一类递归函数的多态类型*

黄文集[1,2+]

[1](中国科学院 软件研究所,北京 100080)

[2](中国科学院 研究生院,北京 100039)

Polymorphic Type for a Kind of Recursive Functions

HUANG Wen-Ji[1,2+]

[1](Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate School, The Chinese Academy of Sciences, Beijing 100039, China)

+ Corresponding author: Phn: +86-10-62562796, Fax: +86-10-62563894, E-mail: hwj@ios.ac.cn, http://www.ios.ac.cn

Received 2003-09-05; Accepted 2004-03-31

Huang WJ. Polymorphic type for a kind of recursive functions. *Journal of Software*, 2004,15(7):969~976.
http://www.jos.org.cn/1000-9825/15/969.htm

Abstract: Based on recursive functions defined on context-free language, LFC (language for context free recursive function) is a formal specification language and fits for dealing with phrase structure. LFC is yet another functional language with many general characteristics. It has been implemented in SAQ (specification acquisition system).The original type system for LFC is not polymorphic. With type variables, the original type system can be augmented and become a polymorphic type system. The type checking algorithm and some problems about implementation are also discussed. Polymorphic type system makes LFC more agile and predicts good future in the applications of LFC.

Key words: functional language; polymorphism; type checking; recursive function; type system

摘 要: 以上下文无关语言上的递归函数为基础的语言 LFC(language for context free recursive function)是一种形式规约语言,适于处理短语结构.LFC 也是函数式语言,具有函数式语言的许多特点.LFC 已经在形式规约获取系统 SAQ(specification acquisition system)中实现,为其最初设计的类型系统不支持多态类型.引入类型变量和相应的类型检查方法,就可以将其类型系统扩充为多态类型系统.对多态类型系统实现中的一些问题也进行了讨论.在实现多态之后,LFC 的灵活性将得到增强,从而会为其应用创造更为有利的条件.

关键词: 函数式语言;多态;类型检查;递归函数;类型系统

中图法分类号: TP301 **文献标识码**: A

形式规约语言 LFC(language for context free recursive function)[1]是形式规约获取系统 SAQ(specification acquisition system)[2,3]提供的一种函数式语言.作为函数式语言,LFC 具备了一般函数式语言的特点(良好的数学基础、引用透明、模式匹配等)[4].LFC 的最大特点是将上下文无关语言(context free language,简称 CFL)作为其

* Supported by the National Natural Science Foundation of China under Grant Nos.60273023, 60103008 (国家自然科学基金)

作者简介: 黄文集(1977－),男,吉林长春人,博士生,主要研究领域为软件设计方法,形式规约语言.

数据类型,支持包括数上递归函数、字上递归函数和上下文无关语言上的递归函数(CFRF)[5,6]在内的多种类递归函数.在 SAQ 系统中为 LFC 设计的类型系统类似于 subtype[7]系统,但与其又存在较大的差别.该类型系统有以下几个问题:

(1) 没有实现多态类型;

(2) 不能进行类型推导;

(3) 不能做到完全的静态类型检查.

其中(2)和(3)是由于上下文无关语言的包含、相等、相交为空是不可判定问题而造成的.

除了 subtype 系统之外,多态系统也是许多语言所采用的类型系统.多态(polymorphism)指的是有很多种形式,从程序语言来说,表示数据或程序有很多种类型或可操作于多种类型之上.程序的多态类型最早由 Strachey 所设想[8],Hindley 在 Curry 的工作[9]基础上提出了主类型(principal type schema)[10]的概念,主类型的存在意味着类型推导算法总能为程序找到唯一一个"最好"的类型.Milner 在他们的研究基础上,又提出了一个重要的扩展:将类型变量分类,实现了第一个实用的多态类型检查器,并证明了类型系统的正确性[11].后来,许多人在此基础上进行了多态类型的研究及应用[12~16],但本质上与 Milner 提出的类型系统没有太多改变.

