第6卷第7期
1995年7月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 6, No. 7
Jul. 1995

合成语言FOPL基于方程逻辑的语义*

梅 宏　　孙永强

（北京大学计算机科学与技术系，北京 100871）　（上海交通大学计算机科学与技术系，上海 200030）

摘要　程序设计语言FOPL是一种同时支持函数式程序设计风格和面向对象程序设计风格的合成语言.本文介绍了FOPL的类型思想，并讨论了表达式纯洁性判断规则、表达式附类型规则及表达式等价判断规则，这些规则描述了FOPL基于方程逻辑的语义.

关键词　数据类型，方程语义，合成语言，函数式程序设计，面向对象程序设计.

新型风格的程序设计语言的研究一直是计算机科学的一个重要研究领域.10多年来，研究者们为之付出了艰辛的努力.探讨其动因，一是我们对用于程序设计语言的设计、分析和实现的数学工具的理解已相当深刻；二是因为计算机科学的应用领域越来越广，如智能模拟、高级数据库、符号计算、并行体系结构、计算机辅助设计等领域需要适用的语言；三是由于大型复杂软件系统的生产率、可靠性等方面的需要.

逻辑式、函数式和面向对象程序设计语言是当今新型语言研究的三大主流.它们既具有优于其他语言的特点，又存在各自的弱点.为此，旨在综合各种语言优势的多范例合成语言的研究也成了一个重要的研究方向.研究者们在此领域取得了较大的进展，已得到了不少的重要成果.其中，又以逻辑式和函数式语言的合成成果为多，这是因为一方面对具有强有力的知识表达能力和有效的逻辑推理能力的语言的需求；另一方面，二者均有较好的数学基础，在语义级合成的可行性较大.相比而言，涉及面向对象语言的合成就有一定难度，其主要原因在于面向对象语言缺乏严密的数学基础，对面向对象语言的语义本质人们也未曾有一致认识.然而，面向对象语言模拟了人类认识问题的较高、较广的层次，其模块化能力和继承性易于知识的组织和管理，其强有力的存贮构造能力和消息传递能力更便于大型复杂系统的构造.因此，它和其他风格语言的合成的研究是非常有意义的.

我们的研究是为了在语义级上平滑地合成函数式语言和面向对象语言各自的优点，从理论上及实践上探讨合成的方法和合成特征的选取，探讨新型智能语言的设计原则及特点.为此，我们设计并实现了函数式面向对象程序设计语言FOPL[1].作为实验语言，我们未曾考虑I/O操作及实现效率等问题.目前，FOPL的实验系统环境已在Sun 386i工作站上用C

* 本文1993-03-26收到，1994-02-21定稿

本项研究受国家自然科学基金和863高科技计划资助.作者梅宏，1963年生，1994年10月博士后出站，副教授，主要研究领域为新型语言及其支撑环境，软件工程等.孙永强，1931年生，教授，博士导师，主要研究领域为新型语言，计算理论等.

本文通讯联系人：梅宏，北京 100871，北京大学计算机科学与技术系

语言编程实现,运行结果已达到了预期设计目标.

限于篇幅,本文不拟给出FOPL的详细介绍和讨论.本文只在简介FOPL的基础上,介绍FOPL的类型思想,进而讨论FOPL中表达式的纯洁性规则、附类型规则及等价判断规则.这些规则均是针对FOPL的源级语法的,以推导规则的形式给出,它们刻划了FOPL的语义特征,从而给出了FOPL基于方程逻辑的语义.

1 FOPL简介

FOPL语言的设计基于OOP≈ADTs+Inheritance这一合成框架[2],除要求FOPL仍能支持纯的函数式程序设计,保持函数语言的简单性、数学优美性和引用透明性等优点外[3],还要求具有高阶能力、多态性、Lazy计值等特征.对OOP特性的选取,要求能保持OOP的信息隐蔽、数据抽象、继承和动态约束等4个主要特征的前3个[4],放弃了对动态约束的要求.和纯函数式语言相比,FOPL不仅保持了纯函数式语言的主要优点,由于OOP机制的引入,它又具有较强的数据抽象封装能力,更有利于大型复杂系统的实现.继承的存在也使代码共享和复用的可能性大大增加,从而有利于软件生产率的提高.和传统OOP语言相比,FOPL又具有函数语言的特点,使之能更好地满足智能型语言的要求,更加彻底地脱离计算细节来看待计算.作为实验系统,FOPL的一个可能的应用领域是作为大型复杂系统的规范原型语言.

