计算机科学2004Vol.31№.6

程序时序属性的自动测试

马晓东 董 威 王 戟 齐治昌
(国防科学技术大学计算机学院 长沙410073)

摘 要 测试预言是一种用来检测被测系统的测试执行是否正确的方法。文中,作者设计并实现了一种根据程序的线性时序逻辑(LTL)的性质产生测试预言的方法。首先,作者将一线性时序逻辑公式转换为一个有限状态自动机,然后,管理源代码,以便抽取与线性时序逻辑性质有关的状态序列。最后,用该信息来模拟状态自动机,并决定程序执行是否满足线性时序逻辑的性质。

关键词 测试预言,线性时序逻辑,FSA

Testing of Temporal Properties of Program

MA XIAO-Dong DONG Wei WANG Ji QI Zhi-Chang
(College of Computer Sicence,National University of Defense Technolgy,Changsha 410073)

Abstract Test oracle is a method of checking whether the system under test has behaved correctly on a particular execution. In this paper, we design and implement a method of producing test oracle from program's LTL(Linear Temporal Logic) property. First, we transfer a LTL formula to a finite state automaton; then, we manage the source code in order to extract state sequence which is relative to the LTL property; in the end, we use this information to stimulate the automaton and decide whether the execution of the program satisfies the LTL property.

Keywords Test oracles, LTL, FSA

1 引言

反应式系统通常由一系列与环境交互的行为组成,通过反应动作来响应外界的激励。这种连续的交互特性使得它与传统的顺序程序不同,后者通常把某一输入转换为输出;它们的另一个不同在于反应式系统通常是并发执行的,而并发系统中所存在的不确定性使得对它属性的描述和测试都变得更加困难。因此,对于反应式系统而言,常用系统中事件发生的相对顺序来描述系统属性。有多种逻辑可以描述这样的属性,而使用最广泛的是由 Pnueli 于1977年引入的时序逻辑(Temporal Logics)。时序逻辑主要分为两类:LTL(Linear Temporal Logic)和 CTL(Compute Tree Logic),它们的表达能力是相交的,没有包含与被包含的关系[1]。时序逻辑提供了对于系统在时间顺序上的行为的公式化描述,使得我们能够利用公式来准确地刻画系统的行为。对于反应式系统而言,测试它的时序逻辑属性很重要,本文主要提出了一种测试系统的 LTL 公式属性的方法及其实现。

测试预言(Test oracle)是用来判断被测程序的测试执行是否正确的方法,判断标准就是描述程序性质的规范。测试预言要做到对规范中的约束的覆盖,但又不能施加额外的约束,而且还要做到高效率的检验。对测试预言的这些要求是相互冲突的,这就要在其实现时做出折衷。理想的测试预言应该能够依据对程序预期行为的自然描述准确地判断程序执行的正确与否。

测试预言的种类有很多。有的通过嵌入式断言来实现,该方法把所期望的表达式直接插入程序源代码中去。典型的嵌入式断言可以在程序的任何控制点放置待检测的属性,C 语言中的 assert 宏可以看作嵌入式断言的原型。有的测试预言是基于规范描述文档,该方式下程序的规范描述与实现分离,这可以使得程序的规范描述语言与程序的实现语言之间的联系更松散,或者完全与程序的实现语言无关[2]。本文中实现的是对 LTL 公式所描述的性质进行判断的一种测试预言,可以用于测试并发反应式系统。

2 测试预言框架

测试预言的种类不同,其实现框架也不相同。对于嵌入式断言系统,在实现时要考虑的主要有下列几个问题:首先,嵌入式断言的通常做法是在特定的位置插入一个表达式,但是对于那些与控制点无关的全局属性,则不是在程序的所有控制点都插入表达式,而是通过为整个过程加上前置条件或后置条件来实现;其次,需要状态缓存,某些过程的后置条件要用到执行该过程之前的程序状态或变量的值,这就要求对这些信息保存一个副本供以后使用;再次,为了保存副本,需要使用一些原本不存在的辅助变量;最后,对于量词的处理,通常使用循环的方法,但是该方法并不总是有效。对于规范描述文档和程序分离的情况,首先要对规范文档进行分析,从文档中生成可执行的判断标准;然后处理待测试系统的源代码,或者对其进行运行时的监控,提取状态信息,实现测试预言[2]。

马晓东 硕士研究生,研究方向为软件测试与验证;董 威 博士,研究方向为软件可靠性与软件工程;王 戟 教授,研究方向为软件理论与软件工程;齐治昌 教授,研究方向为软件理论与软件工程。

