# 逻辑语言中启发式控制*

胡运发 胡子昂 高洪奎 卢肇川

(长沙工学院)

# HEURISTIC CONTROL IN LOGIC PROGRAMMING

Hu Yunfa Hu Ziang Gao Hongkui and Lu Zhaochuan

(*Changsha Institute of Technology*)

## ABSTRACT

This paper discusses how heuristic control information from logic program are get to improve the imcompleteness and inefficiency resulting from the control strategy in PROLOG language system. Several heuristic rules and the proving of their correctness are given. With the use of these rules, the efficiency of logic programming system and the semantics of logic programming language can be improved. At the end of this paper, a heuristic WAM(HWAM) is given, and several examples are listed to explain that HWAM is more efficient and more complete than WAM.

## 摘 要

本文论述从逻辑程序本身提取启发式控制信息，以克服由于逻辑语言系统中控制策略的机械性所带来的不完备性和低效性。具体地给出若干启发式控制规则，并证明了这些规则的正确性。运用这些控制规则可以大大地提高系统的运行效率或改善逻辑程序的语义性质。文章最后给出启发式WAM(记作HWAM)，并且用实例说明HWAM比WAM更有效，更完善。

## §1. 引 言

逻辑语言的倡导者Kowalski明确提出A=L+C(即算法＝逻辑＋控制)，强调算法的逻辑描述和控制分离，让用户关心客观世界的逻辑描述，让计算机注重

* 1989年8月9日收到，1989年10月19日定稿。

控制的有效实现，这无疑是人们努力追求的一个方向。但是，把控制留给系统的实现者，这导致系统实现上的艰巨性和复杂性。

在逻辑语言(例如prolog)发展的初期，系统的实现效率是很低的。到了1978年，DEC-10 Prolog的实现者Warren首先认识到从逻辑程序中提取必要的控制信息，会大大提高系统的运行效率。他根据程序的结构信息，实现了尾调用优化[1]，根据用户提供的mode说明，实现了子句选择的优化控制，使得prolog第一次达到和Lisp语言可比的运行效率，进一步巩固了prolog的历史地位。1983年，Warren进一步认识到子句定义中项素的类型信息，有助于提高一致化的运行效率，他设计了著名的Warren抽象机模型[2]，把推理机的运行效率提高了一个数量级。但是我们不能说Warren模型已经登峰造极，因为Warren模型仅仅局部优化了控制策略。在整体上，Warren模型的控制策略依然是机械式的深度优先加回溯。这一策略存在主要问题是：

1. 可能破坏解的完备性。

例如，深度优先不能求解如下程序

例一

$$
\begin{aligned}
&f(a,b).\\
&f(b,c).\\
&f(X,Y):-f(Y,X).\\
&f(X,Y):-f(X,Z),f(Z,Y).\\
&?-f(c,a).
\end{aligned}
$$

2. 运行效率低

例如，Warren模型不能进一步加速下列问题的求解过程：

例二

$$
\begin{aligned}
&fact(0,1).\\
&fact(N,F):-N_1 \text{ is } N-1,\\
&\qquad\qquad fact(N_1,F_1),\\
&\qquad\qquad F \text{ is } N*F_1.\\
&?-fact(10,X).
\end{aligned}
$$

例三

$$
\begin{aligned}
&p(X,Y):-q(X),r(Y),s(X).\\
&q(a).\quad q(b).\quad q(c).\\
&r(a).\quad r(b).\quad r(c).\\
&s(c).\\
&?-p(X,Y).
\end{aligned}
$$

本文的目的是研究启发式控制规则，以便产生某些控制信息，提高逻辑语言系统的运行效率或改善系统的完备性。

## §2. 改进逻辑语言系统效能的途径

改进逻辑语言的系统效能，人们已经花费了巨大的精力，大致有三种途径:

1. 允许用户显式地提供控制谓词，控制方式或控制标记。

例如普通的prolog中的cut谓词! ； parlog[3]或DEC-10 prolog中允许用户提供mode说明; Concurrent prolog[4]允许用户提供只读变量标记？X。Metalog[5]明确提出元级谓词概念，它的一般形式是:

<元谓词>: $-P_1, \cdots, P_n$. 其中$P_i$为用户定义的谓词或系统定义的谓词。

这一途径虽然具有很大的灵活性，但毕竟是不得已的办法。因为按kowalski的观点，A=L+C。变成了A=L+$C_1$ + $C_2$, 其中C=$C_1$+$C_2$, $C_1$为元级控制，$C_2$为系统内部标准的控制策略。当C → $C_1$时，$C_2$ → 0, 逻辑语言退化为常规语言。这就是说，此种方法损害了逻辑语言的特性和优点，把负担转嫁给用户的办法换取效率或改进性能。

2. 改变系统标准控制策略

既然深度优先加回溯是不完备的，且有时会导致低效，不如改变标准的控制策略，例如宽度优先或完全的或并行，例如Loglisp就是这样做的。宽度优先或完全的或并行既是完备的又无须回溯。但是宽度优先或完全或并行仍是一种盲目的、机械式的控制策略，仍然会造成时空方向的浪费。例如

$$\begin{aligned}&\mathrm{append}([\,],X,X)\\&\mathrm{append}([X|L_1],L_2,[X|L_3]):-\\&\qquad\mathrm{append}(L_1,L_2,L_3).\\&?-\mathrm{append}([1,2],[3,4],L).\end{aligned}$$

在搜索二叉树的过程中永远只有一条成功的分枝。因此，失败分枝上所有的工作，完全是不必要的浪费。(在并行的情况下，浪费相应的计算资源，在宽度优先的情况下，增加系统的开销)。

鉴于上述情况，有人主张受限的深度优先或受限宽度优先搜索。在受限深度优先的情况下，深度为D, 当D=1时为宽度优先。在受限宽度优先的情况下，宽度为M, 当M=1时，即为深度优先，但是根据什么原则来确定适当的D或M呢? 显然，这仍然是有待解决的问题。

3. 提高系统启发式控制能力

启发式控制的基本思想是从具体的逻辑描述出发，研究对象的各种关系，例如子句之间的关系，目标之间的关系，目标和子句的调用关系，项素类型和一致化过程的关系，提取控制信息，从而确定适当的控制策略，使系统更加高效或完善。

在过去的文献中，已经对于目标间的数据相关性(受限与并行或智能回溯)，目标与子句的调用关系(尾调用优化)，子句中项素类型与一致化操作的关系，作过深入的研究[6,7,8]。本文遵循这一路线，重点研究子句间的关系，从中提供控制信息，进一步提高或完善逻辑语言的系统效能。

## §3. 启发式元级控制规则

本文提出的启发式元级控制规则由下列定义和定理给出。

定义1: 对于某一目标G, 如果对其所有的非变量的自变元$A_i$, 使两子句$C_1$, $C_2$同时蕴含此目标G, 则称此两子句$C_1$, $C_2$是非异或的, 反之为异或的。

定义2: 一组子句是异或的, 当且仅当它们是两两异或的。

定理一. 如果两子句头不能一致化, 则此两子句必定为异或的。

证明: 令目标为A, 两子句为$A_1$:-$B_1$, $A_2$:-$B_2$

我们采用反证法。假定两子句是非异或的, 则存在mgu, $\theta_1$, $\theta_2$使得

$A\theta_1 = A_1\theta_1$　且$B_1\theta_1$为真

$A\theta_2 = A_2\theta_2$　且$B_2\theta_2$为真

令$\theta = \theta_1 \cdot \theta_2$则有

$A\theta = A_1\theta$　且$B_1\theta$为真

$A\theta = A_2\theta$　且$B_2\theta$为真

由此导出

$A_1\theta = A_2\theta$

与定理的条件矛盾。证毕。

定理二, 如果两子句构成一归纳定义, 其中一子句为基础情况, 另一子句为归纳情况, 则不可能存在一目标, 同时满足两子句。

证明: 归纳定义是良构的必要条件是在归纳定义中,有一子句的前提含有一次序增加(或减少)的自变量条件, 相应的自变量必定小于(或大于)基础情况的自变量值。两子句不能同时蕴含同一目标, 若蕴含了, 目标中相应的自变量必定等于基础情况子句中的自变量值, 归纳子句的前提成立, 造成相应的自变量值必定大于(或小于)基础情况中的自变量值, 产生矛盾。

在例二中, 第一子句为基础情况, 基础值为0, 第二子句是归纳情况, 与次序有关的条件是$N_1$ is $N$-1, 次序减小, 所以$N$必定大于0, 不可等于0, 如果$N$=0, 则$N_1$=-1, 相应的fact($N_1$, $F_1$), 不可能终止, 这是不可能的。

