第7卷第5期 软 件 学 报 Vol. 7, No. 5
1996年5月 JOURNAL OF SOFTWARE May 1996

函数式语言的部分求值技术*

宋立彤 金成植

(吉林大学计算机科学系 长春 130023)

摘要 本文基于抽象解释技术设计并实现了一函数式语言部分求值器FMIX. FMIX在设计方法和实现策略上具有一定独到之处,系统运行效率较高.

关键词 部分求值,抽象分析,函数例化,剩余程序,可去参数,剩余参数.

部分求值技术对程序优化及软件自动生成,特别是编译程序的自动生成有着极为重要的作用,已引起人们的极大重视. 目前绝大部分部分求值都采用动态求值方法,即利用已知输入对程序进行例化展开,从而得出剩余程序. 由这一方法得出的剩余程序虽执行效率提高,但不可避免地具有程序空间太大的缺点. 因此,目前国外一些相关文章都着眼于基于静态分析技术的部分求值方法的研究,如文献[1~4]. 用这一方法求得的剩余程序兼顾了代码空间和执行效率2大因素,使得部分求值更加有效和实用. 本文基于这一思想具体给出了一种基于静态抽象分析技术的函数式语言的部分求值方法,并设计和实现了一实用的部分求值器FMIX. 利用FMIX所求得的剩余程序不仅代码空间小,而且执行效率较高.

1 函数式语言SFL简介

函数式语言的理论基础是Lambda演算理论、Domain理论和不动点理论等,典型的函数式语言之一是由Edinburgh大学的R. Harper[5]等人在报告《STANDARD ML》中提出的ML语言. 这里的SFL基本等同于去掉了高阶函数部分后的ML. SFL语言的语法结构如下:

$Prog ::= E$

$E ::= V \mid C \mid (E_1,\ldots,E_n) \mid [E_1,\ldots,E_n] \mid ctr(E_1,\ldots,E_n) \mid f(E_1,\ldots,E_n) \mid E_1\ op\ E_2 \mid$
$D \text{ in } E_1 \mid \text{if } E_1 \text{ then } E_2 \text{ else } E_3 \mid \text{case } E \text{ of } Matc \text{ end} \mid \lambda(Pt_1,\ldots,Pt_n).E_1$

$Matc ::= Pt_1 : E_1 \ldots Pt_k : E_k$

$D ::= \text{let } Lb \mid \text{letrec } Lb \mid \text{type } Tb$

$Pt ::= SPt \mid SPt : Tp$

$SPt ::= - \mid C \mid V \mid (Pt_1,\ldots,Pt_n) \mid [Pt_1,\ldots,Pt_n] \mid ctr(Pt_1,\ldots,Pt_n)$

---

* 作者**宋立彤**,1965年生,讲师,主要研究领域为形式语义学及软件自动化,软件工程. **金成植**,1935年生,教授,主要研究领域为软件新技术与软件自动化.

本文通讯联系人,宋立彤,长春 130023,吉林大学计算机科学系

本文1995-02-20收到修改稿

$Lb ::= Pt = E \mid f(Pt_1, \ldots, Pt_n) = E \mid Lb_1 \text{ and } Lb_2$

$Tb ::= id = Tp_1 \mid Tb_1 \text{ and } Tb_2$

$Tp ::= Btp \mid (Tp) \mid Tp * \mid Tp_1 \rightarrow tp_2 \mid Tp_1 \times \ldots \times Tp_2 \mid ctr_1(Tp_1) + \ldots + ctr_n(Tp_n)$

部分求值之前需对程序作 Lambda 提升(局部函数提升). 关于 Lambda 提升,Johnsson[6]已给出算法思想,这里不再赘述.

2 抽象分析

近年来,抽象分析技术已在传统的过程式语言、逻辑式语言及函数式语言的编译优化、程序分析和部分求值等领域取得了广泛的推广和应用. 抽象分析的目标是静态地分析出所需程序的某种性质. 例如,在给定部分输入的情况下分析程序中函数的哪些参变量为已知,哪些为未知. 究竟要分析哪些性质,具体依赖于所要研究的问题. 本文中的抽象分析主要是在静态环境下首先根据已知输入对目标函数定义进行分析,得出其参变量在程序动态执行时的已知/未知信息,进而得出其它函数参数的已知/未知信息.

定义 1. 抽象分析的抽象域定义为 $\Omega = \{\kappa, \upsilon\}$. 其中 κ 代表完全已知表达式的抽象值,υ 代表非完全已知表达式的抽象值.

