中国科学: 信息科学 2014 年 第 44 卷 第 5 期: 588–609
www.scichina.com info.scichina.com
《中国科学》杂志社
SCIENCE CHINA PRESS
论 文

一种基于扩展有限状态机的自动化测试用例生成方法

杨瑞[①②], 陈振宇[①], 张智轶[①], 刘子聪[①], 徐宝文[①②*]

① 南京大学计算机软件新技术国家重点实验室, 南京 210046
② 南京大学计算机科学与技术系, 南京 210046
* 通信作者. E-mail: bwxu@nju.edu.cn

收稿日期: 2013–07–03; 接受日期: 2013–12–20
国家重点基础研究发展计划 (批准号: 2014CB340702) 和国家自然科学基金 (批准号: 61170067, 61170071, 61373013) 资助项目

摘要 扩展有限状态机 (EFSM) 是使用最广泛的测试模型之一. 由于不可行路径的存在, 运用 EFSM 模型生成测试用例仍然是个难题. 本文提出了一种基于 EFSM 模型的自动化测试用例生成方法 (ATGEM). 为解决不可行路径问题, 首先提出一种基于数据流分析的路径可行性度量方法来预测路径的可行性, 以尽可能避开不可行路径, 提高测试用例自动化生成的效率. 然后通过建立动态可执行模型来获取运行时反馈信息作为搜索算法的适应度函数 (fitness function), 实现测试数据和预言信息的自动生成. 该方法结合静态分析和动态分析技术生成一个较优可行路径子集和对应测试用例来达到指定的覆盖准则, 能够应用于多种数据类型的测试用例生成, 适用范围较广. 通过实验在多个 EFSM 模型上验证了 ATGEM 方法中测试用例生成和路径可行性度量方法的有效性, 实验结果表明, 利用路径可行性度量方法可以大幅度提高测试用例生成效率, 与现有方法相比, ATGEM 中的测试用例生成方法具有更高的效率.

关键词 测试用例生成 扩展有限状态机 可执行模型 路径可行性 测试预言

1 引言

软件测试是软件开发过程的重要阶段. 软件测试花费可达整个软件开发过程花费的 30%∼50%[1]. 使用自动化测试方法可以提高测试效率、减少测试代价. 自动化测试已经成为当前软件测试发展的趋势, 而其中一个关键就是自动化测试用例生成. 自动化生成测试用例的一种常用方法是创建软件的测试模型, 并在模型的基础上自动化生成测试用例. 近年来, 基于模型的软件测试方法得到了快速发展, 引起了学术界和工业界的广泛关注. 有限状态机 (finite state machine, FSM) 和扩展有限状态机 (extended finite state machine, EFSM) 是其中 2 种应用最广泛的模型. EFSM 在 FSM 的基础上扩充了输入输出参数、上下文变量 (context variable)、定义在上下文变量及输入参数上的谓词条件和操作 [2], 它既可以表达控制流也可以表达数据流部分, 因而更加适合描述复杂的应用系统.

目前已有许多基于 FSM 的测试序列生成方法 [3∼7], 但基于 EFSM 模型的测试仍然存在许多挑战. 由于数据流和控制流之间的相互影响, 某些 EFSM 模型路径中存在的谓词条件不可能被满足, 从而导致路径不可行, 而 EFSM 模型路径可行性是个不可判定问题 [8]. 此外, 虽然已经存在一些基于

引用格式: 杨瑞, 陈振宇, 张智轶, 等. 一种基于扩展有限状态机的自动化测试用例生成方法. 中国科学: 信息科学, 2014, 44: 588–609, doi: 10.1360/N112013-00091

模型的测试数据生成方法, 但基于 EFSM 模型的测试数据生成技术仍然很不成熟. 在测试预言 (test oracle) 生成方面 (主要包括预言过程 (oracle procedure) 和预言信息 (oracle information), 本文主要讨论预言信息), EFSM 模型中不但包含条件判定, 还包含数据操作, 并且两者之间还可能相互影响, 这就导致 EFSM 不能像 FSM 那样较为直观地获得预言信息. 目前对于 EFSM 模型的测试预言自动化的研究也还相对较少. 一般而言, 测试数据 (test data)、测试预言和测试运行环境等构成了严格意义上的测试用例. 在很多文献中, 测试用例 (test case) 常常与测试数据 (test data) 等同使用. 在本文中, 测试用例包括测试数据和预言信息两部分.

针对上述问题, 本文提出一种新的基于 EFSM 模型的自动化测试用例生成方法 (automated test generation for EFSM method, ATGEM). 为了解决不可行路径问题, 我们提出了一种路径可行性度量方法来预测路径的可行性概率, 使得在测试用例生成过程中尽可能避开不可行路径. 为了在特定的可行路径上生成测试用例, 我们进一步通过建立动态可执行模型获取运行时反馈信息作为搜索算法的适应度函数 (fitness function), 实现测试数据自动生成和测试预言的自动创建. 以往的研究表明, 数量较少但较长的测试用例优于数量较多但较短的测试用例, 更容易发现系统中的错误 [9]. 因此该方法结合静态分析和动态分析技术来寻找一个具有较长路径和较小路径数量的较优可行路径集合, 以及对应的测试用例来达到指定的路径覆盖准则, 并识别多个扩展有限状态机模型的大量不可行路径. 本文是对我们早前工作的扩充 [10~12].

我们首先根据 EFSM 模型生成测试路径候选集 (路径中包括回路和自回路), 利用静态分析技术预先识别出部分谓词条件关系相对简单的不可行路径, 对其余复杂条件路径则利用提出的可行性度量方法计算可行性度量值来预测路径的可行性, 然后对候选路径集中的路径按可行性从高到低进行排序. 由于静态分析技术只能部分识别谓词条件关系相对简单的不可行路径, 因此还需要结合动态分析技术提高不可行路径识别率. 在此基础上, ATGEM 方法通过开发可执行模型, 利用路径遍历算法和语义解析技术使得静态的 EFSM 模型动态可执行. 可执行模型定义了模型的动态行为并通过使用语义执行使得静态模型具有类似程序的动态执行能力. 利用可执行模型就可以获取运行时反馈信息, 在此基础上我们提出了一种新的适应度函数, 并利用分散搜索 (scatter search, SS) 算法在排序后的候选路径集中挑选可行路径并自动生成相应的测试用例. 利用运行时反馈信息计算适应度函数, 就可以忽略不同数据类型带来的差异性问题, 使得该适应度函数能够使用在多种数据类型中, 因此适用范围较广. 可执行模型带来的另一个优点是可以在模型动态执行过程中记录相应的运算输出, 从而自动化的创建预言信息.

本文的贡献主要包括以下几个方面:

1) 提出了一种新的基于动态反馈信息的适应度函数计算方法和 EFSM 静态模型可执行化方法, 利用可执行模型获取动态运行时的反馈信息作为搜索算法的适应度函数, 实现测试数据自动生成和预言信息的自动创建. 相对于已有的方法, 我们提出的适应度函数计算较为简单有效, 适用范围较广.

2) 提出了一种新的路径可行性度量方法, 结合提出的测试数据自动生成方法来找到一个具有较长路径、较少路径数量的可行路径子集和对应的测试用例来达到指定的覆盖准则.

3) 通过多个 EFSM 模型的实验验证了该测试用例生成方法和路径可行性度量方法的有效性, 并检测出大量不可行的路径. 实验还表明利用路径可行性度量方法, 可以大幅度提高测试用例生成的效率, 与现有方法相比, ATGEM 的测试用例生成方法效率更高.

本文的组织结构如下: 第 2 节介绍扩展有限状态机模型的基本概念. 第 3 节介绍 EFSM 测试用例生成的相关工作. 第 4 节介绍自动化测试用例生成方法 ATGEM 的具体实现步骤. 第 5 节为实验研

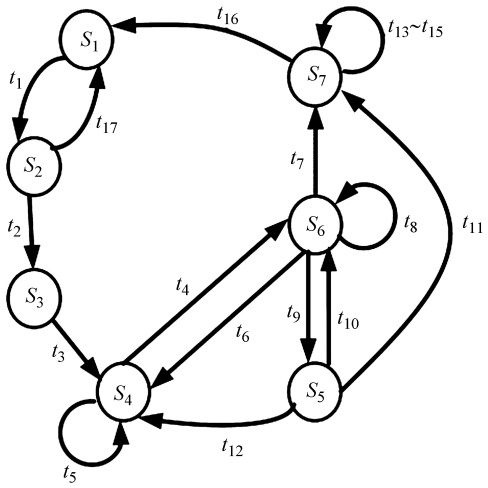


图 1　INRES 协议的扩展有限状态机模型迁移图 (修改自文献[13]中的图 1)

Figure 1　INRES protocol EFSM transition model (modified from Figure 1 in [13])

究与分析. 第 6 节进行总结和展望.

2　扩展有限状态机模型描述

扩展有限状态机模型由一个六元组 $M=(S, s_1, V, I, O, T)$ 构成, 其中 S 是一个有限状态集合, $s_1 \in S$ 是模型的初始状态, V 是内部变量的有限集合, I 是输入集合, O 是输出集合, T 是状态迁移的有限集合. 集合 T 的每个迁移 t 由一个六元组 $T=(s_i, s_j, a_t, o_t, P_t, A_t)$ 构成, 其中 s_i 是迁移 t 的起始状态, s_j 是迁移 t 的终止状态, $a_t \in I$ 是输入符, $o_t \in O$ 是输出符, P_t 是对当前变量值的谓词判定条件, A_t 是输出或赋值等一系列操作语句. P_t 和 A_t 在部分文献中也分别被称作守卫 (guard) 和行为 (action).

如图 1 所示, 扩展有限状态机开始时位于初始状态 $s_1 \in S$, 变量也被赋与相应的初始值, 记为 x_1. 若扩展有限状态机位于状态 s_i, 对应当前状态变量值为 x_i, 当接受输入 a_t 或输入事件, 并且 x_i 对于 P_t 是有效的, 也就是说 $P_t(x_i) = \text{true}$, 则 M 触发迁移 t 并且状态转换为 s_j. 在迁移发生过程中, A_t 操作可能改变或输出状态 s_i 的变量值, 也可能产生输出事件.

路径中触发某些迁移的谓词条件可能为永真而有些谓词条件则较为复杂难以满足. 当一条路径中 2 个迁移上的行为和守卫之间存在矛盾, 不能找到满足谓词条件的输入时, 该路径被认为是不可行的. EFSM 模型中路径可行性识别是个不可判定问题 [8]. 这些不可行路径的存在为基于扩展有限状态机模型的自动化测试带来了很大的困难.

