基于有限状态机的错误诊断算法

赵保华 钱 兰 周 颢 郭雄辉
(中国科学技术大学计算机科学技术系中国科学院计算机科学重点实验室 合肥 230027)

摘 要 基于有限状态机的协议的一致性测试问题已经得到了广泛的研究。在检测到错误后，如何诊断错误是一个很重要的问题。该文在有限状态机模型和单个错误的假设下，提出了一种新的错误诊断算法，该算法利用已经确定正确的转换信息以及可疑转换的下一个输入/输出对的头状态集合等信息来高效地诊断单个错误。文中给出了与已有的错误诊断算法的比较结果，并且用一个具体的实例来详细描述本文提出的算法。

关键词 错误诊断，一致性测试，有限状态机
中图分类号：TN393.04 **文献标识码**：A **文章编号**：1009-5896(2006)09-1679-05

Fault Diagnosis Algorithm Based on Finite State Machine

Zhao Bao-hua Qian Lan Zhou Hao Guo Xiong -hui
(*Computer Science and Technology Department, Laboratory of Computer Science Institute of Software Chinese Academy of SciencesUSTC, Hefei* 230027,*China*)

Abstract A lot of research work has been done for conformance testing of protocols based on the FSM model. After a fault has been detected, it arises the problem that how to diagnose the fault. In this paper, it is assumed that IUT (Implementation Under Testing) exists only one single fault and a fault diagnosis algorithm is proposed. The diagnosis algorithm makes full use of transitions confirmed to be correct and head states set of the next observed input/output pair to the symptom transition and guarantees the diagnostic of any single fault in an FSM. In this paper, the comparison of time complexity with other existing fault diagnosis algorithms is given and an example to demonstrate the algorithm is presented.

Key words Fault diagnosis, Conformance testing, Finite State Machine (FSM)

1 引言

协议一致性测试的目的是为了保证协议实现与协议描述一致，协议描述的控制流部分可以用有限状态机精确地描述。随着通信网络的发展，协议一致性测试变得越来越重要，因而基于有限状态机的测试问题得到了广泛的研究，文献[1]给出了一个很好的综述。基于有限状态机的测试序列的生成方法有很多种，常用的有T 方法，D方法，W方法和UIO方法[2]。但是在检测到错误后，如何诊断错误是一个比较复杂的问题。

Ghedamsi[3]最早研究了这个问题，并在单个错误的前提下给出了错误诊断算法，随后他将这个算法的思想扩展到通信有限状态机模型(Communicating Finite State Machines, CFSM)和多错误情况[4-7]；Lee在文献[8]中给出了另外一个诊断算法，Lee和Ghedamsi两人的算法思想非常相似，都是用枚举可能的错误情形并设计相应的判定实验来排除其中的一些错误，但是Lee的算法是基于自身生成的测试套的，而Ghedamsi的算法并没有限定测试套；Miller在文献[9]中利用逆向路径法给出了单个错误诊断集，该算法时间复杂度较低，Guo在文献[10]中给出了文献[9]的两个改进算法，能缩小错误诊断集合，但是存在一些这两个算法诊断不了的错误，因而有其应用限制。

在文献[4]中指出了诊断多个错误问题等同于标识机器问题，该问题在本质上是一个需要指数时间复杂度的算法才能解决的，所以本文假定协议实现中仅存在单个错误。本文提出了一种新的基于有限状态机模型的错误诊断算法，与前面介绍的已有算法相比，本文算法的时间复杂度最低，而且能给出完全的诊断信息。本文安排如下：第2节给出了一些概念的定义；第3节给出了具体的错误诊断算法描述、复杂度分析及其与其他算法的比较；第4节给出了一个具体的实例；最后总结全文。

2 术语定义

定义1 有限状态机(Finite State Machine, FSM)M是一个五元组：$M=(I,O,S,\delta,\lambda)$，其中I是输入符号集，O是输出符号集，S是状态符集，I，O，S均为非空集合；s_0是初始状态且$s_0\in S$；$\delta: S\times I\to S$是状态转移函数；$\lambda: S\times I\to O$是

2005-03-18 收到, 2005-10-17 改回
国家自然科学基金重大研究计划项目(90104010)，国家自然科学基金项目(60241004)和国家 973 计划项目(2003CB314801)资助课题

