软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(6):1607−1621 [doi: 10.13328/j.cnki.jos.005471]

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

消息传递的 MSVL 通信机制及其实现*

王小兵, 郭文轩, 段振华

(西安电子科技大学 计算机学院,陕西 西安 710071)

通讯作者: 段振华, E-mail: zhhduan@mail.xidian.edu.cn

摘 要: 建模、仿真和验证语言(MSVL)是一种时序逻辑编程语言,它是投影时序逻辑(PTL)的可执行子集.MSVL和PTL可用于并发系统的建模和性质验证.然而,MSVL缺少一种消息传递的通信机制,这种机制对于并发分布式系统的建模和验证至关重要.说明了如何在MSVL中开发和实现合适的机制来对分布式系统进行建模和验证.该机制首先定义了通道结构,对通信语句和进程结构进行形式化描述;接着介绍了这些通信语句的实现机制;最后提供了一个关于电子合同签名协议的建模和验证实例,说明消息传递在MSVL中的工作原理.

关键词: 通道;消息传递;通信机制;PTL;时序逻辑程序设计

中图法分类号: TP311

中文引用格式: 王小兵,郭文轩,段振华.消息传递的 MSVL 通信机制及其实现.软件学报,2018,29(6):1607−1621. http://www.jos.org.cn/1000-9825/5471.htm

英文引用格式: Wang XB, Guo WX, Duan ZH. Communication mechanism and its implementation for MSVL based on message passing. Ruan Jian Xue Bao/Journal of Software, 2018,29(6):1607−1621 (in Chinese). http://www.jos.org.cn/1000-9825/5471.htm

Communication Mechanism and Its Implementation for MSVL Based on Message Passing

WANG Xiao-Bing, GUO Wen-Xuan, DUAN Zhen-Hua

(School of Computer Science and Technology, Xidian University, Xi'an 710071, China)

Abstract: The modeling, simulation and verification language (MSVL) is a temporal logic programming language as well as an executable subset of projection temporal logic (PTL). MSVL and PTL are used for modeling and verifying properties of concurrent systems. However, MSVL lacks a mechanism of communication based on message passing which is essential for modeling and verifying concurrent distributed systems. This paper shows how to develop and implement a suitable mechanism in MSVL to model and verify concurrent distributed systems. First, channel structure is defined while communication statements and process structures are formalized. Then, the implementation mechanisms for those communication statements are presented. Finally, a modeling and verification example involving an electronic contract signing protocol is provided to illustrate how the message passing works in MSVL.

Key words: channel; message passing; communication mechanisms; projection temporal logic; temporal logic programming

形式化验证已成功应用于从数字电路设计到软件工程等领域.在形式化验证中,时序逻辑是描述和推理并发系统性质的有效工具[1,2].投影时序逻辑(projection temporal logic,简称 PTL)[3]扩展了区间时序逻辑(interval temporal logic,简称 ITL)[4]并广泛应用于系统的规范和验证[5−7].在大多数情况下,系统建模技术与时序逻辑无关,而所需的性质则用时序逻辑公式来描述.所以,由于不同的逻辑语言有不同的符号和语义,而导致验证变得

* 基金项目: 国家自然科学基金(61672430, 61420106004, 61732013, 61402347); 中央高校基本科研业务费专项基金(JBG 160306)

Foundation item: National Natural Science Foundation of China (61672430, 61420106004, 61732013, 61402347); Fundamental Research Funds for the Central Universities (JBG160306)

本文由形式化方法的理论基础专题特约编辑傅育熙教授、李国强副教授、田聪教授推荐.

收稿时间: 2017-07-02; 修改时间: 2017-09-01; 采用时间: 2017-11-06; jos 在线出版时间: 2017-12-28

CNKI 网络优先出版: 2017-12-29 13:19:32, http://kns.cnki.net/kcms/detail/11.2560.TP.20171229.1318.012.html

困难.为了改善这种情况,一种方法是使用相同的语言对系统建模和描述性质.建模、仿真和验证语言(modeling, simulation and verification language,简称 MSVL)是一种时序逻辑程序设计语言,它是 PTL 的可执行子集,使用 MSVL 可以对并发系统进行形式化描述、仿真和验证[8].这种方法用 MSVL 程序来对并发系统建模,系统的性质则使用投影时序逻辑(propositional projection temporal logic,简称 PPTL)公式来描述.通过在同一时序逻辑框架内的模型检测方法,可以验证并发系统是否满足性质.

并发编程模型中,进程上最重要的两个问题就是进程通信和进程同步[9].指令式编程语言的通信模型通常基于共享变量或消息传递.在基于共享变量的模型中,同步通常是显式的:除非编程人员进行了特殊处理,否则接收进程在发送方写入变量之前可以读取变量的旧值.相反,在基于消息传递的模型中,同步通常是隐式的:消息必须发送后才能够被接收.如果一个进程试图接收尚未发送的消息,那么它将等待直到发送方发送消息.

目前,大多数时序逻辑编程语言都支持基于共享变量的通信机制,如 MSVL,XYZ/E[10],METATEM[4]和 Tempura[11].但是在这些机制下,并发系统的执行可能代价高昂且缺乏可靠性和可预测性[12].这是因为并行进程通过共享变量进行同步会有些棘手,可能会导致很多问题,比如互斥.特别是在若干节点都分布于不同地理位置的分布式系统中,允许一些变量或全部变量共享是不方便的.换言之,通过消息传递,多个节点之间的通信和同步会具有更好的特性.为了能够描述并发和分布式系统的行为并对其形式化建模和验证,我们将使用消息传递的通信机制来扩展 MSVL.

本文的理论贡献包括:

(1) 形式化的通道结构用来描述基于消息传递的进程间通信,它是传输消息的缓冲区,并且引入面包店算法[13]来解决互斥问题;
(2) 通信语句用来发送或接收进程执行后的消息;
(3) 进程结构被定义为 PTL 的组合谓词,用来描述系统的行为模式.因此,一个系统可以被建模为多个进程的组合;
(4) 提供了这些通信语句的实现机制.

上述概念为时序逻辑编程提供了一种可行的消息传递通信机制.MSVL 通过这种机制的扩展,就能用来对并发和分布式系统进行模拟、仿真和验证.

为了说明上述理论的实用性,我们将应用扩展的 MSVL 对电子合同签名协议[14]进行建模与验证.协议中的所有参与方都用进程描述,各方之间的通信都消息传递并且通过通道传输.然后,所有进程基于消息传递来运行和通信,以实现电子合同签名协议.因此,我们能够检查关注的系统性质是否满足指定协议,系统性质使用 PPTL 公式描述.本文将之前的工作[15]扩展到以下几个方面:首先,增强通道的功能,一个通道可以用于同步多个发送方和接收方,而之前的版本一次只能被一个进程访问;还制定了一组通信语句的锁定策略,以解决不同的访问进程之间的冲突.

本文第 1 节回顾 PTL 相关理论.第 2 节简要介绍 MSVL 语言.第 3 节形式化定义通道结构、消息传递的通信语句和进程结构.第 4 节给出通信语句的实现机制.第 5 节提供采用扩展 MSVL 对电子合同签名协议进行建模和验证的应用实例.第 6 节对比其他方法,分析新机制的优劣.最后,第 7 节得出结论.

1 投影时序逻辑

设 V 是可数的静态和动态变量的集合,Π是可数的命题集合,项 e 和公式 p 的归纳定义如下:

$$e ::= x \mid \bigcirc e \mid \Theta e \mid f(e_1,...,e_m)$$

$$p ::= \pi \mid e_1 = e_2 \mid P(e_1,...,e_m) \mid p_1 \wedge p_2 \mid p_1 \vee p_2 \mid \neg p \mid \exists v : p \mid Op \mid (p_1,...,p_m)\ prj\ p$$

上述语法中,$x \in V$ 是动态变量或者静态变量,$\pi \in \Pi$是命题,且 $m \geq 1$.在 $f(e_1,\ldots,e_m)$与 $P(e_1,\ldots,e_m)$中,f 是函数,P 是谓词.每个函数和谓词都有一个固定的参数数量,其范围是非负整数.注意,参数数量为 0 的函数即常量.一个公式(项)如果不包含任何时态运算符(即$\bigcirc$,Θ或 prj)就称为状态公式(项);否则,它就是一个时序公式(项).

