文章编号:1000-5862(2020)03-0301-06

一种基于 Concurrent Apla 语言的共享内存并发分布式算法 2 层验证方法

王昌晶[1],余小军[1],沈德明[2],罗海梅[3,4],左正康[1*]

(1. 江西师范大学计算机信息工程学院,江西 南昌 330022;2. 江西科技师范大学通信与电子学院,江西 南昌 330013;
3. 江西师范大学物理与通信电子学院,江西 南昌 330022;4. 江西师范大学江西省光电子与通信重点实验室,江西 南昌 330022)

摘要:形式化验证共享内存并发分布式算法已成为当前极具挑战性的问题之一,尤其是在云计算、多核、无线传感器网络、分布式数据库、区块链环境下. 该文基于研究团队在形式化规约语言和方法、算法形式推导和验证方面的已有工作,以自定义泛型抽象顺序设计语言 Apla 为基础,进一步研究并提出简明、高抽象用于并发分布式计算的 Concurrent Apla 语言,使其既支持顺序算法的验证又能有效地验证并发分布式算法. 在依赖-卫式推理的基础上,提出一种新颖的 2 层并发分布式算法形式化验证方法,其中系统层用于处理并发级验证,而组件层用于处理顺序级验证. 最后,通过 2 个实例验证了该方法的有效性和可行性.

关键词:并发分布式计算;依赖-卫式推理;Concurrent Apla;形式化验证

中图分类号:TP 311 **文献标志码**:A **DOI**:10.16357/j.cnki.issn1000-5862.2020.03.14

0 引言

形式化方法在软件验证的应用从串行程序验证开始[1-2],随后运用到并发分布式系统[3]、云计算系统[4-5]、反应式系统[6]等. 形式化方法采用数学和逻辑的方法描述和验证软件. 从描述上来讲,一方面是关于系统或程序的描述,另一方面是关于性质的描述. 可以用 1 种或多种语言来进行描述,包括命题逻辑、1 阶谓词逻辑、高阶谓词逻辑、代数、状态机、自动机、线性时序逻辑、计算树逻辑、进程代数、π 演算、u 演算、特殊的程序语言及其子集等. 从验证方法来讲,主要有 2 类方法:一类是以逻辑推理为基础的演绎验证(Deductive Verification)[7],如定理证明(Theorem Proving)、自然演绎(Natural Deduction)、相继式演算(Sequent Calculus)以及 Hoare 逻辑等方法;另一类是以穷尽搜索算法为基础的模型检测(Model Checking)[8]方法.

演绎验证不同于模型检测,它可以验证无限状态系统,能够处理不同域(如整数、实数)上的程序,以及线性、非线性数据结构(如栈、队列、树、图等),它甚至允许参数化程序的验证,如有任意数目相同进程的程序;而模型检测自动化技术通常只能检验有限状态系统,且只能检验这类程序的特定实例,如检验全部含有 1 ~ 7 个进程的实例,不可判定的结果表明通常不能自动化参数化程序的验证. 除此以外,模型检测还容易发生状态空间爆炸,这在较大程度上限制了能被检验的并发程序的数目.

本文在抽象顺序设计语言 Apla 中加入并发、通讯和同步的语言成分,提出简明、高抽象度并发分布式计算的 Concurrent Apla 语言,包括语法及结构化操作语义;然后在依赖-卫式推理的基础上,提出一种新颖的 2 层并发算法形式化验证方法,其中高层系统层用于处理并行并发级验证,而低层组件层用于处理顺序级验证;最后通过 2 个案例验证了方法的有效性和可行性.

收稿日期:2020-03-19

基金项目:国家自然科学基金(61762049,11804133,61862033,61662035,61902162),国家留学基金(202008360094),江西省科技厅课题(20181BAB206034)和江西省研究生创新基金(YC2019-S161)资助项目.

