程序设计语言的抽象与语言族模型[+]

张乃孝　　郑红军

（北京大学·计算机科学技术系　北京　100871）

摘要　　程序设计语言的模型对于研究语言的性质具有重要作用。本文基于语言的抽象这一概念，在本文建立的语言之代数模型下，给出了程序设计语言间的三种关系：继承、扩充、屏蔽的语义，并提出了在这三种关系下构成的语言族模型，作为研究语言间的关系和面向模型的变换型软件开发方法的一种理论基础。

关键词　程序设计语言　　语言抽象　　语言族模型

Abstraction of Programming Languages and a Model of Language Family

Zhang Naixiao　　Zheng Hongjun

Department of Computer Science & Technology
Peking University, Beijing 100871

Abstract　Models for programming languages play an important role for studying properties of programming languages. Based on the concept of language abstraction and under an algebraic model for programming languages we build, this paper presents the semantics of three kinds of relation between languages, i.e., inheritance, extension and shielding. Then the paper proposes a model of language family which is constructed under the three kinds of relation. This model can be a theoretical basis for researching relations between languages and model-oriented transformational software development methodology.

Keywords　programming language　　language abstraction　　language family model

1 引言

随着计算机科学的发展，不断地出现各种新的程序设计语言。语言的不同是语言设计原则的选择和各原则相对优先级不同的结果，但所有语言的目的基本上是相同的，即：能够使程序员易于在计算机中建立客观事物的模型。从语言的历史来看，新语言的出现主要是由于当前所有语言都不易于描述待处理的某类客观事物，这样，人们不断地设计、开发他们所需要的语言，如：数据库语言、数学计算语言、图形绘制语言等等，而这些语言在风格、内容和形式上都各有明显的不同之处。语言种类的繁杂性和语言之间的显著差别，不仅增加了语言的使用难度，也给研究语言之间的联系带来了诸多不便[7]。程序语言的设计者们曾试图通

[+] 本项研究得到国家自然科学基金和 863-306 课题的资助.

过结合不同风范于同一语言这种方式来缓解这些问题，但目前很难看到这种结合的成功迹象[6]。若能对程序设计语言取得更广泛、更普遍的理解，并能在语言设计者和使用者选择语言设计原则的问题上取得一致，也许会找到解决这些问题的一个途径。

本文的工作是在研究Polya语言[++] 及面向模型的变换型软件开发方法[14]的基础上，通过对程序设计语言本质的研究，提出了语言的抽象这一概念和语言的一种抽象模型，并以此为基础，给出了语言的继承、扩充、屏蔽的语义及语言族的语义。希望本文提出的程序设计语言的抽象模型能够对研究语言之间的关系有所益处，并能建立其研究的理论基础；也希望能够为程序设计语言的设计与开发提供一个可实现的理论模型。此理论模型的实现可以使得语言的使用与语言的定义分离、语言的开发与程序的开发分离成为可能，从而有可能提高语言本身的简明性，达到降低软件开发复杂度的目的，为解决由当前软件技术的局限性所引起的问题提供一条可行的途径。

2 语言的抽象

用代数方法描述程序设计语言，能够使得程序设计语言具有清晰性、层次性和可操纵性[2]，是对研究程序设计语言之间的联系的一个有益尝试(提供语言的一个抽象模型)。具体的描述方法是：用基调描述语言的抽象语法[8]， 一个基调$\sum=(S,F)$由它的类集S(语言的基本语法元素)和操作集F(语法元素间的组合关系)两部分组成。通过引进一组语义方程为基调$\sum$指定语义，也即为基调$\sum$所对应的语言指定语义，不同的语义方程为其指定不同的语义。这样，基调与引进的语义方程构成语言的一个抽象模型，语言的这种抽象模型独立于一个语言的具体表示。许多研究者[1][10][12]都曾以不同的方式用基调描述了语言的抽象语法。

用基调$\sum=(S,F)$描述语言的抽象语法的一般原则是:S中的每个类对应于语言的一个基本语法元素——BNF中的非终结符；F中的一个操作对应于基于语法元素间的组合关系。这样，一个BNF产生式$P: s_0::=t_1s_1t_2s_2 \ldots t_ns_nt_{n+1}$ (其中，t_i为终结符号或空串 ε ，s_i为非终结符)就对应于基调$\sum$中的一个操作$P: s_1\times s_2\times\cdots\times s_n\rightarrow s_0\in F$ ($s_i\in S,\ 0\le i\le n$)。

考虑一个如下的小型类Pascal语言，我们称之为核心语言K：

```
prog ::= BEGIN stmt END
stmt ::= stmt ';' stmt
       | IF expr THEN stmt ELSE stmt FI
       | WHILE expr DO stmt OD
       | vid:=expr | skip
expr ::= int | vid
       | expr '+' expr | expr '*' expr | NOT expr
```

[++] Polya 语言是 Cornell 大学的 D. Gries 教授提出的.