输出函数。

我们用 $t:s_i \xrightarrow{x/y} s_j$ 表示有限状态机M处于状态 s_i 时接受输入x，使状态转移到 s_j 并且产生输出y，s_i 称为转换 t 的头状态，s_j 称为 t 的尾状态。一般将有限状态机M用一个图 $G=(V,E)$ 来表示，图的顶点集合 V 代表状态机M的状态集，图的边集合代表M的转换集合，边上的标号是对应转换的输入/输出。

下面是本文算法考虑的两种错误，也就是错误模型：

(1) 输出错误：当一个转换发生时，输入和尾状态都符合描述，但输出不符合；

(2) 转换错误：当一个转换发生时，输入和输出都符合描述，但到达的尾状态不同。

假定实现中只存在单个错误(单个转换错误或者单个输出错误)，且已有测试套 $\mathrm{TS}=\{\mathrm{tc}_1,\cdots,\mathrm{tc}_p\}$，其中每个 tc_i 构成一个测试序列，包含有 m_i 个输入 $i_{i,1}i_{i,2}\cdots i_{i,mi}$，预期的输出序列记为 $o_i=o_{i,1}o_{i,2}\cdots o_{i,mi}$，其中 $o_{i,j}$ 表示在输入 $i_{i,j}$ 后的预期输出。用该测试套进行实际测试后得到每个测试序列的实际输出序列 $\hat{o}_i=\hat{o}_{i,1}\hat{o}_{i,2}\cdots\hat{o}_{i,mi}$。

定义 2 若在 o_i 和 $\hat{o}_i$ 中 $o_{i,j}\neq\hat{o}_{i,j}$ 则称为症状(symptom)，对应描述中的转换 $t_{i,j}$ 则称为可疑转换(symptom transition)。

定义3 如果所有症状对应同一个可疑转换，则将该可疑转换称为唯一可疑转换(unique symptom transition，ust)，由该唯一可疑转换产生的输出叫做唯一症状输出(unique symptom output，uso)。

定义4 观察到某个症状e时，测试序列所经过的转换构成该症状的冲突集，将该集合记为 $\mathrm{CS}(e)$。

定义5 若在 o_i 和 $\hat{o}_i$ 中 $o_{i,j}\neq\hat{o}_{i,j}$ 且对任意 $1\le k<j$ 有 $o_{i,k}=\hat{o}_{i,k}$ 则称 $o_{i,j}\neq\hat{o}_{i,j}$ 为测试序列 tc_i 的初始症状。

定义6 标号 x/y 的头状态集 $H(x/y)$={转换t的头状态|转换t的标号是x/y}。

3 诊断算法

3.1 错误诊断算法描述

下面给出本文的错误诊断算法，一共包括 3 个步骤。

第 1 步 比较已有测试套的预期输出和观察到的输出，生成所有的初始症状。

第 2 步 为每一个观察到的初始症状产生冲突集。这一步直观上的意义就是，在只存在单个错误的假设下，可能存在的错误肯定发生在症状之前。取这些冲突集的交集做为初始候选集(Initial Tentative Candidate set，ITC)。如果 ITC 中存在唯一可疑转换(ust)，则将 ITC 划分为两个非交子集合，一个仅包含唯一可疑转换(ust)，另外一个子集则称为最终候选集(Final Fentative Candidate set，FTC)；否则，FTC 即为 ITC。

第 3 步 如果存在 ust，首先处理 ust，否则把 ustset 设为空集。对 ust 的处理如下，遍历测试套中的所有转换，若转换是 ust 则检查观察到的输出是否是 uso，否则检查观察到的输出是否和预期的一致，只有在测试套 TS 中的所有转换对应的回答都为是的时候才将 ust 输出作为候选诊断集，直观上看这一步就是在检查 ust 是否能产生所有观察到的输出序列。处理 ust 的算法如下。

```
Procedure ust-processing(ust)
  For all  tc_m ∈ TS  Do
      For all  I_{m,n} ∈ tc_m  Do
          If ( t_{m,n} = ust ) Then
              If ( ô_{m,n} ≠ uso ) Then exit
          Else
              If ( o_{m,n} ≠ ô_{m,n} ) Then exit
      End
  End
  ustset ＝{ust}
```

