国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第13卷第2期 1991年6月 Vol. 13 No. 2

Prolog 编译器中暂时变量分配方法研究

黄志球 张晨曦

（南京航空学院计算中心） （国防科技大学计算机系）

摘 要 文中研究对 prolog 编译器的暂时变量的分配，介绍了回溯分配法，着重介绍了一种高效的无回溯分配法的实现原理及其优点。

关键词 prolog 程序，编译器，WAM 模型，暂时变量、暂时变量分配

分类号 TP314

1 暂时变量的分配

采用编译技术是一种可行的 prolog 高效实现方法。文中选择 warran 抽象指令集[1]作为目标代码。这是因为 WAM[1]是一个非常高效的 prolog 编译执行模型[2]。

编译技术的重要的问题就是为变量分配生存单元，用于存放其约束值或约束值的指针[3]。在 prolog 程序中，可以按生存方式将变量分为两类，即暂时变量和永久变量。

生存期：设子句的一次执行过程中，变量 X 首次出现时刻为 t1，最后一次出现时刻为 t2，那么时间区间 [t1，t2] 就是变量 X 的生存期。

暂时变量：生存期不跨越非内部谓词过程调用指令 call 的变量称为暂时变量。反之，则称为永久变量。

鉴于寄存器高速存取的特性，为了提高效率，传统语言的编译器总是尽可能多地将寄存器分配给局部变量或中间临时变量[3]。基于同样目的，考虑到暂时变量的生存期不跨越 call，因而在其生存期内，寄存器内容不被破坏这个因素，我们便可以将寄存器分配给暂时变量，这个分配过程称为暂时变量的分配。

在 WAM 模型中，寄存器在执行时，用于实现调用目标和被调用过程之间的变元传递。这要求在执行过程调用时，第 i 号寄存器必须装载调用目标的第 i 个变元。因此变元和寄存器之间存在着一种约束关系。由于 WAM 模型下，变元传递与暂时变量共用一组寄存器，约束关系便同时制约着暂时变量分配。若违背这个约束关系，便会使得某一寄存器在某一确定时刻代表多个内容，引起有效信息丢失，导致错误执行，这种现象称为冲突。例如子句：

例 lh (A, B): —g (B, A)

国家自然科学基金资助项目
1990年8月25日收稿

若将寄存器 R1 分配给暂时变量 A，则当执行到子目标时，便违背了约束关系 B—R1，这样 R1 中保存的子句头中 A 的内容便被 B 冲掉，导致子目标 g (B, B) 的错误执行。所以暂时变量分配必须满足约束关系，即不导致冲突。

美国加州大学 Berkerly 分校研制的 PLM[4]，采用了一种带回溯的暂时变量分配法。下面简单介绍并分析这种方法，并据此提出一种高效的无回溯分配法，重点说明其实现原理。

2 回溯分配法简述

回溯分配法是一种临时分配法，它由美国的 Peter Van Roy 提出并实现，该方法主要由三个部分组成。

2.1 回溯分配法的组成

(1) 变量表的计算

变量表是一个反映变量出现次序的一个排列表。对于例 1，其变量表如下：

[arity(2),A,B,arity(2),B,A,fence(g)]

其中 arity 为每一个目标变元序列的起始标志，括号中数值为目标的变元数，fence 为非内部谓词过程调用目标之变元序列的末尾界限标志项。

(2) 生存期表的计算

生存期表通过正向和反向两遍扫描变量表计算获得。它记录了时序中各个时刻点存活着的暂时变量，即反映了暂时变量的生存期。

例 1 的生存期表如下：

[[ ], [ ], [A], [A, B], [A, B], [A], [ ], [ ]]

