基于Petri网的并发程序测试用例产生方法

霍敏霞[1] 丁晓明[1,2]
(西南大学计算机与信息科学学院 重庆 400715)[1]
(重庆市智能软件与软件工程重点实验室 重庆 400715)[2]

摘 要 并发程序的测试路径具有不可预测性,而Petri网在描述并发方面具有其它系统模型无法比拟的优势。因此通过Petri网来产生并发程序的测试路径:对有并发程序的源代码构造的Petri网模型进行图形矩阵转换;按照一定的规则得出相应的独立段组;合并独立段组得出网的独立段群,此独立段群即为该并发程序的测试路径。实验证明,将Petri网用于并发程序测试用的例生成降低了测试难度,提高了测试效率。

关键词 并发程序,Petri网,独立段群
中图法分类号 TP311 **文献标识码** A

Test Case Generation Method for Concurrent Programs Based on Petri Net

HUO Min-xia[1] DING Xiao-ming[1,2]
(College of Computer and Information Science, Southwest University, Chongqing 400715, China)[1]
(The Key Lab of Intelligent Software and Software Engineering, Chongqing 400715, China)[2]

Abstract Petri nets have an incomparable advantage of describing the unpredictable testing path of concurrent program. This article made test path of concurrent programs with Petri nets, transformed graphic matrix about Petri net model which is made of concurrent program code, found the corresponding independent segment group according to certain rules, and obtained the independent segment groups of Petri nets which is the test path of the concurrent programs, by merging independent segment group. Experiment shows that the using of Petri nets in concurrent programs testing reduces the test difficulty and improves the test efficiency in concurrent programs testing.

Keywords Concurrent programs, Petri nets, Independent segment groups

1 引言

目前,并发程序的测试已日益成为人们关注和重视的研究领域。但是,由于并发程序自身的一些特性:失去程序的封闭性和可再现性、程序与计算不再一一对应和并发执行的相互制约,使得并发程序的执行效果取决于程序中各独立事件发生的先后次序,因此测试的结果具有不确定性。针对这种不确定性,很多学者提出了相应的测试用例生成算法。

W. E. Wong 提出针对可达图(RG)的测试用例生成算法,并通过实验指出该算法具有较高的效率[1]。D. L. Long 为组成并发程序的每个任务构造任务交互图(TIG),以分析程序的并发行为[2]。C. Demartini 针对并发 Java 程序构造约减的 Java 控制流图(JRCFG),并根据 JRCFG 构造 Promela 模型,然后借助模型检验工具 SPIN 对并发 Java 程序行为进行分析和验证[3]。T. Katayama 构造事件交互图(EIAG),并给出了基于 EIAG 的并发程序测试用例生成算法[4]。

基于 Petri 网的并发程序相关研究有:O. Herzog 最早使用 Petri 网对并发程序进行建模以分析其具有的属性[5];其后,T. Murata 通过计算 Petri 网可达图来分析并发程序中可能存在的死锁[6,7];J. Cheng 和 A. T. Chamillard 在建模时对 Petri 网进行了扩展,使得分析的准确性进一步提高[8,9];蒋昌俊通过研究Petri网的弱活特征得出有界 Petri 网活性的充要条件,从而给出了并发系统活性测试的形式化方法[10];范洪达和叶文通过研究 Time Petri 网在实时软件中的应用就 Time Petri 网在实时软件测试中有关时间的处理上提出了大体框架[11];林红昌等人运用 Petri 网来描述顺序程序,并按照一定的数学规则分解 Petri 网,得到独立的段组并产生测试用例[12]。

