模型验证软件的归纳推理算法的设计与实现

摘 要：随着软件和硬件系统的日益复杂化，确保这些系统的可靠性和安全性的需求也不断增长。领域特定语言（DSL）例如 L2C，因其能够提供更贴近特定应用领域的抽象而受到青睐。然而，这些语言的特殊性也带来了验证和验证工具可用性的挑战。Lustre 语言，作为一种在安全关键系统领域广泛使用的同步数据流语言，通过其严格的形式化语义提供了一个强大的平台，用于实现这些系统的模型验证。因此，将 L2C 代码转换为 Lustre 代码，不仅有助于提高系统的可靠性，还可以利用 Lustre 的强大验证工具，如 Kind2。本文综述了当前关于 L2C 到 Lustre 转换与验证的研究，特别是归纳推理算法在模型验证中的应用。同时描述了基于 CMake 开发的可在 Windows 环境下运行的 Lustre 验证工具 QKind 的基于 K-induction 算法开发模型验证模块内容编写。

关键词：领域特定语言；L2C；Lustre；归纳推理；模型验证；K-induction；

Design and implementation of inductive reasoning algorithms for model verification software

Abstract: As software and hardware systems become more complex, the need to ensure the reliability and security of these systems grows. Domain-specific languages (DSLs) such as L2C are favored for their ability to provide abstractions that are more relevant to specific application domains. However, the specificities of these languages also present challenges in verification and the usability of verification tools. The Lustre language, as a synchronous data flow language widely used in the field of safety-critical systems, provides a powerful platform for implementing model verification of these systems through its strict formal semantics. Therefore, converting L2C code to Lustre code not only helps improve the reliability of the system, but also takes advantage of Lustre's powerful verification tools, such as Kind2. This article reviews current research on L2C to Luster conversion, especially the application of inductive inference algorithms in model verification. At the same time, it describes the writing of the model detection module based on the K-induction algorithm developed by QKind, a Luster verification tool developed based on CMake that can run in the Windows environment.

Key words: Domain-specific language; L2C; Lustre; inductive reasoning; model verification; K-induction

目 录

第一章　绪论

1.1　课题的研究背景

1.1.1 同步数据流语言研究现状

同步数据流语言（如 Lustre、Signal）已在航空、高铁、核电等安全关键领域得到广泛应用[1]。举例来说，适用于这些领域实时控制系统建模和开发的 Scade 工具都是基于类似 Lustre 语言的。这类语言相关的开发工具，尤其是编译器的安全性问题备受关注。近年来，采用形式化验证实现可信编译器的研究成为程序设计语言领域的焦点之一，取得了显著成果，例如 CompCert 项目成功实现了产品级的可信 C 编译器[2]。类似地，人们也采用这种方法开展了同步数据流语言可信编译器的研究工作。

在现实中，应用于核领域中的软件往往对于高性能计算和数据处理任务的要求比较高。而随着核研究领域中的科学计算和工程应用的复杂性不断增加，开发人员对于并行计算和数据流处理的需求也日益提高。而我们通过调查发现，传统的并行编程模型往往需要开发人员深入理解硬件架构和并发控制原理，并且手动管理软件的并发性和同步，这增加了核领域软件开发的复杂性和错误率。然而，随着近年来市场上大数据和人工智能等技术的快速发展，数据密集型应用的需求也越来越多，这些应用对于高效的数据处理和并行计算的要求也越来越高。

就像在核领域中，我们发现有些应用对实时性要求较高，比如控制系统和数据采集系统，这使得我们需要更高效、更可靠的编程范式。而当下许多研究人员通过研究发现，运用同步数据流语言为解决这些棘手的问题提供了一种全新的思路。同步数据流语言能将计算视为数据流的传递和转换过程，这样的话使用者只需关注数据流之间的依赖关系而不用显式地管理并发和同步，从而简化了开发过程，减少了出错的可能性，并有助于优化软件的并行性和性能。虽然当今市面上已经有一些同步数据流语言模型被应用于核领域，但在实际应用中仍然存在一些困难。例如该如何优化编译器以提高代码的执行效率，或者研究人员该如何进行

有效的调试和测试从而确保程序的正确性等。但我们可以确定的是，同步数据流语言在核领域的软件开发中一定具有极大的潜力。

1.1.2 模型验证软件现状

随着科技的发展，各种复杂的软硬件系统在人们的日常生活和社会的关键行业中扮演着愈发重要的角色。如交通控制、医疗监护、航空航天及自动化制造等系统均承担着至关重要的任务，其可靠性和安全性直接关系到人们生活与财产安全。因此，如何验证和保障这些系统的正确性已成为研究与工业界共同关注的焦点。

领域特定语言（DSL）因具备为特定领域提供定制化抽象和表达方式的特性，在复杂系统开发中扮演着愈发关键的角色。L2C 作为一种基于 Lustre 语言改良的 DSL，以其简洁高效的语言特性，使得开发者能够迅速实现系统功能，同时确保代码的可读性与可维护性。然而，随着系统复杂度的攀升，单纯依赖人工审核代码以确保系统正确性已变得愈发困难，这凸显了自动化形式化验证工具的必要性。

Lustre[3,4,5]语言，作为一种成熟的同步数据流语言，最早出现在 P.Caspi 的论文中，多用于嵌入式控制系统和信号处理系统[6]。其已在实时和安全关键系统开发中得到广泛应用[7]。其强大的形式化语义为系统模型验证提供了理想平台。可是，当前市面上并没有针对 L2C 语言的解析验证工具，因此开发一款软件来对 L2C 语言进行解析与验证便成了当下的一大需求。

研究人员在实际应用中发现，虽然将 L2C 语言转化为 Lustre 语言的确带来了许多好处，但是转换过程本身就充满挑战，尤其是在确保转换后代码与原始代码在逻辑上的一致性上开发人员们遇到了不少问题。此外，由于系统的多样性和在实现过程中的各种细节，导致开发一个通用且高效的验证工具实际上极具困难。这不但要求我们在转换过程中必须要使用更好的技术与方法，如归纳推理算法，来确保转换过程的正确性与有效性。

同时，Kind2 软件在运行环境等方面存在一定局限性，为开发者带来诸多不便。因此，开发一款更加实用的 Lustre 验证软件以满足当前市场需求显得尤为重要。

1.2 市面存在的模型验证软件相关研究

1.2.1 模型验证软件现状

模型验证作为形式化方法中的核心环节，其重要性不言而喻。随着计算机科学和工程学的不断进步，模型验证技术已成为确保系统正确性和可靠性的关键手段。当前，市场上存在众多模型验证工具和软件，它们各具特色，以满足不同领域和场景的验证需求，例如张智慧等人开发的用于核电站控制保护的主要用于核安全级控制算法描述语言 G-Lustre 编译器[8]。本文旨在深入探讨模型验证软件的现状，将针对 Kind2 软件展开详细分析。

Kind2 是一款开源的模型验证工具，专注于连续和混合系统的验证[9]。其独特之处在于，它集成了多种验证技术，如抽象解释、SMT 求解器以及模型验证等。这些技术的融合使得 Kind2 在处理复杂系统模型时具有更高的灵活性和适应性。同时，得益于优化的算法和数据结构，Kind2 能够应对大规模和复杂的系统模型，展现出卓越的验证效率。

除了功能强大和高效性能，Kind2 还提供了丰富的 API 和插件机制，为用户提供了扩展功能和定制验证流程的便利。这意味着用户可以根据自身需求，灵活地定制和扩展 Kind2 的功能，从而更好地满足特定领域的验证需求。

此外，Kind2 还具备直观的图形用户界面和命令行接口，使得用户无论是初学者还是专业人士，都能够轻松地进行模型验证和分析。这种友好的交互设计，无疑进一步增强了 Kind2 在实际应用中的易用性和普及性。

同样，我们在研究后发现，正因为 Kind2 的功能强大、性能高效，所以它在航空航天、自动驾驶汽车、医疗设备等关键领域都得到了广泛应用。例如在自动驾驶汽车中 Kind2 就可以可用于验证汽车控制算法的正确性和可靠性从而保证车辆在各种道路条件下自动驾驶的安全性。这极大的推动了自动驾驶汽车在提高了安全性和可靠性后的普及工作。

综上我们可以发现 Kind2 作为一款领先的模型验证工具，凭借其强大的功能、高效的性能以及友好的用户界面使其在各个领域都有着广泛的应用。随着科技的不断进步我们有充分的理由相信，Kind2 将在未来扮演更为重要的角色并且为系统验证领域带来更多的创新和突破。

1.2.2 模型验证软件中归纳推理算法的使用

随着模型验证技术的不断革新，归纳推理算法在现代模型验证领域中的应用愈发广泛。这类算法以其高效和精确的特点，为复杂系统模型的验证提供了强有力的支持。特别是在开源模型验证工具 Kind2 中，归纳推理算法的应用尤为突出，对于连续和混合系统的验证具有显著效果。

我们发现在 Kind2 软件中，归纳推理算法被应用于检测代码的环路不变性、定界分析、健壮性评估以及系统模型的不变性证明等方面。而正是这些功能使得 Kind2 能够深入探索系统模型的内在逻辑和特性从而为验证过程提供坚实的理论支撑。不过值得一提的是，Kind2 软件在同样在算法和数据结构方面也进行了优化从而确保其在进行大规模数据处理时的高效性和准确性。

同样以自动驾驶汽车为例，Kind2 的归纳推理算法被广泛应用于检测和验证控制算法的环路不变性。这都能有效确保自动驾驶汽车在各种复杂道路条件下的安全性和稳定性。

可以预见的是，随着模型验证技术的不断进步，归纳推理算法在现代模型验证软件中的应用将越来越广泛。而 Kind2 软件作为其中的佼佼者，其高效、精确和灵活的特点将为其在更多领域的应用提供有力支持，为推动相关行业的发展和进步发挥重要作用。

1.2.3 模型验证软件的缺陷与不足

尽管目前市面上的各种模型验证软件在提供高效、精确的验证能力等方面都已经有了显著的进步，但还是仍然存在一些明显的缺陷和不足。例如 Kind2 尽管具有强大的验证能力，但在处理某些非线性和高度复杂的系统模型时，其性能和效率可能会大大降低，这主要是由于归纳推理算法在处理复杂性较高的系统模型时可能会遇到困难，导致验证过程变得非常耗时。此外，Kind2 虽然提供了图形用户界面和命令行接口，但对于非专业用户来说，操作和配置过程可能相对复杂，这些不足都限制了广大工程师和研究人员在软件开发时的使用意愿和效率。并且 Kind2 在处理其他类型的系统模型，如并发系统和分布式系统时，也存在一定的

局限性。除了 Kind2 以外，市面上一些其他的知名的例如 NuSMV、Spin 和 Uppaal 等的模型验证软件同样也存在一定的问题，例如在处理大规模系统模型时可能会遇到性能瓶颈问题，因而导致验证过程变得非常耗时。这些软件在个别验证任务或特定应用领域中功能不足，需要进一步的算法和技术优化。以及在特定验证需求时可能存在一定的局限性。

除此之外，Kind2 这款模型验证软件在运行环境上的局限性显而易见。它目前仅能在 Linux 系统上稳定运行而在 Windows 环境下则无法顺畅运行。这一限制不但给众多采用 Lustre 及其相关语言编写的模型开发者带来了诸多不便，而且在某种程度上也制约了这些模型在跨平台应用中的广泛推广和实际应用。

再者，随着信息技术的发展，跨平台兼容性已经成为衡量软件质量的一项重要指标。一个能在不同操作系统中无缝切换运行的软件，往往能够获得更广泛的用户基础和市场份额。而 Kind2 目前在这方面的表现显然不够理想，这无疑也限制了其在市场中的竞争力和影响力。

