第6卷第1期
1995年1月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 6, No. 1
Jan. 1995

Prolog 实现技术中的原型共享思想*

王 健 程 虎

(中国科学院软件研究所,北京 100080)

摘要 本文介绍了GOAM——一个新的Prolog语言抽象机器中的原型共享思想.采用原型共享技术可以明显地降低Prolog程序运行时的空间消耗量,在一定程度上解决了Prolog语言特有的难题——"栈溢出"问题.

关键词 Prolog,栈溢出,问题图*,原版图*,复制图*,环境桢面*.

1 Prolog 实现技术现状

1972年,法国马赛大学以Colemaurer为首的研究小组实现了第一个Prolog语言解释系统,这就是Prolog的最初形式.5年之后,D. H. Warren在英国研制了第一个编译程序.大约又是5年之后,D. H. Warren在斯坦福国际研究中心提出了著名的WAM(Warren's abstract machine),WAM取得了巨大的成功,成为Prolog语言的标准中间代码.在WAM之后,许多Prolog系统不断问世,虽然每个系统各有自己的特点,但它们基本上都是以WAM为基础,经过一些扩充而成的,对WAM的基本结构和风格没有任何改变.

一般地,Prolog语言的WAM实现方式或者类WAM实现方式都要维护3个栈:环境栈,运行栈和解除栈(trail栈).

Prolog程序执行过程可以理解为对一棵SLD(反驳—消解证明)树穷尽搜索的过程.Prolog程序的每一次合一操作就是对SLD树当前结点的操作.下面简单介绍一下WAM类Prolog系统在运行某时刻需要回溯时的执行过程,特别是其中解除栈的作用.

设当前结点为A且程序需要回溯,在回溯到结点A的前一个结点之前,必须把A结点对当前环境变量的约束解除,然而如何判断在环境栈中的哪些变量是由结点A约束的,哪些变量不是呢?WAM把这些信息保存在解除栈中.各个结点约束的变量数量是不相同的,也是不固定的,因此解除栈中每个栈元素占的内存空间既是不相同的也是不固定的,于是WAM在环境栈中A结点的位置记载两个地址,一个是结点A在trail栈中保存变量信息的起始地址,一个是结点A在trail栈中保存变量信息的终止地址.回溯时,从A结点环境栈结点中取出起始地址和终止地址,根据解除栈中两个地址之间保存的变量信息,将运行栈中的约束信息删除.上述过程形象地见图1(假设结点A将变量X和Y约束到了具

*本文1992-06-16收到,1992-12-12定稿

作者王健,1967年生,研究实习员,主要研究领域为程序设计语言,人工智能.程虎,1938年生,研究员,主要研究领域为计算机软件,语言编译,软件工程,人工智能.

本文通讯联系人:王健,北京 100080,中国科学院软件研究所

体的值).

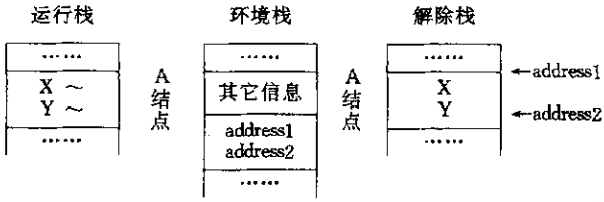


图1

综上,为了在回溯时解除无效的约束信息,必须做下面的工作:(1)维护一个解除栈,(2)在环境栈中保存两个地址信息.而且,这些信息随着 Prolog 程序的运行而动态地增长.

Prolog 程序在 SLD 树的结点生成之后,结点中保存的信息一般不能退栈,因为:

(1)Prolog 程序的过程是不确定的,某子句返回时得出了一个解,回溯希望得出下一个解,因此子句返回时运行栈内的连接信息不能抹掉.

(2)返回时,下面的运算虽不会直接引用该过程的变量,但是可能间接引用它们.

因此运行栈和环境栈中的结点信息在子句返回时一般都不出栈.这样,Prolog 的栈常常只进不出,运行时极易溢出.任何一个实用的 Prolog 系统必须缓解内存溢出的难题,本文就是为解决内存溢出问题而做的一个尝试.关于 Prolog 系统实现技术,详见文献[1—3].

