ISSN 1000-1239/CN 11-1777/TP
45(8): 1408-1416, 2008

横切侵入性和横切不变性

吕 嘉[1] 应 晶[1,2] 吴明晖[1,2] 蒋 涛[1]
[1](浙江大学计算机科学与技术学院 杭州 310027)
[2](浙江大学城市学院计算机与计算科学学院 杭州 310015)
(samlv2000@163.com)

Crosscutting Invasion and Crosscutting Invariant

Lü Jia[1], Ying Jing[1,2], Wu Minghui[1,2], and Jiang Tao[1]
[1](*College of Computer Science and Technology, Zhejiang University, Hangzhou* 310027)
[2](*School of Computer and Computing Science, City College of Zhejiang University, Hangzhou* 310015)

Abstract Owing to the characteristics of quantification and obliviousness of aspect-oriented language, modular behavioral analysis and modular reasoning are more difficult than that of the traditional paradigms. To deal with crosscutting safety and crosscutting quality in aspect-oriented language, crosscutting modules and affected modules are constrained with pre-conditions and post-conditions, but assigning blame for pre-condition and post-condition failures during the process of crosscutting poses subtle and complex problems. To analyze behavioral effect of a crosscutting concern, the programmer should consider the aspect itself and the part of the system it affects. Furthermore, when several aspects are woven at a same pointcut, the analysis of possible dangerous interferences becomes more complex. Similar to the notion of behavioral subtyping in object-oriented language, a notion of crosscutting invariant is proposed. In order to check the behavioral errors of violating crosscutting invariability and four other simple behavioral errors, an algorithm based on software behavioral contracts is proposed. To formalize this algorithm, crosscutting contract calculus and a set of contract elaboration rules are presented. The contract soundness theorem which ensures the correctness of the contract elaboration process is stated and proved. An example is also represented to show how to use these contract elaboration rules to check and analyze the behavioral errors.

Key words aspect-oriented language; crosscutting safety; crosscutting quality; crosscutting interference; modular analysis

摘 要 由于面向方面语言的不知觉性和多量化特点,模块分析和模块推理比传统方法学更加困难.为了解决面向方面语言的横切安全和横切质量问题,使用前提条件和后验条件约束横切模块和被横切模块,然而在横切过程中寻找前提条件和后验条件的失败原因十分微妙和复杂.为了分析一个横切关注点的行为影响,程序员需要考虑方面本身和这个方面影响的系统其他部分.当几个方面编织在同一个切入点,危险干扰分析变得更加复杂.类似面向对象语言中的行为子类型概念,引入横切不变性概念.为了检查由于破坏横切不变性引起的行为错误和其他4种简单行为错误,基于软件行为契约提出一个横切不变性检测算法.为了形式化这个算法,提出 Crosscutting Contract 演算和一组契约求解规则,并通过定义和证明契约完备性来保证契约求解过程的正确性.还使用一个例子说明如何使用这些契约求解规则检测和分析行为错误.

收稿日期:2007-06-04;**修回日期**:2008-01-15
基金项目:霍英东教育基金会资助项目(94030)

关键词　面向方面语言;横切安全;横切质量;横切干扰;模块分析

中图法分类号　TP311.1

面向方面语言[1]的出现,有效地模块化了横切关注点,解决了由于关注点分离引起的代码分散和代码混杂问题.由于横切不知觉性和多量化[2],面向方面语言的模块行为分析和模块推理比传统语言更加复杂.当程序员实现一个横切关注点,必须保证其余程序部分不受这个横切关注点的干扰,目前的主流面向方面语言还没有机制有效支持这个要求,引起横切关注点实现与系统其他部分紧耦合,从而降低面向方面软件系统的质量和安全性.

许多学者研究在保持面向方面语言基本特性的前提下,通过维护传统的模块封装和信息隐藏原则来解决横切安全和横切质量问题.文献[3-5]通过扩展被横切模块的封装机制来保证被横切模块的封装性;文献[6-7]通过改进切入点语言避免或者约束横切关注点的侵入特性,面向方面语言的基本特性和传统的模块封装和信息隐藏原则存在着矛盾;文献[8]认为面向方面语言存在着是似而非的胜利:面向方面语言总是在模块化和结构化代码的同时又违反代码模块化和结构化原则.

对于一种语言仅仅强调独立模块的封装性和质量是不够的,模块组合能力和质量同样重要,而模块组合能力的上升往往导致模块局部性和封装性的下降.面向方面语言引入更加强大的模块组合能力,同时也引入复杂的模块行为干扰问题,为了片面追求传统的模块封装和信息隐藏原则,过度牺牲语言的模块组合能力是得不偿失的.

本文借鉴面向对象语言中行为子类型概念[9-10]及其相关度量和检测方法[11-12],给横切模块和被横切模块附加行为契约[13],基于横切不变性概念对方面横切过程进行约束,从而解决面向方面语言的横切安全和横切质量问题.

