# 逻辑框架的语法、语用及语义
## ——(Ⅰ)新型逻辑框架的语法和语用*

傅育熙　宋哲炫　孙永强

(上海交通大学计算机科学与工程系　上海　200030)

**摘要**　逻辑框架是用以定义逻辑的类型系统. 在爱丁堡逻辑框架ELF(Edinburgh logical framework)和马丁诺夫逻辑框架的基础上,本文提出了一个集两者优点于一身的新型逻辑框架. 此逻辑框架特别适用于语义分析. 文中还给出一些如何在此逻辑框架中表示目标语言的应用实例.

**关键词**　逻辑框架,类型理论,泛类,类型.

逻辑框架这个概念是由马丁诺夫(Martin－Löf)首先提出的,其目的是给出一种非常基本的演算,用于严格地定义马丁诺夫类型理论. 这一想法在文献[1,2]中得到发展. 在这2篇文章中,作者的观点是:一个逻辑框架是一个用以定义其它逻辑框架的类型系统. 文献[1,2]中提出的语言现象被称为ELF(Edlinburgh logical framework),ELF的核心部分与马丁诺夫逻辑框架的核心部分是一样的. 但由于2种语言的出发点有所不同,它们的设计策略也有所区别. 在马丁诺夫的逻辑框架中,目标语言(Object Language)的判断相等(Judgmental Equality)是由逻辑框架的判断相等来定义的,而在ELF中,目标语言的判断相等是由框架中的常项来模拟的,2种方法各有优缺点,前者有助于对逻辑框架模型论的研究,后者则保持了语言的可判定性.

关于逻辑框架的文章已有很多[3~6],到目前为止,有关逻辑框架的研究侧重于语言的证明论及实现策略. 本文是我们撰写的一系列关于逻辑框架文章中的第1篇. 我们将对逻辑框架的语法、语用及语义进行讨论. 我们的主要贡献是提出了一种将马丁诺夫逻辑框架和ELF的特点集成于一身的新型逻辑框架. 在此语言的基础上,我们将在后继文章中对逻辑框架的模型论进行研究.

## 1　马丁诺夫逻辑框架与ELF

为了将我们的逻辑框架和马丁诺夫逻辑框架及ELF进行比较,我们首先简单回顾一下马丁诺夫逻辑框架和ELF.

---

* 本文研究得到国家863高科技项目的资助. 作者傅育熙,1962年生,副教授,主要研究领域为类型理论,范畴语义,并行理论等. 宋哲炫,1973年生,硕士生,主要研究领域为逻辑类型理论. 孙永强,1931年生,教授,主要研究领域为计算机语言,计算理论,并行处理.

本文通讯联系人:傅育熙,上海200030,上海交通大学计算机科学与工程系

本文1995－06－19收到修改稿

### 1.1 马丁诺夫逻辑框架

马丁诺夫逻辑框架中有外部定义等价，具体规则如下：

$$\textit{Context}\quad \frac{}{\langle\,\rangle valid}\ \textit{Empty Context}\qquad \frac{\Gamma\vdash A\ type\quad x\ fresh}{\Gamma,x:A\ valid}\ \textit{Context Extension}$$

$$\textit{Assumption}\quad \frac{\Gamma,x:A,\Gamma' valid}{\Gamma,x:A,\Gamma'\vdash x:A}\ \textit{Variable}$$

$$\textit{Type}\quad \frac{\Gamma\vdash A\ type\ \Gamma,x:A\vdash B\ type}{\Gamma\vdash(x:A)B\ type}\ \textit{Prod}$$

$$\textit{Construction}\quad \frac{\Gamma,x:A\vdash b:B}{\Gamma\vdash(x)b:(x:A)B}\ \textit{Abs}\qquad \frac{\Gamma\vdash f:(x:A)B\quad \Gamma\vdash a:A}{\Gamma\vdash f(a):B[a]}\ \textit{App}$$

$$\textit{Set}\quad \frac{\Gamma\ Valid}{\Gamma\vdash Set\ type}\ \textit{Set}\qquad \frac{\Gamma\vdash S:Set}{\Gamma\vdash El(S)type}\ \textit{Reflection}$$

$$\textit{Conversion}\quad \frac{\Gamma\vdash a:A\quad \Gamma\vdash A=B}{\Gamma\vdash a:B}\ \textit{Conv}$$

