ISSN1000-9825 Journal of Software 软件学报 2000,11(8):1066~1070

基于重写技术的程序开发与验证*

孙永强[1] 陆朝俊[1] 邵志清[2]

[1](上海交通大学计算机科学与工程系 上海 200030)

[2](华东理工大学计算机系 上海 200237)

E-mail: sun-yq@cs.sjtu.edu.cn

摘要 完整地介绍了一个基于重写技术的程序开发和验证系统,重点展示验证子系统的理论、方法和技术.验证子系统使得系统能自动证明程序和规范中的优化规则及测试等式,从而进一步保证程序开发过程的正确性.验证子系统所采用的主要技术是以成批证明方法和证据测试集为特色的重写归纳方法.

关键词 函数程序设计语言,代数规范,项重写系统,定理证明,无归纳的归纳法.

中图法分类号 TP311

在文献[1]中,我们提出了基于重写技术的程序开发方法,并设计实现了一个初步的系统,该系统具有以下特色:(1) 提供了一个结合代数规范语言和函数式语言特点的混合语言;(2) 提供了约束数据类型;(3) 程序中可加入优化规则;(4) 代数规范可直接执行.以上特性都是基于项重写系统的理论和技术[2].

但是,该系统还存在不足之处.系统中引入了优化规则以描述程序的某些特定需求,这些优化规则可以改变程序的执行过程,从而改善程序的时间和空间效率.问题是,这些优化规则的正确性是由程序员保证的,即由程序员确定优化规则不会与程序中的函数定义冲突,从而不会改变程序的语义.这对程序员来说是一个负担.另一个问题是,在软件开发过程中通常无法从一开始就了解系统的所有细节,快速原型法通过多阶段的程序测试来逐步完善原型系统.如果我们能在设计和实现的过程当中而不是在程序编成之后来构造测试实例,那么我们就能更早地发现错误,从而提高开发效率.

本文将给出解决这些问题的方法.这里描述的系统是文献[1]中系统的扩充,主要是集成了验证子系统.这个验证系统可以证明优化规则是否与程序规范和函数定义协调,从而去除了程序员的负担.同时,可以在开发过程中测试程序,以便尽早发现错误.

本文使用的是标准的重写系统术语,可参见文献[2,3]等.

1 验证子系统的功能

我们的系统提供了一个混合语言,它结合了一个增强的函数式语言与一个代数规范语言的特色.代数规范语言使我们能够在较高的抽象级别上开发程序,从而使正确性更容易得到验证.同时,规范可以通过项重写直接实现,这意味着系统支持快速原型开发方法.增强的函数式语言提供了函数式程序设计的全部表达能力和优越性[4].更重要的特性是,由规范和函数定义组成的混合程序可以在系统中执行,这使我们能够得到正确而高效的程序.总之,我们的系统不仅支持高级的、更灵活的程序开发,还提高了程序的效率,这在纯代数规范和纯函数语言中是很难做到的.上述特色通过使用优化规则来实现.

* 本文研究得到国家"九五"重点科技攻关项目基金(No.96-729-01-06)资助.作者孙永强,1931年生,教授,博士生导师,主要研究领域为计算机软件.陆朝俊,1964年生,博士,副教授,主要研究领域为重写系统,数据库技术,数据仓库技术,数据挖掘.邵志清,1966年生,博士,教授,主要研究领域为逻辑,重写系统,并行理论,智能系统.

本文通讯联系人:孙永强,上海 200030,上海交通大学计算机科学与工程系

本文 2000-01-17 收到原稿;2000-05-12 收到修改稿

1.1 优化规则的验证

本系统的函数式程序设计语言是增强的,这是指它提供了定义优化规则的机制.优化规则是描述被定义函数之间各种关系的计算规则.程序员可以根据实际应用中的特定知识,利用这个机制来定义优化规则,从而产生满足某种特定要求的计算过程,如效率、空间开销及并行性等方面的考虑.具体例子可参见文献[1].

作为程序中的一个相对独立的部分,优化规则不应改变函数程序主体的指称语义.如果将函数定义看作是一个理论,则优化规则应当是该理论的逻辑后果或定理.在本系统的早期版本中,这个事实不能由系统建立,即证明优化规则与原程序的协调性是程序员的责任.为了解决这个问题,我们在原系统中集成了一个验证子系统.

验证子系统的主体是一个基于重写归纳技术的定理证明器[9],它将函数式程序或代数规范视为等式理论,而优化规则视为命题.为证明优化规则与程序是协调的,只需证明它是等式理论的归纳定理即可.如果证明过程终止并给出肯定的结果,那么优化规则就可以应用,因为它没有改变程序的语义,所以整个混合程序的正确性就得到了保证.但是需要说明的是,如果证明过程无法给出肯定的结果,我们就不能推出优化规则绝对与程序冲突的结论.这时,只好由程序员来决定优化规则是否正确.下面的例子显示了本系统的能力.