综上我们发现，开发一款能在 Windows 和 Linux 两大主流操作系统中均能顺畅运行的 Lustre 语言模型验证软件显得尤为迫切和必要。这样一款软件不仅能有效解决当前 Lustre 模型开发者在跨平台应用中所面临的困境，也能为模型的广泛推广和实际应用提供有力支持。

1.3 本文的主要贡献

在当今科技高速发展的时代背景下，模型验证软件在多个领域，尤其是核动力研究领域扮演了不可或缺的角色。但市场上现有的模型验证软件普遍存在一定的局限性，无法充分满足相关领域开发者日益增长的需求。所以我们针对中国核动力研究设计院的具体需求，开发了一款名为 Qkind 的模型验证软件。

Qkind 软件的设计初衷在于，开发一款在 Windows 和 Linux 环境下，对使用同步数据流语言 L2C 编写的模型进行量词约束安全性属性的验证的软件。这得益于 Qkind 软件独特的架构设计和创新的算法应用。在 Qkind 中，模型的量词约束安全性属性通过独特的注解语言进行表达，以不变式或"假设-保证"风格契约的形式展现，这种表达方式既清晰直观，又为用户提供了极大的便利。

Qkind 软件的核心在于形式化方法的应用。它能够对系统的行为进行精确分析，并将 L2C 程序和约束属性转化为 符号系统。在此基础上，Qkind 借助 SMT 求解器进行验证，从而确保模型的正确性和安全性。值得一提的是 ，Qkind 软件在模型验证模块的设计上，充分借鉴了市场上已有的基于 K-induction 算法[5]的模型验证设计。这一 算法在模型验证领域具有广泛的应用，能够高效处理复杂模型的验证问题。通过结合 K-induction 算法，Qkind 的模型验证模块在性能和效率上得到了显著提升，为用户提供了更加可靠和高效的验证服务。

总体而言，Qkind 软件的开发是基于对市场上现有模型验证软件的深入研究，以及对中国核动力研究设计院具体需求的精准把握。通过运用形式化方法和 K-induction 算法，Qkind 软件在模型验证领域展现出强大的实力和广阔的应用前景。我们有理由相信，随着 Qkind 软件的不断完善和优化，它将在核动力研究以及其他领域发挥更加重要的作用。在当今的软件开发领域，自动化验证和模型验证已成为确保软件质量的重要手段。Qkind 软件作为一款前沿的模型验证工具，其整体框架与功能设计得相当完备。本文将先简要概述 Qkind 的整体架构及其主要功能，然后重点探讨其中运用 K-induction 算法完成的模型验证模块内容。

1.4 本章总结

本章深入阐述了研究的背景以及领域特定语言（DSL）在多个领域中的关键作用，特别是在复杂系统开发中对 L2C 和 Lustre 语言的运用。同时对目前市场上存在的模型验证软件进行了细致的分析，探讨了这些软件所使用的归纳推理算法及其存在的局限性。在此基础上，本文的主要创新点在于开发了一款基于归纳推理算法的 Lustre 验证工具，旨在提升现有工具的效能和性能，为相关领域的研究和实践提供更为强大和高效的支持。

第二章 系统开发相关技术与工具

2.1 系统涉及语言与工具

2.1.1 Lustre 语言

Lustre 语言是一种用于并行和分布式计算的声明式编程语言，适用于各种数据流编程模型。其最初由法国国家研究中心 INRIA 开发后被用于实时系统和嵌入式系统的开发。Lustre 语言的主要特点是其清晰、数学化的语法和语义，这些特点使得它成为高可靠性、高可预测性的应用的开发过程中的理想语言，尤其是在各种需要确保正确性和可维护性的应用的开发中。在 Lustre 语言里程序由数据流方程（equations）组成，并且由这些方程共同定义了系统的状态随时间的变化而产生的变化。这种声明式的编程范式不但有助于提高程序的可读性和可维护性，同时也方便进行各种形式化验证，以确保程序的正确性与可靠性。

2.1.2 L2C 语言

L2C 语言，全称 Lustre to C Language。是由中国核动力研究设计院研发的同步数据流语言，它允许将 Lustre 程序转换为 C 语言代码。它以 Lustre 语法为基础，通过增添一系列实用的语法功能，进一步丰富了 Lustre 的表达能力。相较于传统的 Lustre 语言，L2C 语言在现实工作中的适用性得到了显著增强，能够更好地满足复杂系统的建模与仿真需求。

2.1.3 QKind

Qkind 是一款由中国核动力研究设计院委托南华大学计算机学院研发的模型验证软件，其主要设计目标是借助 SMT 求解器来对使用同步数据流语言 L2C 编写的模型进行量词约束安全性属性的证明或反证。在此过程中，这些属性将被提取，然后以不变式或"假设-保证"契约的形式进行表示。为了实现这一目标，Qkind 将采用形式化方法分析系统行为对 L2C 程序进行解析，随后再利用 SMT 求解器进行验证。

2.2 系统主要开发技术

2.2.1 CMake

CMake 是一个跨平台的安装（编译）工具，可以用简单的语句来描述所有平台的安装(编译过程)[10]。他可以完成各种的 makefile 以及 project 文件的输出，并且可以测试编译器所支持的 C++特性,就像 UNIX 下的 automake。只是 CMake 的组态档取名为 CMakeLists.txt。同时，Cmake 并不会直接建构出最终的软件，而是会产生标准的建构档（如 Unix 的 Makefile 或 Windows Visual C++ 的 projects/workspaces），然后再根据一般的建构方式使用。这让熟悉某个集成开发环境（IDE）的开发者可以用标准方式建构自己的软件，这种允许使用各平台的原生建构系统的能力是 CMake 和 SCons 等其他类似系统的区别之处。Cmake 在实际开发中有着开放源代码、跨平台、能够管理大型项目、能简化编译构建过程和编译过程、高效率、可扩展等一系列优点[11]。

2.2.2 ANTLR

ANTLR（全名：ANother Tool for Language Recognition）是基于 LL(*)算法实现的语法解析器生成器（parser generator），用 Java 语言编写，使用自上而下（top-down）的递归下降 LL 剖析器方法。由旧金山大学的 Terence Parr 博士等人于 1989 年开始发展。

2.3 本章总结

本章节对系统开发过程中采用的核心技术与工具进行了深入探讨，涵盖了开发工具、编程语言以及特定技术如 CMake 和 ANTLR 等。其中，对 Lustre 和 L2C 语言的独特性质及其在项目实践中的应用进行了重点阐述。此外，还介绍了 QKind 工具的研发背景及其核心目标。QKind 工具专为 Windows 环境设计，其运行基础是基于 K-induction 的模型验证算法。

第三章　模型验证基础

3.1　介绍

模型验证技术[12,13]是一种自动判断一个程序是否满足其规范的方法[14]。模型验证的经典范式包括建模、规范和算法[15],其中建模是在保留实际系统特点的情况下对其进行抽象,而规范则是以时态逻辑对待验证性质进行的刻画,算法则用于对给定性质和模型进行自动化的判断，并且在发现违反规定性质的反例后可以将其输出，对纠正软件设计与发现软件缺陷上能够起到很好的参考作用[16]。同时模型验证技术作为自动化验证技术的核心环节，致力于确保系统或模型与既定规范标准的一致性[17,18]。在软件工程、硬件设计、通信协议等关键领域，模型验证技术的运用至关重要，为系统正确性与可靠性的提升提供了坚实支撑。该技术通过全面且系统的分析，深入探索系统模型的各个层面，自动揭示潜在错误或违规行为。这一过程不仅帮助开发者在设计与实施阶段及时发现并修正问题，还显著降低了软件或系统运行时的错误和故障率，极大地增强了系统的稳定性与可靠性。此外，模型验证技术的价值不容忽视，对于保障系统质量与提升用户体验具有关键性作用。随着技术的持续进步与完善，我们有充分理由期待，模型验证将在软件工程、硬件设计等未来领域中发挥更加重要的角色。

3.2　模型验证的基本原理

模型验证的核心思路在于，利用状态迁移系统（S）全面描述并发系统的行为，并运用模态/时序公式（F）来精确地刻画系统的特性[19]。因此，原先的问题“系统是否满足既定性质？”被转化为一个数学化的疑问：“S 是否是 F 的模型？”对于有穷状态系统，该问题可以通过特定的算法进行准确的判断。

相对于其他验证方法，模型验证具有两大显著优势：首先，它具有高度的自动化程度，减少了人为介入的需求；其次，当系统未能满足既定性质时，模型验证能够生成反例，从而准确地揭示出错的位置。这两个特性对模型验证在实际应用中的广泛成功起到了至关重要的作用。

进一步深入地理解模型验证，状态迁移系统（S）通常由一组状态、初始状态、迁移关系和终止条件组成，用于描述并发系统的所有可能行为。模态/时序公式（F）是一种数学逻辑表达式，用于描述系统所需满足的性质或约束。因此，模型验证的核心目标是验证是否存在一个从初始状态出发的状态迁移路径，使得该路径满足给定的性质公式（F）。如果存在这样的路径，则认为系统满足既定的性质；否则，系统存在错误或不满足特定性质。

在实际应用中，模型验证技术已被广泛应用于许多领域。尤其是被开发人员用来验证系统的正确性和可靠性。在软件开发过程中，模型验证技术可以帮助开发者在软件开发的早期阶段就发现和修复软件的缺陷，从而提高软件的最终质量和稳定性。而在系统设计和硬件验证领域模型验证技术也同样可以用于验证各种复杂系统的功能正确性、时序约束和性能要求，以便于提高系统的可靠性和效率。

总之，模型验证通过状态迁移系统和模态/时序公式的结合，为并发系统的行为描述和性质验证提供了一种高效、准确的自动化方法。其独特的自动化特性和能够生成反例的能力使其在实际应用中展现出显著的优势和潜力。

3.3 常用的模型验证技术与算法

模型验证技术在软件工程和系统验证中占据重要地位，它是一种自动化的验证方法,并且以其简洁明了和自动化程度高而引人注目[20]。通过对系统模型进行全面而系统的分析，模型验证能够有效地揭示潜在错误或违规行为，协助开发者迅速识别和修复问题，从而极大地减少了软件或系统运行时的错误和故障。

目前，常用的模型验证技术和算法涵盖了许多方面，如有限状态模型验证、带参并发系统模型验证、实时系统模型验证、mu-演算模型验证、归纳算法模型验证、限界模型验证、组合式模型验证以及时间自动机模型验证等等领域。其中，有限状态模型验证作为一种基础方法，通过穷举搜索系统状态空间来验证系统性质。带参并发系统模型验证则专注于分析系统中的并发行为和参数，确定参数的上界，从而将无穷状态验证问题转化为有界模型验证问题。实时系统模型验证则特别关注系统中时间约束和实时性质的验证，确保系统在时间维度上的正确性。mu-演算作为一种逻辑推理工具，用于描述系统行为和性质，并与模型验证算法

相结合进行系统验证。限界模型验证则通过限制状态空间大小来解决状态爆炸问题，提高验证效率。组合式模型验证则将系统分解为多个组件进行验证，以降低分析的复杂度。时间自动机模型验证则聚焦于系统中时间约束和时序性质的验证，确保系统在时间维度上的正确性。

这些模型验证技术和算法都为验证系统性质提供了有效的手段，推动了软件验证领域的持续发展和进步。通过结合不同的技术和算法，研究人员可以够更全面地分析系统行为和性质，发现系统的潜在问题后改进系统设计，从而极大提高软件和系统的可靠性和稳定性。随着模型验证技术的不断完善和应用范围的扩大，我们相信未来将会涌现出更多的方法和工具来为软件工程和系统验证领域带来更多的技术进步和突破。

