高阶函数式语言的部分求值器*

陆卫东 金成植

(吉林大学计算机科学系 长春 130061)

摘要 本文提出了一种新的基于 CLOSURE 分析的高阶函数式语言的部分求值方法,对表达式中的高阶值采取了有效的抽象分析手段,从而提高了剩余程序的质量.文中给出了 CLOSURE 分析、抽象分析、标记和例化较完整的形式描述.

关键词 部分求值,高阶函数,抽象分析.

部分求值是对程序优化的一种技术,近年来,由于它在软件自动生成方面的重要作用,特别是在编译器和编译器的生成器自动生成方面的特殊贡献[1~3],已引起了人们的广泛重视.高阶函数式语言的部分求值是部分求值领域的一大难题[4],其主要难点是:(1)在抽象分析时高阶值的抽象值如何确定;(2)在例化时如何产生新函数以形成合理且效率较高的剩余程序. Nielson[5]和 Mogenson[6]采用基于类型推导的抽象分析方法,用结构来表示高阶值的抽象值.他们的抽象分析过程过于复杂了. Bondorf[7]基于 CLOSURE 分析,将高阶值的抽象域定义在$\langle \perp, CL, D\rangle$上,简化了抽象分析算法.但此算法过于保守,得到的剩余程序的质量还不够高.为保证例化产生正确的剩余程序,当某一表达式的结果是高阶值,且这一高阶值不能静态确定时,则所有可能成为该表达式值的高阶函数,它们在原程序中的函数定义都必须保留在剩余程序中. Bondorf[7]为达此目的,采取了将这些应保留函数的每一个形参的抽象值强制设定为 D 的办法,很明显由此而得的剩余程序的质量并不很高.

本文的主要工作:①放宽了对象程序的限制,允许其输入的已知部分是高阶值;②进一步简化了抽象域,将其定义在$\langle S, D\rangle$上;③对必须保留在剩余程序中的原程序里的函数定义采取单独收集的方法,在抽象分析时不把这些函数的形参强设成 D,从而使例化更彻底,剩余程序的质量更高.文中对 CLOSURE 分析、抽象分析和例化算法都给出了较完整的形式化描述.

1 高阶函数式语言 $HOFL_0$

为讨论方便,本文定义了一个小的无类型的高阶函数式语言 $HOFL_0$.其抽象语法如:

$Prog::=\{f_0(x_{01},\ldots,x_0n_0)=L\_e_0; f_1(x_{11},\ldots,x_1n_1)=L\_e_1;\cdots; f_m(xm_1,\ldots,xmn_m)$

---

* 作者**陆卫东**,1966 年生,博士,主要研究领域为形式语义,部分求值,函数式语言及实现.**金成植**,1935 年生,教授,主要研究领域为程序理论与软件自动生成.

本文通讯联系人:陆卫东,北京 100080,中国科学院国家智能计算机研究开发中心

本文 1995-08-14 收到修改稿

$=L\_e_m\}$

$L\_e::=L\ e$ (L 是标号,具体文法中表达式可不带标号)

$e::=c$(常量)$|v$(变量)$|f_j$(函数名)$|$op $L_e_0\ L_e_1|$if $L_e_0\ L_e_1\ L_e_2|$Let $v=L\_e_0$ in $L\_e_1|L\_e_0(L\_e_1,\ldots,L\_e_n)$(函数应用)

在表达式中不包括 λ—抽象,λ—抽象可以经过提升转换成函数形式.[10]高阶是通过函数名可以作为表达式的值来体现的,另外在语义上有如下的限制:函数调用必须满参;基本操作(op)只作用于一阶值并且结果也是一阶值;目标函数设定为 f_0.

2 CLOSURE 分析

在一阶函数式语言程序中,从文本上看,函数调用是显式的,即形如 $g(e_1\ldots e_n)$,其中 g 一定是程序中一个确定的函数;而在高阶函数式语言中,由于函数的参数可以是函数,函数调用是隐式的,即形如 $e_0(e_1,\ldots,e_n)$,其中 e_0 可以是一个静态确定的函数 g,也可以不是.抽象分析是对程序中每一函数的每一参数寻求收敛的抽象解,因此针对这种隐式调用的现象,必须在抽象分析之前加进 CLOSURE 分析,用于确定程序中每一函数调用点上被调用的函数可能是哪些.

