ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.8, August 2009, pp.2051−2061
doi: 10.3724/SP.J.1001.2009.03636

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

基于分离逻辑的程序验证技术*

黄达明[1,2], 曾庆凯[1,2+]

[1](南京大学 计算机软件新技术国家重点实验室,江苏 南京 210093)

[2](南京大学 计算机科学与技术系,江苏 南京 210093)

Program Verification Techniques Based on Separation Logic

HUANG Da-Ming[1,2], ZENG Qing-Kai[1,2+]

[1](State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China)

[2](Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

+ Corresponding author: E-mail: zqk@nju.edu.cn

Huang DM, Zeng QK. Program verification techniques based on separation logic. *Journal of Software*, 2009, 20(8):2051−2061. http://www.jos.org.cn/1000-9825/3636.htm

Abstract: This paper introduces the verification theory of separation logic, characteristics of separation logic, and some successful applications of separation logic. Researches on separation logic to support program verification are analyzed, including the properties of separation logic, its relation to other logics, its support to programming languages and design patterns, and the theorem provers' support to separation logic. The problems encountered when separation logic is applied more widely are pointed out, and the future research directions are discussed.

Key words: trusted software; program verification; Hoare logic; separation logic; theorem proving

摘 要: 介绍了分离逻辑的验证原理和特点及其在程序验证方面的应用实例,分析了为支持程序验证的若干分离逻辑研究进展,包括分离逻辑的自身属性、与其他逻辑的关系、对程序语言和设计模式的支持以及定理证明器等内容.指出了分离逻辑进一步深入应用所面临的问题和解决方向.

关键词: 可信软件;程序验证;霍尔逻辑;分离逻辑;定理证明

中图法分类号: TP311 **文献标识码**: A

可信计算正成为信息安全领域的一个新潮流[1−3].TCG(trusted computing group)对可信的定义是:一个实体在实现预定目标时,若其行为总是符合预期,则该实体是可信的[2].一个软件若是可信的,则必然是正确的,能够如预期那样工作.因此,程序正确性验证是可信软件的基本要求.程序正确性验证通常采用测试、模型检验、逻辑推理验证以及类型推导和抽象解释等其他静态方法.测试、类型推导等方法主要用以发现错误;模型检验可以证明程序属性的正确性,能够发现错误,但是受限于状态爆炸问题;逻辑推理验证方法通过建立一定的逻辑系

* Supported by the National Natural Science Foundation of China under Grant Nos.60773170, 60721002, 90818022 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2006AA01Z432 (国家高技术研究发展计划(863)); the Specialized Research Fund for the Doctoral Program of Higher Education of China under Grant No.200802840002 (高等学校博士学科点专项科研基金)

Received 2008-09-29; Accepted 2009-04-10

统将程序验证问题转换为逻辑推理问题,比较而言,能够更为全面地验证程序属性.

在命令式程序验证方面,基于经典逻辑的霍尔逻辑(Hoare logic)[4]得到了广泛的应用.但是,对使用指针的命令式语言程序进行推理验证是困难的.其中一个典型问题是对包含共享可操作数据结构程序的验证.所谓共享是指在多个位置对同一数据进行引用和修改,例如指针造成的别名.由于经典逻辑没有表达非别名的原语,在程序分析时必须考虑所有的别名模式;而当牵涉到分配和释放操作时,最弱前置条件和最强后置条件等计算的复杂度为程序变量数的指数级[5].此外,由于任意两个指针变量无论作用域如何都有可能是别名的,因此程序分析必须考虑到全局上下文.

分离逻辑[6−9]是对霍尔逻辑的一个扩展,通过提供表达显式分离的逻辑连接词以及相应的推导规则,消除了共享的可能,能够以自然的方式来描述计算过程中内存的属性和相关操作,从而简化了对指针程序的验证工作.同时,分离逻辑能够支持局部推理,可以将推理的注意力仅仅集中到所要证明程序片段的覆盖区(即这个程序片段所访问或修改的局部内存),然后再通过相关的推导规则扩展到全局状态.而且,分离逻辑被证明具有更强的验证能力,如对并发程序和资源管理的验证,使得程序验证和推理技术前进了一大步.因此,继霍尔逻辑之后,分离逻辑有望成为程序可信验证的一种重要方法.

本文介绍分离逻辑的基本思想、描述和推导规则,分析分离逻辑的研究进展及其应用,总结基于分离逻辑进行程序验证所面临的问题和解决方向.

1 分离逻辑

1.1 分离逻辑的基本思想

分离逻辑[6−9]是霍尔逻辑的一种扩展.霍尔逻辑[4]是广泛应用的程序验证逻辑系统,用于对命令式语言程序进行推理验证.其基本思想是:在代码段及其调用者之间构建一种合同似的规格说明,由一个前置条件和一个后置条件构成.前置条件是一个断言,描述这个代码段执行前程序状态必须满足的条件;后置条件也是一个断言,描述在代码段正确运行后程序状态所需要满足的条件,调用者可以确信在代码段执行结束后这个状态条件为真.

