软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*, 2022,33(6):2172−2188 [doi: 10.13328/j.cnki.jos.006576]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

# 支持索引式的 PPTL 定理证明器的实现*

王小兵, 寇蒙莎, 李春奕, 赵 亮

(西安电子科技大学 计算机科学与技术学院, 陕西 西安 710071)

通信作者: 赵亮, E-mail: lzhao@xidian.edu.cn

**摘 要**: 定理证明是目前主流的形式化验证方法, 拥有强大的抽象和逻辑表达能力, 且不存在状态空间爆炸问题, 可用于有穷和无穷状态系统, 但其不能完全自动化, 并且要求用户掌握较强的数学知识. 含索引式的命题投影时序逻辑(PPTL)是一种具有完全正则表达能力, 并且包含 LTL 的时序逻辑, 具有较强的建模和性质描述能力. 目前, 一个可靠完备的含索引式的 PPTL 公理系统已被构建, 然而基于该公理系统的定理证明尚未得到良好工具的支持, 存在证明自动化程度较低以及证明冗长易错的问题. 鉴于此, 首先设计了支持索引式的 PPTL 定理证明器的实现框架, 包括公理系统的形式化与交互式定理证明; 然后, 在Coq中形式化定义了含索引式的PPTL公式、公理与推理规则, 完成了框架中公理系统的实现; 最后, 通过两个实例的交互式证明验证了该定理证明器的可用性.

**关键词**: 定理证明; Coq; 索引式; 命题投影时序逻辑; 公理系统

**中图法分类号**: TP311

中文引用格式: 王小兵, 寇蒙莎, 李春奕, 赵亮. 支持索引式的PPTL定理证明器的实现. 软件学报, 2022, 33(6): 2172–2188. http://www.jos.org.cn/1000-9825/6576.htm

英文引用格式: Wang XB, Kou MS, Li CY, Zhao L. Implementation of Theorem Prover for PPTL with Indexed Expressions. Ruan Jian Xue Bao/Journal of Software, 2022, 33(6): 2172−2188 (in Chinese). http://www.jos.org.cn/1000-9825/6576.htm

## Implementation of Theorem Prover for PPTL with Indexed Expressions

WANG Xiao-Bing, KOU Meng-Sha, LI Chun-Yi, ZHAO Liang

(School of Computer Science and Technology, Xidian University, Xi'an 710071, China)

**Abstract**: Theorem proving is a mainstream formal verification method, with a strong ability of abstraction and logical expression. It does not suffer from state space explosion and can be used to verify finite and infinite systems. Nevertheless, it cannot be fully automated and requires users to have deep mathematical knowledge. Propositional projection temporal logic with indexed expressions is a temporal logic with full regular expressiveness and subsumes LTL, having strong modeling and property describing ability. At present, a sound and complete axiom system for PPTL with indexed expressions is presented while the theorem proving based on it is not yet well supported by tools, which leads to the low automaticity, redundancy, and fallibility of theorem proving. Therefore, firstly, the implementation framework of the theorem prover for PPTL with indexed expressions is designed, including two parts, the formalization of the PPTL axiom system and interactive theorem proving. Then the formulas, axioms, and inference rules are formally defined in Coq, implementing the axiom system of the framework. Finally, the availability of the theorem prover is proved by the interactive proving of two proof examples.

**Key words**: theorem proving; Coq; indexed expressions; PPTL; axiom system

算法式的模型检测[1]和演绎式的定理证明[2]是目前两种主流的形式化验证方法. 对于数据密集型的待验证系统, 模型检测往往需要巨大甚至无限的状态空间[3]. 而定理证明方法的优势是高度的抽象能力及强有力的逻辑表达能力, 其可以利用归纳法对无限状态空间进行推理, 原理是: 在一个基于某种形式逻辑的证明系

* 基金项目: 国家自然科学基金(61672403, 61972301); 陕西省重点研发计划(2020GY-043, 2020GY-210)
本文由"定理证明理论与应用"专题特约编辑曹钦翔副教授、詹博华副研究员、赵永望教授推荐.
收稿时间: 2021-09-04; 修改时间: 2021-10-14; 采用时间: 2022-01-04; jos 在线出版时间: 2022-01-28

统中, 将待验证系统的模型与规范表示为用该形式逻辑语言描述的公式 $M$ 和 $P$, 再将两者满足的逻辑关系表示为一个有待证明的定理$\vdash M\rightarrow P$, 最后利用该证明系统中的公理和推理规则完成上述定理的逻辑推理与证明, 从而证明"系统满足其规范".

在过去的几十年中, 许多时序逻辑的公理系统被构建, 以用于定理证明. 如经典的时序逻辑 LTL (linear temporal logic)、CTL (computation tree logic)、ITL (interval temporal logic)等[4−6]; 以及对它们进行拓展得到的表达能力更强的时序逻辑: 用于运行时验证的 LTL、在 LTL 中增加了不动点算子的 $L_{\mu}^{lin}$ (linear time μ-calculus)、在 LTL 中增加了量词的 QPTL (quantified propositional linear-time temporal logic)及支持无穷时间的 PITL (propositional interval temporal logic)等[7−10]. 在 PITL 的基础上, Duan 增加了投影操作符 *prj*, 提出了命题投影时序逻辑 PPTL (propositional projection temporal logic), 其是投影时序逻辑 PTL 的命题子集, 具有完全正则的表达能力[11,12]. 之后, Zhang 等人建立了一个 PPTL 公理系统, 证明了其可靠性与完备性[13], 并成功应用于硬件描述和验证程序、通信协议[14−16]. 为进一步提高 PPTL 的表达能力, Duan 引入了索引式, 形如 $OP_{i\in\mathbb{N}}X^{i}$, 意为将操作符 $OP$ 运用于结构含自然数索引 $i$ 的结构 $X^{i}$ 无穷次, 其中, $\mathbb{N}$是自然数集合, 使其成为 LTL 的超集, 并在描述具有无限循环结构的性质时更为简洁[17]. 为了将其在定理证明方法中应用, Zhao 对 Zhang 提出的基本 PPTL 公理系统进行了拓展和完善, 建立了一个可靠完备的含索引式的 PPTL 公理系统[18]. 这些逻辑公理系统的出现与发展, 使得定理证明的应用范围不断扩大.

然而, 由于定理证明方法对使用者的专业程度要求较高, 以及自动化程度不够高等原因, 使其发展速度与程度都不及模型检测. 直到 20 世纪 60 年代, 交互式定理证明(interactive theorem proving, ITP)[19]的出现, 才极大地提高了定理证明的自动化程度. ITP 发展至今, 诞生了许多较为成熟的证明助手, Isabelle/HOL、Coq、ALC2、PVS 等[20−23]许多研究围绕着将逻辑系统与证明助手进行结合而展开.

一些工作将逻辑系统形式化于证明助手中, 利用证明助手对系统自身的相关性质进行验证. van Doorn 在 Coq 中形式化了经典命题逻辑系统, 证明了系统的可靠性和完备性, 相继式演算的切割消除定理, 自然演绎演算、希尔伯特系统、相继式演算三者之间的等价性定理[24]. Tsai 在 Coq 中实现了 CTL*中状态、路径概念的形式化定义, 并形式化了 PTL 公理系统, 以及证明了 PTL 中公理和推理规则的有效性及可靠性[5]. Moszkowski 使用 PVS 规范语言对 ITL 的语法语义进行编码, 实现了 ITL 证明系统, 并验证了 100 余个 ITL 定理. 后续又使用 Isabelle/HOL 助手, 形式化验证了 ITL 定理证明系统的可靠性和 ITL 定理库中大量定理的正确性[25].

一些工作将逻辑编程语言的公理系统形式化于证明助手中, 对特定系统, 特别是无穷状态系统进行建模与性质验证. Pnueli 在 PVS 中实现了 LTL 的验证系统, 包括一系列证明规则和策略, 验证了参数化的并行素数筛选算法的正确性[26]. 马倩首次将时序逻辑编程语言 MSVL 的语法、语义及公理系统实现于 PVS 中, 并验证了一个进程调度算法的安全性[27]. 随后, Lin 提出了一种基于公理语义的 MSVL 程序定理证明方法, 并开发了一个基于 Coq 的 MSVL 程序定理证明器, 在其中实现了两个具体实例的性质验证[28].

含索引式的 PPTL 公理系统被提出后, 基于该公理系统的定理证明尚未得到良好的工具支持, 纯手工的定理证明可能导致证明过程冗长易错. 为提高 PPTL 定理证明的自动化程度, 同时保证证明的正确性和简洁性, 开发合适的定理证明器是必要的. 针对以上问题, 本文基于 Coq 实现了支持索引式的 PPTL 定理证明器, 通过使用含索引式的 PPTL 公理系统进行半自动定理证明. 首先, 基于 Coq 的系统结构和 PPTL 公理系统设计定理证明器的实现框架; 然后, 分模块逐步使用 Coq 形式化公理系统, 实现支持索引式的 PPTL 定理证明器; 最后对两个证明实例进行半自动证明, 表明该定理证明器的可用性.

本文第 1 节对背景知识进行简要介绍, 包括交互式定理证明器 Coq 的相关知识、含索引式的 PPTL 以及含索引式的 PPTL 公理系统. 第 2 节阐述支持索引式的 PPTL 定理证明器的实现框架和具体实现方法. 第 3 节提供基本 PPTL 公式和含索引式的 PPTL 公式两个实例, 在定理证明器中进行交互式证明. 第 4 节给出总结与展望.

## 1 背景知识

### 1.1 交互式定理证明器Coq

Coq 是交互式定理证明领域内较为主流的证明助手, 可被用来开发满足规范说明的程序, 亦可被用作一个逻辑框架[21]. Coq 具有很强的扩展性, 可面向模态逻辑、时序逻辑、面向资源的逻辑等逻辑系统进行证明. 该助手拥有简洁的操作界面, 并向用户提供自动定理证明的策略, 使得定理证明以交互式的方式进行. 相较于其他证明助手, Coq 拥有很多优势, 比如基于高阶逻辑、支持归纳定义等.

Coq 之所以能够有效地对程序、协议等进行性质验证, 是由于其本身严谨的系统结构. Coq 的系统结构主要由证明开发系统和证明检查器构成, 如图 1 所示[21].

