算术计算Petri网模型及实现

邵叱风

（安徽理工大学，安徽　合肥　232001）

摘　要：为便捷证明算术计算Petri网模型的计算能力，分析其具体计算过程.结合面向对象编程语言Java开发插件Arithmetic Petri Net Simulation(APNS)，对网中的库所、变迁、弧元素进行实例化，重写Fire方法生成自定义格式的模型运行日志；利用轻量级控件Swing实现交互界面，在模拟运行时对可触发变迁的发生进行选择，利于模型计算过程是否唯一的分析；提出(A+B)*(C−D)与A*B−C*D两个计算模型.实验对幂次方运算、(A+B)*(A−B)以及A^2−B^2模型进行模拟，对插件的可交互性、模型的可行性、幂次方模型运算过程的唯一性以及(A+B)*(A−B)与A^2−B^2模型的等价进行了分析与证明.

关键词：算术计算Petri网；面向对象；Java；Swing；日志

中图分类号：TP391.9　　**文献标识码**：A　　**文章编号**：1673-260X(2020)08-0021-04

DOI:10.13398/j.cnki.issn1673-260x.2020.08.005

0　引言

算术运算Petri网是带抑制弧的增广Petri网的一类模型举例，通过对带抑制弧的Petri网模型的理解辅助解决实际建模中的问题[1,2].文献[3]提出了一个增广Petri网模型实现乘法运算和除法运算；文献[4]提出了一个增广Petri网模型模拟乘方和开方运算的方法；以上运算模型均从理论方面证明了模型的可行性. 在计算机高速发展的今天，通过编程可以辅助科研人员进行许多数据的处理与可视化、算法的证明等繁杂工作.文献[5]利用Cpn-tools及分析代码对SAMG问题的验证进行辅助及补充，弥补人类专业知识对SAMG工作流程验证的不足；文献[6]使用PIPE对具有多个控制器攻击的SDN问题进行建模，以分析攻击者的不确定性；文献[7]使用Tina对具有时间因素的无线传感器网络丢包问题进行建模，并分析模型的有界性、可逆性等，以证明协议的正确性；以上工具均为科研工作提供了很多便捷之处以及助力，但验证的模型结构是静态的.综上所述，由于计算模型大多仅有理论证明，且如幂次方计算模型的动态性，一般仿真软件不可对其证明.文章在此开发出一个对算术计算Petri网模型的正确性、可行性进行编程化验证分析的插件.

1　算数计算的增广Petri网模型

本节首先展示幂次方运算的Petri网模型[4]，随后基于对网模型的理解提出计算模型(A+B)*(C−D)与A*B−C*D.出于篇幅限制，本文密切相关的Petri网知识见文献[2].

定义1 [1]　带抑制弧的Petri网是一个五元组$\Sigma=(S,T,F,I,M)$，其中，(S,T,F)是一个网，M是网的一个标识，$I\subset S\times T$称为抑制弧集，$I\cap F=\Phi$(即$\forall s\in S\wedge\forall t\in T,(s,t)\in F\rightarrow(s,t)\notin I$).

对于$t\in T$，如果

$\forall s\in S:(s,t)\in F\rightarrow M(s)\geq 1$，

$\forall s\in S:(s,t)\in I\rightarrow M(s)=1$，

则t在标识M有发生权（记为M[t>）.

若M[t>，则变迁t在M可以发生，t在M发生产生新的标识M'：

$$M'(s)=\begin{cases}M(s)-1, & 若(s,t)\in F\wedge(t,s)\notin F,\\ M(s)+1, & 若(s,t)\in F\wedge(s,t)\notin F,\\ M(s), & 其他.\end{cases}$$

1.1　幂次方运算Petri网模型

正此小节给出了计算x^m的增广Petri网模型如图1所示，主要思路是$x^m=x*x*x*\cdots*x$，将多个乘法运算模型通过变迁t_{4n+i}和库所$P_{5n+1+i}(i=1,2,3\cdots m-1)$进行关联，其中$t_{4n+i}$相关联的抑制弧保证了完成

