计算机科学 1995 Vol. 22 №. 1

软件开发过程中的形式化方法

朱 冰 梅 宏 杨芙清

（北京大学计算机科学与技术系 北京 100871）

摘 要 本文简要介绍了软件开发过程中使用的形式化方法、典型的规范说明语言以及有关的实践活动。

一、引言

软件系统的开发过程是一个从客观世界到认识世界、再到程序世界的协调过程。按照软件工程方法论，从认识世界到程序世界的转换须先写规范说明，再据此规范说明实现程序。规范说明的主要任务在于说明“做什么”，程序实现主要解决“怎么做”。为了实现符合要求的软件系统，规范说明应具有可理解性和精确性，这是程序实现的根本保证。因此，将形式化的理论和方法用于规范说明极为必要，也是实现软件自动化生产的根本前提。

一阶谓词演算是一个形式逻辑系统，具有一整套证明技术，采用一阶谓词演算方法进行规范说明曾被认为是很理想的。将一阶谓词演算与程序联系起来，实际上要解决两个问题：将谓词演算转换成程序和将程序转换成谓词演算；前者是软件自动生成问题，后者是程序验证问题。众多形式化方法中，有三种已为工业界接收和使用的形式化方法，它们是Hoare的基于程序逻辑的公理方法，Dijkstra的基于最弱前置谓词和谓词转换器的方法，和Jone的VDM方法[15]。从Hoare提出程序设计公理方法算起，许多计算机科学家为之奋斗了20多年，但在软件自动生成和程序正确性等方面，无突破性进展，对软件危机的缓解作用也不大。

Martin-Löf指出：一阶谓词演算不适合于计算机，而适合于计算机的数学是构造数学。他提出了著名的Marin-Löf类型理论。目前，比较流行的类型理论有如下两类：逻辑公式派和代数派。逻辑公式派的典型工作有Gordon和Miliner的ML系统，Liskov等人的CLU系统。Martin-Löf的类型理论是逻辑公式派的优秀代表。代数派又分为类别代数理论派和建立在λ-演算和Curry的组合逻辑基础上的类型理论派。前者的典型工作有OBJ2和ALPHARD系统，后者有Constable等人的PL/CV3，Mitchell与Plotkin和SOL系统，Cardeli与Wegners和FUN系统等。

有许多用于分析对象的规范说明技术，如America的表示变换函数的前置/后置条件规范说明[17]，Leaven的述模拟模型，Meyer的Eiffel语言中pre/post条件和类的不变性等[18]。

尽管形式化方法能为软件和硬件系统带来较高的性能价格比，但在实际的系统开发中并没有得到广泛应用。本文简要介绍大型软件开发过程中使用的形式化方法，并讨论利用形式化方法的途径。

二、软件开发过程中的形式化方法

形式化方法的基本含义是借助数学的方法来研究计算机科学中的有关问题。数学史和计算机科学对“形式化方法”的观点已有相当大的变化。人们认为以前的计算机科学离数学太近，试图在实现不现实的完全形式化的目标。另外，与数学家的形式化不完全一样，计算机科学家利用形式化技术来描述具体的问题，如形式化语言、形式化语义、形式化推理规则、形式化证明等；而数学家则通常使用一种严格的，非常不便于交流的形式化手段来进行高抽象层次的推理和证明。

已有的形式化方法主要是形式化的规范说明语言，用以在用某种程序设计语言实现之前先准备一个形式化的规范说明，这种观点已被广泛接受。规范说明指明程序的内容是什么，而不考虑如何实现。衡量规范说明语言的标准是其表达能力、术语的合适性、支持形式处理的能力及准确性等[10]。规范说明语言有许多，例如Anna，Estelle，Gist，Gypsy，Larch，LOTOS，OBJ，VDM和Z等。每个语言的侧

朱 冰 博士生。 梅 宏 博士后、杨芙清 教授，学部委员。

重点都不同，均针对所产生的背景而设计其特性。语言涉及的范围也很广，从通用的到专用的，从具有较高表达能力的（如集合理论）到有某些限制的（如等值逻辑），从多态的到不可变类型的，有些语言进行语法分析、语义分析并提供运行机制，而另一些主要起规范说明的作用。一个可执行的规范可作为将要开发的应用系统的一个原型。

