Garment 中的归约语义*

郑红军¹ 张乃孝²
(北京大学, 北京 100871)

摘要 本文用代数方法研究了 Garment 中程序设计语言的归约语义, 首先给出了归约语义在形式语言理论中的含义, 然后提出了 Garment 中语言的代数模型. 在此代数模型下讨论了归约语义及其性质, 并给出了语言可归约的充分条件.

关键词 Garment 程序设计语言 归约语义 基调 项代数

1. 引言

通过研究程序设计语言与一类专用软件及其学科领域间的内在联系, 我们提出了一种系统的软件开发方法称为“面向模型的变换型软件开发方法” [1]. 这种方法把数据抽象的思想提高到语言抽象层次; 将软件规范归结为语言的抽象; 把软件的实现抽象为语言的归约. 为支持这种方法, 我们提出了一种语言的抽象与封装机制 *Garment* [2].

Garment 能够对已有的语言进行抽象, 使得抽象出的语言易于描述某一学科领域的问题. 每个 *Garment* 封装了一个语言的定义和实现, 是一种更高层次的可重用部件. *Garment* 对一个语言 L 的抽象可简单地记为: $L' = Gt(L)$, 其中 L 或者是核心语言, 或者是由 *Garment* 已抽象出的语言, 并且 L 的语义是已知的, Gt 描述了 L' 对语言 L 的继承, 屏蔽和扩充, 构成了 L' 的 *Garment* 描述. 在不考虑屏蔽的情况下 (在 *Garment* 中描述 L' 的归约语义时, 不存在屏蔽的问题, 被屏蔽的语言成分只是对语言的使用者来说是不可用的), L 与 L' 间的关系可以表示为 $L = L' \setminus F_1, \dots, F_n$, 其中, $F_1, \dots, F_n$ 是语言 L' 中的成分, 但不是 L 中的成分, L' 包含 L 的所有语言成分, 并且与在 L 中的语义相同. 简言之, 在语法和语义上, L 都是 L' 的子语言. 所以, Gt 只需描述 $F_1, \dots, F_n$ 的语法和语义, 即可构成一个新语言 L' . 本文将讨论描述这些扩充成分 $F_1, \dots, F_n$ 语义的方法及其代数基础. 关于 *Garment* 的语法结构及实现, 参见文献 [2-3].

我们采用 Backus 提出的归约语义 [8] 描述 $F_1, \dots, F_n$ 的语义. 基本思想是用

* 本项研究得到国家自然科学基金(69683006)的资助.

¹ 北京大学 计算机科学与技术系(北京 100871)博士生, 1969 年生, 研究方向为软件开发方法学与程序设计语言.

² 本文联系人: 北京大学 数学科学院 信息科学系(北京 100871)教授, 1942 年生, 1965 年毕业于北京大学数学力学系, 研究方向为软件开发方法学与程序设计语言.

语言 L 来描述 L 中 $F_1, \dots, F_n$ 的语义, 将 $F_1, \dots, F_n$ 归约至语言 L 的成分 (L 的语义是已知的). 相当于语言 L 是 L' 的基本结构 (固定部分), $F_1, \dots, F_n$ 是 L' 中由基本结构所确定的可变部分, 所以, 可以说 L 是 L' 可变部分的通用环境. 归约语义是用一组归约规则描述的一种操作语义, 这些规则构成了一个易于理解和语言实现的归约系统. 直观地看, 归约语义的描述方法是用已有的语言解释新引入的语言成分, 如:

$$e_1 \text{ mod } e_2 \Rightarrow e_1 - (e_1 \text{ div } e_2) * e_2$$

$$\text{repeat } s \text{ until } e \Rightarrow s ; \text{ while not } e \text{ do } s$$

$$\text{let } x = e_1 \text{ in } e_2 \Rightarrow \lambda x. e_2 [e_1 / x]$$

Pepper 将归约语义发展为变换语义 [9], 在此, 我们从语言的归约实现角度在程序设计语言的代数模型中讨论归约语义及其性质, 这种代数方法是 Garment 中语义描述的数学基础, 语言归约系统的性质对在 Garment 中正确使用归约语义具有指导作用.

2. 归约语义

为能更清晰地理解 Garment 中语言的代数模型及归约系统的代数性质, 我们首先在形式语言理论中简单地讨论一下程序设计语言及归约语义中的一些概念.