2 GOAM 中的原型共享

2.1 GOAM 简介

GOAM(graph oriented abstract machine)意为基于图的抽象机器.它是不同于 WAM 的一种新的顺序 Prolog 实现和运行方式.GOAM 系统建立在三个基本概念——原版图、复制图、问题图之上,整个 Prolog 程序就是一幅问题图和一组原版图的集合.GOAM 采用的控制策略是智能回溯算法,可以说,GOAM 是面向智能回溯算法的抽象机器.当然,这不仅仅是 GOAM 所有的特点.关于 GOAM,详见文献[4].

2.2 GOAM 的运行原理

为叙述方便,首先介绍几个 GOAM 基本概念.

定义 1. GOAM 原版图结点是下列信息的集合:本结点标志,有限个形为〈x,y〉的二元组序列.二元组用来描述子体中元素的语法构成形式,其中 $x\in\{0,1,2,3,4\}$,$y\in I$,I 为整数集合,x 取 0—4 不同的值分别表示该组成元素为常数,字符串,变量,表或结构,y 表示该组成元素在相应存储区里的位置.

定义 2. 原版图是有限原版图结点和一个尾结点的线性队列.假设有一原版图 A,则其队列之首称为头结点,队列之尾称为尾结点,分别记为 HEAD(A)和 TAIL(A).设 NUM(A)代表该队列中结点的数目,那么有 $NUM(A)\geqslant 2$,即该变量集合至少有一个头结点和一个尾结点.

定义 3. 复制图结点是标志位和推理进程记录的集合.

定义 4. 复制图由一个环境帧面和复制图结点的线性队列组成.

定义 5. 问题子句的原版图叫作问题图.

定义 6. 环境帧面是父结点标志,推理号和有限个二元组〈X,Y〉的集合,每个二元组用

来表示一个子句推理某时刻变量的约束状态,其中 X 表示变量约束到的值,Y 表示生成当前值的推理步,二元组的个数与 Prolog 子句中的变量个数相等.

原版图与 Prolog 程序中的子句是一一对应的关系.给定一个程序子句,就可以为它构造出一幅原版图.同样的,由原版图可以得到程序子句.在编译的静态分析阶段,每个种类原版图的首地址,每幅原版图占用的内存空间数量,都是可以确定的.一个程序的执行由原版问题图开始作为当前图,寻找与程序指针指向的当前图的当前结点可能匹配的原版图的地址.如当前结点为问题原版图头结点,且没有候选原版图,则程序结束.如当前结点不是问题原版图头结点且候选原版图集为空,则回溯,GOAM 回溯过程是这样的:从寄存器里读出程序状态指针,根据当前程序状态找到本复制图对应的原版图,由原版图的定义可知,本结点变量集合中包含了败因分析时需要的有关变量信息,而当前各变量的约束信息保存在当前复制图头结点中,所以不必记载其它信息就可以执行智能回溯方法.如合一成功,则根据候选原版图的模式生成复制图,并调整程序指针,指向刚刚生成的复制图的结点.如果合一失败,则根据本复制图相对应的原版图及复制图中的相对地址,计算出下一幅候选图的地址,如果得出的地址是一个非法地址,说明候选图集合为空,否则继续进行合一.按照这个步骤执行下去直至结束,于是程序的执行过程可以看作是复制图的生成和作废过程.

形象地,GOAM 系统工作区内存配置图见图 2.

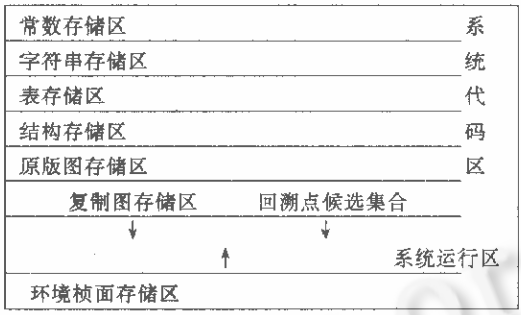


图2

2.3　GOAM 的回溯——原型共享思想

GOAM 执行至某一子目标时,根据该子目标可用的原版图模式生成一幅复制图.复制图的环境桢面里记载的是当前子目标各变量的约束状况,复制图的每一个子结点进行合一时得到的新约束都记录在环境桢面中,由环境桢面的定义可知,变量约束信息就是推理步,它相当于一个标记,记录了变量是在哪个推理步被约束的.例如,假设某子句第 i 个子体合一成功,则将子体 i 产生的新约束打上标记 i,标明该变量是由子体 i 合一时约束的(即在哪一个推理步被约束).下面例子程序中子句(1)的环境桢面示意图如下(复制图生成时约束信息是空的,随着程序的运行,变量的约束信息将陆续写入环境桢面中).

