中国科学院大学

University of Chinese Academy of Sciences

博士学位论文

基于模型检测和定理证明的形式化验证：基础理论与相关工具

作者姓名：刘坚

指导教师：蒋颖 研究员

中国科学院软件研究所

学位类别：工学博士

学科专业：计算机软件与理论

培养单位：中国科学院软件研究所

2018 年 6 月

Towards Combing Model Checking and Theorem Proving: Theory and Related Tools

A dissertation submitted to the
University of Chinese Academy of Sciences
in partial fulfillment of the requirement
for the degree of
Doctor of Philosophy
in Computer Software and Theory

By

Jian Liu

Supervisor: Professor Ying Jiang

Institute of Software, Chinese Academy of Sciences

June, 2018

中国科学院大学
研究生学位论文原创性声明

本人郑重声明：所呈交的学位论文是本人在导师的指导下独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明或致谢。本人完全意识到本声明的法律结果由本人承担。

作者签名：

日　　期：

中国科学院大学
学位论文授权使用声明

本人完全了解并同意遵守中国科学院大学有关保存和使用学位论文的规定，即中国科学院大学有权保留送交学位论文的副本，允许该论文被查阅，可以按照学术研究公开原则和保护知识产权的原则公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存、汇编本学位论文。

涉密及延迟公开的学位论文在解密或延迟期后适用本声明。

作者签名：　　导师签名：

日　　期：　　日　　期：

摘 要

模型检测和逻辑推演是目前用于形式化验证系统正确性的主要的两种方法。模型检测的优点是可以实现完全自动化，验证过程不需要人工干预，缺点是在处理大型系统或者无穷状态系统的过程中通常会面临状态爆炸问题；逻辑推演的优点是通常不关心系统的状态，从而避免了状态爆炸问题的产生，而缺点则是通常不能实现完全自动化，在验证的过程中通常需要人工干预。因此，如何结合模型检测和逻辑推演两种方法的优点来建立新的形式化验证方法是本文的主要研究内容。

本文的第一个工作是，建立一个逻辑系统CTL_P，CTL_P以Kripke模型为参数，对于一个给定的Kripke模型$\mathcal{M}$，一个$\mathsf{CTL}_P(\mathcal{M})$公式是有效的当且仅当这个公式在M中是满足的。相比于在CTL逻辑中只能讨论模型中当前状态的性质，在CTL_P中我们可以定义模型中的状态之间的关系，从而丰富了CTL逻辑的表达性；然后，我们根据逻辑CTL_P建立了一个证明系统SCTL（Sequent-calculus-like proof system for CTL_P），并证明了SCTL系统的可靠性和完备性，使得一个公式在SCTL中是可证的当且仅当它在给定的模型中是满足的。

本文的第二个工作是，提出了一个SCTL证明系统的工具实现SCTLProV，SCTLProV既可以看作为定理证明器也可看作为模型检测工具：相比于定理证明器，SCTLProV可以应用更多的优化策略，比如利用BDD（Binary Decision Diagram）来存储状态集合从而减小内存占用；相比于模型检测工具，SCTLProV的输出是完整的证明树，比模型检测工具的输出更丰富。

本文的第三个工作是，实现了一个3D证明可视化工具VMDV（Visualization for Modeling, Demonstration, and Verification）。VMDV目前可以完整的显示SCTLProV的证明树以及高亮显示SCTLProV的证明过程。同时VMDV是一个一般化的证明可视化工具，并提供了不同的接口用来与不同的定理证明器（比如coq）协同工作。

关键词： 模型检测，定理证明，工具实现

Abstract

This paper is a help documentation for the LaTeX class ucasthesis, which is a thesis template for the University of Chinese Academy of Sciences. The main content is about how to use the ucasthesis, as well as how to write thesis efficiently by using LaTeX.

