软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(8):2947−2963 [doi: 10.13328/j.cnki.jos.006607]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

模拟实时系统的点区间优先级时间 Petri 网与 TCTL 验证*

何雷锋, 刘关俊

(同济大学 计算机科学与技术系, 上海 201804)

通信作者: 刘关俊, E-mail: liuguanjun@tongji.edu.cn

摘 要: 时间 Petri 网为实时系统提供了一种形式化的建模方法, 时间计算树逻辑(TCTL)为描述实时系统与时间相关的设计需求提供了一种逻辑化的表达方式, 因此, 基于时间 Petri 网的 TCTL 模型检测广泛应用于实时系统的正确性验证. 然而对于一些涉及优先级的实时系统, 例如多核多任务实时系统, 这里不仅需要考虑任务之间的时间约束, 还要考虑任务执行的优先级以及引入优先级带来的抢占式调度问题, 致使相应的建模和分析变得更加困难. 为此, 提出了点区间优先级时间 Petri 网, 通过在时间 Petri 网上定义变迁发生的优先级以及变迁的可挂起性, 从而可以模拟实时系统的抢占式调度机制. 首先, 高优先级的任务抢占低优先级的任务所占用的资源, 导致后者被中断; 然后, 前者执行完毕后释放资源; 最后, 后者再次获得资源, 从中断的地方恢复. 通过点区间优先级时间 Petri 网来模拟多核多任务实时系统, 使用 TCTL 来描述它们的设计需求, 设计了相应的模型检测算法, 开发了相应的模型检测器以验证它们的正确性. 通过一个实例, 来说明该模型和方法的有效性.

关键词: 点区间优先级时间 Petri 网; 多核多任务实时系统; 时间计算树逻辑(TCTL); 模型检测; 抢占式调度
中图法分类号: TP311

中文引用格式: 何雷锋, 刘关俊. 模拟实时系统的点区间优先级时间 Petri 网与 TCTL 验证. 软件学报, 2022, 33(8): 2947–2963. http://www.jos.org.cn/1000-9825/6607.htm

英文引用格式: He LF, Liu GJ. Time-point-interval Prioritized Time Petri Nets Modelling Real-time Systems and TCTL Checking. Ruan Jian Xue Bao/Journal of Software, 2022, 33(8): 2947–2963 (in Chinese). http://www.jos.org.cn/1000-9825/6607.htm

Time-point-interval Prioritized Time Petri Nets Modelling Real-time Systems and TCTL Checking

HE Lei-Feng, LIU Guan-Jun

(Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)

Abstract: Time Petri nets is a formal method for modelling real-time systems, and timed computation tree logic (TCTL) is a logical expression for specifying time-related design requirements of real-time systems, so time Petri net based TCTL model checking has been widely used to verify the correction of real-time systems. For those real-time systems with priority such as multi-core multi-task real-time systems, it not only needs to consider time constraints among tasks but also needs to consider priority of task execution and the preemptive scheduling problem caused by priority, which results that modelling and analysis of these systems become more difficult. Therefore, this study proposes time-point-interval prioritized time Petri nets. By defining priority of transition firing and suspendable transitions in time Petri nets, time-point-interval prioritized time Petri nets can model preemptive scheduling of real-time systems, i.e., first of all, a high-priority task preempts the resource of a low-priority task, which results in the interruption of the latter, then the former is completed and releases the resource, and finally the latter gets the resource again and resumes from the interruption. This study uses time-point-interval prioritized time Petri nets to model multi-core multi-task real-time systems, uses TCTL to describe their design requirements, designs the corresponding model checking algorithms, and develops the corresponding model checker to verify their correctness. An example is used to show the effectiveness of the proposed model and method.

* 基金项目: 国家自然科学基金(62172299, 62032019); 上海市级科技重大专项(2021SHZDZX0100); 中央高校基本科研业务费专项资金

本文由“形式化方法与应用”专题特约编辑陈立前副教授、孙猛教授推荐.

收稿时间: 2021-09-05; 修改时间: 2021-10-14; 采用时间: 2022-01-10; jos 在线出版时间: 2022-01-28

Key words: time-point-interval prioritized time Petri nets; multi-core multi-task real-time systems; timed computation tree logic (TCTL); model checking; preemptive scheduling

对于一些安全性至关重要的系统, 例如航空航天系统, 微小的设计错误就可能会导致严重的后果. 因此, 保证系统的正确性是完全必要的. 然而, 随着系统规模的日益扩大, 可能出现的错误也呈指数级增长, 导致传统的检测手段无法胜任如此繁重的工作. 因此, 对系统设计者来说, 寻找一种逻辑上精确验证系统正确性的方法是很有必要的, 例如各种形式化方法[1−3]. 模型检测[4,5]作为一种完全自动化的形式化方法得到了深入的研究, 已成功应用于一些实时系统的正确性验证[6−8]. 模型检测采用一种形式化语言来模拟各种软硬件系统, 例如 Petri 网[9,10]、反应式模块[11]、解释系统编程语言(ISPL)[12]等, 使用一种逻辑公式来描述这些系统的设计需求, 例如线性时态逻辑(LTL)[13,14]、计算树逻辑(CTL)[15]、交替时态逻辑(ATL)[16]等, 从而把验证一个系统是否满足它的设计需求的问题等价为验证相应的模型是否满足相应的逻辑公式的问题, 而验证模型是否满足逻辑公式则可以完全通过各种模型检测器自动化完成.

Petri 网作为一种形式化语言, 广泛应用于建模和分析各种软硬件系统, 例如柔性制造系统[17]、业务过程管理系统[18]、并发程序形式化验证[19]等. 时间 Petri 网[20,21]是一种包含时间因素的高级 Petri 网, 适用于实时系统的建模和分析[22,23]. 对于一个时间 Petri 网, 每个变迁伴随着一个静态发生区间, 它表示该变迁从获得发生权到发生所需要的最短等待时间和最长等待时间, 而变迁本身发生不需要时间. 计算树逻辑(CTL)[15]作为模型检测技术最常用的逻辑公式之一, 用来描述系统常见的一些设计需求, 例如无死锁、安全性、活性和公平性等. 时间计算树逻辑(TCTL)[24,25]则是在 CTL 的基础之上添加了具体的时间约束, 通过量化 CTL 的时态算子来描述系统与时间相关的一些设计需求, 例如一个任务一旦启动就必须在某个限定的时间内执行完毕. 基于时间 Petri 网的 TCTL 模型检测已成功应用于一些实时系统的正确性验证[26−28].

对于时间 Petri 网, 如果在一个状态下多个变迁均具有发生权, 且彼此之间不存在时间冲突(在一个状态下两个具有发生权的变迁存在时间冲突, 即一个变迁发生所需要的最短等待时间大于另一个变迁发生所需要的最长等待时间), 则按变迁发生规则, 可随机发生其中一个变迁. 然而这种情况并不适用于一些涉及优先级的实时系统, 因为这里不仅需要考虑任务之间的时间约束, 还要考虑任务执行的优先级. 对于一些更为紧要的任务, 系统通常会给它们更高的优先级, 使得它们在与低优先级的任务竞争资源时优先获得资源执行. 因此, Berthomieu[29,30]等人提出了优先级时间 Petri 网, 它是在时间 Petri 网的基础之上添加了变迁发生的优先级, 即一个变迁在一个状态下是可发生的当且仅当它满足以下 3 个条件: (1) 在当前状态的标识下, 该变迁具有发生权; (2) 在当前状态下, 该变迁的已等待时间在它的静态发生区间内; (3) 在当前状态下, 所有满足条件(1)和条件(2)变迁的优先级不会高于该变迁. 优先级时间 Petri 网显然增强了时间 Petri 网的建模能力, 使得时间 Petri 网在实时系统中得到了更广泛的应用, 例如一些涉及线程调度、仲裁和同步的实时系统, 均可以通过优先级时间 Petri 网来建模和分析.

Berthomieu 等人提出的优先级时间 Petri 网虽然可以模拟一些涉及优先级的实时系统, 但是并不能满足所有涉及优先级的实时系统的建模需求, 我们可以通过一个简单的例子来说明这种优先级时间 Petri 网的局限性. 通常, 用优先级时间 Petri 网来模拟一个实时系统时, 一些变迁是代表任务的, 称为任务型变迁. 一个任务型变迁 t_i 代表一个任务 $Task_i$, 伴随 t_i 的静态时间区间代表任务 $Task_i$ 执行完毕所需要的最短和最长时间,变迁 t_i 已等待的时间代表任务 $Task_i$ 已执行的时间, 变迁 t_i 发生代表任务 $Task_i$ 执行完毕. 在一个状态下, 结构上是并发关系的任务型变迁 t_1 和 t_2 同时获得发生权且彼此之间无时间冲突, 经过 τ 个时间后 t_1 发生, 则 t_1 发生后 t_2 的已等待时间为 τ, 即任务 $Task_2$ 已执行时间为 τ. 这时, t_1 发生后产生的新状态导致变迁 t_3 可发生, 它的优先级高于 t_2 且彼此之间无时间冲突, 则 t_3 会先于 t_2 发生, 假设 t_2 和 t_3 在结构上是冲突关系, 则 t_3 的发生会导致 t_2 失去发生权. 即使 t_2 后来重新获得发生权, 但它的已等待时间已从 0 开始计时. 这意味着任务 $Task_2$ 在占有资源且已执行一段时间的情况下, 被另一个更高优先级的任务 $Task_3$ 抢占资源而中断, 而后再次获得资源却不会从中断的地方恢复而是从头开始执行, 这显然不符合一些实时系统的抢占式调度机制. 例如多核多任务实时

系统, 即高优先级的任务可抢占低优先级的任务所占用的资源导致后者被中断, 前者执行完毕后释放资源, 后者获得资源从中断的地方恢复, 继续执行剩余的一段时间而不是从头开始执行. 这意味着 Berthomieu 等人提出的优先级时间 Petri 网不能模拟支持抢占式调度机制的实时系统.

