面向方面程序的静态语义研究

谢 刚[1] 韦 立[1] 吴 祥[2]
(贵州师范大学大数据与计算机科学学院 贵阳 550001)[1] (贵州师范大学数学科学学院 贵阳 550001)[2]

摘 要 针对面向方面程序,许多研究者已定义了各种各样的形式语义。但是这些语义都不能够全面、准确地对面向方面程序的规范和方面声明部分进行描述。针对该问题,首先定义一种统一的面向方面程序的规范语言;其次对面向方面程序中的连接点和切点这两个重要概念进行形式化定义;再次引入结构变量表示面向方面程序的基本结构;最后应用统一程序理论中的设计定义面向方面的静态语义,并对其可靠性进行证明。同时,用一个例子说明该语义的使用。

关键词 面向方面程序,静态,语义
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.09.035

Static Semantics of Aspect-oriented Programming

XIE Gang[1] WEI Li[1] WU Xiang[2]
(School of Big Data and Computer Science, Guizhou Normal University, Guiyang 550001, China)[1]
(School of Mathematics Science, Guizhou Normal University, Guiyang 550001, China)[2]

Abstract Till now, many researchers have developed various formal semantics for aspect-oriented program. However, none of the semantics have provided the characterization of aspect-oriented programming specification and the declaration section of an aspect comprehensively and precisely. To make a further step, we defined a unified aspect-oriented programming specification language in our research. Then, we provided a formal definition for joinpoint and pointcut for aspect-oriented programs. Next, we introduced structural variables into the static structure to represent the aspect-oriented programs. Finally, we defined static semantics of aspect-oriented programs using the definition of design in unifying theories of programming, and proved its soundness afterwards. The approach was enumerated with a case to demonstrate the usage of the semantics.

Keywords Aspect-oriented programming, Static, Semantics

1 引言

Kiczlaes 等人[1]提出了面向方面程序设计(AOP)的思想。他们认为,AOP 提供一种对横切关注点(如安全、同步等模块)模块化的方法,是 OOP 的一个扩充。这使得软件工程、程序设计语言等领域的大量研究者对 AOP 的研究产生了极大的兴趣。

近年来,许多研究者已开发出各种各样的 AOP 语言,如 AspectJ[2],AspectC++[3],AspectR[4],AspectWerkz[5],AspectS[6],JBoss-AOP[7]和 HyperJ[8]等。在这些语言中,AspectJ 是最广泛使用的语言,显然对其语义进行研究是有意义的。

何积丰等人[9]指出软件开发与形式化方法是两个重要且独立的领域,只有将软件开发与形式化方法统一起来,才能提高软件的生产率和可靠性。因此,需要采用一种软件开发人员能理解的语义来描述软件需求和行为,以进行形式化验证。

经研究发现,对程序静态结构的改变可导致程序行为的改变,因此对程序静态结构的形式化,有助于理解程序的行为,保证需求规范的正确性[9]。本文将程序静态结构的形式化表示称为静态语义。在已有工作中[10-18,20,23-24,29],只有文献[20]关注面向方面程序的静态语义。本文将对文献[20]提出的语义进行扩充,使其能描述 AOP 中的主要结构。具体来说,首先增加了连接点、切点和通知的形式化表示;其次定义了切点、通知、方面及方面程序的静态语义,并证明了语义的可靠性。

本文第 2 节对相关知识进行介绍;第 3 节对 AOP 的相关概念进行介绍;第 4 节为 AspectJ 定义一种静态语义;第 5 节通过一个示例说明本文所提语义的正确性;最后总结全文,并对未来的研究进行初步探讨。

2 预备知识

2.1 逻辑形式推演规则[25]

本文用到的谓词逻辑形式推演规则如下。

规则 1 $A \wedge \exists x B(x) \Leftrightarrow \exists x(A \wedge B(x))$。

到稿日期:2016-08-05 返修日期:2016-11-11 本文受国家自然科学基金(61309006,61563011),贵州师范大学资助博士科研项目资助。
谢 刚(1980—),男,博士,副教授,主要研究方向为计算机软件与理论,E-mail:xiegang@gznu.edu.cn;韦 立(1981—),男,博士,副教授,主要研究方向为计算机软件与理论;吴 祥(1980—),男,博士,副教授,主要研究方向为最优控制。

规则 2　$\exists x(A(x)\wedge B(x))\Leftrightarrow\exists xA(x)\wedge\exists xB(x)$。

规则 3　$A(t)\Leftrightarrow\exists x[x=t\wedge A(x)]$,其中 $A(x)$ 是在 $A(t)$ 中将 t 的某些出现替换成 x 而得。

2.2　谓词的顺序复合[22]

假设 P 和 Q 是两个谓词,并且 $in\alpha'Q=out\alpha P=\{v'\}$,即 P 的输出变量是 Q 的输入变量,则:

$$P(v');Q(v)=_{df}\exists v_0\cdot(P(v_0)\wedge Q(v_0)) \quad (1)$$

推论 1[22]　P;false=false。

2.3　重要符号定义[9]

定义 1　对任意的 $f:\{b\vdash\!\!\!\rightarrow X\mid b\in D,x\in E\}$,$d\in D$,$r\in E$,

$$f\oplus\{d\vdash\!\!\!\rightarrow r\}=_{df}\begin{cases}f\cup\{d\vdash\!\!\!\rightarrow r\}, & \text{if } f(d)=\Phi\\(f-\{d\vdash\!\!\!\rightarrow y\})\cup\{d\vdash\!\!\!\rightarrow r\}, & \text{if } f(d)=y\end{cases}$$

例 1　假设 $D=\{M,N,X,A,B,C,E\}$,$f=\{M\vdash\!\!\!\rightarrow N,N\vdash\!\!\!\rightarrow X,A\vdash\!\!\!\rightarrow B\}$,则

$(f\oplus\{G\vdash\!\!\!\rightarrow H\})\oplus\{M\vdash\!\!\!\rightarrow Y\}=\{G\vdash\!\!\!\rightarrow H,M\vdash\!\!\!\rightarrow Y,N\vdash\!\!\!\rightarrow X,A\vdash\!\!\!\rightarrow B\}$

