λ-演算 程序语言 编译程序

7-10

带类型 λ-演算*

杨祥金

(东南大学计算机系 南京 210018)

TP301.2¹²

摘要 This paper described the fundation of LAMBDA calculus with type and its application in strong type programming language.

关键词 λ-calculus with type, Polymorphic type, Natural semantics.

一、引言

λ-演算是一个用函数描述计算的形式理论体系^[1],它由许多计算模型所组成。文[1]指出,应用λ-演算比纯粹λ-演算更接近程序语言,但是由于内建函子和常数的引入常会发生一些问题,例如,常数0和函子+在机器中同样看待,使得(0x)和if(λx. x)y z会产生不正确的操作。为此必须对λ-wff的类型加以限制,编译或运行时进行类型校验,对发现错误、提高编译质量是十分有必要的。

对于现代许多程序语言,例如Miranda,程序员无须规定程序中对象的类型,编译程序能根据程序内容分析出对象的类型,通过类型推理,判断本程序的类型是否相容,若类型出错,指出错误的地点及出错的原因。对类型相容性分析称为类型校验,本文介绍的带类型λ-演算在类型校验及在程序语言的理论研究中有重要的理论价值和实用意义,是计算机科学的重要组成部分。

二、带类型λ-演算的良好构式

λL-演算(带类型λ-演算)中,所有常数、变量都应

属于一定的类型,例如,所有的自然数都应属于类型N,所有的真值都应属于类型B,象这样的基本类型称为原子类型。

定义1 类型定义如下:(a)每个原子类型是类型。(b)如果α,β是类型,则(α→β)是类型。

(α→β)称为复合类型,它表示从α集到β集合上的函数类型,所有(α→β)中的函数,其定义域为α,其值域为β的子集,→称为构子。约定用希腊字母α,β,γ表示类型,并采用如下的约定:

- 最外层的括号可省略。
- 当→构子复合时采用右结合律省略。

因此,α₁→(α₂→(…(α_n→β)…))可以简写为α₁→α₂→…α_n→β

定义2 设有任意变量v,其类型为α,在λL-演算中记作v^α;同理,任意具有类型为α的常数c,记作c^α,设λL-演算良好构式的集合为λL-WFF,λL-演算良好构式(λL-wff)定义如下:

- (a) v^α, c^α ∈ λL-WFF;
- (b) 如果 M^{α→β}, N^α ∈ λL-WFF, 则 (M^{α→β} N^α)^β ∈ λL-WFF;
- (c) 如果 x^α, M^β ∈ λL-WFF, 则 (λx^α. M^β)^{α→β};

* 国家自然科学基金资助项目, 杨祥金 副教授, 主要研究兴趣为智能语言的理论与实现、知识库、应用CAD等。

[10] Kifer M. et al., Theory of Generalized Annotated Logic Programming and Its Applications, J. of Logic Programming, 12(1992)

[11] Hahnle R., Uniform Notation Tableau Rules for Multi-Valued Logics, Proceedings of the Int'l Symposium on Multiple-Valued Logic, 1991

[12] Murray N. et al., Resolution and Path Dissolution in Multiple-Valued Logics, Proceedings of MIS, LNAI No. 542, 1991

[13] Lu J. et al., Signed Formulas and Annotated Logics, Proceedings of ISMVL, 1993

[14] Fitting M., Bilattices and the Semantics of Logic Programming, J. of Logic Programming, 11(1991)

[15] Ginsberg M., Multivalued Logics, An Uniform approach to Reasoning in Artificial Intelligence, CI3, 4 (1987)

[16] McDermott D., Nonmonotonic Logic II, J. ACM, 1982

[17] Somolenov H., Paraconsistency, Paracompleteness and Intensional Contradiction, J. Non-Classical Logic 4.4(1987)

(d)只有应用(a)(b)(c)构成的表达式才称为带λt-演算中的良构式,又称为λt-项或λt-表达式。

定义(b)指出:如果M是α→β类型的函数,N的类型为α,则“函数应用”可以实现,其结果类型为β;(c)指出:抽象是一个函数,如果x的类型为α,体M的类型为β,则抽象的类型为α→β。根据定义1和定义2可以推导λt-wff的类型。

