11-17 线性逻辑 逻辑程序设计 ②

计算机科学1992 Vol.19 №.6

线性逻辑和逻辑式程序设计

黄林鹏 孙永强（200030 上海交通大学计算机系） TP302.2

摘 要

所谓的逻辑式程序设计它涉及两个逻辑层次：外部逻辑和内部逻辑，前者描述对象之间的逻辑关系，而后者涉及目标求解遵循的逻辑法则。如PROLOG，它的外部逻辑是古典逻辑，而其计算遵循的，我们将证明可以使用线性逻辑中的记法加以刻画。本文从证明论角度出发，研究PROLOG目标求解成功或失败的公理化，并说明对任一给定的程序P，可定义一个线性理论LT(P)，关于该理论，PROLOG目标求解是正确且完备的。

1. Gentzen 定理和消解法

Gentzen 的cut消去定理，或称Hauptsatz，是证明论中最为重要的结果之一，它和Robinson 的消解法有着密切的关系。

Gentzen 定理说明，如果$\vdash B$是公理$\Gamma_i \vdash \Delta_i$的逻辑推论，则存在一个以原公理$\Gamma_i \vdash \Delta_i$的实例$\Gamma_i' \vdash \Delta_i'$为公理的B的证明，称为范式证明，在该证明中任一 cut 规则的cut公式必在上述某一实例化的公理中出现。（注：这里的 cut 规则和 PROLOG中截断的概念没有任何关系，在古典逻辑中，它的形式为

$$\frac{\Gamma \vdash C,\ \Delta \quad \Gamma',\ C \vdash \Delta'}{\Gamma,\ \Gamma' \vdash \Delta, \Delta'}\ \text{cut}$$

其中公式C称为 cut 公式）

若所有公理的形式皆为原子直觉主义矢列，即形为

$$A_1,\ \cdots,\ A_n \vdash B$$

（这种特殊形式的矢列，就是我们所熟悉的Horn子句$B:-A_1, \cdots, A_n$的矢列记法）。注意到在范式证明中，cut 公式必在某实例化的公理中出现，而这些公理中出现的公式均是原子公式，因此逻辑规则不可能被使用，这是因为它们引入的逻辑符号无法消去。这样在范式证明中，我们仅可使用下述规则：

1）实例化的公理（包括逻辑公理和非逻辑公理，在 PROLOG 中非逻辑公理被称为规则）；

2）cut 规则；

3）弱规则（weakening）；

4）缩规则（contraction）；

5）交换规则（exchange）。

事实上，若我们的推导仅限于 Horn 子句（即原子直觉主义矢列），则结构规则（即规则3、4、5）的使用也可以去除，在给出形式叙述之前，我们先作一直观的说明。

·弱规则 若我们用$\Gamma, A \vdash C$代替$\Gamma \vdash C$，因为目标矢列$\vdash B$的左边为空，因此 A 的使命就是在某个时刻由于$\vdash A$的证明而通过 cut 规则消去，即A的出现没有任何意义，弱规则的使用是多余的。

·缩规则 如果我们用$\Gamma, A \vdash C$代替$\Gamma, A, A \vdash C$，同弱规则一样，$\Gamma, A, A \vdash C$中出现的A，也必将在某个时刻由于$\vdash A$的证明而由 cut 规则消去，显然$\Gamma, A, A \vdash C$中出现的A也可以通过两次 cut 规则的使用而消去。

事实上，我们有下述引理：

引理1 若原子矢列$A \vdash B$使用规则1），…，5）可证，则存在一个不使用规则3）、4）的原子直觉主义矢列$A' \vdash B'$的证明，其中A'由A中公式构造而得，而B'是B中一公

式。

证：对A⊢B的证明π进行归纳。

若我们应用上述引理于目标矢列⊢B时，有A′为空，B′＝B，进一步我们可以看到交换规则的使用可以通过 cut 规则的适当排列而消去。逻辑公理的使用

$$\frac{A\vdash C \quad C\vdash C}{A\vdash C}\ \text{cut}$$

显然是多余的，因此，我们有

推论 若⊢B是一个以原子直觉主义矢列（即 Horn 子句的矢列表示）为公理的逻辑推论，则存在B的一个证明，在其中我们仅需使用非逻辑公理的实例及 cut 规则。

Robinson 的消解法，就是寻找这样证明的一种策略，它的基本思想就是尝试所有 cut 和由合一限制的代换的可能组合，它可以被认为是使用规则1)、2)的自动证明。

