计算机科学1990 №.5

关于编译程序测试的若干问题

李友仁 （西安交通大学）

摘 要

In this paper we have analysed and researched Objective, principles, design and implementation method for compiler testing. This will implement and improve the theory and implementation methods for compiler testing.

一、编译程序测试的目标和特点

对软件系统进行测试的主要方法是通过设计测试实例，执行程序和分析运行结果来发现软件系统的错误。编译程序测试的方法与上述方法相同，但编译程序测试的依据是语言标准文本，而不是通常的软件规格说明；测试实例是由一组源程序组成，而不是一组数据；每完成一个测试都需要执行编译-装入-执行(compile-load-go execute)过程。因此与一般软件系统测试不同，编译程序测试有自已特殊的要求和方法。

编译程序测试的目标是要验证编译程序是否遵守语言标准文本的规定。换句话说，它是要检查编译程序与语言标准文本规定的语法、语义和语用的符合程度。通常，编译程序的测试是为了确认编译程序的正确性，

PC/AT上实现了其中的结构编辑功能[9]。

六、结束语

Jackson方法中的JSD方法是一种很有潜力的软件开发方法，它在规格说明阶段并不关心硬件环境，只在最后实现步才确定目标机的环境，并施于各种变换以适应系统环境。随着硬件环境的发展，在并行多处理器系统环境下开发大中型的应用软件系统，JSD有它的优越性。对进程的调度规则作深入的研究，JSD的规格说明可在任意个数的处理器支持下运行。

主要参考文献

〔1〕M. A. Jackson, Principles of program design, Academic Press, 1975

〔2〕M. A. Jackson, System development, Prentice—Hall, 1983

〔3〕J. R. Cameron, JSP & JSD, The Jackson approach to software development, IEEE Computer Society, 1983

〔4〕M. A. Jackson, Ioformation Systems: Modelling, Sequencing and Transformations, Proceedings of the Third International Conference on Software Engineering, 1978

〔5〕苏建文、潘锦平，Jackson 设计方法介绍，计算机科学，1984.1

〔6〕胡绮峰，吴湛兵，陈友君，JSD——一种系统开发方法，计算机科学，1986.1

〔7〕仲萃豪、叶农，JSD方法对软件工程研究的启示，计算机科学，1988.6

〔8〕蔡建明，一种面向数据结构的软件设计技术，南京航空学院科技报告，1988.11

〔9〕蔡建明，Jackson方法及其开发环境的研究，南京航空学院研究生硕士学位论文，1989.4

〔10〕董士海，计算机软件工程环境和软件工具，科学出版社，1988

不涉及或较少涉及编译程序的性能和效率（例如，编译程序的长度、优化程度，目标程序的质量等）。一个好的编译程序的测试系统不仅是衡量编译程序的重要工具，而且也是广大用户选择编译程序的工具。

为了实现这个目标，对编译程序测试的设计应注意以下特殊问题。

1. 由于任何测试程序系统都不可能检查出编译程序的全部错误，所以应使用户易于了解哪些语言元素已被测试，并且能方便地对测试系统进行调整和扩充。

2. 运行测试程序时，可能发生以下情况：

1)在编译时出错，终止编译程序。输出若干诊断信息。

2)由于测试程序不满足装入条件，在联接和装入时出错，运行终止。输出若干信息。

3)由于运行环境或编译程序故障，在输出测试程序结果以前终止测试程序执行。

4)与期望的输出结果不相符，出于以下原因：语言标准文本中语义的二义性或其他缺陷，编译程序和测试程序对某些语言元素做了不同的解释；测试程序自身的错误；编译程序中的实际错误。

为了使用户容易辨别以上不同的情况，在设计测试程序时应注意到上述问题，提供必要支持。例如，在测试程序中增加注解等。

3. 由于语言标准文本的语义几乎都是用自然语言描述的，没有给出形式的或严格的定义。因此，容易被不同的人给出不同的理解和解释。

4. 为了对各种环境下的编译程序进行测试，测试程序应尽量减少I/O的种类与数量。

5. 多数编译程序对数值的精度、范围，用户源程序的长度、结构复杂性等都做了一定的限制。因此，对测试程序输入数据的设计要适中。

6. 编译程序的测试程序既可由一个可执行的源程序组成(它包括全部测试)，也可由大量小的可执行的源程序组成（每个小测试源程序只测试某一语言元素)。小的测试程序运行简便，易于发现错误。但增加了运行测试程序的总工作量。大测试程序的情况与此相反。因此，在设计测试程序时必须对测试程序的大小进行决择。此外，还需提供把小测试程序结合成更大的测试程序和把大测试程序分解成更小的测试程序的机制。

