基于时间多栈下推网络的实时系统验证

钱俊彦[1)] 甘鹏程[1)] 郭云川[2)] 赵岭忠[1)] 古天龙[1)]

[1)](桂林电子科技大学广西可信软件重点实验室 广西 桂林 541004)

[2)](中国科学院信息工程研究所 北京 100093)

摘 要 多栈下推网络(MPDN)是利用多个栈来描述并发递归程序线程之间交互的一种下推系统模型.为了描述基于线程之间交互的实时并发递归程序,首先将描述连续时间的时钟引入到 MPDN,提出了时间多栈下推网络(TMPDN),并给出了语法及其操作语义;其次利用基于时间关键点的时钟等价优化技术,缩减等价域状态空间;然后通过静态转换方法,获得所有的可达域状态格局,将连续时间的 TMPDN 模型转换成离散的 MPDN 模型.在此基础上,基于 on-the-fly 技术,采用动态转换方法,仅关心栈顶域状态转换,进一步缩减转换后的状态空间.同时证明了状态 q_F 在 TMPDN 可达当且仅当其转换状态 q'_F 在 MPDN 可达,并给出了模型转换算法;最后可采用现有模型检验工具验证转换后的 MPDN.

关键词 MPDN;TMPDN;并发递归程序;时钟等价;可达性

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2016.02253

Verification of Real-Time Systems with Time Multi-Pushdown Net

QIAN Jun-Yan[1)] GAN Peng-Cheng[1)] GUO Yun-Chuan[2)] ZHAO Ling-Zhong[1)] GU Tian-Long[1)]

[1)](*Guangxi Key Laboratory of Trusted Software*, *Guilin University of Electronic Technology*, *Guilin*, *Guangxi* 541004)

[2)](*Institute of Information Engineering*, *Chinese Academy of Sciences*, *Beijing* 100093)

Abstract Multi-Pushdown Net (MPDN) is a model of pushdown systems that makes use of multiple stacks to describe the interaction between threads of concurrent recursive programs. In order to model real-time concurrent recursive procedures based on thread interaction, a new model, i. e. time multiple stack pushdown network (TMPDN), is proposed by introducing a clock into MPDN to describe the continuous time. We give the syntax and operational semantics of TMPDN. Based on this definition, an optimization technique of clock domain equivalence that makes use of key time points is proposed to reduce the space of equivalence domains. Then static conversion methods are presented to obtain all reachable domains and based on which to convert a continuous time TMPDN into a discrete MPDN. Next a dynamic conversion method, the so-called on-the-fly technique, is used to further reduce the state space obtained by static methods, and in this process it is concerned only with the state transformation of the stack-top domains. Corresponding model transformation algorithms are given in this paper. It is proved that the state q_F in TMPDN can be reached if and only if the transition state q'_F in the corresponding MPDN can

收稿日期:2015-02-11;在线出版日期:2016-02-26.本课题得到国家自然科学基金(61262008,61562015,61572146,U1501252)、广西自然科学基金(2012GXNSFAA053220,2014GXNSFAA118365,2015GXNSFDA139038)、广西高等学校高水平创新团队及卓越学者计划、广西可信软件重点实验室重点基金、桂林电子科技大学创新团队资助.**钱俊彦**,男,1973 年生,博士,教授,中国计算机学会(CCF)高级会员,主要研究领域为软件工程、模型检验和程序验证. E-mail: qjy2000@gmail.com. **甘鹏程**,男,1983 年生,博士研究生,主要研究方向为软件工程、程序分析与验证. **郭云川**,男,1977 年生,博士,主要研究方向为安全评估与验证. **赵岭忠**(通信作者),男,1977 年生,博士,教授,主要研究领域为形式化技术与软件验证. E-mail: zhaolingzhong163@163.com. **古天龙**,男,1964 年生,博士,教授,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为形式化方法、符号计算.

be reached. The MPDN obtained by the above transformation can be verified using existing model checkers.

Keywords MPDN; TMPDN; concurrent recursive programs; clock equivalence; reachability

1 引 言

随着多核处理器技术日益发展,并发软件已渗透到国民经济和国防建设的各个领域,然而在某些安全攸关的领域,诸如交通控制和航空航天等领域,软件上的微小错误可能导致重大事故,甚至危及人身安全.如何提高软件系统的可靠性和安全性已成为当前紧迫问题.对于一些安全攸关的实时系统来说,其安全性要求更高,需充分验证软件系统的某些关键性质,这种背景下,对其形式化分析验证显得尤为重要.

目前,诸多学者已致力于实时系统分析与验证的研究,为了描述不同类型的实时系统,提出了时间自动机、时间下推自动机、时间递归状态自动机、嵌套时间自动机等理论模型.具体而言,1994 年 Alur 和 Dill[1]在自动机的基础上,引入描述连续时间的时钟,提出了时间自动机,并采用时钟等价技术实现模型检验时间自动机[2].为了解决含有递归的实时系统建模,Abdulla 等人[3]提出时间下推自动机,并通过时间离散化,将其转换为下推自动机后进行分析,并在文献[4-5]中加入权值,求解最小时间开销的可达性问题;2013 年[6]将时间下推自动机的研究对象扩展到无限迁移的实时系统上,同时证明了该情况下的实时系统可达性分析为 EXPTIME 完全问题;2014 年 Cai 和 Ogawa[7]给出了一个时间下推自动机的实例;Trivedi 和 Wojtczak[8]和 Benerecetti 等人[9]提出了时间递归状态自动机,通过相互调用来实现递归描述,调用过程中,时钟通过值或者引用的方式传递;2013 年 Li 等人[10]提出了嵌套时间自动机(NeTAs),其整体是一个下推自动机,栈的每一层为时间自动机,利用嵌套的思想来解决实时系统中的递归问题.上述模型能描述实时系统的并发及递归问题,但是无法描述实时并发系统中线程间交互的情况.

另外,Atig 等人[11-12]给出的序列多栈下推自动机(OMPA),其在栈上增加了线性序列描述线程间交互,并证明了 OMPA 的可达性为 2ETIME 完全问题;Torre 等人[13-14]给出基于上下文限界的多栈下推系统(SMPDS),并分析及论证了系统可达性为 PSPACE 完全问题.这一类模型能描述并发递归程序的线程间交互,但不能描述实时性.

时间下推自动机相当于在时间自动机上增加了栈,能够有效地解决时间自动机难于描述递归的情况,其缺点是无法很好的描述线程间的交互.为了能对实时并发系统中的递归机制及线程间交互进行建模,本文结合时间下推自动机与多栈下推网络的优点,类似于在时间下推自动机中增加了栈的数目,通过栈之间交互来模拟并发程序运行时线程间操作,提出时间多栈下推网络(TMPDN),并利用多栈下推系统理论来验证实时并发递归程序中的可达性.

形式化验证实时系统将可能产生状态空间爆炸问题,从而给验证带来极大困难.当前验证实时系统的主要思想:将实时系统的连续模型转换为不含时间约束的离散模型.离散模型的状态用等价域描述,域等价的基本思想都是把状态位置和时钟区间结合,形成一个新的状态,主要方法是域图(Region Graph)和带图(Zone Graph)[15].由于带图生成的符号状态图具有不稳定性,产生的符号状态迁移序列存在不确定性,故本文将采用域图方法.通过基于时钟关键点的时钟域优化等价技术,将 TMPDN 转换为对并发程序研究较为成熟的 MPDN 模型.

通常时钟域等价使用实数取整、小数部分大小比较来划分时间域.为了获得更少的时钟域,本文采用时钟关键点技术,对时钟域划分进行优化,从而缩减转换后的模型状态空间.为了进一步缩减转换后的 MPDN 状态空间,在静态转换的基础上,采用 on-the-fly 技术,给出了仅关心栈顶及其下一层的动态转换方法,状态空间仅于栈顶及其下一层的项集相关,而与整个项集无关,从而大幅度缩减转换后的状态空间.

本文第 2 节介绍相关基础知识;第 3 节提出 TMPDN 模型;第 4 节从静态和动态两个角度分析 TMPDN 转换为 MPDN 的转换过程;第 5 节证明 TMPDN 与转换后的 MPDN 可达性等价;第 6 节给出 TMPDN 转换为 MPDN 的具体算法;第 7 节为 TMPDN 的上下文限界验证;第 8 节给出一个具体实例分析;第 9 节为全文总结.

2　基础知识

MPDN 模型是一种描述并发递归程序的下推系统，利用栈之间的操作来模拟程序中线程间的交互. MPDN 是一个五元组 $M=(n,Q,q_0,\Gamma,\Delta)$，其中 $n\in\mathbb{N}$ 表示系统栈的数量，Q 表示有限状态集，$q_0\in Q$ 表示初始状态，Γ 表示有限栈内字符集，$\Delta=\Delta_i^{int}\cup\Delta_i^{push}\cup\Delta_i^{pop}$ 表示转换关系，其中 i 表示第 i 个栈，$0<i\leq n$，$\Delta_i^{int}\subseteq(Q\times Q)$ 描述第 i 个栈的内部转换，Δ_i^{push}，$\Delta_i^{pop}\subseteq(Q\times\Gamma\times Q)$ 表示第 i 个栈的压栈和出栈. 为了描述方便，文中用 $[n]$ 表示 1 到 n 的集合.

MPDN 格局 $C=\langle Work[n],q,\{\omega_i\}_{i\in[n]}\rangle$，其中 $Work[n]\subseteq[n]$ 为 $[n]$ 的子集，表示当前工作栈集，$q\in Q$ 表示状态，$\omega_i\in(\Gamma)^*$ 表示第 i 个栈的栈内字符集，$C_0=\langle Work[n],q_0,\{\omega_i\}_{i\in[n]}\rangle$ 表示 MPDN 的初始格局. 第 i 个栈的格局转换描述 $\langle Work[n],q,\{\omega_i\}_{i\in[n]}\rangle\xrightarrow{\delta}\langle Work[n],q',\{\omega_i'\}_{i\in[n]}\rangle$，其中迁移动作 δ 包括空操作 nop、压栈操作 $push(\gamma)$ 和出栈操作 $pop(\gamma)$，格局转换成立当且仅当满足以下条件：

(1) $\Delta=\Delta_i^{int}:op=nop,(q,q')\in\Delta_i^{int},\omega_i=\omega_i',i\in[n]$，表示内部转换，描述状态迁移而栈内字符未发生改变.

(2) $\Delta=\Delta_i^{push}:op=push(\gamma),(q,\gamma,q')\in\Delta_i^{push},\omega_i'=\gamma.\ \omega_i,\omega_j=\omega_j',j\in[n\backslash i]$，表示当前栈 i 压栈时，其栈内字符新增栈符 γ，其他栈的栈内字符未变化.

(3) $\Delta=\Delta_i^{pop}:op=pop(\gamma),(q,\gamma,q')\in\Delta_i^{pop},\omega_i=\gamma.\ \omega_i',\omega_j=\omega_j',j\in[n\backslash i]$，表示当前栈 i 出栈时，其栈内字符删除了栈符 γ，其他栈的栈内字符未变化.

一个上下文定义为在单一线程中的计算或变迁序列. 栈 i 的一个上下文可表示成 $\Pi=C_0\,t_1\,C_1\,t_2\cdots t_mC_m$，其中 $t_1\cdots t_m\in\Delta_i$. 计算从一个线程切换至另一个线程，定义为上下文切换. 用 Π_i 和 Π_j 分别表示栈 i,j 上的上下文，两者发生上下文切换用 $\Pi_i\cdot\Pi_j$ 表示. 当上下文切换限制为 k 时，称之为上下文 k 限界.

3　时间多栈下推网络(TMPDN)

为了描述实时系统中并发递归机制以及线程之间的交互，在 MPDN 的基础上引入了时钟，提出了 TMPDN 模型. 时钟是在正实数范围内取值的变量.

定义 1.　给定时钟集 T 以及时钟 $t_1,t_2\in T$，则在 T 上的时钟约束定义为 $\sigma:=t_1\prec c\,|\,t_1-t_2\prec c\,|\,\neg\sigma\,|\,\sigma\wedge\sigma$，其中 $c\in\mathbb{R},\prec\in\{<,\leq\}$.

时钟解释 $v:T\to\mathbb{R}^+$，表示时钟 $t\in T$ 在当前的取值. 假设实数 $c\in\mathbb{R}$，$v(t)+c$ 表示解释后的时钟值加 c，在没有歧义的情况下，用 $v+c$ 表示.

定义 2.　TMPDN 是一个六元组 $M_T=(n,Q,q_0,\Gamma,T,\Delta)$，其中 n 表示系统中栈的个数；Q 表示有限状态集；q_0 表示初始状态；Γ 表示有限栈符集；$T=T_G\cup T_L$ 表示有限时钟集，其中 T_G 表示全局时钟，用于标识实时并发系统中全局变量或事务的时间，T_L 表示栈内局部时钟，用于标识系统中局部变量或子事务的时间；迁移关系 $\Delta\subseteq 2^{[n]}\times Q\times T_G\times(\Gamma\times T_L)^*\times 2^{[n]}\times Q\times T_G\times(\Gamma\times T_L)^*$ 为描述系统格局的迁移.

TMPDN 的格局 $C=\langle Work[n],q,\{w_i\}_{i\in[n]},v\rangle$，其中 $q\in Q$，$Work[n]\subseteq[n]$ 为 $[n]$ 子集，表示当前工作栈集，w_i 为栈 i 的字，表示栈 i 的内容，描述为 $w_i\in(\Gamma\times\mathbb{R}^+)^*$，$v$ 表示时钟当前取值.