预定义域和函数:

$Label$:表达式标号集; $FunNameSet=\{f_0,f_1,\cdots\}$:函数名集; $Index=\{1,2,\ldots\}$;

$ClSet=P(FunNameSet)$; $\sigma,\beta\in ClMap=Label\to ClSet$; $\varepsilon\in ClEnv=Var\to ClSet$;

$\llbracket\ \rrbracket: Var+FunNameSet\to FunNameSet\times Index+Label+Label\times Index$

$$\llbracket f\rrbracket=L \qquad \text{if } (f(x_0\ldots x_n)=Le)\in Prog \tag{1}$$

$$\llbracket v\rrbracket=(f,i) \qquad \text{if } v=x_i \text{ and } (f(x_0\ldots x_i\ldots x_n)=Le)\in Prog \tag{2}$$

$$\llbracket v\rrbracket=(L_1,1) \qquad \text{if } v=x \text{ and } (L \text{ let } x=L_0e_0 \text{ in } L_1e_1)\in Prog \tag{3}$$

(2)、(3)式分别重新命名形参名和局部变量名以避免在环境 $ClMap$ 中变量名字的冲突.$\cup$是扩充的集合运算:

$$(\sigma,\beta)\cup(\sigma',\beta')=(\sigma\cup\sigma',\beta\cup\beta');\ \sigma\cup\sigma'=\lambda k.\,\sigma(k)\cup\sigma'(k)$$

$upd: A\times B\times(A\to B)\to(A\to B)$　$upd(k,c,\sigma)=\sigma\cup[k\to c]\perp_{Map}$环境更改函数.

CLOSURE 分析:

$CL: Prog\to ClEnv\to ClMap\times ClEnv\times ClMap$

$CL\llbracket f_0(x_{01},\ldots,x_{0n_1})=L_1e_1;\cdots;f_m(x_{m1},\ldots x_{mn_m})=L_me_m\rrbracket\varepsilon_{ini}$

$=\mathrm{fix}(\lambda(\sigma,\varepsilon,\beta).(\bigcup_i(cl\llbracket L_ie_i\rrbracket\sigma\varepsilon\beta)\cup(\sigma_{ini},\varepsilon_{ini},\beta_{ini}))$

where ε_{ini}由对象程序的输入确定;$\sigma_{ini}=\perp_{ClMap}$;$\beta_{ini}=\perp_{ClMap}$

$cl: L\_e\to ClMap\to ClEnv\to ClMap\to(ClMap\times ClEnv\times ClMap)$

$cl\llbracket Lc\rrbracket\sigma\ \varepsilon\ \beta=(\sigma,\varepsilon,\beta)$

$cl\llbracket Lx\rrbracket\sigma\ \varepsilon\ \beta=(upd(L,\varepsilon(\llbracket x\rrbracket),\sigma),\varepsilon,\beta)$

$cl\llbracket Lf\rrbracket\sigma\ \varepsilon\ \beta=(upd(L,\{f\},\sigma),\varepsilon,upd(L,\{f\},\beta))$

$cl\llbracket L \text{ if } L_0e_0\ L_1e_1\ L_2e_2\rrbracket\sigma\ \varepsilon\ \beta=\text{let } (\sigma',\varepsilon',\beta')=\bigcup_i(cl\llbracket L_ie_i\rrbracket\sigma\ \varepsilon\ \beta)$

$\text{in}(upd(L,\sigma'(L_1)\cup\sigma'(L_2),\sigma'),\varepsilon',upd(L,\bigcup_i\beta'(L_i),\beta'))$

