国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第16卷第1期 1994年3月 Vol. 16 No. 1

PTL证明器的实现技术*

贲可荣 陈火旺

（国防科技大学电子计算机系 长沙 410073）

摘 要 基于文[10]中的理论，我们用Turbo－Prolog编程，在386微机上成功地实现了命题时态逻辑定理的证明器。该证明器在处理next幂次、归纳、归结、◇(x∧y)、until等方面，均有独到之处。这些方面，克服了以往工作的不足。证明器界面友好、速度快、能力强。

关键词 时态逻辑，定理证明，自动推理

分类号 TP18，TP311.51

时态逻辑产生于语言学和哲学，但随着软件工程、人工智能的产生和发展，时态逻辑在计算机科学中开始产生重要影响，反过来，也为时态逻辑自身的研究注入了新的活力。

时态逻辑很多，随时间结构、算子的选择而异。本文主要研究Manna和Pnueli在并发程序验证方面提出的一种时态逻辑，其时间结构与自然数结构同构，时态算子选择为□，◇，○，U。

程序设计语言和时态逻辑两者均可看做对序列的描述。通过程序设计语言的操作语义，可导出程序的可能执行序列集。平行地，时态逻辑的模型论语义建立了逻辑公式与模型集之间的关系，而且这些模型取序列形式。将此与执行序列等同，时态公式与计算机程序以共同的结构集合互相关联。借助这种非直接的关系，可在程序与时态逻辑公式之间建立一种直接关系。

Manna和Pnueli引入时态语义，目的是使用时态逻辑证明一程序满足一定规范——正确性陈述以及公平性等特性。其证明是取一明显被程序满足的时态公式做它的前提，用时态逻辑导出不明显被程序满足的别的公式。为了能够证明表达一程序正确性的时态逻辑公式是一公式集的逻辑推论，我们必须构造时态逻辑定理证明器。

Wolper，Cavalli，Farinas，Venketesh，Abadi，Plaisted等人在时态逻辑证明方法上均做了有益的探讨，但至今尚未有效地实现，以实际用于自动程序验证的目的。其关键问题是时态逻辑定理机器证明尚待进一步研究。

* 八六三计划资助项目
1993年2月23日收稿，1993年10月3日修改。

1 命题时态逻辑语法和语义

1.1 语法

命题常元：true，false

命题符号：p，q，r，…

连接词：$\neg$，$\wedge$，$\vee$

时态算子：$\Box$（总是），$\Diamond$（可能），$\bigcirc$（下一时刻），$\cup$（直到）

合式公式：(1) true，false 和命题符号是 wffs，

(2) 如果 u，v 是 wffs，则 ($\neg$u)，(u$\wedge$v)，(u$\vee$v)，($\Box$u)，($\Diamond$u)，($\bigcirc$u)，(u$\cup$v) 也是 wffs，

(3) 合式公式仅为 (1)、(2) 所构造。

1.2 语义

定义 1 模型是三元组 (M，S，$s_0s_1\cdots$)，这里，S 是状态集，M：对每一状态都指派一命题符号子集，该集合中元素在该状态下为真，$s_0s_1\cdots$是 S 中无穷状态序列，满足关系"$\models$"定义如下：

(M，S，$s_i\cdots$) $\models$true，对所有 i$\geqslant$；

not (M，S，$s_i\cdots$) $\models$false，对所有 i$\geqslant$0；

(M，S，$s_0\cdots$) $\models$p，iff p$\in$M (s_0)，这里 p 是命题符号

(M，S，$s_0\cdots$) $\models$ u，iff not (M，S，$s_0\cdots$) $\models$u

(M，S，$s_0\cdots$) $\models$u$\vee$v，iff 或者 (M，S，$s_0\cdots$) $\models$u 或者 (M，S，$s_0\cdots$) $\models$v

(M，S，$s_0\cdots$) $\models$u$\wedge$v，iff (M，S，$s_0\cdots$) $\models$u 并且 (M，S，$s_0\cdots$) $\models$v

