编译程序自动测试中的知识表示*)

周继鹏　顾元祥　华庆一

（西北大学计算机科学系）

摘　要

软件测试是软件工程的重要研究课题之一。本文把人工智能（AI），知识工程和属性文法多种知识和技术应用于软件测试，主要对编译程序测试中的知识分类和知识表示问题进行讨论，它是开发知识型编译程序测试环境的基本出发点。

一　引言

在软件（尤其是大程序系统）开发过程中，保证软件质量的困难是众所周知的，保证软件质量的形式化技术（例如程序正确性证明）目前还未达到实用水平，因此，人们广泛使用的方法是软件测试（software testing），它包括构造有代表性的测试实例，使用测试数据执行程序，并将运行结果和预测结果进行比较。软件测试的目标就是尽可能多地发现程序中存在的错误。

与其它软件系统相比，保证编译程序的可靠性尤为重要。因为编译程序是保证其它采用源语言描述的软件能够正确执行的基础。

至今为止，用于编译程序测试的唯一可行的方法仍然是手工测试——就是手工地编译一组被认为有效的程序，根据这些程序校验编译程序的性能。目前，已经研制出有助于编译程序自动测试的方法，例如〔8〕，〔9〕〔11〕。这些方法涉及到运用句子生成器，采用系统的方法生成源程序，但是这些程序都是人为生成的一些固定类型，不可能帮助测试人员得出高度智能化的测试用例选择方案，这是由于过程式的测试系统有着一定的局限性，不可能开发出通用、实用的测试系统，因为，软件测试，尤其是编译程序测试是一种知识密集型的智能活动，除了本身的测试理论之外，它与测试人员知识的深度、广度、经验的丰富程度，以及对源语言和编译程序的理解程度都有着密切的关系。目前，虽有人将人工智能（AI）方法应用到编译测试研究，给出了测试模型，但是，这些测试系统中知识缺乏，主要原因在于：1.对于测试知识没有给出一个统一的表示方式；2.对于测试经验没有很好总结、整理并显式地给出，在系统中无法利用；3.原来过程式的测试实例生成方法不适用于知识系统。我们经过研究发现，人工智能中的几种知识表示方式很难满足编译程序测试的特殊要求。本文把人工智能、知识工程和软件测试技术相结合，对编译程序测试中的知识分类和知识表示问题进行研究。

二　编译程序测试的基础知识

人们通常讲话不一定要说出所有要说的东西，而是含有大量隐含的意思，但听众在上下文的基础上足以理解讲话的含意。追其原因，就是人类听讲者应用了它具有的基础知识（Background knowledge）去揭示大量隐含的东西。在编译程序测试中，计算机要象人类一样自动地生成测试实例，就需要存贮和应用大量的基础知识。这就提出了两个问题：1.编译程序测试中的基础知识都包括

*)此项目由国家自然科学基金资助

哪些；2.提取的知识（包括域知识和推理知识）怎样表示。这也是人工智能（AI）和知识工程研究的中心问题。经过对编译程序手工测试及有助于自动测试方法和系统的分析、研究，我们认为有关编译程序测试的知识大体可以分为以下四类，1.测试方法学知识；2.目标源语言的知识；3.具体编译程序的特征知识；4.有关测试的经验知识。

1.测试方法学知识

软件测试是软件工程中要研究的一个重要问题。软件测试的主要方法是根据软件开发各个阶段的说明和内部结构，精心设计一批测试实例（由输入数据和输入数据所预测的输出结果两部分组成），并利用这些测试实例执行程序和检查程序的输出结果与预测结果的一致性，以发现程序的错误。现有的软件测试方法如图1所示：

软件测试方法 { 穷举测试 / 黑箱测试 / 白箱测试 }　　图1

〈1〉穷举测试（Exhaustive Testing）

为了进行测试，如何设计测试实例呢？在理想情况下，人们希望通过测试能发现程序的全部错误。这要么把程序的整个输入域选取为测试实例的输入数据，或选取这样的测试数据，使程序的全部路径都被遍历。这就是所谓的穷举测试。我们知道程序的输入域是无穷的，而程序的路径数一般也是用天文数字来衡量。因此，穷举测试一般是不可能的。

