

**学士学位论文开题报告**

**论文题目**：B\*语言解释器的实现

**专 业**：计算机科学与技术

**本 科 生**：赵继伟

**学 号**：10061118

**指导教师**：马殿富

**北京航空航天大学计算机学院**

2014年3月19日

目录

[1 题目背景与意义 1](#_Toc382937790)

[1.1 课题来源 1](#_Toc382937791)

[1.2 选题的背景与意义 1](#_Toc382937792)

[2 研究现状 2](#_Toc382937793)

[2.1 国外研究现状 2](#_Toc382937794)

[2.1.1 VFiasco 2](#_Toc382937795)

[2.1.2 EROS/Coyotos 3](#_Toc382937796)

[2.1.3 Verisoft/Verisoft XT 3](#_Toc382937797)

[2.1.4 L4.verified/seL4 4](#_Toc382937798)

[2.2 国内研究现状 5](#_Toc382937799)

[2.3 小结 5](#_Toc382937800)

[3 理论模型 6](#_Toc382937801)

[3.1 目标模型 6](#_Toc382937802)

[3.2 程序行为模型 6](#_Toc382937803)

[3.3 验证模型 7](#_Toc382937804)

[3.3.1 宏观验证模型 7](#_Toc382937805)

[3.3.2 程序行为的表达 7](#_Toc382937806)

[3.3.3 算法验证模型 7](#_Toc382937807)

[3.3.4 数据结构验证模型 8](#_Toc382937808)

[4 研究内容 9](#_Toc382937809)

[4.1 研究目标 9](#_Toc382937810)

[4.2 研究核心内容 9](#_Toc382937811)

[4.3 预期研究效果 10](#_Toc382937812)

[5 技术路线 10](#_Toc382937813)

[5.1 基本研究流程 10](#_Toc382937814)

[5.2 解决方案 11](#_Toc382937815)

[5.2.1 编程语言选择 11](#_Toc382937816)

[5.2.2 语法分析器 11](#_Toc382937817)

[5.2.3 非线性语境的设计 11](#_Toc382937818)

[6 进度安排 12](#_Toc382937819)

[7 参考文献 13](#_Toc382937820)

# 题目背景与意义

## 课题来源

民用飞机专项科研项目“多核处理的机载实时操作系统关键技术研究”

## 选题的背景与意义

随着航空工业的不断发展，机载软件的规模、复杂程度日趋提高，并且是呈倍级增加的趋势。根据美国相关报道，美国空军新一代联合攻击机F-35的Final Block 3 Software计划约860万行代码[1]。根据相关资料，空中客车公司的历代飞机的在板软件大小（Volume of onboard software）也是逐渐提升，如下表[2]

|  |  |  |  |
| --- | --- | --- | --- |
| 飞机型号 | A310 | A320 | A340 |
| 在板软件大小（Mbytes） | 4 | 10 | 20 |

同时，在规模和复杂度提高的前提下，机载软件的安全性与可靠性都被做出了极高的要求。机载软件的标准于2011年从旧的DO-178B提升到了DO-178C[3],使得我们需要对于机载软件这类安全关键软件做出更加可靠的验证，对软件的质量提出了更高的要求。在DO-178C标准中，着重强调了采用形式化方法开展基于模型的开发和验证。

另外，于2009年提出的ISO/IEC 15408国际信息技术安全评估标准[4]，对于软件的形式化验证也提出了严格的要求。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CC级别 | 需求 | 功能规约 | 高层设计 | 低层设计 | 代码实现 |
| EAL1 | 非形式化 | 非形式化 | 非形式化 | 非形式化 | 非形式化 |
| EAL2 | 非形式化 | 非形式化 | 非形式化 | 非形式化 | 非形式化 |
| EAL3 | 非形式化 | 非形式化 | 非形式化 | 非形式化 | 非形式化 |
| EAL4 | 非形式化 | 非形式化 | 非形式化 | 非形式化 | 非形式化 |
| EAL5 | 形式化 | 部分形式化 | 部分形式化 | 非形式化 | 非形式化 |
| EAL6 | 形式化 | 部分形式化 | 部分形式化 | 部分形式化 | 非形式化 |
| EAL7 | 形式化 | 形式化 | 形式化 | 部分形式化 | 非形式化 |
| 完全验证 | 形式化 | 形式化 | 形式化 | 形式化 | 形式化 |