用如上框架，$\prod$ -类型可如下定义：

$\prod:(X:Set)(Y:(x:El(x))Set)Set$

$\wedge:(X:Set)(Y:(x:El(x))Set)(f:(x:El(x))El(Y(x)))El(\prod(X,Y))$

$\cdot:(X:Set)(Y:(x:El(x))Set)(f:El(\prod(X,Y)))(x:El(x))El(Y(x))$

计算规则可表示：$\cdot(A,B,\wedge(A,B,f),a)=f(a):El(B(a))$

其中 $A:Set$，$B:(x:El(A))Set$，$f:(x:El(A))El(B(x))$，$a:El(A)$.

### 1.2 爱丁堡逻辑框架 ELF

ELF 是基于马丁诺夫理论发展的，在马丁诺夫逻辑框架中，仅能用 *Set* 表示一类逻辑，而 ELF 则试图表示全部逻辑. 为此要有许多类似在马丁诺夫逻辑框架中 *Set* 的结构. 为了刻画这种结构，该系统定义了 3 个层次：对象(Object)，类型(Type)和超类(Kind). 顶层超类用来引入不同的范类. 这样描述范围便扩大了. 在每一个具体应用中，都要引入一些常量，构成一个标志(Signature). 以下是 ELF 的规则：

*Valid Signatures*

$$\frac{}{\langle\,\rangle}\ \textit{Empty Sig}\qquad \frac{\Sigma Sig\vdash_\Sigma K\ a\ fresh}{\Sigma,a:K\ Sig}\ \textit{Kind Sig}$$

$$\frac{\Sigma Sig\vdash_\Sigma A:Type\ c\ fresh}{\Sigma,c:A\ Sig}\ \textit{Type Sig}$$

*Valid Contexts*

$$\frac{\Sigma Sig}{\vdash_\Sigma\langle\,\rangle}\ \textit{Empty Ctxt}\qquad \frac{\vdash_\Sigma\Gamma\ \Gamma\vdash_\Sigma A:Type\ x\ fresh}{\vdash_\Sigma\Gamma,x:A}\ \textit{Type Ctxt}$$

*Valid Kinds*

$$\frac{\vdash_\Sigma\Gamma}{\Gamma\vdash_\Sigma Type}\ Type\ Kind\qquad \frac{\Gamma,x:A\vdash_\Sigma K}{\Gamma\vdash_\Sigma\prod x:A.K}\ Pi\ Kind$$

*Valid Constructors*

$$\frac{\vdash_\Sigma\Gamma\ c:K\in\Sigma}{\Gamma\vdash_\Sigma c:K}\ \textit{Cst Con}\qquad \frac{\Gamma,x:A\vdash_\Sigma B:Type}{\Gamma\vdash_\Sigma\prod x:A.B:type}\ \textit{Pi Con}$$

$$\frac{\Gamma,x:A\vdash_\Sigma B:k}{\Gamma\vdash_\Sigma\lambda x:A.b:\prod x:A.K}\ Abs\ Con \qquad \frac{\Gamma\vdash_\Sigma A:\prod x:B.K\ \Gamma\vdash_\Sigma M:B}{\Gamma\vdash_\Sigma AM:K[M/x]}\ App\ Con$$

$$\frac{\Gamma\vdash_\Sigma A:K\ \Gamma\vdash_\Sigma K'\ \Gamma\vdash_\Sigma K\cong K'}{\Gamma\vdash_\Sigma A:K'}\ Conv\ Con$$

