高阶函数式语言到逻辑式语言的转换*

宋立彤 金成植 王丹茹

(吉林大学计算机科学系 长春 130023)

摘要 本文给出高阶函数式语言到Prolog语言的一种转换技术.其中主要用到抽象闭包分析、Lambda提升、顺序化以及Prolog中的Call技术.本文的研究重点在于高阶性的处理上.

关键词 高阶函数,延迟计算,抽象闭包,call谓词,顺序化.

程序设计语言的研究包括语言的实现、语言的语义描述、语言的转换以及程序的自动生成等内容.特别是不同风格语言之间的程序转换理论,有助于研究不同语言之间的语义关系,同时也有助于语言的实现.众所周知,高阶性是函数式语言所特有的高雅而用得非常多的一种语言机制,本文考虑了高阶性和延迟性以及复杂模式,并在SUN工作站上加以实现.作为逻辑式语言,我们考虑了Prolog语言.实现FuncPro(函数—Prolog)型语言的一种有效方法是通过预处理将函数型部分转换成Prolog型.故本文提出的转换方法是很实用的一种方法.

1 高阶函数式语言HFL简介

基于Robert的Standard ML[1],我们设计3HFL语言,下面是HFL的语法结构.其中V表示非函数变量,f表示自定义函数名,F表示形参函数名.我们假设函数都是有名的,因为$(\lambda x.E)E1$可写成let $f\ x = E$ in $f\ E1$.

$Prog$ ::= E

E ::= $V|C|(E1,\ldots,En)|[E1,\ldots,En]|ctr(E1,\ldots,En)|E1\ op\ Em|f\ E1\ldots Em|F\ E1\ldots Em|$if $E1$ then $E2$ else $E3|$case $E0$ of $Matc$ end$|D$ in $E1$

$Matc$ ::= $Pt1: E1 \square \ldots\ldots \square Ptk: Ek$

D ::= let $Lb|$letrec $Lb|$type Tb

Pt ::= $SPt|Pt:Tp$

SPt ::= $-|C|V|F|(Pt1,\ldots,Ptn)|[Pt1,\ldots,Ptn]|tr(Pt1,\ldots,Ptn)$

Lb ::= $Pt=E|f\ Pt1,\ldots,Ptn=E|Lb1$ and $Lb2$

Tb ::= $id=Tp1|Tb1$ and $Tb2$

---

* 作者**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新技术与软件自动生成.**王丹茹**,女,1969年生,助教,主要研究领域为软件自动化与软件工程.

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—07—10收到修改稿

© 中国科学院软件研究所 http://www.jos.org.cn

$Tp ::= Btp|(Tp)|Tp^*|Tp1 \to Tp2|Tp1 \times .. \times Tpn|ctr1(Tp1)+...+ctrn(Tpn)$

2 抽象闭包分析

当 F 是形参名时其对应实参为一高阶表达式,这时需要有关 F 的信息,称之为 F 的抽象闭包集.并记为 $ACS(F)$.抽象闭包的具体表示为(f,m),其中 f 是自定义函数的名字,而 m 是一个整数.我们称 $arity(f)-m$ 为抽象闭包(f,m)的度数.F 可有多个抽象闭包.但其中每个闭包的度数是一样的.F 的抽象闭包的度数也称为 F 的度数,并记为 $degree(F)$.

下面将给出求抽象闭包的算法.由于篇幅所限,在此只给出部分抽象算法.

$\Gamma: Expr \to Env \to AbsCloSet \times Env$

$s \in AbsCloSet = \{(f^i, m_i) | f^i$ 为自定义函数名,$m_i < arity(f^i)\}$

$\varepsilon \in Env = Params \to p(AbsCloSet)$

下面 fix 表示最小不动点算子,snd 为取二元组的第 2 个分量的操作,另外$\mathbb{R}$和 inc 的定义如下:

$\mathbb{R}: Env \times Env \to Env$

$inc: AbsCloSet \times Int \to AbsCloset$

$\varepsilon_1 \mathbb{R} \varepsilon_2 = \lambda x.\ \varepsilon_1(x) \cup \varepsilon_2(x)$

$inc(s,m) = \{(f^i, m_i+m) | (f^i, m_i) \in s\}$

所求 ε 由式子 $\varepsilon = \mathrm{fix}(\lambda\varepsilon.\ \mathbb{R}_i\ snd(\Gamma[\![E^i]\!]\varepsilon))$来确定,其中 E_i 为函数 f^i 的体表达式.

