国外计算机图形系统文献

译 文 集

国家机械工业委员会武汉计算机外部设备研究所

一九八七年十月

目　录

软件的标准符合性测试

英国国家物理实验室
R.S Scowen

摘　要

凡是购买软件的用户都希望能够保证他所购买的软件正常运行。当他们感到不满意或者对程序设计人员不能完全相信时，则会自然而然地求助第三方对软件进行认证。到目前为止，人们只对编译程序采用形式化验证方法。本文讨论了一些理论性的问题，分析了编译程序现有的测试模式。文章最后简要介绍了另外几种质量保证方法，或许我们能够从中导出更好的方法来。

导　言

任何一个大而复杂的系统，都存在着认证系统是否符合设计要求的问题。工业用系统或商用系统，无论是在最初设计阶段还是后来投入使用，往往是越来越主要地依靠计算机。因而，计算机及其程序是否正确、可靠，就显得十分重要。

计算机硬件得到了很大的改进，它们的体积变小了，价格降低了，功能增强了，可靠性也提高了。而计算机的软件却并非如此。程序越复杂，也就变得越庞大，而且这些程序还在不断地修改和扩充，往往使得谁也弄不明白其意思。那么，我们怎样才能判断一个软件是否正确呢？软件的测试工作是由某个独立的机构来完成还是由软件的生产厂家承担？说明中是否需要把将要进行的测试包括进去？生产厂家或测试人员是否要作出任何形式的保证？所有版本是否都要作相同程度的测试？由谁来支付论证费用？是由顾客直接支付？由软件生产厂家支付？还是由社会支付（“免费”服务）？统系的测试范围是否包括安全可靠（是否会忽然发生故障？）和性能（是否运行良好？）这两个方面？

软件设计还是一项需要花费大量劳力的工作，与其他计算成本相反，价格在不断上涨。因此，程序的反复使用或将其移植到具有不同结构的计算机上运行，越来越受到人们的重视。事实上，用一种程序员很熟悉的高级语言编制并在新机器上运行的程序移植起来是比较容易的。许多程序设计语言的标准版本已经有了规定，所以，很多用户需要的是符合这些标准的编译程序。如能直接解释有效程序的编译程序。有经验的用户还喜欢那些把非标准程序当作错误程序处理的编译程序，因为这些程序既得不到前后一致的结果也不能进行移植。

由此可见，编译程序对绝大多数软件来说是最基本的程序。如果编译程序不正确，即使软件已经过证明是正确，但运行时也会出错。许多其他软件与编译程序同样重要，但目前还

没有人打算将其交付第三方测试。因此，本文只讨论验证编译程序是否符合标准的测试，至于以下提出的几种概念中哪些适用于软件的验证，例如某一标准图形软件包的验证，则应由读者自己来判断。

令人遗憾的是还存在着许多问题。

问　题

本小节将较详细地分析编译程序验证中存在的几个问题以及如何才能将这些问题化小。

黑盒子测试

目前所有编译程序的测试程序都是把被测试的编译程序当作“黑盒子”来处理的，即只观察各种不同测试实例的输出。我们大家都清楚地知道，这种方法能够检测出错误来，但不足以用来证明无错。例如，可能存在着某些异常实例，采用随机测试的办法是根本不可能检测出这些异常实例的。即使一个编译程序的每一部分本身是合乎要求的，但也有可能会出现错误，这是因为其中任意两个部分会意外地发生相互作用。有些编译程序具有任选功能，如列表、交叉引用索引，运行时错误检测，提供不同字符集，或独立编译等，这些编译程序也可能出现类似的问题。在其他未经测试的实例可能出错时，测试编译程序时使用的任选功能有可能是合乎要求的。但是，这样的组合实在太多，根本不可能对其作彻底的测试。

测试人员必须把编译程序当作“黑盒子”看待，因为编译程序的原始文件内容在商业上是不对外公开的，这几乎是没有例外的常规。即使生产厂家提供原始文件，以目前现有的技术也不可能对其性能特点进行分析。

实际上，测试程序可能要比理论上的推断更为有用。测试程序能够以过去取得的经验作为基础，例如，利用有关编译程序构造方法的知识。另外，测试程序还可以不断地进行扩充。虽然我们知道我们并非能够检测出所有的错误，但我们至少能够保证不会再次出现相同的错误。

反复测试的必要性

发表编译程序和其他软件的新版本是软件厂家的共同习惯做法。但所有的程序设计人员都会从他们的实际经验知道改变一个程序可能会在看来是无关的地方出错。所以，有关该软件老版本的知识就变得几乎没有什么价值了。因此，如果没有对编译程序进行反复测试，用任何一种编译程序测试系统来进行论证，都会出现一连串的错误。随着编译程序新版本的发表，其试测程序相应作了修改，原来的测试结果就不再被认为是有效。这时，证实编译程序是否仍能顺利运行的唯一办法是坚持由独立的第三方重新对其作彻底的定期测试，或者委托编译程序编制者自行测试。

语言扩展

销售市场的压力迫使计算机生产厂家对编译程序增加附加的功能，以期提高其效率。增加编译程序的附加功能方便了编程人员，即给程序的可移植性添设了障碍，这是因为多数用户不能将这种语言的扩展与标准语言区分开来。即使编译程序经验证明是正确的，当多数用