一个上下文无关语法是一个四元组 $G=(N,T,P,B)$, 其中: N 是非终结符的有限集; T 是终结符的有限集; P 是产生式集合, 每个产生式的形式为 $A \rightarrow \alpha$, 且 $A \in N$, $\alpha \in (N \cup T)^*$; B 是起始符. 由语法 G 生成的语言记为 $L(G)$.

G 中的句型可在 $(N \cup T)^*$ 上递归定义如下: $v_N \in N$ 是一个句型, 如果 $\alpha\beta\gamma$ 是句型, 且 $\beta \rightarrow \delta$ 是 P 中的产生式, 则 $\alpha\delta\gamma$ 也是句型, 并称 $\alpha\beta\gamma$ 与 $\alpha\delta\gamma$ 具有直接派生关系, 表示为 $\alpha\beta\gamma \Rightarrow \alpha\delta\gamma$, 用 $\overset{*}{\Rightarrow}$ 表示 $\Rightarrow$ 的自反传递闭包.

设 $G=(N,T,P,B)$, 对于 T^* 中的句型 p , 若满足 $(\exists v_N \in N). v_N \overset{*}{\Rightarrow} p$, 则称 p 是一段程序. 对于 $(N \cup T)^*$ 中的句型 σ , 若满足 $(\exists v_N \in N). v_N \overset{*}{\Rightarrow} \sigma$, 则称 σ 是一个程序模式. 出现于 σ 中的非终结符称为模式变量, σ 中所有模式变量构成的集合记为 $SV(\sigma)$. 一个程序模式 σ 对应于一组程序: $\{p \in T^* | \sigma \overset{*}{\Rightarrow} p\}$. $\sigma \overset{*}{\Rightarrow} p$ 是产生式制导的合一过程, 得到关于 σ 中模式变量 $v_1, \dots, v_n$ 的一组替换 $[p_1/v_1, \dots, p_n/v_n]$, 满足 $v_i \overset{*}{\Rightarrow} p_i$ ($v_i \in SV(\sigma), p_i \in T^*, 1 \leq i \leq n$). 这样, σ 中的每个模式变量 v_i 都对应于一段程序 p_i , 称 p_i 是变量 v_i 的例化, 将这组替换作用于程序模式 σ 得到程序 $p = \sigma[p_1/v_1, \dots, p_n/v_n]$, 称 p 是 σ 的例化.

一对程序模式 σ_i (输入模式), σ_o (输出模式) 与一个谓词 Pre 一起称为一个规则 R , 记为 $Pre: \sigma_i \rightarrow \sigma_o$, 其中 $SV(Pre) \subseteq SV(\sigma_i), SV(\sigma_o) \subseteq SV(\sigma_i)$, Pre 为规则 R 的应用

条件. 一组规则构成的集合记为 $\mathfrak{R}$. 对于一个程序 p 和一个规则 $Pre: \sigma_i \rightarrow \sigma_o$, 如果 p 中存在一程序段 s , 满足 $\sigma_i \xrightarrow{*} s$, 得到一组关于 σ_i 中模式变量的替换 θ , 如果将 θ 作用于 Pre 后得到的谓词 $Pre\theta$ 是可满足的, 则说规则 R 可应用于 p , 应用的结果是 $p' = p[(\sigma_o\theta)/s]$ (将 p 中的程序段 s 替换为 $\sigma_o\theta$), 记为 $p \xrightarrow{R} p'$. 对应地, $p \xrightarrow{\mathfrak{R}} p'$ 意为 $p \xrightarrow{R1} p_1 \xrightarrow{R2} p_2 \dots \xrightarrow{Rn} p'$, 其中 $R_i \in \mathfrak{R} (1 \leq i \leq n)$. 在此, 我们仅讨论无条件的规则, 关于谓词的可满足性判定是谓词逻辑中的问题, 不是本文的重点.

