基于扩展CHAM模型的软件集成测试方法*)

叶俊民[1,2,3] 罗 景[1,3] 朱 凯[1,3] 赵 良[1,3] 赵 恒[2] 王振宇[2]

(华中师范大学计算机科学系 武汉430079)[1]

(哈尔滨工程大学计算机科学与技术学院 哈尔滨150001)[2]

(武汉大学软件工程国家重点实验室 武汉430072)[3]

摘 要 基于构件的软件工程(Component-Based Software Engineering,CBSE)正逐渐成为软件开发的一种新趋势。目前构件提供者所交付的构件仅仅包括其功能描述和接口描述,而代码通常是不可见的,这给软件系统的集成测试带来较大困难。本文针对软件体系结构(Software Architecture,SA)层中的构件规格说明,提出一种集成测试方法。首先使用扩展的化学抽象自动机(Extend Chemical Abstract Machine,E-CHAM)模型描述软件系统的体系结构;接着使用LTS状态树生成算法,生成标号迁移系统(Labeled Transition System,LTS)表示单一构件的动态行为;最后按照自底向上的集成测试策略完成了整个系统的测试。

关键词 集成测试,软件体系结构,化学抽象自动机,标号迁移系统,覆盖准则

Software Integration Testing Method Based on E-CHAM Model

YE Jun-Min[1,2,3] LUO Jing[1,3] ZHU Kai[1,3] ZHAO Liang[1,3] ZHAO Heng[2] WANG Zhen-Yu[2]

(Department of Computer Science, Central China Normal University, Wuhan 430079)[1]

(College of Computer and Technology, Harbin Engineering University, Harbin 150001)[2]

(State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072)[3]

Abstract CBSE (Component-Based Software Engineering) is increasingly being adopted for software development. Currently, components delivered by component provider only include specifications of functions and interfaces, which source code may not be available to component user. This imposes significant difficulties on the testing of an integrated component-based system. The paper proposes an approach to derive test plan for integration testing of component-based software system. Firstly, this method describes software architecture using E-CHAM (Extend Chemical Abstract Machine) model, and then models dynamical behavior of single component using LTS state tree generation algorithm. Finally, we adopt bottom-up incremental testing strategy performing the whole integration testing process.

Keywords Integration testing, Software architecture, CHAM, LTS, Coverage criteria

1 引言

基于构件的软件工程正逐渐成为软件开发的新趋势,但是也给基于构件的软件系统测试带来了新的问题。Harrold[1]认为应该从构件提供者和构件使用者两个不同的角度来看待测试问题。两者的一个重要区别在于,提供者有构件的源代码,而使用者没有源代码。当构件从提供者交付到使用者时,往往假定单个构件已经进行了彻底的测试,但是一旦这些构件集成到应用环境时,就可能会产生不匹配问题。因此,对集成后的构件进行测试成为CBSE实用化的关键所在。

A. Bertoline[2]等人提出了从软件体系结构描述中导出实现层的测试用例,以指导构件系统的集成测试,其思想是使用化学抽象自动机CHAM描述应用系统,从中导出描述整个系统行为的标号迁移系统LTS,利用抽象的原则导出高层ALTS(Abstract LTS, ALTS),再利用观察函数导出SA层次的测试计划,以此为依据并结合UML的活动图导出指导实现系统的测试用例。

我们在SA测试计划生成的研究中,提出了通用LTS状态树生成算法[3],该算法是解决从CHAM模型的应用系统中导出测试计划的关键步骤,我们实现了该算法并采用文[2]中提到的一个远程医疗系统(Teleservice and Remote Medical Care System,TRMCS)为例进行了实验,得到了该系统的一个LTS状态树。

在上述工作的基础上,本文提出一种基于CBSE的软件集成测试方法。首先用CHAM中膜的概念对SA进行建模,接着用通用LTS状态树生成算法导出单个构件行为的LTS状态树,最后利用SA设计结论和集成测试策略完成基于CBSE的构件系统的集成测试。

2 基本概念

2.1 化学抽象自动机CHAM

构件的功能描述,可采用化学抽象机[4]这一形式化模型。CHAM主要使用了化学隐喻来表现软件中的各种概念,如构件和构件的运行等,即分子(molecules)$m_1, m_2, \cdots$构成了CHAM的基本元素,而溶液(solutions)$s_1, s_2, \cdots$是多个分子的集合,溶液可解释成一个CHAM状态的定义。一个CHAM规格说明包含迁移规则(transformation rules)$T_1, T_2, \cdots$,这些规则定义了一个迁移关系$s_i \rightarrow s_j$,该迁移关系说明了在

