抽象精化和可满足性结合的 EFSM 模型测试用例优化生成

陆公正[1),2)] 缪淮扣[2),3)]

[1)](苏州市职业大学计算机工程学院 江苏 苏州 215104)

[2)](上海大学计算机工程与科学学院 上海 200072)

[3)](上海市计算机软件测评重点实验室 上海 201112)

摘 要 基于模型的测试是测试自动化的重要手段,通常采用模型检验技术从系统模型自动生成测试用例集,但生成的测试用例集往往存在冗余,这将影响测试用例执行的性能和成本.该文以扩展有限状态机(Extended Finite Machine,EFSM)为建模工具,根据公式簇建立状态等价关系,构建抽象模型,采用模型检验技术生成抽象反例(测试用例);给出了判定生成的抽象反例是否为伪反例的方法;采用反例引导的方法精化抽象模型,删除伪反例;最后,使用我们之前提出的基于可满足性的测试用例生成方法在抽象模型上生成约简的测试用例集.实验表明:该方法的测试用例数目约简比例最高达 76%(警报侦测组件 EFSM),总长度约简比例最高达 68%(ATM EFSM),同时不会影响测试用例集的迁移覆盖率和查错能力.

关键词 测试用例约简;扩展有穷状态机;公式簇;抽象;反例引导的精化;可满足性

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2016.02236

Optimized Test Cases Generation for EFSM Model Combining Abstraction Refinement and Satisfiability

LU Gong-Zheng[1),2)] MIAO Huai-Kou[2),3)]

[1)](*School of Computer Engineering, Suzhou Vocational University, Suzhou, Jiangsu* 215104)

[2)](*School of Computer Engineering and Science, Shanghai University, Shanghai* 200072)

[3)](*Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai* 201112)

Abstract Model based testing is an important means of test automation. Usually, Test Suite can be generated automatically from system model using model checking technique. It generates one test case for each test goal, and the final test suite usually has redundancy which will affect the performance and cost of the test execution. Moreover, the cost of calling model checker is expensive. In this paper, Extended Finite State Machine (EFSM) was used to model the system, then equivalence relation was defined according to formula clusters consist of the variables occurring in the predicates, then an abstraction model can be built, and the counterexamples (test cases) were generated from the abstraction model using model checking. The method deciding whether a counterexample is a spurious counterexample was proposed. If it is a spurious counterexample, a counterexample-guided abstraction refinement framework was used to refine the abstraction model. Finally, we got the optimized test suite using the test case generation method based on satisfiability we proposed before, which reduced the redundant test cases during test case generation.

收稿日期:2015-06-25;在线出版日期:2016-02-26.本课题得到国家自然科学基金(61170044,61572306)、上海市重点学科建设项目基金(J50153)和苏州市职业大学预研基金(SVU2015YY01)资助.**陆公正**,男,1981 年生,博士,讲师,中国计算机学会(CCF)会员,主要研究方向为形式化方法、软件测试.E-mail:tmks0863@sina.com.cn.**缪淮扣**,男,1953 年生,教授,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为形式化方法、软件工程.

This method can not only optimize the test case generation, it also decreases the number of calling model checker. The experimental results show that the method proposed in this paper can reduce the number of test suite up to 76% (Alarm Detection EFSM) and its total length up to 68% (ATM EFSM). Meanwhile, it will not affect the transition coverage rate and error detection ability.

Keywords　test case reduction; extended finite state machine; formula cluster; abstraction; counterexample-guided refinement; satisfiability

1 引　言

软件测试是发现软件错误，保证软件质量的一种重要手段. 传统软件测试方法所需成本占软件开发总成本的 50%以上[1]. 测试自动化是一种能够降低测试成本的有效方法. 测试用例的自动生成是测试自动化的一个重要环节和体现. 基于模型的测试使测试自动化成为可能. 它根据指定的测试覆盖准则从软件的行为模型自动生成测试用例(抽象测试用例)，然后对测试用例进行实例化，最后在实际系统上执行测试用例，观察系统运行的结果是否与测试用例的预期行为一致. 有时根据指定的测试覆盖准则，从软件行为模型生成的测试用例集规模庞大，导致测试成本高、效率低. 因此，测试用例集约简是软件测试的一项重要工作.

模型检验[2]是一种自动验证有穷状态系统的方法，在很多领域有着广泛的应用. 近年来，有人把它应用到软件测试中. 在模型检验器验证可达性性质的过程中，如果该性质不成立，会生成一条反例作为证据序列，它与覆盖测试目标的测试用例极其相似，所以可以应用模型检验技术自动生成测试用例. 但使用模型检验技术生成测试用例的一个最主要缺点是生成的测试用例集会存在大量冗余，这增加了测试生成和执行成本. 抽象是能够约简测试用例的主要技术之一，其主要思想是构造软件行为模型的抽象模型，使状态数和迁移数大大约简. 抽象模型中的状态称为抽象状态，通过抽象得到抽象模型的软件行为模型称为具体模型，具体模型中的状态称为具体状态.

抽象移除或简化那些与验证性质无关的细节. 验证抽象模型比验证具体模型有效，但是在抽象过程中会丢失一些信息，在验证抽象模型的过程中可能会导致错误的结果[3]. 抽象分为欠近似和过近似两种. 欠近似抽象减少了具体模型中的一些行为，在抽象模型中不成立的性质在具体模型中也不成立；过近似抽象具有性质保留性，即在抽象模型满足的性质在具体模型也满足，它在抽象模型中引入一些额外的行为，因此，当抽象模型违反某一性质时，模型检验器生成的反例可能是抽象模型引入的额外行为，不能说明具体模型也违反该性质，这样的反例我们称为伪反例(spurious counterexample). 为了能够更准确地验证性质，专家学者提出了反例引导的抽象精化(CounterExample-Guided Abstraction Refinement，CEGAR)框架[4]. 模型检验器在抽象模型生成反例时，需要判定在具体模型中是否存在对应的反例，一旦为伪反例，则对抽象模型进行精化以删除伪反例.

本文采用反例引导的抽象精化框架抽象具体模型，在抽象模型上生成优化的测试用例集. 本文主要完成的是：

(1) 构造待测软件的抽象模型. 将 EFSM 模型中迁移上的谓词(监护条件)划分成公式簇(Formula Cluster). 根据公式簇建立状态的等价关系，获得抽象状态，然后分几种情况讨论如何合并相关迁移，最后构造初始抽象模型.

(2) 反例判定和模型精化. 对每个测试目标，用 LTL 表示成一个陷阱性质(Trap Property)，使用模型检验技术在抽象模型上为测试目标生成反例，判定具体模型中是否存在对应的具体反例. 如存在，则说明抽象反例可以作为测试用例；否则，采用反例引导的精化框架精化抽象模型，删除伪反例，继续为该测试目标反例，直到抽象模型不存在伪反例.

(3) 约简的测试用例集生成. 使用我们先前提出的基于可满足性的测试用例优化方法[5]在最终的抽象模型上生成约简的测试用例集.

本文第 2 节介绍文中用到的一些基本概念；第 3 节介绍我们之前提出的基于可满足性的测试用例优化方法；第 4 节给出公式簇的概念和抽象的形式

定义;第 5 节为测试用例方法;第 6 节是判定伪反例和精化抽象模型的方法;第 7 节是实验及结果;第 8 节是相关工作;最后是总结和进一步工作.

2 基本概念

模型检验需要构造系统模型和描述系统性质.我们用 EFSM[6] 作为建模工具,用线性时序逻辑(Linear Temporal Logic,LTL)[7] 描述系统性质.

定义 1[6]. EFSM M 是一个六元组$(S,init,V,I,O,T)$,其中 S 是一个有穷状态集;$init$ 是初始状态集;V 是有穷上下文变量集;I 是有穷输入集;O 是有穷输出集;T 是有穷迁移集.迁移 $t\in T$ 是一个五元组(s_s,i,g,op,s_t),其中 s_s是迁移 t 的起始状态;$i\in I$ 是输入消息,它可能包含输入参数;g 是谓词;op 是顺序操作,它包括赋值语句和输出消息;s_t是迁移 t 的结束状态.在 g 和 op 中可以同时包含输入参数和上下文变量.

本文只考虑确定的 EFSM.图 1 是一个 EFSM 模型的例子,将用它解释本文的方法.

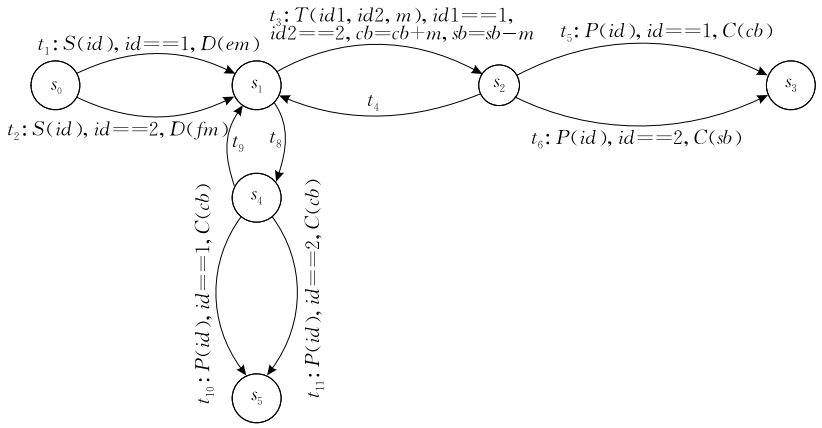

图 1 EFSM 模型示例

定义 2. EFSM 中长度为 n 的路径是一个状态序列 $\pi=\langle s_0,\cdots,s_n\rangle$,其中 $\forall 0\leq i\leq n-1:(s_i,s_{i+1})\in T,s_0\in init$,$T$ 是 EFSM 中的迁移集,s_i 和 s_{i+1} 分别是 $t_i\in T$ 的开始状态和结束状态.

定义 3[7]. LTL 的语法可表示为

$$\phi ::= \text{true}\mid\text{false}\mid p\mid\neg\phi\mid\phi_1\wedge\phi_2\mid\phi_1\vee\phi_2\mid\phi_1\Rightarrow\phi_2\mid X\phi\mid F\phi\mid G\phi\mid\phi_1 U\phi_2,$$

其中,p 是原子命题,ϕ_1 和 ϕ_2 是 LTL 公式,时序算子 X,F,G,U 分别表示下一状态、将来的某个状态、将来的所有状态和直到某个状态.

测试覆盖准则是设计测试用例的依据,它表示测试用例执行的基本单元.在基于模型的测试中主要有状态覆盖准则、迁移覆盖准则和迁移对覆盖准则等[8].这里我们只考虑迁移覆盖准则.

