对象式 Lambda 演算的自作用部分计值*

王明文, 孙永强

(上海交通大学 计算机科学与工程系,上海 200030)

E-mail: sun-yq@cs.sjtu.edu.cn

http://www.sjtu.edu.cn

摘要: 讨论了一个对象式 Lambda 演算的部分计值器. 对象式 Lambda 演算在 Lambda 演算的基础上添加了对象机制. 部分计值器的构造是采用传统的三步法,首先定义对象式 Lambda 演算的元解释器;然后提出对象式 Lambda 演算的约束时间分析方法(binding-time analysis),约束时间分析决定哪些计算可以在编译时完成,哪些计算需留在运行时执行;最后定义部分计值器. 同时,给出了元解释器和部分计值器的正确性证明.

关键词: 部分计值;对象式 Lambda 演算;约束时间分析

中图法分类号: TP301 **文献标识码**: A

部分计值是一种程序转换技术. 在给定部分参数输入的情况下,对程序进行转换约简,完成尽可能多的运算而产生剩余程序(residual program). 当剩余程序作用在余下的参数上时,应产生与原程序作用在全部参数上时一样的结果. 部分计值在研究初期是被作为优化工具在编译阶段对可完成部分展开,以产生优化代码. 在 20 世纪 70 年代,日本的 Futamura[1]以及前苏联的 Turchin[2]和 Ershov[3]分别独立指出:部分计值器可以生成语言的编译器. 我们知道解释器远比编译器更易于实现,编译器的实现者要考虑两个约束时间(编译时的程序输入和运行时的参数输入),而解释器只需考虑一个(运行时的程序输入和参数输入). 因此,通过部分计值器,我们只要给出用语义方法描述的语言的解释器,就能高效地、自动地得到语言的编译器.

自 20 世纪 80 年代中期以来,部分计值技术的研究在国外得到较广泛的展开,但据我们所知,绝大部分工作是基于函数式语言和过程式语言的,也有少数工作研究并行语言的部分计值. 对于面向对象语言还没有文献表明有过研究,主要是由于目前对对象式语言还没有一个较为成熟的抽象机制. 由于对象式语言已成为当今程序设计的主流语言,因此我们希望能给出一个一般方法来高效地实现对象式语言的编译器. 我们采用的方法是部分计值技术. 本文将讨论对象式 Lambda 演算应用部分计值技术的可能性,并给出其形式化的描述.

本文采用大家所熟知的开发 Lambda 演算部分计值器的三步法[4~7]:① 定义对象式 Lambda 演算的元解释器,并且证明其正确性;② 提出对象式 Lambda 演算的约束时间分析方法(binding-time analysis,简称 BTA),约束时间分析决定哪些计算可以在编译时完成,哪些计算需留在运行时执行,为了证明部分计值器的正确性,我们引入了合式项(well-formed-term)这一概念;③ 定义了部分计值器(例化程序),并且证明了部分计值器的正确性.

* **收稿日期**: 1999-06-21; **修改日期**: 2000-05-10

基金项目: 国家 863 高科技发展规划资助项目(863-306-05-04-3)

作者简介: 王明文(1965--),男,江西南昌人,博士,教授,主要研究领域为程序理论,并行计算;孙永强(1937--),男,浙江嘉善人,教授,博士生导师,主要研究领域为计算机软件.

1 基本定义和术语

为了描述部分计值器,我们需要引入一些基本定义和术语.

定义 1. 程序语言 L 由三元组 $\langle P,\Delta,R\rangle$ 所构成,其中 P 是 L-程序集,Δ 是 L-数据集,R 是由 R_n-$(n+2)$元关系$(n=0,1,2,\ldots)$所构成的关系簇,且满足:若

$$(\rho,d_1,\ldots,d_n,d)\in R_n,\quad \text{则有 } I_n(\rho,d_1,\ldots,d_n)=d.$$

这里,$\rho\in P,d_1,\ldots,d_n,d\in\Delta$,$I_n$ 是一个部分函数,也即,对 $n\geqslant 0$,$(\rho,d_1,\ldots,d_n,d)\in R_n\Leftrightarrow$程序 ρ 作用在数据 $d_1,\ldots,d_n$ 上产生的输出是 d.

