基于CPN对系统的并发行为进行测试

李 华[1,2] 孙 涛[1] 王显荣[1] 邢 熠[1] 李颖杰[3] 夏兴行[1]
(内蒙古大学计算机学院 呼和浩特 010021)[1] (内蒙古大学网络信息中心 呼和浩特 010021)[2]
(西安飞行学院 兰州 733003)[3]

摘 要 首先对基本并发行为进行CPN建模及状态空间生成,逐渐增大模型的复杂性,展示了CPN建模并发行为可能存在的状态空间快速增大问题。在保证并发覆盖的前提下,将测试序列的生成划分为3个阶段,重点讨论了覆盖并发行为的测试序列生成方法。对于并发开始(末)库所按照CPN执行产生的状态空间中的节点进行了映射,得到状态空间中对应并发的开始(末)库所的开始(末)节点集合。通过对并发的开始(末)节点集内的节点间的关系进行分析,依据它们在状态空间中的前驱后继关系,生成一个由开始(末)节点的序列构成的序列的集合,然后以此序列集内的序列作为覆盖并发行为的测试序列的开始部分或者结尾部分,生成覆盖并发的测试序列。通过一个自行实现的P2P软件,使用了提出的建模方法及测试序列生成算法。最后通过得到的测试序列设计了测试场景及测试方案,包括Tracker并发行为测试系统结构、服务器测试方案及典型的测试场景设计。将P2P软件和TTCN-3测试机部署在一起实现了TTCN-3测试套的执行,测试结果表明测试工作的设计与实现是正确的。

关键词 并发行为,CPN,测试序列生成,TTCN-3
中图法分类号 TP393.06 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.1.048

Testing Concurrent Behavior of System Based on CPN

LI Hua[1,2] SUN Tao[1] WANG Xian-rong[1] XING Yi[1] LI Ying-jie[3] XIA Xing-hang[1]
(College of Computer Science, Inner Mongolia University, Hohhot 010021, China)[1]
(Center of Network and Information, Inner Mongolia University, Hohhot 010021, China)[2]
(Xi'an Flight Academy, Lanzhou 733003, China)[3]

Abstract The basic concurrent behavior was firstly modeled with CPN and the state space was obtained through CPN Tools. After that the complexity of the CPN was increased to show the possibly problems along with the state space quickly increasing. Secondly the phase of test generation was divided into three parts to guarantee the test coverage of concurrent behavior. Among them, the test sequences which only focused on the coverage of concurrent behaviors were generated and the other two parts were generated according to the regular path generation methods. The concurrency start(end) places were mapped with the state space nodes according to the CPN model execution and the sets of start (end) nodes in the state space were achieved. After analyzing the relationship between the nodes in the sets of start (end) nodes, the sequences sets were built according to the pre or pro relationship in the sets. The start(end) parts of the test sequences were selected from such sets, and the middle test sequences between start sequence to end sequence were generated. Furthermore, to illustrate the usage of the modeling method and the test generation, a simple P2P software system which is inherited concurrent behaviors was implemented and modeled with hierarchy CPN and the test sequences were generated to coverage the concurrent behavior. Finally, a TTCN-3 test scheme was designed according to the requirement of test sequences and a test scenario was designed. And the implemented software system and the TTCN-3 tester were deploying in one scenario to execute designed TTCN-3 testing suite. The test results show the correctness of the designed and implemented test work.

Keywords Concurrent behavior, CPN, Test sequence generation, TTCN-3

1 引言

随着互联网的迅速发展,基于网络的软件系统越来越复杂,软件的并发变得越来越普遍,目前基于网络实现的分布式应用系统多有此类特点,因此如何进行并发软件的测试变得尤为重要。基于形式化方法进行测试的优点是将问题的主要特性通过形式化模型进行刻画,将想要测试的功能、性能或者其它属性刻画出来以便有针对性地推导测试。

到稿日期:2014-11-14 返修日期:2015-04-30 本文受国家自然科学基金项目(61562064,61163011,61262082),内蒙古自然科学基金项目(2012MS0922)资助。

李 华(1964—),女,教授,主要研究方向为形式化方法、测试工程;孙 涛(1980—),男,博士,副教授,主要研究方向为软件测试、形式化方法;王显荣(1964—),男,副教授,主要研究方向为计算机网络、分布式软件。

常规的基于形式化方法进行测试的过程可以分为如下几步：形式化建模、基于模型生成测试序列、设计测试例、执行测试。其中建模是很重要的一步，特别是对于软件系统的并发性而言，并非所有的形式化方法都可以描述。而由于并发软件的执行具有一定的不确定性，基于形式化建模大多面临状态爆炸问题，导致自动化测试生成的测试序列数目巨大，测试执行的方案设计与实施都具有较大的难度。

本文贡献：

(1)通过示例展示了并发行为建立 CPN 模型时状态空间的变化。

(2)给出了基于 CPN 模型及状态空间，自动生成覆盖并发行为的测试序列的算法。首先把整个状态空间依据并发部分及其它部分进行了划分，使得测试序列生成所基于的状态空间规模变小；然后对并发行为的开始点和末节点间的关系进行了分析，减少测试序列生成时所使用的序列的开始节点和末节点数量，最坏情形时这两项指标不变。进而可以进行低冗余的测试序列生成。

(3)以一个自行实现的 P2P 软件为例，基于层次 CPN 对实现的 P2P 软件进行了建模，并对软件使用场景到并发行为测试方案进行了详细的设计，最后采用 TTCN-3 进行了测试执行，证明了所提方法的可用性。

2 预备知识

形式化建模的方法有很多种，常用的有：有限状态机 FSM(Finite State Machine)[1]、带标记转换系统 LTS(Labeled Transition System)[2]、输入输出变迁系统 IoTS(Input Output Transition System)[3,4]、Petri 网(Petri Net)[5]等。通常的软件说明中，如果有形式化描述，一般都是以 FSM 或者类似方法进行形式化说明，FSM 作为一种建模描述技术，优点是描述抽象简练，可以很容易地转化为其它形式的模型，缺点是在进行细化的描述时描述能力不足；而 Petri 网模型则适合描述系统的并发行为，并且可以动态地执行系统，检查系统的活性、可达性等性质[6]。本文采用着色 Petri 网(Coloured Petri Nets, CPN)[7,8]进行建模，工具采用 CPN Tools[7]。

