基于 FSM 的测试理论、方法及评估

刘 攀[1),2)] 缪淮扣[1),2)] 曾红卫[1)] 刘 阳[1)]

[1)](上海大学计算机工程与科学学院 上海 200072)

[2)](上海市计算机软件评测重点实验室 上海 200072)

摘 要 为搭建基于 FSM 的测试方法由理论研究通往工业应用的桥梁,文中讨论了若干基于 FSM 模型的测试方法及其相关理论,提出构造区分序列的理论及测试序列集合冗余约简的理论,补充并实现了若干基于 FSM 的测试生成算法.随后文中提出了基于 FSM 的测试方法评估的 5 项量化指标,实验评估了若干基于 FSM 模型的测试方法,给出了一些基于 FSM 测试的经验建议.文中的评估有助于基于 FSM 的测试方法在工业中的推广.

关键词 基于 FSM 的测试;测试评估;测试生成算法;基于 FSM 的测试理论;冗余约简

中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2011.00965

FSM-Based Testing: Theory, Method and Evaluation

LIU Pan[1),2)] MIAO Huai-Kou[1),2)] ZENG Hong-Wei[1)] LIU Yang[1)]

[1)](*School of Computer Engineering & Science, Shanghai University, Shanghai* 200072)

[2)](*Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai* 200072)

Abstract To build the bridge between theoretical studies and industrial application towards FSM-based testing, the authors study some FSM-based testing theories and methods, and present the theories for building distinguishable sequence and reducing the redundancies among test set derived from FSM-based testing methods. Next some practical algorithms for realizing some FSM-based testing methods are designed in the paper. Finally, the authors evaluate some FSM-based testing methods by using five suggested evaluation indicators on some experiments. According to experimental results, the authors give some experiential suggestions for the implementation of FSM-based testing methods. It is intended to help with understanding benefits and limitations of FSM-based testing, selecting the approach used in a particular FSM-based testing method and popularizing FSM-based testing in the industry.

Keywords FSM-based testing; test evaluation; test generation algorithm; FSM-based testing theory; redundancy reduction

1 引 言

软件测试是提高软件可靠性和保证软件质量的一种重要手段.传统的软件测试依赖于人工测试,是一项耗费最大的、劳动密集型的活动,因此传统软件测试效率低下、易出错、不完全和不可再生.为保证软件质量,提高软件测试效率,降低测试费用,测试自动化成为一种必然的趋势[1].

基于模型的测试是测试自动化研究的一个重要

收稿日期:2010-10-20;最终修改稿收到日期:2011-04-27.本课题得到国家自然科学基金(60970007,61073050)、国家"九七三"重点基础研究发展规划项目基金(2007CB310800)、上海市自然科学基金(09ZR1412100)、上海市科学技术委员会(10510704900)、上海市重点学科建设项目(J50103)资助.**刘 攀**,男,1976 年生,博士,讲师,主要研究方向为软件测试、基于模型的测试、软件形式化方法及算法设计.E-mail: liupan@shu.edu.cn.**缪淮扣**,男,1953 年生,教授,博士生导师,主要研究领域为软件工程和软件形式化方法.**曾红卫**,男,1966 年生,博士,研究员,主要研究领域为形式化方法、形式验证和软件测试.**刘 阳**,男,1981 年生,博士研究生,主要研究方向为形式化方法和形式验证.

方向.该方法能由描述软件需求的形式规格说明生成测试序列,再检测待测系统(System Under Test,SUT)[2]的运行是否严格遵守它的形式规格说明.由于测试序列的生成、测试执行及测试结果的分析能在一定程度上自动实现,因此降低了软件的测试成本,提高了测试效率.

有限状态机(Finite State Machine,FSM)作为一种形式建模语言已被广泛用于描述软件的需求,基于FSM的测试方法亦被广泛研究.然而,基于FSM的测试仍然处于理论研究状态,并未在工业界得到广泛应用[1,3].为该方法搭建一条由理论界通往工业应用界的桥梁就显得非常必要,而充分的实验评估是实现这一目标的有效方式之一.

现有研究对基于FSM的测试评估并不完善.Sidhu等人[4]通过一个案例研究了一些基于FSM的测试方法,并比较了由这些方法生成的测试序列总长度及错误覆盖能力,但该文仅讨论了若干基于FSM的测试方法的构造思想,而没有提供可实践的算法,且文中讨论的错误类型非常有限,与实际软件中的错误类型不符;Dalal等人[5]利用一个基于模型的测试工具测试了4个实际应用,并讨论了基于模型的测试生成时的注意事项,但支持工具接受的建模语言是定制的AETGSpec,而非通用的FSM模型;Broy等人[6]对若干基于FSM的测试方法的理论进行综述,而并未对这种测试方法进行实验评估;Neto等人[3,7]通过对五大电子图书馆的索引,分析和讨论了现有基于模型的测试研究工作.然而他们没有比较不同方法的优劣;Ammann和Offutt[8]则从测试覆盖准则的角度评估了基于FSM的测试方法,但并未给出测试覆盖准则的实现算法,导致实际应用的困难.

本文重点讨论基于FSM的测试方法的理论、实现算法及冗余约简原理.以6个FSM模型及其应用为实验对象,评估若干基于FSM的测试方法,并提出了5个量化评价指标.依据实验的结果,本文对基于FSM的测试提出了一些经验性的指导意见.

本文的创新点在于:(1)补充了基于FSM的测试理论,完善了区分序列的构造理论,提出了新的冗余约简方法;(2)提出了评估基于FSM的测试方法的5项量化指标,并评估了若干基于FSM的测试方法;(3)给出了基于FSM的测试的一些经验性建议.

本文第2节介绍基于FSM测试的一些基础知识;第3节综述若干基于FSM的测试方法;第4节讨论测试序列集的冗余问题,并提出了冗余约简算法;第5节提出5项量化指标,评估基于FSM的测试方法;第6节对实验结果进行讨论,并为基于FSM的测试方法给出一些经验性的意见;第7节分析国内外同类研究;第8节总结全文,并指出今后的研究方向.

2 预备知识

有限自动机包括Moore机和Mealy机,其中Mealy机是由状态集合、输入符号集合、状态迁移函数、输出符号集合和输出函数构成,与通信协议的描述相一致.因此,Mealy机最早被用于对通信协议的建模和测试[9].Mealy机的定义[2,4,6,9]如下.

定义1(Mealy机). 一个Mealy机是一个五元组$M=\langle I,O,S,\delta,\lambda\rangle$,其中$I$是一个有限非空输入符号集合,$O$是一个有限非空输出符号集合,$S$是一个有限非空的状态集合,$\delta:S\times I\rightarrow S$为状态迁移函数,$\lambda:S\times I\rightarrow O$为输出函数.

在Mealy机中输入一条序列$x=a_1a_2\cdots a_k$,由状态迁移函数$\delta(s_i,a_i)=s_{i+1}(i=1,2,\cdots,k)$可获得一组连续的状态$s_1s_2\cdots s_k$,本文采用函数$\delta(s_1,x)$表示该组连续序列.同理,可以由输出函数$\lambda(s_1,x)$产生一条输出序列$b_1\cdots b_k$,其中$\lambda(s_i,a_i)=b_k(i=1,2,\cdots,k)$.在本文中,$x\cdot y$表示两个序列或输入符号的连接,$\lambda(s_1,x)=-$表示输出为空.

本文随后规定M_S表示软件需求的Mealy机,M_I表示软件实际应用的Mealy机.基于FSM的测试过程:依据测试方法(准则)由M_S中产生测试序列,再应用到M_I中,观察M_I是否严格符合它的规格说明M_S,即通过比较实际的测试序列和输出结果是否与预期一致,探测M_S和M_I之间所有的不一致性错误.因此通常称这种测试为一致性测试[1,2,6].现有基于FSM的测试研究是建立在一些前提和假设[6,10]的基础上.

假设1. M_S是被约简的或最小化的.对于任意两个状态s和t,存在一个输入序列x,使得x能依据输出函数$\lambda(s,x)$和$\lambda(t,x)$的不同,区分s和t.形式描述为$\forall s,t:S,\exists x:I^*\cdot(\lambda(s,x)\neq\lambda(t,x))$.

假设2. M_S是具体的.S中的每一个状态和I中的每一个输入符号都能应用状态迁移函数δ和输出函数λ.形式描述为$\forall s_i:S,\forall x:I\cdot(\exists s_j\in S,\exists y\in O^*\cdot(\delta(s_i,x)=s_j)\wedge(\lambda(s_i,x)=y))$.

假设3. M_S是强连通的.M_S中的每一个状态都可经过一条或多条连续的迁移由其它状态到达.

形式描述为 $\forall s_i, s_j : S \bullet (\exists p : I^* \bullet (\delta(s_i, p) = s_j))$.

某些基于 FSM 的测试方法并不严格遵守假设 3，而仅要求 M_S 中的每一个状态都能由初态到达（本文称为假设 3′）.

假设 4.　在测试过程中，M_I 并不会改变，且 M_I 与 M_S 拥有相同的输入和输出符号集合. 形式描述为 $I_{M_S} = I_{M_I} \wedge O_{M_S} = O_{M_I}$.

假设 5.　初始状态. M_S 和 M_I 都有一个初始状态 s_1，且测试之前 M_I 已经处在它的初始状态.

假设 5 并非严格要求，如 n 端口自动机则拥有多个初始状态.

假设 6.　状态数目相同：M_S 和 M_I 有相同数目的状态，M_S 和 M_I 之间的不一致性错误并不会造成 M_I 中状态的增加.

依据假设 6，M_I 中包含两类错误：(1) 输出错误，即 M_I 的输出与预期输出 $\lambda(s, x)$ 不符；(2) 迁移错误，即 M_I 的状态迁移与预期状态迁移 $\delta(s, x)$ 不符. 然而，实际软件应用 M_I 中状态数目可能增加或减少，造成迁移的增加或减少，本文将状态的变更带来的错误称为额外迁移错误. 迁移错误与额外迁移错误不同，迁移错误是指，在 M_S 的状态 s 上存在输入 a，但实际 M_I 的输出状态与预期状态不符；额外迁移错误是指，在 M_S 的状态 s 上不存在输入 a，但实际 M_I 的状态 s 上存在输入 a 使得 $\delta(s, a)$ 不空.

假设 7.　重置消息. M_S 和 M_I 都有一个重置消息 reset（简称 r），使得状态机在任何状态时都能回到初始状态 s_1 且不会产生任何输出.

假设 8.　状态位消息. M_S 和 M_I 都有一个特定的状态位（status），使得在任意时刻，它们所处的状态都能被状态位标识，例如状态 s_i 表明了状态所处的状态位是 i.

假设 9.　设置消息. 输入符号集合 I 包含一个特殊的集合 $set(s)$，当系统处于初始状态时，自动机能够获得一个 $set(s)$ 消息后，能够移动到状态 s，同时不产生任何输出.

在基于 FSM 的测试中，上述的假设不一定都需要满足，大多数的基于 FSM 的测试生成方法仅需满足其中的几条假设.

3　基于 FSM 的测试方法

3.1　图形的遍历方法

由 FSM 产生测试用例的思想来源于有向图的遍历[11]. 基于图形覆盖的程度，研究人员提出两类基本覆盖：状态覆盖和迁移覆盖[8]，并给出了由状态覆盖和迁移覆盖生成测试用例的方法. 在 FSM 中，一条迁移[12-13] 被定义为 $t = (s_i, a/b, s_j)$，其中 $f(s_i, a) = s_j$，$a \in I, b \in O$，s_i 为迁移 t 的前状态，s_j 为迁移 t 的后状态.

定义 2（状态覆盖）.　测试用例集合 T 至少覆盖了 M_S 中的所有状态一次.

定义 3（迁移覆盖）.　测试用例集合 T 至少覆盖了 M_S 中的所有迁移一次.

依据有向图的遍历方法，由初态开始遍历 FSM，生成一棵广度优先生成树，再获得由树根结点到叶子结点的迁移序列作为测试用例. 文献[14]中给出了广度优先遍历 FSM 的算法，但该算法并未考虑到迁移之间可能存在的约束关系[13,15]. 例如 BBS 论坛中存在两类访客，一类是未注册的游客，另一类为注册用户. 注册用户可以发表帖子和浏览其他用户的帖子，而游客只能浏览已发表的帖子，因此注册用户与发表帖子之间存在约束关系.

定义 4（迁移约束集）.　在 FSM 中，设 $RS = \{(t_i, t_j) \mid t_i \rightarrow t_j\}$ 为迁移约束集，其中 t_i 和 t_j 为 FSM 中的两条迁移且 t_j 的存在取决于 t_i 的存在.

假设图 1(a) 中 M_S 的迁移约束集为 $RS = \{(t_3, t_4)\}$. 若直接采用广度优先遍历算法，可获得一棵测试生成树，如图 1(b) 所示. 由测试生成树产生测试用例集合 $T = \{t_1, t_2t_3, t_2t_4t_5, t_2t_4t_6\}$. 但 T 并不满足迁移约束集 RS，为此本文提出了基于迁移约束的广度优先遍历算法 TBFS. 算法 TBFS 的思想：采用广度优先遍历 FSM，若存在迁移 $t \in \mathrm{ran}\, RS$，则回溯访问树中所有父亲结点，查看是否满足 $\exists t' : \mathrm{dom}\, RS \wedge t' \rightarrow t$；若满足，则采用迁移 t 构造生成树，否则跳过迁移 t 继续构造生成树. 例如在图 1(b) 中，当遍历到迁移 t_4 时，由于 $t_4 \in \mathrm{ran}\, RS$，则回溯遍历父亲结点，找到迁移 t_2，但由于 $t_2 \notin \mathrm{dom}\, RS$，则放弃 t_4 作为 t_3 的前状态 s_2 的后继. 依据广度优先遍历，获取 t_3 的后状态 s_2，选取 t_4 作为后续迁移，回溯遍历父亲结点，找到迁移 t_3. 由于 $t_3 \in \mathrm{dom}\, RS \wedge t_3 \rightarrow t_4$，则将 t_4 作为状态 s_2 的后继. 依据这种构造思想，最终获得的生成树如图 1(c) 所示. 若 FSM 中不存在迁移约束集时，算法 TBFS 转变为普通的广度优先遍历算法.