自七十年代以来，软件测试理论与技术获得巨大的进展，建立了软件测试的理论基础，各种技术与方法大量涌现。这使软件测试从过去的方法收集发展成为新的科学分支。其间有关编译程序测试也受到了人们的重视，编译程序测试技术日趋成熟[1]。像Fortran, Pascal, Cobol, Ada编译程序的测试系统或确认系统相继出现[2,3,4,7]。近几年来，国际上对编译程序的测试和确认工作逐步走向专业化。一些组织或公司受权专门从事编译程序的确认工作。例如，美国的National Institute of Standards and Technology(NIST)下属的SoftwareStandards Validation Group(SSVG)，它负责Cobol, Fortran, Ada和Pascal四种编译程序的验证。任何单位开发的这四种语言的编译程序，只要交付一定费用（从2千美元到1万美元不等）之后，都可申请由该组织或由该组织委托的有关公司进行确认。验证合格的编译程序将在该组织发行的 "Programming Lauguages CERTIFIED COMPILER LIST" 中公布[5,6]，这个LIST每季度发行一次。美国商业部规定，只有经过NIST认可，并在上述LIST中公布的编译程序才允许进入美国政府部门的计算机系统。对这四种语言以外的其他语言的编译程序测试的专业化工作尚不清楚，再考虑到各种新的程序设计语言不断涌现，这些都说明了对编译程序测试的研究仍然十分必要。此外，编译程序测试的理论与方法还对大量出现的人机接口语言、图形语

言；操作命令语言、汉字系统等的设计、编译和测试产生重要影响。

二、测试程序的设计方法

本文第一节提出了编译程序测试的若干特点，以下将针对这些特点仔细讨论其设计与实现的方法。

1. 设计原则

1)测试程序的设计不是针对编译程序的纠错，而只是关心编译程序与标准文本是否一致，因此不必提供更多的纠错信息。此外，测试程序的设计是以下面的假定为基础，即编译程序基本可正常工作，但并未达到与语言标准文本的完全一致。

2)确定语言标准文本，把它作为全部测试设计的依据。

3)对语言标准文本中没有明确规定的实现方法，计算结果及运行环境，应尽量减少对它们进行测试。例如，计算机的规模，输入/输出设备；编译程序的加工能力：如源程序的最大长度和复杂度；数值的精度与范围；提交程序和数据的形式与方法；编译和运行程序的过程等。

4)简化测试程序的使用。首先应尽量减少理解测试，分析测试结果和确定测试"通过"或"失败"的工作量。其次使测试程序的编译和运行所需机时达到极小。

5)只有通过测试的语言元素才能用于支持其他语言元素的测试。

6)保持测试程序的开放性，以便用户进行增、删、改。

2. 若干策略与技巧

1)因为一个测试程序通常只测试一个语言元素，或一条规定，或一个语句，或语句的一部分。所以，测试程序都比较短。每个测试程序都是可执行的，它所需要的输入数据是自备的。这些数据或作为常量嵌入程序的内部，或作为输入数据紧跟在测试程序自身的后面。

2)一般选择语言的核心子集作为支持测试的语言元素，核心子集以外的语言元素从不用于支持其他语言元素的测试。核心子集通常是指基本的I/O语句，基本的说明语句，赋值、条件、转向语句等。

3)测试顺序的设计问题，即按什么原则安排语言标准文本的语言元素的测试顺序。根据上面提到的设计原则，先应该对语言的核心子集进行测试。然后利用核心子集的语句编写测试程序，对其他语言元素从简单到复杂逐个进行测试。例如，对Fortran语言编译程序的测试正是按这种方式安排测试顺序的[2]。但近年来有一种趋势，认为编译程序要交付进行公开确认的前提是编译程序基本可以正常工作，也就是说，至少语言的核心子集已通过测试。因此，测试的顺序可依据语言标准文本的章节号顺序排列。例如，对Cobol编译程序的测试是按这种方式安排的[4]。由于美国标准(ANSI Document Reference X3.23—1985)规定Cobol 85由11个功能模块组成，Cobol编译程序的测试系统CCVS 85也对应成11个部分，每个部分的测试程序再按对应的标准文本的章节号顺序排列。对Pascal编译程序的测试[3]，先对测试程序按某些方法分类，然后每类测试程序再按对应标准文本的章节号次序排列。

