基于LNFG和Büchi自动机的规范挖掘

作者姓名 宁新亚

指导教师姓名、职称 张南 教授

申请学位类别 工学硕士

学校代码 10701 学 号 20031211417
分 类 号 TP31 密 级 公开

西安电子科技大学

硕士学位论文

基于LNFG和Büchi自动机的规范挖掘

作者姓名：宁新亚

一级学科：计算机科学与技术

二级学科（研究方向）：

学位类别：工学硕士

指导教师姓名、职称：张南 教授

学　　院：计算机科学与技术学院

提交日期：2023 年 6 月

Specification Mining based on LNFG and Büchi Automaton

A thesis submitted to

XIDIAN UNIVERSITY

in partial fulfillment of the requirements

for the degree of Master

in Computer Science and Technology

By

Ning Xinya

Supervisor: Zhang Nan Title: Professor

June 2023

摘要

近年来，国内外由于软硬件系统故障带来的灾难不计其数。形式化验证技术是指采用形式化方法对计算机软件或硬件系统进行验证的方法，可以帮助开发人员发现系统中潜在的问题，减少开发时间和成本。为系统书写规范是形式化验证中不可或缺也是极为重要的一步。但是，由于系统设计、建模的复杂性和运行的困难性等原因，手动编写规范仍是个耗时费力的任务。规范挖掘是一种自动化挖掘系统规范的机器学习方法，目前在计算机领域已被广泛研究。

但是，当前挖掘算法和工具存在一些不足。首先，线性时序逻辑（Linear Temporal Logic，LTL）作为目前挖掘算法的主流性质描述语言，表达能力有限，如现有挖掘工具 Texada 可以挖掘任意 LTL 公式表示的规范。其次，命题投影时序逻辑（Propositional Projection Temporal Logic，PPTL）为挖掘程序中区间相关和周期性重复的性质提供了理论基础，虽然基于模式库的 PPTL 规范挖掘算法在表达能力上有了显著的提升，但是基于此算法开发的工具 PPTLMiner 在实际应用中存在挖掘时间过长的问题，甚至会出现内存崩溃的情况。另外，现有挖掘工具的功能较为单一，不能满足用户的个性化需求。本文基于上述问题开展研究，主要贡献如下：

（1）针对当前时序逻辑语言有限的性质描述能力，本文提出并实现了挖掘 LTL 和 PPTL 规范的算法。PPTL 在表达能力方面是完全正则的，既能够描述有穷区间的性质，也能描述无穷区间的性质。更重要的是，PPTL 公式能够描述系统某些更为复杂的性质，如区间相关性质和周期性重复性质。本文提出的挖掘算法支持使用任意的 LTL 公式和 PPTL 公式来描述系统的性质，并通过挖掘基于蚁群算法解决旅行商问题的规范，验证了本文提出的挖掘算法的有效性和实用性。

（2）针对目前挖掘 PPTL 规范的工具 PPTLMiner 存在的时间和内存效率较低的问题，本文提出了一种基于带标记的范式图（Labeled Norma Form Graph，LNFG）和 Büchi 自动机的挖掘算法，PPTLMiner+是基于此算法设计开发的挖掘工具。若性质公式使用 PPTL 公式表示，则将其转为 LNFG。反之，将性质公式转为 Büchi 自动机。然后将 LNFG（或 Büchi 自动机）和程序执行轨迹进行检测，从而挖掘出满足轨迹的规范。本文将挖掘工具 PPTLMiner+与 PPTLMiner 和 Texada 进行了对比实验，实验结果表明本文提出的挖掘算法表达能力较强、挖掘所需的时间少且没有出现内存崩溃的情况。同时，通过分析时间复杂度和空间复杂度，利用理论方法证明了本文挖掘算法的优越性。

（3）针对现有挖掘工具功能较为单一的问题，本文开发挖掘工具 PPTLMiner+时，以实现规范挖掘的基础功能为前提，添加了多个特色功能。如：支持用户自定

义轨迹分隔符和事件分隔符、使用正则表达式定义轨迹中的事件、指定性质公式中的原子命题为不变事件和使用多原子命题表示轨迹的事件等。

关 键 词：形式化验证， 规范挖掘， 时序逻辑， 带标记的范式图， Büchi 自动机

ABSTRACT

In recent years, numerous disasters have occurred as a result of software and hardware system failures both at home and abroad. The method of applying formal methods to check computer software and hardware systems is known as formal verification technology. It can help developers uncover possible issues within the system, ultimately decreasing on both the time and the cost of development. The process of writing system specifications is an important and crucial step in formal verification. The complexity of system design and the growing complexity of modeling and operation, however, make manual specification writing a time-consuming and hard task. Specification mining is a machine learning technique that has gained a lot of attention in computer fields for automatically mining specifications.

The current mining tools and algorithms, however, also have several limitations. First of all, Linear Temporal Logic(LTL) is generally used as the property description language, which has a limited expression ability. For example, the current specification mining tool Texada can mine the specifications represented by any LTL formula. Secondly, Propositional Projection Temporal Logic(PPTL) establishes a theoretical foundation for mining the properties of interval correlation and periodic repetition in programs. The limited expression capabilities of LTL can be efficiently improved by the pattern-library-based PPTL specification mining method, but the tool PPTLMiner built based on this algorithm has issues with too long mining time and even memory corruption in implementations. In addition, the functions of the existing mining tools are relatively unitary, and cannot meet the personalized requirements of users. The research topic of this paper is carried out based on the issues mentioned above, the main contributions are as follows:

(1) In view of the limited expression ability of the property description language used by the current specification mining algorithms, this paper proposes and implements an algorithm for mining both LTL and PPTL specifications. PPTL is completely regular in terms of expression ability, which can describe the properties of both finite intervals and infinite intervals. More significantly, the PPTL formula may describe some more complex properties of the system, such as interval correlation and periodic repetition. The mining algorithm proposed in this paper supports any LTL formula and PPTL formula to describe the property of the system. In this paper, the specifications for solving the traveling salesman problem

using the Ant Colony Algorithm are mined in order to verify the effectiveness and practicability of the mining method and tools.

(2) Considering the long time and low memory efficiency of the current PPTL-based mining tool PPTLMiner, this paper proposes a mining algorithm based on Labeled Normal Form Graph (LNFG) and Büchi automata. PPTLMiner+ is a mining tool designed and developed based on this algorithm. If the property formula is expressed by PPTL formula, it will be converted to LNFG. On the contrary, it is transformed into Büchi automata. Then, the program execution traces are used to detect with LNFG(or Büchi automata) in order to mine the specifications that satisfy the traces.This paper compares the mining tool PPTLMiner+ with PPTLMiner and Texada. The results show that the mining algorithm proposed in this paper has a better expression ability, less time required for mining and no memory corruption. Meanwhile, by analyzing the time complexity and space complexity, the superiority of the mining algorithm in this paper is proved by theoretical methods.

(3) In consideration of existing mining tools, when developing the mining tool PPTLMiner+ in this paper, on the premise of realizing the basic function of specification mining, multiple features are added. For example, PPTLMiner+ supports user-defined trace separators and event separators, using regular expressions to define events in a trace, specifying atomic propositions in property formulas as invariant events, and using multi-atomic propositions to represent events in a trace.

Keywords: Formal Verification, Specification Mining, Temporal Logic, Labeled Normal Form Graph, Büchi Automata

插图索引

表格索引

符号对照表

符号	符号名称
¬	逻辑非
∨	逻辑或
∧	逻辑与
→	蕴含
↔	等价
◊	存在
□	总是
prj	投影
;	Chop
U	Until
◯、X	Next
⊙、W	Weak Next
s	状态
σ	区间
ω	时间上的无穷
τ	轨迹

缩略语对照表

缩略语	英文全称	中文对照
SAT	Satisfiability	布尔可满足性问题
SMT	Satisfiability Modulo Theories	可满足性模理论
TL	Temporal Logic	时序逻辑
FA	Finite Automaton	有限自动机
LTL	Linear Temporal Logic	线性时序逻辑
CTL	Computing Tree Logic	计算树逻辑
NF	Normal Form	范式
PPTL	Propositional Projection Temporal Logic	命题投影时序逻辑
LNFG	Labeled Normal Form Graph	带标记的范式图
TLA	Temporal Logic of Actions	行为时序逻辑
PITL	Propositional Interval Temporal Logic	命题区间时序逻辑
MSVL	Modeling Simulation and Verification Language	建模仿真验证语言
FSM	Finite State Machine	有穷状态机
RTL	Register Transfer Level	寄存器转换级电路
VWAA	Very Weak Alternating co-Büchi Automata	极弱交替式 co-Büchi 自动机
NNF	Negation Normal Form	否定范式
TGBA	Transitions Bases Generalized Büchi Automata	基于变迁的 Büchi 自动机
TSP	Traveling Salesman Problem	旅行商问题
NP	Non-deterministic Polynomial	非确定性多项式
DFS	Depth First Search	深度优先搜索
BFS	Breadth First Search	广度优先搜索

目 录

第一章　绪论

1.1 研究背景和意义

1.1.1　研究背景

1994 年，Lynchburg College 数学系教授 Thomas Nicely 发现了 Intel 的 Pentium 处理器在浮点运算单元上存在的错误（FDIV Bug），使得 Intel 公司强制召回了多个批次的该款处理器，造成了 4.7 亿美元的经济损失。1996 年，欧洲航天局发射的 Ariane5 火箭仅在发射 37 秒之后就解体爆炸，不仅直接造成了 3.7 亿美元的经济损失，更使科研人员十年的努力付之东流，而造成这一严重后果的原因仅仅是一行代码的错误。2011 年 7 月 23 日，发生了一场列车追尾事故，当事列车分别是由杭州站开往福州南站的 D3115 次列车和由北京南站开往福州站的 D301 次列车。由于温州南站的信号设备系统存在严重缺陷，这次事故造成了六节轨道完全脱离轨道，40 人死亡，172 人受伤，给人们带来了巨大的痛苦。1965 年，图灵奖得主托尼·霍尔（Tony Hoare）在设计第一个全面的引用系统时，引入了 Null 引用，他之所以这样做，仅仅是因为 Null 引用实现起来非常容易，然而这样的引用系统导致了无数系统故障和程序错误等严重后果的发生，且可能在未来几十年内带来十亿美元的损失。2009 年 Hoare 发表了以《Null References: The Billion Dollar Mistake》为主题的演讲，并在同年出版的《Communication of the ACM》杂志中表示，如何证明程序的正确性仍然是计算机领域中有待探究的难点。

以上种种案例表明，软硬件系统的缺陷和错误可能会导致巨大的财力损失甚至是生命财产损失，尤其是对安全性要求较高的系统。因此，系统在投入使用之前进行验证是必不可少的一步。传统的验证技术以基于仿真的测试为中心，在测试中设计大量的定向或随机测试用例，并检查设计行为的正确性。但是人工设计的测试用例很难做到覆盖全部程序路径，而且“设计仿真系统→编写测试用例→验证”这一过程耗时耗力，在实际工业生产中并不是最优选择。

形式化方法（Formal Methods）是基于严格的数学语言对系统进行描述、开发和验证的技术。形式化方法开发程序的流程可以描述为：Requirements→$spec_1$→⋯$spec_n$→Program。每一个箭头都表示一次精化，每一个 Spec（Specification）的抽象角度和层次是不同的。如果能够保证需求是完全正确的，且通过验证确保每一步的精化也是保持正确的，那么就能证明最后编写的程序满足最初提出的需求。上述过程中书写规范就是使用数学语言描述程序中的函数应该做的事情，并且需要指定函数执行的顺序。

形式化验证（Formal Verification）技术是指采用形式化方法对计算机软件系统或硬件系统进行验证的过程[1]，图 1.1 简要展示了形式化验证的流程。第一步，根据需求对系统建立模型。第二步，用时序逻辑语言描述系统需要满足的条件，即待验证的性质。第三步，选择合适的形式化工具，并将模型和性质输入验证工具，运行工具获得验证结果。第四步，分析结果从而判断验证是否通过，若通过，则执行第五步；若不通过，则重新修改模型，再次运行验证，直至验证通过。第五步，记录形式化验证的结果、过程和方法，形成验证文档。形式化验证的结果指明系统能够正常运行需要满足的性质，即规范。

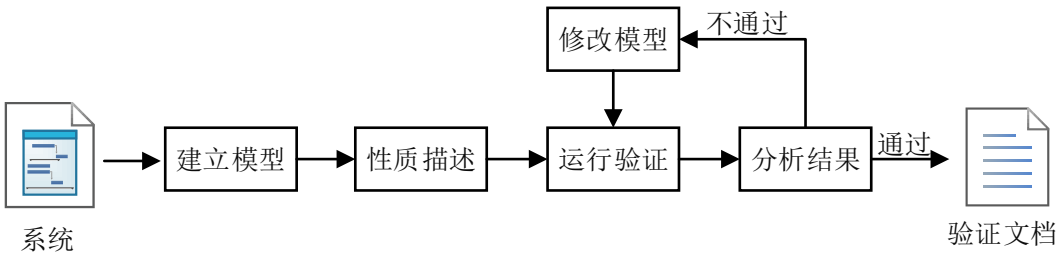


图1.1 形式化验证流程图

由此可以得出，形式化验证技术中编写规范是非常重要的步骤。规范是计算机软硬件系统行为和性质的形式化描述[1,2]，通过使用数学语言和逻辑符号来描述系统的行为和约束条件，以此来确保系统的正确性和安全性。形式化的规范可以精确的描述系统行为和约束条件，从而避免歧义和不确定性；可以用于检查和验证系统的正确性和安全性，防止系统在正式投入使用时出现漏洞和错误；可以为系统的测试和调试提供准确的参照标准，减少测试和调试的时间和成本。综上所述，形式化规范可以提高系统的质量和可靠性，降低系统开发和维护的成本和风险，是软件和硬件系统开发过程中不可或缺的重要工具。

但是人工书写规范在实际生产中也存在诸多问题。如：（1）开发人员的抽象能力有限，因为形式化规范需要对系统进行抽象和建模，需要书写规范的人员具备一定的抽象能力和逻辑思维能力，否则可能会写出不完备、不准确的规范。（2）开发人员的知识储备不足，形式化规范需要涉及到数学、逻辑、计算机科学等多个领域的知识，如果相关知识储备不足，可能会影响规范的准确性。（3）工具支持不足，形式化规范需要使用专门的工具进行验证、模拟等操作，但是目前的形式化工具支持不够完善，可能会导致规范验证效率不高、操作不便等问题。（4）规范维护困难，因为形式化规范需要不断地更新和维护，否则可能会出现规范不适用于最新版本的系统、规范与实现不一致等问题。综上所述，人工书写形式化规范需要具备一定的能力和知识储备，同时还需要面对语言复杂、规范维护困难等问题，需要花费较多

的时间和精力来完成规范的书写和维护。

为解决以上问题，Ammons 等人最早提出规范挖掘是一种机器学习方法[3]。与传统的人工书写规范相比，规范挖掘具有诸多优点。首先，可以自动化地生成规范，避免了人工书写规范的复杂性和错误率。其次，可以通过系统的自动化分析和验证来检查和排除潜在的错误和漏洞，提高了系统的可靠性和安全性。最后，可以有效地降低规范的维护成本，并且能够与其他测试手段结合使用，形成完整的系统验证流程。因此，规范挖掘是当前软件和硬件系统开发中非常有前途的一种技术手段。

为了避免自然语言的二义性，计算机领域使用形式化语言编写规范，常用的形式化语言包括时序逻辑（Temporal Logic，TL）和有限自动机（Finite Automaton，FA）。这些形式化语言的应用，为计算机科学领域中的系统设计、验证和测试提供了重要的工具和方法。

1.1.2 研究意义

形式化语言的表达能力越强，系统性质的描述就越全面。目前规范挖掘算法通常使用 LTL 作为性质描述语言，如 Perracotta[4]、GoldMine[5]、IODINE[6]和 Texada[7]等，其中 Texada 可以挖掘任意用 LTL 公式描述的性质。虽然 LTL 在描述时序性质方面具有一定的优势，但由于其表达能力有限，无法对一些较为复杂的性质进行完备的描述。命题投影时序逻辑（Propositional Projection Temporal Logic，PPTL）则在表达能力上做到了完全正则[8,9]，除了 LTL 能够描述的时序性质之外，PPTL 还能描述软硬件系统的区间相关性质和周期性重复性质。虽然基于模式库的 PPTL 规范挖掘算法[10,11]可以在一定程度上弥补 LTL 表达能力有限的不足，但在实际应用中基于上述算法开发的工具 PPTLMiner 存在挖掘时间过长的问题，甚至出现内存崩溃的情况。因此，需要进一步优化挖掘 PPTL 规范的算法，减少挖掘时间，提升内存使用效率。

综上，本文的研究课题主要解决三个问题：

（1）LTL 公式的表达能力有限，不能描述一些比较复杂的系统性质。本文采用具有完全正则表达能力的 PPTL 公式描述性质，不仅能够描述无穷区间的性质，还可以描述有穷区间的性质。更重要的是，PPTL 公式能够描述系统的区间相关性质和周期性重复性质。

（2）现有挖掘 PPTL 规范的工具 PPTLMiner 在实际应用中存在挖掘时间过长和内存效率低下的问题，本文将优化挖掘算法，提高时间和内存效率。

（3）PPTLMiner 工具的功能较为单一，不能满足用户的多种需求。本文将调研现有挖掘工具的功能，增加多个特色功能。

针对以上问题，本文展开挖掘规范的算法设计和工具开发的研究，本文研究意义如下：

（1）使用规范描述计算机软件或硬件系统性质，有助于提高系统的可维护性、正确性和可靠性等，避免系统缺陷和漏洞的出现导致不可挽回的损失。

（2）自动化地挖掘规范，避免人工书写规范造成程序路径覆盖率低、人力财力消耗大等问题，减轻了形式化验证的负担。

（3）使用表达能力更强的命题投影时序逻辑 PPTL 描述系统规范，挖掘系统的区间相关性质和周期性重复性质。

（4）提出一个既能挖掘 LTL 规范也能挖掘 PPTL 规范的算法，本文将 LTL 公式转为 Büchi 自动机或将 PPTL 公式转为带标记的范式图 LNFG 后再进行规范挖掘，算法性能较好，进行规范挖掘时在不出现内存崩溃的基础上，挖掘时间更短。

（5）开发出一个自动化的并且具有一定健壮性的功能丰富的规范挖掘工具，用户只需指定系统的可执行程序和待挖掘的系统性质，即可通过挖掘工具获得系统的规范。

1.2 国内外研究现状

1.2.1 时序逻辑语言的研究现状

时序逻辑语言是在计算机科学领域中发展起来的一种形式化语言，用于描述系统的时间和行为之间的关系。随着计算机科学和人工智能领域的不断发展，时序逻辑语言也得到了不断的完善和发展。Pnueli 开创性地把线性时序逻辑引入计算机科学领域[12]，该逻辑提供了一种即直观又精确的表达方法来描述时序逻辑性质，且定义了常用的时序操作符X（next）、W（weak next）、U（until）、R（release）、F（future）、G（globally）等。

行为时序逻辑（Temporal Logic of Actions，TLA）[13]是 Lamport 提出的组合了时序逻辑和行为逻辑的一门语言。相较于 LTL，TLA 没有定义操作符X，而是使用x'代表x在下一状态的值。另外，TLA 还有一些特点，如：（1）TLA 强调状态变化和行为，不仅关注系统的时间顺序，还强调系统的状态变化和行为。（2）TLA 采用数学符号和语言，如谓词逻辑、集合论和函数论等，使得它的表达能力更加精确和严谨。（3）由于采用了比其他时序逻辑更丰富的表达方式，TLA 可以描述复杂的系统，如分布式系统、网络系统、并发系统等。

Harel 等人提出了一类基于 chop（;）操作符的逻辑[14]，公式$p;q$在一个区间上成立，当且仅当该区间被分为前后两个区间，且在前一个区间p成立，在后一个区间q成立。Chandra 等人深入探究 chop 操作符的定义并进行了细化[15]，从而更加准确地阐明了其含义和作用，使得基于 chop 操作符的逻辑得到了更为精确和细致的描述，并且在应用中表现出更高的效率和精度。

后来，Moszkowski 提出了命题区间时序逻辑（Propositional Interval Temporal Logic，PITL）[16]。相对于之前提出的时序逻辑，PITL 具有诸多优势。首先，PITL 可以描述时间区间内的性质，而不仅仅是单个时间点上的性质，因此 PITL 能够更全面地描述系统的性质和特征。其次，PITL 可以描述系统中性质的动态变化，如从某一状态到另一状态的变化，因此 PITL 可以更准确地描述系统的行为和演化过程。另外，PITL 是基于命题逻辑提出的时序逻辑，故语法比较简单，易于理解和使用。同时，PITL 也可以与其他逻辑语言相结合，如模态逻辑和一阶逻辑等。

Duan、Koutny 和 Holt 提出了命题投影时序逻辑 PPTL[17]，PPTL 将 PITL 的定义从有穷区间扩展到无穷区间范围，还扩展了 prj 操作符的含义。公式 $(P_1,\cdots,P_m)\ prj\ Q$ 表示 $(P_1,\cdots,P_m)$ 和 Q 在两个区间上成立，且这两个区间并行，因此 PPTL 更适合描述并行系统的性质。Tian 给出了 PPTL 的可满足性、复杂性和表达性的判定过程，还证明了 PPTL 表达能力的完全正则性[18,19]。近年来，PPTL 已广泛应用于模型检测领域和形式化验证领域，目前已经提出了基于 PPTL 的限界模型检测方法[20]和基于 PPTL 的符号模型检测方法[21]、基于 PPTL 的偏序模型检测方法[22]和支持索引式的 PPTL 定理证明器的实现[23]等研究成果。

目前，一个建模仿真验证平台 MSV 已经被开发并在大型软件系统的验证中得到应用。该平台包括一个基于 PPTL 的形式化验证工具集，该工具集又包含 PPTL 可满足性检测工具[19,22]和 PPTL 模型检测工具[20,21,23,24]。PPTL 可满足性检测工具包括四个子工具，分别是 PPTL2NF、PPTL2LNFG、PPTL2BA 和 PPTLSAT。PPTL 模型检测工具包括四个工具，分别是 PPTL4PMC、PPTL4BMC、PPTL4SMC 和 PPTL4PNMC。因此，MSV 可以为后续的形式化验证奠定基础，挖掘出系统规范后，可以依靠 PPTL 的形式化验证工具和 MSVL 验证工具以及语言之间的转换工具 C-to-MSVL 等对挖掘系统做进一步的验证。

1.2.2 规范挖掘的研究现状

将程序源代码输入不变量挖掘工具可以生成程序执行的轨迹。根据输入数据的类型不同，规范挖掘分析方法可以分为三类：静态分析、动态分析和混合分析方法。其中，静态分析是指将系统的程序源代码作为规范挖掘的原始输入，直接分析源代码，这种分析方法适用于结构较简单、函数个数较少的程序。动态分析是指将程序执行的轨迹作为规范挖掘的原始输入，执行程序源代码获取轨迹然后再分析。混合分析又叫做动静结合分析，即既分析程序源代码又分析轨迹。

根据性质描述公式在挖掘算法中所处的位置不同，规范挖掘又分为主动挖掘和被动挖掘。在主动挖掘中，挖掘算法以系统模型和描述性质的公式作为输入，输出满足系统模型的实例化公式。该过程大多将公式转为语法树或范式等形式，再根据

操作符的语义挖掘出规范。在被动挖掘中，性质公式是挖掘算法的输出，此时输入是描述性质所采用形式化语言的全部操作符、正例样本和反例样本。被动挖掘大多借助 SAT[25]、SMT[26]等求解器实现，通过求解器被动学习出满足正例样本而不满足反例样本的性质描述公式。

规范挖掘已广泛用于程序行为分析、系统模型构建和软件评估[27-29]。挖掘工具 Perracotta 从程序执行轨迹和性质模板中挖掘规范[30]，性质模板包含 8 种不同类型的属性，例如性质$MultiCause \stackrel{\text{def}}{=} (xx * y)*$表示 x 出现一次或多次会导致 y 的出现。Van der Aalst 等人开发了一种基于 finite-length 轨迹的通用 LTL 挖掘算法[31]，并将时序操作符与全称量词∀和存在量词∃相结合。Dwyer 等人定义了一组基于多种规格的 LTL 规范[32]，在这种规范中，一个描述规范的模板包括一个性质公式$Pattern$和这个公式成立的区间范围$Scope$，如模板“$Pattern$; $Scope$：never p;between q and r”表示在q成立后和r成立前的这个区间，$\neg p$成立。Synoptic[33]是一个从系统已经生产的日志中推断有穷状态机（Finite State Machine，FSM）模型的工具，能够挖掘三种用 LTL 公式描述的规范，准确性高且使用细化和粗化技术优化生成的模型。Lemieux 介绍了一种规范挖掘工具 Texada[7]，该工具提出了两个可以挖掘任何 LTL 公式的递归算法，并引入了 confidence、support 和 support-potential 的概念用以评估 finite-length 轨迹上的性质。第一个算法将程序执行的轨迹解析成线性形式，第二个算法将轨迹解析成形如“event:time-point”的映射形式。如第一个算法解析的轨迹是“a,a,b,c,a”；第二个算法解析的结果是“a:(0;1;4),b:(2),c:(3)”。Neider 和 Gavran 提出了两个从样本中学习 LTL 规范的算法，并提供了规范挖掘工具包 samples2LTL[34]。第一个学习算法将学习任务简化为一系列命题布尔逻辑中的可满足性问题，最终生成了最小的 LTL 公式。第二个学习算法将基于 SAT 的学习算法与用于学习决策树的算法相结合，更适用于大规模样本的规范挖掘，但无法保证生成最小的 LTL 公式。

规范挖掘也被用于硬件设计的验证，通常是为了构建系统的形式化规范。规范挖掘工具 IODINE[6]提供了许多规范挖掘分析，包括从分析波形中提取请求对、互斥锁、one-hot 信号等信息。IODINE 通过提取 FSM 转换模型，挖掘出片上协议的隐含规范，并揭示了测试覆盖率的不足。Li 等人引入 LTL 模板和增量跟踪的概念来处理多位信号[35]，将其应用于各种 RTL（Register Transfer Level）设计的挖掘规范，并使用挖掘出的规范来定位故障。GoldMine[5]将约束随机激励测试、静态分析和规范挖掘与形式化验证技术相结合，为 RTL 设计并生成一组经过验证的性质。Lyer 和 Kim 等人[36]提出了基于常见 RTL 设计习惯将多位信号转换为原子命题的策略，通过分析挖掘出细粒度的 LTL 规范，定位出只有在长时间运行的全系统模拟违反高级性质时才检测到的错误。

1.3 本文研究内容

本文采用动态分析和主动的分析方法来完成规范挖掘。主动的挖掘方法需要以系统性质的描述公式为输入，动态分析需要获得程序执行的轨迹。因此，首先需要给出性质描述公式和执行轨迹，然后用程序轨迹中的函数将公式进行实例化，最后通过检测性质公式和执行轨迹从而得到满足轨迹的实例化公式，即规范。本文在已有的 PPTL 规范挖掘算法的基础上进行提升和创新，主要工作如下：

（1）使用动态不变量挖掘工具 Daikon[37]获得程序执行的轨迹，同时使用轨迹过滤工具 DtraceFilter 提取程序中函数的工作轨迹作为本文挖掘算法的输入之一。

（2）用户自定义输入的性质公式，本文的挖掘算法支持使用任意的 LTL 公式和 PPTL 公式来描述待验证系统的ω−正则性质和完全正则性质。本文的挖掘算法先将 LTL 公式转为 Büchi 自动机、PPTL 公式转为 LNFG，然后再进行后续的挖掘流程。

（3）设计一个完整的规范挖掘算法，并开发出对应的挖掘工具，输入是程序执行轨迹和性质公式，输出是满足轨迹的规范。扩展本文开发的挖掘工具的功能，增加多个特色功能，满足用户的个性化需求。

（4）使用本文开发的挖掘工具挖掘经典算法的规范，证明本文提出算法的有效性和实用性。将本文开发的挖掘工具与其他挖掘工具进行对比实验，并进行理论分析，证明本文提出的挖掘算法的高效性和优越性。

1.4 论文组织架构

本文围绕基于 LNFG 和 Büchi 自动机的规范挖掘的算法以及与现有工具的对比实验展开研究，各个章节的内容安排如下：

第一章：绪论

本章首先列举了几个由于系统故障导致巨大损失的实例，从而引出本文的研究背景和研究意义；然后介绍了时序逻辑语言和规范挖掘算法的国内外研究现状，并对规范挖掘的类型进行了简要介绍；接着，阐述了本文的研究内容；最后，介绍了本文的组织架构。

第二章：相关理论与技术研究

本章主要介绍了与本文相关的理论和技术研究成果，以便读者能够更好地理解本文。理论研究成果首先介绍了命题投影时序逻辑 PPTL 和线性时序逻辑 LTL，然后阐述了 PPTL 的范式形式和带标记的范式图形式，最后介绍了 LTL 的 Büchi 自动机形式；技术研究成果包括程序不变量挖掘技术、PPTL 转 LNFG 技术和 LTL 转 Büchi 自动技术。除此之外，还简要介绍了这些技术的发展历程，并说明了在本文挖掘算法中采用的工具。