在基于模型的测试中,测试用例是一条可以覆盖某个测试目标的有穷状态(迁移)序列.模型检验中的可达性分析用来验证可达性性质在模型中是否成立,即是否存在一条有穷状态序列能够到达某个模型元素,这样的状态序列和测试用例很相似,所以可以把模型检验技术用于测试用例的自动生成.在用模型检验技术生成测试用例[9] 的过程中,根据测试覆盖准则,列出测试目标,把测试目标表示成陷阱性质(可达性性质的否定).如迁移覆盖准则中的一个测试目标是能够到达迁移 t_i,用陷阱性质表示"在模型中不存在到达迁移 t_i 的路径",对应的 LTL 公式为 $G\neg(s_i\Rightarrow Xs_{i+1})$.状态 s_i 和 s_{i+1} 分别是迁移 t_i 的开始状态和结束状态.模型检验器验证陷阱性质,事实上,模型中存在到达迁移 t_i 的路径,所以,模型检验器生成一条到达 t_i 的反例来证明这种情况,从而构成了能够覆盖迁移 t_i 的测试用例.

3 基于可满足性的测试用例优化方法

用模型检验生成测试用例时,要为每个测试目标的陷阱性质生成一条测试用例.这样做可能会导致出现重复的测试用例或一条长的测试用例包含一条短的测试用例的情况,这些重复的或被包含的测试用例是冗余的.比如,为迁移 t_5 生成的一条测试用例是 $tc_1=\langle s_0,t_1,s_1,t_3,s_2,t_5,s_3\rangle$(这里给出迁移是为了区分不同的路径),为迁移 t_3 生成的一条测试用例是 $tc_1=\langle s_0,t_1,s_1,t_3,s_2\rangle$,测试用例 tc_1 包含 tc_2,tc_2 是冗余的.通过约简冗余的测试用例可以降低测试生成和执行的代价.为了不再为已被已有测试用例覆盖的陷阱性质生成测试用例,通过可满足性理论

约简冗余的测试用例.

限界模型检验[10]主要是基于可满足性(SAT)理论[11]判定模型是否满足时序逻辑性质,与传统的模型检验不同,它不用搜索状态空间.它把系统行为表示成形式模型,要验证的性质表示成 LTL 公式,设定边界上界 k,状态间的迁移关系和 LTL 公式的否定通过合取构成 BMC 公式,把 BMC 公式编码成 SAT 实例(CNF 公式),通过 SAT 工具进行求解.如果该实例有解,则找到反例;反之,如果不可满足,则说明模型在运行 k 步后满足性质.这里,我们把 SAT 理论应用于约简模型检验器生成的测试用例.首先,根据测试覆盖准则(如迁移覆盖),列出所有测试目标,从中选择一个测试目标,把它表示成陷阱性质,通过模型检验器验证该陷阱性质是否成立.如果不成立,则生成一条测试用例,表示存在一条状态序列可以到达该测试目标(迁移),即这条测试用例可以覆盖该测试目标(迁移),把那些还未被覆盖的测试目标表示成陷阱性质,对于每个陷阱性质,把它和测试用例的合取转换成 CNF 公式,通过 SAT 工具求解该公式的可满足性.如果该公式不可满足,根据定理 1 可知,这条测试用例能够覆盖这个陷阱性质所对应的测试目标,不需再调用模型检验器为该测试目标生成测试用例,从而可以约简测试用例的数目和测试用例集的总长度.

根据什么样的次序选择测试目标来生成测试用例会很大程度的影响测试用例约简的效果,在以往的工作中,都是随机选择测试目标.在 SAT 理论中,子句长度越短满足子句的难度就越大[11],基于此,我们根据系统模型 M 与陷阱性质 $\neg\phi$ 的合取式转换得到的 CNF 公式的难度来选择测试目标.

定义 4.　设 CNF 公式中子句数为 n,第 i 个子句中的文字数为 l_i,则 CNF 公式的难度 h:

$$h=\frac{\sum_{i=1}^{n} l_i}{n}.$$

对于每个陷阱性质,把它与模型进行合取,转换得到相应的 CNF 公式,然后计算每个 CNF 公式的难度,得到的难度集合为 H.测试目标根据他们对应的 CNF 的难度按升序排列,得到测试目标集 RTG.从 RTG 中顺序地选择测试目标生成测试用例.对于具有相同难度的测试目标,我们随机地进行选择.h 的值越小,生成的测试用例长度越长,这样,避免生成与已有测试用例相同的或被它包含的测试用例.

使用 SAT 理论检查已生成的测试用例是否覆盖测试目标集中其余目标的方法主要基于以下定理.

定理 1.　设对于测试目标 tg_1,生成测试用例 $\pi:=\langle s_0,\cdots,s_t\rangle$,在测试目标集 TG 中选择未被覆盖的测试目标 tg_2,如果 $\pi\wedge G\neg tg2$ 转换得到的 CNF 公式是不可满足的,则测试用例 π 覆盖测试目标 tg_2.

证明.　因为 $\pi\wedge G\neg tg2$ 转换得到的 CNF 公式是不可满足的,那么也就是 $\pi\not\models G\neg tg2$,从此可以看出 π 是 $G\neg tg2$ 的反例,因而 π 可以作为覆盖测试目标 tg_2 的测试用例.　　证毕.

使用 SAT 理论约简测试用例集的具体过程见算法 1 和算法 2.

算法 1.　TestSuiteReduction(M,TG,TS).

```
输入:模型 M,测试目标集 TG
输出:约简的测试用例集 TS
BEGIN
  H={};
  FOR each test goal φ in TG DO
  BEGIN
    i=1;
    //生成 M∧¬φ 的 CNF 公式
    f=GenerateCNF (M∧¬φ);
    //计算每个 CNF 公式 f 的难度
    h(i)=ComputeHardness(f);
    //得到难度集合 H
    H=H∪h(i)
    i=i+1;
  END
    //按 CNF 的难度排列测试目标
    RTG=RankAscending(TG,H);
    TS={};
    WHILE RTG !=empty DO
    BEGIN
      Select a test goal φ from RTG;
      RTG=RTG-{φ};
          //在模型 M 检测测试目标的陷阱性质
          //¬φ,生成反例 π
      π=ModelChecking(M,¬φ);
      IF TS={}
      THEN TS=TS∪{π};
          //使用测试用例集 TS 检查 π,检查它是否
          //是冗余的测试用例
      ELSE TS=Winnow(TS,π)
    FOR each remain test goal φ in RTG DO
        BEGIN
```

```
            f=GenerateCNF(π,φ);
            //使用 SAT 求解 CNF 公式 f
            result=SAT(f);
            IF result=unsatisfiable
            THEN RTG=RTG−{φ};
          END
        END
END
```

算法 2. Winnow(TS,π).

```
输入:已有的测试用例集 TS 和新生成的测试用例 π
输出:新的测试用例集
BEGIN
  FOR each test case ζ in TS DO
  BEGIN
  //对于新生成的测试用例 π 和测试用例集的测试用
  //例 ζ,求两个测试用例的长度的较小值
    j=MinLength(π,ζ);
    FOR i=1 to j DO
    BEGIN
    //检查两个测试用例中的第 i 个状态是否相同
      IF !Match(π(i),ζ(i))
    //如果不同则说明新的测试用例不是冗余的
      THEN TS=TS∪{π};
        EXIT;
    //如果 ζ 被 π 包含
      ELSE IF (i==j) and (Length(π)>Length(ζ))
    //在测试用例集中删除 ζ 并把 π 添加进去
      THEN TS=TS−{ζ}∪{π};
    END
  END
END
```

下面采用归纳法分析算法 1 的正确性和终止性,假设测试目标的数目为 n.

(1) 当 $n=1$ 时,只有一个测试目标,通过模型检验生成一条测试用例;

(2) 假设当 $n=p$ 时,算法能够正确的约简测试用例;

(3) 那么当 $n=p+1$ 时,算法先为每个测试目标生成相应的 CNF 公式,然后计算每个 CNF 公式的难度,假设针对前 p 个测试目标,约简后得到的测试用例数为 m(设该用例集为 TS),从这 m 条测试用例中选择一条,生成该测试用例与第 $p+1$ 个测试目标的 CNF 公式,通过 SAT 求解器检查这条测试用例是否覆盖这个测试目标,如果覆盖,算法终止,最终测试集中有 m 条测试用例,如果不覆盖,取 TS 中的下一条测试用例,直到取完 TS 中的所有测试用例,最终的测试集中有 $m+1$ 条测试用例,所以算法 1 能够正确的约简测试用例集并终止.

算法 1 的主要操作有模型检验、生成 CNF 公式、SAT 求解 CNF 公式可满足性、计算 CNF 公式复杂度、排序测试目标以及 Winnow 过程,其中计算 CNF 公式复杂度、排序测试目标和 Winnow 过程的时间复杂度都是线性的,模型检验的时间复杂 $O(|M|\times 2^{|\phi|})$,生成CNF公式和SAT求解的时间复杂度也是 $O(|M|\times 2^{|\phi|})$,设测试目标数为 n,算法 1 的时间复杂度为 $O(n^2\times|M|\times 2^{|\phi|})$.

在使用 SAT 求解器求解 CNF 公式的可满足性时,由于设定了边界上界为 k,所以在公式满足或者不满足时,SAT 求解过程都能在 k 步内终止.

4 抽象模型

抽象的目的是使模型的状态数和迁移数减少,但同时保留要验证的性质. 本文采用 EFSM 模型中迁移上的谓词来抽象原始模型. 在给出抽象过程之前,下面列出一些需要用到的概念.

抽象函数 α 是一个满射 $\alpha: S\rightarrow S^{\alpha}$,$S^{\alpha}$ 是抽象状态集. 抽象函数 α 定义了状态集 S 的一个等价关系 $\equiv_{\alpha}$,令 $s,s'\in S$,那么 $s\equiv_{\alpha}s'$ iff $\alpha(s)=\alpha(s')$[12].

给定原子公式 f,$var(f)$ 是出现在公式 f 中的变量集合. 例如 $var(x==y)$ 是 $\{x,y\}$. 对于原子公式集合 F,$var(F)=\bigcup\limits_{f\in F}var(f)$. 两个原子公式 f_1 和 f_2 是干扰的 iff $var(f_1)\bigcap var(f_2)\neq\varnothing$. 原子公式 $f\in F$ 的等价类称为 f 的公式簇,记为 $[f]$. 如果 $var(f_1)\bigcap var(f_2)\neq\varnothing$,有 $[f_1]=[f_2]$. 也就是说,一个变量 v 不能出现在属于两个不同公式簇的公式中. 称两个状态 $s,s'\in S$ 是等价的,iff $\bigwedge\limits_{f\in[f]}s\models f\Leftrightarrow s'\models f$. 也就是说两个状态是等价的当且仅当它们不能被公式簇中的公式区分[12].

对于 EFSM 模型,我们采用迁移上的谓词定义等价状态. 假设 $t.g$ 表示的是迁移 t 上的谓词集合,$t.s_s$ 是迁移 t 的起始状态. 令 t_i 和 t_j 是两条迁移,如果 $var(t_i.g)\bigcap var(t_j.g)\neq\varnothing$,并且状态 $t_i.s_s$ 和 $t_j.s_s$ 不能被 $t_i.g$ 所属的公式簇中的公式区分,称两个状态等价,即 $t_i.s_s\equiv_{\alpha}t_j.s_s$. 特别地,如果 $t_i.s_s\equiv_{\alpha}t_j.s_s$,$t_i.s_t$ 和 $t_j.s_t$ 都没有输出迁移,那么 $t_i.s_t\equiv_{\alpha}t_j.s_t$.