1　横切侵入性

图1的例子使用AspectJ[14]进行描述.程序由类 A、方面 *Crosscut*1,*Crosscut*2 和客户程序 M 组成.方面 *Crosscut*1 和 *Crosscut*2 定义相同的切入点 *set*1 和相应的通知;客户程序 M 声明句柄 a 使用参数 $x=5$,$y=0$ 调用构造函数实例化类 A,然后使用参数 $x=10$ 调用方法 setX. 由于方面 *Crosscut*2 中声明横切优先级 *Crosscut*1>*Crosscut*2,程序先触发方面 *Crosscut*1 的执行前通知,此时满足通知的先验条件函数参数 $x=10>5$,通知调用 setY 方法将 y 赋值为10;然后再触发方面 *Crosscut*2 的执行前通知,此时满足通知先验条件 $a.\ getY()>5$;最后执行类的方法 $setX()$,此时无法满足方法的先验条件 $getY()<9$.

```
public class A {
  int x,y;
  public A(int x,int y) { this.x=x; this.y=y;}
  public int getX() {return this.x; }
  public void setX(int x){this.x=x; }@pre { getY()<9}
  public int getY(){return this.y; }
  public void setY(int y){ this.y=y; }
}
public aspect Crosscut1 {
  pointcut set1(A a, int x):
    execution(void A.setX(int))&&target(a)&&args(x);
    before(A a,int x):set1(a,x){
      a.setY(x);
      ⋮
    }@pre {x>5}
}
public aspect Crosscut2{
  declare precedence: Crosscut1,Crosscut2;
  pointcut set1(A a, int x):
    execution(void A.setX(int))&&target(a)&&args(x);
  before(A a, int x):set1(a,x){
    ⋮
  }@pre{a.getY()>5}
}
Client program M
  A a=new A(5,0);
  a.setX(10);
```

Fig. 1　An AspectJ example.

图1　一个 AspectJ 例子

例子中由于横切引起的行为干扰主要有两个:1) 方面 *Crosscut*1 的优先级高于 *Crosscut*2,方面 *Crosscut*1 先执行将 y 赋值为10,再执行方面 *Crosscut*2,从而满足 *Crosscut*2 通知的先验条件 $getY()>5$,即方面 *Crosscut*2 的正确执行依赖于方面 *Crosscut*1 的执行.2) 由于方面 *Crosscut*1 的方法执行前通知将 y 赋值为10,使得类 A 的方法 $setX()$ 的先验条件 $getY()<9$ 无法满足而发生行为错误,

即 *setX*()方法的行为正确性受到方面 *Crosscut*1 的影响.

从上面的例子可以看出,横切模块和被横切模块之间会发生相互干扰,横切模块之间也会发生相互干扰,为了把这一类行为问题和其他行为问题区别开来,本文把这一类行为问题称为横切侵入性.

2 横切不变性和横切不变性判定算法

为了解决横切侵入性问题,本文引入横切不变性概念对横切行为进行约束.

定义 1. 当一个软件模块受到几个横切关注点的影响,如果每一个横切关注点保持原有软件模块的行为约束,则横切过程满足横切不变性.

如果程序在执行过程中违反横切不变性约束,则程序发生横切不变性错误. 由于某些横切关注点的行为错误被其他横切关注点的执行所掩盖,发生这一类错误的原因十分复杂. 这一类行为错误直接违反面向方面语言的横切不知觉性和横切多量化,属于系统的核心行为错误. 同时由于横切不知觉性和横切多量化,这种行为错误具有很大的不确定性和扩散性,严重影响面向方面软件系统的可靠性和质量. 由于涉及与横切相关的所有模块,这种行为错误的检查和分析比较复杂,不但要检测和分析与横切相关的所有模块,而且还要考虑横切发生的整个过程.

除了检测和分析横切不变性错误,还需要检测和分析 4 种简单的行为错误:被横切模块先验错误、被横切模块后验错误、横切模块先验错误和横切模块后验错误. 这些行为错误与传统的模块行为错误是一致的,属于系统的枝节性行为错误. 这些行为错误的查找和分析仅仅涉及独立的软件模块及其部分程序上下文.

为了检查以上 5 种行为错误,本文给出一种横切不变性检测算法(本算法不考虑行为子类型约束),如图 2 所示. 要保证横切不变性,对于多个横切同一个切入点的方法执行前通知,优先级高的通知先验条件必须蕴含优先级低的通知先验条件,后验条件必须蕴含方法的先验条件;对于多个横切同一个切入点的方法执行后通知,优先级低的通知先验条件必须蕴含优先级高的通知先验条件,后验条件必须蕴含方法的后验条件,否则引起横切不变性错误.

Step1. When a method executes, if the pre-condition of this method is false, then go to setp19.
Step2. If the pre-condition of this method is true, then the before advice which is waiting for execution and has highest precedence is to execute.
Step3. If the pre-condition of the selected before advice is false, then go to Step21.
Step4. If the pre-condition of the selected before advice is true and any pre-condition of other before advice which is waiting for execution is false, then go to Step23.
Step5. The selected before advice executes.
Step6. If the post-condition of selected before advice is false, then go to Step22.
Step7. If the post-condition of selected before advice is true and the pre-condition of the method is false, then go to Step23.
Step8. If there are still other before advices which have not executed, then go to Step2.
Step9. The affected method executes.
Step10. If the post-condition of the method is false, then go to Step20.
Step11. If the post-condition of the method is true, then the after advice which is waiting for execution and has lowest precedence is to execute.
Step12. If the pre-condition of the selected after advice is false, then go to Step21.
Step13. If the pre-condition of the selected after advice is true and any pre-condition of other after advice which is waiting for execution is false, then go to Step23.
Step14. The selected after advice executes.
Step15. If the post-condition of selected after advice is false, then go to Step22.
Step16. If the post-condition of selected after advice is true and the post-condition of the method is false, then go to Step23.
Step17. If there are still other after advices which have not executed, then go to Step11.
Step18. The program continues to execute.
Step19. Throw a pre-condition error of the method.
Step20. Throw a post-condition error of the method.
Step21. Throw a pre-condition error of the advice.
setp22. Throw a post-condition error of the advice.
Step23. Throw a error of violating crosscutting invariant.

