基于分离逻辑的并行程序性质验证方法

万 良[1,2,3] 石文昌[1,2] 冯 慧[1,2]
(中国人民大学信息学院 北京 100872)[1]
(中国人民大学数据工程与知识工程重点实验室 北京 100872)[2]
(贵州大学计算机科学与信息学院 贵阳 550025)[3]

摘 要 随着多核多线程并行执行方式的普及,并行程序形式化验证的需求日显突出。并行程序验证中执行流程的不确定性使验证的内容与目标的关系难以确定,且从并行程序直接进行性质验证会导致验证规模大。为此,提出一种基于分离逻辑的新的验证方法。该方法根据分离逻辑的程序语义描述兼有解释语义和公理语义的特点,从验证的性质出发,把要验证的性质式转换成并行语句序列的逻辑组合式,并进行整理和化简;然后,利用分离逻辑公理系统对语句序列进行验证,用验证了的断言集来求出性质的真值。实例进一步说明,此方法更有效,同时也简化了验证的规模。

关键词 霍尔逻辑,分离逻辑,并行程序,逻辑组合式,性质验证
中图法分类号 TP301 **文献标识码** A

Verification Method for Concurrent Programs Properties Based on Separation Logic

WAN Liang[1,2,3] SHI Wen-chang[1,2] FENG Hui[1,2]
(School of Information, Renmin University of China, Beijing 100872, China)[1]
(Key Laboratory of Data Engineering Knowledge Engineering, Renmin University of China, Beijing 100872)[2]
(Department of Computer Science, Guizhou University, Guiyang 550025, China)[3]

Abstract With the popularity of multi-core, multi-thread and parallel execution, there is an increasing demand for formal verification of parallel programs. The uncertainty of execution flows in parallel program verification makes it difficult to determine the relation between verification contents and targets. Verifying directly from the parallel programs will lead to large-scale verification. To this end, we proposed a new verification method based on separation logic. On the basis of the feature that the semantics of separation logic's programming language are both interpretive and axiomatic, our method transforms the property formulae to be verified into logical composition expression, and reforms and simplifies them. Then separation logic's axiom system is used to verify the expression and calculate the value of property formulae with verified assertions. Case studies further illustrate that the proposed method is effective and can reduce verification scales.

Keywords Hoare logic, Separation logic, Concurrent program, Combination expression, Property checking

1 引言

随着计算机程序的多核多线程并行执行方式的广泛使用,并行程序正确性等性质验证的需求日益突出。验证可以采用测试方法或数学逻辑方法。测试方法的弱点在于只能检验一些数据样本,测试覆盖的数据空间较小,很难对程序的性质作进一步的验证。数学逻辑方法是保证程序正确可靠的根本途径。它采用二元的思想对软件系统进行形式化,一方面对程序的形式规约(formal specification)进行形式化,另一方面对程序的性质进行形式化,然后通过相应的公理语义规则在形式规约上对性质进行验证。

在并行程序形式化验证的各种方法中,基于Petri网的验证方法可以直接反映死锁、互斥等性质,但其形式化表示的内容有限,如缺乏描述变量变化的手段[6];基于公理系统或逻辑系统的并行程序分析及验证方法(如基于时序逻辑的方法)可以描述程序的宏观功能需求,可分析程序的状态迁移过程[7],然而,使用时序逻辑的推理过程表达的细度不够且过于繁杂;基于语义的CSP等并行程序描述与分析方法尽管在数学方面有其特色,但描述和分析能力显得不足[8];SPIN[9]和SMV[10]等著名的模型检测方法在描述并行程序上非常方便,但在复杂数据结构的表达方面存在不足,且难以克服状态爆炸的问题;行为时序逻辑TLA能同时表示系统和性质[11],但

到稿日期:2013-02-03 返修日期:2013-05-22 本文受国家自然科学基金项目(61070192,91018008,61170240),北京自然科学基金(4122041),国家高技术研究发展计划(2007AA01Z414),中国人民大学科学研究基金(中央高校基本科研业务费专项资金)项目成果(+12XNLF06),贵州自然科学基金项目(J[2011]2328)资助。

万 良(1974-),男,博士,副教授,主要研究方向为信息安全、形式化方法,E-mail:wanliang@ruc.edu.cn;石文昌(1964-),男,博士,教授,主要研究方向为信息安全、可信计算与系统软件;冯 慧(1989-),女,硕士生,主要研究方向为信息安全、形式化方法。

它建立的系统模型与并行程序难以保持较好的一致性。

Reynolds 与 Hearn 在霍尔逻辑[1]基础上提出的分离逻辑[2-5](separation logic)能表示复杂的存储结构、数据结构、程序语句和其它逻辑关系。基于分离逻辑对并行程序进行形式化验证遇到的主要问题是在完全相同的环境下每次运行并行程序的执行流程和执行结果并不一定相同,这种不确定性导致无法有针对性地进行性质的验证,且从并行程序直接验证性质会导致验证的规模较大[12]。针对这些问题,本文提出一种基于分离逻辑的并行程序性质验证方法,它能对并行程序进行有效验证。

本文第 2 节介绍与本文相关的分离逻辑基本知识;第 3 节介绍提出的基于分离逻辑的验证方法;第 4 节使用提出的方法对银行柜台业务办理功能模块进行性质验证,并对验证结果进行分析说明;最后对提出的方法和解决的问题进行归纳。