第三章：规范挖掘算法的设计与实现

本章是本文的重点章节，首先提出了基于 LNFG 和 Büchi 自动机的规范挖掘算法的整体框架，并介绍了各模块的功能和工作框架；然后，从伪代码和算法流程图两个方面描述了算法的具体实现过程；接着，简要介绍了本文基于此算法开发的规范挖掘工具 PPTLMiner+的安装与使用方法；最后通过具体示例列举了 PPTLMiner+的功能，包括基本功能和特色功能。

第四章：基于蚁群算法解决旅行商问题的规范挖掘

本章是本文的重点章节，挖掘了基于蚁群算法解决旅行商问题的规范，是本文提出的算法和工具的具体应用。本文首先介绍了旅行商问题和蚁群算法的相关背景，并在此基础上对基于蚁群算法解决旅行商问题的算法框架进行了简要概述；然后，使用 PPTLMiner+挖掘解决旅行商问题的程序规范，并展示了挖掘过程的中间结果；最后对挖掘结果进行分析。

第五章：对比实验与理论分析

本章是本文的重点章节，首先通过 PPTLMiner+与 PPTLMiner 和 Texada 的对比实验，展示本文的算法和工具的表达能力强和挖掘效率高的优点；然后对三个挖掘工具的挖掘算法进行了理论分析，包括时间复杂度公式的推导和空间复杂度的分析。

第六章：总结与展望

本章对本文的贡献进行了总结，并针对当前工作的不足进行了展望。

第二章　相关理论与技术研究

本章介绍与基于LNFG和Büchi自动机的规范挖掘相关的理论与技术研究成果，以便读者能更好地理解全文。本章首先介绍两种时序逻辑语言：命题投影时序逻辑和线性时序逻辑，包括语法、语义等知识点。接着，介绍命题投影时序逻辑对应的范式形式和带标记的范式图形式的定义。然后介绍线性时序逻辑对应的 Büchi 自动机形式。本章还介绍本文提出的规范挖掘算法中用到的一些技术及相关的发展历程，包括程序不变量挖掘技术、PPTL 转为 LNFG 技术和 LTL 转为 Büchi 自动机技术，在介绍程序不变量挖掘技术时还说明了事件（event）、轨迹（trace）、事件分隔符（event seperator）和轨迹分隔符（trace seperator）的概念。

2.1 相关理论研究

2.1.1　命题投影时序逻辑

命题投影时序逻辑 PPTL[17]是命题区间时序逻辑 PITL[16]的一个扩展。PPTL 将 PITL 的定义从有穷区间扩展到无穷区间，具有完全正则的表达能力[19]，可以描述区间相关和区间周期重复性质。下面介绍将阐述 PPTL 的语法和语义。

（1）　语法

根据原子命题的集合$Prop$和任意一个原子命题$p \epsilon Prop$，PPTL 公式P可以归纳定义如下：

$$P ::= p \mid \neg P \mid \bigcirc P \mid P_1 \vee P_2 \mid (P_1, \cdots, P_m)\ prj\ P \mid P^+ \qquad (2-1)$$

其中，P_1、P_2、…、P_m是定义良好的 PPTL 公式，$\bigcirc(next)$和$prj(projection)$是 PPTL 公式中含有的基本时序操作符。若 PPTL 公式中含有任一时序操作符，则该公式被称为时序公式，否则为状态公式。

（2）　语义

状态：若$B = \{true, false\}$，则状态（state）s定义为$s: Prop \to B$，原子命题p在状态s上的值用$s[p]$表示。

区间：定义一个不为空的、有穷的或无穷的状态序列为区间σ，区间σ的长度用$|\sigma|$表示。若区间是有穷的，则区间长度$|\sigma|$在数值上等于组成该区间的状态数减 1；若无穷，则$|\sigma| = \omega$，ω表示无穷的状态序列。为了统一标记有穷区间和无穷区间，将非负整数集N_0扩展到N_ω，且$N_\omega = N_0 \cup \omega$，$\omega = \omega$，则对于任意的$i \in N_0$，$i < \omega$。扩展$\leqslant$的定义为$\leq -(\omega, \omega)$，则区间$\sigma$形式化定义为$\sigma =< s_1, \cdots, s_{|\sigma|} >$，当且仅当$\sigma$为

无穷区间时，$s_{|\sigma|}$没有定义。$\sigma_{(i,\cdots,j)}$表示子区间$< s_i,\cdots,s_j >$，其中$0 \le i \leqslant j \le |\sigma|$。

若$\sigma_1 =< s_1,s_2,\cdots,s_k >$，$\sigma_2 =< s_j,s_{j+1},\cdots >$，则 $\sigma_1 \cdot \sigma_2$表示有穷区间σ_1和另一区间σ_2（或一个空串）的连接。$\sigma_1 \cdot \sigma_2$形式化定义如下：

$$\sigma_1 \cdot \sigma_2 =< s_1,\cdots,s_k,s_j,s_{j+1},\cdots > \qquad (2-2)$$

若$(r_1,\cdots,r_h)$表示一个整数序列，且$0 \le r_1 \le r_2 \le \cdots \le r_h \leqslant |\sigma|$，$h \ge 1$，则投影区间$\sigma \downarrow (r_1,\cdots,r_h)$定义为$\sigma$在$(r_1,\cdots,r_h)$上投影构成的一个新区间，此定义在投影结构中有着重要意义。$\sigma \downarrow (r_1,\cdots,r_h)$形式化定义如下：

$$\sigma \downarrow (r_1,\cdots,r_h) =< s_{t_1},s_{t_2},\cdots,s_{t_k} > \qquad (2-3)$$

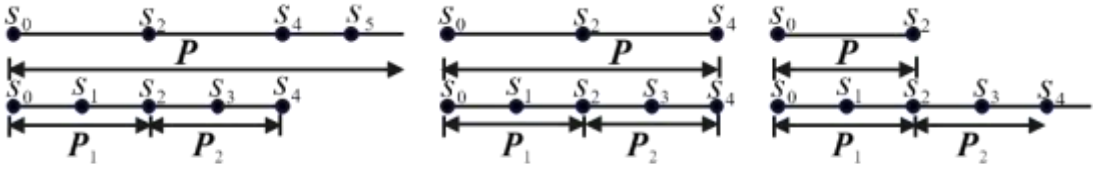
其中$(t_1,t_2,\cdots,t_k)$表示$(r_1,\cdots,r_h)$的最长严格递增序列。如：

$$< s_1,s_2,s_3,s_4,s_5 >\downarrow (1,1,2,3,3,3) =< s_1,s_2,s_3 >$$

解释：PPTL 公式是在区间上进行解释的。解释可以用一个三元组 I= (σ,k,j)来表示，其中k为整数，j为整数或ω，且满足$k \le j \le |\sigma|$。$(\sigma,k,j) \Vdash P$表示 PPTL 公式P在区间σ的子区间$< s_k,\cdots,s_j >$上解释，且s_k表示当前状态。可满足关系$\Vdash$可以递归定义如下：

$I \Vdash p$	**iff** $s_k[p] = true, p \in P$
$I \Vdash \neg P$	**iff** $I \nVdash P$
$I \Vdash P_1 \vee P_2$	**iff** $I \Vdash P_1$ 或$I \Vdash P_2$
$I \Vdash \bigcirc P$	**iff** $(\sigma,k+1,j) \Vdash P$
$I \Vdash (P_1,\cdots,P_m)\ prj\ P$	**iff** 若存在整数$k = r_0 \le r_1 \le \cdots \le r_m \le j$ 使$(\sigma,r_{l-1},r_l) \Vdash P_l$ 成立，且$1 \le l \le m$，则对于以下任一情况，$(\sigma_1,0,|\sigma_1|) \Vdash P$成立。 (1)$r_m < j$，且$\sigma_1 = \sigma \downarrow (r_0,\cdots,r_m) \cdot \sigma_{r_m+1,\cdots,j}$ (2)$r_m = j$，且$\sigma_1 = (r_0,\cdots,r_h)$，$0 \le h \le m$
$I \Vdash P^+$	**iff** 若存在整数$k = r_0 \le r_1 \le \cdots \le r_m = j$ 使$(\sigma,r_{l-1},r_l) \Vdash P_l$ 成立，且$1 \le l \le m$

为了更好地理解 PPTL 公式的特点，对投影操作符prj做进一步的解释。PPTL 公式$F \stackrel{\text{def}}{=} (P_1,P_2)prj\ P$的语义如图 2.1 所示。在图 2.1 中，公式$P_1$和$P$同时在$s_0$开始解释，在$P_1$解释结束时，$P_2$开始解释。由图 2.1 所示，$P$与$P_1;P_2$在时间上并行，$P$在$P_1$和$P_2$的解释区间所构成的区间连接（$P_1 \cdot P_2$）上解释。图 2.1 中（1）、（2）和（3）的不同点在于P的解释区间是否先于$P_1;P_2$的解释区间结束。由上可知，投影操作符prj适用于描述并发系统的性质。



(1) (2) (3)

图2.1 prj语义

操作符：操作符“$true, false, \wedge, \rightarrow, \leftrightarrow$”的定义与经典命题逻辑一致，如：$true \stackrel{\text{def}}{=} P \vee \neg P$，$false \stackrel{\text{def}}{=} P \wedge \neg P$，$p \rightarrow q \stackrel{\text{def}}{=} \neg p \vee q$，$p \leftrightarrow q \stackrel{\text{def}}{=} (p \rightarrow q) \wedge (q \rightarrow p)$。此外，PPTL 公式还有一些常用的导出公式，导出公式及对应名称和在程序中的编码符号如表 2.1 所示，其中$n > 1$。

表2.1 PPTL 常见导出公式

序号	名称	导出公式	编码符号
1	EMPTY	$\varepsilon \stackrel{\text{def}}{=} \neg \bigcirc true$	empty
2	MORE	$more \stackrel{\text{def}}{=} \neg empty$	more
3	0-LEN	$len(0) \stackrel{\text{def}}{=} empty$	len(0)
4	N-LEN	$len(n) \stackrel{\text{def}}{=} \bigcirc^n empty$	len(n)
5	SKIP	$skip \stackrel{\text{def}}{=} len(1)$	skip
6	0-NEXT	$\bigcirc^0 P \stackrel{\text{def}}{=} P$	p
7	N-NEXT	$\bigcirc^n P \stackrel{\text{def}}{=} \bigcirc(\bigcirc^{n-1} P)$	$\underbrace{() \cdots ()}_{n}$
8	W-NEXT	$\odot P \stackrel{\text{def}}{=} empty \vee \bigcirc P$	(.)
9	CHOP	$P_1; P_2 \stackrel{\text{def}}{=} (P_1, P_2) prj\ empty$	;
10	SOMETIMES	$\Diamond P \stackrel{\text{def}}{=} true; P$	<>
11	ALWAYS	$\Box P \stackrel{\text{def}}{=} \neg \Diamond \neg P$	[]
12	STAR	$P^* \stackrel{\text{def}}{=} \varepsilon \vee P^+$	*
13	KEEP	$keep(P) \stackrel{\text{def}}{=} \Box (\neg empty \rightarrow P)$	keep(P)
14	HALT	$halt(P) \stackrel{\text{def}}{=} \Box (empty \leftrightarrow P)$	halt(P)
15	FIN	$fin(P) \stackrel{\text{def}}{=} \Box (empty \rightarrow P)$	fin(P)

其中，$empty$代表当前状态为区间的最后一个状态，即区间长度为 0。$more$表示当前状态后至少存在一个状态，此时区间长度至少为 1。$skip$表示跳过长度为 1 的区间。$len(n)$表示长度为n的区间。$keep(P)$表示当且仅当P在该区间除最后一个状态的其

他状态都成立时，P某个区间成立。$halt(P)$表示P在某个区间成立当且仅当是P在这个区间的最后一个状态成立。$fin(P)$表示若P在该区间的最后一个状态成立，则P在某个区间成立。

优先级：表 2.2 展示了 PPTL 操作符的优先级规则，优先级从上到下递减。

表2.2 PPTL 操作符优先级

优先级别	操作符
1	¬
2	+ *
3	◯ ⊙ ◊ □
4	∧
5	;
6	∨
7	prj
8	→ ↔

2.1.2 线性时序逻辑

在计算机科学领域，通常使用线性时序逻辑 LTL[12]描述系统性质。LTL 是一个$\omega-$正则语言，并不是完全正则的。PPTL 是一个完全正则语言，即可以描述有穷状态也可以描述无穷状态。本文的挖掘算法支持输入 LTL 公式和 PPTL 公式，为了更直观的了解 LTL 和 PPTL 的区别，本节介绍 LTL 的语法和语义。

（1） 语法

若AP表示原子命题集，$\eta \in AP$表示任意的一个原子命题，则 LTL 公式φ可以形式化定义如下：

$$\varphi ::= \eta \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid X\varphi \mid F\varphi \mid G\varphi \mid \varphi U\psi \mid \varphi W\psi \qquad (2-4)$$

其中ψ是一个定义良好的 LTL 公式，¬和∨是布尔操作符，X、F、G、U和W是时序操作符。LTL 和 PPTL 含有的相同操作符如表 2.3，LTL 公式和 PPTL 公式的关系如图 2.2 所示。由图 2.2 可知，PPTL 公式包含 LTL 公式，这里的“包含”不仅指 PPTL 公式含有操作符集包含 LTL 公式的操作符集，还指 PPTL 公式能重写 LTL 能描述的性质。

（2） 语义

LTL 公式解释在一个无穷状态序列ξ上，且$\xi = x_0x_1x_2\cdots$。若$\xi^k = x_kx_{k+1}x_{k+2}\cdots$

表示一个从第k项开始的状态序列，且当前状态为x_k，则LTL公式的语义可形式化定义如下：

$\xi^k \Vdash \eta$ **iff** $x_k \Vdash \eta$，且$\eta \in AP$

$\xi^k \Vdash \neg\varphi$ **iff** $\xi^k \nVdash \varphi$

$\xi^k \Vdash \varphi \wedge \psi$ **iff** $\xi^k \Vdash \varphi$且$\xi^k \Vdash \psi$

$\xi^k \Vdash \varphi \vee \psi$ **iff** $\xi^k \Vdash \varphi$或$\xi^k \Vdash \psi$

$\xi^k \Vdash X\varphi$ **iff** $\xi^{k+1} \Vdash \varphi$

$\xi^k \Vdash F\varphi$ **iff** 若存在$i \geq k$，$\xi^i \Vdash \varphi$

$\xi^k \Vdash G\varphi$ **iff** 若对于任意的$i \geq k$，$\xi^i \Vdash \varphi$

$\xi^k \Vdash \varphi U\psi$ **iff** 若存在$i \geq k$，使得对于任意的j，$k \leq j \leq i-1$，$\xi^j \Vdash \varphi$成立，并且$\xi^i \Vdash \psi$成立。

$\xi^k \Vdash \varphi W\psi$ **iff** 若至少以下两种情况之一成立：

(1)对于每个$i \geq k$，$\xi^i \Vdash \psi$；

(2)存在$j \geq k$，$\xi^i \Vdash \varphi$，且对于任意的$k \leq i \leq$ j，$\xi^i \Vdash \psi$。

表2.3 LTL 和 PPTL 相同操作符

序号	名称	LTL 操作符	PPTL 操作符
1	NOT	¬	¬
2	AND	∧	∧
3	OR	∨	∨
4	NEXT	X	◯
5	SOMETIMES	F	◊
6	ALWAYS	G	□
7	UNTIL	U	U
8	UNLESS	W	W

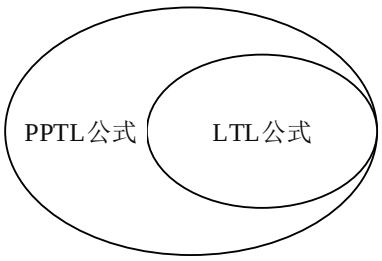


图2.2 PPTL 公式和 LTL 公式关系图

2.1.3 范式

为了检验 PPTL 公式的可满足性，提出了一种用于决策过程的范式（Normal Formal，NF）形式[9]。任何一个 PPTL 公式都可以重写成范式形式，PPTL 公式和重写的范式的等价性已经被证明，重写过程通过 PPTL2NF 工具实现[20]。本节介绍 PPTL 范式的定义。

若P是一个定义良好的 PPTL 公式，P_p是P中出现的原子命题的集合，则P是范式形式当且仅当P可以重写为如下形式：

$$P ::= \bigvee_{j=1}^{n_0}\left(P_{ej} \wedge \varepsilon\right) \vee \bigvee_{i=1}^{n}(P_{ci} \wedge \bigcirc P_i') \tag{2-5}$$

其中，$P_{ej} \equiv \bigwedge_{k=1}^{m_0} q_{jk}^{\cdot}$，$P_{ci} \equiv \bigwedge_{h=1}^{m} q_{ih}^{\cdot}$，$l = |P_p|$，$1 \le n \le 3^l$，$1 \le n_0 \le 3^l$，$1 \le m \le l$，$1 \le m_0 \le l$，$q_{jk}, q_{ih} \in P_p$，且对于任意的$r \in P_p$，$\dot{r}$表示$r$或$\neg r$。为了方便书写，用$P_e \wedge \varepsilon$代替$\bigvee_{j=1}^{n_0}\left(P_{ej} \wedge \varepsilon\right)$，用$\bigvee_{i=1}^{r}(P_i \wedge \bigcirc P_i')$代替$\bigvee_{i=1}^{n}(P_{ci} \wedge \bigcirc P_i')$，则$P$的范式可简写为如下形式：

$$P ::= P_e \wedge \varepsilon \vee \bigvee_{i=1}^{r}(P_i \wedge \bigcirc P_i') \tag{2-6}$$

其中，P_e和P_i是状态公式，$P_i^{'}$ 表示的 PPTL 公式是定义良好的。即P是 PPTL 公式的范式形式时，P可以被重写为当前状态为P_e且当前状态为最后一个状态（ε），或当前状态为P_i且下一状态为P_i'（$\bigcirc P_i'$）。

2.1.4 带标记的范式图

任何一个 PPTL 公式都可以转换为等价的带标记的范式图 LNFG [20]形式，本文提出了基于 LNFG 的规范挖掘算法，下面介绍 LNFG 的定义。

若P是一个定义良好的 PPTL 公式，则P的 LNFG 形式可以用一个带权值的有向图G来描述，且$G = (CL(P), EL(P), V_0, \mathbb{L} = \mathbb{L}_1, \cdots, \mathbb{L}_m)$。其中，$CL(P)$表示结点集合；$EL(P)$表示边集；$V_0 \subset CL(P)$表示根结点（开始结点）的集合；$\mathbb{L}_k \subseteq CL(P)$表示带有$l_k^{\sim}$标记的结点集合，其中$1 \le k \le m$，$l_k^{\sim}$标记定义参考文献[19,20]。在$CL(P)$中，每个结点用一个时序公式标识。在$EL(P)$中，每条边用一个状态公式$Q_t$标识，且都是从结点$Q_S$到结点$Q_e$的有向边，这条边可以用$Q_s \xrightarrow{Q_t} Q_e$表示，$Q_t$为结点$Q_S$到结点$Q_e$的迁移条件。

2.1.5 Büchi 自动机

计算机科学领域常用 Büchi 自动机[38]来描述系统模型。Büchi 自动机接收ω－正则语言，即接收无穷字。转换工具可以将 LTL 转换为 Büchi 自动机，在描述系统性

质时显得尤为重要。本文提出了基于 Büchi 自动机的规范挖掘算法，下面介绍 Büchi 自动机的定义。

Büchi 自动机由一个五元组B构成，且$B = (Q, \Sigma, q_0, F, \delta)$。其中，$Q$表示一组状态的集合；$\Sigma$表示输入字母表；$q_0$表示初始状态；$F$表示终止（接收）状态的集合；$\delta$表示转换函数$Q \times \Sigma \to 2^Q$。若输入一个无限字$word = A_0A_1A_2\cdots$，自动机的运行序列可以表示为$\pi = q_0q_1q_2\cdots$，若无限字$word$可以使 Büchi 自动机无限经常次访问终止状态，则说明π可以被 Büchi 自动机接收。

2.2 相关技术研究

2.2.1 程序不变量挖掘技术

在计算机科学领域，程序不变量（software invariant）是系统运行时在程序中必须保持不变的属性[39]（property），能够体现系统行为，因此广泛用于程序验证、程序维护、程序缺陷定位及维护等领域。根据在程序中位置不同，不变量可以分为三类：循环不变量、结构不变量和类不变量。循环不变量表示在循环初始和结束位置都保持不变的属性；结构不变量表示描述数据结构实例之间的关系；类不变量表示描述对象方法或域的属性。本节介绍不变量挖掘技术及相关工具。

程序不变量分析技术主要分为静态分析和动态分析。静态分析无需运行程序，而是直接分析程序源码或可执行文件来获得不变量信息。然而，由于静态分析非常复杂，因此在实际应用中，它只适用于小规模的软件，并且所获得的不变量信息比较简单。动态分析则通过运行程序，根据程序的执行轨迹来分析得到不变量信息。

本文采用动态分析挖掘程序不变量信息。当前基于动态分析来挖掘不变量的技术分为四类[39]：基于模板穷举的方法、基于数值计算的方法、基于机器学习的方法和基于符号执行的方法。基于模板穷举的挖掘方法简单直观，且具有成熟的工具体系支持，应用最广。但是，基于模板穷举的挖掘方法高度依赖模板，且不变量信息过于冗杂，因此需要根据研究目的开发相应的过滤工具。基于模板穷举的挖掘方法实现的工具有 Daikon[37]、DIDUCE[40]和 IODINE[6]等。基于数值计算的方法表现力强、不变量种类多，但精确度和实用性较低。工具 DIG[41]、NumINV[42]均基于数值计算的方法实现。基于机器学习的方法优点是形式较为简单，能够快速地产生不变量信息；缺点是表现力有限、部分结果具有不确定性。ICE[43]就是基于此方法实现的一个工具。基于符号执行的方法即依赖于符号执行来挖掘不变量，具有路径覆盖率高、精确度高的优点，但是容易产生路径爆炸，不适用于挖掘复杂程序的不变量信息。基于符号执行挖掘不变量的代表工具有 DySy[44]、KRYSTAL[45]等。

综合上述四类技术的特点及现有工具，本文采用基于模板穷举的技术挖掘程序

不变量信息，使用经典不变量挖掘工具 Daikon。Daikon 作为在程序不变量挖掘领域里程碑式的工具，应用广且直至目前仍在更新改进，Daikon-5.8.16 是 2022 年 11 月 9 日更新的 Daikon 最新版本，官方提供了《用户使用文档》、《开发者文档》和开源代码。Daikon 支持输入 Java、C、C++、C#、Perl 和 Visual Basic 语言编写的程序，并通过执行程序输出一个不变量文件。

Daikon 工作框架图如图 2.3 所示，首先编配器对源程序进行编配，获得编配后的程序和轨迹文件格式定义文件。然后编写测试用例，并与编配后的程序一同输入到编译器，从而获得轨迹文件。最后将轨迹文件和轨迹文件格式定义文件输入检测器进行程序不变量信息的挖掘。Daikon 最终获得的不变量文件中含有很多伪不变量和大量用户不感兴趣的文件，主要包括方法、全局/局部变量、形参和返回值等信息。因此，需要根据用户需求过滤不变量，最终获得感兴趣的不变量信息。

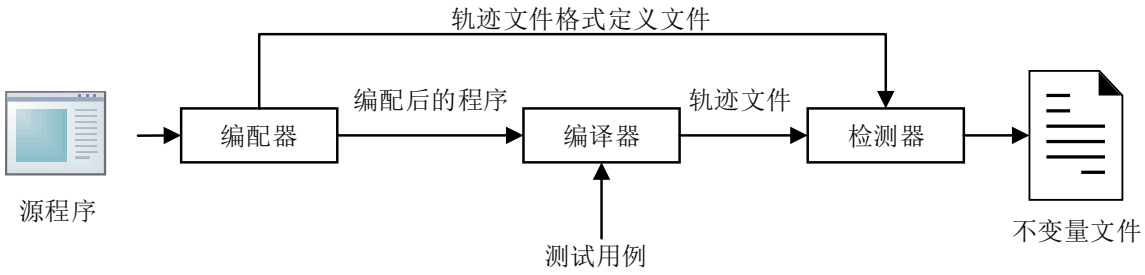


图2.3 Daikon 工作框架图

本文挖掘程序在执行过程中方法（函数）之间的时序关系，因此还需要将 Daikon 生成的不变量信息进行过滤，最终仅保留方法信息，过滤工具在后续章节介绍。下面给出事件（event）、轨迹（trace）、事件分隔符（event seperator）和轨迹分隔符（trace seperator）的概念。

概念 1. (event、trace) 将程序执行中的方法（method）定义为事件，过滤工具生成的方法序列定义为轨迹（trace），trace 可以形式化定义为$\tau = (e_1, e_2, \cdots, e_n)$，其中$n$为正整数，$e_i$表示事件，$e_j$与$e_k$可以是相同的事件，且在程序执行过程中从$e_1$到$e_n$在时间上有序，$1 \le j, k \le n$。即 trace 是 event 的一条有序序列。

概念 2. (event seperator、trace seperator) 若多个事件在同一时刻发生，则用事件分隔符分割这些事件，这种情况多发生在并行系统中。多次执行程序可以生成多条 trace，将多条 trace 共同存储在一个 trace 文件中，相邻 trace 之间用 trace 分隔符分隔。

2.2.2 PPTL 转 LNFG 技术

本文实现 PPTL 转为 LNFG 的工作是由 PPTL2LNFG 工具[20]完成的，工作框架如图 2.4 所示。PPTL2LNFG 整个工作流程由分析模块、重写 NF 模块和构造 LNFG 模

块组成。分析模块由词法解析器 Flex 和语法解析器 Bison 构成，Flex 将字符串存储的 PPTL 公式解析成词法流，Bison 将词法流解析成语法树。重写范式模块通过预处理、重写范式和简化过程将语法树重写为 PPTL 的范式形式。构造 LNFG 模块通过初始化、构造 LNFG 和简化过程将 PPTL 范式形式转为 LNFG 形式。PPTL2LNFG 工具的难点在于 PPTL 导出公式中 CHOP 公式的处理，该工具将 CHOP 公式$P;Q$重写成$P \wedge fin(l_i);Q$形式，再进一步处理 FIN 算子和l_i标记。PPTL2LNFG 具体实现算法详见参考文献[20]。

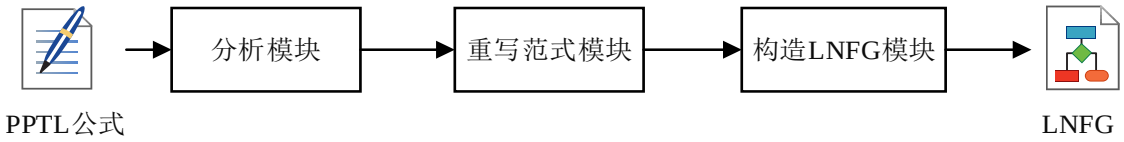


图2.4 PPTL2LNFG 工作框架图

2.2.3 LTL 转 Büchi 自动机技术

自 1983 年 Wolper 和 Vardi 等人[46]首次提出将 LTL 公式转换为 Büchi 自动机以来，计算机领域一直在研究这种转换技术，尤其是在模型检测领域。目前主流的转换算法有两种，分别是 on-the-fly 直接转换算法和基于极弱交替式 co-Büchi 自动机（Very Weak Alternating co-Büchi Automata，VWAA）的转换算法。本节简单介绍这两种算法的实现步骤以及对应的工具。

Gerth 和 Peled 等人提出的 on-the-fly 直接转换算法是一种基于状态搜索的算法[47]，算法流程如图 2.5 所示。第一步，对 LTL 公式进行语法分析，将其转为语法树形式。第二步，将语法树转换为否定范式（Negation Normal Form，NNF）形式，即把否定符号移到命题变量前面。第三步，设定 NNF 形式的 LTL 公式为初始状态，并将其加入到状态队列中。第四步，从状态队列中取出当前状态，并根据其类型生成后继状态。根据 LTL 公式的结构，生成的后继状态可能是命题变量、逻辑连接符或未来的状态。第五步，将新生成的状态与已有状态进行比较，若等价，则将新生成的状态丢弃，否则将其加入到状态队列中。第六步，在满足 LTL 公式的情况下，将当前状态转移为后继状态。如果后继状态满足 LTL 公式，将其加入到状态队列中。重复第四步到第六步，直到找到一个违反 LTL 公式的状态或者遍历完所有状态。然后根据情况返回结果，如违反 LTL 公式的状态，或者证明 LTL 公式成立的状态序列等。最后将得到的状态图转换为 Büchi 自动机，包括状态合并和确定化等步骤。on-the-fly 直接转换算法的时间和空间复杂度都比较高，特别是在处理复杂的 LTL 公式时，算法的效率会受到很大的影响。SPIN[48]就是基于此算法实现的一个著名模型检测工具。