为了能够实现对于航空机载软件这类安全关键性软件的形式化验证，我们需要一种工具，可以将验证过程中机械化的过程交给软件自动完成，并且针对不可自动完成的部分，优化数据呈现模式，使得验证者可以更加清晰、方便的完成人工验证过程。最终达到可以加快形式化验证速度，提升形式化验证效率，提高形式化验证准确率的目标。

本人的毕业设计即是要完成上述目标，完成一个软件供形式化验证人员使用。

# 研究现状

## 国外研究现状

### VFiasco

该项目的主要目标是完成对于操作系统微内核Fiasco的形式化验证工作，工作周期为2001年至2005年。

该项目在起始阶段发现了在操作系统内核与一般应用软件之间的内存模型的不同，为解决在操作系统内核的内存管理的形式化验证问题，研究人员提出了设置一个高效内存范围不变量的研究方法，以期解决这个问题。

相较于内存模型方面，该项目最大的贡献在于提出了一种方法，来将内核的底层实现的C++代码，在定理证明器PVS**［5］**中建模，完成对于内核底层实现的形式化验证。这种方法避免了在定理证明器中直接定义C++的语义，绕开实现代码体系本身没有形式化验证支持的问题。该方法在其他的软件验证项目（LOOP项目**［6］**，Robin项目**［7］**）中也被采用了。

但是，该项目没有对于底层实现的比较重要的部分完成形式化验证**［8］**。

### EROS/Coyotos

该项目的主要目标是形式化验证操作系统EROS内核的安全模型，而后通过形式化的手段设计并实现一个新的操作系统内核，其工作的周期是从1999年至2009年。

该项目的研究人员对于EROS操作系统的内核的安全模型，用手工验证的方法完成了形式化验证。而后，为了设计新的操作系统内核，克服从前的内核形式化验证困难的问题，研究人员先行创造一种适合与形式化验证的编程语言——BitC。该项目于2006年完成了BitC的规格说明书以及BitC的创新提高部分。

然后，由于核心研究人员的离开，该项目始终没有发布内核形式化验证的成果，且与2010年以来再无更新。

### Verisoft/Verisoft XT

这是两个前后承接的项目，两者共同的目标是证明从硬件到应用软件的完整的计算机系统的形式化验证，存在普适的方法。其中Verisoft的工作周期为从2003年至2007年，而其后继项目Verisoft XT的工作周期为从2007年至2010年。

在Verisoft项目中，研究人员对一个完整的计算机操作系统进行了分层的形式化建模，如下图

|  |
| --- |
| Verisoft Email System |
| 图 2.1.3-1 Verisoft项目结构图［9］ |

研究人员的主要方法是首先采用手工证明的方法完成大部分形式化验证部分，然后将手工完成的证明输入定理证明器完成机械检查，保证验证的正确性。该项目中较为突出的特点是，研究人员不依赖于编译器以及硬件指令集的正确性假设，而是动手对于这些关键性部分进行形式化验证，是对于以往的研究项目中的一个很大的超越。该项目研究人员提出了一套有效地对于编译器的验证手段，即用完成形式化验证的编译器进行自编译。