2.1 CPN 简介

Petri 网分为低级 Petri 网和高级 Petri 网。针对不同的应用，Petri 网也有不同的种类，例如层次 Petri 网、时间 Petri 网、CPN 等。CPN 属于高级 Petri 网，它可以对并发系统进行建模并且对其进行属性分析，是一种结合了 Petri 网和高级编程语言特点的离散事件建模方法。CPN 提供了对并发、通信和同步建模的基本的图形符号和原语。

CPN 模型的优点在于可以同时描述状态(用库所表示)和动作(用变迁表示)，将模型及其属性都很直观地展现在同一个模型中，易于理解，增强了模型的可读性。CPN 十分适合对系统进行模拟和验证，并且能很好地描述具有信息交互、同步、并发和分布等特征的复杂软件系统。同时，CPN Tools 自身携带的功能可以对所建模型进行自动仿真，对模型的活性、可达性等性质进行检测，还可以自动生成状态空间并进行状态空间的分析，增强了模型的正确性检测。

定义 1[8] 一个非层次的 CPN 是一个九元组，$CPN=(P,T,A,\Sigma,V,C,G,E,I)$。

(1)P 为库所(Place)的有限集合；

(2)T 为变迁(Transition)的有限集合，且 $P\cap T=\varnothing$；

(3)A 为有向弧的集合，满足 $A\subseteq P\times T\cup T\times P$；

(4)Σ 为颜色(Color)集的非空有限集合；

(5)V 为变量的有限集合，当 $v\in V$ 时，有 $Type[v]\subseteq\Sigma$；

(6)$C:P\rightarrow\Sigma$ 为颜色集函数，定义库所的数据类型；

(7)$G:T\rightarrow EXPR_V$ 为防卫表达式，定义变迁的点火条件；

(8)$E:A\rightarrow EXPR_V$ 为弧表达式，定义弧的表达式；

(9)$I:T\rightarrow EXPR_\varphi$ 为初始化函数，定义库所的初始标记，且 $p\in P$，有 $Type[I(p)]=C(p)_{MS}$。

定义 2[8] 一个层次 CPN 的定义是一个四元组，$CPN_H=(S,SM,PS,FS)$。

(1)S 为模块的有限集。当 $s_1,s_2\in S,s_1\neq s_2$ 时，有 $(P^{s1}\cup T^{s1})\cap(P^{s2}\cup T^{s2})=\varphi$；

(2)$SM:Tsub\rightarrow S$ 为子模块函数；

(3)PS 为端口关联函数，为替代变迁 t 分配端口关联 $PS(t)\subseteq P_{sock}(t)\times P_{port}^{SM(t)}$。当 $(p,p')\in PS(t)$ 时，有 $PT(p)=PT(p')$，$CP=C(P')$，并且 $I(p)\langle\rangle=I(p')\langle\rangle$；

(4)$FS\subseteq 2^P$ 为非空交集，当 $(p,p')\in fs$ 且 $fs\in FS$ 时，有 $C(p)=C(p')$，$I(p)\langle\rangle=I(p')\langle\rangle$。

本文利用层次 CPN 进行网络软件系统的建模，目的是将该类系统的复杂性通过分层的方式进行抽象，使得测试可以更集中于想要测试的部分。

2.2 TTCN-3 简介

TTCN-3 是 ETSI(欧洲电信标准协会)推出的一种专门为测试领域制定的通用测试语言，主要应用于通信和电信领域。它不仅包含一般高级语言所具备的语言特性，还具有其自身的一些测试语言结构和语言元素，如并行测试成分、测试匹配机制、时钟处理及测试判决等。其特别关注对测试判断的处理、将 SUT 的反馈信息和期望接收到的反馈信息进行匹配、时钟处理、测试成分的分布方式以及信息编码的能力等。基于 TTCN-3进行测试，实现的过程如图 1 所示。首先 TTthree 把用 TTCN-3 核心语言编写的测试例模块编译成 Java 源程序，然后 Java 程序将被编译成 Java class 文件并被压缩成一个 JAR 文件包。将编译好的 JAR 文件包放置在与 TTCN-3 核心语言模块文件相同的目录下，为了能够在目标测试主机上运行针对实际被测系统的可执行测试集，还需要实现一些辅助的功能，即需要实现与被测系统相关的测试适配器及编解码功能，因此用 TTCN-3 实现测试工作的整个过程符合 ETSI 提出的国际测试标准。

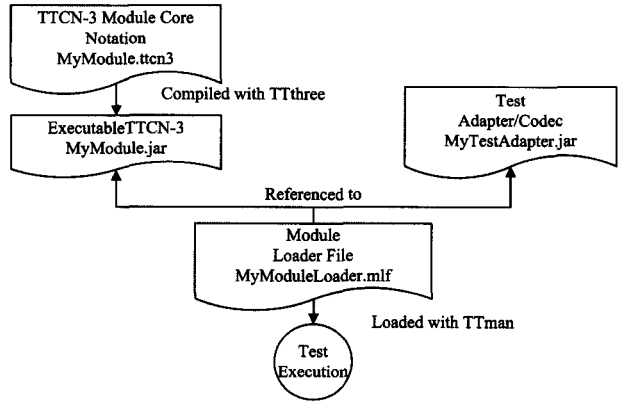

图 1 基于 TTCN-3 进行测试实现的过程

2.3 测试相关的概念

由于本文主要研究并发行为的测试，因此给出如下定义。

定义 3(并发开始点) CPN 模型中并发起始的库所在状态空间可能对应的多个状态节点。

定义 4(并发末节点) CPN 模型中并发结束库所在状态空间可能对应的多个状态节点。

定义 5(测试序列) 由状态空间中的节点组成的一个序列 $n_1 n_2 \cdots n_i \cdots n_m$，是状态空间的一条路径。

定义 6(并发覆盖) 状态空间中从并发开始点到末节点的所有测试序列。

由于在状态空间中两点间的测试序列(路径)都是可执行的，因此不区分测试序列和可执行测试序列。在无歧义时，也不区分序列和路径两个词。即在本文中就测试生成推导而言，测试序列、可执行测试序列、测试路径可以看成是一个含义。

3 并发行为的 CPN 建模及测试生成

Petri 网具有丰富的结构描述能力，可以描述顺序、并发、冲突关系。设 M_k、M_l、M_r 是 Petri 网 PN 的 3 个标识，t_i 和 t_j 是 Petri 网的两个变迁，则顺序、并发、冲突关系描述如下。

