Garment 中多态类型的 Ideal 模型*

郑红军[1]　张乃孝[2]

（北京大学，北京 100871）

摘要　本文从 Ideal 的基本概念出发，研究了 Ideal 作为类型的语义模型所具有的性质；在类型的 Ideal 模型下，讨论了 Garment 中参数化多态类型和约束多态类型的语义；在此基础上，证明了 Garment 中类型规则的语义可靠性.

关键词　Ideal　多态类型　类型系统　语义　Garment

1　背景

通过研究软件与程序的本质差别以及一类软件(专用软件)与程序设计语言的内在联系，我们提出了一种系统的软件开发方法 — 面向模型的变换型软件开发方法[1]. 该方法发展了 VDM 的模型概念；根据面向对象的原理，发展了数据抽象的概念，将其升华到语言抽象的层次，由此提出了一种语言的抽象与封装机制 — Garment，并将抽象数据类型作为语言抽象的一个基本单位. 类型定义是 Garment 的核心，而且在 Garment 所抽象出来的语言中也占核心地位. 要定义类型，首先要清楚类型的含义，这样才能知道定义出的是什么. 在使用 Garment 所抽象出的语言或一般的程序设计语言进行程序设计时，程序员也应清楚这些语言所提供的类型的含义. 因此，类型的语义不仅是 Garment 的语义基础，而且也是理解和使用一般程序设计语言的语义基础.

除了一般的单态类型外，Garment 不仅能定义出参数化多态类型，如：求序列长度的函数 length:$\forall\alpha.\text{seq}[\alpha]\to$ Int，其中，seq[α]表示元素类型为α的所有序列，而且还可以定义出约束多态类型，如：用于比较序列的函数类型即为约束多态类型 $\le:\forall\alpha.(\le:\alpha\times\alpha\to\text{Bool}).\text{seq}[\alpha]\times\text{seq}[\alpha]\to\text{Bool}$，意为满足约束 $\le:\alpha\times\alpha\to\text{Bool}$ 的所有函数 $\text{seq}[\alpha]\times\text{seq}[\alpha]\to\text{Bool}$，其中，$\le:\alpha\times\alpha\to\text{Bool}$ 意为标识符‘≤’具有类型 $\alpha\times\alpha\to\text{Bool}$. 具体地，只有在序列的元素类型上定义了比较操作 $\le:\alpha\times\alpha\to\text{Bool}$，才可应用序列比较函数 $\le:\text{seq}[\alpha]\times\text{seq}[\alpha]\to\text{Bool}$，符号‘≤’在此是重载的. 约束与全称量化的结合弥补了参数化多态表示的类型范围过宽，而简单的重载表示的类型范围又可能过窄的不足[2].

为了研究多态类型的语义，许多研究者都曾为类型设计了语义模型，如：Retract 模型[3]，Set 模型[4]，Ideal 模型[5][6]等，这些模型都从不同的角度给出了类型的指称. 这些研究工作的共同特点是：都将二阶λ-演算中自应用(self-application)

* 本研究工作得到国家自然科学基金(编号 69683006)的资助. 联系人张乃孝(naixiao@pku.edu.cn).

[1] 郑红军，北京大学计算机科学与技术系 博士生，主要研究方向：软件方法学，程序设计语言.

[2] 张乃孝，北京大学数学学院 信息科学系 教授，主要研究方向：软件方法学，程序设计语言.

函数的多态类型(即: 函数 *xx* 的类型)作为其语义研究的重点, 并取得了许多研究结果, 但未发现有人研究过约束多态类型的语义.

本文从 Ideal 的基本概念出发, 研究 Ideal 作为类型的语义模型所具有的性质, 然后以[7]中的 *Exp* 语言[3]为对象, 在类型的 Ideal 模型下, 讨论 Garment 中多态类型的语义(包括参数化多态和约束多态), 以及类型规则的语义可靠性.

2 Ideal

直观地看, 可以将一般的类型视为一组值的集合(简称为值集合), 但并不是所有的值集合都是合法的类型. Reynolds 在文献[8]中证明了二阶λ-演算中自应用多态函数(如 *xx*)的类型无集合论模型, 在此我们不讨论这类多态类型, 因为在 Garment 中无自应用函数.

从类型的组织方式看, 同一类型的值通常具有相同或相似的结构, 如: 序对, 函数, 整数等, 若将所有的值集合都视为类型, 显然是不合理的. 在此, 我们将 Ideal 作为类型的指称, 即: 一个值集合是一个类型, 当且仅当该集合是一个 Ideal.

定义 1 设(D, ≤)是一个完全偏序集(cpo)[4], D 上的一个子集 I 是 Ideal, 当且仅当

(i) $I \neq \phi$;

(ii) $\forall y \in I, \forall x \in D$, 如果 $x \le y$, 则 $x \in I$, ≤是 D 上的偏序关系;

(iii) 对于 I 中的任一ω-链[5] $X=\{x_i\}$, X的上确界存在且 $lub(X) \in I$.

