MFS:一个基于重写技术的程序开发系统*

杨继锋[1] 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科学与工程系 上海 200237)

摘要 MFS是一个基于重写技术的程序开发系统.它提供的程序设计语言MFSL是扩展的函数式语言与代数规约语言相结合的混合语言.在这种混合语言中引入了能够提高效率和满足用户特殊需求的优化规则定义机制.语言的类型系统以及在语言和系统中引入的证明和测试机制能够使人们在开发过程中较早地发现问题,提高所开发系统的正确性.在系统的实现中采用的必要平行最外归约策略、围归约、证据测试集等技术能够使所开发的系统具有很高的实现效率.应用这一程序开发系统,能够较快地开发出正确且效率较高的程序.

关键词 函数式语言,代数规约语言,混合语言,类型系统,测试.

中图法分类号 TP311

高效地开发出正确的、易维护的程序,是计算机软件研究的重要课题之一.造成程序开发困难的原因,一方面是由于程序设计本身的困难,另一方面在于完整的、精确的程序规约难以确定(① 程序规约设计的困难;② 程序员对所开发系统的认识需要一个由模糊到清晰的过程).程序规约和程序是软件的两个不同层次的描述.将可执行规约和程序设计语言有机地结合在一起的系统是一种很好的程序开发系统[1,2],这种系统具有如下的优点:(1) 可以帮助设计完整的规约;(2) 可以给出更为高效的编译代码.

在我们开发的基于重写技术的程序开发系统MFS中,其提供的程序设计语言MFSL就是将扩展的函数式语言和代数规约语言结合在一起的一种混合式程序设计语言,它具有上面提到的优点.MFSL还具有一个静态类型和动态类型相结合的类型系统,其静态类型系统又是一个将类型推导和类型检查相结合的类型系统.在MFS和MFSL中同时引入了证明和测试机制.

在以函数式语言和代数规约语言为基础的MFSL中,约束类型和优化规则定义机制的引入,大大增强了语言使用的灵活性,给程序设计带来了方便.类型系统以及证明和测试机制的引入,一方面方便了程序规约和程序的设计,能够帮助程序员澄清对所开发系统的认识,有利于缩短系统的开发周期;另一方面还能够增加所开发系统的正确性.

1 程序设计语言MFSL

MFSL首先对函数式语言进行扩展,引入约束类型定义机制和函数的类型声明机制,然后再把扩展的函数式语言和代数规约语言相结合,最后,在语言中引入优化规则定义机制和测试说明机制.

1.1 扩展的函数式语言

有关函数式语言的介绍见文献[3,4].函数式语言具有很多优点,但在使用中仍有一些不足之处,例如,对于

* 本文研究得到国家"九五"科技攻关项目基金资助.
作者杨继锋,1971年生,博士,主要研究领域为程序设计语言,重写系统.孙永强,1931年生,教授,博士生导师,主要研究领域为计算机科学理论,程序设计语言,并行处理.陆朝俊,1964年生,博士,副教授,主要研究领域为新型程序设计语言,重写系统.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:杨继锋,上海 200030,上海交通大学计算机科学与工程系
本文1998-06-02收到原稿,1998-09-02收到修改稿

有限长度队列、平衡树等数据类型,用现有的函数式语言来处理,是极其繁琐且容易出错的.因此,在MFSL中引入了约束类型定义机制.约束类型是由存在特定关系的构造子所生成的数据集合.约束类型通过在通常的类型定义基础上加入约束条件而得到:约束条件分为布尔约束和计算约束.布尔约束要求数据满足一定的条件,当不满足时,就认为不是该数据类型;计算约束也要求数据满足一定的条件,但它是通过"当满足某一条件 P_i 时,进行某一操作 $OP_i, i=1,2,\ldots,n$"的方式实现的,此时,它对数据的约束条件就是 $\neg(P_1 \vee P_2 \vee \ldots \vee P_n)$.