需求阶段又分为信息收集和信息分析两阶段，前一阶段要求规范语言易修改，可维护；后一阶段要求规范语言精确，无二义并能进行完整性、正确性和一致性检查和证明。如何填平非形式化需求说明和形式化规范说明语言之间的鸿沟[23]也是研究形式化时需要解决的问题，其途径有以下三个：

一是将非形式化处理部分与形式化处理部分联系起来，如将SA结构化分析和VDM维也纳开发方法相结合[28]。

二是采用规范说明语言系列的方法。此系列由非形式化、半形式化和形式化规范说明语言组成。文[25]给出一个规范说明语言系列，该系列基于模块概念，支持软件开发各阶段的一致的规范说明。其非形式化的自然描述给出结构化正文描述信息，用以说明模块的框架；其半形式化的过程规范说明语言用BNF描述其语法，缺乏形式化的语义定义，可以用伪码形式对模块进行描述；其形式化的代数规范语言基于数据类型和参量化数据类型的代数规范，具有形式化语法和语义定义。这样，使模块系统的形式化规范说明确实可行。

三是开发规范说明语言的支撑环境。一个支撑环境应满足以下要求[11]：1）由于规范说明的探索性特点，环境应具有高度可交互性和可适应性。2）环境能允许用户以累积形式提交规范，能提供从非形式化到形式化的平滑过渡。3）规范语言应是图形化的，并且允许使用应用领域中广泛采用的符号表示。4）环境应能自动处理规范说明，即规范说明可核查、可执行，如ECASET工程[14]已开发出一个工具，提供图形编辑和解释功能。设计者通过图形编辑器布置一个VLP规范说明；再通过VLP规范说明的运行来直接从规范说明切换到原型。

软件的文档也非常重要，以形式化方式说明系统文档时，要考虑环境状态变量和函数关系，并采用表格和模式形式来表示[9]。

形式化方法贯穿软件开发的整个生命周期，目前，除继续软件开发前期的形式化方法的研究外，还开始了软件开发后期，如软件测试阶段的形式化研究。

三、典型规范说明语言

下面，给出几个形式化规范说明语言的简要说明，并围绕规范说明语言，介绍形式化方法的使用情况。

1. Z语言

Z形式化的规范说明语言基于模式演算，提供了丰富的操作运算，但不可执行。Z有模式匹配的符号，使用了具有强大功能的操作符，还用了一些非形式化的英语解释，给出的规范说明比较短，很容易阅读和理解。Z通过规范说明的一些特性的证明来检测错误。

文[6]中，考察了形式化方法在12个工业应用软件的开发过程中的使用情况，发觉用到的形式化方法有：Hoare的逻辑方法，Z规范说明语言，B工具，GYPSY记号，HP-SL等。12个项目中有4个使用了Z规范说明语言。文[2]中，用Z语言给出一个书店管理信息系统的规范说明描述。文[3]中，使用Z规范描述语言设计一个对话系统。这说明Z语言适合于图形界面的设计。文[4]中，以一个行编辑程序作为例子，对Z语言和一个可执行的规范说明语言me too进行了比较。

2. Larch语言

Larch语言基于标准的一阶谓词逻辑，可用于说明程序序列的功能，特别是说明抽象数据类型的特性。一个Larch规范说明包括接口部分和核心部分；前者描述了数据的操作的效果，而后者描述了独立于计算模型的固有特性，因此基于状态的特性和独立于状态的特性被清楚地区分开来。此外，可以根据需要对Larch进行扩充，如增加描述处理的能力。

Larch将规范描述分成两部分，因而需考虑被描述对象的哪部分描述放在接口部分，哪部分描述放在核心部分，但并没有一致的限制，只要规范描述可读性好就行。一个接口描述必须说明对象操作的前置条件和后置条件，而核心描述要尽可能抽象和通用，不必考虑特定的接口。CMU的Avalon项目涉及支持容错的分布式系统的应用程序设计，其中Avalon/C^{++}语言是对C^{++}语言的扩充，Avalon/C^{++}的三种基本类：可覆盖的类，原子类，子原子类可由Larch来描述[1]，其描述都分为接口部分和核心部分。