户了解到他们程序中的许多部分因不合乎标准而不能保证是正确的，他们会感到十分吃惊。

测试程序包显然是不可能对被实现语言的不合规定的扩展形式进行测试。独立的验证机构因无法知道程序员手册是否完整甚至连扩展表都提供不了。有些扩展形式是隐蔽的，不对外公开，只有程序设计人员才知道。有些扩展只是偶然性的，没有任何一个人能够知道，只有等着靠偶然机会发现。

因此，在程序设计人员违背标准语言时编译程序能够提醒他，这点显得十分重要。但说起来容易做起来却难。有些扩展形式在编译过程中很容易被检查出来，如FORTRAN77中增加一个RECORD型或LOOP语句。而有些扩展形式只有在程序运行时才能被检测出来。例如，如果FORTRAN77经过扩展后使得：

REAL AA(1:0)

被当作一个空数组处理，这时，只有当程序企图访问该数组中的某一元素时才返回出错信息。对于这样的扩展，编译程序必须在运行时检查可调节的数组说明（即子例程中具有确定上下限的变量的数组），而且必要时还要打印出相应的提示报文。因为数组说明有可能被多次执行，所以要一项基本的改进，使得程序每运行一次至多打印出一次这类提示报文，因为多数用户不愿支付这类检测的费用。

令人遗憾的是有些扩展几乎是不可能检测出来的。设FORTRAN77语句：

XX=ZZ+FF(XX)

句中FF是一个函数，其作用是给ZZ赋值。该语句的语义不用确，所以不符合标准，这是因为FORTRAN77与其他大多数语言一样未规定对两个初始项赋值的先后顺序。这类错误在解释程序时或程序运行时不易被检测出来，而必须要作一次数据流向分析。在更为复杂的情况下，例如 XX=ZZ(II)+FF(XX)

句中ZZ是一个数组，II=JJ，FF为一函数，其作用是对ZZ(JJ)赋值。这时，作静态分析无法检查出错误，而只能给出程序可能不正确的提示。

编译程序验证模式本身也可以经过扩展，加入用以验证普通扩展形式的附加任选测试程序。这些程序将用来检查扩展实现时是否前后一致，他们与被扩展语言及该语言的其他扩展形式的标准特性是否完全吻合。在对编译程序重新测试时，若这些附加的测试程序获得正确的执行，就能够确认扩展没有经过修改。

现行标准中规定不明确之处。

现行程序设计语言标准是用形式英语编写的，有时语义不够准确，也不够完整，往往也并非是释义清楚的样板。因为标准委员会对于某项定义只能采用这种方式，所以往往出现一些难以解决的问题。定义过于简要会使得一些软件生产厂家的编译程序不合乎标准，对这些编译程序和相应的软件程序进行修改，使之符合标准，开销相当大。

更经常出现的情况是程序设计语言标准往往是由计算机专家们制订的，而他们以前未曾受过标准起草工作方面的专门训练，所以常常会出现释义不清楚明确的现象，导致定义不明确不完全。而人们却会轻易地作出这样的假设，即标准中所有地方规定得非常明确。例如，现行语言标准对语言的基本部分的规定往往是十分含糊的，有时还几乎未作什么说明。因此，FORTRAN77中有关实数运算（常常是但并非必须是浮点运算）的唯一说明仅仅只是以

下简要的注释：

· “解一个算术表达式得出一个数值”（P6.1，11—22行）；

· “整数值的标准表示总是一个整型数据”（P13，3—4行）；

· “整型数据是一个实数值的处理机近似值”（P13,14—15行）；

· “本标准对以下几点并未作出规定……（6）数值量的范围和精度以及运算结果的舍入方法”（P11,33行，53—54行）。

因为对乘法运算结果必须正确这点作出要求，即使连近似于准确值的要求都没有，所以，由于运算结果不够准确，这就似乎不可能会引起编译程序出错。目前，计算机运算精度还没有统一的测定方法（虽然ada语言对浮点运算所需求的精度作了规定），所以这方面的问题显得并不重要。

除非标准经修改后变得更加完整，更加精确，否则，在编译程序验证工作再多花精力则纯属一种浪费。

编译程序的质量

编译程序的许多功能是十分重要，而还未经大多数验证程序的测试。有些功能是针对具体机器的，所以无法测试。特别是：

· 编译程序应该与操纵系统配合较好，以便使源程序、数据和运行结果都能进行编辑，作文件编排或被打印出来。

· 如果某项编辑、编辑程序或运行周期的周转时间的度量单位是小时而不是秒，则程序开发就要困难得多。即使编译程序的速度对一种特定型号的机器是很适合的，但有时也会因主存或辅助存贮器容量太小而降至原来速度的百分之一。

· 程序在计算方面的成本不宜过高，这点显得十分重要。但计算方面的成本不仅包括程序的编译和执行，而且还包括程序的编制、存贮、装入和修改。编译程序的某些其他特性因人们主观方面的原因而无法测试，如使用是否方便，是否好用等。

