基于逻辑框架 LF 的语义性质之验证*)

庞建民[1,3] 赵荣彩[1] 王怀民[2]
(信息工程大学信息工程学院 郑州 450002)[1] (国防科技大学计算机学院 长沙 410073)[2]
(英国 DURHAM 大学 Durham, UK, DH1 3LE)[3]

摘 要 本文对基于类型理论逻辑框架(LF)的语义性质验证加以研究,针对函数式语言 LAZY-PCF+SHAR,利用计算机辅助推理方法和技术给出相应的形式化描述及相关性质证明,从而提倡严格的和计算机辅助的证明在语义性质验证方面的应用。同时,考察了 LF 以及其实现系统 Plastic 的能力。

关键词 类型理论,逻辑框架,语义性质验证,操作语义,函数式语言

Verification of Semantic Properties Based on Logical Framework

PANG Jian-Min[1,3] ZHAO Rong-Cai[1] WANG Huai-Min[2]
(Institute of Information Engineering, University of Information Engineering, Zhengzhou 450002)[1]
(School of Computer Science, National University of Defense Technology, Changsha 410073)[2]
(University of Durham, Durham, DH1 3LE, UK)[3]

Abstract In this paper we study the verification of semantic properties based on type-theoretic logical framework (LF). For functional programming language LAZY-PCF+SHAR we present formal descriptions of its concepts and proofs of the relevant properties by using methods and technologies of computer assisted reasoning. Consequently the applications of this strict and computer assisted proof in the area of verification of semantic properties are advocated. Meanwhile we investigate the power of LF and its implementation system Plastic.

Keywords Type theory, Logic framework, Verification of semantic properties, Operational semantics, Functional programming

在计算机科学领域,对语言语义的描述主要用操作语义、指称语义、公理化语义或代数语义来进行,对语义性质的验证则往往采用手工完成。本文利用基于类型理论逻辑框架(LF)的方法,对语义的性质进行形式化的计算机辅助验证,从而提倡严格的和计算机辅助的证明在语义性质验证方面的应用。

从理论的观点而言,函数式程序设计语言在推理方面是很优秀的,在传名调用(call-by-name)或传值调用(call-by-value)的框架下其优越性更加突出。但是,严格按照传名机制实现的函数式语言,由于被多次引用的变量要被复制多次,有些情况还涉及到每次需要的时候还要重新求值,因此效率不高。而由于函数式语言的参照透明性(referential transparency),这些值应该始终不变。在实践中,不必要的重新求值可以通过共享变元的方式得以避免。当该值首次需要的时候,求值开始,原来的值被该值代替,并且该值就是之后对该参量引用时的值。所以共享可以被刻画为无需复制并且在求值后更新原值这样的机制。这种方法通常被称为按需调用(call-by-need)。它提供与传名调用相同的结果值,但是在对不必要的重新求值的归约方面却有不同的行为。在按需调用的实现方面,依靠对实现中给定的程序的行为进行分析,或者进行不影响结果但改进程序行为的程序变换,可以做另外的一些改进。由于许多优化依赖于程序的行为,因此在实现惰性计算(lazy evaluation)方面所涉及的共享必须被考虑。为了正确地进行这些程序的分析,一个便于推理的含有按需调用的操作语义模型就成为了基本的需求。在文[1]中给出的带有共享及惰性计算的操作语义模型已经被证明能够和传名调用语义产生相同的结果。

传名和传值的简单性部分归功于函数应用是借助于替换(substitution)而定义这一事实。而替换却是一种在语义规则外部定义的操作。不幸的是,替换是一个允许参量被复制的操作,这使得它不适合用于形式化共享。因此,为了避免复制,一个带有共享及惰性计算的操作语义模型必须明确地指定何时以及如何替换一个参量。这项工作可以通过将其合并到能够显式地进行替换的语义规则中去的办法得到解决。值得庆幸的是,在显式替换方面,已经有像文[2]和文[3]这样的工作。这些文章研究了包含能够进行显式替换的语法及规则的归约系统 $\lambda\sigma$。但是,这个系统对出现在惰性计算中的共享并不支持,因为它对参量进行复制,并且对参量在求值后并不进行更新。这些文章强调对归约机制的最优化研究,而文[1]的目标则是模型化在实现中发生的共享,从而得到一个更加精确的用于分析的模型。因此,文[1]依赖操作语义来决定归约策略,强调系统对共享进行推理的合适性,从而为分析提供了基础。这也是为什么我们要与西蒙等人基于 Coq 系统的工作[4]相似,选择一种惰性函数式语言 LAZY-PCF+SHAR 的操作语义的性质验证作为一个我们感兴趣的论域案例来研究的原因。我们已经完成了感兴趣的所有性质的验证,可以发送 email 到 jianmin. pang@durham. ac. uk 或 jianmin_pang@hotmail. com,来索取验证代码。与西蒙等人的工作不同,

*)本课题得到欧盟项目 TYPES 资助(项目编号 types project 29001)。庞建民 教授,主要研究方向:计算机辅助推理与软件理论;赵荣彩 教授,博士生导师,主要研究方向:分布式计算与软件理论;王怀民 教授,博士生导师,主要研究方向:分布式计算与软件理论。

我们关注基于逻辑框架 LF[5,6] 的推理，这种方式摆脱了 Coq 等系统所可能带来的自身所固有的性质多于要研究的论域所具有的性质这样的问题。Isabelle[7,8] 也是一个基于推理框架的定理证明器，它并不依赖于类型理论，不强调构造性逻辑的使用，如果想在其上基于类型理论，则必须先定制。这样就增加了一定的工作量，与我们所追求的直接基于类型理论框架的理念有一定的差距。

