

文章编号 1000-5013(2001) 01-100-05

# Prolog 的语义保持变换

余 金 山

(华侨大学信息科学与工程学院, 泉州 362011)

**摘要** 程序变换方法在逻辑程序中的应用主要是基于这样的理论结果, 即 Fold/Unfold 变换保持了逻辑程序的最小 Herbrand 模型语义和答复置换集语义. 但是当逻辑程序的实现采用标准 Prolog 系统的最左计算规则和深度优先查找规则时, 变换后的程序未必能保持原程序的语义. 此外, 程序的语义等价性证明也往往是难以理解和阅读的. 为此, 首先用与计算 SLD-树叶结点有关的答复置换序列算法的方式, 引入一个相对易于理解的 Prolog 语义定义, 然后给出有关的语义保持变换规则并加以证明.

**关键词** 程序变换, Prolog, 语义

**中图分类号** TP 301.2; TP 311.1

**文献标识码** A

程序变换方法的基本思想是使用一些规则(称为变换规则)把某个程序变换成另一个等价(语义保持)程序, 并使变换后的程序效率得到提高, 从而实现软件开发自动化和保证正确性的宏伟目标. 因此, 几十年来一直是计算机界研究的重要方向之一<sup>[1]</sup>. 变换方法在逻辑程序中的应用主要是基于如下的理论结果, 即 Fold/Unfold 变换规则保持了逻辑程序的最小 Herbrand 模型语义和答复置换集语义<sup>[2]</sup>. 但是, 当逻辑程序的实现采用标准 Prolog 系统的最左计算规则和深度优先查找规则时, 由于程序的终止等计算特性问题, 变换后的程序未必能保证与原程序等价<sup>[3]</sup>. 另一个存在的问题是语义等价性的证明, 往往是难以理解和阅读的<sup>[4]</sup>. 有趣的是这一类逻辑程序却很象一种用置换序列来定义计算的语言. 本文首先用与计算 SLD-树叶结点有关的答复置换序列算法的方法, 引入一个相对易于理解的 Prolog 语义定义, 然后给出有关的语义保持变换规则并加以证明和讨论.

## 1 Prolog 语义和程序变换

### 1.1 记号和基本定义

设  $\text{Funct}$  为函数符号(包括常数)的有限集,  $\text{Pred}$  为谓词符号的有限集,  $\text{Vars}$  为变量符号的无限可数集. 用  $\text{Terms}$ ,  $\text{Atoms}$ ,  $\text{Clauses}$ ,  $\text{Progs}$  分别表示项、原子、子句和程序的集合. 给定一个表达式  $E$  ( $\text{Terms} \text{ Atoms} \text{ Clauses}$ ), 用  $\text{vars}(E)$  表示存在于  $E$  中的变量集合. 用  $P(S)$  表示集合  $S$  的幂集. 用  $\text{Subst}$  表示置换的集合. 置换  $\theta$  的定义域记为  $\text{dom}(\theta)$ .

**定义 1** 函数限定.  $\text{Subst} \times P(\text{Vars}) \rightarrow \text{Subst}$ .  $\text{confine}(\theta, S) = \{b \mid b = \theta \text{ and } b = X/t \text{ and } X \in S\}$ . 考虑自然数集  $\omega$  的有限或无限初始部分的集合  $I, I = \{\}, \{0\}, \{0, 1\}, \dots, \omega$ . 显然  $I$  按集合包含的意义完全排序. 令  $\text{Subst} = \text{Subst} \{ \}; \text{Subst}^* = \{n \text{ Subst} \mid n \in I\}$ .  $\text{Subst}$  表示从  $n$  到  $\text{Subst}$  的函数集合.

定义域为  $\{0, \dots, n\}$  的  $\text{Subst}$  的元素可记为  $\theta_0, \dots, \theta_n$ .

**定义 2**  $\text{Subst}^*$  的子集. (1)  $\text{Subst}^*$  是所有非模糊 (non-undefined) 置换的有限序列的集合. 即序列  $\theta_0, \dots, \theta_n$  的集合, 其中  $\theta_0, \dots, \theta_n$  是  $\text{Subst}$  所有的成员. (2)  $\text{Subst}^*$  是置换的有限序列的集合, 其中只有最后一个元素是. 即序列  $\theta_0, \dots, \theta_{n-1},$ .