L 语言解释器是作用在两种数据上的一个程序,这两种数据分别是:要被执行的程序 ρ 和程序 ρ 在执行过程中要用的数据 $d_1,\ldots,d_n$. L 语言的自解释器是解释程序,本身也是 L-程序.

定义 2. 一个 L-程序的自解释器 $Eval\in P$,若存在一个代码函数$\ulcorner\cdot\urcorner:P\to\Delta$,满足:

$$(Eval,\ulcorner\rho\urcorner,d_1,\ldots,d_n,d)\in R_{n+1}\Leftrightarrow(\rho,d_1,\ldots,d_n,d)\in R_n. \tag{1}$$

式(1)表明,程序 ρ 作为自解释器的数据所产生的输出是由程序 ρ 的语义确定的.

部分计值器 Pev 是把程序 ρ 和已知数据部分进行例化的程序,注意到这种例化可以对需要的输入数据逐个地进行,因此,程序 ρ 作用的数据可分成静态和动态两部分.不失一般性,在以后的讨论中,我们只考虑输入数据为静态数据和动态数据这两种情况.为了确定程序的静态部分,我们用的方法是约束时间分析(BTA)方法,也就是构造函数 $\Phi:P\to P_a$,将 $\Phi(\rho)\in P_a$ 称作程序 ρ 的注释版(annotated version),而 $\Phi(\rho)$ 也必须转换成 L-数据才能被部分计值作用.

定义 3. 一个自作用部分计值是一个程序 $Pev\in P$,满足:若存在一个约束时间分析函数 $\Phi:P\to P_a$ 和代码生成函数$\lfloor\cdot\rfloor:P_a\to\Delta$,使得

$$(Pev,\lfloor\Phi(\rho)\rfloor,d_1,\rho')\in R_2\Rightarrow(Eval,\rho',d_2,d)\in R_2. \tag{2}$$

这里,ρ' 满足$(\rho',d_2)=(\rho,d_1,d_2)$,$\rho'$ 是经部分计值后所得到的剩余程序.

结合式(1)和式(2)可得:

$$(Pev,\lfloor\Phi(\rho)\rfloor,d_1,\rho')\in R_2\Rightarrow(\rho,d_1,d_2,d)\in R_2.$$

在本文中,我们讨论的程序语言是对象式 Lambda 演算,即 $L=\lambda_{obj+}$,且考虑离线(off-line)部分计值器.对象式 Lambda 演算是在无类型 Lambda 演算的基础上增加了对象语法形式,其抽象语法表示如下[8~12]:

$$e::=x\mid\lambda_x.e\mid e_1e_2\mid e\Leftarrow m\mid\langle e_1\leftarrow m=e_2\rangle,$$

这里,$e\Leftarrow m$ 表示送一个消息 m 到对象 e 所得到的结果;$\langle e_1\leftarrow m=e_2\rangle$ 表示由 e_2 定义方法名 m 的方法体,把方法 m 扩充或重载到 e_1 中.

为方便起见,记$\langle m_1-e_1,\ldots,m_k-e_k\rangle\equiv\langle\ldots\langle\langle\rangle\leftarrow m_1=e_1\rangle\ldots\leftarrow m_k=e_k\rangle$,$m_1,\ldots,m_k$ 为互不相同的方法名,当 $k=0$ 时,为空对象.其操作语义为

$$(\lambda_x.e_1)e_2\triangleright[e_2/x]e_1,$$

$$\langle e_1\leftarrow m=e_2\rangle\Leftarrow m\triangleright e_2\langle e_1\leftarrow m=e_2\rangle.$$

