

基于优先级时间 Petri 网的实时嵌入式多核系统分析*

张凯文¹, 刘关俊¹, 孙彦韬¹, 李晓锋², 关健², 解毅², 顾斌²



¹(同济大学 计算机科学与技术系, 上海 201804)

²(北京控制工程研究所, 北京 100190)

通信作者: 刘关俊, Email: liuguanjun@tongji.edu.cn; 李晓锋, E-mail: li_x_feng@126.com

摘要: 已有的基于点区间优先级时间 Petri 网分析实时嵌入式多核系统的工作, 存在以下不足: (1) 点区间优先级时间 Petri 网只考虑每个任务的执行时间是一个固定值的情况, 而更多的实际应用中每个任务的执行时间是在一个区间范围内, 因此不能模拟这些应用; (2) 没有实现从任务依赖图到点区间优先级时间 Petri 网的自动转化, 不便于工程设计人员使用; (3) 没有考虑任务间互斥访问共享变量的情况. 为此, 定义了优先级时间 Petri 网 (Pri-TPN) 以弥补第 1 个不足; 定义带有资源分配与优先级的任务依赖图 (TDG-RAP) 以弥补第 3 个不足; 给出从 TDG-RAP 到 Pri-TPN 的转化规则与算法以弥补第 2 个不足, 以及基于 Pri-TPN 分析任务最坏执行时间与系统死锁的算法; 开发工具软件, 方便工程设计人员使用.

关键词: 实时嵌入式多核系统; 优先级时间 Petri 网; 可达图; 任务依赖图; 最坏执行时间 (WCET); 死锁

中图法分类号: TP311

中文引用格式: 张凯文, 刘关俊, 孙彦韬, 李晓锋, 关健, 解毅, 顾斌. 基于优先级时间 Petri 网的实时嵌入式多核系统分析. 软件学报, 2024, 35(9): 4123–4140. <http://www.jos.org.cn/1000-9825/7129.htm>

英文引用格式: Zhang KW, Liu GJ, Si YT, Li XF, Guan J, Xie Y, Gu B. Analysis of Real-time Embedded Multi-core System Based on Prioritized Time Petri Net. Ruan Jian Xue Bao/Journal of Software, 2024, 35(9): 4123–4140 (in Chinese). <http://www.jos.org.cn/1000-9825/7129.htm>

Analysis of Real-time Embedded Multi-core System Based on Prioritized Time Petri Net

ZHANG Kai-Wen¹, LIU Guan-Jun¹, SUN Yan-Tao¹, LI Xiao-Feng², GUAN Jian², XIE Yi², GU Bin²

¹(Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)

²(Beijing Institute of Control Engineering, Beijing 100190, China)

Abstract: Existing work on the analysis of real-time embedded multi-core systems using point-interval prioritized time Petri nets has the following limitations. (1) Point-interval prioritized time Petri nets only consider the case where the execution time of each task is a fixed value, but in many practical applications, the execution time of a task is generally within a range so that this kind of model cannot be used to analyze these applications. (2) There is a lack of automatic transformation from task dependency graphs to this point-interval prioritized time Petri nets, and thus it is inconvenient for engineering designers. (3) The case of mutually exclusive access to shared variables has not been considered. To address these issues, this study defines prioritized time Petri nets (Pri-TPN) to overcome the first limitation and introduces a task dependency graph with resource allocation and priority (TDG-RAP) to overcome the third limitation. It develops algorithms based on Pri-TPN for analyzing the worst-case execution time (WCET) and system deadlocks of tasks. Additionally, a tool software is developed to facilitate its use by engineering designers.

Key words: real-time embedded multi-core system; prioritized time Petri net (Pri-TPN); reachability graph; task dependency graph; worst-case execution time (WCET); deadlock

* 基金项目: 国家自然科学基金 (62172299, 62192730, 62032019); 北京控制工程研究所高可信嵌入式软件工程技术实验室开放基金 (LHCESET202201); 北京控制工程研究所空间光电测量与感知实验室开放基金 (LabSOMP-2023-03); CCF-华为胡杨林基金-形式化专项 (CCF-HuaweiFM202305)

收稿时间: 2023-09-11; 修改时间: 2023-10-30; 采用时间: 2023-12-13; jos 在线出版时间: 2024-05-15

CNKI 网络首发时间: 2024-05-17

随着实时嵌入式系统规模的增大,单核系统正在向多核系统转变.由于可靠性的要求,多核系统通常在裸机或简单操作系统上开发,缺少有效的并发支持.此外,任务的分配方案由用户的经验决定.如何保证迁移后的系统无死锁、并且满足实时性要求等问题,目前还缺少非常有效的检测手段.因此,对于系统设计来说,分析多核系统上的任务分配是否满足实时性约束并且保证系统无死锁是十分必要的.

关于多核系统的最坏执行时间 (worst-case execution time, WCET),已存在多种技术来分析,包括基于抽象解释^[1]和隐式路径枚举^[2]的方法.这些方法在准确性、复杂性和适用性方面既有其优点又有局限性.其中大部分的研究工作基于有向无环图 (directed acyclic graph, DAG)^[3,4],使用 DAG 图描述任务之间的依赖关系,在给定的调度算法下找到任务的 WCET,从而判断是否满足系统设计的要求.这些工作描述的任务依赖关系较简单,如仅考虑顺序和选择的任务关系,没有考虑任务分发和同步,以及任务的优先级.另一种方案是形式化方法^[5,6],其中模型检测^[7]在实时系统的正确性验证与性能分析问题上已经成功应用.

Petri 网作为用于模型检测的一种形式化建模语言^[8-12],能够直观地刻画事件间的关系(顺序、并发、选择等)与资源分配等原因,广泛应用于实时嵌入式多核系统的调度算法建模和性能分析.时间 Petri 网是一种高级 Petri 网,引入了时间属性,适用于描述系统中与时间相关的行为.在时间 Petri 网^[13,14]中,对于每一个变迁都分配了一对非负整数表示的时间区间,区间的下界与上界分别表示变迁从使能到实际发生所需要的最短和最长时间,而变迁本身的发生是零延迟的.在时间 Petri 网的一个状态下,可能存在多个使能变迁,若变迁之间的可发生时间(变迁已等待的时间与自身静态发生区间的差值)存在交集时,那么根据变迁发生规则,将随机选择一个变迁触发.实际的系统往往包含一些特殊属性,如任务的优先级,对于安全关键任务在与其他任务竞争同一资源时获得优先使用权,系统通常会赋予它们更高的优先级,因此带有优先级的时间 Petri 网更适合描述实时嵌入式多核系统.

在之前的研究中,定义了多种时间 Petri 网的扩展形式来保证与实际系统执行语义的一致性,为了模拟抢占式调度导致的任务挂起和恢复,在库所或者变迁中加入优先级来表示任务的优先需求^[15-17],同时将每个变迁的发生区间定义为一个上界与下界相等的点区间,表示单任务的最坏执行时间(不被中断挂起情况下)^[13].基于时序逻辑公式分析任务的最坏执行时间(执行过程中可能被中断挂起的情况),但提出的模型并不能满足所有任务的依赖关系;同时点区间的描述方式与中断或随机任务的发生语义不符.

本文扩展了之前工作中定义的点区间优先级时间 Petri 网^[13,18-22]的变迁发生规则,以解决对于实时系统中任务波动或者中断任务等非确定性时间建模能力的不足,我们也称之为优先级时间 Petri 网,它不仅考虑了任务执行的优先级和调度需求,还扩展了任务执行时间的发生语义:允许变迁的静态发生区间是一个时间范围而不是一个固定的值,以此表示中断程序和随机任务的发生语义;还考虑任务之间的并发机制,如锁的使用^[23].在本文中,根据性能分析的需求,定义变迁的静态发生区间为实际系统中任务的最好执行时间和最坏执行时间,相比于点区间,这种变迁从获得发生权到发生的等待时间可以不同,因此优先级时间 Petri 网模型具有更强的建模能力,以及性能分析的能力.对于锁的使用,在多核系统中,自旋锁和互斥锁是两种常用的锁类型,一种保证处理器之间的共享资源访问,另一种保证同一个处理器下的共享资源访问^[24],在向多核系统移植的过程中,需要考虑并发机制带来的与时间相关问题,这对于实时系统是十分必要的,而本文提出的优先级时间 Petri 网以及相应的任务依赖图能够描述这些锁的应用.

本文第 1 节介绍一些基本概念,包括任务依赖图、Petri 网、时间 Petri 网,并给出优先级时间 Petri 网的定义与发生语义,给出了求解 WCET 的算法.第 2 节介绍一种带有资源分配与优先级的任务依赖图,并给出这种任务依赖图到优先级时间 Petri 网的转换规则与实现算法.第 3 节通过一个实例来验证模型、算法的正确性以及所开发的工具的有效性.第 4 节介绍基于时间 Petri 网的扩展网在实时系统建模与分析方面的相关工作.第 5 节对全文总结并阐述下一步工作.