我们先非形式地看一下K的部分语义。在K中，只有整数类型的表达式，布尔类型可以由整数类型实现: 以非0整数表示真值，以0表示假值。*eval*: EXPR→INT用来表示对K中表达式求值的函数，可按如下方式将整型数作为布尔值来计算：

$$eval(\underline{\textbf{NOT}}\ \text{expr}) = \underline{\textbf{LET}}\ v = eval(\text{expr})\ \underline{\textbf{IN}}\ v = 0 \rightarrow 1, 0$$

于是，**IF** expr **THEN** stmt_1 **ELSE** stmt_2 **FI**的语义可非形式地定义为：

$$eval(\text{expr}) = 0 \rightarrow \text{stmt}_2, \text{stmt}_1$$

根据上述描述抽象语言的一般原则，我们可以用基调$\sum_K=(S_K, F_K)$描述K的语法，其中：

S_K = { int, vid, stmt, expr, prog } ;

F_K = {

program : stmt→prog ;	comp : stmt×stmt→stmt ;
if : expr×stmt×stmt→stmt ;	while : expr×stmt→stmt ;
assign : vid×expr→stmt ;	skip : →stmt ;
int_expr : int→expr ;	id_expr : vid→expr ;
add : expr×expr→expr ;	mul : expr×expr→expr ;
not : expr→expr ;	}

在此，终结符号被认为是非本质的，它们可以被*"忘却"*，因为它们可由操作名P唯一确定[2]。由此"忘却"原则可以看出，基调$\sum$中不仅类集合是抽象的，而且基调中的操作表示也是抽象的，因此，利用上述原则构造的语言L的基调$\sum_L$，可称之为语言L的抽象语法；构造基调$\sum_L$的过程即是语言L的语法抽象过程，此过程类似于[1][11]中所述的过程。$\sum_L$中的类体现了抽象语法的分析性，$\sum_L$中的操作体现了抽象语法的综合性。

为了给基调$\sum_L$指定语义，还需要引入一组描述$\sum_L$语义的语义方程——$\sum_L$等式。$\sum_L$与$\sum_L$等式集合E_L一起构成了一个语言$(\sum_L, E_L)$。不同的E_L为$\sum_L$指定不同的语义。

我们采用变换语义[2][8]的方法为$\sum_L$指定语义，将$\sum_L$等式视为变换规则。所谓变换语义是指，假定语言L_n的语义已知，以变换规则的形式将L_n的成分用来描述一个新语言L_{n-1}的每个成分，于是即可得到语言L_{n-1}的语义[8]，我们称语言L_{n-1}为L_n的抽象语言，语言L_n为语言L_{n-1}的具体语言，亦称实现语言[13]。若继续上述过程，则可以得到一个语言序列：$L_n, L_{n-1}, ..., L_1, L$，在这个语言序列中，抽象语言与具体语言是相对的，随着语言在上述语言序列中所处的位置不同，抽象语言可以为具体语言，同样，具体语言也可为抽象语言。例如，在上面的语言序列中，若将语言L作为抽象语言，则语言$L_n, L_{n-1}, ..., L_1$均为其具体语言；若将语言L_1作为抽象语言，则语言$L_n, L_{n-1}, ..., L_2$均为其具体语言，如此等等。称序列中最具体的语言L_n为核心语言，序列中的其他语言均为中间语言。如果L_n是一种机器语言，则上面的序列表示的就是一个语言的编译过程，即由高级语言到机器语言的变换实现过程。

定义1 一个$\sum_L$等式是一个二元组<*lhs*,*rhs*>，通常写成*lhs*=*rhs*，其中，*lhs*是基项代数$T(\sum_L)$[8][4]中的项，*rhs*是基项代数$T(\sum_{L"})$中的项，$L"$是抽象语言L的一个具体语言。

