第6卷第7期 软件学报 Vol. 6, No. 7
1995年7月 JOURNAL OF SOFTWARE Jul. 1995

Smalltalk—80的指称语义研究*

李舟军 王兵山

(长沙工学院计算机系,长沙 410073)

摘要 Smalltalk—80是原型的面向对象程序设计语言和环境.本文简要地给出了Smalltalk—80的形式模型,并基于该模型描述了Smalltalk—80的静态和动态指称语义.

关键词 面向对象程序设计,形式定义和理论,指称语义.

Smalltalk—80[1](以下简称Smalltalk)是最有代表性的面向对象程序设计(OOP)语言和环境,它所体现的OOP方法学已广泛应用于计算机科学的各个领域.OOP被誉为80年代的结构程序设计,许多学者认为OOP将成为90年代计算机主流技术之一.

鉴于Smalltalk在面向对象语言中的典型性和重要性,有必要对它进行更深入的研究.文献[2]给出了一个描述Smalltalk指称语义的初步框架,但有许多关键问题尚未得到处理.我们扩充了该模型,并已完成除进程以外的所有Smalltalk成份的指称语义描述[3].

1 Smalltalk的基本概念和对象模型

1.1 Smalltalk的基本概念

Smalltalk的信息表示和信息处理是基于"对象—消息"模式的,其中最根本的设计原则就是高度的统一性:它试图用极少几个基本概念(对象,类,消息和方法)统一语言的各个方面,表达系统的不同层次——从用户界面到程序模块.

对象是对客观世界实体的抽象和模拟,Smalltalk系统内的任何元素都表示为对象.对象由若干实例变量和一组方法(消息)构成.对象的实例变量构成其内部状态,对象所能响应的消息构成其外部接口,接口是对象之间相互作用的唯一途径.

类是定义和产生实例对象的标准模板,是Smalltalk系统的基本构件.为了有效地组织各种类,Smalltalk引入了子类法和继承机制,这特别适宜于模拟现实世界中具有层次结构的模型.对象都是类的实例,类是其相应元类的唯一实例.

Smalltalk的所有处理(计算/通讯/控制)都是通过向对象发送消息来实现的.消息是要求某个对象完成指定操作的请求,而方法描述对象为响应消息而实施的操作的具体细节.

1.2 对象存储器和对象的表示

* 本文1993—05—26收到,1994—01—03定稿

本课题得到国家高技术计划和国防科技预研基金的资助.作者李舟军,1963年生,讲师,主要研究领域为计算机科学理论,形式语义学,面向对象方法学.王兵山,1938年生,教授,主要研究领域为计算机科学理论.

本文通讯联系人:李舟军,长沙 410073,长沙工学院计算机系

Smalltalk 系统中的所有对象(包括类和方法)都驻存于统一的永久的对象存储器中,对象存储器(记为 σ)是运行 Smalltalk 程序的基础,可抽象为:

$$\text{Object}-\text{memory}=\text{Oop}\xrightarrow[m]{}\text{Object}$$

每个对象由系统内部统一生成的内部名来标识,通常称为对象指针,记为 Oop. 因为每个对象都是某个类的实例,故对象由类标记和对象体(即内部状态)构成,因此对象可抽象为:

$$\text{Object}::\text{Class}:\text{Oop}$$
$$\text{Body}:\text{Object}-\text{body}$$
$$\text{Object}-\text{body}=\text{Primitive}-\text{object}\mid\text{Plain}-\text{object}$$
$$\text{Primitive}-\text{object}=\text{Int}\mid\text{Real}\mid\text{Bool}\mid\text{Char}\mid\text{Symb}\mid\text{Nil}\mid\text{Block}\mid\text{Primitive}-\text{method}\mid\text{Method}-\text{body}$$
$$\text{Plain}-\text{object}=(\text{Id}\mid\text{Nat}_1)\xrightarrow[m]{}\text{Oop}$$

Smalltalk 系统中的对象,根据其内部状态的不同可分为原始对象和一般对象:

(1)原始对象不含实例变量,其内部状态一成不变,可用于定义基本的数据值,例如整数,实数,真值,字符,符号和无定义值均为原始对象,同时我们把方法和块的语义指称物作为原始对象存放在对象存储器中,以供语义解释过程使用.