1 优先级时间 Petri 网

本节形式化定义 Petri 网、时间 Petri 网以及优先级时间 Petri 网 (prioritized time Petri net, Pri-TPN),展示如何

使用 Pri-TPN 模拟带优先级的多核多任务系统, 展示资源 (如处理器、锁) 的分配与抢占等, 并给出 Pri-TPN 的状态类图的定义以及基于状态类图分析死锁、计算 WCET 的算法。N 是自然数集, \mathbb{R} 是实数集。定义实数闭区间 $I = \{x \in \mathbb{R} \mid a \leq x \leq b\}$, 记为 $I = [a, b]$, 这里 $\downarrow I = a$, $\uparrow I = b$ 。I 表示所有实数闭区间的集合。

1.1 Petri 网

定义 1. 一个 Petri 网定义为 4 元组 (P, T, F, M_0) , 其中 P 为库所 (place) 的集合, T 为变迁 (transition) 的集合, $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ 为流关系 (flow relation), M_0 为初始标识 (initial marking)^[15]。

通常将 $N = (P, T, F)$ 记为一个网, 将 (N, M_0) 记为一个 Petri 网。一个 Petri 网可以看作一个有向二分图, 其中圆圈型的节点代表库所, 方框型的节点代表变迁, 连接圆圈与方框的弧代表流关系。给定一个节点 $x \in P \cup T$, 节点 x 的前驱定义为 $\bullet x = \{y \in P \cup T \mid F(y, x) = 1\}$, 后继定义为 $x^\bullet = \{y \in P \cup T \mid F(x, y) = 1\}$ 。库所中的小黑点称作托肯 (token), 托肯在库所中的一个分布表示一个标识 (marking, 表示系统的一个状态), Petri 网的标识可以形式化定义为一个库所集上的一个袋集 $M: P \rightarrow \mathbb{N}$, $M(p)$ 表示一个库所 p 在 M 下被标识的托肯数量, 代表一个局部状态。如果 $\forall p \in \bullet t: M(p) > 0$, 则称变迁 t 在标识 M 下使能, 即该变迁获得发生权, 记为 $M[t]$ 。在 M 下发生使能的变迁 t , 产生新标识 M' , 记作 $M[t]M'$, 其中 M' 的求解如下: (1) $M'(p) = M(p) - 1, \forall p \in \bullet t \setminus t^\bullet$; (2) $M'(p) = M(p) + 1, \forall p \in t^\bullet \setminus \bullet t$; (3) 其他: $M'(p) = M(p)$ 。从 M 出发的所有可达标识集合记作 $R(N, M)$ 。如果在可达标识 M 下, 每个变迁都不能使能并且 M 不是终止标识 M_d , 即: $M \neq M_d \wedge \forall t \in T: \neg M[t]$, 则称 M 为 Petri 网的一个死锁 (deadlock)。如果 $\exists k \in \mathbb{N}, \forall p \in P, \forall M \in R(N, M): M(p) \leq k$, 称此 Petri 网是 k 有界的, 当 $k=1$ 时, 则称此 Petri 网是安全的 (safe)。本文考虑的 Petri 网都是安全的。

1.2 时间 Petri 网

定义 2. 一个时间 Petri 网可以用五元组 (P, T, F, M_0, SI) 来表示, 其中, (P, T, F, M_0) 为一个 Petri 网, $SI(t): t \in T \rightarrow I$ 表示此 Petri 网中变迁 t 的静态发生区间 (static firing interval)^[13]。

$En(M)$ 是在标识 M 下使能变迁的集合, (M, h) 表示时间 Petri 网的一个局部状态, 其中, $h: En(M) \rightarrow \mathbb{R}^+$ 是一个获得发生权的变迁的已等待时间的时钟映射, $h(t)$ 表示变迁 t 的已等待时间。当变迁 t 标识 M 下使能, 并且已等待时间在该变迁的静态发生区间内, 称变迁在状态 (M, h) 下是可发生。

1.3 带有优先级的时间 Petri 网

定义 3. 一个 Pri-TPN 是一个 7 元组 $Z = (P, T_1, T_2, F, M_0, SI, Pri)$, 其中, $(P, T_1 \cup T_2, F, M_0, SI)$ 为一个时间 Petri 网, T_1 为不可挂起变迁集, T_2 为可挂起变迁集, 且二者不相交; $Pri \subseteq (T_1 \cup T_2) \times (T_1 \cup T_2)$ 是变迁之间的优先级关系^[13]。

与时间 Petri 网不同, 判断一个变迁在一个状态下是否可发生还需要不存在与其他变迁的优先级关系, 为了区别, 本文称一个变迁在优先级时间 Petri 网的一个状态下是可调度的 (schedulable)。另外, $(t_1, t_2) \in Pri$ 意味着 t_1 的优先级高于 t_2 , 在 Petri 网图中我们通常用数字表示变迁的优先级, 数字越大优先级越高。优先级时间 Petri 网中存在两种状态迁移规则, 令 $S = (M, h, H)$ 表示优先级时间 Petri 网的一个状态, $H(t): t \in T_2 \rightarrow \mathbb{R}^+$ 表示可挂起变迁挂起前的已等待时间。

(1) $S \xrightarrow{\tau} S'$ 表示从状态 $S = (M, h, H)$ 经过 τ 个时间后到达状态 $S' = (M', h', H')$, 这种状态迁移并不会产生新的标识, 在标识 M 下的所有使能变迁, 其已等待时间同步增加, 同时, 若可挂起变迁没有获得发生权, 那么它们的中断时间也保持不变, 即:

- ① $M = M'$;
- ② $\forall t \in En(M): h'(t) = h(t) + \tau \leq \uparrow SI(t)$;
- ③ $\forall t \in T_2 \setminus En(M): H'(t) = H(t)$ 。

(2) $S \xrightarrow{t} S'$ 表示在状态 $S = (M, h, H)$ 下发生变迁 t 到达状态 $S' = (M', h', H')$, 首先, 在当前标识下, 不存在其他变迁与发生变迁存在优先级关系, 如果有, 那么此变迁不会发生; 如果在新的标识下, 除发生变迁 t 之外的变迁一直使能, 那么这些变迁的已等待时间保持不变, 而变迁 t 和其他不在具有发生权的变迁的已等待时间会被重新置

0; 如果在原有标识下没有发生权的变迁在新的状态标识下获得发生权, 那我们需要区分该变迁属于不可挂起变迁还是可挂起变迁, 对于可挂起变迁, 将从上一次被挂起的时刻继续等待发生 ($H_0(t) = 0$), 对于不可挂起变迁, 已等待时间为 0; 如果在原有标识下的具有发生权的不可挂起变迁不再使能, 那么它的已等待时间也会置 0, 而可挂起变迁则会记录当前已等待时间, 即:

- ① $t \in En(M)$;
- ② $h(t) \in SI(t)$;
- ③ $\forall t' \in T_1 \cup T_2$; 如果 $t' \in En(M)$ 且 $h(t') \in SI(t')$, 则 $(t', t) \notin Pri$;
- ④ $M[t]M'$;
- 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 $\forall t' \notin En(M)$: 如果 $t' \in En(M')$ 且 $t' \in T_2$, 则 $h'(t') = H(t')$, 否则 $H'(t') = H(t')$;
- ⑦ $\forall t' \notin En(M)$: 如果 $t' \in En(M')$ 且 $t' \in T_1$, 则 $h'(t') = 0$;
- ⑧ $\forall t' \in En(M) \cap T_2$ 且 $t' \neq t$: 如果 $t' \notin En(M')$, 则 $H'(t') = h(t')$.

下面通过一个简单的例子来区别之前的工作并解释本文所定义 Pri-TPN 的相关概念.

图 1 为一个优先级时间 Petri 网, 左侧表示任务 A ($p_2 \rightarrow t_2 \rightarrow p_3 \rightarrow t_3 \rightarrow p_4$, $p_2 \rightarrow t_4 \rightarrow p_5 \rightarrow t_5 \rightarrow p_4$), 右侧表示任务 B ($p_7 \rightarrow t_7 \rightarrow p_8 \rightarrow t_8 \rightarrow p_9$), 其中, $\{t_8\}$ 为可挂起变迁集, 用实心矩形表示, 其余变迁构成不可挂起变迁集, 用空心矩形表示. 此示例描述的是一个分配在同一处理器上的两个不同任务. 任务 A 的优先级为 98, 高于任务 B. 在图 1 所示 Pri-TPN 的初始标识下的 10–15 个时间单位间可发生变迁 t_1 驱动任务 A, 在 8–12 个时间单位间可发生变迁 t_6 驱动任务 B, 任务 A 和任务 B 可以按照不同的先后顺序发生 (与点区间优先级时间 Petri 网不同, 其任务执行时间固定, 会忽略掉任务的一些执行顺序), 假设在 9 个时间单位后发生变迁 t_6 , 而此时变迁 t_1 还未发生, 因此此时任务 B 获得处理器被调度执行 (即, 发生变迁 t_7), 任务 B 需要 9–10 个时间单位才能执行完毕, 在此期间任务 A 也被触发 (发生变迁 t_1). 由于任务 A 的优先级高于任务 B, 所以任务 A 可以抢占处理器 c_1 (发生 t_4), 导致任务 B 被挂起 (t_8 的时钟停止计时), 在任务 A 执行完毕释放处理器后, 任务 B 继续执行 (t_8 的时钟从挂起处重新计时).

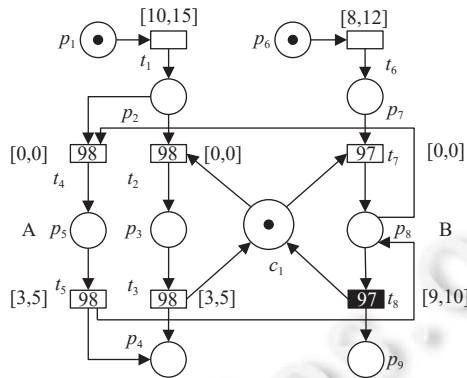


图 1 一个可抢占系统的优先级时间 Petri 网

1.4 使用 Pri-TPN 表示分配处理器与锁的例子

在多核场景下, 存在数据竞争的场景, 正常使用锁可以大大提高系统的性能, 否则不仅拖慢性能, 甚至会造成死锁. 加锁的目的是保证共享资源在任意时间内, 只有一个线程可以访问, 一个带锁的任务可以用 Pri-TPN 表示, 其中锁的类型分为互斥锁和自旋锁. 在多核系统中, 如 Linux, 自旋锁的实现方式为: 禁止内核抢占且自旋, 在考虑中断请求的状况下, 对于自旋锁, 为了保证互斥性, 必须禁用当前处理器的中断, 否则可能导致多个核心同时持有自旋锁而出现的竞争情况, 在 Pri-TPN 中实现为将自旋锁加锁后的变迁标注为不可挂起变迁.

