硕士学位论文

基于决策树的带参协议自动化验证方法

The Automated Verification Method of Parametric Protocols Based on Decision Tree

作者：曹泰丰

导师：魏小涛

北京交通大学

2019 年 5 月

学位论文版权使用授权书

本学位论文作者完全了解北京交通大学有关保留、使用学位论文的规定。特授权北京交通大学可以将学位论文的全部或部分内容编入有关数据库进行检索，提供阅览服务，并采用影印、缩印或扫描等复制手段保存、汇编以供查阅和借阅。同意学校向国家有关部门或机构送交论文的复印件和磁盘。学校可以为存在馆际合作关系的兄弟高校用户提供文献传递服务和交换服务。

（保密的学位论文在解密后适用本授权说明）

学位论文作者签名：　　导师签名：

签字日期：　　年　　月　　日　　签字日期：　　年　　月　　日

学校代码：10004 密级：公开

北京交通大学

硕士学位论文

基于决策树的带参协议自动化验证方法

The Automated Verification Method of Parametric Protocols Based on Decision Tree

作者姓名：曹泰丰 学　　号：16121728

导师姓名：魏小涛 职　　称：副教授

学位类别：工学硕士 学位级别：硕士

学科专业：软件工程 研究方向：软件工程

北京交通大学

2019 年 5 月

致谢

研究生的学习生涯即将结束，在这三年时间中，我学习到了许多关于软件工程的专业知识，也了解到很多关于软件工程的前沿动态，提升了自身的专业水平。与此同时，我也结

交了许多志同道合的同学和朋友。通过与他们共同探讨学术问题，帮助我克服了一道道难关。在论文即将完成之际，我要对在我研究期间给予我关心和帮助的老师、同学和朋友表达诚挚的谢意。

首先，衷心感谢我的导师魏小涛老师。在我的研究生学习期间，魏小涛老师对我耐心指导，对我的学术研究和毕业论文都提出了许多指导性的意见。魏老师严谨治学的态度和勤劳刻苦的工作方式都深深地影响了我。老师不仅在学习上教导我，而且还经常和学生们分享自己的人生感悟，充分尊重学生的兴趣与发展，给予学生们充分的自由发展空间。让我受益匪浅。

其次，特别感谢 ParaVerifier 的开发者李勇坚老师。通过李老师对 ParaVerifier 的详细讲解，让我了解带参协议验证中的难点与主流研究方向，帮助我突破带参协议研究中的瓶颈。

第三，感谢在研究生期间，所有指导与帮助过我的其他老师们、同学们和朋友们。正是有大家的陪伴与帮助，让我每天都能够全力投入于学习研究当中，使我的研究生生活更加丰富多彩。

最后，感谢我的家人，感谢您们这三年中对我的关心与帮助，您们一直是我坚强的后盾。正是因为您们对我在学习和生活方面的教导，使我少走人生中的弯路，顺利完成学业。

毕业不是结束，而是人生中一个新的起点，我将牢记北京交通大学“知行”的校训，在人生的道路上全力奔跑。

摘要

协议广义上指网络协议，协议的功能是用于描述模块间的通讯规则，本文中，协议狭义地指由卫式命令语言（Guarded Command Language）描述的协议程序。

带参协议是参数化后的协议，是协议的推广形式。带参协议应用广泛，如互斥协议、缓存一致性协议、网络安全通信协议等都属于带参协议。带参协议验证是对带参协议逻辑正确性进行校验的过程，其目的是为了保障带参协议满足设计要求。协议的传统形式化验证方法，是通过枚举协议的所有可达状态，来依次验证协议的设计要求是否在每个状态下均成立，然而，这种验证方式不适合带参协议验证。其主要原因是：一是带参协议拥有无数个实例，而每个实例都等价于一个协议，直接验证所有实例非常困难；二是带参协议实例的状态数量随参数增加成指数级增长，造成状态爆炸问题，导致大参数带参协议实例难以验证。

为了解决协议验证方法难以应用于带参协议验证的问题，带参协议验证专家提出了各自的解决方法，如抽象方法、截止方法和归纳证明方法等等。其中，归纳证明方法被认为是最可行的解决方案之一。使用归纳证明方法验证带参协议需要寻找辅助不变式，传统的辅助不变式寻找方式为人工推导，这种方法只适用于小型带参协议的验证，对于大型带参协议，人工推导方法不仅实现困难而且容易出错，这使得归纳证明方法难以推广。

为了解决归纳证明方法中辅助不变式的寻找问题，本文提出了基于决策树的不变式判断方法，并结合现有的带参协议归纳证明技术，提出了基于决策树的带参协议自动化验证方法。基于决策树的带参协议自动化验证方法是首个将决策树方法应用于带参协议验证的方法。该方法可以高效地寻找辅助不变式并完成带参协议的归纳证明。基于决策树的带参协议自动化验证方法主要有以下创新点：

(1)提出了使用带参协议实例的对称削减可达状态集合作为决策树训练数据的方法。使用这种训练数据不仅使决策树能够准确判断候选不变式，而且减少了决策树的生成和判断候选不变式所需要的内存与时间。

(2)提出了针对带参协议实例的决策树算法。这种算法以 ID3 算法为基础，通过修改决策树叶子节点的生成条件与生成方法，使决策树包含了更多有关带参协议实例的信息，从而保证决策树能够准确无误地判断候选不变式。

(3)提出了使用决策树路径判断候选不变式的方法。这种方法的原理是利用决策树路径与候选不变式的关系，使用可达状态类别判断候选不变式是否为带参协议实例的不变式。

基于决策树的带参协议自动化验证方法针对的带参协议类型是缓存一致性协议。试验证明，基于决策树的带参协议自动化验证方法能够应用于包含 Flash 协议

的多种典型带参协议，并且和 ParaVerifier 方法相比，在验证大型带参协议时，时间消耗与内存占用方面均有明显优势。

关键词：带参协议验证；归纳证明方法；不变式；决策树

ABSTRACT

In broad sense, the protocol refers to the network protocol. The function of the protocol is to describe the communication rules of each module. In this paper, protocol narrowly refers to the protocol program described by Guarded Command Language.

Parametric protocol is a parameterized protocol, which is the extension form of the protocol. Parametric protocols are widely used, such as mutually exclusive protocols, cache consistency protocols, network security communication protocols and so on. Parametric protocol verification is the process of verifying the logical correctness of parametric protocol. Its purpose is to ensure that the parametric protocol meets the design requirements. The traditional formal verification method of protocol is to verify whether the protocol requirements are valid in each state by enumerating all reachable states of the protocol. However, this method does not work well for the parametric protocol verification. The reasons are divided into two points. First, the parametric protocol has numerous instances, and each instance is equivalent to a protocol. It is very difficult to verify all instances directly. Secondly, the state space of the parametric protocol instances increases exponentially with the increase of parameters, resulting in the state explosion problem, which makes it difficult to verify the large parameter parametric protocol instances.

In order to solve the problem that protocol validation methods are difficult to apply to parametric protocol verification, experts of parametric protocol verification put forward their own solutions, such as abstract method, cut-off method, inductive proof method and so on. Among them, inductive proof method is one of the most feasible solutions. However, using inductive proof to verify the parametric protocol requires finding auxiliary invariants. The traditional method of finding auxiliary invariants is manual derivation. This method can only be applied to small-scale parametric protocol. For large-scale parametric protocol, the manual derivation method is not only difficult to implement but also error-prone, which makes it difficult to generalize the inductive proof method.

In order to solve the problem of finding auxiliary invariants in inductive proof method, this paper proposes an invariant judgment method based on decision tree, and combines with the existing inductive proof technology of parametric protocol, proposes an automatic verification method of parametric protocol based on decision tree. The automated verification method of parametric protocols based on decision tree is the first

method to apply decision tree method to parametric protocol verification. This method can efficiently find auxiliary invariants and complete the inductive proof of the parametric protocol. The automated verification method of parametric protocols based on decision tree includes the following main innovations:

(1)A method of using symmetrically reduced reachable state sets with reference protocol instances as training data of decision tree is proposed. Using this training data not only enables decision tree to accurately judge candidate invariants, but also reduces the memory and time required to generate decision tree and to judge candidate invariants.

(2)A decision tree algorithm for instance of parametric protocols is proposed. This algorithm is based on ID3 algorithm. By modifying the generating conditions and methods of the leaf nodes, the decision tree contains more information about the instance of the parametric protocols, to ensure that the decision tree can accurately and correctly judge the candidate invariants.

(3)A method of judging candidate invariants by decision tree path is proposed. The principle of this method is to use the relationship between the decision tree path and candidate invariants, and use the category of reachable state to judge whether candidate invariants are invariants of parametric protocols.

The automated verification method of parametric protocols based on decision tree aims at cache consistency protocol. Experiments show that the automated verification method of parametric protocols based on decision tree can be applied to many typical parametric protocols including Flash protocol. Compared with the ParaVerifier method, it has obvious advantages in time consumption and memory consumption when validating large-scale parametric protocols.

KEYWORDS：Parametric Protocol Verification; Inductive Proof Method; Invariant; Decision Tree

目录

1 引言

随着工业的发展与进步，带参协议的应用变得越发广泛，如缓存一致性协议、网络通讯协议、系统安全协议等均属于带参协议。为了保证带参协议满足设计要求，带参协议验证专家们提出了多种带参协议验证方法，这使带参协议验证成为了活跃的研究领域。许多带参协议验证方法都需要寻找不变式，而寻找带参协议的不变式一直是一个难点。在带参协议验证领域中，不变式的传统寻找手段是人工推导，但这种方法难以应用于大型带参协议，于是很多研究致力于自动寻找不变式。如今，机器学习方法的应用与发展为自动寻找带参协议不变式提供了新的可能。虽然在带参协议验证领域中，鲜有使用机器学习方法自动寻找不变式的例子，但在循环程序验证领域，决策树方法已被应用于寻找不变式。本章主要介绍基于决策树的带参协议自动化验证方法的研究背景与意义，其中包括带参协议与带参协议实例的介绍、基于决策树的带参协议自动化验证方法的相关工作介绍、本文的主要工作和本文的组织结构。

1.1 带参协议与带参协议实例

在本文中，协议狭义地指由卫式命令语言（Guarded Command Language）[1]描述的协议程序。这种协议程序用于模拟计算机协议的工作方式。卫式命令语言是一种用于定义谓词语义转换的语言，它的简单性能够使霍尔逻辑更容易证明协议的正确性。

带参协议是参数化的协议，是协议的推广形式。带参协议可以实例化，当确定带参协议的参数取值后可以得到带参协议实例。带参协议实例等价于具体的协议。

带参协议拥有无数个实例。这是因为带参协议中包含两种主体，它们分别是数量由参数而定的相同结构同时并发执行主体和固定数量的结构不同主体。因此，对于不同参数的带参协议实例，它们包含的主体数量不同。假设把带参协议定义为 P（N），其中 N 是带参协议的参数，那么 N 的每一种自然数取值，如 1，2，3，……，都会对应一个带参协议实例，因此，带参协议拥有无穷多个实例。

带参协议实例分为两部分：第一部分是状态变量；第二部分是规则。在带参协议实例中，带参协议实例状态是各个状态变量的取值组合，因此各个状态变量的所有取值组合构成了带参协议实例的状态空间。带参协议实例的状态分为两类。第一类是可达状态，这种状态指在带参协议实例运行中出现的状态变量的取值组合。第二类是不可达状态，这种状态指可达状态以外的带参协议实例状态。由于规则的限

制，不可达状态的数量往往远超可达状态的数量。

规则是带参协议实例的主要内容，它规定了带参协议实例状态间的转化条件，其中状态间的转化也被称为状态迁移。在带参协议实例中，规则限制了各个主体的交互方式，并以此使带参协议实例满足某种要求。

规则可分为初始化规则与一般规则。初始化规则定义了带参协议实例的初始状态集合。一般规则定义了带参协议实例的迁移规则。为了更加形象地介绍带参协议与带参协议实例，下文以带参协议 German[2]为例进行举例介绍。

German 协议是一种基于目录的缓存一致性协议，其特点是通过维护主节点 Home 中的目录，保证各个缓存的缓存一致性。German 协议包含两种主体，第一种是拥有目录的主节点 Home。该主体属于固定数量的结构不同主体，不会被参数影响。第二种是拥有可执行功能的 Client 节点。该节点是数量由参数而定的相同结构同时并发执行主体。

在 German 协议中，协议规则规定着主节点 Home 与各个 Client 节点的交互方式，例如主节点 Home 会接收 Client 节点的数据缓存副本请求。接收请求后，主节点 Home 会根据带参协议状态执行相应的处理，例如读写主存储器中的数据、对内存进行写回或写穿操作、标记节点的缓存副本无效（Invalid）等等。

German 协议不能直接应用于某个系统。因为系统的配置是固定的，需要根据系统的结构确定 Client 节点的数量，将 German 协议的实例应用于系统。

1.2 相关工作

基于决策树的带参协议自动化验证方法是第一个将决策树方法应用于带参协议验证的方法。这种方法的设计目的是为了探索决策树方法与带参协议验证的结合方法，为带参协议的不变式自动寻找提供新的方法与思路。为了介绍与决策树的带参协议自动化验证方法相关的研究工作，本章将从带参协议验证的主流验证方法和寻找不变式与决策树之间的关系两方面进行介绍。

1.2.1 带参协议验证的主流方法

带参协议验证是指验证带参协议是否满足设计要求。虽然带参协议是协议的参数化，但是协议验证方法难以直接用于验证带参协议。传统的协议验证方法是枚举并检测协议的所有可达状态。这种可达性分析方法可分为两类：第一类是显式状态枚举法，这种方法将所有可达状态显式枚举[3,4]并单独存储于哈希表中；第二类是符号化方法[5,6]，这种方法通过 BDDs（Binary Decision Diagrams）方法，用命

题公式表达可达状态空间[7]，或使用一些非规范表达式（如 AIGs 方法[8]）来表示可达状态空间。例如 SAT 技术能够将限制行为步数的模型编码为 SAT 的实例[9]。然而在将这些方法应用于验证带参协议时，都遭遇了带参协议实例过多问题与可达状态空间爆炸问题[10]。

为了解决上述问题，带参协议验证专家们提出了各自的解决方法，其中主流的解决方法可被分为截止方法、组合验证方法、抽象方法和归纳证明方法。

1.2.1.1 截止方法

截止方法（cutoff）的中心思想是限制带参协议验证中参数的取值。这种方法通过证明存在阈值 M，使带参协议验证简化为多个带参协议实例的验证。其中，这些带参协议实例包括参数取值为 M 的实例和参数小于 M 的实例。例如 Emerson 等人提出的针对简单广播协议的规约思想[11]，这种方法最早运用了截止方法。Bouajjani 等人根据先进先出队列的特性，优化了截止方法，完成了带参协议的验证[12]。Alan hu 等人针对简单的目录协议并根据规约思想完成了形式化验证[13]。Daniel Kroening 等人改进了截止方法并对多线程协议的动态性质进行了验证[14]。

截止方法解决了带参协议验证中实例过多的问题，但是确定阈值 M 是截止方法的难点。阈值 M 依赖于人工寻找，并且截止方法验证结果的正确性依赖于非形式化的纸面证明，这不仅要求证明工作者对带参协议内容十分熟悉，而且难以应用于复杂的带参协议。

1.2.1.2 组合验证方法

组合验证方法的中心思想是拆分再验证（divide-and-conquer）。这种方法通过将带参协议验工作证拆分为多个简单的验证工作，从而降低了带参协议验证的难度。Mcmillan K L 以针对时间顺序的超限归纳法为理论基础，通过结合显式状态枚举中的对称规约、时态划分等技术，提出了组合验证技术[15]。通过应用该技术，Mcmillan K L 编写了相应的验证程序 Cadence SMV，并首次验证了 Flash 协议的活性性质。

组合验证技术降低了带参协议验证的难度，然而使用组合验证技术需要验证者掌握多种专业知识，如带参协议的语义内容，Cadence SMV 的各种归约机制等等。此外，应用组合验证方法还需要提供额外的辅助不变式，以用于分解带参协议验证工作。

1.2.1.3 抽象方法

抽象方法的中心思想是构造一种抽象系统以简化带参协议验证。其中，带参协议拥有的性质，抽象系统也拥有。带参协议没有的性质，抽象系统也没有。相比验证带参协议，验证抽象系统更为简单。例如，抽象系统可以根据验证的需要，将所有的无用的参数简化为一个抽象参数，从而降低了抽象系统的验证难度。

抽象系统是带参协议的映射，其中存在抽象函数 f（x），使抽象系统状态与带参协议状态一一对应，并且使抽象后的状态间转换关系保持不变。

按照抽象对象划分，抽象方法可分为针对参数抽象与卫式加强的抽象方法和针对谓词的抽象方法。

针对参数抽象与卫式加强的抽象方法最早由 Chou C T 等人提出[16]。该方法的特点是参数抽象与卫式加强。参数抽象是指保留带参协议中所需的参数，将不需要的参数抽象为一个抽象参数，以得出相应的抽象系统。卫式加强是指强化规则的进入条件。针对参数抽象与卫式加强的抽象方法根据人工构造反例与找到的辅助不变式，采取逐步精化的方式改进抽象系统，使抽象系统能够模拟带参协议。Chen 等人应用针对参数抽象与卫式加强的抽象方法验证了多核缓存协议与多层缓存协议[17,18]。Clarke E 等人对参数的抽象方法进行了改进，提出了环境抽象方法并将其应用于并发系统的验证[19,20]。

以上方法中，辅助不变式需要人工寻找，这不仅需要验证者熟悉带参协议的具体内容，而且难以应用于大型协议。对此 Sethi D 等人提出了“FLOW”概念，并将其与 CMP 方法结合，降低了寻找辅助不变式的难度[21,22]。Yi L 等人试图从带参协议实例中寻找辅助不变式，该方法从可达状态空间中导出布尔公式，并将布尔公式作为辅助不变式用于卫式加强，实现了 German 协议的自动化验证。但是这种方法无法验证 Flash 协议[23,24]。

针对谓词的抽象方法最早由 Qadeer C F S 提出[25]。谓词抽象是指利用谓词表达带参协议的状态空间，从而得出相应的抽象系统。寻找用于抽象的谓词需要相应的谓词发现技术。因此，针对谓词的抽象方法通常与相应的谓词发现技术一起使用。针对谓词的抽象方法通常应用于自动验证非有穷状态系统的安全性质。例如 Baukus K 等人使用针对谓词的抽象方法验证了 German 协议[26]。Das S 等人使用针对谓词的抽象方法验证了简化版 Flash 协议的安全性质[27]。针对谓词的抽象技术的难点在于寻找合适的抽象谓词。一般而言，抽象谓词的发现依赖于人工寻找，因此难以应用于大型带参协议。

1.2.1.4 归纳证明方法

归纳证明方法是指将带参协议的结构以及功能用公式表示出来，然后使用基于逻辑的推理系统去证明结构定义公式可以推出功能定义公式。由于高阶逻辑系统表达能力强大，因此它能够描述带参协议的状态变量、规则和设计要求。如果把带参协议的参数看作符号，则可以使用数学中的归纳法来证明带参协议满足设计要求。

