第6卷第1期 软 件 学 报 Vol. 6, No. 1
1995年1月 JOURNAL OF SOFTWARE Jan. 1995

结构化证明搜索*

谭庆平 陈火旺

(长沙工学院计算机科学系，长沙 410073)

摘要 本文在简介证明开发环境的元语言TML之后，提出两类结构化设施：模块化机制为元级程序设计提供模块化手段；抽象理论机制用来描述定理证明赖以进行的背景理论.联合使用模块机制和结构化理论描述，系统可自动实现结构化证明搜索.

关键词 结构化理论，证明开发，类型理论，元程序设计.

由于软件开发过程中需要进行各种形式化的证明和构造，我们往往希望在统一的框架下实施定理证明、程序自动生成等任务.因此，在同一证明开发/程序设计环境中同时为多种演绎系统提供机器支持已成为许多研究计划的主要目标[1].显然，为了描述目标逻辑系统并对其中的各种推理过程进行编程，元程序设计机制是必需的.

模块化程序设计手段在大型软件开发过程中几乎是必不可少的.由此不难理解结构化设施对于元语言的重要意义.本文在简介证明开发环境的元语言TML之后，重点讨论TML的两类结构化设施：

(1)模块化机制为元级程序设计提供模块化手段，并减小元程序执行过程中的搜索空间.

(2)抽象理论用来描述定理证明赖以进行的背景理论.通过理论组装操作，这种描述构成一种结构化理论空间，系统将依据各理论之间的结构关系实现结构化推理.

1 元语言TML简介

LF[2]作为定义各种逻辑系统的一般元理论，为编码各种逻辑系统的语法、推理规则和证明提供了统一的途径.但LF只能说明逻辑系统，不能对系统中的推理过程进行元级编程.

TML的基本思想是：通过对类型进行操作解释，将LF的逻辑定义风格和λProlog[3]的逻辑程序设计风格融于一体，从而为描述多种逻辑系统以及对这些系统中的推理过程进行元级编程提供统一的手段.

TML所基于的类型理论在LF的基础上增加了类型构造子Σ(dependent product或

* 本文1992—06—09收到，1992—11—12定稿

本研究得到"863"高技术计划及国家自然科学基金的部分资助.作者谭庆平，1968年生，讲师，主要研究领域为计算机科学理论，软件工程.陈火旺，1935年生，教授，主要研究领域为计算机科学理论，软件工程，逻辑学.

本文通讯联系人：谭庆平，长沙 410073，长沙工学院计算机科学系

strong sum),其语法形式如下:

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

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

(3)种别(kind) $K::=TYPE\mid\Pi x:A.K$

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

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

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.形如 u、$M_1\cdots M_n$ 的类型称为原子类型,以 C 表记.u 是类型常元或类型变元.

关于 Σ-类型,有以下规则:

$$\frac{\Gamma\vdash_\Sigma M:A\quad\Gamma\vdash_\Sigma N:[M/x]B}{\Gamma\vdash_\Sigma(M,N):\Sigma x:A.B}\qquad\frac{\Gamma\vdash_\Sigma M:\Sigma x:A.B}{\Gamma\vdash_\Sigma fst(M):A}\qquad\frac{\Gamma\vdash_\Sigma M:\Sigma x:A.B}{\Gamma\vdash_\Sigma snd(M):[fst(M)/x]B}$$

至于其它的类型规则和归约规则,请见文献[2].

类似于 $\lambda Prolog$[3],我们定义 Σ,Γ 上的证明目标(G)和定义子句(D)如下:

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

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

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

TML 解释器可描述为关于证明状态$(D;G)$的非确定转换系统.

$R:(D;G)\Rightarrow(D';G_1',\cdots,G_n')$表示规则 R 将 D 和 G 分别转换为 D'和子目标序列 G_1',$\cdots$, G_n'.

1.1 逻辑连结词的操作解释

下述规则将逻辑连结词解释为搜索操作.