例1:这是关于二叉树的程序.Datatype 定义了自然数数据类型 Nat 和二叉树类型 Tree;Fun 用于定义函数;函数 insert 向二叉树中插入元素;count 计算二叉树中元素的个数;Optimal 用于声明优化规则.

```
Datatype Nat=O|S of Nat;
Datatype Tree=Leaf|Branch of Nat * Tree * Tree;
Fun add(O,y)=y
and add(S(x),y)=S(add(x,y));
Fun insert(a,Leaf)=Branch(a,Leaf,Leaf)
and insert(a,Branch(b,l,r))=if (a<b) Branch(b,l,insert(a,r)) Branch(b,insert(a,l),r;
Fun count(Leaf)=O
and count(Branch(a,l,r))=S(add(count(l),count(r)));
Optimal;
count(insert(a,t))=S(count(t));
```

定理证明器将证明规则 count(insert(a,t))=S(count(t))是一条归纳定理.如果加入 count(insert(a,t))=count(t),验证系统将发现这不是定理.

1.2 测试等式的验证

我们的系统可以在设计和实现过程中测试程序或规范,而不是在整个系统完全实现之后才能测试.这是因为我们可以在程序或规范中加入测试等式.测试等式由程序或/和规范中的数据构造子、函数符号、算子及变量组成,测试等式可以是系统的定理,也可以不是(程序员知道),将测试等式交给验证子系统来确定其真假.如果证明结果与程序员知道的情况不一致,说明程序或规范有问题,程序员据此可以检查并改进程序和规范.更有价值的是,测试过程可以帮助程序员定位程序中的错误.如果某个已知不是定理的测试等式被证明是定理(或相反),那么经常能在该测试等式所依赖的函数定义或规范公理中找到错误.

例2:本程序包含测试等式.Test 用于给出测试等式;append 是合并两个表的函数.

```
Fun append NIL ys=ys
and append x::xs ys=x::(append xs ys);
Test;
append (append x y)z=append x (append y z); //associativity
append x y= append y x;                     //commutativity
```

在这个例子中,定理证明器将证明第1条测试等式确实是定理,而第2条不是.这正是我们所预期的.如果我们将函数 append 的第2条定义式写成 append x::xs ys=x::(append ys xs),则系统将发现两个测试等式都不是定理.这与我们所具备的常识不符,从而提醒我们程序中有错误,这样就做到了在设计和实现过程当中进行测试.

2 验证子系统的理论基础

文献[1]中的系统是基于重写技术的.为了与原系统保持一致,我们采用了同样基于等式和重写技术的定理

证明方法.本节将给出与验证子系统有关的主要理论,详见文献[3].

2.1 隐式归纳证明方法

隐式归纳是基于 Knuth-Bendix 完备化算法的归纳证明方法.具体地说,将归纳证明问题化为待证等式与一个等式理论(或与其等价的重写系统)的协调性问题.这种协调性证明方法(又称无归纳的归纳法)的基础工作是由 Kapur 和 Musser[6]首先提出来的.

定理 1. 等式 $s=t$ 是等式理论 E 的归纳定理,当且仅当 $s=t$ 在 E 的所有归纳模型中为真.

这条定理意味着可以通过检查 E 与 $s=t$ 的协调性(将 $s=t$ 加入 E 不会产生新的由基项组成的等式)来证明 $s=t$.根据下面的定理[3],这种协调性可以通过检查任意基项的可归约性在 R_1 和 R_2 中是否保持相同而得到验证,其中 R_1 和 R_2 分别是用 KB 完备化算法从 E 和 $E\cup\{s=t\}$ 得到的完备的项重写系统.

定理 2. 令 R_1 和 R_2 是完备的项重写系统,其中 R_2 由 $R\cup\{l=r\}$ 生成,则 $l=r$ 是 R_1 的归纳定理当且仅当 $IRG(R_1)=IRG(R_2)$,这里,$IRG(R)$ 表示相对于 R 不可归约的全体基项的集合.

2.2 归纳定理的成批证明方法

在上述结果的基础上,我们采用一种成批证明技术来证明归纳定理.迄今为止,重写归纳方法都是一次证明一条等式的,但我们经常需要一次检查多条等式.例如,优化规则正确性的证明如果能够一次性进行的话,则效率会很高.除了效率高的优点,成批证明经常可以避免单条等式证明方式中的发散性.