如果扩展有限状态机中对任意一组迁移使用相同的输入触发迁移产生状态转换, 不存在多个迁移同时满足守卫, 则称该 EFSM 模型是确定性的 [14], 否则该 EFSM 模型是非确定性的. 本文讨论确定性有限状态机. 此外, 实际应用中有些扩展有限状态机模型存在特定的终止状态, 有些没有特定的终止状态, 即封闭的.

3　相关工作

基于 EFSM 的测试中, 在确定了特定的测试覆盖准则来满足测试的充分性后, 首先需要找到能够

满足指定测试覆盖准则的测试序列 (路径) 集合. 由于 EFSM 模型中数据流和控制流的相互影响导致存在不可行测试路径, 在生成测试用例的过程中如果能避开这些不可行路径, 则可以减少无谓的工作, 较大的提高测试用例生成的效率. 因此, 测试路径的可执行性是 EFSM 测试的关键问题之一.

通常测试序列的生成分为 2 个步骤:

1) 为 EFSM 的每个状态生成一个状态识别序列.

2) 生成满足指定覆盖准则的测试路径来检测迁移是否被正确触发, 并将该路径和状态识别序列连接, 形成完整的测试序列.

在状态识别方面, Ramaligom 等 [15,16] 提出了上下文无关唯一序列 (context independent unique sequence, CIUS) 用于 EFSM 的状态识别. 但并非每个 EFSM 都存在 CIUS 序列, 因此该方法的适用范围受到一定限制. Huang 等 [17] 提出了 UIO_E 方法用于 EFSM 状态识别序列的生成. 周晓煜等 [18] 则在此基础上引入了逆向判定性的概念对迁移可执行性分析 (transition executability analysis, TEA) 方法进行了改进, 用来缩短生成的测试序列的长度, 减小所需的扩展空间. Petrenko 等 [19] 使用配置 (状态向量) 确认序列 (configuration confirming sequence) 来解决扩展有限状态机的状态识别问题, 但嫌疑配置子集需要由用户来提供. 以上这部分的内容并不在本文的讨论范围之内.

在测试序列生成方面, 文献 [20~23] 分别提出了几种基于数据流覆盖准则的测试序列生成方法. 为了解决测试路径 (序列) 的可行性问题, Huang 等 [13,24] 提出了基于迁移可执行性分析 (transition executability analysis) 方法, 利用迁移可执行性分析, 以状态配置为节点, 以可执行迁移为边, 采用广度优先遍历算法将 EFSM 模型扩展生成一颗 TEA 树, 并由此生成一个可执行的迁移序列. Hierons 等 [25,26] 采用展开 EFSM 的方法来避免不可行路径问题, 国内的一些研究者如庞其祥 [27]、赵保华 [28] 等也使用了类似方法来解决 EFSM 模型的可执行问题, 但这类展开方法的缺点是可能引起状态的爆炸. Duale 等 [29,30] 将 EFSM 中 2 条边之间是否存在条件冲突的判定问题设定为一个不考虑代价函数的简化线性规划 (linear programming) 问题, 然后利用单纯形算法和图分裂技术来避免生成的测试序列不可执行, 这种方法只能用在所有行为和守卫为线性的情况. Chanson 等 [22] 则使用迁移自循环分析和约束满足问题 (constraint satisfaction problem, CSP) 来解决可执行问题, 而 Bourhfir 等 [31] 的方法在路径生成过程中就检查其可行性, 试图避免路径生成以后因为不可行而被丢弃, 但该方法缺少输入数据方面的考虑. Derderian 等 [32] 则通过给定路径中谓词操作符的类型和个数来评价路径的可行性. Kalaji 等 [33,34] 则通过数据流间的依赖分析, 并利用遗传算法来生成可执行路径. Yano 等 [35,36] 提出了使用多目标优化的方法来生成测试序列. 以上 3 种方法和我们的研究具有较高的相关性, 与 ATGEM 方法相比, 文献 [32] 中的方法考虑了谓词操作符的类型和个数, 但没有考虑测试路径长度的影响及测试数据的生成; 文献 [33,34] 中方法生成的路径长度需要用户事先指定, 生成的所有路径长度相同, 并且该方法较难考虑复杂的测试覆盖准则; 文献 [35,36] 中方法生成的测试序列是无序随机的, 未考虑路径和测试序列的对应关系, 因此生成过程中会产生冗余序列. 此外, 以上 3 种方法都没有考虑预言信息的自动化生成 (oracle information).

在生成能够满足指定测试覆盖准则的可行路径集合后, 还需要生成真实的测试数据来触发这些测试路径. 在 EFSM 测试中, 测试数据自动生成仍然是个难题. Chanson 等 [37] 使用了后向扩张的符号执行方法来获得路径条件表达式, 并提出了一种启发式方法来寻找约束系统的解. 张涌等 [38] 提出了一种基于 EFSM 模型的测试数据自动选取方法 (automated data selection, ADS), 该方法在假定给定路径可行的情况下, 通过区间削减和分段梯度最优下降算法来自动选取测试数据, 该方法在大多数情况下能够得到一组确定的解, 但在某些无法找到解的情况下, 只能通过手工选取的方式来选择测试数

据, 而且该方法中没考虑路径可行性问题. 目前, 动态测试数据生成方法已有不少, 但主要用于基于源代码的测试数据生成. 基于模型 (尤其是 EFSM) 的动态测试数据生成方法较少. Lefticaru 等 [39] 首先将遗传算法应用于 FSM 的测试序列生成. Kalaji 等 [40] 则在文献 [41,42] 的基础上提出了针对 EFSM 路径的适应度函数, 用于测试数据的生成. 与我们的 ATGEM 相比该方法适用的数据类型相对有限, 主要适用于数值类型和简单字符类型, 而 ATGEM 利用运行时反馈信息计算适应度函数, 可以忽略数据类型差异性带来的函数计算问题, 适用于多种数据类型. 张健等 [43] 首先将利用 C 语言子集编写的程序转换成 EFSM 模型, 然后利用图搜索、符号执行和约束求解技术生成可行的输入数据和路径. Wu 等 [44] 在此基础上首先利用搜索算法生成可行路径集合, 然后使用符号执行和约束求解技术来生成满足这些路径的数据. 赵瑞莲等 [45] 则通过实验发现测试数据的生成代价和一条可行路径中条件判定内数值等于运算符的个数 (number of numerical equal operators in conditions, NNEOC) 呈正比. 此外, 当 NNEOC 增加时, 测试生成代价与含有事件变量的路径长度 (length of path with events variables, LPEV) 或路径中含有的数值型事件变量的数量 (number of numerical event variables, NNEV) 这两者的相关性也变高, 并且 NNEV 和 LPEV 之间呈线性增长关系. 此外, 在 EFSM 主动测试 (active testing) 领域, 对于测试预言的研究还相对很少.

4 自动化测试用例生成方法

本文提出的测试用例自动化生成方法 ATGEM 主要步骤如下:

1) 首先从 EFSM 模型的起始状态通过图遍历算法生成候选路径集, 集合中的路径包含回路和自回路.

2) 利用静态分析技术移除候选路径集中能够被预先识别出的部分谓词条件关系相对简单的不可行路径, 对其余路径则利用提出的评估度量方法计算可行性度量值, 并按可行性从高到低进行排序.

3) 从路径候选集中按顺序选取一条路径, 若该路径已包含在测试用例子集中, 则丢弃, 转步骤 3), 否则转步骤 4); 若路径候选集中已无路径可选, 算法结束.

4) 生成能够触发该路径的测试用例, 若生成成功, 将路径及测试用例加入测试用例子集, 检查覆盖准则是否已经达到, 若是, 则算法结束, 否则转步骤 3).

4.1 候选路径集生成算法

ATGEM 方法首先需要从 EFSM 模型的起始状态开始生成候选路径集, 集合中的路径包含回路和自回路. 然后寻找满足特定覆盖准则的可执行路径子集并生成能够触发这些路径的测试数据和对应的预言信息. 生成的路径集合考虑 EFSM 中的回路和自回路部分, 由于路径中可能包含无限多个回路, 会导致生成的候选路径的数量无上界, 因此我们引入一个约束条件, 规定生成的路径中最多只包含 EFSM 模型中的回路或自回路一次. 该方法也可扩展用于生成包含特定次数回路的路径集合. EFSM 模型的多个状态序列构成了状态路径 (state path) 而多个迁移序列就构成了迁移路径 (transition path).

定义 1 扩展有限状态机的状态路径 (state path, SP) 是一个状态序列 $s_1 s_2 \ldots s_n$, 其中状态 s_i 与 s_{i+1} 之间存在一个或多个迁移, 其中 $1 \leqslant i \leqslant n-1$.

定义 2 扩展有限状态机的迁移路径 (transition path, TP) 是一个迁移序列 $t_1 t_2 \ldots t_n$, 迁移 t_i 的结束状态是迁移 t_{i+1} 的起始状态, 其中 $1 \leqslant i \leqslant n-1$.

```
1: Algorithm Path_Gen
2:   Input: store structure of EFSM model, roadList; state set S of EFSM where s0 is initial state;
3:   Output: state paths set contains all SP, SPS; transition paths set contains all TP, TPS;
4:   Goal: generate all transitions paths with one time loops from s0 to other states;

5:   back List:=ϕ;        //store the states have been traversed
6:   SP_Result:=ϕ;        //store the simple paths
7:   SP_L_Result:=ϕ;      //store the simple loop paths
8:   CirList:=ϕ;         //store the start-end state pairs of simple loops
9:   for each state sk in S
10:    {Getall_SP (s0, sk)}
11:      for (each start-end state pair t in CirList)
12:        //si is the start state of t where sj is end state, each t in CirList
13:        // indicates that there exists simple loops from sj to si
14:        SP_L_Result←Getall_SP (sj, si)
15:      for (each path spx in SP_L_Result)
16:          for (each path py in SP_Result)
17:              if py contains initial state of spx
18:                  SPS←insert spx into py before initial state of spx;
19:      Transform state paths set SPS into transition path set TPS;
20:    }

21: Getall_SP (start_node, destination_node)
22: { add start_node to backList;
23:   for (each transition t in roadList)
24:     // si is the start state of t where sj is end state
25:     {if (si==start_node)
26:         if (sj==destination_node)
27:           SP_Result←concatenate backList and destination_node;
28:         else
29:           if (backList not contains sj)
30:               Getall_SP (sj, destination_node);
31:           else
32:               CirList←start-end state pairs of simple loops;
33:     }
34:     Remove start-node from backList;
35: }
```

图 2 候选路径集生成算法

Figure 2 Candidate path set generation algorithm

由于实际应用的复杂性, 在 EFSM 模型中, 同样 2 个状态之间可能存在多个迁移, 因此状态路径与迁移路径之间可能是一对多的关系, 当迁移的起始状态和终止状态相同就形成自回路. 例如, 在图 1 中存在从起始状态 s_7 到终止状态 s_7 的 3 个自回路迁移 $t_{13}t_{14}t_{15}$, 而状态路径 $s_1s_2s_1$ 则构成了一个回路. 因此, 在候选路径集生成算法中使用了全组合算法来实现状态路径到迁移路径转换. 与基于代码的测试不同, EFSM 模型中的迁移可能是事件驱动的, 而并非如基于代码的测试由谓词来决定路径的执行分支, 因此在很多情况下需要预先生成测试路径.