收稿日期：2020-05-23

基金项目：国家自然科学基金项目(61402011，61572035)；安徽省自然科学基金项目(1508085MF111，1608085QF149)；安徽理工大学研究生创新基金项目(2019CX2068)

当前乘法运算模型的运行后在继续下一个乘法模型的计算；通过 P_{6n+2} 输入幂次方运算的底数x，限制第一个 x^2 的计算；并利用 t_{5n}，对每个乘法模型输入乘数 x；最后使用 P_{6n+1} 中 m−2 个 Token 个数限制乘法模型的关联个数为 m−1，自此完成计算 x^m 的增广 Petri 网模型，其中 P_{5n+1} 输出计算结果.

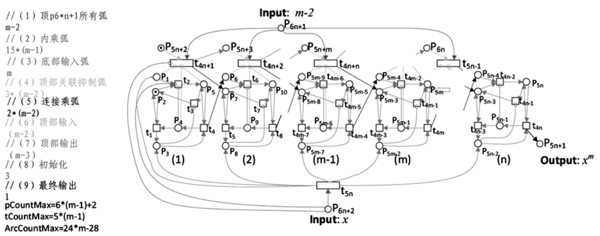
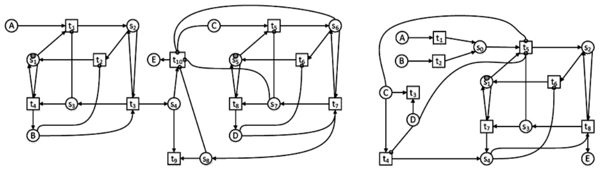


图 1　幂次方计算的 Petri 网模型

1.2　复合算术运算 Petri 网模型

此小节给出了计算 A*B−C*D=E 与(A+B)*(C−D)=E 的增广 Petri 网模型如图 2 所示，主要思路利用加减法模型与乘法模型[8]的复合生成新的计算模型，通过抑制弧的加入限制复合后模型的分块执行顺序.



a) A*B-C*D　　b) (A+B)*(C-D)

图 2　复合算术计算 Petri 网模型

通过乘法模型实现 A*B 与 C*D 的计算，用减法模型对乘法模型模型连接，加入抑制弧(C,t_{10})，(s_6,t_{10})，(s_7,t_{10})生成 A*B−C*D 算术计算模型.抑制弧的加入是为了确保 C*D 计算在 s_4−s_8 的运算开始之前完成，避免抑制弧(s_8,t_{10})因减数的缺失而失去应有抑制效果，导致复合模型中减法运算结果异常.

通过加法模型与减法模型实现(A+B)、(C−D)的运算，再利用乘法模型对加减法模型连接，加入抑制弧(C,t_5)，(D,t_5)生成最终的(A+B)*(C−D)算术计算模型.抑制弧的加入是为了(C−D)的计算在 t_5 触发之前完成，避免因乘数的异常，导致复合模型中乘法运算结果异常.

2　APNS 插件的开发以及计算模型的分析

2.1　APNS 插件的开发以及计算模型的分析

在此利用 Java 编码实现带抑制弧的增广 Petri 网的运行逻辑，Eclipse 插件 WindowBuilder 实现可交互图形界面.利用面向对象的思想，创建 Arc.java，InhibitorArc.java，Petrinet.java，Petrinet−Obj.java，Place.java，Transition.java 共六个类，对库所、变迁、流弧、抑制弧和网结构进行实例化，且对变迁是否可触发及触发规则进行了代码化（类图如图 3 所示）；并在主界面定义多个触发事件，利用 java.io.File 包实现变迁触发日志的导出；创建 Operation.java 将算术运算 Petri 网的模型代码抽象化；创建 Gui_Main.java 和 PetrinetGUI.java 利用 WindowBuilder Editor 实现插件主控、交互以及文件导出界面如图 4 所示.

