ELNF 演算的解释技术*

金 海 李智勇 张运桢 阳富民 银 海

(华中理工大学计算机系,武汉 430074)

摘要 ELNF 演算是我们在 LNF 演算的基础上扩充逻辑程序设计能力而得到的一种函数/逻辑演算系统,它构成了作者设计的函数/逻辑语言 RFUNLOG 的基础. 本文介绍 ELNF 演算的解释实现技术,包括数据结构、系统结构以及各个模块的设计思想. 最后给出了在此解释系统下,几个典型程序的运行时间.

关键词 函数程序设计,逻辑程序设计,图归约.

组合子归约技术是函数程序设计语言的一种有效实现方法[1,2],在函数程序设计语言和新一代机核心语言的研究中,受到了广泛的重视[3,4].

我们对纯函数演算系统 LNF 演算[5]进行扩充,引入逻辑函子以及逻辑归约规则得到了一种函数/逻辑演算系统 ELNF[7]. ELNF 演算可以看作为某种中间语言或机器语言,可以用软件模拟或用硬件实现. 本文介绍 ELNF 演算的软件模拟实现,即 ELNF 演算的解释系统,包括系统的设计思想、系统结构、数据结构以及各个模块的功能,并且给出了在此解释系统下,几个典型程序的运行时间.

1 系统的设计思想

ELNF 演算解释系统的设计原理是利用各函子的归约规则对 ELNF 代码进行归约,产生惰性范式作为系统的输出结果. 解释系统的核心是归约器,其余模块都是为归约器服务的. 图 1 为系统结构总图,以归约器为核心调用其它辅助模块,归约器下方为各辅助模块,上方为其使用的数据结构. 其中,函子专有归约器负责对函子独立归约规则进行处理,归约控制器负责控制调用函子专有归约器的临界条件,资源管理主要包括申请和回收空间、环境和知识库管理是为逻辑部分增加的处理过程.

整个解释系统的实现采用了图归约技术,我们设计的数据结构和有关操作有效地支持

* 本文 1991 年 5 月 13 日收到,1991 年 11 月 25 日定稿

本课题为国家"七·五"期间基础研究项目. 作者**金海**,26 岁,博士,主要研究领域为并行处理,人工智能. **李智勇**,31 岁,讲师,主要研究领域为人工智能,机器学习,专家系统,图形识别. **张运桢**,女,61 岁,教授,主要研究领域为人工智能,专家系统,计算机视觉. **阳富民**,27 岁,助教,主要研究领域为人工智能,数据库管理系统. **银海**,26 岁,助工,主要研究领域为人工智能,应用软件开发.

本文通讯联系人:金海,武汉 430074,华中理工大学计算机系

了图归约的实现，并保证了图归约的一切动作与线性演算相对应. 下面我们先介绍数据结构，然后讨论各个模块的功能.

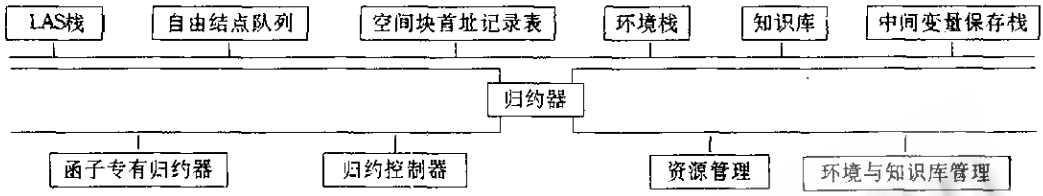
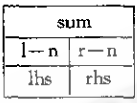


图1 系统结构总图

2 数据结构



解释系统的基本数据结构是结点和各种栈，结点的数据结构如图所示. 其中，sum 为整数，用以标记被共享的次数并作为废料收集的标志. 当 sum 大于 1 时，表明被多个父结点共享；若等于 1，表明只有一个父结点指向它；若等于 0，表明此结点为废结点，废料收集器将择机回收它. l—n (r—n)分别为左(右)部类型标志，lhs(rhs)为左(右)部数据类型对应的内容. 在 ELNF 解释系统中，定义的数据类型有 18 种.

ELNF 解释系统中的栈包括以下几类：LAS 控制栈，用于控制图归约过程，标示图归约的先后次序；自由结点栈，为可供使用的自由结点的集合，由资源管理模块调度使用；中间变量保存栈，用于暂存正在使用的中间变量，以免被回收器回收；空间块记录栈，系统以分块方式申请内存，此栈用于记录每个空间块的首址以便访问；环境栈，用于存储逻辑变量的约束值.

除了上述的结点和栈以外，系统的另一数据结构是用户函子定义表，包括以 Horn 子句方式描述的事实和规则(可看作特殊的函子)，它们以多链表的方式存放.