〈2〉黑箱测试（Black-box Testing）

黑箱测试又称功能测试，数据驱动测试或输入/输出驱动测试。黑箱测试把程序看作黑箱，即测试员完全不涉及程序的内部动作和结构，相反测试员仅注意程序的执行结果与说明书之间的矛盾。测试实例的设计仅从要求、说明书、设计等开发阶段的说明或功能方面考虑，没有利用程序内部结构方面的信息。但是，人们不能期望通过测试来保证程序完全不包含错误，而是通过有限的测试实例达到发现尽可能多的错误的目的。

〈3〉白箱测试（White-box Testing）

白箱测试也称结构测试或逻辑驱动测试，它允许测试员检查程序的内部结构，利用程序的内部结构信息设计测试实例；一个好的白箱测试方法应是穷举的路径测试，即通过测试实例人们可以检查程序控制流的全部可能路径。前面讲过，这也是不可能的。

编译程序测试普遍采用的是黑箱测试，它主要在于编译程序的测试实例生成方法。到目前为止，自动生成测试程序普遍采用的构造策略是基于文法的测试方法，以语法为基础的生成器处理用BNF表示的测试程序规范（实际上就是编译程序处理的语法规范）。每给定一个测试程序文法，生成器就产生满足该文法的一个测试程序集合。例如〔14〕，〔11〕。基于文法的测试程序生成并不依赖于程序的内部结构，即使用的仍然是黑箱测试原则，所利用的信息源主要是语言的语法公式。这以Purdom给出的算法〔14〕为代表，该算法的输入是高级语言的上下文无关的语法产生式集，输出是满足该语法规范的符号串或语言。Purdom算法的生成策略有以下两个特点　1)生成最短句子的最小集合（在终结符串长度的意义上是最小）；2)保证每一个语法产生式至少使用一遍。

在编译程序测试中，较复杂的问题是如何生成上下文依赖的程序，测试编译程序的语义分析器。在这方面人们已作了许多工作，例如〔6〕，〔9〕，〔10〕。一种是以Purdom算法作为程序自动生成的基础，加上处理上下文相关的信息。例如〔9〕中，Celentano等人使用了动作文法（Grammar with action）的概念，它首先采用Pvrdom算法生成语法上正确的程序，然后利用定义在语法符号上的动作来修改已生成的程序，保证上下文一致性。另一种就是研究开发一些上下文有关的文法，保证上下文的一致性，在这一方面有代表性的是关于属性文法和参数文法的研究。例如，Franco，Bazzich〔8〕等人开发的参数文法（parametric grammar）作为测试文法，可以一次性生成满足语法和语义的测试程序。但是要写一个好的参数文法不太容易，且只能局限于某语言，实现上也有一定的困难。

经过现有测试方法的分析，传统的测试方法难于满足编译程序的测试要求，不能给利用测试的知识和经验等方面提供支持，这就促使我们研究开发新一代的测试环境——基于知识的编译程序自动测试系统（KCATS）。KCATS[1]是由一个知识库和一个知识库支持环境组成，如图2所示，对于KCATS来说知识库是可变的，用户可以修

KCATS
图2

改或向知识库中添加知识，而知识库支持环境是不变的，它通用于各种语言的源程序生成。知识库支持环境是KCATS的推理机制，它包括测试实例的生成方法和生成策略，它控制某种语言的源程序生成按照自顶向下，从左到右带有回溯的语法和语义引导的生成策略和生成与检验相结合的生成策略。这是对测试实例的生成树来说，它的生成就是自顶向下，从左到右带有回溯的深度优先生成。在测试实例生成中，我们认为最困难的问题是上下文依赖的语义信息的表示和处理，经我们研究的KCATS生成方法，模拟人在程序设计时的思维方式，由于人在程序设计时，总是先考虑到语句体中可能要用到的标识符及语义信息，然后在说明部分给出说明。因此，KCATS运用回溯机制主要体现在先生成语句体后生成说明部分，用于保证说明部分和语句体部分的语义信息一致性。对某程序块的语句体部分生成按生成与检验相结合的生成策略，例如，每生成一个标识符都要检验它与以前所生成标识符的语义一致性。这样就保证了语句体部分的语义一致性。KCATS的生成策略克服了以前过程式的测试系统，只生成一些固定类型程序的不足，由于基于识识系统的可扩充性，能使KCATS成为一个实用、通用的系统。