3. OBJ语言

OBJ代数规范说明语言提供了以独立于实现的

方式描述数据结构的基础，是一种宽域语言，刻画公理可描述所有抽象级上的性质；规范说明级和可执行级等。OBJ 由对象网层组成，每个对象说明一个 ADT（抽象数据类型）。基于 OBJ 的研究已有许多。如 CLEAR，OBJ2，ACT1 等代数规范语言在 OBJ 基础上，引入了层次构造，参数化，模块封装，移入移出接口，重命名和模块组合等设施。OBJEX 是一个工具，完成语法和类型检查，通过将方程作为重写规则，来执行规范说明。OBJ 的可执行性使得可按源代码测试方式来验证规范说明。

4. LOTOS 语言

LOTOS（时序规范说明语言）是至今最精确的定义语言，其静态语义基于属性文法，动态语义基于代数。LOTOS 是可执行的，也是可证明的。ISO 开发了标准数据通讯协议和服务 OSI，但要使 OSI 真正成为标准，形式描述技术是必不可少的。LOTOS 正是作为这样的形式描述技术的标准而产生的。

LOTOS[20]语言有两部分，一是主要基于 CCS（通讯演算系统）和 CSP（通讯顺序进程）的进程代数，用于表示系统的时序行为；一是基于代数语言 ACT1 的抽象代数部分，根据数据的类型或类别以及构造和操纵数据的操作来描述数据。

LOTOS 的一个进程的行为由行为表达式来描述。进程通过共享事件来相互作用。

LOTOS 不提供内部的数据类型，必须在语言中定义所有的数据和操作。但 LOTOS 提供类型库，ISO 为 LOTOS 定义了标准基本类型集，如自然数和布尔量。

LOTOS 已成为 OSI 的形式描述技术的标准，不仅对 OSI 适用，也适用于许多分布式系统。如文[20]中，利用 LOTOS 语言对一并行例子进行形式化描述，具体采用了基于事件的风格和基于对象的风格。然后将形式化描述作为需求分析的结果和设计的基础，将形式化描述手工转换成对应的程序语言表示形式。

5. VDM 语言

VDM 语言有一套数学符号系统和基于形式谓词逻辑和集合理论的证明规则。广泛地借用了六十年代在欧洲和美国发展的形式的和基于数学的技术。由 IBM 公司在维也纳于七十年代初开发出其早期形式。

VDM 方法是一种基于形式语义的开发软件的方法，规定了大型软件开发的三个阶段：提出要求、形式定义、开发步骤。它也是自顶向下逐步求精的开发方法，这与常用软件开发方法类似，覆盖软件开发的所有阶段，从非形式的功能需求开始，到各抽象层次的形式规范说明，最后产生代码开发，还可进行正确性证明。VDM 主要特点是中间的形式定义阶段，体现了用形式语义来刻画语言功能的思想。目前，VDM 已广泛用于程序设计语言、数据库、操作系统、办公自动化系统和其它应用领域中。

6. Trace 迹语言

由于不能将一个复杂系统作为一个单一单元来处理，因而要将复杂的软件系统模块化，这就要严格定义模块间的接口。为抽象地说明每个模块的接口需求，将模块看作黑箱，用基于有限状态自动机的模块互连规范方法[12]的迹断言方法来说明接口需求。Trace 迹规范说明语言中，将软件模块实现的抽象数据对象作为有限状态自动机。对一个确定的对象，其外部可见行为是由 I/O 历史（即对象的迹）完全决定的。

为支持迹规范方法，开发了一个迹规范模拟工具，以有助于模块设计者校验其设计和用户理解模块的行为及模块测试者测试实现的结果。

代数规范语言中，定义抽象数据对象的方程式集常形成一个等式系统，这样的系统能用项重写来模拟。例如，Affirm，Larch 和 OBJ 都是由项重写来实现的。与代数规范相似，将一个迹断言规则看作迹重写规则，将规范作为迹重写系统，这样，就能形式化地得到相应的迹重写机制。

7. GLIDE 规范说明语言

图形用户界面的使用已成为流行，这要求利用图形术语来编写程序。图形设计系统的变化很广，但其操作主要基于图形结构的操作，因此可以考虑用形式化的方法来描述这种基于图形的程序设计环境。与图形用户界面库，图形显示、动画和操纵库不同，用规范说明语言来说明基于图形的语言和其程序设计环境，其抽象级别高些，并且编译时可能用到前者。规范说明语言 GLIDE[11]由三个部分组成：说明一个基于图形的程序设计环境的数据结构的产品集；一个简单查询；描述转化和转化不变性的谓词集。以前正文程序设计语言的形式规范中所使用的术语要经修改和扩充，才能适应对图形程序设计语言的描述。