D 一般是程序设计语言中所有可计算值在平坦序下构成的 cpo. 有时也把 D 中的最小元记为$\perp_D$, D上的偏序记为$\le_D$, 本文在一般情况下不加以区分. 将 D 上所有理想构成的集合记为 $p(D)$. 由于 D 中的最小元⊥可以是任意类型, 所以任何一个 Ideal 中均应含有⊥, 因此定义 1 中的条件(i)要求 $I \neq \phi$ 是必要的; 条件(ii)是要求一个 Ideal(类型) I 对 D 上的偏序关系≤向下封闭; 条件(iii)要求对 I 中ω-链的上确界封闭, 由于此条件, 使得 Ideal 本身在 D 上的偏序≤下也构成一个 cpo. Retract 模型[3]只满足定义 1 中的条件(i)和条件(iii), 而不满足条件(ii).

定理 1 $p(D)$在集合包含关系⊆下, 构成一个 cpo.

证明: 显然, 集合包含关系⊆是一个偏序关系. 任取 $p(D)$中的一个链X, 只需证链X有上确界 lub(X), 且 $lub(X) \in p(D)$.

(1) 以∪X表示链X中所有 Ideal 的广义并. 易证: $lub(X)=\cup X$, 即∪X是链X的上确界.

(2) 欲证$\cup X \in p(D)$, 即是证∪X亦是 D 上的 Ideal. 根据 Ideal 的定义:

<i> ∪X显然非空, 至少$\perp \in \cup X$.

---

[3] 我们将 *Exp* 语言作为 Garment 抽象出语言的模型.

[4] 一个完全偏序集(cpo)是一个序对(D, ≤), 其中 D 是一个偏序集, ≤是 D 上的偏序关系, 并且满足:
(i) 在(D, ≤)中存在一个最小元⊥; (ii) D 中的每个链X都有上确界, 记为 lub(X).
设$X \subseteq D$, 若对任意 $a, b \in X$都有 $a \le b$ 或 $b \le a$, 则称X为 D 的链.

[5] 设$X \subseteq D$, X中元素的序列$\{x_i\}$称为是关于 D 上偏序≤的一个ω-链, 若有 $x_i \le x_{i+1}$, (i = 1, 2, 3, . . .).

<ii> **任取** $y\in\bigcup X$, **必存在一个** $I\in X$, **使得** $y\in I$. **因** I **是** Ideal, **所以**, **若有** $x\in D$, **且** $x\le y$, **则有** $x\in I$. **因** $I\subseteq\bigcup X$, **故有** $x\in\bigcup X$.

<iii> **任取** $\bigcup X$ **中的一个** ω-**链** $Z=\{z_i\}$, **则必存在一个** $I\in X$, **使得** $z_n\in I$. **因** $z_1\le z_2\le \ldots \le z_n$, **若** $z_n\in I$, **根据定义** 1 **中的条件**(ii), **必有** $z_1, z_2, \ldots, z_{n-1}\in I$, **所以**Z**亦是** I **中的一个**ω-**链**, **由条件**(iii), $lub(Z)\in I$, **故** $lub(Z)\in\bigcup X$.

由上可得, $\bigcup X$**是** D **上的** Ideal, **故** $\bigcup X\in p(D)$.

因此, $p(D)$**在集合包含关系**$\subseteq$**下**, **构成一个** cpo.

在定义 $\forall I, J\in p(D).\ I\subseteq J \Leftrightarrow I\cap J=I \Leftrightarrow I\cup J=J$ **下**, **由文献**[9]**的定义** 1.2.2 **易证**:

定理 2 $(p(D), \subseteq)$**在集合的交**, **并运算**$(\cap, \cup)$**下构成一个格**.

推论 $(p(D), \subseteq)$**封闭于**$\cap, \cup$**运算**, **即对** $\forall I, J\in p(D)$, $I\cap J\in p(D)$**并且** $I\cup J\in p(D)$.

定义 2 **设** $I, J\in p(D)$, **定义**: $I\otimes J =_{df} \{(a,b) \mid a\in I, b\in J\}\cup\{\perp\}$

$I\oplus J =_{df} \{(i,d) \mid i=0 \wedge d\in I \text{ or } i=1 \wedge d\in J\}\cup\{\perp\}$

$I\mapsto J =_{df} \{f\in [D\to D] \mid f(I)\subseteq J\}\cup\{\perp\}$

定义 3 **设**$(a_1, b_1), (a_2, b_2)\in I\otimes J$, **定义**: $(a_1, b_1)\le(a_2, b_2) =_{df} a_1\le_I a_2 \wedge b_1\le_J b_2$, $\perp$**是** $I\otimes J$ **中的最小元**; **设**$(i,d_1), (j,d_2)\in I\oplus J$, **定义**: $(i,d_1)\le(j,d_2) =_{df} i=j \wedge$ if $i=0$ then $d_1\le_I d_2$ else $d_1\le_J d_2$, $\perp$**是** $I\oplus J$ **中的最小元**; **设** $f, g\in I\mapsto J$, **定义**: $f\le g =_{df} \forall d\in I.\ f(d)\le_J g(d)$, $\perp$ **是** $I\mapsto J$ **中的最小元**.