如：假定$\sum_K$等式集合E_K已为上述核心语言K指定了语义，我们可以用$K=(\sum_K, E_K)$说明一个语言$M=(\sum_M, E_M)$，其中：

$S_M = S_K$;

$F_M = F_K$+{ and_expr : expr×expr→expr ;
or_expr : expr×expr→expr ;
repeat : stmt×expr→stmt }

$E_M=E_K$+{ and_expr(expr, expr) = mul(expr, expr) ;
or_expr(expr, expr) = add(expr, expr) ;
repeat(stmt, expr) = comp(stmt , while(not(expr) , stmt)) }

在[5]中，语言M中新增加的语法结构*and_expr*, *or_expr*和*repeat*被称为可表达结构，即：上面三种语法结构的语义在K中是可表达的，正如E_M中所示。以Felleisen[5]的研究结果来看，M是K的一个定义性扩充，这种扩充是本文第三部分的一个基础。另外，与扩充相反，我们还能以另一种方式定义一个新语言$N=(\sum_N, E_N)$，其中：

$S_N=S_K$;
$F_N=F_K$－{ add:expr×expr→expr } ;
$E_N=E_K$－{ *eval*(expr '+' expr) } ;

这就是本文第三部分将要讨论的语言的屏蔽问题。

由于$\sum_L$和$\sum_{L"}$分别表示的是语言L和$L"$的抽象语法，故基项代数$T(\sum_L)$和$T(\sum_{L"})$中的项分别是符合抽象语法$\sum_L$和$\sum_{L"}$的有限生成的项，即由$\sum_L$和$\sum_{L"}$中的操作符号构造出的项。在此，E_L中的等式即是语言L的变换语义描述中的变换规则，它们封闭于等价(自反，传递，对称)和替换关系下的语义(等价)推理。在保持语义一致性的前提下，等式*lhs=rhs*意为可将语言L的语句*lhs*变换为语言$L"$的语句*rhs*。

定义2 一个语言$L=(\sum_L, E_L)$由一个基调$\sum_L$和一个$\sum_L$等式集合E_L组成，定义$(\sum_L, E_L)$为程序设计语言L的一个抽象模型。

由这里给出的语言的抽象模型可以看出，一个语言L的语法可以抽象为类的集合及在各类上的操作，即基调$\sum_L$；其语义可以抽象为$\sum_L$等式集合E_L。这样，一个语言L即可抽象为如定义2所述的抽象模型$(\sum_L, E_L)$，包括语法抽象和语义抽象。语言的上述抽象模型为本文下面研究语言之间的联系提供了一个理论框架。

为了下面描述方便，我们在S_L, F_L, E_L中分别分别引入一个未定义元素⊥，并假设$S_L \cap F_L \cap E_L=\{\perp\}$。

3 语言的继承、扩充和屏蔽 —— 语言族

下面，我们将讨论由语言L通过继承、扩充和屏蔽构造新语言L'的语义。L与L'之间的关系是具体语言与抽象语言的关系。

定义3 令$\sum_L=(S_L, F_L)$和$\sum_{L'}=(S_{L'}, F_{L'})$分别是语言$L$和$L'$的基调，定义从$\sum_L$到$\sum_{L'}$基调射 σ : $\sum_L \to \sum_{L'}$为对偶(h,g)，其中：h: $S_L \to S_{L'}$是S_L到$S_{L'}$内的映射，且$h(\perp)=\perp$；g: $F_L \to F_{L'}$是从F_L到$F_{L'}$的一个部分同态，且$g(\perp)=\perp$，对$s_i \in S_L$ $(0 \le i \le n)$及一个操作P: $s_1 \times s_2 \times \ldots \times s_n \to s_0 \in F_L$，定义

$$g(P: s_1 \times s_2 \times \ldots \times s_n \to s_0) = \begin{cases} \perp & \exists s_i\ h(s_i) = \perp \ \text{or}\ P \notin F_{L'} \\ P: h(s_1) \times h(s_2) \times \ldots \times h(s_n) \to h(s_0) & \end{cases}$$

