第 7 卷第 8 期
1996 年 8 月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 7, No. 8
Aug. 1996

LISP 语言的增量式部分求值器*

李 航 宋立彤 金成植

(吉林大学计算机科学系 长春 130023)

摘要 部分求值在软件优化中有着重要的作用,而增量式计算是避免重复计算的一种技术.本文基于部分求值和增量式计算的技术,实现了一个 LISP 语言的增量式部分求值器,使得函数例化尽量在前次已经产生的剩余程序的基础上进行,从而很好地提高了效率.

关键词 部分求值,函数例化,剩余程序,实参模式,增量式计算.

部分求值在软件优化,尤其在编译器的自动生成中起着重要的作用.[1,2]我们课题组近几年来对函数式、过程式语言的部分求值技术研究取得了一定的成果.[3,4]

在部分求值过程中,前后几个目标表达式经常在已知输入上相似,若对每个目标表达式都从原始函数开始进行例化,则会重复许多工作.如何充分利用上一次已经产生的剩余函数,避免不必要的计算,提高效率,就成为部分求值研究中的一个新问题.为此,我们在传统的部分求值器中加入增量式方法,构造出了增量式部分求值器.

增量式计算方法是避免重复计算、提高效率的一种技术.其核心思想在于,当输入变化不大时,若能充分利用上一次的计算结果,只进行必须的计算,则效率和反应速度都能大大提高.目前,增量式方法已被应用于语法分析[5]、语义分析[6]等很多方面,显示了广阔的应用前景.

1 部分求值器的概念

定义 1. 剩余程序(Residual Program):设 L 是一个程序设计语言,$p,r\in L$,$\langle x_1,\cdots,x_m,y_1,\cdots,y_n\rangle$ 为 p 的任一输入,r 称作 p 关于已知输入 $\langle x_1,\cdots,x_m\rangle$ 的剩余程序,若有 $Lp\langle x_1,\cdots,x_m,y_1,\cdots,y_n\rangle = Lr\langle y_1,\cdots,y_n\rangle$(注:$p\in L$ 表示 p 是 L 语言的程序,下同.).

定义 2. 部分求值器(Partial Evaluator):设 L,L' 是程序设计语言,$\mathrm{mix}\in L'$,mix 称作 L 语言的部分求值器,若对于任一 $p\in L$ 和 p(称作目标程序)的部分已知输入 $\langle x_1,\cdots,x_m\rangle$,$\mathrm{mix}\langle p,x_1,\cdots,x_m\rangle$ 是一个 p 关于 $\langle x_1,\cdots,x_m\rangle$ 的剩余程序,即对于任何未知输入 $\langle y_1,\cdots,y_n\rangle$,有 $L(\mathrm{mix}\langle p,x_1,\cdots,x_m\rangle)\langle y_1,\cdots,y_n\rangle = Lp\langle x_1,\cdots,x_m,y_1,\cdots,y_n\rangle$.

---

* 作者李航,1972 年生,硕士生,主要研究领域为软件自动化.宋立彤,1965 年生,讲师,主要研究领域为软件自动化与软件工程.金成植,1935 年生,教授,主要研究领域为软件自动化与软件工程.

本文通讯联系人:李航,北京 100080,北京 8718 信箱中国科学院软件研究所

本文 1995-07-10 收到修改稿

2　增量式部分求值

LISP 是一种常用的表处理语言,它兼有函数式和过程式语言的特性.

我们在这里对目标程序作如下限制:禁止自定义函数中对全局变量的改变(函数副作用).在部分求值前,我们先要对程序进行 λ—提升(局部函数定义全局化),使其表达式中不再包含 Lambda 表达式.对于 λ—提升,Johnsson[7]已给出详细算法,在此不再赘述.

本文采用动态部分求值技术.

2.1　基本定义

语义域:变量值域 $VAL=ATOM+VAL^*$

函数的实参模式 $PAT=VAL^{arity(FID)}$

变量状态环境 $STAT=ID\rightarrow VAL$

函数定义环境 $GENV=FID\rightarrow ID^{arity(FID)}\times EXP$

函数例化环境 $SENV=FID\rightarrow PAT^*$

为了方便,将 VAL 域增加一个元素 $\perp$,指示"未知"值.

