基于分离逻辑的云存储系统验证

金 钊[1),3)] 王捍贫[1),2),3)] 张博闻[1),3)] 张 磊[1),3)] 曹永知[1),3)]

[1)](北京大学计算机科学与技术系 北京 100871)

[2)](广州大学计算机科学与网络工程学院 广州 510006)

[3)](高可信软件技术教育部重点实验室(北京大学) 北京 100871)

摘 要 数据的快速增长限制了传统存储技术存储和管理数据的能力,云存储系统应运而生.云存储系统最重要的特征是数据以块的形式存储,且每个块被视为一个独立的存储单元.因此,云存储系统通常包含两种存储单元:原始地址和块地址,这使得云存储系统与传统内存系统相比有着显著的不同.从而,如何保证云存储系统的可靠性成为了有待解决的难题.本文基于分离逻辑提出了一种系统方法来验证云存储系统管理程序的正确性.主要贡献包括:(1)提出了一种建模语言来描述云存储管理;(2)扩展了分离逻辑的断言语言来描述云存储系统中有关块的属性;(3)在上述两个语言的基础上,提出了一套霍尔型的规范规则对云存储系统进行推理.规范的前置和后置条件都以断言对的形式给出.运用这些方法,能够验证云存储管理程序的正确性.

关键词 分离逻辑;霍尔逻辑;云存储系统;建模语言;形式验证

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2020.02227

Reasoning about Cloud Storage Systems Based on Separation Logic

JIN Zhao[1),3)] WANG Han-Pin[1),2),3)] ZHANG Bo-Wen[1),3)] ZHANG Lei[1),3)] CAO Yong-Zhi[1),3)]

[1)](*Department of Computer Science and Technology, Peking University, Beijing* 100871)

[2)](*School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou* 510006)

[3)](*Key Laboratory of High Confidence Software Technologies (MOE), Peking University, Beijing* 100871)

Abstract The rapid growth of data has restricted the capability of traditional storage technologies to store and manage data. Massive growth in big data generated through Cloud Storage Systems (CSSs) has been observed. Compared with traditional storage systems, CSSs have lots of advantages, such as higher capacity, lower cost, and easier scalability. An important feature of CSSs is that data files are stored in a way of block. That is, the storage resources of CSSs consist of small block spaces with a fixed size. When storing a data file, the system may fist cut the file into some segments, and then put these segments into proper blocks, taking every segment as a whole. Hence, CSSs usually have two kinds of storage units: ordinary locations and block locations. At first glance, the above procedure of saving a file into blocks in a CSS looks similar to that of saving values into memory cells in a memory management system. But there are still some substantial differences. When user submit their data file to CSSs, the file of user uploaded is divided into lots of 128 MB segments firstly (the last segment may be less than the others). Then the system allocate those segments to the blocks one by one, and the size of each block is also 128 MB. Finally, the system will generate a file table to record the block addresses and the

收稿日期:2019-11-22;在线发布日期:2020-05-02.本课题得到国家自然科学基金(61772035,61972005,61932001)、国家科技攻关计划(2018YFB1003904,2018YFC1314200)资助.**金 钊**,博士研究生,主要研究方向为云存储程序的推理验证、数理逻辑、可计算理论.E-mail:jinzhao@pku.edu.cn.**王捍贫**(通信作者),博士,教授,中国计算机学会(CCF)会员,主要研究领域为计算机系统的形式语义和验证、算法和计算复杂性.E-mail:whpxhy@pku.edu.cn.**张博闻**,博士研究生,主要研究方向为云存储系统架构、形式化验证、辅助证明工具.**张 磊**,博士研究生,主要研究方向为基于分离逻辑的云计算建模与验证.**曹永知**,博士,教授,中国计算机学会(CCF)会员,主要研究领域为形式化方法及其应用、隐私性与安全性、不确定性推理.

relation between file segments and blocks. These characteristics make the properties of CSS management very different from those of traditional memory management. Although the concept of block storage has been proposed for many years, it becomes more complicated in CSSs. In order to improve the reliability and safety of cloud storage service, the correctness of the cloud storage management problem needs to be verified, which is an urgent problem to solve. While the special characteristics of the cloud storage system make the problem challenging. Then how do we appeal formal methods to model, describe and reason about CSSs? Separation Logic (SL), which is a Hoare-style logic, has a strong theoretical and practical significance. By using SL, some verification systems have been implemented. In this paper, based on SL, we propose a systematic method to verify the correctness of management programs in CSSs. The main contributions are as follows. (1) A language is defined to describe the cloud storage management. The new commands mainly focus on block operations. Accordingly, its operational semantics is given by using the concepts of heap and store. (2) Assertions in SL are extended to describe the properties of blocks in CSSs. Quantifiers over block variables and file variables are incorporated with ordinary SL assertion. (3) Hoare-style rules are proposed to reason about the CSSs. Pre- and post-conditions are pairs of assertions. One component of every pair is used to describe the properties of ordinary locations, while the other component is used to describe the properties of block locations. The soundness of the rules is also proved. Using these methods, the partial correctness of cloud storage management can be verified.

Keywords separation logic; Hoare logic; cloud storage systems; modeling language; formal verification

1 引 言

随着信息技术和计算机科学的发展,大数据正以创纪录的速度产生,并已成为一种公认的发展趋势[1].近年来,大数据引起了学术界、政府和业界的广泛关注.云计算是执行大规模复杂计算的一项快速发展的技术.大数据和云计算是相辅相成的.前者为用户提供了跨多个数据集的查询与处理数据的能力,而后者提供了底层引擎.云存储作为云计算的基础部分,在底层数据的采集、存储、维护和输出等方面起着重要作用.如今,许多云计算系统已投入使用,并拥有着自身的云存储系统.一个典型的例子是"谷歌文件系统"(Google File System,GFS)[2],它是一种可扩展的分布式文件系统,适用于大型数据密集型应用程序,在谷歌云计算的相关实际应用中使用广泛.

正如 GFS 一样,云存储系统具有比传统存储系统更多的特征,其中最重要的特征是数据文件以块的方式进行存储.也就是说,云存储系统的存储资源是由固定大小的小型块空间组成的.在存储数据文件时,系统会将文件分割成若干片段,然后将这些片段放入适当的块中,并将每个片段视为一个整体.以 GFS 为例,其工作方式大体如下.当 GFS 的客户发出文件存储请求时,系统首先将文件切分成几个片段,除最后一个之外,每个片段的大小都是固定的(通常为 64 MB).然后,系统会分配一系列块(在 GFS 中称为"chunks"[2])来逐一存放这些文件片段,相应得每个块的大小为 64 MB.最后,系统会返回一个块地址.这样以来就能构建一个文件表来记录文件片段和块之间的关系.其他云存储系统,比如"Hadoop 分布式文件系统"(Hadoop Distributed File System,HDFS)[3],有着类似的工作原理.

表面上看,上述云存储系统中将文件保存到块的操作与传统内存管理系统中将值保存到内存单元的过程类似,但两者有着实质性的区别.例如,数组允许放入到一系列连续的内存单元,而在大多数云存储系统中并没有"连续"的块地址.所不同的是,云存储系统或许将一个文件放置于不同的块,甚至有可能散落到不同的块服务器.这些差异使得云存储系统管理的特征相比传统内存管理有着显著区别.虽然块存储的概念已经提出多年,但在云存储系统

中其具有很高的复杂性.因此,如何保证云存储系统的可靠性成为了有待解决的难题.

通常云存储系统的可靠性体现在正确性和安全性两个方面.正确性是指对于每个输入经过算法执行都能得到预期的输出.安全性则是指需要一系列的机制来保护数据不被窃取或者损坏.显然,正确性是可靠性的最基本要求.正确性分析不仅能够证明数据处理过程的正确性,而且有助于发现潜在的系统设计错误.因此,本文的目标是验证云存储系统的正确性.在云存储系统中,正确性指的是系统管理程序的正确性.云存储系统管理程序是指处理用户请求或管理存储系统的一系列命令,例如 **create**、**append**、**delete** 等.验证程序正确性的方法有很多,比如软件测试技术和形式验证.前者因其低成本而饱受欢迎,但缺点是具有不完整性.尽管后者代价稍高,但可以提供严格的数学分析支持,并且在查找错误方面非常有效.因此,在高可信系统中得到了广泛应用.目前,形式验证方法已经足够成熟,能够用以验证大多数计算机程序的正确性,并且在云存储系统的形式化方面也有了一些尝试.我们简要得从以下四个方面来阐述这些工作.

云存储系统的建模方面:早在 1987 年,文献[4]就基于 Z 语言提出了一种针对分布式块存储服务的形式化描述方法,其中包括用户手册和实现者手册,分别描述了块服务接口与实现,是一种可用于块存储服务的形式规范.随后,文献[5]提出的抽象模型将文件视为一组大型数据记录,其中作者采用了实时进程代数的方法对文件管理系统和相关系统行为进行建模.近年来,Serbanescu 等人[6]提出了一种描述分布式数据聚集服务的模型架构,并在随后的工作中[7],提出了 Rule Markup 语言,可以用来表达数据检索和聚合规则.

云存储系统的安全性验证方面:Bansal 等人[8]创建了一种针对强安全意识的网络应用程序的模型,使用到了网络安全数据库 WebSpi,并在此基础上提出了一个具备鲁棒性的应对策略.文献[9]考虑了通过互联网将数据传输到公共云存储这一过程的安全性问题,并使用 Z 记号给出了一个云工作流的形式建模,从而能够分析其安全性和成本属性.James Stephen 等人[10]使用形式化方法分析数据流,提出了一种在不牺牲数据保密性的情况下的执行模型,用于验证云存储系统中的 Pig Latin 脚本执行过程.

验证云存储系统中的数据完整性方面:Ateniese 等人[11-12]在他们的"数据持有性验证"模型中使用了所谓的"第三方可审核性",从而确保了不受信任的存储文件的完整性.此外,文献[13]实现了一种针对公共云数据审计系统的隐私保护方法,并提出了一种基于轻量级通信和计算成本的审计机制.与此同时,还有许多其他审计机制,例如文献[14]和[15].

验证云存储系统的通用性方面:文献[16]给出了针对一种基本类 Unix 文件系统的形式化规范,并提出了证明系统 Athena 用以验证该文件系统的正确性.Pereverzeva 等人[17]为云存储系统的开发提供了一种形式化解决方案,能够对大型弹性数据存储系统进行建模,从而开发人员可以形式化的定义、开发以及验证其存储系统的体系结构.为了实现这一目标,作者使用 Event-B 语言和 Rodin 平台来创建形式化模型,并通过规则推理逐步完善该模型.