3　AOP 的相关概念

3.1　基本概念

定义 2(核心关注点)　指系统要实现的主要功能和目标,如学生成绩查询等[22]。

定义 3(横切关注点)　指各个核心关注点间共享的功能,例如安全。对于已实现的系统,横切关注点还包含对系统的更新需求[22]。

定义 4(连接点)　指程序的执行过程中合适定义(well-defined)的点。例如对构造方法的调用和对方法的执行[1]。

定义 5(切点)　指一系列连接点的集合,指定横切关注点需要插入的位置[1]。

定义 6(通知)　指在切点所选定的连接点处要执行的代码[1]。

定义 7(方面)　指对横切关注点进行封装的基本模块[1]。

3.2　面向方面程序示例

图 1 示出 AspectJ 的程序代码,该程序由类 Main,Point 和方面 A 与 B 组成。其中,类 Point 是基本代码,包含成员变量 x,成员方法 setX,getX 和 add;类 Main 为主程序,是程序的入口,只包含一个类静态方法 main();方面 A 是横切代码,包含切点 p 和一个 After 通知,通知的作用是显示 Point 类的成员变量 x 的值;方面 B 也是横切代码,包含切点 $p2$ 和一个 After 通知,该通知的作用是检查 Point 类的成员变量 x 的值,若值大于 100,则将 x 的值重新设置为 0,否则不变。

该程序的执行过程如下:首先由主方法 main 创建 Point 的实例 c,实例 c 调用成员方法 setX 将成员变量 x 赋值为 100,接着调用成员方法 add,由于对 add 的调用与切点 p 和 $p2$ 所捕获的连接点匹配,而与切点 p 和 $p2$ 绑定的都是 After 通知,因此先调用 add,即 x 的值变为 105,然后再根据通知的执行规则先执行方面 B 的通知,即将 x 的值置为 0,最后再执行方面 A 的通知,在屏幕上显示“x is changed to 0”。

```
class Point{
    int x;
    public int getX(){
        return x;
    }
    public void setX(int x){
        this. x=x;
    }
    public void add(int y){
        this. x=this. x+y;
    }
}
class Main{
    public static void main(String[] arg s)
    {
        Point c=new Point();
        c. setX(100);
        c. add(5);
      }
}
aspect A{
    pointcut p(Point d):call(void Point. add (int))&&target(d)
    after(Point e) returning:p(e){
    System. out. println("x is changed to"+e. getX());
    }
}
aspect B{
    pointcut p2(Point c):call(void Point. add (int))&&target(c)
    after(Point d) returning p2(d){
    if(d. getX()>100) d. setX(0);
    }
}
```

图 1　AspectJ 示例代码

4　语义

4.1　一种 AOP 语言

目前,不同的 AO 程序语义使用不同的 AOP 语言,这使得软件设计人员和开发人员难以理解。为解决上述问题,本文采用类似 AspectJ 的抽象 AOP 语言,该语言的 BNF 范式如表 1 所列。

表 1 列出 AO 程序的语法,相比主流 AOP 语言 AspectJ,该语言只包含 AOP 中最基本的元素。

该语言中的一个程序 prog 包括类声明序列 cdecls、方面声明序列 adecls 和主方法 Main。其中,adecls 可以为空序列。

该语言中的方面声明 adecl 包含方面名 C、父方面或父类名 D、属性声明序列 adefs、方法声明序列 mdefs、切点声明序列 pcdefs、通知声明序列 addefs。其中,父方面或父类名 D 是可选的,只用于表示方面的继承,本语言不支持多重继承。

该语言中的一个切点声明 pcdef 由切点名 pc、形式参数表(prameters)和基本切点描述符 pcd 构成。其中,pcd 对应于 AspectJ 的 17 种基本切点描述符。

该语言中的一个通知声明 addef 由通知类型、通知名 ad、形式参数表(prameters)、切点 pc 和命令 c 构成。其中,通知类型包括 before,after 和 around,其对应于 AspectJ 的 3 类通知,本文为通知增加通知名,以便更好地对通知进行重用和维护。

表 1 AOP 语言的 BNF

程序	prog::=cdecls;adecls;Main
方面声明序列	adecls::=ε|adecl|adecl;adecls
单个方面声明	adecl::=aspect A[extends D]{adefs;mdefs;pcdefs;addefs}
切点声明序列	pcdefs::=pcdef|pcdef;pcdefs
单个切点声明	pcdef::=pointcut pc(prameters):pcd
原始切点描述符	pcd::=call(MethodPattern)|execution(MethodPattern) |get(FieldPattern)|set(FiledPattern) |call(ConstructorPattern)|execution(ConstructorPattern) |initialization(ConstructorPattern) |preinitialization(ConsstructorPattern) |withincode(MethodPattern) |withincode(ConstructorPattern) |this(T)|this(x)|target(T)|target(x) |pcd&&pcd|pcd‖pcd|!pcd
匹配模式	TP::=*|C+|C TyPattern::=TP|T|TyPattern‖TyPattern|TyPattern&&TyPattern|!TyPattern MethodPattern::=TyPattern TP.m(TS) ConstructorPattern::=TyPattern TP.new(TS) FieldPattern::=TyPattern TP.x|TyPattern x
通知声明序列	addefs::=addef|addef;addefs
单个通知声明	addef::=advicetype ad(prameters):pc(y){c} advicetype::=before|after|around
类声明序列	cdecls::=cdecl|cdecl;cdecls
单个类声明	cdecl::=visibity class C[extendsD]{adefs;mdefs}
属性声明序列	adefs::=adef|adef;adefs
单个属性声明	adef::=visib:Ta[=l]
类的可见性	visibility::=private|public
属性的可见性	visib::=private|public|protected
类型说明符序列	TS::=T|T;TS
单个类型说明符	T::=int|float|string|char|Boolean|C
方法声明序列	mdefs::=mdef|mdef;mdefs
单个方法声明	mdef::=m(prameters){c}
主方法	Main::=(exts;c)
外部变量声明序列	exts::=ext|ext;exts
单个外部变量声明	ext::=T x[=l]
形式参数序列	prameters::=ext|ext; prameters
命令	c::=skip|chaos|le:=C.new(eS)|var T x[=e]|end x|le.m(eS)|c;c| c◁b▷c|c⊓c|b*c|les:=eS
表达式序列	eS=e|e;eS
单个表达式	e::=x|le|C(e)|l|f(eS) leS=le|le;leS le::=o|le.a|self l::=cn|null

