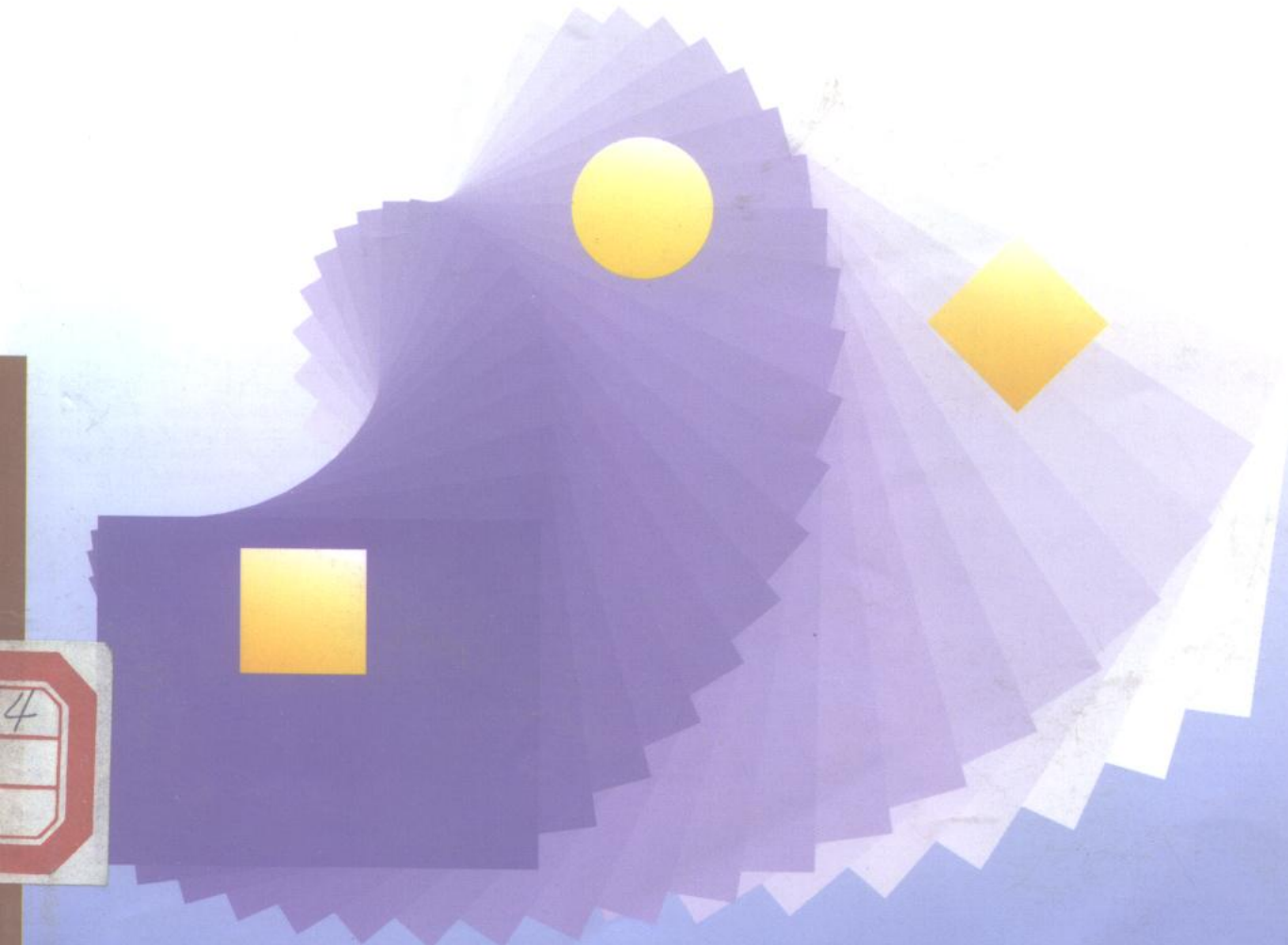
时序逻辑程序设计与软件工程

上 册

（时序逻辑语言）

唐稚松 等 著



科学出版社

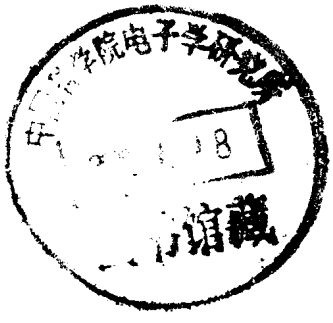
中国科学院科学出版基金资助出版

时序逻辑程序设计与软件工程

上册

（时序逻辑语言）

唐稚松 等 著



科 学 出 版 社

1999

内 容 简 介

本书旨在介绍一种面向软件工程的时序逻辑语言(XYZ/E)及以该语言为基础的支撑软件开发全过程的软件工程系统(XYZ 系统),目标是希望能为一般工业界用户服务,以提高软件开发的自动化水平及所开发的软件的可靠性与可维护性。

本书是作者近 20 年来研究成果的总结。全书共分上、下两册出版。上册介绍时序逻辑语言 XYZ/E,内容包括 XYZ 系统研制的技术和哲学背景,XYZ/E 的逻辑基础,XYZ/E 的基本特征和基本成分,XYZ/E 的控制结构,XYZ/E 中所表示的各种机制,XYZ/E 的实现,基于 XYZ/E 的实时程序设计,以及在 XYZ/E 框架内的程序规范与 Hoare 逻辑验证等。下册介绍软件工程方法与工具,内容包括基于模块程序设计的可视化图形工具,面向形式规范的逐步求精过程与速成原型方法,以可视化图形表示的体系结构描述语言 XYZ/ADL 及其在软件开发过程中的应用;除以上三种不同软件开发方法外,最后还介绍了基于共享变量的程序验证方法、语言转换工具及其在软件再造工程和某些专用领域的应用等。

本书适用于软件工程研究人员及教学人员,也能为实际软件工作者使用。特别希望本书对于关心可靠性及语义精确性的人们能有实用价值。

图书在版编目(CIP)数据

时序逻辑程序设计与软件工程 (上册): 时序逻辑语言/唐稚松等著. -北京: 科学出版社, 1999. 2

ISBN 7-03-007006-2

Ⅰ. 时… Ⅱ. 唐… Ⅲ. 时序逻辑-程序语言-程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字 (98) 第 26626 号