分离逻辑对程序的推理仍然是采用霍尔三元组$\{P\}C\{Q\}$**,其中,P表示前置条件,Q表示后置条件,而C表示代码段.在分离逻辑中,前置条件和后置条件中的程序状态主要由栈S和堆H构成,栈是变量到值的映射,而堆是有限的地址集合到值的映射.在程序验证时,可以将栈看作对寄存器内容的描述,而堆是对可寻址内存内容的描述[6].

$$S \stackrel{\text{def}}{=} Variables \to Values,\ H \stackrel{\text{def}}{=} Addresses \to_{fin} Values,\ States \stackrel{\text{def}}{=} S \times H.$$

分离逻辑对霍尔逻辑的最重要扩展,在于引入了两个新的分离逻辑连接词:分离合取*和分离蕴含−*,它们的语义形式化表示如下:

如果令 s 表示栈,h 表示堆,$h_0 \perp h_1$ 表示堆 h_0 和 h_1 不相交,$h_0 \cdot h_1$ 表示堆 h_0 和 h_1 的联合,则

$$[P * Q]\ s\ h \stackrel{\text{def}}{=} \exists h_0\ h_1.\ h_0 \perp h_1 \text{ and } h_0 \cdot h_1 = h \text{ and } P\ s\ h_0 \text{ and } Q\ s\ h_1.$$

$[P*Q]\ s\ h$ 断言整个堆 h 被分成两个不相交的部分 h_0 和 h_1,并且对子堆 h_0 断言 P 成立,而对子堆 h_1 断言 Q 成立.

$$[P -\!\!* Q]\ s\ h \stackrel{\text{def}}{=} \forall h'.(h' \perp h \text{ and } P\ s\ h') \text{ implies } Q\ s\ (h \cdot h').$$

分离蕴含表示,如果当前堆 h 通过一个分离的部分 h'扩展,并且对 h'断言 P 成立,则对扩展后的堆$(h \cdot h')$断言 Q 成立.

** 形如$\{P\}C\{Q\}$的三元组描述的是程序的部分正确性,形如$[P]C[Q]$的三元组表示的是程序的完全正确性.对程序的完全正确性证明通常分两步,首先证明程序的部分正确性,然后再证明程序的可终止性.

分离逻辑对局部推理的支持主要是通过引入 Frame 规则来实现的:

$$\frac{\{P\}\,C\,\{Q\}}{\{P*R\}\,C\,\{Q*R\}},$$

其中,代码段 C 不会对断言 R 中的自由变量赋值.

这里,断言 P 和 Q 是针对代码段 C 所牵涉到的栈变量和堆空间(称为 C 的覆盖区),因此是一个局部推理;R 代表的是 C 不会影响到的栈变量和堆空间,因此,通过 Frame 规则,可以把推理的注意力集中到当前代码段相关的局部状态空间,然后再扩展到整个状态空间.

1.2 分离逻辑的断言语言、规格说明语言及语义

1.2.1 断言语言

与霍尔逻辑一致,分离逻辑使用断言表示各种规格说明和约束,但是它提供 4 种描述堆的新形式,从而比霍尔逻辑中使用的谓词逻辑更强(在对分离逻辑的描述中,表达式和普通命令式语言中的表达式具有相同的含义):

⟨assert⟩::=…

	emp	空堆
|	⟨exp1⟩$\mapsto$⟨exp2⟩	仅包含 1 个单元的堆,其地址是 exp1,内容为 exp2, exp1 和 exp2 是表达式,并且不依赖于堆,不会产生副作用
|	⟨assert⟩*⟨assert⟩	分离合取
|	⟨assert⟩−*⟨assert⟩	分离蕴含

分离逻辑还提供一些有用的复杂形式的断言缩写:

$$e\mapsto -\overset{\text{def}}{=}\exists x.\,e\mapsto x,\text{其中,}x\text{在表达式}e\text{中不自由出现.}$$

$$e\not\mapsto\hookrightarrow|e'\overset{\text{def}}{=}e\mapsto e'*\text{true}.$$

$$e\mapsto e_1,...,e_n\overset{\text{def}}{=}e\mapsto e_1,...,e+n-1\mapsto e_n.$$

$$e\hookrightarrow|e_1,...,e_n\overset{\text{def}}{=}e\hookrightarrow|e_1*...*e+n-1\hookrightarrow|e_n\ \text{iff}\ e\mapsto e_1,...,e_n*\text{true}.$$

上面的断言中的 true 表示除了要刻画的堆单元外以,堆还可能包含其他单元,但在推理时并不关注这些单元.

通过使用$\mapsto$和$\hookrightarrow|$以及两种形式的合取,可以容易且精确地描述数据的共享模式、对数据的操作以及一些数据结构.

例如,可以通过结构化归纳定义谓词来刻画如图 1 所示的单向链表.

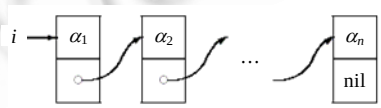

Fig.1 Singly-Linked list

图 1 单向链表

谓词 list a i 可以通过在序列 a 上进行结构化归纳来定义:

$$\text{list}\ \varepsilon\ i\overset{\text{def}}{=}\text{emp}\wedge i=\text{nil}.$$

$$\text{list}\ (a\cdot\alpha)\ i\overset{\text{def}}{=}\exists j.\,i\mapsto a,j*\text{list}\ \alpha\ j.$$

1.2.2 规格说明语言

除了用于表示状态的断言语言以外,分离逻辑还包括用于程序规格说明的语言,仍然采用霍尔三元组的

形式:

⟨spec⟩:={{⟨assert⟩}⟨command⟩{⟨assert⟩}} 部分正确语义

|[⟨assert⟩]⟨command⟩[⟨assert⟩] 完全正确语义

其中,对数据的操作命令主要有 4 种,即分配、查询、修改和释放.其形式化语义如下(局部推理表示在前,全局推理表示在后):

分配将当前堆扩展一个堆单元并赋予一个值:

$$\overline{\{\text{emp}\}v := cons(e')\{v \mapsto e'\}}\ ,\ \overline{\{r\}v := cons(e')\{(v \mapsto e') * r\}}\ .$$

查询表示将规定地址的堆单元的值读取到指定的变量中:

$$\overline{\{v = v' \wedge (e \mapsto v'')\}v := [e]\{v = v'' \wedge (e \mapsto v'')\}}\ ,\ \overline{\{\exists v''.(e \mapsto v'') * (r / v' \mapsto v)\}v := [e]\{\exists v'.(e \mapsto v) * (r / v'' \mapsto v)\}}\ .$$