状态 s 是一个二元组(I_v,I_p),对于每个变量 v 有 $s[x]=I_v[x]$,每个命题π有 $s[\pi]=I_p[\pi].I_v[x]$是集合 $D=\mathbb{Z}\cup\mathbb{S}\cup\mathbb{L}$中

的值或 *nil*(表示未定义值),$\mathbb{Z}$表示整数集,$\mathbb{S}$={0,…,9,*a*,…,*z*,*A*,…,*Z*}*表示字符串集,而$\mathbb{L}$={$\langle l_0,\ldots,l_n\rangle|l_i\in\mathbb{Z}\cup\mathbb{S},n\in N_0$}表示列表集合.$I_p[\pi]$是集合 *B*={true,false}中的值.区间$\sigma=\langle s_0,s_1,\ldots\rangle$是一个非空(可能无穷大)的状态序列.区间长度记为$|\sigma|$,当$\sigma$为无穷区间时,$|\sigma|$为$\omega$,否则为其状态数减 1.为了统一有穷和无穷区间的符号,我们使用扩展整数作为索引.也就是说,我们考虑自然数集合$\mathbb{N}$和ω,即$\mathbb{N}_\omega=\mathbb{N}\cup\{\omega\}$,并且扩展比较运算符=,<,≤到$\mathbb{N}_\omega$,考虑到$\omega=\omega$,且对于所有 $i\in\mathbb{N}$,有 $i<\omega$.$\preceq$定义为$\leqslant-\{\omega,\omega\}$.通过这些符号,用$\sigma_{(i\ldots j)}(0\leqslant i\preceq j\leqslant|\sigma|)$表示子区间$\langle s_i,\ldots,s_j\rangle$,$\sigma(k)(0\leqslant k\preceq|\sigma|)$表示$\langle s_k,\ldots,s_{|\sigma|}\rangle$.用$\sigma\cdot\sigma'$来表示$\sigma$和另一区间$\sigma'$的连接.为了定义投影操作的语义,我们需要一个辅助运算符来进行区间运算.设$\sigma=\langle s_0,s_1,\ldots\rangle$是一个区间,$n_0,\ldots,n_h(h\geqslant 0)$是整数且有 $0\leqslant n_0\leqslant n_1\leqslant\ldots\leqslant n_h\preceq|\sigma|$.则$\sigma$在 $n_0,\ldots,n_h$ 上的投影是一个区间(称作投影区间),即 $\sigma\downarrow(n_0,\ldots,n_h)=\langle s_{m_0},\ldots,s_{m_k}\rangle$,其中,$m_0,\ldots,m_k$ 是通过删除 $n_0,\ldots,n_h$ 中的所有冗余元素得到的,比如:$\langle s_0,s_1,s_2,s_3,s_4\rangle\downarrow\langle 0,0,2,2,2,3\rangle=\langle s_0,s_2,s_3\rangle$.

PTL 项或者公式的解释$\mathcal{I}=(\sigma,i,k,j)$是一个四元组,$\sigma=\langle s_0,s_1,\ldots\rangle$是一个区间,*i* 和 *k* 都是非负整数,*j* 是整数或者ω,且 $i\leqslant k\preceq j\leqslant|\sigma|$.我们用$(\sigma,i,k,j)$表示一个项或公式在一个当前的状态是 s_k 的子区间$\sigma_{(i\ldots j)}$上被解释.对于每个项 *e*,*e* 的赋值和解释$\mathcal{I}=(\sigma,i,k,j)$相关,表示为$\mathcal{I}[e]$,它是集合 *D* 中的值.对项的解释归纳定义如下所示:

$$\mathcal{I}[x]=s_k[x]=I_v^k[x]$$

$$\mathcal{I}[\mathrm{O}e]=\begin{cases}(\sigma,i,k+1,j)[e], & \text{如果}k<j\\ nil, & \text{否则}\end{cases}$$

$$\mathcal{I}[\Theta e]=\begin{cases}(\sigma,i,k-1,j)[e], & \text{如果}i<k\\ nil, & \text{否则}\end{cases}$$

$$\mathcal{I}[f(e_1,\ldots,e_m)]=\begin{cases}\mathcal{I}[f](\mathcal{I}[e_1],\ldots,\mathcal{I}[e_m]), & \text{如果对}1\leqslant h\leqslant m,\text{有}\mathcal{I}[e_h]\neq nil\\ nil, & \text{否则}\end{cases}$$

解释与公式间的满足关系归纳定义如下.

(1) $\mathcal{I}\vDash\pi$当且仅当 $s_k[\pi]=I_p^k[\pi]=\text{true}$;

(2) $\mathcal{I}\vDash e_1=e_2$ 当且仅当$\mathcal{I}(e_1)=\mathcal{I}(e_2)$;

(3) $\mathcal{I}\vDash P(e_1,\ldots,e_m)$当且仅当对 $1\leqslant h\leqslant m$,有$\mathcal{I}[e_h]\neq nil$ 并且$\mathcal{I}[P](\mathcal{I}[e_1],\ldots,\mathcal{I}[e_m])$=true;

(4) $\mathcal{I}\vDash\neg p$ 当且仅当$\mathcal{I}\nvDash p$;

(5) $\mathcal{I}\vDash p_1\wedge p_2$ 当且仅当$\mathcal{I}\vDash p_1$ 并且$\mathcal{I}\vDash p_2$;

(6) $\mathcal{I}\vDash\exists v{:}p$ 当且仅当存在区间σ',满足$|\sigma'|=|\sigma|$,$(\sigma',i,k,j)\vDash p$,并且σ和σ'除了在状态 *k* 对 *v* 的解释可以不同外,对其他变量的解释都相同;

(7) $\mathcal{I}\vDash\mathrm{O}p$ 当且仅当 $k<j$ 并且$(\sigma,i,k+1,j)\vDash p$;

(8) $\mathcal{I}\vDash(p_1,\ldots,p_m)\ prj\ q$,若存在整数 $k=k_0\leqslant\ldots\preceq k_m\leqslant j$ 使$(\sigma,i,k_0,k_1)\vDash p_1,(\sigma,k_{h-1},k_{h-1},k_h)\vDash p_h$(对 $1<h\leqslant m$)并且对下面的某种情况有$(\sigma',0,0,|\sigma'|)\vDash q$:

- $k_m<j$ 并且 $\sigma'=\sigma\downarrow(k_0,\ldots,k_m)\cdot\sigma_{(k_m+1\cdots j)}$;
- $k_m=j$ 并且$\sigma'=\sigma\downarrow(k_0,\ldots,k_h)(0\leqslant h\leqslant m)$.

公式 *p* 被称为满足区间σ当且仅当$(\sigma,0,0,|\sigma|)\vDash p$.如果在某些$\sigma$上有$(\sigma,0,0,|\sigma|)\vDash p$,那么 *p* 是可满足的.如果对于所有的$\sigma$有$(\sigma,0,0,|\sigma|)\vDash p$,那么 *p* 是有效的,记为$\vDash p$.公式 *p* 和 *q* 是等价的,当且仅当$\vDash(p\leftrightarrow q)$,记为 $p\equiv q$.

连接符$\vee$,$\rightarrow$和$\leftrightarrow$的定义和经典逻辑中的定义相同.特别地,对于任意公式 *p*,有 true$\triangleq p\vee\neg p$ 和 false$\triangleq p\wedge\neg p$.导出公式和复合谓词如下所示.

empty$\triangleq\neg$Otrue　　*more*$\triangleq\neg$*empty*

$p;q\triangleq(p,q)\ prj\ empty$　　$\Diamond p\triangleq$true;p

$\Box p\triangleq\neg\Diamond\neg p$　　*halt*$(p)\triangleq\Box(empty\leftrightarrow p)$

$keep(p)\triangleq\Box(\neg empty\rightarrow p)$ $fin(p)\triangleq\Box(empty\rightarrow p)$

$skip\triangleq O empty$ $p^*\triangleq empty\vee(p;p^*)\vee p\wedge\Box more$

$len(0)\triangleq empty$ $len(n)\triangleq O len(n-1)(n>0)$

2 建模、仿真和验证语言 MSVL

MSVL 是 PTL 的一个可执行子集,使用 MSVL 可以对并发系统进行建模.MSVL 的算术表达式 e 和布尔表达式 b 都是 PTL 的项,语法定义如下:

$$op ::= + \mid - \mid * \mid / \mid \mathrm{mod}$$
$$e ::= n \mid x \mid Ox \mid \Theta x \mid \max(e_0,...,e_m) \mid \min(e_0,...,e_m) \mid e_0\ op\ e_1$$
$$b ::= \mathrm{true} \mid \mathrm{false} \mid e_0 = e_1 \mid e_0 < e_1 \mid \neg b \mid b_0 \wedge b_1$$

其中,n 是整数,x 是变量.p,q 是 PTL 公式,MSVL 语句或程序定义如下.