**定义 3** 设  $\text{Subst}^\omega$  为置换的无限序列集, 即从  $\omega$  到  $\text{Subst}$  的函数集合.  $\text{Subst}^\omega$  的元素可记为  $\theta_0, \theta_1, \dots$ , 则集合  $\text{Substseq} = \text{Subst}^* \cup \text{Subst}^\omega$ . 如果  $\eta$  是一个置换, 并且  $\theta_0, \theta_1, \dots \in \text{Substseq}$ , 那么  $\eta \circ \theta_0, \theta_1, \dots$  为置换序列, 即  $\eta \theta_0, \eta \theta_1, \dots$ . 规定  $\eta \circ = = \eta$ , 并且  $\eta \circ = = \eta$ .

**定义 4** 令  $\diamond$  表示  $\text{Substseq}$  中的序列的连接 (concatenation). 假设  $S$  和  $\eta_0, \eta_1, \dots$  是  $\text{Substseq}$  中的 (有限的或无限的) 序列. 那么  $S \diamond \eta_0, \eta_1, \dots = (\text{if } S = \theta_0, \dots, \theta_n \text{ and } S \in \text{Subst}^* \text{ then } \theta_0, \dots, \theta_n, \eta_0, \eta_1, \dots \text{ else } S)$ . 用运算符  $\diamond (S_i \mid i = 1, \dots, n)$  表示有限序列  $S_1, S_2, \dots, S_n$  的连接, 即  $S_1 \diamond S_2 \diamond \dots \diamond S_n$ .

**定义 5**  $\text{Substseq}$  上的半序关系  $\triangle$  可定义为  $S_1 \triangle S_2$  当且仅当  $S_1 \in (\text{Subst}^* \cup \text{Subst}^\omega)$  and  $S_1 = S_2$ , 或  $\exists S_3 \in \text{Subst}^*$  and  $S_4 \in \text{Substseq}$ , 使得  $S_1 = S_2 \diamond S_3$ , and  $S_2 = S_3 \diamond S_4$ . 不难看出  $\text{Substseq}, \triangle$  是一个完全半序 (partial order), 其中  $\theta_0$  是最小的元素. 于是, 关系  $\text{Ren} \subseteq (\text{Subst} \times \text{Subst})$  是一个等价关系. 序列  $S$  的  $\text{Ren}$ -等价类表示为  $\text{Ren}(S)$ .

## 1.2 Prolog 程序的语义

对于每个由  $n$  个子句组成的程序  $P$ , 定义函数为

$$\tau: [(\text{Atoms}^* \times P(\text{Vars})) \rightarrow \text{Substseq}] \rightarrow [(\text{Atoms}^* \times P(\text{Vars})) \rightarrow \text{Substseq}]$$

$$\tau(f)[\text{nil}, V] = \epsilon$$

$$\tau(f)[A.T, V] = \diamond \text{confine}[S_i, \text{vars}(A.T)] \mid i = 1, \dots, n$$

其中  $S_i = \theta \{f[\text{body}(D_i), T], \theta, V \text{ vars}(D_i)\}$ ,  $\theta = \text{mgu}[A, \text{head}(D_i)]$ ,  $D_i = \text{rename}(C_i, V)$ .

对于任何  $\text{Substseq}$  中的  $S$ , 有  $\text{Fail } S =$ .

设函数的  $\text{head}$  和  $\text{body}$  分别返回一个给定子句的头和体. 第 1 个参数  $A.T$  是要求证的目标, 第 2 个参数  $V$  是当前在 SLD-推导路径中使用的变量名集合.

