软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(8):2010−2025 [doi: 10.13328/j.cnki.jos.005272]

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

基于分离逻辑的程序验证研究综述*

秦胜潮, 许智武, 明 仲

(深圳大学 计算机与软件学院,广东 深圳 518060)

通讯作者: 许智武, E-mail: xuzhiwu@szu.edu.cn

摘 要: 自 20 世纪 60 年代以来,虽然有 Floyd-Hoare 逻辑的出现,但使用形式化工具对命令式程序的正确性和可靠性进行自动验证,一直被认为是极具挑战性、神圣不可及的工作.20 世纪末,由于更多科研的投入,特别是微软、IBM 等大型公司研发部门的大量人力、物力的投入,程序验证方面在 21 世纪初取得了不少进展,例如用于验证空客代码无运行时错误的 ASTRÉE 工具、用于 Windows 设备驱动里关于过程调用的协议验证的 SLAM 工具.但这些工具并没有考虑动态创建的堆(heap):ASTRÉE 工具假设待验证代码没有动态创建的堆,也没有递归;SLAM 假设待验证系统已经有了内存安全性.事实上,很多重要的程序,例如 Linux 内核、Apache、操作系统设备驱动程序等,都涉及到对动态创建堆的操作.如何对这类操作堆的程序(heap-manipulating programs)进行自动验证仍然是一个难题.2001 年~2002 年,分离逻辑(separation logic)提出后,其分离(separation)思想和相应的框(frame)规则使得局部推理(local reasoning)可以很好地应用到程序验证中.自 2004 年以来,基于分离逻辑对操作动态创建堆的程序进行自动验证方面的研究有了很大的进展,取得了很多令人瞩目的成果,例如 SpaceInvader/Abductor,Slayer,HIP/SLEEK,CSL 等工作.着重对这方面的部分重要工作进行阐述.

关键词: 分离逻辑;程序分析;程序验证;内存安全性;功能正确性
中图法分类号: TP311

中文引用格式: 秦胜潮,许智武,明仲.基于分离逻辑的程序验证研究综述.软件学报,2017,28(8):2010−2025. http://www.jos.org.cn/1000-9825/5272.htm

英文引用格式: Qin SC, Xu ZW, Ming Z. Survey of research on program verification via separation logic. Ruan Jian Xue Bao/ Journal of Software, 2017,28(8):2010−2025 (in Chinese). http://www.jos.org.cn/1000-9825/5272.htm

Survey of Research on Program Verification via Separation Logic

QIN Sheng-Chao, XU Zhi-Wu, MING Zhong

(College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China)

Abstract: Since 1960s, automated program verification has been considered as an extremely difficult topic in Computer Science, despite of the emergence of Floyd-Hoare logic. Since 1990s, with more efforts and resources devoted to this area, especially the contributions from industry communities including Microsoft Research and IBM Research Centre, there has been noticeable progress made in program verification early this century. Two typical examples include ASTRÉE, used to verify the absence of runtime errors in Airbus code, and SLAM, employed to verify protocol properties of procedural calls in device drivers by Microsoft. However, none of these tools considers heap. Specifically, ASTRÉE assumes no dynamic pointer allocation and no recursion, while SLAM assumes memory safety. Many important programs/systems that exist nowadays, such as Linux, Apache, and device drivers, all make frequent use of heap. Automated verification of heap-manipulating programs remains a very challenging topic. The emergence of separation logic in 2001-2002 has shed

* 基金项目: 国家自然科学基金(61373033, 61502308, 61672358); 深圳市科技创新委员会基础研究项目(JCYJ201418193546117)
Foundation item: National Natural Science Foundation of China (61373033, 61502308, 61672358); Shenzhen Science and Technology Innovation Commission Project (JCYJ201418193546117)
收稿时间: 2016-11-25; 修改时间: 2017-01-01; 采用时间: 2017-02-08; jos 在线出版时间: 2017-03-17
CNKI 网络优先出版: 2017-03-17 14:37:37, http://kns.cnki.net/kcms/detail/11.2560.TP.20170317.1437.006.html

some light into this area. With the key idea of separation and the elegant frame rule, local reasoning can now be readily employed in program verification. Since 2004, there have been a large body of research work dedicated to automated program verification via separation logic, e.g. SpaceInvader/Abductor, Slayer, HIP/SLEEK, CSL etc. This paper offers a survey on a number of important research work along this line.

Key words: separation logic; program analysis; program verification; memory safety; functional correctness

自 20 世纪 60 年代以来,虽然有 Floyd-Hoare 逻辑的出现,但是使用形式化工具对命令式程序的正确性和可靠性进行自动验证,一直被认为是极具挑战性、神圣不可及的工作.20 世纪末,由于更多的科研投入,特别是微软、IBM 等大型公司研发部门的大量人力物力的投入,程序验证方面在 21 世纪初取得了不少进展,例如用于验证空客代码无运行时错误的 ASTRÉE 工具、用于 Windows 设备驱动里关于过程调用的协议验证的 SLAM 工具.但这些工具并没有考虑动态创建的堆(heap):ASTRÉE 工具假设没有动态创建的堆,也没有递归;SLAM 假设待验证系统已经有了内存安全性.事实上,很多重要的程序,例如 Linux 内核、Apache、操作系统设备驱动程序等,都涉及到对动态创建堆的操作.如何对这类操作堆的程序进行自动验证仍然是个难题.2001 年~2002 年,分离逻辑(separation logic)提出后,其分离(separation)思想和相应的框(frame)规则使得局部推理(local reasoning)可以很好地应用到程序验证中.自 2004 年以来,基于分离逻辑对操作动态创建堆的程序进行自动验证方面的研究有了很大的进展,取得了很多令人瞩目的成果,例如 SpaceInvader/Abductor,Slayer,HIP/SLEEK,Verifast,CSL 等工作.有些验证工作针对程序的内存/指针安全性(memory/pointer safety),如 SpaceInvader/Abductor;有些工作则同时考虑内存安全性和功能正确性,如 HIP/SLEEK 系统.有些工作只注重于程序验证,依赖用户提供验证所需的程序规范和断言,如 Smallfoot.有些工作则将静态分析和验证结合起来,以提高程序验证的自动化程度,如 Abductor,HIP/SLEEK 等.

本文第 1 节介绍分离逻辑基础,包括 Hoare 逻辑的基本知识.第 2 节着重介绍一些使用分离逻辑进行程序分析与验证的典型工作.第 3 节介绍用于并发程序验证的分离逻辑及相关工作,包括基于高阶逻辑的相关工作和用于在弱内存模型上进行程序验证的逻辑系统.最后总结全文并展望未来的研究重点.

1 分离逻辑基础

分离逻辑是 Hoare 逻辑的一种扩展,所以我们先对 Hoare 逻辑进行简要介绍.

1.1 Hoare逻辑

Hoare 逻辑[1]的核心是一套基于公理语义的程序正确性推理系统,自提出以来,已成为所有程序验证系统的核心基础.Hoare 逻辑里使用的断言(assertion)概念也被很多现代语言所采用.Hoare 在其关于 Hoare 逻辑的开创性论文[1]里首次提出了用于刻画程序行为的 Hoare 三元组$\{P\}\ C\ \{Q\}$,这里,C 是所考虑的程序语言里的一个程序,P 和 Q 通常都是一阶逻辑里的公式,用于描述程序 C 里的程序变量需满足的约束条件,其中,P 称为前置条件(precondition),Q 称为后置条件(postcondition).

Hoare 三元组$\{P\}\ C\ \{Q\}$刻画的是程序 C 的一个部分正确性规范(partial correctness specification).我们说程序 C 满足规范$\{P\}\ C\ \{Q\}$(或者说规范$\{P\}\ C\ \{Q\}$为真),当且仅当下面的条件成立:如果程序 C 从满足前置条件 P 的任何初始状态开始执行,并且如果该执行终止,则执行结束时的程序状态满足后置条件 Q.部分正确性规范并没有对程序的终止性做任何要求,所以如果从满足前置条件 P 的初始状态开始,程序 C 的执行不终止,规范$\{P\}\ C\ \{Q\}$也是满足的.举个简单的例子,规范{True} WHILE True DO *SKIP* $\{Q\}$为真.值得注意的是,这里部分正确性规范的定义只局限于那些不涉及堆(heap)上任何操作的程序.后面在介绍分离逻辑部分时,我们会讨论到这个定义对操作堆的程序(heap-manipulating programs)是不够的.

下面我们通过一个简单的程序设计语言来展示 Hoare 逻辑的推理规则.假设我们考虑的程序语言的程序可以通过并且只能通过如下规则生成:

$$C::=SKIP|X:=E|C;C|\text{IF } B \text{ THEN } C \text{ ELSE } C|\text{WHILE } B \text{ DO } C.$$

这里,C 表示程序,B 和 E 分别代表布尔表达式和算术表达式.

- $B::=E=E|E\geq E|B\wedge B|\neg B$;
- $E::=X|N|E+E|-E|\ldots$

对于这个简单程序设计语言,一个最基本的 Hoare 逻辑推理系统可以由如下的公理和规则组成.

- 空语句公理:$\{P\}\ SKIP\ \{P\}$.
- 赋值语句公理:$\{P[E/X]\}\ X:=E\ \{P\}$.
- 顺序复合规则:$\dfrac{\{P\}\ C_1\ \{R\}\ \ \{R\}\ C_2\ \{Q\}}{\{P\}\ C_1;C_2\ \{Q\}}$.
- 条件复合规则:$\dfrac{\{P\wedge B\}\ C_1\ \{Q\}\ \ \{P\wedge\neg B\}\ C_2\ \{Q\}}{\{P\}\ \text{IF}\ B\ \text{THEN}\ C_1\ \text{ELSE}\ C_2\ \{Q\}}$.
- 基本循环规则:$\dfrac{\{R\wedge B\}\ C\ \{R\}}{\{R\}\ \text{WHILE}\ B\ \text{DO}\ C\ \{R\wedge\neg B\}}$.
- 后果规则:$\dfrac{P\Rightarrow P_1\ \ \{P_1\}\ C\ \{Q_1\}\ \ Q_1\Rightarrow Q}{\{P\}\ \ C\ \{Q\}}$.