*)基金项目:国防预研基金资助项目(413150601);武汉大学软件工程国家重点实验室基金资助项目(SKLSE04-20)。叶俊民 副教授,硕士生导师,主要研究领域为软件工程,软件体系结构;罗 景 硕士研究生,主要研究领域为软件体系结构,软件测试。

CHAM 中溶液可以进化的依据，在此的进化就是指溶液的状态改变。

使用 CHAM 描述的 SA 层的构件规格说明，主要包括有四个部分：(1)一个可以表达系统构件和连接件(即分子)语法描述；(2)表示系统初始状态的溶液；(3)一组描述构件之间是怎样进行交互以完成系统的动态行为的迁移规则；(4)一组表示所希望系统的最终状态的溶液。

定义1 一个化学抽象自动机 C 是一个4元组(S,T,s_0,S_F)，在此 S 是溶液的集合，$s_0\in S$ 是初始溶液，$T\subseteq S\times S$ 是迁移规则的集合，$S_F\subseteq S$ 是终止溶液的集合。

2.2 膜 membrane

膜[4](membrane)是一种封装结构，以区别溶液内环境和溶液外环境，膜内的溶液可以独立进化。膜具有半可渗透性(semi-permeable)，允许某些分子进入和离开。通过膜上的气孔(airlock)，可以有选择性地从膜中抽取分子。膜的反应方式和进化规则形式与溶液的反应方式和进化规则的形式是一致的。膜实际上提供了一种刻画系统模块化的途径。

定义2 封闭在$\{|\cdot|\}$中的溶液 $s=m_1,m_2,\cdots,m_n$ 称为膜。通过气孔可将溶液 s 上分子 m_i 抽取出来，这一过程表示为：$s'=m_i\lhd\{|m_1,\cdots,m_{i-1},m_{i+1},\cdots,m_n|\}$。

2.3 标号迁移系统 LTS

在 LTS[2]中的每一个状态对应于一个溶液，由描述构件状态的分子的集合构成。LTS 弧上的标号表示迁移规则，该规则表示系统从尾部节点状态到头部节点状态。

定义3 一个标号迁移系统 A 是一个五元组(S,L,s_0,S_F,T)，在此 S 是状态的集合，L 是标号的集合，$s_0\in S$ 是初始状态，$S_F\subseteq S$ 是终止状态的集合，$T=\{\xrightarrow{l}\subseteq S\times S\mid l\in L\}$是迁移关系。

定义4 $A_1||A_2=(S,L,s_0,S_F,T)$表示两个 LTS 的合成，其中，$A_1=(S_1,L_1,s_{01},S_{F1},T_1)$，$A_2=(S_2,L_2,s_{02},S_{F2},T_2)$，$S=S_1\times S_2$，$L=L_1\cup L_2$，$s_0=(s_{01},s_{02})$，$S_F=S_{F1}\times S_{F2}$。$T$ 表示为下述三种情况($P,P'\in S_1,Q,Q'\in S_2$)：

$$\frac{P\xrightarrow{l}{}_1P'}{P\|Q\xrightarrow{l}P'\|Q}(l\notin L_2)\qquad(1)$$

$$\frac{Q\xrightarrow{l}{}_2Q'}{P\|Q\xrightarrow{l}P\|Q'}(l\notin L_1)\qquad(2)$$

$$\frac{P\xrightarrow{l}{}_1P'\quad Q\xrightarrow{l}{}_2Q'}{P\|Q\xrightarrow{l}P'\|Q'}(l\in L_1\cap L_2)\qquad(3)$$

3 从体系结构层导出测试用例

本文方法分为三个步骤：(1) 采用 CHAM 描述软件系统及其构件的动态行为，根据需要使用膜对系统模块化；(2) 根据通用 LTS 状态树生成算法，导出单一构件的 LTS 状态树；(3) 按复合 LTS 来构造子系统测试用例，并采用软件系统集成测试策略，以递增方式完成整个系统的集成测试。下面以 TRMCS 为例，说明整个方法中的各个步骤。