在最坏情况下，算法 TBFS 的时间复杂度为 $O(n^3)$，其中 n 为 $|S|$. 由图 1(c) 中的树可得，序列 $t_2t_3t_4$ 满足状态覆盖，序列 t_1，$t_2t_3t_4t_5$ 和 $t_2t_3t_4t_6$ 满足迁移覆盖. 图 1(a) 中 M_I 是 M_S 的一个错误应用，其中 $t' = (s_3, a/0, s_2)$ 是一个错误迁移. 由序列 $t_2t_3t_4$ 得到的输入序列 bab 和状态序列 $s_1s_2s_3$. 在 M_I 中输入 bab，获得实际运行的状态序列为 $s_1s_2s_3$，因此满足状

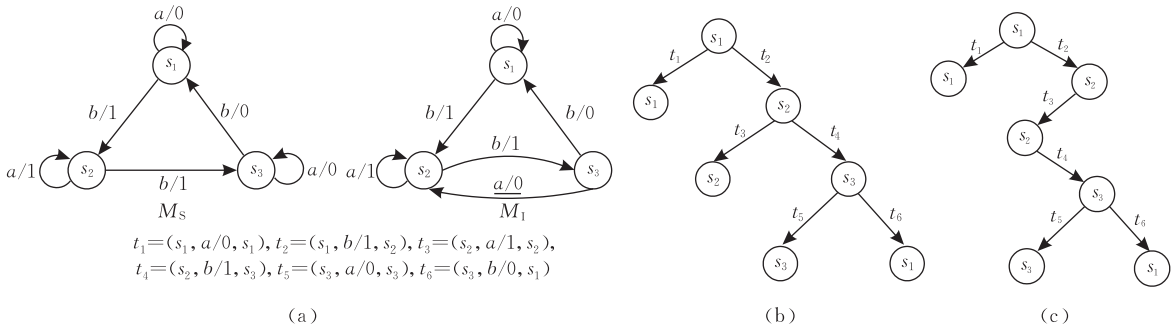


图 1

态覆盖的序列 $t_2t_3t_4$ 并不能发现迁移错误 t'. 而序列 $t_2t_3t_4t_5$ 包含的输入序列 $baba$ 和状态序列为 $s_1s_2s_3s_3$，在 M_1 中输入 $baba$，获得的实际状态序列为 $s_1s_2s_3s_2$ 与预期序列 $s_1s_2s_3s_3$ 不一致，因此满足迁移覆盖的序列能够发现 M_1 中的迁移错误 t'.

算法 TBFS. 带迁移约束的广度优先遍历生成树.

输入：存储 FSM 的邻接矩阵 ***array***[n][n]及它的迁移约束集 RS

输出：广度优先遍历生成树 $Tree$

1. 初始化列表 $list1=s1$，$list2=$ null 及其游标位置 $pre=0$，$post=1$；
2. 标记 $s1$ 为树 Tree 的根结点；
3. While ($pre!=post$)
4. $s=list1.get(pre)$; //取 $list1$ 中 pre 位置的值；
5. While (存在前状态为 s 的迁移 t)
6. 从数组 ***array***[n][n]中取下一个迁移 $t=(s,a/b,s_i)$；
7. If (t 在 $list2$ 中)
8. If ($t\in$ ran RS) //存在迁移约束
9. 访问从结点 s_i 到根结点的所有分枝；如果存在 $t_i\in$ dom RS 和 $(t_i,t)\in RS$，则标记 t 为一个分枝，s_i 作为 s 的后继结点；将 s_i 加入到 $list1$ 中，$post$++及 t 加入 $list2$ 中；
10. End if
11. Else
12. 标记 t 为一个分枝，s_i 作为 s 的后继结点；将 s_i 加入到 $list1$ 中，$post$++，t 加入 $list2$ 中；
13. End if
14. End while
15. pre++；
16. End while
17. Return Tree

3.2 W 方法

图形遍历的方法是通过观察系统中的状态是否到达来判断系统是否存在不一致性错误. 若应用系统的状态不可观察，则需要利用输出序列来判断系统中的错误. 这类测试生成方法包括 W 方法、Wp 方法、U 方法和 DS 方法等. 它们需要解决两个问题：(1) 如何生成覆盖 FSM 的输入序列；(2) 如何辨别最终到达的状态为目标状态.

W 方法最早是由 Chow[9] 提出的，并被应用于电信系统的测试. 该方法需要遵循 3 条性质：(1) 仅仅检测系统设计中的控制结构；(2) 规格说明不必是可执行的；(3) 测试序列一定能揭示控制结构中的错误，并能解释错误的来源. W 方法并不需要使用状态位信息，而是采用分离序列 (separating sequence)[6,16] 来区分自动机中的每个状态.

定义 5(分离序列). 对于 S 中的任意两个状态 s 和 t，它们的分离序列是 $x\in I^*$，以便使得 $\lambda(s,x)\neq\lambda(t,x)$. 形式描述为 $\forall s,t:S,\ \exists x:I^*\cdot(\lambda(s,x)\neq\lambda(t,x))$.

由假设 1 可知，规格说明 M_S 是最小的，则 M_S 中不存在两个等效的状态，即任意两个状态都存在分离序列. W 方法构造了一个输入序列覆盖集 P 来实现对 FSM 的所有迁移覆盖，同时构造了一个由分离序列组成的特征集 W，以区分最终到达的状态.

定义 6(输入序列覆盖集). M_S 的输入序列覆盖集 P 是由输入序列构成，以便对于 S 中的任意状态 s 和 I 中的每一个输入 a，都存在一个输入序列 $x\in P$，x 从 M_S 的初始状态 s_1 开始，到 $\lambda(s,a)$ 产生的输出结束. 形式描述为 $\forall s:S,\ \forall a:I\cdot(\exists x,y:P\cdot(x=y\cdot a\wedge\delta(s_1,y)=s)$.

由定义 6 可知，输入序列覆盖集 P 是由空输入 ε 和所有从初始状态开始到任意状态结束的输入序列构成，输入序列覆盖集 P 可由生成树获得.

定义 7(特征集). M_S 的特征集 W 是由分离序列构成的集合，以便对 S 中的任意状态 s 和 t，W 中都存在一个输入序列 x 使得 $\lambda(s,x)\neq\lambda(t,x)$. 形式描述为 $\forall s,t:S\cdot(\exists x:W\cdot(\lambda(s,x)\neq\lambda(t,x))$.

由于特征集能够区分 S 中任意两个状态，因此利用特征集构造测试序列可以不需要知道 M_S 的状态信息. W 方法通过广度优先遍历 FSM 的方式，构

造一棵测试生成树，获取由树的根结点到任意结点的输入序列构成了 P 集；同时，对状态集合 S 中的每一对状态 s 和 t，寻找这两个状态的分离序列，并用这些分离序列构成 W 集. W 方法的输入序列为 $r \cdot P \cdot W$，预期的输出序列为 $\lambda(s_1, r \cdot P \cdot W)$. 现有构造 P 集和 W 集的方法[6,9]中存在两个问题：(1) 对 P 集的构造并没有考虑 FSM 中的迁移约束；(2) 对 W 集而言，仅给出构造思想，并未提供分离序列的构造算法. 为克服第 1 个缺点，本文建议采用算法 TBFS 来生成测试树，再构造 P 集. 为克服第 2 个问题，本文提出了分离序列构造的新方法.

以图 1(a)中的 M_S 为例，本节演示了分离序列的构造方法. 首先为 M_S 建造一个输入/输出与状态之间的关系表，如表 1 所示. 在表 1 中，符号×表示某个状态应用了某个输入/输出. 由表 1 可知，状态 s_1 应用 $a/0$ 和 $b/1$，状态 s_2 应用 $a/1$ 和 $b/1$，状态 s_3 应用 $a/0$ 和 $b/0$.

表 1　M_S 的输入/输出与状态的关系表

I/O	$a/0$	$a/1$	$b/0$	$b/1$
s_1	×			×
s_2		×		×
s_3	×		×	

定义 8(可识别函数).　M_S 的关系表被定义为一个可识别函数 $\varphi: S \to I/O$，其中 S 是 M_S 的状态集，I/O 表示输入/输出集合，且满足 $\forall s: S, \exists x/y: I/O \cdot (\exists p: S \cdot \delta(s,x)=p \wedge \lambda(s,x)=y)$.

依据定义 8，图 1(a)中 M_S 的可识别函数为 $\varphi(s_1)=\{a/0, b/1\}$，$\varphi(s_2)=\{a/1, b/1\}$，$\varphi(s_3)=\{a/0, b/0\}$. 借助一组特定的运算，本文构造了一个三角矩阵.

定义 9(三角矩阵).　在 M_S 中，可识别函数的三角矩阵是由 $D_{ij}(i<j)$ 构成，其中

$$D_{ij}=\begin{cases}\varphi(s_i)-\varphi(s_j), & \varphi(s_i)\not\subset\varphi(s_j)\\ \varphi(s_j)-\varphi(s_i), & \varphi(s_i)\subset\varphi(s_j)\\ \varnothing, & \varphi(s_i)=\varphi(s_j)\end{cases}.$$

依据定义 9，$D_{12}=\varphi(s_1)-\varphi(s_2)=\{a/0\}$，$D_{13}=\varphi(s_1)-\varphi(s_3)=\{b/1\}$，$D_{23}=\varphi(s_2)-\varphi(s_3)=\{a/1, b/1\}$. 则图 1(a)中 M_S 的三角矩阵如图 2 所示. 取 D_{12} 中的元素 $a/0$，则输入 a 是状态 s_1 和 s_2 的分离序列，取 D_{13} 中的元素 $b/1$，则输入 b 是状态 s_1 和 s_3 的分离序列，取 D_{23} 中的元素 $a/1$ 或 $b/1$，则输入 a 或 b 是状态 s_2 和 s_3 的分离序列. 因此最终获得的特征集 $W=\{a,b\}$. 为保证该方法的正确性，本文随后给出了 4 条定理及证明，其中证明部分的推理规则可参见文献[17].

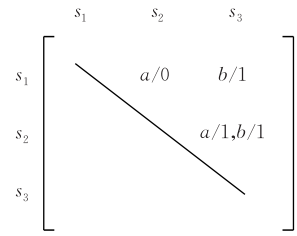


图 2　一个三角矩阵的例子

引理 1.　$\vdash \exists s_1, s_2: S \cdot (\varphi(s_1)=\varphi(s_2) \Rightarrow \forall x/y: \varphi(s_1) \cdot \lambda(s_1,x)=\lambda(s_2,x))$.

可由定义 8 证明引理 1，证明过程略.

引理 2.　$\vdash \exists s_1, s_2: S \cdot (\varphi(s_1)=\varphi(s_2) \Rightarrow \forall x/y \notin \varphi(s_1) \cdot \lambda(s_1,x)=\lambda(s_2,x))$.

证明.　如果 $x/y \notin \varphi(s_1)$，则由定义 8 可知，$\lambda(s_1,x)=-$ 或者存在 $a \in O$ 使得 $\lambda(s_1,x)=a$. 因为 $\varphi(s_1)=\varphi(s_2)$，则 $\lambda(s_2,x)=-$ 或者存在 $a \in O$ 使得 $\lambda(s_2,x)=a$. 若 $\lambda(s_1,x)\neq\lambda(s_2,x)$，则有 $x/a \in \varphi(s_2) \wedge x/a \notin \varphi(s_1)$ 或者 $x/a \in \varphi(s_1) \wedge x/a \notin \varphi(s_2)$. 则由集合的定义可知 $\varphi(s_1)\neq\varphi(s_2)$，与已知 $\varphi(s_1)=\varphi(s_2)$ 矛盾，因此，必有 $\lambda(s_1,x)=\lambda(s_2,x)$.　证毕.

引理 3.　$\vdash \forall t_1, t_2: S, \forall x: I \cdot (\lambda(t_1,x)=\lambda(t_2,x)) \Rightarrow \forall t_1, t_2: S \cdot (\forall x: I^* \cdot \lambda(t_1,x)=\lambda(t_2,x))$.

可由克林闭包的定义证明引理 3，证明过程略.

定理 1.　对最小 Mealy 机 M_S 而言，状态集合 S 中至少存在两个不同的状态 s 和 t，使得 M_S 的可识别函数 $\varphi(s_1)$ 不等于 $\varphi(s_2)$. 形式表示为 $\forall s_1, s_2: S \cdot (\exists x: I^* \cdot (\lambda(s_1,x)\neq\lambda(s_2,x))) \vdash \exists t_1, t_2: S \cdot (\varphi(t_1)\neq\varphi(t_2))$.

证明(反证法).

1. $\neg \exists t_1, t_2: S \cdot (\varphi(t_1)\neq\varphi(t_2))$　假设
2. $\forall t_1, t_2: S \cdot (\varphi(t_1)=\varphi(t_2))$　1, $\forall$-int
3. $\varphi(t_1)=\varphi(t_2)$　2, $\forall$-elim
4. $\forall x/y: \varphi(t_1) \cdot \lambda(t_1,x)=\lambda(t_2,x)$　3, 引理 1
5. $\lambda(t_1,x)=\lambda(t_2,x)$　4, $\forall$-elim
6. $\forall x/y \notin \varphi(t_1) \cdot \lambda(t_1,x)=\lambda(t_2,x)$　3, 引理 2
7. $\lambda(t_1,x)=\lambda(t_2,x)$　6, $\forall$-elim
8. $\forall x/y: I/O \cdot (x/y \in \varphi(t_1) \vee x/y \notin \varphi(t_1))$　定义 8
9. $\forall x/y: I/O \cdot \lambda(t_1,x)=\lambda(t_2,x)$　4～8, $\forall$-int
10. $\forall x: I \cdot \lambda(t_1,x)=\lambda(t_2,x)$　9 与 y 无关
11. $\forall t_1, t_2: S, \forall x: I \cdot \lambda(t_1,x)=\lambda(t_2,x)$　10, $\forall$-int
12. $\forall t_1, t_2: S \cdot (\forall x: I^* \cdot \lambda(t_1,x)=\lambda(t_2,x))$　11, 引理 3
13. $\forall s_1, s_2: S \cdot (\exists x: I^* \cdot \lambda(s_1,x)\neq\lambda(s_2,x))$　前提

14. $\exists t_1,t_2:S\bullet(\varphi(t_1)\neq\varphi(t_2))$ 12 和 13 矛盾

证毕.

定理 1 保证了三角矩阵不全为空.

定理 2. 对于 S 中的任意两个状态 s_1 和 s_2,如果 $\varphi(s_1)$不等于 $\varphi(s_2)$,那么差集 $D_{12}=\varphi(s_1)-\varphi(s_2)$ 或 $D_{12}=\varphi(s_2)-\varphi(s_1)$能够识别状态 s_1 和 s_2. 形式表示为

$$\vdash\forall s_1,s_2:S\bullet(\varphi(s_1)\neq\varphi(s_2)\Rightarrow\forall x/y:D_{12}\bullet\lambda(s_1,x)\neq\lambda(s_2,x)).$$

证明.