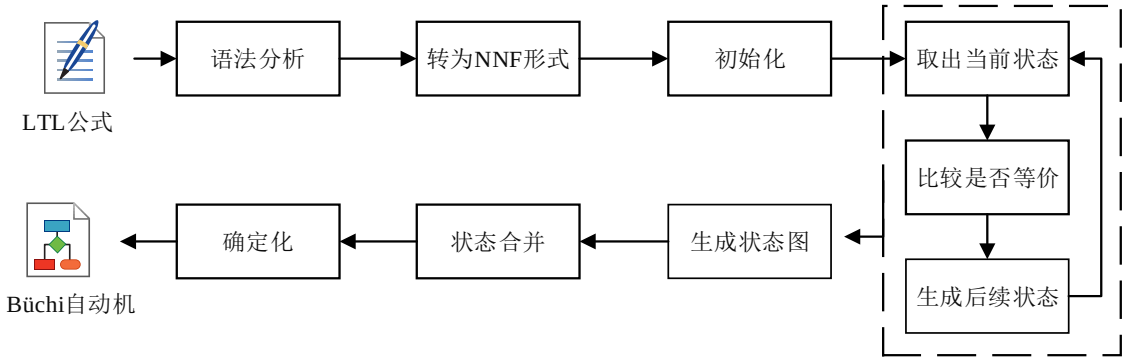


图2.5 on-the-fly 直接转换算法流程图

为了弥补上述算法在实际应用中的缺陷，Paul 和 Denis 提出了基于 VMAA 的转换算法[49]。这种算法的提出旨在通过提高自动机的转换效率，改进 LTL 公式的验证过程。除此之外，基于 VMAA 的转换算法通过数学证明，可以保证算法的正确性和精度，可以对 LTL 公式进行准确的自动化验证。该算法不仅具有高效性和准确性，而且支持多种 LTL 公式，可应用于多种领域，并且能够生成可读性强的 Büchi 自动机，具有良好的可用性和扩展性。LTL2BA 工具[49]实现了基于 VMAA 的转换算法，而 LTL3BA 工具[50]基于 alternating 公式对 LTL2BA 进行了改进。

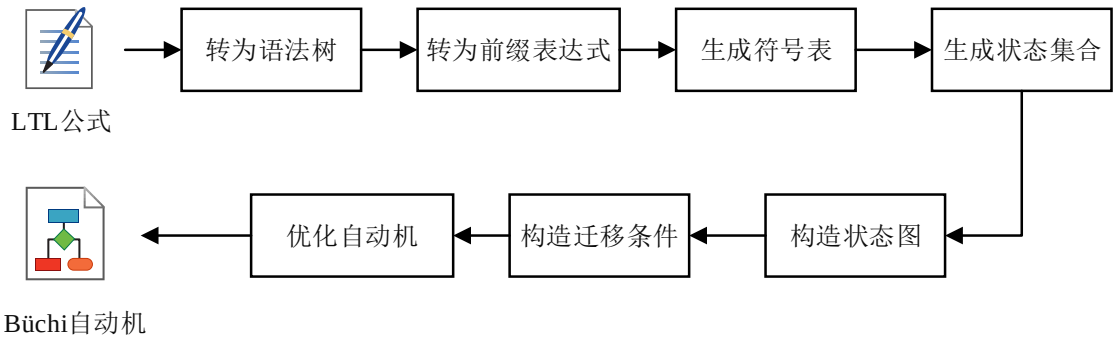


图2.6 基于 VMAA 的转换算法流程图

不同于以上两种算法，Wolper 提出了 Tableau 算法[51]，算法流程图如图 2.6 所示。首先将 LTL 公式转换为语法树，然后将语法树转换为简化的前缀表达式，以便于后续处理。其次，对前缀表达式进行展开，生成一个符号表。符号表中的每个符号都代表一个 LTL 公式的子表达式。展开过程中，需要根据 LTL 公式的不同性质，采用不同的展开规则。接着，生成每个符号对应的状态集合。状态集合表示 LTL 公式在某个时间点或某个状态下的取值情况。另外，需要生成每个状态集合对应的自动机状态，即表示 LTL 公式满足或不满足的状态。然后，将所有自动机状态连接起来，形成自动机的状态图。在状态图中，每个节点表示一个 Büchi 自动机状态，每条边表示一个状态之间的转移关系。最后，将生成的 Büchi 自动机进行优化，去除不必

要的状态和转移条件，以提高自动机的效率和可读性。著名的模型检测工具 SPOT[52] 就是基于此算法实现的，本文提到的挖掘工具 Texada 就使用了 SPOT 处理 LTL 公式。

本文提出的基于 Büchi 自动机的规范挖掘算法使用的是 LTL3BA 工具，原因是 LTL3BA 开源、功能丰富，且使用方便，无需安装额外的动态链接库。LTL3BA 工作过程如图 2.7 所示，LTL3BA 首先将 LTL 公式进行化简，如去除冗余的逻辑运算符等，然后转换为 VWAA。接着，将 VWAA 转换为基于变迁的 Büchi 自动机（Transitions Bases Generalized Büchi Automata，TGBA），TGBA 是一个能接受无限长度字符串的自动机，包括状态集合、初始状态、转移关系和接受状态等。最后，优化 TGBA 从而获得 Büchi 自动机，优化通常包括合并等价状态、去除不可达状态和最小化状态数等。

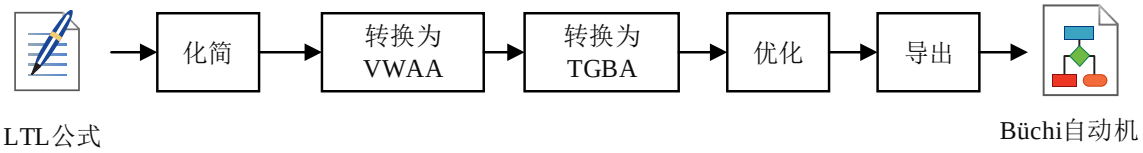


图2.7 LTL3BA 转换过程

2.3 本章小结

本章介绍了与本文挖掘算法相关的理论和技术研究成果。首先介绍了 PPTL 和 LTL 的语法、语义等知识点，同时分析了 PPTL 公式和 LTL 公式的关系。接着，阐述了 PPTL 对应的 NF、LNFG 的定义，以及 LTL 对应的 Büchi 自动机的定义。然后简单介绍了程序不变量挖掘技术、PPTL 转 LNFG 技术和 LTL 转 Büchi 自动机技术的研究现状，最后确定了本文使用的工具，即动态不变量挖掘工具 Daikon、PPTL 转 LNFG 工具 PPTL2LNFG 和 LTL 转 Büchi 自动机工具 LTL3BA。

第三章　规范挖掘算法的设计与实现

针对现有规范挖掘算法使用的时序逻辑语言的表达能力有限，以及算法的挖掘时间过长和内存崩溃等问题，本文提出基于 LNFG 和 Büchi 自动机的规范挖掘算法，并开发出对应的规范挖掘工具 PPTLMiner+。本章首先介绍挖掘算法的整体框架，并对每一模块的工作框架和功能进行简单介绍。接着，通过伪代码和算法流程图的形式介绍规范挖掘算法的具体实现，以便更好地理解挖掘算法的设计思想。然后简单介绍挖掘工具 PPTLMiner+的安装与使用方法。最后，通过列举几个具体实例，介绍本文开发的挖掘工具的基本功能和特色功能，以便更好地理解 PPTLMiner+的功能和使用方法。同时，为了规范本文用词，本章在介绍算法框架时，给出了性质公式（Property Formula）、映射（Mapping）和实例化（Instancing）的概念。

3.1 整体框架

本文提出基于 LNFG 和 Büchi 自动机规范挖掘算法，整体框架如图 3.1。以可执行程序的源代码和待挖掘的性质公式为输入，输出一组满足程序的实例化性质公式，即程序的规范。该框架由轨迹生成模块（Trace Generator）、轨迹解析模块（Trace Parser）、公式解析模块（Formula Parser）、公式转换模块（Formula Translator）、公式实例化模块（Formula Instantiator）和公式验证模块（Formula Checker）六部分构成，其中轨迹生成模块和轨迹解析模块在文献[10]中已经实现。

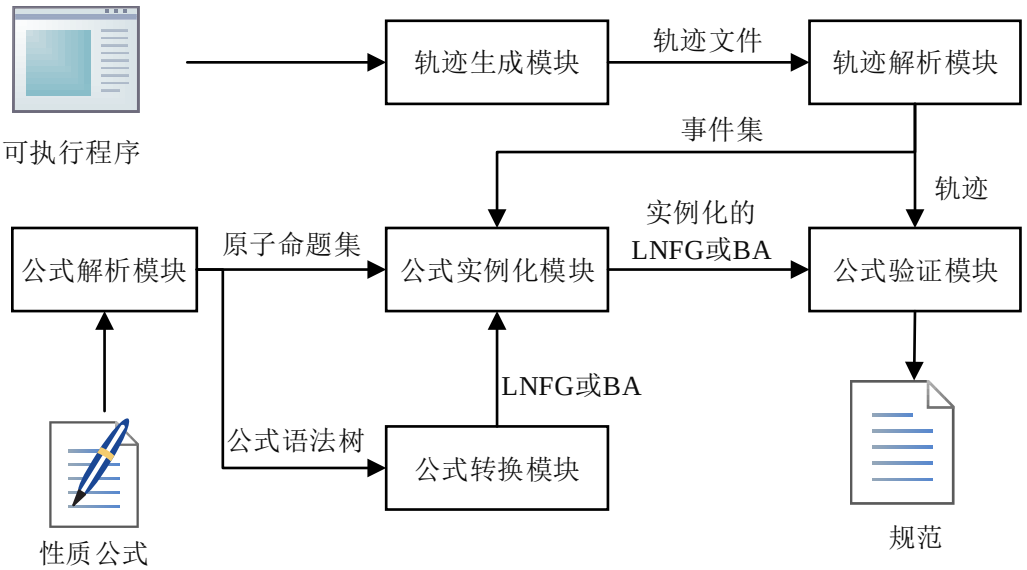


图3.1 规范挖掘算法整体框架图

Trace Generator：轨迹生成模块通过编译执行程序的源代码生成输入规范挖掘的 trace 文件，工作框架如图 3.2 所示。轨迹生成模块的输入是用 Java、C、C++、Perl 等语言编写的待验证系统的可执行程序源代码，输出是一条能表示程序执行过程中方法调用顺序的轨迹（trace）。Trace 生成模块通过调用 Daikon 和过滤 DtraceFilter 工具实现。将可执行程序源代码输入 Daikon，输出一个包含程序运行时不变量信息的文件*.dtrace。由于*.dtrace 文件包含的不变量信息过于冗杂，而本文主要研究程序中方法的规范，故需根据 dtrace 文件的特点，设计开发相应的不变量过滤工具 DtraceFilter。

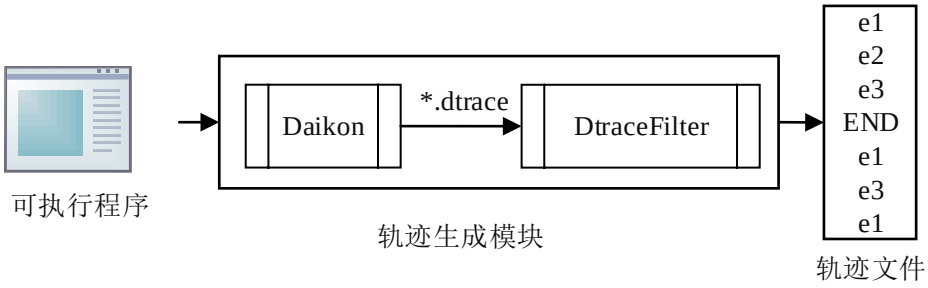


图3.2 Trace 生成模块工作框架图

Trace Parser：轨迹解析模块通过解析 trace 文件为后续输入公式实例化模块和公式验证模块做准备，工作框架如图 3.3 所示。将轨迹生成器生成的 trace 文件输入轨迹解析模块，通过顺序遍历文件获得 trace 中出现的事件集合（$EventSet$），并用合适的数据结构存储 trace τ。

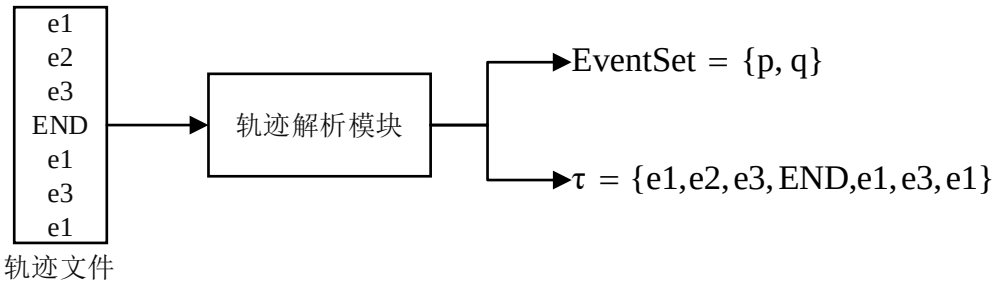


图3.3 Trace 解析模块工作框架图

Formula Parser：公式解析模块通过解析性质公式为实现实例化公式和转换性质公式的形式做准备，工作框架如图 3.4，输入用字符串存储的待挖掘的性质公式，输出用语法树形式表示的公式，并通过先序遍历语法树获得公式中出现的原子命题集合$AtomicSet$。公式解析模块通过词法分析器 Flex 和语法分析器 Bison 实现，用户输入的性质公式为字符串形式，Flex 对字符串进行词法分析得到对应的词法流，

Bison 对词法流进行语法分析从而生成性质公式的语法树形式。下面给出本文提到的性质公式的概念。

概念 3. (Property Formula) 待验证的系统性质，支持用户自定义输入，本文的性质公式可以用 PPTL 公式表示，也可以用 LTL 公式表示。

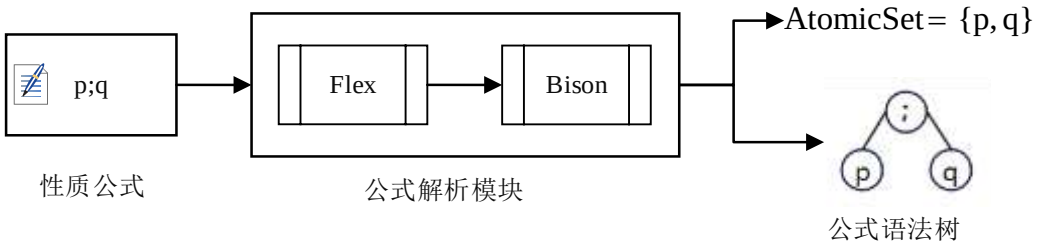


图3.4 公式解析模块工作框架图

Formula Translator：公式转换模块实现转换性质公式形式的功能，图 3.5 给出了公式转换器的工作框架图。公式转换模块的输入是待验证的性质公式的语法树，输出是使用邻接链表存储的 LNFG 或 Büchi 自动机，在图论领域，LNFG 和 Büchi 自动机都属于带权值的有向图。若用户输入用 PPTL 公式表示的性质公式，则选择使用基于 LNFG 的规范挖掘算法，并调用 PPTL2LNFG 工具将 PPTL 公式转换为 LNFG 形式。若性质公式是 LTL 公式，且用户选择使用基于 Büchi 自动机的规范挖掘算法，则调用 LTL3BA 工具将 LTL 公式转换为 Büchi 自动机形式。

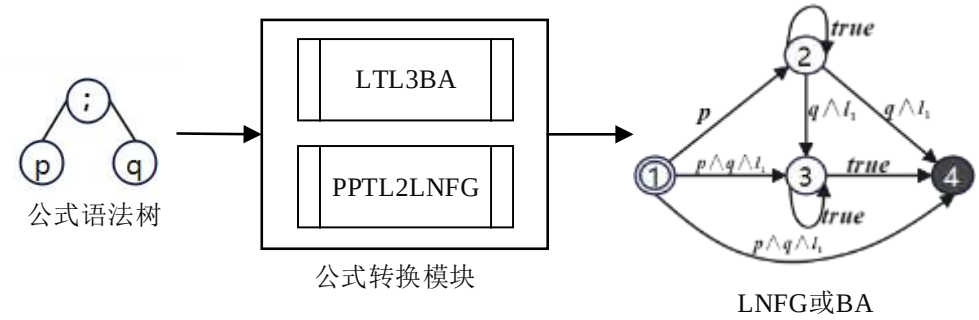


图3.5 公式转换模块工作框架图

Formula Instantiator：公式实例化模块用 trace 中的事件替换 LNFG 或 Büchi 自动机中出现的原子命题来实现实例化公式功能，工作框架如图 3.6。公式实例化模块包含三个输入：（1）公式解析模块生成的原子命题集$AtomicSet$；（2）轨迹解析模块生成的事件集$EventSet$；（3）公式转换模块生成的用邻接表存储的 LNFG 或 Büchi 自动机，输出是实例化的 LNFG 或 Büchi 自动机。下面给出映射和实例化的概念。

概念 4.(Mapping、Instancing) 将原子命题和事件形成的对应关系称作映射

（Mapping），形如（atomic，event）。用映射中的事件替换 LNFG 或 Büchi 自动机中对应的原子命题的过程叫做实例化（Instancing）。实例化过程将规范挖掘的两个输入联系起来，这样性质公式中的原子命题就有了实际含义。

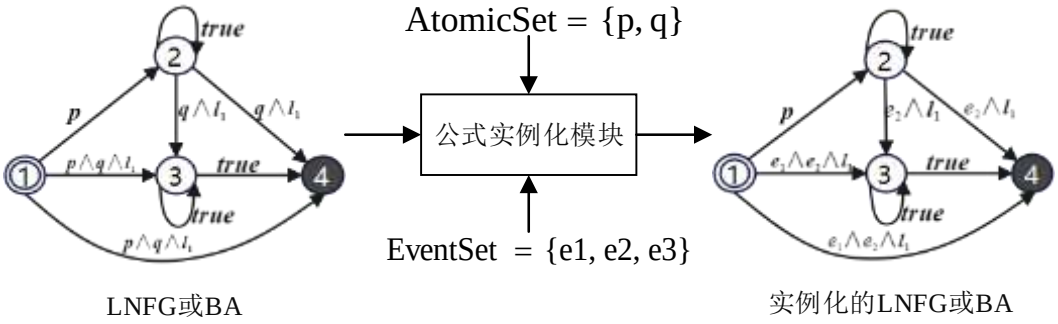


图3.6 公式实例化模块工作框架图

Formula Checker：通过调用检测算法来实现规范挖掘功能，工作框架图如图 3.7 所示。公式验证模块的输入有两个：（1）轨迹解析模块生成的用合适的数据结构存储的 trace τ和（2）公式实例化模块生成的实例化 LNFG 或 Büchi 自动机，输出是满足τ的实例化性质公式，即规范。

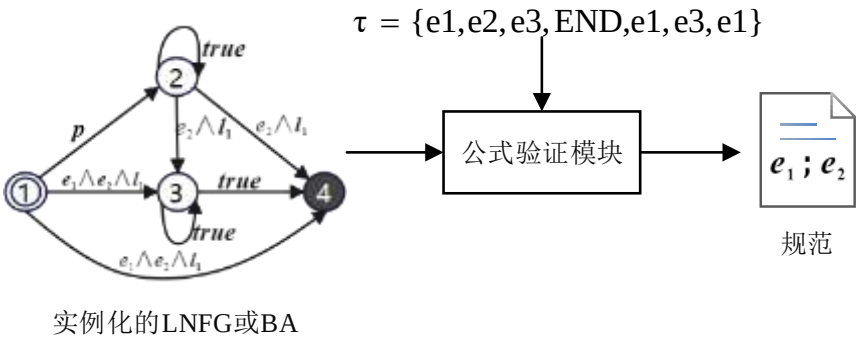


图3.7 公式验证模块工作框架图

3.2 具体实现

上一节对基于 LNFG 和 Büchi 自动机的规范挖掘算法的整体框架以及各模块工作框架进行了描述，本节基于上述算法开发对应的挖掘工具 PPTLMiner+。本节将从算法流程图和关键算法伪代码两方面介绍各模块的具体实现，以便读者能够更好地理解本文挖掘算法的设计思想。

3.2.1 生成 trace 文件

轨迹生成模块实现生成 trace 文件的功能，为后续的挖掘过程做准备。轨迹生成

模块调用 Daikon 生成不变量文件*.dtrace，本文开发不变量过滤工具 DtraceFilter 来过滤 dtrace 文件，实现生成程序运行时函数的执行轨迹，算法流程图如图 3.8 所示。

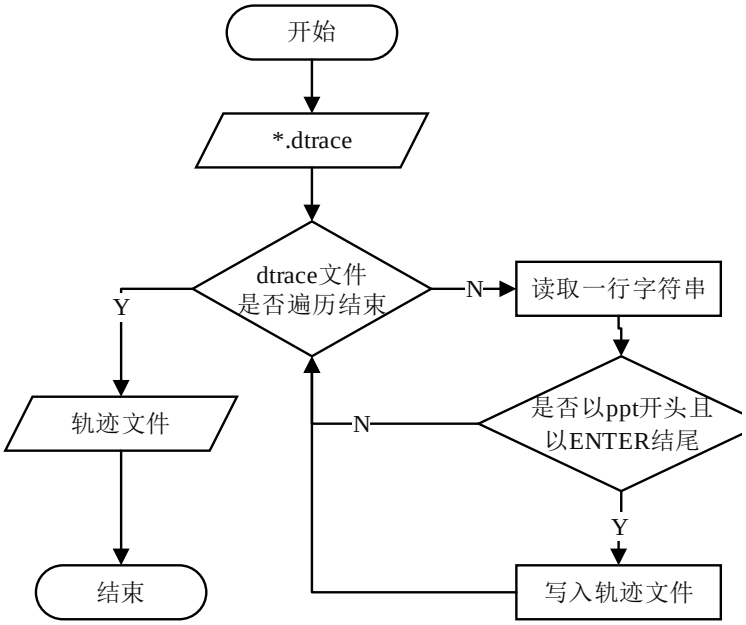


图3.8 生成 trace 文件算法流程图

Algorithm 1 生成trace文件

Input: File *program*

Output: File *traceFile*

```
1: dtraceFile←Daikon(program)
2: file←fopen(program)
3: while !fopen(file) do
4:     lineText←fscanf(file)
5:     a← lineText.startsWith("ppt")
6:     b← lineText.endsWith("ENTER")
7:     if a & b then
8:         functionENTER←lineText.deleteStr("ppt") \\字符串以ppt开始
9:         functionName← functionENTER.deleteStr("ENTER") \\字符串以ENTER结束
10:        event← functionName
11:        traceFile.write(event)
12:    end if
13: end while
```

轨迹生成模块的关键算法如算法 1 所示，首先调用 Daikon 工具生成$dtraceFile$，然后按行遍历$dtraceFile$的内容$lineText$。通过阅读 Daikon 的《使用者文档》可知，

不变量文件中函数的入口信息使用“ppt+函数名+ENTER”的形式注明。故算法第5-11行先判断$lineText$是否以“ppt”开始且以“ENTER”结尾，若满足判断条件，则依次删除$lineText$中的“ppt”和“ENTER”两个字符串，从而得到函数名$functionName$，然后将$functionName$作为事件存入$traceFile$中。

3.2.2 解析 trace 文件

轨迹解析模块完成对 trace 文件的解析功能，关键算法如算法 2 所示，算法流程图如图 3.9 所示。算法第 1、2 行首先对$EventSet$和τ进行初始化，然后打开文件$tracefile$，按行读取直至文件尾 EOF。令每行读取的字符串为$lineText$，若算法第 6、7 行判断$lineText$是 trace 分隔符$traceSeperator$时，将$traceSeperator$加入到τ；反之算法第 8、9 行将$lineText$定义为一个$event$，并将$event$添加到τ。同时判断$EventSet$中是否包含$event$，若不包含则向$EventSet$中添加$event$。

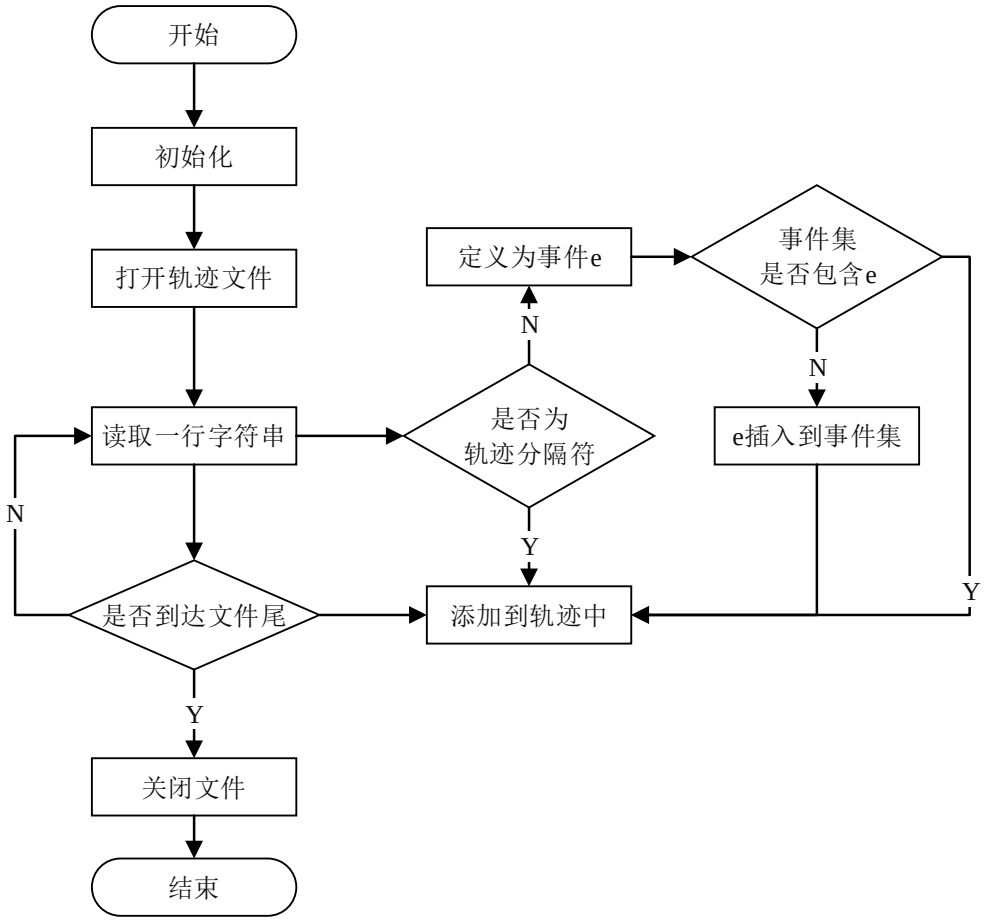


图3.9 解析 trace 算法流程图

Algorithm 2 解析trace文件

Input: File *traceFile*

Output: set<event>*EventSet* vector<event> τ

```
1: EventSet ← null
2: τ ← null
3: file ← fopen(tracefile)
4: while !feof(file) do
5:     lineText←fscanf(file)
6:     if lineText==traceSeperator then \\lineText是trace分隔符
7:         τ.add(traceSeperator)
8:     else\\ lineText是事件
9:         event← lineText
10:        τ.add(event)
11:        if event∉EventSet then
12:            EventSet.add(event)
13:        end if
14:    end if
15: end while
```

3.2.3 解析性质公式

公式解析模块完成对性质公式的解析功能，算法流程图如图 3.10 所示。首先定义性质公式对应语法树的结点类型，定义了性质公式中所有的操作符，然后定义了语法树的结构，即语法树包括结点、节点名和两个孩子节点，相关定义如下所示。

```
1.  enum NodeType{/*定义结点类型*/
2.        IDENT_EXP,        OR_STA,       AND_STA,          NEGATION_STA,
3.        CHOP_STA,         NEXT_STA,     WNEXT_STA,        ALWAYS_STA,
4.        SOMETIME_STA,     UNTIL_STA,    PROJECTION_STA,   FIN_EXP,
5.        TRUE_EXP,         FALSE_EXP,    EMPTY_EXP,        MORE_EXP,
6.        SKIP_EXP,         IDENT_EXP,    CHOPSTAR_STA,     CHOPPLUS_STA
7.  }
8.  typedef struct syntaxTree{/*定义语法树*/
9.          NodeType type,
10.       string nameStr,
11.       syntaxTree *child[2]
12. }
```

关键算法由算法 3 和算法 4 组成，算法 3 输入是用字符串表示的性质公式 $formula$，输出是公式对应的语法树$syntaxTree$，分别调用 Flex 和 Bison 两个工具实现将字符串形式的性质公式转化为公式语法树形式的功能。