14 (美)普瑞斯曼(Pressman,R.S.)著,黄柏素,梅宏译. 软件工程-实践者的研究方法. 北京:机械工业出版社,1999
15 毛新军,王怀民,齐治昌. 基于 Agent 技术的软件重用模式. 计算机科学,2001,28(3):90~92
16 魏晓斌,Unland R,Bacmendo B. 一个基于 XML 的 Agent 通信框架. 计算机应用研究,2001,10:108~112

本文的目的是测试程序的时序逻辑性质。LTL 公式是解释在无穷状态序列上的，但是，这样的解释就需要功能更强大的自动机，例如 ω-自动机。对于产生无穷状态序列的程序——即不终止的程序，要判定其是否符合某一性质，需对其进行结构上的分析，并加以证明，这要用到模型检验或者是形式化验证方面的技术。通常在进行软件测试时，要么待测程序是终止的，要么是截取不终止程序的一个有限的状态前序。本文使用了文[3]中从 LTL 公式生成可接受有穷状态序列自动机的方法，生成的测试预言可使被测程序输出一个状态序列，并对该序列进行测试，其实现框架如图1所示。

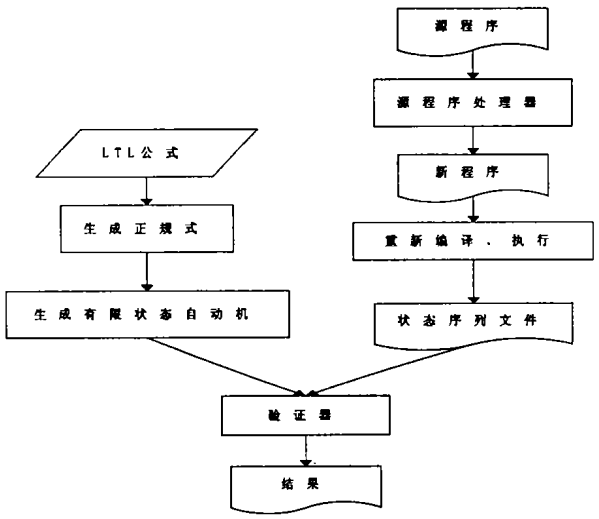

图1 测试预言的实现框架

从图1中可知，该测试预言的实现主要分为三大部分：一是对于 LTL 公式的处理，先把该公式转换为取非正规式(NNF:Negation Normal Form)——下面简称正规式，再把它变成一个接受有限状态序列的自动机；二是对于源程序的处理，通过在适当的地方插入输出语句，产生一个新的程序，对新程序进行编译，使它在执行过程中不断输出与被测属性相关的变量，得到相应的状态序列；三是结果的判定，判别输出的状态序列能否为自动机接受，从而得出程序的本次测试执行是否满足对应的 LTL 公式。

3 基于时序性质的测试预言实现

本节将具体阐述图1提出的时序性质测试预言框架的实现方法。

3.1 LTL 在有限状态序列上的语义和正规式

为了判断 LTL 公式在有限状态序列上的正确性，首先要定义公式在这样的序列上的语义[3]。本文采用的操作符有!，&，|，⇒，G，F，X，N，U 和 P，其中!，&，|，⇒是命题逻辑中常见的取非、合取、析取和蕴含，其语义与时序属性无关。其他操作符的语义如表1所示，其中 S 表示状态序列，(S,i) 表示该状态序列上的第 i 个状态，状态的编号从1开始。|S|表示状态序列 S 中的状态个数，符号|=表示满足关系。

其中，有下列等价关系成立：

$Ff \equiv$ true U f；$Gf \equiv !(F!(f))$；$!(f_1 U f_2) \equiv (!(f_1))P f_2$；$!Xf = X(!f)$；$!Nf = N(!f)$。

一个公式为正规式当且仅当其所有的取非操作都仅施加于原子命题上。例如，公式 G(!(F(b))|G(a)) 的正规式为 (false)P ((trueUb)&(trueU!a))。在从公式转换为自动机的过程中，把公式化为正规式不是必需的，但是很有帮助，否则，在进行公式分解时会导致许多额外操作。表1中的 N 和 X 操作符都是用来描述下一个状态，但因为状态序列是有穷的，所以使用了两个操作符。其中 X 表示必须有下一个状态，而 N 表示可以没有下一个状态。