G_{any} :$(D;\forall\ x:A.G)\Rightarrow(D\wedge y\in A;[y/x]G)$,($y$ 是新变元);

G_{exist} :$(D;\exists\ x:A.G)\Rightarrow(D;(M\in A)\ \&\ [M/x]G)$;

G_{and} :$(D;G_1\wedge G_2)\Rightarrow(D;G_1)\ \&\ (D;G_2)$;

G_{or}^i :$(D;G_1\vee G_2)\Rightarrow(D;G_i)$, $(i=1,2)$;

G_{imp} :$(D;D'\rightarrow G)\Rightarrow(D,D';G)$;

D_{any} :$(D,\forall\ x:A'.D';M\in C)\Rightarrow(D;M'\in A')\ \&\ (D,[M'/x]D';M\in C)$;

D_{imp} :$(D;G\rightarrow M'\in C';M\in C)\Rightarrow(D;C=C'\ \&\ M=M'\ \&\ G)$.

(注:"$D_1,\cdots,D_n$"是"$D_1\wedge\cdots\wedge D_n$"的简写.)

1.2 类型的操作解释

非形式地,如果当前目标是 $z\in\Sigma x:A.B$,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前述的 Σ-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 $y\in B$.

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

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

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

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

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

(2)解释器对待解目标的证明搜索空间将局限于合适的元程序模块,提高了系统效率.

(3)模块机制的引进将提高人机交互粒度.用户可以直接观察TML元模块施用于当前证明目标之后的证明状态,从而以模块为基本单位对证明开发/程序设计过程实施控制,不必拘泥于通常的规则级交互.

3 结构化理论

本节介绍TML的另一类结构化设施:抽象理论及其组装操作.

3.1 抽象理论

与一般的数学理论及代数规范中的抽象数据类型相仿,TML的抽象理论有标记、公理集、定理集三部分内容.标记包括通常意义上的载体集说明、操作符说明.此外,由于TML采用LF对目标逻辑系统的编码风格,目标理论中的"断言形式"由TML的类型常元描述,这种特殊的类型常元也将出现在抽象理论的标记部分.例如,一阶逻辑中的基本断言形式"$\vdash A$"可表示为$True:Bool\rightarrow TYPE$[2].

定义 3.1. TML中的抽象理论$T=(Sig,Axm,Thm)$,其中:

(1) $Sig=(Car,OP,J)$是T的标记部分.

Car为T的载体集说明.载体又分为抽象载体和具体载体两种,它们分别由TML的类型变元和类型常元表示,种别均为$TYPE$.

OP是T的操作符集合.每个操作符的说明形如$op:A$.

J是T的断言描述.每个断言的说明形如$c:K$.

(2) Axm是T的公理集.公理的描述形式是$d:A$.

(3) Thm是T的定理集.每条定理形如$M:A$.

使用TML描述抽象理论须遵守以下语法限制:

假定Σ,Γ分别表示由Sig和Axm得到的标记和上下文,那么:

• $\Sigma\vdash\Gamma\ context$在TML的类型理论中成立;并且

• 对于Thm中的每条定理$M:A$,$\Gamma\vdash_\Sigma M:A$成立. □

例 3.2. 抽象半群理论SG在TML中表示为:

(1)标记:抽象载体 $\alpha:TYPE$,

操作符 $*:\alpha\rightarrow\alpha\rightarrow\alpha$,

断言形式 $eq:\alpha\rightarrow\alpha\rightarrow TYPE$;

(2)公理:$ass:\Pi a,b,c:\alpha.\ eq(a*(b*c),(a*b)*c)$;

(3)定理:略. □

3.2 理论组装

以下研究由原有理论构造新理论的三种方法:实例化、继承、联合.

首先说明一些术语和记号:

(1)如果$T=(Sig,Axm,Thm)$,由Sig和Axm得到的标记和上下文的混合记为$\varepsilon(T)$,$\varepsilon(T)=Sig\oplus Axm$.

