ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.1, January 2007, pp.62−73
DOI: 10.1360/jos180062

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种基于时间自动机的实时系统测试方法*

陈 伟[1,2+], 薛云志[1,2], 赵 琛[1], 李明树[1,3]

[1](中国科学院 软件研究所 互联网软件技术实验室,北京 100080)

[2](中国科学院 研究生院,北京 100049)

[3](计算机科学重点实验室(中国科学院 软件研究所),北京 100080)

A Method for Testing Real-Time System Based on Timed Automata

CHEN Wei[1,2+], XUE Yun-Zhi[1,2], ZHAO Chen[1], LI Ming-Shu[1,3]

[1](Laboratory for Internet Software Technologies, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate School, The Chinese Academy of Sciences, Beijing 100049, China)

[3](Laboratory of Computer Science (Institute of Software, The Chinese Academy of Sciences), Beijing 100080, China)

+ Corresponding author: Phn: +86-10-62565272, E-mail: chenwei@itechs.iscas.ac.cn, http://www.iscas.ac.cn

Chen W, Xue YZ, Zhao C, Li MS. A method for testing real-time system based on timed automata. *Journal of Software*, 2007,18(1):62−73. http://www.jos.org.cn/1000-9825/18/62.htm

Abstract: This paper provides an approach to test real-time systems modeled by timed input/output automata (TSIOA), which is a variant of TA (timed automata). This method consists of three steps. Firstly, system model depicted by TSIOA is transformed into an USTGSS (untimed stable label transition graph of symbolic state) which does not contain abstract time delay transitions. Then, the testing methods based on LTS (labeled transition system) are used to the generate transition sequences from USTGSS according to structural coverage criteria. Finally, a process of constructing and executing the test cases is given, in which object functions of time delay variables are imported, and time delay variables used in the transition sequences are solved dynamically by linear programming techniques.

Key words: timed safety input/output automata; real-time system testing; minimized stable label transition graph of symbolic state; test cases generation

摘 要: 基于时间自动机(timed automata,简称 TA)的一种变体——时间安全输入/输出自动机(timed safety input/output automata,简称 TSIOA),提出了一种实时系统测试方法.该方法首先将时间安全输入/输出自动机描述的系统模型转换为不含抽象时间延迟迁移的稳定符号状态迁移图(untimed stable transition graph of symbolic state,简称 USTGSS);然后采用基于标号迁移系统(labeled transition system,简称 LTS)的测试方法来静态生成满足各种结构覆盖标准的包含时间延迟变量迁移动作序列;最后,给出了一个根据迁移动作序列构造和执行测试用例的过程,该过程引入了时间延迟变量目标函数,并采用线性约束求解方法动态求解迁移动作序列中的时间

* Supported by the National Natural Science Foundation of China under Grant No.60373053 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2004AA1Z2100 (国家高技术研究发展计划(863)); the Hundred Talents of the Chinese Academy of Sciences (中国科学院"百人计划")

Received 2005-10-31; Accepted 2005-12-13

延迟变量.

关键词: 时间安全输入/输出自动机;实时系统测试;最简稳定符号状态迁移图;测试用例生成
中图法分类号: TP311 文献标识码: A

与运行环境的交互行为存在时间约束的系统,称为实时系统.对于很多实时系统而言,系统的功能性错误或者对时间约束的偏移,都会产生灾难性的后果.为了提高实时系统的质量,一般采用验证或测试的方法,其中,测试是唯一能够在运行时刻检验实时系统动态行为的方法[1].从 20 世纪 70 年代开始,研究人员就已经基于各种时间无关形式模型,如有限状态机、扩展有限状态机、标号迁移系统等等,提出了许多测试用例生成算法,其中有些算法,如 U-方法[2,3]、D-方法[4]、W-方法[4,5]和 Wp-方法[6],已经在通信协议、硬件电路设计等领域得到了较为广泛的应用.但是,这类方法无法描述实时系统中的时间约束.从 20 世纪 90 年代中期开始,随着时序逻辑(temporal logic,简称 TL)[7]、时间自动机(timed automata,简称 TA)[8−10]、时间标号迁移系统(timed labeled transition system,简称 TLTS)[11]等时间相关形式模型理论的逐步成熟,人们开始研究如何利用时间相关的形式模型来对实时系统进行测试.然而,由于实时系统引入了时间维,理论上来说其状态空间是无限的.另外,系统中的时间约束也增加了分析系统可能行为的难度,这些都给实时系统测试带来了极大的困难.

一些学者提出首先将实时系统模型转换为不含时间约束系统模型,以消除时间约束对系统行为分析造成的影响,然后使用时间无关的测试方法进行测试.其中:文献[12]提出一种增加了时间度量的时序逻辑,并基于这一逻辑给出了测试用例生成方法.该方法首先对时间域进行离散化处理,并根据系统历史信息生成测试用例;文献[13]提出了一种基于 TLTS 的测试用例生成方法,首先将 TLTS 模型转换为一个标号迁移系统(labeled transition system,简称 LTS),然后使用 W-方法来生成测试用例;文献[14,15]给出了一种用于测试 TA 的方法,该方法将 TA 转换为格自动机(grid automata,简称 GA),并分别利用 W-方法和 Wp-方法来生成测试用例;文献[16]采用时间抽象互模拟方法[17]将 TA 模型转换为有限状态迁移图,并将有限状态迁移图转换为非确定 FSM,从而采用基于 FSM 的方法进行测试.然而,上述这些方法为了能够实现时间模型向非时间模型的转换,大多对模型的时间描述能力进行了限制.另外,尽管转换生成的时间无关模型的状态数有限,但仍然相当庞大,还需要进一步简化.

为了降低状态空间爆炸给测试用例生成时带来的难度,文献[7,8,17−22]采用各种状态等价关系来对系统进行简化.这些方法的基本思想均是将 TA 中的一个位置和一个时间域一起构成一个符号状态(symbolic state)以生成有限状态模型,其中最为典型的是域图(region graph)[7]和区图(zone graph)[8].大多数实时系统模型检测方法都使用区图,这是因为区图所产生的符号状态数较少,并且可以采用 on-the-fly 的方法在测试过程中逐步生成.但是,区图并不适用于测试,其主要原因是区图中的符号状态不具有稳定性*属性,这使得根据区图生成的符号状态迁移序列并不一定可行.因此,采用模型检测工具来生成测试用例的方法[23,24]往往只能用于测试系统中与状态可达性相关的属性.域图生成的符号状态图虽然满足稳定性,但是该方法生成的符号状态数会随着时间自动机中时钟个数以及时间约束常数的大小而呈指数增长[7].文献[17−19]采用符号状态拆分(partition-refinement,简称 PR)算法以生成最简稳定符号状态迁移图(minimal stable transition graph of symbolic state,简称 MSTGSS).尽管对于状态可达性分析而言,该算法生成的符号状态迁移图是最简稳定的,但是对于测试而言,该算法所生成的迁移图中仍然存在冗余的抽象时间延迟迁移.另外,该算法的执行效率也有待改进.