针对传统的优先级时间 Petri 网对实时系统建模能力的不足, 我们提出了另一种类型的优先级时间 Petri 网, 它不仅考虑了任务执行的优先级, 还考虑了任务之间的抢占式调度需求. 由于一些多核多任务实时系统的任务执行时间不是一个不确定的区间而是一个确定的值, 因此我们考虑时间 Petri 网这样的一类子网, 即伴随每个变迁的静态发生区间为点区间, 我们称为点区间时间 Petri 网. 因此, 本文定义的这种优先级时间 Petri 网被称为点区间优先级时间 Petri 网. 这意味着每个变迁从获得发生权到发生所需要的最短和最长等待时间是相同的, 这种变迁从获得发生权到发生的等待时间完全确定的时间 Petri 网, 减少了所生成的状态图中的状态数, 降低了 TCTL 模型检测算法的时间和空间复杂度, 提高了对实时系统的验证效率. 除了变迁的静态发生区间不同外, 我们定义的点区间优先级时间 Petri 网与 Berthomieu 等人提出的优先级时间 Petri 网从网的结构、状态的定义到变迁的发生规则等方面也是不同的, 本文将在第 2 节详细介绍它们的区别. 我们通过点区间优先级时间 Petri 网来模拟多核多任务实时系统, 使用 TCTL 来描述它们的设计需求, 设计了相应的 TCTL 模型检测算法, 并开发了相应的模型检测器, 实现了对多核多任务实时系统的正确性验证.

本文第 1 节介绍一些基本知识, 包括 Petri 网、时间 Petri 网以及相关概念. 第 2 节介绍我们提出的点区间优先级时间 Petri 网及它的状态图. 第 3 节介绍 TCTL 的语法和语义. 第 4 节介绍我们的模型检测算法和模型检测器. 第 5 节通过一个实例来说明模型和方法的有效性. 第 6 节回顾一些相关的工作. 第 7 节对全文进行总结并阐述下一步的工作.

1 预备知识

本节简单介绍 Petri 网、时间 Petri 网及其相关概念, 有关 Petri 网的详情见文献[31], 有关时间 Petri 网的详情见文献[20,21]. 假设 $\mathbb{N}$是自然数集, R 是实数集, R^+是非负实数集. 定义实数闭区间 $I=\{x\in R|a\leqslant x\leqslant b\}$, 记为 $I=[a,b]$. 这里, $a,b\in R$ 且 $a\leqslant b$. 用 IR 和 IR^+分别表示所有实数闭区间和非负实数闭区间的集合. 若 $a=b$, 则 $I=[a,a]$称为点区间. 设 $I\in IR$ 且 $I=[a,b]$, 定义$\downarrow I=a$, $\uparrow I=b$.

1.1 Petri网

一个网定义为三元组 $N=(P,T,F)$, 其中, P 为库所集, T 为变迁集, $P\cap T=\varnothing$, $F\subset(P\times T)\cup(T\times P)$为 P 与 T 之间的流关系. 从形式上看, 一个网就是一个没有孤立结点的有向二分图, 一般用小圆圈表示库所, 小矩形表示变迁. 给定一个节点 $x\in P\cup T$, 节点 x 的前集定义为$^\bullet x=\{y\in P\cup T|F(y,x)=1\}$, 后集定义为 $x^\bullet=\{y\in P\cup T|F(x,y)=1\}$. 网的标识是一个映射函数 $M:P\to\mathbb{N}$, $M(p)$表示库所 p 里的 token 数. 一个标识通常表示为库所上的一个多重集, 例如在标识 M 下, p_1 里有 3 个 token, p_3 里有 1 个 token, 而其他库所均无 token, 则 M 表示为 $\{3p_1,p_3\}$ 或者 $3p_1+p_3$.

一个带有初始标识 M_0 的网 N 被称作为一个 Petri 网, 记作$\Sigma=(N,M_0)$. 如果对$\forall p\in P$, 满足 $F(p,t)\leqslant M(p)$, 则称变迁 t 在标识 M 下具有发生权. t 的发生, 使系统进入一个新的标识 M', 记作 $M[t\rangle M'$, 即: 对$\forall p\in P:M'(p)=M(p)\dot{-}F(p,t)+F(t,p)$. 标识 M_k 从 M 出发是可达的当且仅当 $M_k=M$, 或者存在一个可发生序列$\sigma=t_1t_2\ldots t_{n-1}t_k$, 使得 $M_0[t_1\rangle M_1[t_2\rangle\ldots M_{k-1}[t_k\rangle M_k$ 成立. 上式也可以表示为 $M_0[\sigma\rangle M_k$. 从 M 出发的所有可达标识记作 $R(N,M)$, 则 Petri 网的所有可达标识即为 $R(N,M_0)$. 如果在一个标识下没有变迁具有发生权, 则称该标识为 Petri 网的一个死锁(deadlock). 如果对$\forall M\in R(N,M_0)$和$\forall p\in P$, 满足 $M(p)\leqslant 1$, 则称此 Petri 网是安全的. 本文只考虑安全的 Petri 网. 为了保证安全性, 首先要求在初始标识 M_0 里的每一个库所最多含有 1 个 token; 其次, Petri 网的变迁发生规则稍作修改, 即, 如果 $M[t\rangle M'$, 则 M'满足对$\forall p\in P:M'(p)=\min\{M(p)-F(p,t)+F(t,p),1\}$. 也就是说, 如果一个库所在一个标识下含有一个 token, 此后变迁发生又有新 token 进入该库所, 但它仍然保持为一个 token 而不是两个 token. 本文后面所提到的 Petri 网均为这种类型的 Petri 网, 因此下文所提到的每一个标识, 实际上表示为库所

上的一个集合而不是多重集.

1.2 时间Petri网

一个时间 Petri 网可以定义为五元组(P,T,F,M_0,SI), 其中, (P,T,F,M_0)为一个 Petri 网, $SI:T\to IR^+$为一个映射函数. $SI(t)$表示变迁 t 的静态发生区间, $\downarrow SI(t)$为 t 获得发生权到发生所需要的最短等待时间, $\uparrow SI(t)$为 t 获得发生权到发生所需要的最长等待时间. $En(M)$表示在标识 M 下所有具有发生权的变迁, (M,h)表示时间 Petri 网的一个状态, 其中, $h:En(M)\to R^+$是一个时钟函数. $h(t)$表示在标识 M 下具有发生权的变迁 t 的已等待时间. 变迁 t 在状态(M,h)下是可发生的当且仅当它满足以下两个条件: (1) $t\in En(M)$; (2) $h(t)\in SI(t)$. 也就是说, 一个变迁在一个状态下是可发生的当且仅当该变迁在当前状态的标识下具有发生权, 且该变迁的已等待时间在它的静态发生区间内.

时间 Petri 网的初始状态为 $S_0=(M_0,h_0)$, 其中, M_0 为初始标识, 对$\forall t\in En(M_0):h_0(t)=0$. 假设 $S=(M,h)$和 $S'=(M',h')$为时间 Petri 网的两个状态, 则时间 Petri 网中存在着两种状态迁移规则.

1. $S\xrightarrow{\tau}S'$ 当且仅当状态 S'从状态 S 经过τ个时间后可达, 即:
   (1) $M=M'$;
   (2) $\forall t\in En(M):h'(t)=h(t)+\tau\leqslant\uparrow SI(t)$;
2. $S\xrightarrow{\tau}S'$ 当且仅当状态 S'从状态 S 经过变迁 t 发生后可达, 即:
   (1) $t\in En(M)$;
   (2) $h(t)\in SI(t)$;
   (3) $M[t\rangle M'$;
   (4) $\forall t'\in En(M')$: 如果 $t'\in En(M)$且 $t'\neq t$, 则 $h'(t')=h(t')$; 否则, $h'(t')=0$.

第 1 种状态迁移为时间流逝导致的, 即标识不变但每个具有发生权的变迁的已等待时间增加. 第 2 种状态迁移为变迁发生导致的, 即新标识产生, 如果在变迁发生前后, 一些变迁(非发生变迁)一直具有发生权, 则在变迁发生后, 这些变迁的已等待时间保持不变; 如果之前没有发生权的变迁在变迁发生后获得发生权或者发生变迁本身在变迁发生后再次具有发生权, 则在变迁发生后, 这些变迁的已等待时间为 0.

如果一个时间 Petri 网(P,T,F,M_0,SI)满足对$\forall t\in T:\downarrow SI(t)=\uparrow SI(t)$, 则称它为点区间时间 Petri 网. 对于点区间时间 Petri 网, 每个变迁的静态发生区间为点区间, 即, 每个变迁从获得发生权到发生所需要的等待时间是确定的.

2 点区间优先级时间 Petri 网

2.1 点区间优先级时间Petri网

定义 1(**点区间优先级时间 Petri 网**). 一个点区间优先级时间 Petri 网(time-point-interval prioritized time Petri nets)可以定义为七元组 $Z=(P,T_1,T_2,F,M_0,SI,Pr)$, 其中, $(P,T_1\cup T_2,F,M_0,SI)$为一个点区间时间 Petri 网, T_1 为不可挂起变迁集, T_2 为可挂起变迁集; $Pr\subset(T_1\cup T_2)\times(T_1\cup T_2)$为变迁之间的优先级关系.

$(t,t')\in Pr$ 表示变迁 t 发生的优先级高于变迁 t'. Pr 满足非自反性、非对称性和传递性. $En(M)$表示在标识 M 下所有具有发生权的变迁, $Pend(M)$表示在标识 M 下所有已挂起的可挂起变迁, (M,h,H)表示 Z 的一个状态, 其中, M 是一个标识; $h:En(M)\to R^+$是一个时钟函数, $h(t)$表示在标识 M 下具有发生权的变迁 t 的已等待时间; $H:Pend(M)\to R^+$是一个时钟函数, $H(t)$表示在标识 M 下已挂起变迁 t 的中断时间. 变迁 t 在 Z 的一个状态(M,h,H)下是可发生的当且仅当它满足以下 3 个条件.