2 分离逻辑

作为后文的基础,下面简要介绍分离逻辑和并发分离逻辑。分离逻辑能表达存储并进行分离,能描述各种数据结构与操作,能表达指针数据结构及其操作等;它支持逻辑运算,可以与一阶逻辑、高阶逻辑和时态逻辑等一起构造与验证程序的性质;特别地,分离逻辑在扩展后可以实现多种程序的验证,如高级语言程序、低级语言程序[13,14]、面向对象语言[15]和脚本语言程序等等,还可以对数据库系统[16]和操作系统[17]验证等等。

2.1 分离逻辑的两个运算

分离逻辑中有两个重要的基本运算,它们是分离合取和分离蕴含。

定义 1(分离合取,separating conjunction)[2,3] $*$,表示某堆能被分离成两个不相交的部分并且包含的两个断言成立。如 $s,h|=P*Q$,表示存在 h_1,h_2,有 h_1 与 h_2 空间没有相交部分,$h_1 \perp h_2$,并 $h=h_1 \cup h_2$,并且 $s,h_1|=P,s,h_2|=Q$。

定义 2(分离蕴含,pronounced magic wand or separating implication)[2,3] $-\!\!*$,宣称堆的扩展满足断言一,堆的扩展与堆的并满足断言二。如 $s,h|=P-\!\!*Q$,对任意 $h \perp h'$,$s,h'|=P,s,h \cup h'|=Q$。

2.2 分离逻辑的主要规则

分离逻辑验证的思想基于霍尔逻辑三段式。分离逻辑的规则主要有基本规则和推导规则。基本规则实现对程序语句进行形式规约,推导规则实现程序形式规约的推导演算,以证明相关断言。

(1)基本规则

基本规则可以对程序语句进行逻辑等价表示,如赋值规则可以表达程序中的赋值语句,分支规则可以表达 if 语句,循环规则可以表达 while 语句等等。主要的基本规则如下:

赋值规则:$\dfrac{}{\{f[x \leftarrow E]\}x:=E\,f(x)}$

顺序规则:$\dfrac{\{p_1\}c_1\{q_1\} \quad \{p_2\}c_2\{q_2\}}{\{p_1\}c_1;c_2\{q_2\}}$

分支规则:$\dfrac{\{p \wedge B\}c_1\{q\} \quad \{p \wedge \neg B\}c_2\{q\}}{\{p\}\text{if } B \text{ then } c_1 \text{ else } c_2\{q\}}$

循环规则:$\dfrac{\{p \wedge B\}c_1\{q\}}{\{p\}\text{while}(B)c_1\{q \wedge \neg B\}}$

(2)推导规则

推导规则实现程序形式规约的推导演算。特别在验证包含指针的程序时,分离逻辑能够较好地支持局部推理和模块化推理。扩展的分离逻辑支持更多的运算,包含一系列推导规则[3-5],适用更广泛的验证。主要的几条推导规则如下:

合取规则:$\dfrac{\{p_1\}c\{q_1\} \quad \{p_2\}c\{q_2\}}{\{p_1 \wedge p_2\}c\{q_1 \wedge q_2\}}$

析取规则:$\dfrac{\{p_1\}c\{q_1\} \quad \{p_2\}c\{q_2\}}{\{p_1 \vee p_2\}c\{q_1 \vee q_2\}}$

存量规则:$\dfrac{\{p\}c\{q\}}{\{\exists v.\ p\}c\{\exists v.\ q\}}$

全量规则:$\dfrac{\{p\}c\{q\}}{\{\forall v.\ p\}c\{\forall v.\ q\}}$

框架规则:$\dfrac{\{p_1\}c\{q_1\} \quad \{p_2\}c\{q_2\}}{\{p_1 * p_2\}c\{q_1 * q_2\}}$

2.3 并发分离逻辑

基于并发分离逻辑[18-21]可以实现对并行程序的形式验证。并行程序的执行方式是并行执行,即同时运行,用符号 || 表示这种执行方式。如下规则的条件是:有两段程序 c_1 和 c_2,谓词 p_1、p_2、q_1 和 q_2,$\{p_1\}c_1\{q_1\}$ 和 $\{p_2\}c_2\{q_2\}$ 成立,结论是:在 $p_1 * p_2$ 条件下当 c_1 和 c_2 并行执行时,有 $q_1 * q_2$ 成立。规则如下:

$$\frac{\{p_1\}c_1\{q_1\} \quad \{p_2\}c_2\{q_2\}}{\{p_1 * p_2\}c_1 \,||\, c_2\{q_1 * q_2\}}$$

其中,程序 c_1 中不能改变 p_2、c_2 和 q_2 中变量,同样,程序 c_2 中不能改变 p_1、c_1 和 q_1 中变量。这条规则表示在分离 $\{p_1\}c_1\{q_1\}$ 与 $\{p_2\}c_2\{q_2\}$ 的前提互相访问资源的情况下,c_1 与 c_2 可以并行执行。如存在临界资源时,要使用锁的措施,也可以使用 PV 原语。

3 并行程序性质验证方法

本节说明提出的基于分离逻辑的并行程序性质验证方法,方法解决的是对指定并行程序进行性质的有效验证问题。方法的验证过程如图 1 所示,分 5 个步骤:第(1)步引入时态逻辑等来表示性质表达式;第(2)步根据并行程序画出变量执行关系图;第(3)步根据变量执行关系图,找出性质中变量对应的程序语句序列,形成语句序列组合式,从而确定证明的内容;第(4)步基于分离逻辑公理系统验证组合式中语句序列,把语句序列的验证结果代入性质表达式,计算其真值,得到程序是否满足性质的结果;第(5)步对验证的结果进行分析,对验证出的缺陷进行修改。下面分 5 小节来说明这 5 个步骤。