本文采用 TA 的一种变体——时间安全输入/输出自动机(timed safety input/output automata,简称 TSIOA)作为系统的形式模型,基于该模型通过增加预处理过程以及采用更加简洁的符号状态拆分算子等方法改进了 PR 算法,并从所生成的 MSTGSS 中去除了与测试无关的抽象时间延迟迁移,从而得到更加简单的不含抽象时间迁移稳定符号状态迁移图(untimed stable transition graph of symbolic state,简称 USTGSS).获得 USTGSS 之后,便利用它来构造和执行测试用例,其过程主要分为两步:首先,根据 USTGSS 来静态生成含时间延迟变量的迁移

* 参见第 2.1 节中的定义.

动作序列;然后,采用线性约束求解方法动态求解迁移动作序列中的时间延迟量并执行测试.为了提高测试用例的错误检测能力并减少生成的测试用例的数目,在生成迁移动作序列时,要考虑针对 USTGSS 的各种结构覆盖标准,而在求解时间延迟量时还需要引入时间延迟策略.

本文第 1 节介绍我们所使用的 TSIOA 的形式语法和语义.第 2 节提出一种 TSIOA 简化方法,将 TSIOA 描述的系统模型转换为 USTGSS.第 3 节说明如何根据 USTGSS 来构造和执行测试.第 4 节则是结论及后续研究工作的设想.

1 TSIOA 模型

从 Alur 最早提出 TA[7]以来,一些学者提出了一些 TA 的变体[10,14,21,22],本文所使用的 TSIOA 主要是从时间安全自动机(timed safety automata,简称 TSA)[21]发展而来.TSA 通过对自动机中各个位置赋一个时间不变量,非常简洁地解决了 TA 中迁移必须发生的问题,从而避免了建立复杂的接收条件.然而,TSA 并不对外部环境的动作和系统本身的动作进行区分,因而无法用于对系统的测试.TSIOA 在 TSA 的基础上对自动机中的迁移动作进行区分,根据迁移动作是由外部环境触发的还是由系统本身产生的,将动作分为输入动作和输出动作.下面具体介绍 TSIOA 形式模型.

1.1 形式语法

一个时间安全输入/输出自动机 A 是一个五元组(L,l_0,T,C,I),其中:

- L 是一个位置(或节点)的有限集合;
- l_0 是初始位置;
- $\Sigma=\Sigma^I\cup\Sigma^O$ 是迁移动作集合,用 a,b,c 等符号表示,其中动作分为输入动作和输出动作,分别在动作之前用“?”和“!”表示;
- C 是连续时钟的集合,其中每一个元素用 x,y 等符号表示,可以取实数值;
- $\Phi(C)$是时钟约束的集合,其中每一个元素用 g 或 Z 表示,具体形式由下面的语法产生:

$$\psi:=x<n|x\le n|x>n|x\ge n|x-y<n|x-y\le n|\psi_1\wedge\psi_2,\text{其中:}x,y\in C,n\in N;$$

- $T\subseteq L\times\Phi(C)\times\Sigma\times 2^C\times L$ 是迁移的集合;
- $I:L\to\Phi(C)$,对每一个位置赋时间不变量.

对于一个迁移 $e=(l,g,(?,!)a,r,l')\in T$,可以表示为 $l\xrightarrow{g,(?,!)a,r}l'$, 是指从位置 l 经过输入或输出动作 a 迁移到位置 l'.TSIOA 假定动作迁移都是瞬间迁移,迁移发生时的时钟值属于 $I(l)$并满足约束条件 g,迁移发生后的时钟值属于 $I(l')$.为了描述方便,令 $e.souce=l,e.guard=g,e.action=a,e.reset=r,e.target=l'$.图 1 是一个 TSIOA 模型 A,其中并没有对位置 l_2,l_3 标记时间不变量,这表示它们的时间不变量均为 $x\ge0\wedge y\ge0$.在本文介绍的过程中,主要采用 A 作为系统模型来进行示例说明.

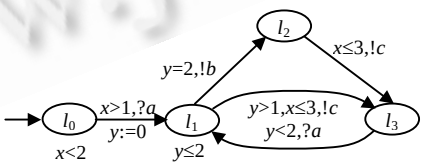

Fig.1 A model of TSIOA: A

图 1 一个 TSIOA 模型:A

1.2 操作语义

一个 TSIOA A 的语义是由与它对应的迁移系统 S_A 来定义的.S_A 的状态是二元组$\langle l,v\rangle$,l 是 A 的一个位置;v 是一个时钟解释,它对所有的时钟进行赋值.A 的所有状态集合由 Q_A 表示.如果 l 是 A 的一个初始位置并且对所有时钟 x 有 $v(x)=0$,则$\langle l,v\rangle$是初始状态.S_A 中有以下两种状态转换:

- 时间延迟迁移:如果对于一个状态$\langle l,v\rangle$和一个时间增量$\delta\geq 0$,其中 $v\in I(l),v+\delta\in I(l)$,那么,$\langle l,v\rangle \xrightarrow{\delta} \langle l,v+\delta\rangle$;
- 动作迁移:对于一个状态$\langle l,v\rangle$和一个迁移 $l \xrightarrow{g,(?,!)a,r} l'$, 其中 $v\in I(l)\cap g$,并且 $v[r\rightarrow 0]\in I(l')$,那么,$\langle l,v\rangle \xrightarrow{\{?,!\}a} \langle l',v\rangle$.

当不区分时间延迟迁移和动作迁移时,可以采用 $s\Rightarrow s'$表示从状态 s 到 s'的一次迁移.显然,S_A 是一个以 $\Sigma^I\cup\Sigma^O\cup R$ 为标号集合的 LTS.如图 1 所示的模型 A,与之对应的迁移系统的状态集 $Q_A=\{l_0,l_1,l_2,l_3\}\times R^2$,其中,后两个实数分别表示时钟 x 和 y 的值.下面是 A 的一个可能的执行行为:

$$\langle l_0,0,0\rangle \xrightarrow{1.2} \langle l_0,1.2,1.2\rangle \xrightarrow{?a} \langle l_1,1.2,0\rangle \xrightarrow{1.5} \langle l_1,2.7,1.5\rangle \xrightarrow{!c} \langle l_3,2.7,1.5\rangle \xrightarrow{0.2} \langle l_3,2.9,1.7\rangle$$
$$\xrightarrow{?a} \langle l_1,2.9,1.7\rangle \xrightarrow{0.3} \langle l_1,3.2,2\rangle \xrightarrow{!b} \langle l_2,3.2,2\rangle.$$

2 TSIOA 模型的转换

因为 TSIOA 中时间变量可以取实数值,所以,即使一个非常简单的 TSIOA 模型也会存在无穷多个状态.另外,TSIOA 的动作迁移中含有时钟约束条件,只有当时钟约束条件满足时,动作迁移才可以进行.这种状态空间爆炸以及时钟约束条件对模型中行为的影响,给测试带来了极大的困难,为此,需要将 TSIOA 模型转换为更加适合测试的模型.