例1:利用布尔约束和计算约束,可以实现对组成元素类型相同的聚集类型(aggregate)中的特殊元素进行的自动检测和处理.下面两例都定义了不含具有性质 $P(x)$ 元素的 ty 类型的二叉树.(A)例检测二叉树中是否包含具有性质 $P(x)$ 的元素;(B)例自动滤除二叉树中具有性质 $P(x)$ 的元素,其中 $merge(x,y)$ 是二叉树的合并操作.

(A) datatype $d\_tree = dTIP \mid dFORK$ of $ty * d\_tree * d\_tree$

　　by bool Not ($P(a)$) for $dFORK(a,l,r)$

(B) datatype $f\_tree = fTIP \mid fFORK$ of $ty * f\_tree * f\_tree$

　　by computation $fFORK(a,l,r) = merge(l,r)$ when $P(a)$

1.2 代数规约

在MFSL中,程序员可以用代数规约描述程序的功能规约.系统能够把代数规约自动地转换为可执行的程序.代数规约的形式如下:

specification spec-name

　sort $s_1, s_2, \ldots, s_n$

　op 〈操作类型说明〉and 〈操作类型说明〉and...and 〈操作类型说明〉

　eq 〈等式〉and 〈等式〉and...and 〈等式〉

　optimal 〈等式〉and 〈等式〉and...and 〈等式〉

　test 〈等式〉and 〈等式〉and...and 〈等式〉

end

其中"optimal"后面为一组优化规则(见第1.3节);"test"后面为一组测试等式(见第3节).

1.3 优化规则

优化规则能够影响计算过程.因此,程序员可通过定义优化规则来提高程序的效率,使得程序满足某些特殊要求.优化规则所定义的操作之间的关系可以是函数之间的关系,也可以是代数规约中操作之间的关系,还可以是函数和代数规约中操作之间的关系.

优化规则应是定理.在MFS中,系统会对优化规则进行自动证明(见第3节).

1.4 混合语言

在MFSL中,程序员可以用扩展的函数式语言和代数规约语言进行混合程序设计,并根据需要使用优化规则和测试等式(见第3节).这里的混合设计有两方面的含义:(1) 程序中一些功能模块可以使用代数规约语言,而另一些功能模块可以使用扩展的函数式语言;(2) 同一功能模块的代数规约说明和扩展的函数式语言实现可以同时出现在程序中,这时不但能够生成正确的程序,具有较高的效率,而且代数规约说明也成为程序中的文档,有利于系统的维护.

2 类　型

MFSL具有类型系统.类型系统能够给程序开发带来很大的好处[3,5].与通常的函数式语言不同,在MFSL中,还引入了函数的类型声明机制.对于函数,程序员可以声明其类型,也可以不声明其类型,还可以暂时只声明其类型而不进行定义(这时仍可生成可执行系统,只是这时不对该函数进行计算处理,在计算结果中可能会含有该函数符号).

在MFSL中引入类型声明机制是出于下面的考虑:(1) 程序员可以在类型这一层次上考虑程序规约和程序

的设计,而暂时不考虑具体的处理细节.这时,问题会变得简单,不易出错.(2) 程序员可以利用类型声明来对函数施加一定的约束.函数的类型声明能够配合函数定义本身,使得函数满足程序员的某种要求.在 MFSL 中,程序员可以把函数的类型声明处理成一个比从函数定义本身推导出的类型更加具体的类型,这时 MFSL 把该函数的类型处理为程序员定义的类型,从而使该函数不会出现在程序员不希望它出现的地方.

MFSL 的类型系统分成静态类型和动态类型两部分.动态类型部分是对具有布尔约束的数据类型进行布尔约束检查,它是在运行时进行的.静态类型部分是在编译时进行的,它是一个类型推导与类型检查相结合的类型系统:若程序员对函数无类型声明,系统能够自动推导出类型;若程序员对函数 f 进行了类型声明,则对其进行检查,设声明的类型为 δ,推导出的类型为 τ,那么,若 δ 是 τ 的实例,则 f 的类型为 δ.若 τ 是 δ 的实例,则 f 的类型是 τ.

