国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第10卷第2期 1988年6月 Vol.10.No.2

一种结构化的编译型PROLOG数据库及其管理策略

李艮艮 慈云桂
（计算机科学系）

摘 要 PROLOG数据库存放着构成PROLOG程序的子句以及表示成子句的全局数据，它的组织与管理是实用PROLOG系统的关键，这一点在提高执行效率的编译型PROLOG系统中尤为如此。本文以YH-SIM——一个扩充了的WAM模型为基础，提出一种新颖的编译型PROLOG数据库结构形式，其中

- 过程中的子句索引与子句彼此分离；
- 子句索引块分成多个专门的子索引块；
- 一个子句按两种形式存放，子句代码和子句项（源子句形式）。二者分离且共享同一索引，因此代码库和项库的管理是一体化的。

这些结构化的特征，有力地支持PROLOG数据库的操作及管理（子句的插入删除，以及空间分配与回收）。文中还介绍了一种简明有效的管理方法。

关键词 PROLOG，逻辑程序设计，数据库，空间回收

1. 引 言

PROLOG语言作为一种人工智能程序设计语言，受到愈来愈广泛的重视，其原因不仅在于它有着良好的数学基础（一阶Horn逻辑）而且在于它提供了功能丰富的内部谓词，这些内部谓词的实现效率对于整个PROLOG系统的实用性和效率，都是十分重要的[4]。本文介绍一种新的编译型Prolog数据库结构以及相应的管理方法，它能有效地支持了一类重要的内部谓词——数据库操作类谓词（简称DBOP谓词）的快速实现。

PROLOG数据库系由构成程序的子句以及表示成子句的全局数据结构等内容所组成的。DBOP谓词的功能即向数据库中动态地插入或删除子句。由于DBOP谓词将子句作为操作数，因而在编译型PROLOG系统中，子句需要以两种形式存放：即代码形式和某种数

国家自然科学基金资助项目
本文1987年11月20日收到

据形式(项)。本文第二节首先介绍一个新的顺序PROLOG执行模型—YH—SIM．YH—SIM系以著名的WAM模型为基础，扩充了相对于PROLOG中非逻辑成份、尤其是DBOP谓词的执行机制。第三节介绍一种在YH—SIM支持下的PROLOG数据库结构形式，以及相应的管理策略。第四节介绍了子句的表示方法，这是编译型系统的重要问题之一。文中对几种表示方法的优缺点作了评价。

假定读者熟悉WAM模型。

2. YH—SIM模型简介[6]

YH—SIM以WAM模型为基础扩充了非逻辑成分的执行机制，下面首先介绍WAM模型的基本特点及其不足，然后介绍YH—SIM模型所作的扩充部分。

2.1 WAM模型

WAM模型系由D.H.D.Warren提出[2]，旨在提高PROLOG程序的执行效率。作为顺序Prolog系统研究领域的既成事实的基础和标准，WAM模型有如下两个基本特点：

1．顺序解释原理

通常，人们将PROLOG问题求解空间描述成搜索树，而PROLOG问题求解的过程就是一个在搜索树中寻找一个子树——证明树的过程。顺序解释原理，系采用深度优先的搜索策略、以及归结与回溯机制，逐步地构造证明树。搜索的结果，或者所解的问题无解，或者找到一个解，而为求该解所遍历的搜索子树(不包括因失败或回溯而剪去的部分)，即是相应于此解的证明树。

这样，WAM模型中的存储空间由如下几个部分组成(见图1)，以支持上述求解机制。其中代码区存放PROLOG程序（代码形式的规则和事实）；控制区记录着逐步构造的、非完全的证明树信息；数据区用以存储动态生成的数据结构。控制区和数据区按栈方式管理，这是归结和回溯机制所要求的。