设 $G=(N,T,P,B), G'=(N',T',P',B)$, 如果 $N \subseteq N', T \subseteq T', P \subseteq P'$, 则称 $L(G)$ 在语法上是 $L(G')$ 的子语言, 记为 $L(G) \subseteq L(G')$, 在不产生混淆的情况下, 有时也简单地记为 $L \subseteq L'$. 如果存在一组规则 $\mathfrak{R}$ 满足 $(\forall p' \in L', \exists p \in L). p' \xrightarrow{\mathfrak{R}} p$, 则说 L' 可通过 $\mathfrak{R}$ 归约为 L , 称 $\mathfrak{R}$ 是 L' 到 L 的归约系统, $\mathfrak{R}$ 中的规则为归约规则. 在 Garment 中, 将语言 L' 归约至 L 的过程实际是将 L' 中的每个成分通过 $\mathfrak{R}$ 归约为 L 中等价的成分, 这种等价性是语言归约所应保持的性质, 我们采用语义等价原则.

抽象地看, $L(G)$ 的语义是一个可计算函数 $Sm: L(G) \rightarrow (Sd_1 \rightarrow Sd_2)$, 其中 $L(G)$ 表示语法域, Sd 是语义域, Sm 将 $L(G)$ 的程序解释为语义域上的函数 $Sd_1 \rightarrow Sd_2$. 假设 $L \subseteq L', Sm_L$ 与 $Sm_{L'}$ 分别是它们的语义, 如果一个归约规则 R 满足: 若 $p' \xrightarrow{R} p$, 则 $Sm_{L'}(p') = Sm_L(p)$, 那么, 就说 R 是正确的. 其中, $p \in L, p' \in L'$.

由上面的讨论可知, 对于一般的程序设计语言, 其语言的语法是上下文无关语法, 语义是一种从语法域到语义域的同态, 描述这种同态的一种比较好的方法是采用抽象数据类型中的初始代数方法, 而且在代数系统中易于研究归约系统的一般性质, 这正是我们给出语言的代数模型的主要原因.

3. Garment 中语言的代数模型

在[7]中给出的语言代数模型中, 用基调 $\Sigma = (S,O)$ 描述语言 $L(G)$ 的语法 G , 其中 S 和 O 分别为 Σ 的类子集和操作集. S 中的每个类子 s_i 对应于 G 中的一个基本语法元素(非终结符); O 中的每个操作(语法元素间的组合关系) $o_h: s_1 \times s_2 \times \dots \times s_n \rightarrow s_0 (s_i \in S, 0 \leq i \leq n)$ 对应于 G 中的产生式 $o_h: s_0 ::= t_1 s_1 t_2 s_2 \dots t_n s_n t_{n+1} (s_i \in N, t_i \in T^*)$. 可以看出, 基调 Σ 中的操作是语法 G 中产生式 o_h 的抽象描述[6]. 关于语法 G 与基调 Σ 间对应关系的详细讨论, 参见文献[7].

设基调 $\Sigma = (S,O)$, 如果为 S 中的每个类子 s_i 指定一组具体的对象 As_i (称为 s_i 的载体集), 为 O 中的每个操作 $o_h: s_1 \times s_2 \times \dots \times s_n \rightarrow s_0$ 指定一个具体的在诸 As_i 元素之上的函数 $f_h: As_1 \times As_2 \times \dots \times As_n \rightarrow As_0$, 则 $\langle A, F \rangle$ 构成了一个 Σ 代数, 记为 $\Sigma(A)$ 或 A , 其中 $A = \{ As_i \mid s_i \in S \}, F = \{ f_h \mid o_h \in O \}$. Σ 的基项代数记为 $T(\Sigma)$. 每个 As_i 中的每个元素称为一个基项, 程序即是 $T(\Sigma)$ 中的基项, 程序模式是有限生成的项代数 $T(\Sigma, X)$ 中的项, 其中 $X = \{ Xs_i \mid s_i \in S \}$ 是模式变量集组, Xs_i 是为类子 s_i

引进的变量集. 若有一指派 φ , 把 X_{S_i} 中的每个元素与上面 A_{S_i} 中的一个基项相对应, 则指派 $\varphi: X \rightarrow T(\Sigma)$ 称为在 $T(\Sigma)$ 中对 X 的赋值, 这种赋值对应于 $\sigma \xrightarrow{*} p$ 过程中得到的一组替换, 由[6]中的定理 5.2.4, φ 可以唯一地扩充为项代数 $T(\Sigma, X)$ 到基项代数 $T(\Sigma)$ 的 Σ 同态 $\varphi^*: T(\Sigma, X) \rightarrow T(\Sigma)$, 它保持 X 的赋值不变, 这就是对程序模式 σ 的例化过程, 即 $p = \varphi^*(\sigma)$.