然而,所有上述工作都不能对现有的云存储系统进行推理,并且没有统一的建模语言和规范.在过去的十五年中,分离逻辑(Separation Logic)[18-19]已经从一种推理指针程序的新方法发展成了一个主流的针对可扩展程序的验证技术.分离逻辑之所以能够成功,是因为它能够将软件工程师对于计算机内存操作程序的概念模型与逻辑中解释语句正确性的逻辑模型相结合,从而形成一套理论证明系统来帮助解决扩展推理任务的关键问题.而在计算机科学的逻辑应用中,可扩展性是一个重要问题,甚至有人称之为核心问题[20].通过使用分离逻辑,一些工作实现了验证系统的构建[21-25].本文作者在前期工作中[26-27],基于分离逻辑尝试了对大规模数据存储系统的初步建模.然而,由于分离逻辑基于低阶存储模型,并不足以对云存储系统进行推理.

本文基于分离逻辑提出了一种系统方法来验证云存储管理程序的正确性.主要贡献包括:

(1)提出了一种建模语言来描述云存储管理,新增命令主要关注块操作.在此基础上,利用堆和储存的概念给出了该语言的操作语义.

(2)扩展了分离逻辑的断言语言,从而能够描述云存储系统中有关块的属性.量化了块变量和文件变量,并且将其与传统的分离逻辑断言相结合,以此作为形式验证的基础.

(3)提出了一组霍尔型的规范规则对云存储系统进行推理.规则中的前置和后置条件以断言对的形式给出.断言对的第一个分量用于描述传统原始地址的性质,而第二个分量描述块地址的性质.此外,给出了一个包含 While 循环的实际云存储系统

算法的验证示例,说明了本文所述方法的可行性.

本文第 2 节介绍相关预备知识;第 3 节提出一种云存储系统的建模语言;在第 4 节给出基于分离逻辑的断言语言;并在第 5 节提出规范语言对云存储系统进行推理;第 6 节通过实际算法的验证示例说明本文方法的可行性;最后一节对全文进行总结.

2 预备知识

分离逻辑是一种霍尔型的逻辑,用于对包含共享可操作数据结构的命令式程序进行推理.分离逻辑由三个部分构成,分别为一个建模语言、一个断言语言以及一个霍尔三元组型的规范语言.该规范语言结合了两个断言和一个命令,从而指定目标系统程序的(部分)正确性.此外,规范语言还包含一组推理规则,也是对程序进行演绎推理所必需的.

2.1 分离逻辑的建模语言

分离逻辑定义了一种建模语言来描述内存系统中有关共享可变数据结构的操作,其语法如下所示:

$e ::= n \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2$

$be ::= \mathbf{true} \mid \mathbf{false} \mid e_1 = e_2 \mid e_1 \leq e_2 \mid \neg\, be \mid be_1 \wedge be_2 \mid be_1 \vee be_2$

$c ::= \mathbf{skip} \mid x := e \mid \mathbf{if}\ be\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \mid \mathbf{while}\ be\ \mathbf{do}\ c \mid c_1 ; c_2 \mid x := \mathbf{cons}(e_1, \cdots, e_n) \mid x := [e] \mid [e] := e' \mid \mathbf{dispose}\ e$

其中,e 表示算术表达式,be 表示布尔表达式,c 表示命令.

在定义上述语言的语义时,分离逻辑使用了储存-堆的二元对来描述计算状态.储存 Stores_V 将变量映射到值,而 Heaps 将地址映射到值.

$\text{Stores}_V = V \rightharpoonup \text{Values}$ $\text{Heaps} = \bigcup_{A \subseteq_{\text{fin}} \text{Addresses}} (A \rightarrow \text{Values})$

$\text{Values} = \mathbb{Z}$ $\text{States}_V = \text{Stores}_V \times \text{Heaps}$

因此,一个状态 $\sigma \in \text{States}_V$ 是一个二元对的形式 $(s,h) \in \text{Stores}_V \times \text{Heaps}$.对于每个命令,分离逻辑通过状态间的转换关系定义了命令的结构性操作语言.以变量分配命令为例,其操作语义规则如下所示:

$$\frac{(e_1,(s,h)) \rightarrow k_1, \cdots, (e_n,(s,h)) \rightarrow k_n}{(x := \mathbf{cons}(e_1, \cdots, e_n),(s,h)) \rightarrow (s,[h \mid l : k_1 \mid \cdots \mid l+n-1 : k_n])}$$

其中,$l, \cdots, l+n-1 \in \text{Addresses} - \text{dom}(h_V)$,$[f \mid x : a]$ 表示将变量 x 映射到值 a,且将函数 f 定义域中的所有其他变量 y 映射到 $f(y)$.

2.2 分离逻辑的断言语言

分离逻辑的断言语言用以描述内存系统的属性,特别是有关堆的属性.因此,在 Hoare 逻辑[28]的断言基础上增加了堆运算符.分离逻辑断言的语法以 BNF 范式的形式表示如下:

$p ::= \mathbf{true} \mid \mathbf{false} \mid e_1 = e_2 \mid e_1 \leq e_2 \mid \neg\, p \mid p_1 \wedge p_2 \mid p_1 \vee p_2 \mid \forall x.p \mid \exists x.p \mid \mathbf{emp} \mid e_1 \mapsto e_2 \mid p_1 * p_2 \mid p_1 -\!\!* p_2$

最后四个断言用来描述堆操作,分别描述空堆、单堆、堆的分离合取以及分离蕴含.断言的语义通过真值给出,而其真值通过当前系统状态确定.为了形式化得给出断言的语义,首先引入两个关于堆的记号.令 h_1,h_2 和 h 表示堆,则 $h_1 \perp h_2$ 表示堆 h_1 和 h_2 的定义域是互斥的.此外,如果 $h_1 \cdot h_2 = h$ 表示堆 h_1 和 h_2 的联合恰好是堆 h.$\text{dom}(h)$ 用来表示堆的定义域 h.

利用上述记号,有关堆操作的断言语义如下所示:

$(s,h) \models \mathbf{emp}$ 当且仅当 h 的定义域为空,

即:$\text{dom}(h) = \varnothing$;

$(s,h) \models e_1 \mapsto e_2$ 当且仅当 $\text{dom}(h) = \{[\![e_1]\!](s,h)\}$,

且 $h([\![e_1]\!](s,h)) = [\![e_2]\!](s,h)$,

其中 $[\![e_i]\!](s,h) = k_i$,若 $(e_i,(s,h)) \rightarrow k_i$,$i = 1,2$;

$(s,h) \models (p_1 * p_2)$ 当且仅当存在两个子堆 h_1,h_2,使得,$h_1 \perp h_2$,$h_1 \cdot h_2 = h$,且 $(s,h_1) \models p_1$,$(s,h_2) \models p_2$;

$(s,h) \models (p_1 -\!\!* p_2)$ 当且仅当对任意堆 h' 有 $h' \perp h$,则 $(s,h \cdot h') \models p_2$,若 $(s,h') \models p_1$.

2.3 分离逻辑的规范语言

分离逻辑的规范是一个霍尔型三元组,形式为 $\{p_1\}c\{p_2\}$,其中 p_1 和 p_2 是断言,通常分别称为前置条件和后置条件,c 是上述建模语言中的命令.规范可以描述两种正确性,即部分正确性和完全正确性.本文只讨论部分正确性,其定义如下所示:

$\models \{p_1\}c\{p_2\}$ 当且仅当对任意系统状态 $(s,h) \in \text{States}_V$,有 $(s,h) \models p_1$ 蕴含

(1) $(s,h) \models p_1$ 蕴含 $(c,(s,h)) \rightsquigarrow^* \mathbf{abort}$ 不成立且

(2) 对任意状态 (s',h'),有

$(c,(s,h)) \rightsquigarrow^* (s',h')$ 蕴含 $(s',h') \models q$.

其中 $\rightsquigarrow^*$ 表示上述结构性操作语义中的多步转换关系,**abort** 表示中断.

分离逻辑给出了如下所示的部分正确性推理规则,对于建模语言中的每条基础命令,都给出了一条相应的规则.文献[18]中已经证明了其可靠性.

$\{\mathbf{emp}\}x:=\mathbf{cons}(e_1,\cdots,e_n)\{x\mapsto e_1,\cdots,e_n\}$；

$\{v=v'\wedge e\mapsto v''\}v:=[e]\{v=v''\wedge e[v'/v]\mapsto v''\}$；

$\{e\mapsto -\}[e]:=e'\{e\mapsto e'\}$；

$\{e\mapsto -\}\mathbf{dispose}(e)\{\mathbf{emp}\}$.

此外，分离逻辑还给出了一条如下所示的特殊且十分重要的规则，称为“框架规则”. 该规则的提出使得分离逻辑能够进行组合验证.

$$\frac{\{p_1\}c\{p_2\}}{\{p_1*p_3\}c\{p_2*p_3\}}$$

需要特别注意的是，分离逻辑中由于堆的特殊性，传统霍尔中的一些规则并不成立，比如不变性规则，如下所示的：

$$\frac{\{p_1\}c\{p_2\}}{\{p_1\wedge p_3\}c\{p_2\wedge p_3\}}$$

文献[24]已经证明，上述规则构成的推理系统是相对完备的.

3　云存储系统的建模语言

云存储系统的建模语言是程序设计语言 WHILEh[24] 的扩展，加入了新的元素来反应云存储系统的独特性质，即数据文件以块的形式进行存储. 自然地，我们通过添加描述文件和块的新结构来构造云存储系统的建模语言. 为了简洁起见，本语言将块的内容视为一个整数值. 事实上，这些值通常表示上下文中块的某些属性，比如日志文件中块的时间戳等.

3.1　语　法

本建模语言包含四种表达式：(算术)地址表达式、文件表达式、块表达式和布尔表达式.