科学出版社 出版

北京东黄城根北街 16 号

邮政编码:100717

中国科学院印刷厂 印刷

新华书店北京发行所发行 各地新华书店经售

*

1999 年 2 月第 一 版 开本:787×1092 1/16

1999 年 2 月第一次印刷 印张:15 1/2

印数:1—2 000 字数:342 000

定价:25.00 元

(如有印装质量问题,我社负责调换(科印))

序

XYZ 系统是一种以时序逻辑语言 XYZ/E 为基础的软件工程工具系统，该系统的研制计划自 80 年代初开始实施，在国家多种科研经费的支持下，共历 3 个五年计划的不断工作与改进，终于在 1995 年夏在中国科学院软件研究所实现。从 1996 年起，我们一方面开展在实时过程控制、动画片等领域的一些应用研究，一方面又从理论与技术结合方面对这系统进行改进并提出一种新的方法与工具，且将其应用于实际。为便于今后开展有关 XYZ 系统的推广与应用，也为了适应我国在计算机科学与软件工程等领域培养高层次的研究及开发应用人才的需要，我们编写了这本书。它可以说是我本人以及我所领导的研究小组近 20 年来的研究工作的总结。在本书中，我们力求明确地说明在软件工程及形式化方法等方面有哪些是应该解决的方向性的、根本性的理论与技术问题，以及我们在 XYZ 系统中是如何解决这些问题的，并将力求说明我们所走的道路与解决问题的方法与当前国外主流方向的分歧所在及各自的得失如何，以期得到读者的理解，从而做出合适的判断。我之所以认为这样做是必要的，不仅是为了使我们的工作能得到读者的认可与支持，更重要的是希望能提醒我国青年同行：虽然在软件领域中，不论理论或技术方面，我国总的讲还比较落后，还应该向先进工业国的同行学习，但请注意，千万不要盲目追赶新潮，要以分析批判的态度对待他们的工作，这些工作不但可能有待改进，而且在某种条件下，甚至有时在重大的方向性问题上还长期走在值得怀疑的道路上。在这种情况出现时，不但可能旁观者清，而且因为我们所处的文化背景不同，思维方式有异，也许我们比他们更易发现问题并认清道路。古人所谓智者千虑必有一失，其信然乎！日本是以学习西方科学技术而闻名的，软件技术更是力追美国。可是近年来，日本也有人对此提出疑问。例如，以向日本产业界介绍外界高新科技最新情况著称的经济新闻社与日经产业消费研究所合办的《日经高科技情报》(日经ハイテリ情报)在 1993 年第 200 期发了一期“东方的智慧”专号，由该所研究员岛井弘之等写了一篇总论，其中有一段话说明办此专号的目的：“现代科学技术是否已陷入某种滞塞状态?期待甚高的高温超导和常温核裂变研究虽仍在继续下去，而其现有技术已迈进成熟阶段，但是它缺乏发展革新的苗头。这种停滞的原因也许包括西方理性主义(rationalism)对待事物的看法和行动原理，至少可以说是根本原因之一。当今是重新考虑现代科学基础的思想方法的好时机。目前世界上存在着许多与西方不同的各种文化思想区域，因此其中会隐藏着打开局面的思想。比如，自古代文明发达以来，中国人发明了印刷术、指南针、火药等丰富而实用的技术，其智慧最能够作为参考对象。”由此可见，日本人已开始注意到这个问题，我们岂可妄自菲薄?我认为，这不失为值得探索的达到知识创新目标的道路之一。当然这件事不能流于空谈，捕风捉影，只有在具体科学技术研究中作出由于具有我国文化特征的思想指导而确见实效的工作才有真正的意义。此外，我希望，我的上述看法不要被误解为鼓励人们丢掉踏实的研究作风而去做没有科学根据的比附与胡思乱想。

回顾近20年来XYZ系统的研制过程，深感在科研工作中走与世界主流方向不一致的道路是多么艰难。XYZ系统提出之初，虽得到美欧软件工程方面几位著名专家[如ISSI(国际软件系统公司)的叶祖尧[1]、CMU的N. Habermann及Trondheim的A. Solvberg等教授]的赞赏与支持[唐16,XYZ]，但由于这些工作在思想倾向上与西方理论研究的主流方向相背离，自然不易很快得到西方理论界的理解与赞许，因此他们虽然对于XYZ系统因其独特思路而乐意邀请我去介绍，但对其意义与价值却长期不作评述，实际上即表示怀疑。直至1988年才有两位英国著名理论家H. R. Barringer和D. Gabbay在一总结性文章中指出："将时序逻辑应用于软件工程的主要步骤即找到可执行时序逻辑"[BG]，并承认1983年我发表的介绍XYZ/E的论文[T1]是这方面最早的"先驱"。但这一评价也不过是"一叶知秋"而已。真正表示国际理论界对XYZ系统态度的变化是在90年代才发生的。1994年，时序逻辑的创立者、1996年图灵奖得主、以色列著名理论家A. Pnueli教授第一次访问我国。他在仔细参观了XYZ系统的演示后对我说："说实话，我过去一直认为你的野心太大，不可能成功，而这次看了XYZ系统的演示后，我发现你已经成功了。"在他与他的长期合作者斯坦福大学的Z. Manna教授的提议下，于1995年在北京召开了"逻辑与软件工程"国际研讨会。在他们向会议提交的论文前面有一段对XYZ/E的评语："唐稚松教授……将时序逻辑的理念处理得超乎任何人的想象，并将之应用在许多方面，在他之前，没有人认为这是可能的。"[MP8]接着，在他作为这次国际研讨会论文集的主编所写的序言中，更清楚地说明了他对以时序逻辑为基础的XYZ系统的认识的变化过程。他说："我仍然记得，当我首次听唐教授谈到以时序逻辑作为软件开发全过程……的统一基础时所感到的惊讶。随着时间的流逝，这一梦幻般的系统，由软件研究所一个致力于此的小组所实现并加以扩充，这系统……随之逐步成形。我的这种惊讶也渐渐转变成为钦慕。"接着，他说明以时序逻辑这种协调的形式语义理论作为软件开发全过程基础的重要意义。最后，他说："我盼望由唐教授所构想并发展的XYZ系统作为先驱所倡导的这一途径今后能引起软件工程系统的研究人员以及构造人员的巨大兴趣。"[PL]事实上，Pnueli教授所说明的他对XYZ系统看法的变化，正是近年来国际形式化理论界在形势迫使下已开始出现改变其原来思路的动向的一种反映。比如国际代数语义权威SRI(斯坦福研究所)的Mesequer教授将状态转换机制引入代数语义就是这方面的又一有意义的迹象。(他曾来函索取XYZ的资料，并说瑞士苏黎士ETH的Engeler教授向他推荐我们的工作。1998年2月在Dagstuhl召开的会议上，我们初次相遇，又提起Engeler教授向他推荐我们的工作；此外在这次会上，我还第一次遇到另一位代数语义的权威学者Wirsing教授，他也提到Ehrich教授向他推荐我们的工作。)