2.1 基本术语

符号状态:对于一个 TSIOA A,符号状态由 A 中的某一个位置和一个时间域一起构成,表示为$\langle l,Z\rangle$,其中:$l\in A.L$;而 Z 是一个时间域,$Z\in A.\Phi(C)$.对于一个符号状态 $s=\langle l,Z\rangle$,$s.loction=l$,$s.time=Z$.

动作紧前状态:$Actionpred(e,S)=\{s|\exists s'\in S.s \xrightarrow{e} s'\}$,这里,$Actionpred(e,S)$是一个符号状态,包含了经过动作迁移 e 可以到达符号状态 S 的所有状态,当没有这样的状态时,$Actionpred(e,S)=\varnothing$.

时间紧前状态:$Timepred(S)=\{s|\exists s'\in S,\delta\in R.s \xrightarrow{\delta} s'\}$,这里,$Timepred(e,S)$是一个符号状态,包含了经过时间延迟迁移可以到达符号状态 S 的所有状态.

紧前状态:$preds(S)=\{s|s\in Actionpred(e,S),e\in\Sigma\}\cup\{s|s\in Timepred(e,S)\}$,经过一次动作迁移或者一次时间延迟迁移可以到达符号状态 S 的所有状态的集合.

紧前符号状态:$pre_\rho(X)=\{Y|Y\in\rho,Y\cap preds(X)\neq\varnothing\}$,$\rho$是符号状态的集合,对于$\rho$中的任意符号状态 Y,如果 Y 含有可以经过一次迁移到达 X 的状态,那么 Y 是 X 的紧前符号状态.

紧后符号状态:$pre_\rho(X)=\{Y|Y\in\rho,X\cap preds(Y)\neq\varnothing\}$,$\rho$是符号状态的集合,对于$\rho$中的任意符号状态 Y,如果 X 含有可以经过一次迁移到达 Y 的状态,那么 Y 是 X 的紧后符号状态.

稳定性:对于任意两个符号状态 S 和 S',S 关于 S'稳定,当且仅当,如果$\exists s\in S,\exists s'\in S'$,使得 $s\Rightarrow s'$,那么从 S 中的任意状态出发,都可以经过相同的迁移到达 S'.

对于一个符号状态集合 W,如果$\forall S,S'\in W$,满足 S 关于 S'稳定,则 W 具有稳定性.

对于一个 TSIOA,可通过等价划分的方法,构造出包含该 TSIOA 所有可达状态且满足稳定性的符号状态集合,例如根据域等价[7]、强(弱)时间抽象互模拟[17]等关系所得到的符号状态集合均满足稳定性.

最简稳定性:对于一个 TSIOA 的所有状态,如果存在某种等价划分方法得到的符号状态集合满足稳定性,并且不存在状态数更少且满足稳定性的划分,则称这种划分方法得到的符号状态集合满足最简稳定性.

拆分操作:拆分操作是将 TSIOA 的状态进行等价划分的一个非常重要的操作,通过该操作可以对符号状态进行拆分,并且拆分的结果满足最简稳定性.

首先定义时间域上的拆分操作,下面的 Z,Z'以及 $Z_1,\ldots,Z_k$ 均为时间域.

$$split(\{Z,Z'\})=\{Z\cap Z'\}\cup\{Z\backslash Z'\} \tag{1}$$

$$split(\{Z_1,\ldots,Z_k\},Z')=\bigcup_{i=1}^{k} split(Z_i,Z') \tag{2}$$

$$split(Z,\{Z_1,\ldots,Z_k\})=split(split(\ldots split(Z,Z_1),\ldots,Z_{k-1})Z_k) \tag{3}$$

式(1)表示根据 Z'来对 Z进行拆分;式(2)表示根据 Z'来对时间域集合$\{Z_1,\ldots,Z_k\}$进行拆分;式(3)表示根据时间域集合$\{Z_1,\ldots,Z_k\}$来对 Z 进行拆分.

还可以对这个算子进行扩展,使其适用于符号状态.$S=\langle l,Z\rangle$,$S'=\langle l',Z'\rangle$,$S''=\langle l,Z''\rangle$,$S_1=\langle l_1,Z_1\rangle,\ldots,S_k=\langle l_k,Z_k\rangle$均为符号状态,其中,$l\neq l'$.

$$split(S,\{Z_1,\ldots,Z_k\})=\{\langle l,Z'\rangle|Z'\in split(Z,\{Z_1,\ldots,Z_k\})\} \tag{4}$$

$$split(S,S')=split\left(S,\bigcup_{e\in\Sigma}\{Actionpred(e,S_i).time\mid l\xrightarrow{e}l'\}\right) \tag{5}$$

$$split(S,S'')=split\left(S,\bigcup_{e\in\Sigma}\{Actionpred(e,S'').time\mid l\xrightarrow{e}l''\}\cup\{Timepred(S'').time\}\right) \tag{6}$$

$$split(S,\{S_1,\ldots,S_k\})=split\left(S,\bigcup_{i=1}^{i=k}\bigcup_{e\in\Sigma}\{Actionpred(e,S_i).time\mid l\xrightarrow{e}l_i\}\cup\bigcup_{i=1}^{i=k}\{Timepred(S_i).time\mid l=l_i\}\right) \tag{7}$$

式(4)表示根据时间域集合$\{Z_1,\ldots,Z_k\}$来对符号状态 S 进行拆分;式(5)表示根据符号状态 S'来对符号状态 S 进行拆分,其中 S 和 S'的位置不同;式(6)表示根据符号状态 S''来对符号状态 S 进行拆分,因为 S 和 S''的位置相同,所以可能存在时间延迟迁移,故而与式(5)相比,还需要根据可能的时间延迟迁移来对 S 进行拆分;式(7)表示根据符号状态集合$\{S_1,\ldots,S_k\}$来对 S 进行拆分.

与文献[17−19]不同的是,以上在 *split* 的定义过程中没有采用 U 算子,并利用 *Actionpred* 和 *Timepred* 代替了⇑算子.通过这种定义方式,不需要在式(1)中求解 $Z'\backslash Z$,同时还避免了在式(5)~式(7)中将第 2 个参数中的每一个时间域与第 1 个参数的时间域作交集运算,从而有效地提高了拆分操作的执行效率.