$n\in\mathbb{Z}$：整数常量　　**true**，**false**：布尔常量

nil$\in$Atoms：保留字　　$e\in\langle\mathrm{Exp}\rangle$：地址表达式

$fe\in\langle\mathrm{FExp}\rangle$：文件表达式　　$bk\in\langle\mathrm{BKExp}\rangle$：块表达式

$be\in\langle\mathrm{BExp}\rangle$：布尔表达式　　$x,y\in\mathrm{Var}$：地址变量

$f_1,f_2\in\mathrm{BKVar}$：文件变量　　$b_1,b_2\in\mathrm{BKVar}$：块变量

$C\in\mathrm{Com}$：命令

表达式和命令的完整语法如下所示：

$e::=n\mid x,y,\cdots\mid e_1+e_2\mid e_1-e_2\mid e_1*e_2\mid \#f$

$fe::=\mathbf{nil}\mid f_1,f_2,\cdots\mid fe\bullet bk\mid fe_1\bullet fe_2$

$bk::=n\mid b_1,b_2,\cdots\mid f(e)$

$be::=e_1=e_2\mid e_1\leq e_2\mid bk_1==bk_2\mid\mathbf{true}\mid\mathbf{false}\mid$
$\neg\, be\mid be_1\wedge be_2\mid be_1\vee be_2$

$C::=\mathbf{skip}\mid x:=e\mid x:=\mathbf{cons}(e_1,\cdots,e_n)\mid x:=[e]\mid$
$[e]:=e'\mid\mathbf{despose}\ e\mid f:=\mathbf{create}(bk^*)\mid$
$f:=\mathbf{append}(bk^*)\mid f_1:=f_2\bullet bk^*\mid$
$\mathbf{delete}\ f\mid b:=bk\mid b:=\{bk\}\mid\{bk\}:=bk'\mid$
$\mathbf{delete}\ bk\mid C;C'\mid\mathbf{if}\ be\ \mathbf{then}\ C\ \mathbf{else}\ C'\mid$
$\mathbf{while}\ be\ \mathbf{do}\ C$

表达式方面，$\#f$ 表示文件 f 中所包含的块的数目，$f(e)$指明文件 f 中的第 i 个块，其中 i 是表达式 e 的值. 指令集方面，除了包含 WHILEh 语言中的所有命令之外，本建模语言还引入了一些新的命令用以描述云存储管理程序中关于文件和块的特殊操作. 其中，文件命令包含四个核心操作，足以描述云存储系统中的大多数日常文件操作，块命令描述块操作. 新增命令的直观含义如下所示：

- $f:=\mathbf{create}(bk^*)$：文件创建；
- $f:=\mathbf{append}(bk^*)$：文件内容添加；
- $f_1:=f_2\bullet bk^*$：文件地址添加；
- **delete** f：文件删除；
- $b:=bk$：块赋值；
- $b:=\{bk\}$：块查询；
- $\{bk\}:=bk'$：块修改；
- **delete** bk：块删除.

3.2　论　域

本模型包含 5 个组件，分别为：Stores_V，Stores_B，Stores_F，Heaps_V和 Heaps_B. Stores_V是一个将地址变量映射到地址的全函数. Stores_B是一个将块变量映射到块地址的全函数. 严格地讲，值、地址以及块地址属于不同的类型，但在本语言中，为了允许非受限地址运算，我们假设所有值、地址以及块地址都是整数. Stores_F是一个将文件变量映射到一个块地址序列的全函数. Heaps_V由一个整数的子集 Loc 索引，并使用间接寻址$[e]$进行访问，其中 e 是一个地址表达式. Heaps_B由一个整数的子集 BLoc 索引，并使用间接寻址$\{bk\}$进行访问，其中 bk 是一个块表达式.

$\mathrm{Value}\triangleq\{\cdots,-1,0,1,\cdots\}=\mathbb{Z}$

$\mathrm{Loc},\mathrm{BLoc},\mathrm{Atoms}\subseteq\mathrm{Values}$

$\mathrm{Var}\triangleq\{x,y,\cdots\}$　$\mathrm{BKVar}\triangleq\{b_1,b_2,\cdots\}$

$\mathrm{FVar}\triangleq\{f_1,f_2,\cdots\}$

$\mathrm{Stores}_V\triangleq\mathrm{Var}\rightarrow\mathrm{Values}$　$\mathrm{Stores}_B\triangleq\mathrm{BKVar}\rightarrow\mathrm{Values}$

$\mathrm{Stores}_F\triangleq\mathrm{Var}\rightarrow\mathrm{BLoc}^*$

$\mathrm{Heaps}_V\triangleq\mathrm{Loc}\xrightarrow[\mathrm{fin}]{}\mathrm{Values}$

$\mathrm{Heaps}_B\triangleq\mathrm{BLoc}\xrightarrow[\mathrm{fin}]{}\mathrm{Values}$

其中$\rightarrow$和$\xrightarrow[\mathrm{fin}]{}$的定义参见文献[28]. Loc、BLoc 和 Atoms 是互斥的. $\mathrm{BLoc}^*=\{(bloc_1,\cdots,bloc_n)\mid bloc_i\in\mathrm{BLoc},n\in\mathbb{N}\}$，其中对于任意 BLoc^* 中的元素

$(bloc_1,\cdots,bloc_n)$，$|(bloc_1,\cdots,bloc_n)|$表示其长度，也就是说，$|(bloc_1,\cdots,bloc_n)|=n$.

为了保证地址分配总是能够成功，我们对集合 Loc 和 BLoc 设置了一个要求. 假定对于任意正整数 m，Loc 中有无穷多个长度为 m 的连续整数序列. 同样地，对 BLoc 做类似的设定. 并将 Loc 和 BLoc 的元素置为非负整数，从而就得以满足上述要求. 此外，将 Atoms 置为负整数，**nil** 置为−1.

本建模语言的系统状态定义如下：

$$\text{States}\triangleq \text{Stores}_V\times\text{Stores}_B\times\text{Stores}_F\times \text{Heaps}_V\times\text{Heaps}_B$$

一个系统状态 $\sigma\in$ States 是一个五元组：(s_V,s_B,s_F,h_V,h_B).

3.3 语 义

一旦完成了系统状态定义，就可以指定表达式的求值规则. 值得注意的是，当我们试图给出一个表达式的语义时，一些储存可能是不相关的. 例如，表达式 $\#f$ 的求值只涉及 Stores_F 和 Stores_V. 因此，在求值过程中只需要列出与该表达式相关的储存. 此外，本语言的表达式求值不依赖于堆. 表达式的指称语义按照表达式的种类分别给出.

(1) 地址表达式的指称语义：

$\mathcal{E}:\langle\text{Exp}\rangle\to\text{Stores}_V\to\text{Stores}_F\to\mathbb{Z}$，

$[\![e]\!]:\text{Stores}_V\to\text{Stores}_F\to\mathbb{Z}$，

$[\![n]\!](s_V)(s_F)=n$；

$[\![x]\!](s_V)(s_F)=s_V(x)$；

$[\![e_1*e_2]\!](s_V)(s_F)=[\![e_1]\!](s_V)(s_F)*[\![e_2]\!](s_V)(s_F)$，

其中 $*$ 为+，−或 $*$；

$[\![\#f]\!](s_V)(s_F)=|s_F(f)|$.

(2) 文件表达式的指称语义：

$\mathcal{F}:\text{FExp}\to\text{Stores}_B\to\text{Stores}_F\to\mathbb{Z}^*$，

$[\![fe]\!]:\text{Stores}_B\to\text{Stores}_F\to\mathbb{Z}^*$，

$[\![\textbf{nil}]\!](s_B)(s_F)=()$；

$[\![f]\!](s_B)(s_F)=s_F(f)$；

$[\![fe\bullet bk]\!](s_B)(s_F)=(bloc_1,\cdots,bloc_n,bloc')$，

如果$[\![fe]\!](s_B)(s_F)=(bloc_1,\cdots,bloc_n)$

且$[\![bk]\!](s_B)(s_F)=bloc'$；

$[\![fe_1\bullet fe_2]\!](s_B)(s_F)=(bloc_1,\cdots,bloc_n,bloc_1',\cdots,bloc_n')$，

如果$[\![fe_1]\!](s_B)(s_F)=(bloc_1,\cdots,bloc_n)$

且$[\![fe_2]\!](s_B)(s_F)=(bloc_1',\cdots,bloc_n')$.

(3) 块表达式的指称语义：

$\mathcal{K}:\langle\text{BKExp}\rangle\to\text{Stores}_V\to\text{Stores}_B\to\text{Stores}_F\to\mathbb{Z}$，

$[\![bk]\!]:\text{Stores}_V\to\text{Stores}_B\to\text{Stores}_F\to\mathbb{Z}$，

$[\![n]\!](s_V)(s_B)(s_F)=n$；

$[\![b]\!](s_V)(s_B)(s_F)=s_B(b)$；

$[\![f(e)]\!](s_V)(s_B)(s_F)=bloc_i$，

如果 $s_F(f)=(bloc_1,\cdots,bloc_n)$，$[\![e]\!](s_V)(s_F)=i$

且 $1\le i\le n$.

(4) 布尔表达式的指称语义：

$\mathcal{B}:\langle\text{BExp}\rangle\to\text{Stores}_V\to\text{Stores}_B\to\text{Stores}_F\to\mathbb{B}$，

$$[\![e_1=e_2]\!](s_V)(s_F)=\begin{cases}T, & \text{如果}[\![e_1]\!](s_V)(s_F)=[\![e_2]\!](s_V)(s_F)\\ F, & \text{其他}\end{cases};$$

$[\![e_1\le e_2]\!](s_V)(s_F)$可类似定义；

$$[\![bk_1==bk_2]\!](s_V)(s_B)(s_F)=\begin{cases}T, & \text{如果}[\![bk_1]\!](s_V)(s_B)(s_F)=[\![bk_2]\!](s_V)(s_B)(s_F)\\ F, & \text{其他}\end{cases};$$

$[\![\textbf{true}]\!](s_V)(s_B)(s_F)=T$；$[\![\textbf{false}]\!](s_V)(s_B)(s_F)=F$；

$$[\![\neg\, be]\!](s_V)(s_B)(s_F)=\begin{cases}T, & \text{if } [\![be]\!](s_V)(s_B)(s_F)=F\\ F, & \text{if } [\![be]\!](s_V)(s_B)(s_F)=T\end{cases};$$

$$[\![be_1\wedge be_2]\!](s_V)(s_B)(s_F)=\begin{cases}T, & \text{如果}[\![be_1]\!](s_V)(s_B)(s_F)=T\text{ 且}\\ & \quad [\![be_2]\!](s_V)(s_B)(s_F)=T\\ F, & \text{其他}\end{cases};$$

$[\![be_1\vee be_2]\!](s_V)(s_B)(s_F)$可类似定义.

注意到，在块表达式 $f(e)$ 的求值中，如果 e 的求值结果 i 大于等于 $n+1$，则条件不满足. 为了处理这一特殊情况，我们定义记号 **abort** 表示中断. 由于 bk 可以为 $f(e)$，因此布尔表达式 $bk_1==bk_2$ 的求值也需要作补充定义，如下所示. 由上述求值过程可知，表达式的求值不依赖堆 Heaps_V 和 Heaps_B，从而不会导致内存错误.