(2)一般对象含有命名(或索引)实例变量,实例变量通过相应的指针指向其它对象,一般对象的内部状态只能通过其本身具有的操作(方法)来改变.

Smalltalk 系统将类(元类)作为结构相同的完全对象存储在对象存储器中. 类对象由其类标记(指向元类的指针)和相应的体两部分构成,类对象具有若干实例变量以描述该类各方面的属性. 元类对象由其类标记(指向类 Metaclass 的指针)和相应的体两部分构成,其主要作用是保存类方法.

1.3 基于对象存储器的若干抽象函数

由上可知,Smalltalk 程序运行所需的任何信息都保存在对象存储器中,因此必须构造一组抽象函数,从这些信息的具体存储表示映射到其抽象表示,使得我们在语义描述时既能利用这些信息,又能避免琐碎的细节处理. 抽象函数分为以下两种:

(1)涉及对象存储器的整体性质,而不针对具体类(元类). 如下面两个抽象函数分别用于确定对象存储器中所有类对象的类名和指针:

$$\text{Class}-\text{name}:\text{Object}-\text{memory}\to\text{Id}_{-\text{set}}$$
$$\text{Class}-\text{oop}:\ \text{Object}-\text{memory}\to\text{Oop}_{-\text{set}}$$

(2)用于确定对象存储器中给定类(元类)的有关方面的信息. 例如:

$$\text{Super}-\text{class}_1:\text{Id}\times\text{Object}-\text{memory}\xrightarrow{\sim}[\text{Id}]$$
$$\text{Super}-\text{class}_2:\text{Oop}\times\text{Object}-\text{memory}\xrightarrow{\sim}\text{Oop}$$

关于这些抽象函数的完整定义和详细说明见文献[3].

2 Smalltalk 的静态语义描述

静态语义用于描述语言的语法结构出现在任何地方所应满足的最低限度的上下文条件,即说明什么样的程序在形式上是良构的(well-formed).

Smalltalk 不是强类型语言,无变量的类型说明,且具有动态链接的特性,因此其语法检查必须分两步进行:

(1)静态语法检查:相当于编译时的语法检查,这是通过静态语义来完成的,主要包括以下几方面:变量的使用是否正确?某些必要的语法成分是否省缺?赋值是否针对伪变量?

(2)动态语法检查:相当于解释执行时的错误检查,我们在动态语义中加入必要的检查,以弥补静态语义之不足.

为了描述静态语义,必须引入静态环境 SEnv、表达式层次论域 HExp 和真值论域 Bool 这三个静态语义论域,并定义如下一组静态语义函数:

(1)$WF_{Program}$: Program→Object－memory→Bool

(2)$WF_{Expression}$: Expression→HExp→SEnv→Bool

(3)WF_{Class}: Class－body→SEnv→Bool

(4)WF_{Method}: Method－body→HExp→SEnv→Bool

上述静态语义函数分别用于确定什么样的程序(表达式、类和方法)是良构的.关于静态语义定义的细节见文献[3].

3　Smalltalk 的动态语义描述

3.1　Smalltalk 的语义描述方法及特点

为简明直观起见,我们在描述 Smalltalk 的指称语义时,以 VDM 元语言为工具,同时采用直接语义的方法,即每个语法结构直接映射到其语义指称物(状态转换函数).VDM 元语言具有严格的数学基础,用它表出的语义论域的存在性(构成完全偏序集)和算子(函数)的连续性,可以得到充分的保证[4,5],故联立语义方程存在唯一的最小不动点,可作为语法结构的指称物,从而完成语义函数的定义.