FOPL语言中提供了3类程序构件的定义机制:类型、类和函数.类型描述了对象的行为规范,而类则对应了该规范的实现,一个规范可以对应多个实现.函数定义是为了纯函数风格的保留而设置的,同时也为进一步考虑FOPL中引入并发机制提供可能性.这3种构件都是独立编辑、编译的.FOPL程序的主体是表达式,这是计值的对象.程序中也可能有构件定义和对象说明等可选项.FOPL语言还采用了不同于传统的继承概念,用子类型关系来描述对象间的行为规范继承,而继承仅表示语法意义上的代码共享和复用,从而分离了行为规范继承网层和实现继承网层,这在一定程度上解决了传统的继承和封装间的矛盾[5],同时也使语义更为清晰.为了追求更大程度的代码复用,方便可复用程序构件的管理和使用,一个集成的语言环境是需要的.该环境应保持类型库、类库及函数库,将库中构件的管理、编辑、编译及程序的编辑、编译和运行集成为一体,FOPL语言正是如此实现的.

在合成语言研究领域关于函数式语言和面向对象语言的合成已有许多令人瞩目的成果.如基于Lisp做语法结构扩展而得到的Commonloops, Oklisp, Commonobject等,虽然它们未曾考虑统一语义框架,但其实用性却得到了较大程度的保证,且对面向对象语言中的多路继承、封装等一些问题提出了不少处理方案.在语义级上合成比较成功的例子是FOOPS[6],这是在OBJ基础上开发的函数面向对象设计语言,它具有函数级模块和对象级模块,用具有状态的ADT,即抽象机作为类的语义,用隐蔽类型模拟状态,类序代数作为继承的语义基础,自反的类序代数给出了FOOPS的抽象操作语义.然而,高阶能力的缺乏,两种模块定义间的平衡,类仅有单实现等均是FOOPS的不足.文献[2]讨论了类型理论和面向对象程序设计间的关系,综述了一些基于类型理论的合成语言,如FUN便是体现OOP特征的代表,其他如DL,SOL等没有体现继承的思想.FUN是在一阶附类型λ演算基础上

加上二阶特性以模拟多态性和 OO 风格,将 ADT 表示为存在类型,记录表示对象,其中方法用函数记录分量模拟,子类型关系对应了类继承关系,只是其子类型关系决定于关于对象表示的实现决策而非对象在使用中的可视行为关系,子类型关系非用户指明而是内在决定的.这也达到了分离实现和行为网层的要求.目前,EATCS 的 ESPRIT 基础研究计划 3020 中也有针对函数式和 OOP 语言合成的研究计划[7],其合成亦是基于类型理论之上.

和上面工作相比,FOPL 的特点主要体现在以下几个方面:① FOPL 的设计思想是集大成的思想,将尽可能多地采用现有合成语言的优点并克服其不足;② FOPL 也是基于类型理论的,但采用的是 Russell 的类型思想,从而提供了对 ADT 的更灵活和自然的支持;③ FOPL 中分离了行为和实现网层,用子类型关系表达了行为继承关系,而用继承关系表示实现中的代码共享和复用;④ FOPL 类型系统中的子类型关系的引入突破了 Russell 对继承的不支持,同时这里的子类型关系不同于 FUN 中基于实现细节的子类型关系.从类型理论的角度看,基于行为规范的子类型关系更为适宜;⑤ 由于 FOPL 的类型概念,FOPL 的子类型关系能在更大范围内保持一致性,可描述范围比 FUN 要大;⑥ FOPL 支持参数化、多态性、多继承,具有高阶能力,还支持一个规范和多个实现间的对应;⑦ FOPL 有较好的语义基础,具有清晰的形式化的指称语义和操作语义.