Fig. 2 Crosscutting invariant detecting algorithm.

图 2 横切不变性检测算法

3 Crosscutting Contract 演算

为了给横切不变性检测算法建立一个形式化基础,定义基于 AspectJ 核心语法的形式化模型 Crosscutting Contract 演算. Crosscutting Contract 演算由 Contract Java 演算[12] 和 Classic Java 演算[15]扩展而来,Classic Java 演算以类型系统理论[16-17]为基础.

3.1 语法和类型求解

Crosscutting Contract 演算(见附录 A)忽略复杂的 AspectJ 语法要素,仅讨论在成员方法的执行

前后附加通知.主要语法包括程序 P 由一组类、接口、方面定义和程序主方法组成.类由一组成员变量和成员方法组成,接口由一组成员方法声明组成,方面由一组静态横切、动态横切组成,静态横切可以认为是类的外部分类方式[18],动态横切包括切入点和通知定义,切入点在程序执行过程中选择特定的联结点进行横切,通知在特定的切入点附加行为.

方法体和通知体是一个任意表达式,结果是方法或者通知的返回值(通知的返回值为空).方法体和通知体可以附加先验条件或者后验条件,先验条件和后验条件是任意布尔类型的表达式,返回值是 true 或者 false.new 操作符表示实例化类.

Crosscutting Contract 演算在 Contract Java 演算的谓词和关系[12]基础上扩展了以下谓词和关系.

定义 2. $ad\ Before_P\langle a,c,m\rangle$:在程序 P 中,通知 ad 是方面 a 中类 c 的方法 m 执行前通知.

定义 3. $ad\ After_P\langle a,c,m\rangle$:在程序 P 中,通知 ad 是方面 a 中类 c 的方法 m 执行后通知.

定义 4. $e\ Pre_P\langle c,m\rangle$:在程序 P 中,表达式 e 是类 c 的方法 m 的先验条件.

定义 5. $e\ Post_P\langle c,m\rangle$:在程序 P 中,表达式 e 是类 c 的方法 m 的后验条件.

定义 6. $e\ Pre_P\langle a,ad\rangle$:在程序 P 中,表达式 e 是方面 a 的通知 ad 的先验条件.

定义 7. $e\ Post_P\langle a,ad\rangle$:在程序 P 中,表达式 e 是方面 a 的通知 ad 的后验条件.

定义 8. $a_i \leqslant_P a_j$:在程序 P 中,方面 a_i 的横切优先级高于方面 a_j.

一个 Crosscutting Contract 程序是良类型的当且仅当类定义、方面定义和最终表达式是良类型,Crosscutting Contract 演算的类型求解与 Contract Java 演算的类型求解[12]类似,本文不再论述.

3.2　契约求解和契约求值

要定义契约求解规则,首先定义一组前提性判定[12](如图 3 所示).

根据图 3 的前提性判定,Crosscutting Contract 演算的契约求解规则包括(见附录 B):$\rightarrow_P$ 规则改写整个程序 P.$\rightarrow_d$ 规则调用 $\rightarrow_m$ 规则、$\rightarrow_w$ 规则、$\rightarrow_{pre}$ 规则、$\rightarrow_{post}$ 规则和 $\rightarrow_e$ 规则改写方法和通知.$\rightarrow_m$ 规则去除方法和通知的先验条件和后验条件.$\rightarrow_w$ 规则根据方法的先验条件和后验条件生成新的包装方法,包装方法检查方法的先验错误和后验错误;对于通知,根据通知先验条件和后验条件生成新的包装通知,包装通知直接检查通知先验错误和后验错误,同时调用 $\rightarrow_{pre}$ 规则和 $\rightarrow_{post}$ 规则检测横切不变性错误.$\rightarrow_{pre}$ 规则和 $\rightarrow_{post}$ 规则根据方法和通知的行为契约的隐含关系检查横切不变性错误.$\rightarrow_e$ 规则改写表达式,将对原方法的调用转变为对包装方法的调用.

$\vdash\!\!\rightarrow P \vdash\!\!\rightarrow_P P'$　Program P is compiled to program P'.
$P \vdash\!\!\rightarrow defn \rightarrow_P defn'$　Definition defn in program P is compiled to $defn'$.
$P,c \vdash\!\!\rightarrow meth \rightarrow_m meth'$　Method meth in class c is compiled to $meth'$.
$P,a \vdash\!\!\rightarrow ad \rightarrow_m ad'$　Advice ad in advice a is compiled to ad'.
$P,c \vdash\!\!\rightarrow meth \rightarrow_w wrap\_meth'$　$wrap\_meth'$ check pre-condition and post-condition of method meth.
$P,a \vdash\!\!\rightarrow ad \rightarrow_w wrap\_ad'$　$wrap\_ad'$ check pre-condition and post-condition of advice ad'.
$P,a \vdash\!\!\rightarrow ad \rightarrow_{pre} ad_{pre}$　ad_{pre} check error of violating crosscutting invariant.
$P,a \vdash\!\!\rightarrow ad \rightarrow_{post} ad_{post}$　ad_{post} check error of violating crosscutting invariant.
$P,c \vdash\!\!\rightarrow e \rightarrow_e e'$　Expression e is compiled to e'.

