分类号 TP311.5 学号 10069042

UDC 密级 公开

工学博士学位论文

基于 SAT 的符号化模型检验技术研究

博士生姓名 王瑞

学 科 专 业 计算机科学与技术

研 究 方 向 计算机软件与理论

指 导 教 师 毛晓光 教授

国防科学技术大学研究生院

二〇一四 年 六 月

Research on SAT based Symbolic Model Checking

Candidate: **Wang Rui**

Supervisor: **Professor Mao Xiaoguang**

A dissertation

Submitted in partial fulfillment of the requirements

for the degree of **Doctor of Engineering**

in **Computer Science and Technology**

Graduate School of National University of Defense Technology

Changsha, Hunan, P. R. China

June 9, 2014

独 创 性 声 明

本人声明所呈交的学位论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表和撰写过的研究成果，也不包含为获得国防科学技术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谢意。

学位论文题目：基于 SAT 的符号化模型检验技术研究

学位论文作者签名：王 婷　　日期：2014 年 4 月 22 日

学位论文版权使用授权书

本人完全了解国防科学技术大学有关保留、使用学位论文的规定。本人授权国防科学技术大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；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，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：基于 SAT 的符号化模型检验技术研究

学位论文作者签名：王 婷　　日期：2014 年 4 月 22 日

作者指导教师签名：毛晓光　　日期：2014年 4月22 日

目 录

表 目 录

图 目 录

摘　要

随着计算机技术的不断进步，计算机系统的软硬件设计越来越复杂，过于复杂和庞大的设计必定会带来越来越多的设计缺陷和错误。传统的方法很难检测到系统设计中的所有错误。然而，系统的安全性在航空、航天、国防、医疗等安全攸关的领域却非常重要。因此，用以保证系统正确性的形式化方法亟待出现。

模型检验是上世纪 80 年代提出的一种针对有穷状态系统进行自动化验证的技术。经过 30 多年的发展，模型检验技术已经是计算机辅助验证领域非常重要的技术之一。然而，状态空间爆炸问题始终是制约模型检验技术发展的最大障碍。为此，研究人员提出了很多技术来对付该问题，比如基于 BDD 的符号化模型检验技术、偏序归约技术、抽象精化技术等。近年来，随着 SAT 技术的进展，Biere[1] 等人提出了基于 SAT 的限界模型检验技术。该技术利用 SAT 求解器的能力进一步提高了模型检验技术处理问题的规模。目前，基于 SAT 的符号化模型检验技术已经成为计算机辅助验证领域研究的热点问题之一。

然而，在实际应用中，基于 SAT 的符号化验证技术还存在诸多问题，比如验证效率、验证能力、完全性等。本文围绕基于 SAT 的符号化模型检验技术进行了一系列研究，旨在进一步提高该技术的效率以及扩展该技术处理问题的能力。

本文的主要贡献包括：

1）优化了 LTL 的限界模型检验技术

针对现有的 LTL 的限界模型检验技术，本文提出两种优化的方法。第一种优化技术是一种称为反例集合保持的化简技术，该技术的核心思想就是在进行模型检验之前，首先使用一种轻量级的技术对待验证的 LTL 性质进行化简，使得化简后的性质与原性质关于待验证的模型是等价的。因此，在真正的模型检验中，就可以使用化简后的性质代替原性质进行验证，该技术对于复杂的性质规约具有非常明显的作用。第二种优化技术，是利用 SAT 求解器的增量式求解能力，优化了基于语义的限界模型检验编码，实验表明，新的编码在验证效率上，相对于其他编码有很大的优势。这两种技术可以很自然地结合起来，从而提高 LTL 的限界模型检验效率。

2）提出 ω-正规性质的限界模型检验算法。

由于在工业应用中，许多重要的时序性质无法采用 LTL 表达，因此，针对 ω-正规性质的限界模型检验具有十分重要的理论和实际意义。本文针对两种具有 ω-正规表达能力的时序逻辑 ETL_{l+f} 和 QTL ，分别提出了对应的限界模型检验算

法。针对 ETL_{l+f}，本文提出了基于语义的限界模型检验算法。该技术的核心是构造 ETL_{l+f} 的 tableau，通过扩展 LTL 的 tableau 构造技术，本文提出了 ETL_{l+f} 的 tableau 构造算法。针对 QTL，本文提出了两种限界模型检验算法，分别为基于语法的限界模型检验和基于语义的限界模型检验。基于语法的限界模型检验的核心思想是将 QTL 公式的限界语义编码成 QBF 公式，然后利用 QBF 求解器来验证该问题。基于语义的限界模型检验同样是通过扩展 LTL 的 tableau 构造方法得到 QTL 的 tableau 构造方法，然后将限界模型检验问题转化为受限路径上的公平性路径查找问题。然后，将公平性路径查找问题编码成布尔公式，最后利用 SAT 求解器来验证该问题。

3）提出了基于 SAT 的符号化模型检验的完全性保证的算法。

制约限界模型检验技术应用的一个主要瓶颈在于其不是一种完全的技术，即它不能证明给定的模型是否满足给定的性质。然而，线性时序逻辑的模型检验问题最终都可以转化为公平性路径查找问题，而公平性路径查找问题可以转化为若干个可达性问题。因此，开发高效的可达性算法是解决基于 SAT 符号化模型检验完全性保证技术的核心。本文基于目前最好的可达性算法，即属性指导的可达性分析算法（Property Directed Reachability，简称 PDR），提出了两种新的可达性算法。两种算法的核心思想都是试图通过双向逼近可达状态空间，从而加快算法的收敛速度。实验结果表明，本文提出的可达性算法相对于基本的算法有了很大的提高。

4）实现了两个实用的工具。

模型检验技术的最终目的是应用，将理论上的算法转化为实用的工具是模型检验技术非常重要的一部分。本文实现了两个实用工具，分别支持本文提出的各个算法。工具 ENuSMV 是基于开源的符号化模型检验工具 NuSMV 实现的，它支持本文提出的 CEPRE （CounterExample Preserving REduction，简称 CEPRE）算法、增量式的基于语义的 LTL 限界模型检验算法、基于语义的 ETL_{l+f} 的限界模型检验算法。工具 Reach 是一个全新的基于 SAT 的符号化模型检验工具，它不仅支持本文提出的交叠的 PDR 算法以及并行的 PDR 算法，还支持其他著名的可达性算法，包括限界模型检验算法、k 步归纳算法、基于插值的符号化模型检验算法以及基本的 PDR 算法。

关键词: **模型检验**; SAT **求解器**; **限界模型检验**; LTL; ETL_{l+f}; QTL; PDR

ABSTRACT

With the advances in computer technology, the designs of hardware and software become more and more complexity, and the complex designs tend to contain more and more design flaws and errors. The traditional methods can not find all the design errors in the system. However, the safety of the systems are very important, especially for safety-critical applications such as aerospace, defense and health care. Therefore, verification technologies are urgently required in such fields.

Model checking is a technique for automatically verifying correctness properties of finite state systems which is proposed in 1980s. After 30 years of development, model checking has been one of the most important technique in the field of computer aided verification. However, this technique it limited by the state explosion problem ——the number of states of the system is exponential in the number of state variables. Researchers in this field developed different algorithms to cope with this problem; e.g., BDD based symbolic model checking, partial order reduction and CEGAR. In recent years, with the advance of SAT technologies, Biere et al. first proposed the Bounded Model Checking technique. It has attracted extensive interest from both academia and industry because of its advantage in finding bugs compared to BDD based model checking. At present, SAT based model checking is still a hot topic in computer aided verification field.

However, in practice, there are still many problems with SAT based model checking, such as efficiency, ability, completeness. In this thesis, we focus on how to improve the efficiency, ability and completeness of SAT based model checking. The contributions of this thesis are listed as follows.

1) We optimize the LTL bounded model checking technique to improve the efficiency of bounded model checking for LTL.

It consists of two optimization techniques. We name the first technique as ”Counterexample-Presering Reduction (CePRe. for short)”. The main idea of CePRe is to reduce the input LTL formula into an ”equivalent” formula which is an easier-to-handle one before model checking with a light-weight technique. Hence, the reduced formula will replace the original formula in model checking. The technique is very efficient in model checking, especially for complex formulae. We propose a new encoding technique in the the second optimization technique which makes use of the incremental SAT technologies. These two

optimization techniques can be used together to improve the LTL bounded model checking. Experimental results show that, the new encoding is very efficient compared to other encodings.

2) We propose bounded model checking techniques for ω-regular property.

LTL is not powerful enough to express all the ω-regular properties used in industrial verification. Hence, it is very important to development bounded model checking for ω-regular properties. In this thesis, we propose three encodings for bounded model checking two temporal logics which can express all ω-regular properties. We propose the semantic based encoding for bounded model checking ETL_{l+f} . The key part of the technique is to construct tableau for ETL_{l+f} . We extend the LTL tableau construction technique to construct tableau for ETL_{l+f} . We propose two encodings for bounded model checking QTL. The first encoding is based on the syntax of QTL which we name as syntax based bounded model checking. The key part of syntax based bounded model checking is to encode the bounded model checking problem of QTL into an QBF formula which can be solved by an QBF solver. The other encoding technique is also an semantic based encoding. We achieve the tableau of QTL by extending the LTL tableau construction technique. The semantic encoding convert the model checking problem into an constraint fair cycle detection problem which can be encoding into an boolean formula that can be solved by an SAT solver.

3) We present a complete SAT based model checking technique.

Bounded model checking technique is not a complete technique, it can not prove properties. However, the model checking problem of linear temporal logics can be converted into a fair cycle detection problem, and the fair cycle detection problem can be handled by solving some reachability problems. Thus, the key part of a complete SAT based model checking algorithm for linear temporal logics is efficient reachability algorithms. In this thesis, we propose two efficient reachability algorithms. Both of them are based on Property Directed Reachability (PDR, for short). The intuitive idea behind these two algorithms is to approximate the reachable state space by bidirectional PDR. Experimental results has demonstrated its efficiency.

4) We have implemented two available tools.

We implement the CEPRE technique, the incremental semantic based encoding for LTL and ETL_{l+f} bounded model checking algorithm in ENuSMV which is an extension of NuSMV. An reachability analysis tool, namely Reach is also designed and implement-

ed. It also supports interleaving PDR algorithm and parallel PDR algorithm presents in this thesis. Except that, the tool also supports many other algorithms, such as BMC, k-induction based model checking, interpolation based model checking and basic PDR algorithm.

Key Words: Model Checking; SAT solver; Bounded Model Checking; LTL; ETL_{l+f}; QTL; PDR

第一章　绪论

1.1　研究背景

随着计算机技术的不断进步，计算机软硬件的规模不断增大，这使得计算机软硬件的设计越来越复杂，过于复杂的设计必定会带来越来越多的设计缺陷和错误。然而，现代社会对计算机的依赖越来越强，尤其是在安全攸关系统领域，软硬件设计和实现的正确性越来越重要。用传统的方法和技术很难检测到系统中存在的所有错误，并且验证代价很高。在安全攸关领域，有些问题一旦出现，可能会导致严重的后果。

由软硬件设计和实现方面的错误而导致的重大事故有很多。1985 年至 1987 年间，用于治疗癌症的 Therac-25 辐射治疗机由于软件设计的瑕疵，剂量设定超过安全范围，导致 6 位医疗患者死亡或者受到严重辐射灼伤。1990 年，AT&T 网络瘫痪导致 7500 万用户受影响，直接经济损失超过 100 亿美元，事后发现是由于一个细小的软件错误导致。1994 年，Intel 公司的奔腾处理器的浮点运算部件运算时出错，使得 Intel 公司付出了近 5 亿美元的代价。1996 年，欧洲航天局研制的 Ariane 5 火箭在发射升空后不到 37 秒爆炸，调查发现，事故源于一个 64 位浮点数转换异常。为了避免事故的发生，必须对计算机软硬件设计和实现进行充分的验证。

传统的用于保证系统正确性的方法主要包括模拟和测试。在系统设计的早期阶段，一般通过模拟技术来保证系统的正确性。模拟的基本原理是通过模拟系统实现后的运行结果来观察早期的设计是否存在缺陷。在系统设计的早期阶段，模拟技术通常能发现大量明显的错误。测试技术一般是指在已经存在的实际系统中，通过输入测试用例来观察输出的结果是否与期望的输出结果相符。如若输出结果与期望结果不相同，则说明实际系统中存在错误，这就需要重新设计和开发。这也直接导致了成本的增加。但是无论模拟还是测试，都严重的依赖于测试人员输入的数据，对于过于复杂和庞大的系统，试图通过 100% 的数据覆盖率来进行模拟和测试几乎是不可能的；一旦实验数据覆盖率不够，那些隐蔽的错误就很难被发现[2]。

形式化验证是通过数学和逻辑的方法来确保一个设计或系统符合一些准确表达的性质[2]。形式化验证技术是一种完备的技术，它可以弥补模拟和测试在验证上的不足，可以保证系统正确性和安全性。形式化验证技术的另一个优点是它适用于系统研制周期的早期阶段[2]。市场的激烈竞争要求各个 IT 厂商不断缩短新产品的研制周期。为提高研制的速度，在研制周期的早期进行分析和验证，可以大

大降低返工的成本，从而提高生产效率[2]。因此，研究形式化验证的技术和方法就有很重要的意义。

实际上，形式化分析和验证一直以来也都是计算机科学领域的重要研究内容，并且产生了一大类相关的技术，大体上来讲主要包括以下几大类，下面对几种主要技术做简单的介绍。

- **定理证明**技术是一种特殊的自动推理技术，它将待验证的模型或者程序抽象为逻辑公式，然后使用自动的逻辑推理技术来验证是否正确[3]。定理证明技术能在一定的范围内证明程序或者系统的正确性，但自动化程度比较低，证明的过程需要有经验的用户提供大量的公理、前提条件以及其他的系统信息。基于定理证明的验证方法适用于一般的模型，其难点在于需要用户输入推理策略以及大量的定理，并且定理证明技术很难适用于大规模的系统。

- **静态分析**是一种通过不真正执行程序而到达分析程序性质目的技术。通常，静态分析技术都是针对程序源代码进行的。一般来讲，编译器中涉及的很多技术都属于静态分析技术，包括词法语法分析技术、程序控制流分析技术、程序数据流分析技术等。但是，一般情况下，程序的状态空间都是无穷的，因此，直接对程序进行分析基本是不可行的，这时就要对源程序进行抽象。为了保证分析的可靠性，静态分析中使用的抽象技术一般要求是程序状态空间的上抽象。但是使用上抽象又会给静态分析技术带来误报的问题。事实上，很多静态分析技术就是困扰于这一问题。如何在可接受的代价范围内提高分析精度并消除误报，是目前静态分析领域面临的主要挑战。

- **抽象解释**是由 Patrick Cousot 和 Radhia Cousot 于上世纪 70 年代提出的一种基于程序语义计算程序状态空间技术[4]。它可以看作是一种的特殊的静态分析的方法。抽象解释通过分析程序的部分执行而不是全部执行来计算整个程序的可达状态空间的上逼近。因此，抽象解释技术是一种可靠的技术，它可以用来证明程序的正确性，但是它不是一种完备的技术，即抽象解释可能会产生大量的误报。而一旦产生误报，抽象解释技术需要更具体的模型来计算程序语义，因此抽象解释需要对程序进行精化，然后对精化后的程序做进一步的分析。

- **符号执行**是一种经典的程序分析技术，它以符号值作为程序的输入，符号化的执行程序，同时程序的输出也为变量的表达式。符号执行技术使用约束求解器来判定程序的路径是否可行。符号执行技术能够遍历程序的路径空间，检查程序是否满足特定的性质。本质上讲，符号执行技术是一种显式的基于

路径的限界模型检验。符号执行技术受限于路径爆炸问题和约束求解器的能力。对于规模稍微大的程序，其很难给出结果，并且符号执行技术本身是一种非完全的技术。

- **谓词抽象**是一种将无穷状态系统抽象成有穷状态系统的技术，其本质是抽象解释理论的一种实例化。谓词抽象技术通过分析待验证的程序，从而抽象出一组有穷数量的谓词，而通过这些谓词，我们可以抽象出程序的有穷状态机模型，从而可以使用有穷状态系统上的模型检验算法对其进行验证。谓词抽象一般都会和反例精化技术相结合，即 CEGAR（CounterExmaple Guided Abstraction Refinement）技术。基于谓词抽象的技术是目前模型检验与抽象解释相结合应对源代码程序验证的重要手段。该技术最大的瓶颈是虚假反例的处理和模型的精化。

- **模型检验**技术将待验证的对象抽象成有穷或者无穷状态的迁移系统，对迁移系统的状态空间进行遍历来验证系统是否满足给定的性质。模型检验技术具有非常高的自动化程度，并且当程序违反给定的性质时，模型检验技术可以给出对应的反例。反例可以作为诊断信息，从而帮助设计者修正系统的错误。自上世纪 80 年代以来，模型检验技术得到了非常广泛的应用，已经被证明是一种行之有效的自动化的软硬件验证技术。**该技术也是本课题研究的对象**。

1.2 模型检验技术

模型检验的输入有两个：一个是待验证的模型 M, 另一个是待验证的规约 φ。需要验证的目标为：在时序逻辑的语义下，是否有 $M \models \varphi$ 成立[5]。直观的讲，就是要判断 M 的每条执行轨迹（针对线性时序逻辑而言）或者每个可能的展开树（针对分支时序逻辑而言）是否都在初始时刻满足 φ。模型检验技术的框架如图 1.1所示。

1.2.1 模型描述

模型检验技术中的“模型”，一般是指软硬件系统的设计模型，根据不同的应用环境，需要使用不同的模型描述语言来描述待验证的系统[6]。一般情况下，模型描述语言可以分为以下几类：面向硬件的模型描述语言，面向软件的模型描述语言以及抽象系统描述语言[6]。

- **面向硬件的模型描述语言：**这类语言一般是指某种硬件描述语言，如 Verilog、SystemC 等。针对 Verilog 语言进行模型检验的工具包括：Candence

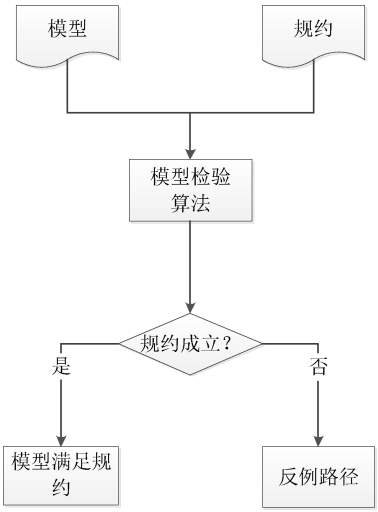


图 1.1　模型检验技术框架

公司和伯克利（Berkeley）大学的 Cadence SMV 模型检验器[7]、IBM 的 Rulebase[8]、Intel 的 Forecast[9] 等。

- **面向软件的模型描述语言：**针对软件系统的模型检验技术就是针对源代码的一种形式化验证技术，其模型描述语言一般是某种高级程序语言或者其子集。比如针对 C 语言进行模型检验的工具包括：卡耐基梅隆（CMU）大学的 CBMC[10] 和 MAGIC[11]、微软研究中心的 SLAM[12]、贝尔实验室的 Verisoft[13]、伯克利大学的 Blast[14]、斯坦福（Stanford）大学的 Murphi[15] 和 CMC[16]，以及针对 Java 语言的工具，NASA 的 JPF[17] 等。

- **抽象系统描述语言：**这类语言一般是为了学术研究的方便自定义的一些模型描述语言。不同的模型检验器有不同的模型描述语言。比较流行的包括卡耐基梅隆大学 SMV 支持的 SMV[18]，后来的很多工具如 NuSMV[19]，Cadence SMV[7] 都支持该语言。贝尔实验室的 Spin 所支持的 Promela[20]、以及卡罗拉多（Colorado）大学的 IIMC 工具支持的 AIG[21] 等。

1.2.2　规约描述

模型检验中，用作规约描述的语言一般是某种时序逻辑。时序逻辑于上世纪 70 年代提出，主要用来描述系统的时序行为，即系统中“事件”发生的顺序。从广义上讲可以分为线性时序逻辑和分支时序逻辑两大类。

- **线性时序逻辑：**所谓线性时序逻辑，即语义定义在线性结构上的时序逻辑，通常可以用来描述系统中有穷或无穷路径。该类时序逻辑的典型代表有 LTL[22]、ETL[23]、PSL[24]、QTL[25]、线性 μ-演算[26] 等。

- **分支时序逻辑：**所谓分支时序逻辑，即语义定义在分支结构上的时序逻辑，该类逻辑一般用来描述系统以某一个状态为根的展开树的所有行为。代表性的分支时序逻辑有 CTL[27]、CTL*[28]、模态 μ-演算[29] 等。

在实际应用中，线性时序逻辑和分支时序逻辑各有优缺点，本文的第二章将详细介绍各种时序逻辑的语法、语义以及它们的表达能力。

1.2.3 模型检验算法

模型检验算法是模型检验技术的核心，下面按照技术发展的时间简单介绍。

1.2.3.1 显式的模型检验技术

上世纪 80 年代初，Clarke 和 Emerson 首先给出了分支时序逻辑 CTL 的显式模型检验算法[30]，该方法显式的存储和遍历可达状态空间，故称为显式的模型检验技术。该技术的核心是将 CTL 嵌入到模态 μ-演算中，而模态 μ-演算的语义解释是基于状态集的，这样一来，对于任意的 CTL 公式，其语义都可以映射成为一个状态集合。因此，对于任意的 CTL 公式，都可以迭代的计算其对应的满足集。后来，Wolper、Vardi、Sistla 等人[31] 又提出了基于自动机的线性时序逻辑的模型检验框架，该技术最早是针对 LTL 的模型检验的。给定模型 M 以及 LTL 公式 φ，基于自动机的 LTL 显式模型检验算法首先将模型 M 转换成等价的 Büchi 自动机 $\mathcal{A}_M$，然后构造规约性质 $\neg\varphi$ 对应的 Büchi 自动机 $\mathcal{A}_{\neg\varphi}$，使得自动机 $\mathcal{A}_{\neg\varphi}$ 恰好接收满足 $\neg\varphi$ 的线性结构。接下来，构造 $\mathcal{A}_M$ 与 $\mathcal{A}_{\neg\varphi}$ 的乘积 $\mathcal{A}_M \otimes \mathcal{A}_{\neg\varphi}$。最后对自动机 $\mathcal{A}_M \otimes \mathcal{A}_{\neg\varphi}$ 接收的语言进行判空，若为空，则性质正确；否则，$\mathcal{A}_M$ 中至少存在一条违反 φ 的路径。在该过程中，由 M 到 Büchi 自动机 $\mathcal{A}_M$ 的复杂度是线性的；而构造自动机 $\mathcal{A}_{\neg\varphi}$ 的代价却为 $2^{\mathcal{O}(|\varphi|)}$，其中 $|\varphi|$ 为 φ 的公式长度。因此，基于自动机的 LTL 的模型检验的复杂度为 $\mathcal{O}(|M| \times 2^{|\varphi|})$。

基于自动机的显式模型检验算法框架也可以扩展到其他线性时序逻辑中，其核心是构造公式 φ 对应的自动机。该技术主要适用于软件和协议的验证。早期的显式模型检验技术是仅能处理几千个状态的实验方法，根本无法满足实际应用需求。随着新的优化算法和优化技术的提出，目前显式模型检验技术已经能处理一定规模的系统，并在工业界中发挥了很大的作用。因为显式模型检验技术是显式的存储状态和迁移信息，所以其最大的瓶颈仍在于其处理问题的规模十分有限。

1.2.3.2 基于 BDD 的符号化模型检验技术

BDD（Binary Decision Diagram，即二叉决策图）最初是由 Bryant[32] 引入的用以表示布尔公式的数据结构。Bryant 在文献 [32] 中发现，使用 BDD 表示布尔公式可以很高效的实现各种布尔运算。为了提高模型检验技术处理问题的规模，

McMillan 首先尝试将 BDD 引入到模型检验技术中来，他将布尔公式编码成 BBD 数据结构，从而在 BDD 上实现状态空间搜索。实验表明 BDD 的引入极大的提高了模型检验处理问题的规模[33]。在模型检验技术中，最先采用基于 BDD 的是针对 CTL 的模型检验算法。同时，CTL 的基于 BDD 的符号化模型检验也是其他类型时序逻辑符号化模型检验的基础。

到 90 年代中期，Clarke 等人又给出了基于 BDD 的 LTL 的符号化模型检验算法[34]。其核心思想是给定模型 M 以及 LTL 公式 φ，首先构造公式 $\neg\varphi$ 的 tableau $\mathcal{T}_{\neg\varphi}$，使得 $\mathcal{T}_{\neg\varphi}$ 恰好接收满足 $\neg\varphi$ 的所有路径，即违反公式 φ 的所有路径。最后，求模型 M 和 $\mathcal{T}_{\neg\varphi}$ 的符号化乘积 $M \otimes \mathcal{T}_{\neg\varphi}$，然后验证该乘积模型中是否存在一条公平路径。

随后，研究人员又提出了各种针对基于 BDD 的符号化模型检验算法的优化技术，包括：偏序归约技术、抽象技术、组合推理技术、对称归约技术、COI（Cone Of Influence）等技术。配合各种优化技术，基于 BDD 的符号化模型检验算法技术大大的提高了模型检验处理问题的规模，随后，针对各种时序逻辑的符号化模型检验算法相继被提出。文献 [35] 又给出了 CTL* 的符号化模型检验算法，在文献 [36] 中，苏开乐等人实现了该算法并给出了其工具 MCTK。文献 [37] 基于 LTL 的 tableau 构造技术，给出了扩展时序逻辑 ETL 的符号化模型检验算法。文献 [38] 给出了 PSL 的变种 APSL 的符号化模型检验算法并给出了其工具实现。

1.2.3.3 基于 SAT 的符号化模型检验技术

基于 BDD 的符号化模型检验在学术界和工业界取得了巨大的成功，其处理问题的规模提高了很多。但是，基于 BDD 的符号化模型检验技术仍旧有很多不足：

- 作为布尔公式的一种标准表示，BDD 的引入虽然在一定程度上缓解了状态空间爆炸问题，但是，它仍然无法避免状态空间爆炸。当模型中的状态变量达到上百个时，所需的存储空间就会很庞大。
- 对于给定公式，其对应的 BDD 的大小与公式中涉及的变量排序有非常大关系，而选择最好的变量排序的复杂度与模型检验问题的复杂度是一样的。正如 Bryant 所指出，对于有些布尔公式，无论变量如何排序，BDD 的存储都是指数级的，根本没有优化的空间[32]。

进入上世纪 90 年代，随着 SAT 技术在规划问题（Planning）和人工智能领域的成功应用，SAT 技术取得了飞速的发展。Biere 等人[39] 率先将 SAT 技术引入到了模型检验领域，文献 [39] 称其提出的技术为限界模型检验 (Bouned Model Checking，简称 BMC)。BMC 技术最初是针对线性时序逻辑 LTL 的，它的基本思想就是把受限的 LTL 的模型检验问题编码成布尔公式求解问题。给定模型 M 和

LTL 性质 φ，BMC 试图回答这样的一个问题：模型 M 中是否存在一条 k 步的反例路径? 如果存在，输出这条反例路径；否则，k 递增 1。这个问题被编码成布尔公式，利用 SAT 求解器回答其是否可满足。BMC 技术的本质是由于对任意 LTL 公式都可以找到一条使用有穷状态表示的反例路径。相对于基于 BDD 的符号化模型检验技术，BMC 技术主要有以下优势：

- 尽管 SAT 求解算法也是以布尔公式为输入的，但是它使用更加灵活和紧致的 CNF 表示，因此，它不存在存储空间爆炸问题。
- SAT 求解过程与变量排序无关。针对不同问题，SAT 求解器会动态的产生不同的变量排序从而加快 SAT 算法的求解速度。
- 可以利用现代 SAT 求解器的最新技术。

BMC 技术的提出进一步提高了模型检验技术处理问题的规模，尤其是在错误查找方面，其相对 BDD 具有很大的优势。BMC 技术一经提出，就引起了学术界和工业界的广泛关注，它为模型检验技术的发展提供了一种新的思路，成为了近年来，模型检验领域的研究热点。

1.2.4 典型工具

模型检验技术经过 30 年的发展，无论是显式的模型检验，还是符号化模型检验技术，都涌现了许多优秀的工具，这些工具实现了各种算法和优化技术，根据其面向的领域和对象不同，会有一些不同的处理技术。

1.2.4.1 SPIN

SPIN（Simple Promela Interpreter）是贝尔实验室的 Holzmann 开发的开源的模型检验工具[40]。SPIN 最早的版本可以追溯到 1980 年的 Pan 工具[41]，当时主要使用进程代数理论作为其技术基础。到 1983 年，其又改名为 Trace[42]，这时的技术主要是基于自动机理论的。1989 年其推出第一个版本的基于自动机模型检验工具，也正式更名为 SPIN[43]。随后的二十多年，SPIN 逐渐采用了深度优先搜索算法、偏序归约、极小化自动机、面向性质的切片等优化技术，使得 SPIN 真正走向了工业应用。

SPIN 主要实现了基于自动机的 On-the-fly 的显式的 LTL 模型检验算法。它的输入语言为 Promela，该语言能较好的描述网络协议以及软件设计，因此 SPIN 也更方便验证软件设计和协议的安全性。

目前，SPIN 仍是模型检验领域的重要工具，也是现在使用最广泛的工具之一，并有专门的组织负责维护，感兴趣的读者可以参考 [20]。

1.2.4.2 Murphi

Murphi 最初是由斯坦福大学 Dill 教授领导的课题组开发的[44]，现在由犹他 (Utah) 大学的 Gopalakrishnan 教授所在的课题组维护。它是一个基于显式模型检

验技术的工具。它定义了自己的输入语言 murphi，其是一种描述异步并发系统的语言，具有很多高级语言的特征，如允许用户自定义数据类型、过程和描述参数等。其实现的显式的模型检验算法，既支持对状态空间深度优先的搜索又支持宽度优先的搜索。为了对付状态空间爆炸问题，Murphi 中使用的主要优化技术包含对称归约技术[45]、可逆向规则消减技术[46] 以及重复构造子技术[47]。

Murphi 已经成功的应用在很多验证任务和项目，其中包括斯坦福的 DASH 和 FLASH 多核处理器中的 cache 一致性协议、Sun 公司的 S3.mp 多核处理器的 cache 一致性协议等。目前 Murphi 仍旧被广泛的使用，尤其是在多核处理器的协议一致性验证方面，其有很大优势。目前最新的 Murphi 已经有了并行化版本并且支持带概率的模型检验算法。感兴趣的读者可以参考 [15]。

1.2.4.3 SMV

SMV(Symbolic Model Verifier) 是卡耐基梅隆大学在 1992 年开发的一个开源的符号化模型检验工具[18]，它第一次使用了基于 BDD 的符号化模型检验算法，在模型检验领域具有里程碑的意义。该工具实现了基于 BDD 的 CTL 的符号化模型检验算法。它使用 BDD 来表示模型的初始状态集合和迁移关系，将待验证的 CTL 公式嵌入到模态 μ-演算中，通过计算不动点验证性质。

基于 BDD 的使用极大的节省了存储空间，使得可搜索的状态空间大大增加，相比同样是卡耐基梅隆大学开发的显式的模型检验工具 EMC(Explicit Model Checker)[48]，SMV 在性能、效率方面提高了很多。

SMV 的输入语言为自定义的模型和规约描述语言 SMV 格式，使用该语言用户可以自定义迁移系统，该语法结构支持模块化系统描述，根据系统需要，可以将整个系统分为几个子系统，每个子系统完全相应的工作，SMV 工具会自动的对系统进行平坦化处理。SMV 支持完整的 CTL 的语法描述，用户可以用 CTL 描述系统中待验证的性质，然后 SMV 将自动化给出验证结果。

SMV 的出现使得符号化模型检验技术得到了广泛的应用，例如，1992 年，Clarke 和他的学生使用 SMV 验证了 IEEE Future+ cache 一致性协议[49]，这个协议在 1988 年提出，之前利用非形式化的技术没有发现任何错误，而使用 SMV 发现了很多之前没有发现的错误。这是形式化方法第一次发现 IEEE 标准中的错误。

SMV 目前最新的版本为 2.5 版，从 2002 年之后，由于 NuSMV（后面会讲到）完全兼容了 SMV，因此，SMV 系统不再更新，不过用户还可以进行下载使用。

1.2.4.4 NuSMV

NuSMV(New Symbolic Model Verifier) 是由 FBK-IRST 和卡耐基梅隆大学联合开发的一个模型检验工具[19]。它对 SMV 进行了重新设计、实现和扩充。相对于 SMV，其主要有以下不同：

- 功能上，除了可以验证用 CTL 描述的规约，其还支持 LTL 描述的规约。它不仅实现了基于 BDD 的符号化模型检验算法，还集成了限界模型检验技术（BMC）。
- 相对于 SMV，NuSMV 定义了一个良好的系统架构，其实现也利于开发和添加新的模块，更容易对其进行扩充以实现新的功能；比如，本文就在 NuSMV 的基础上实现了一个 ETL 的限界模型检验模块。
- NuSMV 的各种文档更加齐全，其源代码对应的注释也更多，比 SMV 更加容易阅读和修改。
- 它还提供了一个友好的图形化交互界面，方便用户使用。

本文提出的一些算法是基于 NuSMV 实现的，在后续的章节中，我们会给出每个模块的具体描述，在这里不再详细介绍。

1.2.4.5 CBMC

CBMC(C Bounded Model Checker) 是卡耐基梅隆大学开发的一个面向 ANSI-C 的限界模型检验工具[10]，目前最新的 CBMC 还支持 C++，SystemC 语言。CBMC 可以验证 C 程序中数组越界、指针安全性、异常以及用户自定义的断言等性质。它还可以检查 C 语言与其他语言的之间的等价一致性，如 Verilog。

CBMC 的主要思想是通过把程序和待验证的性质展开成一个逻辑公式（对于循环则需要限定一个展开次数），然后利用约束求解器来求解公式是否可满足。CBMC 的工作原理如下：首先，它先将给定的源程序转换成控制流图（Control Flow Graph），对于待检测的性质 P（一般是程序点），显式地构造一个公式 φ，使得性质 P 是正确的（一般指程序点是可达的）当且仅当公式 φ 是可满足的。最后，利用约束求解器判断公式 φ 是否可满足。

最初 CBMC 是使用 SAT 求解器作为其后端引擎的，目前的 CBMC 不仅支持 SAT 求解器，还支持目前最先进的 SMT 求解器，比如微软的 Z3[50]，这很大程度上提高了 CBMC 的验证规模。

发展到现在，CBMC 版本已经是 4.7，并且 Daniel Kroening 一直在维护这个工具，其最新的面向内存模型的优化技术进一步提高了 CBMC 验证程序代码的规模[51]。

1.2.4.6 EBMC

EBMC(Embedded system Bounded Model Checker) 是一个非开源的面向硬件设计的模型检验器[52]，该工具由 Kroeninig 和 Purandare 开发并维护。它实现了限界模型检验技术，但提供了几种保证限界模型检验完全性的技术，因此，它不仅能发现硬件设计中错误，还能证明硬件设计模型是否满足规约要求，它还支持多种输入格式，包括 Netlists、Verilog 以及 SMV。

EBMC 的主要特征包括:

- 支持限界模型检验算法。
- 使用静态分析技术给出一个逼近的完备阈值，从而保证 BMC 的完全性。
- 支持基于反例制导的抽象精化技术。
- 实现了 k 步归纳算法，从而保证 BMC 技术的完全性。
- 支持基于插值的符号化模型检验算法。

目前 EBMC 的最新版本为 4.1，它给出了 Linux、Windows 以及 Mac OS 平台的二进制可运行程序。感兴趣的读者可以参考 [53]。

1.2.4.7 VIS

VIS（Verification Interacting with Synthesis）是由伯克利大学、卡罗拉多大学和得克萨斯（Texas）大学联合开发的一个面向有穷状态系统的验证、综合以及模拟的集成化工具[54]。VIS 不仅可以支持传统的模拟技术，它还支持综合以及各种验证算法。VIS 主要有以下功能:

- 逻辑电路的模拟测试。
- 组合电路和时序电路在设计和实现层面上的形式化验证。
- 逻辑综合。

VIS 在形式化验证技术上主要实现了基于 BDD 的符号化模型检验技术，它支持 CTL 和 LTL 的符号化模型检验算法，它还支持 LTL 的 BMC 以及基于 SAT 的完全性保证技术。

VIS 的输入为 Verilog 语言，为用户提供了很方便的验证环境，它还有一个交互式的图形化前端，方便用户使用。该工具是开源的，目前已经开发到 2.4 版本，感兴趣的读者可以参考 [55]。

1.2.4.8 ABC

ABC 是伯克利大学逻辑综合与验证组开发的一个面向时序逻辑电路的综合与验证工具[56]。ABC 组合实现了基于 AIG（And-Inverter-Graphs）图的逻辑优化技术以及各种面向时序逻辑的综合与验证算法。ABC 的主要特征如下:

- 支持多种基本数据结构用来描述时序电路的结构信息，包括 BDD、SOPs、AIG 等。
- 支持多种输入文件格式，如 BLIF、PLA、BENCH、EDIF 以及 Verilog 的子集。
- 它用启发式的算法来检查输入的电路使用哪种数据结构表示。
- 实现了基于 BDD 和基于 SAT 的等价性检查算法，以及限界的时序等价性检查。
- 各种基于 BDD 和基于 SAT 的符号化模型检验算法。

ABC 模块化的实现了各项功能，非常方便开发人员扩展和实现新的算法，并且是一个开源工具。现在它仍由伯克利大学的综合和验证研究组负责维护和更新，感兴趣的读者可以参阅 [57]。

1.2.4.9 IImc

IImc（Incremental Inductive Model Checker）是卡罗拉多大学的 Zyad Hassan 等人在 2012 年发布的一个增量式的基于归纳技术的模型检验器[21]。它最初仅是实现了属性制导的可达性算法，随着其不停的开发，它目前了支持了多种符号化的模型检验算法，其主要特征如下：

- 目前，其输入格式仅支持 AIG，但是使用自动转化工具，可以很方便的将其他常用的建模语言转换成 AIG 格式。
- IImc 现在实现了多种符号化模型检验算法，主要包括：基于 BDD 的前向和后向可达性算法，BMC 算法、属性制导的可达性分析算法、增量式基于归纳的公平路径查找算法、基于 BMC 的公平路径查找算法以及面向 CTL 的增量式基于归纳的验证算法等。
- 采用了多种优化技术，包括切片技术、局部抽象技术、COI 技术等。
- 它还可以根据模型结构信息，启发式的选取合适的模型检验算法进行验证，实验效果非常好。

IImc 推出以后，引起了模型检验领域的极大关注，其最新的增量式的基于归纳的模型检验算法取得了非常好的成果，在 2013 年的硬件模型检验比赛（HardWare Model Checking Competetion 2013，简写为 HWMCC2013）中综合排名第一，充分说明了其验证能力。目前，该工具仍在扩展新的功能和算法，感兴趣的读者可以参阅 [21]。

1.2.4.10 其他工具

除了上面介绍的工具之外，还有很多优秀的模型检验工具。包括针对源代码的模型检验工具：BLAST[14]、SATabs[58]、Zing[59]、MAGIC[11]、SLAM[12]、JPF[17]、ESBMC[60]、CMC[16]、CPAChecker[61]、LLBMC[62] 等；针对实时系统的模型检验工具 UPPAAL[63]、KRONOS[64]、Verus[65] 等；针对概率系统的模型检验工具 PRISM[66]、MRMC[67]、ETMCC[68]、IscasMC[69] 等，以及针对混成系统的模型检验工具：HyTech[70]、CheckMate[71]、PHAVer[72] 等。

1.3 研究动机与研究内容

如前所述，模型检验技术经过近 30 年的发展，已经从学术界成功的走向了工业应用，并在实际的工程项目中发挥了很重要的作用。但是受限于模型检验技术固有的复杂度，该技术仍面临着很多困难和挑战。本文的研究工作旨在进一步提

高模型检验技术的效率和能力。本小节将简单的讲述本文工作的研究动机与研究内容。

1.3.1 优化的 LTL 限界模型检验技术

Biere 等人在文献 [39] 中最早提出了限界模型检验的技术，该技术利用 SAT 的求解能力来解决模型检验问题。其核心思想是将模型中所有 k 步展开路径编码成布尔公式，同时将 LTL 公式的非在 k 步之内的语义编码成对应的布尔公式，然后使用 SAT 求解器求解两个布尔公式的合取，如果公式满足，说明模型中存在一条长度不超过 k 的反例路径。否则，说明模型在 k 步之内不存在反例路径。

虽然 BMC 在查找系统错误上有非常明显的进步，但是 BMC 技术非常受限于 SAT 求解器的求解能力，如果 k 过大就会导致 SAT 求解器无法在给定的时空内给出结果。尽管 SAT 求解器对一个公式的求解所需的时空与公式的大小理论上没有直接的关系，但是 BMC 的实验表明公式的大小基本上反映了 SAT 求解的时空开销，随着公式的增大，其求解也会变的越来越难。本质上来说，BMC 问题就是一个基于 SAT 的宽度优先搜索的问题，而 k 决定了 BMC 探索路径的深度。不难理解，k 每一次加 1，BMC 所需探索的新的路径要增加的远远大于 1。因此，随着 k 的增加，BMC 对应的 SAT 求解问题会越来越复杂。

本文针对以上的问题提出了一种优化的基于语义的 LTL 限界模型检验技术，该技术首先会对给定的 LTL 规约进行化简，化简的基本原则就是相对于给定的模型 M 保持反例集合相等，该技术我们称之为反例集合保持的化简技术（CounterExample-Preserving REduction, 简写为 CEPRE）。该技术是一种轻量级的化简技术，其带来的额外开销基本可以忽略不计。对于化简后的公式，我们提出一种增量式的基于语义的 BMC 编码，利用 SAT 求解器的增量式求解的能力来求解新的 BMC 问题，实验表明，本文提出的技术极大的提高了 BMC 处理问题的规模以及 BMC 的验证效率。本文提出的 CEPRE技术是一种通用的技术，其不仅仅适用于 BMC 问题，对于所有的 LTL 验证算法，都可以用 CEPRE 先对公式进行化简。

1.3.2 ω-正规性质的限界模型检验技术

线性时间的时序逻辑在实际应用中因其具有简洁、直观、兼容性好等特点得到了广泛的应用[73]。但是 LTL 并不能够表达全部的 ω-正规性质。比如，Wolper 最早发现的诸如“p 在所有偶数时刻均被满足”等重要的 ω-正规性质无法被 LTL 公式表达。然而，规约语言能否表达全部的 ω-正规性质是至关重要的：比如，该条件是能够采用组合验证技术的一个前提[74]。Boigelot、Legay、Wolper 等人在文 [75] 中，进一步强调了具有 ω-正规表达能力时序逻辑的重要性。

具备完全 ω-正规表达能力的时序逻辑主要分为两类：一类是使用二阶量词或者不动点算子的时序逻辑（如线性 μ-演算[26]）；另外一类是使用无穷多时序连接子构成的时序逻辑（如 PSL[24]）。前者的逻辑中包含有穷多个算子，但其公式往往难以理解——一般认为，当交错深度超过 2 时，μ-演算公式的可读性就会变的非常差；相对而言，虽然后一类逻辑的语法中使用了无穷多的时序连接子，但其公式往往比较直观，易理解。扩展时序逻辑（Extended Temporal Logic，简称 ETL）是后一类逻辑的代表。这种语言最大的特点是支持各类 ω-自动机作为时序连接子（参考文献 [76] [77] [78] [79]）。

2006 年之后，模型检验界开始重视能够完全表达 ω-正规性质的时序逻辑的模型检验算法的研究。研究人员针对多种具有 ω-正规表达能力的时序逻辑，提出了各种符号化模型检验算法，如文献 [80]、[37]、[81]、[82]、[83] 等。

上述符号化模型检验算法均是基于 BDD 的。事实上，很多时候，限界模型检验算法能够处理的模型规模更大，尤其在查找设计中的错误时，限界模型检验算法相对于 BDD 的技术有很大的优势。目前，针对全部 ω-正规性质的限界模型检验工作并不多。为了使 BMC 技术支持 ω-正规性质的验证，本文针对两种具有 ω-正规表达能力的时序逻辑，分别提出了对应 BMC 编码。首先，针对扩展时序逻辑 ETL_{l+f}，本文提出了一种基于语义的 BMC 编码技术，该编码复杂度是与公式的长度成线性关系的。针对 QTL，本文给出了两种限界模型检验技术，第一种是称为基于语法的限界模型检验技术，其核心思想就是将 QTL 受限路径上的语义编码成 QBF 公式，利用 QBF 求解器进行验证。第二种方法称为基于语义的限界模型检验，该方法首先通过扩展 LTL 的 tableau 构造技术，给出了一种 QTL 的 tableau 构造方法，然后基于该 tableau 构造技术，本文给出了一种基于语义的 BMC 编码，该编码与变量个数也是成线性关系的。

1.3.3 限界模型检验的完全性保证技术

基于 SAT 的限界模型检验技术在查找系统错误的时候非常有效，能处理的模型规模相对于基于 BDD 的技术有了很大的提高。但是限界模型检验技术的主要问题在于：它不是一个完全的技术，它无法证明模型是否满足给定的规约性质。这是由于限界模型检验技术的本质是在可达状态空间的子集上的一种验证技术，因此，它仅能给出在受限的状态空间内的满足情况，而不能证明整个状态空间是否满足给定的规约性质。

为了保证 BMC 技术的完全性，研究人员提出了多种技术和方法，大体上可以分为以下几类：

- 通过计算完备阈值的上逼近来保证 BMC 技术的完全性。这类技术的本质就是给定一个上界 k 使得若模型在 k 步之内满足性质，那么模型的整个可达状态空间也满足性质。这类技术的主要问题是计算 k 的代价太高。
- 混合使用 BDD 和 SAT 来保证系统的完全性。这类技术一般都和抽象精化结合起来进行，一般都是先对抽象后的模型做验证，如果存在反例，再用具体模型确定反例是否正确。这类技术的主要问题在于处理虚假反例上代价比较大。
- 通过求解上逼近的系统可达状态空间来保证系统的完全性。这类技术的本质是通过迭代计算包含系统可达状态空间的不动点来验证系统是否满足给定的规约。目前这类技术在实际应用中是最好的一类方法。

为了保证限界模型检验技术的完全性，本文提出了两个算法。基于属性制导的可达性算法（Property Directed Reachability，简写为 PDR），本文给出了一个交叠的双向 PDR 算法（Interleaving Bidirectional PDR，简写为 IBPDR）和一个并行的双向 PDR 算法（Parallel Bidirectional PDR，简写为 PBPDR），这两个算法的核心思想都是从初始状态集合和“坏”状态集合分别执行 PDR 算法，同时它们互相利用已经探测到状态空间的信息从而加快算法的收敛速度。不同的是，一种是顺序交叠的执行，另一种是并行的执行。

1.3.4 工具实现

模型检验技术的最终目的在于应用，即：能够根据算法设计出对应计算机软硬件进行辅助验证的自动化工具。一方面，它是将具体理论技术转化为实际应用的体现；另一方面，它也是对理论技术可行性以及效率的实际检验。

本文提出的大部分算法都有具体的实现工具，首先，基于开源的符号化模型检验工具 NuSMV，我们实现了 CePRe算法、增量式的基于语义的 LTL 限界模型检验算法、ETL_{l+f} 的限界模型检验算法。该工具已经公布在互联网上，目前已经吸引了来自美国、俄罗斯、印度、法国等 10 余个国家的学者和工程师的访问。

本文还给出了 IBPDR 算法以及 PBPDR 算法的实现工具 Reach，为了与其他流行工具输入上保持一致，这些工具的实现是基于 AIG 输入格式的。使用 AIG 格式也方便与相关工具进行比较。对于 AIG 格式而言，它也能很方便的与 NuSMV 的格式 SMV 进行互相转换。因此，对于任意限界模型检验问题，我们仅需要将该问题利用 SMV2AIG 工具将 SMV 格式转换成 AIG 格式，然后利用工具 Reach 就可以进行完全性验证了。

1.4 相关工作

本节主要介绍与本文相关的研究工作。主要包括：限界模型检验中的编码技术、ω-正规性质的符号化模型检验技术、限界模型检验的完全性保证技术等。

1.4.1 限界模型检验技术

尽管 BMC 在实际应用中取得了很好的效果，但还是无法满足大规模系统的验证需求，尤其是在完全性验证上，最初的 BMC 算法给不出正确性证明。提高 BMC 技术有很多途径，最重要的有两种方法：一是提供更好的编码方法；二是利用最新的 SAT 求解技术。所谓好的编码方法就是要使得同等问题规模下变量更少，公式规模更小以及更方便 SAT 求解器进行求解。第二种技术就是与 SAT 技术最新的进展相结合，比如使用增量式求解技术。

最初的 BMC 编码与界值 k 是成平方关系的，Cimatti 等人通过分析 BMC 的原始编码，针对原始编码中不同的时序算子提出了不同的优化技术[84]，尽管他们提出的编码相对于原始的 BMC 编码有了很大的提高，但是该编码方法与界值 k 仍是成平方关系的。Frisch 等人提出了一种基于不动点的编码技术[85]，他们利用 LTL 的分离范式进行编码，在一定程度上提高了 BMC 技术的效率和验证规模。但是该编码方法在最坏的情况下与界值 k 仍是成平方关系的，而不是线性的。Clarke 等人提出了一种基于语义的编码方式[86]，该编码方式与基于 BDD 的 LTL 模型检验算法类似。给定模型 M 以及性质 φ，他们先构造公式 $\neg\varphi$ 的 tableau $\mathcal{T}_{\neg\varphi}$，然后求出模型 M 与 $\mathcal{T}_{\neg\varphi}$ 的乘积 $M \otimes \mathcal{T}_{\neg\varphi}$，最后利用 SAT 求解器判断 $M \otimes \mathcal{T}_{\neg\varphi}$ 在 k 步之内是否存在公平路径。基于语义的编码技术有很多优势，它不是基于 LTL 的语法展开的，而是将任意的 LTL 公式转换成了公平路径判断问题。而对于公平路径查找问题的编码要比基于语法的编码简单，它的编码相对于界值 k 是成线性关系。但是，文献 [86] 中的编码并没有利用增量式的求解技术。Latvala 等人利用 LTL 与 CTL 在循环路径上的语义相同的性质提出了一种基于不动点的编码[87]。该编码方式与公式的长度和界值都是成线性关系的。随后，他们又将这种编码方式实现到了带过去算子的 LTL 上[88]。

1.4.2 ω-正规性质的符号化模型检验技术

目前工业界主要应用的规约语言是能完全表达 ω-正规性质的规约语言，例如 PSL（已经是 IEEE 的标准）。因此，ω-正规性质的验证是模型检验领域关注的热点问题之一。针对 ω-正规性质的验证的研究目前主要集中在基于 BDD 和基于 SAT 的符号化模型检验技术。

Pnueli 等人给出了一种基于 tester 的 PSL 符号化模型检验算法[80]。刘万伟、王戟、王昭飞等人扩展了 LTL 的 tableau 方法，给出了基于 BDD 的 ETL_f 以及 PSL 某个变种（称为 APSL）的符号化模型检验算法，并在 NuSMV 的基础上实现了相应的检验工具 ENuSMV（见文 [37] 和 [81]）。商用的模型检验工具中，一个公开的算法是将正规表达式取反后转化为形如 $\mathsf{AG}f$ 的 CTL 公式进行检验[89]。在文 [82] 和 [83] 中，Legay、Wolper 等人还给出了基于重写和可达闭包计算的 ω-正规性质的符号化模型检验算法。刘万伟在其博士论文中，进一步将符号化模型检验算法扩展至三种使用交错自动机作为连接子 ETL 中；同时，还给出了针对具有特定形式线性 μ-演算公式的一种符号化模型检验算法（见文 [90]）。

在 2006 年，Jehle、Johannsen、Lange 和 Rachinsky 给出了线性 μ-演算的 BMC 算法[91]，他们基于线性 μ-演算的限界语义，根据其语法结构给出了一种递归编码方式，这样得到的编码与界值 k 是成立方关系的；随后，Heljanko 等人给出了一种针对 WABA（Weak Alternating Büchi Automata）的 BMC 编码[92]，该编码与界值 k 是成线性关系，它能比较方便的将编码方式应用到 PSL 上。

1.4.3 限界模型检验的完全性保证技术

BMC 是一种非完全的技术，即：它不能证明整个系统状态空间是否满足给定的规约性质。为了使 BMC 技术可以验证性质规约，研究人员提出了各种技术。最直接的办法就是找到 BMC 的完备阈值（即 Complete Threshold，简称 CT），使得在给定的界值内性质正确，就可以保证正个模型是满足性质的。但是计算完备阈值的复杂度非常高，正如文 [86] 所述，计算完备阈值的复杂度与模型检验问题的复杂度是一样的。因此，在实际应用中，大部分的 BMC 技术，都是给出完备阙值的上逼近，如文献 [1]、[86]、[93]、[94]、[95] 等都是通过计算完备阙值的上逼近来保证限界模型检验技术的完全性。

通过给出完备阈值的上逼近来保证 BMC 的完全性在实际实验中效果并不好，这是由于目前的算法给出的上逼近相对来说还是太大，经常超过 SAT 求解器的求解能力。因此，研究人员又探索了其他的技术来保证完全性。文献 [96]、[97] 提出了使用 CNF 代替 BDD 来表示模型信息，然后使用 SAT 求解器来迭代计算可达状态空间。该技术完全采用基于 BDD 的符号化模型检验算法，它仅仅是在存储结构以及操作子计算上根据 SAT 求解器做了适当的修改。实验表明，该技术在一定程度上提高了模型检验技术处理问题的规模。文献 [98]、[99]、[100]、[101] 提出了混合使用 BDD 和 SAT 来保证限界模型检验技术的完全性，其核心思想，都是通过 BMC 来查找系统的错误，通过 BDD 技术来保证完全性。该技术进一步提高了基于 SAT 的符号化模型检验算法处理系统的规模。文献 [102] 提出了一种

称为 k 步归纳的算法，该算法将原始的 BMC 编码分解成两部分，第一部分用来确定当前已经探测的状态空间是否满足性质；第二部分用来判断到达不满足给定性质的状态空间的最短路径长度。结合两部分的求解结果就可以确定性质是否被满足。

上述的技术都是通过精确的计算可达状态空间来保证 BMC 技术的完全性。2003 年，McMillan 提出了一种基于插值的符号化模型检验算法[103]，该算法通过计算可达状态空间的上逼近来验证给定的性质。该技术利用插值的性质，逐步逼近状态空间直到状态空间到达一个不动点。该算法极大的提高了基于 SAT 的符号模型检验技术处理问题的规模，至今，该技术仍是模型检验领域的研究重点。文献 [104]、[105]、[106]、[107]、[108]、[109] 等都是利用插值的属性来进行不同领域的验证的工作。2010 年，Bradley 提出了一种新的计算可达状态空间上逼近的技术，称为 IC3 算法（也称作 PDR），该技术通过增量式的求解可达状态空间的逼近来计算不动点。该算法效率非常高，一经提出，就引起了学术界和工业的界的广泛关注，基于该技术开发的模型检验工具 IImc 在 HWMCC2013 上取得了排名第一的成绩。目前，很多著名的研究机构都在基于该算法研究新的模型检验技术[110][111][112] [113][114][115]。该技术也被认为是近 10 年来，符号化模型检验领域最大的进展。该算法也是本文保证完全性算法的基础，本文基于 PDR 提出了两种新的可达性算法。本文的第5章将详细介绍这两个算法。