· 在程序中存在语法、语义或运行时间等方面错误时编译程序应能给出简单清楚的出错信息。出错信息应只针对编程人员所用语言编制的程序，而不应具体指出机器地址或指令。在程序表现“奇异”时，即其具有某些虽然符合标准但可能表明有错误存在的特点时，编译程序还应给程序员以提示。

· 编译程序必须十分耐用，无论是编译程序还是被编译的程序都不应失去控制，运行结束时只给出一个十六进制的转贮地址。

· 编译程序应能与其他语言的编译程序采用相同的使用方式，即通过一种简单自然的作业控制语言进行运用。

· 即使编译程序在以上几个方面都十分完善，若被编译的程序太庞大而不能编译或运行，将会没有多大的价值。

· 若存在良好的过程库，会使普通的任务简单化，在这些过程虽用另一种语言编写但也能运行的情况下尤其如此。

编译程序测试

如果测试程序公开提供，显然，软件生产厂家为了通过验证测试将会对编译程序作相应的小修改，很少或根本不去发现并克服其他的错误。测试程序的设计者一向强调说测试程序是以编译程序能顺利运行几乎没有什么错误为前提的，所以将测试程序公开是不适宜的。而编译程序设计人员则自然希望公开测试程序，了解编译程序的哪些性能合乎测试要求以及语言标准定义中不明确地方所要作出的解释。若要满足编译程序设计人员的这一要求，则必须公开测试程序的细节及编译程序通过测试所需要的结果。测试程序在众多不同场合下使用，在众多不同机器上运行，要想保守测试程序的秘密无论如何是做不到的。使某些测试实现参数化，能部分保守住测试程序的秘密。但是，完全依靠随机测试不可能在适当的场合下证明经过修改的编译程序给出的结果不变。

因此，测试程序必须尽可能地做到测试范围广泛，程序设计人员也必须要认识到“某特定编译程序业已通过验证测试”倒底有多大的意义。测试程序要达到使用范围广泛这一要求，必然会增加编写测试程序和实施测试工作的费用。

优化编译程序

优化编译程序总是要力图找出程序中能够用较短或较快的编码进行编译的那些部分。编译过程愈长，所得结果愈好。但令人遗憾的是这样的编译程序会带来一些额外的问题。

有些测试程序太简单，使得编译程序在优化时将正要测试的功能给略掉了。例如，FORTPAN77的优化编译程序在编译时可能会把

```
PROGRAM INT GT
II=3
JJ=2
IF((II.GT.JJ).AND.(NOT.(II.GT.II)))
*THEN
  WRITE(b,*) "INT GT--OK"
ELSE
  WRITE(b,*) "INT GT--**FAIL***"
END IF
END
```

简化成

```
PROGRAM INT GT
    WRITE(b,*) "INT GT--OK"
END
```

从中可以看出“>”在编译得到正确优化，但看不出任何关于运行时性能的情况。这是一个比较严重的问题，因为测试程序的打印输出不能表示出所要求进行的测试是否成功。

另一方面，还有可能会出现这样的情况，即测试程序太复杂，编译程序不对其作任何优化。这时，优化功能实际上没有得到测试。

对于某些语言，如Ada或FORTRAN，对两个子例程作独立的编译能保证使得测试获得令人满意的结果。但遗憾的是这时也可能会出现更加难以解决的问题。可能会出现这样的情况，即对于一些较为简单的实例以及在编译程序能判断出程序不能优化这样一些很复杂的实例来说采用优化是非常理想的。但是，世界上的某些实例的编译会出现错误。这时除非对编译程序的代码或其输出进行检查，否则，能够发现这些错误就是十分幸运的了。这是不足为奇的，经验告诉我们优化编译程序的可靠性不高。正如一位好挖苦人的人说的那样"优化编译程序给出错误回答的速度还要快。"

编译程序的测试程序

目前业已生产出几种不同语言验证用的测试程序，本小节将对其作分析比较。Cobol 和FORTRAN77用的测试程序最有名气，是最初由美国海军研究出的测试程序发展而来的，是当前美国联邦软件测试中心（1982年前称为联邦编译程序测试中心）从事编译程序测试工作的基础。 九八三年英国标准学会开展了PASCAL测试及美国Ada编译程序验证机构（ACVO）开展Ada测试这两项新的测试业务。目前还在成立若干附属测试中心，如英国国防部的Coral和美国的Jovial。还有一些其他的测试程序能够作自测试，如Algol 60，Algol 68，Minimal BAISC和RTL/2。但据我所知目前还未发表PL/1的测试程序。

方法论

有些测试程序，如BASIC，Pearl和RTL/2的测试程序，对语言的每一特性先测试好才在以后的测试中使用这些经测试过的特性。乍听起来这似乎是一个很合理的想法，决不会因为编译程序对语言中未经测试部分出错而导致测试工作失败。如果我们只希望了解编译程序是否能通过所有测试，这倒是一个很好的办法。但是，情况往往并非如此。一般地说，测试应努力尽可能多地发现错误，但在因某个错误而导致以后的测试失败时很难更多地发现错误。