作者简介:王昌晶(1977-),男,江西南昌人,教授,博士,博士生导师,主要从事可信软件、智能化软件与智能化教育的研究. E-mail:wcj771006@163.com

通信作者:左正康(1980-),男,江西抚州人,教授,博士,主要从事泛型程序设计和可信软件的研究. E-mail:kerrykaren@126.com

1 基于共享内存的并发分布式计算的 Concurrent Apla 语言

1.1 Concurrency Apla 语法

在抽象顺序设计语言 Apla[9] 中加入并发、通讯和同步的语言成分,构造并发分布式计算语言 Concurrency Apla,其语法如下所示:

$P::= \bar{x}=\bar{e} \mid P_1;P_2 \mid$ if $b_1 \to P_1 \cdots b_n \to P_n$ fi $\mid$ while b do P od $\mid$ await b then P end $\mid P_1 \parallel P_2$,

其中 $\bar{x}=\bar{e}$(赋值语句)、$P_1;P_2$(顺序语句)、if $b_1 \to P_1 \cdots b_n \to P_n$ fi(不确定选择语句)、while b do P od(循环语句)的语法来自 Apla 语言,与 Dijkstra 卫式命令语言类似. Concurrency Apla 在 Apla 语言基础上扩展了 await b then P end、$P_1 \parallel P_2$ 这 2 个算子,使其既支持顺序算法的验证又能有效地验证并发分布式算法. 同步使用 await b then P end 来描述. 同步要求进程之间互相协调,实现统一性和一致性,当 2 个进程不同步时,可能导致其内存共享变量的读写错误,这时让不同步进程进行等待,使其满足条件 b(即满足同步要求)再进行执行. 并发使用 $P_1 \parallel P_2$ 来描述,P_1、P_2 进程的原子行动交替执行从而实现其并发. 通讯使用共享变量来描述,用共享变量来约定其读写步骤的进行.

1.2 Concurrency Apla 结构化操作语义

下面给出了 Concurrency Apla 语言的结构化操作语义,包括赋值语句(Assignment)、顺序语句(Sequence)、不确定选择语句(Conditional)、循环语句(Loop)、等待语句(Await)和并发语句(Parallel)的操作语义. 其中顺序语句操作语义包含 2 种情况:第 1 种情况是在第 1 个程序执行过程中,程序顺利执行完成,此时转到第 2 个程序进行执行;第 2 种情况是第 1 个程序还没有执行完成,接下来继续执行. 不确定选择语句和循环语句操作语义分别在选择条件和循环条件为真和假的 2 种情况下的语义. 等待语句操作语义是程序进入了等待的状态,直到满足获得相关的资源条件等待终止. 并发语句操作语义在满足并发条件的情况下触发其中一个程序,此程序满足执行条件,从而使其程序可以正确地执行,当执行完后其程序的状态发生了改变,执行条件的状态也相应地进行了改变.

(i) 赋值语句为

$$\frac{}{(\bar{x}=\bar{e},s) \to (\perp, f\,s)};$$

(ii) 顺序语句为

$$[\text{SEQ1}]:\frac{(p_1,s) \to (\perp,s')}{(p_1;p_2,s) \to (p_2,s')},$$

$$[\text{SEQ2}]:\frac{(p_1,s) \to (p'_1,s')}{(p_1;p_2,s) \to (p'_1;p_2,s')};$$

(iii) 条件语句为

$$[\text{CONDT}]:\frac{s \in b}{(Cond\ b\ p_1\ p_2,s) \to (p_1,s)},$$

$$[\text{CONDF}]:\frac{s \notin b}{(Cond\ b\ p_1\ p_2,s) \to (p_2,s)};$$

(iv) 循环语句为

$$[\text{while } T]:\frac{s \in b}{(\text{while } b\ p,s) \to (p;(\text{while } b\ p),s)},$$

$$[\text{while } F]:\frac{s \notin b}{(\text{while } b\ p,s) \to (\perp,s)};$$

(v) 等待语句为

$$\frac{s \in b(p,s) \to (\perp,s')}{(\text{await } b\ p,s) \to (\perp\ s')};$$

(vi) 并发语句为

$$[\text{parallel1}]:\frac{(P,s) \to (P',s')}{(P \parallel Q,s) \to (P' \parallel Q,s')},$$

$$[\text{parallel2}]:\frac{(Q,s) \to (Q',s')}{(P \parallel Q,s) \to (P \parallel Q',s')}.$$