$cl\llbracket L \text{ op } L_0e_0\ L_1e_1\rrbracket\sigma\ \varepsilon\ \beta$=let $(\sigma',\varepsilon',\beta')=\bigcup_i(cl\llbracket L_ie_i\rrbracket\sigma\ \varepsilon\ \beta)$
in $(\sigma',\varepsilon',upd(L,\beta'(L_0)\cup\beta'(L_1),\beta'))$
$cl\llbracket L \text{ let } v=L_0e_0 \text{ in } L_1e_1\rrbracket\sigma\ \varepsilon\ \beta$=let $(\sigma',\varepsilon',\beta')=\bigcup_i(cl\llbracket L_ie_i\rrbracket\sigma\ \varepsilon\ \beta)$
in $(upd(L,\sigma'(L_1),\sigma'),\ upd(\llbracket v\rrbracket,\sigma'(L_0),\varepsilon'),\ upd(L,\beta'(L_0)\cup\beta'(L_1),\beta'))$
$cl\llbracket L\ L_0e_0(L_1e_1\ldots L_ne_n)\rrbracket\sigma\ \varepsilon\ \beta$=let $(\sigma',\varepsilon',\beta')=\bigcup_i(cl\llbracket L_ie_i\rrbracket\sigma\ \varepsilon\ \beta)$
and $\sigma''=upd(L,\ \bigcup_{f\in\sigma'(L_0)}\sigma'(\llbracket f\rrbracket)\ ,\sigma')$ and $\beta''=upd(L,\bigcup_{i\beta'}(L_i),\beta')$
and $\varepsilon''=\bigcup_{(f\in\sigma'(L_0),1\leqslant i\leqslant n)}upd((f,i),\sigma'(L_i),\varepsilon')$ in $(\sigma'',\varepsilon'',\beta'')$

以上算法计算的结果:$\sigma(L)$确定了表达式 L 可能的高阶值;$\varepsilon(x)$确定了形参 x 可能受约的高阶值;$\beta(L)$确定了表达式 L 中直接用到的函数.

3 抽象分析

抽象分析是将表达式的值域解释为某一抽象域,在抽象环境下解释执行对象程序,以获得一个收敛的抽象环境,该抽象环境确定了程序中的每一变量是静态值还是动态值.在本文定义的高阶函数式语言 $HOFL_0$ 中,表达式的值有 2 类:一类是一阶值,另一类是高阶值.经过 CLOSURE 分析后,对每一函数调用都能估计出可能被调用的函数是哪些,因而可将表达式的抽象域统一定义在$\{S,D\}$上,其中 S 是静态值的抽象表示,D 是动态值的抽象表示.

以下说明在抽象分析过程中用到的域和变量:

(1)抽象值域:$BtValue=\{S,D\}$;域 $BtValue$ 上的关系$<$:$S<D$

(2)剩余程序中需要保留的原程序中函数的函数名集 $\tau\in RFunSet=P(FunNameSet)$

(3)抽象环境 $\mu\in BtMap=Label\to BtValue$,$\mu(L)$确定标号为 L 的表达式的抽象值.

(4)抽象环境 $\rho\in BtEnv=Var\to BtValue$,$\rho(\llbracket x\rrbracket)$确定变量 x 的抽象值;初始抽象环境 ρ_{ini} 由部分输入的抽象解释得到.

(5)$\sqcup$ 是扩充的求抽象值最小上界的运算:

$S\sqcup S=S$; $S\sqcup D=D$; $D\sqcup S=D$; $D\sqcup D=D$;

$(a,b,c)\sqcup(a',b',c')=(a\sqcup a',b\sqcup b',c\sqcup c')$;

$\mu\sqcup\mu'=\lambda l.\ \mu(l)\sqcup\mu'(l)$; $\rho\sqcup\rho'=\lambda x.\ \rho(x)\sqcup\rho'(x)$;

(6)σ,ε,β 是 CLOSURE 分析的结果.

抽象分析:

$bt:Prog\to BtEnv\to BtMap\times BtEnv\times RFunSet$

$bt\llbracket f_0(x_{01},\ldots,x_{0n_1})=L_0e_0;\cdots;f_m(x_{m1},\ldots x_{mnm})=L_{mem}\rrbracket\rho_{ini}$
$=\text{fix}(\lambda(\mu,\rho,\tau).(\sqcup_i F\llbracket L_ie_i\rrbracket\mu\rho\tau)\sqcup(\perp\mu,\rho_{ini},\sigma(L_0)))$

$F:Exp\to BtEnv\to BtMap\to RFunSet\to BtEnv\times BtEnv\times RFunSet$

$F\llbracket L\ c\rrbracket\mu\ \rho\ \tau=(\mu\sqcup[L\to S],\rho,\tau)$

$F\llbracket L\ x\rrbracket\mu\ \rho\ \tau$=let τ'=if $\rho(\llbracket x\rrbracket)=D$ then $\varepsilon(\llbracket x\rrbracket)\cup\tau$ else τ
in $(\mu\sqcup[L\to\rho(\llbracket x\rrbracket)],\rho,\tau')$