路径生成过程以 EFSM 模型的起始状态作为所有路径的起始状态, 目的是在对实际系统的测试中可以通过对被测实现的重置 (reset) 来使被测系统重新处于初始状态并初始化内部变量的值, 便于输入新的测试用例. 候选路径集生成算法首先遍历 EFSM 模型来寻找从起始状态结点到其他结点的简单路径, 同时识别其中可能包含的简单回路. 简单路径指的是 EFSM 模型的一条路径中不包含 2 个或 2 个以上相同的状态, 与此类似, 简单回路指路径中除了具有相同的起始和终止状态外, 不含有其他相同的状态. 然后算法通过状态识别将每个简单回路插入简单路径中一次形成包含一次回路的状态路径候选集. 具体的候选路径集生成算法 Path_Gen 如图 2 所示.

在 Path_Gen 算法中有 2 种类型的路径被生成. 一种是状态路径 (SP), 存储在集合 SP_Result 和 SP_L_Result 中. 另一种类型的路径是迁移路径 (TP), 存储在集合 TPS 中. 算法首先调用函数 Getall_SP(s_0, s_k) 来获得从 s_0 到 s_k 的简单状态路径并存储在 SP_Result 中. 算法还识别简单回路并以 start-end 状态对的形式存储在 CirList 中, 表明存在从 end 状态到 start 状态的回路 (见 32 行). 然后 CirList 中的每个 start-end 状态对被用来生成从 end 状态到 start 状态的简单路径作为简单状态回路, 并通过状态识别将每个简单回路插入简单路径中 (见 15~18 行). 最后通过全组合算法将包含多迁移的路径转换为单迁移路径, 所有 SPS 中包含回路的状态路径被转换为包含回路的迁移路径存储在 TPS 中 (见 19 行). 例如, 图 1 中的状态路径 $s_1s_2s_3s_4s_6s_7s_7$ 可以转化为 3 条迁移路径 $t_1t_2t_3t_4t_7t_{13}$, $t_1t_2t_3t_4t_7t_{14}$, $t_1t_2t_3t_4t_7t_{15}$. 转换算法具体细节不再赘述.

4.2 路径可行性度量方法

基于 EFSM 测试的关键问题之一是测试路径的可行性. 尽可能挖掘迁移中谓词条件和操作之间存在的关系来预先识别或预测不可行路径, 并在生成测试用例的过程中避开这些不可行路径, 可以减少无谓的工作, 提高测试用例生成效率. 为了检测候选路径集中路径的可行性, 我们在 Kalaji 依赖分析建议值 [33] 的基础上提出一种评估路径可行性度量方法, 同时利用数据流分析技术来直接识别部分不可行路径.

定义 3 一条迁移路径 (TP) 被称作可行迁移路径 (feasible transition path, FTP) 当且仅当存在输入使其能够按顺序遍历 TP 中的每个迁移 t_i, 其中 $1 \leqslant i \leqslant n$, n 为路径的长度. EFSM 模型的迁移 t 中存在的守卫具有一般形式 $eg^{\mathrm{op}}e'$, e 和 e' 为表达式而 g^{op} 是关系运算符 (如 $<, >, \neq, =, \leqslant, \geqslant$). 给定一个表达式 e, $\mathrm{Ref}(e)$ 表示该表达式中的变量集合, 根据 e 和 e' 的类型, 迁移中的守卫 (guard) 可以归纳为以下 3 种类型:

1. g^{pv}: g^{pv} 为参量 p 与包含上下文变量 v 的表达式之间的关系运算符, 其中 $p \in \mathrm{Ref}(e) \cup \mathrm{Ref}(e')$.

2. g^{vv}: g^{vv} 为包含上下文变量 v 的表达式之间的关系运算符, $\mathrm{Ref}(e) \cup \mathrm{Ref}(e')$ 中的元素为上下文变量.

3. g^{vc}: g^{vc} 为包含常量 c 与上下文变量 v 的表达式之间的关系运算符, $\mathrm{Ref}(e) \cup \mathrm{Ref}(e')$ 中的元素为上下文变量而 e 或 e' 其中之一为常量.

EFSM 中的迁移 t 中存在的行为 (action) 一般形式为 $v := e$, 其中 v 为上下文变量而 e 为表达式. 对于上下文变量 v 的赋值操作可以归纳为以下 3 种类型:

1. $\mathrm{op}^{\mathrm{vp}}$: $\mathrm{op}^{\mathrm{vp}}$ 操作将一个依赖于参量 p 的值赋值给上下文变量 v, 其中 $p \in \mathrm{Ref}(e)$.

2. $\mathrm{op}^{\mathrm{vv}}$: $\mathrm{op}^{\mathrm{vv}}$ 操作将一个依赖于一个或多个上下文变量的值赋值给上下文变量 v, $\mathrm{Ref}(e)$ 中的元素为上下文变量.

3. $\mathrm{op}^{\mathrm{vc}}$: $\mathrm{op}^{\mathrm{vc}}$ 操作将一个常量值赋值给上下文变量 v, e 为常量.

定义 4 在一个迁移路径 TP 中存在定义 – 谓词 – 使用对 (definition-p-use pair)$\langle t_i, t_j \rangle$ 当且仅当 TP 中的迁移 t_i 存在变量 v 的赋值操作, TP 中的迁移 t_j 存在变量 v 的判定谓词使用, 且路径从 t_i 到 t_j 不存在对变量 v 的重新赋值定义, 其中 $1 < i < j < n$, n 为路径长度.

在对路径可行性进行评估时, 需要一个度量准则. 本文从以下几个方面来考虑迁移路径中存在定义 – 谓词 – 使用对 (definition-p-use pair) 对路径可行性的影响: 若一条路径的所有迁移中都不存在谓词, 那么该路径一定是可行的, 因为守卫 (guard) 默认都是可以满足的. 若路径的迁移中存在条件运算符, 那么 "$\neq$" 是最容易满足的运算符而 "$=$" 则是最难满足的运算符. 与此类似, 迁移路径存在的 definition-p-use 对中定义和使用的类型, 决定了存在变量定义的迁移对存在变量谓词使用的迁移中条件成立的影响程度. 例如, g^{pv} 是最容易满足的守卫 (guard) 类型, 因为可以通过寻找一个输入参数的值来尽量满足该守卫 (guard) 条件, 与此相反, g^{vc} 是最难满足的守卫 (guard) 类型. 此外, 迁移中行为 (action) 的类型也会对存在谓词使用的迁移条件是否成立产生影响, 特别是存在常量赋值操作情况. 对于部分存在常量操作的 definition-p-use 对, 可以通过静态分析方法直接判定路径的可行性. 我们给出如下性质:

性质 1 设 $\langle t_i, t_j \rangle$ 为路径 TP 的 definition-p-use 对, t_i 定义变量 v 的值为常量 C_1, 而 t_j 的守卫 (guard) 为变量 v 和常量 C_2 的关系运算, 若出现以下 6 种情况之一, 则路径不可行:

Case 1. 当 t_j 中的关系运算符 g^{vc} 为 "$=$", 并且常量 $C_1 \neq C_2$.

Case 2. 当 t_j 中的关系运算符 g^{vc} 为 "$>$", 并且常量 $C_1 \leqslant C_2$.

表 1 迁移路径中 definition-p-use 对的建议惩罚值

Table 1 The suggested penalty values of definition-p-use pair

Guard	Action		
	$\mathrm{op}^{\mathrm{pv}}$	$\mathrm{op}^{\mathrm{vv}}$	$\mathrm{op}^{\mathrm{vc}}$
$g^{\mathrm{pv}}(=)$	8	6	24
$g^{\mathrm{pv}}(<,>)$	6	12	18
$g^{\mathrm{pv}}(\leqslant,\geqslant)$	4	8	12
$g^{\mathrm{pv}}(\neq)$	2	4	6
$g^{\mathrm{vv}}(=)$	20	40	60
$g^{\mathrm{vv}}(<,>)$	16	32	48
$g^{\mathrm{vv}}(\leqslant,\geqslant)$	12	24	36
$g^{\mathrm{vv}}(\neq)$	8	16	24
$g^{\mathrm{vc}}(=)$	30	60	500 if c is different and 0 otherwise
$g^{\mathrm{vc}}(<,>)$	24	48	0 if c is different and 500 otherwise
$g^{\mathrm{vc}}(\leqslant,\geqslant)$	18	36	500 if c is different and 0 otherwise
$g^{\mathrm{vc}}(\neq)$	12	24	0 if c is different and 500 otherwise

Case 3. 当 t_j 中的关系运算符 g^{vc} 为 "$<$", 并且常量 $C_1 \geqslant C_2$.

Case 4. 当 t_j 中的关系运算符 g^{vc} 为 "$\geqslant$", 并且常量 $C_1 < C_2$.

Case 5. 当 t_j 中的关系运算符 g^{vc} 为 "$\leqslant$", 并且常量 $C_1 > C_2$.

Case 6. 当 t_j 中的关系运算符 g^{vc} 为 "$\neq$", 并且常量 $C_1 = C_2$.

以上性质可以通过反证法来进行证明, 这里只给出 Case 2 证明过程, 其他情况利用相同的方法也可以得到. 证明: 根据 Case 2, 在迁移 t_i 中存在对变量 v 的赋值定义 $v := C_1$, 迁移 t_j 中则存在对变量 v 的关系运算 $v > C_2$. 若路径可行, $v > C_2$ 的值为 true, 则可得 $C_1 > C_2$, 这与 Case 2 假设条件 $C_1 \leqslant C_2$ 矛盾; 因此 $v > C_2$ 不成立, 包含 definition-p-use 对 $\langle t_i, t_j\rangle$ 的路径不可行; 证毕. 除了上述情况, 对于包含复杂条件的路径, 评估路径的可行性需要考虑 definition-p-use 对之间行为 (action) 和守卫 (guard) 的类型, 不同操作类型的建议惩罚值 [33] 情况如表 1 所示.

以上考虑的是单操作的情况, 实际应用中一个迁移中的守卫 (guard) 可能存在多个关系表达式, 这些关系表达式通常利用逻辑运算符进行连接. 在这种情况下, 当使用 "AND" 逻辑运算符进行连接时, 则将各惩罚值相加; 当使用 "OR" 运算符时, 则取其中最小的惩罚值.