图 2 对图 1 的 Petri 网进行扩展, 当任务 B 获取处理器资源后, 对后续任务的执行使用互斥锁, 如图 2(a), 可挂起变迁为 $\{t_{18}, t_{19}, t_{20}\}$, 每个可挂起变迁都表示有一个高优先级任务抢占相关资源执行, 如可挂起变迁 t_{18} 对应任务

A 执行 $(p_1 \rightarrow t_5 \rightarrow p_6 \rightarrow t_6 \rightarrow p_7 \rightarrow t_7 \rightarrow p_8 \rightarrow t_8 \rightarrow p_5)$, 其余变迁构成不可挂起变迁集, 所以当高优先级任务抢占任务 B 所持有的资源时, 可在库所 $\{p_{16}, p_{17}, p_{18}\}$ 处中断任务 B. 当任务使用自旋锁时, 如图 2(b), 可挂起变迁为 $\{t_{18}, t_{20}\}$, 其余变迁构成不可挂起变迁集.

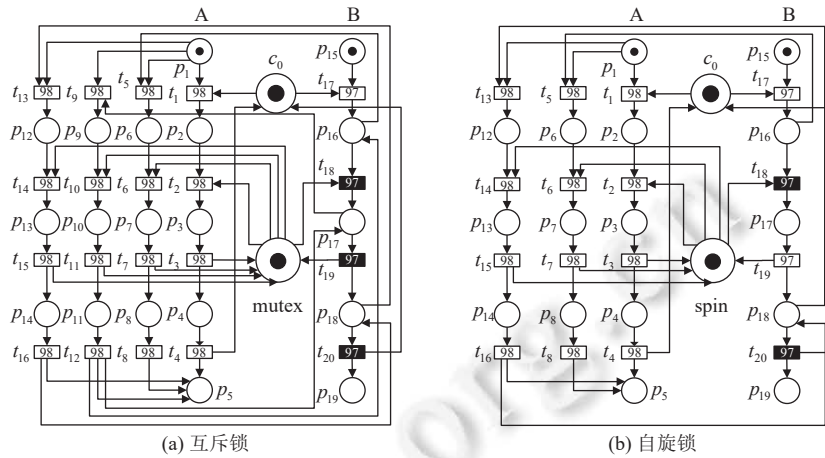


图 2 带互斥锁和自旋锁的优先级时间 Petri 网

1.5 优先级时间 Petri 网的状态类图

根据时间 Petri 网的状态迁移规则, 将变迁的发生划分为两个阶段: 一是变迁可发生但未发生, 即标识保持不变, 而具有发生权的变迁随着时间流逝, 该变迁的已等待时间到达了静态发生区内; 二是发生该变迁. 因此, 在时间 Petri 网中, 这两种状态迁移规则可以用同一个状态序列表示, 即 $S \xrightarrow{\tau} S' \xrightarrow{t} S''$ 等价于 $S \xrightarrow{(\tau, t)} S''$, 表示状态 S 在等待 τ 个时间后到达第 2 阶段发生变迁 t 到达状态 S'' . 但在 Pri-TPN 中, 从获得发生权到发生要经历 3 个阶段, 第 1 个阶段与时间 Petri 网相同; 第 2 阶段, 该变迁在发生区内任意时间段发生, 但由于其他变迁更早到达截止时间, 其没有到达最终时间; 第 3 阶段和第 2 阶段是并列的关系, 是该变迁到达最长执行时间后发生, 从而产生新的状态, 在新状态下继续发生, 此阶段按实际情况可继续扩展. 前两种规则合并方式与点区间时间 Petri 网相同, 第 3 种规则定义为 $t_1, t_2, \dots, t_{n-1} \in T, S \xrightarrow{\tau} S^1 \xrightarrow{t_i} S^2 \xrightarrow{t_n} S^n$, 它表示在变迁获得发生权后, 可在发生区间的任意时刻发生, 到达新状态.

优先级时间 Petri 网的状态图 $\Delta = (\Omega, S_0, \Gamma, label, time)$ 定义与点区间优先级时间 Petri 网的状态图相同^[13], 不同之处在于状态之间的迁移关系的时钟映射可以表示为一个区间, 表示发生变迁 t 在状态 S 下的可发生区间.

状态图可能有无限个状态, 因此不适合做分析工具, 但幸运的是文献 [25] 提出了状态类图, 用有限个状态类以及状态类的迁移表示整个系统的所有可能行为, 更多原理性的描述可参见文献 [26], 在本文中, 为方便讨论, 仍然使用符号 S 表示状态类. 算法 1 给出了一个抽象的生成过程. 下面先通过例子来展示其生成过程. 图 3 是图 1 中的 Pri-TPN 的状态类图, 共有 14 个状态类. 下面通过几个简单例子来理解状态类的生成.

- 在状态类 S_0 下, 即初始状态类下, 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_6 发生所需要的时间为 $[8, 12]$ 个时间单位, t_1 所需要的最短时间为 10 个时间单位. 如果在 S_0 下选择 t_1 作为发生的变迁, 由于其时钟到达 12 个时间单位时 t_6 必须发生, 所以 t_1 在 S_0 下的动态发生区间为 $[10, 12]$ (尽管它的静态发生区间是 $[10, 15]$). 因此, 在 S_0 下发生 t_1 到达状态类 S_7 , 而对于状态类 S_7 来说, t_6 的时钟域为 $[10, 12]$, 表示 t_6 所有可能的已等待的时间. 在状态类 S_7 下, t_2 刚获得发生权, 因此它的已等待时间为 0.

- 在状态类 S_8 下, t_2 和 t_7 的动态发生区间均为 $[0, 0]$, 但由于变迁 t_2 的优先级高于 t_7 , 故在状态类 S_8 下, 变迁 t_7 不允许被发生, 所以, 在状态类 S_8 处只有发生 t_2 而产生的状态类 S_9 .

- 在状态类 S_3 下, 变迁 t_4 的优先级高于可挂起变迁 t_8 的优先级, 并且发生 t_4 使得 t_8 失去发生权, 因此发生 t_4

而产生状态类 S_4 , 在 S_4 中 t_8 并且被挂起其挂起的时钟域为 $H(t_8) \in [0, 7]$. 在状态类 S_4 下只有 t_5 有发生权, 发生 t_5 到达状态类 S_5 , 此时可挂起变迁 t_8 重新获得发生权, 因此 t_8 的时钟又开始计时, 其可能的已等待时间为 $h(t_8) \in [0, 7]$.

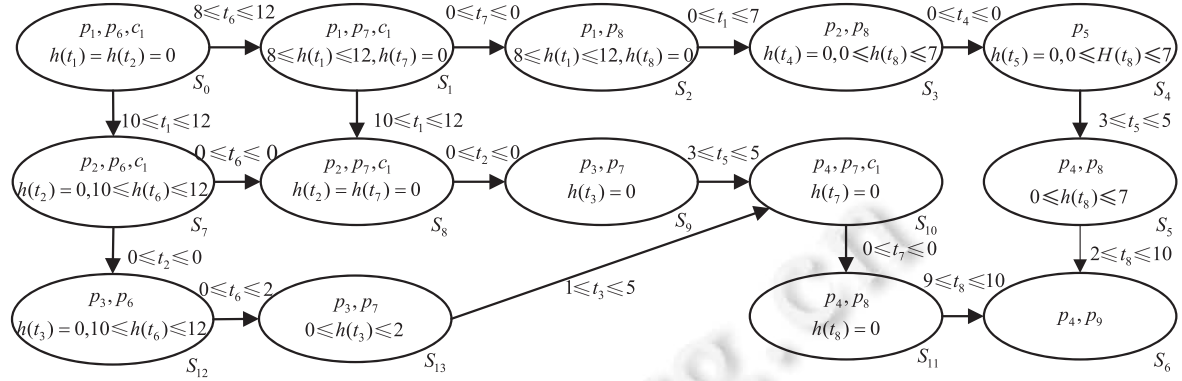


图3 图1的优先级时间Petri网的状态类图

生成状态类图算法如算法1, 首先根据当前状态类下, 有哪些变迁具有发生权, 获取所有可能发生的变迁, 在算法中由 *find_sched_t* 函数实现: 找到在当前时钟域下具有发生权的变迁, 获得当前状态类下所有变迁最大发生时间和最小发生时间减去已运行时间的时钟域, 找到其中最小的发生时间和最大的发生时间, 此区间即为当前状态类下的可发生时钟域(动态发生区间), 在该时钟域下的变迁为可调度变迁. 变迁在到达最大发生后, 需要重新将其时钟置0, 当下次获得发生权后, 重新进行计时. 其中, 对于可挂起变迁和不可挂起变迁来说, 其任务的发生时间的处理方式相同, 但对于不可挂起变迁, 其从获得发生权开始, 一直到产生新状态, 一直保持计时; 对于可挂起变迁, 当发生权被抢占时, 计时停止. 在获得当前状态类下的可调度变迁后, 判断变迁所属任务之间的优先级和处理器资源分配关系, 在同一个处理器上的高优先级变迁可抢占低优先级变迁, 即在此状态类下, 低优先级变迁不会发生.

算法1. 生成状态类图.

输入: 优先级时间Petri网 $Z = (P, T_1, T_2, F, M_0, SI, Pri)$;

输出: Z 的状态类图 Δ .