定义 2. Ω 上的实例化序为:$\upsilon \angle \kappa$, $\angle$ 具有自反性、传递性和反对称性. $(\Omega, \angle)$ 构成了完全格.

定义 3. 设 $\tau_1, \tau_2 \in \Omega$,则定义运算 $\Lambda: \tau_1 \Lambda \tau_2 = (\tau_1 \angle \tau_2 \rightarrow \tau_1, \tau_2)$.

3 部分求值

首先给出几个有关部分求值的概念.

定义 4. 令 L 为程序设计语言,p 为 L 语言程序. 则 r 为 p 关于已知输入 $\langle x_1, \ldots, x_m \rangle \in D^*$ 的剩余程序. iff $L\ p\langle x_1, \ldots, x_m, y_1, \ldots, y_n \rangle = L\ r\langle y_1, \ldots, y_n \rangle$ 对所有余下输入序列 $\langle y_1, \ldots, y_n \rangle \in D^*$.

定义 5. 一个部分求值器 MIX 是一程序,它对任何 L 语言程序 p 和部分已知输入 $\langle x_1, \ldots, x_m \rangle \in D^*$,$MIX\langle p, x_1, \ldots, x_m \rangle$ 为 p 关于已知输入 $\langle x_1, \ldots, x_m \rangle \in D^*$ 的剩余程序,换句话说,$(MIX\langle p, x_1, \ldots, x_m \rangle)\langle y_1, \ldots, y_n \rangle = p\langle x_1, \ldots, x_m, y_1, \ldots, y_n \rangle$ 对所有余下输入序列 $\langle y_1, \ldots, y_n \rangle \in D^*$.

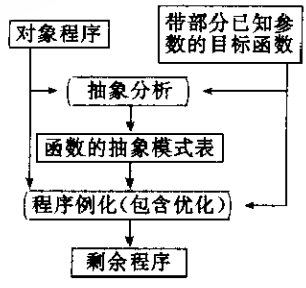

图1 部分求值器(FMIX)的工作流程图

程序 p 称为对象程序,$\langle x_1, \ldots, x_m \rangle$ 称为已知输入,$\langle y_1, \ldots, y_n \rangle$ 称为未知或剩余输入. 部分求值器 $FMIX$ 的流程如图 1,从流程图不难看出,$FMIX$ 的构造分为抽象分析和程序例化 2 个阶段,下面分述这 2 个阶段.

3.1 抽象分析

本阶段的目的是计算对象程序中每一函数参数的稳定抽象值,用以描述它们在部分求值时是确定已知的还是可能未知的. 该阶段不使用已知输入的实际值,而使用参数的已/未

知描述.

该阶段的输入为:①带有部分已知参数的函数调用表达式,②对象程序;输出为描述,即每一函数到其参数描述的映射.

本阶段,我们使用一种信息流分析方法.该方法是在函数参数和被其调用的函数参数之间建立 Ω 域上的相关方程组,然后对整个程序建立起来的方程组求解,从而最终求出各函数参数的抽象描述.[6]

函数 $P:EXP\to\Sigma\to E\times[IDE\to E]$, $\Sigma=VAR\to E,\sigma\in\Sigma$

$$E=VAR^*+(E,\ldots,E)+ctr(E,\ldots,E)+[E]$$

$P\llbracket C\rrbracket_\sigma=(\varphi(\text{空集}),\varepsilon(\text{空}))$

$P\llbracket V\rrbracket_\sigma=(\sigma(V),\varepsilon)$

$P\llbracket(E_1,\ldots,E_n)\rrbracket_\sigma=((\rho_1,\ldots,\rho_n),\pi)$

where $(\rho_i,\pi_i)=P\llbracket E_i\rrbracket\sigma,\pi=\pi_1+\ldots+\pi_n,(1\leqslant i\leqslant n),+:operator\ append.$

$P\llbracket ctr(E_1,\ldots,E_n)\rrbracket_\sigma=(ctr(\rho_1,\ldots,\rho_n),\pi)$

