基于常微分方程的死锁检测实验分析

丁佐华[1)] 江明月[1)] 刘 静[2)]

[1)](浙江理工大学数学计算与软件工程中心 杭州 310018)

[2)](华东师范大学计算机理论研究所 上海 200062)

摘 要 用静态分析方法对并发程序进行死锁检测通常比较困难,其原因是会遇到状态空间爆炸问题.文中针对作者曾提出的一种可有效避免状态爆炸问题的死锁检测方法,进行进一步实验验证.该方法的基本框架是首先将表示并发系统的离散 Petri 网模型连续化,得到一种新的连续 Petri 网模型;在此基础上,建立系统的常微分方程模型;通过分析常微分方程组的解来检测系统中是否存在死锁.与传统方法不同点在于:该方法不需要遍历状态空间,而是分析一组常微分方程组的解.为了减少在求解常微分方程模型过程中的计算机系统的开销,作者还采取了一系列优化策略.哲学家进餐问题被用来说明死锁检测的方法.大量的实验结果说明作者所提出的方法有着较强的静态分析能力.作为副产品,这种分析方法还可以用来判定系统的有界性.

关键词 死锁检测;并发程序;状态爆炸;连续 Petri 网;常微分方程

中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2009.01736

Experiments on Detecting Program Deadlock with Ordinary Differential Equation Model

DING Zuo-Hua[1)] JIANG Ming-Yue[1)] LIU Jing[2)]

[1)](*Center of Math Computing and Software Engineering, Zhejiang Sci-Tech University, Hangzhou* 310018)

[2)](*Institute of Software Engineering, East China Normal University, Shanghai* 200062)

Abstract Detection of deadlock of concurrent programs by static analysis is in general difficult because of state-space explosion. This paper gives the feasibility test of a deadlock detection method avoiding explosion of state space entirely, which method was proposed by author, by conducting more experiments. The frame of this method is as following: Firstly a Petri net, which represents a concurrent system, is continued to a new continuous Petri net. Then based on this continuous Petri net model, a concurrent system can be modeled by a family of ordinary differential equations. Finally the system deadlocks can be checked by analyzing the solution of this family of ordinary differential equations. Thus instead of exploring reachable states as in traditional methods, the solution of a family of ordinary differential equations is analyzed. Some strategies are developed in order to optimize the method. Dining philosopher problem has been employed to illustrate the method. More experiments have been conducted, and the experimental data shows that the method has strong ability in static analysis. As a byproduct, the method can also been used to check the boundedness of a system.

Keywords deadlock detection; concurrent program; state explosion; continuous Petri net; ordinary differential equation

---

收稿日期:2009-04-19;最终修改稿收到日期:2009-07-24.本课题得到国家自然科学基金(90818013,90718014)资助.**丁佐华**,男,1964 年生,博士,教授,主要研究领域为程序分析、软件测试、服务计算等.E-mail: zouhuading@hotmail.com.**江明月**,女,1985 年生,硕士研究生,主要研究方向为程序分析、服务计算.**刘 静**,女,1965 年生,教授,博士生导师,主要研究领域为程序分析、软件模型与服务计算.

1 引　言

在对并发程序进行静态分析时,常常会遇到状态空间爆炸问题.目前解决状态空间爆炸问题主要有状态空间压缩、符号化模型检测和基于可达性的分析技术等,但是这些方法都受分析程序的规模限制,一旦程序中的并发结构增多,或其中的异步进程增多,仍然会出现状态爆炸问题.

我们认为,状态爆炸问题不能避免的根本原因是:在所有的程序分析中,状态连续变化的并发系统被建模为离散的系统.在这种情况下,要分析系统,就需要列举出系统所有的状态.因此,要想彻底解决状态爆炸问题,必须寻找一种连续系统建模方法,将目前的离散系统转化为连续系统,使用解析表达式来描述.在此基础上,就可以通过分析解析表达式的解来分析整个系统,而不需要计算每一个系统状态.

我们在分析了有限状态机、标记迁移系统(LTS)、Petri 网等形式化建模方法后,最后选择用 Petri 网来为并发系统建模.理由是 Petri 网是一个适合于为行为系统建模及分析的工具,可以很好地描述并行、资源共享、同步通信、异步通信等.

虽然 Petri 网使用了一些状态减化技巧[1],但在使用它做可达性分析的时候,仍然会遇到状态爆炸问题.解决这个问题的方法之一是使用松弛原则来消除 Petri 网的完整性约束.这种松弛原则导致了一种基于连续时间的形式化表示:连续 Petri 网[2].从离散 Petri 网转化为连续 Petri 网,是以失去某些分析的可能性为代价的,但是连续 Petri 网可以避免状态爆炸问题.

连续 Petri 网是时间 Petri 网的逼近,它的语义是由一组常微分方程定义的,其中每一个方程描述了一个给定库所中标识(marking)值的连续变化.对于连续 Petri 网来说,点火方式不同,其语义也就不一样.我们所采用的是一种连续 Petri 网,它的即时点火速率与其所有输入库所的标识值的乘积成正比例.

在连续 Petri 网的基础上,我们用 6 类常微分方程来为并发系统建模,方程组中的每一个方程描述了系统的一个状态的变化.一个状态可以用一个依赖于时间的非负实数来度量,称为状态度量,它描述程序运行过程中此状态所被到达的程度.对于一个给定的时间,程序中所有状态的度量值可以同时显现出来,我们可以通过分析这些值来检测程序中的死锁.

我们在文献[3]中证明了程序中存在死锁当且仅当每一个状态的状态度量值在时间趋于无穷时或者趋于 0 或者趋于 1.文献[3]中考虑两种类型的死锁模型:通信(OR)模型和资源共享(AND)模型.通信模型是指并发程序中的进程无共享地址空间且进程间的通信和同步是通过信息传递来完成的.资源共享模型描述的是并发程序中的进程共享一个地址空间,进程间的同步通过使用监控和条件队列来实现.本文主要是对资源共享模型来进一步做实验,来说明文献[3]中方法的可行性.经典的哲学家问题被用来作为我们的实验对象.

作为副产品,我们用该方法研究了并发系统的有界性.我们证明了并发系统的离散模型是有界的当且仅当它所对应的连续模型是稳定的.这里的连续模型稳定是指系统中所有的状态度量值趋于[0,1]内的常数.故我们可以通过常微分方程模型的解来判断系统离散模型的有界性,从而确定系统是否溢出或堵塞.

因为通常很难得到非线性常微分方程的分析解,我们用数值解代替.通过 Matlab 求解器,我们可得到相应的数值解.我们发现由此得到的计算误差不会影响到分析结果.

2 并发程序的常微分方程描述

2.1 并发程序的 Petri 网表示

对于并发程序,进程间的同步分两种情况:

(1)进程间无共享地址空间.在这种情况下,进程间的同步是通过消息传递来实现的.消息传递分为同步和异步两种.同步消息传递指发送方的进程发送消息后要等待接收一个回复信息,只有接收到回复信息后发送方进程才可以继续往下运行,而在异步消息传递方式下,发送方进程无需等待一个回复信息就可以继续往下执行.不管是同步消息传递还是异步消息传递,接收方在接收到消息之前处于锁住状态.

(2)进程间共享地址空间.在这种情况下,进程间的同步是通过使用控制机制和条件队列来完成的.由于进程间共享地址空间,进程需要依赖于其它的进程而完成运行.当一个进程需要其它进程释放的资源时,在资源得到之前此进程处于锁住状态,一旦得到所有需要的资源,进程恢复运行.

定义 1(程序状态).　一个程序状态定义为一

组变量，这组变量的值在程序运行过程中会发生变化.

定义 2(事件). 事件即为程序中的能改变程序状态的活动.

定义 3(Petri 网). 一个 Petri 网是一个有向图，它可以用一个四元组(P,T,F,M_0)来描述，其$P=\{p_1,p_2,\cdots,p_n\}$是库所(place)的集合，$T=\{t_1,t_2,\cdots,t_n\}$为变迁(transition)的集合，$F\subset(P\times T)\cup(T\times P)$为边的集合，$M_0$为初始的标识值.

在利用 Petri 网为并发程序建模时，一个库所用来表示程序的一个状态，变迁用来表示事件触发的活动，Petri 网中的初始标识值描述程序的初始状态.

事件一般被看作瞬间的. 如果要描述一个有持续时段的事件，可以用两个事件分别表示此持续事件的开始和结束，在开始和结束事件之间可以有多个其它事件发生.

