基于模型检测技术的变异测试用例生成方法

杨 红 洪 玫 屈媛媛
(四川大学计算机学院 成都 610065)

摘 要 为了进行基于模型的软件测试变异分析,文中提出了一种基于模型检测的变异测试用例生成方法。基于模型检测工具 UPPAAL 的形式化分析与测试框架,首先用符合规范的时间自动机模型描述被测系统;然后基于时间自动机模型的基本结构和语法,对系统模型进行一组变异操作,并模拟实现时可能出现的一些错误;对变异后的模型分别使用 UPPAAL Yggdrasil 工具,生成一组能覆盖变异区域的测试用例;在系统变异模型上执行生成的测试用例,根据测试执行结果(是否能"杀死"变异体)筛选出一组有效的测试用例。通过实例验证,所提方案生成的测试用例是有效的,且测试用例集变异分数优于现有的基于状态机复制的变异测试用例自动生成方法和基于模型中变换覆盖的变异测试用例生成方法。

关键词 变异测试,变异测试用例,测试用例生成,模型检测,时间自动机
中图法分类号 TP311 **文献标识码** A

Approach of Mutation Test Case Generation Based on Model Checking

YANG Hong HONG Mei QU Yuan-yuan
(College of Computer Science, Sichuan University, Chengdu 610065, China)

Abstract In order to carry out mutation analysis of software testing, this paper proposed a method generating mutation test case based on model checking. Using formalized analysis of UPPAAL and test framework, the system under test is firstly described as a timed automata which conforms to its specifications. Then a set of mutation operation following timed automata's basic structure and grammar are injected into the original model to simulate some implementation error which may occur. The mutated models and accessibility property are used as inputs to UPPAAL Yggdrasil to generate test cases covering mutation area. Lastly, test cases are executed on the mutated model and a set of valid test cases is selected on the basis of test execution result (whether the test case can kill the mutation). Experimental results show that the generating test case of the proposed method is valid. Moreover, the mutation score of test case set is higher than that of the existing one based on state machine duplication and model's transition coverage.

Keywords Mutation testing, Mutation test case, Test case generation, Model checking, Timed automata

1 引言

变异测试(变异分析\程序变异)是用于评估现有测试用例质量、发现"无用"测试用例和设计新的软件测试用例的技术。变异分析的一般方法是基于语法结构,按照定义好的规则,系统地变更软件源代码,产生一系列的软件"变异体",基于变异体评估测试用例发现错误的能力。随着大规模计算能力的提升,变异分析有了新的发展,出现了针对面向对象编程语言、非过程化语言(XML,SMV)、有限状态机等的变异分析,以支持新兴的软件测试。目前,基于变异体的测试用例生成方法主要有 3 类:符号执行、基于搜索的方法(SBST)和混合法[1]。

模型检测是自动验证有限状态系统的正确性的技术,对给定的系统模型,穷尽地、自动地检查这个模型是否满足给定的特性,如系统是否死锁、是否会达到导致系统崩溃的状态等。为了用算法解决这样的问题,需要将被验证的系统和需要验证的特性用准确的数学模型表达,如逻辑公式。模型检测以系统模型及系统规约的时序逻辑表达(验证特性)为输入,通过自动穷尽地搜索模型的状态空间,验证特性是否在模型的所有路径上成立,若不成立,则模型检测会给出一条反例(违背特性的路径)。基于模型检测发现的反例,可以自动生成相应的测试用例。

为了支持在基于模型的软件测试中的变异分析和测试,本文研究了基于模型的变异测试方法,并同时考虑使变异分析和测试实现高效和自动化,提出基于模型检测技术的变异测试用例生成方法。通过验证实验,该方法可以有效生成对变异体具有"杀伤力"的测试用例,较其他方法有一定的优势。该方法针对面向对象的软件开发,可以用于类级别的单元测试,也可以用于系统级别的功能测试,具有较广泛的应用前景。

2 相关工作

基于模型的变异测试用例生成方法,根据其变异对象模型可以分为:基于 Petri 网[2]、FSM[3-4]有限状态机、概率有限状态机[5]、Statecharts[6]、SMV[7-8]、UML 状态图[9-11]、LLVM

本文受四川省应用基础研究项目基金(2014JY0112)资助。

杨 红 女,硕士,主要研究方向为软件质量保障与测试;洪 玫 教授,硕士生导师,CCF 高级会员,主要研究方向为软件工程、软件质量保障与测试,E-mail:hongmei@scu.edu.cn;屈媛媛 女,硕士,主要研究方向为软件质量保障与测试。

底层虚拟机[12]、Action System[13-17]、Simulink[18]、HLPSL[19]、Event-B[20]、Circus 进程代数[3],以及时间自动机[21-24]等的变异测试用例生成方法,这些方法在一定程度上支持了基于模型的变异测试,但都有一定局限,且自动化程度低。

Aichernig 等[9,13]提出在原始模型和变异模型中间定义一致性关系,Aichernig 等[14]在此基础上进行改进,以迁移关系编码为约束可满足问题求解,生成抽象的测试用例。结合可达性分析和符号化一致性检测技术[16-17]生成变异测试的思想也应用用时间自动机模型[21]上,支持多种类似 UML 状态图结果的模型[11]。Belli 等[25]在用 Statecharts 描述的模型中,使用了两种基本变异,即插入或省略,并提出了一套完整的基于模型的变异测试生成方案。Black 等[7]以变异后的 SMV 模型为输入,并利用 SMV 模型器产生的反例导出测试用例。Enoiu 等[23]提出了一套基于时间自动机的变异分析及测试生成方案,并将其应用于高铁控制系统。Savary 等[20]也提出了以 Event-B 模型为变异对象,生成适用于鲁棒性测试的变异测试用例。