$P\llbracket[E_1\mid E_2]\rrbracket_\sigma=([\rho_1\mid\rho_2],\pi)$

$P\llbracket E_1\ op\ E_2)\rrbracket_\sigma=(\rho_1\vee\rho_2,\pi)$

where

$\vee:E\times E\to E$

$\rho_1\vee\rho_2=$case (ρ_1,ρ_2) of

$((\rho_1,\ldots,\rho_n),(\rho_1',\ldots,\rho_n'))$	$:(\rho_1\vee\rho_1',\ldots,\rho_n\vee\rho_n')$
$(ctr(\rho_1,\ldots,\rho_n),ctr(\rho_1',\ldots,\rho_n'))$	$:ctr(\rho_1\vee\rho_1',\ldots,\rho_n\vee\rho_n')$
$([\rho_1\mid\rho_2],[\rho_1'\mid\rho_2'])$	$:[\rho_1\vee\rho_1'\mid\rho_2\vee\rho_2']$
$((\rho_1,\ldots,\rho_n),\rho)$	$:(\rho_1\vee\rho,\ldots,\rho_n\vee\rho)$
$(ctr(\rho_1,\ldots,\rho_n),\rho)$	$:ctr(\rho_1\vee\rho,\ldots,\rho_n\vee\rho)$
$([\rho_1\mid\rho_2],\rho)$	$:[\rho_1\vee\rho\mid\rho_2\vee\rho]$
$(\rho_1,\rho_2):\rho_1\cup\rho_2$	

end

$P\llbracket$if E_1 then E_2 else $E_3\rrbracket_\sigma=(\rho_2\vee\rho_3,\pi)$

$P\llbracket$let $P_1=E_1\ldots P_n=E_n$ in $E\rrbracket_\sigma=(\rho,\pi')$

where $\sigma_0=\sigma,(\rho_i,\pi_i)=P\llbracket E_i\rrbracket_{\sigma_{i-1}},\sigma_i=\sigma_{i-1}\&mat_1\llbracket P_i,\rho_i\rrbracket,(1\leqslant i\leqslant n),(\rho,\pi)=P\llbracket E\rrbracket_{\sigma_n},\pi'=\pi_1+\ldots\pi_n+\pi.$

$\&:\Sigma\times\Sigma\to\Sigma$ $\quad\sigma_1(V)\&\sigma_2(V)=\begin{cases}\sigma_2(V) & V\in domain(\sigma_2)\\ \sigma_1(V) & V\in domain(\sigma_1)\\ \{V\} & \text{否则}\end{cases}$

$mat_1:PAT\times E\to\Sigma$

$mat_1\llbracket C,\rho\rrbracket=\varepsilon,mat_1\llbracket V,\rho\rrbracket=[\rho/V]$,

$mat_1\llbracket(P_1,\ldots,P_n),(\rho_1,\ldots,\rho_n)\rrbracket=\sigma_1\&\ldots\&\sigma_n$ $\quad mat_1\llbracket ctr(P_1,\ldots,P_n),ctr(\rho_1,\ldots,\rho_n)\rrbracket=$同前

$mat_1\llbracket(P_1,\ldots,P_n),\rho\rrbracket=\sigma_1'\&\ldots\&\sigma_n'$, $\quad mat_1\llbracket ctr(P_1,\ldots,P_n),\rho\rrbracket=$同前

$mat_1\llbracket[P_1\mid P_2],[\rho_1\mid\rho_2\rrbracket=\sigma_1\cup\sigma_2$, $\quad mat_1\llbracket[P_1\mid P_2],\rho\rrbracket=\sigma_1'\cup\sigma_2'$

where $\sigma_i=mat_1\llbracket P_i,\rho_i\rrbracket,\sigma_i'=mat_1\llbracket P_i,\rho\rrbracket\ (1\leqslant i\leqslant n)$

$P\llbracket$letrec $P_1=E_1\ldots P_n=E_n$ in $E\rrbracket_\sigma=(\rho,\pi')$

where $(\rho_i,\pi_i)=P\llbracket E_i\rrbracket_\sigma',\sigma'=\sigma\&mat_1\llbracket P_1,\rho_1\rrbracket\&\ldots\&mat_1\llbracket P_n,\rho_n\rrbracket\ (1\leqslant i\leqslant n)$,

$(\rho,\pi)=P\llbracket E\rrbracket_\sigma',\pi'=\pi_1+\ldots\pi_n+\pi$,此处需求关于 σ' 的最小不动点.