*Valid Objects*

$$\frac{\vdash_\Sigma\Gamma\ c:A\in\Sigma}{\Gamma\vdash_\Sigma c:A}\ Const\ Obj \qquad \frac{\vdash_\Sigma\Gamma\ x:A\in\Gamma}{\Gamma\vdash_\Sigma x:A}\ Var\ Obj$$

$$\frac{\Gamma,x:A\vdash_\Sigma M:B}{\Gamma\vdash_\Sigma\lambda x:A.M:\prod x:A.B}\ Abs\ Obj \qquad \frac{\Gamma\vdash_\Sigma M:\prod x:A.B\ \Gamma\vdash_\Sigma N:A}{\Gamma\vdash_\Sigma MN:B[N/x]}\ App\ Obj$$

$$\frac{\Gamma\vdash_\Sigma M:A\ \Gamma\vdash_\Sigma A'\ \Gamma\vdash_\Sigma A\cong A'}{\Gamma\vdash_\Sigma M:A'}\ Conv\ Obj$$

让我们看一下如何用 ELF 来表示简单类型 λ 演算. 一个完整的形式化表示应包括对 λ 抽象与合成等式进行表示. 所以标志中至少包括以下常量:

$U:Type$

$T:U\to Type$

$=:\prod\sigma:U.T(\sigma)\to T(\sigma)\to Type$

$\Rightarrow:U\to U\to U$

$abs:\prod\sigma:U.\prod\tau:U.[T(\sigma)\to T(\tau)]\to T(\sigma\Rightarrow\tau)$

$app:\prod\sigma:U.\prod\tau:U.T(\sigma\Rightarrow\tau)\to T(\sigma)\to T(\tau)$

$\beta:\prod\sigma:U.\prod\tau:U.\prod f:T(\sigma)\to T(\tau).\prod x:T(\sigma).[app(\sigma,\tau)(abs(\sigma,\tau)(f);x)=_\beta fx]$

## 2 $\lambda_{TT}$,一个用以定义类型理论的框架

严格地说,在一个逻辑类型框架中所能表示的是其它类型演算. 某些逻辑之所以能在逻辑框架中表示是因为这些逻辑对应于某些类型演算. 为此,将我们的逻辑框架叫做 $\lambda_{TT}$,这里 $TT$ 是 *Type Theory* 的缩写.

$\lambda_{TT}$的语法定义如下:定义这门语言的目的在于将 ELF 和马丁诺夫逻辑框架的一些特点融合起来:

$Type\ A::=C\mid U\mid A\times A'\mid\prod x:A.A'\mid t_U(M)$

$Object\ M::=c\mid x\mid\langle M,M'\rangle\mid\pi_1M\mid\pi_2M\mid\lambda x:A.M\mid M(M')$

$C$ 是在标志中说明的类型常量,$U$ 是泛类说明中的泛类,$c$ 是一个定义于标志的对象,$x$ 是对象变量.

$\lambda_{TT}$有如下形式的断言:

• $\Omega\ Uni$—$\Omega$ 说明了一些特殊类型泛类,这里的泛类是一个其对象能提升为类型的特殊类型.

• $\vdash_{\Omega;\Sigma}Sig$—$\Sigma$ 说明了一些常类型和/或常对象,这些对象的类型可以包括 $\Omega$ 中说明的泛类.

· $\vdash_{\Omega,\Sigma}\Gamma\ Con$—$\Gamma$ 说明了一些变量，这些变量的类型为含有 $\Omega$ 和 $\Sigma$ 中说明了的项.