在将并发程序转换成 Petri 网模型时，由于不同的语言有不同的语法、语义及并发结构，因此转换规则可能不同. 由于我们的例子是用 Ada 语言描述的，所以我们借助于文献[4]中所提出的规则. 具体细节这里省略了.

2.2 离散 Petri 网连续化

离散 Petri 网中库所的标记(token)是一个整数，一个变迁可发生的条件是它的所有输入库所中都有一个标记，一个变迁发生后，它的所有输入库所的标记值减 1 且所有的输出库所的标记值加 1.

一个并发程序可以用多个进程来表示，进程间通过输入输出库所进行交互. 假设在开始的时候，每一个进程中只有一个库所中有一个标记，即每一个进程都只有一个开始库所. 在运行过程中，如果某个库所中有一个标记，则表示进程当前运行至此库所. 如果某个输入输出库所的标记数很大，表示在缓存中等待被处理的数据很多.

离散 Petri 网中数据处理的过程如图 1(a)中所示. 一个进程含有两个库所 p_1，p_2 及一个输入库所 p_i. 假设在初始状态下，p_1 中有一个标记，表示进程正在访问此库所；p_i 中有 3 个标记，则表示缓存中有 3 个待处理的数据.

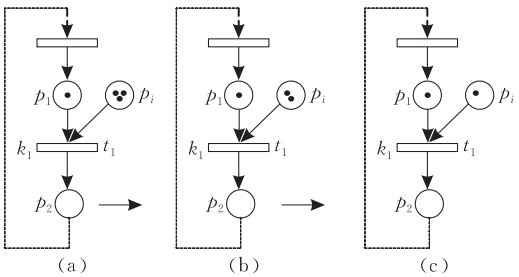

图 1 离散 Petri 网运行过程中标记值的变化

如果要将 p_i 中的所有标记都移走，p_1 将会被访问 3 次，如图 1(b)、(c)中所示，对于 p_1 来说，会有 3×1 个标记从中移走. 故当从 p_1 中移出标记时，p_i 可以被当作一个影响因子. 如果缓存中的数据很多，并且一个程序有多个这样的缓存，Petri 网模型的可到达的标识数将会很大，这将会限制离散 Petri 网的使用.

假设库所中的标记是以一种流的形式移动的，那么标记移动的速率可以表示为 $k_1\times m_1(t)\times m_i(t)$，其中 k_1 是 t_1 的最大点火速率且 $k_1=1/\delta_1$，这里是变迁 t_1 的时间延迟，可以由程序设计阶段或运行时得到. $m_1(t)$，$m_i(t)$指在时间 t 时 p_1，p_i 中的标识. 如图 2 所示.

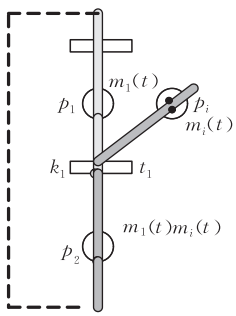

图 2 Petri 网中的连续数据流

基于这个思想，我们提出了一种新的连续 Petri 模型. 在这种模型中，一个变迁的即时触发速率是与它的所有输入库所的标识的乘积成正比.

定义 4(连续 Petri 网). 一个连续 Petri 网是一个五元组 $CPN=\langle P,T,A_{pre},A_{post},v\rangle$，其中 $P=\{p_1,p_2,\cdots,p_n\}$是库所的集合，$T=\{t_1,t_2,\cdots,t_n\}$是变迁的集合，P 与 T 都是非空集合；$A_{pre}=\{p\rightarrow t\}$是从库所指向变迁的有向弧的集合；$A_{post}=\{t\rightarrow p\}$是从变迁指向库所的有向弧的集合；$v:T\rightarrow(0,\infty)$是一个映射，每一个变迁的点火常数赋值.

定义 5. 若 $I=[0,\infty)$是一个时间区间，$m_i:I\rightarrow[0,\infty)$，$i=1,2,\cdots,n$ 是一个与库所 p_i 有关的映射，连续 Petri 网 $CPN=\langle P,T,A_{pre},A_{post},v\rangle$的标识为映射 $m_i:I\rightarrow[0,\infty]^n$，$m(s)=(m_1(s),m_2(s),\cdots,m_n(s))$.

定义 6(带标识的连续 Petri 网). 一个带标识的连续 Petri 网是一个二元组(N,M_0)，其中 N 是一个连续 Petri 网，$M_0=(m_1(0),m_2(0),\cdots,m_n(0))$是此 Petri 网的初始标识值，$m_i(0)$的值为 0 或 1.

如果在初始状态下，一个库所的标识值为 1，则它是一个开始库所. 库所的标识值可以用来衡量库

所被访问的频率.所以,我们有如下定义.

定义 7(状态度量).　给定一个时间点 $t\in[0,\infty)$,一个状态被到达的程度称为它的状态度量,用 $m(t)$来表示.一个状态的状态度量值是一个非负的实数.

m_i表示状态度量.对于一个状态来说,如果 $m(t)=1$,则说明完全到达这个状态,如果 $m(t)=0$,则说明程序没有到达这个状态.

对于一个连续 Petri 网,当一个变迁的所在输入库所的标识值都大于 0,那么这个变迁就可以发生了.

2.3　常微分方程模型

基于上面定义的连续 Petri 网的语义,对于一个连续 Petri 网,其中库所的标识值的变化可以用一个常微分方程描述,即系统状态的状态度量值的变化可以用一个常微分方程表示.根据 Petri 网的输入情况,我们将得到如下 6 类常微分方程.其中变量 x 表示系统状态,常量 d 表示变迁的点火常数,输入变迁即有外部输入库所的变迁,内部变迁即无外部输入库所的变迁.

(1)无输入

$$x_i'=d_{i-1}x_{i-1}-d_ix_i,$$

其中 x_{i-1},x_i为同一进程中的状态.

(2)输入变迁在前,内部变迁在后

$$x_i'=d_{i-1}x_{i-1}x_k-d_ix_i,$$

其中 x_{i-1},x_i为同一进程中的状态,x_k为此进程的输入.

(3)内部变迁在前,输入变迁在后

$$x_i'=d_{i-1}x_{i-1}-d_ix_ix_k,$$

其中 x_{i-1},x_i为同一进程中的状态,x_k为此进程的输入.

(4)前后均为输入变迁

$$x_i'=d_{i-1}x_{i-1}x_k-d_ix_ix_l,$$

其中 x_{i-1},x_i为同一进程中的状态,x_k,x_l为此进程的输入.

(5)异步

$$x_k'=d_ix_i-d_jx_jx_k,$$

其中 x_i,x_j为两个不同进程中的状态,x_k是两个进程间的消息.

(6)同步

$$x_k'=d_ix_ix_l-d_jx_jx_k,$$

其中 x_i,x_j为两个不同进程中的状态,x_k,x_l为两个进程间的消息.通常,x_l表示请求消息,x_k表示回复消息.

3　利用常微分方程模型检测死锁

死锁检测问题是一个广被人们研究的问题,文献[3]只考虑两种死锁模型:信息传递模型(OR)与资源共享模型(AND).

信息传递模型(OR).在这种模型中,一个并发系统由一系列的进程组成,进程间的通信通过信息传递来实现.一个进程发送一个请求信息给它所依赖的进程,并等待至少一个回复信息.对于接收到请求信息的进程,会对信息进行处理,并回复一个确认信息.发送请求的进程接收到确认信息后,会继续运行.

资源共享模型(AND).在这种模型中,进程可以请求获得一系列的资源,也可以请求获得共享的资源.在进程得到它所请求的所有资源之前,它是处于阻塞状态的.对于一个共享资源,除非它所有的共享锁都被释放了,不然它是不能被其它进程使用的.在这种情况下,死锁的产生有 4 个充要条件:(1)互斥条件;(2)请求与保持条件;(3)不剥夺条件;(4)循环等待条件.

对于以上两种类型的死锁问题,文献[3]已证明以下两个定理.

定理 1.　一个消息传递类型的程序存在死锁的充要条件是:不管点火常数如何选择,程序的每一个状态度量值在时间趋于无穷时或者趋于 0 或者趋于 1.

定理 2.　一个资源共享类型的程序存在死锁的充要条件是:不管点火常数如何选择,至少存在一组控制函数的值使得程序的每一个状态度量值在时间趋于无穷时或者趋于 0 或者趋于 1.