若h,g均为恒等射，则基调射σ为恒等射;若h,g均为单射，则基调射σ为单射;若h,g均为满射，则基调射σ为满射。

定义4 **令**$L=(\sum_L, E_L)$**和**$L'=(\sum_{L'}, E_{L'})$**是两个语言，如果σ是一个基调射**: $\sum_L \to \sum_{L'}$**，并且对于**E_L**中的任一等式**$lhs=rhs$**，等式(**$\sigma(lhs)=rhs$**)**$\in E_{L'}$**，则同时称σ为一个语言射**: $L \to L'$**。**

定义4**中，**lhs**是基项代数**$T(\sum_L)$**的项，**rhs**是基项代数**$T(\sum_{L''})$**的项，其中**L''**是**L**和**L'**的具体语言，以本文第二部分中的语言序列来表示语言**L**，**L'**和**L''**间的关系，即为：**L''**，**L**，**L'**。基项是由**$\sum_L$**中的类及类上的操作构成，故σ(**lhs**)意指,将**lhs**中的类及类上的操作映射到对应于**$\sum_{L'}$**中的类及类上的操作。设**t**是**lhs**中的一个类或一些类上的某个操作，定义**: σ(t)=⊥**,当且仅当**$h(t)=\perp$**或**$g(t)=\perp$**；如果σ(**t**)=⊥,则σ(**lhs**)=⊥；如果σ(**lhs**)=⊥；则等式(σ(**lhs**)=**rhs**)**$\in\{\perp\}$**。**

对于本文第二部分中的语言K**和**M**，根据定义**3**， 存在一个**$\sum_K$**到**$\sum_M$**的基调射**$\sigma_1=(h_1, g_1)$**，其中**h_1**和**g_1**都是恒等射，故**σ_1**是恒等射。注意到，**h_1**是满射且为单射，因为**$S_M=S_K$**；而**g_1**却不是满射，因为**$F_K \subseteq F_M$**。所以，**σ_1**不是满射；根据定义**4**，**σ_1**也是一个语言射：**$K \to M$**。另外，对于语言**K**和**N**，也存在一个**$\sum_K$**到**$\sum_N$**的基调射**$\sigma_2=(h_2, g_2)$**，其中**h_2**是恒等射：**$S_K \to S_K$**，**g_2**是一个满射，因为操作**$add \notin F_N$**，即**$F_K \subseteq F_N$**，而且**

$$g_2(\text{add: expr} \times \text{expr} \to \text{expr}) = \perp; \quad g_2(\perp) = \perp;$$

所以，g_2**是满射但不是单射，故**σ_2**是满射但不是单射。**

事实上，定义3**和定义**4**中的基调射与语言射是**[11]**中可交换图的一个特例。我们可以将此图根据这两个定义简化为图**4.1**，其中**sem_L**和**$sem_{L'}$**分别为**L**和**L'**的变换语义，**syn_L**和**$syn_{L'}$**分别为语言**L**和**L'**的语法。更详细的信息，参见**[11][12]**。**

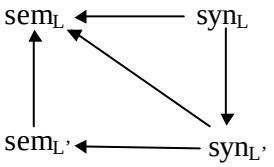

图 4.1

不失一般性，若基调射σ: $\sum_L \to \sum_{L'}$**是单射且亦是满射，则其语言射σ**: $L \to L'$**表示的是语言**L**与**L'**间的语义继承关系，语言**L'**完全继承了语言**L**，正如恒等的语言射**: σ_0: $K \to K$**；若基调射σ**: $\sum_L \to \sum_{L'}$**是单射但不是满射，则其语言射σ**: $L \to L'$**表示的是语言**L**与**L'**间在语义继承意义下的扩充关系，语言**L'**是在语义继承意义下语言**L**的扩充语言和定义性扩充，如语言射**σ_1**所示，**M**是**K**的一个定义性扩充；若基调射σ**: $\sum_L \to \sum_{L'}$**是满射但不是单射，则其语言射σ**: $L \to L'$**表示的是语言**L**与**L'**间的语义屏蔽关系，语言**L'**在语义上屏蔽了语言**L**的某些成分结构,如在**σ_2**中，语言**N**屏蔽了**K**的**add**操作。**

定理　以语言为对象，语言间的射为射，构成一个范畴[4]AL**。**

证明: (1) **恒等射的存在性**.