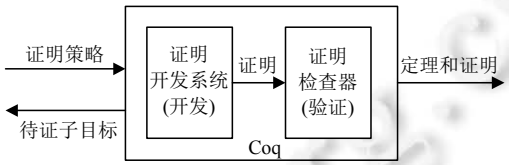


图 1 Coq 系统结构

证明开发系统是 Coq 的核心, 提供环境帮助用户开发证明, 允许用户形式化定义推理系统和逻辑系统. 证明检查器用于验证被形式化表示的待证目标, 核心是类型检查算法, 提供证明检查范围包括程序的验证、数理逻辑的自动验证和定义、公理和证明的自动验证等. 证明开发系统及证明检查器对用户自定义的推理系统等具有很强的兼容性, 降低了证明的复杂度, 保证了证明的正确性和简洁性. 在证明过程中, Coq 采用反向推理机制, 用户使用证明策略与 Coq 进行交互, 证明目标被逐步拆解成待证子目标, 直至无子目标产生时, 完成证明.

在语法上, Coq 支持 4 套不同的语言.

(1) 命令语言 Vernacular: 用来处理定义, 使用大写字母开头, 例如 Theorem、*Proof*、Qed;

(2) 规范说明语言 Gallina: 用来描述定理, 例如 **forall** *A*: *Prop*, *A*→*A*;

(3) 证明策略语言 Tactics: 用作证明过程, 以小写字母开头, 例如 *intros*、*exacts*;

(4) 证明策略定义语言 Ltac: 用于定义新的证明策略, 可结合 match with 等控制结构定义带变元的策略.

Coq 的主要特征是其基于归纳构造演算, 为用户提供归纳类型. Coq 中的归纳类型与大多函数式程序设计语言中的递归类型定义类似, 然而 Coq 可以融合递归类型和依赖积, 使得 Coq 中的归纳类型具有更强的表达能力, 甚至可被用于描述纯逻辑程序设计. 归纳类型具有两种结构: 非递归类型可以对已确定规模的数据类型进行建模; 而递归类型可以对规模可变化的数据类型进行建模, 使得 Coq 可以描述无限集合, 并实现基于归纳证明的系统推理模式和基于递归函数的系统计算模式.

另外, Coq 支持命题的描述和证明, 提供了证明策略和语境来完成证明. 语境包含声明、定义、公理、假设、定理、引理等[29]. 命题的证明分为 3 步.

- 首先, 声明命题变量, 即使用关键字 Variable 声明所需的变量;
- 其次, 证明和使用关键字描述的命题, 常用的关键字有如下 4 个: Hypothesis、Axiom、Theorem 和 Lemma. 假设 Hypothesis 和公理 Axiom 描述的命题可以在证明中直接使用, 无需证明; 定理 Theorem 和引理 Lemma 描述的命题则需要证明;
- 最后, 使用证明策略进行交互式证明.

Coq 证明的基本形式为:

Theorem *ident*: *type*.

*Proof*.

*Tactics*

Qed.

Theorem 声明了一个需要被证明的新定理; *ident* 是新定理的名称; *type* 为类型; *Proof* 是完整的定理证明执行的开始符; *Tactics* 为证明策略, 是可以运用于待证目标的命令, 能够生成一个新的目标链, 以实现反向推理, 常用的证明策略有 *intros*、*apply*、*assert*、*unfold*、*rewrite*、*assumption*、*left*/*right*、*split*、*auto* 等; Qed 命令为证明结尾符, 表示证明完成.

### 1.2 含索引式的PPTL

(1) PPTL 的语法

为保证定义的严谨性, 对符号做如下约定: 原子命题通常用小写字母(可带下标)表示, 如 $p$、$q$、$r_1$; PPTL 公式通常用大写字母(可带下标)表示, 如 $P$、$Q$、$R_1$; 可数原子命题集用 $AP$ 表示. 在此基础上, PPTL 公式可被形式化地归纳定义为下式:

$$P ::= p \mid \neg P \mid P_1 \wedge P_2 \mid \bigcirc P \mid P^{[+]} prj\ P$$
$$P^{[+]} ::= P \mid P, P^{[+]}$$

其中, $P$ 为 PPTL 公式, $P^{[+]}$表示一个有限的 PPTL 公式序列, ○ (next), *prj* (projection)为时序操作符, ¬和∧的定义与经典逻辑相同. 该定义亦可被等价表示为

$$P::=p|\neg P|P_1\wedge P_2|\bigcirc P|(P_1,\ldots,P_m)\ prj\ P.$$

(2) PPTL 的语义

设 $B$={true,false}为布尔域. 为解释上述公式, 定义状态 $s$ 为从 $AP$ 到 $B$ 的映射, $s$:$AP\to B$. 使用 $s[p]$表示在状态 $s$ 下 $p$ 的真值. 定义区间$\sigma$为一个非空的状态序列, 可表示为 $\sigma'=\langle s'_0,s'_1,...,s'_{|\sigma'|}\rangle$, 其中, $|\sigma|$为区间的长度. 若区间为有穷, $|\sigma|$的值为$\sigma$中的状态数减 1; 若区间无穷, 区间长度为无穷, 将 $p\in AP$ 表示为$|\sigma|=\omega$. 为统一对区间长度$|\sigma|$的描述, 将非负整数集 $N_0$ 扩展为 $N_\omega=N_0\cup\{\omega\}$. 在 $N_\omega$中, $\omega=\omega$, 对于$\forall i\in N_0$, 有 $i<\omega$, 将 $N_0$ 中的操作符<、≤、=扩展到 $N_\omega$中, 并定义操作符$\preceq$为$\leqslant-\{(\omega,\omega)\}$. 两个区间$\sigma=\langle s_0,s_1,\ldots,s_{|\sigma|}\rangle$和 $\sigma'=\langle s'_0,s'_1,...,s'_{|\sigma'|}\rangle$ 可进行连接, 成为一个更大的区间, 连接操作符记为 $\sigma\cdot\sigma'=\langle s_0,s_1,...,s_{|\sigma|},s'_0,s'_1,...,s'_{|\sigma'|}\rangle$, 其中, $\sigma$需为有穷区间, 且$\sigma$与$\sigma'$交集为空. 为定义投影操作符, 引入↓操作符, 若 $r_1,r_2,\ldots,r_h$ ($h$>1)为一个整数序列, 且满足 $0\leqslant r_1\leqslant\ldots\leqslant r_h\preceq|\sigma|$, 则 $\sigma\downarrow(r_1,r_2,\ldots,r_h)=\langle s_{t_1},s_{t_2},...,s_{t_l}\rangle$, 其中, $t_1,t_2,\ldots,t_l$为 $r_1,r_2,\ldots,r_h$删除重复项得到的递增序列.

一个解释是一个三元组 $I=(\sigma,k,j)$, $\sigma$为一个区间, $k$ 为非负整数, $j$ 为整数或$\omega$, 且满足 $0\leqslant k\preceq j\leqslant|\sigma|$. 在当前状态为 $s_k$时, 我们用符号$(\sigma,k,j)\models p$ 来表示在子区间$\langle s_k,\ldots,s_j\rangle$上为可解释的且可满足的. 那么对于一个 PPTL 公式, ⊨可被定义为:

$I\models p$ 当且仅当对于任意给定的命题 $p$, 满足 $s_k[p]$=true;

$I\models\neg P$ 当且仅当 $I\not\models P$;

$I\models P_1\wedge P_2$ 当且仅当 $I\models P_1$ 并且 $I\models P_2$;

$I\models\bigcirc P$ 当且仅当 $k<j$, 且$(\sigma,k+1,j)\models P$;

$I\models(P_1,\ldots,P_m)prj\ P$ 当且仅当存在整数 $k=r_0\leqslant r_1\leqslant\ldots\preceq r_m\leqslant j$,
使得对于所有 $1\leqslant l\leqslant m$, 有$(\sigma,r_{l-1},r_l)\models P_l$
对于下列两种$\sigma'$之一, 满足$(\sigma',0,|\sigma|)\models P$.
(1) $r_m<j$ 且$\sigma'=\sigma\downarrow(r_0,\ldots,r_m)\cdot\sigma(r_{m+1},\ldots,j)$;
(2) $r_m=j$ 且$\sigma'=\sigma\downarrow(r_0,\ldots,r_h)$, $(0\leqslant h\leqslant m)$.

一个公式 $P$ 被一个区间$\sigma$满足, 或者$\sigma$是 $P$ 的一个模型, 用$\sigma\models P$ 表示. 如果$(\sigma,0,|\sigma|)\models P$, 称一个公式 $P$ 是可满足的. 如果$\sigma\models P$ 对于某些$\sigma$是可满足的, 则 $P$ 至少有一个模型; 否则, 称 $P$ 是不可满足的. 对于所有区间$\sigma$有$\sigma\models P$ 成立, 称一个公式 $P$ 是有效的, 用$\models P$ 表示.

(3) PPTL 导出公式

定义 PPTL 的部分导出公式如下所示, 有助于简化时序性质的描述.

$$\begin{array}{llll}
\varepsilon & \overset{def}{=}\neg\bigcirc tt & P;Q & \overset{def}{=}(P,Q)\ prj\ \varepsilon \\
\Diamond P & \overset{def}{=}tt;P & \Box P & \overset{def}{=}\neg\Diamond\neg P \\
P^{+} & \overset{def}{=}(P^{[+]})\ prj\ \varepsilon & P^{*} & \overset{def}{=}\varepsilon\vee P^{+} \\
halt(P) & \overset{def}{=}\Box(\varepsilon\leftrightarrow P) & final(P) & \overset{def}{=}\Box(\varepsilon\rightarrow P) \\
keep(P) & \overset{def}{=}\Box(\bigcirc tt\rightarrow P) & rem(P) & \overset{def}{=}\Box(\bigcirc tt\rightarrow\bigcirc P) \\
inf & \overset{def}{=}\Box\bigcirc tt & P\,\|\,Q & \overset{def}{=}P\wedge(Q;tt)\vee Q\wedge(P;tt) \\
fin & \overset{def}{=}\Diamond\varepsilon & \ln(n) & \overset{def}{=}\begin{cases}\varepsilon, & \text{if } n=0\\ \bigcirc\ln(n-1), & \text{if } n\geqslant 1\end{cases}
\end{array}$$

其中, *tt* 代表 ture, *ff* 代表 false, ∨、→、↔等缩写词的定义与经典逻辑相同.