本文在分析和研究现有方法的基础上,基于模型检测技术,探索变异测试用例生成的自动化和动态适应性问题,以提高基于模型的变异测试的效率和灵活性。

3　方案设计

在本文提出的基于模型检测的变异测试方法中,其输入是被测系统的时间自动机模型,描述被测系统或被测类。首先采用一些专门针对时间自动机模型的变异操作(又称变异算子),模拟实现时的错误,生成相应的变异模型(变异体);然后设计针对变异模型的验证特性,如覆盖变异点的状态可达性;最后采用模型检测工具 UPPAAL Yggdrasil 进行模型检测,并基于模型检测生成的反例,自动生成能够揭示由变异操作符植入到规范模型中的错误的测试用例。图 1 给出了本文提出的基于模型检测的变异测试用例生成方案的流程。

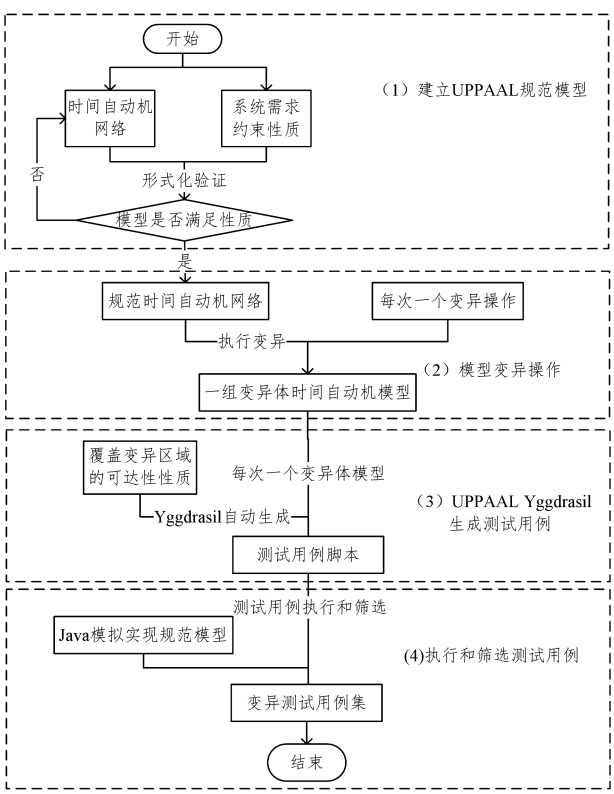

图 1　基于模型检测的变异测试用例生成方案的流程图

4　针对时间自动机的变异操作

输入输出时间自动机(Timed Automata with Input and Output)定义:一个输入输出时间自动机 TAIO 为一个元组:

$$\{Q,\hat{q},\Sigma_I,\Sigma_O,\mathscr{C},I,\Delta\}$$

其中,Q 是一组有限个的节点集合;$\hat{q}\in Q$ 是 TAIO 初始节点;Σ_I 是一个关于输入动作的有限集,Σ_O 是一个关于输出动作的有限集,$\Sigma_I\cap\Sigma_O=\varnothing$ 并且 Σ 是更新动作集合 $\Sigma_I\cup\Sigma_O$;$\mathscr{C}$ 是一组有限的时钟变量集合;I 是一组有限的节点不变量集合,其形如 $c<d$ 或 $c\leqslant d$ 是关于时钟约束的合取范式,其中 $c\in\mathscr{C}$ 并且 $d\in\mathbb{N}$,同时每一个不变量都唯一对应其指定的节点;Δ 是一组有限的形如 (q,a,g,p,q') 的迁移集合,其中,q,$q'\in Q$ 分别代表源头节点和目标节点,$a\in\Sigma$ 代表迁移上的更新动作,g 是一个守卫条件,形如 $c\circ d$ 的合取范式,$\circ\in\{<,\leqslant,=,\geqslant,>\}$并且 $d\in\mathbb{N}$;$p\in C$ 是一组将要被重置的时钟变量。若在一个 TAIO 的 Δ 中,迁移 (q,a,g_1,p_1,q_1) 和 (q,a,g_2,p_2,q_2) 中,$q_1\neq q_2$,$g_1\cap g_2=\varnothing$,则这样的 TAIO 是一个确定性的时间自动机。我们用 $\Delta_O\subseteq\Delta$ 表示被一组输出更新操作标记的迁移集合 $\{\delta=(q,a,g,p,q')\mid\delta\in\Delta,a\in\Sigma_O\}$,而 $\Delta_I=\Delta\backslash\Delta_O$ 则代表被一组输入更新操作所标记的迁移集合。用 $|\mathcal{G}|$ 来表示 TAIO 中出现在所有迁移上的守卫条件的个数,即 $|\mathcal{G}|=\sum_{\delta\in\Delta}|\mathcal{I}|$,其中 $\delta=(q,a,g,p,q')$,并且 g 形如 $\bigwedge_{j\in\mathcal{I}}c_j\circ d_j$;用 $|\mathcal{I}|$ 来表示 TAIO 中出现在所有节点上的不变量的个数。