· $\vdash_{\Omega,\Sigma}Th\ Theory$—$Th$ 是定义在 $\Omega$ 与 $\Sigma$ 之下的有限等式集. 项中的替换同一般语言中定义的相似，唯一值得一提的是

$$T_U(M)[M'/x] \underline{def}\ t_U(M[M'/x])$$

关于 $\lambda_{TT}$ 中合式实体的规则如下：

*Universe*

$$\frac{}{\vdash\ (\ )ni}\ Empty\ Universe \qquad \frac{\vdash\Omega ni\ U\ fresh}{\vdash\Omega,U\ ni}\ Universe\ Intro$$

*Signature*

$$\frac{\vdash\Omega Uni}{\vdash_\Omega(\ )Sig}\ Empty\ Sig \qquad \frac{\vdash_\Omega\Sigma Sig\ C\ fresh}{\vdash_\Omega\Sigma,C\ Sig}\ Sig-Type$$

$$\frac{[]\vdash_{\Omega,\Sigma}A\ Type\ c\ fresh}{\vdash_\Omega\Sigma,c:A\ Sig}Sig-Obj$$

*Context*

$$\frac{\vdash_\Omega\Sigma Sig}{\vdash_{\Omega,\Sigma}[\ ]Con}\ Empty\ Context \qquad \frac{\Gamma\vdash_{\Omega,\Sigma}A\ Type}{\vdash_{\Omega,\Sigma}\Gamma,x:A\ Con}\ Context\ Intro$$

*Type*

$$\frac{\vdash_{\Omega,\Sigma}\Gamma Con\ U\ in\ \Omega}{\Gamma\vdash_{\Omega,\Sigma}U\ Type}\ U-Type \qquad \frac{\vdash_{\Omega,\Sigma}\Gamma Con\ C\ in\ \Sigma}{\Gamma\vdash_{\Omega,\Sigma}C\ Type}\ C-Type$$

$$\frac{\Gamma\vdash_{\Omega,\Sigma}A\ Type\ \Gamma\vdash_{\Omega,\Sigma}B\ Type}{\Gamma\vdash_{\Omega,\Sigma}A\times B\ Type}\ Prod \qquad \frac{\Gamma,x:A\vdash_{\Omega,\Sigma}B\ Type}{\Gamma\vdash_{\Omega,\Sigma}\prod x:A.\ B\ Type}\ \prod-Prod$$

$$\frac{\Gamma\vdash_{\Omega,\Sigma}M:U\ U\ in\ \Omega}{\Gamma\vdash_{\Omega,\Sigma}t_U(M)\ Type}\ Reflection$$

*Object*

$$\frac{\vdash_{\Omega,\Sigma}\Gamma Con\ c:C\ in\ \Sigma}{\Gamma\vdash_{\Omega,\Sigma}c:C}\ C-Obj \qquad \frac{\vdash_{\Omega,\Sigma}\Gamma Con\ x:A\ in\ \Gamma}{\Gamma\vdash_{\Omega,\Sigma}x:A}\ V-Obj$$

$$\frac{\Gamma,x:A\vdash_{\Omega,\Sigma}M:B}{\Gamma\vdash_{\Omega,\Sigma}\lambda x:A.\ M:\prod x:A.\ B}\ Abs \qquad \frac{\Gamma\vdash_{\Omega,\Sigma}M:\prod x:A.\ B\ \Gamma\vdash_{\Omega,\Sigma}N:A}{\Gamma\vdash_{\Omega,\Sigma}MN:B[N/x]}\ App$$

$$\frac{\Gamma\vdash_{\Omega,\Sigma}M:A\times B}{\Gamma\vdash_{\Omega,\Sigma}\pi_1M:A}\ L-Proj \qquad \frac{\Gamma\vdash_{\Omega,\Sigma}M:A\times B}{\Gamma\vdash_{\Omega,\Sigma}\pi_2M:A}\ R-Proj$$

$$\frac{\Gamma\vdash_{\Omega,\Sigma}M:A\ \Gamma\vdash_{\Omega,\Sigma}N:B}{\Gamma\vdash_{\Omega,\Sigma}\langle M,N\rangle:A\times B}\ Pair$$