4　死锁检测实例分析:哲学家进餐的问题

我们选择文献[5]中哲学家进餐的问题来说明我们方法的应用效果.假设 N 个哲学家围坐在一个桌子旁边,对于每一个哲学家,都有两种选择:思考或进餐.开始的时候,有 n 把餐叉放在桌子上,每两个哲学家之间有一把.如果要进餐的话,哲学家需要把自己旁边的两把叉子都拿起来,当他吃完了,再把叉子放回原处.如果所有的哲学家都同时按照相同的顺序拿起叉子,比如说同时拿起左边的叉子,这时将会产生死锁.5 个哲学家进餐问题的 Petri 网模型如图 3 所示.

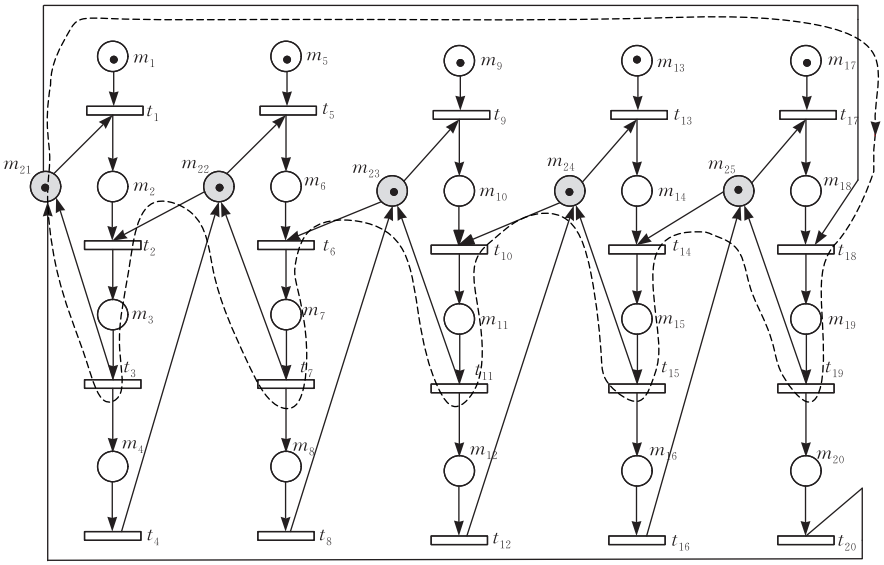

图 3 5 个哲学家进餐问题的离散 Petri 网模型

哲学家进餐问题是一个资源共享型的死锁问题，基于 2.2 节中的连续 Petri 网的语义，我们可以得到 5 个哲学家进餐问题的常微分方程模型：

$$
\begin{cases}
m_1' = m_4 - m_1 \times r_1 \times m_{21} \\
m_2' = m_1 \times r_1 \times m_{21} - m_2 \times (1-r_2) \times m_{22} \\
m_3' = m_2 \times (1-r_2) \times m_{22} - m_3 \\
m_4' = m_3 - m_4 \\
m_5' = m_8 - m_5 \times r_2 \times m_{22} \\
m_6' = m_5 \times r_2 \times m_{22} - m_6 \times (1-r_3) \times m_{23} \\
m_7' = m_6 \times (1-r_3) \times m_{23} - m_7 \\
m_8' = m_7 - m_8 \\
m_9' = m_{12} - m_9 \times r_3 \times m_{23} \\
m_{10}' = m_9 \times r_3 \times m_{23} - m_{10} \times (1-r_4) \times m_{24} \\
m_{11}' = m_{10} \times (1-r_4) \times m_{24} - m_{11} \\
m_{12}' = m_{11} - m_{12} \\
m_{13}' = m_{16} - m_{13} \times r_4 \times m_{24} \\
m_{14}' = m_{13} \times r_4 \times m_{24} - m_{14} \times (1-r_5) \times m_{25} \\
m_{15}' = m_{14} \times (1-r_5) \times m_{25} - m_{15} \\
m_{16}' = m_{15} - m_{16} \\
m_{17}' = m_{20} - m_{17} \times r_5 \times m_{25} \\
m_{18}' = m_{17} \times r_5 \times m_{25} - m_{18} \times (1-r_1) \times m_{21} \\
m_{19}' = m_{18} \times (1-r_1) \times m_{21} - m_{19} \\
m_{20}' = m_{19} - m_{20} \\
m_{21}' = m_{20} + m_3 - m_1 \times r_1 \times m_{21} - m_{18} \times (1-r_1) \times m_{21} \\
m_{22}' = m_4 + m_7 - m_2 \times (1-r_2) \times m_{22} - m_5 \times r_2 \times m_{22} \\
m_{23}' = m_8 + m_{11} - m_6 \times (1-r_3) \times m_{23} - m_9 \times r_3 \times m_{23} \\
m_{24}' = m_{12} + m_{15} - m_{10} \times (1-r_4) \times m_{24} - m_{13} \times r_4 \times m_{24} \\
m_{25}' = m_{16} + m_{19} - m_{14} \times (1-r_5) \times m_{25} - m_{17} \times r_5 \times m_{25}
\end{cases}
$$

m 的初始值为 $m_1(0)=1, m_5(0)=1, m_9(0)=1, m_{13}(0)=1, m_{17}(0)=1, m_{21}(0)=1, m_{22}(0)=1, m_{23}(0)=1, m_{24}(0)=1, m_{25}(0)=1$，其它的 m 值为 0. 方程组中的 r_1, r_2, r_3, r_4, r_5 是依赖于时间的控制函数，在某个时刻它们的取值为 0 或 1. 如果有死锁，那么从那时刻起所有的 r 值将会固定. 因此我们只需考虑 $t=0$ 时的情况. 根据 r 的不同取值，将会有 2^5 种常微分方程组. 在某些 r 值的组合下，所得到的微分方程解是相似的，故我们只选择一部分 r 值组合，利用 Matlab 对相应的常微分方程组进行求解，并得到其解的曲线. 在这里，我们考虑 $(r_1, r_2, r_3, r_4, r_5)=(1,0,0,0,0)$ 及 $(r_1, r_2, r_3, r_4, r_5)=(1,1,1,1,1)$ 这两种情况，它们的解的曲线分别如图 4、图 5 所示.

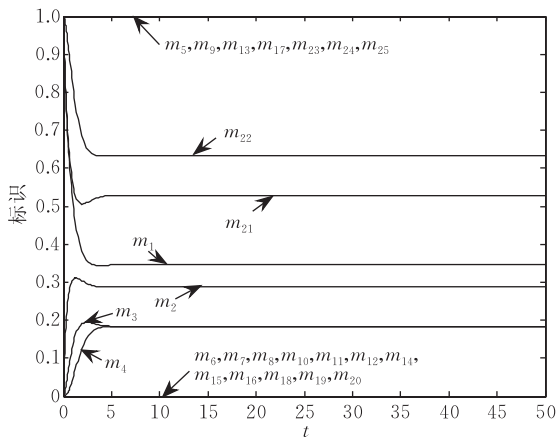

图 4 $(r_1, r_2, r_3, r_4, r_5)=(1,0,0,0,0)$ 时解的曲线

通过观察常微分方程组的解，我们可以发现，在 $(r_1, r_2, r_3, r_4, r_5)=(1,1,1,1,1)$ 的情况下，方程组

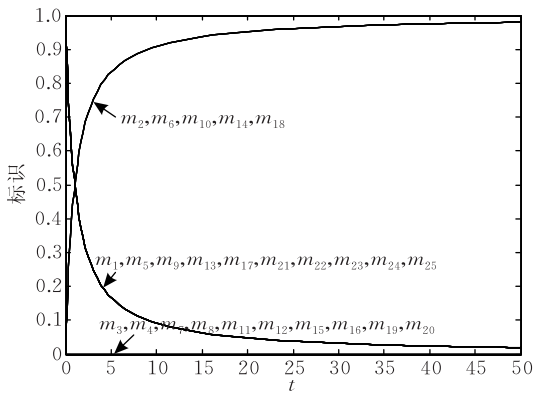

图 5　$(r_1,r_2,r_3,r_4,r_5)=(1,1,1,1,1)$ 时解的曲线

的解只存在两种情况:趋于 0 或趋于 1.依据我们前面定理 2 可知,在这种情况下,系统中存在死锁.

相关实验及结果分析

利用常微分方程模型,可以通过分析方程的解来判断系统中的死锁,但是当系统比较大时,它的微分方程模型会变得复杂.