(4) 索引式

通常, PPTL 合式公式是指有限次运用语法规则得到的公式, 然而索引式是通过无穷次运用语法规则得到的公式, 其一般形式为 $OP_{i\in\mathbb{N}}X^{i}$, *OP* 为操作符, $X^{i}$ 为 PPTL 公式, ℕ为自然数集合.

特别地, 当 *OP* 被实例化为∨时, $\vee_{i\in\mathbb{N}}R[i]$ 即为一个常见的索引式, 其中, $R[i]$为含有索引 *i* 的 PPTL 公式, 将其称为索引项. 常见的索引项有以下 3 种形式.

(1) $\bigcirc^{i}P$: 表示对 *P* 应用 *i* 次 next 操作符;

(2) $P^{i}$: *P* 重复成立 *i* 次;

(3) $P^{(i)}$: *P* 从当前状态开始, 持续成立 *i* 个状态.

分别形式化定义为

$$\bigcirc^{i}\overset{def}{=}\begin{cases}P, & \text{if } i=0\\ \bigcirc\bigcirc^{i-1}P, & \text{if } i\geqslant 1\end{cases},\ P^{i}\overset{def}{=}\begin{cases}\varepsilon, & \text{if } i=0\\ P^{i-1};P, & \text{if } i\geqslant 1\end{cases},\ P^{(i)}\overset{def}{=}\begin{cases}tt, & \text{if } i=0\\ P, & \text{if } i=1.\\ P\wedge\bigcirc P^{(i-1)}, & \text{if } i>1\end{cases}$$

分别对以上 3 种索引项进行无穷次∨操作, 对应得到 3 种索引式: (1) $\vee_{i\in\mathbb{N}}\bigcirc^{i}P$, 该索引式等价于 $P\vee\bigcirc P\vee\bigcirc\bigcirc P\vee\bigcirc\bigcirc\bigcirc P\ldots$; (2) $\vee_{i\in\mathbb{N}}P^{i}$, 该索引式等价于$\varepsilon\vee P\vee(P;P)\vee(P;P;P)\ldots$; (3) $\vee_{i\in\mathbb{N}}P^{(i)}$, 该索引式等价于 $tt\vee P\vee(P\wedge\bigcirc P)\vee(P\wedge\bigcirc P\wedge\bigcirc\bigcirc P)\ldots$ 虽然从形式上看索引式并不是合式的, 但实际在某些情况下, 其已被证明可以等价于具有简明语法的合式 PPTL 公式或 LTL 公式.

### 1.3 含索引式的PPTL公理系统

公理系统包括公理和推理规则. 每条公理定义了一个可由公理系统直接导出的公式, 每条推理规则定义了一步推导, 此步推导可以从一个或多个假设得到一条结论公式.

一个公式 *P* 的形式化证明是一个公式序列 $P_0,\ldots,P_n(n\in\mathbb{N})$, 其中, $P=P_n$, 每个 $P_i$ 为:

(1) 一条公理, 或

(2) 一条推理规则的结论公式, 且该规则的每个假设都属于已证明过的公式 $P_0,\ldots,P_{n-1}$.

若公式 *P* 存在以上形式化证明, 称 *P* 可被公理系统证明, 即 *P* 为一条定理, 记为⊢*P*.

含索引式的 PPTL 公理系统$\Pi$, 包括两个公理子系统$\Pi_B$和$\Pi_I$, $\Pi=\Pi_B\cup\Pi_I$. $\Pi_B$包括用于基本结构的 PPTL 公式的公理和推理规则, 如$\bigcirc P$(next)、$(P_1,\ldots,P_m)\ prj\ P$(投影)和$\Box P$(always)等; $\Pi_I$包括用于索引式的公理和推理规则. 以下分别对两个公理子系统进行简要介绍, 其中, *S* 表示状态公式, $\Omega$表示一个有限的公式序列.

(1) 用于基本结构的 PPTL 公式的公理子系统$\Pi_B$.

其包含的公理如下.

TAU $\psi$ $\psi$为一个永真命题

POF $(\Omega_1,P_1\vee P_2,\Omega_2)\ prj\ Q\leftrightarrow(\Omega_1,P_1,\Omega_2)\ prj\ Q\vee(\Omega_1,P_2,\Omega_2)\ prj\ Q$

PIN $(\Omega,P\wedge inf)\ prj\ Q\leftrightarrow(\Omega,P\wedge inf)\ prj\ (Q\wedge fin)$

PFN $(\Omega_1,P_1,P_2,\Omega_2)\ prj\ Q\leftrightarrow(\Omega_1,P_1\wedge inf,P_2,\Omega_2)\ prj\ Q$

PSM $(\Omega_1,S\wedge\varepsilon,P,\Omega_2)\ prj\ Q\leftrightarrow(\Omega_1,S\wedge P,\Omega_2)\ prj\ Q$

POB $(\Omega)\ prj\ (Q_1\vee Q_2)\leftrightarrow(\Omega)\ prj\ Q_1\vee(\Omega)\ prj\ Q_2$

PSB $(\Omega)\ prj\ (S\wedge Q)\leftrightarrow S\wedge(\Omega)\ prj\ Q$ PSF $(S\wedge P,\Omega)prj\ Q\leftrightarrow S\wedge(P,\Omega)prj\ Q$

PNE $(\bigcirc P,\Omega)\ prj\ \varepsilon\leftrightarrow\bigcirc((P,\Omega)\ prj\ \varepsilon)$ DEF $(\Omega,\varepsilon)\ prj\ Q\leftrightarrow(\Omega)\ prj\ Q$

PEB $(P_1,P_2,\Omega)\ prj\ \varepsilon\leftrightarrow P_1;\ (P_2,\Omega)\ prj\ \varepsilon$ CPR $P^+\leftrightarrow P\vee(P\wedge\bigcirc tt;P^+)$

PNX $(\bigcirc P,\Omega)\ prj\ \bigcirc Q\leftrightarrow\bigcirc(P;(\Omega)\ prj\ Q)$ NXA $\bigcirc(P\wedge Q)\leftrightarrow\bigcirc P\wedge\bigcirc Q$

NXN $\bigcirc\neg P\leftrightarrow\neg(\varepsilon\vee\bigcirc P)$ CPC $(P^+;P^+)\rightarrow P^+$

NXO $\bigcirc(P\vee Q)\leftrightarrow\bigcirc P\vee\bigcirc Q$ CNX $\bigcirc P;Q\leftrightarrow\bigcirc(P;Q)$

*CAS* $P_1;(P_2;P_3)\leftrightarrow(P_1;P_2);P_3$ *STN* $P\wedge\Diamond\neg P\rightarrow\Diamond(P\wedge\bigcirc\neg P)$

*ALR* $\Box P\leftrightarrow P\wedge(\varepsilon\vee\bigcirc\Box P)$

推理规则如下:

$$\text{MP}\frac{P\ \ P\rightarrow Q}{Q}\qquad \text{SUB}\frac{P(Q)\ \ Q\leftrightarrow Q'}{P(Q')}$$

$$\text{NXM}\frac{P\rightarrow Q}{\bigcirc P\rightarrow\bigcirc Q}\qquad \text{PRM}\frac{P\rightarrow P'\ \ Q\rightarrow Q'}{(\Omega_1,P,\Omega_2)\ prj\ Q\rightarrow(\Omega_1,P',\Omega_2)\ prj\ Q'}$$

$$\text{ALW}\frac{P}{\Box P}\qquad \text{CPM}\frac{P\rightarrow Q}{P^+\rightarrow Q^+}\qquad \text{REC}\frac{P\rightarrow Q\vee\bigcirc P}{P\rightarrow\Diamond Q\vee\Box\bigcirc P}$$

(2) 用于索引式的公理子系统$\Pi_I$

其包含的公理如下.

IST $\vee_{i\in\mathbb{N}}Q^i\leftrightarrow Q^*$ IND $\vee_{i\in\mathbb{N}}(R[i]\vee R'[i])\leftrightarrow\vee_{i\in\mathbb{N}}R[i]\vee\vee_{i\in\mathbb{N}}R'[i]$

INR $\vee_{i\in\mathbb{N}}R[i]\leftrightarrow R[0]\vee\vee_{i\in\mathbb{N}}R[i+1]$ INA $\vee_{i\in\mathbb{N}}P\wedge R[i]\leftrightarrow P\wedge\vee_{i\in\mathbb{N}}R[i]$

INO $\vee_{i\in\mathbb{N}}(P\vee R[i])\leftrightarrow P\vee\vee_{i\in\mathbb{N}}R[i]$ INN $\vee_{i\in\mathbb{N}}\bigcirc R[i]\leftrightarrow\bigcirc\vee_{i\in\mathbb{N}}R[i]$

INC $\vee_{i\in\mathbb{N}}(P;R[i])\leftrightarrow P;\vee_{i\in\mathbb{N}}R[i]$ INS $R[i]\rightarrow\vee_{i\in\mathbb{N}}R[i]$

推理规则如下:

$$\text{INM}\frac{R[i]\rightarrow R'[i]}{\vee_{i\in\mathbb{N}}R[i]\rightarrow\vee_{i\in\mathbb{N}}R'[i]},\ \text{REF}\frac{R\leftrightarrow Q\vee P\wedge\bigcirc R\ R\rightarrow\Diamond Q}{\vee_{i\in\mathbb{N}}P^{(i)}\wedge\bigcirc^iQ\leftrightarrow R},\ \text{REI}\frac{R\leftrightarrow Q\vee P\wedge\bigcirc R\ \Box(P\wedge\bigcirc tt)\rightarrow R}{\vee_{i\in\mathbb{N}}P^{(i)}\wedge\bigcirc^iQ\vee\Box(P\wedge\bigcirc tt)\leftrightarrow R}$$

## 2 支持索引式的 PPTL 定理证明器

基于上一节中对 Coq 证明助手和含索引式的 PPTL 公理系统的介绍, 本节将使用 Coq 设计和构建支持索引式的 PPTL 定理证明器. 具体步骤是给出定理证明器的实现框架, 使用 Coq 完成公理系统的形式化定义, 分为以下 4 个部分.

