基于类型理论的递归元程序设计*

谭庆平 陈火旺

(长沙工学院计算机科学系,长沙 410073)

摘要 本文提出在LF类型理论中定义一组相互递归类型的方法,并对递归类型赋予操作语义.这样,递归类型不仅可以表示通常的递归数据结构,还可描述一般的递归问题求解、递归证明构造和递归程序构造过程.

关键词 类型理论,证明开发环境,递归.

逻辑程序设计语言(如Prolog和λProlog)通过对逻辑公式进行操作解释使得逻辑程序既具有清晰的说明语义,又有操作语义,便于计算机执行.有鉴于此,F. Pfenning[3]提出对类型理论中的类型表达式赋予操作语义.这样,类型表达式即构成可执行的(元级)程序.因为类型理论具有丰富的表达能力,例如,Edinburgh LF(Logical Framework)可以表达多种逻辑系统的语法、规则及证明.因此,这种基于类型理论的程序设计风格可以对多个逻辑系统的定理证明过程进行元级编程,并保证元程序的正确性和语义清晰性.

众所周知,递归(循环)是一种重要的程序设计机制.因此,本文提出一种在LF类型理论中定义一组相互递归类型的方法,并对递归类型赋予操作语义.这样,递归类型不仅可以表示通常的递归数据结构(如Nat,List,Tree等),还可以描述递归问题求解、递归证明构造和递归程序构造过程.

1 递归类型的定义

在类型理论中定义递归类型的方法可见于文献[1].本节对Coquant和Paulin的方法进行拓广,给出在LF类型理论中定义一组相互递归类型的方法.

本文所使用的类型理论在LF的基础上增加了类型构造子Σ(dependent product或strong sum),以丰富元语言的表达能力.

以下给出完整的语法定义及一部分类型规则:

标记(*signature*) $\Sigma ::= [] \mid \Sigma \oplus c:K \mid \Sigma \oplus d:A$

上下文(*context*) $\Gamma ::= [] \mid \Gamma \oplus \alpha:K \mid \Gamma \oplus x:A$

种类(*kind*) $K ::= TYPE \mid \Pi x:A.K$

---

* 本文1991-05-05收到,1992-05-11定稿

作者谭庆平,29岁,1992年博士毕业于长沙工学院,主要研究领域为计算机科学理论,软件工程.陈火旺,58岁,教授,主要研究领域为计算机科学理论,逻辑学,软件工程.

本文通讯联系人:谭庆平,长沙 410073,长沙工学院计算机科学系

类型(*type*)　　$A::=c|\alpha|AM|\Pi x:A.B|\Sigma x:A.B|\lambda x:A.B$

项(*term*)　　$M::=d|x|MN|\lambda x:A.M|(M,N)|fst(M)|snd(M)$

其中 c,d 分别表示类型常元和个体常元，α,x 分别表示类型变元和个体变元. 其它记号与 *LF* 类似.

Σ 引入规则：$\dfrac{\Gamma\vdash_\Sigma M:A \quad \Gamma\vdash_\Sigma N:[M/x]B}{\Gamma\vdash_\Sigma(M,N):\Sigma x:A.B}$

Σ_消去规则：$\dfrac{\Gamma\vdash_\Sigma M:\Sigma x:A.B}{\Gamma\vdash_\Sigma fst(M):A} \qquad \dfrac{\Gamma\vdash_\Sigma M:\Sigma x:A.B}{\Gamma\vdash_\Sigma snd(M):[fst(M)/x]B}$

现在开始讨论递归类型(族)的定义方法.