定理 3 **设** D **是由** '+', '×', '→'**构造的** cpo, **即** $D=[D+D]+[D\times D]+[D\to D]$, $p(D)$**封闭于** Ideal **构造符**: $\otimes, \oplus, \mapsto$.

证明: **因篇幅所限**, **在此我们只给出关于**$\mapsto$**构造符封闭的证明**, **即需证**: **若** $I, J\in p(D)$, **则** $I\mapsto J\in p(D)$, **根据定义** 1:

<i> $\perp\in I\mapsto J$, **故** $I\mapsto J\ne\phi$.

<ii> **取** $g\in I\mapsto J$, **若有** $f\in[D\to D]$**满足** $f\le g$, **则由定义** 3 **有**$\forall d\in I.\ f(d)\le_J g(d)$, **因** $g(d)\in J$ **且** J **是** Ideal, **所以** $f(d)\in J$, **由定义** 2: $f\in I\mapsto J$.

<iii> **任取** $I\mapsto J$ **中的一个**ω-**链**$X=\{f_i\}$**和** $d\in I$, **构造集合**$X_d=\{f_i(d) \mid f_i\in X\}$, **则**$X_d$**是** J **中的一个**ω-**链**. **事实上**, **由于**X**是**ω-**链**, **所以有** $f_i\le f_{i+1}$, **故由定义** 3, $f_i(d)\le_J f_{i+1}(d)$, **因此**X_d**是** J **中的一个**ω-**链**. **由于** J **是** Ideal, **所以** $lub(X_d)\in J$.

构造函数 $f_0: I\mapsto J$, **满足** $f_0(d)=lub\{f_i(d) \mid f_i\in X\}=lub(X_d)$, **则** $lub(X)=f_0$. **事实上**, **由** f_0**的构造过程可以看出**: $f_0\in I\mapsto J$. **任取** $f\in X$**都满足**$\forall d\in I.\ f(d)\in X_d$, **因** $f(d)\le_J lub(Xd)$, **即**: $f(d)\le_J f_0(d)$, **由定义** 3, **即** $f\le f_0$, **所以**, f_0**是**X**的一个上界**.

若 $g\in I\mapsto J$ **也是**X**的一个上界**, **则**$\forall f\in X.f\le g$. **这样**, **任取** $f\in X$, **都满足** $\forall d\in I.f(d)\le_J g(d)$, **因此**, g(d)**是**$X_d$**的一个上界**, **故有** $lub(X_d)\le_J g(d)$, **即**: $f_0(d)\le_J g(d)$. **由定义** 3, $f_0\le g$, **所以** $lub(X)=f_0$.

上述<i>, <ii>, <iii>**说明** $I\mapsto J\in p(D)$, **即**: $I\mapsto J$ **亦是** Ideal. **因此**, $p(D)$**封闭于构造符**$\mapsto$.

同理可证 $p(D)$**封闭于构造符**$\otimes$**和**$\oplus$.

定理 4　**假设** F: p(D)→ p(D)**是将一个** Ideal **映射到另一个** Ideal **的*连续*函数**. **若** c=fix(F), **则** c **也是** Ideal. **其中** fix(F)**表示** F **的最小不动点**.

易见, c **亦是** Retract[3]. **定理** 4 **是** Kleene **第一递归定理**[9][10]**的变形**, **其证明方法与** Kleene **定理类似**. **此定理说明了递归类型的** Ideal **模型的存在性**. **如**: **对于类型定义** *tree*=*int*+(*tree*×*tree*), **若取** F=λx.Int⊕(x⊗x), **可以证明** F **是保上确界的连续函数**, **因此保证了所定义的** *tree* **的确是一个类型**.

一个程序设计语言所能表示的类型是 D **上所有** Ideal **的一个子集**, D **是该语言可计算值的集合**. **若欲完整地描述一个语言的类型系统**, **通常需给出一个类型表达式语言**, **类型表达式到** Ideal **的一个映射以及类型规则**. **不同的语言可以具有不同的类型系统**.

3　*Exp* 语言及其语义

在 Ideal **下**, "**一个值** *v* **具有类型**σ "**就可以解释为**" *v* **属于与**σ**对应的** Ideal ". **在单态类型系统中**, **一个值只属于一个类型**(**除了** D **的最小元**⊥, **根据** Ideal **的定义**, **它属于所有的类型**); **在多态类型系统中**, **一个值可以属于多个类型**, **因为** D **上的** Ideal **可能重叠**.

下面以 *Exp* **语言**[7]**为研究对象**, **以** Ideal **作为类型的语义模型**, **讨论** Garment **中多态类型的指称语义**. *Exp* **语言的语法如下所示**:

e ::= x | λ$x.e$ | e_1 (e_2) | **let** x=e_1 **in** e_2 | **if** e_1 **then** e_2 **else** e_3

在此, x **为标识符**(**包含常量**); e_1 (e_2)**表示函数的应用**; **let-表达式表示局部标识符** x **在其作用域** e_2 **内的值为** e_1 ; **if -表达式即为一般的条件表达式**.