$F\llbracket L\ f\rrbracket\mu\ \rho\ \tau=(\mu\sqcup[L\to S],\rho,\tau)$

$F\llbracket L \text{ if } L_0e_0\ L_1e_1\ L_2e_2\rrbracket\mu\ \rho\ \tau$=let $(\mu',\rho',\tau')=\sqcup_iF\llbracket e_i\rrbracket\mu\ \rho\ \tau$ and $b_i=\mu'(L_i)$

$$\text{in if } b_0=D \text{ then } (\mu' \amalg [L\to D],\ \rho',\tau' \cup \sigma(L))$$
$$\text{else } (\mu' \amalg [L\to b_1 \amalg b_2],\rho',\tau')$$

$$F\llbracket L \text{ op } L_0e_0\ L_1e_1\rrbracket \mu\ \rho\ \tau = \text{let}(\mu',\rho',\tau')=\amalg_i F\llbracket L_ie_i\rrbracket \mu\ \rho\ \tau$$
$$\text{and } b_i=\mu'(L_i) \quad \text{in } (\mu' \amalg [L\to b_1 \amalg b_2],\rho',\tau')$$

$$F\llbracket L \text{ let } v=L_0e_0 \text{ in } L_1e_1\rrbracket \mu\rho\tau = \text{let } (\mu',\rho',\tau')=\amalg_i F\llbracket L_ie_i\rrbracket \mu\rho\tau$$
$$\text{in if } \mu'(L_0)=D \text{ then } (\mu' \amalg [L\to D],\rho' \amalg [\llbracket v\rrbracket \to D],\tau' \cup \sigma(L_0) \cup \sigma(L_1))$$
$$\text{else } (\mu' \amalg [L\to \mu'(L_1)],\rho' \amalg [\llbracket v\rrbracket \to S],\tau')$$

$$F\llbracket L\ L_0e_0(L_1e_1 \ldots L_ne_n)\rrbracket \mu\ \rho\ \tau = \text{let } (\mu',\rho',\tau')=\amalg_i F\llbracket L_ie_i\rrbracket \mu\ \rho\ \tau \text{ for } 0\leqslant i\leqslant n$$
$$\text{and } b_i=\mu'(L_i) \text{ in if } b_0=D \text{ then } (\mu' \amalg [L\to D],\rho',\tau' \cup (\cup_i \sigma(L_i)))$$
$$\text{else let } \rho''=\rho' \amalg (\amalg_{f\in\sigma(L_0),1\leqslant i\leqslant n}[(f,i)\to b_i])$$
$$\text{and } \tau''=\cup_{f\in\sigma(L_0),1\leqslant i\leqslant n}(\text{if } \rho(f,i)=D \text{ then } \sigma(L_i) \text{ else } \{\})$$
$$\text{in } (\mu' \amalg [L\to \amalg_i b_i],\rho'',\tau' \cup \tau'')$$

$$rfun: RFunSet \to ClFun \to RFunSet$$
$$rfun(\tau,\beta)=\text{fix}(\lambda\Omega.\ \cup_{f\in\Omega}(\beta(\llbracket f\rrbracket) \cup \Omega \cup \tau))$$

抽象分析的结果：$\mu(L)$确定了表达式 L 在部分求值时是静态的还是动态的；$\rho(x)$确定了变量 x 对应的实参是静态的还是动态的；τ 收集了剩余程序中需要保留的原程序中函数的函数名，Ω 利用函数 $rfun$ 收集了剩余程序中所有必须保留的原程序中函数的函数名.

4　例化形式描述

例化是在部分输入已知的情况下，执行那些只依赖于静态值的操作；而将那些有可能依赖于动态值的操作保留下来以组成剩余程序. 为使例化过程简单明了，通常根据抽象环境将原程序转换为用二级语言表示的形式，即标记过程.[1]标记过程的主要功能有：(1)将函数定义中的形参表根据抽象环境 ρ 分解成静态参数表和动态参数表 2 部分；并对函数调用的形式作相应的改动. 即 $f_i(x_{i1},\ldots,x_{ini})=L_ie_i$ 转换为 $f_i(x_{s1},\ldots,x_{sm})(x_{d1},\ldots,x_{dk})=e_i^{ann}$，其中 $\forall\ 1\leqslant j\leqslant m\ \rho(\llbracket x_{sj}\rrbracket)=S$ 且 $\forall\ 1\leqslant j\leqslant k\ \rho(\llbracket x_{dj}\rrbracket)=D$. (2)将表达式 e 根据抽象环境 μ 转换为 e^{ann}. 标记后可将表达式的标号去掉；表达式中的动态操作加下划线；静态操作不加下划线；对动态表达式中的静态子表达式 e 加前缀 lift 表示静态值需保留在剩余程序中.