定义 5. 假设 EFSM $M=(S,init,V,I,O,T)$ 和 $M^{\alpha}=(S^{\alpha},init^{\alpha},V^{\alpha},I^{\alpha},O^{\alpha},T^{\alpha})$. M^{α} 是 M 的抽象,其中:

(1) S^{α} 是关于 α 的等价类集合;

(2) $init^{\alpha}=init\bigcup\{s\mid\exists s\in S\cdot s_0\in init\wedge\alpha(s_0)=\alpha(s)\}$；

(3) $V^{\alpha}=V$；

(4) $I^{\alpha}=I$；

(5) $O^{\alpha}=O$；

(6) $T^{\alpha}=\{t^{\alpha}:(s_s^{\alpha},in,g,op,s_t^{\alpha})\mid\exists s_s\in s_s^{\alpha},s_t\in s_t^{\alpha}\cdot T(s_s,s_t)\}$.

其中 $T(s_s,s_t)$ 表示状态 s_s 和 s_t 间的迁移.

对于抽象状态间的迁移 t^{α}，分几种情况讨论：

(1) 如果存在两条迁移 t_i 和 t_j 满足 $\alpha(t_i.s_s)=\alpha(t_j.s_s)=t^{\alpha}.s_s^{\alpha}\wedge\alpha(t_i.s_t)=\alpha(t_j.s_t)=t^{\alpha}.s_t^{\alpha}$，$var(t_i.g)\bigcap var(t_j.g)\neq\varnothing$ 且 $t_i.g\wedge t_j.g\neq$ false 则保留两条迁移，两条迁移上的操作作为它们目标状态的内部动作.

(2) 或者假设 t_i 的谓词不为空，而 t_j 的谓词条件为空即认为是 true，t_i 和 t_j 的开始状态和结束状态分别都等价，那么保留两条迁移，两条迁移上的操作作为它们目标状态的内部动作.

(3) 或者 t_i 和 t_j 的谓词条件都为空，并且它们的开始状态和结束状态分别都等价，如果 $t_i.in\neq t_j.in$，那么保留两条迁移，否则，合并这两条迁移，$t^{\alpha}.in=t_i.in$，$t^{\alpha}.g=$ true，两条迁移上的操作同样作为它们结束状态的内部动作.

(4) 两条迁移 t_i 和 t_j 上的谓词有共有变量，但谓词条件存在矛盾即 $t_j\wedge\rightharpoonup\phi t_i.g\wedge t_j.g=$ false，并且 $t_i.in=t_j.in$，那么对这两个迁移进行合并，$t^{\alpha}.in=t_i.in$，$t^{\alpha}.g=t_i.g\vee t_j.g$，两条迁移上的操作作为它们目标状态的内部动作. 迁移的输入序列、谓词条件和操作可以为空.

根据状态等价的定义，我们可知在图 1 中的状态 s_0，s_2，s_4 等价，$\alpha(s_0)=\alpha(s_2)=\alpha(s_4)=s_0^{\alpha}$，状态 s_1，s_3 等价，$\alpha(s_1)=\alpha(s_3)=s_1^{\alpha}$. 状态 s_5 单独形成一个等价类，$\alpha(s_5)=s_2^{\alpha}$. 图 1 的抽象 EFSM 如图 2.

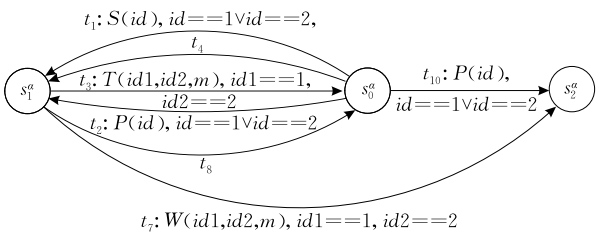

图 2　图 1 的抽象 EFSM 模型

5　测试用例生成

得到抽象模型后，采用模型检验技术在抽象模型上生成测试用例. 抽象约简了具体模型的状态数和迁移数，但验证的性质在抽象模型上仍保留.

定理 2.　假设两个 EFSM M 和 M^{α}，M^{α} 是 M 的抽象，ϕ 是 LTL 公式，$M^{\alpha}\models\phi$，那么 $M\models\phi$[3].

采用模型检验生成测试用例的过程是：根据测试覆盖准则，列出测试目标，把每个测试目标表示成陷阱性质，模型检验器验证陷阱性质，为每条陷阱性质生成一条反例，这些反例可作为测试用例集.

这里，我们在抽象模型上生成抽象测试用例，使用抽象精化方法对测试用例优化的同时要保证：(1) 抽象测试用例能够对应具体模型上的具体测试用例；(2) 满足抽象模型测试覆盖的抽象测试用例也满足具体模型的测试覆盖.

上述的第(1)点将在下一节讨论. 在分析第(2)点之前我们假定第(1)点已经满足. 根据定义 4，一个抽象状态对应一个具体状态集，设 β 是抽象函数 α 的逆，通过 $\beta(s^{\alpha})=\{s\mid s\in S\}$ 求出与抽象状态对应的具体状态集，所以能够覆盖所有抽象状态的测试用例能够覆盖所有具体状态. 抽象函数对谓词变量属于同一公式簇且谓词条件矛盾的迁移进行了合并，对其他迁移进行了保留. 对于迁移覆盖准则，我们定义抽象迁移 t^{α} 上的投影函数 $Proj(t^{\alpha})=\{(\beta(s_s^{\alpha}),in,g_i,op,\beta(s_t^{\alpha}))\mid\exists s_s^{\alpha},s_t^{\alpha}\in S^{\alpha},i\in\{1\cdots n\}\cdot T^{\alpha}(s_s^{\alpha},in,\bigvee g_i,op,s_t^{\alpha})\}$，其中 i 表示抽象迁移的谓词由 i 个具体迁移的谓词合并得到，通过这样的投影操作，能够把抽象迁移还原成具体模型上的迁移，加上那些保留的迁移，抽象迁移集经过投影操作后可以还原得到所有具体迁移.

从抽象模型生成的抽象测试用例是抽象状态的序列，对状态序列中的每个抽象状态使用函数 β 求出它所对应的具体状态集，然后对抽象状态间的每条抽象迁移，通过投影函数 $Proj$ 还原成具体状态间的具体迁移集，就得到了与抽象测试用例对应的具体测试用例集. 例如，为图 3 中的模型生成了一条抽象测试用例 $\langle s_0^{\alpha},t_1,s_1^{\alpha},t_8,s_0^{\alpha}\rangle$，抽象状态 s_0^{α} 和 s_1^{α} 对应的具体状态集分别 $\beta(s_0^{\alpha})=\{s_0,s_2,s_4\}$ 和 $\beta(s_1^{\alpha})=\{s_1\}$，抽象迁移 t_1 和 t_8 通过投影函数 $Proj$ 还原得到的具体迁移集分别是 $Proj(t_1)=\{t_1,t_2\}$ 和 $Proj(t_8)=\{t_8\}$，可以得到两条具体测试用例 $\langle s_0,t_1,s_1,t_8,s_4\rangle$ 和 $\langle s_0,t_2,s_1,t_8,s_4\rangle$. 根据迁移覆盖准则为图 3 中的模型生成了 8 条抽象测试用例，经过上述过程得到 11 条具体测试用例，这 11 条测试用例可以覆盖图 1 中的所有具体迁移. 所以可知与满足抽象模型迁移覆盖的抽象测试用例集对应的具体测试用例集也满足具体模型的迁移覆盖. 其他的覆盖准则我们也可以进行类似地分析.

6 反例判定和模型精化

如果 $M^{\alpha}\not\models\phi$,那么模型检验器就会返回一个抽象反例 $\pi^{\alpha}=\langle s_0^{\alpha},s_1^{\alpha},\cdots,s_n^{\alpha}\rangle$,根据定理 1 不能保证 $M\not\models\phi$,需要判定在具体模型中是否存在一个与 π^{α} 对应的具体反例 π. 如果存在这样的具体反例 π,那么 π^{α} 就可以作为抽象模型的一个测试用例,称这样的 π^{α} 为有效反例;否则就认为 π^{α} 是一个伪反例,它不能作为抽象模型的一个测试用例,需要对抽象模型进行精化以获得它的测试用例集.

6.1 反例判定

由于 π^{α} 的每个状态 $s_i^{\alpha}(i\geq 0)$ 对应一个具体状态集,因此,π^{α} 对应具体模型 M 中路径的集合,用 $path(\pi^{\alpha})$ 表示:$path(\pi^{\alpha})=\{\langle s_0,s_1,\cdots,s_n\rangle\mid \bigwedge_{0\leq i\leq n}\beta(s_i^{\alpha})=s_i\wedge s_0\in init\wedge \bigwedge_{0\leq i\leq n-1}(s_i,i,g,op,s_{i+1})\in T\}$.

π^{α} 是一个有效的反例当且仅当 $path(\pi^{\alpha})\neq\varnothing$. 可以通过迭代的方法判定 $path(\pi^{\alpha})\neq\varnothing$ 是否成立. 用 RS_i 表示 s_i^{α} 中的可达具体状态集,特别的 $RS_0=init$,对于任意状态 $s\in RS_i$,存在 RS_0 中的某一状态到达 s 的一条路径. 可以通过公式 $RS_{i+1}=\beta(s_{i+1}^{\alpha})\bigcap\{s\mid\exists s'\in\beta(s_i^{\alpha})\cdot s'\in RS_i\wedge T(s',s)\in T\}$ 计算每个抽象状态中的可达具体状态集. 那么,判断一条抽象反例是否是有效反例可以通过定理 3 完成.

定理 3. $path(\pi^{\alpha})\neq\varnothing\Leftrightarrow\forall 0\leq i\leq n:RS_i\neq\varnothing$.

证明. 先证"⇒",$path(\pi^{\alpha})\neq\varnothing$ 说明存在一条具体路径 $\langle s_0,s_1,\cdots,s_n\rangle$,每个状态都属于一个抽象状态 $\forall 0\leq i\leq n:s_i\in\beta(s_i^{\alpha})$,因为 $RS_0=s_0$,所以 $RS_0\neq\varnothing$,假设 $RS_0\neq\varnothing$ 且 $s_i\in\beta(s_i^{\alpha})$,根据 $path(\pi^{\alpha})$ 的定义有 $s_{i+1}\in\beta(s_{i+1}^{\alpha})$ 并且 $T(s_i,s_{i+1})\in T$,因此我们可得 $RS_{i+1}=s_{i+1}\neq\varnothing$.

再证"⇐",$\forall 0\leq i\leq n:RS_i\neq\varnothing$,可以从每个 RS_i 中选择满足 $T(s_i,s_{i+1})\in T$ 且 $s_i\in\beta(s_i^{\alpha})$ 的状态 s_i 同时满足初始状态为 s_0 来构造具体路径 $\langle s_0,s_1,\cdots,s_n\rangle$,则 $path(\pi^{\alpha})\neq\varnothing$. 证毕.