2．目标求解成功和失败的合理化

一般地说，所谓的逻辑式程序设计它涉及两个逻辑层次：(a) 外部逻辑 (external logic)它描述事物之间的逻辑关系，如建立在一阶谓词逻辑上的PROLOG语言，它的外部逻辑是古典逻辑；b)内部逻辑(internal logic)，它涉及逻辑程序目标求解所遵循的逻辑法则。内部逻辑和外部逻辑是不同的两个概念，若把它们等同起来将导致没有意义的结果。

在古典逻辑中有16种逻辑连接符，而只有一种和合取行为有关，即我们所熟悉的满足交换律和幂等律的布尔合取。但PROLOG内部逻辑的合取也如此吗？考虑

子句1：　　A，B⊢C

子句2：　　B，A⊢C

（假设子目标按书写顺序从左到右求解），当求解子目标A失败，而求解子目标B既不成功也不失败即循环时，子句1失败，而子句2循环。若把古典逻辑也看成 PROLOG的内部逻辑，因为合取的可交换性，因此避免上述出现的矛盾的最直接方式就把“失败”(failure) 和“不成功” (absence of success) 看成是一样的。由此引入不可证性(unprovability)，而所谓的封闭世界假设就来源于内部逻辑和外部逻辑的等同。因此，Girard 说：从证明论角度看“如果A不能被证明则非A”，是没有计算意义的，而所谓的缺省推理 (default reasoning) 应被称之为“deficient reasoning”[2]。

事实上，失败不等于不成功，求解目标失败是一个积极的信息，它表示我们遵循一个特定的操作模式（搜索过程）直到结束，这显然不同于求解目标不成功，因为后者可能是由于其过程中包含了循环造成的。因此，如果我们能确切地使用某种内部逻辑表示计算所遵循的模式，则就有可能把失败解释为某个语句的证明，而可证性是一个我们能机械验证的性质，因此求解目标失败就有了其计算上的含义。这样，如果我们用非A表示求解目标A失败（当然这里的非不是古典逻辑意义下的非），则所谓的“否定解释为失败”则是正确的。为了简单，我们仅讨论命题 PROLOG 语言中目标求解成功或失败的公理化。我们的目标就是要证明类似如下的陈述：

目标A求解成功　当且仅当A可证；

目标A求解失败　当且仅当非A可证。

要注意的是，对一般的 SLDNF 的公理化和对PROLOG(它是SLDNF的一种实现)的公理化的结果是不同的。

在SLDNF中，求解目标A成功指存在一个回答A为真的搜索，而（标准）PROLOG的搜索方式是固定的。考虑关于合取命题A and B 求解失败的例子：

若A失败或B失败，则A and B 关于 SLDNF 失败；

若A失败或A成功但B失败，则A and B 关于 PROLOG 失败。当A循环而B失败时，A and B 关于 SLDNF 失败，而关于PROLOG循环。

一般地说，对“成功”和“失败”的公理化

方法有两种，一是非逻辑公理化，它仅考虑目标求解算法的形式化描述，而不对目标间的逻辑关系进行刻画，如Mints[2]就曾给出了一个说明 PROLOG 目标求解方式的形式演算；二是逻辑公理化方法，在这种方式下，可使用逻辑连接符来表示概念之间的关系，如用"非"互换"成功"和"失败"，用析取表示诸如"或A成功或A失败"这样的陈述。那么我们该用何种逻辑来对 PROLOG 目标求解进行形式化，换言之，什么是 PROLOG 的内部逻辑？