2　类型思想

大多数传统语言中,类型概念是指一些具有某种共性的值的集合.FOPL 的类型思想则直接受益于程序设计语言 Russell 的类型思想[8,9]和 EATCS 的 ESPRIT 基础研究计划 3020 中考虑函数式程序设计和 OOP 合成子计划中的类型思想[7].Russell 中将值看成无类型的基本项,而类型则为操作的集合,该集合提供了值的一致解释.文献[7]将类型定义为对象的对外可视行为规范,即类型作为具有对外可视的固有共性的对象的集合.实质上,我们考虑一个对象时并不关心其内部表示和具体实现,只是关心其可被使用的方式.要使用一个对象,只需了解其对外可视行为.因此,类型应该给出对象的对外可视行为的描述.FOPL 语言将行为规范作为类型.下面给出一些非形式的定义.限于篇幅,形式化描述将另文给出.

定义 1. 类型是描述某类对象的行为规范的一组命名操作的集合.

一个类型定义实质上构成了一个 ADT 的规范,其中刻划的操作分成 3 类:描述对象结构,用于对象创建的构造子;用于对象属性选取的选择子;描述对象所能承受操作的一般方法.对一个值,这些操作一致地将它解释为属于某类型.

定义 2. 类是某类型的具体实现.

类即构成了一个 ADT 的实现.一个类型可对应多个类.

定义 3. 如类 C 实现类型 T,则类 C 的例化对象的类型是 T.

在 FOPL 中,一个类定义本身隐含了一个类型定义,一个类型定义也可以没有具体的实现类与之对应.对象则只能通过对类的例化而产生.

定义 4. TYPE 表示元类型,即所有类型的类型.

定义 5. CLASS 表示元类,是 TYPE 的实现.

定义 6. 子类型关系:$\sigma \leqslant \tau$ 是指行为规范 σ 强于行为规范 τ.

定义 7. 继承:C_1 继承 C_2 是指 C_1 共享 C_2 提供的代码.

子类型关系存在于类型之间,而继承关系既可存在于类之间,也可存在于类型之间.相对而言,类间继承更为重要和有用.

3 环境和记号

在推导系统中,推导总是基于一定环境进行的.本节首先给出环境的定义和有关记号.

定义 8. 环境△是一个公式集合,$\triangle=\theta\cup\varphi$,这里 θ 是附类型公式集合,行为 $x::s$,x 为标识符,s 为类型标记;φ 是等价公式的集合,行为 $e_1\equiv e_2$.当 θ 中含有附类型公式 $x::s$,我们称 θ 定义标识符 x,亦即△定义了 x.

一个正确的推导使用的环境应有如下性质:

性质 1. △是函数的,即它定义的每个标识符都是唯一定义的.

性质 2. △是封闭的,即它定义了自由出现在它的任意公式中的所有标识符.

直觉上,一个环境包含了在 FOPL 程序中说明的所有标识符的标记,即类型说明.这样性质 1 保证了同一标识符不能在程序中作不同类型说明,性质 2 保证了不会有标识符未被说明而为程序所用.FOPL 中有对象标识符和值标识的区分,分别用关键字 Var 和 Val 指明.Var 用于对象标识符说明,Val 用于值标识说明.

给定△和标识符集 $x_1\cdots,x_n$ 我们可构造:

$$\triangle/x_1,\cdots,x_n \overset{\text{def}}{=} \bigcup(\triangle'\mid\triangle'\leqslant\triangle\wedge\triangle'\text{是封闭的}\wedge\triangle'\text{不定义}x_1,\cdots,x_n\text{之一})$$

这是△的未定义 $x_1,\cdots,x_n$ 的最大封闭子集.这个构造删去了△中 $x_1,\cdots,x_n$ 的定义和任何依赖它们的公式.进而我们可定义:

$$(\triangle,x_1::s_1,\cdots,x_n::s_n)\overset{\text{def}}{=}\triangle/x_1,\cdots,x_n\cup\{x_1::s_1,\cdots,x_n::s_n\}$$

类似也可定义$\triangle\mid x_1,\cdots,x_n$,称为△到 $x_1,\cdots,x_n$ 的变量限制.

$$\triangle\mid x_1,\cdots,x_n\overset{\text{def}}{=}\bigcup(\triangle'\mid\triangle'\leqslant\triangle\wedge\triangle'\text{是封闭的}\wedge(y::\text{Var}\,T)\in\triangle'\Rightarrow\exists\, i.\,y=x_i)$$

这个限制使得到△中由 Var 说明的对象标识仅剩下 $x_1,\cdots,x_n$.最后可定义:

$\triangle_{Val}\overset{\text{def}}{=}\triangle\mid$,即△中所有 Var 说明的标识符被删去.