$\text{Substseq}$  的阶  $\triangle$  包括一个函数空间  $[(\text{Atoms}^* \times P(\text{Vars})) \rightarrow \text{Substseq}]$  上的完全半序, 其中最小的元素是对于每个输入都给出  $(\epsilon)$  的函数. 显然  $\tau$  是连续的. 于是, 若给定目标  $G$ , 则逻辑程序  $P$  的语义可定义成商集  $\text{Substseq}/\text{Ren}$  中的元素  $\text{Answers}(P, G)$  为  $\text{Answers}(P, G) = \text{Ren}[\text{fix}(\tau)(G, \Phi)]$ , 其中  $\text{fix}$  表示最小不动点操作符. 上述 Prolog 语义的定义实际上是一个算法的描述, 该算法可计算出与  $P, G$  的 SLD-树的叶子有关的答复置换序列.

## 1.3 变换规则

变换序列是程序  $P_0, P_1, \dots, P_m$  的一个序列, 对于  $i = 1, \dots, m$ , 程序  $P_{i+1}$  可以通过使用下面的变换规则从  $P_0, P_1, \dots, P_i$  中获得. (1) 定义规则. 定义规则的要点是给程序  $P_i$  增加一个新的子句  $D$ , 即  $\text{newp}(A_1, \dots, A_k): \text{Body}$ . 其中  $\text{newp}$  是一个新的谓词符号, 且  $\text{Body}$  中的谓

谓词符号已经在  $P_i$  中出现过. 注意递归定义是不允许的. 所以, 如果  $P_i = C_1, \dots, C_m$ , 则  $P_{i+1} = D, C_1, \dots, C_m$ . (2) 展开(Unfolding)规则. Unfolding 实际上是一步 SLD 归结. (3) 折叠(Folding)规则. 它用形为  $H:-P \quad K\theta \quad R$  的新子句  $F$ , 替换程序  $P_i$  中形为  $H:-P \quad Q$  的子句  $C$ . 假定在  $P_0, P_1, \dots, P_i$  的某个  $P_h$  存在一个形为  $K:-\text{Body}$  的重命名句  $D$ , 使得: (1)  $\text{vars}(D) \cap \text{vars}(C) = \emptyset$ ; (2)  $D$  的头部的谓词符号不出现在  $P_h$  的其他任何位置; (3)  $Q = \text{Body} \theta$ ; (4)  $C$  是子句  $(H:-P \quad \text{Body} \quad R) \text{confine}[\theta, \text{vars}(K)]$  的一个变体. 条件(4)保证了如果  $h=i$ , 并在程序  $P_{i+1}$  中展开派生子句  $F$  的原子  $K\theta$ , 我们将重新获得程序  $P_i$ . (4) 子句合成. 假设程序  $P_i$  有形式  $S \quad C, D \quad T$ , (可能被重新命名过) 子句  $C$  和  $D$  分别拥有形式  $H:-P \quad Q$  和  $H:-R \quad Q$ . 子句  $C$  和  $D$  合成后得到形如  $S \quad F_1, F_2, F_3 \quad T$  的程序  $P_{i+1}$ , 其中  $F_1: H:-A \quad Q, F_2: A:-P, F_3: A:-R$ . 并且 (i)  $A$  是新的谓词符号; (ii) 若在  $P_{i+1}$  的子句  $F_1$  中展开  $A$ , 可得  $S \quad C, D, F_2, F_3 \quad T$ . 子句合成规则易推广到  $n(n=2)$  个子句的情况. (5) 子句倒置. 通过倒置头部谓词符号不同, 但位置相邻的两个子句的顺序来构造程序的一个新版本. (6) 引入等式. 如果  $C$  是形为  $(H:-\text{Body})\theta$  的一个子句, 其中  $\theta = \{X_1/t_1, \dots, X_n/t_n\}$  是一个置换且  $\text{dom}(\theta) \cap \text{vars}(C) = \emptyset$  那么, 可删除  $C$  并插入下面两个子句以获得一个具有新版本的程序. 即  $H:-\text{ep}(X_1, t_1), \dots, \text{ep}(X_n, t_n) \quad \text{Body}, \text{ep}(X, X)$ , 其中  $\text{ep}$  是一个新的谓词. (7) 删除失败子句. 设  $C$  是程序  $P_i$  中形为  $H:-A \quad P$  的一个子句. 如果原子  $A$  与  $P_i$  中任何子句的头部均不能合一, 则子句  $C$  可以从  $P_i$  中删除, 从而获得  $P_{i+1}$ .