在众多的研究中,尚没有关于 Petri 网在并发程序测试用例生成方面的应用。本文根据 Petri 网产生顺序程序测试用例的思想,对其方法进行改进,提出一个适用于并发程序的测试用例生成算法。构造测试用例通常需要对被测程序的结构进行一定的分析。当前主流的分析方法可分为静态分析和动态分析两大类。静态分析直接分析程序源代码,而动态分析则先记录程序的执行过程,然后对记录的结果进行分析。静态分析方法可进一步划分为控制流分析(并发分析)和数据流分析(依赖分析)两种。本文主要围绕控制流分析方法展开讨论。

到稿日期:2010-10-15 返修日期:2010-12-30
霍敏霞(1983—),女,硕士生,主要研究方向为软件测试,E-mail:huomin@swu.edu.cn;丁晓明 男,副教授,硕士生导师,主要研究方向为软件工程、软件测试。

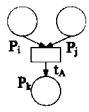
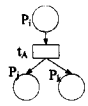
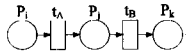
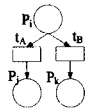
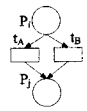
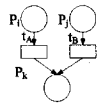
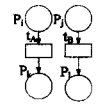
2 Petri 网概述

本文给出 Petri 网的基本介绍，详细内容参见文献[11-14]。

Petri 网是一种正式的用于实时系统建模和分析的图表模型。一个 Petri 网是一个包含有限个库所和变迁(或称为转移)的图，库所和变迁之间通过有向弧和一系列标记(或称为令牌)联系起来，弧和标记把每一个位置都联系起来。它是一种用来处理事件之间的并发、不确定以及随机联系的最优的模式。

Petri 网可以由表 1 所列的 8 种基本形式的图形段组成。

表 1 各类型的描述及图形段表示

类型	描述	图形段
1	一个变迁有两个输入库所和一个输出库所	
2	一个变迁有一个输入库所和两个输出库所	
3	两个顺序的变迁	
4	两个变迁有一个输入库所和两个输出库所	
5	两个变迁有一个输入库所和一个输出库所	
6	两个变迁有两个输入库所和一个输出库所	
7	两个变迁有各自输入库所和各自输出库所	
8	变迁的输入库所和输出库所是相同的，例如，自循环条件	

定义 1 基本段是图形段中特殊的单一的输入和单一的输出(在矩阵中对应着前/后项部分)，它的表示方法为 $[T_A]_i^j$，表示变迁 T_A 的第 i 个输入库所和第 j 个输出库所。

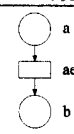
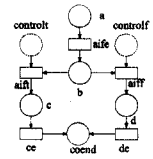
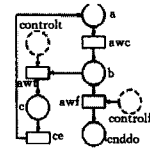
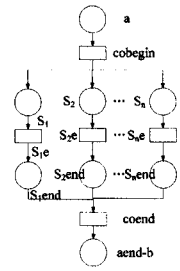
定义 2 Petri 网分解的逻辑操作包括 AND(∧)，OR(∨)和 ExclusiveOR(⊕)，优先级从高到低为{∧，⊕，∨}。

根据定义 1 和定义 2，Petri 网的 8 种基本形式的基本段表示分别为 $[t_A]_i^k \wedge [t_A]_j^k$，$[t_A]_i^j \wedge [t_A]_i^k$，$[t_A]_i^j \wedge [t_B]_j^k$，$[t_A]_i^j \oplus [t_B]_j^k$，$[t_A]_i^j \oplus [t_B]_i^j$，$[t_A]_i^k \vee [t_B]_j^k$，$[t_A]_i^k \vee [t_B]_j^j$，$[t_A]_i^j$。

3 并发程序语言及其 Petri 网描述

表 2 是有关常用的基本语言构成及其 Petri 网的对应关系。

表 2 各语句及其 Petri 网模型

语句	Petri 网模型
1. 赋值语句 a:p1=true; b:"next sentence";	
2. if 语句 a:if b then c; else d; endif;	
3. while 语句 a:while b do c; enddo;	
4. 并发语句 a:cobegin S1; Sn; ■ coend; … b; ■	