(M，S，$s_0\cdots$) $\models\bigcirc$u，iff (M，S，$s_i\cdots$) $\models$u

(M，S，$s_0\cdots$) $\models\Box$u，iff 对所有 i$\geqslant$0，(M，S，$s_i\cdots$) $\models$u

(M，S，$s_i\cdots$) $\models\Diamond$u，iff 存在 i$\geqslant$0，(M，S，$s_i\cdots$) $\models$u

(M，S，$s_i\cdots$) $\models$u$\cup$v，iff 存在 i$\geqslant$0，(M，S，$s_i\cdots$) $\models$v 并且对满足 0$\leqslant$j$<$i 的所有 j，(M，S，$s_i\cdots$) $\models$u。

定义 2 存在一模型，即存在 i，(M，S，$s_i\cdots$) $\models$u，称 u 可满足，若对所有模型，即对所有 M，对所有 i，(M，S，$s_i\cdots$) $\models$u，称 u 逻辑有效，有时简称 u 真。

2 命题时态逻辑定理证明器的实现

以文 [10] 中命题时态逻辑相继式演算为基础，我们用 Turbo－Prolog 语言编程，在 386 微机上很好地实现了命题时态逻辑定理证明器。

该证明器，界面友好，输入直观方便，速度快，能够证明目前在文章和著作中所遇到的符合第 1 节语法的所有命题时态逻辑定理。能够快速证明文 [4] 中专门用来测试定理证明器的关于命题逻辑的全部定理。

在实现的同时，我们用 GKD－Prolog 编程，将上述证明器移植到 SUN 工作站及我系研制的银河智能工具机上。在银河 智能工具机鉴定时，专家们观看了该证明器的演示。

值得指出的是，我们针对时态逻辑中的 next 算子、归纳、归结、until、$\Diamond$ (n$\wedge$v) 等

关键问题，提出了独到的解决办法，从而为定理证明器的成功实现铺平了道路。

我们选用 Prolog 做为编程语言，基于下面几点考虑：

(1) Prolog 本身做为一种成熟的逻辑程序设计语言，集成了许多成熟的定理机器证明技术。

(2) 描述能力强，可集中精力考虑高层控制，无需花精力研究低层实现。

(3) 目前，许多证明器的实现，也是采用 Prolog 编程。

(4) 可以比较自然地对 Prolog 表达能力进行延伸。

(5) 针对 Turbo－Prolog，它提供了完善的界面描述函数，实现了动态关系数据库，为我们具体实现提供诸多便利

2.1 证明器流程图(见图 1)

2.2 证明器的实现技术

2.2.1 界面

我们在 386 微机上实现的 PTL 证明器，具有良好的输入、输出界面。

(1) 输入界面

创建窗口，窗口顶端提“Input a formula that you want to prove:”窗口底部显示中止证明、列定理、返回上级主菜单的提示符：

<ESC> to quit，< * > LIST THM，(!) to MAIN MENU

以及替代符号信息：－(否定)，&(合取)，#(析取)，<－>(等价)，@(下一时刻)，[](总是)，〈〉(终将)，$(直到)