我认为，XYZ系统之所以过去长时间难以为西方理论家所理解，从根本上说是由于我们采用了一些我国传统哲学思想方法与西方流行的逻辑分析方法相结合的思路。这样

---

1) 叶祖尧对XYZ的评语："我心中绝对相信唐的XYZ语言将是一次重大突破的基础。……他的工作非常有创造性，而且有使软件生产率取得重大提高的实际应用前景。"Solvberg的评语："XYZ语言为我们提供了一种关于信息系统……软件功能描述的令人着迷的思维方式。……我看到，如果我们将唐教授的XYZ系统组织到我们的方法论基础中去，将有巨大的潜力推进软件设计与构造的……学科技术水平。"

的思路强调理论与技术紧密结合而不偏向一个极端，与西方片面重视形式化数学理论的理性主义思想差距甚大，但与国外（包括美、日、欧）一些对理论与技术不怀偏见、但较重视能实际提高软件生产率的软件工程专家（如叶祖尧、Solvberg 及日本软件工程学会主席岸田孝一等）的思路较为接近。这就说明，为什么在开始阶段 XYZ 系统从理论界与软件工程界这两方面所得到的反应是如此的不同！特别应该一提的是日本友人岸田孝一先生，他不但对西方软件技术十分了解，而且对中国传统文化有很深的素养，因此他对 XYZ 系统的哲学背景的兴趣也显得特别浓厚。他不但多次邀请我到日本作为软件学术会议的特邀主题报告人讲 XYZ 系统的哲学背景，而且亲自在《朝日新闻》（夕刊）上写专文介绍 XYZ 系统（1995 年 12 月 4 日）。文中说："唐教授的成就之一就是花了近 15 年的时间成功地开发了称为 XYZ 的软件系统。尽管系统所采用的最基础的数学理论来源于西方，但构造此系统的基本思想却来自孔子的中庸哲学和佛教的禅宗认识论哲学。这也许可以说是东方文明对于新的 21 世纪计算机技术发展的一大贡献吧。"[岸田]这一情况当然不易为西方多数理论家所理解，而瑞士苏黎士 ETH 的 Engeler 教授及德国 Braunschweig 的 Ehrich 教授等少数理论家算是例外。

在 XYZ 系统设计中，我之所以强调哲学思想指导，并不是出于为哲学而哲学的学院式兴趣，而是由于 80 年代初对国际软件理论与技术的发展潮流及其思想背景感到怀疑与忧虑所致。事实上，国际计算机科学家中有类似怀疑和忧虑的人并非少见，如斯坦福大学的 D. Knuth 教授就是较著名的一位[1]。

下面让我们回顾一下计算机科学技术的发展历史。如所周知，在 70 年代中以前，计算机科学领域的理论与技术是紧密相联的，如形式文法（包括属性文法）及语法分析方法的研究与编译技术是相互依存的。可是自从语义形式化问题提出以来，由于这问题难度很大，希望从传统数学理论中找新的工具与方法的要求很普遍，许多有才能的青年数学家逐渐进入这一研究领域。他们在软件技术方面的根基不很深厚，但受理性主义传统影响很大，数学的职业倾向性远远大于软件的职业倾向性。从 80 年代以来，在这类人员集中的机构、地区与会议上，其价值标准（如脱离实用意义，单纯从理论的逻辑严密性及深度与难度对工作进行评价等），以及学术气氛也就逐渐发生了变化，从而不能不影响学科发展方向，使计算机科学理论研究日益脱离软件的现实实用性考虑而成为一种较深的数学探索。这种情况自然难被关心市场效益的工业界所接受。特别是以实用主义传统著称的美国工业界，他们虽然也很关心提高软件生产率，关心软件的可靠性与可维护性这类关键问题，但他们对日益远离工程技术实用性的形式化语义理论及规范语言的研究逐渐失去信心与兴趣，终于导致几乎完全抛弃了理论研究，而单纯从技术上找出路。他们企盼软件像硬件一样，用从技术上提高自动化水平或可重用性的途径找到提高生产率的方法。这种思路大大推动了软件工具及与之相联系的操作系统的发展。其后果是整个 80 年代，对于软件研究与发展来说，就是一个理论与技术相互分离脱节、各走极端的年代。这种情况在一段时间

1）1995 年他在访问 Duke 大学时，与我过去一位名字叫金伟的学生（现在该校作博士生）谈到过我，他说："唐教授是我遇到的唯一一位关心计算机科学在 10 年、20 年后如何发展这个问题的中国计算机科学家。因此，我建议斯坦福大学计算机科学系邀请他来访问。"事实上，XYZ 系统的设计思想正是这次访问时开始形成的[T13]。