1.5 论文贡献

本文围绕基于 SAT 的符号化模型检验技术开展了一系列研究，旨在推动基于 SAT 的符号化模型检验技术在计算机辅助验证领域的应用并取得了一些成果。本文的主要贡献和创新点体现在以下几个方面:

- 提出了两种用于优化 LTL 限界模型检验算法的技术。

  第一种优化技术是一种称为反例集合保持的化简技术。该技术的核心思想是在模型检验算法验证之前先对待验证的性质归约进行一种轻量级的化简，使得化简后的性质与原性质关于待验证的模型是等价的。因此，在真正的模型检验中，就可以使用化简后的性质代替原性质进行验证，从而提高模型检验算法的验证效率。第二种优化技术则是利用 SAT 求解器的增量式的求解能力，优化基于语义的限界模型检验编码，实验表明，新的编码在验证效率上，相对于其他编码技术有很大的优势。这两种优化技术可以很自然的结合起来，从而提高 LTL 的限界模型的效率。

- 提出了 ω-正规性质的限界模型检验技术。

在工业应用中，许多重要的时序性质无法采用 LTL 表达，因此，针对 ω-正规性质的限界模型检验具有十分重要的理论和实际意义。本文针对两种具有 ω-正规表达能力的时序逻辑 ETL_{l+f} 和 QTL，分别提出了对应的限界模型检验算法。针对 ETL_{l+f}，本文提出了基于语义的限界模型检验技术。该技术的核心是构造 ETL_{l+f} 的 tableau，通过扩展 LTL 的 tableau 构造技术，本文提出了 ETL_{l+f} 的 tableau 构造技术。针对 QTL，本文提出了两种限界模型检验技术，分别为基于语法的限界模型检验和基于语义的限界模型检验。基于语法的限界模型检验的核心思想是将 QTL 公式的限界语义编码成 QBF 公式，然后利用 QBF 求解器来验证该问题。基于语义的限界模型检验同样是通过扩展 LTL 的 tableau 构造方法得到 QTL 的 tableau 构造方法，然后将限界模型检验问题转化为受限路径上的公平性路径查找问题。然后，将公平性路径查找问题编码成布尔公式，最后利用 SAT 求解器来验证该问题。

- 提出了基于 SAT 的符号化模型检验的完全性保证技术。

  制约限界模型检验技术应用的一个主要瓶颈在于其不是一种完全的技术，即它不能证明给定的模型是否满足给定的性质。然而，线性时序逻辑的模型检验问题最终都可以转化为公平性路径查找问题，而公平性路径查找问题可以转化为若干个可达性问题。因此，开发高效的可达性算法是解决基于 SAT 符号化模型检验完全性保证技术的核心。本文基于目前最好的可达性算法，即属性指导的可达性分析算法，提出了两种新的可达性算法。两种算法的核心思想都是试图通过双向逼近可达状态空间，从而加快算法的收敛速度。实验结果表明，本文提出的可达性算法相对于基本的算法有了很大的提高。

- 最后，本文还实现了两个实用的工具。

  模型检验技术的最终目的是应用，将理论上的算法转化为实用的工具是模型检验技术非常重要的一部分。本文实现了两个实用工具，分别支持本文提出的各个算法。工具 ENuSMV 是基于开源的符号化模型检验工具 NuSMV 实现的，它支持本文提出的 CEPRE技术、增量式的基于语义的 LTL 限界模型检验技术、基于语义的 ETL_{l+f} 的限界模型检验技术。工具 Reach 是一个全新的基于 SAT 的符号化模型检验工具，它不仅支持本文提出的交叠的 PDR 算法以及并行的 PDR 算法，还支持其他著名的可达性算法，包括限界模型检验算法、k 步归纳算法、基于插值的符号化模型检验算法以及基本的 PDR 算法。

1.6 论文结构

本文共分 7 章，其中：

- 第 1 章是绪论，主要介绍本课题的研究背景、研究动机、研究内容以及与本课题相关的研究工作。
- 第 2 章主要是回顾基本的概念以及基本算法，包括：ω-自动机、时序逻辑、迁移系统等基本概念，DPLL 算法、基本的 BMC 编码、基于插值的符号化模型算法以及属性制导的可达性分析算法等基本算法。
- 第 3 章的研究内容旨在提高目前基于 SAT 的 LTL 模型检验技术的效率。主要包括两个技术：反例集合保持的化简技术（CePRe）；增量式的基于语义的编码技术。
- 第 4 章研究内容旨在扩展目前基于 SAT 的限界模型检验技术的能力，使其支持 ω-正规性质的限界模型检验。本章针对两种具有完全 ω-正规性质表达能力的时序逻辑：ETL_{l+f} 以及 QTL，分别提出了对应的限界模型检验算法。
- 第 5 章的研究内容旨在保证基于 SAT 的模型检验技术的完全性。这一章将给出两个保证完全性的算法，分别为交叠的双向 PDR 算法以及并行的双向 PDR 算法。
- 第 6 章的研究内容旨在将理论算法转化成工具实现。这一章将给出以上算法的工具实现以及工具使用示例。
- 第 7 章主要是回顾本文的全部工作以及对未来工作的展望。

第二章　基础知识

本章将介绍本文所需要的基本知识，包括：ω-自动机，时序逻辑等基本概念，以及 DPLL 算法、基本的 BMC 编码、基于插值的模型检验算法、属性制导的可达性分析算法等基于 SAT 的符号化模型检验的几种主要算法。

2.1　基本概念

2.1.1　字、布尔公式

定义 2.1 (字): 给定有穷非空的字母表 Σ，Σ 上的一个字 w 是自然数集合 $\mathbb{N}$ 到 Σ 的一个映射。

通常，对于任意的 $i \in \mathbb{N}$，用 $w(i)$ 表示 w 中的第 i 个字母（i 从 0 开始）；字 w 通常写成序列 $w(0)w(1)\ldots$；用 $w[n]$ 表示字 w 的长度为 n 的有穷前缀。

定义 2.2 (布尔公式): 任给非空的布尔变量集合 $\mathcal{V}$，其上的布尔公式集合 $\mathrm{B}(\mathcal{V})$ 是满足下列条件的最小集合。

- $\top \in \mathrm{B}(\mathcal{V}); \bot \in \mathrm{B}(\mathcal{V})$。
- 若 $q \in \mathcal{V}$，则 $q \in \mathrm{B}(\mathcal{V})$。
- 若 $\psi \in \mathrm{B}(\mathcal{V})$，则 $\neg\psi \in \mathrm{B}(\mathcal{V})$。
- 若 $\varphi_1, \varphi_2 \in \mathrm{B}(\mathcal{V})$，则 $\varphi_1 \wedge \varphi_2 \in \mathrm{B}(\mathcal{V})$。

习惯上，还引入下列派生的布尔连接词。

$$\begin{aligned} \varphi_1 \vee \varphi_2 &\stackrel{\text{def}}{=} \neg(\neg\varphi_1 \wedge \neg\varphi_2) \\ \varphi_1 \rightarrow \varphi_2 &\stackrel{\text{def}}{=} \neg\varphi_1 \vee \varphi_2 \\ \varphi_1 \leftrightarrow \varphi_2 &\stackrel{\text{def}}{=} (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1) \end{aligned} \tag{2.1}$$

通常，称符号“$\neg$”为取非操作符，“$\wedge$”为合取操作符，“$\vee$”为析取操作符，“$\rightarrow$”为蕴含操作符，“$\leftrightarrow$”为等价操作符。

定义 2.3 (文字): 给定布尔变量集合 $\mathcal{V}$，文字 l 是一个变元 $v \in \mathcal{V}$ 或者它的非 $\neg v$。

定义 2.4 (短句): 一个短句 c 就是一组文字的析取。

我们使用 $|c|$ 来表示短句中的文字个数。由 c 中所包含文字的子集构成的短句 d 称为 c 的子句，记为 $d \subseteq c$。

一组短句的合取称为 CNF 范式。一个公式是 CNF 范式当且仅当其是一组短句的合取。

定义 2.5 (指派): 给定布尔变量集合 $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$，称映射 $s : \mathcal{V} \to \{0, 1\}$ 为 $\mathcal{V}$ 上的一个指派。

通常，我们将指派 s 记成一个文字的集合，也就是对于任意的 $v \in \mathcal{V}$，若 $s(v) = 0$，那么文字 $\neg v \in s$，若 $s(v) = 1$，那么 $v \in s$。

通常，我们使用 $var(\varphi)$ 来表示布尔公式的变量集合，比如 $var(\varphi) = \mathcal{V}$ 表示公式 φ 是变量集合 $\mathcal{V}$ 上的布尔公式。给定指派 s，用 $\varphi[s]$ 表示布尔公式 φ 在指派 s 下的真值。

定义 2.6 (布尔公式的满足性): 任给非空的布尔变量集合 $\mathcal{V}$，布尔公式 $\varphi \in \mathrm{B}(\mathcal{V})$，以及指派 $s \subseteq \mathcal{V}$，布尔公式的满足关系 $\models$ 定义如下：

- $s \models \top$。
- $s \not\models \bot$。
- 若 $\varphi = v \in \mathcal{V}$，则 $s \models \varphi$ 当且仅当 $v \in s$。
- 若 $\varphi = \neg\psi$，则 $s \models \varphi$ 当且仅当 $s \not\models \psi$。
- 若 $\varphi = \varphi_1 \wedge \varphi_2$，则 $s \models \varphi$ 当且仅当 $s \models \varphi_1$ 且 $s \models \varphi_2$。

若 $s \models \varphi$，称指派 s 满足φ。

给定布尔公式 φ 和 ψ，如果满足公式 φ 的所有指派都满足公式 ψ，我们就称公式 φ 蕴含 ψ，记为 $\varphi \to \psi$。不难得出，$\varphi \to \psi$ 成立当且仅当 $\varphi \wedge \neg\psi$ 是不可满足的。

定义 2.7 (消解): 给定两个短句 $c_1 = v \vee A$ 和 $c_2 = \neg v \vee B$，称短句 $A \vee B$（$A \vee B$ 不包含互补文字）是 c_1 和 c_2 的消解。

比如，短句 $\neg x_1 \vee x_2$ 和 $x_1 \vee x_3$ 的消解为 $x_2 \vee x_3$。

2.1.2 非确定的 ω-自动机

自动机理论一直都是计算科学领域一个重要的分支，上世纪 60 年代，Büchi[116]、Trakhtenbrot[117]、McNaughton[118]、Rabin[119] 等人提出了一套完整的无穷语言（包括无穷字语言和无穷树语言）上的有穷自动机框架[90]。

无穷语言上的自动机分为“**字自动机**”和“**树自动机**”两大类。本文涉及的研究对象是字自动机，因此本节主要介绍 ω-字自动机。

定义 2.8 (**非确定的** ω-**自动机**): 一个非确定的 ω-自动机是一个五元序偶 $\mathcal{A} = \langle \Sigma, Q, \delta, q, F \rangle$，其中：

- Σ 是一个有穷的字母表。
- Q 是一个有穷的状态集合。
- $\delta : Q \times \Sigma \to 2^Q$ 是迁移关系。
- $q \in Q$ 是初始状态。
- $F \subseteq Q$ 是一个接收状态集合。

给定无穷字 w 和 ω-自动机 $\mathcal{A} = \langle \Sigma, Q, \delta, q, F \rangle$，字 w 在 $\mathcal{A}$ 上的 **运行**是一个无穷的状态序列 $\sigma = q_0 q_1 \ldots \in Q^\omega$，其中 $q_0 = q$，对于任意的 $i \in \mathbb{N}$，都有 $q_{i+1} \in \delta(q_i, w(i))$。通常，我们称该序列的任意一个有穷前缀 $q_0 \ldots q_{n+1}$ 为子字 $w[n]$ 的有穷运行。

对于一个非确定的 ω-自动机，可以根据其接收条件对其进行分类。文献 [90] 根据接收条件对 ω-自动机进行了详细的分类，本文仅关心两种接收类型的自动机，分别为:

- Looping **自动机**: 无穷字 w 能被 ω-自动机 $\mathcal{A}$ 接收当且仅当存在一条 w 在 $\mathcal{A}$ 上的无穷运行;
- Finite **自动机**: 无穷字 w 能被 ω-自动机 $\mathcal{A}$ 接收当且仅当字 w 的有穷前缀 $w[n]$，且在其对应的有穷运行 $q_0 \ldots q_{n+1}$ 中，有 $q_{n+1} \in F$，也称该前缀为接收前缀。

通常，我们用 $\mathscr{L}(\mathcal{A})$ 来表示 ω-自动机 $\mathcal{A}$ 接收的无穷字集合，也称为自动机 $\mathcal{A}$ 所能识别的语言。

给定一个 ω-自动机 $\mathcal{A} = \langle \Sigma, Q, \delta, q, F \rangle$ 和一个状态 $r \in Q$，我们用 $\mathcal{A}^r$ 表示自动机 $\langle \Sigma, Q, \delta, r, F \rangle$，自动机 $\mathcal{A}^r$ 与自动机 $\mathcal{A}$ 除了初始状态，其他元素全部一样，显然，$\mathcal{A}$ 和 $\mathcal{A}^q$ 是同一个自动机。

2.1.3 时序逻辑

时序逻辑在上世纪 70 年代提出[120][121]，它主要用来描述系统的时序行为，即:“事件”发生的（相对或绝对）顺序。根据时序逻辑的语义是定义在线性结构还是定义在树型结构上，时序逻辑可以分为 **线性**时序逻辑和**分支**时序逻辑两大类。本文的工作主要是基于线性时序逻辑的，因此，本节将介绍本文中涉及到各种线性时序逻辑的语法和语义。

2.1.3.1 线性结构

定义 2.9 (**线性结构**): 给定原子命题集合 AP，一个 (无穷) 线性结构是一个映射 $\pi : \mathbb{N} \to 2^{AP}$，它的每个时刻 i 对应一个 AP 的子集。同时，一个线性结构可

以看作是以 2^{AP} 为字母表的 ω-字，其中 $\pi(i)$ 是其第 i 个字母。另外，对于任意的 $j \in \mathbb{N}$，定义函数 $\pi^j : \mathbb{N} \to 2^{AP}$ 为 $\pi^j(i) = \pi(i+j)$。

有了线性结构的定义，我们就可以给出各种时序逻辑的语法和语义了。

2.1.3.2 LTL 的语法和语义

LTL 是非常重要的一种时序逻辑，它已经被广泛的应用于学术界和工业界。目前，很多模型检验器都支持对 LTL 或者其变种的验证。通常，LTL 是在命题逻辑的基础上通过添加时序连接子 X 和 U 所获得的时序逻辑。但是，在实际应用中，带过去算子的 LTL 具有非常重要的实用性。例如，对于行为“若灯灭了，则开关一定被关掉过”，用带过去算子的 LTL 就很容易描述出来，但是仅用将来时序算子描述就不直观了。因此，LTL 中加入过去时序算子非常实用。本文的研究对象就是带过去算子的 LTL，因此，这里的 LTL 就是指带过去算子的 LTL。

定义 2.10 (LTL **语法**): 给定原子命题集合 AP，LTL 的语法可以定义如下：

- $\bot$ 和 $\top$ 是 LTL 公式。
- 任意的 $p \in AP$ 是 LTL 公式。
- 若 φ 是 LTL 公式，则 $\neg\varphi$ 也是 LTL 公式。
- 若 φ_1 和 φ_2 是 LTL 公式，则 $\varphi_1 \wedge \varphi_2$ 也是 LTL 公式。
- 若 φ 是 LTL 公式，则 $\mathsf{X}\varphi$ 和 $\mathsf{Y}\varphi$ 也是 LTL 公式。
- 若 φ_1 和 φ_2 是 LTL 公式，则 $\varphi_1 \mathsf{U} \varphi_2$ 和 $\varphi_1 \mathsf{S} \varphi_2$ 都是 LTL 公式。

LTL 的语义是定义在线性结构上的。

定义 2.11 (LTL **语义**): 给定原子命题集合 AP，线性结构 $\pi \in (2^{AP})^\omega$ 以及线性结构 π 上的位置 $i \in \mathbb{N}$，LTL 公式 φ 的语义可以归纳定义如下：

- 若 $\varphi = \top$，则 $\pi, i \models \varphi$ 一定成立。
- 若 $\varphi = \bot$，则 $\pi, i \not\models \varphi$。
- 若 $\varphi = p \in AP$，则 $\pi, i \models \varphi$ 当且仅当 $p \in \pi(i)$。
- 若 $\varphi = \neg\psi$，则 $\pi, i \models \varphi$ 当且仅当 $\pi, i \not\models \psi$。
- 若 $\varphi = \varphi_1 \wedge \varphi_2$，则 $\pi, i \models \varphi$ 当且仅当 $\pi, i \models \varphi_1$ 且 $\pi, i \models \varphi_2$。
- 若 $\varphi = \mathsf{X}\psi$，则 $\pi, i \models \varphi$ 当且仅当 $\pi, i+1 \models \psi$。
- 若 $\varphi = \mathsf{Y}\psi$，则 $\pi, i \models \varphi$ 当且仅当 $i > 0$ 且 $\pi, i-1 \models \psi$。
- 若 $\varphi = \varphi_1 \mathsf{U} \varphi_2$，则 $\pi, i \models \varphi$ 当且仅当存在 $j \geq i$，使得 $\pi, j \models \varphi_2$，并且对于任意的 $i \leq k < j$ 都有 $\pi, k \models \varphi_1$。

- 若 $\varphi = \varphi_1 \mathsf{S} \varphi_2$，则 $\pi, i \models \varphi$ 当且仅当存在 $0 \leq j \leq i$，使得 $\pi, j \models \varphi_2$ 且对于任意的 $j < k \leq i$，有 $\pi, k \models \varphi_1$。

通常，将 $\pi, 0 \models \varphi$ 直接记作 $\pi \models \varphi$。

在本文所涉及的任意一种时序逻辑中，都存在如下的缩写

$$\bot \stackrel{\text{def}}{=} \neg \top \tag{2.2}$$

为使用方便，LTL 中还可使用公式 2.1中定义的布尔连接子。除此之外，LTL 还定义了一些派生的时序连接子。

$$\begin{array}{ccc} \mathsf{F}\varphi \stackrel{\text{def}}{=} \top \mathsf{U} \varphi & \mathsf{Z}\varphi \stackrel{\text{def}}{=} \neg \mathsf{Y} \neg \varphi & \mathsf{O}\varphi \stackrel{\text{def}}{=} \top \mathsf{S} \varphi \\ \mathsf{G}\varphi \stackrel{\text{def}}{=} \neg \mathsf{F} \neg \varphi & & \mathsf{H}\varphi \stackrel{\text{def}}{=} \neg \mathsf{O} \neg \varphi \\ \varphi \mathsf{R} \psi \stackrel{\text{def}}{=} \neg(\neg \varphi \mathsf{U} \neg \psi) & & \varphi \mathsf{T} \psi \stackrel{\text{def}}{=} \neg(\neg \varphi \mathsf{S} \neg \psi) \end{array} \tag{2.3}$$

LTL 中的时序连接子 X，U，F，G，R 称为将来时序连接子 (future operators)，而 Y，Z，S，O，H 和 T 则成为过去时序连接子 (past operators)。如果一个 LTL 公式中不存在**过去时序连接子**（resp. **将来时序连接子**），我们称该公式为是**纯将来时**（resp. **纯过去时**）的 LTL 公式。

定理 2.1: 任意的 LTL 公式 φ 都存在一个等价的纯将来时的表达式[122]。

定理 2.1说明过去时序连接子并没有增强 LTL 的表达能力，但是，正如前面所说，过去时序算子可以使得 LTL 更加实用。不仅如此，从理论上来讲，添加过去时序算子也有优势。文献 [123] 指出，相对于仅适用纯将来时的 LTL 公式来描述规约性质，使用带过去算子的 LTL 所得到的规约更加紧致。

2.1.3.3 ETL_{l+f} 的语法和语义

根据所选用的自动机连接子的类型，可以对 ETL 进行分类：比如，以 Looping 和 Finite 自动机为连接子的 ETL 分别命名为 ETL_l 和 ETL_f。本文的主要研究对象是混合使用 Looping 和 Finite 自动机作为连接子的扩展时序逻辑，记为 ETL_{l+f}。下面将给出其具体的语法和语义。

定义 2.12 (ETL_{l+f} **语法**): 给定原子命题集合 AP，ETL_{l+f} 的语法可以归纳定义如下：

- $\bot$ 和 $\top$ 都是 ETL_{l+f} 公式。
- 任意的 $p \in AP$ 都是 ETL_{l+f} 公式。
- 若 φ 是 ETL_{l+f} 公式，则 $\neg\varphi$ 和 $\bigcirc\varphi$ 是 ETL_{l+f} 公式。
- 若 φ_1 和 φ_2 是 ETL_{l+f} 公式，则 $\varphi_1 \wedge \varphi_2$ 和 $\varphi_1 \vee \varphi_2$ 都是 ETL_{l+f} 公式。

- 若 $\mathcal{A}$ 是字母表 $\Sigma = \{a_1, \ldots, a_n\}$ 上 Looping 或 Finite 自动机，$\varphi_1, \ldots, \varphi_n$ 是 ETL$_{l+f}$ 公式，则 $\mathcal{A}(\varphi_1, \ldots, \varphi_n)$ 也是 ETL$_{l+f}$ 公式。

在各种 ETL 的原始定义中并没有显式的定义连接子“$\bigcirc$”，它可以通过自动机连接子定义。但是它在本文后续的基于语义的 BMC 编码中具有非常重要的作用，因此，在这里我们显式的给出其定义。显示的声明连接子“$\bigcirc$”并不会改变该逻辑的表达能力。

由于我们既使用 Looping 自动机作为时序连接子又使用 Finite 自动机作为时序连接子，因此，ETL$_{l+f}$ 其实是混合 ETL$_l$ 和 ETL$_f$ 的一种逻辑。一方面，ETL$_{l+f}$ 一般化了这两种逻辑；另一方面，它的描述能力更加紧致。

在 ETL$_{l+f}$ 中，连接子 $\vee$ 同前面布尔连接子相同。我们称形如 $\mathcal{A}(\varphi_1, \ldots, \varphi_n)$ 的公式为**自动机公式**。同自动机中对应的定义类似，若自动机公式 $\varphi = \mathcal{A}(\varphi_1, \ldots, \varphi_n)$，则将公式 $\mathcal{A}^q(\varphi_1, \ldots, \varphi_n)$ 记作 φ^q。因此，若 q 是 $\mathcal{A}$ 的初始状态，则 φ 与 φ^q 所指相同。

定义 2.13 (ETL$_{l+f}$ **语义**): 设原子命题集合为 AP，给定线性结构 $\pi \in (2^{AP})^\omega$，位置 $i \in \mathbb{N}$，ETL$_{l+f}$ 公式 φ 的语义可以归纳定义如下：

- 若 $\varphi = \top$，则 $\pi, i \models \varphi$ 一定成立。
- 若 $\varphi = \bot$，则 $\pi, i \not\models \varphi$。
- 若 $\varphi = \neg\psi$，则 $\pi, i \models \varphi$ 当且仅当 $\pi, i \not\models \psi$。
- 若 $\varphi = \bigcirc\psi$，则 $\pi, i \models \varphi$ 当且仅当 $\pi, i+1 \models \psi$。
- 若 $\varphi = \varphi_1 \wedge \varphi_2$，则 $\pi, i \models \varphi$ 当且仅当 $\pi, i \models \varphi_1$ 且 $\pi, i \models \varphi_2$。
- 若 $\varphi = \varphi_1 \vee \varphi_2$，则 $\pi, i \models \varphi$ 当且仅当 $\pi, i \models \varphi_1$ 或 $\pi, i \models \varphi_2$。
- 若 $\mathcal{A}$ 是字母表 $\{a_1, \ldots, a_n\}$ 上的 Looping 自动机且 $\varphi = \mathcal{A}(\varphi_1, \ldots, \varphi_n)$，则 $\pi, i \models \varphi$ 当且仅当存在无穷字 $w \in \mathscr{L}(\mathcal{A})$，且对于任意的 $j \in \mathbb{N}$，若 $w(j) = a_k$，则 $\pi, i+j \models \varphi_k$。
- 若 $\mathcal{A}$ 是字母表 $\{a_1, \ldots, a_n\}$ 上的 Finite 自动机且 $\varphi = \mathcal{A}(\varphi_1, \ldots, \varphi_n)$，则 $\pi, i \models \varphi$ 当且仅当存在无穷字 $w \in \mathscr{L}(\mathcal{A})$ 有一个有穷的接收前缀 $w[n]$，并且对于任意的 $j < n$，若 $w(j) = a_k$，则 $\pi, i+j \models \varphi_k$。

同样，记 $\pi, 0 \models \varphi$ 为 $\pi \models \varphi$。记集合 $\{\pi \mid \pi \models \varphi\}$ 为 $\mathscr{L}(\varphi)$，也称公式 φ 的语言。

值得注意的是，在自动机公式中，用于定义时序连接子的自动机的字母表中字母顺序是非常重要的，因此，在使用自动机作为时序连接子时，其字母表应该被看作是一个**向量**，而不是集合。

给定 ETL_{l+f} 公式 φ，记 $sub(\varphi)$ 为公式 φ 的所有子公式的集合。如果公式 φ 中的所有“$\neg$”操作符仅出现在原子命题或者自动机连接子之前，称公式 φ 是**否定范式**（NNF）。对于任意的 ETL_{l+f} 公式，反复利用 DeMorgan 定律以及公式2.4所示的两条规则，都可以化简为等价的否定范式。

$$\neg\bigcirc\varphi \stackrel{\text{def}}{=} \bigcirc\neg\varphi \tag{2.4}$$

2.1.3.4 QTL 语法和语义

正如之前所述，尽管 LTL 已经被广泛应用于工业界，但是其不能表达完全的 ω-正规性质是其最大的瓶颈。而 QTL（Quantified Temporal Logic）是通过向 LTL 中添加二阶量词得到的一种时序逻辑[76][77][25]。与 ETL 相比，QTL 仅仅对 LTL 添加了一个存在量词，这对于已经熟悉 LTL 的人员来说，语法上能更快的接受和理解。文献 [124][125] 指出，QTL 在验证复杂系统时具有很重要的应用，这是由于量词可以很自然的用来推理程序之间的精化关系。下面将给出 QTL 的语法和语义。

定义 2.14 (QTL **语法**): 设原子命题集合为 AP，QTL 公式 φ 可以归纳定义如下：

- $\top$ 和 $\bot$ 是 QTL 公式。
- 任意的 $q \in AP$ 是 QTL 公式。
- 若 φ 是 QTL 公式，则 $\neg\varphi$ 是 QTL 公式。
- 若 φ_1 和 φ_2 是 QTL 公式，则 $\varphi_1 \wedge \varphi_2$ 是 QTL 公式。
- 若 φ 是 QTL 公式，则 $\mathsf{X}\varphi$ 是 QTL 公式。
- 若 φ_1 和 φ_2 是 QTL 公式，则 $\varphi_1\mathsf{U}\varphi_2$ 是 QTL 公式。
- 若 φ 是 QTL 公式，则 $\exists q.\,\varphi(q \in AP)$ 是 QTL 公式。

在 QTL 中，派生布尔连接子 $\vee$、$\rightarrow$ 以及 $\leftrightarrow$ 和派生时序连接子 F、G以及 R与之前的定义相同。为方便起见，QTL 还定义了一个派生量词。

$$\forall q.\,\varphi \stackrel{\text{def}}{=} \neg\exists q.\,\neg\varphi \tag{2.5}$$

定义 2.15 (QTL **语义**): 设原子命题集合为 AP，给定线性结构 $\pi \in (2^{AP})^{\omega}$，位置 $i \in \mathbb{N}$，QTL 公式 φ 的语义可以归纳定义如下：

- 若 $\varphi = \top$，则 $\pi, i \models \varphi$。
- 若 $\varphi = p \in AP$，则 $\pi, i \models \varphi$ 当且仅当 $p \in \pi(i)$。
- 若 $\varphi = \neg\psi$，则 $\pi, i \models \varphi$ 当且仅当 $\pi, i \not\models \psi$。
- 若 $\varphi = \varphi_1 \wedge \varphi_2$，则 $\pi, i \models \varphi$ 当且仅当 $\pi, i \models \varphi_1$ 且 $\pi, i \models \varphi_2$。

- 若 $\varphi = \varphi_1 \mathsf{U} \varphi_2$，则 $\pi, i \models \varphi$ 当且仅当存在 $j \geq i$，使得 $\pi, j \models \varphi_2$，且对于任意的 $i \leq k < j$ 有 $\pi, k \models \varphi_1$。
- 若 $\varphi = \exists q.\psi$，则 $\pi, i \models \varphi$ 当且仅当存在 $\hat{\pi} \in (2^{AP})^{\omega}$ 使得 $\hat{\pi}, i \models \psi$，其中 $\hat{\pi}$ 满足：对于任意的 $j \geq i$ 以及任意的 $q' \in AP \backslash \{q\}$，有 $q' \in \pi(j)$ 当且仅当 $q' \in \hat{\pi}(j)$，

给定 QTL 公式 φ，通常，记 $\pi, 0 \models \varphi$ 为 $\pi \models \varphi$，记集合 $\{\pi \mid \pi \models \varphi\}$ 为 $\mathscr{L}(\varphi)$，也称公式 φ 的语言。

定理 2.2: QTL 的表达能力等价于 ω-正规语言[126][77]。

定理 2.2说明了 QTL 的表达能力。文献 [127] 进一步讨论了各种 QTL 变种和片段的表达能力，感兴趣的读者可以参阅 [127]。

2.1.4 迁移系统

定义 2.16 (**迁移系统**): 给定原子命题集合 AP，一个迁移系统（也称为 $Kripke$ 结构）是一个序偶 $M = \langle S, \rho, S_I, L \rangle$，其中：

- S 是一个有穷的状态集合。
- $\rho \subseteq S \times S$ 是一组迁移关系 (本文中均要求迁移关系是连续的，即：对于任意的 $s \in S$，存在 $s' \in S$ 使得 $(s, s') \in \rho$)。
- $S_I \subseteq S$，是初始状态集合。
- $L : S \to 2^{AP}$ 是命题标记函数。

对于任意一个迁移系统，都可以从线性和分支的角度分别派生出线性结构和分支结构。本文关心的时序逻辑都是基于线性结构定义语义的，因此本节主要给出线性结构的形式定义。

定义 2.17 (**路径**): 给定迁移系统 $M = \langle S, \rho, S_I, L \rangle$，称无穷序列 $\sigma = s_0 s_1 \ldots \in S^{\omega}$ 是 M 的一条路径，其中 $s_0 \in S_I$，对于任意的 $i \in \mathbb{N}$，有 $(s_i, s_{i+1}) \in \rho$。

给定迁移系统 M 和一条路径 σ，对于任意的 $j \in \mathbb{N}$，通常记 $\sigma(j) = s_j$。

任给一个迁移系统和其上的一条路径，都存在一个派生字，其定义如下。

定义 2.18 (**派生字**): 给定迁移系统 $M = \langle S, \rho, S_I, L \rangle$ 和一条路径 $\sigma = s_0 s_1 \ldots \in S^{\omega}$，称无穷字 $\pi = a_0 a_1 \ldots$ 是由 M 上的路径 σ 得到的派生字，其中对于任意的 $i \in \mathbb{N}$，有 $a_i = L(s_i)$。

通常，记 $\pi = L(\sigma)$，记 $\mathscr{L}(M)$ 为 M 所有的展开路径对应的派生字的集合。

定义 2.19 (迁移系统的乘积): 给定迁移系统 $M_i = \langle S_i, \rho_i, S_{Ii}, L_i \rangle (i = 1, 2)$，则 M_1 与 M_2 的乘积，记 $M_1 \otimes M_2$，是一个迁移系统 $\langle S, \rho, S_I, L \rangle$，其中：

- $S = \{(s, t) \mid s \in S_1, t \in S_2, L_1(s_1) = L_2(s_2)\}$。
- $((s, t), (s', t')) \in \rho$ 当且仅当 $(s, s') \in \rho_1$ 并且 $(t, t') \in \rho_2$。
- $S_I = (S_{I1} \times S_{I2}) \cap S$。
- $L((s, t)) = L_1(s) = L_2(t)$。

但是，上述的迁移系统的描述能力并不等价于 ω-正规性质。为了使迁移系统具有 ω-正规的表达能力，通常需要向迁移系统中加入额外的约束，也称为公平性约束。

定义 2.20 (公平迁移系统): 一个公平迁移系统是一个序偶 $\mathcal{M} = \langle M, \mathcal{F} \rangle$。其中，$M = \langle S, \rho, S_I, L \rangle$ 是一个迁移系统，称之为 $\mathcal{M}$ 的基迁移系统；$\mathcal{F} = \{F_1, \ldots, F_n\}$，是一组公平性约束，其中每个公平性约束 $F_i \subseteq S$，通常，记 $\mathcal{M}$ 为序偶 $\langle S, \rho, S_I, L, \mathcal{F} \rangle$。

有了公平迁移系统之后，我们可以定义其上的公平路径。

定义 2.21 (公平路径): 给定公平迁移系统 $\mathcal{M} = \langle M, \mathcal{F} \rangle$。其中，$M = \langle S, \rho, S_I, L \rangle$，$\mathcal{F} = \{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$，且 σ 是 M 中的一条路径。令 $\mathrm{Inf}(\sigma) = \{s \in S \mid$ 存在无穷多个$i \in \mathbb{N}$使得$\sigma(i) = s\}$。称 σ 是 $\mathcal{M}$ 中的一条公平路径，当且仅当对每个 $1 \leq i \leq n$，有 $\mathrm{Inf}(\sigma) \cap F_i \neq \emptyset$

通常，记 $\mathscr{L}(\mathcal{M})$ 为 M 所有的公平路径对应的派生字。

同样，我们可以定义两个公平迁移系统的乘积。

定义 2.22 (公平迁移系统的乘积): 给定公平迁移系统 $\mathcal{M}_i = \langle M_i, \mathcal{F}_i \rangle$，且 M_i 的状态集为 S_i $(i = 1, 2)$，则 $\mathcal{M}_1$ 与 $\mathcal{M}_2$ 的乘积 $\mathcal{M}_1 \otimes \mathcal{M}_2 = \langle M_1 \otimes M_2, \mathcal{F} \rangle$。其中 $\mathcal{F} = \{(F \times S_2) \cap S \mid F \in \mathcal{F}_1\} \cup \{(S_1 \times F) \cap S \mid F \in \mathcal{F}_2\}$。

对于迁移系统的乘积，很容易证明下面的定理。

定理 2.3: $\mathscr{L}(M_1 \otimes M_2) = \mathscr{L}(M_1) \cap \mathscr{L}(M_2)$；$\mathscr{L}(\mathcal{M}_1 \otimes \mathcal{M}_2) = \mathscr{L}(\mathcal{M}_1) \cap \mathscr{L}(\mathcal{M}_2)$。

一般地，对于公平迁移系统，我们只关心那些满足公平性约束的路径。

对于符号化模型检验来说，一个迁移系统一般是用一组布尔公式表示的，下面我们介绍迁移系统的布尔编码。

定义 2.23 (迁移系统的布尔编码): 给定原子命题集合 $AP = \{p_1, \ldots, p_n\}$，迁移系统 $M = \langle S, \rho, S_I, L\rangle$，其对应的布尔编码为序偶 $\langle AP, I, T\rangle$，其中：

- I 是 AP 上的布尔公式，且对 M 中任意的状态 s，$s \in S_I$ 当且仅当 $I[L(s)] = 1$。
- 令 $AP' = \{p_1', \ldots, p_n'\}$，$T$ 是 $AP \cup AP'$ 上的布尔公式，且对于任意的 $s, s' \in S$，$(s, s') \in \rho$ 当且仅当 $T[L(s), L(s')] = 1$。

根据定义可知，对于迁移系统 M，其布尔编码并没有显式的给出其对应的状态集合。这是由于 M 的状态完全可以通过初始条件和迁移关系来确定。这也是对迁移系统进行布尔编码的好处之一——不需要显式的存储每一个状态，这样就节省了大量的存储空间。在迁移系统的布尔编码中，任意的状态 s 都相当于命题集合 AP 的上的一个指派，因此，标记函数 L 也不需要显式的给出了。

通常，在命题集合 AP 确定的情况下，也记 $M = \langle I, T\rangle$，且 I 在状态 s 上的表示记为 $I[s]$，T 在状态对 (s, s') 下的表示记为 $T[s, s']$。

布尔迁移系统上的路径定义同之前的定义一样，只不过其要求初始状态是满足公式 I 的布尔指派，同样，任意的两个连续状态是满足公式 T 的布尔指派。

对于公平迁移系统的布尔编码，其定义仅仅对每一个公平性约束添加了一个对应的布尔公式，这里不再显式给出其定义。通常，在原子命题集合确定的情况下，我们使用 $\mathcal{M} = \langle I, T, \mathcal{C}\rangle$ 来表示公平迁移系统的布尔编码，其中 $\mathcal{C} = \{C_1, \ldots, C_m\}$，且每一个 $C_i \in \mathcal{C}$ 都是 AP 上的一个布尔公式。

在本文的后续章节中，如无特殊说明，迁移系统 M 就是指其对应的布尔编码 $\langle I, T\rangle$，公平迁移系统 $\mathcal{M}$ 也同样对应其布尔编码 $\langle I, T, \mathcal{C}\rangle$。为方便起见，对于**后续章节中提到的迁移系统中路径就是指该路径对应派生字**。

2.1.5 线性时序逻辑的模型检验问题

定义 2.24 (线性时序逻辑的模型检验问题): 给定迁移系统 $M = \langle I, T\rangle$ 以及线性时序逻辑公式 φ，如果对于任意的 $\pi \in \mathscr{L}(M)$，都有 $\pi \models \varphi$，称 M 满足 φ（记作 $M \models \varphi$）。

定义 2.24 并没有涉及公平性约束，这是因为对于带公平性约束的线性时序逻辑模型检验问题，其都可以转化成不带公平性约束的模型检验问题。假设给定公

平迁移系统 $\mathcal{M}=\langle M,\mathcal{C}\rangle$，其中 $\mathcal{C}=\{C_1,\ldots,C_n\}$。为每一个 $C_i\in\mathcal{C}$ 引入一个新的原子命题 p_i，使得 $p_i\in s$ 当且仅当 $C_i[s]=1$，令

$$\Psi=\bigwedge_{i=1}^{m}\mathsf{GF}p_i$$

不难证明：$\mathcal{M}\models\varphi$ 当且仅当 $M\models\Psi\rightarrow\varphi$。

2.2 基本算法

本节将给出本文所涉及到的几种基本算法。主要包括 SAT 求解器中经典的 DPLL 算法，基本的 BMC 编码，基于插值的符号化模型检验算法以及属性制导的可达性分析算法等。

2.2.1 DPLL 算法

布尔可满足问题，即 SAT 问题，是非常著名的受限约束的可满足问题。给定一个命题公式 φ，其 SAT 问题就是判定是否存在一个指派使得公式 φ 在该解释下为真，如果存在，则将该指派称为公式 φ 的可满足指派并称 φ 是可满足的；否则，称 φ 是不可满足的。尽管 SAT 问题是 NP-完全的[128]，但是 SAT 求解有很多实际应用，尤其是在大规模集成电路 (VLSI) 的辅助设计以及人工智能中。事实上，SAT 求解技术在过去的几十年有了很大的进步并已经成功解决了很多实际中问题，关于 SAT 求解技术的详细发展情况，可以参考文献 [129]。

现在大部分的 SAT 求解器都是基于 DPLL 算法[130][131] 的，DPLL 算法是一种基于深度优先搜索的可靠完全的算法，也被认为是目前为止 SAT 问题的标准解决方案。1960 年，Davis 和 Putnam 首先提出了 DP 算法[131]，该算法是基于消解的求解技术，其空间复杂度是指数级的，因此，该算法在实际应用中并不使用。1962 年，Davis、Logemann 和 Loveland 又提出了著名的 DLL 算法[130]。该算法是一种基于深度优先搜索思想的算法。该算法的核心是在可满足状态空间的深度优先搜索技术。虽然 DLL 算法相对于 DP 算法有了很大的提高，但是其也只能处理几十个命题变元的情况，因此也很难应用于实际问题。随后的几十年，SAT 求解领域涌现出很多优化技术，如局部搜索算法[132]、基于 BDD 图的算法[133]、基于广度优先搜索的算法[134][135] 等，但是其整体框架都没有脱离 DP 算法和 DLL 算法。因此，统称这类算法为 DPLL 算法。经过几十年的研究，1995 年，GRSAP[136] 系统总结了 DPLL 算法的基本流程，才使 DPLL 算法又重新成为研究的重点，而且被逐渐应用于实际问题的解决。除了 GRASP，还有 relsat[137]、SATO[138]、zChaff[139]、Berkmin[140]、Minisat[141] 都是效率比较高的 DPLL 算法的实现。2001 年，zChaff 的提出很多优化技术又进一步提高了 SAT 求解器处理问题

的规模。在 zChaff 中，布尔约束推导 (BCP) 过程了使用了懒散式的数据结构，极大提高了 BCP 蕴含推理的效率。zChaff 重新设计了 SAT 求解器的学习算法，找了一组非常高效的学习方式。目前大多数 SAT 求解器都是以 zChaff 为蓝本的。本节将给出 zChaff 实现中基本 DPLL 算法框架。

大部分 DPLL 算法的输入都是 CNF 范式，本文给出的算法框架也是以 CNF 为输入的。任意的布尔公式都可以转化成等价的 CNF 表示。最基本的方法就是利用德摩根定律进行展开。但是，利用德摩根定律获取 CNF 会导致公式长度指数性的增长。例如，将公式 2.6化简成等价的 CNF 需要 2^n 个短句。

$$(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n) \tag{2.6}$$

公式 2.6对应的 CNF 表示：

$$(x_1 \vee \ldots \vee x_{n-1} \vee x_n) \wedge (x_1 \vee \ldots \vee x_{n-1} \vee y_n) \wedge \ldots \wedge (y_1 \vee \ldots \vee y_{n-1} \vee y_n) \tag{2.7}$$

Tseitin 算法[142] 是一个线性的转换算法。对于任意的布尔公式 φ，Tseitin 算法能生成一个与该公式等价的 CNF 表示 ψ，并且 ψ 的大小与 φ 是成线性关系的。但是它需要引入一些新的变元。例如，对于公式 2.6，利用 Tseitin 算法得到的 CNF 表示如公式所示。

$$\begin{aligned}
&(z_1 \vee z_2 \vee \ldots \vee z_n) \wedge \\
&(\neg x_1 \vee \neg y_1 \vee z_1) \wedge (\neg z_1 \vee x_1) \wedge (\neg z_1 \vee y_1) \wedge \\
&(\neg x_2 \vee \neg y_2 \vee z_2) \wedge (\neg z_2 \vee x_2) \wedge (\neg z_2 \vee y_2) \wedge \\
&\ldots \wedge \\
&(\neg x_n \vee \neg y_n \vee z_n) \wedge (\neg z_n \vee x_n) \wedge (\neg z_n \vee y_n)
\end{aligned} \tag{2.8}$$

基本的 DPLL 算法主要包含两个子过程，分别为布尔约束推导 (Boolean Constraint Propagation, 简称 BCP) 和冲突学习 (Conflict-based Learning，简称 CL)。算法 2.1给出了 DPLL 算法的主要框架。

首先，算法初始化一个指派 $s = \emptyset$，然后进入一个主迭代过程。主迭代过程首先判断当前的 CNF 公式是否包含空短句，若包含，算法终止，回答 UNSAT，并给出导致这个空短句的消解树，即 UNSAT Core（即不可满足核，该概念在后面的技术中非常重要，在这里给出其产生原理）；若当前 CNF 公式中不包含空短句，算法根据蕴含图判断 CNF 中是否存在短句是冲突的，若存在短句是冲突的，调用冲突子句分析模块（$deduce()$ 函数）分析产生冲突的原因，并将新的短句加入到当前的 CNF 公式中；若没有子句是冲突的并且当前的蕴含图包含了 CNF 公式中所有的变量，那么算法找到一个可满足指派，即该蕴含图所包含的所有文字的集

算法 2.1: DPLL 算法

输入: CNF 公式 φ

输出: SAT 或者 UNSAT

```
1  begin
2      s = ∅;
3      while true do
4          if φ contains empty clause then
5              return UNSAT with UNSAT Core;
6          else if some clause c is in conflict then
7              add clause deduce(c, s, φ) to φ;
8              remove some literals in s;
9          else if IG(s, φ) contains all the variables in φ then
10             return SAT;
11         else
12             choose a new literal l;
13             add l to s;
```

合，算法终止并返回 SAT 和该指派；否则，算法选择一个不在当前指派中的文字加入到指派中，继续进行主迭代过程，直到算法终止。

BCP 过程的本质是根据当前的指派信息推导出其他文字的指派值。其工作原理如下，随着 DPLL 搜索过程的深入，给定公式中的变量不断的被赋值，短句中未赋值的变量数会越来越少，对于已经被赋值为 1 的短句，DPLL 可以暂时将这些短句从 CNF 中删除掉了，对于剩余的短句，如果短句中除了一个文字未被赋值之外，其他文字都已经被赋值为 0，那么就可以将该文字赋值成 1 了，这是因为，要使得当前的指派可以扩展成一个可满足指派，它必须能满足所有的短句。比如，给定的 CNF 中包含短句 $\neg x_1 \vee \neg x_2 \vee x_3$，若此时的指派为 $x_1 = 1, x_2 = 1$，那么 BCP 将强制对 x_3 赋值为 1。给定布尔公式 φ 以及当前指派 s，BCP 过程会构造一个蕴含图 (Implication Graph，记为 IG(φ, s))，它是一个有向无循环图 (Directed Acyclic Graph，简写为 DAG)。比如，若公式 $\varphi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3 \vee x_4)$ 以及当前指派 $s = \{\neg x_1, x_3\}$，那么 BCP 就会构造一个如图 2.1所示的 DAG。

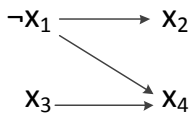


图 2.1　BCP 构造的 IG 图示例

给定一个 CNF 公式 φ 以及一个当前指派 s，设 BCP 构造的蕴含图 IG(φ, s) 为 (V, E)，其中 V 代表图中的节点，也就是文字的集合，对于任意的 $l \in V$，定义 $preds(l) = \{l' \in V \mid (l', l) \in E\}$，那么蕴含图 IG$(\varphi, s)$ 有下面几个特征：

- 指派 s 中包含的文字都是该图的根节点；
- 对于任意的 $l \in V$，如 $l \notin s$，那么 CNF 公式 φ 中一定存在短句 $l \vee \bigvee_{m \in preds(l)} \neg m$，记这个短句为 $cl(l, A, \varphi)$。
- 对于任意的 $v \in \mathcal{V}$，V 中不包含互补的文字。

BCP 会根据当前指派去构造一个蕴含图，如果蕴含图包含了公式中所有的变量，那么 DPLL 就找到了一组可满足指派，算法终止。如果 BCP 构造的蕴含图使得 CNF 公式中一个短句为 0，那么称这个短句是冲突的，此时 DPLL 算法会调用冲突学习模块来分析导致这个短句为 0 的原因，冲突学习的结果是产生一个**冲突子句**，然后将该冲突子句加入到已知的 CNF 中，从而避免再次搜索导致产生这个冲突的分支。

冲突学习的过程如下进行，给定 CNF 公式 φ，若 DPLL 发现当前的指派 s 使得其中一个短句为 0，设这个短句为 $c = l_1 \vee \ldots \vee l_n$，根据 IG$(s, \varphi)$ 的构造过程，其一定包含文字 $\neg l_1, \ldots, \neg l_n$。设 $\neg l_i \notin s$，即：不是 IG(s, φ) 的根节点，那么根据 IG(s, φ) 的第二个特征可知，φ 一定包含短句 $\neg l_i \vee \neg m_1 \vee \ldots \vee \neg m_k$，其中 $m_1, \ldots, m_k$ 是 $\neg l_i$ 的前驱节点，那么令短句 c 为 c 和 $cl(\neg l_i, s, \varphi)$ 的消解，不难发现，新的短句 c 还是冲突的。这个过程持续进行，直到消解出空短句或者得到一个短句 c 仅包含根节点的非。

例如，仍使用上面的例子，添加一个短句 $\neg x_2 \vee \neg x_4$，根据已知的蕴含图 (图 2.1) 可知，该短句是冲突的，冲突学习模块就会根据上面的过程随机选择一个文字，设为 $\neg x_2$，判断 $x_2 \notin s$ 且 $cl(x_2, s, \varphi) = x_1 \vee x_2$，令 c 为 $cl(x_2, s, \varphi) = x_1 \vee x_2$ 和 $\neg x_2 \vee \neg x_4$ 的消解 $x_1 \vee \neg x_4$，由蕴含图可知，该短句仍为冲突的，且存在 $x_4 \notin s$，迭代继续进行，令 c 为 $cl(x_4, s, \varphi) = x_1 \vee \neg x_3 \vee x_4$ 和 $x_1 \vee \neg x_4$ 的消解 $x_1 \vee \neg x_3$，根据蕴含图，此短句仍为冲突的，且该短句包含的所有文字的非都是蕴含图的根节点。迭代终止，将生成的 c 加入到 CNF 中。

给定 CNF 公式 φ，当前指派 s 以及冲突短句 c，记冲突学习的子过程为 $deduce(c, s, \varphi)$，其返回结果是一个短句，该短句由蕴含图的根节点的子集的非组成。

以上就是整个 DPLL 算法的基本过程，现代的 SAT 求解器，还加入了很多其他的优化策略，包括选择文字的优先级，非时序的回溯等，这里不再详细介绍。

2.2.2 基本的 BMC 编码

在介绍基本的 BMC 编码之前，我们首先给出 LTL 的限界语义。最初 BMC 技术并没有给出带过去时序连接子的 LTL 的布尔编码，因此，这里的 LTL 不包含过去时序连接子。LTL 的限界语义，顾名思义，它是将 LTL 公式定义在受限的路径上的。一般地，LTL 的限界语义都是定义在无穷路径上的有穷前缀上。若给定界 $k \in \mathbb{N}$，限界语义就是定义在路径 $s_0 \dots s_k$ 上的语义。

定义 2.25 (k-loop **路径**): 给定迁移系统上的路径 $\pi = s_0 s_1 \dots$，如果 $\pi = u \cdot v^\omega$，其中 $u = s_0 \dots s_{l-1}$，$v = s_l \dots s_k$，且 $T[s_k, s_l]$ 为真，则称 π 是一条 (k,l)-loop 的路径。给定 $k \in \mathbb{N}$，若存在一个 $0 \leq l \leq k$ 使得 π 是 (k,l)-loop 的路径，则称 π 是 k-loop 路径。

不难看出，如果给定的路径 π 是一条 k-loop 的路径，其在界 k 的限界语义与普通的语义是一样的。

定义 2.26 (LTL **的** k-loop **限界语义**): 给定界 $k \in \mathbb{N}$ 和一条 k-loop 路径 π，对于任意的 LTL 公式 φ，$\pi \models_k \varphi$ 当且仅当 $\pi \models \varphi$。(记 $\models_k$ 为路径 π 在 k 前缀上的满足关系)

对于非 k-loop 的限界语义，它就不一定满足上面的定义了。考虑性质 $\varphi = \mathsf{F}p$，其中 $p \in AP$，给定路径 π 以及界 k。根据 LTL 的原始语义，不难得出，若要使得 $\pi \models \varphi$ 成立，必须存在 $i \in \mathbb{N}$，使得 $p \in L(\pi(i))$ 成立。如果这个 $i > k$，我们就不能得出其在 k 步之内的语义为真。这是由于在受限的路径中，第 $k+1$ 状态不存在后继，因此，无法递归定义语义。因此，定义 LTL 在非 k-loop 路径上限界语义如下。

定义 2.27 (LTL **的非** k-loop **限界语义**): 给定界 $k \in \mathbb{N}$，非 k-loop 路径 π 以及位置 $0 \leq i \leq k$，LTL 公式 φ 的非 k-loop 限界语义可以定义如下。

- 若 $\varphi = \top$，则 $\pi, i \models_k \varphi$。
- 若 $\varphi = \bot$，则 $\pi, i \not\models_k \varphi$。
- 若 $\varphi = p \in AP$，则 $\pi, i \models_k \varphi$ 当且仅当 $p \in L(\pi(i))$。
- 若 $\varphi = \neg\psi$，则 $\pi, i \models_k \varphi$ 当且仅当 $\pi, i \not\models_k \psi$。
- 若 $\varphi = \varphi_1 \wedge \varphi_2$，则 $\pi, i \models_k \varphi$ 当且仅当 $\pi, i \models_k \varphi_1$ 且 $\pi, i \models_k \varphi_2$。
- 若 $\varphi = \varphi_1 \vee \varphi_2$，则 $\pi, i \models_k \varphi$ 当且仅当 $\pi, i \models_k \varphi_1$ 或 $\pi, i \models_k \varphi_2$。
- 若 $\varphi = \mathsf{X}\psi$，则 $\pi, i \models_k \varphi$ 当且仅当 $\pi, i+1 \models_k \psi$ 且 $i+1 \leq k$。

- 若 $\varphi = \varphi_1 \mathsf{U} \varphi_2$，则 $\pi, i \models_k \varphi$ 当且仅当存在 $i \leq j \leq k$，使得 $\pi, j \models_k \varphi_2$ 且对于任意的 $i \leq m < j$ 有 $\pi, m \models_k \varphi_1$。
- 若 $\varphi = \mathsf{G}\psi$，则 $\pi, i \not\models_k \varphi$。
- 若 $\varphi = \mathsf{F}\psi$，则 $\pi, i \models_k \varphi$ 当且仅当存在 $i \leq j \leq k$ 使得 $\pi, j \models_k \psi$。
- 若 $\varphi = \varphi_1 \mathsf{R} \varphi_2$，则 $\pi, i \models_k \varphi$ 当且仅当存在 $i \leq j \leq k$ 使得 $\pi, j \models_k \varphi_1$，且对于任意的 $i \leq m < j$，有 $\pi, m \models_k \varphi_2$。

同样，记 $\pi, 0 \models_k \varphi$ 为 $\pi \models_k \varphi$。为方便起见，上面的定义显示的给出了派生连接子的语义。

从上面的定义可以得到下面的引理。

引理 2.1: 给定无穷路径 π，对于任意的 LTL 公式 φ，若 $\pi \models_k \varphi$，则 $\pi \models \varphi$。

BMC 技术的核心就是将 k 步之内的模型检验问题编码成布尔公式，然后利用 SAT 求解器来求解。给定迁移系统 M 以及 LTL 公式 φ，BMC 编码的最终目标是得到一个布尔公式 $[\![M, \varphi]\!]$，使得该公式是可满足的当且仅当 M 中存在一条路径满足公式 φ。下面我们分三部分来介绍其编码。