为了增加 LFC 的描述能力,我们扩充原来为其设计的 subtype 系统,实现多态. LFC 是以 CFRF 为计算模型的语言,而大多函数式语言源于 lambda 演算.另外,LFC 的数据类型通过产生式来定义,这是其特有的,以前的工作没有遇到过这种情况.针对 LFC 的特性,我们为其设计的新类型系统将是一个集中多态和 subtype 特点的混合系统.

本文第 1 节介绍一些预备知识.第 2 节提出 LFC 的多态类型.第 3 节给出 LFC 的类型检查方法.第 4 节讨论 LFC 的多态类型系统实现中的一些问题.第 5 节总结全文.

1 预备知识

上下文无关文法定义为一个四元组 $G=(V_N,V_T,P,S)$.其中 VN 是非终极符的集合;V_T 为文法 G 的终极符集合;V_T*表示 V_T 上的终极符串集合;$(V_N \cup V_T)$*表示非终极符和终极符的串的集合;产生式是形如 $A\rightarrow\alpha$的推导规则,其中 A 称为产生式的左部,α称为产生式的右部,P 为产生式的有穷集合,S 为开始符号,空字用λ表示.空字满足 $\lambda u=u\lambda=u$.

我们用 $\alpha\Rightarrow\beta$表示:存在 γ,δ,η和 A,满足 $\alpha=\gamma A\delta,\beta=\gamma\eta\delta$,且 $A\rightarrow\eta$是 P 中的一个产生式.$\alpha\overset{*}{\Rightarrow}\beta$表示:存在 $\alpha_1,\ldots,\alpha_m(m\geq 1)$使得

$$\alpha=\alpha_1\Rightarrow\ldots\Rightarrow\alpha_m=\beta.$$

LFC 程序的数据类型用上下文无关文法来定义.程序主要由产生式定义、函数定义和函数调用组成.函数定义包括函数声明、变量定义、函数定义体 3 部分.因为不能进行类型推导,所以所有变量及函数的类型必须是显式声明.

我们从一个具体的 LFC 程序实例(计算数字列表元素个数)出发,加以说明.

例如:

```
〈List〉->〈Num〉
〈List〉->〈Num〉,〈List〉
dec Length1:List->Num;
var     x:Num;
        y:List;
def     Length1(x)=1;
        Length1(x[]",")[]y)=add("1",Length1(y))
```

上面的例子是一个计算数字列表元素个数的函数.程序前面是产生式定义,后面是函数部分.其中 Num 是系统提供的数值类型,[]是连接函数的中缀运算符,add 是内建函数,是 Num 上的加运算.dec,var,def 都是系统的保留字,分别表示函数声明、变量声明、函数定义体.从这个例子可以看出,LFC 的函数通过模式匹配来定义.

(*x*)便是一个模式,只有一个模式分量 *x*.

在为 LFC 最初设计的类型系统中,没有实现多态类型.如果在程序中还需要计算字符串列表的函数,我们不得不重新定义函数,尽管其与 Length1 的定义方式完全相同,并不依赖于具体的数据类型.为了解决这个问题,我们将对 LFC 原来的类型系统进行扩充,引入多态类型.

2 LFC 的多态类型

为了在LFC中实现多态类型,我们首先引入类型变量.由于LFC的数据类型是通过产生式定义的,在不导致混乱的情况下,我们将非终极符名与类型名等同使用.在产生式中增加特殊的非终极符表示类型变量,同时为了与原来定义 LFC 的文法保持一致,我们用_Type1,_Type2,...来表示类型变量.

例如:

⟨List⟩->⟨_Type1⟩

⟨List⟩->⟨_Type1⟩,⟨List⟩

上面的例子表示一个 List 的定义,其基本成员类型可以是任何类型(用类型变量表示),列表元素之间用“,”分隔.