3.4 模型验证的应用场景与挑战

模型验证，作为一种自动化验证技术，已广泛应用于软件工程和系统验证领域，展现出其实用性与挑战性。该技术主要用于验证计算机软硬件系统的正确性和可靠性，确保系统在运行时符合特定的规范和性能要求。例如利用模型验证技术来检测程序的内存泄漏问题[21]。在本研究中，模型验证被应用于多个关键领域，包括系统安全、系统性能分析、实时调度以及 Web 服务等领域。

在系统安全方面，模型验证技术可以用于测试安全操作系统是否满足特定需求，并且生成高效测试集来检测系统潜在的安全问题。此外，该技术还能检测缓冲区溢出漏洞，提高软件安全性。而在系统性能分析方面，模型验证可以用来评估高不确定性复杂系统的性能，验证该系统是否满足时间或空间性能需求。这有助于开发人员评估系统性能，为设计优化提供依据。而在实时调度方面，模型验证技术还能被用于分析任务组在不同调度策略下的可调度性，从而确保系统能按时完成任务。该技术还能被用于开发分布式实时调度分析工具，提升系统实时性能。

在应用模型验证时，我们也面临许多挑战，如状态空间爆炸[22]、复杂性分析和性能效率问题。状态空间爆炸导致模型验证效率急剧下降，因系统状态空间随规模增长呈指数级增长。复杂性分析涉及全面分析系统行为和性质，需克服规模

和复杂性带来的挑战。性能效率指模型验证算法处理大规模系统的效率和准确性，需不断优化算法以提高验证效率。

3.5 本章总结

本章节对模型验证的基本概念、原理和技术进行了全面深入的阐述，其中涉及了对模型验证技术以以及它们各自适用的场景的讨论。同时，本章节还指出了模型验证技术在具体实践中面临的挑战，如对状态空间爆炸问题。并强调了这些技术在确保系统正确性方面所起到的关键性作用。

第四章 归纳推理算法设计

4.1 介绍

在目前已有的模型验证与验证的研究中，使用 k-induction 算法进行模型验证是目前一个比较成熟的解决方法。在 Alastair F. Donaldson 等人的论文[23]中，他们汇集了当时已有研究中的两条主要方法：使用归纳不变式(The inductive invariant approach)进行程序验证的标准方法。和被他们称为分裂情况 k 归纳算法(split-case k-induction)的传统的 k-inducion 算法。提出了组合情况 k 归纳算法(combined-case k-induction)。并且经过实验表明组合情况 k 归纳算法经常允许程序在使用明显较弱的循环不变式成功验证，从而有效减小验证的开销。

本文将首先回顾论文中提到的这三种算法，并在介绍完关键概念后简要描述其算法原理。之后我们将对这三种算法进行实用性分析，探讨它们在 Lustre 模型验证软件中的可行性、可靠性及其各自在实际运用中的的优缺点。我们将详细介绍将这些算法应用于 Lustre 模型验证软件的具体方法，并对其效果进行深入分析。

4.2 概述

研究人员在归纳算法的描述中常会将程序以控制流图(CFGs)的形式表示。因此在接下来的内容中，我们会将“CFG”和“程序”视为同义词。此外，在后文中出现的 CFG 里我们将遵循使用不确定性分支和 *assume* 语句的组合来建模控制流的标准方法。在执行过程中，假如 CFG 中的 *assume* ϕ语句的计算结果为 false，那么程序将无错误的停止执行后续分支，否则不执行任何操作。而与此相反的是，如果 CFG 中的 *assert* ϕ语句的计算结果为 false，将导致程序错误的停止执行后续分支。

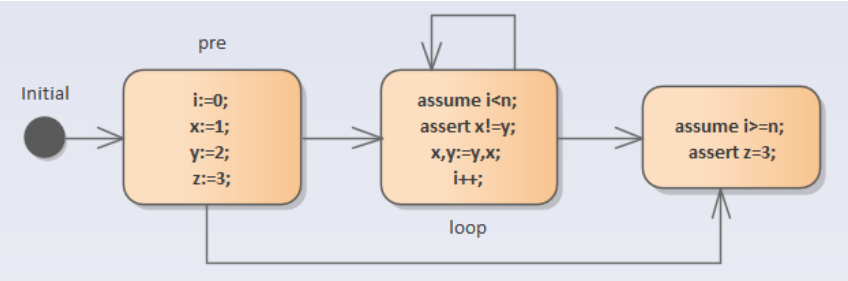

图 4.1 初始 CFG

4.2.1 归纳不变式

归纳不变式方法，作为一种经典的程序验证方法，其核心在于将程序分解为多个循环，并为每个循环关联一个候选不变式。为证实这些候选不变式确实为循环不变式，并具备足够的强度以防止程序中的任何断言被违反，需要进行严格的证明。该方法通过将具有循环的控制流图（CFG）转换为无循环的 CFG 来实现目标，这一转换过程通过在每个循环头部添加一个基本块来完成。该基本块负责验证循环不变式，重置每个循环修改的变量，并预设循环不变式。在原始 CFG 中，每个循环入口边都被重新定向至这些新块，而每个回边则指向一个新的、无子节点的基本块，该块负责为相关循环断言不变式。

虽然在 Lustre 语言的程序验证中，归纳不变式方法通过利用循环不变式替代循环操作，成功消除了 CFG 中的循环。但是该方法在程序语言验证中仍存在两大显著缺陷。第一，归纳不变式方法依赖于人工构造和验证归纳不变式，这通常需要开发者具有较高的数学技巧和经验。第二，该方法可能无法有效应对某些复杂的程序结构，如程序的递归和动态分配，这些问题都在程序验证中构成了一定的挑战。尤其是在处理嵌套循环这一常见情况时，归纳不变式方法的局限性更为明显。同时，程序验证软件需要具备较高的自动化程度以减少对外部输入的依赖。虽然以 Lustre 语言为代表的同步数据流语言在语法层面并未明确包含循环，但在实际的 Lustre 程序设计中，仍会遇到与传统程序语言相似的循环结构。因此，归纳不变式方法更适用于理论研究中的分析工具，而不是运用于具体的模型验证软件的开发中。

如下图 3.2 中，归纳不变式法破坏了在循环中被修改的变量 i,x,y。随后指定循环不变式 *assert* ϕ，将不变式模块插入 pre 与 loop 模块中然后消去 loop 中的后

向边。利用循环不变式来代替原 CFG 中循环从而消除循环准备进行下一步的正确性验证。

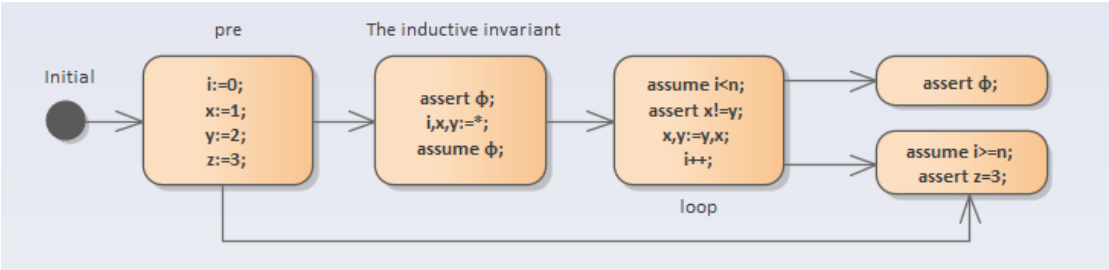

图 4.2 归纳不变式法消除循环后的 CFG

4.2.2 分裂情况 k 归纳算法

k-induction 是一种用于有限状态转换系统的 SAT-based 验证的技术[24]。它被提出为一种用于对状态变量集 s 和 s0 的集合进行编码的初始状态和转换关系进行建模的技术。在 k-induction 方法中，我们首先定义安全属性 P，即程序在循环过程中不能违背的安全条件。接着，我们需要证明在从初始状态开始的 k 步内可以到达的所有状态中，状态满足安全属性 P，我们将这一步称为 Base case。其次，我们需要证明当 P 在 k 个连续的状态中成立时，P 也会在系统的下一个状态中成立，我们将这一步则称为 Step case。在程序循环中应用分裂情况 k 归纳算法时，我们需要将所有的循环分裂为 Base case 与 Step case 进行证明，并且逐步验证整个程序的正确性。

在具体的程序验证中，分裂情况 k 归纳算法相比归纳不变式法具有更弱的循环不变式要求，在 Lustre 语言的验证中相比归纳不变式法也更加高效便捷。但在具体使用中仍然存在一定问题：在分裂情况 k 归纳算法的具体使用中，在 Step case 中，因为 Base Case 与 Step Case 二者之间的验证相对独立，会完全丢失不参与循环的变量信息。同时与归纳不变式相比，分离情况 k 归纳算法在保留赋值信息方面存在一定劣势。 同样，分裂情况 k 归纳算法的逻辑相对简单，利于研究人员和开发人员在实际使用中进行学习与理解。同样，分裂情况 k 归纳算法在编码实现上也相对简单易于实现。若程序中循环数量较少或者嵌套循环出现较少，则验证软件适合使用分裂情况 k 归纳算法对程序进行验证。

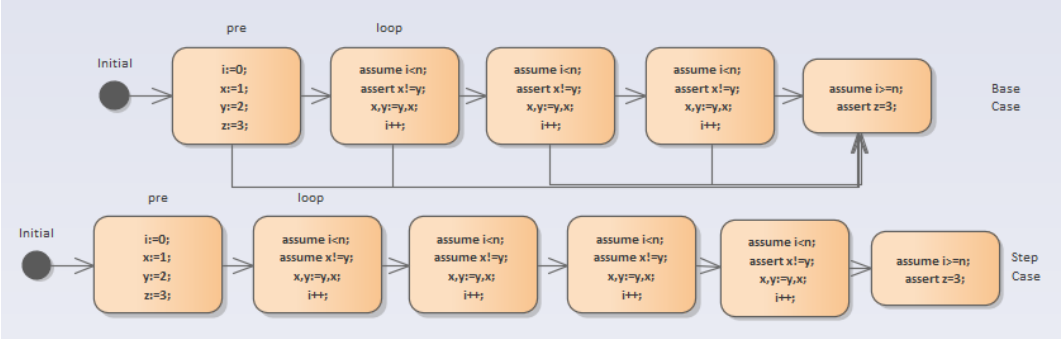

图 4.3 k = 3 时的分裂情况 k 归纳算法

如图 3.3 所示，在 k=3 的条件下，Base Case 与 Step Case 的核心差异在于 Base Case 中的"assert x!=y"条件在 Step Case 中转变为"assume x!=y"。在 k 归纳算法中，Base Case 负责验证循环在 k 轮迭代中是否始终满足特定断言。相对而言，Step Case 则着重于在确认 k 轮迭代满足断言后，进一步确保在第 k+1 次迭代时也能满足该断言。这种转变反映了两种情况下对断言的不同处理方式和验证重点。

4.2.3 组合情况 k 归纳算法

与分裂情况 k 归纳算法将循环分割为 Base Case 与 Step Case 分割为两个完全独立的部分单独验证不同，组合情况 k 归纳算法尝试将二者的证明统一，即在证明完 Base Case 后在不依赖二次输入的情况下进行 Step Case 的验证。其算法流程如图 4.4 所示。