1. $\varphi(s_1)\neq\varphi(s_2)$ 假设
2. $D_{12}\neq\varnothing$ 1,定义 9
3. $D_{12}=\varphi(s_1)-\varphi(s_2)\vee D_{12}=\varphi(s_2)-\varphi(s_1)$ 2,定义 9
4. $D_{12}=\varphi(s_1)-\varphi(s_2)$ 3,假设
5. $\forall x/y:D_{12}\bullet(x/y\in\varphi(s_1)\wedge x/y\notin\varphi(s_2))$ $\forall$-int,定义 9
6. $x/y\in\varphi(s_1)\wedge x/y\notin\varphi(s_2)$ $\forall$-elim
7. $\forall x/y\in\varphi(s_1)\bullet(\lambda(s_1,x)=y)$ 定义 8
8. $\lambda(s_1,x)=y$ 7,$\forall$-elim
9. $\forall x/y\notin\varphi(s_2)\bullet(\lambda(s_2,x)\neq y)$定义 8
10. $\lambda(s_2,x)\neq y$ 9,$\forall$-elim
11. $\lambda(s_1,x)\neq\lambda(s_2,x)$ 8,10,替换
12. $\forall x/y:D_{12}\bullet\lambda(s_1,x)\neq\lambda(s_2,x)$ 11,$\forall$-int,定义 9
13. $D_{12}=\varphi(s_2)-\varphi(s_1)$ 3,假设
14. $\forall x/y:D_{12}\bullet(\lambda(s_1,x)\neq\lambda(s_2,x))$ 证明类似 4~12
15. $\varphi(s_1)\neq\varphi(s_2)\Rightarrow\forall x/y:D_{12}\bullet\lambda(s_1,x)\neq\lambda(s_2,x)$ 1~14,$\Rightarrow$-int
16. $\forall s_1,s_2:S\bullet(\varphi(s_1)\neq\varphi(s_2)\Rightarrow\forall x/y:D_{12}\bullet\lambda(s_1,x)\neq\lambda(s_2,x))$ 15,$\forall$-int

由定理 2 可知,在三角矩阵内,若 $D_{ij}\neq\varnothing$,则对于任意 $x/y\in D_{ij}$,输入 x 一定能识别状态 s_i 和 s_j.

证毕.

引理 4. $\vdash\exists s_1,s_2:S,\exists x,x_1:I\bullet(\delta(s_1,x)=s_2\Rightarrow\lambda(s_1,x\bullet x_1)=\lambda(s_1,x)\bullet\lambda(s_2,x_1))$.

可由定义 1 和序列的连接操作证明引理 4,证明过程略.

引理 5. $\vdash\exists a,b,c,d:O\bullet(a=b\wedge c\neq d\Rightarrow a\bullet c\neq b\bullet d)$,或者$\vdash\exists a,b,c,d:O\bullet(a\neq b\wedge c=d\Rightarrow a\bullet c\neq b\bullet d)$,或者$\vdash\exists a,b,c,d:O\bullet(a\neq b\wedge\#c=\#d\Rightarrow a\bullet c\neq b\bullet d)$.

可由序列的连接操作证明引理 5,证明过程略.

定理 3. 对识别函数 φ 而言,如果 $\varphi(s_1)$等于 $\varphi(s_2)$,且I中存在x和y,使得$\delta(s_1,x)=s_i$,$\delta(s_2,y)=s_j$及$D_{ij}\neq\varnothing$,那么对于 D_{ij} 中任意 x_1/y_1,输入序列 $x\bullet x_1$ 都能识别状态 s_1 和 s_2. 形式表示为 $\exists s_1,s_2:S\bullet(\varphi(s_1)=\varphi(s_2))\vdash\exists x,y:I,\exists s_i,s_j:S\bullet(\delta(s_1,x)=s_i\wedge\delta(s_2,y)=s_j\wedge D_{ij}\neq\varnothing)\Rightarrow\forall x_1/y_1:D_{ij}\bullet(\lambda(s_1,x\bullet x_1)\neq\lambda(s_2,x\bullet x_1))$.

证明.

1. $\exists s_1,s_2:S\bullet(\varphi(s_1)=\varphi(s_2))$ 前提
2. $\varphi(s_1)=\varphi(s_2)$ 1,$\exists$-elim
3. $\forall x/y\in\varphi(s_1)\bullet(\lambda(s_1,x)=\lambda(s_2,x))$ 2,引理 1
4. $\lambda(s_1,x)=\lambda(s_2,x)$ 3,$\forall$-elim
5. $\exists x,y:I,\exists s_i,s_j:S\bullet(\delta(s_1,x)=s_i\wedge\delta(s_2,y)=s_j\wedge D_{ij}\neq\varnothing)$ 假设
6. $\delta(s_1,x)=s_i\wedge\delta(s_2,y)=s_j\wedge D_{ij}\neq\varnothing$ 5,$\exists$-elim
7. $D_{ij}\neq\varnothing$ 6,$\wedge$-elim
8. $D_{ij}=\varphi(s_i)-\varphi(s_j)\vee D_{ij}=\varphi(s_j)-\varphi(s_i)$ 定义 9
9. $\varphi(s_i)\neq\varphi(s_j)$ 7,8,定义 9
10. $\forall x_1/y_1:D_{ij}\bullet\lambda(s_i,x_1)\neq\lambda(s_j,x_1)$ 9,定理 2
11. $\lambda(s_i,x_1)\neq\lambda(s_j,x_1)$ 10,$\forall$-elim
12. $\delta(s_1,x)=s_i$ 6,$\wedge$-elim
13. $\delta(s_2,y)=s_j$ 6,$\wedge$-elim
14. $\lambda(s_1,x\bullet x_1)=\lambda(s_1,x)\bullet\lambda(s_i,x_1)$ 12,引理 4
15. $\lambda(s_2,x\bullet x_1)=\lambda(s_2,x)\bullet\lambda(s_j,x_1)$ 13,引理 4
16. $\lambda(s_1,x)\bullet\lambda(s_i,x_1)\neq\lambda(s_2,x)\bullet\lambda(s_j,x_1)$ 4,11,引理 5
17. $\lambda(s_1,x\bullet x_1)\neq\lambda(s_2,x)\bullet\lambda(s_j,x_1)$ 14,16,替换
18. $\lambda(s_1,x\bullet x_1)\neq\lambda(s_2,x\bullet x_1)$ 15,17,替换
19. $\forall x_1/y_1:D_{ij}\bullet(\lambda(s_1,x\bullet x_1)\neq\lambda(s_2,x\bullet x_1))$ 18,$\forall$-int
20. $\exists x,y:I,\exists s_i,s_j:S\bullet(\delta(s_1,x)=s_i\wedge\delta(s_2,y)=s_j\wedge D_{ij}\neq\varnothing)\Rightarrow\forall x_1/y_1:D_{ij}(\lambda(s_1,x\bullet x_1)\neq\lambda(s_2,x\bullet x_1))$ 5~19,$\Rightarrow$-int

证毕.

由定理 3 可知,若 $D_{12}=\varnothing$,则能利用 s_1 和 s_2 的相邻状态 s_i 和 s_j 的 $D_{ij}(D_{ij}\neq\varnothing)$来获得 s_1 和 s_2 的分离序列. 若 $D_{ij}=\varnothing$,由定理 1 可知,可识别函数的三角矩阵中一定存在一个 $D_{pq}\neq\varnothing$,又由于最小 Mealy 机一定存在特征集[6],因此可由 D_{pq} 来获得 s_1 和 s_2 的分离序列.

引理 6. $\vdash\exists s:S,\exists y,a:I^*,\exists b:O^*\bullet(\lambda(s,y\bullet a)=\lambda(s,y)\bullet b\wedge\#b=\#(y\bullet a)-\#y)$.

可由定义 1 和序列的连接操作证明引理 6,证明过程略.

定理 4. 对 M_s 的特征集 W 而言,若存在分离序列 x 和 y 满足 $x=y\bullet a$,其中 a 是一个输入序列,

那么集合 $W=W-\{y\}$ 同样是 M_S 的一个特征集. 形式表示为 $\vdash \forall s_1,s_2:S,\exists x,y:W,\exists a:I^*\cdot(\lambda(s_1,y)\neq\lambda(s_2,y)\wedge x=y\cdot a)\Rightarrow\lambda(s_1,x)\neq\lambda(s_2,x)$.

证明.

1. $\forall s_1,s_2:S,\exists x,y:W,\exists a:I^*\cdot(\lambda(s_1,y)\neq\lambda(s_2,y)\wedge x=y\cdot a)$　假设
2. $\exists x,y:W,\exists a:I^*\cdot(\lambda(s_1,y)\neq\lambda(s_2,y)\wedge x=y\cdot a)$　1,∀-elim
3. $\lambda(s_1,y)\neq\lambda(s_2,y)\wedge x=y\cdot a$　2,∃-elim
4. $\lambda(s_1,y)\neq\lambda(s_2,y)$　3,∧-elim
5. $x=y\cdot a$　3,∧-elim
6. $\lambda(s_1,x)=\lambda(s_1,x)$　自反性
7. $\lambda(s_1,x)=\lambda(s_1,y\cdot a)$　5,6,替换
8. $\lambda(s_2,x)=\lambda(s_2,x)$　自反性
9. $\lambda(s_2,x)=\lambda(s_2,y\cdot a)$　5,8,替换
10. $\exists s_1:S,\exists y,a\in I^*,\exists b_1:O^*\cdot(\lambda(s_1,y\cdot a)=\lambda(s_1,y)\cdot b_1\wedge\#b_1=\#(y\cdot a)-\#y)$　引理 6
11. $\lambda(s_1,y\cdot a)=\lambda(s_1,y)\cdot b_1\wedge\#b_1=\#(y\cdot a)-\#y$　10,∃-elim
12. $\lambda(s_1,y\cdot a)=\lambda(s_1,y)\cdot b_1$　11,∧-elim
13. $\#b_1=\#(y\cdot a)-\#y$　11,∧-elim
14. $\exists s_2:S,\exists y,a\in I^*,\exists b_2:O^*\cdot(\lambda(s_2,y\cdot a)=\lambda(s_2,y)\cdot b_2\wedge\#b_1=\#y\cdot a-\#y)$　引理 6
15. $\lambda(s_2,y\cdot a)=\lambda(s_2,y)\cdot b_2\wedge\#b_2=\#y\cdot a-\#y$　14,∃-elim
16. $\lambda(s_2,y\cdot a)=\lambda(s_2,y)\cdot b_2$　15,∧-elim
17. $\#b_2=\#(y\cdot a)-\#y$　15,∧-elim
18. $\#b_1=\#b_2$　13,17,替换
19. $\lambda(s_1,y)\cdot b_1\neq\lambda(s_2,y)\cdot b_2$　4,18,引理 5
20. $\lambda(s_1,y\cdot a)\neq\lambda(s_2,y)\cdot b_2$　12,19,替换
21. $\lambda(s_1,y\cdot a)\neq\lambda(s_2,y\cdot a)$　16,20,替换
22. $\lambda(s_1,x)\neq\lambda(s_2,x)$　5,21,替换
23. $\forall s_1,s_2:S,\exists x,y:W,\exists a:I\cdot(\lambda(s_1,y)\neq\lambda(s_2,y)\wedge x=y\cdot a)\Rightarrow\lambda(s_1,x)\neq\lambda(s_2,x)$　1～22,⇒-int

证毕.

依据定理 4,可以对特征集 W 进行冗余约简. 基于定理 1～4,本文设计了特征集 W 的构造算法 WSA. 算法 WSA 的时间复杂度为 $O(n^2)$.

算法 WSA.　特征集 W 的构造.

输入：Mealy 机 M_s

输出：M_s 的特征集 W

1. 初始化 $W=\varnothing$；
2. 构造 M_s 的一个可识别函数 φ；
3. For ($i=1$; $i<n$; $i++$) //构造 D_{ij}.
4.　For ($j=i+1$; $j<=n$; $j++$)
5.　　If $\varphi(s_i)\not\subset\varphi(s_j)\neq\varnothing$ then $D_{ij}=\varphi(s_i)-\varphi(s_j)$;
6.　　Else $D_{ij}=\varphi(s_j)-\varphi(s_i)$;
7.　End for
8. End for
9. 对于每个 D_{ij}，如果 $D_{ij}=\varnothing$，则依据定理 3，从 x/y 中得到分离序列 x，令 $D_{ij}=\{x/y\}$ 和 $W=W\cup\{x\}$；否则，从 D_{ij} 中取 x/y，令 $W=W\cup\{x\}$；
10. 对于 W 中任意两条序列 x 和 y，如果存在 $x=y\cdot a$，其中 a 为 W 中另外一条序列，则 $W=W-\{y\}$；
11. Return W.

以图 1(a)中的 M_S 为例，依据算法 TBFS 产生一棵测试生成树. 为简化测试生成，本节并不考虑 M_S 中的迁移约束. 图 3(a)是 M_S 的一个错误应用，存在一个输出错误 $\underline{b/1}$. 由算法 TBFS 获得的测试生成树如图 3(b)所示，由这棵树得到输入序列覆盖集 $P=\{\varepsilon,a,b,ba,bb,bba,bbb\}$，其中 ε 表示空输入. 依据算法 WSA 得到特征集 $W=\{a,b\}$. 依据 W 方法，使用输入序列 $r\cdot P\cdot W$ 和输出序列 $\lambda(s_1,r\cdot P\cdot W)$ 建造表 2，其中的 r 表示重置消息(假设 7).

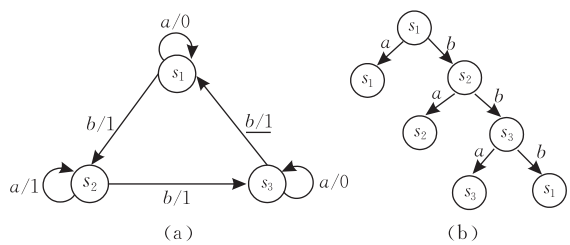


图　3

对于图 1(a)中 M_1 的迁移错误 $t'=(s_3,a/0,s_2)$，测试序列 $rbbaa$ 的实际输出为 1101，而预期输出为 1100，因此迁移错误 t' 能被发现. 对于图 2(a)中 M_1 的输出错误 $\underline{b/1}$，测试序列 $rbbb$，$rbbba$，$rbbbb$ 的实际输出为 111，1110，1111，而预期输出为 110，1100，1101，因此这 3 条测试序列都能够发现输出错误 $\underline{b/1}$. 在实验[4]中，W 方法既表现出很强的错误探测能力，又包含不可容忍的测试序列总长度，导致测试成本的急剧增加. 如表 2 所示，W 方法的测试序列总长度为 52.