该语言共支持 5 种匹配模式,分别与 AspectJ 的 5 种匹配模式对应,它们是类名模式(TP)、类型模式(TyPattern)、方法匹配模式(MethodPattern)、构造方法匹配模式(ConstructorPattern)和域模式(FieldPattern)。

5 种匹配模式的描述如下。

类名模式(TP):* 表示任意的类名,C+表示类 C 的所有子类。

类型模式(TyPattern):包括基本模式和复合类型复式。其中,基本模式包括基本类型(T)和类名模式,复合类型模式利用逻辑运算符 &&(与)、‖(或)和!(非)将基本模式连接而成。例如((Foo+&&!Foo)表示 Foo 的所有子类,但不包括 Foo 本身。

方法匹配模式(MethodPattern):由类型模式、类名模式、方法名、参数类型序列组成。

构造方法匹配模式(ConstructorPattern):由类型模式、类名模式、new 和参数类型序列组成。

域模式(FieldPattern):由类型模式、类名模式和域名组成。

4.2 结构变量

为了描述程序的静态结构,引入下列结构变量。

1)$ASNAME$:表示所声明的方面名字构成的集合。

2)$Advice$:表示所有通知构成的集合。

3)$Pointcut$:表示所有切点构成的集合。

4)如果 $ANAME=_{df}\{M\mapsto\{\langle a_1:T_1,d_1\rangle,\cdots,\langle a_n:T_n,d_n\rangle\mid(T_ia_i=d_i)(i=1,\cdots,n)$在方面声明中作为 M 的属性$\}\}$,则 $ANAME(M)=\{\langle a_1:T_1,d_1\rangle,\cdots,\langle a_n:T_n,d_n\rangle\}$。

5)如果 $superclass=_{df}\{M\mapsto N\mid(aspect\ M\ extends\ N)$出现在方面声明中且 $N\in CNAME\}$,则 $superclass(M)=\{N\}$。

6)如果 $superaspect=_{df}\{A\mapsto B\mid(aspect\ A\ extends\ B)$出现在方面声明中且 $B\in ASNAME\}$,则 $superaspect(A)=\{B\}$。

上述结构变量和文献[9]中定义的结构变量构成的集合称为 AOP 的结构变量集,记为 Ω,即 $\Omega=\{ASNAME,Advice,Pointcut,ANAME,superclass,superaspect,pricname^{[9]},pri^{[9]},prot^{[9]},pub^{[9]},op^{[9]},pubcname^{[9]}\}$。

4.3 AOP 中重要概念的形式化定义

定义 8 一个连接点是一个五元组$\langle jpt,o,id,vs,returntype\rangle$,其中:

1)jpt 表示连接点类型,即 $jpt\in JPT$,$JPT=_{df}\{pcall,pexecution,set,fget,finit,fpreinit,aexecution,pwithin,pwithincode\}$。

2)o 表示与方法或属性绑定的对象名,$o\in INDENTIFIER$,其中,$INDENTIFIER$ 表示标识符构成的集合,下同。

3)id 表示方法名或属性名,$id\in INDENTIFIER$。

4)vs 表示方法的实际参数,即 $vs\in VAL^*\cup\varepsilon$,其中,ε 表示空序列,$VAL$ 表示变量的集合[9],下同。

5)$returntype$:表示方法的返回值类型,即 $returntype\in T\cup CNAME\cup\varepsilon$,其中 $CNAME$ 表示声明的类的集合[9],T 表示变量类型的集合[9],下同。

定义 9 一个切点 p 是一个五元组$\langle A,pc,PVT,PV,pcd\rangle$,其中:

1)A 表示切点所在的方面名,$A\in ASNAME$。

2)pc 表示程序中声明的切点名,$pc\in IDENTIFIER$。

3)PVT 表示切点的形式参数类型,即 $PVT\in(T\cup CNAME)^*\cup\varepsilon$。

4)PV 表示切点的形式参数名字,$PV\in IDENTIFIER$。

5)pcd 表示切点描述符,即 $pcd\in PCD$,其中 PCD 表示所有基本切点描述符构成的集合。

例 2 $\langle pc,\varepsilon,\varepsilon,call(C.m())\rangle$表示切点(*pointcut pc*():*call*(*C.m*()))。

$\langle pc,C,b,target(b)\rangle$表示切点(*pointcut pc*(*C b*):*target*(*b*))。

定义 10　一个通知 adv 是一个七元组$\langle A, ad, advicetype, ADVT, ADV, pc, c\rangle$,其中:

1)A 表示切点所在的方面名,$A\in ASNAME$。

2)ad 表示通知的名字,即 $ad\in IDENTIFIER$。

3)$advicetype$ 表示通知的类型,即 $advicetype\in\{before, after, around\}$。

4)$ADVT$ 表示通知的形式参数类型,即 $ADVT\in(T\cup CNAME)^*\cup\varepsilon$。

5)ADV 表示通知的形式参数名,即 $ADV\in VAR^*\cup\varepsilon$,$VAR$ 表示变量的集合[9],下同。

6)pc 表示程序中声明的切点名,即 $pc\in IDENTIFIER$。

7)c 表示通知体,与方法体类似,由一系列的命令构成[9]。

4.4　设计

本文将静态语义定义为统一程序理论(Unifying Theories of Programming,UTP)[22]中的设计。设计表示用户和程序之间的合同,该合同表示如果程序能在满足前置条件 p 的状态下开始,那么在满足后置条件 R 的状态下一定终止。

为了方便,我们采用 rCOS 定义的设计,该定义把设计定义成如下形式[9]:

$$\beta:p\vdash R=_{df}p\vdash(R\wedge\underline{w}'=\underline{w})\qquad(2)$$

其中,$\underline{w}$表示设计过程中不会发生变化的变量集合,$\underline{w}\subseteq in\alpha$;$\beta$ 表示设计过程中会发生变化的变量集合,$\beta\subseteq in\alpha$;$\underline{w}\cup\beta=in\alpha$;$\underline{w}'\subseteq out\alpha$。