基于时间自动机模型的变异操作,定义了从规范模型到变异模型的一种映射关系,其形式化定义为:

$$Func(TAIO)\rightarrow 2^{\mathrm{TAIO}}$$

该映射的定义域是一组变异操作符,其中任意一个参数即一个变异操作符 m,能够将一个确定性的 TAIO 映射到集合 $\mu_m(TAIO)$ 上,即一组有限个数的 TAIOs,每一个 $M\in\mu_m(TAIO)$ 代表了一个对规范模型实施变异操作 m 后对应的变异体模型。图 2 给出了一个规范时间自动机模型。

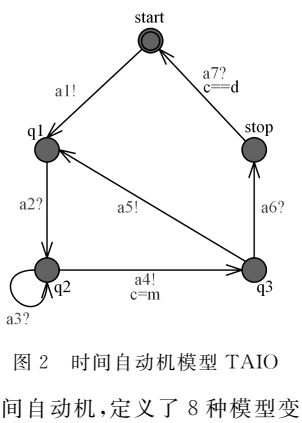

图 2　时间自动机模型 TAIO

基于这一时间自动机,定义了 8 种模型变异,覆盖了模型变异中的基本操作。

4.1　变异操作 1:更改模型迁移上的更新动作标签(Change Action m_{ca})

该变异操作可在模型上生成 $|\Delta_I|(|\Sigma_O|)+|\Delta_O|(|\Sigma_O|-1)$ 个变异体,该变异操作将单个迁移上的、用来标记该迁移的更新动作的标签,更改为另一个不同于该标签的输出标签。该变异操作的形式化定义为:

$$MTAIO\in\mu_{m_{ca}}(TAIO)$$

$MTAIO$ 应该形如 $\{Q,\hat{q},\Sigma_I,\Sigma_O,\mathscr{C},I,(\Delta\backslash\{\delta\}\cup\{\delta_{m_{ca}}\})\}$,

其中 $\delta=(q,a,g,p,q')\in\Delta$；而 $\delta_{m_{ca}}=(q,a_{m_{ca}},g,p,q')$，其中，在变异操作符 m_{ca} 实施前的动作更新标签 $a_{m_{ca}}\in\Sigma_O$ 并且实施后的动作更新标签需要满足 $a_{m_{ca}}\neq a$ 这一约束。

对应的 UPPAAL 模型和变异模型如图 3 所示。

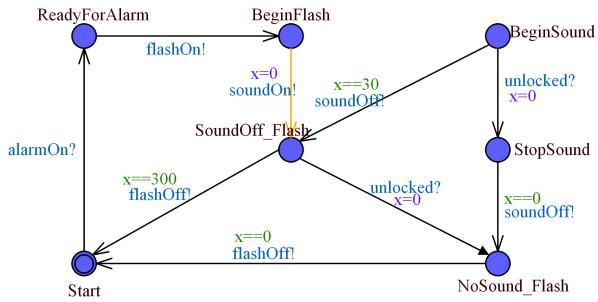

图 3 更改迁移同步更新动作变异操作

需考虑一种特殊情况：若模型中从源头节点到目标节点的迁移上没有更新动作，或更新动作是接收与该模型并行的模型发出的信号时，需要在这个迁移的源头节点和目标节点之间添加一个新节点，同时将该节点设置成 urgent 紧急节点类型，表示系统达到该节点对应的状态时不会停留，而是立即发生下一个迁移，除将原迁移指向的目标节点更改为新添加的节点，还需要在新节点到原目标节点之间添加一个迁移，该迁移上的更新动作为原模型中任意一个同步输出信号。

4.2 变异操作 2：更改模型迁移的目标节点（Change Target m_{ct}）

该变异操作符可在模型 TAIO 上生成 $|\Delta|(|Q|-1)$ 个变异体，该变异操作将单个迁移指向上的目标节点更改为模型中不同于该节点的其他节点。该变异操作的形式化定义为：

$MTAIO\in\mu_{m_{ct}}(TAIO)$

$MTAIO$ 应该形如 $\{Q,\hat{q},\Sigma_I,\Sigma_O,\mathscr{C},I,(\Delta\backslash\{\delta\}\cup\{\delta_{m_{ct}}\})\}$，其中 $\delta=(q,a,g,p,q')\in\Delta$；而 $\delta_{m_{ct}}=(q,a,g,p,q'_{m_{ct}})$，其中在变异操作符 m_{ct} 实施前 $q'_{m_{ct}}\in Q$，并且实施后的迁移指向目标节点需要满足 $q'_{m_{ct}}\neq q'$ 这一约束。对应的 UPPAAL 模型和变异模型如图 4 所示。

