软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(10):4533−4554 [doi: 10.13328/j.cnki.jos.006969]

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

抢占式调度问题的 PPTA 模型与验证方法*

左正康[1,2], 赵 帅[1], 王昌晶[1,2], 谢武平[3], 黄 箐[2]

[1](江西师范大学 数字产业学院, 江西 上饶 334006)

[2](江西师范大学 计算机信息工程学院, 江西 南昌 330022)

[3](江西师范大学 网络化支撑软件国家科技合作基地, 江西 南昌 330022)

通信作者: 谢武平, E-mail: wupingxie@jxnu.edu.cn

摘 要: 优先级用于解决诸如在资源共享和安全设计等方面的冲突, 已经成为实时系统设计中不可或缺的一部分. 对于引入优先级的实时系统, 每个任务都会被分配优先级, 这就导致低优先级的任务在运行时可能会被高优先级的任务抢占资源, 进而给实时系统带来抢占式调度问题. 现有研究, 缺乏一种可以直观表示任务的优先级以及任务之间的依赖关系的建模及自动验证方法. 为此, 提出抢占式优先级时间自动机 (PPTA) 并引入抢占式优先级时间自动机网络 (PPTAN). 首先, 通过在时间自动机上添加变迁的优先级来表示任务的优先级, 再利用变迁将具有依赖关系的任务相关联, 从而可以利用 PPTA 建模带有优先级的实时任务. 在时间自动机上添加阻塞位置, 进而利用 PPTAN 建模优先级抢占式调度问题. 其次, 提出基于模型的转换方法, 将抢占式优先级时间自动机映射到自动验证工具 UPPAAL 中. 最后, 通过建模多核多任务实时系统实例并与其他模型进行对比, 说明所提模型不仅适用于建模优先级抢占式调度问题并可对其进行准确验证分析.

关键词: 优先级抢占式调度; 抢占式优先级时间自动机; 多核多任务实时系统; UPPAAL

中图法分类号: TP311

中文引用格式: 左正康, 赵帅, 王昌晶, 谢武平, 黄箐. 抢占式调度问题的PPTA模型与验证方法. 软件学报, 2024, 35(10): 4533–4554. http://www.jos.org.cn/1000-9825/6969.htm

英文引用格式: Zuo ZK, Zhao S, Wang CJ, Xie WP, Huang Q. PPTA Model and Verification Method for Preemptive Scheduling Problem. Ruan Jian Xue Bao/Journal of Software, 2024, 35(10): 4533–4554 (in Chinese). http://www.jos.org.cn/1000-9825/6969.htm

PPTA Model and Verification Method for Preemptive Scheduling Problem

ZUO Zheng-Kang[1,2], ZHAO Shuai[1], WANG Chang-Jing[1,2], XIE Wu-Ping[3], HUANG Qing[2]

[1](School of Digital Industry, Jiangxi Normal University, Shangrao 334006, China)

[2](School of Computer and Information Engineering, Jiangxi Normal University, Nanchang 330022, China)

[3](National-level International S&T Cooperation Base of Networked Supporting Software, Jiangxi Normal University, Nanchang 330022, China)

Abstract: As an essential component of real-time system design, priority is utilized to resolve conflicts in resource sharing and design for safety. For real-time systems that introduce priorities, each task is assigned a priority, which leads to the possibility of low-priority tasks being preempted by high-priority tasks at runtime, thus creating a preemptive scheduling problem for real-time systems. Existing research on this problem lacks a modeling and automatic verification method that can visually represent the priority of tasks and the dependencies between tasks. To this end, a preemptive priority timed automata (PPTA) is proposed and a preemptive priority timed automata network (PPTAN) is introduced. First, the priority of a task is represented by adding the priority of migration to the timed automata, and then the migration is adopted to correlate tasks with dependencies so that PPTA can be applied to model real-time tasks with priority. The blocking

* 基金项目: 国家自然科学基金 (61862033, 62262031); 江西省教育厅科技项目 (GJJ210307, GJJ210334)
收稿时间: 2022-12-20; 修改时间: 2023-03-14; 采用时间: 2023-05-16; jos 在线出版时间: 2023-09-06
CNKI 网络首发时间: 2023-09-07

position is also added to the timed automata, so PPTAN can be used to model the priority preemptive scheduling problem. Second, a model-based transformation method is proposed to map the PPTA to the automatic verification tool UPPAAL. Finally, by modeling an example of a multi-core multi-task real-time system and comparing it with other models, it is shown that this model is not only suitable for modeling the priority preemptive scheduling problem but also for accurately verifying and analyzing it.

Key words: preemptive priority scheduling; preemptive priority timed automata (PPTA); multi-core multi-task real-time system; UPPAAL

实时系统被广泛应用于汽车、飞机等安全性攸关的系统中, 实时系统的正确性对整个系统的安全运行有至关重要的影响. 为此, 我们必须保证在任何情况下实时系统都必须满足时间约束[1]. 可调度性分析就是用来验证系统中任务是否满足规定的时间约束的重要方法. 优先级过去常用于解决资源共享和安全设计中的冲突问题, 现在它已成为实时系统设计中不可缺少的部分. 优先级抢占式调度通常被应用于许多实时软件系统, 如基于自动存储的操作系统[2]、时间敏感网络[3]、离散事件系统[4]、信息物理融合系统[5]、多核多任务实时系统[6]等. 在优先级抢占式调度中, 具有较高优先级且即将运行的任务 T_2 可以中断和阻止具有较低优先级的当前正在运行的任务 T_1, 使其进入休眠状态. 更高优先级的任务 T_2 将获得处理器并运行到完成, 之后被阻塞的任务 T_1 将从休眠状态中唤醒并恢复执行. 显然, 上述抢占可能会导致低优先级任务错过其截止时间, 因此需要对优先级抢占式调度进行分析, 及时发现是否有任务错过截止时间并进行有效修正.

形式化方法是基于严格数学基础, 对计算机硬件和软件进行描述、开发和验证的技术. 其数学基础建立在形式语言、语义和推理证明三位一体的形式逻辑系统之上[7]. 模型检测作为形式化验证的一种主要技术, 可以对系统属性实现完全自动化的验证; 同时, 模型检测方法可以得到更准确的分析结果, 提高实时系统的可靠性. 一些基于时间 Petri 网[8]和时间自动机 (timed automata, TA)[9]的模型已经被提出, 并用来对优先级抢占式调度问题进行建模与验证. 文献 [6,10,11−14] 使用扩展时间 Petri 网对实时系统进行建模和时间属性的验证. 其中, 文献 [6,12] 开发了相应的模型检测工具进行验证. 然而, 文献 [6] 中的模型检测工具只支持文本型文件输入建模模型和性质公式, 而不像 UPPAAL 一样可以在工具内部直接编辑 TA 模型和公式. 虽然文献 [12] 的模型检测工具支持对模型的可视化编辑, 但无法自动验证描述性质的公式, 需要通过手动分析生成的路径来判断性质是否成立. 文献 [10,11,13,14] 在最终验证时, 是针对模型的可达性提出分析方法, 因此该部分工作在验证属性时无自动验证工具的支持. 此外, Berthomieu 等人[10]提出了优先级时间 Petri 网建模优先级抢占式调度问题, 当低优先级任务再次获得资源时, 并不是在中断的地方继续执行而是从头开始, 这显然不符合一些实时系统的优先级抢占式调度机制. 对于何雷峰等人[6]提出的点区间优先级时间 Petri 网在建模优先级抢占式调度问题时未考虑任务的截止时间, 可能会导致系统出现不可调度问题. 文献 [15] 在利用 TA 建模优先级抢占式调度问题时, 需要在全局声明中声明任务的优先级以及任务之间的依赖关系, 不能在 TA 上直观地来表示. 文献 [16] 提出的优先级时间自动机虽然能直观地显示任务的优先级, 但无法建模任务之间的依赖关系以及优先级抢占式调度问题. 此外, David 等人[15]提出的验证实时系统可调度性的时间自动机建模框架, 在验证优先级抢占式调度问题的相关属性时会出现验证结果不符预期的情况.

基于 TA 的模型能够同时表示实时系统的逻辑属性和时间属性, 通过系统建模, 将优先级抢占式调度问题转化为时间自动机的可达性问题. 因此, 本文针对实时系统引入优先级所带来的抢占式调度问题, 提出了抢占式优先级时间自动机 (preemptive priority timed automata, PPTA), 同时给出了转换方法, 将 PPTA 映射到 UPPAAL[17]中, 最后通过一个实例, 利用计算树逻辑 (computation tree logic, CTL)[18]描述相关属性, 在 UPPAAL 中验证, 来证明我们所提模型的准确性和有效性.

本文的主要贡献如下.

(1) 扩展了 TA, 为变迁添加优先级并增加阻塞位置, 提出了 PPTA 并引入了抢占式优先级时间自动机网络 (preemptive priority timed automata network, PPTAN). 相比 TA, 我们在利用 PPTA 对带有优先级的任务建模时, 通过迁移的优先级可以直观地表示任务的优先级. 相比优先级时间时自动机, 我们在建模任务之间的依赖关系时, 通过迁移将两个具有依赖关系的任务相关联, 可以直观表示任务的依赖关系. 同时由于添加了阻塞位置, 当低优先级的任务被抢占资源时, 会进入阻塞位置, 因此我们可以利用 PPTAN 来模拟优先级抢占式调度问题.

(2) 给出了 PPTA 转换为 TA 的转换规则与转换算法, 相比于利用时间 Petri 网验证优先级抢占式调度问题的相关属性, 我们不需要开发相关的验证工具, 只需要将 PPTA 模型映射到 UPPAAL 中, 输入属性描述, 实现对该问题的自动验证, 进而简化验证过程.

(3) 通过多核多任务实时系统案例, 利用 CTL 描述相关属性并在 UPPAAL 中验证, 从而确保本文所提模型可以对优先级抢占式调度问题进行准确建模与验证分析. 相比于文献 [6], 我们不仅可以验证有界活性属性, 还可以验证系统是否可以调度, 是否死锁等属性, 且可以直接利用已有的 UPPAAL 工具验证, 验证成本降低.

本文第 1 节介绍了一些基本知识, 包括时间自动机和验证工具 UPPAAL. 第 2 节介绍了我们提出的 PPTA. 第 3 节介绍了 CTL、转换规则、转换算法和验证属性. 第 4 节通过一个实例来说明我们模型的准确性和有效性. 第 5 节回顾了一些相关工作. 第 6 节对全文进行了总结并阐述我们下一步的工作.

1 预备知识

本节简单介绍 TA 的语法和语义以及基于 TA 模型的自动验证工具 UPPAAL.

1.1 时间自动机

语法: TA 用一个六元组 $A=(L,l_0,\Sigma,X,I,T)$ 来表示, 其中,

$L=\{l_0,l_1,l_2,\ldots,l_n\}$ 是包含 n 个位置的有穷集.

$l_0\in L$ 是初始位置.

$\Sigma=\{a,b,c,\ldots,n\}$ 是包含 n 个事件标号的有穷集.

$X=\{x_0,x_1,x_2,\ldots,x_n\}$ 是包含 n 个时钟变量的有穷集; 对于 X 中的所有时钟 x, 时间的流逝速度是相同的.

$I:L\to\phi(X)$ 是一个映射, 它为 L 中的每一个位置 l 指定 $\phi(X)$ 中的某一个时钟约束作为 l 的不变式. 时钟约束 $\varphi=x\sim c|\varphi_1\wedge\varphi_2|\text{true}$, 其中 x 是一个时钟, c 是一个非负整数, φ_1 和 φ_2 是时钟约束公式, $\sim\in\{<,>,\leqslant,\geqslant\}$.

$T\subseteq L\times\Sigma\times\phi(X)\times 2^X\times L$ 是一个迁移集合. 一个五元组 $<l,a,\varphi,\lambda,l'>\in T$ 表示一个事件标号为 a, 从位置 l 到 l' 的迁移. φ 是定义在时钟集 X 上的一个时钟约束函数, 在迁移发生时必须被满足. λ 表示迁移发生时被重置的所有时钟组成的集合, 且 $\lambda\subseteq X$. a,φ,λ 在必要时都是可以省略的.