古典逻辑除了上面的一些缺点外，它的一些连接词关于可证性也不是可分配的，而我们要求在内部逻辑中A或B可证蕴含A可证或B可证，这很自然地让我们想到直觉主义逻辑，但在直觉主义逻辑中非非A和A不等价，而我们却期望使用逻辑非来交换"成功"和"失败"的概念，因此直觉主义逻辑也不是我们期待的选择，下面我们看几个例子。

例1 设 P1 是一个包含下列子句的程序：

(1) R:—P

(2) R:—NOT(P)

(3) P:—P

显然查询R关于程序P1存在循环。给定否定的一种实现方式(如否定解释为失败)，容易看到上述程序在古典逻辑中将包括

$$R \longleftrightarrow (P \vee \neg P)$$

$$P \longleftrightarrow P$$

作为定理，因为 $P \vee \neg P$在古典逻辑中为真，因此，R也是一个定理。

例2 设程序 P2 仅含如下形式的唯一子句

A:—NOT(A)

显然查询A关于该程序循环。事实上，我们有理由认为关于该程序的任何公理化结果中$A \longleftrightarrow \neg A$是一个定理，这样当内部逻辑是古典逻辑或直觉主义逻辑时，我们将得到一个不一致的理论，从而平凡地证明了A。

例3 设程序 P3 仅包含子句

A:—A, B

在标准PROLOG目标求解方式下，查询A关于 P3 是循环的。对于P3，我们要求它的公理化应能表示下述事实：

(1) A失败　　如果A失败或A成功但B失败；

(2) B失败

(1)、(2)到逻辑语言的翻译结果可以写成

(3) $\neg A \vee (A \wedge \neg B) \rightarrow \neg A$

(4) $\neg B$

在古典逻辑或直觉主义逻辑中，从(3)、(4)我们可以推出$\neg A$。

综上，古典逻辑和直觉主义逻辑都不适合于作为 PROLOG 的内部逻辑，Cerrito指出其原因是由于这两种逻辑中都隐含结构规则的存在，在第1节．我们已说明消解法就是使用非逻辑公理实例和 cut 规则的自动证明，因此我们有理由认为PROLOG的内部逻辑不含结构规则，而这正是（非交换）线性逻辑所具备的性质。

3．线性逻辑简介

线性逻辑是法国Girard 教授在研究二阶λ演算F系统的指称语义时发展起来的一个与证明论及计算机科学有着密切关系的新型逻辑系统。线性逻辑（简记为LL），从根本上说由古典逻辑的Gentzen 型矢列演算中去除结构规则（弱规则和缩规则）所决定，关于线性逻辑的详细介绍，读者可参见[1]、[4]。这里我们仅涉及与本文讨论有关的部分内容和性质。

由于去除了结构规则，在线性逻辑中自然地有两种形式的合取（分别为$\otimes$，&）和析取（分别为$\parr$，$\oplus$），它们各有其不同的含义。进一步由于线性逻辑的对称性，一般把矢列$G_1, \cdots, G_n \vdash D_1, \cdots, D_m$写成单边形式 $\vdash G_1^{\perp}, \cdots, G_n^{\perp}, D_1, \cdots, D_m$。

3.1 语言 $\mathbb{L}$

令$P_1, P_1^{\perp}, \cdots, P_i, P_i^{\perp}, \cdots$是命题变量的无穷序列，语言 $\mathbb{L}$ 包含从命题变量出发由下述连接符构造的公式。

乘连接符　　$\otimes$ (times)，$\parr$ (par)

加连接符 & (with), ⊕ (plus) 其中连接符⊗关于⊕是可分配的，即，对任何公式A，B，C 有

$A\otimes(B\oplus C)=(A\otimes B)\oplus(A\otimes C)$。

线性非上，它的作用定义如下：

$(P_i)^{\perp}=d\ P_i^{\perp}$

$(P_i^{\perp})^{\perp}=d\ P_i$

$(A\otimes B)^{\perp}=dA_1^{\perp}\parr B_2^{\perp}$

$(A\parr B)^{\perp}=d\ A^{\perp}\otimes B^{\perp}$

$(A\oplus B)^{\perp}=d\ A^{\perp}\&B^{\perp}$

$(A\&B)^{\perp}=d\ A^{\perp}\oplus B^{\perp}$

线性隐含$\multimap$定义如下

$A\multimap B=d\ A^{\perp}\parr B$

3.2 系统LL

L中的矢列，称线性矢列，是形为$\vdash\Gamma$的表达式，其中Γ是公式的有穷序列，系统LL的规则有：

逻辑公理 $\vdash A, A^{\perp}$

cut规则 $\dfrac{\vdash\Gamma, A \quad \vdash A^{\perp}, \Delta}{\vdash\Gamma, \Delta}cut$