带参协议的归纳证明方法源于定理证明中的归纳证明方法，根据所选的定理证明器的不同，带参协议的归纳定义与证明过程也可能不同。例如 Park S 等人使用 PSV 定理证明器，通过分析 Flash 协议的语义构造辅助不变式，首次完成了 Flash 协议的安全性质验证[28]。Pnueli A 等人使用 TLV 定理证明器，基于带参协议实例的可达状态空间的 BDDs 符号化表示提出了“不可见的不变式”的概念，并以此辅助计算辅助不变式，完成了带参协议的验证[29]。Pandav 等人使用 UCLID 定理证明器，提出了“辅助不变式模式”概念，以用于辅助人工寻找辅助不变式，并手工完成了 German 协议、简化 Flash 协议的验证[30]。Conchon S 等人使用 Coq 定理证明器，提出了向后搜索算法，根据带参协议实例的可达状态集合自动计算辅助不变式，完成了带参协议的证明[31, 32]。李勇坚等人使用 Isabelle 定理证明器，提出了从带参协议的小参数实例中自动寻找辅助不变式的方法，完成了包括 Flash 协议的典型带参协议的自动化验证。与以往的研究相比，这种方法发现的辅助不变式的含义更加简单易懂[33, 34]。

从前人对归纳证明方法的研究中不难看出，归纳证明方法依赖于对辅助不变式的寻找。因此，如何高效寻找辅助不变式是归纳证明方法需要解决的主要问题之一。

1.2.2 不变式与决策树

在带参协议验证中，不变式是带参协议运行过程中时刻保持为真的谓词公式。如上文提到的辅助不变式，以及带参协议的某些设计要求均属于不变式。在带参协议验证领域中，不变式常常被用于解决带参协议验证问题。

带参协议验证领域中，许多研究尝试枚举带参协议实例的可达状态空间[16,23,29,30,31,32,35,36,37,38,39]。然而对于大型带参协议实例，这些方法在对时间和内存的消耗方面上都到达了极限。为了解决上述问题，许多包含寻找不变式的解决方法被提出。例如 Mcmilan 等人提出的“参数抽象和卫式加强”技术[15]。Talupur 等人提出的使用信息流加速的抽象技术[22]。Xiaofang Chen 等人提出的元-循环假设与保证技术[4

[0]。然而，这些方法均需要手工寻找的辅助不变式。这不仅需要对带参协议有深刻的理解，而且验证大型带参协议时容易出现错误[16]382-384。

为了高效获取不变式，自动寻找不变式一直是一个活跃的研究领域。Daikon[41]是自动寻找不变式方法中的一个早期的突出例子，它使用布尔连接学习技术，从动态测试的程序配置中自动寻找不变式。Arons 等人在有限系统实例中自动寻找辅助不变式，帮助对目标不变式的归纳验证[37]226-229。虽然这种方法实现了不变式的自动寻找，但是找到的不变式是由 TLV（一种基于 BDD 的 SMV 模型检测器的变体）计算的 BDDs[42]，这种公式不仅是一种“原始”的布尔公式，而且让人难以理解。受到上述 Arons 等人的方法与“参数抽象与卫式加强”方法的启发，Yi Lv 等人将带参协议中的一个小参数实例作为“引用实例”来自动寻找不变式[23]33-36。Conchon 等人开发一种基于 SMT（Satisfiability Modulo Theories）的模型检测程序 Cubicle，并提出了一种与 Cubicle 配套的新算法 BRAB。这种方法根据带参协议实例的状态集合自动寻找不变式[31]719-721。应用这种方法实现的验证工具 Cubicle 是第一个可以自动验证 Flash 协议安全性质的工具。然而，Conchon 等人的方法在推广方面十分困难，因为这种方法不能应用于其他定理证明器，如 Isabelle[43]或 Coq[44]。李勇坚等人提出了一种从带参协议的小参数实例中自动寻找不变式的方法，并在定理证明器 Isabelle 中构造了一种参数化形式证明。与以往的研究相比，这种方法发现不变式的形式更为简单易懂[34]11-14。李勇坚、曹嘉伦等人基于关联规则的学习技术自动寻找不变式，建立了与 CMP 抽象技术相关的学习框架，验证了一些典型的带参协议[45]。

使用决策树方法寻找不变式是一种新颖的思想，它将机器学习方法与形式化验证方法相互结合，为寻找不变式提供了新思路。目前，决策树方法已被应用于寻找循环程序中的不变式。

在使用决策树方法寻找循环程序不变式的方法中，最成功的是 Garg P 等人提出的 ICE 方法[46,47]。该方法提出了程序状态间的蕴含关系，并将蕴含关系、好状态和坏状态共同用于训练决策树。该方法对经典的决策树方法[48,49]进行了扩展，使这些决策树方法可以使用蕴含关系判断未知分类的训练数据。提出蕴含关系的目的是为了解决训练数据无法分类的问题。通过对蕴含关系的学习，扩展的决策树算法可以动态划分之前无法分类的训练数据，从而避免随机划分不可分类的训练数据。ICE 方法是一种逐步精华的决策树方法，通过不断完善训练数据，使决策树具有很高的精度。通过对决策树数据结构的改动，该工作实现了利用统计测度构建决策树。通过将得到的决策树归纳为谓词公式，ICE 方法找到了所需的不变式。实验结果证明，对于一大组程序而言，ICE 方法是可收敛且有效的。

在 Siddharth Krishna 等人提出的寻找不变式方法中，决策树方法也被用于来寻

找循环程序中的不变式[50]。其中，该方法使用八边形域[51]表示训练数据的属性，并且生成的决策树是二叉树。决策树的训练数据由好状态与坏状态组成。该方法除了应用逐步求精的思想训练决策树外，还加入了截止思想。截止思想通过计算所需训练数据数量的最大值，减少了逐步求精的循环数量。试验证明，该方法可以从典型的 C 程序中寻找循环不变式并完成验证。值得一提的是，在该方法中训练状态的最大值思想效果明显，在绝大多数的程序中，只需一步精化便可以得到精度良好的决策树。

通过应用决策树方法，循环程序实现了自动寻找不变式方法，而在带参协议验证领域，决策树却鲜有应用。因此，在不变式寻找趋势是由人工方式向自动方式转变的背景下，使用决策树方法寻找带参协议中的不变式是一个值得探索的领域。

1.3 本文工作

为了探索在带参协议验证中使用决策树寻找不变式的方法，提升带参协议验证的效率，本文提出了一种基于决策树的不变式判断方法，并结合 ParaVerifier 方法[34]，提出了一种基于决策树的带参协议自动化验证方法。这种方法能够自动且高效地寻找带参协议的归纳不变式并完成带参协议的归纳证明。

基于决策树的带参协议自动化验证方法应用的对象是由卫式命令语言描述的缓存一致性协议。这种带参协议仅由变量与规则构成，不包含函数等其他结构，例如通讯协议中的加密函数等等。本文的工作可以分为以下几点：

(1)提出了一种决策树训练数据的新取法。该方法使用带参协议实例的对称削减可达状态集合作为训练数据。这种训练数据可以在保证决策树精度的同时，减少决策树生成与判断候选不变式所需要的时间与内存消耗。

(2)提出了针对带参协议实例的决策树算法。该算法以 ID3 算法为基础，通过修改决策树叶子节点的生成条件与生成方法，使决策树包含了更多有关带参协议实例的信息，保证决策树能够准确判断候选不变式。

(3)提出了基于决策树的不变式判断方法。该方法通过分析决策树路径与候选不变式之间的关系，使用决策树路径判断候选不变式是否为带参协议实例的不变式。与一流的模型检测软件（NuSMV、Murphi）相比，在验证大型带参协议实例时，基于这种方法实现的程序能够有效减少候选不变式判断所需要的时间消耗与内存占用。

(4)提出一种基于决策树的带参协议自动化验证方法。基于决策树的带参协议自动化验证方法是基于决策树的不变式判断方法能与 ParaVerifier 方法结合的产物。其中，基于决策树的不变式判断方法可以高效判断 ParaVerifier 方法得出的候选辅

助不变式，从而减少了归纳证明过程所需要的时间消耗与内存占用。

本文的工作是第一个将决策树方法应用于带参协议验证的工作。其中，使用决策树方法判断候选不变式是简单易行的，这种方法不需要操控复杂的数据结构或具有复杂语义的程序控制流，仅需遵循决策树的建立策略。这样得出的决策树可以高效分类可达状态空间，并使用可达状态类别判断候选不变式。试验证明，基于决策树的带参协议自动化验证方法能够应用于包含 Flash 协议的多种典型带参协议，并且和基于 ParaVerifier 方法的带参协议自动验证工具 ParaVerifier 相比，在验证大型带参协议时，时间消耗与内存占用方面均有明显优势。

1.4 本文组织结构

本文一共分为六个章节，它们分别是引言、基础理论、基于决策树的带参协议自动化验证方法、基于决策树的带参协议自动化验证方法的实现、试验结果和总结与展望。本文中各个章节的关系如图 1-1 中所示。其中，前两章的主要内容是他人所提出的理论与所做的工作。而本文的主要工作体现在第 3 章到第 6 章。下面简要介绍各章节的内容。

第一章是引言。本章节主要介绍本文的研究背景与意义。其中包括带参协议与带参协议的简要介绍、带参协议验证的主流研究方法、不变式寻找与决策树方法之间存在的联系、本文所做的主要贡献以及本文的组织结构。

第二章是基础理论。本章节主要介绍在本文的研究中，用到的他人理论基础。其中包括带参协议验证的定义、ParaVerifier 方法的简要介绍和 Murphi 与对称削减方法的简要介绍。

第三章是基于决策树的带参协议自动化验证方法。本章主要介绍基于决策树的带参协议自动化验证方法的理论与设计。其中包括关于基于决策树的不变式判断方法的介绍与介绍基于决策树的不变式判断方法与 ParaVerifier 方法的结合方法。

第四章是基于决策树的带参协议自动化验证方法的实现。本章主要介绍如何实现基于决策树的带参协议自动化验证方法，其中包括对称削减可达状态集合收集器的实现方法与基于决策树的不变式判断服务器的实现方法。

第五章是实验结果。本章主要介绍基于决策树的带参协议自动化验证方法的试验结果，并通过与 ParaVerifier 实验结果的对比，说明基于决策树的带参协议自动化验证方法的优势。

第六章是总结与展望。本章主要总结基于决策树的带参协议自动化验证方法的创新点，并对下一步工作进行简要介绍。

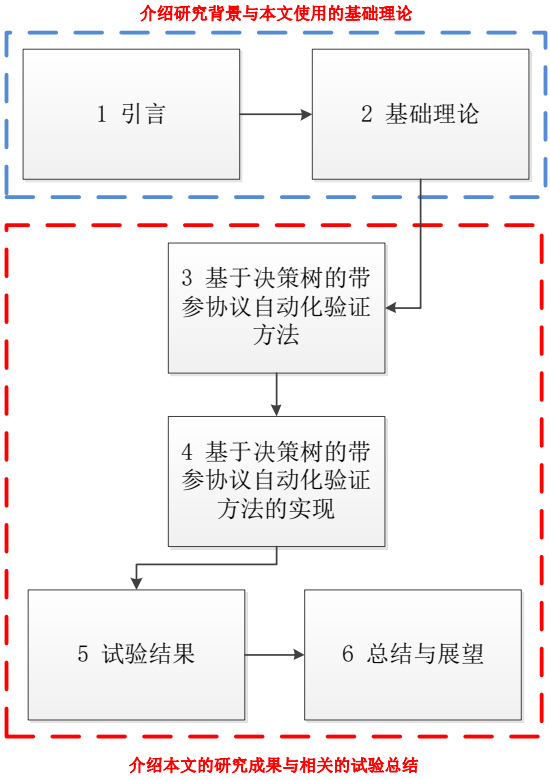

图 1-1 文章的组织结构

Figure 1-1 Structure of Articles

1.5 本章小结

本章主要介绍基于决策树的带参协议自动化验证方法的研究背景与意义，其中包括简单介绍带参协议与带参协议实例、介绍了与基于决策树的带参协议自动化验证方法相关的工作，阐述了本文的主要工作内容和组织结构。

2 理论基础

本章主要介绍在基于决策树的带参协议自动化验证方法中用到的他人理论知识。本章内容包括带参协议验证中的名词定义、ParaVerifier 方法的简介和 Murphi 与对称削减方法的介绍。

2.1 带参协议验证

带参协议验证是确认带参协议是否满足设计要求的过程，这需要证明在任意参数下，带参协议实例均满足带参协议的设计要求。本章将从节点、状态、协议、带参协议、带参协议验证问题这几个方面介绍相关定义。

2.1.1 节点

带参协议的应用环境是计算机中的通讯网络。在该网络中，每个接受或发送信息的单位被称为节点，而每个节点拥有相同的带参协议（实例）以便规范每个节点的工作方式。因此通讯网络中的节点对应着带参协议中的主体。在本文中，称数量由参数而定的相同结构同时并发执行主体对应的节点为称等价节点，称固定数量的结构不同主体对应的节点为称非等价节点。在带参协议实例中，每个等价节点会被赋予一个独立的标号。一般情况下，如果等价节点的数量为 N，则可将这些等价节点分别命名为 1，2，... ，N。这些标号会出现在带参协议实例的数组状态变量下标中，用于代表数组元素描绘的节点。

2.1.2 状态

在协议（带参协议实例）运行过程中，协议状态（states）用于描述某一时刻协议的运行状态。协议状态是通过状态变量表示的，其中每种状态变量的取值组合决定一种协议状态。在本文中，假设协议拥有有限的状态变量集合 $V = \{v_1, v_2, \dots, v_m\}$，并且所有状态变量的取值范围在有限的集合 D 中。那么协议状态可被看作是 V 到 D 的一种映射。关于集合 V，$v_1, v_2, \dots, v_m$代表协议中的所有状态变量，这些状态变量分为数组类型和非数组类型。数组类型状态变量代表等价节点的状态，数组下标代表对应节点的名称。换句话说，数组类型状态变量代表节点中的一个局部变量（local variables）。非数组类型状态变量代表非等价节点的状态。换句话说，非数组类型状态变量代表着整个协议的全局变量（global variables）。通过使用这些状态

变量，协议能够定义一阶表达式 e 和谓词表达式 f 。

在本文中，用$\mathrm{s}(v_i)$代表在协议状态 s 中某一状态变量v_i的取值。用$\mathrm{s}(e)$代表一阶表达式 e 在状态 s 下的取值结果。用$\mathrm{s} \mid = f$ 代表在状态 s 下，f 等价于真（true）。用$\mid = f$代表在协议中的任意状态 s 均满足$\mathrm{s} \mid = f$。

对于并行赋值（parallel assignments），本文定义如下：

$$\mathrm{A} = \{v_l := e_l \mid l = 1,2,\ldots,\mathrm{m}\} \tag{2-1}$$

其中v_l代表状态变量，它既可以是拥有明确数组下标的数组变量，也可以是非数组变量。“$v_l := e_l$”代表将e_l的结果赋值给状态变量v_l。

对于最弱前置条件（weakest precondition），本文定义如下：

$$\mathrm{WP}\ (\mathrm{A}, f) \equiv f\ [v_1 := e_1, v_2 := e_2, \ldots, v_m := e_m\ ] \tag{2-2}$$

其中符号“≡”代表两个公式在语法上是等价的。“$v_l := e_l$”代表用表达式e_l的结果替换 f 中出现的所有v_l。

2.1.3 协议

本文定义 P 为表示协议（带参协议实例）的符号，（I，R）是协议的形式化表达，其中 I 用于描述协议的初始状态，它是由状态变量组成的有限谓词公式集合。R 代表协议的规则集合。规则包含卫式与规则内容两部分，这里给出协议规则的定义：

$$\mathrm{r} \equiv \mathrm{g} \rhd \mathrm{A} \tag{2-3}$$

在公式(2-3)中，r 代表某条规则，即$\mathrm{r} \in \mathrm{R}$。g 代表规则 r 中的卫式，卫式是由状态变量组成的谓词表达式。A 代表规则内容，规则内容是对一组状态变量的并行赋值，即$\{v_l := e_l \mid l = 1,2,\ldots,\mathrm{m}\}$。规则中的卫式 g 决定规则的触发条件，即当协议状态$\mathrm{s} \mid =$ g时，规则内容 A 将会被执行，这会导致部分状态变量的取值发生变化，使协议进入新的协议状态。

在本文中，用guard(r)代表规则 r 的卫式 g，即：

$$\mathrm{guard}(\mathrm{r}) = g \tag{2-4}$$

用 action(r)代表规则 r 的规则内容 A，即：

$$\mathrm{action}(\mathrm{r}) = A \tag{2-5}$$

用 vars(A)代表被规则内容 A 所赋值的变量集合，即：

$$\mathrm{vars}(\mathrm{A}) = \{v_l \mid v_l := e_l \in A\} \tag{2-6}$$

定义由于触发协议规则 r 后，导致的状态迁移如下：

$$\begin{aligned} \mathrm{s} \xrightarrow{r} s' \equiv s \mid = g \wedge \forall\ v = e\ \in action(r) \rightarrow s'(v) \\ = s(e) \wedge \forall\, w \notin vars(action(r)) \rightarrow s'(w) \\ = s(w) \end{aligned} \tag{2-7}$$

定义协议的初始状态合集s_{init}如下：

$$s_{init} = \{s \mid s| = f, f \in I\} \tag{2-8}$$

2.1.4 可达状态集合

本文将协议 P=(I，R)运行过程中出现的协议状态称为协议的可达状态。将协议的所有可达状态称为可达状态集合，并用 RS(P)表示。RS(P)可归纳定义为：

(1)如果协议状态$\mathrm{s} \in s_{init}$，则 s 是协议的可达状态。

(2)如果协议状态s'是由协议状态 s 经过某条规则 r 转换得到，且 s 是协议的可达状态，即$\mathrm{s} \xrightarrow{r} s' \wedge s \in RS(P)$，则$s'$也是协议的可达状态。

2.1.5 带参协议

带参谓词公式是一种函数$f(i_1, i_2, \ldots, i_j)$，其中$\{i_x，\ 1 \le x \le j\}$代表参与谓词公式描述的各个节点。类似地，带参规则也是一种函数$\mathrm{r}(\mathrm{i}R_1, \mathrm{i}R_2, \ldots, \mathrm{i}R_k)$，其中$\{\mathrm{i}R_x，\ 1 \le x \le k\}$代表参与带参规则描述的各个节点。为了方便描述，本文使用 i 代表带参谓词公式的参数，用 iR 代表带参规则中的参数。带参协议是使用带参谓词公式与带参规则的协议，此处给出带参协议的定义：

$$\mathrm{P}\left(i_1, i_2, \ldots, i_j, \mathrm{i}R_1, \mathrm{i}R_2, \ldots, \mathrm{i}R_k\right) = (\mathrm{I}, \mathrm{R}) \tag{2-9}$$