语义: TA 的语义模型为迁移系统 $T_A=<S,s_0,\Sigma_{(a,\varphi,\lambda)},\to>$. 其中,

$S\subseteq L\times\mu$, μ 表示 X 上的时钟赋值集合. S 中的每一个状态由一个位置 l, 一个时钟赋值 μ 组成, 记为 (l,μ).

$s_0\subseteq(l_0,\mu_0)\in S$ 是初始状态.

$\Sigma_{(a,\varphi,\lambda)}=\{(a_0,\varphi_0,\lambda_0),(a_1,\varphi_1,\lambda_1),\ldots,(a_n,\varphi_n,\lambda_n)\}$ 是带有事件标号、迁移上时钟约束和迁移后时钟重置的有穷事件集.

$\to$是迁移关系, 有延迟迁移和离散迁移两种形式.

假设 $s=(l,\mu)$ 和 $s'=(l',\mu')$ 为 T_A 中的两个状态, 则 T_A 中存在两种状态迁移规则.

(1) 状态因时间的流逝而改变, 也称延迟迁移: $s\xrightarrow{\tau}s'$ 当且仅当状态 s' 从状态 s 经过 τ 个时间后到达, 即:

1) $l'=l$.

2) $\mu\vDash I(l)$.

3) $\mu'=\mu+\tau\vDash I(l)$.

(2) 状态因位置的转换而改变, 也称离散迁移: $s\xrightarrow{t}s'$ 表示存在一个迁移 $t=<l,\Sigma_{(a,\varphi,\lambda)},l'>$, 即:

1) $\mu\vDash\varphi(t)$.

2) $l\xrightarrow{t}l'$.

3) $\mu'=\mu[\lambda:=0]\vDash I(l')$.

1.2 UPPAAL

模型检测工具 UPPAAL 是由瑞典 Uppsala 大学和 Aalborg 大学于 1995 年联合开发的一种基于时间自动机的

模型检测工具[17], 可以有效地对实时系统建模和自动验证, 特别适合对实时系统的安全性和有界活性进行自动验证. 它的主要优点是其高效性和使用的方便性.

UPPAAL 主要采用一组带有整型时钟变量的时间自动机对实时系统的行为进行建模、对它的属性进行验证. UPPAAL 采用的模型验证机制可以有效缓解状态空间爆炸问题[19]. 一个实时系统模型包含一组带有有限控制结构和实数时钟值的进程, 这些控制结构和时钟变量可以通过通道以及共享全局变量进行相互通信.

2 抢占式优先级时间自动机

PPTA 是在 TA 的基础上进一步的扩展, 观察到实时系统引入优先级后所带来的抢占式调度问题, 通过在传统 TA 的基础上增加变迁的优先级以及阻塞位置, 使其能够建模带有优先级的任务, 同时引入 PPTAN, 使其能够模拟优先级抢占式调度问题, 从而对引入优先级的实时系统进行建模, 下面给出 PPTA 以及 PPTAN 的形式化定义, 并证明了 PPTA 是在 TA 的基础上扩展的.

2.1 抢占式优先级时间自动机

2.1.1 语 法

一个 PPTA 用一个九元组 $A=(L,l_0,\Sigma,X,I,T,\chi,\alpha,B)$ 表示, 其中 L,l_0,Σ,X,I 与时间自动机的语法一样.

$\chi\in X$ 是计时时钟变量. 其只作用于阻塞位置, 用于记录系统处于阻塞位置的时间长短. 当阻塞位置发生变迁时, 计时时钟变量重置为 0.

$\alpha:T\to N^+$: 是一个优先级标签函数, 代表迁移的优先级. 将迁移映射到正整数上, 其中较大的值意味着该迁移具有更高的优先级, 每一个迁移的优先级值默认为 1.

$B=\{b_1,b_2,b_3,\ldots,b_n\}\subseteq L$ 是包含 n 个阻塞位置的有穷集, 其目的是描述抢占关系. 当任务 i 处于 l 位置且占有资源时, 由于被任 j 抢占资源被阻塞, 那么任务 i 会进入阻塞位置 b.

$T\subseteq L\times\Sigma\times\alpha(T)\times\phi(X)\times 2^X\times L$ 是一个迁移集合. 一个六元组 $<l,a,\alpha,\varphi,\lambda,l'>\in T$ 表示一个事件标号为 a, 优先级为 α, 从位置 l 到 l' 的迁移. φ 是定义在时钟集 X 上的一个时钟约束函数, 在迁移发生时必须被满足. λ 表示迁移发生时被重置的所有时钟组成的集合, 且 $\lambda\subseteq X$. a,α,φ,λ 在必要时都是可以省略的.

2.1.2 语 义

PPTA 的语义模型为迁移系统 $T_{\text{PPTA}}=<S,s_0,\Sigma_{(a,\alpha,\varphi,\lambda)},\to>$, 其中,

$S\subseteq L\times\mu\times\chi$, $\mu:X\to R^{\geqslant 0}$ 表示 X 上的时钟赋值集合. S 中的每一个状态由一个位置 l, 一个时钟赋值 μ 和一个计时时钟变量 χ 组成, 记为 (l,μ,χ).

$s_0=(l_0,\mu_0,\chi_0)\in S$ 是初始状态.

$\Sigma_{(a,\alpha,\varphi,\lambda)}=\{(a_0,\alpha_0,\varphi_0,\lambda_0),(a_1,\alpha_1,\varphi_1,\lambda_1),\ldots,(a_n,\alpha_n,\varphi_n,\lambda_n)\}$ 是带有事件标号、迁移优先级值、迁移上时钟约束和迁移后时钟重置的有穷事件集.

$\to$是迁移关系. 有延迟迁移和离散迁移两种.

假设 $s=(l,\mu,\chi)$ 和 $s'=(l',\mu',\chi')$ 为 T_{PPTA} 中的两个状态, 则 T_{PPTA} 中存在两种状态迁移规则.

(1) 延迟迁移: $s\xrightarrow{\tau}s'$ 当且仅当状态 s' 从状态 s 经过 τ 个时间后到达, 即:

1) $l'=l$.

2) $\mu\vDash I(l)$.

3) $\mu'=\mu+\tau\vDash I(l)$.

4) $\chi'=\chi+\tau$.

此种状态迁移是由时间流逝导致的: (1) 迁移发生前后位置不变; (2) 位置 l 上的时钟变量 μ 要满足该位置上的不变式 $I(l)$; (3) 迁移发生后位置上的时钟变量 μ 会增加 τ 个时间且要满足该位置上的不变式 $I(l)$; (4) 系统处于该位置上的计时时钟变量 χ 也会增加 τ 个时间.

(2) 离散迁移: $s\xrightarrow{t}s'$ 表示存在一个迁移 $t=<l,\Sigma_{(a,\alpha,\varphi,\lambda)},l'>$, 即:

1) 如果另一个 PPTA 内部存在迁移 t', 有 $s_1 \xrightarrow{t'} s_2$, 则 $\alpha(t) > \alpha(t')$.

2) $\mu \vDash \varphi(t)$.

3) $l \xrightarrow{t} l'$.

4) ① 如果 $l, l' \notin B, \mu' = \mu[\lambda := 0] \vDash I(l')$.

② 如果 $l \notin B, l' \in B, \mu' = \mu \vDash I(l')$.

③ 如果 $l \in B, l' \notin B, \mu' = \mu[\lambda := \mu - \chi] \vDash I(l')$.

5) $\chi' = \chi[\chi := 0]$.

此种状态迁移是由变迁发生导致的, 即会产生新的状态. (1) 如果在另一个 PPTA 内部存在可以与 t 同时发生的变迁 t', 那么 t' 的优先级要低于 t 的优先级; (2) l 位置上的时钟变量 μ 要满足迁移 t 上的时钟约束 $\varphi(t)$; (3) 位置 l 可以通过迁移 t 到达位置 l'; (4) ① 如果 l, l' 不属于阻塞位置, 则 μ' 重置为 0; ② 如果 l 不属于阻塞位置, l' 属于阻塞位置, 则 μ' 等于 μ; ③ 如果 l 属于阻塞位置, l' 不属于阻塞位置, 则 μ' 等于 $\mu - \chi$, 即恢复被阻塞之前的时钟赋值. 结合第②、③两点, 我们定义: 非阻塞位置是由阻塞位置迁移过去的, 那么当阻塞位置迁移回非阻塞位置后, 非阻塞位置上的时钟赋值要恢复到之前被阻塞的时间点, 可以做到在建模低优先级任务时, 当其再次获得资源后, 会在被中断的地方继续执行. 具体在第 2.4 节会详细说明; (5) 最后计时时钟变量 χ' 重置为 0.

2.2 抢占式优先级时间自动机网络

由于实时系统通常由多个任务交互而成, 而一个 PPTA 不能对整个系统建模, 因此需要引入 PPTAN. PPTAN 是一组抢占式优先级时间自动机 $\text{PPTA}_1,\ldots,\text{PPTA}_n$ 并行组合, 它们之间通过通道或共享全局变量的方式进行通信, 同步通信使用输入/输出动作以握手同步的方式进行, 异步通信以共享全局变量的方式进行. 为了模拟握手的同步, PPTA 中的标记集 Σ 中应包括输入动作符号 $a?$ 和输出动作符号 $a!$. 下面给出 PPTAN 的形式定义.

2.2.1 语 法

对于一组抢占式优先级时间自动机 $\text{PPTA}_i = (L_i, l_{i0}, \Sigma_i, X_i, I_i, T_i, \chi_i, \alpha_i, B_i)(1 \leqslant i \leqslant n)$, 其中, $\text{PPTAN} = (L, l_0, \Sigma, X, I, T, \chi, \alpha, B)$, 其中,

L 是位置集, $l \in L$ 是一个位置向量, $l = (l_1, \ldots, l_n)$, 其中 $l_i \in L_i\ (1 \leqslant i \leqslant n)$.

$l_0 = (l_{10}, \ldots, l_{n0})$ 是初始位置集.

$\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$ 为 PPTAN 中的标记集合.

$X = X_1 \cup \ldots \cup X_n$ 表示 PPTAN 中的时钟变量集合.

$I : L \to \phi(X)$ 给 PPTAN 中每个位置向量赋一个时钟约束, 称为该位置向量的位置不变式, $I(l) = I_1(l_1) \wedge \ldots \wedge I_n(l_n)$.

T 是 PPTAN 中的迁移集合, 其形式为 $l \xrightarrow{a,\alpha,\varphi,\lambda} l'$, 其中 a 为触发迁移的动作, α 为迁移的优先级, φ 为迁移使能条件, λ 为迁移发生后重置的时钟变量集合, a、α、φ、λ 必要时都可以省略.

$\chi = \chi_1 \cup \ldots \cup \chi_n$ 是 PPTAN 中计时时钟变量集合.

$\alpha : T \to N^+$ 给 PPTAN 中的迁移赋一个正整数值, $\alpha(t) = \alpha_1(t_1) \wedge \ldots \wedge \alpha_n(t_n)$.

$B \subseteq L$ 是 PPTAN 中阻塞位置的集合.

2.2.2 语 义

PPTAN 的语义为 $T_{\text{PPTAN}} = < S, s_0, \to >$.

$S \subseteq L \times \mu \times \chi$. S 中的每一个状态由一个位置向量 l, 一个时钟赋值集合 μ 和一个计时时钟变量集合 χ 组成, 记为 (l, μ, χ).

$s_0 = (l_0, \mu_0, \chi_0) \in S$ 是初始状态.