根据这些语义规则，可以得出判别时序公式在有穷状态序列上的正确性标准；而从上述等价关系出发，再加上命题逻辑中!，&，|，⇒操作符的一些转换规则，可以把公式化为正规式。

表1 LTL 操作符在有穷状态序列上的语义

操作符	语义
N	$(S,i) \models Nf$ 当且仅当 $\|S\|>i \Rightarrow (S,i+1) \models f$
X	$(S,i) \models Xf$ 当且仅当 $\|S\|>i$ 且 $(S,i+1) \models f$
G	$(S,i) \models Gf$ 当且仅当对于所有的 j，若 $i \leqslant j \leqslant \|S\|$，则 $(S,j) \models f$
F	$(S,i) \models Ff$ 当且仅当存在某个 j，$i \leqslant j \leqslant \|S\|$ 且 $(S,j) \models f$
U	$(S,i) \models f_1 U f_2$ 当且仅当存在某个 k，$i \leqslant k \leqslant \|S\|$ 且 $(S,k) \models f_2$，并且对于所有的 j，若 $i \leqslant j < k$，则 $(S,j) \models f_1$
P	$(S,i) \models f_1 P f_2$ 当且仅当对于所有的 k，若 $i \leqslant k \leqslant \|S\|$ 并且 $(S,k) \models f_2$，则存在某个 j，$i \leqslant j < k$ 且 $(S,j) \models f_1$ 。

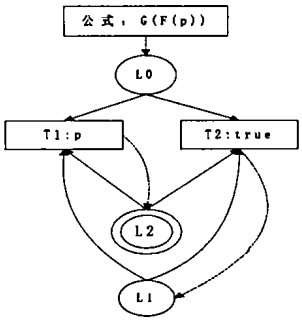

图2 一个自动机实例

3.2 有限状态自动机

如前所述，解释在无穷状态序列上的 LTL 公式可以用 ω-自动机来表示；因这里把 LTL 公式限制在有穷序列上，就要求不同的自动机，即接受有限字的有穷状态自动机。图2是本文中使用的一个自动机的实例。这里所用的自动机是一个七元组[3]。如下：

$A = (Alph, Locs, Trans, Lab, Flow, Init, Fin)$。其中：

Alph：有限输入字符表，即是一些原子命题。

Locs：一个状态集合，每个状态中包含该状态下要满足的时序逻辑公式集合和自身是否为终止状态的信息。图中用圆圈表示。

Trans：一个有限的迁移集合。迁移包含迁移条件和一系列的公式。若满足迁移条件，则可以从自动机中的一个状态到达另一个状态。图中用方框表示。

表2 分解规则表

操作符	分解规则
&	$f_1 \& f_2 \rightarrow (\{f_1, f_2\})$
|	$f_1 \| f_2 \rightarrow (\{f_1\}, \{f_2\})$
⇒	$f_1 \Rightarrow f_2 \rightarrow (\{!f_1\}, \{f_2\})$
G	$Gf \rightarrow (\{f, NGf\})$
F	$Ff \rightarrow (\{f\}, \{XFf\})$
U	$f_1 U f_2 \rightarrow (\{f_2\}, \{f_1, X(f_1 U f_2)\})$
P	$f_1 P f_2 \rightarrow (\{f_1, !f_2\}, \{!f_2, N(f_1 P f_2)\})$

Lab：给迁移标记相应条件的函数。

Flow：流关系，指明迁移如何连接两个状态。$Flow \subseteq (Locs * Trans) \cup (Trans * Locs)$。

Init：一个初始状态，一般用标号为0的状态来表示，如图2中的 *L*0。

Fin：终止状态集合。图中用两层的圆圈表示。

为了能够从 LTL 公式生成自动机，还需要一些分解规则，表2为一个分解规则表。分解规则的作用将在后面说明。自动机的生成的过程可以描述如下：

1)运用3.1中的等价关系，把公式中的取非操作符都转换到原子命题之前，使公式变为正规式。

2)把待测的正规式作为初始状态下要满足的公式，也把它作为与初始状态相连接的迁移中的公式。

3)对迁移中的公式运用分解规则进行分解，直到迁移中的所有公式仅包含下列几种形式：原子命题；原子命题之非；最外层操作符为 N 的公式；最外层操作符为 X 的公式。只包含这四种情况的公式的迁移被称为真迁移。