2 基于 Concurrent Apla 语言的共享内存并发分布式算法 2 层验证方法

2.1 依赖-卫式规约

并行并发算法的验证难点在于其线程的交错和变量的状态共享. C. B. Jones[10] 提出了依赖-卫式(Rely-Guarantee)推理方法,解决了可组合性问题. 该方法在众多验证中[11-13] 使用,应用范围较为广泛. Rely-Guarantee 方法将并发任务间的接口抽象为 Rely 和 Guarantee 2 种,Guarantee 是对任务自身行为的抽象,用于描述自身行为的变化;而 Rely 则是对任务所能接受的环境行为的抽象,用于描述任务所能接受的环境行为所带来的干扰. 在检查任务之间的无干扰性时,不需要逐行检查任务的代码,只要检查不同任务之间的 Rely 和 Guarantee 接口的匹配即可,要求每个任务的 Rely 被其他每一个任务的 Guarantee 蕴含(即 Rely 得到满足). 由于不再需要逐行检查所有任务的代码,Rely-Guarantee 方法允许对单个任务进行独立验证,因此是一种具备可组合性的方法. 这可以看作是一个组合的 Owicki-Gries Method[14-15].

Rely-Guarantee 方法是目前国际主流的并行分

布式算法验证方法. Rely-Guarantee 规约是在 Hoare 逻辑的基础上将断言的 2 元组(*pre*,*post*) 的形式进行扩展,在中间过程引入 *R*、*G*,使其规约扩展为 4 元组(*pre*,*rely*,*guar*,*post*) ,因此 Rely-Guarantee 方法较好地兼容了 Hoare 逻辑方法[16].

2.2　2 层共享内存并发算法验证方法

2 层共享内存并发分布式算法验证方法思想来源于 Rely-Guarantee 方法,是一种可组合的推理方法,在推理时不需要检查任务之间的无干扰性,只需检查不同任务之间的 Rely 和 Guarantee 接口的匹配,它允许对单个任务进行独立验证. 因此,可以将基于共享内存的并发分布式算法的验证划分为系统层和组件层,其中系统层用于处理并行并发级,而组件层用于处理顺序级. 在系统层中主要的难点是并发进程之间的共享变量的干扰性,使用 Rely-Guarantee 推理来形式化验证; 在组件层中重点是带 while 循环的顺序程序,而难点是循环不变式的生成技术,使用 Hoare 逻辑来解决. 因此,将共享内存并发分布式算法验证涉及的推理规则相应的设计为系统层推理规则和组件层推理规则 2 大类.

其中组件层推理规则包括 Hoare 逻辑的基本规则: 赋值规则(Assignment rule) 、跳跃规则(Skip rule) 、顺序规则(Sequence rule) 、不确定选择规则(Conditional rule) 、循环规则(Loop rule) 等. 具体说明如下.

(i) 赋值规则为

$$pre\ [e/x]\{x:=e\}\ pre;$$

(ii) 跳跃规则为

$$pre\{skip\}\ pre;$$

(iii) 顺序规则为

$$\frac{pre\{S_1\}\ R_1, R_1 \rightarrow R_2, R_2\{S_2\}\ post}{pre\{S_1; S_2\}\ post};$$

(iv) 不确定选择规则为

$$\frac{B \wedge pre\{S_1\}\ post, \neg B \wedge pre\{S_2\}\ post}{pre\{\text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi}\}\ post};$$

(v) 循环规则为

$$\frac{P(x) \wedge B\{S\}\ P(x)}{P(x)\{\text{while } B \text{ do } S \text{ od}\} \neg B \wedge P(x)}.$$

而系统层推理规则主要包括基本规则(Basic rule) 、并发规则(Parallel rule) 、等待规则(Await rule) 以及辅助变量规则(Auxiliary variable rule) . 其中基本规则可以表示组件层的任一推理规则,它是系统层和组件层的接口. 具体说明如下:

(i) 基本规则为

$$\frac{\begin{array}{l} pre\ \underline{stable}\ \text{when}\ rely \\ post\ \underline{stable}\ \text{when}\ guar \\ \vdash\{pre\}\ C\{post\} \\ \text{the effect of } C \text{ is contained in } guar \end{array}}{rely, guar \vdash\{pre\}\ C\{post\}};$$

(ii) 并发规则为

$$\frac{\begin{array}{l} R_1, G_1 \vdash P_1\{C_1\}\ Q_1; R_2, G_2 \vdash P_2\{C_2\}\ Q_2 \\ R = R_1 \wedge R_2, G_1 \vee G_2 \Rightarrow G, G_1 \vdash R_2, G_2 \vdash R_1 \end{array}}{R, G \vdash\{P_1 \wedge P_2\}\ C_1 \parallel C_2\{Q_1 \wedge Q_2\}};$$

(iii) 等待规则为

$$\frac{\begin{array}{l} pre\ \underline{stable}\ \text{when}\ rely \\ post\ \underline{stable}\ \text{when}\ guar \\ P\ \underline{sat}(pre \wedge b \wedge y = v_0, y' = y, \text{true}, guar\ [v_0/y, y/y'] \wedge post) \end{array}}{\text{await } b \text{ then } P \text{ end}\ \underline{sat}(pre, rely, guar, post)};$$

(iv) 辅助变量规则为

$$\frac{\begin{array}{l} \exists z.\ pre_1(y, z, y_0) \\ \exists z'.\ rely_1((y, z).(y', z'), y_0) \\ P\ \underline{sat}(pre \wedge pre_1, rely \wedge rely_1, guar, post) \end{array}}{Q\ \underline{sat}(pre, rely, guar, post)}.$$

这里简要介绍组件层推理规则 Hoare 逻辑的基本规则.

系统层 Basic rule 扩展了 Hoare 逻辑,只需要证明前置条件 *pre* 和后置条件 *post* 在 *rely* 的保证下能够平稳,以及自身的改变都写入到 *guar* 中,最后加上 Hoare 逻辑的证明规则,即可得出要证明的规则为 $rely, guar \vdash\{pre\}\ C\{post\}$,也可写成 4 元组的形式 $C\ \underline{sat}(pre, rely, guar, post)$. 其中 the effect of *C* is contained in *guar* 定义如下: $\forall \alpha, \alpha' P(\alpha) \wedge (\alpha, \alpha') \in [\![C]\!] \Rightarrow G(\alpha, \alpha')$ 表示程序执行时的每一句的前状态和其程序执行后得到的后状态都被写进 *guar* 中,用以保证程序的正确执行. 这里的难点在于需要仔细地分析程序以确保自身的改变都写入了 *guar* 中. 程序自身的改变在程序运行时是复杂的,特别是并发算法程序,这就要求对要验证的算法步骤以及程序状态的自身改变有一个很好的认识和理解.

在 Parallel rule 中,定义了如何使程序 *C* 并行的一些规则,它是对单个 Basic rule 的组合,使得程序满足 Basic rule 的同时能够通过自身的 *rely*、*guar* 以及并行全局的 *Rely*、*Guar* 来保障并行并发的正确执行.

在 Await rule 中,缓和了并行并发程序竞争共享资源的干扰,使其在通过 Await rule 时能够使并

行并发程序不出错的同时,不会因为共享资源的使用不当导致死锁或者饥饿的发生.

Auxiliary variable rule 是在证明并发算法正确性的一个辅助规则,当在证明程序中引入了辅助变量时,使用此规则对此辅助变量进行消解,从而擦除辅助变量对关于程序的前后置条件的影响.

3 案例研究

3.1 并发执行 $x:=x+1$

证明如下程序的正确性:

$pre:\{x:=0\}\langle x:=x+1\rangle \parallel \langle x:=x+1\rangle post:\{x:=2\}$.

(i) 符号化表示为 $pre\{C_1 \parallel C_2\}post$.

(ii) 引入辅助变量 $y:=0;z:=0$.