定义 1.1. 给定类型 A,B. A 在 B 中呈正出现(*occurs positively*)当且仅当：

(1)$A\equiv B$；或

(2)存在类型 B_0 及 B_1，使得 $B\equiv\Pi x:B_0.B_1$，且 A 在 B_0 中不出现，且 A 在 B_1 中呈正出现；或

(3)存在类型 B_0 及 B_1，使得 $B\equiv\Sigma x:B_0.B_1$，且 A 在 B_0 或 B_1 中出现，且若 A 在 B_i 中出现，则必呈正出现($i=0,1$). □

定义 1.2. (递归类型族)

假定 *IS* 为下标集. 对每个 $i\in IS$，$RAS(i)$ 为类型，也即，$RAS:IS\to TYPE$ 为递归类型族. 每个 $RAS(i)$ 定义如下：

Ind_Type $RAS(i)$ *With* $d_{i1}:A_{i1},\cdots,d_{in_i}:A_{in_i}$；

d_{ij} 称为 *RAS* 的构造子，A_{ij} 则称为 *RAS* 的构造子类型. 每个 A_{ij} 满足以下语法限制：

(1)$A_{ij}\equiv\Pi x_{j1}:B_{j1},\cdots\Pi x_{jk}:B_{jk}.RAS(i)$.　$(k_j\geqslant 0)$

(2)如果 $RAS(p)$ 在 B_m 中出现，则 $RAS(p)$ 必呈正出现.　$(p\in IS,m=1,\cdots,k_i)$

(3)A_{ij} 中不出现自由变元，即 $FV(A_{ij})=\{\}$. □

递归类型形成规则(RAS_F)：$\dfrac{\Gamma\vdash_{\Sigma(RAS)} IS:TYPE \quad \Gamma\vdash_{\Sigma(RAS)} i:IS}{\Gamma\vdash_{\Sigma(RAS)} RAS(i):TYPE}$

递归类型引入规则(RAS_I)：$\dfrac{\Gamma\vdash_{\Sigma(RAS)} i:IS}{\Gamma\vdash_{\Sigma(RAS)} d_{ij}:A_{ij}}$

$\Sigma(RAS)$ 表示 *RAS* 的定义在 Σ 中出现.

按定义 1.2，由(RAS_I)可得到以下派生规则(RAS_I')：

$$\frac{\Gamma\vdash_{\Sigma(RAS)} i:IS \qquad \Gamma\vdash_{\Sigma(RAS)} M_j:[M_1/x_1,\cdots,M_{j-1}/x_{j-1}]B_j \quad (j=1,\cdots,k)}{\Gamma\vdash_{\Sigma(RAS)} d_{ij}(M_1,\cdots,M_{ki}):RAS(i)}$$

为了描述递归类型消去规则，需要以下术语：

定义 1.3. (参量表)

递归类型 *RAS* 的定义如前所述. 对 *RAS* 的每个构造子 d_{ij} 定义其参量表和递归参量表如下：

(1)d_{ij} 的参量表为$[\langle x_1,B_1\rangle,\cdots,\langle x_{ki},B_{ki}\rangle]$；

(2)d_{ij} 的递归参量表为$[B_{p1},\cdots,B_{pq}]$，其中：

$\{B_{p1},\cdots,B_{pq}\}=\{B_j|1\leqslant j\leqslant k,RAS$ 在 B_j 中出现$\}$，且 $p_m<p_{m+1}(m=1,\cdots,q-1)$. □

对递归参量表中的任意类型 B_j，显然 $FV(B_j)\subseteq\{x_1,\cdots,x_{j-1}\}$. 我们以 $\Gamma(B_j)$ 表记上下文 $[x_1:B_1,\cdots,x_{j-1}:B_{j-1}]$. 由于 $x_1,\cdots,x_{j-1}$ 为约束变元，因此不妨假设对任意上下文 Γ，$dom(\Gamma)\cap dom(\Gamma(B_j))=\{\}$.

定义 1.4. 给定 Γ,Σ 及类型 P. 对于递归参量表中的任一类型 B，定义类型 B^P 如下：

(1)如果 RAS 在 B 中不出现，则 $B^P=_{df}\lambda z:B.\ B$. 否则：

(2)若 $B\equiv RAS(i)$，则 $B^P=_{df}P(i)$；

(3)若 $B\equiv\Pi x:B_0.\ B_1$，则 $B^P=_{df}\lambda z:B.\ (\Pi x:B_0.\ B_1^P(z(x)))$；

(4)若 $B\equiv\Sigma x:B_0.\ B_1$，则 $B^P=_{df}\lambda z:B.\ (\Sigma x:B_0^P(fst(z)).\ B_{11}^P(snd(z)))$，这里 $B_{11}\equiv[fst(z)/x]B_1$. □

关于定义 1.4 有结论：

定理 1.5. 若 $\Gamma\vdash_{\Sigma(RAS)}P:\Pi i:IS.\ RAS(i)\to TYPE$，则：

$\Gamma\oplus\Gamma(B)\vdash_{\Sigma(RAS)}B^P:B\to TYPE$.