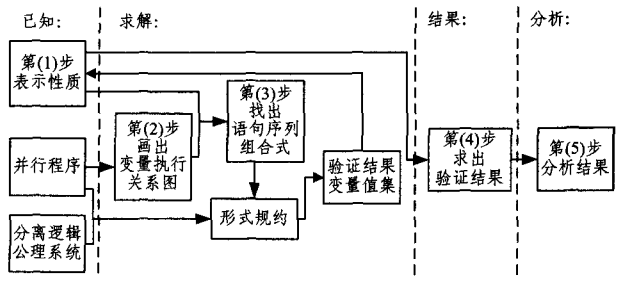

图 1 验证方法的步骤

3.1 性质的形式化表示

程序性质是程序的静态规约，为了更好地表达程序性质，引入一阶逻辑、时态逻辑、行为时态逻辑等方法表示要验证的性质。引入公平性、活性和安全性等可以进一步对性质表达式进行分类与分析。

时态逻辑是特殊的模态逻辑，模态逻辑是经典命题逻辑和一阶谓词逻辑的扩展形式，基于模态逻辑引入“可能”(◇)和“必然”(□)两个模态词，从而能够对可能世界中的命题进行描述和演算。这两个模态词有如下的关系：

$\Box p \Leftrightarrow \neg \Diamond \neg p$

$\Diamond p \Leftrightarrow \Diamond \neg \Box \neg p$

系统形式化中重要的是系统性质的表达与验证，其中系统性质中基本的是公平性。公平性[21,22]条件使转移系统定义可能的转移。在验证中利用公平性可以更好地表达相关的性质，如活性、安全性等，以更准确地描述系统的性质和验证的内容。比如，用 p 表示性质，可以用简单式子 $\Box p$，$\Diamond p$，$\Box\Diamond p$，$\Diamond\Box p$ 和复杂式子表示安全性、活性等性质。

3.2 程序的变量执行关系图

变量执行关系图用于表示变量赋值语句和语句间先后顺序。根据执行关系图可以方便地找到导致变量值改变的语句，从而进一步理解证明的内容与执行的关系。在画图之前要对程序的语句进行编号，以便用标号来表示语句。变量执行关系图中各种方框的含义如图 2 所示。

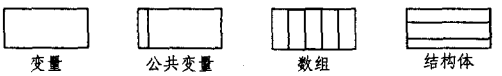

图 2 执行关系图的方框类型

方框：表示变量；

加线方框：表示公共变量；

竖表框：表示数组；

横表框：表示结构体；

实箭头：箭头的方向指向被赋值变量，箭尾连接上一次被赋值的变量，箭身的数字表示程序中对应的语句编号。它表示图中出现的语句的先后顺序，图中主要关注变量值的改变，出现的语句主要是与变量值改变相关的语句，不是所有程序的语句都会出现。其中 L 表示加锁，RL 表示解锁；

虚箭头：箭头的方向指向被赋值变量，箭尾连接在赋值式右边的变量。它表示变量之间存在赋值关系。

例 1 两个并行程序：F1，F2 各有 5 条程序语句，程序、语句编号、与变量执行关系图如下：

F1.1	L(s)	F2.1	L(s)
F1.2	x=4	F2.2	x=5
F1.3	y=5	F2.3	y=4
F1.4	if(!L(s))	F2.4	if(!L(s))
	s=y−x		s=y−x
F1.5	RL(s)	F2.5	RL(s)

程序的执行关系图，如图 3 所示。

从图 3 中很容易地找出改变 x，y，s 的语句及变量之间的关系。如变量 s，改变其值的语句是 F1.4 和 F2.4；F1.1 和 F2.1 是对其加锁的语句，F1.5 和 F2.5 是对其解锁的语句；在变量的关系上，s 与 x，y 有关。

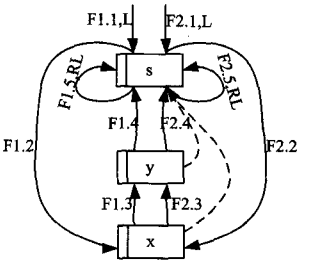

图 3 并行程序 F1 与 F2 执行关系图

3.3 语句序列组合式的转换与化简

语句序列组合式是把与性质表达式中变量相关的语句序列按一定次序组合起来的式子，一个组合式是与性质相关的一种语句组合。构造序列组合式的目的是找到验证的切入点与内容，且通过组合式的整理与化简减小验证的规模。语句序列组合式形如：

$L\downarrow(\mathrm{F1}.m1(x1),\downarrow \mathrm{F2}.m2(x2),\cdots,\downarrow \mathrm{Fn}.mn(xn))$

其中，$\downarrow Fi.mi(xi)$ 表示语句序列，它包含的语句从程序 Fi 的第一句到语句 $Fi.mi$。语句序列之间用逗号来连接。下面说明语句序列组合式中的符号、规则、转换与化简。