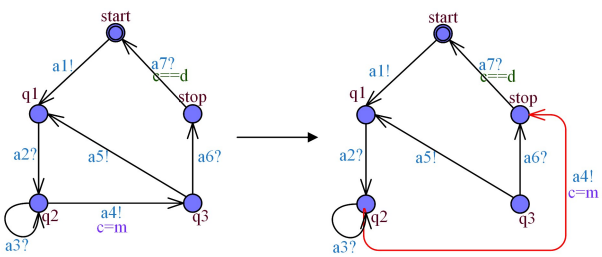

图 4 更改迁移的目标节点变异操作

需考虑一种特殊情况：即迁移的目标节点可能被更改为原迁移的源头节点，这样就会产生一个源头节点和目标节点都是同一个节点的带环迁移。

4.3 变异操作 3：更改模型迁移指向的源头节点（Change Source m_{cs}）

该变异操作可在模型 TAIO 上生成 $|\Delta|(|Q|-1)$ 个变异体，这个变异操作符将单个迁移指向上的源头节点更改为模型中不同于该节点的其他节点。该变异操作的形式化定义为：

$MTAIO\in\mu_{m_{cs}}(TAIO)$

$MTAIO$ 形如 $\{Q,\hat{q},\Sigma_I,\Sigma_O,\mathscr{C},I,(\Delta\backslash\{\delta\}\cup\{\delta_{m_{cs}}\})\}$，其中，$\delta=(q,a,g,p,q')\in\Delta$；而 $\delta_{m_{ct}}=(q_{m_{cs}},a,g,p,q')$，在变异操作符 m_{ct} 实施前，$q_{m_{cs}}\in Q$，并且实施后的迁移指向目标节点，需要满足 $q_{m_{cs}}\neq q$ 的约束。该变异操作的 UPPAAL 模型和变异模型如图 5 所示。

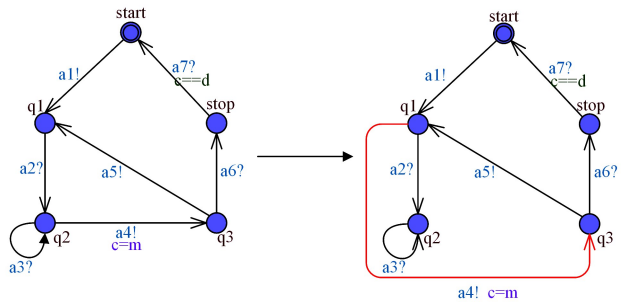

图 5 更改迁移指向的源头节点变异操作

需考虑一种特殊情况：迁移的源节点可能被更改为原迁移的目标节点，可能产生源节点和目标节点为同一个节点的带环迁移。

4.4 变异操作 4：更改模型迁移上守卫条件的连接符（Change Guard m_{cg}）

该变异操作可在模型上生成 $4|\mathcal{G}|$ 个变异体，该变异操作将单个迁移上的守卫条件中的原始等于/不等于条件连接符号，用另一个与该连接符不同的其他连接符代替。该变异操作的形式化定义为：

$MTAIO\in\mu_{m_{cg}}(TAIO)$

$MTAIO$ 形如 $\{Q,\hat{q},\Sigma_I,\Sigma_O,\mathscr{C},I,(\Delta\backslash\{\delta\}\cup\{\delta_{m_{cg}}\})\}$，其中，$\delta=(q,a,g,p,q')\in\Delta$；而 $\delta_{m_{cg}}=(q_{m_{cg}},a,g,p,q')$，$g=\underset{i\in I}{\wedge}c_i\circ_i d_i$，$g_m=\underset{i\in I}{\wedge}c_i\circ_i^m d_i$，在变异操作符 m_{cg} 实施前，$\circ_i\in\{<,\leqslant,=,\geqslant,>\}$，而实施后的迁移上的守卫条件连接符需要满足 $\circ_i^m\in\{<,\leqslant,=,\geqslant,>`\}$ 和 $\circ_i\neq\circ_i^m(i\in I,\circ_j\neq\circ_j^m,i\neq j)$ 约束。该变异操作的 UPPAAL 模型和变异模型如图 6 所示。

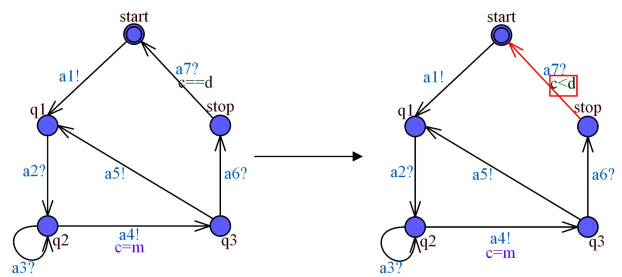

图 6 更改迁移守卫条件连接符变异操作

4.5 变异操作 5：模型迁移上的守卫条件取反（Negate Guard m_{ng}）

该变异操作可在模型上生成 $|\Delta|$ 个变异体，这个变异操作将单个迁移上的原始守卫条件取反。该变异操作的形式化定义为：

$MTAIO\in\mu_{m_{ng}}(TAIO)$

$MTAIO$ 形如 $\{Q,\hat{q},\Sigma_I,\Sigma_O,\mathscr{C},I,(\Delta\backslash\{\delta\}\cup\{\delta_{m_{ng}}\})\}$，其中，$\delta=(q,a,g,p,q')\in\Delta$，$\delta_{m_{ng}}=(q_{m_{ng}},a,\neg g,p,q')$。为了简单起见，即使 $\neg g$ 可能包含析取式，我们也用 $\delta_{m_{ng}}$ 表示一个单独的迁移，迁移上的首位条件 $\neg g$ 可以用 DNF 范式表达，并且每一个守卫条件的析取形式都可以用于单个迁移的守卫条件。该变异操作的 UPPAAL 模型和变异模型如图 7 所示。