例1 设x^α,求I_α=λx^α·x^α的类型。

解:I_α=(λx^α·x^α)^{α→α}。

例2 设x^α,y^β∈λt-WFF,则K_{α,β}=(λx^αy^β·x^α)^{α→β→α}

证明:(λx^αy^β·x^α)^{α→β→α}=(λx^α·(λy^β·x^α)^{β→α})^{α→(β→α)}

例3 设x^{α→β→γ},y^{α→β},z^α∈λt-WFF,则S_{α,β,γ}=(λx^{α→β→γ}y^{α→β}z^α·xz(yz))^{(α→β→γ)→(α→β)→α→γ}

证明:推导的过程就是求出抽象体类型的过程。用图表示如下:

$$\frac{\frac{\frac{x^{\alpha \rightarrow \beta \rightarrow \gamma}}{(xz)^{\beta \rightarrow \gamma}} \quad \frac{y^{\alpha \rightarrow \beta}}{(yz)^{\beta}}}{(xz(yz))^{\gamma}}}{(\lambda z^{\alpha} \cdot (xz(yz))^{\gamma})^{\alpha \rightarrow \gamma}}}{\frac{(\lambda y^{\alpha \rightarrow \beta} \cdot (\lambda z \cdot xz(yz))^{\alpha \rightarrow \gamma})^{\alpha \rightarrow \beta}}{(\lambda x^{\alpha \rightarrow \beta \rightarrow \gamma} \cdot (\lambda y^{\alpha \rightarrow \beta} \cdot xz(yz))^{\beta})^{\alpha \rightarrow \gamma}}}$$

其中:δ=(α→β)→(α→γ)=(α→β)→α→γ;Q=(α→β)→γ)→(α→β)→α→γ。这个树型图称为类型推理。

λt-演算中有关代换,函数抽象,函数应用的概念来源于纯粹λ-演算[1]。

定义3 λt-演算中代换的概念来源于纯粹λ-演算中的代换定义5。设x的类型为α,M的类型为β,据抽象元理论[1],λx^α·M的类型取为α→β是合理的,所以设计成(λx^α·M^β)^{α→β}。若把这个函数应用于自变量N,且N的类型为α,则((λx^α·M^β)^{α→β}N^α)^β∈λt-WFF。当然,如果N的类型不为α,则“应用”就不是λt-演算中合适公式。据文[1]中有关代换的元理论和建议取下面二个等式作为公理是合适的。

公理1 ((λx^α·M^β)^{α→β}N^α)^β=[N^α/x^α]M^β

可见,如果x的类型和N的类型不相同,则代换无意义。

公理2 (λx^α·M^β)^{α→β}=(λy^α·[y^α/x^α]M^β)^{α→β},

其中,“=”表示语义上的等同。

可见,λt-演算中约束变量改名时其“新”“老”名字应具有相同的类型,否则,改名出错。

由这两条公理得下面的系理。

系理1 设P^α,M^β,N^α∈λt-WFF,P^α出现在M^β中,若用N^α去代换M^β中P^α的一切出现,则代换所得结果为β(与M^β同类型);若N的类型不为α,则代换出错。

系理2 设P^α,M^β,N^α∈λt-WFF,则([N^α/P^α]M^β)^β∈λt-WFF。

三、λt-演算中的变换规则

一个λt-wff变换成另一个λt-wff的法则称为变换规则。

• β归约规则 用⇒_β表示,称β-reduction。

$$(\lambda x^{\alpha} \cdot M^{\beta})^{\alpha \rightarrow \beta} N^{\alpha} \Rightarrow_{\beta} [N^{\alpha}/x^{\alpha}]M^{\beta}$$

• β抽象规则 用⇐_β表示,称β-abstraction。

$$[N^{\alpha}/x^{\alpha}]M^{\beta} \Leftarrow_{\beta} (\lambda x^{\alpha} \cdot M^{\beta})^{\alpha \rightarrow \beta} N^{\alpha}$$

• β转换规则(β相等规则) 用⇔_β(或(β))表示,称β-conversion或β-equality。

$$(\beta) (\lambda x^{\alpha} \cdot M^{\beta})^{\alpha \rightarrow \beta} N^{\alpha} \Leftrightarrow_{\beta} ([N^{\alpha}/x^{\alpha}]M^{\beta})^{\beta}$$

