新一代计算的若干问题

鲁汉榕（武汉空军雷达学院）

摘 要

新一代计算与传统计算迥然不同，前者以AI问题求解(搜索、启发式方法、符号计算、非确定性算法、基于知识的问题求解等)为核心，以并行处理体系结构为基础，以知识信息处理（自然语言理解、图象识别、语音识别、专家系统等）为应用；而后者以确定性过程算法为核心，以顺序（von Neuman）体系结构为基础，以数值数据处理为应用。传统计算强调的是严格的"计算"，而新一代计算强调的则是"搜索与推理"。

本文概略讨论新一代计算中的若干重要问题，包括新一代计算语言与程序设计，新一代计算模型与体系结构，新一代计算知识表示，新一代计算与软件工程，新一代计算与非标准逻辑，以及新一代计算与超并行结构等。

新一代（或称第五代）计算与传统的计算（第一代至第四代）是相对而言的，并无明显的界限。特别是，早期计算的分代一直是基于器件技术的且分代是在事后，而这一次分代则是基于处理对象（知识与数据）的且分代是在事先，没有分代基础（标准）的连续性。总之，新一代计算除了包容了传统计算之外，还能处理传统计算无力胜任的一些问题（应用）。

从计算的角度看，求解问题有两种类型，一种是信息完全的问题求解，另一种是信息不完全的问题求解。传统计算是第一类问题求解，主要涉及数据的变换或运算（通过确定性算法来实现）。新一代计算是第二类问题求解，主要涉及在不完全信息（缺少知识，知识不确定等）情况下的计算，包括推理，选择，结论，决策等。由于信息不完全，导致有必要在计算中枚举（所有）可能性。由于朴素枚举所有可能性的计算代价一般高得不可接受，所以新一代问题求解致力于减少所需枚举（试探）的可能性数量，因而其核心是基于知识的启发式搜索或智能选择，即在有限的（不完全）知识下求解问题。不完全知识导致新一代问题求解的非确定性，非确定性是并行性的来源。传统计算的解的质量和效率主要取决于算法，而新一代计算的解的质量主要取决于领域的知识量，解的效率则主要取决于（控制）算法。

一、 新一代计算语言与程序设计

新一代语言是新一代计算的描述工具；任何一种编程语言都应首先是面向用户的，其次才是面向系统（实现）的。

新一代语言的许多特点都是由新一代应用所决定的，这些特点包括：

·符号计算。新一代计算的基本操作主要是符号处理，包括分类，比较，匹配等。

·非确定性。新一代计算往往要处理不完全信息或不足够／不确定知识，因而计算本质上是非确定性的。

·并行性。并行性主要导源于非确定性。

·链表（List）和递归。链表作为符号计算的最基本的最重要的数据结构，递归技术经常与链表一道使用，它实际上是一种分治技术。

·晚期确定类型／无类型。由于新一代计算涉及的客体的复杂和灵活多变，语言中的变量类型不是事先确定，而是在执行中确定的。

·动态数据结构。它与晚期确定类型都是由非确定性引起的。

·强表达能力。

·高抽象性。

·低过程性。与非确定性有关，对并行处理有利。

·单赋值性·与副作用有关，对并行处理和良性质程序有利。

新一代语言编程范例主要有三大类，即面向逻辑的，面向函数的，以及面向对象（object，目标，客体）的。

逻辑程序设计一般以一阶谓词逻辑（的子集）作为编程语言，以PROLOG为其主要代表。逻辑程序设计（语言）基本上满足上述一切特征。它具有双重语义，即说明性语义和过程性语义。前者面向用户，后者面向系统实现。说明性编程允许用户采用关于某一领域问题的有关知识（即事实/断言，规则/关系等）作为公理，以及要解决的问题作为待证的定理，而不关心如何用这些知识来逐步解决给定问题。求解的实现是通过过程性语义自动完成的。过程性语义可以用任何一种机械定理证明方法来实现，例如SLD方法等。

