维普资讯 http://www.cqvip.com

# 多范例语言概观

张运桢　刘　震　贺武汉

（华中工学院计算机系）

摘　要

本文简要地介绍目前智能语言研究领域中的一种趋向——多程序设计范例的语言研究，给出了多范例语言结合的一般方法，并评述了几个典型的多范例语言。

## 一、引　言

目前，多种程序设计语言范例相互结合在一起（亦即所谓多范例语言）的研究正受到人们的极大关注。在这方面已开展了许多的研究与开发工作，并且取得了不少成果。这主要有两方面的原因或动机。一方面，第五代机的研究促进了这种多范例语言的研究工作。第五代机的关键是知识处理或进行知识程序设计，而知识处理和知识程序设计则要求有一些理想的语言工具或良好的程序设计环境，以便于人们用计算机表示和处理知识，但是现有的语言（或环境）没有一个能方便地进行知识处理或知识程序设计。另一方面，人工智能研究要求具有适合于解决人工智能问题的程序设计语言。人工智能问题种类繁多，领域广泛，各有其特点，一种范例（或风格）的语言不可能适于解决所有的人工智能问题。现有的人工智能语言，如Lisp，Prolog等无论哪一个都不能完全满足人工智能程序设计的需求。而多范例语言如同木工的工具箱，具备多种适于解决各种问题的工具。使用多范例语言，人们就可以更快地建立一个容易理解且便于求解问题的程序，从而也提高了软件生产率并改进了软件质量。因此，促使计算机研究人员开始研制多种程序设计范例相结合的语言。今天，越来越多的人认识到了这种多范例结合的意义和价值，而且多规范语言也正从实验室走向应用领域。

## 二、结合方法

程序设计范例是在长期的程序设计实践中形成的。常见的范例有：面向存取，数据流，面向数据结构，函数式，过程式（强制式），面向对象，并行，实时，面向规则、逻辑，等等。在人工智能领域，经常采用的程序设计范例如下：

·函数程序设计：如具有递归函数的Lisp语言。

·过程程序设计：它的主要特征是顺序性和使用副作用（尽管有时可能在LISP中也不可避免，但一般认为，这种程序设计风格缺乏美感）。

·逻辑程序设计：实质上是PROLOG，或一阶逻辑的较大子集，这种风格代表程序设计技术的一个巨大进步：从函数到关系，从确定性到不确定性的转折；人们只需说明要求解的问题，而不必描述求解它的算法。

·面向对象的程序设计：将数据组织成对象或抽象数据类型，每一对象（或一类对象）有一组操作（方法）处理存贮于该对象的数据。

目前大多数多范例语言所结合的范例似乎没有超出上述四种范例，而且其中最经常被结合的是函数程序设计和逻辑程序设计。

研制与开发多范例语言一般有四种方

维普资讯 http://www.cqvip.com

法：

(1)将现有的程序设计语言结合起来，例如，结合PROLOG、LISP和C的句法(和语义)成为一个系统。这种方法的优点是，熟悉其中某种语言成份的用户可以很快地使用该系统，缺点是，成份语言结构间产生了不希望有的复杂语义。采取这种方法研制的多范例语言有LOGLISP，RF-Maple等。

(2)在一个现存的系统中加入新的语言结构，例如，把对象、方法加入到Pascal中，这样扩充后得到的系统仍可使用主语言中原有的软件，如C++语言。

(3)对一种现有语言在新的理论目标下重新定义语言的语义、语法等。这种方法允许新语言的设计者校正它原来语言中的不足，并在其中增加新的范例，为产生新语言提供基础，例如QUTE语言。

(4)从头开始，吸取各种语言的经验，开发一个一致的形式基础，建造一个新系统。这种方法较易保持新语言的范例及设计风格的一致性。例如UNIFORM语言。

除此之外，还有一些结合多种程序设计范例的方法，如Lisp/prolog系统，它不是一个新语言，而只是纯Lisp和Prolog加上一个语言接口，从而实现函数程序设计范例和逻辑程序设计范例的结合。这些方法不具有普遍性，因此本文未加讨论。形形色色的多范例语言孰优孰劣，莫衷一是，本文不打算一一评价。