1 类型理论与逻辑框架 LF

类型理论最初被设计用来作为形式化构造性数学的基础[9]，但近年来，科学家们发现了它在计算机科学方面的很多应用。类型理论为计算机科学中的计算和推理这两种基本概念提供了统一的处理方法。这使得人们可以在一个形式框架中同时完成编程、理解和推理三项任务，并且不增加相应的负担。另外，类型理论能够提供一种很好的抽象机制同时支持规范的开发、程序的编写和证明的构造。因此，其它规范描述语言中所存在的程序设计语言和规范语言之间的差距在类型理论中就消失了。所以，我们可以说，类型理论是一种非常有前途的支持计算机辅助推理技术的理论，在形式化数学和程序验证领域更是如此。

本文中对概念的描述基于马丁·洛夫的工作，特别是马丁·洛夫的著作[10]和诺思多姆(Nordström)等人的著作[11]。但我们将马丁·洛夫在他的著作中所称的“集合”(set)的概念称为后来大家所通用的概念“类型”(type)。

在一个类型的对象当中，有些被称为典范对象，它们是这个类型的对象经过计算之后的值。一个典范对象一定具有一种称为典范形式(canonical form)的形式。所谓典范形式是指其最外层的构造子是引入构造规则时引入的常量。计算是类型理论中的一个基本概念，由它产生一个类型理论语言中的表达式之间的计算相等这样的等价关系。为了保证类型理论中各种实体(entities)的不同使用之间协调一致，计算应该具有以下性质，即：每个对象在计算之后具有唯一的值，计算相等的对象应该具有相同的值。

1.1 命题作为类型法则

命题作为类型法则来源于柯里(Curry)[12]和霍华德(Howard)[13]对直觉的逻辑推理的自然演绎系统和类型系统之间的相关性之间的观察。类型理论既可以作为构造性数学的基础，又可以作为程序的规范和验证基础的理由。

该法则的基本思想是：任何命题 P 与一个由该命题的证明形成的类型 Prf(P)相对应；命题 P 的一个证明对应类型 Prf(P)的一个对象。进一步而言，一个命题是真的，当且仅当存在该命题的一个证明(即：它的证明构成的类型的一个对象)。因此一个命题的真假性问题可以被理解为该命题的证明所构成的类型的居住性(inhabitance)问题。类型 Prf(P)的典范对象的概念与命题 P 的典范或直接证明的概念一致；类型 Prf(P)的非典范对象可被称为命题 P 的间接证明。

命题实质上就是描述性质和事实的公式，而判定(judgement)则是公式是否为真的断言，两者是根本不同的。基于这种不同，一个拥有足够逻辑类型结构的类型理论，具有一种内部逻辑，提供一种不同于集合论和逻辑程序设计的逻辑语言。

柯里和霍华德研究的系统是在命题和类型之间存在等价性的系统，这种等价性对不同的逻辑和类型理论都成立。例如：一种简单类型(λ-演算的扩张与一阶直觉主义命题逻辑(由霍华德[13]提出)相对应；系统 F 与二阶命题逻辑相对应(季拉德(Girard)[14]对此问题进行了研究)。因此，命题作为类型法则也被称为同构。按照命题作为类型法则，我们把一个命题的证明映射到该命题的证明构成的类型的对象。类型理论的判定(judgement)因此必须是可判定的(decidable)，以便我们可以从一个判定 M:A 来断定 M 确实是类型 A 的一个对象。

1.2 逻辑框架(LF)

逻辑框架可以以多种不同的方式应用。基于判定作为类型(judgements-as-types)法则的爱丁堡逻辑框架[15]已经被用来作为形式化逻辑系统的框架而研究。马丁·洛夫的逻辑框架(参见文[11]的第三部分)被其用来表示他的内涵类型理论(intensional type theory)。我们感兴趣的逻辑框架(LF)[5]是马丁·洛夫逻辑框架的一个类型化了的版本。可以把它作为元语言来定制类型理论。

LF 中的规则在图 1 中列出，即可以基于该框架定制特定的类型理论，也可以把它本身作为一个小的类型理论来直接使用。

2 对显式替换的需求

传名调用和传值调用方法可以借用下面的替换加以描述。为了按传名的方式对项$((\lambda x:t.e)e')$求值，只要用项 e' 替换 e 中的 x 并对替换后的项求值即可。为了按传值的方式对项$((\lambda x:t.e)e')$求值，首先求 e' 的值，得到结果 v'，然后用 v' 替换 e 中的 x，再求值。

应用(application)的形式语义将被描述为推理规则。为了得到中线下面的结论，中线上面的前提必须为真。另外，如果一个项 e 计算后得到 v，则表示为 $e \downarrow v$。这样，应用的计值规则可以如图 2 那样形式地描述。

从图 2 我们可以清楚地看出，这种替换的定义简化了这些计值次序。但是关于这种替换的定义如何能够被用于描述按需调用的求值问题，却完全不清楚。在 x 被首次访问的时候，参量 e' 被用来替换 x 并求值。对 x 的后续出现，则用 e' 的结果来替换。不幸的是，在程序被执行之前并不知道 x 的哪个出现将被首先计算，并且 e' 应该只是在需要的时候才被计算。由替换的定义所带来的问题基于这样的事实：用一个项替换另一个项中的变量的实际过程中的细节已经被抽象掉了。

为了实现惰性计算，语义应该能够控制替换过程，以便替换和参量的求值恰好发生在函数体被计算的时候，而不是之前。

值得庆幸的是，将规则融入能够直接进行替换的语义中去，这种想法(通常被称为显式替换(explicit substitutions))已经在文[2,3]中研究过。在这些文章里，显式替换用于定义不带求值策略的 λ-演算的重写规则系统的定义。尽管这些规则融合了一些显式的规则来进行替换，但它们并没有考虑共享。不过，这种机制确实为惰性计算提供了相对简单的形式化。在文[2]中给出了一个被称为 λσ 的使用显式替换的系统，该演算可以对包含不可计值的替换的 λ 项计值。