处理FTC集合的算法主要思想是：遍历测试套中的所有测试用例，若某个测试用例中的某个转换 t_k 是FTC中还没处理过的元素，则根据该条测试用例所提供的信息来刷新FTC集合和转换 t_k 对应的错误尾状态集合 $\mathrm{EndStates}_k$，其中，t_k 的初始错误尾状态集合为 $\mathrm{EndStates}_k$ $\{s\,|\,s\in S\ \&\ s$不是t_k的尾状态$\}$。当所有测试用例均处理完后，FTC集合中的转换所对应的错误尾状态集合即为算法输出的最终错误诊断集合。直观上理解该算法是在检查FTC集合中的元素是否是可能的转换错误。

处理 FTC 的具体算法如下：

```
Procedure  FTC-processing(FTC)
  For all  t_k ∈ FTC  Do
      EndStates_k = {s | s ∈ S & s不是t_k的尾状态}
  End
  For all  tc_m ∈ TS  Do
      /*FTCUnprocessed为测试用例中还没处理的可疑转换的集合，初始值设为 FTC*/
      FTCUnprocessed = FTC
  For  all  I_{m,n} ∈ tc_m  Do
      If ( t_{m,n} = t_k ∈ FTCUnprocessed ) Then {
          /*根据实现中t_k后的下一个输入/输出对的头状态集合来缩小t_k的错误尾状态集合*/
          EndStates_k =EndStates_k ∩ H(I_{m,n+1} / Ô_{m,n+1})
          /*该算法提前结束的条件*/
          If (ustset = φ ) Then
              If  (|FTC| = 1 and |EndStates_j |=1
( t_j ∈ FTC )) Then exit
```

Else

If (FTC = ϕ) Then exit

/*对EndStates$_k$中的每个状态s，假定t_k的尾状态为s，如果测试例的期望的输出和观察到的输出不一致，就从EndStates$_k$中去掉s*/

For all state $s \in \text{EndStates}_k$ Do

$S_{\text{current}} = s_0$

For all $I_{m,n} \in \text{tc}_m$ Do

If ($\lambda(S_{\text{current}}, I_{m,n}) \neq \hat{o}_{m,n}$) Then

$\text{EndStates}_k = \text{EndStates}_k - \{s\}$; break

Else

If ($t_{m,n} = t_k$) Then

$S_{\text{current}} = s$

Else

$S_{\text{current}} = \delta(S_{\text{current}}, I_{m,n})$

End

End

/*把处理过的可疑转换从FTCUnprocessed中去掉；根据EndStates$_k$的值刷新FTC；如果测试例中不存在未处理的可疑转换，则提前结束该测试例的遍历 */

$\text{FTCUnprocessed} = \text{FTCUnprocessed} - \{t_k\}$

If ($\text{EndStates}_k = \phi$) Then FTC=FTC $- \{t_k\}$

If (FTCUnprocessed = ϕ) Then break

}

End

End

该错误诊断算法通过观察可能的错误对应的测试套的输出和实际观察到的测试套输出是否一致来逐步排除不可能的错误，从而得到最终错误诊断集合。由于实现中的错误对应的测试套输出跟实际观察到的测试套输出一致，所以实现中的错误一定包含在最终错误诊断集合里。

3.2 算法复杂度分析和比较

假定有限状态机的状态数目为n，测试套中包含L_s个测试用例，测试套中最长的测试用例所包含的输入数目为L_c。算法的第 1 步和第 2 步因为只需要扫描测试套一次即可，显然在 $O(L_sL_c)$ 时间内可以完成，算法第 3 步的过程ust-processing的双重循环亦只需要 $O(L_sL_c)$ 时间，由FTC的构造过程知其中包含的转换数目最大为L_c，因而过程FTC-processing中前 3 行的时间复杂度最大为 $O(L_c)$，第 4 行的循环变量最大值为L_s，第 6 行的循环变量最大值为L_c，第 13 行的循环变量最大值为n，第 15 行的循环变量最大值为L_c(代码中的注释不计算行)，因此该过程的其余代码时间复杂度最大为 $O(nL_s{L_c}^2)$。综上，本文算法的时间复杂度最大为 $O(nL_s{L_c}^2)$。