$\to$是迁移关系. PPTAN 的状态迁移也分为延迟迁移和离散迁移两种类型, 其中离散迁移又分为两类: 一类是 PPTAN 中的子自动机独立地执行其内部迁移而引发的 PPTAN 的状态迁移; 另一类是 PPTAN 中的两个子自动机之间通过通道进行同步, 然后这两个子自动机执行其内部迁移而引发的 PPTAN 的状态迁移.

令 l_i 表示位置向量 l 的第 i 个位置, $l[l'_i / l_i]$ 表示在位置向量 l 中用位置 l'_i 替换位置 l_i.

(1) $(l,\mu,\chi)\xrightarrow{\tau}(l,\mu+\tau,\chi+\tau)$, 当且仅当 $\tau\in R^+,\mu\vDash I(l)$ 且 $\mu+\tau\vDash I(l)$.

该迁移为延迟迁移, 仅由时间流逝导致. 迁移前后位置 l 没有发生变化, 只是迁移后时钟变量 μ 和计时时钟变量 χ 增加了 τ 个时间单位, 并且迁移前后位置 l 上的时钟变量都要满足位置 l 上的不变式 $I(l)$.

(2) $(l,\mu,\chi)\xrightarrow{t}(l[l_i'/l_i],\mu',\chi')$, 当且仅当在第 i 个 PPTA 内部存在迁移 $t=<l_i,a,\alpha,\varphi_i,\lambda_i,l_i'>$:

1) 如果在第 j 个 PPTA 内部存在一个迁移 $t'=<l_j,b,\alpha,\varphi_j,\lambda_j,l_j'>$, 则 $\alpha_i(t)>\alpha_j(t')\ (i\neq j)$.

2) $\mu\vDash\varphi_i(t)$.

3) ① 如果 $l_i,l_i'\notin B,\mu'=\mu[\lambda_i:=0]\vDash \mathrm{I}(l[l_i'/l_i])$.

② 如果 $l_i\notin B,l_i'\in B,\mu'=\mu\vDash \mathrm{I}(l[l_i'/l_i])$.

③ 如果 $l_i\in B,l_i'\notin B,\mu'=\mu[\lambda_i:=\mu-\chi]\vDash \mathrm{I}(l[l_i'/l_i])$.

4) $\chi'=\chi[\chi:=0]$.

该迁移为离散迁移第 1 类, 即由 PPTAN 中的第 i 个自动机独立地执行其内部迁移 t 而引发的状态迁移. 该状态迁移应满足 4 个条件: (1) 如果 PPTAN 中第 j 个自动机也存在迁移 t', 则迁移 t 的优先级大于 t' 的优先级; (2) 迁移前时钟变量要满足迁移 t 上的时钟约束 $\varphi_i(t)$; (3) ① 如果 l_i, l_i' 不属于阻塞位置, 则迁移后时钟变量 μ' 重置为 0; ②如果 l_i 不属于阻塞位置, l_i' 属于阻塞位置, 则迁移后时钟变量 μ' 等于 μ; ③ 如果 l_i 属于阻塞位置, l_i' 不属于阻塞位置, 则迁移后时钟变量 μ' 等于 $\mu-\chi$, 即恢复被阻塞之前的时钟赋值; (4) 迁移后计时时钟变量 χ' 重置为 0.

(3) $(l,\mu,\chi)\xrightarrow{t}(l[l_i'/l_i,l_j'/l_j],\mu',\chi')$, 当且仅当存在 $i\neq j$ 满足:

1) $l_i\xrightarrow{a?,\varphi_i,\lambda_i}l_i',l_j\xrightarrow{a!,\varphi_j,\lambda_j}l_j'$.

2) $\mu\vDash\varphi_i\wedge\varphi_j$.

3) ① 如果 $l_i,l_i'\notin B,\mu'=\mu[\lambda_i:=0]\vDash I(l[l_i'/l_i,l_j'/l_j])$.

② 如果 $l_i\notin B,l_i'\in B,\mu'=\mu\vDash I(l[l_i'/l_i,l_j'/l_j])$.

③ 如果 $l_i\in B,l_i'\notin B,\mu'=\mu[\lambda_i:=\mu-\chi]\vDash I(l[l_i'/l_i,l_j'/l_j])$.

4) $\chi'=\chi[\chi:=0]$.

该迁移为离散迁移的第 2 类, 即由 PPTAN 中的第 i 个和第 j 个自动机之间通过同步信道事件 a 进行同步, 然后这两个子自动机执行其内部迁移而引发的状态迁移. 该状态迁移应满足 4 个条件: (1) 第 i 个和第 j 个自动机通过信道 a 进行同步迁移; (2) PPTAN 中位置 l 上的时钟变量要同时满足两个子自动机各自迁移上的时钟约束; (3) 由于两个子自动机不会同时在阻塞位置发生同步迁移, 因此这里针对其中一个自动机, 对其迁移前后的位置是否为阻塞位置进行了讨论, 讨论情况与上述离散迁移第 1 类中第 (3) 点类似. 由于该迁移是由两个子自动机同步迁移完成, 所以时钟变量 μ' 重置之后需要满足两个子自动机迁移后位置上的不变式, 即 $I(l[l_i'/l_i,l_j'/l_j])$; (4) PPTAN 中状态发生迁移后, 计时时钟变量重置为 0.

2.3 精化关系

下面证明 TA 的语义模型是 PPTA 的子语义模型: 首先, 给出子语义模型的定义; 其次给出 PPTA 包含 TA 语义的定理证明, 证明 PPTA 是在 TA 的基础上进行扩展.

定义 1. 设一个 A 类具有语义 S_A 的时间自动机 TA_A 能够通过一个精化关系得到另一个 B 类具有语义 S_B 的时间自动机 TA_B, 那么称语义 S_B 是语义 S_A 的子语义模型[20]. TA_A 与 TA_B 的精化模型如下:

$$\frac{TA_B\ \text{sat}\ s_B\ \text{in}\ S_B}{TA_A\ \text{sat}\ s_A\ \text{in}\ S_A}\ (TA_B\subseteq TA_A),$$

其中, s_A 是 TA_A 的语义, s_B 是 TA_B 的语义.

定义 2. 抢占式优先级时间自动机 PPTA 所有可接受的语言是抢占式优先级迁移序列的集合.

定理 1. 时间自动机 TA 的语义模型是抢占式优先级时间自动机 PPTA 的子语义模型.

证明: 由 TA 的语义定义可知, TA 是一个迁移系统 $T_A=<S,s_0,\Sigma_{(a,\varphi,\lambda)},\to>$, 其接受的语言为 L. 在 L 中任取一

段迁移序列 $R=\{(a_0,\varphi_0,\lambda_0),(a_1,\varphi_1,\lambda_1),\ldots,(a_{n-1},\varphi_{n-1},\lambda_{n-1})\}$, 可以给出 T_A 的状态迁移序列:

$$(l_0,\mu_0)\xrightarrow{a_0,\varphi_0,\lambda_0}(l_1,\mu_1)\xrightarrow{a_1,\varphi_1,\lambda_1}\ldots\xrightarrow{a_{n-1},\varphi_{n-1},\lambda_{n-1}}(l_n,\mu_n).$$

由 PPTA 的语义可知, PPTA 也是一个迁移系统 $T_{\text{PPTA}}=<S,s_0,\Sigma_{(a,\alpha,\varphi,\lambda)},\rightarrow>$, 其接受的语言为 L'. 在 L' 中任取一段迁移序列 R', 可以给出 T_{PPTA} 的状态迁移序列.

(1) 当 $R'=\{(a_0,\alpha_0,\varphi_0,\lambda_0),(a_1,\alpha_1,\varphi_1,\lambda_1),\ldots,(a_{n-1},\alpha_{n-1},\varphi_{n-1},\lambda_{n-1})\}$, 状态迁移序列中存在阻塞位置, 即 $B\neq\{\varnothing\}$ 时. 此种情况下的状态迁移序列为:

$$(l_0,\mu_0,\chi_0)\xrightarrow{a_0,\alpha_0,\varphi_0,\lambda_0}\ldots\xrightarrow{a_{i-1},\alpha_{i-1},\varphi_{i-1},\lambda_{i-1}}(l_i,\mu_i,\chi_i)\xrightarrow{a_i,\alpha_i,\varphi_i,\lambda_i}\ldots\xrightarrow{a_{n-1},\alpha_{n-1},\varphi_{n-1},\lambda_{n-1}}(l_n,\mu_n,\chi_n),$$

其中, $l_i\in B\ (0<i<n)$ 为阻塞位置.

(2) 当每一个迁移的优先级都为 1 且状态迁移过程中不存在阻塞位置时, 即 $\alpha=1\wedge B=\{\varnothing\}$, 此时 $R'=\{(a_0,1,\varphi_0,\lambda_0),(a_1,1,\varphi_1,\lambda_1),\ldots,(a_{n-1},1,\varphi_{n-1},\lambda_{n-1})\}$. 由于每一个迁移的优先级都为 1, 所以迁移优先级的限制就不存在, 此时 $R'=R$; 又因为状态迁移过程中不存在阻塞位置, 计时时钟变量不起作用, 所以状态中的计时时钟变量可以忽略掉. 此时的状态迁移序列为:

$$(l_0,\mu_0)\xrightarrow{a_0,\varphi_0,\lambda_0}(l_1,\mu_1)\xrightarrow{a_1,\varphi_1,\lambda_1}\ldots\xrightarrow{a_{n-1},\varphi_{n-1},\lambda_{n-1}}(l_n,\mu_n).$$

综上所述, 在 L 中任取一个迁移序列 R, 在 L' 中都有一个 R' 与之对应且 TA 中迁移序列 R 所对应的状态迁移序列在 PPTA 中也存在一个迁移序列与之对应, 所以 $\Sigma_{(a,\varphi,\lambda)}\subseteq\Sigma_{(a,\alpha,\varphi,\lambda)}$, 其中, 当 $\alpha=1\wedge B=\{\varnothing\}$ 时, $\Sigma_{(a,\varphi,\lambda)}=\Sigma_{(a,\alpha,\varphi,\lambda)}$. 因此, PPTA 与 TA 是精化关系. 根据定义 1 可得, TA 的语义模型为 PPTA 的子语义模型, 所以, PPTA 是在 TA 的基础上进行的扩展.

2.4 实 例

下面利用本文提出的 PPTA 对单核多任务实时系统进行建模来解释本文提出的相关概念.

单核多任务实时系统是指 3 个任务 (A, B, C) 在执行时共用一个处理器 (P). 任务之间存在依赖关系, 任务 B 依赖任务 A, 即任务 A 的执行结束作为任务 B 的启动. 系统的任务依赖图如图 1 所示 (图中箭头表示依赖关系). 任务 A 和任务 C 的执行周期分别是 30 ms, 40 ms. 任务 A 获得处理器后执行 5 ms 结束执行, 任务 B 获得处理器后执行 10 ms 结束执行, 任务 C 获得处理器后执行 5 ms 结束执行. 其中每一个任务的执行截止时间都为 20 ms. 任务之间除了依赖关系, 还存在优先级关系, 任务 A, B, C 的优先级分别是 6, 7, 8. 因此任务在执行时, 不仅要考虑优先级关系, 还要考虑依赖关系.

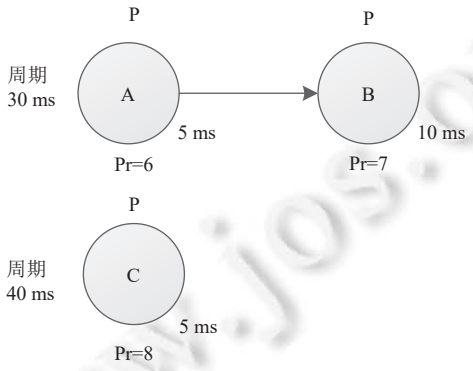


图 1 单核多任务实时系统任务依赖图

建模该系统的 PPTAN (其中 n=2), 如图 2 所示. PPTA_1 是建模任务 A 和任务 B 的 PPTA, PPTA_2 是建模任务 C 的 PPTA.

对于 PPTA_1, 其中各部分含义如表 1 所示.