尽管按需调用的操作语义与 λσ-演算有一些相似之处，但为了支持共享，它与显式替换有如下不同：

(1)语言中关于项的语法不允许替换在表达式中出现。取而代之的是，一个项只是针对最外层的被称为操作语义环境的一个单一的替换而定值。该操作语义环境与显式替换对应，它是一个相关变量约束到表达式的列表。

(2)另一个不同来自对共享第一个特征的支持，这基于这样的一个事实：在函数应用中，不像在 λσ-演算的 app 规则中破坏共享那样，环境并不被复制和分布到子表达式中。

上下文与假设(context and assumptions)

$$\frac{}{\langle\rangle\text{valid}} \quad \frac{\Gamma\vdash K\text{kind } x\notin FV(\Gamma)}{\Gamma,x:K\text{ valid}} \quad \frac{\Gamma,x:K,\Gamma'\text{valid}}{\Gamma,x:\Gamma'\vdash x:K}$$

一般相等规则(general equality rules)

$$\frac{\Gamma\vdash K\text{ kind}}{\Gamma\vdash K=K} \quad \frac{\Gamma\vdash K=K'}{\Gamma\vdash K'=K} \quad \frac{\Gamma\vdash K=K' \quad \Gamma\vdash K'=K''}{\Gamma\vdash K=K''}$$

$$\frac{\Gamma\vdash k:K}{\Gamma\vdash k=k:K} \qquad \frac{\Gamma\vdash k=k':K}{\Gamma\vdash k'=k:K}$$

$$\frac{\Gamma\vdash k=k':K \quad \Gamma\vdash k'=k'':K}{\Gamma\vdash k=k'':K}$$

相等类型规则(equality type rules)

$$\frac{\Gamma\vdash k:K \quad \Gamma\vdash K=K'}{\Gamma\vdash k:K'} \qquad \frac{\Gamma\vdash k=k':K \quad \Gamma\vdash K=K'}{\Gamma\vdash k=k':K'}$$

替换规则(substitution rules)

$$\frac{\Gamma,x:K,\Gamma'\text{valid} \quad \Gamma\vdash k:K}{\Gamma,[k/x]\Gamma'\text{valid}}$$

$$\frac{\Gamma,x:K,\Gamma'\vdash K'\text{kind} \quad \Gamma\vdash k:K}{\Gamma,[k/x]\Gamma'\vdash[k/x]K'\text{kind}}$$

$$\frac{\Gamma,x:K,\Gamma'\vdash K'\text{kind} \quad \Gamma\vdash k=k':K}{\Gamma,[k/x]\Gamma'\vdash[k/x]K'=[k'/x]K'}$$

$$\frac{\Gamma,x:K,\Gamma'\vdash k':K' \quad \Gamma\vdash k:K}{\Gamma,[k/x]\Gamma'\vdash[k/x]k':[k/x]K'}$$

$$\frac{\Gamma,x:K,\Gamma'\vdash k':K' \quad \Gamma\vdash k_1=k_2:K}{\Gamma,[k_1/x]\Gamma'\vdash[k_1/x]k'=[k_2/x]k':[k_1/x]K'}$$

$$\frac{\Gamma,x:K,\Gamma'\vdash K'=K'' \quad \Gamma\vdash k:K}{\Gamma,[k/x]\Gamma'\vdash[k/x]K'=[k/x]K''}$$

$$\frac{\Gamma,x:K,\Gamma'\vdash k'=k'':K' \quad \Gamma\vdash k:K}{\Gamma,[k/x]\Gamma'\vdash[k/x]k'=[k/x]k'':[k/x]K'}$$

类属(The kind)Type

$$\frac{\Gamma\text{ valid}}{\Gamma\vdash\text{Type kind}} \quad \frac{\Gamma\vdash A:\text{Type}}{\Gamma\vdash El(A)\text{kind}} \quad \frac{\Gamma\vdash A=B:\text{Type}}{\Gamma\vdash El(A)=El(B)}$$

依赖积类属(Dependent product kinds)

$$\frac{\Gamma\vdash K\text{ kind }\Gamma,x:K\vdash K'\text{kind}}{\Gamma\vdash(x:K)K'\text{kind}}$$

$$\frac{\Gamma\vdash K_1=K_2 \quad \Gamma,x:K_1\vdash K_1'=K_2'}{\Gamma\vdash[x:K_1]K_1'=(x:K_2)K_2'}$$

$$\frac{\Gamma,x:K\vdash k:K'}{\Gamma\vdash[x:K]k:(x:K)K'}$$

$$(\xi)\quad\frac{\Gamma\vdash K_1=K_2 \quad \Gamma,x:K_1\vdash k_1=k_2:K}{\Gamma\vdash[x:K_1]k_1=[x:K_2]k_2:(x:K_1)K}$$

$$\frac{\Gamma\vdash f:(x:K)K' \quad \Gamma\vdash k:K}{\Gamma\vdash f(k):[k/x]K'}$$

$$\frac{\Gamma\vdash f=f':(x:K)K' \quad \Gamma\vdash k_1=k_2:K}{\Gamma\vdash f(k_1)=f'(k_2):[k_1/x]K'}$$

$$(\beta)\quad\frac{\Gamma,x:K\vdash k':K' \quad \Gamma\vdash k:K}{\Gamma\vdash([x:K]k')(k)=[k/x]k':[k/x]K'}$$

$$(\eta)\quad\frac{\Gamma\vdash f:(x:K)K' \quad x\notin FV(f)}{\Gamma\vdash[x:K]f(x)=f:(x:K)K'}$$

图 1　LF 的推理规则

传名: $\dfrac{e[e'/x]\downarrow v}{((\lambda x:t.e)e')\downarrow v}$　　传值: $\dfrac{(e'\downarrow v')\wedge(e[v'/x]\downarrow v)}{((\lambda x:t.e)e')\downarrow v}$

