一种新型递归函数的求值算法*

陈海明+

(中国科学院 软件研究所 计算机科学重点实验室,北京 100080)

Evaluation Algorithms of a New Kind of Recursive Functions

CHEN Hai-Ming+

(Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

+ Corresponding author: Phn: +86-10-62644486, Fax: +86-10-62644486, E-mail: chm@ios.ac.cn, http://lcs.ios.ac.cn/~chm

Received 2003-09-01; Accepted 2004-01-08

Chen HM. Evaluation algorithms of a new kind of recursive functions. *Journal of Software*, 2004,15(9): 1277~1291.

http://www.jos.org.cn/1000-9825/15/1277.htm

Abstract: Recursive functions on context-free languages (CFRF) are a kind of new recursive functions proposed especially for describing non-numerical algorithms used on computers. An important research aspect of this kind of functions is the exploration of evaluation algorithms. The paper summarizes the author's research on this issue. Beginning by a discussion on possible combinations of calculation and parsing, it presents a comprehensive introduction to the major algorithms in an order in which the applicable ranges of the algorithms increase (this is also the order that the algorithms were devised). The introduction emphasizes on a new, efficient, and general evaluation algorithm, i.e. the tree-oriented evaluation algorithm. The paper also explains the evaluation method for the many-sorted recursive functions extended from CFRF. The algorithms of CFRF have been realized on computers, and have been validated by practice.

Key words: context-free language; recursive function; evaluation algorithm

摘 要: 上下文无关语言上递归函数(recursive functions on context-free languages,简称 CFRF)是为描述计算机上用的非数值算法而提出的一种新型递归函数.该函数的一个重要研究方面是函数的求值算法研究.对此问题的一些研究结果进行了总结.在讨论计算和语法分析的结合方式之后,对主要算法按照算法适用范围从小到大的顺序(同时也是算法研究和提出的顺序)做了较为全面的介绍,着重介绍一种通用的新的高效求值算法,即面向树的求值算法.同时对把 CFRF 扩充为多种类递归函数后的求值方法进行了说明.CFRF 的几个求值算法均已在机器上实现,得到了实践的检验.

关键词: 上下文无关语言;递归函数;求值算法

中图法分类号: TP18 **文献标识码**: A

上下文无关语言上递归函数(recursive functions on context-free languages,简称 CFRF)是为描述计算机上用

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

作者简介: 陈海明(1966－),男,天津人,博士,副研究员,主要研究领域为形式规约,软件设计方法,程序设计语言.

© 中国科学院软件研究所 http://www.jos.org.cn

的非数值算法而提出的一种新型递归函数[1,2].这是一种定义在上下文无关语言(Context-free language,简称CFL)上的递归函数,即函数的自变量和函数值具有CFL结构.这种函数可以看成是字上递归函数的一个自然而非平凡的扩展:其论域是具有短语结构的字.由此,CFRF能够相当直接地描述结构对象的加工算法.实验表明,许多非数值算法用这种递归函数来描述更为简捷、自然[3~7].因而CFRF为描述非数值算法提供了一种新的理论工具.

目前,CFRF的基本理论体系已经建立.在文献[1,2]中给出了CFRF的基本定义形式.其中,原始递归函数(primitive recursive functions on context-free language,简称CFPRF,是CFRF的真子类)的基本定义形式是一种联立递归式,一般函数类用极小算子定义.作为理论研究手段,这些基本定义形式已足够.在实际的应用中,可能用到各种各样的函数定义形式.为了方便CFRF的使用,文献[8]给出了CFPRF的几种其他定义形式,包括另一种联立递归式(即联立递归式-II)、多重归纳式和部分构造式,并给出它们是CFPRF的证明.由前述联立递归式和这些定义形式出发,文献[8]进一步推广得到一个函数类$CFPRF^+$,并证明$CFPRF^+$与CFRF等价.由于$CFPRF^+$的定义不使用极小算子,因此更为实用.本文的研究即针对$CFPRF^+$,下文中的CFRF是指$CFPRF^+$.

为了使CFRF能够应用于软件研究,除了理论体系的研究,还有一个重要方面是函数的求值算法研究,即如何由给定的参数值求函数值.其主要目标是算法的效率.在这方面,由于这种函数结构复杂,是一种新的函数,无法套用已有的方法,因此需要采用新的方法和技术.对这种函数的求值算法已进行过一定的研究,并针对不同的情形提出了一些方法.对于CFPRF中的联立递归式,文献[1,2]提出了一种基于语法树剪枝的自底向上求值算法.经过本文所述的定义式化归,这个算法可以推广到CFPRF的其他定义形式,因此适用于CFPRF函数类,但很难推广到一般函数类.对于CFRF,文献[9]给出的一个简单的求值算法,经本文所做的修改后,可用于只有一个递归变元的单重归纳式.本文还给出了可用于包括多重归纳式在内的各种函数定义形式的一个基本求值算法,文献[10]给出了该算法的一个改进版本.但它们离解决实际问题还有不少距离.为此,本文又提出了一种新的高效求值方法.在实际应用中,CFRF扩充为多种类递归函数往往更为有用,因此我们也研究了相应的求值算法问题.本文对主要算法做了一个较为全面和系统的介绍,并着重介绍CFRF的新的高效求值方法.

下面介绍函数的一个简单的例子,并通过这个例子介绍本文用到的一些名词术语.

例:假设二进制数的文法定义如图1所示.可定义二进制数的加一函数$Inc{:}Bin\to Bin$,如图2所示.

$$\langle Bin\rangle \to 0\,|\,1$$
$$\langle Bin\rangle \to \langle Bin\rangle 0\,|\,\langle Bin\rangle 1$$

Fig.1 A grammar for binary numbers

图1 二进制数的文法

$$Inc(0)=1 \quad Inc(1)=10$$
$$Inc(b0)=b1 \quad Inc(b1)=Inc(b)0$$

Fig.2 Function *Inc*

图2 *Inc*函数

以上例子中的文法定义了表示二进制数的一个上下文无关语言.每个产生式的右部称为该产生式左部非终极符*Bin*的一个项.如0,1,*Bin*0,*Bin*1都是*Bin*的项.不含非终极符的项称为基项,而含非终极符的项称为复合项.*Bin*的复合项中的非终极符又称为*Bin*的成分概念.

CFRF的定义采用结构归纳法,即对定义域中的一部分或全部上下文无关语言进行结构归纳.这些用来进行结构归纳的语言称为函数的归纳语言,如*Inc*函数定义域中的*Bin*.定义式中与归纳语言相应的参数称为函数的递归参数.*Inc*函数只对一个归纳语言进行归纳,这样的定义式又称为单重归纳式.反之,如果函数对多个归纳语言进行归纳,则称为多重归纳式.

本文第1节讨论在CFRF的计算中,计算和语法分析的结合.第2节介绍CFPRF的求值,包括联立递归式的自底向上求值算法和定义式的化归.第3节介绍CFRF的求值,包括单重归纳式的一个求值算法、CFRF的基本求值算法以及面向树的求值方法.第4节进一步说明把CFRF扩充为多种类递归函数时的求值方法.第5节是总结,其中将给出CFRF求值的实现情况.

为方便读者,从文献[1,2,8]摘录相关内容作为本文附录,正文不再赘述.其中附录A列出CFRF的基本定义形式,附录B列出几种其他定义形式.

© 中国科学院软件研究所 http://www.jos.org.cn

1 计算和语法分析的结合方式

我们先来分析 CFRF 的求值过程中包含哪些内容.CFRF 是定义于 CFL 上的递归函数,求值过程中包括递归函数计算的一般性问题.CFRF 的特殊性在于,CFRF 的数据是 CFL 的句子,计算中需要进行结构的识别和分解(即语法分析).因此,在求值过程中需要考虑计算和语法分析的结合方式.本节对此问题进行讨论.

设有函数 $f:L_1\times...\times L_n\to L$,其中 $L_i(i=1,\ldots,n)$和 L 都是 CFL.对于一组句子 $u_i\in L_i(i=1,\ldots,n)$,在计算 $f(u_1,\ldots,u_n)$时,如果 f 是构造定义的,则需要知道参数的结构才能把计算进行下去.因此在计算中需要嵌入语法分析.其实,在其他递归函数中也存在计算和结构分析的结合,只不过 CFL 的结构比自然数集或字集更复杂.

