****



**研 究 生 毕 业 论 文**

**（申请硕士学位）**

**论文题目** 循环不变式生成方法研究

**作者姓名** 刘自恒

**学科、专业名称**  计算机软件与理论

**研 究 方 向**  信息安全

**指导教师** 曾庆凯 教授

二零一二年 五月 二十日

**学 号：MG0933021**

**论文答辩日期： 年 月 日**

**指 导 教 师： （签字）**

**Nanjing University**

(Master Degree Dissertation)

**Research on Static Software Security**

**Flaw Detection**

By

**Zi-heng Liu**

Supervised by

Professor **Qing-kai Zeng**

Department of Computer Science and Technology

Nanjing University

May 2012

Nanjing, P.R.China

# 摘要

软件作为信息系统的实现载体，广泛应用在各个领域，软件中的任何安全漏洞或实现错误都可能导致非常严重的后果。解决软件安全问题的根本在于软件交付运行之前成功消除所有潜在的安全漏洞，提高软件质量。因此，研究如何有效分析和验证软件，找出软件安全漏洞，成为信息安全领域的一个重要研究课题。本文主要针对软件验证技术中的循环不变式生成技术进行研究和实现：

（1）对形式化方法进行总结和研究。本文依据代码检测层次的不同对模型检测技术进行分类。该分类方法可以较好的将目前模型检测工具进行有效归类，同时分类的结果不存在二义性。在此分类的基础上，分析和总结了各种模型检测技术并介绍了每一类的代表性工具。

洞。

**关键字：验证、循环不变式、值分析、数据流分析、指针别名分析**

# Abstract

With the quick development of information technology, software has been used widely in politics, economics, science, national defense and medicine, and any vulnerability in software will cause serious result. The final solution to address the safety problem is to eliminate all potential vulnerabilities before software is released and to improve software quality. Therefore, the research on software analysis and verification to find out vulnerabilities efficiently has become a hot research topic in the field of information security. In this paper, we will do further research on model checking techniques, which is one of software analysis techniques.

cost.

**Keywords：static analysis、model checking、dataflow analysis、alias analysis、counterexample path verification**

# 目录

[摘要 I](#_Toc317862746)

[Abstract II](#_Toc317862747)

[目录 III](#_Toc317862748)

[图片目录 IV](#_Toc317862749)

[表格目录 IV](#_Toc317862750)

[第一章 引言 1](#_Toc317862751)

[1.1 研究背景 1](#_Toc317862752)

[1.2 研究内容 2](#_Toc317862753)

[1.3 本文组织 3](#_Toc317862754)

[第二章 形式化方法基础 4](#_Toc317862755)

[第三章 循环不变式生成技术研究 5](#_Toc317862756)

[第五章 开发平台介绍 6](#_Toc317862757)

# 图片目录

[**图 1.1 CVE 数据库近10年来的脆弱性数量统计** 1](#_Toc317862373)

[**图 5.2 Frama-C结构图** 7](#_Toc317862374)

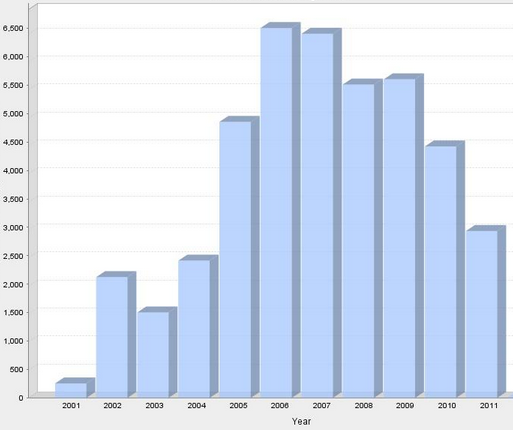
# 表格目录

[**表 6.1 比较** 8](#_Toc317860539)