8. SXL 语言

通过考察整个软件生命周期，得知尤其必要的是分析和校验软件设计的最原始想法并对简单行为进行描述。SXL 语言[23]是一种说明性语言，可执行，

能交互调试。它基于实体关系模型和量化的一阶逻辑，是在状态转换框架基础上定义的；由引用特定的模型状态的逻辑表达式来描述所有模型化的简单行为。在SXL中，原语谓词是最简单的逻辑表达式，引用实体或关系的单个实例。逻辑操作符和定义将原语谓词组合成复杂表达式，用于行为规则以说明行为的变化。行为除由用户的点火事件决定外，还取决于约束，用户定义的约束包括绝对约束和动态约束。在语言执行过程中，对事件规则的点火会引起模型状态的改变。

使用SXL语言的步骤是：从实际问题中导出非形式描述的E-R框架；根据E-R框架，给出SXL语言的对象和事实；将E-R框架中的转换用某种SXL事件的组合来表示，其中可能还包括绝对或动态约束；最后执行SXL语言。使用SXL的经验告诉我们：可执行规范语言比不可执行规范语言优越。另外，使用规范语言有助于开发人员之间的交流和理解。

9. TLG语言

二级文法语言TLG[12]是针对软件系统的形式化规范说明和自动生成而提出的。TLG将自然语言嵌入严格的数学函数和逻辑程序设计范型中。表示形式上，TLG与Prolog很相似，但其表达能力更接近函数语言和数据流语言，允许并行运算。TLG不仅适应于许多不同的计算模型，而且其自然语言的特点在软件开发过程中极为有用。TLG规范说明语言仅说明做什么，而不是如何做，因而书写简单，易于论证是否与用户的需求一致。TLG程序自成文档，且有多层结构，遵循自顶向下设计的一般法则。

10. ADTS语言

ADTS[21]是一种代数数据类型规范说明语言，支持可构造性、模块化、参量化和错误及例外处理。ADTS的框架包括声明、构造子、一般操作、构造公理和一般操作公理等。ADTS单元为参量化模块，由接口部分和体部分组成。现已开发了ADTS的可执行原型环境，在该环境中，以类似项重写的方式将ADTS转换成PASCAL。

抽象数据类型的代数技术的优点在于支持抽象，其基于严格的多类别代数的数学框架允许对规范进行形式化推理。抽象数据类型也是软件开发过程中极为自然的概念。一般代数规范有两部分：标记(signature)和公理；前者提供类型信息，后者给出操作的定义。

一个规范说明语言应支持大型软件系统的设计和维护。为此需将软件系统看成由小的可管理的规范说明单元组成，在这种情况下，ADTS是非常有用的。

四、采用形式化方法的若干实践活动

1. Cleanroom软件工程小组[13]

无故障软件是没有的，但可能存在故障概率为0的软件。这是IBM的Cleanroom软件工程小组的观点，该小组以较高的生产率开发出0故障概率极高的软件。这样的性能主要依赖于程序规范说明的数学基础、设计、正确性验证、软件质量控制和遵循所要求的工程规则。

传统的软件开发过程中，错误被看成是不可避免的，尽管各子部分都查了错，但软件整体仍隐藏了很深的接口和设计错误，只能通过调试或测试来消除。在Cleanroom小组中，通过形式化规范说明、设计和验证来保证软件的正确性。小组在单元测试和调试时进行正确性验证，再直接进入系统测试。从第一次运行时就开始计错，不允许私下调试。实践表明，Cleanroom开发的软件的系统测试常常为0故障概率。

Cleanroom软件规范说明中采用箱(box)结构的面向对象的技术。箱结构以黑箱，状态箱和清理箱形式定义需求系统，再经过推导，将对象联接成一个系统的体系结构。整个过程是逐步求精和逐步验证的。

2. MAPS半自动化程序构造系统[8]

MAPS系统由三个主要部分组成：伪自然语言规范说明分析器、细化子系统和段分析器。MAPS系统的模块库的结构是分层的，库中大约有50个基本模块和70个应用模块。这些模块是有关表操作、字符串处理、文件处理和语言处理的。细化子系统是由MAPS系统自身产生的，其输出的程序可能是C或是Lisp源码，用户可以选择。