- 整型声明语句:$int(x)\triangleq x\in\mathbb{Z}$;
- 字符串声明语句:$str(x)\triangleq x\in\mathbb{S}$;
- 列表声明语句:$list(x)\triangleq x\in\mathbb{L}$;
- 立即赋值语句:$x\Leftarrow e\triangleq x=e\wedge\pi_x$;
- 下一状态赋值语句:$x:=e\triangleq skip\wedge Ox\Leftarrow e$;
- 顺序语句:$p;q$;
- 条件语句:if b then p else $q\triangleq(b\rightarrow p)\wedge(\neg b\rightarrow q)$;
- While 语句:while b do $p\triangleq(b\wedge p)^*\wedge\Box(empty\rightarrow\neg b)$;
- 合取语句:$p\wedge q$;
- 选择语句:$p\vee q$;
- 并行语句:$p\|q\triangleq p\wedge(q;\mathrm{true})\vee q\wedge(p;\mathrm{true})$;
- Next 语句:Op;
- Always 语句:$\Box p$;
- 区间长度语句:$empty,skip,len(n)$;
- 存在语句:$\exists x:p$;
- 状态框架语句:$lbf(x)\triangleq\neg af(x)\rightarrow\exists b:(\Theta x=b\wedge x=b)$;
- 区间框架语句:$frame(x)\triangleq\Box(more\rightarrow O lbf(x))$;
- 投影语句:$(p_1,\ldots,p_m)\ prj\ q$;
- 等待语句:$await(b)\triangleq(frame(x_1)\wedge\ldots\wedge frame(x_h))\wedge\Box(empty\leftrightarrow b)$,其中,$x_i\in V_b=\{x\mid x$ 在 b 中$\}$.

上述语句中,$int(x),str(x),list(x),x\Leftarrow e,x:=e,empty,skip,len(n),frame(x),await(b)$是基础语句,而其他的则是复合语句.声明语句 $int(x),str(x),list(x)$分别表示 x 的值为整数、字符串、列表.MSVL 有两种赋值语句.

- 立即赋值语句 $x\Leftarrow e$ 将 e 的值赋给变量 x,并将特殊命题 π_x 置为 true.对于每个变量 x,用 π_x 来指示 x 是否被赋值,因此也称作 x 的赋值标志.赋值标志可以用来实现重要的框架技术,该技术保持变量从一个状态到下一状态的值不变[16];
- 另一种赋值方法是下一状态赋值 $x:=e$,应用于长度为 1 的区间,即只有两个状态.下一状态赋值将前一个状态中 e 的值赋给第 2 个状态中的 x,并将相应的赋值标志设置为 true.

区间长度语句 $empty,skip,len(n)$采取了 PTL 公式的形式,分别表示区间长度为 0,1,n.状态框架语句 $lbf(x)$表示变量 x 若未赋值,则其当前值与前一状态值相同.区间框架语句 $frame(x)$表示变量 x 在区间上保持值不变,除非它被显式地赋值.等待语句 $await(b)$中,表达式 b 包含若干变量,当其他进程改变这些变量使得表达式为真时,该

进程才跳过等待语句继续执行.因此,等待语句可以实现进程间的同步.复合语句中,$\exists x{:}p$ 表示局部变量 x 的范围为 p.Op 和$\Box p$ 分别表示 p 的值在当前区间的下一状态或所有状态上成立.合取语句 $p \wedge q$ 表示 p 和 q 并行执行,在共同执行期间共享相同的区间和变量,而选择语句 $p \vee q$ 表示 p 或者 q 执行.和合取语句相似,并行语句 $p \| q$ 表示 p 和 q 并行执行,但是允许 p 和 q 在不同的时间节点终止.也就是说,p 的区间可能只是 q 区间的一个前缀,反之亦然.例如, $len(3) \| len(4)$是可满足的,而 $len(3) \wedge len(4)$是不可满足的.顺序语句 $p;q$ 表示 p 和 q 顺序执行.if b then p else q 和 while b do p 语句的含义与传统命令式编程语言相同.

为了避免括号数量过多,给出操作符优先规则见表 1.数字越小的运算符具有更高的优先级,而数字相同的运算符具有相同的优先级.

Table 1 Precedence rules of MSVL

表 1 MSVL 操作符优先级

优先级	操作符	优先级	操作符	优先级	操作符
1	¬	2	OΘ□	3	*/mod
4	+–	5	<≤>≥=≠	6	∃
7	:=⇐	8	∧	9	∨
10	→↔	11	*prj*	12	;

MSV 平台是用 Visual C++实现的,有建模、仿真和验证 3 种工作模式.第 1 种模式下,用 MSVL 程序来描述系统并在平台中执行.系统的所有模型都会以范式图(normal form graph,简称 NFG)[16]的形式给出.如图 1(a)所示,NFG 中的一条路径终止于双环节点.仿真类似于建模,但是只输出 NFG 的一条路径,即程序的一个模型.只要给出描述系统的 MSVL 程序和描述期望性质的 PPTL 公式,平台就能自动验证系统是否满足性质.如果系统不满足性质,还会给出反例路径.如图 1(b)所示,NFG 中不满足性质的路径终止于终止节点;而如图 1(c)中,NFG 中满足性质的路径终止于圆节点.

(a) (b) (c)

Fig.1 Three types of nodes

图 1 节点的 3 种类型

3 基于消息传递的通信

本节中,我们为 MSVL 开发了一种消息传递的通信机制,该机制的核心概念是通道、通信语句以及进程.

3.1 通 道

通道的概念可以用来描述同步和异步通信[17].在同步通信中,发送方通过通道发送消息后等待,直到接收方接收消息.而异步通信中,通道在通信实体之间起一个消息缓冲区的作用,这样,一个通信实体可以在不考虑其他通信实体的情况下发送或接收消息.

MSVL 中,消息传递的通信机制的非形式化描述如下:两个 MSVL 进程需要相互通信,第 1 个进程是发送方

而第 2 个是接收方.两个进程之间有一个通道,该通道被视为一个消息缓冲区,这是一个有界先入先出(first-in-first-out,简称 FIFO)队列.通道变量可以作为一个实参在两个进程间传输,因此,进程都可以通过访问它来进行通信.当发送方发送消息时,它检查通道中是否至少有一个可用空间:如果是,那么将信息附加到通道的队尾;否则它可以等待,直到有可用空间来发送消息,或撤销发送操作.当接收方准备接收消息时,它检查通道中是否存在至少有一条消息:如果是,那么将该消息从通道队头移除;否则它可以等待,直到新消息进入通道,或者撤消接收操作.

有界 FIFO 队列可以由一个列表来实现[18].受此思想启发,我们定义了以下符号.

$$|l| \triangleq \begin{cases} 0, & \text{如果} l = \langle\rangle \\ n, & \text{如果} l = \langle l_1,...,l_n\rangle, n \geqslant 1 \\ nil, & \text{否则} \end{cases}$$

$$head(l) \triangleq \begin{cases} l_1, & \text{如果} l = \langle l_1,...,l_n\rangle, n \geqslant 1 \\ nil, & \text{否则} \end{cases}$$

$$tail(l) \triangleq \begin{cases} \langle\rangle, & \text{如果} l = \langle l_1\rangle \\ \langle l_2,...,l_n\rangle, & \text{如果} l = \langle l_1,l_2,...,l_n\rangle, n > 1 \\ nil, & \text{否则} \end{cases}$$

一个通道上的进程之间基于消息传递的通信如图 2 所示.