# 第一章 引言

## 研究背景

随着计算机科学的迅速发展和信息化时代的深刻变革，信息系统已经深入到人类生产生活的各个领域，软件作为信息系统的实现载体，广泛应用在各个领域，软件的任何漏洞都可能导致非常严重的后果。无论是软件设计人员还是软件的用户，均希望在软件系统投入正式使用之前，能得到软件正确性的保证，避免造成软件后期维护成本的上升和使用有缺陷软件造成的损失。软件的安全性一直是系统安全性、可靠性、稳定性的一个重要方面。已经有很多系统由于软件缺陷的存在，造成了巨大的损失，CVE（[cve.mitre.org](http://cve.mitre.org/)）不断发布最新的软件安全漏洞报告，由图 1.1（[nvd.nist.gov](http://nvd.nist.gov/)对CVE数据的统计）可以看出，软件的脆弱性形势一直十分严峻，软件的安全性和可靠性一直困扰着软件开发和使用人员。



**图 1.1 CVE 数据库近10年来的脆弱性数量统计**

另一方面，软件的实现是否和设计需求和规范是否一致决定了软件的正确性，不正确的软件开发实现同样会带来重大损失。

首先用来试图得到无错误软件的方法是测试；通过在程序上运行一组输入集合并检查软件的行为是否符合预期；对于关键性软件，必须达到很高的测试覆盖度才可以获得相对较高的可信度，但是这样的测试输入有时候甚至是无穷的，因而代价很高[[Mauborgne 2004](#_ENREF_9)]。

为了最大限度地保证软件的正确性和无脆弱性，从另一个角度考虑，如果可以从理论上证明程序符合预期的行为或者没有某种错误，同样可以认为软件是可靠的。这需要开发人员对已编码软件的正确性进行分析证明，同时证明软件中没有某些导致软件安全缺陷的代码实现。正确性是指代码的实现和设计的算法目标一致，代码的执行与算法的预期是相同的，代码没有错误地实现算法。没有安全缺陷是指代码的实现过程没有引入可能导致软件崩溃、被其他恶意软件利用篡改等的部分。

证明的过程需要对算法、编码和安全漏洞有着深刻的认识，需要严密的逻辑演算过程，同时也会有一些琐碎的证明过程。这样的过程如果全部由人工来完成，无疑需要大量的人力、物力、财力，而且人工实施难免会出差错。

出现了形式化的软件脆弱性分析方法，以严格的数学和逻辑理论为基础，自动地帮助分析人员发现其它方法不容易发现的系统描述不一致性或脆弱性，增加软件开发人员对系统的理解，实现开发可靠的软件产品的目标。形式化方法并不排斥测试，但可以认为是一种更好的测试框架[[Airchinnigh 1995](#_ENREF_1)]。形式方法的主要研究内容包括:形式规范说明 (formal specification)和形式验证(formal verification)。基于模型的；基于公理系统和过程代数的

定理证明 (theorem proving)是一种形式验证方法，它是根据已构造的规格说明生成反映该规格说明应具有的性质，将其表示成定理形式，并加以证明，从而达到对系统规格说明验证的目的。形式化的方式主要思路是对程序源代码、规范说明进行建模、分析、判断，可以不需要或少需要人工干预。

## 研究内容

本文主要包括研究内容如下

（1）阐述了形式化方法的主要理论基础，总结了现有的形式化方法的主要种类。

主要从前置条件、后置条件、不变式、状态机等方面阐述，比较了常用的形式化方法的特点、适用范围、典型工具、局限等内容。

（2）分析、总结、比较了现有的生成循环不变式的方法。

本文列出了目前使用的方法的针对情形，指出了用到的关键算法，指出了各自的特点和不足，并提出了可能的解决方法。

（3）针对数值型计算程序和含有数组的程序，基于条件赋值转换和自适应模版生成技术，提出了改进的循环不变式生成方法。

（4）基于Frama-C和APRON库，将上述方法实现为Frama-C的一个插件，自动地完成循环不变式的生成和插桩工作。

（5）选取若干样例程序，与其他不变式生成工具做对比。选取若干实际使用的程序，计算其中的不变式，并人工统计分析生成的不变式的质量。

## 本文组织

本文的组织结构如下：

第二章阐述了形式化方法的主要理论基础，给出了目前常用的形式化方法，

第三章探讨生成循环不变式的方法。

第四章提出了一种自动生成循环不变式的方法，该方法基于条件赋值转换和自适应模板生成技术，自动化程度高，生成尽可能多的不变式。

第五章基于Frama-C和APRON库，将上述方法加以实现，成为Frama-C的一个插件。

简要介绍了Frama-C的主要功能和特点，介绍了APRON库的作用和效果。遵从Frama-C的开发要求，设计了一个Frama-C的插件。描述了该插件的主要组成、工作原理和实现的主要算法。