与同一基调 Σ 对应的代数不是唯一的, 基调 Σ 的所有代数在 Σ 同态关系下构成一个范畴 $\mathbf{Alg}(\Sigma)$, 基调 Σ 的基项代数 $T(\Sigma)$ 是 $\mathbf{Alg}(\Sigma)$ 中的初始代数[6]. 由 $T(\Sigma)$ 的构造过程可以看出, $T(\Sigma)$ 中的每个载体集 A_{S_i} 与 Σ 中的类子 s_i (基本语法元素) 相对应, $T(\Sigma)$ 中的 A_{S_i} 即可作为类子 s_i 的语法域, 因此, $T(\Sigma)$ 即是语义描述所必需的抽象语法.

设 $\Sigma = (S, O)$ 是语言 L 的基调, 如果为 Σ 中的每个类子 $s_i \in S$ 指定一个载体集 M_{S_i} , 为每个操作 $o_h \in O$ 指定一个函数 $o_{hM}: M_{S_1} \times \dots \times M_{S_n} \rightarrow M_{S_0}$, 则这些载体集和函数构成了一个 Σ 代数 M . 因 $T(\Sigma)$ 是初始代数, 故存在唯一的同态 $Sm: T(\Sigma) \rightarrow M$. 若将载体集 M_{S_i} 作为语义域, 函数 o_{hM} 为语义域上的函数, 那么, 同态 Sm 为 $T(\Sigma)$ 中的每个语法域 A_{S_i} 指定了一个语义域, $T(\Sigma)$ 中每一个基项在 M 中都有解释. 在此意义下, 可以说语义是从语法域到语义域的同态 $Sm: T(\Sigma) \rightarrow M$, 称 M 为 L 的语义代数. 同时, 同态 Sm 导出了一个基项上的同余关系 $\equiv_M: t_1 \equiv_M t_2$ iff $M(t_1) = M(t_2)$, 由此同余关系构成的同余类记为 $[t_1]_M$.

在下面的讨论中, 如不特别指出, 均有如下基本假设: 给定两个语言 $L \subseteq L'$, 且 $L = L' \setminus \{F_1, \dots, F_n\}$, 它们的基调分别是 $\Sigma = (S, O)$, $\Sigma' = (S \cup S', O \cup O')$, 记为 $\Sigma \subseteq \Sigma'$, 其中 S' 和 O' 是由 $F_1, \dots, F_n$ 引入的类子和操作构成的集合, L 和 L' 的语义代数分别为 M 和 M' , $\varphi^*: T(\Sigma', X) \rightarrow T(\Sigma)$.

在 Garment 中, L' 全部继承 L 的语法和语义, 只描述 L' 中 $F_1, \dots, F_n$ 的语法和语义 (Σ' 中的 S' 和 O' 描述了 $F_1, \dots, F_n$ 的语法), 在语义上要求 Σ 中的任一类子 s_i 在 M 和 M' 中对应的载体集 M_{S_i} 和 M_{S_i}' 恒等 (同构), 从而对 Σ 中的任一操作 $o_h: s_1 \times s_2 \times \dots \times s_n \rightarrow s_0$, 其对应的语义函数 $o_{hM}: M_{S_1} \times \dots \times M_{S_n} \rightarrow M_{S_0}$ 和 $o_{hM}': M_{S_1}' \times \dots \times M_{S_n}' \rightarrow M_{S_0}'$ 亦恒等, 即 M 是 M' 的子代数. 这样, 只需在 M' 中为 $s_i \in S'$ 指定语义域 M_{S_i}' , 为 $o_h \in O'$ 指定语义函数即可. 易见, 在 M' 中语义域和语义函数的指定并不是唯一的, 具有一定的自由度, 指定的自由度取决于所选择的子语言 L . Garment 中的表示不变式隐式地给出了 L' 的语义代数 M' 以及它与 M 的关系, 关于表示不变式的讨论详见[3-4].