3 各模块介绍

上节介绍了 ELNF 解释系统中的主要数据结构，本节讨论各个模块的具体功能.

3.1 函子专有归约器

ELNF 解释系统内有一百多个函子，每个函子都有几条归约规则相对应，系统为每个函子均设计了一个专有归约器，对于有多条归约规则的函子，采用哪条规则亦由专有归约器识别和判断.

函子的归约规则包括两类，即独立归约规则和上下文归约规则. 函子专有归约器只负责独立归约规则的实现，上下文归约规则的处理由总控算法进行.

下面以 s 函子为例，说明归约规则的处理思想. s 函子的线性归约规则如下

$$S\ X\ Y\ Z \rightarrow X\ Z\ (Y\ Z)$$

相对应的归约器可用如下算法描述.

```
(Defun S－redex(　)
    {(设(redex 为 LAS－item－4)
        (X 为 LAS－arg－1)
        (Y 为 LAS－arg－2)
        (Z 为 LAS－arg－3);
     创造一个新联合 X　Z;
     创造一个新联合 Y　Z;
     用 X　Z、Y　Z 分别代替 redex 的操作符和操作数;
     退栈 3 次;
     返回(LAS－item－4)}
```

此处栈用于控制归约的过程,演算中其余函子的归约过程与此类似,不再赘述.

3.2　归约控制器

控制器是对中间代码进行循环归约的过程,其算法如下.

```
(Defun reduce(　)
  {循环
    {标准化栈;
     if (and 变量个数匹配
             变量类型匹配
             含头部归约子)
     then (进行上下文处理
           对头归约子进行归约)
     else 退出循环 }
    退栈 } )
```

下面介绍上述算法中用到的各个子过程的功能.

3.2.1　栈标准化

控制器中使用了控制栈 LAS,利用 LAS 可以方便地访问 ELNF 代码(亦称为 ELNF 合式公式)的各个参数.其中,栈底项指向公式的根结点,其余栈项依次指向公式的各层操作符结点,当栈顶项是公式的头原子时,称 LAS 是标准的.由于表示的习惯,树结构都是倒画的,即根在上,叶在下,因而栈表示也是倒长的,即底在上,顶在下,栈的增长与归约图的增长是同步的.

一个标准 LAS 示例如图2,从 LAS 栈的组织可以看出,LAS 栈实际上是用来确定将要归约的归约子,LAS 栈底项表示了整个 ELNF 公式,如果 ELNF 公式的第 i 个参数是该公式的归约上下文,则下一个栈项将用来表示这第 i 个参数,而 LAS 栈顶项是系统目前注视的焦点,图3标示了公式:

(＋ (＊ (ADD 1 3) 7)(＋ 2 2))

的进栈情况.

标准化栈算法的主要作用是将图形公式与 LAS 栈建立起对应关系以便由 LAS 栈访问图归约过程,标准化栈算法如下.

```
(Defun 标准化栈( )
  { 设 curr 为根,
    循环　{case curr
           原子:if 左部为函子
```

then curr 进栈,函子进栈,退出循环;
else curr 进栈,退出循环;
联合:curr 进栈,
curr 的操作符→curr;
向前结点:向前弧所指结点→curr;
在向前结点处进行紧缩;}
end }

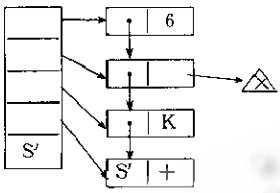


图2

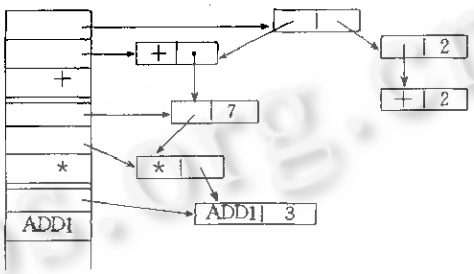


图3 (+(*(ADD1 3)7)(+2 2))的进栈情况

3.2.2 变量个数检查

变量个数检查,变量类型检查以及上下文归约规则的处理构成了归约控制的临界条件.在这三个处理过程中,各自建立了一张表,存储各函子的有关信息.

对于每个函子 f,它的变量个数(arity)是唯一的.f 的 arity 由 f 的归约规则决定.例如,若 f 的归约规则左部含 k 个变量,则 arity[f]=k,任一公式 f x_1 x_2…x_n 可归约的必要条件是 n≥arity[f].

变量个数表按 ASCII 顺序由大到小排列,表中存放所有函子与其对应变量的个数.变量个数检查模块访问该表时,将表中对应的值与实际变量个数进行比较,并返回三种结果,然后分别处理:

若:1.变量个数<arity,则退出归约控制模块;
2.变量个数=arity,直接进入控制模块中的下一子模块;
3.变量个数>arity,取前 arity 个变量进入控制模块的下一子模块.