我们知道,在属性文法(attribute grammar,简称 AG)的实现中,属性的计算大多是在对 CFL 句子的语法分析中完成的.也有先对 CFL 句子进行语法分析,再根据分析树进行计算的(如文献[11]).我们自然可以设想,对于 CFRF,计算和语法分析的结合也有两种可能:边分析边计算和先分析后计算.所谓边分析边计算是指,在语法分析的同时完成相应函数的计算.先分析后计算是指先进行语法分析生成分析树,再进行函数计算.先计算后分析是不可能的,因为计算中必须知道数据的结构;边计算边分析在原理上是可行的,但在实现中不大可能,因为这意味着要增加结构分解原语,在递归计算的每一步,都调用这些原语对数据进行分解,而这些原语就相当于语法分析程序,这样计算,效率是可想而知的.

在 AG 中,一般使用 CFL 的子类,可以有较多的语法分析方法供选择.在 CFRF 中,对 CFL 无限制,可选用的语法分析方法相对较少.因此,CFRF 的计算和语法分析的结合还依赖于所选用的语法分析方法.

(1) 通用上下文无关文法(Context-free grammar,简称 CFG)的语法分析方法

CFRF 不对 CFG 有任何限制,文法中可以出现空字产生式.因此需要通用的 CFG 语法分析方法,并且能够处理空字.对于这样的分析方法,选择的余地很小,Earley 算法[12]能够满足需要.我们就采用 Earley 算法作为 CFRF 计算中句子的语法分析方法.Earley 算法以构造分析表列的方式找出句子的一个右分析序列,以产生式的编号表示.对输入长度为 n 的句子,最差情形下分析时间为 $O(n^3)$.

(2) 边分析边计算

根据这种方式,数据的语法分析和函数的计算同时进行.预期的好处是由于不生成完整的分析树,空间的消耗量可能较小.

且不论对一般 CFRF 能否做到边分析边计算.由于是边分析边计算,因此要求在分析方法中嵌入计算过程.Earley 算法本身并不适合于此.寻找新的通用 CFG 分析方法则有相当难度.

(3) 先分析后计算

在这种方式下,先进行数据的语法分析,再进行函数计算.优点是计算和语法分析的结合较简单,因为它们的耦合程度降低了.这也为计算的优化提供了较大的空间.缺点是空间消耗比较多.

由于通用 CFG 分析方法的选择余地很小,因此采用这种方法比较现实.

综上所述,我们选择先分析后计算的方式.

2 CFPRF 求值方法

CFPRF 是 CFRF 的真子类,计算机上用到的许多算法,都可以用这个子类中的函数来表达.同时,CFPRF 具有一个很好的性质,即函数的计算一定终止.因此,研究 CFPRF 的求值方法具有实际的意义.

2.1 联立递归式递推求值算法

CFPRF 的基本定义形式使用联立递归式(见附录 A).文献[1,2]中给出了联立递归式的一个求值算法.这个算法所进行的是对分析树的求值和剪枝过程.大致过程是,对由联立递归式定义的一组联立递归函数和给定的参数值,对函数的递归参数值进行语法分析,得到分析树,并由分析树的叶子起,自下而上由下层结点值和关联函数值求出当前结点值和函数值(函数值由函数定义得到),并剪掉当前结点的子树树枝,使当前结点成为新的叶子.重复此过程,直到求出树根的函数值.注意,由于联立递归式定义的一组函数具有一个相同的归纳语言,因此上述计算过程只需要一个分析树即可.

© 中国科学院软件研究所 http://www.jos.org.cn

我们把此方法称为函数的递推计算(或称自下而上计算),即根据句子的结构,在自下而上的归约过程中完成计算.一些简单的分析如下.

在函数的递推计算过程中,遇到待计算函数的定义中的已知函数 h_{Pi} (参见附录A中CFPRF的定义)时,需要调用 h_{Pi} 的计算过程.如果 h_{Pi} 是用构造定义式定义的,那么在这个计算过程中往往需要建立新的分析树进行求值和剪枝.在这种情况下,整个计算包括对多个分析树的求值和剪枝.

对于定义 CFPRF 的另一联立递归式(见附录 B),虽然各函数的归纳语言不同,但由于各函数联立递归,这些归纳语言必然是相互依赖的,相应的非终极符都在同一分析树上.因此,上述算法可以推广到这种递归式.

该算法的主要特点是:

(1) 一个计算过程同时计算一组联立递归函数.

(2) 一组联立递归函数的计算,是在对它们的归纳语言的一个分析树的剪枝过程中完成的.

(3) 计算过程中遇到的其他已知函数的计算,在其他计算过程中完成.

这种计算方法是以分析树为主体来控制计算过程,与 AG 自下而上计算相似,适合像 CFPRF 这样的单递归变元的函数.

递推计算的主要缺点如下:

(1) 冗余计算.在计算过程中,可能计算一些后面不用的函数值,即有冗余的函数计算.冗余的函数计算会使时间效率变低.根据 AG 的研究结果,自下而上计算中不可避免地会产生冗余计算,并且不易优化.

(2) 空间效率.冗余的函数计算不仅会对时间效率产生影响,而且每次计算中都可能生成分析树,因此也增加了空间消耗.

2.2 定义式化归

以上算法是针对联立递归式的.对于 CFPRF 的其他几种定义形式(包括多重归纳式和部分构造式),可以先化归为联立递归式,然后采用以上算法求值.这几种定义式的化归方法在文献[8]中已有形式化的详细介绍,下面我们只结合例子进行简单介绍.

(1) 部分构造式

如果函数是部分构造式,那么把它化为完全构造式(包括单重和多重归纳式).方法是根据部分构造式的定义,构造出一个等价的完全构造式.如果这个完全构造式是单重的,即它只有一个递归参数,则它已经是联立递归式,否则还要继续化归,见下面的(2).

例:设有函数 f: $Bin \to Bin$(文法如图 1 所示):

$$\begin{aligned} f(b0) &= h_1(b0) \\ f(b1) &= h_2(b1) \\ f(b) &= h_3(b) \end{aligned}$$

可以把 f 化归为以下的完全构造式 f':

$$\begin{aligned} f'(b0) &= h_1(b0) \\ f'(b1) &= h_2(b1) \\ f'(0) &= h_3(0) \\ f'(1) &= h_3(1) \end{aligned}$$

(2) 多重归纳式

如果函数是多重归纳式,那么把多个递归参数化为一个递归参数.方法是利用连接操作,把函数定义域中的多个归纳语言合为一个语言.对于 CFL $L_1,\ldots,L_n$,记

$$L = L_1 \circ L_2 \circ \ldots \circ L_n = \{a_1 \circ a_2 \circ \ldots \circ a_n \mid a_i \in L_i, 1 \le i \le n\},$$

其中∘为不在 L_i 终极符表中的分隔符,$1 \le i \le n$,称 L 为 $L_1,\ldots,L_n$ 的广义连接,即带有分隔符的连接.