假定 $\Sigma \subseteq \Sigma'$, 通过为 Σ' 引入一组归约规则 $\mathcal{R}$, 可以为 $T(\Sigma')$ 中的基项指定相对于 M 的归约语义. 一个归约规则是一个二元组 (lhs, rhs) , 记为 $lhs \Rightarrow rhs$, 其中, lhs

和 rhs 是项代数 $T(\Sigma', X)$ 中的项. 我们将 $L' = (\Sigma', \mathfrak{R})$ 作为 Garment 所抽象出语言的代数模型, 称 $\mathfrak{R}$ 为 Σ' 的归约系统. 下面, 就在这种代数模型中讨论 Garment 中归约系统 $\mathfrak{R}$ 具有的性质

4. 归约系统 $\hat{A}$ 的代数性质

定义 1 归约规则 $lhs \Rightarrow rhs$ 是正确的, 如果对所有的 $T(\Sigma')$ 中的赋值 φ , $[\varphi^*(lhs)]_{M'} = [\varphi^*(rhs)]_{M'}$. 若 rhs 是 $T(\Sigma, X)$ 中的项, 则要求 $[\varphi^*(lhs)]_{M'} = [\varphi^*(rhs)]_M$.

$[\varphi^*(lhs)]_{M'} = [\varphi^*(rhs)]_{M'}$ 意为 $\varphi^*(lhs)$ 与 $\varphi^*(rhs)$ 属于 M' 下的同一同余类. 显然, 定义 1 与第二节中归约规则正确性的定义是一致的. 在 Garment 中, 归约规则的正确性由表示不变式来保证, 可以用表示不变式证明归约系统 $\mathfrak{R}$ 中每个归约规则的正确性[3]. 如不特别指出, 在本文中假设归约系统 $\mathfrak{R}$ 是由正确的规则构成的. 容易得出: 归约系统 $\mathfrak{R}$ 中的规则在语义等价关系下具有自反性, 传递性, 对称性.

定义 2 $\mathfrak{R}$ 是一组归约规则构成的归约系统, 若在 $\mathfrak{R}$ 中有 $t_1 \xRightarrow{\mathfrak{R}} t_2$, 且 $SV(t_2) \subseteq SV(t_1)$, $t_2 \in T(\Sigma, X)$, 则称 t_2 为 t_1 的规范项.

与第二节中 L' 可归约的定义一致, 在 Σ' 的归约系统 $\mathfrak{R}$ 下, 若 $T(\Sigma', X)$ 中的每个项都有规范项, 则 L' 可归约至 L , 此时称 $\mathfrak{R}$ 是完备的.

定义 3 如果对任意的类子 $s_i \in S$ 和 $T(\Sigma)$ 中任意的以 s_i 为结果类子的两个基项 t 和 t' , 若 $[t]_M = [t']_M$, 则在 $T(\Sigma')$ 亦有 $[t]_{M'} = [t']_{M'}$, 那么称 L' 是 L 的保守扩充.

所谓在 $T(\Sigma)$ 中以 s_i 为结果类子的基项 t , 指的是构造 t 的所有操作必须属于 O , 不得使用 O' 中的操作. 上节中 $\Sigma \subseteq \Sigma'$ 和 M 是 M' 的子代数这两个事实保证了 L' 是 L 的保守扩充, 这是 L' 可归约至 L 的必要条件之一.

为了下面证明中的归纳方便, 象研究抽象数据类型那样[6], 也对基项的形式加以限制, 只考虑最外层操作是 O' 中操作的情况, 这并不妨碍本文所得的结果.

定义 4 令 $L' = L + \langle S', O', \mathfrak{R} \rangle$, 若项 $t \in T(\Sigma')$ 具有形式: $t = f(t_1, \dots, t_n)$, 其中 $f \in O'$, 则称 t 为一个 Σ' 项; 若 Σ' 项 t 中的所有参量 t_i 都属于 $T(\Sigma)$, 称 t 为一个规范 Σ' 项.

