计算机科学 1996 Vol.23 №.3

关于计算机研究的理论框架*)

方滨兴 李晓明 王 群
(哈尔滨工业大学计算机系 哈尔滨 150006)

摘 要 Computational model, programming paradigm, and execution mechanism are often heard terminologies in different contexts among researchers of computing technology, especially in AI community. The meaning of them seem varying from context to context, sometimes cause confusion. In this work, we are trying to clarify these concepts and give a general framework to unify them. In particular, we are proposing a hierarchical structure with four layers: computational model, computing model (including calculus model and operational model), programming paradigm, and computer model (including abstract machine and execution mechanism), and discussing the relationship between them.

关键词 Computational model, Calculus model, Programming paradigm, Abstract machine, Execution mechanism.

一、引 言

计算机的研究始终离不开理论模型作为背景。计算问题、计算机语言问题、程序设计问题以及计算机体系结构问题，无不建立在相应的模型之上。

谈到计算模型，我们很容易联想到图灵机模型，这是每一个计算机工作者无所不知的，因为图灵机是构成现代计算机理论的基石。同时我们还很容易想到λ-演算模型，产生式模型是作为 LISP 语言、OPS5 语言的计算模型而被研究的。

在从事开发 Prolog 语言的并行性的研究中，研究者们讨论着诸如与并行模型、或并行模型、与/或并行模型等术语[1]。与此同时，研究产生式并行性的人们则在讨论着 RETE 网络、RETE+网络、TREAT 并行模型等[2]。

从事人工智能计算机的研究者往往更关心的是执行机制与抽象机，如数据驱动推理机、归约机制推理机、图归约机[3]等，而抽象机则首推 Warren 抽象机影响最大。

从程序语言角度来说，过程式语言、非过程式语言、函数式语言、面向对象式语言，如此种种，各种语言相去甚远，那么其理论基础的何种差异决定了其形式上的巨大差别呢？

以上种种可以看出，在计算机领域的各个层次的研究中，都有着相应的理论模型作为研究基础，问题的关键在于这么多的模型及这么多的术语并不是一个有机的整体。也就是说，还没有一个完整的理论框架，把这么多理论模型统一起来，使之各自有着自己的位置。

本文的研究旨在尝试这样一个工作，建立一个理论框架，综合各自独立提出的理论模型，使之在一个完整的理论框架中有着各自的位置，从宏观角度指导对这些理论问题的研究。

二、计算机研究的理论框架

在讨论整体框架之前，首先讨论一下计算机研究的层次问题。计算机的研究从抽象到具体，从理论到实现，从软件到硬件存在这样的研究层次。

类 1：关于计算的模型。这是计算的理论基础，决定了可计算的理论，应该说是整个计算机的理论基础。

类 2：关于计算描述的模型。这是程序语言的理论基础，决定了程序语句的作用形式。

类 3：关于计算方法的模型。这是程序执行算法的理论基础，决定了程序的组织形式与执行方式。

类 4：关于程序的模型。这是程序设计框架的理论基础，决定了程序设计的风格。

类 5：关于计算系统的模型。这是计算系统的理论基础，描述了计算系统的基本能力。

类 6：关于计算系统执行方式的模型。这是计算系统控制策略的理论基础，决定了一个计算系统当前的操作方式，即当前执行语句的选择方式。

*)本工作受八六三计划(306 主题)"智能机执行机制的研究"项目资助。

上述六个层次描述了计算机研究中几个基本问题。很显然，关于计算的模型就是图灵机这类模型，是计算机研究的基础。关于计算描述的模型、关于计算方法的模型及关于程序语言结构的模型所涉及的是程序设计语言，可以说是软件部分的研究领域。关于计算系统的模型及关于计算系统执行方式的模型则是涉及计算机体系结构的，可以说是硬件部份的研究领域。