若一个系统由 k 个进程组成,每个进程中的状态数为 $n_i(i=1,2,\cdots,k)$,那么此系统的可达状态为 $n_1n_2\cdots n_k$ 个.也就是说,系统状态空间会随着系统进程数的增多而呈指数级增长.我们将离散的系统连续化,得到系统相应的常微分方程组模型,此常微分方程组将含有 $n=n_1+n_2+\cdots+n_k$ 个常微分方程.如果此常微分方程模型不含有控制函数,那么只需要解一个含有 n 个方程的常微分方程组,此种情况下常微分方程模型的复杂度增加得较慢.但是如果常微分方程模型中含有控制函数,假设控制函数的个数为 m,则将会有 2^m 种常微分方程组,我们需要对这 2^m 种常微分方程组求解,进而分析系统的状况.可以看到,此时常微分方程模型的规模呈指数增长.这时的常微分方程模型变得很复杂,利用它来分析死锁情况开销会很大.

为了解决这个问题,我们采用了多线程方法及条件检查技术,在保证可以完全分析系统的前提下,尽量地减小系统的开销.

(1)利用多线程来解常微分方程组.

为了使系统的开销变小,我们不再依据每一个控制函数值的组合去解常微分方程组,而是将控制函数值的组合分成若干组,然后用一个线程去计算一组控制函数值的组合对应的常微分方程组.定义线程个数及相应的 m 文件中控制函数值的处理规则如下(假设控制函数组合为 $r=(r_1,r_2,\cdots,r_k)$):

①线程的个数为 $2^m(m<k)$.

②若线程个数为 2^m,则对应的 m 文件中 r_1 到 r_m 通过传递进来的参数得到值,r_{m+1} 至 r_k 通过循环得到相应的 0、1 组合.具体的方法如下:

在线程中先调用 Matlab 引擎,然后调用在 Matlab 中所写的函数.

```
UINT threadi(LPVOID lparam)
{
  Engine *ep1;
  if(!(ep1=engOpen(NULL)))
  {
    total++;
    //标记已运行线程的个数
    return -1;
  }
  engEvalString(ep1,"test20_1(0)");
  //调用 Matlab 函数,并通过参数传值
  total++;
  engClose(ep1);
  return 1;
}
```

如果有 2^m 个线程,则分别调用 Matlab 中的函数,所传递的参数值为 $0\sim2^m-1$.

在 Matlab 中定义函数,给 r 赋值,并调用 ode45 来解相应的常微分方程组.下面以 *number* 为 64,控制函数个数为 10 为例,来说明我们的方法.

```
function test10n(number) /* number 传递具体的值给
                            函数 */
/* 在线程个数为 64 的情况下,r1~r6 通过得到的参
   数取值 */
{
  r1=mod(number,2);
  number=fix(number/2);
  r2=mod(number,2);
  number=fix(number/2);
  r3=mod(number,2);
  number=fix(number/2);
  r4=mod(number,2);
  number=fix(number/2);
  r5=mod(number,2);
  number=fix(number/2);
  r6=mod(number,2);
/* r7~r10 通过循环遍历所有的值 */
  for r7=0:1
   for r8=0:1
    for r9=0:1
     for r10=0:1
      [t,m]=ode45(@phexp10,[0 50],[1,0,0,0,
      1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,
      1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,1,1,1,
      1,1,1,1,1,1],[],[r1,r2,r3,r4,r5,r6,r7,r8,
      r9,r10]);/* 对应一个 r 值组合,调用 ode45 求
```

```
        解相应的微分方程组 * /
      end
    end
  end
end
}
```

利用这种方法，我们依次对 5 个、10 个、20 个及 30 个哲学家进餐的常微分方程模型进行计算. 对于 5 个哲学家进餐的问题，我们使用了 0 个线程，10 个哲学家进餐问题使用了 16 个线程，20 个哲学家进餐问题使用了 64 个线程，30 个哲学家进餐问题使用了 127 个线程. 计算所用时间和内存的情况如表 1 所示.

表 1 使用多线程方法处理常微分方程组模型的实验数据

哲学家数	每组方程数	组数	时间/s	占内存量/KB
5	25	2^5	0.9754	29.328
10	50	2^{10}	39.0067	57.528
20	100	2^{20}	17192.9190	1822.8480
30	150	2^{30}	44090.5236	7255.296

从表 1 中可以看出，当哲学家个数增多时，处理所消耗的时间和内存仍然很大. 导致这种结果的原因有两个：① 虽然我们使用了多个线程来解常微分方程组，但是解方程组的动作不是在线程中完成的，线程只是调用 Matlab 引擎及相应的 m 文件，实质上解方程是在 Matlab 中完成的. 虽然线程可以并行运行，但是在同一时间，只有一个线程的 Matlab 引擎在计算，其它线程在等待 Matlab 引擎被其它线程释放并为自己获得；② 随着哲学家个数的增多，控制函数也相应增多，哲学家进餐问题的常微分方程模型中的方程个数是基于控制函数个数呈指数增长的.

为了进一步地减少系统开销，我们引进了条件检查技术.

(2) 使用条件检查技术.

条件检查技术的主要思想是：针对每一组控制函数值解微分方程组，在解微分方程组之前，依据一定的条件对控制函数值的组合进行筛选，只取出会导致系统产生死锁的控制函数值的组合，然后基于这些控制函数值的组合，再对微分方程组求解.

经过筛选后，控制函数值组合的个数不随控制函数的个数呈指数增长，也就是说，常微分方程模型中的方程个数不再存在指数增长的问题.

下面以哲学家进餐问题为例，说明筛选条件的确定. 如果有死锁，那么从那时刻起所有的控制函数值将会固定. 因此我们只需考虑 $t=0$ 时的情况.

① 假设 k 是一个控制函数在 $t=0$ 时的值，若 $k=1$ 表示为条件 $cond$，则 $k=0$ 表示为 $\neg cond$.

② k_1^i、k_2^i 分别为两个控制函数在 $t=0$ 时的值(哲学家 i 旁边的两个控制函数)，p_i 表示哲学家 i. $cond_1^i$ 表示 p_i 获得资源 r_i，$cond_2^i$ 表示 p_i 获得资源 r_{i-1}.

③ 哲学家进餐问题产生死锁的条件是所有的哲学家都同时都按照相同的顺序拿起叉子. 对于其中的一个哲学家来说，就是拿完一边的叉子后，再拿另一边的叉子时，但如果此时已经没有叉子可以拿，就会处于等待状态. 也就是说，每一个哲学家不能同时拿到左、右两边的叉子，依据(2)中的描述可知，$cond_1^i$、$cond_2^i$ 不能同时为真. 在这种情况下，我们称 $cond_1^i$、$cond_2^i$ 不兼容. $cond_1^i$、$cond_2^i$ 不兼容即 $k_1^i==1$ 与 $k_2^i==1$ 不可能同时为真，也就是说 $k_1^i+k_2^i==2$ 不可能成立.

故我们可以通过 $k_1^i+k_2^i==2$ 这个表达式来对 k_1^i、k_2^i 的值的组合进行筛选，被去掉的组合值有(0,0)，(0,1)，(1,0)，最后得到的组合值为(1,1)，而此组合值恰恰是使得死锁产生的控制函数的值.

依据以上方法，可以确定一个系统的控制函数组合值的筛选条件，从而去除不必要的控制函数值的组合，只留下必要的组合，以作为下一步解常微分方程组的参数.

基于条件检查技术，我们以哲学家问题为例，分别处理了 5 个、10 个、20 个、30 个、100 个、200 个及 400 个哲学家进餐问题，系统开销如表 2 所示.

表 2 使用条件检查技术解决哲学家问题的实验数据

哲学家数	组合数	占内存量/MB	整个时间/s
5	2^5	0.029	0.016
10	2^{10}	0.057	0.021
20	2^{20}	0.113	0.03
30	2^{30}	0.169	0.05
100	2^{100}	4.01	0.179
200	2^{200}	9.37	2.53
400	2^{400}	20.55	4.87

如表 2 中所示，当哲学家个数增加到 400 时，处理问题所需的时间及内存仍然是很小的，所以使用这种方法，模型的规模不再存在指数增长的问题，可以对大型的问题进行处理.

(3) 相关实验数据比较.

我们使用其它工具对哲学家进餐问题进行处理，并将它们的实验数据与我们的数据进行比较. 这

里我们主要比较不同方法处理问题所消耗的时间与内存.

① SPIN

SPIN 是一个传统的模型检测工具,它的输入语言是 Promela 语言.通过把我们的 Petri 网模型转换成由 Promela 语言描述的模型,我们就可以使用 SPIN 对系统进行模拟及验证了.在此实验中,我们所使用的计算机的配置为 Intel(R) Xeon(R) with CPU E5410@2.33GHz and memory 12GB.

首先,我们采用 exhaustive 的方法对哲学家进餐问题进行验证,结果如表 3 所示.