定理 令 $L' = L + \langle S', O', \mathfrak{R} \rangle$, L' 是 L 的保守扩充, $\mathfrak{R}$ 中的规则都是正确的, 并且对于每个规则 $lhs \Rightarrow rhs$, 都有 $SV(rhs) \subseteq SV(lhs)$. 那么, 如果对所有的规范 Σ' 项 $f(t_1, \dots, t_n)$, 都满足:

1) 在 $\mathfrak{R}$ 中存在一个规则 $lhs_1 \Rightarrow rhs_1$ 和赋值 $\varphi: SV(lhs_1) \rightarrow T(\Sigma')$, 使得 f

- $(t_1, \dots, t_n) = \varphi^*(lhs_1)$, 并且
- 2) 在 $\mathfrak{R}$ 中存在归约序列 $lhs_1 \xrightarrow{\mathfrak{R}} rhs_2$, 使得 $\varphi^*(rhs_2) \in T(\Sigma)$, 即 $\varphi^*(rhs_2)$ 是 $f(t_1, \dots, t_n)$ 的规范项.

则 $\mathfrak{R}$ 是完备的, 在此归约系统下, L' 可归约至 L .

对此定理我们只需证: 如果对每个规范的 Σ' 项 t 都满足定理中的条件 1) 和 2), 则当 t 是非规范的 Σ' 项时, 同样满足条件 1) 和条件 2). 因此对构成 t 的 O' 操作调用次数进行归纳即可得证.

由上面的定义和定理易证, 完备的归约系统 $\mathfrak{R}$ 具有如下性质:

性质 1 对于 $T(\Sigma', X)$ 中的项 t_1, t_2, t_3 , 若在 $\mathfrak{R}$ 下有 $t_1 \xrightarrow{\mathfrak{R}} t_2, t_1 \xrightarrow{\mathfrak{R}} t_3$, 则必有 $[\varphi^*(t_2)]_M = [\varphi^*(t_3)]_M$, 且在此同余类中必有一个规范项.

性质 2 如果 t_1 和 t_2 都是 t 的规范项, 则必有 $[\varphi^*(t_1)]_M = [\varphi^*(t_2)]_M$.

性质 3 设项代数 $T(\Sigma', X)$ 由 $\langle A(X), F \rangle$ 构成, 其中 $A(X) = \{As_i \cup X_{s_i} \mid s_i \in S \cup S'\}$, 若在 $\mathfrak{R}$ 中有 $t_1 \Rightarrow t_1', t_2 \Rightarrow t_2'$, 且 t_2, t_2' 是 $T(\Sigma', X)$ 中以 s_i 为结果类子的两个项, $x \in X_{s_i}, x \in SV(t_1)$, 则 $t_1[t_2/x] \Rightarrow t_1'[t_2'/x]$ 是正确的.

性质 4 一个语言 L 可归约至自身 L .

性质 5 给定语言 L_1, L_2, L_3 , 若 L_1 可归约至 L_2, L_2 可归约至 L_3 , 则 L_1 可归约至 L_3 .

性质 1 和性质 2 说明: 完备的语言归约系统 $\mathfrak{R}$ 具有语义等价意义下的 Church-Rosser 性质以及规范项的唯一性; 性质 3 表明了归约规则的语义单调性; 性质 4 和性质 5 表明了语言间的可归约关系具有自反性和传递性, Garment 中对语言的实现正是这种可归约关系传递性的应用.

5. 相关的工作

大多数研究者都从比较语言的表达能力出发, 以不同的角度和深度研究了程序设计语言的归约, 语言间的可归约关系可作为比较程序语言表达能力的—个度量依据.

Landin 第一个提出语言比较的形式框架[9], Backus 在其著名的 Turing 奖演讲[8]中以程序代数方式提出了 FP 语言的归约语义, 将 Landin 思想应用于函数式语言中.

Pepper[9]将归约语义发展为变换语义, 研究了 CIP 系统中广谱语言的变换语义, 并称这种变换为定义性变换. 本文的研究借鉴了变换语义的思想和研究方法. 对 Garment 抽象出的语言, 其内容和作用不同于广谱语言.

Felleisen[10]将形式逻辑中定义性扩充这一概念引入程序设计语言领域,从形式语言的角度给出了可表达性和表达能力的形式模型,在行为等价原则下建立了比较语言之间的表达能力的形式基础.[10]中语言成分的可消去性和可表达性与本文给出的可归约概念实质是相同的,但引入这些概念的目的和环境有所不同.

Mitchell[11]以终结语义中的观察等价理论为基础,着重讨论了保持抽象(信息隐蔽)的归约和扩充.其结果与本文中的归约正确性和归约规则的语义单调性一致.最后,Mitchell得出结论:具有抽象机制的语言无法归约至无抽象机制的语言.对于本文中的语义代数 M ,若将其中的语义域定义为一组可观察值,并以终结代数方法研究归约语义,可得出与 Mitchell 一致的结果.