令语言$L=(\sum_L, E_L)$

显然，存在基调$\sum_L$**上的恒等射σ**: $\sum_L \to \sum_L$**（如**σ_0**）**

于是，对E_L**中的任一等式**$lhs=rhs$**，有：**

σ(lhs**)=**lhs**，且等式(**$lhs=rhs$**)**$\in E_L$

所以，等式($\sigma(lhs)=rhs)\in E_L$，由定义4，
基调射$\sigma:\sum_L\to\sum_L$亦是语言射:$L\to L$，且为恒等射。

(2) 语言射的复合仍是语言射.
设语言$L_1=(\sum_{L1},E_{L1})$，$L_2=(\sum_{L2},E_{L2})$，$L_3=(\sum_{L3},E_{L3})$，
且有语言射$\sigma_1:L_1\to L_2$，$\sigma_2:L_2\to L_3$，则由定义4，
存在与这些语言射相对应的基调射$\sigma_1:\sum_{L1}\to\sum_{L2}$，$\sigma_2:\sum_{L2}\to\sum_{L3}$，
这样，即可根据定义3构造基调射σ_1和σ_2的复合

$$\sigma_3=\sigma_1\cdot\sigma_2:\sum_{L1}\to\sum_{L3}$$

σ_3的形式为$<h_1\cdot h_2\,,\,g_1\cdot g_2>$，其中，$<h_1\,,\,g_1>$，$<h_2\,,\,g_2>$分别是$\sigma_1$和$\sigma_2$的表示，“·”是射的复合操作。
由定义4可得，

对于任一等式$(lhs=rhs)\in E_{L1}$，等式$(\sigma_1(lhs)=rhs)\in E_{L2}$
若等式$(\sigma_1(lhs)=rhs)\notin\{\perp\}$，则等式$(\sigma_2(\sigma_1(lhs))=rhs)\in E_{L3}$，
即：等式$(\sigma_3(lhs)=rhs)\in E_{L3}$；
若等式$(\sigma_1(lhs)=rhs)\in\{\perp\}$，则等式$(\sigma_2(\sigma_1(lhs))=rhs)\in\{\perp\}$，
即，等式$(\sigma_3(lhs)=rhs)\in\{\perp\}$
由上可得，对于E_{L1}中的任一等式$lhs=rhs$，等式$(\sigma_3(lhs)=rhs)\in E_{L3}$，
由定义4，σ_3为语言射:$L_2\to L_3$，也即语言射的复合仍是语言射。

由(1)(2)及范畴的定义[4]可知：

以语言为对象，语言间的射为射，构成一个范畴。 □

在范畴AL中，一个有基语言由一个语言L和所有指向L的射$\{L_i\to L\}$组成，表示为$<L,\{L_i\to L\}>$，$L_i\to L$ 表示L_i与L间的语言射，L_i称为语言L的基语言。

一个有基语言可以有多个基语言，其中某些基语言也有可能是有基语言，即某些有基语言可能以另一些有基语言为其基语言，如图4.2所示。

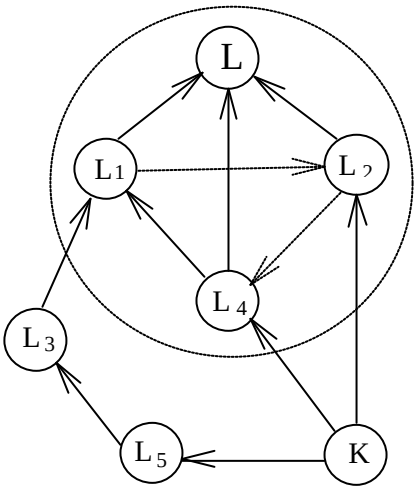

图 4.2

在此图中，L_1，L_2，L_4是L的基语言，有基语言L有三个基语言L_1，L_2，L_4，而基语言L_1，L_2，L_4本身又是有基语言，图中的所有语言构成了继承、扩充、屏蔽关系下的语言族模型。L_1，L_2之间、L_2，L_4之间可以根据定义3和定义4利用未定义元素⊥及[12]中的变换T_1来构造语言间的上述三种关系，实际语言中可能没有这些关系，故用虚箭头表示。由这三种关系构造的语言族具有层次性，每个层次上的语言都有其自身的表达抽象度，语言的层次越高，其表达抽象度越高，语言的描述能力越强。最底层的基语言(如图4.2中的K)称为核心语言[14]，即为最具体的语言。在核心语言的语义是已知的假设下，语言族中所有的语言都是由核心语言经继承、扩充、屏蔽而得到的。从范畴论的角度来看，一个有基语言(如图4.2中的L)由另一些基语言(如图4.2中的L_1，L_2，L_4)加上彼此之间的射构成，它们在一起构成了一个共锥[4]。图4.2中的语言L，L_1，L_2，L_4及其它们之间的射构成了一个共锥，这是因为它们满足：