例2:对于下面的程序:

datatype $'a\ list = NIL \mid CONS\ of\ 'a * 'a\ list$;

fun $append\ NIL\ xs = xs$

and $append\ (CONS(x,xs))\ ys = CONS(x,(append\ xs\ ys))$;

type $string\_cat: alphabet\ list -> alphabet\ list -> alphabet\ list$;

fun $string\_cat\ x\ y = append\ x\ y$;

函数 $append$ 的类型为 $'a\ list -> 'a\ list -> 'a\ list$,函数 $string\_cat$ 的类型为程序中声明的类型.

3 证明和测试

优化规则给程序设计带来了好处,但是它的引入不应改变程序的指称语义.这就要求它是已定义规约(函数或者/和代数规约)的定理.MFS 中提供了定理自动证明机制,可以对优化规则进行一定的证明,从而可以减轻程序员的负担,提高程序的正确性.

例3:MFS 能够自动证明此例中的优化规则是归纳定理.

fun $add(O,y) = y$

and $add(S(x),y) = S(add(x,y))$;

fun $count\ NIL = O$

and $count\ (CONS(x,xs)) = S(count\ xs)$;

fun $append\ NIL\ xs = xs$

and $append\ (CONS(x,xs))\ ys = CONS(x,(append\ xs\ ys))$;

optimal $count\ (append\ x\ y) = add(count(x),count(y))$

MFSL 中还引入了测试说明机制.在 MFSL 的代数规约或者整个程序中,可以通过给出一些有关的函数符号或者(和)操作符号的等式(称之为测试等式),来对代数规约或者整个程序进行测试.这些等式可以是定理,也可以不是定理.系统会自动地对它们进行一定的判断.由此可以检验代数规约或者整个程序的正确性以及程序员对系统的认识是否正确.

测试机制的引入首先为程序员提供了一种测试程序正确性的手段.程序员可以利用某些特殊知识或者直觉上的知识,测试规约和程序是否满足这些性质(等式),从而为程序员提供了利用操作之间的相互关系这一方式来测试程序正确性的手段;其次,在程序中引入测试等式,便于更早地发现问题,通过对测试等式的证明,在编译阶段就能够发现问题;再次,测试等式有利于程序员确定问题的出处,澄清程序员对系统的模糊认识,如果某条测试等式应该是定理而不是定理,或者相反,那么,程序中的问题就出现在与该条测试等式相关的函数定义或者代数规约的公理中(参见相关式集的定义).由此,测试机制的引入能够加速程序的开发,缩短开发周期,提高程序的正确性.

MFS 中含有一个定理证明器,利用它,可以对待证等式进行一定的证明.定理证明器采用了文献[6]中提出的基于证据测试集的无归纳的归纳证明方法和定理成批证明等技术,它的特点是,具有很高的证明效率和很强的证明能力.有关它的详细介绍,请参见文献[6].

在MFS系统中,对优化规则和测试等式的处理方法如下:(1) 对代数规约中的测试等式和优化规则,把该代数规约的公理作为等式公理,进行证明;(2) 对整个程序中的优化规则和测试等式,把程序中与优化规则和测试等式相关的函数定义式和代数规约中的公理组成的等式集(称之为相关式集,下面将给出它的定义)作为等式公理,对它们进行证明.

在定义相关式集前,需要先给出表达式 t 和等式 eq 中函数符号和操作符号组成的集合 $F(t)$ 和 $F(eq)$ 的定义.