对于 PPTA_2, 其中各部分含义如表 2 所示.

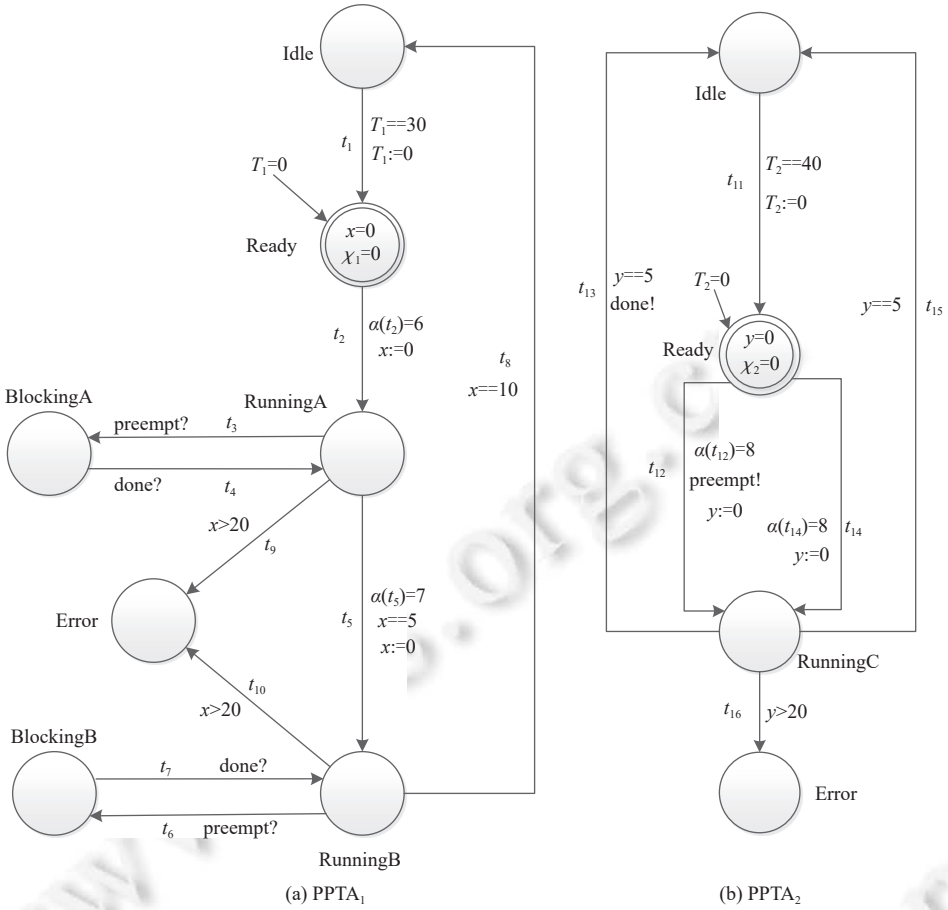


图 2 PPTAN

表 1 图 2(a) 各部分含义

名称	元素	说明
状态	$S=\{\text{Idle},\text{Ready},\text{RunningA},\text{BlockingA},\text{RunningB},\text{BlockingB},\text{RunningC},\text{Error}\}$	分别代表空闲、准备、执行任务A、任务A处于阻塞、执行任务B、任务B处于阻塞、执行任务C、错误状态 (任务执行时间如果超过其截止时间则进入错误状态)
初始状态	$s_0=\{\text{Ready}\}$	初始状态为Ready
优先级标签函数	$\alpha:\alpha(t_2)=6,\ \alpha(t_5)=7$	分别表示任务A的优先级为6, 任务B的优先级为7
时钟变量	$X=\{x,T_1,\chi_1\}$	x记录任务A, B的执行时间, T_1记录任务A从Ready状态到Idle状态的时间, χ_1 为计时时钟变量
时钟约束	$\phi(x)=\{T_1==30,x==5,x==10\}$	表示任务A每隔30 ms启动一次, 以及任务A和任务B分别执行5 ms、10 ms后结束执行
重置时钟	$\lambda=\{T_1:=0,x:=0\}$	表示将时钟T_1重置为0, 时钟x置为0
迁移	$T=\{t_1,t_2,t_3,t_4,t_5,t_6,t_7,t_8,t_9,t_{10}\}$	迁移 t_5 表示任务A驱动任务B执行

对于整个系统来说, 开始时, 任务 A, C 处于 Ready 状态, 由于任务 C 的优先级高于任务 A 且不需要抢占处理器, 因此任务 C 通过变迁 t_{14} 进入 RunningC 状态, 执行 5 ms 后通过变迁 t_{15} 进入空闲状态; 此时任务 A 开始执行, 执行 5 ms 之后驱动任务 B 执行, 任务 B 执行 10 ms 之后回到空闲状态. 当整个系统运行到 30 ms 时, 任务 A 再次被驱动, 执行 5 ms 后驱动任务 B 执行, 当任务 B 执行到第 5 ms 时, 整个系统的运行时间来到 40 ms, 此时任务 C

被驱动, 由于任务 B 占用处理器, 且任务 C 的优先级比任务 B 高, 则任务 C 通过变迁 t_{12} 发出抢占信号 preempt, 当变迁 t_6 接收到抢占信号后被激发, 导致任务 B 进入阻塞状态 BlockingB (即阻塞前状态 RunningB 的时钟变量 x=5). 由于 t_6 迁移是从非阻塞位 RunningB 置到阻塞位置 BlockingB, 所以根据 PPTA 离散迁移语义第 (4) 条第②点, 此时 BlockingB 的时钟变量为 x=5, 计时时钟变量 χ=0. 当任务 C 执行 10 ms 后 (此时 BlockingB 的时钟变量 x=15, 计时时钟变量 χ=10), 通过变迁 t_{13} 回到空闲状态, 并发出执行完毕信号 done, 当变迁 t_7 接收到任务 C 执行完毕信号后, 使得任务 B 从阻塞状态回到执行状态. 由于 t_7 迁移是从阻塞位置 BlockingB 到非阻塞位置 RunningB, 所以根据 PPTA 离散迁移语义第 (4) 条第③点, 此时 RunningB 的时钟变量 x=5 (阻塞位置 BlockingB 的时钟变量-计时时钟变量), 因此任务 B 又恢复到被阻塞之前的中断时间点, 所以任务 B 会从中断的地方继续执行. 任务 B 在执行 5 ms 之后, 通过迁移 t_8 回到空闲状态. 如果任务的执行时间超过其截止时间, 则该任务会进入错误状态, 此时该系统不可调度.

表 2　图 2(b) 各部分含义

名称	元素	说明
状态	$S=\{\text{Idle},\text{Ready},\text{RunningC},\text{Error}\}$	分别代表空闲、准备、执行任务C、错误状态
初始状态	$s_0=\{\text{Ready}\}$	初始状态为Ready
标签函数	$\alpha:\alpha(t_{12})=8,\alpha(t_{14})=8$	都是表示任务C的优先级为8
时钟变量	$X=\{y,T_2,\chi_2\}$	y记录任务C的执行时间, T_2记录任务C从Ready状态到Idle状态的时间, χ_2 为计时时钟变量
时钟约束	$\phi(X)=\{T_2==40,y==5\}$	表示任务C每隔40 ms启动一次, 以及任务C执行10 ms后结束执行
重置时钟	$\lambda=\{T_2:=0,y:=0\}$	表示将时钟T_2重置为0, 时钟y置为0
迁移	$T=\{t_{11},t_{12},t_{13},t_{14},t_{15},t_{16}\}$	其中 t_{12},t_{13} 表示任务C通过抢占任务A或任务B所占用的处理器来执行, t_{14},t_{15} 表示任务C不是通过抢占处理器, 而是通过正常请求处理器来执行

3 转换方法与验证

由于实时系统中的一个状态可以对应多个状态, 因此, 从一个状态出发有多条路径, 形成一个树状结构, 那么在验证相关性质时就会涉及多条路径的性质. CTL 非常适合于表达计算树上的正确性, 因为其使用显示路径量词 (A, E) 来区分必要和可能的行为的分支时间能力提供了显著的表达能力[21], 所以 CTL 适合于本文对于系统性质的描述. 并且, CTL 模型检验的复杂度与原迁移系统和公式的长度都呈线性关系, 然而 LTL 模型检验的复杂度与性质的公式长度成指数关系[22]. 因此本节使用 CTL 语言来描述实时系统的设计需求. 首先给出了 CTL 语言的语法和语义, 其次提出了 PPTA 转换到 UPPAAL 所支持输入模型 TA 的转换规则与转换算法, 最后利用 CTL 语言描述了本文所要验证的实时系统的相关性质.

3.1 CTL

首先给出 CTL 的语法, 然后 CTL 的语义通过 PPTAN 迁移系统 T_{PPTAN} 来解释.

CTL 的语法: p 是原子命题.

$$\phi ::= \text{true} \mid p \mid \neg p \mid p_1 \wedge p_2 \mid EXp \mid E(p_1\ U\ p_2) \mid A(p_1\ U\ p_2).$$

CTL 中的其他算子可以由上述的算子推出: $p_1 \vee p_2 = \neg(\neg p_1 \wedge \neg p_2)$; $EFp = E(\text{true}\ U\ p)$; $AFp = A(\text{true}\ U\ p)$; $EGp = \neg AF\neg p$; $AGp = \neg EF\neg p$; $AXp = \neg EX\neg p$.

CTL 的语义: 对于一个迁移系统 $T_{\text{PPTAN}} =< S, s_0, \rightarrow >$, 给出以下定义:

定义 3. 令 $\omega = (s_0, s_1, s_2, \ldots)$ 表示一条状态路径. 对于 $\forall i \in \{0,1,2,\ldots\} : (s_i, s_{i+1}) \in \rightarrow$ 且 $\omega(i) = s_i$.

定义 4. 令 S^w 表示状态路径集合, 则以状态 s 为初始点的路径集合可以定义为 $\Omega(s) = \{\omega \in S^w \mid \omega(0) = s\}$.

定义 5. $Label : S \rightarrow 2^{AP}$ 称为标签函数. 标签函数 $Label$ 将原子命题的一个集合 $Label(s) \in 2^{AP}$ 与状态 s 联系起

来. $Label(s)=\{a|a\in AP$ 且 a 完全满足状态 $s\}$.

给定一个 T_{PPTAN} 中的状态 $s=(l,\mu,\chi)$, 一个 CTL 公式 ϕ, $(T_{\text{PPTAN}},s)\vDash\phi$ 意味着公式 ϕ 在迁移系统 T_{PPTAN} 中的状态 s 下为真, 即迁移系统 T_{PPTAN} 中的状态 s 满足公式 ϕ. 若没有歧义, T_{PPTAN} 可以省略, 满足关系 $\vDash$ 归纳定义如下:

$s\vDash p$ 当且仅当 $p\in Label(l)$.

$s\vDash\neg p$ 当且仅当 $\neg s\vDash p$.

$s\vDash p_1\wedge p_2$ 当且仅当 $s\vDash p_1$ 且 $s\vDash p_2$.

$s\vDash EXp$ 当且仅当 $\exists\omega\in\Omega(s),\omega(1)\vDash p$.

$s\vDash E(p_1\ U\ p_2)$ 当且仅当 $\exists\omega\in\Omega(s),(\exists j\geqslant 0,\omega(j)\vDash p_2\wedge(\forall 0\leqslant k<j,\omega(k)\vDash p_1))$.

$s\vDash A(p_1\ U\ p_2)$ 当且仅当 $\forall\omega\in\Omega(s),(\exists j\geqslant 0,\omega(j)\vDash p_2\wedge(\forall 0\leqslant k<j,\omega(k)\vDash p_1))$.