图 2　传名与传值

(3)第三个不同来自对共享第二个特征的支持,那就是在环境中被变量绑定的表达式,可能被它计算后的值所代替。这就允许一个参量的初始拷贝被它计算后的值所代替。为了这个值在以后再被使用,环境不像在 λσ-演算中那样,在达到值之后就消去,而是在整个计算过程中保持。所以,一个表达式的求值结果和它的环境将是一个带有环境的值,一个计值可以被看作是表达式-环境对之间的关系。这种表达式与环境对被作为配置(configuration)看待,并被表示为$\langle e,A\rangle$,其中 e 为表达式,A 为环境。环境的结构就是一个带类型的变量到表达式的绑定的列表,可以形式地定义如下:

$A::=[]\mid[x:t\mapsto e]A$

通常在一个含有多于一个绑定的环境中,绑定之间用逗号而不是方括号分隔。

LAZY-PCF+SHAR 的操作语义被定义为自然语义[16,17],它定义了在程序和它的借助于推理和公理而得到的最终值之间的关系。

展示计值过程中的中间步骤序列没有太大意义,因为表达式可以直接定值到最终值。这种语义风格被称为"一步(one step)"或"大步骤(big step)"语义。用这种语义定义的计值关系方面的定理,可以通过对计值推导过程的步骤数进行归纳的方法加以证明。

3　语言 LAZY-PCF+SHAR

LAZY-PCF+SHAR 是函数式语言 PCF(Programming language for Computable Functions)的一个惰性版本,另外增加了显式替换(explicit substitution),以便形式化惰性计算(lazy-evaluation)的语义。它的语义被定义为自然操作语义[17]风格中的推理规则或被称为演绎定义系统的大步骤(big step)语义。

通常,这类性质的验证是靠手工在纸上完成的,但近年来计算机理论工作者试图用交互式定理证明器来完成验证工作。本文用 Plastic 系统来尝试这一工作,从而说明 Plastic 系统在辅助推理方面是足够强大的。这也构成了我们研究的另一个领域[18]的又一个案例。

3.1　语言 LAZY-PCF+SHAR 的语法

LAZY-PCF+SHAR 的语法如图 3 所示,它包含常量、变量、条件式、λ 抽象、原始递归函数、函数应用、μ 操作和闭包。这就构成了一个惰性函数式语言。

类型:

$$t = nat \mid bool \mid t_1 \to t_2$$

表达式:

$$e = 0 \mid true \mid false \mid x \mid succ(e) \mid pred(e) \mid iszero(e) \mid if(e_1, e_2, e_3) \mid \lambda x:t.e \mid e_1\ e_2 \mid \mu x:t.e \mid \langle e,[x:t\mapsto e_1]\rangle$$

图 3　LAZY-PCF+SHAR 的语法

3.2　语言 LAZY-PCF+SHAR 的操作语义

$$C0:\frac{}{\vdash o:nat} \quad CT:\frac{}{\vdash ture:bool}$$

$$CF:\frac{}{\vdash false:bool} \quad VAR:\frac{}{\Gamma(x)=t\vdash x:t}$$

$$CS:\frac{\Gamma\vdash e:nat}{\vdash succ(e):nat} \quad CP:\frac{\Gamma\vdash e:nat}{\vdash pred(e):nat}$$

$$CZ:\frac{\Gamma\vdash e:nat}{\vdash iszero(e):bool} \quad COND:\frac{\Gamma\vdash e_1:bool \quad \Gamma\vdash e_2:t \quad \Gamma\vdash e_3:t}{\Gamma\vdash if(e_1,e_2,e_3):t}$$

$$ABS:\frac{\Gamma[s/x]\vdash e:t}{\Gamma\vdash\lambda x:s.\,e:s\to t} \quad APP:\frac{\Gamma\vdash e_1:s\to t \quad \Gamma\vdash e_2:s}{\Gamma\vdash e_1\,e_2:t}$$

$$REC:\frac{\Gamma[s/x]\vdash e:t}{\Gamma\vdash\mu x:s.\,e:s\to t} \quad CLO:\frac{\Gamma\vdash e_1:s \quad \Gamma[s/x]\vdash e:t}{\Gamma\vdash\langle e,[x:s\mapsto e_1]\rangle:t}$$

图 4　类型规则

图 4 列出了该语言的类型判定规则,其中 Γ 是一个类型环境,它是一个从变量到类型的映射。$\Gamma[s/x]$表示一个与 Γ

相似的环境，它与 Γ 的不同之处仅在于变量 x，它将变量 x 绑定到类型 s。我们说表达式 e 在环境 Γ 中具有类型 t，如果 $\Gamma \vdash e : t$ 可以用图 4 中的类型判定规则推出。

$$\text{OS\_C:}\ \frac{}{\langle 0,A\rangle \downarrow \langle 0,A\rangle} \qquad \text{OS\_CT:}\ \frac{}{\langle true,A\rangle \downarrow \langle true,A\rangle}$$

$$\text{OS\_CF:}\ \frac{}{\langle false,A\rangle \downarrow \langle false,A\rangle} \qquad \text{OS\_L:}\ \frac{}{\langle \lambda x:t.e,A\rangle \downarrow \langle \lambda x:t.e,A\rangle}$$

$$\text{OS\_P0:}\ \frac{\langle e,A\rangle \downarrow \langle 0,A'\rangle}{\langle pred(e),A\rangle \downarrow \langle 0,A'\rangle} \qquad \text{OS\_P:}\ \frac{\langle e,A\rangle \downarrow \langle succ(e'),A'\rangle}{\langle pred(e),A\rangle \downarrow \langle e',A'\rangle}$$

$$\text{OS\_ZT:}\ \frac{\langle e,A\rangle \downarrow \langle 0,A'\rangle}{\langle iszero(e),A\rangle \downarrow \langle true,A'\rangle} \qquad \text{OS\_ZF:}\ \frac{\langle e,A\rangle \downarrow \langle succ(e'),A'\rangle}{\langle iszero(e),A\rangle \downarrow \langle false,A'\rangle}$$