TMPDN 作为实时并发程序的模型，能描述多个栈同时产生迁移，迁移关系 Δ 分为栈内迁移 Δ^{in}、栈间切换 Δ^{inter} 和并发执行 $\Delta^{\|}$. 为了表述方便，首先讨论一个栈 i 内迁移的执行，用形式 $\langle\{i\},q,\{w_i\}_{i\in[n]},v\rangle\xrightarrow{op_{in}}\langle\{i\},q',\{w_i'\}_{i\in[n]},v'\rangle$ 表示，其中 op_{in} 为栈内迁移的动作集，包括空操作 nop、时间约束判断 $t\in I?$、时钟重置 $t\leftarrow I$、时间流逝 $Time\leftarrow c$、压栈操作 $push(a,I)$ 和出栈操作 $pop(a,I)$，其中 I 表示时钟取值范围，$t\in T$，故栈内迁移关系 Δ^{in} 可表示为 $\Delta^{nop}\cup\Delta^{?}\cup\Delta^{=}\cup\Delta^{\vdash}\cup\Delta^{push}\cup\Delta^{pop}$，其中 Δ^{nop}，$\Delta^{?}$，$\Delta^{=}$，$\Delta^{\vdash}$，Δ^{push}，Δ^{pop} 分别表示上述操作的迁移. 下文根据不同的栈内迁移动作 op_{in} 给出其执行含义.

(1) $\Delta^{in}=\Delta^{nop}:op_{in}=nop,w_i'=w_i,v'=v$，表示格局内元素未发生变化.

(2) $\Delta^{in}=\Delta^{?}:op_{in}=t\in I?,w_i'=w_i,v'=v,v(t)\in I$，表示当 t 的时钟值在 I 范围内时，执行该操作，格局内元素未发生变化.

(3) $\Delta^{in}=\Delta^{=}:op_{in}=t\leftarrow I,w_i'=w_i,v'=v[t\leftarrow c]$，$c\in I$，表示给时钟 t 指定 I 范围内的任意值 c，其他格局内元素未发生变化.

(4) $\Delta^{in}=\Delta^{\vdash}:op_{in}=Time\leftarrow c,w_i'=w_i^{+c}$，假设 $w_i=\langle a_1,v_1\rangle\langle a_2,v_2\rangle\cdots\langle a_n,v_n\rangle$，那么 $w_i^{+c}=\langle a_1,v_1+c\rangle\langle a_2,v_2+c\rangle\cdots\langle a_n,v_n+c\rangle$，$v'=v+c$，表示格局内所有时钟增加 c，格局内非时钟内容未发生变化.

(5) $\Delta^{in}=\Delta^{push}:op_{in}=push(a,I),w_i'=w_i\cdot\langle a,c\rangle$，

$c\in I, v'=v$,表示将变量 a 压入栈顶,并设定相应时钟为 c,其时钟值为 I 范围内的任意值.

(6) $\Delta^{in}=\Delta^{pop}: op_{in}=pop(a,I), w_i=w_i'\langle a,c\rangle$, $c\in I, v'=v$. 表示将栈顶内时钟值为 I 范围的变量 a 弹出.

其次描述栈间切换 Δ^{inter}. 假设栈间切换操作 $op_{inter}=i\triangleright j$ 表示栈 i 切换到栈 j,表示形式为 $\langle\{i\},q,\{w_i\}_{i\in[n]},v\rangle\xrightarrow{op_{inter}}\langle\{j\},q',\{w_j\}_{j\in[n]},v'\rangle$,其中 $q,q'\in Q; v'=v; w_i,w_j$ 分别表示栈 i,j 的字,$i,j\in[n]$且 $i\neq j$. 上下文切换时,将栈 j 从等待栈切换到工作栈,同时将栈 i 从工作栈切换到等待栈,则切换后的工作栈 $Work[n]'=Work[n]\backslash\{i\}\cup\{j\}$.

最后描述并发操作 $\Delta^{\|}$. 假设 $op=op_{in}\cup op_{inter}$ 为栈迁移动作集,并发执行用形式 $\langle Work[n],q,\{w_i\}_{i\in[n]},v\rangle\xrightarrow{(op_i\,\|\,op_j)_{i,j\in[n],\text{and}\,i\neq j}}\langle Work[n]',q',\{w_i\}_{i\in[n]},v'\rangle$ 表示,$Work[n]$ 为当前工作栈集,如果不发生栈间切换,则 $Work[n]'=Work[n]$,如果发生栈间切换 $i\triangleright j$,则 $Work[n]'=Work[n]\backslash\{i\}\cup\{j\}$. 其他类似一个栈的情况.

为了进一步理解栈的迁移关系,通过实例描述入栈、出栈、时间重置、时间流逝和并发的动作,如图 1 所示. 假定有 n 个栈,全局时钟 t_1,全局变量 g,其相关联时钟为 t_g,$\{a_1,b_1,a_i,b_i,c_i,a_j,b_j,c_j\}\subseteq\Gamma$,其中 $i,j\in[n]$,时钟区间分别表示为 $[v_i:v_j]$,$[v_i:v_j)$,$[v_i:\infty)$,其中 $[v_i:v_j]$ 表示 v_i 到 v_j 区间内(包含 v_j)的所有时钟值,$[v_i:v_j)$ 表示 v_i 到 v_j 区间内(不包含 v_j)的所有时钟值,$[v_i:\infty)$ 表示 v_i 到无限大 ∞ 的所有时钟值.

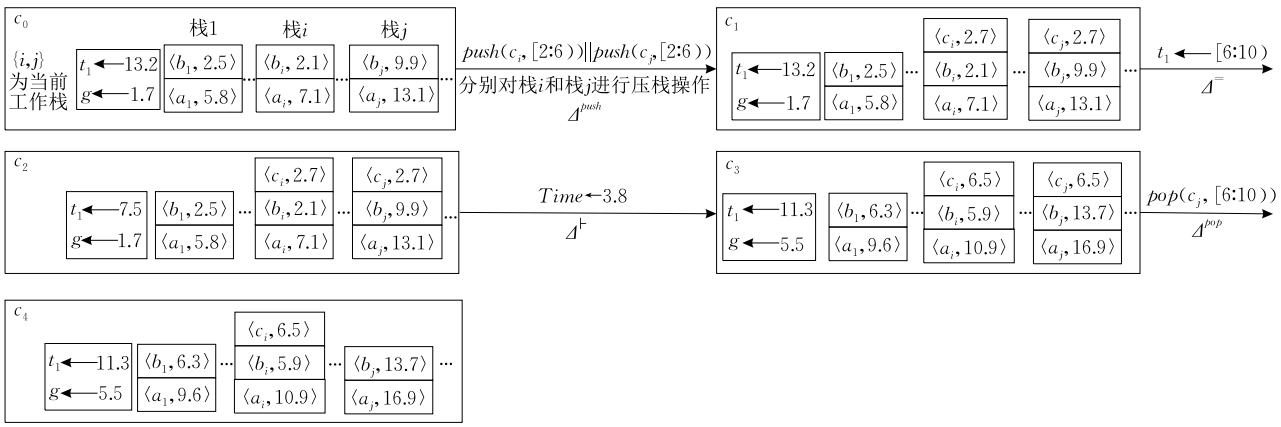


图 1 TMPDN 中栈内操作下的格局迁移

(1)并发操作. 从格局 C_0 到格局 C_1,将 c_i 压入栈 i,且时钟赋值为 $2.7\in[2:6)$,同时将 c_j 压入栈 j,且时钟赋值为 $2.7\in[2:6)$.

(2)时钟重置. 从格局 C_1 到格局 C_2,时钟 t_1 的时钟值重置为 $7.5\in[6:10)$.

(3)时间流逝. 从格局 C_2 到格局 C_3,栈内所有全局时钟、全局变量相关联的时钟、局部变量相关联的时钟,其时钟值都增加 3.8 个时间单位.

(4)出栈操作. 从格局 C_3 到格局 C_4,在出栈操作前,检查 c_j 值是否属于区间 $[6:10)$,由于 $c_j=6.5$ 满足条件,则执行出栈操作.

4 TMPDN 的模型转换分析

因 TMPDN 的时钟取值为正实数域,故在求解可达状态格局时,可能会出现状态空间爆炸问题,从而使可达性分析变得异常困难. 为了分析 TMPDN 的可达性,本文利用时钟等价技术将无限状态格局抽象转换为有限状态格局,也就是说将 TMPDN $M_T=(n,Q,q_0,\Gamma,T,\Delta)$ 转换为 MPDN $M_M=(n^M,Q^M,q_0^M,\Gamma^M,\Delta^M)$.

4.1 域

把 TMPDN 转换为 MPDN,最主要的问题是把 TMPDN 中的栈符以及描述栈符的时钟抽象为 MPDN 的栈符集. 为了描述转换后的栈符集,需把 TMPDN 的时钟和栈符转换为域,转换后的域用符号 R 表示. 域 R 是由时钟关键点和项集组成,首先定义时钟关键点及时钟等价,然后定义项集.

为了把描述连续时间的 TMPDN 转换为离散的 MPDN,采用离散的时钟区间描述连续时间. Alur 和 Dill[1] 在 1994 年提出时钟等价技术,采用连续正整数作为时钟区间划分. 因实时系统动作迁移及事件通常在关键的时间点触发,时钟关键点是非连续正整数,与原时钟等价技术相比,甚至减少一、

两个数量级，并在多个时钟的情况下，划分的等价区域将减少几个数量级. 基于此，本文给出了优化的时钟等价技术，依据时钟关键点作为时钟区间划分的界限，减少时钟区间划分区域，从而达到缩减等价后的状态空间.

时钟关键点为系统动作迁移及其他事件的时间触发点，用 k_i 表示 M_T 时钟上的关键点，$k_{\max}$ 表示时钟关键点上限阈值，$k_\infty > k_{\max}$ 表示所有大于 $k_{\max}$ 的值，时钟关键点集合可描述为 $key=\{0,k_1,\cdots,k_i,\cdots,k_{\max},k_\infty\}$.

为了描述时钟等价，实数时钟值 $t=\lfloor t\rfloor+re(t)$ 分成两个部分，关键点部分 $\lfloor t\rfloor$：实数时钟值向下取最近关键点的值；剩余部分 $re(t)$：$re(t)=t-\lfloor t\rfloor$.

给定时钟集 T，假设有任意两个时钟 $t_i,t_j\in T$，并且存在关键点 $key=\{0,k_1,\cdots,k_i,\cdots,k_{\max},k_\infty\}$. 当满足如下规则的时钟值，为等价时钟：

(1) $v(t_i)>k_{\max}$ 当且仅当 $\lfloor t_i\rfloor=k_\infty$，即 t_i 的时钟值大于关键点上限阈值时，t_i 取无穷大 k_∞.

(2) $k_i\leqslant v(t_i)<k_{i+1}$ 当且仅当 $\lfloor t_i\rfloor=k_i$，即 t_i 的时钟值小于关键点 k_{i+1} 且大于等于 k_i 时，t_i 取关键点 k_i.

(3) 假定 $\lfloor t_i\rfloor=k_i$，$\lfloor t_j\rfloor=k_j$，$v(t_i)-k_i<v(t_j)-k_j$ 当且仅当 $re(t_i)<re(t_j)$，即 t_i 取关键点 k_i，t_j 取关键点 k_j，当 t_i 的时钟值与关键点 k_i 的差值小于 t_j 的时钟值与关键点 k_j 的差值时，记为 $re(t_i)<re(t_j)$.

假设时钟集 T，有任意两个时钟 $t_i,t_j\in T$ 和 $key=\{0,k_1,k_2,\cdots,k_\infty\}$，在 $(0<t_i<k_1,k_1<t_j<k_2)$ 区域内，基于时钟关键点的时钟域等价规则，在 $\lfloor t_i\rfloor=0$，$\lfloor t_j\rfloor=k_1$，且 $re(t_i)<re(t_j)$ 这区间内的时钟值为等价时钟，如图 2 所示的阴影部分.

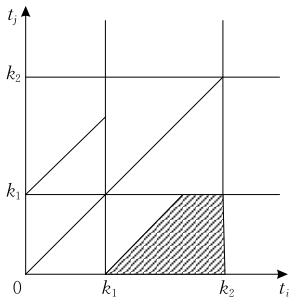


图 2　t_i，t_j 时钟等价区间

下面描述域 R 的另一部分——项集. 普通项集 Y 包含：(1) 时钟 T，T 用来描述 M_T 的全局时钟；(2) 栈字符 Γ，Γ 用来描述 M_T 的栈顶字符；(3) 基础时钟字符 $\vdash$，用来描述栈的基础时钟，是栈内域中其他项的时间参照点，初始值为 0；故普通项集 Y 描述为 $Y=T\cup\Gamma\cup\{\vdash\}$. 标识项集 Y' 表示普通项 Y 的时间流逝，在普通项上添加上标阴影 $'$ 的方式表示，描述为 $Y'=T'\cup\Gamma'\cup\{\vdash'\}$，其中 $T'=\{t'\mid t\in T\}$，$\Gamma'=\{a'\mid a\in\Gamma\}$，$\vdash'$ 描述栈字符出栈时的时间流逝，主要用于动态转换，当时间流逝时，非栈顶域中的时钟无法随时间流逝进行更新，故在栈顶域中增加 $\vdash'$ 以记录时间的流逝，当栈顶域进行出栈操作时，对栈顶下一层域中的时钟进行同步更新.