4)迁移分解完成之后，建立状态到迁移的连接，再根据迁移中所包含的公式生成新的状态，再根据状态生成迁移，再分解，重复操作，直到不再生成新的迁移和状态为止。

可以利用图2中的例子来详细说明。首先生成一个初始状态 L0，它所要满足的公式为{GF(p)}；同时生成一个迁移 T0，该迁移中的公式也为{GF(p)}，它不是真迁移，要对其进行分解。根据表2，该迁移可使用 G 规则分解为{F(p)，NGF(p)}；再使用 F 规则分解为 T1＝{p，NGF(p)}和 T2＝{XF(p)，NGF(p)}，它们是真迁移。取每个迁移中最外层操作符不是 X 或 N 的公式作条件，则它们的条件为分别为 p 和 true。由 T1生成后继状态 L2，它所要满足的公式为{GF(p)}，和 L0中的公式相同，但 T1中的操作符 N 表示可以没有下一个状态，所以 L2是终结状态。由 T2生成后继状态 L1，它所要满足的公式为{F(p)，GF(p)}，因为 T2中的 X 操作符表示必须有下一个状态，所以，L1不能是终止状态。按照这种算法循环下去，就可以得到图2。

该算法的正确性和可终止性证明见文[3]。

在对迁移中公式进行分解的过程中，如不使用任何优化规则，则生成的自动机可能会非常大，冗余很多。在分解公式的过程中，我们使用了下述几条简单的规则[3]：

1)迁移包含的公式集合中有 $f_1 \vee f_2$ 和 f_1，则从集合中删去 $f_1 \vee f_2$；

2)若迁移包含的公式集合中有 $f_1 \vee f_2$ 和 f_2，则从集合中删去 $f_1 \vee f_2$；

3)若迁移包含的公式集合中有 X*f* 和 NG*f*，则把 X*f* 去掉，换为 X(true)。

4)在分解的时候，先分解比较长的公式，再分解短的。

经实验证明，这几条简单的规则很有效，图3是对公式 G(p)使用优化规则前后所产生的自动机的比较。总的来说，对生成的自动机进行优化需要一个定理证明器，要做到尽可能的优化是一个非常困难的工作。

3.3 自动机上字的判定

由上一节可以看出，从 LTL 公式所生成的是一个不确定的自动机。要判断一个字——在这里是状态序列——是否满足该自动机，需要用该状态序列走遍自动机上所有可能的路径。我们使用宽度优先搜索来判断状态序列是否满足自动机。算法描述如下：

数据结构：状态队列 Present，Next；迁移队列 Trans。

```
Present.Add(L0);
Num=GetStateSequenceSize( );
for(int Index=0; Index<Num; Index++)
  for(int P=0; P<Present.GetSize( ); P++)
    Trsition=Present.GetAt(P).GetAllConnectTrans()
    if(!HasExist(Trans, Trsition))
      GetConditionValue(Trsition)
      Trans.Add(Trsition)
    Endif
  Endfor //生成与 Present 相连的转移 Trans
  for(int T=0; T<Trans.GetSize();T++)
    Loca=Trans.GetAt(T).GetConnectLoca();
    if(!HasExist(Next, Loca))
      Next.Add(Loca);
    Endif
  Endfor //生成与转移 Trans 相连的状态 Next
  if(Next.IsEmpty( ))
    return false;//没有路径可走，说明不满足属性
  Endif
  Present.RemoveAll( );
  Trans.RemoveAll( );
  Present=Next;//Next 赋给 Present
Endfor
if(Present.HasFinalState( ))
  return true;//满足属性
else
  return false;//不满足属性
endif
```

在上述宽度优先的搜索算法中，状态序列的长短是影响算法时间消耗的关键因素。通过实验，一万多个状态的状态序列可以在几秒钟内完成计算。

在做状态序列是否满足自动机的判断时，有两种方式：在线测试和离线测试。在线测试是指待测程序和测试预言分别在两个进程中运行。待测程序运行时，每发生一次与 LTL 公式中变量相关的状态变化，则生成一个新的状态，把结果传送给测试预言。测试预言接到新状态后，读入该状态，控制自动机向前走一步。这种测试方式可以在程序的运行时及时发现错误，避免可能发生的严重后果，但是它给测试预言的实现带来了困难，需要把待测程序作为一个新的进程加入整个测试程序中，并且要控制两个进程之间的同步执行。离线方式是本文中实现的方式，该方式下待测程序先执行，生成状态序列文件，测试预言在待测程序执行完之后再执行，读入状态序列，判断该状态序列是否满足自动机。