定义 2.28 (**迁移系统的 k 步展开**): 给定迁移系统 M 以及界 $k \geq 0$，那么 M 的 k 步展开为：

$$[\![M]\!]_k = I[s_0] \wedge \bigwedge_{i=0}^{k-1} T[s_i, s_{i+1}]$$

上面的公式构造了 M 中所有长度 k 的路径，其中每一个状态 s_i 都是由一组状态变量版本来表示的。对于任意的 $p \in AP$，记 p_i 为 s_i 上的变量。

LTL 公式的布尔编码依赖于路径 π 的形状，因此我们首先给出一个定义判断路径是否为循环路径的布尔公式。

定义 2.29 (**循环条件**): 给定路径 π 以及界 k，令 $L_k \leftrightarrow \bigvee_{l=0}^{k} {}_lL_k$，其中 ${}_lL_k \leftrightarrow T[s_k, s_l]$。

下面将给出 LTL 的布尔编码，根据路径是否是 k-loop 的，对于一个 LTL 公式有两种编码方式。定义 2.30（“$[\![\cdot]\!]_k^i$”）给出了无循环路径的编码，其中 k 是给定的界，它表示了路径前缀的长度，i 代表了前缀中的位置。定义 2.32（“${}_l[\![\cdot]\!]_k^i$”）给出了 k-loop 路径上的编码。

定义 2.30 (**无循环的 LTL 布尔编码**): 给定 LTL 公式 φ 以及 $k, i \in \mathbb{N}$，且 $i \leq k$，那么 φ 在长度为 k 的无循环路径上的布尔编码可以如下递归的进行。

- 若 $\varphi = p \in AP$，那么 $[\![\varphi]\!]_k^i := p_i$。
- 若 $\varphi = \neg p$，那么 $[\![\varphi]\!]_k^i := \neg p_i$。
- 若 $\varphi = \varphi_1 \wedge \varphi_2$，那么 $[\![\varphi]\!]_k^i := [\![\varphi_1]\!]_k^i \wedge [\![\varphi_2]\!]_k^i$。
- 若 $\varphi = \varphi_1 \vee \varphi_2$，那么 $[\![\varphi]\!]_k^i := [\![\varphi_1]\!]_k^i \vee [\![\varphi_2]\!]_k^i$。
- 若 $\varphi = \mathsf{G}\psi$，那么 $[\![\varphi]\!]_k^i := \bot$
- 若 $\varphi = \mathsf{F}\psi$，那么 $[\![\varphi]\!]_k^i := \bigvee_{j=i}^{k} [\![\psi]\!]_k^j$。
- 若 $\varphi = \mathsf{X}\psi$，如果 $i < k$，则 $[\![\varphi]\!]_k^i := [\![\psi]\!]_k^{i+1}$；否则，$[\![\varphi]\!]_k^i := \bot$。
- 若 $\varphi = \varphi_1 \mathsf{U} \varphi_2$，那么 $[\![\varphi]\!]_k^i := \bigvee_{j=i}^{k} ([\![\varphi_2]\!]_k^j \wedge \bigwedge_{m=i}^{j-1} [\![\varphi_1]\!]_k^m)$。
- 若 $\varphi = \varphi_1 \mathsf{R} \varphi_2$，那么 $[\![\varphi]\!]_k^i := \bigvee_{j=i}^{k} ([\![\varphi_1]\!]_k^j \wedge \bigwedge_{m=i}^{j} [\![\varphi_2]\!]_k^m)$。

下面将给出 LTL 公式在 k-loop 路径上的编码（“${}_l[\![\cdot]\!]_k^i$”），该编码不仅依赖于受限路径长度 k 和当前位置 i，它还依赖于路径循环的起始位置 l。

首先给出 k-loop 路径中后继的定义。

定义 2.31 (k-loop **中的后继**): 令 $k, l, i \in \mathbb{N}$，其中 $l, i \leq k$，那么

$$succ(i) = \begin{cases} i+1 & \text{if } i < k \\ l & \text{if } i = k \end{cases} \tag{2.9}$$

定义 2.32 (k-loop **上的** LTL **布尔编码**): 给定 LTL 公式 φ 以及 $k, l, i \in \mathbb{N}$，且 $l, i \leq k$，那么 φ 在长度为 k 的 k-loop 路径上的布尔编码可如下递归的进行。

- 若 $\varphi = p \in AP$，那么 ${}_l[\![\varphi]\!]_k^i := p_i$。
- 若 $\varphi = \neg p$，那么 ${}_l[\![\varphi]\!]_k^i := \neg p_i$。
- 若 $\varphi = \varphi_1 \wedge \varphi_2$，那么 ${}_l[\![\varphi]\!]_k^i := {}_l[\![\varphi_1]\!]_k^i \wedge {}_l[\![\varphi_2]\!]_k^i$。
- 若 $\varphi = \varphi_1 \vee \varphi_2$，那么 ${}_l[\![\varphi]\!]_k^i := {}_l[\![\varphi_1]\!]_k^i \vee {}_l[\![\varphi_2]\!]_k^i$。
- 若 $\varphi = \mathsf{G}\psi$，那么 ${}_l[\![\varphi]\!]_k^i := \bigwedge_{j=min(i,l)}^{k} {}_l[\![\psi]\!]_k^j$。
- 若 $\varphi = \mathsf{F}\psi$，那么 ${}_l[\![\varphi]\!]_k^i := \bigvee_{j=min(i,l)}^{k} {}_l[\![\psi]\!]_k^j$。
- 若 $\varphi = \mathsf{X}\psi$，那么 ${}_l[\![\varphi]\!]_k^i := {}_l[\![\psi]\!]_k^{succ(i)}$。
- 若 $\varphi = \varphi_1 \mathsf{U} \varphi_2$，那么

$${}_l[\![\varphi]\!]_k^i := \bigvee_{j=i}^{k} ({}_l[\![\varphi_2]\!]_k^j \wedge \bigwedge_{m=i}^{j-1} {}_l[\![\varphi_1]\!]_k^m) \vee \bigvee_{j=l}^{i-1} ({}_l[\![\varphi_2]\!]_k^j \wedge \bigwedge_{m=i}^{k} {}_l[\![\varphi_1]\!]_k^m \wedge \bigwedge_{m=l}^{j-1} {}_l[\![\varphi_1]\!]_k^m)$$

- 若 $\varphi = \varphi_1 \mathsf{R} \varphi_2$，那么

$${}_l[\![\varphi]\!]_k^i := \bigwedge_{j=min(i,l)}^{k} ({}_l[\![\varphi_2]\!]_k^j) \vee \bigvee_{j=i}^{k} ({}_l[\![\varphi_1]\!]_k^j \wedge \bigwedge_{m=i}^{j} {}_l[\![\varphi_2]\!]_k^m) \vee \bigvee_{j=l}^{i-1} ({}_l[\![\varphi_1]\!]_k^j \wedge \bigwedge_{m=i}^{k} {}_l[\![\varphi_2]\!]_k^m \wedge \bigwedge_{m=l}^{j} {}_l[\![\varphi_2]\!]_k^m)$$

得到 LTL 公式的布尔编码后，受限路径的模型检验问题就可以表示成布尔公式了，给定迁移系统 M，LTL 公式 φ 以及界 $k \in \mathbb{N}$，公式 2.10给出了整体编码。

$$[\![M,\varphi]\!]_k := [\![M]\!]_k \wedge ((\neg L_k \wedge [\![\varphi]\!]_k^0) \vee \bigvee_{l=0}^{k} ({}_lL_k \wedge {}_l[\![\varphi]\!]_k^0)) \tag{2.10}$$

习惯上，公式 2.10中的编码也写成如下形式:

$$[\![M,\varphi]\!]_k := [\![M]\!]_k \wedge [\![\neg\varphi]\!]_k \tag{2.11}$$

其中，

$$[\![\neg\varphi]\!]_k = (\neg L_k \wedge [\![\varphi]\!]_k^0) \vee \bigvee_{l=0}^{k} ({}_lL_k \wedge {}_l[\![\varphi]\!]_k^0)$$

定理 2.4: $[\![M,\varphi]\!]_k$ 是可满足的当且仅当 M 中存在一条路径 π 且 $\pi \models_k \varphi$。

特殊的，如果待验证的性质为 $\varphi = \mathsf{F}P$ 且 P 为布尔公式，那么其编码如下:

$$[\![M,\varphi]\!]_k := [\![M]\!]_k \wedge \bigvee_{i=0}^{k} [\![P]\!]_k^i \tag{2.12}$$

公式 2.12就是 BMC 验证**可达性问题**时的布尔编码，直观上讲，其描述了这样的一个问题：模型 M 中是否存在一条路径在 k 步之内可达满足 P 的状态。通常，我们还会将该可达性问题描述成下面的公式。

$$[\![M,\varphi]\!]_k := I[s_0] \wedge T[s_0,s_1] \wedge \ldots \wedge T[s_{k-1,s_k}] \wedge \bigvee_{i=0}^{k} P[s_i] \tag{2.13}$$

给定一个迁移系统 M，一个布尔公式 P，若 M 中的所有的可达状态都满足 P，记作 $M \models P$。称 P 为 M 的不变式，也称判断 P 是否为 M 的不变式的问题为**安全性问题**。对于安全性问题而言，由 BMC 的定义可知，在给定的界 k 内，BMC 无法回答 $M \models P$ 是否成立。但是 BMC 可以回答这样的问题：模型 M 是否存在一条长度 k 的路径可达满足 $\neg P$ 的状态，即可以查找 k 步之内的反例路径。其编码公式如下所示

$$[\![M,\varphi]\!]_k := I[s_0] \wedge T[s_0,s-1] \wedge \ldots \wedge T[s_{k-1,s_k}] \wedge \bigvee_{i=0}^{k} \neg P[s_i] \tag{2.14}$$

公式 2.14可以转化成 CNF 表示，然后调用 SAT 求解器判断其是否可满足。若公式 2.14 是可满足的，那么证明存在一条路径 k 步之内可达满足 $\neg P$ 的状态，

即反例路径；否则，若公式 2.14不可满足，说明模型 M 中不存在长度小于等于 k 的反例路径，通常，记上面的编码为 $BMC(M,P,k)$。**本章后面所提到的算法都是对安全性验证的算法**，在这里，我们给出其基本描述。BMC 技术是不完全的技术，其在安全性验证中主要用来查找反例。

2.2.3 基于反例的抽象精化算法

由本文的第 2.2.2 可知，BMC 技术是一种不完全的技术，在验证安全性问题时，其并不能回答性质是否正确。而基于反例的抽象精化是结合基于 BDD 和 BMC 的一种可靠完全的技术[100]。给定系统模型 M 以及性质 φ，它的基本原理是首先对模型 M 做保守抽象得到 M'，然后使用基于 BDD 的技术验证 $M' \models \varphi$ 是否成立，若成立，则 $M \models \varphi$；否则设产生反例 A，利用 BMC 技术判断 A 是否为真反例。如果 A 是真反例，算法终止；否则，根据反例 A 精化模型 M' 得到 M''，算法继续迭代。算法 2.2 给出了基于反例的抽象精化技术的主要框架。

算法 2.2: 基于反例的抽象精化算法

输入: M，P

输出: safe 或者 counterexample

```
1  begin
2      initial abstraction M';
3      while true do
4          if BDD_MC(M', P) then
5              return safe;
6          extract counterexample A;
7          let k=length of A;
8          if BMC(M, P, k, A) is SAT then
9              return counterexample;
10         refine M' with proof of BMC(M, P, k, A);
```

算法中的 $BMC(M,P,k,A)$ 的编码如公式 2.15 所示，其中 A_i 表示反例 A 中第 i 步的具体指派。如果公式 2.15 是可满足的，说明 A 是一条真反例，否则；算法将利用公式的不可满足核用来精化抽象模型 M'。

$$BMC(M,P,k,A) = I[s_0] \wedge \bigwedge_{i=0}^{k-1} T[s_i, s_{i+1}] \wedge \bigvee_{i=0}^{k} \neg P[s_i] \wedge \bigwedge_{i=0}^{k} A_i \tag{2.15}$$

2.2.4 基于证明的抽象精化算法

本文的第 2.2.3给出了一种基于反例的抽象精化技术，本节我们将介绍另外一个抽象精化算法——基于证明的抽象精化算法[101]。

在基于反例的抽象精化中，一旦发现了一条长度为 k 的假反例，算法将根据这条假反例精化抽象模型从而排除掉这条假反例。但是抽象后的模型中仍然有可能存在长度为 k 的假反例。基于证明的抽象精化技术可以一次性排除掉所有长度为 k 的假反例，从而保证算法得到的抽象模型更加精确。算法2.3 描述了基于证明的抽象精化技术的主要框架。

算法 2.3: 基于证明的抽象精化算法

输入: M，P

输出: safe 或者 counterexample

```
1  begin
2      initial k;
3      while true do
4          if BMC(M, P, k) is SAT then
5              return counterexample;
6          derive new abstraction M' with proof of BMC(M, P, k);
7          if BDD_MC(M', P) then
8              return safe;
9          else
10             k = k'; // k' is the length of counterexample
```

算法 2.3 中的 $BMC(M,P,k)$ 的编码在本文的公式 2.14 已经给出。若其是可满足的，则算法找到一条真反例；否则，SAT 求解器会给出一个公式 $BMC(M,P,k)$ 不可满足的证明，即不可满足核。利用这个不可满足核，算法抽象模型 M 得到 M'，并且 M' 保证没有小于等于 k 的反例路径。然后算法调用基于 BDD 的算法验证该抽象模型是否满足给定的性质，若满足，算法终止，回答性质正确；否则，将 k 设为该反例路径的长度，算法继续进行迭代。

2.2.5 基于归纳的模型检验算法

文献 [102] 给出一种称为 k 步归纳的技术（k-induction)。该技术试图通过用归纳的方式来证明给定的性质。算法 2.4 给出了该技术的基本框架。

算法 2.4: 基于归纳的模型检验算法

输入: M，P

输出: safe 或者 counterexample

```
1 begin
2     initial k := 0;
3     while true do
4         if Base(M, P, k) is SAT then
5             return counterexample;
6         else if Step(M, P, k) is UNSAT then
7             return safe;
8         k++;
```

算法 2.4中的 $Base(M,P,k)$ 和 $Step(M,P,k)$ 的编码分别如公式 2.16 和 2.17 所示。

$$Base(M,P,k) = I[s_0] \wedge \bigwedge_{i=0}^{k-1} T[s_i, s_{i+1}] \wedge \bigvee_{i=0}^{k} \neg P[s_i] \tag{2.16}$$

$$Step(M,P,k) = \bigwedge_{0 \le i < j \le k} (s_i \neq s_j) \wedge \bigwedge_{i=0}^{k} T[s_i, s_{i+1}] \wedge \bigwedge_{i=0}^{k} P[s_i] \wedge \neg P[s_{k+1}] \tag{2.17}$$

不难看出，公式 $Base(M,P,k)$ 其实就是公式 $BMC(M,P,k)$。如算法 2.4 所示，若 $Base(M,P,k)$ 是不可满足的，说明 M 中不存在长度小于等于 k 的反例路径。直观上讲，公式 $Step(M,P,k)$ 表示的是从满足 P 的状态出发，长度为 k 的可达 $\neg P$ 的无循环路径。若 $Step(M,P,k)$ 是可满足的，令 k 加 1。否则，说明从满足 P 的状态出发不存在一条长度为 k 的可达 $\neg P$ 的无循环路径。如果该公式不可满足，不难证明，从 I 出发可达的所有状态都不会到达 $\neg P$。因此，性质是正确的。

2.2.6 基于插值的模型检验算法

定义 2.33 (**插值**): 给定布尔公式 $\varphi = A \wedge B$，若 φ 是不可满足的，那么一定存在一个布尔公式 $\mathcal{I}$，且 $\mathcal{I}$ 满足：

- $A \rightarrow \mathcal{I}$；
- $\mathcal{I} \wedge B$ 是不可满足的；
- 公式 $\mathcal{I}$ 中仅包含 A 和 B 的公共变量。

称 $\mathcal{I}$ 是公式 $A \wedge B$ 的插值。

直观上来讲，插值代表了 SAT 求解器用来证明公式 $A \wedge B$ 不可满足的信息。对于命题逻辑来说，如果公式 $A \wedge B$ 是不可满足的，那么 SAT 求解器基于不可满足的证明可以在线性的时间内给出其插值[143]。在本文的第 2.2.1 节中，我们给出了基本的 DPLL 算法框架。从算法 2.1 中可以看出，给定一个布尔公式 φ，若 φ 是可满足的，那么 DPLL 算法会给出一个可满足指派；若公式 φ 是不可满足的，那么 DPLL 算法会给出一个不可满足核（UNSAT Core）。文献 [103] 基于不可满足证明给出了一个线性时间内生成插值的具体算法，这里我们不再详细讨论其具体细节，感兴趣的读者可以参阅文献 [103]。

基于插值的符号化模型检验算法就是使用插值来计算上逼近的可达状态空间，从而加快算法找到不动点的技术。算法 2.5给出基于插值的符号化模型检验算法的基本框架。

算法 2.5: 基于插值的符号化模型检验算法

输入: $M = \langle S, I, T, L \rangle$，$P$

输出: safe 或者 counterexample

```
1  begin
2      initialize k;
3      while true do
4          if BMC(M, P, k) is SAT then
5              return counterexample;
6          R := I;
7          while true do
8              M' := ⟨S, R, T, L⟩;
9              let C := Pref(M', P, k) ∧ Suff(M', P, k);
10             if C is SAT then
11                 goto 17;
12             compute interpolant ℐ of Pref(M', P, k) ∧ Suff(M', P, k);
13             R' := ℐ ∨ R;
14             if R' = R then
15                 return safe;
16             R := R';
17         k++;
```

下面我们对算法的执行过程进行简单的介绍。算法首先调用调用 SAT 求解器验证 $BMC(M,P,k)$ 是否可满足，若可满足，则返回反例路径。否则，将 $BMC(M,P,k)$ 分解成两部分 $Pref(M,P,k)$ 和 $Suff(M,P,k)$，其中 $Pref(M,P,k) = I[s_0] \wedge T[s_0,s_1]$，$Suff(M,P,k) = T[s_1,s_2] \wedge \ldots \wedge T[s_{k-1},s_k] \wedge \bigvee_{i=0}^{k} \neg P[s_i]$，直观上讲，$Pref(M,P,k)$ 表示了模型中从初始状态集合出发一步可到达的状态空间。如果我们将 $Pref(M,P,k)$ 看作公式 A，$Suff(M,P,k)$ 看作公式 B，由于 $A \wedge B$ 是不可满足的，那么一定存在一个插值 $\mathcal{I}$ 且 $A \rightarrow \mathcal{I}$，即满足 A 的状态一定满足 $\mathcal{I}$，所以 A 中的状态也都在 $\mathcal{I}$ 中，由于 $\mathcal{I} \wedge B$ 仍是不可满足的，因此 $\mathcal{I}$ 中的状态经过 $k-1$ 步仍然不能到达 $\neg P$，所以 $\mathcal{I}$ 的状态是安全的。事实上，$\mathcal{I}$ 就是 A 的一个上逼近，它代表了从模型 M 出发经过一步迁移可达的状态空间的上逼近。若 $\mathcal{I}$ 中不包含任何新的状态，即 $\mathcal{I} \vee I = I$，那么算法找到了了一个不动点，算法终止，并回答性质正确，否则，令新得到的状态空间 $R = \mathcal{I} \vee I$ 代替初始状态集合 I，重新进行上面的迭代过程。

在算法 2.5中，可以将 R 看做是当前的可达状态空间的一个上逼近。将 $BMC(M',P,k)$ 看作是基于 R 的 k 步展开。若 $BMC(M',P,k)$ 是可满足的，算法不能确定这是否是一条真反例，因为 M' 中的初始状态集合是 R，而 R 中已经引入了可能不属于 M 的状态。因此，算法跳转到第 17 行，重新进行迭代。

从算法的执行过程可知，基于插值的符号化模型检验算法是一种逼近的技术。它不精确的计算一步可达的状态空间，而是利用插值的性质计算可达状态空间的上逼近，这一方面加快了算法遍历整个状态空间的速度，另一方面则有可能引起虚假反例的问题。但是，McMillan 在文献 [103] 中的实验结果表明，该算法大大的提高了基于 SAT 模型检验技术的验证效率。直到现在，基于插值的模型检验技术仍旧是模型检验领域的研究热点之一。

2.2.7 属性制导的可达性分析算法

本节将介绍一种基于 SAT 的可达性分析技术，该技术的主要特点就是基于属性制导的方法。因此该方法也称作 Property Directed Reachability(PDR)。该技术最初的算法是由 Bradley 等人提出的[144]，其核心是增量式的求解相对归纳不变式，通过这些不变式不断的增强要验证的属性，从而找到一个迭代不动点。在最初文献 [144] 中，该技术被命名为 Incremental Construction of Inductive Clauses for Indubitable Correctness(IC3)。后来，N. Eén 等人[110] 给出了 IC3 的一种高效的实现技术，提出了多种优化策略并将其命名为 PDR。本文我们采用文献 [110] 的命名方式，本质上，其与 IC3 描述的是同一种技术。

具体介绍 PDR 算法之前，我们先给出算法需要用到的基本概念和定义。

给定一个布尔迁移系统 M 和一个状态 s，如果存在一条 M 中的路径可达 s，我们就称状态 s 是可达的。给定布尔迁移系统 M 和布尔公式 φ，如果满足 M 的所有可达状态都是满足公式 φ 的，我们就称公式 φ 是 M 的一个不变式；否则，一定存在一条 M 上的路径包含一个不满足公式 φ 的状态 s，我们称这条路径为**反例路径**。

定义 2.34 (归纳不变式)：给定迁移系统 M 和布尔公式 φ，我们称公式 φ 关于迁移系统 M 是归纳的，其中：

- φ 是初始可满足的，即：$I \rightarrow \varphi$；
- φ 关于迁移关系是归纳的，即：$\varphi \wedge T \rightarrow \varphi'$，其中 φ' 是 $\varphi[AP']$ 的简写。

定义 2.35 (相对归纳不变式)：给定迁移系统 M 以及 M 的归纳不变式 ψ，我们称公式 φ 相对于 ψ 是归纳不变式，如果它满足下面的两个条件：

- φ 是初始可满足的，即：$I \rightarrow \varphi$；
- $\psi \wedge \varphi \wedge T \rightarrow \varphi'$ 成立。

2.2.7.1 最小归纳子句生成算法

在具体介绍 PDR 算法之前，我们首先介绍一下归纳子句生成算法。给定迁移系统 $M = \langle I, T\rangle$ 以及待验证性质 P，如果短句 c 相对于性质 P 是归纳的，也就是 $c \wedge P \wedge T \rightarrow c'$ 成立，归纳子句生成算法就是找到短句 c 的子句 d，使得 $d \wedge P \wedge T \rightarrow d'$ 成立并且对于任意的 $e \subseteq c$，若 $e \wedge P \wedge T \rightarrow e'$ 成立，那么 $d \subseteq e$。直观的讲，该算法就是生成一个包含于 c 的相对于 P 归纳的最小子句。文献 [145] 给出了一个线性的实现算法，本文的工作也是基于这个最小归纳子句生成算法来开展的，在这里给出其算法的框架并简单的介绍。

算法 2.6 给出了伪码描述。算法最终生成的 c 就是要找的短句。算法的基本思想是首先尝试从短句 c 中随机去掉一个文字，然后探测去掉文字之后的短句 $\hat{c}$ 相对于 P 是否是归纳的，如果是归纳的，那么算法将更新 c 为 $\hat{c}$。算法继续查找新生成的短句 c 的归纳子句。否则，如果短句 $\hat{c}$ 相对于 P 不是归纳的，那么必定存在满足 $\hat{c}$ 的一个状态，设为 s，可以到达 $\neg\hat{c}$。此时，算法将取 $\neg\hat{c}$ 和 s 中公共的文字组成 q，令 $\hat{c} = \neg q$。该过程的本质是找到了使得 $\hat{c}$ 不归纳的原因 s，那么算法将试图将 s 从 $\hat{c}$ 中排除掉，但算法又要求生成的新的 $\hat{c}$ 必须是 c 的子句，因此，算法取 $\neg\hat{c}$ 和 s 的公共文字组成 q，q 事实上代表了新的状态集合，其中包含 s 和 $\neg\hat{c}$，且它的非是 c 的子句。

算法继续探测 $\hat{c}$ 是否归纳，直到 $\hat{c}$ 中包含一个初始状态，说明短句 c 去掉文字 l 后，没有归纳子句存在，因此，算法尝试去掉其他的文字，迭代继续，直到所有文字尝试完毕，算法结束。文献 [145] 给出了算法的正确性证明，本文不再详细阐述，感兴趣的读者，参考文献 [145]。

算法 2.6: 最小归纳子句生成算法

输入: $M = \langle S, I, T\rangle$, P, inductive clause c

输出: c

```
1  begin
2      for l ∈ c do
3          ĉ := c/l;
4          while true do
5              if I ↛ ĉ then
6                  Goto 2;
7              if P ∧ ĉ ∧ T → ĉ′ then
8                  c := ĉ;
9                  Goto 2;
10             else
11                 extract state s form P ∧ ĉ;
12                 q = s ⊔ ¬ĉ;
13                 ĉ := ¬q;
```

2.2.7.2 PDR 算法

算法 2.7 给出了 PDR 的伪码描述。

PDR 算法主要维护一个逼近序列 $F_0, F_1, \ldots, F_k$，并且对于任意的 $F_i(i > 0)$，其都是 P 和短句组成的集合。逼近序列 $F_0, F_1, \ldots, F_k$ 在算法迭代的过程中满足下面四个不变式:

- $F_0 = I$;
- $F_i \rightarrow F_{i+1}$;
- $F_i \wedge T \rightarrow F_{i+1}^{'}$;
- $F_i \rightarrow P, 0 \leq i \leq k$.

直观的讲，逼近序列 F_i 就是迁移系统 M 的 i 步可达的状态空间的上逼近，即，系统 M 中 i 步可达的状态集合都包含在 F_i 中。

算法的迭代过程分为两部分，首先是算法的增强阶段，也就是算法中的 $Strengthen(F_k)$。它显式的去构造一条“反例”，也就是不停的探测当前的逼近状态集合经过一步迁移是否可以到达“坏”状态。如果可以到达坏状态，那么提取 F_k 中可以一步迁移到坏状态的那些状态，利用最小归纳短句生成算法得到相对于 F_k 成归纳的短句，通过这个归纳短句就排除掉了可达坏状态的状态。如果待验证的系统 M 是不满足性质 P 的，那么在该阶段，一定可以构造一条反例路径。

算法 2.7: PDR 算法框架

输入: M, P

输出: safe 或者 counterexample

```
1  begin
2      k := 1;
3      F_0 = I, F_1 = P;
4      while true do
5          create F_{k+1};
6          /* 增强阶段 */
7          if ¬Strengthen(F_k) then
8              return counterexample;
9          /* 传播阶段 */
10         if propogate() then
11             return safe;
12         k++;
```

算法的第二部分是传播阶段，也就是算法中的 $propogate()$。在传播阶段，对于任意的 i，算法会测试 F_i 中的每一个短句是否可以在不违反前面四条不变式的条件下传播到 F_{i+1}，如果可以传播，则可以进一步收紧可达状态空间信息，这传播的过程中一旦发现存在某个 $j \leq k$，使得 $F_j = F_{j+1}$，算法就找到了一个不动点，算法终止，并回答性质正确。

下面我们分两个模块分别介绍这两个过程。首先给出增强阶段 $Strengthen(F_k)$ 的算法。

算法的增强阶段的主要目标就是排除掉当前状态空间逼近 F_k 中所有经过一步迁移可达坏状态的状态。首先探测 $F_k \wedge T \wedge \neg P'$ 是否成立，若成立，说明存在 F_k 中的状态经过一步迁移可达 $\neg P$。此时，算法生成一个任务对 $\langle s, k-1 \rangle$ 加入到任务优先级队列 Q 中，直观上讲，任意任务对 $\langle s, i \rangle$ 的意义就是要证明 F_i 中的所有状态经过一步迁移不能到达状态 s。

接下来，算法按照优先级处理任务序列 Q 中的任务。对于任意的任务 $\langle s, i \rangle$，首先检查 $\neg s$ 相对于 F_i 是否为归纳的，即 $F_i \wedge \neg s \wedge T \wedge s'$ 是否可满足，若不可满足，说明 $\neg s$ 相对于 F_i 是归纳的，即不存在 F_i 中的状态经过一步迁移可达 s。此时，增强算法将调用最小归纳子句生成算法 MIC（即算法 2）生成归纳子句 c。归纳子句 c 排除了包含 s 的状态集合。然后将短句 c 加入到所有的 F_j $(0 < j \leq i+1)$ 中。

算法 2.8: 增强算法

输入: F_k

输出: $true$ 或者 $false$

```
1  begin
2      while F_k ∧ T ∧ ¬P' is SAT do
3          extract state s ∈ F_k;
4          priorityQueue Q := ⟨s, k − 1⟩;
5          while Q is not empty do
6              pop some obligation ⟨s, i⟩;
7              if F_i ∧ ¬s ∧ T ∧ s' is UNSAT then
8                  c := MIC(M, F_i, ¬s);
9                  for 0 < j ≤ i + 1 do
10                     F_j := F_j ∪ {c};
11                 if i < k then
12                     Q := Q ∪ {⟨s, i + 1⟩};
13             else if i = 0 then
14                 return false;
15             else
16                 extract a predecessor t of s;
17                 Q := Q ∪ {⟨t, i − 1⟩};
18     return true;
```

若公式 $F_i \wedge \neg s \wedge T \wedge s'$ 是可满足的，说明存在 F_i 中的状态经过一步迁移可达状态 s，设这个状态为 t，如果此时 $i = 0$，说明算法构造出了一条从初始状态出发可达 $\neg P$ 的路径，即反例路径，算法终止并返回 $false$；否则，算法将生成新的任务 $\langle t, i-1\rangle$，试图从 F_i 中排除掉状态 t。

增强算法的另一个终止条件就是当 $F_k \wedge T \wedge \neg P'$ 是 UNSAT 时，说明 F_k 中不包含一步可达坏状态的状态了，此时，算法终止，并返回 $true$。

PDR 算法的另外一个主要过程称为传播过程，传播算法的本质是进一步收缩已知的逼近空间，在收缩的过程中，一旦探测到两个相邻状态空间相等，则算法找到了一个迭代不动点，算法终止。算法2.9 给出了函数 $propogate()$ 的基本描述。

以上就是整个基本 PDR 算法的执行过程。PDR 算法是进 10 年来符号化模型检验领域最大的进展[110]。它提出了一种全新的符号化的探索状态空间的方式。

算法 2.9: 传播算法

输入: M, P

输出: $true$ 或者 $false$

```
1 begin
2     for 0 ≤ i ≤ k do
3         for each c ∈ F_i/F_{i+1} do
4             if F_i ∧ T ∧ ¬c' is UNSAT then
5                 F_{i+1} = F_{i+1} ∪ {c}
6         if F_i = F_{i+1} then
7             return true;
8     return false;
```

该技术最大优点就是它结合了基于 SAT 的向前搜索的优点和向后搜索的优点。整个 PDR 算法的执行过程中不需要对迁移关系进行展开从而避免了 BMC 技术中 SAT 问题过大导致的无法求解的问题。而 PDR 算法又利用了向后搜索算法中属性制导的优点。因此，PDR 算法是目前模型检验领域的热点技术。本文提出的两个可达性算法都是基于 PDR 算法的。

2.3 本章总结

本章主要给出了本文所涉及的基本概念和基本算法。

本章的第 2.1节主要给出了基本概念。首先给出了字、布尔公式、非确定的 ω-自动机等基本定义，随后在第 2.1.3节中给出了本文研究内容涉及的三种时序逻辑（LTL，ETL，QTL）的语法和语义。接着，第 2.1.4节给出了迁移系统以及迁移系统的布尔编码。有了上面基本定义，在第 2.1.5 节，我们给出了线性时序逻辑模型检验问题的形式化定义。第 2.2节主要给出了几种与本文工作相关的算法。2.2.1节给出了 SAT 求解器的 DPLL 算法框架。第 2.2.2节至第 2.2.7分别介绍了几种基于 SAT 的模型检验算法。

第三章　优化的 LTL 限界模型检验技术

如前所述，LTL 的模型检验的复杂度为 $\mathcal{O}(|M| \times 2^{|\varphi|})$，因此，无论是基于自动机的显式模型检验技术还是基于 BDD 的符号化模型检验技术，LTL 的验证复杂度都与其公式长度成指数级关系。因此，对于给定的系统模型 M 和待验证的 LTL 公式 φ，若能在模型检验算法验证之前对 LTL 公式进行轻量式的化简，这将显著的提高模型检验算法的效率并大大的减少模型检验算法的时空开销。本章将首先介绍一种轻量级的 LTL 化简技术，该技术的核心思想就是在验证给定的 LTL 公式之前，先根据给定的模型 M 对其进行化简，尽量缩短待验证公式的长度，该技术的时空开销非常小，在一般的验证问题中基本上可以忽略不计，但是，一旦待验证的公式可以进行化简，那么这将极大的提高模型检验算法的验证效率。

LTL 的限界模型检验编码技术分两种：在本文的第 2.2.2节，我们给出了其中一种编码技术，这也是最初的 BMC 编码的技术，该编码技术是基于 LTL 的语法展开的，一般称这种编码方式为**基于语法的编码**；还有一种编码技术，是 Clarke 等人在文献 [86] 中提出的，该技术的核心思想是首先将 LTL 的模型检验问题转换成有穷自动机的语言判空问题，本质上就是公平路径查找问题，然后再根据给定的界值 k 将该问题编码成特定的布尔公式，最后利用 SAT 求解器求解 LTL 公式在给定的模型 M 上 k 步之内是否存在反例路径。一般的，称这种编码方式为 **基于语义的编码**。增量式求解 (Incremental SAT technology) 技术是 SAT 中一种非常重要的优化技术。本文将结合增量式求解技术给出一种优化的基于语义的编码。

3.1　研究动机

回顾 LTL 的符号化模型检验算法，给定迁移系统 M，LTL 性质 $\varphi = \mathsf{F}(p\mathsf{U}q)$ (p 和 q 是原子命题)。如果直接对公式 φ 做模型检验，首先要构造公式 $\neg\varphi$ 的 tableau $\mathcal{T}_{\neg\varphi}$，由于公式 φ 的长度为 4，因此，得到的乘积状态空间最多将会达到 $|M| \times 2^4$ 个状态。事实上，公式 $\mathsf{F}(p\mathsf{U}q)$ 与公式 $\mathsf{F}q$ 逻辑上是等价的。这是由于满足公式 $p\mathsf{U}q$ 的路径一定满足公式 $\mathsf{F}q$，因此，满足 $\mathsf{F}(p\mathsf{U}q)$ 的路径一定满足 $\mathsf{FF}q$，而满足 $\mathsf{FF}q$ 一定满足 $\mathsf{F}q$；反过来，满足 q 的路径一定会满足 $p\mathsf{U}q$，因此，满足 $\mathsf{F}q$ 的路径一定满足 $\mathsf{F}(p\mathsf{U}q)$。因此，对公式的 $\mathsf{F}(p\mathsf{U}q)$ 的验证可以直接使用公式 $\mathsf{F}q$ 来代替，其结果是一致的。而由于公式 $\mathsf{F}q$ 的长度为 2，因此，得到的乘积状态空间最多会到达 $|M| \times 2^2$，这极大的缩减了算法需要搜索的状态空间。

事实上，在真正的验证中，经常会出现一些可以进一步化简的规约，这是由于：

- 有些看似比较简单的规约还可以进一步化简。类似 $\mathsf{F}(p\mathsf{U}q)$ 的规约，看似已经非常简单，但是验证人员往往忽略掉了它还可以进一步化简。
- 在真正的工程应用中，确实存在一些比较复杂的规约描述，而这些规约往往是可以进一步化简的，如文献 [146] 中提到的一些工业中真实案例。
- 由于自动规约生成工具的出现，有些规约性质并不是人工写的，而是自动工具根据系统需求描述自动生成的，如工具 ProSPEC[147]，而这些规约通常具有很大的化简空间。

因此，对于给定的迁移系统和 LTL 公式，在模型检验算法开始验证之前，对 LTL 公式先进行化简将会很有必要。本文正是基于这点观察提出了一种轻量级的反例保持的化简技术（CounterExample Preserving Reduction，简称 CePRe）。

BMC 的效率跟所使用的编码技术有很大的关系，从广义上分，BMC 编码分两种：

- **基于语法的编码：**这种编码方法是根据 LTL 公式的语法结构，递归的进行布尔编码。本文第2.2.2节介绍的 BMC 编码就是最原始的基于语法的编码技术。
- **基于语义的编码：**这种编码方法类似于基于 BDD 的符号模型检验技术，它首先要求构造 tableau $\mathcal{T}_{\neg\varphi}$，然后构造模型和 $\mathcal{T}_{\neg\varphi}$ 的乘积 $M\otimes\mathcal{T}_{\neg\varphi}$，将 $M\otimes\mathcal{T}_{\neg\varphi}$ 在 k 步之内是否存在公平路径的问题编码成布尔公式。

基于语义编码复杂度非常依赖于 tableau 构造的复杂度，因此，在进行基于语义的 BMC 编码之前，先利用 CePRe技术对公式化简会减小基于语义编码的规模。

提高 BMC 效率的途径有两种：一是提供更好的编码技术；二是使用先进的 SAT 求解技术。增量式的 SAT 求解技术是目前 SAT 求解领域非常重要的优化技术。它的基本思想是利用已经求解过的结果对下一次 SAT 求解问题进行优化，本质是利用一些缓存的信息对 SAT 搜索的状态空间进行剪枝。增量式求解技术非常适用于结构化明显的 SAT 问题，而基于语义的 BMC 编码就是一种结构化非常明显的 SAT 问题。虽然文献 [86] 最先给出了基于语义的编码技术，但是根据其编码结构，它还有很多可以优化的空间，尤其是结合增量式 SAT 求解技术，还可以使编码结构更好，更利于 SAT 求解器进行求解。

3.2 反例保持的 LTL 化简技术

3.2.1 基本规则

定义 3.1: 给定迁移系统 M 以及 LTL 公式 φ，定义反例集合$\mathrm{CE}(\varphi, M)$，

$$\mathrm{CE}(\varphi, M) \stackrel{\text{def}}{=} \{\pi \in \mathscr{L}(M) \mid \pi \not\models \varphi\} \tag{3.1}$$

事实上，反例集合 $\mathrm{CE}(\varphi, M)$ 就是模型 M 中违反 φ 的路径集合。给定模型 M，若 LTL 公式 φ 和 ψ 相对于 M，有相同的反例集合，即 $\mathrm{CE}(\varphi, M) = \mathrm{CE}(\psi, M)$，则称 φ 和 ψ 相对于 M 是互相可化简的，记为 $\varphi \approx_M \psi$。由定义可知，若 $\varphi \approx_M \psi$，则 $M \models \varphi$ 当且仅当 $M \models \psi$。

给定模型 M 和 LTL 公式 φ，CEPRE技术的核心就是一些规则，使用这些规则来挖掘与公式 φ 互相可化简的公式 ψ 且使得验证 ψ 的复杂度要比验证 φ 的复杂度低。

本节将使用如公式3.2 所示的形式来描述 CEPRE技术中的规则，其中，称 Cond 为**附加条件**。

$$\mathrm{Cond} \rhd \varphi \approx_M \psi \quad (\textsc{Name}) \tag{3.2}$$

在公式3.2 中，尽管运算符 $\approx_M$ 是对称的，我们习惯上都是将待化简的公式放在 $\approx_M$ 的左边，而化简后的公式放在 $\approx_M$ 的右边。在上下文环境中，模型 M 固定的情况下，通常将 $\approx_M$ 简写为 $\approx$。在附加条件为 $\top$ 的情况下，简记 $\top \rhd \varphi \approx \psi$ 为 $\varphi \approx \psi$，我们称这样的规则为**模型无关规则**，否则称为**模型相关规则**。

本节将介绍 CEPRE技术中的一些基本规则，根据这些规则很容易派生出其他的化简规则。由于在符号化表示中，给定模型 $M = \langle I, T, \mathcal{C} \rangle$，它的各个组件都是通过一组布尔公式表示的。容易证明，给定模型 M，图 3.1 给出的几条基本规则一定成立。其中，“$\vdash$”是命题逻辑中可证关系，即：对于任意的命题公式 φ 和 ψ，若 $\varphi \vdash \psi$ 当且仅当 $\varphi \wedge \neg\psi$ 是不可满足的。

$$I \vdash \theta \ \rhd\ \theta \approx \top \ (\textsc{Init}) \qquad T \vdash \theta \ \rhd\ \mathsf{G}\theta \approx \top \ (\textsc{Trans})$$

$$\theta \in \mathcal{C} \ \rhd\ \mathsf{GF}\theta \approx \top \ (\textsc{Fair}) \qquad I \vdash \theta;\ T \vdash \theta \to \theta' \ \rhd\ \mathsf{G}\theta \approx \top \ (\textsc{Ind})$$

图 3.1　基本消减规则

定义一元时序连接子的偏序关系如公式 3.3 所示。

$$\begin{gathered} \mathsf{F} \sqsubseteq \mathsf{GF} \sqsubseteq \mathsf{FG} \sqsubseteq \mathsf{G} \\ \mathsf{F} \sqsubseteq \mathsf{X}^i \sqsubseteq \mathsf{G} \quad (i \prec \omega) \\ \mathsf{O} \sqsubseteq \mathsf{HO} \sqsubseteq \mathsf{OH} \sqsubseteq \mathsf{H} \end{gathered} \tag{3.3}$$

其中，$\mathsf{X}^0\varphi \stackrel{\mathrm{def}}{=} \varphi$，$\mathsf{X}^{i+1}\varphi \stackrel{\mathrm{def}}{=} \mathsf{X}(\mathsf{X}^i\varphi)$。

设 $\mathsf{P}^w,\ \mathsf{P}^s \in \{\mathsf{F}, \mathsf{FG}, \mathsf{GF}, \mathsf{G}, \mathsf{O}, \mathsf{HO}, \mathsf{OH}, \mathsf{H}\} \cup \{\mathsf{X}^i \mid i \prec \omega\}$ 且 $\mathsf{P}^w \sqsubseteq \mathsf{P}^s$，那么关于 P^w 和 P^s 就有两条模型无关的规则，如图3.2 所示。

$$(\mathrm{P}^{w}\varphi \wedge \mathrm{P}^{s}\varphi) \approx \mathrm{P}^{s}\varphi \ \text{(Conj)} \qquad (\mathrm{P}^{w}\varphi \vee \mathrm{P}^{s}\varphi) \approx \mathrm{P}^{w}\varphi \ \text{(Disj)}$$

图 3.2　Conj和 Disj化简规则

图3.3 提供了一组用于化简嵌套的将来时序连接子的规则。

$$\mathsf{F}(\varphi \mathsf{U} \psi) \approx \mathsf{F}\psi \ \text{(FU)} \qquad \varphi \mathsf{U}(\mathsf{F}\psi) \approx \mathsf{F}\psi \ (\mathrm{U_F})$$

$$\mathsf{FF}\varphi \approx \mathsf{F}\varphi \ \text{(FF)} \qquad \mathsf{GFG}\varphi \approx \mathsf{FG}\varphi \ \text{(GFG)}$$

图 3.3　将来时序连接子的化简规则

对于线性时序逻辑的模型检验，其目标是对于任意的路径 $\pi \in \mathscr{L}(M)$，验证 $\pi, 0 \models \varphi$ 是否成立，即：验证路径的起始点是否满足给定的性质。因此，对于任意的一个时序逻辑规约，如果其最外层时序连接子是过去时序连接子，那么这个连接子在模型检验过程中是没有实际意义的，因此，我们可以得到图3.4 所示的一组规则。

$$\mathsf{Y}\varphi \approx \bot \ \text{(Y)} \qquad \mathsf{O}\varphi \approx \varphi \ \text{(O)} \qquad \varphi \mathsf{S} \psi \approx \varphi \ \text{(S)}$$

图 3.4　过去时序连接子（最外层）化简规则

图3.3 和图3.4 分别提供了用于化简将来时序连接子和过去时序连接子的规则，图3.5 则提供了一组用于化简相邻时序连接子分别为将来时序连接子和过去时序连接子的规则。

设 $\theta_1, \theta_2, \ldots$ 是命题变元 AP 上的布尔公式，$\varphi_1, \varphi_2, \ldots$ 是任意的 LTL 公式，那么我们得到三组模型相关的规则。第一组如图3.6 所示，这是一组比较简单的规则。

当一个 LTL 公式中连续出现多个 U时序连接子时，公式构造 tableau 的复杂度将指数级的增长，因此，如果利用模型中的信息，对其进行化简可以去掉一个或多个 U连接子将极大的消减 tableau 构造的复杂度。比如，对于公式 $(\theta_1 \mathsf{U} \varphi)\mathsf{U}\theta_3$，若已知 $T \vdash \theta_1 \rightarrow \theta_3$ 成立，那么容易证明 $M \models (\theta_1 \mathsf{U} \varphi)\mathsf{U}\theta_3$ 当且仅当 $M \models \varphi \mathsf{U} \theta_3$。图3.7 给出了一组类似于这样的规则。

混合使用 U时序连接子和 R 时序连接子时，同样存在一组模型相关的规则，如图3.8 所示。

$$\mathsf{XY}\varphi \approx \varphi \ \text{(XY)} \qquad \mathsf{FH}\varphi \approx \mathsf{H}\varphi \ \text{(FH)}$$

$$\mathsf{FO}\varphi \approx \mathsf{F}\varphi \vee \mathsf{O}\varphi \ \text{(FO)} \qquad \mathsf{F}(\varphi \mathsf{S} \psi) \approx \mathsf{F}\psi \vee \varphi \mathsf{S} \psi \ \text{(FS)}$$

图 3.5　相邻将来和过去时序连接子的化简规则

$$T \vdash \theta_1 \vee \theta_2 \ \rhd \ \theta_1 \mathsf{U} \theta_2 \approx \mathsf{F}\theta_2 \ \text{(U)}$$

$$T \vdash \theta_2 \rightarrow \theta_1 \vee \theta_2' \ \rhd \ \theta_1 \mathsf{R} \theta_2 \approx \theta_2 \ \text{(R)}$$

图 3.6　化简规则 (U) 和 (R)

3.2.2　派生原则

事实上，根据3.2.1 中的基本规则，很容易得到更多的规则。本节将介绍 4 种用于生成新规则的派生原则。

$$T \vdash \theta_1 \rightarrow \theta_2 \ \rhd \ \theta_1 \mathsf{U} \theta_2 \approx \theta_2 \qquad (\mathrm{U}[1 \rightarrow 2])$$

$$T \vdash \theta_1 \rightarrow \theta_3 \ \rhd \ (\theta_1 \mathsf{U} \varphi_2) \mathsf{U} \theta_3 \approx \varphi_2 \mathsf{U} \theta_3 \qquad (\mathrm{U}^{\mathrm{U}}[1 \rightarrow 3])$$

$$T \vdash \theta_2 \rightarrow \theta_3 \ \rhd \ (\varphi_1 \mathsf{U} \theta_2) \mathsf{U} \theta_3 \approx \theta_3 \vee (\varphi_1 \mathsf{U} \theta_2) \qquad (\mathrm{U}^{\mathrm{U}}[2 \rightarrow 3])$$

$$T \vdash \theta_3 \rightarrow \theta_2 \ \rhd \ (\varphi_1 \mathsf{U} \theta_2) \mathsf{U} \theta_3 \approx (\varphi_1 \vee \theta_2) \mathsf{U} \theta_3 \qquad (\mathrm{U}^{\mathrm{U}}[3 \rightarrow 2])$$

$$T \vdash \theta_2 \rightarrow \theta_3' \ \rhd \ (\varphi_1 \mathsf{U} \theta_2) \mathsf{U} \theta_3 \approx (\varphi_1 \vee \theta_2) \mathsf{U} \theta_3 \qquad (\mathrm{U}^{\mathrm{U}}[2 \rightarrow 3'])$$

$$T \vdash \neg\theta_2 \rightarrow \theta_3 \ \rhd \ (\varphi_1 \mathsf{U} \theta_2) \mathsf{U} \theta_3 \approx \mathsf{F}\theta_3 \qquad (\mathrm{U}^{\mathrm{U}}[\neg 2 \rightarrow 3])$$

$$T \vdash \theta_1 \rightarrow \theta_2 \ \rhd \ \theta_1 \mathsf{U} (\theta_2 \mathsf{U} \varphi_3) \approx \theta_2 \mathsf{U} \varphi_3 \qquad (\mathrm{U}_{\mathrm{U}}[1 \rightarrow 2])$$

$$T \vdash \theta_1 \rightarrow \theta_3 \ \rhd \ \theta_1 \mathsf{U} (\varphi_2 \mathsf{U} \theta_3) \approx \varphi_2 \mathsf{U} \theta_3 \qquad (\mathrm{U}_{\mathrm{U}}[1 \rightarrow 3])$$

$$T \vdash \theta_2 \rightarrow \theta_1 \ \rhd \ \theta_1 \mathsf{U} (\theta_2 \mathsf{U} \varphi_3) \approx \theta_1 \mathsf{U} \varphi_3 \qquad (\mathrm{U}_{\mathrm{U}}[2 \rightarrow 1])$$

图 3.7　相邻 U时序连接子的化简规则

$$T \vdash \theta_1 \to \theta_3 \ \rhd\ (\theta_1 \mathsf{R} \varphi_2)\mathsf{U}\theta_3 \approx ((\theta_1 \mathsf{R}\varphi_2) \vee \theta_3) \wedge \mathsf{F}\theta_3 \qquad (\mathrm{U}^{\mathrm{R}}[1 \to 3])$$

$$T \vdash \neg\theta_1 \to \theta_3 \ \rhd\ (\theta_1 \mathsf{R} \varphi_2)\mathsf{U}\theta_3 \approx \varphi_2 \mathsf{U}\theta_3 \qquad (\mathrm{U}^{\mathrm{R}}[\neg 1 \to 3])$$

$$T \vdash \theta_1 \to \theta_3 \ \rhd\ \theta_1 \mathsf{U}(\varphi_2 \mathsf{R}\theta_3) \approx \varphi_2 \mathsf{R}\theta_3 \qquad (\mathrm{U}_{\mathrm{R}}[1 \to 3])$$

图 3.8　相邻 U和 R的化简规则

3.2.2.1　反转原则

我们称时序连接子对：‘U和 S’、‘R和 T’、‘G和 H’ 以及 ‘F和 O’ 是“**相反的**”时序连接子。注意，对于时序连接子 X，它对应的相反的时序连接子可能有两个，分别为 Z和 Y，根据上下文环境的不同，其对应的相反的时序连接子也不相同。在本文中提到的规则中并没有涉及这种情况，因此不再详细讨论。

$$\frac{T \vdash \theta \ \rhd\ \varphi \approx \psi}{T \vdash \tilde{\theta} \ \rhd\ \mathsf{inv}(\varphi) \approx \mathsf{inv}(\psi)} \tag{3.4}$$

反转原则是如公式3.4 描述的规则，其中：

- 若 θ 是 AP 上的布尔公式，则 $\tilde{\theta} = \theta$；若 θ 是 $AP \cup AP'$ 上的布尔公式，则 $\tilde{\theta}$ 是将 θ 中当前状态变量和下一刻状态变量互换得到的。
- $\mathsf{inv}(\varphi)$ 和 $\mathsf{inv}(\psi)$ 分别是将 φ 和 ψ 中的时序连接子换成其相反的时序连接子得到的公式。

例如，对于 (U) 和 (R) 规则，使用反转规则很容易得到规则 (S) 和 (T)，如公式3.5 所示。

$$\begin{aligned} &T \vdash \theta_1 \vee \theta_2 \ \rhd\ \theta_1 \mathsf{S}\theta_2 \approx \mathsf{O}\theta_2 \qquad (\mathrm{S}) \\ &T \vdash \theta_2' \to \theta_1' \vee \theta_2 \ \rhd\ \theta_1 \mathsf{T}\theta_2 \approx \theta_2 \quad (\mathrm{T}) \end{aligned} \tag{3.5}$$

同样，对于图3.3、图3.7 以及图3.8 中的公式使用反转规则，也可以得到对应的化简规则，在这里不再一一列举。

3.2.2.2　对偶原则

通常，称操作子对：‘$\top$ 和 $\bot$’、‘$\wedge$ 和 $\vee$’、‘F和 G’、‘O和 H’、‘Y和 Z’、‘X和 X’、‘U和 R’ 以及 ‘T和 S’ 为“**对偶**”操作子。

$$\frac{\mathrm{Cond} \ \rhd\ \varphi \approx \psi}{\overline{\mathrm{Cond}} \ \rhd\ \mathsf{dual}(\varphi) \approx \mathsf{dual}(\psi)} \tag{3.6}$$

对偶原则是如公式3.6 所示的规则。在使用对偶原则，对附加条件 Cond 是有要求的，它要求 Cond 为 $\top$（说明是模型无关规则，可以省略）或者为形如 $T \vdash \theta_1 \to \theta_2$ 的公式。这样得到 $\overline{\text{Cond}}$ 就是 $\top$（省略）或者 $T \vdash \theta_2 \to \theta_1$。公式 $\mathsf{dual}(\varphi)$ 和 $\mathsf{dual}(\psi)$ 分别是将公式 φ 和 ψ 中的时序连接子替换成对应的对偶连接子得到的公式。

例如，对于规则 (FU)、(U_F)、(FF) 以及 (GFG) 使用对偶原则，可以立即得到如图3.9 所示的一组规则。

$$\mathsf{G}(\varphi\mathsf{R}\psi) \approx \mathsf{G}\psi \ \ \text{(GR)} \qquad\qquad \varphi\mathsf{R}(\mathsf{G}\psi) \approx \mathsf{G}\psi \ \ (\text{R}_\text{G})$$

$$\mathsf{GG}\varphi \approx \mathsf{G}\varphi \ \ \text{(GG)} \qquad\qquad \mathsf{FGF}\varphi \approx \mathsf{GF}\varphi \ \ \text{(FGF)}$$

图 3.9　(GR)、(R_G)、(GG) 以及 (FGF) 化简规则

注意，将对偶原则应用到模型相关的规则时，它不仅需要替换对应的对偶连接子，还要将附加条件中关于蕴含操作符（$\to$）的前项和后项条件交换过来。例如，使用对偶原则可以从规则 ($\text{U}^\text{U}[2 \to 3]$) 得到如公式3.7 所示的规则 ($\text{R}^\text{R}[3 \to 2]$)。

$$T \vdash \theta_3 \to \theta_2 \ \ \rhd \ \ (\varphi_1\mathsf{R}\theta_2)\mathsf{R}\theta_3 \approx \theta_3 \wedge (\varphi_1\mathsf{R}\theta_2) \qquad (\text{R}^\text{R}[3 \to 2]) \tag{3.7}$$

3.2.2.3　组合原则

组合原则的描述如公式3.8 所示。

$$\frac{\begin{array}{rcl}\text{Cond}_1 & \rhd & \varphi_1 \approx \psi_1 \\ \text{Cond}_2 & \rhd & \varphi_2 \approx \psi_2\end{array}}{\text{Cond}_1\&\text{Cond}_2 \ \ \rhd \ \ \varphi_1 \approx \psi_1{}^{\varphi_2}_{\psi_2}} \tag{3.8}$$