TMPDN 模型转换为 MPDN 可采用两种方法：静态转换和动态转换. 使用静态转换时，项集 Z 为普通项集 Y，即 $Z:=Y$. 当采用动态转换时，项集 Z 为普通项集 Y 和标识项集 Y' 的并集，即 $Z:=Y\cup Y'$. 假定栈 i，栈深度为 d，域 $R_i=R_{i1}\cdots R_{ij}\cdots R_{id}\in(2^{Z\times key})^+$，其中 $i\in[n]$，$j\in[d]$，$R_{ij}=\langle z_j,k_j\rangle$，$z_j\in Z$，$k_j\in key$，从而可得域 $R=\{R_i\}_{i\in[n]}$. 域 R_i 存在，当且仅当域 R_i 中包含以下 3 类：

(1) $\{|\langle\Gamma,key\rangle\cap R_i|=1\}\wedge\{|\langle\Gamma',key\rangle\cap R_i|=1\}$，表示域 R_i 与普通栈符和标识栈符的交集非空，换而言之，域 R_i 中一定存在普通栈符和标识栈符；

(2) $\{|\langle\vdash,key\rangle\cap R_i|=1\}\wedge\{|\langle\vdash',key\rangle\cap R_i|=1\}$，表示域 R_i 与 $\vdash$ 和 $\vdash'$ 的交集非空，换而言之，域 R_i 中一定存在 $\vdash$ 和 $\vdash'$；

(3) $\{|\langle T,key\rangle\cap R_i|=1\}\wedge\{|\langle T',key\rangle\cap R_i|=1\}$，表示域 R_i 与时钟字符和时钟标识符的交集非空，换而言之，域 R_i 中一定存在时钟字符和时钟标识符.

4.2　TMPDN 转换为 MPDN

为了描述 TMPDN 模型转换为 MPDN，通过上节给出的时钟等价优化技术，把 M_T 栈符及记录栈符的时钟等价为域，表示 M_M 的栈符. TMPDN 与 MPDN 的转换分为静态转换和动态转换两种，其中静态转换是把 TMPDN 模型整体抽象转换为 MPDN 模型，获得所有可达的域状态格局，在每个抽象状态格局中，栈的每一层都需抽象域描述，故构造过程，其状态空间及时间耗费都较为庞大；而动态转换采用 on-the-fly 技术，无需整体计算抽象 MPDN 模型，而是依据 TMPDN 模型的状态迁移过程，动态计算与其等价的 MPDN 域状态，并且只关注栈顶层及其下一层，不关心整个栈内容，因此降低了转换状态空间规模及时间耗费. 由于动态转换仅对栈顶域操作，故需增加基础时钟字符和标识项，在非栈顶域成为栈顶域时对时钟值正确更新，跟静态转换相比，增加了额外更新操作. M_M 的栈间切换 Δ^{inter} 与 M_T 类似. M_T 的并发操作 $\Delta^{\|}$ 转换时，工作栈集 $Work[n]$ 内的每个栈，调用栈内迁移转换.

4.2.1 静态转换

假定模型 TMPDN $M_T=(\lceil n\rceil,Q,q_0,\Gamma,T,\Delta)$，通过静态转换获得的模型 MPDN $M_M=(\lceil n\rceil^M,Q^M,q_0^M,\Gamma^M,\Delta^M)$，具体转换如下：

(1) 栈个数$\lceil n\rceil^M$的转换. $\lceil n\rceil^M=\lceil n\rceil$，即 M_M的栈数量与 M_T的栈数量相等.

(2) 状态 Q^M的转换. $Q^M=Q$，即 M_T的状态集与 M_M的状态集相同.

(3) 初始状态 q_0^M的转换. $q_0^M=q_0$，即 M_M的初始状态与 M_T的初始状态相同.

(4) 栈字符 Γ^M的转换. $\Gamma^M=2^{Z\times key}$，其中 Z 为普通项集 Y，即 $Z:=Y$，key 为时钟关键点集.

(5) 迁移关系 Δ 到Δ^M的转换规则.

M_T的格局迁移为$\langle\{i\},q,\{w_i\}_{i\in\lceil n\rceil},v\rangle\xrightarrow{op}\langle\{i\},q',\{w_i'\}_{i\in\lceil n\rceil},v'\rangle\in\Delta,w_i\in(\Gamma\times\mathbb{R}^+)^*$称为栈 i 的字，设 d_i 为栈 i 的栈深度，用 w_{ij} 表示栈 i 内第 j 层的子字，$j\in\lceil d_i\rceil$，$w_{ij}|_\Gamma$表示 w_{ij} 投影在 Γ 的栈字符，$v(w_{ij}|_\Gamma)$表示投影与栈字符 Γ 相关联的时钟值，$\lfloor v(w_{ij}|_\Gamma)\rfloor$表示投影与栈字符 Γ 相关联的时钟值在时钟域等价后的关键点，T_G为全局时钟，$\lfloor v(T_G)\rfloor$表示全局时钟的时钟值在时钟域等价后的关键点，则可得 w_{ij}在 M_M中等价项 $R_{ij}=((w_{ij}|_\Gamma)\times\lfloor v(w_{ij}|_\Gamma)\rfloor)\cup(T_G\times\lfloor v(T_G)\rfloor)\in\Gamma^M$，栈 i 内容 $R_i=R_{i1}\cdots R_{ij}\cdots R_{id_i}$，其中 $i\in\lceil n\rceil,j\in\lceil d_i\rceil$. 从而可知对应于 M_M的格局迁移为$\langle\{i\},q,\{R_i\}_{i\in\lceil n\rceil}\rangle\xrightarrow{\delta}\langle\{i\},q',\{R_i'\}_{i\in\lceil n\rceil}\rangle$，其中 $q,q'\in Q^M$，δ 为迁移动作. 如图 3 所示.

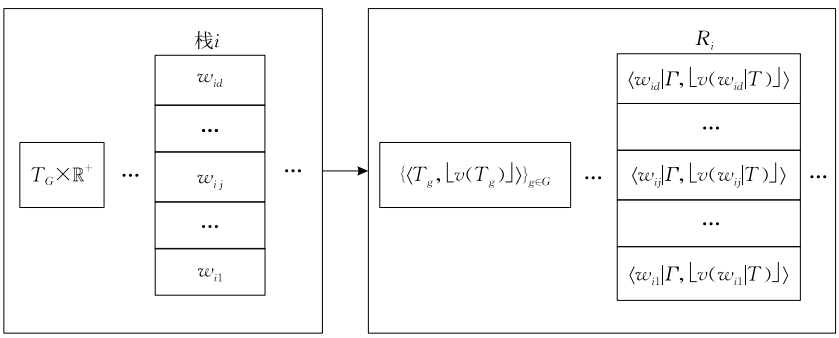


图 3 M_T的格局静态转换为 M_M的格局

(1) $\Delta=\Delta^{nop}$时，$op=nop$，如果$\langle\{i\},q,\{w_i\}_{i\in\lceil n\rceil},v\rangle\xrightarrow{nop}\langle\{i\},q',\{w_i'\}_{i\in\lceil n\rceil},v'\rangle\in\Delta$，那么在 M_M中有$\langle\{i\},q,\{R_i\}_{i\in\lceil n\rceil}\rangle\xrightarrow{nop}\langle\{i\},q,\{R_i\}_{i\in\lceil n\rceil}\rangle\in\Delta^M$与之对应，即两者 nop 操作相同.

(2) $\Delta=\Delta^?$时，$op=t\in I?$，在 M_M中有迁移关系 $\Delta^{M?}$与 $\Delta^?$ 相对应，此时 M_M的格局迁移为$\langle\{i\},q,\{R_i\}_{i\in\lceil n\rceil}\rangle\xrightarrow{\Delta^{M?}}\langle\{i\},q',\{R_i\}_{i\in\lceil n\rceil}\rangle$，其中$\{R_i\}_{i\in\lceil n\rceil}$表示 M_M域 R 中任意子域，迁移关系 $\Delta^{M?}$表示在$\{R_i\}$上进行判断操作.

(3) $\Delta=\Delta^=$时，$op=t\leftarrow I$，在 M_M中有迁移关系 $\Delta^{M=}$与 $\Delta^=$ 相对应，此时 M_M的格局迁移为$\langle\{i\},q,\{R_i\}_{i\in\lceil n\rceil}\rangle\xrightarrow{\Delta^{M=}}\langle\{i\},q',\{R_i'\}_{i\in\lceil n\rceil}\rangle$，其中迁移关系 $\Delta^{M=}$表示在对$\{R_i\}$进行 $t\leftarrow\lfloor v(t)\rfloor$赋值操作，$v(t)\in I$为 M_T赋值时钟值，$\lfloor v(t)\rfloor$为赋值时钟值在时间域等价后的关键点，$t\in Z$为项内字符，$t\leftarrow\lfloor v(t)\rfloor$表示将关键点$\lfloor v(t)\rfloor$赋值给字符 t.

(4) $\Delta=\Delta^\vdash$时，$op=Time\leftarrow c$，在 M_M中有迁移关系$\Delta^{M\vdash}$与$\Delta^\vdash$相对应，此时 M_M的格局迁移为$\langle\{i\},q,\{R_i\}_{i\in\lceil n\rceil}\rangle\xrightarrow{\Delta^{M\vdash}}\langle\{i\},q',\{R_i'\}_{i\in\lceil n\rceil}\rangle$，其中迁移关系 $\Delta^{M\vdash}$表示在对$\{R_i\}$中所有字符的时钟值增加 c 后取关键点，对任意 $i\in\lceil n\rceil$，$R_{ij}'=((w_{ij}|_\Gamma)\times\lfloor v(w_{ij}|_\Gamma)+c\rfloor)\cup(T_G\times\lfloor v(T_G)+c\rfloor)$，$R_i'=R_{i1}'\cdots R_{ij}'\cdots R_{id}'$，$v(w_{ij}|_\Gamma)+c$ 表示投影出的栈字符 Γ 的时钟值增加 c，$v(T_G)+c$表示全局时钟的时钟值增加 c.

(5) $\Delta=\Delta^{pop}$时，$op=pop(a,I)$，在 M_M中有迁移关系Δ^{Mpop}与 Δ^{pop} 相对应，此时 M_M的格局迁移为$\langle\{i\},q,\{R_i\}_{i\in\lceil n\rceil}\rangle\xrightarrow{\Delta^{Mpop}}\langle\{i\},q',\{R_i'\}_{i\in\lceil n\rceil}\rangle$，其中迁移关系 Δ^{Mpop}表示关键点等于$\lfloor v(a)\rfloor$的字符 $a\in\Gamma$ 出栈，$R_{ij}'=((((w_{ij}|_\Gamma)-a)\times\lfloor v((w_{ij}|_\Gamma)-a)\rfloor)\cup(T_G\times\lfloor v(T_G)\rfloor))$，$R_i'=R_{i1}'\cdots R_{ij}'\cdots R_{id}'$，$((w_{ij}|_\Gamma)-a)$表示在投影出的栈字符 Γ 内删除字符 a.

(6) $\Delta=\Delta^{push}$时，$op=push(a,I)$，在 M_M中有迁移关系Δ^{Mpush}与Δ^{push}相对应，此时 M_M的格局迁移为$\langle\{i\},q,\{R_i\}_{i\in\lceil n\rceil}\rangle\xrightarrow{\Delta^{Mpush}}\langle\{i\},q',\{R_i'\}_{i\in\lceil n\rceil}\rangle$，其中迁移关系 Δ^{Mpop}表示关键点等于$\lfloor v(a)\rfloor$的字符 $a\in(\Gamma\cup T)$入栈，$R_{ij}'=((((w_{ij}|_\Gamma)+a)\times\lfloor v((w_{ij}|_\Gamma)+a)\rfloor)\cup(T_G\times\lfloor v(T_G)\rfloor))\vee(((w_{ij}|_\Gamma)\times\lfloor v(w_{ij}|_\Gamma)\rfloor)\cup((T_G+a)\times\lfloor v(T_G+a)\rfloor))$，$R_i'=R_{i1}'\cdots R_{ij}'\cdots R_{id}'$，$((w_{ij}|_\Gamma)+a)$

表示在投影出的栈字符 Γ 内增加字符 a，(T_G+a) 表示在全局时钟内增加字符 a 的时钟.

(7) $\Delta=\Delta^{inter}$ 时，在 M_M 中有迁移关系 Δ^{Minter} 与 Δ^{inter} 相对应，此时 M_M 的格局迁移为 $\langle\{i\},q,\{R_i\}_{i\in[n]}\rangle\xrightarrow{\Delta^{Minter}}\langle\{j\},q',\{R_j\}_{j\in[n]}\rangle$，其中 $\{R_i\},\{R_j\}$ 分别为栈 i,j 的域，$i,j\in[n]$ 且 $i\neq j$. 上下文切换时，将栈 j 从等待栈切换到工作栈，同时将栈 i 从工作栈切换到等待栈.

(8) $\Delta=\Delta^{\|}$ 时，在 M_M 中有迁移关系 $\Delta^{M\|}$ 与 $\Delta^{\|}$ 相对应，此时 M_M 的格局迁移为 $\langle Work[n],q,\{R_i\}_{i\in[n]}\rangle\xrightarrow{\Delta^{M\|}}\langle Work[n]',q',\{R_j\}_{j\in[n]}\rangle$，工作栈集 $Work[n]$ 内的每个栈，调用相应的栈内迁移转换，转换后格局为 $\langle Work[n]',q',\{R_j\}_{j\in[n]}\rangle$.

4.2.2　动态转换

假定模型 TMPDN $M_T=([n],Q,q_0,\Gamma,T,\Delta)$，使用动态转换获得的模型 MPDN $M_M=([n]^M,Q^M,q_0^M,\Gamma^M,\Delta^M)$，具体转换如下：

(1) 栈个数 $[n]^M$、状态 Q^M、初始状态 q_0^M 的转换分别等同于静态转换.

(2) 栈字符 Γ^M 的转换：$\Gamma^M=2^{Z\times key}$，其中 $Z:=Y\cup Y^{\cdot}$，增加了标识项以及基础时钟字符，key 为时钟关键点集.

(3) 迁移关系 Δ 到 Δ^M 的转换规则.