4)为了提高测试程序的可阅读性，应对测试程序的名字，编号，以及程序中的符号名（如变量名，子程序名等）和标号进行约定。对测试程序除命名外，还给出编号是必要的。编号通常就是测试程序出现的顺序号。测试程序的名字通常是该程序功能的缩写。在[3]中还给出了一种方法，它把测试程序的名字与相应的标准文本的章节号一一对应起来。例如，与章节号为6·5·4·2—1对应的测试程序名字是T6p5p4p2D1，其中T表示测试程序，P表示"·"，D表示"—"。对符号名我们认为不宜规定太细。为了提高测试程序的可移植性，对输入/输出设备号不能在测试程序里分配，一般应引入整型变量表示设备号，由用户装入时再赋值。

5)对语言标准文本中未明确规定的语

义，不同的编译程序可能有不同的解释。为适应对各编译程序进行测试，设计测试程序时，应对以下几个问题给出明确说明。

i. 嵌套的深度：例如，嵌套深度可多达五层。

ii. 变元的个数：例如，测试程序包含的子程序可具有多达25变元。

iii. 数组的大小：例如，小于30个字。

iv. 括号层数：例如，表达式中的括号的层数不超过10层。

v. 分支数：例如，分支语句的分支数不超过12个。

vi. 常数的长度：常数保持不超过某个计算机存贮单元的长度。

3. 测试程序分类

对编译程序可能出错的性质、形式和种类进行分析和归类，再根据错误类型对测试程序分类。各类测试程序在程序设计方面都具有某些特点。测试程序的分类有利于对编译程序做更完全的测试和对测试结果的分析。通常，对标准文本中的每个语言元素（即标准文本中的段或节的内容）要编写多类测试程序，从多方位对语言元素进行测试，以便获得更完全的测试。以下我们给出一种较合理的分类方法[8]。

1）一致性测试 这类测试程序是正确的标准源程序。如果它们按所期望的要求执行，则输出"Pass—〈程序号〉"；出错时，则输出"Fail—〈程序号〉"；既不输出"Pass—〈程序号〉"又不输出"Fail—〈程序号〉"时，则程序在执行上述输出前就发生了错误。根据标准文本要求，对每个语言元素可直接编写一致性测试程序。若编译程序通过了全部一致性测试，则只能说明编译程序实现的解释或完成的功能与标准文本的定义不矛盾。

2）非一致性测试 这类测试程序违反标准文本的语义或语法规定，主要用于检查：编译程序是否偏离或合理限制语言元素的语义或结构；编译程序是否对某些语言元素做了扩充；编译程序是否包含一般性的语法错误。如果某个非一致性测试程序完成执行，则输出"Deviates—〈程序号〉"。这表示编译程序与标准文本不一致。这时测试者应分析是编译程序没有发现这种错误，还是在功能上做了扩充。如果没有输出"Deviates—〈程序号〉"，则表示测试程序没有通过。这时测试者应分析是编译程序发现了这种错误，还是由于其他原因造成的。一般，对应一个一致性测试可设计多个非一致性测试。我们应该适当地限制非正常条件的组合数，以减少非一致性测试的数量。

3）错误校正测试 如果在标准文本中对某个语言元素指定了可能发生的若干情况，每个情况的出现将引起一种错误，那么就引入本类测试。这类测试中的每个测试将对应一个这种错误。测试程序由不符合标准的源程序组成。当测试程序没有完成执行，这可能是编译程序发现了这种错误，也可能是由其他原因引起的。为了更准确地指出测试程序失败的原因，对每个测试程序编写一个预测试程序，它是校正了测试程序的错误的正确源程序。根据测试程序和预测试程序两者是否都完成执行，或者预测试程序完成执行，而测试程序失败来判断编译程序是否发现了测试程序中的错误。

4）能力测试 这类测试程序检验编译程序的能力／复杂性和输入／输出功能。例如，在过程或函数的说明部分允许类型、常数和标号的最多个数；数组最大的下界值；过程允许最多的嵌套层数；过程包含参数的最大个数；表达式允许的最大嵌套深度；复合语句允许的最多层数；循环语句嵌套最大深度等。全部能力测试程序都是符合标准文本的源程序。通常，编译程序不应拒绝这类测试程序，并输出相应的能力指标。如果测试程序设置的能力指标过高，则测试程序失败。