$$\text{OS\_S:}\ \frac{\langle e,A\rangle \downarrow \langle e',A'\rangle}{\langle succ(e),A\rangle \downarrow \langle succ(e'),A'\rangle}$$

$$\text{OS\_Var1:}\ \frac{\langle e,A\rangle \downarrow \langle e',A'\rangle}{\langle x,[x:t\mapsto e]A\rangle \downarrow \langle e',[x:t\mapsto e']A'\rangle}$$

$$\text{OS\_Var2:}\ \frac{\langle y,A\rangle \downarrow \langle e',A'\rangle \wedge y\neq x}{\langle y,[x:t\mapsto e]A\rangle \downarrow \langle e',[x:t\mapsto e]A'\rangle}$$

$$\text{Rec:}\ \frac{\langle e[\mu x:t.e/x],[\mu x:s\mapsto \mu x:t.e]A\rangle \downarrow \langle e',A'\rangle}{\langle \mu x:t.e,A\rangle \downarrow \langle e',A'\rangle}$$

$$\text{OS\_Appl:}\ \frac{\langle e_1,A\rangle \downarrow \langle \lambda x:s.e,A'\rangle \quad \langle e[\pi x/x],[\pi x:s\mapsto e_2]A'\rangle \downarrow \langle e'',A''\rangle}{\langle (e_1e_2),A\rangle \downarrow \langle e'',A''\rangle}$$

$$\text{IfTrue:}\ \frac{\langle e_1,A\rangle \downarrow \langle true,A'\rangle \quad \langle e_2,A'\rangle \downarrow \langle e'',A''\rangle}{\langle if(e_1,e_2,e_3),A\rangle \downarrow \langle e'',A''\rangle}$$

$$\text{IfFalse:}\ \frac{\langle e_1,A\rangle \downarrow \langle false,A'\rangle \quad \langle e_3,A'\rangle \downarrow \langle e'',A''\rangle}{\langle if(e_1,e_2,e_3),A\rangle \downarrow \langle e'',A''\rangle}$$

图 5　LAZY-PCF+SHAR 的操作语义

图 5 给出了 LAZY-PCF+SHAR 的操作语义。另外，有一个表达式的集合，它给出了该语言的表达式的规范形式(normal form)。这些规范形式如下所示：

$NF = 0 \mid true \mid false \mid succ^n(0) \mid F$

$F = \lambda x : t.\ e \mid \langle F, [x : t \mapsto e_1] \rangle$

4　LAZY-PCF+SHAR 的语义性质验证

我们首先使用 LF 作为元语言来表示目标语言 LAZY-PCF+SHAR 的各个组成部分，然后进行相关性质验证。但是在我们的下述编码中，为了更清楚地反映显式替换(explicit substitution)，虽然 LF 也具有与我们在文[19]和文[20]介绍的关于 Haskell 语言的高阶特性和列表内涵特性那样的性质，但在此我们并不强调 LF 在这两个方面的应用。

4.1　从 LAZY-PCF+SHAR 的表达式和类型到 LF 表达式的翻译

4.1.1　LAZY-PCF+SHAR 语法的归纳定义

图 6 中的模块 Syntax 是我们利用 LF 的归纳定义机制对 LAZY-PCF+SHAR 语法的归纳定义。

```
> module Syntax where;
(*************************************************)
(*syntax.lf:LAZY-PCF+SHAR 的语法的归纳定义,      *)
(* 包含类型、变量和项                            *)
(*************************************************)
> import Pi;                          --导入 Pi 类型
> import Nat;                         --导入自然数类型
> Inductive
>          [Ty : Type]                --新定义的类型 Ty
>   Constructors
>           [ nat_Ty : Ty]                -- 自然数
>           [ bool_Ty : Ty]               -- 布尔值
>           [ arr : (e1: Ty)(e2: Ty)Ty];-- 函数类型
```

```
> Inductive
>        [Vari : Type]              --新定义的类型 Vari
>    Constructors
>       [X : (i: El Nat)Vari];     --Vari 构造子

> Inductive
>          [Tm : Type]              --新定义的类型 Tm
>  Constructors
>  [o : Tm]                          --零
>  [ttt : Tm]                        --逻辑真值
>  [fff : Tm]                        --逻辑假值
> [abs : (v: El Vari)(t : El Ty)(tm1 : Tm)Tm]
                                     --λ 抽象
> [appl : (tm1 : Tm)(tm2 : Tm)Tm]
                                     --函数应用
> [cond : (tm1 : Tm)(tm2 : Tm)(tm3 : Tm)Tm]
                                     --条件式
> [var : (v : El Vari)Tm]             --变量
> [suc : (tm1 : Tm)Tm]               --后继操作
> [prd : (tm1 : Tm)Tm]               --前趋操作
> [is_o : (tm1 : Tm)Tm]              --零测试
> [fix : (v: El Vari)(t1: El Ty)(tm1: Tm)Tm]
                                     --不动点操作
> [clos: (tm1 : Tm)(v: El Vari)(t: El Ty)(tm2 : Tm)Tm];
                                     --闭包
```

图 6

在上面的描述中，以(*开头的行是注释行，以>开头的行是 Plastic 的代码行，--后面的字符串为相关说明。从上述定义可以看出，所有的定义都是归纳定义，这也为后面的基于结构归纳的证明奠定了基础。