(iii) 根据赋值规则构造变换前置条件

$\{x:=0\}y:=0;z=0\{x=y+z\wedge y=0\wedge z=0\}$.

(iv) 令 $P=\{x=y+z\wedge y=0\wedge z=0\}$ 将 P 分解有

$P_1:\{x=y+z\wedge y=0\}P_2:\{x=y+z\wedge z=0\}$,则问题分解为

$P_1\{C_1\}Q_1 \parallel P_2\{C_2\}Q_2;Q_1\wedge Q_2\Rightarrow post$.

(v) 在第 1 个程序中根据赋值规则有

$P_1:\{x=y+z\wedge y=0\}\langle x:=x+1;y:=1\rangle\{x=y+z\wedge y=1\}:Q_1$,

可得 $P_1\{C_1\}Q_1$ 显然成立. $P_2\{C_2\}Q_2$ 也是同样的过程,同理可证.

运用基本规则通过分析 $stable(P_1,R_1)$; $stable(Q_1,R_1)$ 以及程序可得

$G_1\equiv y=0\wedge y'=1\wedge x'=x+1\wedge z'=z$,

$G_2\equiv z=0\wedge z'=1\wedge x'=x+1\wedge y'=y$,

$R_1\equiv G_2;R_2\equiv G_1$.

对于 $R_1\{C_1\}Q_1$ 有

$\vdash P_1\{C_1\}Q_1;stable(P_1,R_1);stable(Q_1,R_1)$.

(vi) 通过深入分析理解程序以及定义: $\forall\alpha,\alpha',P(\alpha)\wedge(\alpha,\alpha')\in[\![C]\!]$,可得

$x=y+z\wedge y=0\wedge z=0\wedge x'=y'+z'\wedge y'=1\wedge z'=z\Rightarrow y=0\wedge y'=1\wedge x'=x+1\wedge z'=z$,

则有 $\forall\alpha,\alpha' P(\alpha)\wedge(\alpha,\alpha')\in[\![C]\!]\Rightarrow G(\alpha,\alpha')$.

可证 $R_1,G_1\vdash P_1\{C_1\}Q_1$ 成立. 同理可证有 R_2, $G_2\vdash P_2\{C_2\}Q_2$ 成立.

(vii) 由程序性质可知: 共享变量在并行系统上而言,没有外界干扰,所以不需要依赖于外界条件,同理也不需要保证什么,所以可以设置 $R=\text{False}$, $G=\text{True}$,则有下式成立:

$\text{False}=y=0\wedge y'=1\wedge x'=x+1\wedge z'=z\wedge z=0\wedge z'=1\wedge x'=x+1\wedge y'=y$,

$(y=0\wedge y'=1\wedge x'=x+1\wedge z'=z)\vee(z=0\wedge z'=1\wedge x'=x+1\wedge y'=y)\Rightarrow\text{True}$.

可得 $R=R_1\wedge R_2;G_1\vee G_2\Rightarrow G$ 成立.

(viii) 根据并发规则和辅助变量规则以及 $pre\rightarrow P;P=P_1\wedge P_2;Q_1\wedge Q_2\rightarrow post$ 可得

$\{x=y+z\wedge y=1\}\wedge\{x=y+z\wedge z=1\}\rightarrow\{x=:2\}$.

3.2 并发求 M,N 最大公约数

已知 $pre:\{x\rightarrow M,N\}$ 用 2 个程序 $pre\{C_1\parallel C_2\}post$ 求最大公约数并证明其程序的正确性得到的后置条件为

$$\exists Z.\ x\rightarrow Z,Z\wedge Z=GCD(M,N).$$

(i) 引入辅助变量 t_{11},t_{12}:

$$t_{21}=[\![x+1]\!],t_{22}=[\![x]\!].$$

(ii) 变换前置条件构造最大公约数函数

$P=GCD([\![x]\!],[\![x+1]\!])=GCD(M,N)$.

(iii) 分解问题为 2 个子问题:

$P_1\{C_1\}Q_1\parallel P_2\{C_2\}Q_2,Q_1\wedge Q_2\Rightarrow post$,

其中 $P=P_1=P_2$.