(1) $t\in En(M)$;
(2) $h(t)\in SI(t)$;
(3) 对$\forall t'\in T_1\cup T_2$: 如果 $t'\in En(M)$且 $h(t')\in SI(t')$, 则$(t',t)\notin Pr$.

也就是说, 一个变迁在一个状态下是可发生的当且仅当该变迁在当前状态的标识下具有发生权, 且它的

已等待时间在它的静态发生区间内, 同时不存在满足同样条件的其他变迁优先级高于该变迁.

点区间优先级时间 Petri 网 Z 的初始状态为 $S_0=(M_0,h_0,H_0)$, 其中, M_0 为初始标识, 对$\forall t\in En(M_0):h_0(t)=0$. 由于 $Pend(M_0)=\varnothing$, 故 $H_0=\varnothing$. 假设 $S=(M,h,H)$和 $S'=(M',h',H')$为 Z 的两个状态, 则 Z 中存在着两种状态迁移规则.

1. $S\xrightarrow{\tau}S'$ 当且仅当状态 S'从状态 S 经过τ个时间后可达, 即:
   (1) $M=M'$;
   (2) $\forall t\in En(M):h'(t)=h(t)+\tau\leqslant\uparrow SI(t)$;
   (3) $\forall t\in Pend(M):H'(t)=H(t)$;
2. $S\xrightarrow{t}S'$ 当且仅当状态 S'从状态 S 经过变迁 t 发生后可达, 即:
   (1) $t\in En(M)$;
   (2) $h(t)\in SI(t)$;
   (3) 对$\forall t'\in T_1\cup T_2$: 如果 $t'\in En(M)$且 $h(t')\in SI(t')$, 则$(t',t)\notin Pr$;
   (4) $M[t\rangle M'$;
   (5) $\forall t'\in En(M')$: 如果 $t'\in En(M)$且 $t'\neq t$, 则 $h'(t')=h(t')$; 否则, $h'(t')=0$;
   (6) $\forall t'\in Pend(M)$: 如果 $t'\in En(M')$, 则 $t'\notin Pend(M')$且 $h'(t')=H(t')$; 否则, $H'(t')=H(t')$;
   (7) $\forall t'\in En(M)\cap T_2$ 且 $t'\neq t$: 如果 $t'\notin En(M')$, 则 $t'\in Pend(M)$且 $H'(t')=h(t')$.

第1种状态迁移为时间流逝导致的, 即标识不变, 每个已挂起变迁的中断时间不变, 但每个具有发生权的变迁的已等待时间增加. 第 2 种状态迁移为可发生变迁发生导致的, 即新标识产生, 如果在变迁发生前后,一些变迁(非发生变迁)一直具有发生权, 则在变迁发生后, 这些变迁的已等待时间保持不变; 如果之前没有发生权的变迁在变迁发生后获得发生权或者发生变迁本身再次具有发生权, 则在变迁发生后, 这些变迁的已等待时间为 0. 如果可挂起变迁在变迁发生后失去发生权(非发生变迁), 则它在变迁发生后被挂起且它的中断时间记为它在变迁发生前的已等待时间. 如果已挂起变迁在变迁发生后获得发起权, 则它在变迁发生后恢复为具有发生权的变迁, 且它的已等待时间恢复为它挂起时的中断时间; 否则, 已挂起变迁在变迁发生后, 保持它之前的中断时间不变.

注意, 我们定义的点区间优先级时间 Petri 网与 Berthomieu 等人定义的优先级时间 Petri 网是完全不同的, 主要区别在于以下 4 点.

(1) 在结构上, 我们把变迁集分为可挂起变迁集和不可挂起变迁集两种类型; 而 Berth-omieu 等人定义的变迁集没有分类;

(2) 在变迁的静态发生区间上, 我们定义的 Petri 网是基于点区间时间 Petri 网而扩展的; 而 Berthomieu 等人定义的 Petri 网是基于一般的时间 Petri 网而扩展的;

(3) 在状态的定义上, 我们定义的状态包含 3 个参数, 即标识、该标识下每个具有发生权的变迁的已等待时间以及该标识下每个已挂起变迁的中断时间; 而 Berthomieu 等人定义的状态只包含两个参数, 即标识和该标识下每个具有发生权的变迁的已等待时间;

(4) 在变迁发生规则上, 我们定义: 如果一个可挂起变迁由于别的变迁发生而失去发生权, 则它会被挂起, 它的中断时间会被保存; 反之, 如果一个已挂起变迁由于别的变迁发生而再次具有发生权, 则它会被恢复为具有发生权的变迁, 且它的已等待时间恢复为它挂起时的中断时间. 而 Berthomieu 等人的定义与时间 Petri 的定义一样, 即: 如果一个变迁由于别的变迁发生而失去发生权, 则它的已等待时间会被清空, 而该变迁以后再次具有发生权时, 它的已等待时间会从 0 开始计时.

我们通过一个简单的例子来解释本文的相关概念. 图 1 为一个点区间优先级时间 Petri 网, 其中, $(t_2,t_7)\in Pr$, 可挂起变迁集 $T_2=\{t_8\}$, 其余变迁构成不可挂起变迁集 T_1. 它描述了一个单核双任务实时系统. 该系统有两个任务 A 和 B, 它们共用一个处理器 c_1. 任务 A 在 15ms 后启动(t_1), 任务 B 在 10ms 后启动(t_6). 每个任务均需要先获得处理器 c_1, 然后才能执行(t_2,t_7). 任务 A 需要 5ms 执行完毕(t_3), 任务 B 需要 10ms 执行完毕(t_8). 由于任务 A 的优先级高于任务 B, 所以任务 A 可抢占任务 B 所占用的处理器 c_1(t_4), 任务 A 执行完毕后释放处理

器 $c_1(t_5)$, 之后, 任务 B 获得处理器 c_1 从中断的地方恢复, 继续执行剩余的一段时间(t_8). 因此, t_8 是唯一的可挂起变迁, 即 $T_2=\{t_8\}$. $(t_2,t_7)\in Pr$ 意味着任务 A, B 同时竞争处理器 c_1 时, 任务 A 的高优先级使得任务 A 优先获得处理器 c_1 执行.

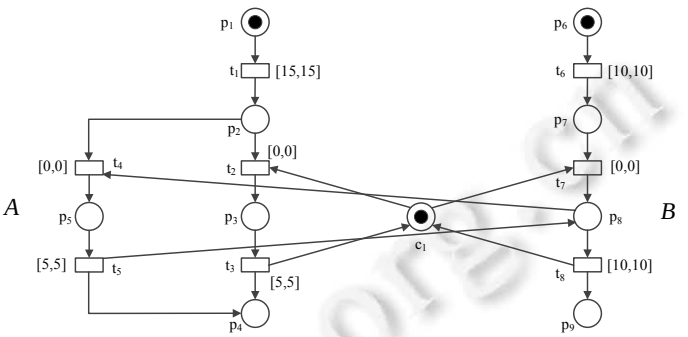


图 1 描述一个单核双任务实时系统的点区间优先级时间 Petri 网

2.2 状态图

对于时间 Petri 网来说, 一个变迁从获得发生权到发生一般要经历两个阶段: 一是从获得发生权到可发生, 即它从获得发生权一直等待到它的已等待时间在它的静态发生区间内; 二是该变迁从可发生到发生. 因此, 时间 Petri 网的两种状态迁移规则完全可以合并到一起, 即 $S\xrightarrow{\tau}S'\xrightarrow{t}S''$ 等价为 $S\xrightarrow{(\tau,t)}S''$, 它表示状态 S 在等待 τ 个时间后发生变迁 t 到达状态 S''. 我们称变迁 t 在状态 S 下是可调度的(schedulable). 对于时间 Petri 网的状态图, 每一步的状态迁移均为时间流逝+变迁发生导致的. 同理, 我们定义的点区间优先级时间 Petri 网的两种状态迁移规则也可以合并到一起. 基于此, 我们给出了点区间优先级时间 Petri 网的状态图的定义.

定义 2(状态图). 点区间优先级时间 Petri 网 Z 的状态图可以定义为五元组 $\Delta=(\Omega,S_0,\rightarrow,L,time)$, 其中, Ω 是 Z 中所有的可达状态; $S_0=(M_0,h_0,H_0)$ 是 Z 的初始状态; $\rightarrow\subset\Omega\times\Omega$ 表示状态之间的迁移关系; L 为迁移关系上的一个标签函数: $(S,S')\rightarrow T_1\cup T_2$, $L(S,S')$ 表示从状态 S 迁移到状态 S' 所发生的变迁; $time$ 为迁移关系上的一个时钟函数: $(S,S')\rightarrow R^+$, $time(S,S')$ 表示从状态 S 迁移到状态 S' 所等待的时间. 即: 如果 $\exists\tau\in R^+$ 和 $\exists t\in T_1\cup T_2$ 使得 $S\xrightarrow{(\tau,t)}S'$, 则称 $S\rightarrow S'$, $L(S,S')=t$, $time(S,S')=\tau$.

注意, 状态图中并非包含了所有的可达状态. 在状态图中, 除了初始状态 S_0, 其他所有的状态均是变迁发生后产生的新状态. 对于时间流逝导致的状态迁移, 它所产生的新状态并不会出现在状态图中. 例如在图 1 中时间流逝导致的一个状态迁移 $(p_1+p_6+c_1,h(t_1)=h(t_6)=0)\xrightarrow{5}(p_1+p_6+c_1,h(t_1)=h(t_6)=5)$, 它所产生的新状态 $(p_1+p_6+c_1,h(t_1)=h(t_6)=5)$ 并不会出现在状态图中. 然而对于状态图中未出现的状态, 我们可以很容易地由状态图中的已知状态通过时间流逝得到.

对于点区间优先级时间 Petri 网, 由于它是安全 Petri 网且每个变迁发生前的等待时间是确定的, 因此它的状态图中的状态数必然是有限的. 图 2 为图 1 点区间优先级时间 Petri 网的状态图, 图中共有 7 个状态, 除了初始状态 S_0 外, 其他状态均是变迁发生后产生的新状态.

