第5卷第1期
1994年1月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 5, No. 1
Jan. 1994

FFP语言及其环境的实现*

王 鑫

(北京计算机学院,北京 100044)

摘要 本文介绍了以DC—LISP语言支持的FFP系统,它包括连接在一起的解释系统和编辑系统.文章主要讲述了实现的特点和方法.

关键词 函数程序设计,LISP语言,程序设计环境.

FFP语言(formal functional programming)是J. BACKUS于1978年提出的一种泛函程序设计语言.[1]为了提供一个供教学与科研使用的FFP语言及其相应的环境,本文介绍了以DC—LISP语言开发的FFP语言及其环境,包括FFP的解释系统和编辑系统.

本文利用DC—LISP语言及其编辑系统的一些底层功能,建立了一个完整的FFP语言运行和编辑环境,这个FFP系统建立在DC—LISP系统之上,并把DC—LISP系统全部藏起来.对于一般用户只需了解FFP语言,而可以完全不懂LISP语言.而对一些高级的用户,则FFP系统也具备向他们开放LISP语言的功能.FFP系统的解释系统与编辑系统是连结在一起的.因此,对于调试、编写、运行程序都十分方便.

本文对FFP语言的语义函数μ做了修改,对语法形式给予补充,在后面的叙述中,把本文的FFP语言称之为W—FFP语言,把本文的FFP系统称之为W—FFP系统.

1 W—FFP系统

FFP语言和LISP语言相比较,不同的地方在于FFP的函数不是通过描述函数施用前后数据之间的关系来定义,而是用泛函来定义.相同的地方在于都是将函数作用在数据上,得到新的数据.特别是,FFP语言的程序与数据的形式和LISP语言很接近,这使本文的工作显得十分自然和方便.

W—FFP系统包括解释和编辑二个部分.当系统运行时,首先进入编辑系统,在这个系统中,除了一般的通用编辑命令之外,还有专门为FFP语言设计的一些编辑命令,特别是它还有二个运行命令,一个可解释运行W—FFP语言,另一个是当用户需要时用来运行LISP语言.不论哪个语言运行之后,系统都自动地返回到编辑状态等待下一步的命令.

下面将分别介绍W—FFP系统的解释系统和编辑系统.

* 本文1991年8月20日收到

作者**王鑫**,44岁,副教授,主要研究领域为人工智能.

本文通讯联系人:王鑫,北京 100044,北京计算机学院

1.1 解释系统

关于FFP语言的语法和语义,[1]中有详细的说明.在本文中,除了本系统使用的语义规则外,不对W—FFP语言再做说明.在下文中用到的有关概念,例如:表达式、泛函、函数等,均是和FFP语言中的意义相同.

W—FFP语言的语义函数μ定义如下:

$\mu X=$

$X\in A\rightarrow X$

$X=\langle X_1,\cdots,X_n\rangle\rightarrow\langle\mu X_1,\cdots,\mu X_n\rangle$

$X=(Y;Z)\rightarrow\alpha(\mu Y,\mu Z)$

$\alpha(Y,Z)=$

$Y\in A\rightarrow\alpha(\rho Y,Z)$;Y是定义函数名或定义泛函名

$\beta(Y,Z)$;Y是系统函数名或系统泛函名

$Y=\langle Y_1,Y_2,\cdots,Y_n\rangle\rightarrow\alpha(Y_1,\langle Z,Y_2,\cdots,Y_n\rangle)$

其中:X表示FFP语言中的表达式,A表示原子的集合,ρ对定义函数或泛函来说是取其定义.β是描述系统泛函和函数的语义,它是一个二元的函数,一元是函数或泛函,一元是数据.β的作用就是将函数或泛函作用在数据上,提出新的结果.关于β的详细定义在此就不叙述了.关于函数和泛函的定义,采用如下形式:

def 函数名(或泛函名)=(表达式)

对这个定义式的运行,就是使函数或泛函有了由定义式中的表达式表示的定义.

在上述语义中,凡无规则可用之处,均视为无定义.

在W—FFP系统中,我们对FFP语义函数μ做了一些修改,例如,将泛函中的函数与数据合并时,去掉泛函名部分,并且在安排的位置上与原FFP语言的语义也不相同.做这些调整都是从实用的效率角度考虑.修改之后,在用户定义泛函时也应有相应的调整.

