第41卷 第2期 西 安 交 通 大 学 学 报 Vol.41 №2
2007年2月 JOURNAL OF XI'AN JIAOTONG UNIVERSITY Feb. 2007

一种基于测试序列失败树的动态测试方法

赵保华[1,2]，高存皓[1,2]，姜振海[1,2]，周颢[1,2]，潘全科[1,2]
(1.中国科学技术大学计算机系，230027，合肥；2.安徽省计算与通讯软件重点实验室，230027，合肥)

摘要：针对传统的测试方法按照静态的顺序执行预先生成的固定测试集，而实际测试范围可能被缩小，同时测试结果受到测试序列间、测试序列内的影响等问题，提出了一种构造测试序列失败树(TSFT)并用其指导动态测试的方法.该方法考虑到了测试序列之间的相关性和已测结果对后续测试的影响，通过TSFT指导测试序列的动态执行，同时还考虑了引导序列和验证序列对测试结果的影响，在避开TSFT的前提下，可在引导序列和验证序列集合中进行搜索和替换，动态地生成新的有效测试序列.实验结果表明，所提方法避免了一些重复测试以及错误转换对正确转换的测试结果的影响，较明显地提高了实际的测试效率和测试覆盖率.

关键词：动态测试；测试效率；测试覆盖率；测试序列失败树
中图分类号：TP311 **文献标识码**：A **文章编号**：0253-987X(2007)02-0149-04

Dynamic Testing Method Based on Test Sequence Fail Tree

Zhao Baohua[1,2], Gao Cunhao[1,2], Jiang Zhenhai[1,2], Zhou Hao[1,2], Pan Quanke[1,2]
(1. Department of Computer, University of Science and Technology of China, Hefei 230027, China; 2. Anhui Province Key Laboratory of Software in Computing and Communication, Hefei 230027, China)

Abstract: The common methods of the communication protocol conformance testing include two steps: generate test suite, and execute the test suite in a static order. These methods may contain the problem of reducing the actual range of testing and the influence caused by other test sequence. A method of alterable test suite is presented, which considers the dependence of different test sequences and the effect caused by the leading sequence and verifying sequence on the test results The test sequence fail tree (TSFT) is used in the method, not only for guiding the dynamic execution of the test cases but also for searching and replacing in leading sequence and verifying sequence. The method generates valid test sequences in testing process dynamically, avoids the influence of fault transition on testing result of correct transition and will improve the test efficiency and test coverage.

Keywords: dynamic testing; test efficiency; test coverage; test sequence fail tree

协议测试[1-2]的目的是为了保证协议的实现符合协议的定义和规范，从而保证网络高效、稳定的运行.其中，协议一致性测试是检验网络协议正确性的最重要的手段.由于完全测试的代价是不可接受的，所以如何在开销和测试覆盖率上取得一个折中的结果是一个值得研究的课题.

传统的测试方法[3]是对待测协议进行形式化建模，通常使用有限状态机(Finite State Machine，FSM)，然后根据一定的测试序列生成算法生成测试集，随后在测试过程中按照一个固定的次序独立地运行各个测试子序列.此时，人们关注的是尽可能使用高效的测试序列生成算法生成更小的测试集，

收稿日期：2006-06-15. 赵保华(1947～)，男，教授，博士生导师. 基金项目：国家自然科学基金重大研究计划资助项目(90104010)；国家自然科学基金资助项目(60602016，60241004)；国家重点基础研究发展规划资助项目(2003CB314801)；华为基金资助项目(YJCB2006044TS).

并期待能检测更多的错误.

然而,此类方法忽略了测试子序列之间的相关性,这样就可能在实际测试之中产生如下问题:①重复测试,测试效率低;②因错误转换的干扰而未能对某些转换进行测试,测试覆盖率低.

为了解决这些问题,本文提出了一种基于测试序列失败树的动态测试方法,包括测试集的动态执行和测试集的动态生成,测试集的动态执行可以根据测试期间的反馈来决定测试子序列的测试顺序,从而提高测试效率,而测试集的动态生成可以根据测试期间的反馈来动态地生成新的、有效的测试子序列,进而提高测试覆盖率.

1 动态测试的必要性及其原理

可以通过一个实例来说明传统测试的弊端. FSM 状态机 M 及其各个状态的惟一输入/输出(Unique Input/Output,UIO)[4]序列如图 1 所示.