Keywords: University of Chinese Academy of Sciences (UCAS), Thesis, LaTeX Template

目 录

图形列表

表格列表

第1章　定理证明与模型检测的结合

本章首先介绍逻辑系统CTL$_P$，CTL$_P$是计算树逻辑CTL的一个扩展；然后介绍针对CTL$_P$的一个证明系统SCTL; 最后介绍针对证明系统SCTL的一个证明搜索策略及其伪代码。

1.1　CTL$_P$

我们用逻辑 CTL$_P(\mathcal{M})$ 来刻画要验证的系统 $\mathcal{M}$ 的性质，其中 $\mathcal{M}$ 通常指的是一个 Kripke 结构，其定义如下。

定义 1.1 (Kripke 结构)**.** 一个 *Kripke* 结构 $\mathcal{M} = (\mathcal{S}, \longrightarrow, \mathcal{P})$ 包含如下三个部分：

1. S 是一个有穷的状态集合；
2. $\longrightarrow \subseteq S \times S$ 是一个一个二元关系；对于每一个状态 $s \in S$，至少存在一个 $s' \in S$ 使得 $s \longrightarrow s'$;
3. $\mathcal{P}$ 是一个有穷的关系符号的集合；对于每个关系符号 $P \in \mathcal{P}$，都存在自然数 n 使得$P \in S^n$ 。

对于一个状态 $s \in S$，我们将 s 的所有的下一个状态的集合定义为

$$\text{Next}(s) = \{s' \mid s \longrightarrow s'\}。$$

一个路径是一个有穷或无穷的状态序列，通常形式为 $s_0, \ldots, s_n$ 或者 $s_0, s_1, \ldots$ ，其中，对于任意自然数i，如果 s_i 不是该序列的最后一个元素，那么就有 $s_{i+1} \in \text{Next}(s_i)$ 。

我们称 T 是一棵路径树当且仅当对于 T 上的所有由 s 标记的非叶子节点，该节点的所有后继节点正好由 Next(s) 中的所有元素一一标记。一棵路径树上的所有节点既可以是有穷个也可以是无穷个。

语法。　一个Kripke 结构 $\mathcal{M}$ 的性质由 CTL$_P(\mathcal{M})$ 公式表示：

定义 1.2. 对于一个给定的*Kripke*模型 $\mathcal{M} = (S, \longrightarrow, \mathcal{P})$，$CTL_P(\mathcal{M})$公式的语法定义如下：

$$\phi := \begin{cases} \top \mid \bot \mid P(t_1, ..., t_n) \mid \neg P(t_1, ..., t_n) \mid \phi \wedge \phi \mid \phi \vee \phi \mid \\ AX_x(\phi)(t) \mid EX_x(\phi)(t) \mid AF_x(\phi)(t) \mid EG_x(\phi)(t) \mid \\ AR_{x,y}(\phi_1, \phi_2)(t) \mid EU_{x,y}(\phi_1, \phi_2)(t) \end{cases}$$

其中，x 与 y 为变量，取值范围为 S，而 $t_1, ..., t_n$ 既可以是代表状态的常量，也可以是取值范围为 S 的变量。

在定义 1.2 中，我们用模态词来绑定公式中的变量。比如，模态词 AX，EX， AF 以及 EG 在公式 ϕ 中绑定了变量 x；而模态词 AR 和 EU 则在公式 ϕ_1 和 ϕ_2 中分别绑定了变量 x 和 y. 变量的替换则写为 $(t/x)\phi$，表示将公式 ϕ 中所有自由出现的变量 x 都替换为 t。

不失一般性地来说，我们假定所有的否定符号都出现在原子命题上；而且有如下缩写：

- $\phi_1 \Longrightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$,
- $EF_x(\phi)(t) \equiv EU_{z,x}(\top, \phi)(t)$,
- $ER_{x,y}(\phi_1, \phi_2)(t) \equiv EU_{y,z}(\phi_2, ((z/x)\phi_1 \wedge (z/y)\phi_2))(t) \vee EG_y(\phi_2)(t)$, 其中变量 z 既不在 ϕ_1，也不在 ϕ_2 中出现，
- $AG_x(\phi)(t) \equiv \neg(EF_x(\neg\phi)(t))$,
- $AU_{x,y}(\phi_1, \phi_2)(t) \equiv \neg(ER_{x,y}(\neg\phi_1, \neg\phi_2)(t))$.