其中 I 代表用于初始化带参协议的带参谓词公式集合，R 代表带参规则集合。由于协议的参数化，协议状态也被参数化。参数化的协议状态被称为带参协议状态。

2.1.6 带参协议验证问题

在本文中，假设带参协议用于描述有 N 个节点的通讯网络，带参协议状态由状态变量合集 V 所描述，带参协议的定义是$\mathrm{P}\left(i_1, i_2, \ldots, i_j, \mathrm{i}R_1, \mathrm{i}R_2, \ldots, \mathrm{i}R_k\right) = (\mathrm{I}, \mathrm{R})$，带参协议验证的要求是带参协议中的不变式，并且这种不变式是由带参谓词公式所描述，则带参协议验证所要回答的问题是：

“在带参协议P运行时，是否所有可达状态均满足不变式？”

或等价地回答：

“在带参协议P运行时，不变式是否在所有可达状态中都得到满足？”

一般而言，带参协议中的不变式可以由合适的时态逻辑公式描述，如线性时间逻辑（LTL）或分支时间逻辑（CTL）。而在本文的研究中，主要任务是确定 ParaVerifier[34]中的候选辅助不变式是否为带参协议实例的不变式，因此本文将不变式描述为：

$$\neg(\bigwedge_{n=1}^{k} f_n) \tag{2-10}$$

其中f_n代表判断公式，其中包括两种形式，它们分别是$e_1 = e_2$和$e_1 \neq e_2$。为了更加清晰地说明上述定义，下文将以包含 N 个节点的简单互斥协议为例，介绍带参协议中的不变式。

在包含 N 个节点的简单互斥协议中，每个节点拥有四种节点状态，它们分别是 I(dle)、T（rying）、C（ritical）和 E（xiting），它们分别代表空闲状态、尝试状态、占用状态和退出状态。x 代表一个全局布尔变量（非数组状态变量），它用于标记重要资源是否被占用。a 代表带参协议中的数组状态变量，a[i]用于描述节点 i 的节点状态。在简单互斥协议中，包含的带参规则如下：

$$\mathrm{idle}(iR) \equiv \mathrm{a}[iR] = \mathrm{E} \rhd \{\mathrm{a}[iR] \coloneqq I, x \coloneqq true\} \tag{2-11}$$

$$\mathrm{try}(iR) \equiv \mathrm{a}[iR] = \mathrm{I} \rhd \{\mathrm{a}[iR] \coloneqq \mathrm{T}\} \tag{2-12}$$

$$\mathrm{crit}(iR) \equiv \mathrm{x} = \mathrm{true} \wedge \mathrm{a}[iR] = \mathrm{T} \rhd \{\mathrm{a}[iR] \coloneqq C, x \coloneqq false\} \tag{2-13}$$

$$\mathrm{exit}(iR) \equiv \mathrm{a}[iR] = \mathrm{C} \rhd \{\mathrm{a}[iR] \coloneqq \mathrm{E}\} \tag{2-14}$$

在本文中，用$mutex_{initN}$表示简单互斥协议的初始化规则 I，用$mutex_{ruleN}$表示带参规则集合 R，用$MutualEx_N$表示包含 N 个节点的简单互斥协议。它们的定义分别如下：

$$mutex_{initN} \equiv \mathrm{x} = \mathrm{true} \wedge \bigwedge_{i=1}^{N} \mathrm{a}[i] = I \tag{2-15}$$

$$mutex_{ruleN} \equiv \{\mathrm{idle}(iR), \mathrm{try}(iR), \mathrm{crit}(iR), \mathrm{exit}(iR) \mid iR = 1,2,\dots,\mathrm{N}\} \tag{2-16}$$

$$MutualEx_N = (mutex_{initN}, mutex_{ruleN}) \tag{2-17}$$

用$\mathrm{mutualInv}(i_1, i_2)$表示简单互斥协议的不变式，其定义为：

$$\mathrm{mutualInv}(i_1, i_2) \equiv \neg(\mathrm{a}[i_1] = \mathrm{C} \wedge \mathrm{a}[i_2] = \mathrm{C}) \tag{2-18}$$

该不变式的代表在简单互斥协议中，描述节点状态的两个变量$\mathrm{a}[i_1]$和$\mathrm{a}[i_2]$不能同时取值为 C（i_1与i_2不相等）。

2.2 ParaVerifier 方法简述

ParaVerifier 方法是李勇坚等人提出的带参协议自动验证方法，基于该方法实现的带参协议自动验证程序被称为 ParaVerifier[34]。ParaVerifier 方法属于归纳证明方法。该方法具有以下优点：

(1)自动化验证。ParaVerifier 方法可以实现自动验证带参协议，其中包括自动寻找并判断候选辅助不变式，自动合成归纳不变式，自动生成带参协议证明脚本等等。

(2)找到的辅助不变式的结构简单易于理解。ParaVerifier 方法找到的辅助不变式的形式如公式(2-10)所示，是由“¬”、“∧”、括号和判断公式组成。这种辅助不变式代表一类带参协议的不可达状态。

(3)高兼容性与高效率。ParaVerifier 方法能够适用于一系列的典型带参协议。此外，ParaVerifier 方法能够验证中型带参协议 German 和大型带参协议 Flash。

在这些优势中，能够验证 Flash 协议足以证明 ParaVerifier 方法属于一流的带参协议自动验证方法。在带参协议验证领域中，Flash 协议属于验证难度最高的可验证带参协议之一。这是由于 Flash 协议结构复杂且可达状态空间巨大，即使是参数均为 2 的 Flash 协议实例，也包含上亿个可达状态，这对试验设备的内存带来了极大的挑战。此外 Flash 协议是最接近现实缓存一致性协议的带参协议。Chou 等人的文章中所讲：“如果某种带参协议验证方法能应用于 Flash 协议，那么该方法也可能应用于许多现实中的缓存一致性协议”[15]194。在目前的带参协议研究中，都没能验证实际应用的缓存一致性协议，因为这些带参协议十分复杂，这导致其拥有巨量的可达状态，发生状态爆炸问题。通过验证 Flash 协议，有望探索验证现实中的缓存一致性协议。因此 ParaVerifier 方法是一个优秀的带参协议自动化验证方法。

2.2.1 依赖证明定理

ParaVerifier 方法验证带参协议的基础理论是依赖证明定理[33,34]。通过使用依赖证明定理，ParaVerifier 方法可以找到带参协议的归纳不变式，从而证明带参协议满足设计需求。其中，设计需求必须是带参协议的不变式。

归纳不变式是指既满足带参协议初始规则 I，又可以经过协议规则 R 变换后得到保持为真的不变式。如果带参协议的归纳不变式可以推出设计需求，则可判断带参协议满足设计需求。

依赖证明定理包含三条依赖证明规则，它们分别代表在验证归纳不变式时，带参协议规则遇到的三种情况。下面给出三条依赖证明规则的定义。

定义 1：假设存在带参协议$P = (I, R)$，s 代表带参协议状态，R 代表带参协议规则集合，r 代表某条带参协议规则（$r \in R$），V 代表带参协议中的状态变量集合，*f* 代表由 V 中变量组成的不变式，F 代表带参协议中的不变式集合，*inv* 代表带参协议需要验证的设计需求。则有如下依赖证明：

⑴$CR_{estbl}(inv,r)$代表：$|=\text{guard(r)}$ 可推出$|=\text{WP(action(r)},inv)$。这条依赖规则的含义是 r 保证 *inv* 与状态转换中的前一状态相互独立。换句话说，如果存在$\text{s}\xrightarrow{r}s'$，虽然 r 改变了 s 中 *inv* 包含的状态变量的取值，但在经过转化后，如果s满足 *inv*，则s'依然满足 *inv*。

⑵$CR_{presv}(inv,r)$代表：$|=inv$可推出$|=\text{WP(action(r)},inv)$。这条依赖规则的含义是 r 保证 *inv* 成立。换句话说，如果存在$\text{s}\xrightarrow{r}s'$， r 没有改变 s 中 *inv* 包含的状态变量的取值，这导致如果 s 满足 *inv*，则s'也一定满足 *inv*。

⑶$CR_{aux}(inv,r,F)$代表：存在辅助不变式$aux\in\text{F}$，使$|=\text{aux}\wedge\text{guard(r)}$可推出$|=\text{WP(action(r)},\text{inv})$。这条依赖规则的含义是存在 r 和对应的辅助不变式$aux\in\text{F}$，能够保证在状态迁移过程中 *inv* 得到保持。换句话说，如果存在$\text{s}\xrightarrow{r}s'$并且$\text{s}|=aux$（其中$aux\in\text{F}$），如果 s 满足 *inv*，则$s'$也一定满足 *inv*。

本文使用缩写CRS(inv, r, F)代表$CR_{estbl}(inv,r)\vee CR_{presv}(inv,r)\vee CR_{aux}(inv,r,F)$。使用$\wedge$F代表将 F 中所有谓词公式用“∧”连接起来的结果。CRS(inv, r, F)意在表达 inv、r 和 F 之间的依赖关系，并且如果这种关系成立，则$\wedge$F是带参协议的归纳不变式。需要注意，F 中每个单独的谓词公式并非归纳不变式。基于上述观点，归纳证明定理的定义如下。

定理 1：假设存在带参协议$\text{P}=(\text{I},\text{R})$，如果$|=\text{I}\rightarrow\text{inv}\ \&\ \text{CRS}(inv,\text{r},\text{F})$，其中$inv\in\text{F}$并且$\text{r}\in\text{R}$，则$\wedge$F是带参协议 P 的归纳不变式。

如定理 1 所示，如果能够找到不变式集合 F，那$\wedge$F是带参协议的归纳不变式。由于$inv\in\text{F}$，因此可以推出带参协议 P 满足不变式 inv，即满足带参协议的设计需求。

在依赖证明定理中，设计需求 *inv* 是带参协议的不变式，但不一定是归纳不变式。如果没有辅助不变式的限制，在经过带参协议规则$\text{r}\in\text{R}$转变后，不变式 *inv* 可能难以保持为真。因此，仅靠初始规则 I 与规则集合 R 来证明 inv 非常困难。而找到包含 *inv* 的归纳不变式$\wedge$F，可以证明 I 与 R 能够保证 inv 一直为真。

下文给出归纳不变式的定义与用于验证带参协议的引理。

定义 2：如果谓词公式 Inv 同时拥有如下两种性质，则 Inv 是带参协议$\text{P}=(\text{I},\text{R})$的归纳不变式：

⑴$|=\text{I}\rightarrow\text{Inv}$，即对于带参协议的所有初始状态，Inv 均为真。

⑵对于每个$\text{r}\in\text{R}$，拥有性质$|=\text{Inv}\wedge\text{guard(r)}\rightarrow\text{WP(action(r)},\text{Inv})$，即状态迁移不会影响带参协议状态满足 Inv。

引理 1：假设存在带参协议$\text{P}=(\text{I},\text{R})$和需要验证的设计要求 inv。如果 P 中存在归纳不变式$\wedge$F，则$|=\wedge\text{F}\rightarrow\text{inv}$。

引理 1 意在说明设计要求 inv 一定可被归纳不变式推出。ParaVerifier 方法通

过使用引理 1 判断带参协议是否满足设计需求。在 ParaVerifier 方法中，inv 会被添加到 F 中，这导致∧ F一定可以推出 inv。因此，如果 ParaVerifier 方法能够找到归纳不变式∧ F，则可证明带参协议满足设计要求。在 ParaVerifier 方法中，寻找归纳不变式分为以下步骤：

(1)实例化需要证明的不变式。假设不变式包含 n 个参数$i_1, i_2, \ldots, i_n$，其中每个参数各不相同。实例化指将每个参数与自然数形成单射，例如$i_1 \to 1, i_2 \to 2, \ldots, i_n \to n$。

(2)实例化带参协议。根据第一步的实例化方式对带参谓词集合 I 与带参规则集合 R 进行实例化。

(3)寻找带参协议实例的归纳不变式。检查第一步实例化的不变式是否为带参协议实例的归纳不变式。如果是，则将需要证明的不变式实例作为带参协议实例的归纳不变式。如果不是，则寻找辅助不变式集合，并将所有得到的辅助不变式与需要证明的不变式实例合取，从而得到带参协议实例的归纳不变式。其中，在寻找辅助不变式的过程中会生成多个候选辅助不变式，这些候选辅助不变式是根据带参规则的实例产生的。如果候选辅助不变式是带参协议实例的不变式，则判断其为辅助不变式。

(4)得到带参协议的归纳不变式。根据前两步的实例化方法，倒推各个参数的泛化方法，将带参协议实例的归纳不变式泛化为带参协议的归纳不变式。

(5)对找到的带参协议归纳不变式进行归纳证明。首先，根据找到的归纳不变式和寻找辅助不变式时使用过的依赖关系生成证明脚本，其中依赖关系指生成辅助不变式时依赖证明定理的应用情况。然后，使用证明脚本与脚本对应的定理证明器验证归纳不变式的正确性。

2.2.2 ParaVerifier 方法的设计与实现

ParaVerifier 方法可分为三步，它们分别是寻找归纳不变式、生成归纳证明脚本和归纳证明。ParaVerifier 是基于 ParaVerifier 方法实现的带参协议自动验证程序，ParaVerifier 的工作过程如图 2-1 所示。

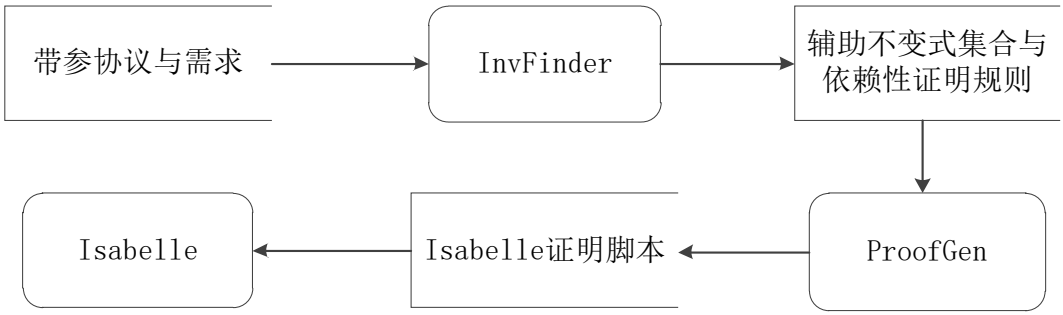

图 2-1 ParaVerifier 的工作过程

Figure 2-1 The Working Process of ParaVerifier

ParaVerifier 分为三部分，它们分别是 InvFinder、ProofGen 和 Isabelle[43]。它们分别对应着 ParaVerifier 方法中的寻找归纳不变式、生成归纳证明脚本和归纳证明。

InvFinder 的功能是找到归纳证明所需的归纳不变式，并提供找到归纳不变式所用到的依赖关系。InvFinder 的架构如图 2-2 所示，InvFinder 采用服务器—客户端架构。InvFinder 的服务器主要由定理证明器 Z3[52]和模型检测软件（NuSMV、Murphi）组成，其功能是判断客户端传来的谓词公式或候选辅助不变式。InvFinder 客户端负责生产候选辅助不变式、合成归纳不变式和提供依赖关系。

寻找归纳不变式需要 InvFinder 完成三条依赖规则的证明。其中依赖规则$CR_{estbl}(inv,r)$和$CR_{presv}(inv,r)$的证明由 InvFinder 客户端与定理证明器 Z3 完成证明，证明方法是由定理证明器 Z3 判断 InvFinder 客户端询问的依赖关系公式。依赖规则$CR_{aux}(inv,r,F)$由 InvFinder 客户端与模型检测软件完成证明，证明方法是由模型检测软件判断 InvFinder 客户端询问的候选辅助不变式。InvFinder 的工作步骤如下：

(1)实例化带参协议和不变式 inv（带参协议的设计要求）。

(2)根据实例化的带参规则与不变式 inv 生成依赖规则$CR_{estbl}(inv,r)$和$CR_{presv}(inv,r)$的满足条件并询问 Z3，若满足则跳至第五步。若不满足则进入下一步。

(3)根据实例化的带参规则、inv 和依赖规则$CR_{aux}(inv,r,F)$，合成候选辅助不变式，并将候选辅助不变式交由模型检测软件证明。如果候选辅助不变式是带参协议实例的不变式，则判断候选辅助不变式是辅助不变式，进入第四步。若候选辅助不变式不是带参协议实例的不变式，则继续合成其他候选辅助不变式并判断，如果所有候选辅助不变式均不是带参协议实例的不变式，则表示带参协议不满足设计要求。

(4)将第 3 步得到的辅助不变式放入待证明不变式队列。

(5)完成当前不变式与所有实例化带参规则的三种依赖关系证明（第二、三步）。如果当前不变式与所有实例化的带参协议规则均完成了三种依赖证明，则弹出待证明不变式队列中的不变式，并将该不变式作为不变式 inv 进行第二步。如果带参协议设计要求的实例与所有找到辅助不变式均完成了三种依赖证明，则将 inv 与所有辅助不变式连接，获得带参协议实例的归纳不变式。

(6)泛化带参协议实例的归纳不变式，得到带参协议的归纳不变式。

(7)返回带参协议的归纳不变式与用到的依赖关系。

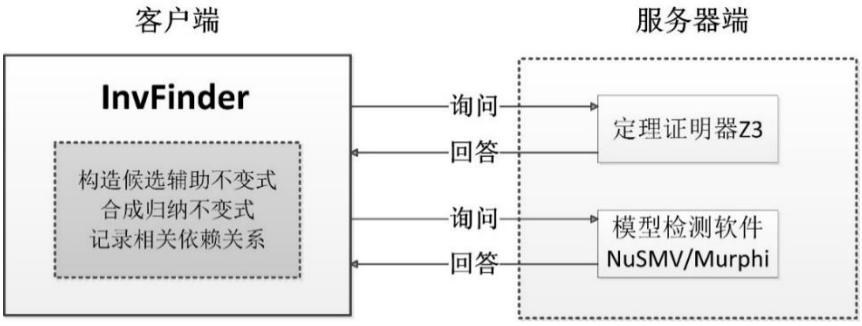

图 2-2 InvFinder 程序架构

Figure 2-2 The Program Architecture of InvFinder

ProofGen 和 Isabelle 分别用于生成 Isabelle 归纳证明脚本和完成归纳证明。其中，ProofGen 会根据 InvFinder 找到的归纳不变式和依赖关系生成 Isabelle 证明脚本，Isabelle 将根据证明脚本完成对归纳不变式的证明。

2.3 Murphi 与对称削减方法

Murphi[4]是一种基于显式状态搜索的模型检测工具。Murphi 最早是由 David L. Dill 提出，并由斯坦福大学进行维护。Murphi 拥有对称削减方法，可以大幅度减少可达状态的搜寻空间，减少模型检测所需要的时间消耗和内存占用。如今 Murphi 由罗马大学负责维护并且进行了多次版本更新，最新的 Murphi 改称为 CMurphi。CMurphi 除了改正了老版本存在的问题外，充分利用了 C++语言的特性提高了模型检测效率，成为了在学术界广泛应用模型检测程序。