• α转换规则 用⇔_α或(α)表示,称α-conversion或α-equality。

$$(\alpha) (\lambda x^{\alpha} \cdot M^{\beta})^{\alpha \rightarrow \beta} \Leftrightarrow_{\alpha} (\lambda y^{\alpha} \cdot [y^{\alpha}/x^{\alpha}]M^{\beta})^{\alpha \rightarrow \beta}, y^{\alpha} \notin FV(M^{\beta})$$

• η转换规则 用⇔_η或(η)表示,称η-conversion或η-equality。

$$(\eta) (\lambda x^{\alpha} \cdot M^{\alpha \rightarrow \beta} x^{\alpha})^{\alpha \rightarrow \beta} \Leftrightarrow_{\eta} M^{\alpha \rightarrow \beta}, x^{\alpha} \notin FV(M^{\alpha \rightarrow \beta})$$

相应地有η归约规则,η抽象规则。

• δ归约规则 用⇒_δ表示,称δ-reduction。

δ规则随着内建函子及常数的不同而不同,其类型处理也随着δ规则及系统而改变。例如:

$$(If P^{\alpha} Q^{\beta} R^{\gamma})^{\delta} \Rightarrow_{\delta} Q^{\beta}, \text{如果 } P^{\alpha} \text{ 为真。}$$

$$(If P^{\alpha} Q^{\beta} R^{\gamma})^{\delta} \Rightarrow_{\delta} R^{\gamma}, \text{如果 } P^{\alpha} \text{ 为假。}$$

下面描述其中β转换的性质。

∀ M, N, P ∈ λt-WFF。

$$(\rho) M^{\alpha} \Leftrightarrow_{\beta} M^{\alpha}, \text{即 } \Leftrightarrow_{\beta} \text{ 关系满足自反性。}$$

$$(\tau) M^{\alpha} \Leftrightarrow_{\beta} N^{\alpha}, N^{\alpha} \Leftrightarrow_{\beta} P^{\alpha} \vdash M^{\alpha} \Leftrightarrow_{\beta} P^{\alpha} \text{ 传递性。}$$

$$(\delta) M^{\alpha} \Leftrightarrow_{\beta} N^{\alpha} \vdash N^{\alpha} \Leftrightarrow_{\beta} M^{\alpha} \text{ 对称性。}$$

$$(\mu) M^{\alpha} \Leftrightarrow_{\beta} P^{\alpha} \vdash N^{\alpha \rightarrow \beta} M^{\alpha} \Leftrightarrow_{\beta} N^{\alpha \rightarrow \beta} P^{\alpha}。$$

$$(\nu) M^{\alpha \rightarrow \beta} \Leftrightarrow_{\beta} P^{\alpha \rightarrow \beta} \vdash M^{\alpha \rightarrow \beta} N^{\alpha} \Leftrightarrow_{\beta} P^{\alpha \rightarrow \beta} N^{\alpha}。$$

$$(\xi) M^{\beta} \Leftrightarrow_{\beta} P^{\beta} \vdash \lambda x^{\alpha} \cdot M^{\beta} \Leftrightarrow_{\beta} \lambda x^{\alpha} \cdot P^{\beta}。$$

注:⇔_β中的β和表达式中“指数”β是不同意义的字母,前者表示交换规则的类别,后者表示表达式的类型。

四、有关λt-演算的几个计算模型

定义4 计算模型用三元组表示,形如:(λt-WFF, λt-Axiom, λt-RULE)

其中, λt -wff 是 λt -演算中良构式的集合, 据定义 2 可得:

$$\lambda t\text{-wff} ::= v^a | c^a | (M^a \rightarrow N^a) | (\lambda x^a \cdot M^b) \rightarrow^b$$

λt -Axiom 是 λt 演算中公理集合, λt -RULE 是 λt -演算中进行变换时所需要的推理规则集, 它们随着计算模型的不同而不同. λt -演算中有如下几个最基本的计算模型.