$P\llbracket$case E of $P_1:E_1\ldots P_n:E_n$ end$\rrbracket_\sigma=(\rho',\pi')$

where $(\rho,\pi)=P[\![E]\!]_\sigma,\sigma_i=\sigma\&mat_1[\![P_i,\rho]\!],(\rho_i,\pi_i)=P[\![E_i]\!]_{\sigma_i},\rho'=\rho_1\vee\ldots\vee\rho_n,\pi'=\pi_1+\ldots+\pi_n$

$P[\![F(E_1,\ldots,E_n)]\!]_\sigma=(\rho',\pi')$

where $\rho'=H[\![\rho_1\vee\ldots\vee\rho_n]\!],\pi'=\pi_1+\ldots+\pi_n+[F_{a1}=Z[\![H[\![\rho_1]\!]]\!],\ldots,F_{an}=Z[\![H[\![\rho_n]\!]]\!]]$

$Z:VAR*\to AEXP\quad Z[\![\{V_1,\ldots,V_n\}]\!]=V_1\wedge\ldots\wedge V_n\quad H:E\to VAR*$

$H[\![\{V_1,\ldots,V_n\}]\!]=\{V_1,\ldots,V_n\},H[\![(\rho_1,\ldots,\rho_n)]\!]=H[\![\rho_1]\!]\cup\ldots\cup H[\![\rho_n]\!]$

$H[\![ctr(\rho_1,\ldots,\rho_n)]\!]=$同上,$H[\![[\rho_1|\rho_2]]\!]=H[\![\rho_1]\!]\cup H[\![\rho_2]\!]$

函数 dfa:　$PROG\to[IDE\to E],PROG=IDE\to FDEF,\theta\in PROG$

该函数作数据流分析并在一函数和被其调用函数参数之间构造相关方程组.

$dfa_\varepsilon=[\ ]$

$dfa_{[F(E_1,\ldots,E_n)=Exp/F]\theta}=$let $\sigma=\Gamma[\![F_{a1},E_1]\!]\&\ldots\&\Gamma[\![F_{an},E_n]\!](\rho,\pi)=P[\![Exp]\!]_\sigma$

in $\pi+dfa_\theta$

where

$\Gamma:IDE\times EXP\to\Sigma$

$\Gamma[\![Ide,C]\!]=\varepsilon,\Gamma[\![Ide,V]\!]=[Ide/V],\Gamma[\![Ide,(E_1,\ldots,E_n)]\!]=\Gamma[\![Ide,E_1]\!]\&\ldots\&\Gamma[\![Ide,E_n]\!]$

$\Gamma[\![Ide,ctr(E_1,\ldots,E_n)]\!]=$同上,$\Gamma[\![Ide,[E_1|E_2]]\!]=\Gamma[\![Ide,E_1]\!]\&\Gamma[\![Ide,E_2]\!]$

利用函数 dfa 建立对象程序中所有函数参数的相关方程组.利用迭代法解方程组得出各函数参数抽象模式表.假设输入为 $g(ex_1,\ldots,ex_m,rx_1,\ldots,rx_n)$,$g$ 为目标函数,$ex_i(1\leqslant i\leqslant m)$已知,$rx_j(1\leqslant j\leqslant n)$未知.迭代之前,$F_{ai}(1\leqslant i\leqslant m)$ 的抽象值为 κ,而 $F_{aj}(m+1\leqslant j\leqslant m+n)$的抽象值为 υ,其它函数参数的抽象值均为 κ.

3.2　函数例化

本阶段通过生成对象程序中函数的变体来构造剩余程序.

输入:①代有部分已知参数的函数调用表达式;②对象程序;③抽象模式表.

输出:相对于对象程序的剩余程序.剩余程序中的函数是原对象程序中函数关于部分已知参数生成的变体.设原始函数 f 的形参为 $x_1,\ldots,x_m,y_1,\ldots,y_n$,其中 $x_i(1\leqslant i\leqslant m)$为可去参数,$y_j(1\leqslant j\leqslant n)$为剩余参数.当程序中有调用 $f(ev_1,\ldots,ev_m,rv_1,\ldots,rv_n)$时,剩余程序中该调用将变成 $f(ev_1,\ldots,ev_m,\_,\ldots,\_)(rv_1,\ldots,rv_n)$,其中 $f(ev_1,\ldots,ev_m,_,\ldots,_)$为 f 对应的剩余函数名,而 $rv_1,\ldots,rv_n$ 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均在参数表的前面.