表 3　使用 SPIN 处理哲学家问题(exhaustive)的实验数据

哲学家数	占内存数/MB	状态数	时间/s
5	2.696	4201	0.009
6	4.454	2.7×10^4	0.053
7	16.856	1.7×10^5	0.370
8	106.114	1.2×10^6	2.780
9	776.583	7.7×10^6	87.800
10	9318.996	9.24×10^7	1053.600

如表 3 中所示,当处理 10 个哲学家进餐问题时,所用的时间和内存已经很大,当我们使用这种方法对 11 个哲学家进餐问题进行处理时,SPIN 提示系统内存不足,不能继续运行.

基于这种情况,我们采用 supertrace/bitstate 的方式对系统进行验证,实验结果如表 4 所示.

表 4　使用 SPIN 处理哲学家问题(supertrace/bitstate)的实验数据

哲学家数	占内存量/MB	状态数	时间/s
5	16.539	4201	0.0114
10	16.539	3×10^7	91.9000
13	16.539	3.7×10^7	124.0000
14	16.539	57	0.0070
20	16.539	125	0.0520
30	16.539	235	0.0880
100	16.539	1005	0.3630
200	18.437	2135	4.5900
400	38.621	4225	9.6300

从表 4 中数据可以看到,在 supertrace/bitstate 的处理方式下,SPIN 处理哲学家进餐问题消耗的内存与时间差不多是使用我们的方法的两倍.

② PAT

使用 PAT 对哲学家进餐问题进行处理,所需的系统内存与时间如表 5 所示.

表 5　使用 PAT 处理哲学家进餐问题的实验数据

哲学家数	占内存量/MB	状态数	时间/s
5	0.196	53	0.001
10	0.614	138	0.002
20	1.496	383	0.010
100	9.497	5943	1.070
200	51.145	21893	9.110
400	365.731	83794	113.700

与我们的实验结果相比,当哲学家个数增多时,PAT 所需的时间和内存增长较快.

③ TOTAL

在文献[6]中,他们以哲学家进餐问题为例,使用 TOTAL 对系统进行检验.他们所使用的哲学家模型中,每个哲学家中只含有 3 个状态.在实验过程中,他们使用了一些优化方法如 net reduction, stubborn sets 等来减小运行过程中的状态空间.使用 TOTAL 处理哲学家进餐问题所需的时间如表 6 所示.

表 6　使用 TOTAL 处理哲学家进餐问题的实验数据

哲学家数	方法	时间/s
5	Sym/sl	164/2
10	St	58
20	Nrt	626
100	Nrt+sl	27
200	Nrt+sl	111
400	Nrt+st+sym	681

当任何优化方法都没有使用时,他们最多只能处理 3 个哲学家进餐问题,当使用了 sysmmetry 或者 sleep sets 方法时,可以处理 5 个哲学家进餐问题.当使用了 stubborn sets 方法时,可以处理 10 个哲学家进餐问题.当使用了 net reduction 方法时,可以处理 20 个哲学家进餐问题.如果把所有的优化方法结合起来使用,他们的方法最多可以处理 400 个哲学家进餐问题.从表中数据可以看到,他们使用 TOTAL 处理哲学家进餐问题所需的时间仍然较多.

基于上面的实验数据,我们可以看出,我们的方法可以减少处理过程中所用的内存和时间.

5　系统有界性判断

一个离散系统如果是无界的,则说明系统中存在一些问题,如拥塞问题,影响了系统的效率.通常情况下,要判断一个离散系统是不是有界的,需要通过生成系统的可覆盖性树,根据系统的可覆盖树来判断该系统的有界性.

利用可覆盖性树可以判断一个离散系统的有界性,但是这种方法的工作量比较大,当离散系统很大时,将会花费很长的时间去生成可覆盖性树,很难由此进行性能分析.

针对这个问题,我们可以分析常微分方程的解来确定离散 Petri 网的有界性.我们有如下定义.

定义 8.　当 $t\to+\infty$ 时,若系统中所有的状态度量值趋于非负有界数,则此系统是稳定的.

我们证明了以下定理.

定理 3. 一个离散 Petri 网是有界的当且仅当它所对应的连续 Petri 网是稳定的.

证明概要.

必要性. 假设离散 Petri 网是有界的,也就是说,所有可到达的标识是有界的. 这里我们考虑两种情况:(1) 系统中不存在控制函数;(2) 系统中含有控制函数.

第 1 种情况,系统中不存在资源共享. 假设有 n 个进程中的库所:$p_1,p_2,\cdots,p_n$ 及 m 个连接库所 $c_1,c_2,\cdots,c_m$. 由于所有的可到达的标识为有界的,故系统的可到达状态数是有限的. 在 $n+m$ 维空间中,每一个标识组成了一个向量. 这些向量组成了一个多面体 G. 由于进程中的每一个库所最多可以获得一个标记,因此多面体有 n 个长度为 1 的边. 多面体的其它 m 个边的长度分别为 $l_1,l_2,\cdots,l_m$. 如图 6 所示. 我们将证明连续 Petri 网中所有的状态度量也是有界的. 由文献[7]知,进程中所有库所的状态度量都有一个界限 1,故我们只需证明连接状态的有界性.

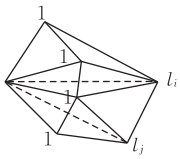

图 6 可达标识的多面体

令 m_1 为进程 P_1 的一个状态度量,m_2 为进程 P_2 的一个状态度量,m 为一个连接状态,如图 7 所示. 状态度量 m 可以通过下式计算:

$$m'=d_1m_1-d_2m_2m,\quad m(0)=l_i\neq 0.$$

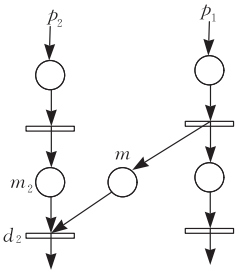

图 7 连接状态

由文献[7]知,$m_1(t),m_2(t)\in[0,1]$,假设 $m_1(t)\leq\alpha_1$ 且 $m_2(t)\geq\beta_1$. 同样,由文献[7]可知,由于进程 P_2 的输入库所 m 为正数,可以得到 $\beta_1\neq 0$. 故 m 可以被估计为

$$\begin{aligned}m(t)&=\int_0^t d_1m_1(s)\mathrm{e}^{-\int_s^t d_2m_2(x)\mathrm{d}x}\mathrm{d}s+l_i\mathrm{e}^{-\int_0^t d_2m_2(s)\mathrm{d}s}\\&\leq\int_0^t d_1\alpha_1\mathrm{e}^{-\int_s^t d_2\beta_1\mathrm{d}x}\mathrm{d}s+l_i\mathrm{e}^{-\int_0^t d_2\beta_1\mathrm{d}s}\\&=\frac{d_1\alpha_1}{d_2\beta_1}(1-\mathrm{e}^{-d_2\beta_1t})+l_i\mathrm{e}^{-d_2\beta_1t}.\end{aligned}$$

如果用 Ω 来表示所有的状态度量,则 Ω 是有界的. 如图 8 中所描述. 如果在 m 中,输入标识总是比输入标识小,则有 $d_1\alpha_1\leq l_id_2\beta_1$,即 $\frac{d_1\alpha_1}{d_2\beta_1}\leq l_i$,则上面对 m 的估计可以进一步推导得到 $m(t)\leq l_i(1-\mathrm{e}^{-d_2\beta_1t})+l_i\mathrm{e}^{-d_2\beta_1t}=l_i$.

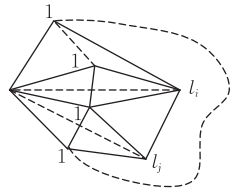

图 8 状态度量的范围

在这种情况下,Ω 位于多面体 G 内部.

第 2 种情况,系统中包含资源共享. 假设系统中有 l 个控制函数:$k_1(t),k_2(t),\cdots,k_l(t)$. 由于在系统运行过程中存在如下可能:第一次资源共享源被分配给一个进程但第二次资源却被分配给另一个进程. 为了解决方程模型中的这个问题,在$[t_1,t_2]$时间段,使用一组控制值所得到的方程模型的解将作为$[t_2,t_3]$时间段使用另一组控制值的方程模型的初始值. 因此,我们需要证明在所有情况下,方程组的解是有界的.