例:设有二元函数 f: $Bin \times Bin \to Bin$(文法如图 1 所示):

© 中国科学院软件研究所 http://www.jos.org.cn

$$
\begin{aligned}
&f(0,0)=f(0,1)=f(1,0)=f(1,1)=0\\
&f(0,b0)=f(1,b0)=f(b0,0)=f(b1,0)=g_1(b)\\
&f(0,b1)=f(1,b1)=f(b0,1)=f(b1,1)=g_2(b)\\
&f(b_1 0,b_2 0)=f(b_1 1,b_2 1)=h_1(f(b_1,b_2))\\
&f(b_1 0,b_2 1)=f(b_1 1,b_2 0)=h_1(f(b_1,b_2))
\end{aligned}
$$

通过广义连接 $\overline{Bin}=Bin\circ Bin$,可以把 f 化归为 $f':\overline{Bin}\to Bin$. $\overline{Bin}$ 的产生式集合为 $\{\overline{Bin}\to\alpha\circ\alpha\mid\alpha\in\{0,1,Bin0,Bin1\}\}$, f' 有以下定义:

$$
\begin{aligned}
&f'(0\circ 0)=f'(0\circ 1)=f'(1\circ 0)=f'(1\circ 1)=0\\
&f'(0\circ b0)=f'(1\circ b0)=f'(b0\circ 0)=f'(b1\circ 0)=g_1(b)\\
&f'(0\circ b1)=f'(1\circ b1)=f'(b0\circ 1)=f'(b1\circ 1)=g_2(b)\\
&f'(b_1 0\circ b_2 0)=f'(b_1 1\circ b_2 1)=h_1(f'(b_1\circ b_2))\\
&f'(b_1 0,\circ b_2 1)=f'(b_1 1\circ b_2 0)=h_1(f'(b_1\circ b_2))
\end{aligned}
$$

由于各种 CFPRF 的定义形式都可以化归到联立递归式,它们可以采用上述求值算法.因此,CFPRF 的求值算法可以只针对联立递归式.

在对一个函数进行上述变换时,还要修改在各函数定义中出现的此函数的应用形式.在计算表达式时,也要对表达式进行改造.因此,需要有函数变换的开销.由于参数在合并时需要加入分隔符,使句子长度增加,文法也相应变复杂,因此会带来语法分析时间的增加.

3 CFRF 求值方法

以上讨论了 CFPRF 的求值方法.对于 CFRF,由于在等式的右端,待计算函数的参数可以是函数,能否递推计算需要根据不同的情况来讨论.总的说来,一般情况下,函数的计算顺序与分析树的自下而上归约顺序不一致,难以通过分析树的归约完成函数计算.

此问题的一般性提法是研究计算顺序的反转,已有许多研究人员针对自然数上递归函数的情况进行过研究(例如文献[13]).这些研究对一些受限形式的函数(如线性递归函数)提出了一些策略.对于 CFRF 中的函数,也可以研究能够进行计算顺序反转的函数形式.但这并不能代替 CFRF 的求值方法,而只能作为 CFRF 求值方法的一种优化.本文不打算就此问题进行深入的讨论.

因此,对于 CFRF 需要另行寻找其他实现方法.下面我们先介绍 CFRF 中单重归纳式的求值方法,然后介绍可用于各种函数定义形式的通用求值方法.

3.1 单重归纳式的求值方法

理论上,单重归纳式可定义所有函数,多重归纳式也可化归为单重归纳式,因此,研究单重归纳式的求值算法是有意义的.由于只有一个递归参数,因此在求值上也相应地简化.文献[9]中给出了一个只有一个递归参数的完全构造式的求值算法,此处略作修改,使该算法还可以用于部分构造式.

设有 n 元函数 $g:L_1\times\ldots\times L_n\to L$,归纳语言为 L_1,L_1 由文法 $G_1=(V_N,V_T,X_1,P_1)$ 产生.

对于 $L_1\times\ldots\times L_n$ 的一组句子 $u_1,\ldots,u_n$,在计算 $g(u_1,\ldots,u_n)$ 时,首先对 u_1 按 L_1 的文法进行分析,得到以 P_1 中产生式编号表示的右分析序列 $i_1 i_2\ldots i_m$,即线性表示的分析树.u_1 的右分析序列有两种情况:

① 只有一个产生式 i_1.此时产生式 i_1 必定是 $X_1\to u_1$,从 g 的定义式中找到结构模式与 u_1 对应的等式(由 g 的结构归纳定义方法,这个等式必然存在),将右端表达式 e 中的变元替换为相应的值.若表达式 e 中没有除了连接算子以外的函数,则得到计算结果.若有这样的函数,则找到定义,继续进行计算.

② 有多个产生式.此时产生式 i_1 是 $X_1\to P$,P 是 X_1 的复合项.从 g 的定义式中找到结构模式与 P 匹配的等式,将右端表达式 e 中的变元替换为相应的值.并且,若与 P 匹配的是 L_1 的变元,则将右分析序列 $i_1 i_2\ldots i_n$ 记入此变元,否则,设 P 中含有成分概念 $x_1,\ldots,x_r$,由 $i_2\ldots i_m$ 容易求出 $x_1,\ldots,x_r$ 各自对应的子树的右分析序列和值 $R_1,\ldots,R_r$,成分概念的子树右分析序列分别记入表达式 e 中相应变元.若表达式 e 中没有除了连接算子以外的函数,则可以得到计算结果.若有这样的函数,则找到定义,继续进行计算.

设表达式 e 中有函数 g',如果 g'的归纳语言对应于 X_1 的成分概念 x_i,即 g'归纳语言值为 R_i,则可用 x_i 对应的

© 中国科学院软件研究所 http://www.jos.org.cn

右分析序列继续进行上述计算.如果 g'的归纳语言不对应于 X 的成分概念,则对 g'的计算需要重新进行结构分析.对于嵌套的函数(即函数的参数是函数),外层函数也无法利用传下来的分析序列.只有内层函数才有可能用得着传下来的分析序列.可见,在计算过程中,可能经常地调用语法分析程序对句子进行分析.另外,基本运算的计算无须对句子进行语法分析.

实际上,在 CFRF 中单重归纳式的计算与 AG 最为接近,因为两者最为相似.多重归纳式由于对定义域中的多个 CFL 进行归纳,在计算中需要对付多个分析树,因此比 AG 的计算更为复杂.

3.2 可用于多重归纳式的通用求值方法

虽然单重归纳式具有足够强的表达能力,多重归纳式也可以化归为单重归纳式,但是在实用中还是直接使用多重归纳式更为方便.因此,还应研究可直接用于多重归纳式的算法.从本节开始,我们研究可用于包括多重归纳式在内的各种函数定义形式的通用求值算法.

3.2.1 一种语言

为方便以后的研究,我们首先定义一种表示这种函数的语言 L_{CFRF},作为函数的语言模型.如图 3 所示(其中“·”是连接算子,$\overline{p}$ 表示模式的一个序列 $p_1,\ldots,p_n$).

c	Constant
x	Variable
f	Function name
$e\ ::=c\|x\|e_1\cdot e_2\|f(e_{1\ldots}e_n)$	Expression
$p\ ::=c\|x\|p_1\cdot p_2$	Pattern
$prog ::= f_1(\overline{p_{1,1}})=e_{1,1}$	Program
$f_1(\overline{p_{1,m_1}})=e_{1,m_1}$	
$\ldots$	
$f_n(\overline{p_{n,1}})=e_{n,1}$	
$f_n(\overline{p_{n,m_n}})=e_{n,m_n}$	

Fig.3 Syntax of LCFRF

图 3 语言 LCFRF 的语法

任意 n 元函数 $f:L_1\times\ldots\times L_n\to L$ 的一般形式可以表示如下:

$$f_1(p_1^1,\ldots,p_1^n)=e_1$$
$$\vdots$$
$$f_1(p_m^1,\ldots,p_m^n)=e_m$$

其中 p_i^j 称为 L_j 的结构模式,它或者对应于 L_j 的项(即把 p_i^j 中的变量替换为相应的非终极符后,就是 L_j 的一个项),或者是类型 L_j 的变量;e_i 是表达式.

函数 f 的求值规则如下:

当计算函数应用 $f(v_1,\ldots,v_n)$ 时,其中 v_i 是 L_i 的值,对 f 的等式自上而下逐个检查,选择满足下列条件的第 1 个等式(设为等式 i):

p_i^j 或者是 L_j 的变量,或者对应于 v_j 的推导序列中第 1 个产生式的右部.

以上规则即函数的结构模式匹配规则.

对于这样的函数,在计算中需要同时有多个分析树,因此,以函数定义为主体来控制计算过程的自上而下递归计算是比较合适的选择.

3.2.2 基本求值算法

本节介绍 L_{CFRF} 的一个基本求值算法,旨在解释求值的基本原理.