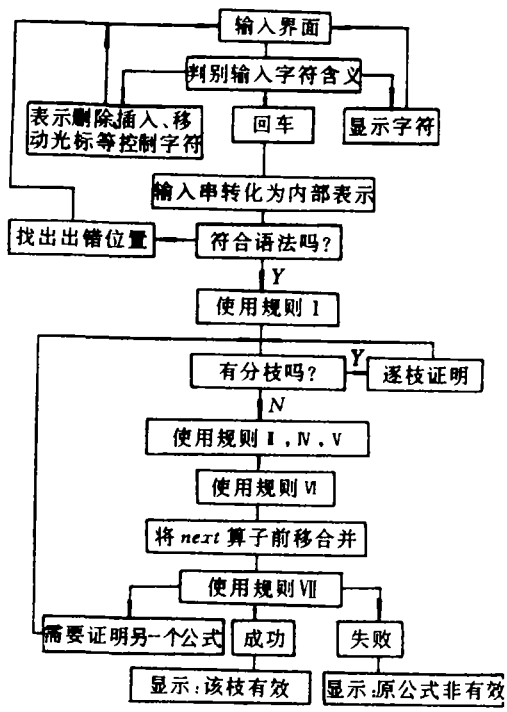

图 1

(2) 输入语言

PTL 证明器的目的，是判断 PTL 公式的有效性，因此，我们选择通常手写中缀形式做为输入语言。例如：

[] [] @p→ 〈〉 p

考虑到公式输入容易出错，因而要求方便修改，要求空格不影响内部表示等等。回车键、终止程序键 (ESC)，光标左移 (^ S)，光标右移 (^ D)，删除当前字符 (^ U) 相应的 ASC Ⅱ 码分别为 13，27，19，4，21，在程序中，利用“\n”(n 为 ASC Ⅱ 码) 表示相应键入字符，经过匹配，实施相应操作。

如果键入字符的 ASC Ⅱ 码介于 32～126 之间，则屏幕显示相应字符。

(3) 输出

在证明过程中，我们将分支信息，显示在屏幕上，例如，“First branch，to prove：α”，公式 α 若以内部方式表示，则既长，又难懂。为此，将前缀形式转化为中缀形式。符

号→，¬←→，u 对应的 ASC Ⅱ 码分别为 26，170，29，150，由此可产生直观易懂的中缀形式。

如果某枝求证失败，则界面显示："此枝无效"，"待证公式无效"，并伴有声音提示。如果某枝求证成功，则显示"该枝有效"，"按任一键，继续……"，所有枝均成功，则最终显示"待证公式有效"，并伴有声音提示。

对后一种情况，按任一键后，右半屏幕分屏显示已证过的定理（及编号），让用户看是否有必要加入内部定理数据库；回车后，提示输入定理编号；若键入 0，则意味着不加入内部定理数据库；若为一正整数，则在下一次列定理时，我们将会看到刚刚的定理及相应编号。

按"!"键，返回主菜单。主菜单包括：装入定理数据库文件，将内部定理数据库做文件存储，以及我们实现了的命题模态逻辑 S_4，S_5，部分实现了的一阶时态逻辑定理证明器，以及等价演算快速证明器。这里，用到了 Turbo－prolog 所提供的模块化程序设计技术及菜单技术。我们已将所有这些模块做工程连接，形成了一个完整的可执行文件。

2.2.2 编译器

通过编译器，将输入语言（时态公式）转变为内部形式，并测试输入语言的合法性。

内部表达式定义为

EXP＝var (STRING)；con (t)；con (f)；disj (EXP，EXP)；conj (EXP，EXP)；

imp (EXP，EXP)；equi (EXP，EXP)；not_ (EXP)；next (INTEGER，EXP)；

always (EXP)；eventu (EXP)；until (EXP，EXP)

命题变元的名字可以是大写字母，或小写字母，也可以是以一个字母开头．后跟一个或多个字母、数字或下划线组成的串。

首先，将输入串转化为由词（token）组成的表，然后，通过匹配 &，#，@等符号，变成相应内部表达式。

在串转化为内部表达式过程中，用一表存储多余括号、不可匹配符号等。若此表非空，则原输入不符合语法。在此情况下，通过计算，确定出错位置，激活光标，指出错误。

必须指出，我们的编译器，采用了适当技术，可自动识别一元连接词和一元时态算子的迭代。例如，直接输入 ¬□◇p→◇□ ¬p 经编译器，可自动认别为（ ¬（□（◇p））） →（◇（□（ ¬p）））。

此外，对二元连接词，我们设定 ∧ 优先级最高，其次是 ∨，最后是→，←→，这样，输入时可省略不必要的括号。例如：输入 p∧○q∨r→□s 自动识别为（（p∧○ q） ∨ r）→（□ s）。

2.2.3 化简及转换

（1）元规则。欲证□α，只须证 α，其 Prolog 语句表示为：

theorem (always (EXP))：－prove（ []，[]，[]，[EXP]),!