## 三、多范例语言浏览

出于研究工作的需要，我们考察了现今已出现的一些多范例语言，下面将介绍和分析其中有代表性的几个。

出现较早的多范例语言是J.A. Robinson等人于1983年设计和实现的逻辑程序设计语言**LOGLISP**，它是LISP中逻辑程序设计的实现，本质上是LOGIC和LISP的混合物，它的机制是在类似LISP程序设计环境下提供了与PROLOG等价的作用。LOGLISP由LISP和嵌套在LISP中表示LOGIC的新函数的集合组成，其中LOGIC部分是按照Kowalski描述的LOGIC程序设计语言的"纯"版本实现的。LOGLISP可以看成是LOGLISP=LOGIC+LISP，它结合了LISP语言和PROLOG语言两者的优点，不仅具有极强的表处理能力，而且还具有极强的推理演绎能力。它能在LISP工作空间建立断言知识库，通过执行合适的函数调用来计算询问语句。这样LOGIC部分能够使用LISP丰富的程序设计环境。在处理方式上，用户既可以选择纯LISP，纯LOGIC，也可以选择包含两者的混合风格，而且两者之间可以相互自由嵌套。因而便于解决它们单独使用时不易解决的问题，为用户提供了极大的方便。

LOGLISP仍有一些不足，如语义问题，并行问题等，因而远不能满足"新一代"的要求。不过，它毕竟率先向研制新一代语言迈出了可喜的一步，也奠定了分析和研究的基础，促进了一大批多范例语言相继诞生。

日本在多范例语言研究方面最显著的成果是多范例语言**TAO**。TAO是日本公共电报电话公司Musashino电气通信实验室的Ikuo TAKEUCHI等人为研制智能程序设计系统NUE (New Unified Environment) 而设计的核心语言。TAO是一种表处理语言，它以LISP语言的S-表达式的形式，支持逻辑程序设计范例、面向对象程序设计范例和传统的过程程序设计范例。TAO允许用户混合使用这些程序设计范例来求解复杂且广泛的人工智能问题。这些程序设计范例的基础，即合一，消息传递和函数调用，能在表达式中互相嵌套，于是用户可以在合一中直接使用函数调用或消息传递的结果，反之亦然。TAO也能够支持并行程序设计。TAO是Lisp，Prolog和Smalltalk在下面意义上的"谐和"：

$$TAO = 3/\left(\frac{1}{Lisp}+\frac{1}{Prolog}+\frac{1}{Smalltalk}\right)$$

维普资讯 http://www.cqvip.com

而不是简单的算术意义：

$$TAO = \frac{Lisp + Prolog + Smalltalk}{3}$$

需要指出一点，TAO的解释程序是在Lisp机ELIS上实现的，而不是通用的商用机器。ELIS的最初设计先于TAO，但TAO与ELIS是相互影响，相互促进的，ELIS体系结构的特性对TAO的设计有很大的影响，这也是保证TAO的解释程序效率的根本所在。

TAO作为多种范例（三者以上）结合语言的尝试是很有意义的。它目前还很幼稚，存在一些问题值得研究，如合一调用方式，结合方式和回溯处理等。如何更合理更自然地结合更多范例仍是一个值得深究的问题，然而TAO至少证明了多种范例（三者以上）结合的可能性。

日本研制的另一个多范例语言是**QUTE**。它是日本东京大学理学院信息科学系Masahiko Sato等设计的。QUTE的早期版本结合了Prolog和Lisp的特点，提供了一个开发交互程序的良好环境，QUTE的用户不仅能够享用Prolog和Lisp风格的程序设计，而且能够以独特的方式结合它们。这里介绍的QUTE是原QUTE的改进，并继承了它的许多优点。以后论及的QUTE均指新QUTE。

QUTE是一种能够执行并行计算的函数程序设计语言。大多数函数语言使用模式匹配作为基本的变量-值联系机制，而QUTE使用合一作为联系机制。因为合一是双向的，而模式匹配是单向的，因此QUTE比现有大多数函数式语言的功能强。QUTE自然地统一了逻辑程序设计语言和函数程序设计语言，可以用QUTE写一个程序，很象用传统的逻辑程序设计语言（如Prolog）所写的那样，同时，也可以写一个QUTE程序，看起来就象一个ML（一种函数式语言）程序。QUTE程序能够并行计算（仅仅实现"与"-并行），并且不论以什么次序计算都能得到同样的结果。这一点由计算算法具有的Church-Rosser性质保证。QUTE计算一个表达式都得在某一环境下进行，计算的过程可以看作为对所给表达式的归约过程。当表达式归约到不可能再归约的正规表达式时计算停止，所给的环境也随着计算过程由合一改变成另一个环境，因而计算的最后结果可看作正规表达式和环境的序偶。

