第 18 卷 第 5 期
1992 年 9 月

自 动 化 学 报
ACTA AUTOMATICA SINICA

Vol. 18, No. 5
Sep., 1992

短文

一阶逻辑知识库的检测

王 申 康

(浙江大学计算机系，杭州 310027)

摘 要

本文提出的方法是以 Loveland 的 MESON 一阶逻辑定理证明过程为基础，用于一阶逻辑规则知识库的冗余性和不一致性的检测. 知识库的规则可包含非真、或及 if-and-only-if 规则. 系统以交互形式从正、反向推理研究知识库规则增加时的变化.

关键词: 人工智能，知识库，一阶逻辑，定理证明.

一、序 言

FOLDS(First Order Logic Debugging System) 的设计背景是 CODE系统[7]，CODE系统是综合人工智能，面向对象的程序设计、逻辑程序设计和自然语言的知识获取工具. CODE 的基本表达单元是“概念描述器”(cd). cd 是一组分类的属性集. 其中“逻辑特性”类属性是用一阶逻辑规则和事实表达相关概念之间关系. FOLDS 是 CODE 的一个模块，但它又是独立的，可以依附于任何的专家系统和其它规则基系统，也可单独用于大型知识库的调试.

虽然许多严格的定理证明提供了知识检测方法[4]，但这些系统要求用户具有相当的逻辑和定理证明的知识，而且也不具备有流行的 Prolog 那样方便的交互的形式. 如 Ginsberg 有过一些实践的考虑[2]，Poole 和 Goebel 也引用 MESON 过程的思想到 Prolog 中[6]，Flann 把前向推理加到 Prolog 中[1]. 而 FOLDS 则是一个完整的知识检测系统，它既考虑到 Prolog 的简易性，又不涉及太深奥的逻辑知识. 由于大多数用户的知识处理不太复杂，他们乐于用类似于 Prolog 的一阶逻辑来直观地表达其知识的逻辑约束，所以 FOLDS 扩充的全一阶逻辑表达形式(～，V，和 if-and-only-if) 使系统有更广的用途.

二、MESON 定理证明

FOLDS 是交互式的一阶逻辑定理证明系统，是 Prolog 一阶逻辑的扩充. 其主要功能有交互式的知识演绎、知识冗余性和不一致性检测. 系统采用了类似 MESON 证明过程. 本文只给出其工作过程的直观理解，其理论基础可参阅文献 [3].

MESON 定理证明过程中的语法定义如下:

本文于 1990 年 10 月 23 日收到.

〈项〉::＝〈谓词〉|〈谓词〉(〈变量表〉)

〈字〉::＝〈项〉|～〈项〉

〈子句〉::＝〈组合字〉if〈组合字〉

〈组合字〉::＝〈字〉|〈字〉 or 〈组合字〉|

〈字〉 and 〈组合字〉

〈逆反式〉::＝〈字〉 if 〈字合取式〉

〈字合取式〉::＝〈字〉|〈字合取式〉

MESON 过程中的每个子句必须是逆反式 (contraposition form), 即子句的 then-side 只是一个字 (literal), if-side 是字的合取. 例如子句～ a or c if b 有以下三种等价逆反式:

～a if～c and b

c if a and b

～b if a and～c

证明目标 G 成功的 MESON 规则为

S1: 如果目标 G 和某一子句的头 (then-side) 合一, 而子句体 (if-side) 中每一字均被证明,则目标 G 成功.

S2: 如果目标 G 和证明 G 过程中出现的某一前子目标的反合一,则目标 G 成功.

S1 是把 G 的证明化简为一些简单子目标的证明, S2 是反驳证明 (Proof by contradiction). 下面举例说明.

rule1:A if B　　(B 隐含 A)

知识库　rule2:B if C and～A　(C 隐含 A or B)

rule3:C if true　　(C 为事实)

证明 A 为真的过程如下:

A is true if (rule 1)

B is ture if (rule 2)

C is true and (rule 3)

if～A is true then A can be proven(S2).

上述最后一行证明中～A 是证明 A 的先前一个子目标,且为目标 A 之反,根据 S2,则可证明 A.

当然实际 MESON 证明过程远为复杂,因为它还要包含优化、等价、循环等处理技术[3].

三、 FOLDS

3.1. 源知识编译

FOLDS 的源知识库 (KB) 是规则和事实的集合,其语法如下:

〈源规则〉::＝〈组合字〉if〈组合字〉

|〈组合字〉iff〈组合字〉

〈源事实〉::＝〈字〉

用户用上述语法编写的规则、事实在 FOLDS 中被编译成 MESON 过程中所要求的所有可能的等价逆反式.

