算法实验报告

——自动生成业务过程模型日志的算法设计

姓　　名： 张　香　玉

学　　号： MF1932243

指导老师： 李　传　艺

学　　院： 软件学院/软件工程

时　　间： 2019/12/27

目录

一、解决方案的整体描述：

1.1、问题描述

要求：

生成输入的业务过程模型包含的所有可能的任务执行序列，其中循环任务执行不超过 1 次。提交算法源代码，实现接口 getLogOfModel(String modelFile, String logFile) 。

说明：

使用 Workflow-net 表示的过程模型包含四个元素，库所(Place)、变迁(Transition)、弧线(Arc)和令牌(Token)。没有输入弧线的库所称为输入库所，没有输出弧线的库所称为输出库所。每一个过程模型只能包含一个输入库所和一个输出库所。过程模型每一次运行的初始状态是输入库所中填入一个令牌，结束状态是在输出库所中填入一个令牌。每一个库所有一个或多个输入、输出变迁，每一个变迁有一个或多个输入、输出库所。所有输入库所中都填有令牌时，变迁达到可执行状态；当变迁执行时消耗每一个输入库所中的 1 个令牌，并在所有输出库所中填入一个新的令牌。过程模型运行的过程是从初始状态开始，所有可执行的变迁都执行一次，最终达到结束状态。将一次运行中所有执行的变迁按照执行顺序记录下来，形成一个变迁的执行序列，称为过程模型的一条日志，并且使用 CaseID(可以是整型数值)作为该条日志的唯一标识符。所有能够使得过程模型从开始状态到达结束状态的变迁执行序列构成全局完整的过程日志。

输入：

Workflow-net 表示的业务过程模型（Business Process Model），XML 格式。

输出：

过程日志

1.2、解决方案

1. 解析 PNML 文件，提取库所(place)、变迁(transition)、弧(arc)的信息。
   1) PNML 是基于 XML 的 Petri 网表示方法。为了支持不同类型的 petri 网，其中允许定义新的 petri 网类型。PNML 能够表示各种不同的 petri 网。PNML 把一个 Petri 网看作一个加了标签的图，所有存储在标签中的附加信息都可以在网、网的节点或者连接弧之上。
   2) 增加全局变量存储提取出的信息：
      a) String p_sou//存放模型的源点，即输入库所
      b) String p_de//存放模型的终点，即输出库所
      c) List<String> idList//存放所有变迁的 id
      d) List<String> nameList//存放所有变迁的 name，与 id 对应
      e) List<String> countList//存放变迁被访问的次数，与 id 对应
      f) HashMap<String ,List<String>> mappos//存放所有的正向弧
      g) HashMap<String ,List<String>> mappos//存放所有的正向弧
      h) Set<String > res=new TreeSet<>()//存放日志信息
2. 根据解析出的信息，对模型进行处理
   1) 处理顺序、选择结构：以深度优先的思想，递归遍历模型
   2) 处理并行结构：依次遍历并行结构起点指向的多个库所，直到遍历完指向并行结构终点的库所。
   3) 处理循环结构：找出每一个可能进入循环的库所，控制对其指向的变迁的访问次数。
3. 将处理结果逐条写入文件

1）处理过程中的得到的每一条变迁执行记录存入 set 中

2）把 set 中的记录逐条写入日志文件中

二、解决整体方案的伪代码：

该部分为整体方案，具体模块的解释见下一节

Algorithm：get logs from the model

```
getLogOfModel(String modelFile,String logFile)
input:
    modelFile: 模型的存储路径
    logfile: 存放日志的路径
parser(modelFile)
places ← p_sou
path ← “”
ActivityTrancer(places, path)
foreach path ∈ res do
    writeFile(path,logFile)
end
```

三、解决方案各模块的具体描述与伪代码：

1.1、解析 pnml 文件

1.1.1、具体描述

1. DocumentBuilderFactory 解析 pnml

1）调用 DocumentBuilderFactory.newInstance() 方法得到创建 DOM 解析器的工厂

2）调用工厂对象的 newDocumentBuilder 方法得到 DOM 解析器对象。

3) 调用 DOM 解析器对象的 parse() 方法解析 pnml 文档，得到代表整个文档的 Document 对象。

2. 得到变迁列表将每个变迁的 id 放入 list 中

1) 得到标签为“transition”的元素列表并遍历

2) 通过“id”属性保存变迁的 id 到集合中

3) 遍历变迁子节点，通过“text”保存变迁的 name 到 list 集合中

3. 得到正向弧、反向弧并放到 map 中

1) 得到标签为“arc”的元素列表并遍历

2) 通过“source”属性、“target”属性把正向弧、反向弧分别保存到 map 中

4. 遍历 place，找到源点和终点

1) 得到标签为“place”的元素列表并遍历

2) 保存源点（输入库所）、终点（输出库所）

1.1.2、伪代码

Algorithm ： parse the PNML file.

```
parserPnml(modelFIle)
Input:
    modelFile: 模型的存储路径
foreach node ∈ Elements(transition) do
    tranId ← nodeValue(id)
    idList.add(tranId)
    foreach node ∈ node.childNodes do
        tranName ← nodeValue(text)
    end
end
foreach node ∈ Elements(arc) do
    source ← Attribute(source)
    target  ← Attribute(target)
    mappos.put(source,target)   //positive arc
    mapneg.put(target,source)   //negative arc
end
foreach node ∈ Element(place) do
    placeId ← nodeValue(id)
    if !mapneg.containsKey(placeId) then
        p_sou ← placeId
    end
    if !mappos.containsKey(placeId) then
        p_des ← placeId
    end
end
```