引理. 对于$\forall X\in split(Z,\{Z_1,\ldots,Z_m\})$,满足 $X\cap Z_i\in\{X,\varnothing\}(1\leq i\leq m)$,并且所有 $split(Z,\{Z_1,\ldots,Z_m\})$的元素构成了对 Z 的一个划分.

证明:采用归纳法证明.当 $m=1$ 时,根据式(1),该结论显然成立;假设 $m=n$ 时该结论成立,令 $split(Z,\{Z_1,\ldots,Z_n\})=\{Z'_1,\ldots,Z'_p\}$,则 $Z'_j\cap Z_i\in\{Z'_j,\varnothing\}$,$\bigcup_{d=1}^{p}Z'_d=Z$,$Z'_j\cap Z'_h=\varnothing(1\leq i\leq m,1\leq j\leq p,1\leq h\leq p,j\neq h)$;当 $m=n+1$ 时,由式(3),$split(Z,\{Z_1,\ldots,Z_n\})=split(\{Z'_1,\ldots,Z'_p\},Z_{n+1})=\bigcup_{d=1}^{p}split(Z'_i,Z_{n+1})=\{Z''_1,Z''_2,\ldots,Z''_{2p-1},Z''_{2p}\}$,其中:$Z''_{2i-1}=Z'_i\cap Z_{n+1}$,$Z''_{2i}=Z'_i\backslash Z_{n+1}(1\leq i\leq p)$.因为 $Z'_j\cap Z'_h=\varnothing(1\leq j\leq p,1\leq h\leq p,j\neq h)$,所以,$Z'_i\cap Z'_j\in\{Z''_i,\varnothing\}(1\leq i\leq 2p,1\leq j\leq p)$;显然,$Z''_i\cap Z_{n+1}\in\{Z''_i,\varnothing\}(1\leq i\leq 2p)$,结合 $m=n$ 时的结论,$Z''_j\cap Z_i\in\{Z''_j,\varnothing\}$,$\bigcup_{d=1}^{2p}Z''_d=Z$,$Z''_j\cap Z''_h=\varnothing(1\leq i\leq m,1\leq j\leq 2p,1\leq h\leq 2p,j\neq h)$. □

定理. $split(S,\{S_1,\ldots,S_k\})$得到的符号状态集相对于$\{S_1,\ldots,S_k\}$是最简而稳定的.

证明:首先证明稳定性.不妨令 $S=\langle l,Z\rangle$,$split(S,\{S_1,\ldots,S_k\})=split(S,\{Z_1,\ldots,Z_m\})$,根据式(7),对于 $1\leq i\leq m$,$Z_i=\{Actionpred(e,S_j).time\mid l\xrightarrow{e}S_j.location\}$,或 $Z_i=\{Timepred(S_i).time\mid l\xrightarrow{e}S_i.location\}$.根据前面引理,易得$\forall Y\in split(S,\{Z_1,\ldots,Z_m\})$,$1\leq j\leq k$,满足 $Y\cap\{Timepred(S_j)|l=S_j.location\}\in\{Y,\varnothing\}$,$Y\cap\{Actionpred(e,S_j)\mid l\xrightarrow{e}S_j.location\}\in\{Y,\varnothing\}$,故而,$split(S,\{S_1,\ldots,S_k\})$得到的符号状态集相对于$\{S_1,\ldots,S_k\}$是稳定的.

采用反证法证明最简稳定性,假设 $split(S,\{S_1,\ldots,S_k\})=split(S,\{Z_1,\ldots,Z_m\})=\{S'_1,\ldots,S'_n\}$,这里,$\{S'_1,\ldots,S'_n\}$不是 S 相对于$\{S_1,\ldots,S_k\}$的最简稳定的拆分,那么存在一个数目小于 n 的稳定拆分结果.显然,其中必然存在一个符号状态 Y,满足 $Y\cap S'_i=Y_1\neq\varnothing$,$Y\cap S'_j=Y_2\neq\varnothing$,$1\leq i<j\leq n$.因为 Y 相对于$\{S_1,\ldots,S_k\}$是稳定的,所以,对于 $1\leq i\leq k$,$Y\cap S_i\in\{Y,\varnothing\}$;对于 $1\leq j\leq m$,$Y.time\cap Z_i\in\{Y.time,\varnothing\}$.因此,根据式(3)的计算过程,$Y$ 中的元素不会被分割开.然而,假设中 Y 中的元素至少分布在两个由 $split(S,\{Z_1,\ldots,Z_m\})$计算得到的符号状态 S'_i 和 S'_j 中,故而产生矛盾.所以,$split(S,\{S_1,\ldots,S_k\})$得到的符号状态集最简稳定的. □

2.2 改进的符号状态拆分算法

对于一个时间自动机模型,PR 算法从其初始符号状态出发,按照迁移关系进行遍历,每遇到一个符号状态

便利用拆分操作来判断其是否满足稳定性**,如果不满足,则回溯到该符号状态的紧前符号状态重新判断其稳定性.算法不断递归地进行符号状态拆分,直至获得满足稳定性的符号状态集合为止.关于 PR 算法的详细描述请参考文献[17–19].

由于 PR 算法是基于一般时间自动机的,为了使其适用于 TSIOA 模型,需要改变初始符号状态集合的构造方法,根据 TSIOA 模型中的所有位置以及位置对应的时间不变量,创建最初的符号状态集合.另外,该算法中拆分操作是按照迁移关系遍历进行的,如果一个符号状态与初始状态的距离比较远,并且该符号状态的拆分操作使得之前的稳定的符号状态变为不稳定,那么必然会导致符号状态拆分操作反复迭代执行,从而影响算法的执行效率.观察发现:每当第一次对某个初始符号状态 S 进行拆分时,都必然要根据 S 自身的时间域和从 S 出发的迁移的约束条件来进行拆分,所以可以提前执行这个拆分操作,而不是等到根据迁移关系遍历到 S 时才对 S 进行拆分.这样就避免了每次对一个不稳定的初始符号状态拆分之后,回溯到其紧前符号状态重新进行拆分操作,可以有效地减少算法执行过程中回溯的次数.