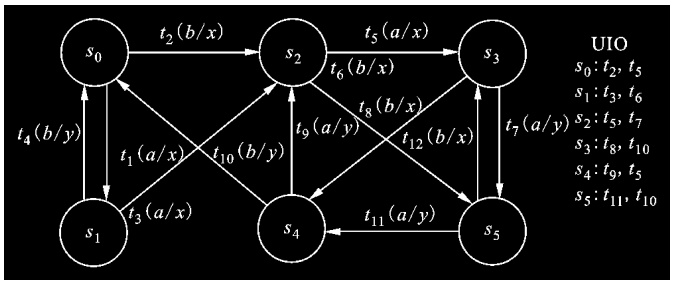

图 1 FSM 状态机 M 及其各个状态的 UIO 序列

采用 UIO 序列算法生成测试集,而状态机 M 的测试集如表 1 所示. 由测试集构成的测试树如图 2 所示. 假设其中的转换 t_5 发生了输出错误[5-6],此时传统的测试会体现出以下 3 个问题.

(1)存在一些虚假的测试结果. 因为在很多测试子序列中都包含 t_5,故无论待测转换是否正确,测试结果都是失败的,如 t_2、t_3、t_4、t_5、t_7、t_{10}、t_8、t_9、t_{11}.

(2)存在部分不必要的重复测试. 图 2 虚线中的 t_5 为根的子树,其实没有必要对它进行测试,因为$[t_2,t_5]$这个子序列必定出错,所以这棵子树代表的 3 个测试子序列只需要运行一次测试,然而在实际中运行了 3 次,增加了的测试代价是毫无意义的.

(3)有些待测转换实际并没有被测试,如 t_{10} 的测试子序列$[t_2,t_5,t_8,t_{10},t_2,t_5]$在运行至$[t_2,t_5]$时就已经出错,而这部分只是 t_{10} 的引导序列,其实还没有真正地测试 t_{10},结果降低了测试的覆盖率.

下面就以上 3 个问题的原理以及解决方案依次进行讨论.

(1)测试序列返回虚假的测试结果的根本原因,是引导序列和验证序列对待测转换的影响所致. 静态测试即使发现引导和验证序列包含了错误的转换,也无能为力. 因此,如果能够在测试过程中剔除测试序列中的引导和验证序列的错误转换,则可以保证测试序列的测试结果是有效的.

表 1 FSM 状态机 M 的测试集

测试目的	测试序列	测试目的	测试序列
$t_1:s_0\to s_1$	t_1,t_3,t_6	$t_6:s_2\to s_5$	t_2,t_6,t_{11},t_{10}
$t_2:s_0\to s_2$ $t_7:s_3\to s_5$	$t_2,t_5,t_7,t_{11},t_{10}$	$t_8:s_3\to s_4$ $t_9:s_4\to s_2$	t_2,t_5,t_8,t_9,t_5,t_7
$t_3:s_1\to s_2$	t_1,t_3,t_5,t_7	$t_{11}:s_5\to s_4$	t_2,t_6,t_{11},t_9,t_5
$t_4:s_1\to s_0$	t_1,t_4,t_2,t_5	$t_{12}:s_5\to s_3$	$t_2,t_6,t_{12},t_8,t_{10}$
$t_5:s_2\to s_3$ $t_{10}:s_4\to s_0$	$t_2,t_5,t_8,t_{10},t_2,t_5$		

(2)静态测试的测试序列之间相互独立,实际上各个测试序列之间具有相关性,并且在测试过程中每个测试序列执行后都会对待测的协议实现(Implementation Under Test,IUT)有一定的认识,进而指导其后的测试序列的运行. 如图 2 所示,t_2 的测试序列$(t_2,t_5,t_7,t_{11},t_{10})$在执行出错之后,可以得到的信息是 IUT 在$(t_2,t_5)$这条子序列上会出错,而尚未运行测试序列的 t_5、t_7、t_8、t_9、t_{10} 均经过子序列(t_2,t_5),因此这几个测试序列不必运行,可直接跳过. 在测试过程中,可以动态地维护一棵树,树中记录了已知的测试出错的子序列,每个测试序列在运行之前可以与这棵树进行比较,如果匹配则直接跳过而不执行该测试序列,以期提高测试效率.