本文中,$in\alpha=\Omega$,$out\alpha=\Omega'=\{ASNAME', Advice', Pointcut', ANAME', superclass', superaspect', pricname', pri', prot', pub', op', pubname\}$。

定理 1[9]　$((p_1\vdash R_2);(p_2\vdash R_2))=((p_1\wedge\neg(R_1;\neg p_2))\vdash(R_1;R_2))$。

4.5　静态语义

4.5.1　切点声明部分的语义

本节为表 1 中的单个切点声明 pcdef 和切点声明序列 pcdefs 定义语义。

定义 11　假设一个切点声明为 pcdef,其语义定义如下:

$$[[pcdef]]=_{df}\{Pointcut\}:\mathbb{D}(pcdef)\vdash ModifyPointcut$$

其中:

1)$\mathbb{D}(pcdef)$[20]表示 pcdef 是否被良好定义(well-defined),其值为 false 或 ture,下同。

2)$ModifyPointcut=_{df}\ Pointcut'=Pointcut\cup\{\langle A, pc, PVT, PV, pcd\rangle\}$。

该语义的直观意义是:每声明一个切点,就会使切点集合变量 $Pointcut$ 在原来的基础上增加一个新的元素,其余结构变量不变。

定义 12　假设 $pcdefs=(pcdef_1;\ pcdef_2;\cdots;\ pcdef_n)$,其语义定义如下:

$$[[pcdefs]]=[[pcdef_1]];[[pcdef_2]];\cdots;[[pcdef_n]]$$

该定义的直观意义是:切点声明部分的语义为每个切点声明语义的复合,从而揭示增量开发的本质,以支持模块化推理。

引理 1(切点声明部分的语义可靠性)　任意一个切点声明部分,其语义都可表示为一个设计。

证明:对切点声明部分的结构进行归纳,证明过程如下:

1)当 $pcdefs=(pcdef_1)$,根据定义 11 显然结论成立。

2)假设 $pcdefs=(pcdef_1;\ pcdef_2;\cdots;\ pcdef_m)$,则有 $[[pcdefs]]=\{Pointcut\}:\mathbb{D}(pcdefs)\vdash R(Pointcut')$成立,其中 $R(Pointcut')=(Pointcut'=\{\langle A, pc_i, PVT_i, PV_i, pcd_i\rangle\mid i=1,2,\cdots,m\})$。

3)当 $pcdefs=(pcdef_1;pcdef_2;\cdots;pcdef_m;pcdef_{m+1})$ 时,令 $R'(Pointcut')=R(Pointcut')\wedge\underline{w}'=\underline{w}$,其中$\underline{w}'=\Omega'-\{Pointcut'\}$,$\underline{w}=\Omega-\{Pointcut\}$,$R1(Pointcut)=(Pointcut'=Pointcut\cup\{\langle A, pc_{m+1}, PVT_{m+1}, PV_{m+1}, pcd_{m+1}\rangle\})$,$R1'(Pointcut)=R1(Pointcut)\wedge\underline{w}'=\underline{w}$,$p=(\mathbb{D}(pcdefs)\wedge\neg(R'(Pointcut');\neg\mathbb{D}(pcdef_{m+1}))$,则

$[[pcdefs]]$

$=[[pcdef_1]];[[pcdef_2]];\cdots;[[pcdef_m]];[[pcdef_{m+1}]]$　(定义 12)

$=\{Pointcut\}:\mathbb{D}(pcdefs)\vdash R(Pointcut');[[pcdef_{m+1}]]$　(归纳假设)

$=\mathbb{D}((pcdefs)\vdash R'(Pointcut');[[pcdef_{m+1}]]$　(式(2))

$=\mathbb{D}(pcdefs)\vdash R'(Pointcut');\{Pointcut\}:\mathbb{D}(pcdef_{m+1})\vdash R1(Pointcut)$　(定义 11)

$=\mathbb{D}(pcdefs)\vdash R'(Pointcut');\mathbb{D}(pcdef_{m+1})\vdash R1'(Pointcut)$　(式(2))

$=p\vdash(R'(Pointcut');R1'(Pointcut))$　(定理 1)

又$\because R'(Pointcut');R1'(Pointcut)$

$=\exists v(R'(v)\wedge R1'(v))$　(式(1))

$=\exists v((v=\{\langle A, pc_i, PVT_i, PV_i, pcd_i\rangle\mid i=1,2,\cdots,m\}\wedge\underline{w}'=\underline{w})\wedge(Pointcut'=v\cup\{\langle A, pc_{m+1}, PVT_{m+1}, PV_{m+1}, pcd_{m+1}\rangle\}\wedge\underline{w}'=\underline{w}))$

(将 R 与 $R1$ 中的 $Pointcut$ 替换为 v)

$=\exists v(v=\{\langle A, pc_i, PVT_i, PV_i, pcd_i\rangle\mid i=1,2,\cdots,m\}\wedge Pointcut'=v\cup\{\langle A, pc_{m+1}, PVT_{m+1}, PV_{m+1}, pcd_{m+1}\rangle\}\wedge\underline{w}'=\underline{w})$

(根据"∧"的结合律、幂等律和交换律)

$=\exists v(v=\{\langle A, pc_i, PVT_i, PV_i, pcd_i\rangle\mid i=1,2,\cdots,m\}\wedge Poin\ tcut'=v\cup\{\langle A, pc_{m+1}, PVT_{m+1}, PV_{m+1}, pcd_{m+1}\rangle\})\wedge\underline{w}'=\underline{w}$　(规则 1)

$=Pointcut'=(\{\langle A, pc_i, PVT_i, PV_i, pcd_i\rangle\mid i=1,2,\cdots,m\}\cup\{\langle A, pc_{m+1}, PVT_{m+1}, PV_{m+1}, pcd_{m+1}\rangle\})\wedge\underline{w}'=\underline{w}$　(规则 3)