定义 1. 如果在定义类型 *A* 的某个产生式的右部含有非终极符 *B*(*B* 不为类型变量),那么我们称 *A* 类型依赖于 *B*.

定义 2. 类型 *A* 的定义产生式(又称为主产生式)是指所有以 *A* 为左部的产生式.

定义 3. 产生式定义团包括类型的定义产生式和所有其依赖的类型的产生式定义团.

定义 4. 如果类型 *A* 的产生式定义团中含有类型变量,那么类型 *A* 为框架类型;否则,*A* 为普通类型.

定义 5. 类型替换定义为用普通类型来代替框架类型中的所有类型变量.

定义 6. 类型连接定义为设文法串 $\alpha=\alpha_1\ldots\alpha_n$,其中 $\alpha_i\in V_T^*$ 或 $\alpha_i\in V_N$,则 $L(\alpha)=\{a_1\ldots a_n|$若 $\alpha_i\in V_T^*, a_i=\alpha_i$;若 $\alpha_i\in V_N, a_i\in L(\alpha_i)\}$,类型 $L(\alpha)$是类型 $L(\alpha_1),\ldots,L(\alpha_n)$的连接.

定义 7. 类型并定义为设产生式组 $X\to\alpha_1|\ldots|\alpha_n$,其中 $\alpha_i\in(V_N\cup V_T)^*$,则 $L(X)=L(\alpha_1)\cup\ldots\cup L(\alpha_n)$,$L(X)$是类型 $L(\alpha_1),\ldots,L(\alpha_n)$的并.

根据上面的定义,对于框架类型和类型变量,我们有如下性质及规定:

(1) 类型变量名以_Type 开头,其后缀以非空的数字串,并且不与其他符号相混淆.

(2) 类型变量的作用域是全局的,这样考虑的原因是为了避免类型的并和连接的时候发生名字冲突.如:

⟨tree⟩->⟨_Type1⟩\⟨*l*tree⟩/⟨*r*tree⟩

⟨*l*tree⟩->⟨leftleaf⟩|⟨tree⟩

⟨*r*tree⟩->⟨rightleaf⟩|⟨tree⟩

⟨leftleaf⟩->⟨_Type2⟩

⟨rightleaf⟩->⟨_Type3⟩

定义了一个二叉树,其中树的非叶节点和左叶子节点、右叶子节点的定义均使用类型变量,采用了 3 个不同的名字,表示可以用 3 种普通类型来替换.

(3) 框架类型可以转换为普通类型,这时候框架类型的产生式定义团内的所有类型变量都被替换成普通类型.

(4) 由于 LFC 的类型系统不能进行类型推导,所以类型替换的时候必须显式说明作为替换者的普通类型,如对于类型 List,⟨Char⟩“A,B,C”是一个字母列表,而⟨Num⟩“1,2,3,12”是一个整数列表,Char 和 Num 分别替换原来 List 中的类型变量_Type1.当多个类型变量需要替换时,替换顺序按类型定义团内类型变量出现的先后顺序进行,如对于类型 tree,⟨Char⟩⟨Num⟩⟨String⟩“A\1/bc”便是 tree 的一个句子,Char 替换_Type1,Num 替换_Type2,String 替换_Type3.

(5) 带有类型替换声明的常量的连接运算⟨Type1⟩...⟨Type*i*⟩α[]⟨Type*j*⟩...⟨Type*n*⟩β=⟨Type1⟩...⟨Type*i*⟩⟨Type*j*⟩...⟨Type*n*⟩scat(α,β),其中 scat 为字符串连接函数.

(6) LFC 程序中函数的模式定义主要依靠产生式,而类型变量无定义产生式,编译器无法进行模式翻译,所以函数的定义域中不能出现类型变量.同样,也不允许类型变量作为函数的值域.如果需要,可以用一个变通的方法,如定义一个类型⟨*A*⟩->⟨_Type1⟩,类型 *A* 是一个框架类型,但没有任何其他定义,其实质作用与类型变量相当.