注释：例 1 中 a、b 是语句标号，分别代表各自语句。语句 a 即为 $p1$=true，语句 a、b 之间为连接关系，即执行完语句 a 后接着是执行语句 b。变迁 ae 引发表示正在执行语句 a。例 2 中库所 controlt、controlf 为一对互斥库所，即二者同时至多只有一个拥有 token，表示测试条件中的布尔表达式的真假值。例 4 中■代表进程间分割符。并发语句的语义为：当程序执行 cobegin 语句后，程序将并发执行 S1，…，Sn 等语句体。一般地，称 Si 为进程，而且必须在所有的并发进程全部执行后，程序才能继续执行 coend 后的语句。这里基本图形段的声明主要是基于变迁的，并且将其输入和输出库所限制为两个来获取 Petri 网的基本特征。多重输入输出库所可以用图形段重叠起来，这就意味着这些独立的图形段可以重叠起来而不影响 Petri 网的本来特性。运用这种方法，就可以将复杂的图用简单的图形段表示。

4 Petri 网在并发程序测试中的应用

首先建立并发程序的 Petri 网模型，将 Petri 网分解为图形段，把 Petri 网中库所和变迁的关系用矩阵形式表示出来。再通过数学方法将 Petri 网分解为独立的图形段组(ISG)，这些图形段组就可以用来表示程序的动态特征，得出基本的测试路径。最后，分析并剔除测试用例中的不可行路径。

4.1 建立 Petri 网模型

Petri 网模型的建立是根据并发程序的语句来实现的。库所和变迁之间的关系代表流关系，流关系是通过它们构造出来的。每个 Petri 网库所代表一条语句，语句的执行由流关系规定，所以变迁只能与库所有直接的流关系。Petri 网能够提供系统状态变迁和定时特性，充分描述并发程序的不确定现象。

4.2 Petri 网模型转化为图形矩阵

图形矩阵是一个方阵，它的大小与 Petri 网中库所的数目

相对应，每一行和列对应着一个已定义的库所，矩阵的元素对应着库所之间的变迁。矩阵的列代表 Petri 网变迁的输入集，矩阵的行代表 Petri 网变迁的输出集。

4.3 合并相关基本段

得到图形矩阵之后，我们可以从图形矩阵中得到一系列的基本段。用定义 2 中的逻辑操作合并它们。对于基本段的处理规则有以下几种。

规则 1　具有相同变迁并且以逻辑 AND 连接的基本段可进一步合并。如 $[t_A]_i^j \vee [t_B]_k^j$ 和 $[t_A]_i^j \vee [t_C]_m^j$ 有相同输入 $[t_A]_i^j$，因此可以将其进一步合并为 $[t_A]_i^j \vee [t_B]_k^j \vee [t_C]_m^j$。