例化算法以标记程序 p^{ann} 和目标函数 f_0 的静态实参值表 vs 为输入，输出 p 关于 vs 的剩余程序 π. 其中 $Value=BasicVal$(基本值域)$+FunVal$(函数值域)；$Rexp=Value+Code$(表达式的文本)；组成剩余程序的函数定义的集合 $\pi\in RProg$；需例化的函数的集合 $\eta\in Pending$；已例化的函数的集合 $\zeta\in Pending$；$build\_no: Value^*\to Code$ 根据 no 的不同将值序列转换成文本形式，$no\in NODE=\{\text{constant},\text{let},\text{app},\text{op},\text{if}\}$.

例化算法：

$$fsp: Ann\_Prog\to Value^*\to RProg$$
$$fsp\llbracket p^{ann}\rrbracket vs_0=sp(\{\langle f_0,vs_0\rangle\},p^{ann},\{\},\{\})$$
$$sp: Pending\times Ann\_Prog\times Pending\times RProg\to RProg$$
$$sp\ (\eta,p^{ann},\zeta,\pi)=\text{if } \eta=\{\} \text{ then } \pi$$

else let$(f_i,vs_i)\in\eta$

and $(f_i(x_{s1},\ldots,x_{sm})(x_{d1},\ldots,x_{dk})=e_i^{ann})\in p^{ann}$ and $(v_1,\ldots,v_m)=vsi$

and $(\eta',re)=RE\llbracket e_i^{ann}\rrbracket(v_1,\ldots,v_m,x_{d1},\ldots,x_{dk})(x_{s1},\ldots,x_{sm},x_{d1},\ldots,x_{dk})p^{ann}$

and $\zeta'=\zeta\cup\{\langle f_i,v_{si}\rangle\}$ and $\eta''=\eta\cup\eta'\setminus\zeta'$

in $sp(\eta'',p^{ann},\zeta',\pi\cup\{(\langle f_i,v_s\rangle(x_{d1},\ldots,x_{dk})=re)\})$

$RE:Ann\_Exp\rightarrow(Value+Var)^*\rightarrow Var^*\rightarrow Ann\_Prog\rightarrow Pending\times RExp$

$RE\llbracket c\rrbracket v_s\ x_s\ p^{ann}=(\{\},c)$

$RE\llbracket$lift $e\rrbracket v_s\ x_s\ p^{ann}=let(\eta,re)=RE\llbracket e\rrbracket v_s\ x_s\ p^{ann}$ in $(\eta,$build_constant$(re))$

$RE\llbracket x_j\rrbracket v_s\ x_s\ p^{ann}=let(v_1,\ldots,v_j,\ldots,v_k)=v_s$ and $(x_1,\ldots,x_j,\ldots,x_k)=x_s$ in v_j

$RE\llbracket f_i\rrbracket v_s\ x_s\ p^{ann}=(\{\},f_i)$

$RE\llbracket$if $e_0\ e_1\ e_2\rrbracket v_s\ x_s\ p^{ann}=let(\eta_i,re_i)=RE\llbracket e_i\rrbracket v_s\ x_s\ p^{ann}$ for $i=0,1,2$

in if $re_0=$true then $(\bigcup_i\eta_i,re_1)$ else $(\bigcup_i\eta_i,re_2)$

$RE\llbracket$if $e_0\ e_1\ e_2\rrbracket v_s\ x_s\ p^{ann}=$let $(\eta_i,re_i)=RE\llbracket e_i\rrbracket v_s\ x_s\ p^{ann}$ for $i=0,1,2$

in $(\bigcup_i\eta_i,$build_if$(re_0,re_1,re_2))$

$RE\llbracket$op $e_0\ e_1\rrbracket v_s\ x_s\ p^{ann}=$let $(\eta_i,v_i)=RE\llbracket e_i\rrbracket v_s\ x_s\ p^{ann}$ for $i=0,1$

in $(\bigcup_i\eta_i,$op $v_0\ v_1)$

$RE\llbracket$op $e_0\ e_1\rrbracket v_s\ x_s\ p^{ann}=$let $(\eta_i,re_i)=RE\llbracket e_i\rrbracket v_s\ x_s\ p^{ann}$ for $i=0,1$

