计算机科学2004Vol.31№.2

基于模型的软件测试综述*)

颜 炯 王 戟 陈火旺
(国防科技大学计算机学院 长沙410073)

摘 要 随着面向对象软件开发技术的广泛应用和软件测试自动化的要求,特别是基于UML的软件开发技术的逐渐普及,基于模型的软件测试逐渐得到了软件开发人员和软件测试人员的认可和接受。针对被测试软件的不同特征和不同测试目的,已经提出了多种测试模型。本文详细阐述了基于模型的软件测试研究现状和应用现状,并对测试中使用的不同模型进行了比较,着重介绍了状态机模型、UML模型和马尔可夫链模型。最后提出了未来的研究方向。

关键词 软件测试,基于模型的测试,有限状态机,UML,马尔可夫链

Survey of Model-Based Software Testing

YAN Jiong WANG Ji CHEN Huo-Wang
(School of Computer Science, National University of Defense Technology, Changsha 410073)

Abstract Software testing requires the use of models to guide test selection and test verification. Thus is the so-called model-based software testing, which has recently gained attention with the popularization of models (including UML) in software design and development. There are a number of models of software in use today, for example, finite state machines, UML model and Markov chain, which server as models for testing. This paper introduces model-based testing and discusses its applications in general terms with finite state models and Markov chain model as examples. In addition, advantages, difficulties, and shortcoming of various model-based approaches are concisely presented. Finally, we close with a discussion of the future research directions of model-based testing in software engineering.

Keywords Software testing, Model-based testing, Finite state machines, UML, Markov chain

1 引言

如何提高软件质量是软件工程致力解决的关键问题之一。软件测试和验证是保证软件正确性和提高软件可靠性的最基本和最重要的手段,也是工业界使用的主流技术。软件测试有不同的方法,基本可以划分为基于程序的测试和基于规范的测试两大范畴。基于程序的测试根据程序代码特征(如语句、分支、路径)产生测试例,并以相应程序代码特征是否得到了充分测试作为测试终止的判定标准,即测试充分性准则。如果测试充分性准则确定的程序特征在测试过程中都得到完全的执行,则认为对代码的测试是充分的。基于规范的测试要求根据功能规范和设计规范产生测试用例,针对相应的功能属性和设计属性进行测试,并以相关属性是否得到了充分测试作为测试充分性准则,如果测试充分性准则确定的相关属性在测试过程中都得到完全的执行,则认为测试是充分的。

然而,目前工业界在软件开发过程中采用的测试技术大多是非系统化的,通常面向特定软件产品。航空航天、武器装备、医疗设备等领域的软件具有实时性强、可靠性要求高的特点,软件失效将导致人员和财产的重大损失,典型的例子有1996年Ariane五型火箭发射失败和1997年火星探测器的失控。这类软件系统即所谓安全攸关实时软件系统,对其进行正确性验证十分困难。安全攸关实时软件系统的复杂性和高安全需求使得传统的基于程序代码的测试在技术上难以进行,开销上难以接受。

随着面向对象软件开发技术的广泛应用和软件测试自动化的要求,基于模型的软件测试(Model-Based Software Testing)逐渐得到重视。基于模型的软件测试属于基于规范的软件测试范畴,其特点是:在产生测试例和进行测试结果评价时,都是根据被测试应用程序的模型及其派生模型(一般称作测试模型)进行的。基于模型的测试最初应用于硬件测试,广泛应用于电信交换系统测试,目前在软件测试中得到了一定应用,并在学术界和工业界得到了越来越多的重视[1]。本文分析讨论基于模型的软件测试相关理论和技术。

本文组织如下:首先介绍了基于模型的软件测试及其特点,分析了主要的测试模型及如何选择合适的测试模型,重点是有限状态机模型、UML模型和马尔可夫链模型;然后介绍如何收集必要的信息来构造模型,以及如何由模型生成测试用例和进行测试结果评价;最后讨论了基于模型的软件测试在软件开发过程中的应用及未来发展方向。