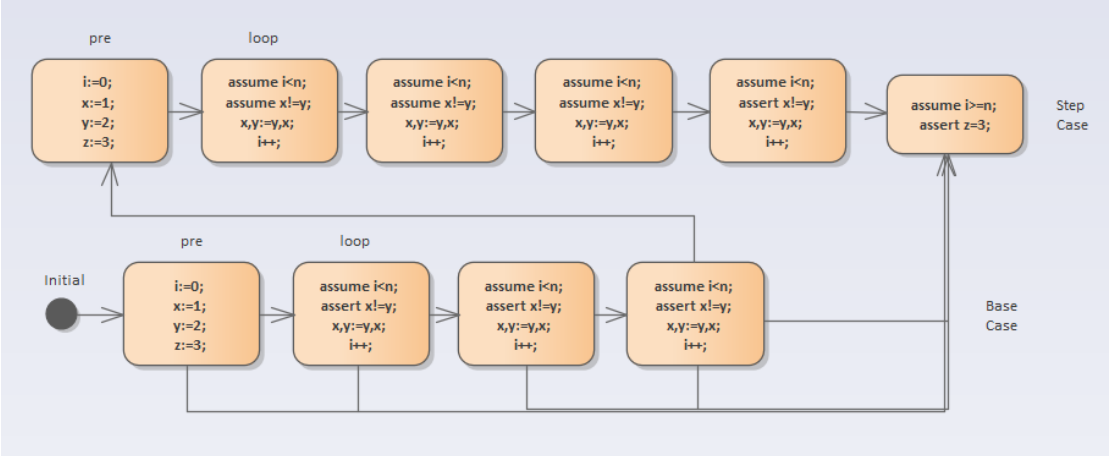

图 4.4 k = 3 的组合情况 k 归纳算法

相较于分裂情况 k 归纳算法需要通过两次输入才能获取两次输出，组合情况 k 归纳算法仅需一次输入即可得到最后输出。这一特性在程序验证中展现了显著的优势，因为它能够直接依据算法将代码中的所有循环结构转化为非循环结构而不必破坏代码的其余部分。因此在指定 k 值的情况下，我们可以统一地将代码中的所有循环结构根据组合情况 k 归纳算法的规则转化为非循环结构，然后再进行整体程序正确性的验证。而相比之下，分裂情况 k 归纳算法需要将程序中的每个循环分别提取并单独验证，这无疑增加了验证软件的复杂性。

此外，由于组合情况 k 归纳算法具有统一循环处理规则和整体统一验证两大特性，使其在处理多重嵌套循环时也能游刃有余。根据这些规则，我们可以在做到消除循环的同时确保结构统一的代码在经过组合情况 k 归纳算法处理后，仍然能保持为一个整体的有向无环图。因为这种结构特点能在为后续操作中开发人员使用图论中的各种算法来对程序进行进一步的验证或研究提供了便利。

在具体的程序循环消除工作中，组合情况 k 归纳算法在无指定 k 值的情况下一次切割 CFG 中的一个循环并且用一组数据进行验证，将 Base Case 的输出作为 Step Case 的输入从而保证了代码的完整性，而不是用两组数据分别验证 Base Case 和 Step Case，从而对比分裂情况 k 归纳算法做到了不丢失循环中的未修改变量信息。但其不足之处在于程序验证中的断言不变式仍然需要手动添加或者由其它模块进行生成。但实验证明即使仍然需要外部输入或指定 k 归纳算法中需要的不变量，组合情况 k 归纳算法所需的不变量可以比分裂情况 k 归纳算法较弱。从而增强了模型验证软件的验证能力，并且减小了验证所需的前期成本。

4.2.4 k 归纳算法性质

Alastair F. Donaldson 等人在论文中进行了一项研究，深入探讨了组合情况 k 归纳算法在处理程序循环结构时的性质和行为。他们的研究发现，在 k 值相同的情况下，算法在处理嵌套循环时无论是先处理内层还是先处理外层循环结果均不受影响。这使得算法在处理嵌套循环程序时更加灵活。

此外，该研究还发现，在实际应用中，优先消除内侧循环的策略能显著提高算法效率。因此，在编写 k 归纳程序正确性验证软件时，建议采用此策略。然而，

使用组合情况 k 归纳算法将循环结构拆解为非循环结构时，会产生一定的空间开销。处理后的程序相较于处理前会有所增长。研究结果显示，处理不相交循环时，程序大小呈现线性增长，而处理嵌套循环时，程序大小将呈现指数增长。考虑到 Lustre 语言的特性，多层嵌套循环的嵌套深度通常不会过高，因此空间开销在可接受的范围内。

总的来说，组合情况 k 归纳算法在处理程序循环结构时表现出灵活性和高效性，并且能够有效控制空间开销。这些特性使其成为处理带循环程序的重要工具之一，对于提高程序正确性验证的效率和准确性具有重要意义。因此在接下来具体的 Lustre 语言验证软件的开发中，在对比上述 3 中主流程序验证算法后，我们将选用各方面更加占优势的组合情况 k 归纳算法进行程序的编写工作。

4.3 本章总结

本章详细介绍了归纳算法的设计过程，包括普通的归纳不变式方法以及两种 k 归纳算法、分裂情况 k 归纳算法与组合情况 k 归纳算法。同时，本章还深入剖析了这些算法的特性，并探讨了它们在模型验证软件的实际开发中的优劣性与实用价值。

第五章 归纳推理算法的代码实现

5.1 Z3 库

本程序在归纳算法的实现上使用 Z3 库实现。Z3 是一个开源的高性能定理证明器，由微软研究院开发[25]。它提供了一个功能强大的库，用于解决多种形式的自动定理证明问题。该库支持多种语言，包括 C、C++、Python 等，使得它在各种领域的应用变得更加广泛和灵活。Z3 库的核心功能包括布尔逻辑、线性整数和实数算术、位向量和数组理论等。其强大的求解能力和高效的性能使得它成为了学术界和工业界的研究人员首选的自动定理证明工具之一。除了基本的定理证明功能外，Z3 库还提供了丰富的 API 和文档，以便用户可以轻松地集成和扩展其功能。同时，它还支持各种算法和优化技术，以提高求解效率和准确性。综上，Z3 库是一个功能强大、灵活易用的自动定理证明工具，为研究人员提供了一个强大的工具，用于解决各种复杂的逻辑和数学问题，同时也适合用来完成归纳推理算法的代码实现工作。

5.2 同步数据流语言的循环语句处理

由于同步数据流语言特性，其语言本身并不存在严格的循环语句。但却拥有特殊形式的具有循环性质的”->”语句，其形式与其等价 C++语句如图 5.1 所示：

Lustre语言"->"语句:

```
count = init -> next;
```

等价C++循环语句:

```
int time = 1;
while(true){
    if(time == 1) count = init;
    else count = next;
    time ++;
}
```

图 5.1 Lustre 语句与等价 C++代码

5.3 分裂情况 k 归纳算法的代码实现

本节将介绍如何使用 Z3 库实现分裂情况 k 归纳算法来验证给定的 C++代码片段。该算法将被分成两个部分：Base Case 和递推情况 Step Case。我们将使用 Z3 库的约束求解器来对基本情况和递推情况进行验证，手动指定对应循环不变式，并最终确定程序的正确性。

5.3.1 预处理部分

在进行验证前，需要利用 Z3 库中的 expr 元素定义好后期验证所需要的变量与不变式。其重点代码部分如图 5.2 所示：

预处理部分关键代码：

```
// 初始化 Z3 context
context ctx;
// 变量声明
expr time = ctx.int_const("time");
expr count = ctx.int_const("count");
expr init = ctx.int_const("init");
expr next = ctx.int_const("next");
// 创建 Z3 solver
solver s(ctx);
// 赋初值操作
s.add(time == 1);
s.add(count == init);
// 循环不变式: time > 0
expr loop_invariant = time > 0;
```

图 5.2 预处理部分 C++代码

预处理部分的重点和难点在于，之前所存储的代码信息皆以字符串的形式存储，但如果要使用 Z3 来进行代码求解的话就必须将字符串形式的信息转换为符合 Z3 库规则的信息。只有成功将信息转换为合法的 Z3 求解器中的内容，我们才能进一步对代码进行解析。

5.3.2 Base Case 验证

Base Case 的重点在于验证初始 k 次迭代，其重点代码部分如图 5.3 所示。

Base Case部分关键代码：

```
// Base case: 验证初始k步成立
for (int k = 1; k <= 5; k++) {
    s.push();
    s.add(time == k);
    s.add(count == (ite(time == 1, init, next)));
    s.add(loop_invariant);
    if (s.check() == sat) std::cout << "Base case verified for k = " << k << std::endl;
    else std::cout << "Base case failed for k = " << k << std::endl;
    s.pop();
}
```

图 5.3 Base Case 部分 C++代码

5.3.3 Step Case 验证

Step Case 的重点在于验证在任意成功的 k 次迭代后，第 k+1 次迭代仍然能成功验证。其重点代码部分如图 5.4 所示。

Step Case部分关键代码：

```
// Step case: 验证k步成立后第k+1步仍然成立
s.push();
int k = k_init;
for (k = k_init; k <= k_init + 5; k++) {
    s.add(time == k);
    s.add(count == (ite(time == 1, init, next)));
}
k++;
s.add(time == k);
s.add(loop_invariant);
if (s.check() == sat) std::cout << "Verification successful for k = " << std::endl;
else std::cout << "verification failed for k = " << k << std::endl;
s.pop();
```

图 5.4 Step Case 部分 C++代码

5.4 组合情况 k 归纳算法的代码实现

本节将介绍如何使用 Z3 库实现组合情况 k 归纳算法来验证给定的 C++代码片段。该算法将 Base Case 和递 Step Case 相统一求解。我们同样使用 Z3 库的约束求解器进行验证，手动指定对应循环不变式，并最终确定程序的正确性。

5.4.1 预处理部分

组合情况 k 归纳算法与分裂情况 k 归纳算法在预处理环节存在显著的一致性。为避免冗余，我们将不再赘述其共通之处，相关内容直接参考本文第 5.3.1 部分的详细内容即可。

5.4.2 组合验证

组合情况 k 归纳算法的验证部分重点将 Base Case 与 Step Case 统一验证。其重点代码部分如图 5.5 所示。

相比与分裂情况 k 归纳算法的分开验证，由图我们可以清楚的感受到前文提到的组合验证 k 归纳算法带来的代码完整性。同时结合验证也可以带来更少的代码信息丢失。

组合验证部分关键代码：

```
s.push();
int k = k_init;
for (k = k_init; k <= k_init + 5; k++) {
    s.push();
    s.add(time == k);
    s.add(count == (ite(time == 1, init, next)));
    s.add(loop_invariant);
    if (s.check() == sat) std::cout << "Base case verified for k = " << k <<
std::endl;
    else std::cout << "Base case failed for k = " << k << std::endl;
    s.pop();
}
for(int i = 1; i <= 5; i ++){
    s.add(time == k + i);
    s.add(count == (ite(time == 1, init, next)));
}
k = k + 6;
s.add(time == k);
s.add(loop_invariant);
if (s.check() == sat) std::cout << "Verification successful for k = " << std::endl;
else std::cout << "verification failed for k = " << k << std::endl;
s.pop();
```

图 5.5 组合验证部分 C++代码

5.5 本章总结

本章节主要对两种归纳验证算法的编程实现进行了清晰且精炼的说明。鉴于本文的主要目标是聚焦于归纳算法的设计及其实现方式，故对于涉及 Z3 库的具体应用、循环不变式的构建流程，以及 Lustre 代码的编译解析等细节内容，本文将不予详细探讨。