我们称模态词 AF，EF，AU，以及 EU 为*归纳模态词*；模态词 AR，ER，AG，以及 EG 为*余归纳模态词*。

语义。 相应地，对于一个给定的 Kripke 模型 $\mathcal{M}$，$\mathsf{CTL}_P(\mathcal{M})$的语义定义如下：

- $\mathcal{M} \models P(s_1, ..., s_n)$：如果$\langle s_1, ..., s_n\rangle \in P$，而且$P$是一个$\mathcal{M}$上的$n$元关系；
- $\mathcal{M} \models \neg P(s_1, ..., s_n)$：如果$\langle s_1, ..., s_n\rangle \notin P$，而且$P$是一个$\mathcal{M}$上的$n$元关系；
- $\mathcal{M} \models \top$ 永远成立；
- $\mathcal{M} \models \bot$ 永远不成立；

- $\mathcal{M} \models \phi_1 \wedge \phi_2$：如果$\mathcal{M} \models \phi_1$和 $\mathcal{M} \models \phi_2$同时成立；
- $\mathcal{M} \models \phi_1 \vee \phi_2$：如果$\mathcal{M} \models \phi_1$成立，或者 $\mathcal{M} \models \phi_2$成立；
- $\mathcal{M} \models AX_x(\phi_1)(s)$：如果对于每个状态$s' \in \mathsf{Next}(s)$，都有$\mathcal{M} \models (s'/x)\phi_1$成立；
- $\mathcal{M} \models EX_x(\phi_1)(s)$：如果存在一个状态$s' \in \mathsf{Next}(s)$，使得$\mathcal{M} \models (s'/x)\phi_1$成立；
- $\mathcal{M} \models AF_x(\phi_1)(s)$：如果存在一个有无穷个节点的树$T$，而且 T的根节点是s,那么对于T的任何一个非叶子节点s'，s'的子节点为$\mathsf{Next}(s')$，对于T的任何一个叶子节点s'，$\vdash (s'/x)\phi_1$成立；
- $\mathcal{M} \models EG_x(\phi_1)(s)$：如果存在$\mathcal{M}$上的一个无穷路径 $s_0, s_1, \ldots$（其中 $s_0 = s$），那么对于任意的自然数 i，都有 $\mathcal{M} \models (s_i/x)\phi_1$成立;
- $\mathcal{M} \models AR_{x,y}(\phi_1, \phi_2)(s)$：如果存在一棵路径树 T，T 的根节点由 s 标记，对于任意节点 $s' \in T$ 都有 $\mathcal{M} \models (s'/y)\phi_2$ 成立，而且对于任意的叶子节点 $s'' \in T$ 都有 $\mathcal{M} \models (s''/x)\phi_1$ 成立；
- $\mathcal{M} \models EU_{x,y}(\phi_1, \phi_2)(s)$：如果存在一个无穷路径 $s_0, s_1, \ldots$（其中 $s_0 = s$）和一个自然数 j，$\mathcal{M} \models (s_j/y)\phi_2$ 成立，而且对于任意的自然数 $i < j$ 都有 $\mathcal{M} \models (s_i/x)\phi_1$ 成立。

CTL vs. CTL$_P$。 在计算树逻辑（CTL）[29, 30] 的语法中，原子公式通常用命题符号来表示，而命题符号在计算树逻辑的语义中通常解释为一个 Kripke 结构上的状态集合。在逻辑系统CTL$_P$中，相比于计算树逻辑，我们通过引入多元谓词来增加逻辑系统中公式的表达能力。CTL$_P$相比于CTL的表达能力的提升可由如下的例子表示出来：

例子 1.1. 本例子受多机器人路径规划系统[31, 32]启发。在原例子中，多机器人路径规划系统的规范可以写成 CTL 公式：在一个多个区块的地图上，每从初始位置出发的机器人都能到达指定的最终位置，而且在行进的同时，每个机器人都会避免经过某些位置。

在本例子中，除了CTL所能表示的时序性质之外，我们考虑一种“空间”性质，即表示状态之间的关系。

假定有一个无人车正在一个星球表面行驶，这个星球的表面已经被分成了有穷个小的区域。无人车一次能从一个区域行走到另一个区域，那么我们将无人车的位置看作成一个状态，无人车所有的可能的所在位置则可看作为状态空间，而且无人车从一个位置到另一个位置的移动规律则可看作成迁移关系。无人车的设计需要满足一个基本的性质，即无人车不能永远在一个很小的范围内移动。准确地说，对于给定的距离 σ，在任意状态 s，随着无人车的移动会到达状态 s'，使得 s 和 s' 的位置之间的距离大于 σ。该性质可以由公式 $AG_x(AF_y(D_\sigma(x,y))(x))(s_0)$ 来刻画，其中 s_0 是初始状态，即无人车的降落点；原子公式 $D_\sigma(x,y)$ 则刻画了一种空间性质，即状态 x 和 y 的位置的距离大于 σ。