(3) 暂时变量分配

该过程根据暂时变量出现次序，尝试性地依次为之分配寄存器，同时利用变量生存期表检查是否发生冲突。若冲突，则回溯到引起冲突的分配处重新分配；若不冲突，则继续向下进行分配，直至分配完毕且不发生冲突。分配过程中，若待分配的暂时变量是第 i 个变元，则优先考虑将 Ri 分配给它。这种 Ri→Ai 的分配是一种优化分配，它可以减少目标代码中用于变元传递的一条 put 或 get 指令。

分配过程的核心部分如下：(详见文[4])

```
alloc (Var, Alive, N): —N notin Alive, Var=R (N).
alloc (Var, Alive, N): — (cause (none); cause (N), retract (cause (none)), assert (cause
                          (none))),
                        N1 is N+1, alloc (Var, Alive, N1).
```

其中，Alive 是仍然被使用之寄存器的集合，R (N) 代表第 N 号寄存器，cause(N)表示冲突原因为第 N 号寄存器，cause (none) 表示无冲突。

2.2 回溯分配法的不足

回溯分配法是一种可行的且被普遍接受的暂时变量分配法。我们曾采用该方法实现过一个 prolog 编译器[2][5]，但进一步的开发研究使我们看到它有如下的局限和不足：

(1) 回溯法“尝试”的特性，使其不可避免失败分配而产生的冗余计算。算法研究表明[6]，回溯法的复杂性与穷举法几乎都是指数形式数量级，只是减小了时间复杂性表达式的常数因子；最坏情况下等同于穷举法。这便降低了编译器的执行效率。

(2) 回溯法蕴含的优化分配是不完全的，它只能实施在暂时变量的首次出现处。这由下例可见。

例 2　　h(1,2,A):-B=f1(A),C=f2(B),D=f3(B),g(C,D).

若采用回溯法，其得出结果如下（打点处表示优化分配，下同）：R3→A　R1→B　R2→C　R3→D，而最佳方案是：R3→A　R4→B　R1→C　R2→D. 后者优化率是前者的 3 倍。这主要是由于回溯法按出现次序进行分配，优化可能若不存在于首次出现处，则当遍历到后续出现时，该暂时变量分配已定，而丧失了优化分配的机会。

(3) 回溯法适宜采用 prolog 等逻辑语言实现，但要求系统必须提供 assert、retract 等动态内部谓词[2][4]。若编译器采用传统语言实现，则很难采用回溯分配法[7]，所以说回溯分配法不便于实现。

3　无回溯分配法

针对回溯分配法的不足和限制，我们提出了无回溯分配法，其核心在于提供“预见”冲突和所有优化的可能性，提高编译效率及目标代码质量。

3.1　数据的组织

通过对子句的一次自左向右的扫描，构造出以下三个数据区。

(1) 变元——寄存器约束关系时序表　.

该表（下简称时序表）类似于回溯分配法中的变量表，区别在于它显式记录了变元和寄存器之间的约束关系，并显式地标明了其发生的时刻（据此表示时序关系）。

(2) 寄存器引用区间复合（SR）

对于每个寄存器，建立一个引用区间集合，其元素是该寄存器在子句中用来传递变元时被引用的时间区间（简称引用区间）。引用区间集合初值为空，当子句中发生一次 Ri 的引用，便将其引用区间 [t1，t2] 作为一个新元素并入 SR′(i)，即：SR(i)=SR′(i)∪{[t1,t2]}（SR′（i）为 SR（i）的旧值）。

子句中寄存器的每次引用，是通过扫描时序表发现的。若时序表中 t 时刻的约束关系是 argi→Ri，则说明发生了一次对 Ri 的引用：

A：若处于子句头，则该次引用区间为 [1，t]；

B：若处于子句体，则该次引用区间为 [t，t′]，(t′=t+Ar+1−i，Ar 是变元数，即 t 时刻之前最近之 arity（Ar）中所记录的变元数目 Ar)。

(3) 暂时变量激活时刻集合（ST）

对于任意一个暂时变量 X，建立一个对应的激活时刻集合 ST（X），其中元素是时序表中那些发生了 X 的一次激活使用的时刻点，所有暂时变量的激活时刻集合初值为空，当扫描时序表时，发现 t 时刻的约束关系中包含暂时变量 X，则说明 X 在该时刻被激活使用，则做 ST(X)=ST′(X)∪{t}(ST′(X)为 ST(X)的旧值)。

根据以上的描述，可以构造出例 1 的三个数据区，分别如图 1，图 2，图 3.