内使双方都得到一定程度的满足，理论家获得了很高的荣誉，而工业界也获得了不小的利润。可是，提高软件生产率，及提高其可靠性与可维护性等根本性问题不但没有解决，而且似乎希望日益渺茫。这种情况是否合理？这些根本性问题是否已失去意义？事实并非如此。不但像美欧各国关键技术委员会每年一次向国家最高当局提出的报告中都将这个问题放在首位；而且甚至像微软这样一贯强调软件技术且已取得大量利润的公司也已感到，由于缺少精确语义基础，大型软件的可靠性与可维护性问题已制约他们进一步的发展，因此不久前该公司已在剑桥大学等处投入巨资开展这方面的研究。事实上，语义的精确性与技术的自动化二者是相互依存不可分割的，而且只有紧密结合才能达到提高软件生产率的目标。因此，这个问题如果不从根本上找出理论与技术脱节的病根来辨明道路，单靠投入资金岂能解决。那么，产生这个问题的根本原因何在呢？

40 多年来，有关计算机科学的研究实际上是围绕以某种模型为基础的三个彼此相关的层次进行的，这三个层次即计算机体系、适应该体系的可执行程序语言及表示程序抽象语义的规范语言。如所周知，40 多年来流行的计算机体系都是建立在冯·诺依曼(von Neumann)模型上的，流行的程序语言即为适应这种机器体系的常见命令式语言(如 Pascal，Ada，C，C++等)。这种模型主要的特征即为自动机状态转换机制，通俗地说，表现为变量在运行过程中可通过赋值命令(语句)改变其值，且其控制流可分解成“循环”、“分支”及“继续”等基本结构，其中尤以“循环”最具特殊的意义。由于这些特征使这种语言能高效地在冯·诺依曼型计算机上运行。至于适应这样模型的规范语言，人们较为广泛接受的是 Hoare 逻辑中的由前置断言(pre assertion)与后续断言(post assertion)所表示的逻辑语义。以上三个层次对冯·诺依曼模型的表示方式相互间是紧密关联，协调一致的。它有其优点与弱点，优点是其常见命令式语言执行效率高，为广大计算机用户所习用(特别是用循环表示重复计算)，在 Hoare 逻辑的作用下，命令式语言的程序与由前置和后续断言表示的规范协调地联系在一起，而且对于数学水平不高的初学者来说，按照自动机状态转换方式设计一规模不太大的程序或模块较为直观而易于掌握；其弱点是这种基于自动机状态转换的机制不够抽象，包含细节太多(赋值本身即是一种细节)，不能像数学中常用的函数那样便于嵌套与连接，平常数学中许多理论与技巧(如证明技巧)不易在其中直接应用等等；而更重要的是，在这种命令式语言与规范语言的关系中，后来发现一些语义方面的难题长期找不到解决的办法。比如，常见命令式语言中递归过程(及进程)这类模块的后续条件如何表示其递归性并如何验证其正确性？更困难而又更重要的一个问题是，对于由通信进程组成的并行语句，如何表示其语义？如何验证其正确性？事实上，在冯·诺依曼模型的框架内，唯一可行的办法是将并行语句的语义归结为该语句所包含的串型进程语义的合取，但这一命题成立的条件是语义可组合性。而事实表明，在通信的情况下这一条件是可能不成立的。因此，困难在于如何保证并行语句语义可组合性条件成立。容易看出，上述优点是工程界所重视的实用性方面的优点，上述弱点是理论界所强调的数学精美性方面的弱点，而上述困难问题则的确是冯·诺依曼模型所面临的实质性问题。不解决这些问题必将严重影响程序的可靠性与安全性，将使冯·诺依曼模型不论从理论上还是从工程上说都将难以采用。(C. A. R. Hoare 在文献[Ho2]中讨论 CSP 的数学模型时明确指出，他在文献

[Ho2]中放弃了文献[Ho1]中的语义模型，而采用了与 CCS 相同的进程代数所要求的函数式模型，其中三条原因中的前两条就与这里所述的两项困难问题直接相关。）由于上述情况，从 70 年代末起，计算机科学领域出现寻找其他非冯·诺依曼模型的高潮。其中最有影响的即函数式模型的研究，包括函数式机器体系、函数式程序语言，以及以递归性与函数映射为特征的基于可计算函数（包括 λ 演算）及代数的语义理论与规范语言等三个层次的研究。这些研究与上述基于冯·诺依曼模型的三个层次研究可以说是相互对立的，一方的优点正是另一方的弱点，两方实际上是难以协调的。由于函数式模型具有与传统数学理论一致的许多特征，从 80 年代以来有较强数学兴趣的计算机科学家参与这方面语义理论及规范语言研究的人越来越多，在理论研究方面形成热潮，特别是西欧受理性主义传统影响较深的学术机构更是如此。因此，可以说，从 80 年代到现在计算机科学理论界（特别是以英、法、德、荷、意等国为代表的西欧）是以函数式模型为基础的理论研究占绝对统治地位的年代。但另一方面，这种理论研究所依附的基于函数式模型的计算机体系则很不成功，这类计算机造价昂贵、效率低，而且功能很有局限性。这是与现在流行而且在可预见时间内难以取代的硬件基础紧密相关的，由于 RISC 技术及工作站流行，这类函数式机器已退出市场。这方面所长期探索的研究似乎已销声匿迹。处于这类理论及机器体系之间的函数式语言研究虽多数已消失，但仍有少数（如从爱丁堡发源的 SML）因在某些领域（如作为表示形式理论的验证系统的工具）有其应用价值而在学术界保持其生命力。这一情况与下述情况颇为相似：长期以来尽管冯·诺依曼计算机及与之相应的命令式语言占绝对统治地位，但函数式语言 Lisp 在人工智能领域仍保持其生命力。类似地，估计今后像 SML 等少数函数式语言在某些专用领域可能会长存不衰。可是，只要冯·诺依曼计算机维持其在机器结构方面的统治地位，则函数式语言因与机器体系不一致而影响其效率与功能，故不可能在可执行语言领域占统治地位，广大用户和工业界仍将会以使用基于冯·诺依曼模型的命令式语言为主，这一点是无可怀疑的[1]。现在的问题是关于形式语义理论及规范语言研究怎么样？我认为，只要机器体系及可执行语言是以冯·诺依曼模型为主流，若理论研究却反其道而行，坚持以函数式模型为基础进行，则必然出现的结果是要么理论与技术脱节成为一种近期与技术无关的纯基础数学研究，要么这样的理论终被抛弃。从长远来看，两者必居其一，也就是说，“皮之不存，毛将焉附？”从 80 年代初以来，XYZ 系统的研究即是基于这样的信念开始的。我们走的是一条以冯·诺依曼计算机为基础、面向常见程序设计的理论与技术紧密相联系的研究道路。作为少数派，孤单地走这条道路是艰辛的，为此所必须解决的问题也是十分困难的。但我认为只要思想方法正确，坚持不懈，终能找到解决困难问题的途径，因此对这条坎坷道路的前途一直满怀信心。事实上，几乎与我们同时，得克萨斯州（Texas）大学奥斯汀分校的 Chandy 与 Misra 教授很可能是在 Dijkstra 教授的影响下，也走上了与我们类似的面向冯·诺依曼模型建立形式语义理论与可执行