(2)对于混合上下文ε_1、ε_2,$f:\varepsilon_1\rightarrow\varepsilon_2$实指$f:dom(\varepsilon_1)\rightarrow dom(\varepsilon_2)$,也即,$f$是从$dom(\varepsilon_1)$到

$dom(\varepsilon_2)$的映射.

(3)给定映射 $f:\varepsilon_1\to\varepsilon_2$. 可将 f 的作用域自然扩展到 ε_1 上的任意语法对象(项/类型或种别).

定义 3.3. 给定载体集 Car 和 Car_0,$ins:Car\to Car_0$ 称为实例化映射,如果:

(1)ins 是满射;

(2)对 Car 中的任意类型常元 c,$ins(c)$是 Car_0 中的类型常元;

(3)Car 中至少有一个类型变元 α 被实例化,也即,$ins(\alpha)$是类型常元. □

定义 3.4. 给定理论 $T=(Sig,Axm,Thm)$ 及 $T_0=(Sig_0,Axm_0,Thm_0)$,$Sig=(Car,OP,J)$,$Sig_0=(Car_0,OP_0,J_0)$. $ins:Car\to Car_0$ 为实例化映射.

理论 T_0 是 T 的实例化,当且仅当:

(1)存在双射 $f_{OP}:OP\to OP_0$ 及双射 $f_J:J\to J_0$,使得:

$op:A\in OP$ *iff* $h(op):h(A)\in OP_0$;

$c:K\in J$ *iff* $h(c):h(K)\in J_0$.

(2)存在内射 $f_{axm}:Axm\to Axm_0$使得:

若 $d:A\in Axiom$,则 $h(d):h(A)\in Axiom_0$. 这里,$h=ins\cup f_{OP}\cup f_J\cup f_{axm}$. 称$(ins,f_{OP},f_J,f_{axm})$为 T 到 T_0 的实例化映射.

一般情况下,上述定义中的 f_{Op},f_J,f_{axm}可由 ins 唯一确定. 所以,也称 ins 为 T 到 T_0 的实例化映射. □

定理 3.5. 假设 $f=(ins,f_{OP},f_J,f_{axm})$是 T 到 T_0 的实例化映射,那么:

(1)若 $\varepsilon(T)\vdash M:A$,则 $\varepsilon(T_0)\vdash f(M):f(A)$;

(2)若 $\varepsilon(T)\vdash A:K$,则 $\varepsilon(T_0)\vdash f(A):f(K)$.

证明:对 $\varepsilon(T)\vdash M:A$ 的派生过程归纳,利用定义 3.4,易证. □

定义 3.6. 给定标记 $Sig=(Car,OP,J)$, $Sig'=(Car',OP',J')$. $f_{sig}=(f_{car},f_{OP},f_J)$称为从 Sig 到 Sig'的标记态射,其中:

(1)$f_{car}:Car\to Car'$为内射,且对任意的 $s\in Car$,s 为 Sig 的抽象载体当且仅当 $f_{car}(s)$为 Sig'的抽象载体.

(2)$f_{OP}:OP\to OP'$为内射. 若 $d:A$ 在 OP 中出现,则 $f_{OP}(d):f_{sig}(A)$在 OP'中必出现.

(3)类似于(2),$f_J:J\to J'$为内射;若 $c:K$ 在 J 中出现,那么 $f_J(c):f_{sig}(K)$在 J'中出现.

定义 3.7. 给定理论 $T=(Sig,Axm,Thm)$和 $T'=(Sig',Axm',Thm')$,$f=(f_{sig},f_{axm})$是从 T 到 T'的理论态射,其中:

(1)f_{sig}是从 Sig 到 Sig'的标记态射;

(2)$f_{axm}:Axm\to Axm'$是内射. 若 $d:A$ 是 T 的公理,那么 $f_{axm}(d):f(A)$ 是 T'的公理. 此时称 T'为 T 的继承. □

直观上,若 T'是 T 的继承,那么 T'将(通过换名)沿用 T 的所有载体集、操作符,断言形式及公理. 利用继承关系,可以对原有理论进行丰富,增加新的操作符、公理、定理等,从而形成新的理论.