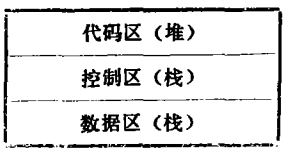

图1　WAM的存储区域划分

2．抽象机指令系统

WAM提供了一个较低级别的指令系统（与PROLOG语言相比）。借助于编译器[3]，PROLOG程序被转化为功能相对简单和确定的指令序列。与传统的解释型PROLOG系统相比，原来一些隐式地表示的复杂操作，如参数的传递和取代、结构数据的表示和动态生成、以及执行流程的控制等等，都通过WAM指令而显式地表示出来。从而导致了PROLOG程序执行效率的大幅度提高。WAM指令包括用于参数传递的get和put类指令，用于构造结构数据的unify类指令，用于目标调用和控制执行流的call类指令，以及用于快速检索候选子句的indexing类指令等等。

不难看出，WAM有力地支持了顺序PROLOG语言的逻辑程序设计成分(指归结、一致化和回溯等)。然而对于非逻辑成分的支持，却有待扩充。例如，WAM模型对控制区、数据区的管理，提供了专门的指令(如allocate和deallocate)，而对于PROLOG数据库（代码区）的管理则几乎没有涉及。进而言之，数据库操作类谓词（DBOP谓词）系以子句为基本单位，对数据库施行插入、删除操作。而在WAM模型中，一个过程是一个

单位，过程中的子句代码与子句间的索引代码不加区分，故上述谓词是难以实现的。最后，WAM是一个编译型的模型，数据和程序具有相当大的区别，DBOP 谓词要对子句的数据形式进行操作。如何在动态中获得库中子句的数据形式，也是待解决的问题之一。

2.2 YH-SIM——一个扩充的WAM模型

与WAM相比，YH-SIM增加了对PROLOG的非逻辑程序设计成分的支持机制，扩充了主要工作栈的功能。下面列出一些具体的功能指令。

1. 通用内部谓词接口指令——escape

"eacape Name/Arity" 的功能与传统计算机中的I/O转子指令类似。操作数Name和Arity描述了所要调用的内部谓词的名字和变元数。

2. 专用内部谓词指令

有些内部谓词使用频度很高，如类型测试类 谓词 var、nonvar、atomic、…，程序控制类谓词cut等等，为之单独设置指令，而不借助通用调用指令escape. 这样扩充，有助于提高执行效率。

3. 模块说明指令

前面曾经指出，DBOP 类谓词的操作对象是数据库中的过程，乃至更基本的单位—子句。此外，为了使PROLOG能够支持多种风格的程序设计，比如支持面向目标的程序设计风格，也需要对PROLOG加以扩充，使之具有模块化特征。为此，在抽象PROLOG机这一层次，我们提出了两条模块说明指令：

i) 过程说明指令 "procedure PDB 0"

这是编译器生成的过程代码的首条指令，其操作数 PDB 0 给出了有关该过程若干说明性信息，如过程的种类、索引特征等等；

ii) 子句说明指令 "clause CDB 0"

这是编译器生成的子句代码的首条指令，其操作数CDB 0给出了有关该子句的若干说明性信息，如子句的索引标志等等。

3. 编译型PROLOG数据库的一体化结构及其管理策略[8]