(1) 基本公式和导出公式的实现;

(2) 索引式的实现;

(3) 基本公理及推理规则的实现;

(4) 含索引式的公理和推理规则的实现.

### 2.1 实现框架

公理系统基于形式推演, 只涉及公式的语法结构, 其正确性可被机械检验. Coq 就是一个基于机械检验的交互式证明助手, 其半自动化证明体现在对部分简单的证明步骤进行自动证明, 复杂的由人工指导证明. 同时, 若提供可靠完备的 PPTL 公理系统, Coq 可以保证定理证明器中推理所得公式的正确性. 以下结合 Coq 的系统结构, 对支持索引式的 PPTL 定理证明器的框架进行设计. Coq 系统结构分为证明开发系统和证明检查器, 相对应地, 实现框架大致分为两个部分: PPTL 公理系统的形式化定义部分和交互式定理证明部分, 如图 2 所示. 图中阴影部分为 Coq 内置的语言, 其余部分由本文设计实现.

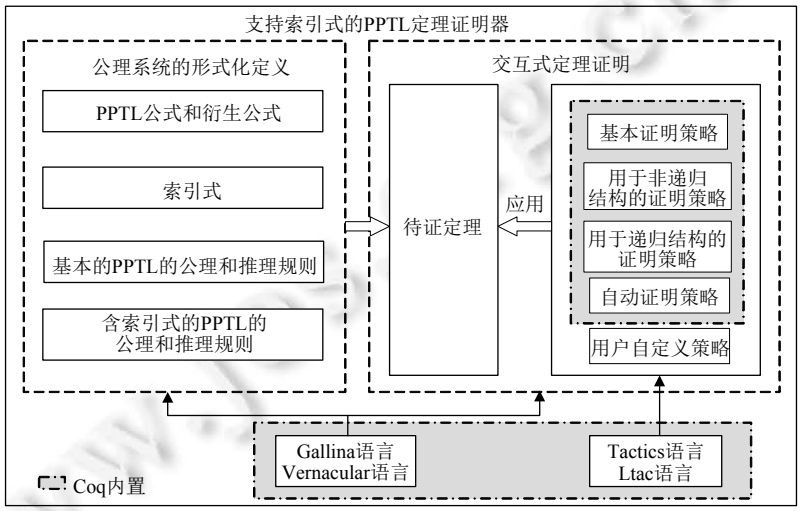


图 2 实现框架

第 1 部分的实现基于 Coq 系统结构的证明开发系统, 首先对 PPTL 公式和索引式进行语法定义, 其次对两个公理子系统 $\Pi_B$ 和 $\Pi_I$ 进行形式化定义, 最终实现在 Coq 中构建出可靠完备的公理系统. 该部分具体使用 Gallina 语言和 Vernacular 语言依次对以下 4 部分内容进行形式化定义.

(1) PPTL 公式和导出公式

在 Coq 中构建逻辑公理系统, 最基础的步骤是先为 PPTL 公式给出合理的定义, 即为名字赋予具体含义, 以便在后续代码中能够正确识别该名字并执行相应操作. 为方便对性质的描述, 还需在 PPTL 公式被定义的基础上, 对常见的导出公式进行定义, 并使用 Coq 的记号结构对其进行简写, 以简化公式的表述.

(2) 索引式

索引式是一种可将操作符无穷次运用于索引项的具有特殊结构的公式, 如 $\vee_{i\in\mathbb{N}}R[i]$ 即为一个索引式, 其将逻辑操作符$\vee$对索引项 $R[i]$进行无穷次的运用, 记为 $\vee_{i\in\mathbb{N}}$, 表达“无穷或”的含义, 自然数索引 $i$ 致使其与基本结构的 PPTL 公式具有不同的结构.

(3) 基本的 PPTL 的公理和推理规则

PPTL 公理系统为形式推演系统, 即在语法层面用人工符号语言来表示形式推演, 推演表现为一系列符号与符号之间的变形, 公理和推理规则均为形式可推演公式. 首先在 Coq 中形式化定义推演关系, 然后基于基本 PPTL 公式和导出公式的定义, 描述基本 PPTL 的公理和推理规则.

(4) 含索引式的 PPTL 的公理和推理规则

基于索引式的语法定义, 该部分也使用推演关系完成对公理子系统 $\Pi_I$ 的形式化描述. 需要说明的是: 形

如 $\vee_{i\in\mathbb{N}}$ 的索引式已被证明可以等价于 PPTL 合式公式, 因此, $\Pi_I$ 中的公理和推理规则也同样适用于含索引式的 PPTL 公式.

在两个公理子系统的所有的公理和推理规则都被形式化后, 可以被 Coq 识别并使用在定理证明的过程中.

第 2 部分基于 Coq 实现对待证定理的交互式定理证明. 待证定理的描述是基于已在 Coq 形式化的公理系统, 使用 PPTL 公式和索引式表示, 其形式为 Coq 代码; 交互式证明的核心部分体现在证明者与证明工具的交互操作中, 不断地对已证明的定理、假设或公理运用适当的推理策略, 来产生新的定理, 直至能够推出待证定理为止.

推理策略使用 Tactics 语言进行描述, 针对不同结构的待证定理和证明时不同的上下文环境, 可应用的推理策略也分为以下几个类别: 基本证明策略、用于非递归结构的证明策略、用于递归结构的证明策略、自动证明策略和用户自定义策略, 其中, 前四者属于 Coq 的内置策略. 基本证明策略适用于所有类型的递归定义结构; 用于非递归结构的证明策略适用于不包含索引式的归纳定义结构; 用于递归结构的证明策略适用于包含索引式的递归定义结构; 自动证明策略使用一个策略库, 将其作用在初始的目标上, 然后依次作用于其产生的每个子目标上, 直至所有目标都完成证明, 若任意子目标无法被证明, 则返回初始目标, 该类策略对证明上下文环境有较严苛的要求. 证明过程中, 用户也可根据需求对 Coq 内置策略进行组合并封装形成新的策略.

以上交互式证明的实现基于 Coq 的证明检查器, 证明过程的原理如图 3 所示. 在给定的逻辑框架内, 证明检查器检查定义和其相应的规约一致性, 并将定理的验证问题转化为匹配问题.

- 首先由用户给证明检查器提供待证目标, 使用 PPTL 对待验证的模型 $M$ 进行建模和待验证的模型性质 $P$ 进行描述, 并将 PPTL 公式转化为 Coq 代码, 构建出待证定理$\vdash M\rightarrow P$;
- 其次, 用户逐步对此目标运用合适的 Coq 策略, 对构造子的参数是否正确进行检验, 并检查该策略是否可以匹配当前的条件: 若匹配, 则会将待证目标进行化简或拆解为多个小目标; 否则, Coq 会给出错误信息, 证明无法继续进行. 当逐步运用策略直至无待证子目标时, 证明被完成, 此时使用 Qed 命令来结束证明过程, 并依次记录下证明过程中使用过的策略序列.

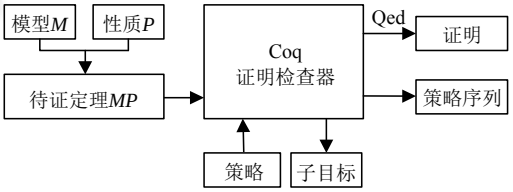


图 3 证明过程原理

基于 Coq 的证明检查机制, PPTL 定理证明器的交互式证明可以保证每个证明步骤的正确性, 但其证明效率取决于用户对证明策略选择是否合适, 更佳的策略选择会使证明更简洁快速. 因此, 这对用户的专业能力有较高的要求, 也需要用户对待证定理的证明方向有整体的把握.

### 2.2 基本公式及导出公式的实现

Coq 系统为有穷类型和递归类型提供了归纳类型机制. 一个归纳类型可以是一个常量, 该常量的类型为 *Set*、*Prop*、*Type* 这三大基础类型之一; 也可以是一个函数, 该函数可以是依赖类型, 其变元可为该归纳类型的参数, 但返回类型需为三大基础类型. Inductive 的构造子的返回类型也必须为新定义的这个归纳类型, 既可以是常量, 也可以是函数.

在定义基本 PPTL 公式前, 首先使用 Parameter 全局关键字定义原子命题集 *AP*, 如下:

Parameter *Ap*: *Set*.

原子命题的类型称为 *Ap*, 将 *Ap* 定义为一个普通的 Coq 类型 *Set* 类, 意味着可以像声明普通变量一样声明一个原子命题. 例如, 原子命题 *p* 可被声明为“Variable *p*:*Ap*.”.

基本 PPTL 公式的语法定义基于归纳定义, 选择使用 Inductive 关键字对其进行形式化. PPTL 公式的类型

名称为 *Formula*. 对应上一节中 PPTL 公式的定义, 类型 *Formula* 有 7 个构造子, 每个构造子都为一个函数类型. Coq 中使用箭头结构"→"来表示函数类型, 如"$A\to B$"是一个函数类型, 将类型 $A$ 的一个对象作为参数, 返回类型 $B$ 中的一个对象. 此外, 箭头结构的括号在没有歧义的情况下都可以省略, "$A\to B\to C$"可以表示参数类型为 $A$ 且返回一个参数类型为 $B\to C$ 的函数类型, 也可以表示读入两个类型分别为 $A$ 和 $B$ 的参数, 返回类型 $C$ 的函数类型. 在该归纳定义中, 前 6 个构造子用于归纳定义基本的 PPTL 公式, 将其标记为模块 *a*, Atom 表示如果参数 $p$ 是原子命题, 则 Atom $p$ 是 PPTL 公式; Not 表示如果参数 $P$ 是 PPTL 公式, 则 Not $P$ 是 PPTL 公式; And 表示如果参数 $P_1$ 和 $P_2$ 都是 PPTL 公式, 则 $P_1$ And $P_2$ 是 PPTL 公式; Next 表示如果 $P$ 是 PPTL 公式, 则 Next $P$ 是 PPTL 公式; *Prj* 表示如果$(P_1,P_2,\ldots,P_n)$是一个有穷 PPTL 公式序列, $Q$ 是 PPTL 公式, 则 *Prj* $(P_1,P_2,\ldots,P_n)$ $Q$ 是 PPTL 公式; *pPlus* 表示如果 $P$ 是 PPTL 公式, 则 *pPlus p* 是 PPTL 公式. 同理, 我们可以直接声明一个 PPTL 公式, 如"Variable *P*: *Formula*.". 定义如下所示:

Inductive *Formula* : *Set* :=

| Atom : $Ap \to Formula$

| Not : $Formula \to Formula$

| And : $Formula \to Formula \to Formula$ *a*

| Next : $Formula \to Formula$

| *Prj* : $list\ Formula \to Formula \to Formula$

| *pPlus* : $Formula \to Formula$

---

| *infiniteOr* : $(nat \to Formula) \to Formula$ *b*

*infiniteOr* 构造子被标记为模块 *b*, 含义将在第 2.3 节中给出具体阐述.

