63-6715

程序验证

程序设计
验证

EIFFEL 132 (12)

计算机科学 1994Vol. 21No. 2

面向对象程序设计语言 EIFFEL 的 程序正确性验证工具

TP312

姚卿达 唐长宁 董慧红

(中山大学计算机软件研究所 广州 510275)

A

摘要 本文简单介绍了国家八五科研重点攻关项目《实用自动工具集的开发》的子专题《自动验证工具》的理论基础和开发成果。

关键词 Hoare 逻辑, OOP(面向对象的程序设计)

一、引言

程序正确性验证已是一个有几十年历史的课题了。过去几十年中,过程性语言占计算机程序设计语言的大多数,所研究的主要是一些过程性语言程序的正确性问题,并由此而引出了几种程序正确性验证的方法,其中最主要的方法当推公理方法。

因实际需要而发展起来的面向对象的程序设计语言现在也遇到了程序正确性验证的问题。由于面向对象的程序设计语言在概念上与传统的过程性程序设计语言有较大的差别,因此传统的程序正确性验证的方法是否还可以适用于面向对象程序设计语言的程序,是值得考虑的。这既涉及到面向对象程序设计语言本身的定义,也涉及到程序正确性的定义。

下面,我们讨论的面向对象的程序设计语言是 EIFFEL 语言,其验证方法是以 Hoare 逻辑为基础的公理方法。

二、面向对象的程序设计语言

尽管目前已有一些程序设计语言被称为面向对象的程序设计语言,但并不存在一个完美的数学基础,甚至连精确的定义也没有,只有一些非形式化的描述。文[2]中指出,对象式设计是把软件系统构作成类(或谓抽象数据类型的实现)的结构化集合。那么,所谓

面向对象的程序设计语言,应该具有支持对象式设计的机制。目前使用的面向对象的程序设计语言主要分为两个大类:一是提供面向对象的支持机制,但本身仍然具有主程序概念的,如 C++;二是已没有主程序概念、以类为程序结构单元的,如 EIFFEL。单从对客观世界的模拟而忽略其它方面来说,EIFFEL 更接近于“纯粹”的面向对象的语言,但这样也就带来了一个问题:在 EIFFEL 语言的程序文本中,没有对整个程序的运行的直接描述。

三、程序正确性验证的公理方法

在目前所见的大部分文章和书籍当中,用公理方法描述的程序正确性被分为两种:

1. 完全正确性断言 $\{\varphi\} S \{\psi\}$: 若程序 S 的执行开始于一个满足 φ 的状态,则这个执行必在有限时间内终止且终止时的状态满足 ψ 。

2. 部分正确性断言 $\{\varphi\} S \{\psi\}$: 若程序 S 的执行开始于一个满足 φ 的状态,且若此执行能在有限时间内终止,则终止时的状态满足 ψ 。

在文[4]中,是这样描述程序正确性证明的定义的:

设 $\{\varphi, \psi\}$ 是一个规范, S 是按照这个规范要求设计的,且是由语句 $S_1, S_2, \dots, S_n$ 组

姚卿达 教授,软件所所长,岭南(大学)学院副院长。唐长宁 硕士生,研究方向为程序正确性验证。

成的一个枚举型程序（所谓枚举型程序是指其执行等于组成它的各语句的逐一顺序执行，其中每个语句都只有一个入口和一个出口，且没有 GOTO 转移）。令 $\varphi_1, \psi_1, \varphi_2, \psi_2, \dots, \varphi_n, \psi_n$ 是 $2n$ 个谓词， $\varphi_i = \varphi, \psi_n = \psi$ ，如果所有断言 $\{\varphi_i\} S \{\psi_i\}$ ($i=1, 2, \dots, N$) 为真，且每个蕴涵 $\psi_i \Rightarrow \varphi_{i+1}$ ($i=1, 2, \dots, n-1$) 成立，就称 $(\varphi_1, \psi_1), (\varphi_2, \psi_2), \dots, (\varphi_n, \psi_n)$ 是 $\{\varphi\} S \{\psi\}$ 的一个证明。