5）与实现有关的测试 在标准文本中对某些特性允许实现者在实现时有一定的自由度，或者在标准文本中没有明确规定，完

全由实现者决定。对这些特性的测试就构成这类测试。例如数值精度、表达式的计值顺序等。

4. 测试程序的格式与测试程序文档

1）程序的格式　为了阅读和使用方便，通常采用以下格式书写测试程序：

〈标题注释〉
〈说明性注释〉
〈版本注释〉
〈程序名〉—〈程序号〉
〈程序〉

每个测试程序都是从标题注释开始，它指明这个测试程序与之对应的标准文本的章节号、程序的类型和参考号。参考号表示标准文本中的一个具体定义或规定。例如，标题注释为[3]：（*TEST〈章节号〉，CLASS=〈程序类型〉，NUMBER=〈参考号〉*）。说明性注释解释测试程序的目的、特点或限制、可能出现的情况、预期结果和用法。版本注释表示程序所属的版本号，若本程序是对以前版本程序的修改或是新增加的，则在此处给予说明。在程序中可任意插入必要的注释，以利于阅读和修改。

2）程序文档　作为一个测试工具，程序文档应包括以下部分：

系统说明：本系统的目的、特点、使用方法。

索引：i）程序功能索引，该索引以程序名（或程序号）为序逐个说明各测试程序的功能和输入数据。ii）章节号——程序名索引，该索引以章节号为序说明每个章节对应的全部测试程序名。iii）程序名——章节号索引，该索引以程序名（或程序号）为序说明每个测试程序涉及的全部章节号。

测试程序文件：把全部测试程序按名组成一个文件，或按类型组成多个文件。

5. 测试结果报告

测试结果报告应反映测试实施的全面情况，其主要内容包括以下部分：

1）测试环境：机器型号、编译程序名、操作系统、语言标准文本。

2）测试条件：日期、测试者、测试程序版本、评审者、其他说明。

3）测试结果及说明：按测试程序分类、逐类说明通过的测试个数、没有通过的测试个数、原因分析。

4）测试结论与统计：按类统计测试通过与失败的百分比、统计全部测试通过与失败的百分比、对编译程序质量的评审意见。

三、测试程序的组织

为了对编译程序进行更完全的测试，需要设计大量的测试程序。通常测试程序的数量多达上千或数千个，总长度可达数万个语句行。若单独运行各测试程序，每次都要完成compile-Lood-go execute过程和运行结果分析等一套例行工作。这将降低测试效率，增加测试费用。反之，若把全部测试组成一个可执行的测试程序，则可避免上述缺点，但将会使测试结果分析和对测试程序的增、删、改发生很大的困难。因此，关于测试程序大小的决择问题一直受到人们的关注。也就是说，测试应由数量多的小测试程序组成，还是应由数量少的大测试程序组成是对编译程序测试的一个重要设计策略问题。对这个问题已有不少研究工作，我们从结合方式进行分类，讨论如下。

1. 紧密结合方式

所谓紧密结合方式就是把多个可执行的测试程序在结构上结合成为一个更大的可执行程序，或者把一个可执行的测试程序分解为多个可单独执行的小程序。为此，需要删除多余的或增加必要的结构语句（如begin，end）和移动某些语句（如说明语句）的位置。也就是说，这种结合方式要对被结合（或被分解）的程序进行修改。例如，对Fortran的测试程序设计[2]就采用这种方式。它在版本1中设计了116个可执行的Fortran测试程序，在版本3中结合成14个可执行的测试程序。

1）程序结构　大多数程序设计语言对

说明语句、语句括号（begin, end），程序结束语句stop, end等在程序中出现的位置都是固定的，并且作为一个完整的可执行语句它们是不可缺少的。若把若干小程序结合成一个大的可执行程序，那么就要把各小程序中的说明语句搬到适当的位置，并删除多余的 stop, end 语句和重复的说明等语句。同样，把一个大程序分解为多个小程序也存在类似的语句搬移和增加问题。解决这类问题的一个有效办法是在编写小测试程序时，把说明语句，begin, end, stop 等语句都标记为特殊的注解行（例如，对Fortran的这类注解行标记为"C="[2]）。若需要单独运行小测试程序时，则把注解行标记"C="取消。若将小测试程序结合成一个大的可执行程序时，则需把各小程序中带有标记"C="的说明语句移动到适当位置，删除最后一个小测试程序结束语句的标记"C="。同样，大程序分解为小程序时，只需对"C="注解行语句做相应的修改。

2）避免符号名的同名　为了消除合并后的重复说明，或分解后漏掉必要的说明，我们规定在程序中使用的全部符号名不允许同名，即不仅在一个程序内，而且在各程序间没有相同符号名。为此，建立符号名表，列出全部符号名供用户参考。