完成了 PPTL 公式的归纳定义后, 可以定义一些导出公式, 并用 Coq 记号结构简写这些公式, 以便于简化建模和验证. 同时, 也可使用 at level 为它们定义优先级, 如, $\Box P \overset{def}{=} \neg\Diamond\neg P$ 可被形式化为

Definition Alw(*p*: *Formula*): =Not (Som(Not *p*)).

并且被记号结构简写为

Notation $\Box p$: =(Alw *p*) (at level 60).

同理, 其余的导出公式也可被类似地形式化, 见表 1.

表 1 部分导出公式定义及记号

| 导出公式 | 对应定义及记号 | 导出公式 | 对应定义及记号 |
|---|---|---|---|
| $\neg P$ | Notation "¬*p*": =(Not *p*) (at level 60) | $\bigcirc P$ | Notation "○*p*:: =(Next *p*) (at level 60) |
| $P\wedge Q$ | Notation "*p*∧*q*": =(And *p q*) (at level 65) | $P\vee Q$ | Notation "*p*∨*q*": =(Or *p q*) (at level 75) |
| $\varepsilon$ | Definition *Emp*: =Not(Next *tt*) | *ff* | Definition *ff*: =Not *tt* |
| *tt* | Definition *tt*: =Or (*formula*) (Not *formula*) | final($P$) | Definition Final(*p*: *Formula*): =Alw(Imply *Emp p*) |
| *inf* | Definition *inf*: =Alw(Next *tt*) | *fin* | Definition *fin*: =Som(*Emp*) |
| $\Box P$ | Definition Alw(*p*:*Formula*): =Not (Som (Not *p*))<br>Notation "□*p*": =(Alw *p*) (at level 60) | $\Diamond P$ | Definition Som(*p*: *Formula*): =Sequ *tt p*<br>Notation "◇*p*": =(Som *p*) (at level 60) |
| $P;Q$ | Definition Sequ(*p q*:*Formula*): =*Prj* [*p*;*q*] *Emp*<br>Notation "*p*;;*q*": =(Sequ *p q*) (at level 80) | *keep*($P$) | Definition *Keep*(*p*: *Formula*):=<br>Alw(Imply(Next *tt*) *p*) |
| *rem*($P$) | Definition Rem(*p*:*Formula*): =<br>Alw(Imply(Next *tt*) (Next *p*)) | halt($P$) | Definition Halt (*p*: *Formula*):=<br>Alw(And(Imply *Emp p*) (Imply *p Emp*)) |
| ln($n$) | Fixpoint ln(*n*:*nat*): *Formula*:=<br>match *n* with<br>\|0⇒*Emp*<br>\|*S n*⇒Next(ln(*n*))<br>end | $P\|\|Q$ | Definition Para(*p*: *Formula*) (*q*: *Formula*): *Formula*: =<br>Or (And *p*(Sequ *q tt*)) (And *q* (Sequ *p tt*)) |

为避免过度使用括号, 将含索引式的 PPTL 中各操作符的优先级定义为(从 0–9, 优先级逐渐降低)

0. +,*; 1. ¬,○,□,◇, *prj*; 2. ∧; 3. $\vee_{i\in\mathbb{N}}$; 4. ∨; 5. ;; 6. ||; 7. →; 8. ↔

### 2.3 索引式的实现

索引式是将操作符无穷次作用于索引项得到的结构, 由于公理系统的原理是不断使用公理和推理规则与待证明定理进行匹配与应用, 属于形式推演系统, 且 Coq 证明检查器的工作原理也是在语法层面上对参数和构造子进行类型检查和匹配, 这两者的共通点致使在构建 PPTL 定理证明器时仅需考虑语法层面的匹配, 无需语义的支撑. 因此, 在 Coq 中对索引式进行实现, 使 Coq 能够正确识别出索引式的语法, 并可以对其应用含索引式的 PPTL 公理和推理规则.

在定义索引式前, 首先定义索引项, 索引项为一个以自然数 *i* 为参数的函数, 表示一个 PPTL 公式被运用了 *i* 次语法规则. 定义类型 *R* 表示索引项, 含义为: 任给定自然数 *i*, 返回值 *R i* 具有类型 *Formula*, *R* 函数称为索引项类型.

```
Parameter R: nat→Formula.
```

索引项类型中有 3 个特例, $\bigcirc^i P$、$P^i$ 和 $P^{(i)}$, 因为在公理子系统 $\Pi_I$ 中有部分特殊公理仅适用于特例的索引项, 故在此对其进行另外的定义.

(1) 定义 $\bigcirc^i P$ 如下:

```
Fixpoint nextiP(p : Formula)(i : nat) : Formula :=
match i with
    | 0 ⇒ p
    | S i ⇒ (○(nextiP p i))
end.
```

该结构表示对 PPTL 公式 *p* 进行 *i* 次 next 操作, 使用递归式函数编程中的关键字 Fixpoint 进行实现. 如上式, 该函数名称为 *nextiP*, 参数 *A* 是类型为 *Formula* 的 PPTL 公式, 参数 *B* 为自然数 *i*, 函数返回类型 *Formula*. 而函数内部的构造包含一个模式匹配结构, 其变元就是递归函数的变元, 对递归次数 *i* 进行匹配, 若递归 0 次, 即执行 next 操作 0 次, 则返回值为原公式 *P*; 若递归 *n* 次, 则对 *P* 执 *i* 次 next, 即为对执行 *i*–1 次 next 后的结果再执行一次 next, 该递归过程通过 Fixpoint 关键字的内置变量自动递减实现.

(2) 定义 $P^i$ 如下, 原理同步骤(1):

```
Fixpoint chopiP(p : Formula)(i : nat) : Formula :=
match i with
    | 0 ⇒ Emp
    | S i ⇒ (chopiP p i;; p)
end.
```

(3) 定义 $P^{(i)}$ 如下, 原理同步骤(1):

```
Fixpoint consiP(p : Formula)(i : nat) : Formula :=
match i with
    | 0 ⇒ tt
    |1 ⇒ p
    | S i ⇒ (p ∧ ○consiP p i)
end.
```

需要说明的是: 文献[17]中提出了索引项和索引式的概念, 但其存在的局限性是尚未对这两种新的结构进行显式归纳定义, 不同的索引项具有不同的定义方法和含义. 目前, 文献[18]对索引项和索引式进行了进一步的研究, 依据文献[30]提出的范式理论, 证明了每个索引项 *R*[*i*]和当操作符 *OP* 实例化为∨时的每个索引式 $\vee_{i\in\mathbb{N}} R[i]$, 均为 PPTL 合式公式.

$\vee_{i\in\mathbb{N}}$ 称为无穷或, 是以索引项类型为参数的函数, 对索引项进行无穷次∨操作, 形如 *R*[1]∨*R*[2]∨*R*[3]∨…, 在 *Formula* 的归纳定义中, 使用构造子 *infiniteOr* 定义该函数为(*nat*→*Formula*)→*Formula* 类型, 如第 2.2 节模块 *b* 所示, 将 *nat*→*Formula* 类型的索引项作为参数, 返回 *Formula* 类型的 PPTL 公式. 将 *infiniteOr* 简记为“∪”:

```
infiniteOr: (nat→Formula)→Formula
Notation “∪R”:=(infiniteOr R) (at level 70)
```

### 2.4 基本公理及推理规则的实现

含索引式的 PPTL 公理系统$\Pi$包括两个公理子系统, $\Pi_B$和$\Pi_I$, 具体内容参考第 1.3 节, 基于前面 PPTL 公式和索引式形式化定义的完成, 下一步对这两个子系统的公理和推理规则进行实现.

由于$\Pi_B$中部分公理仅适用于状态公式, 如 PSM、PSB、PSF 等, 因此在形式化公理和推理规则前, 需要先对状态公式进行定义. 状态公式为不包含时序操作符 next 和 *prj* 的 PPTL 公式, 使用 Fixpoint 关键字, 结合模式匹配结构定义函数 *ifSF*, 以判断 PPTL 公式是否为状态公式. 该函数接收一个 PPTL 公式 *p* 作为参数, 返回值为布尔值, 实现思想是: 对公式 *p* 中的操作符进行递归地拆解, 若遇到 Not、And, 则进行拆解, 直至递归到 Atom 原子命题, 则函数返回 true, 即该公式为仅包含 Not 和 And 操作符的状态公式; 否则, 函数返回 false, 即该公式为含有时序操作符的 PPTL 公式, 具体实现如下.

```
Fixpoint if SF(p : Formula) : bool :=
match p with
        | Atom p1 ⇒ true
        | Not p1 ⇒ if SF p1
        | And p1 q1 ⇒ match if SF p1 with
                                | true ⇒ if SF q1
                                | false ⇒ false
                      end
        | other ⇒ false
end.
```

PPTL 公理系统是基于语法的形式推演系统, 建立在公理和推理规则之上, 公理为一系列形式可推演公式, 推理规则由公式推导出公式. 在形式化公理和推理规则前, 有必要先对"推演关系⊢"进行形式化定义, 当且仅当 PPTL 公式 *P* 满足⊢*P* 时, 该公式为形式可推演公式. 因此, 从公理和推理规则出发, 若经过一系列公式的形式推演推导出公式 *P*, 则可证明 *P* 为无条件成立的永真公式.