3.2.3 变量类型检查

函子专有归约器只能处理类型与该函子所需变量类型完全匹配的情形,否则按惰性范式输出.例如,+ 1 2可归约为3,但 + a b 无法归约.因而我们说,+所对应的变量类型为整型或实型,为此我们构造了一个变量类型表,表中对函子的每一个变量类型均进行了说明.

3.2.4 归约上下文规则处理

在调用专有函子归约器前,先要将函子的各个变量中符合上下文归约规则的变量归约为惰性范式.为此,将各函子变量的上下文特性以表的形式记载存放.从表中即可知道,任一函子的某一参量是否可作上下文归约.

3.3 资源管理模块

使用结点为单位的数据结构在管理过程中存在着申请、回收两种主要操作.结点来源于自由结点队列,自由结点队列从内存中划分.当自由结点队列为空时,调用废料收集器回收

结点.

3.4 环境与知识库管理模块

逻辑函子需作一些特殊的处理,主要包括:知识库的建立与保存,合一算法的实现以及环境表的设计与保存.

知识库是一个多层链表结构,以谓词为基本单位.谓词名相同的规则和事实存放在一起,谓词名的存放先后以字典次序为准.

合一算法包含对两个合一对象均是表达式时的处理.扩展后的合一算法遇到此情况时,判断该表达式是否可归约,若然,则归约,归约后的结果再行被合一.此合一过程作为本系统的元语操作出现在逻辑函子的归约规则中.

4 系统运行情况

ELNF 演算是一个面向机器的归约演算系统,用它进行用户级程序设计是不方便的;实际上,我们已在 ELNF 演算的基础上设计了一个用户友好的函数/逻辑语言 RFUNLOG[7],并完成了从 RFUNLOG 程序到 ELNF 合式公式的编译,整个系统已在 IBM－PC 机上用 C 语言实现了[9];本文介绍的 ELNF 解释技术正是整个系统的一个重要部分,下表中列出了几个典型程序被编译后用 ELNF 解释器解释执行所需的时间.

典型程序测试　　单位:秒　PC386

程序列名	SARE
Sort(20个元素)	3
4皇后	3
整数生成器(50个元素)	6
表的反转(40个元素)	1
阶乘(12!)	<1

从表中可以看出,该系统运行的速度与一般的人工智能语言(如 LISP)相比要慢,但与[8]中采用分层 λ－抽象的实现系统相比,四皇后的运行时间要快得多.

结束语:本文给出了基于组合子归约技术的函数/逻辑语言的一种实现方案,详细介绍了系统结构、数据结构及各个模块的实现思想,并给出了一些实测数据,希望它能对函数语言的实现技术以及函数/逻辑语言的实现技术的研究提供一些帮助.

参考文献

1 Turner D A. A new implementation technique for applicative language. Software Practice and Experimence, Sept. 1979.

2 Turner D A. Combinator reduction machines. Proc. of the Int. Workshop on High－level Computer Architecture, May 1984.

3 Robinson J A. SUPER language. Syracuse University, Technical Report, No. 8801.

4 李智勇. 智能系统核心语言的研究. 计算机科学, 1988(2).

5 Greene K J. Fully lazy higher order purely functional programming language with reduction semantics. CASE Center, Syracuse University, Technical Report, No. 8503.

6 张运桢, 李智勇. LISP－ELP 的设计与实现. 计算机学报, 1989; **12**(2):157－160.

7 李智勇, 阳富民, 银海, 张运桢. 函数/逻辑程序设计语言 RFUNLOG. 计算机杂志, 1991; **19**(1):51－60.

8 廖湖声. 函数式程序的分层 λ 抽象. 计算机学报, 1989; 12(12): 892—899.
9 金海, 李智勇, 张运桢. 并行函数/逻辑程序设计语言 PRFUNLOG. 计算机工程与设计, 1992(6): 13—17.

INTERPRETER TECHNIQUE OF ELNF CALCULUS

Jin Hai, Li Zhiyong, Zhang Yunzhen, Yang Fumin and Yin Hai

(*Huazhong University of Science and Technology, Wuhan* 430074)

Abstract ELNF calculus is a functional/logic calculus system which based on LNF calculus by extending it to include logic programming ability. It is the base of the functional/logic programming language RFUNLOG. This paper introduces the interpreter implementation technique of ELNF calculus, include data structure, system structure and the design idea of each modular. At last, the run time of several typical programs under this interpreter system is also presented.

Key words Functional programming, logic programming, graph reduction.