令 m 为一个资源,则 m 的状态度量可以通过下式计算:

$$\begin{aligned}m'=&d_1m_1+d_3m_3-d_2k(t)m_2m-\\&d_4(1-k(t))m_4m,(\cdots,m(0),\cdots)\in\Omega.\end{aligned}$$

由于 $m_1(t),m_2(t),m_3(t),m_4(t)\in[0,1]$,我们假设 $m_1(t)\leq\alpha_1,m_3(t)\leq\alpha_2$ 且 $m_2(t)\leq\beta_1,m_4(t)\leq\beta_2$. 由于 P_1 或 P_2 的输入库所 m 的状态度量为正值,这里 β_1、β_2 不会同时为 0. 因此,m 的值可以被估计为

$$\begin{aligned}m(t)=&\int_{t_0}^t(d_1m_1(s)+\\&d_3m_3(s))\mathrm{e}^{-\int_s^t(d_2k(x)m_2(x)+d_4(1-k(x))m_4(x))\mathrm{d}x}\mathrm{d}s+\\&m(0)\mathrm{e}^{-\int_{t_0}^t(d_2k(s)m_2(s)+d_4(1-k(s))m_4(s))\mathrm{d}s}\\\leq&\int_{t_0}^t(d_1\alpha_1+d_3\alpha_2)\mathrm{e}^{-\int_s^t(d_2k(x)\beta_1+d_4(1-k(x))\beta_2)\mathrm{d}x}\mathrm{d}s+\\&m(t_0)\mathrm{e}^{-\int_0^t(d_2k(s)\beta_1+d_4(1-k(s))\beta_2)\mathrm{d}s}\\\leq&\int_{t_0}^t(d_1\alpha_1+d_3\alpha_2)\mathrm{e}^{-\int_s^t\min\{d_2\beta_1,d_4\beta_2\}\mathrm{d}x}\mathrm{d}s+\\&m(t_0)\mathrm{e}^{-\int_0^t\min\{d_2\beta_1,d_4\beta_2\}\mathrm{d}s}\\=&\frac{d_1\alpha_1+d_3\alpha_2}{\min\{d_2\beta_1,d_4\beta_2\}}(1-\mathrm{e}^{-\min\{d_2\beta_1,d_4\beta_2\}(t-t_0)})+\\&m(t_0)\mathrm{e}^{-\min\{d_2\beta_1,d_4\beta_2\}(t-t_0)}\\<&\frac{d_1\alpha_1+d_3\alpha_2}{\min\{d_2\beta_1,d_4\beta_2\}}+m(t_0).\end{aligned}$$

由于 β_1、β_2 不会同时为 0,因此,可知 $m(t)$ 是有界的.

充分性. 假设所有的状态度量都是有界的,且界为 L,我们证明系统的离散 Petri 网模型是有界的. 如果离散 Petri 网不是有界的,则至少有一个库所是无界的. 假设这个无界的库所是一个连接库所 p,则随着 Petri 网的执行,p 的标识将会无限增大. 由于每一个变迁与一个时间间隔相关联,p 的标识在时间段 $t_0,t_1,t_2,\cdots,t_n\cdots$ 被计算,得到 p 的一系列标识值:$m(p,t_0),m(p,t_1)\cdots$,这些标识值一直无限增大. 如果库所在 t_n 时刻开始被连续,则考虑初值问题:

$$m'=d_1m_1-d_2m_2m,\ (\cdots,m(t_n)=m(p,t_n),\cdots).$$

由于 $m_1(t)\geq\alpha_2$ 且 $m_2(t)\leq\beta_2$,则 m 可以被估计为

$$\begin{aligned}m(t)&=\int_{t_0}^{t}d_1m_1(s)\mathrm{e}^{-\int_s^t d_2m_2(x)\mathrm{d}x}\mathrm{d}s+m(t_n)\mathrm{e}^{-\int_{t_n}^t d_2m_2(s)\mathrm{d}s}\\&\geq\int_{t_n}^{t}d_1\alpha_2\mathrm{e}^{-\int_s^t d_2\beta_2\mathrm{d}x}\mathrm{d}s+m(t_n)\mathrm{e}^{-\int_{t_n}^t d_2\beta_2\mathrm{d}s}\\&=\frac{d_1\alpha_2}{d_2\beta_2}(1-\mathrm{e}^{-d_2\beta_2(t-t_n)})+m(t_n)\mathrm{e}^{-d_2\beta_2(t-t_n)}.\end{aligned}$$

令 $t_n<t<t_{n+1}$,则 $t\to\infty\leftrightarrow t_n\to\infty$. 故有

$$\begin{aligned}\lim_{t\to+\infty}m(t)&=\lim_{t\to+\infty}\left(\frac{d_1\alpha_2}{d_2\beta_2}(1-\mathrm{e}^{-d_2\beta_2(t-t_n)})\right)+m(p,t_n)\mathrm{e}^{-d_2\beta_2(t-t_n)}\\&\geq\lim_{t\to+\infty}\left(\frac{d_1\alpha_2}{d_2\beta_2}(1-\mathrm{e}^{-d_2\beta_2(t-t_n)})+m(p,t_n)\mathrm{e}^{-d_2\beta_2(t_{n+1}-t_n)}\right)\\&=\frac{d_1\alpha_2}{d_2\beta_2}+\lim_{t_n\to+\infty}m(p,t_n)\mathrm{e}^{-d_2\beta_2(t_{n+1}-t_n)}\\&\geq\frac{d_1\alpha_2}{d_2\beta_2}+\lim_{t_n\to+\infty}m(p,t_n)\mathrm{e}^{-d_2\beta_2T}\\&=\infty.\end{aligned}$$

从而得到矛盾的结果,故定理成立.　　证毕.

根据该定理,我们可以通过判断连续 Petri 网的稳定性来判断离散 Petri 网的有界性,也即可以通过分析常微分方程组的解来判断离散系统的有界性.

6　有界性判断实例分析:交通模型

本节以拥有一个交叉口的交通模型为例[8-9],对它的有界性进行判断,并做一些简单分析.

单交叉口的交通模型的连续 Petri 网模型如图 9 所示. 其中,d_1、d_2、d_3 和 d_4 分别代表四相位的绿灯变换速率,$d_i(i=5,\cdots,36)$ 为交通流速率,m_1、m_2、m_3 和 m_4 分别代表四相位的绿灯状态,m_5、m_6、m_7 和 m_8 为交互库所,m_{11}、m_{15}、m_{19} 和 m_{23} 分别表示等

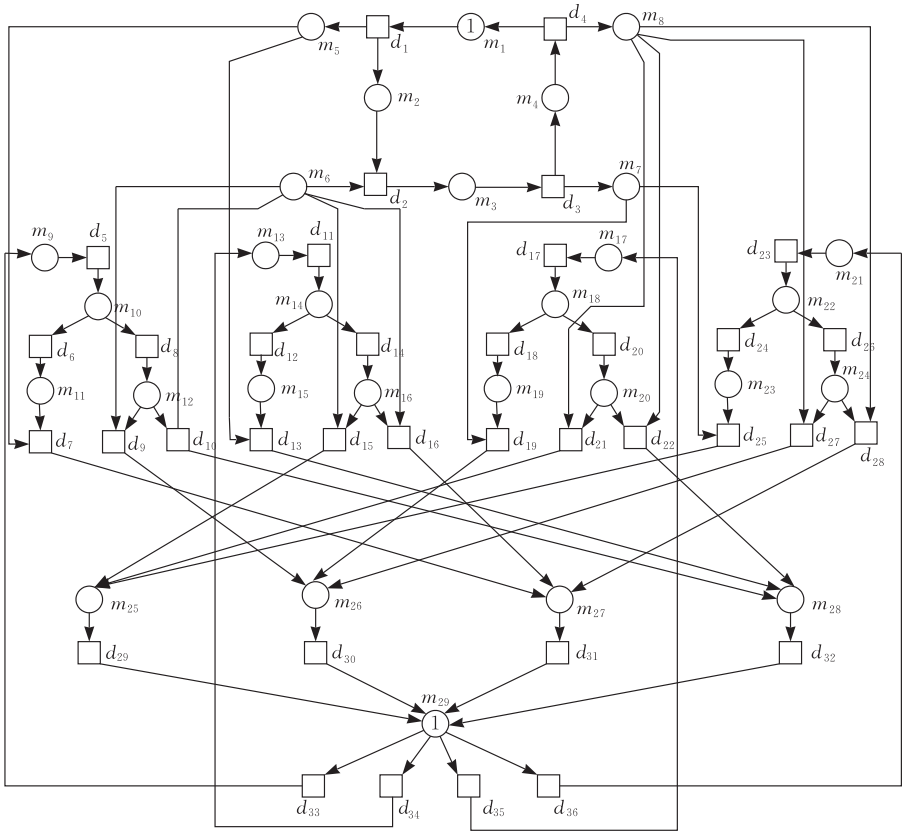

图 9　单交叉口交通模型的连续 Petri 网模型

待左转的交通流，m_{12}、m_{16}、m_{20}和m_{24}分别表示等待右转或直行的交通流.我们用m_{29}抽象地表示除此交叉口及各支路以外的整个外部交通网络.本文假设d_i(信号灯变换速率及交通流速率)值为1.