2 基于模型的软件测试和测试模型

2.1 基于模型的软件测试

软件模型是对软件行为和软件结构的抽象描述。软件行为可以用系统输入序列、活动、条件、输出逻辑或者数据流进行描述,软件结构则使用组件图、部署图等进行描述。针对测试任务,通过对软件功能和结构进行抽象并用易于理解的方式进行描述,获得的模型就是对被测试软件系统精确的描述,可以用于软件测试。一般对软件不同行为要用不同模型进行

*) 本研究得到国家自然科学基金(90104007,60233020)和863计划(2001AA113202,2001AA113190)资助。

描述。例如:控制流图、数据流图和程序依赖图表达了程序和代码结构间的行为关系,决策表和状态机则可以描述软件外部行为。基于模型的软件测试可以根据软件行为模型和结构模型生成测试用例。当前软件规模庞大也使基于程序的测试十分困难,而基于模型的软件测试方法不仅可以有效地提高测试效率,提高测试例生成的自动化程度,进行测试失效辨识,也有利于评价测试结果。

2.2 软件测试模型

软件测试中使用的典型模型有:有限状态机、UML 模型和马尔可夫链等模型。

2.2.1 *有限状态机* 基于有限状态机的测试模型假设软件在某个时刻总处于某个状态,并且当前状态决定了软件可能的输入,而且从该状态向其它状态的迁移决定于当前的输入。有限状态机模型特别适用于把测试数据表达为输入序列的测试方法,并可以利用图的遍历算法自动产生输入序列。

有限状态机可以用状态迁移图或状态迁移矩阵表示,可以根据状态覆盖或迁移覆盖产生测试用例。有限状态机模型有成熟的理论基础[2],并且可以利用形式语言和自动机理论来设计、操纵和分析,特别适合描述反应式软件系统,是最常用的软件描述和软件测试的模型。Beizer 等详细讨论了基于状态机的软件测试[5,7],基于有限状态机模型的测试研究已经取得一定研究成果[8~10],但是复杂软件往往要用很复杂的状态机表示,构造状态机模型的工作量比较大,因此自动构造软件的有限状态机模型非常关键。

2.2.2 *UML 模型* 作为事实上的面向对象建模标准语言,统一建模语言(UML)在面向对象开发过程中得到了广泛应用,出现了大量商品化的支持工具,如 Rational Rose,基于 UML 模型的测试也得到了广泛关注。基于 UML 模型的测试研究主要集中于 UML 的状态图(statechart),还很少利用其它图(如部署图、组件图)的模型信息[14]。状态图是有限状态机的扩展,强调了对复杂实时系统进行建模,提供了层次状态机的框架,即一个单独状态可以扩展为更低级别的状态机,并提供了并发机制的描述[11],因此 UML 使用状态图作对单个类的行为建模。文[3]研究了基于状态机的测试充分性准则,文[13]研究了基于状态图的软件测试。

2.2.3 *马尔可夫链* 是一种以统计理论为基础的统计模型,可以描述软件的使用,在软件统计测试中得到了广泛应用。马尔可夫链实际上是一种迁移具有概率特征的有限状态机,不仅可以根据状态间迁移概率自动产生测试例,还可以分析测试结果,对软件性能指标和可靠性指标等进行度量[18,20]。另外,马尔可夫链模型适用于对多种软件进行统计测试,并可以通过仿真得到状态和迁移覆盖的平均期望时间,有利于在开发早期对大规模软件系统进行测试时间和费用的规划。马尔可夫链是统计测试的基本模型,在净室软件工程中得到了深入研究,在微软、Raytheon 及美国联邦航空署(FAA)都得到了成功应用[4]。

马尔可夫链可以用随机迁移矩阵或者带迁移概率的状态迁移图表示。基于马尔可夫链的测试充分性准则一般要求测试过程中对马尔可夫链迁移的覆盖与实际使用相同[20]。

2.2.4 *文法模型* 可以描述程序的语法。由于不同的文法等价于不同的状态机,因此也可以视为状态机模型的变体。有关基于文法的测试可见文[12],这方面研究工作相对较少。

