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，计算机已经深入到人类生产生活的各个领域，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。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[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。

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

本文的研究内容如下

对形式化方法的主要种类和理论基础进行了总结

对现有的生成循环不变式的方法进行了比较，

本文组织

本文的组织结果如下

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

现有方法总结

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。

主要开发平台介绍

ACSL

Frama-C

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。

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.