例如:设 $P\equiv\langle x=\lambda_{self}.3,\text{move}=\lambda_{self}.\lambda_{dx}.\langle\text{self}\leftarrow x=\lambda_{\_}.(\text{self}\Leftarrow x)+dx\rangle\rangle$,则

$$P\Leftarrow\text{move }2=(\lambda_{self}.\lambda_{dx}.\langle\ldots\rangle)P2=\langle P\leftarrow x-\lambda_{\_}.(P\Leftarrow x+2)=\langle P\leftarrow x=\lambda_{\_}.3+2\rangle$$

$$=\langle P\leftarrow x=\lambda_{self}.5\rangle=\langle x=\lambda_{self}.5,\text{move}=\lambda_{self}.\lambda_{dx}.\langle\ldots\rangle\rangle.$$

此外,对于对象我们有:若 $m\neq n$ 且不在 e_1 中出现,则有:

$$\langle\langle e_1 \leftarrow m = e_2\rangle \leftarrow n = e_3\rangle \xrightarrow{book} \langle\langle e_1 \leftarrow n = e_3\rangle \leftarrow m = e_2\rangle,$$

$$\langle\langle e_1 \leftarrow m = e_2\rangle \leftarrow m = e_3\rangle \xrightarrow{book} \langle e_1 \leftarrow m = e_3\rangle.$$

记"$\rightarrow\!\!>$"是"$\rightarrow$"的自反、传递闭包.

定理 1. "$\rightarrow$"归约满足 Church-Rosser 性质.

证明:见文献[13].

2 解释器的构造

在本节中,我们介绍对象式 Lambda 演算的自解释器,记作 Eval.首先我们来定义对象式 Lambda 项转换成代码的函数$\ulcorner\cdot\urcorner$,然后证明其正确性.

结合高阶抽象语法(higher order abstract syntax)和符号代码化(the coding of signatures)[14],我们获得对象式 Lambda 演算的程序转换成代码的表态方式为

$$\ulcorner x\urcorner \equiv \lambda_{abcde}.\, ax,$$

$$\ulcorner e_1 e_2\urcorner \equiv \lambda_{abcde}.\, b\ulcorner e_1\urcorner\ulcorner e_2\urcorner,$$

$$\ulcorner \lambda_x.\, e_1\urcorner \equiv \lambda_{abcde}.\, c(\lambda_x.\ulcorner e_1\urcorner),$$

$$\ulcorner e_1 \Leftarrow m\urcorner \equiv \lambda_{abcde}.\, d\ulcorner e_1\urcorner \Leftarrow m,$$

$$\ulcorner \langle e_1 \leftarrow m = e_2\rangle\urcorner \equiv \lambda_{abcde}.\, e(\langle e_1 \leftarrow m = e_2\rangle).$$

注:① $e \Leftarrow m$ 中 e 为对象,形式为$\langle m_1 = e_1, \ldots, m_n = e_n\rangle$,

$$\ulcorner e\urcorner \equiv \langle m_1 = \ulcorner e_1\urcorner, \ldots, m_n = \ulcorner e_n\urcorner\rangle.$$

② 对一个对象$\langle e_1 \leftarrow m = e_2\rangle$,由定义可知,当 m 在 e_1 中不出现时,在 e_1 中添加方法 m,当 e_1 中已有方法 m 时,用新方法替换原方法 m.因此,作为一个对象的定义我们在转换中保留,只有发生消息传递时,才有计算发生,此时,在程序中由$\ulcorner e \Leftarrow m\urcorner$高阶抽象来转换成代码.

我们结合不动点算子,则可得到自解释器为

$$\begin{aligned} Eval \equiv Y(&\lambda_p.\, \lambda_q.\, q(\lambda_x.\, x)\\ &(\lambda_{yz}.\, (py)(pz))\\ &(\lambda_y.\, \lambda_v.\, p(yv))\\ &(\lambda_y.\, py \Leftarrow m)\\ &(\lambda_x.\, x)). \end{aligned}$$

这里,$Y \equiv \lambda_h.\, (\lambda_x.\, h(x,x))(\lambda_x.\, h(x,x))$.