4.1.2　操作语义规则的翻译

图 7 的模块 OSrules 是操作语义规则在 LF 的实现 Plastic 中的表示。

```
> module OSrules where;
(*************************************************)
(* OSrules.lf                                    *)
(* 该模块包含 LAZY-PCF+SHAR 的操作语义规则的定义*)
(* 以及 Ap 函数的定义和相关性质的证明            *)
(*************************************************)
> import Typecheck;     --导入类型检查模块
> import Rename;        --导入重命名模块

(*************************************************)
(* OScons 是一种缩写形式                         *)
> [OScons = [v: El Vari][t : El Ty][e : El Tm]
>        [A: El OS_env]
>        cons VTT (pair VT Tm (pair Vari Ty v t) e) A];
(*************************************************)
(* Ap:归纳定义被 Ap 函数刻划的关系               *)
(* (Ap a F A F' n t) <--> Ap(F,a)=<F',[n:t->a]>  *)
(*************************************************)
> Inductive
>       [Ap : (ptm1,ptm2: El Tm)(pose: El OS_env)
>           (ptm3: El Tm)
>           (pv: El Vari)(pt: El Ty) El Prop]
>           Relation_LE
>    Constructors
>     [Ap_abs: (nv,v:El Vari)(t: El Ty)
>     (a,e,ne: El Tm)(A: El OS_env)
>     (p1: El (Prf(not (member Vari  nv (OS_Dom A)))))
>     (p2: El (Prf(Rename nv v e ne)))
>     Prf( Ap a (abs v t e) A ne nv t)]
>     [Ap_clos: (n,v:El Vari)(s,t: El Ty)
>     (a,e,ne,e1: El Tm)(A: El OS_env)
>     (p1: Prf(Ap a e (OScons v s e1 A) ne n t))
>     Prf (Ap a (clos e v s e1) A
>           (clos ne v s e1) n t)];
```

```
(*************************************************)
(*操作语义的定义，它的构造子分别与同名的前面关于类*)
(*型规则的定义相对应                             *)
(*<e,A> ↓ <e',A'>                                *)
(*************************************************)
> Inductive
>    [OSred : (conf1:El Config)(conf2: El Config)
>     El Prop] Relation_LE
>   Constructors
>    [OS_C0: (A: El OS_env)Prf(OSred (cfg o A)
>    (cfg o A))]
>    [OS_CT: (A: El OS_env)Prf(OSred (cfg ttt A)
>     (cfg ttt A))]
>    [OS_CF: (A: El OS_env)Prf(OSred (cfg fff A)
>     (cfg fff A))]
>    [OS_L: (A: El OS_env)(e : El Tm)(t: El Ty)
>     (x: El Vari)Prf(OSred (cfg (abs x t e) A)
>     (cfg (abs x t e)  A))]
>    [OS_P0: (A,A1: El OS_env)(e: El Tm)
>            (p1: Prf(OSred (cfg e A) (cfg o A1)))
>             Prf(OSred (cfg (prd e) A) (cfg o A1))]
>    [OS_P: (A,A1: El OS_env)(e,e1: El Tm)
>          (p1: Prf(OSred (cfg e A) (cfg (suc e1) A1)))
>          Prf(OSred (cfg (prd e) A) (cfg e1 A1))]
>    [OS_ZT: (A,A1: El OS_env)(e: El Tm)
>         (p1: Prf(OSred (cfg e A) (cfg o A1)))
>          Prf(OSred (cfg (is_o e) A) (cfg ttt A1))]
>    [OS_ZF: (A,A1: El OS_env)(e,e1: El Tm)
>         (p1: Prf(OSred (cfg e A) (cfg (suc e1) A1)))
>         Prf(OSred (cfg (is_o e) A) (cfg fff A1))]
>    [OS_S: (A,A1: El OS_env)(e,e1: El Tm)
>        (p1: Prf(OSred (cfg e A) (cfg e1 A1)))
>         Prf(OSred (cfg (suc e) A) (cfg (suc e1) A1))]
>   [OS_Var1: (A,A1: El OS_env)
>       (e,e1: El Tm)(t:El Ty)(x: El Vari)
>       (p1: El (Prf(not (member Vari x (OS_Dom A)))))
>       (p2: Prf(OSred (cfg e A) (cfg e1 A1)))
>      Prf(OSred (cfg (var x) (OScons x t e A))
>                (cfg e1 (OScons x t e1 A1)))]
>   [OS_Var2: (A,A1: El OS_env)(e,e1: El Tm)
>       (t:El Ty)(x,y: El Vari)
>       (p1 : El (Prf (not (Eq Vari x y))))
>       (p2: El (Prf(not (member Vari x (OS_Dom A)))))
>       (p3: Prf(OSred (cfg (var y) A) (cfg e1 A1)))
>        Prf(OSred (cfg (var y) (OScons x t e A))
>                  (cfg e1 (OScons x t e1 A1)))]
>           [OS_Appl: (A,A1,A2: El OS_env)
>                (e1,e2,en1,en2,enf: El Tm)
>                (t:El Ty)(n: El Vari)
>            (p1 : Prf (OSred (cfg e1 A) (cfg en1 A1)))
>            (p2: El (Prf(Ap e2 en1 A en2 n t)))
>            (p3: Prf(OSred (cfg (clos en2 n t e2) A1)
>             (cfg enf A2)))
>         Prf(OSred (cfg (appl e1 e2) A) (cfg enf A2))]
> .....
```

图 7

4.2 语义性质验证举例

利用上面的定义，就可以对语义性质展开证明。我们已经证明了与 LAZY-PCF+SHAR 相关的一系列性质。对这些性质的成功证明也反映了 Plastic 系统的能力。首先，我们需要引入一些定义。

定义 1(环境的类型上下文 Context)

$Context([]) = \perp$

$Context([x : t \mapsto e]A) = Context(A)[t/x]$

这里⊥是一个映射，它对每个变元都是无定义的。

定义 2(定义域 Dom)　Dom(H)被用于表示上下文 H 的定义域，且 Dom(A)被用于表示在操作语义环境 A 中绑定的变量集合。

定义 3(环境类型上下文的扩展(Context extension))