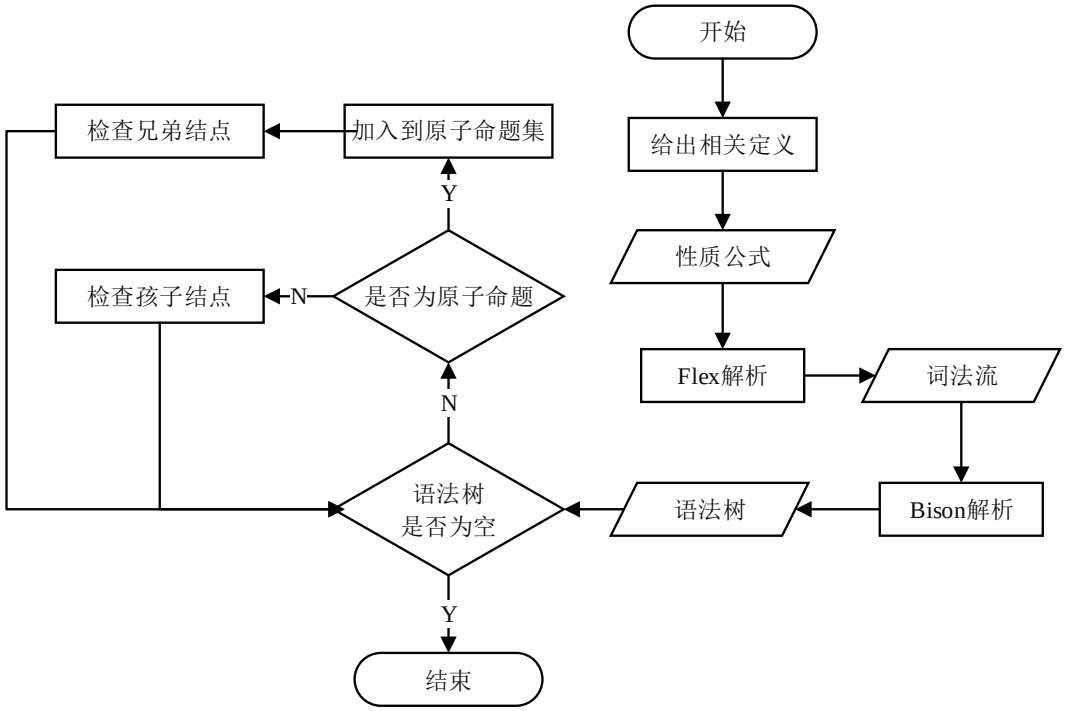


图3.10 解析性质公式算法流程图

Algorithm 3 生成语法树

Input: string *formula*

Output: syntaxTree *syntaxTree*

1: *formula*← getline() \\获取用户输入
2: *buffer_state*← FlexParser(*formula*)
3: *syntaxTree*← BisonParser(*buffer_state*)
4: **return** *syntaxTree*

算法 4 输入是公式的语法树$syntaxTree$，输出收集的原子命题集$AtomicSet$，通过先序遍历的方式遍历公式的语法树，从而实现收集公式中的原子命题集合的功能。算法第 1、14、15 行判断$syntaxTree$是否为空，若为空则跳出递归；若不为空，则判断$syntaxTree$结点类型是否为原子命题类型（IDENT_EXP）。若为原子命题类型则将该结点的名字属性值$nameStr$定义为一个原子命题命题$atomic$，并加入到$AtomicSet$，然后返回上一层递归，检查兄弟结点是否为空，如算法第 2-7 行所示；若不为原子命题类型，则判断$syntaxTree$的左孩子是否为空，若左孩子不为空，则递归调用算法 4，$syntaxTree$的右孩子操作同左孩子。

Algorithm 4 收集原子命题集

Input: syntaxTree *syntaxTree*

Output: set<string>*AtomicSet*

```
1: function CollectionAtomic(syntaxTree)
2:     if syntaxTree≠null then
3:         if syntaxTree.type==IDENT_EXP then \\结点是原子命题
4:             atomic←syntaxTree.nameStr
5:             if atomic∉AtomicSet then
6:                 AtomicSet.add(atomic)
7:             end if
8:         end if
9:         if syntaxTree.child[0]≠null then\\左孩子不为空
10:            CollectionAtomic(syntaxTree.child[0])
11:        end if
12:        if syntaxTree.child[1]≠null then \\右孩子不为空
13:            CollectionAtomic(syntaxTree.child[1])
14:        end if
15:    else
16:        break
17:    end if
18: end function
```

3.2.4 转换公式语法树

公式转换模块根据用户输入的性质公式和选择的挖掘算法将公式转为 LNFG 或 Büchi 自动机，关键算法由算法 5 和算法 6 组成，算法流程图如图 3.11 所示。

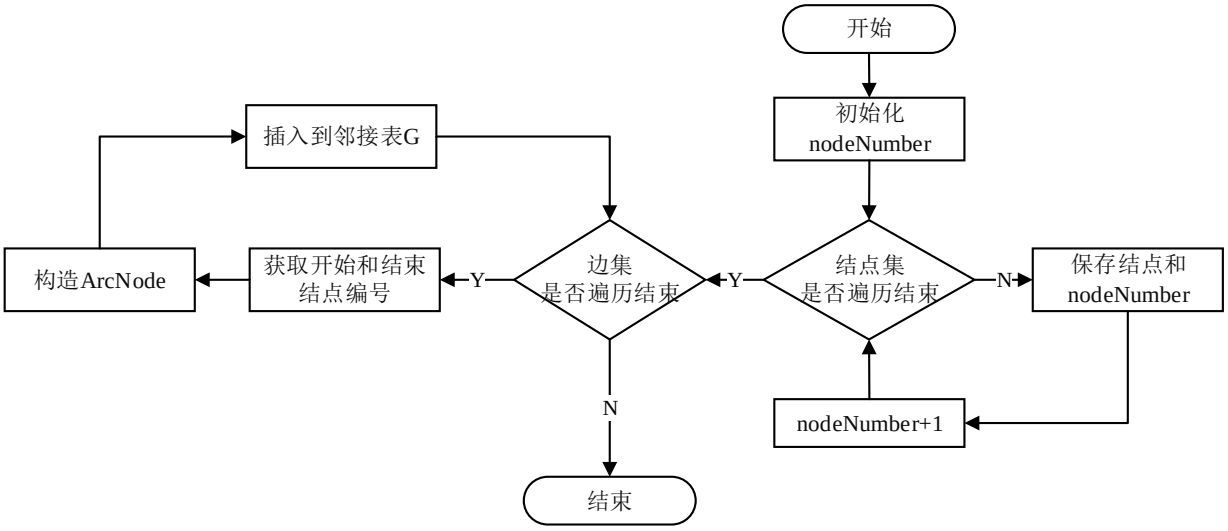


图3.11 转换公式语法树算法流程图

公式转换模块由 PPTL2LNFG 工具和 LTL3BA 工具组成，两个工具的输入都是公式语法树，PPTL2LNFG 工具的输出是 LNFG，LTL3BA 的输出是 Büchi 自动机。如第二章所述，PPTL2LNFG 转化生成的 PPTL 公式P的 LNFG 是一个有向图$G = (CL(P), EL(P), V_0, \mathbb{L} = \mathbb{L}_1, \cdots, \mathbb{L}_m)$，本文提出的挖掘算法不关注处于 LNFG 顶点的时序公式，而仅仅应用于边上的状态公式，故需要将结点集合$CL(P)$匿名化为$\{N_1, N_2, \cdots, N_{|CL(P)|}\}$，即用非负整数唯一标识 LNFG 的结点。LTL3BA 生成的 Büchi 自动机使用 Hanoi Omega-Automata (HOA)格式[53]保存，结点已经进行了匿名化处理。

算法 5 完成 LNFG 结点集的匿名化过程，输入是一个封装了 LNFG 结点的语法树形式的容器CL，输出是结点匿名化后的中间信息NC。在实现算法 5 前需要定义一个包含两个属性的结构体 nodeNumMessage，属性分别是时序公式的语法树形式和整数类型的结点编号，具体定义如下所示。

```
1.  typedef struct nodeNumMessage{
2.  syntaxTree temporalformula
3.  int nodeNum
4.  }
```

算法 5 定义一个初始值为 1 的整数型局部变量$nodeNumber$，用以表示匿名化后的结点整数值。算法第 2-6 行顺序遍历CL，定义 nodeNumMessage 类型的局部变量$nodeDetail$，将每个结点的语法树赋值给$nodeDetail$的$temporalformula$属性，结点编号赋值给$nodeNum$属性，并且$nodeNum$+1 作为下一个结点的编号。最后生成一个封装了包含每个结点的语法树和匿名化整数值的 nodeNumMessage 容器NC。

Algorithm 5 匿名化LNFG结点集

Input: vector<temporalformula > *CL*

Output: vector<nodeNumMessage> *NC*

1: *nodeNumber*←1
2: **for** *iter*=*CL*.begin$\cdots$ *CL*.end **do**
3: *nodeDetail*.temporalformula← *iter*
4: *nodeDetail*.nodeNum←*nodeNumber*
5: *nodeNumber*←*nodeNumber*+1
6: *NC*.insert(*nodeDetail*)
7: **end for**

在上一节的叙述中，LNFG 和 Büchi 自动机本质上都是一个带权值的有向图。有向图有开始结点和终止节点，其中，开始结点表示初始状态，终止节点表示可接受状态。在计算机领域，常采用邻接链表和邻接矩阵存储带权值的有向图，由于

LNFG 和 Büchi 自动机边上存储的语法树大小不确定，故为了提高空间利用率，本文采用邻接链表存储 LNFG 和 Büchi 自动机。邻接链表的头结点$VNode$和边结点$ArcNode$的定义如图 3.12 所示，其中，$vertex$为当前结点匿名化后的整数值，表示当前状态，$firstarc$表示指向相邻结点的指针。边结点的$info$域存储边上状态公式的语法树形式，用于表示迁移条件；$adjvex$表示当前结点经过迁移条件$info$到达的下一结点并匿名化后的整数值，表示下一状态，$nextarc$表示指向当前结点的下一相邻接点的指针。

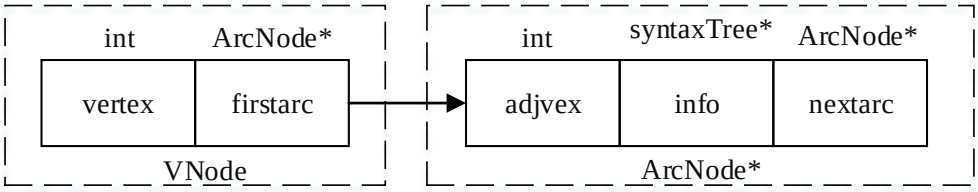


图3.12 邻接表定义示意图

算法 6 实现用邻接链表存储带权值的有向图的功能。共包含三个函数：函数 creatG、函数 getNumFromNC 和函数 insertG，在介绍算法 6 前，给出 LNFG 和 Büchi 自动机的边集定义，具体定义如下所示。

```
1.  typedef struct Edge{
2.      syntaxTree start, /*开始结点*/
3.      syntaxTree stateFormula, /*状态公式*/
4.      syntaxTree end /*终止结点*/
5.  }
```

函数 creatG 的输入是 LNFG 或 Büchi 自动机的结点信息NC和边集EL，输出是邻接链表G，G表示 LNFG 或 Büchi 自动机，参见算法 6 第 1-7 行。函数 creatG 顺序遍历EL的每条边，分别获得每条边的开始和结束结点的编号，最后将这条边插入G中。函数 getNumFromNC 的输入是语法树$temporalFormula$和算法 5 生成的NC，输出是$temporalformula$对应的编号$nodeNum$，参见算法 6 第 8-14 行。getNumFromNC 顺序遍历NC，若某个元素的$temporalformula$属性与输入相同，则返回该元素的$nodeNum$属性。函数 insertG 的输入有四个：（1）表示 LNFG 和 Büchi 自动机G；（2）一条有向边的弧尾结点编号$startNum$；（3）一条有向边的弧头结点编号$endNum$；（4）迁移条件$stateFormula$，输出是插入了输入的有向边的G，参见算法 6 第 15-22 行。insertG 首先根据$startNum$获得新边的头结点$Pfirst$，然后构造一个新的边结点$pNewNode$，然后根据输入信息为$pNewNode$各个属性赋值，最后使用头插法插入到G中。

Algorithm 6 构造带权值的有向图

Input: vector<nodeNumMessage> *NC* vector<edge> *EL*

Output: VNode* *G*

```
1: function CREATG(NC,EL)
2:     for iter=EL.begin···EL.end do
3:         startNum← GETNUMFORMNC(iter.start)
4:         endNum← GETNUMFORMNC(iter.end)
5:         INSERTG(G,startNum,stateFormula,endNum)
6:     end for
7: end function
8: function GETNUMFORMNC(temporalformula,NC)
9:     for iter=NC.begin···NC.end do
10:        if temporalformula==iter.temporalformula then
11:            return iter.nodeNum
12:        end if
13:    end for
14: end function
15: function INSERTG(G,startNum,stateFormula,endNum)
16:    *pFirst←G[startNum].firstarc
17:    *pNewNode← new ArcNode
18:    *pNewNode.adjvex← endNum
19:    *pNewNode.info← stateFormula
20:    *pNewNode.nextarc←pFirst
21:    G[startNum].firstarc←pNewNode
22: end function
```

3.2.5 实例化 LNFG 和 Büchi 自动机

公式实例化模块实现实例化 LNFG 和 Büchi 自动机的功能，关键算法如算法 7 和算法 8，算法流程图如图 3.13 所示。公式实例化模块首先根据$EventSet$和$AtomicSet$构造形如$(atomic, event)$的实例化映射对，然后将有向图G边上状态公式的原子命题$atomic$替换成事件$event$。

算法 7 的输入是$EventSet$和$AtomicSet$，输出是一个封装了结构体 iterStore 的容器$iterationTracker$。iterStore 首先定义了包含三个属性的结构体 iterStore：（1）指向原子命题的字符串 mapFrom；（2）指向$EventSet$的字符串指针 mapTo；（3）步长 switchVar。令$|EventSet| = m$，$|AtomicSet| = n$，则第 i 个原子命题切换成下一个事件的步长$\text{switchVar} = \text{m}^{(\text{n}-\text{i}-1)}$，$1 \le i \le n$，由算法 7 可知，$iterationTracker$共包含$n$个结构体 iterStore，iterStore 具体定义如下所示。

```
1.  typedef struct iterStore{
2.      int switchVar,
3.      string mapFrom,
4.      set<string>::iterator mapTo
5.  }
```

Algorithm 7 构造实例化构造器

Input: set<string> *AtomicSet* set<string> *EventSet*

Output: vector<iterStore> *iterationTracker*

1: $i \leftarrow 0$
2: **for** *atomic*=*AtomicSet*.begin$\cdots$*AtomicSet*.end **do**
3: *insertVar*.mapFrom$\leftarrow$ *atomic*
4: *insertVar*.mapTo$\leftarrow$ *EventSet*.begin
5: insertVar.switchVar$\leftarrow$pow(*EventSize*.size,*AtomicSet*.size-1-i)
6: $i \leftarrow i$-1
7: *iterationTracker*.insert(*insertVar*)
8: **end for**

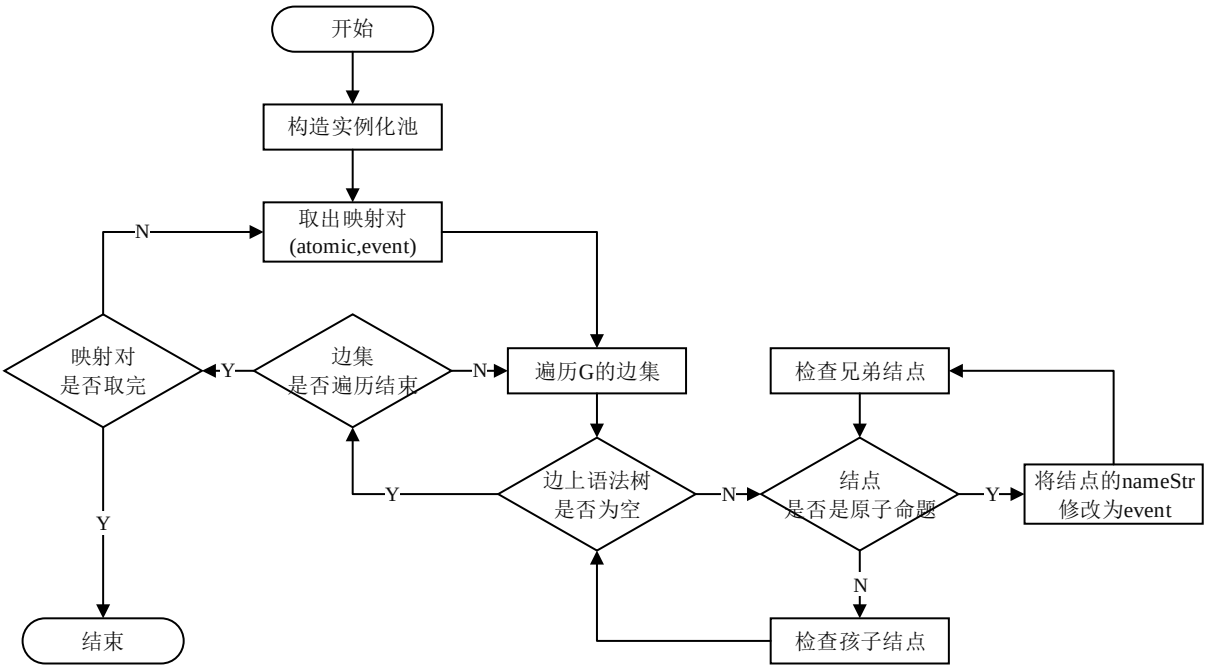


图3.13 实例化 LNFG 和 Büchi 自动机

将$iterationTracker$输入算法 8（函数 getInstantiation），输出映射对$instantMap$。trackerVar 表示一个初始值为 0 的整数值，记录已经构成的映射对的个数，当 trackerVar 是该原子命题对应的 switchVar 的整数倍时，切换为下一事件。算法 8 第 2-10 行定义第一个 for 循环，构造了一个包含n个的映射对$instantMap$，且 key 为原子命题，value 为事件。算法 8 第 12-19 行定义第二个 for 循环，用来检查$instantMap$

是否合理，若不同原子命题对应的事件不同，则说明$instantMap$合理，反之则重新调用函数 getInstantiation 生成$instantMap$。由算法 8 可知，公式实例化模块构造了m^n个$instantMap$，且仅有P_m^n个合理，其中$P_m^n = m \times (m-1) \times \cdots \times (m-n+1)$。

Algorithm 8 构造实例化对

```
Input: vector<iterStore> iterationTracker
Output: map<string,string> instantMap
1: function GETINSTANTIATION(iterationTracker)
2:     for iter=iterationTracker.begin··· iterationTracker.end do
3:         if trackerVar%iter.switchVar==0 then \\切换为下一事件
4:             iter.mapTo←iter.mapTo+1
5:             if iter.mapTo==EventSize.end then \\循环遍历
6:                 iter.mapTo←EventSize.begin
7:             end if
8:         end if
9:         instantMap.insert(iter.mapForm,iter.mapTo)
10:    end for
11:    eventUsed←set<string>
12:    for iter1=instantMap.begin··· instantMap.end do
13:        if iter1.second∉eventUsed then \\不同原子命题对应不同事件
14:            eventUsed.insert(iter1.second)
15:        else
16:            trackerVar←trackerVar+1
17:            return GETINSTANTIATION(iterationTracker)
18:        end if
19:    end for
20: end function
```

算法 9 包含两个函数：函数 instantiateTree（算法第 1-13）和函数 iterationMap（算法 14-23）。函数 instantiateTree 的输入是语法树$syntaxTree$和映射对$iterationMap$，输出是实例化后的语法树。函数 instantiateTree 采用先序遍历的方式实现实例化，若$syntaxTree$结点类型是原子命题，则在 iterationMap 中获得对应的 value 值，即事件 event，并用 event 替换原子命题的 nameStr 属性值；若$syntaxTree$的左右孩子不为空，则递归调用函数 instantiateTree。函数 instantiateG 的输入是G和$iterationMap$，输出是$instantiatedG$。函数采用广度优先遍历的方式遍历邻接表G，在遍历的过程中调用 instantiateTree 函数实例化边结点的权值$syntaxTree$。

Algorithm 9 实例化带权值的有向图

Input: VNode* *G* map<string,string> *instantMap*

Output: VNode* *instantiatedG*

```
1: function INSTANTIATETREE(syntaxTree,instantMap)
2:     if syntaxTree.type==INDET_EXP then \\ 结点为原子命题
3:         atomic←syntaxTree.nameStr
4:         event←instantMap.get(atomic)
5:         syntaxTree.nameStr←event
6:     end if
7:     if syntaxTree.child[0]≠null then \\左孩子不为空
8:         INSTANTIATETREE(syntaxTree.child[0])
9:     end if
10:    if syntaxTree.child[1]≠null then \\右孩子不为空
11:        INSTANTIATETREE(syntaxTree.child[1])
12:    end if
13: end function
14: function INSTANTIATEG(G,instantMap)
15:    for iter=1··· CL.size+1 do
16:        p←G[i].firstarc
17:        while p≠null do \\存在下一结点
18:            syntaxTree←p.info
19:            INSTANTIATETREE(syntaxTree,instantMap)
20:            p←p.nextarc
21:        end while
22:    end for
23: end function
```

3.2.6 验证实例化性质公式

公式验证模块完成τ和实例化 LNFG 或 Büchi 自动机instantiateG的检测，算法流程图如图 3.14 所示，关键算法由算法 10（CheckBaseOnBFS）、算法 11（StateTransition）和算法 12（StateFormulaCheck）组成。由邻接表的定义可知，头结点表示当前状态，边结点的数据域表示下一状态，边结点的权值域表示公式对应的语法树形式P，公式验证模块主要通过检测τ的当前事件是否满足P来确定下一状态是否可达。

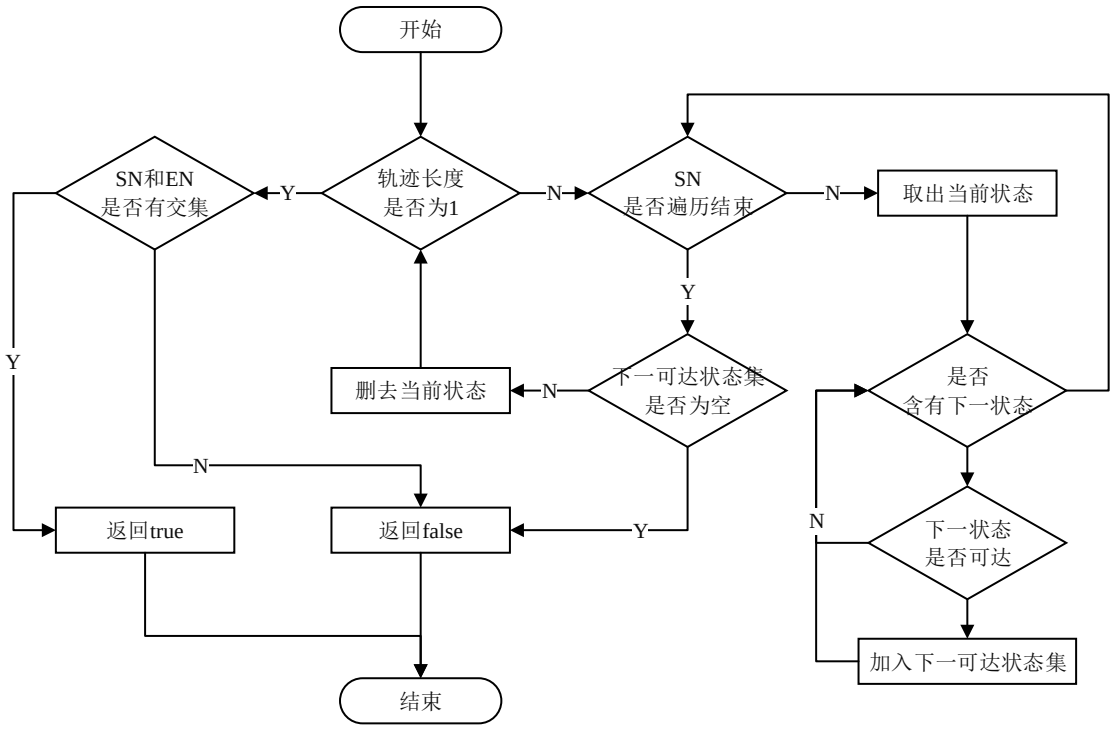


图3.14 公式检测模块算法流程图

算法 10 是公式验证模块的入口算法，由函数 checkIsSatisfied 实现，输入包含四个，分别是实例化的邻接表$instantiatedG$、封装了 event 的 trace τ、当前状态集SN和终止状态EN，输出是表示$instantiatedG$是否满足τ的布尔变量$isSatisfied$，且$isSatisfied \in \{true, false\}$。函数 checkIsSatisfied 首先判断τ中是否还有当前事件，若τ只含有一个 trace 分隔符（算法 10 第 1 行），即长度为 1 时，表示没有当前事件，则继续判断当前状态SN是否含有终止状态ε，即判断SN和EN是否有交集（算法 10 第 2 行），若含有ε，则返回$true$，表示$instantiatedG$满足τ，反之则返回$false$。若τ含有当前事件时，则通过调用算法 11 获得当前状态集的下一可达状态集$SNNext$，若$SNNext$为空，则返回$false$，否则删去当前事件再次调用算法 10。

算法 11 的功能由函数 stateTransition 实现，函数的输入包含三个变量，分别是（1）τ；（2）实例化的邻接表$instantiatiatedG$；（3）当前状态集中的一个状态$nodeNum$。输出是$nodeNum$的下一可达状态集$reachableSet$。函数 stateTransition 首先获取当前状态的下一状态链表的头指针q，然后依次遍历链表中的边结点，并通过算法 12 探测下一状态是否为可达。若下一状态可达，则将该状态加入下一可达状态集$reachableSet$中；若不可达，则继续判断下一状态是否可达。

Algorithm 10 入口算法

Input: VNode* *instantiatedG* vector<event> τ set<int> *SN* set<int> *EN*

Output: boolean *isSatisfied*

```
1: function checkIsSatisfied(VNode* instantiatedG vector<event> τ set<int> SN
   set<int> EN)
2:     if τ.length==1 then \\trace遍历结束
3:         if SN∩EN≠null then \\ 结点集包含终止结点
4:             return true
5:         else
6:             return false
7:         end if
8:     else
9:         for iter=SN.begin···SN.end do
10:            reachableSet←stateTransition(*iter,G,τ)
11:            SNNext.insert(reachableSet.begin,reachableSet.end)
12:        end for
13:        if SNNext.size==0 then \\ 下一可达节点集为空
14:            return false
15:        else\\删除事件(tail)继续检查
16:            checkIsSatisfied(SNNext,EN,tail(τ))
17:        end if
18:    end if
19: end function
```

Algorithm 11 状态转移算法

Input: VNode* *instantiatedG* vector<event> τ set<int> *nodeNum*

Output: set<int> *reachableSet*

```
1: function stateTransition(VNode* instantiatedG vector<event> τ set<int> node-
   Num)
2:     p←instantiatedG[nodeNum].firstarc
3:     while p≠null do \\存在下一结点
4:         if syntaxTreeCheck(p.info,τ) then
5:             reachableSet.insert(p.adjvex)
6:         end if
7:         p←p.nextarc
8:     end while
9: end function
```

Algorithm 12 状态可达性探测算法

Input: syntaxTree *syntaxTree* vector<event> τ

Output: boolean *isReachable*

```
1: function SYNTAXTREECHECK(syntaxTree,τ)
2:     switch syntaxTree.type do
3:         case ∨
4:             q1←SYNTAXTREECHECK(syntaxTree.child[0],τ)
5:             if q1==true then
6:                 return true
7:             else
8:                 return SYNTAXTREECHECK(syntaxTree.child[1],τ)
9:             end if
10:        case ∧
11:            q1←SYNTAXTREECHECK(syntaxTree.child[0],τ)
12:            if q1==true then
13:                return SYNTAXTREECHECK(syntaxTree.child[1],τ)
14:            else
15:                return false
16:            end if
17:        case ¬
18:            q1←SYNTAXTREECHECK(syntaxTree.child[0],τ)
19:            if q1==true then
20:                return false
21:            else
22:                return true
23:            end if
24:        case INENT_EXP
25:            if syntaxTree.nameStr==head(τ) then
26:                return true
27:            else
28:                return false
29:            end if
30:        case TRUE
31:            return true
32:        case FALSE
33:            return false
34: end function
```