定义1. (1) 设 $\alpha(t)(\omega)$ 表示表达式 t 的出现 ω 处的符号,$occ(t)$ 表示表达式 t 的出现集,则 $F(t)=\{\alpha(t)(\omega)\mid \alpha(t)(\omega)$ 是程序中定义的函数符号或者代数规约中的操作符号,$\omega\in occ(t)\}$;

(2) 设等式 eq 为 $l=r$,则 $F(eq)=F(l)\cup F(r)$.

下面给出等式 eq 的相关式集 $RSet(eq)$ 的定义,同时它也是计算相关式集的算法.

定义2. (1) 对于定义函数 f 的函数定义式 df,若 $f\in F(eq)$,则 $df\in RSet(eq)$;

(2) 对于代数规约 S 中的等式 eq',若 $F(eq)\cap F(eq')\neq\varnothing$,则对代数规约 S 中的每条公理 eq_s,均有 $eq_s\in RSet(eq)$;

(3) 对于 $RSet(eq)$ 中的任一等式 eq' 和定义函数 f 的函数定义式 df,若 $f\in F(eq')$,则 $df\in RSet(eq)$;

(4) 对于 $RSet(eq)$ 中的任一等式 eq' 和代数规约 S 中的等式 eq'',若 $F(eq')\cap F(eq'')\neq\varnothing$,则对代数规约 S 中的每条公理 eq_s,均有 $eq_s\in RSet(eq)$;

(5) 相关式集仅由以上规则定义出.

对于等式集 S,定义 S 的相关式集 $RSet(S)=\bigcup_{s\in S}RSet(s)$.

4　系统的实现

MFSL的操作语义由重写系统给出,函数定义、代数规约、优化规则以及计算约束,都转换为重写规则.由代数规约向重写系统转换以及定理证明等理论都源于有关重写的理论[2,6].有关重写系统的介绍请参见文献[7].

MFS的实现基于重写系统.所有重写规则构成一个重写系统,由它对计算目标进行归约.归约中使用了平行最外模式匹配[8]、图归约[9]等技术来提高归约效率,定理证明中使用了证据测试集、定理成批证明[6]等技术来增强证明能力和提高证明效率.MFS的实现包括语法分析、类型推导和检查、准完备化、自动机生成、求值器和定理证明器等模块(如图1所示).

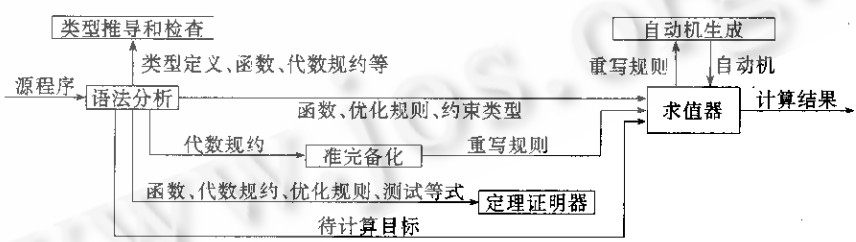

图1　MFS系统的实现结构

各模块有以下几个主要功能.

(1) 语法分析.对输入的程序分析出类型定义、函数类型声明、函数定义、代数规约、优化规则、测试等式和待计算目标等,把这些语言构造成分分别输送给相关模块.

(2) 类型推导和检查.对程序进行静态类型推导和类型检查.

(3) 准完备化.将代数规约转化成等价的、合流的重写系统[2].

(4) 自动机生成.对重写规则进行预处理,生成平行最外模式匹配自动机.归约中使用此自动机进行模式匹配.

(5) 求值器.用重写系统对待计算目标进行归约.采用必要平行最外辅以优化规则优先的策略:在平行最外可归约表达式中,若有优化规则,则对其进行归约;否则,若有必要可归约表达式,则对其进行归约,否则归约所

有平行最外可归约表达式.

(6) 定理证明器.它输入函数和代数规约的定义以及优化规则和测试等式,对优化规则和测试等式进行证明,给出证明结果.