(1)顺序关系：若存在 t_i 和 t_j 使得 $M_k[t_i\rangle M_r$，且 $\neg M_k[t_j\rangle$，$M_r[t_j\rangle$，亦即，在 M_k 标识下，t_i 使能，而 t_j 不使能，且 t_i 的引发会使 t_j 使能，即 t_j 的使能以 t_i 的引发为条件，则称 t_i 和 t_j 有顺序关系。

(2)并发关系：若存在 t_i 和 t_j，使得 $M_k[t_i\rangle$ 和 $M_k[t_j\rangle$，并满足 $M_k[t_i\rangle M_l -> M_l[t_j\rangle$，且 $M_k[t_j\rangle M_r -> M_r[t_i\rangle$，亦即，它们当中任何一个变迁的执行都不会影响另一个变迁的执行，则称 t_i 和 t_j 有并发关系。

(3)冲突关系：若存在 t_i 和 t_j 使得 $M_k[t_i\rangle$ 和 $M_k[t_j\rangle$，并满足 $M_k[t_i\rangle M_r -> \neg M_r[t_j\rangle$，且 $M_k[t_j\rangle M_l -> \neg M_l[t_i\rangle$，亦即，$M_k$ 标识下，t_i 和 t_j 都能点火，但它们当中任何一个变迁的引发都会使另一个变迁不能点火，则称 t_i 和 t_j 在 M_k 下冲突。

3.1 最基本并发行为的 CPN 建模

为了有针对性地考虑并发的测试，先对基本的并发行为进行 CPN 建模，假设并发行为如图 2 所示。

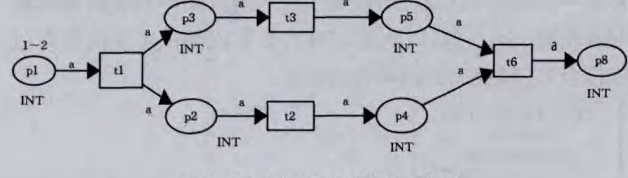

图 2 CPN 模型的基本并发行为

图 2 的 CPN 基本模型中用到的数据 colour 集的定义和变量声明如图 3 所示。

```
▼Standard declarations
  ▼colset UNIT=unit;
  ▼colset INT
  ▼colset STRING
  ▼var a:INT;
  ▼var b:INT;
```

图 3 数据 colour 集的定义和变量声明

在图 2 中，假设 $p1$ 中有一个 token，那么 $t1$ 点火，之后 $p2$ 和 $p3$ 都可以有 token 了，在无点火时间限制的情况下，$t2$ 和 $t3$ 可以先后或者同时点火，会形成多个 marking。例如该 CPN 模型从 $p1$ 开始执行，$p3->t3->p5->t6->p8$ 和 $p2->t2->p4->t6->p8$ 在执行的时候在 $t6$ 之前互不影响彼此的执行，只是在 $t6$ 需要同步。

通过 CPN 工具的执行，由图 2 的 CPN 可以得到图 4 的状态空间，其中有 6 个节点，也包括了每个节点所对应的库所中 token 的分布情况(图 4 粗线框内表示的是节点 1 的详细信息：concurrent 子模型中的 $p1$ 库所中 token 为 1～2，其余库所中的 token 均为 empty)。

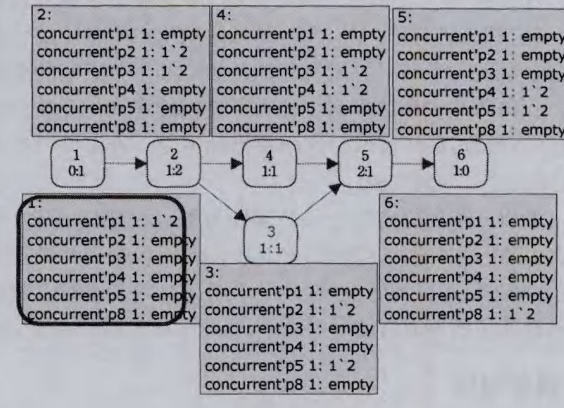

图 4 状态空间的基本并发行为

由图 4 可知，模型中初始库所 $p1$ 和末库所 $p8$ 对应状态空间中的节点分别是节点 1 和节点 6，那么只要生成节点 1 和节点 6 之间的测试序列即可。由于是单独描述了一个并发行为，因此状态空间相对比较简单，可以直接基于图 4 手工生成测试序列：1->2->4->5->6 和 1->2->3->5->6。另外，从生成的状态空间也可以看出，模型可以描述并发行为的执行情况。

3.2 含有并发的 CPN 建模

一般并发行为只是系统中的一个部分，因此将该典型并发行为放在一个更为复杂的环境中进行 CPN 建模，以考察其状态空间的变化，如图 5 所示。图 5 的 CPN 模型中用到的数据 colour 集的定义和变量声明仍如图 3 所示。

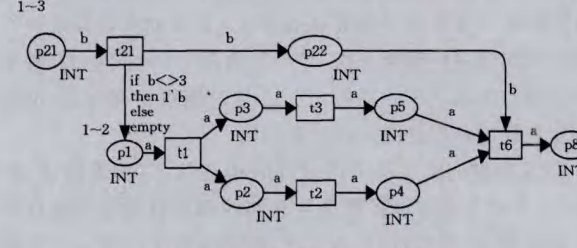

图 5 含有并发行为的 CPN 模型

图 5 所示的模型主要描述了包含并发行为的模型的执行情况。模型的状态空间如图 6 所示，状态空间图中包含 7 条路径，表示模型中并发行为的 7 种执行方式。使用 CPN Tools 生成的状态空间分析如表 1 所列。在图 6 所示的状态空间中，有 11 个节点、16 条弧，强连通图中的节点数和弧数与状态空间的相同，说明所建的模型中没有环路。终端节点是 11，说明所建模型能够正常终止。

