计算机科学1992 Vol.19№2

类型理论，ELF，计算机 程序设计

ELF类型理论及其应用

20-24

王飓安　　李未

TP311

（100083北京航空航天大学计算机科学系）

摘　要

Type theory and type-theoretic approaches to software development are currently active research areas in computer science. ELF (Edinburgh Logic Framework) is one of the representative results in these fields. The goal of ELF is to build a general theory of defining a wide class of logics, and to set up a valid foundation for developing logic-independent software tools. This paper gives a briefly introduction to the main idea of ELF and basic methods to define logics. Other type theories, such as Martin-Löf's intuitionistic type theory, Nuprl type theory and ALT type theory proposed by Prof. Li are compared with ELF.

一、引言

近几年来，类型理论和基于类型理论的软件开发方法受到计算机界，特别是理论计算机科学界的广泛关注，吸引了不少研究人员，也取得了丰硕的成果。其中较有代表性的工作有Martin-Löf 的直觉主义类型论，Constable等人的Nuprl系统，Huet等人的构造演算，爱丁堡大学的逻辑框架ELF，东京大学的QJ，PDL/QJ等等。国内有北京航空航天大学，国防科技大学，上海交通大学，南京大学，哈尔滨工业大学，华北计算所等许多高校和研究机构也开展了这个领域的研究。北京航空航天大学李未教授主持的ALT类型系统及其开发环境的研究已得到国际上同行的好评。

类型理论近来倍受计算机界关注的原因可以归纳为如下三点：

1.类型理论方法是形式化方法的一种。由于形式化方法有可靠的数学基础和严格的描述手段，可以精确地定义规范、实现、正确性等概念，这就使得软件开发的某些过程有可能实现自动化，而且最终软件产品的正确性有可靠的保证。软件生产自动化是软件工作者执著追求的目标，而类型理论为此提供了一种途径。

2.随着程序设计语言的不断更新和发展，需要越来越丰富的数据类型及其理论的支持。

3.类型理论与软件开发中的概念、原则和方法有着密切的对应关系。以a:A 为例，它在类型理论中有两层含意：

- A是一个类型；
- a是类型A中的一个对象。

如果我们把类型解释为某种逻辑中的命题，那么a:A就表示该命题有一种证明a，从而用类型理论就可以构造相应逻辑系统的定理证明系统或证明检查系统。如果我们把类型解释为软件的规范说明，那么a:A就表明a是一

个满足规范A的程序。于是利用相应的类型理论，就可以进行软件的形式化开发。

ELF对类型解释的方式与上述解释不同，我们将在第四节详述。

二、ELF的基本思想

ELF是用来定义各种形式的逻辑系统的统一框架，它的主要目的是要建立关于逻辑系统的一般理论，给出一种形式化语言，对计算机科学中出现的各种逻辑系统建立统一的描述方式。ELF和Martin-Löf的直觉主义类型理论，AUTOMATH，LCF，Nuprl等系统都有密切关系。

根据应用领域的不同需要，计算机科学使用各种不同的逻辑演算或形式系统。为了用计算机进行定理证明或问题求解，就需要在机器上实现这些逻辑系统。如果对每一种逻辑系统都建立一套实现机制，那将是非常耗资费时的，比如对每种对象逻辑都要处理语法分析、证明规则、导出规则、推导策略、缩写定义、引理与定理、文库的构造等等问题。于是一个很自然的问题是：能否找出各类逻辑系统的共同点，建立能定义各类逻辑系统的统一框架？如果答案是肯定的，那么我们只要实现这个逻辑框架，就相当于一劳永逸地实现了各种逻辑系统。这就是ELF的基本出发点，它要提供一套最一般的理论机制，其表达能力要足够强，以便使尽可能多的逻辑系统都能在其中定义和表述。