1.2、判断当前变迁是否为并行结构的终点

1.2.1、描述

并从结构终点：至少两条正向弧指向该变迁。

1.2.2、伪代码

Algorithm ： Determines if it is the end point of the parallel structure.

```
isEnd(String tran)
input:
    tran: 要判断的变迁
list  ← in(tran)
if list.size>1 then
    return true
else do
    return false
end
```

1.3、判断是否遍历完指向并行结构终点的变迁

1.3.1、描述

当前遍历完的库所都有一条指向该变迁的正向弧

1.3.2、伪代码

Algorithm ： Has the transition to the end of the parallel structure been traversed.

```
isOk(List<String> places, String tranId)
input:
    places: 当前已经遍历完的库所集合
    tranId: 要判断的变迁
foreach place ∈ places do
    if tra ∉ out(place) then
        return false
    end
end
return true
```

1.4、增加当前变迁的访问次数

1.4.1、描述

根据变迁的 id 得到当前变迁的访问次数并对其执行加一操作

1.4.2、伪代码

Algorithm ： count of transition +1

```
addCount(String tranId)
input:
    tranId: 当前访问的变迁 id
index ← tranId
count ← index
count++
```

1.5、减少当前变迁的访问次数

1.5.1、描述

根据变迁的 id 得到当前变迁的访问次数并对其执行减一操作

1.5.2、伪代码

Algorithm ： count of transition -1

```
addCount(String tranId)
input:
    tranId: 当前访问的变迁 id
index ← tranId
count ← index
count--
```

1.6、以深度优先的思想，递归遍历模型

1.6.1、描述

1. 处理顺序、选择结构：以深度优先的思想，递归遍历模型
2. 处理并行结构：依次遍历并行结构起点指向的多个库所，直到遍历完指向并行结构终点的库所。
3. 处理循环结构：找出每一个可能进入循环的库所，控制对其指向的变迁的访问次数。

1.6.2、伪代码

Algorithm ： for the execution of an activity and its successors.

```
ActivityTracer(places，path)
Input：
    places：List<String> places，当前可以遍历的库所集合
    path：当前正在遍历的路径
foreach place ∈ places do
    if place=destination then
        res.add(path)
    else if place≠ destination then
        foreach tran_i ∈ out(place) do
            newPath=tran.name
            tran.count ++
            if maybeLoop and tran.count>1 then
                tran.count --
                continue
            end
            if tran→ end of AND and all of places→ tra then
                ActivityTracer(out(tran), newPath)
            else if tran→ end of AND then
                break
            else do
                ActivityTracer(out(tran)+places-place, newPath)
            end
            tran.count --
        end
    end
end
```

四、测试报告：

4.1、测试概要

针对模型中所含的不同结构分别进行测试

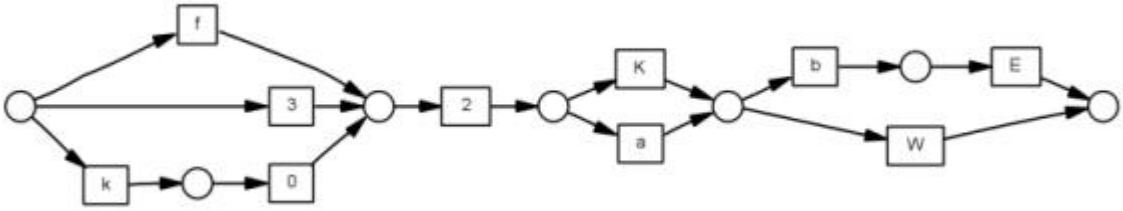
表 4-1 测试计划一览表

测试用例编号	测试内容	测试实际结果与测试计划有无误差
Model1	顺序结构、选择结构	无差别
Model2	顺序结构、选择结构、并行结构及其嵌套结构	无差别
Model3	顺序结构、选择结构、并行结构、循环结构及其嵌套结构	无差别

4.2、测试结果及发现

4.2.1、测试用例 1:

输入的过程模型 Model1：



输出结果：

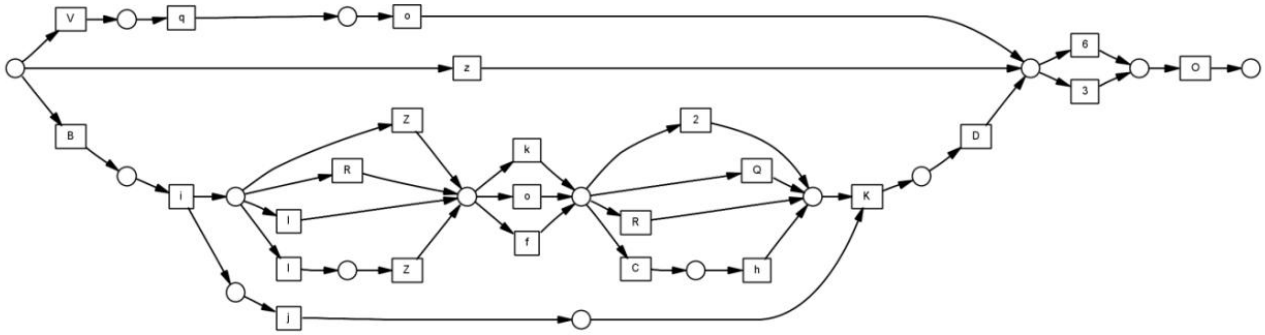
共 12 条变迁执行记录，如下：

case1: 3 2 K W
case2: 3 2 K b E
case3: 3 2 a W
case4: 3 2 a b E
case5: f 2 K W

case6: f 2 K b E
case7: f 2 a W
case8: f 2 a b E
case9: k 0 2 K W
case10: k 0 2 K b E
case11: k 0 2 a W
case12: k 0 2 a b E

4.2.2、测试用例 2:

输入的过程模型 Model2：



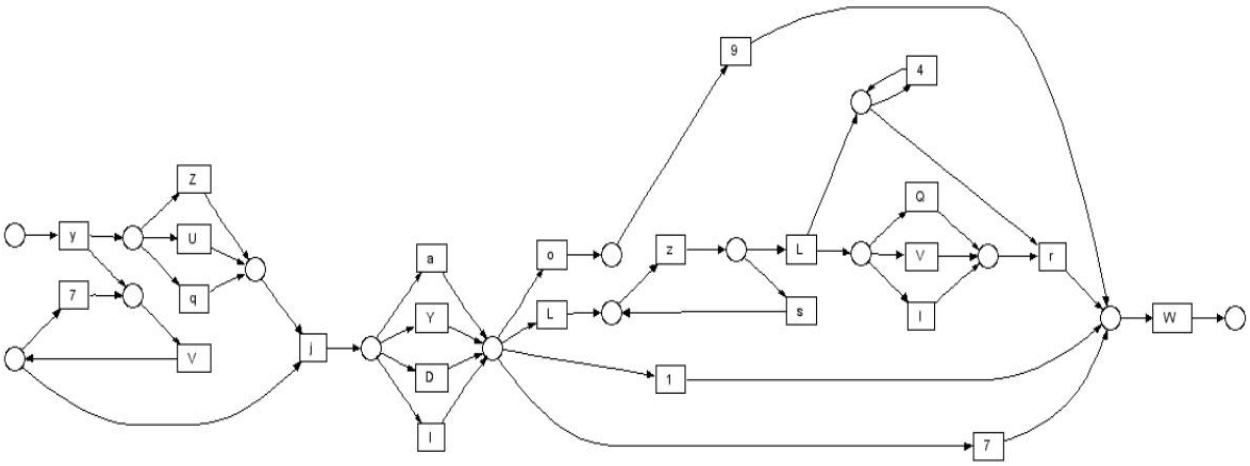
输出结果：

共 436 条，数量过多，部分如下:

case1: B i R f 2 j K D 3 O
case2: B i R f 2 j K D 6 O
case3: B i R f C h j K D 3 O
case4: B i R f C h j K D 6 O
case5: B i R f C j h K D 3 O
case6: B i R f C j h K D 6 O
case7: B i R f Q j K D 3 O
case8: B i R f Q j K D 6 O
case9: B i R f R j K D 3 O
case10: B i R f R j K D 6 O
case11: B i R f j 2 K D 3 O
case12: B i R f j 2 K D 6 O
case13: B i R f j C h K D 3 O
case14: B i R f j C h K D 6 O
case15: B i R f j Q K D 3 O

4.2.3、测试用例 3:

输入的过程模型 Model：



输出结果：

共 1512 条，数量过多，部分如下:

case1: y U V 7 V j D 1 W
case2: y U V 7 V j D 7 W
case3: y U V 7 V j D L z L 4 Q r W
case4: y U V 7 V j D L z L 4 V r W
case5: y U V 7 V j D L z L 4 I r W
case6: y U V 7 V j D L z L Q 4 r W
case7: y U V 7 V j D L z L Q r W
case8: y U V 7 V j D L z L V 4 r W
case9: y U V 7 V j D L z L V r W
case10: y U V 7 V j D L z L I 4 r W
case11: y U V 7 V j D L z L I r W
case12: y U V 7 V j D L z s z L 4 Q r W
case13: y U V 7 V j D L z s z L 4 V r W
case14: y U V 7 V j D L z s z L 4 I r W
case15: y U V 7 V j D L z s z L Q 4 r W