关于继承关系,有以下结论:

定理 3.8. 假定 f 是从 T 到 T'的理论态射.

(1)若 $\epsilon(T)\vdash M:A$,那么 $\epsilon(T')\vdash f(M):f(A)$.

(2)若 $\epsilon(T)\vdash A:K$,那么 $\epsilon(T')\vdash f(A):f(K)$. □

定义 3.9. 给定理论 T,T_1,T_2. 如果存在理论态射 $f_1:T_1\to T$ 和 $f_2:T_2\to T$,则称 T 为 T_1,T_2 经(f_1,f_2)形成的理论联合. □

在理论的联合过程中,允许增加新的标记内容及公理、定理. 并且,利用理论态射,可实现结构共享. 例如,为了形成环理论 *Ring*,可联合半群理论 *SG* 和 *Abelian* 群理论 *AG*,并利用理论态射 $f_1:SG\to Ring$,$f_2:AG\to Ring$,将 *SG* 和 *AG* 的抽象载体(以及断言形式)合二为一.

4 结构化推理

联合使用模块机制和结构化理论描述,TML 解释器可自动实现结构化证明搜索. 这是 TML 结构化设施的主要特色之一.

为了描述 TML 解释器的结构化推理行为,我们在证明状态中显式表示当前的背景理论.

下述规则说明,解释器在抽象理论 T 中求解证明目标时可使用 T 的所有定理、公理.

$G_{abs}:(T;D;G)\Rightarrow(T;D,M\in A;G)$,这里 $M:A$ 是 T 的定理或公理.

4.1 抽象推理

抽象推理的思想是:不必针对多个具体理论反复证明相似的定理,只需在抽象理论中证明一抽象定理,然后在各具体理论中分别将该抽象证明作相应的实例化,即可获得具体定理的证明.

定义 4.1. $f=(f_{ins},f_{OP},f_J,f_{axm})$为从 T 至 T_0 的实例化映射. 给定 T_0 中的项 M_0,以下算法试图求出 T 中相应于 M_0 的项 M,使得 $f(M)\equiv M_0$.

(1)关于项.

$Abs(d)=\{d'\mid f(d')=d\}$;

$Abs(x)=\{x\}$;

$Abs(\lambda x:A.M)=\{\lambda x:A'.M'\mid A'\in Abs(A),\ M'\in Abs(M)\}$;

$Abs((M,N))=\{(M',N')\mid M'\in Abs(M),N'\in Abs(N)\}$;

$Abs(fst(M))=\{fst(M')\mid M'\in Abs(M)\}$;

$Abs(snd(M))=\{snd(M')\mid M'\in Abs(M)\}$;

(2)关于类型可完全类似地定义 *Abs*.

(3)显然,对任意 $M\in Abs(M_0)$,有 $f(M)\equiv M_0$. 如果有 $M\in Abs(M_0)$,且 M 在 T 中是良类的,则称 M 是 M_0 的抽象版本. □

假定 f 是从 T 到 T_0 的实例化映射,且 M_0,A_0 分别有抽象版本 M,A,那么有以下规则:

$G_{ins}:(T_0;D_0;M_0\in A_0)\Rightarrow(T;D;M\in A)\&(T_0;D_0;M_0=f(M)\wedge A_0=f(A))$.

基于 G_{ins},解释器动作如下:

(1)生成抽象版本 $M\in A$;

(2)对证明目标 $M\in A$,尝试 D_0 中以 T 为背景理论的子模块;

(3)如果没有可用的子模块，解释器再寻找其它基于 T 的可用模块；当 T 的所有元模块都不适于求解 $M\in A$ 时，解释器将 D 置为空模块，然后使用 G_{abs} 直接访问 T 中的定理、公理.