为了实现上述目标，ELF采用的基本方法是：第一，抓住各类逻辑系统的共性；第二，充分利用类型理论的表达能力。

首先，我们在描述一个逻辑系统或进行推理时，主要依靠的是公理和推导规则的模式，对一条公理（推导规则）模式进行实例化就会得到多个具体的公理(推导规则)。从前提到结论的证明可以看作从前提的证明到结论的证明的函数，证明的正确性主要由公理（推导规则）模式实例化的正确性以及把推导规则应用于前提及其证明的正确性来保证。于是完全可以利用λ-演算中的运算和归约机制来刻画推理过程。具体的做法可归纳为三点：

1.尽可能把逻辑系统中所有规则模式的抽象形式和参数化都化归为λ-抽象；

2.尽可能把所有规则模式的实例化都转化为λ-施用；

3.尽可能把所有的代换形式都转化为β-归约。

上述转化的正确性可通过类型匹配和类型检查来保证，从而逻辑推理中的证明检查转化成了类型理论中的类型检查。

三、ELF的类型系统

ELF类型理论中有五种基本概念：对象（objects），类型（types）和类型族（type families），超类型（kinds），上下文（contexts），和基调（signatures）。它们的语法定义如下：

基调　$\Sigma ::= \langle\rangle \mid \Sigma, c:K \mid \Sigma, c:A$

上下文　$\Gamma ::= \langle\rangle \mid \Gamma, x:A$

超类型　$K ::= TYPE \mid \prod x:A.K$

类型族　$A ::= c \mid \prod x:A.B \mid \lambda x:A.B \mid AM$

对象　$M ::= c \mid x \mid \lambda x:A.M \mid MN$

我们用M，N表示对象，A，B表示类型和类型族，K表示超类型，x，y表示变元，c表示常元。当x不在B中自由出现时把$\prod x:A.B$简记为$A \to B$。ELF类型系统的结论形式有三种：

$\Gamma \vdash_\Sigma K$　表示K是一个超类型；

$\Gamma \vdash_\Sigma A:K$　表示A具有超类型K；

$\Gamma \vdash_\Sigma M:A$　表示M具有类型A。

ELF类型理论的推导规则如下：

1.有效超类型

$$\frac{}{\vdash TYPE} \quad (1)$$

$$\frac{\Gamma \vdash_\Sigma A:TYPE \quad \Gamma, x:A \vdash_\Sigma K}{\Gamma \vdash_\Sigma \prod x:A.K} \quad (2)$$

2.超类型的有效元素

$$\frac{\Gamma \vdash_\Sigma TYPE \quad c:K \in \Sigma}{\Gamma \vdash_\Sigma c:K} \quad (3)$$

$$\frac{\Gamma\vdash_\Sigma K \quad c\notin Dom(\Sigma)}{\vdash_{\Sigma,c:K}TYPE} \qquad (4)$$

$$\frac{\Gamma\vdash_\Sigma A:TYPE \quad \Gamma,\ x:A\vdash_\Sigma B:TYPE}{\Gamma\vdash_\Sigma \prod x:A.B:TYPE} \qquad (5)$$

$$\frac{\Gamma\vdash_\Sigma A:TYPE \quad \Gamma,\ x:A\vdash_\Sigma B:K}{\Gamma\vdash_\Sigma \lambda x:A.B:\prod x:A.K} \qquad (6)$$

$$\frac{\Gamma\vdash_\Sigma B:\prod x:A.K \quad \Gamma\vdash_\Sigma N:A}{\Gamma\vdash_\Sigma BN:[N/x]K} \qquad (7)$$

$$\frac{\Gamma\vdash_\Sigma A:K \quad \Gamma\vdash_\Sigma K' \quad K\equiv K'}{\Gamma\vdash_\Sigma A:K'} \qquad (8)$$