由此可见,框架类型可以有很多种形式,在进行类型替换转换成普通类型后,尽管其名称没有被改变,但其实质却发生了变化.

有了类型变量之后,便可以定义多态函数.

例如:

⟨List⟩->⟨_Type1⟩

⟨List⟩->⟨_Type1⟩,⟨List⟩

dec Length2:List->Num

var *x*1:_Type1;

*x*2:List;

def Length2(*x*1)=1;

Length2(*x*1[]“,”[]*x2*)=add(“1”,Length2(*x*2));

函数 Length2 可以计算任意列表中元素的个数,如 Length2(⟨Char⟩“A,B,C”)=3,Length2(⟨Num⟩“1,2,3,12”)=4.这样,我们不必另外定义其他函数来计算不同列表中元素的个数,只需对 Length2 进行调用即可,从而实现了参数化多态.

为了引入多态,需要对 LFC 原来的类型系统进行扩展.下面给出对文献[1]中的类型系统进行扩展所涉及的一些概念定义.

令 *Exp* 表示表达式的集合,*Dec* 表示变量、函数声明集,*Gstr* 表示文法串(终极符和非终极符组成的符号串,类型变量当作非终极符对待)集合,*Coer* 为替换类型变量的普通类型的队列.

定义函数:

γ:*Exp*×*Dec*→(*Gstr*,*Coer*)

γ(*e*,*dec*)把表达式 *e* 根据相应的变量、函数声明集映射为 *Gstr* 和 *Coer* 组成的对偶.γ是获取表达式 *e* 的静态类型的方法.

γ可定义如下:

γ(α,*dec*)=(α,EMPTY)

γ(⟨Type1⟩...⟨Type*n*⟩α,*dec*)=(α,Type1▷...▷Type*n*)

γ(*v*,*dec*)=(*dec*|*v*,EMPTY)

γ(*f*(*e*$_1$,...,*e*$_n$),*dec*)=(range(*f*),EMPTY)

γ(⟨Type1⟩...⟨Type*n*⟩*f*(*e*$_1$,...,*e*$_n$),*dec*)=(range(*f*),Type1▷...▷Type*n*)

γ(*e*$_1$[]*e*$_2$,*dec*)=(*fst*(γ(*e*$_1$,*dec*))*fst*(γ(*e*$_2$,*dec*)),*sec*(γ(*e*$_1$,*dec*))▷*sec*(γ(*e*$_2$,*dec*)))

其中α表示常量,*v* 表示变量,*f* 为函数名,*dec*|*v* 表示求 *v* 在 *dec* 中声明的类型,range 表示求函数的值域,▷表示队列连接,*fst* 和 *sec* 分别表示取对偶的第 1 个和第 2 个分量.

定义 8. 最大类型为任意终极符串集合,记为 STRING.

显然,任何 LFC 的数据都属于最大类型 STRING,LFC 的任何类型都是 STRING 的 subtype.

推论 1. 表达式 *e* 静态属于类型 *l* 当且仅当 substitution(*l*,*sec*(γ(*e*,*dec*)))⇒**fst*(γ(*e*,*dec*)).

其中 substitution 返回用 *sec*(γ(*e*,*dec*))中的普通类型替换 *l* 的产生式定义团中类型变量后新的 *l*,如果 *l* 为普通类型或 *sec*(γ(*e*,*dec*))队列为空,则不做任何操作,返回 *l*.

定义 9. 运行环境为变量名到值的映射(包括类型环境,即类型变量到普通类型的映射)的集合.

推论 2. 表达式 *e* 的类型在运行环境 *E* 下可转换为 *l* 当且仅当 *E*|*e*∈substitution′(*l*,*E*).

其中 substitution′返回用 *E* 中的普通类型替换 *l* 的产生式定义团中类型变量后新的 *l*,如果 *l* 为普通类型或 *E*

中普通类型队列为空,则不做任何操作,返回 l.

