DOI: 10.19650/j.cnki.cjsi.J1601278

时序约束条件下序列测试建模方法

盛云龙,魏长安,刘玉奇,姜守达
(哈尔滨工业大学自动化测试与控制研究所 哈尔滨 150080)

摘 要:时序约束的描述是测试序列生成前必须面对的问题,但是目前还没有完善的方法能够对时序约束进行描述,为此提出了一种时序约束序列测试建模方法。该建模方法中提出了状态转移允许区间,使得可以对转移发生时前一个状态的连续出现次数进行描述。针对目前欠缺能够有效评价测试序列目标覆盖程度的方法的问题,通过引入核函数来评价测试序列的目标覆盖程度对该问题给予了解决。最后利用提出的建模方法对实例进行建模,验证了建模方法的有效性和可行性。

关键词: 软件测试;序列测试;时序约束;约束描述;目标评价
中图分类号: TH701 **文献标识码**: A **国家标准学科分类代码**: 460.40

Modeling method for sequence testing with temporal constraints

Sheng Yunlong, Wei Chang'an, Liu Yuqi, Jiang Shouda
(*Automatic Test and Control Institute, Harbin Institute of Technology, Harbin 150080, China*)

Abstract: To generate the test sequence, it has to achieve the description of temporal constraints. At present, there is no effective method to realize this objective. Therefore, a modeling method for sequence testing with temporal constraints is proposed in this article. The allowable intervals of state transition are proposed, which can describe the continuous appearing number of a former state when there is a state transition. Another problem of sequence testing is how to evaluate the target coverage degree effectively. This problem is solved by introducing kernel functions to evaluate the target coverage degree of test sequences. Finally, the real cases are modeled based on the proposed method. The availability and feasibility of the method are validated.

Keywords: software test; sequence test; temporal constraints; constraint description; target evaluation

0 引 言

序列测试(sequence testing)旨在覆盖尽可能多的软件执行路径来验证软件的有效性,其测试用例集通常是一组按一定顺序排列的测试用例,又被称为测试序列[1]。测试序列是由测试用例按照时间顺序排列而成的测试方案,可用于某些特定场景或功能的测试,其中测试用例又被称为测试步骤[2-3]。每一条测试用例在测试序列中都对应了一个时间点,这个时间点既可以表示该条测试用例的执行时刻,又可以表示该条测试用例的持续执行时间[4-5]。测试序列既可以用于测试时间连续的待测系统(software under test, SUT),如汽车控制系统[5-6]、视频游戏[7]等;又可以用于测试外部具有输入事件的待测系统,如 GUI(graphical user interface)应用程序等。当测试前者时,测试序列可以表示待测系统中连续变化的内部状态或行为,如测试汽车控制系统时,测试序列可以表示随时间连续变化的方向盘或制动器的位置。当测试后者时,测试序列可以表示待测系统外部连续输入的事件或激励,如测试 GUI 应用程序时,测试序列表示用户界面的连续操作,这时测试序列又被称为事件序列(event sequence)[8-9]。

国内外对序列测试已经进行了多年的研究和探索。如文献[2]和[6]分别介绍了利用分类树模型(classification tree model)为嵌入式系统和嵌入式汽车控制单元生成测试序列的方法,测试人员首先使用分类树对 SUT 的事件、参数或行为建模,然后根据分类树模型设计组合表,即测试序列。组合表中每一行代表一条测试用例,每一条测试用例都对应一个时间点。在测试时,每条测试用例依次按照对应的时间点执行。为了自动生

收稿日期:2016-11-11 Received Date:2016-11-11

成测试序列,Kamal 等[8]将组合交互测试(combinatorial interaction testing,CIT)技术引入到序列测试中,提出了一种 t 维覆盖标准并给出了递推生成算法,但是该算法生成的测试序列冗余较多。Kuhn 等[9]提出了一种基于贪心算法的 t 维事件序列生成方法,弥补了递推算法的不足,同时该算法可以支持事件次序约束。高翔等[10]基于有限状态机提出了利用整数线性规划来求解最短测试序列的方法。Nguyen 等[11]将基于模型的测试方法和组合交互测试进行了有效的结合,首先根据待测系统有限状态机模型生成一条可执行路径;然后采用分类树模型为可执行路径中的状态选择输入。Kruse 等[12]认为测试序列缺少类似组合交互测试的通用的覆盖标准,同时指出对于测试序列中状态或状态间的时序约束还无法有效描述。经过研究,Kruse[2]提出了适用于序列测试的覆盖标准,分别是状态覆盖、状态转移覆盖等,并指出采用线性时序逻辑(linear temporal logic,LTL)和计算树逻辑(computation tree logic,CTL)可以对时序约束进行建模,但是如何建模并没有明确说明。随后,Kruse 等[7]分别基于遗传算法、蚁群算法和贪心算法实现了其提出的状态覆盖和状态转移覆盖测试序列的生成。在时序约束描述方面,Henno[13]在其论文中描述了如何使用 LTL 对状态间的时序约束进行建模和验证的方法,但是使用 LTL 并不能对涉及测试步骤的约束[12]进行描述。目前,基于分类树的测试序列设计工具有 TESTONA[14]和 TESSY[15]。