在软件测试中常选取一种或多种测试覆盖准则来生成对应的测试用例, 并衡量测试的充分性. 软件测试的覆盖准则有很多, 对于基于 EFSM 模型的测试主要使用基于状态/迁移的覆盖准则, 如全状态覆盖 (all-states coverage)、全迁移覆盖 (all-transitions coverage)、全迁移对覆盖 (all-transition-pairs coverage) 和全一次循环路径覆盖 (all-one-loop path coverage) 等 [46], 用于指导可执行迁移路径的生成. 测试覆盖标准的选择是一个权衡的过程, 所选的标准越强, 对被测系统实施的检测就越全面, 所需的测试用例也往往越多. 所以选择测试覆盖标准的时候需要对测试代价和测试效果进行权衡. 以往的研究成果还表明, 数量较少但较长的测试用例相比数量较多但较短的测试用例更容易发现系统中的错误 [9]. 因此, 为了找到一个含有较长路径的较小可行路径集合来达到指定的路径覆盖准则, 我们提出以下路径可行性度量模型来达到 all-transitions 覆盖, 该度量模型经过简单修改后, 也可以满足其他覆

盖准则.

$$f=\begin{cases}\dfrac{\sum_{i=1}^{k}v(df_i)}{|\mathrm{TP}|^{\mathrm{d}}}, & \sum_{i=1}^{k}v(df_i)\neq 0,\\ 0-|\mathrm{TP}|, & \sum_{i=1}^{k}v(df_i)=0,\end{cases}$$

其中 df_i 是迁移路径中的 definition-p-use 对, $v(df_i)$ 则为表 1 给出的 2 个迁移间相互影响的建议惩罚值, 而 k 是路径中存在的 definition-p-use 对的个数, |TP| 是路径的长度, d 是用来调整路径长度对评估影响的权值. f 的值越小, 说明路径的可行性概率越大; 反之, 路径的可行性概率就越小. 由于越长的路径可能存在越多的谓词条件和 definition-p-use 迁移对, 导致的取值可能越大, 因此使用 |TP| 来平衡路径长度与惩罚值之间的矛盾. 当一条路径中 definition-p-use 迁移对惩罚值的和为 0, 说明路径中的迁移不存在 definition-p-use 依赖关系, 路径具有最好的可行性概率, 这时越长的路径越容易覆盖更多的迁移, 因此 f 的取值设为 0∼ |TP|. 该度量准则也可较容易的扩展到其他的覆盖准则, 如通过判定路径中相同迁移的数量作为权值来达到 all-one-loop 覆盖准则.

在路径可行性评估策略中, 我们使用后向遍历算法来识别候选路径集 (transition path set, TPS) 中每条路径的 definition-p-use 对, 并利用式 1 计算 f 的值作为路径可行性度量. 在评估过程中, 若路径中的 definition-p-use 对符合性质 1 中所述 6 种情况之一, 则从候选路径集 TPS 移除该路径. 然后对 TPS 中的路径根据评估值的大小按升序进行排序, 这样 TPS 中排在前面的路径具有更好的可行性概率. 排序后的 TPS 被最终用来选择生成一个较优的可行路径子集并生成相应的测试数据和预言信息.

4.3 测试数据及预言信息生成

本节引入动态分析技术来进一步检测不可行路径. 本文主要通过搜索算法来实现动态检测. 与静态分析技术不同, 动态分析技术并不能够直接检测不可行路径, 通常的策略是限制搜索的深度或迭代的次数, 也就是说如果通过搜索最终未能找到能够遍历指定路径的输入数据, 则认为该路径是不可行的 [47]. 因此, 动态分析的过程可以和测试数据生成过程相结合. 与动态分析技术相比, 静态分析技术的效率较高, 但只能检测部分谓词相对简单的不可行路径. 由于不可行路径检测是不可判定问题, 因此对于含有复杂条件的路径, 目前只能通过动态分析技术来检测.

4.4 建立可执行模型

一般的模型通常是静态的, 以抽象图表或图形的形式存在, 不具备动态行为. 可执行模型是一个能够模拟真实系统动态行为的模型系统, 可执行模型甚至可以作为一个原型系统来进行测试. 与静态模型不同, 可执行模型定义了动态行为作为模型的一部分. 通过执行语义, 可执行模型可以像程序代码那样直接动态的执行.

为了让一个静态的 EFSM 模型可执行化, 对于迁移中表达式的语义分析就成了构建动态模型的关键. 当守卫 (guard) 中的条件表达式和行为中的表达式能够进行语义上的解析, 含有输入参数、上下文变量以及事件的测试数据就可以作为迁移中表达式的输入进行执行并获得相应的输出. 在此基础上, 结合路径遍历算法就可以使得静态 EFSM 模型具有类似程序的动态执行能力. 构建可执行模型后, 就可以在测试数据的生成过程中通过观察执行的轨迹来验证测试数据的有效性. 为了构建可执行模型, 我们利用 Java 语义解析库 JEva[1)]来解析 EFSM 模型的静态表达式. JEval 是可进行表达式

1)http://jeval.sourceforge.net.

解析和运算的 (application programming interface, API) 资源包, 可以用来在运行时解析数学表达式、布尔表达式、赋值表达式、字符串表达式以及函数表达式等多种表达式, JEval 支持绝大部分的运算符. 当 EFSM 模型处于拥有对应配置变量值的特定状态并且获得相应输入数据和输入事件, 若守卫 (guard) 被解析并且条件成立, EFSM 模型就可以激发相应的迁移并转入下一状态. 在迁移发生时, 行为 (action) 中的表达式也可以被 JEval 解析, 从而改变当前变量的值并获得相应的输出值或输出事件. 这些值被记录在一个对应的数据结构中, 以便在触发下一个迁移时使用.

4.5 运行时信息反馈

运行时信息反馈指的是在测试执行过程中动态的收集运行信息, 该技术由 Miller 和 Spooner[48] 率先提出, 用于在基于代码的测试中获得运行时反馈信息. 构建可执行模型之后, 就可以利用模型的动态执行来获取运行时反馈信息用于搜索算法的适应度计算, 以此来指导测试数据的生成. 生成的测试数据能够遍历对应的路径, 在生成测试数据的过程中还可以动态的识别不可行路径. 可执行模型带来的另一个优势是在模型动态执行过程中可以解析并记录相应的输出, 从而自动化的创建测试预言. 从广义上来说测试预言由预言信息和预言过程 2 部分构成 [49]. 预言信息指获得系统的预期输出, 即获得抽象模型或规约在输入特定测试数据时所期望的输出. 但在测试中, 如基于模型或规约的测试, 除了生成预言信息外, 还需要在实际的被测系统 (system under test, SUT) 中根据相应的预言信息来检验测试执行结果的正确性. 运用预言信息来判定系统在实际运行中的行为或动作是否正确就称为预言过程. 本文所指的测试预言是狭义上的测试预言, 也就是预言信息. 在 EFSM 测试领域, 对于测试预言的研究还相对较少. 在主动测试 (active testing) 中, 首先需要生成包含预言信息的测试用例. EFSM 的抽象行为在某种程度上更像一个静态的程序, 不同的是静态的 EFSM 并不具有程序的动态执行能力. 然而 EFSM 不但包含条件判定, 还包含对于数据的操作, 并且两者之间还可能存在相互影响, 这就导致 EFSM 不能像 FSM 那样较为直观的获得预言信息. 但通过构建可执行模型则可以较为方便的获得预言信息.

在生成测试用例时, 首先需要通过特定方法生成测试数据作为起始的种子数据, 并且立即将该数据作为测试输入在可执行模型中动态运行, 在模型执行过程中通过获得的反馈信息来指导测试数据的搜索来生成新的、改进的测试数据. 在 EFSM 模型动态执行的过程中, 我们收集如下反馈信息作为搜索算法的适应度函数:

$$F_e = \frac{|SP_t|}{|\mathrm{TP}|} \times 100, \tag{1}$$

其中 |TP| 是指定用来生成测试用例的路径的长度 (这里长度是指路径中包含的迁移的数量), $|SP_t|$ 是当前测试用例从源状态结点开始能够连续触发的路径长度, 该方法中反馈信息的计算代价较小. 当 F_e 的值为 100, 表示路径中所有的迁移都被触发, 则自动化生成了相应的测试数据. 利用运行时反馈信息计算适应度函数, 不涉及到数据类型带来的计算差异性, 使得该适应度函数能够使用在多种数据类型的测试用例生成中.

4.6 分散搜索算法框架 (scatter search, SS)

分散搜索 (scatter search, SS) 算法是一种基于种群进化的元搜索算法框架 [50], 为解决优化问题提供了一种方法. 算法框架由多个不同的子算法组成, 可以根据需要选择和修改子算法及其参数来应用于不同优化问题. SS 算法首先通过多样化方法产生初始种群 (population set), 然后在初始种群中选

```
1: Algorithm TS_Gen
2: Input: sorted TPS;
3: Outputs: test suit Resultset and its corresponding feasible paths
subset of TPS;
4: Corresponding outputs set O_p when generating test cases.

5: for each path p_t in sorted TPS
6: {
7:     if (all transitions of model are covered) exit;
8:     if (p_t contains transitions not be covered)
9:         {Tempset←SS_Path_TC(pt);
10:          if (Tempset covered p_t)
11:              { add Tempset into Resultset and record p_t;
12:                  update transition coverage information;    }}
13:   else
14:          skip and get next path;
15: }


16:Solution set SS_Path_TC (transition path p_t)
17:{ Generate an initial improved test cases Solution set P for p_t;
18:   while (iteration times<max iteration times)
19:    {     Apply populations in P to executable model;
20:          Refset←m best solution of P set;
21:          if (all paths value equals 100)
22:             {   O_p←outputs of executing populations;
23:                 return Refset;
24:             }
25:          while (B←Combine solutions from RefSet and B≠φ)
26:             { Improve solution in B;
27:               Apply populations in Refset to executable model;
28:               RefSet←keep m best from RefSet∪B;
29:               if (all paths value equals 100)
30:                  {   O_p←outputs of executing populations;
31:                      return Refset;
32:                  }
33:             }
34:           clear P set;
35:           P set←RefSet;
36:           fill the P set with diverse solutions;
37:      }
38:    return Refset;
39:}
```

图 3　测试数据及预言信息生成算法

Figure 3　Test data and oracle information generation algorithm on executable EFSM model

择部分质量最优的解和多样化特性最好的解组成一个较小的参考集 (reference set); 接着将这些解通过组合方法产生新解集合, 并通过提升方法对新解进行质量改进; 利用新解来更新参考集以进行新的迭代搜索.