由以上定义及命题,不难得到下面的性质:

推论 3. 表达式 e 是否静态属于类型 l 是可以判定的.

这个问题便是 CFL 的句型识别问题,与 CFL 的句子识别问题一样,也是可以判定的.

推论 4. 表达式 e 的类型在运行环境 E 下可否转换为 l 是可以判定的.

这个问题便是 CFL 的句子识别问题.

从上述讨论可知,在增加了类型变量之后,LFC 语言将变得更加灵活,类型系统也会更加复杂,类型检查的内容也要随之增多.

3 类型检查

LFC 的类型检查,分为静态类型检查和动态类型检查两部分.静态类型检查的目的在于较早地发现程序中的类型错误,而动态类型检查是因为不能进行完全静态类型检查而不得不进行的检查,但由于 LFC 程序在运行时刻也需要进行上下文无关语言句子分析来得到参数的结构信息,因此动态类型检查可以结合句子分析进行.

LFC 的静态类型检查的内容:

(1) 表达式的类型是否为预期类型或能否转换为预期类型.

(2) 预期类型为框架类型的表达式的显式类型替换声明是否正确,包括替换类型是否为普通类型,替换类型的个数是否与框架类型产生式定义团中的类型变量个数相等.

静态类型检查方法:

1. 函数 f 的第 i 个模式分量检查(假设对应的定义域分量为 $\mathrm{dom}(f)_i$,类型变量与普通类型相同对待)
   1.1. 常量α:如果 $\mathrm{dom}(f)_i \overset{*}{\Rightarrow} \alpha$不成立,则类型错误.
   1.2. 变量 v:如果 $\mathrm{dom}(f)_i \overset{*}{\Rightarrow} dec|v$ 不成立,则类型错误.
   1.3. $\alpha_1[]v_1[]...[]\alpha_n[]v_n[]\alpha$:如果 $\mathrm{dom}(f)_i \overset{*}{\Rightarrow} \alpha_1[]dec|v_1[]...[]\alpha_n[]dec|v_n[]\alpha$不成立,则类型错误.
2. 函数 f 定义体等式右部检查(即表达式 e,预期类型为 range(f)的检查):
   2.1. 常量α:
      2.1.1. range(f)为普通类型,如果 $\mathrm{range}(f) \overset{*}{\Rightarrow} \alpha$不成立,则类型错误.
      2.1.2. range(f)为框架类型,类型错误.
   2.2. ⟨Type1⟩...⟨Type$n$⟩α:
      2.2.1. range(f)为普通类型,类型错误.
      2.2.2. range(f)为框架类型:
         2.2.2.1. Type1,...,Type$n$ 不全是普通类型,类型错误.
         2.2.2.2. 在 range(f)的产生式定义团中,类型变量的个数不为 n,类型错误.
         2.2.2.3. 如果 $\mathrm{substitution}(\mathrm{range}(f),\mathrm{Type1}\triangleright...\triangleright\mathrm{Type}n) \overset{*}{\Rightarrow} \alpha$不成立,则类型错误.
   2.3. 函数 $g(e_1,...,e_n)$:
      2.3.1. 检查参数 $e_1,...,e_n$,预期类型分别为 $\mathrm{dom}(g)_1,...,\mathrm{dom}(g)_n$.
      2.3.2. range(f)为框架类型,类型错误.
   2.4. 函数⟨Type1⟩...⟨Type$n$⟩$g(e_1,...,e_n)$:
      2.4.1. 检查参数 $e_1,...,e_n$,预期类型分别为 $\mathrm{dom}(g)_1,...,\mathrm{dom}(g)_n$.
      2.4.2. range(f)为普通类型,类型错误.
      2.4.3. Type1,...,Type$n$ 不全是普通类型,类型错误.
      2.4.4. range(f)为框架类型,若其产生式定义团中类型变量个数不为 n,则类型错误.
   2.5. 混合形式 $e_1[]e_2$:
      2.5.1. 检查 e_1 和 e_2,预期类型均为最大类型 STRING.
      2.5.2. $e_1[]e_2$ 可转化为常量α,转 2.1.