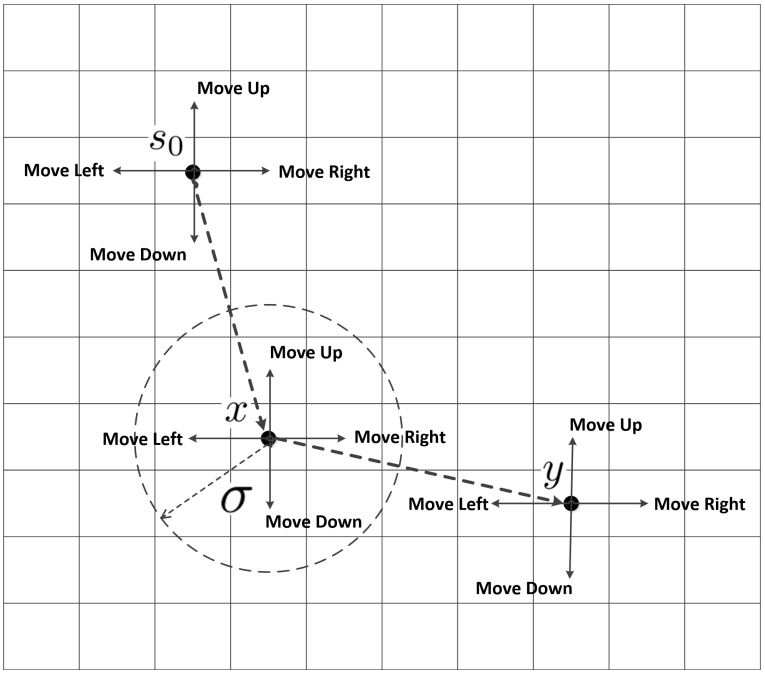


图 1.1 无人车可能的所在位置

例子1.1中的性质可以很容易由 CTL_P 中的公式进行刻画，然而很难用传统的时序逻辑的公式进行表示。原因是在传统的时序逻辑的语法中通常没有表述一个特定的状态或者多个状态之间的关系的机制，即使在语义中，传统的时序逻辑通常只考虑当前的状态，而无法考虑多个状态之间的关系。

1.2 CTL_P的证明系统：SCTL

在本节，我们针对逻辑 $\mathsf{CTL}_P(\mathcal{M})$ 给出一个证明系统 $\mathsf{SCTL}(\mathcal{M})$（Sequent-calculus-like proof system for CTL_P）。在通常意义下的证明系统中，一个公式是可证的当且仅当该公式在所有的模型中都成立，而在 $\mathsf{SCTL}(\mathcal{M})$ 中，一个公式是

可证的当且仅当该公式在模型 $\mathcal{M}$ 中是可证的。

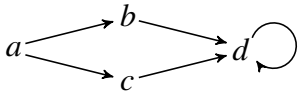
首先，让我们考虑一个$\mathsf{CTL}_P(\mathcal{M})$公式 $AF_x(P(x))(s)$。该公式在模型 $\mathcal{M}$ 中成立当且仅当存在一个路径树 T，T 的根节点由 s 标记，而且 T 上的每个叶子节点都满足 P。

然后，我们考虑一个具有嵌套模态词的 $\mathsf{CTL}_P(\mathcal{M})$ 公式 $AF_x(AF_y(P(x,y))(x))(s)$。如果试图说明该公式在模型 $\mathcal{M}$ 中是成立的，那么就需要找到一个路径树 T，使得 T 的根节点由 s 标记，而且对于 T 中的所有叶子节点 a，$AF_y(P(a,y))(a)$ 是成立的。为了说明 $AF_y(P(a,y))(a)$ 是成立的，则需要又找到一棵路径树 T' 使得 T' 的根节点由 a 标记，而且 T' 上的所有叶子节点 b 都满足 $P(a,b)$。我们可以用以下的两个规则来刻画当前的嵌套的路径树。

$$\frac{\vdash (s/x)\phi}{\vdash AF_x(\phi)(s)}\ \mathbf{AF}\text{-}\mathsf{R}_1$$

$$\frac{\vdash AF_x(\phi)(s_1) \quad \ldots \quad \vdash AF_x(\phi)(s_n)}{\vdash AF_x(\phi)(s)}\ \mathbf{AF}\text{-}\mathsf{R}_2\ \{s_1,\ldots s_n\}=\mathsf{Next}(s)$$