上面是将 θ 看成公式集,由于其函数性,亦可将 θ 看成一将标识符映射到类型标记的部分函数,对 $x::s\in\triangle$,亦可表示成$\triangle(x)=s$.

下面再给出几个记号的解释:$FV(e)$—产生 e 中自由标识符的集合;$dom(\triangle)$—表示△的定义域,即被定义标识符的集合;$e[e_1/x]$—表示用 e_1 替代 e 中所有 x 的自由出现.

4 纯洁性判断

表达式是 FOPL 程序的主体.本文后几节正是通过一系列的推导规则来刻划表达式的语义.在此,有必要先给出 FOPL 中表达式和类型表达式的描述.

⟨*exp*⟩::=⟨*CONST*⟩;常量,形为,⟨*cid*⟩$⟨*id*⟩

|(⟨*exp*⟩),括号表达式

|⟨*exp*⟩::⟨*texp*⟩;附类型表达式

|⟨*id*⟩;标识符

|⟨*exp*⟩⟨*exp*⟩;函数作用

|⟨*id def*⟩;⟨*exp*⟩;带标识定义的表达式

|⟨*cid*⟩.*new*[⟨*exp group*⟩];对象创建

|⟨*exp*⟩.⟨*fid*⟩[*exp group*⟩];消息传递表达式

|⟨*exp*⟩,⟨*bexp*⟩,…,⟨*exp*⟩,⟨*bexp*⟩;条件表达式

|⟨*texp*⟩;类型表达式

⟨*id def*⟩::=⟨*id*⟩←⟨*exp*⟩;对象标识赋值

|⟨*id*⟩=⟨*exp*⟩　;值标识约束

其中⟨*cid*⟩为类名,⟨*fid*⟩为方法名,⟨*bexp*⟩表示布尔表达式.

⟨*texp*⟩::=⟨*tCONST*⟩　;已定义类型

|⟨*tid*⟩　;类型变量

|*TYPE*　;元类型

|⟨*texp*⟩→⟨*texp*⟩　;函数类型

纯洁性判断是形为 *pure e* 的公式,它断言表达式 *e* 独立于任何变量的值,没有副作用,也不会产生依赖当前状态的值.如我们可推导出 *pure e*,则称 *e* 是纯结的,亦即是透明的,不管多少次不同计值,其计值结果总是一样.

P_0(常量纯结性)　$\triangle \vdash pure\ \langle CONST\rangle$

P_1(标识符纯洁性)　$\dfrac{\triangle \vdash x::s}{\triangle \vdash pure\ x}$

P_2(值纯洁性)　$\dfrac{\triangle \mathrm{val} \vdash e::T}{\triangle \vdash pure\ e}$

P_3(替代纯洁性)　$\dfrac{\triangle \vdash e_1 \equiv e_2::T, \triangle \vdash pure\ e_1}{\triangle \vdash pure\ e_2}$

P_4(括号无关性)　$\dfrac{\triangle \vdash pure\ e}{\triangle \vdash pure\ (e)}$

P_5(附类型无关性)　$\dfrac{\triangle \vdash pure\ e, \triangle \vdash e::T}{\triangle \vdash pure(e::T)}$

P_6(标识符值约束)　$\dfrac{\triangle \vdash pure\ e_1, pure\ e_2}{\triangle \vdash pure\ x = e_1; e_2}$

P_6'(标识符赋值)　$\dfrac{\triangle \vdash pure\ e_1, pure\ e_2}{\triangle \vdash pure\ x \leftarrow e_1; e_2}$

P_7(函数作用)　$\triangle \vdash e_1::T_1 \to T_2, e_2::T_1$

$$\frac{\triangle \vdash pure\ e_1, pure\ e_2}{\triangle \vdash pure\ e_1\ e_2}$$