CMurphi 主要用于协议验证领域，因此在面对带参协议时，只能验证带参协议的实例。在验证带参协议实例时 CMurphi 的使用步骤包括：

(1)使用 Murphi 语言编写带参协议的实例和需要验证的不变式实例，得到相关的 Murphi 文件。

⑵根据第一步得到的 Murphi 文件，使用 Mu 程序生成对应的 Cpp 文件。

⑶使用 g++编译第二步得到的 Cpp 文件以及 CMurphi 的 include 库，得到带参协议实例的验证程序。

⑷运行第三步得到的验证程序，获得带参协议实例的验证报告。

状态爆炸问题是协议验证领域中的主要问题之一。对称削减方法[54]能够有效减少所需搜寻状态空间，从而减轻状态爆炸问题对协议验证的影响。CMurphi 拥有对称削减功能，并以此适应某些大型带参协议实例的验证。通过使用对称削减功能，CMurphi 甚至能够利用显式状态枚举法验证小参数 Flash 协议实例，并且其验证时间与内存占用均是可以接受的。对称削减方法的效果是十分显著，例如对于参数均为 2 的 Flash 协议，对称削减前可达状态数量一共有 107866864 个，而对称削减后的可达状态数量仅有 857453 个。

在 CMurphi 中。对称削减指在搜索状态时忽略已找到的可达状态和已找到的可达状态的对称形式，应用对称削减后得到的可达状态集合被称为对称削减可达状态集合。

在带参协议实例中，带参协议实例状态间的对称关系是由带参协议实例状态中各个等价节点的排列恒等式来定义。这种定义是在保持各个下标的排列顺序不变的情况下，如果某个带参协议实例状态仅依靠变换数组状态变量的下标得到新的带参协议实例状态，那这两个带参协议实例状态所表达的带参协议实例的性质是相同的，因此称这两个带参协议实例状态具有对称性。

使用对称削减技术可以将带参协议实例的状态空间划分为多个等价的状态子空间，其中每个子空间足以体现带参协议实例的所有性质。因此只需遍历一个子空间内的带参协议实例状态即可验证候选不变式是否为带参协议实例的不变式。

为了更好解释带参协议实例状态间的对称关系，表 2-1 与表 2-2 分别给出节点数（参数）为 2 的简单互斥协议的可达状态集合和对称可达状态集合。

表 2-1 中一共拥有 12 种带参协议实例状态。表 2-2 中一共拥有 7 种带参协议实例状态。经过对称削减后，简单互斥协议实例的可达状态空间由 12 个带参协议实例状态减少到 7 个带参协议实例状态，其中被忽略的带参协议实例状态是 $s_3, s_6, s_9, s_{10}, s_{12}$。

如图 2-3 所示，简单互斥协议实例的状态空间被分为状态空间 1 和状态空间 2。其中，带箭头的黑色实线代表带参协议实例状态间的转换关系，例如s_1到s_2存在一条指向s_2的带箭头黑色实线，这代表带参协议实例中存在规则 r，使s_1到s_2存在转换关系，即$s_1 \xrightarrow{r} s_2$。红色虚线框中的带参协议实例状态集合代表对称削减后的可达状态集合，即$\{s_1, s_2, s_4, s_5, s_7, s_8, s_{11}\}$；蓝色虚线框中的带参协议实例状态集合代表对称削减后被忽略的可达状态集合，即$\{s_1, s_3, s_5, s_6, s_9, s_{10}, s_{12}\}$，其中，由于带

参协议实例状态s_1和带参协议实例状态s_5没有对称状态，因此两者均存在于两个状态空间中。

表 2-1 简单互斥协议的可达状态合集

Table 2-1 Collection of Reachable States for Simple Mutual Exclusion Protocols

状态	n[1]	n[2]	x
s_1	I	I	true
s_2	T	I	true
s_3	I	T	true
s_4	C	I	false
s_5	T	T	true
s_6	I	C	false
s_7	E	I	false
s_8	C	T	false
s_9	T	C	false
s_{10}	I	E	false
s_{11}	E	T	false
s_{12}	T	E	false

表 2-2 对称削减后的简单互斥协议的可达状态合集

Table 2-2 Collection of Reachable States for Simple Mutual Exclusion Protocols after Symmetric Reduction

状态	n[1]	n[2]	x
s_1	I	I	true
s_2	T	I	true
s_4	C	I	false
s_5	T	T	true
s_7	E	I	false
s_8	C	T	false
s_{11}	E	T	false

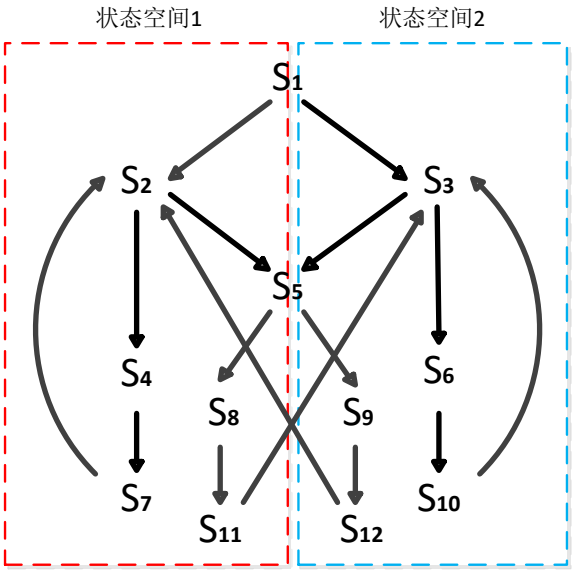

图 2-3 对称状态空间

Figure 2-3 Symmetric State Space

本文以‘~’代表两个带参协议实例状态对称，则在对称削减前，12 个可达状态中包含的对称状态组分别为s_2~s_3，s_4~s_6，s_7~s_{10}，s_8~s_9，s_{11}~s_{12}。以对称状态组s_2~s_3为例，$s_2=\{T,I,true\}$与$s_3=\{I,T,true\}$的不同之处为特征 n[1]和特征 n[2]的取值，其中 1 与 2 代表的节点是等价节点。在保持节点排列顺序不变式的情况下，只需将下标‘1’与下标‘2’对换即可实现两个带参协议实例状态的相互转化，因此判断s_2~s_3。

2.4 本章小结

本章用于介绍基于决策树的带参协议自动化验证方法中用到的他人基础理论，这些理论分别为带参协议验证的相关定义、ParaVerifier 方法和 Murphi 与对称削减技术。其中，带参协议验证是本文研究的领域。ParaVerifier 方法是基于决策树的带参协议自动化验证方法的重要借鉴。Murphi 与对称削减技术是基于决策树的带参协议自动化验证方法中获取训练数据的基础。

3 基于决策树的带参协议自动化验证方法

为了探索决策树方法在自动寻找带参协议不变式中的应用方法，提升带参协议验证的效率，本文提出了基于决策树的不变式判断方法。基于决策树的不变式判断方法的目的是判断候选不变式是否为带参协议实例的不变式。相比主流的模型检测软件（NuSMV、Murphi），在验证大型带参协议实例时，使用基于决策树的不变式判断方法做出的不变式判断器，能够使用更少的时间与内存完成候选不变式的判断。

通过将基于决策树的不变式判断方法与 ParaVerifier 方法[34]的结合，本文提出了基于决策树的带参协议自动化验证方法。通过试验证明，在验证大型带参协议时，相比 ParaVerifier 方法，基于决策树的带参协议自动化验证方法能够减少带参协议验证所需要的时间与内存。

3.1 基于决策树的不变式判断方法

基于决策树的不变式判断方法的设计目的是，通过判断 ParaVerifier 方法中的候选辅助不变式来辅助寻找带参协议的归纳不变式。在 ParaVerifier 方法中，候选辅助不变式是为了寻找辅助不变式而生成的候选不变式，候选辅助不变式的判断工作由模型检测程序 NuSMV 或 Murphi 完成。虽然两者均是优秀的模型验证软件，但在验证大型带参协议实例时，均会耗费大量的时间和内存。为了提高不变式的判断效率，本文利用决策树的分类特性，提出了基于决策树的不变式判断方法。通过试验证明，相比 NuSMV 或 Murphi，基于决策树的不变式判断方法能够使用更少的时间与内存完成候选不变式的判断。基于决策树的不变式判断方法可分为以下步骤：

(1) 获取决策树训练数据。将带参协议实例的对称削减可达状态集合作为训练数据。

(2) 训练数据预处理。根据候选不变式包含的特征集合简化训练数据并挑选目标特征。

(3) 训练决策树。使用第二步得到的训练数据，根据针对带参协议实例的决策树算法训练决策树。

(4) 判断候选不变式。将决策树转化为决策树路径，利用决策树路径与候选不变式的关系，判断候选不变式是否为带参协议实例的不变式。

3.1.1 训练数据的选取

为了保证决策树能够准确、高效地判断候选不变式，本文提出了使用对称削减可达状态集合作为决策树训练数据的方法。

决策树方法属于一种机器学习方法，需要有训练数据才能获得决策树。一般而言，每条训练数据的形式如下：

$$(X, \mathrm{Y}) = (x_1, x_2, x_3, \ldots, x_k, \mathrm{y}) \tag{3-1}$$

其中，向量 X 代表基础特征集合，其中$x_1, x_2, x_3, \ldots, x_k$代表基础特征。Y 代表目标特征，该特征用于分类基础数据，其中 y 代表具体的目标特征。

3.1.1.1 使用可达状态与不可达状态作为训练数据的问题

训练数据决定着决策树的品质，因此选择合适的训练数据尤为重要。在利用决策树方法寻找循环程序的不变式的工作中，训练数据被分为好状态和坏状态。其中，好状态代表循环程序的可达状态，坏状态代表循环程序的不可达状态。

通过使用由好状态与坏状态组成的训练数据，决策树能够判断循环程序状态是好状态还是坏状态。因此，如果将决策树归纳为一种谓词表达式，那么这种谓词表达式可以被看作是循环程序的不变式。例如在 Garg P 等人提出的 ICE 方法中[46,47]，训练决策树需要训练数据和状态间蕴含关系。其中，状态间蕴含关系是为了将训练数据中的不可分类数据划分为好状态或坏状态，并使用科学划分后的训练数据训练决策树。在 Siddharth Krishna 等人提出的寻找不变式方法中[50]，决策树训练数据被分为好状态与坏状态。

好状态的获取方法分为随机获取方法与运行获取方法。随机获取方法指根据初始化条件随机生成循环程序初始状态。运行获取方法指通过运行循环程序得到循环程序的可达状态。好状态的具体获取流程为：第一，根据初始化条件随机生成初始化状态，并将这些状态标记为好状态；第二，使用初始化状态作为输入并运行循环程序，在运行过程中，将得到的新状态标记为好状态。

坏状态的获取方法有三种，它们分别是随机获取方法、按“距离”获取方法和逆向运行获取方法。随机获取方法指根据循环程序的性质随机生成不可达状态，并将这些状态标记为坏状态。按“距离”获取方法指，根据循环程序的性质与“距离”概念生成不可达状态。其中，“距离”代表循环程序状态之间的差异，如果循环程序状态之间存在取值不同的状态变量越多，则“距离”越大。在按“距离”获取方法中，寻找与好状态“距离”较近的坏状态可以提升决策树的精度。逆向运行获取方法指将随机获取方法或按“距离”获取方法得到的坏状态作为输入，通过循环程

序的逆向循环得到新状态，并将新状态标记为坏状态。坏状态的具体获取流程为：第一，通过随机获取方法或按“距离”获取方法找到一部分坏状态；第二，将第一步找到的坏状态作为输入状态，使用逆向循环方法继续寻找坏状态。

虽然在循环程序中，使用包含好状态与坏状态的训练数据起到了良好的效果，但是这种方法在应用于带参协议时却出现了问题。

对于 Garg P 等人提出的 ICE 方法，蕴含关系用于处理无法被分类的训练数据，然而在带参协议中，通过获取带参协议的状态空间，每个状态均可被清晰划分。其中，好状态对应带参协议的可达状态，坏状态对应带参协议的不可达状态，因此蕴含关系概念在带参协议中无法使用。

对于 Siddharth Krishna 等人提出的寻找不变式方法，单纯使用好状态与坏状态作为训练数据的方法是值得借鉴的。但通过试验证明，这种训练数据选取方法不适用于带参协议验证。

在实验中，对于通过好状态随机获取方法、坏状态随机获取方法、运行获取方法和逆向运行获取方法联用得到的好状态与坏状态。通过使用这些数据训练得到的决策树精度较差。在针对 German[2]协议实例的试验中，决策树对带参协议实例状态的判断错误率较大。虽然通过增加训练数据的数量使决策树的精度得到提升，但是依然不能符合带参协议验证的严谨性，即达到百分之百的判断正确率。

对于通过好状态随机获取方法、按“距离”获取方法、运行获取方法和逆向运行获取方法联用得到的训练数据。通过使用这些数据训练得到的决策树的精度得到了明显的提升。在实验中，以带参协议实例状态中状态变量取值不同的数量作为“距离”。例如，对于带参协议实例状态$s_1 = \{I, I, true\}$与带参协议实例状态$s_2 = \{I, T, true\}$，带参协议实例状态中一共包含 3 个特征，由于两个带参协议实例状态间只有第二个特征取值不同（s_1中的第二个特征取值为 I，而s_2中的第二特征取值为 T），因此判断两个带参协议实例状态的“距离”为 1。通过使用与好状态“距离”为 1 的坏状态，决策树的精度得到提升，除了能够识别已学习的好状态外，也能辨别大部分的可达状态。然而，决策树依然会错误判断某些带参协议实例状态。

造成由好状态和坏状态组成的训练数据难以应用于带参协议实例验证的原因是由于训练数据的不完整。对于一般的训练数据取法，训练数据只是全部数据的一部分，因此决策树总会存在误差。虽然可以通过调整训练数据数量、逐步精华、剪枝等手段减少决策树的误差，但难以根除误差。

带参协议实例验证要求验证过程具有高度的严谨性，这要求决策树够准确判断所有的带参协议实例状态。因此，带有误差的决策树难以完成这种工作。

决策树误差的产生原因是由于决策树无法学习到所有的带参协议实例状态。因此，假设将所有带参协议实例状态均用于训练决策树，决策树误差将不会存在。

然而，这种假设在现实中无法实现，因为带参协议实例的状态空间巨大，对实验设备造成巨大的压力。例如对于所有参数均为 2 的 Flash 协议实例，可达状态数量达到了 107866864 个，而不可达状态是可达状态数量的千万倍以上。由于当前试验设备的性能有限，即使能够勉强获得全部可达状态，不可达状态也难以全部得到。因此这种由好状态和坏状态组成的训练数据难以应用于带参协议实例验证。下文将举例说明因带参协议实例状态无法找全对决策树训练的影响。

由于决策树用于判断带参协议实例状态是好状态或坏状态，因此可以将决策树抽象为可达状态与不好状态的边界，对于不同的训练数据，决策树的表现如图 3-1、图 3-2 所示。

图 3-1、图 3-2 是对训练数据效果的抽象表达，整个点阵代表带参协议实例的状态空间，其中白色圆圈代表没有选入训练数据的带参协议实例状态，黑色圆圈代表选入训练数据的带参协议实例状态，蓝色虚线框内的黑/白色圆圈代表系统中的可达状态，蓝色虚线框外的黑/白色圆圈代表系统中的不可达状态。带箭头的红色实线代表训练出的决策树。

如图 3-1 所示，所有带参协议实例状态均用于训练决策树，得到的决策树可以区分所有的好状态与坏状态。在图 3-1 中，箭头的红色实线将所有数据分为两类，左边是带参协议实例的可达状态，右边是带参协议实例的不可达状态。

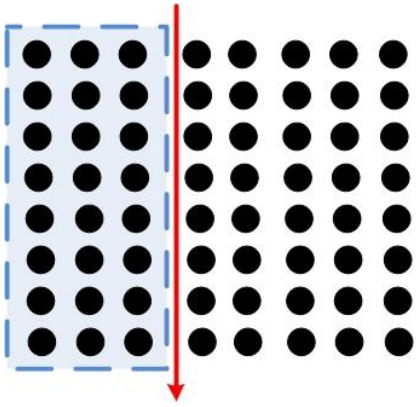

图 3-1 精准度达标的决策树

Figure 3-1 High Precision Decision Tree

如图 3-2 所示，带参协议实例的部分状态（包含好状态与坏状态）被用于训练决策树，由于训练数据数量不足，决策树会产生误差。在图 3-2 中，箭头的红色实线将所有数据分为两类，上方是带参协议实例的不可达状态，下方是带参协议实例的可达状态。从图 3-2 中可以看出，决策树具有较大的误差，会误判部分可达状态和不可达状态。

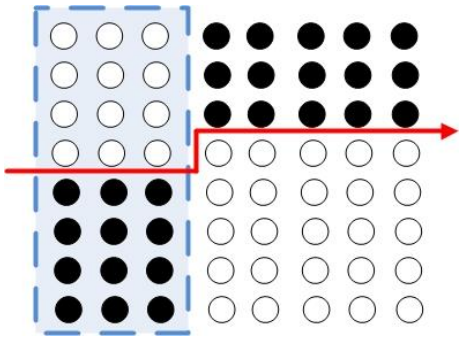

图 3-2 精准度不达标的决策树

Figure 3-2 Low Precision Decision Tree

3.1.1.2 使用对称削减可达状态集合作为训练数据

为了避免决策树出现误差，本文使用对称削减可达状态集合作为训练数据。使用这种训练数据后，决策树的功能将从“判断带参协议实例状态”变为“判断候选不变式”。

使用对称削减可达状态集合能够判断候选不变式。基于显式状态枚举法的模型检测程序能够通过遍历带参协议实例的对称削减可达状态集合，来判断候选不变式是否为带参协议实例的不变式。其原因是参协议实例的对称削减可达状态集合能够体现带参协议实例的全部性质。因此，使用对称削减可达状态集合作为训练数据后，决策树能够判断候选不变式是否为带参协议实例的不变式。

找到带参协议实例的对称削减可达状态集合是可行的。虽然获取全部带参协议实例状态十分困难，但是只获取参协议实例的对称削减可达状态集合却能够办到。当参数较小时，即使目标是大型带参协议实例，现有的技术也能够收集带参协议实例的对称削减可达状态集合。例如，Murphi 能够通过遍历参数均为 2 的 Flash 协议实例的对称削减可达状态集合来验证 Flash 协议实例的性质。

因此，基于决策树的不变式判断方法结合 Murphi 中的对称削减技术，使用对称削减可达状态集合作为训练数据。这样做不仅能够避免决策树出现误差，而且能够减少判断候选不变式需要的时间消耗和内存占用。

3.1.2 训练数据的预处理

训练数据预处理的主要内容包括简化训练数据与选择目标特征。简化训练数据是指去除可达状态中与候选不变式无关的特征，即保留候选不变式中包含的特征并去除其他特征。本文将选不变式中包含的特征称为相关特征，将不是相关特征