3.类型的有效元素

$$\frac{\vdash_\Sigma A:TYPE \quad c\notin Dom(\Sigma)}{\vdash_{\Sigma,\ c:A}\ TYPE} \qquad (9)$$

$$\frac{\Gamma\vdash_\Sigma A:TYPE \quad x\notin Dom(\Gamma)}{\Gamma,\ x:A\vdash_\Sigma TYPE} \qquad (10)$$

$$\frac{\Gamma\vdash_\Sigma TYPE \quad M:A\in\Sigma\cup\Gamma}{\Gamma\vdash_\Sigma M:A} \qquad (11)$$

$$\frac{\Gamma\vdash_\Sigma A:TYPE \quad \Gamma,\ x:A\vdash_\Sigma M:B}{\Gamma\vdash_\Sigma \lambda x:A.M:\prod x:A.B} \qquad (12)$$

$$\frac{\Gamma\vdash_\Sigma M:\prod x:A.B \quad \Gamma\vdash_\Sigma N:A}{\Gamma\vdash_\Sigma MN:[N/x]B} \qquad (13)$$

$$\frac{\Gamma\vdash_\Sigma M:A \quad \Gamma\vdash_\Sigma A':TYPE \quad A\equiv A'}{\Gamma\vdash_\Sigma M:A'} \qquad (14)$$

如果一个项在基调Σ和上下文Γ中被证明是一个超类型，或具有一个超类型，或具有一个类型，则称该项在Σ和Γ中是良类型化的。在上述规则(8)和(14)中的关系≡，是由$\rightarrow_\beta^*$生成的对称传递闭包。ELF类型系统具有良好的性质，主要体现在下述五条定理之中。

1.类型和超类型唯一性定理

如果$\Gamma\vdash_\Sigma M:A$且$\Gamma\vdash_\Sigma M:A'$，则$A\equiv A'$。对超类型亦然。

2.主体归约定理

如果$\Gamma\vdash_\Sigma M:A$，且$M\rightarrow_\beta^* M'$，则$\Gamma\vdash_\Sigma M':A$。对类型亦然。

3.Church-Rosser定理

所有良类型化的项具有Church-Rosser性质(合流性)。

4.强范式化定理

所有良类型化的项都不存在无穷归约。

5.可判定性定理

ELF类型系统中的三种结论形式都是递归可判定的。

ELF类型理论的这些良好性质，特别是可判定性定理，保证了我们的确可以机械地把对象逻辑中的证明转化为ELF中的类型检查。

四、用ELF定义逻辑系统的基本方法

给定对象逻辑系统L，用ELF定义L就需要给出相对于L的基调Σ_L，并对公理、推导规则和证明都给出相应的表示。用ELF定义逻辑系统的过程分为三步：

第一步 定义对象逻辑中的抽象语法。对象逻辑中的抽象语法包括项和公式，它们又分别称为个体表达式和命题表达式，而表达式是通过表达式构子生成的。例如一阶算术系统中项t，u以及公式φ，ψ分别由下述抽象语法给出：

$$t::=x\mid o\mid succ(t)\mid t+u\mid t\times u$$

$$\varphi::=t=u\mid t<u\mid\neg\varphi\mid\varphi\wedge\psi\mid\varphi\vee\psi\mid\varphi\supset\psi\mid\forall x.\varphi\mid\exists x.\varphi$$

对抽象语法中的每种语法范畴都用一个类型表示，并对每种表达式构子都用Σ_L中一个适当的常元表示。比如上述抽象语法中的项和公式可分别用类型i和o表示：

i:TYPE, o:TYPE

于是项构子和公式构子分别对应如下常元：

o: i	$\times$: $i\rightarrow i\rightarrow i$
succ: $i\rightarrow i$	$=$: $i\rightarrow i\rightarrow o$
$+$: $i\rightarrow i\rightarrow i$	$<$: $i\rightarrow i\rightarrow o$
$\neg$: $o\rightarrow o$	$\supset$: $o\rightarrow o\rightarrow o$