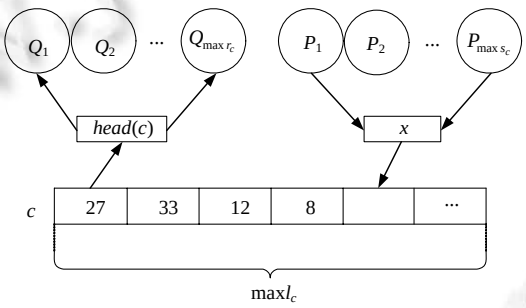


Fig.2 Channel structure

图 2 通道结构

通道 c 有 $\max l_c$ 大小的空间来存储发送方和接收方之间的消息.通信最多允许 $\max s_c$ 个发送进程 $P_1,...,P_{\max s_c}$ 通过向通道 c 的队尾写入值来发送消息.另一方面,最多允许 $\max r_c$ 个接收进程 $Q_1,...,Q_{\max r_c}$ 通过 $head(c)$从通道 c 的队头中读出值来接收消息.Lamport 的面包店算法用于解决多个进程同时通过通道发送消息或接收消息时的互斥问题.面包店锁无死锁并满足互斥,且有先到先得的特性:每个进程获取一个号码,然后等待,直到前面没有号码比自己小的进程试图进入临界区.如果这些进程读取相同的最大标签并选择了相同的标签,那么进程 ID 小的将获得锁.通道的形式化定义如下:

$$chn\ c(n,s,r) \triangleq frame(c, sflag_c, slabel_c) \wedge frame(rflag_c, rlabel_c) \wedge c = \langle\rangle \wedge \bigwedge_{i=1}^{s}(sflag_c[i] = 0 \wedge slabel_c[i] = 0) \wedge$$

$$\bigwedge_{i=1}^{r}(rflag_c[i] = 0 \wedge rlabel_c[i] = 0) \wedge \max l_c = n \wedge \max s_c = s \wedge \max r_c = r,$$

其中,*chn* 是 *channel* 的保留字,*chn* $c(n,s,r)$表示一个通道(即一个列表)名为 c 容量为 n,最多可以被 s 个发送方和 r 个接收方访问.这 3 个值分别被设置为 3 个静态变量 $\max l_c$,$\max s_c$ 和 $\max r_c$.这里的 c 是框架变量.面包店算法中,ID 为 i 的进程标志 $sflag_c[i]$是用来指示进程是否要进入临界区发送消息,而 $slabel_c[i]$是整数,用于指示进程进入临界区接收消息时的相对顺序.在初始状态,所有进程的 $sflag_c$ 和 $slabel_c$ 都设置为 0.$rflag_c$ 和 $rlabel_c$ 在进程从通道接收消息时起类似的作用.

3.2 通信语句

为了判断一个进程的锁的状态,定义了两个谓词如下.

- $isslocking(i,c)\triangleq sflag_c[i]=1$;
- $isrlocking(i,c)\triangleq rflag_c[i]=1$.

这里 $isslocking(i,c)$表示 ID 为 i 的进程正在获取或已经获取通道 c 的发送锁,$isrlocking(i,c)$表示 ID 为 i 的进程正在获取或已经获取通道 c 的接收锁.

面包店算法发送和接收信息的锁形式化定义 6 条语句如下所示:

$$slock(i,c)\triangleq sflag_c[i]:=1\wedge slabel_c[i]:=\max(slabel_c[1],...,slabel_c[\max s_c])+1;$$

$$await\left(\neg\left(\bigvee_{j=1}^{i-1}(sflag_c[j]=1\wedge slabel_c[j]\Leftarrow slabel_c[i])\vee\bigvee_{j=i+1}^{\max s_c}(sflag_c[j]=1\wedge slabel_c[j]<slabel_c[i])\right)\right)$$

$$tryslock(i,c)\triangleq sflag_c[i]:=1;\ \text{if}\left(\bigwedge_{j=1,j\neq i}^{\max s_c}(sflag_c[j]=0)\right)\text{ then }\{skip\}\text{ else }\{sflag_c[i]:=0\}$$

$$unslock(i,c)\triangleq sflag_c[i]:=0;$$

$$rlock(i,c)\triangleq rflag_c[i]:=1\wedge rlabel_c[i]:=\max(rlabel_c[1],...,rlabel_c[\max r_c])+1;$$

$$await\left(\neg\left(\bigvee_{j=1}^{i-1}(rflag_c[j]=1\wedge rlabel_c[j]\Leftarrow rlabel_c[i])\vee\bigvee_{j=i+1}^{\max r_c}(rflag_c[j]=1\wedge rlabel_c[j]<rlabel_c[i])\right)\right)$$

$$tryrlock(i,c)\triangleq rflag_c[i]:=1;\ \text{if}\left(\bigwedge_{j=1,j\neq i}^{\max r_c}(rflag_c[j]=0)\right)\text{ then }\{skip\}\text{ else }\{rflag_c[i]:=0\}$$

$$unrlock(i,c)\triangleq rflag_c[i]:=0.$$

每次 ID 为 i 的进程调用 $slock(i,c)$为发送消息获取锁时,将 $sflag_c[i]$置为 1,并生成一个比最大标签大 1 的 $slabel_c[i]$,这就确定该进程相对于其他尝试获取锁的进程的顺序.当多个发送进程同时执行 *slock* 时,拥有最小标签的进程获得锁.如果这些进程读取到相同的最大标签并选择相同的标签,那么进程 ID 最小的进程将获得锁.这种情况由 *slock* 中的 *await* 语句来定义.*slock* 直到进程调用它获得锁后才会终止.进程发送消息后,通过执行 *unslock* 来释放锁.相反,如果没有其他进程尝试获取锁时,*tryslock* 才会将 $sflag_c$ 置为 0 并获得发送锁;否则,将 $sflag_c$ 置为 0 并返回.谓词 *isslocking* 可用于判断通道的发送锁的状态:如果它为真,那么进程能够发送消息;否则,进程需要再次调用 *tryslock* 对通道加锁.因此,使用 *tryslock* 和 *isslocking* 可以获取具有时间约束的锁,而 *slock* 没有这种能力.当进程尝试从通道接收消息,*rlock*,*tryrlock*,*isrlocking* 和 *unrlock* 的用法同前面发送锁的语句一样.

获取通道的发送锁后,如果通道未满,发送方可以使用通信语句将消息发送到通道.同样,在获得通道的接收锁后,如果通道不为空,那么接收方可以接收消息.为了判断通道的状态,定义了两个谓词如下.

- $isfull(c)\triangleq|c|=\max l_c$;
- $isempty(c)\triangleq|c|=0$.

$isfull(c)$在通道 c 为满时值为真,否则为假;而 $isempty(c)$在通道 c 为空时值为真,否则为假.

通信语句 *send* 和 *receive* 对于同步和异步通信都适用:在前者中,发送方必须要收到接收方发送到通道中的应答消息,而后者并不需要.*send* 和 *receive* 定义如下.

- $send(c,x)\triangleq await(\neg isfull(c));c:=c\cdot\langle x\rangle$;
- $recevie(c,y)\triangleq await(\neg isempty(c));y:=head(c)\wedge c:=tail(c)$.

$send(c,x)$语句将会阻塞直到通道 c 最终有空位为止.$await(\neg isfull(c))$在 c 未满时才会返回,然后,x 的值会在下一状态被附加到 c 的队尾.$recevie(c,y)$语句将会阻塞直到通道 c 最终有消息为止.$await(\neg isempty(c))$在 c 为非空是才会返回,然后,通道 c 队头的值会在下一状态被分配给 y.如果谓词 *isfull* 在开始状态为假,*send* 的区间长度至少为 2.但是如果 *isfull* 总是为真,*send* 将会一直阻塞,区间长度无穷长.同样的,*recevie* 的最小区间长度也是 2,区间长度在 *isempty* 总为真的情况下也是无穷长.

在分布式并发系统中,当接收方在限定时间内没有接收到消息时,它将放弃等待.发送方也有类似的情况.根据 *send* 和 *recevie* 的定义,它们的区间可能无限长,所以它们不适合用于分布式并发系统的建模.因此,定义另一对通信语句来处理超时机制,如下:

- $put(c,x)\triangleq$if $(\neg isfull(c))$ then $\{c:=c\cdot\langle x\rangle\}$ else $\{skip\}$;
- $get(c,y)\triangleq$if $(\neg isempty(c))$then $\{y:=head(c)\wedge c:=tail(c)\}$ else $\{skip\}$.

put 和 *get* 语句用 if-else 结构测试通道状态,避免无限制时间的等待.如果当前状态 *isfull* 为真,*put* 语句中的 *skip* 执行且 *put* 返回;如果当前状态 *isempty* 为真,*get* 语句中的 *skip* 执行且 *get* 返回.*put* 和 *get* 可以在限定时间内执行,否则返回.这对通信语句适合用于具有时间约束的分布式并发系统的建模,而另一对通信语句的发送和接收则容易应用于其他系统进行建模.