$[\![f(e)]\!](s_V)(s_B)(s_F)=\textbf{abort}$，

如果 $s_F(f)=(bloc_1,\cdots,bloc_n)$，$[\![e]\!](s_V)(s_F)=i$

且 $n+1\le i$；

$[\![bk_1==bk_2]\!](s_V)(s_B)(s_F)=\textbf{abort}$，

如果$[\![bk_1]\!](s_V)(s_B)(s_F)=\textbf{abort}$

或$[\![bk_2]\!](s_V)(s_B)(s_F)=\textbf{abort}$.

为了给出本建模语言的形式语义，参照文献[24]，采用了堆上的一些关键操作.

- $\text{dom}(h_H)$表示一个堆 $h_H\in\text{Heaps}_H$ 的定义域，其中 h_H 论及 h_V 和 h_B，Heaps_H 论及 Heaps_V 和 Heaps_B；类似地，$\text{dom}(s_S)$ 表示一个储存 $s_S\in\text{Stores}_S$ 的定义域，其中 s_S 论及 s_V，s_B 和 s_F，Stores_S 论及 Stores_V，Stores_B 和 Stores_F；

- $h_H\#h_H'$表示 h_H 和 h_H' 的定义域互斥；

- h_H*h_H'是一个有限函数取定于 h_H 和 h_H' 的联

合，并且 $h_H \# h'_H$ 成立；

- $i \mapsto j$ 是一个单堆部分函数将 i 映射到 j；

- 对于一个从 U 到 W 的部分函数 f 以及从 V 到 W 的部分函数 f'，从 $U \cup V$ 到 W 的部分函数 $f[f']$（称为更新函数）定义如下：

$$f[f'](i) \triangleq \begin{cases} f'(i), & \text{如果 } i \in \mathrm{dom}(f') \\ f(i), & \text{如果 } i \in \mathrm{dom}(f) \\ \text{无定义}, & \text{其他} \end{cases}.$$

本节余下部分将给出本建模语言新增命令的操作语义. 各命令的解释用到了一个格局间的关系“$\rightsquigarrow$”. 而每个格局可以为一个系统状态 σ、一个命令-状态对$\langle C,\sigma\rangle$或者一个特殊的格局 $fault$ 表示内存错误.

$$\frac{(bk_1,\sigma)\to m_1,\cdots,(bk_n,\sigma)\to m_n}{\begin{array}{l}\langle f:=\mathbf{create}(bk_1,\cdots,bk_n),\sigma\rangle \rightsquigarrow \\ (s_V,s_B,s_F[(m'_1,\cdots,m'_n)/f], \\ h_V,[h_B|m'_1:m_1|\cdots|m'_n:m_n])\end{array}}$$

其中 $m'_1,\cdots,m'_n \in \mathrm{BLoc}-\mathrm{dom}(h_B)$

$$\frac{(bk_1,\sigma)\to m_1,\cdots,(bk_n,\sigma)\to m_n}{\begin{array}{l}\langle f:=\mathbf{append}(bk_1,\cdots,bk_n),\sigma\rangle \rightsquigarrow \\ (s_V,s_B,s_F[(s_F(f)\cdot(m'_1,\cdots,m'_n))/f], \\ h_V,[h_B|m'_1:m_1|\cdots|m'_n:m_n])\end{array}}$$

其中 $m'_1,\cdots,m'_n \in \mathrm{BLoc}-\mathrm{dom}(h_B)$

$$\frac{(bk_1,\sigma)\to m_1,\cdots,(bk_n,\sigma)\to m_n}{\langle f_1:=f_2\cdot(bk_1,\cdots,bk_n),\sigma\rangle \rightsquigarrow (s_V,s_B,s_F[(s_F(f_2)\cdot(m_1,\cdots,m_n))/f_1],h_V,h_B)}$$

$$\langle \mathbf{delete}\ f,\sigma\rangle \rightsquigarrow (s_V,s_B,s_F|(\mathrm{dom}(s_F)\backslash\{f\}),h_V,h_B)$$

$$\frac{(bk,\sigma)\to m}{\langle b:=\{bk\},\sigma\rangle \rightsquigarrow (s_V,s_B[h_B(m)/b],s_F,h_V,h_B)}$$

$$\frac{(bk,\sigma)\to m}{\langle b:=\{bk\},\sigma\rangle \rightsquigarrow (s_V,s_B[m/b],s_F,h_V,h_B)}$$

$$\frac{(bk,\sigma)\to m_1,(bk',\sigma)\to m_2}{\langle \{bk\}:=bk',\sigma\rangle \rightsquigarrow (s_V,s_B,s_F,h_V,h_B[m_2/m_1])}$$

$$\frac{(bk,\sigma)\to m}{\langle \mathbf{delete}\ bk,\sigma\rangle \rightsquigarrow (s_V,s_B|(\mathrm{dom}(s_B)\backslash\{m\}),s_F,h_V,h_B)}$$

其中，函数$[f|x:a]$表示将变量 x 映射到值 a，且将函数 f 定义域中的所有其他变量 y 映射到 $f(y)$. 函数$[f|s]$表示与 f 类似的函数只是将 s 从 f 的定义域中移除. $(m_1,\cdots,m_n)\cdot(m'_1,\cdots,m'_n)$表示两个序列的连接. 由于 $s_F(f)$是一个块地址的序列，从而 $s_F(f)\cdot(m_1,\cdots,m_n)$也是一个块地址的系列，其表示 $s_F(f)$后追加了块地址序列$(m_1,\cdots,m_n)$.

文件创建命令首先在 h_B中分配一组块空间，并将初始值存入其中. 然后按顺序返回这些块地址，并在储存 s_F中将文件变量映射到这些地址. 文件内容添加命令的工作方式类似，但区别在于 h_B中块分配完成后，返回的块地址将在 s_F中追加到当前文件所指向的块地址之后. 文件地址添加命令不需要分配块空间及为其赋初值，而是在 s_F中直接将指定的块地址追加到当前文件的块地址列表. 文件删除命令将从 s_F中移除变量 f 的映射关系，然而并不改变 h_B.

块查询命令首先通过块表达式求值获取块地址，然后在 h_B中查询该块地址下的块内容. 查询得到的块内容值将在储存 s_B中赋值给一个块变量. 块地址命令在 s_B中为块变量赋值. 块修改命令在 h_B中将指定地址下的块内容更改. 块删除命令在 s_B中删除块变量的映射，但不会更改 h_B. 注意，文件删除和块删除命令都不会改变堆 h_B. 这是因为在大多数云存储系统中，删除操作不会直接回收块空间，而是在一段预设的时间内将其隐藏以防可能的数据恢复情况，预设时间过后才会释放块空间.

4　云存储系统的断言语言

为了描述云存储系统的性质，本文构造了一种能够同时处理原始地址和块地址的逻辑. BI Pointer Logic[24] 为原始地址的描述提供了一种优雅而强大的形式化方法. 本文扩展了 BI Pointer Logic，加入了文件和块表达式来描述云存储系统的文件与块属性. 断言语言的形式语义通过满足关系“$\models$”来定义，该关系作用于一个系统状态与断言之间，即 $(s_V,s_B,s_F,h_V,h_B)\models p$ 表示断言 p 在状态(s_V,s_B,s_F,h_V,h_B)下成立.

本断言语言重写了 BI Pointer Logic 用以处理原始地址，称之为“地址断言”(location assertions). 这是一种保守的扩展，从而使良好的逻辑性质得以保留. 此外，定义了全新的“块断言”(block assertions). 两种断言都是基于前文的表达式构建的.

4.1　地址断言

地址断言扩展了 BI Pointer Logic 的断言，用以描述原始地址属性. 但是两者之间存在一些差异，最显著的一点是地址断言中加入了文件表达式的量化. 地址断言的语法如下所示：

$$\begin{array}{l}\alpha ::= \mathbf{true}_V \mid \mathbf{false}_V \mid \neg\alpha \mid \alpha_1\wedge\alpha_2 \mid \alpha_1\vee\alpha_2 \mid \\ \quad \alpha_1\to\alpha_2 \mid e_1=e_2 \mid e_1\le e_2 \mid \forall x.\alpha \mid \exists x.\alpha \mid \\ \quad \mid \forall f.\alpha \mid \exists f.\alpha \mid \mathbf{emp}_V \mid e\mapsto e' \mid \alpha_1 * \alpha_2 \mid \\ \quad \alpha_1 -\!\!* \alpha_2\end{array}$$

直观地，地址断言的真值只取决于地址变量相

关的储存和堆.值得注意的一点是,地址断言可能包含诸如 $\#f$ 之类的表达式,因此满足关系也取决于文件储存.给定一个地址断言 α,通过对 α 的结构归纳来定义 $s_V,s_F,h_V\models\alpha$,新增断言的语义如下所示,其他与 BI Pointer Logic 类似的断言定义不再赘述.

$s_V,s_F,h_V\models\mathbf{true}_V$;

$s_V,s_F,h_V\models\alpha_1\rightarrow\alpha_2$ 当且仅当如果 $s_V,s_F,h_V\models\alpha_1$,则 $s_V,s_F,h_V\models\alpha_2$;

$s_V,s_F,h_V\models(\forall x.\alpha)$ 当且仅当 $s_V[n/x],s_F,h_V\models\alpha$,对任意 $n\in\mathrm{Loc}$;

$s_V,s_F,h_V\models(\forall f.\alpha)$ 当且仅当 $s_V,s_F[\bar{n}/f],h_V\models\alpha$,对任意 $\bar{n}=(n_1,n_2,\cdots,n_k)\in\mathrm{BLoc}^*$;

$s_V,s_F,h_V\models(\exists f.\alpha)$ 当且仅当 $s_V,s_F[\bar{n}/f],h_V\models\alpha$,对某个 $\bar{n}=(n_1,n_2,\cdots,n_k)\in\mathrm{BLoc}^*$;

$s_V,s_F,h_V\models\mathbf{emp}_V$ 当且仅当 $\mathrm{dom}(h_V)=\varnothing$;

$s_V,s_F,h_V\models\alpha_1*\alpha_2$ 当且仅当存在 h_V^1,h_V^2,其中 $h_V^1\#h_V^2$ 且 $h_V=h_V^1*h_V^2$,使得 $s_V,s_F,h_V^1\models\alpha_1$ 且 $s_V,s_F,h_V^2\models\alpha_2$.