广泛使用的产生式系统（语言）与逻辑程序设计没有很大的差别，其典型代表是OPS系列。

函数式程序设计将程序与数据统一表示成表达式和函数，程序的执行是一系列（数学）函数的计算或调用。函数式程序设计语言具体还包括几类，即纯函数式程序设计语言如FFP，表处理语言如LISP，以及数据流语言或作用式（Applicative）语言如VAL，ID等。函数式程序设计的优点是表述简洁清晰，数学性质好，隐含大量的并行性，并且比逻辑程序设计的效率要高（因为它在一定程度上规定了执行的细节，即过程性）。

面向对象的程序设计一反将数据操作（命令，过程，算法等）作为核心的传统做法，而是以数据为核心。数据加上有关的操作即构成对象。计算是通过对象之间传送消息（操作要求），要求数据对象对自身完成某种操作来实现的。这种编程范例的特点是信息隐藏，数据抽象，性质继承，以及动态类型。其典型代表是Smalltalk-80。

二、新一代计算模型与体系结构

新一代计算模型应便于描述并行处理和推理计算。计算模型是语言（程序设计）与机器（体系结构）之间的一种桥梁。一方面，它是编程语言的理论基础，另一方面，它作为并行体系结构设计的指南。

新一代的基础计算模型主要有两大类，即数据流模型和归约计算模型。数据流模型适宜于实现函数式程序设计，归约模型适宜于实现逻辑程序设计和函数式程序设计。

新一代的体系结构正处于研究早期阶段。体系结构可以面向语言（如LP机，FP机，OOP机等），也可以面向应用（如专家系统知识库机），还可以面向计算模型（如数据流机，归约机，逻辑推理机等）。一个原则是，要求新一代计算问题、描述（编程语言）、计算模型、以及体系结构等之间具有良好的匹配和合适的间隙。

新一代体系结构应对新一代计算提供各级硬件支持。新一代的体系结构试图克服传统的von Neumann顺序体系结构对于新一代应用的不适应。其特点由新一代应用的特点所确定，主要有：

·各种符号处理专用部件，例如模式匹配处理器等。

·动态可重构结构，处理动态数据结构以及适应新一代计算问题的多变结构。

·并行处理。新一代体系结构必须是并行处理结构，用来（部分）实现非确定性，另一方面，新一代计算的计算复杂度极高，传统单机的性能最终无法满足其计算资源的需要。

·堆栈结构，用来有效地处理链表、递归以及函数调用等。

·上下文切换器，用硬件来实现计算环境的变化。

·带类型标志存储，支持无类型的新一代计算语言，并有利于动态类型检查和符号串合一操作。

·无用单元回收器，新一代计算语言对内存的要求苛刻，经常需要作单元回收操作。

·相联存贮器，用来快速搜索知识库。

·通迅处理器，用来支持并行处理，控制通讯时间。

·对于知识表示及推理的专门支持。

·对各种编程范例的专门支持（例如固件PROLOG解释器）。

三、 新一代计算的知识表示

从某种意义上讲，新一代计算也可称做基于知识的计算，因此知识在新一代计算中占有重要的位置，而知识表示又是其中最为紧要的问题。

人工智能系统特别是专家系统成功的必要条件是尽可能狭小的问题域和尽可能多的知识。人工智能系统的智能行为水平取决于其具有的知识的数量和质量，而在计算机中如何表示用于问题求解的知识则是人工智能中的一大难题。知识表示方式研究的是如何用符号结构来表示各种一般的和专门的知识，这种表示还必须满足一定的条件，例如存在有相应的操纵知识的自动（机械）过程等。

知识表示方式的典型是逻辑，产生式系统，语义网，框架，过程，（关系）数据库等，它们各有长短，适用于不同的应用。关于知识表示，已经发表了大量的研究结果。知识表示的若干重要方面是：