3 基于模型的软件测试过程及支持工具

3.1 分析理解被测试软件

基于模型的软件测试要求充分理解被测试软件。构造可以用于测试的模型的工作主要是根据测试目的确定测试对象和测试特征,针对被测试软件的相关属性建立相应模型。这个阶段的具体工作包括:

·充分了解软件需求规范和设计文档、用户手册,和开发队伍充分交流。

·识别软件系统的用户,枚举每个用户的输入序列,研究每项输入的可能取值范围,包括:合法值、边界值、非法值,以及预期输出。这项工作往往需要工具支持。

·记录输入发生条件和响应发生条件。软件系统的响应是指用户能够得到的输出或可见的软件内部状态的改变。其目的是设计可以引发特定响应的测试例和评价测试结果。

·研究输入序列,如:输入发生时刻,软件系统接收特定输入的条件,输入处理顺序。

·理解软件内部数据交换和计算过程,产生可能发现缺陷的测试数据。

3.2 选择合适的测试模型

不同的模型适用于不同类型软件的测试,因此需要根据软件特点选择模型。Sommerville 等讨论了模型选择标准[17],本文进一步总结如下:

3.2.1 *了解可用的模型* 不同的应用领域要使用不同的测试模型。例如,电话交换系统多使用状态模型;并发软件系统中不同组件并发运行用状态图建模;马尔可夫链可以对软件进行失效统计分析。模型的选择还依赖于软件系统的工作特点,例如,测试长期运行软件系统可以使用状态机模型。

3.2.2 *根据模型特征进行选择* 只有充分理解模型和软件系统,才能选择合适的模型对软件进行测试。以状态机为例,自动机理论可以对状态机进行分类,说明不同的状态机可以表达什么语言,从而可以根据应用程序的功能和特点,选择不同的状态机模型。由于根据有限状态机产生测试数据相当于遍历有向图,因此图论算法可以指导产生测试例。

3.2.3 *人员、组织和工具的影响* 基于模型的软件测试对测试人员的知识结构和技术水平提出了一定要求,如果一个开发组织使用模型(如 UML)完成需求分析和系统设计,开展基于模型的软件测试就比较容易,因为根据系统分析和设计的模型进行测试,往往可以直接应用基于模型的软件测试技术,还可以根据测试的进展不断调整模型或模型的细节,并有利于在开发过程早期进行测试规划。另外还要根据开发组织使用的测试工具选择特定的模型。

3.3 构造测试模型

我们以基于状态机模型的测试为例说明如何构造测试模型。首先要抽象出软件系统状态,状态抽象一般要根据输入及输出条件进行,一般包括以下过程[9]:

·生成一个输入序列并说明每个输入的适用条件,称作输入约束。例如电话未摘机时,才允许有摘机动作发生。

·对每个输入要说明产生不同响应的上下文环境,称作响应约束。例如,电话摘机时,如果当前状态为振铃,则进行通话,否则为拨号音。

·根据输入序列、输入约束和响应约束构造相应状态机模型。

El-Far 等描述了一个框架,可以从软件非形式规范自动建立有限状态机模型[22]。Whittaker 研究了如何用有限状态机模型和层次方法建立使用模型[9];Walton 等讨论了如何构造马尔可夫链模型[21];上述技术都可以用于模型构造。

3.4 生成和执行测试例

测试例的自动生成依赖于测试所使用的模型。以有限状态机模型为例，被遍历路径中弧的标记构成的序列就是测试例。在构造满足测试准则的路径时，必须考虑约束条件，如路径长度限制，统计测试还要考虑迁移概率[19]。

生成了满足特定的测试充分性准则的测试例集合后就可以执行测试例。以状态机模型为例，首先要写出仿真该软件系统的每个不同外界激励的脚本，称为仿真脚本(simulation script)，每个仿真脚本对应模型中一个不同的迁移；然后把测试例集合翻译为测试脚本(test script)。也可以用测试生成器通过遍历状态机的迁移直接产生测试脚本。测试例的执行就是测试脚本的执行过程。脚本是可以重复利用的资源，维护测试脚本也是测试工作的一部分。