6. 结语

本文用代数方法研究了 Garment 中语言的归约语义,研究的目的是希望能够指导 Garment 的设计,实现与使用.本文中语言的代数模型是抽象数据类型的升华,为研究语言之间的联系建立了一个理论模型,并可望成为研究语言归约的一种统一框架,在语言的代数模型中易于讨论归约语义及其性质.

归约语义的优点在于,它为语言的实现提供了一条可行的途径,在语言中引入新的更复杂的成分不影响已有成分的描述和语义,不需另外引入描述语义的元语言,因此描述出的语义直观,易理解.归约语义的缺点在于,它描述出的语义不象指称语义和公理语义那样有严格的数学基础,但它们之间具有一种相辅相成的关系:如果将指称语义和公理语义视为语言的定义性规范,则归约语义提供了正确的,抽象的实现;如果将归约语义视为语言的定义性规范,则指称语义和公理语义给出了一种有效的证明工具.

归约语义具有广泛的应用.软件开发形式化方法中的过程精化与归约语义非常相似,精化规则与归约规则的含义是一致的,精化的过程就相当于对语言的归约过程;用过程式语言模拟面向对象语言[12];C语言到 λ -演算的变换[13];以及程序变换系统中的变换规则[5][14]都是归约语义的应用.

关于语法的保守扩充,类型系统的正交性扩充以及用类型系统的归约实现数据精化过程是进一步的研究问题.

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 (吴文俊主编) Vol.2 (1994), 54-64.
- [2] 张乃孝, 郑红军, 裘宗燕. Garment—A Mechanism for Abstraction and Encapsulation of Languages 《ACM SIGPLAN Notices》 Vol.32. No.6 p53-60 June,1997
- [3] 张乃孝. 程序变换在程序语言中的一种表示 — 兼论变换型语言. 软件学报 4, 5 (1993), 17 - 23.
- [4] 张乃孝. 程序变换过程的分析与设计. 计算机学报 17, 6 (1994), 473-476.
- [5] 屈婉玲, 张乃孝. 用变换型方法模拟开发电话交换系统. 计算机研究与发展 32, 7(1995), 11-16.
- [6] 陆汝钤. 计算机语言的形式语义. 科学出版社, 1992.
- [7] Zheng Hongjun and Zhang Naixiao. An Abstract Model for Programming Languages. In: Keijiro Araki and Keqin He, *Proc. of ChangSha International CASE Symposium'95*, P69-75, ChangSha, China.
- [8] John Backus. Can Programming be Liberated from the VonNeumann Style? A Functional Style and Its Algebra of Programs. *CACM* 21, 8(1978), 613-641.
- [9] Peter Pepper. A Study on Transformational Semantics. In: G. Goos and J.Hartmanis Edited. *LNCS 69* Springer Verlag (1979), 322-405.
- [10] Matthias Felleisen. On the Expressive Power of Programming Languages. In: G. Goos and J.Hartmanis Edited. *LNCS 432* Springer Verlag LNCS (ESOP'90), 134-151.
- [11] John C. Mitchell. On Abstraction and the Expressive Power of Programming Languages. *Science of Computer Programming* 21 (1993), 141-163.
- [12] Jennifer Hamilton. A Model for Implementing and Object-Oriented Design without Language Extensions. *ACM SIGPLAN Notices* 13, 1(1996), 36-43.
- [13] Michael Karr. A Translator from C to a λ -Calculus Representation. *SIG SOFT SE Notes* 21, 4(1996), 28-31.
- [14] Helmut.A.Partsch. Specification and Transformation of Programming. *Springer-Verlag*, 1990.

The Reduction Semantics in Garment

Zheng Hongjun Zhang Naixiao
(Peking University)

Abstract The reduction semantics for programming languages in Garment are considered with algebraic approach in this paper. This paper first explains, in formal language theory, the meaning of the reduction semantics. An algebraic model for programming languages in Garment is proposed then. By the algebraic model, the paper discusses the reduction semantics and some properties, and finally proves the sufficient conditions for reducible languages.

Keywords Garment Programming language Reduction semantics Signature
Term algebra