$$(L_2 \to L)\cdot(L_1 \to L_2) = L_1 \to L$$
$$(L_4 \to L)\cdot(L_2 \to L_4) = L_2 \to L$$
$$(L_1 \to L)\cdot(L_4 \to L_1) = L_4 \to L$$

由上面三式可以看出，L、L_1、L_2，L、L_2、L_4，L、L_1、L_4均具有可交换关系[4]，其中，“·”是射的复合操作。语言族中，共锥的含义表明，语言的开发有多种途径。仍以图4.2为例来讨论开发语言L的多种路径，我们可以直接由基语言L_4以继承、扩充、屏蔽三种方式开发出语言L；亦可由L_4仍以上述三种方式开发出语言L_1，然后再以基语言L_1为基础以上述三种方式开发出语言L。语言开发方法及途径的多样性之意义在于：以一种语言L_i(如图4.2中的L_1，L_2，L_4)为基础，可以直接得到与L_i描述能力及方式不同的语言L，亦可得到描述能力介于语言L_i和语言L之间的多种不同描述能力及方式的语言，而且这些语言均是基于核心语言(如图4.2中的K)而开发出来的。这样，语言的使用者亦可有多种选择，可以根据需要选择适合于特定领域表达能力和表达方式的语言以解决其领域问题，从而可以解决本文第一部分中所提到的程序设计语言种类的繁杂性和语言间的显著差别问题。

4 语言的抽象是数据抽象的升华

抽象数据类型[3]是数据抽象发展的一个新阶段，一个抽象数据类型A的语法亦可用基调来描述，我们称此基调为$\sum_A$，基调$\sum_A$中的类表示数据类型，其中的操作表示数据类型之间的操作关系；而对于语言L的基调$\sum_L$,其中的类表示语言L的语法元素，操作表示语法元素间的组合关系。从$\sum_L$和$\sum_A$所表达的内容及层次上看，$\sum_L$是$\sum_A$的升华：$\sum_L$描述的是语言L的语法，$\sum_A$描述的是抽象数据类型A的语法。

同样，也可用一个等式集合E_A(公理序列)为抽象数据类型A指定语义。在E_A中，等式$lhs=rhs$表示抽象数据类型A中操作及数据类型间的逻辑关系，其中lhs和rhs均是项代数[8]中的项(对数据类型的有限操作)。等式集合E_L中的等式表示语义等价关系下语言间的变换关系(抽象语言与具体语言间的关系)。易见，E_L与E_A中的等式所使用的对象层次及所描述的关系性质不同。

程序设计语言可以视为具有层次结构的类型(抽象数据类型)。这样，语言的上下文无关文法对应于类型的基调；语言结构的上下文环境(条件)可由完全布尔项来表示；语言的语义由一组条件等式来说明[2]。以这种方法，我们可以得到一个程序设计语言完整的规范。不过，在语言的这种描述方式下，很难表示清楚语言之间的关系，原因就在于，语言对应于具有层次结构的类型这一观点增加了语言表示的复杂度。由本文第三部分中的语言抽象和由此而得的抽象模型，我们可以清楚地看出语言间的继承、扩充和屏蔽关系。在这三种关系下，语言构成了如上所述的语言族及范畴AL。事实上，语言族是以层次结构方式组织的，这种层次结构是抽象数据类型层次结构的升华。