证明：按上述定义，对 B 的结构进行归纳，易证. □

定义 1.6. Γ,Σ,P 如上所述. 对 $RAS(i)$ 的任一构造子 d，假定其参量表和递归参量表分别为 $[\langle x_1,B_1\rangle,\cdots,\langle x_k,B_k\rangle]$ 和 $[B_{p1},\cdots,B_{pq}]$. 定义类型 d^P 如下：

$$d^P=_{df}\Pi x_1:B_1.\cdots\Pi x_k:B_k.\ B_{p1}^P(x_{p1})\to\cdots\to B_{pq}^P(x_{pq})\to P(i)(d(x_1,\cdots,x_k))$$ □

定理 1.7. 若 $\Gamma\vdash_{\Sigma(RAS)}P:\Pi i:IS.\ RAS(i)\to TYPE$，则：

$$\Gamma\vdash_{\Sigma(RAS)}d^P:TYPE.$$

证明：利用定理 1.5 及规则(RAS_I)，易证. □

定义 1.8. Γ,Σ,P 同上. 给定项 M，对 RAS 的任一构造子 d 的递归参量表中的所有 B，定义项 B^M 如下：

(1)如果 RAS 在 B 中不出现，则 $B^M=_{df}\lambda z:B.\ z$. 否则

(2)若 $B\equiv RAS(i)$，则 $B^M=_{df}M(i)$；

(3)若 $B\equiv\Pi x:B_0.\ B_1$，则 $B^M=_{df}\lambda z:B.\ (\lambda x:B_0.\ B_1^M(z(x)))$；

(4)若 $B\equiv\Sigma x:B_0.\ B_1$，则 $B^M=_{df}\lambda z:B.\ (B_0^M(fst(z)),B_{11}^M(snd(z)))$，这里 $B_{11}=[fst(z)/x]B_1$. □

类似于定理 1.5，我们有：

定理 1.9. 若 $\Gamma\vdash_{\Sigma(RAS)}P:\Pi i:IS.\ RAS(i)\to TYPE,\Gamma\vdash_{\Sigma(RAS)}M:\Pi i:IS.\ \Pi x:RAS(i).\ P(i,x)$，则：$\Gamma\oplus\Gamma(B)\vdash_{\Sigma(RAS)}B^M:\Pi y:B.\ B^P(y)$.

证明：与定理 1.5 的证明相同. □

利用上述定义，可给出递归类型消去规则(RAS_E)：

$$\frac{\Gamma\vdash_{\Sigma(RAS)}P:\Pi i:IS.\ RAS(i)\to TYPE\qquad \Gamma\vdash_{\Sigma(RAS)}M_{ij}:d_{ij}^P\quad(j=1,\cdots,n_i)}{\Gamma\vdash_{\Sigma(RAS)}rec_{RAS,P}(M_{i1},\cdots,M_{in_i}):\Pi i:IS.\ \Pi x:RAS(i).\ P(i,x)}$$

其中递归算子 rec 的意义以下归约规则给出：

以 REC 记 $rec_{RAS,P}(M_{i1},\cdots,M_{in_i})$，那么：

$$REC(i)(d_{ij}(N_1,\cdots,N_{ik}))\ r\ M_{ij}(N_1,\cdots,N_{ik},B_{p_1}^{REC}(N_{p_1}),\cdots,B_{p_q}^{REC}(N_{p_q})),$$

这里$[\langle x_1,B_1\rangle,\cdots,\langle x_k,B_k\rangle]$和$[B_{p_1},\cdots,B_{p_q}]$分别为构造子$d_{ij}$的参量表和递归参量表.

最后，我们指出，文献[1]提出的递归类型定义方法是上述方法当下标集为单元素集时的特例.

2　类型作为操作

本节概述对类型表达式进行操作解释的主要思想，详见文献[3,5].