例 3:考虑整数加法的重写系统,标记为 $\sum=\{O,succ,pred,+\}$:

O:int
succ:(int→int)
pred:(int→int)
+:(int,int→int)
$O+y\to y$
$succ(x)+y\to x+succ(y)$
$pred(x)+y\to x+pred(y)$
$succ(pred(x))\to x$
$pred(succ(x))\to x$

利用成批证明方法我们可以证明等式:

$$x+O=x,$$
$$x+succ(y)=succ(x+y),$$
$$x+pred(y)=pred(x+y)$$

都是归纳定理.但如果采用逐条证明的方法,$x+O=x$ 将无法首先得到证明,因为它在完备化过程中会产生无穷多条规则.

有关成批证明方法的基本定理是定理 2 的推广[3].

定理 3. 令 R_1 和 R_2 是完备的项重写系统,其中 R_2 由 $R_1\cup\{l_1=r_1,\ldots,l_n=r_n\}$ 经 KB 完备化过程生成,则 $l_1=r_1,\ldots,l_n=r_n$ 是 R_1 的归纳定理当且仅当 $IRG(R_1)=IRG(R_2)$.

2.3 证据测试集

无论是采用逐条还是成批证明方法,协调性问题都化简为检查 $IRG(R_1)=IRG(R_2)$ 的问题.对此有许多方法,其中标准测试集方法及其若干变种被广泛采用.我们提出了一种新技术,称为证据测试集方法.证据和证据测试集等概念涉及许多复杂细节[7],本文不作详述.

一个证据测试集必定包含于一个标准测试集之中,这意味着前者的规模不会大于后者.另一方面,证据测试集的构造过程不需要拟归约性判别算法.总之,采用证据测试集效率更高,对归纳定理证明更具一般性.

定理 4. 令 R_1 是由 E 生成的完备的重写系统,$s=t$ 为一个等式,R_2 是由 $E\cup\{s=t\}$ 生成的完备的重写系统,则 $s=t$ 是 E 的归纳定理当且仅当 R_2 中每条规则的左端相对于 R_1 是拟可归约的.

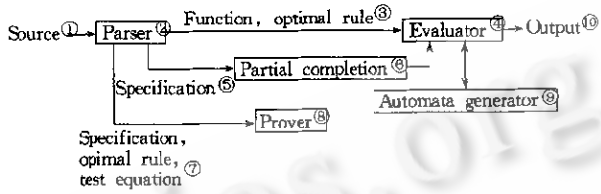
这样,我们只需计算原重写系统的测试集和新的完备重写系统,然后用测试集统一检查新系统中所有新规则的左端的拟可归约性即可.

3 集成系统的实现

目前的系统是在系统[1]中集成了验证子系统之后形成的.本节概述整个集成的程序开发和验证系统,并着重于验证子系统的实现.

3.1 系统模块

程序开发与验证系统的主要模块及其相互间的关系如图1所示.本节我们简要描述其他模块的功能,验证系统的描述将在后面给出.



①源,②语法分析器,③函数,优化规则,④求值器,⑤规范,⑥部分完备化,⑦规范,优化规则,测试等式,⑧证明器,⑨自动机生成器,⑩输出.

Fig. 1 System modules
图1 系统模块

各模块的主要功能描述如下:

(1) 语法分析器对输入程序进行语法分析,抽取出类型定义、函数定义、代数规范、优化规则和待计算的目标表达式.本模块还包含类型检查和类型推导部分.

(2) 部分完备化过程将代数规范转换成等价的合流的项重写系统.所谓"部分"是指并不要求所得系统是终止的.

(3) 自动机生成器对重写系统进行处理,以便生成平行最外模式匹配自动机.这将加速寻找归约表达式的过程,从而提高系统的效率.

(4) 求值器完成目标表达式的归约.归约策略基本上是平行最外策略,如果有优化规则或必要归约表达式可以应用,则它们将被优先使用.

3.2 验证子系统

验证子系统的输入是由规范和待证等式及项集上的序关系组成.如果目标等式确实是规范的定理,系统将能证明这一事实;如果某些目标表达式不是定理,系统将识别并删除那些可能不成立的等式,以便确保剩下的等式是真正的定理.

定理证明器所采用的主要技术是成批证明方法和证据测试集算法.这两个技术以及其他一些技术使验证子系统具有许多优点:

(1) 能够同时证明一批等式而不是逐条证明,故系统的效率大大改善了.

(2) 在许多逐条证明方法会发散的情况下,成批证明方法可以避免发散性.这是因为同时被证明的多条等式可以互为引理,这样就增强了系统的证明能力.