最后这条后果规则(rules of consequences)也可以拆成两条规则:一条注重于前置条件的强化,另一条注重于后置条件的弱化.如果将后果规则和基本循环规则组合起来,就可以得到更一般性的循环规则(R 为循环不变式).

- 一般循环规则:$\dfrac{P\Rightarrow R\ \ \{R\wedge B\}\ C\ \{R\}\ \ R\wedge\neg B\Rightarrow Q}{\{P\}\ \text{WHILE}\ B\ \text{DO}\ C\ \{Q\}}$.

在引入分离逻辑之前,我们简要介绍一下程序验证器的一般架构(如图 1 所示).机械化的程序验证器(mechanised program verifier)的输入通常是待验证的程序和需要验证的程序性质,例如 Hoare 逻辑里的一个部分正确性规范(Hoare 三元组)$\{P\}\ C\ \{Q\}$.一般情况下,需要人工对这样的程序规范做些标注处理,例如给出中间的断言、循环不变式等标注.当然,在很多情况下,人们也尝试通过静态分析的方法来尽可能地减少对人工标注的需求.下一步由程序验证器里的验证条件自动生成器来对合理标注的程序规范进行处理,生成一些验证条件(verification condition).这些验证条件会交由合适的(自动)定理证明器处理.一般情况下,自动定理证明器会处理掉很多验证条件,而无法自动证明的验证条件会交回给用户来查看,用户决定是否加入新的引理、是否再次调用同一个或者不同的定理证明器,直到证明完所有的验证条件.自动程序验证希望达到的目标是所有上述步骤都由计算机程序自动完成.

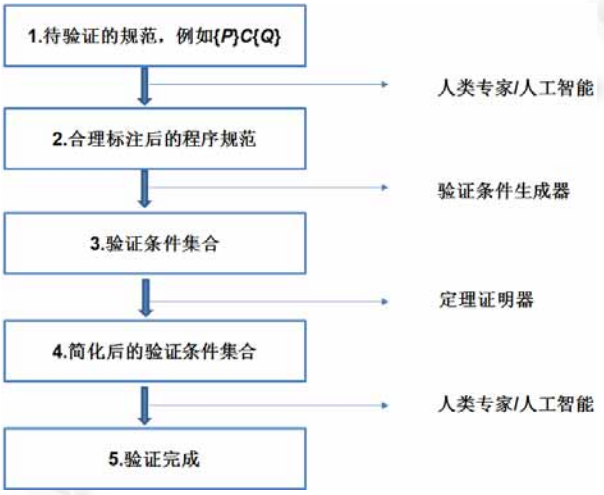


Fig.1 A general framework for mechanized program verifier

图 1 机械化程序验证器的一般架构

1.2 分离逻辑

在介绍分离逻辑之前,我们通过例子来说明为什么上面介绍的 Hoare 逻辑在有堆访问(例如指针)的情况下是不够的.在没有堆访问的情况下,Hoare 逻辑有一条非常有用的恒常性规则(rule of constancy).

- 恒常性规则: $\dfrac{\{P\}\ C\ \{Q\}}{\{P\wedge R\}\ C\ \{Q\wedge R\}}$ $(Modifies(C)\cap Free(R)=\varnothing)$.

但在有堆访问(heap accesses)的情况下,由于指针变量别名等原因,这条规则就变得不正确了.例如,使用恒常性规则得到的下面的推理是不可靠的(虽然在这一特殊情况下,结论中的 Hoare 三元组仍然成立).

$$\frac{\{\exists t.X\mapsto t\}[X]:=5\ \{X\mapsto 5\}}{\{(\exists t.X\mapsto t)\wedge(Y\mapsto 5)\}[X]:=5\ \{(X\mapsto 5)\wedge(Y\mapsto 5)\}}.$$

原因是变量 X 和 Y 可能互为别名(即 $X=Y$),而 X 和 Y 所指向的堆单元里存储的内容可能不一样.这启发人们去寻找新的推理规则来处理堆访问和指针别名问题,而分离逻辑恰好可以为此提供很好的解决方案.

为了更好地支持对操作堆上动态创建数据结构的程序的验证,分离逻辑[2−4]对 Hoare 逻辑做了扩展,(在基础逻辑中)新引入了两个逻辑算子,分离合取(separation conjunction)*和分离蕴含(separation implication)−−*操作.分离逻辑有经典分离逻辑(classic separation logic)和直觉分离逻辑(intuitionistic separation logic)两种.已有的基于分离逻辑的程序验证基本上都是基于前者.所以本文的讨论仅限于经典分离逻辑.

直观上来说,分离逻辑公式 $P*Q$ 刻画了所有可以"分割"为两部分,其中一部分满足 P,另一部分满足 Q 的内存状态.拿分离逻辑最常用的基于栈(stack)和堆(heap)的语义模型来解释:内存状态(s,h)满足 $P*Q$,也就是说 $s,h\vDash P*Q$,当且仅当堆 h 可以被分割为定义域不相交的两部分 h_1,h_2,即 $h=h_1\cup h_2$,$dom(h_1)\cap dom(h_2)=\varnothing$,并且 $s,h_1\vDash P$,$s,h_2\vDash Q$.需要注意的是,堆是从地址(location)到值的映射,这里的分割指的是对堆的定义域的分割.分离逻辑公式 $P-\!\!-\!\!*Q$ 从直观上可以大概理解为从满足公式 Q 的内存区域里挖掉满足公式 P 的内存区域后剩下的部分.严格来说,内存状态(s,h)满足 $P-\!\!-\!\!*Q$,即 $s,h\vDash P-\!\!-\!\!*Q$,当且仅当任何一个满足公式 P 的内存状态(s,h_1)和(s,h)合并之后的状态都满足公式 Q,即$\forall h_1.s,h_1\vDash P$,我们都有 $s,h*h_1\vDash Q$.这里,$h*h_1$ 代表 $h\cup h_1$,但要求 $dom(h)\cap dom(h_1)=\varnothing$.

已有的基于分离逻辑的程序验证大都只涉及分离与算子,最主要的原因之一是使用分离逻辑进行逆向推理(backwards reasoning)比较复杂.对于不太熟悉分离逻辑的读者来说,如图 2 所示的例子[2,5]可以较为形象地展示分离与算子(*)和经典的逻辑与算子(∧)之间的区别.

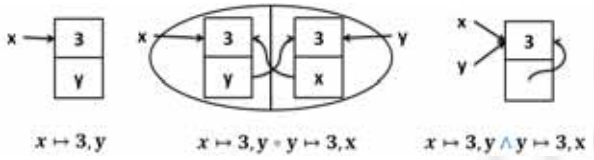


Fig.2 Examples for separation conjunction and classic conjunction

图 2 分离与和逻辑与示例

分离逻辑最大的优点之一就是可以很有效地支持局部推理,得益于该逻辑特有的框(frame)规则[4,5].

- Frame 规则: $\dfrac{\{P\}\ C\ \{Q\}}{\{P*R\}\ C\ \{Q*R\}}$ $(Modifies(C)\cap Free(R)=\varnothing)$.

这一规则使得人们在通过符号执行来分析验证局部程序时只需考虑该局部程序所访问的内存区域,而不需要把全局状态都带上.

2 基于分离逻辑的程序分析与验证

这一节我们着重介绍基于分离逻辑的对顺序程序进行分析与验证的一些典型工作.既有侧重于自动验证的,也有着眼于交互式验证的;既有侧重于对内存(指针)安全性的验证,也有同时着眼于内存安全性和功能正确性的验证工作.

2.1 用于内存(指针)安全性的自动程序验证

2.1.1 Smallfoot 等早期工作

Berdine 等人最早在 Smallfoot 项目里[6,7]将分离逻辑用于程序验证.不同于前面提到的验证条件生成方法,Smallfoot 自动程序验证采用符号执行(symbolic execution)和一套简单的蕴涵检查(entailment checking)规则交替进行的方法.表示抽象程序状态的符号公式在符号执行过程中就地得到更新,所以不需要复杂的别名检查过程.就地更新时,一般先要对公式的相关部分进行再排列(rearrangement),如果涉及到对抽象谓词所描述的数据结构,如树、链表段等进行访问或更新,Smallfoot 会先使用相应的展开规则将谓词展开.例如,链表段 $ls(E,F)$在 $E\neq F$ 的情况下会被展开为 $E\mapsto[n:x']*ls(x',F)$,然后再对变量 E 所对应的堆单元进行访问(包括更新).这里,x'是一个新的逻辑变量.Smallfoot 能够自动验证一些简单程序的轻量级安全性质,但需要用户提供循环不变式、前后置规范等.

为了减少对用户提供标注的依赖并提高验证系统的自动化程度,2006 年,Distefano 等人[8]和 Magill 等人[9]独立提出了将符号执行和基于抽象解释的不动点求解相结合的方法来自动推导循环不变式.这两项工作仅限于对(单)链表处理程序(list-processing programs)的循环不变式的推导.他们的规范描述语言对堆的抽象描述(即符号堆)仅使用堆单元 $E\mapsto F$ 和链表段谓词 $ls(E,F)$,其中,前者描述一个堆单元,后者描述从 E 到 F 的一个单链表段.他们所分析和验证的程序性质也仅局限于指针安全性(pointer safety,即程序不会间接引用空指针或悬空指针,不会导致内存泄漏).另外,他们提出的形状分析(shape analysis)也仅限于过程内分析.