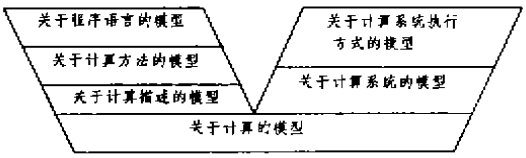

图 1

另外，有了计算语言的描述才有语言的集合，这种集合便涉及到计算语句的组织问题，在此称之为计算方法的问题，最终将涉及到程序框架问题。同样，有了计算系统的模型便涉及到了计算系统的执行方式问题。因此这六个层次有着图 1 所示的层次关系。

根据这一描述，我们构思一个计算机理论研究框架由如下的四元组构成：

$$F=(T,C_o,P,C_r)$$

T 是计算机理论模型集合，它的元素决定了可计算性的问题。它的构成有三个原则：①是对计算进行描述的；②是可以运行的且具有判断能力的；③所处理的集合中的对象足够简单。它的构成包含了四个要素：操作对象、操作规则、初始状态、终止状态。

C_o 是计算模型集合，由二元组构成：$C_o=(C_e,O)$。其中：C_e 是演算模型集合，它的元素决定了计算语句的描述与操作问题。它的构成有三个原则：①描述的是一个独立的计算语句；②所描述的语句包含有动作及动作的结束；③所描述的语句包含有信息的传递形式。它的构成包含三个要素：操作算子、传递算子、操作对象。

O 是操作模型集合，它的元素决定了计算语句集合运行过程中的算法问题，也可以说是决定了程序运行的执行机制问题。它的构成有两个原则：①所描述的对象是信息的流动方向和选择方式；②所描述的内容是程序的控制策略。

P 是程序设计风范集合，它的元素决定了程序设计语言的使用风格，是面向用户使用的一种描述形式。它的构成包含三要素：数据类型集合、程序控制语句集合、输入输出语句集合。

C_r 是计算机模型，由一个二元组所构成：$C_r=(A,E)$。其中：

A 是抽象机的集合，它的元素决定了一个计算机系统的基本功能。它的构成包含有三个要素：原始基本指令集合、基本功能部件、基本拓扑结构。

E 是计算机执行机制的集合，它的元素决定了一个计算机系统的控制方式，或者说是决定了计算机系统的调度方式。它的构成包含有二要素：作用的对象、对作用对象的操作方式的集合。

根据这一定义，任何一种关于程序语言或体系结构的理论模型的研究均应该属于这一理论框架的某一个集合。

上述定义所涉及的六个元素集合与前面六个层次类型的模型相对应，它们之间的关系为：理论模型是关于计算的模型，演算模型是关于计算描述的模型，操作模型是关于计算方法的模型，抽象机是关于计算系统的模型，执行机制是关于计算系统执行方式的模型，如图 2 所示。

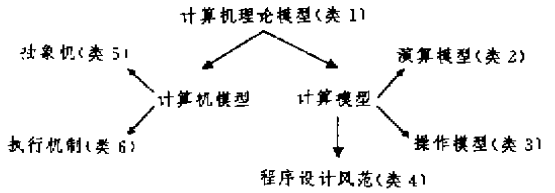

图 2

三、对一些常见模型的解释

对于理论模型来说，最典型的是图灵机模型、Post 机模型及递归函数[1]，可以证明从可计算性判断的角度来看，这三个模型之间是等价的。其中图灵机是站在机器模型的角度来描述问题，因而被视为计算机结构的理论模型；Post 机是站在符号处理的角度来描述问题，因而被视为程序设计语言的理论模型；而递归函数则是以哥德尔编码原理为依据而站在数字运算的角度上来讨论问题，因而被视为可计算函数类的理论模型。由此可见，理论模型从自动机符号的形式化及函数等各个角度描述了可计算性问题。

从计算模型的角度，理论模型可分为演算模型与操作模型两个部分。对于演算模型来说，已经有很多成熟的被人们所接受的模型。最具有普遍意义的演算模型是“赋值语句”，或者被称为“传送-加”模型。这是典型的过程式程序的语句模型。尽管很少有人这样称呼这个模型，但过程程序的计算型语句是赋值语句这一事实是被公认的。因此说 Pascal，C，Fortran，Basic 等语言尽管如此驰名，且并存了如此长久的时期，但由于其演算模型是相同的，所以属于