综上所述,目前对于测试序列的研究正处于起步阶段,测试序列中的时序约束还不能有效的描述。测试序列中的时序约束通常是指测试序列可选状态之间在时间或测试步骤上存在的约束。状态之间在时间上的约束主要是测试序列中的状态转移约束[2,7],例如状态 e 可以转移到状态 e'(状态 e' 可以出现在状态 e 的后面),目前无法对转移发生时前一个状态连续出现的次数进行描述,例如状态 e 连续出现 3 次之后必须转移到状态 e'。对连续出现次数进行限制具有实际意义,如测试汽车控制系统时,需要对加速过程连续出现的次数进行限制。状态之间的另一种约束涉及测试步骤,例如当状态 e 出现在测试序列第 5 步时,第 6 到第 10 步必须出现状态 e'[2,12]。

由于测试序列满足约束是测试序列可以有效测试的前提条件,因此对测试序列的约束进行有效性验证就显得尤为重要,目前还缺少对应的时序约束有效性验证方法。同时,还缺少对测试序列的目标覆盖程度进行有效评价的方法。

钟控计算树逻辑(clocked computation tree logic,CCTL)[16]的提出使得在时间点上描述模型内部条件成为了可能,而且 CCTL 已经在可靠性测试的输入特性描述上取得了一定的成果[17]。这为在时间点上描述模型内部状态关系提供了一个解决方案。随着基于核的模式分析方向的快速发展,模式分析的应用已经覆盖了从生物信息学到文档检索等领域[18],推动了可以用于各种普遍形式的各种算法的发展。因此,把覆盖目标当作需要分析的模式,测试序列的目标覆盖程度评价可以看作是模式分析的一种应用,这样就可以利用现有的模式分析方法对测试序列进行目标评价。

本文提出了一种时序约束条件下的序列测试建模方法,通过提出状态转移允许区间来表示状态转移发生时前一项状态连续出现的次数,扩展了状态转移约束。同时基于核的模式分析方法提出了一种测试序列目标覆盖程度评价方法。最后,利用提出的建模方法对实例进行建模,验证了提出方法的可行性和有效性。

1 序列测试模型

测试序列是一组连续时间下的测试用例集合,每一个测试用例对应了一个时间点,本文只考虑测试用例间是等时间间隔的情况。本节介绍我们提出的序列测试模型。

1.1 序列测试模型定义

时序约束条件下序列测试的测试模型是一个六元组 $M_{ST}(E,R,L,C,t,O_t)$。其中:

E 是测试序列中可选的状态集;

R 是状态转移关系集,表示 SUT 中允许的按顺序排列的状态对;

L 是状态转移允许区间集;

C 是采用 CCTL 公式描述的时序约束集,详情参见 1.2 节约束描述;

t 表示测试序列的覆盖维度,O_t 表示对应覆盖维度下的覆盖目标集,详情参见 1.3 节覆盖标准。

对于每一个需要测试的状态 e,始终存在 $e \in E$。对 E 中任意两个不同状态 e 和 e',如果其构成的连续的状态为被系统所允许则有 $(e,e') \in R$。对于状态转移允许区间集 L,有 $L:R \to N_0 \times N_0$, $\forall (e,e') \in R$, $\exists L(e,e')=[a,b]$ $(0 \leqslant a \leqslant b < +\infty)$,对于实际系统,状态出现次数上限 $b < +\infty$,即状态 e 转移到状态 e' 前允许状态 e 连续出现的次数,在出现 a 次前不允许转移到状态 e',在出现 b 次后禁止状态 e 再出现。对于区间中的任何一个次数,状态转移都可能发生。转移允许区间实际上是一种约束,限制状态连续出现的次数,区间取值由实际情况决定。

如果 $\exists (e,e') \in R$,那么一定有 $e,e' \in E$;如果 $\exists L(e,e')=[a,b]$,那么一定有 $(e,e') \in R$。为了更充分地描述测试序列的时序关系,我们还提出了相关的一些概念:

状态最多出现次数 $step_{max}(e)$ 用来表示状态 e 在测试

序列中能够连续出现的最多次数,其可表示为:

$$step_{\max}(e)=\max\{c \mid L(e,e')=[b,c],\forall(e,e')\in R\} \tag{1}$$

状态次数对集 G 用来表示状态 e 在测试序列中存在的标准形式,其可表示为:

$$G=\{(e,c) \mid (e,c)\in E\times \mathrm{N}_0 \wedge 0\leqslant c\leqslant step_{\max}(e)\} \tag{2}$$

式中:c 表示状态 e 在序列中连续出现的次数;$\forall g\in G,g$ 表示一组状态次数对。

次数值特征函数 Λ_G 表示状态 e 的次数值集合,具体形式为:

$$\Lambda_G(e)=\{c\in \mathrm{N}_0 \mid (e,c)\in G\}=\{c \mid 0\leqslant c\leqslant step_{\max}(e)\} \tag{3}$$

次数值特征函数 Λ_G 的扩展形式 $\Lambda_L(e,e')$ 表示转移(e,e')发生时状态 e 连续出现的次数,其定义为:

$$\Lambda_L(e,e')=\{c\in \mathrm{N}_0 \mid L(e,e')=[a,b]\wedge a\leqslant c\leqslant b\} \tag{4}$$

给定一个序列测试模型 $\mathrm{M}_{ST}(E,R,L,C,t,O_t)$,如果存在一组测试序列 $S=\{g_0,g_1,\cdots,g_{|S|-1}\}$,其中 $g_i=(e_i,c_i)\in G(0\leqslant i\leqslant(|S|-1))$,且测试序列 S 同时满足约束集 C 和目标集 O_t,那么称测试序列 S 是模型 $\mathrm{M}_{ST}(E,R,L,C,t,O_t)$ 下的一组实例,G 是模型全部实例中取值的全集。其中 g_i 表示第 i 个测试用例中的状态次数值对,如用符号 $step_{in}$ 表示测试步骤,那么 g_i 对应的测试步骤 i 可表示为 $step_{in}(g_i)=i$。