解释系统的实现由三部分组成:读入,解释,打印.

读入程序是用DC—LISP语言编写的.在本系统中,W—FFP语言的书写形式,除可采用FFP语言的尖括号表达式之外,也可同时采用FP语言的那种简化形式.读入程序把上述的两种并存的外部形式,均转化成内部的LISP表达形式,交给解释程序,同时进行语法检查.

关于解释程序,除了把语义函数写成LISP的函数之外,还有关于定义式的处理.由于本系统是在LISP系统支持之下的,所以可以把在W—FFP程序中的函数和泛函的定义用LISP的环境保存起来.因此,查找定义的效率很高.正因为如此,在描述语义时,在泛函的处理上可做相应的修改.

对于打印程序,本文通过DC—LISP的字符转换函数,可直接调用LISP的打印函数,输出FFP语言格式的数据.

1.2 编辑系统

LISP语言主要是处理S表达式,做非数值计算,因此在表结构上做处理是得心应手.但是对于处理编辑工作中所遇到的问题,例如处理字符串的插入、删除、修改等等,则是非常困难的.如果我们把所编辑的程序都转换成LISP的表结构,虽然LISP可以处理了,但所占的空间及其效率则是很不尽人意的.

在 DC—LISP 系统中,有功能齐全的效率很好的 DCIE 编辑系统.[2]而在 LISP 中,又有模拟编辑过程中键入字符的函数,还有直接与 BIOS 打交道的 BIOS 函数,通过它可以接收字符,显示字符,控制整个屏幕. 由于 DC—LISP 系统有了上述的基本功能,所以,可以使用 LISP 语言在编辑系统上面建立一个新的完整的编辑工作层面.

本文所述的编辑系统不是直接控制所编辑的字符串,只是把接收到的键入字符或编辑命令,通过 LISP 函数转化成 DCLE 编辑系统接收的键入字符和编辑命令,而实在的工作则交给 DCLE 编辑系统去完成. 这样,我们在实现 FFP 编辑系统时,省去了大量的有关贮存安排,细节上的连接等等工作,缩短了开发工作的时间. 同时,因为没有用 LISP 的表结构去模拟编辑的字符序列,而是间接地使用了由汇编语言编写的 DCLE 编辑系统,因而大大地提高了效率. 使得 W—FFP 系统的编辑运行速度和用汇编完成的 DCLE 编辑系统在使用时几乎感觉不出速度上的差异. 不仅如此,重要的是,在 DCLE 编辑命令的基础上,W—FFP 编辑系统开发了适用于 W—FFP 语言的新的编辑命令. 例如,寻找〈、〉、:、[和]等等,另外,本文的编辑系统还有运行 W—FFP 语言和 LISP 语言的命令,成为一个既可服务于 W—FFP 语言,又可服务于 LISP 语言,并能编辑一般文章的编辑系统.

结束语:本文实现的 W—FFP 系统,充分利用原有的 DC—LISP 的编辑系统功能,在 LISP 语言之上建立了一个新的完整的编辑运行工作层面,并且具有良好的编辑运行速度. 同时,开发周期短,效率高. 对于本文所实现的用于教学和科研的对运行速度要求并不很高,而需较好的使用环境的系统来说,本文的实现方法无疑是一个很好的选择. 同时,这个系统为今后实现一个带有良好环境的高效率的 FFP 系统提供了良好的基础. 本文所介绍的 W—FFP 系统已在 PC 机上实现.

致谢　本文的工作得到了马希文教授和宋柔副教授的指导和帮助,在此致以衷心地感谢!

参考文献

1　Backus J W. Can programming be liberated from the von neumann style? a functional style and its algebra of programs. CACM, August 1978.

2　朱连山. DC—LISP 程序开发环境. 中国计算机学会人工智能与模式识别专业委员会第二届学术讨论会论文,武汉,1988.

IMPLEMENTATION OF FFP LANGUAGE AND ENVIRONMENT

Wang Xin

(*Beijing Computer Institute, Beijing* 100044)

Abstract　This paper describes FFP system supported by DC—LISP language, including interpreter and editorial system which is connected. The paper is mainly explained to characteristic and method of implementation.

Key words　Function programming, LISP language, programming environment.