P_8(函数纯洁性)　$\dfrac{\triangle \vdash f::T_1 \to T_2}{\triangle \vdash pure\ f}$

P_9(对象创建)　$\triangle \vdash pure\langle cid\rangle.new[e_1, \cdots, e_n]$

P_{10}(消息传递)　$\triangle \vdash e::T, e_1::T_1, \cdots e_n::T_n$

$$\triangle \vdash f::T \to T_1 \to \cdots T_n \to T'$$

$$\frac{\triangle \vdash pure\ e, pure\ e_1, \cdots, pure\ e_n}{\triangle \vdash pure\ e.f[e_1, \cdots e_n]}$$

P_{11}(条件表达式)　$\triangle \vdash e_1::T_1, \cdots, e_n::T_n, \triangle \vdash pure\ e_1, \cdots, pure\ e_n$

$$\frac{\triangle \vdash be_1::BOOL,\cdots,be_n::BOOL \quad \triangle \vdash pure\ be_1,\cdots,pure\ be_n}{\triangle \vdash pure\ e_1,be_1,\cdots,e_n,be_n}$$

P_{12}(类型表达式纯洁性) $\triangle \vdash pure\langle texp\rangle$

在FOPL中,纯洁表达式的应用将导致纯函数风格的程序.

5 附类型规则

本节给出FOPL表达式的附类型规则,FOPL程序的类型检查将以这些规则为基础.

T_1(前提1) $\triangle \vdash x::TYPE$, if $\triangle(x)=TYPE$

T_2(前提2) $\dfrac{\triangle \vdash T::TYPE}{\triangle \vdash x::T}$ if $\triangle(x)=Var\ T \vee \triangle(x)=Val\ T$

T_3(前提3) $\dfrac{\triangle \vdash T::TYPE}{\triangle \vdash const::T}$ if $const=T\ \$\ \langle id\rangle$

T_4(类型构造) $\dfrac{\triangle \vdash A::TYPE,B::TYPE}{\triangle \vdash A\rightarrow B::TYPE}$

T_5(括号表达式) $\dfrac{\triangle \vdash e::T}{\triangle \vdash (e)::T}$

T_6(附类型表达式) $\dfrac{\triangle \vdash e::T}{\triangle \vdash (e::T)::T}$

T_7(函数作用) $\dfrac{\triangle \vdash e_1::T_1\rightarrow T_2,e_2::T_1}{\triangle \vdash e_1e_2::T_2[e_2/x]}$ $x\in FV(T_1)$

这里当e_2为类型时,T_1必形为$TYPE[x]$,这是我们在FOPL原级语法上引入的语法标记,表示在此处需一类型作为参数传递.变量x为类型变量,它和$TYPE$的联用表明它在T_2中将代表本位置的类型参数,类型参数对x的替代表示了参数和结果类型间的依赖关系.这种依赖关系可表达在多态二阶λ演算中,也可用依赖类型表示.但在FOPL的源级语法中,只能用此方式来表达依赖性.本质上,$TYPE[x]$就等同于$TYPE$,x仅是语法标记.$FV(TYPE[x])=\{x\}$.

T_8(标识符赋值) $\dfrac{\triangle \vdash e::T,\ \triangle,\langle id\rangle::\mathrm{Var}\ T \vdash e'::T'}{\triangle,\langle id\rangle::\mathrm{Var}\ T \vdash \langle id\rangle\leftarrow e;e'::T'}$

T'_8(标识符值约束) $\dfrac{\triangle \vdash e::T,\ \triangle,\langle id\rangle::\mathrm{Val}\ T \vdash e'::T'}{\triangle,\langle id\rangle::\mathrm{Val}\ T \vdash \langle id\rangle=e;e'::T'}$

T_9(对象创建) $\dfrac{\triangle \vdash C::TYPE}{\triangle \vdash C.new::C}$, $C\in CLASS$

这里C是某已存在类名,且其中有零元构造子.

T'_9(对象创建) $\dfrac{\triangle \vdash C::TYPE,\triangle \vdash e_1::T_1,\cdots,e_n::T_n \quad \triangle \vdash C\ \$\ c::T_1\rightarrow\cdots\rightarrow T_n\rightarrow C}{\triangle \vdash C.new[e_1,\cdots,e_n]::C}$, $C\in CLASS$

这里$C\ \$\ c$为$C$中$n$元构造子.