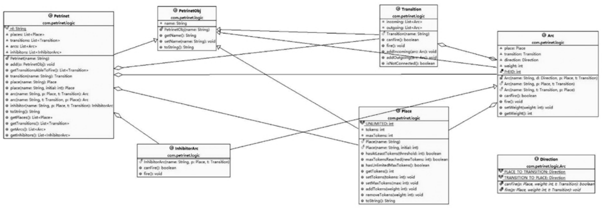
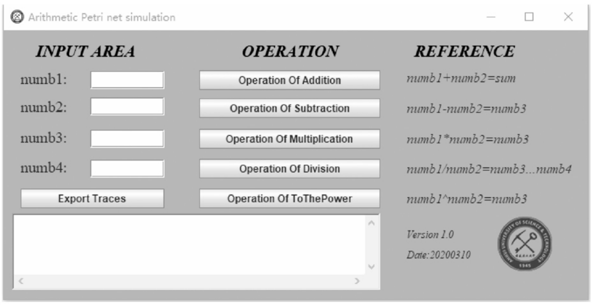
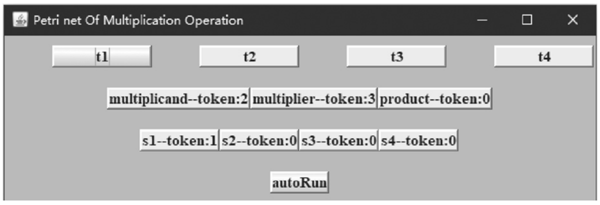


图 3　APNS 项目的 Java 类图



a)主控界面



c)计算过程交互界面

图 4　APNS 插件的部分运行界面

交互界面可以观察可触发变迁，通过点击进行触发，并在主界面生成对应触发记录；同样可以点击 autoRun 按钮，对 Petri 网中的变迁进行遍历，对当前可触发变迁利用 ArrayList 进行缓存，然后随机触发，直到当前无变迁可触发即停止运行.对应代码如 Code−1：

从插件大小、模型设定、交互运行、日志导出共 4 个方面，用此插件与 PIPE、CPNTools、Tina3 个 Petri 网仿真软件进行对比，结果如表 1 所示.插件 APNS 在实现算术 Petri 网模型仿真方面，具有体积小、可交互以及自定义日志导出的优势.由于幂次方计算模型的动态性，将在 2.2 节详细介绍其模型构建方式.

2.2　幂次方运算的实现

```
Code-1
autoRun.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int canFire = 1;
        int logLength = 0;
        while (canFire > 0) {
            ArrayList<Transition> waitRun = new ArrayList<Transition>();
            canFire = 0;
            for (int i = 0; i < pn.getTransitions().size(); i++) {
                if (pn.getTransitions().get(i).canFire()) {
                    waitRun.add(pn.getTransitions().get(i));
                    canFire++;
                }
            }
            if (waitRun.size() > 0) {
                int max = waitRun.size(), min = 1;
                int ranNum = (int) (Math.random() * (max - min) + min);
                waitRun.get(ranNum - 1).fire();
                logArea.append(waitRun.get(ranNum - 1).getName() + ",");
                logLength++;
            }
        }
        logArea.append("\r\nLogLength" + logLength);
        logArea.append("\r\n" + pn.getPlaces().toString());
        repaint();
    }
});
```

表 1　插件与仿真软件对比

名称	插件大小	模型设定	交互运行	日志导出
APNS	320KB	代码预设,动态生成模型	可以	自定义输出
PIPE	44.7MB	交互绘制静态模型	否	控制台输出
Tina	41.9MB	交互绘制静态模型	可以	无
CPNTools	46.2MB	交互绘制静态模型	可以	代码实现

此小节介绍了将 x^m 对应算术运算 Petri 网模型输入至插件的过程.在此增广 Petri 中,当网处于初始状态 M_0 时,库所 P_{6n+2} 输入 x 个 Token 用作乘法结构的乘数、库所 P_{6n+1} 输入 m−2 个 Token 用作限制乘法结构个数为 m−1.此时可计算出图中变迁总数为 5*(m−1)、库所总数为 6*(m−1)+2.