定理三, 如果有形如

$$A_1: \quad -P, B_1.$$
$$A_2: \quad -\neg P, B_2.$$

两子句, 且$A_1$和$A_2$可一致化, 则此二子句必定是异或的。

证明: $A_1 : -P, B_1$和

$A_2 : -\neg P, B_2$为异或的, 则存在一目标A, 两子句$A_1 : -P, B_1$和$A_2 : -\neg, B_2$均满足A, 即存在mgu$\theta_1$和$\theta_2$使得$A\theta_1 = A_1\theta_1$且$A\theta_2 = A_2\theta_2$, 同时$(P, B_1)Q$为真, $(\neg P, B_2)\theta$为真, 由此我们导出$\theta = \theta_1 \cdot \theta_2$, $A\theta = A_1\theta = A_2\theta$, 且$(P, B_1)\theta$和$(\neg P, B_2)\theta$为真, 即$P\theta$和$\neg P\theta$同时为真, 这是不可能的。

## §4. 启发式控制规则的应用

我们在第三节已经给出了若干启发式元级控制规则，并证明了它们的正确性。本节讨论它们的应用。

1. 应用启发式控制规则，改进WAM模型中的索引机制。

众所周知，WAM中的索引机制是加速逻辑程序运行效率的重要手段，因为它体现了某种启发式选择子句的策略。但是，用我们提出的定理一，二，三来衡量，WAM启发能力是不够的。根据定理一，二，三，我们至少可作如下扩充：

(1). 索引对象可不限于第一个自变元。

(2). 索引对象可不限于一个自变元。

(3). 索引对象可不限于子句头中常变元或结构自变元。

(4). 索引对象可扩展到子句头中为表的自变元，如果表的头元素为常量的话。

(5). 索引操作不限于等同比较，可扩展到一般的算术比较操作，例如等于，小于，小于等于，大于，大于等于等。

为了改进WAM中子句的选择能力，我们设计的索引指令为：

(1) Switch_typ_term $A_i, L_v, L_c, L_l, L_s$

它的含义是当$A_i$的类型typ为

a) 变量，则转移到$L_v$

b) 常量，则转移到$L_c$

c) 表，则转移到$L_l$；在转移之前取出对应的表头值。

d) 结构，则转移到$L_s$.

显然，除c)以外，(1)与WAM中的Switch-on-term是很相似的。

(2) Switch-op-constant, $A_i$, $\$(a_1 : L_1, \cdots, a_n : L_n, otherwise : L_{n+1})$.

其中op为$\{=, <, >, \leq, \geq\}$之一。此条指令的语义是(其中$\overline{op}$表示op的否定)：

IF $\bigcup_{j=1}^{n}$ $A_i$ $\overline{op}$ $a_j$ THEN $L_j$ ELSE

IF $\bigcap_{j=i}^{n}$ $A_i$ $op$ $a_j$ THEN $L_{n+1}, j = 1, \cdots, n$.

(3) Swithe_on_structure $A_i$, $\$(a_i : L_1, \cdots, a_n : L_n)$ 此指令除增加$A_i$外，和WAM中同名指令含义相同。

(4) Switch_op_listhead constant $S_i$ $\$(A_j : L_1$, otherwise:$L_2)$ 其中$S_i$表示当Switch_typ_term $A_i, \ldots$时，$A_i$的类型typ为list，则该表头元素放在$S_i$中，(4)的语义是：IF $S_i$ op $A_j$ THEN $L_1$ ELSE $L_2$

有上述扩充后，WAM改为HWAM。HWAM比WAM有了更强有力的子句选择能力。

2. HWAM比WAM更高效，下面举例说明：

例4. 梵塔问题

```
move (0, _, _, _).
move (N, L, C, R): -sub1(N, M), move (M, L, R, C), move(M, R, C, L).
```

按WAM编译后的代码为：

```
procedure move/4
 witch_on_term      3, 3, 2, 2          sub           x1, y4
3:                                      put_value_y   y4, x1
 try_me_else        4                   put_value_y   y1, x2
1:                                      put_value_y   y3, x3
 get_integer        0, x1               put_value_y   y2, x4
 proceed                                call          move/4, y4
4:                                      put_value_y   y4, x1
 trust_me_else      fail                put_value_y   y3, x2
2:                                      put_value_y   y2, x3
 allocate                               put_value_y   y1, x4
 get_variable_y     y1, x2              deallocate
 get_variable_y     y2, x3              execute       move/4, y0
 get_variable_y     y3, x4
```