3.3 进 程

通常来说,进程可以用来描述对象的行为模式[19].在 MSVL 中,进程被定义为一个复合谓词:

$$proc\ ProcName(fid,x_1,\ldots,x_n)=ProcBody.$$

其中,

$$\begin{aligned}ProcBody \triangleq\ & int(x)\mid str(x)\mid list(x)\mid x\Leftarrow e\mid x:=e\mid \text{if } b \text{ then } P_1 \text{ else } P_2\mid \text{while } b \text{ do } P\mid P_1\wedge P_2\mid P_1\vee P_2\ \mid\bigcirc P\mid\Box P\mid\\ & empty\mid skip\mid len(n)\mid P_1;P_2\mid\exists x:P\mid lbf(x)\mid\ frame(x)\mid P_1\|P_2\mid(p_1,\ldots,p_m)\ \ prj\ q\mid await(b)\mid\\ & chn\ c(n,s,r)\mid slock(fid,c)\mid tryslock(fid,c)\mid send(c,e)\mid receive(c,y)\mid put(c,e)\mid get(c,y)\end{aligned}$$

$$ProcName(aid,y_1,\ldots,y_n)\triangleq(aid,y_1,\ldots,y_n/fid,x_1,\ldots,x_n)ProcBody.$$

这里,*ProcBody* 就是进程 *ProcName* 声明的主体.进程的结构由声明部分和调用部分组成.

- 声明部分 $proc\ ProcName(fid,x_1,\ldots,x_n)=ProcBody$ 包括进程名 *ProcName*、形参列表 $fid,x_1,\ldots,x_n$ 以及进程主体 *ProcBody*,它是一个通常的 MSVL 的声明或通信语句.特殊地,第一个形参 *fid* 是一个表示进程 ID 的自然数;
- 调用部分 $ProcName(aid,y_1,\ldots,y_n)$定义了一个进程的调用,这是通过将进程主体 *ProcBody* 中的形参 $fid,x_1,\ldots,x_n$ 分别替换为实参 $aid,y_1,\ldots,y_n$ 得到的.

值得一提的是:除了一般的程序语句之外,一个进程还可能涉及与其他进程通信的通信语句.通过互相通信的进程组合,可以描述复杂的并发系统.

4 通信的实现机制

本节重点介绍在 MSV 平台中具有重要作用的通信语句的实现机制.为了以严格的方式对程序进行形式化验证和分析,需要对通信语句进行归约.语句或程序的归约包括两个阶段:状态归约和区间归约[20].状态归约主要是指如何将程序转换成一个状态内的范式,而区间归约则涉及程序从一种状态到另一种状态的执行.在 MSVL 中引入通信机制,只需要考虑通信语句在状态归约时如何转换为范式,区间归约与一般语句的处理相同.

定义 1. MSVL 程序 q 是范式,如果:

$$q\triangleq\left(\bigvee_{i=1}^{k}q_{ei}\wedge empty\right)\vee\left(\bigvee_{j=1}^{h}q_{cj}\wedge\bigcirc q_{fj}\right),$$

其中,$k+h\geqslant 1$,且有以下规则:

(1) q_{ei} 和 q_{cj} 为 true,或形为 $p_1\wedge\ldots\wedge p_m,p_l(1\leqslant l\leqslant m)$是 $type(x)$,其中,$x\in V,type\in\{int,str,list\}$或 $x=e$,其中,$e\in D$ 或 π_x 或$\neg\pi_x$;

(2) q_{fj} 为内部程序,即:一个变量可以引用前一种状态,但不超过该程序当前执行区间的第 1 状态.

MSVL 程序 q 当 $k+h=1$ 成立时是确定,否则 $k+h>1$ 成立.我们将合取 $q_{ei}\wedge empty$ 和 $q_{cj}\wedge\bigcirc q_{fj}$ 称为基本积:前者称为终止分量,后者称为将来分量.此外,我们将执行在当前状态下的 q_{ei} 和 q_{cj} 称为当前部分,执行于后继状态的 $\bigcirc q_{fj}$ 则称为将来部分.任何 MSVL 程序包括通信语句都能可以归约为一个逻辑等价的范式.

定理 1. 设 P 是扩展了类型声明和通信语句(如通道声明、加锁语句、通信语句、进程声明和调用)的 MSVL 程序.存在一个程序 Q 是范式,使得 $P \equiv Q$.

证明:通过对语句结构的归纳能得出证明.对于 MSVL 的无类型声明、通道声明、加锁语句、通信语句、进程声明和调用的证明可以在文献[16,21]中找到.

整型声明 $int(x)$的证明如下所示:

$$\begin{aligned} int(x) &\equiv int(x) \wedge (empty \vee \neg empty) && \text{(i)} \\ &\equiv int(x) \wedge empty \vee int(x) \wedge \neg empty && \text{(ii)} \\ &\equiv int(x) \wedge empty \vee int(x) \wedge \bigcirc \text{true} && \text{(iii)} \end{aligned}$$

上述证明过程中,公式(i)遵从文献[20]中的 T1、公式(ii)文献[21]中的定理 2.1、公式(iii)以及 *empty* 的定义.另外两个类型声明 $str(x)$和 $list(x)$可以用相同的方法来证明. □

通道声明 *chn*、加锁语句 *slock*,*tryslock*,*unslock*,*rlock*,*tryrlock* 和 *unrlock*、通讯语句 *seng*,*recevie*,*put* 和 *get* 都能根据它们的定义被重写为等价的 MSVL 语句,然后转换为等价的范式.同样,进程调用语句 $ProcName(aid, y_1,\ldots,y_n)$能被$(aid,y_1,\ldots,y_n/fid,x_1,\ldots,x_n)ProcBody$ 替代,且能归约为等价的范式.

我们在 MSV 平台上实现了这些通信语句.考虑如下所示的实例程序:

$proc\ p(id,c) = \{exists\ x : \{\ frame(x)\ \text{and}\ x = id + 1\ \text{and}\ (slock(id,c); send(*c,x); unslock(id,c))\}\};$

$proc\ q(id,c) = \{exists\ y : \{\ frame(y)\ \text{and}\ y = id + 3\ \text{and}\ (rlock(id,c); receive(*c,y); unrlock(id,c))\}\};$

$chn\ c(1,2,2)\ \text{and}\ (p(1,\&c) \| p(2,\&c) \| q(1,\&c) \| q(2,\&c)).$

该程序说明了在容量为 1 的通道上两个发送方和两个接收方之间基于消息传递的通信.指针操作符&和*的定义在文献[22]中.该实例程序的 MSVL 代码及其执行结果如图 3 所示.MSV 平台执行实例程序并生成 10 个状态.这里省略了繁琐的归约过程,结果如表 2 所示.表中列出了每个状态上相关变量的值.首先,进程 p_1 和 p_2 都请求发送锁,但是因为 p_1 的进程 ID 比 p_2 小,所以 p_1 获得发送锁并将 x 发送到通道 c.同时,q_1 和 q_2 都请求接收锁,q_1 获得锁,但是 q_1 必须等待直到通道为非空.然后,q_1 接收 p_1 的消息后在状态 s_3 将 y 设置为 2.在 p_1 释放发送锁后,p_2 获得发送锁,并且在 q_1 接收 p_1 的消息并清空通道前等待发送消息.最后,q_2 获得接收锁并接收 p_2 发送的消息.

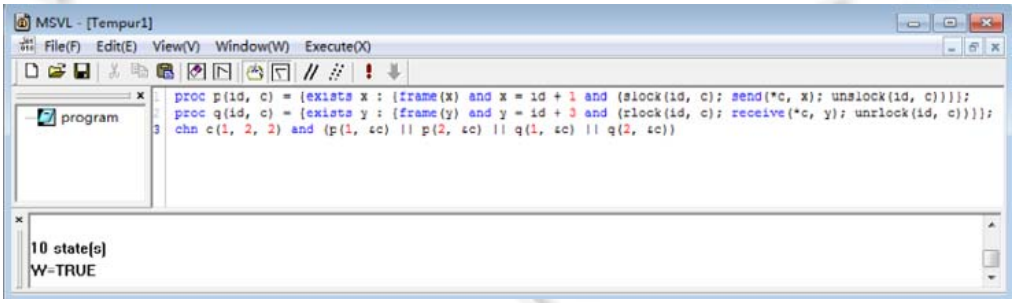


Fig.3 MSVL code of example program and its execution results

图 3 实例程序的 MSVL 代码和运行结果

Table 2 States of the program

表 2 程序的状态

状态	p_1	p_2	q_1	q_2	c
s_0	x=2	x=3	y=4	y=5	⟨⟩
s_1	x=2	x=3	y=4	y=5	⟨⟩
s_2	x=2	x=3	y=4	y=5	⟨2⟩
s_3	x=2	x=3	y=2	y=5	⟨⟩
s_4	x=2	x=3	y=2	y=5	⟨⟩
s_5		x=3	y=2	y=5	⟨3⟩
s_6		x=3		y=5	⟨3⟩
s_7		x=3		y=3	⟨⟩
s_8				y=3	⟨⟩
s_9				y=3	⟨⟩