在后续的Verisoft项目中，根据2010年该组研究人员所发表的论文，他们对虚拟化技术以及进程间通信的形式化验证做出了努力，并取得了成果。

可是，该项目为对于系统的性能以及兼容性完成研究，其硬件的兼容性存在普适性问题，另外该项目未对高层的安全策略完成验证。

### L4.verified/seL4

该项目的目标是完成一个经过机械化检查的，且通过形式化正确性证明的高性能的seL4微内核，其工作周期为从2005年至2008年。

这个项目总体上是由两个项目共同组成的，其中seL4是负责微内核的直接开放，而L4.verified是对于seL4的开发进行形式化验证，两者相辅相成，以期达到项目的总体目标。研究人员在验证过程中，将seL4微内核分成了3层，如下图

|  |
| --- |
|  |
| 图 2.1.4-1 L4.verified验证层次结构［10］ |

为了在开发过程中形式化验证变得更加方便，研究人员采用Haskell来进行底层设计的开发，因为Haskell这门语言，可以自动化的转换成定理证明器Isabelle的模型。而对于实际实现语言采用的是C语言，以保证seL4的性能。同时，为了保证C语言代码的正确性，在编写时，研究人员避免使用C语言中不安全的功能，同时通过建模转换，将C语言代码自动化的转换成定理证明器Isabelle中的模型，从而能够对C语言代码进行检查。

虽然这个项目属于当前比较成功的项目，但是还是缺乏C语言代码级实现的形式化验证。

## 国内研究现状

耶鲁大学及中科大-耶鲁高可信软件联合研究中心于2008年验证了一个较小的操作系统内核，该内核属于实验性系统，系统规模较小，功能比上文提到的seL4少很多。该研究中心在Certified OS Kernel［11］项目中，力求完成一个证明框架，可以使操作系统内核完成模块化验证，并且可以组装在一起形成完整操作系统内核。根据该组2010年更新在网站上的信息，他们完成了一个小型内核的协议，其中包括中断以及同步，同时，他们在线程上下文切换的验证方面有了很好的进展。

## 小结

目前近十年来，国外在针对操作系统内核的形式化验证方面已经做出了许多的成果，其中比较突出的是Verisoft/Verisoft XT与L4.verified/seL4两组项目，向着商业化和实用化已经迈出了一大步。然而，国内对于这方面的研究相对较少，没有对于一个实用的完整的操作系统内核完成形式化验证。而目前我国的航空工业发展迅速，日趋复杂的软件迫切需求形式化验证的支撑，国内应该重视对于操作系统内核，这一机载软件中的安全关键部分，完成形式化验证。

# 理论模型

## 目标模型

对于一个程序的规格描述，简单来说，可以分为高层规格描述和低层规格描述。其中，高层规格描述表示的是，程序要做什么，而低层规格描述表示的是，程序具体要如何做。本软件主要的目标在于解决低层规格描述的形式化验证问题，在后文中提到的程序的行为主要指的是在低层规格描述中，程序的具体的行为。

## 程序行为模型

对于绝大多数程序的行为，我们可以将其归纳为针对不同的输入完成不同的输出。根据不同的输入，我们可以将程序的行为分为两类

1. 功能性行为：在正确的输入的作用下产生输出的行为
2. 鲁棒性行为：在错误的输入的作用下产生输出的行为

而在形式化验证中，我们需要验证也就是程序的行为。

在蒙太古语用学中提到过，在不同语境中，相同对象有不同的语义。也就是说，对于每一段程序而言，它的行为不仅受到其自身的影响，还受到其语境——程序上下文的影响。

所以，我们可以将程序的行为，归纳为以下的一个表达式

其中表示作用于某一段程序的语境，下标k表示不同语境的编号；表示程序，可以是整个程序，也可以是某一段程序或者某一行程序；而R表示的是代表鲁棒性行为的逻辑语句，Q表示的是代表功能性行为的逻辑语句。

## 验证模型

### 宏观验证模型

既然在研究验证，就必须明确一个问题——什么叫验证通过。根据上面的对程序模型的理解，我们可以认为，对于绝大多数程序，其正确性即是根据输入得到了满足需求说明的正确的输出，其中的正确指的是程序只产生了正确的输出，没有产生多余的输出。