1. β 转换计算模型 又称 β 相等性计算模型记作:

$$\lambda t(\Rightarrow)_\beta$$

$$\lambda t\text{-Axiom} = \{(\alpha), (\beta), (\rho)\}$$

$$\lambda t\text{-Rule} = \{(\tau), (\delta), (\mu), (\nu), (\xi)\}$$

2. 归约计算模型 记作: $\lambda t(\Rightarrow)_\rho$.

$$\lambda t\text{-Axiom} = \{(\alpha), \beta\text{-reduction}, (\rho)\}$$

$$\lambda t\text{-Rule} = \{(\tau'), (\mu'), (\nu'), (\xi')\}$$

其中: $(\tau') M^a \Rightarrow N^a, N^a \Rightarrow P^a \vdash M^a \Rightarrow P^a$

$$(\mu') M^a \Rightarrow P^a \vdash N^a \rightarrow^b M^a \Rightarrow N^a \rightarrow^b P^a$$

$$(\nu') M^a \rightarrow^b P^a \vdash M^a \rightarrow^b N^a \Rightarrow P^a \rightarrow^b N^a$$

$$(\xi') M^b \Rightarrow P^b \vdash \lambda x^a \cdot M^b \Rightarrow \lambda x^a \cdot P^b$$

有关归约表达式 (REDEX)、归约结果 (REDUCTUM)、Church-Rosser 定理、正则形式、归约等概念的定义和纯粹 λ -演算^[1]一样, 这里不再赘述.

下面介绍一条在无类型系统中不成立而在 λt 中成立的定理.

定理 1 在有类型 λ -演算模型中, 没有无限归约.

证明见文[2]、[3].

推论 1 (标准化理论) 每一个有类型的 λ -表达式 M^a 有正则形式, M^a 的有限步归约可得它的正则形式.

推论 2 λt -wff 的 β 相等性是是可判定的.

3. $\beta\eta$ 转换计算模型 记作: $\lambda t(\Rightarrow)_{\beta\eta}$.

在 β 转换模型的规则中加入 (η) 转换规则, 则成为 $\beta\eta$ 转换计算模型. 即 $\lambda t(\Rightarrow)_{\beta\eta} = \lambda t(\Rightarrow)_\beta + (\eta)$.

4. $\beta\eta$ 归约计算模型 记作: $\lambda t(\Rightarrow)_{\beta\eta}$.

$$\lambda t(\Rightarrow)_{\beta\eta} = \lambda t(\Rightarrow)_\beta + (\lambda x^a \cdot M^a \rightarrow^b x^a) \rightarrow^b M^a \rightarrow^b, \text{ 如果 } x^a \notin FV(M^a \rightarrow^b)$$

有关 η -redex、 $\beta\eta$ 正则形式、归约序、Church-Rosser 定理如同纯粹 λ -演算^[1]. 定理 1、推论 2 在 $(\lambda t \Rightarrow)_{\beta\eta}$ 计算模型中同样成立. 有关 $\lambda t(\Rightarrow)_{\beta\eta}$ 、 $\lambda t(\Rightarrow)_\beta$ 、 $\lambda t(\Rightarrow)_{\beta\eta}$ 、 $\lambda t(\Rightarrow)_{\beta\eta}$ 模型可用类似方法建立.

五、自然语义解释

自然语义源于逻辑中的自然演绎法. 它用逻辑中

的推理规则描述抽象语法的语义, 可以找到很多应用例子. 如 Prolog 的实现就可以是基于自然语义解释实现. DDFP^[5]语言中的函数定义的实现就是采用 DDFP 表达式的自然语义解释实现的.

1 数学表达式的自然语义解释

设有抽象语法, 用 BNF 表示如下:

$$E ::= \text{num}(\text{val}) \\ | (\text{plus } E_1 E_2) \\ | (\text{times } E_1 E_2)$$

自然语义解释上述语法表达式时, 要引进下面三个概念:

(a) $E : v$ 在自然语义解释时表示表达式 E 取值 v .

(b) 表达式的自然语义解释采用逻辑中的条件-结论推理规则形式. 横线上面列出条件, 横线下是推出的结论. 当不用条件-结论形式解释时, 此规则称为公理, 如 $\text{num}(\text{val})$ 的解释.

(c) 环境 (Environment) 是自然语义解释中的关键概念. 环境是一个数据结构, 它可以用 C 语言中的结构来实现, 也可以用 Lisp 语言中的 a-list 实现. 环境可以形式地表示为: $\{ \langle \text{var}_1, \text{val}_1 \rangle, \langle \text{var}_2, \text{val}_2 \rangle, \dots, \langle \text{var}_n, \text{val}_n \rangle \}$. 每个元组 $\langle \text{var}_i, \text{val}_i \rangle$ 称为捆 (binding), 捆的第一个分量为变量, 第二个分量为变量的值.