修改表示修改规定地址的堆单元的值:

$$\overline{\{e \mapsto -\}[e] := e'\{e \mapsto e'\}}\ ,\ \overline{\{(e \mapsto -) * r\}[e] := e'\{(e \mapsto e') * r\}}\ .$$

释放通过去掉当前堆中的一个单元从而可以忽略其值:

$$\overline{\{e \mapsto -\}\ \text{dispose}\ e\ \{\text{emp}\}}\ ,\ \overline{\{(e \mapsto -) * r\}\ \text{dispose}\ e\ \{r\}}\ .$$

1.3 分离逻辑的推导规则

由于对霍尔逻辑进行了扩展,分离逻辑引入了一些新的推导规则.这些规则主要是因离合取以及分离蕴含而引入的.

分离合取*具有结合性和传递性,同时,emp 是其恒等元素(neutral element).与断言语言相关的比较重要的推导规则如下:

$$\frac{p1 \Rightarrow p2 \quad q1 \Rightarrow q2}{p1 * q1 \Rightarrow p2 * q2}\ ,\ \frac{p1 * p2 \Rightarrow p3}{p1 \Rightarrow (p2 -\!\!* p3)}\ ,\ \frac{p1 \Rightarrow (p2 -\!\!* p3)}{p1 * p2 \Rightarrow p3}\ .$$

在程序推理和验证中起主要作用的是与规格说明相关的推导规则.在分离逻辑中,最重要的是 Frame 规则,而且霍尔逻辑中的大部分推导规则在分离逻辑中仍然是正确的.

Consequence规则 $$\frac{p' \Rightarrow p \quad \{p\}c\{q\} \quad q \Rightarrow q'}{\{p'\}c\{q'\}}\ ,$$

辅助变量消除规则 $$\frac{\{p\}c\{q\}}{\{\exists v.p\}c\{\exists v.q\}}\ ,$$

替换规则 $$\frac{\{p\}c\{q\}}{(\{p\}c\{q\})/v_1 \to e_1,...,v_n \to e_n}$$,其中,$v_1,...,v_n$在p,c,q中都是自由出现的.

但是,并非所有霍尔逻辑中的推导规则都成立,例如 Constancy 规则就不再成立:

$$\frac{\{p\}c\{q\}}{\{p \wedge r\}c\{q \wedge r\}}\ .$$

在分离逻辑中,可以通过 Frame 规则由局部推导规则得到相应的全局推导规则,这在前面提到的数据操作命令的局部推理表示和全局推理表示中已经可以看到.此外,还可以通过分离蕴含由局部推导规则得出反向推理的相关规则.例如,对于修改命令,其反向推导规则为

$$\overline{\{(e \mapsto -) * ((e \mapsto e') -\!\!* p)\}[e] := e'\{p\}}\ .$$

1.4 分离逻辑证明实例

下面演示如何使用分离逻辑对实现单向链表原地反转的程序段进行部分正确性证明[6].其程序段如下:

j=nil; while i≠nil do (k=[i+1]; [i+1]=j; j=i; i=k;).

开始时,链表由 i 指向;当程序段执行完成后,反转的链表由 j 指向.使用谓词 list α (i,j)代表第 1 个元素由 i 指向,最后一个元素指向 j 的链表片断α,则一个完整的链表应该形如 list α(i,nil).如果使用传统的霍尔逻辑,为了刻画在反转过程中由 i 和 j 指向的链表没有共享的部分,则 while 循环的循环不变式必须形如

$$\{(\exists\alpha,\beta.(\text{list}\ \alpha(i,\text{nil})\wedge\text{list}\ \beta(j,\text{nil})\wedge\alpha_0^+=\alpha^+\cdot\beta)\wedge(\forall k.\text{reach}(i,k)\wedge\text{reach}(j,k)\Rightarrow k=\text{nil})\},$$

其中,reach(*i*,*j*)是新引入的一个谓词,表示从 i 能够沿着单向链表到达 k.使用这种形式,断言的推理是很难扩展的.如果采用分离逻辑,循环不变式的表示就很简洁了:

$$\{\exists\alpha,\beta.(\text{list}\ \alpha(i,\text{nil})*\text{list}\ \beta(j,\text{nil})\wedge\alpha_0^+=\alpha^+\cdot\beta\}.$$

下面给出采用分离逻辑对链表原地反转程序进行证明的主要步骤.

$\{\text{list}\ \alpha_0(i,\text{nil})\}$

j:=nil;

$\{\exists\alpha,\beta.(\text{list}\ \alpha(i,\text{nil})*\text{list}\ \beta(j,\text{nil})\wedge j=\text{nil}\wedge\alpha_0^+=\alpha^+\cdot\beta\}$

while *i*≠nil do

(

$\{\exists\alpha,\beta.(\text{list}\ \alpha(i,\text{nil})*\text{list}\ \beta(j,\text{nil})\wedge\alpha_0^+=\alpha^+\cdot\beta\wedge i\neq\text{nil}\}$

$\{\exists a,\alpha,\beta.(\text{list}\ a\cdot\alpha(i,\text{nil})*\text{list}\ \beta(j,\text{nil})\wedge\alpha_0^+=(a\cdot\alpha)^+\cdot\beta\}$

$\{\exists a,\alpha,\beta,k.(i\mapsto a,k*\text{list}\ \alpha(k,\text{nil})*\text{list}\ \beta(j,\text{nil})\wedge\alpha_0^+=(a\cdot\alpha)^+\cdot\beta\}$

k:=[*i*+1];