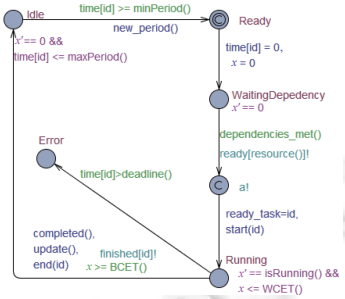
3.2 转换规则

一个特定调度问题模型应由两个不同的时间自动机模板组成: 一个通用任务模板、一个通用处理器模板[15]. 为了验证 PPTA 模型能否建模优先级实时系统, 我们将 PPTA 模型转换为自动验证工具 UPPAAL 所支持的通用任务 TA 模板. 同时我们在 UPPAAL 中设计了通用处理器 TA 模板.

3.2.1 PPTA 到通用任务 TA 模板

表 3 给出了 PPTA 到任务 TA 模板的转换规则, 下面给出几点说明.

表 3 转换规则

规则名称	PPTA	UPPAAL模型	描述
TimingClock2Clock	clock variable c timing clock variable χ	clock c	计时时钟变量转换为时钟变量
PriLabelFunction2PriArray	$\alpha(t_i)=z$	const int $P[N]$={…, z, …}	优先级标签函数转换为任务优先级数组声明 (其中 $P[i]=z$ 表示的是第 i 个任务的优先级为 z)
TaskDepend2DependMarix		const bool *Depend*[2][2]{{0,0},{1,0}}	任务依赖关系转换为任务依赖矩阵声明(*Depend*[2][1]表示任务B依赖任务A)
RunningState2TaskTemp	 		PPTA建模的每个任务转换为任务TA模板 (id.Running表示PPTA中任务的执行状态, 利用该状态表示PPTA所建模的每一个任务)

(1) 根据表 3 中的 TimingClock2Clock 规则, PPTA 模型中的计时时钟变量转换为 UPPAAL 中 Declarations 的时钟变量声明.

(2) 根据表 3 中的 PriLabelFunction2PriArray 规则, PPTA 模型中的优先级标签函数转换为 UPPAAL 中 Declarations 的任务优先级数组声明.

(3) 根据表 3 中的 TaskDepend2DependMarix 规则, PPTA 模型中的任务依赖关系转换为 UPPAAL 中 Declarations 的任务依赖矩阵声明.

(4) 根据表 3 中的 RunningState2TaskTemp 规则, PPTA 中的每一个任务的执行状态转换为 UPPAAL 中的 TA 任务模板. 由于在 UPPAAL 中一个 TA 只能建模一个任务, 所以我们首先在 UPPAAL 中设计了一个可以建模任务的通用 TA 模板, 如图 3 所示. 其次, 我们定义的 PPTA 在建模任务时, 一个任务存在一个执行状态, 所以, 我们在

将 PPTA 所建模的任务模型转换到 UPPAAL 中时, 直接是将 PPTA 中的执行状态转换为我们所得到的 TA 模板, 验证时只需要对模板实例化任务 id 即可.

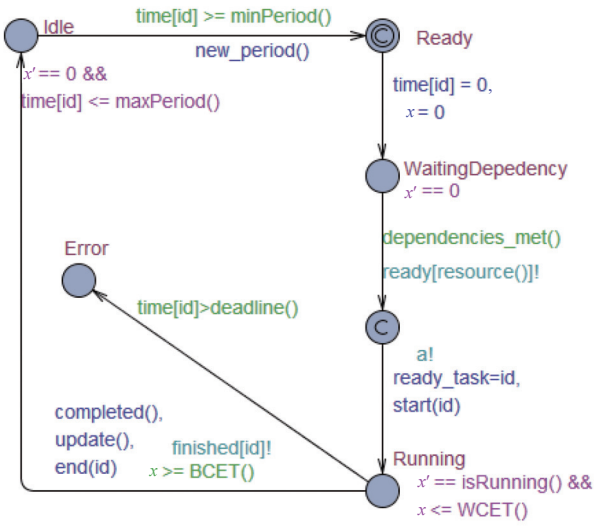


图 3 任务 TA 模板

任务模板有 5 个状态, 分别是 Idle (空闲)、Ready (准备)、WaitingDepedency (等待依赖)、Running (运行) 以及 Error (错误). 该模板用于建模任务, 其中任务的优先级通过在 UPPAAL 中声明优先级数组 (priority array) 表示, 任务之间的依赖关系通过在 UPPAAL 中声明依赖矩阵 (dependency matrix) 表示. 模板上的变量, 函数等也是通过在 UPPAAL 中进行声明.

任务的初始状态为准备状态, 下一个状态则为等待依赖状态, 其中 dependencies_met() 函数通过检查任务依赖矩阵来判断该任务所依赖的其他任务是否已经执行完毕, 如果该任务的依赖条件满足, 则进入执行状态. 只有在执行状态下, 时钟变量 x 才会增加, 用来记录任务的执行时间. 当该任务被其他高优先级的任务阻塞时, isRunning() 函数被置为 0, 此时 x' ==0, 表示时钟变量 x 保持当前的值且不会增加; 当该任务再次获得资源时, isRunning() 函数被置为 1, 此时 x' ==1, 表示时钟变量继续从被阻塞时的数值开始增加. 这样便可以实现当低优先级的任务再次获得资源时, 继续从中断的位置执行. 当该任务一个周期的执行时间 time[id] 超过了任务的截止时间, 那么该任务则会进入错误状态, 进而导致整个系统不可调度, 如果没有超过截止时间, 任务执行完毕后就会回到空闲状态. 至此, 该任务第 1 个周期的执行结束, 等待第 2 个周期执行的到来.

3.2.2 通用处理器 TA 模板

我们在 UPPAAL 中设计通用处理器 TA 模板, 如图 4 所示. 通过转换规则得到建模任务的 TA 模板以及 UPPAAL 声明后, 把处理器模板加入到 UPPAAL 中即可.

处理器模板有 4 个状态, 分别是 Idle (空闲)、TaskInsert (任务插入)、InUse (使用) 以及 BufferEmpty (缓冲序列是否为空). 处理器的初始状态为空闲状态, 当其通过通道 ready[id] 收到任务发来的申请处理器的请求时, 进入到任务插入状态. 在该状态判断缓冲区队列是否为空: 如果为空, 通过 insert_at() 函数将任务直接插入到缓冲区队头, 此时处理器进入使用状态; 如果不为空, insert() 函数通过访问优先级数组将任务按优先级的大小以冒泡排序的方式插入到待处理任务缓冲区序列中, 以此来实现静态优先级策略, 此时处理器进入使用状态.

当处理器处于使用状态时: 如果有其他任务申请处理器, 那么处理器会再次回到任务插入状态, 通过 insert() 函数将高优先级的任务插入到缓冲区队列中正在执行的任务前面, 将低优先级的任务插入到缓冲区队列中正在执行的任务后面, 以此实现不同优先级任务之间的抢占关系; 如果没有其他任务申请处理器, 那么当前任务执行结束后, 处理器会进入判断缓冲区队列是否为空状态: 如果缓冲区队列为空, 处理器回到空闲状态, 等待其他任务的申

请; 如果缓冲区队列不为空, 处理器再次回到使用状态, 继续执行队列中的其他任务. 通过处理器模板与任务模板之间的交互, 便可以模拟抢占式调度问题.

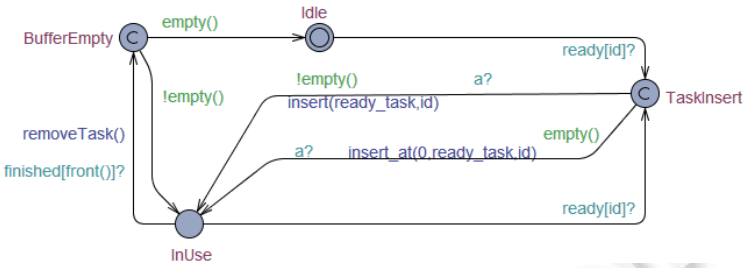


图 4 处理器 TA 模板

3.3 转换算法

本节通过广度优先遍历算法将 PPTA 模型转换到 UPPAAL 中的 TA 任务模板. 算法的伪代码见算法 1.

算法 1. PPTA2TA.

输入: PPTA;

输出: TA.

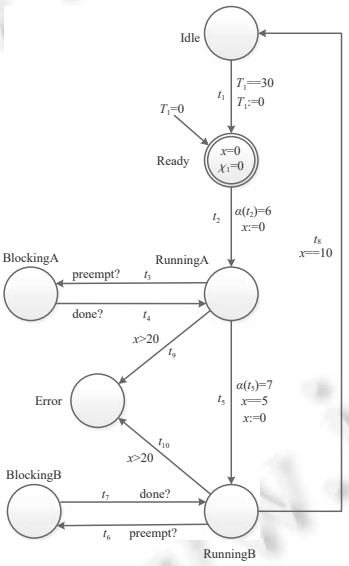
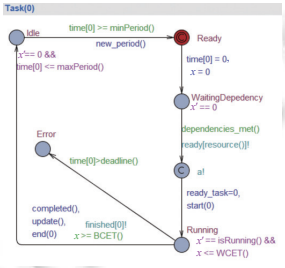
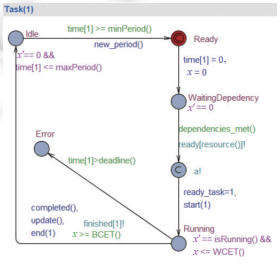
1. **initialization**: visited[]=null, Queue Q=null; //visited 数组存储遍历过的状态节点, 队列 Q 存储状态节点
2. V=getVar(PPTA); // getVar() 函数为获取 PPTA 中的时钟变量声明集合
3. **while**(V)
4. transRule1(v); // 利用 TimingClock2Clock 规则将获得的 PPTA 中的时钟变量声明转换为 UPPAAL 中的时钟变量声明
5. **endwhile**
6. BFS(PPTA, s_0)
7. visited←add(s_0); //add() 函数表示将 s_0 状态节点加入到 visited 数组中
8. Enque(Q, s_0);
9. **while**(Q)
10. s=Deque(Q);
11. Post_s=getpost(s); //getpost() 函数的作用是获得状态 s 的所有后继状态
12. **for** post_s : Post_s //对 Post_s 集中的每一个状态元素 post_s 执行
13. **if**(post_s ∉ visited)
14. α =getpriority(s, post_s); //获得 s 状态到 post_s 状态迁移的优先级函数
15. **if**(α)
16. transRule2(α); **endif** //利用 PriLabelFunction2PriArray 规则将变迁优先级转换到 UPPAAL 中任务优先级数组声明中
17. **if** s ∈ RunningX and post_s ∈ RunningX
18. transRule3(s, post_s); **endif** //利用 TaskDepend2DependMarix 规则将 PPTA 中任务依赖关系转换到 UPPAAL 中任务依赖矩阵声明中
19. **if** post_s ∈ RunningX
20. transRule4(post_s); **endif** //利用 RunningState2TaskTemp 规则将建模任务的 PPTA 转换为 UPPAAL 中的建模任务的 TA 模板

21. visited←add(post_s);
22. Enque(Q, post_s);
23. **endif**
24. **endfor**
25. **endwhile**
26. **return** TA

算法 1 中, 首先通过 getVar() 函数得到 PPTA 中的时钟变量, 利用 TimingClock2Clock 规则将其转换到 UPPAAL 中的时钟变量声明. 然后从 PPTA 模型的初始状态进行广度优先遍历, 在遍历过程中: 当遍历到状态之间迁移上的优先级标签函数时, 利用 PriLabelFunction2PriArray 规则, 将其转换到 UPPAAL 中的任务优先级数组声明中; 当遍历到任务之间存在依赖关系时, 利用 TaskDepend2DependMarix 规则将其转换为 UPPAAL 中的任务依赖矩阵声明中; 当遍历到任务的执行状态时, 利用 RunningState2TaskTemp 规则将建模任务的 PPTA 转换为 UPPAAL 中的建模任务的 TA 模板.

下面利用本节提出的转换算法将第 2.4 节中图 2(a) 的 $PPTA_1$ 转换成 UPPAAL 的输入模型 TA, 如表 4 所示.