3.5 收集测试结果进行分析

基于模型的软件测试方法并没有解决测试失效辨识(Test Oracle)问题，仍然要人工检查输出是否正确。但是通过状态验证可以部分解决测试失效辨识问题，状态是内部数据的抽象，比较容易验证。另外，与传统测试相比，基于模型的软件测试的优势之一就是可以根据测试结果分析软件的其它质量因素，如可靠性[20]。

3.6 支持工具

基于模型的软件测试必须有相关工具支持。当前支持基于模型的软件测试工具中比较具有代表性的有：

·支持状态机模型的工具。包括：Software Engineering Technology 的测试工具 toolSET_Certify，运行于 RISC6000 和 SUN 平台；现在此工具属于 Q-Lab，并已经提供了对统计测试的支持；IBM 的 GOTCHA，可以根据用户事先确定的测试充分性准则进行基于软件状态模型的测试例生成；IBM 的 TCBean 是一个提供测试脚本管理功能的基于状态机的测试引擎。

·支持马尔可夫链模型的工具。包括：Cleanroom Software Engineering 的 CleanTest，支持统计测试，是商用的使用模型及统计测试例生成工具；IBM 的 Cleanroom Certification Assistant，可以自动化统计验证过程，通过使用概率分布产生测试例，并对测试结果进行分析。

·对 UML 模型提供测试支持的工具。包括：SilverMark 公司针对 IBM 公司的 VisualAge 开发的支持测试用例生成和回归测试的 TestMetor 和 UML Designer Connection。

·对统计测试结果进行分析的工具。包括 AT&T 的软件可靠性工程工具箱和美国海军水面战中心开发的 SMERFS，均提供多种时间域和区间域模型进行可靠性估计；美国空军喷气动力实验室开发的 CASRE，支持可靠性估计和预测。

4 基于模型的软件测试的评价

4.1 基于模型的软件测试的优势

基于模型的软件测试大大提高了测试自动化水平，部分解决了测试失效辨识问题，可以进行测试结果分析，有利于测试制品的重用，并可以应用成熟的理论和技术获得比较完善的分析结果。现代软件工程强调增量和迭代的开发过程，采用面向对象开发技术，提高软件开发质量和加快软件开发速度。由于面向对象软件系统的规范多数使用模型表达，这些模型中包含了大量可以用于软件测试的信息，因此基于模型的软件测试可以把软件测试工作提前到开发过程早期，如进行测试工作的早期规划。

统计测试是最成功的基于模型的软件测试，已经在工业界得到了广泛应用。统计测试可以通过仿真对测试时间和费用进行量化估计，并有利于跟踪控制整个测试过程。目前基于模型的软件测试多应用于嵌入式系统和 GUI 设计和开发[18]，并已经逐步扩展到可编程硬件接口和基于 WEB 的应用[6]。随着社会对软件质量重视程度的提高，特别是模型逐渐成为规范和设计制品一部分的情况下，基于模型的软件测试必然会得到广泛应用。

4.2 基于模型的软件测试的缺点

基于模型的软件测试存在以下缺点：

·测试人员需要具备一定的理论基础，如状态机理论和随机过程的知识。还要掌握相关工具使用方法。

·需要一定的前期投入，如模型的选择、软件功能划分、模型构造等。

·有时无法克服模型的固有缺陷，如状态爆炸，这对检验、评审和维护模型都提出了要求，也直接影响了测试自动化。状态爆炸问题一般通过抽象和排除方法(abstraction & exclusion)减小测试规模，降低测试难度来解决[9]。

·现有的基于马尔可夫链的统计测试多数应用于通用商业软件开发，很少用于超高可靠性软件测试。统计测试只能说明软件在正常使用情况下的可靠性，而在某些特定使用情况下(如核电站紧急关机)的测试实施和可靠性评估，还需要进一步深入研究。如何把统计测试应用于超高可靠性软件是统计测试得到进一步应用的关键。