$\{\exists a,\alpha,\beta.(i\mapsto a,k*\text{list}\ \alpha(k,\text{nil})*\text{list}\ \beta(j,\text{nil})\wedge\alpha_0^+=(a\cdot\alpha)^+\cdot\beta\}$

[*i*+1]:=*j*;

$\{\exists a,\alpha,\beta.(i\mapsto a,j*\text{list}\ \alpha(k,\text{nil})*\text{list}\ \beta(j,\text{nil})\wedge\alpha_0^+=(a\cdot\alpha)^+\cdot\beta\}$

$\{\exists a,\alpha,\beta.(\text{list}\ \alpha(k,\text{nil})*\text{list}\ a\cdot\beta(i,\text{nil})\wedge\alpha_0^+=\alpha^+\cdot a\cdot\beta\}$

$\{\exists\alpha,\beta.(\text{list}\ \alpha(k,\text{nil})*\text{list}\ \beta(i,\text{nil})\wedge\alpha_0^+=\alpha^+\cdot\beta\}$

j:=*i*; *i*:=*k*;

$\{\exists\alpha,\beta.(\text{list}\ \alpha(i,\text{nil})*\text{list}\ \beta(j,\text{nil})\wedge\alpha_0^+=\alpha^+\cdot\beta\}$

)

$\{\exists\alpha,\beta.(\text{list}\ \alpha(i,\text{nil})*\text{list}\ \beta(j,\text{nil})\wedge i=\text{nil}\wedge\alpha_0^+=\alpha^+\cdot\beta\}$

$\{\exists\beta.(\text{list}\ \beta(j,\text{nil})\wedge\alpha_0^+=\beta\}$

2 采用分离逻辑的验证

分离逻辑的提出源于对包含共享可操作数据结构的命令式程序进行验证的需要.近几年来,已经使用分离逻辑对一些系统和算法进行了源代码级的推理和验证[10−20].

2.1 系统推理验证

对 Topsy 教学操作系统的堆管理器进行的源代码级的验证[10]是分离逻辑在程序验证方面的一个典型的成功应用.通过使用分离逻辑和 Coq 定理证明器,证明了 Topsy 操作系统的内存隔离属性,并在验证过程中发现了堆管理器源代码中的若干问题.这里,给出在验证过程中使用分离逻辑描述内存单元的 Array 谓词和描述堆列表的 Heap-list 谓词:

Array 谓词用于刻画从地址 l 开始长度为 sz 的内存单元组

$$\text{Array}\ l\ sz\overset{\text{def}}{=}(sz=0\wedge\text{emp})\vee(sz>0\wedge(\exists e.(l\mapsto e)*(\text{Array}(l+1)(sz-1))).$$

使用 Array 谓词,可以定义堆列表 Heap-list 谓词

$$\begin{aligned}\text{Heap-list}\ x\overset{\text{def}}{=}&\exists st.(x\mapsto st,\text{nil})\vee\\&\exists next.(next\neq\text{nil})\wedge(x\mapsto\text{free},next)*(\text{Array}(x+2)(next-x-2))*(\text{Heap-list}\ next)\vee\\&\exists next.(next\neq\text{nil})\wedge(x\mapsto\text{allocated},next)*(\text{Array}(x+2)(next-x-2))*(\text{Heap-list}\ next).\end{aligned}$$

其中,第 1 个子句刻画了空列表(只包含尾部的头),第 2 个子句刻画了第 1 个块为空闲块的列表,而第 3 个子句刻

画了第 1 个块为已分配块的列表.

2.2 算法验证

经典逻辑难以处理的一些重要算法,也可以使用分离逻辑简洁地进行推理和验证.采用 Cheney 算法的复制垃圾收集程序是一个内存管理程序[11],曾被应用到一些编译系统中.其处理的数据可以是多种类型的,甚至包括带环的数据结构,不仅会更新数据,而且会移动数据.由于缺乏足够的证明手段,在过去很长时间内未能成功地形式化证明这个算法的正确性.利用扩展的分离逻辑,能够对采用 Cheney 算法的复制垃圾收集程序进行正确性验证,其主要方法包括:

1) 建立一个合适的存储模型,其中,分离逻辑的堆仅包含 *cons* 单元;
2) 加入针对有限集合和关系的断言,并扩展了迭代的分离合取,然后,以此为基础形成规格说明和推理所需要的断言语言;
3) 给出了相应的语义,可以处理 Cheney 算法中指向某个旧内存单元的指针和指向旧内存单元拷贝的指针之间的同质问题,还可以将内存地址分成不相交的部分,可以表达每个这种内存地址集合的特殊属性.

2.3 程序属性验证

分离逻辑还能对程序的某些特殊属性进行推理.

(1) 数据结构形状分析.数据形状分析为程序属性验证提供了良好的基础.利用分离逻辑在局部推理上的优势,以基于分离逻辑的符号化堆[12]为基础,可以对程序执行点上的数据结构的形状问题进行分析研究[13],从而验证程序的某些属性.数据结构形状问题不仅是指两个变量是否同名的浅层问题,而且主要研究如链表是否包含环等深层存储属性问题.虽然目前研究分析仅限于过程内部,在一些小程序上做实验,但是由于分离逻辑对局部推理和模块化证明的支持,文献[13]中的研究可成为过程间分析、并发分析以及大程序分析验证的基础.

(2) 数据隐藏属性验证.信息隐藏是模块化设计中的重要要求,一个模块的内部数据对其调用者应该是隐藏的.当程序中包含指针时,就难以追踪是否有指针指向一个模块的内部数据,从而很难验证信息隐藏属性.文献[14]利用分离逻辑对程序的模块间信息隐藏属性进行了推理验证,研究了模块和调用者之间的资源属主转移情况,给出了成功和失败的程序例子.值得一提的是,研究提出了精确谓词的概念,利用分离逻辑的精确谓词可以无二义性地刻画系统状态的某一部分.