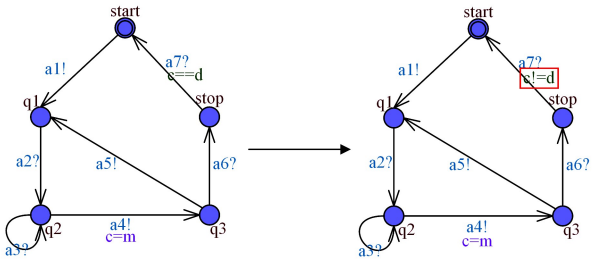

图 7　对迁移守卫条件取反变异操作

4.6　变异操作 6：更改模型中单个节点的不变量约束（Change Invariant m_{ci}）

该变异操作可以在模型上生成 $|\mathscr{I}|$ 个变异体，该变异操作符将单个节点上的原始不变量约束替换成新的不变量约束，而新的不变量约束即是在原始不变量约束条件的右边加 1。该变异操作的形式化定义为：

$MTAIO \in \upsilon_{m_{ci}}(TAIO)$

$MTAIO$ 形如 $\{Q, \hat{q}, \Sigma_I, \Sigma_O, \mathscr{C}, I_{m_{ci}}, \Delta\}$，并且存在一个 $q \in Q$，使得 $I(q) = \bigwedge_{i \in I} c_i \circ d_i$（其中 $\circ \in \{<, \leqslant\}$），对于 $i \in I$，$d_j^m = d_j$，有 $I_m(q) = \bigwedge_{i \in I} c_i \circ d_i^m$，$d_i^m = d_i + 1$（对于所有 $j \neq i$ 成立），并且对于所有 $q' \neq q$，都有 $I(q') = I_m(q')$ 成立。该变异操作的 UPPAAL 模型和变异模型如图 8 所示。

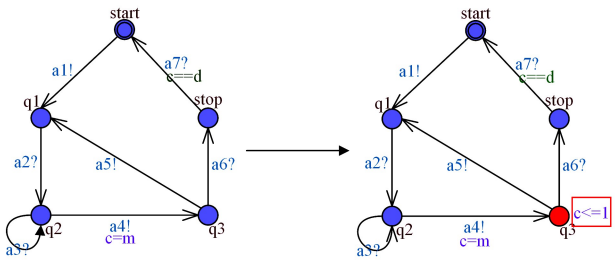

图 8　更改节点不变量约束变异操作

4.7　变异操作 7：新添加的汇聚节点（sink location m_{sl}）

该变异操作可在模型 TAIO 上生成 $|\Delta|$ 个变异体，这个变异操作符将迁移指向的目标节点用新添加的汇聚节点代替，这样的节点是模型中能够接收所有输入但却对该输入无响应的节点。该变异操作的形式化定义为：

$MTAIO \in \mu_{m_{sl}}(TAIO)$

$MTAIO$ 形如 $\{Q \cup \{sink\}, \hat{q}, \Sigma_I, \Sigma_O, \mathscr{C}, I, (\Delta \backslash \{\delta\}) \cup \{\delta_{m_{sl}}\} \cup \Delta_{sink})\}$，其中，指向节点包含汇聚节点的迁移集合 $\Delta_{sink} = \{(sink, a, true, \{\}, sink) \mid a \in \Sigma_I\}$，而迁移 $\delta = (q, a, g, p, q') \in \Delta$，在该变异操作实施后，存在迁移 $\delta_{m_{sl}} = (q, a, g, p_m, q')$。该变异操作的 UPPAAL 模型和变异模型如图 9 所示。

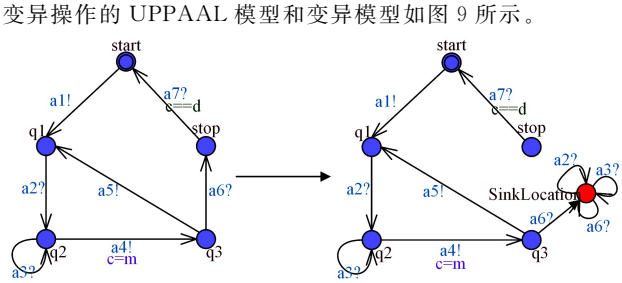

图 9　添加汇聚节点变异操作

4.8　变异操作 8：时钟变量反转重置（invert reset m_{ir}）

该变异操作可在模型 TAIO 上生成 $|\mathscr{C}| |\Delta|$ 个变异体，这个变异操作符将迁移上的更新动作中的原始时钟用另一个快速反转的时钟代替。该变异操作的形式化定义为：

$MTAIO \in \mu_{m_{ir}}(TAIO)$

$MTAIO$ 形如 $\{Q, \hat{q}, \Sigma_I, \Sigma_O, \mathscr{C}, I, (\Delta \backslash \{\delta\}) \cup \{\delta_{m_{ir}}\})\}$，其中，$\delta = (q, a, g, p, q') \in \Delta$，在实施变异操作符后，存在迁移 $\delta_{m_{sl}} = (q, a, g, p_m, q')$，并且对模型中任意的时钟变量 $c \in \mathscr{C}$，当 $c_m \notin p$ 时，有 $p_m = p \cup \{c_m\}$，当 $c_m \in p$ 时，有 $p_m = p \backslash \{c_m\}$。该变异操作的 UPPAAL 模型和变异模型如图 10 所示。