表 4 PPTA 转换为 UPPAAL 的输入模型 TA

规则名称	PPTA	TA
TimingClock2Clock	clock variable x clock variable T_1 timing clock variable χ_1	clock x clock time[0], time[1] clock x
PriLabelFunction2PriArray	$\alpha(t_2)=6, \alpha(t_5)=7$	const int P[2]={6, 7}
TaskDepend2DependMarix		const bool *Depend*[2][2]{{0, 0}, {1, 0}}
RunningState2TaskTemp		 实例化后的任务 A  实例化后的任务 B

根据算法 1 第 2–5 行获得 $PPTA_1$ 中的时钟变量声明 $\{x, T_1, \chi_1\}$ 并利用 TimingClock2Clock 规则将其转换为 UPPAAL 中的时钟变量声明.

根据算法 1 第 14–16 行, 当遍历到 Ready 状态的后继状态 RunningA 状态时, 两个状态的迁移上存在优先级标签函数 $\alpha(t_2)=6$, 表示任务 A 的优先级为 6; 当遍历到 RunningA 状态的后继状态 RunningB 状态时, 两个状态的

迁移上存在优先级标签函数 $\alpha(t_5)=7$, 表示任务 B 的优先级为 7. 利用 PriLabelFunction2PriArray 规则将上述优先级标签函数转换到 UPPAAL 中的优先级数组声明中.

根据算法 1 第 17, 18 行, 当遍历到 RunningA 状态的后继状态 RunningB 状态时, 由于 RunningA, RunningB 状态均表示执行状态, 表明 $PPTA_1$ 建模了任务 A 与任务 B 之间的依赖关系, 利用 TaskDepend2DependMarix 规则将任务 B 依赖于任务 A 的依赖关系转换到 UPPAAL 的任务依赖矩阵声明中.

根据算法 1 第 19, 20 行, 当遍历到 RunningA 状态时, 该状态表示任务 A 的执行状态; 当遍历到 RunningB 状态时, 该状态表示任务 B 的执行状态. 利用 RunningState2TaskTemp 规则将 $PPTA_1$ 转换为建模任务的 TA 模板. 表 4 中直接给出了任务 TA 模板实例化后建模两个任务的 TA.

3.4 验 证

参照针对优先级抢占式调度问题的一些常见验证属性[6,23], 本文所要验证的属性主要有以下两个方面 (以单核双任务实时系统为例, 两个任务分别为任务 M, 任务 N).

(1) 安全性验证

属性 1: 系统死锁验证. 即验证系统在运行时会不会出现死锁. 用 CTL 公式表示为:

$$\text{AG not deadlock.}$$

属性 2: 在系统运行过程中, 同一时刻, 一个处理器只允许一个任务执行. 即任务 M 与任务 N 不能同时执行. 用 CTL 公式表示为:

$$\text{AG(RunningM+RunningN}\leqslant 1),$$

其中, RunningM, RunningN 表示任务 M, N 处于运行状态.

属性 3: 在系统运行过程中, 任务 M 和任务 N 从一个周期开始到结束所需要的时间不能超出其截止时间, 从而证明该实时系统是可以调度的. 用 CTL 公式表示为:

$$\text{AG}(X_{\text{M}}\leqslant \text{deadlineM}\wedge X_{\text{N}}\leqslant \text{deadlineN}),$$

其中, X_{M}, X_{N} 表示任务 M, N 从一个周期开始到结束所需要的时间, deadlineM, deadlineN 表示任务 M, N 的执行截止时间.

(2) 有界活性验证

属性 4: 在系统运行过程中, 假设任务 M 的优先级比任务 N 的优先级高, 该属性验证任务 N 能否在其规定的时间内完成. 如若不能, 修改执行时间参数, 则可以用来判断任务 N 是否是因为被任务 M 抢占资源而导致执行结束时间延长. 用 CTL 公式表示为:

$$\text{AG}(x_{\text{M}}\leqslant \text{RunningTimeM}),$$

其中, x_{M} 表示任务 M 执行了多长时间, RunningTimeM 表示任务 M 的执行时间.

4 应 用

本节通过一个多核多任务实时系统案例来说明本文所提出模型的正确性和有效性.

该实时系统有 6 个任务 (A, B, C, D, E, F) 以及两个处理器 (c_0, c_1), 其中任务 A, B, F 共用一个处理器 c_1, 任务 C, D, E 共用一个处理器 c_0, 任务之间存在着依赖关系 (图 5 中箭头表示), 即一个任务的执行结束意味着下一个任务的开始. 图 5 表示了整个系统的依赖关系. 如图所示任务 A 和任务 D 周期执行, 任务 A 每隔 80 ms 执行一次, 任务 D 每隔 55 ms 执行一次. 任务 A 请求到处理器 c_1 后, 执行 6 ms 后驱动任务 B 执行, 任务 B 请求到处理器 c_1 后, 执行 10 ms 后驱动任务 C 执行, 任务 C 请求到处理器 c_0 后, 执行 8 ms 后结束执行; 任务 D 请求到处理器 c_0 后, 执行 4 ms 后驱动任务 E 执行, 任务 E 请求到处理器 c_0 后, 执行 22 ms 后驱动任务 F 执行, 任务 F 请求到处理器 c_1 后, 执行 12 ms 后结束执行. 其中每一个任务的执行截止时间都为 80 ms. 此外, 任务之间还存在优先级关系. 任务 A, B, F 的优先级分别是 7, 8, 6, 所以任务 B 可以抢占任务 A, F 所占用的处理器, 任务 A 可以抢占任务 F 所占用的处理器; 任务 C, D, E 的优先级分别是 9, 7, 8, 所以任务 C 可以抢占任务 D, E 所占用的处理器, 任务 E 可以抢占任务 D 所占用的处理器.

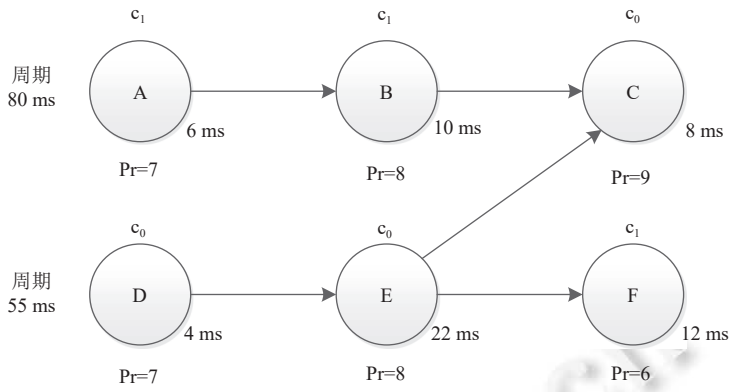


图 5　多核多任务实时系统任务依赖图

4.1 建　模

这样一个多核多任务实时系统可以通过我们定义的 PPTAN (n=3) 来建模, 如图 6 所示. 其中图 6(a) 是建模任务 A, 任务 B 的 PPTA, 图 6(b) 是建模任务 D, 任务 E 以及任务 F 的 PPTA, 图 6(c) 是建模任务 C 的 PPTA.

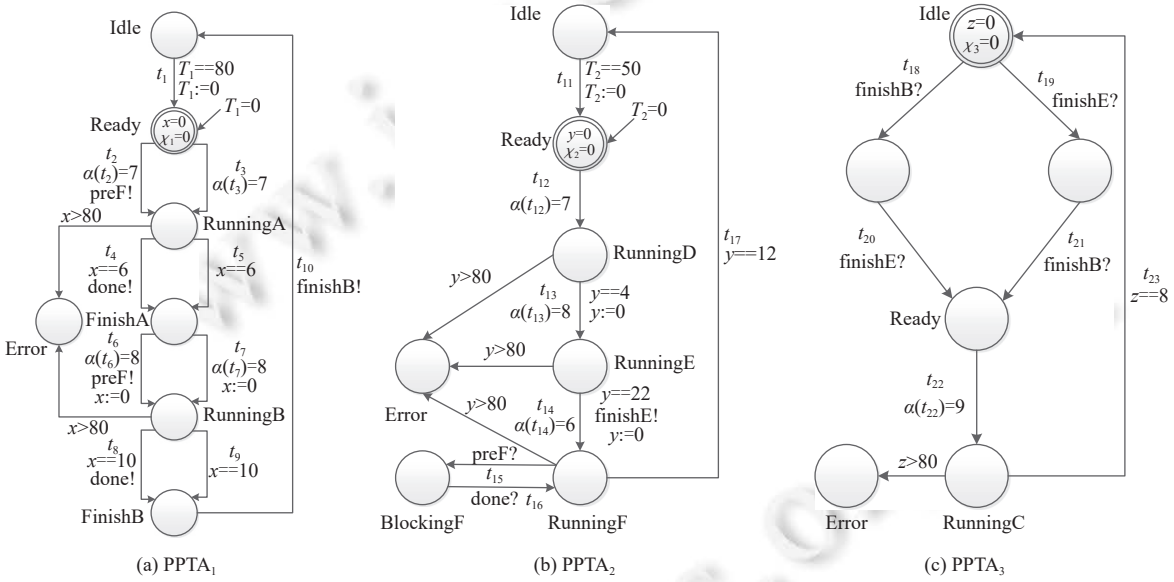


图 6　描述图 5 中的多核多任务实时系统的 PPTAN

由于在第 2.4 节实例部分已经通过单核多任务实时系统解释了相关的概念, 因此这里只解释前面没有解释的部分. 其中, 由于任务 C 依赖于任务 B 和任务 E, 因此, 我们单独对任务 C 进行建模. 变迁 t_{18}, t_{20} 以及变迁 t_{19}, t_{21} 表示任务 C 等待任务 B 和任务 E 执行结束后进入准备状态, 进而进入执行状态. 由于任务 B 和任务 E 执行结束的先后顺序可能不一样, 所以我们用变迁 t_{18}, t_{20} 表示任务 B 先执行结束, 任务 E 后执行结束; 用变迁 t_{19}, t_{21} 任务 E 先执行结束, 任务 B 后执行结束. 其中 finishB、finishE 信号表示任务 B、任务 E 执行结束.

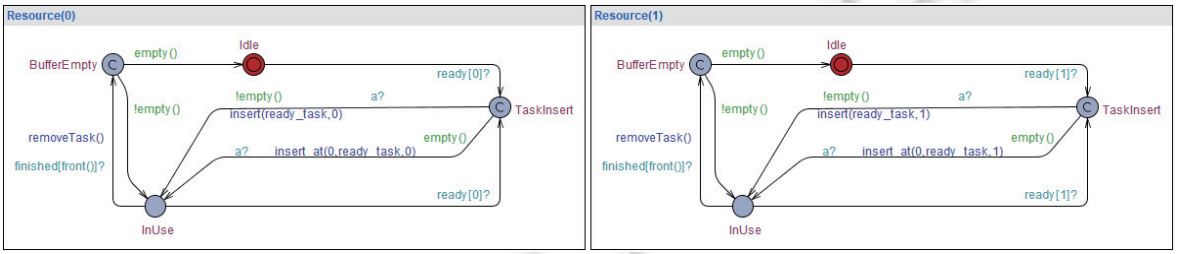
4.2 模型转换

通过第 3.2 节的转换规则与第 3.3 节的转换算法, 我们这里直接给出图 6 中 PPTA 转换后的结果. 其中 UPPAAL 中的相关声明如表 5 所示, 实例化后建模任务的 TA, 如图 7(a) 所示. 同时我们也给出了实例化后建模处理器的 TA, 如图 7(b) 所示.