其它信息
变量 X　约束信息(标记)
变量 Y　约束信息(标记)
结束标志

下面举一个例子,说明标记的使用方法:

?-g(A,B).

(1) g(X,Y):- p(X,Y),q(Y),r(X).

(2) p(U,V):- e(U),f(V).

(3) e(a).

(4) e(b).

(5) f(g(W)).

(6) f(d).

(7) q(g(a)).

(8) q(b).

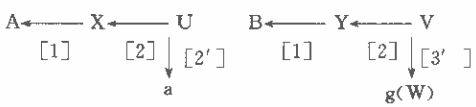
(9) r(b).

设各子目标的标号分别为:[1]→g(X,Y) [2]→p(X,Y) [3]→q(Y) [4]→r(X) [2′]→e(U) [3′]→f(V),则

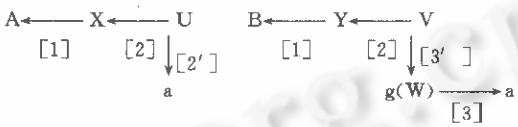
标号为[1]的子目标合一后:

标号为[2]的子目标合一后:

标号为[2′],[3′]的子目标合一后:



标号为[3]的子目标合一后:



当程序执行到某时刻需要回溯时,GOAM 执行下面的算法:

(1)从寄存器里读出程序状态指针,根据当前程序状态找到本复制图对应的原版图,从原版图结点的构成特征描述中可以获得本结点所包含的变量信息.

(2)判断是否还有其它可用的候选子句.

a. 无可用候选子句:

执行回收内存空间的指令,回收本结点所占用的内存.执行选择回溯点的算法,从败因候选集合中选择回溯点,回收回溯点和当前结点之间的结点所占用的空间并解除无效约束信息,调整程序状态指针使之指向回溯点,关于回溯算法的细节详见文献[4].

b. 有可用候选子句:

根据(1)中获得的变量信息及环境桢面中变量的标记信息,判断该变量是否是本结点约束的,如果是则撤消约束,否则判断下一个变量直至结束.

由上面的算法可知,若回溯需要本结点包含变量信息,到原版图对应的结点中去取.程序的每个子句可能被调用多次,然而不论调用多少次都到同一幅原版图中获取变量信息,这是"程序静态文本信息"的共享,故称该技术为"程序原型共享",简称原型共享.原型共享技术与 WAM 中的解除栈技术不同,WAM 中子句每被调用一次在 trail 栈中就为子句的每个

结点记录信息,调用次数越多记录的信息越多,即所占用内存空间数量是动态变化的.而原型共享技术则不然,原版图的大小在编译阶段就可以确定,且不随子句调用次数的增加而增加,是静态的,故降低了内存消耗量.

综上,为顺利地解除无效约束信息,GOAM 只需要做一项工作:对每个本结点中的变量记载一个标记信息,而标记信息就是推理步,它是基本运行系统所必需的,所以 GOAM 中的原型共享技术在不增加任何新信息的同时减少了 WAM 中的冗余信息所占的空间.

2.4　与 WAM 实现方式的比较

首先,在程序运行的基本结构上,GOAM 不要求比 WAM 多的空间.变量约束信息 GOAM 记录在环境桢面里,WAM 记录在运行栈中,推理过程信息 GOAM 记录在复制图结点的推理进程记录中,WAM 记录在环境栈里.因为记录的信息是相同的,所以需要的空间数量也是相同的.

其次,与 WAM 类 Prolog 实现方式相比,GOAM 不必维护解除栈,及环境栈中的起始和终止地址信息,运行时的空间消耗量大为减少,缓解了"栈溢出"的难题.

GOAM 是一种新的 Prolog 实现和运行方式,其内存管理方法是堆式策略,而不是 WAM 的栈式策略,所以具有潜在的并行性,关于这方面的内容详见文献[4].