2.5.3. $e_1[]e_2$ 可转化为常量⟨Type1⟩...⟨Type$n$⟩α,转 2.2.

3. 程序主函数调用 $f(e_1,...,e_n)$的检查:

3.1. 检查参数 $e_1,...,e_n$,预期类型为 dom(f)$_1$,...,dom(f)$_n$.

在上面的检查方法中,只列出了可以静态判断为类型错误的各种情况,对于运行时刻可能会发生的类型错误,将在下面的动态类型检查中加以判断.

动态类型检查方法(表达式 e,预期类型为 l):

1. l 为普通类型,如果 $l \overset{*}{\Rightarrow} *e$,则 e 属于 l,否则类型错误.

2. l 为框架类型:

2.1. 如果 substitution′(l,E)不为普通类型,则类型错误.

2.2. 如果 substitution′(l,E)$\overset{*}{\Rightarrow} e$,则 e 属于 l,否则类型错误.

4 LFC 的多态类型系统实现方法

对 LFC 的类型系统的扩充,使之实现比以前变得复杂,在编译实现过程中,为了能够处理多态类型,需要解决如下几个问题:

(1) 模式分量的翻译:在 LFC 的编译实现过程中,模式分量的翻译主要依赖于上下文无关语言句子分析,即对模式分量用 Earley[17]算法进行句子分析,产生最右推导序列,构造 Vtree[18],得到模式编码.如前面函数 Length2,被变换为

```
0    1
1    add("1",Length2(_evaltree(_child(_y1,1))))
```

其中左边的模式编码 0,1 来自于产生式⟨List⟩->⟨_Type1⟩和⟨List⟩->⟨_Type1⟩,⟨List⟩的编号,可见模式的编码与产生式的编号紧密相关._child 和_evaltree 是内建的求子树和分析树求值函数.

在扩充为多态类型系统之后,类型变量会被普通类型所替代,产生式将发生变化,如在进行函数调用 Length2(⟨Num⟩"1,2,3,4")时,需要用类型 Num 来替换 List 产生式定义团中的类型变量_Type1,这时,List 的产生式定义团被改变了.为了能够使函数运算正确选择模式以及 *V*tree 运算(包括_child 和_evaltree)保持正确性,需要作如下处理:

- 为了保证框架类型可以多次被实例化,每次类型替换前都先要对框架类型的产生式定义团进行复制,然后对副本进行操作.
- 框架类型在进行类型替换的时候,其类型变量全部被替换为相应的普通类型,作为替换者的普通类型的产生式定义团放在原来的框架类型的产生式定义团后面.
- _evaltree 利用替换后的类型产生式进行计算.
- 在执行 LFC 的抽象机中,需要有相应的产生式复制、替换、增加等指令.

(2) 函数递归调用的处理:函数递归调用中当参数未指定类型替换信息,如 Length2,这时所涉及类型的产生式定义团为上一次调用后的产生式定义团.如果重新指定了类型替换信息,则要保存当前的类型产生式定义团,并从框架类型生成替换实例,作为当前的类型产生式定义团.因此,在执行 LFC 的抽象机中,需要有产生式的存储空间和相应的保存、恢复等操作指令.

(3) 文法的二义性问题:LFC 以上下文无关语言上的递归函数(CFRF)作为计算模型,CFRF 中规定函数定义所需的文法是无二义的.由于任意的上下文无关文法有无二义性是不可判定的,因此我们无法对程序中的文法进行二义性检查,只能由使用者来保证.如果扩充 LFC 的类型系统为多态类型系统,有了类型替换,会出现这种情况:用 n 个无二义文法来替换某个无二义文法中的 n 个非终极符,替换后的文法是否为无二义的?