$\Gamma[\![V]\!]\varepsilon = (\{\}.\ \varepsilon)$

$\Gamma[\![\text{if } E_0\ E_1\ E_2]\!]\varepsilon = \text{Let } (s_i, \varepsilon_i) = \Gamma[\![E_i]\!]\varepsilon \text{ in } (s_1 \cup s_2, \mathbb{R}_j \varepsilon_j)$

$\Gamma[\![f^i\ E_1 ... E_m]\!]\varepsilon = \text{Let } (s_i, \varepsilon_i) = \Gamma[\![E_i]\!]\varepsilon \text{ and } \varepsilon' = \mathbb{R}_j \varepsilon_j$

$\text{in if } m < arity(f^i) \text{ then } (\{(f^i, m)\}, \varepsilon')$

$\text{else Let } (s'', \varepsilon'') = \Gamma[\![E_i]\!]\varepsilon' \text{ in } (s'', \varepsilon'' \mathbb{R}\ \mathbb{R}_j[x_j^i \to s_j])$

$\Gamma[\![F\ E_1 ... E_m]\!]\varepsilon = \text{Let } (s_i, \varepsilon_i) = \Gamma[\![E_i]\!]\varepsilon \text{ and } \varepsilon' = \mathbb{R}_j \varepsilon_j \text{ and } ss = \varepsilon'(F)$

$\text{in if } m < degree(F) \text{ then } (inc(ss, m), \varepsilon')$

$\text{else Let } (s'', \varepsilon'') = \mathbb{R}_{(f^i, \_) \in ss} \Gamma[\![E_i]\!]\varepsilon' \text{ in } (s'', \varepsilon'' \mathbb{R}_j[x_j^i \to s_j])$

3 转换策略

(1)高阶转换策略.在转换中最难的是高阶表达式的转换.高阶性产生于下面 2 种情形:

$$f\ E_1 ... E_m (m < arity(f)), \quad F\ E_1 ... E_m (m < degree(F))$$

其中 F 是函数型形参名,f 是自定义函数名.下面分别介绍转换策略.(a)假设有 LET 子句 let $Tv = f\ Pt_1 ... Pt_m$,则非高阶情形将生成 Prolog 目标:$f\#(Pt_1, ..., Pt_m, Tv)$,高阶情形则将生成如下目标:$Tv = [f\#, Pt_1, ..., Pt_m]$.(b)假设有 LET 子句 let $Tv = F\ Pt_1 ... Pt_m$,则非高阶情形将生成 Prolog 目标序列:append$\#(F, [Pt_1, ..., Pt_m, Tv], Z)$,call0$\#(Z)$,高阶情形则将生成如下目标:append$\#(F, [Pt_1, ..., Pt_m], Tv)$.其中的 call0$\#$表示下面子句:call0$\#(Z) :- X = ... Z$,call$(X)$.

(2)延迟与非延迟的转换策略.函数式语言分为延迟与非延迟式 2 种.非延迟式的主要

© 中国科学院软件研究所 http://www.jos.org.cn

思想是当调用函数时,立即计算实参表达式的值,而延迟式是把实参表达式的计算延迟到被调用函数体的内部,且只有当需要计算时才计算并只计算 1 次,而不重复计算实参值,因此在转换时必须考虑到这 2 种情形.我们采用的主要转换策略是:假设有函数调用的 LET 子句 let $y=f(2+1)$,则当规定为延迟式时,将生成子目标 $f\#(2+1,Y)$;而规定为非延迟式时将生成子目标序列"T is $2+1$,$f\#(T,Y)$".其中 $f\#$ 表示对应于 f 的 Prolog 谓词名.因为 Prolog 语言实际上是延迟式的,因此延迟情形更为容易一些.本文将考虑非延迟情形.