规则 2　如果图形段符合类型 2，则有唯一的输入库所。当库所中存在标识时，这个变迁最终能发生。

规则 3　以逻辑 OR 连接的基本段在以后的步骤中可以忽略。

4.4 网的独立段群

定义 3　初始段是带有 token 的输入库所的基本段。

定义 4　网的独立段组 ISG 是一系列的不能同时发生的基本段。

规则 4　ISG 模型可以通过以逻辑异或结合的图形段来进一步分解，同时避免不可行基本段组在同一条 ISG 模型中。另外，使满足变迁覆盖准则的 ISG 路径最少。

按照前面的操作一步步来分解 Petri 网，然后运用规则 4 的操作就可以得出 ISG 模型。

总结前面的步骤，可以提出下面的一个分解 Petri 网到 ISG 的数学算法。这种算法代表了软件在执行过程中的动态特性。

输入：代表 Petri 网的图形矩阵

输出：基于 Petri 网的 ISG 模型

1. 开始。
2. 找出符合类型 1 的 Petri 网基本段。
3. 找出符合类型 2 的 Petri 网基本段。
4. 找出符合类型 3 的 Petri 网基本段。
5. 找出符合类型 4 的 Petri 网基本段。
6. 找出符合类型 5 的 Petri 网基本段。
7. 找出符合类型 6 的 Petri 网基本段。
8. 找出符合类型 7 的 Petri 网基本段。
9. 找出符合类型 8 的 Petri 网基本段。
10. 找出竞争的变迁对。
11. 根据初始 token 找出 Petri 网的初始段。
12. 根据规则 1 将基本段以初始段开始合并成组。
13. 根据规则 2 找出自动触发的变迁和其相应的基本段。
14. 找出 Petri 网中不可行路径的基本段组。
15. 进一步将基本段合并为组。
16. 根据规则 4 重新组织这些组成为 ISG。
17. 结束。

5 应用举例

以下代码是用 Ada 语言描述的生产者-消费者问题，它由 3 部分组成：生产者、消费者和缓存区。生产者线程生产物品，然后将物品放置在一个空缓冲区中供消费者线程消费。消费者线程从缓冲区中获得物品，然后释放缓冲区。当生产者线程生产物品时，如果没有空缓冲区可用，那么生产者线程必须等待消费者线程释放出一个空缓冲区。当消费者线程消费物品时，如果没有满的缓冲区，那么消费者线程将被阻塞，直到新的物品被生产出来。

利用上面介绍的方法步骤来描述下面一段代码。先将程序代码转化为 Petri 网，然后运用上述算法来产生测试用例。

程序代码如表 3 所列。

表 3　生产者-消费者程序代码

Task	Statements of the program	

Producer	begin
	loop
	buffer. put(x);
	exit when x=ASCII. EOT;
	end loop;
	end;	

consumer	begin
	loop
	buffer. get(y);
	exit when y=ASCII. EOT;
	end loop;
	end;	

buffer	begin loop select accept put(z: in character) do accept get(z: out character) do	terminater; end select; end loop; end;

其 Petri 网模型如图 1 所示。

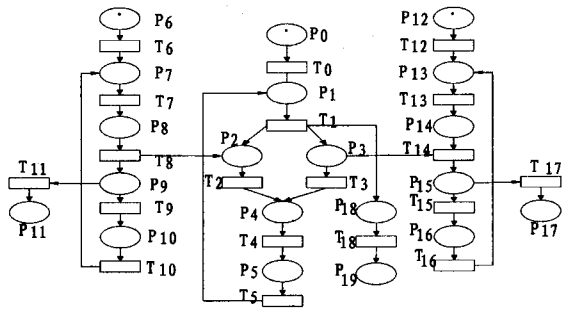


图 1　生产者-消费者 Petri 网模型

把图 1 所示的 Petri 网模型转化为图 2 所示的图形矩阵。

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19
P0	-	T0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
P1	-	-	T1	T1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	T1	-
P2	-	-	-	-	T2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
P3	-	-	-	-	T3	-	-	-	-	-	-	-	-	-	-	T14	-	-	-	-
P4	-	-	-	-	-	T4	-	-	-	-	-	-	-	-	-	-	-	-	-	-
P5	-	T5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
P6	-	-	-	-	-	-	T6	-	-	-	-	-	-	-	-	-	-	-	-	-
P7	-	-	-	-	-	-	-	T7	-	-	-	-	-	-	-	-	-	-	-	-
P8	-	-	T8	-	-	-	-	-	T8	-	-	-	-	-	-	-	-	-	-	-
P9	-	-	-	-	-	-	-	-	-	T9	T11	-	-	-	-	-	-	-	-	-
P10	-	-	-	-	-	-	-	T10	-	-	-	-	-	-	-	-	-	-	-	-
P11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
P12	-	-	-	-	-	-	-	-	-	-	-	-	T12	-	-	-	-	-	-	-
P13	-	-	-	-	-	-	-	-	-	-	-	-	-	T13	-	-	-	-	-	-
P14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	T14	-	-	-	-	-
P15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	T15	T17	-	-	-
P16	-	-	-	-	-	-	-	-	-	-	-	-	T16	-	-	-	-	-	-	-
P17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
P18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	T18
P19	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