时序值	约束关系
1	arity (2)
2	A—R1
3	B—R2
4	arity (2)
5	B—R1
6	A—R2
7	fence (g)

图 1 时序表

SR(1)={[1,2],[5,7]}
SR(2)={[1,3],[6,7]}
SR(i)=Φ
(i=3,4,…,m;m 为寄存器数)

图 2 寄存器引用区间集合

ST(A)={2,6}
ST(B)={3,5}

图 3 暂时变量激活时刻集合

3.2 冲突判定规则

首先引入三个记号：

(1) S_lifetime (X)：它是由暂时变量 X 的生存期内所有时刻点所构成的集合。由于 ST (X) 中最小元素 t1 和最大元素 t2 分别表示 X 的首次出现和最后一次出现时的时刻，故有 S_lifetime (X) = {t1, t1+1, …, t2}

(2) S_reference $(Ri)_k$：它是由寄存器 Ri 的第 k 次引用区间（即 SR (i) 中的第 k 个元素）中的所有时刻点所构成的集合，设 SR (i) 的第 k 个元素为 [t1, t2] 则有 S_reference $(Ri)_k$= {t1, t1+1, …, t2}.

(3) T $(Ri)_k$：它是 Ri 的第 k 次引用的作用时刻，即实现变元传递（对应执行 get 或 put 指令）的时刻。设 SR (i) 中第 k 个元素为 [t1, t2]，由前面 SR (i) 的构造方法可知，若 t1=1，则 T $(Ri)_k$=t2；反之，若 t1≠1，则 T $(Ri)_k$=t1.

利用上述三个记号，可以将冲突判定规则表述如下：设 X 是暂时变量，且

$$S_k = S\_reference\ (Ri)_k \cap S\_lifetime\ (X)$$

对于任意 k，$1 \leqslant k \leqslant n$ (n 为 Ri 的引用次数)，分配 Ri→X 导致冲突的充要条件是：

(1) 若第 k 次引用发生在子句头，且

$$S_k - \{T\ (Ri)_k\} \neq \Phi$$

或

(2) 若第 k 次引用发生在子句体，且

$$S_k \neq \Phi，且\ T\ (Ri)_k \notin ST\ (X)$$

则分配 Ri→X 必然导致冲突，反之亦然。

我们采用举例方法说明规则的正确性而略去证明。

对于例 1，若将 R1 分配给 A，即 R1→A，由图 2、图 3 可得：

$$S-reference\ (R1)_2 = \{5, 6, 7\}$$

$$S—lifetime\ (A) = \{2, 3, 4, 5, 6\}$$

所以

$$S_2 = S\_reference\ (R1)_2 \cap S\_lifetime\ (A)$$
$$= \{5, 6, 7\} \cap \{2, 3, 4, 5, 6\}$$
$$= \{5, 6\} = \Phi$$

因为 R1 的第二次引用发生在子句体，故有

$$T\ (R1)_2=5\notin ST\ (A)\ =\ \{2,\ 6\}$$

所以，R1→A 的分配必然导致冲突。

3.3 分配及优化分配

无回溯分配法的分配过程可以按照任意顺序进行。若现在正对暂时变量 X 进行分配，则依次扫描 ST (X)，对于其中元素 t，查时序表中 t 时刻的约束关系。若其为 X—Ri，则考虑分配 Ri→X，利用冲突判定规则判是否冲突。(1)若不引起冲突：则确定此次分配，同时将 X 的生存期记录下来（此时间区间内 Ri 不再分配给其它暂时变量）。(2)若引起冲突：则取 ST (X) 中的下一个元素，重复上面的操作；若此时 ST (X) 已经扫描完毕，即 X 的所有优化可能均已尝试，且均被判定引起冲突，则暂不对 X 进行分配。当所有暂时变量的所有可能的优化分配均已尝试后，再对那些不可进行优化分配的暂时变量，顺序地分配寄存器，直到所有暂时变量分配完毕为止。