那么，在我们已知需求的前提下，形式化验证程序的正确性即转化为了确定由程序所建立的输入与输出之间的关系是否和需求确立的输入与输出之间的关系一致。

### 程序行为的表达

程序行为的表达是通过语言来实现的，为了使形式化验证更加便捷，我们选择了B\*语言作为低层规格描述（即本文中所说的程序的行为）的语言。这个语言的理论基础是一阶逻辑与集合论，十分适合对软件进行形式化验证，其基本语句的语法与C语言类似，而在数据结构方面仅有基本数据类型和集合。

### 算法验证模型

由于程序整体往往是庞大而复杂，因此我们无法直接对其直接进行验证，我们需要将程序化成小块逐一验证，并且以正确的形式统和到一起，以期完成对程序整体的验证。

程序是由一行一行的语句组成的，其中，主要可以分为三类语句

1. 顺序执行的语句，如赋值语句、加减乘除等等
2. 分支语句，即if语句
3. 循环语句

根据以上的语句分类，以及上文提到的程序行为模型，我们可以定义出如下的语境代换公理

1. 验证的初始状态为

终止条件为可以通过数学归纳法求出逻辑公式

### 数据结构验证模型

B\*语言中的数据结构仅有基本数据结构和集合，但是一般程序的数据结构却比B\*语言更加丰富，那么，为了验证其他的数据结构，我们需要完成一个数据结构与集合之间相互映射的关系

1. 线性表

定义：

(1)空表< >是线性表。

(2)若x0是元素，则< x0 >是线性表。

(3)若x0, ... , xn-1是不同元素，则<x0, … ,xn-1>是线性表。

线性表集合模型

(1){<nil, nil>}

(2){<nil, x0>, <x0, nil>}

(3){<nil, x0>, <x0, x1>, … , <xn-2, xn-1>, <xn-1, nil>}

1. 二叉树

定义：

(1)空是二叉树。

(2)若T是元素，, 是二叉树，并且，则<T,,>是二叉树。

其中，称为T的左子树，称为T的右子树。

二叉树集合模型

设bitree和bitree'是树，t1, t, t2是项，则

(1)<nil, nil, nil>bitree。

(2)若<nil, t, t2>bitree，则bitree - {<nil, t, t2>}{<t1, t, t2>, <nil, t1, nil>}bitree'。

(3)若<t1, t, nil>bitree，则bitree - {<t1, t, nil>}{<t1, t, t2>, <nil, t2, nil>}bitree'。

有关于一般的树和图的内容目前还在研究与探讨中。

# 研究内容

## 研究目标

根据上文的理论模型，完成一个形式化验证工具，将用B\*语言代码编写的低层规格描述，通过交互式的方式，实现逻辑语句的生成，完成整个形式化验证的过程，取代过去完全人工验证的方式，加快形式化验证速度，提高形式化验证准确度。

## 研究核心内容

为了完成研究目标，需要一个B\*语言的解释器，来自动化处理用B\*语言代码。然而，该项目需要的解释器不是一个传统的解释器，它核心目的是进行形式化验证，导出有关于低层需求的逻辑语句。所以，本解释器虽然在前端语法分析方面与传统解释器上相似，但是其后端的语言虚拟机部分却有很大区别。

传统解释器每个变量在某一个时刻只会存在一个值，也就是说整个语言的语境（在具体的程序中，指的是全部变量的值）的变化是线性的，每个分支语句（if）只会完成一种true/false的分支并执行。根据上文提到的算法验证模型的分支语句部分，本解释器需要对分支语句的每种运行可能进行检查，也就是语境的变化是非线性的，是一个树状展开的过程，这便是本解释器面临的第二个问题。

再者，传统解释器整个的解释过程是全自动不需要人工干预的，但是根据上文提到的算法验证模型，一旦遇到循环语句，其逻辑语句的导出方法是基于数学归纳法的，这个过程目前还是必须由人工来完成，所以说，如何在利用自动化的手段，让这个唯一的人工干预的过程变得更加快速而准确，便是本解释器面临的一个问题。