(iv) 构造求最大公约数程序 C_1:

$t_{11}=[\![x]\!];t_{12}=[\![x+1]\!];$
while $(t_{11}\neq t_{12})$ do{
if $(t_{11}>t_{12})$ {
$t_{11}=t_{11}-t_{12};$
$[\![x]\!]=t_{11};\}$
fi
$t_{12}=[\![x+1]\!]\}$.
od.

同理构造最大最大公约数程序 C_2:

$t_{21}=[\![x]\!];t_{22}=[\![x+1]\!];$
while $(t_{21}\neq t_{22})$ do{
if $(t_{21}>t_{22})$ {
$t_{21}=t_{21}-t_{22};$
$[\![x+1]\!]=t_{21};\}$
fi
$t_{22}=[\![x+1]\!];\}$.
od.

(v) 证明程序 C_1 的正确性:

$\{GCD([\![x]\!],[\![x+1]\!])=GCD(M,N)\}$,
$t_{11}=[\![x]\!]$.

赋值规则:

$GCD([\![x]\!],[\![x+1]\!])=GCD(M,N)\wedge t_{11}=[\![x]\!]$,

$t_{12} = [x+1]$.

赋值规则和 R,G: 因为可能受程序 C_2 的影响使得变量 $[x+1]$ 减小,故对于变量 t_{12} 有

$t_{11} = [x] \wedge t_{12} \geqslant [x+1] \wedge [x] \geqslant [x+1] \Rightarrow t_{12} = [x+1]$,

则

$GCD(t_{11},t_{12}) = GCD(M,N) \wedge t_{11} = [x] \wedge t_{12} \geqslant [x+1] \wedge [x] \geqslant [x+1] \Rightarrow t_{12} = [x+1]$;

while $(t_{11} \neq t_{12})$ do {

$GCD(t_{11},t_{12}) = GCD(M,N) \wedge t_{11} = [x] \wedge t_{12} \geqslant [x+1] \wedge [x] \geqslant [x+1] \Rightarrow t_{12} = [x+1]$;

if $(t_{11} > t_{12})$ {

// 不确定选择规则

$GCD(t_{11},t_{12}) = GCD(M,N) \wedge t_{11} = [x] \wedge t_{12} = [x+1] \wedge t_{11} > t_{12}$;

$t_{11} = t_{11} - t_{12}$; $[x] = t_{11}$.

// 赋值规则

$GCD(t_{11},t_{12}) = GCD(M,N) \wedge t_{12} = [x+1] \wedge t_{11} > t_{12}$;}.

$t_{12} = [x+1]$.

// 赋值规则

$GCD(t_{11},t_{12}) = GCD(M,N) \wedge t_{11} \leqslant t_{12}$;}.

// 循环规则

$GCD(t_{11},t_{12}) = GCD(M,N) \wedge t_{11} = t_{12}$.

令 $Z = t_{11} = t_{12}$ 可得 Z 即为所求的最大公约数,可写为约束条件

$\exists Z.\, x \rightarrow Z, Z \wedge Z = GCD([x], [x+1])$.

同理可证程序 C_2 也成立.

(vi) 通过分析程序 C_1 和 $stable(P_1,R_1)$, $stable(Q_1,R_1)$ 可得

$G_1 \equiv GCD([x'], [x'+1]) = GCD(M,N) \wedge [x+1] > [x'+1] \wedge ([x] > [x+1] \Rightarrow [x] > [x'] \wedge [x] < [x+1] \Rightarrow [x] = [x'])$.

通过分析程序 C_2 和 C_1, $stable(P_1,R_1)$, $stable(Q_1,R_1)$ 可得

$G_2 \equiv GCD([x'], [x'+1]) = GCD(M,N) \wedge [x] > [x'] \wedge ([x+1] > [x] \Rightarrow [x+1] > [x'+1] \wedge [x+1] < [x] \Rightarrow [x+1] = [x'+1])$,

其中 $[x]$ 表示变量 x 的前状态, $[x']$ 表示变量 x 的后状态. 同时有 $R_1 \equiv G_2$; $R_2 \equiv G_1$.