Exp **语言的类型结构可简单地描述如下**:

σ ::= Int | Bool | α | σ→σ | σ×σ | (x:σ).σ
　　∀α.σ | ∃α.σ

其中, α**为类型变量**; σ→σ**为函数类型**; σ×σ**是通过笛卡儿乘积构造的序对类型**; (x:σ).σ**为约束类型**, (x:σ)**称为一个约束**, **并限定约束只出现在全称量化类型中**, **关于约束类型引入的必要性及其作用**, **见另文**; ∀α.σ**是全称量化类型**, **它是参数化多态的表现形式**; ∃α.σ**是存在量化类型**, **我们将这种类型作为** Garment **中数据抽象过程的表示**.

我们通过定义一个语义函数 *Se*(Semantics of expressions)**给出** *Exp* **语言中表达式的语义**, *Se* **将每个表达式映射到某个指称域的一个元素**. **与研究**λ-**演算类似**, **根据** *Exp* **的类型结构**, **将域**

D = Bool + Int + [D→D] + [D×D] + {*wrong* , ⊥}

作为表达式 *e* **的指称域**, **其中**, Bool={*true*, *false*, ⊥}, Int **为包含最小元**⊥**的整数集合**, **易见**: **在平坦序下**, Bool, Int **所对应的集合是** Ideal , *wrong* **用于标识错误**, ⊥**为** D **中的最小元**, [D→D]**为**λ**项的语义域**, [D×D]**为序对类型值的语义域**. **由**[9][10]**中的讨**

论可知: D 是 cpo.

类似地, 通过定义另一个语义函数 *St*(Semantics of types)给出类型表达式σ的语义. *St* 将每个类型表达式映射到类型表达式的指称域 p(D).

下面通过定义 *Se*, *St* 给出 *Exp* 的形式语义.

基本域

EXP 表达式 e 的语法域; TE 类型表达式σ的语法域;

ID 标识符域; Tid 类型标识符域;

D 表达式的指称域; p(D) 类型表达式的指称域.

语义函数

Se : EXP→Env→D , 其中: Env = ID→D, Env 给出表达式中自由标识符的语义. 设

$$\rho\in \text{Env},\ \text{定义}\rho[a/x](y) = \begin{cases} a & if\ y = x \\ r(y) & if\ y \neq x \end{cases}$$

St : TE →Te→p(D) , 其中: Te = Tid→p(D), Te 给出类型标识符及类型变量α的语义.

$$\text{设}\eta\in \text{Te},\ \text{定义}\eta[\text{I}/a](b) = \begin{cases} I & if\ b = a \\ h(b) & if\ b \neq a \end{cases}$$

Cond : Bool→D→D→D, Cond' : Bool→p(D)→p(D)→p(D). 在此, 为表示方便将 Cond *t v v*'和 Cond' *t v v*'统一记为 *t* ~≻ *v* , *v*'. 定义

$$t \sim\succ v\ , v' = \begin{cases} v & if\ t = true \\ v' & if\ t = false \\ \perp & if\ t = \perp \end{cases}.$$

语义方程

Se [*x*]ρ = ρ(*x*) ; *Se* [λ*x*.*e*]ρ = λ*a*∈D. *Se*[*e*](ρ[*a*/*x*])

Se [e_1 (e_2)]ρ = v_1 ∈ [D→D] ~≻ v_1 (v_2), *wrong* where v_i = *Se*[e_i]ρ , (i=1,2)

Se [**let** *x*=e_1 **in** e_2]ρ = *Se*[e_2](ρ[*v*/*x*]) where *v* = *Se*[e_1]ρ

Se [**if** e_1 **then** e_2 **else** e_3]ρ = (v_1=*true*) ~≻ v_2 , v_3 where v_i = *Se*[e_i]ρ , (i=1,2,3)

St [bool]η = Bool ; *St* [int]η = Int

St [α]η = η(α) ; *St* [$\sigma_1\times\sigma_2$]η = *St*[σ_1]η ⊗ *St*[σ_2]η

St [$\sigma_1 \to\sigma_2$]η = *St*[σ_1]η ↦ *St*[σ_2]η

St [(*x*:σ_1).σ_2]η = (*Se*[*x*]ρ∈*St*[σ_1]η) ~≻ *St*[σ_2]η , {⊥}

St [∀α.σ]η = $\cap_{I\in ℵ}$ *St*[σ](η[*I*/α]) , 其中ℵ⊆ p(D)

St [∃α.σ]η = $\cup_{I\in ℵ}$ *St*[σ](η[I/α]) , 其中ℵ⊆ p(D), ∪表示广义并.