- 首先, 在状态 S_0 下, 只有变迁 t_1 和 t_6 具有发生权且它们的已等待时间为 0. t_1 发生所需要的时间为 15 而 t_6 发生所需要的时间为 10, 因此只有 t_6 可调度. 在等待 10 个时间单元后, t_6 发生到达状态 S_1;
- 在状态 S_1 下, t_1 仍然具有发生权, 因此它的已等待时间为 10. t_7 刚获得发生权, 因此它的已等待时间为 0. t_1 发生所需要的时间为 5 而 t_7 发生所需要的时间为 0, 因此 t_7 发生到达状态 S_2;
- 在状态 S_2 下, t_1 仍然具有发生权, 因此它的已等待时间为 10. t_8 刚获得发生权, 因此它的已等待时间为 0. t_1 发生所需要的时间为 5 而 t_8 发生所需要的时间为 10, 因此只有 t_1 可调度. 在等待 5 个时间单元后, t_1 发生到达状态 S_3;
- 在状态 S_3 下, t_8 仍然具有发生权, 因此它的已等待时间为 5. t_4 刚获得发生权, 因此它的已等待时间为 0. 由于 t_8 发生所需要的时间为 5 而 t_4 发生所需要的时间为 0, 因此 t_4 发生到达状态 S_4;

- 在状态 S_4 下, t_5 刚获得发生权, 因此它的已等待时间为 0. 由于可挂起变迁 t_8 失去发生权, 因此 t_8 被挂起, 且挂起时的中断时间为挂起前的已等待时间, 即 $H(t_8)$=5. t_5 发生所需要的时间为 5, 因此在等待 5 个时间后, t_5 发生到达状态 S_5;
- 在状态 S_5 下, 已挂起变迁 t_8 重新获得发生权, 因此由已挂起变迁恢复为具有发生权的变迁, 且它的已等待时间恢复为挂起时的中断时间, 即 $h(t_8)$=5. t_8 发生所需要的时间为 5, 因此在等待 5 个时间单元后, t_8 发生到达终点状态 S_6.

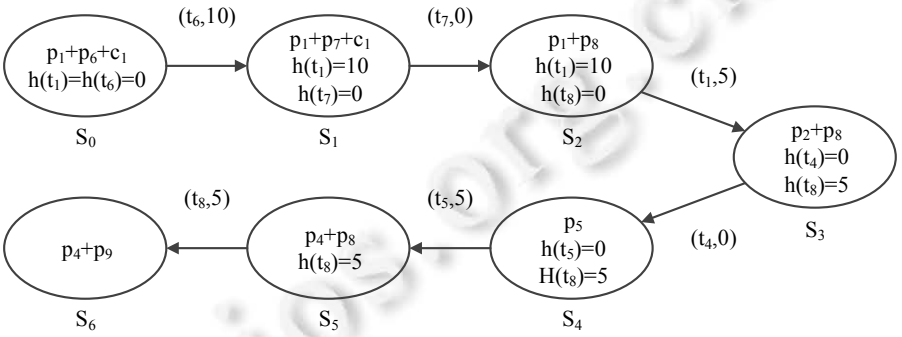


图 2 图 1 中点区间优先级时间 Petri 网的状态图

注意: 虽然图 1 的点区间优先级时间 Petri 网定义了优先级关系(t_2,t_7), 但由于任务 A 和 B 的启动时间不一致和任务的非周期性执行等原因, 在生成状态图的过程中并没有出现 t_2 和 t_7 同时可发生的情况. 然而对于后面要提到的多核多任务实时系统, 模拟它的点区间优先级时间 Petri 网中的优先级关系, 就会起到两个变迁同时可发生时, 让高优先级的变迁先发生的作用.

3 TCTL

本文使用时间计算树逻辑(TCTL)[24,25]来描述实时系统的设计需求. 首先, TCTL 的语法基于点区间优先级时间 Petri 网来定义; 然后, TCTL 的语义通过它的状态图来解释.

定义 3(TCTL 的语法). 基于一个点区间优先级时间 Petri 网 $Z=(P,T_1,T_2,F,M_0,SI,Pr)$, TCTL 定义如下:

$$\phi::=\mathbf{true}|p|\neg\phi|\phi_1\wedge\phi_2|EX\phi|AX\phi|EG\phi|AG\phi|E(\phi_1U_{\bowtie c}\phi_2)|A(\phi_1U_{\bowtie c}\phi_2),$$

其中, $p\in P$, $\bowtie\in\{<,\leqslant,>,\geqslant\}$, $c\in R^+$.

TCTL 中的其他算子可以由以上的算子推导出: $deadlock=\neg EX\mathbf{true}$, $\phi_1\vee\phi_2=\neg(\neg\phi_1\wedge\neg\phi_2)$, $\phi_1\rightarrow\phi_2=\neg\phi_1\vee\phi_2$, $EF_{\bowtie c}\phi=E(\mathbf{true}U_{\bowtie c}\phi)$, $AF_{\bowtie c}\phi=A(\mathbf{true}U_{\bowtie c}\phi)$.

给定一个状态图Δ和它的一个状态 S, 我们定义在Δ中从 S 出发的一个计算(computation)是一个最大状态序列, 即$\omega=(S^0,S^1,\ldots)$, 其中, $S^0=S$. 一个计算可能是无限的也可能是有限的(遇到死锁): 对于一个有限的计算$\omega=(S^0,S^1,\ldots,S^n,\ldots)$, 假设 S^n 是一个死锁, 则对$\forall i\in\{0,1,\ldots,n\}:(S^i,S^{i+1})\in\rightarrow$且$\omega(i)=S^i$, 当 $i>n$, 则$\omega(i)=\varnothing$; 对于一个无限的计算$\omega=(S^0,S^1,\ldots)$, 对$\forall i\in\mathbb{N}$: $(S^i,S^{i+1})\in\rightarrow$且$\omega(i)=S^i$. $\mho(S)$表示在Δ中所有从 S 出发的计算.

定义 4(TCTL 的语义). 给定一个点区间优先级时间 Petri 网的状态图$\Delta=(\Omega,S_0,\rightarrow,L,time)$, 一个状态 $S\in\Omega$和一个 TCTL 公式ϕ, $(\Delta,S)\models\phi$意味着公式ϕ在状态图Δ的状态 S 下为真. 换句话说, 状态图Δ中的状态 S 满足公式ϕ. 如果没有歧义的话, Δ可省略, 满足关系$\models$归纳定义如下.

- $S\models\mathbf{true}$;
- $S\models p$ 当且仅当 S 的标识 M 满足 $M(p)$=1;
- $S\models\neg\phi$当且仅当 $S\models\phi$不成立;
- $S\models\phi_1\wedge\phi_2$ 当且仅当 $S\models\phi_1$ 且 $S|=\phi_2$;

- $S\vDash EX\phi$当且仅当$\exists S'\in\Omega$, 满足 $S\to S'$且 $S'\vDash\phi$;
- $S\vDash AX\phi$当且仅当对$\forall S'\in\Omega$: 如果 $S\to S'$, 则 $S'\vDash\phi$;
- $S\vDash EG\phi$当且仅当$\exists\omega\in\mathcal{O}(S)$: 对$\forall i\in\mathbb{N}$:$\omega(i)\vDash\phi$;
- $S\vDash AG\phi$当且仅当对$\forall\omega\in\mathcal{O}(S)$: 对$\forall i\in\mathbb{N}$:$\omega(i)\vDash\phi$;
- $S\vDash E(\phi_1U_{\bowtie c}\phi_2)$当且仅当$\exists\omega\in\mathcal{O}(S)$满足这样的条件: (1) $\exists i\in\mathbb{N}$, 满足$\omega(i)\vDash\phi_2$; (2) 对$\forall j\in\{0,1,\ldots,i-1\}$, 满足$\omega(j)\vDash\phi_1$; (3) $time(\omega(0),\omega(1))+time(\omega(1),\omega(2))+\ldots+time(\omega(i-1),\omega(i))\bowtie c$;
- $S\vDash A(\phi_1U_{\bowtie c}\phi_2)$当且仅当$\forall\omega\in\mathcal{O}(S)$满足这样的条件: (1) $\exists i\in\mathbb{N}$, 满足$\omega(i)\vDash\phi_2$; (2) 对$\forall j\in\{0,1,\ldots,i-1\}$, 满足$\omega(j)|=\phi_1$; (3) $time(\omega(0),\omega(1))+time(\omega(1),\omega(2))+\ldots+time(\omega(i-1),\omega(i))\bowtie c$.

定义 5(有效性). 一个 TCTL 公式ϕ在点区间优先级时间 Petri 网 Z 上是有效的(记作 $Z\vDash\phi$)当且仅当 Z 的状态图Δ满足$(\Delta,S_0)\vDash\phi$, 即状态图Δ中的初始状态 S_0 满足ϕ.

TCTL 是在 CTL 的基础之上添加了具体的时间约束, 通过量化 CTL 的时态算子来描述实时系统与时间相关的一些设计需求. 例如: 对于图 1 中的实时系统, 我们可要求任务 B 在执行过程中不会被中断, 即任务 B 一旦启动就必须在 10ms 内执行完毕, 这样的设计需求可通过 TCTL 公式$\phi_1=AG(p_7\to AF_{\leqslant 10}p_9)$来描述, 其中, p_7 意味着任务 B 启动, p_9 意味着任务 B 执行完毕. 后面我们开发的模型检测器可验证ϕ_1 在图 1 上是无效的.

4 算法和工具

给定一个点区间优先级时间 Petri 网 Z 和一个 TCTL 公式ϕ, 验证 $Z\vDash\phi$是否成立主要包含以下 3 个步骤.

(1) 由点区间优先级时间 Petri 网 Z 生成它的状态图Δ;

(2) 在状态图Δ上递归寻找所有满足ϕ的状态(记作 $Sat(\phi)$);

(3) $Z\vDash\phi$当且仅当 $S_0\in Sat(\phi)$.

由点区间优先级时间 Petri 网生成其状态图的伪代码见算法 1.

算法 1.