定理 2. 对所有对象式 Lambda 表达式 e,有 $Eval(\ulcorner e\urcorner) \rightarrow\!\!> e$.

证明:记

$$\begin{aligned} E' \equiv &\lambda_p.\, \lambda_q.\, q(\lambda_x.\, x)\\ &(\lambda_{yz}.\, (py)(pz))\\ &(\lambda_y.\, \lambda_v.\, p(yv))\\ &(\lambda_y.\, py \Leftarrow m)\\ &(\lambda_x.\, x), \end{aligned}$$

则 $$Eval\ulcorner e\urcorner = YE'\ulcorner e\urcorner \rightarrow (\lambda_x.\, E'(xx)(\lambda_x.\, E'(xx))\ulcorner e\urcorner \equiv E''\ulcorner e\urcorner.$$

为证明方便,记

$$E_1 \equiv (\lambda_x.\, x);\quad E_2 \equiv (\lambda_{yz}.\, (E''y)(E''z));\quad E_3 \equiv (\lambda_y.\, \lambda_v.\, E''(yv));$$

$$E_4\equiv(\lambda_y.\,E''y\Leftarrow m);\quad E_5\equiv(\lambda_x.\,x),$$

则有 $$E''\ulcorner e\urcorner=(\lambda_x.\,E'(xx))(\lambda_x.\,E'(xx))\ulcorner e\urcorner\rightarrow\succ\ulcorner e\urcorner E_1E_2E_3E_4E_5.$$

下面我们证明 $E''\ulcorner e\urcorner\rightarrow\succ e$.

用表达式 e 的结构归纳法证明. 我们假设对 e 的所有子表达式性质成立,则对 e 本身也成立,我们分情形加以证明.

当 $e=x$ 时:

$$\begin{aligned}E''\ulcorner e\urcorner&\rightarrow\succ\ulcorner e\urcorner E_1E_2E_3E_4E_5\\&\equiv(\lambda_{abcde}.\,ax)E_1E_2E_3E_4E_5\\&\rightarrow\succ E_1x\\&\equiv(\lambda_x.\,x)x\\&\rightarrow x\equiv e.\end{aligned}$$

当 $e\equiv e_1e_2$ 时:

$$\begin{aligned}E''\ulcorner e\urcorner&\rightarrow\succ\ulcorner e\urcorner E_1E_2E_3E_4E_5\\&\equiv(\lambda_{abcde}.\,b\ulcorner e_1\urcorner\ulcorner e_2\urcorner)E_1E_2E_3E_4E_5\\&\rightarrow\succ E_2\ulcorner e_1\urcorner\ulcorner e_2\urcorner\\&\equiv(\lambda_{yz}.\,(E''y)(E''z))\ulcorner e_1\urcorner\ulcorner e_2\urcorner\\&\rightarrow\succ(E''\ulcorner e_1\urcorner)(E''\ulcorner e_2\urcorner)\\&\rightarrow\succ e_1e_2\qquad\text{由归纳假设}\\&\equiv e.\end{aligned}$$

当 $e\equiv\lambda_x.\,e_1$ 时:

$$\begin{aligned}E''\ulcorner e\urcorner&\rightarrow\succ\ulcorner e\urcorner E_1E_2E_3E_4E_5\\&\equiv(\lambda_{abcde}.\,c(\lambda_x.\ulcorner e_1\urcorner))E_1E_2E_3E_4E_5\\&\rightarrow\succ E_3(\lambda_x.\ulcorner e_1\urcorner)\\&\equiv(\lambda_y.\,\lambda_v.\,E''(yv))(\lambda_x.\ulcorner e_1\urcorner)\\&\rightarrow\lambda_v.\,E''((\lambda_x.\ulcorner e_1\urcorner)v)\\&\rightarrow\lambda_x.\,E''\ulcorner e_1\urcorner\\&\rightarrow\succ\lambda_x.\,e_1\qquad\text{由归纳假设}\\&\equiv e.\end{aligned}$$