值得指出的是，QUTE具有完全形式化的语义，形式体系较完备，这在其他多范例语言中是不多见的。它向着探求多范例语言的理想数学模型前进了一步。

**Funlog**是把逻辑程序设计和函数程序设计范例结合起来的一种语言，它是美国犹他大学计算机系P.A.Subrahmanyam和J.-H.You两人研制的。Funlog程序典型地由一个函数定义的集合和一个子句集合构成。Prolog可认为是Funlog的一个子语言，Funlog中的函数用一组方程来定义，可执行的函数可以在Horn子句中使用。Funlog的函数模型具有以下三个基本特征：

- 模型是基于归约的，并且具有"惰性"(lazy)；
- 具有执行模式匹配的能力，这使得我们可以建立一个基于模式驱动归约的机制；
- 模型是不确定的，并且不涉及回溯的概念。

Funlog的实现机制采用了归结和合一算法，但合一算法是语义合一算法，归结也是基于语义合一的。正是这种语义合一使得可执行函数能够在Horn子句中使用，并且是"按需归约"的。传统的合一是语法结构上的，而语义合一则是语义上的，这样，语法上不能合一的项，经过归约后，可能是语义合一的。

Funlog支持无穷结构上的计算，并且提供了无回溯（函数型的）和非确定性（基于逻辑的）计算机制供用户选择，Funlog通过把程序写成函数方程消除或减轻了在Prolog中的一些控制性问题并且显式地在程序中使用"计算"的概念。

维普资讯 http://www.cqvip.com

Funlog的成功之处在于它采用语义合一机制，实现了逻辑和函数程序设计范例的结合，这是对传统的句法合一的丰富和发展。由于增加了适当终止对无穷数据结构计算的能力，增大了基于逻辑计算的环境；由于提供了基于知识的推理能力，扩大了函数计算环境。

RF-Maple语言也是试图把逻辑程序设计范例和函数程序设计范例结合起来的一种语言，但事实上，它结合的是关系型程序设计语言R-Maple和函数型程序设计语言F-Maple。RF-Maple是加拿大的British Columbia大学的P.J.Voda和Benjamin Yu研制的。R-Maple是一个关系型逻辑程序设计语言，它试图在控制与语义之间进行平衡。程序的顺序执行和并行执行都比Concurrent Prolog说明得更为详细。R-Maple使用了显式的量词，并具有否定，因此，说明性地理解R-Maple程序不会受Prolog和Concurrent Prolog中的cut和commit的影响。F-Maple则是一个很简单的类型化函数型程序设计语言，它只有四个语法单位，它同时也作为一个操作系统。它是语法上可扩充的语言，其数据类型和函数的语法完全受程序员控制。

在设计RF-Maple时，选择了在一个函数型环境中引入关系的方法，这样得到的逻辑仅具有项而不具有公式，关系仅仅是具有布尔值的函数，逻辑程序设计的功能来自于生成符（generator），它是具有输出参变量的布尔函数。

RF-Maple的计算由两个组成语言F-Maple和R-maple来承担，而不作任何改动。函数由F-Maple的惰性求值来计算，生成符则由R-Maple的重写规则来计算。后一个计算要慢些，因为它必须借助于回溯，而函数计算则无此开销。由于结合R-Maple和F-Maple，使程序的可读性和执行速度都得到了改进。

另一个很有特色的多范例语言是Uniform，它是由瑞典Uppsula大学的K.M.Kahn研制的，Uniform是在扩充合一基础上开发的一个人工智能语言，它试图以一种简单而一致的结构方式把Lisp、Prolog和Act1的重要特性都结合起来。这个简单而一致的结合方式就是合一。所有的程序都是合一过程的扩充，合一起着模式匹配、求值、信息传递、继承和符号求值等多种作用。由于Uniform使用的合一是经过扩充的，因此其他机制，如归结、自动回溯或求值等都不需要了。

Uniform是建立在maclisp之上的，一般地说它含有Lisp的所有功能。它有一个直接与Lisp接口的原语。Uniform写的程序看起来非常象Lisp程序，但其含义与对应的Lisp程序是不同的。在Uniform中含有描述并发计算的原语，它也含有Act1的许多设施，而Act1是一个面向信息传递的（即面向对象）语言。