$\Omega = \emptyset; Form = \emptyset; S_0 = (M_0, h_0, H_0)$;

//找到当前状态下的可调度变迁

function *find_sched_t*(S) $\rightarrow T_{sched}$;

$T_{sched} = \emptyset; En(S) = \emptyset$;

for 每一个 $t \in T_1 \cup T_2$ 在状态类 S 下 do

if $\bullet t = 1$ then

$Add(t) \rightarrow En(S)$;

endif

endfor

$SI_{sched} = (0, 0)$;

for 每一个 $t \in En(S)$ do

if $(\downarrow SI(t) - \uparrow h(t)) \leq \downarrow SI_{sched}$ then

$\downarrow SI_{sched} = \downarrow SI(t) - \uparrow h(t)$

endif

```

if ( $\uparrow SI(t) - \downarrow h(t) \leq \uparrow SI_{sched}$ ) then
     $\uparrow SI_{sched} = \uparrow SI(t) - \downarrow h(t)$ ;
endif
endfor
for 每一个  $t \in En(S)$  do
     $SI_t = (0, 0)$ ;
    if ( $\downarrow SI(t) - \uparrow h(t) \geq \downarrow SI_{sched}$ ) then
        if ( $\uparrow SI(t) - \downarrow h(t) \leq \uparrow SI_{sched}$ ) then
             $SI_t = (\downarrow SI(t) - \uparrow h(t), \uparrow SI(t) - \downarrow h(t))$ ;
        else
             $SI_t = (\downarrow SI(t) - \uparrow h(t), \uparrow SI_{sched})$ ;
        endif
         $Add(t, SI_t) \rightarrow T_{sched}$ ;
    endif
endfor
for 每一对  $(t_1, t_2) \in T_{sched}$  do
    //  $t.c$  表示该变迁所代表任务需要的处理器资源
    if ( $(\exists (t_1, t_2) \in Pri) \wedge (t_1.c == t_2.c)$ ) then
         $remove(T_{sched}, t_2)$ ;
        if  $t_2 \in T_2$  then
             $H(t_2) = h(t_2)$ ;
        endif
    endif
endfor
return  $T_{sched}$ ;
// 从初始状态开始生成状态类图
repeat
     $S = Form.front()$ ;
    for 每一个  $t \in find\_sched\_t(S)$  do
         $\forall \tau \in SI_t$  such that  $S \xrightarrow{\tau} S'$ ;
        if  $S' \notin \Delta$  then
             $\Omega = \Omega + S'$ ;  $Add(S, S')$  to  $\Gamma$ ;
             $label(S, S') = SI_t$ ;  $time(S, S') = SI_{sched}$ ;
        endif
    endfor
     $Add(S') \rightarrow Form$ ;
until  $Form = \emptyset$ ;
return  $(\Omega, S_0, \Gamma, label, time)$ ;

```

1.6 基于状态类的最坏执行时间分析算法

任务的开始和结束在 Petri 网中有对应的库所表示, 如图 1 中, 任务 A 的开始以库所 p_2 表示, 结束以库所 p_4

表示, 任务 B 的开始以库所 p_7 表示, 结束以库所 p_9 表示. 在生成优先级时间 Petri 网的状态类图后, 计算给定起始任务和结束任务 (a_1, a_2) 的 WCET, 起始任务和结束任务可为相同任务, 以 $a_1.start$ 表示任务 a_1 的开始库所, $a_2.end$ 表示 a_2 的结束库所. 计算它们最坏执行时间的算法如算法 2.

算法 2. WCET 分析.

输入: 给定任务 (a_1, a_2) , 状态类图 Δ ;

输出: 任务的 WCET.

```

entry =  $\emptyset$ ; exit =  $\emptyset$ ; weight =  $\emptyset$ ; wcet = 0;
for 每一个  $S \in \Omega$  do
//  $M$  为状态类  $S$  下的标识
  if  $a_1.start \in S.M$  then
    Add( $S$ )  $\rightarrow$  entry;
  endif
  if  $a_2.end \in S.M$  then
    Add( $S$ )  $\rightarrow$  exit;
  endif
endfor
for 每一个  $S_{start} \in entry$  do
  for 每一个  $S_{end} \in end$  do
    //若  $time$  出现负数, 则计为 0
    Add(DFS( $S_{start}, S_{end}, A$ ))  $\rightarrow$  weight;
  endfor
endfor
return wcet = max(weight);

```

在 WCET 分析算法中, 可指定两个不同的任务作为输入, 在状态类图中找到起始任务的进入状态, 结束任务的退出状态, 对于两个状态节点进行深度优先遍历, 找到路径权重的最大值即为 WCET. 在深度优先遍历时, 遇到包含任务开始和结束的库所时结束遍历以防止进入循环导致的任务重启状态.

如计算图 1 中任务的 WCET, 给定任务 A, 在图 3 所示的状态类图中找到包含 p_2 的所有状态类 $S_{p_2} = \{S_3, S_7, S_8\}$ 和包含 p_4 的所有状态类 $S_{p_4} = \{S_5, S_6, S_{10}, S_{11}\}$, 依据算法以及图 3 可得到最大权重的路径有 3 条, 分别为 $S_3 \rightarrow S_4 \rightarrow S_5$ 、 $S_7 \rightarrow S_8 \rightarrow S_9 \rightarrow S_{10}$ 和 $S_7 \rightarrow S_{12} \rightarrow S_{13} \rightarrow S_{10}$, 计算得到任务 A 的 WCET 为 5 个时间单位. 类似地, 可以求出任务 B 的 WCET 为 15 个时间单位.

2 带有资源分配与优先级的任务依赖图

2.1 任务依赖图

DAG 是有向图的一种特例, 其主要特点是图中没有环. 由于有向无环图能够较好地表示任务之间的依赖关系, 因此常被用于直观描述调度问题. 但在嵌入式多核系统中, 周期任务普遍存在, 因此本文对 DAG 图进行扩展, 加入自环等, 并且在节点上加上任务名称、资源使用、优先级、执行时间等标签, 用于描述更复杂的多核多任务系统.

定义 4. 一个带有资源分配与优先级的任务依赖图 (task-dependence graph with resource allocation and priority, TDG-RAP) 是一个七元组 $G = (V, E, A, Lock, C, Pr, l)$, 其中,

(1) $V = V_{task} \cup V_{wait} \cup V_{dist}$ 是节点的集合, 节点分为 3 类, 分别表示任务节点, 同步节点和分发节点, 一个同步节

点表示: 只有当它的前驱节点所对应的任务均执行完以后, 它的后继节点所对应的任务才可以执行; 一个分发节点表示: 在前驱节点完成后, 它的后继节点对应的任务同时被触发. 需要注意的是, 本文暂未考虑同步节点和分发节点互为前驱后继的情况.

(2) $E \subseteq V \times V$ 是边的集合, E 中一个自环表示相应节点对应的任务是周期性发生的, 为简便起见, 本文把发生的周期标注在自环上.

(3) $A = \{a_1, a_2, \dots, a_m\}$ 是任务的集合.

(4) $Lock = Lock_{mu} \cup Lock_{sp}$ 是任务所用到的锁的集合, 锁的类型分为互斥锁和自旋锁.

(5) $C = \{c_1, c_2, \dots, c_s\}$ 表示计算资源集合, 本文考虑的计算资源是处理器, 而从 Petri 网模型的角度, 锁也被看作是一种资源.

(6) $Pr \subseteq \mathbb{N}$ 是一组正整数, 代表任务的优先级.

(7) $l: V_{task} \rightarrow A \times C \times (Lock \times Lock \dots \cup \emptyset) \times (\mathbb{I} \times \mathbb{I} \times \mathbb{I} \dots) \times Pr$ 是一个标签函数, 它指出了在一个任务节点所对应的任务的名称, 该任务所使用的计算资源, 该任务使用的锁的顺序 (\emptyset 的情况表示该任务没有使用任何锁), $\mathbb{I} \times \mathbb{I} \times \mathbb{I} \dots$ 指明了该任务由于使用锁而被细化的各子任务的执行时间区间 (当该任务没有使用锁时, 这里只有一个区间, 表示该任务的执行时间; 当该任务使用了 k 个锁时, 这里就对应 $k+1$ 个区间), Pr 表示该任务的优先级.

图 4 是一个 TDG-RAP, 任务 a_1 (节点 v_1 表示) 和任务 a_4 (节点 v_5 表示) 作为起始任务被周期性驱动, 任务 a_1 和任务 a_4 的周期驱动时间分别为 80 ms 和 50 ms. 节点 v_3 是一个同步节点, 表示任务 a_3 (节点 v_4 表示) 需要等待任务 a_2 (节点 v_2 表示) 和任务 a_5 (节点 v_7 表示) 都完成之后, 才开始执行. ($a_1, c_1, \emptyset, [3, 5], 97$) 是节点 v_1 的标签属性, 其中 a_1 代表对应于节点 v_1 的任务名称, 该任务分配在处理器 c_1 上执行, $[3, 5]$ 是任务的执行时间区间, 98 是该任务的优先级, 优先级数值越大表示优先级越高, 在该任务中没有使用任何锁资源. ($a_4, c_0, lock_1, ([10, 12, 2, 4]), 97$) 表示任务 a_4 分配到处理器 c_0 上执行, 优先级为 97, 该任务使用了自旋锁 $lock_1$, 因此该任务被细化为两个子任务, 一个是任务加锁后处于临界区内的执行时间 $[10, 12]$, 另一个是处于临界区外的任务执行时间 $[2, 4]$, 临界区内执行时间在前, 并按照加锁顺序对应于每个锁的临界区时间.