这里,我们对基于抽象解释和符号执行的不动点求解过程进行简要介绍.给定一个循环程序 WHILE B DO C,其循环不变式的不动点求解过程本身也是一个循环算法.假设不动点求解算法的第 i 次循环开始时程序的抽象状态为 S_i,第 i 次循环从初始状态 $S_i\wedge B$ 出发调用符号执行规则来逐步执行循环体,并在循环体执行结束时得到状态 S_i',经过适当的抽象处理后,得到状态 S_i''.不动点求解算法会将这一抽象状态和第 i 次循环开始的抽象状态 S_i 进行比较:如果两者等价,算法停止并返回 S_i'' 为循环不变式;否则,算法会将 S_i'' 和 S_i 合并(join)起来得到 S_{i+1},然后进入第 i+1 次循环.Distefano 等人的局部形状分析采用了一种所谓的规范化符号堆(canonical symbolic heap)来描述抽象状态.这种规范化描述的特点是不保留任何辅助变量切点(cutpoint),举例来说,如果 x 是一个辅助逻辑变量(即分析过程中新引入的临时变量,不是程序指针变量或者堆单元的域名),那么 $P_1(E,x)*P_2(x,nil)$可以被抽象为 $ls(E,nil)$.这里,$P_i(E,F)$代表 $E\mapsto F$ 或者 $ls(E,F)$.对他们所使用的程序语言和规范描述语言来说,只有有限多个规范化符号堆存在.他们的算法使用的抽象机制保证了每次循环都从规范化符号堆开始,并且每次循环结束时都能得到规范化符号堆,因此,他们的不动点求解算法的终止性得到了保证.

2.1.2 SpaceInvader 和 Abductor 等工作

Yang 等人将前面所述的局部形状分析技术[8,9]做了进一步的精化和提高,实现了一个新的原型工具 SpaceInvader,并首次将其成功应用于对 1 万行以上的 Windows 和 Linux 设备驱动代码的自动验证[10].虽然这一工具所验证的性质非常简单,仅限于指针安全性,所处理的程序也仅限于链表操作程序,但却是世界上首次将自动分析与验证技术应用到工业级程序代码中.在这之前的自动程序分析与验证技术要么过于侧重于精确度(precision)而只能处理很小的程序,要么过于侧重可扩展性(scalability)而无法达到需验证性质要求的精确度.

SpaceInvader 在对堆的抽象描述方面与前述局部形状分析[8,9]的不同之处在于:它将链表段谓词($ls\ E\ F$)进一步划分为两个谓词($ls\ PE\ E\ F$)和($ls\ NE\ E\ F$),前者允许从 E 到 F 的链表段为空(即 $E=F$),后者要求从 E 到 F 的链表段必须非空(即至少包含 1 个堆单元).

$$ls\ PE\ E\ F \Leftrightarrow (E = F \wedge emp) \vee (ls\ NE\ E\ F),$$

$$ls\ NE\ E\ F \Leftrightarrow (E \mapsto F) \vee (\exists F'.E \mapsto F' * ls\ NE\ F'\ F).$$

在此基础上,SpaceInvader 提出了一个新的合并(join)操作[10],既保持了该自动分析的可扩展性,也达到了验证指针安全性所需的精确度.SpaceInvader 的合并操作对符号堆$\Pi_0\wedge\Sigma_0$和$\Pi_1\wedge\Sigma_1$的合并分两步进行:第 1 步对堆部分Σ_0和Σ_1进行抽象合并,例如,合并$\Sigma_0\equiv(ls\ NE\ x\ nil*y\mapsto nil)$和$\Sigma_1\equiv(x\mapsto x'*ls\ NE\ y\ x'*ls\ NE\ x'\ nil)$后得到$\Sigma\cong(ls\ NE\ x\ v'*ls\ NE\ y\ v'*ls\ PE\ v'\ nil)$,并同时生成证据三元组集合$\{(nil,x',v')\}$;第 2 步是处理栈变量的近似抽象,尽可能地保

留有关指针变量和逻辑变量间的关系(包括别名关系、非别名关系、非空性).对这个具体例子而言,第 1 步对Σ_1进行抽象时,将 x'的非空性丢失,第 2 步会将 $x'\neq nil$ 这一信息保留下来.虽然这里的 x'有可能只是一个辅助逻辑变量,在分析过程中保留这样的信息是有必要的,因为符号堆状态里可能同时有类似于($y=x'$)这样的别名关系,因此,第 2 步保留的信息($x'\neq nil$)就表明了指针变量 y 是非空的.

SpaceInvader 分析处理的对象限于单链表数据结构,因此在分析验证操作系统驱动程序的指针安全性时需要做一些简化,例如,先将双链表修改成单链表(忽略回指针).和很多其他工具一样,SpaceInvader 的另外一个局限点是无法对数组,特别是指针数组做自动分析和验证.尽管如此,Yang 等人的工作[10]还是将基于分离逻辑的自动程序分析与验证向工业界实际应用的方向推进了非常大的一步.

在 SpaceInvader 工作的基础上,Calcagno 等人提出了基于双向诱导(bi-abduction)技术的可组合形状分析系统 Abductor[11,12],将指针安全性的自动分析与验证又向前推进了一大步.Abductor 的最大优点之一就是可组合性,即对程序的每个过程(procedure)可以单独分析.对整个程序的分析与验证,根据过程调用关系自底向上地分解成对每个过程的单独分析与验证.Abductor 系统的最大创新点在于,它首次将人工智能中的诱导(abduction)技术[13]成功用到基于分离逻辑的程序分析与验证中.诱导推理为程序的前置条件的推导提供了很好的技术支持,而双向诱导推理则既可以推算前置条件,也可用于后置条件的推导.

基于分离逻辑的诱导推理用于解决如下逻辑推理问题:$A*[??]\vdash G$.这里,A 和 G 是两个已知的分离逻辑公式(用于表示程序的抽象状态),诱导推理的结果是满足逻辑蕴含 $A*[AF]\vdash G$ 的一个分离逻辑公式 AF.举一个简单的例子,诱导推理 $emp*[??]\vdash x\mapsto nil$ 返回的结果可以是 $x\mapsto nil$.双向诱导推理解决的是如下逻辑问题:

$$A*[?anti\text{-}frame]\vdash G*[?frame],$$

即从给定的 A 和 G 出发,推导出反框架(?anti-frame)和框架(?frame).例如,双向诱导 $x\mapsto nil*z\mapsto nil*[?anti\text{-}frame]\vdash list(x)*list(y)*[?frame]$可能推导出如下结果:$?anti\text{-}frame=list(y)$,$?frame=z\mapsto nil$.这里需要说明的是:(双向)诱导推理问题可能存在很多答案,Abductor 系统给出的答案不一定是最优解,但必须是比较容易计算得到的较好的结果[11,12].

在对没有给定任何前后置条件的一个程序过程进行分析时,Abductor 会从空堆状态 emp 出发,对过程体逐步进行符号执行.一旦当前抽象程序状态无法满足下一条程序指令所需要的前置条件,Abductor 就使用诱导推理来诱导出缺失的部分,并将其回传到程序过程的起始处.然后,以分离组合的方式将其添加到原有的前置条件中得到新的前置条件,并重新开始对过程体进行符号执行.等到整个过程体的符号执行完全结束时,也就得到了该过程的一个可能的前后置规范.Abductor 使用的符号执行机制和 SpaceInvader 一样,由于可能用到抽象近似,计算出的前置规范有可能是不可靠的.因此,Abductor 还会对推导出的前后置规范做进一步的验证,排除掉可能不可靠的那些前后置规范对.

和 SpaceInvader 工作一样,Abductor 分析处理的对象限于单链表数据结构.因此,在分析验证操作系统驱动程序的指针安全性时需要做一些简化,不能较好地处理数组和指针偏移等.Abductor 在做跨过程(inter-procedural)程序分析时是自底向上进行的,即先分析和验证被调用过程(callee),再分析验证调用过程(caller).在这方面,Abductor 的一个不足之处是无法很好地处理未知库函数,只是简单地将其替换为一个非确定的赋值操作.Luo 等人提出了一套自顶向下的分析方法,用于对未知过程和未知代码段的前后置规范的自动推导[14,15].

O'Hearn 等人在 2011 年~2012 年将基于 SpaceInvader/Abductor 方面的工作商业化,成功地孵化出代码公司 Monodics.该公司于 2013 年被 Facebook 收购,相应的代码验证工具 Infer 已被用于分析和验证 Facebook 移动应用代码的内存安全性.该工具可供人们下载使用(http://fbinfer.com).

2.2 用于包含内存安全性在内的更复杂性质的自动程序验证

目前已经介绍的相关工作通过抽象描述单链表的形状性质来分析和验证操作单链表程序的指针安全性.而如果要验证一般堆处理程序的内存安全性和更复杂的功能正确性,通常还需要在规范描述语言和程序分析的抽象域里添加有关堆上数据结构的其他性质.这一节我们着重介绍这方面的相关工作.

2.2.1 SLAyer 和 THOR 等带有简单数值信息的分析与验证工作

Berdine 等人在微软剑桥研究院时开发的 SLAyer 系统[16]被用于系统代码的内存安全性分析.SLAyer 使用与 Distefano 等人的局部形状分析[8]同样的分析框架,但将其扩展到跨过程分析.另外,SLAyer 系统也允许将单链表的长度信息标注到链表段谓词里,例如,$ls^k(p,q)$描述的是长度为 k 的单链表段.这样的数值信息可以使得单链表的规范描述更精确.

Magill 等人提出的 THOR 系统[17,18]拟通过对数据结构的一些数值方面的性质(如链表长度)的刻画,来增强对堆处理程序的推理与验证.THOR 的核心思路是将基于分离逻辑的形状分析和传统数值程序的循环不变式分析相结合.THOR 在形状分析阶段将待分析的堆处理程序转为一个数值程序,然后在下一阶段,使用传统的循环不变式推导算法对获得的数值程序做进一步的分析.与前述工作相比,THOR 的形状分析可以处理双链表,而且由于引入了额外的数值分析步骤,可以提供比 SpaceInvader 更高的精度,可以处理链表处理程序的一些需要额外数值信息才能验证的程序性质.