5 多方电子合同签名协议的建模与验证

多方电子合同签名协议的目标是允许 n 个参与方通过网络来签署数字合同.该协议应该满足一些重要的性质,比如公平性和乐观性[14].公平性确保协议中只有两种情况发生:要么所有参与方都收到签名的合同,要么没有任何一方能收到签名的合同.实现公平性的一个简单方法是,让可信的第三方(trusted third party,简称 TTP)参与该协议.TTP 简单地收集各方的签名,然后将它们分发给各方.由于 TTP 将参与所有各方的通信,那么 TTP 很容易成为系统的瓶颈.这个缺点需要引入乐观性来解决:仅当存在参与方试图欺骗或网络发生不可恢复的异常的情况下才需要 TTP 参与仲裁.乐观的情况是:在没有 TTP 的介入下,各方和通信网络的行为表现都是正确的.多方电子合同签名协议由两部分组成:主协议和恢复协议.由参与方执行的主协议在所有参与方之间有两轮通信.经证明:如果欺骗方少于所有参与方的一半,则该协议至多需要两轮[14].如果没有欺骗方或网络异常,则不涉及 TTP 所执行的恢复协议.协议细节描述如下.

1) 主协议

- 第 1 轮:
  - P_i 发送 $m_{[1,i]}=sign_i(1,c)$给其他参与方;
  - P_i 尝试从所有 $m_{[1,j]}$类型的消息生成向量 $M_1=(m_{[1,1]},\ldots,m_{[1,n]})$.如果成功且每个 $m_{[1,j]}$都是有效的签名,那么 P_i 进入第 2 轮;否则,P_i 等待 TTP 发来消息;
- 第 2 轮
  - P_i 发送 $m_{[2,i]}=sign_i(2,c)$给其他参与方;
  - P_i 尝试从所有 $m_{[2,j]}$类型的消息生成向量 $M_2=(m_{[2,1]},\ldots,m_{[2,n]})$.如果成功且每个 $m_{[2,j]}$都是有效的签名,那么 P_i 决定签署合同和并终止协议;否则,P_i 发送 $m_{[3,i]}=sign_i(3,M_1)$给 TTP 并等待回复.

2) 恢复协议

- TTP:如果 TTP 接收到至少一条消息 $m_{[3,i]}$,其中包含一个完整且连续的 M_1,那么 TTP 会发送消息 $M_{ttp}=sign_{ttp}(M_1)$给所有参与方,且每个收到该消息的 P_i 都会签署合同;否则,TTP 不发送任何消息.如果等待 TTP 应答消息的 P_i 没有接受到消息,合同签署失败,或者在接收 M_{ttp} 后签署合同,协议终止.

如图 4 所示,我们讨论了同步网络中的合同签名协议.为了提高效率,任何两个不同的参与方通过两个单向通道进行通信.当没有异常发生时,TTP 不参与协议.不失一般性,假定各方之间的所有消息都是简单的字符串.

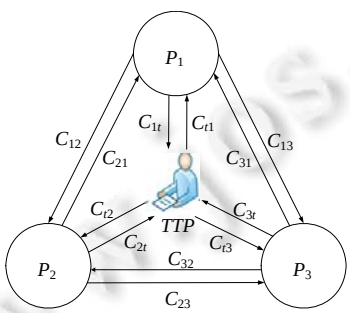


Fig.4 Protocol structure for three parties

图 4 3 个参与方的协议结构

在第 2 轮中,参与方可能不发送消息,因为它不可信或网络不可靠.按第 1 轮中不发送消息的参与方数量,协议中的参与方总共可分为 4 种情形.

- 情形 1:第 1 轮中,所有三方都向其他各方发送信息.进入第 2 轮后,三方都可以向其他方各方发送或不发送消息,总共有 $2^3=8$ 个实例.但是,所有三方最终都可以得到一份已签名的合同;
- 情形 2:第 1 轮中,所有三方中只有两个能发送信息.因此,只有第 3 个参与方能获得所有消息并进入第 2

轮.根据恢复协议,第 3 个参与方发送一个恢复请求给 TTP,然后,TTP 广播已签名的合同,所以所有三方都可以获得签名的合同.在发送恢复请求之前,第 3 方可以选择发送消息或不发送消息,因此有 $C_3^2 \times 2 = 6$ 个实例;

- 情形 3:第 1 轮中,只有一个参与方能发送消息,因此有 $C_3^1 = 3$ 种情况.所有三方都不能计算 M_1,所以不能进入第 2 轮;那么所有的三方都等待 TTP 发来消息.由于没有一个参与方能进入第 2 轮并向 TTP 发出恢复信息,因此 TTP 不会发送应答信息.三方均不能在限定时间内收到已签名的合同;
- 情形 4:第 1 轮中,所有三方都不能发送信息,因此只有 1 个实例.像前一种情形一样,没有参与方进入第 2 轮,所有三方都没有收到已签名的合同.

经上述分析可知:该协议的算法在只有 3 个参与方和 TTP 的情况下,一共有 $2^3 + C_3^2 \times 2 + C_3^1 + 1 = 18$ 种可能的执行路径,即模型 NFG 路径数.

我们使用一个基于 NFG 的 PPTL 统一模型检测方法[8].统一模型检测方法首先用 MSVL 程序 P 对协议进行建模;然后,用 PPTL 公式 Φ 来描述每条关注的性质.在 PTL 逻辑框架下的 MSV 平台执行 P 并检查协议是否满足性质 Φ.建模模式下,扩展的 MSV 平台执行建模程序,并生成如图 5 所示的 18 条执行路径.仿真模式下,输出 18 条路径其中一条的 NFG.如图 5 所示,18 条终止于圆节点的路径就是协议的模型.

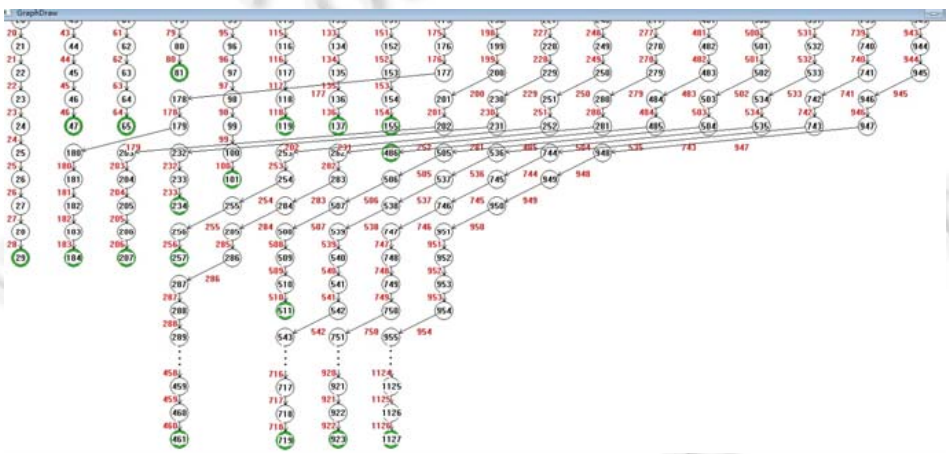


Fig.5 Modeling result of the protocol

图 5 协议的建模结果

在验证模式下,我们需要用 PPTL 公式来描述公平性和乐观性.将程序以及这些性质输入到 MSV 平台,然后输出验证结果.

- 公平性
  - *define a*: $sign_1$="*nil*";
  - *define b*: $sign_2$="*nil*";
  - *define c*: $sign_3$="*nil*";
  - *define d*: $sign_1$="*signed*";
  - *define e*: $sign_2$="*signed*";
  - *define f*: $sign_3$="*signed*";
  - *fin*(*a* and *b* and *c*) or (*d* and *e* and *f*).

命题 $sign_i$="*nil*"表示第 *i* 个参与方没有获得签名的合同,而 $sign_i$="*signed*"表示第 *i* 个参与方已经获得了签名的合同.公式 *fin*(*a* and *b* and *c*) or (*d* and *e* and *f*)代表公平性,即:所有三方都没有得到签名的合同,或都获得签名的合同.图 6 中,有 18 个以圆节点结束的路径,因此协议满足公平性.