表 1 算法复杂度比较结果

Tab.1 Compare complexity of the algorithm with other algorithms

	本文算法	文献[9,10]中算法	文献[3]中算法
时间复杂度	$O(nL_sL_c^2)$	$O(n^2L_sL_c)$	$O(nL_sL_c^3)$

表 1 中给出了本文算法和和已有的一些单个错误诊断算法的时间复杂度比较结果。在状态数目 $n > L_c$ 时本文算法的时间复杂度是最低的，而在文献[3]中指出通常实际协议的测试用例中 $L_c \le 5$，由此n通常是大于L_c的，更何况文献[9,10]中的算法对某些重复错误不能给出诊断信息。综合考虑，本文的算法能够高效地诊断单个错误。

4 实例

图 1(a)是一个协议描述的示例有限状态机，图 1(b)是实现后的有限状态机，其中转换 $s2 \xrightarrow{a/0} s3$ 被错误地实现为 $s2 \xrightarrow{a/0} s5$ (在图中用虚线标识出来的)，测试套 TS＝{raab, rbaa, raaaa, rabba, rbaba, rbbab, rbaaab, rbabaa, rbbaaa, rbbbba, rbabaaa, rbabbba}，是根据 UIO 方法生成的，每个状态的 UIO 序列在表 2 中给出。

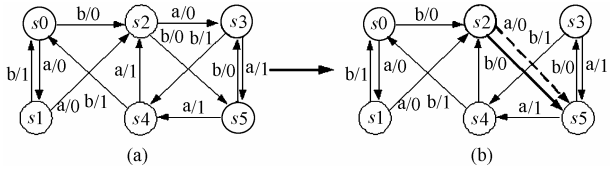

图 1　一个示例有限状态机描述及其实现

Fig.1　Specification and implementation of an example FSM

表 2　示例有限状态机的 UIO 序列

Tab.2　UIO sequence of each state in Fig.1(a)

状态	UIO序列	状态	UIO序列	状态	UIO序列
s0	b/0 a/0	s2	a/0 a/1	s4	a/1 a/0
s1	a/0 b/0	s3	b/1 a/1	s5	a/1 b/1

测试时各个测试子序列的预期输出和观察到的输出如表 3 所示。

下面，根据本文的算法给出错误诊断信息。

第 1 步　生成所有的初始症状，表 3 中已用下划线标出了所有的初始症状。这些初始症状为：tc5 中的初始症状 Symp_5，对应的可疑转换为 $s3 \xrightarrow{b/1} s4$；tc7 中的初始症状 Symp_7，对应的可疑转换为 $s4 \xrightarrow{b/1} s0$；tc8 中的初始症状 Symp_8，对应的可疑转换为 $s3 \xrightarrow{b/1} s4$；tc11 中的初始症状 Symp_{11}，对应的可疑转换为 $s3 \xrightarrow{b/1} s4$；tc12 中的初始症状 Symp_{12}，对应的可疑转换为 $s3 \xrightarrow{b/1} s4$。

表 3 测试套的预期输出和观察到的输出

Tab.3 Expected output and observed output of test suite

测试子序列	输入序列	预期输出	观察到的输出
tc1	raab	000	000
tc2	rbaa	001	001
tc3	raaaa	0001	0001
tc4	rabba	0100	0100
tc5	rbaba	0011	00<u>0</u>1
tc6	rbbab	0011	0011
tc7	rbaaab	00111	0011<u>0</u>
tc8	rbabaa	00110	00<u>0</u>11
tc9	rbbaaa	00110	00110
tc10	rbbbba	00011	00011
tc11	rbabaaa	001101	00<u>0</u>111
tc12	rbabbba	001100	00<u>0</u>110

第 2 步 对每一个观察到的初始症状产生冲突集，由此得到，

$CS\{Symp_5\}=\{ s0\xrightarrow{b/0}s2,\ s2\xrightarrow{a/0}s3,\ s3\xrightarrow{b/1}s4\}$

$CS\{Symp_7\}=\{ s0\xrightarrow{b/0}s2,\ s2\xrightarrow{a/0}s3,\ s3\xrightarrow{a/1}s5,\ s5\xrightarrow{a/1}s4,\ s4\xrightarrow{b/1}s0\}$