MAPS使用一个伪自然语言或类CASE语言来描述用户的需求和模块的规范说明。给定一个规范说明时，MAPS系统检索库中与给定的规范相匹配的模块，用库中的模块替换该规范说明。

库模块程序构造系统有以下特性：

1)把用伪自然语言描述的规范说明翻译成用CASE语法写的规范说明。

2)使用了一阶和二阶逻辑，具有强大的匹配功能，能把库中的重用模块与目标程序联系起来。

3)当一个求解的问题用类 CASE 语言给出规范说明后,它能自动地推导出由库中模块组成的集合,此集合中的模块满足规范说明的有关输入和输出之间的约束条件。

3. 其它

Draco 系统采用一种用户可说明的领域语言。规范描述被转换成一种树结构的 Draco 内部形式,树的每一个结点都是领域的构件名。对可重用模块,Draco 系统是用算法语言 SIMAL 描述的。从程序合成的角度来讲,Draco 系统将领域规范说明转换成内部表示,再优化之,最后用替换法将内部树的结点由已实现的模块来替换。Draco 系统中,经过优化的 Draco 内部表示形式即为文档。

SAFE/TI 是用英语描述的,由用户的英语描述的规范说明得到 GIST 的规范说明,再经程序转换后得到最终程序;其文档由 GIST 段分析器从 GIST 规范说明中产生。对于可复用构件,SAFE/TI 以程序转换规则表示之。

MODEL 用该系统的非过程语言描述。其程序合成是在 MODEL 语言中利用数据流图而完成。

五、结束语

在系统开发模型中,系统分析和规范说明非常重要。最终的规范文档既是设计者和用户的合同的一部分,又是软件开发的起点。用于描述规范说明的语言范围很广:从非形式化到形式化,从实时应用到传统的信息系统。目前,开发小型软件应用系统时,用得最多的规范说明还是非形式化的,常常有表、图和其它图形记号的说明性文体[1]。开发大型软件系统时,大多采用形式化的规范说明语言来提高软件开发的正确性和效率。究其原因在于:形式化方法虽然有很多优点,但却非常复杂,程序设计人员一般不愿意学习。如果将形式化的表达方式更加简化,则会得到更大的发展和使用[5]。

目前,关于软件开发过程中的形式化方法有如下共识:

1)集中于软件生命周期的早期活动。在系统开发的早期采用形式化方法收效快,效果好。特别是用形式化方法来说明需求分析,有助于发现细小的错误和二义性,可以简化设计,方便与系统设计进行交流。

2)完善系统使用的术语。好的术语是使用形式化语言的关键。一个语言应基于熟悉的简单概念,具有功能强的但简单的组合形式。

3)减少开发费用。大多数有关形式化方法的工作着重于获取高质量的软件,这点对诸如航空电子、安全等领域的关键系统的研制极为重要。另外,由于许多商用公司目标在于市场,因此形式化方法的研究方向应朝提高产品生产率上发展。

4)对软件过程的影响。已有形式化方法的研究对软件过程没有给予足够的重视,新的软件过程模型会反映出形式化方法的使用。

5)证明非功能特性。已有的形式化分析的主要重点放在一个系统的功能特性证明上,对有关保密、安全和结构等非功能特性的研究很少。非功能特性很难用传统的测试技术来核查,但能用形式化方法有效地进行核查。具体而言,应辨别和指出可确定特性(如期望的临时序列是否满足过程集)和不可确定特性(如是否满足保密或安全性约束),并从不可确定特性导出确定的近似解。

6)将不同的分析方法组合起来。由于缺乏定理证明的机制,很难证明程序功能的正确性,困难主要在于一个证明过程中称为"引理"的某种假设,没有相当的人类智能常常很难证明。行之有效的是采有一种混合方法,将形式化证明与实用的测试、容错和运行时核查相结合。

数学已经过了二百多年的历史,而有关模型和计算机程序的谓词特性的逻辑学的研究仅仅才几十年,形式化方法的研究尚处于初期,更大的成功指望将来。

参考文献

[1] J. M. Wing, Using Larch to Specify Avalon/C++ Objects, IEEE T-SE, Vol. 16, No. 9, pp. 1076-1087, 1990, 9