在这里我们强调一点：无论是完全正确性证明还是部分正确性证明，它所描述的是程序的执行。

四、EIFFEL 语言程序的正确性验证公理方法

过程性程序的文本中已经指出程序如何执行，因此所谓程序执行时的验证，只要按照程序文本的“执行”（一种想象中的动态过程）进行验证，如果程序的执行结果（抽象机而非物理机的执行结果）与其规范说明所描述的语义是一致的，则该程序是正确的。

但对于 EIFFEL 这种面向对象的程序，情况却不是这样。在面向对象的程序文本中根本见不到整个程序执行的过程，所见到的都只是一些静态的类，如果不修改程序正确性验证的定义，则在这种情况下原有的按照程序文本进行证明的方式无法保证整个程序系统的正确性，必须采用新的手段。立即可以想象到的办法有两种：①采用适应于面向对象程序设计语言的新证明方式；②根据面向对象的程序的文本生成一个虚拟的执行过程，即将面向对象的程序转换为一种离散的、可依照原有公理方法进行验证的语句序列，在这种转化中，并不改变面向对象程序的基本性质——封装、继承等等。

在上述的两种手段中，方式①从目前来看其难度不好评价，因为这样做意味着也许需要一整套新的理论和方法，可能难以利用现有的研究成果和各种理论。方式②相对来说比较自然一些，因为就目前的计算机技术和计算机体系结构而言，面向对象主要是一

种设计方法而非执行方法，面向对象的程序的最终执行仍然是过程性的。

注意，这种做法并不是将程序的抽象语义与具体的实现不恰当地结合起来。对于面向对象的程序系统本身来说，要完成一个任务，必定会有一个启动和执行的流程。

但是由此也引出了这样一个问题：将面向对象的程序转换为一个虚拟的执行过程，就脱离了传统的程序正确性验证的基于“抽象机”的概念，即不再是验证原本的源程序，而是验证一个已经经过加工的“程序”，从而存在着因转换程序的错误而导致验证结果不能正确反映程序实际情况的危险。要在理论上克服这个问题，会涉及到程序正确性证明本身的局限性，即对程序语义所必须采取的抽象观点以及形式验证过的程序与其行为之间的差异。由于从一个程序文本所描述的抽象机到其物理执行之间必须经过许多转换过程，所以每个过程都有可能引入导致程序的物理执行偏离程序语义的因素。目前这个问题在程序验证学界仍然有争议（从而导致对程序正确性验证的实际意义的怀疑）。我们打算介入这种争议，而是采用一些实际的准则来完成这种转换，这些准则使得转换停留在尽可能高的抽象层次。

按照以上思路，我们把 EIFFEL 语言的验证分为两个层次：一是类级的正确性，二是系统级的正确性。

五、类级正确性

在文[1]中给出了类的正确性定义如下：

定义 1 一个类称为是正确的，当且仅当它是一致的，并且类中的每个程式都是检查正确的、循环正确的。

所谓一个类 C 是一致的，是指它满足：① $\{\text{pre}_{\text{Create}}\} \text{do}_{\text{Create}} \{\text{INV}_C\}$ ；② 对于类 C 的每个程式 r ， $\{\text{pre}_r \wedge \text{INV}_C\} \text{do}_r \{\text{post}_r \wedge \text{INV}_C\}$ 。

所谓程式是检查正确性的，是指该程式的每条检查语句 c 的执行均满足其中所列的断言。

所谓程式是循环正确的，是指对于该程

式包含的每个循环都有:① {true} INIT {INV};② {true} INIT {VAR ≥ 0};③ {INV and not EXIT} BODY {INV};④ {INV and not EXIT and VAR = v₀} BODY {0 ≤ VAR ≤ v₀}。

这里 INV_c 表示类 C 的不变式,pre_r 表示程式 r 的前置断言,post_r 表示程式 r 的后置断言,VAR 是循环变式,EXIT 是循环出口条件,BODY 是循环体。

