献:

- [1] Weiser Mark. Program slicing[J]. IEEE Transactions on Software Engineering, 1984, 10(4): 352-357.
- [2] 郝学蛟, 汤小春. 基于依赖标识的并发程序动态切片方法[J]. 微电子学与计算机, 2009, 26(7): 5-8.
- [3] 李必信, 王云峰, 张勇翔, 等. 基于简化系统依赖图的静态粗粒度切片方法[J]. 软件学报, 2002, 12(2): 204-211.
- [4] 戚晓芳, 徐宝文. 一种粗粒度并发程序切片方法[J]. 微电子学与计算机, 2004, 21(10): 1-3.
- [5] Sinha Saurabh, Harrold Mary Jean. Analysis of programs with exception-handling constructs[C] // Proceedings of the International Conference on Software Maintenance, Washington, DC, USA: IEEE, 1998: 348-357.
- [6] 姜淑娟, 徐宝文, 史亮, 等. 一种基于异常传播分析的依赖性分析方法[J]. 软件学报, 2007, 18(4): 10.
- [7] 张艳梅, 姜淑娟, 袁冠. 一种包含异常传播的类间数据依赖分析方法[J]. 软件时空, 2010, 26(2/3): 201-203.

作者简介:



郝杰女, (1980-), 硕士, 讲师. 研究方向为程序设计语言、异常处理.