算法 12 由函数 syntaxTreeCheck 实现，用来检测迁移条件是否与当前事件在语义上保持一致，输入是边上表示迁移条件的语法树$syntacTree$和τ，输出是表示事件是否满足语法树的布尔变量$isReachable$。由于状态公式只包含“∧、∨、p、¬、$true$、$false$”操作符，因此只需按照逻辑语义判断语法树的结点类型即可，其中$p \in Prop$。函数 syntaxTreeCheck 根据$syntacTree$根结点的类型进行判断，若结点类型为“∨”时（算法第 3-9 行），则需要先对左结点递归调用函数 syntaxTreeCheck，若返回值为 false 时，则对右结点递归调用函数 syntaxTreeCheck，反之则直接返回 true。若结点类型为“∧”（算法第 10-16 行），则需要先对左结点递归调用函数 syntaxTreeCheck，若返回值为 true 时，则对右结点递归调用本函数，反之则直接返回 false。若结点类型为“¬”（算法第 17-23 行），则需要对其子结点递归调用函数 syntaxTreeCheck 并对其结果取反。若根节点为“p”（算法第 24-29 行），则将结点的 nameStr 属性值与当前事件进行比较，若相同则表示下一状态为下一可达状态，反之则表示下一状态不可达。若根节点为“$true$、$false$”时，对应返回“$true$、$false$”即可。

3.3 安装与使用方法

根据上一节提出的基于 LNFG 和 Büchi 自动机的挖掘算法，本文开发出对应的规范挖掘工具 PPTLMiner+。PPTLMiner+是一个使用 C++语言编写的命令行工具，包含 25 个头文件*.h、23 个程序文件*.cpp、1 个词法文件*.l、1 个语法文件*.y 和一个 Makefile 文件，代码量约 19000 行左右，代码框架树如下所示：

```
*/pptlminer -- 最顶层目录，包含程序的 makefile 文件、trace 文件、配置文件等
    */pptlminer-src —包括程序的全部源码和生成的全部.o、.d 文件
        */bin— makefile 生成的包含*.o、*.d 文件
        */src—源码
            */main—包含 main 方法、命令行配置文件和规范挖掘算法的入口算法
            */pptlparser —包含解析语法树和转为 NF、LNFG、Büchi 自动机等全
                          部方法
            */parsers—包含解析 trace 文件、设置事件分隔符、设置 trace 分隔符、
                       解析正则表达式表示的事件、收集事件集等全部方法
            */pptlcheckers—包含基于 NF、LNFG 和 Büchi 自动机的规范挖掘方法
            */checkers—包含置信度等配置的全部方法
            */instantiation-tools—包含构造实例化池和收集原子命题集的全部方法
            */trace—包含事件的定义以及对事件的一些操作的全部方法
```

PPTLMiner+在使用前需要安装 Boost 库、不变量挖掘工具 Daikon、Flex 工具和 Bison 工具，其中，Boost 用来解析执行指令；Daikon 通过运行可执行程序生成不变量文件；Flex 和 Bison 用来解析性质公式。将 PPTLMiner+工具包放在任意目录下即可使用。PPTLMiner+通过执行用户在终端输入的指令完成规范挖掘工作，输入“./pptlminer”启动挖掘工具，输入“-h”或“--help”查看 PPTLMiner+的帮助文档，帮助文档对应指令的功能如表 3.1 所示。

表3.1 PPTLMiner+帮助文档

指令	功能
-h [--help]	查看帮助文档
-f [--property-type] arg	待验证的性质
--trace-file arg	待挖掘的轨迹文件
-n [--NF-trace]	基于 NF 挖掘轨迹文件的性质
-g [--LNFG-trace]	基于 LNFG 挖掘轨迹文件的性质
-b [--BA-trace]	基于 BA 挖掘轨迹文件的性质
--print-stats	打印挖掘结果
--trace-separator arg	设置轨迹文件中的 trace 分隔符，默认：--
--event-separator arg	设置轨迹文件中的事件分隔符，默认：..
-e [--event] arg	指定公式中的原子命题解释为常量
-r [--regex] arg	用于从轨迹文件中分析事件类型的正则表达式
	默认：(?<ETYPE>.*)
--parse-mult-prop	指定事件为多命题类型，在事件分隔符前后的行被解释为单个命题
-i [--ignore-nm-lines]	忽略不匹配的行，默认设置为 false
--out-file arg	将输出结果保存在指定文件中

其中，“-f arg”表示待挖掘的性质公式是 arg。“--trace-file arg”表示待挖掘的 trace 文件名是 arg。“-n”、“-b”、“-g”指明挖掘算法，有且只能出现一个。其中“-n”表示使用基于 NF 的挖掘方法[11]；“-b”表示使用基于 Büchi 自动机的挖掘方法；“-g”表示使用基于 LNFG 的挖掘方法。“-r arg”表示使用正则表达式 arg 来解析 trace 中的事件，默认是“(?<ETYPE>.*)”。“--trace-separator arg”表示 trace 分隔符是 arg，默认是“--”。“--event-separator arg”表示 event 分隔符是 arg，默认是“..”。“--parse-mult-prop”表示将事件解析成多原子命题，即一个事件由多个原子命题组成，且表示同一个原子命题的事件之间使用事件分隔符隔开。“-i”表示在遍历 trace 时忽略以“//”开始的事件，即不将注释行解析成事件。“--print-stats”表示打印最终挖掘结

果。“-e arg”表示将公式中的原子命题 arg 解析成一个不变事件，即在实例化时，arg 不会映射成其他事件。“--out-file arg”将输出结果保存在 arg 文件中。

输入命令“./pptlminer -g -f ‘p;q’ –log-file traceFile.txt”到终端，PPTLMiner+执行结果示意图如图 3.15 所示。其中，区域 1 表示轨迹解析模块生成的事件集$EventSet$；区域 2 表示公式解析模块生成的原子命题集$AtomicSet$；区域 3 表明采用基于 LNFG 的挖掘算法；区域 4 展示公式转换模块生成的 LNFG 的边信息；区域 5 表示挖掘时长；区域 6 表示挖掘结果，指明轨迹文件、性质公式和规范。

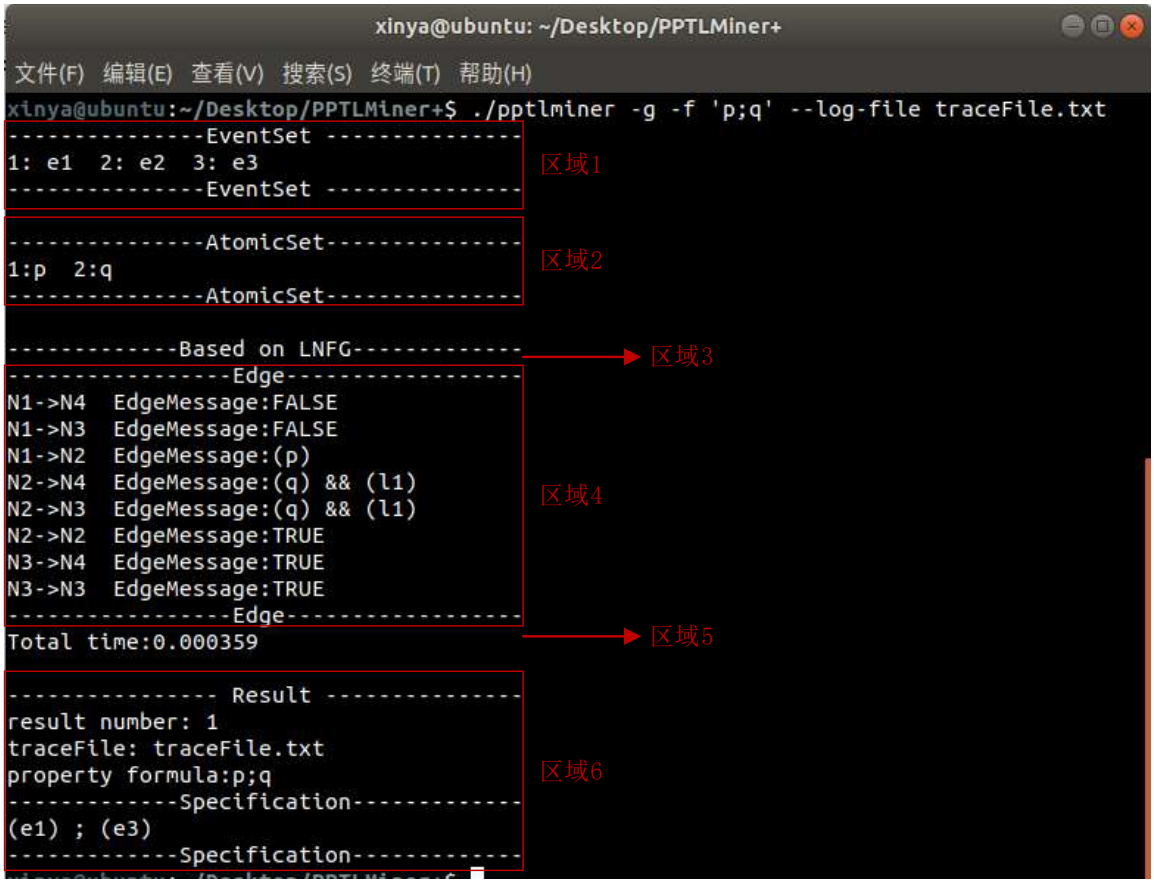


图3.15 PPTLMiner+运行结果示意图

3.4 功能示例

为了更加直观地展示 PPTLMiner+的功能，本小节的示例中直接给出轨迹生成模块生成的 trace 文件。其中示例一为基础功能，示例二、示例三和示例四为特色功能，根据上一节列出的帮助文档，用户可以配置不同的命令参数来满足需求。

示例一

若 basic_log.txt 是一个 trace 文件，解析后生成 trace $\tau_1 \stackrel{\text{def}}{=}$(a, c, d, --, b, d, c, d)。输入命令$comd_1 \stackrel{\text{def}}{=}$“./pptlminer -b -f ‘$(x \rightarrow \bigcirc \Diamond\ y)$’ --log-file basic_log.txt --trace-separator

'END' ”。默认情况下，trace分隔符为“--”，$comd_1$将分隔符指定为“END”，故τ_1包含两条 trace，即“a,c,d”和“b,d,c,d”，且$EventSet_1 \overset{\text{def}}{\equiv} (a,b,c,d)$。待挖掘的性质公式$f_1 \overset{\text{def}}{\equiv} \square\,(x \rightarrow \bigcirc \Diamond\ y)$表示“原子命题$x$发生后总是会发生原子命题$y$”，公式解析模块解析$f_1$后获得原子命题集$AtomicSet_1 \overset{\text{def}}{\equiv} \{x, y\}$。$comd_1$选择基于 Büchi 自动机的挖掘算法。

执行$comd_1$后，PPTLMiner+挖掘规范的结果如表 3.2，如编号 1 的结果“$\square\,(a \rightarrow \bigcirc \Diamond\ c)$”表示事件$a$发生后总是为发生事件$c$。在“a,c,d”这条 trace 中，事件$a$发生后紧接着发生事件$c$，故命题为真，即$\square\,(a \rightarrow \bigcirc \Diamond\ c)$ 满足“a,c,d”；在“b,d,c,d”这条 trace 中，事件a并不发生，即蕴含（→）的前提条件为假，则命题为真，即$\square\,(a \rightarrow \bigcirc \Diamond\ c)$ 满足“b,d,c,d”。因此$\square\,(a \rightarrow \bigcirc \Diamond\ c)$满足$\tau_1$。

表3.2 PPTLMiner+挖掘规范结果 1

序号	结果
1	$\square\,(a \rightarrow \bigcirc \Diamond\ c)$
2	$\square\,(a \rightarrow \bigcirc \Diamond\ d)$
3	$\square\,(b \rightarrow \bigcirc \Diamond\ c)$
4	$\square\,(b \rightarrow \bigcirc \Diamond\ d)$
5	$\square\,(c \rightarrow \bigcirc \Diamond\ d)$

示例二

若 structured_log.txt 是一个 trace 文件，文件内容如图 3.16 所示。输入命令$comd_2 \overset{\text{def}}{\equiv}$“./pptlminer -g -f '[](x->()<>y)' --log-file structured_log.txt -r 'SUCCESS: (?<ETYPE>.*)' -r 'FAIL: (?<ETYPE>.*)' -i”。

```
// This is a comment
SUCCESS: a
SUCCESS: c
// This is another commment
FAIL: d
--
SUCCESS: b
FAIL: d
SUCCESS: c
SUCCESS: d
--
```

图3.16 structured_log.txt 文件内容

若使用 PPTLMiner+的默认设置，“// This is a comment”将被认为是一个单独的事件。但命令$comd_2$中使用“-i”设置忽视注释行，则这一行在解析过程中将被忽略。“-r 'SUCCESS: (?<ETYPE>.*)' -r 'FAIL: (?<ETYPE>.*)' ”表示根据正则表达式解析事件，如“SUCCESS:d”和“FAIL：d”被认为是同一个事件“d”。解析 structured_log.txt 后生成 trace $\tau_2 \stackrel{\text{def}}{=}$(a, c, d, --, b, d, c, d)。

在设置特殊指令“-i”和“-r”后，解析获得的 trace 和挖掘的性质公式与示例一相同，故执行$comd_2$后，PPTLMiner+得到的规范挖掘结果同示例一。

示例三

若 trace 文件 constant_events_log.txt 如图 3.17 所示，trace 解析模块解析获得 $\tau_3 \stackrel{\text{def}}{=}$(a, a, q, b, b, c, c, --, c, c, q, b, b, a, a)，输入命令$comd_3 \stackrel{\text{def}}{=}$“./pptlminer -g -f '[](p->()<>q)' -e 'q' --log-file constant_events_log.txt”。trace 解析模块解析τ_3后获得事件集$EventSet_{2-1} \stackrel{\text{def}}{=}${a, b, c, q}。命令中性质公式为$f_2 \stackrel{\text{def}}{=} \square(p \rightarrow \bigcirc \Diamond\ q)$，表示这种情况总是为真，即若原子命题$p$发生，则接下来一定存在原子命题$q$。公式解析模块解析$f$获得原子命题集$AtomicSet_{2-1} \stackrel{\text{def}}{=}${p,q}。但 $comd_3$通过设置特殊指令“-e q”将原子命题 q 指定为不变事件，故在实例化时，原子命题集$AtomicSet_2 \stackrel{\text{def}}{=}${p}，事件集$EventSet_2 \stackrel{\text{def}}{=}${ a, b, c}。

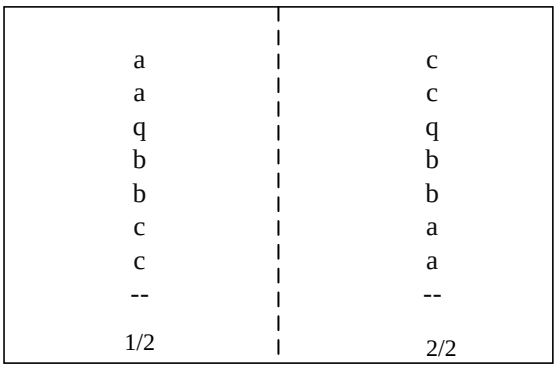


图3.17 constant_events_log.txt 文件内容

执行$comd_3$后，挖掘结果如表 3.3 中编号 1-3 所示。如编号 1 的结果是“$\square(a \rightarrow \bigcirc \Diamond\ q)$”表示若第一个事件是“a”，则后续必存在事件“q”为真的情况。如第一条 trace“a, a, q, b, b, c, c”中，第一个事件是“a”，且第三个事件是“q”，则事件“a”发生后，后续状态存在事件“q”。在第二条 trace“c, c, q, b, b, a, a”中，第一个事件为“c”，即蕴含命题的前提条件为假，故检测结果为真。则$\square(a \rightarrow \bigcirc \Diamond\ q)$满足$\tau_3$。若$comd_3$不设置“-e”参数，则 PPTLMIner+的规范挖掘结果有 12 条，见表 3.3 编号 1-12。由此可见，设置“-e”参数后，结果更简洁更精确，比较适用于用户想要探究特定事件与其他事件关系的场景。

表3.3 PPTLMiner+挖掘规范结果 2

序号	结果	序号	结果
1	$\square(a \rightarrow \bigcirc \Diamond\ q)$	7	$\square(b \rightarrow \bigcirc \Diamond\ c)$
2	$\square(b \rightarrow \bigcirc \Diamond\ q)$	8	$\square(c \rightarrow \bigcirc \Diamond\ a)$
3	$\square(c \rightarrow \bigcirc \Diamond\ q)$	9	$\square(c \rightarrow \bigcirc \Diamond\ b)$
4	$\square(a \rightarrow \bigcirc \Diamond\ b)$	10	$\square(q \rightarrow \bigcirc \Diamond\ a)$
5	$\square(a \rightarrow \bigcirc \Diamond\ c)$	11	$\square(q \rightarrow \bigcirc \Diamond\ b)$
6	$\square(b \rightarrow \bigcirc \Diamond\ a)$	12	$\square(q \rightarrow \bigcirc \Diamond\ c)$

示例四

若 trace 文件 mult_prop_log.txt 的文件内容如图 3.18 所示，输入命令$comd_4 \triangleq$“./pptlminer -g -f '(p;q) prj r' --log-file mult_prop_log.txt --parse-mult-prop”。性质公式为$f_3 \triangleq (p;q)\ prj\ r$，表示“$p;q$”和原子命题“$r$”存在投影关系，公式解析模块解析$f_3$后获得原子命题集$AtomicSet_3 \triangleq${p, q, r}。由于$comd_4$ 中指定“--parse-mult-prop”参数且没有重新指定事件分隔符，则按照默认情况处理，将 mult_prop_log.txt 认为是一个由多命题事件组成的 trace 文件，多命题事件即在一个时间点有多个原子命题。解析 mult_prop_log.txt 后生成 trace τ_4的示意图如图 3.19。由示意图可知，在第一个和第二个时间节点上，“a = 0，b = 1，c = 0”同时为真；在第三个和第四个时间节点上，“a = 1，b = 1，c = 0” 同时为真，在第五个时间节点上，“c = 0”为真。轨迹解析模块解析τ_4后获得事件集$EventSet_3 \triangleq${a = 0，a = 1，b = 1，c = 0}。“-g”表示采用基于 LNFG 的规范挖掘方法。

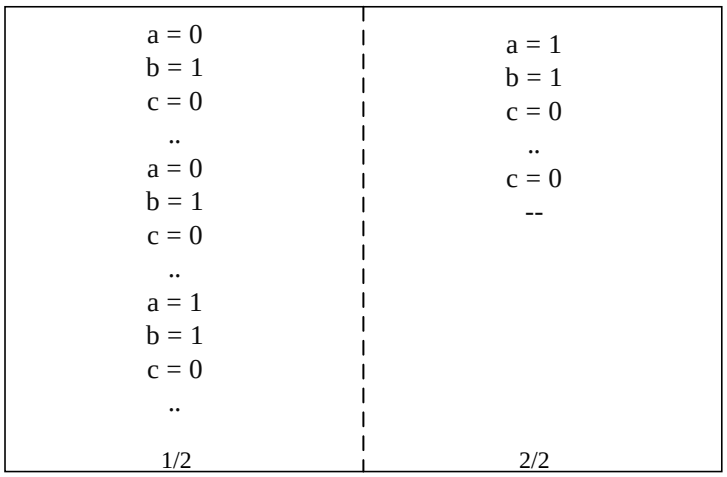


图3.18 mult_prop_log.txt 文件内容

执行$comd_4$后，PPTLMiner+挖掘规范的结果如表 3.4 所示。其中编号 1-5 的结果

对应图 2.1 的第一个语义，编号 6-7 对应第 2 个语义，编号 8-12 对应第三个语义。如：在第一个和第二个时间节点构成的区间上“a = 0”成立，在第三个和第四个时间节点构成的区间上“a=1”成立，且在整个区间上“c=0”成立，故编号 1 的结果“$((a = 0);(a = 1))\ prj\ ((c = 0))$”符合$prj$的第一个语义。

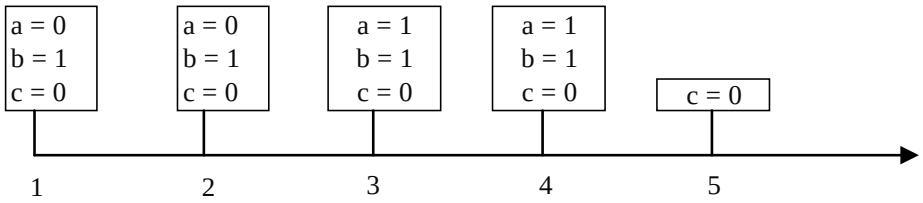


图3.19 mult_prop_log.txt 的 trace 示意图

表3.4 PPTLMiner+挖掘规范结果 3

序号	结果	序号	结果
1	$((a = 0);(a = 1))\ prj\ ((c = 0))$	7	$((c = 0);(a = 1))\ prj\ ((b = 1))$
2	$((a = 0);(b = 1))\ prj\ ((c = 0))$	8	$((a = 0);(c = 0))\ prj\ ((b = 1))$
3	$((b = 1);(a = 0))\ prj\ ((c = 0))$	9	$((b = 1);(a = 1))\ prj\ ((a = 0))$
4	$((b = 1);(a = 1))\ prj\ ((c = 0))$	10	$((b = 1);(c = 0))\ prj\ ((a = 0))$
5	$((c = 0);(a = 0))\ prj\ ((b = 1))$	11	$((c = 0);(a = 1))\ prj\ ((a = 0))$
6	$((a = 0);(a = 1))\ prj\ ((b = 1))$	12	$((c = 0);(b = 1))\ prj\ ((a = 0))$

3.5 本章小结

本章首先提出了基于 LNFG 和 Büchi 自动机的规范挖掘框架，然后分别说明了框架中各模块的功能。接着，从算法流程图和伪代码两个角度详细阐述了挖掘算法的实现。其次，基于本文提出的算法开发了挖掘工具——PPTLMiner+，介绍了PPTLMiner+的安装与使用方法。最后通过四个具体的示例，介绍了 PPTLMiner+的基础功能和特色功能。至此，本文提出的挖掘算法以及挖掘工具已经介绍完毕，下一章将通过具体的应用说明算法和工具的有效性和实用性。

第四章　基于蚁群算法解决旅行商问题的规范挖掘

规范挖掘可用于软硬件系统的形式化验证领域，本章通过挖掘群体智能中的典型算法——蚁群算法解决旅行商问题的规范，验证解决旅行商问题的程序与算法设计时提出的需求保持一致，即程序中函数的时序关系与算法流程图一致。本章首先介绍旅行商问题（Traveling Salesman Problem，TSP）[54]的背景并进行建模，然后介绍了蚁群算法[55]以及基于此算法解决旅行商问题的算法流程图。接着，从规范挖掘角度讲述了挖掘解决旅行商问题规范的步骤，并详细展示了每一模块的工作结果，以便读者能够更好地理解本文提出算法的设计思想，以此证明了本文算法和工具的有效性和实用性。最后分析规范挖掘的结果，比较结果与流程图是否保持一致。

4.1 背景介绍

4.1.1　旅行商问题

旅行商问题是一个经典的组合优化问题，分为经典旅行商问题（CTSP）、不对称旅行商问题（CTSP）、配送收集旅行商问题（TSPPD）、多人旅行商问题（MTSP）和多目标旅行商问题（CRISP）。本文探究目前研究成果较多的经典旅行商问题，以下旅行商问题均指经典旅行商问题。CTSP 描述为：一个旅行商人需要到多个城市推销商品。商人需要从一个城市出发，经过每个城市一次，最终回到出发城市。商人希望选择一条最短的路径，使得他可以完成所有城市的拜访。从图论角度阐述 CTSP 为：在给定带权值的无向完全图中找一个权值最小的哈密顿（Hamilton）回路。

下面对旅行商问题进行建模。给定带权值的无向完全图$G(V,E)$，其中，V是点集，表示城市编号的集合；E是边集，表示相邻城市的距离。假设V中编号为$1\cdots N$，c_{ij}表示城市i到城市j的距离，且$i\neq j$，$c_{ii}=M$，保证M至少比图G中最长的距离值大；$x_{ij}\in\{0,1\}$且满足如下约束：

$$x_{ij}=\begin{cases}1, \text{存在从城市 i 到城市 j 的 TSP 解决方案}\\ 0, \text{其他}\end{cases}$$

Miller、Tucker 和 Zemlin 提出了描述旅行商问题的经典模型 MTZ[56]，模型如下：

$$min\sum_{i}\sum_{j}c_{ij}x_{ij}$$

其中，模型的约束条件共 3 个，分别是（1）$\sum_{j=1}^{j=N}x_{ij}=1$，$\forall i\in V$；（2）$\sum_{i=1}^{i=N}x_{ij}=$

1，$\forall j \in V$；（3）$u_i - u_j + Nx_{ij} \leq N-1$，且$1 < i \neq j \leq N$，$x_{ij} \in 0,1$，$\forall u_j \geq 0$。

在上述约束条件中，约束（1）的数学含义是任意节点（城市）i的出度为 1，在 TSP 问题中对应着每个城市只有一个出边；约束（2）的数学含义是任意节点（城市）i的出度为 1，在 TSP 问题中对应着每个城市只有一个出边；约束（3）目的是消除子环约束，分为两部分：i)$u_i - u_j + Nx_{ij} \leq N-1$：若一条可行的访问路径中经过$< i,j >$，即$x_{ij} = 1$，则不等式变换为$u_i \leq u_j - 1$，即$u_i < u_j$；当一条路径存在重复节点（形成环）时，例如$i \to j \to k \to i$，则有$u_i < u_j < u_k < u_i$，出现矛盾$u_i < u_i$，故$u_i - u_j + Nx_{ij} \leq N-1$能限制路径不包含任意环；ii) 由于 TSP 问题本身是一个经过所有节点的最大的环，目的是消除子环而不是消除 (最大的) 环，故使用$1 < i \neq j \leq N$作为约束，即剔除第一个节点，只针对剩余节点，使其不能成环 (是子环)，从而实现消除子环。

4.1.2 蚁群算法

蚁群算法是受到蚂蚁觅食行为启发而提出的一种算法[55]。在蚂蚁觅食时，蚂蚁会释放一种叫做信息素的物质，其他蚂蚁可以通过察觉信息素浓度来选择路径。当一条路径上的蚂蚁越来越多时，该路径上的信息素浓度会不断增加，从而形成了一种正反馈机制。这种机制最终会导致整个蚁群沿着最短路径找到食物。因此，整个蚁群可以看作是一个增强型学习系统。蚁群算法就是基于这种觅食行为提出的一种优化算法，可以应用于许多优化问题的求解。蚁群觅食行为如图 4.1 所示。

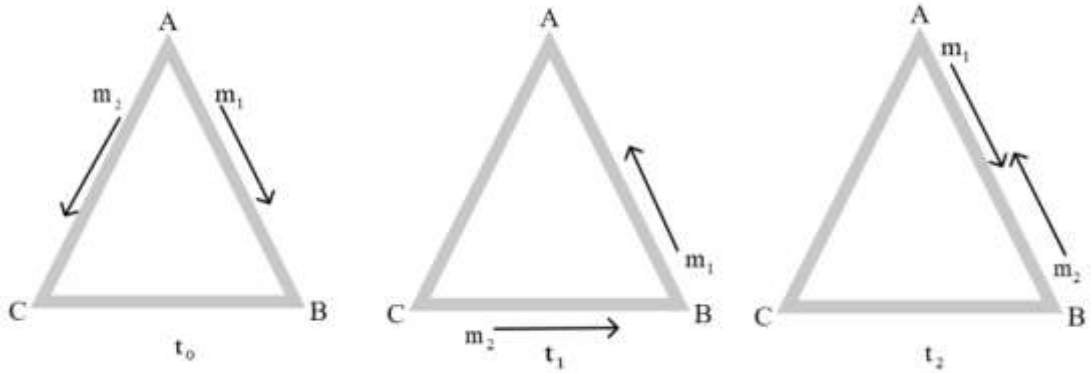


图4.1 蚁群觅食行为示意图

如图 4.1 所示，A 点是蚁穴，B 点是食物源，C 点是路径上的一点，A、B、C 形成一个等边三角形。蚁穴有两只蚂蚁：m_1和m_2，假设m_1和m_2同时出发且移动速度相同。在t_0时刻，m_1选择 AB 路径，m_2选择 AC 路径。在t_1时刻，m_2到达 C 点，m_1到达 B 点，此时 AB 路径有信息素而 BC 路径没有，因此在下一时刻，m_1选择 AB 路

径返回 A 点。在t_2时刻，m_1到达 A 点，此时 AB 路径上信息素浓度是 AC 路径的两倍，故在下一时刻，m_1选择 AB 路径继续前往 B 点搬运食物；m_2到达 B 点，且 AB 路径上信息素浓度是 BC 路径的两倍，故在下一时刻，m_2选择 AB 路径继续前往 A 点。经过不断的迭代，蚂蚁在 AB 路径上释放的信息素浓度会不断增加，而 AC 路径和 BC 路径上的信息素浓度会逐渐减少，这种机制使得蚁群最终能够找到从蚁穴到食物源的最短路径。

4.1.3 蚁群算法解决旅行商问题

旅行商问题是一种经典的 NP 问题[54]，目前有多种解决方法，比较经典的解决方法有：贪心算法、遗传算法和模拟退火算法。本文采用遗传算法中的蚁群算法来解决旅行商问题。蚁群算法解决旅行商问题的基本框架就是让m个旅行商分别去构造路径并留下信息素，且在下一次迭代时利用信息素再次构造路径，蚁群算法解决旅行商问题的流程如图 4.2 所示。