文[2]扩充了类的正确性,增加了异常处理正确性的要求。

所谓程式是异常处理正确的,是指:①补救语句通过重执命令 retry 而结束时,满足程式的后置条件和类不变式;或②补救语句没有通过执行重执命令 retry 而结束时满足类不变式。

详细的定义和推导请参阅文[1]和[2]。

在面向对象的程序系统的类的正确性证明中,我们提请注意这样一点:按照面向对象的思想原则,任何一个类的内部数据和操作对其它类来说是不可见的,当其它类的实例按照正确的手段来调用该类的实例而产生的错误应由该类负责。因此,在任何一个类的正确性证明过程中,不应该涉及其它与之相关的类的正确性证明。即在该类的正确性证明过程中,凡是涉及到其它类的过程调用,我们都假定该调用是正确的。

定义 2(调用无关条件下的正确性) 一个满足异常处理正确性的类 C,对于其中的任何一个程式 r,如果该程式中存在对其它类的特征调用 {pre}S{post},这里 {pre}和 {post}分别是程式 r 中语句的前置条件和后置条件,只要在进行如下置换之后程式 r 满足检查正确性和循环正确性,就称类 C 在调用无关条件下是正确的。

①对于形如 {pre}TARGET.FNAME {post}的过程性特征调用,将其后置条件换为 {pre ⊢ post}。

②对于形如 X=TARGET.FANME(函数调用)或 X:TARGET.FANME(属性调

用)的非过程性特征调用,把程式 r 中所有对 X 的操作的后置条件 {post}都置换成后置条件 {pre ⊢ post}。

显然,类在调用无关条件下的正确性不强于类的正确性,因为在类的正确性定义中并没有明确地指出调用无关条件。

定义 3(面向对象程序系统的类级正确性) 一个面向对象的程序系统是类级正确的,如果该程序系统的任何一个类在调用无关条件下都是正确的。

六、系统级的正确性

为了给出系统级正确性的定义,首先要对面向对象的系统进行形式化的定义。

在面向对象的系统(以下简称系统)中包含两种机制:一是以类为模板产生的对象,二是对象之间的通信。

对象进行通信时相互之间传递的消息分为两个子集,一个称为请求消息集,记为 ReqM,表示计算任务;另一个称为完成消息集,记为 FinM,表示计算结果。这里有 $ReqM \cap FinM = \emptyset$ 。

将一个对象定义为一个六元组 (MA, RM, DS, CS, ds, B)。其中 MA 为通信地址;RM 为识认消息集;DS 为数据状态集;CS 为通信状态集;ds ∈ DS 为初始数据状态集;B DS × CS × E → DS × CS × E × 2⁰ 为行为,这里 E 为事件集,O 为对象集。

定义 4(事件) 事件是一个三元组 (s, d, m),其中 s 和 d 是对象地址,m 是消息;事件 (s, d, m)表示对象 s 向对象 d 发送消息 m。

定义 5(通信) 对象 a 和对象 b 的一次通信是指对象 a 和对象 b 之间先后发生的一对事件 (a, b, m)和 (b, a, m'),这里 m ∈ ReqM, m' ∈ Fin(m)。通信 (a, b, m)(b, a, m')可以统一描述为 (a, b, (m, m'))。

为了定义系统级的正确性,我们关心的是程序的系统运行的形式描述。面向对象的程序系统的运行是用系统格局序列来描述的。

定义 6(格局) 面向对象的系统的格局是一个三元组(当前事件,当前所含对象,每个对象的状态),记为 (e, obj, CON) , $e \in E$ 为事件, obj 为对象集; CON 定义为集合 $\{(a, f) \mid A \in obj, MA(A) = a, f \in DS(A) \times CS(A)\}$, 且 $A \in obj, (MA(A), f) \in CON \rightarrow (a, f) \in CON, A \in obj$, 使得 $MA(A) = a$, 且 $f \in DS(A) \times CS(A)$ 。