SS 算法与遗传算法 (genetic algorithm, GA) 以及其他进化搜索算法的主要区别在于遗传算法的初始种群数目较多, 新组合的产生一般通过随机抽样来选取; 而 SS 算法参考集中解的数量则相对较少, 用系统化的方法选择两个或多个解进行组合产生新解. SS 算法中还可以使用局部搜索方法来改善解的质量. 算法在考虑解质量的同时也考虑了解的多样性. Rafael Martí等 [51] 还通过实验对 SS 算法和 GA 算法进行了比较, 结果表明 SS 算法在解决一些问题时候具有更好的表现. SS 算法框架主要由 5 个部分构成:

1) 多样化生成方法 (a diversification generation method): 生成解的种群作为建立初始参考集的候选集 (population set), 解应该尽量的多样化, 这样才能散布于整个搜索空间.

2) 提升方法 (an improvement method): 该方法利用局部搜索方法来改善新产生的解.

3) 参考集更新方法 (a reference set update method): 用来创建一个参考集 (reference set) 来存放高质量和多样化的解, 并用于解组合方法来产生新的解, 初始的参考集 (reference set) 由部分最优解和部分多样化解构成.

4) 子集生成方法 (a subset generation method): 这个过程生成用于组合的解子集, 在分散搜索算法框架中, 每个子集中的解都是可以用于合并的候选解.

5) 解组合方法 (a solution combination method): 用一个或多个参考集中的解通过交叉算子生成新的解.

SS 算法原则上要求尽量避免随机操作, 但在测试用例生成过程中, 我们并没有完全遵循这一指导原则, 因为一定程度上的随机在进化算法和其他一些研究领域能够获得性能上的改善 [52]. 在参考集更新方法中, 使用了式 2 中的 F_e 反馈信息来度量参考集中解的优劣, 最优的那部分解被保留在参考集中, 最差的部分则从参考集中移除, 同时引入部分新的解. 利用分散搜索算法框架, 并利用模型可执行化技术, 就可以获得运行时反馈信息来计算上节提出的适应度函数, 实现测试用例的自动化生成, 算法具体过程如图 3 所示. TS_Gen 算法首先从排序后的 TPS 中按顺序选择一条路径检查该路径中的

表 2 实验目标模型属性

Table 2 The property of objective model for empirical study

Object model	State number	Transition number	Input parameters
ATM	9	23	6
INRES	8	18	2
CLASS2	6	21	10
SCP	4	8	4

迁移是否已被指定的测试充分性准则覆盖, 若所有迁移都已经被覆盖, 则跳过该路径选取候选集中的下一条路径. 否则利用开发的可执行模型和收集的动态反馈信息, 结合分散搜索方法 Scatter Search, 对选择的路径进行语义执行, 并利用反馈信息搜索状态空间中的解. 若在规定迭代次数范围内找到解集则将结果加入测试用例结果集. 若在达到规定的迭代次数后, 仍未能找到最终的解, 则认为该路径是不可行的. 函数 SS_Path_TC 用来为指定的路径 p_t 生成测试数据和预言信息, 运行时反馈信息 (见式 2) 被用于指导搜索的过程. 如果 F_e 的值等于 100, 则说明能够触发路径 p_t 的测试数据已经生成, 该测试数据对应的执行输出则被记录在 O_p 中用来创建预言信息.

5 实验与分析

5.1 实验目标模型

我们设计了一系列的实验, 将提出的方法应用于实际的 EFSM 模型, 用来检验 ATGEM 方法中测试用例生成和路径可行性度量方法的有效性及效率, 并通过实验识别实际 EFSM 模型中的不可行路径, 分析原因. ATGEM 方法利用 Java 语言及 Eclipse 3.5 开发平台 +JDK1.6 来实现, 实验运行的平台是 Windows 7 32 位操作系统. 计算机的硬件配置为 Intel Core2 Quad Q8400 CPU, 主频 2.66 GHz, 4 G 内存. 为了使实验结果更具一般性, 我们选择了 4 个较为典型 EFSM 模型, 分别是: (automated teller machine, ATM)[53], (initiator responder protocol, INRES) protocol[13], Class 2 transport protocol[16] 和 (secure copy protocol, SCP) protocol[54]. 以上目标模型收集自目前已有的研究, 且迁移之间的逻辑关系相对复杂. 其中 ATM 模型包含明确的终止状态, 而其他 3 个模型则形成环路, 没有明确的终止状态. 对 ATM 模型的实验没有设定路径的结束状态必须是模型规定的终止状态, 因为我们认为模型规定的终止状态为路径的终止状态只是路径中的一个特例. 表 2 中列出了 4 个目标模型对应的属性, “状态” 栏是指目标模型存在的状态数量, “迁移” 栏是指 EFSM 模型中迁移的数量, “输入参数” 栏是指触发模型行为属性所需要的输入参数的个数, 也就是需要生成的测试用例数量.

5.2 实验研究问题及步骤

通过设定的实验主要是为了回答以下几个研究中的问题:

(RQ1)ATGEM 方法的有效性及效率如何?

(RQ2) 路径可行性度量方法预测路径可行性的效果如何?

(RQ3)ATGEM 方法中静态分析技术与动态分析技术在识别不可行路径方面效果如何?

实验首先将 4 个目标模型以文件方式进行存储, 然后依次调入我们实现的程序进行处理. 程序首先生成路径候选集, 然后对集合中的路径进行分析, 预测路径的可行性, 在此基础上最终生成一个达

表 3　SS 算法的主要配置参数

Table 3　Main configurations of SS algorithm

Properties	Values
Test Cases Solution set P size	50
Refset size	2
Max iteration times	20000
SBXCrossover probability	1.0
SBXCrossover Distribution Index	20.0
Polynomial Mutation Operator	$1/n$ (n equals to the number of decision variables)

表 4　测试用例生成及覆盖情况结果

Table 4　The results of test case generation and coverage ratio

Object model	Number of TPS	FTP number of result set	Covered transitions	Coverage ratio(%)
ATM	349	16	22	95.6
INRES	109	10	18	100
CLASS2	334	16	20	95.2
SCP	29	2	6	75

到指定覆盖的测试用例集. 为了验证路径可行性度量方法及测试用例生成方法的有效性, 我们还对程序进行了一些修改, 用来在运行中额外记录一些实验统计数据. 实验中 SS 算法的主要配置参数如表 3 所示.

表 3 中 Max iteration times 指的是 SS_Path_TC 函数中对每个新解的评估次数, 也就是最大迭代次数, 新解由多样化生成 (diversification generation)、子解集生成 (subset generation) 和解提升方法 (improvement method) 产生. 在 SS 算法中我们使用了 SBXCrossover 交叉算子和 Polynomial Mutation Operator 变异算子, SBXCrossover 算子的交叉概率和分布指数 (distribution index) 分别设置为 1.0 和 20, Polynomial Mutation 算子的变异概率为 $1/n$(n= 解变量个数). 测试覆盖准则设定为基于模型的测试中常用的 All-transitions 覆盖. 可行性度量方法 (请参见式 1) 中参数 d 的值设置为 0.8. 在运行时间上, 由于搜索算法在同一数据集上每次运行的时间会略有差异, 为了获得更准确的数据, 实验对每个模型分别运算 10 次并取运行时间的平均值.

5.3　实验结果与分析

为了回答 (RQ1) 中 ATGEM 方法的有效性问题, 我们从以下 3 个方面来进行衡量:

a) 是否能够从候选路径集中挑选出相应的可行路径子集, 并在这些可行路径上生成测试数据和预言信息;

b) 生成的测试用例集 (包括可行路径路径、对应的测试数据和预言信息) 能否达到指定的覆盖;

c) 挑选出的可行路径子集数量和长度情况.

实验总体运行结果如表 4 所示. 表 4 中 “候选路径数量” 栏指生成的候选路径集中路径的数量, 从表中可以看出 ATM 模型和 CLASS2 模型生成的候选路径集相对较大, 这是因为这 2 个模型中相同起始和终止状态之间存在多个迁移的情况较多, 因此组合后的迁移路径数量相对较多. “测试用例结

果集中可行路径数量" 栏是指最终选定的测试用例集中可行路径的数量, "迁移覆盖数量" 栏是指被测试用例结果集中可行路径覆盖的迁移数量. "迁移覆盖率" 栏是指模型中全部迁移被覆盖的百分比.

经过实验发现, 在所有的实验模型上, ATGEM 方法都能够从候选路径集中挑选出相应的可行路径子集, 并能成功的在这些可行路径上生成测试数据和预言信息. 就覆盖情况而言, 生成的测试用例结果集总体上达到了较高的覆盖. 有些模型并未能达到 100% 覆盖, 主要是因为这些模型中的部分迁移对循环的次数有特定的要求, 例如在 ATM 模型中迁移 t_3 未能被覆盖是因为只有当 t_3 前的自回路迁移 t_2 被触发 3 次以后 t_3 才能被覆盖, 也就是说只有迁移路径 $t_1t_2t_2t_2t_3$ 才能被触发, 但为了控制候选路径的数量, 本文限定只包含循环路径和自循环迁移一次, 所生成的路径在 t_3 之前只含有 t_2 一次, 导致含有 t_3 的路径都是不可行的. 因此, 在很多情况下, 只有预先知道所需回路的次数才能生成可行的路径. Chanson 等 [22] 在这方面做了一定的尝试, 试图通过回路插入来尝试解决这一问题. 我们的方法也可较为容易的通过设定插入回路的次数来尝试使路径可行, 但这只能用于很简单的情况, 并不能解决根本问题, 关于这方面的研究并不是本文的重点. 其他模型也有类似的情况, SCP 模型含有的状态和迁移的数量相对较少, 但却包含较多不能通过静态分析识别的矛盾关系以及回路、自回路 (通过分析最终发现 SCP 模型的 TPS 中几乎 65.5% 的路径不可行), 因此覆盖率也相对较低. 从上述实验结果可以看出, ATGEM 方法在可行路径上能够有效的生成测试用例. 在有些模型上, 选取的可行路径集还不能达到 100% 的迁移覆盖率, 这与模型本身的性质有一定相关性. 就挑选出的可行路径子集数量和长度而言, 我们的目标是生成一个具有较长路径和较小路径数量的较优可行路径集合. 由于现有的 EFSM 测试生成研究中在这方面还缺乏相关的基准可以参考比较, 因此我们在保证相同覆盖率情况下, 对使用和未使用路径可行性预测方法生成测试用例的结果进行了比较 (详细结果请见表 6, 路径可行性预测方法有效性分析部分). 结果表明. 在使用了路径可行性预测方法后, 挑选出的可行路径子集数量有较大的减少, 而路径的平均长度却有所增加, 这说明路径可行性预测方法对选取具有较长路径和较小路径数量的较优可行路径集合是有效的. 综合以上分析, ATGEM 方法的方法在生成较优可行路径子集、生成测试数据和预言信息和达到指定覆盖方面是行之有效的.