在此我们只考虑信号灯变换速率的影响(不考虑交通流速率).由于Petri网中的状态具有选择结构，其状态对于路径的选择是等可能的，若要判断系统的稳定性，需要遍历系统所有的可能性，故在此情况下，可以把一个选择结构的两种选择当作是等同可能的，因此可设其状态对应的控制函数值为0.5.

根据2.2节定义的连续Petri网的语义，建立系统的常微分方程模型如下.

$$
\begin{cases}
m_1'=m_4-m_1\\
m_2'=m_1-m_2\\
m_3'=m_2-m_3\\
m_4'=m_3-m_4\\
m_5'=m_1-0.5\times m_5\times m_{11}-0.5\times m_5\times m_{15}\\
m_6'=m_2-0.25\times m_6\times m_{12}-0.25\times m_6\times m_{16}\\
m_7'=m_3-0.5\times m_7\times m_{19}-0.5\times m_7\times m_{23}\\
m_8'=m_4-0.25\times m_8\times m_{20}-0.25\times m_8\times m_{24}\\
m_9'=0.25\times m_{29}-m_9\\
m_{10}'=m_9-m_{10}\\
m_{11}'=0.5\times m_{10}-0.5\times m_5\times m_{11}\\
m_{12}'=0.5\times m_{10}-0.25\times m_6\times m_{12}\\
m_{13}'=0.25\times m_{29}-m_{13}\\
m_{14}'=m_{13}-m_{14}\\
m_{15}'=0.5\times m_{14}-0.5\times m_5\times m_{15}\\
m_{16}'=0.5\times m_{14}-0.25\times m_6\times m_{16}\\
m_{17}'=0.25\times m_{29}-m_{17}\\
m_{18}'=m_{17}-m_{18}\\
m_{19}'=0.5\times m_{18}-0.5\times m_7\times m_{19}\\
m_{20}'=0.5\times m_{18}-0.25\times m_8\times m_{20}\\
m_{21}'=0.25\times m_{29}-m_{21}\\
m_{22}'=m_{21}-m_{22}\\
m_{23}'=0.5\times m_{22}-0.5\times m_7\times m_{23}\\
m_{24}'=0.5\times m_{22}-0.25\times m_8\times m_{24}\\
m_{25}'=0.5\times m_7\times m_{23}+0.125\times m_6\times m_{16}+0.125\times m_8\times m_{20}-m_{25}\\
m_{26}'=0.5\times m_7\times m_{19}+0.125\times m_6\times m_{12}+0.125\times m_8\times m_{24}-m_{26}\\
m_{27}'=0.5\times m_5\times m_{11}+0.125\times m_6\times m_{16}+0.125\times m_8\times m_{24}-m_{27}\\
m_{28}'=0.5\times m_5\times m_{15}+0.125\times m_6\times m_{12}+0.125\times m_8\times m_{20}-m_{28}\\
m_{29}'=m_{25}+m_{26}+m_{27}+m_{28}-m_{29}
\end{cases}
$$

利用Matlab解常微分方程组，所得的解的曲线如图10所示.

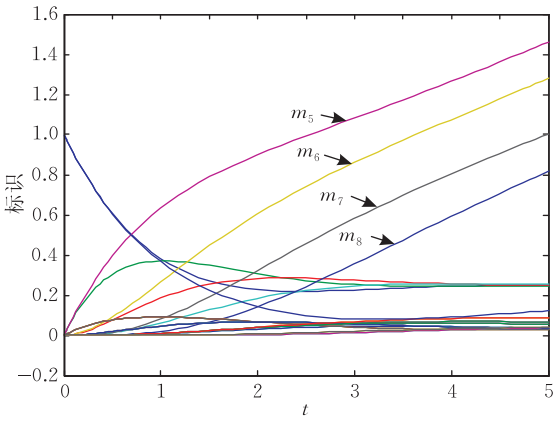

图10 交通模型的常微分方程模型的解

从图10中可以看到，当$t\to+\infty$时，m_5、m_6、m_7和m_8的值远远大于1，说明连续Petri网是不稳定的，从而对应的离散Petri网无界，所以系统发生拥塞现象.这与现实中的情形是吻合的.因为m_5、m_6、m_7、m_8代表了信号灯相位的触发状态，它们的值大于1，说明了相位的触发程度很高，即交通流获得通行权的程度很高，这是由信号灯的变换速率远大于交通流的速率造成的.虽然路口的交通次序与安全性得到了改善，却降低了交通流通过速率，延长车辆在路口的等待时间，从而说明了系统的运行效率低，系统中存在拥塞现象.

相关实验结果比较

为了充分说明我们所提出方法的优越性，我们使用生成可覆盖树的方法对离散系统的有界性进行判断，并将实验结果与方程模型方法的结果相比较.

依据文献[10]中给出的生成离散Petri网可覆盖树的算法，我们编写了相应的程序来生成离散Petri网的可覆盖树.以此交通模型为例，生成其离散Petri网的可覆盖树花费的时间为13.188s.而我们使用常微分方程模型的方法，只需要0.015s.

由此可以看出，使用微方分程模型来判断离散系统的有界性，是一种比较有效且开销小的方法.

7 数值解的近似度分析

一般来说，要想找到非线性常微分方程组的分析解是很难的，所以大多数情况下，我们只得到它的数值解.通过分析非线性常微分方程组的数值解的行为，我们可以检验系统的行为.

我们通常使用Matlab来解常微分方程组以得到方程组的数值解.由于我们的常微分方程组模型

是非刚性的,我们使用 ode45 这个函数来解我们的常微分方程组.

Ode45 通常是用来解如下方程组的:$\frac{d\boldsymbol{x}}{dt}=f(t,\boldsymbol{x}),\boldsymbol{x}(t_0)=x_0$.其中,$t$ 是一个独立的变量,表示时间、位置等,$\boldsymbol{x}$ 是一个向量,其中包含了一些与系统相关的变量.

Ode45 是 4 阶与 5 阶龙格-库塔方法的结合.对于 4 阶龙格-库塔方法,它的整体截断误差为 $O(h^4)$,五阶龙格-库塔方法的整体截断误差为 $O(h^5)$,其中 h 为步长.Ode45 通过自动调节步长,达到所给的精度,即整体截断误差为 $O(h^5)$.非线性常微分方程组的解可以无限接近它的初始可分析解,因此,我们可以通过分析它的数值解来替代分析它的初始可分析解,进而检测系统中是否存在死锁.

8　相关工作

本文涉及到两个方面的工作:系统的死锁检测及有界性判定.文献中已有很多这方面的研究结果,下面简述其中的一部分.

死锁检测

文献[11]中提出一种用有色 Petri 网来检测死锁的方法.由于使用有色 Petri 网为系统建模,所建模型大为简化,且模型的规模不会因系统中进程个数或资源种类数增多而增大.文中给出了构造有色 Petri 网可达标识图的算法,并基于可达标识图,给出了判断系统是否存在死锁的方法.文献[12]中基于 Petri 网为数据库建模,通过使用一些技术,解决了 Petri 网的可达性分析中可能出现的状态爆炸问题,并给出了判断系统中是否会出现死锁的充分必要条件.文献[13]中给出了网系统子类的定义及它的结构活性的充要条件,在此基础上考虑了使用请求/应答机制进行通信的网系统子类的正确性规范以及无死锁性条件.文献[14]给出了操作系统的一种基于 Petri 网的形式化模型,从而将死锁问题转化成线性代数问题,并给出了存在死锁的条件,依据条件对系统中的死锁进行检测.文献[15]中提出了 MPI 程序的同步通信模型及 3 个基本简化模型,基于这些模型对系统中的死锁进行检测.其中两个模型可以在程序编译前确定程序中是否存在死锁,而另一个模型需要在程序编译前及运行中分别对程序进行检测,以确定其中有没有死锁.文献[16]中利用模型检验与精化检验方法对系统的性质进行分析.文献[17]中对模型检测方法进行了阐述和分析.文献[18]中给出了离散系统存在死锁的条件,并给出了基于条件检测死锁的算法.