公式3.8 中，若公式 φ_i 和 $\psi_i (i = 1, 2)$ 是互相可化简的，那么公式 φ_1 和 $\psi_1{}^{\varphi_2}_{\psi_2}$ 是互相可化简的。在使用组合原则时，需要主要以下几点：

- 附加条件 Cond_2 必须与初始条件 I 无关。
- 如果公式中的第二条规则是图3.4 中规则，那么需要 ψ_1 中的所有 φ_2 的出现是不受任何时序逻辑连接子约束的。
- 由于 $\eta \approx \eta$ 是成立的，因此，公式3.9 可以认为是组合原则的一种特殊情况。

$$\frac{\text{Cond} \ \ \rhd \ \ \varphi \approx \psi}{\text{Cond} \ \ \rhd \ \ \eta \approx \eta^{\varphi}_{\psi}} \tag{3.9}$$

举个例子，给定 LTL 公式 $(\theta_1 \mathsf{U} \theta_2)\mathsf{U}\theta_3$ 和公式 $\theta_2 \mathsf{U}\theta_3$，若有 $\text{Cond}_1 = \theta_1 \to \theta_3$ 以及 $\text{Cond}_2 = \theta_2 \to \theta_3$ 成立，那么根据组合规则，可以如下得到新的规则。

$$\frac{\begin{array}{rcl} T \vdash \theta_1 \to \theta_3 & \rhd & (\theta_1 \mathsf{U} \theta_2)\mathsf{U}\theta_3 \approx \theta_2 \mathsf{U}\theta_3 \\ T \vdash \theta_2 \to \theta_3 & \rhd & \theta_2 \mathsf{U}\theta_3 \approx \theta_3 \end{array}}{T \vdash (\theta_1 \to \theta_3)\&(\theta_2 \to \theta_3) \quad \rhd \quad (\theta_1 \mathsf{U} \theta_2)\mathsf{U}\theta_3 \approx \theta_3}$$

3.2.2.4 分解原则

分解原则可以描述如下:

$$\frac{\begin{array}{rcl} \text{Cond}_1 & \rhd & \varphi_1 \approx_{M_1} \psi_1 \\ \text{Cond}_2 & \rhd & \varphi_2 \approx_{M_2} \psi_2 \end{array}}{\text{Cond}_1 \& \text{Cond}_2 \quad \rhd \quad \varphi_1 \approx_{M_1 \otimes M_2} \psi_1{}^{\varphi_2}_{\psi_2}} \tag{3.10}$$

在使用分解原则时同样需要满足如组合原则中的条件。这里 $M_1 \otimes M_2$ 是模型 M_1 和 M_2 的乘积。

3.2.3 化简策略

我们先结合一个示例分析一下如何使用这些规则。给定模型 M 以及规约性质 $\varphi = (\theta_1 \mathsf{U} \theta_2)\mathsf{U}\theta_3$:

- 首先，我们试图使用规则 ($\mathrm{U}^{\mathrm{U}}[1 \to 3]$) 化简，调用一次 SAT 求解器判断 $T \vdash \theta_1 \to \theta_3$ 是否可永真，即: $T \wedge \theta_1 \wedge \neg\theta_3$ 是否可满足。
- 若 $T \wedge \theta_1 \wedge \neg\theta_3$ 是不可满足的，那么使用规则 ($\mathrm{U}^{\mathrm{U}}[1 \to 3]$) 得到公式 $\theta_2 \mathsf{U}\theta_3$。
- 否则，选取另外一条化简规则继续化简，如 ($\mathrm{U}^{\mathrm{U}}[2 \to 3]$)。

通常，组合使用这些规则效果会更好，比如，对于性质 $\mathsf{FO}p$ 可以如下化简:

- 首先使用 (FO) 规则，使得 $M \models \mathsf{FO}p$ 化简为 $M \models \mathsf{F}p \vee \mathsf{O}p$。
- 这时，对于子公式 $\mathsf{O}p$，其最外层时序连接子为过去时序连接子，按照规则 (O) 化简可得到 $M \models \mathsf{F}p \vee p$。
- 最后，使用 DISJ化简得到 $M \models \mathsf{F}p$。

在算法的真正实现中，我们是基于“**最大匹配**”的原则来选择优先使用哪一条规则进行化简的。具体过程如算法3.1 所示。在算法3.1 中，仅基本规则和由基本规则通过反转规则或对偶规则派生出来的规则可以直接使用。这保证了算法选取规则的终止性。

在算法的第12 行，规则“$\text{Cond} \rhd \psi \approx \eta$”的优先级是根据 Cond 和 η 确定的，Cond 和 η 越简单，优先级越高。因此，模型无关规则的优先级一定比模型相关规则的优先级高。

最后，根据算法的执行过程，很容易证明对于任意的公式，算法在 $\mathcal{O}(|\varphi|)$ 内会终止。

算法 3.1: 基于最大匹配原则的化简算法

输入: M, φ

输出: 化简后的 φ

```
1  begin
2      Γ := ∅;        /* Γ 用来存储不会化简的子公式 */
3      Δ := {ψ ∈ (sub(φ)\Γ), ψ满足一些化简规则 rule(s)};
4      foreach ψ1, ψ2 ∈ Δ s.t. ψ1 ≠ ψ2 do
5          if ψ1 ∈ sub(ψ2) then
6              Δ := Δ\{ψ1};        /* 我们仅化简“最大匹配”的子公式 */
7      if Δ = ∅ then
8          return φ;
9      foreach ψ ∈ Δ do
10         Θ := {可以化简φ的规则};        /* 对于任意的 ψ，要求 |Θ| ≤ 5 */
11         while Θ ≠ ∅ do
12             choose R := (Cond ▷ ψ ≈ η) in Θ;
13             if Cond is stated then
14                 φ := φ_η^ψ;        /* φ_η^ψ 是将公式 φ 中的 ψ 替换成 η 得到的公式 */
15                 break;
16             Θ := Θ\{R};
17         Δ := Δ\{ψ};
18         if Θ = ∅ then
19             Γ := Γ ∪ {ψ};        /* ψ 不会出现在下一次迭代中 */
20     goto 3;
```

3.2.4 性能分析

由于 CePRe是在模型检验算法执行之前对公式进行化简的技术，因此，CePRe技术可以结合各种模型检验算法使用，不仅仅局限于 BMC 算法。我们首先根据具体模型检验算法分析 CePRe技术的性能。这里我们主要关注三个算法：

- 基于 BDD 的符号化模型检验算法。
- 基于语法编码和基于语义编码的 BMC 算法。
- 属性制导的可达性分析算法（PDR）。

3.2.4.1 基于 BDD 的 CePRe性能分析

给定 LTL 公式 φ，基于 BDD 的 LTL 符号化模型检验技术的核心是构造公式 $\neg\varphi$ 的 tableau $\mathcal{T}_{\neg\varphi}$。下面我们逐个分析影响模型检验算法各个组件。

状态空间：根据文献 [34] 可知，决定状态空间大小的是基础公式集 $el(\varphi)$，基础公式集 $el(\varphi)$ 可归纳定义如下:

- $el(\top) = el(\bot) = \emptyset$。
- 若 $\varphi = p \in AP$，则 $el(p) = \{p\}$。
- 若 $\varphi = \neg\psi$，则 $el(\varphi) = el(\psi)$。
- 若 $\varphi = \varphi_1 \wedge \varphi_2$，则 $el(\varphi) = el(\varphi_1) \cup el(\varphi_2)$。
- 若 $\varphi = \mathsf{X}\psi$，则 $el(\varphi) = \{\mathsf{X}\psi\} \cup el(\psi)$。
- 若 $\varphi = \mathsf{Y}\psi$，则 $el(\varphi) = \{\mathsf{Y}\psi\} \cup el(\psi)$。
- 若 $\varphi = \varphi_1 \mathsf{U} \varphi_2$，则 $el(\varphi) = \{\mathsf{X}(\varphi_1 \mathsf{U} \varphi_2)\} \cup el(\varphi_1) \cup el(\varphi_2)$。
- 若 $\varphi = \varphi_1 \mathsf{S} \varphi_2$，则 $el(\varphi) = \{\mathsf{Y}(\varphi_1 \mathsf{S} \varphi_2)\} \cup el(\varphi_1) \cup el(\varphi_2)$。

在符号化的表示中，任意的 $\psi \in el(\varphi)$ 都要对应一个命题变量。若 $\psi \in AP$，则不需要引入新的变量，因为在构造模型 M 的过程中已经存在这个变量，否则需要引入新的变量 p_ψ，因此，构造 tableau 需要引入新变量的数为 $|el(\varphi)\backslash AP|$。根据公式结构归纳法，我们可以得到下面的命题。

命题 3.1: $|el(\varphi)\backslash AP|$ 等于 LTL 公式 φ 中时序连接子的数目。

迁移关系：$\mathcal{T}_{\neg\varphi}$ 中的迁移关系是一组约束的析取，其中每一个约束都是形如 $p_{\mathsf{X}\psi} \leftrightarrow (\lambda(\psi))'$ 或 $p'_{\mathsf{Y}\eta} \leftrightarrow \lambda(\eta)$ 的布尔公式，其中 $\mathsf{X}\psi, \mathsf{Y}\eta \in el(\varphi)$，函数 λ 可以如下定义:

- $\lambda(\bot) = \bot$，$\lambda(\top) = \top$。
- 若 $\varphi = p \in AP$，则 $\lambda(\varphi) = p$。
- 若 $\varphi = \neg\psi$，则 $\lambda(\varphi) = \neg\lambda(\psi)$。
- 若 $\varphi = \varphi_1 \wedge \varphi_2$，则 $\lambda(\varphi) = \lambda(\varphi_1) \wedge \lambda(\varphi_2)$。
- 若 $\varphi = \mathsf{X}\psi$，则 $\lambda(\varphi) = p_{\mathsf{X}\psi}$。
- 若 $\varphi = \mathsf{Y}\psi$，则 $\lambda(\varphi) = p_{\mathsf{Y}\psi}$。
- 若 $\varphi = \varphi_1 \mathsf{U} \varphi_2$，则 $\lambda(\varphi) = \lambda(\varphi_2) \vee \lambda(\varphi_1) \wedge p_{\mathsf{X}(\varphi_1 \mathsf{U} \varphi_2)}$。
- 若 $\varphi = \varphi_1 \mathsf{S} \varphi_2$，则 $\lambda(\varphi) = \lambda(\varphi_2) \vee \lambda(\varphi_1) \wedge p_{\mathsf{Y}(\varphi_1 \mathsf{S} \varphi_2)}$。

根据 $el(\varphi)$ 的定义可知，对于新产生的每一个变量 $p_{\mathsf{X}\psi}$ 或者 $p_{\mathsf{Y}\eta}$ 都会产生一个迁移关系的约束，因此，很容易得到下面的命题。

命题 3.2: 对于给定的 LTL 公式 φ，$\mathcal{T}_{\neg\varphi}$ 的迁移关系中约束的个数等于 φ 中时序连接子的个数（即 $|el(\varphi)\backslash AP|$）。

公平性约束：根据文献 [34] 中的 tableau 构造算法，对于任意子公式 $\psi \in sub_{\mathsf{U}}(\neg\varphi)$，都会产生一个对应的公平性约束 F_ψ。其中，$sub_{\mathsf{U}}(\neg\varphi)$ 是指公式 $\neg\varphi$ 中带 U 时序算子的子集。

逐项分析 CePRe中所提到的所有规则，可以得出定理3.1。

定理 3.1: 设“Cond $\triangleright$ $\varphi \approx \psi$”是CePRe中的一条化简规则，则 $|el(\psi)\backslash AP| \leq |el(\varphi)\backslash AP|$ 且 $|sub_{\mathsf{U}}(\psi)| \leq |sub_{\mathsf{U}}(\varphi)|$。

3.2.4.2 基于 BMC 的 CePRe性能分析

如前所述，BMC 的效率跟所使用的编码技术有很大的关系。由于基于语义的编码也要构造 tableau，因此，对于基于语义的编码技术，CePRe技术能够获取的好处跟基于 BDD 的技术是一样的。这里不再详细讨论。

对于基于语法的编码技术，由本文第2.2.2节可知，在给定界值 k 的情况下，BMC 的编码如公式3.11所示。

$$[\![M, \varphi]\!]_k := [\![M]\!]_k \wedge [\![\neg\varphi]\!]_k \tag{3.11}$$

其中，$[\![M]\!]_k$ 表示模型 M 的 k 步展开，$[\![\neg\varphi]\!]_k$ 表示在 k 步之内违反公式 φ 的条件。事实上，在所有的语法编码中，模型的 k 步展开的编码都是一样的，影响基于语法的 BMC 编码主要部分在于 $[\![\neg\varphi]\!]_k$ 的编码。

给定 LTL 公式 φ 的子公式，归纳定义 ψ 在模型 k 步展开上布尔编码的最大长度 $f([\![\psi]\!]_k)$:

- $f([\![\bot]\!]_k) = f([\![\top]\!]_k) = 0$。
- 若 $\psi = p \in AP$，则 $f([\![\psi]\!]_k) = 1$。
- 若 $\psi = \neg\psi_1$，则 $f([\![\psi]\!]_k) = f([\![\psi_1]\!]_k)$。
- 若 $\psi = \psi_1 \wedge \psi_2$，则 $f([\![\psi]\!]_k) = f([\![\psi_1]\!]_k) + f([\![\psi_2]\!]_k) + 1$。
- 若 $\psi = \mathsf{X}\psi_1$，则 $f([\![\psi]\!]_k) = f([\![\psi_1]\!]_k)$。
- 若 $\psi = \mathsf{Y}\psi_1$，则 $f([\![\psi]\!]_k) = f([\![\psi_1]\!]_k)$。
- 若 $\psi = \psi_1 \mathsf{U} \psi_2$，则 $f([\![\psi]\!]_k) = g(k) \times f([\![\psi_1]\!]_k) + k \times f([\![\psi_2]\!]_k)$。
- 若 $\psi = \psi_1 \mathsf{S} \psi_2$，则 $f([\![\psi]\!]_k) = g(k) \times f([\![\psi_1]\!]_k) + k \times f([\![\psi_2]\!]_k)$。

其中，根据编码技术的不同，$g(k)$ 的取值也不同。文献 [1][84][85] 提出的编码技术中 $g(k) \in \mathcal{O}(k^2)$。文献 [87][88] 中提出的编码技术中 $g(k) \in \mathcal{O}(k)$。无论是哪一种编码技术，尽量减少公式中的 U和 S都会在一定程度上减少编码的复杂度。

另外一个影响基于语法编码复杂度的因素是公式中的命题变元数目。若使用 $var_k(\varphi)$ 来表示 $[\![\neg\psi]\!]_k$ 中新增加的命题变元集合，我们可以得到如下结论:

- 文献 [1][84][85] 的编码技术不需要引入新的变量，因此 $|var_k(\varphi)| = 0$。
- 文献 [87][88] 的编码技术需要为每一个 U子公式和 R子公式引入 $\mathcal{O}(k)$ 个新的变量。

同样，逐项分析 CePRe中所提到的所有规则，可以得出定理3.2。

定理 3.2: 设“Cond $\triangleright$ $\varphi \approx \psi$”是CEPRE的化简规则，那么 $f([\![\psi]\!]_k) \leq f([\![\varphi]\!]_k)$ 且 $|var_k(\psi)| \leq |var_k(\varphi)|$。

3.2.4.3 基于 PDR 的 CEPRE性能分析

本文的第2.2.7节介绍了属性制导的可达性分析算法，即 PDR 算法。由于该算法是一种基于 SAT 的可达性分析技术，因此，它不能直接验证 LTL 性质，这里首先需要将 LTL 验证转换成可达性问题。给定模型 M 和 LTL 公式 φ，主要的转换步骤为:

- 首先要构造 tableau $\mathcal{T}_{\neg\varphi}$;
- 构造乘积 $M \otimes \mathcal{T}_{\neg\varphi}$，将 LTL 的验证问题转换为乘积模型 $M \otimes \mathcal{T}_{\neg\varphi}$ 上的公平路径查找问题。
- 将公平路径查找问题分解成若干个个可达性问题。

这样就可以使用 PDR 算法来验证 LTL 性质了。从上面的转换步骤可以看出，影响 PDR 时空开销的主要包含 3 方面:

- 乘积系统的状态空间，这是由 M 和 $\mathcal{T}_{\neg\varphi}$ 的变量数目确定的。
- 迁移关系大小，即 M 和 $\mathcal{T}_{\neg\varphi}$ 迁移关系合取。
- 公平性约束的个数，一般是 M 和 $\mathcal{T}_{\neg\varphi}$ 中公平性约束之和。

给定模型 M，如果两个 LTL 公式 φ 和 ψ 且 $\varphi \approx_M \psi$，由于 M 是不变的，因此影响乘积模型状态空间、迁移关系大小以及公平性约束的因素主要是两个 tableau: $\mathcal{T}_{\neg\varphi}$ 和 $\mathcal{T}_{\neg\psi}$。因此，CEPRE技术与 PDR 算法结合的优点如第3.2.4.1的结论一样，同样满足命题3.1 和3.2以及定理3.1。

3.3 增量式的基于语义的编码技术

3.3.1 基本的基于语义的编码

给定模型 M 和 LTL 公式 φ，基于语义的编码依赖于 tableau $\mathcal{T}_{\neg\varphi}$ 的构造。基于语义的编码主要有以下步骤:

- 首先构造 tableau $\mathcal{T}_{\neg\varphi}$。
- 构造乘积 $\mathcal{M} = M \otimes \mathcal{T}_{\neg\varphi}$。
- 编码模型 $\mathcal{M} = \langle I_{\mathcal{M}}, T_{\mathcal{M}}, \mathcal{C}\rangle$ 在 k 步之内是否存在公平路径的问题，如公式3.12所示。

$$[\![\mathcal{M}]\!]_k = I_{\mathcal{M}}[s_0] \wedge \bigwedge_{i=0}^{k-1} T_{\mathcal{M}}[s_i, s_{i+1}] \wedge \bigvee_{l=0}^{k-1} ({}_lL_k \wedge \bigwedge_{C_i \in \mathcal{C}} (\bigvee_{j=l}^{k} C_i[s_j])) \tag{3.12}$$

直观上讲，公式3.12可以两部分，前两个合取项是第一部分，它表示乘积模型 $\mathcal{M}$ 的 k 步展开。剩余的编码为第二部分，主要用来约束路径必须满足两个条件：1. 是一条带循环的路径，用 ${}_lL_k$ 来约束（见本文定义2.25）。2. 路径的循环部分必须满足所有的公平性约束，用 $\bigwedge_{C_i\in\mathcal{C}}(\bigvee_{j=l}^{k}C_i[s_j])$ 来约束。

与基于语法的编码相比，基于语义的编码有以下好处：

- 基于语义的编码可以利用 tableau 构造算法中的优化技术。比如，对于 LTL 性质 $\mathsf{FF}p$，若使用基于语法的编码，它产生的编码规模要比公式 $\mathsf{F}p$ 大的多。然而，事实上，性质 $\mathsf{FF}p$ 和 $\mathsf{F}p$ 是逻辑等价的，基于语义的编码在构造 tableau 的过程中将会把 $\mathsf{FF}p$ 化简成 $\mathsf{F}p$。本文的 CEPRE技术正是基于这一点来优化基于语义编码技术的。
- 基于语义的编码产生的新的变量个数是与 k 成线性关系的，这一点也是获益于 tableau 的构造，本章的第3.2.4.2 节已经分析了这个结论。
- 基于语义的编码相对于基于语法的编码，更容易计算完备阈值。这是由于基于语义的编码的公式结构与具体的 LTL 性质无关，其最终都是在一个有穷状态的迁移系统上查找公平路径，这就方便利用一些图上的性质来帮助计算完备阈值。

定理 3.3: *基于语义编码产生的变量数目的复杂度为* $\mathcal{O}(k\times(|AP|+|\varphi|))$。

3.3.2 增量式的基于语义的编码

增量式的基于语义编码的核心是尽量多的利用上一次 SAT 求解的结果。这就需要将基本的编码分解成两部分，一部分称为 k 无关编码，另一部分称为 k 相关编码。k 无关编码就是指，该部分的布尔公式不会随着 k 的变化而变化。k 相关编码则是指随着 k 的增长会产生变化的部分。显然，为了尽量多的利用信息，k 无关部分在整个编码中占的比例越大，则可利用的信息就越多。

因此，增量式的基于语义的编码就是通过分析原始编码的结构尽量多的分离出与 k 无关的布尔编码。

首先，分析基本的基于语义编码的模型展开部分。

$$I_{\mathcal{M}}[s_0]\wedge\bigwedge_{i=0}^{k-1}T_{\mathcal{M}}[s_i,s_{i+1}]$$

该部分的布尔编码显然是 k 无关的编码。这是因为，当 k 递增加 1 后，该部分的编码不会随着 k 的增加而变化，它仍是这个编码形式。

例如，当 $k=2$ 时，模型的 2 步展开如公式3.13 所示。

$$I_{\mathcal{M}}[s_0]\wedge T_{\mathcal{M}}[s_0,s_1]\wedge T_{\mathcal{M}}[s_1,s_2] \tag{3.13}$$

而当 $k=3$ 时，模型的 3 步展开如公式3.14 所示。

$$I_{\mathcal{M}}[s_0] \wedge T_{\mathcal{M}}[s_0,s_1] \wedge T_{\mathcal{M}}[s_1,s_2] \wedge T_{\mathcal{M}}[s_2,s_3] \tag{3.14}$$

观察公式3.13 和3.14 可知，公式 3.14 的前面和公式3.13 是完全一样的，仅仅多了一步展开 $T_{\mathcal{M}}[s_2,s_3]$，因此，在下一次求解时，相同的展开部分就不需要再次展开了，而仅仅把新添加的部分加入到 SAT 求解器中即可。

观察公式3.12 可知，在基于语义的编码中，公平性约束的编码占了整个编码的很大部分。因此，如果能将公平性约束部分的编码分解成 k 无关编码和 k 相关编码两部分将非常利于 SAT 求解。

为了将公平性约束的编码部分分解成 k 无关编码和 k 相关编码两部分，我们需要引入一些新的变量，这些变量用来辅助编码。为每一个公平性约束 $C_i \in \mathcal{C}$ 引入一个新的变量 w_i，令 W 是包含所有 w_i 的集合。同时，对任意的 $w_i \in W$ 以及 $C_i \in \mathcal{C}$，要求满足公式3.15。

$$\begin{array}{lcl} w_i^j & \leftrightarrow & C_i[s_j] \vee w_i^{j+1} \\ w_i^k & \leftrightarrow & C_i[s_k] \end{array} \tag{3.15}$$

引入变量集合 W 之后，再对路径约束部分进行编码，就可以得到公式3.16。

$$\begin{array}{ll} \bigwedge_{w_i \in W} \bigwedge_{j=0}^{k-1} (w_i^j \leftrightarrow C_i[s_j] \vee w_i^{j+1}) & (1) \\ \wedge & \\ \bigwedge_{w_i \in W} (w_i^k \leftrightarrow C_i[s_k]) \wedge \bigvee_{l=0}^{k} ({}_lL_k \wedge \bigwedge_{w_i \in W} (w_i^l)) & (2) \end{array} \tag{3.16}$$

公式3.16 分两部分：(1) 和 (2)。第 (1) 部分就是 k 无关的编码，第 (2) 部分则是 k 相关编码。编码里最关键的部分在于 w_i^j 的引入。根据公式3.15，可以将 w_i^j 展开如下：

$$w_i^j \leftrightarrow C_i[s_j] \vee C_i[s_{j+1}] \vee \ldots \vee C_i[s_k]$$

直观上讲，w_i^j 的主要作用就是用来约束在路径的循环体内至少有一个状态 s_m 需要公平性约束 C_j。

通过公式3.16 对路径约束部分进行编码的好处就是 k 无关编码可以重复利用。比如，假设模型 $\mathcal{M}$ 仅有 1 个公平性约束 C_1，当 $k=2$ 时，按照公式3.16 的编码方式，路径约束的编码为：

$$\begin{array}{ll} (w_1^0 \leftrightarrow C_1[s_0] \vee w_1^1) \wedge (w_1^1 \leftrightarrow C_1[s_1] \vee w_1^2) & (1) \\ \wedge (w_1^2 \leftrightarrow C_1[s_2]) \wedge \bigvee_{l=0}^{2} ({}_lL_2 \wedge w_1^l) & (2) \end{array} \tag{3.17}$$

当 $k=3$ 时，路径约束的编码为:

$$\begin{aligned}&(w_1^0 \leftrightarrow C_1[s_0] \vee w_1^1) \wedge (w_1^1 \leftrightarrow C_1[s_1] \vee w_1^2) \quad (1)\\ &\wedge(w_1^2 \leftrightarrow C_1[s_2] \vee w_1^3) \wedge (w_1^3 \leftrightarrow C_2[s_3]) \wedge \bigvee_{l=0}^{3}({}_lL_3 \wedge w_1^l) \quad (2)\end{aligned} \tag{3.18}$$

观察可知，公式3.17 和公式3.18 的第一部分编码是相同的，因此，在进行第 3 步编码时就可以利用第 2 步编码的相同部分。

通过引入变量集合 W，新的对路径约束的编码将很大部分转换成了 k 无关的编码。公式3.19 给出了整个增量式的基于语义的编码。

$$[\![\mathcal{M}]\!]_k = \left(\begin{array}{l} I_{\mathcal{M}}[s_0] \wedge \bigwedge\limits_{0\le i\le k-1} T_{\mathcal{M}}[s_i, s_{i+1}] \\ \wedge \bigwedge\limits_{w_i\in W} \bigwedge\limits_{0\le j\le k-1} (w_i^j \leftrightarrow C_i[s_j] \vee w_i^{j+1}) \wedge \bigwedge\limits_{w_i\in W} (w_i^k \leftrightarrow C_i[s_k]) \\ \wedge \bigvee\limits_{0\le l\le k-1} ({}_lL_k \wedge \bigwedge\limits_{C_i\in\mathcal{C}} (w_i^l)) \end{array}\right) \tag{3.19}$$

结合本章第3.2 节的 CEPRE技术，算法3.2 给出了增量式的基于语义 BMC 框架。函数 $cepre()$ 表示本章提出的反例集合保持的化简技术的算法实现。算法中的 tableau 构造技术（即函数 $ConstructLTLTableau()$）使用文献 [148] 提出的算法。相对于最初的 tableau 构造算法[34]，本文使用的新的 tableau 构造技术对原有的算法做了很多优化，包括利用时序连接子的单调性优化 tableau 构造、尽量少的引入公平性约束等。还有很重要的一点是，它还支持带过去时序算子的 LTL 的 tableau 构造。这里变量 $curr\_k$ 代表当前的界值，pre_k 则代表前面已经求解过的界值。$k\_invraint$ 对应的就是增量式编码中的 k 无关编码，而 $k\_dependent$ 则代表 k 相关编码。增量式的 SAT 求解器都会有缓存信息的功能，该功能的本质就是记录 SAT 求解的一些可重用信息。因此，函数 $AddToPermanentGroup$ 就是将那些需要缓存的信息记录下来以便以后求解时使用。

3.4 实验结果

我们在 ENuSMV 的基础上实现了 CEPRE 以及增量式的基于语义的 BMC 算法。由于 CEPRE本身是一种通用的技术，它跟后端使用哪一种符号化模型检验算法并没有直接的关系。因此本文的实验分两部分。第一部分主要用来评估 CEPRE技术的效果；第二部分主要是评估增量式基于语义 BMC 的效果。本节将分两小节分别讨论实验结果。

算法 3.2: 增量式的基于语义的 BMC

输入: M, φ, k

输出: $false$ or $counterexample$

```
1  begin
2      ψ := cepre(M, φ);
3      solver := CreatIncrementalSolver();
4      T_¬ψ := ConstructLTLTableau(ψ);
5      M := M ⊗ T_¬ψ; W := ∅;
6      foreach C_i ∈ C do
7          W := W ∪ {w_i};
8      pre_k := 0;
9      AddToPermanentGroup(solver, I_M);
10     for curr_k := 1 to k do
11         if pre_k < curr_k then
12             k_invaraint := ConstructKInvaraint(M, pre_k, curr_k);
13             AddToPermanentGroup(solver, k_invaraint);
14             pre_k := curr_k;
15         k_dependent := ConstructDependent(M, curr_k);
16         AddToAdditionalGroup(solver, k_dependent);
17         if Solve(solver) is SAT then
18             return counterexample;
19         DestroyAdditionalGroup(solver);
20     return false;
```

3.4.1 CePRe实验结果

如前所述，CePRe技术本身与符号化模型检验算法并没有直接的关系。因此，本文针对基于 BDD 的 LTL 模型检验算法，基于 SAT 的限界模型检验算法，以及属性制导的可达性分析算法，分别做了两组实验。第一组实验是一些工业案例，第二组实验则是随机实验。

注意，本实验中使用的基于 BDD 的模型检验算法以及 BMC 算法都是 NuSMV 中标准实现。由于 PDR 算法使用的是 AIG 格式，在做这组实验时，需要使用一个中间工具 SMV2AIG[149]，将 SMV 格式转换成 AIG 格式，然后利用 IImc[21] 来执行验证任务。

表 3.1　基于 BDD 的 LTL 模型检验算法的实验结果

模型	性质	不使用 CePRe			使用 CePRe		
		#BDD 节点	# 可达状态空间	# 时间 (sec.)	#BDD 节点	# 可达状态空间	# 时间 (sec.)
srg5	Ptimo.ltl	7946	720	0.024	2751	720	0.016
	Pti.gnv.ltl	29704	11460	0.092	5712	2880	0.028
	Pti.g.ltl	64749	130048	0.080	8119	32768	0.024
abp4	P2false.ltl	99501	559104	0.200	99549	559104	0.202
	P2true.ltl	61132	904384	0.116	56417	419296	0.120
	Pold.ltl	52224	353536	0.112	52272	353536	0.116
	Ptimo.ltl	78022	219616	0.168	78070	219616	0.164
	Pti.g.ltl	8377	200704	0.060	8245	200704	0.062
dme3	P0.ltl	889773	35964	5.756	527983	26316	5.096
	P1.ltl	508036	8775	0.460	508081	8775	0.474
dme5	Mdl.ltl	793905	8.64316e+06	167.346	814494	3.2097e+06	114.599
	Wat.ltl	302686	1.79217e+07	302.005	307936	1.12567e+07	286.850
	Pitmo.neg	508036	1.26202e+06	3.260	508081	1.26202e+06	3.280
msi_w-trans	Sched.ltl	2275558	7.31055e+07	6.612	2275655	7.31055e+07	6.632
	Safety.ltl	1213275	3.6528e+07	7.568	1213368	3.6528e+07	7.644
	Seq.ltl	1921787	3.5946e+07	93.570	1702585	1.7973e+07	94.085

表3.1 给出了一组基于 BDD 的 LTL 符号化模型检验算法使用和不使用 CePRe技术的对比实验结果。注意，表中使用 CePRe技术时的实验结果也包含了 CePRe技术带来的额外时空开销。从表3.1 可以得出如下结果：

- 16 条性质中，有 8 条是可以用 CePRe化简的。
- 一般情况下，如果性质可以化简，那么在后续的验证中将节省相当大的时空开销。例如，对于性质 Pti.g.ltl，相对于不使用 CePRe技术的验证，使用 CePRe技术之后，它的 BDD 节点少了 12.5% 左右。
- 当性质不能化简时，使用 CePRe技术带来的额外开销非常小。
- 还有一些额外的实验结果，在这里并没有展示出来。比如，如果性质可以化简，当模型不满足性质时，化简后的性质产生的反例路径一般都比化简前的反例路径要短。在表3.1 中，Pti.nuv.ltl、Pti.g.ltl以及 Seq.ltl，在不使用 CePRe技术时，它们产生的反例路径长度分别为 16、12 和 217，而使用 CePRe技术后，它们的反例路径长度分别为 15、10 和 194。

表3.2 给出了 BMC 算法使用和不使用 CePRe技术的对比实验结果。同样，表中使用 CePRe技术的实验结果也包含了 CePRe技术带来的额外时空开销。由于

BMC 算法是不完全的算法，因此，在使用 BMC 算法做验证时，除了模型和性质之外，一般还要求用户给定一个终止条件，一般是时间限制或者界值。在本组实验中，我们使用两个条件来保证算法终止，首先设定时间为 600 秒，对任意一条性质，如果超过 600 秒，算法将自动终止，另一个条件就是给定一个上界值，即表中的“最大深度”，如果算法在 600 秒之内超过 20 步，算法将继续执行直到超过 600。分析表3.2，可以得出如下结果：

- 对于性质 Pti.gnv.ltl，不使用 CePRe技术，BMC 算法终止时，它将产生 2101 个短句，而使用 CePRe技术仅产生 299 个短句。
- 对于性质 P0.ltl，不使用 CePRe技术，BMC 算法在探测到 35 步时，SAT 求解器会报段错误而无法继续执行，而使用 CePRe技术会在第 62 步时发现一条反例路径。

表 3.2　BMC 使用和不使用 CePRe技术的实验结果

模型	性质	不使用 CePRe		使用 CePRe		# 最大深度
		# 短句个数	# 时间 (sec.)	# 短句个数	# 时间 (sec.)	
srg5	Ptimo.ltl	272567	67.391	1371	0.143	20
	Pti.gnv.ltl	2101	0.116	299	0.024	6
	Pti.g.ltl	21	0.016	21	0.016	1
abp4	P2false.ltl	7532	3.972	7532	3.972	17
	P2true.ltl	12639	8.145	9369	7.753	20*
	Pold.ltl	7499	9.087	7499	9.488	20*
	Ptimo.ltl	6332	2.500	6332	2.512	16
	Pti.g.ltl	11952	0.841	11952	0.976	20*
dme3	P0.ltl	-	-	35102	524.207	62
	P1.ltl	216	0.036	167	0.048	1
dme5	Mdl.ltl	90	0.044	90	0.048	0
	Wat.ltl	367	0.048	274	0.052	1
	Pitmo.neg	367	0.050	277	0.058	1
msi_w-trans	Sched.ltl	14235	1.076	14235	1.078	20*
	Safety.ltl	12439	8.441	12439	8.448	20*
	Seq.ltl	1907	0.064	81	0.052	3

表3.3 给出了基于 PDR 的符号化模型检验算法使用和不使用 CePRe技术的对比实验结果。PDR 算法本身对空间的消耗比较稳定，它与问题规模的大小并没有多大的关系。因此，在本组实验中并没有比较空间消耗，而主要关注 CePRe技术带来的时间开销上的优势。由于 PDR 算法不直接支持 LTL 的化简，我们是利用

CEPRE算法单独对其化简之后，做的实验，因此，在本组实验的最后一栏，我们单独列出了 CEPRE技术的开销。

分析整个实验数据可知，如果性质可以进行化简，它将明显提高 PDR 验证的效率。即使性质不能进行化简，CEPRE带来的额外开销也是非常小的。

表 3.3　PDR 使用和不使用 CEPRE技术的实验结果

模型	性质	不使用 CePRe # 时间 (sec.)	使用 CePRe # 时间 (sec.)	# 额外时间 (sec.)
srg5	Ptimo.ltl	0.117	0.056	< 0.001
	Pti.gnv.ltl	0.084	0.043	< 0.001
	Pti.g.ltl	0.070	0.032	< 0.001
abp4	P2false.ltl	3.187	3.185	0.004
	P2true.ltl	2.996	1.239	0.004
	Pold.ltl	6.937	6.939	< 0.001
	Ptimo.ltl	0.910	0.912	< 0.001
	Pti.ltl	5.812	5.816	< 0.001
dme3	P0.ltl	9.815	6.086	< 0.001
	P1.ltl	0.147	0.142	0.004
dme5	Mdl.ltl	3.072	1.731	0.004
	Wat.ltl	4.575	3.043	0.004
	Ptimo.neg	0.073	0.073	0.004
msi_wtrans	Sched.ltl	0.135	0.136	< 0.001
	Safety.ltl	0.200	0.201	0.004
	Seq.ltl	0.091	0.045	<0.006

表3.1、3.2 和3.3 分别给出了三组工业案例的实验结果，这在一定程度说明了CEPRE技术是一种非常实用的技术。接下来，我们将给出另外一组实验结果，这组实验是一种随机实验，从统计角度来评估 CEPRE技术的效果。本组实验需要用到一个工具 LBTT[150]，该工具主要用来生成一组随机的 LTL 公式和模型。

在基于 BDD 的 LTL 模型检验算法和 BMC 算法中，我们首先随机产生长度为 3-7 的 LTL 性质各 40 条。然后，随机产生 200 个模型用于基于 BDD 的 LTL 模型检验，随机产生 200 个模型用于 BMC 验证。

在基于 BDD 的模型检验算法中，我们主要统计了以下结果：1. BDD 节点的规模；2. 可达状态空间数；3. 时间开销。其中，前两个结果从一定程度上可以反映算法的空间开销，后一个结果主要反映了时间开销。实验结果如图3.10、3.11 和3.12 所示。

在基于 BDD 的验证中，有 123（共 200）条规约是可以用 CEPRE技术化简的。从图中可以看出，随着公式长度的增加，CEPRE技术能够化简的可能性越大，其带来的优势也越来越明显。

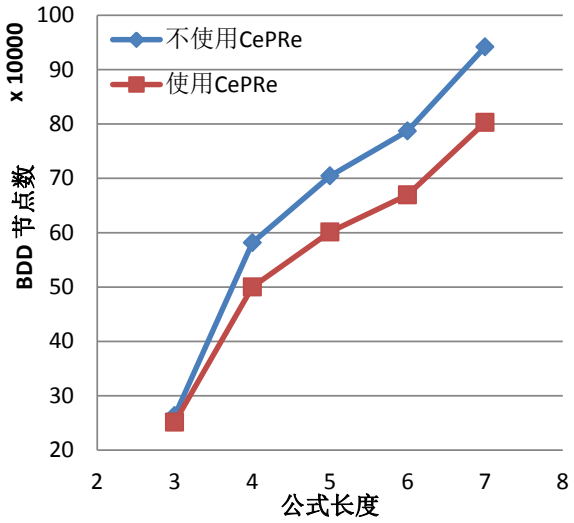


图 3.10　BDD 节点数对比

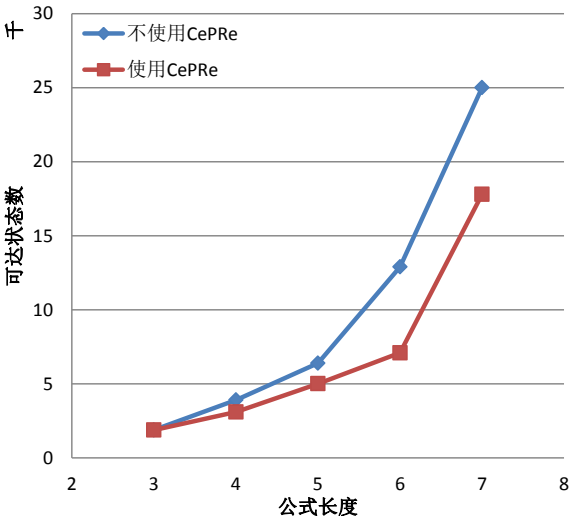


图 3.11　可达状态数对比

在 BMC 的随机实验中，我们将界值设为 20。实验主要统计两个结果：1. 短句个数（一定程度上反映了空间开销）2. 执行时间。实验结果如图3.13 和3.14 所示。

在这组实验中，200 条性质中有 118 条是可以用 CEPRE技术化简的。同样，随着公式长度的增加，CEPRE技术带来的优势在时空开销上越来越明显。

在基于 PDR 的模型检验算法的随机实验中，我们同样随机生成 200 个模型和 200 条性质（长度为 3-7，各 40 条）。由于 PDR 技术本身对空间消耗不敏感，因

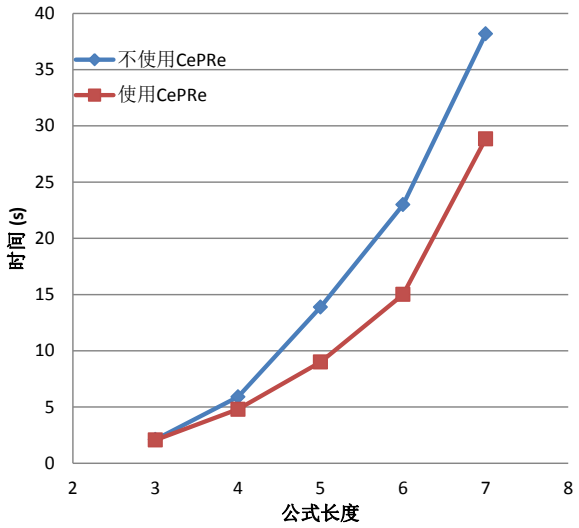


图 3.12　时间开销对比

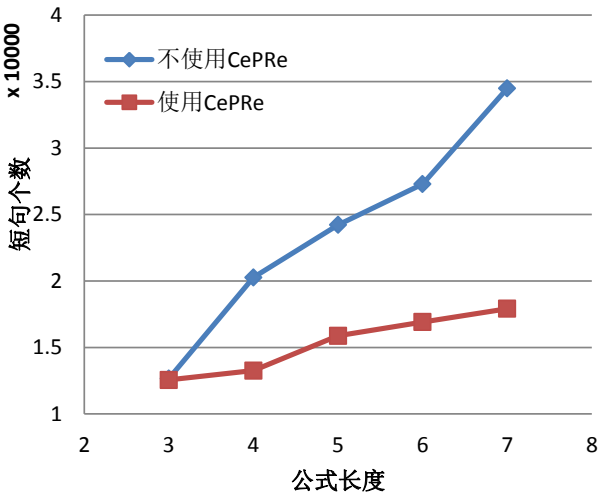


图 3.13　短句数对比

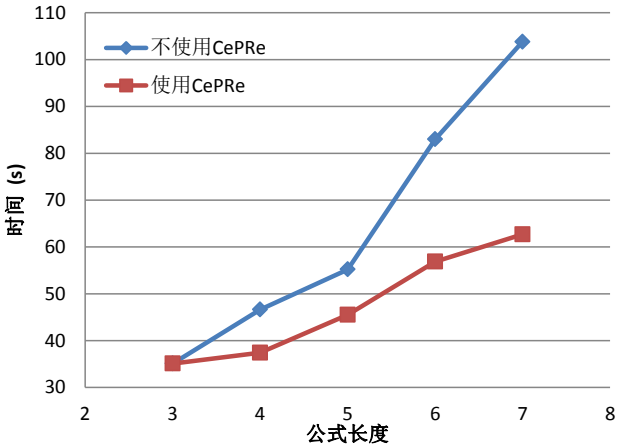


图 3.14　时间对比

此，我们主要从时间开销上做对比。主要指标就是在规定的时间 (600 秒) 求解出结果的个数。实验结果如图3.15 所示。从实验结果可以看出，随着公式长度的增加，使用 CEPRE技术带来的好处越来越明显。

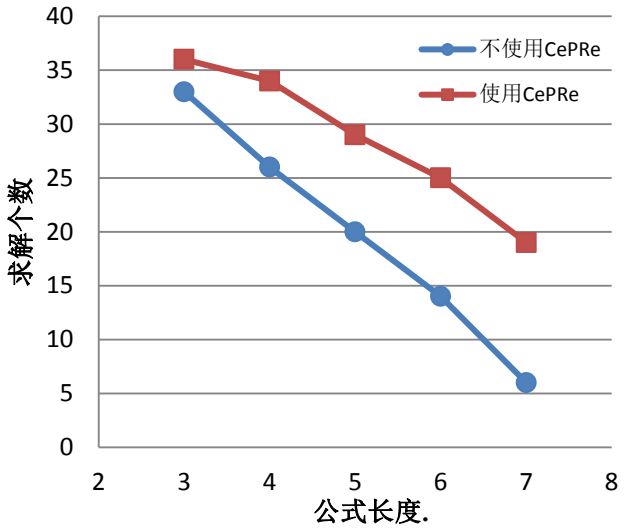


图 3.15　使用 CEPRE和不使用 CEPRE技术的 PDR 算法的随机实验结果

3.4.2　增量式的基于语义的 BMC 实验结果

本小节将给出增量式的基于语义的 BMC 实验结果。同 CEPRE技术的实验一样，在增量式的基于语义的 BMC 中，我们同样做了两组实验，一组主要是来自工业中的案例，另一组则是随机试验。不同的是，本节的实验主要对比了目前 BMC 的几种主要编码技术，包括文献 [84]、[88]、[93] 以及 [86] 的编码技术。表3.4 给出了增量式的基于语义的 BMC 在工业案例中的实验结果。表中“NuSMV”一栏表示的是 NuSMV 中的标准编码，也就是文献 [84] 中提出的编码。“VMCAI2005”则代表文献 [88] 中的编码技术。“CAV2005”则表示文献 [93] 中提到的编码技术。而“BMC_TAB”则代表文献 [86] 中提到的技术，最后一栏代表本文的编码。主要比较三个指标：1. 验证结果，即“a”栏对应的结果；2. 规定时间内的最大步数 (1 小时)，即“k”栏对于的数据；3. 执行时间 (“TO”代表超时，“MO”表示超出指定内存空间)，“t”栏对应的数据。分析实验结果可知，大多数情况，本文提出的编码技术要优于其他的编码技术。

评估本文增量式的基于语义编码的另外一组实验是随机实验。在随机实验中，我们采用 CEPRE技术中实验步骤，随机产生 200 个模型和 200 条性质，分别

表 3.4　增量式基于语义 BMC 的实验结果

Model	Prop.	NuSMV			VMCAI2005			CAV2005			BMC_TAB			BMC_INC		
		a	k	t	a	k	t	a	k	t	a	k	t	a	k	t
abp4	0	f	16	62	f	16	46	f	16	27	f	16	24	**f**	**16**	**2**
	¬0		47	TO		52	TO		354	TO		378	TO		**496**	**TO**
	1		30	TO		29	TO		45	TO		38	TO		**51**	**TO**
	2	f	17	70	f	17	36	f	17	39	f	17	f	**f**	**17**	**1**
	3		29	TO		30	TO		37	TO		33	TO		**45**	**TO**
brp	0		31	TO		241	TO		3040	TO		786	TO		**2798**	**TO**
	¬0	f	1	0	f	1	0	f	1	0	f	1	0	**f**	**1**	**0**
	¬0, nv		22	TO		21	TO		24	600	f	25	65	**f**	**25**	**19**
	1		25	TO		38	TO		196	TO		76	TO		**205**	**TO**
	¬1	f	1	0	f	1	0	f	1	0	f	1	0	**f**	**1**	**0**
counter	0		202	TO		1263	MO		11849	TO		1567	TO		**12116**	**TO**
	¬0	f	8	0	f	8	0	f	8	0	f	8	0	**f**	**8**	**0**
pci	0		17	TO	f	18	3092	f	18	1339	f	18	435	**f**	**18**	**102**
	¬0	f	0	0	f	0	0	f	0	0	f	0	0	**f**	**0**	**0**
	F0		14	TO	f	18	1121	f	18	514	f	18	60	**f**	**18**	**TO**
	1		16	TO		18	TO		20	TO		21	TO		**28**	**TO**
	¬1	f	1	0	f	1	0	f	1	0	f	1	0	**f**	**1**	**0**
dme3	0		27	MO		49	TO		48	TO	f	63	1021	**f**	**63**	**307**
	¬0	f	1	0	f	1	0	f	1	0	f	1	0	**f**	**1**	**0**
	¬0, nv		27	MO	f	59	2330	f	59	641	f	59	403	**f**	**59**	**170**
	1		42	TO		53	TO		58	TO		67	TO		**103**	**TO**
	¬1	f	1	0	f	1	0	f	1	0	f	1	0	**f**	**1**	**0**
srg5	0		13	TO		312	TO		805	MO		732	TO		**863**	**TO**
	¬0	f	1	0	f	1	0	f	1	0	f	1	0	**f**	**1**	**0**
	¬0, nv	f	6	8	f	6	0	f	6	0	f	6	0	**f**	**6**	**0**

使用上述 5 种编码技术进行测试。在这里我们将界值设置为 20，在 20 步内，统计平均的短句个数和时间开销。

实验结果如图3.16 和3.17 所示，分析实验结果可知，本文的编码技术相较其他技术有很大的优势，尤其是随着公式长度的增长，优势越明显。

3.5　本章总结

本章针对 LTL 的符号化模型检验技术提出了两种优化技术。首先，在本章的3.2 节给出一种轻量级的 LTL 化简技术 CePRe，该技术的核心思想是在模型检验之前，根据模型中的信息先对待验证的性质化简。CePRe技术并不依赖于模型检验算法，因此它是一种通用的优化技术，本节给出了 CePRe技术与三种符号化模型检验算法结合的性能分析，分别为基于 BDD 的 LTL 模型检验算法，基于语法的 BMC 算法以及基于 PDR 的模型检验算法。

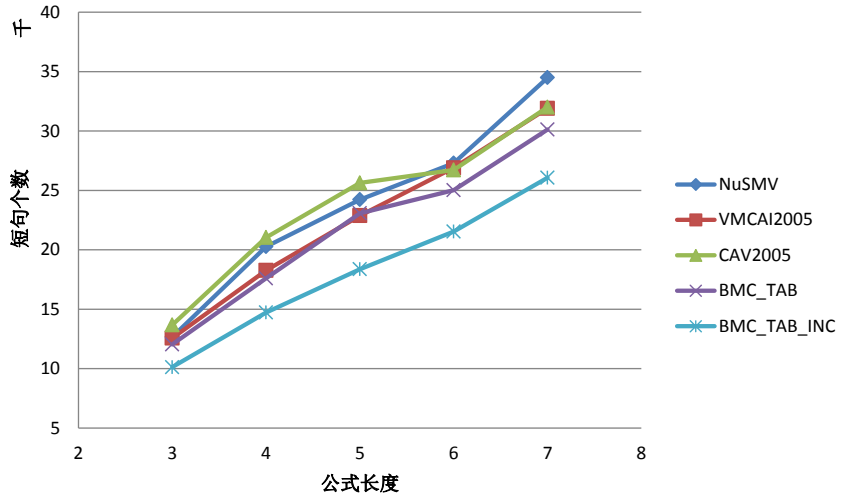


图 3.16　增量式的基于语义的随机实验结果：短句个数对比

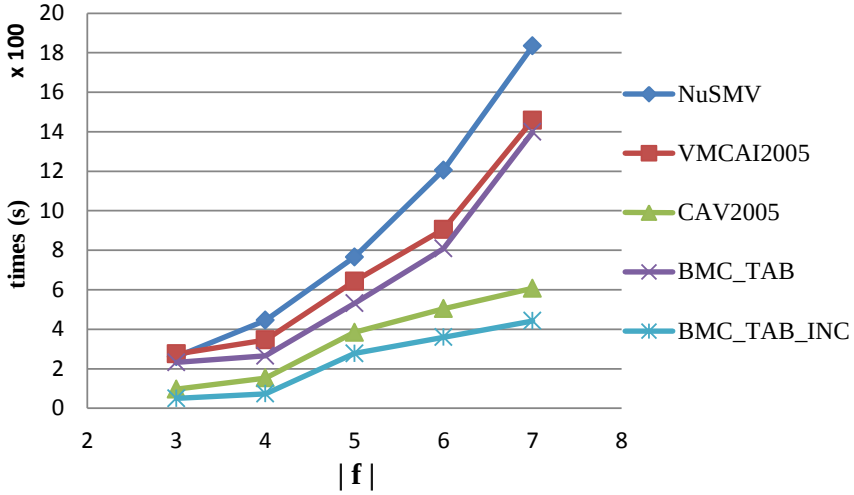


图 3.17　增量式的基于语义的随机实验结果：时间开销对比

基于 SAT 的 LTL 限界模型检验广义上讲包含两种编码技术：基于语法的编码和基于语义的编码。基于语义的编码相对于基于语法的编码有三个好处：1. 它可以很自然地利用 tableau 构造中优化技术；2. 在不引入任何其它变量的情况下，它的编码与界值 k 是成线性关系的。3. 它的编码结构更加稳定，方便计算完备阈值。本章的第二种技术就是提出了一种增量式的基于语义的编码技术。实验表明，相对于目前最好的几种 BMC 编码，我们的编码技术在时空开销上都有比较明显的优势。

第四章　ω-正规性质的限界模型检验技术

随着 SAT 求解技术的不断进步，基于 SAT 求解器的限界模型检验技术已经成功的应用于线性时序逻辑（LTL），尤其是在查找系统错误时，其相对于基于 BDD 的符号化的模型检验技术有着很大的优势。但是，LTL 并不能描述所有的 ω-正规性质，因此，有很多非常重要的时序性质，LTL 不能刻画。在工业应用中，验证人员也主要使用与 ω-正规语言等价的时序逻辑来描述要验证的规约（如 PSL）。这极大的限制了限界模型检验技术在工业中的应用。因此，扩展限界模型检验技术的验证能力，使其支持完全 ω-正规性质的验证将有很重要的理论和实际意义。

本章将讨论两种能够完全表达 ω-正规性质的扩展时序逻辑的限界模型检验技术：第一种是使用 ω-自动机作为连接子的扩展时序逻辑，即 ETL_{l+f}；第二种是带量词的时序逻辑，即 QTL。本章将分别给出这两种扩展时序逻辑的限界模型检验编码以及实验结果。

4.1　研究动机

如前所述，许多重要的 ω-正规性质不能被 LTL 所描述。事实上，为弥补 LTL 表达能力的缺陷，工业界使用了诸多能够完全表达 ω-正规性质的时序逻辑作为规约语言。因此，ω-正规性质的模型检验问题是研究领域的热点之一。

采用非确定的 ω-自动机作为时序连接子的时序逻辑 ETL_{l+f} 是能够完全表达 ω-正规性质的扩展时序逻辑。研究 ETL_{l+f} 的限界模型检验技术具有非常的重要的理论和实际意义：

- 从理论上讲，使用 Looping 自动机和使用 Finite 自动机作为时序连接子的扩展时序逻辑恰好分别描述了 **安全性**和**活性**性质。因此，混合使用这两种自动机得到的扩展时序逻辑 ETL_{l+f} 在描述性质时更加方便，并且 ETL_{l+f} 还有一个非常重要的片段，LTL。因此，研究 ETL_{l+f} 的限界模型检验技术，可以帮助建立一个线性时序逻辑的限界模型检验技术的框架。
- 刘万伟等人在文献 [37] 提出了一种基于 BDD 的 ETL_f 的符号化模型检验技术。作为与基于 BDD 的符号化模型检验互补的一种技术，限界模型检验技术在查找系统错误上具有很大的优势。因此，研究 ETL_{l+f} 的限界模型检验技术具有非常重要的实际意义。

QTL 是通过向 LTL 中添加二阶量词得到的一种时序逻辑。尽管 QTL 在实际中并没有被广泛使用，但是作为能够描述 ω-正规性质的时序逻辑，研究 QTL 的限界模型检验也具有比较重要的理论和实际意义。

- QTL 作为 LTL 的一个自然扩充，对于已经熟悉 LTL 的验证人员来说，它更容易被验证人员理解和接受。
- 相对很多其他逻辑，QTL 在描述 ω-正规性质时更加紧致，比如对于性质“p 在偶数时刻成立”的 ω-正规性质，可以用很简单直观的公式描述如下。

$$\exists q.\, q \wedge \mathsf{G}(q \rightarrow p \wedge \mathsf{XX}q)$$

- QBF 是目前形式化验证领域的热点之一，也是非常重要的研究点。探索基于 QBF 的 QTL 限界模型检验将有很重要的理论意义。
- 作为具有完全 ω-正规表达能力的时序逻辑，将 QTL 的限界模型检验技术纳入到基于语义的限界模型框架中有利于建立线性时序逻辑的限界模型检验框架。