·直观性。知识表示应在一定程度上反映人类问题求解中使用知识的方式，具有一定的直观性，便于理解和交流。

·简洁性。知识表示的简洁性意味着时空效率。

·抽象性。抽象知识表示更接近于问题域。

·易结合性。任何一种表示方式都有一定的缺陷，通过接合其他方式，往往可以取长补短。

·可分划性。知识项之间的联系应不是微妙的或复杂得不可控制，知识库的分划有利于并行处理。

·层次体系。知识组织应具有一定的结构性和访问局部性，以便提高处理效率。

·同态性。知识表示结构应真实直接地反映问题域结构。

·合适的知识粒度。知识粒度影响推理步长和平均进程生命期等，对于并行多处理系统性能影响很大。

·推理（释）效率。

·模块性。

·易扩充／修改。

·可维护性。

·并行性。

·知识深度。

·易实现性。

·表达能力。

四、 新一代计算与软件工程

新一代计算的软件工程具有极为重要的意义，它包括智能语言的设计与实现，新一代智能（系统与应用）软件的开发与维护等。由于新一代计算较传统计算要更为复杂，所以新一代软件工程较之传统软件工程具有很多不同的地方。逻辑程序设计代表了新一代计算程序设计的主流。本节以逻辑程序设计为例来讨论新一代计算软件工程的若干问题。

软件工程涉及的是如何用系统化、工程化、以及形式化的方法来生产大型的、复杂的、高效的、以及可靠的软件系统。软件工程的发展主要是与传统的过程性（强制性）程序设计语言及相应的编程方法（范例）相联系的。可以说，程序设计语言在很大的程度上影响着软件工程的各个方面。逻辑程序设计与软件工程的关系正是如此。一方面，从软件工程的角度看，逻辑程序设计具有很多优点；另一方面，逻辑程序设计使得传统软件工程的许多概念、原理或者方法失去了原来的意义。

1. 逻辑程序设计的经济性

·文本更短。逻辑程序比完成同样任务的其他语言程序更简短。

·自动搜索。逻辑程序设计问题求解是基于符号串（模式）搜索的，但这种搜索是自动完成的，用户毋需作任何程序设计。

·自动合一操作。合一操作是逻辑程序执行的主要基本操作，这是一种双向模式匹配，也是由系统自动完成的，不必进行编码；换言之，搜索与合一操作占了逻辑程序设计和执行的绝大比重，但程序设计者却可以不加考虑（这也是逻辑程序设计低过程性的一个缘由）。

·多解。同一个逻辑程序可以用来求得一个问题的不同的解，而不必在程序中编码各自的求解逻辑。

·多查询。同一个逻辑程序可以被查询无限多

次、而且查询可以是任意的，独立于程序。

·数据库·逻辑程序系统用于实现（关系型演绎（数据库时，程序就是数据库，不用编制数据库管理系统软件；数据表示、操纵、以及查询都使用同一种语言。

2.逻辑程序设计的均一性 在逻辑程序开发的整个生命期，软件工程各阶段处理的都是同一种表示形式，即逻辑语句（一阶Horn子句）。由于各生命期阶段的输入输出的语法语义间隙小，因此各阶段的工作变得相对容易，并且有利于软件工程的自动化。

3. 逻辑程序设计与复杂性 对于构造大型程序系统，传统的方法是通过建立一个复杂的过程体系来完成，其中除每个过程本身的复杂外，尤为困难的是各过程（模块）之间的复杂联系。逻辑程序设计用简单的规则来取代复杂的过程，各规则之间也没有复杂的数据流联系（变量只局限于单条规则），不存在赋值副作用等，因此，大型逻辑程序的系统复杂性要更容易对付。

4.逻辑程序设计与软件设计方法学 传统的软件开发方法无非结构化编程和模块化编程。结构化编程考虑的实际上是程序中控制对于逻辑（众所周知，算法＝逻辑＋控制）的分配。由于在纯逻辑程序设计中没有控制成份，因此结构化逻辑编程就不再有什么意义。另一方面，逻辑程序的子句或谓词（关系）定义自然地构成模块结构。逻辑程序中考虑"结构化"的主要地方是关于逻辑程序中语句排序的安排，不过这主要是出于对执行效率的考虑。