2.2.2 支持包含内存安全性在内的更复杂性质的 HIP/SLEEK 程序验证系统

SpaceInvader 等系统虽然被成功用于 1 万行以上代码的验证,但其最大的局限性在于只能处理链表操作程序的指针安全性,无法处理一般堆处理程序的内存安全性和功能正确性.对内存安全性进行验证时,仅描述数据结构形状方面的性质是不够的,例如,红黑树节点的红、黑特性可能与指针是否为空相关联,AVL 树左右子树的平衡性质也有可能与程序的内存安全性相关联.另外,SpaceInvader 等系统只能处理少量事先定义好的形状谓词(如 ls 等),而且对这些谓词的推理规则必须提前编制到其系统中.Chin 等人提出了一套强大的基于分离逻辑的规范描述机制与自动验证方法,并研发出 HIP/SLEEK 验证系统,很好地克服了这些局限性[19,20].

HIP/SLEEK 的特点之一在于其提供的表达力丰富的规范描述机制,允许用户自定义用于描述数据结构性质的递归谓词,而且支持对动态创建数据结构更复杂特性的描述,例如形状、尺度和内容等.举例来说,下面的用户自定义谓词描述了单链表的形状和长度信息.

$$ll(x,n)::=(x=nil\wedge n=0)\vee\exists v,p.(x\mapsto node(v,p)*ll(p,n-1)).$$

这里的 $node$ 是单链表节点的定义:$data\ node\{int\ val;\ node\ next;\}$.

下面的两个谓词都描述了排好序的非空链表.

$$sortl(x,\min,n)::=(x\mapsto node(\min,nil)\wedge n=1)\vee$$
$$\exists k,p.(x\mapsto node(\min,p)*sortl(p,k,n-1)\wedge \min\le k),$$
$$slB(x,\min,B)::=(x\mapsto node(\min,nil)\wedge B=\{x\})\vee$$
$$\exists k,p,B'.(x\mapsto node(\min,p)*slB(p,k,B')\wedge \min\le k\wedge B=B'\cup\{x\}).$$

除了链表的形状特性和链表节点储存的最小值以外,第 1 个谓词还刻画了链表长度,而第 2 个谓词则使用多重集变量刻画了链表节点存储的所有值.

下面的谓词则刻画了 AVL 树的节点数信息(n)和高度信息(h).

$$avl(x,n,h)::=(x=nil\wedge n=0\wedge h=0)\vee\exists v,l,r,nl,hl,nr,hr.$$
$$(x\mapsto treenode(v,l,r)*avl(l,nl,hl)*avl(r,nr,hr)\wedge\quad.$$
$$n=1+nl+nr\wedge h=1+\max(hl,hr)\wedge -1\le hl-hr\le 1).$$

这里的数据结构 $treenode$ 定义的是二叉树的节点.

$$data\ treenode\{int\ val;treenode\ left;treenode\ right;\}.$$

这套极具表达力的用户自定义谓词描述方法给使用者提供了一套非常灵活的机制.用户可以根据所需验证性质、所需处理的程序的具体情况来决定合理的抽象层次,并定义合适的递归谓词.而用户所定义的谓词一经 HIP/SLEEK 检验合法,将和 HIP/SLEEK 已有库里的谓词一样,可被用来描述程序的前后置规范.

如图 3 所示,HIP/SLEEK 程序验证系统包括前端的 HIP 程序验证器和后端的 SLEEK 分离逻辑证明器.给定一个待验证的 Hoare 规范$\{P\}\ C\ \{Q\}$(用户自定义递归谓词可能会出现在程序 C 的前后置规范 P 和 Q 中),HIP 程序验证器从 P 出发,调用相应的前向验证规则对 C 进行验证.对验证过程中产生的验证条件,HIP 会调用

SLEEK 分离逻辑验证器来进行自动证明.SLEEK 证明器支持对用户自定义谓词的处理,并会自动调用有关递归谓词的引理(包括由用户提供的经 SLEEK 证明过的引理).HIP 程序验证具有模块化的特点,即对拟验证程序的每个过程逐一进行验证.对一个程序过程的验证,HIP 会从其前置条件(前置规范)出发,前向验证过程体,最后调用 SLEEK 来证明验证过程体所生成的后状态蕴含该程序过程的给定后置条件(后置规范).HIP 对程序过程的验证采用的是自底向上的方法,即先验证 callee,再验证 caller(相互递归的过程会放到一起作为一个强连通体(strongly connected component)一起处理).因此,在验证一个过程调用语句时,只需调用 SLEEK 来证明当前程序抽象状态满足该过程的前置条件即可,如果成功,则可以认为该过程调用结束处其后置条件自动成立.

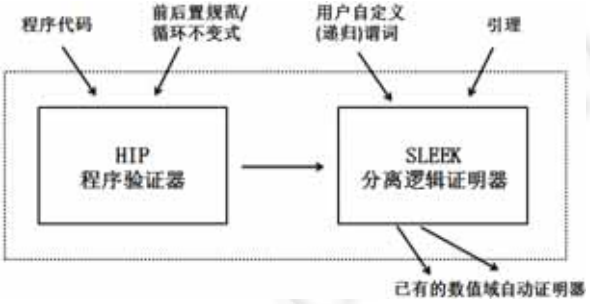


Fig.3 HIP/SLEEK: Automated program verification via separation logic

图 3 HIP/SLEEK 基于分离逻辑的程序自动验证

HIP 规范描述语言支持用户自定义谓词、逻辑析取,也支持对数据结构形状、尺度和内容等不同抽象域信息的刻画,具有很强的表达力.用 HIP 规范描述语言表达的公式可以转换成如下标准型:$\Phi=\vee(\exists v^*.\kappa\wedge\pi)^*$,其中,$\kappa$是由空堆、堆单元节点、或者用户自定义谓词实例经分离合取操作生成的分离逻辑公式;π刻画了其他数值域上的信息,既包括对所描述数据结构尺度信息(如链表长度、节点个数等)的刻画,也包括对数据结构内容(如链表、树里存储的内容、节点地址信息等)的描述.SLEEK 证明器可用来证明 HIP 规范描述语言描述的分离逻辑公式间的蕴含关系.简单来说,给定 HIP 公式Δ_A,Δ_C,SLEEK 处理蕴含关系$\Delta_A\vdash\Delta_C$*??,即判断Δ_A是否蕴含Δ_C:如果是,则给出Δ_F,使得$\Delta_A\vdash\Delta_C*\Delta_F$.在使用相应规则处理完逻辑析取和量词后,SLEEK 证明的基本思想是:使用相应的规则将Δ_A和Δ_C里的单元(具体堆单元或者谓词实例)进行匹配,直到Δ_C变为一个经典逻辑公式.在这个过程中,需要对用户自定义谓词进行适当的处理,例如,根据其定义将其展开,或者将一个具体公式根据谓词定义合并成一个谓词.在Δ_C变为纯数值域上的公式后,SLEEK 会将Δ_A和匹配积累的公式近似成一个纯数值域上的公式,然后调用数值域上的已有证明器(如 Omega Calculator,Isabelle,MONA,CVC Lite,Z3 等)来自动完成余下的证明.

HIP/SLEEK(如图 3 所示)的另一个强大特点是对用户自定义引理的支持[21].前面提到的相关研究工作只支持对有限几个预定义的堆谓词进行推理,并且,在对特定程序进行分析验证时,大多只能处理单个堆谓词.由于对用户自定义谓词和用户自定义引理的支持,HIP/SLEEK 可以用来分析验证很多原来的工作无法处理的程序.HIP/SLEEK 支持的引理可以描述公式间的弱化、增强和等价等关系,其既可以用来描述不同谓词之间的关系,如

$$\forall v.(sortl(root,v,n)\rightarrow ll(root,n)\wedge n>0),$$

也可以用来描述同一个堆谓词之间的关系,如

$$\forall a,b.(lseg(root,p,n)\wedge n=a+b\wedge a,b\geq 0\rightarrow\exists r.lseg(root,r,a)*lseg(r,p,b)).$$

这里的 $lseg(root,p,n)$谓词描述的是从 $root$ 到 p 的长度为 n 的链表段.

$$lseg(root,p,n)::=root=p\wedge n=0\vee\exists r.root\mapsto node(\_,r)*lseg(r,p,n-1).$$

这一引理可以用来证明类似下面的蕴含关系(对链表段任意切割,而不是仅限于分割头节点或者尾节点):

$$lseg(x,p,n)\wedge n=8\vdash\exists r.lseg(x,r,a)*lseg(r,p,b)\wedge a=2\wedge b=6*\Phi_R.$$

这些引理支持对全称量词变量的实例化,因而极大地增强了 SLEEK 证明器的表达能力.用户自定义引理会先经

由 SLEEK 证明器进行证明,可靠的引理才会被 SLEEK 接受.SLEEK 在程序验证过程中会自动调用已知可靠的引理来完成新的蕴含关系的证明.

HIP/SLEEK 的规范描述语言还支持使用多对前后置条件来描述程序过程的规范[22].这样的规范描述机制一方面可以更具精确性(每对前后置条件只需刻画特定的过程调用场景),另一方面,也使得程序过程的规范描述更加全面.HIP/SLEEK 使用抽象状态集合(而不是单一抽象状态)一次性同时处理针对多对前后置规范的验证.为了提供更精确的规范描述,并让规范描述更好地引导程序验证过程,HIP/SLEEK 也支持更为结构化的规范描述[23,24].相对于通常的平坦结构的规范描述,结构化的规范描述使得用户对验证过程可以有更多的控制,例如,更多的案例分析可用来避免在验证过程中出现太多的逻辑或操作,更早的变量实例化可以有效减少存在量词的使用,使用阶段式公式(staged formulae)描述的规范可以支持对验证过程更多的重用.

与大多数程序验证系统一样,对程序过程进行自动验证时,HIP/SLEEK 需要用户提供前后置条件、循环不变式等(如图 3 所示).Qin 等人[25,26]对基于 HIP/SLEEK 复合抽象域上的循环不变式的自动推导进行了深入研究,提出了基于用户自定义谓词的循环不变式推导方法,可有效减少对用户提供循环不变式的依赖.Chang 等人[27]也提出了关系式的递归分析方法,用来推导循环不变式,但该方法在分析过程中不保留任何逻辑变量的共享切点(shared cutpoints),而 Qin 等人的方法[25,26]则尽可能地保留这些共享切点,因此分析结果可以更加精确.另外,Chang 等人的方法在分析时只能处理单个形状谓词,而 Qin 等人的分析则允许多种谓词同时出现在对同一个程序的分析中,因此可以处理一些 Chang 等人工作不能处理的程序[26].