定理 1. 令上下文无关文法 $G_1,G_2,...,G_{n+1}$ 无二义性,开始符号为 $S_1,S_2,...,S_{n+1},G_1,G_2,\ldots,G_{n+1}$ 的产生式中只包括 $S_1,S_2,...,S_{n+1}$ 的产生式定义团,S_1 的产生式定义团中有 n 个类型变量_Type1,...,_Type$n$,$S_2,...,S_{n+1}$ 为普通类型,则对 G_1 用 $S_2,...,S_{n+1}$ 进行类型替换后得到的 G'_1 的二义性是不可判定的.

证明:假设替换后的文法的二义性是可以判定的.

任意上下文无关文法 G,设其开始符号为 S,产生式集合为 P,S 的产生式定义团集合为 PS,如果 $P–PS$ 不为空,即 G 中存在多余的产生式,可将 $P–PS$ 中的产生式从 G 中去掉,并不影响 G.

当 G 中不含有多余的产生式时,产生式集合为 P,构造文法 G',其产生式为 $A_i \rightarrow \_Typei\ \alpha_i, \forall A_i \rightarrow \alpha_i \in P, 1 \le i \le n, n$ 为 P 中产生式的个数,开始符号为 S,显然 G'是无二义的.另有文法 $L_1,...,L_n$,每个文法只有产生式 $B_i \rightarrow \lambda, B_i$ 为在 G'中未出现的非终极符,$1 \le i \le n, L_1,...,L_n$ 也是无二义的.

根据假设,用 $B_1,...,B_n$ 替换 G'的变量_Type1,...,_Type$n$ 后所得到的文法 G''的二义性是可判定的.G''的产生式为 $A_i \rightarrow B_i\ \alpha_i, B_i \rightarrow \lambda, 1 \le i \le n, G''$和 G 是等价的,所以 G 的二义性也是可判定的.由于 G 是任意的上下文无关文法,这与上下文无关文法的二义性是不可判定的相矛盾.因此,假设是荒谬的,替换后的文法的二义性是不可判定的. □

由定理 1 可知,对于类型替换后的文法,我们也无法进行二义性检查,因此其无二义也只能由使用者来保证.但在 LFC 的编译器设计中,我们采用 Earley 算法判定某个终极符串是否为上下文无关语言的句子.对于输入的文法不论其是否有二义,只要待分析终极符串是其句子,Earley 算法总是返回唯一一个最右推导序列.尽管这个推导序列也许并非是我们所期望的,但可以确保操作于有二义文法的函数有确定的返回值,不会发生错误.

5 结束语

本文讨论了形式规约语言 LFC 的类型系统,在原来的 subtype 系统基础上引入类型变量便可实现多态系统,并给出了相应的类型检查方法和实现中一些问题的处理办法.LFC 的多态类型提高了语言的效率,将为其应用提供更为广阔的前景.

致谢 中国科学院软件研究所的董韫美院士曾对本文的初稿提出一些问题和建议,在此表示感谢.

References:

[1] Chen HM. Research on formal specification language based on CFRF [Ph.D. Thesis]. Institute of Software, The Chinese Academy of Sciences,1999 (in Chinese with English abstract).

[2] Dong YM. Collection of SAQ Report No 1-7. Technical Report, ISCAS-LCS-95-09, Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, 1995.

[3] Dong YM, Chen HM, Zhang RL. Collection of SAQ Report No 8-16. Technical Report, ISCAS-LCS-96-1, Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, 1996.

[4] Hudak P. Concept, evolution, and application of functional programming languages. ACM Computing Surveys, 1989, 21(3):359~411.

[5] Dong YM. Recursive functions of context free languages (I). Science in China (Series F), 2002,45(1):25~39.

[6] Dong YM. Recursive functions of context free languages (II). Science in China (Series F), 2002,45(2):81~102.