3.1 结构

在 YH-SIM 系统结构的支持下，我们提出了编译型 PROLOG 数据库的一种结构形式(见图 2)。其中

1. PROLOG数据库中的过程是由过程定义表PDT进行登记和管理的。PDT由过程定义块PDB组成，一个PDB对应数据库中定义的一个谓词；

2. 一个过程由子句及子句索引两部分组成，且彼此分离；

3. 子句索引按照具体的索引机制，又分为多个专门的子索引块；

4. 一个子句按两种方式存于数据库中，即代码形式和数据（项）形式，且通过唯一的子句说明块CDB联系在一起。

上述的数据库结构作为整个PROLOG数据库实现的基础，有如下特点：

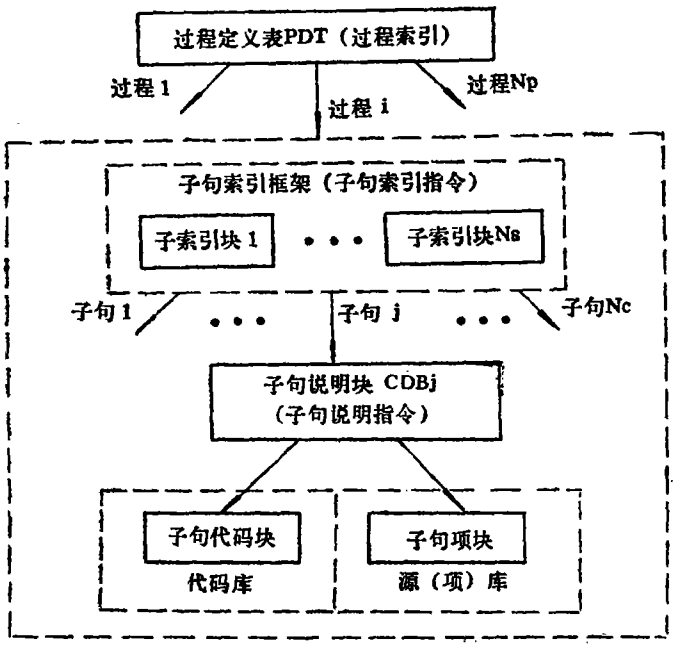

图 2　编译型Prolog数据库的一体化结构

（1）子句作为数据库的基本存储单位，使得数据库的基本操作如子句删除、插入以及动态的存储分配与回收成为可能；

（2）子句索引框架分成多个子部分，将导致索引重构的开销降低。子句索引框架的建立系对组成过程的子句进行静态分析的结果。PROLOG的数据库操作改变了过程的子句组成，从而要求对修改过的过程重新构造索引框架。索引框架分成若干个子部分，能够将索引重构的范围限制在某些子块中；

（3）原则上，编译型PROLOG数据库由两部分组成，即代码库和源程序库。代码库放代码形式的子句，而源程序库存放数据形式的子句。在 YH-SIM 的 PROLOG 数据库中，子句的两种形式由子句说明块CDB统一管理，且共享同一索引框架。因此，

• 代码库管理和源程序库管理是一体化的，管理开销降低一半；

• 数据库操作（访问数据形式的子句）和逻辑推理（执行子句代码）可以一致地使用过程中的索引框架，从而获得了

a）　逻辑与非逻辑成分实现的一致性、简洁性，以及

b）　数据库操作的高效率。

3.2　管理

下面介绍与数据库管理有关的重要数据结构以及相应的几个基本操作，它们是实现 PROLOG 数据库操作（DBOP谓词）的基础。至于DBOP谓词的实现细节，这里不展开讨论，详见[1]。

1．　两个重要的数据结构——PDB和CDB

（1）过程定义块PDB

PDB的主要内容如图 3 所示，其中

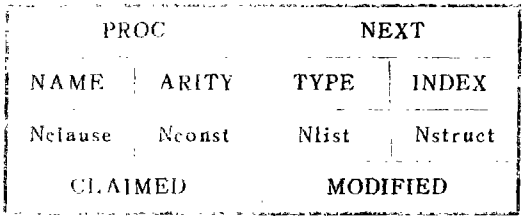

图 3 过程定义块PDB

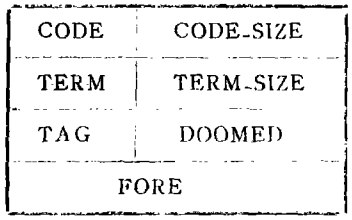

图 4 子句说明块CDB

PROC: 过程体在数据库中的地址;

NAME/ARITY: 过程名及过程的变元数;

NEXT: PDT指针，指向下一个同名但不同变元数的过程PDB;

Nclause: 过程子句数;

Nconst, (Nlist、Nstruct):

子句头首变元能与常数（表或结构）匹配的子句数;

TYPE: 过程类型。例如 System_Owned/User_Defined、DBOP/NON_DBOP等（DBOP指允许对该过程施行数据库操作）;

INDEX: 过程子句的索引状态。子句的索引情况会随着过程中子句的增加和删除而发生变化。为了快速实现索引重构，我们定义了索引状态、以及状态之间的转换规则;

CLAIMED: 过程使用标志;