本章的后面几节将详细讨论这两种时序逻辑的限界模型检验技术。

4.2 ETL_{l+f} 限界模型检验

本节将详细给出基于语义的 ETL_{l+f} 的限界模型检验技术。首先，我们将介绍 ETL_{l+f} 的 tableau 构造技术；然后将介绍 ETL_{l+f} 的基于语义的编码技术。

4.2.1 ETL_{l+f} 的 tableau 构造

给定 ETL_{l+f} 公式 φ，为了方便描述公式 φ 中的自动机子公式，本文使用二字母的缩写对其进行刻画。第一个字母是“P（Positive）”或“N（Negative）”，代表自动机的正负；第二个字母是“F（Finite）”或“L（Looping）”，代表自动机的类型。比如，记负的以 Looping 自动机作为连接子的子公式为 NL-子公式，形如“$\neg\mathcal{A}(\varphi_1, \varphi_2)$”，其中 $\mathcal{A}$ 是 Looping 自动机。

定义 4.1 (ETL_{l+f} **的基础公式集**): 给定 ETL_{l+f} 公式 φ，归纳定义它的基础公式集合 $el(\varphi)$ 如下：

- $el(\top) = el(\bot) = \emptyset$。
- 若 $\varphi = p$ 或者 $\varphi = \neg p$，且 $p \in AP$，则 $el(\varphi) = \{p\}$。
- 若 $\varphi = \varphi_1 \wedge \varphi_2$，则 $el(\varphi) = el(\varphi_1) \cup el(\varphi_2)$。
- 若 $\varphi = \varphi_1 \vee \varphi_2$，则 $el(\varphi) = el(\varphi_1) \cup el(\varphi_2)$。
- 若 $\varphi = \bigcirc\psi$，则 $el(\varphi) = el(\psi) \cup \{\bigcirc\psi\}$。
- 若 $\varphi = \mathcal{A}^q(\varphi_1, \ldots, \varphi_n)$ 或者 $\varphi = \neg\mathcal{A}^q(\varphi_1, \ldots, \varphi_n)$，且自动机 $\mathcal{A}$ 的状态集合为 Q，则

$$el(\varphi) = \bigcup_{1 \le k \le n} el(\varphi_k) \cup \{\bigcirc\mathcal{A}^{q'}(\varphi_1, \ldots, \varphi_n) \mid q' \in Q\}$$

根据定义可知，基础公式集中的元素只能是原子命题或者形如 $\bigcirc\psi$ 的公式。

定义 4.2 (满足集函数): 给定 ETL_{l+f} 公式 φ，归纳定义满足集函数 sat : $sub(\varphi)\cup el(\varphi)\to 2^{el(\varphi)}$:

- $sat(\top)=2^{el(\varphi)}$; $sat(\bot)=\emptyset$。
- $sat(p)=\{\Gamma\subseteq el(\varphi)\mid p\in\Gamma\}$。
- $sat(\neg p)=\{\Gamma\subseteq el(\varphi)\mid p\notin\Gamma\}$。
- $sat(\bigcirc\psi)=\{\Gamma\subseteq el(\varphi)\mid \bigcirc\psi\in\Gamma\}$。
- $sat(\varphi_1\wedge\varphi_2)=sat(\varphi_1)\cap sat(\varphi_2)$。
- $sat(\varphi_1\vee\varphi_2)=sat(\varphi_1)\cup sat(\varphi_2)$。
- 设 $\mathcal{A}=\langle\{a_1,\ldots,a_n\},Q,\delta,q,F\rangle$，
  - 若 $\mathcal{A}$ 是 Looping 自动机或者 Finite 自动机且 $q\notin F$，那么

$$sat(\mathcal{A}^q(\varphi_1,\ldots,\varphi_n))=\bigcup_{1\le k\le n}(sat(\varphi_k)\cap\bigcup_{q'\in\delta(q,a_k)}sat(\bigcirc\mathcal{A}^{q'}(\varphi_1,\ldots,\varphi_n)))$$

  - 若 $\mathcal{A}$ 是 Finite 自动机且 $q\in F$，那么 $sat(\mathcal{A}^q(\varphi_1,\ldots,\varphi_n))=2^{el(\varphi)}$。
- $sat(\neg\mathcal{A}^q(\varphi_1,\ldots,\varphi_n))=2^{el(\varphi)}\backslash sat(\mathcal{A}^q(\varphi_1,\ldots,\varphi_n))$。

例如，给定 ETL_{l+f} 公式 $\varphi=\mathcal{A}^{q_1}(p_1,p_2)$，其中 $p_1,p_2\in AP$ 且 $\mathcal{A}$ 的状态集合为 $\{q_1,q_2\}$，则 $el(\varphi)=\{p_1,p_2,\bigcirc\mathcal{A}^{q_1}(p_1,p_2),\bigcirc\mathcal{A}^{q_2}(p_1,p_2)\}$，$sat(p_1)=\{\{p_1\},\{p_1,p_2\},\{p_1,\bigcirc\mathcal{A}^{q_1}(p_1,p_2)\},\{p_1,\bigcirc\mathcal{A}^{q_2}(p_1,p_2)\},\{p_1,p_2,\bigcirc\mathcal{A}^{q_1}(p_1,p_2)\},\{p_1,p_2,\bigcirc\mathcal{A}^{q_2}(p_1,p_2)\}\},\{p_1,\bigcirc\mathcal{A}^{q_1}(p_1,p_2),\bigcirc\mathcal{A}^{q_2}(p_1,p_2)\},\{p_1,p_2,\bigcirc\mathcal{A}^{q_1}(p_1,p_2),\bigcirc\mathcal{A}^{q_2}(p_1,p_2)\}\}$。

定义 4.3: 给定 ETL_{l+f} 公式 φ，对于 φ 中任意的 PF-子公式 $\psi=\mathcal{A}(\varphi_1,\ldots,\varphi_n)$，其中 $\mathcal{A}=\langle\{a_1,\ldots,a_n\},Q,\delta,q,F\rangle$，定义关系 $\Delta_\psi^+\subseteq(2^{el(\varphi)}\times 2^Q)\times(2^{el(\varphi)}\times 2^Q)$，设 $\Gamma,\Gamma'\subseteq el(\varphi)$ 且 $P,P'\subseteq Q$，那么 $((\Gamma,P),(\Gamma',P'))\in\Delta_\psi^+$ 当且仅当：

- 若 $P\neq\emptyset$，对于任意的 $q\in P\backslash F$，存在 $1\le k\le n$，使得 $\Gamma\in sat(\varphi_k)$ 且 $P'\cap\delta(q,a_k)\neq\emptyset$。
- 若 $P=\emptyset$，对于任意的 $q\in Q$，$q\in P'$ 当且仅当 $\Gamma'\in sat(\mathcal{A}^q(\varphi_1,\ldots,\varphi_n))$。

定义 4.4: 给定 ETL_{l+f} 公式 φ，对于 φ 中任意的 NL-子公式 $\psi=\neg\mathcal{A}(\varphi_1,\ldots,\varphi_n)$，其中 $\mathcal{A}=\langle\{a_1,\ldots,a_n\},Q,\delta,q,F\rangle$，定义关系 $\Delta_\psi^-\subseteq(2^{el(\varphi)}\times 2^Q)\times(2^{el(\varphi)}\times 2^Q)$，设 $\Gamma,\Gamma'\subseteq el(\varphi)$ 且 $P,P'\subseteq Q$，那么 $((\Gamma,P),(\Gamma',P'))\in\Delta_\psi^-$ 当且仅当：

- 若 $P\neq\emptyset$，对于任意的 $q\in P\backslash F$ 以及 $1\le k\le n$，若 $\Gamma\in sat(\varphi_k)$，则 $\delta(q,a_k)\subseteq P'$

- 若 $P=\emptyset$，对于任意的 $q\in Q$，$q\in P'$ 当且仅当 $\Gamma'\notin sat(\mathcal{A}^q(\varphi_1,\ldots,\varphi_n))$。

给定 ETL$_{l+f}$ 公式 φ，设 $\psi_1,\ldots,\psi_m$ 是 φ 中的 PF-子公式且 Finite 自动机连接子对应的状态集合分别为 $Q_1,\ldots,Q_m$；$\neg\eta_1,\ldots,\neg\eta_n$ 是公式 φ 的 NL-子公式且 Looping 自动机连接子对应的状态集合为 $S_1,\ldots,S_n$。根据上面的定义，构造 φ 的 tableau $\mathcal{T}_\varphi=\langle S_\varphi,I_\varphi,\rho_\varphi,\mathcal{F}_\varphi,L_\varphi\rangle$，其中：

- $S_\varphi=\{\langle\Gamma;P_1,\ldots,P_m;R_1,\ldots,R_n\rangle\mid\Gamma\subseteq el(\varphi),P_i\subseteq Q_i(1\leq i\leq m)且R_j\subseteq S_j(1\leq j\leq n)\}$。
- $I_\varphi=\{\langle\Gamma;P_1,\ldots,P_m;R_1,\ldots,R_n\rangle\in S_\varphi\mid\Gamma\in sat(\varphi)\}$。
- 任意两个状态 $s=\langle\Gamma;P_1,\ldots,P_m;R_1,\ldots,R_n\rangle$ 和 $s'=\langle\Gamma';P'_1,\ldots,P'_m;R'_1,\ldots,R'_n\rangle$，$(s,s')\in\rho_\varphi$ 当且仅当满足下面三个条件：
  - 对于任意的 $\bigcirc\psi\in el(\varphi)$，$\Gamma\in sat(\bigcirc\psi)$ 当且仅当 $\Gamma'\in sat(\psi)$。
  - $((\Gamma,P_i),(\Gamma',P'_i))\in\Delta^+_{\psi_i}(1\leq i\leq m)$。
  - $((\Gamma,R_j),(\Gamma',R'_j))\in\Delta^-_{\eta_j}(1\leq j\leq n)$。
- $L_\varphi(\langle\Gamma;P_1,\ldots,P_m;R_1,\ldots,R_n\rangle)=\Gamma\cap AP$。
- $\mathcal{F}_\varphi=\{F_i^+\mid 1\leq i\leq m\}\cup\{F_j^-\mid 1\leq j\leq n\}$，其中

$$F_i^+=\{\langle\Gamma;P_1,\ldots,P_m;R_1,\ldots,R_n\rangle\in S_\varphi\mid P_i=\emptyset\}$$

且

$$F_j^-=\{\langle\Gamma;P_1,\ldots,P_m;R_1,\ldots,R_n\rangle\in S_\varphi\mid R_j=\emptyset\}$$

下面通过两个定理来说明上面构造的 tableau 的语言性质。

定理 4.1：对于任意的 ETL$_{l+f}$ 公式 φ 以及线性结构 $\pi\in(2^{AP})^\omega$，若 $\pi\in\mathscr{L}(\mathcal{T}_\varphi)$，那么 $\pi\models\varphi$。

证明：ETL$_{l+f}$ 是混合使用 Looping 和 Finite 自动机作为时序连接子的时序逻辑，该定理的证明是文献 [90] 中第五章相关定理证明的特殊情况。具体证明如下。设 $\sigma=s_0s_1\ldots$ 是 $\mathcal{T}_\varphi$ 中的公平路径且 $\pi=L_\varphi(\sigma)$，其中 $s_i=\langle\Gamma_i;P_{1,i},\ldots,P_{m,i};R_{1,i},\ldots,R_{n,i}\rangle$。下面用结构归纳法证明：对于任意的 $\psi\in sub(\varphi)\cup el(\varphi)$，若 $\Gamma_i\in sat(\psi)$，则 $\pi,i\models\psi$。

- 基本情形证明如下：
  - 若 $\psi=\top$ 或者 $\psi=\bot$，显然成立。
  - 若 $\psi=p$ 且 $p\in AP$，由于 $\Gamma_i\in sat(p)$ 当且仅当 $p\in\Gamma_i$ 当且仅当 $p\in\pi(i)$，当且仅当 $\pi,i\models p$。同理，可证明 $\psi=\neg p$ 时也成立。

  - 若 $\psi = \psi_1 \wedge \psi_2$，由于 $\Gamma_i \in sat(\psi)$ 当且仅当 $\Gamma_i \in sat(\psi_1)$ 且 $\Gamma_i \in sat(\psi_2)$，根据归纳假设，可知 $\pi, i \models \psi_1$ 且 $\pi, i \models \psi_2$，因此 $\pi, i \models \psi$。同理可证明 $\psi = \psi_1 \vee \psi_2$ 的情况。
  - 若 $\psi = \bigcirc\psi'$，根据 ρ_φ 的定义可知，$\Gamma_i \in sat(\psi)$ 当且仅当 $\Gamma_{i+1} \in sat(\psi')$，根据归纳假设 $\pi, i+1 \models \psi'$，因此 $\pi, i \models \psi$。

- 若 $\psi = \mathcal{A}^q(\varphi_1, \ldots, \varphi_k)$ 是 PF-子公式，设 $\psi = \psi_j (1 \leq j \leq m)$（即 φ 的第 j 个 PF-子公式），且 $\mathcal{A}^q = \langle \{a_1, \ldots, a_k\}, Q, \delta, q, F \rangle$。由于 σ 是一条公平路径，根据构造过程可知，必存在 $i_2 > i_1 > i$ 使得 $P_{j,i_1} = P_{j,i_2} = \emptyset$，且对于任意的 $i_1 < c < i_2$，有 $P_{j,c} \neq \emptyset$。

  - 首先，令 $q_0 = q$，根据已知条件，可得 $\Gamma_i \in sat(\mathcal{A}^{q_0}(\varphi_1, \ldots, \varphi_k))$。
  - 对于任意的 $t \leq i_1 - 1$，若 $\Gamma_{i+t} \in sat(\mathcal{A}^{q_t}(\varphi_1, \ldots, \varphi_k))$，那么根据 sat 的定义可知，存在 $1 \leq k_t \leq k$ 和 q' 使得 $\Gamma_{i+t} \in sat(\psi_{k_t})$，$q' \in \delta(q_t, a_{k_t})$ 且 $\Gamma_{i+t} \in sat(\bigcirc\mathcal{A}^{q'}(\varphi_1 \ldots \varphi_k))$（即，$\Gamma_{i+t+1} \in sat(\mathcal{A}^{q'}(\varphi_1, \ldots, \varphi_k))$），此时，令 $q_{t+1} = q'$。

  到现在为止，若存在 t 使得 $q_t \in F$，则停止构造；否则，继续下面的构造过程：

  - 当 $t = i_1 - i + 1$ 时，$\Gamma_t \in sat(\mathcal{A}^{q_t}(\varphi_1, \ldots, \varphi_k))$。由于 $P_{j,i_1} = \emptyset$，根据 $\Delta^+_{\psi_j}$ 的定义可知，$q_t \in P_{j,i_1+1} = P_{j,i+t}$。
  - 对于任意的 $i_1 - i < t \leq i_2 - i$，若 $q_t \in P_{j,i+t}$ 且 $q_t \notin F$，根据 $\Delta^+_{\psi_j}$ 的定义，存在 $1 \leq k_t \leq k$，使得 $\Gamma_{i+t} \in sat(\varphi_{k_t})$ 且存在 $q' \in P_{j,i+t+1} \cap \delta(q_t, a_{k_t})$，此时，设 $q_{t+1} = q'$。

  由于 $P_{j,i_2} = \emptyset$，根据归纳假设可知，若 $q_t \notin F$，则 $q_t \in P_{j,i+t}$，因此，必存在 $q_t \in F$。至此，我们构造出了一条无穷字上的可以被自动机 $\mathcal{A}$ 接收的有穷前缀 $a_{k_0} a_{k_1} \ldots a_{k_l}$。根据归纳假设可知，若 $\Gamma_{i+t} \in sat(\varphi_{k_t})$，则 $\pi, i+t \models \varphi_{k_t}$，因此 $\pi, i \models \psi$ 成立。

- 若 $\psi = \neg\mathcal{A}^q(\varphi_1, \ldots, \varphi_k)$ 是 NF-子公式，设 $\mathcal{A}^q = \langle \{a_1, \ldots, a_k\}, Q, \delta, q, F \rangle$。用反证法证明，首先设 $\pi, i \not\models \neg\mathcal{A}^q(\varphi_1, \ldots, \varphi_k)$，即 $\pi, i \models \mathcal{A}^q(\varphi_1, \ldots, \varphi_k)$ 成立。因此，存在一个无穷字的有穷前缀 $a_{k_0} a_{k_1} \ldots a_{k_l}$ 以及状态序列 $q_0 q_1 \ldots q_l q_{l+1}$ 满足下列条件：

  - $q_0 = q, q_{l+1} \in F$，且对于任意的 $0 \leq t \leq l$，有 $q_{t+1} \in \delta(q_t, a_{k_t})$。

– 对于任意的 $t \leq l$，有 $\pi, i+t \models \varphi_{k_t}$ 成立。

那么，我们可以得到下面的结论：

– 由于 $q_{l+1} \in F$，因此 $\Gamma_{i+l+1} \in sat(\mathcal{A}^{q_{l+1}}(\varphi_1, \ldots, \varphi_k)) = 2^{el(\varphi)}$。
– 对于任意的 $0 \leq t \leq l$，设 $\Gamma_{i+t+1} \in sat(\mathcal{A}^{q_{t+1}}(\varphi_1, \ldots, \varphi_k))$，那么根据迁移关系的定义可知 $\Gamma_{i+t} \in sat(\bigcirc\mathcal{A}^{q_{t+1}}(\varphi_1, \ldots, \varphi_k))$，另外，由于 $\pi, i+t \models \varphi_{k_t}$ 成立，因此 $\Gamma_{i+t} \in sat(\varphi_{k_t})$。因此，根据 sat 函数的定义可知，$\Gamma_{i+t} \in sat(\mathcal{A}^{q_t}(\varphi_1, \ldots, \varphi_k))$。

根据上面的结论可知，若 $t = 0$，则 $\Gamma_i \in sat(\mathcal{A}^{q_0}(\varphi_1, \ldots, \varphi_k))$，而根据已知条件可知 $\Gamma_i \in sat(\neg\mathcal{A}^{q_0}(\varphi_1, \ldots, \varphi_k))(q = q_0)$，因此得出矛盾。故 $\pi, i \models \mathcal{A}^q(\varphi_1, \ldots, \varphi_k)$ 不成立，因此 $\pi, i \models \psi$ 成立。

- 若 $\psi = \mathcal{A}^q(\varphi_1, \ldots, \varphi_k)$ 是 PL-子公式，其中 $\mathcal{A}^q = \langle\{a_1, \ldots, a_k\}, Q, \delta, q, F\rangle$，证明如下：

  – 令 $q_0 = q$，由于 $\Gamma_i \in sat(\psi)$，因此 $\Gamma_i \in sat(\mathcal{A}^{q_0}(\varphi_1, \ldots, \varphi_k))$。
  – 对于任意的 $t \geq 0$，设 $\Gamma_{i+t} \in sat(\mathcal{A}^{q_t}(\varphi_1, \ldots, \varphi_k))$，根据 sat 定义，存在 $1 \leq k_t \leq k$ 使得 $\Gamma_{i+t} \in sat(\varphi_{k_t})$ 成立且存在 $q' \in \delta(q_t, a_{k_t})$ 使得 $\Gamma_{i+t} \in sat(\bigcirc\mathcal{A}^{q'}(\varphi_1, \ldots, \varphi_k))$ 成立（即 $\Gamma_{i+t+1} \in sat(\mathcal{A}^{q'}(\varphi_1, \ldots, \varphi_k))$ 成立），令 $q_{t+1} = q'$。

  根据上面的构造过程，我么可以得到一个 $\mathcal{A}^q$ 上的无穷字 $a_{k_0}a_{k_1}\ldots$。同时，根据归纳假设可知，若 $\Gamma_{i+t} \in sat(\varphi_{k_t})$，则 $\pi, i+t \models \varphi_{k_t}$，当 $t = 0$ 时，可得 $\pi, i \models \psi$ 成立。

- 最后一种情况，若 $\psi = \neg\mathcal{A}^q(\varphi_1, \ldots, \varphi_k)$ 是 NL-子公式，其中 $\mathcal{A}^q = \langle\{a_1, \ldots, a_k\}, Q, \delta, q, F\rangle$。假设 $\pi, i \not\models \psi$，即 $\pi, i \models \mathcal{A}^q(\varphi_1, \ldots, \varphi_k)$。那么，根据定义可知，存在无穷字 $w = a_{k_0}a_{k_1}\ldots \in \mathscr{L}(\mathcal{A}^q)$ 且对于任意的 $t \geq 0$，有 $\pi, i+t \models \varphi_{k_t}$，令字 w 在 $\mathcal{A}^q$ 上的运行为 $q_0q_1\ldots$ 且 $q_0 = q$。设 $\psi = \neg\eta_j$，即公式 φ 的第 j 个 NL-子公式，由于 σ 是公平路径，因此，必存在 $i_2 > i_1 > i$ 使得 $R_{j,i_1} = R_{j,i_2} = \emptyset$ 且对于任意的 $i_1 < c < i_2$ 有 $R_{j,c} \neq \emptyset$，

  – 由于 $q_0 = q$，因此 $\Gamma_i \in sat(\neg\mathcal{A}^{q_0}(\varphi_1, \ldots, \varphi_k))$。
  – 根据前面的讨论可知（NF-子公式情况），若 $\pi, i+t \models \varphi_{k_t}$，则 $\Gamma_{i+t} \in sat(\varphi_{k_t})$。对于任意的 $t > 0$，若 $\Gamma_{i+t} \in sat(\neg\mathcal{A}^{q_t}(\varphi_1, \ldots, \varphi_k))$ 成立，根

据 sat 的定义可知，对于任意的 $q' \in \delta(q_t, a_{k_t})$，则 $\Gamma_{i+t} \notin sat(\bigcirc\mathcal{A}^{q'}(\varphi_1, \ldots, \varphi_k))$。注意，由于 $q_{t+1} \in \delta(q_t, a_k)$，因此 $\Gamma_{i+t+1} \in sat(\neg\mathcal{A}^{q_{t+1}}(\varphi_1, \ldots, \varphi_k))$。

由上可知，对于任意的 $t \geq 0$，有 $\Gamma_{i+t} \in sat(\neg\mathcal{A}^{q_t}(\varphi_1, \ldots, \varphi_k))$。根据 $\Delta^{-}_{\neg\eta_j}$ 的定义可知，

- 由于 $R_{j,i_1} = \emptyset$，因此 $q_{i_1+1-i} \in R_{j,i_1+1}$。
- 对于任意的 $q_t \in R_{j,i+t}$，由于 $\Gamma_{i+t} \in sat(\varphi_{k_t})$，所以对于 $q' \in \delta(q_t, a_{k_t})$，有 $q' \in R_{j,i+t+1}$，因此 $q_{t+1} \in R_{j,i+t+1}$。

由上述归纳可知，对于任意的 $t \geq i_1 - i + 1$，有 $q_t \in R_{j,i+t}$。而这与 $R_{j,i_2} = \emptyset$ 是矛盾的，因此假设 $\pi, i \models \mathcal{A}^q(\varphi_1, \ldots, \varphi_k)$ 是正确的，因此 $\pi, i \models \psi$ 成立。

综上可知，当 $i = 0$ 时，若 $\Gamma_0 \in sat(\varphi)$，那么 $\pi, 0 \models \varphi$，即 $\pi \models \varphi$，证毕。■

定理 4.2: 对于任意的 ETL_{l+f} 公式 φ 以及线性结构 $\pi \in (2^{AP})^\omega$，若 $\pi \models \varphi$，那么 $\pi \in \mathscr{L}(\mathcal{T}_\varphi)$。

证明：若 $\pi \models \varphi$ 成立，为了证明 $\pi \in \mathscr{L}(\mathcal{T}_\varphi)$ 成立，我们需要构造一条状态序列 $\sigma = s_0 s_1 \ldots$，其中 $s_i = \langle \Gamma_i; P_{1,i}, \ldots, P_{m,i}; R_{1,i}, \ldots, R_{n,i} \rangle \in S_\varphi$，并且我们要证明 σ 是 $\mathcal{T}_\varphi$ 上的一条公平路径且 $\pi = L_\varphi(\sigma)$。我们分三部分构造这条路径。

- 我们首先构造 Γ_i。

  对于任意的 $i \geq 0$，我们令 $\Gamma_i = \{\psi \in el(\varphi) \mid \pi, i \models \psi\}$。下面我们将证明下面的结论成立：对于任意的 $\psi \in sub(\varphi) \cup el(\varphi)$，$\Gamma_i \in sat(\psi)$ 当且仅当 $\pi, i \models \psi$。

  我们用公式结构归纳法证明上述结论。

  - 若 $\psi = \bot$ 或者 $\psi = \top$，显然成立。
  - 若 $\psi = p \in AP$，那么 $\psi \in el(\varphi)$，所以 $\Gamma_i \in sat(p)$ 当且仅当 $p \in \Gamma_i$ 当且仅当 $p \in \pi(i)$ 当且仅当 $\pi, i \models p$。同理可证明 $\psi = \neg p$ 时的情况。
  - 若 $\psi = \bigcirc\psi'$，根据 el 的定义可知，此时 $\psi \in el(\varphi)$，因此，$\Gamma_i \in sat(\bigcirc\psi')$ 当且仅当 $\psi \in \Gamma_i$ 当且仅当 $\pi, i \models \psi$ 成立。
  - 若 $\psi = \psi_1 \wedge \psi_2$，那么 $\Gamma_i \in sat(\psi)$ 当且仅当 $\Gamma_i \in sat(\psi_1)$ 且 $\Gamma_i \in sat(\psi_2)$。根据归纳假设可知，$\pi, i \models \psi_1$ 且 $\pi, i \models \psi_2$，因此 $\pi, i \models \psi$ 成立。同理可证明 $\psi = \psi_1 \vee \psi_2$ 时的情况。

– 若 $\psi = \mathcal{A}^q(\varphi_1, \ldots, \varphi_k)$，其中 $\mathcal{A}^q = \langle\{a_1, \ldots, a_k\}, Q, \delta, q, F\rangle$。若 $\mathcal{A}$ 是 Looping 自动机或者是 Finite 自动机且 $q \notin F$，不难得出下式成立：

$$\mathcal{A}^q(\varphi_1, \ldots, \varphi_k) \leftrightarrow \bigvee_{1 \le t \le k} (\varphi_t \wedge \bigvee_{q' \in \delta(q, a_t)} \bigcirc \mathcal{A}^{q'}(\varphi_1, \ldots, \varphi_k))$$

根据 sat 的定义可知，$sat(\psi' \vee \psi'') = sat(\psi') \cup sat(\psi'')$ 且 $sat(\psi' \wedge \psi'') = sat(\psi') \cap sat(\psi'')$，因此，根据归纳假设可知，$\pi, i \models \psi$ 当且仅当

$$\Gamma_i \in \bigcup_{1 \le t \le k} (sat(\varphi_t) \cap \bigcup_{q' \in \delta(q, a_t)} sat(\bigcirc \mathcal{A}^{q'}(\varphi_1, \ldots, \varphi_k))) = sat(\mathcal{A}^q(\varphi_1, \ldots, \varphi_k))$$

否则，若 $\mathcal{A}$ 是 Finite 自动机且 $q \in F$，那么

$$\mathcal{A}^q(\varphi_1, \ldots, \varphi_k) \leftrightarrow \top$$

这时，$sat(\mathcal{A}^q(\varphi_1, \ldots, \varphi_k)) = 2^{el(\varphi)}$，因此，$\pi, i \models \psi$ 当且仅当 $\Gamma_i \in sat(\psi)$ 也成立。

– 若 $\psi = \neg\mathcal{A}^q(\varphi_1, \ldots, \varphi_k)$，那么 $\Gamma_i \in sat(\psi)$ 当且仅当 $\Gamma_i \notin sat(\mathcal{A}^q(\varphi_1, \ldots, \varphi_k))$ 当且仅当 $\pi, i \not\models \mathcal{A}^q(\varphi_1, \ldots, \varphi_k)$ 当且仅当 $\pi, i \models \psi$。

根据上面的构造过程可知下面的结论成立：

1. 由于 $\pi, 0 \models \varphi$ 成立，因此 $\Gamma_0 \in sat(\varphi)$，因此 $s_0 \in I_\varphi$。
2. 对于任意的 i 以及 $\bigcirc\psi \in el(\varphi)$，$\Gamma_i \in sat(\bigcirc\psi)$ 当且仅当 $\pi, i \models \bigcirc\psi$ 当且仅当 $\pi, i+1 \models \psi$ 当且仅当 $\Gamma_{i+1} \in sat(\psi)$。
3. 对于任意的 $p \in AP$ 以及任意的 i，$p \in \pi(i)$ 当且仅当 $\pi, i \models p$ 当且仅当 $p \in \Gamma_i$，因此 $\pi(i) = \Gamma_i \cap AP$。

- 第二步，我们构造 $P_{j,i}(1 \le j \le m)$。
  设 φ 的第 j 个 PF-子公式 $\psi_j = \mathcal{A}^q(\varphi_1, \ldots, \varphi_k)$，其中自动机连接子 $\mathcal{A}^q = \langle\{a_1, \ldots, a_k\}, Q, \delta, q, F\rangle$，$P_{j,i}$ 的构造过程如下：
  – 首先，我们需要构造一组关键点 $\ell_0, \ell_1, \ldots$，使得在这些关键点处 $P_{j,\ell_t} = \emptyset$。令 $\ell_0 = 0$，对于任意的 $t \ge 0$，一旦 ℓ_t 确定，我们按下面的两个步骤确定 ℓ_{t+1} 以及 $P_{j,c}(\ell_t < c < \ell_{t+1})$。
  – 令 $P_{j,\ell_t} = \emptyset$，令 $P_{j,\ell_t+1} = \{q' \mid \pi, \ell_t + 1 \models \mathcal{A}^{q'}(\varphi_1, \ldots, \varphi_k)\}$。对于任意的 $q' \in P_{j,l_t+1}$，必存在一个有穷字 $w_{q'} = a_{k_0}a_{k_1}\ldots a_{k_s}$ 和一个有穷的状态序列 $\chi_{q'} = q_0 q_1 \ldots q_{s+1}$，且 $q_0 = q', q_{s+1} \in F$，并且对于任意的 $c \le s$，有 $q_{c+1} \in \delta(q_c, a_{k_c})$ 且 $\pi, \ell_t + c + 1 \models \varphi_{k_s}$（由此可知，$\Gamma_{\ell_t+c+1} \in sat(\varphi_{k_s})$）。

– 现在，令 $\ell_{t+1} = \ell_t + \max\{|\chi_{q'}| \mid q' \in P_{j,\ell_t+1}\} + 1$，对于任意的 $\ell_t + 1 < c < \ell_{t+1}$，令

$$P_{j,c} = \{\chi_{q'}(c - \ell_t - 1) \mid |\chi_{q'}| > c - \ell_t - 1\}$$

其中 $\chi_{q'}(s)$ 是指状态序列 $\chi_{q'}$ 的第 s 个状态。

不难证明，根据上面的构造过程得到的 $P_{j,i}$，对于任意的 i，有 $((\Gamma_i, P_{j,i}), (\Gamma_{i+1}, P_{j,i+1})) \in \Delta^+_{\psi_j}$。并且对于任意的 t，$P_{j,\ell_t} = \emptyset$。

- 第三步构造 $R_{j,i}(1 \le j \le n)$。
  设 $\neg\eta_j = \neg\mathcal{A}^q(\varphi_1, \ldots, \varphi_k)$ 是公式 φ 的第 j 个 NL-子公式，其中自动机连接子 $\mathcal{A}^q = \langle\{a_1, \ldots, a_k\}, Q, \delta, q, F\rangle$，$R_{j,i}$ 的构造过程如下进行：
  – 首先，令 $\ell_0 = 0$，对于任意的 $t \ge 0$，一旦 ℓ_t 确定了，使用下面的步骤获取 ℓ_{t+1} 以及每一个集合 $R_{j,c}(\ell_t < c < \ell_{t+1})$。
  – 对于任意的 t，令 $R_{j,\ell_t} = \emptyset$，令 $R_{j,\ell_t+1} = \{q' \mid \Gamma_{\ell_t+1} \in sat(\neg\mathcal{A}^{q'}(\varphi_1, \ldots, \varphi_k))\}$，对于任意的 $c \ge \ell_t + 1$，令

$$R_{j,c+1} = \bigcup_{q' \in R_{j,c}} \{\delta(q', a_s) \mid 1 \le s \le k, \Gamma_c \in sat(\varphi_s)\}$$

  – 由于我们已经证明了对于任意的 $\psi \in sub(\varphi) \cup el(\varphi)$，有 $\Gamma, i \models \psi$ 当且仅当 $\pi, i \models \psi$，因此，不难证明若 $R_{j,\ell_t+1} \ne \emptyset$，必存在 $c > \ell_t + 1$ 使得 $R_{j,c} = \emptyset$（否则，根据 König 引理，可以找到一条 $\mathcal{A}^{q'}$ 上的运行，因此，我们可以证明存在 $q' \in R_{j,\ell_t+1}$，使得 $\pi, \ell_t + 1 \models \mathcal{A}^{q'}(\varphi_1, \ldots, \varphi_k)$，因此 $\Gamma_{\ell_t+1} \in sat(\mathcal{A}^{q'}(\varphi_1, \ldots, \varphi_k))$，这与已知矛盾）。

  因此，若 $R_{j,\ell_t+1} = \emptyset$，令 $\ell_{t+1} = \ell_t + 1$，否则，令 $\ell_t + 1 = c$。根据构造过程，可以验证对于任意的 i，$((\Gamma_i, R_{j,i}), (\Gamma_{i+1}, R_{j,i+1})) \in \Delta^-_{\neg\eta_j}$ 成立，且对于任意的 $t \ge 0$，$R_{j,\ell_t} = \emptyset$。

综上可知，对于任意的 $i \ge 0$，则 $(s_i, s_{i+1}) \in \rho_\varphi$ 成立，因此 σ 是 $\mathcal{T}_\varphi$ 上的公平路径，根据 Γ_i 的构造可知，$\pi = L_\varphi(\sigma)$，因此 $\pi \in \mathscr{L}(\mathcal{T}_\varphi)$。证毕。 ■

根据定理4.1和4.2，可以得到下面的定理。

定理 4.3: 给定模型 M 以及 ETL_{l+f} 性质 φ，$M \not\models \varphi$ 当且仅当 $\mathscr{L}(M) \cap \mathscr{L}(\mathcal{T}_{\neg\varphi}) \ne \emptyset$，即乘积模型 $M \otimes \mathcal{T}_{\neg\varphi}$ 中存在一条公平路径。

从 tableau 的构造过程可得定理4.4。

定理 4.4: 给定 ETL$_{l+f}$ 公式 φ，公式 φ 的 tableau $\mathcal{T}_\varphi$ 最多包含 $4^{|el(\varphi)|}$ 个状态。

证明：根据 taleau 的构造过程可知，$\mathcal{T}_\varphi$ 中的任意一个状态都是形如 $\langle\Gamma; P_1,\ldots,P_m; R_1,\ldots,R_n\rangle$ 的序偶。对于 Γ 而言，由于 $\Gamma \subseteq el(\varphi)$，因此 Γ 最多有 $2^{|el(\varphi)|}$ 中可能。设 PF-子公式 $\psi_i = \mathcal{A}_i(\varphi_1,\ldots,\varphi_k)$ 且 $\mathcal{A}_i$ 的状态集合为 Q_i，对于 Q_i 中的任意一个状态 q，根据基础公式集的定义可知，它一定唯一对应 $el(\varphi)$ 中一个形如 $\bigcirc\mathcal{A}_i^q(\varphi_1,\ldots,\varphi_k)$ 的基础公式。同理，对于任意一个 φ 中的 NL-子公式 $\neg\eta_j = \neg\mathcal{A}_j(\varphi'_1,\ldots,\varphi'_l)$，设 $\mathcal{A}_j$ 的状态集合为 S_j。对于 S_j 中任意一个状态 s，根据基础公式集的定义可知，它一定唯一对应 $el(\varphi)$ 中一个形如 $\bigcirc\mathcal{A}_j^s(\varphi'_1,\ldots,\varphi'_l)$ 的基础公式。由此，可知下面的公式成立。

$$(\sum_{1\le i\le m}|Qi| + \sum_{1\le j\le n}|S_j|) \le |el(\varphi)\backslash AP|$$

由于 $P_i \subseteq Q_i$ 且 $R_j \subseteq S_j$，因此 $|S_\varphi| \le (2^{|el(\varphi)|})^2 = 4^{|el(\varphi)|}$ 成立。 ■

4.2.2 基于语义的 ETL$_{l+f}$ 的 BMC 编码

根据定理4.3可知，ETL$_{l+f}$ 的模型检验问题可以转化为公平路径查找问题。给定模型 M 以及 ETL$_{l+f}$ 公式 φ，$M \models \varphi$ 是否成立只需判断 $M \otimes \mathcal{T}_{\neg\varphi}$ 中是否存在一条公平路径即可。任给一个符号化模型 $\mathcal{M}$，其在受限路径上的公平路径查找问题可以编码成布尔公式。因此 ETL$_{l+f}$ 在受限路径上的模型检验问题（即模型检验问题）可以编码成布尔公式，然后利用 SAT 求解器进行验证。给定模型 M，ETL$_{l+f}$ 公式 φ 以及界值 $k \in \mathbb{N}$，基于语义的限界模型检验编码主要分三步进行:

- 首先，根据本章第4.2.1节介绍的技术构造公式 $\neg\varphi$ 的 tableau $\mathcal{T}_{\neg\varphi}$;
- 构造符号化的乘积模型 $\mathcal{M} = M \otimes \mathcal{T}_{\neg\varphi}$;
- 编码 $\mathcal{M} = \langle I_\mathcal{M}, T_\mathcal{M}, \mathcal{C}_\mathcal{M}\rangle$ 在 k 步之内是否存在公平路径的问题，如公式4.1所示。

$$[\![\mathcal{M}]\!]_k = I_\mathcal{M}[s_0] \wedge \bigwedge_{i=0}^{k-1} T_\mathcal{M}[s_i, s_{i+1}] \wedge \bigvee_{l=0}^{k}({}_lL_k \wedge \bigwedge_{C_i\in\mathcal{C}}(\bigvee_{j=l}^{k} C_i[s_j])) \tag{4.1}$$

公式4.1与本文第3章中的公式3.12是一样的。因此，增量式的基于语义的编码技术同样可以应用到基于语义的 ETL$_{l+f}$ 编码中。

为了获得增量式的编码，我们需要引入一些新的变量。为每一个公平性约束 $C_i \in \mathcal{C}$ 引入变量 r_i 组成变量集合 R，使得公式4.2成立。

$$\begin{aligned} r_i^j &\leftrightarrow C_i[s_j] \vee r_i^{j+1} \\ r_i^k &\leftrightarrow C_i[s_k] \end{aligned} \tag{4.2}$$

这样得到的新的编码如公式4.3所示。

$$[\![\mathcal{M}]\!]_k = \left(\begin{array}{l} I_{\mathcal{M}}[s_0] \wedge \bigwedge\limits_{0\le i\le k-1} T_{\mathcal{M}}[s_i, s_{i+1}] \\ \wedge \bigwedge\limits_{r_i\in R} \bigwedge\limits_{0\le j\le k-1} (r_i^j \leftrightarrow C_i[s_j] \vee r_i^{j+1}) \wedge \bigwedge\limits_{r_i\in R} (r_i^k \leftrightarrow C_i[s_k]) \\ \wedge \bigvee\limits_{0\le l\le k-1} ({}_lL_k \wedge \bigwedge\limits_{C_i\in\mathcal{C}} (r_i^l)) \end{array} \right) \tag{4.3}$$

根据公式4.3，容易得到下面的定理。

定理 4.5: $\mathscr{L}(\mathcal{M}) \neq \emptyset$ 当且仅当存在 $k \in \mathbb{N}$ 使得 $[\![\mathcal{M}]\!]_k$ 是可满足的。

4.3 QTL 限界模型检验

4.3.1 基于语法的 QTL 限界模型检验

QTL 是通过向 LTL 中添加二阶量词得到的一种时序逻辑。本文提出一种基于 QBF 的 QTL 限界模型检验技术。我们首先将给定的 QTL 公式编码成一个带量词的布尔公式，然后利用 QBF 求解器验证给定的模型是否满足 QTL 性质。

同 LTL 的限界模型检验一样，我们首先给出 QTL 的限界语义。给定无穷路径 $\pi = s_0 s_1 \ldots$ 以及界值 k，限界语义就是定义在路径 π 的有穷前缀 $s_0 s_1 \ldots s_k$ 上的语义。根据路径 π 是否为 k-loop 路径（见定义2.25），QTL 的限界语义也要分两种情况。

定义 4.5 (QTL 的 k-loop 限界语义): 给定界值 $k \in \mathbb{N}$ 以及一条 k-loop 路径 π，对于任意的 QTL 公式 φ，$\pi \models_k \varphi$ 当且仅当 $\pi \models \varphi$。

定义4.5 说明，QTL 在 k-loop 上的限界语义与普通的语义定义是一样的。但是，如果路径不是 k-loop 的，其语义就不能这么定义了。为了编码方便，在定义4.6 中，我们显示的给出了派生量词“∀”的限界语义。

定义 4.6 (QTL 的非 k-loop 限界语义): 给定界值 $k \in \mathbb{N}$ 以及非 k-loop 路径 π 以及位置 $0 \le i \le k$，QTL 公式 φ 的非 k-loop 限界语义可以归纳定义如下。

- 若 $\varphi = \top$，那么 $\pi, i \models_k \varphi$。
- 若 $\varphi = \bot$，那么 $\pi, i \not\models_k \varphi$。
- 若 $\varphi = p \in AP$，那么 $\pi, i \models_k p$
- 若 $\varphi = \neg\psi$，那么 $\pi, i \models_k \varphi$ 当且仅当 $\pi, i \not\models_k \psi$。
- 若 $\varphi = \varphi_1 \wedge \varphi_2$，那么 $\pi, i \models_k \varphi$ 当且仅当 $\pi, i \models_k \varphi_1$ 且 $\pi, i \models_k \varphi_2$。

- 若 $\varphi = \mathsf{X}\psi$，那么 $\pi, i \models_k \varphi$ 当且仅当 $\pi, i+1 \models_k \psi$ 且 $i+1 \leq k$。
- 若 $\varphi = \varphi_1 \mathsf{U} \varphi_2$，那么 $\pi, i \models_k \varphi$ 当且仅当存在 $i \leq j \leq k$，使得 $\pi, j \models_k \varphi_2$ 且对于任意的 $i \leq m < j$ 有 $\pi, m \models_k \varphi_1$。
- 若 $\varphi = \exists q.\psi$，那么 $\pi, i \models_k \varphi$ 当且仅当存在 $\hat{\pi} \in (2^{AP})^\omega$，使得 $\hat{\pi}, i \models_k \psi$，且 $\hat{\pi}$ 满足：对于任意的 $i \leq j \leq k$ 以及任意的 $q' \in AP \backslash \{q\}$，有 $q' \in \hat{\pi}(j)$ 当且仅当 $q' \in \pi(j)$。
- 若 $\varphi = \forall q.\psi$，那么 $\pi, i \models_k \varphi$ 当且仅当对于任意的 $\hat{\pi} \in (2^{AP})^\omega$，若 $\hat{\pi}$ 满足：对于任意的 $i \leq j \leq k$ 以及任意的 $q' \in AP \backslash \{q\}$，有 $q' \in \hat{\pi}(j)$ 当且仅当 $q' \in \pi(j)$，则 $\hat{\pi} \models_k \psi$。

同样，当 $i = 0$ 时，记 QTL 公式 φ 在非 k-loop 路径上的语义 $\pi, 0 \models_k \varphi$ 为 $\pi \models_k \varphi$。

根据 QTL 的限界语义可以得到引理4.1。

引理 4.1: 给定无穷路径 π 以及 $k \in \mathbb{N}$，对于任意的 QTL 公式 φ，若 $\pi \models_k \varphi$，则 $\pi \models \varphi$。

QTL 在受限路径上 QBF 编码也分两种情况，一种情况是当路径为非 k-loop 路径时的编码，定义4.7给出了这种情况下的 QBF 编码（记作 $\|\cdot\|_k^i$）；另外一种情况是当路径为 k-loop 路径时的编码，定义4.8给出了这种情况下的 QBF 编码（记作 ${}_l\|\cdot\|_k^i$）。

定义 4.7 (**非 k-loop 上的 QTL 的 QBF 编码**): 给定 QTL 公式 φ 以及 $k, i \in \mathbb{N}$，且 $i \leq k$，那么公式 φ 在非 k-loop 路径上的 QBF 编码可以如下递归的进行。

- 若 $\varphi = p \in AP$，那么 $\|\varphi\|_k^i = p_i$。
- 若 $\varphi = \neg p$，那么 $\|\varphi\|_k^i = \neg p_i$。
- 若 $\varphi = \varphi_1 \wedge \varphi_2$，那么 $\|\varphi\|_k^i = \|\varphi_1\|_k^i \wedge \|\varphi_2\|_k^i$。
- 若 $\varphi = \mathsf{X}\psi$，如果 $i < k$，那么 $\|\varphi\|_k^i = \|\psi\|_k^{i+1}$；否则，$\|\varphi\|_k^i = \bot$。
- 若 $\varphi = \varphi_1 \mathsf{U} \varphi_2$，那么 $\|\varphi\|_k^i = \bigvee_{j=i}^{k} (\|\varphi_2\|_k^j \wedge \bigwedge_{m=i}^{j-1} \|\varphi_1\|_k^m)$。
- 若 $\varphi = \exists q.\psi$，那么 $\|\varphi\|_k^i = \exists r_i \ldots \exists r_k. (\|\psi\|_k^i)_{r_i \ldots r_k}^{q_i \cdots q_k}$ 且 $r \notin AP$。($(\|\psi\|_k^i)_{r_i \ldots r_k}^{q_i \cdots q_k}$ 表示将公式 $\|\psi\|_k^i$ 中的变量 $q_i \ldots q_k$ 分别用变量 $r_i \ldots r_k$ 替换所得到的公式)
- 若 $\varphi = \forall q.\psi$，那么 $\|\varphi\|_k^i = \forall r_i \ldots \forall r_k. (\|\psi\|_k^i)_{r_i \ldots r_k}^{q_i \cdots q_k}$ 且 $r \notin AP$。

定义 4.8 (k-loop **上的 QTL 的 QBF 编码**): 给定 QTL 公式 φ 以及 $k, i \in \mathbb{N}$，且 $i \leq k$，那么公式 φ 在 k-loop 路径上的 QBF 编码可以如下递归的进行。

- 若 $\varphi = p \in AP$，那么 ${}_l\|\varphi\|_k^i = p_i$。

- 若 $\varphi = \neg p$，那么 ${}_l\|\varphi\|_k^i = \neg p_i$。
- 若 $\varphi = \varphi_1 \wedge \varphi_2$，那么 ${}_l\|\varphi\|_k^i = {}_l\|\varphi_1\|_k^i \wedge {}_l\|\varphi_2\|_k^i$。
- 若 $\varphi = \mathsf{X}\psi$，那么 ${}_l\|\varphi\|_k^i = {}_l\|\psi\|_k^{succ(i)}$。
- 若 $\varphi = \varphi_1 \mathsf{U} \varphi_2$，那么

$$ {}_l\|\varphi\|_k^i = \bigvee_{j=i}^{k}({}_l\|\varphi_2\|_k^j \wedge \bigwedge_{m=i}^{j-1} {}_l\|\varphi_1\|_k^m) \vee \bigvee_{j=l}^{i-1}({}_l\|\varphi_2\|_k^j \wedge \bigwedge_{m=i}^{k} {}_l\|\varphi_1\|_k^m \wedge \bigwedge_{m=l}^{j-1} {}_l\|\varphi_1\|_k^m) $$

- 若 $\varphi = \exists q.\, \psi$，那么 ${}_l\|\varphi\|_k^i = \exists r_i \ldots \exists r_k.\, ({}_l\|\psi\|_k^i)_{r_i \ldots r_k}^{q_i \cdots q_k}$ 且 $r \notin AP$。
- 若 $\varphi = \forall q.\, \psi$，那么 ${}_l\|\varphi\|_k^i = \forall r_i \ldots \forall r_k.\, ({}_l\|\psi\|_k^i)_{r_i \ldots r_k}^{q_i \cdots q_k}$ 且 $r \notin AP$。

给定模型 M 以及界值 $k \in \mathbb{N}$，任意的 QTL 公式 φ，公式4.4 给出了 φ 的限界模型检验问题的整体编码。

$$ \|M, \neg\varphi\|_k = \|M\|_k \wedge ((\neg L_k \wedge \|\neg\varphi\|_k^0) \vee \bigvee_{l=0}^{k}({}_l L_k \wedge {}_l\|\neg\varphi\|_k^0)) \tag{4.4} $$

公式4.4中的模型的 k 步展开见和定义2.28一样。

定理 4.6: $\|M, \varphi\|_k$ 是可满足的当且仅当 M 中存在一条路径 π，且 $\pi \models_k \neg\varphi$。

公式4.4给出了基于语法的 QTL 的 QBF 编码，它将受限路径上的 QTL 模型检验问题转换成了带量词的布尔公式求解问题。同 LTL 的限界模型检验技术一样，该技术也是非完全的技术，它无法证明给定的模型是否满足给定的 QTL 公式。

4.3.2 基于语义的 QTL 的限界模型检验

本章的第4.3.1节给出了 QTL 的基于语法的 QBF 编码方法。但是，将 QTL 公式编码成 QBF 需要用到 QBF 求解器，而目前 QBF 求解器技术还不能处理很大规模的问题，因此基于语法的编码并不实用。本节将给出一种基于语义的编码方式，该编码的核心是构造 QTL 公式的 tableau，首先我们介绍 QTL 公式的 tableau 构造，然后给出其基于语义的编码。

4.3.2.1 QTL 的 tableau 构造

本文第2章的定义2.23给出了原子命题集合 AP 上的布尔迁移系统的定义。这里，我们给出原子命题集合 AP 子集上的布尔迁移系统的定义。

给定原子命题集合 AP 的子集 $\mathcal{V}$，记 $M_{\mathcal{V}} = \langle \mathcal{V}, I, T \rangle$ 为 $\mathcal{V}$ 上的布尔迁移系统。对于任意的 $\mathcal{U}, \mathcal{V} \subseteq AP$ 以及 $\pi \in (2^{\mathcal{V}})^{\omega}$，定义路径 π 的 $\mathcal{U}$-*restriction* 为 $\pi \downarrow_{\mathcal{U}}$，其中 $\pi \downarrow_{\mathcal{U}} = \pi(i) \cap \mathcal{U}$，记集合 $\{\pi \downarrow_{\mathcal{U}} | \, \pi \in \mathscr{L}(M)\}$ 为 $\mathscr{L}_{\mathcal{U}}(M)$。

给定 QTL 公式 φ，我们分别使用 $PF(\varphi)$ 和 $PB(\varphi)$ 代表公式 φ 中的自由变元集合和约束变元集合。若 $PF(\varphi) \cap PB(\varphi) = \emptyset$ 且对于任意的 $q \in PB(\varphi)$，q 在公式 φ 中仅被约束一次，则称 QTL 公式 φ 是**良命名** (well named)。本文假设所有的 QTL 公式都是良命名的。给定 $\pi \in (2^{\mathcal{V}})^{\omega}$ 且 $q \notin \mathcal{V}$，称 $\overline{w} \in (2^{\mathcal{V} \cup \{q\}})^{\omega}$ 是 π 的 $q\text{-}extension$，其中 $\overline{w}(i) \cap \mathcal{V} = \pi(i)$。

若 QTL 公式

$$\varphi = Q_1 q_1 \ldots Q_n q_n . \psi \tag{4.5}$$

其中 $Q_i \in \{\forall, \exists\}$ 且公式 ψ 是自由公式（即 ψ 是一个 LTL 公式），称公式 φ 是**前束范式**。本文假设所有的 QTL 公式都是前束范式。

本文提出的 QTL 公式 φ 的 tableau 构造技术是通过扩展文献 [34] 的方法得到的。构造过程根据 QTL 的公式结构分两个步骤进行，分别称为**基本构造**和**归纳构造**，下面我们逐步介绍任意 QTL 公式 φ(形式如公式4.5所示) 的 tableau 构造过程。

基本构造：如果 QTL 公式 $\varphi = \psi$，即公式 φ 是一个 LTL 公式，其 taleau 就是 ψ 的 tableau。按照文献 [34] 的构造过程可得 $\mathcal{T}_{\psi} = \langle el(\psi), \theta_{\psi}, \rho_{\psi}, \Phi_{\psi} \rangle$，其中 $el(\psi)$ 可以递归定义如下:

- $el(\top) = el(\bot) = \emptyset$。
- 若 $\psi = p \in AP$，则 $el(p) = \{p\}$。
- 若 $\psi = \neg\psi_1$，则 $el(\psi) = el(\psi_1)$。
- 若 $\psi = \psi_1 \wedge \psi_2$，则 $el(\psi) = el(\psi_1) \cup el(\psi_2)$。
- 若 $\psi = \mathsf{X}\psi_1$，则 $el(\psi) = \{q_{\mathsf{X}\psi_1}\} \cup el(\psi_1)$。
- 若 $\psi = \psi_1 \mathsf{U} \psi_2$，则 $el(\psi) = \{q_{\mathsf{X}(\psi_1 \mathsf{U} \psi_2)}\} \cup el(\psi_1) \cup el(\psi_2)$。

其中，$q_{\mathsf{X}\psi_1}$ 和 $q_{\mathsf{X}(\psi_1 \mathsf{U} \psi_2)}$ 是两个新引入的命题变元。为了定义迁移关系 ρ_{ψ}，首先需要定义函数 λ 如下:

- $\lambda(\bot) = \bot$，$\lambda(\top) = \top$。
- 若 $\psi = p \in AP$，则 $\lambda(\psi) = p$。
- 若 $\psi = \neg\psi_1$，则 $\lambda(\psi) = \neg\lambda(\psi_1)$。
- 若 $\psi = \psi_1 \wedge \psi_2$，则 $\lambda(\psi) = \lambda(\psi_1) \wedge \lambda(\psi_2)$。
- 若 $\psi = \mathsf{X}\psi_1$，则 $\lambda(\psi) = q_{\mathsf{X}\psi_1}$。
- 若 $\psi = \psi_1 \mathsf{U} \psi_2$，则 $\lambda(\psi) = \lambda(\psi_2) \vee \lambda(\psi_1) \wedge q_{\mathsf{X}(\psi_1 \mathsf{U} \psi_2)}$。

有了函数 λ，迁移关系 ρ_{ψ} 定义如公式4.6所示。

$$\rho_{\psi} = \bigvee_{q_{\mathsf{X}g} \in el(\psi)} (q_{\mathsf{X}g} \leftrightarrow (\lambda(g))') \tag{4.6}$$

公平性约束 Φ_ψ 的定义如公式4.7所示。

$$\Phi_\psi = \{\lambda(g_1 \mathsf{U} g_2) \rightarrow \lambda(g_2) \mid q_{\mathsf{X}(g_1 \mathsf{U} g_2)} \in el(\psi)\} \tag{4.7}$$