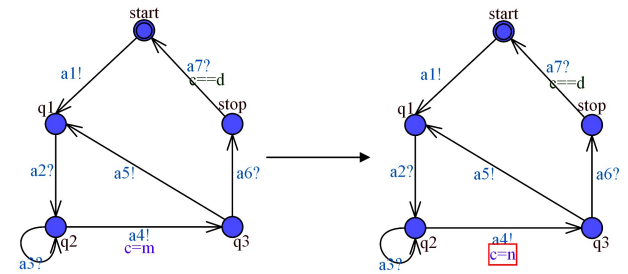

图 10　时钟变量反转重置变异操作

5　案例分析——汽车防盗报警系统变异测试用例生成

以福特汽车报警系统的一个简化版本为例来说明本文方法的实现过程。该系统的需求包括：感应器，当汽车被加锁并且汽车的引擎盖、行李舱以及所有门都被关闭，20 s 后警报系统感应器将激活；发出警报，如果非法打开车门、引擎盖或者行李舱，警报器会发出声音警报 30 s，同时会持续发出灯闪信号 300 s；感应失活，防盗报警系统可以在任意时间失活，例如，警报器正在发出声音警报，只要汽车外按下解锁键，就会使系统失活。

5.1　系统规范的 UPPAAL 模型的建立

使用 UPPAAL 对汽车报警系统建模，使用时间自动机描述 Door，Lock，Arming，Alarm（见图 11）和 User 等几个主要类的状态模型，并验证其是否满足 UPPAAL 模型规范。

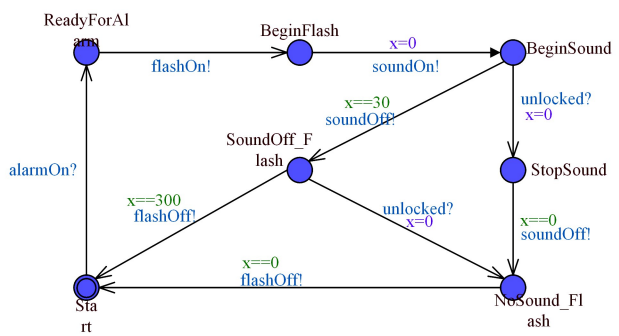

图 11　Alarm（类）的时间自动机模型

验证在 UPPAAL 中对汽车报警系统所建立的模型是否满足系统规范，只有满足系统规范的模型才是规范模型，将系统规范中的约束用 UPPAAL 支持的性质查询语言表达如下：

1）系统不会发生死锁：A[] not deadlock

2）规范约束 R1：$E<>$（$x>=20$）&&（Lock. Locked）imply Arming. Activated

3）规范约束 R2：$E<>$ Arming. Activated imply（Alarm. Start imply Alarm. Start &&$x==300$）

4）规范约束 R3：$A<>$ Lock. ReadyForUnlock imply Arming. Start

在 UPPAAL 验证器中，上述 4 条性质的验证结果显示，我们对汽车防盗警报器系统建立的时间自动机模型满足其规范约束。

5.2 模型的变异操作

本文只考虑适用于时间自动机的一阶变异操作，即最终生成的任意一个变异体的时间自动机模型包含一个特定的变异操作。

例如，对模型进行变异操作 2，更改节点 BeginFlash 到节点 BeginSound 迁移的目标节点至 soundOff_Flash，同时将原迁移条件添加到新迁移上，产生的一个变异体模型如图 12 所示，其他模型变异以此类推。

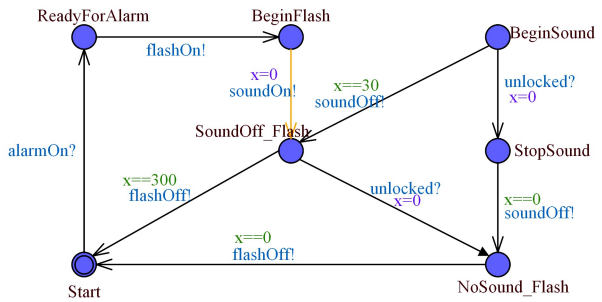

图 12 Alarm(类)模型的变异模型

5.3 基于 UPPAAL Yggdrasil 的测试用例生成

为了用模型检测工具生成测试用例，需要创建针对变异区域的可达性验证性质，UPPAAL Yggdrasil 可以自动搜索模型中满足该可达性的路径并对不满足的路径自动生成测试用例。

比如，针对 5.2 节的更改模型迁移指向的变异操作，其覆盖变异后迁移状态的可达性验证特性描述如下，表示变异模型中存在一条能覆盖 BeginFlash 到 SoundOff_Flash 迁移的路径：

$E<>$BeginFlash imply SoundOff_Flash

将变异模型和验证特性作为输入，通过 UPPAAL Yggdrasil 生成测试用例，测试用例基于模型，将以方法调用序列的方式呈现，如图 13 所示。

```
class Test extends Alarm{
public static void main(String[]args){
expect_BeginFalsh();
soundOn();
setClockZero();
expect_SoundOff_Flash();
}
}
```