由此可知，无回溯分配法考虑了 ST (X) 中提示的所有优化分配可能性，故而是完全实施优化的。

4 析取目标的处理

为了克服二义性，规定：一个变量若是暂时变量，则其在子句中任一路径上都是暂时变量。析取目标中暂时变量的分配与合取目标的不同之处有三点：

4.1 时序表构造中分支路径时序值的标定

图 4 有几条分支路径的析取目标示意图。其中 A/B 分别是入/出口点，t(A)/t(B)代表 A/B 点的时序值，t_start (i) /t_end (i) 分别是分支起始/终止点处的时序值，则构造时序表的方法是：顺序扫描子句。

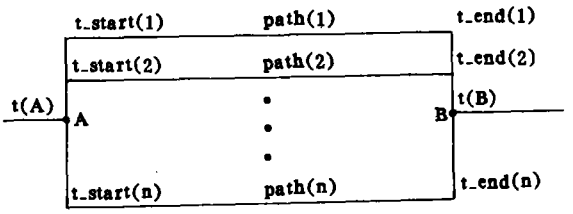

图 4

(1) A 点之前的合取目标，构造方法同前（设其中最大时序值为 t_0）.

(2) A 点处时序值 t (A) $=t_0+1$，该处约束关系为分支标志项 branch (n)，其内容如图 5.

(3) 顺序扫描 path (1)，t_start (1) =t (A) +1，path (1) 上合取目标的处理同前。

(4) 依次地，如(3)一样顺序构造 path(2)，…，path (n)，路径上的时序表，其中 t_start (i) =t_end (i−1)。

(5) 扫描至 B 点处，t (B) =t_end (n)，并继续向后扫描，构造完整的该子句的时

序表。

初看上去，path (i+1) 上的时序值大于 path (i) 上标定的时序值，不符合析取目标并行的特征，但由于分支路径之间相互独立，故而它们的时序值也是互不相关的，所以只要时序值关系保证了 path (i) (i=1, 2, …, n) 上的时序值大于 t (A) 且小于 t (B) 即可。实际上构造时序表时，可按任一顺序扫描 n 条分支路径。

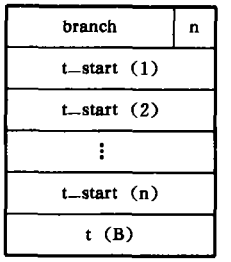

图 5

4.2 生存期的计算

采用上述时序值标定法，使得析取目标中的生存期在数值上不是连续的。对于不同分支有不同的生存期。计算的方法有两种：

设：S= {t1, t1+1, …, t2} (t1/t2 是 ST (X) 中最小/大值)

(1) 屏蔽法：设非 path (i) 的其它分支路径 path (j) (j=1, 2, …n; 且 j≠i) 上的时序值集合为：

$$S\_mask\ (i) = \{t\ (A),\ t\ (A)+1,\ \cdots,\ t\_end\ (i-1)\} \cup \{t\_start\ (i+1),\ t\_start\ (i+1)+1,\ \cdots,\ t\ (B)\}$$

则穿过 path (i) 的执行过程中 X 的生存期 S_lifetime (X) 为：$S\_lifetime\ (X)_i = S\text{-}Smask\ (i)$.