函数 $T:EXP\to IDEA\to PROG\to STAT\to EXP\times IDEP,IDEA=IDE\to\Omega^n,IDEP=IDE\to PAT,STAT=VAR\to EXP,\gamma\in IDEA,\theta\in PROG,\eta\in IDEP,\zeta\in STAT$

本函数在给定一表达式时,根据 3 种环境状态信息生成剩余表达式和原始函数名到模式的映射表(原始函数名和模式将构成原始函数对应的剩余函数名).

$T[\![C]\!]_{\gamma\theta\zeta}=(C,\varepsilon)$

$T[\![V]\!]_{\gamma\theta\zeta}=(\zeta(V),\varepsilon)$

$T[\![(E_1,\ldots,E_n)]\!]_{\gamma\theta\zeta}=((EA_1,\ldots,EA_n),\eta)$

$T[\![ctr(E_1,\ldots,E_n)]\!]_{\gamma\theta\zeta}=(ctr(EA_1,\ldots,EA_n),\eta)$

$T[\![[E_1|E_2]]\!]_{\gamma\theta\zeta}=([EA_1|EA_2],\eta_1+\eta_2)$

$$T[\![E_1\ op\ E_2]\!]_{\gamma\theta\zeta}=\begin{cases}(\text{执行 } EA_1\ op\ EA_2,\eta),\forall_i(1\leqslant i\leqslant 2)X[\![EA_i]\!]=\kappa\\(EA_1\ op\ EA_2,\eta),\text{否则}\end{cases}$$

where $(EA_i,\eta_i)=T[\![E_i]\!]_{\gamma\theta\zeta}$, $\eta=\eta_1+\eta_2$

$X:EXP\to\Omega$

$X[\![C]\!]=\kappa, X[\![V]\!]=\upsilon, X[\![(E_1,\ldots,E_n)]\!]=X[\![E_1]\!]\wedge\ldots\wedge X[\![E_n]\!]$

$X[\![ctr(E_1,\ldots,E_n)]\!]=$同上, $X[\![[E_1|E_2]]\!]=X[\![E_1]\!]\wedge X[\![E_2]\!], X[\![other]\!]=\upsilon$

$$T[\![\text{if } E_1 \text{ then } E_2 \text{ else } E_3]\!]_{\gamma\theta\zeta}=\begin{cases}\begin{cases}(EA_2,\eta_2), EA_1=true\\ (EA_3,\eta_3), \text{else}\end{cases}, X[\![EA_1]\!]=\kappa\\ (\text{if } EA_1 \text{ then } EA_2 \text{ else } EA_3,\eta_1+\eta_2+\eta_3), \text{否则}\end{cases}$$

$$T[\![\text{case } E \text{ of } P_1:E_1\ldots P_n:E_n \text{ end}]\!]_{\gamma\theta\zeta}=\begin{cases}T[\![E_1]\!]_{\gamma\theta\zeta'}, X[\![EA]\!]=\kappa\\ (\text{case } EA \text{ of } P_1: EA_1\ldots P_n:EA_n \text{ end},\eta), \text{否则}\end{cases}$$

where $(EA,\eta_0)=T[\![E]\!]_{\gamma\theta\zeta}, \eta=\eta_0+\eta_1+\ldots+\eta_n, (I,\zeta')=cmat[\![EA,[P_1,\ldots,P_n]]\!]\zeta$

$$\triangle:STAT\times STAT\to STAT\ \zeta_1(V)\triangle\zeta_2(V)=\begin{cases}\zeta_2(V) & V\in domain(\zeta_2)\\ \zeta_1(V) & V\in domain(\zeta_1)\\ V & \text{else}\end{cases}$$

$cmat: PAT\times PAT\to STAT\to INT\times STAT$

$$cmat[\![P,[P_I|Ps]]\!]\zeta=\begin{cases}(I,\zeta\triangle mat_2[\![P_I,P]\!]), PI \text{ 与 } P \text{ 匹配}\\ cmat[\![P,Ps]\!], \text{否则}\end{cases}$$

$mat_2: PAT\times PAT\to STAT$ 该函数定义略去

$$T[\![\text{let } P_1=E_1\ldots P_n=E_n \text{ in } E]\!]_{\gamma\theta\zeta}=(\text{let } P_1'=EA_1'\ldots P_n'=EA_n' \text{ in } EA,\eta)$$

where $(EA_i,\eta_i)=T[\![E_i]\!]_{\gamma\theta\zeta_i}, \zeta'=\zeta\triangle\zeta_n, \zeta_0=\zeta, \zeta_i=\zeta_{i-1}\triangle mat_2[\![P_i,EA_i]\!]$