M_T 的格局迁移为 $\langle\{i\},q,\{w_i\}_{i\in[n]},v\rangle\xrightarrow{op}\langle\{i\},q',\{w_i'\}_{i\in[n]},v'\rangle\in\Delta$，$w_i\in(\Gamma\times\mathbb{R}^+)^*$ 为栈 i 的字，设 d_i 为栈 i 的栈深度，用 w_{ij} 表示栈 i 内第 j 层的子字，w_{id_i} 为栈 i 的栈顶子字，$j\in[d_i]$，$w_{ij}\mid_\Gamma$ 表示 w_{ij} 投影在 Γ 的栈字符，$v(w_{ij}\mid_\Gamma)$ 表示投影与栈字符 Γ 相关联的时钟值，$\lfloor v(w_{ij}\mid_\Gamma)\rfloor$ 表示投影与栈字符 Γ 相关联的时钟值在时钟等价后的关键点，T_G 为全局时钟，$\lfloor v(T_G)\rfloor$ 表示全局时钟的时钟值在时钟域等价后的关键点. 由以上设定，则可得 R_{id_i} 在 M_M 中等价项 $R_{id_i}=((w_{id_i}\mid_\Gamma)\times\lfloor v(w_{id_i}\mid_\Gamma)\rfloor)\cup((w_{i(d_i-1)}\mid_\Gamma)^{\cdot}\times\lfloor v(w_{i(d_i-1)}\mid_\Gamma)\rfloor)\cup(T_{Gd_i}\times\lfloor v(T_{Gd_i})\rfloor)\cup(\vdash_{d_i}\times 0)\cup(\vdash_{d_i}^{\cdot}\times\lfloor v(\vdash_{d_i}^{\cdot})\rfloor)\in\Gamma^M$，其中 d_i 为 M_M 中栈 i 的栈深度，$(w_{i(d_i-1)}\mid_\Gamma)^{\cdot}$ 表示对 $w_{i(d_i-1)}$ 投影在 Γ 的栈字符的标识字符，T_{Gd_i} 表示域 R_{id_i} 中的全局时钟，$\vdash_{d_i}$ 表示域 R_{id_i} 的基础时钟字符，$\vdash_{d_i}^{\cdot}$ 表示域 R_{id_i} 的相对时间流逝字符，$\lfloor v(\vdash_{d_i}^{\cdot})\rfloor$ 表示域 R_{id_i} 的相对时间流逝的时钟值在时钟域等价后的关键点，栈 i 内容 $R_i=R_{i1}\cdots R_{ij}\cdots R_{id_i}$，其中 $i\in[n]$，$j\in[d_i]$. 从而可知对应于 M_M 的格局迁移为 $\langle q,\{R_i\}_{i\in[n]}\rangle\xrightarrow{\delta}\langle q',\{R_i'\}_{i\in[n]}\rangle$，其中 $q,q'\in Q^M$，δ 为迁移动作，为了描述方便，格局迁移时的中间状态用 $q_{tmp}()$ 表示，如 $q_{tmp}(R_{id_i})$ 表示 R_{id_i} 出栈时的中间状态. 栈内域出栈入栈时分别用 $pop()$，$push()$ 表示，如 $pop(R_{id_i})$，$push(R_{id_i})$ 分别表示栈顶域 R_{id_i} 出栈入栈. 如图 4 所示.

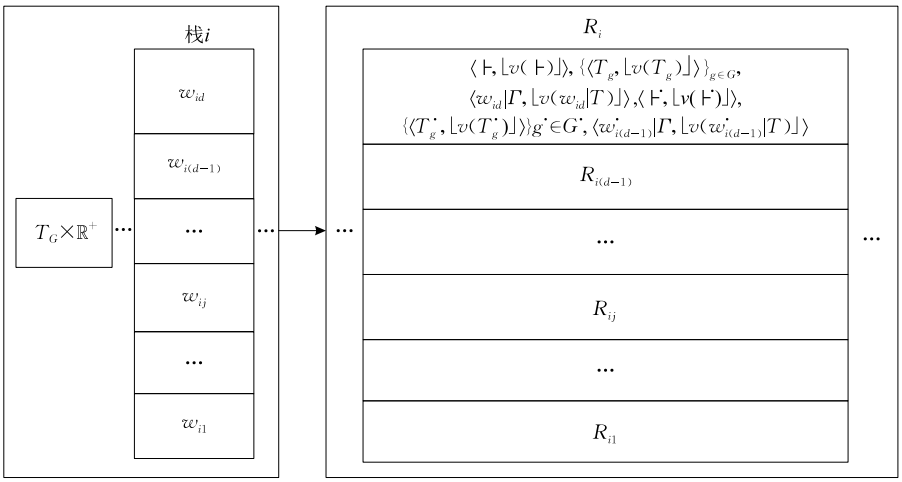


图 4　M_T 的格局动态转换为 M_M 的格局

(1) $\Delta=\Delta^{nop}$ 时，$op=nop$，在 M_M 中有 Δ^{Mnop} 与 Δ^{nop} 对应，$\Delta^{Mnop}=\Delta^{nop}$ 为空操作.

(2) $\Delta=\Delta^{?}$ 时，$op=t\in I?$，在 M_M 中有 $\Delta^{M?}$ 与 $\Delta^{?}$ 相对应，此时 M_M 的格局迁移为 $\langle\{i\},q,R_i\rangle\xrightarrow{pop(R_{id_i})}\langle\{i\},q_{tmp}(R_{id_i}),(R_i-R_{id_i})\rangle\xrightarrow{push(R_{id_i})}\langle\{i\},q',R_i\rangle$，格局迁移解释为：首先将栈顶域 R_{id_i} 出栈，转入状态 $q_{tmp}(R_{id_i})$，对时钟 t 的关键点进行判断后，再将 R_{id_i} 入栈，状态为最终状态 q'，整个过程中栈内字符并未改变.

(3) $\Delta=\Delta^{=}$ 时，$op=t\leftarrow I$，在 M_M 中有迁移关系 $\Delta^{M=}$ 与 $\Delta^{=}$ 相对应，此时 M_M 的格局迁移为 $\langle\{i\},$

$q,R_i\rangle \xrightarrow{pop(R_{id_i})} \langle\{i\},q_{tmp}(R_{id_i}),(R_i-R_{id_i})\rangle \xrightarrow{push(R_{id_i})} \langle\{i\},q',(R_i-R_{id_i}+R'_{id_i})\rangle$，其中迁移关系 $\Delta^{M=}$ 表示对 $\{R_{i1}\}$ 进行 $t\leftarrow\lfloor v(t)\rfloor$ 赋值操作，$v(t)\in I$ 为 M_T 赋值时钟值，$\lfloor v(t)\rfloor$ 为赋值时钟值在时钟域等价后的关键点，t 为 R_{id_i} 项内的某一个栈内时钟或者全局时钟，$t\leftarrow\lfloor v(t)\rfloor$ 表示将关键点 $\lfloor v(t)\rfloor$ 赋值给字符 t. R_{id_i} 赋值后记为 R'_{id_i}，描述为 $R'_{id_i}=R_{id_i}[t\leftarrow\lfloor v(t)\rfloor]$. 对任意 $i\in[n]$，$R'_{id_i}=(((w_{id_i}|_\Gamma)\times\lfloor v(t)\rfloor)\cup(T_{Gd_i}\times\lfloor v(T_{Gd_i})\rfloor)\cup(\vdash_{d_i}\times 0)\cup(\vdash^{\cdot}_{d_i}\times\lfloor v(\vdash^{\cdot}_{d_i})\rfloor)\vee(((w_{id_i}|_\Gamma)\times\lfloor v(w_{id_i}|_\Gamma)\rfloor)\cup(T_{Gd_i}\times\lfloor v(t)\rfloor)\cup(\vdash_{d_i}\times 0)\cup(\vdash^{\cdot}_{d_i}\times\lfloor v(\vdash^{\cdot}_{d_i})\rfloor))$. 格局迁移解释为：首先栈顶域 R_{id_i} 出栈，转入状态 $q_{tmp}(R_{id_i})$，给 R_{id_i} 内任意栈内时钟或者全局时钟 t，赋予 I 中任意一个值对应的关键点 $\lfloor v(t)\rfloor$，之后再将 R_{id_i} 赋值后的 R'_{id_i} 入栈，获得状态 q'.

(4) $\Delta=\Delta^{\vdash}$ 时，$op=Time\leftarrow c$，在 M_M 中有迁移关系 $\Delta^{M\vdash}$ 与 $\Delta^{\vdash}$ 相对应，此时 M_M 的格局迁移为 $\langle\{i\},q,R_i\rangle \xrightarrow{pop(R_{id_i})} \langle\{i\},q_{tmp}(R_{id_i}),(R_i-R_{id_i})\rangle \xrightarrow{push(R_{id_i})} \langle\{i\},q',(R_i-R_{id_i}+R'_{id_i})\rangle$，其中迁移关系 $\Delta^{M\vdash}$ 表示在对 $\{R_i\}$ 中所有字符的时钟值增加 c 后取关键点，R_{id_i} 增加时钟值后记为 R'_{id_i}，描述为 $R'_{id_i}=R_{id_i}[t\leftarrow\lfloor v(t+c)\rfloor]$，$t$ 为 R_{id_i} 项内所有非基础时钟. 对任意 $i\in[n]$，$R'_{id_i}=((w_{id_i}|_\Gamma)\times\lfloor v(w_{id_i}|_\Gamma)+c\rfloor)\cup(T_{Gd_i}\times\lfloor v(T_{Gd_i})+c\rfloor)\cup(\vdash_{d_i}\times 0)\cup(\vdash^{\cdot}_{d_i}\times\lfloor v(\vdash^{\cdot}_{d_i})+c\rfloor)$. 格局迁移解释：首先栈顶域 R_{id_i} 出栈，得到状态 $q_{tmp}(R_{id_i})$，之后将 R_{id_i} 项内非基础时钟 t 的时钟值，增加时钟值 c 后取关键点，再将 R_{id_i} 赋值后的 R'_{id_i} 入栈，最后获得状态 q'.

(5) $\Delta=\Delta^{pop}$ 时，$op=pop(a,I)$，在 M_M 中有迁移关系 Δ^{Mpop} 与 Δ^{pop} 相对应，此时 M_M 的格局迁移为 $\langle\{i\},q,R_i\rangle \xrightarrow{pop(R_{id_i})} \langle\{i\},q_{tmp}(R_{id_i}),(R_i-R_{id_i})\rangle \xrightarrow{pop(R_{i(d_i-1)})} \langle\{i\},q_{tmp}(R_{id_i},R_{i(d_i-1)}),(R_i-R_{id_i}-R_{i(d_i-1)})\rangle \xrightarrow{push((R_{id_i}+R_{i(d_i-1)})')} \langle\{i\},q',(R_i-R_{id_i}-R_{i(d_i-1)}+(R_{id_i}+R_{i(d_i-1)})')\rangle$，其中 $(R_{id_i}+R_{i(d_i-1)})'$ 为 R_{id_i}，$R_{i(d_i-1)}$ 的组合域，迁移关系 Δ^{Mpop} 表示关键点等于 $\lfloor v(a)\rfloor$ 的字符 $a\in\Gamma$ 出栈，$(R_{id_i}+R_{i(d_i-1)})'=((w_{id_i}|_\Gamma-a)\times\lfloor v(w_{id_i}|_\Gamma-a)\rfloor)\cup(T_{Gd_i}\times\lfloor v(T_{Gd_i})\rfloor)\cup(w^{\cdot}_{i(d_i-1)}|_\Gamma\times([v(w_{i(d_i-1)}|_\Gamma)+v(\vdash^{\cdot}_{d_i})\rfloor))\cup(w^{\cdot}_{i(d_i-1)}|_\Gamma\times(\lfloor v(w^{\cdot}_{i(d_i-1)}|_\Gamma)+v(\vdash^{\cdot}_{d_i})\rfloor))\cup(\vdash^{\cdot}\times(\lfloor v(\vdash^{\cdot}_{d_i-1})+v(\vdash^{\cdot}_{d_i})\rfloor))\cup(\vdash\times 0)$，其中 $((w_{id_i}|_\Gamma)-a)$ 表示投影出栈 i 的栈顶子字 w_{id_i} 的字符 Γ，并删除 Γ 内的 a 字符. 当 M_M 栈顶域的项内字符出栈时，其下一层域的项内字符的时钟需考虑相对时间流逝情况. 格局迁移解释为：首先栈顶域 R_{id_i} 出栈，得状态 $q_{tmp}(R_{id_i})$，然后将下一层域 $R_{i(d_i-1)}$ 出栈，转入状态 $q_{tmp}(R_{id_i},R_{i(d_i-1)})$，最后将 R_{id_i}，$R_{i(d_i-1)}$ 的组合域 $(R_{id_i}+R_{i(d_i-1)})'$ 入栈，转入状态 q'.

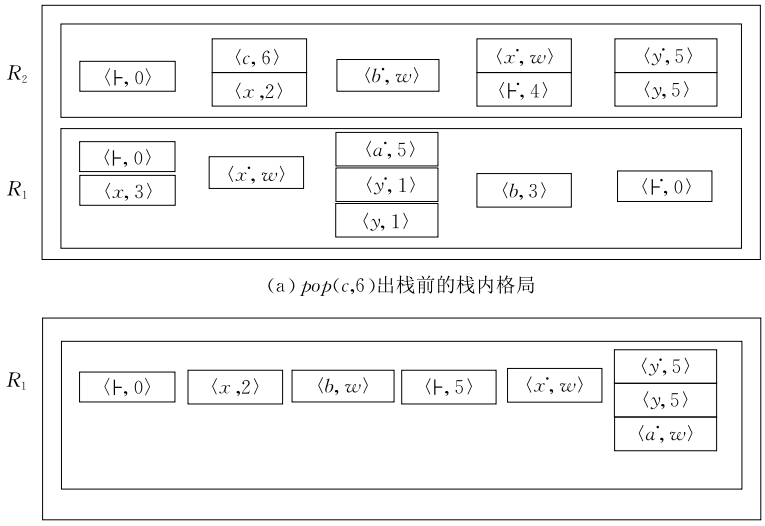
组合域构造规则：栈顶域 R_{id_i} 的项内栈字符出栈时，需将 R_{id_i}，$R_{i(d_i-1)}$ 合并成一个组合域 $(R_{id_i}+R_{i(d_i-1)})'$.