图 2　生产者-消费者图形矩阵

图形矩阵可以表示为 $[T_0]_0^1[T_1]_1^2[T_1]_1^3[T_1]_1^{18}[T_2]_2^4[T_3]_3^4[T_4]_4^5[T_5]_5^1[T_6]_6^7[T_7]_7^8[T_8]_8^2[T_8]_8^9[T_9]_9^{10}[T_{10}]_{10}^7[T_{11}]_9^{11}[T_{12}]_{12}^{13}[T_{13}]_{13}^{14}[T_{14}]_3^{15}[T_{14}]_{14}^{15}[T_{15}]_{15}^{16}[T_{16}]_{16}^{13}[T_{17}]_{15}^{17}[T_{18}]_{18}^{18}$。在 Petri 网中，具有变迁对 $\{T_3, T_{14}\}$，$\{T_9, T_{11}\}$，$\{T_{15}, T_{17}\}$ 和自动点火变迁 $\{T_1, T_8\}$

合并相关基本段：分别找出符合以上所述 8 种基本形式的基本段进行合并。

类型 1:$[T_{14}]_3^{15} \wedge [T_{14}]_{14}^{15}$;

类型 2:$[T_1]_1^2 \wedge [T_1]_1^3$,$[T_1]_1^2 \wedge [T_1]_1^{18}$,$[T_1]_1^{18} \wedge [T_1]_1^3$,$[T_8]_8^2 \wedge [T_8]_8^9$;

类型 3:$[T_0]_0^1 \wedge [T_1]_1^2$,$[T_0]_0^1 \wedge [T_1]_1^3$,$[T_0]_0^1 \wedge [T_1]_1^{18}$,$[T_1]_1^2 \wedge [T_2]_2^4$,$[T_1]_1^3 \wedge [T_3]_3^4$,$[T_1]_1^{18} \wedge [T_{18}]_{18}^{19}$,$[T_2]_2^4 \wedge [T_4]_4^5$,$[T_1]_1^3 \wedge [T_{14}]_3^{15}$,$[T_3]_3^4 \wedge [T_4]_4^5$,$[T_4]_4^5 \wedge [T_5]_5^1$,$[T_5]_5^1 \wedge [T_1]_1^2$,$[T_5]_5^1 \wedge [T_1]_1^3$,$[T_5]_5^1 \wedge [T_1]_1^{18}$,$[T_6]_6^7 \wedge [T_7]_7^8$,$[T_6]_{10}^{11} \wedge [T_7]_{11}^{12}$,$[T_7]_7^8 \wedge [T_8]_8^2$,$[T_7]_7^8 \wedge [T_8]_8^9$,$[T_8]_8^2 \wedge [T_2]_2^4$,$[T_8]_8^9 \wedge [T_9]_9^{10}$,$[T_8]_8^9 \wedge [T_{11}]_9^{11}$,$[T_9]_9^{10} \wedge [T_{10}]_{10}^7$,$[T_{10}]_{10}^7 \wedge [T_7]_7^8$,$[T_{12}]_{12}^{13} \wedge [T_{13}]_{13}^{14}$,$[T_{13}]_{13}^{14} \wedge [T_{14}]_{14}^{15}$,$[T_{14}]_3^{15} \wedge [T_{15}]_{15}^{16}$,$[T_{14}]_3^{15} \wedge [T_{17}]_{15}^{17}$,$[T_{14}]_{14}^{15} \wedge [T_{15}]_{15}^{16}$,$[T_{15}]_{15}^{16} \wedge [T_{16}]_{16}^{13}$,$[T_{16}]_{16}^{13} \wedge [T_{13}]_{13}^{14}$;