对于图 1 中流弧个数可分为 9 个部分来求取:顶部 P_{6n+1} 直接关联的所有流弧共(m−1)条;乘法增广 Petri 网结构内的所有流弧共 15*(m−1)条;底部输入弧,t_{5n} 的所有输出弧共 m 条;顶部关联抑制弧乘法结构抑制 t_{4n+h} 的 3 条抑制弧共 3*(m−2)条;连接乘法结构的流弧共 2*(m−2)条;顶部输入弧,对 t_{4n+h} 输入的流弧共(m−2)条;顶部输出流弧共(m−3)条;初始化弧共 3 条;输出流弧共 1 条,库所与变迁之前的关系弧共 24*m−28 条.

对于模型的输入代码,首先初始化库所集、变迁集以及弧集,其中重要的部分为弧集的初始化问题,部分代码如 Code−2:

```
Code-2
        for (int i = 1; i < numb2-1; i++) {
Part-1      arc[i] = pn.arc("arc"+ i, p[6*n+1], t[4*n+i]);
        }
        for (int i = 1; i < numb2; i++) {
            arc[numb2-2+15*i-14] = pn.arc("arc"+(numb2-2+15*i-14), p[5*i-4], t[4*i-2]);
            ...
Part-2      arc[numb2-2+15*i-2] = pn.arc("arc"+(numb2-2+15*i-2),□p[    i-   ] t[    i])
            arc[numb2-2+15*i-1] = pn.inhibitor("arc"+(numb2-2+15*i-1), p[5*i-1], t[4*i-2]);
            arc[numb2-2+15*i] = pn.inhibitor("arc"+(numb2-2+15*i), p[5*i-2], t[4*i-1]);
        }
        arc[16*numb2-17+numb2] = pn.arc("arc"+(16*numb2-17+numb2), t[5*n], p[1]);
        for (int i = 1; i < numb2; i++) {
Part-3      arc[16*numb2-17+i] = pn.arc("arc"+(16*numb2-17+i), t[5*n], p[5*i-2]);
        }
        for (int i = 1; i < numb2-1; i++) {
            arc[17*numb2-17+3*i-2] = pn.inhibitor("arc"+(17*numb2-17+3*i-2), p[5*i-4], t[4*n+i]);
Part-4      arc[17*numb2-17+3*i-1] = pn.inhibitor("arc"+(17*numb2-17+3*i-1), p[5*i-1], t[4*n+i]);
            arc[17*numb2-17+3*i] = pn.inhibitor("arc"+(17*numb2-17+3*i), p[5*i], t[4*n+i]);
        }
        for (int i = 1; i < numb2-1; i++) {
            arc[20*numb2-23+2*i-1] = pn.arc("arc"+(20*numb2-23+2*i-1), t[4*n+i], p[5*i+2]);
Part-5      arc[20*numb2-23+2*i] = pn.arc("arc"+(20*numb2-23+2*i), t[4*i], p[5*i+1]);
        }
        for (int i = 1; i < numb2-1; i++) {
Part-6      arc[22*numb2-27+i] = pn.arc("arc"+(22*numb2-27+i), p[5*n+1+i], t[4*n+i]);
        }
        for (int i = 1; i < numb2-2; i++) {
Part-7      arc[23*numb2-29+i] = pn.arc("arc"+(23*numb2-29+i), t[4*n+i], p[5*n+2+i]);
        }
        arc[24*numb2-32+1] = pn.arc("arc"+(24*numb2-32+1), p[6*n+2], t[5*n]);
Part-8  arc[24*numb2-32+2] = pn.inhibitor("arc"+(24*numb2-32+2), p[6*n+2], t[4*n+1]);
        arc[24*numb2-32+3] = pn.inhibitor("arc"+(24*numb2-32+3), p[6*n+2], t[2]);
Part-9  arc[24*numb2-29+1] = pn.arc("arc"+(24*numb2-29+1), t[4*n], p[5*n+1]);
```