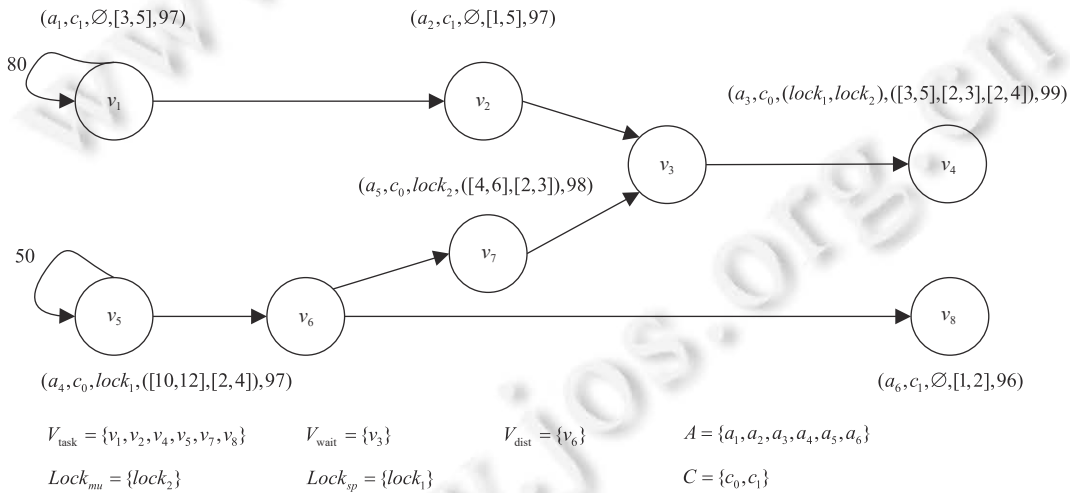


图 4 一种 TDG-RAP 表示的任务依赖关系模型

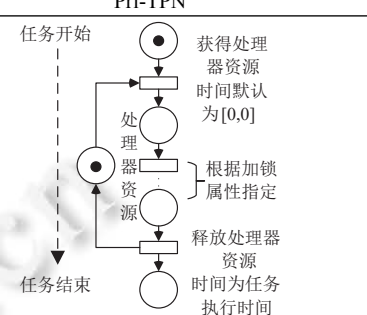
2.2 TDG-RAP 到 Pri-TPN 的转换规则

从 TDG-RAP 中收集所有的任务信息, 针对不同节点给出了 5 种转换规则, 如表 1 所示.

$Rule_1$: TDG-RAP 中的任务节点可以转换为 $(P \rightarrow T \rightarrow P \rightarrow T \rightarrow P)$ 的基本 Petri 网模型, 托肯的流动表示为任务激活、请求处理器资源、任务执行、释放处理器资源和任务终止的状态转移, 每个任务节点对应的 Petri 网都会绑定相应的处理器上. 如果任务中使用锁变量, 将任务执行对应的库所 P 进行扩展, 如图 2(a) 所示, 任务 B 中使

用了锁之后的模型表示为 $p_{15} \rightarrow t_{17} \rightarrow (p_{16} \rightarrow t_{18} \rightarrow p_{17} \rightarrow t_{19} \rightarrow p_{18}) \rightarrow t_{20} \rightarrow p_{19}$, 括号中的部分对应模型的扩展, 分别为表示请求锁资源、加锁、临界区执行、解锁和继续执行任务.

表 1 TDG-RAP 到 Pri-TPN 的转换规则

| 规则 | TDG-RAP | Pri-TPN |
|----------|--|---|
| $Rule_1$ | V_{task} |  |
| $Rule_2$ | V_{wait} | 前置任务 >1 后继任务 |
| $Rule_3$ | V_{dist} | 前置任务 后继任务 >1 |
| $Rule_4$ | $e \in \{V \times V \mid v_1, v_2 \in V \wedge v_1 \neq v_2\}$ | 前置任务 后继任务 |
| $Rule_5$ | $e \in \{V \times V \mid v_1, v_2 \in V \wedge v_1 = v_2\}$ | 周期任务 任务开始 |

$Rule_2$: 为表示任务之间的同步关系, 定义了同步节点, 即后续任务的执行需要所有前置任务的完成.

$Rule_3$: 为表示任务的并发关系, 定义了分发节点, 即后续任务同时开始执行.

$Rule_4$: TDG-RAP 中的边表示任务间的顺序关系, 定义为变迁, 连接前后任务的结束库所与开始库所.

$Rule_5$: 自环是周期任务的描述方式, 需要为该任务定义 $(P \leftrightarrow T)$ 型结构, 在一次周期到达后, 任务被激活, 并重新获得发生权, 开始新一轮计时.

2.3 TDG-RAP 到 Pri-TPN 的转换算法

由 TDG-RAP 到 Pri-TPN 的转换基于表 1 中给出的规则进行匹配. 首先遍历所有节点, 对任务节点的属性进行判定, 分配到的计算资源以及是否使用锁和锁的顺序, 这决定了 $Rule_1$ 为每个任务创建的模型大小, 在转换过程中, 将请求处理器资源和释放处理器资源的变迁与相应的处理器资源对应的库所连接, 加锁和解锁变迁与锁资源对应的库所连接; 对于分发和同步节点, 将节点转换为相应的变迁. 然后, 遍历所有的边, 当源节点和目标节点相同时, 此边为自环边, 对这种边转换为周期性任务模型; 当源节点和目标节点不同时, 分别为边构造变迁. 此时, 得到了不带优先级的时间 Petri 网模型, 基于此模型对抢占关系进行绑定, 在相同处理器资源上的高优先级任务可抢占低优先级任务, 为保证抢占关系的完整性, 我们对优先级进行排列后从低优先级任务开始依次绑定, 同时, 在绑定过程中, 需要判断高优先级任务和低优先级任务是否使用锁, 即对于高优先级任务来说, 抢占模型需要 $Rule_1$ 中建立的模型相同, 对于低优先级任务来说, 可被抢占的位置也不相同. 算法 3 中对任务节点的标签函数映射的属性简

化表示, 如 $v.c$ 表示该任务节点分配的处理器, $e.l$ 表示任务周期.

算法 3. TDG-RAP 到 Pri-TPN 的转换算法.

输入: 任务依赖图 $G = \langle V, E, A, Lock, C, Pr, l \rangle$;

输出: 优先级时间 Petri 网 $Z = (P, T_1, T_2, F, M_0, SI, Pri)$.

```

Z = ∅;
for 每一个  $v \in V$  do
  if  $v \in V_{wait}$  then
    Add( $t \in T_1$ ) → Z; //Rule2
  else if  $v \in V_{dist}$  then
    Add( $t \in T_1$ ) → Z; //Rule3
  else
    Add( $P, T, F, SI = v.I, Pri = v.Pr$ ) → Z; //Rule1
  endif
endfor
for 每一个  $e \in E$  do
  if  $e$  是自环 then
    Add( $P, T, F, SI = e.l, Pri = v.Pr$ ) → Z; //Rule4
  else
    Add( $T, F$ ) → Z; //Rule5
  endif
endfor
//为处理器资源和锁资源创建库所
Add( $P$ ) → Z;
//为任务创建优先级绑定, 所有任务按照处理器分组并按优先级排列
for 每一个  $c \in C$  do
  for 每一个  $(v_1, v_2) = \{v \mid v \in V_{task} \wedge v.c = c\}$  do
    if  $(\exists (v_1.Pr < v_2.Pr))$  then
      if  $v_1.lock$  非空 then
        for 每一个  $lock \in Lock$  do
          if  $lock \in Lock_{sp}$  then
            Add( $P, T, F, SI = v.I$ ) → Z; // 图 1
            /*在可被抢占的任务中, 除获得处理器变迁外, 其余变迁为可挂起变迁*/
            Change( $\forall t \in Rule_1.T \wedge SI(t) \neq [0, 0] \rightarrow t \in T_2$ )
            break;
          else
            Add( $P, T, F, SI = v.I, Pri = v.Pr$ ) → Z; // 图 2
            Change( $\forall t \in Rule_1.T \wedge SI(t) \neq [0, 0] \rightarrow t \in T_2$ ) // 同上
          endif
        endfor
      endif
    endfor
  endif
endfor

```

```

endif
endfor
endif
return Z;

```

2.4 算法正确性和复杂性分析

算法的正确性分为两个部分来说明. 第 1 部分是 TDG-RAP 到 Petri 网的基本转换关系, 其中不包含高优先级抢占低优先级任务的关系 (完成此步后构建的 Petri 网可表示先进先出的调度关系); 第 2 部分是在 Petri 网的基础上描述任务间的抢占关系, 即抢占式调度机制.

(1) TDG-RAP 到 Petri 网仅需转换节点和边, 根据节点和边的前驱后继关系分类将 TDG-RAP 表示为 $G = (V_{\text{task}} \cup V_{\text{wait}} \cup V_{\text{dist}}, E_{\text{task-wait}} \cup E_{\text{task-dist}} \cup E_{\text{task-task}} \cup E_{\text{wait-task}} \cup E_{\text{dist-task}})$, 其中 $E_{\text{task-wait}}$ 表示前驱节点为任务节点, 后继节点为同步节点的边, 其他边类似. 对于 3 种节点的转换关系在表 1 中给出, 简化为 $V_{\text{task}} = (P \rightarrow T \rightarrow P \rightarrow T \rightarrow P)$ 、 $V_{\text{wait}} = T$ 和 $V_{\text{dist}} = T$, 节点的转换不改变任务的依赖关系; 在对边的转换中, 需要区分不同边连接的前驱节点和后继节点, 如 $E_{\text{task-task}}$ 是连接两个任务节点的边, 将其转换为 $E_{\text{task1-task2}} = (P_{\text{task1.end}} \rightarrow T \rightarrow P_{\text{task2.start}})$, 新增加一个变迁连接前驱任务的结束库所 ($P_{\text{task1.end}}$) 和后继任务的开始库所 ($P_{\text{task2.start}}$), 消耗前驱任务的结束库所中的托肯, 才会驱动变迁的发生 (后继任务的执行), 即在后继任务的开始库所中增加托肯; $E_{\text{task-wait}}$ 和 $E_{\text{wait-task}}$ 表示同步节点的转换, 对于此类边, 不需要额外的增加变迁, 而是直接与同步节点对应的变迁连接, 即 $E_{\text{task-wait}} : \forall v \in \{V_{\text{task}} \cap \text{prev}(V_{\text{wait}})\} \Rightarrow (P_{v.end} \rightarrow T)$ 和 $E_{\text{wait-task}} : \forall v \in \{V_{\text{task}} \cap \text{succ}(V_{\text{wait}})\} \Rightarrow (T \rightarrow P_{v.start})$, 其中, $\text{prev}(V)$ 和 $\text{succ}(V)$ 分别表示节点的前驱和后继. 当前驱任务的结束库所中全部存在托肯时 (即前驱任务全部结束), 才会触发同步节点表示的变迁, 从而驱动后继任务. 分发节点的转换同理. 对于同步和分发节点的转换可以发现, 在转换完成后的 Petri 网中并没有改变任务的发生语义, 即任务的依赖关系.