还有一些测试程序能对语言的每一特点进行测试，但测试工作不是按特定的顺序展开的。除非是对正在测试的实际概念，一般说来，这些程序每次测试中只利用语言的一些基本特点，所以每次测试工作量小而且较为简单。每部分相互作用发生的错误仍无法发现。因此，这些程序往往含有若干附加的复杂测试，以检查耐用性与通用性。对于编译程序的容量及机器有关的数据的测定，则需作进一步的测试。

一小部分测试程序是采用系统方法生成的，因而从某种意义上讲较完整，比如说能够保证至少在一次测试中显示出全部有效终止符号对。这些测试程序往往依赖于专门编制的将语言语法或标准作为数据读出的软件。例如，Ciechanowicz它经完全使得 Pascal 测试程序执行标准PAscal静态检查程序的每一语句。

Algol 68 的测试程序比较起来不太形式化，是一组由不同程序员提供的程序。这些程序有时令人感兴趣，有时发现以前的编译程序存在的某个错误。这套测试程序是自然形成而不是系统地设计出来的。所以，Grune对于"没有哪一种测试程序能够检测出所有错误"这一问题持一种实用主义态度。他说："我认为某个编译程序能很好地处理测试问题并能对普遍程序较好地进行编译，那么这就是一个非常成功的编译程序。若测试十分复杂，通过这样的测试将会发现绝大部分不正确的简洁表达法。而且，始终不断地运用那些简单的特点，可

以避免对编译程序作过多的调整以适应于测试。”

测试程序设计人员参与编写测试程序获得的好处是他们成为了判断各语言间细微差异及各语言细节的专家。因而，他们能发表程序设计员指导并能参与语言标准的制订。

测试程序的应用范围与规模

不同的测试程序在适用范围和规模上存在着很大的差异。生产新的测试程序，其规模基本上与用户支付的费用多少成正比。

较老的测试程序只是一些合乎语言标准的程序。测试程序设计人员坚持的方针是若标准没有关于错误或非标准程序处理结果的规定，则没有什么需要测试的。

近期的测试程序的适用范围较广，其中包含有检查编译程序是否能检测特定错误的测试；检查编译程序是有时会失效或循环呢还是给出的结果虽前后一致但不合标准？还包含有检查编译程序质量的测试。这些测试程序的用处也多一些。程序员不可避免地编制出不合乎标准的不正确的程序，所以需要有用的编译程序。

一套综合程序中可能包括有数百条测试程序，程序的长度从10,000行至200,000行。

扩展与子集

目前还没有哪种测试程序用来检查某一语言的任意扩展，但Pascal测试程序内有一些程序可以判断语言是否实现了常用的扩展，而且还能检查若有扩展，则是如何扩展的。美国的Cobol测试程序采用了子集。联邦标准Cobol定义了四级，以其构成一个ANSI标准嵌套子集序列。测试要求若采用了具体作出规定的那一级中未包括进去的语言特点则应列表标识出来。程序设计员还必须提供一个编译时间开关点，能通过它来监示编译程序支持的级。这一功能的验证办法是对程序编译四次，每次检查开关点的一个位置。

测试方法

随着测试程序的规模不断扩大，实际测试工作及测试结果检查的自动化已成为必不可少的了。因而，一个大型的测试程序具有一个控制器，由这个控制器把测试集中在一起并在必要时插入作业控制语句。为了便于判别每测试的起始点和终止点，几乎所有测试的格式均实现自动化。同时，为了能自动地生成认证报告，多数测试结果也是自动地生成的。

测试必须是能够重复进行的，以便在人们对编译程序是否已作修改产生异议并认为若已作修改则其不能再算作是已通过认证时，能够使测试程序再运行一次。

认　证

认证工作一般要花一周时间。首先，认证中心委托一些独立的咨询公司实施测试或作为测试工作的见证人。测证工作一般在硬件所在地进行。测试程序运行后，要进行讨论，确定测试中发现的与标准不一致问题是否属于编译程序或测试程序本身的错误。最后写出认证报告确认编译程序已经过测试，标明运行测试程序的计算机、归纳测试结果。报告格式是标准的，统一的，所以用户可对不同编译程序作比较。在这所有一切工作完成后，将确认该编译程序合格。认证中心有时要公布这些测试报告。而ACVO将只发表已获验证的Ada编译程序表。

绝大多数认识中心已认识到软硬件还在不断地变化，所以他们坚持要求对编译程序作定期测试，即在联邦软件测试中心每两年对Pascal进行一次测试，对Cobol和FORTRAN则每年进行一次测试。ACVO曾一度打算将编译程序的源代码与目标代码存档，以便在人们对某Ada编译程序在经过认识之后进行修改发生争议时备查：

收　费

多数大型测试程序（如FORTRAN，Cobol，Ada和Pascal的测试程序）是公费开发出来的在开发测试程序时通常以标准价格提供付本。这样有利于测试程序的推广，便于其他专家反馈信息、提出增加有关测试的建议以及提出其中不正确或引起争议的测试。若测试程序很完善，则可能会提高收费。这样虽可以提高收入，但可能阻碍业余人员及较费时间的咨询工作的开展。一个综合性的测试程序的编制一般需要五个人年以上，所以，几乎还没有哪个程序能从销售中收回成本。如果语言在不断地修改，如Ada在一九七九年至一九八三年间的情景，则要大大地增加综合性测试程序的研制费用。但最好是先研究出标准语言，两三年后再出现其测试程序。