第六章　模型验证软件的归纳算法模块架构

6.1　模型验证软件的其它模块简介

6.1.1 L2C 解析子系统模块

L2C 解析子系统模块的核心职责在于对 L2C 程序进行建模与解析，进而将其转化为符号化形式，为后续深入分析与处理奠定基础。其具体功能细分如下：

词法分析：L2CLexer 组件承担识别输入文件中符号与标记的任务，将这些元素转化为标准化的词法单元。

语法分析：L2CParse 组件依据输入文件的语法规则，将词法单元有序组织成抽象语法树，清晰展现元素间的结构关系与层次组织。

语义分析与中间代码生成：在此阶段，系统将对抽象语法树进行语义一致性校验，并生成中间代码表示，这对后续的优化转换步骤具有至关重要的作用。

内容生成：基于程序中的声明与定义，系统能够自动生成变量、数据类型及相关约束，为模型构建与验证提供便利。

转换系统生成：最终，根据输入文件的定义与属性，系统将创建完整的系统模型及待验证的属性集合，为后续验证工作提供坚实基础。

6.1.2 不变式生成模块

模型验证软件中的“不变式生成模块”旨在发掘在系统执行过程中始终维持的属性，即不变式，本质上是一种数学思想[26]。这些不变式对于识别系统错误、潜在死锁以及证明系统正确性具有关键性意义。不变式的类型主要分为以下四类：

普通不变式生成包括生成一阶和二阶不变式。一阶不变式用于描述在任何给定状态下，状态变量之间成立的关系；而二阶不变式则用于描述在不同状态之间或系统执行序列中必须始终维持的关系。

不变式生成的定义则是指：在系统演变或修改的过程中能够自动更新或生成新的不变式。这种方法有助于开发人员对系统行为和正确性进行全面分析，从而确保能进行有效的后续验证。

在模型验证软件中，不变式生成过程是自动化的，即自动通过运用先进的算法来推导出描述系统行为和状态之间相互关系的属性。这一过程极大程度提升了

系统验证过程的健壮性和可靠性，使得软件能够有效地分析和验证复杂的模型和系统行为。

6.2 模型验证软件的归纳算法模块

k-归纳模型检查是逐步验证系统属性在执行步骤增加时保持真实性的迭代过程。通过递增验证，增强验证的深度和鲁棒性。在每个步骤 k 上严格检验属性有效性，逐步逼近真实系统行为。

归纳算法过程步骤如下：

1. 模型和属性规范：明确验证的系统模型和相关属性，为后续分析和验证提供明确目标。

2. 参数设置和模型转换：设定 k-归纳过程参数，转换系统模型为适合 k-归纳分析的形式，确保分析准确有效。

3. 有限状态空间生成：根据指定步骤范围 k，运用算法和数据结构生成系统模型的有限状态空间，为后续属性验证提供基础。

4. k-归纳证明：在每个步骤 k 上，验证器尝试证明属性有效性。如属性在某步骤得到证明，则继续下一步；否则，寻找反例证明属性不成立。

5. 结果报告和反例分析：如属性在某步骤被反驳，生成反例说明失败情况。有助于深入分析错误，为系统改进提供反馈。

k-归纳方法对于逐步全面证明或反驳系统属性至关重要。它允许对模型行为在多个执行步骤上进行详细评估和分析，提供全面深入的验证结果。

实施 k-归纳模型检查时，需关注算法的可扩展性和效率。利用优化算法和数据结构有效管理和探索状态空间，提高验证过程的效率和准确性，为系统设计和开发提供可靠保障。

6.2.1 SSA 代码转换模块

本模块的主要任务聚焦于如何接收并处理从 L2C 解析子系统模块生成的静态单一赋值（Static Single Assignment，SSA）格式[27]的 Lustre 代码。该处理过程是通过构建 LustreNode 和 VarStateList 等对象来实现的，这些对象是对

SSA 格式 Lustre 代码中包含的各种元素（如参数、返回值、函数语句等）的抽象表示。

首先，LustreNode 对象创建的目的是为了捕获和封装 Lustre 代码中的主要结构和逻辑单元。即每个 LustreNode 实例实际上代表了一个独立的函数或模块，其中包含对应的输入参数、输出参数、内部变量和逻辑表达式。在通过对输入的 SSA 格式的 Lustre 代码借助正则表达式进行解析后，我们可以从代码中提取出这些信息然后封装成 LustreNode 类。这种封装不仅有助于后续的分析和处理，也便于在模型验证和优化阶段对代码结构进行快速访问和修改。

随后，为了进一步详细的表示代码模块内容，我们创建了 VarStateList 类。该类通过列表形式维护了所有变量的状态信息，其中每个条目详细记录了变量的当前状态和可能的转变路径。这种详尽的状态跟踪对于执行符号执行和模型检查尤为重要，因为它们提供了必要的信息来判断代码的行为是否符合预期规范。

通过上述程序设计，系统便能够高效地处理 SSA 格式的 Lustre 代码，为后续的验证和优化步骤提供了坚实的基础。该模块处理结果样例如图 6.1 所示。

```
====================LustreNode类信息==========================
NodeName: : [Counter]
NodeInput: : [init : int; ]
NodeOutput: [count : int; ]
NodeVars: []
NodeBodys: [count = init -> count + 1;; ]
NodeStates:
From [count] to [init -> count + 1]
====================VarStateList类信息==========================

[count]'s state:
1.VarName is: [count], VarType is: [int]
2.[count]'s Statements:
{init -> count + 1}

[init]'s state:
1.VarName is: [init], VarType is: [int]
2.[init]'s Statements:
```

图 6.1 SSA 代码转换模块调试信息

6.2.2 Z3 求解器生成与验证模块

本模块的主要任务则主要聚焦于该模块的核心任务是根据静态单赋值（SSA）代码转换模块的输出构建相应的 Z3 求解器，并使用 k 归纳法来验证 Lustre 代码的正确性。

模块首先接收由 SSA 代码转换模块生成的中间表示形式，该形式包含了转换后的 Lustre 代码的所有必要信息。基于这些信息，CreateSolver 类的实现负责初始化 Z3 求解器的配置，确保求解器能够理解和处理 Lustre 代码中的逻辑和数据流。此过程包括定义逻辑变量、设置约束条件以及配置求解器参数，以适应复杂的验证任务。

求解器配置完成后即可调用 KInduction 模块对 Lustre 代码的正确性进行系统的验证。完成 k 归纳法的验证后，程序将分析求解器的求解结果，以确定 Lustre 代码是否满足标准。在验证过程中发现的任何问题都会被程序记录并报告，以便开发者进行调整和优化。此外，求解器的输出不仅限于判断代码的可满足性，还包括具体的模型数据，这些数据有助于详细了解代码在特定条件下的行为。

```
====================CreateSolver类信息==========================
Split case K-induction : count = init -> count + 1
Base case verified start:
Base case verified for k = 1
Base case verified for k = 2
Base case verified for k = 3
Base case verified for k = 4
Base case verified for k = 5
Step case verified start:
Step case holds!
Split case K-induction verified successfully.
Combined case K-induction : count = init -> count + 1
Combined case verification start:
Combined case holds for k = 1
Combined case holds for k = 2
Combined case holds for k = 3
Combined case holds for k = 4
Combined case holds for k = 5
Combined case verification completed.
Combined case K-induction verified successfully.
```

图 6.2 Z3 求解器生成与验证模块调试信息

通过将 SSA 转换、Z3 求解器构建和 k 归纳法验证整合到一个连贯的流程中，我们的模块为 Lustre 代码的高效和准确验证提供了强大的支持。这种集成方法不仅提高了验证过程的自动化程度，也优化了处理复杂系统时的性能和可靠性。

6.3 本章总结

本章节详细阐述了模型验证软件中归纳算法模块的整体架构，明确了各模块的具体职责及其协同工作的方式，以实现对归纳算法的全面支持。特别地，对 L2C 解析子系统和不变式生成模块的设计思路及实现细节进行了深入剖析。

第七章 软件系统归纳验证模块设计

7.1 Tool 模块具体实现

Tool 工具模块主要用途为为其它模块提供工具函数，其具体内容如下：

1. 输入输出工具

1) GetInput: 提供一个方法 getInputByFilename，通过文件名读取文件内容，将内容作为字符串返回。

2) ExportOutput: 包含一个方法 exportOutputToFile，将指定内容写入到一个文件中，通常用于输出结果或日志。

2. 字符串处理工具

1) StringTool: 提供多种字符串处理功能，包括：

a> StrSplitting: 根据指定分隔符拆分字符串，并清理拆分结果中的空字符串和多余空格。

b> StrCheck: 检查一个字符串是否包含另一个子字符串。

c> get_substring_between: 获取两个子字符串之间的内容。

d> get_substring_to 和 get_substring_from: 分别获取从字符串开始到指定子字符串的内容，和从指定子字符串之后的内容。

5> StringVectorTool: 提供对字符串向量的操作，如删除空字符串、修剪字符串和合并字符串向量。

3. 检查和调试工具

1) CheckTool: 包含多个版本的 print 方法,用于打印字符串或字符串向量，方便调试和跟踪程序执行过程。

4. Z3 求解器工具

1) Z3Tool: 与 Z3 求解器交互的工具，包括：

a> create_solver: 创建并返回一个 Z3 求解器实例。

b> isOperator: 判断字符是否为运算符。

c> GetPriority: 获取运算符的优先级，用于中缀表达式到后缀表达式的转换。

d> infixToPostfix: 将中缀表达式转换为后缀表达式，便于计算和处理。

Tool 工具模块类图如图 7.1 所示：

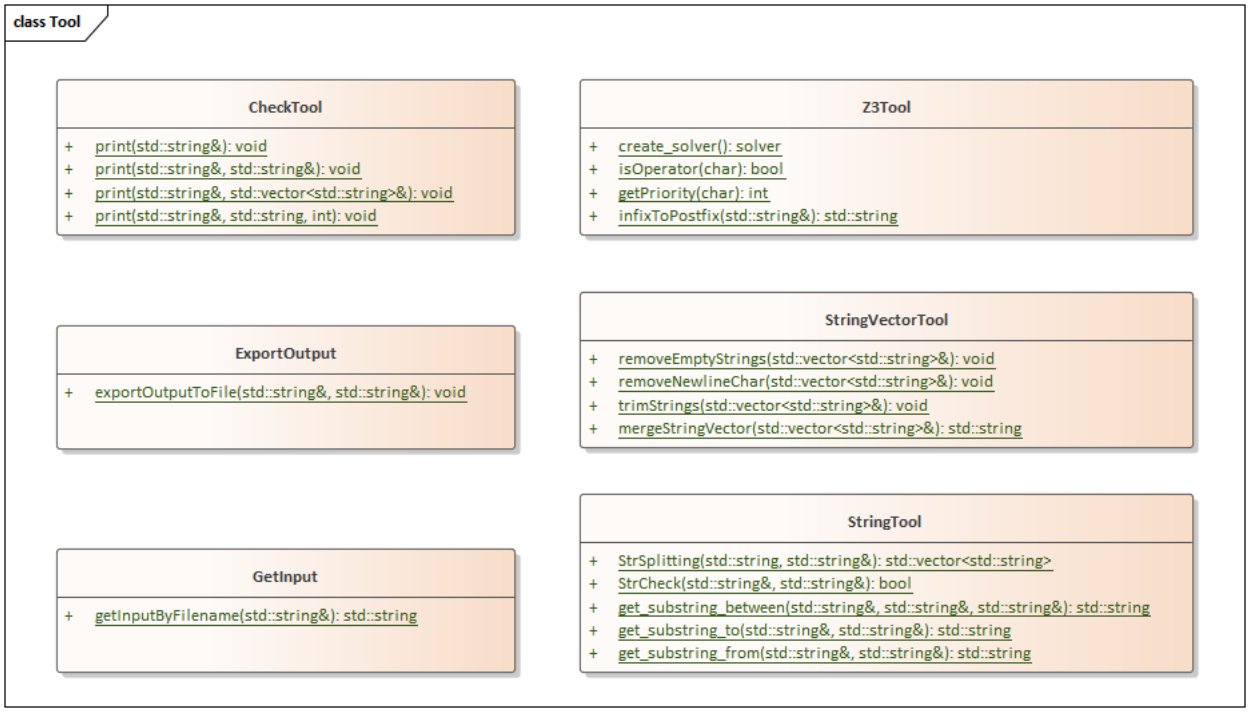

图 7.1 Tool 模块类图

7.2 Translate 模块具体实现

Translate 翻译模块的作用主要用于解析和处理 Lustre 语言代码，即把 Lustre 代码转化为内部定义表示形式以便进一步分析和操作。该模块为 Lustre 程序提供了一套完整的解析和表示工具，令软件能够有效地进行程序分析和验证。该模块具体内容如下：