① 将 R_{id_i} 中所有非出栈的普通栈内字符 $w_{id_i}|_\Gamma-a$、全局时钟 T_{Gd_i} 以及其关键点 $\lfloor v(w_{id_i}|_\Gamma-a)\rfloor$、$\lfloor v(T_{Gd_i})\rfloor$ 保存在组合域中.

② 将 $R_{i(d_i-1)}$ 中所有非全局时钟（普通栈内字符 $w_{i(d_i-1)}|_\Gamma$、普通栈内字符的标识字符 $w^{\cdot}_{i(d_i-1)}|_\Gamma$、基础时钟流逝字符 $\vdash^{\cdot}_{d_i-1}$ 和基础时钟字符 $\vdash_{d_i-1}$）保存到组合域中，其关键点需在原来的时钟值上增加时间流逝，时间流逝为栈顶域 R_{id_i} 中 $\vdash^{\cdot}_{d_i}$ 的时钟值，故保存到组合域时，其关键点分别为 $(\lfloor v(w_{i(d_i-1)}|_\Gamma)+v(\vdash^{\cdot}_{d_i})\rfloor)$，$(\lfloor v(w^{\cdot}_{i(d_i-1)}|_\Gamma)+v(\vdash^{\cdot}_{d_i})\rfloor)$，$(\lfloor v(\vdash^{\cdot}_{d_i-1})+v(\vdash^{\cdot}_{d_i})\rfloor)$，0.

为进一步描述组合域构造规则，通过一个实例来说明，如图 5 所示，图 5(a)中 R_2 的普通项栈内字符、基础时钟字符以及全局时钟$\langle\vdash,0\rangle$、$\langle x,2\rangle$、$\langle y,5\rangle$、$\langle y^{\cdot},5\rangle$、$\langle x^{\cdot},\omega\rangle$在$\langle c,6\rangle$出栈后保持不变，$R_1$ 中的非全局栈字符$\langle a^{\cdot},5\rangle$、$\langle b^{\cdot},3\rangle$、$\langle\vdash^{\cdot},0\rangle$在$\langle c,6\rangle$出栈后增加时间流逝（$R_2$ 中$\langle\vdash^{\cdot},4\rangle$的关键点），分别为$\langle a^{\cdot},\omega\rangle$、$\langle b,\omega\rangle$、$\langle\vdash^{\cdot},5\rangle$，最后将两者合并成组合域，如图 5(b)中的 R_1.

(6) $\Delta=\Delta^{push}$ 时，$op=push(a,I)$，M_M 中有迁移关系 Δ^{Mpush} 与 Δ^{push} 相对应，此时 M_M 的格局迁移为 $\langle\{i\},q,R_i\rangle \xrightarrow{pop(R_{id_i})} \langle\{i\},q_{tmp}(R_{id_i}),(R_i-R_{id_i})\rangle \xrightarrow{push(R_{id_i})} \langle\{i\},q_{tmp}(R_{id_i},R_{id_i}),R_i\rangle \xrightarrow{push(R'_{id_i})} \langle\{i\},q',(R_i+R'_{id_i})\rangle$，迁移关系 Δ^{Mpush} 表示关键点等于 $\lfloor v(a)\rfloor$ 的字符 $a\in\Gamma$ 入栈，$R'_{id_i}=(a\times\lfloor v(a)\rfloor)\cup(w^{\cdot}_{id_i}|_\Gamma\times\lfloor v(w_{id_i}|_\Gamma)\rfloor)\cup(T_{Gd_i}\times\lfloor v(T_{Gd_i})\rfloor)\cup(T^{\cdot}_{Gd_i}\times\lfloor v(T_{Gd_i})\rfloor)\cup(\vdash^{\cdot}\times 0)\cup(\vdash\times 0)$，其中 $a\times\lfloor v(a)\rfloor$ 表示新入栈的字符及其关键点，$w^{\cdot}_{id_i}|_\Gamma\times(\lfloor v(w_{id_i}|_\Gamma)\rfloor)$ 表示将栈顶域 R_{id_i} 内普通栈内字符，标识成标识字符，其关键点不变化，$T^{\cdot}_{Gd_i}\times\lfloor v(T_{Gd_i})\rfloor$ 表示将栈顶域 R_{id_i} 内全局时钟，标识成标识字符，其关键点不变化，$\vdash^{\cdot}\times 0$ 表示将栈顶域 R_{id_i} 内基础时钟字符标识



(a) $pop(c,6)$出栈前的栈内格局

(b) $pop(c,6)$出栈后的栈内格局

图 5　Δ^{pop} 格局迁移转换实例

为标识字符，其关键点为零. 格局迁移解释为：首先将栈顶域 R_{id_i} 出栈，然后再将域 R_{id_i} 入栈，最后将含字符 a 的域 R'_{id_i} 入栈.

用实例进一步说明 Δ^{Mpush} 迁移操作，如图 6 所示，域 R_1 为⟨b,4⟩入栈前的栈顶域，R_1 中的普通项包含基础时钟⟨⊢,0⟩、全局时间⟨x,1⟩、栈内字符⟨a,2⟩，在⟨b,4⟩入栈后，R_1 中的普通项被'标识成标识项⟨⊢',0⟩、⟨x',1⟩、⟨a',2⟩，同时保留⟨⊢,0⟩、⟨x,1⟩，最终结果为新的栈顶域 R_2.

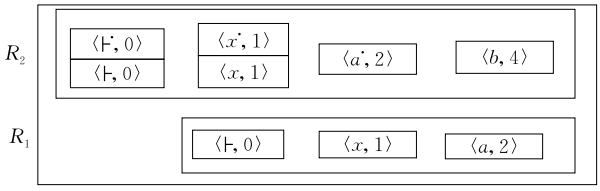


图 6　$push(b,4)$入栈后的格局

(7) $\Delta=\Delta^{inter}$ 时，转换过程与静态算法类似.

(8) $\Delta=\Delta^{\|}$ 时，转换过程与静态算法类似，但是工作栈集 $Work[n]$ 内的每个栈，调用的栈内迁移转换为栈内动态转换.

5　可达性等价的正确性

给出 TMPDN $M_T=([n],Q,q_0,\Gamma,T,\Delta)$，根据转换规则，可构造出 MPDN $M_M=([n]^M,Q^M,q_0^M,\Gamma^M,\Delta^M)$，并参考文献[16]，证明状态 q_F 在 TMPDN 可达当且仅当其转换状态 q_F^M 在 MPDN 可达，从而表明其转换的正确性.

定义 3.　给定 M_T，假设 M_T 的初始格局 $C_0=\langle q_0,\{w_i\}_{i\in[n]},v_0\rangle$，其中 $q_0\in Q$ 表示初始状态，w_i 为栈 i 的字，表示栈 i 的初始内容，描述为 $w_i\in(\Gamma\times\mathbb{R}^+)^*$，$v_0$ 表示初始时钟值. 目标格局 $C_F=\langle q_F,\{w_j\}_{j\in[n]},v_F\rangle$，如果存在格局迁移关系 $C_0\rightarrow^* C_F$，其中 $\rightarrow^*$ 表示经过一步或者多步迁移关系，那么称状态 q_F 在迁移系统中可达.

设 $R=R_0\cdots R_n$ 是 M_M 域集合，域 $R_0,R_1\in R$，Δ 为域 R 上的迁移关系集，并且 Δ 是满足自反性、非对称性、传递性的偏序关系，那么如果 R_0 经过 Δ 迁移到 R_1，并且 Δ 是严格偏序关系，则 R_0，R_1 的关系记为 $R_0\prec R_1$，如果 R_0 经过 Δ 迁移到 R_1，并且 Δ 是非严格偏序关系，则 R_0，R_1 的关系记为 $R_0\leq R_1$. 对于域 R，如果 $R_i\prec R_{i+1}$，$0\leq i\leq n$，则称 R 为相关域，如果 $R_i\leq R_{i+1}$，$0\leq i\leq n$，则称 R 为弱相关域. 如果 R 为(弱)相关域，则称 M_M 的格局 $C=\langle q,R\rangle$ 为(弱)相关格局. 设 $R=R_0\cdots R_n$ 为弱相关域，域 $R'=R'_0\cdots R'_n$ 为相关域，如果 $R'_n=R_n$、$R'_i\in R_i^{++}$(其中 R_i^{++} 是 R_i 经时间流逝之后生成的域集，称为 R_i 的时间迁移域)、且 $R'_i\prec R'_{i+1}$，$0\leq i\leq n$，则称域 R' 是域 R 的强相关域. 设 M_M 的相关格局 $C=\langle q,R\rangle$，且域 R' 是 R 域的强相关域，则称 $C'=\langle q,R'\rangle$ 为 C 的强相关格局.

定理 1.　对于 TMPDN 任意一个格局 C^T，通过时钟域等价转化，在 MPDN 都存在与之对应的格局 C^M.

证明.　设 M_M 的一个格局 $C^M=\langle q^M,R^M\rangle$，$M_T$ 的一个格局 $C^T=\langle q^T,\{w_i\}_{i\in[n]},v\rangle$，$S=\{w_i\}_{i\in[n]}\cup v$ 为 M_T 此刻字和时钟集合，S 经过时间域转换成 R，$w_i=\langle a_1,v_1\rangle\langle a_2,v_2\rangle\cdots\langle a_n,v_n\rangle$，$R=R_0\cdots R_n$，下面表达式成立.

(1) $q^M=q^T$;

(2) $((w_{ij}\,|\Gamma)\times\lfloor v(w_{ij}\,|\Gamma)\rfloor)\in R$ iff $w_{ij}\in S$;

(3) $(T_G\times\lfloor v(T_G)\rfloor)\in R$ iff $v\in S$.

当前仅当 $C^T\models_S C^M$,即对于 M_T 任意一个格局 C^T,通过时间域等价技术转化后,在 M_M 都存在与之对应的格局 C^M. 证毕.

定理 2. 如果目标状态 q'_F 在 M_M 可达,那么存在一个正则可达格局 C^M(q'_F 为格局 C^M 的状态),且可达格局 C^M 为弱相关格局,则可得如下结论:至少存在一个与 C^M 对应的强相关格局 $C'^M=\langle q,R'\rangle$.

证明. 假设 M_M 中存在一个可达目标状态 q'_F,显然在 M_M 存在一个可达格局 $C^M=\langle q'_F,R\rangle$. 再证明 M_M 中所有可达格局都是弱相关,假设 M_M 存在一个弱相关域 $R=R_0R_1\cdots R_n$,域 R 经过迁移得到域 $R'=R'_0R'_1\cdots R'_n$,因 $R_{n-1}\prec R_n$,$R'_n\in R_n^{++}$,所以 $R_{n-1}\prec R'_n$,此时称 R' 为弱相关域,因此由 R' 构成的 M_M 中所有可达格局都为弱相关. 最后证明至少存在一个与 C^M 对应的强相关格局 $C'^M=\langle q,R'\rangle$,假设存在域 $R'=R'_0R'_1\cdots R'_n$,满足 $R'_n=R_n$,$R'_i\in R_i^{++}$ 以及 $R'_i\prec R'_{i+1}$,那么称 R' 是 R 的强相关域,因此 R' 构成的强相关格局 $C'^M=\langle q,R'\rangle$ 一定存在. 证毕.

为了证明可达性的等价,引入如下两个定律[16].

定律 1. 对于属于 M_M 的任意一个正则可达格局 C^M,都有与之对应的强相关格局 $C'^M=\langle q,R'\rangle$,此时在 M_T 中一定存在 S,且必定存在与 S 对应的格局 C^T,满足上述条件后则存在 $C^T\models_S C^M$ 且 $C_{init}\rightarrow^* C_F$.

定律 2. 对于属于 M_T 的任意一个格局 C^T,在 M_M 中必定存在与之对应的格局 C^M,此时 C^M 至少存在一个强相关格局 $C'^M=\langle q,R'\rangle$,并且存在域 R' 的转换集合 S,满足上述条件后则存在 $C^T\models_S C^M$ 并且 $C^M_{init}\rightarrow^* C^M_F$.

定理 3. 状态 q_F 在 M_T 可达当且仅当 q_F 的转换状态 q'_F 在 M_M 可达.

证明. 先证后向蕴涵关系:状态 q_F 在 M_T 可达⇐其转换状态 q'_F 在 M_M 可达.

由定理 2 和定律 1 可知,如果 M_M 中存在一个正则格局 C^M,则必定存在一个强相关格局 C'^M 和转成域 R' 的转换集合 S,那么 M_T 中必定存在与 C^M 对应的格局 C^T,则存在 $C^T\models_S C^M$ 并且 $C_{init}\rightarrow^* C_F$,即状态 q_F(q_F 为格局 C^T 的状态)在 M_T 可达.

再证前向蕴涵关系:状态 q_F 在 M_T 可达⇒其转换状态 q'_F 在 M_M 可达.