在另外一些工作[28–30]中,Qin 等人尝试对程序过程的前后置规范进行自动推导,以减少对用户提供规范的依赖,并进一步提升 HIP/SLEEK 的自动化验证程度.他们尝试了一种半自动的方法[28,29],即在用户提供程序过程前后置条件所需堆谓词模板的前提下,使用静态分析方法来自动推算出缺失的数值域上的信息.这种方法的优点是:尽量利用用户对程序的了解,让用户提供前后置条件里的堆描述部分的模板,然后使用静态分析来处理剩下的比较琐碎的部分.由于不需要对堆形状部分做太多的推导和诱导推理,这种情况下的静态分析效率会比较高.Qin 等人也尝试了完全自动的方法[30],即不需要用户为前后置条件提供任何模板,而是直接构建包含形状和数值信息的复合抽象域上的静态分析方法来推导程序过程在复合抽象域上的前后置规范.这项工作的一大新颖之处是使用了一种叫诱导抽象(abductive abstraction)的机制,即将诱导推理和抽象机制结合起来,可以推导出更精确的前后置规范.这种方法的不足之处在于其抽象机制在使用时需要相应的用户自定义谓词做向导,因此分析结果一定程度上取决于用户自定义谓词的质量.Loc 等人[31]则提出了一套更高级的形状分析方法,可以避免对用户自定义谓词的依赖.在这一工作上,Loc 等人提出了一套非常新颖的二阶诱导方法,通过引入未知谓词变量,自动推算出很难由用户自己提供的复杂谓词定义,以及使用这样的谓词来描述的程序过程的前后置规范.Le 等人的工作[32]则将使用未知谓词变量的方法应用到终止性和非终止性规范的自动推导中.Qin 等人对 HIP/SLEEK 环境下循环不变式和前后置条件的推导工作做了总结[33].He 等人将 HIP/SLEEK 成功应用于程序内存使用行为的验证[34].Ferreira 等人尝试着用 HIP/SLEEK 对实时操作系统内核 FreeRTOS 的调度子系统进行验证[35].

2.3 基于分离逻辑的面向对象程序验证

面向对象程序验证的相关工作非常多,这里我们简要介绍几项使用分离逻辑对面向对象程序进行验证的工作.Chin 等人和 Parkinson 等人分别研究了将分离逻辑用于面向对象程序验证的问题,并各自独立提出了非常相似的两套方案,相关研究结果最终被同一个顶级会议(ACM POPL08)同时收录[36,37].两项工作异曲同工之处是,将面向对象程序的类方法的规范描述区分为静态规范和动态规范两种.静态方法规范可以很精确地描述单个方法的方法体的行为规范,可适用于静态调派的方法调用,如超类调用(super call)、直接调用(direct call)等;动态规范则主要是为了处理一些面向对象特性(如行为子类型、类继承、方法覆盖等),适用于动态调派的方法,因此并不一定能够精确刻画该方法方法体的行为.Chin 等人[36]基于分离逻辑的框架(frame)规则提出了一个增强版的规范包含规则,用于定义方法规范间的“子类型”关系.有了这个定义,在规范描述和验证时,就可以要求每个方法的静态规范为对应动态规范的子类型.他们也提出了一种基于分离逻辑的可扩展对象表示方式和相应

的部分/全局观察,可以很灵活地支持向上和向下强制转换的处理.他们在对面向对象程序进行验证时,会要求每个新定义的方法都有一个相应的静态规范,系统会验证每种方法的代码满足其静态规范;动态规范则既可以由用户给定,也可以由系统自动生成.通过引入几类方法规范间的子类型关系,其验证过程可以大量减少(由于添加、修改代码而引起的)不必要的重新验证.他们的工作的实现基于 HIP/SLEEK 系统,而 Parkinson 和 Bierman 的方案[37]主要基于他们之前提出的一种非常强大的抽象谓词家族机制[38],因此后者更具一般性.Luo 等人[39]则着重研究了多重继承机制下的分离逻辑的设计问题.

Distefano 等人则成功地将分离逻辑用于一些典型 Java 程序的验证[40].其基本思想是,将 Parkinson 和 Bierman 的抽象谓词家族机制[37,38]和 Distefano 等人的基于抽象解释的符号执行和证明机制结合起来[8].他们构建了自动验证工具 jStar,该工具包含两个核心部分:一个针对面向对象程序的分离逻辑定理证明器,一种适用于面向对象程序验证的符号执行和抽象技术.他们将 jStar 成功地用于验证多个面向对象设计范式(design pattern).另一个基于分离逻辑的验证工具 VeriFast[41]支持对命令式程序(如 C 程序)和面向对象程序(如 Java 程序)的验证.Qiu 等人[42,43]则着重研究了用于面向对象程序里的接口(interface)的基于分离逻辑的规范描述与验证问题.

3 基于并发分离逻辑和高阶分离逻辑的程序验证

这一节简要介绍分离逻辑方面更前沿、更深入的一些研究内容.我们首先介绍并发分离逻辑和基于并发抽象谓词的验证思想,然后简要介绍基于高阶并发抽象谓词的一些高级程序逻辑,包括用于弱内存模型(weak memory model)上程序验证的并发程序逻辑.

3.1 O'Hearn的并发分离逻辑CSL

2007 年,O'Hearn 提出了并发分离逻辑(concurrent separation logic,简称 CSL)[44],与此同时,Brookes 则对 CSL 做了详尽的理论分析,给出了 CSL 的一个基于迹模型(trace model)的指称语义[45].Brookes 和 O'Hearn 由于对并行分离逻辑的开创性贡献而获得了 2016 年哥德尔奖(2016 Gödel Prize)[46].这里,我们着重介绍 CSL 的主要想法和核心推理规则.对 CSL 形式化语义感兴趣的读者可以自行阅读 Brookes 的指称语义模型[45].

O'Hearn 将并发程序分为谨慎型(cautious)程序和大胆型(daring)程序两种,前者要求所有并发进程对同一共享状态的访问必须限于同一互斥组里,后者则允许这样的访问跨越多个互斥组.例如,图 4 左边的程序是谨慎型的,因为对地址为 10 的堆单元的所有访问都在由信号量 *free* 控制的同一个互斥组内;右边的程序是大胆型的,因为对地址为 10 的堆单元的访问跨越两个互斥组(分别由信号量 *free* 和 *busy* 控制).

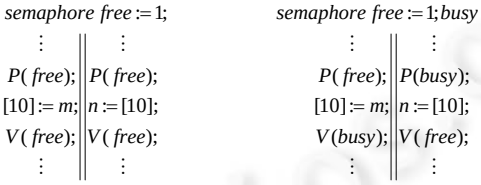


Fig.4 Examples for cautious program and daring program

图 4 谨慎型程序和大胆型程序示例

CSL 的基本想法围绕着两个想法来实现,即拥有权(ownership)假设和分离性(separation).拥有权假设指的是任何代码段只能访问它自己拥有的状态部分.分离性指的是,在任何时候,程序状态可以被分割为由每个进程和每个互斥组各自拥有的部分.这种分割可以是动态变化的,即随着程序的执行,状态拥有权的分割可以动态改变.

为灵活起见,O'Hearn[44]使用一种带条件临界区(conditional critical regions,简称 CCR)的程序语言来描述 CSL 的推理规则,着重考虑了如下带初始化和资源定义的特殊形式程序.

$$\begin{array}{c} init; \\ resource\ r_1(variable\ list),...,resource\ r_m(variable\ list) \\ C_1\|...\|C_n. \end{array}$$

条件临界区语句,如 with r then B do C endwith 则描述一个互斥组,即代码 C 只有在当前进程拥有资源 r 并且条件 B 为真的情况下才能执行.CSL 对变量做了一些语法上的限制:每个变量只能属于最多一个资源;如果某个变量 x 属于某个资源 r,那么该变量不能同时出现在并发进程中,除非是在有关 r 的临界区内;如果一个变量在某个进程中被改变,那么该变量不允许出现在其他进程中,除非该变量隶属于某个资源.

CSL 为上述带初始化和资源定义的程序定义了如下推理规则:

$$\frac{\{P\}\ init\{RI_{r_1}*...*RI_{r_m}*P'\}\quad \{P'\}C_1\|...\|C_n\{Q\}}{\{P\}\left(\begin{array}{c} init; \\ resource\ r_1(variable\ list),...,resource\ r_m(variable\ list) \\ C_1\|...\|C_n \end{array}\right)\{RI_{r_1}*...*RI_{r_m}*Q\}}.$$

从给定的前置条件 P 出发,初始化程序分别建立每个资源的不变式 $RI_{r_1},...,RI_{r_m}$.从剩下的状态部分 P'出发,并行复合 $C_1\|\ldots\|C_n$ 在执行结束后须满足所需的后置条件 Q.并行复合的推理规则比较简单.

$$\frac{\{P_1\}C_1\{Q_1\}\quad ...\quad \{P_n\}C_n\{Q_n\}}{\{P_1*...*P_n\}C_1\|...\|C_n\{Q_1*...*Q_n\}}$$ (对任何 $i\neq j$,C_j 不修改 P_i 或者 Q_i 里的自由变量).

这条规则与一般的不相交并发(disjoint concurrency)规则非常类似,区别在于,这条规则的应用环境里有一些资源不变式,这些资源不变式在验证 $C_1,\ldots,C_n$ 里的条件临界区语句时会发挥作用.条件临界区语句的验证规则如下:

$$\frac{\{(P*RI_r)\wedge B\}\ C\ \{Q*RI_r\}}{\{P\}\ \text{with}\ r\ \text{when}\ B\ \text{do}\ C\ \text{endwith}\ \{Q\}}$$ (没有任何其他进程会修改 P 或者 Q 里的自由变量).

这里的想法是:代码 C 在开始执行时即拥有资源 r 所保护的状态(资源不变式 RI_r),执行过程中可以访问这些共享状态,并且在执行结束时保证这个资源不变式仍然成立.这条规则涉及到资源保护的共享状态拥有权的迁移.