3.3.1 变量及语句的表示

下面分别说明变量的表示和语句的表示。

(1)变量的表示

x 表示没有加锁的变量 x；

$C{:}x$ 表示作为临界资源的变量 x；

$L{:}x$ 表示对变量 x 加锁；

$RL{:}x$ 表示对变量 x 解锁；

$P(sem{:}x)$ 表示信号量为 sem 的共享变量 x 的 P 原语；

$V(sem{:}x)$ 表示信号量为 sem 的共享变量 x 的 V 原语。

(2)语句和语句序列的表示

$F.m$ 表示程序 F 的第 m 条语句；

$F.m(x)$ 表示程序 F 的第 m 条语句，语句改变 x 的值；

$V{:}F.m(x{:}v)$ 表示程序 F 的第 m 条语句导致变量 x 得到值 v，V 表示值 v 与性质表达式中的 x 的值相同；

$W{:}F.m(x{:}v)$ 表示程序 F 的第 m 条语句导致变量 x 得到值 v，W 表示值 v 与性质表达式中的 x 的值不相同；

$\downarrow Fk.m$ 表示语句序列，序列从程序 Fk 的开始到语句 m；

$\downarrow Fk.m(x)$ 表示语句序列，序列从程序 Fk 的开始到语句 m，序列的最后一句改变 x 的值；

$f\downarrow Fk.m(x)$ 表示语句序列，序列从程序 Fk 的第 f 个副本的开始到语句 $Fk.m$，序列的最后一句改变 x 的值；

$Fk.m(x)\rightarrow Fk.n(x)$ 表示语句序列，序列从程序 Fk 的语句 $Fk.m$ 到语句 $Fk.n$，语句改变 x 的值；

$\rightarrow Fk.n(x)$ 表示语句序列，序列从表达式中程序 Fk 的当前语句到语句 $Fk.n(x)$。

(语句序列)表示语句序列的无序组合，圆括号语句序列之间的执行次序是任意的，可以任意组合；语句序列不加括号时默认为圆括号。

〈语句序列〉表示语句序列的有序组合，表达式中所有尖括号括起来的语句序列只能按表达式中的位置从左到右地顺序执行，可见尖括号包含的语句序列的组合是唯一的。

3.3.2 语句的选择运算与组合规则

这里介绍通过运算与规则实现语句序列的组合。运算有“选一”和“全选”运算，规则主要规范语句序列之间的组合方式、合并方式和分解方式。运算与规则如下：

定义 3(语句序列 1/语句序列 2)　选一运算，结果是两个语句序列只能选择其中一个。

定义 4(语句序列 1+语句序列 2)　全选运算，结果是包含这两个语句序列。

规则 1　语句序列的次序关系规则：组合式中用逗号来连接并按从左到右的次序表明先后关系。

规则 2　语句序列的组合规则：圆括号内部的语句序列可以进行前后调整与任意组合，可以形成多个语句序列组合式；尖括号内部的语句序列按从左到右的顺序进行组合，次序关系是唯一的，即只有一个组合式。尖括号和圆括号之间可以任意组合形成多个组合式。

规则 3　语句序列的合并规则：当两个同一程序副本的相邻程序段组合在一起时，可以合并。如 $Fk.m(x)$，$Fk.n(x)$，其中 $n>m$，可以合并为 $Fk.n(x)$。

规则 4　语句序列的分解规则：一个语句序列可以分解成多于两个的相邻语句序列的组合。如 $Fk.n(x)$，可以分成 $\downarrow Fk.m(x)$，$Fk.m+1(x)\rightarrow Fk.n(x)$。

3.3.3 语句序列组合式的转换与化简

首先找出性质式中出现的变量或与这些变量有关的变量；再找出与它们相关的所有语句序列，并按照规则进行整理与化简；最后根据程序语义、性质语义和证明方法找出所有可能的语句序列的组合，并进行整理与化简，得到一个或多个语句序列组合式。

下面举例说明如何把性质表达式转化成语句序列组合式。

例 2　根据例 1 中的程序，证明性质表达式：

$(x=4\wedge y=5)=>\square s=1$

思路：按本步的方法找出所有的语句序列组合式并整理与化简，然后基于分离逻辑对逻辑组合式逐一证明，当得到确定的结果时，不需要再证明剩下的组合式，证明结束。下面分 3 步说明本题的求解。

第 1 步　找出导致性质式中变量发生改变的所有语句。

找出上式中出现的或相关的所有变量：x,y,s；再找与它们相关的语句，从图 3 中容易找出：x 有两个箭头，即 x 的赋值语句有两个：F1.2，F2.2，其中 F1.2 是使表达式中的子表达($x=4$)为真的语句。同样 y 的赋值语句有两个：F1.3，F2.3，其中 F1.3 是使表达式中的子表达($y=5$)为真的语句。s 的赋值语句有两个：F1.4，F2.4，这两个序列是互斥的。于是性质表达式的证明与如下 6 个语句序列有关：

$\downarrow W$:F2.2(x:5)，$\downarrow W$:F2.3(y:4)，$\downarrow V$:F1.2(x:4)，$\downarrow V$:F1.3(y:5)，$\downarrow$F1.4(C:s)，$\downarrow$F2.4(C:s)

因 s 是互斥资源，对 s 的加锁指令 F1.1 和 F2.1 会影响后面程序的执行。即当 F1.1 先执行时，就会执行 F1.4；而当 F2.1 先执行时，就会执行 F2.4。所以把这个语句加到上面 6 个语句序列中，形成所有与性质表达式相关的共 8 个语句或语句序列。

第 2 步　找到由这些语句序列组成的所有可能的逻辑组合并加以验证。

从性质式可见，$\downarrow V$:F1.2(x:4)，$\downarrow V$:F1.3(y:5)是使前件成立的语句序列，在保持前件为真的前提下，其它的语句序列可以任意进行组合，由此可得到所有可能的组合。下面分析其中的一个组合：