[7] Mitchell JC. Type inference with simple subtypes. Journal of Functional Programming, 1991,1(3):245~285.

[8] Strachey C. Fundamental Concepts in Programming Languages. Oxford: Oxford University Press, 1967.

[9] Curry HB, Feys R. Combinatory Logic. Amsterdam: North-Holland, 1958.

[10] Hindley R. The principal type scheme of an object in combinatory logic. Trans. of the American Mathematical Society, 1969, 146(1):29~60.

[11] Milner R. A theory of type polymorphism in programming. Journal of Computer and System Sciences, 1978,17(3):348~375.

[12] Cardelli L. Basic polymorphic type checking. Science of Computer Programming, 1987,8(2):147~172.

[13] Jenkins S, Leavens GT. Polymorphic type-checking in scheme. Computer Languages, 1996,2(4):215~223.

[14] Kfoury AJ, Pericas SM. Type inference for recursive definitions. In: Giuseppe L, Ecole NS, eds. Proc. of the 14th Symp. on Logic in Computer Science. Trento: IEEE Computer Society, 1999. 119~129.

[15] Zheng HJ, Zhang NX. A polymorphic type system with constraints. Chinese Journal of Computers,1999,22(4):343~350 (in Chinese with English abstract).

[16] Yang J, Michaelson G, Trinder P. How do people check polymorphic types? In: Alan FB, Eleonora B, eds. Proc. of the 12th Annual Meeting of the Psychology of Programming Interest Group. Memoria: Corigliano Calabro, 2000. 67~77.

[17] Earley J. An efficient context-free parsing algorithm. Communications of the ACM, 1970,13(2):94~102.

[18] Chen HM, Dong YM. A representation of parse tree for context-free language. Journal of Computer Research and Development, 2000,37(10):1179~1184 (in Chinese with English abstract).

附中文参考文献:

[1] 陈海明.基于上下文无关语言递归函数的规约语言研究[博士学位论文].北京:中国科学院软件研究所,1999.

[15] 郑红军,张乃孝.一种带约束的多态类型系统.计算机学报,1999,22(4):343~350.

[18] 陈海明,董韫美.上下文无关语言语法树的一种表示形式.计算机研究与发展,2000,37(10):1179~1184.

****

2004 年全国开放式分布与并行计算学术会议

征 文 通 知

由中国计算机学会开放系统专业委员会主办,北京航空航天大学计算机学院承办,北京计算机学会协办的 2004 年全国开放式分布与并行计算学术会议将于 2004 年 10 月 15 日在北京召开,有关信息如下:

一、征文范围（包括下列选题及相关内容）

1、开放式分布与并行计算模型、算法与体系结构;

2、下一代开放式网络、数据通信、网络与信息安全、业务管理技术;

3、开放式海量数据存储与 Internet 索引技术,分布与并行数据库及数据/Web 挖掘技术;

4、开放式集群计算、网格计算、Web 服务、P2P 网络及中间件技术;

5、开放式移动计算、自组网与移动代理技术;

6、分布式人工智能、多代理与决策支持技术;

7、开放式虚拟现实技术与分布式仿真;

8、开放式多媒体技术（包括媒体压缩、内容分送、缓存代理、服务发现与管理技术）.

二、征文要求

论文必须是未正式发表的、或者未正式等待刊发的研究成果;

论文格式的具体要求请见会议主页:http://ldmc.buaa.edu.cn/DPCS2004

三、重要日期与联系方式

1、论文须在 2004 年 7 月 15 日之前寄达(胡建平,牛建伟收),录用通知将在 2004 年 7 月 25 日发出.

2、联系方式:

北京航空航天大学联系人:胡建平　牛建伟

通讯地址:100083　北京航空航天大学 601 信箱

联系电话:（010）82317601　Email:niujianwei@263.net

开放系统专委会联系人:陈炳从

通讯地址:100083　北京 619 信箱 63 号

联系电话:（010）62311951