4.2 块断言

块断言用以描述块的属性.因此,除了传统的逻辑连接词和量词外,还包含了描述堆操作的逻辑算子,其构建基于前文的块表达式.块断言的语法如下所示:

$$\begin{aligned}\beta::=&\mathbf{true}_B\mid\mathbf{false}_B\mid\neg\beta\mid\beta_1\wedge\beta_2\mid\beta_1\vee\beta_2\mid\\&\beta_1\rightarrow\beta_2\mid bk_1==bk_2\mid\forall x.\beta\mid\\&\exists x.\beta\mid\forall b.\beta\mid\exists b.\beta\mid\forall f.\beta\mid\exists f.\beta\mid\mathbf{emp}_B\mid\\&bk\mapsto bk'\mid\beta_1*\beta_2\mid\beta_1-\!\!*\beta_2\mid f\hookrightarrow fe\mid fe=fe'\end{aligned}$$

显然,块断言的真值取决于块变量相关的储存和堆.由于块表达式可能包含诸如 $f(e)$ 的形式,从而块断言的真值也取决于文件储存.给定一个块断言 β,通过对 β 的结构归纳来定义 $s_V,s_B,s_F,h_B\models\beta$,一些关键的块断言语义如下所示,其他类似于相近结构的地址断言,此处不再赘述.

$s_V,s_B,s_F,h_B\models\mathbf{true}_B$;

$s_V,s_B,s_F,h_B\models bk_1==bk_2$ 当且仅当 $[\![bk_1]\!]\sigma=[\![bk_2]\!]\sigma$;

$s_V,s_B,s_F,h_B\models(\forall x.\beta)$ 当且仅当 $s_V[n/x],s_B,s_F,h_B\models\beta$,对任意 $n\in\mathrm{Loc}$;

$s_V,s_B,s_F,h_B\models(\forall f.\beta)$ 当且仅当 $s_V,s_B,s_F[\bar{n}/f],h_B\models\beta$,对任意 $\bar{n}=(n_1,n_2,\cdots,n_k)\in\mathrm{BLoc}^*$;

$s_V,s_B,s_F,h_B\models(\exists b.\beta)$ 当且仅当 $s_V,s_B[n/b],s_F,h_B\models\beta$,对某个 $n\in\mathrm{BLoc}$;

$s_V,s_B,s_F,h_B\models(\beta_1-\!\!*\beta_2)$ 当且仅当对任意块堆 h_B',如果 $h_B'\#h_B$ 且 $s_V,s_B,s_F,h_B'\models\beta_1$,则 $s_V,s_B,s_F,h_B*h_B'\models\beta_2$;

$s_V,s_B,s_F,h_B\models f\hookrightarrow fe$ 当且仅当 $[\![f]\!]\sigma=\bar{m}$,$[\![fe]\!]\sigma=\bar{n}$,且 $h_B(m_i)=n_i$ 对于 $1\leq i\leq k$,

其中 $\bar{m}=(m_1,m_2,\cdots,m_k)$,$\bar{n}=(n_1,n_2,\cdots,n_k)\in\mathrm{BLoc}^*$,使得 $|\bar{m}|=|\bar{n}|=k$.

4.3 全局断言

地址断言和块断言能够描述云存储系统的一些性质,但是可以看出它们是相互独立的.为了描述云存储系统的完整属性,需要将它们以恰当的方式组合.本节定义二元组(地址断言,块断言),称之为"全局断言"(global assertions).形式上采用符号 p 来代表全局断言,其语法如下所示:

$$\begin{aligned}p::=&\langle\alpha,\beta\rangle\mid\mathbf{true}\mid\mathbf{false}\mid\neg\ p\mid p_1\wedge p_2\mid p_1\vee p_2\mid\\&p_1\rightarrow p_2\mid\forall x.p\mid\exists x.p\mid\forall b.p\mid\exists b.p\mid\forall f.p\mid\exists f.p\mid\\&\mathbf{emp}\mid p_1*p_2\mid p_1-\!\!*p_2\end{aligned}$$

全局断言的真值取决于所有类型的储存和堆.给定一个全局断言 p,通过对 p 的结构归纳来定义 $s_V,s_B,s_F,h_V,h_B\models p$,完整的语义如下所示:

$\mathbf{true}\triangleq\langle\mathbf{true}_V,\mathbf{true}_B\rangle$;$\mathbf{emp}\triangleq\langle\mathbf{emp}_V,\mathbf{emp}_B\rangle$;

$s_V,s_B,s_F,h_V,h_B\models\langle\alpha,\beta\rangle$ 当且仅当 $s_V,s_F,h_V\models\alpha$,且 $s_V,s_B,s_F,h_B\models\beta$;

$s_V,s_B,s_F,h_V,h_B\models\neg p$ 当且仅当 $s_V,s_B,s_F,h_V,h_B\not\models\neg p$;

$s_V,s_B,s_F,h_V,h_B\models p_1\vee p_2$ 当且仅当 $s_V,s_B,s_F,h_V,h_B\models p_1$,或者 $s_V,s_B,s_F,h_V,h_B\models p_2$;

$s_V,s_B,s_F,h_V,h_B\models p_1\rightarrow p_2$ 当且仅当如果 $s_V,s_B,s_F,h_V,h_B\models p_1$,则 $s_V,s_B,s_F,h_V,h_B\models p_2$;

$s_V,s_B,s_F,h_V,h_B\models\forall x.p$ 当且仅当 $s_V[n/x],s_B,s_F,h_V,h_B\models p$,对任意 $n\in\mathrm{Loc}$;

$s_V,s_B,s_F,h_V,h_B\models\forall b.p$ 当且仅当 $s_V,s_B[n/b],s_F,h_V,h_B\models p$,对任意 $n\in\mathrm{BLoc}$;

$s_V,s_B,s_F,h_V,h_B\models\forall f.p$ 当且仅当 $s_V,s_B,s_F[\bar{n}/f],h_V,h_B\models p$,对任意 $\bar{n}=(n_1,n_2,\cdots,n_k)\in\mathrm{BLoc}^*$;

$s_V,s_B,s_F,h_V,h_B\models\mathbf{emp}$ 当且仅当 $\mathrm{dom}(h_V)=\varnothing$ 且 $\mathrm{dom}(h_B)=\varnothing$;

$s_V,s_B,s_F,h_V,h_B\models p_1*p_2$ 当且仅当存在 h_H^1,h_H^2,其中 $h_H^1\#h_H^2$ 且 $h_H=h_H^1*h_H^2$,使得 $s_V,s_B,s_F,h_V^1,h_B^1\models p_1$ 且 $s_V,s_B,s_F,h_V^2,h_B^2\models p_2$,其中 h_H^i 论及 h_V^i 和 h_B^i,且 $i=1,2$;

$s_V,s_B,s_F,h_V,h_B\models(p_1-\!\!*p_2)$ 当且仅当对任意块堆 h_H',如果 $h_H'\#h_H$ 且 $s_V,s_B,s_F,h_V',h_B'\models p_1$,则 $s_V,s_B,s_F,h_V*h_V',h_B*h_B'\models p_2$,

其中 h_H 论及 h_V 和 h_B.

为了便于进行形式推理，接下来引入一些复杂形式的缩写，如下所示：

$e\mapsto e_0,\cdots,e_n\triangleq(e\mapsto e_0)*\cdots*(e+n\mapsto e_n)$；

$e\mapsto -\triangleq\exists e'.e\mapsto e'$；

$f\mapsto -\triangleq\exists\bar{n}.f\mapsto\bar{n}$，其中 $\bar{n}=(n_1,n_2,\cdots,n_k)\in\mathrm{BLoc}^*$；

$bk\mapsto -\triangleq\exists bk'.bk\mapsto bk$.

本节设定的语义保持了经典逻辑的基本定律，并且分离逻辑中关于堆的常用定律依然有效，例如关于 **emp** 和 $*$ 的可交换独异点定律、关于 $*$ 的并行规则以及关于 $-\!\!*$ 的邻接规则，如下所示：

$$\frac{p\rightarrow q\quad r\rightarrow s}{p*r\rightarrow q*s},\ \frac{p*r\rightarrow s}{p\rightarrow r-\!\!*s},\ \frac{p\rightarrow r-\!\!*s\quad q\rightarrow r}{p*q\rightarrow s}$$

全局断言的优点可以概括为如下三个方面：

(1) 全局断言的两个分量地址断言和块断言可以分别描述云存储系统模型的两个堆 Heaps_V 和 Heaps_B. 同时，二元对的形式又可以准确的表达文件和块的关系，从而能够简洁精确的表示系统状态.

(2) 本文主要关注云存储系统中文件和块的性质，因此扩展了分离逻辑关于低阶存储模型的断言语言. 引入了新的连接词诸如 $==$，$\mapsto$ 和 $\not\mapsto$，从而能够描述云存储系统的性质.

(3) 量化了文件变量和块变量，从而提升了断言语言的表达能力. 结合新定义的断言，能够描述规范语言中的霍尔三元组的前置和后置条件，从而对云存储管理程序进行推理.

综上所示，本文的逻辑系统与分离逻辑有着显著区别，并且更具表达能力. 断言语言能够很好的支持下节所述的规范语言. 有关断言语言性质的更多讨论将在未来工作中展开.

5　云存储系统规范语言

云存储系统的规范规则可以采取与分离逻辑类似的方式来构建，即采用霍尔三元组的形式. 关键的不同点在于需要将前置和后置条件限定在上一节所述的云存储系统的全局断言. 此外，本文只讨论部分正确性. 在形式上，规范 $\{p\}\,C\,\{q\}$ 是一个三元组，其中，p 和 q 是全局断言，C 是命令.

5.1　转移函数

根据语法，任何布尔表达式都不能出现在任何类型的断言中，从而也不能出现在规范中. 这就导致包含布尔表达式的命令，诸如 **if** 和 **while** 语句不能直接用于前置或后置条件. 为了解决这一问题，本节定义转移函数 T，将布尔表达式映射到全局断言.

$T\in$ Transfer functions $=$

Boolean expressions $\rightharpoonup$ Global assertions

$T(e_1=e_2)=\langle e_1=e_2,\mathbf{true}_B\rangle$；

$T(e_1\leq e_2)=\langle e_1\leq e_2,\mathbf{true}_B\rangle$；

$T(bk_1==bk_2)=\langle\mathbf{true}_V,bk_1==bk_2\rangle$；

$T(\mathbf{true})=\mathbf{true}$；$T(\mathbf{false})=\mathbf{false}$；$T(\neg be)=\neg T(be)$；

$T(be_1\wedge be_2)=T(be_1)\wedge T(be_2)$

$T(be_1\vee be_2)=T(be_1)\vee T(be_2)$

本节余下部分将给出云存储系统的规范规则. 通常规则是由前提和结论构成的. 正如大多数逻辑一样，我们提取出了前提为空的规则，称之为“公理”.