*Conversion*

$$\frac{\Gamma\vdash_{\Omega,\Sigma}M:A\ \Gamma\vdash_{\Omega,\Sigma}B\ Type\ \Gamma\vdash_{\Omega,\Sigma}A=B}{\Gamma\vdash_{\Omega,\Sigma}M:B}\ Conv$$

为了有一个对目标语言中等式的外部定义，我们引入等式上下文：

$$\frac{\vdash_\Omega\Sigma Sig}{\vdash_{\Omega,\Sigma}(\ )Theory}\ Empty\ Theory$$

$$\frac{\Gamma\vdash_{\Omega,\Sigma}M:A\ \Gamma\vdash_{\Omega,\Sigma}N:A\ \Gamma\vdash_{\Omega,\Sigma}Th\ Theory}{\vdash_{\Omega,\Sigma}Th[\Gamma\vdash M=N:A]Theory}\ Theory\ Intro$$

等式规则有如下的外延等式：

$$\frac{\Gamma\vdash_{\Omega,\Sigma}M:\prod x:A.B\ \Gamma,x:A\Gamma\vdash_{\Omega,\Sigma}B\ Type}{\Gamma\vdash_{\Omega,\Sigma}\lambda x:A.Mx=M:\prod x:A.B}$$

$$\frac{\Gamma\vdash_{\Omega,\Sigma}M:A\times B\ \Gamma\vdash_{\Omega,\Sigma}A\ Type\ \Gamma\vdash_{\Omega,\Sigma}B\ Type}{\Gamma\vdash_{\Omega,\Sigma}\langle\pi_1M,\pi_2M\rangle=M:A\times B}$$

以及反映泛类中对象之间与相应类型之间等式的规则：

$$\frac{\Gamma\vdash_{\Omega,\Sigma}M=N:U\ U\ in\ \Omega}{\Gamma\vdash_{\Omega,\Sigma}t_U(M)=t_U(N)}$$

其余的均为一些通常的规则(略).

一个等式上下文是一组定义等式的有限集合，如果 $\Gamma\vdash_{\Omega,\Sigma}M=N:A$ 可以由 $Th$ 中等式或 $\lambda_{TT}$的定义等式推导出来，我们记 $\Gamma\vdash_{\Omega,\Sigma}M=_{Th}N:A$.

$\lambda_{TT}$与 ELF 有许多共同之处，但细心的读者会发现我们在规则中除去了 ELF 的高阶结构，没有 ELF 中类似超类的结构，但加入了 $t$ 操作符. 这是因为虽然 ELF 是一个高阶语言，但它并不能对一个超类进行操作，超类的目的在于引入一个用类型索引的常类型族. 这儿也是该语言唯一真正依赖高阶结构的地方. 所以 ELF 并不是一个完全的高阶语言. 为了避免语言的高阶性，在 $\lambda_{TT}$中，我们去除了超类的概念，而引入提升操作符 $t$. $t$ 作用于对象 $M$，由于 $M$ 的类型 $U$ 为一个 $\Omega$ 中的泛类，故 $t_U(M)$成为一种类型. 引入 $t$，便于用类型来索引常类型族. 其具体作用在后面的几个例子中可以更清楚地看到. 引入 $t$ 后，ELF 中超类的作用完全可由 $\lambda_{TT}$完成，这也是 ELF 唯一比 $\lambda_{TT}$多的部分，但 $\lambda_{TT}$还有许多 ELF 所没有的优点，例如 $Th$ 等式集的引入等等，所以 $\lambda_{TT}$的表现能力必然不亚于 ELF.

在真正的类型理论的意义下，$\lambda_{TT}$是马丁诺夫逻辑框架的推广. 其中附加部分是引入积类型和泛类说明. (马丁诺夫仅需处理泛类 $Set$，所以其逻辑框架没有泛类说明). 泛类说明是一种不用类似 ELF 的高阶算子，但又保持语言强大表现力的很好的办法. 在语言中如果仅靠$\prod$类型，其表现能力不如我们所愿，但如果加入 $\Sigma$ 类型，很容易产生一些问题，折衷办法是引入积类型×. 积类型对一些问题的说明更接近实际情况，例如在 ELF 中加法操作符可表示为 $+N\to N\to N$，这有一个缺点，$+(n)$在目标语言中几乎没有对应的实体. 在 $\lambda_{TT}$中可将加法操作符表示为 $+N\times N\to N$，上述问题就不复存在.