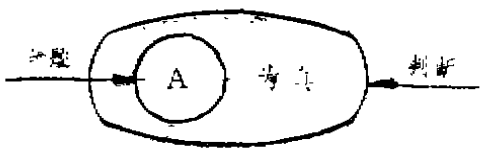
$\wedge$: o→o→o　　$\forall$: $(i\to o)\to o$

$\vee$: o→o→o　　$\exists$: $(i\to o)\to o$

这里将对象逻辑中的变元与ELF中的变元视为等同而不加区别。

第二步 建立对象逻辑中项（公式）与第一步得到的$\Sigma_L(\Gamma_X)$中类型为i(o)的有效项之间的一一对应关系。显然，基调Σ_L相当于逻辑L中语法范畴的一种编码，本步的任务是保证这种编码满足两个条件：(1)充分性，即L的所有语法范畴都已编码，且编码方式唯一；（2）可合成性，即代换和编码函数是可交换的。

第三步 处理对象逻辑L中推导规则和证明，这是ELF方法的核心。每一种逻辑都有一些基本的推理单位称为判断，这里要注意命题和判断是不同层次的两个概念。具有真值的逻辑公式称为命题，而当我们说“命题A为真”时，我们就作了一种判断，用图形可直观表述如下：



判断可以分为两类：（1）基本判断，如一阶逻辑中的φ true或只写成φ，它表示公式φ为真。矢列演算中的基本判断是$\Gamma\vdash\Delta$，它表示Δ中的某些公式是Γ中所有公式的逻辑结论。（2）高阶判断，有两种形式：Ⅰ）假设型高阶判断$J_1\vdash J_2$，它表示根据推导规则J_2是J_1的逻辑结论。Ⅱ）框架型高阶判断$\wedge_{x:A}J(x)$，它表示对于类型为A的任意x，J(x)都有效。

ELF采用“判断作为类型”的原则把判断解释为类型。设解释函数为I，则判断和类型的对应关系如下：

判断	类　型
φ	true(φ) 这里true:o→TYPE是Σ_L中的常元。
$J_1\vdash J_2$	$I(J_1)\to I(J_2)$
$\wedge_{x:A}J(x)$	$\prod x:I(A).I(J(x))$

例如在一阶逻辑中有如下两条推导规则：

$$(\neg E)\ \frac{\neg\varphi}{\varphi}\qquad (\supset I)\ \frac{\begin{matrix}(\varphi)\\ \psi\end{matrix}}{\varphi\supset\psi}$$

它们可分别写成如下高阶判断形式：

$(\neg E)\quad \wedge_{\varphi:o}(\neg\varphi\vdash\varphi)$

$(\supset I)\quad \wedge_{\varphi:o,\psi:o}((\varphi\vdash\psi)\vdash\varphi\supset\psi)$

根据判断作为类型原则，它们分别有类型

$\neg E$: $\prod_{\varphi:o}(\mathrm{true}(\neg\varphi)\to\mathrm{true}\varphi)$

$\supset I$: $\prod_{\varphi:o}\prod_{\psi:o}((\mathrm{true}\varphi\to\mathrm{true}\psi)\to \mathrm{true}(\supset\varphi\psi))$

于是，若M:true(¬φ)，则

$\neg E(\varphi)(M)$: trueφ

$\supset I(\varphi)(\varphi)(\lambda x:\mathrm{true}\varphi.x):\mathrm{true}(\supset\varphi\varphi)$

这意味着$\neg E(\varphi)(M)$是φ的一个证明，$\supset I(\varphi)(\varphi)(\lambda x:\mathrm{true}\varphi.x)$是$\varphi\supset\varphi$的一个证明。

五、评述和结论

熟知，Martin-Löf的直觉主义类型理论和Nuprl类型理论均采用“命题作为类型”的观点，而ELF则采用“判断作为类型”的原则，证明就是判断类型的项。这是比命题作为类型更加一般的方法。直观地说，ELF是把推导规则的形式看作类型，从而规则就变成高阶判断类型的证明。基本判断对应常类型，高阶判断对应高阶类型，推导规则就解释成了高阶类型的项。因此，在ELF中，规则和证明并无本质的区别。