在测试序列中 $\forall g_{i+1}\in G$ 有 $g_{i+1}=(e_i,c_i+1)(c_i+1\leqslant step_{\max}(e_i))$,或者 $g_{i+1}=(e_{i+1},0)$,对于 $L(e_i,e_{i+1})=[a,b]$ 且 $a\leqslant c_i\leqslant b$。

下面举例说明序列测试模型,图 1 所示是一个 SUT 序列测试模型对应的状态转移图,其状态集为 $E=\{q,p,r\}$,其状态转移关系集为 R = {(p,r),(p,q)(r,p)(r,q)(q,r)}。状态 r 的次数值特征函数为 $\Lambda_G(r)=\{1,2,3,4\}$,状态转移(r,q)可以发生的次数值集合为 $\Lambda_L(r,q)=\{2,3,4\}$,因此状态 r 连续出现的最多次数 $step_{\max}(r)=4$。状态转移(q,p)是 SUT 的一个约束,不能发生。该模型中的一组测试序列可以表示为 $S=\{(p,0)_0,(p,1)_1,(p,2)_2,(q,0)_3,(q,1)_4,\cdots\}$。其中每一个状态次数值对都属于集合 G,如$(p,1)\in G$。状态次数值对的下角标表示了该状态次数值对的测试步骤,如 $step_{in}((p,2)_2)=2$。

1.2　约束描述

时序约束条件下测试序列需要满足的转移约束和步骤约束统一采用 CCTL 公式进行描述。每一个 CCTL 公式描述的时序约束都是约束集 C 中的元素。序列测试模型 $\mathrm{M}_{ST}(E,R,L,C,t,O_t)$ 的实例 S 需要满足全部约束。下面简要介绍 CCTL 的公式规则。$\forall m,n\in \mathrm{N}_0$,且 $m\leqslant n$,

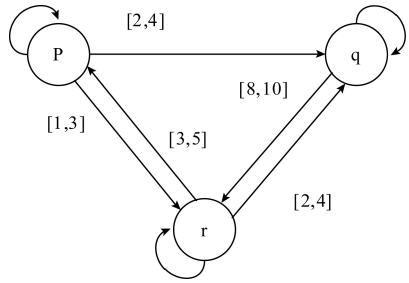

图 1　状态转移图

Fig.1　Diagram of state transition

其公式规则定义如下[19-20]:

$$\begin{aligned}&\phi=AP \mid \neg\phi \mid \phi\wedge\phi \mid \phi\rightarrow\phi \mid \phi\oplus\phi\\&\mid EX_{[n]}\phi \mid EF_{[m,n]}\phi \mid EG_{[m,n]}\phi \mid E(\phi U_{[m,n]}\phi)\\&\mid AX_{[n]}\phi \mid AF_{[m,n]}\phi \mid AG_{[m,n]}\phi \mid A(\phi U_{[m,n]}\phi)\end{aligned} \tag{5}$$

式中:AP 是原子表达式;ϕ 是 CCTL 公式;¬、∧、→和⊕是经典逻辑算子;X、F、U 和 G 是时态算子,其中 X 是下一刻(next)算子,F 是最终(finally)算子,G 是总是(always)算子,U 是直到(until)算子,$[m,n]$ 是时态算子的作用域,表示步骤约束区间;A 和 E 是路径量词,E 表示存在(Exist)一条路径,A 表示对全部(All)路径。

在 CCTL 公式中,路径(trace)表示一组测试序列组成的元组,记为 ρ,$\rho\mid=\phi$ 表示在路径 ρ 上满足 ϕ。$EX_{[n]}\phi$ 表示存在步骤 n 满足 ϕ;$EF_{[m,n]}\phi$ 表示存在步骤区间$[m,n]$最终满足 ϕ;$EG_{[m,n]}\phi$ 表示当前步骤到区间$[m,n]$中存在的一个步骤间始终满足 ϕ;$E(\phi U_{[m,n]}\psi)$ 表示从当前步骤开始直到区间$[m,n]$中的某一步骤满足 ψ 之前一直满足 ϕ。这些公式中的 E 都可以替换成 A 表示对全部路径。由于一个测试序列对应了一条路径,因此符号 A 表示的对全部路径和符号 E 表示的存在一条路径具有相同的含义。

本文采用 CCTL 公式对序列测试问题中的时序约束进行描述。

1)转移约束描述

对于模型 $\mathrm{M}_{ST}(E,R,L,C,t,O_t)$,其中每个测试步骤下对应的状态次数值对都需要满足公式 $EG((e_i,c_i)\in G)$,该公式表示了测试序列中的状态次数值对只能来自于集合 G 中,G 是全集。

状态转移约束可以描述成以下 3 种形式:

(1) $\forall(e,e')\notin R$,

$$EG(e\rightarrow X(\neg e')) \tag{6}$$

式(6)表示在忽略状态对应的次数时,对于不属于状态转移集 R 中的状态转移,一定不能出现在测试序列中,此约束还被称为状态次序约束[8];