## 3 目标语言的表示

这一节中，我们给出 3 个在 $\lambda_{TT}$框架中表示类型演算和逻辑的例子，在下文中可用下标表示函数作用(Application)，如 $App_{\sigma,\tau}$表示 $App(\sigma)(\tau)$.

例 3.1：我们给出一个只有有限常类型的简单类型 $\lambda$ 演算，以下是 $\lambda_{TT}$的公式.

$\Omega_\lambda$ 是 $U$

$\Sigma_\lambda$ 是

$\wedge:U\times U\to U$

$\Rightarrow:U\times U\to U$

$abs:\prod\sigma:U.\prod\tau:U.[t_U(\sigma)\to t_U(\tau)]\to t_U(\sigma\Rightarrow\tau)$

$app: \prod \sigma:U.\ \prod \tau:U.\ t_U(\sigma \Rightarrow \tau) \times t_U(\sigma) \to t_U(\tau)$

$pair: \prod \sigma:U.\ \prod \tau:U.\ t_U(\sigma) \times t_U(\tau) \to t_U(\sigma \wedge \tau)$

$pr_1: \prod \sigma:U.\ \prod \tau:U.\ t_U(\sigma \wedge \tau) \to t_U(\sigma)$

$pr_2: \prod \sigma:U.\ \prod \tau:U.\ t_U(\sigma \wedge \tau) \to t_U(\tau)$

$Th_\lambda$ 是

$\vdash_{\Omega,\Sigma}[\sigma:U,\tau:U,f:t_U(\sigma) \to t_U(\tau),x:t_U(\sigma) \vdash app_{\sigma,\tau}(abs_{\sigma,\tau}(f),x) = fx:t_U(\tau)]$

$\vdash_{\Omega,\Sigma}[\sigma:U,\tau:U,f:t_U(\sigma \Rightarrow \tau) \vdash abs_{\sigma,\tau}(\lambda x:t_U(\sigma).\ app_{\sigma,\tau}(f,x)) = f:t_U(\sigma \Rightarrow \tau)]$

$\vdash_{\Omega,\Sigma}[\sigma:U,\tau:U,x:t_U(\sigma),y:t_U(\tau) \vdash (pr_1)_{\sigma,\tau}(pair_{\sigma,\tau}(x,y)) = x:t_U(\sigma)]$

$\vdash_{\Omega,\Sigma}[\sigma:U,\tau:U,x:t_U(\sigma),y:t_U(\tau) \vdash (pr_2)_{\sigma,\tau}(pair_{\sigma,\tau}(x,y)) = y:t_U(\sigma)]$

$\vdash_{\Omega,\Sigma}[\sigma:U,\tau:U,z:t_U(\sigma \wedge \tau) \vdash pair_{\sigma,\tau}((pr_1)_{\sigma,\tau}z,(pr_2)_{\sigma,\tau}z) = x:t_U(\sigma \wedge \tau)]$

例 3.2：我们将给出一个基于有限常量的马丁诺夫类型理论的表示. 在此例中，略去 $t_{Set}$ 的下标，并且将 $A:Set, B:t(A) \to Set$ 略记为 $\tau$.