5. 验证、测试与调整 由于逻辑程序的良好的基础计算模型，高度的抽象性和简洁性，以及直观和逻辑清晰性等，一般规模的逻辑程序的验证、测试和调整较之传统程序设计要更为容易。

6. 维护 程序维护为的是得到行为改善或调整的软件系统，包括程序的改进，修改，以及扩充等。逻辑程序的改进主要涉及执行效率，相对而言，这方面的活动余地不大。逻辑程序语句之间的联系简洁有利于修改效应的局部化，纯逻辑程序的单调性有利于程序扩充。

7. 自动程序设计 逻辑程序的自动生成或变换要较为容易，因为规格说明已很接近程序，且程序变换是从逻辑到逻辑。自动逻辑程序设计有利于采用形式方法，也有利于采用近来发展的基于知识的方法。

8. 元级处理与部分计算 逻辑程序系统提供元级处理和元一目标级混合编程技术。相应在新一代软件工程中，元程序设计，元解释器及部分计值技术等将起很重要的作用。

9. 并行处理 并行软件开发将是新一代软件工程研究的主旨。逻辑程序中固有的AND／OR并行性为开发并行程序提供了极大的便利。

10. 心理因素 逻辑程序设计的编制和理解要更为容易，这对于程序员的心理具有积极的影响。

五、 新一代计算与非标准逻辑

如前所述，人工智能几十年的研究揭示了知识是提高系统智能行为的关键。众所周知，人工智能的核心部分是知识表示和（基于知识的）推理。

逻辑业已成为新一代计算中知识表示和推理计算的主要工具系统。在知识表示和处理自动化中，经典的一阶谓词逻辑虽起着很大的作用，但是这种逻辑系统具有很大的局限性，主要是1)．难于处理不完全、不确定、不精确、和不一致等的知识以及动态变化的情况；2)．表达能力有限，很多种知识都无法表达；3)．推理效率不高，这在大规模知识库的情形尤为严重。这些不足严重地妨碍了新一代计算应用，因为要取得真正有智能的系统，必须要有深广的知识表示和有效的知识处理。

为了克服这些困难，人们研究采用各种非经典逻辑即所谓非标准逻辑来进行知识表示和推理计算。任何一种用于新一代计算的逻辑系统都必须具有一种良定义的知识表达语言，一种负责解释由这种语言表达的知识的意义的模型（理论）或语义，以及一种负责进行语法级自动知识处理和语句推演的证明理论。

用于新一代计算的非标准逻辑包括多（种）类逻辑（或分类逻辑），情形逻辑，非单调逻辑/缺省逻辑，多值逻辑，模糊逻辑，模态逻辑/动态逻辑，时态逻辑，认识（论）逻辑，高阶逻辑，内涵逻辑，制约逻辑，以及直觉（主义）逻辑/构造逻辑等，其中较为重要或应用较多的是分类逻辑，情形逻辑，

非单调逻辑，模糊逻辑，模态逻辑，以及时态逻辑等。

1. **分类逻辑** 与处理单种类论域的一阶逻辑不同，这种逻辑系统的论域中的实体可以有多个种类，这些种类之间的关系构成种类结构。

在分类逻辑中，指出逻辑元素（如变量）的类型减少了逻辑公式相互关系的复杂程度（如果两个公式中的相应的项或变量属于不同的种类，它们就不能直接发生关系），这在一定程度上减少（避免）标准逻辑推理系统中的一类指数复杂性行为。分类逻辑表示知识更为直观；分类实际上是一种元知识。它有利于知识库的维护（一致性／完整性检查）以及无意义知识项的检测；它能通过缩减搜索空间来改善推理效率。