有界性判定

文献[19]给出了关于扩展强化非对称选择网(ESA 网)结构活和结构有界的一个判定算法,该算法可简单、有效地测试结构活和结构有界的 ESAC 网的初始标识是否是活标识.文献[20]针对国内外对时间 Petri 网 TPN 研究较少的情况,给出了时间 Petri 网保持活性、有界性的时间区间上的两个充分必要条件,从而为利用传统 Petri 网的性质判定结果来判定时间 Petri 网的相应性质提供了可能性.

9　总　结

本文通过大量的实验来说明文献[3]中所提出的检查程序中死锁方法的可行性.也就是说我们可以通过分析常微分方程组的解来检测系统中是否存在死锁.由于不涉及可达状态,我们的方法可以避开状态爆炸问题.同样的方法可以用来判断系统的有界性.通过与常用的可覆盖树方法比较,使用常微分方程的解来判断系统有界性的方法体现出了明显的优越性.

文中对方程组的求解借助于 Matlab.是对方程组进行串行计算.当方程的个数很大时,Matlab 并不能满足我们的需要.目前对微分方程的自动化并行计算已成为可能.我们先把 Petri 网转化为超图,然后利用 hMETIS 软件对超图进行划分,依据划分的结果对微分方程分组.据此我我们设计了求解微分方程组的并行算法及程序.算法中采用了隐式的线性多步方法和牛顿迭代.程序是在 SUNDIAL 上实现的.

在本文的方法中,用以描述并发程序的 Petri 网可以看作为自由选择(free-choice)Petri 网的一个子类.自由选择 Petri 网是目前用的比较普遍的一个 Petri 网,我们下一个阶段就是把连续化的方法延拓到自由选择 Petri 网上.

参　考　文　献

[1] Tu S, Shatz S M, Murata T. Applying Petri net reduction to support Ada tasking deadlock analysis//Proceedings of the 11th International Conference on Distributed Computing Systems. Paris, France, 1990: 96-103

[2] David R, Alla H. Continuous Petri nets//Proceedings of the 8th European Workshop on Application and Theory of Petri nets. Zaragoza, Spain, 1987: 275-294

[3] Ding Z. Static analysis of concurrent programs using ordinary differential equations//Lecture Notes in Computer Science 5684, Springer, 2009: 1-35

[4] Shatz S M, Mai K, Black C, Tu S. Design and implementation of a Petri net-based toolkit for Ada tasking analysis. IEEE Transactions on Parallel and Distributed Systems, 1990, 1(4): 424-441

[5] Dijkstra E W. Hierarchical ordering of sequential processes. Acta Informatica, 1971, 1(2): 115-138

[6] Duri S, Buy U, Devarapalli R, Shatz S M. Application and experimental evaluation of state space reduction methods for deadlock analysis in Ada. ACM Transactions on Software Engineering and Methodology, 1994, 3(4): 340-380

[7] Ding Z, Zhang K. Performance analysis of concurrent programs using ordinary differential equations//Proceedings of the 32nd Annual IEEE International Computer Software and Application Conference(COMPSAC). IEEE Computer Society, Washington, DC, USA, 2008: 841-846

[8] Febbraro A D, Sacco N. Hybrid Petri nets for the performance analysis of transportation systems//Proceedings of the 37th Conference. Tampa, USA, 2002, 3: 3232-3237

[9] Júlvez J, Boel R. Modeling and controlling traffic behaviors with continuous Petri nets//Proceedings of the 16th IFAC World Congress. Prague, Czech Republic, 2005

[10] Wu Zhe-Hui. Introduction to Petri Net. Beijing: China Machine Press, 2006(in Chinese)
(吴哲辉. Petri 网导论. 北京: 机械工业出版社, 2006)

[11] Han Yao-Jun, Jiang Chang-Jun. Colored Petri net-based deadlock detection and avoidance in concurrent operating system. Computer Science, 2002, 29(12): 190-192(in Chinese)
(韩耀军, 蒋昌俊. 并发操作系统中基于有色 Petri 网的死锁检测与避免. 计算机科学, 2002, 29(12): 190-192)

[12] Han Yao-Jun, Jiang Chang-Jun, Luo Xue-Mei. Deadlock detection of concurrency control in distributed database based on composition and reduction of Petri-net. Mini-Micro Systems, 2004, 25(5): 821-826(in Chinese)
(韩耀军, 蒋昌俊, 罗雪梅. 基于 Petri 网合成与化简的分布式数据库系统并发控制的死锁检测. 小型微型计算机系统, 2004, 25(5): 821-826)

[13] Huang Xiao-Wei, Lu Wei-Ming. A modular Petri net for the asynchronous communications. Journal of System Simulation, 2007, 19(Supplement 1): 132-137, 141(in Chinese)
(黄小炜, 陆维明. 用于实现异步通信的一种模块化 Petri 网. 系统仿真学报, 2007, 19(增刊 1): 132-137, 141)

[14] Wu Qi-Ming, Yuan Chong-Yi. A Petri nets model of deadlock. Journal of Software, 1991, 3(1): 5-10, 58(in Chinese)
(吴启明, 袁崇义. 死锁的 PETRI NETS 模型. 软件学报, 1991, 3(1): 5-10, 58)

[15] Liao Ming-Xue, Fan Zhi-Hua. Deadlock detection in basic models of MPI synchronization communication programs. Acta Electronica Sinica, 2008, 36(2): 402-407(in Chinese)
(廖名学, 范植华. MPI 程序同步通信基本模型死锁检测. 电子学报, 2008, 36(2): 402-407)

[16] Wen Yan-Jun, Wang Ji, Qi Zhi-Chang. Compositional model checking and compositional refinement checking of concurrent reactive system. Journal of Software, 2001, 18(6): 1270-1281(in Chinese)
(文艳军, 王戟, 齐治昌. 并发反应式系统的组合模型检验与组合精化检验. 软件学报, 2007, 18(6): 1270-1281)

[17] Lin Hui-Min, Zhang Wen-Hui. Model checking: Theories, techniques and applications. Acta Electronica Sinica, 2002, 30(12A): 1907-1912(in Chinese)
(林惠民, 张文辉. 模型检测:理论、方法与应用. 电子学报, 2002, 30(12A): 1907-1912)

[18] Wojcik B E, Wojcik Z M. Sufficient condition for a communication deadlock and distributed deadlock detection. IEEE Transactions on Software Engineering, 1989, 15(12): 1578-1595

[19] Jiao Li, Lu Wei-Ming. A polynomial algorithm to decide liveness and boundedness of ESAC nets. Journal of Software, 2002, 13(7): 1257-1263(in Chinese)
(焦莉, 陆维明. 关于 ESAC 网活性和有界性的一个多项式算法. 软件学报, 2002, 13(7): 1257-1263)

[20] Zhai Zheng-Li, Wu Zhe-Hui, Yang Yang. Two sufficient and necessary conditions of time Petri net preserving liveness and boundedness. Computer Science, 2006, 33(9): 232-234, 283(in Chinese)
(翟正利, 吴哲辉, 杨扬. 时间 Petri 网保持活性、有界性的两个充要条件. 计算机科学, 2006, 33(9): 232-234, 283)

DING Zuo-Hua, born in 1964, Ph. D., professor. His research interests include program analysis, software testing, service computing, etc.

JIANG Ming-Yue, born in 1985, M. S. candidate. Her research interests include program analysis and service computing.

LIU Jing, born in 1965, professor, Ph. D. supervisor. Her research interests include program analysis, software modeling, service computing, etc.

Background

State explosion problem has bothered us for decades when we do static analysis for concurrent programs. Many techniques have been proposed to combat this explosion, including state space reductions, symbolic model checking, compositional techniques, abstraction, dataflow analysis, integer programming techniques, shape analysis, and reachability-based analysis techniques. In general all existing approaches appear to be very sensitive to the size of the program being analyzed in terms of the use of concurrency constructs and the number of asynchronous processes.

The reason why we can not avoid state explosion, from our point of view, is that the concurrent systems are described as discrete systems. When we do analysis, we will enumerate all the reachable program states such as in reachability analysis techniques. An advantage of these techniques is that they are relatively simple from a conceptual viewpoint. However, we have to exhaustively explore all the reachable state space to detect concurrency errors. We know that state explosion problem is born with the discrete systems. Hence, if we want to thoroughly solve the state explosion problem, the discrete event systems should be continued to continuous systems, such that the systems can be described with analytic expressions. Therefore, instead of counting states, we can analyze the solutions of the analytic expressions.

This paper is supported by NSF for Key Research program of Dependable Software Theory under grant No. 90818013 and No. 90718014. The authors try to use a family of ordinary differential equations to describe concurrent programs, and check the system properties by analyzing the solutions of the equation group.