基本求值算法如下:

Eval(e,env)
输入:表达式 e,环境 env.
输出:结果表达式.
Begin
case e of

© 中国科学院软件研究所 http://www.jos.org.cn

1. 常量,返回 e;
2. 变量,返回 value(e,env);
3. 连接 $e_1 \cdot e_2$,返回 Eval(e_1)·Eval(e_2);
4. n 元函数应用 $f(e_1,...,e_n)$,
   (a) val_i=Eval(e_i,env),i=1,…,n;
   (b) 对 val_i 进行类型检查,i=1,…,n;
   (c) $newenv$=nil,val_i,i=1,…,n 与函数定义中的等式进行模式匹配,得到匹配的等式,各结构变量和约束值加入 $newenv$;
   (d) 设匹配等式的右部表达式为 e',val=Eval(e',$newenv$),释放 $newenv$;
   (e) 返回 val;

endcase

End

其中的计算采用 eager 求值方式,或称 strict 计算,即先求参数值再计算函数值的按值调用方式.类型检查是通过对参数进行语法分析完成的,即判断参数值是否相应 CFL 的句子.函数返回值没有进行类型检查,对于中间函数值,可作为其他函数的参数进行类型检查,对于最终函数值,如果有必要,只需在计算结束之后再进行一次类型检查即可.进行类型检查后,可以得到相应的分析树.

模式匹配所进行的是比较当前分析树的形和函数定义的各等式中相应参数的结构模式,并选取匹配的等式.分析树的形由树根和根的所有子结点决定.当函数有多个递归参数时,就需要有多个分析树,并同时对多个分析树进行模式匹配.

以上的方法实现了函数的递归计算,计算过程较为清晰、自然.此方法的主要优点有:

(1) 计算的过程由函数定义驱动,没有在函数递推计算时所产生的冗余计算.

(2) CFRF 各种函数定义形式都可以实现,是实现 CFRF 的一种通用的方法.

上述基本求值算法的一个改进版本参见文献[10],此处不再赘述.

3.3 面向树的求值方法

前述的基本求值算法是用句子计算的形式表述的.即计算的中间结果(参数值和函数值)均以字符串的形态存在,并且用语法分析实现动态的类型检查.由于 CFL 是字符串的集合,而如前所述,该算法主要用于显示函数的语义和求值原理方面,因此这样的计算是自然的.当用于实现时,直接采用这种句子计算存在的主要问题是,计算中有大量的动态语法分析,从而影响计算效率.该算法产生动态语法分析的原因很多,比如一个主要原因是函数嵌套,即一个函数的值是某个函数的参数.该算法之所以产生大量的动态语法分析,主要原因是没有利用在函数定义中所包含的结构信息.

为了提高计算效率,需要在上述算法的基础上,研究新的算法,减少动态语法分析次数.设计新算法的主要依据是:

(1) 在计算中,我们只需要最终函数计算结果的句子值.因此,只要计算过程能够继续下去,中间结果是否有句子值并不重要.

(2) 函数定义的表达式中存在结构信息.函数计算是结构的分解和重构过程.通过充分利用表达式中的结构信息,可以使计算在结构的层次上进行,在许多情况下就可以避免函数中间结果的动态语法分析.

基于上述考虑,我们把 CFRF 的计算视为分析树的计算.即函数计算以一组分析树为输入,计算的结果是一个分析树,它所对应的句子就是所需函数值.如图 4 所示.如此,在计算中所有中间结果都以树的形式存在.计算以分析树为加工单位,通过树的分解、合并操作完成.一个分析树对应的句子也称作分析树的值.在树的分解、合并过程中,一般不必计算分析树的值.在需要时,也可随时计算树的值.我们把这样的计算方法叫做面向树的求值方法.下面先介绍这种方法的主要过程,然后介绍其中用到的关键技术.

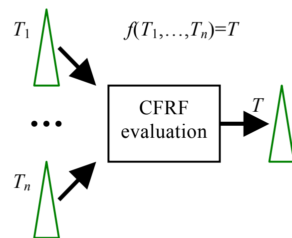

Fig.4 Tree-Oriented CFRF evaluation

图 4 面向分析树的 CFRF 计算

3.3.1 函数计算的基本过程

面向树的计算方法是以分析树为计算对象实现函数的递归计算.因此,此方法的关键是利用函数定义的表达式中存在的结构信息,建立起在计算时可直接把函数计算结果表示为树的机制.亦即可以通过子树的合并和重构,构造起与函数返回类型一致的分析树.为此,只需为表达式预先建立起满足上述需求的树结构即可.在计算时,通过将表达式中的变量和函数值替换为相应的子树,就得到了完整的分析树.这样就减少了动态语法分析(一般情况下不能完全避免动态类型检查,参见第 3.3.2.2 节).

实现这种方法,首先要给出 L_{CFRF} 的可以表示分析树结构的中间表示形式(IR).其次,要能够把原来的表示转换为这种树结构的中间表示.然后是对这种中间表示的函数进行面向树的计算.最后要把计算结果还原为句子.

按照这种方法,函数计算的主要步骤如下(如图 5 所示):

(1) 表达式重构.在计算之前,根据函数的声明类型,把函数定义中的表达式重构为具有树结构的表达式.由于表达式的重构需要根据其对应的类型进行,此问题还与类型系统密切相关.

(2) 面向树的函数计算.进行函数计算,计算过程与上一节中的求值方法相似,只不过计算对象变成了分析树.即在函数的递归求值过程中,通过树的结构操作完成计算.对树的结构操作主要有树的匹配、分解和合并.树的匹配是指识别分析树的形.树的分解是指取得分析树的子树,所得子树是进一步计算的对象.树的合成是由分析子树合成分析树,所得分析树作为当前计算的结果.计算中只在需要时才进行类型检查.计算完成后,得到一棵作为计算结果的分析树.

(3) 求树的值.由分析树求出句子值.

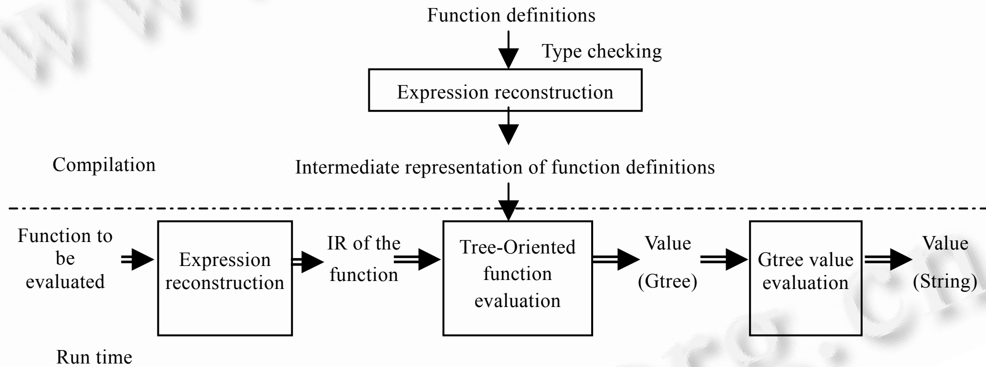

Fig.5 CFRF evaluation procedure

图 5 CFRF 计算过程

3.3.2 关键技术

本节介绍这种方法的一些关键技术.

3.3.2.1 分析树和表达式的中间表示

L_{CFRF} 的表达式是通过连接函数形成的.这种以连接为基础的表达式非常直观,格式紧凑,使用方便,有很灵活的表达能力.但是另一方面,这种表达式的一个基本特点是,表达式的形式反映不出它所具有的 CFL 类型.换句话说,是一种隐式结构的表示.因此,首先需要有一种能显式表示出分析树结构的表达式的表示形式.对分析树的表示形式有两个要求,一是使表达式能表示出树结构,二是高效.作为函数的加工对象,分析树在函数计算中可能同时会存在多个,空间效率问题因此较为突出.有两种途径可以减小分析树的大小.一种途径是优化文法.即通过对文法的变换达到削减分析树的目的.但是这种方法要改变文法的结构,与我们的要求不符.另一种途径是设计高效的数据表示.分析树的表示还要有助于时间效率的提高.