## 2 语义保持变换序列

**定义 6** 变换序列  $P_1, \dots, P_n$  是 Prolog 语义保持(简称为语义保持)当且仅当对于每一个  $i=0, \dots, n-1$ , 且对于每一个谓词在  $P_i$  中出现的原子  $A$ , 有  $\text{Answers}(P_i, A) = \text{Answers}(P_{i+1}, A)$ .

**定理 1** 定义规则的应用是语义保持的.

**证明** 因为该变换不会改变其谓词在原始程序中出现过的子句.

**定理 2** 子句体中最左原子的展开是语义保持的.

**证明** 令  $C$  是当前程序  $P_i = C_1, \dots, C_n$  的第  $k$  个子句, 即  $C = C_k$ . 假设: (1)  $C$  具有形式:  $H:-B \quad Q$ ; (2)  $H_1:-\text{Body}_1, \dots, H_m:-\text{Body}_m$  是  $P_i$  中所有头部可与  $B$  合一的子句的后件. 因此, 变换过的程序  $P_{i+1}$  等价于  $C_1, \dots, C_{k-1}, U_1, \dots, U_m, C_{k+1}, \dots, C_n$ , 其中  $U_j = (H_j:-\text{Body}_j \quad Q) \text{mgu}(B, H_j), j=1, \dots, m$ . 从而, 必须证明  $\text{Answers}(P_i, A) = \text{Answers}(P_{i+1}, A)$ .

因为重命名函数的一个特定选择, 对不动点值的 Ren-等式类没有影响. 所以, 可以省略  $\tau_i$  中的第 2 个参数. 对于每个函数  $f$  和目标  $G$ , 从  $\tau_i(f)[G]$  出发, 用  $\tau_i(f)$  替换函数  $f$  的取值可以得到  $\tau_i(f)[G]$ . 过程为

$$\tau_i(f)[\text{nil}] = \epsilon = \tau_{i+1}(f)[\text{nil}]$$

$$\begin{aligned} \tau_i(f)[F, T] = & \diamond \{ \text{confine}(\theta \text{ef}[\text{body}(C_i) \quad T] \theta), \text{vars}(F, T) \} \mid i=1, \dots, n) = \\ & \diamond \{ \text{confine}(\theta \text{ef}[\text{body}(C_i) \quad T] \theta), \text{vars}(F, T) \} \mid i=1, \dots, k-1) = \\ & \diamond \{ \text{confine}(\theta \text{ef}[B \quad Q \quad T] \theta), \text{vars}(F, T) \} \diamond \\ & \diamond \{ \text{confine}(\theta \text{ef}[\text{body}(C_i) \quad T] \theta), \text{vars}(F, T) \} \mid i=k+1, \dots, n) = \end{aligned}$$

$$\begin{aligned} & \diamond \{ \text{confine}(\Theta \text{ } \ell f[\text{body}(C_i) \text{ } T] \Theta), \text{vars}(F, T) \} \mid i = 1, \dots, k-1 \} \diamond \\ & \diamond \{ \text{confine}(\Phi \Psi_j \text{ } \ell f[\text{body}_j \text{ } Q \text{ } T] \Phi \Psi_j), \text{vars}(F, T) \} \mid j = 1, \dots, m \} \diamond \\ & \diamond \{ \text{confine}(\Theta \text{ } \ell f[\text{body}(C_i) \text{ } T] \Theta), \text{vars}(F, T) \} \mid i = k+1, \dots, n \}, \end{aligned}$$