Coq 的命题证明系统基于语法上的类型检查, Coq 中每个命题均为一个形式可推演公式. 对于命题的证明, Coq 将待证明定理定义为一个目标, 目标包含: (1) 一个需要被证明的命题 *P*; (2) 一个语境, 包含证明一个定理所需的所有公理和证明过程中产生的定理或引理.

因此, PPTL 公理系统中的所有 PPTL 公式, 包括公理、推理规则、待证定理等, 均为形式可推演公式. 为最大化地利用 Coq 内置证明策略进行推演证明, 使用 Inductive 关键字定义归纳类型 *derivable*: *Formula*→*Prop* 表示推演关系, 将每个类型为 *Formula* 的 PPTL 公式转换为 Coq 中的形式可推演公式 *Prop* 类型, 由此, "*derivable p*"即可形式化地表示 PPTL 公式 *P* 满足推演关系"⊢*P*".

公理和推理规则的形式化通过推演关系 *derivable* 归纳定义实现, 每条公理/推理规则均为 *derivable* 的构造子. 下面对公理子系统$\Pi_B$中的公理和推理规则进行形式化描述, 公理根据公式是否包含状态公式被分为两类, 选择公理 POB、PSF 和推理规则 NXM、SUB 作为示例进行说明, 其余定义与示例类似.

**公理描述**

(1) POB $(\Omega)\ prj\ (Q_1 \vee Q_2) \leftrightarrow (\Omega)\ prj\ Q_1 \vee (\Omega)\ prj\ Q_2$

该公理可被形式化表示为

```
POB : forall (Q1 Q2 : Formula) (Ω : list Formula),
derivable (Prj (Ω) (Q1 ∨ Q2) ⇔ (Prj Ω Q1 ∨ Prj Ω Q2))
```

公理 POB 的含义为 *Prj* 后面的∨满足分配率. 对于任意两个类型为 *Formula* 的 PPTL 公式 $Q_1$、$Q_2$和任意一个类型为 list *Formula* 的 PPTL 公式序列$\Omega_1$, 有$(\Omega)\ prj\ (Q_1 \vee Q_2) \leftrightarrow (\Omega)\ prj\ Q_1 \vee (\Omega)\ prj\ Q_2$无条件成立, $Q_1 \vee Q_2$被表示为(And $Q_1$ $Q_2$), $(\Omega)\ prj\ (Q_1 \vee Q_2)$被表示为(*Prj* $(\Omega_1)$(And $Q_1$ $Q_2$)), 其余公式同理. 为了简化表达, 在上一节中使用 Notation 关键字对一些操作符赋予了记号标记, 因此公理最终可被简化为上式的形式.

(2) PSF $(S \wedge P, \Omega)\ prj\ Q \leftrightarrow S \wedge (P, \Omega)\ prj\ Q$

该公理可被形式化表示为

PSF: **forall** ($P$ $Q$ $S$ : *Formula*) ($\Omega$ : list *Formula*),
*derivable* (*Prj* ([*Stateformula* $S \wedge P$]++$\Omega$) $Q$)
$\Leftrightarrow$ (*Stateformula* $S \wedge$ *Prj* ([$P$]++$\Omega$) $Q$)

公理 PSF 的含义为, *Prj* 前面的状态公式 $S$ 可从公式序列中分离出来. 该公理与上条公理的区别为, $S$ 被要求为状态公式. 定义辅助函数 *Stateformula* $S$ 如下: 若 $S$ 为状态公式, 即 if *SF* $S$ 为 true, 函数返回值为该状态公式. 那么对于任意 PPTL 公式 $P$、$Q$、任意时序公式 $S$、任意 PPTL 公式序列$\Omega$, 有($S \wedge P$,$\Omega$) *prj*$\leftrightarrow S \wedge$($P$,$\Omega$) *prj* $Q$ 无条件成立, 在 Coq 中可表示为

*derivable* (*Prj* ([*Stateformula* $S \wedge P$]++$\Omega$) $Q$)$\Leftrightarrow$(*Stateformula* $S \wedge$ *Prj* ([$P$]++$\Omega$) $Q$).

(3) NXM $\dfrac{P \rightarrow Q}{\bigcirc P \rightarrow \bigcirc Q}$

该推理规则可被形式化表示为

NXM: **forall** ($P$ $Q$ : *Formula*),
*derivable* ($P \dashrightarrow Q$) $\rightarrow$ *derivable* ($\bigcirc P \dashrightarrow \bigcirc Q$)

NXM 规则的含义为: 对于任意公式 $P$、$Q$, 若($P \dashrightarrow Q$)永真, 则可推理出($\bigcirc P \dashrightarrow \bigcirc Q$)永真. 使用 *derivable* 归纳类型定义该规则为 *derivable*($P \dashrightarrow Q$)$\rightarrow$*derivable*($\bigcirc P \dashrightarrow \bigcirc Q$), 表示若公式($P \dashrightarrow Q$)为形式可推演公式, 则 $\bigcirc P \dashrightarrow \bigcirc Q$ 也为可推演公式.

(4) SUB $\dfrac{P(Q)\ Q \leftrightarrow Q'}{P(Q')}$

SUB 为替换规则, 若 $P$ 永真, $Q$ 为 $P$ 中的子公式, 且 $Q$ 等价于 $Q'$永真, 则可以推理得到: 若将 $P$ 中的 $Q$ 替换为 $Q'$, $P$ 仍旧永真. 该规则通过 Coq 中的 replace $Q$ with $Q'$策略进行实现. 该策略用于将公式中的某个子表达式替换为另一个子表达式, 并且将这两个子表达式等价作为新的子目标, 即将条件中的所有 $Q$ 替换为 $Q'$, 同时生成子目标 $Q$=$Q'$. 然而, 由于 SUB 替换规则中的条件 $Q \leftrightarrow Q'$被形式化为 *derivable*($Q \Leftrightarrow Q'$), 为完成子目标 $Q$=$Q'$的证明, 定义辅助公理 Equ 如下式, 在保证不改变证明目标含义的前提下, 对证明目标的形式进行转化, 使得条件与目标的形式得到匹配, 从而完成 SUB 规则的形式化:

Axiom Equ: **forall** ($P$ $Q$ : *Formula*),
($P = Q$) $\leftrightarrow$ (*derivable* ($P \Leftrightarrow Q$))

### 2.5 含索引式的公理及推理规则的实现

根据索引项是否属于 3 种特殊索引项, 公理子系统$\Pi_I$中的公理和推理规则也被大致分为两类, 下面分别从两类中以一条公理作为示例进行说明. 推理规则与第 2.4 节中类似.

(1) IST $\vee_{i \in \mathbb{N}} Q^i \leftrightarrow Q^*$

该公理可被形式化表示为

IST: **forall** $Q$ : *Formula*,
*derivable* ((∪ *chopiP* $Q$) $\Leftrightarrow$ (*pStar* $Q$))

IST 公理属于$\Pi_I$子系统中的一条包含特例索引项 $Q^i$ 的公理, 含义为索引式 $\vee_{i \in \mathbb{N}} Q^i$ 可被等价表示为 PPTL 公式 $Q^*$. 对于任意 PPTL 公式 $Q$, $\vee_{i \in \mathbb{N}} Q^i$ 被描述为(∪*chopiP* $Q$), 其中, *chopiP* $Q$ 为 *nat*$\rightarrow$*Formula* 类型, 经过 *infiniteOr* 函数的作用后转换为 *Formula* 类型, $Q^*$被描述为 *pStar* $Q$, 为 *Formula* 类型. 当两个公式都为 *Formula* 类型时, 使用逻辑等价操作符$\Leftrightarrow$将其连接形成命题(∪*chopiP* $Q$)$\Leftrightarrow$(*pStar* $Q$), 该公理即可被描述为 *derivable* ((∪*chopiP* $Q$)$\Leftrightarrow$(*pStar* $Q$)).

(2) IND $\vee_{i \in \mathbb{N}} (R[i] \vee R'[i]) \leftrightarrow \vee_{i \in \mathbb{N}} R[i] \vee \vee_{i \in \mathbb{N}} R'[i]$

该公理可被形式化表示为

IND: **forall** $R$ $R'$: *indexterm*,
*derivable* (∪(*ior* $R$ $R'$) $\Leftrightarrow$ (∪ $R \vee \cup R'$))

IND 公理属于$\Pi_1^1$子系统中的一条不包含特殊索引项的公理, 含义为: 两个∨连接的索引项经过无穷或操作, 等价于两个索引项分别经过无穷或操作后再进行∨操作. 为方便表述, 声明 *indexterm* 为 *nat*→*Formula* 类型. 对于任意类型为 *indexterm* 的一般形式索引项 $R$ 和 $R'$, $R[i]\vee R'[i]$被描述为(*ior R R′*), 其中, *ior* 函数为索引项间的 or 操作, 定义为(*indexterm*)→(*indexterm*)→(*indexterm*), 表示两个索引项经过 *ior* 操作仍为索引项类型. $\cup$(*ior R R′*)则为 *Formula* 类型. $\vee_{i\in\mathbb{N}}R[i]$可被描述为($\cup R$), 类型为 *Formula*; $\vee_{i\in\mathbb{N}}R[i]\vee\vee_{i\in\mathbb{N}}R'[i]$可被描述为$\cup R\vee\cup R'$, 类型为 *Formula*. 当两个公式都为 *Formula* 类型时, 使用逻辑等价操作符⇔将其连接形成命题:

$$(\cup(ior\ R\ R')\Leftrightarrow(\cup R\vee\cup R')).$$

## 3 证明实例

为验证支持索引式的 PPTL 定理证明器的实用性, 本节将选择公理系统中的两条定理: T1: $keep(P)\leftrightarrow\varepsilon\vee P\wedge\bigcirc keep(P)$和 $T5$: $\vee_{i\in\mathbb{N}}P^{(i)}\wedge\bigcirc^i\varepsilon\leftrightarrow keep(P)\wedge\Diamond P$ 作为实例, 阐述如何在 Coq 中使用 PPTL 公理系统和 Coq 策略对待证定理进行交互式证明. 其中, T5 着重于含索引式公式的证明. 以下分别对两条定理的含义及应用进行阐述, 其次, 结合代码对该定理在定理证明器中的推理证明过程进行说明.