$CS\{Symp_8\}=\{ s0\xrightarrow{b/0}s2,\ s2\xrightarrow{a/0}s3,\ s3\xrightarrow{b/1}s4\}$

$CS\{Symp_{11}\}=\{ s0\xrightarrow{b/0}s2,\ s2\xrightarrow{a/0}s3,\ s3\xrightarrow{b/1}s4\}$

$CS\{Symp_{12}\}=\{ s0\xrightarrow{b/0}s2,\ s2\xrightarrow{a/0}s3,\ s3\xrightarrow{b/1}s4\}$

取这些冲突集的交集得到 $ITC=\{ s0\xrightarrow{b/0}s2,\ s2\xrightarrow{a/0}s3\}$，因为不存在唯一可疑转换 ust，所以 $FTC=ITC=\{ s0\xrightarrow{b/0}s2,\ s2\xrightarrow{a/0}s3\}$。

第 3 步 处理 FTC。初始的各个可疑转换的错误尾状态集合为：$EndStates(s0\xrightarrow{b/0}s2)=\{s0,s1,s3,s4,s5\}$，$EndStates(s2\xrightarrow{a/0}s3)=\{s0,s1,s2,s4,s5\}$。

首先对 tc1 进行处理，未发现可疑转换。

接着对 tc2 进行处理。tc2 的第 1 个转换就是可疑转换 $s0\xrightarrow{b/0}s2$，紧跟着该转换的输入/输出对为 a/0, 由此，我们得到 $EndStates(s0\xrightarrow{b/0}s2)=EndStates(s0\xrightarrow{b/0}s2)\cap H(a/0)=\{s0,s1,s3,s4,s5\}\cap\{s0,s1,s2\}=\{s0,s1\}$，这说明转换 $s0\xrightarrow{b/0}s2$ 对应的可能的错误的尾状态为 s0，s1。先假定转换 $s0\xrightarrow{b/0}s2$ 变为 $s0\xrightarrow{b/0}s0$，发现 tc2 的期望的输出 000 和实际观察到的输出 001 不一致，所以从 $EndStates(s0\xrightarrow{b/0}s2)$ 去掉 s0。再接着假定转换 $s0\xrightarrow{b/0}s2$ 变为 $s0\xrightarrow{b/0}s1$，发现 tc2 的期望的输出 000 和实际观察到的输出 001 不一致，因此从 $EndStates(s0\xrightarrow{b/0}s2)$ 中去掉 s1。此时，按照算法的流程可得 $FTCUnprocessed=FTC-\{s0\xrightarrow{b/0}s2\}=\{s2\xrightarrow{a/0}s3\}$，$EndStates(s0\xrightarrow{b/0}s2)$ 为空，所以将 $s0\xrightarrow{b/0}s2$ 从 FTC 中删除，FTC 因此变为 $\{s2\xrightarrow{a/0}s3\}$。继续处理 tc2 的输入，没有再发现可疑转换。

接着对 tc3 进行处理。初始的 $FTCUnprocessed=FTC=\{s2\xrightarrow{a/0}s3\}$。tc3 的第 3 个输入 a 对应的转换是 FTCUnprocessed 中的可疑转换 $s2\xrightarrow{a/0}s3$，紧跟着该转换的输入/输出对为 a/1，由此，我们得到 $EndStates(s2\xrightarrow{a/0}s3)=EndStates(s2\xrightarrow{a/0}s3)\cap H(a/1)=\{s0,s1,s2,s4,s5\}\cap\{s3,s4,s5\}=\{s4,s5\}$。先假设 $s2\xrightarrow{a/0}s3$ 变为 $s2\xrightarrow{a/0}s4$，发现 tc3 的期望的输出 0001 和观察到的输出 0001 一致，所以 $EndStates(s2\xrightarrow{a/0}s3)$ 保持不变。再假设 $s2\xrightarrow{a/0}s3$ 变为 $s2\xrightarrow{a/0}s5$，发现 tc3 的期望的输出 0001 与观察到的输出 0001 一致，所以 $EndStates(s2\xrightarrow{a/0}s3)$ 继续保持不变。此时 $FTCUnprocessed=FTC-\{s2\xrightarrow{a/0}s3\}=\phi$，所以停止对 tc3 的处理。