作为一种简单的表示,也可以用事件序列来描述系统运行。

定义 7(合法格局) 一个格局称为合法格局,是指事件中消息接收者为当前系统中存在的某对象,且该对象中存在处理该消息的行为,并且该消息满足实现该行为的条件。

事实上,由于消息传递是由特征调用来实现的,因此合法格局的另一种说法就是:在调用某个类的特征之前已经创建了该类的实例,且该类中存在所要调用的特征,且满足特征调用的前置条件。

定义 8(格局转换) 格局 (e_2, obj_2, CON_2) 称为是由格局 (e_1, obj_1, CON_1) 转换而来的,如果格局 (e_1, obj_1, CON_1) 是合法格局,并且在当前事件 e_1 完成之后得到格局 (e_2, obj_2, CON_2) 。

关于对象和面向对象的系统的形式化定义详见文[2]。

我们已经知道,在 EIFFEL 语言中没有主程序的概念,是由一些相关(继承,允引等等)的类构成一个系统(SYSTEM),在这些类之中要指定一个根类(ROOT CLASS),以启动整个系统的运行。系统的启动执行过程是:

1. 创建启动类的实例。
2. 向该实例发送消息(特征调用)。

如果启动类的创建方法中已包含了系统的启动动作(发送消息—特征调用),则只要创建启动的实例即可启动整个系统。

定义 9(启动正确性) 以类 C 为根类的面向对象的程序系统 S 称为启动正确的,是指:① C 中有自定义或系统缺省定义的对象

创建程式。②如果 c 是系统初始对象,且由程序系统 S 之外向 S 发消息 (a, c, m) 来启动系统,则必有 $m \in RM$ (RM 是对象 c 的识认消息集)。

定义 10(格局转换正确性) 从合法格局 (e_i, obj_i, CON_i) 到格局 $(e_{i+1}, obj_{i+1}, CON_{i+1})$ 的转换是正确的,如果格局 $(e_{i+1}, obj_{i+1}, CON_{i+1})$ 是合法的。

回想一下过程性语言的正确性,它包括部分正确性和完全正确性两种含义。类似地,由于面向对象程序系统中的特征调用是可以循环的,因此其正确性也包含两个方面:启动正确性和格局转换正确性保证系统的正确运行,而循环特征调用的终止性保证程序系统的运行可以终止。

定义 11(终止正确性) 如果一个面向对象的计算序列的第一个事件是 (a, a_1, m) , 计算序列有限,最后一个事件是 (a_n, a, m') , 这里 m' 是关于请求消息 m 的完成消息,则称该程序系统是终止正确的。

定义 12 由事件 (a, b, m) 引发的面向对象的计算过程是满足下列条件的格局转换有限序列 $c_1, c_2, \dots, c_n$:

$$\textcircled{1} c_i \Rightarrow c_{i+1}, i = 1, 2, \dots, n-1.$$

$\textcircled{2} c_1 = ((a, b, m), Obj, CON)$, 这里为系统接收消息时的初始对象集。 $CON = \{(c, f) \mid A \in Obj, c = MA(A), f = (ds(A), idle)\}$ 。

$\textcircled{3} c_n = ((b, a, m'), Obj, CON')$, $m' \in Fin(m)$ 。

有了上述的定义,系统级正确性的定义描述如下:

定义 13(系统级的正确性) 一个面向对象的程序称为是系统级正确的,当且仅当它的任何一个计算任务其过程格局序列都是合法的,即该计算任务是启动正确的、格局转换正确的和终止正确的。

启动正确性和格局转换正确性可以直接通过被调用特征的前置条件和后置条件来检查,但终止正确性的证明则复杂很多,我们将另文专述。

七. 类级正确性和系统级正确性的关系及证明

在一个正确的软件系统中,必须保证只有当前置条件成立时才可能使用程式,并且程式的调用结果满足后置条件。类级的正确性定义可以保证:如果程式调用开始时满足前置条件,则这个执行必在有限时间内终止且终止时满足后置条件。即 $\{\varphi\}S\{\psi\}$ 表示如果 φ 成立,则执行 S 后 ψ 成立。但是, φ 的正确性不是由程式 S 保证的,而是由调用者决定的。因此类级的正确性不能代表软件系统的正确性,它只代表待证的程序系统的“静态正确性”,只有加上系统级的正确性,即“动态正确性”才完整。而在系统级的正确性证明中,其格局转换正确性是由类级正确性提供保证的。