根据文献 [34]，可得定理4.7。

定理 4.7: $\mathscr{L}(\mathcal{T}_\psi) = \mathscr{L}(\psi)$。

归纳构造，设已知 QTL 公式

$$\psi = Q_{i+1} q_{i+1} \ldots Q_n q_n . g$$

的 tableau $\mathcal{T}_\psi = \langle el(\psi), \theta_\psi, \rho_\psi, \Phi_\psi \rangle$，若 $\varphi = \exists q_i . \psi$，那么令 $\mathcal{T}_\varphi = \mathcal{T}_\psi$。

定理 4.8: $\mathscr{L}_{PF(\varphi)}(\mathcal{T}_\varphi) = \mathscr{L}(\varphi)$。

证明：根据归纳假设可知 $\mathscr{L}_{PF(\psi)}(\mathcal{T}_\psi) = \mathscr{L}(\psi)$。

首先，我们先证明 $\mathscr{L}_{PF(\varphi)}(\mathcal{T}_\varphi) \subseteq \mathscr{L}(\varphi)$。任取无穷路径 $\overline{w} \in \mathscr{L}_{PF(\varphi)}(\mathcal{T}_\varphi)$，由于 $\mathcal{T}_\varphi = \mathcal{T}_\psi$，一定存在 $\mathscr{L}_{PF(\psi)}(\mathcal{T}_\varphi)$ ($\mathscr{L}_{PF(\psi)}(\mathcal{T}_\varphi) = \mathscr{L}_{PF(\psi)}(\mathcal{T}_\psi)$) 中的无穷字 $\pi \in (2^{PF(\psi)})^\omega$，且 $\overline{w} = \pi \downarrow_{PF(\varphi)}$。由于 $\{q_i\} = PF(\psi) \backslash PF(\varphi)$，因此 π 是 $\overline{w}$ 的 q_i-$extension$。由于 $\pi \in \mathscr{L}_{PF(\psi)}(\mathcal{T}_\psi)$，根据归纳假设可知 $\pi \models \psi$，因此 $\overline{w} \models \varphi$。由上可知 $\mathscr{L}_{PF(\varphi)}(\mathcal{T}_\varphi) \subseteq \mathscr{L}(\varphi)$。

下面我们证明 $\mathscr{L}(\varphi) \subseteq \mathscr{L}_{PF(\varphi)}(\mathcal{T}_\varphi)$。设 $\overline{w} \in \mathscr{L}(\varphi)$，根据定义可知，一定存在 $\overline{w}$ 的一个 q_i-$extension$ π 使得 $\pi \models \psi$ 成立。根据归纳假设，$\pi \in \mathscr{L}_{PF(\psi)}(\mathcal{T}_\psi)$，由于 $\mathscr{L}_{PF(\psi)}(\mathcal{T}_\varphi) = \mathscr{L}_{PF(\psi)}(\mathcal{T}_\psi)$ 且 $\{q_i\} = PF(\psi) \backslash PF(\varphi)$，因此 $\overline{w} \in \mathscr{L}_{PF(\varphi)}(\mathcal{T}_\varphi)$。

综上可知 $\mathscr{L}_{PF(\varphi)}(\mathcal{T}_\varphi) = \mathscr{L}(\varphi)$。 ■

归纳构造，设已知 QTL 公式

$$\psi = Q_{i+1} q_{i+1} \ldots Q_n q_n . g$$

的 tableau$\mathcal{T}_\psi = \langle el(\psi), \theta_\psi, \rho_\psi, \Phi_\psi \rangle$，若 $\varphi = \forall q_i . \psi$，令 $\mathcal{T}_\varphi = \langle el(g), \theta_\varphi, \rho_\varphi, \Phi_\varphi \rangle$，其中：

- $\rho_\varphi = (\rho_\psi)^{q_i, q_i'}_{\bot,\bot} \wedge (\rho_\psi)^{q_i, q_i'}_{\bot,\top} \wedge (\rho_\psi)^{q_i, q_i'}_{\top,\bot} \wedge (\rho_\psi)^{q_i, q_i'}_{\top,\top}$。
- $\theta_\varphi = (\theta_\psi)^{q_i}_{\bot} \wedge (\theta_\psi)^{q_i}_{\top}$。
- $\Phi_\varphi = \Phi_\psi$。

定理 4.9: $\mathscr{L}_{PF(\varphi)}(\mathcal{T}_\varphi) = \mathscr{L}(\varphi)$。

证明：根据归纳假设可知 $\mathscr{L}_{PF(\psi)}(\mathcal{T}_\psi)=\mathscr{L}(\psi)$。根据全称量词 $\forall$ 的定义可知，$\pi\models\varphi$ 当且仅当对于任意 π 的 q_i-$extension\,\overline{w}$，都有 $\overline{w}\models\psi$。根据构造过程可，$\pi\in\mathscr{L}_{PF(\varphi)}(\mathcal{T}_\varphi)$ 当且仅当对于 π 的任意的 q_i-$extension\,\overline{w}$，都有 $\overline{w}\in\mathscr{L}(\mathcal{T}_\psi)$，当且仅当 $\overline{w}\models\psi$，当且仅当 $\pi\models\varphi$。因此，$\mathscr{L}_{PF(\varphi)}(\mathcal{T}_\varphi)=\mathscr{L}(\varphi)$ 成立。 ■

根据定理4.7、4.8和4.9 可得定理4.10。

定理 4.10: 给定模型 M，以及 QTL 公式 φ，$M\not\models\varphi$ 当且仅当 $\mathscr{L}(M)\cap\mathscr{L}(\mathcal{T}_{\neg\varphi})\neq\emptyset$，即乘积模型 $M\otimes\mathcal{T}_{\neg\varphi}$ 中存在一条公平路径。

对于任意的 QTL 公式 $\varphi=Q_iq_i\ldots Q_nq_n.\psi$，其中 ψ 是 LTL 公式，根据 QTL 的 tableau 的构造过程可知，QTL 的 tableau $\mathcal{T}_\varphi$ 并没有增加 LTL 公式 ψ 对应的 tableau $\mathcal{T}_\psi$ 中的变量个数，因此，很容易得出定理4.11。

定理 4.11: 对于任意的 QTL 公式 $\varphi=Q_iq_i\ldots Q_nq_n.\psi$，公式 φ 的 tableau $\mathcal{T}_\varphi$ 最多包含 $2^{|el(\psi)|}$ 个状态。

4.3.2.2 基于语义的 QTL 布尔编码

根据定理4.10，QTL 的模型检验问题也转换成了公平路径查找问题。给定模型 M，QTL 公式 φ 以及界值 k，基于语义的 QTL 的限界模型检验编码与 ETL_{l+f} 基于语义的编码一样，也分三步进行。

1. 首先，根据4.3.2节提出的 QTL 的 tableau 构造方法，构造公式 $\neg\varphi$ 对应的 tableau $\mathcal{T}_{\neg\varphi}$。
2. 然后构造模型 M 与 $\mathcal{T}_{\neg\varphi}$ 的乘积模型 $\mathcal{M}=M\otimes\mathcal{T}_{\neg\varphi}$。
3. 最后编码乘积模型 $\mathcal{M}=\langle I_\mathcal{M},T_\mathcal{M},\mathcal{C}_\mathcal{M}\rangle$ 在 k 步展开上的公平路径查找问题，如公式4.8所示。

$$[\![\mathcal{M}]\!]_k=I_\mathcal{M}[s_0]\wedge\bigwedge_{i=0}^{k-1}T_\mathcal{M}[s_i,s_{i+1}]\wedge\bigvee_{l=0}^{k}({}_lL_k\wedge\bigwedge_{C_i\in\mathcal{C}}(\bigvee_{j=l}^{k}C_i[s_j])) \tag{4.8}$$

定理 4.12: $\mathscr{L}(\mathcal{M})\neq\emptyset$ 当且仅当存在 $k\in\mathbb{N}$ 使得 $[\![\mathcal{M}]\!]_k$ 是可满足的。

同样，本文第3章提出的增量式的基于语义的编码技术也可以适用在 QTL 的基于语义的编码上，为此，需要引入变量集合 $D=\{d_i\mid C_i\in\mathcal{C}\}$，且使得公式4.9成立。

$$\begin{aligned} d_i^j &\leftrightarrow C_i[s_j]\vee d_i^{j+1}\\ d_i^k &\leftrightarrow C_i[s_k]\end{aligned} \tag{4.9}$$

其中任意的 d_i^j 代表变量 d_i 在第 j 时刻的版本。这样可以得到新的编码如公式4.10所示。

$$[\![\mathcal{M}]\!]_k = \left(\begin{array}{l} I_{\mathcal{M}}[s_0] \wedge \bigwedge\limits_{0\le i\le k-1} T_{\mathcal{M}}[s_i, s_{i+1}] \\ \wedge \bigwedge\limits_{d_i\in D} \bigwedge\limits_{0\le j\le k-1} (d_i^j \leftrightarrow C_i[s_j] \vee d_i^{j+1}) \wedge \bigwedge\limits_{d_i\in D} (d_i^k \leftrightarrow C_i[s_k]) \\ \wedge \bigvee\limits_{0\le l\le k-1} ({}_lL_k \wedge \bigwedge\limits_{C_i\in\mathcal{C}} (d_i^l)) \end{array} \right) \tag{4.10}$$

4.4 实验结果

本文在 ENuSMV 的基础上，进一步实现了 ETL_{l+f} 的基于语义的限界模型检验算法，之所以优先实现 ETL_{l+f} 的限界模型检验算法是出于下面的考虑:

1. ETL_{l+f} 可以看作是 ETL_l 和 ETL_f 混合的一种时序逻辑，而 ETL_l 和 ETL_f 分别对应“安全性质”和“活性性质”，而它们是最常用的两种时序性质，而嵌套使用它们可以更紧致的描述 ω-正规性质。
2. 对于验证人员来说，相对于 QTL，ETL_{l+f} 更直观，也更容易使用和理解。

因此，本节主要给出 ETL_{l+f} 的限界模型检验的实验结果，具体的实现细节将在本文的第6章中给出。

由于 ENuSMV 不仅支持 ETL_{l+f} 的限界模型检验算法，它还支持基于 BDD 的符号化模型检验算法，因此本文的实验主要分三组，分别验证对比下面三个问题:

1. 第一组主要对比 ETL_{l+f} 的基于 BDD 的符号化模型检验和基于语义的限界模型检验;
2. 第二组主要对比验证仅使用 ETL_f 描述的性质规约以及使用 ETL_{l+f} 描述的性质规约的效率;
3. 第三组主要比较基于语法和语义的 LTL 限界模型检验与 ETL_{l+f} 限界模型检验之间的效率。

为了对比基于 BDD 的符号化模型检验和基于语义的限界模型检验，我们使用工业中的一个非常典型的实例：DME 电路问题（有错误)。一个 DME 电路由若干个仲裁器构成，这些仲裁器构成了一个环形网络[90]（该电路的详细描述见文[151])。我们主要验证 DME 电路中的一个典型的活性性质，即“任何一个仲裁器发出的请求将来必被响应”。实验结果如表4.1所示。

在表4.1中，“C.L.”表示反例路径的长度（即 Counterexample Length)。在本组实验中，我们将界值 k 设为 100。从表4.1的实验结果可以看出，在查找错误时，基于语义的 BMC 相对于基于 BDD 的符号化模型检验要有很大优势。

表 4.1　ETL$_{l+f}$ 的基于 BDD 与基于语义的 BMC 的对比实验结果

	BDD			BMC				
Cells	时间 (s)	内存 (MB)	C.L.	变量个数	短句个数	时间 (s)	内存 (MB)	C.L.
5	170.56	68.32	78	3939	8126	13.50	34.98	39
6	513.06	209.17	84	4677	9620	29.14	42.37	39
7	2372.28	976.14	90	5415	11113	41.32	58.48	39
8	8074.05	1482.15	96	6154	12609	52.45	67.32	39
9	$\geq 5h$	$\geq$ 2G	-	6892	14102	68.17	82.13	39
10	$\geq 5h$	$\geq$ 2G	-	7630	15599	85.13	101.21	39

由于之前的 ENuSMV 版本还支持 ETL$_f$ 的基于符号化模型检验，我们的第二组实验主要是对比，对于同一个性质，分别使用 ETL$_f$ 和 ETL$_{l+f}$ 来描述的验证效率。本组实验首先使用“二进制累加器”的模型来进行验证一个非“star-free”的性质，然后再使用 DME 电路验证一个“star-free”的性质。“二进制累加器”由若干个模二的计数器单元 bit_$0, \ldots,$ bit_$n-1$ 构成，该计数器单元以串联的方式构成，即 bit_$(i+1)$ 的 carray_in 位与 bit_i 的 carry_out 位相连接，整个电路以同步方式运转，该电路如图4.1所示。

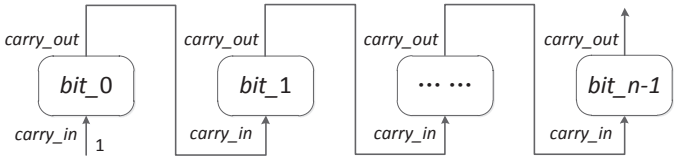


图 4.1　二进制累加器

针对“二进制累加器”，我们主要验证其一个非“star-free”的性质“bit_0 在偶数时刻都会产生进位”，该性质也称为“周期性质”，是一种典型的 LTL 不能表示的 ω-正规性质。我们分别使用 ETL$_f$ 和 ETL$_{l+f}$ 描述了该性质，其验证结果如表4.2所示。该组实验中，我们将时间设置为 1 个小时，表4.2中，“最大界值”表示在 1 个小时之内，使用限界模型检验可以验证的最大展开步数。

表 4.2　ETL$_f$ 和 ETL$_{l+f}$ 分别描述周期性质的实验结果

n	ETL$_f$				ETL$_{l+f}$			
	最大界值	内存 (MB)	变量个数	短句个数	最大界值	内存 (MB)	变量个数	短句个数
10	11032	279.44	154461	1180431	13009	382.784	182140	1417990
11	5138	148.22	77084	631981	6037	173.59	90570	754461
12	3702	129.571	3702	518287	4316	143.802	69072	612888

表4.2的实验对比了使用 ETL_f 和 ETL_{l+f} 分别描述同一个非“star-free”性质的验证效率，下面的一组实验结果则对比了分别使用 ETL_f 和 ETL_{l+f} 描述同一个“star-free”性质的验证效率。在该组实验中，我们同样使用 DME 电路，其验证性质为一个安全性质，即“在任何时刻，不可能有两个仲裁器同时被响应”。同上一组实验一样，在本组实验中，我们依然限制验证时间为 1 个小时，表4.3给出了实验结果。

表 4.3　ETL_f 和 ETL_{l+f} 分别描述安全性质的实验结果

Cells	ETL_f				ETL_{l+f}			
	最大界值	内存 (MB)	变量个数	短句个数	最大界值	内存 (MB)	变量个数	短句个数
3	68	29.14	4002	8091	101	38.03	4958	9898
4	62	34.76	4864	9764	87	40.34	5712	11512
5	54	40.12	5170	10313	73	45.37	6072	12088

表4.3对比了使用 ETL_f 和 ETL_{l+f} 描述安全性质的验证的效率，同样，我们还对比了使用 ETL_f 和 ETL_{l+f} 分别描述同一个活性性质的验证效率，表4.4给出了实验结果，该组实验选用的性质是 DME 电路的活性性质。

表 4.4　ETL_f 和 ETL_{l+f} 分别描述活性性质的实验结果

n	ETL_f				ETL_{l+f}			
	时间 (s)	变量个数	短句个数	内存 (MB)	时间 (s)	变量个数	短句个数	内存 (MB)
3	25.09	2400	4796	29.66	13.78	2340	4747	26.34
4	31.42	3120	6200	38.61	20.06	3042	6115	31.38
5	42.56	3840	7604	44.28	24.70	3744	7843	36.12

从实验结果可以看出，对于同一个性质，使用 ETL_{l+f} 来描述时的验证效率要优于仅使用 ETL_f 描述的效率。

最后一组实验，我们主要比较，对于同一个性质，用 LTL 描述与 ETL_{l+f} 描述的验证效率。本组实验使用的模型也是 DME 电路，其验证性质为其活性性质。对于 LTL 描述的性质，我们同时对比了两种编码技术的限界模型检验算法，分别为基于语法的限界模型检验和基于语义的限界模型检验，表4.5给出了这两种技术验证 DME 电路中活性性质的实验结果，对比表4.5和表4.4可知，对于同一个性质，使用 LTL 描述的验证效率更高一些。

表 4.5　LTL 基于语法和 ETL_{l+f} 基于语义 BMC 的对比实验结果

n	基于语法的 LTL BMC				基于语义的 ETL_{l+f}BMC			
	时间 (s)	变量个数	短句个数	内存 (MB)	时间 (s)	变量个数	短句	内存 (MB)
3	9.34	2052	5474	32.66	8.59	2128	4291	24.79
4	16.71	2736	6806	35.74	10.87	2812	5623	28.44
5	24.06	3420	8138	37.02	15.90	3469	6955	32.28

4.5　本章总结

本章给出了两种时序逻辑——ETL_{l+f}、QTL 的限界模型检验算法。这两种时序逻辑的表达能力均等价于 ω-正规语言。通过给出这两种时序逻辑的限界模型检验算法，本章扩展了限界模型检验技术的能力。

针对扩展时序逻辑 ETL_{l+f}，本章的第4.2节给出了其基于语义的限界模型检验编码，该技术的核心是 ETL_{l+f} 的 tableau 的构造，本章的第4.2.1小节详细介绍了 ETL_{l+f} 公式的 tableau 构造技术。本章的第4.2.2小节则给出了基于 tableau 的语义编码并利用第3章中的增量式编码技术给出了其增量式的编码。

针对时序逻辑 QTL，本章的第4.3节给出了两种编码方法。本章的第4.3.1小节给出第一种编码方法，该方法称为基于语法的编码，该编码方法的核心思想是将 QTL 公式在 k 步展开路径上的限界语义编码成 QBF 公式，但受限于 QBF 的求解能力，这种技术在理论是可行的，但是实际中效果并不好。本章的第4.3.2.1小节给出了 QTL 的第二种编码方法，该方法是基于语义的编码，该编码方法的核心是 QTL 公式的 tableau 构造。我们通过扩展 LTL 的 tableau 构造技术给出了一种 QTL 的 tableau 构造方法，基于 QTL 的 tableau，本章的第4.3.2.2小节给出了 QTL 的基于语义的编码。

本章的第4.4节给出了 ETL_{l+f} 限界模型检验的实验结果。实验主要对比了 ETL_{l+f} 基于 BDD 符号化模型检验与限界模型检验的验证效率、ETL_{l+f} 与 ETL_f 的验证效率以及 ETL_{l+f} 与 LTL 的验证效率。实验表明，ETL_{l+f} 的限界模型检验在查找系统错误时的效率要高于基于 BDD 的符号化模型检验算法，对于同一个性质，使用 ETL_{l+f} 描述的验证的效率要高于仅适用 ETL_f 描述的验证效率。

第五章　基于 SAT 的完全性保证技术

限界模型检验是一种基于 SAT 的非完全的验证技术，它不能保证模型是否满足给定的规约性质。为了使限界模型检验技术可以验证性质，研究人员提出了各种技术。其中最直接的技术就是找一个完备阈值。但是给出一个实用的完备阈值非常困难，它的复杂度与模型检验问题的复杂度是一样的。因此，在实际应用中，通过完备阈值保证完全性的方法基本上不可用。

基于语义的限界模型检验算法的本质就是受限路径上的公平性路径查找问题，所以公平性路径查找问题是线性时序逻辑模型检验的基础，而任何公平性路径查找问题都可以转化为若干个可达性问题。因此，另外一种基于 SAT 的完全性验证技术就是开发高效的可达性算法。本文也是采用第二种思路来保证基于 SAT 的符号化模型检验算法的完全性。目前，研究人员针对符号化的可达性问题已经提出了很多高效的算法，而实际应用中表现最好的算法是属性制导的可达性分析算法（PDR)。在本文的第2章已经介绍了这个基本算法。本章基于这个算法，提出了两个可达性算法，分别为交叠的双向 PDR 算法以及并行的双向 PDR 算法。两个算法的核心思想都是通过同时求解前后向可达状态空间的上逼近来加快算法到达不动点的速度，从而提高算法的效率。

5.1　研究动机

图 5.1 描述了一个迁移系统，设该迁移系统的原子命题集合为 $AP = \{x_1, x_2, x_3\}$，初始条件为 $I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$，迁移关系如图 5.1 所描述，待验证的性质 $P = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$。

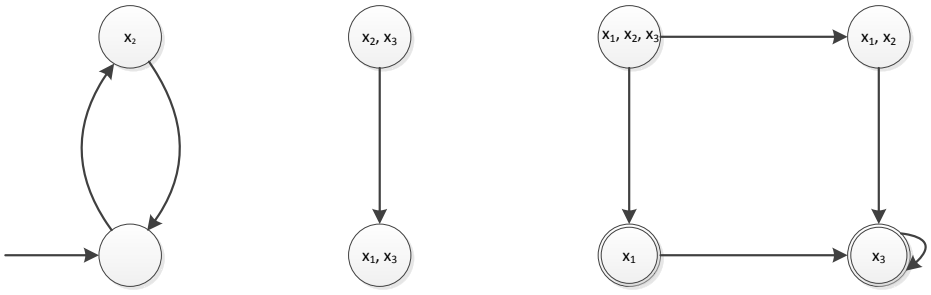


图 5.1　迁移系统 M

本文已经介绍了文献 [144] 提出的基本的 PDR 算法，该算法的核心是最小归纳子句生成算法[145]。对于图5.1 给出的系统 M 以及性质 P，基本的 PDR 算法会如下计算不动点。

- 首先初始化 $k=1$，三个逼近集合 $F_0=I, F_1=P, F_2=P$ 以及一个带优先级的任务队列 $Q_f=\emptyset$。
- 探测 $F_1 \wedge T \rightarrow P'$ 是否永真，由迁移关系以及当前 F_1 的表示，我们可以得出该公式不是永真的，这说明有属于 F_1 的状态经过一步迁移到了 $\neg P$。提取 F_1 中可达 $\neg P$ 的状态 $s=x_1 \wedge x_2 \wedge \neg x_3$。同时生成第一个任务 $\langle s,0\rangle$ 并将其插入到 Q_f 中。任务的直观意思就是证明 F_0 不能经过一步迁移到达 s，如果不能迁移到 s，处理任务的函数就会返回一个归纳短句用来增强 F_1，从而排除掉 F_1 中的状态 s 以及其他状态。
- 从 Q_f 中取出优先级最高的任务，目前只有一个任务 $\langle s,0\rangle$，探测 $F_0 \wedge \neg s \wedge T \rightarrow \neg s'$ 是否永真，由于没有初始状态可达 s，因此，该布尔公式是永真的。接下来，调用最小归纳子句生成算法，生成短句短句 $c=\neg x_1$。更新 $F_1=\{P, \neg x_1\}$。
- 由于任务队列 $Q_f=\emptyset$，算法接着探测 $F_1 \wedge T \rightarrow P'$ 是否永真，由于 F_1 包含的状态只有 $s_1=\neg x_1 \wedge x_2 \wedge \neg x_3, s_2=\neg x_1 \wedge \neg x_2 \wedge \neg x_3, s_3=\neg x_1 \wedge x_2 \wedge x_3$，这三个状态都不能经过一步迁移到达 $\neg P$，因此，$F_1 \wedge T \rightarrow P'$ 是永真的。
- 此时，算法会执行短句传播过程来进一步收紧逼近空间。对于 F_1 中的所有短句 c，探测 $F_1 \wedge T \rightarrow c'$ 是否永真，到现在为止，F_1 中的短句集合仅包含 $c=\neg x_1$，因此，探测 $F_1 \wedge T \rightarrow c'$ 是否为永真，用 SAT 求解器求解之后可以发现，该公式不是永真的，因为存在 F_1 中的状态 $s=\neg x_1 \wedge x_2 \wedge x_3$ 经过一步迁移可达 x_1 中的状态 $x_1 \wedge \neg x_2 \wedge x_3$。因此，$F_1$ 中的短句 $\neg x_1$ 不能传播到下一个逼近空间 F_2。
- 算法生成新的逼近空间 $F_3=P$，然后探测 $F_2 \wedge T \rightarrow P'$ 是否永真，此时 $F_2=P$，显然，该公式不是永真的，存在 F_2 中的状态 $s=x_1 \wedge x_2 \wedge \neg x_3$ 经过一步迁移可达 $\neg P$ 中的状态 $\neg x_1 \wedge \neg x_2 \wedge x_3$。算法生成新的任务 $\langle s,1\rangle$ 并插入到任务队列 Q_f 中。
- 从 Q_f 中取出优先级最高的任务 $\langle s,1\rangle$，探测 $F_1 \wedge \neg s \wedge T \rightarrow \neg s'$ 是否永真，由图可知，该公式是永真的。因此，调用最小归纳子句生成算法，得到短句 $c=\neg x_1 \vee \neg x_2$，将短句加入到 F_1 和 F_2 中，更新 $F_1=\{P, \neg x_1, \neg x_1 \vee \neg x_2\}$，$F_2=\{P, \neg x_1 \vee \neg x_2\}$。由于 $\neg x_1 \subseteq \neg x_1 \vee \neg x_2$，因此 $F_1=\{P, \neg x_1\}$。
- 任务队列 Q_f 为空，算法接着探测 $F_2 \wedge T \rightarrow P'$ 是否永真，调用 SAT 求解器可以该公式是永真的。
- 算法执行传播过程，由于 F_1 中的短句集合没有变化，因此，其不会向前传播。探测 F_2 中的短句 $c=\neg x_1 \vee \neg x_2$ 是否可以传播，$F_2 \wedge T \rightarrow c'$ 是永真的，

因此该短句可以传播到下一个逼近空间 F_3，更新 $F_3 = \{P, \neg x_1 \vee \neg x_2\}$。此时，算法发现 $F_2 = F_3$，找到一个不动点，算法终止，并回答性质 P 正确。

基本的 PDR 算法，也可以称之为前向的 PDR 算法 (Forawrd PDR)，它是通过计算前向的逐步逼近来收缩状态空间从而找到不动点的。如果通过计算后向的逐步逼近来收缩状态空间从而找到不动点，本文称之为后向的 PDR 算法 (Backward PDR)。对于有些系统来说，后向搜索要比前向搜索效果好。因此，我们接着分析一下对于图 5.1 给出的系统 M，用后向的 PDR 算法验证性质 P 的具体执行过程。

- 在执行后向的 PDR 算法之前，首先要把迁移关系取反得到 $\overline{T}$，同时将要验证的性质 P 取反当作初始状态 I，将初始状态 I 的非 $\neg I$ 当作待验证的性质。迁移关系取反之后的系统 $\overline{M}$ 如图 5.2 所示。
- 初始化 $l = 1$，三个逼近集合 $B_0 = \neg P, B_1 = \neg I, B_2 = \neg I$ 以及一个带优先级的任务队列 $Q_b = \emptyset$。
- 算法探测 $B_1 \wedge T \rightarrow \neg I'$ 是否永真，由图 5.2 可知，该式不成立，因为存在一个 B_1 中的状态 $s = \neg x_1 \wedge x_2 \wedge \neg x_3$ 经过一步迁移可达初始状态 $\neg x_1 \wedge \neg x_2 \wedge \neg x_3$，因此提取状态 s 并生成新的任务 $\langle s, 0\rangle$，插入队列 Q_b 中。
- 从 Q_b 中取出优先级最高的任务，目前只有一个任务 $\langle s, 0\rangle$，探测 $B_0 \wedge \neg s \wedge T \rightarrow \neg s'$ 是否永真，调用 SAT 求解器可知，该式是永真的，调用最小归纳子句生成算法得到短句 $c = x_1 \vee \neg x_2$，更新 $B_1 = \{\neg I, x_1 \vee \neg x_2\}$。
- 由于任务队列 Q_b 已经为空，算法接着探测 $B_1 \wedge T \rightarrow \neg I'$ 是否永真，调用 SAT 求解器可知，该公式为永真式。
- 算法执行传播过程来进一步收缩后向逼近空间。对于 B_1 中的短句 $c = x_1 \vee \neg x_2$，探测 $B_1 \wedge T \rightarrow c'$ 是否永真。调用 SAT 求解器可知，该公式不是永真式，因此短句 c 不能传播到下一个逼近空间。
- 算法生成新的逼近空间 $B_3 = \neg I$，然后接着探测 $B_2 \wedge T \rightarrow \neg I'$ 是否是永真式，此时 $B_2 = \neg I$，因此，上式不是永真式，因为存在一个 B_2 的状态 $s = \neg x_1 \wedge x_2 \wedge \neg x_3$ 经过一步迁移可达初始状态。提取状态 s 并生成新的任务 $\langle s, 1\rangle$ 插入到任务队列 Q_b 中。
- 从 Q_b 中取出优先级最高的任务，也就是 $\langle s, 1\rangle$，探测 $B_1 \wedge \neg s \wedge T \rightarrow \neg s'$ 是否是永真式。调用 SAT 求解器可知，该公式是永真式，接着调用最小归纳子句生成算法，得到归纳短句 $c = x_1 \vee x_3$，更新 $B_1 = \{\neg I, x_1 \vee \neg x_2, x_1 \vee x_3\}, B_2 = \{\neg I, x_1 \vee x_3\}$。进一步化简 $B_1 = \{\neg I, x_1 \vee \neg x_2\}$。
- 算法执行传播过程进一步收缩向后逼近的状态空间。对于 B_1 中的短句，其仍然不能向前传播。对于 B_2 中的短句 $c = x_1 \vee x_3$，算法探测 $B_2 \wedge T \rightarrow c'$ 是否为永真式，调用 SAT 求解器可知，该公式是永真式，因此将短句 c 传

播到下一个逼近空间 B_3，此时，算法会检测到 $B_2 = B_3$，找到一个不动点，算法终止，并回答性质 P 正确。

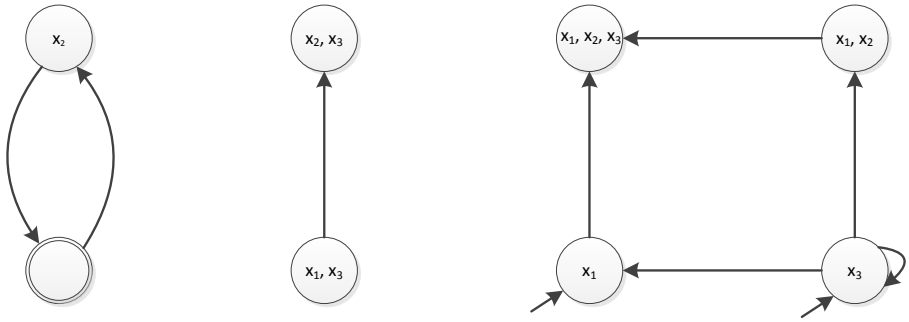


图 5.2　迁移系统 $\overline{M}$

从上面的分析可以看出，无论是前向的 PDR 还是后向的 PDR，它们都需要计算三次逼近空间，才能到达不动点。前向的 PDR 算法的每次迭代都是通过可以迁移到 $\neg P$ 的状态来生成归纳子句，从而逼近状态空间；而后向的 PDR 算法的每次迭代都是通过使用可以逆向迁移到 I 的状态来生成归纳子句，从而逼近状态空间。但是，对于一个符号化的迁移系统而言，它除了包含前向可达的状态和后向可达的状态，系统中还存在一些既不能前向可达也不能后向可达的状态，例如，图5.1 中的状态 $\neg x_1 \wedge x_2 \wedge x_3$ 以及 $x_1 \wedge \neg x_2 \wedge x_3$ 都是这种状态。由于这些状态既是前向不可达又是后向不可达的状态，因此，它们既可以用来生成相对于前向可达状态空间成归纳的短句来增强前向的可达状态空间，又可以生成相对于后向可达状态空间成归纳的短句来增强后向的可达状态空间。例如，如果算法首先探测到状态 $s = \neg x_1 \wedge x_2 \wedge x_3$ 是初始不可达的，那么利用前向 PDR 搜索就可以描述如下。

- 由于状态 $s = \neg x_1 \wedge x_2 \wedge x_3$ 是前向不可达的，因此生成任务 $\langle s, 0\rangle$ 并插入到任务队列 Q_f 中。
- 从 Q_f 中取出优先级最高的任务，也就是 $\langle s, 0\rangle$，探测 $F_0 \wedge \neg s \wedge T \rightarrow \neg s'$ 是否永真，调用 SAT 求解器可知，该公式是永真式，算法接着调用最小归纳子句生成算法，生成短句 $c = \neg x_3$，并将该短句加入到 F_1 中，更新 $F_1 = \{P, \neg x_3\}$。
- 算法继续探测 $F_1 \wedge T \rightarrow P'$ 是否永真，调用 SAT 求解器可知，公式 $F_1 \wedge T \rightarrow P'$ 不成立，因为存在状态 $s = x_1 \wedge x_2 \wedge \neg x_3$ 经过一步迁移到达 $\neg P$ 状态。因此生成任务 $\langle s, 0\rangle$ 并插入到任务队列中。

- 从 Q_f 中取出优先级最高的任务，即 $\langle s, 0\rangle$，探测 $F_0 \wedge \neg s \wedge T \rightarrow \neg s'$ 是否永真，调用 SAT 求解器可知，公式永真，算法接着调用最小归纳子句生成算法，生成短句 $c = \neg x_1$，并将该短句加入到 F_1 中，更新 $F_1 = \{P, \neg x_3, \neg x_1\}$。
- 继续探测 $F_1 \wedge T \rightarrow P'$ 是否永真，调用 SAT 求解器可知，该公式永真，算法尝试把 F_1 中的短句 c 传入到下一个逼近空间 F_2 中，探测 $F_1 \wedge T \rightarrow c'$ 是否永真。首先取第一个短句 $c = \neg x_3$，调用 SAT 求解器可知，公式 $F_1 \wedge T \rightarrow \neg x_3'$ 是永真式，因此将 $\neg x_3$ 传播到下一个逼近空间 F_2，更新 $F_2 = \{P, \neg x_3\}$，接着取下一个短句 $c = \neg x_1$，调用 SAT 求解器可知，公式 $F_1 \wedge T \rightarrow \neg x_1'$ 也是永真式，因此将 $\neg x_1$ 传播到下一个逼近状态空间 F_2 中。此时，算法发现 $F_1 = F_2$，找到一个不动点，算法终止。

通过状态 $s = \neg x_1 \wedge x_2 \wedge x_3$ 来生成归纳子句从而逼近状态空间，只需要两次逼近计算就可以到达不动点。对于后向 PDR 算法也是一样的，也只需要两次逼近计算就可以到达不动点。事实上，这是由于状态 $s = \neg x_1 \wedge x_2 \wedge x_3$ 既不是前向可达的，也不是后向可达的，因此，它既可以生成相对于前向可达状态空间的归纳子句，也可以生成相对于后向可达状态空间的归纳子句。因此，尽快发现那些既不是初始可达的状态，又不是可达 $\neg P$ 的状态对于算法的提高非常有利。基于这种思想，本文提出两种高效的逼近状态空间的算法。

本章将介绍两种双向 PDR 算法，“双向”的意思就是同时从前向和后向进行状态空间的遍历。在双向 PDR 算法中，前向 PDR 算法会维护一个前向可达状态空间的信息，后向 PDR 算法则会维护一个后向可达状态空间的信息。该算法的主要思想是不停的探测两个逼近序列是否有交集，如果有交集，即存在一个状态 s，它既在前向逼近序列中，又在后向逼近序列中，那么状态 s 仅存在四种可能情况:

- 状态 s 既是前向可达的也是后向可达的，那么一定存在一条从初始状态出发经过 s 到达 $\neg P$ 的路径，此时，我们得到一条真的反例路径。
- 状态 s 是前向可达的，但后向是不可达的，即从 I 出发，有一条可达状态 s 的路径，但从 s 出发，不能到达 $\neg P$; 那么 s 可以用来归纳增强后向逼近序列。
- 状态 s 是前向不可达，但后向是可达的，即从 I 出发不可达，而从 $\neg P$ 出发，有一条可达 s 的路径，那么 s 可以用来归纳增强前向逼近序列。
- 如果状态 s 是双向都不可达的，那么可以用它来同时归纳增强前向和后向逼近序列。

因此，对于任意一种情况而言，基于 s 的双向 PDR 搜索都会得到有用的状态空间的可达性信息。本章正是基于这种思想提出了两种双向 PDR 的算法。在具体介绍双向 PDR 算法之前，需要给出逆向的最小归纳子句生成算法

$Reverse\_MIC()$，如算法5.1 所示。这个算法与第2 章的算法2.6 一样，也是产生最小归纳子句，主要区别在于，算法5.1 是相对于逆向迁移关系的归纳子句，因此在算法的初始化阶段，首先要对模型进行“取逆”操作，即将初始条件设为 $\neg P$，将待验证的性质设为 $\neg I$，将迁移关系设为 $\overline{T}$（在符号化的迁移关系中，取逆操作是常数时间的，仅需要把当前时刻的变量与表示下一时刻的变量交换即可)。然后，与最小归纳子句生成算法的迭代过程一样，生成一个相对于当前状态空间 X 成归纳的短句 c。

算法 5.1: 逆向最小归纳子句生成算法

输入: $M = \langle S, I, T\rangle$, X, inductive clause c

输出: clause c

```
1  begin
2      Ī = ¬P;        /* 将 ¬P 设为逆向的初始条件 */
3      T̄ = reverse(T);        /* 获取逆向迁移关系 T̄ */
4      for l ∈ c do
5          ĉ := c/l;
6          while true do
7              if ¬(Ī → ĉ) then
8                  Goto 4;
9              if X ∧ ĉ ∧ T̄ → ĉ′ then
10                 c := ĉ;
11                 Goto 4;
12             else
13                 extract state s form X ∧ ĉ;
14                 q = s ⊔ ¬ĉ;
15                 ĉ := ¬q;
```

有了逆向的最小归纳子句生成算法，下面将详细介绍本文提出的两个双向的 PDR 算法。

5.2 交叠的双向 PDR 算法

交叠的双向 PDR 算法 (Interleaving Bidirectional PDR，简写为 IBPDR)，主要维护两个 PDR 过程，它们分别用来逼近前向可达状态空间以及后向可达状态空间，而这两个过程是顺序交叠进行的，因此，本文称之为交叠的双向 PDR 算法，即 IBPDR 算法。IBPDR 的整体框架如算法5.2 所示。

算法 5.2: IBPDR 算法

输入: System $M = \langle X, I, T \rangle$, safety property P

输出: safe 或者 counterexample

```
1  begin
2      vector F, B;
3      F_0 = I, F_1 = P;
4      B_0 = ¬P, B_1 = ¬I;
5      k = 1, l = 1;
6      while true do
7          choose_Interleave_Strategy(&k_inc, &l_inc);      /* 选择交叠策略 */
8          for i = 1; i ≤ k_inc; i++ do
9              F_{k+1} = P;      /* 初始化前向的最外层逼近空间为 P */
10             if ¬newStrengthen(F_k) then
11                 return counterexample;
12             if Propagate() then
13                 return safe;
14             k++;
15         for j = 1; j ≤ l_inc; j++ do
16             B_{l+1} = ¬I;      /* 初始化后向的最外层逼近空间为 ¬I */
17             if ¬newBStrengthen(B_l) then
18                 return counterexample;
19             if BPropagate() then
20                 return safe;
21             l++;
```

算法主要维护两个逼近序列：一个是前向逼近序列 (Sequence of Forward Over-approximation Reachability，简写为 SFOR)，我们用 $F_0, F_1, \ldots, F_k$ 来表示，其中每一个 F_i 表示 i 步可达的状态空间的上逼近，本文中，我们称之为 i 步前向逼近；另一个是后向逼近序列 (Sequence of Backward Over-approximation Reachability, 简写为 SBOR)，我们用 $B_0, B_1, \ldots, B_l$ 来表示，其中每一个 B_j 表示从 $\neg P$ 出发经过 j 步可达的状态空间的上逼近，同样，在本文中，我们称之为 j 步后向逼近。对于 SFOR 而言，其要满足下面四个条件：

- $F_0 = I$;
- $F_i \Rightarrow F_{i+1}$;
- $F_i \wedge T \Rightarrow F'_{i+1}$;

- $F_i \Rightarrow P, 0 \leq i \leq k$

同样，对于 SBOR 而言，其要满足下面的四个条件：

- $B_0 = \neg P$;
- $B_j \Rightarrow B_{j+1}$;
- $B_j \wedge T \Rightarrow B'_{j+1}$;
- $B_j \Rightarrow \neg I, 0 \leq i \leq l$

算法的第一步是初始化所需的数据结构，其中两个向量 F 和 B 分别用来存储两个逼近序列，初始化阶段，向量 F 和 B 分别有两个元素；整数 k 和 l 分别用来控制前向 PDR 的外层循环以及后向 PDR 的外层循环。初始化工作完成之后，算法进入主迭代过程。主迭代过程的第一步是选择一个交叠策略，IBPDR 算法支持多种交叠策略，后面会详细介绍几种交叠策略。然后算法根据选定的交叠策略顺序进行前向搜索以及后向搜索，直到某一个过程终止。主迭代过程中最重要的一个子过程就是 $newStrengthen(F_k)$ 以及 $newBStrengthen(B_l)$，它与基本的 PDR 算法不同，它需要用到已经搜索到的后向的可达状态空间信息。而传播函数与之前的传播算法是一致的，这里不再做详细的介绍。下面我们详细介绍交叠策略选择函数以及新的增强算法 $newStrengthen(F_k)$ 和 $newBStrengthen(B_l)$。

算法 5.3: $choose\_Interleave\_Strategy$ 函数

输入: $*k\_inc$, $*l\_inc$

```
1  begin
2      if random = true then
3          /* 随机交叠策略 */
4          *k_inc = rand(10);
5          *l_inc = rand(10);
6      else if user_define = true then
7          /* 用户自定义交叠策略 */
8          *k_inc = argv[1];
9          *l_inc = argv[2];
10     else
11         /* 默认交叠策略 */
12         *k_inc = 3;
13         *l_inc = 1;
```

IBPDR 中的选择交叠策略函数如算法5.3所示。IBPDR 支持三种交叠策略，分别为随机的交叠策略、用户自定义的交叠策略以及默认交叠策略。随机的交叠

策略中，利用随机函数产生两个 0 到 10 之间的整数分别用来控制前向 PDR 的步数和后向 PDR 的步数。用户自定义的交叠策略，是允许用户输入两个整数用来控制前向和后向 PDR 的步数。对于默认的交叠策略，我们选择前向执行 3 步后向执行 1 步，这是因为对于大部分的模型而言，正向的状态空间搜索效率还是要优于后向的状态空间搜索的效率。

算法 5.4: 前向增强函数

输入: F_k

输出: $true$ 或者 $false$

```
1 begin
2     while F_k ∧ B_l is SAT do
3         extract state s ∈ F_k ∧ B_l;
4         priorityQueue Q_f := ⟨s, k − 1⟩;
5         if ¬handleForwardObligation(Q_f) then
6             priorityQueue Q_b := ⟨s, l − 1⟩;
7             if ¬handBackwardObligation(Q_b) then
8                 return false;
9     return true;
```

IBPDR 中另外两个非常重要的函数是新的增强函数 $newStrengthen(F_k)$ 和 $newBStrengthen(B_l)$。首先，我们给出新的前向增强函数，如算法5.4所示。前向增强函数的主要目的是收缩前向逼近空间，尽量排除掉那些可达坏状态的状态，从而找到一个前向可达状态空间的上逼近。算法首先探测 k 步前向逼近与 j 步后向逼近是否相交，如相交为空，说明当前 k 步逼近是“安全的”逼近空间，因此增强函数返回 $true$；否则，说明 F_k 中可能会有一个状态可达坏状态，因此，算法提取出一个状态用于归纳增强当前的状态空间信息，具体执行函数为 $handleForwardObligation()$，如算法5.5所示，函数 $handleForwardObligation()$ 的主要目的是处理已生成的任务，从而收缩状态空间。若函数 $handleForwardObligation()$ 返回的是 $true$，则通过函数 $handleForwardObligation()$ 生成的归纳短句必定排除掉了包含状态 s 的很多状态。否则，则说明找到了一条从初始状态 I 到达 s 的路径，但是这条路径有可能是一条“假”反例路径，即需要进行判断 s 是否可以到达 $\neg P$，因此，在算法5.4中，一旦处理前向任务的函数 $handleForwardObligation()$ 返回 $false$，算法立即探测状态 s 是否可达 $\neg P$，即调用处理后向任务的函数 $handleBackwardObligation()$，若处理后向任务的函数也返回 $false$，算法找到一条真的反例路径，返回 $false$；

算法 5.5: $handleForwardObligation()$ 函数

输入: Q_f

输出: $true$ 或者 $false$

```
1  begin
2      pop some obligation ⟨s, i⟩ from Q_f;
3      if F_i ∧ ¬s ∧ T ∧ s' is UNSAT then
4          c := MIC(M, F_i, ¬s);
5          for 0 < j ≤ i + 1 do
6              F_j := F_j ∪ {c};
7          if i < k then
8              Q_f := Q_f ∪ {⟨s, i + 1⟩};
9      else if i = 0 then
10         return false;
11     else
12         extract a predecessor t of s;
13         Q_f := Q_f ∪ {⟨t, i − 1⟩};
14     return true;
```

否则，说明状态 s 不是后向可达的，因此，状态 s 必定可以生成相对于后向状态空间成归纳的短句，从而用来收缩后向状态空间。后向任务函数处理完任务之后，算法再次探测新的 F_k 与 B_l 状态空间是否存在共有的状态，如果存在，继续前面的迭代过程；若没有，则前向增强过程终止，算法返回 $true$。至此，整个前向增强过程就结束了，此时，k 步前向逼近与 l 步后向逼近是不相交的。算法可以执行传播过程，从而进一步收缩状态空间信息。传播过程函数与本文第2的算法2.9是一样的，这里不再详细介绍。在传播的过程中一旦发现有两个相邻的逼近空间相等，算法终止，并回答性质正确。

对于逆向的增强函数 $newBStrengthen()$，其执行过程与前向增强函数是一样的，如算法5.6 所示。后向增强的主要目的是收缩后向的逼近空间，尽量排除那些可达初始状态的状态，从而找到后向可达状态空间的不动点。在后向增强阶段，一旦发现一个状态可达 $\neg P$，算法会立即执行前向任务处理函数 $handleForwardObligation()$，判断其是否是初始可达，如果是初始可达的，说明找到一条反例路径，否则，算法会增强当前的前向状态空间，从前向状态空间中排除掉这个状态。

算法 5.6: 后向增强函数

输入: B_l

输出: $true$ 或者 $false$

```
1 begin
2     while B_l ∧ F_k is SAT do
3         extract state s ∈ B_l ∧ F_k;
4         priorityQueue Q_b := ⟨s, l − 1⟩;
5         if ¬handleBackwardObligation(Q_b) then
6             priorityQueue Q_f := ⟨s, k − 1⟩;
7             if ¬handForwardObligation(Q_f) then
8                 return false;
9     return true;
```

算法 5.7: $handleBackwardObligation()$ 函数

输入: Q_b

输出: $true$ 或者 $false$

```
1  begin
2      pop some obligation ⟨s, j⟩ from Q_b;
3      if B_j ∧ ¬s ∧ T̄ ∧ s′ is UNSAT then
4          c := Reverse_M IC(M, B_j, ¬s);
5          for 0 < i ≤ j + 1 do
6              B_i := B_i ∪ {c};
7          if j < k then
8              Q_b := Q_b ∪ {⟨s, j + 1⟩};
9      else if j = 0 then
10         return false;
11     else
12         extract a predecessor t of s;
13         Q_b := Q_b ∪ {⟨t, i − 1⟩};
14     return true;
```

后向 PDR 算法的另外一个子过程是后向传播算法，与前向传播过程一样，后向传播算法也是遍历后向逼近序列的所有子句，判断其是否可以进行向前传播，若可以，则将其传播到下一层的状态空间。事实上，传播的过程就是进一步收缩

状态空间的过程。若在传播的过程中发现两个相邻的逼近空间相等，那么算法找到一个不动点，算法终止并回答性质正确。

算法 5.8: 后向传播算法

输入: M, P

输出: $true$ 或者 $false$

```
1 begin
2     for 0 ≤ i ≤ l do
3         for each c ∈ B_i/B_{i+1} do
4             if B_i ∧ T̄ ∧ ¬c' is UNSAT then
5                 B_{i+1} = B_{i+1} ∪ {c}
6         if B_i = B_{i+1} then
7             return true;
8     return false;
```

定理 5.1: 给定系统 M 和安全性质 P，算法 IBPDR 一定会终止。

证明：根据算法的执行过程可知，算法5.2是在构造两个非减的逼近序列：$F_0, F_1, \ldots, F_k$ 和 $B_0, B_1, \ldots, B_l$，由于系统状态空间是有穷的，因此，随着算法的执行，这两个递增过程一定会终止，并且 k 和 l 的最大值不会超过状态空间数。证毕。 ■

定理 5.2: 给定系统 M 和安全性质 P，算法 IBPDR 返回 $true$ 当且仅当 $M \models P$。

证明：若算法返回 $true$，那么只有两种情况：

- 存在一个 i，使得 $F_i = F_{i+1}$ 成立；
- 存在一个 j，使得 $B_j = B_{j+1}$ 成立；

首先，如果是第一种情况，也就是存在 i 使得 $F_i = F_{i+1}$ 成立，那么由于算法要保证向前逼近序列要满足四个不变式，因此，可以得出 $M \models P$。同理可以证明，如果 $B_j = B_{j+1}$ 成立，那么 $M \models P$。

接下来，我们证明若 $M \models P$，那么算法会返回 $true$。根据定理 5.3，算法一定会终止，若算法没有返回 $true$，那么算法一定会返回一条反例路径，而这与 $M \models P$ 矛盾。证毕。 ■

5.3 并行的双向 PDR 算法

IBPDR 算法通过维护两个逼近序列分别从前向和后向逼近可达状态空间。本节将介绍另外一种双向 PDR 算法，称为并行的双向 PDR 算法 (Parallel Bidirectional PDR，简写为 PBPDR)。

算法 5.9 给出了 PBPDR 的主要框架。PBPDR 的主要思想是维护两个逼近序列 $F_0, F_1, \ldots, F_k$ 和一个后向逼近序列 $B_0, B_1, \ldots, B_l$，它们分别表示前向可达状态空间的逼近序列和后向可达状态空间的逼近。算法探测当前两个状态空间是否有交集，即是否存在状态既在 F_k 中又在 B_l 中，若存在，则算法利用这个状态使用两个线程同时增强前向和后向逼近序列；否则，则同时对这两个逼近序列执行传播过程，直到算法终止。

由于两个逼近序列之间需要做信息交互，因此这两个逼近序列可以被两个线程访问，但是仅有一个线程可以做写操作。比如，对于前向逼近序列 SFOR，线程 $t2$，即后向 PDR 可以访问它，但不能修改它。同理，对于后向逼近序列 SBOR，线程 $t1$ 只能访问，不能修改。同时算法还维护两个带优先级的任务链表，分别为 Q_f 和 Q_b。这两个任务链表存储每一次迭代时的需要处理的任务。

由于数据可以被两个线程访问，因此需要互斥量来保证它们的访问顺序。在这里，我们设模型 M，性质 P 以及四个互斥量是全局变量，也就是在线程的任意地方都能访问。

算法 5.9: 双向并行 PDR 算法

输入: System $M = \langle X, I, T \rangle$, safety property P

输出: $safe$ 或者 $counterexample$

```
1  begin
2      vector F, B;
3      priorityQueue Q_f, Q_b;
4      thread t1 = new FPDR(&F, &B, &Q_f, &Q_b);
5      thread t2 = new BPDR(&F, &B, &Q_f, &Q_b);
6      while t1.alive() ∧ t2.alive() do
7          sleep(10);
8      if t1.alive() then
9          return t2.result;
10     if t2.alive() then
11         return t1.result;
```

5.3.1 前向 PDR 算法 (FPDR)

PBPDR 算法使用两个线程分别执行前向 PDR 和后向 PDR，但是其前向和后向算法之间的信息交互非常重要，不是简单的把两个过程并行起来，因此，这里的前向 PDR 算法和基本的 PDR 算法有很大的不同。算法 5.10给出了 FPDR 基本框架。

算法 5.10: FPDR 算法

输入: System F, B, Q_f, Q_b

输出: $safe$ 或者 $counterexample$

```
1 begin
2     k := 1;
3     while true do
4         F_{k+1} = P;
5         /* B_l 表示后向逼近的最外层逼近空间 */ if
          ¬ParallelForwardStrengthen(F_k, B_l, Q_f, Q_b) then
6             return counterexample;
7         if ParallelForwardPropagate() then
8             return safe;
9         k++;
```

FPDR 算法同样包括两个子过程，分别为前向增强子过程和前向传播子过程，在算法 5.10 中分别对应两个函数 $ParallelForwardStrengthen()$ 以及 $ParallelForwardPropagate()$。前向增强子过程的目标是收缩前向逼近序列，同时往后向 PDR 维护的任务队列 Q_b 插入新的任务。

前向增强算法首先用 SAT 求解器探测当前的前向状态空间逼近和后向状态空间逼近是否相交。如果两个状态空间相交，设这个状态为 s，说明经过 s 可能存在一条反例路径。此时，算法试图从前后双向证明这个状态 s 是不可达的，因此，算法第一步会将 s 作为种子生成两个任务对 $\langle s, k-1\rangle$ 和 $\langle s, l-1\rangle$，分别加入到两个任务链表 Q_f 和 Q_b 中。注意，由于 Q_f 和 Q_b 是共享数据，在每次对它进行写操作时，当前线程需要获得其对应的互斥锁。产生了种子之后，算法进入内层迭代过程，该迭代过程试图去完成队列 Q_f 的所有任务，也就是去证明所有任务中的状态都不是初始可达的。该迭代过程如下进行：从队列中取出优先级最高的任务，设为 $\langle s, i\rangle$，首先判断其短句 s 相对于 i 步正向逼近 F_i 是否为归纳的。如果是归纳的，则调用最小归纳子句生成算法生成一个 $c \subseteq \neg s$ 使得 $F_i \wedge T \rightarrow c'$ 成立。

算法 5.11: 并行前向增强函数

输入: F_k, B_l, Q_f, Q_b

输出: $true$ 或者 $false$

```
1  begin
2      while F_k ∧ B_l is SAT do
3          extract state s;
4          lock_mutex_Q_f(); Q_f := Q_f ∪ {⟨s, k−1⟩}; unlock_mutex_Q_f();
5          lock_mutex_Q_b(); Q_b := Q_b ∪ {⟨s, l−1⟩}; unlock_mutex_Q_b();
6          while Q_f ≠ ∅ do
7              pop some ⟨s, i⟩ from Q_f;
8              if F_i ∧ T ∧ s′ is UNSAT then
9                  c := MIC(M, F_i, ¬s)
10                 for 0 ≤ j ≤ i + 1 do
11                     F_j := F_j ∪ {c};
12                 if i < k then
13                     lock_mutex_Q_f(); Q_f := {⟨s, i+1⟩}; unlock_mutex_Q_f();
14             else if i = 0 then
15                 extract the path from s to the state t in F_k;
16                 if t with back_reachable_flag then
17                     return false;
18                 lock_mutex_Q_b(); Q_b := {⟨t, l−1⟩}; unlock_mutex_Q_b();
19                 lock_mutex_Q_f();
20                 delete ⟨s, 0⟩, ..., ⟨t, i−1⟩ from Q_f;
21                 unlock_mutex_Q_f();
22             else
23                 extract a predecessor state t;
24                 lock_mutex_Q_f(); Q_f := {⟨t, k−1⟩}; unlock_mutex_Q_f();
25     return true;
```