5 结 论

MFSL 对函数式语言和代数规约语言进行了有机的结合,使得程序员能够在同一个程序中用代数规约语言和函数式语言进行混合程序设计.

约束类型和优化规则定义机制的引入,大大增强了程序设计的灵活性.约束类型使得程序员可以定义特定于问题的数据类型,并同时摆脱了在程序中不断检查数据合法性并在需要时进行一定处理的这一繁琐且容易出错的工作.优化规则的定义机制使得程序员可以在较高层上对程序的运行过程进行控制,显然,这在某种程度上增强了语言的描述能力.

类型系统以及证明和测试机制的引入,一方面,可以提高程序的正确性;另一方面,使得在目标系统生成的过程中,对程序进行一定的测试和检查,而不是等到目标系统生成后,完全通过测试用例来发现问题,这样,一方面可以更早地发现问题,另一方面也有助于程序员发现问题的来源,有利于程序员澄清对所开发系统的模糊认识,由此给程序开发带来了方便,从而有利于缩短软件的开发周期.

在 MFSL 的设计中,考虑了软件开发的过程,融入了原型速成的方法论思想,我们建议,在使用 MFSL 进行一般的软件开发时,采用如下的过程:

(1) 使用代数规约和常用的函数以及已有的程序进行设计,同时使用类型系统和测试机制;

(2) 待得到完善的、正确的代数规约后,用函数实现代数规约,以提高实现效率.这时,可以把代数规约中的公理作为测试等式进行测试,检验函数实现的正确性.此时,可以把代数规约作为程序内注释文档,也可以把它作为程序的组成部分(这时系统可能产生更为高效的编译代码);

(3) 加入优化规则来优化程序和实现某些特殊需求.

参考文献

1 林凯,孙永强,陆朝俊.基于重写方法的程序开发系统的设计和实现.计算机学报,1996,19(9):641～648
(Lin Kai, Sun Yong-qiang, Lu Chao-jun. The design and implementation of a program development system based on rewriting method. Chinese Journal of Computers, 1996,19(9):641～648)

2 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

3 Simon L P J. The Implementation of Function Programming Languages. Englewood Cliffs: Prentice-Hall Inc., 1987

4 Milner R, Tofte M, Harper R. The Definition of Standard ML. London: MIT Press, 1990

5 Milner R A. A theory of type polymorphism in programming. Journal of Computer and System Science, 1978,17(3):348～375

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

7 陆朝俊.重写系统研究[博士学位论文].上海交通大学,1994
(Lu Chao-jun. Studies on term rewriting systems [Ph. D. Thesis]. Shanghai Jiaotong University, 1994)

8 沈理,林凯,孙永强.平行最外模式匹配.软件学报,1996,7(增刊):329～337
(Shen Li, Lin Kai, Sun Yong-qiang. Parallel-outmost pattern matching. Journal of Software, 1996,7(supplement):329～337)

9 杨继锋,孙永强.项重写的图实现.计算机工程,1998,24(4):3～7
(Yang Ji-feng, Sun Yong-qiang. Term rewriting implemented by graph rewriting. Computer Engineering, 1998,24(4):3～7)

MFS: a Program Development System Based on Rewriting Method

YANG Ji-feng[1] SUN Yong-qiang[1] 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 and Engineering East China University of Science and Technology Shanghai* 200237)

Abstract MFS is a program development system based on rewriting techniques. The language provided by MFS, which called MFSL, is a mixed language that combines enhanced functional language and algebraic specification language. Optimal rules in MFSL can improve efficiency and satisfy specific requirements. Both the type system in MFSL and the mechanism of proving and testing in MFS can help the programmers to find problems early and can improve the correctness of program. The efficiency of the implementation of the system developed by MFS is high due to the techniques used by MFS such as needed parallel outermost reduction strategy, graph reduction and witnessed test set approach. Higher efficiency and correctness of program can be developed by MFS in a shorter period.

Key words Functional language, algebraic specification language, mixed language, type system, test.