非形式地，如果当前目标是$z\in\sum x:A.B$，那么解释器将z构造为(x,y)，并依次求解子目标$x\in A$和$y\in B$. 这一行为的正确性由定义1.3中的$\sum$—引入规则保证. 又如，对目标$z\in\Pi x:A.B$，解释器将z构造为$\lambda x:A.y$，在$x\in A$的假设下求解子目标$y\in B$.

为了描述解释器的上述求解行为，我们引进证明目标(G)和定义子句(D)：

$G::=M\in A\mid M=N\mid A=B\mid$

$\quad TRUE\mid G_1\wedge G_2\mid G_1\vee G_2\mid\forall x:A.G\mid\exists x:A.G$

$D::=M\in A\mid G\rightarrow M\in C\mid D_1\wedge D_2\mid\forall x:A.D$

序偶$(D;G)$表示：在假设D下求解目标G. 这种序偶构成解释器操作的基本单位. 因为D可以描述G的求解方法，所以D也称为G的求解程序.

本文将TML解释器描述为关于证明目标和求解程序的非确定转换系统. $R:(D;G)\Rightarrow(D';G_1,\cdots,G_n)$表示解释器使用规则$R$将$D$和$G$分别转换为$D'$和子目标序列$G_1,\cdots,G_n$.

限于篇幅，这里只给出几条主要的转换规则，其直观意义是不言而喻的.

G_{pi}：　$(D;M\in\Pi x:A.B)\Rightarrow(D;\forall x:A.\exists y:B.(Mx=y))$；

G_{sigma}：　$(D;M\in\sum x:A.B)\Rightarrow(D;\exists x:A.\exists y:B.M=(x,y))$；

D_{pi}：　$(D,N\in\Pi x:A.B;M\in C)\Rightarrow(D,\forall x:A.(Nx\in B);M\in C)$；

D^1_{sigma}：　$(D,N\in\sum x:A.B;M\in C)\Rightarrow(D,fst(N)\in A;M\in C)$；

D^2_{sigma}：　$(D,N\in\sum x:A.B;M\in C)\Rightarrow(D,snd(N)\in[fst(N)/x]B;M\in C)$；

D_{atom}：　$(D,M'\in C';M\in C)\Rightarrow(D;C=C'\wedge M=M')$；

这里，C表示原子类型，形如$uM_1\cdots M_n$，u是类型常元或类型变元.

3　递归类型的操作语义

递归类型可以很自然地表示通常的递归数据结构，如Nat，$List$，$Tree$等.

$Ind\_Type\ Nat\ With\ 0:Nat,suc:Nat\rightarrow Nat.$

Nat的消去规则是：

$$\frac{\Gamma\vdash_\Sigma P:Nat\rightarrow TYPE\quad\Gamma\vdash_\Sigma M_0:P(0)\quad\Gamma\vdash_\Sigma M:\Pi x:Nat.P(x)\rightarrow P(suc(x))}{\Gamma\vdash_\Sigma rec_{Nat,P}(M_0,M):\Pi x:Nat.P(x)}$$

根据“公式作为类型”原理，类型P可视为以Nat为论域的断言. 于是，上述规则即可理

解为通常的归纳原理：

如果 $P(0)$ 成立，且对任意的 x，由 $P(x)$ 成立可推出 $P(suc(x))$ 成立，则：对所有的 $x\in Nat$，$P(x)$ 成立.

现在，我们利用递归类型描述递归问题求解、递归证明构造和递归程序构造过程. 为叙述简洁，本节只给出单个递归类型的操作语义. 以下的讨论可以很自然地推广至一般情形.

首先，对证明目标和定义子句进行扩充，允许递归类型作为一般类型在 D,G 中出现.

3.1 递归问题求解

先看一个简单的例子. 假定问题 P 的求解过程如下：在条件 c_0 下，d_0 为 P 的解；在条件 c 下，求解 P 需求解子问题 P_0 和 P_1，若 s_i 为 P_i 的解($i=0,1$)，则 $d(s_0,s_1)$ 构成 P 的解. P_0 可独立求解，求解 P_1 需递归地重复上述过程.