3　实验部分

本节对(A+B)*(A−B)、A^2−B^2 以及 x^m 三个计算模型进行模拟运行,通过调整输入参数获得不同计算结果,以及输出日志.所有的测试均在配有 I5-7300HQ 2.5Ghz 四核处理器和 16GB 运存的机器上进行的,使用的 Java SE 1.7 开发环境.

首先就模型执行过程的唯一性,由于存在加、减法运算的复合,假设(A+B)*(A−B)与 A^2−B^2 运算过程不为一.设 A=3,B=2.(A+B)*(A−B)模拟运行两次获得两条长度为 28 的变迁触发日志:

$L_1=(t_2,t_1,t_1,t_2,t_1,t_3,t_3,t_4,t_5,t_8,t_6,t_7,t_5,t_8,t_6,t_7,t_5,t_8,t_6,t_7,t_5,t_8,t_6,t_7,t_5,t_8,t_6,t_7)$

$L_2=(t_1,t_3,t_3,t_4,t_2,t_5,t_1,t_1,t_2,t_8,t_6,t_7,t_5,t_8,t_6,t_7,t_5,t_8,t_6,t_7,t_5,t_8,t_6,t_7,t_5,t_8,t_6,t_7)$

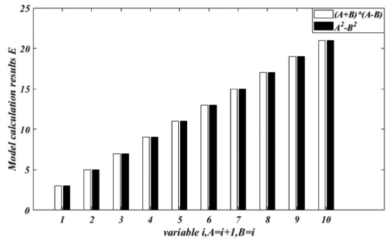
A^2−B^2 模拟运行两次获得长度为 45 的变迁触发日志:

$L_3=(t_5,t_1,t_3,t_7,t_7,t_3,t_3,t_2,t_9,t_9,t_6,t_8,t_4,t_4,t_8,t_5,t_4,t_1,t_3,t_7,t_3,t_7,t_3,t_2,t_4,t_9,t_6,t_8,t_9,t_8,t_4,t_4,t_1,t_{10},t_{10},t_3,t_{10},t_3,t_{10},t_3,t_{10},t_2,t_4,t_4,t_4)$;

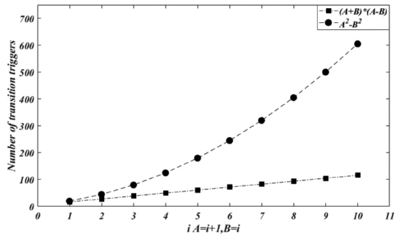
$L_4=(t_1,t_3,t_5,t_3,t_7,t_3,t_7,t_6,t_8,t_2,t_4,t_9,t_9,t_4,t_4,t_1,t_3,t_3,t_8,t_3,t_5,t_2,t_7,t_9,t_4,t_4,t_4,t_7,t_6,t_9,t_8,t_1,t_8,t_{10},t_{10},t_3,t_{10},t_3,t_3,t_{10},t_2,t_4,t_4,t_4,t_{10})$;

其次取 A=i+1,B=i(其中 i=1,2,3,4,5,6),两个计算模型均能正确计算出结果且模型结构不随参数变化,但执行效率随着 i 的增大差异愈发明显,如图 5 所示.