(2)已知状态 e 和其可以转移的状态集合 $E'=\{e' \mid (e,e')\in R\}$,$\forall e'\in E'$,如果 $L(e,e')=[a,b]$,

$$EG((((e,c) \wedge (c < a)) \rightarrow X(\neg e')) \wedge (((e,c) \wedge (a \leqslant c < b)) \rightarrow X((e',0) \wedge (e,c+1))) \wedge (((e,c) \wedge (c = b)) \rightarrow X(e',0))) \tag{7}$$

式(7)表示对任意的状态 e 可以转移到的状态 e',当状态 e 对应的次数值 c 小于转移允许的次数 a 时,转移一定不能发生,当 $c \in [a,b)$ 转移可以发生,当 $c = b$ 时转移一定发生;

(3)已知状态 e 和其可以转移的状态集合 $E' = \{e' \mid (e,e') \in R\}$, $\forall e' \in E'$,如果 $L(e,e') = [a,b]$,

$$EG((e,0) \rightarrow eU_{[a,b]}(e',0)) \tag{8}$$

公式(8)表示当 $(e,0)$ 发生时,状态 e 转移到任意其可以转移的状态 e' 前的出现次数一定属于区间 $L(e,e') = [a,b]$。

上述 3 种描述方式中式(6)和(7)是采用 LTL 和 CTL 描述的,由于 LTL 和 CTL 相比于 CCTL 没有引入步骤约束区间,如式(6)所示,并不能对出现在测试序列中的转移步骤进行描述;如式(7)所示,虽然可以通过合理构造描述转移约束,但是会导致描述较为复杂繁琐。CCTL 的优势正是在于引入了步骤约束区间,由此将时态算子 U 和步骤约束区间相结合,如式(8)所示,就可以简明的对转移约束进行描述。

2)步骤约束描述

对于测试序列中的状态次数值对 $g = (e,c) \in G$,ϕ 和 ψ 是 CCTL 公式,我们将 $g,S \models \phi$ 简记为 $g \models \phi$,表示以 g 开始的测试序列 S 满足公式 ϕ。S 是一个测试序列实例。步骤约束有以下 4 种基本形式分别对应了 4 个时态算子,其中 n 和 m 表示测试序列的步骤 $m,n \in \mathrm{N}_0$,且 $m \leqslant n$。

(1)对于 X 算子,其表示步骤约束的基本形式为:

$$g \models EX_{[n]}\phi \tag{9}$$

式(9)含义是存在序列 $S = \{g_0,g_1,\cdots,g_i,\cdots\}$,$g = g_0$,有 $g_n \models \phi$。表示在第 n 个步骤 $g_n = (e_n,c_n)$ 上,满足 ϕ,ϕ 表示为状态或状态次数对,即 $\phi \in E$ 或 $\phi \in G$;

(2)对于 G 算子,其表示步骤约束的基本形式为:

$$g = EG_{[m,n]}\phi \tag{10}$$

式(10)含义是存在序列 $S = \{g_0,g_1,\cdots,g_i,\cdots\}$,$g = g_0$,$\exists m \leqslant i \leqslant n$, $\forall 0 \leqslant j \leqslant i$ 有 $g_j \models \phi$;

(3)对于 U 算子,其表示步骤约束的基本形式为:

$$g \models E(\phi U_{[m,n]}\psi) \tag{11}$$

式(11)表示存在序列 $S = \{g_0,g_1,\cdots,g_i,\cdots\}$,$g = g_0$,$\exists m \leqslant i \leqslant n$,有 $g_i \models \psi$,且 $\forall 0 \leqslant j < i$ 有 $g_j \models \phi$。

(4)对于 F 算子,其表示步骤约束的基本形式为:

$$g \models EF_{[m,n]}\phi \tag{12}$$

式(12)表示存在序列 $S = \{g_0,g_1,\cdots,g_i,\cdots\}$,$g = g_0$,$\exists m \leqslant i \leqslant n$,有 $g_i \models \phi$。

1.3 覆盖标准

不同的测试覆盖标准下的测试序列的规模不同,一般测试序列规模越大测试越充分,但是测试开销也大,我们提出以下 3 种测试充分性不同的覆盖标准:

1)状态覆盖:$t = 1$,要求每个需要覆盖的状态在测试序列中至少出现一次,满足状态覆盖的测试序列相比其他覆盖标准具有较少的测试用例,其对应的目标集为 $O_1 \subset G$。

2)状态转移覆盖:$t = 2$,要求每个需要的状态转移至少被覆盖一次,转移的出现必须在指定的转移允许区间范围之内。在状态转移覆盖中,还需要进一步指明转移发生时前一个状态出现的次数 c, $\forall \{(e,c)(e',0)\} \in O_2$, $\exists c \in L(e,e') = [a,b]$。状态转移覆盖中的同一组状态转移可以有多个允许转移发生的次数 c,如 $\exists c_1,c_2 \in L(e,e')$,有 $\{(e,c_1)(e',0)\},\{(e,c_2)(e',0)\} \in O_2$。

3)状态转移对覆盖:$t = 3$,要求每个允许的状态转移对至少被覆盖一次,转移的出现必须在指定的转移区间范围之内。如有 $\forall \{(e,c)(e',0),\cdots,(e',c')(e'',0)\} \in O_3$, $\exists c \in L(e,e'),c' \in L(e',e'')$。