(vii) 由程序分析以及规约 $\forall \alpha,\alpha' P(\alpha,\alpha') \subseteq [[C]]$ 可得

$GCD([x'], [x'+1]) = GCD(M,N) \wedge t_{11} = [x] \wedge t_{21} > [x'+1] \wedge ([x+1] \geqslant [x'+1] \wedge [x] > [x+1] \Rightarrow [x] > [x'] \wedge [x] < [x+1] \Rightarrow [x] = [x'])$,

有 $\forall \alpha,\alpha' P(\alpha,\alpha') \subseteq [[C]] \Rightarrow G(\alpha,\alpha')$.

综上所述证得 $R_1,G_1 \vdash P_1\{C_1\}Q_1$.

重复上述证明过程同理可证

$$R_2,G_2 \vdash P_2\{C_1\}Q_2.$$

(viii) 由程序性质可知: 对于共享变量的并行在并行系统上的全局的 R,G 而言,没有外界干扰,所以不需要依赖于外界条件,同理也不需要保证什么,所以可以设置 R = False, G = True, 则有 $R = R_1 \wedge R_2, G_1 \vee G_2 \Rightarrow G$ 成立.

(ix) 根据并发规则和辅助变量规则,同时对于程序 C_1,C_2 当运行结束时满足 $post$: $\exists Z.\, x \rightarrow Z, Z \wedge Z = GCD(M,N)$, 所以 $R,G \vdash \{pre\}C_1 \parallel C_2\{post\}$ 成立.

4　结束语

本文在抽象顺序设计语言 Apla 中加入并发、通讯和同步的语言成分,构造并发分布式计算语言 Concurrency Apla 语法及其结构化操作语义,给出了基于 Concurrent Apla 语言的共享内存并发分布式算法验证的组合验证规则. 在依赖-卫式推理的基础上,提出一种新颖的 2 层并发算法形式化验证方法,其中高层系统层用于处理并行并发级,而低层组件层用于处理顺序级. 最后,通过实例验证了方法的有效性和可行性.

本文的实例验证还在手工阶段,对于复杂的算法难免会因过程烦琐或者人为因素导致易出错等不足,为了弥补这种不足,同时提高验证并发算法的效率和正确性,下一步工作拟用 Isabelle 定理证明器对并发算法进行交互式验证. Isabelle 不仅可以用于结构化程序[17]的证明,还可用于并发算法的证明,如在 Isabelle 里利用 Owicki-Gries 方法[18]和 Rely-Guarantee 方法[19]等主流的并发程序验证方法对并发算法进行交互式验证,使得验证过程减少人为的干预. 通过使用 Isabelle 对多种算法的交互式验证,也可形成定理证明库,以供使用 Isabelle 进行程序验证的其他研究人员学习和参考.

5　参考文献

[1] 王戟,詹乃军,冯新宇,等. 形式化方法概貌[J]. 软件学报,2019,30(1):33-61.

[2] 贺飞,张立军. 软件形式化验证专题前言[J]. 软件学报,2019,30(7):1901-1902.