取底数为 x(x=2,3,4),次数为 i(i=3,4,5,6,7),对应计算的流程有且唯一,幂次方计算模型复杂度



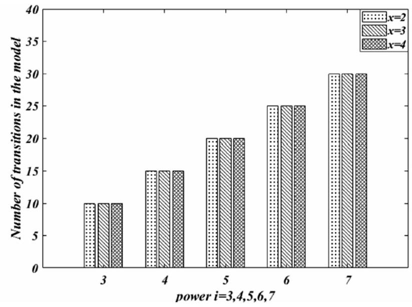
a 模型计算结果随 *i* 变化无偏差



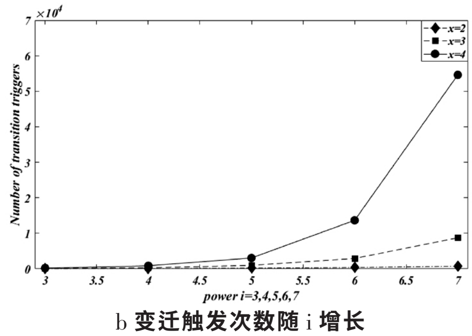
b 模型变迁触发次数随着 *i* 增大变化

图 5　(A+B)*(A-B)、A^2-B^2 计算模型模拟结果 i∈[1,10]

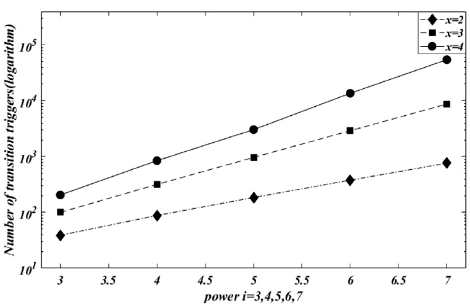
（变迁个数如图 6a 所示），以及对应变迁触发次数如图 6b 所示所示.随着次数 i 的的增加,模型中变迁个数呈线性增加;随着次数或底数的增加,模型



a 模型中变迁个数随 i 呈线性增加



b 变迁触发次数随 i 增长



c 变迁触发次数随 i 呈指数增长

图 6　变迁个数与触发次数随 i 增加的变化

计算变迁触发次数呈指数增长如图 6c 所示.

4　总结

文章通过带抑制弧 Petri 网的强模拟能力引出算术计算 Petri 网模型的构建. 使用 Java 语言开发出插件 APNS 模拟带抑制弧 Petri 网的运行，且可导出变迁触发日志用以分析运行过程;通过对现有网模型的复合,提出平方差公式的计算模型;将模型嵌入 APNS 中，模拟计算证明模型的正确性,导出变迁触发日志分析随自变量 i 的增加 A^2-B^2 计算效率高于(A+B)*(A-B);幂次方运算过程有且唯一，随次数 i 的增加模型中变迁数量线性增加，变迁触发次数呈指数增长.

已开发的插件 APNS 可有效模拟带抑制弧 Petri 网的运行，但模型的代码化输入不够常规.在未来的工作中主要是利用复合网模型的方法生成更多常用模型嵌入插件中并验证,以及增加界面化的模型输入.

参考文献：

〔1〕吴哲辉.Petri 网导论[M].北京:机械工业出版社,2006.
〔2〕邵叱风.基于流程挖掘的并行优化算法[J].赤峰学院学报(自然科学版),2019,35(10):66-70.
〔3〕吴哲辉.实现乘法计算的增广 Petri 网模型[J].山东矿业学院学报,1986,96(02):12-16.
〔4〕许安国.实现自然数 m 次乘方和开 m 次方的增广 Petri 网模型[J].山东矿业学院学报,2017,34(04):81-89.
〔5〕Lee Y S, No Y G, Seong P H. Validation of severe accident management guidelines (SAMGs) for advanced power reactor 1400 (APR1400) using colored Petri net (CPN) Tools[J]. Annals of Nuclear Energy, 2019, 126: 186-193.
〔6〕Almutairi L, Hong L, Shetty S. Security analysis of multiple SDN controllers based on stochastic Petri nets[C]//2019 Spring Simulation Conference (SpringSim). IEEE, 2019: 1-12.
〔7〕Louazani A, Sekhri L. Time Petri Nets based model for CL-MAC protocol with packet loss [J]. Journal of King Saud University-Computer and Information Sciences, 2019.