5.2　公　理

对于每条基础命令，给出一条或者多条公理(额外添加的公理用于处理实际推理中的特殊情况). 简单赋值语句和 **skip** 语句的公理与霍尔逻辑中的相应公理类似.

$$\{p\}\,\mathbf{skip}\,\{p\}\tag{A1}$$

$$\{\langle x=x'\wedge\mathbf{emp}_V,\beta\rangle\}\,x:=e\ \ \{\langle x=e[x'/x]\wedge\mathbf{emp}_V,\beta\rangle\}\tag{A2}$$

变量分配和查询命令将改变 Stores_V，从而会影响到地址断言的真值. 本文为了保证公理系统的简洁、实用性，限定被修改的变量不在块断言中出现.

$$\{\langle x=x'\wedge\mathbf{emp}_V,\beta\rangle\}\,x:=\mathbf{cons}(e_1,\cdots,e_n)\ \ \{\langle x\mapsto e_1[x'/x],e_2[x'/x],\cdots,e_n[x'/x],\beta\rangle\}\tag{A3}$$

其中 x 不出现在 β 中.

$$\{\langle x=x'\wedge e\mapsto x'',\beta\rangle\}\,x:=[e]\ \ \{\langle x=x''\wedge e[x'/x]\mapsto x'',\beta\rangle\}\tag{A4}$$

其中 x 不出现在 β 中.

变量修改和删除命令仅会影响 Heaps_V，因此块断言分量在命令的执行过程中保持不变.

$$\{\langle e\mapsto -,\beta\rangle\}[e]:=e'\{\langle e\mapsto e',\beta\rangle\}\tag{A5}$$

$$\{\langle e\mapsto -,\beta\rangle\}\,\mathbf{dispose}(e)\{\langle\mathbf{emp}_V,\beta\rangle\}\tag{A6}$$

对于块命令，由于块表达式不出现在地址断言，从而经过命令的执行，只有全局断言的块断言分量会发生改变.

$$\{\langle\alpha,b==b'\wedge\mathbf{emp}_B\rangle\}\,b:=bk\ \ \{\langle\alpha,b==bk[b'/b]\wedge\mathbf{emp}_B\rangle\}\tag{A7}$$

$$\{\langle\alpha,b==b'\wedge bk\mapsto b''\rangle\}\,b:=\{bk\}\ \ \{\langle\alpha,b==b''\wedge bk[b'/b]\mapsto b''\rangle\}\tag{A8}$$

$$\{\langle\alpha,bk\mapsto -\rangle\}\{bk\}:=bk'\{\langle\alpha,bk\mapsto bk'\rangle\}\tag{A9}$$

$$\{\langle\alpha,bk\mapsto -\rangle\}\,\mathbf{delete}\ bk\{\langle\alpha,\mathbf{emp}_B\rangle\}\tag{A10}$$

在大多数情况下,块查询命令用于查询某个文件中的块,因此额外添加一条公理作为块查询命令公理的特殊情况,从而提高公理系统的推理能力.

$$\{\langle \#f_2=i-1, f_1 \mapsto f_2 \cdot bk \cdot f_3\rangle\} b:=\{f_1(i)\}$$
$$\{\langle \#f_2=i-1, f_1 \mapsto f_2 \cdot bk \cdot f_3 \wedge b==bk\rangle\} \quad \text{(A11)}$$

其中 b 不出现在 bk 中.

由于命令执行将会改变 Stores_F,因此可能同时影响到地址断言和块断言.而当被修改变量恰好在地址断言中出现,会出现别名引用问题.在实际证明过程中,可采用下节所述的辅助变量替换和辅助变量重命名规则来处理这一问题.因此,为了简化公理,此处只讨论被修改的文件变量不出现在地址断言中的情况.

$$\{\langle \alpha, f=f' \wedge \mathbf{emp}_B\rangle\} f:=\mathbf{create}(bk^*)$$
$$\{\langle \alpha[f'/f], f \mapsto bk_1[f'/f] \cdot \cdots \cdot bk_n[f'/f]\rangle\} \quad \text{(A12)}$$

$$\{\langle \alpha, f=f' \wedge f \mapsto ge\rangle\} f:=\mathbf{append}(bk^*)$$
$$\{\langle \alpha[f'/f], f \mapsto fe \cdot bk_1[f'/f] \cdot \cdots \cdot bk_n[f'/f]\rangle\} \quad \text{(A13)}$$

$$\{\langle \alpha, f \mapsto -\rangle\}\ \mathbf{delete}\ f\{\langle \alpha, \mathbf{emp}_B\rangle\} \quad \text{(A14)}$$

其中 f 不出现在 α 中.

$$\{\langle \alpha, f_1 \mapsto -\wedge f_2 \cdot bk^* \mapsto fe \cdot (bk_1, \cdots, bk_n)\rangle\}$$
$$f_1:=f_2 \cdot bk^* \{\langle \alpha, f_1 \mapsto fe \cdot (bk_1, \cdots, bk_n)\rangle\} \quad \text{(A15)}$$

其中 f 不出现在 α 中.

上述规则的可靠性证明将在附录 1 中给出.

5.3 规 则

与前面的公理不同,本节的规则适用于任意命令.在本文的设定中,复合语句规则和分离逻辑的结构性规则保持有效.

顺序复合语句规则适用于程序的顺序执行.

$$\frac{\{p\}C\{q\} \quad \{q\}C'\{r\}}{\{p\}C;\ C'\{r\}} \quad \text{(R1)}$$

条件语句规则的两个前提分别与条件命令的两个分支对应.

$$\frac{\{p \wedge T(be)\}C\{q\} \quad \{p \wedge \neg\ T(be)\}C'\{q\}}{\{p\}\ \mathbf{if}\ be\ \mathbf{then}\ C\ \mathbf{else}\ C'\{q\}} \quad \text{(R2)}$$

循环语句规则指明循环不变量在循环体中得以保持不变.

$$\frac{\{p \wedge T(be)\}C'\{p\}}{\{p\}\ \mathbf{while}\ \text{be}\ \mathbf{do}\ C'\{p \wedge \neg\ T(be)\}} \quad \text{(R3)}$$

推论规则的含义是我们可以强化前置条件或者弱化后置条件.

$$\frac{\models p' \to p \quad \{p\}C\{q\} \quad \models q \to q'}{\{p'\}C\{q'\}} \quad \text{(R4)}$$

接下来对分离逻辑的结构性规则进行扩展,需要将其低阶系统状态提升到云存储系统的五元组.此处作保守地扩展,从而分离逻辑的良好性质得以保留,特别是有助于处理变量别名引用问题.定义 $\text{FV}(C)$ 表示在命令 C 中出现的自由变量集合,$\text{Modify}(C)$ 表示被命令 C 修改的自由变量集合,即是出现在赋值语句左侧的变量.令 Var_V 为地址变量的集合,Var_B 为块变量的集合,Var_F 为文件变量的集合,Var_S 论及 Var_V,Var_B 和 Var_F.Y_S 为储存中所有已分配变量的集合,由于储存分为地址,块和文件三种类型,从而定义 Y_S 论及 Y_V,Y_B 和 Y_F.类似地,X_S 为被命令 C 修改的变量集合,定义 X_S 论及 X_V,X_B 和 X_F.于是,$X=X_V \cup X_B \cup X_F$,$Y=Y_V \cup Y_B \cup Y_F$.根据上述定义,对于命令 C,其中 $\text{Modify}_V(C)=X_V$,$\text{Modify}_B(C)=X_B$,$\text{Modify}_F(C)=X_F$ 以及 $\text{FV}(C) \subseteq Y$,给出云存储系统的结构性规则,如下所示:

辅助变量消除规则:

$$\frac{\{p\}C\{q\}}{\{\exists x_S.p\}C\{\exists x_S.q\}},\ x_S \notin Y_S \quad \text{(R5)}$$

其中 $x_S \in Var_S$,x_S 论及 x_V,x_B 和 x_F.

辅助变量重命名规则:

$$\frac{\{p\}C\{q\}}{\{p[y_S/x_S]\}C\{q[y_S/x_S]\}},$$
$$x_S \notin Y_S \wedge y_S \notin (\text{FV}(C) \cup \text{FV}(p) \cup \text{FV}(q)) \quad \text{(R6)}$$

其中 x_S,$y_S \in Var_S$,x_S 论及 x_V,x_B 和 x_F,y_S 论及 y_V,y_B 和 y_F.

框架规则:

$$\frac{\{p\}C\{q\}}{\{p * r\}C\{q * r\}},\ \text{Modify}_S(C) \cap \text{FV}(r)=\varnothing \quad \text{(R7)}$$

其中 $\text{Modify}_S(C)$ 论及 $\text{Modify}_V(C)$,$\text{Modify}_B(C)$ 和 $\text{Modify}_F(C)$.

6 程序示例:APPEND 算法验证

本节通过云存储系统中的实际样例 APPEND 算法的正确性验证,演示如何运用上节给出的公理和规则对程序进行推理.

算法 1. APPEND 算法.

输入:文件 f_1,f_2

输出:文件 f_2,其内容为初始内容后追加了文件 f_1 的内容

Algorithm APPEND (f_1, f_2)

1. $i:=1$;
2. WHILE $i <= \#f_1$ DO

3. $bc:=\{f_1(i)\}$;
4. $f_2:=\textbf{append}(bc)$;
5. $i:=i+1$;
6. END WHILE

经过算法 APPEND 的执行，文件的内容应该为其初始内容后追加了文件的内容，下面给出形式化地描述，将以前文定义的云存储系统的规范语言给出，是一个霍尔三元组的形式. 为了能够描述文件内容及算法执行的演变过程，需要引入量词. 此外霍尔三元组的前置和后置条件都是全局断言，是地址断言和块断言的二元对，验证过程能够体现出该二元对形式能够很好地描述云存储的系统状态.

$$\{\exists f_3\langle \#f_3=\#f_1, f_1\rightsquigarrow f_3*f_2\rightsquigarrow f_4\rangle\}$$
$$\text{APPEND}\ (f_1,f_2)$$
$$\{\exists f_3\langle \#f_3=\#f_1, f_1\rightsquigarrow f_3*f_2\rightsquigarrow f_4\bullet f_3\rangle\}$$

其中 f_1 是待添到文件 f_2 后面的文件，f_3 表示文件 f_1 的内容，f_4 表示文件 f_2 的初始内容.

下面将逐步给出完整的程序验证过程.