F2.1，$\downarrow W$:F2.2(x:5)，$\downarrow W$:F2.3(y:4)，($\downarrow$F2.4(L:s))，〈$\downarrow V$:F1.2(x:4)〉，〈$\rightarrow V$:F1.3(y:5)〉

根据合并规则进行合并：

$\downarrow W$:F2.2(x:5)，$\downarrow W$:F2.3(y:4)，($\downarrow$F2.4(L:s))合并为($\downarrow$F2.4(L:s))，〈$\downarrow V$:F1.2(x:4)〉，〈$\rightarrow V$:F1.3(y:5)〉合并为〈$\downarrow V$:F1.3(y:5)〉

合并后序列组合式变成：

($\downarrow$F2.5(L:s))，〈$\downarrow V$:F1.3(y:5)〉

于是对性质的证明转换成对如上两个语句序列的证明。

第 3 步　根据语句序列组合式求出验证结果。

(1)验证语句序列$\downarrow$F2.5(L:s)，得到 $x=5,y=4,s=-1$；

(2)验证语句〈$\downarrow V$:F1.3(y:5)〉后 $x=4,y=5$。最终有 $x=4,y=5,s=-1$，代入性质$(x=4\wedge y=5)=>\square s=1$得：

$4=4\wedge 5=5=>\square -1=1$

显然，后件不成立，所以性质式不成立。

证明结束，其它组合式不需要再证明。

3.4 基于并发分离逻辑公理系统的验证

经过前 3 步，性质表达式被转换成若干个语句序列组合式，接下来就是对每个语句序列进行验证。验证的方法采用霍尔逻辑方法，使用并发分离逻辑规则库中的规则来验证。

一个语句序列组合式的验证可分为 2 小步，第 1 小步是对语句序列组合式进行验证，第 2 小步是根据验证的结果集计算出性质表达式。

3.4.1 组合式中语句序列的验证

一个语句序列组合式由多个语句序列组成，需要对每个语句序列进行验证。其中一个语句序列的验证思想是，在已知谓词条件下，从语句序列的第一条语句开始逐句利用分离逻辑规则进行形式化推导，直到最后一语，得到一个断言。思想表示如下。

如图 4 所示，验证程序语句序列时，是在分离逻辑规则下逐句写出形式规约和验证的结果：在谓词 P_1 条件下，验证语句 1，根据分离逻辑规则，得到谓词 P_2 成立；在谓词 P_2 条件下，验证语句 2，得到谓词 P_3 成立；…，最后验证语句 n 得到谓词 P_{n+1} 成立。根据 2.2 节(2)中的推导规则和如下的延推规则：

$$\frac{p_1\rightarrow p_2 \quad \{p_1\}c\{q_1\} \quad q_1\rightarrow q_2}{\{p_2\}c\{q_2\}}$$

得知在条件 P_1 成立时，执行了该语句序列后的结果是 P_{n+1}。

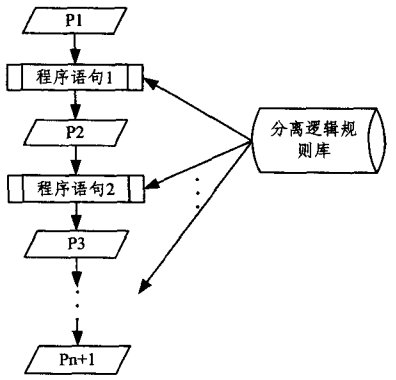

图 4　语句序列验证流程

3.4.2　性质表达式值的求解

根据语句序列组合式的验证结果集来求出性质表达式的值。在第 1 步的基础上，第 2 步容易求出。思路如图 5 所示，按语句序列在组合式中的次序，用验证的结果逐一修改性质表达式中相关变量的值，最后计算性质表达得出结果。如例 2 中的第 3 步。

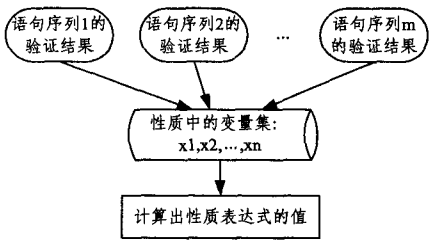

图 5　性质表达式的计算

3.5　验证结果的分析

验证结果表明程序是否具有某种性质。如果性质是安全性质，成立时表明程序具有该安全性，反之表明该安全性缺失，就要进行针对性的修改；如果性质是可靠性质，成立时表明程序满足该可靠性，反之表明程序不可靠，需要进行针对性修改。同时对验证覆盖的空间还要加以评估，如覆盖的空间不足，还要增加验证的性质，直到达到验证的目标为止。

4　银行柜台业务办理功能模块的验证

下面运用本方法验证银行柜台业务办理功能模块的性质，通过验证进一步说明方法的有效性。

4.1　银行柜台业务办理功能模块说明

本系统功能是处理银行分理处的前台业务，实现取款和存款两项业务。在业务区开设多个窗口，可同时服务多个储户完成存取款的业务要求。其中，用户排队用到了队列机制。

队列机制管理用户按先来先服务的原则获得服务。基本操作有：进队操作 enqueue(&q, $*user$, $utype$)，其中 q 表示队列，$user$ 表示进队的用户，$utype$ 表示用户分类；出队操作 $*dequeue(\&q)$；队元素查询操作 alloqueue(&q)。