3.1 用 CHAM 描述 TRMCS

图1给出了 TRMCS 的体系结构描述，将其中的五个构件的行为用对应的 User1、User2、Router、Server、Timer 进程来表示。利用膜结构和气孔来定义模块和接口，从而将要分析的构件独立于系统中的其他构件。如图1所示，User1独立出来，将 User2、Router、Server 和 Timer 作为一个模块 M，用灰色框表示。M 的内部行为独立于 User1，故忽略模块 M 内部溶液的进化，直接观察 User1和模块 M 所构成的整个系统的行为，这个行为其实等同于 User1和外部构件的一个交互过程。

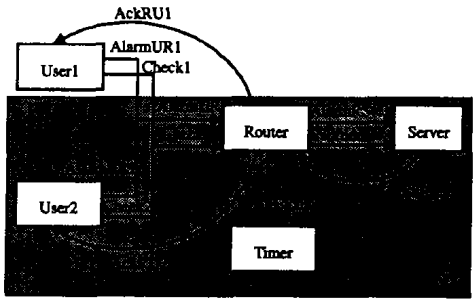

图1 远程医疗系统体系结构

按照定义1，我们给出图1中模块化远程医疗系统的形式化描述。系统的初始溶液 s_0和终止溶液 s_f 定义如下，其中 s_0^M 代表模块 M 的初始状态。

$s_0=User1\diamond o(check),User1\diamond o(alarmUR1)\diamond i(ackRU1),s_0^M$

$s_0^M=(alarmUR)\diamond o(alarmRs)\diamond i(ackSR)\diamond o(ackRU)\diamond Router,$

$i(check)\diamond Router,i(alarmRS)\diamond o(ackSR)\diamond Server,$

$i(nofunc)\diamond Server,Timer,Sent$

$s_f=s_0$

整个系统的迁移规则，前21条与文[2]中定义的迁移规则一样(由于篇幅问题我们在此不给出)。结合测试要求，为了使得模块内外能够通过接口进行反应，还要补充两条 CHAM 的基本规则 T_{22}和 T_{23}[4]。其中，$m,m_1,\cdots,m_k$ 表示分子；$\diamond$表示分子的状态；$\lhd$表示气孔结构。

$T_{22}:\{|m\diamond m_1\lhd\{|m_2,\cdots,m_k|\}|\}=m\diamond\{|m\diamond m_1\lhd\{|m_2,\cdots,m_k|\}|\}$

$T_{23}:\{|m\diamond m_1\lhd\{|m_2,\cdots,m_k|\}|\}\diamond m=\{|m_1\diamond m\lhd\{|m_2,\cdots,m_k|\}|\}$

3.2 从 CHAM 描述导出 LTS

3.2.1 LTS 状态树生成算法 假设规则和溶液的数据结构用广义表来表达。算法描述有 Reaction 数组和 Match 函数。前者的分量为广义表，用以记录发生反应的每一对分子 C_i和 C_2及其反应规则 T_k，该数组是上下文无关的。Match 函数检查每一对分子 C_i和 C_2是否可以发生反应，从而确定状态树中下一个状态的生成。当所生成的状态没有出现过，则该状态变量的所有值都被保留一张 Hash 表中，将该状态放入优先队列中。

下面给出 LTS 状态树生成算法，当溶液反应终止时，对应的状态树就已经构造成功。

Step1 初始化数组 Reaction，使之为空；初始化溶液 S_0，使之为初始溶液；初始化 S_1，使之为终止溶液；

Step2 比较 S_0与 S_1，如果 $S_0=S_1$(即达到终止溶液)，则终止；否则做：

Step3 对 S_0中的每一个分子 $C_i(i=1..n)$进行检查，如 C_i属于自反应型分子，则 C_i 进行重写操作，即用规则的右部重写其左部；同时构造状态树；

Step4 对 S_0中的每一对分子 $C_i,C_j(i,j=1..n)$进行 Match(C_i,C_j,T_k)检查，如果匹配计算结果中的 T_k 不为空，则将 C_i,C_j 和 T_k 的值存入数组 Reaction 中；