在图 2 中的抽象状态对应的具体状态集分别为 $\beta_0^{\alpha}=\{s_0,s_2,s_4\}$,$\beta_1^{\alpha}=\{s_1,s_3\}$,$\beta_2^{\alpha}=\{s_5\}$. 如果抽象反例为 $\langle s_0^{\alpha},s_1^{\alpha},s_2^{\alpha}\rangle$,那么该反例中各个抽象状态的可达具体状态集分别为 $RS_0=\{s_0\}$,$RS_1=\{s_1\}$,$RS_2=\varnothing$,$\langle s_0^{\alpha},s_1^{\alpha},s_2^{\alpha}\rangle$ 是一条伪反例.

在出现伪反例的情况下,必须对模型进行精化以删除伪反例.

6.2 模型精化

根据定理 2,出现伪反例是因为:(1) 存在某个 $i(0\leq i\leq n)$,$\forall 0\leq j\leq i:RS_j\neq\varnothing\wedge RS_{i+1}=\varnothing$,并且在抽象状态 s_i^{α} 和 s_{i+1}^{α} 之间存在迁移,同时 $\exists s,s'\in s_i^{\alpha}$,$s$ 从初始状态是可达的,但是 s 不能到达 $\beta(s_{i+1}^{\alpha})$ 中的状态,而 s' 从初始状态不可达,不过 s' 可以到达 $\beta(s_{i+1}^{\alpha})$ 中的状态. 那么称像 $\beta(s_i^{\alpha})$ 中 s 这样的状态为死路状态(dead-end state),称像 $\beta(s_i^{\alpha})$ 中 s' 这样的状态为坏状态(bad state),称 $\beta(s_i^{\alpha})$ 中除了死路状态和坏状态的状态为无关状态(irrelevant state);(2) 初始状态 $init^{\alpha}$ 和状态 s_1^{α} 之间存在迁移 t^{α},但是 $s_0\in\beta(init^{\alpha})$ 和 $\beta(s_1^{\alpha})$ 之间的迁移不属于 $Proj(t^{\alpha})$. 为了删除伪反例,对于第 1 种情况,需要把死路状态和坏状态划分到不同的抽象状态中;对于第 2 种情况,增加抽象状态 s_0,并在 s_0 和 $init^{\alpha}$ 添加迁移 $\{t\mid t\in T(s_0,\beta(init^{\alpha}))\}$.

使用抽象函数 α' 重新定义等价关系,它是原有抽象函数 α 的精化.

定义 6[13]. 称抽象函数 α' 是 α 的精化当且仅当 $\forall s_1,s_2\in S:(\alpha'(s_1)=\alpha'(s_2)\Rightarrow\alpha(s_1)=\alpha(s_2))$.

为了把死路状态和坏状态划分到不同的抽象状态中,定义抽象函数 $\alpha':\forall s_1,s_2\in\beta'(s_i^{\alpha})\cdot(\alpha'(s_1)=\alpha'(s_2)\Leftrightarrow(\exists s_1',s_2'\in\beta'(s_{i+1}^{\alpha})\cdot T(s_1,s_1')\in T\wedge T(s_2,s_2')\in T)\vee(\forall s\in\beta'(s_{i+1}^{\alpha})\cdot T(s_1,s)\notin T\wedge T(s_2,s)\notin T))$,其中 β' 是 α' 的逆.

也就是说,状态 s_1 和 s_2 是等价的,当且仅当 s_1,s_2 同时有或者没有到达 $\beta'(s_{i+1}^{\alpha})$ 中状态的迁移. 这样,抽象函数 α' 将 $\beta'(s_i^{\alpha})$ 中的死路状态和坏状态划分到了不同的抽象状态.

为了去除图 2 中的伪反例 $\langle s_0^{\alpha},s_1^{\alpha},s_2^{\alpha}\rangle$,分析知 s_1 是一个死路状态,s_3 是一个坏状态,使用 α' 将它们划分到两个不同的抽象状态,精化后的状态等价关系是 $\alpha'(s_0)=\alpha'(s_2)=\alpha'(s_4)=s_0^{\alpha}$,$\alpha'(s_1)=s_1^{\alpha}$,$\alpha'(s_5)=s_2^{\alpha}$,$\alpha'(s_3)=s_3^{\alpha}$. 图 3 是精化后的模型.

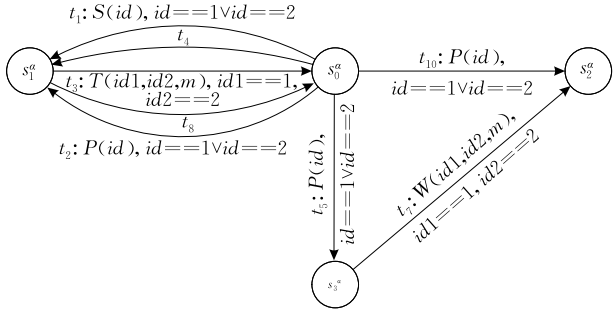

图 3 对图 2 精化后的 EFSM

如果模型中仍然存在伪反例,可以继续使用上述步骤对模型进行精化,直到生成的所有反例能够

满足具体模型的测试覆盖准则. 例如，对于迁移覆盖准则，如果为某一迁移对应的测试目标生成的反例是伪反例，那么精化模型，在精化后的模型上再为该迁移对应的测试目标生成反例，直到生成能够覆盖该迁移的真反例. 可知图 3 中是图 1 最后的抽象模型.

根据迁移覆盖准则，图 1(具体模型)中有 11 条迁移，根据通过模型检验技术生成测试用例的方法，需要为每个迁移生成一条覆盖它的测试用例，故总共生成 11 条测试用例，例如，对于迁移 t_7，生成测试用例$\langle s_0, t_1, s_1, t_3, s_2, t_5, s_3, t_7, s_4\rangle$；图 3(抽象模型)中有 8 条抽象迁移，那么需要 8 条测试用例，例如，对于抽象模型中的迁移 t_7，生成测试用例$\langle s_0^a, t_5, s_3^a, t_7, s_2^a\rangle$因为能够覆盖所有抽象迁移的抽象测试用例也能覆盖所有具体迁移，所以这 8 条迁移能够覆盖图 1 中的所有迁移；下面对在图 3 的抽象模型上采用基于可满足性的方法生成测试用例的过程进行详细描述. 表 1 中是 8 个测试目标所对应的 CNF 公式的难度及排名，根据排名升序排列测试目标，我们首先为迁移 t_8生成测试用例$\langle s_0^a, t_1, s_1^a, t_8, s_0^a\rangle$，并在测试目标集中删除 t_8对应的测试目标，然后再把该测试用例与余下的每个测试目标进行合取得到 CNF 公式，使用 SAT 求解器 Yices① 求解每个 CNF 公式的可满足性，可得到该测试用例覆盖迁移 t_1，从测试目标集中删除 t_1对应的测试目标，重复以上过程，直到所有的测试目标都被测试用例覆盖(即最后的测试目标集为空)，最后生成 6 条测试用例.

表 1　测试目标的难度及排名

测试目标	文字数	子句数	文字数/子句数	排名
$G\rightarrow t_1$	61	52	1.173076923	5
$G\rightarrow t_2$	61	52	1.173076923	6
$G\rightarrow t_3$	57	51	1.117647059	4
$G\rightarrow t_4$	52	51	1.019607843	2
$G\rightarrow t_5$	61	51	1.196078431	7
$G\rightarrow t_7$	58	53	1.094339623	3
$G\rightarrow t_8$	51	51	1	1
$G\rightarrow t_{10}$	61	51	1.196078431	8

由此可见，对具体模型进行抽象精化后，测试用例确实得到了约简，结合基于可满足性的测试用例生成方法可以更好地约简测试用例. 下面以更为详细的实验进行说明.

7　实验及分析

我们用 5 个 EFSMs[1] 进行实验分析，然后对自控镇痛泵系统② 中的警报侦测组件所包含的 EFSM 模型进行实例分析. 本文使用模型检验工具 NuSMV 2.5.4③ 生成反例(测试用例).

在实验中，主要讨论以下几个问题：

(1) 在满足迁移覆盖准则的情况下，以模型检验技术生成的测试用例(Test case generation based on model checking from concrete model，称为传统方法)为基线，分析使用抽象精化方法(Test case generation based on abstraction refinement from abstract model，称为 AR 方法)生成的测试用例数目和总长度的约简情况；

(2) 比较我们之前提出的基于可满足性的测试用例生成方法[5] (Test case generation based on satifiability from concrete model，称为 SAT 方法)和 AR 方法，分析哪种方法能够更有效地约简测试用例集；

(3) 结合 SAT 方法和 AR 方法(SAT+AR)是否比单独使用其中一种方法能够更有效地约简测试用例集；

(4) 在约简测试用例集的同时，SAT、AR 和 SAT+AR 方法的迁移覆盖率是否会减低？

(5) SAT+AR 方法的查错能力是否会减低？

实验中使用的 5 个 EFSM 是：Flight Safety System EFSM，Transport Protocol EFSM，Lift System EFSM，ATM EFSM 和 Inres initiator EFSM. 它们的具体模型如图 4～图 8.

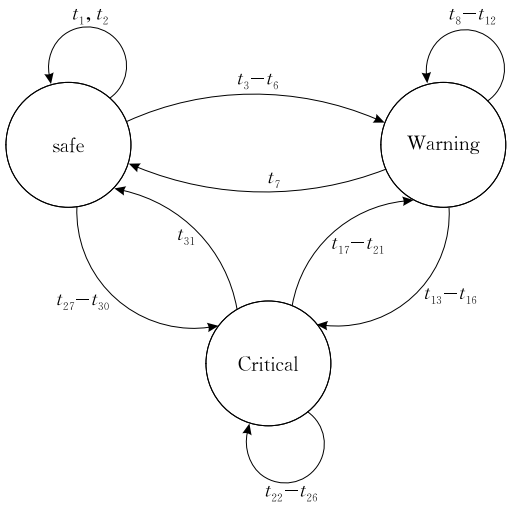

图 4　Flight Safety System EFSM

① The Yices SMT Solver. http://yices.csl.sri.com/documentation.shtml

② Generic Patient Controlled Analgesia Pump Model. http://rtg.cis.upenn.edu/gip.ph

③ NuSMV Model Checker. http://nusmv.fbk.eu/NuSMV/download/gettiNng-v2.html

图 4～图 6 中模型的具体迁移描述参见文献[1].

实验主要包括以下几个步骤：

(1)在具体模型上采用基于模型检验技术生成测试用例.把模型中的每一个迁移(测试目标)表示成一个陷阱性质,然后通过模型检验器为每一个迁移生成一条覆盖它的反例.

(2)在具体模型上采用我们在文献[5]中提出的 SAT 方法生成测试用例.文献[5]中的实验表明 SAT 方法对测试用例有很好的约简效果.