李未教授提出的ALT类型理论将类型理论及其应用领域作了进一步的拓广。第一，ALT类型理论采用“概念作为类型”的原则，这比把命题或判断作为类型的原则更加一般，也更加灵活。第二，ALT类型理论具有更广泛的应用前景。Martin-Löf的直觉主义类型理论是为解释和实现一阶直觉主义逻辑而建立的形式理论；Nuprl类型理论旨在实现数学定理的交互证明；ELF类型理论把解释和实现计算机科学中出现的各种逻辑系统为己任；而ALT类型理论则是要为软件工程、知

识工程提供一种新的工具。用ALT类型理论可以将软件开发过程形式化，建立一种“开发演算”；ALT类型理论支持时态逻辑，模糊逻辑，多值逻辑，尤其在描述变换式软件开发方法，保证变换正确性方面，ALT类型理论已经取得了实际的成果。

诚然，类型理论及其方法亦有自身的缺点。由于类型理论方法属于形式化方法的一种，因而形式化方法所带有的诸如固有复杂性、表达繁琐、低效等局限性和缺点在类型理论中也打下了深深的烙印。

Goguen教授在[1]中指出今后几年值得重视的研究方向是软件开发方法和程序证明方法。我们认为，类型理论及其程序设计方法学是沿着这两个方向推动软件科学前进的可取途径之一。（参考文献略）

（接第80页）

自反公理：E (x, x) = true。

4、基于条件重写的子句迭加法

Zhang和Kapur[11]将条件重写规则引入定理证明过程中，单位子句（即，只含一个文字）变为无条件规则，非单位子句（若干个文字之或）变为条件规则。如，将子句 ¬q (a, a) 和q (x, y) ∨ ¬q (y, x) 分别变成规则 q (a, a) →0，以及 q(x, y) →1, if q (y, x) 。

这种方法中的推理规则也称为子句迭加。与Fribourg 方法所不同的是，这里的迭加是在两个子句的最大文字之间进行，而不是在最右文字之间进行。另外，这里不用自反公理。

小结 由上面可看出，重写证明法的一般过程是：将待证明的定理取反，Skolem化，变成一定的规范形式；再对它和公理集进行推理，直至得出矛盾。在以上几种具体方法中，Hsiang方法简单直观，影响较大，对它的研究较多，得到了很多有效的策略，而且用Hsiang方法得到的证明也较简洁。

四、与归结法之比较

重写证明法的大致过程与归结法相似，其基本操作也很相似。迭加可看成是归结再加上调解 (paramodulation)，临界对相当于归结式。但一般说来，迭加和重写比经典方法中的归结和删除策略从概念上讲要更广泛。可以证明，有一组子句集合，它的基于重写的否证过程比基于归结的过程要短(即，所含的步骤更少）。

当然，从时间耗费上看，重写法并不一定优越，因为每步迭加操作中要进行一致化和归约，而每步归结只需要一致化。

完备化过程、归结原理以及Gröbner基算法具有很多相似的特征，可置于一个统一的框架之中[1]。实际上，归结法和重写法可以互相借签对方的很多有益的东西。

五、 结束语

重写技术为定理证明提供了一条很有潜力的途径，用基于重写的证明方法已经解决了不少难度颇高的问题。（可参看 Journal of Automated Reasoning, Association of Automated Reasoning Newsletter 以及 Journal of Symbolic Computation1991年的专辑。）并且出现了一些实际的证明系统，如 RRL (Rewrite Rule Laboratory)、LP (Larch Prover)、HIPER (High Performance Rewriting)等等。在一些使用经典方法的系统（如OTTER）中也加进了项重写的一些机制。目前，人们正探讨更有效的重写证明策略，使重写方法能证明更多更难的定理。（参考文献略）