(3) 程序终止性证明.程序终止性是很多程序的基本要求.Brotherston 等人基于环形证明方法和分离逻辑技术,提出了一种证明命令式程序将会终止的霍尔风格的证明系统[15].这个系统可以判断从程序中给定的位置和满足给定前置条件的状态出发时的程序终止性.其证明规则主要有两类:操纵前置条件的逻辑规则和刻画程序命令执行效果的符号化执行规则.逻辑前置条件使用归纳定义的谓词刻画堆的属性,这表达为分离逻辑公式;环形证明方法将归纳定义的谓词通过环形推导在可能的路径上展开,从而可以在证明中丢弃所有无限的路径.这种方法还可以避免显式地创建和使用秩函数(rank function).他们相信用于分离逻辑的 Smallfoot 断言检查工具[16]是实现这种方法的一种合适的候选平台.

2.4 并发程序验证

并发程序设计的基础是资源分离以及条件临界区等同步机制.一些并发程序证明方法适用于简单的共享内存程序,但是当程序共享状态中包含指针时,这些方法就不够了.一些研究利用分离逻辑来处理这种情况:

1) 基于资源不变式和资源所有权的推理方法[17].将整个共享状态看作命名资源的划分集合.在任何时刻,整个状态可以为每个进程划分成分离的部分,而每个分离部分都满足相关的资源不变式.这样,即使部分共享状态的所有权在不同进程间转移,共享状态中可能包含指针的并行程序,也可能进行安全的推理.不过,研究工作仅仅作了一些非形式化的正确性证明,没有给出所使用的分离逻辑的扩展的一个合适的语义模型.
2) 包含指针的并发程序中的语义模型[18].基于动作跟踪集合,为并发分离逻辑给出了一个指称语义.这个

语义使用并行组合,可以检测潜在的竞争条件,支持对部分正确性和竞争不存在性的组合推理.以此语义为基础,研究还证明了资源敏感的并发分离逻辑的可靠性.

3) 并发程序设计中的授权审计问题[19].研究给出了基于资源所有权转移和访问授权的逻辑方法来对并发程序进行推理,分析了授权转移、授权划分和授权组合等问题,并研究了授权审计问题,以防止资源泄漏.

4) 并发程序栈中内容的处理[20].前述研究都是针对堆中单元而言,文献[20]将栈和堆中所有的变量都看作资源,研究了如何更好地在并发程序中证明资源管理的相关属性,并使用这种方法完整地对并发程序中的临界区进行形式化处理.

3 分离逻辑研究

为使分离逻辑更好地支持程序验证,围绕分离逻辑开展了一系列研究,包括:1) 分离逻辑的逻辑属性研究.分析分离逻辑的表达验证能力,进行模型检验或者推理证明时的时空复杂度,是否具有可判定性、可满足性等.2) 与其他逻辑之间的关系研究.研究逻辑之间的互相转换性、表达能力及其优、缺点.3) 分离逻辑对编程模式、数据结构和算法支持的研究.研究如何使得分离逻辑能够支持程序设计的各种编程模式、数据结构和算法,为程序推理和验证提供坚实的基础.4) 定理证明器对分离逻辑支持的研究.研究如何使定理证明器能够更好地支持分离逻辑.

3.1 分离逻辑的逻辑属性

研究者主要对分离逻辑的一些子集的逻辑属性进行了研究.

文献[21]对使用分离逻辑的断言语言的一个子集进行了研究.这个子集包含描述数据形状的指向关系和等价关系的谓词,但是不包含描述数据内容本身的算术或者其他表达式之类的谓词.文献[21]对这个子语言的可计算性和复杂性等问题进行研究,以评估这种断言语言的能力.结论是:(1) 即使不使用分离合取*,emp 和分离蕴含−*等分离逻辑连接词,一个断言的合法性问题也不是递归可枚举的;(2) 如果不包括限定词,则使用分离逻辑这个子集的断言的有效性是算法上可判定的;(3) 当包含不同的分离逻辑连接词时,子集语言进行模型检验时的计算复杂度 MC(model checking problem)以及公式的永真性判断问题复杂度 VAL(validity problem)是不一样的,具体可见表 1.

Table 1 Computability and complexity results for a subset of the assertion language of separation logic

表 1 分离逻辑断言语言的某个子集的可计算性和复杂性结果

	Assertion language	MC	VAL
L	$P::=(E\mapsto E,E)\mid(E\not\mapsto-)\mid E=E\mid E\neq E\mid\text{false}\mid P\wedge P\mid P\vee P\mid\text{emp}$	P	coNP
L^*	$P::=L\mid P^*P$	NP	Π_2^P
$L\neg^*$	$P::=L\mid\neg P\mid P^*P$	PSPACE	PSPACE
$L-\!\!*$	$P::=L\mid P-\!\!*P$	PSPACE	PSPACE
$L\neg^*-\!\!*$	$P::=L\mid\neg P\mid P^*P\mid P-\!\!*P$	PSPACE	PSPACE

Josh Berdine 等人针对链表结构专门提出了分离逻辑的一个片段[22],并着重研究了蕴含合法性的判定过程.在分离逻辑的这个片段中,公式间蕴含问题的可判定性问题,探讨了通过算法和工具,将分离逻辑的纸上证明转换为自动化证明过程的可能性.分离逻辑的这个片段主要包括描述链表段的谓词以及相应的断言语言.文中通过一系列引理和证明,给出了两个判定过程:第 1 个判定过程是基于分离逻辑片段的语义判定过程.基于他们所设计的小模型属性,没有靠暴力推导分析来证明可判定性,而是通过调整局部性并通过分离逻辑的局部推理能力证明了可判定性;第 2 个判定过程是理论证明过程,这是目前为止对分离逻辑的第一个完整的证明理论[22],其优点在于,进行模型检验证明时不需要对每个实例都产生指数级的反例.