一个改进的 PR 算法 IPR(improved partition-refinement)如图 2 所示.其中:参数 A 是一个 TSIOA;ρ是当前符号状态集;α是可达符号状态集;σ是当前α集合中稳定的符号状态集合.算法第(3)步~第(6)步是预处理过程,其中:第(3)步、第(4)步将 A 的每一个位置与其时间不变量组合构成最初的符号状态集合 W;第(5)步、第(6)步将 W 中的每一个符号状态根据与其相关的迁移的条件进行拆分.算法从第(8)步开始对符号状态集中符号状态进行拆分,如果一个可达符号状态 X 已经是稳定的,那么第(12)步将 X 加到σ并将其所有紧后符号状态加入α中;否则,将 X 移出α.为了避免α为空,第(14)步对 X 的拆分结果进行判断,如果存在含有初始状态的符号状态 Y,则将其加入到α;第(15)步、第(16)步分别将 X 的紧前符号状态移出σ,并根据拆分结果更新ρ.根据前面的定理,算法第(10)步中 *split* 拆分操作得到的符号状态集合是最简稳定的,所以最终得到的符号状态集合是最简稳定的.

```
(1)   IPR (A){
(2)   let W:=∅, ρ:=∅;
(3)   for every l∈A.L do
(4)      W:=W∪{⟨l,I(l)⟩}
      od
(5)   for every S_i∈W do
(6)    ρ:=ρ∪split(S_i,{e.guard|e.source=S_i.location})
       od
(7)   let α:={[s_0]}, σ:=∅;
(8)   while α≠σ do
(9)      choose X in α\σ;
(10)     let α′:=split(X,ρ);
(11)     if α′:={X} then
(12)        σ:=σ∪{X}; α:=α∪post_ρ(X);
         else
(13)        α:=α\{X};
(14)        if ∃Y∈α′ such that s_0∈Y then α:=α∪{Y};
         fi
(15)        σ:=σ\pre_ρ(X);
(16)        ρ:=(ρ\{X})∪α′;
          fi
      od
(17)  return α;}
```

Fig.2 Improved partition-refinement algorithm

图 2 改进的符号状态拆分算法

以图 1 所示 A 为参数调用算法 IPR,图 3 是在算法 IPR 输出的结果基础上增加迁移关系后得到的迁移图 G.这种迁移图被称为 MSTGSS,它是一个三元组$\{V,E,v_0\}$,分别表示迁移图中符号状态集合、迁移集合以及初始符号状态.与采用 PR 算法对 A 进行化简相比,IPR 算法可以减少 4 次符号状态拆分操作.

** 如果执行拆分操作导致符号状态发生变化,则说明该符号状态不稳定.

2.3 抽象时间迁移去除

图 3 所示的 MSTGSS G 中除了含有 A 中的动作迁移以外,还含有抽象的时间延迟迁移.抽象的时间延迟迁移表示经过了一个时间延迟,但是不能确定具体的时间量.实际上,对于符号状态图中的每一个动作迁移来说,都存在相应的时间延迟迁移位于动作迁移之前.但是,MSTGSS 中只描述了那些会导致符号状态的可达性发生变化的时间延迟迁移,而那些不会导致符号状态可达性发生变化的时间延迟迁移则隐含在符号状态中.这种对时间延迟迁移的区分对于符号状态可达性分析是有意义的.但是,在测试执行过程中只能观察被测系统的输入、输出动作以及动作发生的时刻,而并不知道时间延迟迁移是否对迁移图中符号状态的可达性产生了影响.另外,测试所需要的是具体的动作发生时刻,而并不是抽象的时间延迟.所以,对于测试而言,这种区分没有用处.为此,如果是为了测试的目的,可以将 MSTGSS 中的所有抽象时间延迟迁移去除,从而对迁移图进一步简化.在去除了抽象时间延迟迁移后得到的迁移图中,每一个动作迁移之前都有一个隐含的时间延迟迁移,故而可以在迁移图中每一个迁移动作之前增加一个时间变量来表示时间延迟量,并将这个时间变量与动作迁移合并成新的迁移标号.对于一个含有时间变量的迁移标号 a,$a.first$ 表示 a 中的时间变量;$a.second$ 表示迁移动作.

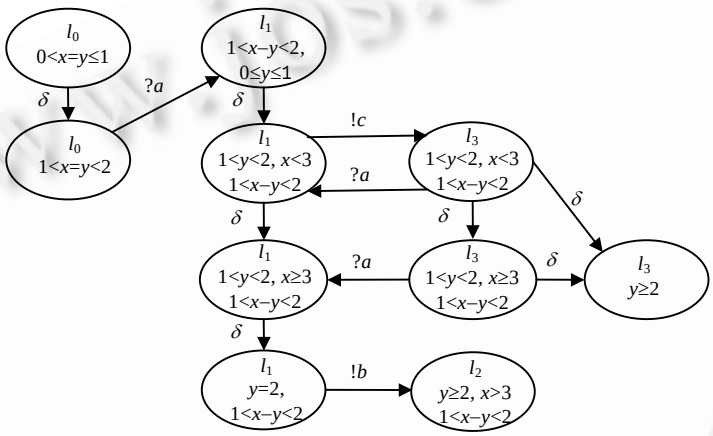

Fig.3 The MSTGSS G of A by applying algorithm IPR

图 3 对 A 实施 IPR 算法后生成的 MSTGSS G

图 4 所示的抽象时间延迟迁移去除(abstract time delay transitions removing,简称 ATDTR)算法用于去除符号状态迁移图中的抽象时间迁移.其主要思想是:将迁移图中的抽象时间延迟迁移与紧跟其后的动作迁移合并,合并的迁移只用动作迁移来表示,从而将抽象时间延迟迁移以及那些只用于表示这种迁移的符号状态从迁移图中去除.其中:算法第(3)步~第(10)步用于将抽象时间延迟迁移从 G 中去除;算法第(11)步~第(13)步用于删除那些由于去除了显示时间延迟迁移而导致在符号状态图中不可达的符号状态及相关迁移;算法第(14)步~第(16)步在符号状态图中每一个迁移动作之前增加一个用于表示时间迁移的变量 t_i.

图 5 是将图 3 所示的 MSTGSS G 作为输入,调用 ATDTR 算法输出得到的符号状态迁移图 G',本文称其为 USTGSS.USTGSS 使用时间延迟变量来统一表示 MSTGSS 中显示的以及隐含的时间延迟迁移,是对 MSTGSS 的一种更简洁的表示.显然,如果只考虑动作迁移而不考虑时间延迟迁移,那么 USTGSS 也满足稳定性.对比图 5 和图 3 可以看出:由于去除了抽象时间延迟迁移,USTGSS 的规模要比 MSTGSS 小很多.

```
(1)  ATDTR (G){
(2)  let ET={e|e∈G.E∧e.act=τ};
(3)  while ET≠∅ do
(4)      choose X from ET
(5)      let  TT={e|e.source=X.target};
(6)      if TT≠∅ then
(7)        for every tt∈TT do
(8)          if {X.source, tt.action, tt.target}∉G.E then
(9)              G.E:=G.E∪{X.source, tt.action, tt.target};
               fi
           od
         fi
(10)     G.E:=G.E\{X};
     od
(11)  while SV={v|v∈G.V, ¬∃e∈G.E, e.target=v}\{G.v0}≠∅
      do
(12)     for every v in SV do
(13)        G.E:=G.E\{e|e.source=v}, G.V:=G.V\{v};
         od
     od
(14)  let i:=1;
(15)  for every e∈G.E do
(16)     e.action:=ti·e.action, i++;
     od
(17)  return G;}
```

