函数式语言的集合描述方法及其实现算法*

宋　凯　　廖湖声

（北京工业大学计算机学院　北京　100044）

摘要　本文介绍函数式语言提供的集合描述方法以及实现集合的一种程序变换优化算法. 集合的引入改善了函数式程序的表达能力，并且为提高程序执行效率创造了条件.

关键词　函数式语言，程序变换，集合.

函数式语言具有抽象描述能力强、程序结构简洁、易于维护、易于实现程序变换和程序正确性证明等特点. 由于这些特点，函数式程序语言被认为最有潜力在软件原型的开发中实现可执行的软件规格说明.[1]

函数式语言 FSL 是一种甚高级程序设计语言.[2]它采用延迟求值方式，允许使用高阶函数. 为了提高语言的抽象描述能力，设计中引入了等式函数定义和集合描述等语言功能. 本文介绍 FSL 语言提供的集合描述方法，讨论用于实现集合的一种程序变换优化算法.

1　集合在函数式语言中的作用

高级程序设计语言的特征之一是具有较强的表达能力. 为了便于算法的描述，程序设计语言通常直接支持数组等各种数学描述方法. PASCAL 语言中提供的集合类型，FSL 语言中引入的集合表达式，都是这一特点的体现. 在函数式语言中，引入了一种集合的表示方法——ZF 表达式(Zemelo－Frankel 集合). 按照这种方法，1 个由 1～100 之间的奇数的平方组成的集合可以表示为　{x * x ; [1..100]－>x; x%2==1}

其中[1..100]－>x 表示 x 来自 1～100 组成的整数表，$x * x$ 表示集合元素，$x\%2==1$ 是 x 必须满足的条件.

下面分析利用 FSL 语言提供的集合描述功能设计的一个程序实例：用筛选法求 2～200 之间的素数.

算法. 从 2～200 组成的集合中，筛去 2 的倍数；然后从 2 的后面找出下一个最小数，再从集合中筛去该数的所有倍数；不断重复这个过程，直到处理完给定的所有数；最后得到所有的素数.

---

* 本文研究得到国家自然科学基金和北京市自然科学基金资助. 作者宋凯，女，1958 年生，工程师，主要研究领域为软件工程与编译技术. 廖湖声，1954 年生，副教授，主要研究领域为自动程序设计，函数式程序设计，软件工程与编译技术.

本文通讯联系人：宋凯，北京 100044，北京工业大学计算机学院

本文 1995－11－06 收到修改稿

```
sieve [2..200]
whererec {
    sieve [] = []
and
    sieve [p|x] = (p,sieve {n:x->n;n%p! =0})
}
```

其中[2..200]表示由2～200的递增的自然数序列;函数*sieve*采用了等式定义的形式,[*p*|*x*]是参数表的模式匹配.{*n*:*x*->*n*;*n*%*p*! =0}是一个集合的描述,表示从表*x*中取出满足条件*n*%*p*! =0的元素*n*.

从程序描述中不难看出,FSL程序与传统的过程型程序相比,能直接反映出简明的数学性质,易于验证程序的正确性.这种程序描述更加简洁,体现出具有较强的抽象描述能力,集合表达式的引入使得直接描述筛选法成为可能.

2　集合的实现方法

在FSL语言中ZF表达式的语法形式是{*e*:*q*1;…;*qn*},其中每个*qi*(*i*=1,…,*n*)或是以*e*->*x*的形式出现,代表元素生成式;或是用来限定变量值范围的约束条件表达式.这种集合在实现时采用了表结构,通过函数的递归调用来实现对表中各元素的循环处理.

FSL语言中有关ZF表达式的语法规则如下:

```
expr: {zf_exp} | …                表达式
zf_exp: expr ':' qual_l           集合表达式
qual_l: qual | qual ';' qual—l
qual: expr '->' var               元素生成式
      | expr                      条件式
var: IDEN | …
```

其中*expr*表示表达式,*var*表示简单变元.

在FSL语言的编译实现中,采用了多级程序变换,将函数式程序逐步变换为易于编译实现的程序形式,以求提高程序的执行效率.语言中集合描述的实现就是利用程序变换,将集合表达式变换成具有相同语义的一组函数调用.这里主要采用了2个高阶函数:用于连接多个表的连接函数*concat*和用于筛选表元素的函数*filter*:

```
(concat lambda (e f)
  (if (null e) 'NIL

      (append (f (hd e)) (concat (tl e) f)) ))
```

其中*append*表示表的连接函数.参数*f*的取值是一个函数,用于处理每个表元素.

```
(filter lambda (e f)
  (if (null e) 'NIL
  (if (f (hd e)) (cons (hd e) (filter (tl e) f))
```

(*filter* (*tl e*) *f*))))

其中 *cons* 表示表的构造函数. 参数 *f* 的取值是一个函数,作用于表元素,控制筛选条件.

集合表达式的实现主要是翻译处理描述中的元素生成式和条件表达式. 如果集合表达式中使用了 n 个元素生成式,每个元素生成式分别生成 $m_1, m_2, \cdots, m_n$ 个元素,则一共产生 $m_1 \times m_2 \times \cdots \times m_n$ 个元素. 例如,$\{x * y * z; [1..5] -> x; [1..4] -> y; [1..3] -> z\}$,将生成 60 个元素($5 \times 4 \times 3$). 最外层的元素生成式所生成的每个元素将与其余元素生成式产生的所有元素相结合. 因此,利用函数 *concat* 将对应于每个最外层元素产生的所有元素连接起来,就构成了所有的生成元素(变换规则 4). 条件表达式的实现是通过 *filter* 函数调用来检查元素生成式产生的元素完成的(变换规则 3).