图 13 生成的变异测试用例

6 实验验证

本文设计了一个验证实验来验证本方案生成的变异测试用例的测试效率，即对变异体的“杀伤力”。实验对象基于前文的案例——汽车报警系统。

6.1 实验设计

针对该系统的类测试(单元测试)，采用本文提出的方法生成一组变异测试用例；然后采用 Java 程序变异系统 μJava[13-15] 对测试用例进行评估。由于 μJava 基于 Java 源代码和 Juint 测试用例，因此在实验之前，需要用 Java 实现被测类，生成的变异测试用例也需要转换成 Junit 测试用例，作为 μJava 的输入。具体实验步骤如下。

1) 基于系统中类的时间自动机模型，使用模型检测工具 UPPAAL 建立模型；

2)对系统模型进行 8 种变异操作，生成相应的变异模型；

3) 设计模型检测的验证特性；

4)用 UPPAAL Yggdrasil 工具生成测试用例；

5)将生成的测试用例转化成 Junit 的测试用例；

6)用 Java 语言实现被测系统的类；

7)采用 μJava 自动生成被测系统的变异程序；

8)在生成的变异程序上运行转化的 Junit 测试用例；

9)用 μJava 对测试用例进行评估。

考虑到本文提出的模型变异操作与 μJava 的程序变异操作的差异，在 μJava 中的变异操作的选择上，考虑与模型变异等价的变异。

6.2 实验结果与分析

实验对 Door、Lock、Arming 类进行变异测试，实现 8 种变异操作，共生成了 1099 个变异模型。其中，去除 137 个不满足可达性的变异模型，剩下 962 个变异模型。针对每一个变异模型，设置覆盖变异区域的可达性性质，使用 UPPAAL Yggdrasil 工具生成 962 条变异测试用例；再在被测类模型上执行这 962 条测试用例，未通过的测试用例有 628 条。由变异模型生成的测试用例是一种反向的测试用例，即正确的模型是不会通过这些测试用例的。因此，留下这 628 条变异测试用例。

在 μJava 工具中输入汽车防盗警报系统的实现，并选择 μJava 中预置的针对 Java 类中方法的变异操作，将这些操作在系统类的所有方法中实施，最终生成了 72 个变异体程序。通过人工对变异后的代码进行阅读分析，人工去除等价的变异程序，最终保留 38 个变异程序。

用变异分数来评估变异测试用例的质量：

$$\text{变异分数}=\frac{\text{被杀死的变异体个数}}{\text{非等价的变异体个数}}$$

变异操作、变异模型、变异测试用例和评分对应表如表 1 所列。

表 1 变异操作、变异模型、变异测试用例和评分对应表

变异操作	变异模型数/个	测试用例数/个	变异评分/%
m_{ca}	139	139	81
m_{ct}	375	267	94
m_{cs}	375	165	92
m_{cg}	24	6	77
m_{ng}	25	3	87
m_{ci}	11	11	80
m_{sl}	25	25	89
m_{ir}	125	12	67
合计	1099	628	83

表 1 中的数据表明，采用本文方法生成的测试用例有效，平均变异评分为 83%，能在较大程度上“杀死”变异体。但针对不同的变异，其“杀伤力”有所区别。

6.3 与其他方法的对比实验

在前文实验的基础上，将本文提出的方法与 Okun 等[26] 提出的基于状态机复制的变异测试用例生成方法、何洋等[27] 提出的基于模型中变换覆盖的变异测试用例生成方法进行比较。实验基于一个汽车雨刷器系统，分别按照各自方法生成变异测试用例，记录变异测试用例生成过程的总耗时。最后，将变异测试用例转化为 Junit 测试用例。用 μJava 的同一组变异程序计算各方法的测试用例集的变异分数，并进行对比评估。

实验结果如图 14 所示，本文方法采用 UPPAAL 模型，比其他两种方法采用的 NuSMV 模型在建模上更容易；且本文方法的测试用例集的变异分数优于其他两种方法。由于其他两种方法都依赖于构造缺陷性质，在模型检测工具 NuSMV

产生的反例中存在较多的冗余，因此影响了最终测试用例集的质量。

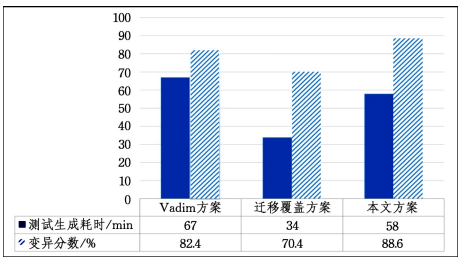

图 14　3 种方法的对比结果

综上所述，本文提出的变异测试用例生成方法是可行且有效的，主要特点是建模方便、自动化程度高、动态适应。同时，该方法基于设计模型，在系统的设计阶段就能生成变异测试用例用于测试系统的设计，这有利于在软件开发中尽早发现软件的缺陷。

结束语　本文提出的基于模型检测的变异测试用例生成方法对被测系统的模型有一定要求，需要满足 UPPAAL 规范，这在一定程度上限制了方法的应用；但该方法还需要在实践中进一步验证其有效性和可靠性。在此基础上，我们将开发工具软件，提升方法的应用价值和自动化程度。

参 考 文 献

[1] 陈翔，顾庆. 变异测试：原理、优化和应用[J]. 计算机科学与探索，2012(12)：1057-1075.

[2] FABBRI S C P F，MASIERO P C，WONG E. Mutation Testing Applied to Validate Specifications Based on Petri Nets[C]// Ifip Tc6 Eighth International Conference on Formal Description Techniques Viii. Chapman & Hall，Ltd. 1995：329-337.