3.2 与其他逻辑之间的关系

分离逻辑依赖于堆结构,但缺乏成熟的判定过程可供使用.而一阶逻辑被广泛支持并且没有这些约束,能否将分离逻辑问题转换为一阶逻辑问题呢?文献[23]研究认为,无法对完整的分离逻辑进行转换,但是可以将命题

化分离逻辑翻译为一阶逻辑的可判定的子集.所谓命题化分离逻辑是指只包含命题连接词、对自然数的限定以及等价关系的分离逻辑部分.在分离逻辑中,最重要的部分是分离合取*以及分离蕴含–*两个连接词及其基本数据结构栈和堆.栈代表变量,堆代表内存,通过*对堆进行分解,通过–*对堆进行扩展.因此,翻译的最主要部分就是对这两个连接词的翻译以及对数据结构的转换.文献[23]研究设计的方法是在一阶逻辑中用向量来表示堆,并将*以及–*用向量操作表达.通过这一研究,分离逻辑的强大表达能力可以通过一种普通的经典逻辑来实现.

Lars Birkedal 等人研究了如何将分离逻辑应用到高阶语言中去[24,25],从而在面临高阶过程时也可以使用一系列高阶 Frame 规则进行局部推理.研究选择带有堆和指针的理想化 Algol 语言扩展作为基础,通过对类型系统的精化,开发了分离逻辑类型系统以及分离逻辑中 Frame 规则的各种相应高阶版本,并使用一种指称语义对所开发的类型系统进行了完全的正确性证明.

3.3 程序推理基础支持

3.3.1 数据结构的支持

分离逻辑的设计者 Reynolds 等人在文献[6]中描述了分离逻辑在处理列表和树等数据结构上的能力.Bornat 等人[26]研究了分离逻辑在处理 DAG(directed acyclic graphs)和图等更复杂数据结构和相关算法上的能力和优势,这使得分离逻辑的应用前景又得到了较大的扩展.文献[26]对分离逻辑在处理指针别名上的优点和方法作了剖析,然后使用分离逻辑对 DAG、部分图和部分 DAG 进行表达,并讨论了部分 DAG 的复制和分解、部分图的复制和分解、使用前向指针的图的复制和分解等问题,总结出了分离逻辑在处理这些数据结构上的一套方法,从而使得在对包含这些数据结构和算法的实际程序进行验证时有了可以借鉴的理论基础.

3.3.2 抽象和类型的支持

类型表示对系统处理对象的抽象.在实际编程中,类型主要包括结构化程序设计中表示数据的语言类型以及在面向对象设计中对数据和行为抽象的更复杂的抽象数据类型 ADT(abstract data type).分离逻辑需要对不同类型和不同层次的抽象进行支持,目前主要研究包括:

1) 对于 C 中结构体类的结构化数据类型的处理[27].澳大利亚 National ICT(Information and Communications Technology Centre)的研究提供了分离逻辑在 Isabelle/HOL(higher order language)中的一种嵌入,通过将类型结构信息嵌入定理证明器和通用规则中,产生了一种能够使用不同证明技术来处理地址别名的框架,以更好地支持 C 的结构化类型.
   - 这个框架提供了结构化类型信息的深嵌入(deep embedding),从而可以处理 C 的结构化类型数据在内存中存储时的大小、对齐和填充等方面的限制以及结构化类型的堆去引用的语义问题.
   - 泛化了早期的多类型堆和分离逻辑的重写以及证明规则,从而可以受益于机械化.虽然开销小涨,但是在验证中仍然可用.
   - 研究者的早期工作中实现的在语义中通过可信 ML(meta language)代码浅表翻译(shallow translation)来处理的结构化类型的各个方面,在这里被提升到 HOL 层次.
2) 对面向对象程序设计中抽象类型的支持,目前分离逻辑能够做到:
   - 引入抽象谓词和抽象谓词类[28],对包含对象的程序编写规格说明和进行推理.
   - 对继承的验证处理[29].在文献[28]研究的基础上,设计了分离逻辑的一个扩展,可以允许重载基类行为的派生类,对所继承下来的已被证明为安全的基类方法避免重新验证.采用的方法是使用两种规格说明:一种静态规格说明,用来验证类的实现和直接的方法调用;一种动态规格说明,用来验证动态进行的调用.
   - 创建了一个用于面向对象程序的原型系统[30].能够处理类与其超类之间的关系,仍然是采用基于静态和动态的规格说明,但是引入了规格说明分类,从而可以避免不需要的代码重验证.并通过谓词机制,支持类不变式和无损的类型转换.

3.3.3 递归和循环的支持

分离逻辑在处理指针程序时难以表达递归逻辑公式,也没有有效的办法表达 while 循环的最弱前置条件和最强后置条件.为此,文献[31,32]对分离逻辑进行了扩展,除了经典逻辑连接词和分离逻辑的分离连接词以外,又针对递归和循环加入了不动点连接词和延迟替换,所产生的分离逻辑扩展能够表达 while 循环的公理化语义,并支持对包含 while 循环的程序进行前向和后向的符号化分析,还可以表达递归定义.

3.3.4 程序推理自动化程度的提高

在对循环进行推理时,循环不变式的确定是关键.文献[5]研究了对于给定的 C 程序,如何自动产生分离逻辑循环不变式.其主要思想是,对循环体应用一个符号化求值,接着执行一个折叠操作.文献[33]基于断言精化给出了针对某类程序的证明自动化搜索方法,文献[34]通过引入相交类(intersection types)和强迫原则(coercion rules)两个特性,研究了如何在分离逻辑中进行系统化的证明搜索,在应用分离逻辑进行程序属性自动化验证方面进行了探索.