程序变换的算法如下:

变换规则:

[1] $Q[\varepsilon]\ x$ = (*cons x nil*)

[2] Q[*expr1 expr2 expr*$\cdots$] $x = Q$[(*expr1* and *expr2*) *expr*$\cdots$] x

[3] Q[*expr1* −> *var expr2 expr*$\cdots$] x =

concat (*filter expr1* (*lambda* (*var*) *expr2*))(*lambda* (*var*) *sexp*)

where *sexp* = Q[*expr*$\cdots$] x

[4] Q[*expr1* −> *var expr*$\cdots$] x = *concat expr1* (*lambda* (*var*) *sexp*)

where *sexp* = Q[*expr*$\cdots$] x

其中[*expr*$\cdots$]表示表达式中的其余项.

3 程序变换方法的优化

上述程序变换,从语义上已经能够实现集合功能,然而 *concat* 函数中的表连接运算将影响执行的效率. 由于函数式程序的执行没有副作用,表的连接运算(函数 *append*)包含了重新建表的过程:

(*append lambda* (*x y*)

(if (*null x*) *y* (*cons* (*hd x*) (*append* (*tl x*) *y*))))

按照上述实现方法,集合表达式的实现将反复调用 *append* 函数,从而明显地降低整个程序的执行效率.

为了解决这个问题,FSL 语言的实现中采用了接续方法(Continuation),也有人称延拓方法,消除了重建表的过程. 所谓接续方法,是改造 *concat* 函数,多增设一个参数 *c*,表示后续处理的结果. 采用这种参数的表处理函数,将当前计算产生的结果和该参数合并,转交给下一函数调用的接续参数. 当所有函数调用结束时,该参数中已经积累了所有输出结果,可用于直接返回,从而消除了使用表连接的必要.

采用接续方法的一个通用函数是 *foldr* 函数:

(*foldr lambda* (*e g c*)

(if (*null e*) *c* (*D*−1)

(*g* (*hd e*) (*foldr* (*tl e*) *g c*))))

利用该函数能够直接定义出函数 *concat*:

(concat lambda (e f)

(foldr e (lambda (x c) (append (f x) c)) 'NIL))　　(*D*—2)

根据定义 2 可以推出函数参数 *g* 和 *f* 的关系:

g x c <=> append (f x) c　　(*I*—1)

以及 *concat* 和 *foldr* 函数调用之间的关系:

append (concat e f) c <=> append (foldr e g 'NIL) c

<=> foldr e g c

按照上述关系,改写程序变换方法,得到以下规则:

变换规则:

[1] $Q[\varepsilon]\ x\ c = (cons\ x\ c)$

[2] *Q*[*expr*1 *expr*2 *expr*…] *x c* =

Q[(and *expr*1 *expr*2) *expr*…] *x c*

[3] *Q*[*expr*1—>*var expr*…] *x c* =

*(foldr expr*1 *(lambda (var y) sexp) c)*

where *sexp* = *Q*[*expr*…] *x y*

and *y* is *a fresh identifier*

[4] *Q*[*expr*1—>*var expr*2 *expr*…] *x c* =

*(foldr (filter expr*1 *(lambda(var) expr*2*))*

(lambda (var y) sexp) c)

where *sexp* = *Q*[*expr*…] *x y*

and *y* is *a fresh identifier*

按照上述算法实现的 2 个程序变换的结果如下:

1. *Q*[*x*+*y*; *e*1—>*x*; *e*2—>*y*] *'NIL* =>

*(foldr e*1

*(lambda (x y*1*) (foldr e*2

*(lambda (y y*2*) (cons(add x y) y*2*))*

*y*1 *))*

'NIL)

2. *Q*[*x*+*y*; *e*1—>*x*; *e*2—>*y*; *x*>3] *'NIL* =>

*(foldr e*1

*(lambda (x y*1*) (foldr (filter e*2 *(lambda (y) (gt x* 3*)))*

*(lambda (y y*2*) (cons (add x y) y*2*))*

*y*1*))*

'NIL)

4　结束语

在函数式程序设计语言中引入集合描述方法,改善了程序的表达能力.采用程序变换优

化算法不仅有效地实现了集合表示，而且为提高函数式程序的执行效率创造了条件.

随着函数式程序设计在软件规格说明和程序设计自动化领域的应用研究和发展，函数式语言本身及其实现技术也将得到进一步的扩充和发展.

参考文献

1 Turner D A. Functional programs as executable specifications. Mathematic Logic and Programming Languages, Prentic—Hall, 1985. 29～54.

2 廖湖声. 一种可执行函数式规格说明语言 FSL. 程序设计语言研究与发展，电子工业出版社，1994. 62～66.

A NOTATION FOR SET IN FUNCTIONAL LANGUAGES AND ITS IMPLEMENTATION

Song Kai Liao Husheng

(*Computer Institute Beijing Polytechnic University Beijing* 100044)

Abstract This paper presents a powerful notation for set comprehension in functional languages and a program transformation for its implementation. Introduction of the set notation not only increases the expressive power of functional programs, but also offers opportunity for improving their running efficiency.

Key words Functional laugnages, program transformation, set.