类级的正确性显然可以采用原有的证明方法。由于有了对面向对象的系统运行的形式化描述,该描述满足正确性证明公理方法的条件,而系统级的证明也可以是用公理证明方法。但在这里,我们略微修改程序正确性证明的本来定义,对程序系统的格局序列而不是程序文本进行证明。

定义 14(面向对象的程序系统的正确性证明) P 是一个面向对象的程序系统,由类 $C_1, C_2, \dots, C_n$ 组成(n 为有限自然数), C_1 是其根类。 S 是这个程序系统的执行的格局转换序列。 (φ, ψ) 是一个规范说明,用于描述 P 的执行序列 S 。令 $\varphi_1, \psi_1, \varphi_2, \psi_2, \dots, \varphi_n, \psi_n$ 是 $2n$ 个谓词, $\varphi_1 = \varphi, \psi_n = \psi$, 如果所有断言: $\{\varphi_i\} S_i \{\psi_i\}, (i=1, 2, \dots, N)$ 为真,且每个蕴涵 $\psi_i \Rightarrow \varphi_{i+1}, (i=1, 2, \dots, n-1)$ 成立,就称 $(\varphi_1, \psi_1), (\varphi_2, \psi_2), \dots, (\varphi_n, \psi_n)$ 是 $\{\varphi\}S\{\psi\}$ 的一个证明。

八. EL 语言的程序正确性自动验证

程序正确性的自动验证的一般方法分为这样几步:

- 1) 给出该语言的语法定义;
- 2) 给出该语言的每个语法单位的语义定义;
- 3) 对于被证明的程序给出其规范说明;

4) 由条件产生器根据程序及其规范说明产生待证定理;

5) 由定理证明器证明条件产生器所产生的条件。

作为 EIFFEL 语言程序自动验证工具的前期开发工作,我们建立了一种“简化”的 EIFFEL 语言,称为 EL 语言,它是 EIFFEL 语言的子集,具有 EIFFEL 语言的所有面向对象的本质的特性。我们的最终目标在于 EIFFEL 语言而非 EL 语言,因此不打算对 EL 语言本身的逻辑问题作过于形式化的讨论。从语法上,EL 语言主要是限制了 EIFFEL 语言的多继承,另外没有延迟类,其余方面与 EIFFEL 语言基本上是一模一样的。

EL 语言的主要部分的公理语义如下:

1. 函数

$$\frac{\vdash [\varphi]F(u)[\psi]}{\vdash [\varphi^f]F(u^f)[\psi^f]}$$

其中 f 为函数名, F 为函数体

2. 属性 视为零元函数

3. 过程

$$\frac{\vdash [\varphi]R(u)[\psi]}{\vdash [\varphi^r]r(v)[\psi^r]}$$

$$\frac{\vdash [\varphi]R(u)[\psi]}{\vdash [\varphi^a, r]a, r(v)[\psi^a, r]}$$

其中 u 为一组形参, v 为对应的一组实参,为过程名,为过程体,为所属类型中的属性组。

4. 类的公理语义

$$\vdash [\varphi^t, \dots, \varphi^t]t, p, (a_1, \dots, a_n)[\psi]$$

其中, t 是被说明为类型 T 的变量,类型 T 的基类就是类 T , p 是类 T 中的某个外部可见的特征, f_i 是与类 T 的特征 p_i 对应的函数,定义这个函数的目的在于,对于类 T 的每个特征调用: $t, p_i(a_1, \dots, a_n)$ 相当于执行赋值语句: $t := f_i(t, a_1, \dots, a_n)$ 。这也就是上述类的公理语义的来源。

其余公理和规则与常见的过程性程序设计语言一致,在此略去。

九. 结束语

程序的正确性验证是一个难度较大的课

(转第 5 页)