Input: A time-point-interval prioritized time Petri nets $Z=(P,T_1,T_2,F,M_0,SI,Pr)$;

Output: The state graph Δ of Z.

$\Omega:=From:=S_0:=(M_0,h_0,H_0)$;

repeat

  $To:=\varnothing$;

  for each $S\in From$

    for each $t\in T_1\cup T_2$

      if t is schedulable at S **then**

        $\exists\tau\in R^+$ such that $S\xrightarrow{(\tau,t)}S'$; $To:=To+S'$; Add (S,S') to $\to$; $L(S,S'):=t$; $time(S,S'):=\tau$;

      endif

    endfor

  endfor

  $New:=To-\Omega$; $From:=New$; $\Omega:=\Omega+New$;

until $New=\varnothing$;

return $(\Omega,S_0,\to,L,time)$.

基于生成的状态图Δ可计算 $Sat(\phi)$. 根据 TCTL 的语义, 计算 $Sat(\phi)$的伪代码见算法 2.

算法 2.

Input: A state graph $\Delta=(\Omega,S_0,\to,L,time)$ and a TCTL formula ϕ;

Output: $\{S\in\Omega|S\vDash\phi\}$.

if ϕ is **true then return** Ω; **endif**

if ϕ is p **then return** $\{S\in\Omega|M(p)=1\}$; **endif**

if ϕ is $\neg\phi$ **then return** $\bar{\Omega}Sat(\phi)$; **endif**

if ϕ is $\phi_1\wedge\phi_2$ **then return** $Sat(\phi_1)\cap Sat(\phi_2)$; **endif**

if ϕ is $EX\phi$ **then return** $Sat_{EX}(\phi)$; **endif**

if ϕ is $AX\phi$ **then return** $Sat_{AX}(\phi)$; **endif**

if ϕ is $EG\phi$ **then return** $Sat_{EG}(\phi)$; **endif**

if ϕ is $AG\phi$ **then return** $Sat_{AG}(\phi)$; **endif**

if ϕ is $E(\phi_1U_{\bowtie c}\phi_2)$ **then return** $Sat_{EU}(\phi_1,\phi_2,\bowtie,c)$; **endif**

if ϕ is $A(\phi_1U_{\bowtie c}\phi_2)$ **then return** $Sat_{AU}(\phi_1,\phi_2,\bowtie,c)$; **endif**

由于在 TCTL 的定义中, 除了最后两个算子外其他均是常见的 CTL 算子, 因此本文只给出关于算子 $E(\phi_1U_{\bowtie c}\phi_2)$和 $A(\phi_1U_{\bowtie c}\phi_2)$的计算 $Sat(\phi)$的算法, 即 $Sat_{EU}(\phi_1,\phi_2,\bowtie,c)$和 $Sat_{AU}(\phi_1,\phi_2,\bowtie,c)$. 关于其他算子的计算 $Sat(\phi)$的算法可参考文献[4,15]. 计算 $Sat_{EU}(\phi_1,\phi_2,\bowtie,c)$的伪代码见算法 3.

算法 3.

X: =∅; Y: =$Sat(\phi_2)$; Z: =$Sat(\phi_1)$;
for each $S\in\Omega$ min(S): =max(S): =−1; **endfor**
for each $S\in Y$ min(S): =max(S): =0; **endfor**
if Y=∅ **then return** ∅; **end if**
repeat
  X: =Y;
  for each $S\in Z$
    *flag*1: =0; *flag*2: =0;
    for each $S'\in Y$
      if $(S,S')\in\rightarrow$ **then**
        if min(S)=−1 **then**
          min(S): =min(S')+*time*(S,S'); max(S): =max(S')+*time*(S,S'); *flag*2: =1;
        else if min(S)!=0
          if min(S)>min(S')+*time*(S,S') **then**
            min(S): =min(S')+*time*(S,S'); *flag*2: =1;
          endif
          if max(S)<max(S')+*time*(S,S') **then**
            max(S): =max(S')+*time*(S,S'); *flag*2: =1;
          endif
        endif
        if *flag*1=0 **then** Y: =Y+S; *flag*1: =1; **endif**
      endif
    endfor
  endfor
until X=Y && *flag*2=0;
Z=∅;
if ⋈ =‘<’||⋈ =‘≤’ **then**
  for each $S\in X$

```
      if min(S)⋈ c then Z: =Z+S; endif
    endfor
  endif
  if ⋈ ='>'||⋈ ='≥' then
    for each S∈X
      if max(S)⋈ c then Z: =Z+S; endif
    endfor
  endif
  return Z.
```

在计算 $Sat_{EU}(\phi_1,\phi_2,\bowtie,c)$的过程中, 首先寻找所有满足 $E(\phi_1 U \phi_2)$的状态, 同时对找到的每一个这样的状态设置两个时间函数, 即 min 和 max, 它们根据每次添加的新的满足 $E(\phi_1 U \phi_2)$的状态动态调整其数值. 若用 x 表示存在一个从状态 S 出发的计算满足$\phi_1 U_{=x} \phi_2$, 则 min(S)和 max(S)分别表示 x 的最小值和最大值. 在找到所有满足 $E(\phi_1 U \phi_2)$的状态后, 且它们的 min 值和 max 值均不再改变, 基于 min 和 max 可从中筛选出满足 $E(\phi_1 U_{\bowtie c} \phi_2)$的状态. 这里分为两种情况: 一是当⋈ 为<或≤时, 只需保证 S 的 min 值<或≤c, 即可保证存在一个从 S 出发的计算满足$\phi_1 U_{\bowtie c} \phi_2$; 二是当⋈ 为>或≥时, 只需保证 S 的 max 值>或≥c, 即可保证存在一个从 S 出发的计算满足$\phi_1 U_{\bowtie c} \phi_2$.

计算 $Sat_{AU}(\phi_1,\phi_2,\bowtie,c)$的伪代码见算法 4.

算法 4.

```
X=∅; Y=Sat(φ2); Z=Sat(φ1);
for each S∈Ω min(S): =max(S):=−1; endfor
for each S∈Y min(S): =max(S):=0; endfor
while X≠Y
  X:=Y;
  for each S∈Z
    flag: =0;
    for each S′∈Ω such that (S,S′)∈→
      if S′∉Y then flag: =1; break; endif
    endfor
    if flag=0
      Y: =Y+S;
      for each S′∈Ω such that (S,S′)∈→
        if min(S)=−1 then min(S): =min(S′)+time(S,S′); max(S): =max(S′)+time(S,S′);
        else
          if min(S)>min(S′)+time(S,S′) then min(S): =min(S′)+time(S,S′); endif
          if max(S)<max(S′)+time(S,S′) then max(S): =max(S′)+time(S,S′); endif
        endif
      endfor
    endif
  endfor
endwhile
Z: =∅;
```

if ⋈ =‘<’||⋈ =‘≤’ **then**
  for each $S\in X$
    if max(S)⋈ c **then** Z:=Z+S; **endif**
  endfor
endif
if ⋈ =‘>’||⋈ =‘≥’ **then**
  for each $S\in X$
    if min(S)⋈ c **then** Z: =Z+S; **endif**
  endfor
endif
return Z.

在计算 $Sat_{AU}(\phi_1,\phi_2,\bowtie,c)$的过程中, 首先寻找所有满足 $A(\phi_1 U\phi_2)$的状态, 同时对找到的每一个这样的状态同样设置 min 和 max 时间函数. 与计算 $Sat_{EU}(\phi_1,\phi_2,\bowtie,c)$的不同之处在于, 所有满足 $A(\phi_1 U\phi_2)$的状态的 min 值和 max 值被设置后就不再调整了. 在找到所有满足 $A(\phi_1 U\phi_2)$的状态后, 基于 min 和 max 可从中筛选出满足 $A(\phi_1 U_{\bowtie c}\phi_2)$的状态. 这里分为两种情况: 一是当⋈ 为<或≤时, 只需保证 S 的 max 值<或≤c, 即可保证从 S 出发的所有计算满足$\phi_1 U_{\bowtie c}\phi_2$; 二是当⋈ 为>或≥时, 只需保证 S 的 min 值>或≥c, 即可保证从 S 出发的所有计算满足$\phi_1 U_{\bowtie c}\phi_2$.

基于 Sat(ϕ), 我们可验证ϕ的有效性. 如果 $S_0\in Sat(\phi)$, 则ϕ在 Z 上有效; 否则, ϕ在 Z 上无效.

我们的模型检测算法的复杂度分为两个部分.

- 首先是由点区间优先级时间 Petri 网生成它的状态图. 由于本文所涉及的 Petri 网均是安全的, 因此状态图中的最大标识数不会超过 $2^{|P|}$. 对$\forall t_1\in T_1\cup T_2$ 和$\forall t_2\in T_2$, 我们假设 $h(t_1)$=–1 表示 t_1 在一个标识下不具有发生权, $H(t_2)$=–1 表示 t_2 在一个标识下未挂起, $SI_{\max}$ 表示变迁的所有静态发生点区间中的最大值, 即 $\max\{SI(t)|t\in T_1\cup T_2\}$. 在一个状态下, 一个变迁具有发生权就不会是已挂起变迁, 是已挂起变迁就不会具有发生权. 本文只考虑每个变迁的静态发生点区间为整数的点区间优先级时间 Petri 网, 因此状态图的最大状态数不会超过 $2^{|P|}\cdot(SI_{\max}+2)\cdot(|T_1\cup T_2|+|T_2|)$, 最大状态迁移关系不会超过 $2^{|P|+1}\cdot(SI_{\max}+2)\cdot(|T_1\cup T_2|+|T_2|)$. 因此, 由点区间优先级时间 Petri 网生成状态图的算法复杂度为 $O(2^{|P|}\cdot SI_{\max}\cdot|T_1\cup T_2|)$.
- 然后, 基于生成的状态图验证 TCTL 公式ϕ的有效性. 该过程主要在于计算 $Sat(\phi)$. 对于一个包含 n 个状态和 k 个状态迁移关系的状态图, 计算 $Sat(\phi)$算法的复杂度为 $O((n+k)\cdot|\phi|)$, 其中, $|\phi|$为ϕ中库所和算子的总数量. 因此综合来说, 我们的模型检测算法的复杂度为 $O(2^{|P|}\cdot SI_{\max}\cdot|T_1\cup T_2|\cdot|\phi|)$.