当 $e\equiv e_1\Leftarrow m$ 时:

$$\begin{aligned}E''\ulcorner e\urcorner&\rightarrow\succ\ulcorner e\urcorner E_1E_2E_3E_4E_5\\&\equiv(\lambda_{abcde}.\,d\ulcorner e_1\urcorner\Leftarrow m)E_1E_2E_3E_4E_5\\&\rightarrow E_4\ulcorner e_1\urcorner\Leftarrow m\\&\equiv(\lambda_y.\,E''y)\ulcorner e_1\urcorner\Leftarrow m\\&\rightarrow E''\ulcorner e_1\urcorner\Leftarrow m\\&\rightarrow\succ e_1\Leftarrow m\qquad\text{由归纳假设}\\&\equiv e.\end{aligned}$$

当 $e=\langle e_1\leftarrow m=e_2\rangle$ 时,类似于当 $e=x$ 时的证明. □

3 约束时间分析(binding-time analysis)

约束时间分析用来确定哪些项是静态的(静态的表达式在部分计值时完成计算),哪些项是动态的(动态的表达式留在剩余程序中).为此,我们引入双层对象式 Lambda 表达式.

定义 4. 一个双层 Lambda 项是由下列抽象语法产生的表达式:

$$e::=x\mid\lambda_x.e\mid e_1e_2\mid e\Leftarrow m\mid\langle e_1\leftarrow m=e_2\rangle\mid,$$

$$\underline{\lambda}_x.e\mid e_1\_e_2\mid e\underline{\Leftarrow}m.$$

这里,前 5 项表示静态项,后 3 项为动态项.

我们采用的约束时间分析基于类型推导,引入类型系统用于对双层 Lambda 项进行标记.同时,在类型系统中定义合式(well-formed)的概念,只有类型合适的表达式才是一个合法的双层 Lambda 项,这样,利用一定的类型规则对原程序进行类型检查,一旦有非合适的表达式就在适当的位置将表达式变换成剩余程序(code-code 变换).BTA 需要完成尽可能多的计算,同时不改变原程序的语义.

定义 5. 双层类型由下列抽象语法产生:

$$\tau::=\iota\mid\tau_1\rightarrow\tau_2\mid\text{classt}.R\mid\text{code},$$

$$R::=r\mid\langle\langle\rangle\rangle\mid\langle\langle R\mid m:\tau\rangle\rangle\mid\lambda_t.R\mid R\tau.$$

我们记 $\Gamma\vdash e:\tau[w]$ 为假设 Γ 下,e 的注释项为 w,且对应的类型为 τ.类型系统的推断规则为

$$\Gamma\vdash x:\Gamma(x)[x]$$

$$\frac{\Gamma,x:\tau_1\vdash e:\tau_2[w]}{\Gamma\vdash\lambda_x.e:\tau_1\rightarrow\tau_2:[\lambda_x.w]}$$

$$\frac{\Gamma\vdash e_1:\tau_1\rightarrow\tau_2[w_1]\quad\Gamma\vdash e_2:\tau_1[w_2]}{\Gamma\vdash e_1e_2:\tau_2[w_1w_2]}$$

$$\frac{\Gamma\vdash *}{\Gamma\vdash\langle\rangle:\text{classt}.\langle\langle\rangle\rangle}$$

$$\frac{\Gamma\vdash e:\text{classt}.\langle\langle R\mid m:\tau\rangle\rangle[w]}{\Gamma\vdash e\Leftarrow m:\{\text{classt}.\langle\langle R\mid m:\tau\rangle\rangle/t\}\tau[w\Leftarrow m]}$$