3.4 定理证明器

程序验证过程通常需要定理证明器的支持,尽管由于定理证明的不可判定问题,不可能自动完成全部推理证明过程.定理证明器的必要性体现在:(1) 定理证明器可以保证每一步证明过程的正确性;(2) 定理证明器一般都包含大量的策略库和引理库,这些库中的规则、引理和策略是通过把频繁使用的证明模式抽象表达出来得到的.在证明过程中,使用者和定理证明器以交互方式进行,使用者控制高层的证明过程,而定理证明器依据策略库完成琐碎而注重细节的重复性工作.在将分离逻辑嵌入到定理证明器方面有不少研究成果.对于目前主流的定理证明器,例如 Isabelle/HOL,Coq 等,分别都有研究将分离逻辑在其中实现[10,27,35].

此外,Appel 等人研究了如何开发和使用针对分离逻辑验证的策略库问题[36].分离逻辑在定理证明中的支持有两种选择:直接在一个逻辑框架(例如 Isabella)中实现分离逻辑,或者在高阶逻辑(例如 Coq 或 Isabelle/HOL)中定义分离逻辑的运算符.他们分析认为,后一种更为妥当:因为在证明一个命令式程序时,很多推理与内存单元无关而仅需关注程序数据结构代表的抽象数学对象.关于这些对象的引理,可以在一个通用目标的高阶逻辑中方便地证明,定理证明器中的高阶逻辑往往拥有大量具有良好文档帮助的引理库和策略库.

4 总 结

分离逻辑作为一种近年来提出的逻辑,因其本身所蕴含的分离思想,在验证包含指针的程序时,能够简洁、优雅地支持进行局部推理和模块化推理,已经在程序验证领域得到了重视和广泛使用.其验证适用范围是使用了指针的程序,包括简单命令式程序、面向对象程序以及并发程序,成功验证实例包括一些实用的算法、具体的系统模块和程序的某些属性.

对分离逻辑的逻辑属性、数据结构和算法支持、与其他逻辑之间关系以及定理证明器等一系列研究为利用分离逻辑进行程序验证打下了良好的基础,给程序验证领域引入了新的希望.同时也要看到,其验证实例的规模都不大,验证过程自动化程度较低.因此,将分离逻辑用于解决更复杂的软件系统的源代码级验证,仍然面临挑战,需要进一步解决许多技术难题.主要问题包括:

1. 分离逻辑的属性和能力.目前,对分离逻辑的研究大都针对其某一个子集.为使分离逻辑更好地支持程序验证,应该进一步研究分离逻辑的各种不同子集和各种不同扩展的逻辑属性、与其他逻辑之间的关系.研究分离逻辑的正确性、可满足性、可判定性、可表达性等逻辑属性,明确分离逻辑的表达、验证能力.
2. 语言类型的全面支持.就分离逻辑主要思想中对堆的建模而言,其对程序设计语言中各种类型的支持仍然是不够的,因此需要更全面地支持语言中类型.这包括从简单的对 C 语言中 union 等结构化类型的支持到比较困难的对复杂类型系统的支持,还要处理对象作为函数参数这样的参数多态性问题,最终将分离逻辑与类型系统集成起来.
3. 验证过程的自动化.目前在采用分离逻辑的验证过程中,从规格说明的编写到验证过程的开展都是以

人工为主的.而对于复杂的大型系统来说,这种方式显然是不足以胜任的.因此,提高验证自动化程度是分离逻辑进入实际应用的一个重要问题.可能的途径包括:(1) 研究如何根据添加适当注释的源程序,自动产生推理所需的断言、不变式乃至规格说明;(2) 研究应用分离逻辑,针对所要验证的程序属性进行自动推理的方法.

4. 定理证明器的支持.以定理证明器支持分离逻辑进行程序验证是必然的选择.为此,需要开展的研究工作包括:(1) 根据上述问题 1 中的研究成果,丰富与分离逻辑本身属性相关的证明策略;(2) 根据上述问题 3 中的研究成果,总结与抽象在程序属性验证过程中频繁使用的策略,并与判定过程集成,更好地进行机械化证明.

References:

[1] Zhang HG, Luo J, Jin G, Zhu ZQ, Yu FJ, Yan F. Development of trusted computing research. Wuhan University Journal of Natural Sciences, 2006,11(6):1407−1413.

[2] Seigneur JM, Farrell S, Jensen CD, Gray E, Chen Y. End-to-End trust starts with recognition. In: Proc. of Conf. on Security in Pervasive Computing (SPC 2003). Boppard, 2003. https://www.cs.tcd.ie/publications/tech-reports/reports.03/TCD-CS-2003-05.pdf

[3] Zhou MT, Tan L. Progress in trusted computing. Journal of University of Electronic Science and Technology of China, 2006,35(4): 686−697 (in Chinese with English abstract).

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

[5] Magill S, Nanevski A, Clarke E, Lee P. Inferring invariants in separation logic for imperative list-processing programs. In: Proc. of the 3rd Workshop on Semantics, Program Analysis and Computing Environments for Memory Management (SPACE 2006). Charleston, 2006. 47−60.

[6] Reynolds JC. Separation logic: A logic for shared mutable data structures. In: Proc. of the LICS 2002. Copenhagen, 2002. 55−74.

[7] Reynolds JC. An overview of separation logic. In: Proc. of the Verified Software: Theories, Tools, Experiments. Zurich, 2005. 460−469.

[8] O'Hearn PW, Reynolds JC, Yang H. Local reasoning about programs that alter data structures. In: Fribourg L, ed. Proc. of the Computer Science Logic. LNCS 2142, Berlin: Springer-Verlag, 2001. 1−19.

[9] Yang HS. Local reasoning for stateful programs [Ph.D. Thesis]. Urbana: University of Illinois at Urbana Champaign, 2001.