的特征称为无关特征。

去除无关特征的好处有两点：第一是减少训练数据的内容，使决策树的生成更快，让决策树更为简单，从而提升决策树判断候选不变式的效率；第二是避免无关特征的干扰，有利于决策树的每条路径包含所有的相关特征。简化训练数据的步骤如下：

(1)收集候选不变式中的特征。

(2)简化训练数据。保留训练数据中的相关特征，去除无关特征，从而简化训练数据中的每个可达状态。

使用简化后的训练数据后，决策树能够对候选不变式进行准确判断。这是因为基于决策树的不变式判断方法依靠可达状态类别判断候选不变式。而可达状态类别仅由相关状态划分。因此去除无关特征不仅对决策树判断候选不变式没有影响，而且能够防止冗余属性的出现。

由于使用简化训练数据，决策树将只能判断包含特定特征的候选不变式，因此在判断多个候选不变式时，基于决策树的不变式判断方法可能会生成多棵决策树。

在早期设计中，本文致力于训练一棵决策树来判断所有候选不变式，然而经过多次试验发现这种做法非常困难。决策树算法是一种贪心算法，通过在每个内部节点中选取“最佳分类特征”对训练数据进行分类，由此训练出精简有效的决策树。但是这种贪心算法必然导致决策树的每条路径中，无法包含验证所有候选不变式所需要的特征。

因此，在基于决策树的不变式判断方法中，会根据包含不同特征的候选不变式生成不同的决策树。由于决策树的特殊性，简化训练数据是可行的。这样做不仅使决策树能够准确地判断候选不变式，而且提升了决策树生成与判断的效率。

以参数均为 2 的简单互斥协议实例为例，假设候选不变式为$\neg(n[2]=C)$，由于该候选不变式只包含特征 n[2]，这使特征 n[1]与特征 x 成为无关属性，因此可将对称削减可达状态集合（表 2-2）简化至表 2-3 的状态。其中，带参协议状态变量集合将变为$V'=\{n[2]\}$，大幅度地减少了训练数据的数据量。

关于选择目标特征。在将决策树归纳为不变式的方法中，决策树的训练数据包含好状态和坏状态，训练决策树的目的是判断状态是好状态还是坏状态，因此决策树的目标特征为状态是否为好状态。然而在基于决策树的不变式判断方法中，由于所有训练数据均属于可达状态，因此决策树的目标特征不能为带参协议实例状态是否为可达状态。

为了寻找合适的目标特征，本文提出了以“可能取值种类最多的特征”作为目标特征的方法。这种方法有利于决策树路径包含所有的相关特征。假如在所有特征中随机确定目标特征，则可能出现由于目标特征中的可能取值过少，导致决策树无

法包含全部相关特征的情况。例如在极端的情况下，特征 m 被选为目标特征，然而经过对称削减后，特征 m 只包含一种可能取值，这导致决策树无法正常建立，因为整个训练数据一开始就属于一类。

表 3-1 简化后的可达状态合集（对称削减）

Table 3-1 Selected Reachable States (Symmetric Reduction)

状态	n[2]
s_1	I
s_2	I
s_4	I
s_5	T
s_7	I
s_8	T
s_{11}	T

3.1.3 针对带参协议实例的决策树算法

为了能够准确判断候选不变式是否为带参协议实例的不变式，基于决策树的不变式判断方法要求决策树的每条路径包含全部相关特征。然而，经典决策树算会将某些相关特征判断为冗余特征并忽略。因此，本文基于经典决策树算法提出了针对带参协议实例的决策树算法，使决策树的每条路径包含全部相关特征。

决策树是一种树状图结构。其中，非叶子节点（包括根节点和内部节点）代表某种特征的测试（例如白菜的颜色是绿的还是黄的），分支代表特征测试的输出结果，叶子节点代表目标特征的分类结果。由根节点、内部节点、分支和叶子节点组成的连线代表从根节点通往叶子节点的路径，即决策树路径。每条决策树路径代表决策树中的一种分类规则。

决策树算法自顶向下构建决策树。在每次划分样本时（即构建非叶子节点），决策树算法会从基础特征集合中选择最佳分类特征。最佳分类特征是指使用该特征划分训练数据后，训练数据的纯度提升最高。训练数据纯度是指目标特征取值的一致性。如果训练数据被划分为几组数据，那么划分后的训练数据纯度是指各组数据纯度的平均值。

对于不同的决策树算法，衡量最佳分类特征的标准也各不相同。经典的决策树算法包括：ID3（Iterative Dichotomiser 3）算法[49]，该算法使用基于香浓熵的信息

增益作为选取最佳分类特征的标准；C4.5（successor of ID3）算法[50]，该算法是 ID3 算法的改进，解决了 ID3 算法偏向小量值的问题，使用基于香浓熵且待惩罚系数的信息增益作为选取最佳分类特征的标准；CART（Classification And Regression Tree）算法[53]，该算法生成的决策树是二叉树，使用基尼指数作为选取最佳分类特征的标准。

在基于决策树的不变式判断方法中，需要决策树能够准确无误地判断候选不变式，因此需要修改决策树算法，防止决策树路径丢失相关属性。

针对带参协议实例的决策树算法是在 ID3 算法的基础上修改而来。选择以 ID3 算法为基础的原因是在本文中，训练数据不存在小量值数据问题，并且每个特征通常拥有两个以上的可取值。因此不考虑 C4.5 算法与 CART 算法。

在基于决策树的不变式判断方法中，如果直接使用 ID3 算法，决策树可能错误判断候选不变式。这是因为 ID3 算法会忽略候选不变式的某些相关特征或可达状态，而这些信息都是判断候选不变式所需的关键信息。

算法 1 中展示的是经典的 ID3 算法。其中会导致决策树精度出现问题的部分是语句 1 和语句 2，它们分别对应着决策树叶子节点的两种生成条件与方法。

语句 1 会造成决策树路径中缺少判断候选不变式所需的特征。一般情况下，当训练数据已被完全分类后不需要再对训练数据进行划分，否则决策树的大小将进一步增加，产生冗余信息。然而在判断候选不变式时，这些冗余信息是必要的。

由于实际数据过于复杂且不便于理解，因此本文以假设的带参协议实例的可达状态集合来简要说明语句 1 带来的问题。表 3-1 中展示的是假设的可达状态集合。图 3-3 展示的是由表 3-2 数据所训练出的决策树。其中，方形代表非叶子结点，圆形代表叶子节点。对于叶子节点，目标特征是 n[2]，即叶子节点 I 和叶子节点 T 分别表示$\mathrm{n}[2]=\mathrm{I}$和$\mathrm{n}[2]=\mathrm{T}$。

从图 3-3 中可以看出，特征 x 并没有包含在决策树的每条路径中，这使决策树可能错误判断候选不变式。例如，假设需要判断的候选不变式为$\neg(n[1]=C \wedge n[2]=I \wedge x=true)$，由于没有关于 x 的任何信息，只能由 n[1]与 n[2]两个特征判断候选不变式。而如果单看 n[1]与 n[2]两个特征，候选不变式将等价于$\neg(n[1]=C \wedge n[2]=I)$，导致决策树会误认为该候选不变式不是带参协议实例的不变式。但是如果考虑特征 x 的话，$\neg(n[1]=C \wedge n[2]=I \wedge x=true)$是不变式，因为状态{C，I，true}不会出现于可达状态中。

算法 1：ID3 算法

算法 1 为 ID3 算法。

输入：训练数据 dataset，训练数据表头 Labels

输出：字典类型的决策树 myTree

```
1： if 所有数据均属于一类 then return 数据类别 endif
2： if labels 中的特征用完 then return 数据集中占多数的数据类别
3： endif
3： var bestFeat ← 根据熵寻找得到的最佳分类特征的编号
4： var bestFeatLabel ← 最佳分类特征的名字
5： 根据最佳分类特征建立根节点。
6： 删除 labels 中的最佳分类特征。
7： var featValues ← 最佳分类特征的所有可能取值
8： for value in featValues：
9：     var subDataSet ← featValues 取值为 value 的数据集
10：    var myTree[bestFeatLabel][value] ← Id3(subDataSet,Labels)
11：end
12：return myTree
```

表 3-2 假设可达状态集合

Table 3-2 Assumption Reachable States

状态	n[1]	n[2]	x
s_4	C	I	false
s_{11}	E	T	false

如图 3-4 所示，冗余属性 x 被添加到每条决策树路径种，这使决策树得到了特征 x 的相关信息，从而可以确定候选不变式$\neg(n[1] = C \wedge n[2] = I \wedge x = true)$是带参协议实例的不变式。

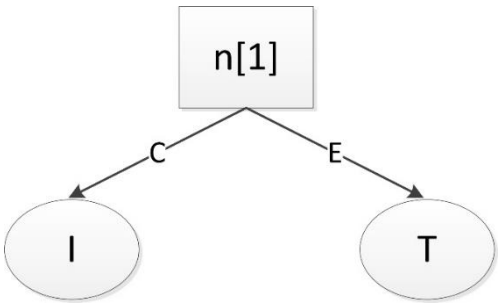

图 3-3 假设可达状态集合的决策树

Figure 3-3 Decision Trees for Assuming Reachable States

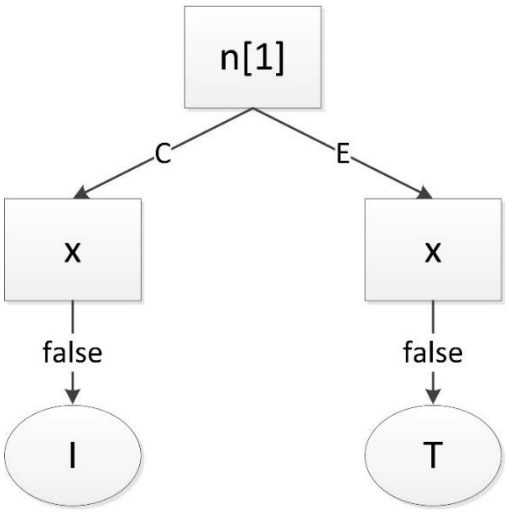

图 3-4 假设可达状态集合的决策树（添加特征 x）

Figure 3-4: Decision Tree for Assuming Reachable States (Adding Feature x)

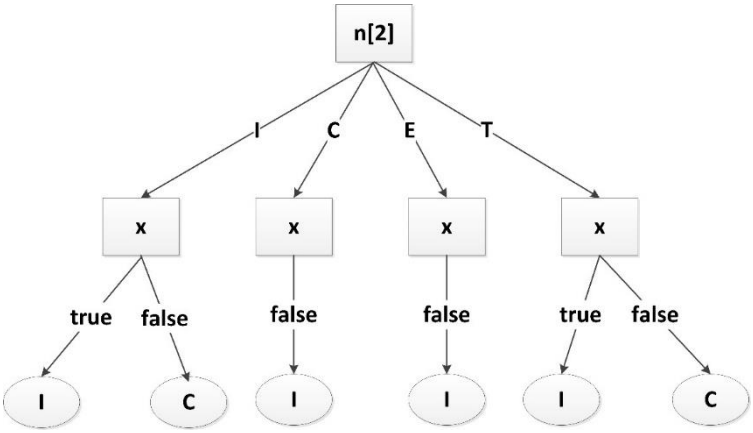

图 3-5 简单互斥协议的决策树（节点数为 2）

Figure 3-5 Decision Tree for Simple Mutual Exclusion Protocol (Number of Nodes is 2)

语句 2 会使叶子节点中缺少目标特征的可能取值，从而忽略部分可达状态。语句 2 表示如果所有基本特征使用完毕后，训练数据依然无法完全分类，则会选取训练数据中数量占比最多的目标特征取值作为叶子节点。然而，这种做法会忽略带参协议实例中的某些性质，使决策树误判对候选不变式。下面以简单的互斥协议为例，说明语句 2 可能造成的问题。

如图 3-5 所示，图中展示的是由表 2-1 中数据作为训练数据，并使用 ID3 算法生成的决策树，其中目标特征是 n[1]。由于使用语句 2 生成叶子节点，状态{T, I, true}、{E, I, false}、{T, C, false}、{T, E, false}、{T, T, true}、{E, T, false}均无法被决策树体现。如果需要判断关于这些状态的候选不变式，如$\neg(n[1] = T \wedge n[2] = I \wedge x = true)$，那么决策树将会把这些候选不变式误判为带参协议实例的不变式。

为了解决 ID3 算法遗失带参协议实例关键信息的问题，本文基于 ID3 算法提出了针对带参协议实例的决策树算法，保证了决策树的每条路径包含所有相关特征，使决策树能够准确判断候选不变式。

算法 2 中展示的是针对带参协议实例的决策树算法。该算法是由 ID3 算法修改而来。其中，修改部分有两处，第一处是删除了算法 1 中的语句 1，这导致在训练数据划分完毕后，如果仍有特征尚未使用，则继续分类训练数据。第二处是修改算法 1 中的语句 2，修改为返回剩余数据中目标特征的全部可取值。

通过删除算法 1 中的语句 1，针对带参协议实例的决策树算法可以生成如图 3-4 中的决策树。其中，在对训练数据正常分类后，“冗余特征”也会被加入到决策树中，解决了决策树路径中缺少相关特征的问题。

图 3-6 展示的是使用表 2-1 中的数据作为训练数据，由针对带参协议实例的决策树算法生成的决策树。在图 3-6 中，当基础特征用完时，叶子节点不再仅由数量占比最大的目标特征的可能取值构成，而是包含了目标特征的所有可能取值，其中每个可能取值用“|”连接。由此，决策树体现了带参协议实例的所有可达状态，解决了决策树叶子节点中缺少目标特征可能取值的问题。

算法 2：针对带参协议实例的决策树算法

算法 2 中展示的是修改后的 ID3 算法

输入：训练数据 dataset，训练数据表头 Labels

输出：字段类型的决策树 myTree

```
if labels 中的特征用完 then return 数据集的所有数据类别
endif
var bestFeat ← 根据熵寻找得到的最佳分类特征的编号
var bestFeatLabel ← 最佳分类特征的名字
根据最佳分类特征建立根节点。
删除 labels 中的最佳分类特征。
var featValues ← 最佳分类特征的所有可能取值
for value in featValues:
    var subDataSet←featValues 取值为 value 的数据集
    var myTree[bestFeatLabel][value] ← Id3forInv（subDataSet, Labels）
end
return myTree
```

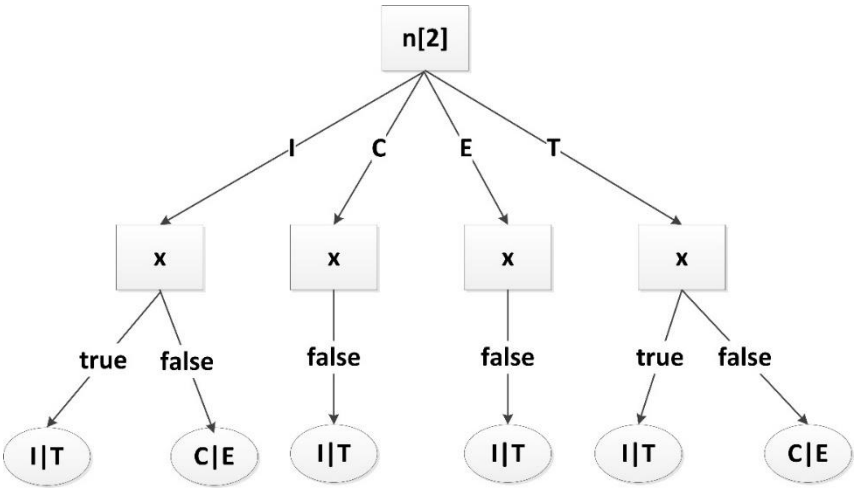

图 3-6 针对带参协议实例的决策树算法生成的决策树

Figure 3-6 Decision Tree Generated by Decision Tree Algorithms for Parametric Protocols

3.1.4 使用决策树判断候选不变式

使用决策树判断候选不变式的原理是利用决策树路径与候选不变式之间的关系。决策树路径是从根节点到叶子节点组成线路，这条路由各种节点和分支代表的谓词公式与“∧”组成。由于这些谓词公式均是判断公式（$e_1 = e_2$），因此每条决策树路径都可以代表一类带参协议实例的可达状态。

在基于决策树的不变式判断算法中，候选不变式是 ParaVerifier 方法中的候选辅助不变式，因此候选不变式的形式为$\neg(\bigwedge_{n=1}^{k} f_n)$。

这种候选不变式的形式可分为两部分：第一部分是最外层的非括号；第二部分是最外层的非括号中的内容。其中，第一部分代表否定。而第二部分由判断公式（$e_1 = e_2$和$e_1 \neq e_2$）与“∧”组成，因此第二部分可以代表一类带参协议实例的状态。如果将候选不变式的第一部分与第二部分联合起来，它们所表达的意思是第二部分代表的一类带参协议实例状态不会出现在带参协议实例的可达状态集合中。例如$\neg(n[1] = T \wedge n[2] = I \wedge x = true)$，代表状态{T, I, true}是不可达状态。

因此，如果候选不变式是带参协议实例的不变式，那么候选不变式的第二部分将与所有的决策树路径均矛盾。反而言之，如果出现某条决策树路径与候选不变式的第二部分不矛盾，则判断该候选不变式不是带参协议实例的不变式。

例如，公式(3-2)中展示的是由图 3-6 中决策树转化而来的决策树路径，这些决策树路径代表可达状态集合的全部类别（由于简单互斥协议较为简单，因此在本例中，每个可达状态单分一类）。在图 3-6 中，决策树一共有 12 条决策树路径，其中每条决策树路径都对应着一类可达状态。

为了利用决策树路径判断候选不变式，本文提出的具体方法是将每条决策树路径与候选不变式中的第二部分用“∧”连接起来，如果连接后的每条谓词公式均产生矛盾，则判定该候选不变式是带参协议实例的不变式。例如，对于候选不变式$\neg(n[1] = T \wedge n[2] = I \wedge x = true)$，如果使用公式(3-2)对其进行判断，则会产生如公式(3-3)所展示的谓词公式集合。

如公式(3-3)所示，由于第二条谓词公式“$(n[2] = I \wedge x = true \wedge n[1] = T) \wedge (n[1] = T \wedge n[2] = I \wedge x = true)$”没有产生矛盾，因此候选不变式$\neg(n[1] = T \wedge n[2] = I \wedge x = true)$不是带参协议实例的不变式。