(2) 根据 TDG-RAP 标签的优先级属性, 在 Petri 网中构建任务间的抢占关系, 存在任务 v_1 的优先级高于任务 v_2 , 表示为 $v_1.Pr > v_2.Pr$, $v_1 = (P_{1.start} \rightarrow T \rightarrow P_1 \rightarrow T \rightarrow P_{1.end})$, $v_2 = (P_{2.start} \rightarrow T \rightarrow P_2 \rightarrow T \rightarrow P_{2.end})$, 为高优先级任务 v_1 创建抢占任务 v_2 的托肯流动路径, 即 $v_1 = ((P_{1.start}, P_2) \rightarrow T \rightarrow P_{12} \rightarrow T \rightarrow (P_2, P_{1.end}))$, 当任务 v_2 中库所 P_2 存在托肯时, 表明 v_2 正在执行, v_1 才会触发抢占变迁, 若有更高优先级的任务 v_3 时, 需要额外创建 3 条抢占路径, 即抢占库所 (P_1, P_2, P_{12}).

算法复杂度主要考虑 3 个部分.

(1) 首先是 TDG-RAP 转换为等价的 Pri-TPN 模型. 本文对于任务的 Petri 网模型是固定的, 因为中断和优先级发生的抢占是影响 Pri-TPN 标识数的不确定部分, 但由于优先级事先给定, 所以假设任务数为 X , 优先级最多有 $X - 1$ 个, 最大的标识数不会超过 $\sum_{i=1}^{n-1} Y_i$, 其中 Y_i 是任务转换为 Pri-TPN 的库所数.

(2) 然后, 基于转换后的 Pri-TPN 模型生成它的状态类图. 由于本文定义的 Pri-TPN 是安全的, 因此状态类图中的最大标识数量不会超过 $2^{|P|}$, 但是本文考虑每个变迁的静态发生区间是不确定的, 其时钟类也是不确定的, 本文考虑在单周期任务时间内, 假设每个区间都是整数区间, 一个标识下最大的状态迁移个数是 $\prod_{t \in T_1 \cup T_2} (\uparrow SI(t) - \downarrow SI(t))$, 所以最大的状态类图不会超过 $2^{|P|+1} \prod_{t \in T_1 \cup T_2} (\uparrow SI(t) - \downarrow SI(t))$, 同时, 产生一个新的状态类的算法复杂度为 $O(1)$. 因此, 由优先级时间 Petri 网生成状态类图的算法复杂度为 $O\left(2^{|P|+1} \prod_{t \in T_1 \cup T_2} (\uparrow SI(t) - \downarrow SI(t))\right)$.

(3) 最后, 遍历生成的状态类图, 找到进入任务的起始节点, 退出任务的结束节点. 从任务的起始节点进行深度优先遍历, 找到最长的时间权重的路径, 即为最坏执行路径, 算法的时间复杂度 $O\left(\prod_{t \in T_1 \cup T_2} (\uparrow SI(t) - \downarrow SI(t)) \cdot m\right)$, 其中 m 为状态类图的最大深度.

3 工具与实例研究

本节通过图 4 所示的案例, 来验证 Pri-TPN 模型和转换算法的正确性和有效性. 图 4 描述的一个实时多核多任务系统有 6 个任务 (即 $a_1, a_2, a_3, a_4, a_5, a_6$) 和两个处理器 (即 c_0, c_1), 其中, 任务 a_1, a_2 和 a_6 共用处理器 c_1 , 任务 a_3, a_4 和 a_5 共用处理器 c_0 , 初始任务 a_1 和 a_4 是两个周期性任务. 任务之间存在多种控制依赖关系, 即一个任务的执行结束可驱动多个任务或者多个任务全部结束才能驱动下一个任务的启动. 任务 a_1 获得处理器 c_1 后执行 [3, 5] ms, 任务 a_4 获得处理器 c_0 后, 需要继续获得锁资源 $lock_1$, 在临界区内执行 [10, 12] ms, 释放锁资源后, 继续执行 [2, 4] ms. 任务 a_2 在任务 a_1 完成且获得处理器 c_1 后执行 [1, 5] ms, 任务 a_4 执行完毕后同时驱动 (节点 v_6) 任务 a_5 和任务 a_6 , 其中任务 a_5 获得处理器 c_0 后, 继续获得锁资源 $lock_2$, 在临界区内执行 [4, 6] ms, 释放 $lock_2$ 后继续执行 [2, 3] ms. 等待任务 a_2 和 a_5 全部执行完毕 (节点 v_3) 后驱动任务 a_3 , 同时, 任务 a_5 执行完毕驱动任务 a_6 . 任务 a_3 获得处理器 c_0 后, 继续获得锁资源 $lock_1$ 和 $lock_2$, 需要注意的是, 本文的任务模型中的锁资源 $lock_1$ 和 $lock_2$ 是连续获得, 在 $lock_2$ 锁定的临界区内执行 [3, 4] ms, 在 $lock_1$ 锁定的临界区内执行 [2, 3] ms, 依次释放 $lock_2$ 和 $lock_1$ 后继续执行 [2, 4] ms. 任务 a_6 获得处理器 c_1 后执行 [1, 2] ms. 在利用 Pri-TPN 分析时, 考虑抢占式调度的需求. 例如: 包含了 4 种抢占式调度情况, 任务 a_6 可被任务 a_1 和 a_2 抢占, 任务 a_1 和已经抢占任务 a_6 所占用处理器的 a_1 可被任务 a_2 抢占; 任务 a_3, a_4 和 a_5 的抢占处理器的情况类似. 注意: 在转换完成的 Pri-TPN 中, 如果当一个低优先级任务持有自旋锁后, 即使有一个高优先级任务准备好运行, 它也必须等待低优先级任务释放锁, 在图 2 中表现为在获得自选锁之后的库所 (p_{17}) 不存在使能高优先级执行变迁 (t_9) 的弧.

图 5 是我们开发的工具的输入界面.

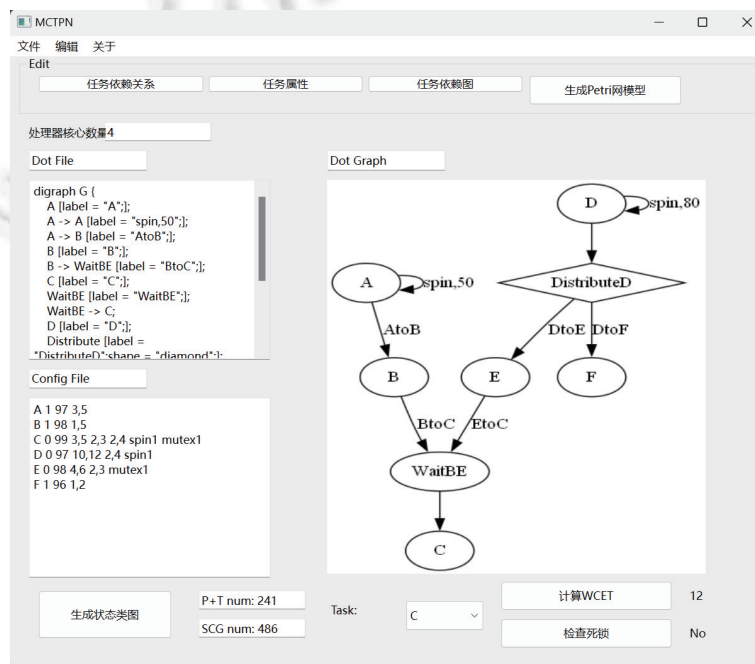


图 5 工具界面展示

在图 5 左上方输入图 4 的任务依赖模型, 在左下方给出节点对应任务的描述, 依次是任务的名称、分配的处理器、任务的执行时间、优先级以及任务使用的锁类型及其顺序. 图 5 的右半部会输出这个 TDG-RAP 的图的形式. 对于这样一个多核多任务实时系统, 我们可以分析它的最坏执行时间来验证任务在多核系统上的正确性. 如图 5 下方所示计算出任务 C 的最坏执行时间为 12 ms, 并且该系统不存在使用锁而造成死锁的情况. 图 6 展示了由这个 TDG-RAP 自动转化成的 Pri-TPN.