Fig.4 The algorithm for abstract time delay transitions removing

图 4 抽象时间迁移去除算法

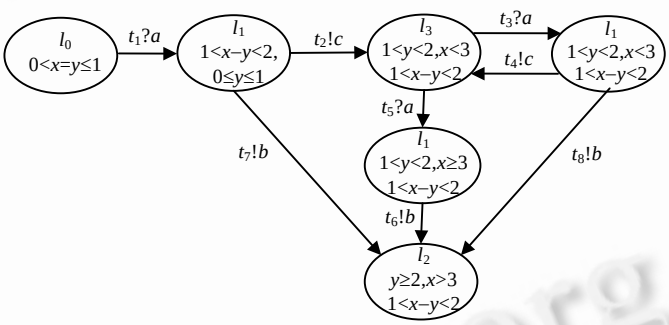

Fig.5 The USTGSS G' of G by applying algorithm ATDTR

图 5 对 G 实施算法 ATDTR 后生成的 USTGSS G'

3 时间测试用例的构造

对于动作迁移而言,USTGSS 具有稳定性.所以,USTGSS 中所有的动作迁移一定是可行的.如果不考虑迁移过程中的时间延迟,便可以采用时间无关的测试方法来获得可行的迁移动作序列.因为根据 USTGSS 生成的迁移动作序列是可行的,故而迁移动作序列中的时间延迟变量一定有解.为了构造和执行测试用例,可以采用线性约束求解方法动态求解迁移动作序列中的时间延迟量.

3.1 迁移动作序列生成

USTGSS 中迁移标号是由一个时间变量和一个迁移动作构成.因为 USTGSS 针对动作迁移具有稳定性,所以生成的迁移动作序列一定可行.如果暂时不考虑时间因素,便可以使用基于 LTS 的测试方法[25−27]来生成迁移动作序列.需要注意的是:基于 LTS 的测试方法包括静态方法和动态方法.因为 USTGSS 中不包含确定的时间信息,所以生成的动作序列无法直接执行,故而只能使用静态方法来生成迁移动作序列.在迁移动作序列的生成过程中,可以考虑各种结构覆盖标准,如状态覆盖、迁移覆盖和路径覆盖[28]等等.另外,为了避免生成的动作序列过多,在生成测试用例时,还可以引入测试目的[29],从而只对系统的某一部分或者特定属性进行测试.

这里,根据图 5 所示的符号状态迁移图生成满足状态覆盖和迁移覆盖的迁移动作序列集合

$$TS=\{(t_1?a)(t_7!b),(t_1?a)(t_2!c)(t_3?a)(t_4!c),(t_1?a)(t_2!c)(t_5?a)(t_6!b),(t_1?a)(t_2!c)(t_3?a)(t_8!b)\}.$$

3.2 测试用例构造与执行

根据 USTGSS 生成的迁移动作序列中含有输入动作和输出动作迁移,并且在每一个迁移动作之前还有一个时间变量,表示迁移动作发生之前的时间延迟量.为了执行测试,需要确定动作序列中输入动作之前时间变量的具体值.另外,对于输出动作之前时间变量,还需要能够判断输出动作实际发生时的时间延迟是否满足系统的要求.因为符号状态迁移图中所有相关的时间约束都是线性的,所以,可以采用线性约束求解方法来对时间变量的取值进行求解和正确性检验.对于一个迁移动作序列,首先按顺序对其中所有的时间变量建立一个线性约束系统,然后采用动态测试方法,一边计算输入动作之前的时间变量,一边执行测试.

图 6 中 RealExecute 是符号状态迁移系统 G 中的一个迁移动作序列σ的执行过程.其中第(2)步首先根据 G 和σ建立一个线形约束系统 CON.因为系统输出动作的时间无法控制,因而算法第(4)步~第(6)步在利用线性约束求解方法求解时间变量时,只对到下一个输出动作之前所有输入动作前的时间延迟量进行求解;第(7)步~第(9)步根据所求得的时间延迟量来执行相应的输入动作;第(10)步~第(17)步处理被测系统期望的输出动作,如果输出动作不是所期望的或者等待时间不满足约束限制,则返回 false;否则继续执行;第(18)步对已经确定值的时间变量进行赋值,并执行下一个转移动作.因为测试系统只能观察被测系统与外部环境交互的行为,如果一个输入动作之后没有输出动作,那么便无法判断该输入动作是否被正确地执行,故而算法 RealExecute 执行完σ中最后一个输出动作便返回.

```
(1)   RealExecute(G,σ){
(2)   construct a set CON of time constraints according to σ and G;
(3)   let k:=0, l:=len(σ);
(4)   while ∃k′ sat. k<k′≤l∧σ[k′].second∈Σ^o∧σ[i].second∈Σ^I,k<i<k′ do
(5)     if k′−k>1 do
(6)       calculate every σ[i].first which sat. corresponding constraints in CON, k<i<k′;
(7)       let i:=k+1;
(8)       while i<k′ do
(9)         execute σ[i].second when time of σ[i].first passed; i++;
          od
        fi
(10)    let timer t:=0, start(t);
(11)    calculate possible value field D of σ[k′].first according to CON.
(12)    while true do
(13)      if t.CurrentValue>Max(D) then return false; fi
(14)      if not receive any action then continue; fi
(15)      if receive action a≠σ[k′].second then return false;
(16)      else if t.CurrentValue<Min(D) then return false;
(17)           else goto (18); fi
          fi
        od
(18)    let t_k′:=t.CurrentValue, k:=k′
      od
(19)  return true;}
```

Fig.6 Constructing and executing test cases

图 6 测试用例构造与执行

为了更好地理解测试用例的构造和执行过程,这里选择上一节所生成的迁移动作序列集合 TS 中动作序列$(t_1?a)(t_2!c)(t_3?a)(t_4!c)$作为示例来说明算法 RealExecute 的执行过程.首先,根据图 5 中符号状态迁移关系和每个符号状态的时间域,可以很容易地为 t_1,t_2,t_3 和 t_4 建立一个约束系统 CON,主要包括以下约束条件***:

(1) $1<t_1<2$;(2) $1<t_2<2$;(3) $t_1+t_2<3$;(4) $1<t_2+t_3<2$;(5) $t_1+t_2+t_3<3$;(6) $1<t_2+t_3+t_4<2$;(7) $t_1+t_2+t_3+t_4<3$.