(3)其他转换策略.函数式语言中的模式对应于 Prolog 中的项结构,因此不必改变其结构.在此,重点说明其他几个问题.(a)if 问题:可有 2 种策略,①若 Prolog 系统提供了 if_then_else形目标,则直接将 if 表达式部分转换成 if 形目标,②将每个 if 表达式转换成 Prolog 子句.这时其复杂性来自于 if 的嵌套性.本文将考虑后一种情形.(b)case 问题:如同 if 情形一样可有 2 种方法.这里采用后一种方法.即假设有 let 子句

$$\text{let } Tv=\text{case } E \text{ of } Pt_1:E_1\square.....\square\ Pt_n:E_k \text{ end}$$

则将产生如下 Prolog 目标列"$Tv = E, f\#(Tv,y_1,...,y_n,Z)$"和 Prolog 子句

$$f\#(Pt_1,y_1,...,y_n,Z1):-goals(E_1),...,f\#(Pt_k,y_1,...,y_n,Z_k):-\ goals(E_k).$$

其中 $y_1,...,y_n$ 为 case 表达式中的自由变量.(a)let 问题:经过顺序化后 let 子句中的 Bind 部分具有以下几种结构,它们分别转换成下面所示的 Prolog 子目标:(1)$Pt=Pt1$;(2)V is $Te1\ op\ Te2$;(3)$f\#(Pt1,...,Ptn,V)$.(b)letrec 问题:letrec 定义 2 种对象(变量,函数).函数的递归定义可用谓词的递归定义来处理,主要是变量的递归定义问题,这是函数式语言的一大特点.逻辑式语言没有这一机制,但 Prolog 有延迟计算的功能,我们可利用这一机制来解决问题.

4　Lambda 提升和顺序化

转换的第 1 步工作是进行 Lambda 提升,使得在表达式中无函数定义.Johnsson[2]曾给过提升算法,我们不再赘述.顺序化的主要目的主要有 2 个:一是把表达式中的所有函数调用部分按其计算顺序提出来;二是使得中间代码便于产生非延迟式的 Prolog 目标程序.下面是 Lambda 提升后的顺序化函数程序的结构定义,其中 SE 是一个函数的调用.

$Prog$　　::= $SFd_1...SFd_n$▲ SE

SFd　　::= $f(Pt1_1,...,Pt1_n)=SE_1\square.....\square\ f(Ptk_1,...,Ptk_n)=SE_n$

Pt　　::= $-|Single|(Pt_1,...,Pt_n)|[Pt_1,...,Pt_n]\ |ctr(Pt_1,...,Pt_n)$

$Single$　　::=$C|V|F$

SE　　::=$TE|SD$ in TE

TE　　::=$Pt|SOPE$

$SOPE$　　::=Single$|(SOPE)|SOPE_1\ op\ SOPE_2$

SD　　::=let Lb|letrec Lb

Lb　　::=$Pt=TE|Pt=f\ Pt_1...Pt_m|Pt=F\ Pt_1...Pt_m|Lb_1$ and Lb_2|empty

5　顺序化程序到 Prolog 的转换

经过前面的顺序化过程,已把通常的函数式语言程序转换成了我们所需要的顺序化程

© 中国科学院软件研究所 http://www.jos.org.cn

序. 下面将给出由顺序化程序到 Prolog 程序的转换方法. 其中 $f\#$ 表示 f 的相应谓词名.

$\varphi: SFD* \blacktriangle SExpr \rightarrow Clause \blacktriangle Goals$ $\gamma: SFD* \rightarrow Clause$

$\tau: SExpr \rightarrow Goals \times Pttern$ $\beta: BIND \rightarrow GOALS$

$\varphi[\![SFd_1 \ldots SFd_k \blacktriangle SE]\!] = \gamma[\![SFd_1]\!] \ldots \gamma[\![SFd_k]\!] \blacktriangle goals$

$\gamma[\![f(Pt1_1,\ldots,Pt1_n)=SE_1 \square \ldots \square f(Ptm_1,\ldots,Ptm_n)=SE_m]\!] =$