in $(\bigcup_i\eta_i,$build_op$(re_0,re_1))$

$RE\llbracket$let $x=e_0$ in $e_1\rrbracket v_s\ x_s\ p^{ann}=$let $(\eta,v_0)=RE\llbracket e_0\rrbracket v_s\ x_s\ p^{ann}$

and $(v_1,\ldots,v_j,\ldots,v_k)=v_s$ and $(x_1,\ldots,x_j,\ldots,x_k)=x_s$

and $v_s'=(v_0,v_1,\ldots,v_j,\ldots,v_k)$ and $x_s'=(x,x_1,\ldots,x_j,\ldots,x_k)$

in $RE\llbracket e_1\rrbracket v_s'\ x_s'\ p^{ann}$

$RE\llbracket$let $x=e_0$ in $e_1\rrbracket v_s\ x_s\ p^{ann}=$let $(\eta,re_0)=RE\llbracket e_0\rrbracket v_s\ x_s\ p^{ann}$

and $(v_1,\ldots,v_j,\ldots,v_k)=v_s$ and $(x_1,\ldots,x_j,\ldots,x_k)=x_s$

and $v_s'=(x,v_1,\ldots,v_j,\ldots,v_k)$ and $x_s'=(x,x_1,\ldots,x_j,\ldots,x_k)$

and $(\eta',re_1)=RE\llbracket e_1\rrbracket v_s'\ x_s'\ p^{ann}$

in $(\eta\cup\eta',$build_let$(x,re_0,re_1))$

$RE\llbracket e_0\ (e_1\ldots e_m)(e_{m+1}\ldots e_{m+k})\rrbracket v_s\ x_s\ p^{ann}=$

let $(\eta_i,re_i)=RE\llbracket e_i\rrbracket v_s\ x_s\ p^{ann}$ for $i=0,1,\ldots,n$

and $re_0=f_j$ and $(f_j(x_{s1},\ldots,x_{sm})(x_{d1},\ldots,x_{dk})=e_j^{ann})\in p^{ann}$

and $v_s'=(re_1,\ldots re_{m+k})$ and $x_s'=(x_{s1},\ldots,x_{sm},x_{d1},\ldots,x_{dk})$

and $(\eta',re')=RE\llbracket e_j^{ann}\rrbracket v_s'\ x_s'\ p^{ann}$

in $(\eta'\cup(\bigcup_i\eta_i)\ ,\ re')$

$RE\llbracket e_0\ (\ e_1\ldots e_m)(e_{m+1}\ldots e_{m+k})\rrbracket v_s\ x_s\ p^{ann}=$

let $(\eta_i,re_i)=RE\llbracket e_i\rrbracket v_s\ x_s\ p^{ann}$ for $i=0,1,\ldots,n$

in if $re_0=f_j$ then

let $(f_j(x_{s1},\ldots,x_{sm})(x_{d1},\ldots,x_{dk})=e_j^{ann})\in p^{ann}$

and $code$=build_app($\langle f_j,(re_1,\ldots,re_m)\rangle,(re_{m+1},\ldots,re_{m+k})$)

in ($\{\langle f_j,(re_1,\ldots,re_m)\rangle\}\cup(\bigcup_i\eta_i),code$)

else let $code$=build_app($re_0,(re_1,\ldots,re_m,re_{m+1},\ldots,re_{m+k})$)

in ($\bigcup_i\eta_i,code$)

由 fsp 得到的剩余程序 π 并不是最终的结果，它还应该加上由 $rfun$ 收集的剩余函数集 Ω. 因为在抽象分析时，对标记为 D 的高阶函数是单独处理的，具体由 $union$ 处理.

$union:Prog\times RProg\times RFunset\rightarrow Rprog$

$union\ (p,\pi,\Omega)$=if $\Omega=\{\}$ then π

else let $f(x_1\ldots x_n)=e\in p$ and $\Omega'=\Omega\backslash\{f\}$

in if $f(x_1\ldots x_n)=e\in\pi$ then $union(p,\pi,\Omega')$

else $union(p,\{f(x_1\ldots x_n)=e\}\cup\pi,\Omega')$

例如程序 p：

$f_0(x_0,y_0,z_0)=f_1(x_0,y_0+1,z_0+1)+f_2(x_0,y_0-1,z_0-1)+f_1(f_4,y_0,z_0)$；

$f_1(x_1,y_1,z_1)=$(if $y_1>0\ x_1\ f_3$) (y_1+5,z_1)；$f_3(x_3,y_3)=f_5(x_3)+y_3$；