例子 1.2. 假设一个模型有如下图所示的迁移规则，



和一个原子谓词 $P=\{b,c\}$，那么公式 $AF_x(P(x))(a)$ 的一个证明如下。

$$\dfrac{\dfrac{\dfrac{}{\vdash P(b)}\ atom\text{-}R}{\vdash AF_x(P(x))(b)}\ \mathbf{AF}\text{-}\mathsf{R}_1 \qquad \dfrac{\dfrac{}{\vdash P(c)}\ atom\text{-}R}{\vdash AF_x(P(x))(c)}\ \mathbf{AF}\text{-}\mathsf{R}_1}{\vdash AF_x(P(x))(a)}\ \mathbf{AF}\text{-}\mathsf{R}_2$$

在此证明树中，除了 $\mathbf{AF}$-R_1 和 $\mathbf{AF}$-R_2，我们还应用了如下规则。

$$\frac{}{\vdash P(s_1,\ldots,s_n)}\ atom\text{-}R\ \langle s_1,\ldots,s_n\rangle\in P$$

例子 1.3. 假设另一个模型，该模型的迁移规则与例子*1.2*中相同，除此之外还有原子谓词 $Q=\{(b,d),(c,d)\}$。公式 $AF_x(AF_y(Q(x,y))(x))(a)$ 的证明如下。

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{Q(b,d)}\ atom\text{-}R}{\vdash AF_y(Q(b,y))(d)}\ \mathbf{AF}\text{-}\mathsf{R}_1}{\vdash AF_y(Q(b,y))(b)}\ \mathbf{AF}\text{-}\mathsf{R}_2}{\vdash AF_x(AF_y(Q(x,y))(x))(b)}\ \mathbf{AF}\text{-}\mathsf{R}_1 \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{Q(c,d)}\ atom\text{-}R}{\vdash AF_y(Q(c,y))(d)}\ \mathbf{AF}\text{-}\mathsf{R}_1}{\vdash AF_y(Q(c,y))(c)}\ \mathbf{AF}\text{-}\mathsf{R}_2}{\vdash AF_x(AF_y(Q(x,y))(x))(b)}\ \mathbf{AF}\text{-}\mathsf{R}_1}{\vdash AF_x(AF_y(Q(x,y))(x))(a)}\ \mathbf{AF}\text{-}\mathsf{R}_2$$

在SCTL中，每个相继式都有$\Gamma \vdash \phi$形式，其中Γ是一个可能为空的SCTL公式集合，ϕ是一个SCTL公式。不同于通常的相继式演算，

So, as all sequents have the form $\vdash \phi$, the left rules and the axiom rule can be dropped as well. In other words, unlike the usual sequent calculus and like Hilbert systems, SCTL is tailored for deduction, not for hypothetical deduction.

As the left-hand side of sequents is not used to record hypotheses, we will use it to record a different kind of information, that occur in the case of co-inductive modalities, such as the modality EG.

Indeed, the case of the co-inductive formula, for example $EG_x(P(x))(s)$, is more complex than that of the inductive one, such as $AF_x(P(x))(s)$. To justify its validity, one needs to provide an infinite sequence starting from s, and each state in the infinite sequence verifies P. However, as the model is finite, we can always restrict to regular sequences and use a finite representation of such sequences. This leads us to introduce a rule, called **EG**-merge, that permits to prove a sequent of the form $\vdash EG_x(P(x))(s)$, provided such a sequent already occurs lower in the proof. To make this rule local, we re-introduce hypotheses Γ to record part of the history of the proof. The sequent have therefore the form $\Gamma \vdash \phi$, with a non empty Γ in this particular case only, and the **EG**-merge rule is then just an instance of the axiom rule, that must be re-introduced in this particular case only.