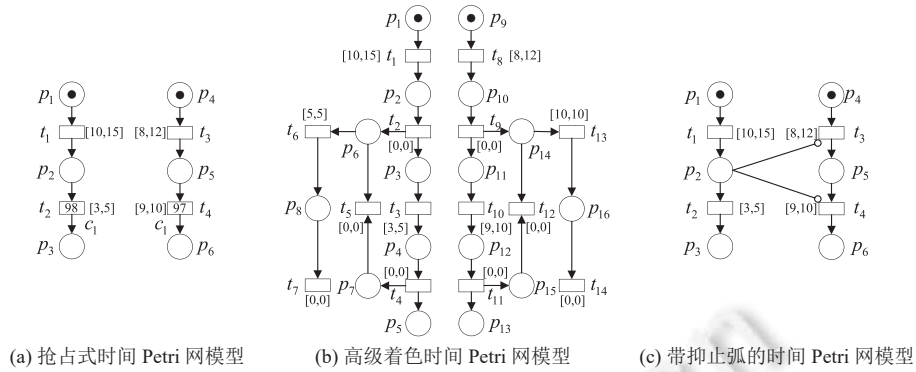


图 6 图 1 示例的其他扩展时间 Petri 网模型

现有研究中对于网的定义和发生规则不同, 所以生成的 Petri 网规模也不相同, 同时暂未发现由 DAG 到 Petri 网的自动转换的相关工具, 故为了展示我们方法与工具的有效性, 本文对图 4 的例子进行了修改 (如表 2 第 1 列所示) 用作测试用例, 统计了所生成网的规模、自动转换时间、状态类的个数以及生成状态类的时间. 结果表明即使当网的规模变大或者状态类图变大时, 我们的工具也能很快地给出输出结果.

表 2 测试用例上的一些实验结果

| 序号 | TDG-RAP | Pri-TPN | | | | |
|----|-----------------------------|-----------|-------|----------|------|--------------|
| | | $ P + T $ | $ F $ | 转换时间 (s) | 状态类数 | 生成状态类图时间 (s) |
| 1 | 初始任务分配 | 241 | 374 | 0.0026 | 486 | 0.06111 |
| 2 | 所有任务分配在不同处理器上 | 61 | 74 | 0.0016 | 468 | 0.2489 |
| 3 | 所有任务分配在同一处理器上 | 658 | 1064 | 0.0358 | 398 | 0.1023 |
| 4 | 所有任务分配不同优先级 | 143 | 218 | 0.0052 | 483 | 0.0568 |
| 5 | 所有任务分配同一优先级 | 61 | 74 | 0.0016 | 503 | 0.0213 |
| 6 | 更改任务 a_2 时间为 [7, 9] | 241 | 374 | 0.0063 | 541 | 0.1977 |
| 7 | 更改任务 a_5 时间为 [2, 3, 4, 6] | 241 | 374 | 0.0035 | 544 | 0.0681 |

简单介绍这几个测试用例. 初始任务分配按照图 4 给定的任务依赖和任务标签转换为 Pri-TPN, 给出它的状态类图, 图 4 中共有 6 个任务, 为方便后续测试, 将实验中处理器资源的个数统一设置为 6. 在任务依赖关系不变的情况下: (1) 改变任务的所使用的处理器资源 (实验 3、实验 4); (2) 改变任务的优先级 (实验 5、实验 6), 其中所有任务不同优先级分配为 ($a_1: 93, a_2: 94, a_3: 95, a_4: 96, a_5: 97, a_6: 98$); (3) 改变某一任务的执行时间 (实验 7、实验 8), 其中改变任务的执行时间不会造成 Petri 网规模的变化, 因此实验生成的 Pri-TPN 库所和变迁数量不变, 但执行的时间域变化导致了状态类个数的不同.

4 相关工作

目前有 5 种时间 Petri 网的扩展形式可模拟实时系统的带有优先级的抢占式调度机制, 即点区间优先级时间 Petri 网^[13]、调度扩展时间 Petri 网^[17]、抢占式时间 Petri 网^[18,19]、带有抑制弧的时间 Petri 网^[20]和高级着色时间 Petri 网^[27]. 图 7(a) 模拟了前文图 1 示例的一个抢占式时间 Petri 网. 抢占式时间 Petri 网是在变迁上添加了资源和优先级, 在变迁获得发生权后, 如果存在其他更高优先级的变迁也使用相同资源, 那么低优先级变迁将被挂起, 当高优先级变迁发生后, 低优先级变迁从挂起状态恢复, 重新计时. 调度扩展时间 Petri 网与之类似, 在库所上添加额外的属性来增强建模能力. 带有抑制弧的时间 Petri 网扩展了弧的形式, 增加了抑制弧 (弧的头部使用圆圈表示), 当弧连接的前驱库所中存在托肯时, 其指向的变迁不会被使能, 即该变迁被挂起, 图 7(c) 使用带抑制弧的时间 Petri 网模拟了图 1 示例. 以上 3 种扩展的时间 Petri 网由于对资源的隐式描述, 在建模分析与资源相关的时间分析上存在限制.

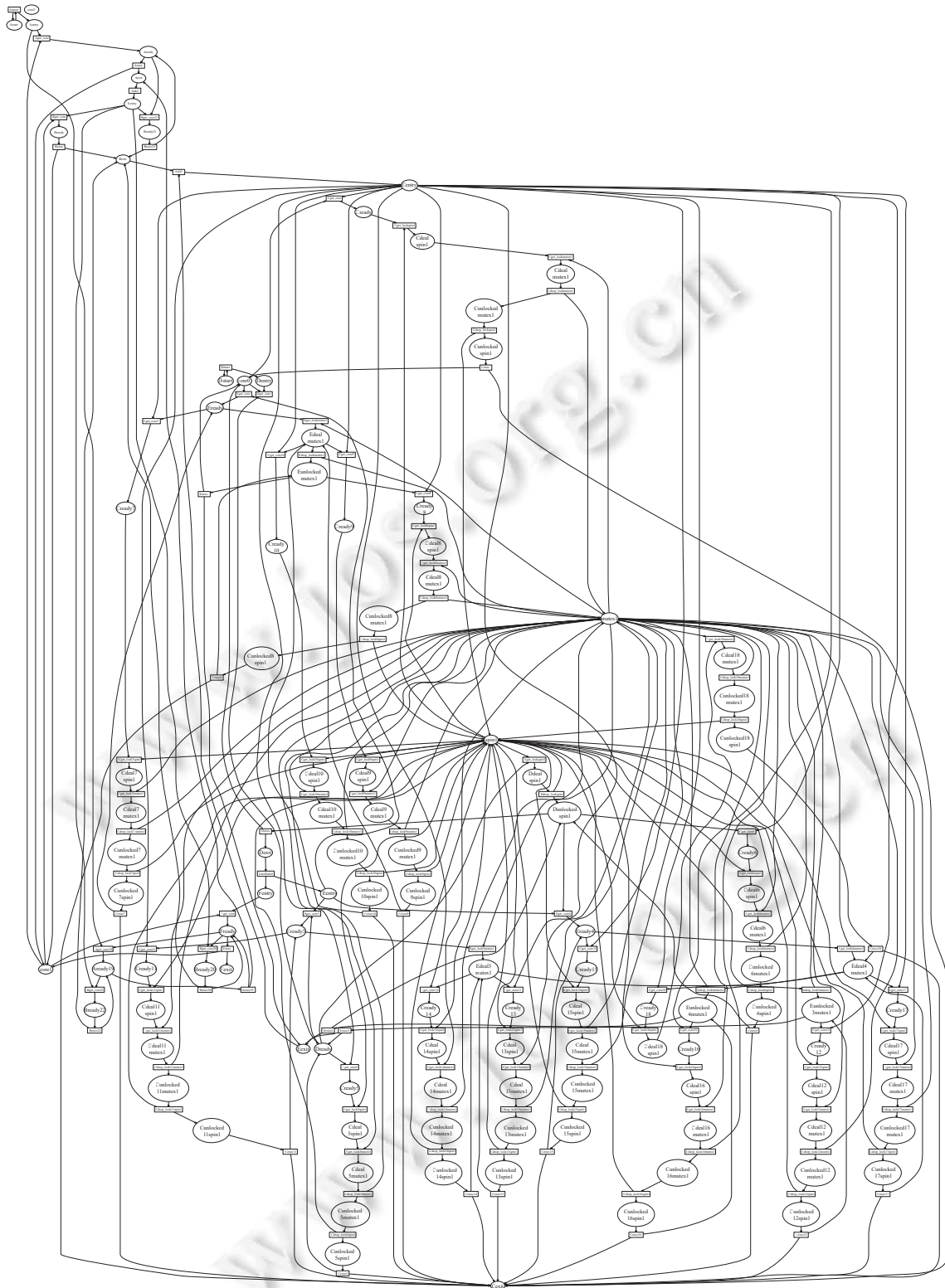


图 7 由图 4 中的 TDG-RAP 生成的 Pri-TPN

本文主要讨论点区间优先级时间 Petri 网和高级着色时间 Petri 网. 点区间优先级时间 Petri 网可以显式刻画资源的占用和释放过程, 可以验证与处理器和锁资源相关的时间属性. 点区间优先级时间 Petri 网是本文所定义的优先级时间 Petri 网的一类子网, 变迁发生的最短时间和最长时间相等, 这种固定时间发生变迁的时间 Petri 网导致一些潜在任务的控制流路径不会发生, 减少了所生成的状态图中的状态数, 提高了对时序逻辑公式对于实时系统时间约束的验证效率, 但如前面所述建模能力不如 Pri-TPN. 高级着色时间 Petri 网使用颜色和高级功能来扩展 Petri 网的时间, 包括定时多重启用转换和顺序伪代码, 并将这种模型用于建模多核自旋锁机制. 图 7(b) 用高级着色时间 Petri 网模拟前文图 1 中示例, 着色网使用额外的变量来定义复杂的转换功能, 这有效地缩减了 Petri 网模型的规模, 但增加了自动转换为此类模型和生成状态类图算法的复杂性, 随着描述任务数量的增长, 建模过程繁琐, 不利于工程人员使用.