1. $\{\exists f_3\langle \#f_3=\#f_1, f_1\rightsquigarrow f_3*f_2\rightsquigarrow f_4\rangle\}$ (Given)
2. $\{\exists f_3\langle 1\leq \#f_1+1\wedge \#\textbf{nil}=0\wedge \#f_3=\#f_1, f_1\rightsquigarrow \textbf{nil}\bullet f_3*f_2\rightsquigarrow f_4\bullet \textbf{nil}\rangle\}$ (1)
3. $\{\exists f_5\exists f_6\langle 1\leq \#f_1+1\wedge \#f_5=0\wedge \#f_6=\#f_1, f_1\rightsquigarrow f_5\bullet f_6*f_2\rightsquigarrow f_4\bullet f_5\rangle\}$ (2)
4. $\{\exists f_5\exists f_6\langle 1\leq \#f_1+1\wedge \#f_5=0\wedge \#f_6=\#f_1, f_1\rightsquigarrow f_5\bullet f_6*f_2\rightsquigarrow f_4\bullet f_5\rangle\}$
$i:=1$
$\{\exists f_5\exists f_6\langle i\leq \#f_1+1\wedge \#f_5=i-1\wedge \#f_6=\#f_1-i+1, f_1\rightsquigarrow f_5\bullet f_6*f_2\rightsquigarrow f_4\bullet f_5\rangle\}$
((3),A1)

经过第一条命令 $i:=1$，得到了一个形式上相比初始条件更为复杂的条件，接下来要面对一个循环语句. 为了能够运用循环语句规则(R3)对其进行推理，首先需要找出循环不变量.

令 $A\equiv \exists f_5\exists f_6\langle i\leq \#f_1+1\wedge \#f_5=i-1\wedge \#f_6=\#f_1-i+1, f_1\rightsquigarrow f_5\bullet f_6*f_2\rightsquigarrow f_4\bullet f_5\rangle$

$C\equiv bc:=\{f_1(i)\};\ f_2:=\textbf{append}(bc);\ i:=i+1$

$be\equiv i\leq \#f_1$

此处 f_5 表示待拷贝的内容，f_6 表示文件中剩余的内容. 接下来将证明 A 是循环不变量，即 $\{A\wedge T(be)\}C\{A\}$.

5. $A\wedge T(be)=\{\exists f_5\exists f_6\langle i\leq \#f_1\wedge \#f_5=i-1\wedge \#f_6=\#f_1-i+1, f_1\rightsquigarrow f_5\bullet f_6*f_2\rightsquigarrow f_4\bullet f_5\rangle\}$
6. $\{\exists f_5\exists f_7\exists bk\langle i\leq \#f_1\wedge \#f_5=i-1\wedge \#f_7=\#f_1-i+1-1, f_1\rightsquigarrow f_5\bullet bk\bullet f_7*f_2\rightsquigarrow f_4\bullet f_5\rangle\}$ (5)
7. $\{\exists f_5\exists f_7\exists bk\langle i\leq \#f_1\wedge \#f_5=i-1\wedge \#f_7=\#f_1-i+1-1, f_1\rightsquigarrow f_5\bullet bk\bullet f_7*f_2\rightsquigarrow f_4\bullet f_5\rangle\}$
$bc:=\{f_1(i)\}$
$\{\exists f_5\exists f_7\exists bk\langle i\leq \#f_1\wedge \#f_5=i-1\wedge \#f_7=\#f_1-i+1-1, f_1\rightsquigarrow f_5\bullet bk\bullet f_7*f_2\rightsquigarrow f_4\bullet f_5\wedge bc==bk\rangle\}$ ((6),A11)
8. $\{\exists f_5\exists f_7\exists bk\langle i\leq \#f_1\wedge \#f_5=i-1\wedge \#f_7=\#f_1-i+1-1, f_1\rightsquigarrow f_5\bullet bk\bullet f_7*f_2\rightsquigarrow f_4\bullet f_5\wedge bc==bk\rangle\}$
$f_2:=\textbf{append}(bc)$
$\{\exists f_5\exists f_7\exists bk\langle i\leq \#f_1\wedge \#f_5=i-1\wedge \#f_7=\#f_1-i+1-1, f_1\rightsquigarrow f_5\bullet bk\bullet f_7*f_2\rightsquigarrow f_4\bullet f_5\bullet bk\rangle\}$ ((7),A13)
9. $\{\exists f_8\exists f_7\langle i\leq \#f_1\wedge \#f_8=i-1+1\wedge \#f_7=\#f_1-i+1-1, f_1\rightsquigarrow f_8\bullet f_7*f_2\rightsquigarrow f_4\bullet f_8\rangle\}$(8)
10. $\{\exists f_5\exists f_6\langle i\leq \#f_1\wedge \#f_5=i-1+1\wedge \#f_6=\#f_1-i+1-1, f_1\rightsquigarrow f_5\bullet f_6*f_2\rightsquigarrow f_4\bullet f_5\rangle\}$ (9)
11. $\{\exists f_5\exists f_6\langle i\leq \#f_1\wedge \#f_5=i-1+1\wedge \#f_6=\#f_1-i+1-1, f_1\rightsquigarrow f_5\bullet f_6*f_2\rightsquigarrow f_4\bullet f_5\rangle\}$
$i:=i+1$
$\{\exists f_5\exists f_6\langle i\leq \#f_1+1\wedge \#f_5=i-1\wedge \#f_6=\#f_1-i+1, f_1\rightsquigarrow f_5\bullet f_6*f_2\rightsquigarrow f_4\bullet f_5\rangle\}$
((10),A2)

令 $f_6=bk\bullet f_7$，可由第 5 条得到第 6 条. 再令 $f_8=f_5\bullet bk$，可由第 8 条得到第 9 条. 将第 9 条中的 f_7 和 f_8 分别替换成 f_6 和 f_5，则可得到第 10 条. 注意到，第 11 条的后置条件 $\{\exists f_5\exists f_6\langle i\leq \#f_1+1\wedge \#f_5=i-1\wedge \#f_6=\#f_1-i+1, f_1\rightsquigarrow f_5\bullet f_6*f_2\rightsquigarrow f_4\bullet f_5\rangle\}$ 就是循环不变量 A 本身. 于是，经过上述推理，得出 $\{A\wedge T(be)\}C\{A\}$. 接下来，使用循环语句规则(R3)，可得出 $\{A\}$ **while** be **do** $C\{A\wedge\neg\ T(be)\}$.

12. $A\wedge\neg\ T(be)=\exists f_5\exists f_6\langle i=\#f_1+1\wedge \#f_5=i-1\wedge \#f_6=\#f_1-i+1, f_1\rightsquigarrow f_5\bullet f_6*f_2\rightsquigarrow f_4*f_5\rangle$
13. $\{\exists f_5\exists f_6\langle i\leq \#f_1+1\wedge \#f_5=i-1\wedge \#f_6=\#f_1-i+1, f_1\rightsquigarrow f_5\bullet f_6*f_2\rightsquigarrow f_4*f_5\rangle\}$
while be **do** C
$\{\exists f_5\exists f_6\langle i=\#f_1+1\wedge \#f_5=i-1\wedge \#f_6=\#f_1-i+1, f_1\rightsquigarrow f_5\bullet f_6*f_2\rightsquigarrow f_4*f_5\rangle\}$
((5)～(11),R3)

经过循环语句的推理可得出，当循环终止时

$i=\#f_1+1$,从而根据第 13 条,可以得出如下结论.

14. $\{\exists f_5 \exists f_6 \langle i=\#f_1+1 \wedge \#f_5=\#f_1 \wedge \#f_6=0, f_1 \curvearrowright f_5 \cdot f_6 * f_2 \curvearrowright f_4 * f_5 \rangle\}$ (13)

15. $\{\exists f_5 \langle \#f_5=\#f_1, f_1 \curvearrowright f_5 * f_2 \curvearrowright f_4 * f_5 \rangle\}$ (14)

16. $\exists f_3 \langle \#f_3=\#f_1, f_1 \curvearrowright f_3 * f_2 \curvearrowright f_4 * f_3 \rangle\}$ (15)

根据 1～16 条推理以及顺序复合语句规则 R1,证明了算法 APPEND 的正确性.由上述程序示例可知,建模语言的指令集能够描述云存储管理程序的常用关键操作,运用规范语言的公理规则可以对程序进行推导,从而完成正确性证明.此外,断言语言具有较强的表达能力,能够以简洁的形式表示循环不变量.因此,利用本方法能够准确、灵活地验证云存储系统的正确性.

7 结 论

本文对分离逻辑进行了扩展,从而验证了云存储系统管理程序的正确性.基于分离逻辑的命令式语言,新增了描述云存储系统中关于文件和块操作的命令.扩展了分离逻辑的断言语言来描述块存储的特殊性质.提出了一套霍尔三元组型的规范规则对云存储系统进行推理.运用这些方法验证了一个包含 While 循环的实际算法.结果表明证明过程是科学有效的.

未来工作将从以下几个方面展开:(1)考虑云存储系统的更多性质,例如并发性、键值对属性以及块地址与原始地址的关系等;(2)研究断言语言的可表达性、可判定性以及模型检查算法;(3)证明本文提出的规范规则的可表达性和相对完全性.

致 谢 本文作者感谢审稿人提出的宝贵意见以及编辑部老师的辛勤工作!