2.编译程序测试的域知识及其表示

编译程序测试的域知识也就是KCATS系统中知识库所包含的知识，它包括目标源语言的知识，具体编译程序的特征知识，有关测试的经验知识，那么这些知识怎样表示呢?在AI中，已有许多知识表示方法，例如，一阶谓词逻辑，语义网络，产生式规则和框架结构等，但是，在编译程序测试研究中，我们发现AI中现有的一般知识表示方法很难满足编译程序测试的具体要求，特别是测试实例生成过程中语义信息的表示和使用还没有很好的方法。属性文法精确地反映了语义属性之间的继承和综合关系，它已广泛地用于描述高级语言的语义和用于编译程序的自动生成。Papakonstantinou提出了一种用属性文法表示产生式规则的方法[15]，在此基础上，我们对属性文法用于知识表示作了深入的研究[3]，把AI中的框架、规则和属性文法相结合，研究了一种适用于编译程序自动测试系统实现的知识表示方法。

1)有关目标语言的语法知识和语义知识

目标语言的语法知识一般比较规范，以BNF的语法公式给出，而语义知识的表示和使用没有现成的方法，在这方面我们作了专门研究，把AI技术和属性文法相结合，提出了一种新的表示方法。

由于属性文法主要研究了控制流的构造，用于编译程序测试是难于实现的。我们认为纯粹的属性文法本身是过程式的，直接用于知识的外部表示并不合适，经过研究我们采用了建立在框架表示基础上经过修改的属性文法概念，在较高的描述层次上较直观的反映语义以及属性之间的关系。并且，我们研究出的文法只有继承属性而没有综合属性，属性计算是用规则的形式来表示，这样就避免了原属性文法计算综合属性和继承属性的繁琐，也与AI中的知识表示方法相统一，便于在基于知识的系统中实现。

语法知识表示采用规则，对程序语言文

法的每一条产生式对应一条语法规则，表示形式如下：

RULE i IF前提 THEN结论

前提是产生式的左部符号，结论是产生式的右部符号串

例如，有上下文无关文法 G=（V_N, V_T, P, S）其中V_N是非终结符号集 V_T是终结符号集 P是产生式集 S是文法开始符号

V_N={S, PROG, BLK, DCL, STM, ID, TYPE, VAR}

V_T={begin, end, ;, ,, :, :=, integer, real, a, b, c, …, x, y, z}

P中的产生式用BNF表示如下：

1. S::=PROG 2. PROG::=BLK
3. BLK::=begin DCL STM end
4. DCL::=ID: TYPE; DCL
5. DCL::=
6. STM::=ID:=VAR; STM
7. STM::=BLK; STM
8. STM::=
9. VAR::=ID
10. TYPE::=integer
11. TYPE::=real
12. ID::=a|b|…|x|y|z

用它的语法规则表示，就得到相应的十二条规则，例如

RULE1 IF S THEN PROG

RULE2 IF PROG THEN BLK

RULE3 IF BLK THEN begin DCL STM end

……

RULE11 IF TYPE THEN real

RULE12 IF ID THEN 调用词法生成器

语义知识表示采用如下框架的表示形式：

〈Frame〉::=(〈Frame-name〉[〈slot〉]*)

〈Slot〉::=(〈slot-name〉[〈Facet〉]*)

〈Facet〉::=(〈Facet-name〉[〈Facet-content〉]*)

其中 Frame-name=文法非终结符号

slot-name={语法规则槽，语义属性槽，属性计算规则槽，…}

语法规则槽的内容是一些IF…THEN…的语法规则

语义属性槽的内容是该Frame-name在测试实例生成中的动态语义（attribute或叫属性）。

属性计算规则槽的内容是一些规则，形如

IF attribute THEN子框架名的属性

当然Frame中的一些槽还有相应的侧面（facet）以及侧面的内容（facet-content）。

例如：文法G中的非终结符号DCL对应一个框架

Frame-name=DCL

语法规则槽的内容={IF DCL THEN ID: TYPE; DCL, IF DCL THEN}

语义属性动态决定，可是integer或real属性计算规则，可以是

IF Attribute=integer THEN (Attr(ID):=integer, Attr(DCL:=real)