根据面向树的函数计算的特点,我们设计了分析树的一种紧凑的表示形式,称为 G 树.有关 G 树的详细介绍参见文献[14],这里对 G 树的定义和主要特点等做一个简单介绍.

首先,假定在一个文法中,每一个产生式对应于唯一的标识,称为产生式的标号.如果产生式 $X\rightarrow\alpha$的标号为 p,则记为 $p{:}X\rightarrow\alpha$.

© 中国科学院软件研究所 http://www.jos.org.cn

为描述方便,我们在表示树时常采用带括号的线性表示形式.

定义 1. 对于一个 CFG $F=(V_N,V_T,S,P)$,以下定义的树称为 F 的 G 树.

(1) 若有产生式 $p:X\to a,a\in V_T^*$,则 p 是根为 X 的 G 树.

(2) 若有产生式 $p:X\to a_1X_1\ldots a_nX_na_{n+1}$(各 $a_i\in V_T^*$,各 $X_i\in V_n$),且 t_i 是根为 X_i 的 G 树,$i=1,\ldots,n$,则 $p(t_1\ldots t_n)$是根为 X 的 G 树.

(3) 除此之外,无其他 G 树.

文法 F 的一个 G 树表示了 F 的一个非终极符的推导序列,它所对应的字称为 G 树的值.

如果 G 树 T 的根为 S,T 的值是 w,则称 T 是 F 关于 w 的 G 树.

G 树的主要特点是,用产生式标号代替非终极符作为树的结点,并去除了分析树中的叶结点.可见,G 树是精简了的分析树,但并不等价于分析树,因为 G 树中忽略了一些信息,G 树和产生式一起才等同于分析树.

G 树的结构匹配只需访问根结点,相当方便和高效;G 树的结构分解和合成也都很方便.虽然求 G 树的值不如传统的分析树那样直接,但是在函数计算中,主要的操作是结构操作,而不是值的操作,因此 G 树正好适合.另一方面,对于需要标记非终极符属性的情况,G 树则不像传统分析树那样方便.

以下的算法给出了求 G 树的值的过程.

GtreeVal(T):由 G 树 T 求 T 的值.

$$\begin{cases}\text{GtreeVal}(p)=a, & \text{其中}p:X\to a,a\in V_T^*\\ \text{GtreeVal}(p(t_1,\ldots,t_n))=a_1\text{GtreeVal}(t_1)\ldots a_n\text{GtreeVal}(t_n)a_{n+1}, & \text{其中}p:X\to a_1X_1\ldots a_nX_na_{n+1},a_i\in V_T^*,X_i\in V_N\end{cases}.$$

可见,G 树在求值时需要有产生式.

以 G 树为基础,我们得到表达式的中间表示,称为 V 树.以下给出 V 树的定义.

定义 2. **V 树**.

(1) 类型为 X 的变量 v 是根为 X 的 V 树.

(2) 若有产生式 $p:X\to a,a\in V_T^*$,则 p 是根为 X 的 V 树.

(3) 若函数 $f:X_1\times\ldots\times X_n\to X$,且 t_i 是根为 X_i 的 V 树,$i=1,\ldots,n$,则 $f(t_1,\ldots,t_n)$是根为 X 的 V 树.

(4) 若有产生式 $p:X\to a_1X_1\ldots a_nX_na_{n+1}$(各 $a_i\in V_T^*$,各 $X_i\in V_n$),且 t_i 是根为 X_i 的 V 树,$i=1,\ldots,n$,则 $p(t_1\ldots t_n)$是根为 X 的 V 树.

(5) 此外无其他 V 树.

如果一个 V 树的根为 X,则称此 V 树的类型是 X.V 树的类型唯一,而值可以不唯一.

当采用基于 V 树的中间表示形式时,函数计算由相对独立的两个步骤组成,即 V 树计算为 G 树以及求 G 树的值.表达式被表示为 V 树,V 树的计算结果就是 G 树,换句话说,G 树是表达式的值的结构化表示.G 树求值方法见上.

3.3.2.2 类型检查和表达式表示形式的转换

为了把表达式转换为树结构表示形式,需要根据表达式的结构和所对应的类型进行分析.这项工作可以结合类型检查完成.

CFL 之间可以有相等、包含或相交关系,因此含有 CFL 类型的类型系统与 Milner 系统[15]不同,而更接近于 subtyping 系统[16].然而,CFL 类型之间的相等、包含、相交关系是不可判定的.因此,含有 CFL 类型的类型系统本质上是动态类型系统,即类型检查需在程序运行时才能完成.但是,如前所述,动态类型检查对于函数计算的时间效率影响较大.解决 CFL 类型问题有两种方法,一是对表达式加以限制,使得对限制后的函数定义可以进行静态类型检查.这样可以提高时间效率,但是会降低表达式的表达能力和灵活性.

另一种方法是采用动态类型系统,同时尽可能多地进行静态类型检查.与前一种方法相比,这样会降低时间效率,但却比完全的动态类型检查高效得多.为了保持表达能力的最大的灵活性,我们选择了这种方法,并设计了一个实用的类型系统.

这个类型系统的主要特点是,虽然 CFL 的包含关系不可判定,但是它的一个子集,即句型关系,是可以判定的.因此对这个子集可以进行静态的判断,而对其余部分仍采用动态类型检查.一个 CFL 的句型是从此 CFL 的文

© 中国科学院软件研究所 http://www.jos.org.cn

法可以推导得到的任何串,这个串的字符包括文法的非终极符和终极符.根据实验统计数据,实际函数定义中的表达式绝大部分都满足句型关系.因此,虽然静态类型检查并不完备,但是它在实际应用中可以承担绝大部分甚至全部的类型检查工作,因此运行时间效率得到有效提高.

进行类型检查时的一项重要工作,是进行表达式结构的重构,建立起树结构.对于可进行静态类型检查的表达式,在静态类型检查时进行结构转换.对于其他表达式,则在运行时对表达式的值建立树结构.

有关 CFRF 的类型系统、类型检查算法和表达式重构方法,在文献[17]中有较为详细的描述.CFRF 类型系统的一个简单介绍参见文献[18].有关内容此处不再赘述.

例:以图 1 和图 2 的 *Inc* 函数为例.重写 *Bin* 的文法如下:

$$p0:\langle Bin\rangle\to 0\quad p1:\langle Bin\rangle\to 1\quad p2:\langle Bin\rangle\to\langle Bin\rangle 0\quad p3:\langle Bin\rangle\to\langle Bin\rangle 1$$

经过类型检查,*Inc* 具有如下形式:

$$Inc(p0)=p1\qquad Inc(p1)=p2(p0)$$
$$Inc(p2(b))=p3(b)\qquad Inc(p3(b))=p2(Inc(b))$$

3.3.2.3 模式匹配

在函数计算中,模式匹配是一项基本的工作,其效率对于计算效率有很大影响.模式匹配是通过比较分析树的形与模式的结构进行的.对于一般的分析树结构,需要比较树根结点的所有子结点,效率较低.而对于 G 树,只需要比较根结点即可,因此可以达到较高的效率.

CFRF 的定义方法是对定义域中的 CFL 进行结构归纳.所得函数定义中的模式是无嵌套的模式,称为简单模式.根据 CFRF 的特点,我们提出了一种编码模式匹配方法.这种方法实现简单,对于 CFRF 一般有较好的效率.有关这一方法的详细介绍参见文献[19].但当函数定义中的模式较为稀疏时(即大多数参数为所对应的归纳语言的变元),编码匹配方法的效率就可能不如传统的顺序匹配方法.为此,文献[20]在保留模式编码的基础上,提出了一种顺序匹配方法.两种方法可结合用于 CFRF 的模式匹配.此处对这两种方法不再赘述.

3.3.3 计算过程的扩充

对图 5 的计算过程还可以进行扩充.函数定义经表达式重构后,可以继续变换,使函数定义变换为效率越来越高的形式.这一过程,即为函数定义的优化过程.在函数定义用原始的隐式结构表示时,函数优化极为不便.例如,此时两个具有相同字面形式的表达式,却不一定对应于相同的分析树.函数定义重构为显式结构后,为优化提供了广阔空间.如公共子表达式的提取、消除冗余解构-重构对等.许多已有的优化技术都可以作为参考.