1) 无变量子句编译

第一步: 生成等价逆反式

(a) 变 "iff" 成两条 "if"规则,如 A iff B → A if B,B if A;

(b) 转换合取,析取操作, 如 A and B if C → A if C, B if C

A if B or C → A if B,A if C;

(c) 转换子句头中的析取操作, 如 A or B if C → A if ~B and C

~(A and B) if C → ~A if B and C.

第二步: 生成全部可能的等价逆反式,如 A if B 派生 ~B if ~A.

KB 中每条规则按上述两步骤转换,最后得到一个和 KB 等价逆反式知识库 (CKB).

2) 含有变量子句的编译

Prolog 中约定变量为全称变量, 然而这种约定在 if-and-only-if 规则中会导致错误,为此 FOLDS 采用 skolem 方法加以修正. 例如下式(1)为 KB 中一条源规则.

$$\text{Sibling}(X,Y)\text{iff parent}(Z,Y)\text{and Parent}(Z,X)\text{and}\sim\text{eq}(X,Y). \quad (1)$$

根据上述第一步 (*a*) 转化为 (1*a*),(1*b*) 两式:

$$\text{Sibling}(X1,Y1)\text{ if parent}(Z1,X1)\text{and parent}(Z1,Y1)\text{and}\sim\text{eq}(X1,Y1), \quad (1a)$$

$$\text{Parent}(Z2,X2)\text{and parent}(Z2,Y2)\text{and}\sim\text{eq}(X2,Y2)\text{if Sibling}(X2,Y2). \quad (1b)$$

式 (1*a*) 已为逆反式,式 (1*b*) 根据第一步 (*b*) 转化为 (1*b*1) 和 (1*b*2) 两式:

$$\text{Parent}(Z3,X3)\text{and Parent}(Z3,Y3)\text{if Sibling}(X3,Y3), \quad (1b1)$$

$$\sim\text{eq}(X4,Y4)\text{if Sibling}(X4,Y4). \quad (1b2)$$

因为 (1*b*2) 已为逆反式, (1*b*1) 由第一步 (*b*) 方法转化为逆反式:

$$\text{Parent}(Z5,X5)\text{if Sibling}(X5,Y5),\ \text{Parent}(Z6,Y6)\text{if Sibling}(X6,Y6).$$

上两式的意义为

if X is sibling of Y then $\forall\ Z$, Z is parent of X and Z is parent of Y. 而 (1*b*) 式的原意为

if X is sibling of Y then $\exists\ Z$, Z is parent of X and Z is parent of Y.

事实上,编译 (1) 式时, Z 为全称变量,而转化到 (1*b*) 时, Z 已转为存在变量,为防止这些错误产生,笔者把 (1*b*1) 中的 Z 变量用 skolem 函数 "common-parent-of(X,Y)" 表达, (FOLDS 自动生成 skolem 函数名),这样 (1*b*1) 就被编译成两个逆反式:

$$\text{Parent}(\text{common-parent-of}(x4,y4),x4)\text{if Sibling}(x4,y4),$$

$$\text{Parent}(\text{common-parent-of}(x5,y5),y5)\text{if Sibling}(x5,y5).$$

3.2. 知识库检查

1) 冗余性检查

如果一条规则或一个事实从知识库中移去而不影响其演绎结果, 则认为其规则为冗余的. 例如,规则 $\{A(x,y)\ \text{if}\ B(x,y)\}$ 从知识库 CKB 移去,生成新知识库 CKB′, 仍能从 CKB′ 中由 B 演绎出 A, 则 $\{A(x,y)\text{if}\ B(x,y)\}$ 为冗余的,检查过程如下:

while $R=\{A(x,y)$if $B(x,y)F\}$ 无标记 do Begin

从 CKB′ 中移去R及其全部等价逆反式；

假设 CKB′ 中对所有 $x,y,B(x,y)$ 为真，证明 $A(x,y)$

if $A(x,y)$为真，then$R$是冗余的 else 恢复R及其全部逆反式，并标记检查.

end.

2）一致性检查

如果知识库中并存两个矛盾的结论(如A和$\sim A$)，则认为知识库是不一致的，这种矛盾结论的累积会增加寻求不一致源的难度，即无法确定矛盾结论相应的规则和事实. 所以 FOLDS 在知识库递增的过程中，每加入一条规则，事实结论时调用不一致性检查，并展示证明过程轨迹. 由用户决定舍去最后加入的结论还是原先已有的结论. 不一致性检查时和冗余性检查中采用全称变量相反，是采用了存在变量，例如，要检查 fact(x,t) 和知识库是矛盾的，只需证明知识库中 $\exists x\sim$fact(x,t) 即可.