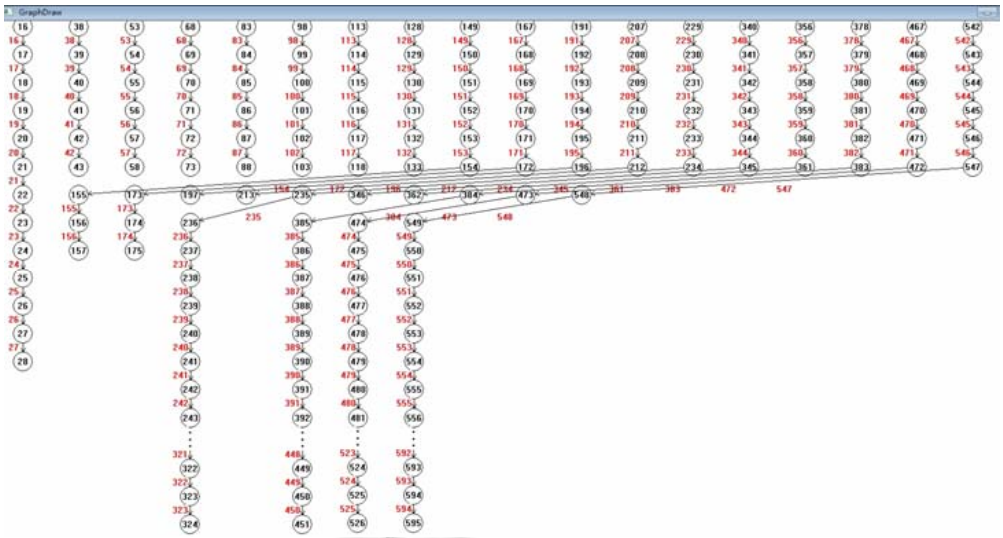


Fig.6 Verification result of the fairness property

图 6 公平性的验证结果

- 乐观性
  - *define a*: $comp_1$=1;
  - *define b*: $comp_2$=1;
  - *define c*: $comp_3$=1;
  - *define d*: *nottp*=1;
  - *fin*((*a* and *b* and *c*)→*d*).

命题 $comp_i$=1 表示第 *i* 个参与方已经进入了第 2 轮且获取了其他参与方的消息,并成功计算出了向量 M_2. 命题 *nottp*=1 表示 TTP 没有参与协议.公式 *fin*((*a* and *b* and *c*)→*d*)表示乐观性,它意味着如果所有三方都能够计算向量 M_2,那么 TTP 将不参与最终的状态.图 7 中,有 18 个以圆节点结束的路径,因此协议满足乐观性.

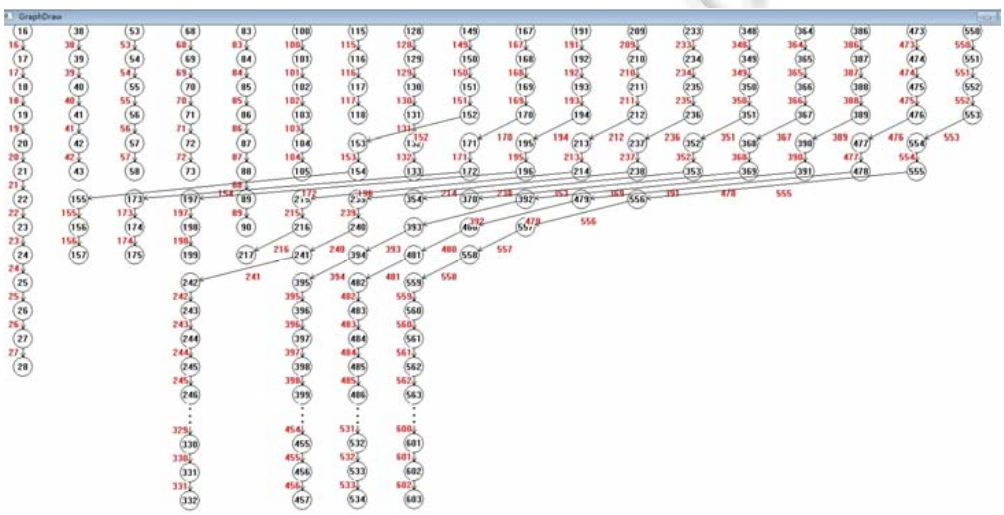


Fig.7 Verification result of the optimism property

图 7 乐观性的验证结果

- 一个测试的性质
  - *define a*: $sign_1$=“*signed*”;
  - *define b*: $sign_2$=“*signed*”;
  - *define c*: $sign_3$=“*signed*”;
  - *fin*(*a* and *b* and *c*).

命题 $sign_i$=“*signed*”表示第 *i* 个参与方已经获得了签名的合同.而公式 *fin*(*a* and *b* and *c*)代表的性质表示所有三方都能在最后一个状态获得签名的合同.根据前面情形分析,在情形 3 和情形 4 下,性质明显不满足.图 8 中,有 4 个反例,4 条路径的结束于终止节点,所以该协议不满足此性质.

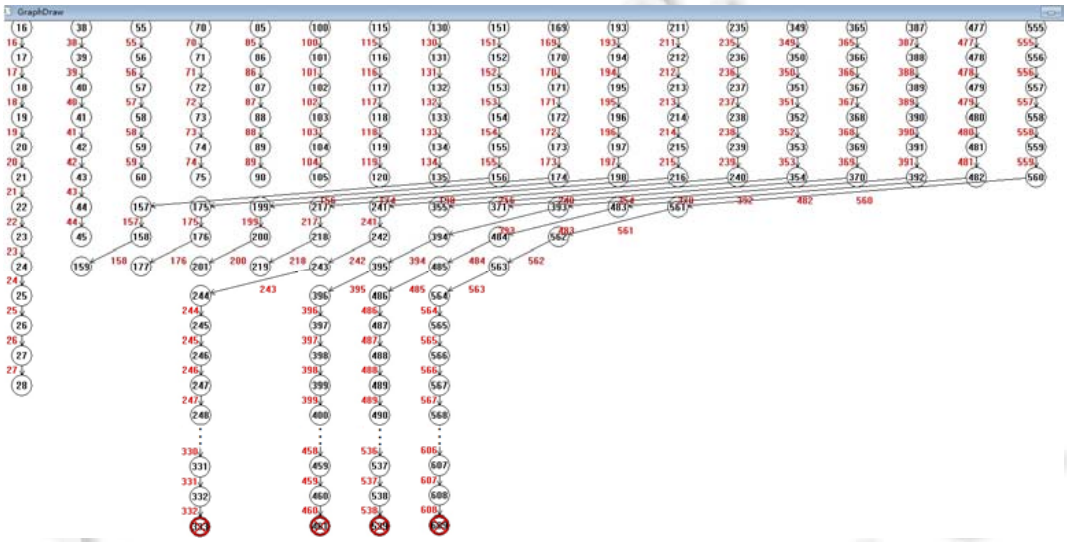


Fig.8 Verification result of an unsatisfiable property

图 8 不满足性质的验证结果

以上分析都是基于只有 3 个参与方和 TTP 参与的情况,在分析了 4~6 个参与方(存在 TTP)之后,我们可以得到如下的表 3 和图 9.

根据图 9(a)可知:在有 TTP 参与的情况下,模型的状态数与参与方数量 *n* 大致成指数关系.而在图 9(b)中,根据 3~6 个参与方的可能执行路径数类推可得,*n* 个参与方的模型 NFG 路径数 $Path_n$ 为

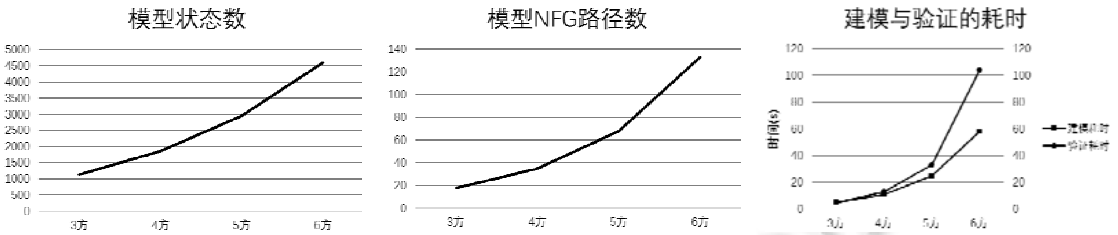
$$Path_n = 2^n + C_n^{n-1} \times 2 + C_n^{n-2} + ... + C_n^2 + C_n^1 + 1.$$

上式经化简计算可得:$Path_n=2^{n+1}+n-1$.所以,模型 NFG 路径数 $Path_n$ 和参与方数量 *n* 呈指数关系.对于建模耗时,由于建模与验证使用的是基于 PPTL 统一模型检测方法,该方法基于 PPTL 的可满足性,经粗略计算,其时间复杂度为非基本时间.但如果模型中没有嵌套的¬(·;·)结构,那么算法的时间复杂度是指数级的.从图 9(c)中可以看出,建模耗时和验证耗时大致符合这一规律.而验证相对于建模,状态越多就要花费更多额外的时间,显然也符合常理.