表 2　W 方法的测试序列

	测试序列													
P	ε		a		b		ba		bb		bba		bbb	
$r\cdot P\cdot W$	ra	rb	raa	rab	rba	rbb	$rbaa$	$rbab$	$rbba$	$rbbb$	$rbbaa$	$rbbab$	$rbbba$	$rbbbb$
输出	0	1	00	11	11	11	111	111	110	110	1100	1100	1100	1101

3.3 Wp 方法

Wp 方法[2]也称部分 W 方法,其主要优点是约简了 W 方法产生测试用例的总长度.该方法将测试分为两个阶段:第 1 个阶段测试 M_S 中的每一个状态是否在 M_I 中;第 2 个阶段检测所有未在第 1 个阶段中被检测的迁移.为实现第 1 个阶段,Wp 方法使用一个状态覆盖集来替代输入序列覆盖集 P.相比输入序列覆盖集 P,状态覆盖集更小.

定义 10(状态覆盖集). 状态覆盖集 Q 是由输入序列组成,且对于任意的 $s\in S$,都存在一个输入序列 $x\in Q$,使得自动机迁移到 s,即 $\delta(s_1,x)=s$.形式描述为 $\forall s:S\cdot(\exists x:Q\cdot\delta(s_1,x)=s)$.

Wp 方法的第 1 阶段的输入序列为 $r\cdot Q\cdot W$,输出序列为 $\lambda(s_1,r\cdot P\cdot W)$.在第 2 个阶段,Wp 方法使用一个识别集 W_i(identification set)[6]来区分 s_i 与其它状态,而不是 W 方法中的特征集 W.

定义 11(识别集). 状态 s_i 的一个识别集 W_i 是由输入序列构成,对于 $\forall s_j:S,i\neq j$,集合 W_i 中都存在一个输入序列 x,使得 $\lambda(s_i,x)\neq\lambda(s_j,x)$,且 W_i 的任意一个真子集均不满足这个特性.

由定义 11 可知,每个状态的识别集都是特征集 W 的子集,且识别集的任意子集都不是该状态的识别集.求解特征集中的最小识别集是一个 NP 难问题.本文采用启发式方法,每次都从特征集 W 中选取一条输入序列 x,使得 x 能够识别最多状态,且这些状态未被已选出序列识别.依据这种思想,本文设计了识别集的生成算法 WPSA.

假定 $|W|=p$,$|S|=n$,则算法 WPSA 的时间复杂度为 $O(pn^2)$.在第 2 阶段,为获取第 1 个阶段中未被检测的迁移,需要构造集合 $R=P-Q$,其中 P 是迁移覆盖集,Q 是状态覆盖集.则第 2 个阶段的输入序列为 $r\cdot R\cdot W_i$,输出序列为 $\lambda(s_1,r\cdot R\cdot W_i)$.第 2 阶段的目的是为了测试第 1 阶段未被测试的迁移,以保证 Wp 方法的错误探测能力.对于 Wp 方法与 W 方法的测试等效性,可以参阅文献[2]中的证明.

算法 WPSA. 识别集的构造算法.

输入:特征集 W 和 FSM 的状态集合 S,$|S|=n$

输出:识别集 $W_1,W_2,\cdots,W_n$

1. 初始化 $W_1=W_2=\cdots=W_n=\varnothing$;
2. for each $s_i\in S$
3. $tmp\_S=S-\{s_i\}$;
4. $tmp\_W=W$;
5. While($tmp\_S\neq\varnothing$)
6. 从 tmp_W 中选出一个 x,使得 x 能将 s_i 与 tmp_S 中最多的状态区分开来,$W_i=W_i\cup\{x\}$,从 tmp_S 中删除那些已被 x 区分的状态;
7. End while
8. End for
9. Return $W_1,W_2,\cdots,W_n$.

随后本文以图 1(a)中的 M_S 为例,演示 Wp 方法,并分析 Wp 方法的错误探测能力和测试序列的总长度.依据算法 TBFS,构造一棵测试生成树,如图 3(b),则这颗树的状态覆盖集 $Q=\{\varepsilon,b,bb\}$.由于图 1(a)中 FSM 的特征集 $W=\{a,b\}$,则第 1 个阶段产生的测试序列显示在表 3 中.

表 3 第 1 阶段的测试序列

	测试序列					
Q	ε		b		bb	
$r\cdot Q\cdot W$	ra	rb	rba	rbb	$rbba$	$rbbb$
输出	0	1	11	11	110	110

依据算法 WPSA,由于 a 能区分状态 s_2 与其它状态,b 能区分状态 s_3 与其它状态,则 $W_2=\{a\}$ 及 $W_3=\{b\}$.而 s_1 不能被 a 或 b 单独识别,则 $W_1=W_2\cup W_3=\{a,b\}$,即 s_1 能被 a 和 b 区分其它状态.$R=P-Q=\{\varepsilon,a,b,ba,bb,bba,bbb\}-\{\varepsilon,b,bb\}=\{a,ba,bba,bbb\}$.第 2 阶段产生的测试序列显示在表 4 中.

表 4 第 2 阶段的测试序列

	测试序列					
R	a		ba	bba	bbb	
$r\cdot R\cdot W_i$	raa	rab	$rbaa$	$rbbab$	$rbbba$	$rbbbb$
输出	00	01	111	1100	1100	1101

由表 3 和表 4 可知,测试序列的总长度是 43.图 1(a)中 M_I 的迁移错误 $t'=(s_3,a/0,s_2)$ 能被测试序列 $rbbab$ 发现,而图 3(a)中 M_I 的输出错误 $\underline{b/1}$ 能被测试序列 $rbbb,rbbba,rbbbb$ 发现.

3.4 UIO 方法

在 Wp 方法中,如果识别集 W_i 仅为一条测试序列,那么它被称为状态签名(state signature)[18]或 UIO(Unique Input/Output)序列[19],且能唯一识别状态 s_i.UIO 方法首先采用 P 集测试每一条从状态 s_i 到状态 s_j 的迁移,再采用 s_j 的 UIO 序列检测终态是否是 s_j.因此该方法同样不需要状态位信息.

定义 12(UIO 序列). 在 Mealy 机中,状态 s 的 UIO 序列是一条输入序列 x,满足任意其它状态 $t\in S$,都有 $\lambda(s,x)\neq\lambda(t,x)$.

UIO 方法采用状态 s 的 UIO 序列替代 W 方法中的特征集 W,即测试输入为 $r\cdot P\cdot UIO_i$,测试输出为 $\lambda(s_1,r\cdot P\cdot UIO_i)$.由于 UIO_i 仅包含一条序列,因此该 UIO 方法的测试序列总长度总是比 W

方法的要短.但 UIO 方法并不能保证揭示应用中的任何错误[20].例如,若一个错误应用的状态 s' 与状态 s 有相同的 UIO 序列,因为没有状态位的信息,因此测试的输出将无法辨别状态 s' 与状态 s.为此 Vuong 等人[20]提出了 UIOv 方法,该方法建造测试组包括3 个阶段:

(1) Uv 过程. 获得每个状态 s 的 UIO 序列,即对 M_S中的每个状态 s 都有一个条区分序列 x,使得 x 能将状态 s 与其它状态区分出来.设状态覆盖集为 Q,则 Uv 过程的输入序列为 $r \cdot Q \cdot UIO_i$,输出序列为 $\lambda(s_1, r \cdot Q \cdot UIO_i)$.

(2) ¬Uv 过程. 获得每个状态 s 的¬UIO 序列,即从 UIO 集合中除去能够区分 s 与其它状态的序列.如 $UIO=\{ab,a,b\}$,其中 ab 和 a 能将状态 s 与其它状态区分出来,则 $UIO-\{ab,a\}=\{b\}$,那么 s 的¬UIO 序列为 b. ¬Uv 过程的输入序列为 $r \cdot Q \cdot \neg UIO_i$,输出序列为 $\lambda(s_1, r \cdot Q \cdot \neg UIO_i)$.

(3) 迁移测试阶段:获得 Uv 过程和¬Uv 过程中没有访问到的迁移,即执行 $R=P-Q$.再对集合 R 中序列到达的最后一个状态应用相应的 UIO_i,最终的输入序列为 $r \cdot R \cdot UIO_i$,输出序列为 $\lambda(s_1, r \cdot R \cdot UIO_i)$.

UIO 方法可以被认为是一类特殊的 Wp 方法,其中特征集 W 是所有 UIO 序列的并,Wp 方法的第 1 阶段为 UIOv 方法的 Uv 过程和¬Uv 过程,Wp 方法的第 2 阶段为 UIOv 方法的迁移测试阶段. UIOv 方法需要构造 Q 集、P 集、UIO 集和 R 集,其中 Q 集、P 集和 R 集的构造算法已在前几节中被描述,因此本节主要描述 UIO 集的构造算法.直接构造所有状态的 UIO 集合是一个 NP 难问题[11,21].文献[22]给出了一个构造所有 UIO 序列的算法,但该算法运用的推理规则会造成 UIO 序列长度的增加[11],最终导致测试成本的增加.

本文提出了一种启发式方法求解某个状态的一条 UIO 序列.该方法的求解步骤:(1)若状态 s 的识别集 $W_i=\{a,b\}$,则获取状态 $s_1=\delta(s,a)$ 和 $s_2=\delta(s,b)$;(2)分别以状态 s_1 和 s_2 为根结点,遍历规格说明 M_S生成两棵广度优先生成树,且树中最后一个叶子结点的状态为 s;(3)分别从根为 s_1 和 s_2 的两棵树中获取由根结点到最后一个叶子结点的序列 x 和 y,若序列 x 的长度小于序列 y,则状态 s 的 $UIO=a \cdot x \cdot b$,否则状态 s 的 $UIO=b \cdot y \cdot a$.依据这种方法本文给出了算法 UIOS.

由于 $|W_i| \le n-1$,因此算法 UIOS 的最坏时间复杂度为 $O(n^4)$. UIO 方法必须满足 M_S是强连通的(假设 3),否则步 5 中状态迁移函数 $\delta(s_i,x)$可能为空,则存在状态没有相应的 UIO 序列.过去对 UIO 方法的研究[6,11,20-22]均是以通信协议为研究对象,而通信协议中的任意两个端口之间均能相互通信,因此协议模型 M_S中的每个状态都存在相应的 UIO 序列.

算法 UIOS.　构造 UIO 集.

输入:规格说明 M_s和识别集 $W_1, W_2, \cdots, W_n$;

输出:集合 $UIO_1, UIO_2, \cdots, UIO_n$;

1. 初始化 $UIO_1=UIO_2=\cdots=UIO_n=\varnothing$, $Tmp=\varnothing$, $Sp=\varnothing$, $seq=""$;
2. for each W_i　//W_i是状态 s_i的识别集;
3. if ($|W_i|=1$), then $UIO_i=W_i$;
4. else
5. For each $x \in W_i$,
6. $s=\delta(s_i,x)$和 $Tmp=Tmp \cup \{s\}$;
7. End for
8. For each $s \in Tmp$
9. 遍历 M_s,构建一棵根为 s 及终止结点为 s_i 的树;从该树中获得从根 s 到叶子 s_i的输入序列 x;令 $Sp=Sp \cup \{x\}$;
10. End for
11. 按递减次序对 Sp 中的所有序列排序,再删除 Sp 中最短的一条序列;
12. For each $x \in Sp$
13. 获得包含序列 x 初始结点 s,从 W_i中取序列 y 使得 $s=\delta(s_i,y)$;令 $seq=seq \cdot y \cdot x$;
14. End for
15. 从 Sp 中获得最长的一条序列 p,如果 $seq=q \cdot p$,那么 $seq=q$ 和 $UIO_i=seq$;
16. End if
17. End for
18. Return $UIO_1, UIO_2, \cdots, UIO_n$.

随后本文以图 1(a)中的 M_S为例,演示 UIO 方法及其改进的 UIOv 方法,并分析这两种方法的错误探测能力和测试用例的总长度. UIO 方法:(1)用 P 集测试所有迁移;(2)用 UIO 序列验证 P 集测试迁移的终态是否正确.依据算法 UIOS,得到 $UIO_1=ab/01$, $UIO_2=a/1$, $UIO_3=b/0$.具体的构造 UIO 方法的过程如表 5.

表 5　UIO 方法的测试序列

	序列构造						
P	ε	a	b	ba	bb	bba	bbb
$r \cdot P \cdot UIO_i$	rab	$raab$	rba	$rbaa$	$rbbb$	$rbbab$	$rbbbb$
输出	01	001	11	111	110	1100	1101

对于图 1(a)中 M_1的迁移错误 $t'=(s_3, a/0, s_2)$, UIO 方法可以通过测序列 $rbbab$ 检测而发现;对于

图 3(a)中 M_1的输出错误$\underline{b/1}$,可以通过序列 *rbbb* 和 *rbbbb* 发现.由表 5 可知,UIO 方法的测试序列总长度为 28.

在 UIOv 方法中,由于集合 $UIO=UIO_1 \cup UIO_2 \cup UIO_3=\{ab/01, a/1, b/0\}$,则 $\neg UIO_1=UIO-UIO_1=\{a/1,b/0\}$,$\neg UIO_2=UIO-UIO_2=\{b/0, ab/01\}$,$\neg UIO_3=UIO-UIO_3=\{a/1, ab/01\}$.$R=P-Q=\{a,ba,bba,bbb\}$,集合 R 与状态之间的关系可以查看表 3.UIOv 方法的构造过程如表 6～8.

表 6 Uv 过程

	序列构造		
Q	ε	b	bb
$r \cdot Q \cdot UIO_i$	*rab*	*rba*	*rbbb*
输出	01	11	110

表 7 ¬Uv 过程

	序列构造					
$r \cdot Q \cdot \neg UIO_i$	*ra*	*rb*	*rbb*	*rbab*	*rbba*	*rbbab*
输出	0	1	11	1100	110	1100

表 8 迁移测试阶段

	序列构造			
UIO_i	UIO_1	UIO_2	UIO_1	UIO_3
$r \cdot R \cdot UIO_i$	*raab*	*rbaa*	*rbbbab*	*rbbab*
输出	001	111	11001	1100

同样 UIOv 方法中的测试序列 *rbbab* 能够发现图 1(a)中 M_1的迁移错误,而输入序列 *rbbb* 能发现图 3(a)中 M_1的输出错误.由表 6～8 可知,测试序列的总长度为 49.但 UIO 方法和 UIOv 方法都必须满足模型 M_S是强连通的.