定义 3. 设定 VAL 上的关系:$X,Y\in VAL, X>Y$ iff $Y=\perp$.

定义 4. $M_1,M_2\in PAT$,定义操作符 $=,\sqsupseteq$ 如下:

$M_1=M_2$ iff $M_1\downarrow i=M_2\downarrow i$;

$M_1\sqsupseteq M_2$ iff $M_1\downarrow i>M_2\downarrow i$ 或 $M_1\downarrow i=M_2\downarrow i, i=1,2,\cdots,n$

2.2　基本部分求值算法

部分求值是对包含部分已知参数的目标表达式进行尽可能的化简,而得到剩余程序,剩余程序中的函数是目标程序中函数定义关于部分已知参数生成的变体(称作剩余函数).设原始函数 f 的形参为 $x_1,\cdots,x_m,y_1,\cdots,y_n$,其中 $x_i(1\leqslant i\leqslant m)$ 为可去(已知)参数,$y_j(1\leqslant j\leqslant n)$ 为剩余(未知)参数.当程序中有一函数调用 $(fev_1\cdots ev_m rv_1\cdots rv_n)$,剩余程序中该调用将变成 $(f_{(ev_1,\cdots,ev_m,\perp,\cdots\perp)}rv_1\cdots rv_n)$,$f_{(ev_1,\cdots,ev_m,\perp,\cdots\perp)}$ 为对应的剩余函数名,而 $rv_1,\cdots,rv_n$ 为剩余实参;同时剩余程序中将生成该剩余函数定义.这里为了说明方便起见,我们假设所有可去参数都在参数表的前面,实际中可以不是这样.

下面给出具体算法.

函数 $PE:EXP\rightarrow PROG\rightarrow STAT\rightarrow PROG$

输入:1. 目标表达式 2. 目标程序 3. 初始变量状态

输出:剩余程序

$PE[\![E]\!]prg\ \sigma=$ let $\langle\rho,\beta\rangle=FD[\![prg]\!][\ ][\ ]$

and $\langle e,f,\rho',\beta',\sigma'\rangle=SP[\![E]\!]\rho\beta\sigma$

in $f\square e$

其中[ ]表示空映射,□表示连接

函数 $FD:PROG\rightarrow GENV\rightarrow SENV\rightarrow GENV\times SENV$

$FD[\![(\mathit{defun}\ fid\ (x_1\cdots x_n)\ E)]\!]\rho\beta=\langle\rho[\langle(x_1,\cdots,x_n),E\rangle/fid],\beta[(\ )/fid]\rangle$

$FD[\![fdef_1\square fdef_2]\!]\rho\beta=$ let $\langle\rho_1,\beta_1\rangle=FD[\![fdef_1]\!]\rho\beta$ in $FD[\![fdef_2]\!]\rho_1\beta_1$

函数 $SP:EXP\to GENV\to SENV\to STAT\to EXP\times FDEF\times GENV\times SENV\times STAT$

SP 对给定表达式,根据3种环境信息生成剩余表达式,并产生剩余函数定义,将其加入到函数定义环境中,同时生成改变后的函数例化环境及变量状态环境.

$SP[\![C]\!]\rho\beta\sigma=\langle C,\varepsilon,\rho,\beta,\sigma\rangle$,$C$ 为 t,nil,数,或$(quote\ E)$,ε 表示空

$SP[\![v]\!]\rho\beta\sigma=$ if $\sigma(v)=\perp$ then $\langle v,\varepsilon,\rho,\beta,\sigma\rangle$ else $\langle\sigma(v),\varepsilon,\rho,\beta,\sigma\rangle$

$SP[\![(setq\ id\ E)]\!]\rho\beta\sigma=$let $\langle e,f,\rho',\beta',\sigma'\rangle=SP[\![E]\!]\rho\beta\sigma$ in

if $AV[\![e]\!]=K$ then $\langle e,f,\rho',\beta',\sigma'[e/id]\rangle$

else $\langle(setq\ id\ e),f,\rho',\beta',\sigma'[\perp/id]\rangle$

函数 $AV:EXP\to\{U,K\}$

$$AV[\![E]\!]=\begin{cases}K,\text{若 }E\text{ 为 }t,nil,\text{数或}(quote\ E)\\U,\text{否则}\end{cases}$$