(2) 相交法：设 path (i) 上所有时序值的集合为 S_path (i)，通过 path (i) 的该次子句执行路径上所有时序值的集合为 S_execute (i)，子句中最大时序值为 t_end. 则有：

$$S\_path\ (i) = \{t\_start\ (i),\ t\_start\ (i)+1,\ \cdots,\ t\_end\ (i)\}$$

$$S\_execute\ (i) = \{1, 2, \cdots, t\ (A)\} \cup S\_path\ (i) \cup \{t\ (B),\ t\ (B)+1,\ \cdots,\ t\_end\}$$

$$S\_lifetime\ (X) = S \cap S\_execute\ (i)$$

4.3 分配过程

析取目标中为一个暂时变量分配寄存器，必须用冲突判定规则来检测是否在任一分支上都不产生冲突，而合取目标中只需检测一次。实际上合取目标可以看成是析取目标的一种特例。

5 结 语

确定性是编译技术的一个特点。我们提出的无回溯方法将以往非确定性的暂时变量分配转化为确定性的过程，这就带来了如下的好处：一是克服了冗余计算，提高了编译效率；二是可以完全实施优化，减少目标代码的数量，获得高质量的目标代码。除此之外，无回溯分配法既可采用 prolog 等逻辑语言实现，也适于用传统语言实现（而回溯分配法很难映射到传统语言实现），所以拓宽了编译器的实现范围。

为了说明问题，本文采用集合的数据形式来描述算法，没有给出实现时的详细的数据结构。我们采用 Turbo—c 实现了 PC—PP 编译器，为之采用了无回溯分配法[7]，采用了其它优化技术进一步提高效率。限于篇幅，这不再赘述。

本课题是在胡守仁教授的悉心指导下进行的，在此表示衷心感谢。

参 考 文 献

[1] Warran D H D. An Abstract Prolog Instruction Set. Technical note 309, SRI Internation, Oct., 1983

[2] 张晨曦. 基于 Warran 抽象机的 Prolog 实现技术的研究. 国防科技大学博士学位论文, 1987

[3] 陈火旺等. 程序设计语言编译原理. 国防工业出版社, 1984

[4] Peter Van Roy. A Prolog Compiler for the PLM. Master's Thesis, Computer Science Division, University of California, Berkerley, Aug., 1984

[5] 张晨曦, 李良良. prolog 编译器的设计. 人工智能学报, 1987

[6] 曹新谱. 算法设计与分析. 湖南科学技术出版社, 1984

[7] 黄志球. 编译型 prolog PC—PP 软件系统的实现研究, 国防科技大学计算机系硕士论文, 1990

A Study of Temporary Variable Allocation in Prolog Compiler

Huang Zhiqiu

(Computer Center of Nanjing Aeronautical Institute)

Zhang Chenxi

(Computer Department of National University of Defense Technology)

Abstract

This paper introduces a study of temporary variable allocation in prolog compiler. It gives a new allocation method without backtracking. The method is compared with the old one with backtracking and the advantages of the new method is introduced.

Key words prolog programming, compiler, WAM model, temporary variable, temporary variable allocation

---

我国“七五”计划十大电子科技成果

我国机电部、中国电子报社和有关专家通过严格评审，评选出“七五”期间十大科技成果。

1. 砷化镓超高速集成电路及单片低噪声放大器

1990 年由机电部 13 所和 55 所分别研制成功的砷化镓超高速门阵列和砷化镓微波单片集成电路，性能指标达到国际 80 年代中期产品水平，可广泛用于超高速计算机、电子对抗、卫星通讯和微波测试仪器等方面，它的研制成功标志着我国砷化镓集成电路开始走上实用化道路。这两项成果已通过机电部鉴定。

2. 高能离子注入机

LC4—700KeV 高能离子注入机，1988 年 4 月由机电部 48 所研制成功。这项成果在光集成、发光材料、金属材料、绝缘材料和超导材料等研究方面，均有广泛的应用前景，技术性能指标接近 80 年代国际先进水平，被誉为注入机的“亚洲之最”。曾获 1989 年机电部科技进步特等奖，1990 年被评为国家科技