众所周知，问题可描述为类型，问题的解则对应于类型的元素. 因此，上述 P 可说明为递归类型 RA_P：

$Ind\_Type\ RA_P:TYPE\ With\ d_0:C_0\rightarrow RA_P, d:C\rightarrow A_0\rightarrow RA_P\rightarrow RA_P$;

这里 C_0,C,A_0 依次是相应于 c_0,c,P_0 的类型.

由递归类型引进规则，可导出以下转换规则：

$$G_{RA}:(D;\ M\in RA)\Rightarrow\{(D;\exists\ z_1:B_1.\ \cdots\ \exists\ z_{ki}:[z_1/x_1,\cdots,z_{ki-1}/x_{ki-1}]B_{ki}.\ (M=d_i(z_1,\cdots,z_{ki}))\mid 1\leqslant i\leqslant n\}.$$

这里 d_i 是 RA 的构造子.

G_{RA} 的直观意义是：如果当前目标是 $M\in RA$，那么不妨将 M 构造为 $d_i(z_1,\cdots,z_{k_i})$，然后分别求解 $z_j\in B_j(j=1,\cdots,k_i)$.

具体地，对于上述 RA_P，解释器将(不确定地)选取下述转换之一：

$(D;\ z\in RA_P)\Rightarrow(D;z_0\in C_0, z=d_0(z_0)\in RA_P)$,

$(D;\ z\in RA_P)\Rightarrow(D;z_0\in C, z_1\in A_0, z'\in RA_P, z=d(z_0,z_1,z')\in RA_P)$.

显然，解释器的上述行为恰好反映了 P 的求解过程. 因此，递归类型的操作语义描述了递归问题求解过程.

3.2 递归证明构造

假定谓词 P 以递归类型 Nat 的元素为参量. 当 $n=0$ 时，s_0 是 $P(0)$ 的证明；当 $n>0$ 时，若 s_{n-1} 和 s' 分别是 $P(n-1)$ 和 P' 的证明，则 $s(s_{n-1},s')$ 是 $P(n)$ 的证明.

$(D;\ z\in \Pi x:Nat.P(x))$ 表示：在假设 D 下，寻找 $P(x)$ 的证明 z. 由递归类型消去规则，可将 z 构造为 $rec(s_0,\lambda x:Nat.\lambda y:P(x).s(y,s'))$. 注意，$s_0:P(0)$，$\lambda x:Nat.\lambda y:P(x).s(y,s'):\Pi x:Nat.P(x)\rightarrow P(suc(x))$.

更一般地，由(RAS—E)可导出以下转换规则：

$G_{\Pi RA}:(D;\ M\in\Pi x:RA.P(x))\Rightarrow(D;\exists\ z_1:d_1^p.\cdots.\exists\ z_n:d_n^p.\ (M=rec(z_1,\cdots,z_n)))$.

因此，以递归类型的元素为参量的问题的解(证明)可描述为 rec 表达式. 并且，递归类型的定义形式决定了解或证明的递归结构.

3.3 递归程序构造

构造递归程序是我们研究递归类型的主要动机之一.

典型地,初始程序规范可由带类型的一阶公式$\forall x:A.\ \exists y:B.W(x,y)$描述.该公式可翻译为以下类型:

$\Pi x:A.\Sigma y:B.P(x,y)$,其中$P(x,y)$是对应于$W(x,y)$的类型表达式.

如果A是递归类型,那么目标程序y可能呈递归形式.y的构造过程对应于目标$z\in\Pi x:A.\Sigma y:B.P(x,y)$的求解过程.使用上述转换规则$G_{\Pi RA}$,解释器将$z$构造为$rec$表达式.一旦证明完成,即可从$z$中抽取目标(递归)程序:设$E_z$是$\Pi x:A.\Sigma y:B.P(x,y)$的证明,则目标程序即为$fst(E_z(x))$.

3.4　出现在定义子句中的递归类型的操作语义

以上给出了两个转换G_{RA}和$G_{\Pi RA}$.为使解释器的非确定转换系统完全化,我们对定义子句中的递归类型赋予操作解释.