表 5 UPPAAL 声明

名称	UPPAAL声明
UPPAAL中时钟变量声明	clock x clock time[id]
UPPAAL中任务优先级声明	const int P[6]={7, 8, 9, 7, 8, 6}
UPPAAL中任务依赖关系声明	const bool *Depend*[6][6]{ {0, 0, 0, 0, 0, 0}, {1, 0, 0, 0, 0, 0}, {0, 1, 0, 0, 1, 0}, {0, 0, 0, 0, 0, 0}, {0, 0, 0, 1, 0, 0}, {0, 0, 0, 0, 1, 0}}

(a) 任务模板实例化后的时间自动机



(b) 处理器模板实例化后的时间自动机

图 7 TA 模板实例化

4.3 属性验证

本节首先利用文献 [15] 提出的模型对本节的实例进行建模, 然后再利用本文提出的模型以及文献 [15] 提出的模型对本节的实例验证分析, 进行对比.

在利用文献 [15] 中的模型建模本文实例后, 在 UPPAAL 中运行时, 发现会出现任务与处理器不匹配的情况,

即任务 D 可能会被分配到处理器 c_1 上执行. 这是因为该模型在运行的过程中当任务 A、D 同时申请各自所需的处理器时, 可能会出现参数传递错误的情况, 从而导致任务 D 被加入到了处理器 c_1 的缓冲区中. 因此我们在该模型的基础上增加了一个 Committed 位置, 该位置的作用是, 当自动机处于该位置时, 下一次迁移必定从此位置迁移到下一个位置. 这样就保证了任务自动机与调度器自动机在通过同步通道传递参数时, 不会出现参数传递错误问题. 为了保证在利用文献 [15] 中的模型验证属性时不会出现任务与处理器不匹配的情况, 以下验证都是利用在其基础上增加了 Committed 位置之后的模型验证的.

● 安全性验证属性 1: 系统死锁验证. 在 UPPAAL 中用 CTL 公式表示为:

A[] not deadlock.

● 安全性验证属性 2: 由于多核多任务实时系统中有两个处理器, 因此系统在执行过程中, 同一时刻最多有两个任务在执行. 在 UPPAAL 中用 CTL 公式表示为:

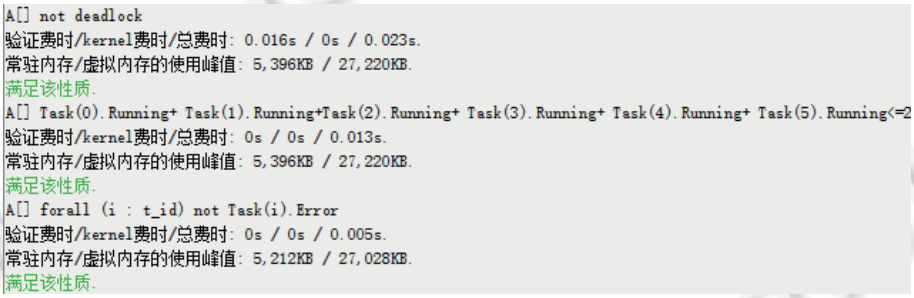
A[] Task(0).Running + Task(1).Running + Task(2).Running + Task(3).Running + Task(4).Running + Task(5).Running≤2.

● 安全性验证属性 3: 如图 7(a) 所示, 每一个任务进入 Error 状态的条件是 time[*id*]>deadline()(time[*id*] 记录了任务从一个周期的开始到结束所花费的时间, 该公式则表示任务 *id* 超过了其截止时间), 因此在验证时, 我们用任务是否会进入 Error 状态来判断任务执行结束的时间是否超过其截止时间. 在 UPPAAL 中用 CTL 公式表示为:

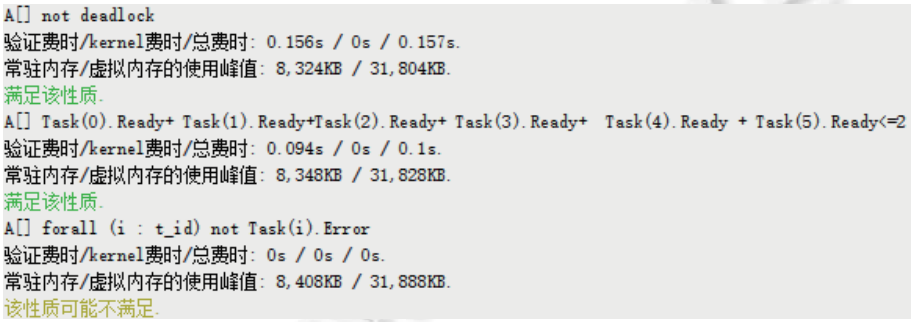
A[] forall (i:t_{id}) not Task(i).Error,

该公式表示所有任务都不会进入 Error 状态.

安全性验证结果如图 8 所示.



(a) 本文模型验证结果



(b) 文献[15]模型验证结果

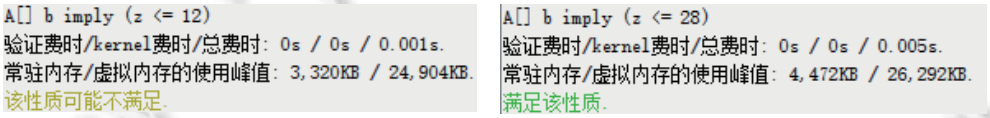
图 8 安全性属性验证结果

从图 8 中可以看出, 我们所提的模型相比于文献 [15] 中的模型, 无论是在验证时间还是在内存使用方面, 都要更好. 但是两种模型在验证属性 3 时出现了不同的结果. 因此我们针对属性 3, 对其进行了分析. 我们在建模时规定任务 A, B, C 的执行周期为 80 ms, 任务 D, E, F 的执行周期为 55 ms, 且每个任务的执行截止时间为 80 ms. 根

据任务之间的依赖关系以及优先级抢占关系来看, 任务 A 在离开执行状态时 time[0] 最多为 6 ms; 任务 B 由于依赖于任务 A, 因此任务 B 在离开执行状态时 time[1] 最多为 16 ms; 任务 C 由于依赖任务 B, E, 因此任务 C 在离开执行状态时 time[2] 最多为 34 ms; 任务 D 由于优先级比任务 C 低, 所以当任务 C 执行时, 任务 D 若是执行的话会被延迟执行, 因此任务 D 在离开执行状态时 time[3] 最多为 12 ms; 任务 E 由于依赖任务 D, 因此任务 E 在离开执行状态时 time[4] 最多为 26 ms; 任务 F 由于依赖于任务 E 且优先级比任务 A, B 低, 所以任务 F 在执行的过程中会被任务 A, B 抢占处理器, 因此任务 F 在离开执行状态时 time[5] 最多为 54 ms. 综上所述每一个任务在离开其执行状态时, time[] 的值都不会超过其截止时间, 所以通过手工推算, 我们可以断定属性 3 是满足的. 因此我们的模型验证结果是对的, 表明了我们模型验证的准确性.

出现上述情况的原因是我们所定义的抢占式优先级时间自动机与时间自动机是不同的. 我们在 TA 的基础上增加了变迁的优先级, 计时时钟变量以及阻塞位置, 使得 PPTA 可以直观表示任务优先级、依赖关系以及抢占关系, 不需要通过变量声明. 而文献 [15] 中利用 TA 来建模上述关系需要在 UPPAAL 中声明相关函数, 优先级数组, 依赖矩阵. 更重要的, 我们是将 PPTA 转换到 UPPAAL 中进行验证的, 我们转换后的模型是在文献 [15] 的模型基础上进行的修改. 首先, 我们转换后的模型要比文献 [15] 的模型状态少, 因为文献 [15] 的模型是单独再使用一个 TA 对静态优先级策略进行建模的, 而我们转换后的模型是将静态优先级策略直接通过函数建模在已有的处理器 TA 上, 使得验证性质的耗时和空间占用量都更少. 其次, 我们对其模型进行了优化, 解决了其模型上一些不符合逻辑的地方, 比如任务与资源可能会出现不匹配的情况, 这就使得我们转换后的模型要比文献 [15] 的模型可靠性更强.

● 有界活性验证属性 4: 该属性我们验证任务 F 是否总是能够在 12 ms 内执行完毕, 如果不能, 修改执行时间参数, 查看任务 F 是否因为被其他任务抢占导致执行时间延长. 图 9 是利用我们提出的模型验证的结果.



(a) 第 1 种情况　　(b) 第 2 种情况

图 9　属性 4 验证结果

第 1 种情况: 由于任务 F 的执行时间为 12 ms, 所以我们要求任务 F 在 12 ms 内执行完成. 在 UPPAAL 中用 CTL 公式表示为:

$$\text{A[] } b \text{ imply } (z \leqslant 12),$$

其中, b 是一个布尔变量, 初始值为 false, z 是一个时钟变量. 当任务 F 到达 Running 状态时, b 被置为 true, z 被重置为 0, 当任务 F 到达 Idle 状态 (表示任务 F 执行完毕) 时, b 被置为 false. 因此, b 的真值表示任务 F 是否能够到达 Idle 状态, z 则记录了任务 F 从 Running 状态到 Idle 状态的时间, 即任务的执行时间.

第 1 种情况的验证结果如图 9(a) 所示. 从图中可以看到, 该属性为可能不满足, 在 UPPAAL 中给出的反例的变量值如图 10 所示. 从图中可以看出, 当 b 的值为 1 (true) 时, z 的值为 15.

第 2 种情况: 我们将公式中的 12 改为 28, 在 UPPAAL 中用 CTL 公式表示为:

$$\text{A[] } b \text{ imply } (z \leqslant 28).$$

第 2 种情况的验证结果如图 9(b) 所示. 从图中可以看到, 该属性为满足.

实际上第 1 种情况是因为任务 A, B, F 共用处理器 c_1, 且任务 A, B 的优先级比任务 F 要高, 所以任务 F 在执行过程中会被任务 A, B 抢占资源, 从而导致任务 F 被阻塞, 到达 Idle 状态的时间会变长, 所以任务 F 不会再 12 ms 执行完毕. 第 2 种情况则是把任务 F 因被抢占资源而中断的时间考虑进来了, 同时我们还可以看到第 2 种情况比第 1 种情况相差的 16 ms 正好是任务 A, B 的执行时间之和.

同样, 我们利用文献 [15] 中的模型验证了第 2 种情况, 验证结果如图 11 所示.

通过图 9(b) 和图 11 可以看出, 关于属性 4 第 2 种情况的验证结果也不一样. 由于任务 F 的优先级比任务 A, B 的优先级都要低, 所以任务 F 在执行时, 是可以被任务 A, B 抢占处理器, 且任务 A, B 的执行时间之和为 16, 所以任务 F 在执行时, 其执行时间就要被延长 16 ms, 因此任务 F 执行完毕的最长时间为 28 ms. 所以经过手工推算, 属性 4 的第 2 种情况是正确的, 因此我们的模型验证结果是对的, 进一步表明了我们模型验证的准确性.

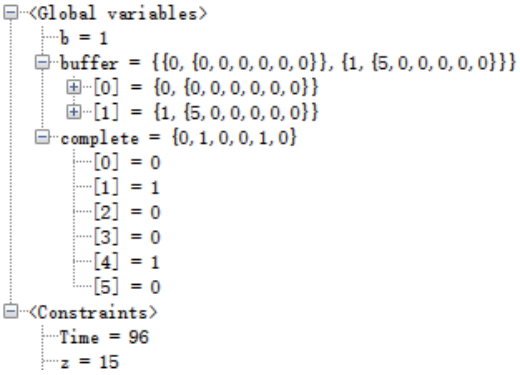


图 10 第 1 种情况反例变量值

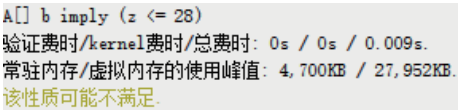


图 11 文献 [15] 模型验证结果

5 相关工作

关于实时系统引入优先级所带来的抢占式调度问题的建模以及验证引起了众多学者的广泛关注并且进行了深入研究, 在模型检测方面主要分为时间 Petri 网模型和时间自动机模型技术.