与环境有关的两个操作如下:

- $\text{bind}(x, v, \text{env})$, 把新的捆 (x, v) 送入环境 env .
- $\text{lookup}(x, \text{env})$, 在环境中查看变量 x 对应的值.

空环境是一个空表, 用 NIL 表示.

现在写出上述语法的自然语义, 如下表示:

$$\left. \begin{array}{l} \text{env} \vdash \text{num}(\text{val}) : \text{val} \text{ (number)} \\ \text{env} \vdash x : \text{lookup}(x, \text{env}) \text{ (variable)} \end{array} \right\} \text{公理}$$

$$\frac{\text{env} \vdash E_1 : V_1 \quad \text{env} \vdash E_2 : V_2}{\text{env} \vdash (\text{plus } E_1 E_2) : V_1 + V_2} \text{ (sum)}$$

$$\frac{\text{env} \vdash E_1 : V_1 \quad \text{env} \vdash E_2 : V_2}{\text{env} \vdash (\text{times } E_1 E_2) : V_1 * V_2} \text{ (product)}$$

2 λt -wff 的自然语义解释

当把环境用于 λt -演算类型推理时, 环境中的捆不再是 $\langle$ 变量, 值 $\rangle$ 的元组, 而是 $\langle$ 变量, 变量类型 $\rangle$ 或 $\langle$ 常量, 常量类型 $\rangle$ 的元组. 现把解释列出如下:

$$\text{env} \vdash x : \text{lookup}(x, \text{env}) \text{ (variable)}$$

$$\text{env} \vdash c : \text{lookup}(c, \text{env}) \text{ (constant)}$$

$$\frac{\text{env} \vdash M : \alpha \rightarrow \beta \quad \text{env} \vdash N : \alpha}{\text{env} \vdash (MN) : \beta} \text{ (application)}$$

$$\frac{\text{bind}(x, \sigma, \text{env}) \vdash M : \tau}{\text{env} \vdash (\lambda x \cdot M) : \sigma \rightarrow \tau} \text{ (abstraction)}$$

若把上述自然语义解释作为推理规则, 就能构成类型推理系统 (逻辑系统), 用来进行逻辑推理以判定类型的相容性和正确性.

六、多态类型和多态函数

在函数式语言设计中有一些函数不是单一的数

据类型。试看下列例子:

例4 恒等函数 id

id 3 = 3 id 的类型为: [数值] → [数值]
 id 'a' = 'a' id 的类型为: [字符] → [字符]
 id (3, 'a') = (3, 'a') id 的类型为: [元组] → [元组]
 id t = t id 的类型为: [t] → [t]
 t ∈ {数值, 字符, 元组...}, t 称为类型变量, t 的每次取值称为实例。

例5 表长函数

length [] = 0 求值。
 length (x : xs) = (length xs) + 1
 length [7, 1, 3] = 3 length 的类型为: [num] → num。
 length ['7', '1', '3', 'z'] = 4 length 的类型为: [char] → num。
 length [(3, 'a'), (26, 'z')] = 2 length 的类型为: [(num, char)] → num。
 length [id, id] = 2 length 的类型为: [A → A] → num。

length 的类型为 [t] → num。

t ∈ {num, char, (num, char), A → A...}
 t 称为类型变量, t 的每次取值称为 t 的实例。

例6

twice fx = f(fx)
 twice = λf. λx. f(fx) 纯粹 λ-演算中的抽象。
 twice = λf^{INT}. λx^{INT}. (f^{INT}(f^{INT}x))^{INT} λ-演算中的抽象。
 twice 的 lambda 定义中有三类变量, x 是自变量, f 为函数变量, t 为类型变量, t 的每一个实例称为类型常量。

定义5 设有有限个类型常量, 及无限多个类型变量, 则可定义类型图 (type-scheme) 如下:

- 所有类型常量和类型变量是类型图。
 - 如果 α 和 β 是类型图, 则 (α → β) 是类型图。
- 一个类型图就是不含变量的类型图, 即类型图的代换实例。实际上, 这里的类型常量就是定义 1 中的原子类型, 定义 5 是由定义 1 引入类型变量而成。为了区分类型变量和良构式变量 (即自变量、函数变量) 可在类型变量上作标记, 如 *t 或 t' 等, 以示区别。