T_{10}(消息传递) $\dfrac{\triangle \vdash e::T,f::T_1\rightarrow\cdots\rightarrow T_n\rightarrow T' \quad \triangle \vdash e_1::T_1,\cdots,e_n::T_n}{\triangle \vdash e.f[e_1,\cdots,e_n]::T'[e_1/x_1]\cdots[e_n/x_n]}$, $X_i\in FV(T_i)$

T_{11}(条件表达式) $\dfrac{\triangle \vdash e_1::T,\cdots,e_n::T,be_i::BOOL}{\triangle \vdash e_1,be_1,\cdots,e_n,be_n,::T}$

由于语言中子类型关系的存在,因此,下列关于子类型的规则也是需要的.

T_{12}(自反性)　$\dfrac{\triangle \vdash T::TYPE}{\triangle \vdash T \leqslant T}$

T_{13}(传递性)　$\dfrac{\triangle \vdash T_1 \leqslant T_2, T_2 \leqslant T_3}{\triangle \vdash T_1 \leqslant T_3}$

T_{14}(函数子类型关系)　$\dfrac{\triangle \vdash T_1 \leqslant T'_1, T'_2, \leqslant T_2}{\triangle \vdash T'_1 \rightarrow T'_2 \leqslant T_1 \rightarrow T_2}$

即在参数部位具有反变性,而在结果部位具正变性.

T_{15}(类和类型)　$\triangle \vdash C \leqslant Typeof(C)$,如 C 是类名

当 C 为类名时,它同时隐含了一个类型,故 C 可作为类型名使用. *Typeof* 是 FOPL 中关键字,指出被实现的类型名.这里,*Typeof* 取 C 所实现的类型名,如无,则为 C 自己.

T_{16}(超类型说明)　$\triangle \vdash T \leqslant Super(T)$

Super 亦是 FOPL 中关键字,用于指明被定义类型的超类型.虽然 FOPL 中子类型关系是用户指明的,但是必须由编译器对此关系进行验证后才可被接受,此关系的验证是语义检查的重要内容之一.

T_{17}(包含多态性)　$\dfrac{\triangle \vdash e::T_1, T_1 \leqslant T_2}{\triangle \vdash e::T_2}$

此项规则表明,在需某类型对象出现的地方,其子类型的对象也是适用的.包含多态性体现于一个函数可作用于某类型及该类型子类型的所有对象.

6　等价判断

本节给出 FOPL 中表达式间的等价关系描述,所谓方程逻辑正是针对等价关系而言.

E_1(自反性)　$\dfrac{\triangle \vdash a::T}{\triangle \vdash a \equiv a}$　　E_2(对称性)　$\dfrac{\triangle \vdash a \equiv b}{\triangle \vdash b \equiv a}$

E_3(传递性)　$\dfrac{\triangle \vdash a \equiv b, \triangle \vdash b \equiv c}{\triangle \vdash a \equiv c}$　　E_4(括号无类性)　$\triangle \vdash (e) \equiv e$

E_5(删去类型信息)　$\triangle \vdash e::T \equiv e$　　E_6(赋值无关性)　$\dfrac{\triangle \vdash e::T, \triangle \vdash pure\ e}{\triangle \vdash e \equiv x \leftarrow e; e}, x \notin FV(e)$

E_7(顺序无关性)　$\dfrac{\triangle \vdash e_1::T_1, e_2::T_2, e_3::T_3, \triangle \vdash pure\ e_1, pure\ e_2}{\triangle \vdash x_1 \leftarrow e_1; x_2 \leftarrow e_2; e_3 \equiv x_2 \leftarrow e_2; x_1 \leftarrow e_1; e_3}$

E_8(赋值删去)　$\dfrac{\triangle \vdash x \leftarrow e_1; e::T, \triangle / x \vdash e::T}{\triangle \vdash x \leftarrow e_1; e \equiv e}$

E'_8(约束删去)　$\dfrac{\triangle \vdash x = e_1; e::T, \triangle / x \vdash e::T}{\triangle \vdash x = e_1; e \equiv e}$