(3)同讨论(1),在测试过程中可以动态地避免已知的错误转换出现在测试序列的引导序列之中.

2 动态测试算法

为了便于讨论,下面给出几个定义.

定义 1 待测转换集合(Transitions Set)为待测的 FSM 状态机 M 中的所有的转换,记为 $T=\{t_1,t_2,\cdots,t_n\}$,其中 t_i 为 M 中的第 i 条转换,n 为 M 中的转换个数.

定义 2 头状态(Head State)记为 Head(t_i),表示转换 t_i 的头状态.

定义 3 验证序列集合(Verify Sequence Set)S_i 为第 i 个状态 s_i 的所有的验证序列的集合,而 S_i^k 为 s_i 的第 k 个验证序列. 验证序列可以是 D 序列、W 序列或者 UIO 序列.

定义 4 待测转换 t_i 对应的测试序列用例

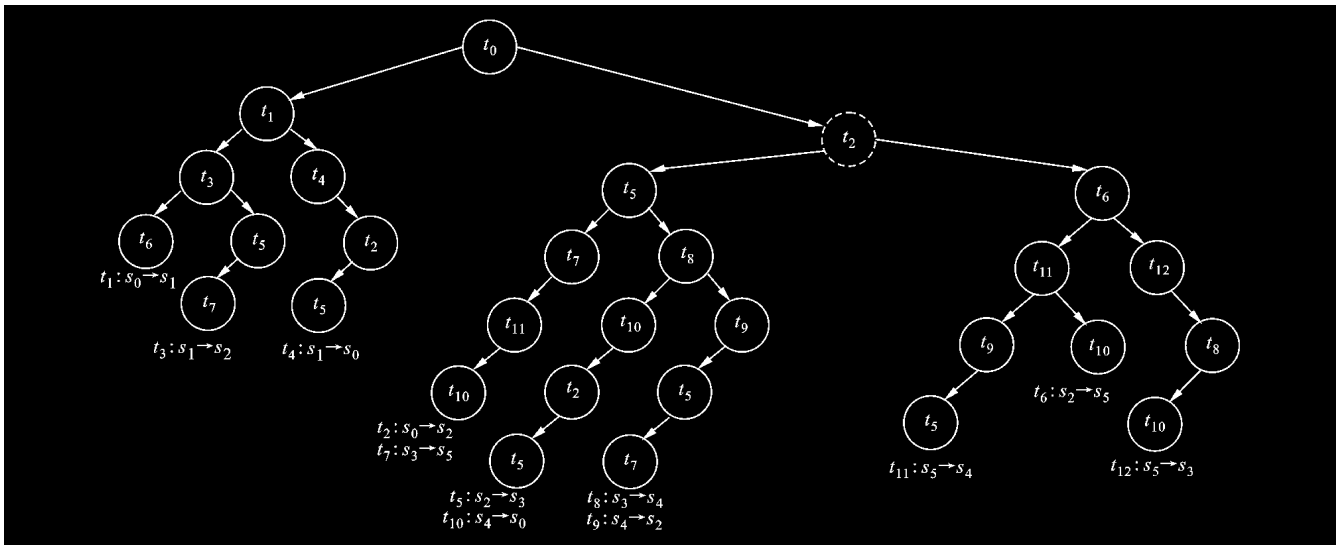

图 2　FSM 状态机 M 的测试树

$$T_i = \mathrm{Pream}(T_i) + t_i + \mathrm{Post}(T_i)$$

式中：$\mathrm{Pream}(T_i)$ 是将 IUT 从初始状态引导到 $\mathrm{Head}(t_i)$ 的输入输出序列；验证序列 $\mathrm{Post}(T_i)$ 是用于检验待测转换尾状态的输入输出序列.

定义 5　设 $T_i=\{t_{i1}, t_{i2}, \cdots, t_{ik}\}$，则 T_i 的子序列 $T'_i=\{t_{i1}, \cdots, t_{ij}\}$ 的执行函数 $\mathrm{Run}(T'_i)$ 表示将 T'_i 应用于待测实体 IUT.