3.5 DS 方法

在某些情况下,一条 UIO 序列能够区分所有的状态,那么这条序列被称为区分序列(Distinguishing Sequence,DS)[4,6].若特征集 W 仅仅包含一条输入序列时,则 DS 方法可以看成是 W 方法的一个特例,但并非是所有的 Mealy 机都存在区分序列[6,20].因此,若 Mealy 机存在区分序列 x,则 DS 方法的输入序列为 $r \cdot P \cdot x$,输出序列为 $\lambda(s_1, r \cdot P \cdot x)$.为构造区分序列 x,文献[21]提出构造分裂树(splitting tree)获取区分序列的方法,该方法的演示如图 4 所示.

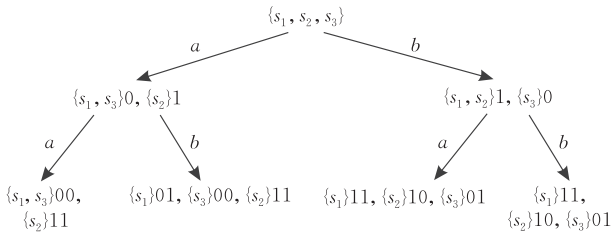


图 4 图 1(a)中 M_S的一棵分裂树

图 4 显示了图 1(a)中 M_S的分裂树,这棵树的构造过程采用了集合的划分方式.当对集合$\{s_1,s_2,s_3\}$输入 a 时,依据输出函数 $\lambda(s,x)$的不同,将集合$\{s_1,s_2,s_3\}$划分为$\{s_1,s_3\}0$ 和$\{s_2\}1$,其中 0 和 1 表示集合内元素的输出.对$\{s_1,s_3\}0$ 和$\{s_2\}1$ 输入 a 时,依据输出函数 $\lambda(s,x)$的不同,得到$\{s_1,s_3\}00$ 和$\{s_2\}11$,由于集合$\{s_1,s_3\}$未被划分,因此这条分枝不必继续输入 a 来划分$\{s_1,s_3\}0$ 和$\{s_2\}1$;再对$\{s_1,s_3\}0$和$\{s_2\}1$ 输入 b,依据输出函数 $\lambda(s,x)$的不同,集合$\{s_1,s_3\}0$ 被划分为$\{s_1\}01$ 和$\{s_3\}00$,$\{s_2\}1$ 得到$\{s_2\}11$.由于每个集合仅有一个元素,因此序列 ab 是一条区分序列.同理,采用这种方式构造继承树中的其它分枝.由于图 4 可知,输入序列 ab,ba 和 bb 均是图 1(a)中 M_S的区分序列.

构造 Mealy 机的分裂树是一个 NP 难问题[6,21],且 Mealy 机并不一定存在一条区分序列,为此需要判断 Mealy 是否存在区分序列.文献[21]给出了一个检测区分序列是否存在的算法,在本文中命名为算法 CDS.与原算法中的自然语言描述不同,算法 CDS 采用了形式语言描述.

算法 CDS. 检测 Mealy 是否存在区分序列.

输入:规格说明 M_S和状态集 S

输出:布尔值,true 表示存在区分序列,false 表示不存在区分序列

1. 初始化 $\pi=\{\{S\}\}$;
2. while $(\exists B\in\pi \wedge \exists a\in I, \exists s,t\in B \cdot ((\lambda(s,x)\neq\lambda(t,x) \vee \exists B_1,B_2\in\pi \cdot (\delta(s,x)\in B_1 \wedge \delta(t,x)\in B_2)))$
3. $\pi'=\pi$;
4. $\pi=\pi-B$; //从 π 中删除 B.
5. $\pi=\pi \cup Partition(B,\pi')$;
6. End while
7. 如果 $\exists B\in\pi \cdot \#B>1$,则返回 false;否则返回 true.

算法 CDS 中的 $Partition(B,\pi')$表示对 B 中状态的划分,对不同的状态输入相同序列 $a\in I$,将具有相同输出的状态移动到 π'中不同的集合内.对于算法 CDS 的正确性,文献[21]中给出了详细的证明.算法 CDS 不能在多项式时间内实现.文献[6,21]给出了构造一棵完全分裂树(splitting tree)算法 SPT.对于该算法的详细描述可以查阅文献[6,21],限于篇幅原因,本文不再赘述.当且仅当 Mealy 机存在一个区分序列时,算法 SPT 才成立.同样算法 SPT 也不能在多项式时间内完成.

依据算法 SPT,将图 1(a)中 M_S转换成一棵分裂树,如图 4 所示.取区分序列 ba,则 M_S的输出函数 $\lambda(s_1,ba)=11$,$\lambda(s_2,ba)=10$,$\lambda(s_3,ba)=01$.DS 方法的构造过程如表 9 所示.

表 9　DS 方法的构造表

P	序列构造						
	ε	*a*	*b*	*ba*	*bb*	*bba*	*bbb*
$r \cdot P \cdot x$	*rba*	*raba*	*rbba*	*rbaba*	*rbbba*	*rbbaba*	*rbbbba*
输出	11	011	110	1110	1100	11000	11011

由表 9 可知，DS 方法的测试序列总长度为 33. 图 1(a)中的迁移错误 $t'=(s_3,a/0,s_2)$可以通过测试序列 *rbbaba* 发现，而图 2(a)的输出错误$\underline{b/1}$能通过测试序列 *rbbba* 发现. DS 方法的缺点是并非所有的 Mealy 机都存在区分序列，另外测试序列的总长度与选取的区分序列有关.

3.6　T 方法

软件测试中可能存在一类额外迁移错误，即描述需求的规格说明中并没有这些迁移，而实际应用中却存在这些额外的迁移. 在某些情况下，这些额外的迁移会导致系统的崩溃. 例如 1999 年 12 月 3 日，美国航天局火星极地登陆者号探测器飞船在试图登陆火星表面时失踪. 错误修正委员会观测到故障，认定出现失误动作的原因极有可能是某一个数据被意外更改. 这表明该系统中至少存在一条额外迁移，在正常情况下，这条额外迁移并不会被执行，但在特定情况下，这条额外迁移的执行会导致系统运行出错.

然而本文前几节中讨论的基于 FSM 的测试方法均不能探测到这类错误，因为上述方法都是由规格说明 M_S 导出测试用例，而额外迁移仅仅出现在实际应用 M_I 中，因此由规格说明产生的测试序列并不能激活实际应用中的额外迁移. 虽然额外迁移的错误已经超出了基于 FSM 的测试能力，但仍然存在少量基于 FSM 的测试方法能够探测这类错误.

图 5 为图 1(a)中 M_S 的一个错误应用. 该错误应用包含了两个额外的迁移 $t_7=(s_3,c/0,s_2)$和 $t_8=(s_3,d/0,s_4)$. 为发现这两条错误迁移，必须寻找一种能够激活额外迁移的方法. T 方法[4]又称为随机测试生成方法，该方法随机产生测试输入，直到 M_S 中的所有迁移都被至少覆盖一次. 传统 T 方法直接由规格说明随机产出覆盖 M_S 的测试序列，因此同样也不能发现额外迁移错误.

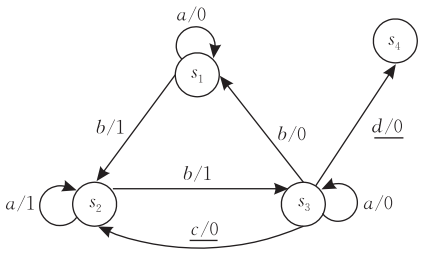


图 5　一个包含额外迁移的错误应用 M_I

本文改进了 T 方法，将随机产生的测试输入直接应用到 Mealy 机 M_I 中，直到由 M_I 产生的迁移覆盖 M_S 中的所有迁移一次. 由于是随机产生测试输入，因此可能在一定的概率下探测到 M_I 中的额外迁移.

为验证这个观点，我们以图 1(a)中的 M_S 为例，利用计算机模拟了随机测试序列的生成. 为避免死循环，在计算机模拟时，若某个状态输入任何符号时都不会发生后续迁移，即 M_I 并非强连通的，则无论 M_S 中所有迁移是否被覆盖，程序都结束. 实验中，假定的测试输入集 $I=\{a,b,c,d\}$，实验随机产生了十条测试输入和相应的测试输出，如表 10 所示. 一些重要的结论可以从表 10 中得出.

表 10　改进 T 方法的计算机模拟过程

实验	随机次数	M_I 的测试输入	输出	长度	覆盖所有迁移	$t_7=(s_3,c/0,s_2)$	$t_8=(s_3,d/0,s_4)$
1	7	*rbabcbd*	111010	7	N	Y	Y
2	27	*rbabcbcbbbbcbabbbad*	111010101101001100	19	N	Y	Y
3	74	*rbbcabcbcabbbaabaab baabcbcbd*	110110101101111000 111101010	28	N	Y	Y
4	4	*rbbd*	110	4	N	N	Y
5	15	*rabbaacabd*	011000110	10	N	Y	Y
6	7	*rbbbbbd*	110110	7	N	Y	N
7	54	*rbbbbabcabcbaabbbca bacababbaaababa*	110111011010001101 100110011111000	34	Y	Y	N
8	4	*rbabd*	1110	5	N	N	Y
9	14	*rbbaaaabbbaad*	110000011000	13	N	N	Y
10	28	*rabbbbbacbbabbd*	01101100100110	15	N	Y	Y

(1) 在实验中，改进 T 方法产生测试输入的内容和长度是随机的，这说明改进 T 方法是不稳定的.

(2) 仅在实验 7 中，改进 T 方法产生的测试输入覆盖了 M_S 中的所有迁移，因此改进 T 方法并不一定能保证探测到 M_I 中的迁移错误和输出错误.

(3) 在实验 1,2,3 和 10 中，改进 T 方法能够同时探测到额外迁移 t_7 和 t_8；在实验 1,2,3,5,6,7 和 10 中，改进 T 方法能够探测到额外迁移 t_7；在实验

1,2,3,4,5,8,9 和 10 中,改进 T 方法能够探测到额外迁移 t_8.实验表明改进 T 方法对额外迁移具有一定的探测能力.

(4)对图 5 中的 M_1 而言,在 10 次实验中,改进 T 方法随机产生测试输入的平均长度为 23.4,实际有效的测试输入平均长度为 14.2.实验中改进 T 方法产生的测试序列长度并不太长.

3.7 扩展 Wp 方法

文献[2]提出了 Wp 方法的扩展应用,引入了区分集的概念.

定义 13(区分集). Mealy 机的区分集被定义为 $Z=(\{\varepsilon\}\cup I\cup I^2\cup\cdots I^{m-n})$. $W=X[m-n]\cdot W$,其中 I 为 Mealy 机的输入符号集,n 为 M_S 中的状态数目,m 表示实际应用 M_I 的状态数目且 $m>n$,$X[m-n]$表示$\{\varepsilon\}\cup I\cup I^2\cup\cdots\cup I^{m-n}$,$W$ 为特征集.

实际应用中,M_I 的状态数目可能比 M_S 中的状态数目多,因此文献[2]提出利用区分集 Z 代替特征集 W.为发现图 5 中 M_I 的额外状态,本文假定 $m-n=1$,即 $Z=I.W$,其中 $I=\{a,b,c,d\}$,特征集 $W=\{a,b\}$,识别集 $W_1=\{a,b\}$,$W_2=\{a\}$,$W_3=\{b\}$.同样扩展 Wp 方法也包含了两个阶段.扩展 Wp 方法得到的测试输入及测试输出如表 11、表 12 所示.

表 11 扩展 Wp 方法的第 1 阶段产生的测试序列

Q	测试序列		
	ε	b	bb
$r.Q.I.W$	$ra,rb,raa,rba,rca,rda,$ rab,rbb,rcb,rdb	$rbaa,rbba,rbca,rbda,$ $rbab,rbbb,rbcb,rbdb$	$rbbaa,rbbba,rbbca,rbbda,$ $rbbab,rbbbb,rbbcb,rbbdb$
输出	$0,1,00,11,-,$ $-,01,11,-,-$	$111,110,1-,1-,$ $111,111,1-,1-$	$1100,1110,11-,11-,$ $1100,1101,11-,11-$

表 12 扩展 Wp 方法的第 2 阶段的测试序列

R	测试序列				
	a	ba	bba	bbb	
测试状态	s_1	s_2	s_3	s_1	s_1
$r.R.I.W_i$	$raaa,raba,raca,rada,$ $raab,rabb,racb,radb$	$rbaaa,rbaba,$ $rbaca,rbada$	$rbbab,rbbbb,$ $rbbcb,rbbdb$	$rbbbaa,rbbbba,$ $rbbbca,rbbbda$	$rbbbab,rbbbbb,$ $rbbbcb,rbbbdb$
输出	$000,011,0-,0-,$ $001,011,0-,0-$	$1111,1110,$ $11-,11-$	$1100,1101,$ $11-,11-$	$11000,11011,$ $110-,110-$	$11001,11011,$ $110-,110-$

在表 11、表 12 中,符号"—"表示输出为空.由表 11、表 12 可知,测试序列的总长度数为 214.图 1(a)中 M_I 的迁移错误 $t'=(s_3,a/0,s_2)$ 能够由测试序列 $rbbab$ 发现,而图 2(a)中 M_I 的输出错误 $b/1$ 能由测试序列 $rbbba$ 发现,图 5 中 M_I 的错误额外迁移错误 t_7 和 t_8 分别由 $rbbca$ 和 $rbbda$ 发现.当 $m-n$ 越大时,扩展 Wp 方法产生的测试序列越长.

为减少扩展 Wp 方法产生的测试序列长度,文献[23]提出了 R-Wp 方法,该方法通过排除 Wp 方法中错误的状态迁移而减少测试输入序列的数目.R-Wp 方法包含 3 个步骤:

1.估计实现中的有限状态机中状态数目的上界 m,其中 m 要大于或等于规约说明中的有限状态机中的状态数目 n.

2.检验在规约说明中的状态在实现中可被识别并验证其正确性,同时从初始状态到这些状态所经历的状态迁移也被验证,该部分的测试输入序列可如下构造:$T_1=Q\cdot X[m-n]\cdot W$,其中 I 表示测试输入集合,$X[m-n]$表示 $\{\varepsilon\}\cup I\cup I^2\cup\cdots\cup I^{m-n}$.

3.验证在上面没有被验证的状态迁移,测试输入序列可构造如下:

$$T_2=(P-Q)\otimes W=\bigcup_{p_1\in(P-Q)}\{p_1\}\cdot W_j,$$

其中 Q 是 S 的状态覆盖集,P 是 S 的迁移覆盖集,W 是有限状态机的特征集.