一般地, 程序设计语言所能表示的类型只是 p(D)的很小一部分. p(D)是一个很大的集合, 如: 整数的任意一个子集就确定一个 Ideal(因此是个类型), p(D)足以容纳许多不同的类型系统. 因此, 讨论类型系统时需确定一个特殊的语言环境(如, *Exp*)中所能表达的那些 Ideal, 这些 Ideal 构成的集合记为ℵ, 这即是ℵ⊆ p(D)的含义. 全称量化类型通过类型变量被替换为实际类型而被例化, 如: ∀α.α→α可例化为 Bool→Bool, Int→Int 等, 但并不是说全称量化的类型变量可以例化为 p(D)中所有的类型, 要求这些类型在ℵ中. 尤其是对约束多态类型, 如:

$\forall\alpha.(\leq:\alpha\times\alpha\to Bool).seq[\alpha]\times seq[\alpha]\to Bool$, 不仅要求$\alpha$的例化类型 T 在$\aleph$中, 而且还要求在类型 T 上有$\leq:T\times T\to Bool$, 即在类型 T 上满足约束$\leq:\alpha\times\alpha\to Bool$.

现在以恒等函数 Id: $\forall\alpha.\alpha\to\alpha$为例具体讨论在 Ideal 模型下对全称量化类型的语义方程的理解. 由定义 2 可得

$$Bool\mapsto Bool = \{f\in[D\to D]|f(Bool)\subseteq Bool\}\cup\{\bot\}$$

我们并不关心这些函数对 D 中其它值的作用, 只要 $f(Bool)\subseteq Bool$ 即可, 而 $Id(Bool)\subseteq Bool$, 那么 $Id\in Bool\mapsto Bool$, 类似地, 亦有 $Id\in Int\mapsto Int$. 事实上, 对于任意类型 $T\in\aleph$, 都有 $Id\in T\mapsto T$. 所以有 $Id\in\bigcap_{T\in\aleph}T\mapsto T$. 同样可以解释 length: $\forall\alpha.seq[\alpha]\to int$ 为 $length\in\bigcap_{T\in\aleph}seq(T)\mapsto Int$. 这样, 不难理解 $St[\ \forall\alpha.\alpha\ ]\eta=\{\bot\}$, 也即$\bot$可以是任意类型. 一般将$\forall\alpha.\alpha$定义为: $\forall\alpha.\alpha=Unit$, 它对应于定理 2 中格的最小元. 对应地, 可以定义$\exists\alpha.\alpha=Top$, 意为对于 Top 中的每个值都存在一个类型, 使得该值具有此类型. 于是 Top 即为所有值的集合, 实际是 $p(D)$中所有 Ideal 的广义并, 它对应于定理 2 中格的最大元.

存在量化类型在 Ideal 模型下不够直观, 如: 类型为$\exists\alpha.\alpha\times\alpha$的值并不是象所期望的那样是相同类型元素的序对, 如: (3,3), (*true*,*true*)等. 事实上, (3,*true*)也是这个类型的值, 因为 3, *true* 都具有类型 Top, 即: 存在α=Top 使得$(3,true):\alpha\times\alpha$. $\exists\alpha.\alpha\times\alpha$实际上表示的是所有序对, 并不只表示由相同类型的元素构成的序对, 它与$\exists\alpha,\beta.\alpha\times\beta$所表示的集合相同. 类似地, 如果取$\alpha$=Top, 任何函数都有类型$\exists\alpha.\alpha\to\alpha$. 由此可以看出, 存在量化类型表示的是相关 Ideal 的广义并.

在[7]中定义了类型环境 A 下全称量化类型间的实例关系$\leq_A$: 对全称量化类型 σ, σ_1 , 若存在一组关于σ中类型变量的替换 S, 使得$\sigma_1=\sigma S$, 则说$\sigma_1\leq_A\sigma$. 在类型的 Ideal 模型中, 这种实例关系可解释为: 若 $St[\ \sigma_1\ ]\eta\supseteq St[\ \sigma\ ]\eta$, 则说$\sigma_1\leq_A\sigma$. 如: 在替换[Bool/$\alpha$]下, $Bool\to Bool\leq_A\forall\alpha.\alpha\to\alpha$, 由定义 2 和语义方程 $St[\ \forall\alpha.\sigma\ ]\eta$可得: $Bool\mapsto Bool\supseteq\bigcap_{I\in\aleph}St[\ \sigma\ ](\eta[I/\alpha])$.

类型环境 A 中的类型假设 $y:(x:\sigma_1).\sigma_2$ 意为若约束 $x:\sigma_1$ 在 A 中是可满足的, 那么有 $A\ |-\ y:\sigma_2$, 否则 y:Unit. 说约束 $x:\sigma_1$ 在 A 中是可满足的, 如果$(x:\sigma_1)\in A$ 或$(x:\sigma)\in A$ 且 $\sigma_1\leq_A\sigma$. 对应地, 在 Ideal 模型下, 约束 $x:\sigma_1$ 在 A 中的可满足性为: $Se[\ x\ ]\rho\in St[\ \sigma_1\ ]\eta$ 或 $Se[\ x\ ]\rho\in St[\ \sigma\ ]\eta$, 且 $St[\ \sigma_1\ ]\eta\supseteq St[\ \sigma\ ]\eta$, 故 $Se[\ x\ ]\rho\in St[\ \sigma_1\ ]\eta$. 合并这两个条件可得: 只需验证 $Se[\ x\ ]\rho\in St[\ \sigma_1\ ]\eta$即可判定约束$(x:\sigma_1)$的可满足性. 因此对约束类型有语义方程. $St[\ (x:\sigma_1).\sigma_2]\eta\ =\ (Se[\ x\ ]\rho\in St[\ \sigma_1\ ]\eta)\ \rightsquigarrow\ St[\ \sigma_2\ ]\eta\ ,\ \{\bot\}$. 事实上, 由下节的结论可以看出, $Se[\ x\ ]\eta\in St[\ \tau\ ]\eta$即是类型推理系统在类型环境 A 下有 $A\ |-\ x:\tau$.