1. 信息提取工具

1) ExtractLustreInfo: 此类提供一个名为 extractLustreInfo 的方法，主要功能是从给定的 Lustre 代码中提取必要的信息，并构建一个对应的 LustreNode 对象。该过程包括解析节点名称、输入输出变量、局部变量及其初始

化和转移关系等关键信息。该方法利用正则表达式和字符串操作技术进行代码解析，确保从复杂的代码结构中准确抽取数据。在编写的过程中需要注意不同的开发人员所编写的代码习惯和风格可能有所区别，因此在正则表达式编写过程中和后续的代码处理过程中需要做到能对不同风格的代码进行正确的信息提取，从而保证后续算法的正确进行。

2. Lustre 节点表示

1) LustreNode: 这个类用于详细存储解析后的 Lustre 节点信息。存储的信息包括节点名、输入参数、输出变量、局部变量以及整个函数体的语句。LustreNode 还包含一系列方法，如 getName、getInputs、getOutputs 等，用于外部访问节点的各个组成部分。此外，提供了 showNodeMessage 方法用于在控制台打印节点的详细信息，帮助开发者进行调试和验证。

3. 变量状态管理

1) VarState: 这个类表示单个变量的状态，其中包括变量名、类型和与该变量相关的所有语句。VarState 提供了设置和获取变量名（setName、getName）、变量类型（setType、getType）以及操作相关语句的方法（statement_pushback、getStatement），从而方便地管理和访问变量的详细状态。由于我们通过 L2C 解析模块将所有的代码均转换为了 SSA 格式代码，保证了每个变量最多只被赋值一次，因此我们可以保证每一个 VarState 实例的相关语句不会超过一句，大大降低了后续归纳验证难度。

2) VarStateList: 此类用于管理一组变量的状态，通常与一个 Lustre 节点关联。VarStateList 通过 BuildVarStateList 方法从 LustreNode 实例中构建变量状态列表，该列表以映射形式存储，键为变量名，值为对应的 VarState 对象。此外，该类还提供 ShowMessage 方法，用于显示所有变量的状态信息，以及 getVarStateList 方法，用于外部访问整个变量状态列表。

总的来说，这些工具和类为 Lustre 代码的解析、表示和状态管理提供了系统性的支持，并为有助于后续的验证和分析。这些工具同样可以让开发者更高效地进行 Lustre 程序的调试、验证和功能扩展。

Translation 模块类图如图 7.2 所示：

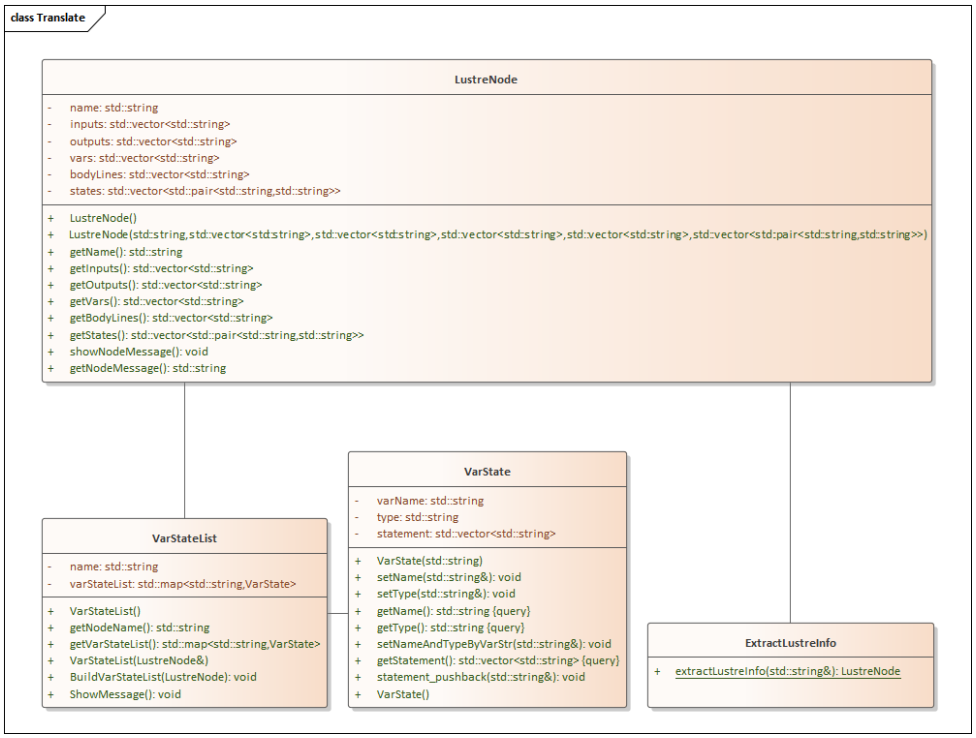

图 7.2 Translate 模块类图

7.3 Z3Solver 模块具体实现

Z3Solver 模块主要用途是使用 Z3 求解器来实现 K 归纳法和解析表达式，以支持项目中的逻辑验证和模型检验，其具体内容如下：

1. 求解器创建工具

1) CreateSolver：提供了多个与 Z3 求解器交互的方法，主要包括：

a> build_z3solver：根据 VarStateList 中的变量状态列表构建 Z3 求解器实例，为每个变量创建对应的 Z3 表达式，并将逻辑约束添加到求解器中。

b> parse_bool_digit_expr：解析布尔值或数字表达式，将字符串表达式转换为 Z3 表达式。

c> parse_expr：解析一般表达式，支持算术和逻辑运算符，可以处理复杂的嵌套和运算表达式。

d> add_expr_to_solver：将表达式添加到求解器中，用于动态调整求解器的约束条件。

2. K 归纳法实现

1) KInduction: 实现了 K 归纳法的两种主要场景：

a> split_case_k_induction: 实现分离情形的 K 归纳法，用于验证基本情况和归纳步骤是否能够满足给定的逻辑约束。

b> combined_case_k_induction: 实现结合情形的 K 归纳法，同时处理基本情况和归纳步骤，验证在多个迭代步骤中逻辑约束的连续性和一致性。

这些工具和方法为项目提供了强大的逻辑验证能力，特别是在处理复杂的逻辑约束和算法验证时，程序能够有效地利用 Z3 求解器的功能来执行模型检验和安全验证操作。同样通过这些工具，开发者可以构建精确的验证逻辑，确保程序行为的正确性和稳定性。

综上，Z3Solver 模块封装了 Z3 求解器的复杂操作后向外提供了一套易于使用的方法。此外，该模块同样通过动态构建求解器和灵活处理表达式显著提升了项目的验证效率和可靠性，是项目核心功能的重要支撑。