3.2 分离逻辑与依赖/保证等方法相结合的技术

在分离逻辑出现之前,并发程序逻辑推理方面出现较早、人所熟知的工作是 Jones 的依赖/保证(rely/guarantee)方法[47].依赖/保证方法可以很好地刻画并发进程间的(对共享内存访问的)冲突(interference),但不足之处在于缺少可组合性(compositionality).Vafeiadis 和 Parkinson 也是在 2007 年提出了 RGSep[48],一套很好地结合了依赖/保证方法和分离逻辑各自优点的并发程序验证框架.RGSep 推理方法极具一般性,可以看作是分离逻辑和 CSL 的一种扩展.Feng 等人在同一时期提出了一套将假设/保证(assume/guarantee)和分离逻辑相结合的方法 SAGL[49],主要用于汇编代码的推理.

Calcagno 等人把 RGSep 思想和 Smallfoot 验证系统结合起来,构建出 SmallfootRG 原型系统[50],用于对并发的单链表处理程序内存安全性的验证.传统的依赖/保证方法只处理并行组合,即所有并发进程同时开始,同时结束.Dodds 等人对这一局限性进行了改进,在结合分离逻辑和资源权限[51,52]的基础上提出了拒绝/保证推理方法(deny/guarantee reasoning)[53],可有效处理动态的并发程序结构(如 fork,join 等).Feng[54]首次将依赖/保证推理和信息隐藏(information hiding)结合起来,对 RGSep 和 SAGL 进行了改进,提出了一套局部依赖/保证推理方法.该方法首次提出了依赖/保证条件上的框架规则,进一步提高了依赖/保证推理的模块化程度,同时,也可以更灵活地共享动态创建堆对象等资源.

3.3 基于并发抽象谓词CAP和高阶并发抽象谓词的并发程序验证

Dinsdale-Young 等人[55]构建了一种能够较好地支持模块化验证的程序逻辑 CAP(concurrent abstract predicate,并发抽象谓词).与传统的抽象谓词仅提供粗粒度推理不同,并发抽象谓词 CAP 借助资源权限控制[53]为(共享)并发数据结构提供细粒度的抽象,进而支持对并发程序模块的细粒度推理.并发抽象谓词可以用来刻

画堆上共享数据结构其中一部分所需满足的性质,以及程序模块对该部分所允许的改动.CAP 验证框架的基本想法如下:定义好的并发抽象谓词用来描述并发数据结构模块的高层规范;在 CAP 验证了模块的实现确实满足其高层规范之后,对调用该模块的客户端程序进行验证时,可以直接使用该模块的高层规范,而不需要再考虑该模块的实现.这样,CAP 所定义的并发抽象谓词为并发程序的可组合/模块化验证提供了必要的抽象.

Dinsdale-Young 等人[56]提出了有关并发程序推理原理的一种元理论(metatheory),称为并发视野框架(concurrent views framework).这个元理论以具有组合性质的状态抽象(称为视野(view))为参数,通过对这个参数做相应的实例化,很多并发推理方法,如依赖/保证方法、CSL、CAP、针对递归引用和针对唯一指针(unique pointer)的类型系统等,都可以变成并发视野框架的具体实例.Dinsdale-Young 等人还发现:这些具体系统的可靠性都可以在并发视野框架下得到证明,而无需回到各方法下对操作语义做归纳证明.例如,Calcagno 等人提出的分离代数(separation algebra)[57]其实是并发视野框架的一个实例,即分离视野幺半群(separation view monoid).这个理论框架对下面提到的一些高阶程序逻辑的出现有一定的推动.这项工作首次提出的视野转换(view shift)的概念非常具有启发性,后续一些高阶程序逻辑大都采用类似的概念.

受 Biering 等人提出的高阶分离逻辑[58]的启发,Dodds 等人[59]对 CAP 逻辑进行了扩展,引入了高阶参数和高阶量词(即取值可能是一阶谓词).他们使用扩展后的并发抽象量词的高阶变体来描述一类确定性并行程序的高层次抽象规范,并对其进行验证.Dodds 等人的高阶并发抽象量词允许使用嵌套的区域断言(nested region assertion)和(用于刻画对共享状态的访问行为的)高阶协议.由于对这些机制没有任何限制,可能导致高阶谓词的不稳定性问题,也导致该高阶逻辑变体在某些边缘情形下的推理变得不可靠(详见文献[60]的扩展版).

Svendsen 等人发现了这个潜在的不可靠性(unsoundness)问题,并提出了一种新的高阶并发抽象谓词 HOCAP(higher-order CAP)[60],支持线程-安全的并发数据结构的规范描述.由于对堆上资源进行抽象描述的谓词可以作为并发抽象谓词的参数,与 CAP 相比,HOCAP 可以更多地支持资源共享,同时支持客户程序对数据结构模块的使用协议规范进行精化,并对数据结构模块添加对额外资源的拥有权.HOCAP 的核心机制包括嵌套区域断言、状态-独立的高阶协议以及卫式递归断言.与 Dodds 等人提出的高阶变体相比,HOCAP 在自引区域断言(self-referential region assertion)等方面做了些限制,仅允许直谓式(predicative)协议描述,使得文献[59]中高阶谓词可能不稳定的情形不会出现,也因此保证了 HOCAP 推理的可靠性.

在描述有些程序的规范时(如同时涉及数据结构内共享和跨数据结构共享的情况),HOCAP 为了保证逻辑可靠性而引入的一些限制,使其变得非常复杂.Svendsen 等人为此对 HOCAP 做了进一步的改进,提出了一种表达力更强大的高阶并发谓词机制 iCAP(impredicative CAP)[61],引入了非直谓的(impredictive)协议描述,可以更好地支持对同时具有数据结构内部共享和外部共享的程序(如并发、高阶、可重入(re-entrant)命令式程序)的模块化验证.

Jung 等人提出了高阶并发分离逻辑的一个一般性理论框架 Iris[62].Iris 使用两种核心机制:幺半群(monoids)和不变式(invariant).部分可交换幺半群用来表达对共享状态进行访问的(用户自定义)协议,而不变式机制则用来实施这些由幺半群描述的协议.通过对视野转换[56]进行扩展,Iris 可以有效支持逻辑意义上的原子规范(logically atomic specification).Iris 逻辑框架既有对物理状态的描述,也有对幽灵状态(ghost state,或称为逻辑状态)的刻画.其中,部分交换幺半群用来描述幽灵状态,而幽灵状态间的变迁则通过一种扩展版的视野转换操作来实现.Iris 为并发分离逻辑提供了一个统一的基础:在 Iris 里构建不同的部分交换幺半群、不同的不变式,能够得到不同的并发分离逻辑.因此,很多其他程序逻辑(如 iCAP[61],CaReSL[63],TaDA[64]等)都是 Iris 逻辑框架的具体实例.

3.4 弱内存模型上的并发程序的验证逻辑

现代硬件结构都支持不同程度的弱内存模型,而 Java[65]和 C/C++11[66]都相继提出了对弱内存模型的语言层面上的支持.因此,如何分析和验证弱内存模型上的并发程序,也是一个更新、更具挑战性的问题.这里,我们简要介绍一些与分离逻辑相关的工作.

Vafeiadis 等人[67]提出了一种松弛型分离逻辑 RSL(relaxed separation logic),这是第 1 个支持 C11 弱内存模

型的并发程序逻辑.RSL 主要考虑了如下几种原子操作:顺序一致 sc(sequentially consistent)、释放 rel (release)、捕获 acq(acquire),还有非原子操作 na(non-atomic).其中,与 C11 最相关的是 rel 和 acq 原子操作.根据 C11 语义,线程(*A*)的一个释放写操作(rel_w)和另外一个线程(*B*)的一个捕获读操作(acq_r)可能会形成同步(synchronized-with)关系.当这种同步关系发生时,先于发生(happens-before)的因果关系就会形成:位于线程 *A* 里 rel_w 前的那些操作的效果就会被位于线程 *B* 里 acq_r 后的那些操作观察到.从程序逻辑上讲,线程 *A* 里的一些资源的拥有权就会转移到线程 *B*.为了对这种拥有权进行建模,RSL 引入了几种新的逻辑谓词,包括 $Rel(l,Q)$和 $Acq(l,Q)$,其中,$Rel(l,Q(v))$代表将值 v 写入位置 l 所需的权限,前提是已拥有资源谓词 $Q(v)$.RSL 释放写操作的推理规则如下:

$$\{Q(v)*Rel(l,Q)\}[l]_{rel}:=v\{Init(l)*Rel(l,Q)\} \quad \text{(W-REL)}$$

其中,$Init(l)$表示位置 l 已经被初始化,而 $Acq(l,Q)$谓词则代表了对位置 l 进行捕获读所需的权限.RSL 捕获读操作的推理规则被表示为

$$\frac{\forall x.precise(Q(x))}{\{Init(l)*Acq(l,Q)\}[l]_{acq}\{v.Q(v)*Acq(l,Q[v:=emp])\}} \quad \text{(R-ACQ)}$$

从这两个例子可以大致看出,RSL 需要非常巧妙定义的权限谓词来完成拥有权的转让.因此,RSL 的实际使用不太容易被一般用户理解和掌握.

受同一时期并发程序验证方面的先进技术的启发,Turon 等人随后提出了用于验证包含释放写-捕获读操作的 C11 子集的程序逻辑 GPS[68].简要来说,GPS 逻辑借鉴了程序验证三大新技术,即拥有权和分离、协议、幽灵状态,并将这些技术巧妙地改进与融合,以用于弱内存模型上的程序验证.由于弱内存模型下没有对单个全局内存的假设,CSL 等的拥有权转移机制和分离性并不能直接照搬过来,因此,GPS 必须引入额外的托管(escrow)机制来实现拥有权的转让.由于弱内存模型支持内存操作的调序(re-ordering),程序执行的效果并不一定按照程序代码的顺序进行,使用前述程序逻辑的单个协议来描述整个内存区域的方法也不可行,因此,GPS 提出了单个位置的协议(per-location protocol).GPS 也定义了额外的幽灵体(ghosts)来刻画一些逻辑资源(如权限等).GPS 验证系统可以处理释放写-捕获读的同步.