MODIFIED: 过程修改标志。

(2) 子句说明块CDB

CDB是子句说明指令 clause 在数据库中的实际操作数。其部分内容由编译器生成。CDB介于子句索引与子句之间，并起着联系两种不同形式子句的作用。

CDB的主要内容如图 4 所示，其中

CODE/CODE_SIZE: 子句代码块指针和块长。子句的代码块不存在时，CODE=null,

TERM/TERM_SIZE: 子句项块指针和块长,

TAG: 子句索引标志，它等于子句头首变元的类型,

DOOMED: 子句删除标志,

FORE: 数据库指针，指向上一子句(CDB),

考虑到顺序索引指令（try_me_else类指令）与CDB指令相邻存放，故可以认为CDB包含一个NEXT项，指向下一子句。

2. 数据库管理元操作

下列基本操作均是以PDB和CDB为操作对象的，必要时用类C语言示意性地描述它们的操作内容。这里，PDB和CDB系指一个具体的过程和子句。

(1) 过程登记 claiming(PDB)

这一元操作系由过程调用指令 call 或 exucute 执行的。

当执行系统将控制提交给一个子目标，则要对与该子目标相对应的过程(即同名、同

变元数的过程）进行登记（PDB.CLAIMED=1）。一经登记，该过程中子句所占的空间将不能因被删除而随时回收。这样，将能够避免悬空访问。过程的登记状态将持续下去，直到发生回溯、退出该子目标时，才被恢复（PDB.CLAIM=0）。为此，要求对登记者进行跟踪，在登记者（子目标调用）失败时，进行恢复工作。一旦某个过程已被登记，后继调用者（递归调用）不需要重复登记：

```
{ if (PDB.CLAIMED=0)
    {PDB.CLAIMED=1;
     trailing(PDB);
    }
}
```

这里，利用TRAIL栈跟踪过程登记操作（trailing(PDB)）。

（2）撤消登记declaiming(PDB)

这一元操作系由TRAIL栈还原指令detrailing调用的。在发生回溯时，利用TRAIL中的过程登记信息，消除相应PDB的登记标志（PDB.CLAIMED=0）。这时，若过程中存在被删除（CDB.DOOMED=1）但其空间尚未回收的子句，即PDB.MODIFIED=1，则回收空间，并重构索引框架（reIndex_erase(PDB, CDB)）：

```
{ PDB.CLAIMED=0;
  if (PDB.MODIFIED=0) return;
  if (PDB.Nclause=0)
     {deallocate_pdb(PDB);
      PDB.PROC=null;
     }
  PDB.MODIFIED=0;
  CDB=过程中的第一个子句;
  do { if (CDB.DOOMED=1)
          {deallocate_cdb(CDB);
       reIndex_erase(PDB, CDB);
          }
       CDB=CDB.NEXT;
     } while (CDB≠null);
}
```

（3）删除子句 dooming(PDB,CDB)

若被删除子句所属的过程处于未使用状态（CLAIMED=0），则回收子句占用的空间：

```
{ if (CDB.DOOMED=0)
    { CDB.DOOMED=1);
      PDB.Nclause--;
      if (PDB.CLAIMED=0)
```

```
                { deallocate_cdb(CDB);
                  reIndex_erase(PDB,CDB);
                }
              else PDB.MODIFIED=1;
            }
        }
```

3.3 讨论

在编译型PROLOG数据库管理技术的研究方面Clocksin做了卓有成效的工作[1]。与它的方法相比，我们提出的结构和管理策略有如下一些特点。

- 从数据库的结构观点来看，子句的两种形式由子句说明块CDB联系起来，享有同一个索引框架。这样，在发生数据库操作时，所需的索引重构操作开销降低一半；
- 从管理角度来看，占用标志CLAIMED未设置在子句说明块 CDB 中，而是在高一层的PDB中，并且采取了"被占用过程中的删除子句不能同时回收空间"的策略。这就保证了过程中子句的安全性，自然地避免了子句的悬浮访问的难题，同时也使登记开销大大降低，再者，在撤销登记的时刻，一次性回收已删除了的子句空间并重构索引，对降低开销同样有利。相应地，它的不足是不能及时地回收有些可以立即回收的子句，且对回溯事件的发生过分依赖。