（2）分枝规则。欲证 $(\alpha \vee \beta) \wedge \gamma \rightarrow \delta$，必须证 $\alpha \wedge \gamma \rightarrow \delta$ 和 $\beta \wedge \gamma \rightarrow \delta$，用 Prolog 语句可表示为

prove (L1，[disj (X，Y) |L2]，L3，L4)：-!，

first_branch，to_prove (L1，[X|L2]，L3，L4)，branch_proved，

second_branch，to_prove (L1，[Y|L2]，L3，L4)，branch_proved.

(3) 内部化简、转换。这里的化简，对应使用文 [10] 中的基本规则，时态等价化简规则。这里的转换，对应使用文 [10] 中的时态等价转换规则。

(4) 将 next 算子前移合并

该过程，必须考虑两个公式集合中的所有公式，而实现这个过程．必须对每个公式分别考虑。在 next 算子前移过程中，应同时考虑化简问题。

例如，将公式○□○□○◇○◇u 中 next 算子前移，则需要逐层考虑，考虑深度≤4，第一次将式中第二个 next 前移合并，得$○^2$□□○◇○◇u，此时，须将两个 alwsys 化简，如果不化简，则程序必然要考虑任意多层 always，这样做是不适当的。化简后，得$○^2$□○◇○◇u，再试图将 next 前移，得$○^3$□◇○◇u，最终，得$○^4$□◇u。

2.2.4 有效性判断

若使用有效性判断规则成功，则意味着相应分枝证明成功。有时，将 L 中公式所蕴涵的信息加入，重新求证。如果使用有效性判断规则，无法判定成功，则认为失败。只要一枝非有效，则无需考虑其它分枝，屏幕显示“所要证的公式非有效”。

我们知道，经典命题逻辑 Gentzen 演算的有效性判断，只有一条：判断前件公式集与后件公式集是否有共同公式。经典命题逻辑的这种判断规则，在命题时态逻辑中仍适用，但是，除此之外，必须增加许多判定规则。

下述情况之一成立，则相继式 L1，[] ==〉 R1，[] 有效。

(1) 处理□，◇，○

①$○^i$u∈L1，◇u∈R1 ②□u∈L1，$○^i$□u∈R1 ③$○^i$◇u∈L1，◇u∈R1 ④□u∈L1，$○^i$◇u∈R1 ⑤□u∈L1，$○^i$u∈R1 ⑥□ (u→v) ∈L1，□u∈L1，□v∈R1 ⑦L1 中同时含 u，○□u，则用□u 替代之后，继续证明。 ⑧R1 中同时含 u，○◇u，则用 ◇u 替代之后，继续证明。 ⑨L1 中同时含□u，◇v，且 ¬u∨¬v 可证。 ⑩R1 中同时含□u，◇v，且 u∨v 可证。

(2) 处理归纳模式

前件中含归纳模式□ (u→○u) 或□ (u→○◇u)，则分别在前件中加入被蕴涵的公式 (u→□u) 和 (◇u→□◇u)。

对于 u 为文字的情形，归纳规则易于理解，也容易实现。如果 u 不是文字，而是一复合公式并且 (u→○u)，(u→○◇u) 已做了变形，比如：(p∧q) →○ (p∧q) 转化为 (¬ [p∨¬q∨○p) ∧ (¬p∨¬q∨○q)，则必须设计处理归纳的子程序。