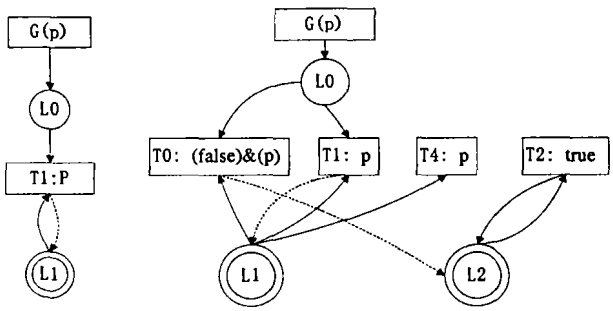

图3 优化后(左图)和未优化(右图)的自动机的比较

3.4 程序中状态序列的提取

因为要测试程序对于某条 LTL 属性的满足性，所以，可以忽略与 LTL 公式中变量无关的那些程序状态的变化，只提取与公式中变量相关的状态。我们把任何一个相关变量值的一次改变视为一次状态变化。

首先，对 LTL 公式进行分析，提出原子命题中所包含的变量，保存到变量列表中。然后对源程序的文本进行分析，找出程序中变量列表中的变量发生改变的地方，在变量改变的位置插入输出语句，把其值输出到一个文件中去，就组成状态序列变化文件。读该状态变化文件，即可判断程序本次测试执

(下转第179页)

N 算法：

s0. 设 $R^{(0)}=(r_{ij}^{(0)})$为 $n\times n$ 阶模糊相似矩阵，给定初始聚类个数 c。
s1. 计算 max-t 相似关系矩阵 $R=(r_{ij})$。
s2. 计算 $b_{ij}=1-r_{ij}$得到 $U=B=(b_{ij})_{n\times n}$矩阵。
s3. 计算 $Q=\sum_{i,j=1}^{n} b_{ij}$。从矩阵 B 的第 i 行找出最大元 mxb_i，令 $mb=\max\{mxb_i\}(i=1,2,\cdots,n)$ $d=mb/(c+1)$
s4. for $1\leqslant i\leqslant n$ and $1\leqslant j\leqslant n$ do
  if $b_{ij}<d$ then $b_{ij}=1$
  else
  $b_{ij}=0$
 enddo
s5. for $1\leqslant i\leqslant n-1$ do
 if $b_{ij}=0$ then $i=i+1$
 for $i+1\leqslant j\leqslant n$ do
  if $b_{ij}\neq 0$ then
  第 j 行加到第 i 行
 enddo
 for $i+1\leqslant j\leqslant n$ do
  if $b_{ij}\neq 0$ then
  第 j 列加到第 i 列
 enddo
 for $i+1\leqslant j\leqslant n$ do
  if $b_{ij}\neq 0$ then
  把 j 归入 i 所在的一类.
  enddo
 置第 i 行与第 j 列所有元素为0。
 enddo
 计算各聚类中心。
s6. 若聚类数小于 c 则 $d=d/2$转步骤 s4，否则合并聚类中心为 c 个。
s8. 令 K,Z_k 分别表示聚类中心的标号和属于 k 聚类中心的元素标号。计算：

$$Q_0=\frac{\sum_{k}\sum_{j\in Z_k} b_{kj}}{\sum_{i\in K,j\in S} b_{ij}}.$$

若 $Q_0<Q$ 则取 $Q=Q_0$转步骤 s4。
列出对应各聚类中心的所有元素。结束。

应用该算法对上文所列文[3]中给出的10×10模糊相似矩阵 $R^{(0)}$进行了聚类计算，分类的结果是{1,2,4,6,8}，{5,9,10}，{3}，{7}。同时，这种分类的结果具有唯一性。

4 不同范数下的 max-t 构成

基于 max-t_3范数的传递闭包聚类算法与基于 max-t_1范数的聚类算法共同的基本计算结果是依据 max-t 构成产生 max-t 相似关系矩阵。其过程如下：

$$r_{ij}^{(1)}=\max_{k_1} t(r_{ik_1}^{(0)},r_{k_1j}^{(0)})$$

$$r_{ij}^{(2)}=\max_{k_2} t(r_{ik_2}^{(1)},r_{k_2j}^{(1)})=\max_{k_1,k_2} t(t(r_{ik_1}^{(0)},r_{k_1k_2}^{(0)}),r_{k_2j}^{(0)})$$