$SP[\![(\text{if } E_1\ E_2\ E_3)]\!]\rho\beta\sigma=$ if $AV[\![e_1]\!]=K$

then if $e_1\neq nil$ then $SP[\![E_2]\!]\rho_1\beta_1\sigma_1$ else $SP[\![E_3]\!]\rho_1\beta_1\sigma_1$

else $\langle(\text{if } e_1\ e_2'\ e_3'),f_1\square f_2\square f_3,\rho_3,\beta_3,\sigma'\rangle$

其中 $\langle e_1,f_1,\rho_1,\beta_1,\sigma_1\rangle=SP[\![E_1]\!]\rho\beta\sigma$,$\langle e_2,f_2,\rho_2,\beta_2,\sigma_2\rangle=SP[\![E_2]\!]\rho_1\beta_1\sigma_1$,$\langle e_3,f_3,\rho_3,\beta_3,\sigma_3\rangle=SP[\![E_3]\!]\rho_2\beta_2\sigma_1$

$\langle s_2,\sigma_2'\rangle=COMP(\sigma_2,\sigma_3)$,$\langle s_3,\sigma_3'\rangle=COMP(\sigma_3,\sigma_2)$

$e_2'=(progn\ s_2\ e_2)$,$e_3'=(progn\ s_3\ e_3)$,$\sigma'=\sigma_2'\odot\sigma_3'$

$COMP(\sigma_1,\sigma_2)=\langle(setq\ x_1\ N_1)\cdots(setq\ x_m\ N_m),\sigma_1[\perp/x_1]\cdots[\perp/x_m]\rangle$

其中 $\sigma_1(x_i)\neq\perp$ 且 $\sigma_1(x_i)\neq\sigma_2(x_i)$,$N_i=\sigma_1(x_i)$,$1\leqslant i\leqslant m$

$\sigma_1\odot\sigma_2(x)=$if $\sigma_1(x)=\sigma_2(x)$ then $\sigma_1(x)$ else $\perp$

$SP[\![(\text{do }(TestExp\ ResExp)\ E_1\cdots E_n)]\!]\rho\beta\sigma=$

let $\langle te,f_1,\rho_1,\beta_1,\sigma_1\rangle=SP[\![TestExp]\!]\rho\beta\sigma$ in

if $AV[\![te]\!]=K$

then if $te\neq nil$ then $SP[\![ResExp]\!]\rho_1\beta_1\sigma_1$

else $SP[\![(progn\ E_1\cdots E_n(\text{do }\langle TestExp\ ResExp\rangle\ E_1\cdots E_n))]\!]\rho_1\beta_1\sigma_1$

else let $\langle S_1,f_2,\rho_2,\beta_2,\sigma_2\rangle=SP[\![(progn\ E_1\cdots E_n)]\!]\rho_1\beta_1\sigma_1$

and $\sigma_3=\sigma_2[\perp/X_1]\cdots[\perp/X_m]$

and $\langle S_2,f_3,\rho_3,\beta_3,\sigma_4\rangle=[\![(progn\ E_1\cdots E_n)]\!]\rho_2\beta_2\sigma_3$

in $\langle(\text{if } TestExp\ (progn\ (setq\ X_1\ N_1)\cdots(setq\ X_m\ N_m)\ ResExp)$

$(progn\ S_1(setq\ Y_1\ M_1)\cdots(setq\ Y_k\ M_k)(\text{do }(TestExp\ ResExp)\ S_2))),f_1\square f_2\square f_3,\rho_3,\beta_3,\sigma_4\rangle$

其中 $X_i\in COL[\![(progn\ E_1\cdots E_n)]\!]$ 且 $\sigma_1(X_i)\neq\perp$,$N_i=\sigma_1(X_i)$,$1\leqslant i\leqslant m$

$Y_j\in COL[\![(progn\ E_1\cdots E_n)]\!]$ 且 $\sigma_2(Y_j)\neq\perp$,$M_j=\sigma_2(Y_j)$,$1\leqslant j\leqslant k$

函数 $COL:EXP\to POW(ID)$

COL 搜集表达式中被改变(被 $setq$ 赋值)的变量,构成一个集合.限于篇幅,具体定义略.