$=Pointcut'=\{\langle A, pc_i, PVT_i, PV_i, pcd_i\rangle\mid i=1,2,\cdots,m+1\}\wedge\underline{w}'=\underline{w}$

$\therefore[[pcdefs]]$

$=p\vdash Pointcut'=\{\langle A, pc_i, PVT_i, PV_i, pcd_i\rangle\mid i=1,2,\cdots,m+1\}\wedge\underline{w}'=\underline{w}$

$=\{Pointcut\}:p\vdash Pointcut'=\{\langle A, pc_i, PVT_i, PV_i, pcd_i\rangle\mid i=1,2,\cdots,m+1\}$　(式(2))

$\therefore$当 $pcdefs=(pcdef_1;pcdef_2;\cdots;pcdef_m;pcdef_{m+1})$ 时,结论成立。

根据 1)－3)可得,任意一个切点声明部分,其语义都可表示为一个设计。

4.5.2 通知声明部分的语义

本节为表 1 中的单个通知声明 addef 和通知声明序列 addefs 定义语义。

定义 13 假设在方面 A 中与切点 p 绑定的一个通知声明为 addefs，其语义如下：

$[[addef]]=_{df}\{Advice\};\mathbb{D}(addef)\vdash Advice'=Advice\cup\{\langle A,adv,advicetype,ADVT,ADV,pc,c\rangle\}$

该定义的直观意义是：每定义一个通知，通知集合 *Advice* 就会增加一个元素。

定义 14 假设 $addefs=(addef_1;addef_2;\cdots;addef_n)$，且在 $addefs$ 中定义 n 个切点 $p_1,\cdots,p_n$，其语义如下：

$[[addefs]]=[[addef_1]];[[addef_2]];\cdots;[[addef_n]]$

引理 2(通知声明部分的语义可靠性) 任意一个通知声明部分，其语义都可表示为一个设计。

此引理的证明与引理 1 类似，此处省略。

4.5.3 方面声明部分的语义

本节为表 1 中的单个方面声明 adecl 和方面声明序列 adecls 定义语义。

定义 15 假设在 adecl 中，定义一个方面 A，其父类为 N，A 中有 m 个属性、n 个方法、l 个切点、k 个通知，则 adecl 的语义定义如下：

$[[adecl]]=_{df}[[aspect\ A\ \text{extends}\ N]];[[adefs]];[[mdefs]];[[pcdefs]];[[addefs]]$

其中：

$[[aspect\ A\ extends\ N]]=_{df}\{ASNAME,supercalss\};\mathbb{D}(A)\wedge\mathbb{D}(N)\vdash ASNAME'=ASNAME\cup\{A\}\wedge supercalss'=superclass\oplus\{A\mapsto N\}$

$[[adefs]]=_{df}\{ANAME\};\mathbb{D}(adefs)\vdash ANAME'=ANAME\oplus\{A\mapsto\{\langle a_{21}:T_{21},d_{21}\rangle,\cdots,\langle a_{2m}:T_{2m},d_{2m}\rangle\}\}$

$[[mdefs]]=_{df}\{op\};\mathbb{D}(mdefs)\vdash op'=op\oplus\{A\mapsto\{m_1\mapsto(exts_{11};exts_{12},c_1),\cdots,m_n\mapsto(exts_{n1};exts_{n2},c_n)\}\}$

该语义的直观意义是：每定义一个方面，就会改变 ASNAME，superclass，op，ANAME 等结构变量的值。

引理 3(方面声明的语义可靠性) 任意一个方面声明的语义都可表示为一个设计。

此引理的证明与引理 1 类似，此处省略。

定义 16 假设 $adecls=(adecl_1;\cdots;adecl_n)$，$pdecls_i=(p_{1i},p_{2i},\cdots,p_{mi})(i=1,\cdots,n)$，则 $adecls$ 的语义定义如下：

$[[adecls]]=_{df}[[adecl_1]];[[adecl_2]];\cdots;[[adecl_n]]$

引理 4(方面声明部分的语义可靠性) 任意一个方面声明部分的语义都可表示为一个设计。

此引理的证明与引理 1 类似，此处省略。

4.5.4 类声明部分的语义

本节使用文献[9]中对单个类声明和类声明序列的定义。

定义 17(私有类声明的语义)[9] 假设在 cdecl 中，定义一个私有类 M，其父类为 N，类 M 有 n 个私有属性，m 个公有属性，k 个保护属性及 l 个方法，则 cdecl 的语义定义如下：

$[[cdecl]]=_{df}\{pricname,superclass,pri,prot,pub,op\};\mathbb{D}(cdecl)\vdash modifyPriCname\wedge modifySuper\wedge modifyPri\wedge modifyProt\wedge modifyPub\wedge modifyOp$

其中：

$modifyPriCname=_{df}pricname'=pricname\cup\{M\}$

$modifySuper=_{df}superclass'=superclass\oplus\{M\vdash N\}$

$modifyPri=_{df}pri'=pri\oplus\{M\vdash\{\langle a_1:T_1,d_1\rangle,\cdots,\langle a_n:T_n,d_n\rangle\}\}$

$modifyPro=_{df}prot'=prot\oplus\{M\vdash\{\langle a_{11}:T_{11},d_{11}\rangle,\cdots,\langle a_{1k}:T_{1k},d_{1k}\rangle\}\}$

$modifyPub=_{df}pub'=pub\oplus\{M\vdash\{\langle a_{21}:T_{21},d_{21}\rangle,\cdots,\langle a_{2m}:T_{2m},d_{2m}\rangle\}\}$

$modifyOp=_{df}op'=op\oplus\{M\vdash\{m_1\mapsto(exts_{11};exts_{12},c_1),\cdots,m_l\mapsto(exts_{l1};exts_{l2},c_l)\}\}$

该语义的直观意义是：类的定义会改变程序的结构，即改变如 pricname，pubcname，superclass，pri，prot，pub，op 这些结构变量的值。

定义 18(公有类声明的语义)[9] 假设在 cdecl 中，定义一个公有类 M，其父类为 N，类 M 有 n 个私有属性、m 个公有属、k 个保护属性及 l 个方法，则 cdecl 的语义定义如下：