第六章选取了一些样例程序，将上述方法和现有方法的效果加以对比；选取了一些实际中使用的程序，对生成的结果进行分析，评价生成的不变式的质量。

第七章总结了目前的工作，并对下一步的研究方向进行了展望。

# 第二章 形式化方法基础

形式化证明的主要方法有。模型检测。定理证明。谓词抽象。

形式化证明的一个关键难点在于程序中循环结构的存在。归纳不变式。目前对循环的处理方法主要有。

什么是不变式

# 第三章 循环不变式生成技术研究

本文主要针对程序中存在的循环结构进行分析，生成其中存在的循环不变式，以辅助完成后续的程序属性证明过程。本章从几个具体程序出发，根据其具体特点，分析了现有的生成循环不变式的方法，指出了它们的优缺点。在现在方法的基础上，有针对性地提出了改进目标。

针对线性算术式的不变式模版求系数法；[[刘自恒报告20110301, Gulwani et al. 2008](#_ENREF_11); [Gupta and Rybalchenko 2009](#_ENREF_8)]由用户指定不变式模版（程序有默认的可以使用），是基于数值约束求解[[Colon, Sankaranarayanan et al. 2003](#_ENREF_3)]的。对于此，用户指定的模板形式固定，适应性不强，可以通过要证明的规范或断言，启发式地得到可能的不变式模板形式，然后逐一进行约束求解。约束求解库可以使用APRON，定义的抽象域包括Box,Oct，NewPolka，PPL等，不同的抽象域求解的效率和精度不同。[[Gupta and Rybalchenko 2009](#_ENREF_8)]将自己集成到Blast中，使Blast的分析效果得到提升。

Interproc使用APRON和fixpoint库实现了一个简单语言程序的不变式生成，fixpoint是按照[[Gopan and Reps 2007](#_ENREF_7)]中的思想实现的，不同的转换系统实例可以得到不同的分析结果，[[Gopan and Reps 2007](#_ENREF_7)]中给出了两个转换系统实例。可以将Interproc看作一个抽象解释器，invGen即在前端使用了Interproc作为解释器，并跟踪它生成的不变式。

多阶段循环的展开法(循环展开，控制流精化)或条件赋值法；当循环体中有if语句时，程序的路径有多条，由于循环控制变量的变化，可以把循环看成一个多阶段的执行过程。可以将循环体做语句展开，做执行路径的具体化，或者将if控制下的语句转换为条件赋值语句[[刘自恒报告20110426, Kovács et al. 2009](#_ENREF_12)]，或者通过分割谓词将一个循环展开为多个基本形式的循环[[Sharma, Dillig et al. 2011](#_ENREF_10)]。[[Beyer, Henzinger et al. 2007](#_ENREF_2)]将这样的循环连同其他部分看做多个路径程序分别分析。将if条件直接转换到不变式中，同时做相应的变量转换，类似[[Furia and Meyer 2010](#_ENREF_6)]，之前黄老师说不加改变的是无意义的，因为在推理的时候，推理证明器可以根据if条件进行推理。

将数组根据状态划分为小数组的分割法；有数组参与时，可以将所有数组元素用互不同的变量代替，也可以用一个变量代替所有元素，或根据循环的执行进程，将数组分为不同状态的子数组[[刘自恒报告20110426, Kovács et al. 2009](#_ENREF_12); [Cousot, Cousot et al. 2011](#_ENREF_4)]，根据状态得到数组变量的不变式性质。有字符串操作的语句也可以看成对数组进行操作，一般通过系统函数如memcpy等完成，这时候可以对系统函数加上规范说明得到关于字符数组的不变式性质。