定义6 含有类型变量的类型图称为多态类型 (polymorphic type)。

把多态类型引入 λt-演算后, 还可引入多态类型函数的概念 (简称类型函数, 或多态函数)。在带有多态类型的 λ-演算中, 用大写 lambda "Λ" 表示类型变量的约束特性。例如, 上述 twice 函数可以看作三个自变量的函数, twice tfx, 其第一自变量为类型变量 t, 故:

$$\text{twice} = \Lambda t. \lambda f^{\text{INT}}. \lambda x^{\text{INT}}. (f(fx))^{\text{INT}}$$

$$\text{twice INT} = \lambda f^{\text{INT}}. \lambda x^{\text{INT}}. (f(fx))^{\text{INT}}$$

$$\text{twice INT square} = \lambda x^{\text{INT}}. (\text{square}(\text{square } x))^{\text{INT}}$$

twice INT square 2 = 16^{INT}

为了引入多态类型, 必须对 λt-wff 作修改。为了简明起见, 直接用 BNF 写出 λ-wff 的修改定义如下:

$$\lambda t\text{-wff} ::= c^a \mid v^a \mid (M^a \rightarrow N^a) \mid (\lambda x^a. M^b)^{a \rightarrow b}$$

$$\mid (M^a \rightarrow \text{poly } \tau) \text{ (type, application)}$$

$$\mid (\Lambda t. M) \text{ (type, abstraction)}$$

M^a → poly^a 是一个多态类型函数, (Λt. M) 称类型抽象。

在 λt-演算的自然语义解释中, 也应加入下面二条规则:

$$\frac{\text{env} \vdash M : \Lambda t. \text{poly}}{\text{env} \vdash (M \tau) : [\tau/t]M} \text{ (type application)}$$

$$\frac{\text{env} \vdash M : \text{poly}}{\text{env} \vdash (\Lambda t. M) : \Lambda t. \text{poly}} \text{ (type abstraction)}$$

poly 是一个新的类型, 其元素为多态类型, 下面以例说明自然语义规则的应用。

例7 设有多态类型函数 Λt. λf^t. λx^t. f(fx), 求此函数的类型图及类型抽象。

$$x : t, f : t \rightarrow t, \text{env} \vdash x : t \text{ (variable)}$$

$$x : t, f : t \rightarrow t, \text{env} \vdash f : t \rightarrow t \text{ (variable)}$$

$$x : t, f : t \rightarrow t, \text{env} \vdash fx : t \text{ (application)}$$

$$x : t, f : t \rightarrow t, \text{env} \vdash f(fx) : t \text{ (application)}$$

$$\beta : t \rightarrow t, \text{env} \vdash \lambda x. (f(fx)) : t \rightarrow t \text{ (abstraction)}$$

$$\text{env} \vdash \Lambda t. \lambda f. \lambda x. f(fx) : (\Lambda t. (t \rightarrow t) \rightarrow t \rightarrow t) \text{ (abstraction)}$$

$$\text{env} \vdash \Lambda t. \lambda f. \lambda x. f(fx) : \Lambda t. (t \rightarrow t) \rightarrow t \rightarrow t \text{ (type abstraction)}$$

多态类型函数 Λt. λf^t. λx^t. f(fx) 的类型图为: (t → t) → t → t, 类型抽象为: Λt. (t → t) → t → t。

七、讨论

λt-演算在强类型函数式语言 (ML, Miranda 等) 中已得到了广泛的应用。程序语言引入类型概念为提高编译质量, 进行程序的正确性证明与验证提供了理论基础, λt-演算为程序的类型理论提供了理论基础, 已成为理论计算机科学的重要组成部分, 对它进行研究既有理论价值又有应用意义。

参考文献

- [1] 杨祥金, λ-演算的理论体系, 待发表
- [2] Tait W. W., Intensional interpretation of functionals of finite, JSL 32, 1967, pp. 198~212
- [3] Troelstra A. S., Metamathematical Investigations of Intuitionistic Arithmetic and Analysis, SLNN 344, 1973
- [4] Robin Milner, A Theory of type polymorphism in programming, Computer Science Dept. University of Edinburgh, Edinburgh, Scotland, 1978
- [5] 杨祥金, 范伯南, 一种基于表达式的泛函数程序设计语言, 《计算机研究和发展》Vol. 30, No. 4, 1993