1）一些著名计算机科学家致力于提高爱丁堡 SML 语言在冯·诺依曼计算机上的执行效率，希望将它改造成一通用程序语言以取代常见命令式语言。对于这一目标是否能完全达到，我的观点是消极的。因为这目标等于要求建立在一种模型上的语言有可能在另一种与之对立的模型的计算机上有效执行，且其效率与建立在后一模型上的可执行语言的执行效率不相上下。这实在是件不可思议的事。

语言相结合的道路。这就是他们关于UNITY语言的研究。这是一可执行命令式语言，为了使此语言在并行情况下具有语义可组合性，他们建立了一整套复杂的理论，并对这种语言进行很多的限制，最后终于解决了以冯·诺依曼模型为基础保证在并发情况下该语言的语义可组合性的难题。从这方面说，该系统是成功的。Dijkstra与Hoare对此语言系统作了极高的评价并予以推荐。该系统问世之初曾经引起了理论界与技术界的注意，但很可能是由于这一语言受限制太多，与常见命令式语言风格差距很大，且表示力所受影响很深，以及它所依据的理论是专为此语言所建立的，相当复杂，一般用户很难深入掌握；此外，它的目标似乎是取代流行的常见命令式语言，而不是为常见命令式语言服务，为它们提供一坚实的语义基础，因此，难以引起广大习惯于使用常见命令式语言用户的兴趣。总之，由于各种原因，这一语言系统问世10年，至今已少有人谈及，也未见到重要的应用效果。无论如何，它即使不被完全抛弃，也既不大可能取代流行的命令式语言，也未能为这些命令式语言提供一合适的理论基础，补充其这些方面的不足，以达到使冯·诺依曼模型基础上的机器体系、可执行语言及语义理论与规范语言三个层次紧密地联系起来的目标。看来，这一使命不可避免地只有落在XYZ系统身上。为此，XYZ系统中提供一时序逻辑语言XYZ/E，它既包含表示规范的子语言XYZ/AE，也包含可在冯·诺依曼计算机上有效执行的子语言XYZ/EE，两方均具有与常见命令式语言相同的控制结构与数据结构，其风格与表示力与常见命令式语言无异，后者可作为常见命令式语言之上的中间语言，而前者可在语义描述及规范语言方面做常见语言的补充；它的理论基础，即Manna-Pnueli线性时间时序逻辑，并不要求用户专门进行复杂的理论训练即可掌握，所以就XYZ系统而言，不存在妨碍UNITY推广应用所发生的那些理论与技术上的困难。更重要的是，前面所述的那些冯·诺依曼模型面临的难题，如并行语句的语义可组合性问题及递归过程规范的表示与验证等问题，都已简单而自然地在XYZ系统中得到解决（见本书第七章）。因此，现在我们可较大胆地说，XYZ系统已较圆满地为冯·诺依曼模型提供了一适应冯·诺依曼计算机体系且理论与技术紧密联系的逻辑语言及软件工程系统。据我所知，当前世界上似乎尚未见到其他同类系统做到了这一点。事实上，正如Pnueli指出的，这一方向正是XYZ系统作为先驱所倡导的。

由于基于技术实现软件生产自动化的工具研究（特别是依靠人工智能实现这一目标的努力），效果并不显著。自80年代中以来，以美国工业界为基地大力发展起以提高可重用性为目标的提高软件生产率的模块程序设计的研究，这也就是面向对象程序设计的潮流。先是以C++为代表的，近年则是以Java为代表的程序语言系统已在美国工业界引起广泛的注意。可以说，这一以软件技术为基础的提高软件生产率的道路是颇有成效的。但在我们看来，它仍存在以下两方面的问题：第一，这一方向对语义精确性问题未予足够的重视，因此可靠性方面仍无足够的保障。比如由于面向对象程序设计的特征之一是可重用性，忽略了形式规范的情况下要保证重用时的语义一致性则只有依靠测试（testing）。可是对于具有继承性的模块而言，测试变得更为复杂与困难。第二，分布式面向对象模块中并发通信将起核心的作用。因此，由并发通讯所引起的语义正确性问题，是无法回避的。这个问题不可能单纯从技术上依靠模块重用来解决。

下面我们对 XYZ 系统的具体特征概括地给予介绍：

1)我们的目标是语义精确性与技术实用性的统一，精确性与实用性缺一不可。在这方面我们的要求是严格的，不容让步的，因为只有这样才能确保提高软件可靠性、可维护性的目标。可是，语义精确并不等于理论精美，虽然在理论完美性与技术实用性这两方面，我们都很重视，但如果双方发生不易调和的矛盾，则在不损伤语义精确性的条件下，我们宁可在理论完美性方面做适当的让步，以保证技术的实用性标准。因为在我们看来，计算机科学毕竟是一门技术科学，它与纯粹数学不同，忽视技术实用性必将失去技术科学的应用价值，最终导致理论与技术脱节。既然冯·诺依曼型计算机是当前及今后可预见的时间内唯一被采用的计算机体系，计算机科学理论与可执行语言理所当然即应适应这种机器体系及其依据的模型。这才是保证理论与技术协调的合理道路。但是如果为了作到这一点而出现关于语义理论及相关技术方面的困难，则是必须解决的问题，不能回避，否则其可靠性无法保证。XYZ 系统的研制过程即是以此为目标，面对这些问题逐步加以解决的历程。事实证明，只要采取正确的思路，往往可以找到出人意料的解决问题的方法。在这一方面，我们是有较多的体会的。我坚信，将我国传统哲学思想与西方逻辑分析方法相结合，常能为我们提供一些巧妙的思路。我希望在本书中我们能够结合理论或技术问题对此有所说明。