Fig. 3　Predicates of Crosscutting Contract calculus.

图 3　Crosscutting Contract 演算的前提性判定

Crosscutting Contract 演算的操作语义是基于表达式和存储器二元组的上下文重写系统[15],每个求值规则的形式是 $P \vdash \langle e,S\rangle \rightarrow \langle e,S\rangle$,$store(S)$ 表示从类对象到(包含类标记)成员记录的映射,成员记录 $record(F)$ 表示从成员名称到值的映射,表达式中的自由变量由存储器中的变量绑定.Crosscutting Contract 演算的求值规则与 Contract Java 演算的求值规则[12]类似,本文不再论述.

3.3　契约完备性

对于原有的去除行为契约的 AspectJ 程序,Crosscutting Contract 程序的行为契约是附加的程序检查,由于契约表达式可以是任意的布尔表达式,它可能引起副作用(引起错误或者影响原有 AspectJ 程序的行为),因此必须限定行为契约是无副作用的.

定义 9[12]. 对于任意的存储器 S,表达式 e 是无副作用当且仅当表达式 e 的自由变量包含在 $dom(S)$ 中,并且存在一个值 v 使得 $\langle e,s\rangle \rightarrow^* \langle v,S\rangle$.

使用 Kleene 等价性定义程序等价性[12]:等价的可终止程序产生相同的结果或者相同的错误,所有非终止程序都是等价的.为了保证契约求解过程

保持原有程序的语义,同时满足各种行为约束,需要定义求解一致性.求解一致性使用平凡运算 Erase, Erase 运算仅仅去除行为契约.

定义 10[12]. Crosscutting Contract 的程序 P 的求解 T 是一致的当且仅当契约表达式是无副作用的,同时满足以下条件之一:

1) $T(P)$与 $Erase(P)$是 Kleene 等价的;

2) $\langle T(P), \varnothing\rangle \rightarrow^{*} \langle error: String, S\rangle$.

Crosscutting Contract 演算的契约求解除了方法调用和通知执行不改变任何原有表达式:如果契约表达式没有任何副作用同时求值为 true,包装方法也不会产生副作用;如果某个契约表达式求值为 false,则契约求解必定产生 5 种行为错误的一种.

定义 11[12]. 契约求解 T 是契约完备的当且仅当对于任意的 Crosscutting Contract 程序 P,它的契约表达式是无副作用的,同时满足以下条件之一:

1) 对于$\langle Erase(P), \varnothing\rangle \rightarrow^{*} \langle P', S'\rangle$的每一个状态$\langle P', S'\rangle$,$\langle P', S'\rangle$对于 Crosscutting Contract 程序 P 而言是局部契约完备的;

2) $\langle T(P), \varnothing\rangle \rightarrow^{*} \langle error: String, S\rangle$.

局部契约完备性要求对于任意的契约求解状态$\langle e, s\rangle$,e 不但保持自身的行为契约,而且必须满足横切不变性检测算法规定的特定隐含关系[15].

定理 1. Contract Crosscut 演算的求解过程是契约完备的.

定理 1 证明的大概思路[12]:对于 Crosscutting Contract 程序 P,假定求解 $T(P)$不产生任何契约错误,如果 $Erase(P)$的每一个规约步骤都是局部完备的,则求解 $T(P)$是契约完备的.

引理 1[12]. 对于任意的 Crosscutting Contract 程序 P,$T(P)$的每一个契约求解步骤与 $Erase(P)$同步.

引理正确性分 3 种情况进行论述[12]:首先考虑没有方法调用(通知触发)和方法返回(通知返回),契约求解不改变任何表达式,$Erase(P)$和 $T(P)$是同步的.其次考虑求解过程遇到第 1 次方法调用(通知触发),$Erase(P)$形式是$\langle Erase(P), \varnothing\rangle \rightarrow \cdots \rightarrow \langle E[o.m: t(v_1, \cdots, v_n)], S\rangle \rightarrow \langle E[return: t, cb(x_1/v_1 \cdots x_n/v_n)], S\rangle$,由于是第 1 次方法调用(通知触发),不再包含其他方法调用,$Erase(P)$与 $T(P)$是相同的,求解过程要调用包装方法(包装通知),如果 $T(P)$产生任何行为错误,$Erase(P)$是同步的违反行为错误;如果 $T(P)$不产生任何行为错误,因为契约表达式是无副作用的,因此包装方法不产生任何副作用.再考虑方法返回(通知返回),分析与第 2 种情况类似.因此这个引理成立.

最后使用这个引理证明契约完备性定理[12]:如果 $T(P)$发生行为错误,从引理可以知道 $Erase(P)$必定发生行为错误.如果 $T(P)$不发生任何行为错误,假设$\langle e, s\rangle$是从 $Erase(P)$开始的规约序列的某个状态,如果 e 不能分解成某些求解上下文和方法调用(通知触发),或者某些求解上下文和方法返回(通知返回),显然是局部完备的.假定 e 分解成某些求解上下文和方法调用(通知触发),求解过程调用包装方法(包装通知),从$\rightarrow_w$规则可以知道方法(通知)的先验条件满足,满足横切不变性检测算法规定的特定隐含关系,因此这一步求解是局部契约完备的.假定 e 分解成某些求解上下文和方法返回(通知返回),分析与第 2 种情况类似.通过上面 3 种情况可证明契约完备性定理成立.