存取款业务办理模块实现储户的存取款业务。每个业务窗口 Window 运行业务模块 business()的一个副本。其中 init()是系统的初始化模块。business()程序语句与语句编号如下：

```
Sysinit;
  init(){q,M,sinM,soutM,inacc=outacc=0;}
Window1,Window2…;
business(){
1.    if(user=dequeue(q)==null) return;
2.    if(!userlogin()) return;
3.    P(user);
4.    while(ask=getvalue()){
4.1     user->banswer='n';
4.2     if(ask==DEPOSIT){
4.2.1     askcount=getvalue();
4.2.2     if(askcount>=0&&
          askcount<sinM){
4.2.2.1     user->banswer='y';
4.2.2.2     user->saving+=askcount;
4.2.2.3     inacc+=askcount;
4.2.2.4     M+=askcount;
          }
        }
4.3   if(ask=WITHDRAW){
4.3.1     P(M);P(outcount);
4.3.2     askcount=getvalue();
4.3.3     if(soutM>=askcount&&
          M>askcount&&
          user->saving>askcount&&
          askcount>=0){
4.3.3.1     user->banswer='y';
4.3.3.2     user->saving-=askcount;
4.3.3.3     outacc+=askcount;
4.3.3.4     M-=askcount;
          }
4.3.4     V(M);V(outcount);
      }
5.    V(user);
      }
    }
```

4.2　业务办理功能模块的性质验证

本小节按照方法中提出的 5 步，对程序银行柜台业务办理系统的性质进行验证。

4.2.1　验证性质的形式化表示

设置要验证性质：对所有客户，如果他提出取款要求，且分理处存入累计与取款累计之差多于用户申请取款额，又当前现金总额大于申请取款数且单笔最大取款数大于等于申请取款数，则取款申请一定能被应答。性质表示如下：

$\forall u \in U(u.ask==\text{WITHDRAW} \wedge inacc\text{-}outacc > u.askcount \wedge M > u.askcount \wedge soutM >= u.askcount \Rightarrow \Box u.banswer='y')$

其中，$u.ask$ 为用户提出的申请，WITHDRAW 为取款申请，$inacc$ 为当天存入累计，$outacc$ 为当天取款累计，$u.askcount$ 为用户提出的申请金额，M 表示现金总额，$soutM$ 表示单笔最大取款数，$u.banswer$ 表示对用户的应答。

4.2.2　business()的变量执行关系图

函数 business()的执行关系图如图 6 所示。

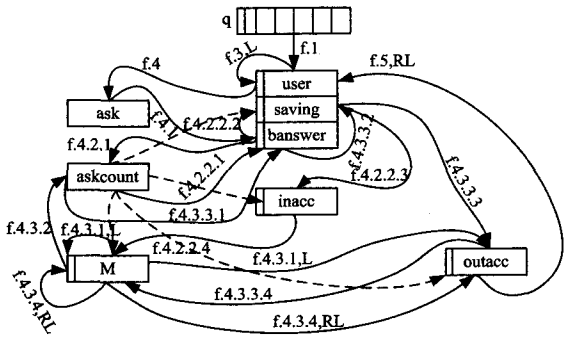

图 6　变量的执行关系图

4.2.3　语句序列组合式的转换与化简

如图 6 所示，可以清楚地找出性质中的变量和相关的变量：*u. ask*，*inacc*，*outacc*，*askcount*，*M*，*soutM* 和 *u. banswer*，并且找出变量相关的语句，如表 1 所列。

表 1　性质中的变量与程序语句对应表

变量	u. ask	inacc	outacc	askcount	M	u. banswer
语句	4	4.2.2.3	(4.3.1,L) 4.3.3.3 (4.3.4,RL)	4.2.1 4.3.2	4.2.2.4 (4.3.1,L) 4.3.3.4 (4.3.4,RL)	(3,L) 4.1 4.2.2.1 4.3.3.1 (5,RL)

根据表 1，把性质表达式转换成语句序列组合式：

$L(\downarrow f.4(ask), \downarrow f.4.2.2.3(inacc), \downarrow f.4.3.3.3(outacc), \downarrow f.4.2.2.4(M)/\downarrow f.4.3.3.4(M), \downarrow f.4.2.1(askcount)/f.4.3.2(askcount)) \Rightarrow L(\downarrow f.4.1(banswer)/\downarrow f.4.2.2.1(banswer)/\downarrow f.4.3.3.1(banswer))$

根据语义对组合式进行分析。从语义看，$ask == WITHDRAW$ 成立，执行取款分支 4.3，而组合式中 4.2.2.1，4.2.2.3，4.2.2.4 属于存款分支 4.2。可见这里有多种可能：执行程序的一个副本或多个副本。现在分别加以讨论。

情况 1　执行一个副本进行取款的情况。

这种情况下没有存款累计，因此不能执行。

情况 2　执行多个副本的情况。

4.3.2 与 4.3.3.1 执行同一个副本 $f1$，4.2.2.3 执行的副本 f 与 4.2.2.4 执行的副本 f 可以相同也可以不同。执行多个副本时，共享变量 *outacc*，*user* 是加锁的，*M* 是执行 4.3 分支加锁的。加锁和解锁在验证中标注，在组合式中省略。于是，性质式转化为：

$L(\downarrow f1.4(ask), \downarrow f2.4(ask), \downarrow f1.4.2.2.3(inacc), \downarrow f2.4.3.3.3(outacc), \downarrow f2.4.3.3.4(M), f2.4.3.2(askcount)) \Rightarrow L(\downarrow f2.4.1(banswer), \downarrow f2.4.3.3.1(banswer))$