同一类程序语言，自然也属于数值计算型语言。

一阶谓词逻辑是 Prolog 语言的演算模型[5]。这种由头部来查找体部的模型决定了 Prolog 语言的递归特性，使其成为适用于反向推理的一种典型语言。

产生式是 OPS 语言的演算模型[6]。这种满足条件便激活动作的模型决定了产生式语言的前向链的特性，使其成为适用于正向推理的一种典型语言。

λ-演算是 LISP 语言的演算模型[7]。这种逐个进行项替换的模型决定了 LISP 语言的链表特性，使其成为适用于表处理的一种典型语言。

数据流模型是数据流语言的演算模型[8]。尽管这种语言并没有被公众接受，但数据流模型决定了这种语言有着极好的并行性，因为这种数据携带着操作符号来匹配以激活每一个具备了全部操作元素的操作符的模型决定了一切满足条件的操作均可以立即被执行。

由此可见演算模型决定了程序设计语言的基本计算能力，因而基于不同演算模型的程序设计语言在计算、推理、表处理等方面各有优势。

对于操作模型来说，所涉及的是程序系统的操作策略问题。如关于产生式系统中的匹配策略问题便属于操作模型的范畴，还有这一领域的研究者们所熟悉的 RETE[9]算法，Oflazer[10]算法、RETE+[2]算法等，人们在研究逻辑程序的并行性中所提出的种种策略，如 AND/OR[11]、TOKEN[12]、RAP[13]等自然也属于操作模型的范畴。可以看出，对操作模型的研究是后期出现的，这是因为程序机制随着并行程序的研究而复杂起来，因而需要有一定的模型作为理论基础来指导相应的研究。

程序设计风范的典型例子是面象对象语言 Smalltalk[14]。在这里我们所要特别强调的是面向对象语言的基本特性是因其程序设计风范而体现出来的，与演算模型无关。这一点从 C++语言中可以看出[15]。如某一个 C++程序中没有使用对象、类等概念，那么这个程序实际上就是 C 语言程序。这是因为所对应的演算模型是相同的，即"赋值语句"模型，而 C++的程序设计风范特性没有被应用到。过程性语言风范是历史最悠久的一种程序设计风范，以至于人们对此类程序的设计风范并未加以关注。这类程序的数据类型的演变并未跳出一些基本的数据结构的框框，如结构变量、指针、流等。程序控制语句更是限制在简单的循环、重复、EXIT 等范围内，因而说这类语言的相互变换是相对容易的。至于 Ada 语言、FORTRAN-90，由于引入了并行数据结构，而演变为并行过程式程序设计风范。

逻辑程序设计的程序风范表现在程序的控制机制隐式地体现在程序机制的内部，而不象 C 语言那样通过 Do，While 语句来显式描述。这种隐式控制的形式我们称之为内部搜索机制。逻辑程序的这种程序设计风格决定了该语言具有推理能力，而这种基于堆栈的归约式风格决定了其反向推理的能力。

与逻辑程序设计语言相同、产生式语言的程序设计风范也是表现在内部搜索机制上，差别是产生式的搜索机制是基于条件匹配与冲突归结。这一特点决定了其正向推理的能力。

函数程序设计语言的程序设计风范应该说是属于过程式程序设计风范，其数据类型与程序控制语句没有更为特殊的变化，仅是受 λ-演算模型或递归函数演算模型的限制，使得程序的表现形式是函数式的。如 LISP 语言、FP 语言。也就是说，从程序设计风范的角度来看并没有作出更多的文章。

从计算机模型的角度，理论模型可分为抽象机和执行机制两个部分。抽象机是指导一个计算机设计的必要手段，影响最大的抽象机模型当属 Prolog 机的 Warren 抽象机。它对从事 Prolog 语言及机器开发的研究者们来说几乎是无人不知的。该抽象机建立了一个高效的 Prolog 执行环境。同样的例子有清华大学的并行图归约模型抽象机[21]，国防科大的并行 Prolog 抽象机(RAP/COP-WAM)[16]。另一个比较知名的抽象机是 ACTOR 抽象机[17]，是为适于开发大规模并行系统而提出的。抽象机的一个显著特点是包含了对一个系统原始功能的一个完备的描述，使得该抽象机是可以运行的。基于层次结构的包控制抽象机也是一个例子[3]。