另外，根据上文中提到的数据结构验证模型，本解释器面临的第三个问题就是如何实现集合数据结构与线性表、二叉树，以及其他的数据结构之间的对等的映射关系。

综上所述，我们研究的核心内容，也是研究中的难点，为以下三点

* 非线性变化的语境
* 便利快捷的循环语句的逻辑语句导出
* 常用数据结构的集合化

## 预期研究效果

较完整的实现B\*语言语法树的功能，实现除了循环语句外其他的语句全部自动化处理，可以高效的对较复杂的以B\*语言描述的实际应用程序的低层规格描述完成形式化验证。

# 技术路线

## 基本研究流程

1. 完成B\*语言的语法分析器
2. 确定一个包含循环语句与分支语句，以及基本数据结构运算的典型B\*语言程序，完成这个程序的四元式表生成
3. 分析这个典型程序生成的四元式表，着力于解决循环语句处理与非线性语境这两个研究核心内容
4. 确定一个包含循环语句与分支语句，以及线性表数据结构运算的典型B\*语言程序，完成这个程序的四元式表生成
5. 分析包含线性表的四元式表，着力于解决常用数据集合化这个研究核心内容
6. 将研究线性表的方法推广至其他常用数据结构
7. 完整实现B\*语言语法树的全部功能

## 解决方案

### 编程语言选择

为应对跨平台的需求，我们选择Java作为开发语言

### 语法分析器

我们选择了Java平台中的Antlr4.1语法分析器生成器作为B\*语言语法分析器的实现工具

### 非线性语境的设计

目前，我们已经完成了对于研究核心内容中非线性语境的解决方案。

|  |
| --- |
|  |
| 图5.2.3-1 条件树部分 类图 |
|  |
| 图 5.2.3-2 条件树更新伪代码 |

该方案的基本数据如图5.2.3-1所示，通过一个条件树来保存代码的分支框架，在条件树的每个叶节点中保存语境，来保证条件与语境相匹配。该方案的以伪代码形式表示的算法如图5.2.3-2所示，给出了更新条件树的一些基本的操作。

### 循环语句设计

目前可以给出的是关于处理循环语句的流程图

|  |
| --- |
|  |
| 图 5.2.4-1 循环语句处理流程图 |

# 进度安排

|  |  |
| --- | --- |
| 时间 | 任务 |
| 2014.03.03---2014.03.17 | 阅读相关材料，准备开题 |
| 2014.03.18---2014.04.13 | 完成软件主体部分 |
| 2014.04.14---2014.04.18 | 中期检查 |
| 2014.04.19---2014.04.31 | 调整软件内容，最终完成软件本体 |
| 2014.05.01---2014.06.02 | 完成毕业论文 |

# 参考文献

1. Warwick, Graham. "Flight Tests Of Next F-35 Block Underway." Aviation Week, 12 June 2010.
2. Jean-Louis Camus. “Efficient Development of Airborne Software with SCADE Suite™.” Handbook. 2003
3. <http://www.rtca.org/store_list.asp>
4. ISO/IEC 15408
5. S. Owre, J. M. Rushby, and N. Shankar, PVS: A Prototype Verification System, from CADE 11, Saratoga Springs, NY, June 1992
6. Joachim van den Berg, Bart Jacobs, “The loop Compiler for Java and JML.” Tools and Algorithms for the Construction and Analysis of Systems Lecture Notes in Computer Science Volume 2031, 2001
7. Hendrik Tews, “Formal Methods in the Robin project: Speciﬁcation and veriﬁcation of the Nova microhypervisor.” Proceedings of the C/C++ Verification Workshop, 2007
8. Gerwin Klein. “Operating System Verification—An Overview.” Sadhana, February 2009
9. http://www.verisoft.de/SubProject2.html
10. Gerwin Klein, etc. seL4: Formal verification of an operating system kernel. Communications of the ACM, 53(6), 107–115, (June, 2010)
11. http://kyhcs.ustcsz.edu.cn/projects/certos