本文定义的 Pri-TPN 相对于点区间优先级时间 Petri 网扩展了变迁的表达, 变迁的发生时间由点区间扩展为区间. 相对于高级着色时间 Petri 网, Pri-TPN 加入了优先级, 显式表达了资源抢占机制. 另外, 由 DAG 描述的任务依赖图可以有效地表达任务之间的依赖关系, 因此常用于实时系统的任务描述建模. 而 TDG-RAP 加入标签函数以表达任务的各种属性, 如优先级、处理器和锁的使用情况, 在此基础上本文给出了 TDG-RAP 到 Pri-TPN 的转换规则和算法, 最后基于 Pri-TPN 计算 TDG-RAP 描述系统的 WCET 和检查死锁.

需要注意的是, 本文定义的 Pri-TPN 没有解决点区间优先级时间 Petri 网的状态数爆炸问题, 主要原因在于变迁发生时间的扩展和对锁资源的建模增加了状态个数, 但相较于点区间优先级时间 Petri 网来说, Pri-TPN 不需要手动对 Petri 网建模, 更精确地表达了系统行为和任务关系, 因此才会有这样的结果.

5 结论与展望

对于多核环境下的嵌入式实时多任务系统, 本文在给定的任务依赖关系和任务分配方案下, 转换为优先级时间 Petri 网, 然后利用这一形式化模型进行最坏执行时间计算和死锁检测, 而系统设计人员只需在工具中输入其所设计系统的任务依赖图以及任务分配、运行时间、优先级等信息, 为系统设计人员提供支撑. 下一步的工作将从 3 个方面展开: (1) 提供更多任务依赖关系, 以及相应的 Petri 网模型的自动转换; (2) 引入概率自动机模型, 表示 Cache 对于实时系统性能的影响; (3) 运用启发式算法, 在给定处理器核心数量的情况下, 自动寻找最佳任务分配方案; (4) 对航天嵌入式系统的实例进行分析验证.

References:

- [1] Cousot P, Cousot R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. of the 4th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages. Los Angeles: ACM, 1977. 238–252. [doi: 10.1145/512950.512973]
- [2] Li YTS, Malik S. Performance analysis of embedded software using implicit path enumeration. In: Proc. of the 32nd Design Automation Conf. San Francisco: IEEE, 1995. 456–461. [doi: 10.1109/DAC.1995.249991]
- [3] Rochange C. Parallel real-time tasks, as viewed by WCET analysis and task scheduling approaches. In: Proc. of the 16th Int'l Workshop on Worst-case Execution Time Analysis (WCET 2016). Open Access Series in Informatics (OASIS), Vol. 55. 11:1–11:11. [doi: 10.4230/OASIS.WCET.2016.11]
- [4] Melani A, Bertogna M, Bonifaci V, Marchetti-Spaccamela A, Buttazzo GC. Response-time analysis of conditional DAG tasks in multiprocessor systems. In: Proc. of the 27th Euromicro Conf. on Real-time Systems. Lund: IEEE, 2015. 211–221. [doi: 10.1109/ECRTS.2015.26]
- [5] Wang J, Zhan NJ, Feng XY, Liu ZY. 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]
- [6] Cui J, Duan ZH, Tian C, Zhang N. 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]
- [7] Clarke Jr EM, Grumberg O, Kroening D, Peled D, Veith H. Model Checking. 2nd ed., Cambridge: MIT Press, 2018.
- [8] Wang C, Feng XJ, Li X, Zhou XH, Chen P. Colored Petri net model with automatic parallelization on real-time multicore architectures. Journal of Systems Architecture, 2014, 60(3): 293–304. [doi: 10.1016/j.sysarc.2013.08.016]

- [9] Huang B, Zhou MC, Lu XS, Abusorrah A. Scheduling of resource allocation systems with timed Petri nets: A survey. *ACM Computing Surveys*, 2023, 55(11): 230. [doi: [10.1145/3570326](https://doi.org/10.1145/3570326)]
- [10] Cui J, Duan ZH, Tian C, Du HW. A novel approach to modeling and verifying real-time systems for high reliability. *IEEE Trans. on Reliability*, 2018, 67(2): 481–493. [doi: [10.1109/TR.2018.2806349](https://doi.org/10.1109/TR.2018.2806349)]
- [11] Zhong WJ, Zhou JT, Sun T. Concurrent software fine-coarse-grained automatic modelling by coloured Petri nets for model checking. *IET Software*, 2023, 17(1): 55–75. [doi: [10.1049/sfw2.12084](https://doi.org/10.1049/sfw2.12084)]
- [12] 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).
- [13] 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 with English abstract). <http://www.jos.org.cn/1000-9825/6607.htm> [doi: [10.13328/j.cnki.jos.006607](https://doi.org/10.13328/j.cnki.jos.006607)]
- [14] 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): 11. [doi: [10.1145/3185502](https://doi.org/10.1145/3185502)]
- [15] Liu GJ. Primary Unfolding of Petri Nets: A Model Checking Method for Concurrent Systems. Beijing: Science Press, 2020 (in Chinese).
- [16] 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*. Yogyakarta: IEEE, 2017. 1–6. [doi: [10.1109/EECSI.2017.8239121](https://doi.org/10.1109/EECSI.2017.8239121)]
- [17] Lime D, Roux OH. Expressiveness and analysis of scheduling extended time Petri nets. *IFAC Proc. Volumes*, 2003, 36(13): 189–197. [doi: [10.1016/S1474-6670\(17\)32483-7](https://doi.org/10.1016/S1474-6670(17)32483-7)]
- [18] Bucci G, Fedeli A, Sassoli L, Vicario E. Modeling flexible real time systems with preemptive time Petri nets. In: *Proc. of the 15th Euromicro Conf. on Real-time Systems*. Porto: IEEE, 2003. 279–286. [doi: [10.1109/EMRTS.2003.1212753](https://doi.org/10.1109/EMRTS.2003.1212753)]
- [19] Bucci G, Fedeli A, Sassoli L, Vicario E. Timed state space analysis of real-time preemptive systems. *IEEE Trans. on Software Engineering*, 2004, 30(2): 97–111. [doi: [10.1109/TSE.2004.1265815](https://doi.org/10.1109/TSE.2004.1265815)]
- [20] Roux OH, Lime D. Time Petri nets with inhibitor hyperarcs. Formal semantics and state space computation. In: *Proc. of the 2004 Int'l Conf. on Application and Theory of Petri Nets*. Bologna: Springer, 2004. 371–390. [doi: [10.1007/978-3-540-27793-4_21](https://doi.org/10.1007/978-3-540-27793-4_21)]
- [21] Berthomieu B, Lime D, Roux OH, Vernadat F. Reachability problems and abstract state spaces for time Petri nets with stopwatches. *Discrete Event Dynamic Systems*, 2007, 17(2): 133–158. [doi: [10.1007/s10626-006-0011-y](https://doi.org/10.1007/s10626-006-0011-y)]
- [22] Liu F, Zhang HM. A class of extended time Petri nets for modeling and simulation of discrete event systems. *Simulation*, 2018, 94(8): 753–762. [doi: [10.1177/0037549717742716](https://doi.org/10.1177/0037549717742716)]
- [23] Lu FM, Tao RR, Du YY, Zeng QT, Bao YX. Deadlock detection-oriented unfolding of unbounded Petri nets. *Information Sciences*, 2019, 497: 1–22. [doi: [10.1016/j.ins.2019.05.021](https://doi.org/10.1016/j.ins.2019.05.021)]
- [24] 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.
- [25] van der Aalst WMP. The application of Petri nets to workflow management. *Journal of Circuits, Systems and Computers*, 1998, 8(1): 21–66. [doi: [10.1142/S0218126698000043](https://doi.org/10.1142/S0218126698000043)]
- [26] Liu GJ. *Petri Nets: Theoretical Models and Analysis Methods for Concurrent Systems*. Singapore: Springer, 2022. [doi: [10.1007/978-981-19-6309-4](https://doi.org/10.1007/978-981-19-6309-4)]
- [27] Haur I, Béchenne JL, Roux OH. High-level colored time Petri nets for true concurrency modeling in real-time software. In: *Proc. of the 8th Int'l Conf. on Control, Decision and Information Technologies (CoDIT)*. Istanbul: IEEE, 2022. 21–26. [doi: [10.1109/CoDIT55151.2022.9803922](https://doi.org/10.1109/CoDIT55151.2022.9803922)]

附中文参考文献:

- [5] 王戟, 詹乃军, 冯新宇, 刘志明. 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](https://doi.org/10.13328/j.cnki.jos.005652)]
- [6] 崔进, 段振华, 田聪, 张南. 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](https://doi.org/10.13328/j.cnki.jos.005472)]
- [12] 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.
- [13] 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> [doi: [10.13328/j.cnki.jos.006607](https://doi.org/10.13328/j.cnki.jos.006607)]
- [15] 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.



张凯文(1995—), 男, 博士生, 主要研究领域为 Petri 网, 形式化方法, 程序分析.



关健(1986—), 女, 高级工程师, 主要研究领域为人工智能软件, 嵌入式系统.



刘关俊(1978—), 男, 博士, 教授, 博士生导师, CCF 高级会员, 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, 工作流程系统, 无人机协同系统.



解毅(1994—), 男, 工程师, 主要研究领域为嵌入式系统.



孙彦韬(1997—), 男, 博士生, CCF 学生会会员, 主要研究领域为嵌入式系统, 高性能计算.



顾斌(1968—), 男, 博士, 研究员, 博士生导师, CCF 高级会员, 主要研究领域为可信软件, 计算机控制, 嵌入式软件.



李晓锋(1982—), 男, 研究员, CCF 专业会员, 主要研究领域为可信软件, 软件自适应, 智能化软件工程.