第 38 卷 第 3 期
Vol.38 No.3

计 算 机 工 程
Computer Engineering

2012 年 2 月
February 2012

• 软件技术与数据库 •　　文章编号：1000—3428(2012)03—0037—02　　文献标识码：A　　中图分类号：TP311

一种多函数间的递归消除方法

潘 欣，石 川

(北京邮电大学智能通信软件与多媒体北京市重点实验室，北京 100876)

摘 要：为解决多函数间互相调用的递归问题，提出一种多函数间的递归消除方法。使用人工栈拆除函数间的互相调用，把递归限制在单个函数内，通过一门多锁法解决单个函数内多处出现递归的问题，研究递归消除深度对程序性能的影响。对占优树的递归消除实验表明，该方法可以解决多函数间的递归问题，且其时间效率是递归消除前的 2 倍。

关键词：递归消除；系统栈；人工栈；一门多锁

Recursion-removal Method Among Multi-function

PAN Xin, SHI Chuan

(Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia,
Beijing University of Posts and Telecommunications, Beijing 100876, China)

【Abstract】In order to solve the recursion problems involving several functions, this paper proposes a systematic recursion-removal method. It cuts the calls among different functions through artificial stack and replaces the recursions inside each function with iteration. It also researches on the relation between recursion-removal depth and the programs' time efficiency. Experiments prove that it can successfully remove the recursion of dominating tree and double the time efficiency before recursion-removal.

【Key words】recursion-removal; system stack; artificial stack; a door with several keys

DOI: 10.3969/j.issn.1000-3428.2012.03.013

1 概述

递归在程序设计中有广泛的使用，它使得代码更加简洁易懂。然而递归程序的效率却比等价的迭代程序低许多，在一些设备上无法支持高深度的递归。

针对这个问题，人们设计了许多消除的递归的方法，文献[1-2]详细描述了一些经典的递归问题和相应的解决方法。文献[3]算法具有普遍的适用性，然而过程相对繁琐。还有一些方法通过面向对象的方式进行递归消除。

以上方法都是关于单个函数自身递归的消除，然而在现实情况下，可能出现多个函数之间互相调用，且其中某些函数包含递归的复杂情况。占优树[4-5]具有高效性、简洁性等优点，在多目标进化算法等多目标问题中有广泛的应用价值。然而由于涉及多个函数间的递归调用，该数据结构有系统栈溢出的隐患。针对以上问题，本文提出了一种系统消除多个函数间递归的方法，同时讨论了递归消除深度对算法性能的影响。

2 占优树的递归问题

占优树算法主要部分的伪代码如下：

```
AT (Link root, Link newnode){
      root->count = root->count + newnode->count;
      if ( root->left-link == null )
           root->left-link = newnode;
else
   AS (root, root->left-link, newnode);
//函数间调用
}

AS (Link parent, Link child, Link newnode){
switch (Better (newnode, child))
    case 1:
          if (child->right-link == null)
              child->right-link = newnode;
else AS (parent, child->right-link, newnode);
//自身调用
    case 2:
   newnode takes the place of child;
         AT (newnode, child);//函数间调用
While() {
delete pnode from the sibling chain;
AT (newnode, pnode);//函数间调用
}
  BalanceTree(parent, newnode, L);
    case 3:
        AT (child, newnode);//函数间调用
BalanceTree (parent, child, L);
}
```

AT()中有一个判断语句，其中一个分支到达 AS()。AS()情况相对复杂，包含 3 个 CASE 语句，其中 CASE1 中含有自身调用，CASE2 和 CASE3 含有对 AT()的调用。即 AT()与 AS()组成了含 2 个函数的递归调用，同时 AS()还含有自身的递归调用。

占优树在多目标进化算法和其他多目标问题中有广泛应用价值。然而在实践中，发现在大数据集的情况下，该数据结构有发生系统栈溢出的情况，特别是在 CASE1 中，有高深度的递归。为此，必须对它进行递归消除。

基金项目：国家自然科学基金资助项目(60905025, 90924029)；国家“863”计划基金资助项目(2009AA04Z136)

作者简介：潘 欣(1989－)，男，本科生，主研方向：机器学习，数据挖掘；石 川，副教授

收稿日期：2011-08-18　**E-mail：**panyx2006@qq.com

然而在递归消除中，碰到了以下 3 个方面的困难：(1)递归过程涉及到 2 个函数的互相调用。(2)单个函数中的递归散布在 3 个 CASE 语句中，彼此之间无明显联系。(3)无明显的条件用于判断递归结束。为了解决递归对该数据结构的影响，有 2 种可能的选择：(1)仅用传统方法对 AS()中递归深度最高的 CASE1 处进行局部递归消除，但是不能彻底消除递归，即仍有系统栈溢出的隐患；(2)将算法进行彻底的递归消除。本文仅介绍彻底递归消除的系统方法。局部递归消除可使用该方法中的步骤。