2 测试序列约束验证和目标覆盖程度评价

对于生成的测试序列,最重要的一点是对其进行约束验证和目标评价。要保证测试序列既要满足约束集 C 又要覆盖目标集 O_t。

2.1 约束验证

对应于约束描述,约束验证也分为对转移约束的验证和对步骤约束的验证。

1)转移约束验证

对于测试序列 $S = \{g_0,g_1,\cdots,g_i,\cdots\}$ 中任意 2 个相邻的状态次数对 $g_i = (e_i,c_i)$ 和 $g_{i+1} = (e_{i+1},c_{i+1})$,首先保证 $g_i,g_{i+1} \in G$;其次如果有 $e_{i+1} = e_i$,那么必须有 $(c_i < time_{\max}(e_i)) \wedge (c_{i+1} = c_i + 1) \wedge (c_{i+1} \leqslant time_{\max}(e_i))$,如果 $e_{i+1} \neq e_i$,那么必须有 $(e_i,e_{i+1}) \in R$,$c_i \in L(e_i,e_{i+1}) = [a,b]$ 且 $c_{i+1} = 0$。如果不满足上述两点,则称测试序列违反了转移约束。

2)步骤约束验证

对于步骤约束,其验证可以分为以下 4 种情况:

(1)对形如 $g \models EX_{[n]}\phi$ 的约束,直接验证第 n 个测试步骤是否满足 ϕ;

(2)对形如 $g \models EG_{[m,n]}\phi$ 的约束,可以通过如下递归公式进行验证:

$$EG_{[m,n]}\phi = \phi \wedge EXEG_{[m-1,n-1]}\phi \tag{13}$$

首先当前步骤下公式 ϕ 需要成立,然后将下一个步骤作为当前步骤,并要保证公式 $EG_{[m-1,n-1]}\phi$ 成立,以此

逐步递归完成验证;

(3)对形如 $g = E(\phi U_{[m,n]}\psi)$ 的约束,首先通过递归公式:

$$E[\phi U_{[m,n]}\psi] = \phi \wedge EXE[\phi U_{[m-1,n-1]}\psi] \tag{14}$$

将 $E[\phi U_{[m,n]}\psi]$ 转化成 $E[\phi U_{[0,n-m]}\psi]$,再通过递归公式:

$$E[\phi U_{[0,n-m]}\psi] = \psi \vee (\phi \wedge EXE[\phi U_{[0,n-m-1]}\psi]) \tag{15}$$

进行验证,直至 $E[\phi U_{[0,0]}\psi] = \psi$,完成验证;

(4)对形如 $g = EF_{[m,n]}\phi$ 的约束,直接验证从第 m 个步骤到第 n 个步骤中,是否至少存在一个步骤满足 ϕ。

对于由多个基本步骤约束组合而成的复杂步骤约束,需要从左到右依次验证每个基本步骤约束是否满足。如果测试序列不满足上述任意一个种步骤约束,都称测试序列违反了步骤约束。

2.2　目标覆盖程度评价

本文采用基于核函数的目标覆盖程度评价方法,评价测试序列覆盖目标的数目,数目越大说明测试序列的覆盖率越高。

定义映射 $\theta_v: S \to \theta_v(S) \in \{0,1\}$,用于评价测试序列 S 是否覆盖了目标集中的状态。

对于状态覆盖,当 $(e,c) \in G$ 时,如果 $(e,c) \in O_1$ 则有公式:

$$\theta_e(S) = \begin{cases} 1 & \{(e,c)\} \subset S \\ 0 & \{(e,c)\} \not\subset S \end{cases} \tag{16}$$

式(16)表示当测试序列 S 包含由状态 e 和次数值 c 构成的状态次数对 (e,c) 时,映射 $\theta_e(S) = 1$,否则 $\theta_e(S) = 0$。与其等价的另一个形式是当 $g = (e,c) \in G$ 时,如果 $g \in O_1$ 则有公式:

$$\theta_g(S) = \begin{cases} 1 & \{g\} \subset S \\ 0 & \{g\} \not\subset S \end{cases} \tag{17}$$

对于状态转移覆盖,当 $(e,c) \in G, (e,e') \in R$ 且 $c \in L(e,e') = [a,b]$ 时,如果 $\{(e,c)(e',0)\} \in O_2$ 那么有公式:

$$\theta_{e\to e'}(S) = \begin{cases} 1 & \{(e,c),(e',0)\} \subset S \\ 0 & \{(e,c),(e',0)\} \not\subset S \end{cases} \tag{18}$$

式(18)表示状态 e 到 e' 的转移是覆盖目标时,如果 S 中存在子序列为该状态转移则 $\theta_{e\to e'}(S) = 1$,否则 $\theta_{e\to e'}(S) = 0$。其还有另一个等价形式,当 $g = (e,c) \in G, g' = (e',0) \in G, (e,e') \in R$ 时,且 $c \in L(e,e') = [a,b]$,等价形式为:

$$\theta_{g\to g'}(S) = \begin{cases} 1 & \{g,g'\} \subset S \\ 0 & \{g,g'\} \not\subset S \end{cases} \tag{19}$$

对于状态转移对覆盖,当 $c \in L(e,e'), c' \in L(e',e'')$ 时,如果有子序列 $o = \{(e,c),(e',0),(e',1),\cdots,(e',c'-1),(e',c')(e'',0)\} \in O_3$,且 $\forall (e,c) \in o, \exists (e,c) \in G$,则有公式:

$$\theta^{e\to e'}_{e'\to e''}(S) = \begin{cases} 1 & o \subset S \\ 0 & o \not\subset S \end{cases} \tag{20}$$