类型 4:$[T_3]_3^4 \oplus [T_{14}]_3^{15}$,$[T_9]_9^{10} \oplus [T_{11}]_9^{11}$,$[T_{15}]_{15}^{16} \oplus [T_{17}]_{15}^{17}$

类型 5:$[T_0]_0^1 \vee [T_5]_5^1$,$[T_1]_1^2 \vee [T_8]_8^2$,$[T_2]_2^4 \vee [T_3]_3^4$,$[T_6]_6^7 \vee [T_{10}]_{10}^7$,$[T_{12}]_{12}^{13} \vee [T_{16}]_{16}^{13}$。

网的独立段群:

(1)$[T_0]_0^1 \wedge [T_1]_1^{18} \wedge [T_{18}]_{18}^{19}$

(2)$[T_0]_0^1 \wedge [T_1]_1^3 \wedge [T_3]_3^4 \wedge [T_4]_4^5 \wedge [T_5]_5^1 \wedge [T_1]_1^{18} \wedge [T_{18}]_{18}^{19}$

(3)$[T_0]_0^1 \wedge [T_1]_1^2 \wedge [T_2]_2^4 \wedge [T_4]_4^5 \wedge [T_5]_5^1 \wedge [T_1]_1^{18} \wedge [T_{18}]_{18}^{19}$

(4)$[T_0]_0^1 \wedge [T_1]_1^2 \wedge [T_2]_2^4 \wedge [T_4]_4^5 \wedge [T_5]_5^1 \wedge [T_1]_1^3 \wedge [T_3]_3^4 \wedge [T_4]_4^5 \wedge [T_5]_5^1 \wedge [T_1]_1^{18} \wedge [T_{18}]_{18}^{19}$

(5)$[T_0]_0^1 \wedge [T_1]_1^3 \wedge [T_3]_3^4 \wedge [T_4]_4^5 \wedge [T_5]_5^1 \wedge [T_1]_1^2 \wedge [T_2]_2^4 \wedge [T_4]_4^5 \wedge [T_5]_5^1 \wedge [T_1]_1^{18} \wedge [T_{18}]_{18}^{19}$

(6)$[T_0]_0^1 \wedge [T_1]_1^3 \wedge [T_{14}]_3^{15} \wedge [T_{17}]_{15}^{17}$

(7)$[T_0]_0^1 \wedge [T_1]_1^3 \wedge [T_{14}]_3^{15} \wedge [T_{15}]_{15}^{16} \wedge [T_{16}]_{16}^{13} \wedge [T_{13}]_{13}^{14} \wedge [T_{14}]_{14}^{15} \wedge [T_{17}]_{15}^{15}$

(8)$[T_6]_6^7 \wedge [T_7]_7^8 \wedge [T_8]_8^9 \wedge [T_{11}]_9^{11}$

(9)$[T_6]_6^7 \wedge [T_7]_7^8 \wedge [T_8]_8^9 \wedge [T_9]_9^{10} \wedge [T_{10}]_{10}^7 \wedge [T_7]_7^8 \wedge [T_8]_8^9 \wedge [T_{11}]_9^{11}$

(10)$[T_6]_6^7 \wedge [T_7]_7^8 \wedge [T_8]_8^2 \wedge [T_2]_2^4 \wedge [T_4]_4^5 \wedge [T_5]_5^1 \wedge [T_1]_1^{18} \wedge [T_{18}]_{18}^{19}$

(11)$[T_{12}]_{12}^{13} \wedge [T_{13}]_{13}^{14} \wedge [T_{14}]_{14}^{15} \wedge [T_{17}]_{15}^{17}$

(12)$[T_{12}]_{12}^{13} \wedge [T_{13}]_{13}^{14} \wedge [T_{14}]_{14}^{15} \wedge [T_{15}]_{15}^{16} \wedge [T_{16}]_{16}^{13} \wedge [T_{13}]_{13}^{14} \wedge [T_{14}]_{14}^{15} \wedge [T_{17}]_{15}^{17}$