$$\frac{\Gamma\vdash e_1:\text{classt}.\langle\langle R\mid\boldsymbol{m}:\tau\rangle\rangle[w_1]\Gamma\vdash e_2:\{\text{classt}.\langle\langle R\mid\boldsymbol{m}:\tau,n:\tau\rangle\rangle/t\}(t\rightarrow\tau)[w_1]}{\Gamma\vdash\langle e_1\leftarrow n=e_2\rangle:\text{classt}.\langle\langle R\mid\boldsymbol{m}:\tau,n:\tau\rangle\rangle\_\langle w_1\leftarrow n=w_2\rangle}$$

$$\frac{\Gamma\vdash e_1:\text{classt}.\langle\langle R\mid\boldsymbol{m}:\tau\rangle\rangle[w_1]\Gamma\vdash e_2:\{\text{classt}.\langle\langle R\mid\boldsymbol{m}:\tau\rangle\rangle/t\}(t\rightarrow\tau,)[w_1]}{\Gamma\vdash\langle e_1\leftarrow m_i=e_2\rangle:\text{classt}.\langle\langle R\mid\boldsymbol{m}:\tau\rangle\rangle[\langle w_1\leftarrow n=w_2\rangle}$$

$$\frac{\Gamma,x:\text{code}\vdash e:\text{code}[w]}{\Gamma\vdash\lambda_x.e:\text{code}[\underline{\lambda}_x.w]}$$

$$\frac{\Gamma\vdash e_1:\text{code}[w_1]\quad\Gamma\vdash e_2:\text{code}[w_2]}{\Gamma\vdash e_1e_2:\text{code}[w_1\ w_2]}$$

$$\frac{\Gamma\vdash e:\text{code}[w]}{\Gamma\vdash e\Leftarrow m:\text{code}[w\underline{\Leftarrow}m]}$$

定义 6. 设双层对象式 Lambda 项 w 为对象式 Lambda 项 e 的注释项(annotation term),w 称作合适的(well-formed),存在 Γ,若 $\Gamma\vdash e:\tau[w]$,则有 $\Gamma\vdash *$.

4 例 化

例化就是由 BAT 分析产生的双层对象式 Lambda 语言程序和静态参数值,经过求值得到的剩

余程序,例化器实际上就是双层对象式 Lambda 语言的解释器.我们首先给出双层对象式 Lambda 项转换成代码的函数$\lfloor\cdot\rfloor$,然后定义例化器 Pev,例化器 Pev 实际上就是 $Eval$ 的自然扩充.最后我们来证明部分计值器的正确性.

$$\lfloor x\rfloor\equiv\lambda_{abcdefgh}.\,ax$$
$$\lfloor e_1.\,e_2\rfloor\equiv\lambda_{abcdefgh}.\,b\lfloor e_1\rfloor\lfloor e_2\rfloor,$$
$$\lfloor\lambda_x.\,e_1\rfloor\equiv\lambda_{abcdefgh}.\,c(\lambda_x.\,\lfloor e_1\rfloor),$$
$$\lfloor e_1\Leftarrow m\rfloor\equiv\lambda_{abcdefgh}.\,d\lfloor e_1\rfloor\Leftarrow m,$$
$$\lfloor\langle e_1\leftarrow m=e_2\rangle\rfloor\equiv\lambda_{abcdefgh}.\,e(\langle e_1\leftarrow m=e_2\rangle),$$
$$\lfloor e_1\_e_2\rfloor\equiv\lambda_{abcdefgh}.\,f\_e_1\rfloor\lfloor e_2\rfloor,$$
$$\lfloor\lambda_*.\,e_1\rfloor\equiv\lambda_{abcdefgh}.\,g(\lambda_*.\,\lfloor e_1\rfloor),$$
$$\lfloor e_1\Leftarrow m\rfloor\equiv\lambda_{abcdefgh}.\,h\lfloor e_1\rfloor\Leftarrow m.$$