(1)H 是 H 自身的扩展

(2)若 H′是 H 的扩展，且 $x \notin Dom(H')$，则 $H'[t/x]$是 H 的扩展。

我们也可以更加形式化地定义如下：

(1)$H \vdash H$

(2)若 $H' \vdash H$，且 $x \notin Dom(H')$，则 $H'[t/x] \vdash H$。

下面给出了一种用 LF 描述的语义性质。

性质 1

$Ap(a,fun,A) = \langle b,[n : t \mapsto a]\rangle \rightarrow n \notin Dom(A)$

该性质表明，在对表达式 a 施用函数 fun 的时候，若用环境变量 n 标记表达式 a，则 n 应该不在施用之前的环境中。我们给其取名为 ApNewVar，并用 Plastic 给出其形式化的计算机辅助证明如图 8(列出这个证明的代码，只是想让读者感受一下证明的轮廓。若想理解其含义和用法，请参照 Plastic 系统使用说明。)：

```
> Claim ApNewVar : (a,fun,b: El Tm)(A: El OS_env)
>                  (n : El Vari)(t: El Ty)
>                  (p1: El (Prf (Ap a fun A b n t)))
>                 Prf (not (member Vari n
>                      (OS_Dom A)));
> Intros a fun b A n t p1;
> Refine E_Ap ([a,fun:El Tm][A: El OS_env][b:El Tm]
>              [n: El Vari][t:El Ty]
>            Prf(not (member Vari n (OS_Dom A)))) ? ?
>                 a fun A b n t p1;
> 2 Intros nv;
> Intros v  t1 tm1 e ne A1 pr1 pr2;
> Refine pr1;
> ReturnAll;
> Intros n1 v s t1 a1 e ne e1 A1 pr1 pr2;
> Refine LL;
> Intros H;
> Refine App ? ? pr2 ?;
> Refine App ? ? (p_inr ? ? ) ?;
> Refine H;
> ReturnAll;
> ApNewVar;
```

图 8

说明：上面证明中的符号？只是起占位符的作用，没有其它含义。

下面是对主要的定理(主题归约(subject reduction)定理)的证明。在此之前，我们还需要引入一些必要的定义和引理。

定义 4(有效环境(Valid environments))

(1)[ ]是有效的环境。

(2)若 A 是一个有效的环境，并且 $Context(A) \vdash e : t$，则 $[x : t \mapsto e]A$ 也是一个有效的环境。

该定义意味着，如果一个表达式中的自由变元在一个有效的环境中被绑定，那么它在环境的剩余部分中也一定是被绑定的。这是由于下面的事实造成的，即：如果一个表达式在某个类型上下文中具有一个类型，那么该表达式的自由变元就会出现在该类型上下文的论域中。有效环境的定义可以被扩展到格局，只要该格局的环境是有效的，并且该格局的表达式具有该环境的类型上下文中的某个类型。

定义 5(有效格局(Valid configurations))　若 A 是一个有效的环境，并且对某个 t 有：$Context(A) \vdash e : t$，则称$\langle e,A\rangle$是一个有效的格局。

(下转第 69 页)

ences Institute, January, 2004. To appear in Frontiers in Distributed Sensor Networks, Iyengar S S and Brooks R R, ed
3 Akkaya K, Younis M. A Survey of Routing Protocols in Wireless Sensor Networks. in the Elsevier Ad Hoc Network Journal (to appear)
4 Chong C Y, Kumar S. Sensor networks: Evolution, opportunities, and challenge. Proceedings of the IEEE, 2003, 91(8):1247～1256
5 Choi W, Shah P, Das S K. A Framework for Energy-Saving Data Gathering Using Two-Phase Clustering in Wireless Sensor Networks. Proceedings of Mobile and Ubiquitous Systems (Mobiquitous): Networking and Services, Boston, Aug. 2004
6 Shah R, Rabaey J. Energy Aware Routing for Low Energy Ad Hoc Sensor Networks. Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC), Orlando, FL, March 2002
7 Wu K, Gao Y, Li F, et al. Lightweight Deployment-Aware Scheduling for Wireless Sensor Networks. To appear in ACM/Kluwer MONET Journal, Special Issue on Energy Constraints and Lifetime Performance in Wireless Sensor Networks
8 Sichitiu ML. Cross-Layer Scheduling for Power in Wireless Sensor Networks. Proceedings INFOCOM'04, Hong Kong, China, March 2004
9 Hong Xiaoyan, Gerla M, Wang Hanbiao, et al. Load balanced, energy aware communications for Mars sensor networks. In: IEEE Aerospace Conference Proceedings, vol. 3, 2002, 3. 3-1109-3-1115
10 Huang Shih-Chang, Jan Rong-Hong. Energy-aware, load balanced routing schemes for sensor networks. In: Proceedings of Tenth International Conference on Parallel and Distributed Systems (ICPADS 2004), 2004. 419～425
11 Akkaya K, Younis M. Energy-aware routing of time-constrained traffic in wireless sensor networks. International Journal of Communication Systems, Special Issue on Service Differentiation and QoS in Ad Hoc Networks, 2004, 17(6): 663～687
12 He T, et al. SPEED: A stateless protocol for real-time communication in sensor networks. Proceedings of International Conference on Distributed Computing Systems, Providence, RI, May 2003
13 Blake S, Black D, Carlson M, et al. An architecture for differentiated services. RFC 2475, IETF, Dec. 1998
14 Nichols K, Blake S, Baker F, et al. Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers. RFC 2474, IETF, Dec. 1998
15 陈敏. OPNET 网络仿真. 北京: 清华大学出版社, 2004. 352

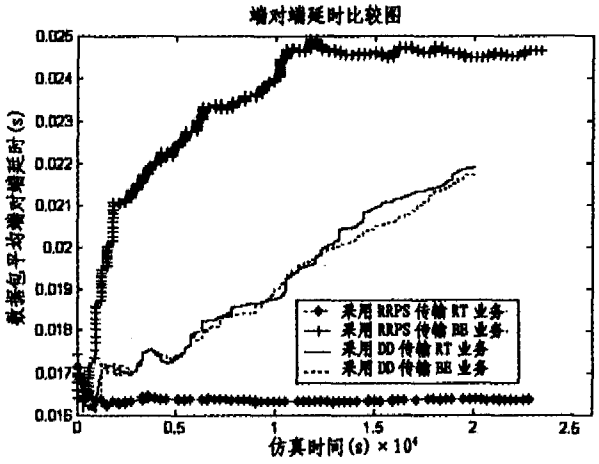

图 15 RT 与 BE 业务量之比为 1∶3 时分别采用 RRPS 与 DD 机制的端对端延时比较

(上接第 16 页)