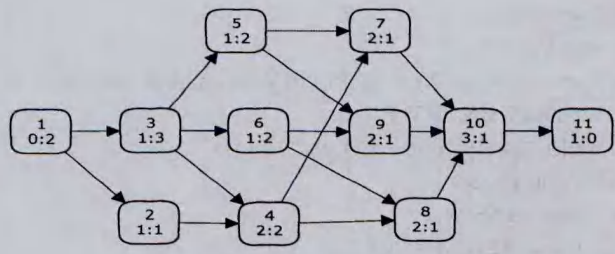

图 6　并发行为模型的状态空间

表 1　状态空间分析表

State Space	节点数:11	弧数:16
Scc Graph	节点数:11	弧数:16
Dead Markings	[11]	

3.3　基于 CPN 的测试序列生成

基于 CPN 模型和得到的状态空间 G_O，设计测试序列生成算法，要求在开始库所标识到末尾标识之间覆盖并发对应的所有标识。

该问题可以分为 3 个部分完成。

第一部分：求状态空间任一首节点到并发开始节点（可能不止一个）的测试序列；

第二部分：求并发开始节点到并发末节点之间的所有测试序列。

第三部分：求并发末节点到状态空间任一末节点间的测试序列。

由上，我们依据覆盖所有并发活动的需要将生成测试序列的工作规模降为生成覆盖并发行为对应的所有测试序列。

在上面 3 部分工作中，第一部分和第三部分可以看成求两点间的路径，可以参阅现成的一点到多点的最短路径算法。

第二部分算法设计如下：

(1)在状态空间 SP 中标记出初始库所中含有 token 的所有节点 s_i，构成初始节点集 S，并标记出末库所中含有 token 的所有结点 d_i，构成终端节点集 D。

(2)通过可达路径分析，根据生成的状态空间中节点间是否具有前驱或者后继关系，将初始节点集 S 中的节点构成序列集合 $StaA_1$，$StaA_2$，…，等多个序列集合；根据生成的状态空间中节点是否具有前驱或者后继关系，将末节点集 D 中的节点构成序列集合 $DesA_1$，$DesA_2$，…，等多个序列集合。

假设 a_i，a_j，a_k，a_l，d_i，d_j，d_k，d_l 是 G_O 的节点，$a_i->a_j->a_k \in StaA_1$，$a_l \in StaA_2$，$d_i->d_j->d_k \in DesA_1$，$d_l \in DesA_2$，符号->表示节点之间具有前驱后继关系；$a_i\_testseq$ 和 $d_k\_testseq$ 分别表示前述第一部分和第三部分生成的测试序列。

(3)从初始节点集合 $StaA_1$，$StaA_2$，…，等中选取最小标识节点，构成一个初始节点（并发行为在状态空间的开始节点，例如 a_i 和 a_l）集合 $StaAMin$；从初始节点集合 $StaA_1$，$StaA_2$，…，等中选取最大标识节点，构成一个初始节点（例如 a_k 和 a_l）集合 $StaAMax$。

(4)从末节点集合 $DesA1$，$DesA2$，…，等中选取最小标记节点，构成一个末节点（并发行为在状态空间的末节点，例如 d_i 和 d_l）集合 $DesAMin$；从末节点集合 $DesA1$，$DesA2$，…，等中选取最大标记节点，构成一个末节点（并发行为在状态空间的末节点，例如 d_k 和 d_l）集合 $DesAMax$。

(5)将初始节点集合 $StaAMax$ 中的节点与终端节点集合 $DesAMin$ 中的节点进行组合。最后，根据深度优先搜索算法，生成覆盖每个组合的测试序列。

为了说明上述算法，构造一个简单的例子，如图 7 所示。

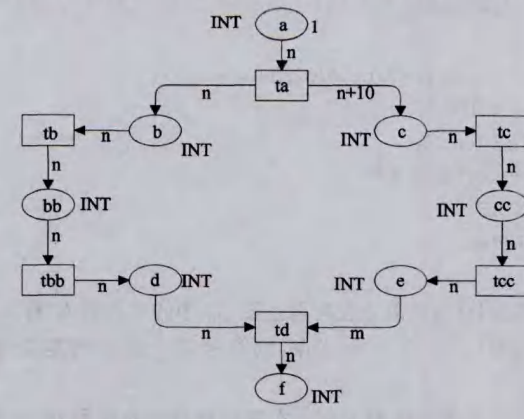

图 7　并发示例

图 7 对应的状态空间如图 8 所示，其中包含 11 个状态节点，记为 $M1$，$M2$，…，$M11$。由于篇幅限制，图 8 中仅给出了 $M1$ 和 $M4$ 的 token 分布情况。

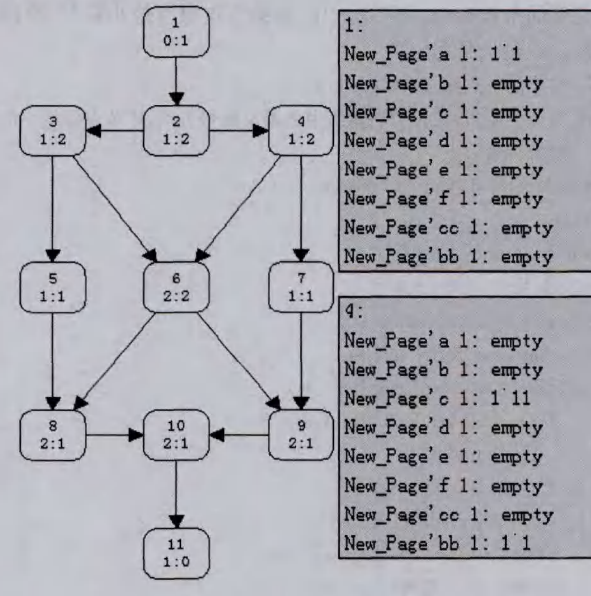

图 8　图 7 对应的状态空间

假设模型中初始库所为 bb，末库所为 d。由于 $M4$ 中 bb 库索含有 token，因此 $M4$ 是初始状态节点。同理可得，初始节点集合为{$M4$，$M6$，$M8$}，末节点集合为{$M7$，$M9$，$M10$}。根据 $M4$，$M6$ 和 $M8$ 的前驱后继关系可知，a_i 为 $M4$；根据 $M7$，$M9$ 和 $M10$ 的前驱后继关系可知，d_k 为 $M10$。所以，测试序列包括 $M4$ 与 $M10$ 之间的 3 条序列，分别为：$M4M6M8M10$，$M4M6M9M10$ 以及 $M4M7M9M10$。