IF Attribute=real THEN (Attr(ID):=real, Attr(DCL):=integer)

2）有关具体编译程序的特征知识

编译程序在实现时的约定与限制以及编译环境、运行环境的限制等都是编译程序的特征知识，它有两个方面的作用，其一，限制了测试程序的模式和行为，当然它也是测试实例产生的先决条件，其二，为边界值测试（Boundary-value testing）提供了必要的信息。例如，标识符的长度，数的上下界，过程的嵌套次数等。

编译程序的边界值限制不是源语言的要求，而是依赖于编译程序的处理能力，用户可以根据它控制被生成程序的结构复杂度，在测试实例生成时，这种限制主要用于描述某些语法产生式可应用的次数。

编译程序的特征知识一般比较规范，它的表示，就是在语法语义知识框架（Frame）中，增加两个槽（slot），一个槽value，它的内容是一个正整数值，在测试实例生成过程中动态确定，另一个槽Assertion，它的内容是一些断言规则，形式如下：

IF Frame-name is Macro THEN Value:=边界值

ELSE Value:=数值

value槽的值，在测试实例生成过程中，

可以由用户确定，也可以由assertion槽以断言规则给出，value中的整数值随着语法规则的运用而变化，即每调用一次语法规则，value的值递减整数1，直到value的值等于1时，调用一次能退出该框架的语法规则，作为最后一次应用该框架的语法规则。

这样的知识表示，既适用对边界值的测试，也适用对编译程序其它特殊重点测试的程序生成，还可以由用户在生成过程中通过用户交互来确定，由于知识系统的可扩充性和易修改性，这样的表示是通用的。

3)有关测试的经验知识

要构造高质量的测试程序集，尽可能多地发现编译程序存在的错误，一个有经验的测试员往往根据编译程序的特点和文本选择一批测试实例，只需要极少的信息，就能对编译程序的错误进行定性和定位的分析，而且所测试的错误具有一定的深度。由此可见，编译程序测试是一个经验性很强的工作。因此，基于知识的编译程序测试系统中，对经验知识的获取、表示和使用是一项非常重要的工作。有关测试的经验知识，包括测试策略的经验知识，测试数据生成的经验知识和测试人员提供的知识。在KCATS的研究中，我们对经验知识的表示，引入宏知识的概念。宏知识包括包括宏符号、宏符号的向上宏信息和以宏符号为左部的语法规则的向下宏信息。(1)宏符号是文法非终结符号，所有宏符号的全体构成一个宏符号集，具体的宏符号集是系统设计者根据测试经验给出，这是因为每一个宏符号都表示对编译程序的某一方面的测试；(2)宏符号的向上宏信息是指能从文法开始符号经过语法规则推导到该宏符号和需要满足完全性的语法规则的集合，一般是满足该条件的最小集合；(3)应符号的每条语法规则的向下宏信息，是指那些能从以宏符号为左部的该语法规则推导出终结符号的所有语法规则集合的一个子集，子集的确定也是由测试人员以经验而定。实质上是测试人员的经验知识。

宏知识的表示采用框架形式

〈Frame〉::=(〈Frame-name〉[〈Slot〉]*)

〈Slot〉::=(〈Slot-name〉[〈Slot-content〉])