结论 基于模型的软件测试得到越来越广泛的应用，并在学术界和工业界都得到了重视。软件测试模型是对软件行为和软件结构的抽象描述，可以用于生成软件测试例，进行测试失效辨识，从而有效提高测试效率，并有利于测试结果评价。

基于模型的软件测试还需要在以下几个方面进行深入研究：

·对软件测试模型本身的研究。一方面要研究针对不同类型软件开发专用模型，另一方面也要对专用模型进行抽象获取通用模型。模型研究包括理论和应用两方面，理论研究包括稀少事件统计模型、贝叶斯可靠性模型、超高可靠性软件的测试模型研究。由于经典统计理论难以直接应用于超高可靠性软件质量验证，因此超高可靠性软件质量验证往往需要根据先验数据，在基于贝叶斯理论的指导下进行。目前多数软件的统计测试是在净室软件过程指导下完成的，开发过程中得到的经验数据可以为超高可靠性软件的质量验证提供先验数据。超高可靠性软件测试是基于模型的软件测试必须解决的关键问题，也是未来测试模型理论研究的主要方向。软件测试模型应用研究主要关注测试工具的研究开发。

·测试充分性准则研究。朱鸿等研究比较了基于程序代码结构的软件测试充分性准则[16]，但没有涉及基于模型的软件测试的充分性。当前基于状态机模型的测试充分性准则一般有输入覆盖、状态覆盖、迁移覆盖，但是这些测试充分性准则在基于模型软件测试中的有效性和效率还有待于进一步深入研究。

·测试失效辨识和测试自动化。测试失效辨识和测试自动化密不可分。一方面，失效辨识是测试自动化不可缺少的内容，另一方面，自动化又使得测试失效辨识更加困难。目前对测试失效辨识的研究比较少，实践中常通过对比不同软件版本的输出结果进行测试失效辨识[15]。因此测试失效辨识是未

来研究的重要内容。

参考文献

1 Gronau I, Hartman A, Kirshin A, Nagin K, Olvovsky S. A methodology and architecture for automated software testing. http://www.haifa.il.ibm.com/projects/verification/gtcb/papers/gtcbmanda.pdf, 2000

2 Chow T S. Testing design modeled by finite-state machines. IEEE Transactions on Software Engineering, 1978,4(3): 178～187

3 Offutt J, Abdurazik A. Generating test cases from UML specifications. UML' 99, USA, 1999

4 Poore J H. Introduction to the special issue on: model-based statistical testing of software intensive systems. Information and Software Technology, 2000, 42(12):797～799

5 Beizer B. Black-Box Testing: Techniques for Functional Testing of Software and Systems, Wiley, New York, USA, 1995

6 Jorgensen A, Whittaker J A. An API Testing Method. STAREAST' 00, USA, 2000

7 Fujiwara S, Bochmann G, Khendek F. Test selection based on finite state models. IEEE Transactions on Software Engineering, 1991, 17(6): 591～603

8 Rosaria S, Robinson H. Applying models in your testing process. Information and Software Technology, 2000, 42(12): 815～824

9 Whittaker J A. Stochastic software testing. The Annals of Software Engineering, 1997, 4:115～131

10 Liu C, Richardson D J. Using application states in software testing. ICSE'00, Ireland, 2000

11 Harel D. Statecharts: a visual formalism for complex systems. Science of Computer Programming, 1987, 8(3): 231～274

12 Maurer P M. The design and implementation of a grammar-based data generator. Software Practice & Experience, 1992, 23(3): 223～244

13 Hong H S, Kim Y G, Cha S D. A test sequence selection method for statecharts. The Journal of Software Testing, Verification & Reliability, 2000, 10(4): 203～227

14 Abdurazik A, Offutt J. Using UML collaboration diagrams for static checking and test generation. UML' 00, UK, 2000

15 Peters D K, Parnas D L. Using test oracles generated from program documentation. IEEE Transactions on Software Engineering, 1998, 24(3): 161～173