由于篇幅限制，仅给出算法核心的 4 个部分。

算法的第一部分：给定一个库所的名称，从可达图 G_O 中找到所有该库所的 token 不为 empty 的节点。

输入：CPN 模型对应的. xml 文件 filename，要查询的库所名称 tokenName

输出：该库所不为 empty 的有序节点编号集 nodeNumArr

```
1. Begin
2. Read(CpnModel. xml);
3. Len=GetLength(ssnode);
4. FOR (i=0;i<Len;i++)
5.    GetText(ssnode[i])
```

```
6.   IF (IsExist(ssnod,tokenName)
7.     IF (IsEmpty(tokenName. Value))
8.       Continue;
9.     ELSE
10.        nodeNumArr. Add(GetId(ssname))
11.    ENDIF
12.  ELSE
13.    Print("ERROR")
14.  ENDIF
15. ENDFOR
16. END
```

算法分析：其复杂度与可达图 G_O 中的节点数 N 有关，假设可达图 G_O 中有 n 个节点，则算法第一部分的复杂度为 $O(n)$。

以图 7 所示模型为例。本算法针对初始库所 *bb* 可生成的初始节点集合为{*M*4,*M*6,*M*8}；针对末库所 *d* 可生成的末节点集合为{*M*7,*M*9,*M*10}。

算法的第二部分：对于一个标记初始状态节点集合或者给定一个标识终端状态节点集合，对集合中具有前驱或者后继关系的节点分组，每一组中的初始节点保留最小标号，终端节点保留最大标号。

输入：有序的节点编号数组 nodeArr

输出：节点分组后，初始节点每组中的最小标号节点，末节点每组中的最大标号节点

```
Function getNodeNumAfterGroup(nodeArr)
1. Begin
2. resultArr=new array[];
3. reNodeArr=new array[];
4. copyedArr=nodeNumArr;
5. arrLength=nodeArr. length;
6.   FOR (i=0;i<arrLength;i++)
7.     IF(reNodeArr. has(nodeArr))
8.       continue;
9.     ENDIF
10.    reNodeArr. add(nodeArr[i]);
11.    FOR (j=0;j<arrLength;j++)
12. //判断两个节点是否可以连通
13.      BooleanisConn = isConnectNode ( nodeArr [i], copyedArr
         [j]);
14.      IF (isConn==True)
15.        reNodeArr. add (copyedArr[j]);
16.      ENDIF
17.      ENDFOR
18.  ENDFOR
19.  return resultArr;
20. END
```

算法分析：假设 nodeArr 有 n 个节点，该算法与冒泡排序算法的复杂度相同，算法第二部分的复杂度为 $O(n^2)$。

以图 7 所示模型为例。本算法针对初始节点集合{*M*4,*M*6,*M*8}可生成最小标号节点 *M*4；针对末节点集合{*M*7,*M*9,*M*10}可生成最大标号节点 *M*10。

算法的第三部分：获取初始节点和末节点之间的测试序列。

输入：基于可达图 G_O 的初始节点集 Vo

输出：基于 Go 的可执行测试序列集合 ExeSeq

```
1. Begin
2. ExeSeq={};
3. Find(V0)
4. Visited. AddNode(V0);//访问并把 V0 加入到集合 visited 中
5. W=FindNextNode(V0);
6. While(Exist(V0. NextNode(). NotVisit()))
7. IF(W in Visited){
8.   pathx=DFS(W);
9.   Exeseq=Exeseq+{pathx};}
10. ENDIF
11. END
```

算法分析：其复杂度与深度优先搜索算法相同。上述算法就是满足覆盖每个变迁至少一次的可执行序列集。

以图 7 所示模型为例。本算法针对最小标号节点 *M*4 和最大标号节点 *M*10，可生成如下 3 条测试序列：

*M*4*M*6*M*8*M*10

*M*4*M*6*M*9*M*10

*M*4*M*7*M*9*M*10

算法的第四部分：将每个测试序列的首节点 V_0 基于 G 扩展至 G 的开始顶点，得到从 G 的开始顶点覆盖 $PN(X)$ 的测试序列集。

输入：基于 PN 的测试序列集 ExeSeqx

输出：扩展至 PN 的可达图 G 的开始顶点 v1 的可执行测试序列集 ExeSeq_EX

```
1. Begin
2. ExeSeq_EX={};
3. Pe1=FindAllTopNode(ExeSeqx);//求得 ExeSeq 的开始顶点集合 Pe1
4. p=GetNode (Pe1);// p 为 Pe1 中的一个顶点，取不到时，p=null
5. While(p!=NULL){
6. path=GetShortestLength(v1,p);//v1 到 p 的最短路径 path
7. seq_p=GetSeq_FirstNode(p);// p 开头的可执行测试序列
8. ExeSeq_EX= ExeSeq_EX+{path+ seq_p};
9. Pe1= Pe1-{p};
10. p=GetNode(Pe1);}
11. ExeSeq_EX=ChangeNodeToPlace(ExeSeqx);
12. END
```

算法分析：假设 *ExeSeq* 中的可执行测试序列含有 m 个不同的开始顶点，状态图中有 n 个节点，则算法第四部分的复杂度为 $O(m*(n^2))$。

以图 7 所示模型为例。本算法针对测试序列：*M*4*M*6*M*8*M*10，*M*4*M*6*M*9*M*10 以及 *M*4*M*7*M*9*M*10，可生成如下 3 条扩展后序列：

*M*1*M*2*M*4*M*6*M*8*M*10

*M*1*M*2*M*4*M*6*M*9*M*10

*M*1*M*2*M*4*M*7*M*9*M*10

上述序列是关于算法给出的测试序列。

4 举例

4.1 CPN 建模

为了说明提出的测试方法的有效性，以简单实现的并发软件为例，对其进行建模及测试。本文实现的并发软件是一个支持多点下载和上传文件资源的 P2P 软件。

在开发的并发软件中，Tracker 服务器接收节点请求消息并返回响应消息。各个节点会根据收到的消息与其它节点建立连接，建立连接的节点会相互交换位图信息，并根据位图信息向其它节点请求文件资源。

本文建立的模型分为 3 层，模型顶层是 Top 层，体现了 Tracker，Peer1，Peer2 和 Seed 4 个节点间的信息交互；第二层是各个节点的功能层，主要体现各个节点接收其它节点发送的信息，并对其进行响应的过程；第三层是各个节点发送和接收信息动作的具体实现。