Step5 当数组 Reaction 不为空时，对非空的数组分量进

行重写操作，以生成新的溶液 S_0，同时构造状态树；当数组 Reaction 为空时，返回 Step2。

3.2.2 *单一构件 LTS 的生成* 由3.1节定义的23条规则中，我们找到对应于初始溶液中分子 User1◇o(check1)，User1◇o(alarmUR1)◇i(ackRU1)和 s_0^M 的反应规则 T_3，T_5，T_7，T_8，T_{11}，T_{17}，其他的规则可以看作是模块 M 内部的迁移规则，对外部分子没有影响力。将这6条规则以及规则 T_{22} 和 T_{23} 共同作用于初始溶液 s_0 上，由 LTS 状态树生成算法，则可导出构件 User1的 LTS 来，如图2所示。同理可以得到其他构件的 LTS 出来，其中构件 Router 的标号迁移系统如图3所示。

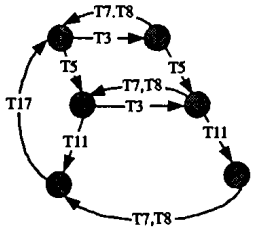

图2 构件 User1的 LTS

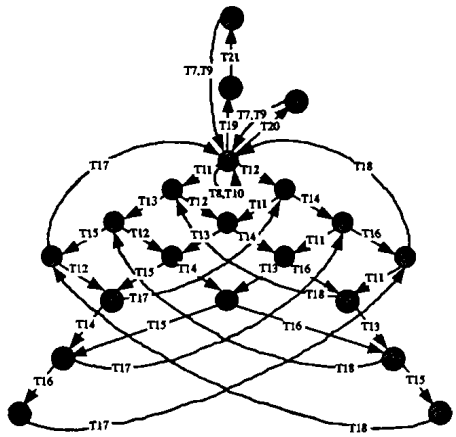

图3 构件 Router 的 LTS

3.3 自底向上增量集成测试策略

3.3.1 *组装 LTSs* 组装分析的核心思想是将一个系统分解为由子系统构成的层次模型 H，接着按照该模型再将其集成起来，同时在集成的过程中进行简化[5]。例如，TRMCS 的一种合理分解，可表示为图4所示的层次结构。圆角方框代表子系统，直角方框则表示标号迁移系统已经被导出的初始构件。

在组装的过程中，表示子系统的复合 LTS 由其包含的所有构件的 LTS 复合而成，由定义4给出。例如，子系统 $G1$ 由构件 *User*1和 *Router* 构成，定义为：

$G1=(User1||Router)$

整个系统的 LTS 定义为：

$TRMCS=(User1||User2||Router||Server||Timer)$

如图5所示，是 $G1$ 的部分 LTS，整个 $G1$ 的复合 LTS 状态数达到120个。用双数字来表示每个状态，第一个数字对应于 User1的当前状态，另一个则表示 Router 的当前状态。

3.3.2 *测试覆盖准则* 依据文[6]，定义如下的基于结构的覆盖准则：

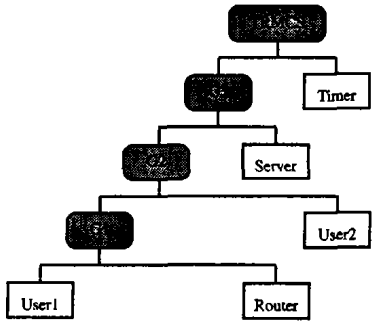

图4 远程医疗系统组装层次模型

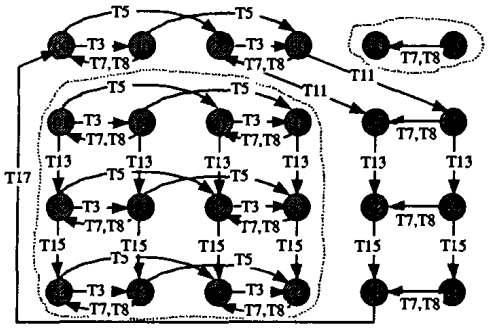

图5 User1||Router 的复合 LTS 略图

1)全路径覆盖：覆盖 LTS 中所有的路径至少一次。当在 LTS 中存在环时，这条准则将不被满足。

2)完全路径覆盖：覆盖 LTS 中所有的完全路径至少一次。如果一条路径中除头尾两个节点外，不包括两个重复的节点，那么这条路径称之为完全路径。

3)迁移覆盖：覆盖 LTS 中所有的迁移至少一次。

4)状态覆盖：覆盖 LTS 中所有的状态(节点)至少一次。

上述的覆盖准则的覆盖能力依次减弱。

3.3.3 *集成测试策略* 进行自底向上增量集成测试[6]，以集合 L 作为标号迁移系统 LTSs 的集合，该 LTSs 代表系统中的构件行为，同时给出一个组装层次 H。我们完成一个自底向上的遍历 H 中的非叶子节点过程。当遍历非叶子节点 N 时，将执行下面的步骤：