其中  $\Theta = \text{mgu}[F, \text{head}(C_i)]$ ,  $\Phi = \text{mgu}(F, H)$ ,  $\Psi_j = \text{mgu}(B \Phi H_j)$ . 另一方面为

$$\begin{aligned} \tau_{i+1}(f)[F, T] &= \diamond \{ \text{confine}(\Theta \text{ } \ell f[\text{body}(C_i) \text{ } T] \Theta), \text{vars}(F, T) \} \mid i = 1, \dots, k-1 \} \diamond \\ & \diamond \{ \text{confine}(\mu_j \text{ } \ell f[\text{body}_j \text{ } Q \text{ } T] \Upsilon_j \mu_j), \text{vars}(F, T) \} \mid j = 1, \dots, m \} \diamond \\ & \diamond \{ \text{confine}(\Theta \text{ } \ell f[\text{body}(C_i) \text{ } T] \Theta), \text{vars}(F, T) \} \mid i = k+1, \dots, n \}, \end{aligned}$$

其中  $\mu_j = \text{mgu}(F, H_j)$ ,  $\Upsilon_j = \text{mgu}(B, H_j)$ . 由  $\text{mgu}$  的定义可知: (i)  $\text{confine}[\mu, \text{vars}(F, T)] = \text{confine}[\Phi \Psi_j, \text{vars}(F, T)]$ ; (ii)  $(\text{Body}_j \text{ } Q \text{ } T) \Upsilon_j \circ \mu_j = (\text{Body}_j \text{ } Q \text{ } T) \Phi \Psi_j$ . 给定一个程序  $P$  和形如:  $H: - Q \text{ } B \text{ } R$  的一个子句  $C$ , 称之为下列(i)和(ii). (i)  $C$  体中的原子  $B$  是确定的, 当且仅当  $P$  中刚好存在一个其头部可与  $B$  合一的子句. (ii)  $C$  中  $B$  的展开决定了一个反向约束传播, 当且仅当  $P$  中存在一个子句  $K: - \text{Body}$ , 其头部可与  $B$  合一, 并且原子  $(H, Q) \text{mgu}(B, K)$  的序列不是序列  $(H, Q)$  的一个变体.

**定理 3** 不能决定一个反向约束传播的确定的非最左原子的展开是语义保持的.

**证明** 令  $P_i = C_1, \dots, C_n$  为当前的程序,  $C$  为  $P_i$  中的第  $k$  个子句, 即  $C = C_k$ . 假设  $C$  有形式  $H: - Q \text{ } B \text{ } R$ . 令  $K: - \text{Body}$  为  $P_j$  中其头部能与  $B$  合一的唯一子句. 假设在  $C$  中展开  $B$  不能决定一个反向约束传播, 可知得到的变换程序为  $P_{i+1} = C_1, \dots, C_{k-1}, D_k, C_{k+1}, \dots, C_n$ , 其中  $D_k$  是子句  $H: - Q \text{ } (\text{body} \text{ } R) \text{mgu}(B, K)$ . 对于任何函数  $f$  和任何目标  $G$  类似定理 2, 可证明  $\tau_i(f)[G] = \tau_{i+1}(f)[G]$ . 定理 3 中的假设在某些情况下可能是偏强的, 但是通过使用子句合并规则和等式引入规则得到满足.

由定理 2 和定理 3 容易得出下面 3 个简单推论.

**推论 1** 若  $C$  和  $D$  都程序  $P$  中的子句, 则相对于子句  $D$  来说, 在  $C$  中折叠一个原子的序列是语义保持的.

**推论 2** 子句合并规则的应用是语义保持的.

**推论 3** 等式引入规则的应用是语义保持的.

**定理 4** 子句倒置规则的应用是语义保持的.

**证明** 原子  $A$  与头部谓词不同的子句不能合一. 因此, 该子句对表达式  $\tau_p(f)[A, T, V]$  的右边的作用是一个空的序列, 且独立于它所在的位置.

**定理 5** 失败子句删除规则的应用是语义保持的.

**证明** 类似定理 2, 使用 Vuillemin 的不动点不变式变换方法. 再根据事实“对于每一个出现在给定程序中的头部  $H$ ,  $\text{mgu}(A, H) = \text{Fail}$  且  $\text{mgu}(A, H) \text{ } \ell \dots = \text{Fail}$ ”; 即得以上结果.