交换规则 $\dfrac{\vdash\Gamma}{\vdash\Gamma'}$

其中Γ'是Γ中公式的排列，(在下面的讨论中我们将限制该规则的使用，事实上如果去除了交换规则，则我们将得到非交换LL系统，为了简单，我们仍在系统LL中讨论目标求解的公理化，但排除交换规则的使用)。

加规则 $\dfrac{\vdash\Gamma, A \quad \vdash\Gamma, B}{\vdash\Gamma, A\parr B}$

$\dfrac{\vdash\Gamma, A}{\vdash\Gamma, A\oplus B}$ $\qquad$ $\dfrac{\vdash\Gamma, B}{\vdash\Gamma, A\oplus B}$

乘规则 $\dfrac{\vdash\Gamma, A \quad \vdash B, \Delta}{\vdash\Gamma, A\otimes B, \Delta}$

$\dfrac{\vdash\Gamma, A, B, \Delta}{\vdash\Gamma, A\parr B, \Delta}$

3.3 系统LL的某些性质

命题1 (cut 消去定理) 令$\vdash\Delta$是一线性矢列，H是Ⅱ中线性矢列的集合，若$\vdash\Delta$从非逻辑公理集H是LL-可导出的，则存在一个$\vdash\Delta$的证明D，满足：若

$$\frac{\vdash\Gamma, A \quad \vdash A^{\perp}, \Delta}{\vdash\Gamma, \Delta}$$

是D中出现的一cut规则，则或A或$A^{\perp}$在H中某一矢列中出现。

命题2 (子公式性) 令$\vdash\Delta$是一线性矢列，H是Ⅱ中线性矢列集合，若$\vdash\Delta$从非逻辑公理集H是LL-可导出的，则存在一个$\vdash\Delta$的证明D，D中出现的每一公式A是

(1) Δ中公式的子公式，或

(2) H中某矢列中的一个公式，或

(3) H中某矢列中公式之一个子公式的线性非。

4. PROLOG的正确性和完备性

4.1 PROLOG程序P到语言Ⅱ的翻译

设P是一个PROLOG程序，本节我们给出P在Ⅱ中的翻译LT(P)，LT(P)将确切给出目标求解成功或失败的信息。在给出翻译的形式定义之前，让我们看一个例子。

例4 设P_4是一个PROLOG程序，头为C的子句有：

(1) A, B⊢C

(2) D, E⊢C

(3) ⊢C

(4) F ⊢C

设C是P_4的目标，则关于子句(1)目标C求解成功(即A成功且B也成功)可表示成线性公式：

$$A\otimes B\multimap C \qquad (a)$$

关于子句(2)目标C求解成功(在于句(1)失败后，即A失败或A成功但B失败后)可表示成：

$$(A^{\perp}\oplus(A\otimes B^{\perp}))\otimes(D\otimes E)\multimap C \qquad (b)$$

而目标C关于子句(3)求解成功可表示成：

$$(A^{\perp}\oplus(A\otimes B^{\perp}))\otimes(D^{\perp}\oplus(D\otimes E^{\perp}))\multimap C \qquad (c)$$

我们定义原子目标C的成功集$S_c=\{a, b, c\}$。

如果所有子句均失败，则程序P_4将以目

标C求解失败终止，由于子句(3)的存在，因此原子目标C的失败集$F_c=\phi$。由⊗关于⊕的分配律，S_c可以写成：

$A\otimes B\multimap C$

$A^{\perp}\otimes(D\otimes E)\multimap C$

$(A\otimes B^{\perp})\otimes(D\otimes E)\multimap C$

$A^{\perp}\otimes D^{\perp}\multimap C$

$A^{\perp}_{\blacksquare}\otimes(D\otimes E^{\blacktriangle}_{\blacksquare})\multimap C$