必须注意：□ (x→○x) 蕴涵 (x→□x)，反之不成立。这是因为从□ (x→○x) 与○○x 可推○○○x，但从 ○○x 与 (x→□x) 推不出○○○x，□ (x→○◇x) 蕴涵 (◇x→□◇x)，反之不成立，因为○x 与前者可推$○^2$◇x，○x 与 (◇x→□◇x) 推不出$○^2$◇x。

(3) 处理归结模式

归结区分为两种基本情况：

①□ (u∨v) ∈L1，$○^i$u∈R1 ②□ (¬u∨v) ∈L1，$○^i$u∈L1

若上述两种情形之一被满足，则在 L2 中加入$\bigcirc^i v$，继续证明。

由$\Box(x\rightarrow y)$和$\bigcirc^i x$可归结得$\bigcirc^i y$，但是，$\Box(x\rightarrow y)$和$\bigcirc^i x$仍须保留。例如（1）欲从$p\wedge\Box(p\rightarrow\bigcirc^6 p)$推出$\bigcirc^{18}p$；第一次归结得$\bigcirc^6 p$，删去$\Box(p\rightarrow\bigcirc^6 p)$后，推不出$\bigcirc^{18}p$；（2）欲从$p\wedge\Box(p\rightarrow q)\wedge\Box(p\rightarrow r)$推出 r，第一次归结得 q，删去 p，则推不出 r。但保留时，必须做技术上的考虑。例如，欲从$p\wedge\Box(p\rightarrow q)\wedge\Box(q\rightarrow r)\wedge\Box(r\rightarrow\neg p)$导出 false，若保留，则可能出现两项重复归结，导致死循环的情形。为此，我们对归结过的公式对做标记，不允许重复归结。

归结规则的使用，是很复杂的。特别，这里还涉及到自然数的运算。例如，证明$p\wedge\Box(p\rightarrow\bigcirc^5 p)\rightarrow\bigcirc^{100}p$需要做 20 次归结，才给出“valid”结果。

（4）处理$\Diamond(x\wedge y)$

形如$\bigcirc^i\Diamond(x\wedge y)$的公式做为前提，在证明过程中，有时需要 x 或 y 的有关信息，$\bigcirc^i\Diamond(x\wedge y)$指存在$j\geqslant 0$，$\bigcirc^{i+j}x\wedge\bigcirc^{i+j}y$，但是，在 i 确定，j 非确定的情况下，机器不可能由$j\geqslant 0$推$i+j\geqslant i$，更不可能让 i+j 去与别的自然数相比较。机器需要具体自然数。在实现时，出于技术上的考虑，我们设 i 为 1000，只要所涉及时态公式中 next 算子的幂远小于 1000，则这不影响我们的证明。我们设 j=1000，仅表明 j 的存在性，而◇恰恰也只说明存在性，这样 i+j 就是一确定自然数，可做有关大小比较。

（5）处理 until

处理 until 区分为四种基本情况：

①$u\cup v\in L1$，$v\in R1$，$\bigcirc(u\cup v)\in R1$

②$u\cup v\in R1$，$u\in L1$，$\bigcirc(u\cup v)\in L1$

③$u\cup v\in L1$且可证$[],[v\vee(u\wedge\bigcirc w)]\Longrightarrow[],[w]$

其中 w 为 R11 中公式之析取，R11 中的公式是 R1 中的公式，并且 R11 中公式与 u 或 v 有共同变元。

④$u\cup v\in R1$且可证

第一枝$[],[w]\Longrightarrow[],[\Diamond v]$

第二枝$[],[w]\Longrightarrow[],\{v\vee(u\wedge\bigcirc w)]$

其中 w 为 L11 中公式之合取，L11 中的公式是 L1 中的公式，并且 L11 中公式与 u 或 v 有共同变元。

情况③④，通过使用不含 until 的公式，替代 until 公式，来判断有效性。在处理 until 子程序中，为避免出现“后产生的相继式比原式复杂”的情况，我们仅处理前、后件集合中含共同变元的公式。