利用定理 3.5，不难证明 G_{ins} 的正确性.

4.2 证明继承

如果存在从 T 至 T' 的理论态射 f，那么 T' 比 T 强，因此对 T' 中新定理的证明可望继承 T 中相应定理的证明.

类似于 G_{ins}，对于 T' 中的证明目标 $M'\in A'$，如果有 T 中的语法对象 M,A，使得 $f(M)$ M' 且 $f(A)\equiv A'$，那么有以下转换规则：

G_{inh}：$(T';D';M'\in A')\Rightarrow(T;D;M\in A)\&(T';D';M'=f(M)\wedge A'=f(A))$

G_{inh} 在形式上与 G_{ins} 非常相似，但它们所引发的解释器行为却大相径庭：

(1)对于证明状态 $(T';D';M'\in A')$，首先计算 M',A' 在 T 中的相应形式 M,A，使得 $f(M)\equiv M'$，$f(A)\equiv A'$. 构造 M,A 的方法类似于定义 4.1. 与实例化关系不同的是，在继承关系下，满足 $f(M)\equiv M'$，$f(A)\equiv A'$ 的 M,A 是唯一的.

(2)对证明目标 $M\in A$，尝试 D' 中以 T 为背景理论的子模块.

(3)如果没有可用的子模块，那么解释器通过 G_{abs} 直接使用 T 的定理求解 $M\in A$. 如果 T 中没有可用定理，那么对 $M'\in A'$ 的证明搜索仍在 T' 中展开，除非用户显式说明了 T 中可用的子模块.

G_{inh} 的正确性由定理 3.8 保证.

5 结束语

本文给出了旨在表示目标逻辑系统并对定理证明过程实施元级编程的元语言 TML. 以此为基础，我们讨论了 TML 的两类结构化设施：模块化机制和抽象理论机制.

Luo Zhaohui[5]在 ECC 中利用类型构造子 Σ 和类型通域提出了一种结构化推理机制. Sannella, Burstall 模仿规范语言 Clear 的结构化抽象数据类型在 LCF 中引进结构化理论[6]. 此外，Harper, Sannella, Tarlscki[7]基于“结果关系”(Consequence relation)给出了一种结构化理论表示方法. 与这些研究工作比较，TML 结构化设施的主要特色在于，联合使用模块机制和结构化理论描述，TML 解释器可自动实现结构化证明搜索.

参考文献

1 Lee P, Pfenning F, Reynolds J *et al*. Semantically based program－design environments. The Ergo Project in 1988, Technical Report, Carnegie Mellon University, CMU－CS－88－118, 1988.

2 Harper R, Honsell F, Plotkin G. A framework for defining logics. Proc. 2nd Ann. Symp. on Logic in Computer Science, 1987.

3 Nadathur G, Miller D. An overview of λProlog. Proc. of 5th International Conf. and Symp. on Logic Programming, 1988.

4 谭庆平，陈火旺. 证明开发环境中的元语言设计. 计算机学报，1995，**18**(4).

5 Luo Zhaohui. An extended calculus of constructions [Ph. D. Thesis]. Edinburgh University, 1990.

6 Sannella D, Burstall R. Structured theories in LCF'. Proc. of 8th Colloquium on Trees in Algebra and Program-

ming, 1983.

7 Harper R. Sannella D, Tarlecki A. Structure and representation in LF. Proc. of 3rd IEEE Symp. on Logic in Computer Science, 1989.

8 Elliott C. Some extensions and applications of higher—order unification [Ph. D. Thesis]. Carnegie Mellon University, 1990.

STRUCTURED PROOF SEARCH

Tan Qingping Chen Huowang

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

Abstract This paper describes TML, a metalanguage intended for proof development and program design environments. The abstract theory and meta—module mechanisms are presented in TML to allow a good modularization of proof development and make it possible to direct the search for a proof in well—structured theories mechanically.

Key words Structured theory, proof development, type theory, metaprogramming.