如果目标状态 q'_F 在 M_T 是可达,由定理 1 可知在 M_M 中必定存在与 C^T 对应的格局 C^M(q_F 为格局 C^M 的状态),因此至少存在一个强相关格局 $C'^M=\langle q,R'\rangle$. 由定律 2 可知,对于 M_T 中的格局 C^T,必定存在一个强相关格局 C'^M 和转成域 R' 的转换集合 S,从而必定存在与 C'^M 对应的格局 C^M,满足上述条件则存在 $C^T\models_S C^M$ 并且 $C^M_{init}\rightarrow^* C^M_F$,即状态 q_F(q_F 为格局 C^T 的状态)在 M_M 可达.

因此,状态 q_F 在 M_T 可达当且仅当其转换状态 q'_F 在 M_M 可达. 证毕.

6 可达性等价算法

基于时钟等价、静态转换和动态转换思想,提出了针对于时间多栈下推网络 TMPDN $M_T=([n],Q,q_0,\Gamma,T,\Delta)$,转换为对应的多栈下推网络 MPDN $M_M=([n]^M,Q^M,q_0^M,\Gamma^M,\Delta^M)$ 的算法. 算法的输入是连续的 TMPDN 迁移系统,输出是离散的 MPDN 迁移系统. 迁移系统 M_T 的初始格局为 $C_0=\langle\{i\},q_0,\{w_i\}_{i\in[n]},v_0\rangle$,其中 q_0 表示初始状态,$\{w_i\}_{i\in[n]}$ 表示栈 i 的初始内容为空,v_0 表示初始时钟值,其值初始为 0. 转换算法分为静态算法和动态算法,根据各自的转换规则逐步进行,首先初始化工作线程列表以及域内项,再依据转换类型相应的执行静态算法、动态算法以及栈间操作和并发迁移算法. 算法 1 中的第 1,2 行为整体算法的初始化,首先初始化工作线程列表 *worklist*,将格局 $\langle\{i\},q_0,\{w_i\}_{i\in[n]},v_0\rangle$ 保存至 *worklist* 中,然后初始化域 R,初始域 R_{init} 由全局时钟、栈底标识符 *bottom* 以及基础时钟字符组成,最后将迁移迁移关系 $\langle q_0,push(R_{init}),q_0\rangle$ 放入迁移关系集 Δ^M 中. 第 3 行为枚举静态算法与动态算法,第 4,5 行为静态算法判断执行部分,第 6,7 行为动态算法判断执行部分,第 8 至 10 行为算法结束条件.

算法 1. 时间多栈下推网络转化为多栈下推网络.

输入:时间多栈下推网络 $M_T=(n,Q,q_0,\Gamma,T,\Delta)$

输出:对应的多栈下推网络 $M_M=(n^M,Q^M,q_0^M,\Gamma^M,\Delta^M)$

1. 初始化工作线程列表 $worklist\leftarrow(q_0,\{w_i\}_{i\in[n]},v_0)$,其对应的初始域 $R_{init}=\{(t,0)\,|\,t\in T\cup T^{\cdot}\}\cup\{(bottom,0),(bottom^{\cdot},0)\}\cup\{\vdash,\vdash^{\cdot}\}$ //其中 *bottom* 表示空栈
2. MOVE $(q_0,push(R_{init}),q_0)$ INTO Δ^M
   //其中 q_0 为 M_M 的初始控制状态位,并且把迁移关系放入到 Δ^M 中
3. ENUM $AlgType\{Algstatic,Algdynamic\}Atype$;
   //枚举算法类型

4. IF ($Atype==Algstatic$)
//判定选择的算法类型，并执行该算法
5. $Algstat()$；
6. ELSE ($Atype==Algdynamic$)
7. $Algdyna()$；
8. FORALL $\Delta'\subseteq\Delta$
//设 Δ'为迁移关系集合 Δ 里的任意一个迁移关系，并设置算法结束条件
9. WHILE $\Delta\neq\varnothing$ DO
10. REMOVE Δ' FROM Δ

静态转换和动态转换算法分别如算法 2 和算法 3 所示，两者依据 M_T中不同的迁移关系Δ'在M_M中作出相应的转换，将 M_M中相应的迁移关系压入到迁移关系集 Δ^M中. 静态转换时分为栈内迁移转换、栈间操作以及并发迁移转换 3 类，其中栈内迁移转换是两个模型中栈层间的一一对应转换，如算法 2 中的第 2 至 17 行，栈间操作为上下文切换操作，如算法 2 中的第 20 至 21 行，等同于 M_T中切换操作，并发迁移转换为工作栈集 $Work[n]$内同时调用栈内迁移转换，如算法 2 中的第 22 至 24 行，动态转换同样分栈内迁移转换、栈间操作以及并发迁移转换 3 类，其中栈内迁移转换只对 M_M模型栈顶及其下一层栈的栈内字符分析讨论，如算法 3 中的第 2 至 17 行，在转换过程中需增加中间状态 $q_{tmp}()$，如状态 $\langle q_{tmp}(R_{id_i}),(R_i-R_{id_i})\rangle$等，栈间操作和并发迁移转换与静态算法类似.

算法 2.　静态算法.

输入：M_T的迁移关系
输出：M_M的静态迁移关系

1. $Algstat()\{$
2. $Alginstat()\{$ //栈内迁移转换
3. SWITCH (Δ')$\{$
4. CASE $\Delta'=\Delta^{nop}$：//空操作，对应 M_M状态和域都不改变
5. MOVE $\langle\{i\},q,R\rangle\xrightarrow{nop}\langle\{i\},q,R\rangle$ INTO Δ^M; BREAK;
6. CASE $\Delta'=\Delta^{?}$：//判断操作，对应M_M状态发生改变
7. MOVE $\langle\{i\},q,\{R_i\}_{i\in[n]}\rangle\xrightarrow{\Delta^{M?}}\langle\{i\},q',\{R_i\}_{i\in[n]}\rangle$ INTO Δ^M; BREAK;
8. CASE $\Delta'=\Delta^{=}$：//时间重置操作，对$\{R_i\}$进行$t\leftarrow\lfloor v(t)\rfloor$赋值、操作来构造新域$\{R_i'\}_{i\in[n]}$
9. MOVE $\langle\{i\},q,\{R_i\}_{i\in[n]}\rangle\xrightarrow{\Delta^{M=}}\langle\{i\},q',\{R_i'\}_{i\in[n]}\rangle$ INTO Δ^M; BREAK;
10. CASE $\Delta'=\Delta^{\vdash}$：//时间流逝操作，在对$\{R_i\}$中所有字符的时钟值增加 c 后取关键点构造新域$\{R_i'\}_{i\in[n]}$
11. MOVE $\langle\{i\},q,\{R_i\}_{i\in[n]}\rangle\xrightarrow{\Delta^{M\vdash}}\langle\{i\},q',\{R_i'\}_{i\in[n]}\rangle$ INTO Δ^M; BREAK;
12. CASE $\Delta'=\Delta^{pop}$：//出栈操作，关键点等于$\lfloor v(a)\rfloor$的字符 a 出栈
13. MOVE $\langle\{i\},q,\{R_i\}_{i\in[n]}\rangle\xrightarrow{\Delta^{Mpop}}\langle\{i\},q',\{R_i'\}_{i\in[n]}\rangle$ INTO Δ^M; BREAK;
14. CASE $\Delta'=\Delta^{push}$：//入栈操作，关键点等于$\lfloor v(a)\rfloor$的字符 a 入栈
15. MOVE $\langle\{i\},q,\{R_i\}_{i\in[n]}\rangle\xrightarrow{\Delta^{Mpush}}\langle\{i\},q',\{R_i'\}_{i\in[n]}\rangle$ INTO Δ^M; BREAK;
16. $\}$
17. $\}$
18. $Algstatconcur()\{$
19. SWITCH (Δ')$\{$
20. CASE $\Delta'=\Delta^{inter}$：//栈间操作，从 i 号工作栈切换到 j 号栈
21. MOVE $\langle\{i\},q,\{R_i\}_{i\in[n]}\rangle\xrightarrow{\Delta^{Minter}}\langle\{j\},q',\{R_j\}_{j\in[n]}\rangle$ INTO Δ^M; BREAK;
22. CASE $\Delta'=\Delta^{\parallel}$：//并发迁移，迁移关系 $\Delta^{M\parallel}$与 $\Delta^{\parallel}$相对应
23. MOVE $\langle Work[n],q,\{R_i\}_{i\in[n]}\rangle\xrightarrow{\Delta^{M\parallel}}\langle Work[n]',q',\{R_j\}_{j\in[n]}\rangle$ INTO Δ^M;
24. $Alginstat()$; BREAK; //调用栈内迁移转换函数
25. $\}$
26. $\}$
27. $\}$

算法 3.　动态算法.

输入：M_T的迁移关系
输出：M_M的动态迁移关系

1. $Algdyna()\{$
2. $Algindyna()\{$ //栈内迁移转换
3. SWITCH(Δ')$\{$
4. CASE $\Delta'=\Delta^{nop}$：//空操作，对应 M_M状态和域都不改变
5. MOVE $\langle\{i\},q,R\rangle\xrightarrow{nop}\langle\{i\},q,R\rangle$ INTO Δ^M; BREAK;
6. CASE $\Delta'=\Delta^{?}$：//判断操作，M_M栈顶域的出栈入栈，增加了中间状态 $q_{tmp}(R_{id_i})$
7. MOVE $\langle\{i\},q,R_i\rangle\xrightarrow{pop(R_{id_i})}\langle\{i\},q_{tmp}(R_{id_i}),(R_i-R_{id_i})\rangle\xrightarrow{push(R_{id_i})}\langle\{i\},q',R_i\rangle$ INTO Δ^M; BREAK;
8. CASE $\Delta'=\Delta^{=}$：/*重置操作，对$\{R_i\}$进行 $t\leftarrow\lfloor v(t)\rfloor$

赋值操作来构造新域$\{R_i-R_{id_i}+R'_{id_i}\}_{i\in[n]}$，构造时需增加中间辅助状态 $q_{tmp}(R_{id_i})$*/

MOVE $\langle\{i\},q,R_i\rangle\xrightarrow{pop(R_{id_i})}\langle\{i\},q_{tmp}(R_{id_i}),(R_i-R_{id_i})\rangle\xrightarrow{push(R_{id_i})}\langle\{i\},q',(R_i-R_{id_i}+R'_{id_i})\rangle$ INTO Δ^M; BREAK;

CASE $\Delta'=\Delta^{\vdash}$: /*时间流逝操作，在对$\{R_i\}$中所有字符的时钟值增加 c 后取关键点构造新域$\{R_i-R_{id_i}+R'_{id_i}\}_{i\in[n]}$，构造时需增加中间辅助状态 $q_{tmp}(R_{id_i})$*/

MOVE $\langle\{i\},q,R_i\rangle\xrightarrow{pop(R_{id_i})}\langle\{i\},q_{tmp}(R_{id_i}),(R_i-R_{id_i})\rangle\xrightarrow{push(R_{id_i})}\langle\{i\},q',(R_i-R_{id_i}+R'_{id_i})\rangle$ INTO Δ^M; BREAK;

CASE $\Delta'=\Delta^{pop}$ //出栈操作，关键点等于$\lfloor v(a)\rfloor$的字符 a 出栈

MOVE $\langle\{i\},q,R_i\rangle\xrightarrow{pop(R_{id_i})}\langle\{i\},q_{tmp}(R_{id_i}),(R_i-R_{id_i})\rangle\xrightarrow{pop(R_{i(d_i-1)})}\langle\{i\},q_{tmp}(R_{id_i},R_{i(d_i-1)}),(R_i-R_{id_i}-R_{i(d_i-1)})\rangle\xrightarrow{push((R_{id_i}+R_{i(d_i-1)})')}\langle\{i\},q',(R_i-R_{id_i}-R_{i(d_i-1)}+(R_{id_i}+R_{i(d_i-1)})')\rangle$ INTO Δ^M; BREAK;

CASE $\Delta'=\Delta^{push}$ //入栈操作，关键点等于$\lfloor v(a)\rfloor$的字符 a 入栈

MOVE $\langle\{i\},q,R_i\rangle\xrightarrow{pop(R_{id_i})}\langle\{i\},q_{tmp}(R_{id_i}),(R_i-R_{id_i})\rangle\xrightarrow{push(R_{id_i})}\langle\{i\},q_{tmp}(R_{id_i},R_{id_i}),R_i\rangle\xrightarrow{push(R'_{id_i})}\langle\{i\},q',(R_i+R'_{id_i})\rangle$ INTO Δ^M; BREAK;

}

}

Algdynaconcur(){

SWITCH(Δ'){

CASE $\Delta'=\Delta^{inter}$: //栈间操作，从 i 号工作栈切换到 j 号栈

MOVE $\langle\{i\},q,\{R_i\}_{i\in[n]}\rangle\xrightarrow{\Delta^{Minter}}\langle\{j\},q',\{R_j\}_{j\in[n]}\rangle$ INTO Δ^M; BREAK;

CASE $\Delta'=\Delta^{\parallel}$://并发迁移，迁移关系$\Delta^{M\parallel}$与$\Delta^{\parallel}$相对应

MOVE $\langle Work[n],q,\{R_i\}_{i\in[n]}\rangle\xrightarrow{\Delta^{M\parallel}}\langle Work[n]',q',\{R_j\}_{j\in[n]}\rangle$ INTO Δ^M;

Algindyna(); BREAK; //调用栈内迁移转换函数

}

}

}

上述算法是可终止的. 设计静态转换算法时，首先根据静态转换规则，将算法分为栈内迁移转换 *Alginstat*()和并发转换 *Algstatconcur*()两部分，其中算法 *Alginstat*()依据不同的迁移规则调用相应的执行操作，算法 *Algstatconcur*()包含了栈间操作和并发迁移，在运行并发迁移算法时需调用算法 *Alginstat*(). 静态转换算法的时间复杂度与栈深度、栈数量、算法转换时的系统状态集、格局迁移关系集以及时钟集等元素有关，在最坏情况下，算法时间复杂度与项元素集合以及关键点集合的笛卡尔积大小呈指数关系，具体可表示为 $O((\lfloor n\rfloor\times d)|Q||\Delta||M_T|2^{Z\times key})$，其中$\lfloor n\rfloor\times d$ 是时间多栈下推系统中栈的数目与栈深度的乘积，$|Q|$是时间多栈下推系统状态集的大小，$|\Delta|$是格局迁移关系的个数，$|M_T|$是时间多栈下推系统的大小，$Z\times key$ 为项元素集合与关键点集合的笛卡尔积. 然而在实际检验过程中，由于一般情况下转换后存在不可达等价状态，以及因某些时刻栈并不包含所有项元素集合，存在合并状态，故其时间复杂度小于 $O((\lfloor n\rfloor\times d)|Q||\Delta||M_T|2^{Z\times key})$. 动态算法设计思路与静态算法类似，但其转换时只对栈顶操作，与栈深度无关. 因动态转换只关心栈顶层及其下一层域状态空间，故等价区域空间比静态转换算法缩减很多. 假设栈顶项元素最大集合为 $\mathrm{Max}(Z_{top})$，理论上最坏情况下其时间复杂度为 $O(\lfloor n\rfloor|Q||\Delta||M_T|2^{(\mathrm{Max}(Z_{top})+\mathrm{Max}(Z_{top-1}))\times key})$.