3）避免标号同名　避免标号同名的方法仍然是规定在各程序中出现的全部符号不允许同名。例如，利用程序号作为标号的高位符号串，其低位符号串表示标号的顺序。

4）把某些公用操作编成子程序，供其他程序引用，以提高测试程序的可读性和透明性。

5）以这种方式结合的测试程序结构紧凑，能减少测试工作量，但对测试程序的结合和分解非常不方便，代价高。

2．松散的結合方式

所谓松散的方式就是把每个测试编写成单独可执行的程序，各测试程序的输入数据是自备的。全部测试程序头尾相接，顺序排列构成一个文件。为了使用方便，也可按测试程序类型分成若干个文件，在文件内测试程序以对应的章节号的顺序为序排列。另外，建立一个索引，指明测试程序在文件中的相对位置，方便用户对测试程序的选择。这种结合方式的优点是结合与分解灵活，其缺点是测试程序的执行和测试结果分析的工作量大。

3．编辑方式

所谓编辑方式就是首先对每个测试编写一个可执行的测试程序，然后把测试程序按某种顺序（如对应的标准文本的章节号）排序构成一个测试文件。我们把一个服务程序（或称为执行程序）插入文件里，并放在全部测试程序的最前面。服务程序的功能是完成对测试程序的选取、结合、修改和在该文件中增加新的测试程序等。用户通过服务程序对测试程序进行编辑和控制。其实现方法如下：

1）调入服务程序　把测试文件装入系统后，首先调进服务程序，为测试程序的编译和执行做好准备。

2）编辑和编译服务程序　为适应不同的运行环境，首先对服务程序中涉及的输入／输出设备等运行条件进行修改。然后对服务程序进行编译生成可执行的目标代码。

3）准备服务程序的输入数据　这些输入数据包括选择命令和各种控制信息。选择命令用以表示用户希望选择哪类测试程序、输出什么信息。控制信息主要包括对测试程序的修改信息。

4）运行服务程序　运行服务程序时，用户定义的相应命令和控制信息被插到各测试程序里，完成对测试程序的修改，使它们变成在指定环境下可运行的程序。同时，服务程序把修改后的测试程序组成新的测试文件，并且输出用户指定的各类信息（如打印新测试文件等）。

5）运行测试程序　在执行服务程序后，即可开始对测试程序进行编译。　首先把测

软件测试技术的发展*

黄 明 李文健 许建潮

（吉林工学院计算机系）

摘 要

软件测试是提高软件可靠性的重要手段，也是软件工程中最活跃的研究领域之一。鉴于今后若干年内软件测试仍将是软件质量保证的主要手段，因而我们在本文中讨论了软件测试的发展历史、测试策略、测试工具、软件测试的现状及其发展趋势。

一、引 言

软件测试是保证软件可靠性的主要方法之一，其目的是发现软件错误，提高软件产品质量。大量统计资料表明，软件测试的工作量往往占软件开发总工作量的40%以上。对软件可靠性来说，尽管测试与程序正确性证明技术相比是一种不完善的技术，但目前程序正确性证明技术还远远没有达到实用阶段。即使有了正确性证明程序，软件测试也仍然需要，因为程序正确性证明只证明程序功能是正确的，并不能证明程序的动态特性是符合要求的，另外，正确性证明过程本身也可能发生错误。所以，到目前为止软件质量主要靠软件测试来保证。要想在短时间内避开成本昂贵的软件测试活动，看来很难做到。

本文首先介绍目前比较流行的软件测试技术，然后以我们正在开发的Prolog软件测试程序从新测试程序文件中取出，然后编译和运行。

由于用户可通过命令和控制信息对测试程序进行灵活的结合和修改，而花费的代价最小。因此，编辑方式优于前两种方式。

*吉林省青年科学基金资助项目

参考文献

[1] 编译程序的测试方法和实现，李友仁等，计算机技术，1985年，第5期。

[2] NBS FORTRAN Test programs, Volume 1—3, F.E.Holberton, E.G.Parker, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, 1974.

[3] The Validation System of Pascal, British Standard Institute, 1982.

[4] The 1985 COBOL Compiler Validation System (CCVS)——User Guide Version 1.6, The National Computing Centre Ltd., Manchester, U. K., January 1988.

[5] Programming Languages CERTIFIED COMPILER LIST, Edited by J. B. Kailey, U.S. DEPARTMENT OF COMMERCE National Institute of Standard and Technology, April, 1989.

[6] COMPILER TESTING PROCEDURE, U. S. DEPARTMENT OF COMMERCE National Institute of Standard and Technology, April, 1989.

[8] The ADA Compiler Validation Capability, J.B.Goodenough, 1980. ACM.