$(EA,\eta_0)=T[\![E]\!]_{\gamma\theta\zeta'}, \eta=\eta_0+\eta_1+\ldots+\eta_n, (P_i',EA_i')=\Phi[\![P_i,EA_i]\!], (1\leqslant i\leqslant n)$

$\Phi: PAT\times EXP\to PAT\times EXP$

$\Phi[\![P,E]\!]=$case (P,E) of

(C,C)	$:(\varepsilon,\varepsilon)$
(V,E)	:if $X[\![E]\!]=\kappa$ then $(\varepsilon,\varepsilon)$ else (V,E)
$((P_1,\ldots,P_n),(E_1,\ldots,E_n))$	$:((P_1',\ldots,P_n'),(E_1',\ldots,E_n'))$, where $(P_i',E_i')=\Phi[\![P_i,E_i]\!]$
$(ctr(P_1,\ldots,P_n),ctr(E_1,\ldots,E_n))$	:*same as above*
$([P_1\|P_2],[E_1\|E_2])$	$:([P_1'\|P_2'],[E_1'\|E_2'])$
default	$:(P,E)$

end

$$T[\![\text{letrec } P_1=E_1\ldots P_n=E_n \text{ in } E]\!]_{\gamma\theta\zeta}=(\text{letrec } P_1'=EA_1'\ldots P_n'=EA_n' \text{ in } EA,\eta)$$

where $(EA_i,\eta_i)=T[\![E_i]\!]_{\gamma\theta\zeta'}, \zeta'=\zeta\triangle mat_2[\![P_1,EA_1]\!]\triangle\ldots\triangle mat_2[\![P_n,EA_n]\!], (EA,\eta_0)=T[\![E]\!]_{\gamma\theta\zeta'}, \eta=\eta_0+\eta_1+\ldots+\eta_n (1\leqslant i\leqslant n), (P_i',EA_i')=\Phi[\![P_i,EA_i]\!]$, 此处仍需求关于 ζ' 的最小不动点

$$T[\![F(E_1,\ldots,E_n)]\!]_{\gamma\theta\zeta}=\begin{cases}rep[\![EA,\theta(F)]\!]_{\gamma\theta}, X[\![EA]\!]=\kappa\\ (FE,\eta[Scr/F]), \text{否则}\end{cases}$$

where $(EA,\eta)=T[\![(E_1,\ldots,E_n)]\!]_{\gamma\theta\zeta}, (FE,Scr)=rem[\![F,\gamma(F),EA]\!]$

$rep: EXP\times FDEF\to IDEA\to PROG\to EXP\times IDEP$

$$rep[\![EA,[F\ Pat=Exp|Def]]\!]_{\gamma\theta}=\begin{cases}T[\![Exp]\!]\gamma\theta mat_2[\![Pat,EA]\!], Pat \text{ 与 } EA \text{ 匹配}\\ rep[\![EA,Def]\!]_{\gamma\theta}, \text{否则}\end{cases}$$

$rem: IDE\times\Omega^n\times EXP\to EXP\times PAT$

$rem[\![F,M,EA]\!]=(F_{Scr}Pat,Scr)$

where $(Scr,Pat)=smat[\![EA,M]\!]$

$smat: EXP\times\Omega^n\to PAT\times PAT$

$smat[\![E,\kappa]\!]=(E,\varepsilon), smat[\![E,\upsilon]\!]=(\_,E),$

$smat[\![(E_1,\ldots,E_n),(A_1,\ldots,A_n)]\!]=((S_1,\ldots,S_n),(P_1,\ldots,P_n))$, where $(S_i,P_i)=smat[\![E_i,A_i]\!]$

$smat[\![ctr(E_1,\ldots,E_n),ctr(A_1,\ldots,A_n)]\!]=(ctr(S_1,\ldots,S_n),ctr(P_1,\ldots,P_n))$

函数 $fs:EXP\rightarrow IDEA\rightarrow PROG\rightarrow PROG$

该函数在给定对象程序、调用表达式和函数抽象模式时,生成对象程序的剩余程序.