扩充 $Eval$ 可得 Pev 为

$$\begin{aligned}Pev\equiv Y(&\lambda_p.\,\lambda_q.\,q(\lambda_x.\,x)\\&(\lambda_{ww'}.\,(pw)(pw'))\\&(\lambda_w.\,\lambda_v.\,p(wv))\\&(\lambda_w.\,pw\Leftarrow m)\\&(\lambda_x,x)\\&(\lambda_{ww'}..\,\lambda_{abcdef}.\,c(pw)(pw'))\\&(\lambda_w.\,\lambda_{abcde}.\,c(\lambda_v.\,p(w(\lambda_{abcde}.\,av))))\\&(\lambda_w.\,\lambda_{abcde}.\,d(pw)\Leftarrow m)).\end{aligned}$$

定理 3(部分计值的正确性). 若 $\Gamma\vdash e:\tau[w]$,且 $\Gamma\vdash *$,则 $Eval(Pev(\lfloor w\rfloor))\rightarrow\!\!\succ e$.

证明:根据双层对象式 Lambda 项的生成语法作结构归纳法证明,我们假设对 e 的所有子表达式性质成立,则对 e 本身也成立,我们分情形加以证明.对静态双层对象式 Lambda 项的证明类似于定理 2 的证明,下面我们对动态双层对象式 Lambda 项给予证明:

对于 $w\equiv w_1\ w_2$,

$$\begin{aligned}&Eval(Pev\lfloor w_1\_w_2\rfloor)\\&\equiv Eval(Pev(\lambda_{abcdefgh}.\,f\lfloor w_1\rfloor\lfloor w_2\rfloor\rfloor&&\text{由}\lfloor\_\cdot\rfloor\text{的定义}\\&\rightarrow\!\!\succ Eval(\lambda_{abcde}.\,b(Pev\lfloor w_1\rfloor)(Pev\lfloor w_2\rfloor)&&\text{由 }Pev\text{ 的定义}\\&\rightarrow\!\!\succ(Eval(Pev\lfloor w_1\rfloor))(Eval(Pev\lfloor w_2\rfloor))&&\text{由 }Eval\text{ 的定义}\\&\rightarrow\!\!\succ e_1e_2&&\text{由归纳假设}\\&\equiv e.\end{aligned}$$

对于 $w\equiv\lambda x.\,w_1$,

$$\begin{aligned}&Eval(Pev\lfloor\lambda_x.\,w_1\rfloor)\\&\equiv Eval(Pev(\lambda_{abcdefgh}.\,g(\lambda_x.\,\lfloor w_1\rfloor)))&&\text{由}\lfloor\cdot\rfloor\text{的定义}\\&\rightarrow\!\!\succ Eval(\lambda_{abcde}.\,c(\lambda_x.\,Pev((\lambda_x.\,\lfloor w_1\rfloor)(\lambda_{abcde}.\,ax))))&&\text{由 }Pev\text{ 的定义}\\&\rightarrow\!\!\succ(Eval(\lambda_{abcde}.\,c(\lambda_x.\,Pev\lfloor w_1\rfloor))\\&\rightarrow\!\!\succ\lambda_x.\,Eval(Pev\lfloor w_1\rfloor)&&\text{由 }Eval\text{ 的定义}\\&\rightarrow\!\!\succ\lambda_x.\,e_1&&\text{由归纳假设}\\&\equiv e.\end{aligned}$$

对于 $w \equiv w_1 \Leftarrow m$,

$$
\begin{aligned}
& Eval(Pev\lfloor w_1 \Leftarrow m \rfloor) \\
\equiv & Eval(Pev(\lambda_{abcdefgh}.h\_w_1\rfloor \Leftarrow m)) && \text{由}\lfloor\cdot\rfloor\text{的定义} \\
\rightarrow\!\succ & Eval(\lambda_{abcde}.dPev\lfloor w\_\rfloor \Leftarrow m) && \text{由 } Pev \text{ 的定义} \\
\rightarrow\!\succ & Eval(Pev\lfloor w\_\rfloor) \Leftarrow m && \text{由 } Eval \text{ 的定义} \\
\rightarrow\!\succ & e_1 \Leftarrow m && \text{由归纳假设} \\
\equiv & e.
\end{aligned}
$$