E_9(函数作用)　$\dfrac{\triangle \vdash e_1::T_1 \rightarrow T_2, e_2::T_1, \triangle \vdash pure\ e_2}{\triangle \vdash e_1 e_2 \equiv x = e_2; e_1 x}, \quad x \notin FV(e_1)$

E_{10}(冗余约束)　$\dfrac{\triangle \vdash e::T}{\triangle \vdash x = e; x \equiv e}, \quad x \notin FV(e)$

E'_{10}(冗余赋值)　$\dfrac{\triangle \vdash e::T}{\triangle \vdash x \leftarrow e; x \equiv e}, \quad x \notin FV(e)$

E_{11}(约束替代)　$\triangle \vdash x = e_1; e::T, e_1::T_1$

$\triangle, x::T_1 \vdash x = e_1; g \equiv x = e_1; h$

$\triangle, x::T_1 \vdash pure\ g, pure\ h$

$$\frac{\triangle, x::T_1, g\equiv h \vdash e\equiv e'}{\triangle \vdash x=e_1; e\equiv x=e_1; e'}$$

E'_{11}(赋值替代)
$$\triangle \vdash x\leftarrow e_1; e::T, e_1::T_1$$
$$\triangle, x::T_1 \vdash x\leftarrow e_1; g\equiv x\leftarrow e_1; h$$
$$\triangle, x::T_1, \vdash pure\ g,\ pure\ h$$
$$\frac{\triangle, x::T_1, g\equiv h \vdash e\equiv e'}{\triangle \vdash x\leftarrow e_1; e\equiv x\leftarrow e_1; e'}$$

如果没有 g,h 对存在,则可有下面规则:

E_{12}(约束等价)
$$\triangle \vdash x=e_1; e::T, e_1::T_1$$
$$\frac{\triangle, x::T_1 \vdash e\equiv e'}{\triangle \vdash x=e_1; e\equiv x=e_1; e'}$$

E'_{12}(赋值等价)
$$\triangle \vdash x\leftarrow e_1; e::T, e_1::T_1$$
$$\frac{\triangle, x::T_1 \vdash e\equiv e'}{\triangle \vdash x\leftarrow e_1; e\equiv x\leftarrow e_1; e'}$$

通常,$g\equiv h$ 可用为描述约束值的谓词,在证明 e, e' 的等价中,我们可用有关约束值的知识. 如令 g 为 x,h 为 e_1,则有如下两规则:

E_{13}(β 变换)
$$\frac{\triangle \vdash x=e_1; e::T, \triangle \vdash pure\ e_1}{\triangle \vdash x=e_1; e\equiv e[e_1/x]}$$

E'_{13}(β 变换)
$$\frac{\triangle \vdash x\leftarrow e_1; e::T, \triangle \vdash pure\ e_1}{\triangle \vdash x\leftarrow e_1; e\equiv e[e_1/x]}$$

E_{14}(参数计值)
$$\frac{\triangle \vdash e\ e_1\cdots e_n::T}{\triangle \vdash ee_1\cdots e_n\equiv y\leftarrow e_1; \cdots; y_n\leftarrow e_n; ey_1\cdots y_n},\quad y_i\notin FV(e)$$

此规则表示当对象作为函数参数时采用按值调用方式. 当值作为参数时,仍采用按需调用方式,故上规则中将←换为=是不成立的.

E_{15}(类的例化)
$$\frac{\triangle \vdash C.new::T, C\ \$\ c::T}{\triangle \vdash C.new\equiv C\ \$\ c}$$

E'_{15}(类的例化)
$$\triangle \vdash C.new[e_1,\cdots,e_n]::T$$
$$\triangle \vdash C\ \$\ c::T_1\rightarrow\cdots\rightarrow T_n\rightarrow T$$
$$\frac{\triangle \vdash e_1::T_1,\cdots,e_n::T_n}{\triangle C.new[e_1,\cdots,e_n]\equiv C\ \$\ ce_1\cdots e_n}$$

这两个规则反映了 *new* 消息和构造子间的关系.