[3] PETRENKO，TIMO A O N，RAMESH S. Multiple Mutation Testing from FSM[C]// International Conference on Formal Techniques for Distributed Objects，Components，and Systems. Springer，2016.

[4] DEVROEY X，PERROUIN G，SCHOBBENS P Y，et al. Poster：VIBeS，Transition System Mutation Made Easy[C]// IEEE/ACM，IEEE International Conference on Software Engineering. IEEE，2015：817-818.

[5] HIERONS R M，MERAYO M G. Mutation testing from probabilistic and stochastic finite state machines[J]. Journal of Systems and Software，2009，82(11)：1804-1818.

[6] FABBRI S C P F，MALDONADO J C，SUGETA T，et al. Mutation Testing Applied to Validate Specifications Based on Statecharts[C]// International Symposium on Software Reliability Engineering. IEEE Computer Society，1999：210.

[7] BLACK P E，OKUN V，YESHA Y. Mutation operators for specifications[C]// The Fifteenth IEEE International Conference on Automated Software Engineering，2000 (ASE 2000). IEEE，2000.

[8] FRASER G，WOTAWA F. Using model-checkers for mutation-based test-case generation，coverage analysis and specification analysis[C]// International Conference on Software Engineering Advances. IEEE，2006.

[9] AICHERNIG B K，BRANDL H，KRENN W. Efficient Mutation Killers in Action[C]// Fourth IEEE International Conference on Software Testing，Verification and Validation. IEEE Computer Society，2011：120-129.

[10] AICHERNIG B K，BRANDL H，KRENN W，et al. Killing strategies for model-based mutation testing[J]. Software Testing Verification & Reliability，2016，25(8)：716-748.

[11] AICHERNIG B，BRANDL H，JOBSTL E，et al. MoMut：UML Model-Based Mutation Testing for UML[C]// IEEE，International Conference on Software Testing，Verification and Validation. IEEE，2015：1-8.

[12] RIENER H，BLOEM R，FEY G. Test Case Generation from Mutants Using Model Checking Techniques[C]// IEEE Fourth International Conference on Software Testing，Verification and Validation Workshops. IEEE，2011：388-397.

[13] AICHERNIG B K，JOBSTL E. Towards symbolic model-based mutation testing：Pitfalls in expressing semantics as constraints [C]// 2012 IEEE Fifth International Conference on Software Testing，Verification and Validation. IEEE，2012.

[14] AICHERNIG B K，JÖBSTL E. Efficient refinement checking for model-based mutation testing[C]// 2012 12th International Conference on Quality Software. IEEE，2012.

[15] AICHERNIG B K，JÖBSTL E. Towards symbolic model-based mutation testing：Combining reachability and refinement checking[J]. arXiv preprint arXiv：1202.6123，2012.

[16] AICHERNIG B K，JÖBSTL E，TIRAN S. Model-based mutation testing via symbolic refinement checking[J]. Science of Computer Programming，2015(97)：383-404.

[17] AICHERNIG B K，TAPPLER M. Symbolic Input-Output Conformance Checking for Model-Based Mutation Testing[J]. Electronic Notes in Theoretical Computer Science，2016，320：3-19.

[18] BINH N T，TUNG K T. A Novel Test Data Generation Approach Based Upon Mutation Testing by Using Artificial Immune System for Simulink Models[C]// Knowledge and Systems Engineering. Springer，2015：169-181.

[19] DADEAU F，HÉAM P，KHEDDAM R，et al. Model-based mutation testing from security protocols in HLPSL[J]. Software Testing，Verification and Reliability，2015，25(5-7)：684-711.

[20] SAVARY A，FRAPPIER M，LEUSCHEL M，et al. Model-based robustness testing in Event-B using mutation[M]// Software Engineering and Formal Methods. Springer，2015：132-147.

[21] AICHERNIG B K，LORBER F，NIČKOVIĆ D. Time for mutants—model-based mutation testing with timed automata[C]// International Conference on Tests and Proofs. Springer，2013.

[22] LI T，LI K，LV J，et al. Mutation Testing for Evaluating the Completeness of Test Cases in High-Speed Train Control System[C]// IEEE，International Conference on Intelligent Transportation Systems. IEEE，2015：777-782.

[23] ENOIU E P，SUNDMARK D，ČAUŠEVIĆ A，et al. Mutation-Based Test Generation for PLC Embedded Software Using Model Checking[M]// Testing Software and Systems. Springer International Publishing，2016：155-171.

[24] LINDSTRÖM B，OFFUTT J，SUNDMARK D，et al. Using mutation to design tests for aspect-oriented models[J]. Information & Software Technology，2017，81(C)：112-130.

[25] BELLI F，HOLLMANN A，HOLLMANN A，et al. Model-based mutation testing-Approach and case studies[J]. Science of Computer Programming，2016，120(C)：25-48.

[26] OKUN V，BLACK P E，YESHA Y. Testing with model checker：Insuring fault visibility[C]// Proceedings of 2002 WSEAS International Conference on System Science，Applied Mathematics & Computer Science，and Power Engineering Systems. 2003.

[27] 何洋，洪玫，祁琳莹，等. 基于模型检测工具 NuSMV 的功能测试用例生成方法[J]. 计算机应用，2015(S2)：155-159.