在增加函数变换后,计算过程如图 6 所示.

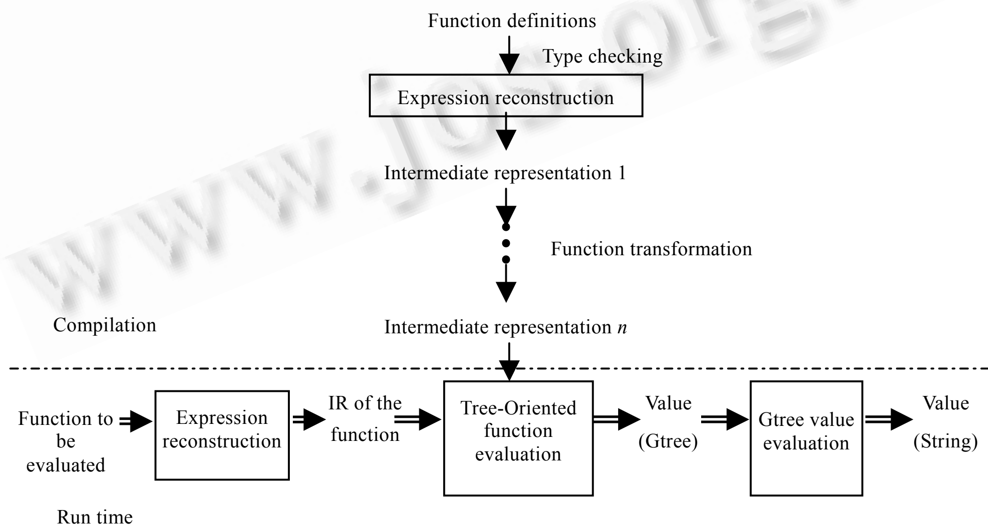

Fig.6 CFRF evaluation procedure (with function transformation)

图 6 CFRF 计算过程(带有函数变换)

© 中国科学院软件研究所 http://www.jos.org.cn

4 扩充到多种类递归函数的求值方法

以 CFRF 为基础,在 CFRF 中加入常用的一些基本类型,如数、字符、字、布尔类型,就得到一种在实际应用中更为有用的多种类递归函数.在理论上,这种函数仍然可适用于 CFRF 理论,因为基本类型都可以用 CFL 来表示.在算法上,则不把基本类型当作 CFL.对于多种类递归函数,在面向树的计算方法中,适当扩充,把基本类型作为一类需要专门处理的特殊对象,在分析树中增加特殊结点,用于在计算中保存基本类型及其值,并增加专门针对基本类型的操作,此方法即可用于这种函数.限于篇幅,具体内容此处不再赘述.

5 总 结

本文详细介绍了CFRF的求值算法.其中,着重介绍了一种新的可用于CFRF的各种定义形式的高效求值算法,即面向树的求值算法.并介绍了把 CFRF 扩充为多种类递归函数后的求值方法.实际上,我们已把 CFRF 用作一种可执行形式规约的基础,实现了一个形式规约获取系统 SAQ(specification acquisition)[21],其中包括对以上介绍的数种求值算法的实现.以下对实现情况作一个简单介绍.

SAQ 系统提供一种函数式语言 LFC(language for CFRF)[22]作为 SAQ 的形式规约语言.最初的 LFC 是基于 CFRF 的单重归纳式的,现在 LFC 则是基于多种类递归函数.LFC 语言有一个函数构造和检验系统 FC[9],它既是 SAQ 的一个子系统,又是一个可脱离 SAQ 单独运行的独立系统.FC 系统可以辅助用户进行函数定义的构造,还带有 LFC 语言的解释器.随着算法研究的深入,LFC 的解释器也具有了几个版本.最初的版本实现了 CFRF 的单重归纳式的求值算法[9];后来又实现了 CFRF 的一个通用求值算法[10];面向树的求值方法提出后,又得到了一个新的版本[14,17,19].根据面向树的求值方法,还实现了 LFC 语言的一个编译器[23].已经使用 SAQ 系统和 LFC 语言进行了许多实验,其中包括一些非平凡的例子[21].通过这些实验,对算法进行了检验.实验表明,采用面向树的求值方法,使函数的计算效率有了显著提高.例如,在对字符串的插入排序、初等函数的形式微分和 Fibonacci 函数等例子的实验中,采用面向树的求值方法的解释器,速度比采用前述通用求值算法的解释器提高了 2.6 倍~22.5 倍不等,平均提高 11.5 倍.计算效率的提高幅度主要与函数定义可进行静态类型检查的程度有关.静态的类型检查越多,所需动态类型检查越少,速度提高越多.另外,采用新的中间表示和新的实现技术也使速度有不少的提高.

用来描述算法的 CFRF,或其拓展成的多种类递归函数,具有丰富的数据类型,其复杂程度远远超过了自然数的递归函数,而更接近于一个编程语言系统.因此,其求值问题已不是一个单纯的数学问题,而是一个包含许多技术问题的综合性的语言实现问题.这正如λ-演算那样.进一步的工作,是对面向树的求值方法的完善和研究函数求值的优化技术.

致谢 本文的工作属于由董韫美院士领导的 SAQ 研究计划的一部分.我们对董韫美院士给予的指导和支持以及 SAQ 小组同事们的支持与合作表示感谢.

References:

[1] Dong YM. Recursive functions of context free languages (I)—The definitions of CFPRF and CFRF. Science in China (F), 2002, 45(1):25~39.

[2] Dong YM. Recursive functions of context free languages (II)—Validity of CFPRF and CFRF definitions. Science in China (F), 2002,45(2):1~21.

[3] Dong YM, *et al*. Collection of SAQ reports no. 1-16. Technical Report, ISCAS-LCS-96-1, Beijing: Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, 1996.

[4] Hu X, Cao J, Wang L, Cai Z, Cao Y, Pan, HJ, Li WX. A specification base of algorithms on basic data structures for SAQ. Technical Report, ISCAS-LCS-97-02, Beijing: Laboratory of Computer Science, Institute of Software, the Chinese Academy of Sciences, 1997 (in Chinese with English abstract).

© 中国科学院软件研究所 http://www.jos.org.cn

[5] Wan ZY. Design and implementation of the music processing prototyping system SMART. Technical Report, ISCAS-LCS-97-19, Beijing: Laboratory of Computer Science, Institute of Software, the Chinese Academy of Sciences, 1997 (in Chinese with English abstract).

[6] Chen ZM. Using SAQ for programming language convertor. Technical Report, ISCAS-LCS-97-20, Beijing: Laboratory of Computer Science, Institute of Software, the Chinese Academy of Sciences, 1997 (in Chinese with English abstract).

[7] Wang L, Wu LH, Zhou HF, Xie LJ, Gu CL, Zhang X, Li WX. Implementing Java to C++ conversion using SAQ. Technical Report, ISCAS-LCS-99-01, Beijing: Laboratory of Computer Science, Institute of Software, the Chinese Academy of Sciences, 1999 (in Chinese with English abstract).

[8] Chen HM, Dong YM. Definition forms of recursive functions defined on context-free languages. Technical Report, ISCAS-LCS-99-15, Beijing: Laboratory of Computer Science, Institute of Software, the Chinese Academy of Sciences, 1999 (in Chinese with English abstract).

[9] Chen HM. Design and implementation of function construction and checking system FC. Journal of Software, 1998,9(10):755~759 (in Chinese with English abstract).

[10] Chen HM. Function definition language FDL and its implementation. Journal of Computer Science and Technology, 1999,14(4): 414~421.

[11] Jourdan M. FNC-2. In: Deransart P, Jourdan M, Lorho B, eds. Attribute Grammars. LNCS 323, Heidelberg: Springer-Verlag, 1988. 78~82.

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

[13] Boiten EA. Improving recursive functions by inverting the order of evaluation. Science of Computer Programming, 1992,18(2): 139~179.

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

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

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

[17] Chen HM, Dong YM. Practical type checking of functions defined on context-free languages. Technical Report, ISCAS-LCS-2k-08, Beijing: Computer Science Laboratory, Institute of Software, The Chinese Academy of Sciences, 2000.