(3)使用本文中提出的抽象精化过程为每个 EFSM 模型构造抽象模型,然后在抽象模型上采用模型检验方法生成测试用例(AR 方法).实验表明 AR 方法也可以很有效地约简测试用例.

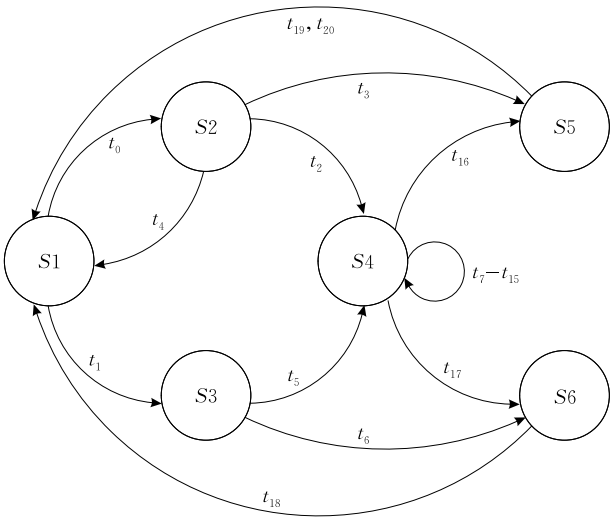

图 5 Transport Protocol EFSM

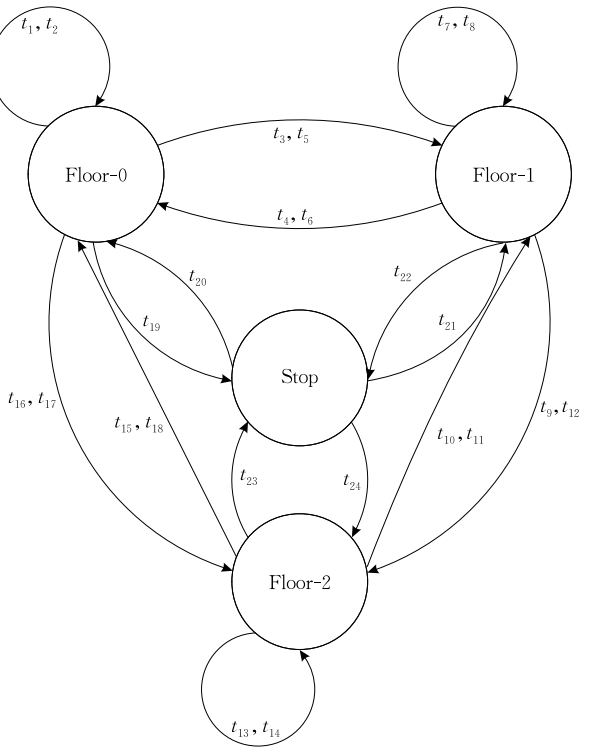

图 6 Lift System EFSM

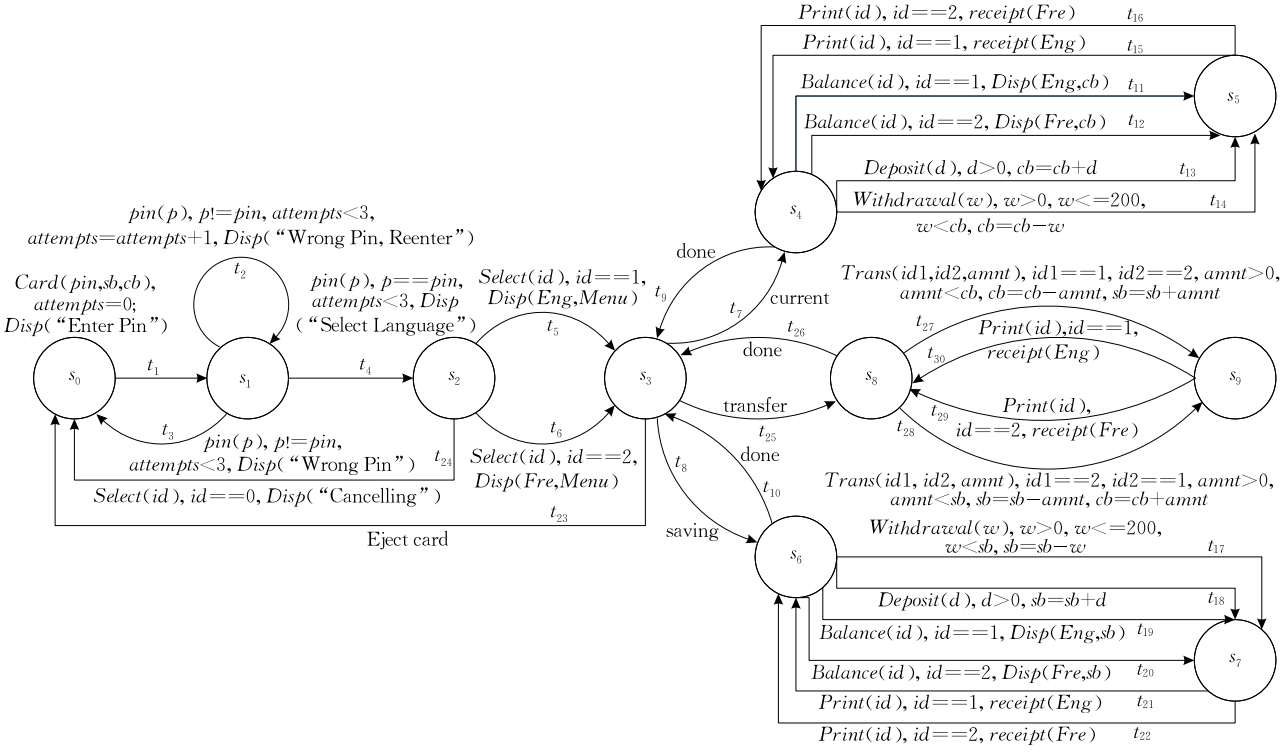

图 7 ATM EFSM

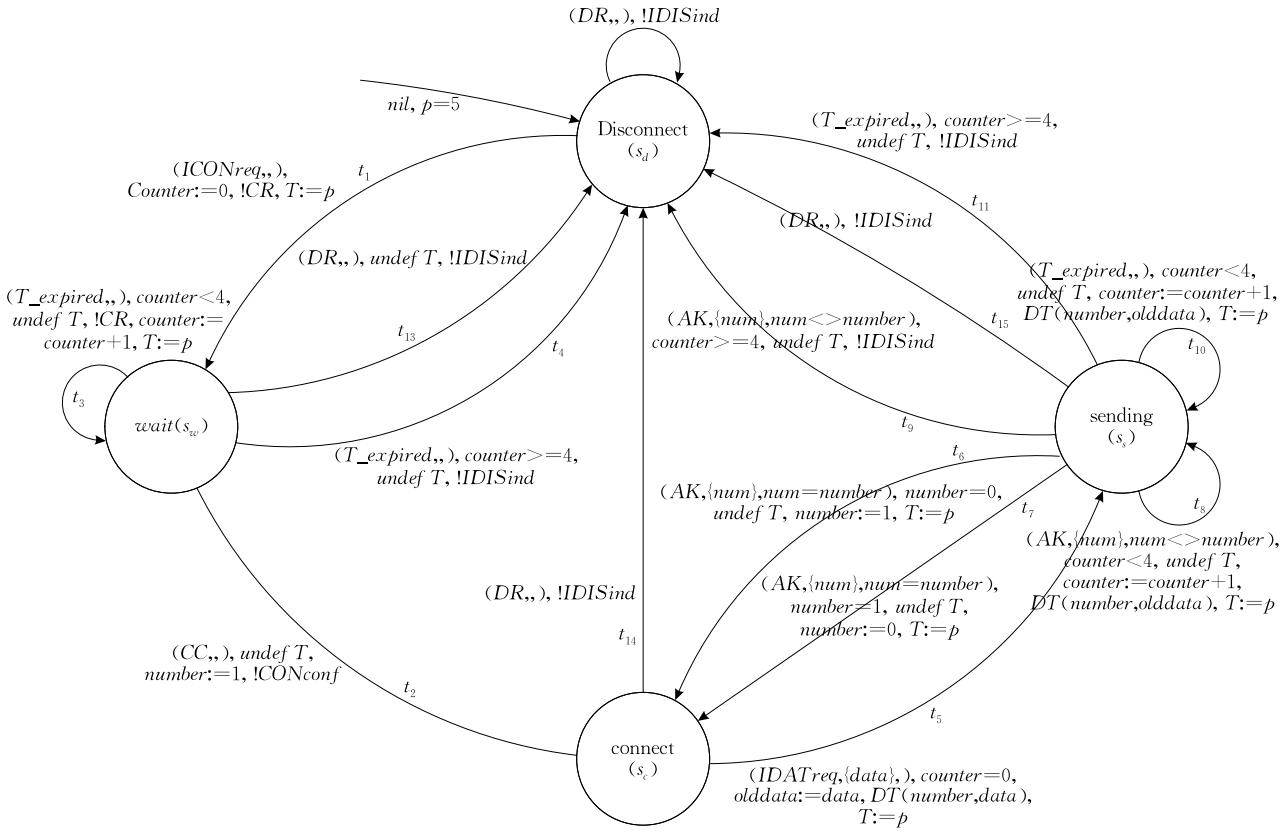

图 8　Inres initiator EFSM

(4)用 AR 和 SAT 结合的方法来生成测试用例. 采用抽象精化过程为每个 EFSM 模型构造抽象模型,在抽象模型上采用 SAT 方法生成测试用例集.

图 9～图 13 是根据本文提出的抽象精化方法为图 4～图 8 生成的抽象模型.

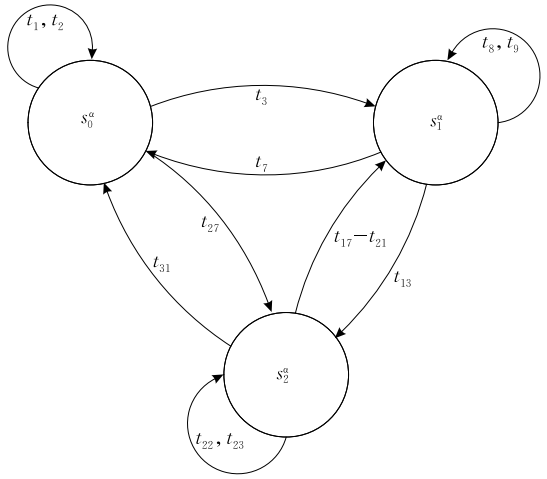

图 9　Flight Safety System EFSM 的抽象模型

表 2 给出了采用本文给出的使用抽象精化过程前后 5 个 EFSM 模型中模型元素的变化情况. 其中 FSS、TP、LS、ATM 和 II 分别代表 Flight Safety System EFSM, Transport Protocol EFSM, Lift System EFSM, ATM EFSM 和 Inres initiator EFSM.