整理合并后，上式转化为：

$L(\downarrow f1.4.2.2.3, \downarrow f2.4.3.3.4)$

显然通过序列组合式的化简式来进行验证，有了针对性且验证的规模减小很多。

4.2.4　基于分离逻辑规则的组合式验证

根据条件，利用相应分离逻辑公理规则对语句序列 $\downarrow f1.4.2.2.3$ 和 $\downarrow f2.4.3.3.4$ 进行验证，把验证的结果代入原性质式，结果不成立。说明如下：

验证语句序列 $\downarrow f1.4.2.2.3$ 时，$\{p\}c:\downarrow f1.4.2.2.3\{q\}$ 这 3 部分中条件 p 是用户 $u1$ 正常登录，并将存入 1000 成立时，求 q。通过对语句序列中语句的逐一验证，结果 q 是：

$u1 -> saving = \_ + 1000 \wedge inacc = \_ + 1000$

验证语句序列 $\downarrow f2.4.3.3.4$ 时，$\{p\}c:\downarrow f2.4.3.3.4\{q\}$ 这 3 部分中条件 p 是用户 $u2$ 是将办理业务的队首用户，$u2$ 正常登录，并将取款 800 元，求 q。结果 q 是：

$banswer = "n"$。

把验证的结果代入性质表达式，后件不成立，即性质不成立。

4.2.5　验证结果的分析

分析：导致问题的原因是现金总额 M 与单笔可取最大数 *soutM* 的关系没有在程序中确定，不是由于没有在存储中给现金总额 M 和存款累计 *inacc* 加锁而造成的。我们给程序中语句 4.1 后面加上一句：

$soutM = 0.2 * M$

即可解决。然后，我们再进行验证，得到断言成立。证明中没有发现没有在存储中给现金总额 M 和存款累计 *inacc* 加锁带来的错误，但不能断定是否存在潜在的风险。

结束语　并行程序验证的复杂性在于执行流程的不确定性和由此带来的执行规模的扩大，这使得验证的内容和验证的目标之间的关系难以确定且验证的规则较大。为更好地解决这个问题，本文提出了一种基于分离逻辑的验证方法。该方法设计变量的执行关系图来描述变量相关的语句及执行关系；设计了语句序列组合式并把验证的性质转换成变量相关的语句序列的组合式，使得性质表达式与并行程序的语句相关联；通过语句序列式的验证结果计算出性质表达式的真值；最后，根据验证的结果对程序进行针对性修改和完善。该方法使验证的内容有针对性且验证规模得以化简，使得验证更为有效。基于本文提出的方法，进一步可以开展的研究工作包括形式验证自动化方法的实现、复杂并行程序的验证和系统的形式化验证等等。

参 考 文 献

[1] Hoare C A R. An Axiomatic Basis for Computer Programming [J]. Communications of the ACM, 1969, 12(10): 576-580

[2] O'Hearn P, Reynolds J, Yang H. Local Reasoning about Programs that Alter Data Structures[C]// Proceedings of 15th Annual Conference of the European Association for Computer Science Logic. LNCS, Springer-Verlag, 2001: 1-19

[3] Reynolds J C. Separation logic: A logic for shared mutable data structures. In LICS[C]// IEEE Computer Society. 2002: 55-74

[4] Reynolds J C. An overview of separation logic[C]// Meyer B, Woodcock J, eds. VSTTE, Lecture Notes in Computer Science. Springer, 2005, 4171: 460-469

[5] O'Hearn P W. Tutorial on separation logic [C]// Gupta A, Malik S, eds. CAV. Springer, 2008, 5123: 19-21

[6] Ding L, Dong L D, Piao Y. Deadlock prevention policy of concurrent programming base on Petri net[J]. Journal of Zhejiang University: Science Edition, 2012, 39(1)

[7] Owicki S, Lamport L. Proving liveness properties of concurrent programs[J]. ACM Transactions on Programming Languages and Systems(TOPLAS), 1982, 4(3): 455-495

[8] Kleine M, Bartels B, Göthel T, et al. LLVM2CSP: extracting csp

models from concurrent programs[M]. NASA Formal Methods, Springer Berlin Heidelberg, 2011:500-505

[9] 肖美华,薛锦云. 基于 SPIN/Promela 的并发系统验证 [J]. 计算机科学,2004,31(8):201-203

[10] Witkowski T, Blanc N, Kroening D, et al. Model checking concurrent linux device drivers[C]// Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering. ACM, 2007:501-504

[11] Grov G, Michaelson G, Ireland A. Formal verification of concurrent scheduling strategies using TLA[C]// Parallel and Distributed Systems, 2007 International Conference on. IEEE, 2007, 2:1-6

[12] Apt K R, De Boer F S, Olderog E R. Verification of sequential and concurrent programs[M]. Springer, 2010

[13] Feng X, Shao Z, Dong Y, et al. Certifying low-level programs with hardware interrupts and preemptive threads[C]// PLDI'08: Conference on Programming Language Design and Implementation. ACM, 2008:170-182

[14] Feng X, Shao Z, Vaynberg A, et al. Modular verification of assembly code with stack-based control abstractions[C]// PLDI'06: Conference on Programming Language Design and Implementation. ACM, 2006:401-414