下面开始具体执行迁移动作序列.根据不等式(1)求解的 t_1 值,然后在 t_1 时刻执行$?a$;根据 t_1 的值和不等式

*** 这里默认所有的时间转移量都是非负实数,并且存在最大上限.

(2)、不等式(3)求解!c 的可能时间域 D 并等待!c:如果!c 没有发生,则返回错误;否则,记录从?a 发生到!c 发生时经过的时间延迟 t_2,判断 t_2 的值是否属于 D,如果不属于,则返回错误;根据 t_1,t_2 的值和不等式(4)、不等式(5)求解 t_3 的值,并在!c 发生后的 t_3 时刻执行?a;根据 t_1,t_2 和 t_3 的值以及不等式(6)、不等式(7)求解!b 的可能时间域 D 并等待!b:如果!b 没有发生,则返回错误;否则,记录从上一个?a 发生到!b 发生时经过的时间延迟 t_4,判断 t_4 的值是否属于 D,如果不属于,则返回错误.因为 USTGSS 满足稳定性,即迁移图中所有的迁移是可以执行的,因此,肯定存在满足这些约束的时间迁移量的解.

需要注意的是:在如 RealExecute 所示的动态测试过程中,并没有考虑求解满足线性约束的时间迁移量所耗费的时间.求解这些时间变量属于线性规划问题,该问题存在多项式时间算法[30],其计算复杂性为 $O(n^{3.5}L^2)$.这里,n 是时间变量的个数;L 是输入长度[****].对于实际的实时系统而言,执行行为中连续的输入动作迁移是相当有限的,所以 n 和 L 都比较小,故而对于那些对时间要求不是非常严格的系统而言,这种计算复杂性是可以接受的.

3.3 时间延迟策略

算法 RealExecute 中第(6)步在采用线性约束求解方法求解时间延迟变量值时,满足时间约束的时间延迟变量的解往往是一个实数集合,因此,必须考虑如何从中选择适合的解来执行测试.对于时间这样的物理量,边界值测试是一种非常有效的测试方法[28].因此,本文采用的策略是选择时间延迟变量的极值来执行测试.例如在求解迁移动作序列$(t_1?a)(t_2!c)(t_3?a)(t_4!c)$中时间变量 t_1 时,可以选择满足时间约束的极小(大)值 $1+\omega(2-\omega)$作为 t_1 的值,这里,ω是一个接近 0 的时间极小值.

为了能够选择时间延迟变量的极值来执行迁移动作序列,需要在算法中引入极值函数 *fun*.考虑到线性约束求解方法的能力,*fun* 可以是算法 RealExecute 第(6)步中所涉及的时间变量构成的线性函数.在算法 RealExecute 的第(6)步求解时间延迟变量时,除了要满足系统中相应的时间约束条件外,还必须让函数 *fun* 取极大或极小值.在实际的测试执行过程中,用户可以根据不同的测试目的来选择相应的极值函数.例如,为了测试一个迁移动作序列的最长(短)时间执行过程,可以将极值函数定义为所涉及的所有时间变量之和,即 $fun=\sum_{i=k+1}^{k'-1}t_i$ [*****],并要求时间变量的值可使 *fun* 最大(小).

假设存在 TSIOA A 的一个实现 B,B 和 A 的不同在于:对于输出动作迁移,只要迁移条件满足,该迁移便立即执行.表 1 所示是第 3.1 节中集合 *TS* 中的所有迁移动作序列在 B 上的最长和最短时间执行过程.从表 1 所示的迁移动作序列的执行过程可以看出,如果一直使用同一个极大(小)值函数,并不能保证所有的输入动作时间变量都能取到相应的极大(小)值.例如,迁移动作序列$(t_1?a)(t_2!c)(t_3?a)(t_4!c)$在采用极大值函数执行时,$t_3$ 非但没有取到最大值,反而是取最小值.这是因为前面时间变量 t_1 的取值影响了 t_3 执行$(t_1?a)(t_2!c)(t_3?a)(t_4!c)$时为了使得 t_3 能够取到最大值,可以在求解变量 t_1 时使用极小值函数,而在求解变量 t_3 时则选用极大值函数,这样计算得到的 t_1,t_3 分别为 $1+\omega$和 $1-2\omega$.总之,极值函数的使用是非常灵活的,通过在算法 RealExecute 执行过程中使用不同的极值函数,可以实现各种时间相关的测试策略.

Table 1 The execution process of transition sequences of *TS*

表 **1** *TS* 中迁移动作序列的执行过程

Transition sequences	$\mathrm{Max}\left(\sum_{i=k+1}^{k'-1}t_i\right)$	$\mathrm{Min}\left(\sum_{i=k+1}^{k'-1}t_i\right)$
$(t_1?a)(t_7!b)$	$(2-\omega)?a(2)!b$	$(1+\omega)?a(2)!b$
$(t_1?a)(t_2!c)(t_3?a)(t_4!c)$	$(2-\omega)?a(1+\omega/2)!b(0)?a(0)!c$	$(1+\omega)?a(1+\omega)!b(0)?a(0)!c$
$(t_1?a)(t_2!c)(t_5?a)(t_6!b)$	$(2-\omega)?a(1+\omega/2)!b(1-\omega)?a(\omega/2)!b$	$(1+\omega)?a(1+\omega)!b(1-2\omega)?a(\omega)!b$
$(t_1?a)(t_2!c)(t_3?a)(t_8!b)$	$(2-\omega)?a(1+\omega/2)!b(0)?a(1-\omega/2)!b$	$(1+\omega)?a(1+\omega)!b(0)?a(1-\omega)!b$

[****] 把一个问题的数据输入计算机所需要的二进制代码的长度称为输入长度.

[*****] 式中,k 和 k'的值等于 RealExecute 执行第(6)步时 k 和 k'的值.

4 结束语

本文利用 TSIOA 模型来描述实时系统,并在该模型的基础上提出了一种新的实时系统测试方法.相对于其他实时系统测试方法而言,该方法采用改进的符号状态拆分算法和抽象时间迁移去除技术,获得了更加简洁且满足稳定性的 USTGSS,从而可以将时间测试用例中的迁移动作和时间延迟量分开处理,其中:前者是在 USTGSS 基础上,利用输入/输出标号迁移系统静态测试方法获得;后者通过建立线性约束系统和引入时间极值函数在测试过程中动态求解得到.利用这种方法,可以根据系统的 TSIOA 模型,很方便地生成满足各种结构覆盖和时间延迟极值覆盖标准的测试用例集合.因此,本文所提出的测试方法对各类实时系统的一致性测试是十分有效的.

下一步的工作将考虑测试系统在动态测试执行过程中求解时间延迟变量的计算时延,以便对被测系统实施更加精确的测试.另外,在一个复杂的 TSIOA 模型中,由于各个自动机之间的并发会导致状态爆炸;在并发执行环境下如何更好地描述和执行测试用例,也是我们下一步工作的重点.

References:

[1] Wegener J, Sthamer H, Jones BF, Eyres DE. Testing real-time systems using genetic algorithms. Software Quality Journal, 1997(6): 127−135.

[2] Chan WYL, Vuong CT, Otp MR. An improved protocol test generation procedure based on UIOS. In: Landweber LH, ed. Symp. Proc. on Communications Architectures & Protocols. New York: ACM Press, 1989. 283−294.