[[Flanagan and Qadeer 2002](#_ENREF_5)]使用了预定义原子谓词来辅助程序证明过程，使用谓词抽象技术猜测后面可能使用到的谓词。

# 第五章 生成插件的设计与实现

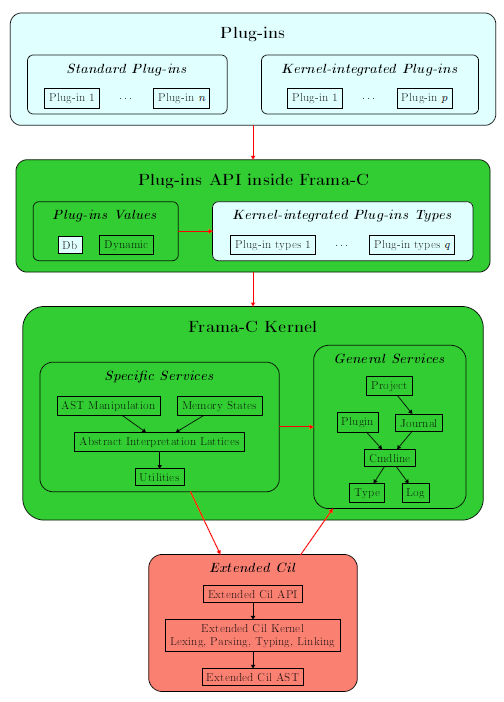
ACSL

ACSL的全称是ANSI C Specification Language，是一种在Frama-C框架下实现的行为接口类规范描述语言（Behavioral Interface Specification Language，BISL），用来描述C语言代码的行为属性。不同于JML（Java Modeling Language）既用于动态检测断言又用于静态验证，ACSL用于静态验证和归纳验证。

更多内容见<http://frama-c.com/acsl.html>。

Frama-C

Frama-C是一个执行C程序静态分析的开发平台工具，收集了多种静态分析技术，将它们作为插件整合在一个统一的框架中，并对外提供开发接口，允许用户使用既有插件的结果快速开发自己的功能插件。**图 5.2**是Frama-C的结构设计图。



**图 5.2 Frama-C结构图**

APRON

获取循环中可能的变量值更新变化的步长集合

获取循环中更新变量值的常数集合

获取循环之后程序中存在的判断表达式集合Adj

提取Adj中的变量集合。通过插件PDG进行依赖关系计算，得到循环中使用每个变量的结点集合。提取其中的常数，作为步长候选值。

根据循环中用的变量，生成模板，使用wp验证是否满足，不确定时加入转换系统，由APRON再次检查。

成立的不变式用ACSL注释表示，加入程序中，输出得到分析后的含有循环不变式的程序。使用WHY等工具来验证程序中的属性是否可以得到满足。

**表 6.1 比较**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

由于理论尚不够成熟，形式化方法还难以证明大型软件系统的正确性[[Airchinnigh 1995](#_ENREF_1)]。希望我们的工作有助于形式化证明的发展。

Airchinnigh, M. a. (1995). Formal Methods & Testing. Tutorials of the Sixth International Software Quality Week, 625 Third Street, San Francisco, CA 94107-1997.

Beyer, D., T. A. Henzinger, et al. (2007). Path invariants. PLDI '07, ACM New York, NY, USA ©2007.

Colon, M., S. Sankaranarayanan, et al. (2003). Linear invariant generation using non-linear constraint solving. CAV'03.

Cousot, P., R. Cousot, et al. (2011). A parametric segmentation functor for fully automatic and scalable array content analysis. POPL'11, ACM.

Flanagan, C. and S. Qadeer (2002). Predicate Abstraction for Software Verification. POPL '02.

Furia, C. A. and B. Meyer (2010). Inferring Loop Invariants Using Postconditions, Springer-Verlag Berlin, Heidelberg ©2010

Gopan, D. and h. Reps (2007). Guided Static Analysis. 14th International Static Analysis Symposium, Kongens Lyngby, Denmark.

Gupta, A. and A. Rybalchenko (2009). InvGen: An Efficient Invariant Generator. CAV '09.

Mauborgne, L. (2004). Astrée: verification of absence of run-time error. Building the Information Society, Kluwer Academic.

Sharma, R., I. Dillig, et al. (2011). Simplifying Loop Invariant Generation Using Splitter Predicates. CAV'11.

刘自恒报告20110301, S. Gulwani, et al. (2008). Program analysis as constraint solving. PLDI '08, Tucson, Arizona, USA, ACM New York, NY, USA ©2008.

刘自恒报告20110426, L. Kovács, et al. (2009). Finding Loop Invariants for Programs over Arrays Using a Theorem Prover. FASE 2009, Springer-Verlag Berlin Heidelberg 2009.