如所周知，分类逻辑的表达能力与标准逻辑相当。

2. **情形逻辑** McCarthy及Hayes（1969）发展了这种逻辑，主要用于表达动态知识。在这种逻辑系统里，所有谓词都有一个附加变元，它指出使得谓词公式为真的"情形"；同一谓词在不同的情形可取不同的真假值。谓词之间的变化由所谓事件引起（事件改变了情形）。

这种逻辑系统适于处理变化的环境和动态的知识，它通过额外的信息来丰富和加深谓词知识项，从而使知识表示更加有力，推理更加有效。情形逻辑主要用于人工智能的机器人学和规划等分枝。

3. **非单调逻辑** 这是人工智能研究中最重要的非标准逻辑系统之一。标准逻辑系统是单调的，即增加公理集中的公理不会减少公理集所蕴涵的定理。设T，T′为公理集，P为某个定理，则有

if $T \vdash P$ and $T \subseteq T'$ then $T' \vdash P$.

如很多人所指出的，标准逻辑的这种性质在人工智能的很多应用中都不合理。

非单调逻辑避免这种不合理性。它主要用在下述情形：1）当知识信息不完全（新一代计算通常如此）时需事先作出某种缺省假设，当后来获得了相应的知识后，这些假设可能要推翻；2）处理过时的或变动的知识；3）在问题求解过程中往往要作一些临时假设，然后可能又要修改这些假设。

4. **模糊逻辑** 模糊逻辑自Zadeh于六十年代提出以来已获得极为广泛的研究和应用。这是一种广为人知的非标准逻辑，在新一代计算应用如专家系统中有越来越多的应用，主要用来实现常识推理及不确定性概念的表示和处置。

5. **模态逻辑** 模态逻辑涉及事物的状态和关于断言的不同的可能的世界。这种逻辑系统允许我们用虚拟语气语句而不仅仅是陈述语气语句来表示知识和进行推理，它可以表示和处理关于"一定是"，"应该是"，"可能是"，"相信是"，"希望是"，"将是"等的断言知识。

模态逻辑中最重要的是模态算子，由它构成模态语句。模态算子的一个重要的特性是它们可以构成其真值不依赖于被模态算子所操作的语句的真值的模态语句。早期的模态逻辑系统主要处理与可能性和必然性有关的知识，后来发展到处理与知道，相信，允许等有关的知识。

6. **时态逻辑** 时态逻辑用来表示和处理与时间有关的概念和动态知识，其思想与情形逻辑类似，即在论域中增加关于时间点的实体，这些时间点实体由通常的谓词相联系。

六、 新一代计算与连接系统（Connection System）

新一代计算的成功将依赖于并行处理。传统并行处理的能力是有限的，由于实现等方面的原因，其性能提高很难超过三个数量级。因此，新一代计算有必要探索新的潜力更大的并行计算模型及其系统实现方法。近年出现的所谓连接理论以及所谓的超并行计算机（Massively Parallel Computer）是这方面的重要代表。

连接理论（连接论，连接机制，连接学派，连接体系结构）在人工智能研究和应用中越来越引人注目。连接论是人工智能中的一种认知模型，它以神经生理学，认知心理学以及并行处理理论等为基础，其主旨是强调系统各处理结点（单元）之间的丰富连接而不是各个处理单元自身。连接论试图作为反映人大脑信息处理的一个合理的计算模型；一个连接系统（网络）由非常多的非常简单的处理结点通过丰富灵活的连接网络联系而构成。知识表示主要通过改变连接拓朴或连接强度来实现；基于知识的推理主要通过大量结点并行完成相联搜索、标记传播及集合运算等来实现。连接论提供所谓超并行

（下转74页）

为什么要开设《高级程序设计语言原理》课程

徐宝文（南京航空学院）

一、缘由

程序设计是计算机科学与工程的中心活动，而高级程序设计语言是进行程序设计的主要工具。计算机科学发展的每一步都在程序设计语言中得到体现。也就是说，计算机科学的发展史紧密联系着程序设计语言的发展史，计算机科学的许多问题都涉及程序设计语言的问题。因此学习与研究程序设计语言是极其重要的。