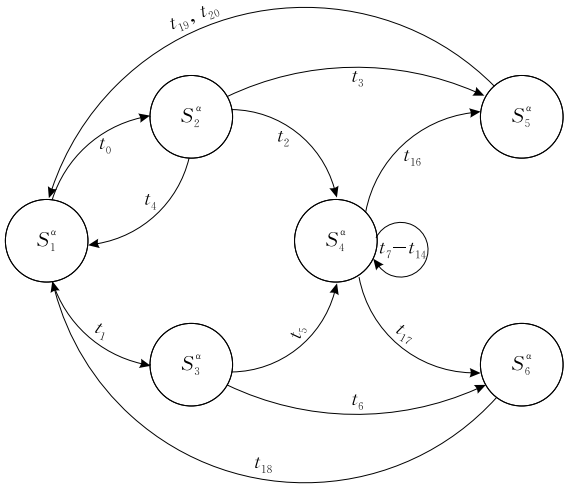

图 10　Transport Protocol EFSM 的抽象模型

表 2　抽象精化前后模型元素的变化情况

	具体模型		抽象模型	
	状态数	迁移数	状态数	迁移数
FSS	3	31	3	18
TP	6	21	6	15
LS	4	24	4	18
ATM	10	22	8	17
II	4	16	4	14

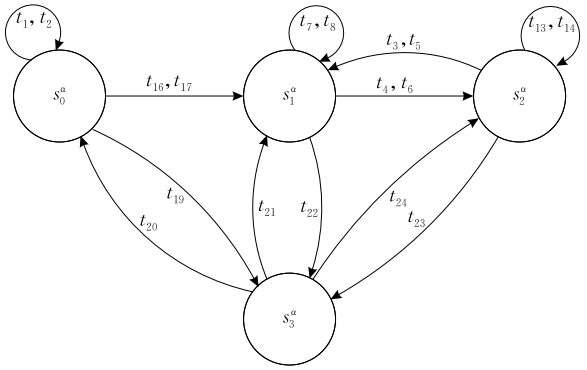

图 11 Lift System EFSM 的抽象模型

在表 3 中列出了使用这 4 种方法为 5 个 EFSM 生成测试用例的情况,包括测试用例的数量(表示为 N)和测试用例的总长度(表示为 L).在表中传统方法用 M 表示.

表 3 4 种方法生成的测试用例情况

模型	M		SAT		AR		AR+SAT	
	N	L	N	L	N	L	N	L
FSS	31	50	23	44	18	32	9	23
TP	20	54	14	44	15	36	7	22
LS	24	43	15	34	18	38	7	27
ATM	30	133	17	82	17	76	9	42
II	15	44	12	38	14	36	7	23

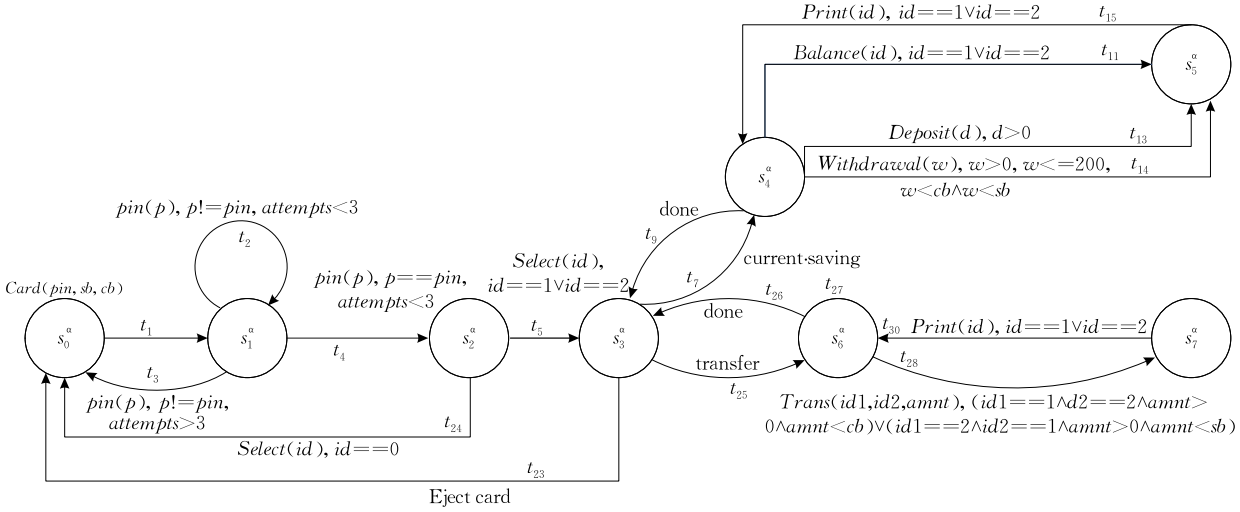

图 12 ATM EFSM 的抽象模型

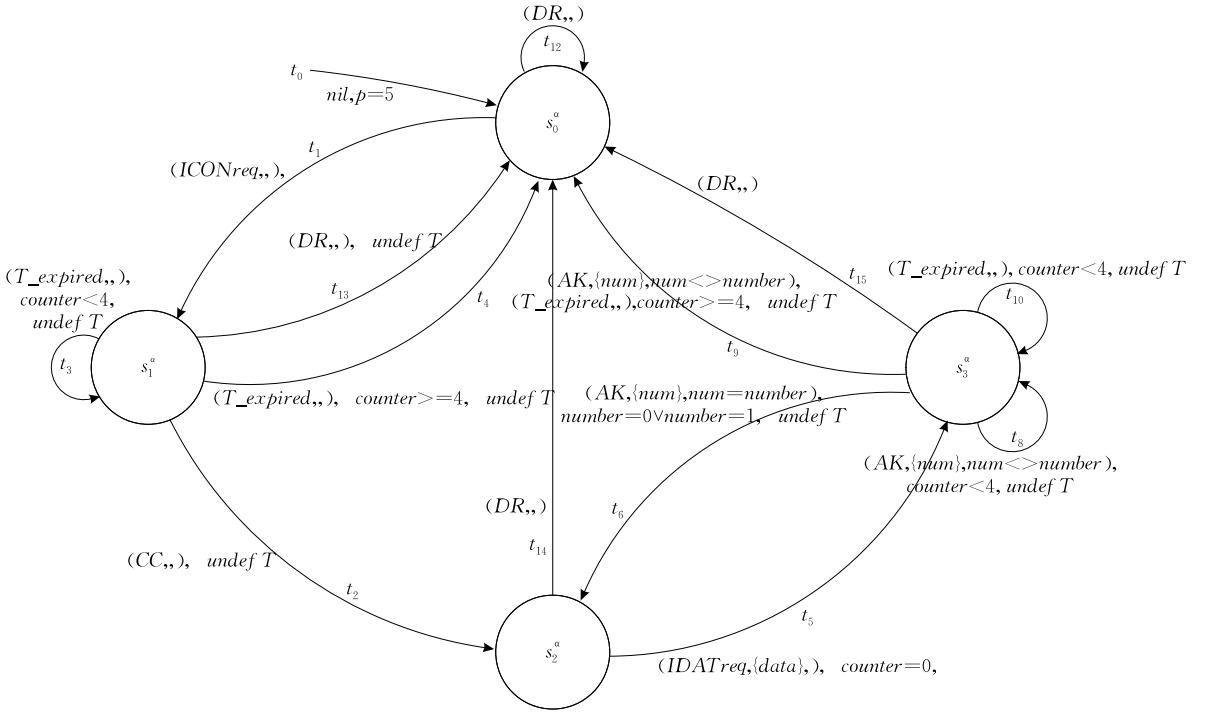

图 13 Inres initiator EFSM 的抽象模型

从表 3 中可以看出，SAT、AR、AR+SAT 对测试用例的数目和总长度有很好的约简效果. 对于 5 个 EFSM 模型，AR 方法都对具体模型进行了一定程度的抽象. 对于 Flight Safety System 模型，AR 方法生成的测试用例数和测试用例总长度要比传统方法小，并且也比 SAT 方法的小. 对于 Lift System 模型、Transport Protocol 模型和 ATM 模型，AR 方法都对传统方法生成的测试用例集进行了很大的约简，而对于 Inres initiator 模型，AR 方法约简效果不佳. 并且对于除了 Flight Safety System 和 ATM 以外的其他模型，它的约简效果要比 SAT 的差，这是因为模型中迁移的共有变量很少，即使迁移有共有变量，但是它们的谓词条件不矛盾，导致它们不能合并，AR 方法对这几个模型没有很好的约简，所以造成测试用例的约简效果不佳；而 SAT 方法生成的测试用例较长，通过可满足性分析不再生成与已有测试用例相同或被它包含的测试用例，所以对于这 3 个不能得到很好抽象的模型，它生成的测试用例数和测试用例总长度都要比 AR 方法的小. AR+SAT 方法具有最佳的约简效果，它首先对模型进行抽象精化，这样，抽象精化对模型的状态和迁移有了一定的约简，然后通过 SAT 方法生成数目更少、总长度更小的测试用例集.

表 4 是 3 种方法的测试用例约简比例. 对于 Flight Safety System 和 ATM 模型，AR 方法对测试用例的约简比例比 SAT 方法高. 对于其他 3 个模型，SAT 方法对测试用例的约简比例比 AR 方法高. 对于所有模型，AR+SAT 方法在 AR 方法和 SAT 方法的基础上对测试用例进一步约简.

表 4　3 种方法的测试用例约简比例

模型	SAT/%		AR/%		(AR+SAT)/%	
	N	*L*	*N*	*L*	*N*	*L*
FSS	26	12	42	36	71	54
TP	30	19	25	33	65	59
LS	38	21	25	12	71	37
ATM	43	38	43	43	70	68
II	20	14	7	18	53	48

为了在约简的同时保证测试用例集的质量，我们从迁移覆盖率和查错能力两方面对约简的测试用例集进行分析.

基于模型检验技术的测试用例生成方法为每个迁移都要生成一条测试用例，所以它的迁移覆盖率为 100%，在此基础上，我们分析 SAT、AR、AR+SAT 这 3 种方法生成的测试用例集是否降低了迁移覆盖率. 在此之前，首先对本文第 5 节中提到的能够覆盖所有抽象迁移的抽象测试用例也能够覆盖所有具体迁移进行分析. 使用 AR 方法分别为 5 个 EFSMs 的抽象模型生成满足迁移覆盖的抽象测试用例集，然后从抽象测试用例集得到对应的具体测试用例集，分析抽象测试用例集对具体模型中具体迁移的覆盖情况. 实验结果见表 5.

表 5　抽象测试用例集对具体迁移的覆盖情况

模型	抽象测试用例数	具体测试用例数	迁移覆盖率/%
FSS	18	31	100
TP	15	20	100
LS	18	24	100
ATM	17	30	100
II	14	15	100

从表 5 可知，满足抽象模型迁移覆盖的抽象测试用例集也能满足具体模型的迁移覆盖. 因而，针对 AR 和 AR+SAT 方法我们可以在抽象模型上分析抽象测试用例的迁移覆盖. 表 6 是 3 种方法生成约简的测试用例集针对 5 个 EFSM 抽象模型的迁移覆盖率.