基于以上算法, 我们开发了一个用 C++语言编写的 TCTL 模型检测器. 它在输入一个描述点区间优先级时间 Petri 网的文本型文件和一个描述 TCTL 公式的文本型文件后, 会自动输出验证结果. 图 3(a)展示了描述图 1 中的点区间优先级时间 Petri 网的文本型文件, 图 3(b)展示了描述 TCTL 公式$\phi_1=AG(p_7\rightarrow AF_{\leqslant 10}p_9)$的文本型文件(图中实际上描述的是它的等价形式 $AG(\neg p_7\vee AF_{\leqslant 10}p_9)$), 图 3(c)为验证结果. 结果显示, ϕ_1 无效. 这意味着任务 B 在执行过程中被中断. 事实上, 任务 B 从启动到执行完毕需要 15ms: 首先, 任务 B 启动后获得处理器 c_1 开始执行, 但在执行 5ms 时被更高优先级的任务 A 抢占处理器 c_1 而被迫中断; 接着, 任务 A 在 5ms 后执行完毕, 释放处理器 c_1; 最后, 任务 B 获得处理器 c_1 从中断的地方恢复, 继续执行剩余的 5ms 后结束. 如果我们把ϕ_1 改成 $AG(p_7\rightarrow AF_{\leqslant 15}p_9)$, 则结果显示$\phi_1$ 有效.

```
transition  preset  postset  time  prior  is_suspend
t1          0.      1.       15    99     0
t2          1,9.    2.       0     98     0
t3          2.      3,9.     5     98     0
t4          1,7.    4.       0     98     0
t5          4.      3,7.     5     98     0
t6          5.      6.       10    99     0
t7          6,9.    7.       0     97     0
t8          7.      8,9.     10    97     1
@
place       name        token
0           p1          1
1           p2          0
2           p3          0
3           p4          0
4           p5          0
5           p6          1
6           p7          0
7           p8          0
8           p9          0
9           c1          1
@
```

(a) 描述图 1 中点区间优先级时间 Petri 网的文本型文件

```
formula  math  time  operator  former_formula  latter_formula
1        0     0     p9        0               0
2        <=    10    AF        0               1
3        0     0     p7        0               0
4        0     0     NOT       0               3
5        0     0     OR        2               4
6        0     0     AG        0               5
@
```

(b) 描述公式ϕ_1的文本型文件

```
*C:\Users\vip-2\Desktop\KPNer for TCTL\...
Input TPN-file: TPN.ppn
TPN-file is found and its data has been read!
Input TCTL-file: TCTL.ppn
TCTL-file is found and its data has been read!
All states include:
 S0  S1  S2  S3  S4  S5  S6
Satisfying states include:
 S2  S3  S4  S5  S6
The TCTL formula is invalid in the TPN!
Press any key to continue
```

(c) 验证结果

图 3

5 应 用

本节通过一个多核多任务实时系统的案例, 来说明本文的模型和方法的有效性. 有这样一个多核多任务实时系统, 它有 6 个任务(即 *A*,*B*,*C*,*D*,*E* 和 *F*)和两个处理器(即 c_0 和 c_1), 其中, 任务 *A*, *B* 和 *F* 共用处理器 c_1, 任务 *C*, *D* 和 *E* 共用处理器 c_0. 任务之间存在依赖关系, 即一个任务的执行结束作为下一个任务的启动. 整个系统的任务依赖图如图 4 所示. 任务 *A* 和 *D* 作为起始任务被周期性驱动, 任务 *A* 每过 80 ms 被驱动一次, 任务 *D* 每过 50 ms 被驱动一次. 任务 *A* 获得处理器 c_1 后执行 6 ms, 任务 *D* 获得处理器 c_0 后执行 4 ms. 任务 *A* 执行完毕驱动任务 *B*, 任务 *D* 执行完毕驱动任务 *E*. 任务 *B* 获得处理器 c_1 后执行 10 ms, 任务 *E* 获得处理器 c_0 后执行 22 ms. 任务 *B* 和 *E* 执行完毕驱动任务 *C*, 同时, 任务 *E* 执行完毕驱动任务 *F*. 任务 *C* 获得处理器 c_0 后执行 8 ms, 任务 *F* 获得处理器 c_1 后执行 12 ms. 在图 4 中, 各个任务之间除了依赖关系, 还存在执行的优先级关系, 这里用优先级值表示. 一个任务的优先级值越大, 代表它的优先级越高. 例如: 任务 *A*, *B* 和 *F* 的优先级值分别为 97, 98, 96, 因此任务 *A*, *B* 和 *F* 获得处理器 c_1 执行的优先级为 *B*>*A*>*F*; 任务 *C*, *D* 和 *E* 的优先级值分别为 99, 97, 98, 因此任务 *C*, *D* 和 *E* 获得处理器 c_0 执行的优先级为 *C*>*E*>*D*. 同时, 引入优先级带来了抢占式调度的需求.例如: 任务 *A*, *B* 和 *F* 的优先级为 *B*>*A*>*F*, 因此任务 *A* 可抢占任务 *F* 所占用的处理器, 任务 *B* 可抢占任务 *A* 和 *F* 所占用的处理器; 任务 *C*, *D* 和 *E* 的优先级为 *C*>*E*>*D*, 因此任务 *E* 可抢占任务 *D* 所占用的处理器, 任务 *C* 可抢占任务 *D* 和 *E* 所占用的处理器. 注意一般的抢占式调度机制要求: 首先, 高优先级任务可剥夺低优先级任务的资源, 导致后者被中断; 然后, 前者执行完毕后释放资源; 最后, 后者再次获得资源从中断的地方恢复, 继续执行剩余的一段时间.

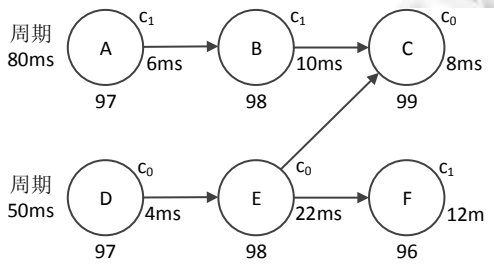


图 4 一个多核多任务实时系统的任务依赖图

这样的一个多核多任务实时系统可通过我们定义的点区间优先级时间 Petri 网来模拟, 如图 5 所示. 由于这里所涉及的优先级关系众多, 因此我们同样给每一个变迁设置一个优先级值, 值越大, 意味着优先级越高. 例如, t_2 的优先级值为 97, t_6 的优先级值为 98, 这意味着存在一个优先级关系$(t_6,t_2)\in Pr$. 此外, 我们定义可挂起变迁集 $T_2=\{t_4,t_5,t_{16},t_{20},t_{21},t_{22}\}$, 其余变迁构成不可挂起变迁集 T_1.

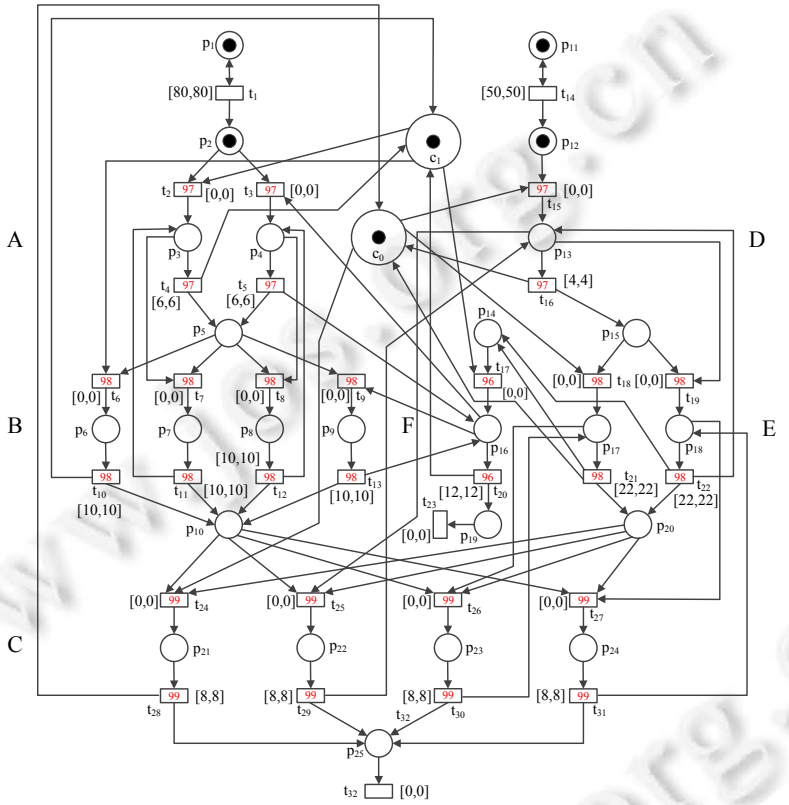


图 5 描述图 4 中的多核多任务实时系统的点区间优先级时间 Petri 网

在图 5 中, 任务 A 和 D 通过变迁 t_1 和 t_{14} 周期性驱动, 任务 A 通过变迁 t_2 和 t_4 表示获得处理器 c_1 然后执行 6 ms, 任务 D 通过变迁 t_{15} 和 t_{16} 表示获得处理器 c_0 然后执行 4 ms, 任务 A 执行完毕通过库所 p_5 驱动任务 B,任务 D 执行完毕通过库所 p_{15} 驱动任务 E, 任务 B 通过变迁 t_6 和 t_{10} 表示获得处理器 c_1 然后执行 10 ms, 任务 E 通过变迁 t_{18} 和 t_{21} 表示获得处理器 c_0 然后执行 22 ms, 任务 B 和 E 执行完毕通过库所 p_{10} 和 p_{20} 驱动任务 C, 同时, 任务 E 执行完毕通过库所 p_{14} 驱动任务 F, 任务 C 通过变迁 t_{24} 和 t_{28} 表示获得处理器 c_1 然后执行 8 ms, 任务 F 通过变迁 t_{17} 和 t_{20} 表示获得处理器 c_0 然后执行 12 ms.