4 实例分析

根据上面的契约求解规则转换图 1 的例子.根据$[defn^c]$和$[wrap^c]$规则,类 A 的 setX 方法去除先验条件,生成新的包装方法 $setX\_A$,包装方法判断 setX 的先验条件 $getY()>9$,如果满足则调用原来的 setX 方法,否则触发方法先验条件错误,如图 4 所示:

```
public void setX(int x) {
  this.x=x;
}
public void setX_A(int x){
  if(getY()<9){
    setX(x);
  }
  else{
    mdPreErr(A);
  }
}
```

Fig. 4 SetX and wrapper method of setX after transformation.

图 4 转换后的 setX 方法和 setX 包装方法

根据$[defn^a]$,$[wrap^a]$和$[pre^a]$规则,将方面 *Crosscut*1 和 *Crosscut*2 的通知生成新的包装通知和行为检查类 *Check_Crosscut*1_*pre* 和 *Check_Crosscut*2_*pre*.方面 *Crosscut*1 转换后代码如图 5 所示,新的通知先判断通知的先验条件,如果不满足则触发通知先验条件错误,如果满足则初始化通知行为检查类 *Check_Crosscut*1_*pre*,同时调用类 *Check_Crosscut*1_*pre* 的 *set*1()函数,*set*1()函数先检查方

面 *Crosscut*1 的通知先验条件,不满足直接返回 false,满足则检查方面 *Crosscut*2 的通知先验条件是否满足,如果不满足则触发横切不变性错误,满足则返回 true.

```
public aspect Crosscut1{
  pointcut set1(A a, int x): execution(void A.setX(int))
  &&target(a)&&args(x);
  before(A a, int x):set1(a,x){
    if (x>5) {
      (new Check_Crosscut1_pre()).set1(a,x);
      a.setY(x);
      }else{
      adPreErr(Crosscut1);
    }
  }
}
public class Check_Crosscut1_pre{
  public boolean set1(A a,int x){
    if(x>5){
      if (new Check_Crosscut2_pre().set1(a,x))
        return true;
      else
        return crosscutInvariantErr(Crosscut1);
      }else{
        return false;
    }
  }
}
```

Fig. 5　Aspect Crosscut1 and behavioral checking class *Check_Crosscut1_pre*.

图 5　转换后的方面 Crosscut1 和行为检查类 *Check_Crosscut1_pre*

根据[*e*]规则改写客户程序 *M*,将对方法 setX 的调用转换为对包装方法 *setX_A* 的调用,代码如图 6 所示:

```
A a=new A(5,0);
a.setX_A(10);
```

Fig. 6　Client program *M* after transformation.

图 6　转换后的客户程序 *M*

运行程序,当程序运行到检查到行为检查类 *Check_Crosscut1_pre* 的横切不变性检查方法 *Check_Crosscut1_pre*,方面 *Crosscut*1 的通知先验条件满足,但是方面 *Crosscut*2 的通知先验条件不满足,即方面 *Crosscut*1 的通知先验条件不蕴含方面 *Crosscut*2 的通知先验条件,发生横切不变性错误.

5　相关研究

文献[19]定义一组扩展规则约束不同的方面组合,这些扩展规则主要针对 mixin 和 MixJuice 的语言的模块组合.本文使用行为子类型来定义这些扩展规则,如果组合后的类是组合前的类的行为子类型,则方面或者模块组合是安全的.我们认为如果要保持面向方面语言的基本特性,仅仅考虑横切的结果是不行,因为一个横切方面的行为错误可能被其他横切方面的行为所掩盖.

文献[20]使用程序切片技术检测横切方面之间的干扰,提出两个方面之间的非干扰条件:A_1 和 A_2 两个方面横切同一切入点,S_1 是 A_1 中使用切入点作为切片准则获得的向后切片,S_2 是 A_2 中使用切入点作为切片准则获得的向前切片方面,方面 A_1 不干扰方面 A_2 当且仅当 $S_1 \cap S_2 \neq \varnothing$.

文献[21]意识到文献[20]的条件过于严格,提出一个弱化的非干扰条件:如果 A_1 和 A_2 是两个方面,S_1 和 S_2 分别是 A_1 和 A_2 中使用 A_1 和 A_2 所有子句作为切片准则获得的向后切片,方面 A_1 不干扰方面 A_2 当且仅当 $A_1 \cap S_2 \neq \varnothing$. 文献[20-21]都通过检查方面之间的非干扰条件来确保横切安全和横切质量,但是避免方面干扰是不切实际的,有时程序员要利用方面之间的相互协作来达到特定的模块协作目的.文献[20-21]没有考虑方面和被横切模块之间可能的危险行为干扰,而横切不变性要求方面必须保持被横切模块的行为约束.

6　总　　结

横切安全和横切质量是保证面向方面软件系统的可靠性和可重用性的核心问题,本文使用软件行为契约,基于横切不变性来解决这个问题.与其他横切行为约束方法相比的优点在于:1) 横切不变性允许安全的行为干扰,禁止危险的行为干扰;2) 横切不变性不但考虑不同横切关注点之间的行为干扰,还考虑横切关注点和被横切模块之间的行为干扰.为了检查程序是否满足横切不变性,本文提出一个横切不变性检测算法,同时定义 Crosscutting Contract 演算和一组契约规则实现这种算法,并通过契约完备性理论证明其正确性.