Smalltalk 语言不同于一般的高级语言,其语义描述具有以下特点:

(1)动态环境 DEnv(其元素记为 δ)是方法(程序,块)执行时的当前局部环境(类似于 Smalltalk 的 Context),主要用于记录接收器,参变量和临时变量的值.接收器和参变量的值是在方法与相应的消息匹配时确定的,且在方法的执行过程中保持不变,而临时变量在方法中被初始化为 Nil,在执行过程中可不断改变.

(2)方法－类环境 MCEnv(其元素记为 ρ)用于确定当前正在执行的方法的类属,主要支持 super 机制的实现.对于任何 Smalltalk 程序(不属于任何类),其方法－类环境为 mk－MCEnv(Niloop).

(3)为处理方法中的返回表达式,我们在相应的语义论域中设置返回值标记 Retvalue,以便确定方法执行后的返回值.

(4)与方法不同,Smalltalk 的块有两种不同的返回:局部返回和非局部返回.非局部返回由块中的返回表达式提示.为处理非局部返回,我们在方法体中设置语法分量 Home,以记录方法体中是否定义有含返回表达式的块,并对非局部返回起定位作用;同时在相应的语义论域中设置非局部返回标记 RetHome,以确定是否发生非局部返回.

(5)方法的语义指称物 Method 的含义如下:在给定的动态环境下,方法以接收器和参数表为输入,以结果对象,返回值标记,非局部返回标记和另一动态环境为输出,同时改变系

统状态.对于其体为 body 的方法,指称物形如 $M_{Method-body}[body]\rho$,对于无体的原始方法,其指称物是直接的.

(6)Smalltalk 的消息发送可以看成是动态约束的方法调用,我们通过递归函数 find 来描述方法的查找过程.

(7)共享变量(全局变量、pool 变量和类变量)均以池的形式成组地存放在相应字典中.

(8)为处理实例变量、类变量和 pool 变量的继承机制,我们提供了若干递归函数.

(9)系统状态 State(其元素记为 s)由对象存储器 Object－memory 和文件系统 File－system 两部分构成.关于文件系统的描述见文献[3].

3.2 Smalltalk 的动态语义论域

为描述 Smalltalk 的动态指称语义,我们构造了一组动态语义论域[3],下面仅列出其中最重要的几个:

(1)DEnv::Rcvr:Oop

Args:$Id\xrightarrow[m]{}Oop$

Temps:$Id\xrightarrow[m]{}Oop$

(2)MCEnv::Class:Oop

(3)State::OM:Object－memory

FS:File－system

(4)Pg=State→State

(5)Exp=DEnv→State→Oop×Retvalue×RetHome×DEnv×State

(6)Method=DEnv→Oop×Oop*×State

→Oop×Retvalue×RetHome×DEnv×State

3.3 Smalltalk 的动态语义定义

下面以 Smalltalk 的若干语法成分为例,分别给出其动态指称语义定义.定义依照抽象文法,语义函数和语义方程的次序表出,并予以必要的注释,完整的动态语义定义见文献[3].

1.程序:

Program::Temps:Id_{-set}

Exprs:Expression－list

$M_{Program}$:Program→Pg

$M_{Program}[mk-Program(temps,exprs)]\triangleq\lambda s.$

let $\rho=mk-MCEnv(Niloop)$ in

let $\delta=mk-DEnv(Niloop,[\ ],[id\mid\rightarrow Niloop\mid id\in temps])$ in

let $(oop,rv,rh,\delta',s')=M_{Expression-list}[exprs]\rho\delta s$ in s'

注:程序由临时变量说明和作为可执行部分的表达式序列组成,不属于任何类,无接收器,无参数,其作用就是改变系统状态.

2. 表达式序列:

Expression－list=URExpression*[RetExpression]

$M_{Expression-list}$:Expression－list→MCEnv→Exp

$M_{Expression-list}$[exprs] ρδs≜

let (oop,rv,rh,δ′,s′) =$M_{Expression}$[hd exprs]ρδs in

if len exprs=1 ∨ rh=true

then (oop,rv,rh,δ′,s′)

else $M_{Expression-list}$[tl exprs] ρδ′s′

注:若表达式序列中含有非局部返回的块,则该块的执行导致序列中剩余的表达式不再执行,且表达式序列的结果即为该块的结果,否则为最后一个表达式的结果.