$(A\otimes B^{\perp})\otimes D^{\perp}_{\blacksquare}\multimap C$

$(A\otimes B^{\blacktriangle}_{\blacksquare})\otimes(D\otimes E^{\perp})\multimap C$

进一步我们可把S_c表示成线性矢列：

$\vdash A^{\perp},\ E^{\perp}_{\blacksquare},\ C$

$\vdash A,\ D^{\perp},\ E^{\perp},\ C$

$\vdash A^{\perp},\ B,\ D^{\perp},\ E^{\perp},\ C$

$\vdash A,\ D,\ C$

$\vdash A,\ D^{\blacktriangle},\ E,\ C$

$\vdash A^{\perp},\ B,\ D,\ C$

$\vdash A^{\perp},\ B,\ D^{\perp}_{\blacksquare},\ E,\ C$

的集合$S^{i\,q}_c$。(要注意的是隐含式前件中每个公式，当它们在矢列右边中出现时都改变了符号，如A改为$A^{\perp}$，$A^{\perp}$改为$A^{\perp\perp}$即A)。显然$F^{i\,q}_c=\phi$。若把否定解释为失败，则负目标NOT(B)求解成功等价于目标B求解失败，而NOT(B)求解失败等价于目标B求解成功。下面我们给出PROLOG程序P到Π翻译的形式定义。

设P是一个PROLOG程序，AT(P)是P中出现的所有原子的集合，设$A\in AT(P)$，我们定义Clause(A)是P中所有头为A的子句序列，它们以其在P中出现的顺序排列。令$B_1,\cdots,B_m\vdash A$是Clause(A)中第j条子句，若B_i是正文字F，我们定义$(B_i)^0=F$，若B_i是负文字NOT(F)，我们定义$(B_i)^0=F^{\perp}$。于是我们可以对子句j目标求解成功或失败定义如下：

定义1 若$m\neq 0$，则令

$\Gamma(A)^i_z=(B_1)^{0\blacktriangle},\ \cdots,(B_{z-1})^{0\perp},\ (B_z)^0;\quad 1\leqslant z\leqslant m$

$\Gamma(A)^i_0=(B_1)^{0\perp},\ \cdots,\ (B_m)^{0\perp}$

若$m=0$，则令$\Gamma(A)^i_0=\phi$ for all $n\geqslant 1$。

这里$\Gamma(A)^i_z$ $(1\leqslant z\leqslant m)$的直观含义为子目标$B_1,\cdots,B_{z-1}$求解成功，但子目标$B_z$求解失败，$\Gamma(A)^i_0$的直观含义为子目标$B_1,\cdots,B_m$皆求解成功。由$\Gamma(A)$我们可以定义目标A求解成功和失败的矢列表示集合$S^{i\,q}_A$和$F^{i\,q}_A$。

定义2

(a) $S^{i\,q}_A$的定义。若Clause(A)$=\phi$，令$S^{i\,q}_A=\phi$，否则设k为Clause(A)中子句个数，我们对$1\leqslant j\leqslant k$定义：

$$S^{i\,q}_A(j)=\{\vdash\Gamma(A)^1_{z(1)},\ \cdots,\Gamma(A)^{j-1}_{z(j-1)},\ \Gamma(A)^j_0,A;\ 1\leqslant z(i)\leqslant m(i),i=1,\cdots,j-1\}$$

$S^{i\,q}_A(j)$中元素表示关于Clause(A)中第j条子句目标A求解成功，定义$S^{i\,q}_A=\bigcup_{j=1,\cdots,k}S^{i\,q}_A(j)$。