认证工作的费用不等，最高约一万美元。大型的公共测试中心一般都力图做到自负盈亏，但并没有打算收入测试程序的研制费用。除非有影响的计算机大客户（如美国国防部）拒绝购进配带未经认识的软件的计算机，否则，软件的认证工作不会完善，也不会获得成功。

可供选择的策略

假若将来还是象以前那样对编译程序加以评价，即构造一套测试程序来测试编译程序，然后检查测试结果，了解编译程序的性能特点，那么就有可能采取若干其他的方法更便宜更方便地获得可靠的软件。本节分析某些可能性。

标准编译程序

有一个老想法值得重新考虑，即把编译程序看作是某种程序设计语言的定义。这一思想首先是由Garwick提出的。但当时人们普遍认为这是一切实际的，因为编译程序一般是用低级语言编写的，不易懂，而且投入使用的许多计算机的总体结构全然不同。但此后已研究并实现了诸如BCPL、Pascal和RTL/2的语言。这些语言的第一批编译程序是用这些语言本身编写的，几乎以后的所有编译程序都或多少是以第一批程序为基础的。“样板程序”将构成Pascal验证程序下一个版本的一部分。

这一方法除了能保证达到程序的可移植性并能便宜快速地实现语言外，还能保证在语言的定义上不发生歧义。标准编译程序比英语写成的标准更加紧凑，当然要求它不能更难懂。

说明语言

一个程序设计语言标准可以被看成是该语言编译程序的说明。近期开展的计算机系统说

明的规范化工作能够带来几个好处，例如，一个形式说明是相应程序正确性证明的基础。形式说明的其他优点可以编写若干处理程序以便检查说明是否：

· 完整：是否对所有可能性均已有规定。

· 一致：说明的各不同部分不能相互矛盾。

· 简洁：要求不得叙述一次以上。

· 准确：这样用户与程序设计人员间无误解。

· 不过于具体：这样作出的要求不致于对程序设计人员限制过死。

对于说明语言，已有一篇论文[17]叙述了其各种构想：

· 有限状态机定义每次输入产生的输出，但在有必要定义同步时将会变得很复杂。

· 刺激一反应序列采用一种对用户刺激所作出反应的伪Algol定义。

· Petri网是一种图形技术，极好地表达了同步要求。但是，由于它们的图形特点使得它们作为要求阶段的文件输出比作为输入说明更为有用些。

英语不宜作为说明语言，它非常容易产生一些不确切的解释。正如参考文献[21]所表明的那样，对于大型系统，英语要比用以实现该系统的代码更冗长。

程序的验证与形式化测试

有些研究人员已经采用欧几里得几何证明方法来证明程序的正确性。然而，结果并不令人满意（就编译程序的验证而言）。程序说明和文本都必须提供出来，而普通的编译程序均不能满足这两方面的要求。有些研究人员（如研究Chill和Ada的D.Bjorner，研究Ada的Belz）在研究工作形式定义（有关概念介绍请参看参考文献[22]），但形式说明一般是不提供的，而且编译程序的文件也是严格保密的。总之，任意程序的分析是一个难以解决的问题，例如，目前还没有哪一个程序能够阅读某任意程序的文本，并报告该程序是否运行完毕。这一领域的研究工作都是通过先构造一个完成某给定任务的程序，然后证明其能完成此项任务而获得成功的。正如Goodenough和Gerhart指出的那样[24]，有时“证明”本身就不正确。

程序测试理论的研究工作同样没有取得令人满意的结果：Weyuker和Ostrand指出[25]，对于一个程序输入定义域中的每一元素D，均有一个正确处理除D以外的每一元素的程序，但此程序对D的处理则不正确。这表明只有对每一可能的情况都作测试才能保证程序的正确性。因此，执行程序每一部分，甚至执行每一程序通道的测试实例无法保证正确性。

一种更切实际的做法是证明无错。在采用这一技术时，要先假设已出错，然后再生成找出此错误的测试数据。

其他的认证模式

系统出错会带来十分明显的危险的行业已经研究出了比“黑盒子”测试更加严格的测试程序。本节简要介绍航空工业，其测试要求是合法的，而且费用要远远高出编译程序测试程序。本节还分析了英国国防部采用的非直接鉴定法和实时计算技术。

适航性认证

无适航证书的民用飞机只允许作试验飞行。航空法令规定飞机在其设计、制造、工艺和材料以及试飞结果均必须符合民航管理局的要求方可颁发适航证书。法令中没有具体规定飞机是否适航的判断标准。但要想使适航证书得到国际上的承认，持证飞机能飞越其他国家，则民航管理局必须做到使飞机完全符合联合王国呈送国际民航组织的国家规则。因此，为了保证飞机制造厂家在飞机设计、制造和试验等方面建立起正确的程序，民航管理局与飞机界协商，研究出了适航测试并提出了一些其他的要求。

一旦飞机制造厂着手设计新型飞机，民航管理局的设计联络巡视员将对该局的调查作协调工作，将适用的标准提交给厂家，并不断地对厂家的生产工艺，质量管理和试验进行检查和监督。许多要求简单，是定量性的，但也有些需要作定性分析，对要求的一般化内容进行解释。