3. 表达式:

Expression=RetExpression|URExpression

RetExpression::URExpression

$M_{Expression}$:Expression→MCEnv→Exp

$M_{Expression}$[mk－RetExpression(expr)]ρδs≜

let(oop,rv,rh,δ′,s′)=$M_{Expression}$[expr]ρδs in (oop,true,rh,δ′,s′)

注:表达式分为返回表达式和无返回表达式,任何无返回表达式可以构成返回表达式,返回表达式的作用就是置返回值标记为 true.

4. 方法:

Method－body::Args: Id*

Temps: Id_{-set}

Exprs: Expression_list

Home: Boolean

$M_{Method-body}$:Method－body→MCEnv→Method

$M_{Method-body}$[mk－Method－body(args,temps,exprs,home)]ρδ(rcvr,arglist,s) ≜

let δ′=mk－DEnv(rcvr,[args(i)|→ arglist(i)|i ∈inds args],

[id|→ Niloop|id ∈temps]) in

let(oop,rv,rh,δ″,s′)=$M_{Expression-list}$[exprs]ρδ′s in

if rv = true

then if rh=true

then if home=true

then (oop,false,false,δ,s′)

else (oop,rv,rh,δ,s′)

else (oop,false,rh,δ,s′)

else (rcvr,rv,rh,δ,s′)

注:方法体由参变量表、临时变量说明、表达式序列和非局部返回说明构成.

(1)方法体在执行时将产生自己的局部动态环境,而不改变外层的动态环境;

(2)若表达式序列的执行结果显示 rv=true,则说明某个返回表达式已被执行,其值即为返回值,否则接收器本身为省缺的返回值;

(3)若表达式序列的执行结果显示 rh=true, 则说明存在块的非局部返回,若该方法体

中 home=true，则返回到此结束，否则必须继续往外层返回.

4 结束语

针对文献[2]中存在的问题，我们在如下几个方面有较大的改进：

(1)提供了一个较为完整的形式模型和语义框架，充分体现了面向对象的哲学思想，保持了 Smalltalk 语言的高度统一性.

(2)所有系统元素(包括类和方法)都表示为对象，统一存放于对象存储器中，并定义有一组基于对象存储器的抽象函数.

(3)给出了 Smalltalk 程序层次的语法表示和语义描述.

(4)考虑了共享变量(类变量、pool 变量和全局变量)的处理，而共享变量正是 Smalltalk 语言的特色之一.

(5)以统一的方式处理类对象和实例对象，类方法和实例方法.

(6)对 Smalltalk 的方法和块作了完整的处理，解决了块的非局部返回和块访问所在方法的临时变量的语义描述问题.

本文给出的指称语义描述将有助于弄清 Smalltalk 的基本概念和机制，比较 OOP 范型和其它程序设计范型在本质上的异同. 对进程等并发特性的研究将是我们今后工作的重点.

致谢 本文所述工作得到了陈火旺教授的大力支持，作者在此深表谢意.

参考文献

1 Goldberg A, Robson D. Smalltalk—80: the language and its implementation. Addison—Wesley, 1983.

2 Wolczko M. Semantics of smalltalk—80. LNCS 276, 1987.

3 李舟军. Smalltalk—80 的形式语义描述[硕士论文]. 国防科技大学, 1991.

4 Bjorner D, Jones C B. Formal specification and software development. Prentice—Hall International, 1983.

5 Jones C B. System software development using VDM. Prentice—Hall International, 1986.

ON DENOTATIONAL SEMANTICS OF SMALLTALK—80

Li Zhoujun Wang Bingshan

(*Department of Computer Science, Changsha Institute of Technology, Changsha* 410073)

Abstract Smalltalk—80 is the archetypal object—oriented programming language and environment. This paper briefly introduces a formal model of Smalltalk—80. The static and dynamic denotational semantics of Smalltalk—80 are described through the formal model.

Key words Object—oriented programming, formal definition and theory, denotational semantics.