(b) $F^{i\,q}_A$的定义。若Clause(A)为空则我们定义$F^{i\,q}_A=\{\vdash A^{\perp}\}$，若$\vdash A\in$Clause(A)则我们定义$F^{i\,q}_A=\phi$，否则我们定义：

$$F^{i\,q}_A=\{\vdash\Gamma(A)^1_{z(1)},\ \cdots,\ \Gamma(A)^k_{z(k)},\ A^{\perp};\ 1\leqslant z(j)\leqslant m(j),\ j=1,\ \cdots,\ k\}$$

从直观上说$F^{i\,q}_A$中元素说明：

关于子句1，$B^1_1,\cdots,B^1_{z(1)-1}$目标求解成功但$B^1_{z(1)}$失败；

关于子句2，$B^2_1,\cdots,B^2_{z(2)-1}$目标求解成功但$B^2_{z(2)}$失败；

…

定义3 给定程序P，对任何$A\in AT(P)$我们定义

$$D_A=S^{i\,q}_A\cup F^{i\,q}_A$$

$$LT(P)=\bigcup_{A\in AT(P)}D_A$$

其中LT(P)即PROLOG程序P在线性逻辑中的翻译。要注意的是LT(P)具有模块化性质，即若P中仅头为A的子句被修改，则在LT(P)中仅D_A被修改。

4.2 PROLOG的正确性和完备性

令P是一个PROLOG程序，A是P中一个文字，本节我们将证明下述定理：

定理1 （PROLOG的完备性）

(a) 若⊢A在LT(P)中LL可证，则关于程序P目标A求解成功；

(b) 若⊢$A^{\perp}$在LT(P)中LL可证，则关于程序P目标A求解失败。

定理2 （PROLOG的正确性）

(a) 若关于程序P，目标A PROLOG求解成功，则⊢A在LT(P)中可证；

(b) 若关于程序P，目标A PROLOG求解失败，则⊢$A^{\perp}$在LT(P)中可证。

事实上LT(P)给出了程序P的一个语义，它和PROLOG的关系类似于Clark的完全化(Completion)和SLDNF的关系。在给出上述两定理的证明之前，我们先证下述引理。

引理2 设P是一个PROLOG程序，E是一形为F或$F^{\perp}$的表达式，$F\in AT(P)$，令E*是一如下定义的语句：

若E是正文字F，E*为"关于P PROLOG目标F求解成功"；

若E是负文字$F^{\perp}$，E*为"关于P PROLOG目标F求解失败"。对任意矢列S：⊢A_1，…，A_n及S中文字A_i，令S*(A_i)及S*是如下定义的语句：

S*(A_i)：如果($A_1^{\perp}$)*且…且($A_{i-1}^{\perp}$)*且(A_{i-1})*且…且($A_n^{\perp}$)*则(A_i)*

S*：S*(A_1)且…且S*(A_n)

则(a)给定LT(P)中任意矢列S，S*真；

(b) 给定任意逻辑公理S，S*真，且若S_1，S_2是cut规则(或交换规则)的上层矢列，S_1*，S_2*真，则关于该规则的结论矢列S_3，S_3*真。

证：(a) 由LT(P)的构造易证；

(b) 显然。

下面我们给出引理的简单说明，引理2(a)告诉我们LT(P)中的矢列其原子公式目标求解成功或失败之间关系的读法，特别，若矢列S形为⊢A，则它可读成"A PROLOG目标求解成功"，而若矢列S形为⊢$A^{\perp}$，则它可读成"A PROLOG目标求解失败"，引理2(b)说明：如果上述读法对cut规则（交换规则）的前提矢列是正确的，则它对规则的结论矢列的读法也是正确的。