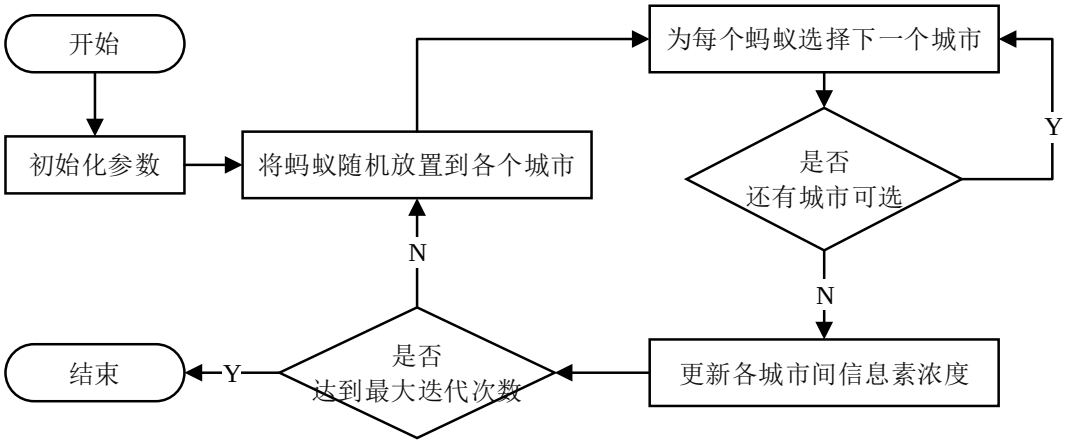


图4.2 蚁群算法解决旅行商问题的流程图

对于每个旅行商，每次都以一定的概率去选择下一个需要访问的城市，概率分布如公式 4-1 所示：

$$p_{rs}^{k}=\begin{cases}\tau_{rs}^{\delta}\cdot\eta_{rs}^{\beta}\Big/\sum_{z\in J_k(r)}\tau_{rz}^{\delta}\cdot\eta_{rz}^{\beta}, & s\in J_k(r)\\ 0 \quad \text{,其他} & \end{cases} \tag{4-1}$$

下面对公式中出现的符号进行解释，p_{rs}^{k}表示旅行商k在城市r选择城市s的概率，$J_k(r)$表示旅行商k在城市r还需要访问的城市集合；τ为信息素浓度，初始值一般设置为$1/|V|\cdot M$，其中$|V|$为点集的大小，M为所有边的平均长度；η是一个依赖于边(r,s)的启发值，在旅行商问题中η通常设置为城市r到城市s的路径值的倒数；δ和β

分别表示τ和η的比重，δ越大，τ的比重越大，通常设置$\delta = 1$，$\beta = 6$。通过以上公式可以构造所有旅行商的路径，当所有旅行商完成一次旅行后，更新信息素。更新规则如下：

$$\tau_{rs} \leftarrow (1-\rho)\cdot\tau_{rs}\forall(r,s) \in E \quad (4-2)$$

$$\tau_{rs} \leftarrow \tau_{rs} + \Delta\tau_{rs}\forall(r,s) \in \mathrm{T^{ib}} \quad (4-3)$$

其中，ρ表示信息素挥发因子，E为边集，ib表示当前迭代最优的旅行商的编号，T^{ib}表示当前迭代最优解的集合，$\Delta\tau_{rs}$表示ib在边(r,s)留下的信息素，且$\Delta\tau_{rs} \leftarrow 1/L^{ib}$，$L^{ib}$是$T^{ib}$的路径长度。

4.2 生成程序的执行轨迹

本文挖掘基于蚁群算法解决旅行商问题的程序规范，C++程序源代码 TSP.cpp 可参考附录 A，程序中每个函数及对应的功能如表 4.1 所示。

表4.1 TSP.cpp 中函数及对应功能

序号	函数名	功能
1	$main$	程序的入口函数
2	io	获取城市的矩阵、旅行商总数等输入信息
3	$init$	初始化所有变量的初始值
4	$judgeMaxIteration$	判断是否还需要下一次迭代
5	$reset$	重置每个旅行商的访问城市
6	$constructSolution$	构造结果
7	$selectNextCity$	选择下一个访问的城市
8	$judgeCityNumber$	判断是否还有城市没有访问
9	$updatePheromone$	更新信息素
19	$printResult$	打印最短路径及对应路径长度

目前常使用的旅行商问题标准数据集包括 TSPLIB、OR-Library 和 CSPLib 等数据集，这些数据集提供了多个旅行商问题的数据，可以用来检测 TSP 算法的效率和正确性。TSPLIB[56]是一个公开的 TSP 数据库，包含了超过 80 组 TSP 问题数据，可以有效评估 TSP 算法的性能。本文使用 TSPLIB 中的 80 组问题数据作为输入来挖掘程序的规范，共获得 80 条 trace。

为了使后续模块工作的表述更加简洁，本章的示例中设置迭代次数为 2，城市

数量为 5，城市间的距离用一个 5×5 的矩阵表示，若矩阵中元素为 0，则表示两个城市之间不相邻，矩阵数据如图 4.4 所示。将 TSP.cpp 输入 trace 生成模块，Daikon 生成的 TSP.dtrace 文件共 1596506 行，DtraceFilter 过滤后的 TSP.trace 文件共 32 行，TSP.trace 文件如图 4.3 所示。将 trace 文件输入 trace 解析模块后，得到事件集 $EventSet$如下：

$$EventSet = \{main, io, init, judgeMaxIteration, reset, constructSolution, selectNextCity, judgeCityNumber, updatePheromone, printResult\}$$

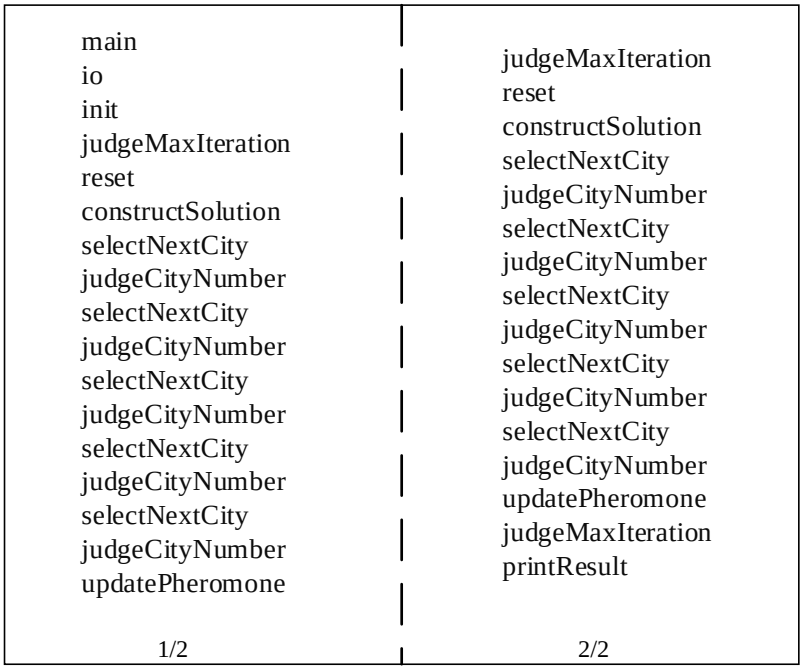


图4.3 TSP.trace 文件内容

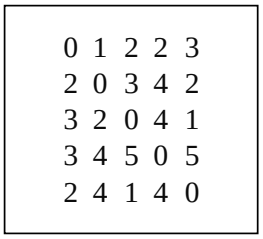


图4.4 5×5 输入矩阵

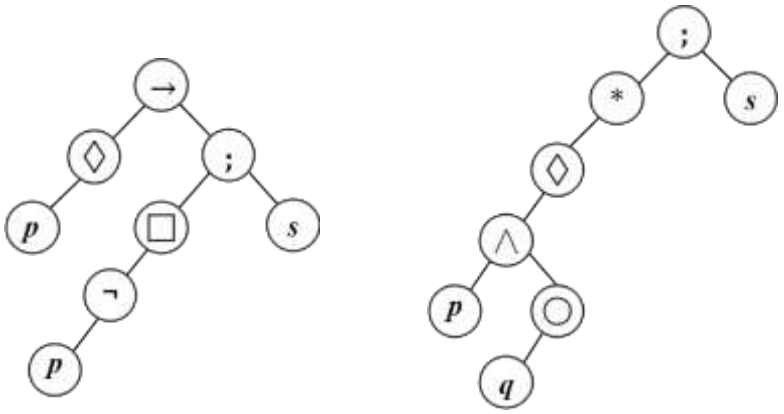
4.3 描述待验证的性质

为了进一步探究蚁群算法解决旅行商问题的流程，验证 TSP.cpp 的工作流程是否与算法设计流程图保持一致。本文挖掘程序 TSP.cpp 中函数之间的时序关系，即函数的规范。本文选择性质公式如下：

$$Formula_1 \stackrel{\text{def}}{=} \Diamond\, p \rightarrow (\Box\, \neg p; s) \qquad (4-4)$$

$$Formula_2 \stackrel{\text{def}}{=} (\Diamond\, (p \wedge \bigcirc q))^*; s \qquad (4-5)$$

其中，$Formula_1$表示原子命题s先于原子命题p发生，用来挖掘程序中各函数的先序关系。$Formula_2$表示原子命题s发生前，$\Diamond\,(p \wedge \bigcirc q)$多次发生，挖掘函数之间的循环调用关系。将$Formula_1$和$Formula_2$分别输入公式解析模块后获得的原子命题集分别是$AtomicSet_1 = \{p, s\}$和$AtomicSet_2 = \{p, q, s\}$，生成的语法树如图 4.5 中（a）、（b）所示。



(a) $Formula_1$　　(b) $Formula_2$

图4.5 性质公式对应的语法树

为了表述更加简洁，后续章节的示例围绕$Formula_2$展开。将图 4.5 中语法树（b）输入公式转化模块后获得的 LNFG 如图 4.6，其中结点 1 为开始结点，结点 5 为终止节点。LNFG 的边信息和结点信息分别如表 4.3 和表 4.2 所示，且表中边和结点的编号与图 4.6 中 LNFG 的边和结点编号一致，使用邻接表G_1存储 LNFG 的结果如图 4.7。

表4.2$Formula_2$对应 LNFG 的结点信息

结点编号	结点对应的时序公式	标记
1	$\left(\left((\Diamond\,(p \wedge \bigcirc q))^*\right) \wedge \left(fin(l_1)\right)\right); s$	l_1
2	$\left(\left(TRUE; \left((p \wedge \bigcirc q); \left((\Diamond\,(p \wedge \bigcirc q))^*\right)\right)\right) \wedge \left(fin(l_1)\right)\right); s$	l_1
3	$\left(\left(q; \left((\Diamond\,(p \wedge \bigcirc q))\right)^*\right)\right) \wedge \left(fin(l_1)\right); s$	l_1
4	$TRUE$	
5	ε	
6	$\left(\left(TRUE; \left((\Diamond\,(p \wedge \bigcirc q))^*\right)\right) \wedge \left(fin(l_1)\right)\right); s$	l_1

表4.3 $Formula_2$对应 LNFG 的边信息

序号	状态公式	所处位置	序号	状态公式	所处位置
1	$TRUE$	$N_1 \to N_2$	10	$s \wedge q \wedge l_1$	$N_3 \to N_4$
2	p	$N_1 \to N_3$	11	$s \wedge q \wedge l_1$	$N_3 \to N_5$
3	$s \wedge l_1$	$N_1 \to N_4$	12	$TRUE$	$N_4 \to N_4$
4	$s \wedge l_1$	$N_1 \to N_5$	13	$TRUE$	$N_4 \to N_5$
5	$TRUE$	$N_2 \to N_2$	14	$TRUE$	$N_6 \to N_6$
6	p	$N_2 \to N_3$	15	$TRUE$	$N_6 \to N_2$
7	q	$N_3 \to N_6$	16	p	$N_6 \to N_3$
8	q	$N_3 \to N_2$	17	$s \wedge l_1$	$N_6 \to N_4$
9	$q \wedge p$	$N_3 \to N_3$	18	$s \wedge l_1$	$N_6 \to N_5$

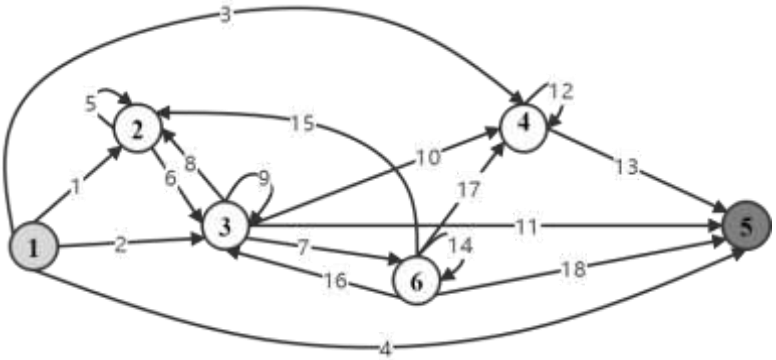


图4.6 $Formula_2$对应的 LNFG

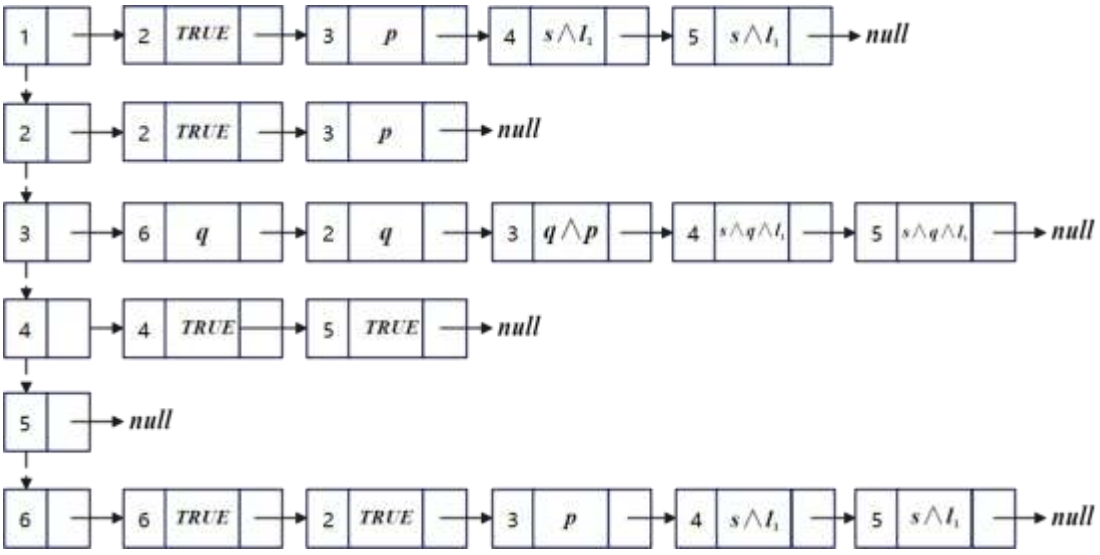


图4.7 邻接表G_1存储 LNFG

4.4 挖掘函数的规范

将$EventSet$、$AtomicSet_1$和图 4.7 中的邻接表G_1输入公式实例化模块后得到P_m^n个实例化的G_1，其中$P_m^n = m \times (m-1) \times \cdots \times (m-n+1) = 720$，$m = |EventSet| = 10$，$n = |AtomicSet| = 3$。任选一个实例化的$G_1$作为示例，记为$G_k^i$，如映射对为$((p,selectNextCity),(q,judgeCityNumber),(s,updatePheromone))$。将$G_k^i$和$\tau$输入公式验证模块挖掘规范，检测中间过程如表 4.4 所示。其中，列“SN”表示当前状态集；列“SNNext”表示下一可达状态集；列“head(trace)”表示当前事件，该列内容与 TSP.trace 文件的内容保持一致；列“检测过程”展示整个检测过程。

表4.4 检测中间过程信息表

No.	SN	检测过程	SNNext	head(trace)
1	1	$1 \xrightarrow{TRUE} 2$	2	$main$
2	2	$2 \xrightarrow{TRUE} 2$	2	io
3	2	$2 \xrightarrow{TRUE} 2$	2	$init$
4	2	$2 \xrightarrow{TRUE} 2$	2	$judgeMaxIteration$
5	2	$2 \xrightarrow{TRUE} 2$	2	$reset$
6	2	$2 \xrightarrow{TRUE} 2$	2	$constructSolution$
7	2	$2 \xrightarrow{TRUE} 2\ \ 2 \xrightarrow{selectNextCity} 3$	2,3	$selectNextCity$
8	2,3	$2 \xrightarrow{TRUE} 2\ \ 3 \xrightarrow{judgeCityNumber} 6$ $3 \xrightarrow{judgeCityNumber} 2$	2,6	$judgeCityNumber$
9	2,6	$2 \xrightarrow{TRUE} 2\ \ 2 \xrightarrow{selectNextCity} 3$ $6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2\ \ 6 \xrightarrow{selectNextCity} 3$	2,3,6	$selectNextCity$
10	2,3,6	$2 \xrightarrow{TRUE} 2\ \ 3 \xrightarrow{judgeCityNumber} 6$ $3 \xrightarrow{judgeCityNumber} 2\ 6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2$	2,6	$judgeCityNumber$
11	2,6	$2 \xrightarrow{TRUE} 2\ \ 2 \xrightarrow{selectNextCity} 3$ $6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2\ \ 6 \xrightarrow{selectNextCity} 3$	2,3,6	$selectNextCity$
12	2,3,6	$2 \xrightarrow{TRUE} 2\ \ 3 \xrightarrow{judgeCityNumber} 6$ $3 \xrightarrow{judgeCityNumber} 2\ 6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2$	2,6	$judgeCityNumber$
13	2,6	$2 \xrightarrow{TRUE} 2\ \ 2 \xrightarrow{selectNextCity} 3$ $6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2\ \ 6 \xrightarrow{selectNextCity} 3$	2,3,6	$selectNextCity$
14	2,3,6	$2 \xrightarrow{TRUE} 2\ \ 3 \xrightarrow{judgeCityNumber} 6$ $3 \xrightarrow{judgeCityNumber} 2\ 6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2$	2,6	$judgeCityNumber$

（续）表 4.4 检测中间过程信息表

No.	SN	检测过程	SNNext	head(trace)
15	2,6	$2 \xrightarrow{TRUE} 2\ \ 2 \xrightarrow{selectNextCity} 3$ $6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2\ \ 6 \xrightarrow{selectNextCity} 3$	2,3,6	$selectNextCity$
16	2,3,6	$2 \xrightarrow{TRUE} 2\ \ 3 \xrightarrow{judgeCityNumber} 6$ $3 \xrightarrow{judgeCityNumber} 2\ 6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2$	2,6	$judgeCityNumber$
17	2,6	$2 \xrightarrow{TRUE} 2\ 6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2$ $6 \xrightarrow{updatePheromone} 4\ 6 \xrightarrow{updatePheromone} 5$	2,4,5,6	$updatePheromone$
18	2,4,5,6	$2 \xrightarrow{TRUE} 2\ \ 4 \xrightarrow{TRUE} 4\ \ 4 \xrightarrow{TRUE} 5$ $6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2$	2,4,5,6	$judgeMaxIteration$
19	2,4,5,6	$2 \xrightarrow{TRUE} 2\ \ 4 \xrightarrow{TRUE} 4\ \ 4 \xrightarrow{TRUE} 5$ $6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2$	2,4,5,6	$reset$
20	2,4,5,6	$2 \xrightarrow{TRUE} 2\ \ 4 \xrightarrow{TRUE} 4\ \ 4 \xrightarrow{TRUE} 5$ $6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2$	2,4,5,6	$constructSolution$
21	2,4,5,6	$2 \xrightarrow{TRUE} 2\ \ 2 \xrightarrow{selectNextCity} 3\ \ 4 \xrightarrow{TRUE} 4$ $4 \xrightarrow{TRUE} 5\ 6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2$ $6 \xrightarrow{selectNextCity} 3$	2,3,4,5,6	$selectNextCity$
22	2,3,4,5,6	$2 \xrightarrow{TRUE} 2\ 3 \xrightarrow{judgeCityNumber} 6$ $3 \xrightarrow{judgeCityNumber} 2\ 4 \xrightarrow{TRUE} 4\ \ 4 \xrightarrow{TRUE} 5$ $6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2$	2,4,5,6	$judgeCityNumber$
23	2,4,5,6	$2 \xrightarrow{TRUE} 2\ \ 2 \xrightarrow{selectNextCity} 3\ \ 4 \xrightarrow{TRUE} 4$ $4 \xrightarrow{TRUE} 5\ 6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2$ $6 \xrightarrow{selectNextCity} 3$	2,3,4,5,6	$selectNextCity$
24	2,3,4,5,6	$2 \xrightarrow{TRUE} 2\ 3 \xrightarrow{judgeCityNumber} 6$ $3 \xrightarrow{judgeCityNumber} 2\ 4 \xrightarrow{TRUE} 4\ \ 4 \xrightarrow{TRUE} 5$ $6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2$	2,4,5,6	$judgeCityNumber$
25	2,4,5,6	$2 \xrightarrow{TRUE} 2\ \ 2 \xrightarrow{selectNextCity} 3\ \ 4 \xrightarrow{TRUE} 4$ $4 \xrightarrow{TRUE} 5\ 6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2$ $6 \xrightarrow{selectNextCity} 3$	2,3,4,5,6	$selectNextCity$
26	2,3,4,5,6	$2 \xrightarrow{TRUE} 2\ 3 \xrightarrow{judgeCityNumber} 6$ $3 \xrightarrow{judgeCityNumber} 2\ 4 \xrightarrow{TRUE} 4\ \ 4 \xrightarrow{TRUE} 5$ $6 \xrightarrow{TRUE} 6\ 6 \xrightarrow{TRUE} 2$	2,4,5,6	$judgeCityNumber$

（续）表 4.4 检测中间过程信息表

No.	SN	检测过程	SNNext	head(trace)
27	2,4,5,6	$2\xrightarrow{TRUE}2\ 2\xrightarrow{selectNextCity}3\ 4\xrightarrow{TRUE}4$ $4\xrightarrow{TRUE}5\ 6\xrightarrow{TRUE}6\ 6\xrightarrow{TRUE}2$ $6\xrightarrow{selectNextCity}3$	2,3,4,5,6	$selectNextCity$
28	2,3,4,5,6	$2\xrightarrow{TRUE}2\ 3\xrightarrow{judgeCityNumber}6$ $3\xrightarrow{judgeCityNumber}2\ 4\xrightarrow{TRUE}4\ \ 4\xrightarrow{TRUE}5$ $6\xrightarrow{TRUE}6\ 6\xrightarrow{TRUE}2$	2,4,5,6	$judgeCityNumber$
29	2,4,5,6	$2\xrightarrow{TRUE}2\ 2\xrightarrow{selectNextCity}3\ 4\xrightarrow{TRUE}4$ $4\xrightarrow{TRUE}5\ 6\xrightarrow{TRUE}6\ 6\xrightarrow{TRUE}2$ $6\xrightarrow{selectNextCity}3$	2,3,4,5,6	$selectNextCity$
30	2,3,4,5,6	$2\xrightarrow{TRUE}2\ 3\xrightarrow{judgeCityNumber}6$ $3\xrightarrow{judgeCityNumber}2\ 4\xrightarrow{TRUE}4\ \ 4\xrightarrow{TRUE}5$ $6\xrightarrow{TRUE}6\ 6\xrightarrow{TRUE}2$	2,4,5,6	$judgeCityNumber$
31	2,4,5,6	$2\xrightarrow{TRUE}2\ 4\xrightarrow{TRUE}4\ \ 4\xrightarrow{TRUE}5\ 6\xrightarrow{TRUE}6$ $6\xrightarrow{TRUE}2\ 6\xrightarrow{updatePheromone}4$ $6\xrightarrow{updatePheromone}5$	2,4,5,6	$updatePheromone$
32	2,4,5,6	$2\xrightarrow{TRUE}2\ 4\xrightarrow{TRUE}4\ \ 4\xrightarrow{TRUE}5$ $6\xrightarrow{TRUE}6\ 6\xrightarrow{TRUE}2$	2,4,5,6	$judgeMaxIteration$

若当前事件满足迁移条件时，则当前状态可经过当前事件到达下一状态，下一状态被添加到下一可达状态集。如表 4.4 中 No.9 这一行中，当前状态集 SN={2, 6}，其中状态“2”可经过迁移条件“$TRUE$”到达状态“2”，经过迁移条件“$selectNextCity$”到达状态“3”，且当前事件“$selectNextCity$”与迁移条件“$TRUE$”和“$selectNextCity$”在语义上保持一致，则下一状态“2”和“3”被加入下一可达状态集。当前状态集 SN={2, 6}中状态“6”经过迁移条件“$TRUE$”到达状态“6”和状态“2”，经过“$selectNextCity$”到达状态“3”，经过“$updatePheromone$”到达状态“4”和状态“5”，当前事件“$selectNextCity$”与迁移条件“$TRUE$”和“$selectNextCity$”在语义上保持一致，与$updatePheromone$不一致，故下一状态中状态“6”、状态“2”和状态“3”被加入下一可达状态集。综上，当前状态“2、6”经过迁移条件“$selectNextCity$”可以到达状态集“2，3，6”。由于表 4.4 的列“head（trace）”中“$judgeMaxIteration$”是最后一个当前事件，且列“SNNext”含有终止状态 ε 对应的状态编号“5”，故返回结果为$true$，即表示$\Diamond(selectNextCity \wedge \bigcirc judgeCityNumber)$多次先于$updatePheromone$事件发生。

4.5 分析挖掘结果

PPTLMiner+挖掘$Formula_1$和$Formula_2$描述的规范结果如附录 B 和附录 C 所示，分别包含 45 个、128 个规范。选择本文感兴趣的结果如表 4.5 所示，根据表中序号 No.1-9 的结果，可得到各事件的先序关系，其中每条边上的序号对应表 4.5 中的序号，表示由这条结果得出的两个事件之间的关系。根据序号 No.10 的结果可知，事件$updatePheromone$发生前，$\Diamond(selectNextCity \wedge \bigcirc judgeCityNumber)$多次发生，No.11 的结果指明，事件$updatePheromone$发生前，$\Diamond(judgeCityNumber \wedge \bigcirc selectNextCity)$多次发生，且根据序号 No.7 的结果可知$selectNextCity$先于$judgeCityNumber$发生，因此可以得出事件$judgeCityNumber$和事件$selectNextCity$存在函数调用关系。同理，根据序号 No.4-8 和 No.12-13 的结果可知，事件$printResult$发生前$\Diamond(updatePheromone \wedge \bigcirc judgeMaxIteration)$多次发生，事件$updatePheromone$和事件$judgeMaxIteration$存在函数调用关系。

表4.5感兴趣的规范

序号	规范
1	$\Diamond\, io \rightarrow (\square\, \neg io; main)$
2	$\Diamond\, init \rightarrow (\square\, \neg init; io)$
3	$\Diamond\, pjudgeMaxIteration \rightarrow (\square\, \neg judgeMaxIteration; init)$
4	$\Diamond\, reset \rightarrow (\square\, \neg reset; judgeMaxIteration)$
5	$\Diamond\, constructSolution \rightarrow (\square\, \neg constructSolution; reset)$
6	$\Diamond\, selectNextCity \rightarrow (\square\, \neg selectNextCity; constructSolution)$
7	$\Diamond\, judgeCityNumber \rightarrow (\square\, \neg judgeCityNumber; selectNextCity)$
8	$\Diamond\, updatePheromone \rightarrow (\square\, \neg updatePheromone; judgeCityNumber)$
9	$\Diamond\, printResult \rightarrow (\square\, \neg printResult; updatePheromone)$
10	$(\Diamond\,(selectNextCity \wedge \bigcirc judgeCityNumber))^*; updatePheromone$
11	$(\Diamond\,(judgeCityNumber \wedge \bigcirc selectNextCity))^*; updatePheromone$
12	$(\Diamond\,(judgeMaxIteration \wedge \bigcirc updatePheromone))^*; printResult$
13	$(\Diamond\,(updatePheromone \wedge \bigcirc judgeMaxIteration))^*; printResult$

综合表4.5的结果可以得到程序TSP.cpp中各函数之间的时序关系如图4.8所示。由图可知，函数 main、io、init、judgeMaxIteration、reset、constructSolution、selectNextCity、judgeCityNumber、updatePheromone 和 printResult 依次发生，且selectNextCity和judgeCityNumber、judgeMaxIteration和updatePheromone之间存在循