最后一个要讨论的是执行机制。关于执行机制应该说已经派生出了许多概念。最脍炙人口的是数据流驱动执行机制[18]、归约驱动机制[19]。一个执行机制是决定硬件系统操作原则的一种规范，也是决定了硬件结构的宏观控制策略的集合。更明确地说执行机制就是决定了硬件系统当前要执行什么样的操作。所谓数据流驱动的含义是依赖一个操作符所需的数据到达与否来决定是否应该执行该操作符。同样，归约机制则是依赖替代、一致化的结果来决定是否进行进一步归约。

与程序设计风范的提出相同，执行机制的提出也是近期的事。由于执行环境越来越复杂，尤其是多机并行执行，使得 CPU 如何确定当前应该执行何种操作变得复杂起来，从而提出了执行机制的概念。可以看出，各种执行机制的思想的提出往往是来自于对某种基于多机环境的人工智能计算机的研究的过程。回顾传统的冯·诺依曼结构，当然也有其自身的执行机制，这就是指令计数器，也就是说决定 CPU 当前应该做什么的根本因素在于对指令计数器的操作能力。至于条件转移、中断、微程序控制等等无非

是基于指令计数器的执行机制而对指令计数器进行某种操作罢了，这就是研制一个冯式结构机器（当然也包括阵列机）的控制系统的关键之一便是指令计数器逻辑的实现方式的原因所在。由于这个问题如此之简单，使得人们过去对执行机制这一术语不屑一提。

其它一些执行机制包括包驱动执行机制[20]、图归约机制[21]、需求驱动机制[22]、任务驱动机制[23]、中断驱动机制[24]、全拷贝机制[25]等。

四、关于冯·诺依曼结构的讨论

我们已经定义了计算机理论研究框架，并且将目前人们孤立地提出的一些研究术语放到了框架中相应的位置。但是单纯地这样统一并不全是我们的目的。我们的目的是建立了统一的理论框架后应更有助于对计算机中的一些本质问题的研究。本节及下一节便是试图进行这样的尝试。

首先，我们要探讨一下在这一理论框架下冯·诺依曼结构机器处于什么样的位置，其目的是为了探讨何为突破冯·诺依曼结构的束搏。

从理论模型角度来看，很显然图灵机是冯·诺依曼的理论模型。这种顺序存取的自动机描述了冯式结构的串行特性。当然，随后有很多图灵机的自动机被推出，试图通过多头来描述并行，但这种描述由于头数的有限而限制在伪并行之中。

从计算模型来看，普遍认为冯式结构所依据的演算模型是赋值语句模型。这种"传送-加"的语句方式决定了冯式结构的数值计算特性，它所依据的程序结构并不复杂，所以不需要建立操作模型。

从程序设计风范来看，传统的过程语言风范被理解为冯式语言。但从实际情况来看，程序设计风范与冯式体系结构没有直接的关系，因为程序设计风范是面向程序设计员的，而且程序设计风范的演变也不是与体系结构的演变而同步进行的。

从计算机模型的角度来看，冯·诺依曼结构所依据的抽象机是以存储器（或CPU）为核心，原始指令集合由传送、加、条件转移、移位等指令所构成，所依据的执行机制是基于冯式结构的指令计数器的控制方式（人们通常称之为控制流）。可以看出，这个计算机模型浓厚地表现出冯·诺依曼结构的色彩来。

从以上分析可见，冯式结构的特性是从两个方面表现出来的。一是计算语言方面。也就是由计算模型及程序设计风范所决定的特色，这一点早已由于众多的演算模型，程序设计风范的出现而不具有统治地位。二是体系结构方面，即由计算机模型所决定的特色。在这方面尽管现存的一些基于非冯抽象机及非冯执行机制的机器并不十分成功，但从这两个角度来突破冯·诺依曼结构的束搏是正确之路。