2)时序逻辑语言 XYZ/E 的一个重要特征即强调动态语义与静态语义并重，将两者结合而不只偏向一方。由于适应冯·诺依曼机器体系的可执行语言的特征即表示自动机的状态转换机制，其方式是命令式的，其语义是动态的；而适于这种模型的规范语言，即 Hoare 逻辑中的 Pre 与 Post 断言(用时序逻辑表示，即 Pre→◇Post)，其语义则是静态的。我认为，将规范所表示的静态语义与可执行语言所表示的动态语义紧密联系起来是使这方面的研究更能在实际软件工程系统中发挥实用价值的关键(这也就是前面所引 Barringer 与 Gabbay 对可执行时序逻辑在软件工程中的作用的评价)。为此，在时序逻辑语言 XYZ/E 中，应找到一种控制框架将规范的静态语义与可执行的动态语义统一地表示出来。这就构成 XYZ/E 的一个重要特色，即表示动态语义与静态语义(即 XYZ/EE 与 XYZ/AE 中)的两种命令(语句)可混合在一程序中出现，用这种混合出现的程序，即能表示出由完全抽象的规范到完全可有效执行的程序之间平滑过渡的过程。在 XYZ 系统中，就是以这样的方式表示逐步求精方法。这种逐步平滑过渡恰好起着理论(表示静态语义的断言)与技术(表示动态语义的可有效执行的程序)相联系的桥梁作用。

动态语义与静态语义的联系，不但反映在抽象规范与可执行程序的关系上，而且也反映在程序的模块结构上。如具有流程图结构的过程与进程显然是表示动态语义的，而作为面向对象程序设计的具有封装性(encapsulation)与继承性(inheritance)的对象模块，显然是面向问题领域的，故其语义是静态的。为了将这两方面联系起来，如上所述，在 XYZ 系统中提供一种面向基于通信的计算过程的模块，称为并行语句进程(PSP)，用它即可将大型程序中静态语义与动态语义联结成为一个整体。

最后，还有一个与语义有关的问题，即如前面所述，XYZ/E 可分成不同层次的子语言。其最基础的一层是表示可有效执行程序的 XYZ/EE，而且这些程序所表示的风格与常

见程序语言是相同的。人们不禁要问，既然已经存在许多流行的程序语言，如C，C++，Concurrent C，Java等，提出XYZ/EE这样的时序逻辑语言还有何必要？一方面，由于它作为具有统一结构的时序逻辑语言的最底层，可与表示不同抽象层次规范相联系，来表示逐步求精过程，从而提高所书写程序的可维护性与可靠性；另一方面，XYZ/EE语言可以看成是一种中间语言，它的程序可在保持其执行效率的前提下自动转换成Unix上的C程序、Java程序，或适应微软操作系统的程序。这样，即可使用XYZ/EE书写的程序具有适应不同软件平台的可移植性。这种特征是很具有实用价值的。

3)关于XYZ系统中工具所实现的方法论，有如下几方面的内容：

①归纳"2)"中所述，XYZ系统所支撑的程序设计可以说是由纵向与横向二维正交而成。纵向方法论即前面谈到的由抽象(静态语义)到具体(动态语义)的逐步求精方法，其中包括不同抽象层次的规范描述、语义一致性检验、验证及速成原型等各种方法及相应的支撑工具，这种设计与语义由静态向动态过渡相关；至于横向方法与工具，则是为了支撑模块程序设计。对此，XYZ系统中将针对各种不同类型模块提供相应的可视化图形设计工具。但是这两种程序设计方法及相关的工具，只是从形式化语义理论及模块程序技术两方面各强调其中的一方面，并未能将二者有机地结合起来。为了达到本书所强调的将这两个方面在时序语言XYZ/E的基础上协调地结合起来的目标，XYZ系统又提出了第三种软件工程方法与工具。近年来，CMU的学者们(D. Garlan，M. Shaw等)特别强调软件体系结构的选择在软件开发过程中的重要意义，我们针对XYZ系统的特征提出一种可视化软件体系结构描述语言XYZ/ADL(Architecture Description Language)，并以它作为应用XYZ系统设计软件时的用户语言。它以组件(component)作为一软件系统的基本构件(事实上它是模块概念的推广)。每一组件包含二个方面：一是其外部界面(interface)，即以其规范来表示；另一是其内部结构，即其体系(architecture)，它以XYZ/ADL的图形来表示。由前者向后者转换，即构成软件开发过程的一步过渡(transition)。因一组件的体系结构中又包含下层的组件，它们亦由上述两方面构成并进行过渡。这样的逐层过渡即构成开发该软件系统的逐步求精过程。对于XYZ/ADL而言，每当描述一组件的体系结构后，即将自动生成一描述该体系结构语义的XYZ/E程序。将此XYZ/E程序与该组件的规范相互结合即可应用XYZ系统中提供的验证或模型检验(亦称速成原型)工具检验这一步过渡的语义一致性，这样即将前二种方法的主要特征结合起来了。在XYZ系统中将提供关于软件开发的上述三类软件工程方法与工具。在开发一软件时，每一种方法与工具均可独立使用，也可将该软件划分成语义独立的模块，分别各用上述一种方法与工具进行开发，然后再(最好是以通信的方式)将它们联成一个整体。在本书下册第八、九、十这三章中将分别对这三种软件工程方法与工具及相应的逐步求精过程作较详细的介绍，并将以具体例子作为示范说明。