4 类型规则的语义可靠性

类型系统是带类型的程序设计语言不可缺少的部分, 类型规则是在语法(直觉)级别上描述类型系统的有力工具, 并且易于将这些类型规则形式化为一个类型检查或推理系统, 而规则的可靠性需通过类型的一种语义来解释[11]. 通过上节的讨论可以看出, 类型的 Ideal模型足以解释参数化多态以及约束多态. 这一节

将用其证明 *Exp* 语言类型系统中类型规则的语义可靠性.

类型系统可以用于证明表达式 e:σ, 表示 e 的类型为σ. 一般来说, e:σ依赖于 e 中标识符的类型, 标识符的类型收集于一个类型假设集合中. 一个类型假设集合 A 是由形为 x:σ的类型假设构成的有限集合, 有时也称该集合为类型环境. 类型规则中 $A \vdash e$:σ表示由类型假设集合 A 可以得出 e 具有类型σ.

类型规则的依据是类型表达式的语法, 大部分对应于各个类型构造符的引入和消去规则. 假设常量标识符的类型假设已在 A 中, 下面给出 *Exp* 语言类型系统中的一部分类型规则:

标识符: [Ide] $A \vdash x : \sigma$ if $x : \sigma \in A$

函数构造符: [→Intro] $\dfrac{A \cup \{x : s_1\} \vdash e : s_2}{A \vdash \lambda x.e : s_1 \to s_2}$ (x 不出现于 A)

[→Elim] $\dfrac{A \vdash e_1 : s \to t,\ A \vdash e_2 : s}{A \vdash e_1(e_2) : t}$

全称量词∀: [∀-Intro] $\dfrac{A \vdash e : s}{A \vdash e : \forall a.s}$ (a 不出现于 A)

[∀-Elim] $\dfrac{A \vdash e : \forall a.s}{A \vdash e : s[t/a]}$

类型约束: [Cons-Intro] $\dfrac{A \cup \{x : s_1\} \vdash e : s_2}{A \vdash e : (x : s_1).s_2}$ (x 在 A 中重载)

[Cons-Elim] $\dfrac{A \vdash e : (x : s_1).s_2 \quad , \quad A \vdash x : s_1}{A \vdash e : s_2}$ (x 在 A 中重载)

存在量词∃: [∃-Intro] $\dfrac{A \vdash e : s[t/a]}{A \vdash e : \exists a.s}$

[∃-Elim] $\dfrac{A \vdash e_1 : \exists a.s \ , \ A \cup \{x : s[g/a]\} \vdash e_2 : t}{A \vdash e_2[e_1/x] : t}$ (a 不出现于 A 或 t 中)

给定类型环境 A 和语义环境ρ∈Env, η∈Te. $\models_{\rho,\eta} A$ 表示: 若 $A \vdash x$:σ, 则 $Se[x]\rho \in St[\sigma]\eta$. $A \models_{\rho,\eta} e$:σ表示: 若$\models_{\rho,\eta} A$, 则 $Se[e]\rho \in St[\sigma]\eta$. $A \models e$:σ表示对所有的ρ∈Env 和η∈Te, 都有 $A \models_{\rho,\eta} e$:σ. 具体地, $\models_{\rho,\eta} A$ 意为: 若从当前环境ρ中取出标识符 x 的值,则此值必与 x 所应具有的类型(在类型环境 A 中标识的类型)相一致, 这与 SECD 抽象机对λ-表达式求值过程非常相似. 在λ-演算的 SECD 定义中, 取出一个标识符 x 的值即是将环境 E 中 x 的当前值 v 压入栈 S 的栈顶, 这一过程由下面的两个动作完成:

<i> 根据 x 的类型在 S 的顶部确定出足够的空间以容纳值 v;

<ii> 将 v 填入此空间.

我们可以将下面的类型规则语义可靠性定理视为上述两个动作的形式描述.

定理 5　对于上面给出的类型规则, 若 $A \vdash e$:σ, 则 $A \models e$:σ.

证明：对 $A \vdash e:\sigma$ 过程所形成的演绎树长度进行归纳：

树的长度为 1 **时，即是**[Ide]**规则**. **根据** $A \models e:\sigma$ **的定义有**：**若** $A \vdash x:\sigma$，**则** $A \models x:\sigma$.

假设树的长度为 n **时有**：**若** $A \vdash e:\sigma$，**则** $A \models e:\sigma$.