He 等人[69]则对 GPS 逻辑做了进一步扩展,得到了一个更强大的程序逻辑 GPS+,可用来验证更大一集 C11 程序,包括栅栏操作(fence)和松弛原子操作(relaxed atomic).由于需要处理更复杂的 C11 原子操作,GPS+的语义模型和可靠性证明比 GPS 要复杂很多.同一时期,Doko 等人[70]则对 RSL 逻辑进行扩展,得到了一个新的逻辑 FSL(fenced separation logic),可以处理栅栏操作等.虽然文献[69,70]采用了非常类似的语义模型,但这两项工作是各自独立完成的.

4 总结与展望

从分离逻辑于本世纪初首次被提出到本文写作完成不过 10 多年的时间,但有关分离逻辑的研究,特别是基于分离逻辑的程序分析与验证的研究,却在这期间得到了蓬勃的发展,取得了很多成果.其中,顺序程序的分析与验证方面的成果最有实际应用成效.本文尝试着对基于分离逻辑的程序验证的主要研究工作做了阐述,介绍了分离逻辑用于堆处理(顺序)程序的分析与验证方面的典型工作,也简要介绍了近些年提出的用于并发程序(包括弱内存模型上的并发程序)的验证的高级逻辑.这些近期工作大多用到高阶逻辑和一些通常需人工提供的很深入的机制,如共享状态使用协议、幽灵状态等,因此,离最终实际应用还有很长的距离.我们认为,这方面研究未来的重点和难点还是在并发程序的分析与验证上,例如,如何尽可能地减少用户提供辅助机制(如协议、标注、幽灵状态等),如何进一步提高验证自动化程度等.本文尝试着涵盖尽可能多的相关工作,但由于时间和篇幅关系,遗漏之处在所难免.此外,虽然部分工作有较为详细的介绍,大部分工作只能有简要的描述.

References:

[1] Hoare CAR. An axiomatic basis for computer programming. Communications of the ACM, 1969,12(10):576−580. [doi: 10.1145/363235.363259]

[2] Reynolds JC. Separation logic: A logic for shared mutable data structures. In: Proc. of the 17th IEEE Symp. on Logic in Computer Science (LICS). IEEE, 2002. 55−74. [doi: 10.1109/LICS.2002.1029817]

[3] Ishtiaq S, O'Hearn PW. BI as an assertion language for mutable data structures. In: Proc. of the 28th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL). ACM Press, 2001. 14−26. [doi: 10.1145/360204.375719]

[4] O'Hearn PW, Reynolds J, Yang H. Local reasoning about programs that alter data structures. In: Proc. of the 15th Annual Conf. of the European Association for Computer Science Logic. 2001. 1−19. [doi: 10.1007/3-540-44802-0_1]

[5] O'Hearn PW. A primer on separation logic. In: Proc. of the Software Safety and Security: Tools for Analysis and Verification, Vol.33 of NATO Science for Peace and Security Series. 2012. 286−318. [doi: 10.3233/978-1-61499-028-4-286]

[6] Berdine J, Calcagno C, O'Hearn PW. Symbolic execution with separation logic. In: Proc. of the 3rd Asian Symp. on Programming Languages and Systems (APLAS). LNCS 3780, 2005. 52−68. [doi: 10.1007/11575467_5]

[7] Berdine J, Calcagno C, O'Hearn PW. Smallfoot: Modular automatic assertion checking with separation logic. In: Proc. of the 5th Int'l Symp. on Formal Methods for Components and Objects (FMCO). LNCS 4111, 2006. [doi: 10.1007/11804192_6]

[8] Distefano D, O'Hearn PW, Yang H. A local shape analysis based on separation logic. In: Proc. of the 12th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. LNCS 3920, 2006. 287−302. [doi: 10.1007/11691372_19]

[9] Magill S, Nanevsky A, Clarke E, Lee P. Inferring invariants in separation logic for imperative list-processing algorithms. In: Proc. of the 3rd Workshop on Semantics, Program Analysis, and Computing Environments for Memory Management (SPACE). 2006.

[10] Yang H, Lee O, Berdine J, Calcagno C, Cook B, Distefano D, O'Hearn PW. Scalable shape analysis for system code. In: Proc. of the 20th Int'l Conf. on Computer Aided Verification (CAV). 2008. 385−398. [doi: 10.1007/978-3-540-70545-1_36]

[11] Calcagno C, Distefano D, O'Hearn PW, Yang H. Compositional shape analysis by means of bi-abduction. In: Proc. of the 36th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL 2009). New York: ACM Press, 2009. 289−300. [doi: 10.1145/1480881.1480917]

[12] Calcagno C, Distefano D, O'Hearn PW, Yang H. Compositional shape analysis by means of bi-abduction. Jounral of the ACM, 2011,58(6):1−66. [doi: 10.1145/2049697.2049700]

[13] Kakas AC, Kowalski RA, Toni F. Abductive logic programming. Journal of Logic and Computation, 1992,2(6):719−770. [doi: 10.1093/logcom/2.6.719]

[14] Luo C, Craciun F, Qin S, He G, Chin WN. Verifying pointer safety for programs with unknown calls. Journal of Symbolic Computation, 2010,45(11):1163−1183. [doi: 10.1016/j.jsc.2010.06.003]

[15] Qin S, Luo C, He G, Craciun F, Chin WN. Verifying heap-manipulating programs with unknown procedure calls. In: Proc. of the Formal Methods and Software Engineering (ICFEM 2010). LNCS, Shanghai: Springer-Verlag, 2010. [doi: 10.1007/978-3-642-16901-4_13]

[16] Berdine J, Cook B, Ishtiaq S. SLAyer: Memory safety for systems-level code. In: Proc. of the 23rd Int'l Conf. on Computer-Aided Verification (CAV). 2011. 178−183. [doi: 10.1007/978-3-642-22110-1_15]

[17] Magill S, Berdine J, Clarke E, Cook B. Arithmetic strengthening for shape analysis. In: Proc. of the Static Analysis Symp. 2007. 419−436. [doi: 10.1007/978-3-540-74061-2_26]

[18] Magill S, Tsai MH, Lee P, Tsay YK. THOR: A tool for reasoning about shape and arithmetic (tool paper). In: Proc. of the 20th Int'l Conf. on Computer-Aided Verification (CAV). 2008. 428−432. [doi: 10.1007/978-3-540-70545-1_41]

[19] Nguyen HH, David C, Qin S, Chin WN. Automated verification of shape and size properties via separation logic. In: Proc. of the VMCAI 2007. LNCS 4349, Nice: Springer-Verlag, 2007. [doi: 10.1007/978-3-540-69738-1_18]

[20] Chin WN, David C, Nguyen HH, Qin S. Automated verification of shape, size and bag properties via user-defined predicates in separation logic. Science of Computer Programming, 2012,77(9):1006−1036. [doi: 10.1016/j.scico.2010.07.004]

[21] Nguyen HH, Chin WN. Enhancing program verification with lemmas. In: Proc. of the Computer Aided Verifications (CAV 2008). LNCS 5123, Princeton, 2008. 355−369. [doi: 10.1007/978-3-540-70545-1_34]

[22] Chin WN, David C, Nguyen HH, Qin S. Multiple pre/post specifications for heap-manipulating methods. In: Proc. of the 10th IEEE High Assurance Systems Engineering Symp. (HASE 2007). Dallas: IEEE CS Press, 2007. [doi: 10.1109/HASE.2007.56]

[23] Gherghina C, David C, Qin S, Chin WN. Structured specifications for better verification of heap-manipulating programs. In: Proc. of the Formal Methods (FM 2011). Limerick: Springer-Verlag, 2011. [doi: 10.1007/978-3-642-21437-0_29]

[24] Gherghina C, David C, Qin S, Chin WN. Expressive program verification via structured specifications. Int'l Journal on Software Tools for Technology Transfer, 2014,16(4):363−380. [doi: 10.1007/s10009-014-0306-5]

[25] Qin S, He G, Luo C, Chin WN. Loop invariant synthesis in a combined domain. In: Proc. of the Formal Methods and Software Engineering (ICFEM 2010). LNCS, Shanghai: Springer-Verlag, 2010. [doi: 10.1007/978-3-642-16901-4_31]

[26] Qin S, He G, Luo C, Chin WN, Chen X. Loop invariant synthesis in a combined abstract domain. Journal of Symbolic Computation, 2013,50(3):386−408. [doi: 10.1016/j.jsc.2012.08.007]

[27] Chang BYE, Rival X. Relational inductive shape analysis. In: Proc. of the 35th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages (POPL). 2008. 247−260. [doi: 10.1145/1328438.1328469]

[28] Qin S, Luo C, Chin WN, He G. Automatically refining partial specifications for program verification. In: Proc. of the Formal Methods (FM 2011). LNCS 6664, Limerick: Springer-Verlag, 2011. 369−385. [doi: 10.1007/978-3-642-21437-0_28]

[29] Qin S, He G, Luo C, Chin WN, Yang H. Automatically refining partial specifications for heap-manipulating programs. Science of Computer Programming, 2014,82(2):56−76. [doi: 10.1016/j.scico.2013.03.004]

[30] He G, Qin S, Chin WN, Craciun F. Automated specification discovery via user-defined predicates. In: Proc. of the 15th Int'l Conf. on Formal Engineering Methods (ICFEM 2013). LNCS 8144, Queenstown, 2013. 398−415. [doi: 10.1007/978-3-642-41202-8_26]

[31] Le QL, Gherghina C, Qin S, Chin WN. Shape analysis via second-order bi-abduction. In: Proc. of the 26th Int'l Conf. on Computer Aided Verification (CAV 2014). LNCS 8559, Vienna, 2014. 52−68.