$fs[\![F(E_1,\ldots,E_n)]\!]_{\gamma\theta}=$let $(Exp,\eta)=T[\![F(E_1,\ldots,E_n)]\!]_{\gamma\theta\epsilon}$ in $Exp\square A_{\gamma\theta\eta}$

$A_{\gamma\theta[Scr/F]\eta}=$ let $(Def,\eta')=B[\![\theta(F),Scr]\!]_{\gamma\theta}\ \eta''=\eta+\eta'$ in $Def\square A_{\gamma\theta\eta'}$

$B[\![[\ ],Scr]\!]_{\gamma\theta}=(\varepsilon,[\ ])$

$B[\![[F\ Pat=Exp\mid Def],Scr]\!]_{\gamma\theta}=$if $mat_2?[\![Pat,Scr]\!]$

then let $\zeta=mat_2[\![Pat,Scr]\!]\ (EA,\_)=T[\![Pat]\!]_{\gamma\theta\zeta}$

$(Scr,Pat')=smat[\![EA,\gamma(F)]\!]\ (Exp',\eta_1)=T[\![Exp]\!]_{\gamma\theta\zeta}$

$(Def',\eta_2)=B[\![Def,Scr]\!]_{\gamma\theta}$

in $(F_{Scr}\ Pat'=Exp'\square Def',\eta_1+\eta_2)$

else $B[\![Def,Scr]\!]_{\gamma\theta}$

此处符号"□"代表连接符.

3.3 优　化

函数例化是较复杂和繁琐的过程,限于篇幅,这里所给的例化算法只是一个基本框架.实际在实现 *FMIX* 系统时,我们在这方面作了大量的工作以力求达到最优,如函数的展开、模式的匹配、模式的化简、变量的深层取值和编译中的常规优化技术等.现只以函数的展开策略为例说明.函数例化后,许多生成的剩余函数体可能相当简单,经常只是对另一例化函数的调用,这样的调用可以用被调用函数的函数体来代替,这样可以减少函数和函数调用数.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测试方法来解决.如有递归函数 f,其可去形参为 $x_1,\ldots,x_m$,剩余形参为 $y_1,\ldots,y_n$.在函数例化中出现 f 的可去实参为 $e_1,\ldots,e_n$,如其中的某些参数(设为 $x_{i1},\ldots,x_{ij},0\leqslant ik\leqslant n,0\leqslant k\leqslant j\leqslant n$)的值是稳定的,即这些形参的实参值 $e_{i1},\ldots,e_{ij}$所组成的模式$(e_{i1},\ldots,e_{ij})$个数有限且这个数足够小,则 $x_{i1},\ldots,x_{ij}$为 f 新的可去形参,而其它 $x_k(k\neq ir,1\leqslant r\leqslant j)$连同 $y_1,\ldots,y_n$ 为 f 的剩余形参.我们的目标是求出具有这种特性参数的最大组合,尽管这一过程是比较复杂的,而且有时需要交互环境以确定最大允许的组合模式数,但实践证明,用这一方法解决递归函数的例化,其例化效果还是很好的.

4　程序例

现以一简单程序为例对我们的部分求值方法作进一步阐述.

$f(x,y)=(y,x+y)$

$s(x,y)=$let $(p,q)=y$ in $x*p+x*q$

$e(x,y,z,w)=$if $x\neq 0$ then if $y>z$ then $e(x-1,y-z,2*z,w+x)$

else $e(x-1,z-y,2*y,w-x)$

else w

$g(x_1,x_2,x_3,x_4)=$let $(t_1,t_2)=f(x_1,x_2)$ $con(t_3,t_4)=x_3$

$$\text{in } (s(t_3,t_4),s(x_2,x_4)-e(100*x_2,x_2+1,x_2,x_4))$$

经第1阶段的抽象分析,可得如下方程组:

$e_{a1}=e_{a1}\wedge g_{a2}, e_{a2}=e_{a2}\wedge e_{a3}\wedge g_{a2}, e_{a3}=e_{a2}\wedge e_{a3}\wedge g_{a3}, e_{a4}=e_{a1}\wedge e_{a4}\wedge g_{a4}$,

$f_{a1}=g_{a1}, f_{a2}=g_{a2}, s_{a1}=g_{a3}\wedge g_{a2}, s_{a2}=g_{a3}\wedge g_{a4}$

假设目标表达式为 $g(x,2,con(1,(0,1)),y)$,经迭代可得各函数参数的抽象模式表(见表1). 迭代之前函数 g 的参数初始抽象值为(υ/g_{a1}, κ/g_{a2}, κ/g_{a3}, υ/g_{a4}),而其它函数参数值均为 κ. 经函数例化阶段,可得如下剩余程序:

表1

f	(U,K)
s	(K,U)
e	(K,K,K,U)
g	(U,K,K,U)

$f_{(\_,2)}(x)=(2,x+2)$

$s_{con(1,\_)}(y)=\text{let } (p,q)=y \text{ in } 1*p+1*q$

$s_{con(2,\_)}(y)=\text{let } (p,q)=y \text{ in } 2*p+2*q$

$e_{(\_,3,2,\_)}(x,w)=\text{if } x\neq 0 \text{ then } e_{(\_,1,4,\_)}(x-1,w+x) \text{ else } w$

$e_{(\_,1,4,\_)}(x,w)=\text{if } x\neq 0 \text{ then } e_{(\_,3,2,\_)}(x-1,w-x) \text{ else } w$

$g_{(\_,2,con(1,(0,1)),\_)}(x_1,x_4)=\text{let } (t_1,t_2)=f_{(-,2)}(x_1) \text{ in } (s_{con(1,\_)}((0,1)),s_{con(2,\_)}(x_4)-e_{(\_,3,2,\_)}(200,x_4))$

递归函数 e 经参数稳定分析可知其已知参数的最大稳定组合为(e_{a2},e_{a3}),故其参数抽象模式改为($\upsilon,\kappa,\kappa,\upsilon$).

经优化,剩余程序变成如下:

$f(x)=x+2$

$s(y)=\text{let } (p,q)=y \text{ in } p+q$

$e_1(x,w)=\text{if } x\neq 0 \text{ then } e_2(x-1,w+x) \text{ else } w$

$e_2(x,w)=\text{if } x\neq 0 \text{ then } e_1(x-1,w-x) \text{ else } w$

$g(x_1,x_4)=\text{let } t_2=f(x_1) \text{ in } (1,s(x_4)-e_1(200,x_4))$

5 结 论

本文较详尽地讨论了函数式语言的部分求值技术,设计并实现了函数式语言的部分求值器 FMIX. 在抽象分析这一环节上,采用了信息流分析技术. 这一方法只需对程序进行一次扫描,避免了对程序的多次扫描,从而极大地提高了抽象分析的效率. 另外,用此方法对构造增量式(Incremental)部分求值器具有突出的优点. 出于描述上的方便,在抽象分析中函数参数的抽象域,这里只表示为 Ω. 事实上,FMIX 系统所采用的抽象域为 Ω',其中 $\Omega'=\Omega+(\Omega',\ldots,\Omega')+ctr(\Omega',\ldots,\Omega')$,即函数参数的抽象值可为结构值. 国外一些文章在讨论函数式语言的部分求值时对程序的例化描述甚少,尤其对函数的展开和递归函数的例化更是如此. 本文对函数例化这一部分进行了较深入细致的研究,对剩余函数的生成和展开运用了大量的优化算法,尤其对递归函数的处理采用了数据稳定测试法,从而相对地提高了剩余程序的时空效率.

本文的进一步工作是研究高阶函数的部分求值技术.

参考文献

1 Sestoft Peter. The structure of a self-applicable partial evaluator. Springer Verlag LNCS 217(1985), 1985. 236～256.

2 Meyer Uwe. Techniques for partial evaluation of imperative languages. SIGPLAN NOTICES. 1991,26(9):94～105.

3 Takano Akihiko. Generalized partial computation for a lazy functional language. SIGPLAN NOTICES, 1991,26(9):1～11.

4 Bondorf A. Towards a self-applicable partial evaluator for term rewriting system. In: D Bjrner, A P Ershov, N D Jones,eds., Partial Evaluation and Mixed Computation, North-Holland, 1988. 27～50.

5 Harper Robert, MacQueen David, Milner Robin. Standard ML Report. Computer Science Dept, Edinburgh University, 1985.

6 Johnsson, Lambda T. Lifting: transforming programs to recursive equations. Conference on Functional Programming Language and Computer Architecture, Nancy, LNCS 201, Springer Verlag, 1985. 190～203.

THE PARTIAL EVALUATION TECHNIQUE OF A FUNCTIONAL LANGUAGE

Song Litong　Jin Chengzhi

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

Abstract　Based on abstract interpretation technique, this paper designs and realizes a partial evaluator FMIX for a functional language. Compared with other similar partial evaluators of foreign countries, FMIX has its original means on realization, and uses efficient technique on some generally difficult problems.

Key words　Partial evaluation, abstract analysis, function specialization, residual program, eliminable parameter, residual parameters.