3 两步综合递归消除法

一个很自然的想法是将非常规的问题归结为常规的问题，然后结合传统的方法进行解决。将该问题分为 2 步：(1)多函数间调用关系拆除；(2)函数内部递归消除。

3.1 多函数间调用关系拆除

通常拆除函数间调用的方法是将被调函数的代码拷贝到主调函数中，然而该方法不适用于多函数间递归调用问题。原因是普通函数调用时，系统会首先保存参数，调用返回时再恢复参数。占优树算法在递归情况下，AS()调用 AT()后，AT()可能重新进入 AS()，并修改 AS()中参数，调用返回后，系统将 AS()原参数恢复。若将 AT()代码拷贝进 AS()，代替函数调用，AT()的代码中需要调用 AS()，进而修改 AS()内的参数，在无递归的环境下，无法恢复调用前的参数。

针对这个问题，本文提出了人工栈法：(1)构建一个以 hold 结构(如伪代码第 1 行所示)为元素的栈，在调用前将 AS()中的参数及递归调用位置压入栈中。(2)拷贝 AT()代码至原来调用 AT()的位置。(3)在 AT()代码末尾添加对 AS()的递归调用(如伪代码第 2 行所示)。(4)在 AS()函数的末尾对人工栈进行判断，栈非空则通过栈进行参数恢复，并 goto 到递归调用位置(如伪代码第 3 行所示)。

```
struct hold{
all parameters in AS();
int   recursion_location;
};
Push  parameters  and  location  in  stack;
codes of AT();
AS();
if(stack is not empty){
   pop the parameters;
   goto the recursion_location;
}
```

通过以上方法，解决了上文提到的第 2 个困难，将递归问题限制在了单个函数中。该方法模拟了系统栈的操作过程，在实践中具有较广泛的适用性。

3.2 单个函数中的递归消除

下文讨论单个函数内部的递归消除问题。通常使用迭代的方法替换递归，然而在占优树的递归消除中，还有困难需要解决：(1)在函数的递归语句分散在 3 个 CASE 语句中，彼此无明显关联。(2)在函数执行过程中，无明显条件用于判断递归的结束。

针对以上问题，本文提出了一门多锁法，将递归的问题转换成迭代问题。具体方法如下：(1)提供一把锁，默认锁为关闭状态，确保可以至少执行一遍函数。(2)使用一个 while 门将整个函数关起来，在每次进入迭代后立即把锁打开，确保在不需要递归时可以离开函数。(3)为每一个递归调用提供一把锁，如果需要进行递归操作，则关闭锁，用迭代替换递归，再执行一次函数。结合一门多锁法和 3.1 节中的人工栈法，可以将占优树的构建算法用迭代实现。彻底消除递归后的伪代码如下：

```
AS (Link parent, Link child, Link newnode){
   Lock the door;
   While(door is locked){
      open the door;
switch (Better (newnode, child)){
      case 1:
         if (child->right-link == null)
child->right-link = newnode;
else {
update the parameters;
lock the door;
}
       break;
      case 2:
 newnode takes the place of child;
        push parameters and location in stack;
        code of AT();
        update the parameters;
        lock the door;
While() {
delete pnode from the sibling chain;
              push parameters and location in stack;
              code of AT();
              update the parameters;
              lock the door;
}
 BalanceTree(parent, newnode, L);
      break;
    case 3：
        push parameters and location in stack
        code of AT();
        update the parameters;
        lock the door;
BalanceTree (parent, child, L);
      break;

}
}
 if(stack is not empty){
      pop the parameters
      goto recursion location;
  }
}
```

4 分析与实验

本节用实验测试递归消除前(PR)，仅消除 AS 的 CASE1 处递归(RR)和彻底递归消除(TR)，在 3 种深度的递归消除后，构建占优树所消耗的时间，然后分析不同递归消除深度对算法性能的影响。

4.1 3 种递归消除深度的性能

本文使用 Windows7 操作系统，2 GHz 的 CPU，2 GB 内存，Visual C++2008 编译环境进行测试。

为了体现不同递归消除深度对占优树性能的影响，取树节点初始个数为 10 000，每次递增 10 000，树节点向量内含 7 个变量，每个变量的变化范围为 50，实验重复 20 次取平均

(下转第 42 页)