E_{16}(消息传递)
$$\frac{\triangle \vdash e::T, T\ \$\ f::T\rightarrow T'}{\triangle \vdash e.f\equiv T\ \$\ fe}$$

E'_{16}(消息传递)
$$\frac{\triangle \vdash e::T, T\ \$\ f::T\rightarrow T_1\rightarrow T', e_1::T_1}{\triangle \vdash e.f[e_1]\equiv T\ \$\ fe\ e_1}$$

这两个规则表明了消息传递和函数作用间的内在关系.

E_{17}(条件表达式)
$$\frac{\triangle \vdash e_1, BOOL\ \$\ true,\cdots,e_n,be_n::T}{\triangle \vdash e_1, BOOL\ \$\ true,\cdots,e_n,be_n\equiv e_1}$$

E'_{17}(条件表达式)
$$\frac{\triangle \vdash e_1, BOOL\ \$\ false,\cdots,e_n,\ be_n::T}{\triangle \vdash e_1, BOOL\ \$\ false,\cdots,e_n,be_n\equiv e_2,be_2,\cdots,e_n,be_n}$$

E_{18}(条件等价)
$$\frac{\triangle \vdash e,be_1,\cdots,e,be_n::T}{\triangle \vdash e,be_1,\cdots,e,be_n\equiv e}$$

E_{19}(条件替代)
$$\triangle \vdash e_1,be_1,\cdots,e_n,be_n::T, \triangle \vdash pure\ be_1,\cdots,pure\ be_n$$

$$\frac{\triangle, be_1 \equiv BOOl\ \$\ true \vdash e_1 \equiv e'_1, \cdots \quad \triangle, be_n \equiv BOOL\ \$\ true \vdash e_n \equiv e'_n}{\triangle \vdash e_1, be_1, \cdots, e_n, be_n \equiv e', be_1, \cdots, e'_n, be_n}$$

最后,我们用下面命题作为本节的结束.

命题1. 等价关系保持类型系统,即,如$\triangle \vdash a :: T$,且$\triangle \vdash a \equiv b$,则有$\triangle \vdash b :: T$.

7 结束语

本文介绍了 FOPL 的类型思想,描述了关于表达式的一系列规则.由这些规则,我们可以更好地理解 FOPL 的语义.在实现 FOPL 时,这些规则是类型检查的基础.另外,我们可以基于这些规则构建 FOPL 的证明系统,这将是更进一步的工作.本文给出的 FOPL 的类型推导系统只能在一定的前提下工作,即需要程序员提供足够的显式类型信息,没有对 FOPL 程序进行自动类型推导的形式基础,程序员必须自己去判断什么地方需要显式的类型标记.为使语言更简洁、易用,研究"什么是必要的类型信息"是未来工作的一个方向.

参考文献

1 梅宏.函数式面向对象程序设计语言 FOPL——设计和实现[博士论文].上海交通大学,1992.
2 Danforth S, Tomlinson C. Type theories and object—oriented programming. ACM Computing Surveys, 1988,**20**(1).
3 Backus J. Can programming be liberated from the von neumam style? a functional style and its algebra of programs. Communication of ACM, 1978,**21**(8).
4 Pascoe G A. Elements of object—oriented programming. BYTE, 1986.
5 Snyder A. Encapsulation and inheritance in object—oriented programming languages. ACM OOPSLA'86, 1986.
6 Goguen J, Meseguer J. Unifying functional, object—oriented and relational programming with logical semantics. SRI INTERNATIONAL, 1987.
7 EATCS BULLETIN, N0.40, 1990.
8 Donahue J, Demers A. Data types are values. ACM Transactions on Programming Languages and Systems, 1985, 7(3).
9 Demers A, Donahue J. Making variables abstract, an equational theory for russell. ACM POPL'83, 1983.

EQUATIONAL SEMANTICS OF HYBRID LANGUAGE FOPL

Mei Hong

(*Department of Computer Science, Beijing University, Beijing* 100871)

Sun Yongqiang

(*Department of Computer Science, Shanghai Jiaotong University, Shanghai* 200030)

Abstract Programming language FOPL is a hybrid language which supports functional programming style and object—oriented programming style. In this paper, the type concepts of FOPL are presented. Also, the rules for purity judgement of expressions, typing expressions and equivalence judgement of expressions are discussed. These rules describe the semantics of FOPL on equational logic.

Key words Data type, equational semantics, hybrid language, functional programming, object—oriented programming.