既然从计算机模型的角度已经存在了非冯的模型，为什么所研制的计算机却无法站住脚呢？这一方面是工业界更崇尚传统的问题，另一方面则是应用面向问题。我们尝试建立一个概念模型来说明这个问题。首先需要说明的是计算机研究理论框架中不同层次间的模型可以组合应用，因此非冯的计算模型、程序设计风范可以运行在基于冯式计算机模型的环境之上。仅从计算机模型的角度来看，冯式结构可以说是硬化程度最高的。也就是说，冯式结构所描述的功能是最基本的。在此基础上可以组合成各种复杂的功能。这就是其它一些计算模型及程序设计风范可以运行在冯式结构下的原因。下面用图3的概念模型来进一步说明。

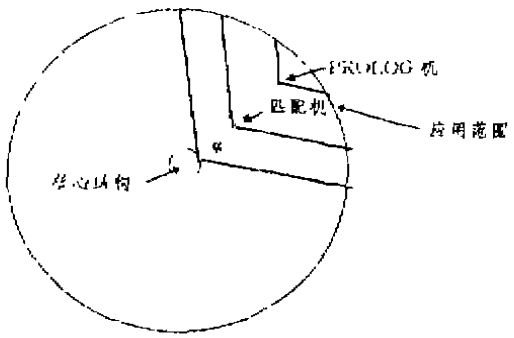

图3

图中圆心表示最基本的功能描述，称为核心结构，圆周表示复杂的应用系统的范围，圆心至圆环的半径虚线表示基本功能与复杂的应用之间的距离。当然，早期的计算机由于应用领域很窄，应用简单，在图中表现的就是半径很短，圆周很小。随着科学的进步，应用领域的拓宽，半径也就相应地增长，圆周也相应地增大，这里我们将计算机模型定义为软硬件界面，即虚线环所在的位置。其中虚线环以内我们称之为硬化部分，虚线环以外则称之为软件应用部分。也就是说，该界面（虚线环）越靠近圆心，意味着硬化部分越少，而越靠近圆周，则意味着硬化部分越多。两个极限是界面处于圆心及圆周上，在圆心上则可看作是纯粹而简洁的冯式结构，在圆周上则可以看作是一种面向应用的专用机，如LISP机。由界面所在处为顶点向圆周发出的的夹角为α的两条射线切割了一段圆弧，我们称这个圆弧为"应用弧"，表示采用了该界面所支持的计算机系统的应用范围。在此我们认为α角是个常数，不随界面位置的变化而变化。由此可以看出，随着界面位置（夹角顶点）向圆心的移动，系统的应用范围（应用弧）将增大，而随着界面位置向圆周的移动，系统的应用范围（应用弧）将减小。这就是专用机很难有市场的缘故，如Prolog

机，LISP 机等。同理，人们研究人工智能计算机的过程，实际上就是研究在这个界面中如何寻找一个最佳位置，使得虚线内硬化的部分是通用的瓶颈口而值得硬化的，同时所选择的"应用孤"又是所期望能够支持的。例如研究一个以硬件匹配器为核心的通用人工智能计算机，姑且称之为匹配机，因为人工智能语言均是以匹配为主要操作，因而其通用性显然比 Prolog 机好。

五、关于各层次间的模型配合问题——兼论多风范程序设计语言

从前面的讨论可以看出，一个程序语言的形式、效率及特色均受计算模型及程序设计风范的约束，但这两个层次是相互独立又相互影响的。考虑到操作模型是指导程序运行系统提高效率的，因而对程序的外部特色并无太多的影响。因此现在仅讨论计算模型中的演算模型与程序设计风范之间的关系。