### 3.1 基本公式的实例证明

(1) T1 的含义及应用

PPTL 公式 $keep(P)$的定义为$\Box(\bigcirc tt\rightarrow p)$, 其含义为: 在区间上, 除终止状态外, 其余状态上总是有公式 $P$ 成立. 定理 T1: $keep(P)\leftrightarrow\varepsilon\vee P\wedge\bigcirc keep(P)$重新递归定义了 $keep(P)$: 在区间上公式 $keep(P)$成立, 若当前状态为终止状态, 或者在当前状态 $P$ 成立且下一状态 $keep(P)$成立.

由于该定理具有递归形式, 不断地对 $keep(P)$应用此定理, 可得到 $keep(P)\leftrightarrow\varepsilon\vee P\wedge\bigcirc\varepsilon\vee P\wedge\bigcirc P\wedge\bigcirc^2\varepsilon\vee P\wedge\bigcirc P\wedge\bigcirc^2 P\wedge\bigcirc^3\varepsilon\vee\ldots$, 等价符右边经整理可表示为索引式 $\vee_{i\in\mathbb{N}}P^{(i)}\wedge\bigcirc^i\varepsilon$. 进而, 索引式 $\vee_{i\in\mathbb{N}}P^{(i)}\wedge\bigcirc^i\varepsilon$ 可被 PPTL 合式公式 $keep(P)$简洁地表达.

Duan 在文献[17]中提出并证明: 对于递归方程 $X\equiv Q\vee P\wedge\bigcirc X$, 若 $X$、$P$、$Q$ 均为 PPTL 公式, 则该方程有且仅有两个解: $\vee_{i\in\mathbb{N}}P^{(i)}\wedge\bigcirc^i Q$ 与 $\vee_{i\in\mathbb{N}}P^{(i)}\wedge\bigcirc^i Q\vee\Box(P\wedge\bigcirc tt)$. 若将方程中的 $X$ 实例化为 PPTL 公式 $keep(P)$, $Q$ 实例化为 PPTL 公式$\varepsilon$, 得到方程 $keep(P)\equiv\varepsilon\vee P\wedge\bigcirc keep(P)$, 该方程在公理系统中被表述为定理 T1. 因此, 在公理系统中证明定理 T1, 可被应用于后续索引式 $\vee_{i\in\mathbb{N}}P^{(i)}\wedge\bigcirc^i\varepsilon$ 和 $\vee_{i\in\mathbb{N}}P^{(i)}\wedge\bigcirc^i\varepsilon\vee\Box(P\wedge\bigcirc tt)$ 合式性的证明.

(2) T1 的证明过程

交互式的定理证明通过平台 CoqIDE 进行实现, 左边部分显示由用户输入的 Coq 代码, 右边部分在证明过程中给出当前的证明状态. 下面结合代码说明定理 T1 在定理证明器中的推理证明步骤.

a) 待证定理描述

首先使用关键字 Theorem 描述 T1 定理, 表示该定理的名称为 T1, 命题被描述为 *derivable*(*Keep p*⇔(*Emp*∨*p*∧(*Keep p*))), 在该命令后, 使用一个可选命令 *Proof*, 使得该会话的脚本更易阅读, 此时 Coq 进入证明模式, 结合当前的上下文和待证命题, 系统会创建一个原始目标, 在显示目标时, 用一条水平线将上下文和待证命题上下分隔开, 用户可应用策略不断地对该目标进行拆解.

b) 引入假设, 展开证明

使用 *intros* 策略引入新的假设, 此时"*p*: *Formula*", 即"*p* 为一个 *Formula* 类型的 PPTL 公式"便成为一个假设, 被加入水平线上方, 用户仅需解释如何运用该假设去构造结论的证明.

c) 引入引理 H1

使用 *assert* 策略引入引理 H1: *derivable*(*Keep p*⇔(○*tt*−−→*p*)), 应用该策略, 可以按照相反的顺序证明两个子目标: 当应用"*assert*(*Q*)"证明 *P* 时, 用户先证明 *Q*, 然后将其作为一个引理使用, 当前的待证子目标变为引理 H1.

d) 证明 H1 引理

使用 *unfold* 策略将 *Keep p* 按照其定义展开, 得到 *derivable*(□(○*tt*−→*p*)⇔□(○*tt*−→*p*)), 此时双向蕴含符左右两边形式完全相同, 使用策略 *apply*, 将公理 REF: **forall** (*p*: *Formula*), *derivable*(*p*⇔*p*)作用于当前子目标, 即完成了 H1 引理的证明. H1 被加入水平线上方, 成为上下文中的一条假设.

e) 引入并证明引理 H2

同步骤(3), 引入引理 H2: *derivable*(*Keep p*⇔((○*tt*−→*p*)∧(*Emp*∨(○(*Keep p*)))), 当前的待证子目标变为引理 H2, 使用 *unfold* 策略对 *Keep p* 进行展开后, 使用 *apply* ALR(○*tt*−→*P*)策略将公理系统中的公理 ALR: □*P*↔*P*∧(*ε*∨○□*P*)作用于待证目标, 即完成了 H2 的证明.

以上 5 步的证明过程及此时的上下文和待证命题如图 4 所示.

| Coq程序代码 | 当前证明状态 |
|---|---|
| Theorem T1: **forall** *p*: *Formula*,<br>*derivable* (*Keep p*⇔(*Emp p* ○(*Keep p*))).<br>*Proof*.<br>*intros p*.<br>*assert* (H1: *derivable* (*Keep p*⇔(○*tt*⟶*p*))).<br>  *unfold Keep*. *Apply Ref*.<br>*assert* (H2: *derivable* (*Keep p*⇔<br>((((○*tt*)⟶*p*) (*Emp* (○(*Keep p*)))))).<br>*replace* (*Keep p*) with ((○*tt*⟶*p*)).<br>*apply* (ALR (○*tt*⟶*p*)). | 1 subgoal<br>*p*: *Formula*<br>H1: *derivable* (*Keep p*⇔(○*tt*⟶*p*))<br>H2: *derivable* (*Keep p*⇔(○*tt*⟶*p*) (*Emp*<br>○*Keep p*))<br>————————————————————(1/1)<br>*derivable* (*Keep p*⇔*Emp p*○*Keep p*) |

图 4 定理 T1 的证明过程

f) 使用 H2 推导 T1

将 SUB 规则运用于 H2, 将 H2 中的子公式○*tt*−→*p* 替换 *Emp*∨*p*, 该规则通过 Coq 的策略 *replace*(*A*) with (*B*)实现, 后依次使用交换律、分配率等 TAU 公理, 将 H2 变换为与 T1 相同的形式.

g) 完成证明

至此, Coq 提示“无更多子目标”, 即定理 T1 证明已完成, 如图 5 所示.

h) 退出证明

最后, 使用 Qed 命令声明证明已完成, 并使 Coq 退出证明模式. 同时, 在左侧记录证明策略的序列. 并且, 定理 T1 可被用于后续的证明中.

| Coq程序代码 | 当前证明状态 |
|---|---|
| Theorem T1: **forall** *p*: *Formula*,<br>*derivable* (*Keep p*⇔(*Emp p* ○(*Keep p*))).<br>*Proof*.<br>...<br>(接图4代码)<br>*replace* (○*tt*⟶*p*) with (*Emp p*) in H2.<br>*replace* (*Emp p* ○ *Keep p*) with<br>  ((*Emp p*) (*Emp* ○ *Keep p*)).<br>*apply* H2.<br>*apply* Equ. *Apply* Exc. *Apply Distri_or*.<br>  *apply* Equ. *Unfold Emp*. *Unfold* “⟶”. *apply Ref*. | No more subgoals. |

图 5 定理 T1 证明结束

### 3.2 含索引式公式的实例证明

(1) T5 的含义

定理 T5: $\vee_{i\in\mathbb{N}} P^{(i)}\wedge\bigcirc^i\varepsilon \leftrightarrow keep(P)\wedge\Diamond\varepsilon$ 表示索引式 $\vee_{i\in\mathbb{N}} P^{(i)}\wedge\bigcirc^i\varepsilon$ 等价于基本 PPTL 公式 *keep*(*P*)∧◇*ε*. 索引式 $\vee_{i\in\mathbb{N}} P^{(i)}\wedge\bigcirc^i\varepsilon=(P^{(0)}\wedge\bigcirc^0\varepsilon)\vee(P^1\wedge\bigcirc^1\varepsilon)\vee(P^{(2)}\wedge\bigcirc^2\varepsilon)\ldots=(tt\wedge\varepsilon)\vee(P\wedge\bigcirc\varepsilon)\vee(P\wedge\bigcirc P\wedge\bigcirc\bigcirc\varepsilon)\ldots$ 直观的含义为区间为有穷区间, 且在每个非终止状态上有 *P* 成立. *keep*(*P*)表示在区间上, 在每个非终止状态上有 *P* 成

立, $\Diamond\varepsilon$表示区间存在终止状态, 因此, $keep(P)\wedge\Diamond\varepsilon$在语义上等价于索引式 $\vee_{i\in\mathbb{N}}P^{(i)}\wedge\bigcirc^{i}\varepsilon$.

(2) T5 的证明过程

在 T1 的证明中已展示了该定理证明器对基本 PPTL 公式的证明过程, 本节将对部分基本公理和推理规则的使用进行一定程度的省略, 重点展示对含索引式的推导步骤进行详细说明.

a) 待证定理描述

首先将 T5 在 Coq 中描述:

Theorem T5: **forall** (*p*: *Formula*), *derivable*((∪*iand*(*consiP p*)(*nextiP Emp*))⇔(*Keep p*∧◇*Emp*)).

b) 展开证明, 引入引理 H1 和 H2

当 *P* 为 *p*、*Q* 为$\varepsilon$、*R* 为 *keep*(*P*)∧◇$\varepsilon$时, 待证定理与推理规则 REF 的结论具有相同的结构特征, 因此引入 REF 的两条前提作为引理 H1 和 H2, 并对其进行证明, 其中,

H1: *derivable*(*Keep*∧◇*Emp*⇔*Emp*∨(*p*∧○*Keep p*)∧○(◇*Emp*)).