树的长度为 n+1 **时，任取** $\rho \in Env, \eta \in Te$, **满足** $\models_{\rho,\eta} A$, **考虑演绎中最后使用的规则**：

[Cons-Intro]**规则**：**需证** $A \models e:(x:\sigma_1).\sigma_2$，**即** $Se[\![e]\!]\rho \in St[\![(x:\sigma_1).\sigma_2]\!]\eta$, **其中**

$$St[\![(x:\sigma_1).\sigma_2]\!]\eta = (Se[\![x]\!]\rho \in St[\![\sigma_1]\!]\eta) \rightsquigarrow St[\![\sigma_2]\!]\eta , \{\bot\}$$

任取 $a \in St[\![\sigma_1]\!]\eta$, **令** $\rho'=\rho[a/x]$, **因** $\models_{\rho,\eta} A, x$ **在** A **中重载，由于** x **的所有重载类型不重叠，故有** $\models_{\rho',\eta} A \cup \{x:\sigma_1\}$, **由归纳假设，有** $A \cup \{x:\sigma_1\} \models e:\sigma_2$ **，那么，** $Se[\![e]\!]\rho' \in St[\![\sigma_2]\!]\eta$.

[Cons-Elim]**规则**：**需证** $A \models e:\sigma_2$，**即** $Se[\![e]\!]\rho \in St[\![\sigma_2]\!]\eta$.

由归纳假设，有 $A \models e:(x:\sigma_1).\sigma_2$，**即** $Se[\![e]\!]\rho \in St[\![(x:\sigma_1).\sigma_2]\!]\eta$, **其中**

$$St[\![(x:\sigma_1).\sigma_2]\!]\eta = (Se[\![x]\!]\rho \in St[\![\sigma_1]\!]\eta) \rightsquigarrow St[\![\sigma_2]\!]\eta, \{\bot\}$$

亦有 $A \models x:\sigma_1$，**即** $Se[\![x]\!]\rho \in St[\![\sigma_1]\!]\eta$. **则** $St[\![(x:\sigma_1).\sigma_2]\!]\eta = St[\![\sigma_2]\!]\eta$, **故** $Se[\![e]\!]\rho \in St[\![\sigma_2]\!]\eta$.

[∃-Intro]**规则**：**需证** $A \models e:\exists\alpha.\sigma$, **即** $Se[\![e]\!]\rho \in St[\![\exists\alpha.\sigma]\!]\eta$. **其中对于任一** $\aleph \subseteq p(D)$

$$St[\![\exists\alpha.\sigma]\!]\eta = \bigcup_{I \in \aleph} St[\![\sigma]\!](\eta[I/\alpha])$$

由归纳假设，有 $A \models e:\sigma[\tau/\alpha]$, **即** $Se[\![e]\!]\rho \in St[\![\sigma[\tau/\alpha]]\!]\eta$，**其中**

$$St[\![\sigma[\tau/\alpha]]\!]\eta = St[\![\sigma]\!](\eta[(St[\![\tau]\!]\eta)/\alpha])$$

因 $St[\![\tau]\!]\eta$ **是一个** Ideal，**故** $St[\![\sigma]\!](\eta[(St[\![\tau]\!]\eta)/\alpha]) \subseteq \bigcup_{I \in \aleph} St[\![\sigma]\!](\eta[I/\alpha])$. **所以**, $Se[\![e]\!]\rho \in \bigcup_{I \in \aleph} St[\![\sigma]\!](\eta[I/\alpha])$, **即** $Se[\![e]\!]\rho \in St[\![\exists\alpha.\sigma]\!]\eta$.

[∃-Elim]**规则**：**需证** $A \models e_2\ [e_1/x]:\tau$.

由归纳假设，有 $A \models e_1:\exists\alpha.\sigma$ **和** $A \cup \{x:\sigma[\gamma/\alpha]\} \models e_2:\tau$, **因此有** $Se[\![e_1]\!]\rho \in St[\![\exists\alpha.\sigma]\!]\eta$. **取** $a=Se[\![e_1]\!]\rho$, **由** $St[\![\exists\alpha.\sigma]\!]\eta$ **的定义可知**：**存在一个** Ideal I，**使得** $a \in St[\![\sigma]\!]\eta'$. **其中**：$\eta'=\eta[I/\alpha]$. **若令** $\rho'=\rho[a/x]$, **因** x **不出现于** A **中，由归纳假设就有** $\models_{\rho',\eta'} A \cup \{x:\sigma[\gamma/\alpha]\}$, **其中** $St[\![\gamma]\!]\eta'=I$, **进而** $Se[\![e_2]\!]\rho' \in St[\![\tau]\!]\eta'$. **因**$\alpha$**不出现于**$\tau$**中**($\alpha$**已被替换为**$\gamma$), **故** $St[\![\tau]\!]\eta' = St[\![\tau]\!]\eta$. **由此可得** $Se[\![e_2]\!]\rho' \in St[\![\tau]\!]\eta$. **于是**

$$Se[\![e_2\ [e_1/x]]\!]\rho = Se[\![e_2]\!](\rho[(Se[\![e_1]\!]\rho)/x]) = Se[\![e_2]\!]\rho' \in St[\![\tau]\!]\eta.$$

同理，易证其它类型规则的语义可靠性.