图 1　move 的 WAM 代码

按 HWAM 编译后的代码为:

```
switch_on_term      3, 5, fail, fail
5:
switch_=_constant   A1, ¥ (0:1, otherwize:2)
```

图 2　move 的 HWAM 索引部分代码

上述代码仅索引部分，和子句相对应的代码，即标号为 1-4 同图 1。

实验证明，对于 move(_, _, _, _), HWAM 代码比 WAM 代码快 5 倍。更多的实例，我们将在附录中给出。

2. 应用启发式控制规则，可以在不降低效率的前提下，保证过程语义和说明性语义的一致性。一个典型的例子是求最大值程序。

例 5 求最大值程序

我们有三个关于求最大值程序。

```
max(X, Y, X): -X>=Y.    max(X, Y, X): -X>=Y, !.    max(X, Y, X): -X>=Y, !.
max(X, Y, Y): -X<Y.     max(X, Y, Y): -X<Y .       max(X, Y, Y).
      (1)                      (2)                        (3)
```

程序(1)的说明性语义和过程性语义是一致的。程序(2)尽管说明性语义和过程性语义是一致的，但效率高于程序(1)，代价是使用语义不清晰的 cut 谓词! 。程序(3)最高效，但却不保证说明性和过程性语义的一致性。遗憾的是，人们为了追求效率，常常大量使用类似于程序(3)那样的程序。

在 HWAM 中，要求人们书写类似程序(1)那样的程序，由于 HWAM 中使用了强有力的元级控制策略(即启发式控制规则)，使程序(1)具有程序(3)那样的效率。

3. 应用启发式控制规则，有利于确定适当的宽度界限 M。

正如前面所述，盲目增加或并行度或搜索宽度，并不一定导致高效。当一个过程中，同名子句两两异或时，使用启发式控制规则，可立即找到所需的子句。因此，此时确定宽度为1也许是适当的。一般说来，如果有$N$个子句，其中有$m$个子句互相异或，$(m < N)$，那么适当的宽度N可定为$N-m+1$。

例如append定义中两子句互相异或，$m=1$，不仅可求出解，而且效率最高。

例一中共有4个子句，其中两个子句互相异或，所以$m=4-2+1=3$，能确保求出解来。

## §5. 结束语

本文研究的启发式控制规则，能够解决某些类别的问题，适用于任何逻辑语言的实现系统。为了说明问题，本文运用启发式控制，改进了WAM，称做启发式WAM(记作HWAM)，运用实例说明，在运行效率和程序语义上，HWAM均优于WAM。但决不是说，该方法仅适用于顺序编译系统，它具有普遍意义，也适用于解释系统(Interpreter)或并行系统。文中所述方法已应用到高效的GKD-PROLOG/VAX编译系统中。附录给出更多的实例，为了强调启发式控制规则具有简单，有效等实用价值。

## 附 录

1. 快速排序

(1) 快速排序的定义

```
qsort([ ], Rest, Rest).
qsort([X| Unsorted], Sorted, Rest):-
    partition (X, Unsorted, Smaller, Larger),
    qsort (Smaller, Sorted, [X| Sorted 1]),
    qsort (Larger, Sorted 1, Rest).
partition (_, [ ], [ ], [ ]).
partition (A, [X|Xs], Smaller, [X|Larger]):-
    A<X, partition (A, Xs, Smaller, Larger).
partition (A, [X|Xs], [X|Smaller], Larger):-
    A>=X, partition (A, Xs, Smaller, Larger).
```

(2) 关于partition 谓词，WAM和HWAM编码对比:

| WAM | | HWAM |
|---|---|---|
| procedure partition/4 | | |
| 7: | | 8: |
| try_me_else | 5 | swith_typ_term A2, 7, 1, 9, fail |
| 1: | | 9: |
| get_nil | x2 | swith_<_listheadconstant $S_2$, |
| get_nil | x3 | ¥ (A1:2, otherwize:3) |
| get_nil | x4 | |
| proceed | | |
| 5: | | |
| retry_me_else | 6 | |
| 2: | | |
| get_list | x2 | |
| unify_variable_x | x5 | |
| unify_variable_x | x2 | |
| get_list | x4 | |
| unify_value_x | x5 | |
| unify_variable_x | x4 | |
| put_value_x | x2, x6 | |
| put_value_x | x5, x2 | |
| escape | </2, y0 | |
| put_value_x | x6, x2 | |
| execute | partition/4, y0 | |
| proceed | | |
| 6: | | |
| trust_me_else | fail | |
| 3: | | |
| get_list | x2 | |
| unify_variable_x | x5 | |
| unify_variable_x | x2 | |
| get_list | x3 | |
| unify_value_x | x5 | |
| unify_variable_x | x3 | |
| put_vablue_x | x2, x6 | |
| put_vablue_x | x5, x2 | |
| escape | >=/2, y0 | |
| put_value_x | x6, x2 | |
| execute | partition/4, y0 | |
| proceed | | |