对于证明状态$(D,N\in RA;M\in C)$.如果N与M,RA与C能够一致化,那么解释器将直接使用D_{atom}完成目标$M\in C$的证明.否则,按照"类型作为操作"原理,我们希望从$N\in RA$中发掘更多的类型信息,以帮助目标$M\in C$的求解.

如果N形如$d_i(N_1,\cdots,N_{ki})$,显然,由$N\in RA$可得出:

$N_1\in B_1,\cdots,N_{ki}\in[N_1/x_1,\cdots N_{ki-1}/x_{ki-1}]B_{ki}$($B_j$出自$RA$的定义),

因此有以下转换规则:

$$D_{RA}:(D,d_i(N_1,\cdots,N_{ki})\in RA;\ M\in C)$$
$$\Rightarrow(D,N_1\in B_1,\cdots,N_{ki}\in[N_1/x_1,\cdots N_{ki-1}/x_{ki-1}]B_{ki};\ M\in C)\ (i=1,\cdots,n).$$

4　归纳定理证明

本节通过归纳定理的证明实例说明:由于对递归类型赋予操作解释,解释器能够自动完成归纳定理证明的一些核心动作:归纳变元的选取、归纳框架的确定、归纳假设的生成与使用.

例 4.1:给定如下的等式系统EQ:

$@\ :NatList\rightarrow NatList\rightarrow NatList$,

$nil\ @\ s=s$,

$(a::l)\ @\ s=a::(l\ @\ s)$;

$rev\ :NatList\rightarrow NatList$,

$rev(nil)=nil$,

$rev(a::l)=rev(l)\ @\ (a::nil)$;

$irev:NatList\rightarrow NatList\rightarrow NatList$,

$irev(nil,s)=s$,

$irev(a::l,s)=irev(l,a::s)$.

其中$NatList$为递归类型,其定义是:

$Ind\_Type\ NatList:TYPE\ With\ nil:NatList,\_::\_:Nat\rightarrow NatList\rightarrow NatList$;

假定用户希望在EQ中证明归纳定理:对任意的$l,s,irev(l,s)=rev(l)\ @\ s$.那么,解

释器将自动对 l 实施归纳，使用 G_{BRA} 生成两个子目标：

$G_0 \equiv irev(nil,s) = rev(nil) @ s$,

$G \equiv \Pi a:nat.\ \Pi l:NatList.$

$(irev(l,s) = rev(l) @ s) \rightarrow (irev(a::l,s) = rev(a::l) @ s))$.

归纳基始 G_0 的证明是容易的；利用前述等式及归纳假设 $irev(l,s) = rev(l) @ s$, $irev(a::l,s) = rev(a::l) @ s)$ 很快获证.

上述证明过程的完整描述请见文献[5].

5 结束语

本文提出了在 *LF* 类型理论中定义一组相互递归类型的方法，并对递归类型赋予操作语义. 这样，递归类型不仅可以表示通常的递归数据结构，还可描述一般的递归问题求解、递归证明构造和递归程序构造过程.

实际上，由于 D, G 中存在 $\forall$ 、$\exists$ 量词，解释器的动作较上述转换复杂. 详见文献[5].

参考文献

1 Coquand T, Paulin C. Inductively defined types. Proc. of International Conf. on Computer Logic, LNCS 417, 1988.

2 Harper R, Honsell F, Plotkin G. A framework for defining logics. Proc. of LICS'87, 1987.

3 Pfenning F. Elf: a language for logic definition and verified metaprogramming. Proc. of LICS'89, 1989.

4 谭庆平，徐锡山，陈火旺. 证明开发环境中的元语言：设计与实现. 全国人工智能与智能计算机学术会议，北京，1991.

5 谭庆平. 基于类型理论的程序设计环境. 工学博士学位论文，国防科技大学研究生院，1992.

RECURSIVE METAPROGRAMMING BASED ON TYPE THEORY

Tan Qingping and Chen Huowang

(*Department of Computer Science, Changsha Institute of Technology, Changsha* 410073)

Abstract This paper presents a new approach to defining a set of mutually inductive types and gives these types an operational interpretation. Therefore, inductive types can express ordinary inductive data structures as well as recursive problem solving and proof construction.

Key words Type theory, proof development environment, recursive.