以上 12 个测试序列能覆盖并发程序的所有可达状态的转移分支,它们都是可达状态的发生序列。对这 12 种测试方案分别进行测试,最终完成并发程序的测试。

结束语 本文根据并发程序的特性结合 Petri 网的特点对并发程序测试用例的生成描绘了大体框架,主要工作是在程序语言和测试用例之间通过 Petri 网搭建桥梁,利用图形矩阵的转换来生成相应测试序列。但本文并没有对测试用例集进行优化。进一步的工作是采用相应的算法对测试用例集进行优化。

参 考 文 献

[1] Wong W E, Lei Y, Ma Xiao. Effective Generation of Test Sequences for Structural Testing of Concurrent Programs[C]// Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems. Shanghai, China. IEEE Computer Society Press, 2005: 539-548

[2] Long D L, Clarke L A. Task Interaction Graphs for Concurrency Analysis[C]// Proceedings of the 11th International Conference on Software Engineering. Pittsburg, PA, USA. IEEE Computer Society/ACM Press, May 1989: 44-52

[3] Demartini C, Sisto R. Static Analysis of Java Multithreaded and Distributed Applications[C]// Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems. Kyoto, Japan. IEEE Computer Society Press, April, 1998: 215-222

[4] Katayama T, Furukawa Z, Ushijima K. Event Interactions Graph for Test-case Generation of Concurrent Programs[C]// Proceedings of the 2nd Asia-Pacific Software Engineering Conference. Brisbane, Queensland, Australia. IEEE Computer Society Press, December 1995: 29-37

[5] Herzog O. Static Analysis of Concurrent Processes for Dynamic Properties Using Petri Nets[C]// Kahn G, ed. Proceedings of the International Symposium on Semantics of Concurrent Computation(SCC'79). Evian, France. Springer Press. July 1979: 66-90

[6] Murata T, Shenker B, Shatz S M. Detection of Ada Static Deadlocks Using Petri Net Invariants[J]. IEEE Transactions on Software Engineering, 1989, 15(3): 314-326

[7] Notomi M, Murata T. Hierarchical Reachability Graph of Bounded Petri Nets for Concurrent-Software Analysis[J]. IEEE Transactions on Software Engineering, 1994, 20(5): 325-336

[8] Cheng J, Ushijima K. Analyzing Ada Tasking Deadlocks and Livelocks Using Extended Petri Nets[C]// Christodoulakis D, ed. Proceedings of the 10th Ada-Europe International Conference, Ada: The Choice for '92. Athens, Greece. Springer Press, May 1991: 125-146

[9] Chamillard A T, Clarke L A. Improving the Accuracy of Petri Net-based Analysis of Concurrent Programs[C]// Zeil S J, ed. Proceedings of the International Symposium on Software Testing and Analysis. San Diego, CA, USA, 1996: 24-38

[10] 蒋昌俊,陆维明. 基于 Petri 网语言的并发系统性质研究[J]. 软件学报,2001,12(4):512-520

[11] 范洪达,叶文. Time Petri 网在实时软件测试中的应用[J]. 计算机应用与软件,2003(12):23-25

[12] 林红昌,胡觉亮,丁佐华. 基于 Petri 网的软件测试用例的产生及分析[J]. 计算机工程与应用,2009,45(17):57-60

[13] 吴哲辉. Petri 网导论[M]. 北京:机械工业出版社,2006

[14] 袁崇义. Petri 网原理与应用[M]. 北京:电子工业出版社,2005

[15] 丁志军,蒋昌俊. 并发程序验证的时序 Petri 网方法[J]. 计算机学报,2002,5:467-475

[16] 孙世新,卢光辉等. 并行算法及其应用[M]. 北京:机械工业出版社,2005