开发Uniform的目标之一是利用和改进Prolog的各种特性。Uniform能象Prolog一样，以多种方式使用同一个程序；Prolog支持一个定义（即程序）的所有用途，Uniform也同样支持，而且还支持一些新的用途。

Uniform语言研究的惊人成果就是合一，即产生一组描述的最一般例示的过程，它能够作为一个程序设计语言强有力的基础。（扩充了的）合一把Lisp，prolog和Act1的本质（即函数型的，基于逻辑的和面向对象的）统一在一个简单而一致的框架之中。

## 四、结束语

多范例语言的研究方兴未艾，本文所介绍的只不过是一个剪影。限于篇幅，本文仅侧重介绍了函数加逻辑程序设计范例中有代表性的语言，其他范例结合的语言还有很多，如卡内基-梅隆大学的Arctic(函数+实时)，AT&T贝尔实验室的C++（强制+面向对象），Case Western Reserve大学的

维普资讯 http://www.cqvip.com

CaseDE（强制＋说明），法国CGE研究中心的Lore（面向对象＋基于集合），日本Keio大学的Orient84/K（面向对象＋基于规则＋面向存取＋并行），IBM研究所的Smallworld(强制＋面向对象)等等。国内也有一些单位开始从事多范例语言的研究，我们期待出现更多的令人满意的成果。

## 主要参考文献


〔1〕 B.Hailpern, Multiparadigm Languages, IEEE Software, Vol.3 , №1 , pp6-9, 1986.

〔2〕 Multiparadigm Research, A Servey of Nine Projects, IEEE Software, Vol.3, №1, pp.70-77, 1986.

〔3〕 J.A.Robinson and E.E.Sibert, LOGLISP: Motivation, Design and Implementation,Logic Programming, 1982, Academic Press, New York.

〔4〕 I.Takeuchi H.Okuno and N.Ohsato, TAO—A harmonic mean of Lisp, Prolog and Smalltalk, SIGPLAN Notices, Vol.18, №7, 1983.

〔5〕 M.Sato, and T.Sakurai, Qute: A Prolog/Lisp type language for logic programming. Proceedings of the Eight International Joint Conference on AI.507—513, 1983.

〔6〕 Paul J.Voda and Benjamin Yu, RF-Maple: A Logic Programming Language with Functions, Types and Concurrency, Proc. Int. Conf. on Fifth Generation Computer Systems, 1984.

〔7〕 K.Kahn, Uniform—A language based upon unification which unifies (much of) Lisp, Prolog and Act 1, IJCAI, 1981.


---

（上接72页）

P

对资源操作

V

而HⅠ、HⅡ中实现同步的一般形式是：

((HC 资源名 操作名))

新同步机构优于信号灯。因为用P、V操作编程时容易漏写某个P或V、或者它们的次序不对，很容易出现错误。而新同步机构中，只要在对资源请求的地方写上一条语句即可。在谓词HC中包括对资源的实际操作，如果请求成功，HC会自动执行名为操作名的谓词来达到对资源的实际操作目的。对各种资源进行操作的谓词可集中起来存放在知识库中，这类似于管程，很便于管理。唯一要做的就是建立好资源的知识库。

以上工作只是我们整个工作的一部分，我们现在正在构造其它适合模型HNM的同步机构，在将构造的同步机构中，将引进人工智能的某些成果，以便更好地对资源进行管理。

## 参考文献


1. Tohru Moto-oka等, The architecture in the fifth generation computer, IFIP, 1983
2. E.W.Dijkstra, Cooperating Sequential Processes, Programming Languages, 1968
3. E.W.Dijkstra, Hierarchical Ordering of Sequential Process, Operating System Techniques, 1972
4. C.A.R.Hoare, Monitor: An Operating System Structuring Concept,CACM,1974
5. P.Brinch Hansen, The Programming Language Concurrent PASCAL, IEEE Transactions on Software Engineering, 1975
6. Campoll. R. H, Hhe Specification of Process Synchronization by PathE xpression, Lecture Notes in Computer Science,1974
7. K.L.Clark等, micro-PROLOG Programming in logic, 1984
8. 周长林，操作系统讲义，吉林大学，1985
9. 代爱军，新一代计算机操作系统NGCOS，吉林大学硕士研究生论文，1987
10. 渊一博等，日本第五代计算机，1987
11. 苏学智等，日本第五代计算机系统工程，计算机科学，85.3
12. 汤子瀛等，计算机操作系统，1984
13. 谢 立等，智能操作系统初探