7 TMPDN 的上下文限界验证

对并发程序形式化验证时，若每个线程建模成一个有限状态系统，整个程序的状态空间将呈指数增长，其可达性判定为 PSPACE-hard. 若建模成下推系统时，其可达性问题也不可判定，但将线程间的上下文切换限定在有限次数内，使得完备性、精确性、覆盖率、以及效率达到一个很好的平衡. 上下文限界使递归并发程序的可达性成为可判定，并且程序中的大部分并发错误，如数据竞争和原子性违背，大部分出现在前期的上下文切换[14].

利用 TMPDN 对实时并发递归程序建模，再将 TMPDN 转换为 MPDN 后，实时并发递归程序的上下文切换限界可达性问题可以转换为 MPDN 的限界可达性问题，之后可利用限界模型检验工具如 JMOPED, CHESS[17], ESBMC[18] 等，对转换后的

MPDN 进行验证分析.

为了验证与分析基于 MPDN 的并发模型，主要集中在对栈的限定上[19-20]，通过栈限定使可达性为可判定，诸如上下文限界技术[21]，在牺牲少许完备性的情况下，能很好分析栈之间的交互情况. 2012 年 Abdulla 等人[22]给出了更多上下文定义，提出了阶段限界方法，并用实验论证了方法的优越性. 2014 年 Qian 等人[23]提出了将进程的消息队列约束为良序的思想，使得递归队列并发程序的可达性成为可判定问题.

在使用 TMPDN 描述并发程序时，将程序分为进程 P(*process*)、过程 F(*procedure*)及语句(*statement*) 3 个部分. 其中进程 $P=\langle G,F_1\cdots F_m\rangle$ 由全局变量集 G 和过程 F_i 组成，进程有一个入口主过程 *main*，过程 F 由参数序列、局部变量集以及语句序列组成，语句包含空语句(*nos*)、假设(*assume*)、断言(*assert*)、原子语句(*atomic*)等语句. 并发程序 $CC=\langle S,P_1\cdots P_n\rangle$ 由共享变量集 S(*shared variables*)和进程序列组成. 每个进程 P_i 定义一个计数器 pc_i，将过程 F 看成一个独立的单元，并把 F 中的每条语句放入到程序执行序列中，并为每条语句增加判断条件以便监视语句是否执行，判断条件包含进程计数器 pc_i 是否正确和局部布尔变量 Yn 的取值，如果 pc_i 正确而 Yn 返回失败标识 *false*，则表明该语句未执行，如果除当前进程计数器外，其他计数器都返回错误，则表明其他进程未被执行. 由以上设定可以模拟及监视程序上下文切换操作，并利于 TMPDN 对实时递归并发程序的建模.

8　实例分析

本节给出了一个具体实例分析，通过文中所提出的静态和动态转换方法，将 TMPDN 模型转换为 MDPN 模型. TMPDN 模型如图 1 所示，即时间多栈下推系统 $M_T=(n,Q,q_0,\Gamma,T,\Delta)$，其中全局变量为 g，时钟为 t_1，栈字符集合 $\Gamma=\{a_1,b_1,a_i,b_i,c_i,a_j,b_j,c_j\}$. 依据优化的时钟等价技术，可获得时钟关键点集合 $key=\{0,2,6,10,\infty\}$.

8.1　静态转换

由于静态转换的状态空间庞大，下文将通过简化状态格局迁移的描述，来表达静态转换的清晰思路. 为了简化描述过程，对时间流逝进行单独分析，而在其他格局迁移分析时暂不考虑时间流逝的情况，并且也没有具体描述时钟剩余部分 $re()$ 的大小关系.

图 7(a)描述入栈操作 Δ^{push} 和时间重置 $\Delta^{=}$ 的静态转换，其中格局 R_0 对应图 1 所示模型的格局 C_0，格局 R_0 包含栈 $1\cdots i,j\cdots n$ 共 n 个栈，其中栈内的全局变量 g、时钟 t_1 以及栈字符 a_1,b_1,a_i,b_i,a_j,b_j 对应的时钟关键点分别为 0，∞，2，2，6，2，∞，6. 在 MPDN中与 TMPDN 的迁移关系 Δ^{push} 相对的格局迁移关系 Δ^{Mpush} 为 $push(\langle c_i,2\rangle)\|push(\langle c_j,2\rangle)$，$R_0$ 经过 Δ^{Mpush} 迁移至格局 R_1，在 R_1 中增加了项元素 $\langle c_i,2\rangle$ 和 $\langle c_j,2\rangle$. TMPDN 的迁移关系 $\Delta^{=}:t_1\leftarrow[6:10)$ 表示时钟 t_1 的时钟值重置为 $7.5\in[6:10)$，与之对应 MPDN 的迁移关系 $\Delta^{M=}$ 为 $t_1\leftarrow 6$ 表示将 t_1 赋值为 6，因为 7.5 在关键点[6,10)之间，所以在对 t_1 赋值时向下取关键点 6，此时 t_1 与 g 取关键点剩余部分关系由 $re(t_1)>re(g)$ 转换为 $re(t_1)<re(g)$，其他不一一描述.

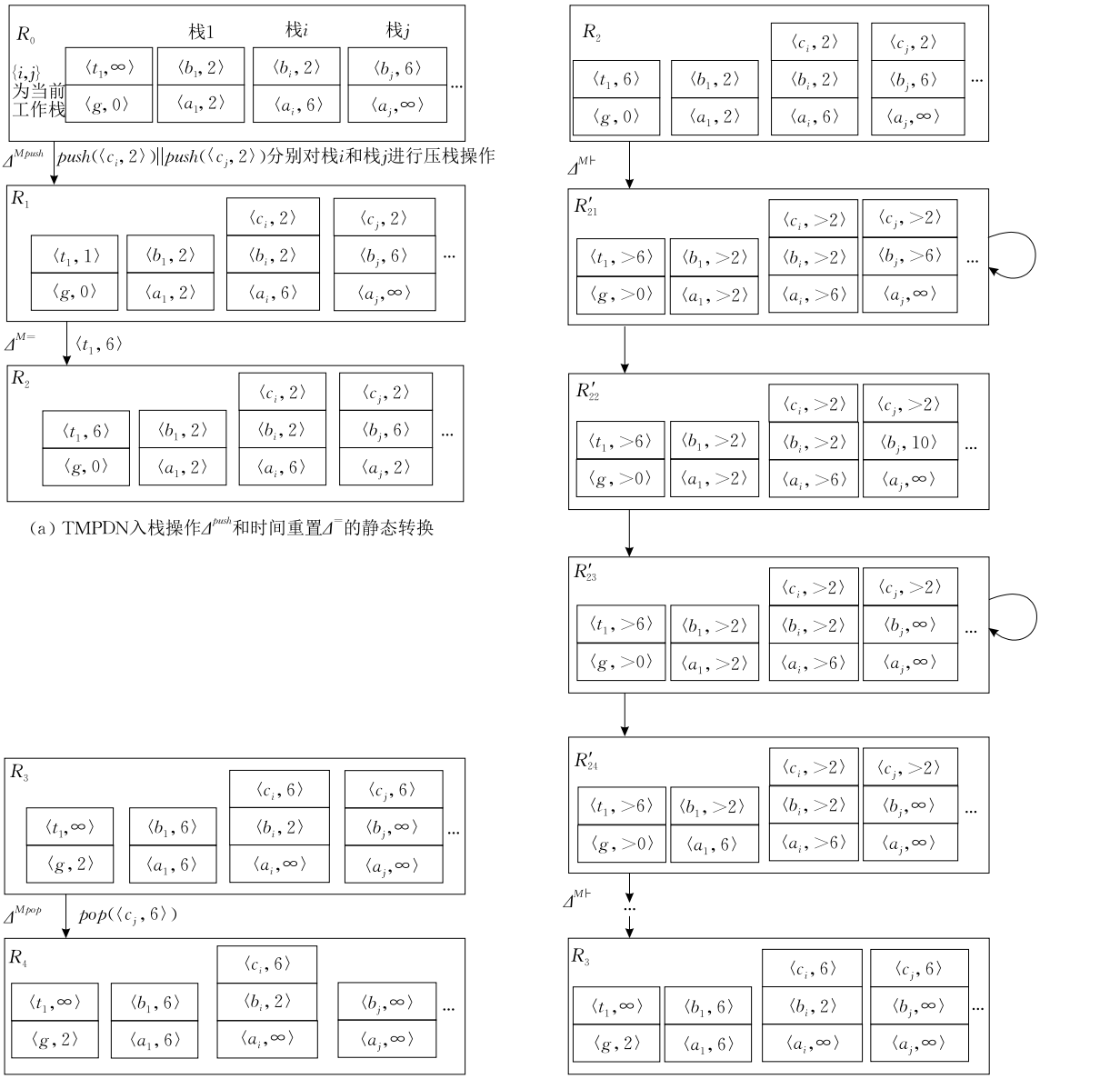
图 7(b)描述 TMPDN 时间流逝 $\Delta^{\vdash}$ 的静态转换，随着时间的流逝，R_2 进入 R'_{21}，即时钟值为大于关键点的区域，例如 $\langle c_i,>2\rangle$ 表示 c_i 时钟值大于时间关键点 2，小于下一个时间关键点 6；进入此状态后，随时间流逝循环，直到 b_j 时钟值到达新时间关键点 10，进入 R'_{22}；继续随时间流逝，进入 R'_{23}，b_j 时钟值大于时间关键点 10，变为∞；随时间流逝循环，直到 a_1 时钟值到达时间关键点 6，进入 R'_{24}；依次类推，经过时间流逝 3.8 后达到状态 R_3，此刻 $t_1,g,a_1,b_1,a_i,b_i,c_i,a_j,b_j,c_j$ 的关键点分别为∞，2，6，6，∞，2，6，∞，∞，6.

图 7(c)描述 TMPDN 出栈操作 Δ^{pop} 的静态转换，在MPDN中与之对应迁移关系 Δ^{Mpop} 为 $pop(\langle c_j,6\rangle)$，将项元素 $\langle c_j,6\rangle$ 出栈，此时 R_3 经过 Δ^{Mpop} 迁移至格局 R_4.

8.2　动态转换

动态转换的操作对象为 TMPDN 中栈顶及其下一层的栈内元素，同一栈内项元素按照对应时钟关键点剩余部分的大小关系依次存储. 初始域 R_1 与 TMPDN 中初始格局 C_0 相对应，其中 R_1 含有基础时钟字符项 $\langle\vdash,0\rangle$、标识基础时钟字符项 $\langle\vdash',0\rangle$、标识项(标识全局变量 $\langle g',0\rangle$、标识时钟 $\langle t_1',\infty\rangle$)以及栈内字符 a_i,a_j 的标识项 $\langle a_i',6\rangle$、$\langle a_j',\infty\rangle$. 具体如图 8 所示.

(1) 描述 TMPDN 入栈操作 Δ^{push} 的动态转换. 执行 MPDN 中与 TMPDN 对应的入栈迁移关系 $\Delta^{Mpush}:push(\langle c_i,2\rangle)\|push(\langle c_j,2\rangle)$，获得的新域 R_2，首先将 R_1 中的普通项 $\langle\vdash,0\rangle$、$\langle b_i,2\rangle$、$\langle b_j,6\rangle$、



(a) TMPDN入栈操作Δ^{push}和时间重置$\Delta^{=}$的静态转换

(c) TMPDN出栈操作Δ^{pop}的静态转换

(b) TMPDN时间流逝$\Delta^{\vdash}$的静态转换

图 7 TMPDN 静态转换后的 MPDN

$\langle g,0\rangle$、$\langle t_1,\infty\rangle$进行标识，然后将普通项标识后的标识项$\langle\vdash',0\rangle$、$\langle b_i',2\rangle$、$\langle b_j',6\rangle$、$\langle g',0\rangle$、$\langle t_1',\infty\rangle$以及基础时钟项、全局变量和全局时钟的普通项$\langle\vdash,0\rangle$、$\langle g,0\rangle$、$\langle t_1,\infty\rangle$保存至新域 R_2 中，最后将新入栈项$\langle c_i,2\rangle$、$\langle c_j,2\rangle$保存至新域 R_2. 具体如图 9 所示.