3　GOAM 的软件模拟实现

在 IBM PC/XT 机上,我们用 C 语言模拟实现了一个 GOAM 原型系统.GOAM 是一种抽象机,其实现方法大致与著名抽象机 JANUS 的实现方法宏加工程序展开相同,将源程序编译成由抽象机指令集中的指令构成的集合,GOAM 的每条指令由一个子函数在目标机上实现.为了迅速地建立原型系统,分析原型设计的可靠性,GOAM 将 C 语言选作中间语言,把 Prolog 源程序翻译成等价的 C 语言程序,然后使用 C 的编译系统将得到的目标程序翻译成机器码.使用 GOAM 编制程序的过程如图 3.

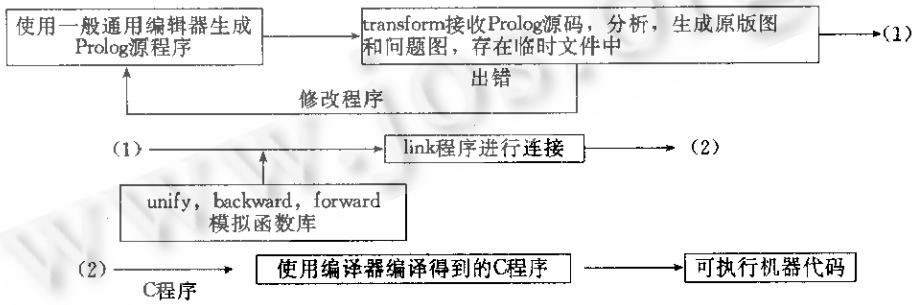


图3

解决栈溢出问题大致有两种途径:(1)减少栈中每个元素所占空间量,即本文提出的原型共享技术.(2)减少栈中元素的数量,即采用智能回溯技术.

从前面的分析可知,如不采用原型共享技术,需要为搜索空间中的每个结点保存一个变量信息集合,而 GOAM 只在原版图中保存,由子句的每次调用所共享,这样就节省了空间.下面以三个典型程序为例,对采用原型共享技术之后变量集合数的变化进行了比较.

	搜索空间变量信息集合数	GOAM 变量信息集合数	变化
query	217	9	−95.85%
map(good)	204	33	−83.82%
map(bad)	505	33	−93.46%

GOAM 缓解栈溢出问题的另外一个努力是实现了智能回溯算法，并对它做了一些试验，下面也把结果列出作为参考. 根据收集到的资料，下表给出了对程序执行时间和程序求出第一个解所需要的回溯次数的比较. 关于智能回溯算法，请见文献[5,6]. 表中的 Semi 指的是 Despain 等人的工作，具体细节详见文献 [6].

		1st solution no.	cost (s)
Semi	query	185	no
	map(good)	186	no
	map(bad)	371	no
GOAM	query	8	7.80
	map(bad)	9	7.07
	map(good)	19	15.99

从图表中可以看出，GOAM 的回溯次数大为减少.

实践表明，GOAM 既减少了运行栈单元所占的空间，又减少了程序运行中栈单元的数量，是一种效率较高的可行的 Prolog 实现方案.

参考文献

1 Maurice Bruynooghe. The memory management of Prolog implementation. In: Clark K L, Tarulund S A ed. Logic Programming, 1982:83−98.

2 Mellish C S. An alternative to structure sharing in the implementation of a Prolog interpreter. In: Clark K L, Tarulund S A ed. Logic Programming, 1982:99−144.

3 David H D Warren. Implementing Prolog−compiling predicate logic programma. D. A. I Research Report No. 39, No. 40, 1977.

4 王健. Prolog 语言抽象机器——GOAM 的设计与实现[硕士论文]. 中国科学院软件研究所，1992.

5 Vipin Kumar, Yow Jian Lin. An intelligent backtracking scheme for Prolog. Proc. of IEEE Symposium Logic Programming, 1987:406−414.

6 Jung herng Division, Alvin M Despain. Semi−intelligent backtracking of Prolog based on static data dependency analysis. Proc. of IEEE Symposium on Logic Programming, 1985:10−21.

PROTOTYPE SHARING THOUGHT IN PROLOG IMPLEMENTATION TECHNIQUES

Wang Jian　Cheng Hu

(*Institute of Software, The Chinese Academy of Sciences, Beijing* 100080)

Abstract　This paper describes the prototype sharing thought in GOAM—— a new abstract machine for Prolog. When backtracking, using prototype sharing technique can reduce the quantities of memory consumed by Prolog system efficiently, this helps to solve the particular problem—"stack overflow" in Prolog system.

Key words　Prolog, stack overflow, question graph*, proto graph*, copy graph*, enviorment frame*.