由 R-Wp 方法的实现步骤可知,该方法的实现需要利用状态覆盖集 Q、迁移覆盖集 P 和特征集 W.因此可参照上述相关算法获得 R-Wp 方法的测试序列.由 R-Wp 方法的实现可知,第 1 阶段的测试序列为表 11 中的测试序列,而第 2 阶段的测试序列为表 4 中的测试序列.综合表 11 和表 4,R-Wp 方法的测试序列总长度为 121.文献[23]证明了 R-Wp 方法与扩展 Wp 方法具有相同的错误探测能力.

4 冗余约简

软件测试的目的是以较小的测试成本为代价,尽可能发现软件中的错误.上述基于 FSM 的测试方法产生的测试序列集中可能包含冗余,如对于 W 方法,表 2 中测试序列 ra 和 raa 之间存在冗余,因为能被序列 ra 发现的错误,一定能够被序列 raa 发现,则序列 ra 就是冗余序列.冗余序列的存在会导致测试成本的激增,为此本文提出在维持错误探测能力的前提下,减少测试序列中的冗余.

文献[14]从测试覆盖准则的角度,分析满足不同测试覆盖准则的测试用例集合内存在的冗余问题,该研究能够保证约简后的测试用例集合同样也

满足某个测试覆盖准则.但是文献[14]中的研究并不能保证约简前后的测试用例集合具备相同的错误探测能力.为此,本文提出采用前置和后置条件分析法,约简测试序列中的冗余,即在相同前置条件和后置条件下,测试序列的冗余约简并不会降低测试序列集合的错误探测能力.

定义 14(前置和后置条件). 测试序列 A 中的某个元素 b 的前置条件是 A 中与 b 相邻的前一个元素 a 的输出,元素 b 的后置条件是元素 b 的输出.

由定义 14 可知,在图 1(a)的 M_S 中,测试序列 ra,其中 a 的前置条件为 r 的输出"—",a 的后置条件为 0.对测试序列为 $raabb$,该序列的前置条件为—001,该序列的后置条件为 0011.

定义 15(冗余). 若测试序列 A 包含了测试序列 B,且包含部分与 B 拥有相同的前置条件和后置条件,则在序列 A 和 B 中,B 是冗余的.

对于图 1(a)中的 M_S 而言,测试序列 raa 包含测试序列 ra,且包含部分与 ra 拥有相同的前置条件与后置条件,则 ra 是冗余的.

定理 5. 对测试序列 A,B 而言,若 B 是冗余的,则测试序列 A 的错误探测能力与序列 A 和 B 的错误探测能力相同.

证明. 假定测试序列 B 能够探测到错误集合为 E,则 $B\rightarrow E$.由定义 15 可知,A 包含 B,则对于序列 B 中的任意一个元素 b,在 A 包含 B 的部分中一定存在一个元素 a,使得 $b=a$ 且它们拥有相同的前置条件和后置条件,则 $a\rightarrow b$,因此 $A\rightarrow B$.由关系的传递性可知,若 $A\rightarrow B$ 和 $B\rightarrow E$,则有 $A\rightarrow E$,即序列 A 也能探测到错误集合 E.因此序列 A 与序列 A 和 B 的错误探测能力相同.

对序列 A 和 B,若 A 仅仅包含 B,则并不能得出 $A\rightarrow B$.因为序列 B 有它的前置条件和后置条件,前置条件是序列 B 出现的前提,后置条件保证了序列 B 的错误探测能力,因此若不能保证 A 中包含 B 的部分也与 B 一样具有相同的前置条件和后置条件,则序列 A 并不能保证发现序列 B 能够探测到的错误.

依据本文的冗余定义,可以对上述所有基于 FSM 的测试生成方法进行冗余约简.本文给出序列集合的冗余约简算法 TSRD.

算法 TSRD. 测试序列的冗余约简算法.

输入:测试序列组 $ts[1],ts[2],\cdots,ts[n]$和 Mealy 机M_S

输出:被约简后的测试组

```
1. StringBuffer sb=new StringBuffer(); //初始化
2. 按序列长度递减排序测试序列组 ts[1], ts[2],…, ts[n];
3. sb.append(ts[1]).append("@");
4. for (int i=0; i<ts.length; i++)
5.   for (int j=i+1; j<ts.length; j++)
6.     String st=λ(s1,ts[i]);
7.     String st1=λ(s1,ts[j]);
8.     If (ts[i].indexOf(ts[j])!=-1&&
         st.indexOf(st1)!=-1&&
         ts[i].indexOf(ts[j])==st.indexOf(st1))
                                  //ts[j]是冗余的.
9.       sb.append(ts[j]).append("@");
10.    End if
11.  End for
12.  删除字符串 sb 中的最后一个符号"@";
13.  ts=new String[sb.toString().split("@").length];
14.  ts=sb.toString().split("@");
15.  sb=new StringBuffer();
16.  for (int k=0; k<=i; k++)
       sb.append(ts[i]).append("@");
17. end for
18. return ts.split("@"); //返回被约简后的测试序列
```

算法 TSRD 的时间主要集中在步 2 和 4,其中步 2 的时间最多为 n^2,步 4 的时间不超过 n^2,因此算法 TSRD 的最坏时间复杂度为 $O(n^2)$.

5　实验评估

实验的内容包括测试成本、错误探测能力、冗余约简效率及改进 T 方法(本节简称 T 方法).实验的对象包含 3 个协议模型 CM1、CM2 和 CM3(来自文献[6,10,24]),一个学生管理系统模型 SIMS,一个在线邮箱模型 EMAIL 和一段程序模型 PM.针对程序模型 PM,我们在实际程序中添加了 5 个错误.有关程序建模的方法可以参考文献[8].实验中包含 3 类错误:错误类型 I 表示迁移错误,错误类型 II 表示输出错误,错误类型 III 表示额外迁移错误.模型 M_S 及其应用模型 M_I 的详细信息如表 13 所示.

表 13　M_S 和 M_I 的信息描述

模型	M_S 信息		M_I 信息		
	状态	迁移	总错误数	错误类型	错误类型数目
CM1	3	6	3	I,II,III	1,1,1
CM2	3	6	4	I,III	2,2
CM3	5	7	3	I,II	1,2
PM	11	12	5	I,II	4,1
SIMS	10	21	4	II,III	2,2
EMAIL	7	13	4	I,II,III	2,1,1

5.1　测试成本

影响测试成本的主要因素是测试用例的总数[25],因为对每条测试用例都需要分配资源,如测试计划、测试设计和测试执行.其中测试执行又与测

试用例的长度有关.为此,本文提及的测试成本包括测试序列的总长度和测试序列的总数.假定测试序列总数产生的测试成本为 $C_1=h(S_q)$,是一个关于序列总数 S_q 的函数,测试序列总长度产生的测试成本为 $C_2=g(L_s)$,是一个关于序列总长度 L_s 的函数,则总的测试成本为 $C=C_1+C_2$.为获得函数 $h(S_q)$ 和 $g(L_s)$,不妨假设一条总长度为 1 的测试序列产生的测试成本为 1,则 $1=h(1)+g(1)$.因此,$g(1)=1-h(1)$,$h(1)<1$ 及 $g(1)<1$.假设 $\alpha=h(1)$,则 $1-\alpha=g(1)$.因此,函数 $h(S_q)=\alpha\times S_q$,$g(L_s)=(1-\alpha)\times L_s$.基于上述分析,本文给出了测试成本的量化评估指标.

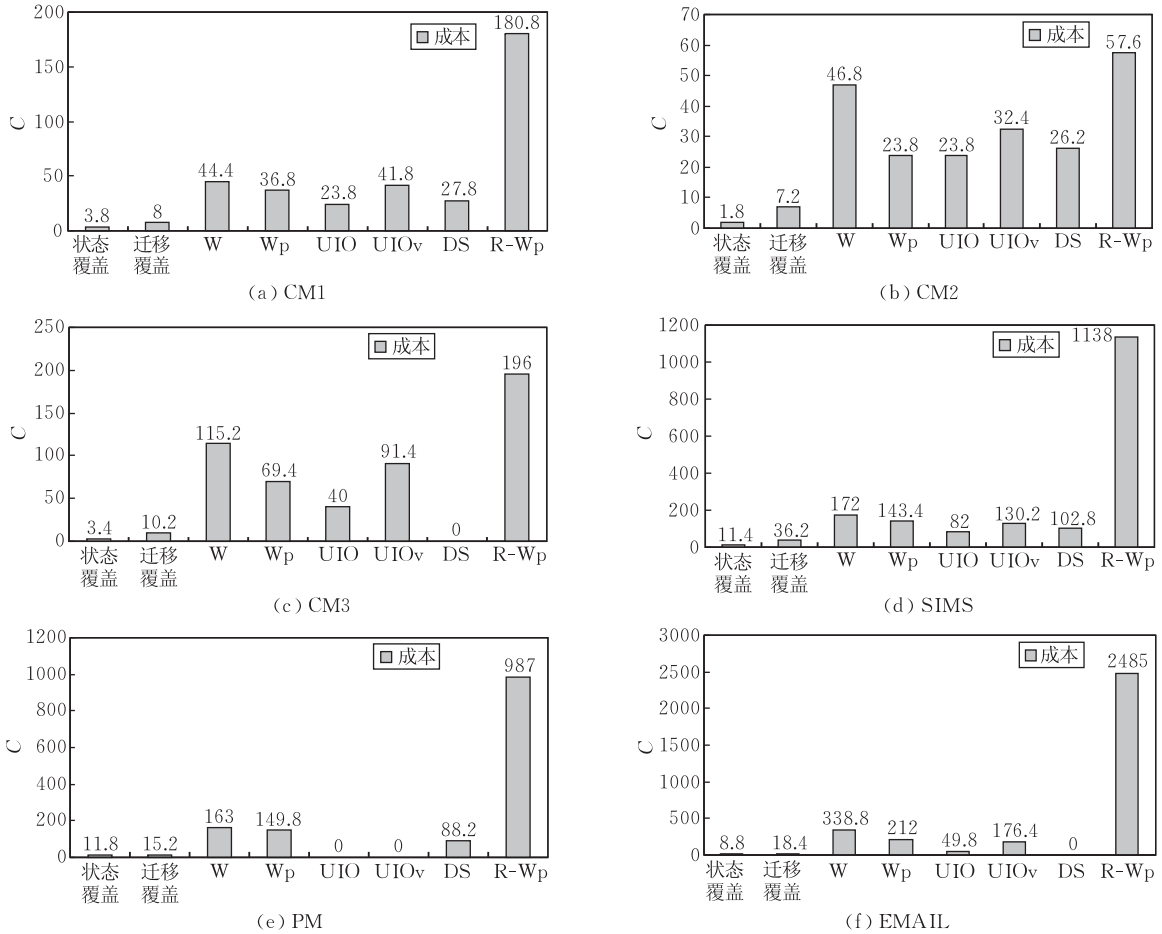
定义 16(测试成本). 基于 FSM 的测试成本 C 被定义为

$$C=\alpha\times S_q+(1-\alpha)\times\sum_{1\le i\le S_q}L_i \qquad (1)$$

其中 S_q 表示测试序列的总数,L_i 表示第 i 条测试序列的长度,α 是小于 1 的测试参数,表示测试序列的数目对测试成本的影响,$1-\alpha$ 表示测试序列的总长度对测试成本的影响.

一般而言参数 α 小于 1/2,即测试序列总长度对测试成本的影响要大于测试序列数目对测试成本的影响.依据式(1),可以量化评估基于 FSM 的测试方法的测试成本,同时也能估算测试时间.若知道人均每日测试成本 C_d,则测试时间 $D=C/C_d$.因此,式(1)可以作为实际软件测试成本花费的一项重要评价指标.

为量化评估上述基于 FSM 的测试生成方法的成本开销,本文假定参数 $\alpha=0.2$,分别计算由不同模型产生的测试序列的成本.在实验中 R-Wp 方法设定的区分集 $Z=X[1]\cdot W$.由于 T 方法的测试序列长度是随机的,导致测试成本的随机变化,因此本节没有将 T 方法与其它方法进行测试成本比较,而在 5.4 节对 T 方进行实验评估.图 6 显示了上述不同测试生成方法应用于不同模型时的测试成本消耗.在图 6(c)和(f)中的 DS 方法结果为 0,其原因是不能由模型 CM3 和 EMAIL 获得区分序列,(d)中的 UIO 和 UIOv 方法的结果为 0,其原因是 PM 模型不存在 UIO 序列,即 PM 模型不是强连通的.



(a) CM1 (b) CM2 (c) CM3 (d) SIMS (e) PM (f) EMAIL

图 6 若干基于 FSM 的测试方法的测试成本

由图 6 可以得出一些实验结论：

(1) 一般而言，基于 FSM 的测试方法的成本递减顺序：R-Wp 方法＞W 方法＞UIOv 方法＞Wp 方法＞DS 方法＞UIO 方法＞迁移覆盖＞状态覆盖.

(2) 若仅从成本角度考虑，R-Wp 方法是最不经济的. 实验中 R-Wp 方法仅考虑 $m-n=1$ 的情况，若考虑 $m-n>1$ 的情况，则 R-Wp 方法的测试成本还将成倍增长. 另外，在程序模型 PM 及两个软件系统模型 SIMS 和 EMAIL 中，R-Wp 方法的测试成本是数倍甚至是数十倍于其它基于 FSM 的测试方法.

(3) DS 方法、UIO 方法和 UIOv 方法并不适用于所有的 FSM 模型.

5.2　错误探测能力

本文讨论的错误探测能力包含错误的类型和错误的数目，其评价指标是指在实际程序中植入错误，考核被探测错误在所植入的同类型错误中所占的比重，即被查杀的变异得分(mutation score)[26]. 为探测实际软件中的错误，本文分别为模型 CM1、CM2 和 CM3 设计了相应的 M_1，为程序模型设计了一段包含错误的程序，而模型 SIMS 和 EMAIL 则直接采用它们的应用. 为了量化评估测试效率，本文给出了错误探测率的定义.

定义 17(错误探测率).　软件的错误探测率 ρ 被定义为

$$\rho=\frac{1}{S_t}\times\sum_{1\le i\le S_t}\frac{N_f(i)}{S_f(i)} \tag{2}$$

其中 S_t 表示错误类型的总数，$N_f(i)$ 表示探测到 i 型错误的数目，$S_f(i)$ 表示软件中存在的 i 型错误的总数.

式(2)将不同类型的错误视为同等重要，错误探测率 ρ 涉及错误类型的种类和同一种错误类型的错误数目. 依据式(2)，可以量化计算各种测试方法的错误探测能力. 图 7 显示了这些测试方法的错误探测率.