由于横切安全和横切质量问题的复杂性,还有许多问题值得继续研究:本文讨论的横切不变性定义比较宽泛,可以借鉴关系数据库理论中的范式理论,对横切不变性的要求进行细化,针对不同的横切要求定义不同的范式,不同的横切不变性范式对应不同的横切不变性错误;算法中包含许多重复的行为契约运算,可以借鉴文献[20-21]的干扰性分析的方法,采用程序切片技术删除不必要的重复契约运算;本文讨论的横切不变性算法针对多个顺序的切

入点,对于更加复杂的横切问题,例如通知执行内部再次发生通知的情况可能不适应;面向方面语言本身在迅速的发展过程中,面向方面的系统也在不断出现,对于横切侵入性的度量和检测手段不能仅仅局限于软件行为契约,可以考虑其他的方法.

参 考 文 献

[1] Kiczales G, Lamping J, Mendhekar A, *et al*. Aspect-oriented programming [C] //Proc of ECOOP1997. Berlin: Springer, 1997: 220-242

[2] Filman R E, Friedman D P. Aspect-oriented programming is quantification and obliviousness, 00000046 [R]. Moffett Field, California: NASA Ames Research Center, 2000

[3] Aldrich J. Open modules: A proposal for modular reasoning in aspect-oriented programming, CMU-ISRI-04-108 [R]. Pittsburgh, Pennsylvania: Carnegie Mellon University, 2004

[4] Aldrich J. Open modules: Reconciling extensibility and information hiding [C] //Proc of AOSD2004. New York: ACM Press, 2004

[5] Aldrich J. Open modules: Modular reasoning about advice [C] //Proc of ECOOP2005. New York: Springer, 2005: 144-168

[6] Gybels K, Brichau J. Arranging language features for more robust pattern-based crosscuts [C] //Proc of AOSD2003. New York: ACM Press, 2003: 60-69

[7] Ostermann K, Mezini M, Bockisch C. Expressive pointcuts for increased modularity [C] //Proc of ECOOP2005. Berlin: Springer, 2005: 214-240

[8] Steimann F. The paradoxical success of aspect-oriented programming [C] //Proc of OOPSLA2006. New York: ACM Press, 2006: 481-497

[9] Liskov B H, Wing J W. A behavioral notion of subtyping [J]. ACM Trans on Programming Languages and Systems, 1994, 16(6): 1811-1841

[10] America P. Designing an object-oriented programming language with behavioural subtyping [C] //Proc of the REX School. Berlin: Springer, 1990: 60-90

[11] Liskov B H, Wing J W. Behavioral subtyping using invariants and constraints, CMU CS-99-156 [R]. Pittsburgh, Pennsylvania: Carnegie Mellon University, 1999

[12] Findler R B. Behavioral software contracts [D]. Houston, TX, USA: Rice University, 2002

[13] Meyer B. Applying "design by contract" [J]. Computer, 1992, 25(10): 40-51

[14] Eclipse Foundation. AspectJ Project [OL]. [2007-07-14]. http://www.eclipse.org/aspectj/

[15] Flatt M, Krishnamurthi S, Felleisen M. Classes and mixins [C] //Proc of the 25th ACM SIGPLAN-SIGACT Symp on Principles of Programming Languages. New York: ACM Press, 1998: 171-183

[16] Zhou Xiaocong. Equation theory of type system $\lambda\omega\times\leqslant$ and the soundness of its semantics [J]. Journal of Computer Research and Development, 2006, 43(5): 874-880 (in Chinese)
(周晓聪. 类型系统的 $\lambda\omega\times\leqslant$ 等式理论及其语义的合理性 [J]. 计算机研究与发展, 2006, 43(5): 874-880)

[17] Pierce B C. Types and Programming Languages [M]. Cambridge: MIT Press, 2002

[18] Hanenberg S, Costanza P. Connecting aspects in aspectJ: Strategies vs. patterns [C] //Proc of AOSD2002. New York: ACM Press, 2002

[19] Yuuji Ichisugi, Akira Tanaka, Takuo Watanabe. Extension rules: Description rules for safely composable aspects, AIST01-J00002-4 [R]. Osaka, Japan: Osaka University, 2003

[20] Blair L, Monga M. Reasoning on Aspect programmes [C] //Proc of AOSD2003. New York: ACM Press, 2003: 45-50

[21] Davide Balzarotti, Mattia Monga. Using program slicing to analyze aspect-oriented composition [C] //Proc of AOSD2004. New York: ACM Press, 2004

Lü Jia, born in 1977. Ph. D. candidate. His current research interests include aspect-oriented software development and software automation.

吕 嘉, 1977 年生, 博士研究生, 主要研究方向为面向方面软件开发和软件自动化.

Ying Jing, born in 1971. Ph. D., professor, and Ph. D. supervisor. His main research interests include software engineering and software automation.

应 晶, 1971 年生, 博士, 教授, 博士生导师, 主要研究方向为软件工程和软件自动化.

Wu Minghui, born in 1975. Ph. D. candidate and associate professor. His main research interests include software engineering, artificial intelligence and internet application.

吴明晖, 1975 年生, 博士研究生, 副教授, 主要研究方向为软件工程、人工智能和网络应用.