[3] 刘震伟,薛锦云,夏鲸,等. PAR 平台中并发分布式事务处理机制及其应用研究[J]. 江西师范大学学报: 自然

科学版,2019,43(6):649-654.
[4] 王捍贫,张磊. 形式化方法在云计算中的应用现状[J]. 广州大学学报:自然科学版,2019,18(4):69-74.
[5] 马凯旋. 基于云计算平台的形式化技术相关并行查询与检测算法的研究[D]. 南京:南京邮电大学,2018.
[6] 许明,开金宇,肖蕾. 反应式软件形式化系统研究系统分析[J]. 哈尔滨商业大学学报:自然科学版,2014,30(4):477-481.
[7] Burrows M,Abadi M,Needham R M. A logic of authentication[J]. Mathematical and Physical Sciences,1989,426(1871): 233-271.
[8] Clarke E M J,Grumberg O,Peled D A. Model checking[M]. San Mateo: MIT Press,1999.
[9] 薛锦云. PAR 方法抽象程序设计语言 Apla 技术报告[R]. 江西师范大学省高性能计算技术重点实验室,2010.
[10] Jones C B. Tentative steps toward a development method for interfering programs[J]. ACM Transactions on Programming Languages and Systems(TOPLAS),1983,5(4):596-619.
[11] Armstrong A,Gomes V B F,Struth G. Algebraic principles for rely-guarantee style concurrency verification tools[EB/OL][2019-10-16]. https://ui.adsabs.harvard.edu/abs/2013arXiv1312.1225A/.
[12] Liang Hongjin,Feng Xinyu,Fu Ming. Rely-guarantee-based simulation for compositional verification of concurrent program transformations[J]. ACM Transactions on Programming Languages and Systems(TOPLAS),2014,36(1):3.
[13] Gavran I,Niksic F,Kanade A,et al. Rely/guarantee reasoning for asynchronous programs[EB/OL][2019-10-16]. https://drops.dagstuhl.de/opus/volltexte/2015/5390/.
[14] Fedorchenko I M,Markovskii E A,Tikhonovich V I,et al. Verification of parallel shared-memory programs,Owicki-Gries method of axiomatic[EB/OL][2019-10-16]. https://link.springer.com/referenceworkentry/10.1007/978-0-387-09766-4_2090.
[15] Lengauer C. Owicki-Gries method of axiomatic verification[EB/OL][2019-10-16]. http://dx.doi.org/10.1007/978-0-387-09766-4_182.
[16] Kojima K,Igarashi A. A hoare logic for SIMT programs[EB/OL][2019-10-16]. http://www.fos.kuis.kyoto-u.ac.jp/~kozima/hl-simt-full.pdf.
[17] Norbert S. Verification of sequential imperative programs in Isabelle-HOL[EB/OL][2019-10-16]. http://www-wjp.cs.uni-sb.de/leute/private_homepages/nschirmer/pub/schirmer_phd.pdf.
[18] Nipkow T,Nieto L P. Owicki/Gries in Isabelle/HOL[EB/OL][2019-10-16]. https://dl.acm.org/doi/10.5555/645367.650813.
[19] Nieto L P. The rely-guarantee method in Isabelle/HOL[EB/OL][2019-10-16]. https://dl.acm.org/doi/10.5555/645367.650813.

The Two-Level Verification Method of Shared Memory Concurrent Distributed Algorithm Based on Concurrent Apla Language

WANG Changjing[1],YU Xiaojun[1],SHEN Deming[2],LUO Haimei[3,4],ZUO Zhengkang[1*]

(1. College of Computer Information Engineering,Jiangxi Normal University,Nanchang Jiangxi 330022,China; 2. School of Communication and Electronics,Jiangxi Science and Technology Normal University,Nanchang Jiangxi 330013,China; 3. College of Physics and Communication Electronics,Jiangxi Normal University,Nanchang Jiangxi 330022,China; 4. Key Laboratory of Photoelectronics and Telecommunication of Jiangxi Province,Jiangxi Normal University,Nanchang Jiangxi 330022,China)

Abstract: Formal verification of concurrent distributed algorithms for shared memory has become one of the most challenging problems,especially in cloud computing,multi-core,wireless sensor networks,distributed databases and blockchain environments. Based on the existing work of the research team in the formal specification language and method,derivation and validation algorithm form aspects,order to customize the generic abstract design language Apla,the concise,high abstraction for concurrent distributed computing concurrent Apla language is put forward,which supports both sequential algorithm verification and validation of concurrent distributed algorithm effectively. On the basis of rely guarantee reasoning,a novel formal verification method of two-layer concurrent distributed algorithm is proposed,in which the system layer is used for concurrency level verification and the component layer is used for sequential level verification. Finally,the validity and feasibility of the method are verified by two examples.

Key words: concurrent distributed computing; rely guarantee reasoning; concurrent Apla; formal verification

(责任编辑:冉小晓)