环调用的关系。对比图 4.8 和图 4.2 可知，程序函数之间调用关系的规范与算法流程图保持一致，程序的设计满足算法需求。

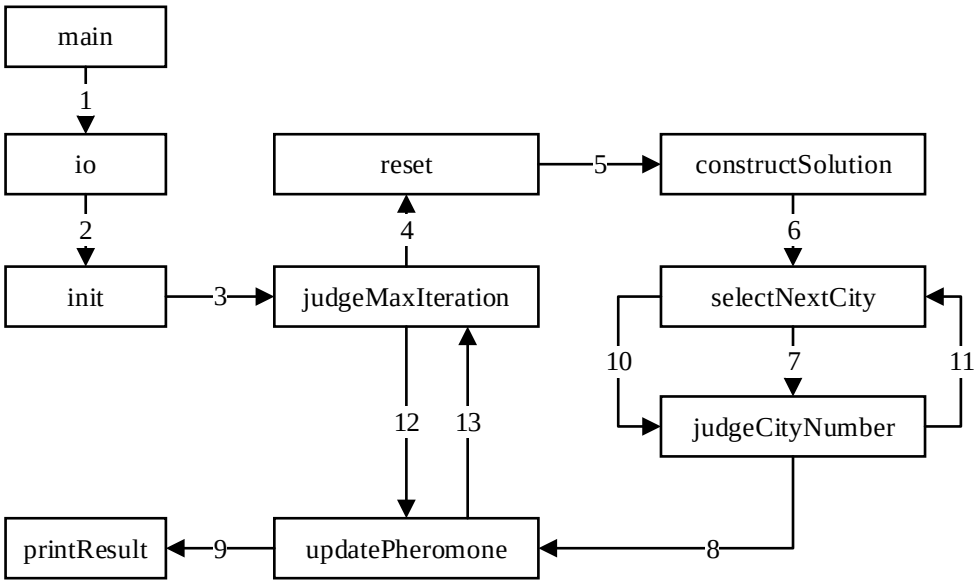


图4.8 事件关系示意图

4.6 本章小结

本章通过挖掘基于蚁群算法解决旅行商问题的规范，说明了本文提出的基于 LNFG 和 Büchi 自动机的挖掘算法和对应工具的有效性和实用性。首先介绍了蚁群算法解决旅行商问题的相关背景知识，并对此问题进行建模。接着，生成了解决旅行商问题程序 TSP.cpp 的执行轨迹文件，且通过解析 trace 获得事件集。其次，描述了待验证的两个性质，并通过解析性质公式获得原子命题集和对应的 LNFG 和 Büchi 自动机。然后，将解析后的 trace 和实例化的 LNFG 和 Büchi 自动机输入公式检测模块，从而获得程序中函数的规范。最后，分析得出规范和 TSP.cpp 的算法流程图保持一致的结论，即从形式化方法角度，证明了 TSP.cpp 符合设计预期。

第五章　对比实验与理论分析

上一章通过介绍 PPTLMiner+在基于蚁群算法解决旅行商问题中的应用，展示了挖掘工具的有效性和实用性，本章论述 PPTLMiner+相较于其他工具的优越性。本章首先通过两组对比实验介绍 PPTLMiner+在挖掘时间和占用内存两方面的优越性，然后对三个工具的挖掘过程进行理论分析。

5.1 对比试验

本节通过与 Texada[7]和 PPTLMiner[11]进行对比来分析 PPTLMiner+的优越性。对比实验共两组，每组对比实验比较三个工具能否挖掘出结果以及挖掘所用时间。两组实验使用同一个随机生成的 trace 文件，文件包含 20 条 trace，且每条 trace 长度为 10000（个事件），事件集大小为 20，trace 中每个事件形如e_i，$1 \le i \le 20$。表 5.1 给出用 LTL 公式表示性质的规范挖掘结果，PPTL 公式的规范挖掘结果如表 5.2 所示。

表5.1挖掘 LTL 公式的规范结果表

序号	性质公式	Texada		PPTLMiner+		PPTLMiner	
		时间(s)	结果	时间(s)	结果	时间(s)	结果
1	$G(x \to XFy)$	8.48901	√	6.97988	√	$+\infty$	MC
2	$\Diamond y \to (\neg y \cup x)$	4.39313	√	3.39191	√	8.58678	√
3	$G\big(x \to XG(\neg y)\big)$	851.069	√	1.52152	√	48.374	√
4	Gq	0.065196	√	0.04781	√	0.047549	√
5	$G\neg q$	0.118999	√	0.048432	√	0.052725	√
6	Fp	2.30369	√	3.90921	√	7.2862	√
7	$F(p \wedge XFp)$	2.37812	√	4.68418	√	7.39379	√
8	GFp	0.307424	√	0.124496	√	$+\infty$	MC
9	$F(p \wedge Fq)$	34.8452	√	128.186	√	180.097	√
10	$F(p \wedge q)$	1.26124	√	2.39601	√	95.532	√
11	p	1.13019	√	0.062632	√	0.090114	√
12	$p \vee q$	0.064815	√	1.38167	√	2.33304	√
13	FGp	33.1954	√	0.119401	√	3.66612	√
14	$FG(p \vee q)$	1947.12	√	8.68189	√	324.613	√
15	$G(p \to XGq)$	593.285	√	1.44637	√	3.72028	√

表5.2 挖掘 PPTL 公式的规范结果表

序号	性质公式	Texada		PPTLMiner+		PPTLMiner	
		时间(s)	结果	时间(s)	结果	时间(s)	结果
1	$true; p \wedge \varepsilon$	+∞	Fail	0.155855	√	3.83038	√
2	$true; (p \vee q) \wedge \varepsilon$	+∞	Fail	4.51631	√	108.392	√
3	$\Box\Diamond p; more$	+∞	Fail	3.6409	√	14.8525	√
4	$\Box\Diamond (p \vee q); more$	+∞	Fail	129.474	√	596.987	√
5	$p \wedge \odot\Box \neg p \vee (\Box \neg p; \bigcirc(p \wedge \odot\Box \neg p))$	+∞	Fail	0.076392	√	0.173566	√
6	$p \rightarrow (\Box p; s)$	+∞	Fail	0.936474	√	1.03313	√
7	$\Box p \rightarrow (\Box \neg p; s \wedge \bigcirc (\Diamond (t \wedge \varepsilon) \wedge \Box \neg p); true)$	+∞	Fail	1619.45	√	3107.68	√
8	$\Diamond (s \wedge \bigcirc \Diamond t) \rightarrow ((\Box \neg t \wedge \Diamond p); true)$	+∞	Fail	62.2253	√	2567.91	√
9	$(\Box p; \bigcirc s) \vee s$	+∞	Fail	1.20698	√	1.51106	√
10	$\Box p; \bigcirc \Box q$	+∞	Fail	0.946656	√	0.931485	√
11	$(\Box \neg q; p) \wedge \Box (p \rightarrow \bigcirc (\Box \neg p; q)) \wedge \Box (q \rightarrow \bigcirc (\Box \neg q \vee (\Box \neg q; p)))$	+∞	Fail	3.78014	√	267.782	√
12	$(\Box \neg q; p) \wedge \Box (p \rightarrow \bigcirc (\Box \neg p \vee (\Box \neg p; q))) \wedge \Box (q \rightarrow \bigcirc (\Box \neg q \vee (\Box \neg q; p)))$	+∞	Fail	70.0315	√	10112.3	√
13	$(\Box \neg q; p) \wedge \Box (p \rightarrow \bigcirc (q \vee (\Box \neg p; \bigcirc q)))$	+∞	Fail	1.13026	√	+∞	MC
14	$(\Box \neg q; p) \wedge \Box (p \rightarrow \bigcirc \Diamond q) \wedge \Box (q \rightarrow \odot (p \vee (\Box \neg q; \bigcirc p)))$	+∞	Fail	59.1714	√	+∞	MC
15	$(\Box \neg q; p) \wedge \Box (p \rightarrow \bigcirc (\Box \neg p; q)) \wedge \Box (q \rightarrow \bigcirc (\Box \neg q \vee (\Box \neg q; p)))$	+∞	Fail	3.51408	√	344.789	√
16	$(\Box \neg q; p) \wedge \Box (p \rightarrow \bigcirc \Diamond q)$	+∞	Fail	12.737	√	12.737	√
17	$\Box (p \rightarrow \bigcirc (q \vee (\Box \neg p; \bigcirc q)))$	+∞	Fail	1.7816	√	+∞	MC
18	$\Box (p \rightarrow \bigcirc \Diamond q) \wedge \Box (q \rightarrow \odot (p \vee (\Box \neg q; \bigcirc p)))$	+∞	Fail	14.4416	√	+∞	MC
19	$\Box (q \rightarrow \odot (p \vee (\Box \neg q; \bigcirc p)))$	+∞	Fail	1.0878	√	+∞	MC

对表 5.1 和表 5.2 中的数据作如下说明：

（1） 实验设备为：Ubuntu 18.04.5 LTS，内存：7.7GB，处理器：Intel® Core™ i7-9750H CPU @ 2.60GHz。

（2） PPTLMiner+包含两种挖掘算法，表 5.1 中 PPTLMiner+采用基于 Büchi 自动机的规范挖掘算法，表 5.2 中采用基于 LNFG 的规范挖掘算法。

（3） 列“性质公式”列举了对比实验中挖掘的性质公式，其中表 5.1 中编号 No.1-3 的性质选自挖掘工具 Synoptic [33]，No.4-15 选自模式库[10]，表 5.2 中所有性质均选自模式库。模式库是在大量论文文献中整理得到的一个性质描述库，包含了发生顺序性质库（Order）和发生次数性质库（Occurrence），且性质均用 PPTL 公式的语法表示。

（4） 列“PPTLMiner+”、“Texada”和“PPTLMiner”列举挖掘时间和挖掘结果。“+∞”表示挖掘时间无穷大；“√”表示能挖掘出结果；“MC”表示内存崩溃（Memory Corruption）；“Fail”表示不能挖掘此性质。

（5） 不同工具挖掘同一性质，若列“结果”的内容为“√”，不仅表示能挖掘出结果，还表示挖掘结果正确且一致。

（6） 统计时间代码如下所示：

```
1. start = clock();
2. //PPTLMiner+ or PPTLMiner or Texada is working
3. end = clock();
4. double endtime = (double)(end-start)/CLOCKS_PER_SEC;
```

观察表 5.1 和表 5.2 中的数据，对每个工具的挖掘时间和挖掘结果进行纵向分析，并对同一个性质的不同挖掘工具进行横向分析，分析结果如下：

由表 5.1 可知，Texada 和 PPTLMiner+能挖掘所有性质的结果，而 PPTLMiner 存在挖掘时间无穷大和内存崩溃的问题。PPTLMiner 挖掘过程主要是首先将 PPTL 公式的语法树转成范式形式；然后将范式中的状态公式与 trace 中的事件进行对比，若一致则向下遍历，反之则向上回溯；不断重复对比和向下遍历或向上回溯的过程直至最终检测结束。因此， PPTLMiner 会出现递归过深而导致内存崩溃的问题，也会出现重复回溯和遍历的过程而导致挖掘时间过长的问题。

由表 5.1 可知，在大多数情况下，PPTLMiner+相对于 Texada 用时较少，如结果 No.3、No.14 和 No.15，Texada 的挖掘用时甚至是 PPTLMiner+所用时间的几百倍。Texada 首先使用 SPOT 库将 LTL 公式转为语法树形式，再根据语法树结点的语义进行检测，因此也会出现重复转为语法树的过程，这一过程也浪费了大量的时间。而 PPTLMiner+一次性将性质公式转为 LNFG 或 Büchi 自动机，用时较少。

由表 5.2 可知，PPTLMiner+和 PPTLMiner 都能挖掘出性质 No.2-7 和 No.9-15 的结果，但在大多数情况下 PPTLMiner+用时比 PPTLMiner 少。PPTLMiner 在挖掘中会出现将大量相同或相似的 PPTL 公式重复转为范式的过程，因此浪费大量时间。相同的公式指的是范式形式中下一状态P_i'通常包含的 PPTL 公式相同。相似的公式指的是不同的实例化公式的语法树中操作符结点类型相同而原子命题对应的事件不同。

由表 5.1 和表 5.2 可知，PPTLMiner 和 PPTLMiner+不仅能挖掘 LTL 公式和 PPTL 公式描述的性质，而 Texada 由于表达能力有限只能挖掘 LTL 公式描述的性质。

5.2 理论分析

由于 PPTLMiner+、Texada 和 PPTLMiner 在公式实例化模块之前的步骤相同，因此本节只分析公式实例化模块和公式验证模块的算法在时间和内存方面的表现。Texada 和 PPTLMiner 的检测算法与深度优先遍历算法（DFS）算法类似，时间复杂度表达式如式 5-1 所示。PPTLMiner+的检测算法类似于广度优先遍历算法（BFS），时间复杂度表达式如式 5-2 所示。

$$O(T_1) = O\left(P_n^m \times L_{trace} \times (t_1 + (t_2 + t_3) \times s)\right) \quad (5-1)$$

$$O(T_2) = O\left(t_4 + P_n^m \times \left(t_5 + L_{trace} \times (t_6 \times L_{list})\right)\right) \quad (5-2)$$

对公式 5-1、公式 5-2 中出现的符号作以下说明：

（1） t_1：实例化公式的语法树形式所需时间。

（2） t_2：对于 Texada，t_2表示识别语法树结点类型所需时间；对于 PPTLMiner，t_2表示将 PPTL 公式的语法树形式转为对应的范式形式所需时间。

（3） t_3：对于 Texada，t_3表示检测事件与结点在语义上是否一致所需时间；对于 PPTLMiner，t_3表示检测事件是否与范式中的状态公式一致所需时间。

（4） s：挖掘算法运行时的分支数，如对于 PPTLMiner 来说，s的数量级与范式形式“$P_e \wedge \varepsilon \vee \bigvee_{i=1}^{r}(P_i \wedge \bigcirc P_i')$”中的$r$一致。

（5） t_4：将 PPTL 公式转为对应的 LNFG 形式或将 LTL 公式转为对应的 Büchi 自动机所需时间。

（6） t_5：实例化 LNFG 或 Büchi 自动机所需时间。

（7） t_6：检测下一状态是否可达所需时间。

（8） L_{list}：L_{list}大小与公式转换的 LNFG 或 Büchi 自动机的边集大小有关。

（9） P_n^m：表示实例化过程中有效映射对的数量。

（10）L_{trace}：表示一条 trace 的长度。

分析公式 5-1、公式 5-2，得出结论如下：

由公式 5-1 可知，Texada 会重复地判断语法树的结点类型，PPTLMiner 会重复将 PPTL 公式转为范式形式，这些重复过程大多是类似的，因此将浪费大量时间，Texada 和 PPTLMiner 的挖掘时间与挖掘算法在运行时的分支数有关。

PPTLMiner+一次性将 PPTL 公式转为 LNFG 形式，或将 LTL 公式转为 Büchi 自动机形式，不存在大量重复转换的过程。PPTLMiner+的挖掘时间与 LNFG 或 Büchi 自动机的边集大小关系较大。

下面对三个工具的空间复杂度进行简单分析。Texada 和 PPTLMiner 的公式验证算法在实现时使用栈（Stack）作为基本数据结构，通过出栈和入栈的操作来实现向上回溯和向下遍历的检测过程。当L_{trace}值越来越大时，容易出现栈溢出而导致段错误的情况。因此，Texada 和 PPTLMiner 更容易出现内存崩溃的情况。PPTLMiner+使用队列（List）作为基本数据结构，通过入队和出队的操作，从而实现公式验证模块依次遍历每个结点的下一可达结点集的检测过程。因此，当L_{list}值越来越大时，PPTLMiner+需要遍历的结点个数也会随之增多，挖掘所用的时间也会随之增大。

5.3 本章小结

本章将 PPTLMiner+与 Texada 和 PPTLMiner 进行了两组对比实验，对比得出，相对于 Texada，PPTLMiner+表达能力更强，在大多数情况下，挖掘时间更短。相对于 PPTLMiner，PPTLMiner+没有出现内存崩溃的状况，且挖掘时间更短。本章对三个挖掘工具检测算法的时间复杂度公式进行了推导，也分析了三个工具的检测算法所使用的基本数据结构，从理论角度证明了本文提出的挖掘算法的高效性和优越性。

第六章　总结和展望

本章对本文的工作进行总结，介绍了本文的主要内容和主要贡献。然后对本文的挖掘算法和挖掘工具存在的不足和未来的工作方向进行了展望。

6.1 总结

规范是系统需求的形式化描述，是形式化验证领域的一个关键点。程序行为的规范和推理已经被广泛研究，规范挖掘是一种自动得出系统规范的机器学习方法。本文首先介绍了规范挖掘的背景与意义，阐述了时序逻辑语言和规范挖掘的国内外研究现状，总结了当前规范挖掘算法和工具存在的一些问题，并以此展开本文的研究。其次，对本文相关理论基础和相关技术的研究成果进行了简要介绍，理论研究包括命题投影时序逻辑 PPTL 和线性时序逻辑 LTL，以及对应的范式 NF、带标记的范式图 LNFG 和 Büchi 自动机；技术研究包括程序不变量挖掘技术、PPTL 转 LNFG 技术和 LTL 转 Büchi 自动机技术。接着，提出了基于 LNFG 和 Büchi 自动机的规范挖掘算法，设计实现了基于该算法的规范挖掘工具 PPTLMiner+，并分别从整体框架和具体实现两方面进行了介绍，同时展示了大量的伪代码和算法流程图来辅助介绍具体实现的过程。此外，介绍了挖掘工具 PPTLMiner+的安装与使用方法，并且列举了具体的示例来介绍 PPTLMiner+的基础功能和特色功能。然后通过基于蚁群算法解决旅行商问题的规范挖掘，进一步介绍了 PPTLMiner+的工作过程，表明了本文提出算法的可用性和实用性。最后通过与 PPTLMiner 和 Texada 的对比实验和理论分析，表明了本文提出的规范挖掘算法和工具的优越性。

本文主要贡献如下：

（1）本文提出了一种挖掘 LTL 规范和 PPTL 规范的算法。PPTL 是一个完全正则的性质描述语言，能够描述有穷区间和无穷区间的性质。相对于现有规范挖掘算法常用的性质描述语言 LTL，本文使用的 PPTL 能够描述区间相关和周期重复性质，能够更全面地描述软硬件系统的性质，以便更详尽地了解和测试系统的功能，有助于检测和修复系统故障等问题的解决。

（2）本文提出了一种基于 LNFG 和 Büchi 自动机的规范挖掘算法，并开发了相应的挖掘工具 PPTLMiner+。现有的 PPTL 挖掘工具 PPTLMiner 将 PPTL 转化成范式形式，然后根据范式中状态公式的语义进行挖掘，在实现时出现将大量相同的 PPTL 公式重复转为范式的情况，造成时间的浪费。且该工具采用了一种类似于深度遍历的递归算法，当输入的程序执行轨迹过长或程序中函数过多时，挖掘算法很容易发

生内存崩溃。而 PPTLMiner+将 PPTL 公式一次性转为对应的 LNFG，或将 LTL 公式转为 Büchi 自动机，节省了大量时间，而且挖掘过程采用了类似广度优先遍历的算法，不易出现内存崩溃的情况。

（3）本文进行了挖掘工具 PPTLMiner+与现有挖掘工具 PPTLMiner 和 Texada 的对比实验，分析了三个工具在性质表达能力和挖掘效率的表现。实验表明，PPTLMiner+相对于 PPTLMiner 而言，没有出现内存崩溃的情况，并且挖掘时间有明显减少。相较于 Texada 工具，PPTLMiner+除了能够挖掘 Texada 所能挖掘的性质外，还能够挖掘完全正则的性质，表达能力更强。并且在对比实验中，PPTLMiner+在大多数情况下挖掘时间比 Texada 要少。本文还进行了理论分析，从时间复杂度角度分析 PPTLMiner+相对于 PPTLMiner 和 Texada 的优越性，并且通过三个工具在算法实现时使用的数据结构来分析其空间复杂度。

（4）本文开发的挖掘工具 PPTLMiner+具有更多丰富的特色功能，更能满足用户的需求。PPTLMiner+能够在挖掘时忽略注释行，从而使结果更准确。PPTLMiner+通过指定感兴趣的事件，从而限定挖掘结果，使其更加精炼简洁。PPTLMiner+能挖掘使用正则表达式表示的事件，也能挖掘多原子命题表示的事件。本文还挖掘了基于蚁群算法解决旅行商问题的规范，经分析，规范表示的函数之间的线性关系与算法流程一致，这一应用也体现了本文挖掘算法和工具的有效性和实用性。

6.2 展望

本文提出了基于 LNFG 和 Büchi 自动机的规范挖掘算法，并开发了对应的挖掘工具，能够有效改善现有挖掘工具表达能力有限、挖掘时间过长和内存崩溃的问题，实验结果也满足预期，但仍然存在不足有待改进：

（1）本文使用的轨迹过滤工具 DtracFilter 能够有效过滤出不变量文件中函数的轨迹，但是无法得到函数内部变量的轨迹。在后续的工作中，需要深入研究推导函数内部变量的轨迹，同时阅读 Daikon 的《开发者文档》，设计能够过滤出函数内部变量信息的方案。

（2）本文依赖 Daikon 挖掘程序的执行轨迹，但 Daikon 挖掘程序的语言有限，只能挖掘 C\C++、C#、Java、F#、Visual Basic、Perl 和 Eiffel 编写的程序。作为在数据分析、人工智能、云计算等领域广泛使用的一门语言，Daikon 却无法挖掘 Python 书写的程序规范。在接下来的工作中，可以从两个角度改善这个问题：（a）扩展 Daikon 的功能使之能够挖掘 Python 程序的轨迹；（b）研究并追踪 Python 语言函数及变量的执行轨迹，完善相应的规范挖掘方案。

（3）观察第四章中检测过程信息表 4.4 可知，在实际的挖掘过程中，会出现相同的当前状态集和当前事件，对于这种情况，检测得到的下一可达状态集也必定相同。在后续的算法改进中，希望以此为突破口，存储挖掘过程中形如“当前状态集-当前事件-下一可达状态集”的中间结果，以达到减少挖掘时间的目的。

（4）当前几乎所有的计算机都支持多线程并发执行，然而，本文开发的挖掘工具 PPTLMiner+只支持串行执行，导致 CPU 利用率过低，挖掘速度有待提高。在接下来的工作中，将研究把规范挖掘技术与多线程同步技术进行融合，提高 CPU 利用率，提出更为高效的规范挖掘方案。

附录 A

TSP.cpp 源程序

```
#include<bits/stdc++.h>
using namespace std;
const int INF = 0x3f3f3f3f;
#define sqr(x) ((x)*(x))
#define eps 1e-8
//variables
string file_name;
int type;// type == 1 全矩阵, type == 2 二维欧拉距离
int N;//城市数量
double **dis;//城市间距离
double **pheromone;//信息素
double **herustic;//启发式值
double **info;// info = pheromone ^ delta * herustic ^ beta
double pheromone_0;//pheromone 初始值，这里是 1 / (avg * N)其中 avg 为图网中所有边边权的平均数。
int m;//种群数量
int delta, beta;//参数
double alpha;
int *r1, *s, *r;//agent k 的出发城市，下一个点，当前点。
int MAX, iteration;//最大迭代次数，迭代计数变量
set<int> empty, *J;
struct vertex {
    double x, y;// 城市坐标
    int id;// 城市编号
    int input(FILE *fp) {
        return fscanf(fp, "%d %lf %lf", &id, &x, &y);
    }
}*node;

typedef pair<int, int> pair_int;
struct Tour {//路径
    vector<pair_int> path;//path[i]，存储一条边(r,s)
    double L;
    void clean() {
        L = INF;
        path.clear();
        path.shrink_to_fit();
    }//清空
    void calc() {
        L = 0;
        int sz = path.size();
        for (int i = 0; i < sz; i++) {
            L += dis[path[i].first][path[i].second];
        }
    }//计算长度
    void push_back(int x, int y) {
        path.push_back(make_pair(x, y));
```

```
    }
    int size() {
        return (int)path.size();
    }
    int r(int i) {
        return path[i].first;
    }
    int s(int i) {
        return path[i].second;
    }
    void print(FILE *fp) {
        int sz = path.size();
        for (int i = 0; i < sz; i++) {
            fprintf(fp, "%d->", path[i].first + 1);
        }
        fprintf(fp, "%d\n", path[sz - 1].second + 1);
    }
    bool operator <(const Tour &a)const {
        return L < a.L;
    }//重载
} *tour, best_so_far;

double EUC_2D(const vertex &a, const vertex &b) {
    return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
}

void io() {//输入
    printf("io\n");
    printf("input file_name and data type\n");
    cin >> file_name >> type;
    FILE *fp = fopen(file_name.c_str(), "r");
    fscanf(fp, "%d", &N);
    node = new vertex[N + 5];
    dis = new double*[N + 5];
    double tmp = 0;
    int cnt = 0;
    if (type == 1) {
        for (int i = 0; i < N; i++) {
            dis[i] = new double[N];
            for (int j = 0; j < N; j++) {
                fscanf(fp, "%lf", &dis[i][j]);
                tmp += i != j ? dis[i][j] : 0;// i == j 的时候 dis 不存在，所以不考虑。
                cnt += i != j ? 1 : 0;// i == j 的时候 dis 不存在，所以不考虑。
            }
        }
    }
    else {
        for (int i = 0; i < N; i++)
            node[i].input(fp);
        for (int i = 0; i < N; i++) {
            dis[i] = new double[N];
            for (int j = 0; j < N; j++) {
                dis[i][j] = EUC_2D(node[i], node[j]);// 计算距离
                tmp += i != j ? dis[i][j] : 0;// i == j 的时候 dis 不存在，所以不考虑。
                cnt += i != j ? 1 : 0;// i == j 的时候 dis 不存在，所以不考虑。
            }
```

```
        }
    }
    pheromone_0 = (double)cnt / (tmp * N);//pheromone 初始值，这里是 1 / (avg * N)其中 avg 为图网中所有边边权的平均数。
    fclose(fp);
    return;
}

void init() {//初始化
    alpha = 0.1;//evaporation parameter，挥发参数，每次信息素要挥发的量
    delta = 1;
    beta = 6;// delta 和 beta 分别表示 pheromones 和 herustic 的比重
    m = N;
    pheromone = new double*[N + 5];
    herustic = new double*[N + 5];
    info = new double*[N + 5];
    r1 = new int[N + 5];
    r = new int[N + 5];
    s = new int[N + 5];
    J = new set<int>[N + 5];
    empty.clear();
    for (int i = 0; i < N; i++) {
        pheromone[i] = new double[N + 5];
        herustic[i] = new double[N + 5];
        info[i] = new double[N + 5];
        empty.insert(i);
        for (int j = 0; j < N; j++) {
            pheromone[i][j] = pheromone_0;
            herustic[i][j] = 1/(dis[i][j] + eps);//加一个小数 eps，防止被零除
        }
    }
    best_so_far.clean();
    iteration = 0;
    MAX = N * N;
}

double power(double x, int y) {//快速幂，计算 x ^ y，时间复杂度 O(logn)
    double ans = 1;
    while (y) {
        if (y & 1) ans *= x;
        x *= x;
        y >>= 1;
    }
    return ans;
}

void reset() {
    tour = new Tour[m + 5];
    for (int i = 0; i < N; i++) {
        tour[i].clean();
        r1[i] = i;//初始化出发城市，
        J[i] = empty;
        J[i].erase(r1[i]);//初始化 agent i 需要访问的城市
        r[i] = r1[i];//当前在出发点
    }
```

```
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++) {
            info[i][j] = power(pheromone[i][j], delta) * power(herustic[i][j], beta);
        }//选择公式
}

int selectNextCity() {
    printf("selectNextCity\n");
    int  next = 0;
    for (int k = 0; k < m; k++) {
        if (J[k].empty()) {//如果 J 是空的，那么返回出发点城市
            next = r1[k];
        }
        else {
            double rnd = (double)(rand()) / (double)RAND_MAX;//产生 0..1 的随机数
            set<int>::iterator it = J[k].begin();
            double sum_prob = 0, sum = 0;
            for (; it != J[k].end(); it++) {
                sum += info[r[k]][*it];
            }//计算概率分布
            rnd *= sum;
            it = J[k].begin();
            for (; it != J[k].end(); it++) {
                sum_prob += info[r[k]][*it];
                if (sum_prob >= rnd) {
                    next = *it;
                    break;
                }
            }//依照概率选取下一步城市
        }
        J[k].erase(next);
        s[k] = next;
        tour[k].push_back(r[k], s[k]);
        r[k] = s[k];
    }
}

bool judgeCityNumber(int cityNum) {
    printf("judgeCityNumber\n");
    if (cityNum < N)
        return true;
    else
        return false;
}

void constructSolution() {
    printf("constructSolution\n");
    int cityNum = 1;
    selectNextCity();//选择下一步的最优决策
    while (judgeCityNumber(cityNum)) {//判断是否还有城市可选
        selectNextCity();
        cityNum++;
    }
}

void updatePheromone() {
```

```
    Tour now_best;
    now_best.clean();//初始化
    for (int i = 0; i < m; i++) {
        tour[i].calc();
        if (tour[i] < now_best)
            now_best = tour[i];//寻找当前迭代最优解
    }
    if (now_best < best_so_far) {
        best_so_far = now_best;//更新最优解
    }
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            pheromone[i][j] *= (1 - alpha);//信息素挥发
    int sz = now_best.size();
    for (int i = 0; i < sz; i++) {
        pheromone[now_best.r(i)][now_best.s(i)] += 1. / (double)now_best.L;
        pheromone[now_best.s(i)][now_best.r(i)] = pheromone[now_best.r(i)][now_best.s(i)];// 对称
    }//更新信息素含量
}

void printResult() {
    printf("best_so_far = %.2lf\n", best_so_far.L);//输出目标值
    best_so_far.print(stdout);//输出路径
}
bool judgeMaxIteration(){
    if(iteration<MAX){//论文中指定 MAX=2
        return true;
    }else{
        return false;
    }
}
int main() {
    srand((unsigned)time(0));//初始化随机种子
    io();
    init();
    double last = INF;
    int bad_times = 0;
    while(judgeMaxIteration()){
        iteration++;
     if (bad_times > N) break;//进入局部最优
        reset();//初始化 agent 信息
        constructSolution();//对于所有的 agent 构造一个完整的 tour
        updatePheromone();//更新信息素
        //printf("iteration %d:best_so_far = %.2lf\n", iteration, best_so_far.L);
        if (last > best_so_far.L)
            last = best_so_far.L, bad_times = 0;
        else bad_times++;//记录当前未更新代数，若迭代多次未更新，认为进入局部最优
    }
    printResult();
}
```