**定理 6** 令  $P_0, \dots, P_n$  为一个语义保持变换序列. 假设对于某个  $i$ ,  $0 \leq i \leq n-1$ , 存在  $P_i, P_{i+1}$  满足下列(i)和(ii). (i)  $P_i$  的一个子句  $D$  的头部谓词  $h$ , 在  $P_i$  的任何其他地方不再出现. (ii)  $P_{i+1}$  是从  $P_i$  中展开  $D$  的体中的最左原子得到的. 对于子句  $D$  来说, 通过展开  $P_n$  的子句得到的  $P_{n+1}$  的变换序列  $P_0, \dots, P_n, P_{n+1}$  是语义保持的.

**证明** 对于  $D$  来说, 令  $C$  为被折叠的  $P_n$  中的子句. 令  $k$  为  $C$  的头部谓词, 则存在 (1)  $h$  和 (2)  $h = k$  两种情况. 在情况 (1) 中, 程序  $P_n$  有  $C^1, \dots, C, \dots, C^r, C^{r+1}, \dots, C^s$ .  $T$  形

式, 其中  $C_1, \dots, C_r$  是关于  $k$  的子句,  $C_{r+1}, \dots, C_s$  是关于  $h$  的子句(不同头部谓词的子句的位置, 可以用子句倒置规则交换). 因为  $P_0, \dots, P_n$  是语义保持变换的序列,  $P_n$  等效于  $C_1, \dots, C_r, \dots, C_s \rightarrow D \rightarrow T$ , 其中用子句  $D$  的替换了关于  $h$  的子句. 于是, 情况(1)简化为推论 1. 在情况(2)中, 对任意目标  $G$  由归纳法可证明  $\text{fix}(\tau_n)[G, \Phi] = \text{fix}(\tau_{n+1}[G, \Phi])$ .

### 3 结论

程序变换必须保证语义的等价性. 以模型论为基础的逻辑程序变换规则不适用于按传统的标准方法实现的 *Prolog* 系统. 本文引入了一个能刻画 *Prolog* 程序计算特性, 且能简化语义等价性证明的 *Prolog* 指称语义; 给出并证明了有关的语义保持变换规则. 该结果和规则也适用于一般的逻辑程序. *Prolog* 不但是一种著名的 AI 语言, 而且一直被认作是一种主要的、具有突出特点和优势的形式化规说明语言、原形工具和通用实现工具而加以应用<sup>[6~5]</sup>. 因此, 有关它的变换方法的研究, 具有广泛的重要意义.

### 参 考 文 献

- 1 应 晶, 吴朝晖, 何志均. 支持软件开发的变换方法[J]. 计算机科学, 1997, 24(5): 90 ~ 94
- 2 余金山. 析取演绎数据库否定信息的推理规则[J]. 华侨大学学报(自然科学版), 1999, 20(3): 303 ~ 307
- 3 Debray K, Mishra P. Denotational and operational semantics for Prolog[J]. J. Logic Programming, 1988, 5: 61 ~ 91
- 4 Cooke D, Gate A, Demirors E, et al. Languages for the specification of software[J]. J. System Software, 1996, 32: 269 ~ 308
- 5 Benichou M, Beringer H, Gauthier J M, et al. Prolog at IBM: An advanced and evolving application development technology[J]. IBM System Journal, 1992, 31(4): 755 ~ 773

## Semantics Hold Transformation for Prolog

Yu Jinshan

(College of Info. Sci. & Eng., Huaqiao Univ., 362011, Quanzhou)

**Abstract** The application of program transformation method to logic program is mainly based on such theory, namely, Fold/Unfold transformation holds the least Herbrand model semantics and the semantics of answer substitution set of logic program. However, when the leftmost computing rule and depth-first searching rule of standard Prolog systems are adopted for implementing logic programs, the semantics of original program cannot necessarily be held by transformed program. Moreover, the proof of semantic equivalence of the program is often difficult to understand and to read. By the style of algorithm for computing answer substitution sequence relating to the node of leaves in SLD-tree, the author puts forward a semantics definition of Prolog which is not difficult to understand; and then, gives the rules for semantics hold transformation and offers proof.

**Keywords** program transformation, Prolog, semantics