定理1的证明：由引理2(b)和命题2即证。

定理2的证明：由$S_A^{i'}$，$F_A^{i'}$的构造及对目标调用的子目标数进行归纳即证。

下面我们看第2节中若干例子的公理化结果：

首先考虑程序P1：

(1) P⊢R

(2) NOT(P)⊢R

(3) P⊢P

关于程序P1的公理化不应使R成为一定理。而P的翻译LT(P1)的元素为：

⊢$P^{\perp}$，R

⊢P，P，R

⊢P，$P^{\perp}$，$R^{\perp}$

⊢$P^{\perp}$，P

⊢P，$P^{\perp}$

显然⊢R在LT(P1)中可证。

关于程序P2，它的子句为NOT(A)⊢A，它的翻译LT(P2)为 ⊢A，A

⊢$A^{\perp}$，$A^{\perp}$

由于不存在缩规则，⊢A和⊢$A^{\perp}$皆不是LT(P2)中的定理。

考虑程序P3，它的翻译LT(P3)为：

⊢$A^{\perp}$，$B^{\perp}$，A

⊢A，$A^{\perp}$

⊢$A^{\perp}$，B，$A^{\perp}$

⊢$B^{\perp}$

而矢列⊢$A^{\perp}$显然在LT(P3)中不可证。

5. 结论

本文讨论了命题PROLOG目标求解成功和失败的公理化问题。实际上，给定程序P，它的线性转换LT(P)不仅说明了P的子句的意义而且给出了目标求解过程的逻辑描述。关于线性逻辑和逻辑式程序设计之间的关系还有许多工作有待研究，如：

(1) 对不同的PROLOG搜索策略公理化；

(2) 对SLDNF进行公理化；

(3) 研究LL中另一证明系统证明网和逻辑式

17-21 模块 抽象数据类型 OOP 抽象数据类 ③

计算机科学1992 Vol. 19No. 6

模块、抽象数据类型和类

梅 宏 孙永强（200030 上海交通大学计算机系） TP312

摘 要

本文分析了面向对象程序设计(OOP)中类(class)和传统程序设计模块(module)及抽象数据类型(ADT)间的差异，以便加深对OOP中类及对象的理解，进而讨论了类间继承关系。

一、引 言

面向对象程序设计(OOP)以其支持模块化设计、代码复用及可扩展性等优点，特别有利于大型复杂软件系统的构造，近几年来已引起越来越多人的注目并逐步流行起来，很多人甚至预料OOP将成为九十年代程序设计方法的主流。

面向对象的思想来源于模块封装和抽象数据类型。最早的OOP语言可追溯到离散事件仿真语言SIMULA67，然而，只是发展自SIMULA67语言的Smalltalk语言问世以来，才逐步形成了现今广为流行的OOP风格。现在，已出现了许多声称支持OOP的语言，但其中最典型和杰出的代表仍是Smalltalk-80[1]，现行OOP的概念和准则大多源自Smalltalk-80。

经过十来年的发展，OOP及其支撑语言的研究已取得了较大成功，面向对象方法已在软件工程、数据库、人工智能等领域得到了较为广泛的应用，然而，和另两种新型风格程序设计——函数式程序设计和逻辑式程序设计相比，OOP缺乏形式的数学基础，有着相当复杂的语义，继承的存在更是增加了语义复杂性。显然，为了更精确地理解OOP，形式的数学模型是需要的，人们已在这方面付出了许多努力，并取得了一定成果[2][3][4]。本文的目的是通过分析OOP和传统模块程序设计间的差异，以加深对OOP的理解。

二、OOP的基本概念

在OOP中，一个系统是对象集合，对象可如下定义，

定义 对象::=〈IO, S, M, IO〉，其中ID为对象的标识，即对象名；M是对象所能承受的操作的集合，在OOP中称为方法集，$M=\{meth_1, \cdots, meth_m\}$；S是对象的实例变量集，构成对象的结构并存贮对象的状态，$S=\{Var_1, \cdots, Var_n\}$；IO是对象的对外接口，即该对象可接受的消息集，它向外界提

程序设计之间的关系；

(4) 使用线性逻辑对并行PROLOG目标求解公理化等。

参考文献

〔1〕 J. Y. Girard, Linear Logic, TCS, 50: 1-102, 1987

〔2〕 J. Y. Girard, Towards a geometry of interaction Contemporary Machs. Vol. 92, 1989

〔3〕 S. Cerrito, A linear axiomatization of negation as failure J, Logic programming 1992: 12:1-24

〔4〕 黄林鹏、孙永强，线性逻辑导论，计算机科学91.1

〔5〕 S. Cerrito, A linear semantics for allowed logic programs. In Proc. LICS, 1990.