对于共用处理器 c_1 的任务 A, B 和 F, 它们的优先级关系为 $B>A>F$.

- 任务 A 抢占任务 F 的过程通过变迁 t_3 和 t_5 表示;
- 任务 B 抢占任务 F 的过程通过变迁 t_9 和 t_{13} 表示;
- 任务 B 抢占任务 A 的过程分为两种情况: 一是抢占通过与任务 F 竞争获得处理器 c_1 执行的任务 A, 这种情况通过变迁 t_7 和 t_{11} 表示; 二是抢占通过抢占任务 F 获得处理器 c_1 执行的任务 A, 这种情况通过变迁 t_8 和 t_{12} 表示.

对于共用处理器 c_0 的任务 C, D 和 E, 它们的优先级关系为 $C>E>D$.

- 任务 E 抢占任务 D 的过程通过变迁 t_{19} 和 t_{22} 表示;
- 任务 C 抢占任务 D 的过程通过变迁 t_{25} 和 t_{29} 表示;

- 任务 C 抢占任务 E 的过程同样分为两种情况: 一是抢占通过与任务 D 竞争获得处理器 c_0 执行的任务 E, 这种情况通过变迁 t_{26} 和 t_{30} 表示; 二是抢占通过抢占任务 D 获得处理器 c_0 执行的任务 E, 这种情况通过变迁 t_{27} 和 t_{31} 表示.

代表任务 C 和 F 执行结束的库所(p_{19} 和 p_{25})获得 token 后会被清空(t_{23} 和 t_{32}), 这意味着整个系统的所有任务执行完一次后不会结束而是周期性执行.

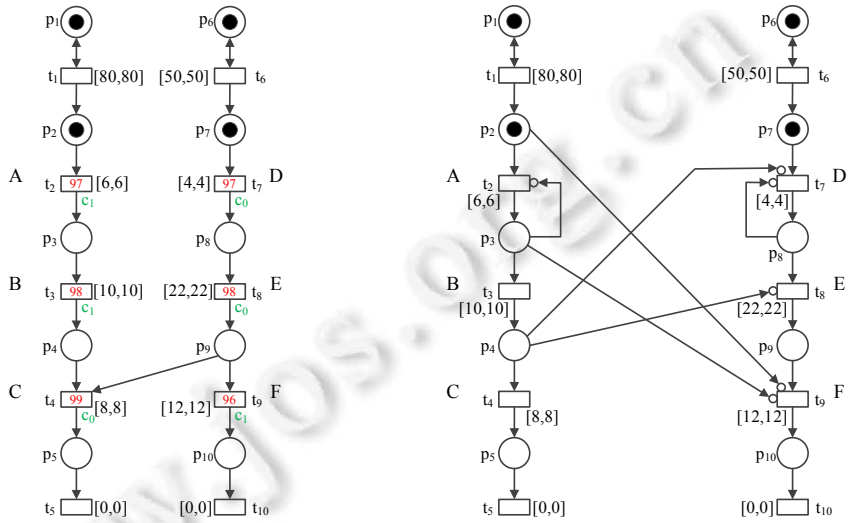
对于这样一个多核多任务实时系统, 我们可验证它与时间相关的一些设计需求, 例如要求任务 D 到任务 C 的延迟不得超过 84 ms. 我们可以使用 TCTL 公式 $\phi_2=AG(p_{12}\rightarrow AF_{\leqslant 84}p_{25})$ 来表示, 其中, p_{12} 意味着任务 D 启动, p_{25} 意味着任务 C 执行完毕. ϕ_2 有效意味着从任务 D 启动到任务 C 执行完毕的最长延迟时间不会超过 84 ms. 首先, 我们用两个文本型文件分别描述图 5 中的点区间优先级时间 Petri 网和公式 ϕ_2; 然后, 把这两个文件输入到我们的模型检测器; 最后, 我们的模型检测器自动输出验证结果(生成的状态图包含 138 个状态, 由于空间限制, 这里不再展示). 结果显示 ϕ_2 无效. 实际上, 当我们把公式 ϕ_2 中的 84 改为任意大于等于 94 的整数时, ϕ_2 就变得有效了. 然而, 把公式 ϕ_2 中的 84 改为任意小于 94 的整数时, ϕ_2 仍然无效. 由于图 5 中变迁的所有静态发生点区间值均为整型变量, 因此我们可以断定: 在系统周期性执行的过程中, 任务 D 到 C 的最长延迟即为 94 ms.

6 相关工作

除了本文提出的时间 Petri 网外, 据我们所知, 目前还有 4 种时间 Petri 网可模拟实时系统的抢占式调度机制, 即调度扩展时间 Petri 网(scheduling extended time Petri nets)[32,33]、抢占式时间 Petri 网(preemptive time Petri nets)[34,35]、带有抑制超弧的时间 Petri 网(time Petri nets with inhibitor hyperarcs)[36]和带有计时器的时间 Petri 网(time Petri nets with stopwatches)[37]. 调度扩展时间 Petri 网是在时间 Petri 网的库所上添加了资源和优先级, 由此从标识中区分出活跃的标识, 即一个标识对应一个活跃(active)标识. 一个变迁在一个标识下具有发生权但在其活跃标识下不具有发生权, 则意味着该变迁被挂起, 一个已挂起的变迁在一个活跃标识下具有发生权, 则意味着该变迁从挂起状态恢复到它挂起前的状态. 抢占式时间 Petri 网与之类似, 它在时间 Petri 网的变迁上添加了资源和优先级, 在一个标识下一个变迁具有发生权但同时和它共享某个资源且优先级更高的变迁也具有发生权, 则意味着该变迁被挂起, 一个已挂起的变迁在一个标识下不存在和它共享某个资源且优先级更高的变迁具有发生权, 则意味着该变迁从挂起状态恢复到它挂起前的状态. 带有抑制超弧的时间 Petri 网是在时间 Petri 网上添加了抑制超弧, 这些抑制超弧蕴含着资源的分配以及任务的优先级, 一个变迁在一个标识下具有发生权但同时存在一个抑制超弧抑制它的发生, 则意味着该变迁被挂起, 一个已挂起的变迁在一个标识下不存在一个抑制超弧抑制它的发生, 则意味着该变迁从挂起状态恢复到它挂起前的状态. 带有计时器的时间 Petri 网是在时间 Petri 网上添加了计时器弧, 这些计时器弧蕴含着资源的分配以及任务的优先级, 一个变迁在一个标识下具有发生权但同时存在着一个和它相关的计时器弧的前集库所为空, 则意味着该变迁被挂起, 一个已挂起的变迁在一个标识下所有和它相关的计时器弧的前集库所非空, 则意味着该变迁从挂起状态恢复到它挂起前的状态.

由于调度扩展时间 Petri 网和抢占式时间 Petri 网之间的相似性以及带有抑制超弧的时间 Petri 网和带有计时器的时间 Petri 网之间的相似性, 本文只选取抢占式时间 Petri 网和带有抑制超弧的时间 Petri 网来与本文定义的点区间优先级时间 Petri 网做对比. 如图 6 所示: 对于图 4 中的多核多任务实时系统, 图 6(a)用抢占式时间 Petri 网来模拟它, 图 6(b)用带有抑制超弧的时间 Petri 网来模拟它. 显然, 抢占式时间 Petri 网和带有抑制超弧的时间 Petri 网均无法显式地刻画资源的占用和释放过程, 因此它们自然不能作为模型来验证一些与资源相关的时间限制需求, 例如一个资源一旦被占用就必须在一个限定的时间内得到释放. 此外, 对于抢占式时间 Petri 网, 由于它在变迁上引入了资源和优先级两个参数, 导致变迁可发生的判定过程要比之前复杂, 从而影响了状态图的生成效率. 对于带有抑制超弧的时间 Petri 网, 它隐藏了资源分配和任务的优先级等信息, 这些参数需要建模者综合考虑, 然后在建模的时候通过抑制超弧隐式地反映出来. 例如在图 4 中, 任务 A 和 B 共用

处理器 c_1 且任务 B 的优先级高于任务 A, 则在图 6(b)中, p_3 与 t_2 之间需要一个抑制超弧以实现 t_3 具有发生权时立即挂起 t_2 的目的. 由于 t_2 代表任务 A, t_3 代表任务 B, 这意味着任务 A 和 B 共享着某个资源且任务 B 的优先级高于任务 A. 显然, 这样给建模者带来了很大的不便, 即使仅仅修改一个任务的优先级值, 就有可能需要重新定义抑制超弧.



(a) 多核多任务实时系统的抢占式时间 Petri 网 (b) 多核多任务实时系统的带有抑制超弧的时间 Petri 网

图 6

对于我们定义的点区间优先级时间 Petri 网, 首先, 它可以显式地刻画资源的占用和释放过程, 因此完全可以用来验证一些与资源相关的时间限制需求; 其次, 它在变迁上引入了优先级这一个参数, 使得变迁可发生的判定过程不会太复杂; 最后, 它通过直接给出优先级关系或优先级值来显式地反应变迁发生的优先级, 使得建模的工作量不至于过大. 注意: 我们定义的点区间优先级时间 Petri 网与以上 4 种时间 Petri 网相比有一个很明显的缺点, 即建模时出现的变迁数爆炸问题, 这也是图 5 中的网结构没有图 6 中的两个网结构简洁的主要原因. 例如在图 4 中, 对于共用处理器 c_1 的任务 A, B 和 F, 优先关系为 $B>A>F$, 因此在图 5 中表示任务 F 需要 1 个变迁, 表示任务 A 需要 2 个变迁, 表示任务 B 需要 4 个变迁. 以此类推, 对于共享资源的、具有不同优先级的多个任务, 表示它们需要的变迁数呈指数级增长. 对于以上 4 种时间 Petri 网来说, 任务和变迁通常是一一对应的. 这是由于抢占式调度机制会导致中断嵌套, 而点区间优先级时间 Petri 网精确模拟了这些中断嵌套过程, 因此才会有这样的结果.