定义 6　设 T 为测试集，则在测试用例 T_i 中转换 t_{ij} 的前缀序列

$$\mathrm{Prefix}(t_{ij}) = \begin{cases} (t_{i1}, \cdots, t_{i,j-1}) & 1 < j \leqslant k \\ \mathrm{NULL} & j = 1 \end{cases}$$

它表示在执行转换 t_{ij} 之前需要执行的转换，其中的 NULL 表示不需要执行任何转换.

定义 7　设 T 是测试集，则测试用例 T_i 和 T_j 的最大公共事件前缀

$$\max \mathrm{Prefix}(T_i, T_j) = \begin{cases} \mathrm{NULL}, & t_{i1} \neq t_{j1} \\ \mathrm{Prefix}(t_{ik}) + t_{ik}, & k = \\ \max(\{k \mid \mathrm{Prefix}(t_{ik}) = \\ \mathrm{Prefix}(t_{jk}) \wedge t_{ik} = t_{jk}\}) \end{cases} \tag{1}$$

表示 T_i 和 T_j 的最大的公共转换前缀集合，其中的 NULL 表示没有公共转换前缀.

定义 8　测试序列失败树（Test Sequence FAIL Tree，TSFT）F_j 表示在 TSFT 中有一条从根到叶子的路径，树中的结点属于 T 并且 $\mathrm{Run}(F_j)$=FAIL.

本文算法的核心思想是，对于转换 t_i，只要 t_i 的任一条测试序列通过，就认为 t_i 正确，而判断 t_i 出错则要穷尽 t_i 所有的引导序列和验证序列之后才能做出判断. 若是引导序列出错，则通过 Path_i 来构造新的有效测试序列；若是验证序列出错，则通过 S_i 构造新的有效测试序列. 为了减少总的测试序列数目，测试期间维护一棵树 F，构造新的测试序列时参照 F 以避免重复测试. 动态测试的伪码算法如图 3 所示.

3　算法分析

本文算法计算量最大的部分为寻找路径 Path_i^k 构造 T_i^k，也可以归结为在有向图中寻找 2 个节点间所有无环路径的问题. 利用宽度优先搜索算法求解最坏情况下的时间复杂度为 $O(n^n)$，其中 n 为状态机中的转换数目. 在实际测试中应基于以下考虑：

(1)实际协议的状态机并非完全图；

(2)寻找 Path_i^k 只在引导序列出错时才进行；

(3)在寻找有效的 Path_i^k 之前利用 F 剪枝.

实验结果表明，本文算法的平均时间复杂度为 $O(n^3)$.

最后，仍以图 1 中的 FSM 为例来说明算法的执行情况. 假设转换 t_5 有输出错误（如果是转移错误，分析方法同理），几个典型的测试流程如下.

(1)t_1→PASS.

(2)t_2→FAIL($\mathrm{Post}(T_2)$，$F=\{[t_2, t_5]\}$)→跳过 T_2^1, T_2^2, T_2^3，选择 $T_2^4, T_2^4=[t_2, t_6, t_{11}, t_{10}]$→$T_2^4$ 测试通过→t_2 标记为 PASS.