其中 〈Frame-name〉::=〈宏符号〉

〈Slot-name〉:={〈Symbol-Rule〉,〈Up-Macro〉, 〈Down-macro〉}

Slot的内容：

〈Symbol-Rule〉：表示以框架名为左部的所有语法规则

〈Up-Macro〉：表示该框架名宏符号的向上宏信息

〈Down-Macro〉：表示该框架的所有symbol-Rule的向下宏信息

例如，上下文无关文法G，假设它的宏符号集Macro-symbol={DCL, STM}，说明要测试的重点是它的说明部分和语句部分以及两部分的组合。

1) 〈Frame-name〉=DCL

〈Symbol-Rule〉={Rule4, Rule5}

〈Up-macro〉={Rule1, Rule2, Rule3, Rule8}

〈Down-Macro〉={Rule10, Rule11, Rule12}

2) 〈Frame-name〉=STM

〈Symbol-Rule〉={Rule6, Rule7, Rule8}

〈Up-Macro〉={Rule1, Rule2, Rule3, Rule5}

〈Down-Macro〉={Rule4, Rule5, Rule9, Rule10, Rule11, Rule12}

宏知识是测试实例生成的经验知识，它通过好的用户接口来使用，这是因为宏符号能通过一定的形式（例如菜单）为用户知道，用户只要向系统提供宏符号，就确定系统要测试的内容，系统就能自动生成一批满足用户要求的测试数据。因此，宏符号的给出和组合隐含着测试人员一定的经验知识。

三 小结

本文用AI方法对编译程序测试问题进行了探讨，主要对编译程序测试中的知识提取和知识表示给出了具体的方法和形式，这是知识工程的难点之一，是关系到知识系统成败的关键，特别对测试实例生成提出了新方法，有利于解决语义的一致性问题；对经验知识的表示，既有利于对编译程序多方面进

自测试，增加测试功能，也方使用户使用，便于了解系统的工作方式。

我们的目的是通过对知识的形式表示的研究，建立起一个对编译程序进行自动测试的工具环境KCATS，它包括用测试数据执行程序代码，检查输出结果，以及将输出结果与预期结果进行比较，并具有某种初步的自学习功能，以实现对先前测试实例执行结果的自动分析，并将分析结果进行反馈，使KCATS能够根据反馈信息进一步生成更加完备的测试程序集，直到某预定义的条件被满足。

参考文献

1. 华庆一，顾元祥，周继鹏，基于知识的编译程序自动测试环境与kcats系统，全国首届智能科学研讨会论文专集，1988。
2. 华庆一，顾元祥，周继鹏，基于知识的自动程序设计环境，计算机科学，1988.3
3. 顾元祥，华庆一，周继鹏，属性文法用于知识表示，kcats研究报告，№.1 1987.6.
4. 顾元祥，李友仁，"属性文法的理论，应用与实现"，微电子学与计算机，(1987.12) PP.1—38.
5. 李友仁，顾元祥，华庆一，编译程序的测试方法与实现，计算机技术，(1985.12) PP. 5—10.
6. Bochman.G.V,"Semantic evaluation from left to right." Commun. Ass. Comput., Vol. 19. (feb. 1976)PP. 55—62.
7. Beizer. Boris. Software Testing Techniques. NewYork, Van Nastrand Reinhold.inc.1984.
8. Bazzichi.F.and I.Spadafora. "An automatic generator for compiler testing." IEEE. Trans on software engineering. Vol.SE-8.№.4 (July 1982) PP. 343—353.
9. Celentano. A."Compiler testing using a sentence generator."Software practice and experience. Vol. 10. (nov 1980) PP.897—918.
10. Duncan.A.C., J.S.Huchison, "using attributed grammar to test design and implementation." IEEE, Trans on software engineering,PP.170—178.(1981).
11. Hanford.J. V. "Automatic generation of test cases." IBM. Syst.F9 (1970) PP.113—128.
12. Minsky.M.,A Framework for Representing knowledge.The Psycholosy of Computer Vision.eds.P.Winston. New York. Megraw—Hill.211—277.1975.
13. Nii.H.P."blackboard systems, the blackboard model of problem solving and evolution of blackboard architecture." AI Magazine(sum.1986) PP.136—151.
14. Purdom.P., "a sentence generator for testing parsers."Bit, 12, (1972) PP. 166—375.
15 Papakonstantinou. G., "knowledge representation with attribute grammars." The Computer Journal. Vol.29. No.3 (1986)PP. 241-245.
16. Papakonstantinou.G."an interpreter of attribute grammers and its application to waveform analysis." IEEE Trans. software Eng.,Vol.SE—7(May 1981). PP.279—283.
17. Roth.F.H., D.A. Waterman and D. B. Lenat, eds. Building Expert System. Tokyo, ADDISON.Wesley.1983.

下期主要内容预告

关于第六代计算机的研究；神经网络的基本结构和功能；软件工程中的AI思想及其研究；软件开发知识的表示与利用模型SOKM；软件CAD：一种革命性方法；分布式专家系统，协同问题求解。