16 Zhu H, Hall P, May J. Software unit test coverage and adequacy. ACM Computing Surveys, 1997, 29(4): 366～427

17 Sommerville I, Sawyer P. Requirements Engineering: A Good Practice, Wiley, Chichester UK, 1997

18 vritzer A, Larson B. Load testing software using deterministic state testing. ISSTA'93, USA, 1993

19 Avritzer A, Weyuker E J. The automatic generation of load test suites and the assessment of the resulting software. IEEE Transactions on Software Engineering, 1995, 21(9): 705～715

20 Whittaker J A, Thomason M G. A Markov chain model for statistical software testing. IEEE Transactions on Software Engineering, 1994, 20(10): 812～824

21 Walton G H, Poore J H. Generating transition probabilities to support model-based software testing. Software: Practice and Experience, 2000, 30(10): 1095～1106

22 El-Far I K. Automated Construction of Software Behavior Models: [Master's Thesis]. Florida Institute of Technology, Melbourne, Florida, USA ,1999

(上接第44页)

Round(i)个回合后为其他i个结点所知,即有i－1条边分别从除u以外的i－1个结点指向v。此时,G_{i+1}成为一个有向完全图。证毕。

5.3 性能分析

5.3.1 *时间复杂度* 双向反馈算法的时间复杂度为$\Omega(d_{initial})$。在最坏的情况下,即initial图中每个结点的出度仅为1(每个结点只向其他一个结点发送自己所知的资源信息),$d_{initial}$为initial图中最长路径的长度,则$d_{initial}=n-1$。n为initial图中结点的个数。(由于$G_{initial}$是连通的,且每个结点必有出度为1的一条边,因此该连通图要连通n个结点,必须有n－1条边。若有(v－>u),(u－>w),则结点w上的资源信息只有在R_i回合传给u后,才能在R_{i+1}回合传给v。v无法直接从w上得知w的资源信息。因此,算法收敛的回合数就是$d_{initial}$。

5.3.2 *连接通信复杂度* 该算法的连接通信复杂度为$O(m_{initial}*Round)$,上限为$2*m_{initial}*d_{initial}$。在每一个回合中所有结点总共发出$m_{initial}$个连接,在反馈时,每个接受连接的结点又会反馈回一个连接。所以,算法的连接通信上限为$2*m_{initial}*d_{initial}$。

5.3.3 *指针通信复杂度* 其指针通信复杂度为$\Omega(2n*m_{initial})$。因为在达到算法收敛的过程中,每个结点的资源信息都必须在初始邻接图的每条边上发送给对方一次,同时又收回一次对方的指针。收敛过程中共有2n个指针在$m_{initial}$条边上传送。

结束语 针对ad-hoc移动网络环境,我们在现有的泛洪算法的基础上经过改进,将移动agent和改进的泛洪算法相结合,提出了一种新的资源发现算法:双向反馈算法,该方法有两个优点:一是能在未知网络其他结点的initial list的情况下,保证网络中每一个结点都能发现网络其他结点的资源,二是可以节约算法收敛的时间,代价是通信量略有增加。

参考文献

1 王敏毅.面向移动环境的分布对象技术:[博士论文].电子科技大学,2002

2 Harchol-Balter M, Leighton T, Lewin D. Resource discovery in distributed networks. In: 18th Annual ACMSIGACT/SIGOPS Symposium on Principles of Distributed Computing, May1999

3 Law T, Siu K Y. An O(log n)Randomized Resource Discovery Algorithm. In: 14th Intl. Symposium on Distributed Computing, Oct. 2000

4 Jun Y, Boloni L, Palacz K, Dan C. Agent-Based Resource Discovery, Oct. 1999

5 Kutten S, Peleg D. Deterministic distributed resource discovery. In: 19th Annual ACM SIGACT/SIGOPS Symposium on Principles of Distributed Computing, July 2000