4. 子句的表示方法

这一问题的核心是如何在执行时刻，得到一特定子句的数据（项）形式，下面仅就迄今提出的三种实现方法的优缺点作一简单分析。

1. 反编译方法[5]。其基本思想是以子句代码为输入、动态地将子句的数据形式反编译出来。根据我们实现的经验，这一方法比较容易实现，只需要提供不多的几个特殊过程和状态，其最大优点是数据库中只需存一种形式的子句（代码）。因而管理简单，空间开销低。主要缺点是要求编译生成的代码是可逆的，限制了一些编译优化技术的采用。

2. 项代码方法。将子句作为一个数据项，再次编译，生成项代码，动态地执行子句的项代码，便可生成相应的子句项。[1]中的source方法就属于此类。优点：充分利用WAM 模型已有的执行机制和环境，速度快，不影响优化技术；缺点：存贮空间、编译时间均加倍，数据库管理复杂。

3. 源项方法，在数据库中直接保存子句的项形式。与方法 2 相比，它的不足是要求提供专门的项访问手段。然而，此方法有一个独特的优点，它支持需求驱动的动态编译。换句话说，数据库中允许仅存子句项，只有当真正需要时（子目标调用），才调用编译器，生成子句代码。在实际应用中，DBOP操作常用于往数据库中插入一些数据项，以实现传统程序设计意义下的全局量，提高效率。在这一过程中，它的代码形式是不需要的。下例中给出了实现数据空间回收的程序，其中的数据项 result(Goal)的插入和删除，不涉及其代码形式，故不需编译。

例　数据区废单元回收

```
gc(Goal): -abolish(result,1), call(Goal),
          (assert(result(Goal)), fail;
           retract(result(Goal))
          ).
```

三种方法的优、缺点总结在表1中。

表 1

	占用空间	编译时间	编译优化	按需驱动编译
反编译方法	1	1	×	×
项代码方法	2	2	V	×
源项方法	1(2)	1(2)	V	V

5. 结束语

不难看出，数据库操作的最终实现，完全依赖于自身的结构化程度。为此，YH—SIM在系统结构这一层次提供了必要的支持，本文提出了编译型PROLOG数据库的一结种构形式，以及相应的管理策略。值得指出的是，同样的系统结构支持向上又可以支持PROLOG语言的模块化扩充，从而使PROLOG增加了支持多种程序设计风格的能力。

PROLOG数据库的结构化，增加了访问的层次。这无疑会影响速度。因此，为用户提供某种可选项是必要的。具体地说，当用户说明了某一过程毋需采用DBOP谓词加以访问，PROLOG编译器（或装配器）则只生成以过程为单位的紧缀代码块。

参考文献

[1] Clocksin, W.F., Implementation techniques for prolog database,Software—Practice and Experience, vol.15(7). pp.669—675(July1985)

[2] Warren, D.H.D., An Abstract Prolog Instruction Set, TR309. (Artificial Inteuegence Center. SRI International)

[3] 张晨曦，李良良，Prolog编译器设计，国防科大TR86—6049，全国第二届逻辑程序设计会议，1986

[4] 李良良，张晨曦，顺序推理机研究中的几个重要问题，国防科大学报，第3期，1987年

[5] 李良良，慈云桂，编译型Prolog系统中的反编译算法及其实现，新一代计算机学术讨论会，杭州，1987.6

[6] 李良良，慈云桂，YH—SIM：一种支持Prolog数据库操作的顺序推理机系统结构，同上

[7] 李良良，DBOP谓词的快速实现算法，内部报告，国防科技大学研究所，1987.9

[8] 李良良，慈云桂，一种结构化的编译型Prolog数据库及其管理方法，第三届全国逻辑程序设计学术讨论会，1987年11月，无锡

A Structured Organization for the Compiler-based PROLOG Database and Its Management

Li Liangliang Ci Yungui

Abstract

In PROLOG database stored are clauses comprising PROLOG procedures and other long-term data structures represented as clauses. The implementation techniques for the manipulation of clauses are very important for PROLOG'S being a powerful AI language. It is even more the case when considering the storage of compiled clauses. With the supports of YH-SIM architecture, we propose a new structured organization for compiler-based PROLOG database, and present a simple and efficient management strategy that a procedure is the unit for claiming, and a clause the unit for dooming. Also different schemes for clause representation in the database are investigated briefly.

KEY WORDS PROLOG, Logic programming, Database, Garbage collection