$SP[\![(progn\ E_1\cdots E_n)]\!]\rho\beta\sigma=$if $AV[\![e_1]\!]=K\wedge AV[\![e_2]\!]=K\wedge\cdots\wedge AV[\![e_{n-1}]\!]=K$

then $\langle e_n,f,\rho_n,\beta_n,\sigma_n\rangle$

else $\langle(progn\ e_1'\cdots e_n'),f,\rho_n,\beta_n,\sigma_n\rangle$

其中 $\rho_0=\rho,\beta_0=\beta,\sigma_0=\sigma,\langle e_i,f_i,\rho_i,\beta_i,\sigma_i\rangle=SP[\![E_i]\!]\rho_{i-1}\beta_{i-1}\sigma_{i-1}$

$e_i'=$if $AV[\![e_i]\!]=K$ then ε else e_i,$1\leqslant i\leqslant n$,$f=f_1\square f_2\square\cdots\square f_n$

$SP[\![(PF\ E_1 \ldots E_n)]\!]\rho\beta\sigma$ = if $AV[\![e_1]\!]=K \wedge AV[\![e_2]\!]=K \wedge \cdots \wedge AV[\![e_n]\!]=K$ '

then $\langle apply(PF, e_1, e_2 \cdots, e_n), \varepsilon, \rho_n, \beta_n, \sigma_n \rangle$

else $\langle (PF\ e_1\ e_2 \cdots e_n), f, \rho_n, \beta_n, \sigma_n \rangle$

其中 PF 为基本运算函数,如 car,add,not 等,$\rho_0=\rho, \beta_0=\beta, \sigma_0=\sigma$

$\langle e_i, f_i, \rho_i, \beta_i, \sigma_i \rangle = SP[\![E_i]\!]\rho_{i-1}\beta_{i-1}\sigma_{i-1}, 1 \leqslant i \leqslant n, f=f_1 \square f_2 \square \cdots \square f_n$,

$apply$ 执行基本运算函数,得出结果

$SP[\![(FID\ E_1 \cdots E_n)]\!]\rho\beta\sigma$ =

let $\langle (x_1, \cdots, x_n), E \rangle = \rho(FID)$

and $\langle (ap_1, \cdots, ap_n), (re_1, \cdots, re_k), (x_1', \cdots, x_k') \rangle = LP[\![(e_1, \cdots, e_n)]\!](x_1, \cdots, x_n)$

and $\langle e, f', \rho', \beta', \sigma' \rangle = SP[\![E]\!]\rho\beta(\sigma_n[ap_1/x_1]\cdots[ap_n/x_n])$

in if $AV[\![e]\!]=K$ then $\langle e, \varepsilon, \rho', \beta', \sigma_n \rangle$

else let $\rho''=\rho'[\langle (x_1', \cdots, x_k'), e \rangle / FID_{(ap_1, \cdots, ap_n)}]$

and $f''=f_1 \square f_2 \square \cdots \square f_n \square f' \square (defun\ FID_{(ap_1, \cdots, ap_n)}(x_1' \cdots x_k')\ e)$

and $\beta''=\beta'[(ap_1, \cdots, ap_n) :: \beta'(FID)/FID][()/FID_{(ap_1, \cdots, ap_n)}]$

in $\langle (FID_{(ap_1, \cdots, ap_n)}\ re_1 \cdots re_k), f'', \rho'', \beta'', \sigma_n \rangle$

其中 $\rho_0=\rho, \beta_0=\beta, \sigma_0=\sigma$, $\langle e_i, f_i, \rho_i, \beta_i, \sigma_i \rangle = SP[\![E_i]\!]\rho_{i-1}\beta_{i-1}\sigma_{i-1}, 1 \leqslant i \leqslant n$

函数 $LP: EXP^n \rightarrow ID^n \rightarrow PAT \times EXP^{n'} \times ID^{n'}$

LP 将实参表中的未知元素换成⊥,形成函数的实参模式、剩余实参表和剩余形参表.具体定义略.