为了回答 (RQ1) 中 ATGEM 方法的效率问题, 在实验中将我们的方法 (ATGEM-SS) 与文献 [40] 的方法 (命名为 Kalaji-GA) 在生成效率上做了比较分析. 主要从 2 个方面进行:

a) 我们的方法 (ATGEM-SS) 与文献 [40] 的方法 (Kalaji-GA) 在总体上对测试用例生成的成功率和时间效率进行比较分析;

b) 相同算法框架下我们提出的 Fitness 函数与文献 [40] 提出的 Fitness 函数在生成测试用例成功率和时间效率上进行比较分析.

实验中我们实现了一个重定义规则的条件表达式解析器来计算 Kalaji-GA 方法的 Fitness 函数. 此外, 实验中 GA 算法在种群大小、最大评估次数、交叉算子和变异算子等参数的设置上和 ATGEM 中使用的分散搜索算法 (ATGEM-SS) 保持一致. 为了在同一算法框架下对 Fitness 函数进行比较, 我们还额外实现了 ATGEM 方法中测试用例生成部分的 GA 版本 (命名为 ATGEM-GA). ATGEM-GA 适应度函数使用了 ATGEM 方法中我们设计的适应度函数, 也就是式 2 在 GA 算法上的实现, 其他配置和 Kalaji-GA 保持一致, 这样能够在同一环境下对 Fitness 函数的效果和效率进行独立比较. 由于谓词和操作间逻辑关系过于简单的可行路径对算法效率的区分度并不大, 为了更客观的验证 ATGEM-SS 方法的效率, 我们在各模型的 TPS 路径候选集中选取了一些谓词和操作之间逻辑关系较为复杂的可行路径, 然后通过在不同算法上为这些可行路径自动生成测试用例来验证算法的效率. 由于 ATM 模型迁移间的相互依赖关系比较简单, 绝大部分可行路径只具有简单的执行逻辑, 测试用例生成过于容

表 5　3 种方法生成测试用例及平均花费时间统计结果

Table 5　The statistical results of three test case generation methods

Method	CLASS2		INRES		SCP	
	SR(%)	AT(s)	SR(%)	AT(s)	SR(%)	AT(s)
ATGEM-SS	100	9.54	100	3.34	100	0.23
Kalaji-GA	100	58.32	100	3.76	100	0.27
ATGEM-GA	100	53.04	100	17.35	100	0.85

易 (所有可行路径只使用了最小迭代次数就可以生成测试用例), 这些路径对于算法效率的区分度并不会很大, 因此在该实验中我们放弃了 ATM 模型. 最终我们在 CLASS2 模型、INRES 模型和 SCP 模型中选取了部分谓词和操作之间逻辑关系较为复杂的可行路径, 具体的选取方法和步骤请参见我们早先的工作 [11].

表 5 给出了 3 种方法在各模型可行路径集上生成测试用例的统计结果. 其中 “成功率” 栏表示在 10 次运行中至少有 1 次成功生成测试用例的路径数量百分比. 从表中我们可以看出, 对于上述 3 个模型中谓词和操作之间逻辑关系较为复杂的可行路径, 3 种方法都能够 100% 为可行路径生成测试用例. “平均时间 (秒)” 是指 3 种方法在各模型可行路径集上生成测试用例的平均花费时间. 从表中可以看出, 在所有 3 个模型的可行路径集上, ATGEM-SS 算法在平均运行时间上都好于文献 [40] 的方法, 在较为复杂、路径数量较多的 CLASS2 模型上更是有巨大的时间效率优势. 在算法框架相同、Fitness 函数不同的情况下, Kalaji-GA 和 ATGEM-GA 方法则在时间效率上互有胜负. 在 INRES 模型和 SCP 模型上 Kalaji-GA 的时间效率要好于 ATGEM-GA; 而在规模较大的 CLASS2 模型上则 ATGEM-GA 的时间效率要好于 Kalaji-GA. 我们认为可能的原因是 Kalaji-GA 的 Fitness 函数的计算比式 2 要复杂, 因此在模型较大时计算所花费的时间也较多, 但总体上 Kalaji-GA 的 Fitness 函数指导性可能更好, 因此在较小的模型上能够更快的找到解. 此外, 我们提出的 Fitness 函数计算时不涉及到数据类型的差异性, 因此可以在多种数据类型的测试用例生成中使用. 通过上述实验结果可以看出, 以上方法都能有效地为指定路径生成测试用例, 与文献 [40] 的方法相比, ATGEM-SS 方法在生成效率上有较大优势, 单纯的 Fitness 函数有效性比较则互有胜负.

为了回答问题 (RQ2), 我们从以下 2 个方面验证路径可行性度量方法的效果:

a) 观察经过可行性度量排序后所有不可行路径在 TPS 中的分布情况, 检验经过可行性排序后不可行路径是否分布在 TPS 的后部;

b) 在保证相同覆盖率的前提下, 对 ATGEM 使用和未使用路径可行性度量方法在生成时间、结果集中路径数量和路径长度等方面进行比较, 检验可行性度量方法能否提高测试用例的生成效率和生成较优可行测试路径子集.

在本实验中, 我们对 TS_Gen 算法和 TPS 的生成算法进行了修改, 对排序后的 TPS 中所有路径生成测试用例 (TPS 中也保留了静态识别出的不可行路径, 根据表 1 将其 f 的值设为一个较大的值参与排序), 并在生成过程中记录所有不可行路径在 TPS 集合中所在的位置. 图 4 列出了 4 个模型中识别出的不可行路径在 TPS 中分布, 其中 X 轴代表不可行路径在 TPS 中的位置编号, Y 轴代表路径的长度. 可以看出, 经过路径可行性排序后, 4 个模型的不可行路径总体都分布在 TPS 的后部. 因此, 通过路径可行性度量方法, 就可以在生成测试用例过程中尽量避开这些不可行路径, 可以减少无谓的工作, 大大提高测试用例生成的效率. 显而易见, 如果没有使用路径可行性度量方法, 最坏的情况是不

表 6 使用和未使用路径可行性度量方法生成测试用例结果比较

Table 6 Comparison results of test case generation with and without feasible metric

Object model	Executed path number			FTP number of RS			Average length of RS			AT(s)		
	Used	Unused	IR(%)	Used	Unused	IR(%)	Used	Unused	IR(%)	Used	Unused	IR(%)
ATM	34	17	**50.00**	21	16	**20.20**	4.6	6.6	**43.40**	42.5	4.3	**89.90**
CLASS2	44	30	**31.90**	16	16	0.00	4	4.6	**15.00**	114.7	56.6	**50.70**
INRES	11	10	**9.10**	10	10	0.00	7.8	9	**15.40**	5.3	1.6	**69.80**
SCP	22	10	**54.50**	3	2	**33.30**	6	9	**50.00**	28.9	12.8	**55.70**

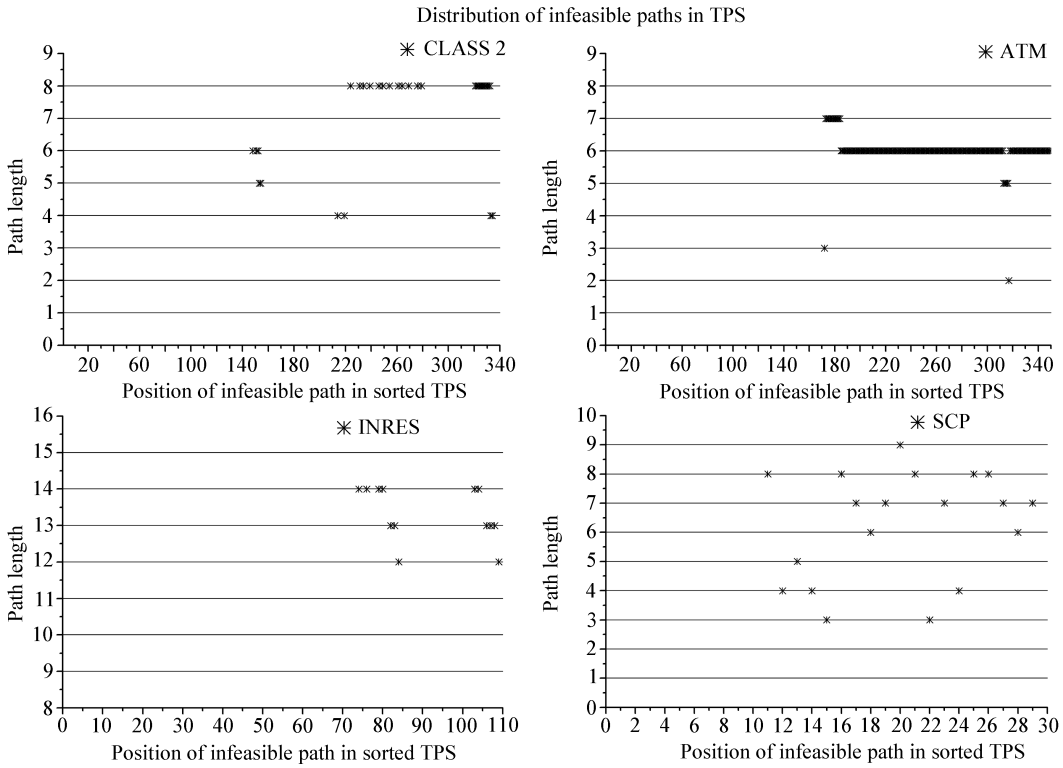


图 4 不可行路径在排序后 TPS 中分布情况

Figure 4 Distribution of infeasible paths in TPS

可行路径都集中在 TPS 的前部. 因为在挑选满足指定覆盖准则的可行路径子集和生成对应的测试用例的过程中是按顺序的, 这样就会首先挑选那些无法生成测试用例的不可行路径来尝试生成, 造成效率低下.