式(20)表示状态 e 到 e' 再到 e'' 的状态转移对是覆盖目标时,如果 S 中存在一个子序列恰好覆盖该目标状态转移对,那么有 $\theta^{e\to e'}_{e'\to e''}(S) = 1$,否则 $\theta^{e\to e'}_{e'\to e''}(S) = 0$。其还有另一种等价形式,如果有测试序列 $o = \{g_0, g_1, g_2, \cdots, g_{i-1}, g_i, g_{i+1}\} \in O_3$,其中 $g_0 = (e,c), g_1 = (e',0), g_i = (e',c'), g_{i+1} = (e'',0) \in G$,当 $(e,e'), (e',e'') \in R$,且有 $c \in L(e,e'), c' \in L(e',e'')$,有公式:

$$\theta^{g\to g'}_{g'\to g''}(S) = \begin{cases} 1 & o \subset S \\ 0 & o \not\subset S \end{cases} \tag{21}$$

映射 θ_v 对应的核函数为:

$$\kappa_v(S,S) = <\theta_v(S), \theta_v(S)> = \sum_{o\in O} (\theta_v(S))^2 \tag{22}$$

通过计算测试序列在对应覆盖标准下的核函数值,可以评价测试序列对目标的覆盖情况。

1)状态覆盖核函数

对于状态覆盖,其核函数的计算公式如下:

$$\kappa_e(S,S) = <\theta_e(S), \theta_e(S)> = \sum_{(e,c)\in O_1} (\theta_e(S))^2 \tag{23}$$

式(23)表示测试序列 S 中包含目标覆盖集 O_1 中状态的个数 $0 \leqslant \kappa_e(S,S) \leqslant |O_1|$,如果有 $\kappa_e(S,S) = |O_1|$,则表明测试序列 S 覆盖了目标集 O_1 中的全部状态。

2)状态转移覆盖核函数

对于状态转移覆盖,其核函数的计算公式如下:

$$\kappa_{e\to e'}(S,S) = <\theta_{e\to e'}(S), \theta_{e\to e'}(S)> = \sum_{\{(e,c),(e',0)\}\in O_2} (\theta_{e\to e'}(S))^2 \tag{24}$$

式(24)表示测试序列 S 中包含目标覆盖集 O_2 中目标的个数 $0 \leqslant \kappa_{e\to e'}(S,S) \leqslant |O_2|$,如果有 $\kappa_{e\to e'}(S,S) = |O_2|$,则表明测试序列 S 覆盖了目标集 O_2 中的全部状态转移。

3)状态转移对覆盖核函数

对于状态转移对覆盖,其核函数的计算公式如下:

$$\kappa^{e\to e'}_{e'\to e''}(S,S) = <\theta^{e\to e'}_{e'\to e''}(S), \theta^{e\to e'}_{e'\to e''}(S)> = \sum_{o\in O_3} (\theta^{e\to e'}_{e'\to e''}(S))^2 \tag{25}$$

式(25)表示测试序列 S 中包含目标覆盖集 O_3 中目标的个数,其中 $o = \{(e,c)(e',0),(e',1),\cdots,(e',c'-1),(e',c')(e'',0)\}$,有 $0 \leqslant \kappa^{e\to e'}_{e'\to e''}(S,S) \leqslant |O_3|$,如果 $\kappa^{e\to e'}_{e'\to e''}(S,S) = |O_3|$,则表明测试序列 S 覆盖了目标集 O_3 中全部的状态转移对。

对于两个不同的测试序列 S 和 S',同样可以通过计算核函数值得到它们之间包含相同状态的个数,公式如下:

$$\kappa_e(S,S') = < \theta_e(S), \theta_e(S') > = \sum_{(e,c)\in O_1} \theta_e(S)\theta_e(S') \tag{26}$$

式(26)表示测试序列 S 和 S' 中覆盖目标集 O_1 中的相同状态的个数。

3 实例分析

本节将首先对一个简单的汽车速度控制系统实例进行分析,最后再针对 3 个复杂实例进行分析。

3.1 简单实例分析

汽车速度控制系统在时序约束条件下的序列测试模型的状态转移图如图 2 所示,该实例来自于序列测试建模工具 TESTONA。

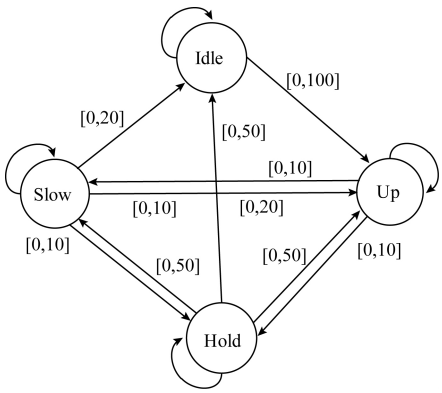

图 2 速度控制系统状态转移图

Fig.2 Diagram of state transition of the speed control system

如图 2 所示,速度控制系统将汽车速度状态划分为 4 类,分别是:空闲(idle speed,I)、加速(speed up,U)、匀速(hold speed,H)和减速(slow speed,S)。状态允许转移区间来自于对 SUT 的实际测试场景分析,本文根据实际情况选择区间[0,100]表示状态转移发生时前一个状态连续出现次数近乎不受限制,实际的汽车从静止到加速启动正与此相符,选择区间[0,10]表示状态转移的发生需要受到严格的限制,如加速过程不能一直持续下去,必须在有限的次数下进入到匀速或减速状态。因此对该系统有:

$E = \{I,S,U,H\}$

$R = \{(I,U),(U,H),(U,S),(H,I),(H,U),(H,S),(S,I),(S,U),(S,H)\}$

状态转移允许区间详见图 2 所示,这里不再详细描述,如 $L(I,U) = [0,100]$。SUT 存在 3 个转移约束,分别是 $EG(U\rightarrow X(\neg I))$ 和 $EG(I\rightarrow X((\neg H)\wedge(\neg S)))$,还存在两个步骤约束,$g| = EX_{[0]}I$ 和 $g| = EX_{[|S|-1]}I$,表示测试序列的起始和终止状态必须是空闲状态。

下面分别给出被测系统的状态覆盖序列 S_1、状态转移覆盖序列 S_2 和状态转移对覆盖序列 S_3。

$S_1 = \{(I,0),(U,0),(U,1),(H,0),(H,1),(S,0),(S,1),(I,0)\}$

$S_2 = \{(I,0),(U,0),(S,0),(I,0),(U,0),(U,1),(H,0),(H,1),(S,0),(U,0),(H,0),(H,1),(U,0),(S,0),(H,0),(I,0)\}$

$S_3 = \{(I,0),(U,0),(S,0),(I,0),(U,0),(U,1),(H,0),(H,1),(S,0),(U,0),(H,0),(H,1),(U,0),(S,0),(H,0),(I,0),(U,0),(H,0),(H,1),(I,0),(U,0),(S,0),(U,0),(S,0),(H,0),(U,0),(H,0),(S,0),(H,0),(S,0),(S,1),(I,0)\}$

上述测试序列 S_1、S_2 和 S_3 满足全部的状态转移约束,同时满足步骤约束 $g| = EX_{[0]}I$ 和 $g| = EX_{[|S|-1]}I$。

对于序列 S_1 有 $\kappa_e(S_1,S_1) = |E| = 4$,$S_1$ 覆盖了状态集 E 中的全部状态,并且每一个状态都覆盖了 2 次。对于序列 S_2 有 $\kappa_{e\rightarrow e'}(S_2,S_2) = 9$,$S_2$ 覆盖了全部 9 组状态转移,对状态转移集 R 中的全部状态转移至少覆盖了 1 次,其中状态转移(U,H)、(I,U) 和(U,S) 被覆盖了 2 次,覆盖(U,H) 的分别是$\{(U,1),(H,0)\}$ 和$\{(U,0),(H,0)\}$;覆盖(I,U) 的是$\{(I,0),(U,0)\}$;覆盖(U,S) 的是$\{(U,0),(S,0)\}$。对于序列 S_3 有 $\kappa_{\substack{e\rightarrow e'\\ e'\rightarrow e''}}(S_3,S_3) = 20$,$S_3$ 覆盖了状态转移集 R 中任意 2 个状态转移组合成的可行的状态转移对,其中对于转移对(I,U,H)、(I,U,S)、(H,I,U)、(U,S,H) 和(U,H,S) 都覆盖了 2 次。由于 3 组序列都覆盖了要求的目标,因此 S_1、S_2 和 S_3 可以作为有效的测试用例对 SUT 进行测试。

3.2 复杂实例分析

本文从文献[2]和[7]中选出了 3 个具有较多状态的复杂实例,如表 1 所示,表 1 中分别给出了 3 个 SUT 的覆盖目标数目及步骤约束,每个 SUT 状态间构成的状态对除了允许的状态转移之外全部是转移约束,这里就不详细说明了,3 个 SUT 的步骤约束只有初始步骤约束 $EX_{[0]}$,具体约束状态也不再详细说明。

表 1 SUT 的覆盖目标数目及步骤约束

Table 1 The number of objectives and step constraints in SUT

SUT	覆盖目标数目			步骤约束
	状态数	转移数	转移对数	
Autoradio 的控制单元	11	23	66	$EX_{[0]}$
Coffee Machine	9	11	15	$EX_{[0]}$
Tetris	10	17	27	$EX_{[0]}$

表 2 所示是采用 TESTONA 工具对上述 3 个 SUT 设计生成的满足约束条件的测试序列的规模,以 Autoradio 的控制单元为例,Autoradio 的控制单元有 11 个状态,其

中状态覆盖序列规模为 18,共冗余了 7 个状态;转移覆盖序列规模为 27,覆盖了全部需要覆盖的 23 个转移,冗余了 3 个转移;转移对覆盖序列规模为 120,覆盖了全部需要覆盖的 66 个转移对,冗余了 52 个转移对。由于测试序列的生成利用了现有的 TESTONA 工具,很难做到最优,并且由于 SUT 本身转移关系的限制,冗余多由状态转移图的路径决定,有时如果要覆盖新的未覆盖的转移就只能对已覆盖过的转移再次覆盖,致使产生冗余。

表 2　SUT 测试序列规模

Table 2　The sizes of test sequences in SUT

SUT	状态覆盖	转移覆盖	转移对覆盖
Autoradio 的控制单元	18	27	120
Coffee Machine	9	14	27
Tetris	12	30	46

经过上述实例分析,可以验证本文提出的时序约束条件下的序列测试建模方法是可行的和有效的。

4　结　　论

目前对时序约束条件下的序列测试研究还处于起步阶段。本文提出了一种时序约束条件下的序列测试建模方法,在该方法中引入了状态转移允许区间的概念,扩展了转移约束,同时进一步引入 CCTL 公式对状态间的时序约束进行描述。与此同时本文提出了基于测试序列核函数的测试序列覆盖目标评价方法,通过计算核函数值可以评价测试序列的目标覆盖程度。最后通过实例分析验证了提出方法的有效性和可行性。

在本文研究成果的基础上,下一步将研究不同覆盖标准下的时序约束测试序列自动生成算法,在算法中利用约束验证方法对测试序列进行约束验证,利用测试序列核函数判断测试序列是否完成目标覆盖。

参考文献

[1] OFFUTT J, LIU S, ABDURAZIK A, et al. Generating test data from state-based specifications[J]. Software Testing, Verification and Reliability, 2003, 13(1): 25-53.