$f\#(Pt1_1,\ldots,Pt1_n,Pt_1) :- goals1. \ldots f\#(Ptm_1,\ldots,Ptm_n,Ptm) :- goalsm.$

其中 $\tau[\![SE_i]\!]=(goals_i,Pt_i), i=1,\ldots,m.$ $\tau[\![SE]\!]=(goals,Tv)$

$\tau[\![V]\!] = \_ \blacktriangle V$

$\tau[\![F]\!] = \_ \blacktriangle F$

$\tau[\![f]\!] = \_ \blacktriangle [f|[]]$

$\tau[\![(Pt1,\ldots,Ptn)]\!] = \_ \blacktriangle tupn(Pt1,\ldots,Ptn)$ (对于不同类型的多元组给不同的构造子 $tupn$)

$\tau[\![SOPE1\ op\ SOPE2]\!] = Tv$ is $SOPE1\ op\ SOPE2 \blacktriangle Tv$

$\tau[\![$letrec $Bind1$ and... and $Bindn$ in $TE]\!] = Goal1,\ldots,Goaln, Goal \blacktriangle term$

其中 $\beta[\![Bindi]\!]=Goali, \tau[\![TE]\!]=Goal \blacktriangle term$

$\beta[\![empty]\!]=$空

$\beta[\![Pt=Pt1]\!]=Pt=Pt1$ (Pt 为非变量)

$\beta[\![V=SOPE]\!]=V$ is $SOPE$

$\beta[\![V=f\ Pt1\ldots Ptm]\!] = f\#(Pt1,\ldots,Ptm, V)$, 当 $m=arity(f)$

$= V=[f\#,Pt1,\ldots,Ptm]$, 当 $m\neq arity(f)$

$\beta[\![V=F\ Pt1\ldots Ptm]\!] =$ append$\#(F,[Pt1,\ldots,Ptm],V,Z)$, call0$\#(Z)$, 当 $m=degree(F)$

$=$ append$\#(F,[Pt1,\ldots,Ptm],V)$, 当 $m\neq degree(F)$

6 转换实例

我们将给出包含函数形参和高阶调用的较复杂的函数式语言程序例. 假设有下列提升后的函数式语言程序:

map F L=if ($null?$ L) then [] else [F (hd L)|map F (tl L)]

add N X=N + X

goal G I=append(map (G 2) I) (map (G 3) I)

▲ goal add [1,2,3]

则转换后的完整 Prolog 程序如下面所示:

map$\#(F,L,Tv) :-$ if$\#(F,L,Tv)$.

if$\#(F,[],[])$.

if$\#(F,L,[Tv2|Tv4]) :- Tv1=[hd,L]$, append$\#(F,[Tv1,Tv2],Z)$, call0$\#(Z)$,

$Tv3=[tl,L]$, append$\#(F,[Tv3,Tv3],Z1)$, call0$\#(Z1)$

add$\#(N,X,Tv) :- Tv$ is $N+X$.

goal$\#(G,I,Tv5) :-$ append$\#(G,[2],Tv1)$, map$\#(Tv1,I,Tv2)$, append$\#(G,[3],Tv3)$, map$\#(Tv3,I,Tv4)$, append$\#(Tv2,Tv4,Tv5)$.

? $-$goal$\#$ (goal$\#$,[1,2,3],W).

参考文献

1 Robert Harper, David MacQueen, Robin Milnern. Standard ML. Computer Science Dept., Edinburgh, Universi-

© 中国科学院软件研究所 http://www.jos.org.cn

ty, Report, 1985.

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

A TECHNIQUE FOR TRANSFORMING HIGHER ORDER FUNCTIONAL LANGUAGE TO LOGICAL LANGUAGE

Song Litong　Jin Chengzhi　Wang Danru

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

Abstract　The paper gives a technique for transforming higher order functional language to Prolog language, such techniques as Lambda lifting, abstract closure analysis, sequentialization and predicate call in Prolog are used. The main research of this paper is aimed at higher order function.

Key words　Higher order function, lazy evaluation, abstract closure, call predicate, sequentialization.

© 中国科学院软件研究所 http://www.jos.org.cn