将归纳短句 c 加入到小于 i 的所有的 F_j 中。此时，如果 $i < k$，说明，状态 s 还会出现在 F_k 中，因此，加入生成新的任务 $\langle s, i+1\rangle$。如果此时 $i = 0$，说明找到了一条从 s 出发可达 k 步正向逼近 F_k 的一条路径。因此，可以提取出一个 F_k 中的状态 t，它反向可达初始状态 s。这时，算法找到了一条正向初始可达 B_l 的状态，但是，由于后向 PDR 在同时进行，这个状态 t 可能已经被证明可达 $\neg P$，算法首先会检测其是否被置上了后向可达的标签，也就是 $back\_reachable\_flag$ 是否

为 $true$。如果为真，那么算法找到了一条反例路径，算法返回这条反例路径并终止。否则，说明这个状态还没有被后向搜索过程探测，因此将状态 t 标记上初始可达的标签 $forward\_reachable\_flag$，并将该状态生成的任务插入到后向搜索的任务链表 Q_b 中。同时，由于前向搜索被这条初始可达的路径阻塞了，如果不把它排除掉，算法将无法继续，因此，我们将删除掉前向搜索的任务队列 Q_f 中状态从 s 出发可达状态 t 的所有状态形成的任务。前向搜索继续迭代过程。如果任务 $\langle s, i\rangle$ 中 $i \neq 0$，那么算法将探测 F_{i-1} 中是否存在一个可达 s 的状态，如果存在，提取这个状态 t，生成新的任务 $\langle t, i-1\rangle$ 插入到前向搜索的任务链表 Q_f。迭代过程继续进行，直到 $F_k \wedge B_l$ 为空。

算法 5.12: 并行前向传播算法

输入: F

输出: $true$ 或者 $false$

```
1 begin
2     for 0 ≤ i ≤ k do
3         for each c ∈ F_i/F_{i+1} do
4             if F_i ∧ T ∧ ¬c' is UNSAT then
5                 F_{i+1} = F_{i+1} ∪ {c}
6         if F_i = F_{i+1} then
7             return true;
8     return false;
```

当 $F_k \wedge B_l$ 为空时，说明 F_k 中已经不存在可达 $\neg P$ 的状态，此时，算法将执行传播过程，也就是对于 i 步正向逼近 ($i \leq k$) 中的所有归纳子句 c，探测其是否可以传播到 $i+1$ 步正向逼近。在传播的过程中，一旦发现某个 j 使得 $F_j = F_{j+1}$ 成立，那么算法就找到了一个不动点，算法将回答待验证的性质 P 安全并终止。

5.3.2 后向 PDR 算法 (BPDR)

后向 PDR 算法跟前向 PDR 算法的框架基本相似，后向搜索是基于逆向迁移关系，逆向的初始状态和待验证性质的。算法 5.13 给出了 BPDR 的实现框架。算法首先是获取逆向的迁移关系 $\overline{T}$。它将原始模型 M 中所有迁移关系取逆关系，也就是对于任意的 $(s_i, s_{i+1}) \in T$ 当且仅当 $(s_{i+1}, s_i) \in \overline{T}$。后向 PDR 算法的目标是求解一个包含从 $\neg P$ 出发的逆向可达状态空间的逼近不动点。同样，该算法的终止条件有两个，第一个就是找到了一条从 $\neg P$ 出发逆向可达 I 的路径，此时，找到

一条反例，算法终止；第二个终止条件就是算法找到了一个不动点，性质正确，算法终止。

算法 5.13: BPDR 算法

输入: System F, B, Q_f, Q_b

输出: $safe$ 或者 $counterexample$

```
1 begin
2     l := 1;
3     while true do
4         B_{l+1} = ¬I;
5         /* F_k 表示后向逼近的最外层逼近空间 */ if
          ¬ParallelBackwardStrengthen(F_k, B_l, Q_f, Q_b) then
6             return counterexample;
7         if ParallelBackwardPropagate() then
8             return safe;
9         l++;
```

该算法的主要迭代过程同样分为两部分，第一部分称之为后向增强算法，第二部分称之为后向传播阶段。后向增强阶段主要是根据性质精化逆向可达状态空间的信息，根据已有信息排除所求不动点中不可能包含的状态。算法的后向传播阶段是将探测关于已有的逆向可达状态空间的信息是否已经足够得到一个不动点，如果有，则算法终止，否则，算法继续探测更大的状态空间。

后向增强算法如5.14 所示，算法不停的探测当前后向可达的逼近信息是否包含前向可达的状态，如果存在，算法试图证明这个状态是不能到达 $\neg P$ 的。如果找到了一条从这个状态出发可达 $\neg P$ 的路径，说明从这个状态出发存在一条可达 $\neg P$ 的后向可达路径，此时，算法要判断该状态是否也是前向可达的，如果这个状态的前向可达标志 $forward\_reachable\_flag$ 被置为 $true$，那么算法找到一条真实的反例路径。如果算法没有找到这么一条路径，说明该状态还没有被前向探测过，因此生成针对该状态的前向搜索任务 $\langle s, k-1\rangle$，并将其插入到前向搜索队列中。然后从后向搜索任务链表里删除掉所有的从 s 到 $\neg P$ 路径上的状态对应的任务。算法继续进行迭代过程。如果算法没有找到一条从 s 到达 $\neg P$ 的路径，说明在后向逼近序列中一定存在某个 i 步后向逼近 B_i 使得 $B_i \wedge \overline{T} \wedge s'$ 不成立。因此，可以生成一个归纳短句用来加强后向逼近序列。

当 $B_l \wedge F_k$ 不可满足时，说明当前的后向逼近序列中已经不包含前向逼近序列中的状态，此时算法将转入到下一阶段，也就是传播阶段，算法5.15 给出了后

算法 5.14: 并行后向增强函数

输入: F_k, B_l, Q_f, Q_b

输出: $true$ 或者 $false$

```
1  begin
2      while B_l ∧ F_k is SAT do
3          extract state s;
4          lock_mutex_Q_b(); Q_b := Q_b ∪ {⟨s, l−1⟩}; unlock_mutex_Q_b();
5          lock_mutex_Q_f(); Q_f := Q_f ∪ {⟨s, k−1⟩}; unlock_mutex_Q_f();
6          while Q_b ≠ ∅ do
7              pop some ⟨s, i⟩ from Q_b;
8              if B_i ∧ ¬T ∧ s′ is UNSAT then
9                  c := Reverse_M IC(M, B_i, ¬s)
10                 for 0 ≤ j ≤ i + 1 do
11                     B_j := B_j ∪ {c};
12                 if i < k then
13                     lock_mutex_Q_b(); Q_b := {⟨s, i+1⟩}; unlock_mutex_Q_b();
14             else if i = 0 then
15                 extract the path from s to the state t in B_l;
16                 if t with forward_reachable_flag then
17                     return false;
18                 lock_mutex_Q_f(); Q_f := {⟨t, k−1⟩}; unlock_mutex_Q_f();
19                 lock_mutex_Q_b();
20                 delete ⟨s, 0⟩, ..., ⟨t, l−1⟩ from Q_b;
21                 unlock_mutex_Q_b();
22             else
23                 extract a predecessor state t;
24                 lock_mutex_Q_f(); Q_f := {⟨t, i−1⟩}; unlock_mutex_Q_f();
25     return true;
```

向传播的伪码描述。同向前传播一样，对于任意的 j 步后向逼近 B_j，算法将检测 B_j 中的所有短句 c，判断其是否可以传播的上一层，判断的条件就是 $B_j \wedge \overline{T} \rightarrow c'$ 是否成立，如果成立，将 c 传播到 B_{j+1}，在传播的过程中，如果发现 $B_j = B_{j+1}$，那么算法找到一个不动点，算法终止。否则，算法继续探测新的更大的状态空间，也就是 B_{l+1}。

算法 5.15: 并行后向传播算法

输入: B

输出: $true$ 或者 $false$

```
1 begin
2     for 0 ≤ i ≤ l do
3         for each c ∈ B_i/B_{i+1} do
4             if B_i ∧ T̄ ∧ ¬c' is UNSAT then
5                 B_{i+1} = B_{i+1} ∪ {c}
6         if B_i = B_{i+1} then
7             return true;
8     return false;
```

5.3.3 算法正确性

定理 5.3: 给定系统 M 和安全性质 P，算法 PBPDR 一定会终止。

证明：算法 PBDPR 终止的条件是 FPDR 和 BPDR 都会终止。首先证明算法 FPDR 会终止。FPDR 会构造一个向前逼近序列 $F_0, F_1, \ldots, F_k$，从构造过程可以发现，其是一个单调非减的状态集合。由于系统 M 的状态是有穷的，因此，算法一定会终止且 k 不会超过系统的状态数。同理可证明算法 BPDR 也会终止。证毕。■

定理 5.4: 给定系统 M 和安全性质 P，算法 PBPDR 返回 $true$ 当且仅当 $M \models P$。

证明：若算法返回 $true$，那么只有两种情况：

- 存在一个 i，使得 $F_i = F_{i+1}$ 成立；
- 存在一个 j，使得 $B_j = B_{j+1}$ 成立；

首先，如果是第一种情况，也就是存在 i 使得 $F_i = F_{i+1}$ 成立，那么由于算法要保证向前逼近序列要满足四个不变式，因此，可以得出 $M \models P$。同理可以证明，如果 $B_j = B_{j+1}$ 成立，那么 $M \models P$。

接下来，我们证明若 $M \models P$，那么算法会返回 $true$。根据定理 5.3，算法一定会终止，若算法没有返回 $true$，那么算法一定会返回一条反例路径，而这与 $M \models P$ 矛盾。证毕。 ■

5.4 优化方法

本节将介绍几种用于上述两个算法的优化技术。这些优化技术都是通用的优化技术，可以适用于上述两个算法。

5.4.1 结合 BMC 技术的优化策略

如前所述，BMC 技术的优势就是在于发现系统的错误，尤其是错误出现的位置比较浅时，它的局限在于当展开步数 k 过大时，会导致 SAT 求解器无法求解，但是在 k 相对比较小时，BMC 的效率还是很高的。本算法结合 BMC 算法对一些细节进行了优化，首先，在 $F_k \wedge B_l$ 成立时，也就是存在状态 s 即在 k 步前向逼近，又在 l 步后向逼近时，我们会将该状态 s 作为怀疑对象同时加入到前向搜索任务链表 Q_f 和后向搜索任务链表 Q_b 中，如前面所分析的，这个状态只有 4 种情况出现：

1. 初始可达，逆向初始也可达；
2. 初始可达，逆向初始不可达；
3. 初始不可达，逆向初始可达；
4. 初始不可达，逆向初始也不可达；

如果是第一种情况，那么系统经过状态 s 一定存在一条真实的反例路径，此时，若反例路径的长度不大，那么 BMC 技术将很适合求解，尤其是算法是从中间向两边求解，BMC 展开 k 步的求解难度远比求解 $2k$ 步的难度要小。如果是第二种情况，如果状态 s 是初始可达的且是逆向初始不可达的，那么 s 就是 M 中的一个状态，该状态可以用来加强后向逼近序列 BSOR，但是不能增强前向逼近序列，因此，尽早发现这样的状态，就可以尽早排除其加入向前搜索的任务链表中，从而减少不必要的开销。因此，在步数比较小的情况下，使用 BMC 展开可以尽早的发现这样的状态。同理，对于第三种情况也是一样的。至于最后一种情况，如果双向都不可达，这样的状态是最好的情况，这样就可以同时对后向搜索逼近序列和前向搜索逼近序列进行增强。因此，用 BMC 先去探测一遍，可以保证状态 s 距离初始状态集合和 $\neg P$ 状态集合尽量的远。

BMC 技术还有一个地方可以用到。在算法 FPDR 的向前搜索过程中，一旦发现 F_k 中的一个状态 s 是初始可达的，那么算法不会直接把状态 s 生成后向搜索的一个任务加入到 Q_b 中，而是，首先探测该状态是否还在当前的 B_l 中，由于后向搜索过程同时在进行，因此，该状态可能已经被 B_l 排除了，如果已经被排除了，该状态就是一个初始可达的状态。可以继续进行向前搜索，如果该状态还在 B_l 中，那么算法将从该状态出发 BMC 展开一定的步数 n 来判断其能否到达 $\neg P$，

如果可以，那么算法找到一条反例路径，且反例路径的长度为 $k+n$。否则，生成一个向后的任务 $\langle s, l-1\rangle$，同时将状态 s 的向前可达标签置为 $true$。

因此，算法根据系统状态空间的大小会基于这个状态做一定步数的 BMC 展开，我们默认的 BMC 展开是 10。

5.4.2 确定两个任务链表 Q_f 和 Q_b 中任务优先级的策略

第二个优化策略就是关于任务优先级的确定。我们使用的策略是，当从 Q_f 取任务时，优先选择后向可达标签为 $true$ 的任务。这意味着该任务已经被证明是后向可达的，那么该状态只有两种可能，第一种可能就是前向也可达，那么我们优先选择这个状态，就可以尽早的发现反例路径；第二种可能就是这个状态不是初始可达的，那么基于这个状态也能尽快的生成归纳短句用来加强前向逼近序列。如果当前任务链表中没有后向可达标签置为 $true$ 的任务，那么优先选择距离初始状态最近的状态，也就是 $\langle s, i\rangle$ 中 i 的值越小，任务优先级就越高。

5.4.3 启发式的归纳子句生成算法

在 MIC 算法中，每次迭代选择丢掉哪一个文字非常影响算法的效率。在基本的 MIC 算法中，文字会被赋予不同优先级，优先级越高的文字，在下次迭代时会先丢掉。影响文字优先级的主要因素就是文字在已有归纳短句中出现的频率。直观的讲，就是文字在已有的归纳短句中出现的频率越高，其优先级就越低。这是不难理解的，因为挖掘出来的归纳短句在一定程度上表示了可达状态空间的信息。而在双向 PDR 算法中，我们不仅能够得到前向可达的状态空间信息，还可以得到后向可达状态空间的信息，因此我们可以利用更多的信息来确定文字的优先级。对于任意的文字，我们都赋予一个权值，文字在前向短句中出现权值就加 1，在后向短句中出现就减 1，这样排序下来，在正向的 MIC 算法中，权值越低的，其优先级就越高。举个例如，若文字 l_1 在前向逼近序列的短句中出现 3 次，而在后向逼近序列中其没有出现，则文字 l_1 的权值就为 3，文字 l_2 在前向逼近序列短句中出现 4 次，而在后向逼近序列中其也出现了 2 次，那么其权值就为 2，此时，MIC 算法在选择优先丢掉文字时，会优先选择 l_2，而在 Reverse_MIC 算法中，文字 l_1 的优先级要高于文字 l_2。

5.5 实验结果

为了评估算法的效率，我们实现了一个原型工具 Reach，具体的工具实现细节，我们将在本文的第6 章给出详细的介绍，本节主要给出算法的实验结果。我们的实验数据包括两部分，一部分来自于 2011 年的硬件模型检验大赛

（HWMCC2011）safety 组的 benchmark，它包含 405 个实例；另一部分来自于 Intel 提供的用于硬件模型检验大赛的 benchmark，包含 60 个实例。

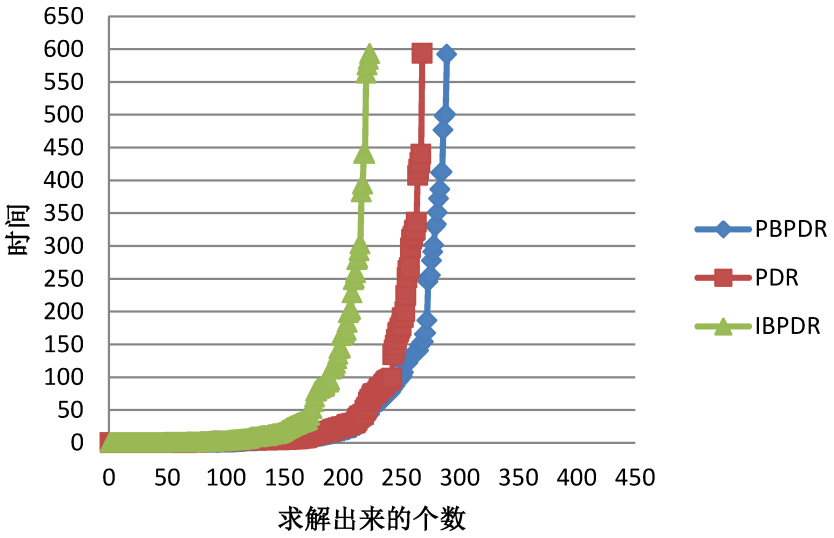


图 5.3　PDR、IBPDR 和 PBPDR 的实验结果

首先，图5.3 给出 PDR、IBPDR 以及 PBPDR 算法在 HWMCC2011 给出的 405 个实例的的对比实验结果。在本实验中，我们将时间设置为 600s，即在 600s 内如果不能给出验证结果，我们将继续验证下一个模型。其中，IBPDR 的交叠策略为随机策略。从实验结果，可以看出，在同一配置环境下，PBPDR 的效率最高，它能在 600s 内验证出更多的性质，而采用随机交叠的 IBPDR，其结果并没有明显提高 PDR 的性能。IBPDR 算法没有提高 PDR 算法的一个原因可能是其在对某一个搜索方向做归纳增强时，另一个方向的搜索必须停止等待，而等待的时间比较长就会导致 IBPDR 算法效率有所下降。而 PBPDR 不存在这一情况，对于 PBPDR 而言，前向和后向归纳增强可以同时进行，而它们之间唯一共用的信息就是任务序列。从实验也可以观察到，PBPDR 算法确实优于 PDR 算法和 IBPDR 算法。

第二组实验，我们比较了 Intel 提供的 60 个案例的实验结果，如表5.5 所示。在本组实验中，我们比较了四个算法，分别为标准 PDR 算法、逆向的 PDR 算法（Reverse PDR 算法）、IBPDR 算法以及 PBPDR 算法。我们将时间设置为 1 个小时。本组实验给针对每一个算法，我们出了两个结果，一个是验证结果（对应表中的“结果”栏）；另一个是验证花费的时间（对应表中的“时间”栏）。从表中可以得出以下结论：

- 正向的 PDR 算法相对于逆向的 PDR 算法，其求解出来的 benchmark 更多，这说明一般情况下，正向的 PDR 算法相对于逆向的 PDR 算法表现要好一些。

- IBPDR 算法虽然在求解个数上没有正向的 PDR 算法好，但是其在平均时间开销上相对于正向 PDR 算法要有优势。
- PBPDR 算法无论是在求解个数上还是在平均求解时间上都是最好的。

表 5.1　Intel 案例实验结果

模型	PDR		Reverse PDR		IBPDR		PBPDR	
	结果	时间	结果	时间	结果	时间	结果	时间
Intel001	0	0.02	0	0.11	0	0.1	0	0.02
Intel002	0	0.04	0	10.54	0	0.2	0	0.21
Intel003	0	0.16	0	10.27	0	0.26	0	2.22
Intel004	0	0.2	0	54.43	0	0.33	0	4.14
Intel005	0	1.14	0	269.28	0	0.77	0	109.07
Intel006	0	17.02	-	-	0	5.62	0	21.89
Intel007	0	78.79	-	-	0	46.29	0	97.12
Intel009	-	-	-	-	-	-	1	3408.74
Intel010	-	-	-	-	-	-	-	-
Intel011	-	-	-	-	-	-	-	-
Intel012	-	-	-	-	-	-	-	-
Intel013	-	-	-	-	-	-	-	-
Intel014	-	-	-	-	-	-	-	-
Intel015	-	-	-	-	-	-	-	-
Intel016	-	-	-	-	-	-	-	-
Intel017	1	821.6	-	-	1	629.52	1	38.25
Intel018	-	-	-	-	-	-	-	-
Intel019	-	-	-	-	-	-	-	-
Intel020	0	1491.35	-	-	0	834.05	0	985.21
Intel021	-	-	-	-	-	-	-	-
Intel022	-	-	-	-	-	-	-	-
Intel023	-	-	-	-	-	-	-	-
Intel024	0	1629.41	-	-	-	-	0	2034.12
Intel025	-	-	-	-	-	-	-	-

Intel 案例实验结果　接下页

接上页	Intel 案例实验结果							
	PDR		Reverse PDR		IBPDR		PBPDR	
模型	结果	时间	结果	时间	结果	时间	结果	时间
Intel026	0	157.3	-	-	0	274.91	0	243.7
Intel027	-	-	-	-	-	-	-	-
Intel028	-	-	-	-	-	-	-	-
Intel029	-	-	-	-	-	-	-	-
Intel030	-	-	-	-	-	-	-	-
Intel031	0	3447.13	-	-	-	-	0	3021.12
Intel032	-	-	-	-	-	-	-	-
Intel033	-	-	-	-	-	-	1	1260.03
Intel034	-	-	-	-	0	3554.89	0	3498.57
Intel035	-	-	-	-	-	-	1	764.16
Intel036	-	-	-	-	-	-	1	2643.42
Intel037	0	179.05	-	-	0	410.7	0	1030.63
Intel038	-	-	-	-	-	-	-	-
Intel039	-	-	-	-	-	-	-	-
Intel040	-	-	-	-	-	-	-	-
Intel041	-	-	-	-	-	-	1	2057.34
Intel042	-	-	-	-	-	-	-	-
Intel043	-	-	-	-	-	-	1	366.48
Intel044	1	674.27	-	-	1	419.69	1	1073.1
Intel045	1	830.5	-	-	1	547.43	1	908.91
Intel046	1	804.88	-	-	1	408.25	1	1904.23
Intel047	1	1153	-	-	1	562.48	1	1670.56
Intel048	-	-	-	-	-	-	-	-
Intel049	0	2.78	-	-	0	1.87	0	2765.29
Intel052	0	0.32	0	9.15	0	0.48	0	2.84
Intel054	0	152.79	-	-	0	125.94	0	198.12
Intel055	0	29.46	-	-	0	10.92	0	78.35
Intel056	0	26.51	-	-	0	16.76	0	19.12
Intel057	0	58.89	-	-	0	30.71	0	43.12
Intel059	0	20.7	-	-	0	11.75	0	19.34
Intel 案例实验结果								接下页

接上页	Intel 案例实验结果							
模型	PDR		Reverse PDR		IBPDR		PBPDR	
	结果	时间	结果	时间	结果	时间	结果	时间
Intel062	0	1287.52	-	-	0	1133.09	0	2043.47
Intel063	0	1.25	-	-	0	3.54	0	12.1
Intel064	-	-	-	-	-	-	-	-
Intel065	-	-	-	-	-	-	-	-
Intel066	-	-	-	-	-	-	-	-
Intel067	-	-	-	-	-	-	-	-
总共	PDR		Reverse PDR		IBPDR		PBPDR	
	求解个数	平均时间	求解个数	平均时间	求解个数	平均时间	求解个数	平均时间
	26	2374.43	7	3185.9	25	2260.50	33	2158.75

5.6 本章总结

制约限界模型检验技术应用的一个主要瓶颈就是它的非完全性，它不能证明给定的模型是否满足给定的性质。研究人员为此提出了诸多技术用以保证技术 SAT 模型检验的完全性，而最直接的方法就是给出一个完备阈值，使得限界模型检验达到该阈值时，算法就可以保证性质正确。但是，求解一个最优的完备阈值的复杂度与模型检验的复杂度是一样的。因此，该技术并不实用。

线性时序逻辑的模型检验问题都能转化成公平路径查找问题，而公平路径查找问题又都可以转化为若干个可达性问题。因此，另外一种保证基于 SAT 符号化模型检验技术的完全性的方法就是开发高效的基于 SAT 的可达性算法，从而将基于 SAT 的模型检验问题转化为基于 SAT 的可达性问题。

本章提出了两个基于 SAT 可达性的算法。两个算法都是基于 PDR 算法实现的，与 PDR 不同的是，这两个算法都是通过双向逼近可达状态空间来寻找不动点。实验表明，本章提出的算法非常高效实用。

第六章　工具实现

模型检验技术的最终目的在于应用，即：能够根据算法设计出对应计算机软硬件进行辅助验证的自动化工具。一方面，它是将具体的理论 / 技术转化为实际应用的体现；另一方面，它也是对理论 / 技术可行性以及效率的实际检验。因此，本章将给出两个工具的具体实现，第一个工具是基于 ENuSMV[90] 实现的 ENuSMV1.2，在该工具中，我们主要实现了本文第3章和第4章提出的算法，包括 CEPRE技术、基于语义的增量式的 LTL 限界模型检验算法、ETL$_{l+f}$ 的 tableau 构造算法以及基于语义的限界模型检验算法。

第二个工具 Reach，主要实现了本文第5章提出的 IBPDR 以及 PBPDR 算法，除了这两个算法之外，在该工具中，我们还实现了一些基本的基于 SAT 的符号化模型检验算法，包括基于语法的限界模型检验算法[1]、k 步归纳算法[102]、基于插值的符号化模型检验算法[103] 以及基本的 PDR 算法[144]。

本章将分别介绍这两个工具的具体实现并介绍其使用方法。

6.1　工具实现

6.1.1　ENuSMV1.2

6.1.1.1　ENuSMV1.2 的逻辑架构

ENuSMV1.2 是在 ENuSMV 的基础上实现的，它们都是基于 NuSMV 的，本文的第1章简单的介绍了 NuSMV 工具，本节将详细分析其主要实现模块及 ENuSMV1.2 新添加的模块。图6.1给出 ENuSMV1.2 的逻辑架构。

NuSMV 是使用 C 语言开发的一个优秀的模型检验工具，其提供了一个非常好的系统架构，开发人员很容易对其进行扩充以实现新的功能。在图6.1中，除去蓝色方框表示的模块就是完整的 NuSMV 的逻辑架构，首先，我们简单的介绍一下它包含的基本模块。

- **模型和规约解析模块** 该模块主要负责对输入的模型和规约做解析，生成 NuSMV 内部的数据结构表示。NuSMV 的输入格式为`.smv`文件，其语法定义可以参考 NuSMV 手册[152]。ENuSMV1.2 对该模块也进行了扩充，使其支持 ETL$_{l+f}$。
- **平坦化、布尔编码模块** 顾名思义，该模块的主要功能是将`.smv`格式的描述转化成等价的布尔函数表示。具体分两步，首先将层次结构的模型和规约转化为一个等价的无层次结构的模型和规约，即平坦化过程；接着将模型和规约中的非布尔类型变量转化成布尔变量，得到相应的布尔函数表示。

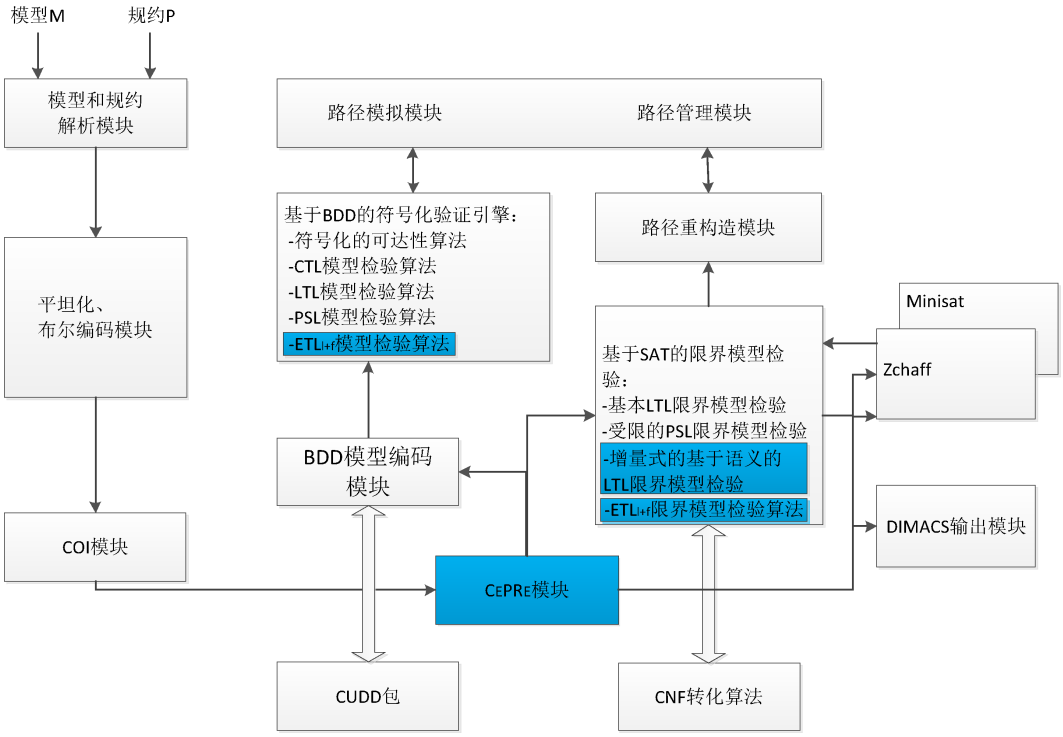


图 6.1　ENuSMV1.2 逻辑架构

- COI **模块** 这一模块是符号化模型检验中非常重要的一种优化技术，这一步主要是把待验证的规约限制在模型的相关部分进行，尽量避免探索整个模型的状态空间，可以有效的缓解状态空间爆炸问题。
- BDD **模型编码模块** 该模块主要是借用 BDD 的实现包（如 CUDD[153] 包）来编码布尔函数，然后将编码后的 BDD 数据结构提供给基于 BDD 的符号化模型检验引擎用来进行验证。
- **基于 BDD 的符号化验证引擎** 该模块是 NuSMV 的核心模块，其实现了多种基于 BDD 的符号化模型检验算法，其中每一个模型检验算法都可以视为一个独立的模块，在这里，为了方便起见，我们将所有的算法放到了一个模块中。目前，最新版本的 NuSMV2.5.4 主要支持基于 BDD 的可达性算法、CTL 的符号化模型检验算法、LTL 的符号化模型检验算法以及受限的 PSL 符号化模型检验算法。
- **路径模拟模块** 该模块主要提供反例路径信息，一旦模型检验发现错误，其会根据已有的验证信息生成一条反例路径，该模块主要负责这个功能。
- **基于 SAT 的限界模型检验** 该模块是 2002 年 NuSMV2.0 之后新增加的模块，其主要是集成了各种基于 SAT 的限界模型检验引擎。到目前为止，NuSMV 中基于 SAT 的限界模型检验主要支持基本的 LTL 限界模型检验算法以及受限的 PSL 模型检验算法。实现该模块的功能还要依赖于两个额外的模块：

CNF **转化算法模块**、SAT **求解器模块**，其中 CNF 转化算法主要是将一般的布尔公式转化为 CNF 表示，SAT 求解器是限界模型检验的底层引擎，目前 NuSMV 主要使用两个 SAT 求解器：Minisat 和 zChaff。该模块涉及的另外一个模块是 DIMACS **输出模块**，该模块主要是将布尔编码输出为 SAT 的标准格式，以方便其他 SAT 求解器求解，因此，NuSMV 事实上可以支持更多的 SAT 求解器。DIMACS **输出模块**另外一个非常重要的功能就是，它提供了一种使 NuSMV 与外界交互的一个接口，事实上，我们后面 Reach 工具的输入就依赖于这个模块的输出。

- **路径重构造模块** 该模块的主要功能就是根据 SAT 求解器返回的可满足指派生成模型中的一条路径，一般用于反例路径的生成。**路径管理模块**主要是管理当模型包含多个性质时，产生多条反例的情况。

ENuSMV1.2 在 NuSMV 的基础上主要实现了四个模块（蓝色方框标志的模块)。事实上，为了支持这四个模块实现的功能，我们还对其他的模块做了很多修改，比如，针对 ETL_{l+f} 的语法解析，我们修改了**模型和规约解析模块**以及**平坦化、布尔编码模块**。由于没有增加新的模块，在这里，我们并没有用蓝色的方框标出仅修改的模块。下面我们简单的介绍一下这四个模块的主要功能。

- CePRe 模块该模块实现了本文第3章第3.2节提出的 CePRe技术，该模块添加在布尔编码模块之后，它同样依赖于 SAT 求解器模块，并且为基于 BDD 的符号化验证引擎以及基于 SAT 的限界模型检验验证引擎提供优化技术。
- **增量式的基于语义的** LTL **限界模型检验模块** 该模块主要实现了本文第3章第3.3节提出的算法，事实上，该模块提供了一种通用的增量式的基于语义的编码框架，基于语义的 ETL_{l+f} 的限界模型检验编码同样可以使用该模块得到增量式的编码。
- ETL_{l+f} **模型检验算法** 该模块依赖于 BDD 编码实现了基于 BDD 的符号化的 ETL_{l+f} 模型检验算法，该模块另外一个很重要的功能就是实现了 ETL_{l+f} 的 tableau 构造，这里没有显示的标识出来，但是它为基于 SAT 的 ETL_{l+f} 的限界模型检验算法提供了底层的支持。
- ETL_{l+f} **限界模型检验算法** 该模块主要是实现了本文第4章第4.2提出的基于语义的 ETL_{l+f} 限界模型检验，该模块同样依赖于 SAT 求解器，因此，我们将该模块放在基于 SAT 的限界模型检验引擎中。

6.1.1.2 ENuSMV1.2 的语法扩展

刘万伟在其博士论文[90] 中已经详细介绍了 NuSMV 的语法以及 ENuSMV 新增加的语法，本文实现的 ENuSMV1.2 在语法上完全兼容 ENuSMV 和 NuSMV，

同时，为了使 ENuSMV1.2 支持 ETL_{l+f} 符号化模型检验，我们新增加了一些语法元素。

在 ENuSMV1.2 中，允许用户通过描述 Looping 自动机或者 Finite 自动机的方式自定义时序连接子，而 ENuSMV 中，仅支持 Finite 自动机作为时序连接子，因此，为了区分 Looping 自动机以及 Finite 自动机，ENuSMV1.2 中新增加了两个关键字 `FIN`和 `LOOP`，分别表示 Finite 自动机和 Looping 自动机。其他语法元素与 ENuSMV 中语法保持一致。下面我们以一个简单的示例来介绍一下 ENuSMV 新增加的语法元素。

图6.2 给出了自动机连接子 `A`的定义。其中 `CONNECTIVE`是一个关键词，它定义了一个自动机连接子，紧跟在其后的 `A(a1, a2): FIN` 则分别表示自动机的名字（`A`），自动机的字母表（`(a1,a2)`）以及自动机的类型（`FIN`）。该示例中的自动机定义说明，该自动机是一个字母表为 `(a1,a2)`的 Finite 自动机。关键字 `STATES`则引到了自动机状态集合的定义，紧跟其后标志符 `>q1`、`q2`、`q3<`表示状态，其中 `q1`表示初始状态（即状态前带有 `>` 的状态为初始状态），`q3`表示接收状态（即状态后带有 `<` 的状态为接收状态）。在 ENuSMV 中，状态集合要求至少要有一个接收状态，而 ENuSMV1.2 只有在定义 Finite 自动机连接子时才有这个要求，定义 Looping 自动机时并没有这个要求。

自动机的迁移关系以关键字 `TRANSITIONS` 引导，并且需要配合 `case` 语句使用，比如，示例中的语句

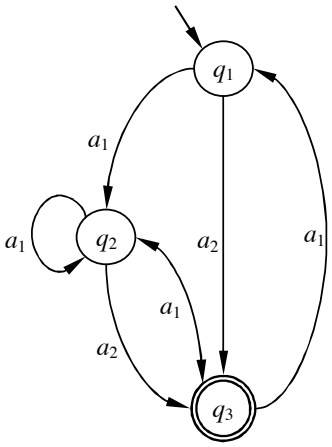
```
TRANSITIONS(q2)
case
    a1 :  {q2, q3};
    a2 :  q3;
esac;
```

就描述了状态 `q2` 的迁移关系：读入字母 `a1` 后，会迁移到状态 `q2` 或者 `q3`，读取字母 `a2` 后，会迁移到 `q3`。

在定义了自动机连接子之后，我们就可以使用自动机连接子定义 ETL_{l+f} 公式了。在 ENuSMV1.2 中，ETL_{l+f} 公式以关键字 `ETLSPEC` 引导，后面跟上 ETL_{l+f} 公式描述。在使用自动机连接子描述 ETL_{l+f} 公式时，除了支持标准的布尔连接子之外，还可以使用 X （即 ETL_{l+f} 公式中的 ○ 连接子）和 U 时序连接子，比如，使用图6.2定义的自动机来声明的公式 6.1 就是一个合法的 ETL_{l+f} 公式。

```
ETLSPEC    A(p, r)
```
(6.1)

除了支持各种连接子，使用自动机定义 ETL_{l+f} 公式还支持自动机公式的嵌套。ENuSMV1.2 与 ENuSMV 不同的是，它还支持用户自定义自动机连接子的初




```
CONNECTIVE  A (a1, a2) : FIN
STATES >q1, q2, q3<;
TRANSITIONS(q1)
case a1 : q2; a2 : q3; esac;
TRANSITIONS(q2)
case
    a1 :  {q2, q3};
    a2 :  q3;
esac;
TRANSITIONS(q3)
case  a1 : {q1, q2}; esac;
```

图 6.2　ENuSMV1.2 中自动机连接子声明示例

始状态，比如，使用图6.2定义的自动机来声明的公式 6.2 也是合法的 ETL$_{l+f}$ 公式。

```
ETLSPEC    A[q2](p, A[q3](p,r))                    (6.2)
```

6.1.1.3　ENuSMV 的验证执行过程

ENuSMV 提供了两种验证执行方式：交互式的验证以及批处理式的验证。

在批处理验证模式下，为了支持本文提出各种算法，我们在标准的 NuSMV 中添加了一些几个命令，下面我们逐一说明。

- 命令 `-cepre` 表示使用 CEPRE 技术；
- 命令 `-bmc_tab` 表示使用基于语义的限界模型检验算法，该命令下支持 LTL 和 ETL$_{l+f}$ 性质验证。
- 命令 `-bmc_tab_inc` 表示使用增量式的基于语义的限界模型检验算法，该命令下同样支持 LTL 和 ETL$_{l+f}$ 性质验证。

在交互式的验证模式下，我们向标准的 NuSMV 的交互式的命令中添加了一些命令，如图 6.3 所示。图 6.3 中蓝色方框内的模块是我们新添加的命令。

- 命令 `go_bmc_tab` 表示进入基于语义的限界模型检验阶段，该过程主要是为 LTL 或者 ETL$_{l+f}$ 的 tableau 构造提供数据结构上支撑，比如编码布尔模型、实例化 SAT 求解器等，事实上，如果待验证的性质是 LTL ，我们默认的将 CEPRE 优化技术也在该阶段实现了。
- 命令 `check_ltlspec_bmc_tab` 表示 ENuSMV 将执行 LTL 的基于语义的限界模型检验验证过程，该命令还包括一个参数，即界值 k，默认情况下 $k = 10$。

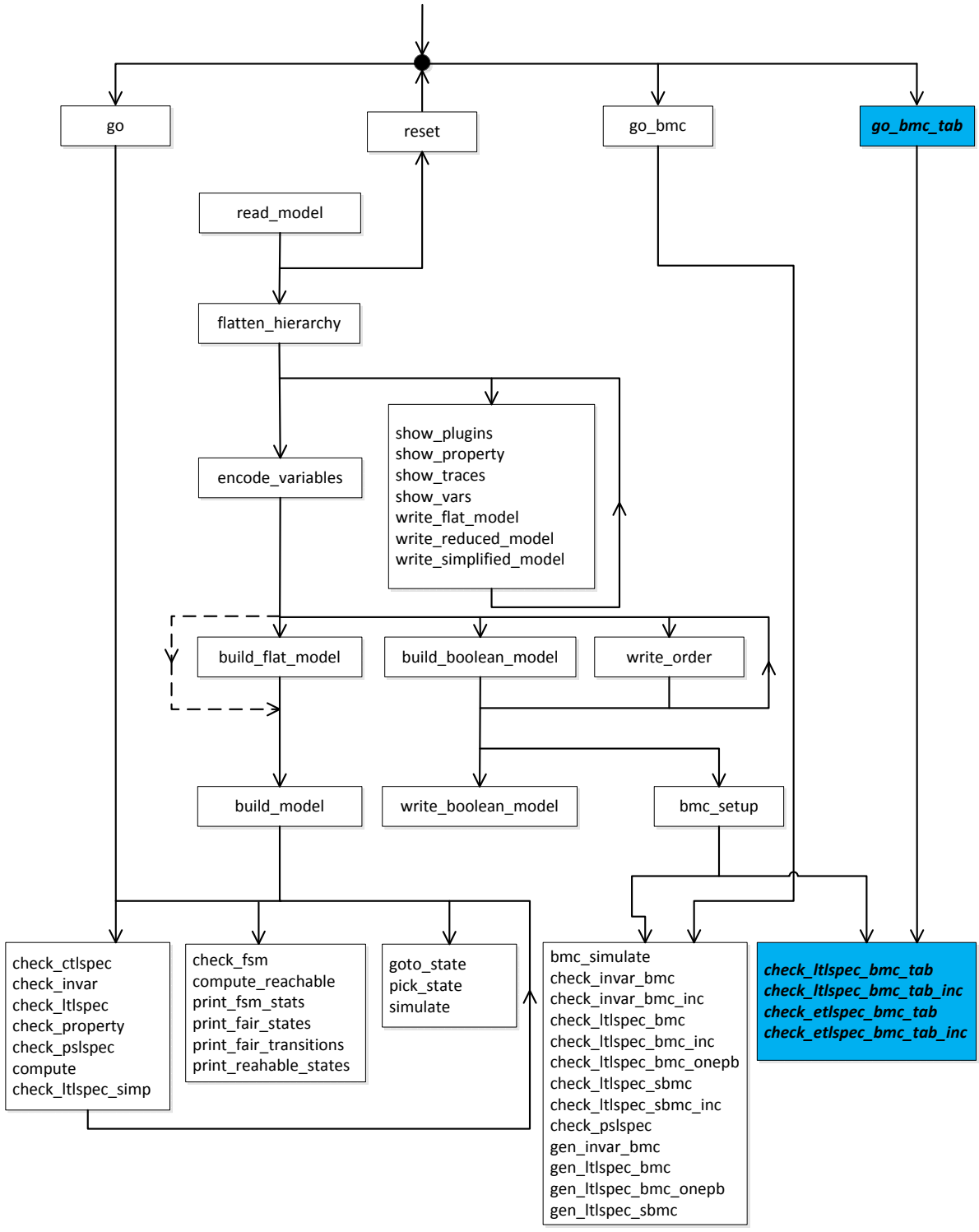


图 6.3　ENuSMV1.2 支持的交互式命令

- 命令 `check_ltlspec_bmc_tab_inc` 表示 ENuSMV 将执行 LTL 的增量式的基于语义的限界模型检验验证过程。同样该命令也包括参数 k，默认情况下 $k = 10$。

- 命令 `check_etlspec_bmc_tab` 表示 ENuSMV 将执行 ETL_{l+f} 的基于语义的限界模型检验验证过程，该命令也包括一个参数 k，默认情况下 $k = 10$。
- 命令 `check_etlspec_bmc_tab_inc` 表示 ENuSMV 将执行 ETL_{l+f} 的增量式的基于语义的限界模型检验验证过程。同样该命令也包括参数 k，默认情况下 $k = 10$。

6.1.2 Reach

本节将给出本文实现的另外一个工具—— Reach 。

6.1.2.1 Reach 的逻辑架构

Reach 工具是使用 C++ 语言开发的一个全新的模型检验工具，它实现了多个基于 SAT 求解器的模型检验算法，其逻辑架构如图 6.4 所示。

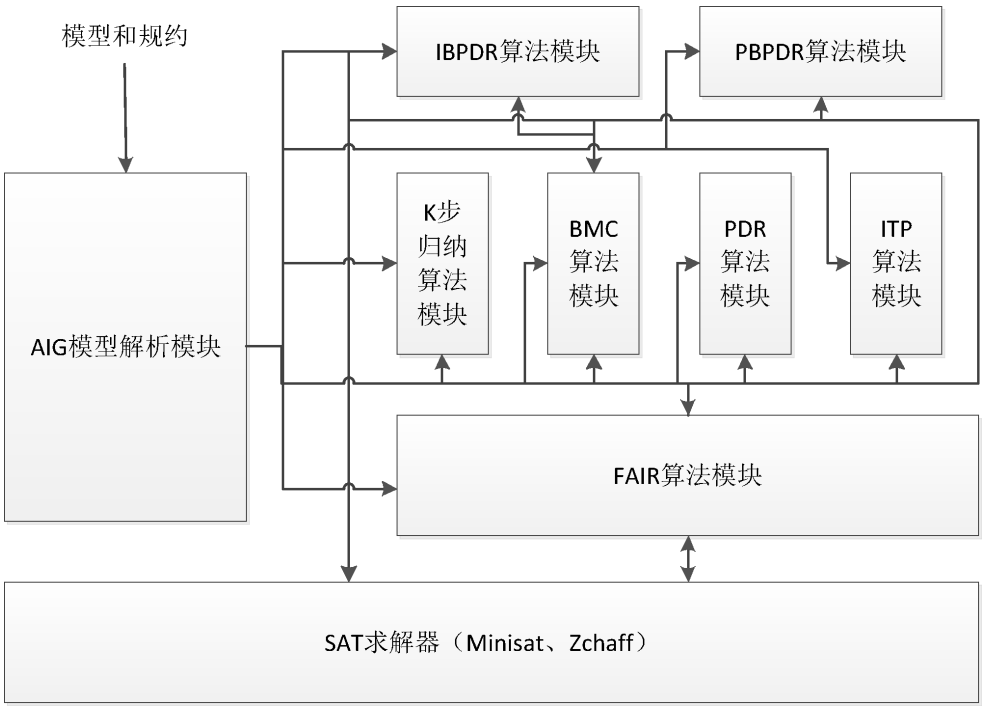


图 6.4 Reach 工具的逻辑架构

下面我们逐一介绍各个模块的主要功能。

- AIG **模型解析模块** 该模块主要负责模型的语法解析，Reach 工具的输入语言为 AIG 格式，该格式是硬件模型检验大赛的主要模式。
- SAT **求解器模块** 该模块主要是提供各种算法的底层引擎，即 SAT 求解器，目前 Reach 工具仅支持 Minisat 求解器，后面将陆续支持更多的求解器。
- FAIR **算法模块** 该模块实现了文献 [154] 提出的 FAIR 算法，该算法将公平性路径查找问题转化为多个可达性问题。

- k **步归纳算法模块** 该模块实现了基本的 k 步归纳算法，本文第 2 章的第 2.2.5 节给出了该算法的伪码描述。
- BMC **算法模块** 该模块实现了安全性质的限界模型检验算法，本文的第 2 章的第 2.2.2 节给出了具体的编码。
- PDR **算法模块** 这个模块主要实现了基本的 PDR 算法，本文的第 2 章的第 2.2.7 节给出了该算法的伪码描述。
- ITP **算法模块** 该模块主要实现了基于插值的符号化模型检验算法，本文的第 2 章的第 2.2.6 节给出了该算法的伪码描述。
- IBPDR **算法模块** 该模块实现了本文第 5 章的第 5.2节提出的交叠的双向 PDR 算法。
- PBPDR **算法模块** 该模块实现了本文第 5 章的第 5.3章提出的并行的双向 PDR 算法。

6.1.2.2 AIG 语法

Reach 的输入格式为 AIG 格式。AIG，即与非图（And Inverter Graph），是一种用来描述电路或者网络逻辑功能的无环图。AIG 由边和节点构成，其中边用来描述逻辑连接关系，节点用来描述逻辑操作或者变量名称。节点有三种：表示变量的叶节点、逻辑与节点、逻辑非节点。任何电路或者网络只要能够采用布尔函数描述，均可以转换成对应的 AIG 格式，如图6.5 所示的逻辑电路，可采用图6.6 所示的 AIG 描述，其中方框表示输入的变量节点，边上的黑点表示逻辑非节点，圆圈则表示逻辑与节点。

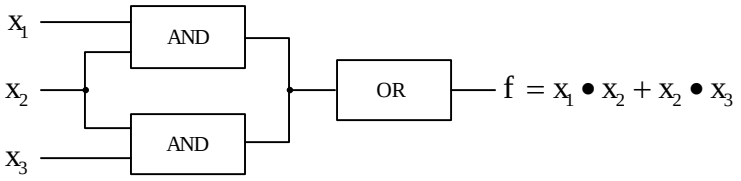


图 6.5 简单电路示例

逻辑门电路可以快速的转换成 AIG 格式，只需要将整个逻辑门电路采用与门和反相器表示，就可直接映射成 AIG 格式，这一优势使得 AIG 可以代替 BDD 或者 CNF 来描述逻辑电路。

AIG 已经被成功的应用于计算机辅助设计，最典型的例子就是 AIG 与布尔可满足理论的结合，为逻辑电路的形式化验证提供了底层的支撑。目前，已经有很多著名的工具都支持 AIG 格式，例如，Berkeley 大学开发的用于逻辑综合和验证的工具 ABC，就采用 AIG 作为描述电路的数据结构；Berkely 大学，Colorado 大学和 Texas 大学联合开发的 VIS 工具也支持 AIG 格式的数据输入，以完成电路的

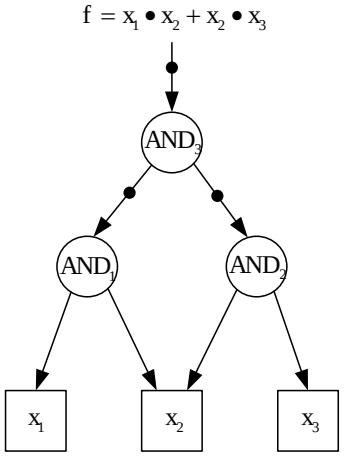


图 6.6　逻辑电路的 AIG 表示示例

验证工作。目前，每年举办的硬件模型检验比赛也采用 AIG 作为标准输入。为了标准化硬件模型检验比赛的数据集，约翰尼斯开普勒大学的 Biere 所领导的验证组专门设计了 AIGER 库，用于描述 AIG 文件的格式。由于本文的部分实验数据采用的是硬件模型检验比赛的标准数据集，数据读入也是基于 AIGER 库，因此，在这里，对 AIGER 库进行简单的介绍。

AIGER 包括 ASCII 码和二进制两种版本，相互之间很容易转换，而且该工具提供了相应的转换函数，这里只介绍 ASCII 码版本。一个 ASCII 码格式的 AIGER 文件描述 AIG 时，最开始有一个标识符 aag，然后包括 5 个非负整数 `M`，`I`，`L`，`O`，`A`依次用空格隔开，其中，

- `M`：表示最大的变量索引号（一般情况下 `M=(I+L+A)`）；
- `I`：表示输入变量的个数；
- `L`：表示锁存器的个数；
- `O`：表示输出的个数
- `A`：表示逻辑与门的个数

文件的第一行为标识符和 5 个整数，下面内容依次为输入索引号、锁存器索输入及输出索引号、输出索引号、与门输入及输出索引号、输入输出变量名和用户注释。其中输入输出变量名和用户注释可以没有，`I`，`L`，`O`，`A`的某个域个数为 0，则表示没有对应的描述。比如，没有任何输入输出的空电路就可以描述如下，它只有头描述，没有具体内容。

```
aag 0 0 0 0 0
```

逻辑 $FALSE$ 的 AIG 表示如下，即只有一个输出，并且输出为 0。

```
aag 0 0 0 1 0
0
```

同样，逻辑 $TRUE$ 的 AIG 描述则如下所示，即只有一个输出，并且输出为 1。

```
aag 0 0 0 1 0
1
```

为了便于理解，这里给出图6.6所表示的 AIG 的文件描述，如下所示。

```
aag 6 3 0 1 3       // header
2                   // input x1
4                   // input x2
6                   // input x3
13                  // output f
8 2 4               // AND gate 1
10 4 6              // AND gate 2
12 9 11             // AND gate 3
```

在 AIG 格式中，每一个变量都是用一个非负整数来表示的，而变量是根据索引号乘以 2 来编号的。如图6.6电路对应的 AIG 描述，由于最大变量索引号为 6，因此，它包含的变量分别为 2、4、6、8、10、12。根据电路描述可知，有三个输入变量，分别对应 AIG 格式的 2-4 行，分别用 2、4、6 来表示，可以理解为它们分别对应电路中的三个输入变量 x_1, x_2, x_3。在这里，偶数表示正文字，其对应的负文字只需将变量加 1 即可。例如，3 就表示 2 的非，即 3 就表示 $\neg x_1$。根据如6.6可知，其还包含两个与门和一个或门电路。由于 AIG 格式只支持与门电路描述，因此，电路中的所有或门都是通过与门取反得到的。在 AIG 描述中，8 表示 2 和 4 的逻辑与结果，10 表示 4 和 6 的逻辑与结果，而 12 表示 9 和 11 的逻辑与结果。根据前面的解释可知，9 是 8 的取非结果，11 是 10 的取非结果，13 是 12 的取非结果。因此 13 其实就表示了逻辑公式 $\neg(\neg(x_1 \wedge x_2) \wedge \neg(x_2 \wedge x_3))$。

下面我们再给出一个带锁存器的例子。如图所示的带锁存器的电路，输入变量有两个分别为’enbale‘和’reset‘，输出变量也有两个，分别为’Q‘和’Q- ‘。整个电路由一个二输入的异或门和一个 D 触发器构成。当’reset‘为 0 时，Q在下一个周期的输出重置为 0，当’reset‘为 1 时，’Q‘在下一个周期的输出为输入端 D 的值。D 触发器数据输入端 D 的值是输入信号’enable‘和上一个周期 D 触发器输出端 Q的结果的异或。即，当’enable‘为 1 时，D 触发器的输入取反，’enable‘为 0 时，D 触发器的输入不变。该电路的功能是一个带复位信号的一位计数器，即’reset‘为复位信号，’enable‘为 0 时，计数器保持，为 1 时，计数器加 1。

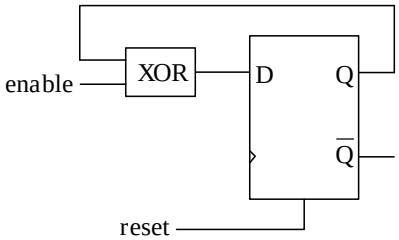


图 6.7　带锁存器的逻辑电路

电路对应的 AIG 文件格式如下所示。其中 2 表示输入变量’enbale‘，4 表示输入变量’reset‘，6 表示一个锁存器的输出，并且 6 的输出为 8 的值。电路的输出为 6 和 7，即锁存器的输出和它的取反。同时电路包含 4 个与门，其中与门 14 的输入为 3 和 7，表示逻辑 `!enable & !Q`；与门 12 的输入为 2 和 6，表示逻辑 `enable & Q`；与门 10 的输入是 13 和 15，即 12 和 14 分别取反的值，表示逻辑 $\texttt{enable} \oplus Q$；与门 8 的输入为 4 和 10，表示’reset‘信号和异或结果的与。锁存器的引入保存了前一个周期的结果，实现了数据保存的功能。

```
aag 7 2 1 2 4     // header
2                 // input 0 'enable'
4                 // input 1 'reset'
6 8               // latch 0 Q next(Q)
6                 // output 0 Q
7                 // output 1 !Q
8 4 10            // AND gate 0
10 13 15          // AND gate 1
12 2 6            // AND gate 2 enable & Q
14 3 7            // AND gate 3 !enable & !Q
```