[10] Aeldt R, Marti N, Yonezawa A. Towards formal verification of memory properties using separation logic. In: Proc. of the 22nd Workshop of the Japan Society for Software Science and Technology. Sendi, 2005.

[11] Birkedal L, Torp-Smith N, Reynolds JC. Local reasoning about a copying garbage collector. In: Proc. of the Conf. Record of POPL 2004: The 31st ACM SIGPLANSIGACT Symp. on Principles of Programming Languages. New York: ACM Press, 2004. 220−231.

[12] Berdine J, Calcagno C, O'Hearn PW. Symbolic execution with separation logic. In: Yi K, ed. Proc. of the APLAS 2005. LNCS 3780, Berlin: Springer-Verlag, 2005. 52−68.

[13] Distefano D, O'Hearn PW, Yang H. A local shape analysis based on separation logic. In: Proc. of the TACAS. Vienna, 2006. 287−302.

[14] O'Hearn PW, Yang H, Reynolds JC. Separation and information hiding. In: Proc. of the POPL 2004. New York: ACM Press, 2004. 268−280.

[15] Brotherston J, Bornat R, Calcagno C. Cyclic proofs of program termination in separation logic. In: Proc. of the POPL 2008. New York: ACM Press, 2008. 101−112.

[16] Berdine J, Calcagno C, O'Hearn PW. Smallfoot: Modular automatic assertion checking with separation logic. In: Proc. of the FMCO 2005. LNCS 4111, Berlin: Springer-Verlag, 2005. 115−137.

[17] O'Hearn PW. Resources, concurrency and local reasoning. Theoretical Computer Science, 2004,375(1-3):271−307.

[18] Brookes SD. A semantics for concurrent separation logic. In: Proc. of the CONCUR 2004—Concurrency Theory, the 15th Int'l Conf. LNCS 3170, Berlin: Springer-Verlag, 2004. 16−34.

[19] Bornat R, Calcagno C, O'Hearn PW, Parkinson M. Permission accounting in separation logic. In: Proc. of the POPL 2005. New York: ACM Press, 2005. 259−270.

[20] Bornat R, Calcagno C, Yang H. Variables as resource in separation logic. In: Proc. of the 21st Annual Conf. on Mathematical Foundations of Programming Semantics (MFPS XXI). Elsevier ENTCS 155, 2006. 247−276.

[21] Calcagno C, Yang H, O'Hearn PW. Computability and complexity results for a spatial assertion language for data structures. In: Proc. of the FSTTCS. LNCS 2245, Berlin: Springer-Verlag, 2001. 108−119.

[22] Berdine J, Calcagno C, O'Hearn PW. A decidable fragment of separation logic. In: Proc. of the FSTTCS 2004. LNCS 3328, Berlin: Springer-Verlag, 2004. 97−109.

[23] Calcagno C, Gardner P, Hague M. From separation logic to first-order logic. In: Proc. of the FoSSaCS 2005. LNCS 3441, Berlin: Springer-Verlag, 2005. 395−409.

[24] Birkedal L, Torp-Smith N, Yang H. Semantics of separation-logic typing and higher-order frame rules for algol-like languages. Logical Methods in Computer Science, 2006,2(5):1−33.

[25] Birkedal L, Torp-Smith N, Yang H. Semantics of separation logic typing and higher-order frame rules. In: Proc. of the LICS 2005. Washington: IEEE Computer Society, 2005. 260−269.

[26] Bornat R, Calcagno C, O'Hearn P. Local reasoning, separation, and aliasing. In: Proc. of the 2nd Workshop on Semantics, Program Analysis and Computing Environments for Memory Management (SPACE 2001). Venice, 2004.

[27] Tuch H. Structured types and separation logic. In: Proc. of the 3rd Int'l Workshop on Systems Software Verification (SSV 2008). Sydney, 2008.

[28] Parkinson MJ, Bierman GM. Separation logic and abstraction. In: Proc. of the POPL 2005. New York: ACM Press, 2005. 247−258.

[29] Parkinson MJ, Bierman GM. Separation logic, abstraction and inheritance. In: Proc. of the POPL 2008. New York: ACM Press, 2008. 75−86.

[30] Chin WN, David C, Nguyen HH, Qin SC. Enhancing modular OO verification with separation logic. In: Proc. of the POPL 2008. New York: ACM Press, 2008. 87−99.

[31] Sims EJ. Extending separation logic with fixpoints and postponed substitution. In: Proc. of the AMAST. LNCS 3116, Berlin: Springer-Verlag, 2004. 475−490.

[32] Sims EJ. Pointer analysis and separation logic [Ph.D. Thesis]. Manhattan: Kansas State University, 2007.

[33] Ireland A. Towards automatic assertion refinement for separation logic. In: Proc. of the ASE 2006. IEEE Computer Society, 2006. 309−312.

[34] Chin WN, Nguyen HH, David C, Qin SC. Supporting proof search in verification using separation logic. Technical Report, School of Computing at NUS, 2007. http://www.comp.nus.edu.sg/~chinwn/papers/expressivity.ps

[35] Weber T. Towards mechanized program verification with separation logic. In: Proc. of the CSL 2004. LNCS 3210, Berlin: Springer-Verlag, 2004. 250−264.

[36] Appel AW. Tactics for separation logic. Department of Computer Science, Princeton University. 2006. http://www.cs.princeton.edu/~appel/papers/septacs.pdf

附中文参考文献:

[3] 周明天,谭良.可信计算及其进展.电子科技大学学报,2006,35(4):686−697.

黄达明(1976－),男,江苏南京人,博士生,讲师,主要研究领域为形式化验证和测试,安全评估.

曾庆凯(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为信息安全,分布计算.