Z3Solver 模块类图如图 7.3 所示：

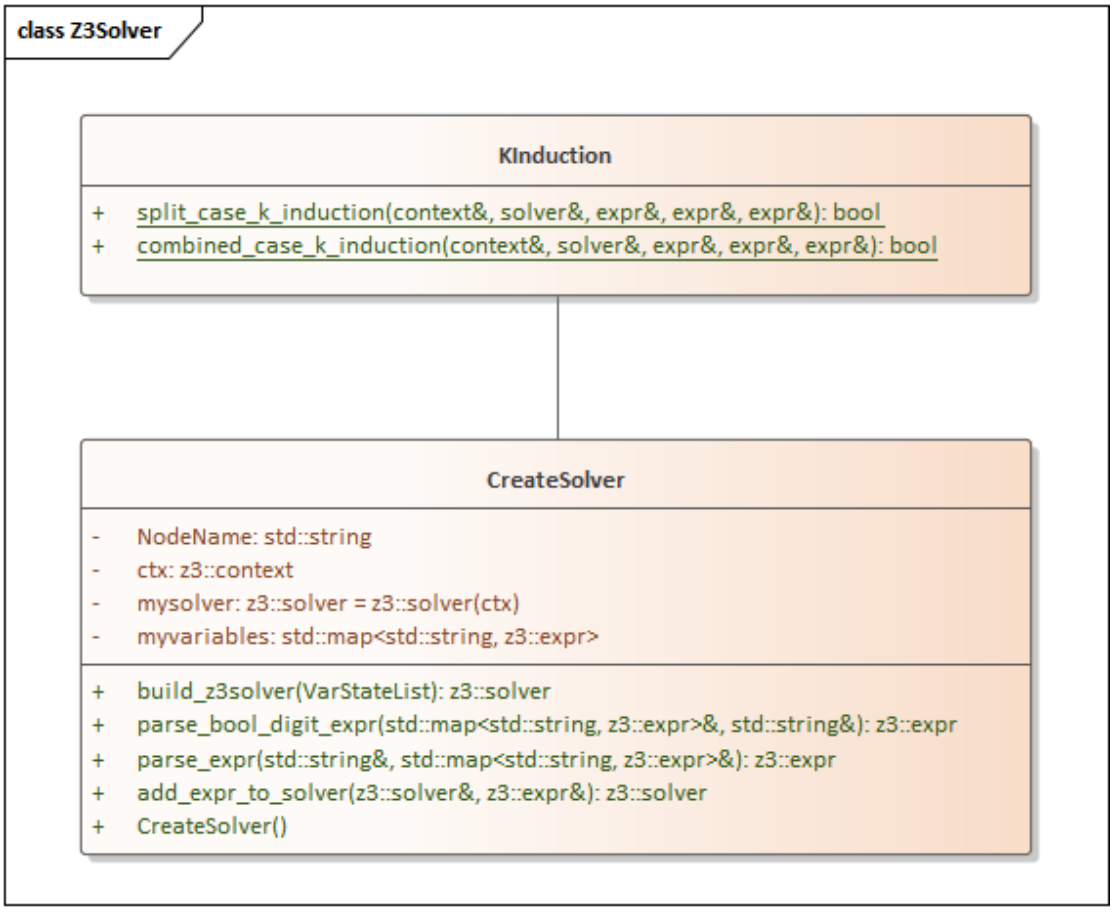

图 7.3 Z3Solver 模块类图

7.4 归纳算法模块总体类图

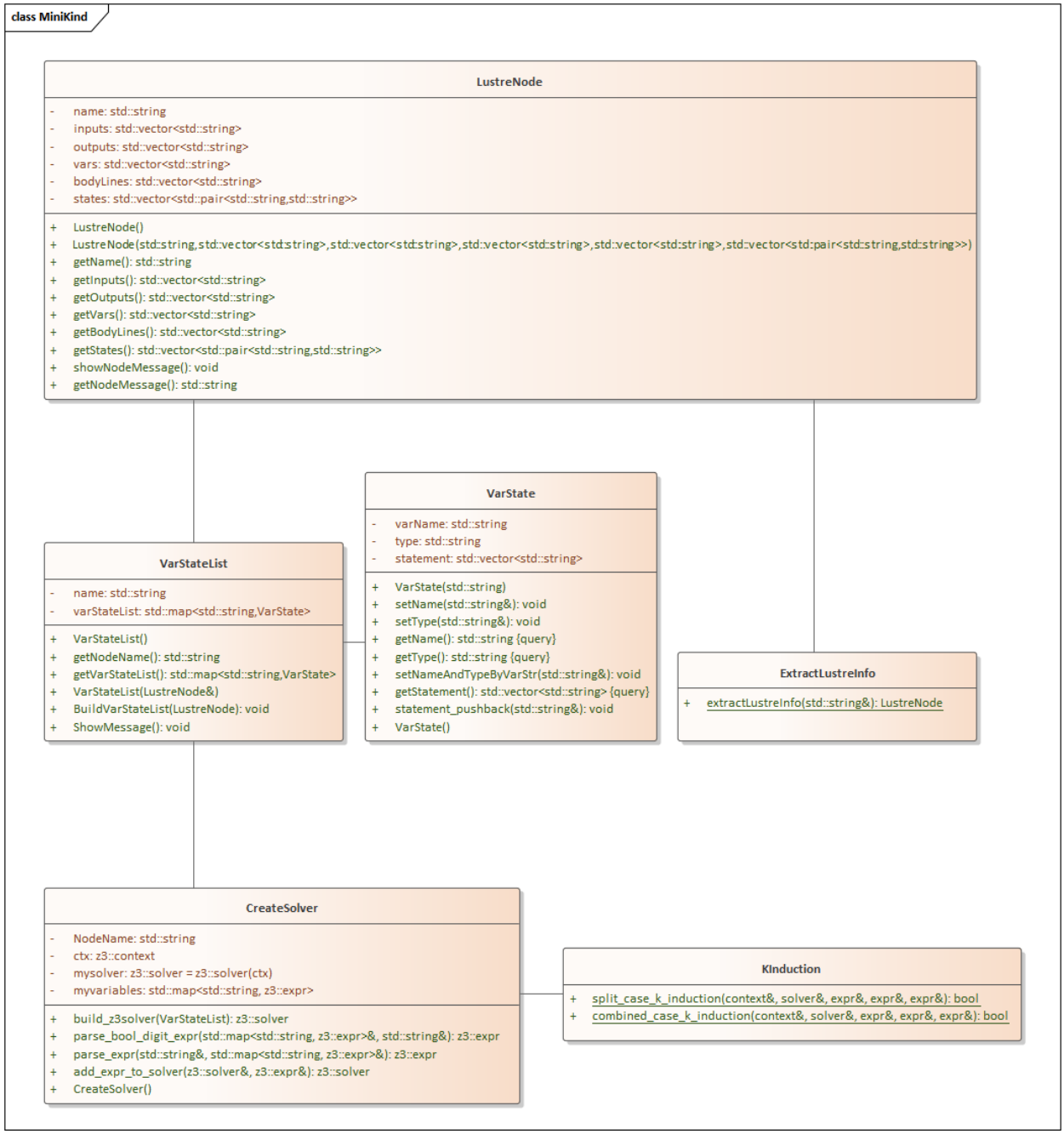

图 7.4 归纳算法模块整体类图

在具体的使用中，归纳算法模块将接收解析子系统模块输出的 SSA 代码，最后将输出一个 Z3::solver 求解器以供进一步处理与验证。

7.5 本章总结

本章着重介绍了软件系统中归纳验证模块的设计与实施过程，详细阐述了Tool 模块、Translate 模块和 Z3Solver 模块等核心组件的功能及其实现机制。通过对这些组件的深入剖析，充分展示了软件验证模块的高度集成性和优化潜力，为软件系统的稳定性和可靠性提供了有力保障。

第八章　系统测试

8.1　系统合法性检查

我们为确保软件开发过程的合规性，将对在开发本软件时使用过的的开发工具和第三方平台进行全面确认，以保证其合法性。我们同时还将仔细审查在编程过程中使用的自主开发且非开发工具提供的控件、组件、函数库等，以确保它们具备合法的发布许可。这些措施将保障我们在软件开发过程中的合法性和规范性，从而确保最终产品的合规性。

开发过程中，我们采用了 Intellij CLion 作为开发工具，并以 C++作为主要编程语言。在此过程中，所使用的所有控件、组件以及函数库均源自开发工具内部，确保其合法性和适用性。

8.2　系统功能测试

8.2.1 模型验证软件其它模块测试

表 8.1 模型验证软件其它模块测试表

项目检查	操作	测试结果	与预期结果偏差	备注
词法分析	输入 L2C 代码执行词法分析操作	成功词法分析	测试结果符合预期结果，无偏差	
语法分析	词法分析结束后执行语法分析操作	成功语法分析	测试结果符合预期结果，无偏差	前提是词法分析正确
语法树生成	语法分析结束后执行生成语法树	成功生成语法树		

	操作		测试结果符合预期结果，无偏差	前提是语法分析正确
SSA 格式 Lustre 代码转换	生成语法树后执行 SSA 格式 Lustre 代码转换操作	成功生成 SSA 格式的 Lustre 代码	测试结果符合预期结果，无偏差	前提是成功生成语法树
错误报告	输入 L2C 代码后启动 L2C 解析子模块	成功提示“输入的 L2C 代码不合法”	测试结果符合预期结果，无偏差	
文件读取	读取 input 文件中的内容	成功读入 input 文件中的 L2C 代码	测试结果符合预期结果，无偏差	
文件输出	输出 SSA 代码	成功将生成的 Lustre 代码写入到 SSA.lustre 文件中	测试结果符合预期结果，无偏差	
不变式生成	接收 SSA 代码并且输出循环不变式	成功生成 Lustre 代码的循环不变式	测试结果符合预期结果，无偏差	

8.2.2 Tool 模块测试

表 8.1 Tool 模块测试表

项目检查	操作	测试结果	与预期结果偏差	备注
字符串格式输出	输入指定参数调用 CheckTool::print 函数	成功输出指定格式字符串	测试结果符合预期结果，无偏差	
字符串 vector 格式输出	输入指定参数调用 CheckTool::print 函数	成功以指定格式输出 string vector 中的所有元素	测试结果符合预期结果，无偏差	重载调用 CheckTool::print 函数
指定路径文件读取	输入指定参数调用 getInputByFilename 函数	成功读取指定文件中的内容并且返回字符串	测试结果符合预期结果，无偏差	文件路径必须是合法路径
指定路径文件输出	输入指定参数调用 exportOutputToFile 函数	成功将字符串写入指定路径文件	测试结果符合预期结果，无偏差	

根据指定字符串分割字符串为字符串 vector	输入指定参数调用 StrSplitting 函数	成功得到预期字符串 vector	测试结果符合预期结果，无偏差	
检查字符串中是否包含指定子字符串	输入指定参数调用 StrCheck 函数	成功返回正确验证结果	测试结果符合预期结果，无偏差	
获取指定两个字符串间子字符串	输入指定参数调用 get_substring_between 函数	成功获取指定字符串	测试结果符合预期结果，无偏差	
获取从指定子字符串开始的子字符串	输入指定参数调用 get_substring_from 函数	成功获取指定字符串	测试结果符合预期结果，无偏差	
去除字符串 vector 中的所有空字符串	输入指定参数调用 removeEmptyStrings 函数	成功去除所有空字符串	测试结果符合预期结果，无偏差	

去除字符串 vector 中的所有字符串无的首尾空格	输入指定参数调用 trimStrings 函数	成功去除 vector 中所有字符串的首尾空格	测试结果符合预期结果，无偏差	
合并字符串 vector 中的所有字符串	输入指定参数调用 mergeStringVector 函数	成功得到指定字符串	测试结果符合预期结果，无偏差	
根据字符串 vector 中所有字符串中的换行符号	输入指定参数调用 removeNewlineChar 函数	成功消除字符串 vector 中的所有元素中的所有换行符	测试结果符合预期结果，无偏差	
字符串中缀表达式转后缀表达式	输入合法四则运算中缀表达式后调用 infixToPostfix 函数	成功获得指定后缀表达式	测试结果符合预期结果，无偏差	输入的中缀表达式需要合法
初始化 z3::solver 元素	调用 create_solver 函数	成功获得 z3::solver	测试结果符合预期结果，无偏差	z3::solver 需要手动构造

8.2.3 Translate 模块测试

表 8.2 Translate 模块测试表

项目检查	操作	测试结果	与预期结果偏差	备注
解析输入的 Lustre 字符串	输入 lustre 代码调用 extractLustreInfo 函数	成功解析 lustre 字符串并将其转换为 LustreNode	测试结果符合预期结果，无偏差	
获取函数名	调用 LustreNode 类的 getName 函数	成功获取函数名字符串	测试结果符合预期结果，无偏差	
获取函数的所有的函数语句	调用 LustreNode 类的 getBodyLines 函数	成功获取包含函数所有语句的字符串 vector	测试结果符合预期结果，无偏差	函数语句以 pair<string,string> 格式储存，对应等号左右内容
获取函数的临时变量列表	调用 LustreNode 类的 getVar 函数	成功获取包含函数所有临时变量的字符串 vector	测试结果符合预期结果，无偏差	
	调用 LustreNode 类的 getInputs 函数	成功获取包含函数所有参数		

获取函数的参数列表		的字符串 vector	测试结果符合预期结果，无偏差	
获取函数的返回值列表	调用 LustreNode 类的 getOutputs 函数	成功获取包含函数所有返回值的字符串 vector	测试结果符合预期结果，无偏差	
获取函数的语句列表	调用 LustreNode 类的 getStates 函数	成功获取包含函数中所有语句的 vector	测试结果符合预期结果，无偏差	
展示函数的所有信息	调用 LustreNode 类的 showNodeMessage 函数	成功输出函数对应的 LustreNode 实例的所有信息	测试结果符合预期结果，无偏差	
获取函数的所有信息	调用 LustreNode 类的 getNodeMessage 函数	成功获得函数对应的 LustreNode 实例的所有信息的字符串	测试结果符合预期结果，无偏差	

8.2.4 Z3Solver 模块测试

表 8.3 Z3Solver 模块测试表

项目检查	操作	测试结果	与预期结果偏差	备注
将函数语句字符串转为 z3::expr 元素	输入对应函数语句的字符串后调用 parse_expr 函数	成功生成对应的 z3::expr 元素	测试结果符合预期结果，无偏差	
构建 z3::solver 元素	输入对应函数语句的字符串后调用 build_z3solver 函数	成功在输入函数对应的 VarStateList 作为参数后生成对应的 z3::solver	测试结果符合预期结果，无偏差	
分离情况 K 归纳算法测试	输入对应的参数后调用 split_case_k_induction 函数	成功通过分离情况 k 归纳算法验证循环语句的正确性	测试结果符合预期结果，无偏差	验证的语句为具有循环特性的 Lustre 语句
组合情况 K 归纳算法测试	输入对应的参数后调用 combined_case_k_induction 函数	成功通过组合情况 k 归纳算法验证循环语句的正确性	测试结果符合预期结果，无偏差	