对于候选不变式$\neg(n[1] = T \wedge n[2] = I \wedge x = false)$，如果使用公式(3-2)对其进行判断，决策树路径与候选不变式的连接结果如公式(3-4)所示，其中的候选不变式的第二部分与公式(3-2)中的所有谓词表达式均矛盾，因此判断$\neg(n[1] = T \wedge n[2] = I \wedge x = false)$是带参协议实例的不变式。

$$
\begin{gathered}
(n[2] = I \ \wedge \ x = true \ \wedge n[1] = I \ ) \\
(n[2] = I \ \wedge \ x = true \ \wedge n[1] = T \ ) \\
(n[2] = I \ \wedge \ x = false \ \wedge n[1] = C \ ) \\
(n[2] = I \ \wedge \ x = false \ \wedge n[1] = E \ ) \\
(n[2] = C \ \wedge \ x = false \ \wedge n[1] = I \ ) \\
(n[2] = C \ \wedge \ x = false \ \wedge n[1] = T \ ) \\
(n[2] = E \ \wedge \ x = false \ \wedge n[1] = I \ ) \\
(n[2] = E \ \wedge \ x = false \ \wedge n[1] = T \ ) \\
(n[2] = T \ \wedge \ x = true \ \wedge n[1] = I \ ) \\
(n[2] = T \ \wedge \ x = true \ \wedge n[1] = T \ ) \\
(n[2] = T \ \wedge \ x = false \ \wedge n[1] = C \ ) \\
(n[2] = T \ \wedge \ x = false \ \wedge n[1] = E \ )
\end{gathered} \tag{3-2}
$$

$$
\begin{gathered}
(n[2] = I \ \wedge \ x = true \ \wedge n[1] = I \ ) \wedge (n[1] = T \wedge n[2] = I \wedge x = true) \\
(n[2] = I \ \wedge \ x = true \ \wedge n[1] = T \ ) \wedge (n[1] = T \wedge n[2] = I \wedge x = true) \\
(n[2] = I \ \wedge \ x = false \ \wedge n[1] = C \ ) \wedge (n[1] = T \wedge n[2] = I \wedge x = true) \\
(n[2] = I \ \wedge \ x = false \ \wedge n[1] = E \ ) \wedge (n[1] = T \wedge n[2] = I \wedge x = true) \\
(n[2] = C \ \wedge \ x = false \ \wedge n[1] = I \ ) \wedge (n[1] = T \wedge n[2] = I \wedge x = true) \\
(n[2] = C \ \wedge \ x = false \ \wedge n[1] = T \ ) \wedge (n[1] = T \wedge n[2] = I \wedge x = true) \\
(n[2] = E \ \wedge \ x = false \ \wedge n[1] = I \ ) \wedge (n[1] = T \wedge n[2] = I \wedge x = true) \\
(n[2] = E \ \wedge \ x = false \ \wedge n[1] = T \ ) \wedge (n[1] = T \wedge n[2] = I \wedge x = true) \\
(n[2] = T \ \wedge \ x = true \ \wedge n[1] = I \ ) \wedge (n[1] = T \wedge n[2] = I \wedge x = true) \\
(n[2] = T \ \wedge \ x = true \ \wedge n[1] = T \ ) \wedge (n[1] = T \wedge n[2] = I \wedge x = true) \\
(n[2] = T \ \wedge \ x = false \ \wedge n[1] = C \ ) \wedge (n[1] = T \wedge n[2] = I \wedge x = true) \\
(n[2] = T \ \wedge \ x = false \ \wedge n[1] = E \ ) \wedge (n[1] = T \wedge n[2] = I \wedge x = true)
\end{gathered} \tag{3-3}
$$

$$
\begin{aligned}
&(n[2]=I\ \wedge\ x=true\ \wedge n[1]=I\ )\wedge(n[1]=T\wedge n[2]=I\wedge x=false)\\
&(n[2]=I\ \wedge\ x=true\ \wedge n[1]=T\ )\wedge(n[1]=T\wedge n[2]=I\wedge x=false)\\
&(n[2]=I\ \wedge\ x=false\ \wedge n[1]=C\ )\wedge(n[1]=T\wedge n[2]=I\wedge x=false)\\
&(n[2]=I\ \wedge\ x=false\ \wedge n[1]=E\ )\wedge(n[1]=T\wedge n[2]=I\wedge x=false)\\
&(n[2]=C\ \wedge\ x=false\ \wedge n[1]=I\ )\wedge(n[1]=T\wedge n[2]=I\wedge x=false)\\
&(n[2]=C\ \wedge\ x=false\ \wedge n[1]=T\ )\wedge(n[1]=T\wedge n[2]=I\wedge x=false)\\
&(n[2]=E\ \wedge\ x=false\ \wedge n[1]=I\ )\wedge(n[1]=T\wedge n[2]=I\wedge x=false)\\
&(n[2]=E\ \wedge\ x=false\ \wedge n[1]=T\ )\wedge(n[1]=T\wedge n[2]=I\wedge x=false)\\
&(n[2]=T\ \wedge\ x=true\ \wedge n[1]=I\ )\wedge(n[1]=T\wedge n[2]=I\wedge x=false)\\
&(n[2]=T\ \wedge\ x=true\ \wedge n[1]=T\ )\wedge(n[1]=T\wedge n[2]=I\wedge x=false)\\
&(n[2]=T\ \wedge\ x=false\ \wedge n[1]=C\ )\wedge(n[1]=T\wedge n[2]=I\wedge x=false)\\
&(n[2]=T\ \wedge\ x=false\ \wedge n[1]=E\ )\wedge(n[1]=T\wedge n[2]=I\wedge x=false)
\end{aligned}
\tag{3-4}
$$

基于决策树的不变式判断方法与显式状态枚举法相似，两者均是利用带参协议实例中的所有可达状态，来验证候选不变式是否为带参协议实例的不变式。

显式状态枚举法会测试每个可达状态与候选不变式，如果候选不变式在任意可达状态下都得到满足，则判定该候选不变式是带参协议实例的不变式。

然而，在基于决策树的不变式判断方法中，决策树会根据所有相关特征对可达状态进行分类，从而将“候选不变式在任意可达状态中满足”的判断条件，变为了“候选不变式在任意可达状态类别中满足”，提升了判断候选不变式的效率。

在本实验中选择决策树算法对带参协议实例可达状态分类的理由如下：

(1) 相比支持向量机（Support Vector Machine, SMV）等其他分类算法，决策树算法的结果更加直观，有利于探索机器学习算法在形式化验证中的应用。

(2) 决策树算法能够处理非线性特征，并且考虑了变量之间的相互作用。这一点使决策树能够顺利应用于带参协议验证。

(3) 决策树算法已被应用于循环程序的不变式寻找方法。这证明决策树算法能够用于形式化验证。

在选取训练数据时，基于决策树的不变式判断方法使用对称削减可达状态集合作为训练数据，减少了获取训练数据、训练决策树和判断候选不变式所需要的时间消耗和内存占用。然而，由于对称削减可达状态集合只是可达状态集合的一部分，单纯使用称削减可达状态集合判断候选不变式会发生错误。例如在 3.1.2 章中，如果使用表 3-1 判断候选不变式$\neg(n[2]=C)$，则会判断$\neg(n[2]=C)$是带参协议实例的不变式。然而，如果使用未使用对称削减的可达状态集合判断候选不变式，如表

3-3 所示，$\neg(n[2]=C)$则不是带参协议实例的不变式，因为表 3-3 中存在$n[2]=C$的可达状态。

表 3-3 筛选后的可达状态合集

Table 3-3 Selected Reachable States

状态	n[2]
s_1	I
s_2	I
s_3	T
s_4	I
s_5	T
s_6	C
s_7	I
s_8	T
s_9	C
s_{10}	E
s_{11}	T
s_{12}	E

出现上述问题的原因是在对称削减后，部分可达状态被忽略，这导致对称削减可达状态集合会将被忽略的可达状态判断为不可达状态。例如在参数均为 2 的简单互斥协议实例中，对称削减可达状态集合（详见表 2-2）会判断$s_3=\{I,T,true\}$（s_3详见表 2-1）是不可达状态，这是因为s_3不在对称削减可达状态集合内。

同理，由于决策树路径是对可达状态集合的分类，这使本章提出的不变式判断方法也受对称削减的影响，会将所有代表被忽略可达状态的候选不变式判断为带参协议实例的不变式。因为候选不变式的第二部分不会出现于对称削减可达状态集合中。

为了解决对称削减后的不变式判断问题，基于决策树的不变式判断方法不仅会判断候选不变式的原型，而且会判断候选不变式的所有对称形式。如果候选不变式的原型与所有对称形式均通过决策树路径的判断，则判断该候选不变式是带参协议实例的不变式。判断候选不变式间的对称关系与判断带参协议实例状态间的对称关系同理，在保持节点排列一致的情况下，通过变化候选不变式的节点名称，即可获得候选不变式的对称形式。例如对于候选不变式$\neg(n[2]=C)$，它的对称形式为$\neg(n[1]=C)$。

上述针对称削减后的不变式判断问题的解决方法是有效的。由于使用对称削减，带参协议实例的可达状态空间被分为多个等价的子空间。由于决策树路径集合

只能代表其中一个子空间的分类情况，因此该决策树路径集合只能判断与其代表子空间相关的候选不变式。因此，为了使决策树路径集合正常工作，需要找到候选不变式在决策树路径集合对应的子空间中的形式，并使用决策树路径对其验证。然而，由于有关子空间的信息过少，难以直接得到决策树路径集合对应子空间中候选不变式的形式。因此，基于决策树的不变式判断方法通过遍历候选不变式的所有对称形式，保证决策树路径集合能够判断属于同一子空间的候选不变式形式。

上述针对称削减问题的解决方法借鉴于 Murphi 语言中对不变式的定义[4]。在 Murphi 语言中，如果需要使用对称削减功能，则会使用 forall 语句定义候选不变式。例如会将$\neg(n[2] = C)$定义为“forall p : NODE do !(n[p] = C) end;”。其中 NODE 代表节点类型。

3.1.5 剪枝技术

剪枝（Pruning）技术是提升决策树精度，解决训练数据中噪音问题的决策树优化方法。这种方法将划分过细的叶子节点去掉或回退到更高的节点，使父节点或更高的节点变为叶子节点。然而这种优化方法不适用于使用决策树获取不变式。无论是 Garg P 等人提出的 ICE 方法，还是 Siddharth Krishna 等人提出的寻找不变式方法均没有使用决策树剪枝技术，两者给出的理由是应用决策树剪枝技术会降低不变式的精度。

对于本文提出的基于决策树的不变式判断方法，决策树剪枝技术也不适合优化决策树。其原因有两点，第一是训练数据中不含噪音数据。由于训练数据是带参协议实例的对称削减可达状态集合，因此每一个训练数据均是必不少的，也就不存在噪音数据。第二是决策树需要完整体现对称削减可达状态集合的分类。通过使用针对带参协议实例的决策树算法，决策树能够高效且完整地体现可达状态集合的分类，如果剪枝优化发挥作用，可能会导致决策树丢失某些可达状态分类。因此，基于决策树的不变式判断方法没有使用剪枝技术优化决策树。

3.2 基于决策树的不变式判断方法与 ParaVerifier 方法的结合

本文将基于决策树的不变式判断方法与 ParaVerifier 方法结合，提出了基于决策树的带参协议自动化验证方法。其中结合方法是使用基于决策树的不变式判断方法替代 ParaVerifier 方法中使用的模型检测方法（NuSMV 或 Murphi）。因此 ParaVerifier 方法中的 InvFinder 框架将会出现变化。

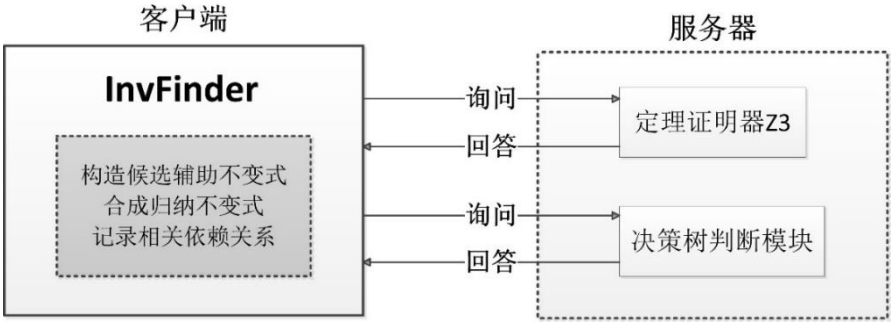

图 3-7 InvFinder 程序架构（决策树）

Figure 3-7 The Program Architecture of InvFinder (Decision Tree)

图 3-7 中展示的是基于决策树的带参协议自动化验证方法中的 InvFinder 框架。InvFinder 中的模型检测程序模块被替换为决策树判断模块。其中，决策树判断模块是使用基于决策树的不变式判断方法实现的。将基于决策树的不变式判断方法与 ParaVerifier 方法结合的原因如下：

(1) 使决策树方法能够应用于带参协议验证领域。

(2) 在判断大型带参协议实例的候选不变式时，相比模型检测程序模块（NuSMV、Murphi），决策树判断模块能够使用更少的时间消耗与内存占用完成判断工作。

相比 ParaVerifier 方法，基于决策树的带参协议自动化验证方法的优势在于使用了基于决策树的不变式判断方法。对于不同的模型检测程序模块，决策树判断模块的优势各有不同。对于是基于 NuSMV 的模型检测程序模块，其工作步骤均可分为：

(1) 接收候选不变式与带参协议实例内容，生成 NuSMV 模型。

(2) 根据 NuSMV 模型，获取带参协议实例的可达状态集合。

(3) 根据可达状态集合判断候选不变式是否为带参协议实例的不变式。

(4) 返回判断结果。

基于 NuSMV 的模型检测程序模块判断候选不变式的原理是，使用 NuSMV 模型检测程序验证包含带参协议实例与候选不变式的 NuSMV 模型。NuSMV 是一种基于符号化可达状态分析的模型检测软件，在获得符号化的可达状态集合后，NuSMV 可以快速对候选不变式进行判断。这一特点使 NuSMV 在验证中小型带参协议实例时，判断速度快于基于显式状态搜索的模型检测软件（如 Murphi）。然而 NuSMV 在验证大型带参协议实例时需要大量的时间和内存来获得符号化的可达状态集合，这对试验设备的内存要求极高。例如对于参数均为 2 的 Flash 协议，基

于 NuSMV 的模型检测程序模块需要 100GB 内存左右的服务器与 5 个小时左右的时间才能得到符号化的可达状态集合。对于是基于 Murphi 的模型检测程序模块，其工作步骤均可分为：

(1)接收候选辅助不变式与带参协议实例内容，生成 Murphi 模型。

(2)根据 Murphi 模型寻找新的可达状态，并判断新的可达状态是否满足候选不变式，如果满足则继续寻找新的可达状态。否则判断候选不变式不是带参协议实例的不变式。

(3)如果在找完所有可达状态后，候选不变式没有被判定为不是带参协议实例的不变式。则判定候选不变式是带参协议实例的不变式。

(4)返回判断结果。

基于 Murphi 的模型检测程序模块的原理是，使用 Murphi 模型检测程序验证包含带参协议实例与候选不变式的 Murphi 模型。Murphi 是一种基于显式状态搜索的模型检测软件。相比 NuSMV，Murphi 在验证大型带参协议实例时占用内存更小。由于大型带参协议实例的可达状态空间巨大，生成符号化的可达状态集合会耗费大量的时间与内存，这使实验设备容易因内存问题导致死机，使验证失败。而由于 Murphi 只是单纯寻找可达状态集合，因此虽然验证所需的时间比 NuSMV 长，但对实验设备的内存要求远比 NuSMV 低。往往经过长时间的运算，Murphi 能够完成大型带参协议实例的验证。

然而在 ParaVerifier 方法中，基于 Murphi 的模型检测程序模块并没有实际应用。这是因为在验证 Murphi 模型时，Murphi 每次找到一个可达状态便会立刻判断候选不变式是否满足，这使 Murphi 在面对不同候选不变式时会重新寻找带参协议实例的可达状态，导致带参协议验证的时间消耗巨大。

决策树判断模块是基于算法 3 工作的，其主要工作步骤均可分为：

(1)获取可达状态集合。

(2)接收候选不变式。

(3)查看候选不变式是否被判断过，如果有判断记录，则返回判断结果。如果没有进入第四步。

(4)查看生成过的决策树路径能否用于判断候选不变式，如果能，则使用生成过的决策树路径判断候选不变式并返回结果。如果没有，则进入第五步。

(5)根据候选不变式，预处理训练数据。

(6)使用训练数据训练决策树，并将决策树转化为决策树路径，使用决策树路径判断候选不变式。

(7)记录决策树路径和候选不变式的判断结果。

(8)返回判断结果。

与基于 NuSMV 的模型检测程序模块相比，决策树判断模块在获取带参协议实例的可达状态集合时，需要的时间更少，占用的内存更低。在基于决策树的不变式判断方法中，可达状态集合的获取方法源于 Murphi 中的显式状态搜索方法和对称削减方法。获取对称削减可达状态集合与获取符号化的可达状态集合相比，所需要的时间与内存更少。

与基于 Murphi 的模型检测程序模块相比，决策树判断模块不仅能够重复利用收集到的可达状态集合，而且判断候选不变式的效率更高。这是因为基于决策树的不变式判断方法发挥了决策树的分类功能，将“按可达状态逐个判断候选不变式”变为“按可达状态类别逐个判断候选不变式”，提升了候选不变式的判断效率。

除上述优点之外，决策树判断模块还有一套复用机制，其中包括候选不变式的复用机制与决策树路径的复用机制。

关于候选不变式的复用机制。决策树判断模块在接收候选不变式$\neg(\bigwedge_{n=1}^{k} f_n)$后，会对$\neg(\bigwedge_{n=1}^{k} f_n)$进行再排序。其中，排序的对象是各个$f$和$f$中左值与右值的位置。$\bigwedge_{n=1}^{k} f_n$是候选不变式的第二部分，该部分由判断公式（$e_1 = e_2$和$e_1 \neq e_2$）与“$\wedge$”组成，因此互换判断公式的左值和右值对第二部分的含义没有影响，重新排列各个判断公式的顺序也对第二部分的含义没有影响。例如候选不变式$\neg(n[1] = T \wedge n[2] = I \wedge x = true)$与候选不变式$\neg(n[2] = I \wedge n[1] = T \wedge true = x)$均表达状态$\{T, I, true\}$不是带参协议实例的可达状态。

为了保证$\neg(\bigwedge_{n=1}^{k} f_n)$所表达的带参协议实例性质不会被重复判断，决策树判断模块根据 ASCII 值重新排列$\neg(\bigwedge_{n=1}^{k} f_n)$，并将$\neg(\bigwedge_{n=1}^{k} f_n)$的判断结果与$\neg(\bigwedge_{n=1}^{k} f_n)$排列后的形式成对保存。如果之后需要判断候选不变式$\neg(\bigwedge_{n=1}^{k} f_n')$,并且$\neg(\bigwedge_{n=1}^{k} f_n')$排序后的形式与$\neg(\bigwedge_{n=1}^{k} f_n)$ 排列后的形式相等，那么可以直接将$\neg(\bigwedge_{n=1}^{k} f_n)$的判断结果作为$\neg(\bigwedge_{n=1}^{k} f_n')$的判断结果。

关于决策树路径的复用机制。在验证带参协议实例时，决策树判断模块会判断许多包含特征种类相同的候选不变式，如$\neg(\mathrm{n}[1] = \mathrm{T} \wedge \mathrm{n}[2] = \mathrm{T})$与$\neg(\mathrm{n}[1] = \mathrm{I} \wedge \mathrm{n}[2] = \mathrm{I})$。由于这些候选不变式包含的特征种类一致，因此对应的训练数据也相同。这导致这些候选不变式对应的决策树和决策树路径都相同。为了避免重复生成决策树路径，决策树判断模块在获得候选不变式包含的特征集合后，会判断是否根据该特征集合生成过决策树路径，如果有，则直接使用生成过的决策树路径判断候选不变式是否为带参协议实例的不变式。此外，如果生成了新的决策树路径，则会将新的决策树路径与其对应的特征集合一起保存，以用于策树路径的复用。