表 6　3 种方法的迁移覆盖率

模型	SAT	AR	(AR+SAT)/%
FSS	100	100	100
TP	100	100	100
LS	100	100	100
ATM	100	100	100
II	100	100	100

从表中可知 3 种方法并未降低针对 EFSM 模型的迁移覆盖率.

在文献[5]的实验中我们采用变异分数来衡量测试用例集的查错能力. 在模型上采用变异操作在状态和迁移中植入错误，通过比较测试用例集在约简前后发现错误的百分比来分析约简的测试用例集的查错能力是否降低. 实验结果表明 SAT 方法在约简测试用例同时，查错能力和传统方法相同，同时抽象模型的迁移集保留了具体模型的迁移或者由具体模型的多个迁移合并而来，因而 AR+SAT 方法的测试用例集的查错能力也与传统方法相同.

下面我们对自控镇痛泵系统中的警报侦测组件所包含的 EFSM 模型进一步对上面得到的实验结果进行论证. 警报侦测组件由 4 个并行子状态机组成：警报侦测控制器、第 1 层警报侦测组件、第 2 层警报侦测组件和警告侦测组件. 警报侦测控制器接收由其他子状态机报告的警报、警告并且决定什么通知应该报告给状态控制器. 第 1 层警报侦测组件

接收来自系统模型的传感信号输入,检查信号值的正值表示泵出现关键性失效的这类信号子集,当任何信号子集的值为正时把第一层警报报告给警报侦测控制器.第 2 层警报侦测组件接收来自系统模型的传感信号输入,检查信号值的正值表示当前注射液没有被正确传输的这类信号子集,检查信号值的正值表示泵出现关键性失效的这类信号子集.警告侦测组件接收来自系统模型的传感信号输入,检查信号值的正值表示泵出现某些异常但仍能安全工作的这类信号子集.4 个子组件的 EFSM 模型在这里不再给出.图 14～图 17 分别是对 4 个子组件的 EFSM模型通过本文的抽象精化方法得到的抽象模型.图 18 是 4 个抽象模型的并行组合.

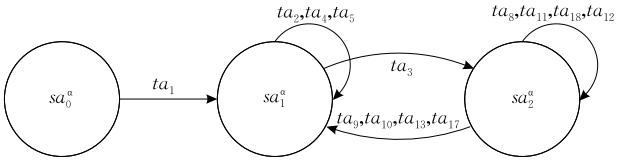

图 14 警报侦测控制器组件 EFSM 抽象模型

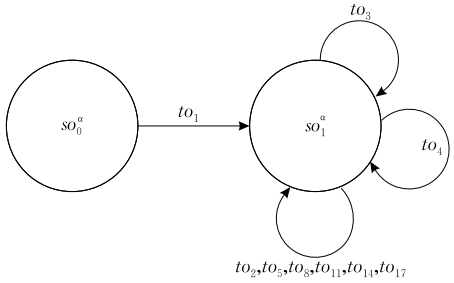

图 15 第 1 层警报侦测组件 EFSM 抽象模型

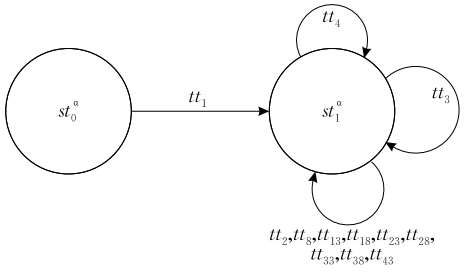

图 16 第 2 层警报侦测组件 EFSM 抽象模型

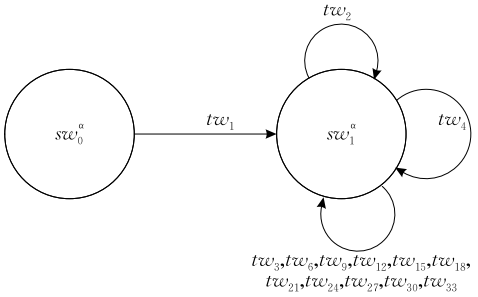

图 17 警告侦测组件 EFSM 抽象模型

表 7 是抽象精化前后并行 EFSM 模型中状态数和迁移数的变化情况.

表 7 抽象精化前后并行模型元素的变化情况

	并行 EFSM 模型	并行 EFSM 抽象模型
状态数	120	15
迁移数	119	96

表 8 是 4 种方法为警报侦测组件的并行 EFSM 生成的测试用例情况.

表 8 4 种方法生成的测试用例情况

模型	M		SAT		AR		AR+SAT	
	N	*L*	*N*	*L*	*N*	*L*	N	L
并行 EFSM	119	729	61	442	96	686	29	284

从表 8 可以看出,SAT 方法、AR 方法和 AR+SAT 结合的方法都在很大程度上约简了测试用例数目和总长度.SAT 方法的约简效果要比 AR 方法的好,主要原因是 AR 方法进行抽象精化时,得到的等价状态较多,但是能够合并的迁移不多,所以导致 AR 方法约简效果比 SAT 方法差.但是,通过 AR 方法对各个子组件进行抽象精化后再并行组合,使得覆盖某些抽象迁移的测试用例缩短,所以 AR+SAT 方法在 SAT 方法的基础上进一步很好的约简了测试用例数目和总长度.

表 9 是 3 种方法为警报侦测组件的并行 EFSM 生成的测试用例的约简比例.

表 9 3 种方法的测试用例约简比例

模型	SAT/%		AR/%		(AR+SAT)/%	
	N	*L*	*N*	*L*	*N*	*L*
并行 EFSM	49	39	19	6	76	61

在表 3 中的数据表明,针对那 5 个 EFSM 模型,SAT 方法的约简比例要比 AR 方法的高,表 9 可得出同样的结论.与前面的 5 个 EFSMs 相比,在并行组合的模型上 AR+SAT 能够约简更多的测试用例数目,同时也能很好的约简测试用例总长度.

最后,3 种方法的迁移覆盖率都是 100%,查错能力也与传统方法相同.

8 相关工作

有关测试用例优化的研究有很多.Hamon 等人[14]在模型检验器 SAL 中集成了测试用例生成,最后得到的测试用例集虽然在测试用例的数目上会减少,但它的总长度不一定会减少,并且也没说明哪

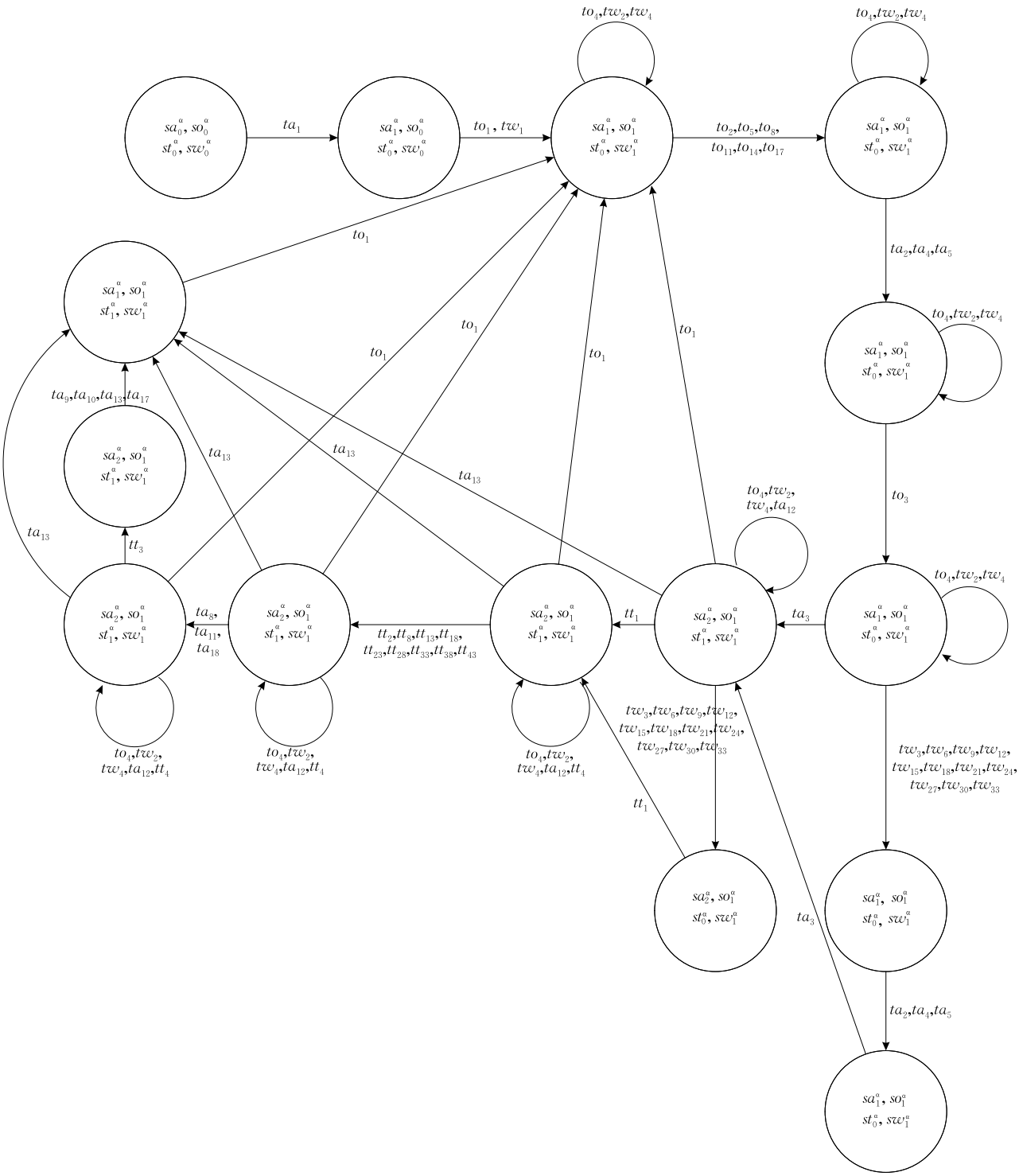

图 18　警报侦测组件的并行 EFSM 抽象模型

些测试用例可以扩展. Ammann 等人[15]把生成的测试用例表示成模型，在该模型上模型检验余下的测试目标，来判定它们是否被该测试用例覆盖，这涉及到频繁地从测试用例到模型的转换及对模型检验器的调用. Fraser 等人[16]在生成测试用例后，使用 LTL 重写规则消除目前已被覆盖的测试目标. Zeng 等人[13]结合 CTL 重写规则进行了测试目标约简和测试用例集约简. 其中都没有给出选择测试目标次序的方法，这恰恰会直接影响测试用例约简的效果. 在文献[5]中我们提出一种在测试用例生成时约简的方法，根据 CNF 公式的可满足性约简冗余的测试用例. 章晓芳等人[17]提出了一种基于测试需求的测试用例约简方法. 郭曦等人[18]使用谓词抽象约简测试用例，使用谓词抽象划分等价状态，从抽象模型生