8.2.5 综合测试

表 8.4 综合测试表

项目检查	操作	测试结果	与预期结果偏差	备注
Lustre 函数(function)与节点(node)定义的处理	分别输入 function 与 node 开头定义的 Lustre 代码执行程序	两种定义的 Lustre 代码程序均能正确识别并且验证	测试结果符合预期结果，无偏差	
不包含循环特性语句的 Lustre 代码验证	输入不包含循环特性语句的 Lustre 代码后执行程序	程序能正确生成对应的 z3::solver 并且进行验证	测试结果符合预期结果，无偏差	
包含循环特性语句的 Lustre 代码的分离情况 k 归纳算法验证	输入包含循环特性语句的 Lustre 代码后执行程序	程序能正确生成对应的 z3::solver 并且进行验证输出“验证满足”	测试结果符合预期结果，无偏差	
包含循环特性语句的				

Lustre 代码的组合情况 k 归纳算法验证	输入包含循环特性语句的 Lustre 代码后执行程序	程序能正确生成对应的 z3::solver 并且进行验证输出“验证满足”	测试结果符合预期结果，无偏差	
不包含循环特性语句的 Lustre 代码验证失败后的反例生成	输入无法通过验证的不包含循环特性语句的 Lustre 代码后执行程序	程序能输出“验证失败”的相关信息并且给出反例	测试结果符合预期结果，无偏差	
包含循环特性语句的 Lustre 代码分离情况 k 归纳算法验证失败后的的反例生成	输入无法通过验证的包含循环特性语句的 Lustre 代码后执行程序	程序能输出“验证失败”的相关信息并且给出反例	测试结果符合预期结果，无偏差	
包含循环特性语句的 Lustre 代码组合情况 k 归纳算法验证失败后的反例生成	输入无法通过验证的包含循环特性语句的 Lustre 代码后执行程序	程序能输出“验证失败”的相关信息并且给出反例	测试结果符合预期结果，无偏差	

8.3 性能测试

在一系列详细性能测试中，我们的系统在实际应用中的表现非常出色。而在测试过程中，我们特别关注了软件的错误率、精度以及响应时限等重要指标。从测试结果来看，这些指标均达到了预期的设计要求，并且在多个场景中均表现出了良好的性能稳定性。特别是在误差控制方面，我们的软件与我们的预期指标基本一致，没有发现任何显著的偏差。所有数据均处于可接受的误差范围内，这充分证明了我们的软件在性能方面的优越性和稳定性。而软件这样的性能表现也将将能够满足用户在实际工作中的各种需求，为他们提供高效、可靠的软件服务。

8.4 跨平台兼容性性测试

为了确保我们的软件能够在不同的操作系统平台上顺利运行，我们进行了全面的测试。经过在 Windows、Linux、MacOS 等主流操作系统上的测试后确认我们的软件均能够正常运行，并且符合软件用户的需求。这证明了我们的软件具有良好的跨平台兼容性，因此能够在不同的环境中稳定运行，为用户提供一致的使用体验。但同时，我们也注意到，在不同的操作系统中，软件的表现可能会受到一些细微的影响。因此，我们在未来的软件优化中，将充分考虑这些因素，进一步提升软件的可移植性和兼容性，确保用户能够在任何平台上都能够享受到优质的软件服务。

8.5 本章总结

本章着重于系统的测试环节，涵盖合法性检查、功能测试、性能测试以及跨平台兼容性测试等多个方面。我们绘制表格详细的阐述了在测试过程中的结果与注意事项来说明验证软件模块的功能和性能是否达到预期标准。

第九章　总结与展望

本文全面剖析了模型验证软件中归纳推理算法的应用，通过对其实现过程及性能的深入研究，充分凸显了归纳推理算法在模型验证中的核心作用及其实际效果。尤其当处理具备循环特性的 Lustre 语句时，无论是分离情况还是组合情况，K 归纳算法均展现出卓越的验证能力，为复杂系统的形式验证提供了新的视角和实用工具。

虽然 QKind 软件在性能测试与跨平台兼容性测试中均展现出了良好的性能表现和足够的的兼容性，保证了软件可以满足不同用户的需求。但是与此同时我们也同样认识到，随着技术的不断进步和实际应用场景的不断拓展，模型验证软件在未来也一定将面临着更高的挑战和要求。

因此，我们将继续致力于归纳推理算法的优化与改进，去探索更高效、精确的验证算法。同时我们也将关注如人工智能、机器学习等的先进技术在模型验证领域的应用，从而进一步提升验证的自动化程度和准确性。我们相信随着技术的不断进步和创新，模型验证软件将在保障系统安全性、可靠性和性能方面发挥更加关键的作用，为复杂系统的设计和开发提供支持。

同样，我们在未来还将进一步拓展软件的适用范围，尝试将归纳推理算法应用于更多类型的系统和模型中以验证其正确性和实用性。同时我们也将加强与行业的合作，从而推动模型验证软件在实际中的应用和推广，争取为提升整个行业的技术水平和竞争力作出一定贡献。

然而，经过深入细致的分析与综合评估，鉴于个人在精力分配方面的局限、时间窗口的紧迫性、对部分功能需求理解的不足、对社会与市场调研的不全面，以及部分技术应用的不熟练，本项目仍存在进一步提升与优化的空间。为此，我们将从以下几个方面着手，持续推动项目的改进与完善：

1) 设计并开发一套直观易用的软件可视化用户界面，旨在提升用户体验和操作便捷性。

2) 对软件展开全面而深入的稳定性、性能及强壮性测试，确保在各种场景下均能稳定运行。

3) 实现对同步数据流语言 L2C 高阶语法的全面解析，并构建高效稳定的解析器。

4) 扩展软件的语言支持范围，以涵盖更多编程语言需求。

5) 丰富软件的代码验证方式，提供更多样化的代码质量保障手段。

6) 针对 K 归纳算法与其他模型验证算法，开展详尽的实验对比研究，全面评估各算法在优劣性、正确性和可靠性方面的表现。

综上所述，归纳推理算法在模型验证领域的研究与应用上具有重大的意义及广阔的发展前景。但我们仍将持续致力于对已有技术的学习与研究，对未知领域的不懈探索与创新，以求不断完善 QKind 软件的功能，为用户提供更加高效的模型验证服务。

参考文献

[1] 石刚,王生原,董渊,等.同步数据流语言可信编译器的构造[J].软件学报,2014,25(02):341-356.DOI:10.13328/j.cnki.jos.004542.

[2] 康跃馨,甘元科,王生原.同步数据流语言可信编译器 Vélus 与 L2C 的比较[J].软件学报,2019,30(07):2003-2017.DOI:10.13328/j.cnki.jos.005755.

[3] Caspi P, Pilaud D, Halbwachs N, Plaice J. Lustre: A declarative language for programming synchronous systems. In: Proc. of the 14th ACM Symp. on Principles of Programming Languages (POPL'87). Munchen, 1987. 178-188.

[4] Halbwachs N, Caspi P, Raymond P, Pilaud D. The synchronous dataflow programming language LUSTRE. Proc. of the IEEE, 1991, 79(9): 1305－1320 . [doi:10.1109/5.97300]

[5] Shang S, Gan YK, Shi G, Wang SY, Dong Y. Key Translations of the Trustworthy Compiler L2C and Its Design and Implementation[J]. Journal of Software, 2017, 28(5): 1233-1246(in Chinese).http://www.jos.org.cn/1000-9825/5213.htm

[6] 兰林,马权,侯荣彬,等.Lustre 语言可信代码生成器研究进展[J].仪器仪表用户,2020,27(05):68-72.

[7] YANG Ping, WANG Sheng-yuan. Survey on Trustworthy Compilers for Synchronous Data-flow Languages[J]. Computer Science, 2019, 46(5): 21-28. https://doi.org/10.11896/j.issn.1002-137X.2019.05.003

[8] 张智慧,冀建伟.核安全级控制算法描述语言的可信编译研究[J].自动化仪表,2021,42(S1):106-111.DOI:10.16086/j.cnki.issn1000-0380.2021030117.

[9] Champion, A., Mebsout, A., Sticksel, C., Tinelli, C. (2016). The KIND 2 Model Checker. In: Chaudhuri, S., Farzan, A. (eds) Computer Aided Verification. CAV 2016. Lecture Notes in Computer Science(), vol 9780. Springer, Cham. https://doi.org/10.1007/978-3-319-41540-6_29

[10] 谢小小 XH.CMake 入门实践(一) 什么是 cmake[EB/OL].[2024.4.19].http://t.csdnimg.cn/xjFyJ.

[11] 孙凯. 基于嵌入式平台的深度学习人脸识别技术研究[D].南京邮电大学,2023.DOI:10.27251/d.cnki.gnjdc.2022.000364.

[12] Edmund M. Clarke;Thomas A. Henzinger;Helmut Veith;Roderick Bloem.Handbook of Model Checking[J].Springer, Cham,2018.

[13] Halbwachs N.;Caspi P..The synchronous data flow programming language LUSTRE[J].Proceedings of the IEEE,1991(9).

[14] Lina Marsso.Specifying a Cryptographical Protocol in Lustre and SCADE[J].Electronic Proceedings in Theoretical Computer Science,2020.

[15] Temesghen Kahsai;;Cesare Tinelli.PKind: A parallel k-induction based model checker[J].Electronic Proceedings in Theoretical Computer Science,2011.

[16] 卫俊杰. Lustre 语言安全性及活性模型检测工具的研究与实现[D].华东师范大学,2024.DOI:10.27149/d.cnki.ghdsu.2023.000630.

[17] Bošnački, D., & Wijs, A. (2018). Model checking: recent improvements and applications. International journal on software tools for technology transfer : STTT, 20(5), 493–497. https://doi.org/10.1007/s10009-018-0501-x

[18] Merz, S. (2000). Model checking: A tutorial overview. Summer School on Modeling and Verification of Parallel Processes, 3-38.

[19] 朱雪阳,张文辉,李广元,吕毅,林惠民.模型检测研究进展[R].北京:中国科学院软件研究所 计算机科学国家重点实验室,2019.

[20] 林惠民,张文辉.模型检测:理论、方法与应用[J].电子学报,2002(S1):1907-1912.

[21] 陈宇星.模型检测技术在程序内存泄漏检测中的应用[J].现代计算机(专业版),2017(04):53-57.

[22] 侯刚,周宽久,勇嘉伟,等.模型检测中状态爆炸问题研究综述[J].计算机科学,2013,40(S1):77-86+111.

[23] Donaldson A F, Haller L, Kroening D, et al. Software verification using k-induction[C]//International Static Analysis Symposium. Springer, Berlin, Heidelberg, 2011: 351-368. https://doi.org/10.1007/978-3-642-23702-7_26

[24] 刘帅, 魏峰玉, 黄怡桐, 江建国. k-归纳模型检测结果的认证器[J]. 应用数学进展, 2022, 11(11): 7729-7737. https://doi.org/10.12677/AAM.2022.1111817

[25] 张赛. 基于静态分析的 c 语言程序安全验证方法研究[D].天津理工大学,2023.DOI:10.27360/d.cnki.gtlgy.2023.000210.

[26] 曹文平,谷琼,宁彬.基于循环不变式的循环结构教学研究[J].电脑知识与技术,2023,19(06):121-122+143.DOI:10.14004/j.cnki.ckt.2023.0289.

[27] 黄强,曾庆凯.基于 SSA 中间表示的源代码信息流分析[J].计算机工程,2009,35(13):166-168+171.

谢 辞

至此，我的本科生涯即将画上圆满的句号，此刻，我满怀感激之情，向南华大学及计算机学院的全体教师致以最崇高的敬意。感谢您们用专业的知识和无私的奉献，为我解疑答惑，指明方向。同时，我也要向我的同学们表示衷心的感谢，是你们的陪伴与支持，让我在这段求学之路上不再孤单。在此，我向所有的老师和同学们送上最真挚的祝福，愿你们身体健康，万事如意，生活美满。