对 tc4 按上述步骤进行处理后，未发现可疑转换，$EndStates(s2\xrightarrow{a/0}s3)$ 不变。

对 tc5 的处理如下。初始的 $FTCUnprocessed=FTC=\{s2\xrightarrow{a/0}s3\}$。tc5 的第 2 个输入 a 对应的转换是 FTCUnprocessed 中的可疑转换 $s2\xrightarrow{a/0}s3$，紧跟着该转换的输入/输出对为 b/0，由此，我们得到 $EndStates(s2\xrightarrow{a/0}s3)=EndStates(s2\xrightarrow{a/0}s3)\cap H(b/0)=\{s4,s5\}\cap\{s0,s2,s5\}=\{s5\}$。此时，ustset 集合为空，FTC 中只有一个元素 $s2\xrightarrow{a/0}s3$，且 $s2\xrightarrow{a/0}s3$ 对应的 $EndStates(s2\xrightarrow{a/0}s3)=\{s5\}$ 只有一个元素，所以我们能够确定错误为：$s2\xrightarrow{a/0}s3$ 变成 $s2\xrightarrow{a/0}s5$。至此，诊断过程结束。

5 结束语

本文提出了一个适用于只存在单个错误的有限状态机的错误诊断算法。该算法因为利用了已经确定正确的转换信息以及可疑转换的下一个输入/输出对的头状态集合等信息，所以能够高效，正确地诊断有限状态机中的单个错误。相对于已有的一些错误诊断算法，本文提出的新算法的时间复杂度最低。本文算法最终给出的诊断信息是一些可能的错误集合，如果该集合中有多个元素，那么还需要生成新的测试序列来唯一地定位错误，这一步我们可以参考Ghedamsi[3]中的算法。

本文提出的算法只适用于存在单个错误的有限状态机，接下来的研究中我们希望提出一个适用于多错误的错误诊断算法。

参 考 文 献

[1] Lee D, Yannakakis M. Principles and methods of testing finite state machines——A survey. Proc. Of the IEEE, 1996, 84: 1090–1126.

[2] 龚正虎. 计算机网络协议工程. 长沙: 国防科技大学出版社, 1993.

[3] Ghedamsi A, Von Bochmann G. Test result analysis and diagnostics for finite state machines. Proceedings of the 12th International Conference on Distributed Computing Systems,

Yokohama, Japan, 1992: 244−251.

[4] Belhassine Cherif R, Ghedamsi A. Multiple fault diagnostics for communicating nondeterministic finite state machines. Proceedings of the Sixth IEEE Symposium on Computers and Communications, Hammamet, Tunisia, 2001: 661−666.

[5] Belhassine Cherif R, Ghedamsi A. Diagnostic tests for communicating nondeterministic finite state machines. Proceedings of the Fifth IEEE Symposium on Computers and Communications, Antibes, France, 2000: 424−429.

[6] Ghedamsi A, Bochmann G V, Dssouli R. Diagnosis of single transition faults in communicating finite state machines. Proceedings of the 13th International Conference on Distributed Computing Systems, Pittsburgh, Pennsylvania, USA, 1993: 157−166.

[7] Ghedamsi A, Bochmann G V, Dssouli R. Multiple fault diagnosis for finite state machines. Proceedings of INFOCOM '93, San Francisco, California, USA, 1993, 2: 782−791.

[8] Lee D, Sabnani K. Reverse-engineering of communication protocols. Proceedings of the International Conference on Network Protocols, San Francisco, California, USA, 1993: 208−216.

[9] Miller R E, Arisha K A. Fault identification in networks by passive testing. Proceedings of the 34th Annual Simulation Symposium, Seattle, WA, USA, 2001: 277−284.

[10] Guo X H, Zhao B H, Qian L. Fault identification by passive testing. Telecommunications and Networking - ICT 2004, Fortaleza, Brazil, 2004, 3124: 826−834.

赵保华： 男，1947 年生，博士生导师，主要研究方向为协议理论工程、无线传感网络.

钱 兰： 女，1981 年生，博士生，研究方向为协议理论与工程.

周 颢： 男，1976 年生，讲师，研究方向为协议理论工程、无线传感网络.

郭雄辉： 男，1981 年生，博士生，研究方向为协议理论与工程.