基于上述优点，决策树判断模块更加适合判断 InvFinder 询问的候选辅助不变式。这使基于决策树的带参协议自动化验证方法能够使用更少的时间，占用更少的内存，找到带参协议的归纳不变式并完成带参协议验证。

算法 3：决策树判断算法

算法 3 中展示的是决策树判断算法

输入：候选辅助不变式$\neg f$，训练数据 data（对称削减可达状态集合）

输出：布尔变量（ture 代表不变式，false 代表不是不变式）

```
if ¬f被判断过 then return 被记录的判断结果
else:
    if ¬f中的变量集合有对应的决策树路径文件 then
        invlist ← ¬f与¬f的对称结果
        for member in invlist:
            result ← 使用路径文件判断 member 是否为不变式
            if result 中的值代表不是不变式 then
                记录¬f的判断结果为 false
                return false
            endif
        endfor
        记录¬f的判断结果为 true
        return true
    else:
        invlist ← ¬f与¬f的对称结果
        for member in invlist:
            data' ← 根据 member 中的特征判断训练数据
            tree ← 根据训练数据训练决策树
            path ← 根据 tree 获得路径文件
            将路径文件与 member 中的特征信息成对存储
            result ← 使用路径文件判断 member 是否为不变式
            if result 中的值代表不是不变式 then
                记录¬f的判断结果为 false
                return false
            endif
        endfor
        记录¬f的判断结果为 true
        return true
    endif
endif
```

3.3 本章小结

本章主要用于介绍基于决策树的带参协议自动化验证方法的设计思路。其中主要介绍了基于决策树的不变式判断方法与基于决策树的不变式判断方法和ParaVerifiver 方法的结合。基于决策树的不变式判断方法是基于决策树的带参协议自动化验证方法的核心。由于使用了决策树的分类性质，基于决策树的不变式判断方法能够高效判断候选不变式。基于决策树的带参协议自动化验证方法是第一个将决策树方法应用于带参协议验证的方法。相比 ParaVerifiver 方法，基于决策树的带参协议自动化验证方法能够使用更少的时间与内存，完成大型带参协议实例的候选不变式判断工作。

4 基于决策树的带参协议自动化验证方法的实现

使用基于决策树的带参协议自动化验证方法的带参协议验证程序与ParaVerifier 相比，主要的区别是使用决策树判断模块替换了模型检测程序模块。因此本章主要介绍决策树判断模块的实现。决策树判断模块包括两部分，分别是对称削减可达状态集合收集器与基于决策树的不变式判断服务器。

4.1 对称削减可达状态集合收集器

对称削减可达状态集合收集器用于自动获取带参协议实例的对称削减可达状态集合。本程序改编自模型检测软件 Murphi 的最新版本（CMurphi 3.4.x）。选择修改 Murphi 得到对称削减可达状态集合收集器的原因有四点，它们分别是：

(1) Murphi 是一种基于显式状态搜索的模型检测软件。

(2) Murphi 具有对称削减功能。

(3) Murphi 经过多个版本的修改，是一流的模型检测软件。

(4) Murphi 是开源模型检测软件。

在上述原因中，选择修改 Murphi 得到对称削减可达状态集合收集器的最主要原因是第一点与第二点。

CMurphi 是一种由 C++语言编写的基于显式状态搜索的模型检测软件，这不仅让 CMurphi 具有较高的运行效率，而且将带参系统实例的可达状态封装为类，这使 CMurphi 能够打印每个可达状态。而对于基于符号化可达状态分析的模型检测软件（如 NuSMV），可达状态集合采用符号化存储，因此难以打印每一个带参协议实例的可达状态。

CMurphi 拥有对称削减功能并能够找到对称削减可达状态集合。在验证带参协议实例时，CMurphi 在找到新的可达状态后，会将新的可达状态存入哈希表。其中，判断找到的可达状态不是新的可达状态的依据有两点，它们分别是：

(1) 找到的可达状态已存于哈希表中。

(2) 找到的可达状态的对称形式已存于哈希表中。

无论上述两点中的任意一点成立，找到的可达状态会被 CMurphi 当做重复可达状态，并且不会将其加入到对称削减可达状态集合中。因此，CMurphi 能够提供称削减可达状态集合。

虽然 CMurphi 具有生成并打印对称削减可达状态集合的功能，但是其中存在两个问题，它们分别是：

(1) CMurphi 每次找到一个新的可达状态，便会立即验证候选不变式。

⑵CMurphi 会以报告的形式打印对称削减可达状态集合。

由于 CMurphi 是模型检测程序，而不是专门用于获取对称削减可达状态的收集器，因此在寻找对称削减可达状态时会同时验证候选不变式，降低了获取对称削减可达状态集合的效率。其中，验证候选不变式的过程是难以避免的，即使在 Murphi 模型中不定义候选不变式，CMurphi 仍然会执行验证流程。

CMurphi 在打印对称削减可达状态集合时会以报告的形式打印，其中包含了许多解释性的信息，如果不经过处理，难以直接应用于训练决策树。

为了解决上述两个问题，本文通过修改 CMurphi 3.4.x，完成了专门用于获取带参协议实例对称削减可达状态集合的收集器。

图 4-1 中展示的是由 CMurphi 生成的带参协议实例验证程序的流程图。图 4-2 中展示的是由修改后的 CMurphi 生成的对称削减可达状态集合收集器的流程图。通过两图对比可以看出，对称削减可达状态集合收集器只用于收集带参协议实例的对称削减可达状态集合，而不会存在候选不变式判断环节。此外，对称削减可达状态集合收集器会打印出纯净的对称削减可达状态集合信息。

在验证带参协议实例前，CMurphi 会先使用 Mu 程序，将包含带参协议实例定义与候选不变式的 Murphi 文件编写为 cpp 文件，其中 cpp 文件包含带参协议实例的变量定义、初始状态定义、规则定义和候选不变式定义。在此之后，CMurphi 根据 cpp 文件与 CMurphi 的 include 库，使用 g++命令生成带参协议实例验证程序。最后通过运行带参协议实例验证程序，完成对带参协议实例的验证。

为了获得对称削减可达状态集合收集器，本文对 CMurphi 做出了以下修改。修改 CMurphi 中的 Mu 程序。Mu 程序负责生成带参协议实例对应的 cpp 文件。cpp 文件包含带参协议实例的具体定义。对 Mu 程序的主要修改内容是向生产的 cpp 文件中添加表头打印函数。表头打印函数负责打印带参协议实例状态中每个特征对应的名称，由于表头的相关信息存储于 cpp 文件中，因此表头打印函数在 cpp 文件中实现。

修改 CMurphi 中的 include 库。CMurphi 中的 include 库是构成带参协议实例验证程序的主要代码。include 库的修改内容包括实现内容打印函数和删除候选不变式的判断环节。内容打印函数负责打印每个对称削减可达状态中各个特征的取值。内容打印函数与 cpp 文件中的表头打印函数共同完成对称削减可达状态集合的打印工作。内容打印函数的工作原理是在搜寻新的可达状态时，记录找到的新的可达状态，并按照表头中的特征顺序，依次打印新的可达状态的每一个特征取值。关于删除候选不变式的判断环节。具体方法是删除判断候选不变式的相关代码，使对称削减可达状态集合收集器专注于搜寻对称削减可达状态集合。

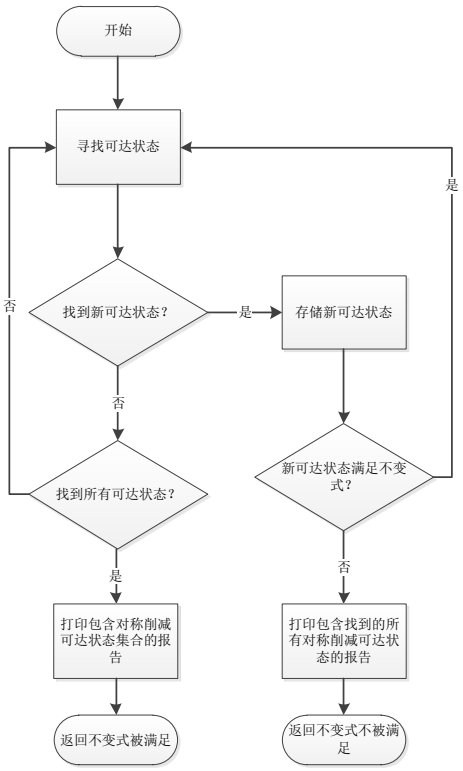

图 4-1 CMurphi 的流程图

Figure 4-1 Flow Chart of CMurphi

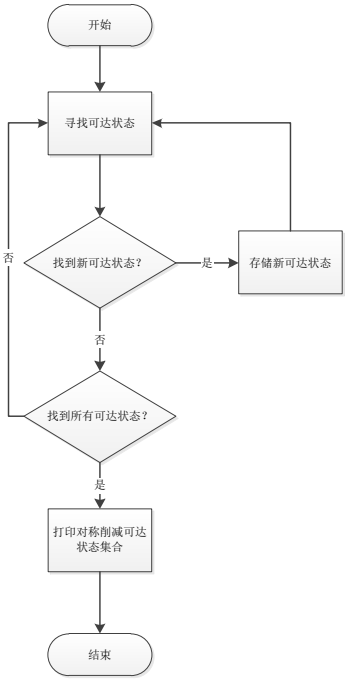

图 4-2 对称削减可达状态集合获取器的流程图

Figure 4-2 Flow Chart of Getter of Symmetric Reduced Reachable State Set

4.2 基于决策树的不变式判断服务器

基于决策树的不变式判断服务器的功能是接收 InvFinder 客户端传来的候选不变式（候选辅助不变式），并根据对称削减可达状态集合获取器获得的对称削减可达状态集合生成决策树路径，判断候选不变式是否为带参协议实例的不变式。

基于决策树的不变式判断服务器使用 Python 语言实现。按功能划分可分为通讯模块、数据预处理模块、复用模块、决策树路径生成模块、候选不变式判断模块。各个模块的协作方式如图 4-3 所示。

图 4-3 描绘的是基于决策树的不变式判断服务器单次判断候选不变式的流程图。在图 4-3 中，通过颜色区别每个流程所属的模块，其中红色流程代表通讯模块，蓝色流程代表数据预处理模块，黄色流程代表复用模块，绿色流程代表决策树路径生成模块，黑色流程代表候选不变式判断模块。

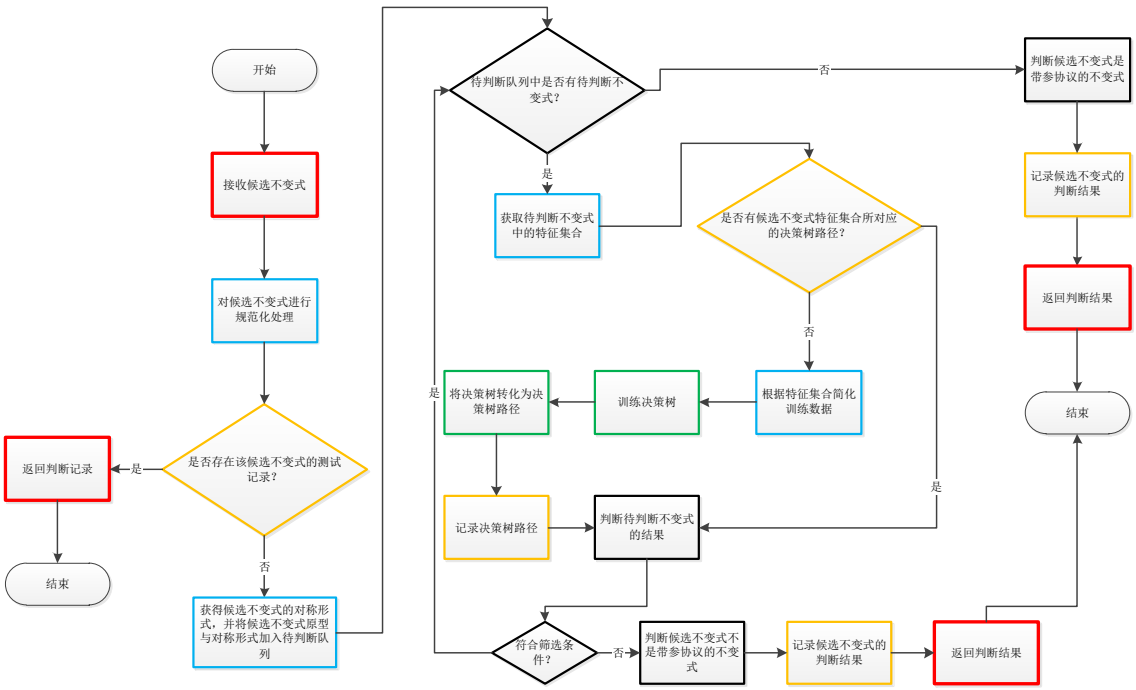

图 4-3 基于决策树的不变式判断服务器的流程图

Figure 4-3 Flow Chart of Invariant Screening Server Based on Decision Tree

通讯模块的功能是从 InvFinder 客户端接收候选不变式，并向 InvFinder 客户端发送候选不变式的判断结果。通讯模块基于 socket 实现，使用的模式是 AF_INET。AF_INET 模式代表使用 IPv4 进行通信。在基于决策树的带参协议自动化验证方法的实现中，基于决策树的不变式判断服务器与 InvFinder 客户端使用局域网连接，使用 IPv4 进行通信更有利于局域网通讯。图 4-3 中只列出单次判断候选不变式的

流程，然而在实际运行中，每当候选不变式的判断结果被发送后，通讯模块会立即等待新的候选不变式并接收，使基于决策树的不变式判断服务器持续判断 InvFinder 客户端传来的候选不变式。

数据预处理模块用于为复用功能、生成决策树、判断候选不变式提供需要的数据。其功能包括对候选不变式进行规范化处理、获得候选不变式的对称形式、获得候选不变式的特征集合、训练数据预处理。对候选不变式进行规范化处理是指对候选不变式的结构进行排序，用于实现候选不变式的复用机制。获得候选不变式的对称形式用于克服对称削减造成的判断问题。获得候选不变式的特征集合用于简化训练数据与标记决策树路径。训练数据预处理是指简化训练数据与挑选目标特征。

复用模块的功能是实现候选不变式与决策树路径的复用机制。关于候选不变式的复用机制。每当新的候选不变式被判断后，复用模块会将候选不变式的规范化形式作为键，候选不变式的判断结果作为键值，把它们存入用于记录候选不变式判断结果的字典中。如此一来，当需要判断其他候选不变式时，可以通过检索候选不变式判断结果字典，快速得到可复用的判断记录。关于决策树路径的复用机制。每当生成新的决策树路径，复用模块以生成决策树路径所用的特征集合为键，以决策树路径为键值，把它们存入用于记录决策树路径的字典中。如此一来，当需要根据特征集合生成决策树路径时，可以先查询记录决策树路径的字典，如果该特征集合已生成过决策树路径，则可立即返回所需的决策树路径。

决策树路径生成模块的功能是生成决策树并将决策树转换为决策树路径。决策树的生成方法如算法 2 所描述。其中，生成的决策树以字典的形式保存。决策树路径的转换方法是搜寻从根节点到叶子节点的每条路径，并将其转化为如公式(3-2)一样的字符串。整个决策树路径集合以列表形式保存。

候选不变式判断模块的功能是判断候选不变式是否为带参协议实例的不变式。候选不变式判断模块会依次判断候选不变式原型与所有的对称形式。如果原型与所有对称形式均通过判断，则判定候选不变式是带参协议实例的不变式。否则判定候选不变式不是带参协议实例的不变式。在判断候选不变式（原型或对称形式）时，候选不变式判断模块使用 Z3 定理证明器的 Python 库函数判断决策树路径与候选不变式的连接形式。每组决策树路径与候选不变式的判断过程为：首先将候选不变式与一条决策树路径连接，得到如公式(3-3)形式的字符串；然后将候选不变式与决策树路径的连接形式转换为前缀表达式；最后使用 Z3 判断前缀表达式是否矛盾。上述判断过程会依次对所有决策树路径与候选不变式的组合进行判断，如果所有判断结果均为 unsat（矛盾），则判定候选不变式通过判断。如果某条决策树路径与候选不变式的连接结果被判断为 sat（不矛盾），则会立刻终止判断。判定候选不变式没有通过判断。

4.3 本章小结

本章主要介绍了基于决策树的带参协议自动化验证方法的实现。由于除决策树判断模块外的其它模块均复用于 ParaVerifiver，因此本章主要介绍决策树判断模块的实现。决策树判断模块共分为对称削减可达状态集合收集器与基于决策树的不变式判断服务器，本章使用流程图与文字介绍等方法对两者的实现方法进行了介绍。

5 试验与结果分析

本章的主要内容是介绍基于决策树的带参协议自动化验证方法对六种带参协议的验证效果，并通过与 ParaVerifier 方法的试验效果进行比较，说明基于决策树的带参协议自动化验证方法的优势。

本文将基于决策树的带参协议自动化验证方法应用于 6 种带参协议。它们分别是 Mutual Exclusion 协议（简单的互斥协议）、MOESI[55]协议、MESI[56]协议、German 协议[2]、Flash 协议简化版和 Flash 协议。其中 Mutual Exclusion 协议、MOESI 协议和 MESI 协议属于小型带参协议。German 协议、Flash 协议简化版和 Flash 协议属于中型或大型协议。在这些带参协议中，Flash 协议是最复杂且最接近真实缓存一致性协议的带参协议。对于节点数量均为 3 的 Flash 协议实例，其可达状态数量是节点数量均为 3 的 German 协议实例的 1500 倍。关于 Flash 协议简化版，该版本为去掉 data 属性后的 Flash 协议，本文称其为 Flash_nodata 协议。

在实验过程中，为了与 ParaVerifier 方法进行比较，本文使用 6 种带参协议与 ParaVerifier 测试时使用的一致[34]18。带参协议实例的参数选取与 ParaVerifier 方法试验中的参数选取相同。

由于基于决策树的带参协议自动化验证方法与 ParaVerifier 方法的区别为 InvFinder，因此在下文的性能比较中没有考虑 ProofGen 与 Isabelle。

本文的实验环境是拥有 Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz 处理器，8GB 内存，使用 64 位 Linux 3.15.10 系统的服务器。

基于决策树的带参协议自动化验证方法对于 6 种带参协议的测试结果如表 5-1 所示。表中包含带参协议、时间和内存 3 种数据。其中，带参协议指带参协议的名称。时间指使用决策树判断模块的 InvFinder 找到归纳不变式所需要的时间。内存指决策树模块在判断候选不变式时所占用的最大内存。