□

5 结论和今后的工作

我们在 Lambda 演算部分计值器的构造的基础上,通过在 Lambda 演算中引入对象扩充为对象式 Lambda 演算,论证了构造对象式 Lambda 演算部分计值器的可能性,并给出了其形式化表示,为今后设计和实现具体的面向对象程序设计语言的部分计值器提供了理论基础.此外,今后还需进一步扩充对象式 Lambda 语言,使其具有并发机制,开发出并发面向对象程序设计语言的部分计值器.

References:

[1] Futamura, Y. Partial evaluation of computation process——an approach to a compiler-compiler. Systems, Computer, Controls, 1971,2(5):45~50.

[2] Turchin, V. F. The use of metasystem transition in theorem proving and program optimization. LNCS 95, 1980. 645~657.

[3] Ershov, A. P. On the essence of compilation. In: Neuhold, E. J., ed. Formal Description of Programming Concept. Amsterdam: North-Holland, 1978. 391~420.

[4] Gomard, C. K., Jones, N. D. A partial evaluator for the untyped lambda-calculus. Journal of Functional Programming, 1991,1(1):21~69.

[5] Gomard, C. K. A self-applicable partial evaluator for the Lambda calculus: correctness and pragmatics. ACM Transactions on Programming Language and Systems, 1992,14(2):147~172.

[6] Mogensen, T. Efficient self-interpretation in Lambda calculus. Journal of Functional Programming, 1992,2(3):345~364.

[7] Palsberg, J. Correctness of binding-time analysis. Journal of Functional Programming, 1993,3(3):347~363.

[8] Bono, V., Buglies, M., Liquori, L. A Lambda calculus of incomplete objects. LNCS 1113, 1996. 218~229.

[9] Bono, V., Liquori, L. A subtyping for the fisher-honsell-mitchell lambda calculus of objects. LNCS 933, 1995. 16~29.

[10] Fisher, K., Mitchell, J. C. A delegation-based object calculus with subtyping. LNCS 965, 1995. 42~61.

[11] Fisher, K. Type systems for object-oriented programming languages [Ph. D. Thesis]. Stanford University, 1996.

[12] Mitchell, J. C., Honsell, F., Fisher, K. A Lambda calculus of object and method specialization. Nordic Journal of Computing, 1994,1(1):3~37.

[13] Gianantonio, P. D., Honsell, F., Liquori, L. A lambda calculus of objects with self-inflicted extension. ACM Sigplan, 1998,33(10):166~177.

[14] Wand, M. Specifying the correctness of binding-time analysis. Journal of Functional Programming, 1993,3(3):365~387.

Self-Applicable Partial Evaluation for the Lambda Calculus of Objects*

WANG Ming-wen, SUN Yong-qiang

(*Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai* 200030, *China*)

E-mail: sun-yq@cs.sjtu.edu.cn

http://www.sjtu.edu.cn

Abstract: A self-applicable partial evaluator for the Lambda calculus of objects is presented in this paper which is an untyped Lambda calculus extended with object primitives. The classic three-steps methodology is used to construct the partial evaluator. First, a meta-interpreter is defined for the language. Second, an abstract analysis (binding-time analysis) is introduced to determine which operations can be executed at compile-time and which operations will be executed at run-time. Finally, the self-applicable partial evaluator is exhibited. Proofs of the correctness of the meta-interpreter and self-applicable partial evaluator are also given in this paper.

Key words: partial evaluation; Lambda calculus of objects; binding-time analysis

* Received June 21, 1999; accepted May 10, 2000

Supported by the National High Technology Development Program of China under Grant No. 863-306-05-04-3