(3) 简单说明，在WAM编码中没有索引指令，即partition是完全不确定的。在HWAM编码中，仅增加了两条索引指令，partition是完全确定的过程。

2. 地图着色问题

(1) 问题的定义

```
color(A, B, D, E):-next(A, B), next(A, C), next(A, D), next(B, C),
                   next(C, D), next(B, E), next(C, E), next(D, E).
```

```
next(X, Y): -next1(X, Y).
next(X,Y): -next2(X, Y).
next1(green, red).
next1(green, yellow).
next1(green, blue).
next1(red, blue).
next1(red, yellow).
next2(X, Y)|-next1(Y, X).
```

(2) 关于next1, WAM 与HWAM 编码对比:

| WAM 编码 | | HWAM 编码 |
|---|---|---|
| procedure next1/2 | | |
| switch_on_term | 6, 12, fail, fail | swith_typ_term A1, 6, 20, fail, fail |
| 6: | | 20: |
| try_me_else | 7 | swith_=_constant A1, |
| 1: | | ¥ (green: 21, red: 22, |
| get_atom | green, x1 | otherwise: fail) |
| get_atom | red, x2 | |
| proceed | | |
| 7: | | 21: |
| retry_me_else | 8 | swith_=_constant A2, |
| 2: | | ¥ (red: proceed, |
| get_atom | green, x1 | yellow: proceed, |
| get_atom | yellow, x2 | blue: proceed, |
| proceed | | otherwise: fail) |
| 8: | | |
| retry_me_else | 9 | |
| 3: | | 22: |
| get_atom | green, x1 | swith_=_constant A2, |
| get_atom | blue, x2 | ¥ (blue: proceed ; |
| proceed | | yellow: proceed, |
| 9: | | otherwise, fail) |
| retry_me_else | 10 | |
| 4: | | |
| get_atom | red, x1 | |
| get_atom | blue, x2 | |
| proceed | | |
| 10: | | |
| trust_me_else | fail | |
| 5: | | |
| get_atom | red, x1 | |
| get_atom | yellow, x2 | |
| proceed | | |

WAM 编码

```
12:
 switch_on_constant   2
 green 13
 red 14
13:
 try                  1
 retry                2
 trust                3
14:
 try                  4
 trust                5
procedure next2/2
 get_variable_x       x3, x1
 get_variable_x       x1, x2
 put_value_x          x3, x2
 execute              next1/2, y0
 proceed
```

(3) 简单说明: HWAM 索引指令远比WAM 索引指令有效。

## 参考文献


[1] Warren, D.H.D, Applied Logic-Its Use and Implementation as a Programming Tool, Technical Note 209, AI Certer, SRI International, June 1983.

[2] Warren, D.H.D., An Abstract Prolog Instruction Set, SRI Technical Note 309, Oct, 1983.

[3] Clark, K. L & Gregory, S, Parlog: Parallal Programming in Prolog, Imperial College, Research Report, Dec. 84/4, 1984.

[4] Shapiro, E. Y,
A Subset of Concurrent Prolog and Its Interpreter, Tech Report TR-003, Institute for New Generation Computer Technology, Tokyo [1983].

[5] Dinchas M. & Pierre Le Pape J.,
Metacontrol of Logic Programs in METALOG, Proceedings of the International Conference on Fifth Generation Computer Systems, 1984.

[6] J. H. Chang & A.M. Despain,
Semi_Intelligent Backtracking of Prolog Based On a Statis Data Dependency Analysis, Proc. of IEEE Symposium on Logic Programming, PP, 10-21, Aug, 1985.

[7] Peter Kursawe, How to Invent a Prolog Machine, In Proc of the Third Int. Conf. On Logic Programming, July, 1986.

[8] 胡运发, 执行Prolog 的项流模型, 知识工程进展, 中国地质大学出版社, 1988。