[3] Aho AV, Dahbura AT, Lee D, Uyar MU. An optimization technique for protocol conformance test sequence generation based on UIO sequence and rural Chinese postman tour. IEEE Trans. on Communications, 1991,39(11):1604−1615.

[4] Sidhu DP, Leung T. Formal methods for protocol testing: A detailed study. IEEE Trans. on Software Engineering, 1989,15(4): 413−426.

[5] Chow TS. Testing software design modeled by finite-state machines. IEEE Trans. on Software Engineering, 1978,4(3):178−187.

[6] Fujiwara S, Bochmann GV. Test selection based on finite state models. IEEE Trans. on Software Engineering, 1991,17(6):591−603.

[7] Lamport L. The temporal logic of actions. ACM Trans. on Programming Language and Systems, 1994,16(3):872−923.

[8] Alur R, Dill D. A theory of timed automata. Theoretical Computer Science, 1994,126(2):183−235.

[9] Alur R. Timed automata. In: Halbwachs N, Peled D, eds. Proc. of the 11th Int'l Conf. on Computer-Aided Verification. LNCS 1633, London: Springer-Verlag, 1999. 8−22.

[10] Kaynar DK, Lynch N, Segala R, Vaandrager F. Timed I/O automata: A mathematical framework for modeling and analyzing real-time systems. In: Kaynar DK, Lynch N, Segala R, Vaandrager F, eds. Proc. of the 24th IEEE Int'l Real-Time Systems Symp. Washington: IEEE Computer Society, 2003. 166−177.

[11] Henzinger TA, Manna Z, Pnueli A. Timed transition system. In: Bakker JWD, Huizing C, Roever WPD, Rozenberg G, eds. Proc. of the Real-Time: Theory in Practice, REX Workshop. LNCS 600, Berlin: Springer-Verlag, 1992. 226−251.

[12] Mandrioli D, Morasca S, Morzenti A. Generating test cases for real-time systems from logic specifications. ACM Trans. on Computer Systems, 1995,13(4):365−398.

[13] Cardell-Oliver R, Glover T. A practical and complete algorithm for testing real-time systems. In: Ravn AP, Rischel H, eds. Proc. of the 5th Int'l Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems. LNCS 1486, Berlin: Springer-Verlag, 1998. 251−261.

[14] Springintveld J, Vadranger F, Dargenio P. Testing timed automata. Theoretical Computer Science, 2001,254(1-2):225−257.

[15] En-Nouaary A, Dssouli R, Khendek F. Timed Wp-method: Testing real-time systems. IEEE Trans. on Software Engineering, 2002, 28(11):1023−1038.

[16] Krichen M, Tripakis S. State identification problems for timed automata. In: Khendek F, Dssouli R, eds. Proc. of the 17th IFIP Int'l Conf. on Testing of Communicating Systems (TestCom 2005). LNCS 3502, Berlin: Springer-Verlag, 2005. 175−191.

[17] Tripakis S, Yovine S. Analysis of timed systems using time-abstracting bisimulations. Formal Methods in System Design, 2001, 18(1):25−68.

[18] Bouajjani A, Fernandez JC, Halbwachs N, Raymond P, Ratel C. Minimal state graph generation. Science of Computer Programming, 1992,18(3):247−269.

[19] Alur R, Courcoubetis C, Dill DL, Halbwachs N, Wong-Toi H. Minimization of timed transition systems. In: Cleaveland R, ed. Proc. of the 3rd Conf. on Concurrency Theory. LNCS 630, Berlin: Springer-Verlag, 1992. 340−354.

[20] Larsen KG, Yi W. Time abstracted bisimiulation: Implicit specifications and decidability. In: Brookes SD, Main MG, Melton A, Mislove MW, Schmidt DA, eds. Proc. of the 9th Int'l Conf. on Mathematical Foundation of Programming Semantics. LNCS 802, Berlin: Springer-Verlag, 1993. 160−176.

[21] Henzinger TA, Nicollin X, Sifakis J, Yovine S. Symbolic model checking for real-time systems. Information and Computation, 1994,111(2):193−244.

[22] Bengtsson J, Yi W. Timed automata: Semantics, algorithms and tools. In: Desel J, Reisig W, Rozenberg G, eds. Lectures on Concurrency and Petri Nets 2003. LNCS 3098, Berlin: Springer-Verlag, 2004. 87−124.

[23] Hessel A, Larsen KG, Nielsen B, Pettersson P, Skou A. Time-Optimal test cases for real-time systems. In: Larsen KG, Niebert P, eds. Proc. of the 1st Int'l Workshop, FORMATS 2003. LNCS 2791, Berlin: Springer-Verlag, 2003. 234−245.

[24] Larsen KG, Mikucionis M, Nielsen B. Online testing of real-time systems using uppaal. In: Grabowski J, Nielsen B, eds. Formal Approaches to Software Testing: 4th Int'l Workshop. LNCS 3395, Linz: Springer-Verlag, 2005. 79−94.

[25] Tretmans J. Conformance testing with labelled transition systems: Implementation relations and test generation. Computer Networks and ISDN Systems, 1996,29(1):49−79.

[26] Vries DRG, Tretmans J. On-the-Fly conformance testing using Spin. Software Tools for Technology Transfer, 2000,2(4):382−393.

[27] Rusu V, Bousquet LD, Jeron T. An approach to symbolic test generation. In: Grieskamp W, Santen T, Stoddart B, eds. Proc. of the 2nd Int'l Conf. on Integerated Formal Methods. LNCS 1945, Springer-Verlag, 2000. 338−357.

[28] Jorgenson PC. Software Testing: A Craftsman's Approach. 2nd ed., Beijing: China Machine Press, 2003. 70−140 (in Chinese).

[29] En-Nouaary A, Dssouli R. A guided method for testing timed input output automata. In: Hogrefe D, Wiles A, eds. Proc. of the Conf. on Testing Communication Systems, TestCom 2003. LNCS 2644, Berlin: Springer-Verlag, 2003. 211−225.

[30] Karmarkar N. A new polynomial-time algorithm for linear programming. In: Demillo R, ed. Proc. of the 16th Annual ACM Symp. on Theory of Computing. New York: ACM Press, 1984. 302−311.

附中文参考文献:

[28] Jorgensen PC.软件测试.第 2 版.北京:机械工业出版社,2003.

陈伟(1977－),男,安徽马鞍山人,博士生,主要研究领域为软件测试方法和工具,编译技术及应用.

赵琛(1967－),男,博士,研究员,CCF 高级会员,主要研究领域为编译技术及应用,软件测试方法和工具.

薛云志(1979－),男,博士生,主要研究领域为软件测试,并发系统建模与测试.

李明树(1966－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为智能软件工程,实时系统.