表 5-2 中展示的是 ParaVerifier 的试验结果[34]。其中，测试环境与基于决策树的带参协议自动化验证方法的测试环境相同。虽然表中也包含带参协议、时间和内存 3 种数据，但是其中的时间和内存的含义与表 5-1 中的不同。在表 5-2 中，时间指在生成可达状态集合后，InvFinder 找到归纳不变式所需要的时间。内存指在生成可达状态集合后，NuSMV 模型检测程序模块在判断候选不变式时占用的最大内存。

在表 5-1 中，无论是时间还是内存都计算了生成可达状态集合的消耗，并且可达状态集合生成的时间与内存消耗均很小。例如，对于参数均为 2 的 Flash 协议实例，生成对称削减可达状态集合所需要的时间为 6-7 分钟，而所需内存为 108MB 左右。然而对于 NuSMV 模型检测程序模块，生成可达状态集合需要大量的时间与

内存。例如对于参数均为 2 的 Flash 协议，使用 NuSMV 生成符号化的可达状态集合需要 100GB 左右内存的服务器与 4 个小时左右的运算时间。

表 5-1 基于决策树的不变式判断方法的实验结果

Table 5 -1 The Experimental Results of Decision Tree Screening

带参协议	时间（秒）	内存（MB）
Mutual Exclusion	8.39	25
MOESI	4.67	25
MESI	4.81	25
German	64.89	42
Flash_nodata	1185.09	511
Flash	7152.68	3200

表 5-2 ParaVerifier 的实验结果（不包含获取可达状态集合的消耗）

Table 5-2 The Experimental Results of ParaVerifier（Consumption not including acquisition of reachable state sets）

带参协议	时间（秒）	内存（MB）
Mutual Exclusion	3.52	25
MOESI	4.67	85
MESI	4.81	84
German	38.67	135
Flash_nodata	280.00	2400
Flash	510.00	4800

由于 InvFinder 需要生成可达状态集合并判断候选不变式。因此，综合考虑表 5-1、表 5-2 与 NuSMV 生成可达状态集合的消耗，相比 ParaVerifier 方法，基于决策树的带参协议自动化验证方法能够使用更少的时间、占用更少的内存完成带参协议的判断。

由于使用基于决策树的带参协议自动化验证方法能够减少对设备内存的要求，这使基于决策树的带参协议自动化验证方法能够在笔记本电脑上使用仅有 4GB 的虚拟机完成 Flash 协议的验证。然而 ParaVerifier 方法由于需要大量的内存生成可达状态集合，因此难以完成对 Flash 协议的验证。

6 结论与展望

基于决策树的带参协议自动化验证方法，用于自动验证由卫式命令语言描述的缓存一致性协议。这种方法的设计目的是为了探索决策树方法与带参协议验证的结合方法，为带参协议的不变式自动寻找提供新的方法与思路。

基于决策树的带参协议自动化验证方法是一种半黑盒半白盒的技术。一方面，这种方法需要通过对带参协议的分析获得需要验证的候选不变式。另一方面，这种方法在判断候选不变式时完全不知道带参协议的内容，只根据可达状态集合和决策树算法来判断候选不变式是否为带参协议实例的不变式。

基于决策树的带参协议自动化验证方法是第一个将决策树方法应用于带参协议验证的方法，其中主要创新点如下：

(1)提出了使用对称削减可达状态集合作为决策树训练数据的方法。在其他将决策树应用于寻找不变式的工作中，训练数据由测试程序的可达状态与不可达状态组成，但是这种训练数据选取方式不适用于带参协议验证。因此，基于决策树的带参协议自动化验证方法使用对称削减可达状态集合作为决策树的训练数据，使决策树能够准确且高效地判断候选不变式。

(2)提出了针对带参协议实例的决策树算法。在基于决策树的带参协议自动化验证方法中，传统决策树算法会忽略判断候选不变式所需的关键信息。为了配合基于决策树的带参协议自动化验证方法，使决策树包含候选不变式的所有相关特征，本文提出了针对带参协议实例的决策树算法。针对带参协议实例的决策树算法能够在生成决策树后添加与分类无关的相关特征信息，并保留分类后目标特征的全部可能取值，使决策树能够准确判断候选不变式。

(3)提出了使用决策树判断候选不变式的方法。由于以直接归纳决策树获得不变式的方法难以应用于带参协议验证，因此本文改变决策树的使用策略，提出了基于决策树的不变式判断方法。这种方法用于判断候选不变式，并以此辅助寻找归纳不变式。基于决策树的不变式判断方法的原理是利用决策树路径与候选不变式的关系，从而使用可达状态类别判断候选不变式。相比具有相同功能的模型检测程序（NuSMV、Murphi），在判断大型带参协议实例的候选不变式时，使用基于决策树的不变式判断方法，构建的决策树判断模块可以使用更少的时间、占用更少的内存完成候选不变式的判断。

基于决策树的带参协议自动化验证方法是基于决策树的不变式判断方法与 ParaVerifier 方法结合的产物。然而在验证大型带参协议时，基于决策树的带参协议自动化验证方法比 ParaVerifier 方法在时间消耗、内存占用方面更有优势。基于决策树的带参协议自动化验证方法能够在笔记本电脑上使用仅有 4GB 的虚拟机完

成 Flash 协议的验证，这是 ParaVerifier 方法难以做到的。

在未来的工作中，基于决策树的带参协议自动化验证方法将尝试验证缓存一致性协议类型以外的带参协议，或者尝试验证循环程序。

参考文献

[1] Dijkstra E W. Guarded commands, non-determinacy and a calculus for the derivation of programs[C]// International Conference on Reliable Software. 1975.

[2] M German, Steven. (2019). Tutorial on Verification of Distributed Cache Memory Protocols[CP/OL]. https://www.researchgate.net/publication/267856070_Tutorial_on_Verification_of_Distributed_Cache_Memory_Protocols.

[3] Norris Ip C, Dill D L. Efficient Verification of Symmetric Concurrent Systems[J]. 1993. 1.

[4] David L. Dill. The murphi verification system[C].//In Proceedings of the 8th International Conference on Computer Aided Verification, CAV 96, London, UK, UK, 1996. Berlin Heidelberg:Springer-Verlag, 1996. 390 - 393.

[5] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without bdds[C].// W. Rance Cleaveland, Tools and Algorithms for the Construction and Analysis of Systems, Berlin, Heidelberg, 1999. Berlin Heidelberg:Springer, 1999. 193 - 207.

[6] McMillan, Kenneth L. Symbolic model checking[M]// Symbolic Model Checking. Springer US, 1993.

[7] Randal E. Bryant. Graph-based algorithms for boolean function manipulation[J]. IEEE Trans. Comput., 1986, 35(8):677 - 691.

[8] Armin Biere, Keijo Heljanko, and Siert Wieringa. AIGER 1.9 and beyond[R]. 4040 Linz, Austria: Johannes Kepler University, Altenbergerstr, 2011. 69.

[9] Niklas Een and Armin Biere. Effective preprocessing in sat through variable and clause ´elimination[C].// In Fahiem Bacchus and Toby Walsh, editors, Theory and Applications of Satisfiability Testing, Berlin, Heidelberg, 2005. Berlin Heidelberg:Springer, 2005. 61 - 75.

[10]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(Z6):77-86.

[11]EMERSON, Allen E, KAHLON, et al. Exact and efficient verification of parameterized cache coherence protocols[J]. Lecture Notes in Computer Science, 2003, 2860:247-262.

[12]Bouajjani A, Habermehl P, Vojnar T. Verification of Parametric Concurrent Systems with Prioritized FIFO Resource Management[J]. Formal Methods in System Design, 2008, 32(2):129-172.

[13]Bingham J, Hu A J. Empirically Efficient Verification for a Class of Infinite-State Systems[C].// International Conference on Tools & Algorithms for the Construction & Analysis of Systems. Springer-Verlag, 2005. 77-92.

[14]Kaiser A, Kroening D, Wahl T. Dynamic Cutoff Detection in Parameterized Concurrent Programs[M]. Computer Aided Verification. DBLP, 2010. 645-659.

[15]Mcmillan K L. Parameterized Verification of the FLASH Cache Coherence Protocol by Compositional Model Checking[M]// Correct Hardware Design and Verification Methods.

Springer Berlin Heidelberg, 2001.179-195.

[16]Chou C T, Mannava P K, Park S. A Simple Method for Parameterized Verification of Cache Coherence Protocols[J]. Lecture Notes in Computer Science, 2004, 3312:382-398.

[17]Chen, Yang, Gopalakrishnan, et al. Reducing Verification Complexity of a Multicore Coherence Protocol Using Assume/Guarantee[C]// Formal Methods in Computer Aided Design. IEEE, 2006. 12-16.

[18]Chen X, Yang Y, Delisi M, et al. Hierarchical cache coherence protocol verification one level at a time through assume guarantee[C]// 2007 IEEE International High Level Design Validation and Test Workshop. IEEE, 2007. 107-114.

[19]Clarke E, Talupur M, Veith H. Environment Abstraction for Parameterized Verification[C].// International Conference on Verification. Springer-Verlag, 2006. 126-141.

[20]Clarke E M, Talupur M, Veith H. Proving Ptolemy Right: The Environment Abstraction Framework for Model Checking Concurrent Systems[C].// Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedi. Springer-Verlag, 2008. 33-47.

[21]Sethi D, Talupur M, Malik S. Using Flow Specifications of Parameterized Cache Coherence Protocols for Verifying Deadlock Freedom[J]. Computer Science, 2014, 8837:330-347.

[22]Talupur M, Tuttle M R. Going with the flow: Parameterized verification using message flows[C].// Formal Methods in Computer-aided Design. IEEE, 2008. 17-20.

[23]Yi L, Huimin L, Hong P. computing invariants for parameter abstraction[C].// IEEE/ACM International Conference on Formal Methods & Models for Codesign. IEEE Computer Society, 2007. 29-38.

[24]Park S, Dill D L. [ACM Press the eighth annual ACM symposium - Padua, Italy (1996.06.24-1996.06.26)] Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures, - SPAA \"96 - Verification of FLASH cache coherence protocol by aggregation of distributed transactions[C].// Acm Symposium on Parallel Algorithms & Architectures. 1996. 288-296.

[25]Qadeer C F S. Predicate Abstraction for Software Verification[J]. ACM SIGPLAN Notices, 2002, 37(1):191-202.

[26]Baukus K, Lakhnech Y, Stahl K. Parameterized Verification of a Cache Coherence Protocol: Safety and Liveness[C]// Revised Papers from the Third International Workshop on Verification. Springer-Verlag, 2002. 317-330.

[27]Das S, Dill D L, Park S. Experience with Predicate Abstraction[C]// International Conference on Computer Aided Verification. 1999. 160-171.

[28]Park S, Dill D L. [ACM Press the eighth annual ACM symposium - Padua, Italy (1996.06.24-1996.06.26)] Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures, - SPAA \"96 - Verification of FLASH cache coherence

protocol by aggregation of distributed transactions[C]. // Acm Symposium on Parallel Algorithms & Architectures. 1996:288-296. 82-97.

[29]Pnueli A, Ruah S, Zuck L. Automatic Deductive Verification with Invisible Invariants[C]. // International Conference on Tools & Algorithms for the Construction & Analysis of Systems. Springer-Verlag, 2001. 82-97.

[30]Pandav S, Slind K, Gopalakrishnan G. Counterexample guided invariant discovery for parameterized cache coherence verification[C]. // 13 Ifip Wg 105 International Conference on Correct Hardware Design & Verification Methods. Springer-Verlag, 2005. 317-331.

[31]Conchon S, Goel A, Sava Krstić, et al. Cubicle: A Parallel SMT-based Model Checker for Parameterized Systems[C]. // International Conference on Computer Aided Verification. Springer-Verlag, 2013. 718-724.

[32]Conchon S, Goel A, Krstic S, et al. Invariants for Finite Instances and Beyond[C]. // Formal Methods in Computer-Aided Design (FMCAD), 2013. IEEE, 2013. 61-68.

[33]Li Y, Duan K, Lv L, et al. A novel approach to parameterized verification of cache coherence protocols[C]. // IEEE International Conference on Computer Design. IEEE, 2016. 560 - 567.

[34]Yongjian Li, Kaiqiang Duan, David N. Jansen, Jun Pang, Lijun Zhang, Yi Lv, and Shaowei Cai. An automatic proving approach to parameterized verification[J]. ACM Transactions on Computational Logic, 19(4):27:1 - 27:25, December 2018.

[35]Pnueli A, Shahar E. A platform for combining deductive with algorithmic verification[J]. Lecture Notes in Computer Science, 2000, 1102(1102):184-195.

[36]Nikolaj Bjørner, Browne A, Manna Z. Automatic generation of invariants and intermediate assertions[J]. Theoretical Computer Science, 1997, 173(1):49-87.

[37]Li A P. Parameterized Verification with Automatically Computed Inductive Assertions[C]. //International Conference on Computer Aided Verification. Springer-Verlag, 2001. 221 - 234.

[38]Tiwari A, H. Rueß, H. Saïdi, et al. A Technique for Invariant Generation[C]. // International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg, 2001. 113-127.

[39]Fredlund L A. Book Review: Design and Validation of Computer Protocols by Gerard J. Holzmann (Prentice Hall, 1991)[J]. Acm Sigcomm Computer Communication Review, 1991, 21(2):14.

[40]Xiaofang Chen and Ganesh Gopalakrishnan. A general compositional approach to verifying hierarchical cache coherence protocols[R]. Technical report, Technical Report, UUCS-06-014, School of Computing, University of Utah, Salt Lake City, UT 84112 USA. Noverber 26, 2006.

[41]Sava Krstic. Parameterized system verification with guard strengthening and parameter abstraction[CP/OL]. Automated verification of infinite state systems, 2005. https://www. researchgate. net/publication/228946280_Parameterized_system_verification_with_guard_strengthening_and_parameter_abstraction.

[42]Minato S. Shared binary decision diagram with attributed edges for efficient boolean function manipulation[C]// IEEE Design Automation Conference. IEEE, 1991.

[43]Nipkow T. Isabelle/Hol a Proof Assistant for Higher-Order Logic[C]. // Springer-Verlag, 2002.

[44]Bertot Y, Castéran P. Interactive Theorem Proving and Program Development[J]. Texts in Theoretical Computer Science An Eatcs, 2008, 75(3):166-169.

[45]Jialun Cao, Yongjian Li, and Jun Pang. L-cmp: An automatic learning-based parameterized verification tool[C]. // In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018,, New York, NY, USA, 2018. 892 - 895.

[46]Garg P, Christof Löding, Madhusudan P, et al. ICE: A Robust Framework for Learning Invariants[C]. // Biere A , Bloem R . Computer Aided Verification - 26th International Conference, Vienna, July 18-22 2014, Germany: Springer, 2014. 69-87.

[47]Garg P, Neider D, Madhusudan P, et al. Learning invariants using decision trees and implication counterexamples[J]. ACM SIGPLAN Notices, 2016, 51(1):499-512.

[48]Quinlan J R. Induction of decision trees[J]. Machine Learning, 1986, 1(1):81-106.

[49]Quinlan J R. C4.5: programs for machine learning[J]. 1992.

[50]Siddharth Krishna, Christian Puhrsch, Thomas Wies. Learning Invariants using Decision Trees[CP/OL]. arXiv e-prints, page arXiv:1501.04725, January 2015. https://arxiv.org/abs/1501.04725.

[51]Min, Antoine. The octagon abstract domain[J]. Higher-Order and Symbolic Computation, 2006, 19(1):31-100.

[52]Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver[J]. In Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2008, 337 - 340.

[53]Kwiatkowska M, Norman G, Parker D. Symmetry Reduction for Probabilistic Model Checking[C]// Proc International Conference on Computer Aided Verification. 2006.

[54]Loh W Y. Classification and regression trees[J]. Wiley Interdisciplinary Reviews Data Mining & Knowledge Discovery, 2011, 1(1):14-23.

[55]Advanced Micro Devices[M]. AMD64 Architecture Programmer's Manual (2013). 2013.

[56]Papamarcos M S, Patel J H. A low-overhead coherence solution for multiprocessors with private cache memories[J]. Acm Sigarch Computer Architecture News, 1984, 12(3):348-354.

作者简历及攻读硕士学位期间取得的研究成果

一、作者简历

曹泰丰 男 1993.11 出生

2012.9-2016.6 大连理工大学 软件工程 本科

2016.9-2019.6 北京交通大学 软件工程 研究生

二、发表论文

[1] Xiaotao Wei、Taifeng Cao、Wenwei Wu、Lei Su A Distributed Resource Access Algorithm Based on DHT[J].Journal of Computers Vol. 28, No. 4, 2017, pp. 189-196 10.3966/199115592017082804020

三、参与科研项目

［1］基于归纳不变式的带参协议验证。国家重点科研项目，资金源于国家自然科学基金，项目批准号：61672503。

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作和取得的研究成果，除了文中特别加以标注和致谢之处外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京交通大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

学位论文作者签名：　　签字日期：　　年　　月　　日

学位论文数据集

表 1.1： 数据集页

<table>
<tr><td>关键词*</td><td>密级*</td><td>中图分类号</td><td>UDC</td><td>论文资助</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="2">学位授予单位名称*</td><td>学位授予单位代码*</td><td>学位类别*</td><td>学位级别*</td></tr>
<tr><td colspan="2">北京交通大学</td><td>10004</td><td></td><td></td></tr>
<tr><td colspan="2">论文题名*</td><td colspan="2">并列题名</td><td>论文语种*</td></tr>
<tr><td colspan="2"></td><td colspan="2"></td><td></td></tr>
<tr><td>作者姓名*</td><td colspan="2"></td><td>学号*</td><td></td></tr>
<tr><td colspan="2">培养单位名称*</td><td>培养单位代码*</td><td>培养单位地址</td><td>邮编</td></tr>
<tr><td colspan="2">北京交通大学</td><td>10004</td><td>北京市海淀区西直门外上园村 3 号</td><td>100044</td></tr>
<tr><td colspan="2">学科专业*</td><td>研究方向*</td><td>学制*</td><td>学位授予年*</td></tr>
<tr><td colspan="2"></td><td></td><td></td><td></td></tr>
<tr><td>论文提交日期*</td><td colspan="4"></td></tr>
<tr><td>导师姓名*</td><td colspan="2"></td><td>职称*</td><td></td></tr>
<tr><td>评阅人</td><td colspan="2">答辩委员会主席*</td><td colspan="2">答辩委员会成员</td></tr>
<tr><td></td><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td colspan="5">电子版论文提交格式 文本（ ） 图像（ ） 视频（ ） 音频（ ） 多媒体（ ） 其他（ ）
推荐格式：application/msword；application/pdf</td></tr>
<tr><td colspan="2">电子版论文出版（发布）者</td><td>电子版论文出版（发布）地</td><td colspan="2">权限声明</td></tr>
<tr><td colspan="2"></td><td></td><td colspan="2"></td></tr>
<tr><td>论文总页数*</td><td colspan="4"></td></tr>
<tr><td colspan="5">共 33 项，其中带*为必填数据，为 21 项。</td></tr>
</table>