为了进一步验证路径可行性度量方法的效果, 在保证覆盖率相同的前提下, 我们还对 ATGEM 方法使用路径可行性度量和未使用路径可行性度量方法生成测试用例情况进行了比较. 具体实验结果如表 6 所示, 其中 “未使用” 栏是指未使用路径可行性度量方法, 也就是对生成的路径候选集 TPS 未做可行性分析和排序, 直接使用 TS_Gen 方法在候选路径集上生成测试用例. “使用” 栏指使用了我们提出的路径可行性度量方法. “提升率” 栏是指在使用路径可行性度量方法后在特定项目上指标提升的

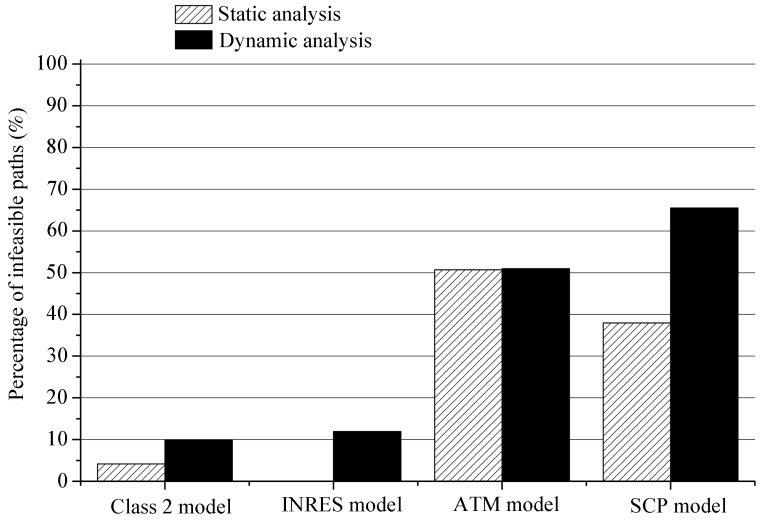


图 5　静态分析和动态分析技术识别的不可行路径的数量比较

Figure 5　Comparison result of infeasible paths that were identified by static analysis and dynamic analysis

百分比, 其中黑体部分说明使用路径可行性度量方法在指标上有所提升, 非黑体部分说明没有提升或有所下降. 例如在 SCP 模型上使用可行性度量方法后, 从 TPS 中挑选用来生成测试用例的路径条数从使用前的 22 条减少到了 10 条; 最终结果集中可行路径的条数从 3 条减少为 2 条, 而路径的平均长度却从 6.0 提高到 9.0; 执行时间从 28.9 秒减少到了 12.8 秒. 也就是说, 被挑选执行的路径数目减少了 54.5%; 结果集中可行路径的数量减少了 33.3%, 路径的平均长度却增加了 50.0%; 测试用例生成的时间减少了 55.7%. 从表中总体可以看出, 在使用了路径可行性度量方法后, 挑选出的可行路径子集数量有所减少, 而路径的平均长度却有较大增加, 这说明路径可行性预测方法对选取具有较长路径和较小路径数量的较优可行路径集合是有效的. 另外, 所有模型在生成测试用例的时间效率上都有较大提升. 综合以上实验分析, 充分说明 ATGEM 中的路径可行性度量方法在预测路径可行性方面具有较好的效果. 利用路径可行性预测方法, 可以减少碰到不可行路径的几率, 较大的提高测试用例生成的效率, 帮助生成一个较优的可行测试路径子集.

为了回答问题 (RQ3), 验证在 ATGEM 方法中动态分析技术及静态分析技术对可行路径的识别效果, 我们在实验中对静态分析技术直接识别出的不可行路径和动态分析技术识别出的不可行路径在数量上进行了对比, 对比的结果如图 5 所示. 从图 5 的实验结果可以看出, 静态分析只能部分识别出不可行路径而动态分析过程则能识别出更多的不可行路径. 通过分析我们发现静态分析技术的效果与 EFSM 模型本身的性质有较大的关系. 由于静态分析方法的局限性, 在实践中, 往往需要动态分析方法进行补充和完善. 例如, 在 INRES 模型中, 静态分析未能识别出任何不可行路径, 而在 ATM 模型中静态分析则预先识别出了较多的不可行路径. 这是因为 ATM 模型中迁移间的关系相对简单, 谓词条件中常量使用较多, 且存在有关键迁移; 而 INRES 模型中迁移间的谓词和操作多为变量, 且存在传递关系, 静态分析方法并不能直接的识别相关路径的可行性, 因此需要利用动态方法进行识别. 但总体来说, 利用静态分析技术预先识别部分不可行路径, 仍然可以提高动态过程中测试用例生成的效率. 总的来说, 静态方法只能直接识别部分不可行路径, 对于很多复杂情况, 仍然需要使用动态技术来进行补充. 也就说, 在测试用例生成过程中仍然可能会碰到很多静态方法无法识别的不可行路径, 而通过路径可行性度量方法可以尽量避开这些不可行路径.

通过实验我们发现有几个方面的因素影响路径可行性: 首先是模型中回路和自回路的影响. 某些迁移的谓词条件的成立依赖于路径中其他迁移被触发的次数, 只有当特定迁移被触发特定次数后, 该迁移才能被触发, 这就导致所有包含这 2 个迁移且不符合以上情况的路径都是不可行的. 而在复杂条件下, 特定迁移需要触发的次数是很难事先确定的. 其次通过观察发现, 包含在一条路径中的迁移上的谓词条件越多、迁移间关联度越高则该路径生成测试用例难度相对也就越大, 路径不可行的概率也就越高. 另外, 输入参数的值域范围也会对路径可行性和测试用例的生成有影响. 例如, 在 CLASS2 模型中, 当输入参数的值域从 [−10000,10000] 缩减至 [−1000,1000], 测试用例生成所需的迭代次数则相应减少, 生成变的更容易; 然而当输入参数的值域缩减至 [0,1000], 生成难度却反而加大. 因此, 一个适当的参数值域, 也会提高测试用例的生成效率.

此外, 实验中还存在影响实验结果的一些客观因素. 首先是所使用的静态分析技术的轻重. 若使用轻量级的技术, 所花费的代价较小, 但预先识别出的不可行路径数目相对较少; 而使用重量级的分析技术, 所花费的代价较大, 但预先识别出的不可行路径数目相对较多. 在本文中, 我们使用了轻量级的静态分析技术. 其次是最大迭代次数的确定. 若最大迭代次数设置的过小, 有些难以生成测试用例的路径就会被认为是不可行的; 相反, 若当迭代次数设置的过大, 在不可行路径上的花费代价则会大大增加. 在实验中, 我们通过逐渐调整最大迭代次数来使生成过程达到一个稳定的状态. 此外, 目前实际可用于实验的 EFSM 模型数量还相对较少, 为了尽量减小该因素带来的影响, 我们在现有的研究中收集了 6 个 EFSM 模型, 其中 2 个由于谓词及操作间关系过于简单而放弃, 选择了 4 个相对复杂模型用于实验分析, 今后我们将通过建立更多的模型来增强实验的效果.

6 总结与展望

本文提出了一种针对 EFSM 模型的测试用例自动化生成方法 ATGEM, 利用提出的路径可行性度量方法来评估路径的可行性, 尽可能的避开不可行路径, 并找到一个具有较长路径和较少路径数量的可行路径集合达到指定的路径覆盖准则. 为了在特定的可行路径上生成测试数据和预言信息, 提出了一种新的基于动态反馈信息的适应度函数计算方法和 EFSM 静态模型可执行化方法, 利用开发的可执行模型来获取运行时反馈信息结合分散搜索技术自动化的生成测试数据和创建预言信息. 该方法中的适应度函数计算代价较小, 利用运行时反馈信息计算适应度函数, 就可以忽略不同数据类型带来的差异性问题, 使得该方法能够使用在多种数据类型的测试用例生成中, 适用范围较广. 通过实验识别出多个 EFSM 模型的大量不可行路径, 实验结果表明该 ATGEM 方法能有效的生成测试用例, 与现有方法相比, ATGEM 的测试用例生成方法效率更高. 实验还表明 ATGEM 中的路径可行性度量方法对路径的可行性预测总体上较为准确, 可以大幅度的提高测试用例生成的效率. 该度量思想也可以使用在其他路径评估领域.

在已有的研究中, 可执行路径的获得和测试用例的生成研究还相对独立, 而这两方面的实际中是紧密集合的, 若一条测试路径中的谓词和对应操作之间的关系较为复杂, 则对于 EFSM 模型中不可执行路径的检测也会变得更加困难, 对应测试用例生成的难度和测试代价也相对较高. 因此, 在未来研究中, 基于 EFSM 模型的测试需要更多的挖掘测试序列、测试数据和测试预言之间的关系, 并结合多种分析技术和优化技术来提高测试的覆盖率和效率, 获得更好的测试效果. 此外, 在保证效率的前提下, 如何利用更多高效的静态分析技术预先识别不可行路径, 减少在动态生成测试用例过程中遇到不可行路径的情况, 也是我们下一步的研究目标之一. 在测试数据生成方面, 目前的方法对简单数据类

型处理比较有效, 对复杂数据类型来说, 处理起来仍然相对困难, 利用元搜索技术、动态符号执行等方法来生成复杂测试数据也将是测试数据生成技术领域未来研究的重要方向之一. 此外, 针对 EFSM 模型的测试预言自动化生成研究还相对较少, 抽象模型通常只能描述被测系统某些方面的约束和行为特性, 只能检验系统的部分行为, 而预言信息最终要在实际运行的系统上进行检验. 因此, 如何生成有效的测试预言, 在测试预言与测试代价之间取得平衡, 更多的挖掘被测系统与抽象模型之间的差异, 将是测试预言生成技术的面临挑战之一.

参考文献

1 Beizer B. Software Testing Techniques. 2nd ed. Van Nostrand Reinhold Company Limited, 1990

2 Petrenko A, Boroday S, Groz R. Confirming configurations in EFSM. In: Proceedings of the International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification, Beijing, 1999. 5–24

3 NAITO S, Tsunoyama M. Fault detection for sequential machines by transition tours. Fault Tolerant Comput Syst, 1981, 1: 238–243

4 Gonenc G. A method for the design of fault detection experiments. IEEE Trans Comput, 1970, 100: 551–558

5 Sabnani K K, Dahbura A T. A protocol testing procedure. Comput Netw ISDN Syst, 1988, 15: 285–297

6 Chow T S. Testing software design modeled by finite-state machines. IEEE Trans Softw Eng, 1978, 4: 178–187

7 Fujiwara S, Khendek F, Amalou M, et al. Testing selection based on finite state models. IEEE Trans Softw Eng, 1991, 17: 591–603

8 Hedley D, Hennell M A. The causes and effects of infeasible paths in computer programs. In: Proceedings of the 8th International Conference on Software Engineering, London, 1985. 259–266

9 Fraser G, Wotawa F. Redundancy based test-suite reduction. Lect Notes Comput Sc, 2007, 4422: 291–305

10 Yang R, Chen Z Y, Xu B W, et al. Improve the effectiveness of test case generation on EFSM via automatic path feasibility analysis. In: Proceedings of the 13th IEEE International High Assur–ance Systems Engineering Symposium, Boca Raton, 2011. 17–24

11 Zhang J, Yang R, Chen Z Y, et al. Automated EFSM–based test case generation with scatter search. In: Proceedings of 7th IEEE/ACM International Conference on Software Engineering Workshop on Automated Software Test, Zurich, 2012. 76–82

12 Yang R, Chen Z Y, Xu B W, et al. A new approach to evaluate path feasibility and coverage ratio of EFSM based on multi-objective optimization. In: Proceedings of the 24th International Conference on Software Engineering and Knowledge Engineering(SEKE'12), San Francisco, 2012. 470–475

13 Huang C, Jang M, Lin Y. Executable EFSM-based data flow and control flow protocol test sequence generation using reachability analysis. J Chin Inst Eng, 1999, 22: 593–615