step1 为节点 N 的所有孩子节点生成一个复合的标号迁移系统 L_N，采用第3.3.1节中所用到的组装方法。

step2 按照第3.3.2节中定义的测试覆盖准则，从 L_N 中选择一个测试路径集 T，T 被用来发现节点 N 的孩子节点中的交互错误。

step3 如果 N 是根节点，则终止。否则，将 L_N 作为节点 N 的标号迁移系统，转 Step1。(L_N 将被用于构造节点 N 的父节点的标号迁移系统)

结束语 本文提出了一种从体系结构层导出测试用例，来指导实现层测试的方法。该方法通过 CHAM 形式化描述整个软件系统，用标号迁移系统 LTS 来对系统的动态行为进行建模。当软件系统的源代码不可见时，该行为描述可以作为生成测试用例的基础，最终得到高质量的软件系统。在一台 P4机(主频1.8GHz，128M 内存)上，对上述远程医疗系统实例进行了实验。实验结果表明，对整个 TRMCS 而言，生成了11518个状态覆盖集元素，生成了28487个迁移覆盖集元素和256392条路径数。这表明，我们的算法思想是实际可行的。

与 A. Bertoline 等人进行相比，本文的工作有以下几点不

(下转第205页)

由表3可知，当进行第一次验证测试时，需要的验证测试持续期为 $t_1=3652.8$小时，当软件无失效通过这么长的测试期，则软件达到规定的可靠性指标，验收合格；如果软件在时刻 $tf_1=2000$小时发生失效，则说明软件可靠性达不到规定的可靠性指标，整个验证测试过程应该停止下来排除错误。那么第二次验证测试所需要的无错测试运行时间为 $t_2=5685.9-tf_1=3685.9$小时。同样，在验证测试过程中，如果软件在时刻 $tf_1=3652$小时发生失效，则第二次验证测试所需要的无错测试运行时间为 $t_2=5685.9-tf_1=2033.9$小时。显然，第一次软件失效时间的早晚直接影响着第二次验证测试所需要的测试持续期。如果软件在进行第二次验证测试时，顺利通过规定的测试持续期 t_2，那么软件验收合格；而如果在测试过程中的 tf_2时刻($tf_2<t_2$)，软件又发生失效，则需要排除错误并确定需要的第三次无失效验证测试持续期 $t_3=7453.5-tf_1-tf_2$。t_3 的具体取值与前面两次失效发生时间的早晚是息息相关的，这充分体现了对前面两次验证测试信息整合的结果。后面的测试过程以此类推。

根据式(4)，我们可以计算出基于经典统计假设检验的FDT方法对同样的可靠性指标进行验证测试时，每次所需要的软件验证测试持续期都固定为4605.2小时。显然，我们所提供的方法不但能根据测试过程中的具体情况动态调整测试持续期，而且由于考虑了软件的先验信息，所需要的测试持续期更短，这都充分体现了该方法的优越性。对于不同的 a_0 和 b_0，都可以方便地根据式(18)产生相应的表，辅助决定在不同情况下需要的验证测试持续期，使得连续执行软件可靠性验证测试过程变得简单易行。

结论 本文提出的基于经验贝叶斯统计推断的连续执行软件可靠性验证测试方法不但考虑了软件在测试过程中的具体表现情况，而且还考虑了软件可靠性的先验信息，客观地反映了被测试软件可靠性的实际，能有效地减少验证测试所需要的测试持续期，为验证高可靠性指标提供了理论和技术支持。同时，本文提供的软件失效强度先验分布超参数的求解方法又使得先验整合方法在工程中的实施成为可能。在本文工作的基础上，加上测试用例的等价类划分方法和形式化的程序转换方法，使得在当前一些不可验证的超高可靠性指标的验证成为可能。当然，一种理论和技术的成熟，需要在实践中接受检验并加以完善，希望更多软件可靠性验证测试专家学者提出指正。