例如:$LP[\![(3, x, (quote\ (1\ 0)), (add\ y\ 1))]\!](x1, x2, x3, x4) = ((3, \bot, (quote\ (1\ 0)), \bot), (x, (add\ y\ 1)), (x2, x4))$

2.3　增量式部分求值

部分求值过程中,前后几个目标表达式经常在已知输入上相类似,增量式部分求值技术的核心在于充分利用原来的剩余程序中的已生成的剩余函数,来对现在的目标表达式进行例化,使得尽量减少重复计算,提高效率.

例如,有函数定义$(defun\ f\ (x\ y\ z)\ E)$,f 已例化过的实参模式有$(1, \bot, \bot)$和$(\bot, 1, \bot)$,当对新的一个函数调用$(f\ 2\ 1\ x)$进行例化时(其中 x 未知),可变为例化$(f_{(\bot,1,\bot)}\ 2\ x)$,这样就会节省计算,提高效率.其中的 $f_{(\bot,1,\bot)}$是一个由原始函数名和它例化过的实参模式共同构成的一个剩余函数名,在实现时用一个新标识符表示,但为了统一,在算法中仍写成原始函数名加实参模式作下标的形式.

下面描述增量式部分求值的具体算法.

函数 $PE2$: $EXP \rightarrow GENV \rightarrow SENV \rightarrow STAT \rightarrow PROG$

输入:(1)目标表达式,(2)前次部分求值得到的函数定义环境,(3)对前次的目标表达式部分求值得出的例化环境,(4)初始状态.

输出:剩余程序

$PE2[\![E]\!]\rho\beta\sigma$ = let $\langle e, f, \rho', \beta', \sigma' \rangle = SP2[\![E]\!]\rho\beta\sigma$ in $f \square e$

函数 $SP2: EXP \rightarrow GENV \rightarrow SENV \rightarrow STAT \rightarrow EXP \times FDEF \times GENV \times SENV \times STAT$

$SP2$ 对给定表达式,根据 3 种环境信息生成剩余表达式,并产生剩余函数,加入到函数定义环境中,同时产生改变后的函数例化环境及变量状态环境.$SP2$ 的定义除了对自定义函数的处理不同外,其余同 SP,故以下只列出处理自定义函数的算法.

$SP2[\![(FID\ E_1\cdots E_n)]\!]\rho\beta\sigma=\langle e,f,\rho',\beta',\sigma'\rangle$

其中 $\rho_0=\rho,\beta_0=\beta,\sigma_0=\sigma$

$\langle e_i,f_i,\rho_i,\beta_i,\sigma_i\rangle=SP2[\![E_i]\!]\rho_{i-1}\beta_{i-1}\sigma_{i-1},1\leqslant i\leqslant n$

$\langle e,f',\rho',\beta',\sigma'\rangle=FS[\![(FID\ e_1\cdots e_n)]\!]\beta(FID)\rho_n\beta_n\sigma_n$

$f=f_1\square f_2\square\cdots\square f_n\square f'$

函数 $FS:EXP\to PAT^*\to GENV\to SENV\to STAT\to EXP\times FDEF\times GENV\times SENV\times STAT$

FS 在函数 FID 例化过的实参模式表中查找 FID 已例化过的实参模式，找出匹配的 1 个 b，在 FID_b 的基础上例化，从而在已经简化的表达式上进行计算，提高效率.

$FS[\![(FID\ e_1\cdots e_n)]\!]L\rho\beta\sigma=$

let $\langle(x_1,\cdots,x_n),E\rangle=\rho(FID)$

and $\langle(ap_1,\cdots,ap_n),(re_1,\cdots,re_k),(x_1',\cdots,x_k')\rangle=LP[\![(e_1\cdots e_n)]\!](x_1,\cdots,x_n)$