这个概念是非常重要的，因为操作语义只有被应用到有效的格局时才能产生有意义的结果。

引理 1 若⟨e,A⟩↓⟨e',A'⟩，

则 Context(A') ⊢ Context(A)。

证明：对⟨e,A⟩↓⟨e',A'⟩的过程中可能用到的推理规则分情形进行归纳证明，即可得证。

该引理表明，一个格局⟨e,A⟩计值到另一个格局⟨e',A'⟩，则第二个环境的类型上下文扩展了第一个环境的类型上下文。

定理 1(主题归约定理)

若 Context(A) ⊢ e : t, A 是有效的(valid)，且⟨e,A⟩↓⟨e',A'⟩，则 Context(A') ⊢ e' : t，且 A'是有效的。

证明：对⟨e,A⟩↓⟨e',A'⟩的过程中可能用到的推理规则分情形进行归纳证明，对 OS_Appl 等情形还需要用到引理 1。

该定理表明，计值过程保持类型不变。也就是说，一个表达式的类型和它的规范形式(normal form)的类型是一样的。

结束语 通过前面对语义性质的描述及证明，我们已经看到，对此类论域，Plastic 系统可被直接用于形式化定义以及对性质的证明工作。事实上，我们已经完成了对语言 LAZY-PCF+SHAR 的所有感兴趣的语义性质的形式化的机器验证(感兴趣的读者可以发送 email 来索取证明代码)。由此可见，Plastic 适合直接处理这类问题。换句话说，对这类论域，我们不需要提供其它的界面。经过这个案例研究之后，我们更加确信 Plastic 在作为底层的定理证明器方面是足够强大的。后续的工作将围绕网络协议性质的形式化与计算机辅助推理的应用而展开，我们期待更加庞大的应用。

致谢 感谢英国伦敦大学罗朝晖(Prof. Zhaohui Luo)教授、英国杜伦大学保罗·卡拉汉博士(Dr. Paul Callaghan)的帮助和指导，感谢来自欧盟和杜伦大学的资助。

参 考 文 献

1 Seaman J, Iyer S P. An Operational Semantics of Sharing in Lazy Evalutation. Journal of Computer Programming, 1996, 27(3): 289～322
2 Abadi M, Cardelli L, Curien P-L, et al. Explicit Substitutions. Journal of Functional Programming. Preliminary version in POPL 1990, 1991, 1(4): 375～416
3 Field J. On laziness and optimality in lambda interpreters: Tools for specification and analysis. In: Proceedings of the seventeenth symposium on Principles of Programming Languages. San Francisco, 1990. 1～15
4 Seaman J, Felty A. Proving Properties About a Lazy Functional Language with the Coq Proof Development System, 1993. Http://www.site.uottawa.ca/～afelty/bib.html
5 Luo Z. Computation and Reasoning: A Type Theory for Computer Science. Oxford University Press, 1994
6 Callaghan P C, Luo Z. Plastic: an Implementation of Typed LF with Coercive Subtyping and Universes. Journal of Automated Reasoning, special issue on Logical Frameworks, 2000
7 Paulson L-C. Isabelle: A Generic Theorem Prover. Lecture Notes in Computer Science, 1994, 828: xvii + 321
8 Isabelle. Isabelle WWW Page. Http://www.cl.cam.ac.uk/Research/HVG/Isabelle
9 陈火旺，庞建民. 直觉主义逻辑、类型理论与软件形式化开发. 计算机科学，1988, 01
10 Per Martin-Löf. Intuitionistic Type Theory. Napoli: Bibioplois, 1984. Notes of Giowanni Sambin on a series of lectues given in Padova
11 Nordström B, Petersson K, Smith J M. Programming in Martin-Löf's Type Theory. International Series of Monographs on Computer Science. New York, NY: Oxford University Press, 1990, 7
12 Curry H B, Feys R. Combinatory Logic. North-Holland, 1958
13 Howard W A. The Formulae-as-types Notion of Construction. Hindley J, Seldin J. Editors To H. B. Curry: Essays on Combinatory Logic. Academic Press, 1980
14 Girard J-Y. Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur: [Ph D thesis]. Université Paris VII, 1972
15 Harper R, Honsell F, Plotkin G. A Framework for Defining Logics. Proc 2nd Ann Symp on Logic in Computer Science IEEE, 1987
16 Plotkin G D. A structural approach to operational semantics: [Tech Rep]. DAIMI FN-19. Computer Science Department, Aarhus University, Aarhus, Denmark, 1981
17 Kahn G. Natural Semantics. In: Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS), Springer-Verlag, of Lecture Notes in Computer Science. 1987, 247: 22～39
18 Pang J, Callaghan P, Luo Zhaohui. LFTOP: An LF-based approach to domain-specific reasoning. Journal of Computer Science and Technology, 2005, 20(4): 526～535
19 庞建民，赵荣彩. Haskell 语言的列表内涵特性及其应用. 计算机工程与应用，2005, 41(4): 99～101
20 庞建民，赵荣彩，王怀民. Haskell 语言的高阶特性及其应用. 计算机科学，2005, 32(6): 167～168, 198
21 周建涛，史美林，叶新铭. 工程流过程建模中的形式化验证技术. 计算机研究与发展，2005, 42(1)
22 Fu Yuxi. Semantics of constructions (I)-The traditional approach. Journal of Computer Science and Technology, 2001, 16(1): 13～24