$[[cdecl]]=_{df}\{pubcname,superclass,pri,prot,pub,op\};\mathbb{D}(cdecl)\vdash modifyPubCname\wedge modifySuper\wedge modifyPri\wedge modifyProt\wedge modifyPub\wedge modifyOp$

其中：

$modifyPubCname=_{df}pubcname'=pubcname\cup\{M\}$

$modifySuper=_{df}superclass'=superclass\oplus\{M\vdash N\}$

$modifyPri=_{df}pri'=pri\oplus\{M\vdash\{\langle a_1:T_1,d_1\rangle,\cdots,\langle a_n:T_n,d_n\rangle\}\}$

$modifyPro=_{df}prot'=prot\oplus\{M\vdash\{\langle a_{11}:T_{11},d_{11}\rangle,\cdots,\langle a_{1k}:T_{1k},d_{1k}\rangle\}\}$

$modifyPub=_{df}pub'=pub\oplus\{M\vdash\{\langle a_{21}:T_{21},d_{21}\rangle,\cdots,\langle a_{2m}:T_{2m},d_{2m}\rangle\}\}$

$modifyOp=_{df}op'=op\oplus\{M\vdash\{m_1\mapsto(exts_{11};exts_{12},c_1),\cdots,m_l\mapsto(exts_{l1};exts_{l2},c_l)\}\}$

定义 19(类声明部分的语义) 假设 $cdecls=_{df}(cdecl_1;\cdots;cdecl_n)$，则 $cdecls$ 的语义定义如下：

$[[cdecls]]=_{df}Empty;[[cdecl_1]];\cdots;[[cdecl_n]]$

其中：$Empty=_{df}\{\Omega\}$：true $\vdash R(\Omega')$，其中 $R(\Omega')=(pricname'=\Phi\wedge pubcname'=\Phi\wedge superclass'=\Phi\wedge pri'=\Phi\wedge prot'=\Phi\wedge pub'=\Phi\wedge op'=\Phi\wedge ASNAME'=\Phi\wedge superaspect'=\Phi\wedge ANAME'=\Phi\wedge Pointcut'=\Phi\wedge Advice'=\Phi)$。

4.5.5 AO 程序的静态语义

鉴于 AO 程序的声明部分包括类声明部分和方面声明部分，我们将 AO 程序的静态语义定义为类声明部分的语义和方面声明部分语义的复合，其形式化定义如下。

定义 20(AO 程序的静态语义)

$[[cdecls;adecls]]=_{df}[[cdecls]];[[adecls]]$

定理 2(静态语义的可靠性) 任意 AO 程序的静态语义可表示为一个设计。

证明：根据引理 4 和 rCOS 中的静态语义的可靠性[9]及定理 1 可知，该定理成立。

5 示例

图 1 示例程序包括一个类和两个方面的声明，其中类

Point 的声明记作 delPoint,方面 A 的声明记作 delA,方面 B 的声明记作 delB,类声明和方面声明分别是合适定义的[20],其对应的静态语义计算过程如下。

令 $R1(\Omega)=\text{true}\vdash(pubcname'=pubcname\cup\{Point\}\wedge pri'=pri\oplus\{Point\mapsto\{\langle x:int,0\rangle\}\}\wedge op'=op\oplus\{Point\mapsto\{getX\mapsto(\Phi;int\ rx,rx:=x),setX\mapsto(int\ y;\Phi,x:=y),add\mapsto(int\ y;\Phi,x:=x+y)\}\}\wedge pricname'=pricname\wedge superclass'=superclass\wedge prot'=prot\wedge pub'=pub\wedge ASNAME'=ASNAME\wedge superaspect'=superaspect\wedge ANAME'=ANAME\wedge Pointcut'=Pointcut\wedge Advice'=Advice)$ ①

$\because$ $[[cdecls]]$

$=Empty;[[delPoint]]$ (定义 19)

$=\text{true}\vdash R(\Omega');\text{true}\vdash R1(\Omega)$ (①,定义 19)

$=\text{ture}\wedge\neg((R(\Omega');\neg\text{true})\vdash R(\Omega');R1(\Omega)$ (定理 1)

$=\text{ture}\vdash R(\Omega');R1(\Omega)$ (推论 1)

又$\because R(\Omega');R1(\Omega)$

$=\exists\Omega1(R(\Omega1)\wedge R1(\Omega1))$ (式(1))

$=\exists v_1v_2v_3v_4v_5v_6v_7v_8v_9v_{10}v_{11}v_{12}((v_1=\Phi\wedge v_2=\Phi\wedge v_3=\Phi\wedge v_4=\Phi\wedge v_5=\Phi\wedge v_6=\Phi\wedge v_7=\Phi\wedge v_8=\Phi\wedge v_9=\Phi\wedge v_{10}=\Phi\wedge v_{11}=\Phi\wedge v_{12}=\Phi)\wedge(pubcname'=v_1\cup\{Point\}\wedge pri'=v_2\oplus\{Point\mapsto\{\langle x:int,0\rangle\}\}\wedge op'=v_3\oplus\{Point\mapsto\{getX\mapsto(\Phi;\text{int}\ rx,rx:=x),setX\mapsto(\text{int}\ y;\Phi,x:=y),add\mapsto(\text{int}\ y;\Phi,x:=x+y)\}\}\wedge pricname'=v_4\wedge superclass'=v_5\wedge prot'=v_6\wedge pub'=v_7)\wedge ASNAME'=v_8\wedge superaspect'=v_9\wedge ANAME'=v_{10}\wedge Pointcut'=v_{11}\wedge Advice'=v_{12}))$