7 结 论

本文基于传统的优先级时间 Petri 网对多核多任务实时系统建模能力的不足, 提出了点区间优先级时间 Petri 网来模拟多核多任务实时系统. 结合 TCTL 模型检测, 设计了相应的算法并开发了相应的工具, 实现了对多核多任务实时系统的正确性验证. 我们下一步的工作主要从以下 4 个方面展开: (1) 优化我们定义的点区间优先级时间 Petri 网, 避免它建模时出现的变迁数爆炸问题; (2) 实现由多核多任务实时系统的任务依赖图自动生成它的点区间优先级时间 Petri 网; (3) 参考基于 Petri 网对 CTL 公式的简化方法[38], 思考基于点区间优先级时间 Petri 网对 TCTL 公式的简化方法; (4) 基于已知的结构化方法如虹吸和陷阱, 以及符号化方法如 OBDD, 来缓解点区间优先级时间 Petri 网的状态爆炸问题.

致谢 本文中的应用实例参考了华为公司的孔辉老师所提供的一个多核多任务实时系统的真实案例, 但考虑其系统安全问题, 本文进行了修改, 并省略了相应任务的具体描述. 在此对孔辉老师表示感谢.

References:

[1] Wang J, Zhan NJ, Feng XY, *et al*. Overview of formal methods. Ruan Jian Xue Bao/Journal of Software, 2019, 30(1): 33−61 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

[2] Cui J, Duan ZH, Tian C, *et al*. Modeling and analysis of nested interrupt systems. Ruan Jian Xue Bao/Journal of Software, 2018, 29(6): 1670−1680 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5472.htm [doi: 10.13328/j.cnki.jos.005472]

[3] Li YN, Deng YX, Liu J. Formal modeling and verification of Paxos based on Coq. Ruan Jian Xue Bao/Journal of Software, 2020, 31(8): 2362−2374 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5960.htm [doi: 10.13328/j.cnki.jos.005690]

[4] Baier C, Katoen JP. Principles of Model Checking. MIT Press, 2008.

[5] Clarke EM, Grumberg O, Kroening D, *et al*. Model Checking. MIT Press, 2018.

[6] Cui J, Duan Z, Tian C, *et al*. A novel approach to modeling and verifying real-time systems for high reliability. IEEE Trans. on Reliability, 2018, 67(2): 481−493.

[7] Bouyer P, Fahrenberg U, Larsen KG, *et al*. Model checking real-time systems. In: Handbook of Model Checking. 2018. 1001−1046.

[8] Dong W, Wang J, Qi ZC. Model checking for concurrent and real-time systems. Journal of Computer Research and Development, 2001, 38(6): 698−705 (in Chinese with English abstract).

[9] Liu GJ, Jiang CJ, Zhou MC. Time-soundness of time Petri nets modelling time-critical systems. ACM Trans. on Cyber-physical Systems, 2018, 2(2): 1−27.

[10] Liu GJ. Primary Unfolding of Petri Nets: A Model Checking Method for Concurrent Systems. Beijing: Science Press, 2020 (in Chinese).

[11] Alur R, Henzinger TA, Mang FY, *et al*. MOCHA: Modularity in model checking. In: Proc. of the Int'l Conf. on Computer Aided Verification. Springer-Verlag, 1998. 521−525.

[12] Lomuscio A, Qu H, Raimondi F. MCMAS: An open-source model checker for the verification of multi-agent systems. Int'l Journal on Software Tools for Technology Transfer, 2017, 19(1): 9−30.

[13] Vardi MY. An automata-theoretic approach to linear temporal logic. In: Proc. of the Logics for Concurrency. Springer-Verlag, 1996. 238−266.

[14] Tao X, Li G. The complexity of linear-time temporal logic model repair. In: Proc. of the Int'l Workshop on Structured Object-oriented Formal Language and Method. Springer-Verlag, 2017. 69−87.

[15] Clarke EM, Emerson EA. Design and synthesis of synchronization skeletons using branching time temporal logic. In: Proc. of the Workshop on Logic of Programs. Springer-Verlag, 1981. 52−71.

[16] Alur R, Henzinger TA, Kupferman O. Alternating-Time temporal logic. In: Proc. of the Int'l Symp. on Compositionality. Springer-Verlag, 1997. 23−60.

[17] Hu L, Liu Z, Hu W, *et al*. Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinfor- cement learning with graph convolutional network. Journal of Manufacturing Systems, 2020, 55: 1−14.

[18] Fauzan AC, Sarno R, Yaqin MA. Performance measurement based on coloured Petri net simulation of scalable business processes. In: Proc. of the 4th Int'l Conf. on Electrical Engineering, Computer Science and Informatics. IEEE, 2017. 1−6.

[19] Lu F, Tao R, Du Y, *et al*. Deadlock detection-oriented unfolding of unbounded Petri nets. Information Sciences, 2019, 497: 1−22.

[20] Merlin PM, Farber DJ. Recoverability of communication protocols implications of a theoretical study. IEEE Trans. on Communications, 1976, 24(9): 1036−1043.

[21] Berthomieu B, Diaz M. Modeling and verification of time dependent systems using time Petri nets. IEEE Trans. on Software Engineering, 1991, 17(3): 259−273.

[22] Liu F, Zhang H. A class of extended time Petri nets for modeling and simulation of discrete event systems. Simulation, 2018, 94(8): 753−762.

[23] Foyo PM, Silva JR. Some issues in real-time systems verification using time Petri Nets. Journal of the Brazilian Society of Mechanical Sciences and Engineering, 2011, 33: 467−474.

[24] Alur R, Courcoubetis C, Dill D. Model-checking for real-time systems. In: Proc. of the 5th Annual IEEE Symp. on Logic in Computer Science. IEEE, 1990. 414−425.

[25] Henzinger TA, Nicollin X, Sifakis J, *et al*. Symbolic model checking for real-time systems. Information and Computation, 1994, 111(2): 193−244.

[26] Hadjidj R, Boucheneb H. On-the-fly TCTL model checking for time Petri net using state class graphs. In: Proc. of the 6th Int'l Conf. on Application of Concurrency to System Design. IEEE, 2006. 111−122.

[27] Hadjidj R, Boucheneb H. On-the-fly TCTL model checking for time Petri nets. Theoretical Computer Science, 2009, 410(42): 4241−4261.

[28] Boucheneb H, Gardey G, Roux OH. TCTL model checking of time Petri nets. Journal of Logic and Computation, 2009, 19(6): 1509−1540.

[29] Berthomieu B, Peres F, Vernadat F. Bridging the gap between timed automata and bounded time Petri nets. In: Proc. of the Int'l Conf. on Formal Modeling and Analysis of Timed Systems. Springer-Verlag, 2006. 82−97.

[30] Berthomieu B, Peres F, Vernadat F. Model checking bounded prioritized time Petri nets. In: Proc. of the Int'l Symp. on Automated Technology for Verification and Analysis. Springer-Verlag, 2007. 523−532.

[31] Reisig W. Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies. Springer-Verlag, 2013.

[32] Roux OH, Déplanche AM. A T-time Petri net extension for real time-task scheduling modeling. European Journal of Automation, 2002, 36(7): 973−987.

[33] Lime D, Roux OH. Expressiveness and analysis of scheduling extended time Petri nets. IFAC Proc. Volumes, 2003, 36 (13): 189−197.

[34] Bucci G, Fedeli A, Sassoli L, *et al*. Modeling flexible real time systems with preemptive time Petri nets. In: Proc. of the 15th Euromicro Conf. on Real-time Systems. IEEE, 2003. 279−286.

[35] Bucci G, Fedeli A, Sassoli L, *et al*. Timed state space analysis of real-time preemptive systems. IEEE Trans. on Software Engineering, 2004, 30(2): 97−111.

[36] Roux OH, Lime D. Time Petri nets with inhibitor hyperarcs: Formal semantics and state space computation. In: Proc. of the Int'l Conf. on Application and Theory of Petri Nets. Springer-Verlag, 2004. 371−390.

[37] Berthomieu B, Lime D, Roux OH, *et al*. Reachability problems and abstract state spaces for time Petri nets with stopwatches. Discrete Event Dynamic Systems, 2007, 17(2): 133−158.

[38] Bønneland F, Dyhr J, Jensen PG, *et al*. Simplification of CTL formulae for efficient model checking of Petri nets. In: Proc. of the Int'l Conf. on Applications and Theory of Petri Nets and Concurrency. Springer-Verlag, 2018. 143−163.

附中文参考文献:

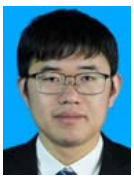
[1] 王戟, 詹乃军, 冯新宇, 等. 形式化方法概貌. 软件学报, 2019, 30(1): 33−61. http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

[2] 崔进, 段振华, 田聪, 等. 一种嵌套中断系统的建模和分析方法. 软件学报, 2018, 29(6): 1670−1680. http://www.jos.org.cn/1000-9825/5472.htm [doi: 10.13328/j.cnki.jos.005472]

[3] 李亚男, 邓玉欣, 刘静. 基于 Coq 的 Paxos 形式化建模与验证. 软件学报, 2020, 31(8): 2362−2374. http://www.jos.org.cn/1000-9825/5690.htm [doi: 10.13328/j.cnki.jos.005690]

[8] 董威, 王戟, 齐治昌. 并发和实时系统的模型检验技术. 计算机研究与发展, 2001, 38(6): 698−705.

[10] 刘关俊. Petri 网的元展: 一种并发系统模型检测方法. 北京: 科学出版社, 2020.



何雷锋(1991－), 男, 博士生, 主要研究领域为 Petri 网, 模型检测, 形式化方法.

刘关俊(1978－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为 Petri 网, 模型检测, 形式化方法, 机器学习, 人机物系统, 工作流系统, 无人机协同系统.