Table 3 Four different modeling situations

表 3 4 种不同的建模情况

	3 方	4 方	5 方	6 方
模型状态数	1 127	1 863	2 947	4 587
模型 NFG 路径数	18	35	68	133
建模耗时	5s	9s	21s	58s
验证耗时	5s	11s	26s	104s



(a) (b) (c)

Fig.9 Four different modeling and verification situations

图 9 4 种不同建模和验证情况

6 相关工作

目前,大多数时序逻辑编程语言,如前文提到过的 METATEM,XYZ/E 和 Tempura,也支持基于消息传递的通信机制,用于描述并发和分布式系统中的通信,但都存在一定的不足.METATEM 代理之间通过广播或多播消息传递来相互通信,且由于 METATEM 代理的组件互相独立,所以对于同步和异步系统都适用[23].虽然从逻辑的角度来看这是个合理模型,但这种方法代价大并且不适合表示非扁平化和结构化系统.XYZ/E 中用一个变量表示通道,这个通道被视为一次存储一条信息的单缓冲区,基于消息传递的通信机制的实现依赖于两个通道:一个用于进程输出数据的写通道,一个用于进程读入数据的读通道[10].这种方法虽然灵活,但当一个通道同时被多个进程访问时,就会产生冲突[15].而 Tempura 中通过引入一个带端口的流包来对并行进程之间的同步通信进行建模,相关文献[11]中也给出了相应的通信语句和简单示例,但具体的实现细节在文献中并没有提及.在经典进程代数中,如 CCS[24]、π-演算[25]和 CSP[19],只支持同步通信,没有消息缓冲区或通道来连接通信代理,但是异步通信可以通过在两个通信实体之间引入缓冲代理进行建模.本文实现的 MSVL 通信机制同样基于消息传递,但是与上述方法不同,采用先进先出队列表示通道,引入面包店算法解决互斥问题,并给出了通道结构、通信语句和进程结构的形式化定义,适用于同步系统和异步系统的建模与验证.

7 结 论

为了更好地对分布式并发系统进行建模与验证,给 MSVL 增加了消息传递的通信机制,给出了通信的形式化定义,包括通道结构、面包店算法、通信语句和进程结构.通道被定义为一个存储消息的有界 FIFO 队列.面包店算法及通信语句用于对通道进行加锁及访问,解决了互斥问题.进程定义为 MSVL 的组合语句,用来描述系统的行为模式.研究了消息传递的通信机制在 MSV 平台中的实现机制.在 MSV 平台中,对多方电子合同签名协议进行建模与验证,表明该通信机制是有效的.验证了该协议的公平性与乐观性,验证了一个不可满足的测试性质并给出了反例.将来的研究工作包括将具有通信机制的 MSVL 应用到更多实际的通信系统中,如社交网络.

References:

[1] Leeuwen JV. Handbook of Theoretical, Computer Science, Vol.B: Formal Models and Semantics. London: The MIT Press, 1994. 298−306. [doi: 10.1016/0167-6423(95)90009-8]

[2] Baier C, Katoen J. Principles of Model Checking. London: The MIT Press, 2008.

[3] Tian C, Duan ZH. Expressiveness of propositional projection temporal logic with star. Theoretical Computer Science, 2011,412: 1729−1744. [doi: 10.1016/j.tcs.2010.12.047]

[4] Fisher M. An Introdution to Practical Formal Methods Using Temporal Logic. Wiley Publishing, 2011.

[5] Wang M, Duan ZH, Tian C. Simulation and verification of the virtual memory management system with MSVL. In: Hou JL, Trappey AJC, eds. Proc. of the CSCWD 2014. Danvers: IEEE, 2014. 360−365. [doi: 10.1109/cscwd.2014.6846870]

[6] Zhang P, Duan ZH, Tian C. Simulation of CTCS-3 protocol with temporal logic programming. In: Shen WM, Li WD, eds. Proc. of the CSCWD 2013. Piscataway: IEEE, 2013. 72−77. [doi: 10.1109/cscwd.2013.6580942]

[7] Yu Y, Duan ZH, Tian C, Yang MF. Model checking C programs with MSVL. In: Liu SY, ed. Proc. of the SOFL 2012. LNCS 7787. London: Springer-Verlag, 2013. 87−103. [doi: 10.1007/978-3-642-39277-1_7]

[8] Duan ZH, Tian C. A unified model checking approach with projection temporal logic. In: Liu SY, Maibaum T, Araki K, eds. Proc. of the ICFEM 2008. LNCS 5256. London: Springer-Verlag, 2008. 167−186. [doi: 10.1007/978-3-540-88194-0_12]

[9] Scott ML. Programming Language Pragmatics. 3rd ed. San Francisco: Morgan Kaufmann Publishers, 2009. [doi: 10.1016/b978-0-12-374514-9.x0001-8]

[10] Ma HD, Liu SQ. Multimedia data modeling based on temporal logic and XYZ system. Journal of Computer Science and Technology, 1999,14(2):188−193. [doi: 10.1007/bf02946527]

[11] Moszkowski B. Executing Temporal Logic Programs. New York: Cambridge University Press, 1986. [doi: 10.1017/s0022481200029169]

[12] Lee EA. The problem with threads. IEEE Computer, 2006,39(5):33-42. [doi: 10.1109/mc.2006.180]

[13] Herlihy M, Shavit N. The Art of Multiprocessor Programming. San Francisco: Morgan Kaufmann Publishers, 2012. [doi: 10.1145/1146381.1146382]

[14] Baum-Waidner B. Optimistic asynchronous multi-party contract signing with reduced number of rounds. In: Orejas F, Spirakis PG, Leeuwen JV, eds. Proc. of the ICALP 2001. LNCS 2076. London: Springer-Verlag, 2001. 898−911. [doi: 10.1007/3-540-48224-5_73]

[15] Mo DP, Wang XB, Duan ZH. Asynchronous communication in MSVL. In: Qin S, Qiu Z, eds. Proc. of the ICFEM 2011. LNCS 6991. London: Springer-Verlag, 2011. 82−97. [doi: 10.1007/978-3-642-24559-6_8]

[16] Duan ZH, Yang XX, Koutny M. Framed temporal logic programming. Science of Computer Programming, 2008,70(1):31−61. [doi: 10.1016/j.scico.2007.09.001]

[17] Charronbost B, Mattern F, Tel G. Synchronous, asynchronous, and causally ordered communication. Distributed Computing, 1996, 9(4):173−191. [doi: 10.1007/s004460050018]

[18] Cormen T, Leiserson C, Rivest R. Introduction to Algorithms. 3rd ed., London: The MIT Press, 2009.

[19] Hoare CAR. Communicating Sequential Processes. Upper Saddle River: Prentice-Hall, 1985. [doi: 10.1007/978-3-662-09507-2_19]

[20] Yang XX, Duan ZH. Operational semantics of framed tempura. The Journal of Logic and Algebraic Programming, 2008,78(1): 22−51. [doi: 10.1016/j.jlap.2008.08.001]

[21] Duan ZH. Temporal Logic and Temporal Logic Programming. Beijing: Science Press, 2006.

[22] Luo L, Duan ZH, Tian C, Wang XB. A structural transformation from $p-\pi$ to MSVL. Journal of Combinatorial Optimization, 2015, 29(1):308−329. [doi: 10.1007/s10878-014-9779-0]

[23] Fisher M. MetateM: The story so far. In: Bordinni RH, ed. Proc. of the ProMAS 2005. LNAI 3862. London: Springer-Verlag, 2005. 3−22. [doi: 10.1007/11678823_1]

[24] Milner R. A Calculus of Communicating Systems. New York: Springer-Verlag, 1980. [doi: 10.1007/3-540-10235-3]

[25] Milner R. Communicating and Mobile Systems: The π-calculus. New York: Cambridge University Press, 1999. [doi: 10.1016/s0167-6423(00)00008-3]

王小兵(1979－),男,湖北武汉人,博士,副教授,CCF 高级会员,主要研究领域为形式化方法,时序逻辑程序设计.

郭文轩(1994－),男,硕士,主要研究领域为形式化方法,时序逻辑程序设计.

段振华(1948－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为网络计算,高可信软件理论和技术.