图 9 给出了软件 Top 层模型，描述了 4 个节点的交互情况，其中 Seed 节点拥有全部的资源，它只接收其它节点发来的资源请求，并把其它节点需要的资源发送给对方，Seed 节点不向其它节点请求资源。Peer1 可以向 Peer2 和 Seed 节点请求该节点没有而对方拥有的资源，同样 Peer2 也可以向 Peer1 和 Seed 节点请求自己没有而对方拥有的资源，Peer1、Peer2 和 Seed 都可以向其它节点发送请求分片资源。

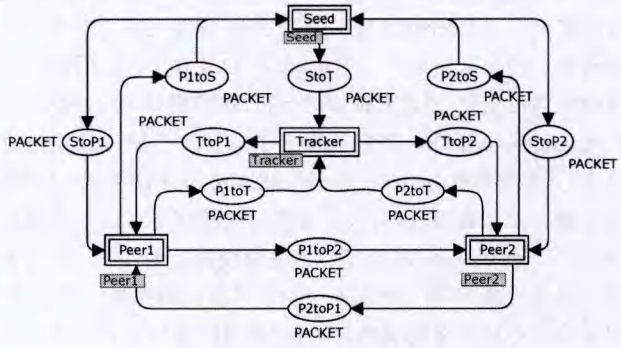

图 9　并发软件的顶层页

在 CPN 模型建成之后，通过 CPN Tools 对其进行了正确性验证，生成了详细的状态空间分析报告。由于含有并发行为的软件系统模型的状态空间通常较为庞大，本文截取了部分报告信息，如表 2 所列。

表 2　状态空间分析

State Space	节点数：3409 弧数：9414
Scc Graph	节点数：3409 弧数：9414

本节对 Tracker 模型中 revpck－＞ T_receive－＞forward 的并发行为进行测试。该并发行为的主要功能是对其它节点发来的消息进行存储，并对其它节点进行正确的信息回应，图 10 是该并发行为的 CPN 模型。

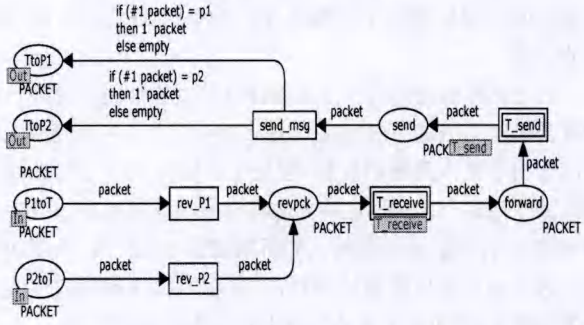

图 10　Tracker 的 CPN 模型

其中，T_send 的 CPN 模型描述如图 11 所示。

使用提出的算法进行测试序列生成，得到如下结果：模型中库所 revpck 中具有活动 token 的情况对应到状态空间中有 7、25、32、40、49、52、56、63、68、81 等 10 个节点，模型中库所 forward 中具有 token 的情况对应到状态空间中有 87、94、101、109、116、125、132、142、149、160、167、179、186、199、206、226 等 16 个节点。通过对状态空间进行路径可达性分析，得到 7、25、32、40、49、52、56、63、68、81 等 10 个节点在状态空间中都具有前驱或者后继关系，库所 forward 对应的 16 个节点也具有前驱或者后继的关系。于是根据提出的算法，只需生成节点 7 到节点 226 之间的全部序列即可。由于篇幅所限，本节只给出部分测试序列，如图 12 所示。

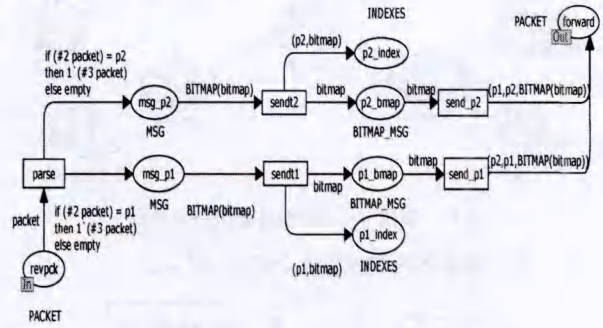

图 11　T_send 的 CPN 模型

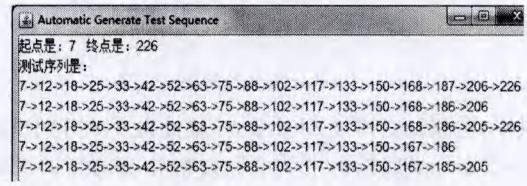

图 12　生成的部分测试序列

5　并发行为的 TTCN-3 测试

5.1　测试设计

根据远程分布测试法[9]的描述，结合 Tracker 服务器中并发行为的测试，设计了如图 13 所示的测试机结构。整个测试机结构由 3 部分组成：主测试机、辅助测试机和 Tracker 服务器，它们由网络连接。Tracker 服务器主动打开 8001 端口，接收测试机的请求信息。测试机上安装 Ttworkbench，运行测试 Tracker 服务器并发行为的测试例。在测试运行后判断 Tracker 服务器是否能够与各个测试机进行正确的信息交互，最后得出对 Tracker 服务器并发行为的测试结果。

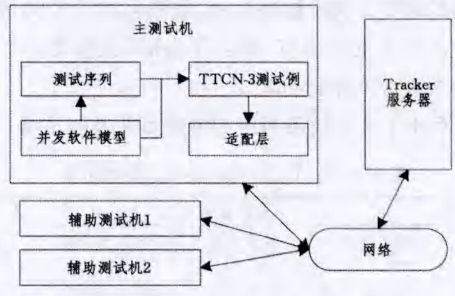

图 13　Tracker 并发行为测试机结构

5.2　测试实施与测试结果

测试 Tracker 服务器的场景设计如图 14 所示，由 1 台路由器、2 台交换机和 4 台计算机组成；其中 4 台计算机分别为 1 台安装 TTCN-3 测试软件的 TTCN-3 测试机、1 台 Tracker 服务器、2 台装有并发软件客户端的辅助测试机。其中 TTCN-3 测试机连接在交换机 1 上，交换机 2 与 Tracker 服务器和 2 台辅助测试机相连。交换机 1 分别与辅助测试机 1、