(3) 可以从目标等式集合中识别出那些可能不是归纳定理的等式,从而确保剩下的等式是归纳定理.当然,这样可能会将未能识别的定理当作非定理剔除,但这种情况很少发生.

(4) 证据测试集方法大大改善了系统的效率.实验表明,效率是成数量级地提高了.

验证子系统的详细情况可参见文献[3,8].

4 相关研究工作

在程序开发中应用重写技术是一件很有意义的工作,因为从中可以发现在实际应用中所需进一步解决的一些有关理论问题.国际上已有这样的系统,例如:Maude系统(美国SRI,Jose Meseguer领导开发)、CiME系统

(法国,Claude Marche 领导开发)以及由 Adel Bouboula 和 Michael Rusinowitch 等人领导开发的系统.

在功能方面,本系统可以将代数规范与函数程序结合起来,这意味着我们的系统能够在某种程度上支持软件生命周期中早期阶段开发的工作,特别是从规范过渡到程序的开发,而前述诸系统还不能做到.

在系统中加入归纳推理并利用重写技术求证并不是新概念,但是利用归纳推理以提高所开发的程序的计算效率应是本系统的一大特色,在其他系统中未发现类似工作.

在证明技术方面,本系统的贡献在于提出了证据测试集方法,在规模大小上证据测试集一定不比标准测试集[9]大,在很多情况下要小得多.一项有意义的工作是如何将这些思想拓展到其他测试集定义中去,例如,由 Adel Bouboula 和 Michael Rusiowitch 等人领导开发的系统是在条件等式理论中用了一种复杂的测试集方法.与此有关的是 Comon 和 Jacquemard(LNCS 97)曾给出了在一般情况下基于可归约的准确复杂度.这些工作将有助于定义一般情况下的优化测试集.

在合流性方面,一般地,为了证明合流性,重写系统首先要求系统是终止的,例如,在 CiME 系统中,采用了一种先进的证明终止的方法.由于我们的函数语言采用的是惰性语义,因此并不要求系统必须是终止的.我们在部分完备化的研究中倾注了较大力量以解决这一假定所带来的一些问题.

参考文献

1 Sun Yong-qiang, Lin Kai, Shen Li. The design and implementation of a program development system based on rewriting method. ACM SIGPLAN Notices, 1997,32(2):27~34

2 Klop J W. Term rewriting systems. In: Abramsky S, Gabbay D M, Maibaum T S E eds. Handbook of Logic in Computer Science, Vol. 2. Oxford: Oxford University Press, 1992. 1~117

3 Shao Zhi-qing. Rewriting induction techniques [Ph. D. Thesis]. Shanghai Jiaotong University, 1998
(邵志清.重写归纳技术[博士学位论文].上海交通大学,1998)

4 Peyton Jones S L. The Implementation of Functional Programming Languages. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987

5 Kapur D, Musser D R. Proof by consistency. Artificial Intelligence, 1987,31(2):125~157

6 Dershowitz N. Applications of the Knuth-Bendix completion procedure. Aerospace Report, ATR-83(8478)-2, Laboratory of Operation, 1983

7 Shao Zhi-qing, Sun Yong-qiang *et al*. Proving inductive theorems using witnessed test sets. In: Staples J, Hinchey M G, Liu S eds. Proceedings of the 2nd International Conference on Formal Engineering Methods. Los Alamitors, CA, 1998. 158~164

8 Xie Gao-hui. Studies on completion of rewrite systems and implementation of a software verification system [MS Thesis]. Shanghai Jiaotong University, 1998
(谢高辉.重写系统完备化的研究及软件验证系统的实现[硕士学位论文].上海交通大学,1998)

9 Kapur D, Narendran P, Zhang H. Automating inductionless induction using test sets. Journal of Symbolic Computation, 1991,11(1/2):83~111

Program Development and Verification Based on Rewriting Techniques

SUN Yong-qiang LU Chao-jun[1] SHAO Zhi-qing[2]

[1](*Department of Computer Science and Engineering Shanghai Jiaotong University Shanghai* 200030)
[2](*Department of Computer Science East China University of Science and Technology Shanghai* 200237)

Abstract In this paper, the authors present a complete introduction of a program development and verification system based on rewriting techniques, focusing on the theory, methods and techniques of the verification subsystem. The verification subsystem enables the system to prove the correctness of the optimization rules and test equations in programs and specifications, hence the soundness of the program development process is further guaranteed. The main technique employed in the verification subsystem is rewriting induction featuring batch proof method and witnessed test sets.

Key words Functional programming language, algebraic specification, term rewriting system, theorem proving, inductionless induction.