in if $L=()$

then let $\sigma_1=\sigma[ap_1/x_1]\cdots[ap_n/x_n]$

and $\langle e,f_1,\rho_1,\beta_1,\sigma_2\rangle=SP2[\![E]\!]\rho\beta\sigma_1$

in if $AV[\![e]\!]=K$

then $\langle e,\varepsilon,\rho_1,\beta_1,\sigma\rangle$

else let $\beta_2=\beta_1[(ap_1,\cdots,ap_n)::\beta_2(FID)/FID][()/FID_{(ap_1,\cdots,ap_n)}]$

and $\rho_2=\rho_2[\langle(x_1',\cdots,x_k'),e\rangle/FID_{(ap_1,\cdots,ap_n)}]$

and $f=f_1\square(defun\ FID_{(ap_1,\cdots,ap_n)}(x_1'\cdots x_k')\ e)$

in $\langle(FID_{(ap_1,\cdots,ap_n)}\ re_1\cdots re_k),f,\rho_2,\beta_2,\sigma\rangle$

else let $b::L_1=L$ in

if $(ap_1,\cdots,ap_n)\supseteq b$

then let $(e_1',\cdots,e_k')=R[\![(e_1,\cdots,e_n)]\!]b$

in $FS[\![(FID_b\ e_1'\cdots e_k')]\!]\beta(FID_b)\ \rho\beta\sigma$

else

if $(ap_1,\cdots,ap_n)=b$

then $\langle(FID_b\ re_1\cdots re_k),\varepsilon,\rho,\beta,\sigma\rangle$

else $FS[\![(FID\ e_1\cdots e_n)]\!]L_1\ \rho\beta\sigma$

函数 $R:EXP^m\to PAT\to EXP^{m'}$

R 将实参表与模式进行相“减”，得到剩余实参表，具体定义略.

例如：$R[\![(2,x,(quote\ (1\ 0)),(add\ y\ 1))]\!](2,\perp,\perp,\perp)=(x,(quote\ (1\ 0)),(add\ y\ 1))$

2.4 优 化

上述方法只是一个大体框架，具体实现时还要处理许多问题，其中最主要的是对递归和循环的处理.

由于递归函数的存在，使得在函数例化过程中剩余函数的生成过程可能不终止，即一个原始函数可能会生成无限多个剩余函数而陷入死循环. 对这一问题我们采取一种数据稳定性测试方法加以解决，具体方法可参见文献[3].

对于循环，在循环终止条件已知时，也并不完全展开，而是对之进行模拟执行，求出其终止时的状态，然后将化简后的循环体构成的循环和终止状态作为例化结果，这样可以避免剩余程序代码过长的毛病，又可以得到本可得到的某些变量的最终状态；在循环终止条件未知时，将在循环体中被改变的变量的值改为⊥，然后对循环体在此基础上进行化简(外

提循环不变式),得到的简化式和状态作为剩余循环的循环体和结果状态.

3　运行实例

设目标程序中有函数定义:

```
(defun f(x y) (progn (if (lessp x y)
                         (progn (setq y (times y 2)) (setq k (add 3 y)))
                         (setq k (add 4 y)))
                     (add k 2)))
(defun s(i y a b) (progn (setq x 3) (do ((greaterp i 100) y)
                                        (setq y (add y x) (setq x (add a b))
                                        (setq i (add i 1)))))
(defun e(x y z w) (if (zerop x) w
                      (if (greaterp y z)(e (differ x 1)(differ y z)(times z 2)(add w x))
                          (e (differ x 1)(differ z y)(times y 2)(differ w x)))))
(defun g(x1 x2 x3 x4)(progn (setq t1 (f x2 x4)) (setq t2 (car x3))
                            (list (s x2 x1 x4 x1) (e (times 10 t1) t2 x4 x1))))
```

令(g x1 x2 (quote (8 (0 1))) 4)为第1次计算时的目标表达式,初始状态为[x1→⊥,x2→⊥],则部分求值后得到剩余程序(经优化)如下:

```
(defun f1 (x) (progn (if (lessp x 4) (setq k 11) (setq k 8)) (add k 2))
(defun s1 (i y b) (if (greaterp i 100) (progn (setq x 3) y)
                      (progn (setq y (add y 3) (setq x (add 4 b)) (setq i (add i 1))
                             (do ((greaterp i 100) y)(setq y (add y x)) (setq i(add i 1)))))))
(defun e1 (x w) (if (zerop x) w (e2 (differ x 1) (add w x))))
(defun e2 (x w) (if (zerop x) w (e2 (differ x 1) (differ w x))))
(defun g1 (x1 x2) (progn (setq t1 (f1 x2 )) (list (s1 x2 x1 x1) (e1 (times 10 t1) x1))))
(g1 x1 x2)
```