目前，各大专院校普遍开设了各种典型高级程序设计语言课程，如BASIC、FORTRAN、COBOL、Pascal、C、Ada等（dBASE等数据定义/操纵语言亦可归入此类），有的甚至还着开设了几门语言课。然而，一些学校对各专门语言课程的讲授重点一般都是如何用该语言编写程序，而不是语言的特点、风格、长处与不足，更不用说语言的原理了。因此，许多学生尽管学了不少语言，却不能很好地掌握程序设计语言的基本概念，不能较好地学习、掌握、分析、接受新语言。为此，近几年，我们在高年级本科生中开设了《高级程序设计语言原理》课程，并制定了教学大纲，编写了相应的教材，试图从实践与理论相结合的角度深入浅出地讲授高级程序设计语言的基本原理与结构。本文主要讨论与该课程及其教材有关的一些问题。

二、目的与用途

自FORTRAN0语言问世后的三十多年间，已经诞生了上千种高级语言。一般程序员或软件开发人员只能熟悉与使用这些语言中少数几个（充其量几十个）。对软件工作人员（不只是语言研究者）来说，学习与研究高级语言原理有以下诸多好处：

1. **加深对所用语言的理解**　对许多语言来说，使用得好与否会产生完全不同的效果，使用得当，可以给程序员带来诸如减少程序设计工作量等许多好处，反之，不仅会加重程序员负担、浪费机器空间与时间，甚至会导致逻辑错误与编码错误。例如递归，使用得当，可以直接实现某些优美而高级的算法，反之，则可能使程序执行时间成天文数字增长，误用递归还会发生大的逻辑错误。掌握了递归及其实现技术的基本知识，就能判断用得对的语言实现递归的开销从而决定在具体的程序设计环境中是否运用递归以及怎样运用。

2. **增进对更多语言设施的了解**　在程序设计中选择适合于所求解问题的数据与控制结构等时，程序员往往只考虑语言中可以直接表达的结构，在这方面本课程试图把现代语言中的各种基本设施介绍出来以增加学生程序设计的"词汇量"，从而扩大视野在进行软件设计时增加灵活性。例如，在许多程序中都要用到协同程序，而几乎没有一种语言直接提供这一设施，但如果一个FORTRAN程序员掌握了协同程序概念及实现技术就很容易用一组FORTRAN编程序来模拟实现所要的协同程序。

3. **更好地选择语言**　如果我们掌握了各种语言的知识，就可以为特定的应用项目正确地选择合适的语言，从而不仅可以大量减少程序设计工作量，少走弯路，还会提高程序执行速度。例如，用FORTRAN或COBOL进行字符串处理是很困难的，用它们编写的字符串处理程序既冗长又费时，但如果掌握了SNOBOL4，用它进行字符串处理就容易了。

4. **易于学习新语言**　语言学家掌握外语的速度比其他人要快得多，这是因为他掌握了语言的基本结构与原理。同样，一个程序员如果掌握了程序设计语言的基本原理、结构及实现技术，那么学习新语言就会变得很容易。

5. **有助于设计新语言**　即使不是语言设计者，也要掌握高级语言的基本原理，因为每个程序员至少都要编写用户接口程序，而用户接口实际上是某种形式的程序设计语言，它包括提供给用户与机器通讯的命令与数据格式，诸如正文编辑程序、操作系统、图形程序包等的设计者都或多或少会遇到与程序设计语言设计相类似的问题。

三、与相关课程的关系

与程序设计基础课及FORTRAN、COBOL、Pascal、C、Ada等具体语言课的关系　程序设计基础课介绍程序设计的基础知识，各具体语言课只是介绍如何运用现有语言编程序，学生限于对有关