[18] Chen HM, Dong YM. Practical type checking of functions defined on context-free languages. In: Wu ZH, *et al*. eds. Proc. of the 6th Int'l Conf. for Young Computer Scientists (ICYCS 2001). Beijing: International Academic Publishers, World Publishing Corporation, 2001. 1261~1263.

[19] Chen HM, Dong YM. Pattern matching compilation of functions defined on context-free languages. Journal of Computer Science and Technology, 2001,16(2):159~167.

[20] Zhang Q, Chen HM. A pattern matching method for simple patterns. Computer Engineering and Applications, 2001,37(17): 63~66 (in Chinese with English abstract).

[21] Dong YM, Li KD, Chen HM, Hu YQ, Zhang RL, Tang RQ, Wan ZY, Chen ZM. Design and implementation of the formal specification acquisition system SAQ. In: Feng YL, Notkin D, Gaudel MC, eds. Proc. of the Conf. on Software: Theory and Practice, IFIP 16th World Computer Congress 2000. Beijing: Publishing House of Electronics Industry, 2000. 201~211.

[22] Chen HM, Dong YM. A formal specification language facilitating specification acquisition. Chinese Journal of Computers, 2002,25(5):459~466 (in Chinese with English abstract).

[23] Chen HM. Compilation of a kind of recursive functions. Technical Report, ISCAS-LCS-2k-08, Beijing: Computer Science Laboratory, Institute of Software, the Chinese Academy of Sciences, 2000 (in Chinese with English abstract).

附中文参考文献:

[4] 胡晓,曹骏,王琳,蔡植,曹勇,潘洪军,李万学.SAQ 系统的一个数据结构算法的规约复用库.技术报告,ISCAS-LCS-97-02,北京:中国科学院软件研究所计算机科学重点实验室,1997.

[5] 万战勇.SMART 音乐处理原型系统的设计和实现.技术报告,ISCAS-LCS-97-19,北京:中国科学院软件研究所计算机科学重点实验室,1997.

© 中国科学院软件研究所 http://www.jos.org.cn

[6] 陈自明.SAQ 系统用于程序语言的转换器.技术报告,ISCAS-LCS-97-20,北京:中国科学院软件研究所计算机科学重点实验室,1997.
[7] 王琳,吴丽辉,周昊芳,谢禄江,顾灿龙,张旭,李万学.用 SAQ 系统实现 JAVA 语言向 C++语言的转换.技术报告,ISCAS-LCS-99-01,北京:中国科学院软件研究所计算机科学重点实验室,1999.
[8] 陈海明,董韫美.上下文无关语言递归函数的定义形式.技术报告,ISCAS-LCS-99-13,北京:中国科学院软件研究所计算机科学重点实验室,1999.
[9] 陈海明.运算构造和检验系统的设计和实现.软件学报,1998,9(10):755~759.
[14] 陈海明,董韫美.上下文无关语言分析树的一种表示形式.计算机研究与发展,2000,37(10):1179~1184.
[20] 张强,陈海明.简单模式的一种匹配方法.计算机工程与应用,2001,37(17):63~66.
[22] 陈海明,董韫美.一个支持规约获取的形式规约语言.计算机学报,2002,25(5):459~466.
[23] 陈海明.一类递归函数的编译实现.技术报告,ISCAS-LCS-2k-08,北京:中国科学院软件研究所计算机科学重点实验室,2000.

附录

A 上下文无关语言递归函数的基本定义形式

A.1 上下文无关语言原始递归函数(简称CFPRF)[1,2]

定义. 函数 $f:L_1\times...\times L_n\to L$ 是 CFPRF,如果 $L_1,\ldots,L_n,L$ 均是 CFL,L_i 由上下文无关文法 $G_i=(V_N,V_T,X_i,P_i)$ 产生,L 由 $G=(V_N,V_T,X,P)$ 产生,所涉及的文法均无二义;f 系由以下步骤生成,而且也只由此所生成:

(1) f 是基本函数:(a) 常字函数 $const_w(x_1,\ldots,x_m)=w,w\in L$;(b) 投影函数 $U_i^m(x_1,...,x_m)=x_i, 1\le i\le m$;(c) 连接函数 $concate(x_1,\ldots,x_m)=x_1...x_m$.

(2) f 是有穷次使用以下算子得到的:

(a) 代入算子.即由 $g_i:L_1\times...\times L_n\to L^i$,$i=1,\ldots,m$ 和 $h:L^1\times...\times L^m\to L$,得到 $f:L_1\times...\times L_n\to L$.其中 $f(x_1,\ldots,x_n)=h(g_1(x_1,\ldots,x_n),\ldots,g_m(x_1,\ldots,x_n))$.

(b) 联立递归式.设要定义的是 m 个函数 $f_1,...,f_m:L_1\times...\times L_n\to L$,L_1 由 $G_1=(V_N,V_T,X_1,P_1)$ 产生,称此 m 个函数是由联立递归式所定义的,如果对每个 $P\in Term(X_1)$($Term(X_1)$表示 X_1 的项的集合),有 CFPRF 类中的 m 个已知函数 $h_{Pi},i=1,\ldots,m$,使得

i. 当 P 中不含非终极符时,有规则组

$$\begin{cases}f_i(P,y_2,...,y_n)=_{df}\ h_{Pi}(y_2,...,y_n)\\ i=1,...,m.\end{cases}\tag{*}$$

ii. 当 P 中含非终极符时,设 $P=u_0Z_1u_1\ldots Z_ru_r$,其中 $u_0,\ldots,u_r$ 为终极符串(可以是空串),$Z_0,\ldots,Z_r$ 为非终极符,则有规则组

$$\begin{cases}f_i(P,y_2,...,y_n)=_{df}\ h_{Pi}(Z_1,...,Z_r,y_2,...,y_n,...,f_k(Z_j,y_2,...,y_n),...)\\ i,k=1,...,m;1\le j\le r\end{cases}\tag{**}$$

上式 h_{Pi} 中,我们把非终极符视为与自变元等同,并在由该非终极符代表的 CFL 中取值.注意,同一个非终极符可以多次出现,各个出现均被认为不同.h_{Pi} 式中的未知函数 f_k 亦可不出现,但若出现,其首自变元位置对应的必须是 X_1.

对所有 $P\in Term(X_1)$,分别按照上述情形取规则组(*)或(**)的全体,便是联立递归式.

只有一个待定义未知函数的联立递归式,即 $m=1$ 的情形,称为原始递归算子.

A.2 上下文无关语言递归函数(简称CFRF)

定义. 设有谓词 $f: L_0\times L_1\times...\times L_n\to L$,$\lambda\in L$(λ表示空字),则极小算子$\mu_y[f]$求得的值定义为集合

$$\{z|z\in L_0,使得\ f(z,x_1,\ldots,x_n)=\lambda\}$$

中长度最短或者构造最简的 z.如果集合为空,则值是未定义的.

这样的 z 决定于 f,并且是 $x_1,\ldots,x_n$ 的函数,即$\mu_y[f(y,x_1,\ldots,x_n)]$定义了一个函数$\phi: L_1\times...\times L_n\to L_0$,$z=\phi(x_1,\ldots,x_n)$,

以后记为$\phi(x_1,\ldots,x_n)=\mu_y[f(y,x_1,\ldots,x_n)]$.

定义. 函数 $f: L_1\times\ldots\times L_n\to L$ 是 CFRF,如果满足 CFPRF 定义中的条件,此外,在f的构造过程中还允许使用极小算子$\mu_y[f]$.

已证明 CFPRF 是 CFRF 的真子类,并且证明 CFRF 类与上述的递归函数类是等价的.

B 上下文无关语言递归函数的其他定义形式

为简化表达,本节采用如下记号.用 $v(\alpha)$表示α中的非终极符排列成一个序列,中间用逗号隔开.即若$\alpha=v_1V_1v_2V_2\ldots v_nV_nv_{n+1}$,其中各 v_i 是终极符串,各 V_i 是非终极符,则 $v(\alpha)=V_1,\ldots,V_n$.