[2] K. T. Narayana et al., The Formal Specification of a Small Bookshop Information System, IEEE T-SE Vol. 14, No. 2, pp. 1089-1103, 1988. 2

[3] D. Gray, Formal Specification of a Look Manager, IEEE T-SE Vol. 16, No. 9, 1990. 9

[4] Comments on Formal Specification of User Interface: A Comparison and Evaluation of Four Axiomatic Approaches, IEEE T-SE Vol. 14, No. 4, 1988. 4

[5] E. C. Hehner, Whats Wrong With Formal Pro-

gramming Methods? LNCS 497 Advances in Computing and Information —ICC'91,pp2—23

[6]S. Gerhart, Observation on Industrial Practice Using Formal Methods, Proc. of 15th Intl. Conf. on SE, IEEE, 1993, pp24—33

[7]From Specifications to Machine Code: Program Construction through Formal Reasoning, 6th Intl. Conf. on SE, 1982

[8]F. Nishida et al., Semi-automatic Program Construction From Specificaions Using Library Modules, IEEE T-SE, pp. 853-871, 1991. 9

[9]P. J. Courtoris et al., Documentation for Safety Critical Software, Proc. of 15th Intl. Conf. on SE, 1993 IEEE, pp. 315—323

[10]A. Borgida et al., '… And Nothing Else Changes': The Frame Problem in Procedure Specifications, Proc. of 15th Intl. Conf. on SE, 1993 IEEE, pp. 303—313

[11]M. F. Kleyn et al., A High Level Language for Specifying Graph Based Languages and their Programming Environments, Proc. of 15th Intl. Conf. on SE, 1993 IEEE, pp. 324-335

[12]Y. Wang et al., Simulating the Behavior of Software Modules by Trace Rewriting, Proc. of 15th Intl. Conf. on SE, 1993 IEEE, pp. 14—23

[13]R. C. Linger, Clearnroom Software Engineering for Zero-Defect Software, Proc. of 15th Intl. Conf. on SE, 1993 IEEE, pp. 2—13

[14]A. Fuggetta et al., Executable Specification with Data-flow Diagrams, Software Practice and Experience, Vol. 23 (6), pp. 629—653, 1991

[15]G. McCluskey et al., Template Instantiation for C++, ACM SIGPLAN Notices, Vol. 27, No. 27 pp. 47-56, 1992

[16]W. R. Cook, Interfaces and Specification for the Smalltalk-80 Collection Classes, OOPSLA'92, pp. 1—15

[17]G. T. Leaven et al., Reasoning about Object-oriented Programs that Use Subtypes, In Proc. of ACM conf. on OOPS-L-A, 1990, pp212—224

[18]B. Meyer, Lessons from the Design of the Eiffel Libraries, CACM33, 9'1991), pp. 68—84

[19]P. America, A Behavioral Approach to Subtyping Object-oriented Programming Languages, In Proc. of the REX School/Workshop on the Foundations of OOL, LNCS 489, Springer-Verlag, 1991

[20]D. W. Bustard, et al., An Exercise in Formalizing the Description of a Concurrent System, Software Practice and Experience, Vol. 22(12), pp. 1069—1098, 1992

[21]S. Lee et al., An Executable Language For Modeling Simple Behavior, IEEE Trans. on SE, Vol. 17, No. 6, pp. 527—543, 1991

[22]B. R. Bryant et al., Formal Specification of Software Systems Using Two-level Grammar, Proc. of 13th Intl. Conf. on SE, 1991 IEEE, pp. 155-161

[23]M. D. Fraser et al., Informal and Formal Requirements Specification Languages: Bridging the Gap, IEEE Trans. on SE, Vol. 17, No. 5, pp. 454—464, 1991

[24]L. Jadoul, et al., An Algebraic Data Type Specification Language and its Rapid Prototyping Environment, 1989 ACM, pp. 74-84

[25]H. Weber et al., Specification of Modular Systems. IEEE Trans. on SE, Vol. 12, No. 7, pp. 784—797, 1986

[26]I. A. Mason et al., Program Transformation for Configuring Components, 1991 ACM, pp. 297—308

[27]杨芙清，软件工程的研究和发展，1993. 11