因此，语言L的抽象$(\sum_L, E_L)$与数据类型A的抽象$(\sum_A, E_A)$相比较，语言的抽象是数据抽象的升华。

5 相关的工作及结语

一些研究者曾经讨论过语言的模型及其相关的问题。Milner[9]提出了带类型的λ-演算的一个完全抽象的模型，并表明：在某些情况下，对于一个扩充的带类型λ-演算存在一个完全抽象的模型，包括代数模型。我们亦可期望：对于程序设计语言，存在一个我们所提出的抽象模型。Pair[10]和Broy等人[2]研究了程序设计语言的代数语义，他们都将基于抽象数据类型的程序设计语言的代数语义作为其研究重点。Goguen曾建议应根据代数理论来研究程序设计语言，Broy等人在此建议下，描述、分析和对比了变换语义和弱终结语义的一些性质。我们将变换语义与抽象数据类型理论相结合用于描述程序设计语言。Rus[11][12]从语言实现的角度，给出了以BNF为中心的程序设计语言的代数模型，在其工作中，未讨论在其模型中如何描述语义，另外，在一些情况下(如本文的未定义情况⊥)，Rus所定义的语言的一致关系可能不存在。Felleisen[5]为比较语言的表达能力提供了一个形式框架，他以Scheme语言为例，讨论了Scheme的保守和定义性扩充。本文的一个目的是希望在本文的模型下应用和解释其研究结果。在程序设计语言的变换语义下，本文第三部分中的射 $\sigma_1: K \to M$ 在此模型下即为语言K的一个保守和定义性扩充，而对于象 $\sigma_2: K \to N$ 的射，在他的结果中很难解释其含义，因为Felleisen只讨论了语言的扩充情形，而语言的扩充和屏蔽在某种程度上毕竟还是有一些差别的。

本文所提出的语言的抽象是数据抽象的升华，它为研究语言之间的联系建立了一个理论模型。由核心语言开发出的语言族是支持面向模型的变换型软件开发方法的有利的语言工具及支持环境，根据上面提出的语言的抽象模型，我们在研制语言开发和程序开发双重环境Polya-BD[13][14]过程中，又进一步提出了一种语言的抽象与封装机制Garment[15]，它是本文所提出的语言的抽象模型的一个实现。利用这种机制可以将抽象出的语言封装于Garment中以能实现语言的继承、扩充、屏蔽，并可形成由面向不同领域的语言构成的语言族。语言的使用者可以根据需要在语言族中任何一个层次上工作，以易于开发出结构清晰、有效、正确、可靠、易维护的软件。利用这种语言的抽象与封装机制，易于语言的扩充和实现，也使得语言的开发具有层次性、系统性。另外，核心语言的充分性、语言变换的正确性、语言变换的语义以及语言扩充的充分完备性、一致性等问题是有待于进一步研究的问题。

参考文献

[1] Bjorner, D. Toward the Meaning of VDM 'M'. *LNCS* 352; TAPSOFT'89: 1—35; 1989.

[2] Broy, M. *et al.* On the Algebraic Definition of Programming Languages. *ACM Transaction on Programming Languages and Systems* 9(1): 54—99; 1987.

[3] Cardelli, L. and Wegner, P. On Understanding Types, Data Abstraction, and Polymorphism, *ACM Computing Surveys* 17(4): 471—522; 1985.

[4] **陈意云**. ***计算机科学中范畴论***. **中国科技大学出版社**; 1994.

[5] Felleisen, M. On the Expressive Power of Programming Languages. *LNCS* 432; ESOP'90: 134—151; 1990.

[6] Hoare, C.A.R. The Varieties of Programming Languages. *LNCS* 351; TAPSOFT'89: 1—18; 1989.

[7] Horowitz, E. *Fundamentals of Programming Languages*. 1989.

[8] **陆汝钤**. ***计算机语言的形式语义***. **科学出版社**; 1994.

[9] Milner, M. Fully Abstract Models of Typed Lambda-Calculi. *Theoretical Computer Science* 4(1): 1—22; 1977.

[10] Pair, C. Abstract Data Types and Algebraic Semantics of Programming Languages. *Theoretical Computer Science* 18: 1—31; 1982.

[11] Rus, T. An Algebraic Model for Programming Languages. *Computer Languages* 12(3/4): 173—195; 1987.

[12] Rus, T. An Algebraic Tool for Language Processing. *Computer Languages* 20(4): 213—238; 1994.

[13] **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): 17—23; 1993.

[14] **张乃孝 等**. **面向模型的变换型软件开发方法研究**. ***理论计算机科学*** 2: 54—64; 1994.

[15] Zhang, Naixiao and Zheng, Hongjun. Garment — A New Mechanism of Abstraction and Encapsulation. *Technical Report*; 1996.