[32] Le TC, Qin S, Chin WN. Termination and non-termination specification inference. In: Proc. of the 36th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI 2015). Portland: ACM Press, 2015. 489−498. [doi: 10.1145/2813885.2737993]

[33] Qin S, He G, Chin WN, Yang H. Invariants synthesis over a combined domain for automated program verification. In: Proc. of the Theories of Programming and Formal Methods. LNCS 8051, 2013. 304−325. [doi: 10.1007/978-3-642-39698-4_19]

[34] He G, Qin S, Luo C, Chin WN. Memory usage verification using Hip/Sleek. In: Proc. of the 7th Int'l Symp. on Automated Technology for Verification and Analysis. LNCS 5799, Macao: Springer-Verlag, 2009. 166−181. [doi: 10.1007/978-3-642-04761-9_14]

[35] Ferreira J, Gherghina C, He G, Qin S, Chin WN. Automated verification of the FreeRTOS scheduler in HIP/SLEEK. Int'l Journal on Software Tools for Technology Transfer, 2014,16(4):381−397. [doi: 10.1007/s10009-014-0307-4]

[36] Chin WN, David C, Nguyen HH, Qin S. Enhancing modular OO verification with separation logic. In: Proc. of the 35th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL 2008). San Francisco: ACM Press, 2008. 87−99. [doi: 10. 1145/1328897.1328452]

[37] Parkinson M, Bierman G. Separation logic, abstraction and inheritance. In: Proc. of the 35th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL 2008). San Francisco: ACM Press, 2008. 75−86. [doi: 10.1145/1328438.1328451]

[38] Parkinson M, Bierman G. Separation logic and abstraction. In: Proc. of the 32nd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL 2005). 2005. 247−258. [doi: 10.1145/1040305.1040326]

[39] Luo C, Qin S. Separation logic for multiple inheritance. In: Proc. of the 1st Int'l Conf. on Foundations of Informatics, Computing and Software (FICS 2008), Vol.212. Shanghai: Electronic Notes in Theoretical Computer Science, 2008. 27−40. [doi: 10.1016/j.entcs.2008.04.051]

[40] Distefano D, Parkinson M. jStar: Towards practical verification for Java. In: Proc. of the 23rd Annual ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2008). 2008. 213−226. [doi: 10.1145/1449764.1449782]

[41] Jacobs B, Smans J, Philippaerts P, Vogels F, Penninckx W, Piessens F. VeriFast: A powerful, sound, predictable, fast verifier for C and Java. In: Proc. of the NASA Formal Methods (NFM). 2011. 41−55. [doi: 10.1007/978-3-642-20398-5_4]

[42] Qiu Z, Hong A. Modular verification of OO programs with interfaces. In: Proc. of the 14th Int'l Conf. on Formal Engineering Methods (ICFEM 2012). LNCS 7504, Kyoto: Springer-Verlag, 2012. 151−166. [doi: 10.1007/978-3-642-34281-3_13]

[43] Hong A, Liu Y, Qiu Z. Axioms and abstract predicates on interfaces in specifying/verifying OO components. In: Proc. of the 10th Int'l Symp. on Formal Aspects of Component Software. LNCS 8348, Nanchang: Springer-Verlag, 2013. 174−195. [doi: 10.1007/978-3-319-07602-7_12]

[44] O'Hearn PW. Resources, concurrency, and local reasoning. Theoretical Computer Science, 2007,375(1-3):271−307. [doi: 10.1016/j.tcs.2006.12.035]

[45] Brookes S. A semantics for concurrent separation logic. Theoretical Computer Science, 2007,375(1-3):227−270. [doi: 10.1016/j.tcs.2006.12.034]

[46] Brookes S, O'Hearn PW. Concurrent separation logic. ACM SIGLOG News, 2016,3(3):47−65. [doi: 10.1145/2984450.2984457]

[47] Jones CB. Specification and design of (parallel) programs. In: Proc. of the IFIP Congress. 1983. 321−332.

[48] Vafeiadis V, Parkinson M. A marriage of reply/guarantee and separation logic. In: Proc. of the 18th Int'l Conf. on Concurrency Theory (CONCUR 2007). LNCS 4703, 2007. 256−271. [doi: 10.1007/978-3-540-74407-8_18]

[49] Feng X, Ferreira R, Shao Z. On the relationship between concurrent separation logic and assume-guarantee reasoning. In: Proc. of the European Symp. of Programming. 2007. 173−188. [doi: 10.1007/978-3-540-71316-6_13]

[50] Calcagno C, Parkinson M, Vafeiadis V. Modular safety checking for fine-grained concurrency. In: Proc. of the 14th Static Analysis Symp. (SAS 2007). LNCS 4634, 2007. 233−248. [doi: 10.1007/978-3-540-74061-2_15]

[51] Boyland J. Checking interference with fractional permissions. In: Proc. of the 10th Static Analysis Symp. (SAS 2007). LNCS 2694, Heidelberg: Springer-Verlag, 2003. 55−72. [doi: 10.1007/3-540-44898-5_4]

[52] Bornat R, Calcagno C, O'Hearn PW, Parkinson M. Permission accountingin separation logic. In: Proc. of the 32nd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL 2005). New York: ACM Press, 2005. 259−270. [doi: 10.1145/1040305.1040327]

[53] Dodds M, Feng X, Parkinson M, Vafeiadis V. Deny-Guarantee reasoning. In: Proc. of the 18th European Symp. on Programming (ESOP 2009). LNCS 5502, 2009. 363−377. [doi: 10.1007/978-3-642-00590-9_26]

[54] Feng X. Local rely-guarantee reasoning. In: Proc. of the 36th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL 2009). 2009. 315−327.

[55] Dinsdale-Young T, Dodds M, Gardner P, Parkinson M, Vafeiadis V. Concurrent abstract predicates. In: Proc. of the 24th European Conf. on Object-Oriented Programming (ECOOP 2010). 2010. 504−528. [doi: 10.1007/978-3-642-14107-2_24]

[56] Dinsdale-Young T, Birkedal L, Gardner P, Parkinson M, Yang H. Views: Compositional reasoning for concurrent programs. In: Proc. of the 40th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL 2013). 2013. 287−299. [doi: 10.1145/2429069.2429104]

[57] Calcagno C, O'Hearn PW, Yang H. Local action and abstract separation logic. In: Proc. of the 22nd IEEE Symp. on Logic in Computer Science (LICS 2007). 2007. 366−378. [doi: 10.1109/LICS.2007.30]

[58] Biering B, Birkedal L, Torp-Smith N. BI-Hyperdoctrines, higher-order separation logic, and abstraction. ACM Trans. on Programming Languages and Systems, 2007,29(5):Article No.24. [doi: 10.1145/1275497.1275499]

[59] Dodds M, Jagannathan S, Parkinson M. Modular reasoning for deterministic parallelism. In: Proc. of the 38th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL 2011). 2011. 259−270. [doi: 10.1145/1925844.1926416]

[60] Svendsen K, Birkedal L, Parkinson M. Modular reasoning about separation of concurrent data structures. In: Proc. of the 22nd European Symp. on Programming (ESOP 2013). 2013. 169−188. [doi: 10.1007/978-3-642-37036-6_11]

[61] Svendsen K, Birkedal L. Impredicative concurrent abstract predicates. In: Proc. of the 23rd European Symp. on Programming (ESOP 2014). 2014. 149−168. [doi: 10.1007/978-3-642-54833-8_9]

[62] Jung R, Swasey D, Sieczkowski F, Svendsen K, Turon A, Birkedal L, Dreyer D. Iris: Monoids and invariants as an orthogonal basis for concurrent reasoning. In: Proc. of the 42nd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL 2015). 2015. 637−650. [doi: 10.1145/2676726.2676980]

[63] Turon A, Dreyer D, Birkedal L. Unifying refinement and Hoare-Style reasoning in a logic for higher-order concurrency. In: Proc. of the Int'l Conf. on Functional Programming (ICFP). 2013. 377−390. [doi: 10.1145/2544174.2500600]

[64] da Rocha Pinto P, Dinsdale-Young T, Gardner P. TaDA: A logic for time and data abstraction. In: Proc. of the 28th European Conf. on Object-Oriented Programming (ECOOP 2014). 2014. 207−231. [doi: 10.1007/978-3-662-44202-9_9]

[65] Manson J, Pugh W, Adve SV. The Java memory model. In: Proc. of the 32nd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL 2005). 2005. 378−391. [doi: 10.1145/1040305.1040336]

[66] Batty M, Owens S, Sarkar S, Sewell P, Weber T. Mathematizing C++ concurrency. In: Proc. of the 38th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL 2011). 2011. 55−66. [doi: 10.1145/1926385.1926394]

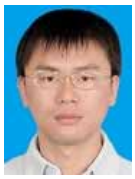
[67] Vafeiadis V, Narayan C. Relaxed separation logic: A program logic for C11 concurrency. In: Proc. of the ACM Int'l Conf. on Object Oriented Programming Systems Languages & Applications (OOSPLA 2013). 2013. 867−884. [doi: 10.1145/2509136.2509532]

[68] Turon A, Vafeiadis V, Dreyer D. GPS: Navigating weak memory with ghosts, protocols, and separation. In: Proc. of the ACM Int'l Conf. on Object Oriented Programming Systems Languages & Applications (OOSPLA 2014). 2014. 691−707. [doi: 10.1145/2660193.2660243]

[69] He M, Vafeiadis V, Qin S, Ferreira JF. Reasoning about fenes and relaxed atomics. In: Proc. of the 24th Euromicro Int'l Conf. on Parallel, Distributed, and Network-Based Processing (PDP 2016). 2016. 520−527. [doi: 10.1109/PDP.2016.103]

[70] Doko M, Vafeiadis V. A program logic for C11 memory fences. In: Proc. of the 17th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI 2016). 2016. 413−430. [doi: 10.1007/978-3-662-49122-5_20]

秦胜潮(1974－),男,湖北红安人,博士,教授,博士生导师,主要研究领域为软件形式化方法和模型,程序理论与程序逻辑,程序分析与验证.



许智武(1983－),男,博士,助理教授,CCF 专业会员,主要研究领域为程序分析与验证,程序理论与程序逻辑,类型系统.

明仲(1967－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,云计算,中间件技术.