(2) 描述 TMPDN 中时钟重置操作 $\Delta^{=}$ 的动态转换. 依照关键点转换规则，在 MPDN 中将 t_1 赋值为 6，对 t_1 赋值时只将栈顶域 R_2 中 t_1 的普通项赋值为 6，如图 10 中 R_2 的项$\langle t_1,6\rangle$所示.

(3) 描述 TMPDN 中时间流逝 $\Delta^{\vdash}$ 的动态转换. 在时间流逝转换时，MPDN 栈内项在不同时间内可能处于不同关键点区域内，但此处只考虑满足迁移条件时栈内项所处的关键点区域，中间过程在此处不进行展开讨论，迁移转换时，基础时钟项$\langle\vdash,0\rangle$保持不变，其标识项$\langle\vdash',2\rangle$随时间变化，反应 TMPDN 模型中的时间流逝，在实例中其关键点区域为[2:6]. 在运行 $\Delta^{M\vdash}$ 迁移后，之前栈顶域内的项$\langle\vdash',0\rangle$、$\langle b_i',2\rangle$、$\langle b_j',6\rangle$、$\langle g,0\rangle$、$\langle g',0\rangle$、$\langle t_1,6\rangle$、$\langle t_1',\infty\rangle$、$\langle c_i,2\rangle$、$\langle c_j,2\rangle$，更新为项$\langle\vdash',2\rangle$、$\langle b_i',6\rangle$、$\langle b_j',\infty\rangle$、$\langle g,2\rangle$、$\langle g',2\rangle$、$\langle t_1,\infty\rangle$、$\langle t_1',\infty\rangle$、$\langle c_i,6\rangle$、$\langle c_j,6\rangle$，如图 11 所示.

(4) 描述 TMPDN 中出栈操作 Δ^{pop} 的动态转换. 利用组合域技术，将栈顶域 R_2 及下一层域 R_1 更新为域 R_3，如文中组合域所述，首先将栈顶域 R_2 中的非出栈的普通项$\langle\vdash,0\rangle$、$\langle g,2\rangle$、$\langle t_1,\infty\rangle$、$\langle c_i,6\rangle$以及栈顶下一层域 R_1 中的非全局时钟字符项$\langle\vdash',0\rangle$、

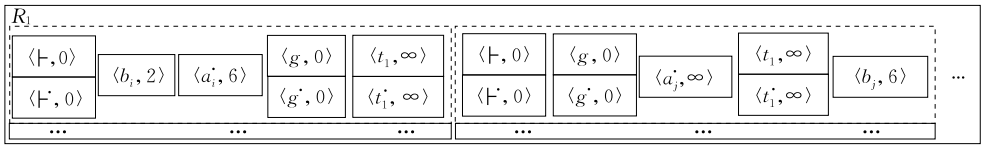


图 8　MPDN 初始域 R_1

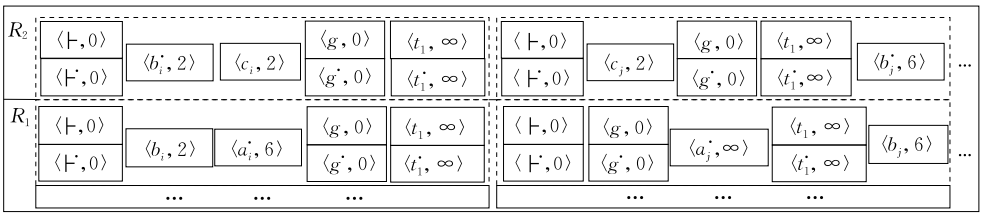


图 9　MPDN 入栈迁移 Δ^{Mpush}

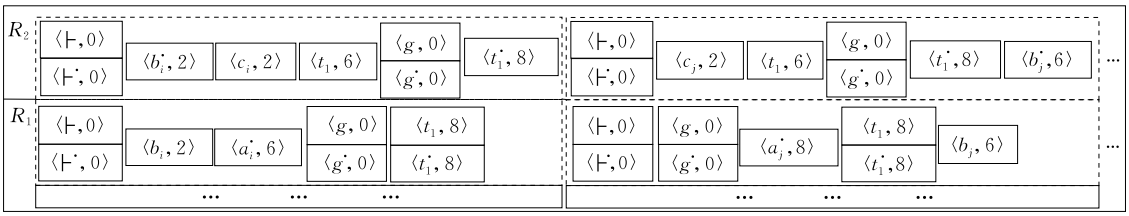


图 10　MPDN 时钟重置 $\Delta^{M=}$

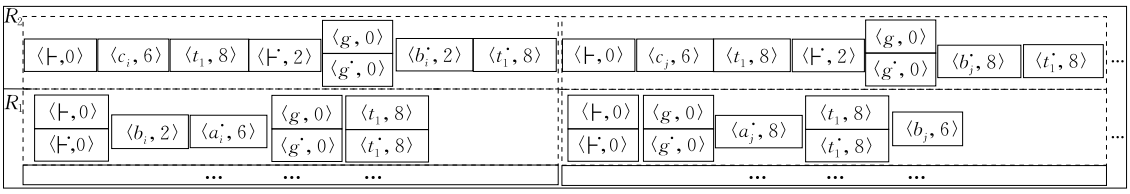


图 11　MPDN 时间流逝 $\Delta^{M\vdash}$

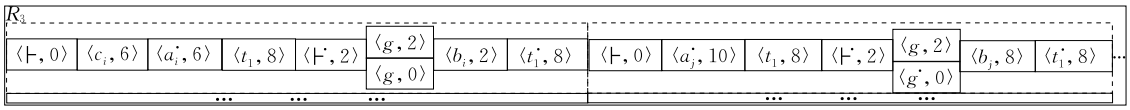


图 12　MPDN 出栈迁移 Δ^{Mpop}

$\langle b_i,2\rangle$、$\langle a_i^{\cdot},6\rangle$、$\langle b_j,\infty\rangle$、$\langle a_j^{\cdot},6\rangle$保存在组合域 R_3，之后组合域中所有项需考虑时间流逝的情况，而时间流逝的时钟值可参考栈顶域 R_2 中项$\langle\vdash^{\cdot},2\rangle$相对基础时钟项$\langle\vdash,0\rangle$增加的关键点. 最后转换后的结果如图 12 所示.

9　总　结

对实时并发系统形式化分析与验证一直是学界的研究难点. 之前诸多学者广泛采用时间自动机理论来分析实时并发程序，但这种方式对递归程序建模非常繁琐. 后来研究者提出了时间下推自动机理论，降低了分析实时递归程序的复杂度，但是该模型很难描述实时并发程序中的线程间交互. 本文将时间下推自动机模型进行了扩展，提出了时间多栈下推网络来形式化验证实时系统中的并发递归机制及线程间交互，为实时并发系统的分析验证提供了一种新的思路.

针对下推系统分析并发程序存在效率和完备性难以兼顾的问题，本文采用动态转换方法，可较好的对实时并发递归程序的线程间交互进行分析验证. 下一步工作，将开发验证工具，实现 TMPDN 到 MPDN 的自动转换，从模型建立到验证分析自动化.

参　考　文　献

[1] Alur R, Dill D. A theory of timed automata. Theoretical Computer Science, 1994, 126(2): 183-235

[2] Bengtsson J, Wang Y. Timed automata: Semantics, algorithms and tools//Proceedings of the 4th Advanced Course on Petri Nets. Eichstaat, Germany, 2004: 87-124

[3] Abdulla P A, Atig M F, Stenamn J. Adding time to pushdown automata//Proceedings of the 1st Workshop on

Quantities in Formal Methods. Paris, France, 2012: 1-16

[4] Abdulla P A, Atig M F, Stenamn J. Minimal cost reachability problem in priced timed pushdown systems//Proceedings of the 6th International Conference on Language and Automata Theory and Applications. Tarragona, Spain, 2012: 58-69

[5] Abdulla P A, Atig M F, Jari S. Computing optimal reachability costs in priced dense-timed pushdown automata//Proceedings of the 6th International Conference on Language and Automata Theory and Applications. Madrid, Spain, 2014: 62-75

[6] Abdulla P A, Atig M F, Stenman J. Zenoness for timed pushdown automata//Proceedings of the 15th International Workshop on Verification of Infinite-State Systems. Hanoi, Vietnam, 2013: 35-47

[7] Cai X, Ogawa M. Well-structured pushdown system: Case of dense timed pushdown automata//Proceedings of the 12th International Symposium on Functional and Logic Programming. Kanazawa, Japan, 2014: 336-352

[8] Trivedi A, Wojtczak D. Recursive timed automata//Proceedings of the 8th International Symposium on Automated Technology for Verification and Analysis. Singapore, 2010: 306-324

[9] Benerecetti M, Minopoli S, Peron A. Analysis of timed recursive state machines//Proceedings of the 17th International Symposium on Temporal Representation and Reasoning. Paris, France, 2010: 61-68

[10] Li G, Cai X, Ogawa M, Yuen S. Nested timed automata//Proceedings of the 11th International Conference on Formal Modeling and Analysis of Timed Systems. Buenos Aires, Argentina, 2013: 168-182

[11] Atig M F, Bollig B, Habermehl P. Emptiness of multi-pushdown automata is 2ETIME-complete//Proceedings of the 12th International Conference on Developments in Language Theory. Kyoto, Japan, 2008: 121-133

[12] Atig M F, Narayan K, Prakash S. Adjacent ordered multi-pushdown systems. International Journal of Foundations of Computer Science, 2014, 25(8): 1083-1096

[13] Torre S L, Madhusudan P, Parlato G. A robust class of context-sensitive languages//Proceedings of the 22nd IEEE Symposium on Logic in Computer Science. Wroclaw, Poland, 2007: 161-170

[14] Torre S L, Napoli M. Reachability of multi-stack pushdown systems with scope-bounded matching relations//Proceedings of the 22nd International Conference on Concurrency Theory. Aachen, Germany, 2011: 203-218

[15] Johan B, Wang Y. Timed automata: Semantics, algorithms and tools//Proceedings of the 4th Advanced Course on Petri Nets. Eichstätt, Germany, 2003: 87-124

[16] Abdulla P A, Atig M F, Stenman J. Dense-timed pushdown automata//Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science. Dubrovnik, Croatia, 2012: 35-44

[17] Musuvathi M, Qadeer S. Iterative context bounding for systematic testing of multithreaded programs//Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation. San Diego, USA, 2007: 446-455

[18] Cordeiro L, Fischer B. Verifying Multi-threaded software using smt-based context-bounded model checking//Proceedings of the 33rd International Conference on Software Engineering. Waikiki, USA, 2011: 331-340

[19] Atig M F. Model-checking of ordered multi-pushdown automata. Logical Methods in Computer Science, 2012, 8(3): 1-31

[20] Atig M F, Abdulla P A, Kumar K N, Saivasan P. Linear-time model-checking for multithreaded programs under scope-bounding//Proceedings of the 10th International Symposium on Automated Technology for Verification and Analysis. Thiruvananthapuram, India, 2012: 152-166

[21] Qadeer S, Rehof J. Context-bounded model checking of concurrent software//Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Edinburgh, UK, 2005: 93-107

[22] Abdulla P A, Atig M F, Rezine O, Stenman J. Multi-pushdown systems with budgets//Formal Methods in Computer-Aided Design. Cambridge, UK, 2012: 24-33

[23] Qian Jun-Yan, Jia Shu-Gui, Zhao Ling-Zhong, Guo Yun-Chuan. Context-bounded reachability analysis of recursive queue Concurrent Programs. Chinese Journal of Computers, 2014, 37(12): 2574-2585(in Chinese)
(钱俊彦, 贾书贵, 赵岭忠, 郭云川. 基于上下文定界的递归队列并发程序可达性分析. 计算机学报, 2014, 37(12): 2574-2585)

QIAN Jun-Yan, born in 1973, Ph.D., professor. His research interests include software engineering, model checking and program verification.

GAN Peng-Cheng, born in 1983, Ph. D. candidate. His research interests include software engineering, program analysis and verification.

GUO Yun-Chuan, born in 1977, Ph. D. His research interests include security evaluation and verification.

ZHAO Ling-Zhong, born in 1977, Ph. D., professor. His research interests include formal technique and software verification.

GU Tian-Long, born in 1964, Ph. D., professor, Ph. D. supervisor. His research interests include formal method, and symbolic computation.

Background

With the development of multi-core processors and intelligent devices, the research of real-time concurrent programs has drawn world-wide attention. The uncertainty of the interactions between two processors makes it difficult to find out hidden errors by traditional testing methods. This situation does occur in the execution of real-time concurrent programs. Model checking, as an automatic formal verification technology, has become an important method to verify real-time concurrent programs. Reachability analysis is the core technology of model checking. Currently, as a model for describing recursive concurrent programs, Multi-Pushdown Net (MPDN) has become a hot research area. Context-bound method is used by most researchers to describe concurrent programs to make the reachability problem decidable. This paper is not an exception. The key difference between real-time systems and non-real-time systems lies in the analysis of time factor authentication. This paper uses clock domain equivalence to make the continuous-time and discrete-time on integer domain equivalent. Specifically, TMPDN is first used to create models for concurrent programs, and then clock equivalence technology is applied to implement the conversion between TMPDN and MPDN. Finally, the existing verification tools, such as ESBMC, JMOPED, can be used to verify the concurrent program.

This work is supported by the National Natural Science Foundation of China under Grant Nos. 61262008, 61562015, 61572146, U1501252, the Guangxi Natural Science Foundation of China under Grant Nos. 2012GXNSFAA053220, 2014GXNSFAA118365, 2015GXNSFDA139038, the High Level Innovation Team of Guangxi Colleges and Universities and Outstanding Scholars Fund, Guangxi Key Laboratory of Trusted Software Focus Fund, Program for Innovative Research Team of Guilin University of Electronic Technology.