②在大型程序设计中往往出现这种情况，即有时需要在某些部分嵌入用其他程序写的久经使用的程序，在软件再造工程及某些专用领域应用系统中有时也有与此类似的需要。为了提供用户处理这类问题的手段，XYZ系统中提供一种将用其他语言书写的程序转换成XYZ程序的简便的形式化方法。我们已用此工具将SDL，ESTELLE，VHDL等这

些专用领域国际标准语言转换到 XYZ/E，看来效果相当不错。关于这部分的讨论即构成本书下册第十二章的内容。

XYZ 系统的特征也就决定了本书的特征。本书既不应是一本主要讨论时序逻辑理论的专著，也不应是一本主要介绍各种软件工程工具技术的教程。它旨在较全面而准确地介绍 XYZ 系统。在介绍时序逻辑语言 XYZ/E 时，一方面首先是面向程序工作者，将 XYZ/E 作为一程序语言，而不是作为一逻辑系统来介绍；但另一方面又应随处指明它仍保持其为时序逻辑理论系统的特征；在介绍各工具时，一方面首先将其作为软件工程工具来介绍；但另一方面又随时说明它与时序逻辑语言 XYZ/E 的内在联系。我深知，这样一本介绍兼具理论与技术双重特征的系统的书，本身存在着内在的矛盾，它所具有的“中道”性质的陈述方式很可能同时引起来自理论界与工程界双方的不满。我记得在钱学森先生所著《工程控制论》英文版的序言中也曾谈到：该书的陈述方式很可能“按数学著作的标准看不够严谨慎重”，而“从介绍工程系统的著作的标准看又包含数学理论太多”。我在书写本书时，实在也感到类似的困境，深觉无法避免。我们认为对于这种深层的矛盾，采用“中庸之道”的方法来对待是较为合适的，这也许正是“技术科学”的内在特征所在，望能得到读者的谅解。

由于 XYZ 系统规模十分庞大，研制时间长达近 15 年，且参加人员流动性很大，这就决定了它的研制人员的结构具有如下的特征：一方面这一系统只可能是集体完成的工作，另一方面为了使这一系统不致成为一松散无章的堆砌物，它又应是一个在一人的思想指导之下按照非常紧凑统一的设计思想与理论完成的系统。事实上，XYZ 系统正是如此。近 20 年来，参加人员 50 余人，平均每人参加时间为 2～3 年；他们绝大多数参加了各子系统或工具有关设计的讨论与实现，唯我一人一直从头到尾主持并实际参与此项工作。其语言设计、工具所支撑的方法及解决关键技术难题的方案研究等，除了少数部分外，主要皆由我负责。当然，不少人也参加了讨论并提出过很好的建议，为了节省篇幅，很难在此列举各参加人员的参与情况。不过在本书后面参考文献中，可查出对 XYZ 系统作出过贡献的人员的姓名（个别人员例外）。本书撰写情况也与上述情况相似，自然也是由我负主要责任，从总体思想到 XYZ/E 语言及各种软件工程方法的设计与介绍都由我完成，而对各工具系统具体实现的技术内容的介绍，则一般由负责实现或修改该工具的人员中目前仍在我组工作或学习的人执笔，但最后在文字上由我加以统一。在书写有关部分、调试例题、对全书进行录入、编辑排版以及其他各项工作中，赵琛、沈武威起了较大的作用，谢育涛、高永祥、闫安、沈伟、唐小平、张小格、骆华俊等都负责完成了与他们有关的部分。中国科技大学的孙淑玲教授与她所领导的科研集体，贵州大学的李广元副教授分别编写了本书第十二章（即语言转换）、第七章与第十一章有关的内容和附录 Ⅱ，北京航空航天大学的张玉平副教授曾参加过附录 Ⅱ 初稿的撰写工作。此外，石民勇博士、王春江博士先后阅读了各章，并检查了各方面可能出现的疏漏，特别是王春江博士其贡献尤为突出。我所副研究员柳军飞曾在完成本系统“八五”计划的项目研究中起了很重要的领导组织作用。本系统及本书得以完成，的确是这个集体所有成员共同努力的结果。值得令人引以为慰的是，近 20 年来，XYZ 小组一直是一个团结的、有向心力的集体，其成员不但在组内工作与学习时是如此，

甚至他们离开多年(现在大多数都在国外),仍对这个集体的工作予以关心,经常为 XYZ 系统的完成提供各种帮助。在本书完成之际,我应指出,如果 XYZ 系统将来被证明是一项有价值的工作,这成绩首先应归功于这个集体。还有一点应当指出,本书下册中许多关于 XYZ 的系统应用的介绍都是基于我组人员与其他单位人员合作应用 XYZ 系统所得到的成果。在此我们对这些兄弟单位及有关人员表示谢意。

最后,借此机会感谢为 XYZ 系统的研制以及本书的出版提供各种支持、关心以及帮助的单位与友人。近 20 年来,我国国家自然科学基金会、国家科委 863 计划、电子工业部一直提供经费支持。中国科学院及我先后所在单位,如计算技术研究所与软件研究所的负责人,一直对我们的工作给予信任、支持与关心,如果没有中国科学院,特别是软件研究所基础部这样的科研环境,我相信,XYZ 系统是不可能完成的。此外,多年来国内外许多朋友也一贯为我提供了各种精神与物质的帮助。在此,我还应特别提到几位国外朋友对我工作的支持和帮助,他们是 J. McCarthy, D. Knuth, Z. Manna, A. Pnueli, R. T. Yeh(叶祖尧), N. Habermann, D. Bjorner, E. Engeler, H. D. Ehrich, A. Solvberg, K. Kishida(岸田孝一)等,过去我多次应邀去他们所在的大学或研究机构的访问对 XYZ 系统的形成很有影响[T13]。在本书的出版过程中,中国科学院科学出版社的有关同志也一直予以支持。最后,还应该特别提到的是 30 余年来与我冷暖相依的伴侣童恩健,以及我的两个儿子其深和其放对我的工作与生活的支持与关心。对所有这些朋友与亲人,我谨在此表示由衷的谢意。