从传统的程序设计语言来看，所涉及的演算模型与程序设计风范如表 1 所示。

表 1

程序语言	演算模型	程序设计风范	一般的理解
C	赋值语句	过程式	过程式
FP	递归函数	过程式	函数式
LISP	λ-演算	过程式	表处理式
Smalltalk	赋值语句	面向对象	面向对象
OPS	产生式	内部搜索	产生式
Prolog	一阶谓词逻辑	内部搜索	逻辑式

可以看出，传统的观念对 C 与 Smalltalk 的理解是基于它们的程序设计风范所进行的，而对 FP，LISP，OPS，Prolog 的理解则是基于它们的演算模型来看的。这是因为前者的程序设计风范所带来的色彩远比演算模型为重，而后者的演算模型所带来的色彩则远比程序设计风范要重。

人们常说 OPS 与 Prolog 是典型的逻辑与控制相分离的语言，尽管 LISP 与 OPS 及 Prolog 一样也常被称为人工智能程序设计语言。但只有 OPS 与 Prolog 被称为推理型语言。其根本原因在于它们的程序设计风范表现在搜索机制被蕴含在程序执行机制内部，而无需用户显式指出。事实上我们完全可以设计这样的语言，其演算模型是产生式的 IF…THEN 模式，而其程序设计风范采用过程式风范，这仍然是一个可进行正向推理的语言，但冲突归结问题，匹配问题则需要由用户来干预了，因而失去了产生式语言的长处，并不能再称之为推理语言。因此说，演算模型与程序设计风范有一定的依存关系，但在不丧失优越性的前提下，进行多种组合是有意义的，可能会有意外的收获。

研究多风范程序设计语言的人们发现，将某一种语言与面向对象语言结合并不是难事，如 C++[35]（赋值语句＋面向对象），Flavors[25]（λ-演算＋面向对象），Spool[27]（一阶谓词逻辑＋具有内部搜索机制的面向对象）。这一点由本理论框架很容易解释，因为这仅是不同层次的模型的不同组合罢了。只要在同一层次中的不同模型可以结合，其语言的综合便是可能的。如面向对象与内部搜索机制，而逻辑语言与函数语言的结合却不是容易的事，其根本原因在于是否能构造一个新的演算模型，其包含有 λ-演算的特性或递归函数的特性及一阶谓词逻辑的特性。尽管也有这方面的尝试，如 LPGLISP[30]（λ-演算与一阶谓词逻辑混合），Tablog[26]（递归函数模型与一阶谓词逻辑混合），TAO[29]（λ-演算与一阶谓词混合＋面向对象）等。但也公认函数与逻辑结合的问题效率很低，没有达到实用水平。这是因为目前所采用的方法无外是这样四种：1）以函数语言的语义为主，结合逻辑语言的本质特征。2）以逻辑语言的语义为主，结合函数语言的特点。3）以等式语义为基础来合成语言。4）约束逻辑程序设计。

用本文提出的计算机理论框架来看，解决问题的根本方法是建立一个新的可满足具有 λ-演算或递归函数与一阶谓词逻辑特性的演算模型。如果这一点是无法做到的，那么这种研究则会导致重复人们走过的路，却得不到有良好效果的结果。

从上面的讨论可以看出，人们常提到的将函数语言风范，逻辑语言风范与面象对象语言风范结合的提法是不严谨的，因为前两者是演算模型所表现出的特色，而后一种才真正是程序设计风范所表现出的特色。事实上，研究面向对象语言的人已经发现，"函数程序风范与逻辑程序风范基于数学或理论模型，而面向对象程序语言通常是由具体的语言结构来定义的，面向对象程序语言缺乏坚实的理论基础"[30]，这一论点从一个侧面说明了由于演算模型与程序设计风范左右一个语言所表现出的特色而带来的差别。

六、小结

本文提出了一个计算机理论研究框架，试图将当前的程序设计语言问题的研究与体系结构的研究中的一些理论模型问题归纳到这一框架中来。由于有了一个统一的研究框架，一个具体问题的研究在开始之前便可知道这一研究处于那一层次，起着什么样作用，应该有什么样的突破等。由此，可以指导涉及这方面问题的研究，这也是提出这一理论框架的目的。（参考文献共 30 篇略）