Jiang Tao, born in 1981. Ph. D. candidate. His current research interests include software engineering and software product-line engineering.

蒋 涛, 1981 年生, 博士研究生, 主要研究方向为软件工程和软件产品线工程.

Research Background

Aspect-oriented programming (AOP) succeeds in localizing, separating, and encapsulating crosscutting concerns. But some characteristics of aspect-oriented language violate modularization and encapsulation principle, and create dependencies between different concerns. Modular behavioral analysis and modular reasoning in aspect-oriented language are more difficult than that of traditional paradigms. In this paper, a notion of crosscutting invariant is proposed to deal with crosscutting safety and crosscutting quality, and a detecting algorithm is proposed based on this notion. To formalize this algorithm, we present crosscutting contract calculus and a set of contract elaboration rules. Our work is supported by the Fork Ying Yung Science Foundation for Yong Teachers under grant No. 94030.

附录 A

横切契约语法(crosscutting contract syntax)

$P ::= defn^*\ e$

$defn ::=$ class c extends c implements i^*
$\{\ field^*\ meth^*\ \}$
| interface i extends $i^*\ \{\ imeth^*\ \}$
| aspect $a\ \{\ field^*\ meth^*$
$crosscut\_static^*\ crosscut\_dynamic^*\ \}$

$crosscut\_static ::= out\_field^*\ out\_meth^*$

$crosscut\_dynamic ::= pt^*\ ad^*$

$field ::= t\ fd$

$out\_field ::= t\ c.\ fd$

$meth ::= t\ md\ (arg^*)\ \{\ body\}$
$@pre\ pn\ \{\ e\}\ @post\ pn\ \{e\}$

$imeth ::= t\ md\ (\ arg^*)$

$out\_meth ::= t\ c.md(\ arg^*)\{\ body\}$

$pt ::= pointcut\ pn : jt^*$

$ad ::=$ (before|after)$(arg^*) : pn\{\ body\}$

$@pre\ pn\ \{\ e\}\ @post\ pn\ \{\ e\}$

$jt ::= t.md(arg^*)$

$arg ::= t$ var

$body ::= e$ | $abstract$

$e ::= new\ c$ | var | null
| $e.\ fd$ | $e.\ fd = e$
| $e.md\ (e^*)$
| $super.md\ (e^*)$
| if (e) e else e | true | false
| $\{\ e\ ;\ e\}$
| return e

$binding ::= var = e$

$t ::= c$ | i | boolean

$var ::=$ a variable name or this

$c ::=$ a class name or Object

$i ::=$ interface name or Empty

$fd ::=$ a field name

$md ::=$ a method name

$a ::=$ a aspect name

$pn ::=$ a pointcut name

附录 B

横切契约演算契约求解规则(contract elaboration rules of crosscutting contract calculus)

$$\frac{^{prog}\ P = defn_1 \cdots defn_n e\ P \vdash e \rightarrow_e e' \qquad P \vdash defn_j \rightarrow_d defn'_j\ j \in [1,n]}{P \vdash defn_1 \cdots defn_n e \rightarrow_P defn'_1 \cdots defn'_n e'}$$

$$\frac{^{defn^c}\ P, c \vdash meth_j \rightarrow_m meth'_j \qquad P, c \vdash meth_j \rightarrow_w wrap\_meth_j\ j \in [1,n]}{P \vdash class\ c \cdots meth_1 \cdots meth_n \rightarrow_d class\ c \cdots meth'_1 \cdots meth'_n\ wrap\_meth_1 \cdots wrap\_meth_n}$$

$$\frac{^{defn^a}\ P, a \vdash ad_j \rightarrow_m ad'_j \qquad P, a \vdash ad_j \rightarrow_{pre} ad_{j_{pre}}\ P, a \vdash ad_j \rightarrow_{post} ad_{j_{post}} \qquad P, a \vdash ad_j \rightarrow_w wrap\_ad_j\ j \in [1,n]}{\begin{array}{l} Pa \vdash spect\ a\ ad_1 \cdots ad_n \\ \rightarrow_d aspect\ a\ ad'_1 \cdots ad'_n wrap\_ad_1 \cdots wrap\_ad_n \\ \quad class\ Check\_a\_pre\ ad_{1_{pre}} \cdots ad_{n_{pre}} \\ \quad class\ Check\_a\_post\ ad_{1_{post}} \cdots ad_{n_{post}} \end{array}}$$

$$\frac{^{meth^c}\ P, c \vdash e \rightarrow_e e'}{\begin{array}{l} P, c \vdash t\ md(t_1\ argn_1 \cdots t_n\ argn_n)\{e\}\ @\ pre\{e_b\}@post\{e_a\} \\ \quad \rightarrow_m t\ md(t_1\ argn_1 \cdots t_n\ argn_n)\{e'\} \end{array}}$$

$$\frac{^{meth^a}\ P, a \vdash e \rightarrow_e e'}{\begin{array}{l} P, a, ad \vdash pn(t_1\ argn_1 \cdots t_n\ argn_n)\{e\}\ @\ pre\{e_b\}@post\{e_a\} \\ \quad \rightarrow_m pn(t_1\ argn_1 \cdots t_n\ argn_n)\{e'\} \end{array}}$$