TTCN-3 测试机和路由器相连，交换机 2 分别与辅助测试机 2、Tracker 服务器和路由器相连。在图 14 中，TTCN-3 测试机模拟 Seed 节点，通过网络环境向 Tracker 服务器发送 IP 地址、端口号以及位图等报文信息，Tracker 服务器接收到相应的报文信息之后，对其进行回应，TTCN-3 测试机将对 Tracker 服务器回应的报文信息的正确性进行验证。

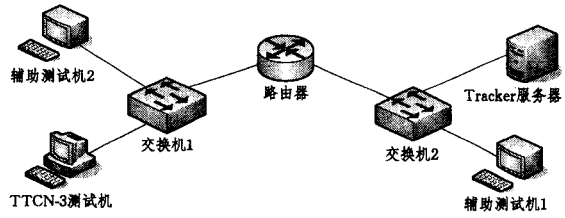

图 14　测试 Tracker 服务器的场景设计

Tracker 服务器的测试方案如图 15 所示。

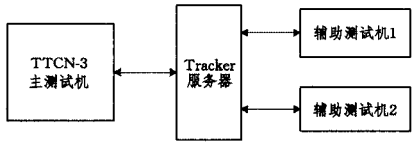

图 15　Tracker 服务器的测试方案

其中，主测试机模拟 Seed 节点，辅助机 1、2 分别模拟两个 Peer 节点。测试部件中的主测试机和辅助测试机都与 Tracker 服务器进行信息交互，由生成的测试序列可知，当各个测试机向 Tracker 服务器发送请求报文信息的顺序不同时，每个测试机收到的响应报文信息也不相同，生成的测试序列包括测试节点对 Tracker 服务器的所有响应动作。根据上一节生成的测试序列，结合 CPN 编写测试 Tracker 服务器中并发行为的测试套。

使用 3.3 节中的测试序列生成算法对 Tracker 中指定的并发行为生成了 27 条测试序列，部分如图 12 所示。通过对测试序列进行分析，节点与被测系统之间进行交互的行为可以分为 6 种情形，在这 6 种情形中，Tracker 服务器对每一个节点发送的消息都是不同的，因此，本节设计了不同的测试例，以测试每一种情形下 Tracker 服务器是否对节点的请求做出正确的响应。实验结果显示，服务器在这 6 种情况下对测试系统进行了正确响应，而且 Tracker 服务器能够对多个节点的请求进行正确的响应。

表 3 列出了 6 种情形对应的测试目的及测试结果。

表 3　6 种情况的测试目的及测试结果

交互情况	测试目的	通过情况
⟨T3,T⟩	服务器对单个主机请求资源是否能做出正确响应	通过
⟨1,T⟩ ⟨T3,T⟩	服务器对两个主机按不同顺序请求资源是否能做出正确响应	通过
⟨2,T⟩ ⟨T3,T⟩	服务器对两个主机按不同顺序请求资源是否能做出正确响应	通过
⟨1,T⟩ ⟨2,T⟩ ⟨T3,T⟩	服务器对 3 个主机按不同顺序请求资源是否能做出正确响应	通过
⟨2,T⟩ ⟨1,T⟩ ⟨T3,T⟩	服务器对 3 个主机按不同顺序请求资源是否能做出正确响应	通过
⟨T3,T⟩	服务器对主机发来的错误信息请求是否能做出响应	通过

其中 T 表示 Tracker 服务器，T3 表示 TTCN-3 主测试系统，1 表示辅助测试系统 1，2 表示辅助测试系统 2，⟨a,T⟩表示测试系统与 Tracker 服务器的交互。a 表示 TTCN-3 主测试、辅助测试系统 1 或辅助测试系统 2。交互情况中列出了各个测试系统与 Tracker 服务器交互的先后顺序。

6　相关工作介绍

软件测试过程通常包括构造测试用例、执行测试输入和检查执行结果 3 个步骤[10]，对并发软件测试方法的研究也是围绕这 3 个步骤展开的。目前国内外较为成熟且可行的两种并发程序测试方法为可达性测试[11]和 Concolic 测试[12]，也有研究学者将模型检验(Model Checking)作为并发程序的测试方法之一。O. Edelstein 针对并发 Java 程序完成了基于动态分析技术的测试框架[13]。Remenska D 使用 UML 语言对并发软件进行建模，通过对模型的分析，在前期找到软件设计上出现的错误问题[14]。杨鹤标提出了一种基于统一建模语言(UML)活动图的功能测试场景生成方法，利用拓扑反蚁群算法处理并发结构模块，解决了并发活动排序导致的场景爆炸问题[15]。使用 Petri 网对并发系统建模比使用 UML 活动图建模更具优势，不但可以精确地分析系统的静态特性，而且可以很好地分析系统的动态行为，它既可以采用形式化直观的图形表示，也可以引入数学方法对其性质进行分析和验证。

U. Farooq 等提出了一种基于随机漫步(Random Walk)算法的测试例生成方法，即在给出一定的顺序覆盖标准(如分支覆盖、边覆盖等)和并发覆盖标准(如节点、边、同步点的覆盖等)的基础上，利用随机漫步算法对 CP-nets 模型的状态空间进行多次搜索，记录搜索路径并形成测试序列，当已生成的测试序列集合满足特定的覆盖标准时，停止状态搜索过程，完成测试序列的生成[16]。

TTCN-3 是全球唯一的标准测试语言，通过广泛的接口描述许多类型的系统测试。其典型的应用领域为系统测试、交互性测试、协议测试、业务测试、模块测试等。TTCN-3 通过顺序、选择、循环、并行激励及响应等为复杂的并行和分布式测试行为提供了简单和有效的描述方式。它可以通过动态创建任意数量的、能以并行方式运行的并行测试成分(PTC)及动态并行测试配置完成测试过程，从而实现对并行测试的充分支持[17]。

上述文献中提到的方法都是针对并发软件提出的基于某种覆盖标准的测试序列生成方法。这些方法在并发覆盖标准方面给出了让人信服的探讨，理论上可以生成满足覆盖标准的测试序列集合；但是，这些方法都没能很好地解决如何生成测试序列的问题，为了达到人们所期望的覆盖标准，只能使用随机法或遍历法生成测试序列，这样做不仅导致测试生成过程效率低下，更重要的是会生成大量无用的测试序列。