[15] Stephan V S, Cristiano C, Bertrand M. Verifying Executable Object-Oriented Specifications with Separation Logic[C]// 24th European Conference. Maribor, Slovenia, June 2010:21-25

[16] Aquinas H, Andrew W A, Francesco Z N. Oracle Semantics for Concurrent Separation Logic[C]// ESOP. April 2008

[17] Alexey G, Honseok Y. Modular verification of Preemptive OS Kernels[C]// ICEP'11. 2011

[18] O'Hearn P W. Resources, concurrency and local reasoning[C]// Gardner P, Yoshida N, eds. CONCUR, volume 3170 of Lecture Notes in Computer Science. Springer, 2004:49-67

[19] Yu S W. Formal verification of concurrent programs[D]. Durham University, 1999

[20] Brookes S. A semantics for concurrent separation logic [J]. Theor. Computer Science, 2007, 375:227-270

[21] Hayman J, Winskel G. Independence and concurrent separation logic[C]// LICS 2006. 2006:147-156

(上接第 143 页)

参 考 文 献

[1] Shamir A. Identity-based cryptosystems and signature schemes [C]// Proceeding of Crypto'84. LNCS 196, Berlin: Springer-Verlag, 1984:47-53

[2] Al-Riyami S S, Paterson K G. Certificateless public key cryptography[C]// Proceeding of ASIACRYPT 2003. LNCS 2894, Berlin: Springer-Verlag, 2003:452-473

[3] 于刚,韩文报. 具有代理解签密功能的无证书签密方案[J]. 计算机学报,2011,34(7):1291-1299

[4] Yang Guo-min, Tan C H. Certificateless cryptography with KGC trust level 3[J]. Theoretical Computer Science, 2011, 412(39): 5446-5457

[5] Barbosa M, Farshim P. Certificateless signcryption[C]// Proceeding of ASIACCS'2008. ACM, 2008:369-372

[6] Zheng Yu-liang. Digital signcryption or how to achieve cost(signature and encryption)<<cost(signature)+cost(encryption) [C]// Proceeding of CRYPTO'1997. LNCS 1294, Berlin: Springer-Verlag, 1997:165-179

[7] Aranha D, Castro R, Lopez J, et al. Efficient certificateless signcryption [EB/OL]. http://sbseg2008.inf.ufrgs.br/anais/data/pdf/st03_01_resumo.pdf, 2009-03-21

[8] Wu Chen-huang, Chen Zhi-xiong. A new efficient certificateless signcryption scheme[C]// Proceedinf of ISISE'2008. IEEE, 2008:661-664

[9] Selvi S S D, Vivek S S, Rangan C P. Cryptanalysis of certificateless signcryption schemes and an efficient construction without pairing[C]// Proceeding of Inscrypt 2009. LNCS 6151, Berlin: Springer-Verlag, 2010:75-92

[10] Selvi S S D, Vivek S S, Shukla D, et al. Efficient and provably secure certificateless multi-receiver signcryption[C]// Proceeding of ProvSec 2008. LNCS 5324, Berlin: Springer-Verlag, 2008:52-67

[11] Selvi S S D, Vivek S S, Rangan C P. A note on the certificateless multi-receiver signcryption scheme[EB/OL]. http://eprint.iacr.org/2009/308, 2009-6-26

[12] Miao Song-qin, Zhang Fu-tai, Zhang Lei. Cryptanalysis of a certificateless multi-receiver signcryption scheme[C]// Proceeding of MIMES 2010. IEEE, 2010:593-597

[13] Li Fa-gen, Shirase M, Takagi T. Certificateless hybrid signcryption[C]// Proceeding of ISPEC 2009. LNCS 5451, Berlin: Springer-Verlag, 2009:112-123

[14] Selvi S S D, Vivek S S, Rangan C P. Certificateless KEM and hybrid signcryption schemes revisited[C]// Proceeding of ISPEC 2010. LNCS 6047, Berlin: Springer-Verlag, 2010:294-307

[15] 孙银霞,李晖. 高效无证书混合签密[J]. 软件学报,2011,22(7):1690-1698

[16] Liu Zhen-hua, Hu Yu-pu, Zhang Xiang-song, et al. Certificateless signcryption scheme in the standard model[J]. Information Sciences, 2010, 180(3):452-464

[17] Weng Jian, Yao Guo-xiang, Deng R H, et al. Cryptanalysis of a certificateless signcryption scheme in the standard model[J]. Information Sciences, 2011, 181(3):661-667

[18] Jin Zheng-ping, Wen Qiao-yan, Zhang Hua. A supplement to Liu et al.'s Certificateless signcryption scheme in the standard model[EB/OL]. http://eprint.iacr.org/2010/252, 2010-05-03

[19] Luo Ming, Zou Chun-hua, Xu Jian-feng. Certificateless Broadcast Signcryption with Forward Secrecy[C]// Conference on Computational Intelligence and Security. 2011:910-914

[20] 刘文浩,许春香. 无双线性配对的无证书签密方案[J]. 软件学报,2011,22(8):1918-1926

[21] Xie Wen-jian, Zhang Zhang. Certificateless Signcryption without Pairing[EB/OL]. http://eprint.iacr.org/2010/187, 2010-06-20

[22] Pointcheval D, Stern J. Security arguments for digital signatures and blind signatures[J]. Journal of Cryptology, 2000, 13(3): 361-396

[23] Chen L, Cheng Z, Smart N P. Identity-based key agreement protocols from pairings[J]. International Journal Information Security, 2007, 6(4):213-241