5 结语

Ideal **模型是** Set **模型的发展，除了** Ideal **以外**，Retract **通常也作为类型的模型**. **与类型的其它模型相比较**，Ideal **模型在解释简单类型和多态类型上比较直观，即以值的集合作为类型的模型，尤其在处理类型继承上也比较自然**[12]. Ideal**本是用来作为函数多态(尤其是自应用函数)语言语义基础的理论，而它却可以作为理解继承的基础，这似乎是偶然发现**[12][13]. **正是为使得类型的解释尽量直观(追求直观)，我们选择了** Ideal **模型作为理解** Garment **中类型的基础**. Ideal **模型的不足之处在于，它在处理参数化类型上不够直接，因为在** Ideal **模型下，类型不是语言值域** D **上的值，另外在处理类型构造符(如**: $\times, +, \rightarrow$**)上超出了** Ideal **模型的范围，不得**

不将类型构造符视为 Ideal 上的函数($\otimes, \oplus, \mapsto$).

类型作为参数的思想已在二阶λ-演算中得到应用，并已扩充进λ-演算中，使其同时支持函数抽象和类型两种抽象机制，这样即能在同一数学模型(如:Retract)中描述λ-演算和多态类型．二阶λ-演算的指称模型一般都取为 Retract，在 Retract 模型中，类型不是值的集合，而是 Retract(对 $f\in[D\to D]$，若 $f=f\cdot f$，则称 f 为 Retract)的集合．Retract 模型解释显式的类型参数很自然，而 Ideal 模型解释隐式的类型参数更自然一些．事实上，Retract 是 D 上的值，Ideal 也是值(因为值的集合亦是一种值)，Retract 之所以可以直观地解释类型参数(抽象)，如[5]中的Λt.e，是因为其值是 D 上的值，因此可以象处理其它表达式(如$\lambda x.e$)一样处理类型表达式，但 Ideal 与 D 上的值不同．

本文从 Ideal 的基本概念出发，讨论了 Ideal 的一般性质，将类型的 Ideal 模型作为理解 Garment 中多态类型(参数化类型，约束类型)，及数据抽象的统一框架，以 *Exp* 语言作为 Garment 抽象出语言的模型，研究了 *Exp* 语言的语义，证明了其类型系统相对于其语义的可靠性．类型语义的表示独立性及其与 Garment 中数据抽象的关系是需要进一步研究的问题．

致谢　感谢王捍贫博士与作者在形式语义方面富有成果的讨论．诚挚地感谢审稿人对本文提出的指正和建议．

参考文献

[1] 张乃孝，许卓群，屈婉玲．面向模型的变换型软件开发方法研究．《理论计算机科学》Vol.2，上海：上海科学技术出版社，1994, 54-64.

[2] Geoffrey Smith. Polymorphism Type Inference with Overloading and Subtyping. ACM Transactions on Programming Languages and Systems, 1996, 18(3): 254-267.

[3] Dana Scott. Data Type as Lattices. SIAM Journal on Computing, 1976, 5(3): 522-587.

[4] Andrzej Borzyszkowski *et al*. A Set-Theoretic Model for a Typed Polymorphic Lambda Calculus. LNCS, 1988, 288: 267-298.

[5] James Donahue. On the Semantics of "Data Type". SIAM Journal on Computing, 1979, 3(4): 546-560.

[6] David Macqueen and Gordon Plotkin. An Ideal Model for Recursive Polymorphic Types. Information and Control, 1986, 71: 95-130.

[7] Robinson Milner. A Theory of Type Polymorphism in Programming. Journal of Computer and System Science, 1978, 17: 348-375.

[8] John Reynolds. Polymorphism is not Set-theoretic. LNCS, 1984, 173: 145-156.

[9] 陆汝钤．计算机语言的形式语义．北京：科学出版社，1992.

[10] 周巢尘．形式语义学导论．长沙：湖南科学技术出版社，1985.

[11] John Mitchell. Type Systems for Programming Languages. In: J. van Leeuwen ed. Handbook of Theoretical Computer Science, Elsevier Science Publisher, 1990, 364-458.

[12] Luca Cardelli and Peter Wegner. On Understanding Types, Data Abstraction, and Polymorphism. ACM Computing Surveys, 1985, 17(4): 471-525.

[13] Scott Danforth and Chris Tomlinson. Type Theory and Object-Oriented Programming. ACM Computing Surveys, 1988, 20(1): 29-70.

An Ideal Model for Polymorphic Types in Garment

Zheng Hongjun Zhang Naixiao

(*Peking Unviversity*)

Abstract Starting with the concept of ideal, the paper studies their properties when ideals are taken as the semantic model for types. Under the ideal model of types, the paper discusses the semantics for polymorphic types in Garment, including parametric polymorphic types and constrained polymorphic types. Finally, the semantic soundness of typing rules in Garment is proved within the ideal model.

Keywords Ideal Polymorphic type Type system Semantics Garment