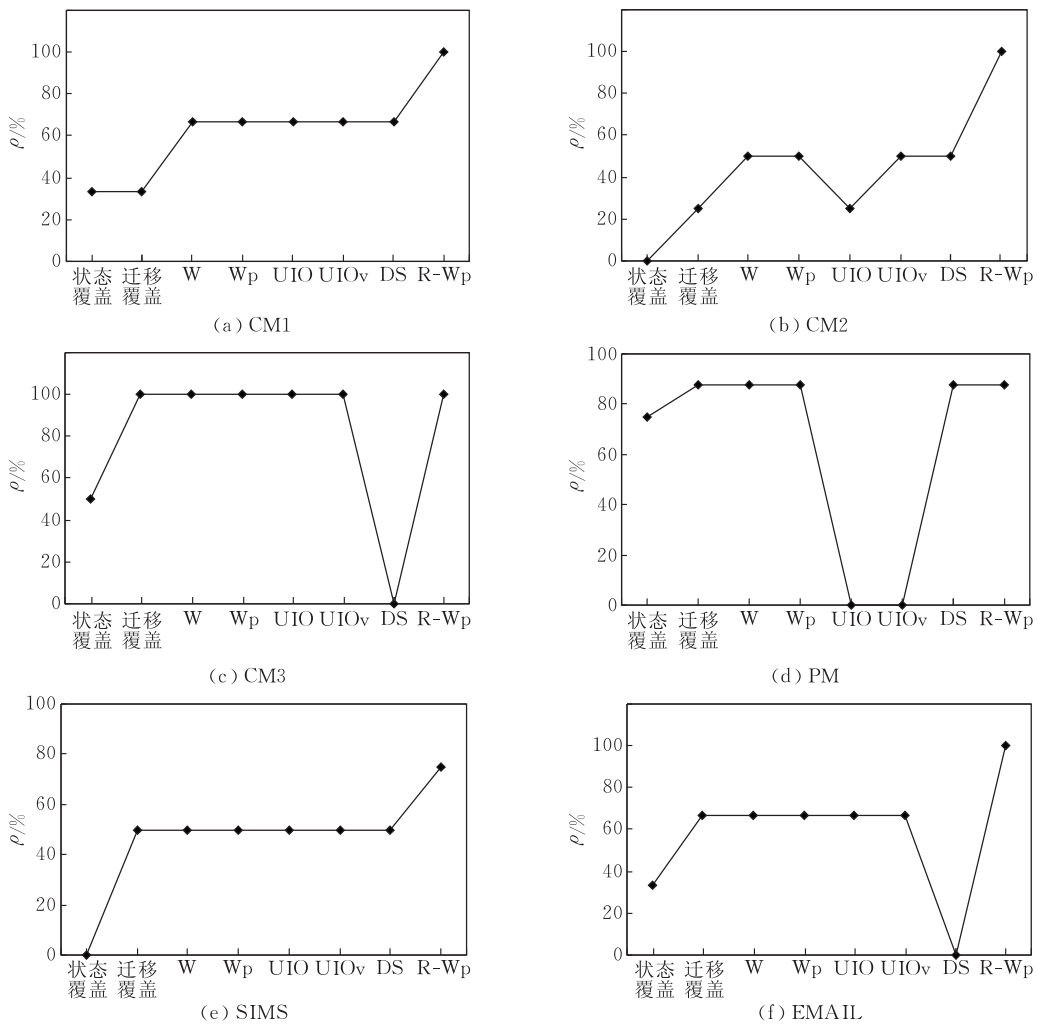


图 7　若干基于 FSM 的测试方法的错误探测率

在图 7(c)和(f)中，DS 方法的错误探测率为 0，其原因是模型 CM3 和 EMAIL 不存在区分序列，(d)中 UIO 方法和 UIOv 方法的错误探测能力为 0，其原因是模型 PM 不是强连通的. 若不考虑图 7 中

DS 方法、UIO 方法和 UIOv 方法的无效情况，则由图 7 可以得出一些重要的结论：

(1) 上述任何一种测试方法都不能保证 100% 探测到所有系统中的错误，其中 R-Wp 方法的错误探测率最高，状态覆盖的错误探测率最低，W 方法、Wp 方法和 UIOv 方法的错误探测率相同.

(2) 在(a)和(b)中，迁移覆盖的错误探测率比 W 方法和 Wp 方法的错误探测率要低. 这是由于协议测试时状态位信息不可获得，造成迁移覆盖产生的测试序列不能辨别最终获得的状态，导致其错误探测能力的下降.

(3) 在(d)、(e)和(f)中，迁移覆盖的错误探测率与 W 方法和 Wp 方法的错误探测率相同，其原因是在对程序和两个系统的测试时，状态位可获得. 因此，若测试时状态位可获得，迁移覆盖的错误探测率等同于 W 方法和 Wp 方法.

(4) 在(b)中，UIO 方法的错误探测率比 UIOv 方法低下，因此 UIO 方法的错误探测率不如 W 方法、Wp 方法和 UIOv 方法.

(5) 在(c)、(d)和(f)中，DS 方法、UIO 方法和 UIOv 方法的错误探测率为 0，因此 DS 方法、UIO 方法和 UIOv 方法的使用范围比 W 方法和 Wp 方法要窄.

为进一步评估这些方法，本文给出了针对某一种错误类型的探测效率的评价指标.

定义 18(i 型错误探测率). 软件的 i 型错误探测率 $\rho(i)$ 被定义为

$$\rho(i)=\frac{\sum\limits_{1\le k\le n} N_{f(i)}(k)}{\sum\limits_{1\le k\le n} S_{f(i)}(k)} \tag{3}$$

其中 $N_{f(i)}(k)$ 表示在第 k 个模型中探测到 i 型错误的数目，$S_{f(i)}(k)$ 表示第 k 个模型中 i 型错误的总数.

定义 18 给出的是某一种错误类型的平均错误探测率. 依据式(3)，我们能够统计上述测试生成方法的 I、II 和 III 型错误探测率，如图 8 所示. 一些重要的结论可以由图 8 得出：

(1) 在实验中，上述方法对错误类型 II 的探测是最有效的，其中迁移覆盖、W 方法、Wp 方法和 R-Wp 方法能够 100% 地探测到系统中错误类型 II 的所有错误.

(2) 错误类型 I 的探测效率要低于错误类型 II 的探测效率，且没有一种方法能够保证发现所有应用中错误类型 I 的错误. 这个问题的原因在于模型 PM. 例如，若程序中的迁移条件为 $a>0$，但被误写成 $a\ge 0$，则输入 $a=1$ 不会发现这个迁移错误.

(3) 错误类型 III 的探测效率最低，在实验中，仅有 R-Wp 方法能够发现这类错误，且不能保证发现错误类型 III 中的所有错误.

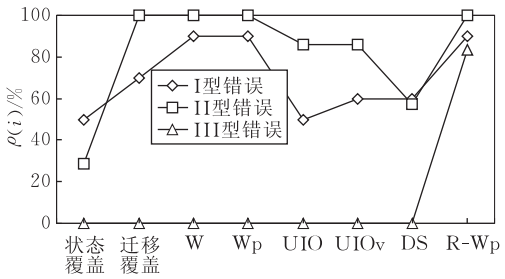


图 8 I 型、II 型和 III 型错误的探测率

5.3 冗余约简效率

本文所指的冗余约简包括测试序列总长度的约简和测试集合大小的约简. 由于本文提出的冗余约简方法对状态覆盖和迁移覆盖没有任何效率，因此本文仅针对 W 方法、Wp 方法、UIO 方法、UIOv 方法和 R-Wp 方法进行冗余约简效率的评估. 本文给出了两项冗余约简的评价指标.

定义 19(测试序列总长度的约简率). 测试序列总长度的约简率 R_L 被定义为

$$R_L=\frac{\sum\limits_{1\le i\le S_q} L_i(r)}{\sum\limits_{1\le 1\le S_q} L_i} \tag{4}$$

其中 S_q 表示测试序列的总数，L_i 表示第 i 条测试序列的长度，$L_i(r)$ 表示第 i 条序列中冗余的长度，若第 i 条序列满足冗余定义(定义 15)，则 $L_i(r)=L_i$，否则 $L_i(r)=0$.

定义 20(测试集合的约简率). 测试集合的约简率 R_S 被定义为

$$R_S=\frac{S_q(r)}{S_q} \tag{5}$$

其中 S_q 表示测试集合中序列的总数，$S_q(r)$ 表示测试集合中被约简的测试序列总数.

依据式(4)和(5)，本文对不同方法的冗余约简效率进行了评估，结果如图 9、图 10 所示. 由于模型 PM 不存在 UIO 序列，因此 UIO 方法和 UIOv 方法不能应用在模型 PM 中，同理 DS 方法不能应用在模型 CM3 和 EMAIL 中. 由图 9、图 10 可以获得一些重要信息：

(1) 本文提出的冗余约简方法对大多数基于 FSM 的测试方法都是有效的. 其中对 W 方法、Wp

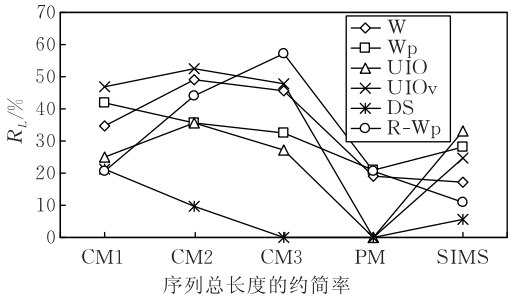


图 9　测试序列总长度的约简率

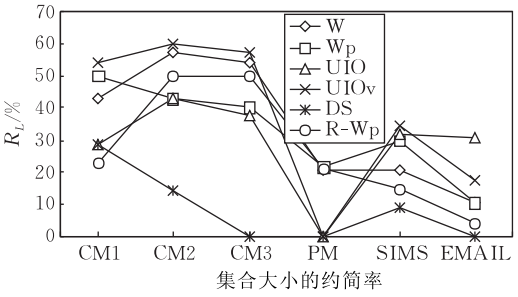


图 10　测试集合大小的约简率

方法、UIOv 方法和 R-Wp 方法的冗余约简效率较高，而对 DS 方法的冗余约简效率相对要低.

(2) 在协议模型 CM1、CM2 和 CM3 的应用中，冗余约简的效率要明显高于对程序模型 PM 和实际的系统模型 SIMS 和 EMAIL.

5.4　T 方法的实验

本节实验的目的是为了评估 T 方法的测试成本和错误探测能力. 在实验中，若 M_I 中的状态比 M_S 中的迁移少，则 T 方法产生的测试序列将无法通过测试 M_I 而获得与 M_S 中一样的迁移. 为此，在实验中本文为 T 方法增加了一条终止条件，即当随机产生 $n(n-1)\times|I|$ 次测试输入时，T 方法强制结束. 与 3.6 节中的计算机模拟实验不同，本节的实验假定错误应用 M_I 不可知. 本节采用 T 方法，分别由模型 CM1、CM2、CM3、PM、SIMS 产生 5 组测试序列，比较各组测试用例的长度及探测到的错误类型. 实验结果如表 14 所示.

表 14　T 方法的实验结果

实验	CM1 的结果		CM2 的结果		CM3 的结果		PM 的结果		SIMS 的结果		EMAIL 的结果	
	长度	错误类型	长度	错误类型	长度	错误类型	长度	错误类型	长度	错误类型	长度	错误类型
1	11	I,II	30	I,III	60	I,II	80	I	1100	I,II,III	151	I,II,III
2	15	I,II	30	I,III	60	I,II	33	I	1100	I,II,III	69	I
3	23	I,II,III	30	I,III	60	I,II	38	I,II	1100	I,II,III	177	II,III
4	12	I,II	30	I,III	60	I,II	25	I,II	1100	I,II,III	143	I,II
5	11	III	30	I,III	60	I,II	19	I	1100	I,II,III	132	I,II,III

由表 14 可知，T 方法即表现出一定的错误探测能力，又表现出令人难已接受的测试序列长度. 在 5 组实验中，随机产生的测试序列都不同程度地发现系统中的错误，而对模型 CM2、CM3 和 SIMS 的实验中，序列长度均为 $n(n-1)\times|I|$，这表明这 3 个模型的实际应用中可能缺少 M_S 中的迁移，通过观察实际的应用，我们发现 CM1 和 CM3 的应用缺少相应的迁移；而对模型 SIMS 的实际应用，由于概率问题，使得其中一条迁移无法在 $n(n-1)\times|I|$ 次被覆盖. 另外由实验可知，测试序列的长度是变化的，最大长度是最短长度的 2～3 倍，这表明 T 方法测试成本的稳定性差.

6　讨　论

依据上述的理论讨论及实验评估，本节对若干基于 FSM 的测试方法进行讨论，重点讨论了基于 FSM 的测试应用的难点、应用范围、测试成本和错误探测能力，并给出了一些基于 FSM 的测试的经验建议.

基于 FSM 的测试应用难点：(1) 建立有效的 FSM 模型. 当软件的应用复杂时，模型的建立将会变得非常困难，因为我们需要预知所有可能的状态、输入/输出信息、存在的迁移约束以及模型中的性质，同时还需要考虑建模粒度、建模工具以及模型的有效性；(2) 缺乏通用的支持工具. 虽然目前国内外一些研究人员开发了若干基于 FSM 的测试支持原型工具[5,13,27]，但这些原型工具并不能完全自动的由 FSM 模型产生有效的测试用例，同时不同的工具原型所针对的应用领域也不同，因此工具不具有通用性.

现有基于 FSM 的测试研究需要一些前提和假设，导致了基于 FSM 的测试方法应用的限制. 表 15 显示了不同测试方法的应用条件及错误探测能力的分析. 虽然文献[4]认为 W 方法、Wp 方法、DS 方法和 T 方法需要 M_S 是强连通的(假设 3)，但在本文的实验中，当 M_S 满足可达性(假设 3′)时，这些方法即可应用. UIO 方法和 UIOv 方法需要 M_S 是强连通的，否则 M_S 不存在 UIO 序列. 表 15 中“—”表示不能确定，l.st 表示成本最低，l 表示低成本，m 表示中等成本，h 表示高成本，h.st 表示最高成本.