[2] KRUSE P M. Enhanced test case generation with the classification tree method[D]. Berlin: Freie Universität Berlin, 2014.

[3] 刘珊珊, 吕超.改进信息熵算法的最优测试序列生成方法[J]. 电子测量技术, 2013, 36(12):28-31.
LIU S S, LV CH. Approach of optimal diagnosis test sequence based on improved information entropy[J]. Electronic Measurement Technology, 2013, 36(12): 28-31.

[4] BüCHNER F. Test case design using the classification tree method[J]. Atzelektronik Worldwide, 2007, 2(1): 15-17.

[5] CONRAD M, KRUPP A. An extension of the classification-tree method for embedded systems for the description of events[J]. Electronic Notes in Theoretical Computer Science, 2006, 164(4):3-11.

[6] CONRAD M, Fey I, SADEGHIPOUR S. Systematic model-based testing of embedded automotive software[J]. Electronic Notes in Theoretical Computer Science, 2005(111):13-26.

[7] FERRER J, KRUSE P M, CHICANO F, et al. Search based algorithms for test sequence generation in functional testing[J]. Information & Software Technology, 2015(58):419-432.

[8] ZAMLI K Z, OTHMAN R R, ZABIL M H M. On sequence based interaction testing[C]. Computers and Informatics, 2011:662-667.

[9] KUHN D R, HIGDON J M, LAWRENCE J F, et al. Combinatorial methods for event sequence testing[C]. 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012:601-609.

[10] 高翔, 王欣. 一种新的测试序列生成方法研究[J]. 电子测量与仪器学报, 2007, 21(2):31-34.
GAO X, WANG X. A new method to generate test sequences for protocol conformance testing[J]. Journal of Electronic Measurement and Instrument, 2007, 21(2): 31-34.

[11] NGUYEN C D, MARCHETTO A, TONELLA P. Combining model-based and combinatorial testing for effective test case generation[C]. The 2012 International Symposium on Software Testing and Analysis, 2012: 100-110.

[12] KRUSE P M, WEGENER J. Test sequence generation from classification trees [C]. 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012:539-548.

[13] SCHOOLJAN H. Test sequence validation and generation using classification trees[D]. Delft: Delft University of Technology, 2013.

[14] AL éGROTH E, MATSUKI S, VOS T E J, et al. Overview of the ICST international software testing contest[C]. 2017 IEEE Tenth International Conference on Software Testing, Verification and Validation, 2017: 550-551.

[15] PITSCHINETZ R, WEGENER J. TESSY-Management of Software Tests [J]. Fifth International Federation of Automatic Control Workshop on Experience with the Management of Software Projects, 1996,29(2):11-16.

[16] RUF J, KROPF T. Symbolic model checking for a discrete clocked temporal logic with intervals[M]. LI H F, PROBST D K. Advances in Hardware Design and Verification. New York: Springer US, 1997:146-163.

[17] 盛云龙, 魏长安, 姜守达. 基于 CCTL 的软件可靠性测试输入特性描述方法[J]. 仪器仪表学报, 2018, 39(4):141-149.
SHENG Y L, WEI C A, JIANG S D. CCTL based description method of input characteristics for software reliability test [J]. Chinese Journal of Scientific Instrument, 2018, 39(4):141-149.

[18] SHAWE T J, CRISTIANINI N. Kernel methods for pattern analysis [M]. Beijing: China Machine Press, 2005.

[19] CAMPOS S V, CLARKE E M. Real-time symbolic model checking for discrete time models [M]. Pittsburgh: Carnegie Mellon University, 2015.

[20] FLAKE S, MüLLER W, RUF J. Mapping of structured English sentences to CCTL formulae[J]. C-LAB, Tech. Rep, 2000(4): 2000.

作者简介

盛云龙, 2013 年于哈尔滨工业大学获得硕士学位,现为哈尔滨工业大学在读博士研究生,主要研究方向为自动测试技术和组合测试方法。
E-mail: 13B901008@hit.edu.cn

Sheng Yunlong received his M. Sc. degree in 2013 from Harbin Institute of Technology. He is currently a Ph. D. candidate at Harbin Institute of Technology. His main research interests include automatic test technique and combinatorial testing method.

魏长安(通信作者),2003 年于哈尔滨工业大学获得学士学位, 2009 年于哈尔滨工业大学获得硕博连读博士学位,现为哈尔滨工业大学副教授,主要研究方向为虚拟试验技术和自动测试技术。
E-mail: weichangan@hit.edu.cn

Wei Chang′an (Corresponding author) received his B. Sc. degree from Harbin Institute of Technology in 2003, and received his Ph. D. degree that involved postgraduate and doctoral study from Harbin Institute of Technology in 2009. He is currently an associate professor at Harbin Institute of Technology. His main research interests include virtual experiment technology and automatic test technique.