附录 B

No.	挖掘$Formula_1$的所有结果
1	$\Diamond\ constructSolution \rightarrow (\Box\ \neg constructSolution; init)$
2	$\Diamond\ constructSolution \rightarrow (\Box\ \neg constructSolution; io)$
3	$\Diamond\ constructSolution \rightarrow (\Box\ \neg constructSolution; judgeMaxIteration)$
4	$\Diamond\ constructSolution \rightarrow (\Box\ \neg constructSolution; main)$
5	$\Diamond\ constructSolution \rightarrow (\Box\ \neg constructSolution; reset)$
6	$\Diamond\ init \rightarrow (\Box\ \neg init; io)$
7	$\Diamond\ init \rightarrow (\Box\ \neg init; main)$
8	$\Diamond\ io \rightarrow (\Box\ \neg io; main)$
9	$\Diamond\ judgeCityNumber \rightarrow (\Box\ \neg judgeCityNumber; constructSolution)$
10	$\Diamond\ judgeCityNumber \rightarrow (\Box\ \neg judgeCityNumber; init)$
11	$\Diamond\ judgeCityNumber \rightarrow (\Box\ \neg judgeCityNumber; io)$
12	$\Diamond\ judgeCityNumber \rightarrow (\Box\ \neg judgeCityNumber; judgeMaxIteration)$
13	$\Diamond\ judgeCityNumber \rightarrow (\Box\ \neg judgeCityNumber; main)$
14	$\Diamond\ judgeCityNumber \rightarrow (\Box\ \neg judgeCityNumber; reset)$
15	$\Diamond\ judgeCityNumber \rightarrow (\Box\ \neg judgeCityNumber; selectNextCity)$
16	$\Diamond\ judgeMaxIteration \rightarrow (\Box\ \neg judgeMaxIteration; init)$
17	$\Diamond\ judgeMaxIteration \rightarrow (\Box\ \neg judgeMaxIteration; io)$
18	$\Diamond\ judgeMaxIteration \rightarrow (\Box\ \neg judgeMaxIteration; main)$
19	$\Diamond\ printResult \rightarrow (\Box\ \neg printResult; constructSolution)$
20	$\Diamond\ printResult \rightarrow (\Box\ \neg printResult; init)$
21	$\Diamond\ printResult \rightarrow (\Box\ \neg printResult; io)$
22	$\Diamond\ printResult \rightarrow (\Box\ \neg printResult; judgeCityNumber)$
23	$\Diamond\ printResult \rightarrow (\Box\ \neg printResult; judgeMaxIteration)$
24	$\Diamond\ printResult \rightarrow (\Box\ \neg printResult; main)$
25	$\Diamond\ printResult \rightarrow (\Box\ \neg printResult; reset)$
26	$\Diamond\ printResult \rightarrow (\Box\ \neg printResult; selectNextCity)$
27	$\Diamond\ printResult \rightarrow (\Box\ \neg printResult; updatePheromone)$
28	$\Diamond\ reset \rightarrow (\Box\ \neg reset; init)$
29	$\Diamond\ reset \rightarrow (\Box\ \neg reset; io)$

续表

No.	挖掘$Formula_1$的所有结果
30	$\Diamond\, reset \rightarrow (\Box\, \neg reset; judgeMaxIteration)$
31	$\Diamond\, reset \rightarrow (\Box\, \neg reset; main)$
32	$\Diamond\, selectNextCity \rightarrow (\Box\, \neg selectNextCity; constructSolution)$
33	$\Diamond\, selectNextCity \rightarrow (\Box\, \neg selectNextCity; init)$
34	$\Diamond\, selectNextCity \rightarrow (\Box\, \neg selectNextCity; io)$
35	$\Diamond\, selectNextCity \rightarrow (\Box\, \neg selectNextCity; judgeMaxIteration)$
36	$\Diamond\, selectNextCity \rightarrow (\Box\, \neg selectNextCity; main)$
37	$\Diamond\, selectNextCity \rightarrow (\Box\, \neg selectNextCity; reset)$
38	$\Diamond\, updatePheromone \rightarrow (\Box\, \neg updatePheromone; constructSolution)$
39	$\Diamond\, updatePheromone \rightarrow (\Box\, \neg updatePheromone; init)$
40	$\Diamond\, updatePheromone \rightarrow (\Box\, \neg updatePheromone; io)$
41	$\Diamond\, updatePheromone \rightarrow (\Box\, \neg updatePheromone; judgeCityNumber)$
42	$\Diamond\, updatePheromone \rightarrow (\Box\, \neg updatePheromone; judgeMaxIteration)$
43	$\Diamond\, updatePheromone \rightarrow (\Box\, \neg updatePheromone; main)$
44	$\Diamond\, updatePheromone \rightarrow (\Box\, \neg updatePheromone; reset)$
45	$\Diamond\, updatePheromone \rightarrow (\Box\, \neg updatePheromone; selectNextCity)$

附录 C

No.	挖掘$Formula_2$的所有规范
1	$(\Diamond\ (constructSolution \wedge \bigcirc init))^*; main$
2	$(\Diamond\ (constructSolution \wedge \bigcirc io))^*; main$
3	$(\Diamond\ (constructSolution \wedge \bigcirc judgeCityNumber))^*; main$
4	$(\Diamond\ (constructSolution \wedge \bigcirc judgeMaxIteration))^*; main$
5	$(\Diamond\ (constructSolution \wedge \bigcirc printResult))^*; main$
6	$(\Diamond\ (constructSolution \wedge \bigcirc reset))^*; main$
7	$(\Diamond\ (constructSolution \wedge \bigcirc selectNextCity))^*; judgeCityNumber$
8	$(\Diamond\ (constructSolution \wedge \bigcirc selectNextCity))^*; judgeMaxIteration$
9	$(\Diamond\ (constructSolution \wedge \bigcirc selectNextCity))^*; main$
10	$(\Diamond\ (constructSolution \wedge \bigcirc selectNextCity))^*; printResult$
11	$(\Diamond\ (constructSolution \wedge \bigcirc selectNextCity))^*; reset$
12	$(\Diamond\ (constructSolution \wedge \bigcirc selectNextCity))^*; updatePheromone$
13	$(\Diamond\ (constructSolution \wedge \bigcirc updatePheromone))^*; main$
14	$(\Diamond\ (init \wedge \bigcirc constructSolution))^*; main$
15	$(\Diamond\ (init \wedge \bigcirc io))^*; main$
16	$(\Diamond\ (init \wedge \bigcirc judgeCityNumber))^*; main$
17	$(\Diamond\ (init \wedge \bigcirc judgeMaxIteration))^*; constructSolution$
18	$(\Diamond\ (init \wedge \bigcirc judgeMaxIteration))^*; judgeCityNumber$
19	$(\Diamond\ (init \wedge \bigcirc judgeMaxIteration))^*; main$
20	$(\Diamond\ (init \wedge \bigcirc judgeMaxIteration))^*; printResult$
21	$(\Diamond\ (init \wedge \bigcirc judgeMaxIteration))^*; reset$
22	$(\Diamond\ (init \wedge \bigcirc judgeMaxIteration))^*; selectNextCity$
23	$(\Diamond\ (init \wedge \bigcirc judgeMaxIteration))^*; updatePheromone$
24	$(\Diamond\ (init \wedge \bigcirc printResult))^*; main$
25	$(\Diamond\ (init \wedge \bigcirc reset))^*; main$
26	$(\Diamond\ (init \wedge \bigcirc selectNextCity))^*; main$
27	$(\Diamond\ (init \wedge \bigcirc updatePheromone))^*; main$
28	$(\Diamond\ (io \wedge \bigcirc constructSolution))^*; main$
29	$(\Diamond\ (io \wedge \bigcirc init))^*; constructSolution$

续表

No.	挖掘$Formula_2$的所有规范
30	$(\Diamond\,(io \wedge \bigcirc init))^*; judgeCityNumber$
31	$(\Diamond\,(io \wedge \bigcirc init))^*; judgeMaxIteration$
32	$(\Diamond\,(io \wedge \bigcirc init))^*; main$
33	$(\Diamond\,(io \wedge \bigcirc init))^*; printResult$
34	$(\Diamond\,(io \wedge \bigcirc init))^*; reset$
35	$(\Diamond\,(io \wedge \bigcirc init))^*; selectNextCity$
36	$(\Diamond\,(io \wedge \bigcirc init))^*; updatePheromone$
37	$(\Diamond\,(io \wedge \bigcirc judgeCityNumber))^*; main$
38	$(\Diamond\,(io \wedge \bigcirc judgeMaxIteration))^*; main$
39	$(\Diamond\,(io \wedge \bigcirc printResult))^*; main$
40	$(\Diamond\,(io \wedge \bigcirc reset))^*; main$
41	$(\Diamond\,(io \wedge \bigcirc selectNextCity))^*; main$
42	$(\Diamond\,(io \wedge \bigcirc updatePheromone))^*; main$
43	$(\Diamond\,(judgeCityNumber \wedge \bigcirc constructSolution))^*; main$
44	$(\Diamond\,(judgeCityNumber \wedge \bigcirc init))^*; main$
45	$(\Diamond\,(judgeCityNumber \wedge \bigcirc io))^*; main$
46	$(\Diamond\,(judgeCityNumber \wedge \bigcirc judgeMaxIteration))^*; main$
47	$(\Diamond\,(judgeCityNumber \wedge \bigcirc printResult))^*; main$
48	$(\Diamond\,(judgeCityNumber \wedge \bigcirc reset))^*; main$
49	$(\Diamond\,(judgeCityNumber \wedge \bigcirc selectNextCity))^*; constructSolution$
50	$(\Diamond\,(judgeCityNumber \wedge \bigcirc selectNextCity))^*; judgeMaxIteration$
51	$(\Diamond\,(judgeCityNumber \wedge \bigcirc selectNextCity))^*; main$
52	$(\Diamond\,(judgeCityNumber \wedge \bigcirc selectNextCity))^*; printResult$
53	$(\Diamond\,(judgeCityNumber \wedge \bigcirc selectNextCity))^*; reset$
54	$(\Diamond\,(judgeCityNumber \wedge \bigcirc selectNextCity))^*; updatePheromone$
55	$(\Diamond\,(judgeCityNumber \wedge \bigcirc updatePheromone))^*; constructSolution$
56	$(\Diamond\,(judgeCityNumber \wedge \bigcirc updatePheromone))^*; judgeMaxIteration$
57	$(\Diamond\,(judgeCityNumber \wedge \bigcirc updatePheromone))^*; main$
58	$(\Diamond\,(judgeCityNumber \wedge \bigcirc updatePheromone))^*; printResult$
59	$(\Diamond\,(judgeCityNumber \wedge \bigcirc updatePheromone))^*; reset$
60	$(\Diamond\,(judgeCityNumber \wedge \bigcirc updatePheromone))^*; selectNextCity$

续表

No.	挖掘$Formula_2$的所有规范
61	$(\Diamond(judgeMaxIteration \wedge \bigcirc constructSolution))^*; main$
62	$(\Diamond(judgeMaxIteration \wedge \bigcirc init))^*; main$
63	$(\Diamond(judgeMaxIteration \wedge \bigcirc io))^*; main$
64	$(\Diamond(judgeMaxIteration \wedge \bigcirc judgeCityNumber))^*; main$
65	$(\Diamond(judgeMaxIteration \wedge \bigcirc printResult))^*; main$
66	$(\Diamond(judgeMaxIteration \wedge \bigcirc reset))^*; constructSolution$
67	$(\Diamond(judgeMaxIteration \wedge \bigcirc reset))^*; judgeCityNumber$
68	$(\Diamond(judgeMaxIteration \wedge \bigcirc reset))^*; main$
69	$(\Diamond(judgeMaxIteration \wedge \bigcirc reset))^*; printResult$
70	$(\Diamond(judgeMaxIteration \wedge \bigcirc reset))^*; selectNextCity$
71	$(\Diamond(judgeMaxIteration \wedge \bigcirc reset))^*; updatePheromone$
72	$(\Diamond(judgeMaxIteration \wedge \bigcirc selectNextCity))^*; main$
73	$(\Diamond(judgeMaxIteration \wedge \bigcirc updatePheromone))^*; main$
74	$(\Diamond(main \wedge \bigcirc io))^*; constructSolution$
75	$(\Diamond(main \wedge \bigcirc io))^*; init$
76	$(\Diamond(main \wedge \bigcirc io))^*; judgeCityNumber$
77	$(\Diamond(main \wedge \bigcirc io))^*; judgeMaxIteration$
78	$(\Diamond(main \wedge \bigcirc io))^*; printResult$
79	$(\Diamond(main \wedge \bigcirc io))^*; reset$
80	$(\Diamond(main \wedge \bigcirc io))^*; selectNextCity$
81	$(\Diamond(main \wedge \bigcirc io))^*; updatePheromone$
82	$(\Diamond(printResult \wedge \bigcirc constructSolution))^*; main$
83	$(\Diamond(printResult \wedge \bigcirc init))^*; main$
84	$(\Diamond(printResult \wedge \bigcirc io))^*; main$
85	$(\Diamond(printResult \wedge \bigcirc judgeCityNumber))^*; main$
86	$(\Diamond(printResult \wedge \bigcirc judgeMaxIteration))^*; main$
87	$(\Diamond(printResult \wedge \bigcirc reset))^*; main$
88	$(\Diamond(printResult \wedge \bigcirc selectNextCity))^*; main$
89	$(\Diamond(printResult \wedge \bigcirc updatePheromone))^*; main$
90	$(\Diamond(reset \wedge \bigcirc constructSolution))^*; judgeCityNumber$
91	$(\Diamond(reset \wedge \bigcirc constructSolution))^*; judgeMaxIteration$

续表

No.	挖掘$Formula_2$的所有规范
92	$(\Diamond(reset \wedge \bigcirc constructSolution))^*; main$
93	$(\Diamond(reset \wedge \bigcirc constructSolution))^*; printResult$
94	$(\Diamond(reset \wedge \bigcirc constructSolution))^*; selectNextCity$
95	$(\Diamond(reset \wedge \bigcirc constructSolution))^*; updatePheromone$
96	$(\Diamond(reset \wedge \bigcirc init))^*; main$
97	$(\Diamond(reset \wedge \bigcirc io))^*; main$
98	$(\Diamond(reset \wedge \bigcirc judgeCityNumber))^*; main$
99	$(\Diamond(reset \wedge \bigcirc judgeMaxIteration))^*; main$
100	$(\Diamond(reset \wedge \bigcirc printResult))^*; main$
101	$(\Diamond(reset \wedge \bigcirc selectNextCity))^*; main$
102	$(\Diamond(reset \wedge \bigcirc updatePheromone))^*; main$
103	$(\Diamond(selectNextCity \wedge \bigcirc constructSolution))^*; main$
104	$(\Diamond(selectNextCity \wedge \bigcirc init))^*; main$
105	$(\Diamond(selectNextCity \wedge \bigcirc io))^*; main$
106	$(\Diamond(selectNextCity \wedge \bigcirc judgeCityNumber))^*; constructSolution$
107	$(\Diamond(selectNextCity \wedge \bigcirc judgeCityNumber))^*; judgeMaxIteration$
108	$(\Diamond(selectNextCity \wedge \bigcirc judgeCityNumber))^*; main$
109	$(\Diamond(selectNextCity \wedge \bigcirc judgeCityNumber))^*; printResult$
110	$(\Diamond(selectNextCity \wedge \bigcirc judgeCityNumber))^*; reset$
111	$(\Diamond(selectNextCity \wedge \bigcirc judgeCityNumber))^*; updatePheromone$
112	$(\Diamond(selectNextCity \wedge \bigcirc judgeMaxIteration))^*; main$
113	$(\Diamond(selectNextCity \wedge \bigcirc printResult))^*; main$
114	$(\Diamond(selectNextCity \wedge \bigcirc reset))^*; main$
115	$(\Diamond(selectNextCity \wedge \bigcirc updatePheromone))^*; main$
116	$(\Diamond(updatePheromone \wedge \bigcirc constructSolution))^*; main$
117	$(\Diamond(updatePheromone \wedge \bigcirc init))^*; main$
118	$(\Diamond(updatePheromone \wedge \bigcirc io))^*; main$
119	$(\Diamond(updatePheromone \wedge \bigcirc judgeCityNumber))^*; main$
120	$(\Diamond(updatePheromone \wedge \bigcirc judgeMaxIteration))^*; constructSolution$
121	$(\Diamond(updatePheromone \wedge \bigcirc judgeMaxIteration))^*; judgeCityNumber$
122	$(\Diamond(updatePheromone \wedge \bigcirc judgeMaxIteration))^*; main$

续表

No.	挖掘$Formula_2$的所有规范
123	$(\Diamond(updatePheromone \wedge \bigcirc judgeMaxIteration))^*; printResult$
124	$(\Diamond(updatePheromone \wedge \bigcirc judgeMaxIteration))^*; reset$
125	$(\Diamond(updatePheromone \wedge \bigcirc judgeMaxIteration))^*; selectNextCity$
126	$(\Diamond(updatePheromone \wedge \bigcirc printResult))^*; main$
127	$(\Diamond(updatePheromone \wedge \bigcirc reset))^*; main$
128	$(\Diamond(updatePheromone \wedge \bigcirc selectNextCity))^*; main$

参考文献

[1] Kern C, Greenstreet MR. Formal verification in hardware design: a survey[J]. ACM Transactions on Design Automation of Electronic Systems (TODAES), 1999, 4.

[2] Shen L, Li H, Wang H, et al. Multifeature-Based Behavior of Privilege Escalation Attack Detection Method for Android Applications[J]. Mobile Information Systems, 2020, 2020(6):1-16.

[3] Ammons G, R Bodík, Larus J R. Mining Specifications[J]. Acm Sigplan Notices, 2002, 37(1):4-16.

[4] Yang J, Evans D, Bhardwaj D, et al. Perracotta: mining temporal API rules from imperfect traces[C]//Proceedings of the 28th international conference on Software engineering. 2006: 282-291.

[5] Vasudevan S, Sheridan D, Patel S, et al. Goldmine: Automatic assertion generation using data mining and static analysis[C]//2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010). IEEE, 2010: 626-629.

[6] Hangal S, Chandra N, Narayanan S, et al. Iodine: a tool to automatically infer dynamic invariants for hardware designs[C]//Proceedings of the 42nd annual Design Automation Conference. 2005: 775-778.

[7] Lemieux C, Park D, Beschastnikh I. General LTL specification mining (T)[C]//2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2015: 81-92.

[8] Wang M, Tian C, Zhang N, et al. Verifying full regular temporal properties of programs via dynamic program execution[J]. IEEE Transactions on Reliability, 2018, 68(3): 1101-1116.

[9] Cong T, Zhenhua D. Proposition projection temporal logic, Büchi automata and omega-expressions [G][C]//LNCS4978: Proc of the 5th Annual Conf Theory and Applications of Models of Comp-utation. Berlin: Springer. 2008: 47-58.

[10] Zhang N, Yuan X, Duan Z. Propositional projection temporal logic specification mining[C]//Combinatorial Optimization and Applications: 14th International Conference, COCOA 2020, Dallas, TX, USA, December 11–13, 2020, Proceedings. Cham: Springer International Publishing, 2020: 289-303.

[11] Zhang N, Yu B, Tian C, et al. Temporal logic specification mining of programs[J]. Theoretical Computer Science, 2021, 857: 29-42.

[12] Pnueli A. The temporal logic of programs[C]//18th Annual Symposium on Foundations of Computer Science (sfcs 1977). ieee, 1977: 46-57.

[13] Lamport L. The temporal logic of actions[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1994, 16(3): 872-923.

[14] Harel D, Kozen D, Parikh R. Process logic: Expressiveness, decidability, completeness[J]. Journal of computer and system sciences, 1982, 25(2): 144-170.

[15] Chandra A, Halpern J, Meyer A, et al. Equations between regular terms and an application to process logic[C]//Proceedings of the thirteenth annual ACM symposium on Theory of computing. 1981: 384-390.

[16] Moszkowski B. Compositional reasoning about projected and infinite time[C]//Proceedings of First IEEE International Conference on Engineering of Complex Computer Systems. ICECCS'95. IEEE, 1995: 238-245.

[17] Duan Z, Koutny M, Holt C. Projection in temporal logic programming[C]//Logic Programming and Automated Reasoning: 5th International Conference, LPAR'94 Kiev, Ukraine, July 16–22, 1994 Proceedings 5. Springer Berlin Heidelberg, 1994: 333-344.

[18] 田聪. 命题投影时序逻辑的判定性, 复杂性, 表达性及模型检测[D]. 西安电子科技大学, 2010.

[19] Duan Z, Tian C, Zhang L. A decision procedure for propositional projection temporal logic with infinite models[J]. Acta Informatica, 2008, 45(1): 43-78.

[20] Duan Z, Tian C, Yang M, et al. Bounded model checking for propositional projection temporal logic[C]//Computing and Combinatorics: 19th International Conference, COCOON 2013, Hangzhou, China, June 21-23, 2013. Proceedings 19. Springer Berlin Heidelberg, 2013: 591-602.

[21] Pang T, Duan Z, Tian C. Symbolic model checking for propositional projection temporal logic[C]//2012 Sixth International Symposium on Theoretical Aspects of Software Engineering. IEEE, 2012: 9-16.

[22] 张晓明. PPTL 的偏序模型检测器的实现与验证实例[D]. 西安电子科技大学, 2015.

[23] Wang X, Kou M, Li C, et al. Implementation of Theorem Prover for PPTL with Indexed Expressions[J]. Journal of Software, 2022, 33(6): 2172-2188.

[24] Shi Y, Tian C, Duan Z, et al. Model checking Petri nets with MSVL[J]. Information Sciences, 2016, 363: 274-291.

[25] Gopinath D, Malik M Z, Khurshid S. Specification-based program repair using SAT[C]//Tools and Algorithms for the Construction and Analysis of Systems: 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26–April 3, 2011. Proceedings 17. Springer Berlin Heidelberg, 2011: 173-188.

[26] Zhang M, Song F, Mallet F, et al. SMT-based bounded schedulability analysis of the clock constraint specification language[C]//Fundamental Approaches to Software Engineering: 22nd

International Conference, FASE 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings. Cham: Springer International Publishing, 2019: 61-78.

[27] Bhushan R C, Yadav D K. Formal Specification and Verification of Data Separation for Muen Separation Kernel[J]. Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science), 2022, 15(2): 274-283.

[28] Bansal S, Pal B, Ghosh K, et al. Automated Debug of Falsified Power-Aware Formal Properties using Static Checker Results: U.S. Patent Application 17/463,040[P]. 2022-3-10.

[29] Deutschbein C. Mining Secure Behavior of Hardware Designs[D]. The University of North Carolina at Chapel Hill, 2021.

[30] Yang J, Evans D, Bhardwaj D, et al. Perracotta: mining temporal API rules from imperfect traces[C]//Proceedings of the 28th international conference on Software engineering. 2006: 282-291.

[31] Van der Aalst W M P, De Beer H T, van Dongen B F. Process mining and verification of properties: An approach based on temporal logic[C]//On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, October 31-November 4, 2005, Proceedings, Part I. Springer Berlin Heidelberg, 2005: 130-147.

[32] Dwyer M B, Avrunin G S, Corbett J C. Patterns in property specifications for finite-state verification[C]//Proceedings of the 21st international conference on Software engineering. 1999: 411-420.

[33] Beschastnikh I, Brun Y, Schneider S, et al. Leveraging existing instrumentation to automatically infer invariant-constrained models[C]//Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. 2011: 267-277.

[34] Neider D, Gavran I. Learning linear temporal properties[C]//2018 Formal Methods in Computer Aided Design (FMCAD). IEEE, 2018: 1-10.

[35] Li W, Forin A, Seshia S A. Scalable specification mining for verification and diagnosis[C]//Proceedings of the 47th design automation conference. 2010: 755-760.

[36] Iyer V, Kim D, Nikolic B, et al. RTL bug localization through LTL specification mining (WIP)[C]//Proceedings of the 17th ACM-IEEE International Conference on Formal Methods and Models for System Design. 2019: 1-5.

[37] Ernst M D, Perkins J H, Guo P J, et al. The Daikon system for dynamic detection of likely invariants[J]. Science of computer programming, 2007, 69(1-3): 35-45.

[38] 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(2-3): 217-237.

[39] Wang B, Lu S, Jiang J, et al. Survey of Dynamic Analysis Based Program Invariant Synthesis Techniques[J]. Journal of Software, 2020, 31(6): 1681-1702.

[40] Hangal S, Lam M S. Tracking down software bugs using automatic anomaly detection[C]//Proceedings of the 24th international conference on Software engineering. 2002: 291-301.

[41] Nguyen T, Kapur D, Weimer W, et al. DIG: A dynamic invariant generator for polynomial and array invariants[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2014, 23(4): 1-30.

[42] Antopoulos T, Ruef A, Hicks M. A Counterexample-guided Approach to Finding Numerical Invariants[J]. 2016.

[43] Garg P, Löding C, Madhusudan P, et al. ICE: A robust framework for learning invariants[C]//Computer Aided Verification: 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings 26. Springer International Publishing, 2014: 69-87.

[44] Csallner C, Tillmann N, Smaragdakis Y. DySy: Dynamic symbolic execution for invariant inference[C]//Proceedings of the 30th international conference on Software engineering. 2008: 281-290.

[45] Kannan Y, Sen K. Universal symbolic execution and its application to likely data structure invariant generation[C]//Proceedings of the 2008 international symposium on Software testing and analysis. 2008: 283-294.

[46] Wolper P, Vardi M Y, Sistla A P. Reasoning about infinite computation paths[C]//24th Annual Symposium on Foundations of Computer Science (sfcs 1983). IEEE, 1983: 185-194.

[47] Vardi M Y, Wolper P. Automata theoretic techniques for modal logics of programs[C] //Proceedings of the sixteenth annual acm symposium on theory of computing. 1984: 446-456.

[48] Holzmann G J. The model checker SPIN[J]. IEEE Transactions on software engineering, 1997, 23(5): 279-295.

[49] Gastin P, Oddoux D. Fast LTL to Büchi automata translation[C]//Computer Aided Verification: 13th International Conference, CAV 2001 Paris, France, July 18–22, 2001 Proceedings 13. Springer Berlin Heidelberg, 2001: 53-65.

[50] Babiak T, Křetínský M, Řehák V, et al. LTL to Büchi automata translation: Fast and more deterministic[C]//Tools and Algorithms for the Construction and Analysis of Systems: 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24–April 1, 2012. Proceedings 18.

Springer Berlin Heidelberg, 2012: 95-109.

[51] Wolper P. The tableau method for temporal logic: An overview[J]. Logique et Analyse, 1985: 119-136.

[52] Duret-Lutz A, Poitrenaud D. SPOT: an extensible model checking library using transition-based generalized B ü chi automata[C]//The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings. IEEE, 2004: 76-83.

[53] Babiak T, Blahoudek F, Duret-Lutz A, et al. The Hanoi omega-automata format[C]//Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I. Cham: Springer International Publishing, 2015: 479-486.

[54] Applegate D L, Bixby R E, Chvátal V, et al. The traveling salesman problem[M]//The Traveling Salesman Problem. Princeton university press, 2011.

[55] Ning J, Zhang Q, Zhang C, et al. A best-path-updating information-guided ant colony optimization algorithm[J]. Information Sciences, 2018, 433: 142-162.

[56] Ismail A H, Hartono N, Zeybek S, et al. Using the Bees Algorithm to solve combinatorial optimisation problems for TSPLIB[C]//IOP conference series: materials science and engineering. IOP Publishing, 2020, 847(1): 012027.

[57] Miller C E, Tucker A W, Zemlin R A. Integer programming formulation of traveling salesman problems[J]. Journal of the ACM (JACM), 1960, 7(4): 326-329.