函数定义环境略,函数例化环境为[f→(⊥,4),s→(⊥,⊥,4,⊥),e→((⊥,8,4,⊥)(⊥,4,8,⊥)),g→(⊥,⊥,(quote (8 (0 1))),4)].

令(g x1 2 (quote (8 (1 0))) 4)为第2次的目标表达式,初始状态[x1→⊥],则可得到剩余程序如下:

```
(defun s2 (y b) (progn (setq y (add y 3) (setq x (add 4 b) (setq i 3)
                       (do ((greaterp i 100) y) (setq y (add y x)) (setq i (add i 1))))))
(defun e1 (x w) (if (zerop x) w (e2 (differ x 1) (add w x))))
(defun e2 (x w) (if (zerop x) w (e2 (differ x 1) (differ w x))))
(defun g2(x1) (list (s2 x1 x1) (e1 130 x1)))
(g2 x1)
```

其中,函数(f 2 4)是在f1的基础上计算而得到13,剩余函数e1和e2直接从上次的剩余程序中得到,而s2是在s1的基础上例化而来,g2是在g的基础上例化而来.例化环境:[f→(⊥,4),s→(⊥,⊥,4,⊥),s1→(2,⊥,⊥),e→((⊥,8,4,⊥),(⊥,4,8,⊥)),g→((⊥,⊥,(quote (8 (0 1))),4),(⊥,2,(quote (8 (1 0))),4))].

4　结论及进一步工作

增量式计算是具有广泛应用价值的一种方法,交互式计算环境的广泛使用,使得增量式

计算更成为必要.它与部分求值技术的结合,是一新的尝试.我们根据上述算法,设计并实现了一个LISP语言的增量式部分求值器,使得部分求值的效率有较大的提高.

但由于在上述算法中寻找匹配的实参模式时只用到线性的信息组织方式和查找方法,故不能保证在最匹配的剩余函数基础上进行例化.如何组织和查找例化实参信息,使得例化在最优的基础上进行,是我们进一步的工作.

参考文献

1 Jones N D, Sestoft P, Sondergoard H. MIX: a self—applicable partial evaluator for experiments in compiler generation. Journal of LISP and Symbolic Computation, 1989,2:9~50.

2 Peter Sestoft. The structure of a self—applicable partial evaluator. In: Jones N D, Ganzinger H ed., Programs as Data Objects LNCS 217, Springer Verlag, 1986. 236~256

3 宋立彬,金成植.函数式语言的部分求值技术.软件学报,1996,7(5):306~313.

4 刘磊,郑红军,金成植.基于信息流分析的部分求值技术.软件学报,1995,6(8):509~513.

5 Ghezz C, Mandrioli D. Augmenting parsers to support incrementality. JACM, 1980,27(3):564~579.

6 Derek Kiong, Jim Welsh. Incremental semantic evaluation in language—based editors. Software—Practice and Experience, 1992,22(2):111~135.

7 Thomas Johnsson. Lambda lifting: transforming programs to recursive equations. In: Conference on Functional Programming Language and Computer Architecture, Nancy, LNCS 201, Springer Verlag, 1985. 190~203.

AN INCREMENTAL PARTIAL EVALUATOR OF LISP

Li Hang Song Litong Jin Chengzhi

(*Department of Computer Science Jilin University Changchun* 130023)

Abstract Partial evaluation plays a very important role in software optimization, while incremental computation is a technique for avoiding duplicative computation. Based on technique of partial evaluation and incremental computation, the authors implemented an incremental partial evaluator of LISP, in which function specialization is done as far as possible on the base of residual program got last time so that efficiency can be improved.

Key words Partial evaluation, function specialization, residual program, actual parameter mode, incremental computation.