成测试用例,文中的建模工具是有限状态机模型并且文中没有考虑出现伪反例求精模型的情况. Gong 等人[19]基于覆盖矩阵和路径向量约简的方法来约简为程序生成的测试用例集,该方法只分析了约简的测试用例集的查错能力但没有分析它的覆盖情况. Quentin 等人[20]基于事件序列的组合覆盖准则进行测试用例约简,并在 GUI 应用上评估了他们的方法. Ng 等人[21]使用形式概念分析获得模型中实体的包含关系,从状态机模型生成约简的测试用例集,文中只分析了测试用例集的迁移覆盖但没有分析它的查错能力. Wu 等人[22]使用商空间理论对测试集进行划分,使用最小测试集生成算法生成最小测试集,文中没有对提出的方法进行实验评估. Kumar 等人[23]使用模糊分簇把相似的测试用例分配到相同的簇,找到冗余的测试用例,该方法是在生成测试用例集后再进行约简. Harald 等人[24]使用模型检验技术从 UML 状态图生成测试用例集,然后合并相关用例对得到约简的测试用例集,该方法会为生成那些冗余的测试用例而调用模型检验器,增加了测试用例生成的成本. 文献[25-26]把测试用例集约简问题看成是优化问题. 此外,有很多研究工作(例如文献[27-28])把启发式算法应用到测试用例集选择,即从已有的测试用例集中选择一个子集. 本文以 EFSM 作为建模工具,根据公式簇建立状态等价关系,使用抽象精化方法构建抽象模型,结合 SAT 方法在生成测试用例的过程中约简测试用例集,一方面减少了调用模型检验器的次数,节省了测试用例生成的成本,另一方面有效地约简了测试用例集,缩减了测试用例执行的成本,此外,在约简测试用例的同时并未降低它的迁移覆盖率和查错能力.

9 总结与进一步工作

本文将迁移上的谓词划分成公式簇,根据公式簇建立状态的等价关系,分几种情况讨论了如何合并相关迁移,进而构建初始的抽象模型. 在抽象模型上通过模型检验技术生成测试用例,并说明了满足抽象模型覆盖的测试用例也能满足具体模型的覆盖. 给出了判定伪反例和精化模型的方法. 在最终的抽象模型(没有反例的抽象模型)上使用基于可满足性的测试用例生成方法生成反例. 实验表明与传统方法、SAT 方法、AR 方法相比,结合 AR 和 SAT 方法能最有效地约简测试用例的数目及总长度,与此同时,并未降低迁移覆盖率和查错能力. 最后并行组合模型的实例也说明,SAT+AR 方法相比单独使用 AR 方法和 SAT 方法都具有更好的约简效果. 本文的方法既节省了测试用例生成的成本,又缩减了测试用例执行的成本.

将来,我们将把本文的方法应用到一个较大的实际系统,并开发一个抽象精化和 SAT 结合的测试用例优化工具. 研究更好的抽象精化过程,获得更约简的抽象模型,从而更好地约简测试用例集.

参 考 文 献

[1] Kalaji A S, Hierons R M, Swift S. An integrated search-based approach for automatic testing from extended finite state machine (EFSM) models. Information and Software Technology, 2011, 53(12): 1297-1318

[2] Gargantini A, Heitmeyer C. Using model checking to generate tests from requirements specifications. ACM SIGSOFT Software Engineering Notes, 1999, 24(6): 146-162

[3] Zeng Hong-Wei, Miao Huai-Kou. Verification of component composition based on abstraction refinement. Journal of Software, 2008, 19(5): 1149-1159(in Chinese)
(曾红卫, 缪淮扣. 构件组合的抽象精化验证. 软件学报, 2008, 19(5): 1149-1159)

[4] Clarke E, Grumberg O, Jha S, et al. Counterexample-guided abstraction refinement//Proceedings of the International Conference on Computer Aided Verfication. Chicago, USA, 2000: 154-169

[5] Lu Gong-Zheng, Miao Huai-Kou, Gao Hong-Hao. Reduction of model-checking based test generation using satisfiability. Applied Mathematics & Information Sciences, 2015, 9(1L): 89-96

[6] Yang Rui, Chen Zhen-Yu, Xu Bao-Wen, et al. Improve the effectiveness of test case generation on EFSM via automatic path feasibility analysis//Proceedings of the 2011 IEEE 13th International Symposium on High-Assurance Systems Engineering. Boca Raton, USA, 2011: 17-24

[7] Keijo H, Ilkka N. Bounded LTL model checking with stable models//Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning. Vienna, Austria, 2001: 200-212

[8] Bourhfir C, Dssouli R, Aboulhamid E M. Automatic test generation for EFSM-based systems. Montreal, Departement d'Informatique et de Recherche Operationnelle, Universite de Montreal, Technique Report: H3C 3J7, 1996

[9] Frase G, Wotawa F, Ammann P. Issues in using model checkers for test case generation. Journal of Systems and Software, 2009, 82(9): 1403-1418

[10] Clarke E, Biere A, Raimi R, Zhu Y S. Bounded model checking using satifiability solving. Formal Method in System Design, 2001, 19(1): 7-34

[11] Marques-Silva J P, Sakallah K A. GRASP: A search algorithm for propositional satisfiability. IEEE Transaction on Computers, 1999, 48(5): 506-521

[12] Clarke E, Grumberg O, Jha S, et al. Counterexample-guided abstraction refinement symbolic model checking. Journal of the ACM, 2003, 50(5): 752-794

[13] Zeng Hong-Wei, Miao Huai-Kou. Optimization of model checking-based test generation. Journal of Computer-Aided Design & Computer Graphics, 2011, 23(3): 496-502(in Chinese)
(曾红卫，缪淮扣. 优化基于模型检测的测试生成. 计算机辅助设计与图形学学报，2011，23(3)：496-502)

[14] Hamon G, Moura L, Rushby J. Generating efficient test set with a model checker//Proceedings of the 2nd International Conference on Software Engineering and Formal Methods. Beijing, China, 2004: 261-270

[15] Ammann P E, Black P E. A specification-based coverage metric to evaluate test set//Proceedings of the 4th IEEE International Symposium on High-Assurance Systems Engineering. Washington DC, USA, 1999: 239-248

[16] Fraser G, Wotawa F. Using LTL rewriting to improve the performance of model checker-based test case generation//Proceedings of the 3rd International Workshop on Advances in Model-Based Testing. London, UK, 2007: 64-74

[17] Zhang Xiao-Fang, Xu Bao-Wen, Nie Chang-Hai, et al. An approach for optimizing test suite based on testing requirement reduction. Journal of Software, 2007, 18(4): 821-831(in Chinese)
(章晓芳，徐宝文，聂长海等. 一种基于测试需求约简的测试用例集优化方法. 软件学报，2007，18(4)：821-831)

[18] Guo Xi, Zhang Huan-Guo. Approach for reduced test suite generation based on predicate abstraction. Journal on Communications, 2012, 33(3): 35-43(in Chinese)
(郭曦，张焕国. 基于谓词抽象的测试用例约简生成方法. 通信学报，2012，33(3)：35-43)

[19] Gong Dan-Dan, Wang Tian-Tian, Su Xiao-Hong, Ma Pei-Jun. A test-suite reduction approach to improving fault-localization effectiveness. Computer Languages, Systems & Structures, 2013, 39(3): 95-108

[20] Quentin M, Ryan M, Renee B. Test suite reduction by combinatorial-based coverage of event sequences//Proceedings of the 7th IEEE International Conference on Software Testing, Verification and Validation. Cleveland, USA, 2014: 128-132

[21] Ng P, Fung R Y K. Model based test suite reduction with concept lattice//Proceedings of the 2008 Advanced Software Engineering & Its Applications. Hainan, China, 2008: 3-8

[22] Wu Lei, Li Long-Shu. A minimal test suite generation method based on quotient space theory//Proceedings of the 6th International Conference on Rough Sets and Knowledge Technology. Banff, Canada, 2011: 579-584

[23] Kumar G, Bhatia P K. Software testing optimization through test suite reduction using fuzzy clustering. Computer Science and Information Technologies, 2013, 1(3): 253-260

[24] Harald C, Thomas S H. Efficient reduction of model-based generated test suites through test case pair prioritization//Proceedings of the 7th Workshop on Model-Driven Engineering, Verification, and Validation. Västerås, Sweden, 2010: 37-42

[25] Gotlieb A, Marijan D. FLOWER: Optimal test suite reduction as a network maximum flow//Proceedings of the 2014 International Symposium on Software Testing and Analysis. San Jose, USA, 2014: 171-180

[26] Arito F, Chicano F, and Alba E. On the application of SAT solvers to the test suite minimization problem//Proceedings of the 4th International Symposium on Search Based Software Engineering. Riva Del Garda, Italy, 2012: 45-59

[27] Nagar R, Kumar A, Kumar S. Implementing test case selection and reduction techniques using meta-heuristics//Proceedings of the 5th International Conference on Confluence—The Next Generation Information Technology Summit. Noida, India, 2014: 837-842

[28] Zamli K Z, Mohd H, Mohd H, et al. Simulated annealing based strategy for test redundancy reduction//Proceedings of the 13th International Conference on Intelligent Software Methodologies, Tools, and Techniques. Langkawi, Malasia, 2014: 818-832

LU Gong-Zheng, born in 1981, Ph. D., lecturer. His current research interests include formal methods and software testing.

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

Background

Software testing is an important method to ensure the software quality, but it is costly. To reduce the cost of software testing, performing testing automatically is necessary. And model-based testing is an attractive research topic in

this area.

But the test suite generated from the model is general large which cause the high cost and low efficient of test execution. So test suite should be reduced before its execution.

There are several methods were proposed to reduce the test suite. They can be fall into: (1) test source reduction, (2) test case reduction after generation, (3) test case reduction before generation. Test source reduction reduces the model or the test goal. And the test case reduction after generation first generation all the test cases and then analyze the relations of the test cases to reduce the redundant test cases, the generation cost of the test cases is high and generating all the test cases is not necessary. Test case reduction before generation can analyze whether the test case generated coverage the remaining test goals or not to reduce the test suite, it should not generate all the test cases. Our method proposed in this paper use the abstraction refinement method to reduce the EFSM model and use the SAT method (test case reduction before generation) to reduce the test cases generated from the abstraction model.

In this paper, we use the formula cluster in the predicates on the transitions of the EFSM model to define the abstraction function which used to separate the equivalent states. We explain that the test suite generated from abstraction model can satisfy the testing coverage of the abstraction model can also satisfy the testing coverage of the original model. The counterexample is generated from the abstraction model using the method based on satisfiability we proposed before. And then we give a method to decide whether the counterexample is a spurious counterexample or not. Counterexample-guided refinement framework is used to refine the model to delete the spurious counterexample. The experimental results show that the method proposed in this paper has the best reduction effect among the methods we compared with.

This paper was supported by the National Natural Science Foundation of China under Grant Nos. 61170044 and 61572306, the Shanghai Leading Academic Discipline Project of China under Grant No. J50153, and the Suzhou Vocational University Research Fund under Grant No. SVU2015YY01.