参考文献

1 Michael L, ed. Handbook of software reliability engineering. McGraw Hill and IEEE Society Press, 1996
2 http://www.openedu.com.cn/file_post/display/read.php?FileID=23060
3 Selding P B. Faulty software caused Ariane 5 failure. Space News, 1996, 7(25): 24～30
4 Leveson N G, Turner C S. An investigation of the Therac-25 accident. IEEE Computer, 1993, 26(7): 18～41
5 何国纬，等著. 软件可靠性. 国防工业出版社，1998
6 徐仁佐. 软件可靠性工程的基本概念、任务及实施方法. 软件世界，1993. 2～3
7 Pham H. Software reliability and testing, IEEE Computer Society Press, 1996
8 Cai K Y, Cai L, Wang W D, et al. On the neural network approach in software reliability modeling. Journal of systems and software, 2001, 58(1): 47～62
9 白成刚，俞蒙槐，胡上序，等. 基于 Bayes 网的软件失效预测模型. 计算机科学，2003，30(6)：162～164
10 毛晓光，邓勇进. 基于构件软件的可靠性通用模型. 软件学报，2004，15(1)：27～32
11 邹丰忠，李伟湘. 软件可靠性混沌模型. 计算机学报，2001，24(3)：281～291
12 董成，毛新军，陈磊，等. 基于 Agent 的软件可靠性评估系统. 计算机科学，2000，17(6)：28～31
13 黎忠文，熊光泽. 软件可靠性评估的误差分析. 系统工程与电子技术，2001，23(6)：87～89
14 US Department of Defense. MIL-HDBK-781A: Reliability test methods, plans and environments. 1996
15 Butler R W, Finelli G B. The infeasibility of quantifying the reliability of life-critical real-time software. IEEE Trans. on Software Engineering, 1993, 19(1): 3～12
16 Tal O. Software dependability demonstration for safety-critical military avionics system by statistical testing. PhD Thesis, Nottingham Trent University, 1999
17 Tal O, MoCollin C, Bendell A. Reliability demonstration for safety-critical systems. IEEE Trans. on Reliability, 2001, 50(2): 194～203
18 覃志东，雷航，熊光泽，等. 一种实时多任务软件可靠性验证方法. 系统工程与电子技术(已录用)
19 覃志东，雷航，桑楠，等. 安全关键软件可靠性验证测试方法研究. 航空学报(已录用)
20 Qin Zhidong, Lei Hang, Sang Nan, et al. Safety-critical software reliability demonstration testing by empirical Bayesian method. International Symposium on Computing and Information, Aug. 2004

(上接第201页)

同：(1) 本文是分别导出系统中每个单一构件的 LTS，通过这些 LTSs 的集成，来构造整个系统的 LTS，后者直接导出整个系统的 LTS 以作为下面的测试准则选择的基础。(2) 本文对测试用例的选择是基于结构化测试覆盖标准，后者采用的是未明确定义的观察函数。(3) 本文测试策略是自底向上的集成测试策略，后者主要采用的是自顶向下的集成测试策略。

参考文献

1 Harrold M J, Liang D, Sinha S. An Approach To Analyzing and Testing Component-Based Systems. In: Proc. of First Intl. ICSE Workshop on Testing Distributed Component-Based Systems. Los Angeles, 1999. 134～140
2 Bertolino A, Corradini F, Inverardi P, Muccini H. Deriving Test plans from architectural descriptions. In: Proc. of the 22nd Intl. Conf. on Software Engineering. ACM Press, 2000. 220～229
3 叶俊民，王振宇，曹瀚，赵恒. 基于 CHAM 模型的 LTS 状态树生成算法. 哈尔滨工程大学学报，2003，24(3)：287～291
4 Inverardi P, Wolf A L. Formal Specifications and Analysis of Software Architectures Using the Chemical Abstract Machine Model. IEEE Transactions on Software Engineering, 1995, 21(4): 373～386
5 Cheung S C, Kramer J. Enhancing Compositional Reachability Analysis with Context Constraints. In: Proc. of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering. D Notkin, 1993. 115～125
6 Koppol P, Carver R H, Tai K C. Incremental Integration Testing of Concurrent Programs. IEEE Transactions on Software Engineering, 2002, 28(6): 607～623
7 Weyuker E J. Testing Component-Based Software: A Cautionary Tale. IEEE Software, 1998, 15(5): 54～59