2.3 归结方法的进一步完善

完全性通常指“真的”一定“可证”。可判定指的是：存在一算法，“真的”给出答案“valid”，“不是真的”给出答案“not valid”。

假设输入公式$(p\wedge\Box(p\rightarrow\bigcirc^5 q)\wedge q\wedge\Box(q\rightarrow\bigcirc^5 p))\rightarrow\bigcirc^{101}p$，按上述算法，不断归结，产生$\bigcirc^5 q$，$\bigcirc^{10}p$，$\bigcirc^{15}q$，$\bigcirc^{20}p$，…，无穷尽地做下去并总是与$\bigcirc^{101}p$做比较，直到机器产生堆栈溢出，还没有给出答案。如此看来，上述算法是半可判定的。

我们知道，PTL 具有小模型性质。如果长为 m 的 PTL 公式可满足，则它在大小至

多为 k^m 的一结构中可满足，其中 k 是固定的整数。因此，对于小于 k^m 状态的所有模型做检查，必可得到满足性答案。由此可见，PTL 理论上可判定。

我们采用如下技术可克服上述问题的不终止性。引进记号 [i] p 表示 p 在 s_0，s_i，s_{2i}，…，s_{ki}，…，状态为真，这样，$\bigcirc^5 q$，$\bigcirc^{15} q$，$\bigcirc^{25} q$，…可记为 $\bigcirc^5$ [10] q. 扩充上面算法，使之能从 $(p \wedge \Box (p \rightarrow \bigcirc^5 q) \wedge q \wedge \Box (q \rightarrow \bigcirc^5 p))$ 导出 $\bigcirc^{10}$ [10] $p \wedge \bigcirc^5$ [10] q，因为 101mod 10≠0，从而证明推不出 $\bigcirc^{101} p$。

需要说明的是，我们在调试定理证明器过程中，遇到了定理不可证的情况，借助 Turbo－Prolog 的追踪功能，均找出了问题的症结，解决了问题。我们并不认为目前的程序是最终程序。

王兵山教授在此项研究中，一直给予关心和指导，作者在此表示衷心感谢。

参 考 文 献

1 Abadi M. The power of temporal proofs. Theoretical Computer Science 1989, 65: 35～83

2 Abadi M, Manna Z. Nonclause deduction in first－order temporal logic. J of ACM, 1990, 37 (2): 279～217

3 Cavalli A R, Farinas Del Cerro L. A decision method for linear temporal logic. In: Proc 7th International Conference on Automated Deduction, LNCS 170, 1984: 113～127

4 Pelletier F J. Seventy－five problems for testing automatic theorem provers. J of Auto Reas 1986, 2: 191～216

5 Plaisted. D A. A decision procedure for combinations of propositional temporal logic and other specialized theories. J of Automated Reasoning 1986, 2: 171～190

6 Venkatesh G A. decision method for temporal logic based on resolution, In: Foundations of Software Technology and Theoretical Computer Science, 5th Conference, 1985, LNCS 206: 272～289.

7 Wolper p, Temporal logic can be more expressive. Information and Control 1983, 56: 72－99

8 贲可荣. 陈火旺. 自动定理证明：十年回顾，计算机科学，1993，20 (4)

9 贲可荣，陈火旺. 命题时态逻辑定理证明新方法，软件学报，1994，5 (5)

10 贲可荣，陈火旺. 命题时态逻辑相继式演算系统，中国科学

The Techniques of PTL Prover

Ben Kerong　Chen Huowang

(Department of Computer Science, Changsha, NUDT, 410073

Abstract

Based on the theory of [10], the propositional temporal logic (PTL) prover has been implemented in Turbo－Prolog and runs on 386 microcomputers. The PTL prover has some original features in dealing with *the power of next*, *induction*, *resolution*, $\diamond (x \wedge y)$, *until*. The prover' s many full-screen facilities make it easy to use. The prover appears to be of reasonable efficiency for most current examples.

Key words Temporal logic, theorem proving, automated reasoning