设计过程中，民航管理局的专家对厂家走访，与之讨论借以建立符合适航标准的方法，必要时亲自观查并复审飞行试验的结果。例如对机体要作全面的测证，在发出适航证书后还将作强度和疲劳试验，以便预测飞机投入使用后机体结构的性能。

飞机试验不仅要逐部分逐系统地进行，而且还要作整机试验。

在分给适航证书后，为了检查生产出的所有飞机是否与样机性能相同，还要作范围不很广泛的飞行试验。另外，在联合王国注册的所有飞机都要作定期试验，以便检查飞机的飞行性能是否有下降的迹象。

如果飞机的改装对机体，航空动力学及动力部分有着重大影响，民航管理局将要求对该机再作一次适航测试。如果联合王国飞机制造厂的飞机在后来发现有故障，不符合证书的基本要求，也应通告给民航管理局。

民航管理局对在国外注册的飞机无管理权，但航空法令规定凡是在联合王国注册的飞机发生的所有故障和事故都必须向民航管理局报告。民航局将对发生的故障和事故进行调查，保证采取必要的补救措施。

民航管理局对联合王国生产的飞机作出的验证对于国际民航组织成员国均适用。许多国家有它们自己的规定，或采用美国的规约。这时需要对联合王国的飞机作进一步的调查方能获该国登记的适航证书。

鉴　定

国防部不仅直接测试产品，还可检查某一公司的能力，看其是否能够生产出合乎要求的产品。这一过程叫做合格审查，在国防部通常称为对照国防标准05—21对厂家作出评价。评价工作涉及到软件生产与管理方法的若干方面，特别要保证：

· 有着良好的软件管理工作和一个负责保证软件质量的管理人员，他应具有独立处理问题的权力。

· 软件的所有部分（即设计、开发、测试和维护）按项目重要阶段作适当规划，以便了解软件项目的进展情况。

· 具体规定并增补文件编制，程序设计和测试标准。

· 对照设计定期检查项目进度，若一致同意修改方案要记录在案，保证达到性能，可靠性和可维护性的要求。

· 因保存有适当的记录，测试和检查效果好。

· 从外面购进的软件必须符合同样高的标准。

· 定期检查质量管理工作，发现问题及时修正。

· 采用有效程序，保证软件付本都经过检查并完全一致。

评价工作视公司产品情况由国防部派出研究与开发，项目管理及质量管理等局的专家实施。评价开始时要拟订初步计划、进行走访。通常要对公司的所有部分进行审查，但这要依公司的要求和国防部的规定而定。两个手册[26,27]对评价过程和评价内容作了规定。公司的一切（如生产方式，内部质量管理、财政结构、人员、设备及项目管理方法）均要受到检查。

审查结果是一份表格，报告各种数据、扼要描述发现的不足之处或提出审查合格的建议。若公司经审查认为是合乎要求的，则被列入国防部"已评价的厂家表"。重新审查工作通常2—3年进行一次。但实际上"一旦认为有必要"就会作重新审查。例如，若客户报告已出故障，或人员已作重大变化，还会作抽查。

审查工作费用自然要决定于公司的规模，大致为几个人周或几个人月。国防部目前正在与几家国有企业合作，组织联合评审小组。这将会降低总费用。

其他行业也在采用评审方式。例如NATLAS对实验室作能力评审，审查其是否能对规定的产品作特定类型的测试工作。NATLAS指定独立评审人员检查某个测试实验室的组织机构、专业能力、设备及文件编制等是否合乎所规定的要求。

实时计算

实时计算中，例如过程控制业中，正确可靠软件生产方面的问题要比编译程序方面的问题更难以解决。我们发现以下技术是有用的：

· 冗余能保证没有哪个组成部分成为关键部分，

· 简单的程序易于认证与检查，

· 不断地监视有助于防止错误扩散，

错误是不可避免的，但系统设计成具有容错能力，即某一部分出错不会造成整个系统出错。系统还要成为"失效保险"，例如，如铁路上的真空闸，信号销有错，会使闸处于"STOP"点与信号联接，有效地构成闭合线路，显示出"STOP"信号。