检查过程如下：

令 S 为知识库中产生(输入)的新假设

$S=f(x)$ 或 $S=\{A$ if $B\}$

if $S=f(x)$

then 证明$\sim f(x)$

if$\sim f(x)$ 为真，then S 为不一致假设 else S 为一致假设.

else 证明$\sim$ A和B

if$\sim A$和B为真 then S 为不一致假设 else S 为一致假设.

3）递归规则和无限循环检查

定理证明中如因递归规则而引起无限循环，用户可设置一个证明所允许的最大深度防止它，但须对那些可能导致无限循环的规则打上不同的标记，然而这种规则的识别也是很麻烦的. NGUYEN 提出了规则识别表的方法[5]，FOLDS 采用类似思想，但不必要求用户标记这些规则. 如果在证明树上一条规则在同一分支上调用次数过多 FOLDS，则认为证明失败. 这次数即搜索深度的约定如果太小，可能导致伪失败，而太大会导致过长的时间，所以 FOLDS 有以下两点约定方法：

(1) 当人为输入一条新规则或事实时，系统自动设置一个较少搜索深度来检查不一致性，如失败则增加深度，继续证明直至证明成功或用户认为证明失败，中止证明(这种技术称为 depth-first iterative deepening).

(2) 当检查整个知识库的冗余和不一致性时，用户给出全部证明的最大搜索深度.

四、结　论

FOLDS 是具有良好用户界面的一阶逻辑规则和事实的检测工具，对被检测的源知识的表达是十分似然的. 如提供 if-and-only-if、析取、非真、函数标记等形式. 系统还提供辅助规则书写、检查、查询和证明轨迹等功能.

系统的高度灵活交互性对递归性较强的知识库可能速度较慢. 但对大多数一般的知

识来说这种速度是完全可以接受的，另外系统还可以脱机来检查一个大型知识的冗余性和不一致性，系统进一步的工作是如何从一个简单的证明轨迹中发现不一致知识根源.

系统在 SUN3/60 机上对有 33 条规则,事实的源数据库的编译时间为 7 秒，输入一条新规则的冗余检查为 1.5 秒,一致性的检查为 147 秒.

参 考 文 献

[1] Nicholas S. Flann, Thomas G. Dietterich and Dan R. Corpron, Forward Chaining Logic Programming with the ATMS, in Proceedings of The Sixth National Conference on Artificial Intelligence, 1987, Seatle, Washington, U. S. A., 24—29.

[2] Allen Ginsberg, A New Approach to Checking Knowledge Bases For Inconsistency and Redundacy, in Proceedings of The Third Annual Expert System in Government conference, (1987), 102—111.

[3] Loveland, D. W., Automated Theorem Proving: A Logical Basis, North-Holland, (1978), Amstersdam, The Netherlands.

[4] Nguyen, T. A., Rerkins, W. A., Laffey, T. J. and Prcora, D., Checking an Expert System For Consistency and Compleness, in Proceedings of the Ninth International Joint Conference on Artificial Intelligence, (1985), Los Angeles, California, U. S. A., 1(1985), 375—378.

[5] David Paulson, Experience with Isabelle. A Generic Theorem Prover. University of Cambridge Computer Laboratory Technical Report 143, August, (1988).

[6] David, L. Poole and Randy Goebel, Gracefully Adding Negation and Disjunction to Prolog, in Proceedings of the Third Logic Programming Conference, (1986), London, 635—641.

[7] Douglas Skuce, Shenkang Wang, Yves Beauville, A portable, Ceneric Knowledge Acquisition Environment that Understands Basic Logic and Language, Proc. Fourth Knowledge Acquistion for Knowledgebased Systems Workshop, Banff, 1989.

DEBUGGING FOR A FIRST ORDER LOGIC KNOWLEDGE BASE

WANG SHENKANG

(*Artificial Intelligence Institute Zhejiang University, Hangzhou* 310027)

ABSTRACT

The method described in this paper is based on Loveland's MESON proof procedure of debugging a knowledge base (kb) of full first order logical rules, including true negation, disjunction, and if-and-only-if rules. It is used to query the kb and to detect redundancies and inconsistencies. Explanations are provided for each contradiction and redundancy found. Both backward and forward chainning capabilities of the system make it possible to investigate the consequences of new additions to the kb.

Key words : Artificial intelligence; knowledge base; first order logic; theorem proving.