$=\exists v_1(v_1=\Phi\wedge pubcname'=v_1\cup\{Point\})\wedge\exists v_2(v_2=\Phi\wedge pri'=v_2\oplus\{Point\mapsto\{\langle x:\text{int},0\rangle\}\})\wedge\exists v_3(v_3(\Phi\wedge op'=v_3\oplus\{Point\mapsto\{getX\mapsto(\Phi;\text{int}\ rx,rx:=x),setX\mapsto(\text{int}\ y;\Phi,x:=y),ad\ d\mapsto(\text{int}\ y;\Phi,x:=x+y)\}\})\wedge\exists v_4(v_4=\Phi\wedge pricname'=v_4)\wedge\exists v_5(v_5=\Phi\wedge superclass'=v_5)\wedge\exists v_6(v_6=\Phi\wedge prot'=v_6)\wedge\exists v_7(v_7=\Phi\wedge pub'=v_7)\wedge\exists v_8(v_8=\Phi\wedge ASNAME'=v_8)\wedge\exists v_9(v_9=\Phi\wedge superaspect'=v_9)\wedge\exists v_{10}(v_{10}=\Phi\wedge ANAME'=v_{10})\wedge\exists v_{11}(v_{11}=\Phi\wedge Pointcut'=v_{11})\wedge\exists v_{12}(v_{12}=\Phi\wedge Advice'=v_{12})$ (规则 1)

$=(pubcname'=\{Point\}\wedge pri'=\{Point\mapsto\{\langle x:\text{int},0\rangle\}\}\wedge op'=\{Point\mapsto\{getX\mapsto(\Phi;\text{int}\ rx,rx:=x),setX\mapsto(\text{int}\ y;\Phi,x:=y),add\mapsto(\text{int}\ y;\Phi,x:=x+y)\}\}\wedge pricname'=\Phi\wedge superclass'=\Phi\wedge prot'=\Phi\wedge pub'=\Phi\wedge ASNAME'=\Phi\wedge superaspect'=\Phi\wedge ANAME'=\Phi\wedge Pointcut'=\Phi\wedge Advice'=\Phi)$(简记为 $R2(\Omega')$) (规则 3)

$\therefore [[\ cdecls]]=\text{ture}\vdash R2(\Omega')$

同理可求 $[[adecls]]=\text{ture}\vdash R3(\Omega)$,其中 $R3(\Omega)=(ASNAME'=ASNAME\cup\{A,B\}\wedge Pointcut'=Pointcut\cup\{\langle A,p,Point,d,call(void\ Point.add(int))\&\&target(d)\rangle,\langle B,p,Point,d,call(void\ Point.add(int))\&\&target(d)\rangle\}\wedge Advice'=Advice\cup\{\langle A,adv_1,after,Point,p,\{\text{System.out.println}(\text{“}output\ finish\text{”});\}\rangle,\langle A,adv_2,before,Point,d,p,\{\text{System.out.println}(\text{“}x\ is\ changed\ to\text{”})\}\rangle,\langle B,adv_3,after,Point,a,p,\{\text{int}\ z;\text{a.getX}(\Phi;z));\text{System.out.println}(z);\text{System.out.println}(\text{“}out\ finish\text{”});\}\rangle\wedge pricname'=pricname\wedge superclass'=superclass\wedge prot'=prot\wedge pub'=pub\wedge superaspect'=superaspec)$

$\therefore [[cdecls;adecls]]$

$=[[cdecls]];[[adecls]]$ (定义 20)

$=\text{ture}\vdash R2(\Omega');\text{ture}\vdash R3(\Omega)$

$=\text{true}\vdash R4(\Omega')$ (定理 1)

其中,$R4(\Omega')=\text{true}\vdash(pubcname'=\{Point\}\wedge pri'=\{Point\vdash\{\langle x:int,0\rangle\}\}\wedge op'=\{Point\mapsto\{getX\vdash(\Phi;\text{int}\ rx,rx:=x),setX\vdash(int\ y;\Phi,x:=y),add\vdash(int\ y;\Phi,x:=x+y)\}\}\wedge ASNAME'=\{A,B\}\wedge Pointcut'=\{\langle A,p,Point,d,call(void\ Point.add(int))\&\&target(d)\rangle,\langle B,p2,Point,d,call(void\ Point.add(int))\&\&target(d)\rangle\}\wedge Advice'=\{\langle A,adv_1,after,Point,p,\{\text{System.out.println}(\text{“}output\ finish\text{”});\}\rangle,\langle A,adv_2,before,Point,d,p,\{\text{System.out.println}(\text{“}x\ is\ changed\ to\text{”})\}\rangle,\langle B,adv_3,after,Point,a,p2,\{int\ z;a.getX(\Phi;z));System.out.println(z);System.out.println(\text{“}output\ finish\text{”});\}\}\wedge pricname'=\Phi\wedge superclass'=\Phi\wedge prot'=\Phi\wedge pub'=\Phi\wedge superaspect'=\Phi\wedge ANAME'=\Phi)$

通过该语义可知,该程序定义类 Point 和方面 A 和 B。其中,类 Point 无父类、无公有属性、无保护属性,但包含私有属性 x 和方法 getX,setX 及 add;方面 A 无父类、无父方面、无属性,包含切点 p 和与 p 绑定的 before 通知系 after 通知;方面 B 包含切点 $p2$、与 $p2$ 绑定的 after 通知。这与程序静态结构是一致的。显然,上述声明部分是合适定义的,即语法上是正确的。

结束语　本文在文献[20]的基础上定义了 AOP 的静态语义,该语义用于对面向方面程序声明部分的形式化描述和描述面向方面程序的需求规范(specification),以对需求的正确性进行验证,从而保证软件开发和再工程过程中的正确性。

本文所定义的语义是源语言级别的,不需要转换就能直接表示面向方面程序的需求规范,从而容易被熟悉 AspectJ 的开发人员接受,为定义 AO 程序的动态语义奠定基础。

由于该语义不能描述 AspectJ 程序的行为,我们认为该工作只是 AspectJ 语义研究的开始,在今后的工作中,我们将进一步证明本文所定义的静态语义是 rCOS[9] 静态语义的扩充,即 OO 程序的语义是 AO 程序语义的特殊情形。与此同时,完善 AspectJ 的语义,使其能描述程序的行为,并将其应用到 AspectJ 的静态分析和程序验证。

参 考 文 献

[1] KICZALES G, LAMPING J, MENDHEKAR A, et al. Aspect-oriented programming[C]// European conference on object-oriented programming. Springer Berlin Heidelberg, 1997: 220-242.