● 时间 Petri 网模型: 文献 [6] 提出了点区间优先级时间 Petri 网模型, 将变迁发生时间定义为点区间并给变迁添加优先级同时将变迁集分别为可挂起变迁集和不可挂起变迁集, 从而可以模仿优先级抢占式调度问题. 但是在建模时, 没有考虑到任务的截止时间, 系统可能出现不可调度问题且在验证属性时只验证了系统的有界活性, 没有验证系统的安全性. 文献 [10] 提出了优先级时间 Petri 网, 是在时间 Petri 网的基础上添加了变迁的优先级, 即一个变迁在一个状态下是可发生的不仅要满足该变迁具有发生权以及发生时间是在它的静态发生区间内还要满足变迁优先级关系. 但是在该模型中, 当低优先级任务再次获得资源时, 并不是在中断的地方继续执行而是从头开始. 文献 [11−14] 都是在时间 Petri 网的基础上, 通过不同的方式给 Petri 网加入了资源和优先级, 根据任务或者变迁的优先级, 根据不同的策略从而使得优先级高的任务可以抢占优先级低的任务所占用的资源, 导致后者对应的变迁被挂起. 以上利用时间 Petri 网模型针对优先级抢占式调度问题时间属性验证时, 大多是针对模型的可达性提出分析方法, 无自动验证工具支持, 或者自己开发相应的模型检测工具, 可视化编辑能力不如 UPPAAL.

● 时间自动机模型: 文献 [24] 介绍了一种具有优先级扩展的时间自动机模型, 通过偏序关系来定义行为上或自动机上的优先级, 可以为带有优先级的时间系统建模, 但该文献没有给出具体实例来验证所提出来的模型是否可以对优先级实时系统建模. 文献 [15] 中研究了抢占式调度问题, 提出了一个 UPPAAL 建模框架, 该框架可以被实例化以适应各种调度场景, 但是该建模框架在利用优先级抢占调度策略验证多处理器实时系统时, 会出现任务与处理器不匹配的问题, 从而可能会导致最终的验证结果不正确. 文献 [16] 提出了优先级时间自动机模型, 通过在变迁上增加优先级来建模优先级实时系统, 同时提出了两种算法, 包括最优 DBM 减法算法和 DBM 合并算法. 本文借鉴文献 [16] 中引入优先级的做法, 并进一步考虑抢占式调度问题. 由于在实时系统引入优先级之后, 高优先级任务可以抢占低优先级任务所占有的资源, 因此需要进一步分析低优先级任务是否会因为资源被抢占而错过截止时间. 文献 [25] 提出了一种使用时间自动机模拟优先级抢占式调度的替代方法. 首先利用时序图建模每个带有优先级的独立任务, 然后利用转换规则将时序图转换为时间自动机, 进而在 UPPAAL 中进行验证. 本文则是采用抢占式优先级时间自动机来建模具有依赖关系而非独立的任务, 这样建模更符合实时系统抢占式调度的机制. 以上利用时间自动机模型在对优先级抢占式调度问题建模研究上并不完善, 无法保证引入优先级的实时系统的安

全性.

本文在已有的工作基础上, 针对实时系统引入优先级所带来的抢占式调度问题, 提出了 PPTA, 不仅可以直观地表示任务的优先级, 还可以直观地表示任务之间的依赖关系. 又引入了 PPTAN, 可以模拟优先级抢占式调度问题. 之后我们又提出了基于 PPTA 的转换方法, 将其映射到验证工具 UPPAAL 中实现属性的自动验证. 最后通过一个案例, 利用 CTL 描述系统的安全属性以及有界活属性, 在 UPPAAL 中验证, 表明了本文提出的模型可以对实时系统引入优先级所带来的抢占式调度问题进行建模并对其进行准确的验证分析.

6 总 结

本文主要针对引入优先级实时系统所带来的抢占式调度问题进行分析.

(1) 提出了 PPTA, 通过建模相应案例, 表明其可以直观地表示任务的优先级和依赖关系. 同时表明所引入的 PPTAN 可以模拟优先级抢占式调度问题.

(2) 我们提出并应用转换规则和转换算法, 将 PPTA 转换为 UPPAAL 的输入模型 TA, 进而在 UPPAAL 中实现对优先级抢占式调度问题的相关属性的自动验证.

(3) 在应用部分, 我们利用本文提出的模型和文献 [15] 中的 UPPAAL 建模框架对多核多任务实时系统建模并验证该系统的安全性, 有界活性等时间属性. 通过实验结果对比, 我们的模型验证属性的结果更加准确, 说明了我们所提模型的准确性与有效性.

下一步工作主要包括以下两个方面: 一是由于本文所提到的优先级是静态的, 我们将扩展模型, 使其能够建模动态优先级实时系统. 二是我们将根据转换规则和转换算法设计一个转换工具, 实现模型的自动转换.

References:

[1] Sun JH, Guan N, Shi RX, Tan GZ, Yi W. Schedulability analysis for timed automata with tasks. ACM Trans. on Embedded Computing Systems, 2021, 20(5s): 89. [doi: 10.1145/3477020]

[2] Copic M, Leupers R, Ascheid G. Efficient sporadic task handling in parallel AUTOSAR applications using runnable migration. In: Proc. of the 24th Asia and South Pacific Design Automation Conf. Tokyo: Association for Computing Machinery, 2019. 603–608. [doi: 10.1145/3287624.3287654]

[3] Dai XT, Zhao S, Jiang Y, Jiao X, Hu XS, Chang WL. Fixed-priority scheduling and controller co-design for time-sensitive networks. In: Proc. of the 2020 IEEE/ACM Int'l Conf. on Computer Aided Design (ICCAD). San Diego: IEEE, 2020. 1–9.

[4] Wang DG, Wang X, Li ZW. Nonblocking supervisory control of state-tree structures with conditional-preemption matrices. IEEE Trans. on Industrial Informatics, 2020, 16(6): 3744–3756. [doi: 10.1109/TII.2019.2939628]

[5] Liu CY, Zhang LC. Dynamic multi-priority scheduling for cyber-physical systems. Computer Science, 2015, 42(1): 28–32 (in Chinese with English abstract). [doi: 10.11896/j.issn.1002-137X.2015.1.006]

[6] 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]

[7] Wang J, Zhan NJ, Feng XY, Liu ZM. 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]

[8] 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. [doi: 10.1109/32.75415]

[9] Alur R, Dill DL. A theory of timed automata. Theoretical Computer Science, 1994, 126(2): 183–235. [doi: 10.1016/0304-3975(94)90010-8]

[10] Berthomieu B, Peres F, Vernadat F. Model checking bounded prioritized time Petri nets. In: Proc. of the 5th Int'l Symp. on Automated Technology for Verification and Analysis. Tokyo: Springer, 2007. 523–532. [doi: 10.1007/978-3-540-75596-8_37]

[11] 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]

[12] 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]

[13] Roux OH, Lime D. Time Petri nets with inhibitor hyperarcs. Formal semantics and state space computation. In: Proc. of the 25th Int'l

Conf. on Application and Theory of Petri Nets. Bologna: Springer, 2004. 371–390. [doi: 10.1007/978-3-540-27793-4_21]

[14] 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]

[15] David A, Illum J, Larsen KG, Skou A. Model-based framework for schedulability analysis using UPPAAL 4.1. In: Nicolescu G, Mosterman PJ, eds. Model-based Design for Embedded Systems. Boca Raton: CRC Press, 2010. 93–119.

[16] Lin SW, Hsiung PA. Model checking prioritized timed systems. IEEE Trans. on Computers, 2012, 61(6): 843–856. [doi: 10.1109/TC.2011.99]

[17] Behrmann G, David A, Larsen KG. A tutorial on UPPAAL. In: Proc. of the 2004 Formal Methods for the Design of Real-time Systems: Int'l School on Formal Methods for the Design of Computer, Communication, and Software Systems. Bertinoro: Springer, 2004. 200–236. [doi: 10.1007/978-3-540-30080-9_7]

[18] Clarke EM, Emerson EA, Sistla AP. Automatic verification of finite state concurrent system using temporal logic specifications: A practical approach. In: Proc. of the 10th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages. Austin: Association for Computing Machinery, 1983. 117–126. [doi: 10.1145/567067.567080]

[19] Zhang GQ. Introduction to Formal Methods. Beijing: Tsinghua University Press, 2015 (in Chinese).

[20] Chen XY, Zhu Y, Zhao Y, Wang JY. Hybrid AADL modeling and model transformation for CPS time and space properties verification. Ruan Jian Xue Bao/Journal of Software, 2021, 32(6): 1779–1798 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6249.htm [doi: 10.13328/j.cnki.jos.006249]

[21] Clarke EM, Emerson EA, Sifakis J. Model checking: Algorithmic verification and debugging. Communications of the ACM, 2009, 52(11): 74–84. [doi: 10.1145/1592761.1592781]

[22] Wang SL, Zhan BH, Sheng HH, Wu H, Yi SC, Wang LT, Jin XY, Xue B, Li JH, Xiang SQ, Xiang Z, Mao BF. Survey on requirements classification and formalization of trustworthy systems. Ruan Jian Xue Bao/Journal of Software, 2022, 33(7): 2367–2410 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6587.htm [doi: 10.13328/j.cnki.jos.006587]

[23] Dai SX, Hong M, Guo B, Yang QH, Huang W, Xu BP. Schedulability analysis model for multiprocessor real-time systems using UPPAAL. Ruan Jian Xue Bao/Journal of Software, 2015, 26(2): 279–296 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4781.htm [doi: 10.13328/j.cnki.jos.004781]

[24] Zhao XH, Zhuang L. Improvement on a DBM subtraction algorithm in timed automata with priorities. Computer Engineering & Science, 2008, 30(11): 56–59 (in Chinese with English abstract). [doi: 10.3969/j.issn.1007-130X.2008.11.018]

[25] Pimkote A, Vatanawood W. Simulation of preemptive scheduling of the independent tasks using timed automata. In: Proc. of the 10th Int'l Conf. on Software and Computer Applications. Kuala Lumpur: Association for Computing Machinery, 2021. 7–13. [doi: 10.1145/3457784.3457786]

附中文参考文献:

[5] 刘纯尧, 张立臣. 信息物理融合系统的动态多优先级调度. 计算机科学, 2015, 42(1): 28–32. [doi: 10.11896/j.issn.1002-137X.2015.1.006]

[6] 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]

[7] 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]

[19] 张广泉. 形式化方法导论. 北京: 清华大学出版社, 2015.

[20] 陈小颖, 祝义, 赵宇, 王金永. 面向CPS时空性质验证的混成AADL建模与模型转换方法. 软件学报, 2021, 32(6): 1779–1798. http://www.jos.org.cn/1000-9825/6249.htm [doi: 10.13328/j.cnki.jos.006249]

[22] 王淑灵, 詹博华, 盛欢欢, 吴昊, 易士程, 王令泰, 金翔宇, 薛白, 李静辉, 向霜晴, 向展, 毛碧飞. 可信系统性质的分类和形式化研究综述. 软件学报, 2022, 33(7): 2367–2410. http://www.jos.org.cn/1000-9825/6587.htm [doi: 10.13328/j.cnki.jos.006587]

[23] 代声馨, 洪玫, 郭兵, 杨秋辉, 黄蔚, 徐保平. 多处理器实时系统可调度性分析的UPPAAL模型. 软件学报, 2015, 26 (2): 279–296. http://www.jos.org.cn/1000-9825/4781.htm [doi: 10.13328/j.cnki.jos.004781]

[24] 赵旭辉, 庄雷. 一种基于优先级扩展的时间自动机模型中DBM减法算法的改进. 计算机工程与科学, 2008, 30(11): 56–59. [doi: 10.3969/j.issn.1007-130X.2008.11.018]

左正康(1980－), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为形式化方法, 智能化软件.

赵帅(1999－), 男, 硕士生, 主要研究领域为模型检测, 形式化方法.

王昌晶(1977－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为高可信软件, 智能化软件.

谢武平(1984－), 男, 博士, 讲师, CCF 专业会员, 主要研究领域为形式化方法, 可信软件.

黄箐(1984－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为智能化软件.