14 Shih C, Huang J, Jou J. Stimulus generation for interface protocol verification using the non-deterministic extended finite state machine model. In: Proceedings of the 10th International High-Level Design Validation and Test Workshop, Napa, 2005. 87–93

15 Ramalingom T, Das A, Thulasiraman K. Context independent unique sequences generation for protocol testing. In: Proceedings of the 15th Annual Joint Conference of the IEEE Computer and Communications Societies, San Francisco, 1996. 1141–1148

16 Ramalingom T, Thulasiraman K, Das A. Context independent unique state identification sequences for testing communication protocols modelled as extended finite state machines. Comput Commun, 2003, 26: 1622–1633

17 Huang C, Chiang M, Jang M. UIOE: a protocol test sequence generation method using the transition executability analysis (TEA). Comput Commun, 1998, 21: 1462–1475

18 Zhou X Y, Qu Y G, Zhao B H. Using invertibility to reduce the length of test sequences in EFSM model. J China Inst Commun, 2000, 21: 48–55 [周晓煜, 屈玉贵, 赵保华. 利用逆向判定性缩短 EFSM 的测试序列的长度. 通信学报, 2000, 21: 48–55]

19 Petrenko A, Boroday S, Groz R. Confirming configurations in EFSM testing. IEEE Trans Softw Eng, 2004, 30: 29–42
20 Ural H, Yang B. A test sequence selection method for protocol testing. IEEE Trans Commun, 1991, 39: 514–523
21 Miller R E, Paul S. Generating conformance test sequences for combined control and data flow of communication protocols. In: Proceedings of 12th International Symposium on Protocol Specifications, Testing and Verification, North-Holland, 1992. 13–27
22 Chanson S T, Zhu J. A unified approach to protocol test sequence generation. In: Proceedings of the 12th Annual Joint Conference of the IEEE Computer and Communications Societies, San Francisco, 1993. 106–114
23 Ramalingom T, Das A, Thulasiraman K. A unified test case generation method for the EFSM model using context independent unique sequences. In: Proceedings of International Workshop on Protocol Test Systems, Evry, 1995. 289–305
24 Huang C M, Lin Y C, Jang M Y. An Executable Protocol Test Sequence Generation Method for EFSM-Specified Protocols. US: Springer, 1996. 20–35
25 Hierons R M, Kim T H, Ural H. Expanding an extended finite state machine to aid testability. In: Proceedings of the 26th Annual International Computer Software and Applications, 2002. 334–339
26 Hierons R M, Kim T H, Ural H. On the testability of SDL specifications. Comput Netw, 2004, 44: 681–700
27 Pang Q X, Cheng S D, Jin Y H. Equivalent transformation of EFSM and protocol conformance testing. J China Inst Commun, 1997, 18: 37–42 [庞其祥, 程时端, 金跃辉. EFSM 的等价转换和通信协议一致性测试. 通信学报, 1997, 18: 37–42]
28 Zhao B H, Chen B, Qu Y G. A test sequence generation algorithm based on improved transition executability analysis. J Univ Sci Technol, 2007, 37: 1096–1100 [赵保华, 陈波, 屈玉贵. 一种改进的转换可执行分析测试序列生成算法. 中国科学技术大学学报, 2007, 37: 1096–1100]
29 Uyar M U, Duale A Y. Test generation for EFSM models of complex army protocols with inconsistencies. In: Proceedings of 21st Century Military Communications Architectures and Technologies for Information Superiority, Los Angeles, 2000. 340–346
30 Duale A Y, Uyar M U. A method enabling feasible conformance test sequence generation for EFSM models. IEEE Trans Comput, 2004, 53: 614–627
31 Bourhfir C, Dssouli R, Aboulhamid E, et al. Automatic executable test case generation for extended finite state machine protocols. In: Proceedings of the International Workshop for Protocol Test Systems, Cheju Island, 1997. 75–90
32 Derderian K, Hierons R M, Harman M, et al. Estimating the feasibility of transition paths in extended finite state machines. Automat Softw Eng, 2010, 17: 33–56
33 Kalaji A S, Hierons R M, Swift S. Generating feasible transition paths for testing from an extended finite state machine (EFSM). In: Proceedings of the International Conference on Software Testing, Verification and Validation, Denver, 2009. 230–239
34 Kalaji A S, Hierons R, Swift S. Generating feasible transition paths for testing from an extended finite state machine (EFSM) with the counter problem. In: Proceedings of the 3th International Conference on Software Testing, Verification, and Validation Workshops, Paris, 2010. 232–235
35 Yano T, Martins E, Sousa F L. Generating feasible test paths from an executable model using a multi-objective approach. In: Proceedings of the 3th International Conference on Software Testing, Verification, and Validation Workshops, Paris, 2010. 236–239
36 Yano T, Martins E, Sousa F L. MOST: a multi-objective search-based testing from EFSM. In: Proceedings of the 4th International Conference on Software Testing, Verification, and Validation Workshops, Berlin, 2011. 164–173
37 Chanson S T, Zhu J. Automatic protocol test suite derivation. In: Proceedings of the 13th IEEE Networking for Global Communications, Toronto, 1994. 792–799
38 Zhang Y, Qian L Q, Wang Y F. Automatic testing data generation in the testing based on EFSM. Chin J Comput, 2003, 26: 1295–1303 [张涌, 钱乐秋, 王渊峰. 基于扩展有限状态机测试中测试输入数据自动选取的研究. 计算机学报, 2003, 26: 1295–1303]
39 Lefticaru R, Ipate F. Automatic state-based test generation using genetic algorithms. In: Proceedings of the 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, 2007. 188–195
40 Kalaji A S, Hierons R M, Swift S. An integrated search-based approach for automatic testing from extended finite state machine (EFSM) models. Inform Software Tech, 2011, 53: 1297–1318

41 McMinn P, Holcombe M. Evolutionary testing of state-based programs. In: Proceedings of the Conference on Genetic and Evolutionary Computation, Washington, 2005. 1013–1020
42 Tracey N, Clark J, Mander K, et al. An automated framework for structural test-data generation. In: Proceedings of the 13th IEEE International Conference on Automated Software Engineering, Hawaii, 1998. 285–288
43 Zhang J, Xu C, Wang X. Path-oriented test data generation using symbolic execution and constraint solving techniques. In: Proceedings of 2nd International Conference on Software Engineering and Formal Methods, Beijing, 2004. 242–250
44 Wu T, Yan J, Zhang J. A path-oriented approach to generating executable test sequences for extended finite state machines. In: Proceedings of the 6th International Symposium on Theoretical Aspects of Software Engineering, Beijing, 2012. 267–270
45 Zhao R L, Harman M, Li Z. Empirical study on the effciency of search based test generation for EFSM models. In: Proceedings of 3rd International Conference on Software Testing, Verification, and Validation Workshops, Paris, 2010. 222–231
46 Tahat L H, Vaysburg B, Korel B, et al. Requirement-based automated black-box test generation. In: Proceedings of the 25th Annual International Computer Software and Applications Conference, Chicago, 2001. 489–495
47 Saingern S, Lursinsap C, Sophatsathit P. An address mapping approach for test data generation of dynamic linked structures. Inform Software Tech, 2005, 47: 199–214
48 Miller W, Spooner D L. Automatic generation of floating-point test data. IEEE Trans Softw Eng, 1976, 2: 223–226
49 Memon A, Banerjee I, Nagarajan A. What test oracle should I use for effective GUI testing? In: Proceedings of the 18th IEEE International Conference on Automated Software Engineering, Montreal, 2003. 164–173
50 Glover F. A template for scatter search and path relinking. In: Proceedings of the 3rd European Conference on Artificial Evolution, London, 1998. 13–54
51 Marti R, Laguna M, Campos V. Scatter search vs. Genetic algorithms: an experimental evaluation with permutation problems. Metaheuristic Optimization via Memory and Evolution. Berlin: Springer, 2005. 263–282
52 Nebro A J, Luna F, Alba E. AbYSS: adapting scatter search to multiobjective optimization. IEEE Trans Evolut Comput, 2008, 12: 439–457
53 Korel B, Singh I, Tahat L, et al. Slicing of state-based models. In: Proceedings of the International Conference on Software Maintenance, Amsterdam, 2003. 34–43
54 Cavalli A, Gervy C, Prokopenko S. New approaches for passive testing using an extended finite state machine specification. Inform Software Tech, 2003, 45: 837–852

A new approach of automated test case generation on extended finite state machine

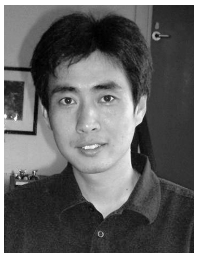
YANG Rui[1,2], CHEN ZhenYu[1], ZHANG ZhiYi[1], LIU ZiCong[1] & XU BaoWen[1,2*]

1 *State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210046*, China;*
2 *Department Computer Science and Technology, Nanjing University, Nanjing* 210046*, China*
*E-mail: bwxu@nju.edu.cn

Abstract Extended finite state machine (EFSM) is among the most popular models for model-based testing. However, automated test case generation on EFSM models is still a challenging task since an EFSM model may contains infeasible paths. This paper proposed a novel approach (ATGEM) to generate test case and construct oracle information from EFSM automatically. To address the infeasible problem, a metric based on data flow analysis is presented to predict the infeasible probability so as to bypass the infeasible paths as far as possible and improve the test case generation efficiency. Afterwards, an executable EFSM model is developed to obtain runtime feedback information as a fitness function in order to generate test data and construct oracle information automatically. This approach, which can generate various types data and has a wide range of applications,

combines static analysis and dynamic analysis aims to find a preferable feasible path subset to generate test cases and meet adequacy coverage criteria. The experimental results on several EFSM models show that test case generation method and path feasibility metric have good effectiveness. Utilizing path feasibility metric can speed up the process of test case generation greatly, and ATGEM is more efficient than existing method.

Keywords test case generation, EFSM model-based testing, executable model, path feasibility analysis, test oracle



YANG Rui received his B.S. and M.S. degrees from Lanzhou University and Soochow University, respectively. He is currently a PhD student in the department of computer science and technology at Nanjing University. His current research interests include software testing and program analysis.

CHEN ZhenYu is currently an Associate Professor at Software Institute, Nanjing University. He received his B.S. and Ph.D. in Mathematics from Nanjing University. He worked as a Postdoctoral Researcher at the School of Computer Science and Engineering, Southeast University, China. His research interests focus on software analysis and testing.

ZHANG ZhiYi received his MS degree from Nanjing University. He is currently a Ph.D. candidate with the Software Institute at Nanjing University. His current research interests include software testing and program analysis.

LIU ZiCong is currently an undergraduate student in Software Engineering at Nanjing University. He will receive his Bachelor Degree from Nanjing University in 2014. His research focuses on software testing and program analysis.