设计人员对"相近差错"作调查。虽然采用冗余法能保证某一孤立的错误不会造成严重损失，但出错频率应降至最小，因为同时出现两个错误可能会相互影响，带来不可预料的损害。

结 束 语

软件是否符合其说明的测试工作存在着许多实际问题。编译程序的验证说明了这些问题，表明了是如何解决的。其他行业的经验表明相同的问题，并提出了一些更好的但成本也

更高的解决办法。

参 考 文 献

1. L.J.Osterweil and L.D. Fosdick, DAVE—a validation error detection and documentation system for Fortran Programs. Software Practice and Experience 6, 473—486(1976).
2. American National Standards Institute Inc., American national standard Programming language Fortran, ANSI X3.9—1978, American National Standards Institute, 1430 Broadway, New York, Ny 10018, U.S.A.
3. R. S. Scowen, The diagnostic facilities in Algol and Fortran compilers. NpL Rep. NAC 81, National Physical Laboratory, Teddington, Middlesex, England(July 1977).
4. R.S.Scowen and Z.J.Ciechanowicz, Seven sorts of Programs. ACM SIGPLAN Notices 17, 74—79, (1982).
5. United States National Technical Information Service, AD A036 174—CCVS74 V3.0 User,s Guide. 5285 Port Royal Road, Springfield, VA 22151, U.S.A.
6. P. M. Hoyt, The Navy Fortran validation system, FCCTS/TR-77/18, May 1977, ADPE Selection Office, Department of the Navy, Washington, DC 2073 6 U.S.A.
7. United States National Technical Information Service, PB82-250395—1978 FO RTRAN ComPiler Validation System Version 2.0 User,s Guide, 5285 Port Royal Road, Springfield, VA 22151, U.S.A.
8. Z.J.Ciechanowicz and B. A. Wichmann, A readers, guide to Pascal compiler Validation reports, NPL Rep.DITC24/83(Way 1983).National Physical Laboratory, Teddington, Middlesex, England.
9. B.A.Wichmann and Z.J.Ciechanowicz (Eds), Pascal Compiler Validation, Wiley, Chichester (March 1983).
10. Softech (J.B.Goodenough and J.R.Kelly), Adacompiler validation capability Long range Plan.Softech Rep.1067—1.1(Feb.1980).Softech Inc,460 Totten Pond Road, Waltham, MA02154, U.S.A.
11. J.B.Goodenough, The Ada compiler validation capability : IEEE Comput.14, 57—64(1981).
12. R.S.Scowen and Z.J.Ciechanowicz Compiler validation—a snrvey.NPL Report CSU 8/81 (Dec.1980).(Later reprinted as Chap. 13. pp.90—141 of[9]),National Physical Laboratory, Teddington, Middlesex, England.
13. Z.J.Ciechanowicz and A.C.de Weever, The "completeness" of the Pascal test suite. Toappear in Software Practice and Experience.
14. J.Welsh and A. Hay, The Pascal Standard from an implementor's viewpoint,

chapter 7, pp. 46—58 of Ref.[9].

15. D.Grune, The revised MC Algol 68 test, Set, IW122/79, November 1979, Department of Computer Science, Mathematisdh Centrum, Amsterdam, The Netherlands.
16. J.V.Garwick, The definition of Programming languages by their compilers.In Formal Language Description Languages for Computer Programming (Edited by I.B.Steel),pp.139—147.North Holland, Amsterdam(1966).
17. A.M.Davis and T.G. Rauscher, Formal techniques and automatic Processing to ensure correctness in requimeents specifications.Proc.Conf.—Specifications of Reliable Software, pp. 15—35, IEEE Catalog No. 79 CH1401-9C (April 1979).
18. C.A.Petri, Kommunikation mit automaten Schriften des Rheinisch-Westfaliesc-hen Institutes fur Instrumentelle Mathematik an der Universitat Bonn, Heft 2, Bonn, W Germany, 1962.
19. J.L.Peterson, Petri nets.Compuing Surveys 9,223—252(1977).
20. W.Brauer(Ed), Net theory and applications. Lecture Notes in Computer Science 84. Springer Verlag, Berlin(1980).
21. C.Jones, A survey of Programming design and specification techniques. Proc. Conf.—Specifications of Reliable Software,pp. 91—103. IEEE Catalog No.79 CH1401-9C(April 1979).
22. D.Bjorner(Ed), Abstract software specifications(1979 Copenhagen Winter Sch-ool Proceedings). Lecture Notes in Computer Science 86.Springer Verlag,Berlin (1980).
23. C.Strachey, An impossible program.Comp.J.7,313(1965).
24. J.B.Goodenough and S.L.Gerhart,Toward a theory of test data selection.IEEE Trans. Software Engng,SE-1(2).156—173(1975).
25. E.J.Weyuker and T.J.Ostrand, Theories of Program testing and the applica-tion of revealing subdomains. IEEE Trans.Software Engng,SE-6(3), 236—216 (1980).
26. Ministry of Defence. Guide to contractor assessment, Book 1. The conduct of contractor assessment (1978). Director General of Quality Assurance MOD (PE).
27. Ministry of Defence, Guide to contractor assessment, Book 4, Computer so-ftware QA systems, (1978), Director General of Quality Assurance MOD (PE).
28. H.K.Seyfer, A bibliography on compiler assessment. AGM SIGPLAN Notices 18(3),14—21 (1983).