束集化简功能通过消去不必要的变量以及一定的冗余元素,简化约束体系。

4)不等关系的综合处理。Prolog II 能够使得不等约束被应用到任何项上。也就是不论原子、变量还是元组,均可施以不等约束。项的这种一般性处理在约束引起不同代数间的影响时,要求对这些约束做良可调的管理,类似于不同类型的量的运算那样。

4. 关于表和元组

Prolog 中,表概念几乎无所不在。Prolog II 中,表概念被扩展为元组的概念。元组是带有连接运算的有限树序列。在这些对象上,可以应用不等或相等约束。可以把一个元组加以约束,使其具有确定个数的元素。

元组上的约束的主要限制是:在任一连接运算中,位于连接符左边的元组(子元组)的元素个数必须是已知的。虽然如此,仍使用一个自动的延迟机制处理约束,从而使得元组上的约束不受前述限制。

表是一个在 Prolog 中具有特定含义的数据结构。通过以上提到的元组代数方式,可以将表表示为更为简单并且可更为高效地加以处理的形式,对它,处理速度之快令人吃惊。例如,通过使用适当的约束,可以获得一个元组的任一元素的直接访问,这使得元组更为接近数组(达到按下标访问的效果)。在一些特别强调效率的场合,比方说自然语言处理,就应充分应用元组这种数据结构的

特点。

Prolog II 中,所有先前 Prolog 中使用的各种类型的表处理可以容易地通过元组和连接的方式加以执行。为了与已有程序兼容,Prolog II 也支持以艾丁堡文法写的标准 Prolog 表的使用(事实上,Prolog II 同时支持艾丁堡文法)。

5. 程序举例

这里给出一个用 Prolog II 写的递归排序程序,它应用了元组的特性。

```
sort ((), ()) →;
sort (L, U·(X)·V) → partition (L, X, U'·V')
                        sort (U', U)
                        sort (V', V);
partition (L, X, U'·V') →
  truncate (N/2, N') /* N 是 L 的元素数,除以 2
  regroup (X, L', U'·V'), (截尾除) */
  {L::N,
   U::N',
   L=U·(X)·V,
   L' =U·V};
regroup (X, (), ()) →;
regroup (X, (X')·L, (X')·U, V) →
  regroup (X, L, U, V),
  {X' <= X};
regroup (X, (X')·L, U, (X')·V) →
  regroup (X, L, U, V),
  {X' > X};
```

四、结束语

当今,欧洲及其它一些国家在 Prolog 方面从事着较以往更为务实的工作,力图在实用性方面也使得它能与其它语言媲美。在工业化应用中充分发挥作用。根据最近从国际上了解到的情况来看,Prolog 语言及其开发应用仍然是一个热点。

本文参考了 IJCAI' 93 会议的有关资料。

致谢 作者对陆汝钫教授的帮助诚致谢意。

(接第 67 页)

题,即使是过程性的语言,其程序正确性验证也尚未完全进入实用阶段。作为后起之秀的面向对象的程序设计语言和设计方法,还有很多问题值得探讨和研究。一方面,面向对象的程序虽然在程序结构上有其特点,但其数据操作与过程性的语言是基本一致的,这使得我们可以采用传统的方法进行正确性验证。另一方面,针对面向对象的特点,必然要采取一些与之相适应的方法,要在传统的理论上作一些新的探讨。

参考文献

- [1] B. Meyer, Eiffel: The language, Prentice Hall, 1992
- [2] 徐家福、王志坚、瞿成祥,对象式程序设计语言,南京大学出版社,1992
- [3] 陆汝钫,计算机语言的形式语义,科学出版社,1992
- [4] 陈火旺、罗朝晖、马庆鸣,程序设计方法学基础,湖南科学技术出版社,1987
- [5] Hoare, C. A. R., An Axiomatic Approach to Computer Programming, Comm. of the ACM, Vol. 12, No. 10, 1969
- [6] Dijkstra, E. W., A Discipline of Programming, Prentice Hall, Englewood Cliffs, 1976