(3)t_8→FAIL($\mathrm{Pream}(T_8)$)→避开 F 构造最短的路径 $\mathrm{Path}_8^1=[t_2, t_6, t_{12}]$，$T_8^1=[t_2, t_6, t_{12}, t_8, t_9, t_5]$→FAIL($\mathrm{Post}(T_8^1)$，$F=\{[t_2, t_6, t_{12}, t_8, t_9, t_5]\}$)→选择 $T_8^2, T_8^2=[t_2, t_6, t_{12}, t_8, t_9, t_6, t_{12}]$→$T_8^2$ 测试通过→t_8 标记为 PASS.

```
FOR(∀ t_i∈T )
    IF (t_i已经被标记) CONTINUE;
    IF (Run(T_i)=PASS)
        标记 t_i 为PASS;
        CONTINUE;
    END IF
    IF (Run(T_i)=FAIL) //测试出错, 向F添加新的出错前缀
        错误序列 f =min((t_1,...,t_k) | (t_1,...,t_k)∈T_i& &Run((t_1,...,t_k)=FAIL));
        如果∀F_j∈F, 有max Prefix(f, F_j)≠F_j, 则F=F∪f; //验证序列出错, 利用S_i组合
        有效的测试序列T_i^k
        IF (f -Path_i^k -t_i≠∅)
          FOR(∀S_i^k∈S_i)
              T_i^k=Pream(T_i)+t_i+S_i^k, 并且∀F_j∈F, 有max Prefix(T_i^k, F_j)≠F_j;
              IF (Run(T_i^k)=PASS)
                  标记 t_i 为PASS;
                  BREAK;
              END IF
              T_i^k测试出错, 向 F 添加T_i^k的出错前缀, 代码略;
          END FOR
        END IF
        //引导序列出错, 利用Path_i组合有效的测试序列T_i^k
        IF (f∈Pream(T_i))
            Path_i=∅;//保存从初始状态到Head(t_i)的路径
            WHILE(∃初始状态到Head(t_i)的路径Path_i^k 且Path_i^k⊄Path_i)
                Path_i=Path_i∪Path_i^k;
                T_i^k=Path_i^k+t_i+Post(T_i), 并且∀F_j∈F, 有max Prefix(T_i^k, F_j)≠F_j;
                IF(Run(T_i^k)=PASS)
                    标记 t_i 为PASS;
                    BREAK;
                END IF
                T_i^k测试出错, ELSE向 F 添加T_i^k的出错前缀, 代码略;
            END WHILE
        END IF
    //引导序列和验证序列的所有匹配经遍历均为FAIL, 判断此转换错误
    IF (t_i 没有被标记)标记 t_i为FAIL;
    END IF
END FOR
```

图 3 动态测试的伪码算法

执行完毕之后，测试结果为 PASS 的转换数为 11，只有转换 t_5 的测试结果为 FAIL，其中 t_1、t_6、t_{12} 为直接测试通过，t_2、t_3、t_4、t_{11} 为验证序列出错，t_7、t_8、t_9、t_{10} 为引导序列出错. 对比而言，静态测试时只有 3 个测试序列的结果为 PASS. 为了能够证明本算法在实际中有效，也为了证明在大的状态机下动态测试可以取得较好的效果，本文在 TCP 协议的状态机上进行动态测试，结果如表2所示. 其中，考虑了所有可能出错的情况，并对其 PASS 的测试序列数目取平均值.

表 2 TCP 协议状态机的测试实验结果

错误转换数目	测试结果为 PASS 的平均数目		
	静态测试	动态测试	理想测试
1	14.1	16.8	18
2	11.4	14.9	17
3	9.1	13.1	16
4	7.2	11.3	15
5	5.4	9.1	14

从结果可以看出，动态测试能显著提高测试覆盖率，接近理想测试结果.

4 结 论

本文提出了一种动态协议测试方法，一方面充分考虑了已测结果对后续测试的影响，通过测试序列失败树指导测试序列的动态执行，有助于提高测试效率，另一方面也考虑了引导序列和验证序列对测试结果的影响，在避开测试序列失败树的前提下，在验证序列集合和 Path 集合中搜索并生成新的有效测试序列，显著提高了测试覆盖率.

参考文献：

[1] Lai R. A survey of communication protocol testing [J]. Journal of Systems and Software, 2002, 62(1): 21-46.

[2] Bowen J P, Bogdanov K. Fortest: formal methods and testing (2002) [C]// Proceedings of 26th IEEE Annual International Computer Software and Applications Conference. Los Alamitos, USA: IEEE Computer Society, 2002: 91-101.

[3] ISO. ISO 9646-2 Information processing systems, open system interconnection, OSI conformance testing methodology and framework, part 2: abstract test suite specification [S]. Geneva, Switzerland: ISO, 1994.

[4] Dahbura A T, Sabnan K, Uyar Ü M. Formal methods for generating protocol conformance test sequences [J]. Proceedings of the IEEE, 1990, 78(8): 1317-1326.

[5] Bochmann G V, Das G, Dssouli A, et al. Fault models in testing [C]// Proceedings of the IFIP IV International Workshop on Protocol Test Systems. Leidschendam, Netherlands: North-Holland, 1991: 17-30.

[6] 郭雄辉，赵保华，钱兰. 被动测试中的错误诊断算法 [J]. 中国科学技术大学学报，2005，35(3)：385-391.
Guo Xionghui, Zhao Baohua, Qian Lan. Fault diagnosis in passive testing [J]. Journal of University of Science and Technology of China, 2005, 35(3): 385-391.

（编辑　苗凌）