…

$$r_{ij}^{(n)}=\max_{k_1,\cdots,k_n} t(\cdots(t(r_{ik_1}^{(0)},r_{k_1k_2}^{(0)}),r_{k_2k_3}^{(0)}),\cdots,r_{k_nj}^{(0)})$$

把相似关系矩阵 $R^{(0)}=(r_{ij}^{(0)})$的元素看作模糊图的模糊权值，则上述过程就是对任意两顶间不同路长上权值的一种度量方法：$r_{ij}^{(1)}$是在 t 范数约束下两顶 i 与 j 间路长为2的最大值，…，$r_{ij}^{(k)}$是在 t 范数约束下两顶 i 与 j 间路长为 $k+1$的最大值。在 t_3范数下，max-t 相似关系矩阵元素即是顶 i 到顶 j 诸路中最小权值的最大者。而在 t_1范数下，max-t 相似关系矩阵元素即是顶 i 到顶 j 诸路权值之和减去路长个数减1后的最大值。显然由同一初始相似矩阵得到的 max-t_3相似关系矩阵比相应的 max-t_1相似关系矩阵的元素要大。以3个顶为例更能说明这个问题。设有3个顶 v_i，v_j 和 v_k，模糊权值分别为 $r_{ik}=0.8$，$r_{kj}=0.7$和 $r_{ij}=0.4$。由 max-t_3构成顶 v_i 与 v_j 的权值得到提升为0.7，而由 max-t_1构成顶 v_i 与 v_j 的权值则得到提升为0.5。可见，基于 max-t_3范数的聚类的粗糙度比基于 max-t_1范数的聚类的粗糙度要大。这就是为什么在文[3]中使用基于 max-t_1构成恢复不完全数据的效果比使用 max-t_3构成要更好的原因。

结论 文章对传递闭包聚类算法和基于 max-t_1范数的聚类算法做了一定的理论探讨，同时针对基于 max-t_1范数的聚类算法的不足提出了一种新的算法。基于不同的范数实际上就是选择不同的聚类尺度和确定数据间的联系度。这方面的研究从另一角度揭示聚类的意义。

参 考 文 献

1 Zadeh L A. Fuzzy sets. Inform. and Control, 1965, 8: 338～353
2 Tamura S, Higuchi S, Tanaka K. Pattern classification based on fuzzy relations. IEEE Trans. Systems Man Cybernet, 1978, 1: 61～66
3 Yang M-S, Shih H-M. Cluster analysis based on fuzzy relations. Fuzzy Sets and Systems, 2001, 120: 197～212
4 Lee H S. An optimal algorithm for computing the max-min transitive closure of a fuzzy similarity matrix. Fuzzy Sets and Systems, 2001, 123: 129～136
5 Zimmermann H J. Fuzzy set theory and its applications. Kluwer, Dordrecht, 1991

（上接第134页）

行的正确性。

结束语 测试预言是软件测试中一个重要部分，一个自动化、高效的测试预言可以大大提高软件测试的效率，缩短软件的开发进程。

本文中实现的测试预言判断程序在测试过程中是否满足时序逻辑属性，是模型检验和软件测试的结合点。今后对该工具的改进工作包括下述几个方面：

1）研究在公式分解时可以采用的优化规则，进一步改进生成的自动机，使得可以生成更小的、更优化的自动机。

2）改进状态提取方法，使状态的提取更为精确。

3）增强对源程序的文本的分析能力，使得能够处理更多的数据类型，进一步支持较为复杂的程序结构。

4）实现运行时监控功能。本文中采用的是宽度优先算法，它可以每次读入一个新状态，控制自动机向前走一步。因此，实现运行时监控在算法上是没有问题的，需要考虑如何把待测程序作为新的线程或进程加入整个测试程序之中，并用适当的方法控制不同进程之间执行的同步。

参 考 文 献

1 Katoen J-P. Concepts, Algorit- hms, and Tools for Model Checking. Friedrich -Alexander Universit?t Erlangen-Nürnberg, Lecture Notes of the Course "Mechanised Validation of Parallel Systems"(course number 10359) Semester 1998/1999. 47～48
2 Baresi L, Young M. Test Ora- cles: [Technical Report CIS-TR-01-02]. Aug. 2001. 6～7, 20
3 Dillon L D, Ramakrishna Y S. In: Proc. 4th ACM SIGSOFT Symp. Foundations of Software Engineering, San Francisco, Oct. 1996. 106～117