张莉莉 译　　王世运 校

GKS标准符合性验证述评

美国莱斯特大学计算机实验室
K.W. Brodlie

摘　要

大量GKS软件产品迅速问世，表明GKS作为一种计算机图形的国际标准，已经广泛地为人们所接受。然而，若要维护GKS标准的基本原则，必须要有一种可行的手段对GKS软件产品进行验证，证实其是否符合GKS标准。本文介绍了GKS验证的一般方法。较详尽地描述了GKS返回给应用程序的数据验证过程，对GKS输出的验证只作了一般性的讨论。

1. 导言

GKS被通过成为国际标准无疑是计算机图形学发展过程中的一个里程碑。过去，基本图形软件产品日繁杂，基本设计原理各异。这些严重地妨碍了应用软件的开发工作。为一种基本图形系统编制的应用程序，需要经过不断地组织和编辑才能与另一种图形系统一道运行。在实际中，一种图形软件包的用户事实上是不能使用为另一种程序包开发出来的软件的。GKS标准的问世，为开发可移植性图形应用软件开辟了前景。虽然在撰写本文时GKS仅仅是一种ISO标准草案，但它对图形学界带来的影响是显而易见的。美国的一些主要公司正在开发GKS软件产品，很多微型计算机一时之间伦促地配上“类似GKS”的软件包，硬件生产厂家宣称它们的“终端具有GKS工作站的能力。”GKS时代已经到来了。

然而，在大家为此感到高兴的时候，应该提醒一句，这么多GKS产品如此迅速地问世，它们是否属于GKS的真正实现吗？

目前，已经有一些人（甚至在标准组织内部也有人）提出严格的GKS符合性规则不应再那么严格，可以放宽一些，以便让那些相互竞争的公司在各自的GKS软件产品上增加若干“附加的”功能。现在，我们常常听到一些人用“类似于GKS的产品”这一句话来说明某种按GKS一般原理实现的但不严格符合GKS标准规则的产品。如果让这一趋势自行发展，只把GKS标准看成是一般性的指导原则，而不将它作为一种设计蓝图，那么，制订此标准可望获得的可移植性的一切优点将化为乌有。为了避免这一严重形势的出现，需要有一种有效的工具对GKS产品进行验证，亦即，采用某种方法来验证一种GKS产品是否符合标准。这是坚持本标准的目标，保护用户利益和维护GKS声誉的根本途径。

GKS验证的研究工作业已开始。这项工作主要在西德的达姆施塔特(Darmstadt)大学和英国的莱斯特(Leicester)大学进行。后者受到了英国科学与工程研究理事会（SERC）的资助。EEC还通过召开一系列讨论会的形式对此项工作给予支持。第一次讨论会于一九八一年

五月在Rixensart召开。会上讨论了软件的另一领域即编译程序正确性验证的经验，提出了若干图形软件验证的基本思想。此后，又围绕这些思想召开了一系列小型EEC讨论会。一九八二年十月召开了第二次讨论会，会上对GKS验证工作进行了梳理汇总。

本文提出了一种切实可行的GKS软件产的验证模式，着重讨论了总的验证方法。

2．说明与认证

GKS标准的规范说明是GKS实现和GKS产品认证的关键。GKS标准文件基本上是用自然语言写成的一种非形式化说明。尽管标准文件中力求避免使用意义不明确的词句，而且有些部分，如函数的参数，还是用半形式化符号来表示的，但在完成GKS实现任务时，需要经过人的脑力劳动，把GKS规范说明译成计算机程序，其间还必须作出某些判断，这样就有可能出现人为的错误。把标准的规范说明变换成软件程序似乎是没有办法实现自动化，因此也无法保证这一转换过程的正确性。这种情况下，常规的认识方法是“证明为伪”法（falsification），这是编译程序验证中采用的方法。这种方法是用一套完整的测试实例对已完成的GKS软件进行测试，力企从中找出错误。若未找出错误，虽然不能保证软件完全正确，但至少可以认为软件几乎是正确的。如果测试实例经过多年的开发，不断地完善，那么测试的彻底性可以达到相当高的水平。这种测试方法在编译程序的测试中，特别是在Cobol编译程序的测试中已经取为了相当大的成功。

未来的图形标准有可能采用形式说明的方式发表。这时就可以采用一种截然不同的测试方法，即验证法(Verification)。在采用这种方法时，标准的规范说明经过一系列的程序变换而成为一种具体的软件程序。如果每步变换是正确的，则最后完成的程序一定是正确的。显然这种测试方法要优越得多，能保证软件的正确性，这是“证明为伪”法所不能做到的。

验证法和证明为伪法在许多方面是相辅相成的，因为一种方法的优点恰好是另一种方法的不足之处。证明为伪法把软件的实现工作与测证工作清楚地分离出来。软件实现者自由地按自己的意愿开发软件，对于以后开展的软件测试工作具有很大的独立性。但他在实现软件过程中得不到任何帮助。而验证法则把软件的实现工作与测试工作结合在一起进行。软件实现者从一开始就参与测试过程，并完全受测试过程的指导。但是，对于那些独立地开发出来的软件，不能采用验证法。

GKS文件目前采用一种非形式说明的形式，这就决定了要采用证明为伪测试法，但其间可以运用验证法中的一些方法。从第4节的介绍可以知道形式说是证明为伪法中的一种有用工具。对验证法继续开展研究，对将来肯定是十分重要的。本文献集中另有文章详细介绍。

3．证明为伪法

证明为伪法的简单模型如图1所示。证明为伪测证过程与查出新陈代谢功能异常的生理检查颇相似。先用各种不同的方法对被测对象（此处是GKS软件程序）进行刺激，然后对被测对象每受到一次刺激时所作出的反应进行检查。刺激采取GKS函数调用的方式，有时还要辅以操作在输入设备上作出输入操作。反应作为输出显示在一台或多台图形设备上，即变换