$\Omega_{ML}$ 是 $Set$

$\Sigma_{ML}$ 是

$\pi: \prod X:Set.\ (t(X) \to Set) \to Set$

$\sigma: \prod X:Set.\ (t(X) \to Set) \to Set$

$\wedge: \prod X:Set.\ \prod Y:t(X) \to Set.\ \prod f:(\prod x:t(X).\ t(Y(x))).\ t(\pi(X,Y))$

$\cdot: \prod X:Set.\ \prod Y:t(X) \to Set.\ \prod f:t(\pi(X,Y)).\ \prod x:t(X).\ t(Y(x))$

$pair: \prod X:Set.\ \prod Y:t(X) \to Set.\ \prod x:t(X).\ \prod y:t(Y(x)).\ t(\sigma(X,Y))$

$P_1: \prod X:Set.\ \prod Y:t(X) \to Set.\ \prod f:t(\sigma(X,Y)).\ t(X)$

$P_2: \prod X:Set.\ \prod Y:t(X) \to Set.\ \prod f:t(\sigma(X,Y)).\ t(Y(P_1(x,Y,f)))$

$Th_{ML}$ 是

$\vdash_{\Omega,\Sigma}[\Gamma,f:(\prod x:t(A).\ t(B(x))),a:t(A) \vdash \cdot(A,B,\wedge(A,B,f),a) = f(a):t(B(a))]$

$\vdash_{\Omega,\Sigma}[\Gamma,f:t(\pi(A,B)) \vdash \wedge(A,B,\lambda x:t(A).\ \cdot(A,B,f,x)) = f:t(\pi(A,B))]$

$\vdash_{\Omega,\Sigma}[\Gamma,a:t(A),b:t(B(a)) \vdash P_1(A,B,pair(A,B,a,b)) = a:t(A)]$

$\vdash_{\Omega,\Sigma}[\Gamma,a:t(A),b:t(B(a)) \vdash P_2(A,B,pair(A,B,a,b)) = b:t(B(a))]$

$\vdash_{\Omega,\Sigma}[\Gamma,f:t(\sigma(A,B)) \vdash pair(A,B,P_1(A,B,f),P_2(A,B,f)) = f:t(\sigma(A,B))]$

上面 2 个例子都是关于如何表示类型演算，这并不意味着 $\lambda_{TT}$ 不能描述逻辑，事实上，$\lambda_{TT}$ 能如 ELF 一样处理逻辑. 下面这个例子我们仅给出一个大致结构.

例 3.3：在 $\lambda_{TT}$ 表示一个目标语言时，我们第一件事要决定是否用一个 $\Omega$ 中泛类或 $\Sigma$ 中常类型来表示一个语法类. 一条常用法则是：如果实体本身有其它实体作为其对象，那么该实体用一个泛类表示. 例如在一阶逻辑中有项的语法类和公式的语法类. 公式有证明作为其对象，所以将其作为一个泛类.

$\rho$ 在 $\Omega$ 中；另一个语法类记为 $\kappa$ 在 $\Sigma$ 中.

2 Harper R, Honsell F, Plotkin G. A framework for defining logics. ECS—LFCS—91—162, Department of Computer Science, University of Edinburgh, June 1991.
3 Fu Yuxi. Some semantic issues in type theory. Ph. D. Thesis, Department of Computer Science, University of Manchester, May 1992.
4 Gardner P. Representing logics in type theory. Ph. D. Thesis, LFCS, University of Edinburgh, July 1992.
5 Huet G, Plotkin G ed. Logical frameworks. Cambridge University Press, 1991.
6 Huet G, Plotkin G, Jones C ed. Logical frameworks. 1991.

# THE SYNTAX, PRAGMATICS AND SEMANTICS OF LOGICAL FRAMEWORKS——(Ⅰ)

## THE SYNTAX AND PRAGMATICS OF A NEW LOGICAL FRAMEWORKS

Fu Yuxi Song Zhexuan Sun Yongqiang

(*Department of Computer Science and Engineering Shanghai Jiaotong University Shanghai* 200030)

**Abstract** Based on the ELF (Edinburgh logical framework) and Martin—Löf logical framework, the authors propose a logical framework that unifies the two languages. Their logical framework is well suited for semantic analysis. Examples are given to show how to encode object languages in the calculus.

**Key words** Logical framework, type theory, universe, type.