结束语　随着社会的发展，网络软件系统在实际生活中得到越来越广泛的应用，其中有许多并发行为。由于并发行为具有不确定和不可复制性的特点，目前对软件中并发行为的测试还没有很好的办法。Petri 网是一种有效描述并发行为的建模工具，它的描述能力强，特别是它的动态执行在形象地展示了系统的能力的同时，间接地检查了模型的活性及可

达性等特性。但由于采用库所和变迁作为基本描述元素，使得其本身的节点数比描述同一系统的 FSM 的节点数多。CPN 通过颜色集进一步增强了 Petri 网的描述能力，并且 CPN Tools 可以基于建立的模型生成状态空间，进而检验模型是否有活锁、死锁以及可达性等。Petri 网的可达性是通过 marking(标记)构成，当系统中有并发行为时，极有可能导致状态空间爆炸。

本文的测试方法是基于 CPN 进行测试推导研究的，并且采用 TTCN-3 进行了测试例的实现，并设计了测试方案进行测试执行。由于本文的主要工作是针对并发系统进行 CPN 建模，并有模型化简的考虑，因此在建模初期就对模型的规模有所控制，然后在并发覆盖测试序列生成时针对模型状态空间进行研究，此时由于令牌的选取，使得状态空间形成的状态图已经成为一个确定的图。本文在建模时针对并发行为测试问题进行了建模的优化考虑；在测试序列生成时，先生成覆盖并发行为的测试子序列，再构成完整测试序列，从而避免了大量冗余序列的生成。因此，本文得到了既满足并发行为覆盖要求，又减少了测试序列数目的测试例。

当系统中的并发行为很多时，仍会导致状态空间的爆炸，因此下一步的研究工作是考虑如何能尽量地化简 CPN 模型；另外，令牌的选取、变迁点火时间的限制方面也需要深入研究。

参 考 文 献

[1] Lee D, Yannakakis M. Principles and methods of testing finite state machines-a survey[J]. Proceedings of the IEEE, 1996, 84(8): 1090-1123

[2] Tretmans J. A formal approach to conformance testing[D]. University of Twente, Enschede, Netherlands, 1992

[3] Tretmans J. Test generation with inputs, outputs and repetitive quiescence [J]. Software Concepts and Tools, 1996, 17(3): 103-120

[4] Petrenko A, Yevtushenko N, Huo J L. Testing transition systems with input and output testers[J]. Testing of Communicating Systems, Springer Berlin Heidelberg, 2003, 2644 (0302-9743): 129-145

[5] Peterson J L. Petri net theory and the modeling of systems [M]. Englewood Cliffs, Nj, Prentic-Hall, Inc. 1981

[6] Wu Z H. Introduction of Petri Nets[M]. Beijing: Mechanical Industry Press, 2006(in Chinese)
吴哲辉. Petri 网导论[M]. 北京：机械工业出版社，2006

[7] Jensen K. Coloured Petri Nets: Basic concepts, analysis methods and practical use. Vol. 3 [M] // Practical use, Monographs in Theoretical Computer Science. Springer, 1997

[8] Jensen K, Kristensen L M. Coloured Petri Nets: modelling and validation of concurrent systems[M]. Spring-Verlag, 2009

[9] Wang G, Wu J, Xu L, et al. Research on test adapter framework for distributed TTCN-3 test execution platform[J]. Acta Electronica Sinica, 2009, 37(1): 125-130(in Chinese)
王冠，吴际，徐珞，等. 面向 TTCN-3 分布式测试执行平台的测试适配器框架的研究与设计[J]. 电子学报，2009，37(1)：125-130

[10] Lei Y, Carver R H. Reachability testing of concurrent programs [J]. IEEE Transactions on Software Engineering, 2006, 32(6): 382-403

[11] Sen K, Marinov D, Agha G. CUTE: A concolic unit testing engine for C[C] // Proceedings of the 13th ACM SIGSOFT Symposium on Foundations of Software Engineering jointly with 10th European Software Engineering Conference. Lisbon, Portugal, ACM Press, 2005

[12] Merz S. Model Checking: A tutorial overview[C] // Proceedings of the 4th Summer School on Modeling and Verification of Parallel Processes. Nantes, France, Springer Press, 2000: 3-38

[13] Edelstein O, Farchi E, Nir Y, et al. Multithreaded java program test generation[J]. IBM Systems Journal, 2002, 41(1): 111-125

[14] Remenska D, Templon J, Willemse T A C, et al. From UML to process algebra and back: An automated approach to model-Checking software design artifacts of concurrent systems[M] // NASA Formal Methods. Springer Berlin Heidelberg, 2013: 244-260

[15] Yang H B, Li Y P. Functional test scenarios generation method based on UML activity diagrams [J]. Computer Engineering, 2011, 37(21): 55-57(in Chinese)
杨鹤标，李云平. 基于 UML 活动图的功能测试场景生成方法[J]. 计算机工程，2011，37(21)：55-57

[16] Farooq U, Lam C P, Li H. Towards automated test sequence generation[C] // Proceedings of the 19th Australian Conference on Software Engineering. Perth, Australia, 2008: 441-450

[17] Din G, Tolea S, Schieferdecker I. Distributed load tests with TTCN-3[M] // Testing of Communicating Systems: TestCom 2006, LNCS 3964, 2006. IFIP International Federation for Information Processing, 2006: 177-196

(上接第 206 页)

[19] Tian Jun-feng, Zhang Ya-jiao. Checkpoint trust evaluation method based on Markov[J]. Journal on Communications, 2015, 36(1): 230-236(in Chinese)
田俊峰，张亚姣. 基于马尔可夫的检查点可信评估方法[J]. 通信学报，2015，36(1)：230-236

[20] Hu Yong. Research on the Evaluation of Information System Security[D]. Chengdu: Sichuan University, 2007(in Chinese)
胡勇. 网络信息系统风险评估方法研究[D]. 成都：四川大学，2007

[21] Forrest S, Hofmeyr S A, Somayaji A, et al. A sense of self for unix processes[C] // Proceedings of IEEE Symposium on Computer Security and Privacy. 1996: 120-128

[22] Xu Chan, Liu Xin, Wu Jian, et al. Software Behavior Evaluation System Based on BP Neural Network[J]. Computer engineering, 2014, 40(9): 149-154(in Chinese)
徐婵，刘新，吴建，等. 基于 BP 神经网络的软件行为评估系统[J]. 计算机工程，2014，40(9)：149-154