B.1 另一形式的联立递归式

设有 m 个函数 $f_i: L_{i,1}\times L_2\times\ldots\times L_n\to L^i$ \$,$i=1,\ldots,m$, 其中 $L_{i,1}=L(G_{i,1})$, $G_{i,1}=(V_N,V_T,X_{i,1},P_{i,1})$, $L_j=L(G_j)$, $G_j=(V_N,V_T,X_j,P_j)$,$L^i=L(G^i)$, $G^i=(V_N,V_T,X^i,P^i)$,$i=1,\ldots,m$,$j=2,\ldots,n$.各文法均无二义.

对于 $L_{i,1}$,有 $\alpha_{i,s}\in Term(X_{i,1})$,其中 $s=1,\ldots,t_i$,t_i 为 $Term(X_{i,1})$元素数,且当 $s\le k_i$ 时, $\alpha_{i,s}$ 为基项,$i=1,\ldots,m$.

定义 f_i 如下:

对于 $\alpha_{i,s}\in Term(X_{i,1})$,有已知函数 $h_{\alpha_{i,s}}$,$s=1,\ldots,t_i$,$i=1,\ldots,m$,

(1) 当 $s\le k_i$,即 $\alpha_{i,s}$ 中不含非终极符时,

$$\begin{cases} f_i(\alpha_{i,s},y_2,\ldots,y_n)=_{df}\ h_{\alpha_{i,s}}(y_2,\ldots,y_n) \\ i=1,\ldots,m \end{cases}.$$

(2) 当 $s>k_i$,即 $\alpha_{i,s}$ 中含非终极符时,

$$\begin{cases} f_i(\alpha_{i,s},y_2,\ldots,y_n)=_{df}\ h_{\alpha_{i,s}}(Z_1,\ldots,Z_{r_{i,s}},y_2,\ldots,y_n,\ldots,f_k(Z_{j_{i,s}},y_2,\ldots,y_n),\ldots) \\ i,k=1,\ldots,m;1\le j_{i,s}\le r_{i,s} \end{cases},$$

其中 $Z_1,\ldots,Z_{r_{i,s}}$ 是 $\alpha_{i,s}$ 中出现的各个非终极符, $h_{\alpha_{i,s}}$ 式中的未知函数 $f_k(Z_{j_{i,s}}$ 对应的语言是 $L_{k,1}$),也可不出现.

以上定义的 f_i 称为是用联立递归式-II 定义的.

此联立递归式与文献[1,2]中定义的联立递归式的区别是,后者定义中各函数的用来进行结构归纳的语言(L_1)相同,值域(L)也相同,而前者可以不同.

定理 1. 如果已知函数都是 CFPRF,那么用联立递归式-II 定义的函数是 CFPRF.

B.2 多重归纳式

设有函数 $f: L_1\times\ldots\times L_n\to L$,其中 $L_i=L(G_i)$, $G_i=(V_N,V_T,X_i,P_i)$,$i=1,\ldots,n$.$L=L(G)$, $G=(V_N,V_T,X,P)$ .G_i,G 无二义. $\alpha_j^{i_j}\in Term(X_j)$,$i_j=1,\ldots,m_j$,$m_j$ 为 $Term(X_j)$的元素数 $i_j=1,\ldots,k_j$ 时 $\alpha_j^{i_j}$ 不含非终极符,$i=1,\ldots,n$.

定义 f 如下:

(1) 当 $i_j\le k_j$,$j=1,\ldots,n$ 时,有已知函数 $h_{l(i_1,\ldots,i_n)}$,

$$f(\alpha_1^{i_1},\ldots,\alpha_n^{i_n})=_{df}\ h_{l(i_1,\ldots,i_n)}().$$

(2) 当 i_j 满足下式

$$\begin{cases} i_j>k_j, & 若j=t_v,1<t_1<\ldots<t_u<n,v=1,\ldots,u,1\le u\le n \\ i_j\le k_j, & 否则 \end{cases}$$

时,有已知函数 $h_{l(i_1,\ldots,i_n)}$,

$$f(\alpha_1^{i_1},\ldots,\alpha_n^{i_n})=_{df}\ h_{l(i_1,\ldots,i_n)}(v(\alpha_{t_1}^{i_{t_1}}),v(\alpha_{t_2}^{i_{t_2}}),\ldots,v(\alpha_{t_u}^{i_{t_u}}),\ldots,f(y_1,\ldots,y_n),\ldots)$$

其中, $h_{l(i_1,\ldots,i_n)}$ 中的未知函数 f($j=t_v$ 时,$y_j\in\{v(\alpha_j^{i_j})\}$,否则 $y_j=\alpha_j^{i_j}$,且 y_j 对应的语言是 L_j,$j=1,\ldots,n$),也可不出现.

定义中引用了已知函数 $h_{l(i_1,\ldots,i_n)}$,$l(i_1,\ldots,i_n)$用下式计算:

$$l(i_1,\ldots,i_n)=\sum_{j=1}^{n}i_j-1\prod_{t=0}^{j-1}m_t,其中令m_0=1,$$

© 中国科学院软件研究所 http://www.jos.org.cn

$l(i_1,\ldots,i_n)$满足 $0\le l(i_1,\ldots,i_n)\le m_1\ldots m_n-1$.

以上形式的定义等式组称为多重归纳式,因为其中对定义域的多个语言进行了归纳.

定理 2. 如果已知函数都是 CFPRF,那么用多重归纳式定义的函数也是 CFPRF.

B.3 部分构造式

设有函数 $f:L_1\times\ldots\times L_n\to L$,其中 $L_i=L(G_i)$, $G_i=(V_N,V_T,X_i,P_i)$,$i=1,\ldots,n$.$L=L(G)$, $G=(V_N,V_T,X,P)$ .G_i,G 无二义.$Term(X_j)$的元素数为 $m_j,i=1,\ldots,n$.

定义 f 如下,设有已知函数 $h_i,i=1,\ldots,m$.

$$\begin{cases} f(A_1^1,\ldots,A_n^1)=h_1(B_1^1,\ldots,B_n^1,\ldots,f(y_1,\ldots,y_n),\ldots) \\ \vdots \\ f(A_1^m,\ldots,A_n^m)=h_m(B_1^m,\ldots,B_n^m,\ldots,f(y_1,\ldots,y_n),\ldots) \end{cases},$$

其中:

(1) A_i^j 是类型为 L_i 的变元,或 X_i 的项.当 A_i^j 是变元或 X_i 的基项时, B_i^j 为空;当 A_i^j 是 X_i 的复合项时, $B_i^j=v(A_i^j)$,$i=1,\ldots,n,j=1,\ldots,m$.

(2) 当 A_i^j ,$i=1,\ldots,n$ 是基项时,h_j 中不含 f.否则 h_j 中可以含 f,且当 A_i^j 是变元或是 X_i 的基项时,$y_i=A_i^j$;当 A_i^j 是 X_i 的复合项时,$y_i\in\{v(A_i^j)\}$,且 $y_i\in L_i,i=1,\ldots,n$.

以上定义等式组须满足如下条件:

(1) (完备性)对于任何 $Q_i\in L_i,i=1,\ldots,n$,等式组中必存在一个等式,设为第 u 个等式,使得 A_i^u 或者是类型为 L_i 的变元,或者是 Q_i 的推导序列中第 1 个产生式的右部,$i=1,\ldots,n$.称此等式为左端与 $f(Q_1,\ldots,Q_n)$匹配的等式.

(2) (一致性)等式组中任两等式的左端不相同.

对于 $Q_j\in L_j,j=1,\ldots,n$,$f(Q_1,\ldots,Q_n)$求值规则如下:

从等式组的第 1 个等式开始,按顺序找到第 1 个左端与 $f(Q_1,\ldots,Q_n)$匹配的等式,计算其右端表达式.

由于在定义等式组中允许用变元替代若干结构归纳情形,以上的等式组称为部分构造式.

定理 3. 如果已知函数都是 CFPRF,那么用部分构造式定义的函数是 CFPRF.

© 中国科学院软件研究所 http://www.jos.org.cn