证明过程使用基本公式的公理和推理规则, 在此不进行详细说明. 至此, 得到的上下文如图 6 所示.

| Coq程序代码 | 当前证明状态 |
|---|---|
| **Theorem T5: forall (*p*: *Formula*),**<br>*derivable* (*iand* (*consiP p*)(*nextiP Emp*))<br>⇔(*Keep p Emp*)).<br>*Proof.*<br>*intros p.*<br>*assert*(H1: *derivable* ((*Emp* (*p*○*Keep p*)<br>(*Emp*○*Emp*))⇔<br>(*Emp* (*p* (○ *Keep p*)○*Emp*)))).<br>-*apply* Exc.<br>*apply Distri_or* with (*p*:=*Emp*) (*q*:=(*p*○*Keep p*))<br>(*r*:=○(*Emp*)).<br>-*replace*(((*Emp p*○*Keep p*))) with (*Keep p*) in H1.<br>*replace*(*Emp*○(*Emp*)) with (*Emp*) in H1.<br>*assert*(H2: *derivable* (*Keep p Emp*)⟶*Emp*).<br>--*apply Simple_q*. | 1 subgoal<br>*p*: *Formula*<br>H1: *derivable* (*Keep p Emp*⇔<br>*Emp* (*p* ○ *Keep p*) ○ (*Emp*))<br>H2: *derivable* (*Keep p Emp*⟶*Emp*)<br>(1/1)<br>*derivable* (*iand* (*consiP p*) (*nextiP Emp*)⇔<br>*Keep p Emp*) |

图 6 引入 H1 和 H2 后的上下文

c) 引入引理 H3

引入引理 H3: *derivable*(*Keep p*∧◇*Emp*⇔*Emp*∨*p*∧○(*Keep p*∧◇*Emp*))∧*derivable*(*Keep p*∧◇*Emp*−−→◇*Emp*).

使用 *split* 策略将其拆分为两个子目标, 使用 *apply* H1 策略和 *apply* NXA 策略等完成对第 1 个子目标的证明, 使用 *apply* H2 完成对第 2 个子目标的证明.

d) 应用推理规则 REF 完成 T5 的证明

此时, 对待证目标使用含索引式的 REF 推理规则, 反向推理将目标改写为

*derivable*(*Keep p*∧◇*Emp*⇔*Emp*∨*p*∧○(*Keep p*∧◇*Emp*))∧*derivable*(*Keep p*∧◇*Emp*−−→◇*Emp*).

与 H3 具有相同形式, 使用 *apply* H3 策略, 此时显示 No more subgoals. 完成定理 T5 的证明.

e) 退出证明

使用 Qed 命令声明证明已完成, 并使 Coq 退出证明模式. 同时, 在左侧记录证明策略的序列. 此时, 定理 T5 的正确性已被证明, 并可被用于后续的证明.

## 4 总结与展望

为了提高基于 PPTL 的定理证明方法的自动化程度, 本文开发了一个支持索引式的 PPTL 定理证明器, 设计了该定理证明器的实现框架, 将含索引式的 PPTL 公理系统形式化地实现于 Coq 证明助手中, 并使用证明实例验证了该定理证明器的可用性. 该半自动化的定理证明器在简化定理证明过程的同时, 也保证了证明的正确性, 提高了证明的可读性.

然而, 该工作仍存在改进空间, 本文对后续工作做出以下展望: (1) 在提高证明自动化程度方面, 探索待证公式尤其是含索引式公式的句法特征, 总结常用于该结构的公理和推理规则, 进而通过 Coq 模式匹配和条件表达式约束, 对已有的证明策略进行组合和控制, 以创建新的自动化策略, 应用于推导中. 即使该自动化策略未能成功完成证明, 也可为下一步的推导提供思路和方向; (2) 目前, 对于该定理证明器的使用还仅限于简单定理的证明, 下一步的研究将针对复杂系统的性质验证展开, 以证明该定理证明器的实用性.

**References**:


[1] Burch JR, Clarke EM, Long DE, McMillan KL, Dill DL. Symbolic model checking for sequential circuit verification. IEEE Trans. on Computer-aided Design of Integrated Circuits and Systems, 1994, 13(4): 401−424.

[2] Bledsoe W, Loveland D. Automated Theorem Proving: After 25 Years. American Mathematical Society, 1984. [doi: http://dx.doi.org/10.1090/conm/029]

[3] Bérard B, Bidoit M, Finkel A, Laroussinie F, Petit A, Petrucci L, Schnoebelen P, McKenzie P. Systems and Software Verification: Model-checking Techniques and Tools. Berlin, Heidelberg: Springer, 2001.

[4] Gabbay D, Pnueli A, Shelah S, Stavi J. On the temporal basis of fairness. In: Proc. of the 7th Annual ACM Symp. on Principles of Programming Languages. 1980. 163−173.

[5] Pnueli A, Kesten Y. A deductive proof system for CTL. In: Proc. of the 13th Int'l Conf. on Concurrency Theory. London: Springer, 2002. 24−40.

[6] Moszkowski BC. An automata-theoretic completeness proof for interval temporal logic. In: Proc. of the Int'l Colloquium on Automata, Languages, and Programming. Berlin, Heidelberg: Springer, 2000. 223−234.

[7] Cini C, Francalanza A. An LTL proof system for runtime verification. In: Proc. of the Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Berlin, Heidelberg: Springer, 2015. 581−595.

[8] Dax C, Hofmann M, Lange M. A proof system for the linear time μ-calculus. In: Proc. of the Int'l Conf. on Foundations of Software Technology & Theoretical Computer Science. Berlin, Heidelberg: Springer, 2006.

[9] Bolotov A, Basukoski A, Grigorievy O, Shangin V. Natural deduction calculus for linear-time temporal logic. In: Proc. of the European Workshop on Logics in Artificial Intelligence. Berlin, Heidelberg: Springer, 2006. 56−58.

[10] Moszkowski B. A complete axiom system for propositional interval temporal logic with infinite time. Logical Methods in Computer Science, 2012, 8(3:10): 1−56.

[11] Duan ZH. Temporal Logic and Temporal Logic Programming. Beijing: Science Press, 2005.

[12] Tian C, Duan ZH. Expressiveness of propositional projection temporal logic with star. Theoretical Computer Science, 2011, 412: 1729−1744.

[13] Duan ZH, Zhang N, Koutny M. A complete proof system for propositional projection temporal logic. Theoretical Computer Science, 2013, 497: 84−107.

[14] Duan ZH, Zhang N. A complete axiomatization for propositional projection temporal logic. In: Proc. of the 2nd IFIP/IEEE Int'l Symp. on Theoretical Aspects of Software Engineering. Washington: IEEE Computer Society, 2008. 271−278.

[15] Zhang N, Duan ZH, Tian C. A formal proof of the deadline driven scheduler in PPTL axiomatic system. Theoretical Computer Science, 2014, 554: 229−253.

[16] Zhang N, Yang MF, Gu B, Duan ZH, Tian C. Verifying safety critical task scheduling systems in PPTL axiom system. Journal of Combinatorial Optimization, 2016, 31(2): 577−603.

[17] Duan ZH, Tian C, Zhang N, Ma Q, Du HW. Index set expressions can represent temporal logic formulas. Theoretical Computer Science, 2019, 788: 21−38.

[18] Zhao L, Wang XB, Shu XF, Zhang N. A sound and complete proof system for a unified temporal logic. Theoretical Computer Science, 2020, 838: 25−44.

[19] Harrison H, Urban J, Wiedijk F. History of interactive theorem proving. Handbook of the History of Logic, 2014, 9: 135−214.

[20] Kalvala S. Using Isabelle to prove simple theorems. In: Proc. of the HOL Users' Group Workshop. Berlin, Heidelberg: Springer, 1993. 514−517.

[21] Bertot Y, Castéran P. Interactive theorem proving and program development. In: Coq'Art: The Calculus of Inductive Constructions. Berlin, Heidelberg: Springer, 2004.

[22] Brock B, Kaufmann M, Moore JS. ACL2 theorems about commercial microprocessors. In: Proc. of the Int'l Conf. on Formal Methods in Computer-aided Design. Berlin, Heidelberg: Springer, 1996. 275−293.

[23] Evans N, Schneider S. Verifying security protocols with PVS: Widening the rank function approach. The Journal of Logic and Algebraic Programming, 2005, 64(2): 253−284.

[24] Doorn FV. Propositional calculus in Coq. 2015. http://arxiv.org/abs/1503.08744v2

[25] Cau A, Zedan H. Refining interval temporal logic specifications. In: Proc. of the 4th Int'l AMAST Workshop on Real-time Systems and Concurrent and Distributed Software. Mallorca, 1997. 79−94.

[26] Pnueli A, Arons T. TLPVS: A PVS-based LTL verification system. In: Verification: Theory and Practice. Berlin, Heidelberg: Springer, 2003. 598−625.

[27] Ma Q. Constraint solving and formal verification with MSVL [Ph.D. Thesis]. Xi'an: Xidian University, 2015 (in Chinese with English abstract).

[28] Qian L, Duan ZH, Zhang N, Tian C. A proof system for MSVL programs in Coq. In: Proc. of the Int'l Workshop on Structured Object-oriented Formal Language and Method. Cham: Springer, 2016. 121−143.

[29] Barras B, Boutin S, Cornes C. The Coq Proof Assistant Reference Manual. 6th ed., Inria, 1997.

[30] Duan ZH, Tian C. A practical decision procedure for propositional projection temporal logic with innite models. Theoretical Computer Science, 2014, 554: 169−190.

**附中文参考文献**:

[27] 马倩. MSVL 语言的约束求解与形式验证 [博士学位论文]. 西安: 西安电子科技大学, 2015.

**王小兵**(1979－), 男, 博士, 副教授, 博士生导师, CCF 高级会员, 主要研究领域为形式化方法, 形式化验证, 时序逻辑.

**李春奕**(1996－), 女, 博士, CCF 学生会员, 主要研究领域为形式化方法, 形式化验证, 时序逻辑.

**寇蒙莎**(1998－), 女, 硕士, CCF 学生会员, 主要研究领域为形式化方法, 形式化验证, 定理证明.

**赵亮**(1984－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为形式化方法, 形式化验证, 时序逻辑.