$$\frac{\begin{array}{l} wrap^c\ e_b\ Pre_p\langle c, md\rangle e_a\ Post_p\langle c, md\rangle \\ P, c \vdash e_b \rightarrow_e e'_b\ P, c \vdash e_a \rightarrow_e e'_a\ j \in [1,n] \end{array}}{\begin{array}{l} P, c \vdash c'\ md(t_1\ x_1 \cdots t_j\ x_j) \rightarrow_w c'\ md\_t(t_1 x_1 \cdots t_j x_j)\{ \\ \quad \text{if}(e'_b)\{ \\ \quad\quad @ret = this.md(x_1 \cdots x_j); \\ \quad\quad \text{if}(!e'_a) \\ \quad\quad\quad mdPostErr(c); \\ \quad\quad \} \\ \quad\quad \text{else}\{ \\ \quad\quad\quad mdPreErr(c); \\ \quad\quad \} \\ \quad\quad \text{return}@ret; \\ \quad \} \end{array}}$$

$$\frac{wrap^{a}\ P,a,ad \vdash\!\!\rightarrow e \rightarrow_{e} e'\ e_b\ Pre_p\langle a,ad\rangle e_a\ Post_p\langle a,ad\rangle \quad P,a,ad \vdash\!\!\rightarrow e_b \rightarrow_{e} e'_b\ P,a,ad \vdash\!\!\rightarrow e_a \rightarrow_{e} e'_a\ j \in [1,n]}{P,a,ad \vdash\!\!\rightarrow pn(t_1\ x_1 \cdots t_n\ x_n) \rightarrow_{w} pn(t_1\ x_1 \cdots t_n\ x_n)\{}$$

```
    if(e'_b){
      (new Check_a_pre()).pn(x_1 ··· x_j);
      e';
      if(e'_a)
        (new Check_a_post()).pn(x_1 ··· x_j);
      else
        adPostErr(c);
    }
    else{
      adPreErr(c);
    }
}
```

$$\frac{pre^{a}\ e_{b_j}\ Pre_p\langle a_j,ad_j\rangle\ a_j \leqslant a_{j+1}\ j=[1,n]}{P,a_j,ad_j \vdash\!\!\rightarrow pn_j(t_1\ argn_1 \cdots t_n\ argn_n)\{e\}\ @pre\{eb_j\}\ @post\{e_{a_j}\} \rightarrow_{pre} boolean\ pn_j(t_1\ argn_1 \cdots t_n\ argn_n)\{}$$

```
  if(e'_{b_j}{
    if (new Check_a_{j+1}_pre(). pn_{j+1}(argn_1 ··· argn_n))
      return true;
    else
      return crosscutInvariantErr(a_j);
    }else{
      return false;
    }
  }
```

$$post^{a}\ ad_j\ Before\langle a_j,c,md\rangle\ e_b\ Pre_p\langle c,md\rangle \quad P,c \vdash\!\!\rightarrow e_b \rightarrow_{e} e'_b \quad e_{a_j}\ Post_p\langle a_j,ad_j\rangle P,\ a_j,ad_j \vdash\!\!\rightarrow e_{a_j} \rightarrow_{e} e'_{a_j} \quad a_j \leqslant a_{j+1}\ j=[1,n]$$

$$P,a_j,\ ad_j \vdash\!\!\rightarrow pn_j(t_1\ argn_1 \cdots t_n\ argn_n)\{e\}\ @pre\{e_{b_j}\ @post\{e_{a_j}\} \rightarrow_{post} boolean\ pn_j(t_1\ argn_1 \cdots t_n\ argn_n)\{}$$

```
  if(e'_{a_j}){
    if(e'_b)
      return true;
    else
      return crosscutInvariantErr(a_j);
  }else{
    return false;
  }
}
```

$$post^{a}\ ad_j\ After\langle a_j,c,md\rangle\ e_a\ Pre_p\langle c,md\rangle \quad P,c \vdash\!\!\rightarrow e_a \rightarrow_{e} e'_a \quad e_{a_j}\ Post_p\langle a_j,ad_j\rangle P,\ a_j,ad_j \vdash\!\!\rightarrow e_{a_j} \rightarrow_{e} e'_{a_j} \quad a_j \leqslant a_{j+1}\ j=[1,n]$$

$$P,a_j,\ ad_j \vdash\!\!\rightarrow pn_j(t_1\ argn_1 \cdots t_n\ argn_n)\{e\}\ @pre\{e_{b_j}\ @post\{e_{a_j}\} \rightarrow_{post} boolean\ pn_j(t_1\ argn_1 \cdots t_n\ argn_n)\{}$$

```
  if(e'_{a_j}){
    if(e'_a)
      return true;
    else
      return crosscutInvariantErr(a_j);
  }else{
    return false;
  }
}
```

$$\frac{call^{t}\ P,c \vdash\!\!\rightarrow e \rightarrow_{e} e' P,c \vdash\!\!\rightarrow e_j:t_j \rightarrow_{e} e'_j\ j \in [1,n]}{e:t.md(e_1 \cdots e_n) \rightarrow_{e} e'.wrap\_md(e_1 \cdots e_n)}$$

$$\frac{P,a \vdash\!\!\rightarrow e \rightarrow_{e} e'\ P,a \vdash\!\!\rightarrow e_j:t_j \rightarrow_{e} e'_j\ j \in [1,n]}{e:t.md(e_1 \cdots e_n) \rightarrow_{e} e'.wrap\_md(e_1 \cdots e_n)}$$