$f_2(x_2,y_2,z_2)=$(if $z_2>0\ x_2\ f_4$) (y_2,z_2)；$f_4(x_4,y_4)=f_5(x_4)-y_4$；

$f_6(x_6,y_6)$=let $v=f_5(x_6)$ in $f_5(v)+y_6$；$f_5(x_5)=6^*x$；

假设目标函数 f_0 的已知参数为：$x_0=f_6,y_0=1$，(z_0 未知)，对其部分求值，生成的剩余程序为：

$f_0\langle f_{6,1},\perp\rangle(z_0)=f_1\langle f_{6,2},\perp\rangle(z_0+1)+f_2\langle f_{6,0},\perp\rangle(z_0-1)+f_1\langle f_{4,1},\perp\rangle(z_0)$；

$f_1\langle f_{6,2},\perp\rangle(z_1)=f_6\langle 7,\perp\rangle(z_1)$；$f_1\langle f_{4,1},\perp\rangle(z_1)=f_4\langle 6,\perp\rangle(z_1)$；

$f_2\langle f_{6,0},\perp\rangle(z_2)=$(if $z_2>0\ f_6\ f_4$) $(0,z_2)$；$f_5(x_5)=6^*x$；

$f_4\langle 6,\perp\rangle(y_4)=36-y_4$；$f_4(x_4,y_4)=f_5(x_4)-y_4$；

$f_6(x_6,y_6)$=let $v=f_5(x_6)$ in $f_5(v)+y_6$；$f_6\langle 7,\perp\rangle(y_6)=252+y_6$；

若按 Bondorf[7]的方法，最终生成的剩余程序中不包括 $f_4\langle 6,\perp\rangle$和 $f_6\langle 7,\perp\rangle$. 在上述例化算法中，为了防止函数调用不终止，所以只在实参都是已知时函数调用才展开. 例化后生成的剩余程序可以通过后处理对可以展开的函数进一步展开，后处理中函数的展开策略在文献[8]中有详细表述.

5　结　论

根据本文提出的方法，部分求值器最终产生的剩余程序的质量比已有方法得到的提高了. 该方法也可直接用于函数调用允许是非满参的情况，但由于没有充分利用非满参的参数信息，剩余程序还有优化的余地. 因此，进一步的工作是针对允许非满参函数调用的函数式语言，寻找更高级的策略，使例化后的代码更有效.

参考文献

1　Jone N D, Sestoft P, Sondergard H. Mix: a self-applicable partial evaluator for experiments in compiler generation. International Journal LISP and Symbolic Computation, 1987, **2**(1): 9～50.

2　Jone N D, Sestoft P, Sondergard H. An experiment in partial evaluation: the generation of a compiler generator.

LNCS202, 1985. 124～140.

3 Futamura Y. Partial evaluation of computing process—an appoarch to a compiler—compiler. System, Computer, Controls, 1971, **2(5)**: 45～50.

4 Jone N D. Challenging problems in partial evaluation and mixed computation. Proceeding of Workship on Partial Evaluation and Mixed Computation, 1988. 1～25.

5 Nieson. Automatic binding time analysis for a typed λ—calculus. Science of Computer Programming, 1988, **1(4)**: 459～494.

6 Torben. Mogensen: binding time analysis for polymorphically typed higher order language. LNCS352, 1989. 298～312.

7 Bondorf A. Automatic autoprojection of higher order recursive equations. Science of Computer Programming, 1991, **17**: 3～34.

8 Sestoft P. Automatic call unfolding in a partial evaluate. Proceeding of Workship on Partial Evaluation and Mixed Computation, 1988. 485～506.

THE PARTIAL EVALUATOR FOR HIGH ORDER FUNCTIONAL LANGUAGE

Lu Weidong Jin Chengzhi

(*Department of Computer Science Jilin University Changchun* 130061)

Abstract This paper develops a new method to construct a partial evaluator for high order functional language. This method is based on the CLOSURE analysis. With the analysis, the authors purpose an efficient binding time analysis so that the quality of the residual code produced by the specializatoin is improved. In this paper, all of the phases in the partial evaluator are described formly.

Key words Partial evaluator, high order function, abstract analysis.