(下转第 215 页)

Conference on World Wide Web. 2015:1198-1208.
[14] PIMPLIKAR R,SARAWAGI S. Answering table queries on the web using column keywords[J]. Proceedings of the Vldb Endowment,2012,5(10):908-919.
[15] GUPTA R,SARAWAGI S. Answering Table Augmentation Queries from Unstructured Lists on the Web[J]. Proceedings of the Vldb Endowment,2009,2(1):289-300.
[16] LEHMBERG O,RITZE D,RISTOSKI P,et al. Extending tables with data from over a million websites[C]// Semantic Web Challenge. 2014.
[17] BIZER C. Search Joins with the Web[C]// ICDT. 2014:3.
[18] LEHMBERG O,RITZE D,RISTOSKI P,et al. The Mannheim Search Join Engine[J]. Web Semantics Science Services & Agents on the World Wide Web,2015,35(P3):159-166.
[19] BRAUNSCHWEIG K,THIELE M,EBERIUS J,et al. Column-specific context extraction for web tables[C]// ACM Symposium on Applied Computing. ACM,2015:1072-1077.
[20] EBERIUS J,THIELE M,BRAUNSCHWEIG K,et al. Top-k entity augmentation using consistent set covering[C]// SSDBM. 2015:1-12.
[21] LAUTERT L R,SCHEIDT M M,DORNELES C F. Web table taxonomy and formalization[J]. ACM Sigmod Record,2013,42(3):28-33.
[22] SONG S,ZHANG A,CHEN L,et al. Enriching data imputation with extensive similarity neighbors[J]. Proceedings of the Vldb Endowment,2015,8(11):1286-1297.

(上接第 189 页)

[2] KICZALES G,HILDALE E,HUGUNIN J,et al. An overview of AspectJ[C]// European Conference on Object-oriented Programming. Springer Berlin Heidelberg,2001:327-354.
[3] SPINCZYK O,GAL A,SCHRÖDER-PREIKSCHAT W. AspectC++:an aspect-oriented extension to the C++ programming language[C]// Proceedings of the Fortieth International Conference on Tools Pacific:Objects for internet,mobile and embedded applications. Australian Computer Society,Inc.,2002:53-60.
[4] Aspect R[EB/OL]. [2016-03-01]. http://sowrceforge.net/projects/aspectr.
[5] BONÉR J. What are the key issues for commercial AOP use:how does AspectWerkz address them? [C]// Proceedings of the 3rd International Conference on Aspect-oriented Software Development. ACM,2004:5-6.
[6] HIRSCHFELD R. Aspects-oriented programming with squeak [C]// Objects,Components,Architectures,Services,and Applications for a Networked World. 2003:216-232.
[7] JBoss AOP homepage[EB/OL]. [2016-03-01]. http:// www.jboss.org/jbossaop.
[8] OSSHER H,TARR P. Hyper/J:multi-dimensional separation of concerns for Java[C]// Proceedings of the 22nd International Conference on Software Engineering. ACM,2000:734-737.
[9] JI F H,LI X,LIU Z. rCOS:A refinement calculus of object systems[J]. Theoretical Computer Science,2006,365(1),109-142.
[10] WAND M,KICZALES G,DUTCHYN C. A semantics for advice and dynamic join points in aspect-or iented programming [J]. ACM Transactions on Programming Languages and Systems (TOPLAS),2004,26(5):890-910.
[11] JAGADEESAN R,JEFFREY A,RIELY J. A calculus of untyped aspect-oriented programs[C]// European Conference on Object-oriented Programming. Springer Berlin Heidelberg,2003:54-73.
[12] LÄMMEL R. A semantical approach to method-call interception [C]// Proceedings of the 1st International Conference on Aspect-oriented Software Development. ACM,2002:41-55.
[13] WALKER D,ZDANCEWIC S,LIGATTI J. A theory of aspects [J]. ACM SIGPLAN Notices,2003,38(9):127-139.
[14] TUCKER D B,KRISHNAMURTHI S. Pointcuts and advice in higher-order languages[C]// Proceedings of the 2nd International Conference on Aspect-oriented Software Development. ACM,2003:158-167.
[15] MASUHARA H,KICZALES G. Modeling crosscutting in aspect-oriented mechanisms[C]// European Conference on Object-Oriented Programming. Springer Berlin Heidelberg,2003:2-28.
[16] TABAREAU N. Aspect Oriented Programming:a language for 2-categories[C]// Proceedings of the 10th International Workshop on Foundations of Aspect-oriented Languages. ACM,2011:13-17.
[17] AVGUSTINOV P,HAJIYEV E,ONGKINGCO N,et al. Semantics of static pointcuts in AspectJ[J]. ACM Sigplan Notices,2007,42(1):11-23.
[18] FIGUEROA I,TANTER É. A semantics for execution levels with exceptions[C]// Proceedings of the 10th International Workshop on Foundations of Aspect-oriented Languages. ACM,2011:7-11.
[19] GANG X,BO Y,MINGYI Z. A Semantics of Poincuts in AspectJ[J]. IERI Procedia,2013,4:323-330.
[20] XIE G,ZHANG M Y,YANG B. A STATIC SEMA NTICS FOR ASPECTJ[J]. Journal of Computer Tational Information Systems,2012,8(16):6951-6962.
[21] 王砚霖,王世耆. 面向方面编程和 AspectJ[OL/EB]. [2016-03-01]. http://www.creativepioneer.com/paper/AOP_and _AspectJ.pdf.
[22] HOARE A R C,HE J. Unifying theories of program ming[M]. Englewood Cliffs:Prentice Hall,1998.
[23] MOLDEREZ T,JANSSENS D. Modular Reasoning in Aspect-oriented Languages from a Substitution Perspec tive[J]. Transactions on Aspect-oriented Software Develop ment XII. Springer Berlin Heidelberg,2015:3-59.
[24] ZHANG Q,KHEDRI R. On the weaving process of aspect-oriented product family algebra [J]. Journal of Logical and AlgebraicMethods in Programming,2016,85(1):146-172.
[25] 陆钟万. 面向计算机科学中的数理逻辑(第二版)[M]. 北京:科学出版社,2002:117-118.