参 考 文 献

[1] Hashem I A T, Yaqoob I, Anuar N B, et al. The rise of "big data" on cloud computing: Review and open research issues. Information Systems, 2015, 47: 98-115

[2] Ghemawat S, Gobioff H, Leung S T. The Google file system //Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003). New York, USA, 2003: 29-43

[3] Shvachko K, Kuang H, Radia S, et al. The Hadoop distributed file system//Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies. Washington, USA, 2010: 1-10

[4] Gimson R. The formal documentation of a block storage service. Oxford University Computing Laboratory, Programming Research Group, 1987

[5] Wang Y, Ngolah C F, Tan X, et al. The formal design model of a file management system (FMS). International Journal of Software Science and Computational Intelligence, 2011, 3(1): 90-113

[6] Serbanescu V, Pop F, Cristea V, et al. Architecture of distributed data aggregation service//Proceedings of the 28th International Conference on Advanced Information Networking and Applications (AINA 2014). Victoria, Canada, 2014: 727-734

[7] Serbanescu V, Pop F, Cristea V, et al. A formal method for rule analysis and validation in distributed data aggregation service. World Wide Web, 2015, 18(6): 1717-1736

[8] Bansal C, Bhargavan K, Delignat-Lavaud A, et al. Keys to the cloud: Formal analysis and concrete attacks on encrypted Web storage//Proceedings of the 2nd International Conference on Principles of Security and Trust. Rome, Italy, 2013: 126-146

[9] Freitas L, Watson P. Formalizing workflows partitioning over federated clouds: Multi-level security and costs. International Journal of Computer Mathematics, 2014, 91(5): 881-906

[10] James Stephen J, Savvides S, Seidel R, et al. Program analysis for secure big data processing//Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering. New York, USA, 2014: 277-288

[11] Ateniese G, Burns R, Curtmola R, et al. Provable data possession at untrusted stores//Proceedings of the 14th ACM Conference on Computer and Communications Security. New York, USA, 2007: 598-609

[12] Ateniese G, Di Pietro R, Mancini L V, et al. Scalable and efficient provable data possession//Proceedings of the 4th International Conference on Security and Privacy in Communication Networks. New York, USA, 2008: 1-10

[13] Wang C, Wang Q, Ren K, et al. Privacy-preserving public auditing for data storage security in cloud computing// Proceedings of the 30th IEEE International Conference on Computer Communications. New York, USA, 2010: 1-9

[14] Shraer A, Cachin C, Cidon A, et al. Venus: Verification for untrusted cloud storage//Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop. New York, USA, 2010: 19-30

[15] Wang B, Li B, Li H. Oruta: Privacy-preserving public auditing for shared data in the cloud//Proceedings of the 5th International Conference on Cloud Computing. New York, USA, 2012: 295-302

[16] Arkoudas K, Zee K, Kuncak V, et al. Verifying a file system implementation//Proceedings of the 6th International Conference on Formal Engineering Methods (ICFEM, 2014). Luxembourg, 2004: 373-390

[17] Pereverzeva I, Laibinis L, Troubitsyna E, et al. Formal modelling of resilient data storage in cloud//Proceedings of the 15th International Conference on Formal Engineering Methods. Queenstown, New Zealand, 2013: 363-379

[18] Reynolds J C. Separation logic: A logic for shared mutable data structures//Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science. Copenhagen, Denmark, 2002: 55-74

[19] Qin Sheng-Chao, Xu Zhi-Wu, Ming Zhong. Survey of research on program verification via separation logic. Journal of Software, 2017, 28(8): 2010-2025(in Chinese)
(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)

[20] David P, Spring J, O'Hearn P. Why separation logic works. Philosophy & Technology, 2019, 32(2): 483-516

[21] Krogh-Jespersen M, Svendsen K, Birkedal L. A relational model of types-and-effects in higher-order concurrent separation logic//Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL, 2017). Paris, France, 2017: 218-231

[22] Lee W, Park S. A proof system for separation logic with magic wand//Proceedings of the 41st ACM SIGPLAN Symposium on Principles of Programming Languages(POPL, 2014). San Diego, USA, 2014: 477-490

[23] Dodds M, Feng X, Parkinson M, et al. Deny-guarantee reasoning//Proceedings of the 18th European Symposium on Programming(ESOP, 2009). York, UK, 2009: 363-377

[24] Yang H, O'Hearn P. A semantic basis for local reasoning//Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS, 2002). Grenoble, France, 2002: 402-416

[25] Svendsen K, Birkedal L. Impredicative concurrent abstract predicates//Proceedings of the 18th European Symposium on Programming(ESOP, 2014). Grenoble, France, 2014: 149-168

[26] Jing Yuxin, Wang Hanpin, Huang Yu, et al. A modeling language to describe massive data storage management in cyber-physical systems. Journal of Parallel and Distributed Computing, 2017, 103: 113-120

[27] Wang H, Jin Z, Zhang L, et al. Reasoning about cloud storage systems//Proceedings of the 3rd International Conference on Data Science in Cyberspace(DSC, 2018). Guangzhou, China, 2018: 107-114

[28] Hoare C A R. An axiomatic basis for computer programming. Communications of the ACM, 1969, 12(10): 576-580

附录 1.　规范语言公理的可靠性.

附录中证明规范语言公理的可靠性，给出公理 A4 可靠性的完整证明过程，该公理对应于分离逻辑的查询语句，其证明说明分离逻辑的公理规则提升到本文的五元组系统状态依然可靠. 从而说明本文保守得扩展了分离逻辑，保持了其良好性质. 其余公理和规则可按照类似的方式进行证明，故本文不再赘述.

首先给出命令 $x:=[e]$ 的指称语义，如下所示：

$$[\![x:=[e]]\!](s_F,s_B,s_V,h_B,h_V)=(s_F,s_B,s_V[h_V([\![e]\!](s_F)(s_V))/x],h_B,h_V)$$

令 $s'_V=s_V[h_V([\![e]\!](s_F)(s_V))/x]$，由于 x 不在 β 中自由出现，从而可得出下述推理：

$$\frac{s_V,s_B,s_F,h_V,h_B\models\langle x=x'\wedge e\mapsto x'',\beta\rangle}{\begin{array}{c}s_V,s_F,h_V\models x=x'\wedge e\mapsto x'',\\ \text{且 } s_V,s_B,s_F,h_B\models\beta\end{array}}$$

$$\frac{\begin{array}{c}[\![x]\!](s_V,s_F)=[\![x']\!](s_V,s_F),\\ \mathrm{dom}(h_V)=\{[\![e]\!](s_V)(s_F)\},\\ h_V([\![e]\!](s_V)(s_F))=[\![x'']\!](s_V)(s_F),\\ \text{且 } s_V,s_B,s_F,h_B\models\beta\end{array}}{\begin{array}{c}[\![x]\!](s_V,s_F)=[\![x']\!](s_V,s_F),\\ \mathrm{dom}(h_V)=\{[\![e[x'/x]]\!](s_V)(s_F)\},\\ h_V([\![e[x'/x]]\!](s_V)(s_F))=[\![x'']\!](s_V)(s_F),\\ \text{且 } s_V,s_B,s_F,h_B\models\beta\end{array}}$$

$$\frac{\begin{array}{c}[\![x]\!](s'_V,s_F)=h_V([\![e]\!](s_V)(s_F))=[\![x'']\!](s'_V,s_F),\\ \mathrm{dom}(h_V)=\{[\![e[x'/x]]\!](s'_V)(s_F)\},\\ h_V([\![e[x'/x]]\!](s'_V)(s_F))=[\![x'']\!](s'_V,s_F),\\ \text{且 } s_V,s_B,s_F,h_B\models\beta\end{array}}{\begin{array}{c}s'_V,s_B,s_F,h_B\models x=x''\wedge e[x'/x]\mapsto x'',\\ \text{且 } s_V,s_B,s_F,h_B\models\beta\end{array}}$$

$$\frac{s'_V,s_B,s_F,h_B,h_V\models\langle x=x''\wedge e[x'/x]\mapsto x'',\beta\rangle}{}$$

根据全局断言(二元组)$\langle x=x'\wedge e\mapsto x'',\beta\rangle$ 的语义定义，由给定系统状态五元组满足该断言，可得出 $s_V,s_F,h_V\models x=x'\wedge e\mapsto x''$ 且 $s_V,s_B,s_F,h_B\models\beta$. 公理 A4 的条件指出，$x$ 不在 β 中自由出现，因此接下来的证明只需考虑全局断言的第一个分量，地址断言. 直观上，该公理涉及的命令 $x:=[e]$ 查询堆 Heaps_V 中的内容. 由于变量 x 可能在表达式 e 中出现，因此公理的前置条件中用 x' 保留了原变量 x 的值，并在后置条件中 e 处的用保留的值进行替换. 在余下两步的推理中可以看出，这一技巧保证了经过命令的执行，s_V 变为 s'_V 后得以满足公理的后置条件.

综上所述，对于给定的系统状态 (s_V,s_B,s_F,h_V,h_B) 满足公理的前置条件 $\langle x=x'\wedge e\mapsto x'',\beta\rangle$，经过命令 $x:=[e]$ 的执行后所得到的状态 $(s_V[h_V([\![e]\!](s_F)(s_V))/x],s_B,s_F,h_V,h_B)$，刚好满足公理的后置条件 $\langle x=x''\wedge e[x'/x]\mapsto x'',\beta\rangle$. 从而证明了公理 A4 是可靠的.

JIN Zhao, Ph. D. candidate. His research interests include verifying and reasoning about cloud storage systems programs, mathematical logic, and computability theory.

WANG Han-Pin, Ph. D., professor. His research interests include formal semantics and verification of computer systems, algorithms and computational complexity.

ZHANG Bo-Wen, Ph. D. candidate. His research interests include cloud storage system architecture, formal verification, and automated verification tools.

ZHANG Lei, Ph. D. candidate. His research interests include modeling and verification in cloud computing based on separation logic.

CAO Yong-Zhi, Ph. D., professor. His research interests include formal methods, privacy and security, and reasoning about uncertainty in artificial intelligence.

Background

Cloud Computing is rising fast with the advances in Information Technology and Computing Science. As an elementary part of Cloud Computing, cloud storage plays an important role in underlying data collection, storage, maintenance and output. Nowadays, many clouding computing systems have already been put into use, with their own Cloud Storage Systems (CSSs). Compare with traditional storage systems, CSSs have more features. An important one is that data are stored in blocks in CSSs, and each block is considered as a storage unit. Hence, CSSs usually have two kinds of storage units: ordinary locations and block locations. These characteristics lead to the different management in CSSs from tradition and higher complexity as well. Therefore, the problem of reliability of CSSs is coming up.

The reliability of CSSs may include many issues, such as retrievability, search efficiency, correctness of management programs, etc. Here we focus on the correctness of management programs, which is the basis of the reliability of CSSs. However, due to the special features of CSSs, it is difficult for existing formal methods to describe the execution process of CSSs management programs.

Memory management is challenging for formal verification, and memory errors are not easily handled during program execution. Separation Logic (SL), which is a Hoare-style logic, is a well-established approach for formal verification of programs that alter shared mutable data structures (e. g., various types of linked lists, binary search trees, and AVL trees). SL is best at reasoning about computer memory, especially random-access memory. However, SL cannot reason about CSSs, since it is based on a low-level storage model.

In the paper, we have extended SL to verify the correctness of management programs in CSSs. A modeling language is defined to describe the cloud storage management. The new commands mainly focus on block operations. Accordingly, its operational semantics is given by using the concepts of heap and store. Assertions in Separation Logic are extended to describe the properties of blocks in CSSs. Quantifiers over block variables and file variables are incorporated with ordinary SL assertions. Hoare-style rules are proposed to reason about the CSSs. Using these methods, an example of practical algorithm with while-loop is verified. The results show that the proving process is scientific and correct.

This work is supported by the National Key R&D Program of China under Grant Nos. 2018YFB1003904 and 2018YFC1314200, the National Natural Science Foundation of China under Grant Nos. 61772035, 61972005 and 61932001.