6.1.2.3　Reach 的执行过程

目前，Reach 工具还没有图形化界面，其只能在命令行下执行，其支持的主要命令如下：

- `-h` 打印帮助信息；
- `-v` $verbose-level$ 输出 Reach 的内部执行信息，$verbose-level$ 的取值是 1-3 的自然数，数值越高，输出的信息越多；
- `-bmc` k 执行 BMC 算法，其包含一个参数 k，即界值，默认的 $k=10$；
- `-pdr` 执行基本的 PDR 算法；
- `-k_induction` 执行基本的 k 步归纳算法；
- `-itp` 执行基本的基于插值的模型检验算法；
- `-ibpdr` k l 执行 IBPDR 算法，该命令包含两个参数，这两个参数是两个整数，主要是方便用户指定交叠策略，如果没有设置这两个参数，Reach 会

选择随机交叠策略来执行 IBPDR 算法，否则，则按照用户自定义的交叠策略来执行 IBPDR 算法；

- `-pbpdr` 执行 PBPDR 算法；
- `-fair` *method* 执行 FAIR 算法，将公平性路径查找问题转化为可达性问题，其后面的参数 *method* 表示后面将要使用的可达性算法，可以选择上面的任意一个可达性算法，如命令 `-fair pdr` 表示将公平性查找问题转化为可达性问题之后，使用基本的 PDR 算法来验证可达性问题。

6.2 工具使用

本节将结合具体的示例介绍工具 ENuSMV1.2 和 Reach 的使用。

我们使用本文第 4 章实验用的 DME 电路问题和“二进制累加器”电路问题作为示例，介绍如何使用两个工具验证 LTL 以及 ETL_{l+f} 性质。

DME 电路的 SMV 代码描述在附录 A.1 中，关于 DME 电路问题，我们主要关心两个性质：

- **安全性：** 在任何时刻，不可能有两个仲裁器同时被响应；
- **响应性：** 任何一个仲裁器发出的请求将来必被响应。

在本文的示例中，我们设置了 3 个仲裁器，上面两个性质对应的 LTL 公式描述如下：

```
-- Safety
LTLSPEC G ((!(e-1.u.ack & e-2.u.ack) & !(e-1.u.ack & e-
3.u.ack)) & !(e-2.u.ack & e-3.u.ack))
-- Liveness
LTLSPEC G((e-1.req -> F e-1.u.ack) & (e-2.req -> F e-2.u.ack) & (e-
3.req -> F e-3.u.ack))
```

使用工具 ENuSMV1.2 ，我们可以对这两个性质做限界模型检验。在批处理模式下，使用如下命令就可以对上述 LTL 性质进行基于语义的限界模型检验了：

```
./NuSMV -bmc_tab -bmc_length 100 dme.smv
```

若需要使用 CEPRE 技术，则只需加上 `-cepre` 命令即可，如下所示

```
./NuSMV -cepre -bmc_tab -bmc_length 100 dme.smv
```

同样，如果需要使用增量式的基于语义的限界模型检验，则可以使用下面的命令来运行 ENuSMV1.2 ，

```
./NuSMV -bmc_tab_inc -bmc_length 100 dme.smv
```

使用 ENuSMV1.2 只能进行限界模型检验，如果要进行完全的基于 SAT 的符号化模型检验，我们需要使用 Reach 工具。而由于 Reach 工具不能直接支持 SMV格式，因此，我们需要用两个工具将 SMV 代码转换成 AIG 格式。首先，使用工具 SMV2AIG 将 SMV 格式的描述转化为 AIG 格式，SMV2AIG 工具，允许用户指定使用外部工具处理 SMV 文件中的 LTL 公式或者 ETL_{l+f} 公式，因此，我们仅需使用下面的命令就可以将 SMV 格式的文件转化成 AIG 格式，

```
./smv2aig -L ltl2smv dme.smv -o dme.aig
```

这里，`-L ltl2smv` 的意思是使用外部可执行文件将 SMV格式里的 LTL 公式的非转化成 tableau，并得到模型与该 tableau 的乘积，然后输出到`.aig`文件，因此，这里得到的 AIG 文件描述的就是一个公平路径查找问题，可以直接使用 Reach 工具对其进行验证。

在附录 A.2 中，我们给出了二进制累加器的 SMV描述。在二进制累加器中，我们主要关心一种非 satr-free 的性质，即在偶数时刻，`bit_0`都会产生进位。使用 ETL_{l+f} 公式可以描述如下。

```
CONNECTIVE C_2(a,b) : LOOP
STATES
      >st_1, st_2 ;
TRANSITIONS(st_1)
     case
     a: st_2;
     esac;
TRANSITIONS(st_2)
     case
     b: st_1;
     esac;

ETLSPEC C_2(TRUE, (bit_0.value=1))
```

使用之前同样的命令，可以利用 ENuSMV1.2 来对该公式进行限界模型检验，如果需要对其进行完全性验证，必须利用 SMV2AIG 工具将 SMV 描述转化成 AIG 格式，这里与之前的命令稍有不同，其命令如下。

```
./smv2aig -L etl2smv counter.smv -o counter.aig
```

这里，`-L etl2smv` 使用的是本文提出的 ETL_{l+f} 的 tableau 构造算法，它是一个单独的可执行文件，可构造 ETL_{l+f} 公式的 tableau 并输出模型与该 tableau 的乘积。

6.3 本章总结

本章给出了本课题实现的两个实用的模型检验工具的具体实现，并通过两个简单的例子演示了其如何使用。

ENuSMV1.2 是通过扩展 NuSMV 实现的，它支持本文提出的 CEPRE 算法、增量式的基于语义的限界模型检验、 ETL_{l+f} 的 tableau 构造算法、基于 BDD 的 ETL_{l+f} 的符号化模型检验算法以及基于语义的 ETL_{l+f} 的限界模型检验算法。到目前为止，ENuSMV1.2 工具已经发布在互联网上，网址为 http://sourceforge.net/projects/enusmv12/。到现在为止，该工具已经吸引了来自美国、俄罗斯、法国、印度等 10 个国家的学者和工程师访问和下载 40 余次，如其网页截图 6.8 所示。

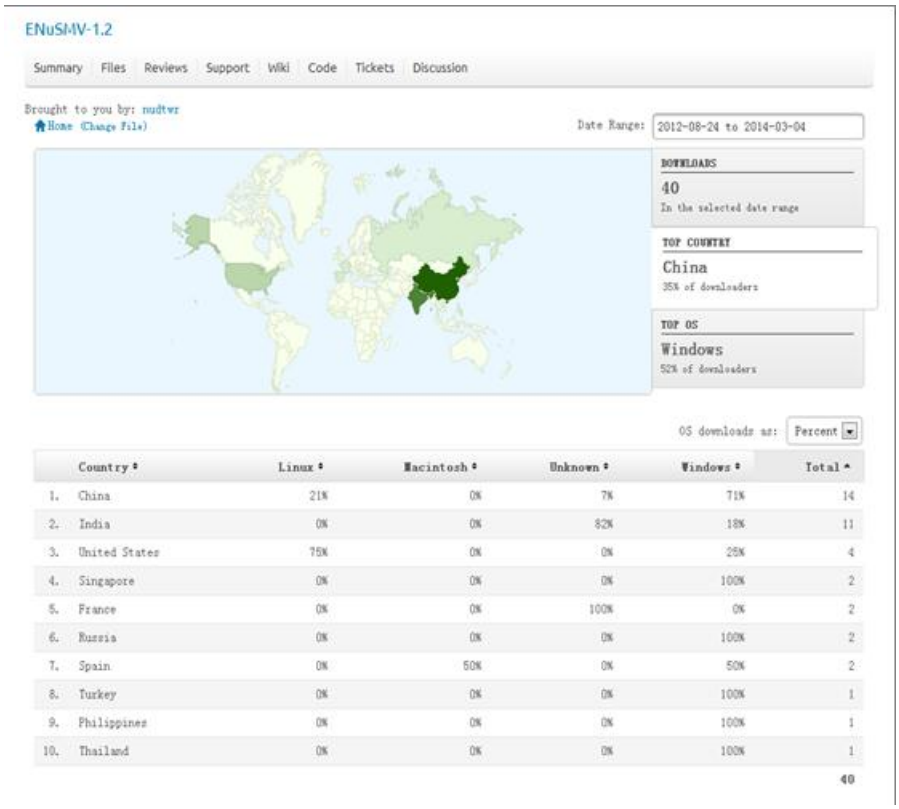


图 6.8 ENuSMV1.2 网页截图

Reach 工具是一个以 AIG 格式为输入的可达性分析工具，它主要支持各种可达性分析算法，通过将 LTL 、 ETL_{l+f} 的模型检验问题转化成公平路径查找问题，Reach 工具可以验证模型是否满足给定的 LTL 或者 ETL_{l+f} 公式。Reach 工具目前还在进一步优化中，预备参加 HWMCC2014。

第七章　总结与展望

本章对论文的内容进行总结，并展望了将来的工作。

7.1　本文总结

本文围绕基于 SAT 的符号化模型检验技术开展了一系列研究，旨在推动基于 SAT 的符号化模型检验技术在软硬件验证领域的应用。

本文的工作主要包括如下四个方面:

1. **优化了现有的 LTL 限界模型检验技术**。

   本文的第 3 章提出了一种优化的 LTL 限界模型检验技术。具体的讲，该技术包含两部分，第一部分，我们称之为 CEPRE 技术，它是一种通用的优化技术，它的主要思想是在模型检验算法工作之前利用一种轻量级的方式先对性质规约做等价的化简，使得化简后得到的公式与原公式的反例集合保持一致，然后利用化简后的公式代替原公式做模型检验。 CEPRE 技术的核心是使用两种类型的规则对公式不停的进行化简直到公式不能再化简为止，这两类规则分别为**模型无关规则**和 **模型相关规则**。如上所述， CEPRE 技术是一种通用的技术，其后端的模型检验算法可以是任何针对 LTL 的模型检验算法，但是**模型相关规则**只适用于符号化的模型检验算法，因为其需要利用模型的布尔表示来优化待验证的规约。该技术的第二部分是针对 LTL 限界模型检验，利用 SAT 求解器的增量式求解能力来优化其编码，通过引入新的变量，对基于语义的 LTL 限界模型检验编码做了增量式编码，实验表明，新的编码相对于已有的编码在验证效率上有很大的提高。整体上讲，这两种技术是在模型检验过程的不同阶段完成的，其并没有任何冲突，它们可以相互支持，从而提高 LTL 的限界模型检验效率。本文针对这两种技术做了大量的实验，实验结果表明，结合这两种技术的限界模型检验算法的验证效率相对于原有的算法有了很大的提高。

2. **提出了 ω-正规性质限界模型检验算法**。

   随着 SAT 求解器的不断发展，限界模型检验技术已经成功的应用在 LTL 的验证中，其在查找系统错误时，相对于基于 BDD 的符号化模型检验技术有着很大的优势，而 LTL 最大的瓶颈在于它的表达能力，它不能刻画很多重要的 ω-正规性质。因此，本文的第 4章针对两种具有 ω-正规性质表达能力的时序逻辑，提出了三种限界模型检验算法，该研究内容的目的是扩展限界模型检验能力，使其支持完全的 ω-正规性质的验证。

针对 ETL_{l+f} ，本文提出了基于语义的限界模型检验技术。该技术的核心是构造 ETL_{l+f} 的 tableau，本文通过扩展 LTL 的 tableau 构造方法给出了 ETL_{l+f} 的 tableau 构造方法，然后将 ETL_{l+f} 的限界模型检验问题转化为受限路径上的公平性路径查找问题，最后利用 SAT 求解器来验证该问题。针对另外一种时序逻辑 QTL ，本文提出了两种限界模型检验技术，第一种是根据 QTL 的语法结构，将 QTL 公式归纳的编码成带量词的布尔公式，然后利用 QBF 求解器来验证 QTL 的限界模型检验问题，称这种技术为基于语法的 QTL 限界模型检验，该技术理论上是可行的，但是受限于 QBF 的处理问题的能力，该技术在实际中处理问题的规模非常有限。第二种针对 QTL 限界模型检验技术同样是通过扩展 LTL 的 tableau 构造方法来构造 QTL 的 tableau ，然后利用基于语义的编码技术验证 QTL 的限界模型检验问题，该技术相对于基于语法的编码有很大的优势，因此，它在实际中更加实用。本文还给出了 ETL_{l+f} 限界模型检验实验结果。实验结果显示，该方法能将正确高效的验证 ETL_{l+f} 公式，尤其在查找错误时，该技术相对于基于 BDD 的符号化模型检验技术有很大的优势。

3. **基于 SAT 符号化模型检验的完全性保证技术**。

   制约限界模型检验技术的主要瓶颈在于其不是一种完全的技术，即它不能证明给定的模型是否满足给定的规约。本文的第 5 章提出了一种保证限界模型检验完全性的技术，称为基于 SAT 的完全性保证技术。该技术的核心是利用 SAT 求解器高效的求解可达性问题。由于 LTL 、 ETL_{l+f} 以及 QTL 的模型检验问题都可以转化为公平路径查找问题，而公平路径查找问题可以转化为若干个可达性问题，因此基于 SAT 完全性保证技术的核心在于高效的求解可达性问题。

   本文基于目前最好的可达性算法，即 PDR 算法，提出了两种新的可达性算法： IBPDR 和 PBPDR 。这两种算法的核心都是通过求解双向逼近的可达性空间从而加快算法的收敛速度。不同的是， IBPDR 是一种交叠的执行策略， PBPDR 是一种并行的执行策略。由于是求解双向逼近的可达性状态空间，因此，在算法的执行过程中，我们会得到比 PDR 算法更多的关于状态空间的信息，利用这些信息，提出了几种优化策略。

   本文给出了 IBPDR 和 PBPDR 的实验结果，实验结果显示，本文提出的算法提高了基本 PDR 算法的效率。

4. **实现了两个实用的工具：** ENuSMV1.2 **和** Reach

模型检验技术的最终目的是应用，将算法转换为工具是模型检验很重要的组成部分。本文另外一个贡献就是实现了两个实用的工具： ENuSMV1.2 和 Reach 。

ENuSMV1.2 是基于 NuSMV 实现的，在该工具中，我们主要实现了四个算法，分别为 CEPRE 算法、增量式的基于语义的 LTL 限界模型检验算法、基于 BDD 的 ETL_{l+f} 符号化模型检验算法、 ETL_{l+f} 基于语义的限界模型检验算法。该工具是开源的并且已经公布在互联网上。

Reach 工具是用 C 语言开发的全新的基于 SAT 的符号化模型检验工具，该工具的输入格式为目前硬件模型检验大赛的标准格式： AIG 格式。该工具不仅实现了本文提出的 IBPDR 和 PBPDR 算法，它还实现了多种其他的可达性算法，包括安全性质的限界模型检验算法、基于 SAT 的 k 步归纳算法、基于插值的符号化模型检验算法以及基本的 PDR 算法。该工具目前还在继续开发中，预备参加 2014 年的硬件模型检验大赛。

7.2 将来的工作

模型检验技术经过 30 多年的发展，已经成为目前计算机科学领域非常重要的技术，其已经被证明在计算机辅助验证领域具有非常重要的作用。基于 SAT 的符号化模型检验技术是模型检验领域研究的热点问题，本文进一步的研究工作包括以下几个方面:

1. **扩展 CEPRE 技术到其他时序逻辑**。

   本文提出的 CEPRE 技术是符号化模型检验中的一种通用的优化技术，它的思想就是在进行复杂的模型检验之前先用一种轻量级的技术对待验证的规约进行化简。目前， CEPRE 技术仅针对 LTL 公式的验证做了优化，后面的工作拟进一步探索 CEPRE 技术在其他时序逻辑中的应用，比如 PSL 、QTL 等。

2. **ω-正规性质限界模型检验将来的工作**。

   本文针对两种具有 ω-正规表达能力的时序逻辑 ETL_{l+f} 和 QTL ，分别提出了限界模型检验算法。ETL_r 是使用非确定的 Repeating 自动机（即 Büchi 自动机）作为时序连接子的扩展时序逻辑，该逻辑的表达能力同样等价于 ω-正规语言，因此，探索 ETL_r 的限界模型检验将是将来工作的一部分。

   PSL 作为工业标准语言，它的模型检验问题一直都是工业界关注的重点。目前，基于 BDD 的 PSL 的符号化模型检验已经比较成熟，而基于 SAT 的 PSL 符号化模型检验还处于研究阶段，比如，最新的 NuSMV 工具也仅支持

受限的 PSL 的限界模型检验，因此，进一步研究 PSL 的基于 SAT 的符号化模型检验也将是本文将来的工作。

3. **基于 SAT 的可达性分析技术将来的工作**。

   目前，最好的两个基于 SAT 的可达性技术分别是基于插值的符号化模型检验技术和 PDR 技术，本文基于 PDR 技术，提出了两种新的可达性算法，而探索结合基于插值的符号化模型检验技术和 PDR 技术将是未来的工作。不仅这两种技术的结合是将来的工作，探索各种可达性算法之间的交互也会是本文未来工作的重点。

4. **工具实现将来的工作**。

   在工具实现上，本文将来的工作主要是将本文提出的算法全部实现。首先，基于 ENuSMV1.2 实现基于 QBF 求解器的 QTL 限界模型检验、实现 QTL 的 tableau 构造算法以及 QTL 的基于语义的限界模型检验算法。在 Reach 工具上进一步优化 PBPDR 算法以及混合的基于插值和 PDR 的可达性算法。

致　谢

衷心感谢我的导师毛晓光教授！从 2007 年 1 月到 2014 年 7 月，不是每一个人都有这样好的运气拥有一份长达七年半的师生情谊的，而我就是其中一个幸运儿。毛老师不仅是我博士阶段的导师，也是我硕士生导师和本科毕业设计的导师。在国防科大十一年的求学生涯中，毛老师是影响我最深的人之一。毛老师学术功底深厚、治学严谨、视野开阔、思维敏捷。从我进入毛老师门下，毛老师就给我充分的自由度去研究自己感兴趣的课题，无条件的支持我做任何方面的学术研究！而每当我在课题中遇到困难时，毛老师又会鼓励我，让我充满信心的继续前行。毛老师不仅在学术研究上影响了我，在生活和工作上，毛老师也给予了我极大的帮助。生活中，毛老师是一个非常有亲和力的人。七年半以来，我有任何困惑和疑虑，都会找毛老师倾诉，而毛老师总是能给我适当的建议，让我乐观的面对一切问题，其积极乐观的生活态度是我今后人生中学习的榜样。同时，感谢我的师母李老师一直以来对我的关心和帮助！衷心祝愿毛老师一家好人一生平安！

衷心感谢王戟老师！王老师理论扎实、视野开阔，对待研究问题总是有独到而深刻的理解！跟王老师为数不多的几次深入交流中，我学到了很多做研究的方法和做研究的态度。攻读博士这几年没有向王老师多请教多学习可能是我读博期间的遗憾之一！衷心感谢 602 教研室的董威老师、文艳军老师、李曦老师、谭庆平老师、毛新军老师，感谢他们在我读博期间的关心和鼓励！衷心感谢计算机系的翟英老师和薛源老师，感谢他们在我读博期间的关心和帮助！

衷心感谢刘万伟师兄！没有刘师兄的帮助，我不可能正常的完成博士课题。永远不会忘记刘师兄在 PDL 301 手把手教我使用 LaTeX 的场景；永远忘不了刘师兄教我时序逻辑和模型检验的场景。博士期间能够在刘师兄的帮助下做课题是我的福气！研究中的刘师兄是一个严谨、细致、思维敏捷的人。对于研究的对象，刘师兄总是能快速的抓住问题的本质并能提出建设性的意见，这一点令我由衷的敬佩！在博士期间，跟刘师兄的每次讨论，我都能收获很多！刘师兄深厚的理论功底，我四年中仅仅学到了一些皮毛，说来真有点惭愧！希望后面的师弟们能吸取我的教训，跟刘师兄学更多的知识和学问。我还要感谢郭欣嫂子，谢谢嫂子容忍刘师兄在我身上投入那么多的时间！衷心祝愿嫂子永远年轻漂亮，祝愿他们的儿子健健康康的成长！

衷心感谢陈振邦师兄和张晓艳师姐！在科大的十一年中，虽说认识师姐的时间还不到 5 年，但是师姐对我的照顾实在太多太多了！是他们，让我在长沙、在

国防科大第一次有了家的感觉，像我的哥哥姐姐一样，我心底里也是把晓艳姐当成了一个姐姐！在科大最后的这几年中，能有幸遇见师姐和师兄，我把这理解成是一种缘分，再次感谢师兄和师姐！祝师兄师姐在以后的人生中幸福美满，祝他们的女儿小诺快乐的成长！感谢陈立前师兄和魏登萍师姐！阿前师兄是一位优秀的研究人员，在为数不多的跟师兄的交流中，我学到了不少做好研究的方法！祝他们永远幸福，也祝他们的儿子快快乐乐的成长。

衷心感谢中科院软件所的张立军教授！感谢张老师在我读博的最后一段时间对我的指导和帮助。在我博士论文撰写期间，张老师给我提供了无与伦比的科研环境！没有张老师给我提供这么好的环境，很难想象我的博士论文能否按时完成！工作上，张老师是一位优秀的研究人员；生活中，他是一个随和、开朗、积极乐观的人！跟张老师的接触中，我学到的不仅仅是做研究的方式和方法，还包括对待生活、对待工作的人生态度。再次由衷的感谢张老师！衷心祝愿张老师和他的家人一生美满幸福！感谢中科院软件所的吴志林老师！跟吴老师讨论 QTL 的限界模型检验问题是愉快而充实的时光。吴老师理论功底非常扎实，把极为抽象的逻辑描述的直观而清晰！在此对吴老师表示衷心的感谢！感谢中科院的张健老师和马菲菲老师在 SAT 求解问题上提供的帮助！感谢中科院的李勇师弟在我实验和毕业论文撰写期间对我的支持和帮助！同时感谢付辰师弟对我在中科院软件所工作学习期间的帮助！

感谢王昭飞、李仁见、万晓敏、吴海亮、陈杰、樊沛、董龙明、张羽丰等师兄，跟各位师兄共同奋斗的日子，让我终身受益！感谢胡翠云师姐在我读博期间的关心和帮助！感谢代子营、齐玉华、傅先进、沈思齐，感谢他们对我学习和生活上的帮助！感谢雷宴、王承松、刘焕、王殿林、韩仲志、周竟文、李唯实、吴学光、毛建辉、姜加红、田楠、潘瑾琨、张卓、张鹏、兰韵、于恒彪、刘江潮、路遥、尹邦虎等师弟对我的帮助和支持！感谢张龙师弟对我博士论文的检查和修改！

感谢同学吴强、贾建斌、倪时策、毛华坚、鲁晓佩，跟他们一起吃饭，一起吹牛的日子是我博士期间最惬意的时光！感谢李荣春、周旭、方旭东、王晓斌、谭霜、柴俊、梅松竹、刘权、何力、马行空、朱玄、王涛、邓科峰、孙永林、于康、王春光、陈志广等各位同学和朋友在我读博期间的支持和帮助！感谢王新国、朱涛、林红锦、杨红运、王艳丰、欧阳登轶、孙友佳等学员队领导的关怀和帮助！感谢教务赵欣、潘晓辉参谋，感谢他们不辞劳苦，在我论文评阅、申请答辩等各种繁杂的手续和事务上提供的帮助！

感谢国防科大第一干休所的姜叔叔和张阿姨一家，感谢他们一直以来对我们的关心和照顾！在干休所暂住的这几年，品尝了很多阿姨亲手下厨的美味佳肴！在此对他们表示衷心的感谢！衷心的祝愿他们一家人一生平安，幸福美满！同时还要感谢所里的小皮师傅的关心和照顾！

感谢 Corolado 大学的 Aaron Bradley 教授和他的博士生 Zyad Hassan！他们都是非常优秀的研究人员。感谢他们热心的回答我关于 IC3 技术的每一个问题！

感谢王留东、周菲夫妇！感谢他们在我人生道路上的一直陪伴！衷心的祝福他们相亲相爱共度一生！

感谢我的女友布凡！在科大求学的这么多年，一直是她在背后支持我、鼓励我！没有她默默的付出，我不可能顺利的拿到这个博士学位！读博期间，我们基本没有逛过街、没有一起看过电影，甚至连我一再承诺的带她去凤凰旅游都没有实现过！我总是在“忙、忙、忙”，忽略了她太多太多！而她最想去的海边，我一直说搞好一个“课题点”就带她去，结果一个接着一个..... 到现在还没有实现！很难想象她一直守在我这样一个人身边过了这么多年！在此，我要向她说声：谢谢你为我付出的一切，我希望用后面的人生来回报你！

感谢家人对我的支持和付出！感谢我的父母含辛茹苦的把我养大，默默无闻的支持我这么长时间的读书生涯！感谢我的两个姐姐，从小她们就对我疼爱有加；长大后，她们又无条件的支持我的任何决定！家人永远是我前进的动力源泉！

最后，再次感谢我生命中最重要的一个人 ——我的妈妈！任何言语都无法表达她在我心中的地位，没有让她看到我博士毕业是我终生的遗憾！谨以此文献给我天堂里的妈妈！

参考文献

[1] Biere A, Cimatti A, Clarke E M, et al. Symbolic Model Checking without BDDs [C]. In Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). 1999: 193–207.

[2] 杨晋吉．基于 SAT 的有界模型检验及其应用研究 [D]．广东中山：中山大学，2008．

[3] 屈婉霞．多处理机系统 Cache 一致性协议模型检验关键技术研究 [D]．湖南长沙：国防科学技术大学，2008．

[4] Cousot P, Cousot R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints [C]. In Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. 1977: 238–252.

[5] Clarke E M. The birth of model checking [M] // Clarke E M. 25 Years of Model Checking. Springer, 2008: 2008: 1–26.

[6] 沈胜宇．模型检验的反例解释 [D]．湖南长沙：国防科学技术大学，2005．

[7] McMillan K. Cadence smv [J]. Cadence Berkeley Labs, CA. 2000.

[8] Beer I, Ben-David S, Eisner C, et al. RuleBase: an Industry-Oriented Formal Verification Tool [C]. In 33rd Design Automation Conference. 1996: 655–660.

[9] Copty F, Fix L, Fraer R, et al. Benefits of bounded model checking at an industrial setting [C]. In Computer Aided Verification. 2001: 436–453.

[10] Clarke E, Kroening D, Lerda F. A Tool for Checking ANSI-C Programs [C]. In Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04). 2004: 168–176.

[11] Chaki S, Clarke E M, Ouaknine J, et al. State/event-based software model checking [C]. In Integrated Formal Methods. 2004: 128–147.

[12] Ball T, Rajamani S K. The SLAM toolkit [C]. In Computer aided verification. 2001: 260–264.

[13] Godefroid P. VeriSoft: A tool for the automatic analysis of concurrent reactive software [C]. In Computer Aided Verification. 1997: 476–479.

[14] Beyer D, Henzinger T A, Jhala R, et al. The software model checker Blast [J]. International Journal on Software Tools for Technology Transfer. 2007, 9 (5-6): 505–525.

[15] Dill D. Murphi model checker. http://www.cs.utah.edu/formal_verification/Murphi/.

[16] Musuvathi M, Park D Y, Chou A, et al. CMC: A pragmatic approach to model checking real code [J]. ACM SIGOPS Operating Systems Review. 2002, 36 (SI): 75–88.

[17] NASA. JPF. http://babelfish.arc.nasa.gov/trac/jpf. 2013.

[18] McMillan K L. The SMV system [M] // McMillan K L. Symbolic Model Checking. Springer, 1993: 1993: 61–85.

[19] Cimatti A, Clarke E, Giunchiglia1 F, et al. NuSMV: A New Symbolic Model Verifier [C]. In CAV'99. 1999: 495–499.

[20] Holzmann G J. The SPIN model checker. http://spinroot.com/spin/whatispin.html.

[21] Hassan Z. Incremental Inductive Model Checker. http://ecee.colorado.edu/wpmu/iimc/.

[22] Pnueli A. The Temporal Logic of Programs [C]. In Proc. of 18th IEEE Symposium on Foundation of Computer Science (FOCS' 77). 1977: 46–57.

[23] Wolper P. Temporal Logic Can Be More Expressive [C]. In Proceedings of the 22nd IEEE Symposium on Foundations of Computer Science. Nashville, October 1981: 340–348.

[24] Accellera. Accellera Property Languages Reference Manual. http://www.eda.org/vfv/docs/PSL-v1.1.pdf. June 2004.

[25] Sistla A. Theoretical Issues in The Design and Verification of Distributed Systems [D]. [S. l.]: CMU Dept. of Computer Science, 1983.

[26] Banieqbal B, Barringer H. Temporal logic with fixed points [C]. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, Temporal Logic in Specification. 1987: 62–74.

[27] Emerson E A, Clarke E M. Characterizing Correctness Properties of Parallel Programs Using Fixpoints [C]. In Proc. of the 7th Int. Colloquium on Automata, Languages and Programming (ICALP'80). 1980: 169–181.

[28] Clarke E, Draghicescu I. Expressibility Results for Linear-time and Branching Time Logics [C] // de Bakker J, de Roever W, Rozenberg G. In Proc.Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency. 1988: 428–437.

[29] Kozen D. Results on the Propositional μ-calculus [J]. Theoretical Computer Science. 1983, 27: 333–354.

[30] Clarke E M, Emerson E A. Design and synthesis of synchronization skeletons using branching time temporal logic [M]. Springer, 1982.

[31] Wolper P, Vardi M Y, Sistla A P. Reasoning about Infinite Computation Paths [C]. In Proc. 24th IEEE Symposium on Foundations of Computer Science. Tucson, 1983: 185–194.

[32] Bryant R E. Graph-based algorithms for Boolean function manipulation [J]. IEEE Transactions on Computers. 1986, C-35 (8): 677–691.

[33] McMillan K L. Symbolic Model Checking, An Approach to the State Explosion Problem [D]. Kluwer Academic Publishers: Carnegie Mellon University, 1993.

[34] Clarke E, Grumberg O, Hamaguchi K. Another Look at LTL Model Checking [C]. In CAV'94. 1994: 415–427.

[35] Clarke E M, Grumberg O, Peled D A. Model Checking [M]. Longdon, England: The MIT Press, 1999.

[36] 苏开乐 , 吕关锋. CTL* 的符号化模型检测 [J]. 计算机学报. 2005, 28 (11): 1798–1806.

[37] Liu W, Wang J, Wang Z. Symbolic Model Checking of ETL (with English abstract) [J]. Journal of Software. 2009, 20 (8): 2015–2025.

[38] Liu W, Wang J, Chen H, et al. Symbolic model checking apsl [C]. In Theoretical Aspects of Software Engineering, 2008. TASE'08. 2nd IFIP/IEEE International Symposium on. 2008: 39–46.

[39] Biere A, Cimatti A, Clarke E, et al. Symbolic model checking without BDDs [M]. Springer, 1999.

[40] Holzmann G. The Model Checker SPIN [J]. IEEE Transactions on Software Engineering. 1997, 23 (5): 279–295. Special Issue on Methods in Software Practice.

[41] Holzmann G J. PAN: a protocol specification analyzer [R]. 1981.

[42] Holzmann G J. Tracing protocols [J]. AT&T technical journal. 1985, 64 (10): 2413–2433.

[43] Holzmann G J. Design and validation of protocols [C]. In Tutorial Computer Networks and ISDN Systems. 1990.

[44] Dill D L, Drexler A J, Hu A J, et al. Protocol Verification as a Hardware Design Aid. [C]. In ICCD. 1992: 522–525.

[45] Ip C N, Dill D L. Better verification through symmetry [J]. Formal methods in system design. 1996, 9 (1-2): 41–75.

[46] Ip C N, Dill D L. State reduction using reversible rules [C]. In Proceedings of the 33rd annual Design Automation Conference. 1996: 564–567.

[47] Ip C N, Dill D L. Verifying systems with replicated components in Murϕ [C]. In Computer aided verification. 1996: 147–158.

[48] Clarke E M, Emerson E A, Sistla A P. Automatic verification of finite-state concurrent systems using temporal logic specifications [J]. ACM Transactions on Programming Languages and Systems (TOPLAS). 1986, 8 (2): 244–263.

[49] Clarke E, Grumberg O, Hiraishi H, et al. Verification of the Futurebus+ cache coherence protocol [R]. 1992.

[50] De Moura L, Bjørner N. Z3: An efficient SMT solver [J]. Tools and Algorithms for the Construction and Analysis of Systems. 2008: 337–340.

[51] Alglave J, Kroening D, Tautschnig M. Partial Orders for Efficient Bounded Model Checking of Concurrent Software [C]. In Computer Aided Verification (CAV). 2013: 141–157.

[52] Clarke E, Kroening D, Ouaknine J, et al. Computational Challenges in Bounded Model Checking [J]. Software Tools for Technology Transfer (STTT). 2005, 7 (2): 174–183.

[53] Daniel Kroening M P. EBMC model checker. `http://www.cs.utah.edu/formal_verification/Murphi/`.

[54] Brayton R K, Hachtel G D, Sangiovanni-Vincentelli A, et al. VIS: A system for verification and synthesis [C]. In Computer Aided Verification. 1996: 428–432.

[55] Group T V. Verification Interacting with Synthesis. `http://vlsi.colorado.edu/~vis/`.

[56] Berkeley A. A system for sequential synthesis and verification. 2005.

[57] Berkeley A. A system for sequential synthesis and verification. `http://www.eecs.berkeley.edu/~alanmi/abc/`.

[58] Clarke E, Kroening D, Sharygina N, et al. SATABS: SAT-based predicate abstraction for ANSI-C [M] // Clarke E, Kroening D, Sharygina N, et al. Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2005: 2005: 570–574.

[59] Andrews T, Qadeer S, Rajamani S K, et al. Zing: A model checker for concurrent software [C]. In Computer Aided Verification. 2004: 484–487.

[60] Cordeiro L, Fischer B, Marques-Silva J. SMT-based bounded model checking for embedded ANSI-C software [C]. In Automated Software Engineering, 2009. ASE'09. 24th IEEE/ACM International Conference on. 2009: 137–148.

[61] Beyer D, Keremoglu M E. CPACHECKER: A tool for configurable software verification [C]. In Computer Aided Verification. 2011: 184–190.

[62] Merz F, Falke S, Sinz C. LLBMC: Bounded model checking of C and C++ programs using a compiler IR [M] // Merz F, Falke S, Sinz C. Verified Software: Theories, Tools, Experiments. Springer, 2012: 2012: 146–161.

[63] Larsen K G, Pettersson P, Yi W. UppAal in a NutShell [J]. 1997, 1: 134–153.

[64] Bozga M, Daws C, Maler O, et al. Kronos: A model-checking tool for real-time systems [C]. In Formal Techniques in Real-Time and Fault-Tolerant Systems. 1998: 298–302.

[65] Campos S, Clarke E, Marrero W, et al. Verus: a tool for quantitative analysis of finite-state real-time systems [J]. ACM SIGPLAN Notices. 1995, 30 (11): 70–78.

[66] Kwiatkowska M, Norman G, Parker D. PRISM: Probabilistic symbolic model checker [M] // Kwiatkowska M, Norman G, Parker D. Computer performance evaluation: modelling techniques and tools. Springer, 2002: 2002: 200–204.

[67] Katoen J-P, Khattri M, Zapreev I S. A Markov reward model checker [C]. In Quantitative Evaluation of Systems, 2005. Second International Conference on the. 2005: 243–244.

[68] Hermanns H, Katoen J-P, Meyer-Kayser J, et al. A tool for model-checking Markov chains [J]. International Journal on Software Tools for Technology Transfer. 2003, 4 (2): 153–172.

[69] Ernst Moritz Hahn S S, Guangyuan Li, Zhang L. IscasMC model checker. `http://iscasmc.ios.ac.cn/IscasMC`.

[70] Henzinger T A, Ho P-H, Wong-Toi H. HyTech: A model checker for hybrid systems [C]. In Computer aided verification. 1997: 460–463.

[71] Silva B I, Krogh B H. Formal verification of hybrid systems using CheckMate: a case study [C]. In American Control Conference, 2000. Proceedings of the 2000. 2000: 1679–1683.

[72] Frehse G. PHAVer: Algorithmic verification of hybrid systems past HyTech [M] // Frehse G. Hybrid Systems: Computation and Control. Springer, 2005: 2005: 258–273.

[73] Vardi M Y. Branching vs. Linear Time: Final Showdown [C]. In TACAS'01. 2001: 1–22.

[74] Pnueli A. Linear and Branching Structures in the Semantics and Logics of Reactive Systems [C] // Brauer W. In International Colloquium on Automata, Language and Programming. 1985: 15–32.

[75] Boigelot B, Legay A, Wolper P. Iterating Transducers in the Large [C]. In Proc. 15th Int. Conf. on Computer Aided Verification. Boulder, July 2003: 223–235.

[76] Sistla A P, Vardi M Y, Wolper P. The Complementation Problem for Büchi Automata with Applications to Temporal Logic [C]. In Proc. 10th Int. Colloquium on Automata, Languages and Programming. Nafplion, July 1985: 465–474.

[77] Sistla A P, Vardi M Y, Wolper P. The Complementation Problem for Büchi Automata with Applications to Temporal Logic [J]. Theoretical Computer Science. 1987, 49: 217–237.

[78] Vardi M Y, Wolper P. Reasoning about Infinite Computations [J]. Information and Computation. 1994, 115 (1): 1–37.

[79] Kupferman O, Piterman N, Vardi M Y. Extended Temporal Logic Revisited. [C]. In Proc. 12th International Conference on Concurrency Theory. Denmark, 2001: 519–535.

[80] Pnueli A, Zaks A. PSL Model Checking and Run-time Verification via Testers [C] // Nipkow J M T, Sekerinski E. In Formal Methods (2006). 2006: 573–586.

[81] WLiu, Wang J, HChen, et al. Symbolic Model Checking APSL [J]. Frontiers of Computer Science in China. 2009, 3 (1): 130–141.

[82] Legay A, Wolper P. On the Use of Automata-based Techniques in Symbolic Model Checking [J]. Electronic Notes in Theoretical Computer Science. 2006, 150 (1).

[83] Legay A. T(O)RMC: A Tool for (ω)-Regular Model Checking [C] // Jupta A, Malik S. In The 20th International Conference on Computer Aided Verification (CAV'08). Princeton, N,J, USA, July 2008: 548–551.

[84] Cimatti A, Pistore M, Roveri M, et al. Improving the encoding of LTL model checking into SAT [C]. In Verification, Model Checking, and Abstract Interpretation. 2002: 196–207.

[85] Frisch A, Sheridan D, Walsh T. A fixpoint based encoding for bounded model checking [C]. In Formal Methods in Computer-Aided Design. 2002: 238–255.

[86] Clarke E, Kroening D, Ouaknine J, et al. Completeness and Complexity of Bounded Model Checking [C] // Steffen B, Levi G. In VMCAI'04. 2004: 85–96.

[87] Latvala T, Biere A, Heljanko K, et al. Simple Bounded LTL Model Checking [C] // Hu A, Martin A. In Formal Methods in Computer-Aided Design 2004 (FMCAD'04). 2004: 186–200.

[88] Latvala T, Biere A, Heljanko K, et al. Simple is better: Efficient bounded model checking for past LTL [C]. In VMCAI'05. 2005: 380–395.

[89] Ben-David S, Fisman D, Ruah S. Embedding Finite Automata within Regular Expressions [J]. Theoretical Computer Science. 2008, 404: 202–218.

[90] 刘万伟．扩展时序逻辑的推理及符号化模型检验技术 [D]．湖南长沙：国防科学技术大学，2009．

[91] Jehle M, Johannsen J, Lange M, et al. Bounded model checking for all regular properties [J]. Electronic Notes in Theoretical Computer Science. 2006, 144 (1): 3–18.

[92] Heljanko K, Junttila T, Keinänen M, et al. Bounded model checking for weak alternating büchi automata [C]. In Computer Aided Verification. 2006: 95–108.

[93] Heljanko K, Junttila T, Latvala T. Incremental and Complete Bounded Model Checking for Full LTL [C] // Etessami K, Rajamani S K. In 17th International Conference of Computer Aided Verification (CAV'05). 2005: 98–111.

[94] Kroening D, Ouaknine J, Strichman O, et al. Linear completeness thresholds for bounded model checking [C]. In Computer Aided Verification. 2011: 557–572.

[95] Bundala D, Ouaknine J, Worrell J. On the Magnitude of Completeness Thresholds in Bounded Model Checking [C]. In Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science. 2012: 155–164.

[96] Chauhan P, Clarke E M, Kroening D. Using SAT based image computation for reachability analysis [J]. 2003.

[97] McMillan K L. Applying SAT methods in unbounded symbolic model checking [C]. In Computer Aided Verification. 2002: 250–264.

[98] Gupta A, Yang Z, Ashar P, et al. SAT-Based Image Computation with Application in Reachability Analysis [C]. In FMCAD. 2000: 354–371.

[99] Gupta A, Yang Z, Ashar P, et al. Partition-Based Decision Heuristics for Image Computation Using SAT and BDDs [C]. In ICCAD. 2001: 286–292.

[100] Chauhan P, Clarke E, Kukula J, et al. Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis [C]. In Formal Methods in Computer-Aided Design. 2002: 33–51.

[101] McMillan K L, Amla N. Automatic abstraction without counterexamples [M] // McMillan K L, Amla N. Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2003: 2003: 2–17.

[102] Sheeran M, Singh S, Stålmarck G. Checking safety properties using induction and a SAT-solver [C]. In Formal Methods in Computer-Aided Design. 2000: 127–144.

[103] McMillan K L. Interpolation and SAT-based model checking [C]. In Computer Aided Verification. 2003: 1–13.

[104] Henzinger T A, Jhala R, Majumdar R, et al. Abstractions from proofs [C]. In ACM SIGPLAN Notices. 2004: 232–244.

[105] McMillan K L. Lazy abstraction with interpolants [C]. In Computer Aided Verification. 2006: 123–136.

[106] Jhala R, McMillan K L. A practical and complete approach to predicate refinement [M] // Jhala R, McMillan K L. Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2006: 2006: 459–473.

[107] Jhala R, McMillan K L. Interpolant-based transition relation approximation [C]. In Computer Aided Verification. 2005: 39–51.

[108] Dai L, Xia B, Zhan N. Generating non-linear interpolants by semidefinite programming [C]. In Computer Aided Verification. 2013: 364–380.

[109] Vizel Y, Grumberg O, Shoham S. Intertwined forward-backward reachability analysis using interpolants [M] // Vizel Y, Grumberg O, Shoham S. Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2013: 2013: 308–323.

[110] Een N, Mishchenko A, Brayton R. Efficient implementation of property directed reachability [C]. In Formal Methods in Computer-Aided Design (FMCAD), 2011. 2011: 125–134.

[111] Hoder K, Bjørner N. Generalized property directed reachability [M] // Hoder K, Bjørner N. Theory and Applications of Satisfiability Testing–SAT 2012. Springer, 2012: 2012: 157–171.

[112] Cimatti A, Griggio A. Software model checking via IC3 [C]. In Computer Aided Verification. 2012: 277–293.

[113] Chockler H, Ivrii A, Matsliah A, et al. Incremental formal verification of hardware [C]. In Proceedings of the International Conference on Formal Methods in Computer-Aided Design. 2011: 135–143.

[114] Hassan Z, Bradley A R, Somenzi F. Incremental, inductive CTL model checking [C]. In Computer Aided Verification. 2012: 532–547.

[115] Welp T, Kuehlmann A. QF BV model checking with property directed reachability [C]. In Proceedings of the Conference on Design, Automation and Test in Europe. 2013: 791–796.

[116] Büchi J. On A Decision Method in Restricted Second Order Arithmetic [C]. In Proc. Int. Congr. Method and Philosophy of Science 1960. Palo Alto, CA, USA, 1962: 1–12.

[117] Trakhtenbrot B. Finite Automata and The Logic of One-Place Predication [J]. Sibirian Mathnatical Journal. 1962, 3 (2): 102–131.

[118] McNaughton R. Testing and Generating Infinite Sequences by A Finite Automaton [J]. Information and Computation. 1966, 9: 521–530.

[119] Rabin M. Decidability of Second Order Theories and Automata on Infinite Trees [J]. Transaction of the AMS. 1969, 141: 1–35.

[120] Burstall R M. Program Proving as Hand Simulation with A Little Induction [C]. In IFIP Congress. 1974: 308–312.

[121] Kröger F. Lar: A Logic of Algorithmic Reasoning [J]. Acta Inf. 1977, 8: 243–266.

[122] Gabbay D. The Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems [C] // Banieqbal B. In Temporal Logic in Specification. 1989: 431–448.

[123] Laroussinie F, Markey N, Schnoebelen P. Temporal logic with forgettable past [C]. In Logic in Computer Science, Symposium on. 2002: 383–383.

[124] French T, Reynolds M. A Sound and Complete Proof System for QPTL. [C]. In Proceedings of 4th Advances in Modal Logic. 2003: 127–148.

[125] Kesten Y, Pnueli A. A Complete Proof Systems for QPTL [C]. In Logic in Computer Science. 1995: 2–12.

[126] Etessami K. Stutter-invariant languages, ω-automata, and temporal logic [C]. In Computer Aided Verification. 1999: 236–248.

[127] Wu Z. On the Expressive Power of QLTL [C]. In Proc. 4th International Colloquium on Theoretical Aspects of Computing (ICTAC'07). Macau, China, September 2007: 337–341.

[128] Garey M R, Johnson D S. Computers and intractability [M]. freeman San Francisco, 1979.

[129] Zhang L, Malik S. The quest for efficient boolean satisfiability solvers [C]. In Computer Aided Verification. 2002: 17–36.

[130] Davis M, Logemann G, Loveland D. A machine program for theorem-proving [J]. Communications of the ACM. 1962, 5 (7): 394–397.

[131] Davis M, Putnam H. A computing procedure for quantification theory [J]. Journal of the ACM (JACM). 1960, 7 (3): 201–215.

[132] Gu J. Local search for satisfiability (SAT) problem [J]. Systems, Man and Cybernetics, IEEE Transactions on. 1993, 23 (4): 1108–1129.

[133] Puri R, Gu J. A BDD SAT solver for satisfiability testing: an industrial case study [J]. Annals of Mathematics and Artificial Intelligence. 1996, 17 (2): 315–337.

[134] Sheeran M, Stålmarck G. A tutorial on Stålmarck’s proof procedure for propositional logic [C]. In Formal Methods in Computer-Aided Design. 1998: 82–99.

[135] Groote J F, Warners J P. The propositional formula checker HeerHugo [J]. Journal of Automated Reasoning. 2000, 24 (1-2): 101–125.

[136] Marques-Silva J P, Sakallah K A. GRASP: A search algorithm for propositional satisfiability [J]. Computers, IEEE Transactions on. 1999, 48 (5): 506–521.

[137] Bayardo Jr R J, Schrag R. Using CSP look-back techniques to solve real-world SAT instances [C]. In AAAI/IAAI. 1997: 203–208.

[138] Zhang H. SATO: An efficient prepositional prover [M] // Zhang H. Automated Deduction—CADE-14. Springer, 1997: 1997: 272–275.

[139] Zhang L, Madigan C F, Moskewicz M H, et al. Efficient conflict driven learning in a boolean satisfiability solver [C]. In Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design. 2001: 279–285.

[140] Goldberg E, Novikov Y. BerkMin: A fast and robust SAT-solver [J]. Discrete Applied Mathematics. 2007, 155 (12): 1549–1561.

[141] Eén N, Sörensson N. An extensible SAT-solver [C]. In Theory and applications of satisfiability testing. 2004: 502–518.

[142] Tseitin G S. On the complexity of derivation in propositional calculus [M] // Tseitin G S. Automation of Reasoning. Springer, 1983: 1983: 466–483.

[143] Pudlák P. Lower bounds for resolution and cutting plane proofs and monotone computations [J]. Journal of Symbolic Logic. 1997: 981–998.

[144] Bradley A R. SAT-based model checking without unrolling [C]. In Verification, Model Checking, and Abstract Interpretation. 2011: 70–87.

[145] Bradley A R, Manna Z. Checking safety by inductive generalization of counterexamples to induction [C]. In Formal Methods in Computer Aided Design, 2007. FMCAD'07. 2007: 173–180.

[146] Biere A, Heljanko K, Junttila T, et al. Linear encodings of bounded LTL model checking [J]. Logical Methods in Computer Science. 2006.

[147] Gallegos I, Ochoa O, Gates A Q, et al. A Property Specification Tool for Generating Formal Specifications: Prospec 2.0. [C]. In SEKE. 2008: 273–278.

[148] Schneider K. Improving automata generation for linear temporal logic by considering the automaton hierarchy [C]. In Logic for Programming, Artificial Intelligence, and Reasoning. 2001: 39–54.

[149] Armin Biere K H, Wieringa S. AIGER.

[150] Tauriainen H, Heljanko K. Testing LTL formula translation into Büchi automata [J]. International Journal on Software Tools for Technology Transfer. 2002, 4 (1): 57–70.

[151] Martin A J. The Design of A Self-Timed Circuit for Distributed Mutual Exclusion [C] // Fuchs H. In Proceedings of the 1985 Chapel Hill Conference on Very Large Scale Integration. 1985.

[152] Cavada R, Cimatti A, Jochim C A, et al. NuSMV 2.5 User Manual. http://nusmv.fbk.eu/NuSMV/userman/v25/nusmv.pdf, Apr., 2010.

[153] Somenzi F. CUDD: CU decision diagram package release 2.3. 0 [J]. University of Colorado at Boulder. 1998.

[154] Bradley A R, Somenzi F, Hassan Z, et al. An incremental approach to model checking progress properties [C]. In Formal Methods in Computer-Aided Design (FMCAD), 2011. 2011: 144–153.

作者在学期间取得的学术成果

发表的学术论文

[1] Rui Wang, Wanwei Liu, Tun Li, Xiaoguang Mao and Ji Wang. Bounded Model Checking ETL Cooperating with Finite and Looping Automata Connectives. Journal of Applied Mathematics, Volume 2013, Article ID 462532. (SCI 收录，检索号：000322477200001)

[2] Wanwei Liu, Rui Wang, Xianjin Fu, Ji Wang, Wei Dong and Xiaoguang Mao. Counterexample-Preserving Reduction for Symbolic Model Checking. Journal of Applied Mathematics, 2014. (SCI 收录，已录用)

[3] Rui Wang, Wanwei Liu, Xiaoguang Mao and Tun Li. Incremental semantic Bounded Model Checking LTL. IEEE 2nd International Conference on Computer Science and Electronic Engineering (ICCSEE), 2013. (EI 收录，检索号：20134116828831)

[4] Rui Wang. Interpolation based model checking with macro step. IEEE 3rd International Conference on Computer Science and Automation Engineering (CSAE), 2013. (EI 收录，已录用)

[5] Wanwei Liu, Rui Wang, Xianjin Fu, Ji Wang, Wei Dong and Xiaoguang Mao. Counterexample-Preserving Reduction for Symbolic Model Checking. The 10th Theoretical Aspects of Computing (ICTAC), 2013. (EI 收录，检索号：20134116837189)

[6] Wanwei Liu, Xiaoguang Mao, Geguang Pu and Rui Wang. Combining Syntactic and Semantic Encoding for LTL Bounded Model Checking. The 8th International Symposium on Theoretical Aspects of Software Engineering (TASE), 2014. (EI 收录，已录用)

[7] Rui Wang, Wanwei Liu, Lijun Zhang, Yong Li, Xiaoguang Mao. Bidirectional Property Directed Reachability. (投稿中)

附录 A　附录

A.1　DME 电路问题

```
MODULE and-gate(in1,in2)
VAR
  out : boolean;
ASSIGN
  init(out) := FALSE;
  next(out) := (in1 & in2) union out;

MODULE and-gate-init(in1,in2,init-out)
VAR
  out : boolean;
ASSIGN
  init(out) := init-out;
  next(out) := (in1 & in2) union out;

MODULE or-gate(in1,in2)
VAR
  out : boolean;
ASSIGN
  init(out) := FALSE;
  next(out) := (in1 | in2) union out;

MODULE c-element(in1,in2)
VAR
  out : boolean;
ASSIGN
  init(out) := FALSE;
  next(out) :=
    case
      in1 = in2 : in1 union out;
      TRUE : out;
    esac;

MODULE mutex-half(inp,other-out)
VAR
```

```
  out : boolean;
ASSIGN
  init(out) := FALSE;
  next(out) := inp union out;
TRANS
  !(next(out) & next(other-out))

MODULE user
VAR
  req : boolean;
ASSIGN
  init(req) := FALSE;
  next(req) := (!ack) union req;

MODULE cell(left,right,token)
VAR
  q : and-gate(f.out,n.out);
  f : c-element(d.out,i.out);
  d : and-gate(b.out,!u.ack);
  b : mutex-half(left.req,a.out);
  i : and-gate(h.out,!j.out);
  h : c-element(g.out,j.out);
  n : and-gate-init(!e.out,!m.out,!token);
  u : user;
  a : mutex-half(u.req,b.out);
  c : and-gate(a.out,!left.ack);
  g : or-gate(c.out,d.out);
  e : c-element(c.out,i.out);
  k : and-gate(g.out,!h.out);
  l : and-gate(k.out,m.out);
  p : and-gate(k.out,n.out);
  m : and-gate-init(!f.out,!n.out,token);
  r : and-gate(e.out,m.out);
  j : or-gate(l.out,ack);
DEFINE
  req := p.out;
  left.ack := q.out;
  u.ack := r.out;
```

```
MODULE main
VAR
  e-3 : process cell(e-1,e-2,TRUE);
  e-2 : process cell(e-3,e-1,FALSE);
  e-1 : process cell(e-2,e-3,FALSE);

-- Safety
LTLSPEC G ((!(e-1.u.ack & e-2.u.ack) & !(e-1.u.ack & e-
3.u.ack)) & !(e-2.u.ack & e-3.u.ack))
-- Liveness
LTLSPEC G((e-1.req -> F e-1.u.ack) & (e-2.req -> F e-2.u.ack) & (e-
3.req -> F e-3.u.ack))
```

A.2 二进制累加器问题

```
MODULE main
VAR
   bit_0: counter(1);
   bit_1: counter(bit_0.carry_out);
   bit_2: counter(bit_1.carry_out);

MODULE counter(carry_in)
VAR
   pre_value: 0..1;
   value: 0..1;
ASSIGN
   init(value):= 0;
   init(pre_value) := 0;
   next(pre_value) := value;
   next(value):= (carry_in + value) mod 2;
DEFINE
   carry_out:= carry_in * pre_value;

CONNECTIVE C_2(a,b) : LOOP
STATES
   >st_1, st_2;
TRANSITIONS(st_1)
  case
a: st_2;
```

```
  esac;
TRANSITIONS(st_2)
  case
b: st_1;
  esac;

ETLSPEC C_2(TRUE, (bit_0.value=1))
```