唐稚松

1998 年 10 月

目　录

上册　时序逻辑语言

下册 软件工程方法与工具
（预告）

第一章 绪 论

本章着重从技术发展历史与设计思想两个方面说明XYZ系统的背景与意义。1.1节分析程序技术研究30年来的发展情况，用以说明构成XYZ系统的两个主要方面——作为其统一基础的时序逻辑语言XYZ/E与该系统中所包含的各种CASE工具的功能、特征及意义，以及这两方面的关系。1.2节则从哲学方法论的角度说明软件工程领域过去10多年来以西欧学术界为代表的理性主义道路及以美国工业界为代表的实用主义道路这两个极端所显示的片面性，从而说明XYZ系统在设计思想方面的哲学特征及其意义。最后在1.3节中，扼要地说明XYZ系统在时序逻辑语言方面与CASE工具方面的主要内容，从而使读者在深入学习XYZ系统各方面理论与技术细节之前，对此系统的轮廓及来龙去脉有一较全面的印象。只关心本书的理论与技术内容，而对其历史背景、设计思想及有关的哲学方法暂不想都关心的读者，对本章前两节内容可省略不读。

本章旨在使读者在第一次阅读时对本书的轮廓留下一较全面的(但可能是模糊的)印象，然后读者即可以顺利地阅读以下各章。当在后面阅读时遇到本章提到过的内容，不妨回顾本章中相应的内容，也许可以帮助读者在了解以下各章内容时，能知道它们并非是彼此孤立的，从而逐步弄清来龙去脉。

1.1 程序技术研究30年

30多年来，程序技术研究的一个主要问题是如何提高软件生产率，解决这一问题是较困难的，因为计算机程序是一种极为复杂的人工制造的实体，非常难读、难写，易于出错且难以修改和维护，从而其生产自动化水平很低。对比之下，计算机硬件自动化水平在过去10多年中却提高得很快，软、硬件生产率很不匹配。由于计算技术的飞速发展，机器硬件的性能日益增强，而造价却大幅度下降，加上应用的开展使软件系统规模越来越大，其结构越来越复杂，软件生产率低的问题更显突出。过去30多年程序技术研究的发展，围绕如何提高软件生产率的问题，可分成以下三个时期：一是高级语言时期，二是结构程序设计时期，三是软件工程工具时期。通过对这三个时期情况的介绍与分析，人们不难看到程序技术研究发展的脉络与趋势。我们深信，在介绍XYZ系统之前先对这段历史作一回顾，将有助于读者对XYZ系统的了解，也有助于读者将XYZ系统与类似的系统相比较时，对其得失作出合适的评估，因为近20年来XYZ系统研究与程序技术研究是沿着同一路径开展的。

1.1.1 高级语言时期

大致从50年代末起，人们已开始认识到，为了解决程序难读、易于出错及难以维护的问题，很应该将书写程序时所用的程序语言与机器上执行的机器语言区分开来。后者主要表示程序如何执行，即表示“怎么做”(How to do)，而前者主要表示程序的含义或功能，即

表示“做什么”(What to do)。提高程序设计生产率的途径应该是:一方面设计出表示力强,能直接表示程序含义的程序语言,用户即用这种易于理解的语言书写程序;另一方面还要找到一种有效的方法,它能实现由上述程序语言的程序到可在常见的(即冯·诺依曼型的)计算机上高效执行的机器语言程序的自动转换,这种转换由计算机来完成。60年代的巨大成就之一,就是在当时的水平上成功地解决了这两个问题:一方面从 Fortran 及 Algol60 开始设计出了具有高级控制结构及数据结构便于表示算法的程序语言,即所谓高级程序语言(简称高级语言)或算法语言;另一方面又发明了将高级语言程序变成机器语言程序的自动转换技术,即编译技术。这一巨大成就给人们带来的欣慰大约维持了近10年。然而,随着计算机应用的广泛开展,软件规模与复杂性迅速增大,特别是随着以并发性为特征的系统程序的发展,高级语言的局限性也日益显露。当程序复杂性增加到一定程度以后,用这类高级语言书写程序时其中包含的细节仍然太多,太复杂。这样的程序仍具有难读、难写,易于出错和难以维护的毛病。换言之,关于如何保证一程序正确及便于维护这些方面的难题仍有待解决。从70年代开始,程序技术研究即围绕着程序可靠性与可维护性这些问题而展开的。

1.1.2 结构程序设计时期

如何保证程序可靠性的讨论,大致是从三方面的问题展开的:程序语言、方法论、软件开发过程的管理。

1. 程序语言

大致说来,这一时期关于程序语言的讨论是沿着三个不同的方向开展的:

1)改进常见高级语言的控制结构与数据结构,即**结构程序语言**(Structured Programming Language),或简称为**结构化语言**(Structured Language)。

2)提出新的程序模块概念,这种程序语言后来称为**面向对象**(或**基于对象**)**的程序语言**(Object-Oriented or Object-Based Programming Language)。

3)提出直接描述程序的抽象含义的新语言,即**抽象功能描述语言**或**描述语言**,亦称**规范语言**(Specification Languages)。

以下从这三个方面分别讨论。

(1)结构程序语言

关于这类语言的讨论在结构程序设计时期中开展得最早。一般以 Dijkstra 1968 年致“Communications of the ACM”的一封著名的信作为标志。他认为,原来常见的高级语言的主要缺点在于控制结构,特别是转(goto)语句。它使程序的结构变得复杂,使程序的整体不能分解成条理分明的局部组合,从而使程序难读难懂,易于出错。“结构程序设计”这一名词也就是由此而得。这一讨论的主要成果是产生了由 Pascal 到 Ada 这一序列的结构化语言。这些语言的确比原来的常见高级程序语言有很大的改进,具有较为清晰的控制结构。但这仍只是一种改良,它只改进了程序语言的结构,并未提高其抽象性和直接表示程序含义的能力。当程序的规模与复杂性增大到一定程度时,它仍显得包含细节太多,同样具有难读、易错及难于验证其正确性等方面的不足。正如卡内基-梅隆(Carnegie-Mellon)