表 15 若干基于 FSM 的测试方法的分析

比较项目	假设	成本	错误探测能力	算法时间复杂度
状态覆盖	2,3′,7,8	l.st	I,II	$O(n^3)$
迁移覆盖	2,3′,7,8	l	I,II	$O(n^3)$
W	1,2,3′,4,7	h	I,II	$O(n^3)$
Wp	1,2,3′,4,7	h	I,II	$O(n^3)$
UIO	1～4,7	m	I,II	$O(n^4)$
UIOv	1～4,7	h	I,II	$O(n^4)$
DS	1,2,3′,4,7	m	I,II	—
R-Wp	1,2,3′,7	h.st	I,II,III	$O(n^3)$
T	1,2, 3′	—	I,II,III	—

依据表 15,本文给出了基于 FSM 的测试的一些经验建议:

(1)若不计较测试成本,建议选择 R-Wp 方法进行软件测试.虽然 R-Wp 方法并不能保证探测出所有的错误,但对比其它方法,该方法的错误探测率最高.

(2)若测试成本受限时,当状态信息可获得时,建议选择迁移覆盖方法;当状态不能获得时,建议选择 UIO 方法或 DS 方法.

(3)综合考虑测试成本和错误探测能力,我们建议采用组合测试.当状态信息可获得时,建议选择迁移覆盖方法与 T 方法进行组合测试;当状态信息不能获得时,建议选择 Wp 方法或 UIOv 方法与 T 方法进行组合测试.

7 相关工作

基于 FSM 模型的测试已被广泛研究,形成了一系列的测试生成方法及相关理论,但工业界却很少在实际中应用这些理论成果[1,28].为搭建由理论研究领域通往工业界应用领域的桥梁,研究人员对基于(FSM)模型的测试进行综述和评估.

Sidhu 等人[4]以运输协议为例,学习了若干基于 FSM 的测试方法.实验表明,UIO 方法、DS 方法和 W 方法在强一致性测试(strong conformance test)中能够探测所有的错误,而 T 方法并不能发现所有的错误,且测试序列集中存在大量的冗余.然而,在我们的实验中 UIO 方法的错误探测能力比 W 方法弱,且 UIO 方法和 DS 方法都存在失效的情况.我们认为其原因是 Sidhu 等人的实验案例是一个个案,因此实验结果不具有代表性,同样强一致性测试中的错误类型也较实际软件中的错误类型少.Dalal 等人[5]利用一个基于模型的测试支持工具测试了 4 个实际项目,并讨论了实现基于模型的测试时的注意事项.然而测试支持工具仅支持特定建模语言 AETGSpec,而非理论研究非常成熟的有限自动机.Broy 等人[6]综述了若干基于 FSM 的测试生成方法,讨论了方法的理论基础,但缺乏相关的实验比较和评估.另外,文中的算法并不完善,部分算法难以应用.Neto 等人[3,7,28]利用网上 5 大数字图书馆资源,对基于模型的测试的论文进行统计,讨论了现有研究工作状况、分类、工具支持及不同方法的特性等,这些都为工业应用提供了参考价值.然而他们的研究并没有比较各种测试方法的好坏,判断一种基于模型的测试方法好坏的依据是能否有相应的工具支持.Mohagheghi 等人[29]对基于模型的测试中模型的质量提出了 6 个评价指标:正确性、完整性、一致性、可理解性、工具支持性和局限性.Pretschner 等人[30]提出了基于模型的测试有 3 个关键因素:描述软件行为的模型、支持测试生成的工具和测试生成算法(准则),并对前两项关键因素进行了讨论.Dorofeeva 等人[31]采用实例学习了多种测试生成方法,评估了测试序列的总长度,但并未对错误探测能力进行讨论及测试成本进行定量的分析.Ammann 等人[8]对若干基于 FSM 的测试覆盖准则进行了评估,但缺少对不同准则的实现算法.

与现有基于 FSM 模型的测试评估不同,本文讨论并完善了基于 FSM 的测试方法的理论和若干实现算法,从多个角度对若干基于 FSM 的测试方法进行实验比较.为了实现比较的客观性和可操作性,本文提出了实验评估的 5 项量化指标.

另一项与本文研究工作相关的是测试序列集合的冗余约简.早期关于测试用例集合的约简技术主要是采用一些启发式算法[32-33]选取测试有效的用例,随后聂长海等人[34]提出利用需求间的相互关系对测试集合进行约简,这些研究非常适合回归测试中的冗余约简,而并不适合基于 FSM 的测试序列集合的约简.一些研究者提出利用序列的重叠(overlap)[12,35]部分对测试序列集合进行合并,但这种约简技术并未考虑到约简前后测试序列集合的错误探测能力.我们之前研究了由测试覆盖准则产生测试序列集合的冗余约简[13],提出了利用字符串匹配方式的冗余约简方法,该方法能够保证约简前后都满足某项测试覆盖准则.与之前的研究不同,本文的研究保证了约简前后测试序列集合拥有相同的错误探测能力.

8 结 论

基于 FSM 的测试能够由描述软件行为的模型自动产生测试序列集合,极大地降低了测试用例设计的成本.然而由于现有理论的不完善及通用性支持工具的缺乏,目前基于 FSM 的测试还未在工业界得到广泛的应用.为提高软件自动化测试的水平,

推广基于 FSM 的测试在工业界的应用，本文对基于 FSM 的测试理论和方法进行了补充及实验评估.

本文的主要贡献如下：(1) 补充了基于 FSM 模型的测试理论，如提出了区分序列的构造理论及测试序列集合冗余约简的理论；(2) 改进并实现了相关的基于 FSM 的测试生成算法；(3) 提出了对基于 FSM 的测试评估的 5 项量化指标，实验评估了若干基于 FSM 的测试方法，并给出了应用这种测试方法的相关经验性建议.

测试支持工具是基于 FSM 的测试方法应用的一项重要指标. 我们已经开发了一个基于 FSM 的、面向 Web 应用的测试原型工具，将来的主要工作是开发一个通用的、基于 FSM 的测试支持工具，包括 FSM 的建模、测试生成、测试执行和测试评估等，以便更好地推广基于 FSM 的测试方法.

参　考　文　献

[1] Bertolino A. Software testing research: Achievements, challenges, dreams//Proceedings of the Future of Software Engineering (FOSE 2007). Minneapolis, MN, USA, 2007: 85-103

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

[3] Neto A D, Subramanyan R, Vieira M, Travassos G H, Shull F. Improving evidence about software technologies: A Look at model-based testing. IEEE Software, 2008, 25(3): 10-13

[4] Sidhu Deepinder P, Leung Ting-Kau. Formal methods for protocol testing: A detailed study. IEEE Transactions on Software Engineering, 1989, 15(4): 413-426

[5] Dalal S R, Jain A, Karunanithi N, Leaton J M, Lott C M, Patton G C, Horowitz B M. Model-based testing in practice//Proceedings of the 21st International Conference on Software Engineering. California, United States, 1999: 285-294

[6] Broy M, Jonsson B, Katoen J-P, Leucker M, Pretschner A. Model-based testing of reactive systems—Advanced lectures. LNCS 3472. Springer Verlag, 2005

[7] Neto A D, Subramanyan R, Vieira M, Travassos G H. A survey on model-based testing approaches: A systematic review//Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering. Atlanta, Georgia, 2007: 31-36

[8] Ammann P, Offutt J. Introduction to Software Testing. Cambridge, UK: Cambridge University Press, 2008

[9] Chow T. Testing software designs modeled by finite-state machines. IEEE Transactions on Software Engineering, 1978, SE-4(3): 178-187

[10] Miao Huaikou, Liu Pan, Mei Jia. An improved algorithm for building the characterizing set//Proceedings of the 4th Theoretical Aspects of Software Engineering Conference. Taipei, China, 2010: 67-74

[11] Guo Q. Improving fault coverage and minimising the cost of fault identification when testing from finite state machines [Ph. D. dissertation]. Brunel University, Uxbridge, West London, British UK, 2006

[12] Duan Lihua, Chen Jessica. Reducing test sequence length using invertible sequences//Proceedings of the 9th International Conference on Formal Engineering Methods. Boca Raton, FL, United states, 2007: 171-190

[13] Miao Huaikou, Liu Pan, Mei Jia. A new approach to automated redundancy reduction for test sequences//Proceedings of the 15th Pacific Rim International Symposium on Dependable Computing. Shanghai, China, 2009: 93-98

[14] Qian Zhongsheng, Miao Huaikou, Zeng Hongwei. A practical web testing model for web application testing//Proceedings of the 3rd International Conference on Signal-Image Technology & Internet-based Systems. Shanghai, China, 2007: 404-411

[15] Yan Jun, Zhang Jian, Xu Zhongxing. Finding relations among linear constraints//Proceedings of the 8th International Conference on Artificial Intelligence and Symbolic Computation. Beijing, China. LNAI 4120. Heidelberg: Springer-Verlag, 2006: 226-240

[16] Lee D, Yannakakis M. Principles and methods of testing finite state machines — A survey. Proceedings of the IEEE, 1996, 84(8): 1090-1126

[17] Miao Huai-Kou, Li Gang, Zhu Guan-Ming. Software engineering language — Z. Shanghai: Scientific and Technical Documents Publishing House, 1999(in Chinese)
(缪淮扣，李刚，朱关铭. 软件工程语言—Z. 上海：上海科学技术文献出版社，1999)

[18] Yannakakis M, Lee D. Testing finite state machines//Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing. New York, NY, USA, 1991: 476-485

[19] Sabnani K, Dahbura A. A protocol test generation procedure. Computer Networks and ISDN Systems, 1988, 15(4): 285-297

[20] Vuong S T, Chan W Y L, Ito M R. The UIOv-method for protocol test sequence generation//Proceedings of the 2nd International Workshop on Protocol Test Systems. North-Holland, 1990: 161-176

[21] Lee D, Yannakakis M. Testing finite state machines: State identification and verification. IEEE Transactions on Computers, 1994, 43(3): 306-320

[22] Naik K. Efficient computation of unique input/output sequences in finite-state machines. IEEE/ACM Transactions on Networking, 1997, 5(4): 585-599

[23] Zhang Yong, Qian Le-Qiu, Wang Yuan-Feng. Test sequences selection based on deterministic finite-state machines. Journal of Computer Research and Development, 2002, 39(9): 1144-1150(in Chinese)
(张涌，钱乐秋，王渊峰. 基于确定有限状态机的测试输入序列选取. 计算机研究与发展，2002，39(9)：1144-1150)

[24] Chan W Y L, Vuong C T, Otp M R. An improved protocol test generation procedure based on UIOS//Proceedings of the Symposium Proceedings on Communications Architectures & Protocols. Austin, Texas, United States, 1989: 283-294

[25] Juristo N, Moreno A M, Vegas S. Reviewing 25 years of testing technique experiments. Empirical Software Engineering, 2004, 9(1): 7-44

[26] Wu P, Shi X C, Tang J J, Lin H M, Chen T Y. Metamorphic testing and special case testing: A case study. Journal of Software, 2005, 16(7): 1210-1220

[27] Belinfante A, Frantzen L, Schallhart C. Tools for test case generation. LNCS3472. Heidelberg: Springer-Verlag, 2005: 391-438. http://www.cs.ru.nl/~lf/publications/BFS05.pdf

[28] Neto A D, Travassos G H. Model-based testing approaches selection for software projects. Information and Software Technology, 2009, 51(11): 1487-1504

[29] Mohagheghi P, Dehlen V, Neple T. Definitions and approaches to model quality in model-based software development—A review of literature. Information and Software Technology, 2009, 5(12): 1646-1669

[30] Pretschner A. Model-based testing//Proceedings of the 27th International Conference on Software Engineering. St. Louis, MO, USA, 2005: 722-723

[31] Dorofeeva R, Yevtushenko N, El-Fakih K, Cavalli A R. Experimental evaluation of FSM-based testing methods//Proceedings of the 3rd IEEE International Conference on Software Engineering and Formal Methods. Koblenz, Germany, 2005: 23-32

[32] Chen T Y, Lau M F. A new heuristic for test suite reduction. Information and Software Technology, 1998, 40(5): 347-354

[33] Zhong Hao, Zhang Lu, Mei Hong. An experimental comparison of four test suite reduction techniques//Proceedings of the 28th International Conference on Software Engineering. Shanghai, China, 2006: 636-640

[34] Nie Chang-Hai, Xu Bao-Wen. A minimal test suite generation method. Chinese Journal of Computers, 2003, 26(12): 1690-1695(in Chinese)
(聂长海，徐宝文. 一种最小测试用例集生成方法. 计算机学报，2003, 26(12): 1690-1695)

[35] Simão A, Petrenko A, Yevtushenko N. Generating reduced tests for FSMs with extra states//Proceedings of the 21st IFIP WG 6.1 International Conference on Testing of Software and Communication Systems and 9th International FATES Workshop. Heidelberg: Springer-Verlag, 2009: 129-145

LIU Pan, born in 1976, Ph. D., lecturer. His current research interests include software testing, model-based testing, formal method, and algorithm design.

MIAO Huai-Kou, born in 1953, professor, Ph. D. supervisor. His current research interests include software engineering and formal method.

ZENG Hong-Wei, born in 1966, Ph. D., professor. His current research interests include formal method, formal verification, and software testing.

LIU Yang, born in 1981, Ph. D. candidate. His current research interests include formal method and formal verification.

Background

Model-based testing has been highlighted for several decades to search a way of test cases generation from system's behavior model. The finite state machine (FSM) has been taken as a tool for describing software requirements to build its behavior model. Test cases derived from FSM of software does not need to consider the real implementation of software. So time cost of test cases generation generated from the FSM model of software is markedly decreased than the traditional methods in software testing, especially, in regression testing.

However, it is in the last few years that we have seen a groundswell of interest in applying it to real applications. In fact, industrial adoption of FSM-based testing remains low and signals of the research-anticipated breakthrough are still weak. Therefore, researchers are today laying emphasis on how to beat the barriers to wide adoption of FSM-based testing in the industry.

To build the bridge between theoretical studies and industrial application towards FSM-based testing, the authors study and improve some FSM-based testing theories and methods, and present the theories for building distinguishable sequence and reducing the redundancies among test set derived from FSM-based testing methods. Next some practical algorithms for realizing some FSM-based testing methods are designed in the paper. Finally, the authors evaluate some FSM-based testing methods by using five suggested evaluation indicators on some experiments, and according to experimental results, give some experiential suggestions for the implementation of FSM-based testing methods.

The work is partially supported by the National Natural Science Foundation of China under Grant No. 60970007 and No. 61073050, the National High-Technology Research and Development Program of China under Grant No. 2007AA01Z144, the National Grand Basic Research Program of China under Grant No. 2007CB310800, the Natural Science Foundation of Shanghai Municipality of China under Grant No. 09ZR1412100, and the Shanghai Leading Academic Discipline Project (Project Number: J50103).