SCTL($\mathcal{M}$) 的证明规则如图1.2所示。

可靠性与完备性。 命题1.1和命题1.2的作用是将有穷结构转换为无穷结构，这两个命题被用来证明SCTL的可靠性；命题1.3和命题1.4的作用是将无穷结构转换到有穷结构，这两个命题被用来证明SCTL的完备性。

命题 1.1 (有穷状态序列到无穷状态序列)**.** 给定一个有穷的状态序列$s_0, ..., s_n$，其中对于任意$0 \leq i \leq n-1$都有$s_i \longrightarrow s_{i+1}$，而且存在$0 \leq p \leq n-1$使得$s_n = s_p$。那么，一定存在一个无穷的状态序列$s'_0, s'_1, ...$使得$s_0 = s'_0$，而且对于任意$i \geq 0$都有$s'_i \longrightarrow s'_{i+1}$，同时此无穷状态序列中的每个状态都在$s_0, ..., s_n$中。

证明. 本命题所述无穷序列为：$s_0, ..., s_{p-1}, s_p, ..., s_{n-1}, s_p, ...$，其中 $s_0 = s'_0$. □

命题 1.2 (有穷路径树到无穷路径树)**.** 设Φ为一个状态集合，T为一个有穷的路径树，T的每个叶子节点都由某个状态s来标记，其中，$s \in \Phi$；或者存在从T的根

$$\frac{}{\vdash P(s_1,\ldots,s_n)}\ \text{atom-R}_{\langle s_1,\ldots,s_n\rangle\in P} \qquad \frac{}{\vdash \neg P(s_1,\ldots,s_n)}\ \neg\text{-R}_{\langle s_1,\ldots,s_n\rangle\notin P}$$

$$\frac{}{\vdash \top}\ \top\text{-R} \qquad \frac{\vdash \phi_1 \quad \vdash \phi_2}{\vdash \phi_1 \wedge \phi_2}\ \wedge\text{-R} \qquad \frac{\vdash \phi_1}{\vdash \phi_1 \vee \phi_2}\ \vee\text{-R}_1 \qquad \frac{\vdash \phi_2}{\vdash \phi_1 \vee \phi_2}\ \vee\text{-R}_2$$

$$\frac{\vdash (s'/x)\phi}{\vdash EX_x(\phi)(s)}\ \mathbf{EX}\text{-R}_{s'\in\text{Next}(s)} \qquad \frac{\vdash (s_1/x)\phi \quad \ldots \quad \vdash (s_n/x)\phi}{\vdash AX_x(\phi)(s)}\ \mathbf{AX}\text{-R}_{\{s_1,\ldots,s_n\}=\text{Next}(s)}$$

$$\frac{\vdash (s/x)\phi}{\vdash AF_x(\phi)(s)}\ \mathbf{AF}\text{-R}_1 \qquad \frac{\vdash AF_x(\phi)(s_1) \quad \ldots \quad \vdash AF_x(\phi)(s_n)}{\vdash AF_x(\phi)(s)}\ \mathbf{AF}\text{-R}_2{}_{\{s_1,\ldots,s_n\}=\text{Next}(s)}$$

$$\frac{\vdash (s/x)\phi \quad \Gamma, EG_x(\phi)(s) \vdash EG_x(\phi)(s')}{\Gamma \vdash EG_x(\phi)(s)}\ \mathbf{EG}\text{-R}_{s'\in\text{Next}(s)} \qquad \frac{}{\Gamma \vdash EG_x(\phi)(s)}\ \mathbf{EG}\text{-merge}_{EG_x(\phi)(s)\in\Gamma}$$

$$\frac{\vdash (s/y)\phi_2 \quad \Gamma' \vdash AR_{x,y}(\phi_1,\phi_2)(s_1) \ldots \Gamma' \vdash AR_{x,y}(\phi_1,\phi_2)(s_n)}{\Gamma \vdash AR_{x,y}(\phi_1,\phi_2)(s)}\ \mathbf{AR}\text{-R}_1{}_{\{s_1,\ldots,s_n\}=\text{Next}(s),\Gamma'=\Gamma,AR_{x,y}(\phi_1,\phi_2)(s)}$$

$$\frac{\vdash (s/x)\phi_1 \quad \vdash (s/y)\phi_2}{\Gamma \vdash AR_{x,y}(\phi_1,\phi_2)(s)}\ \mathbf{AR}\text{-R}_2 \qquad \frac{}{\Gamma \vdash AR_{x,y}(\phi_1,\phi_2)(s)}\ \mathbf{AR}\text{-merge}_{AR_{x,y}(\phi_1,\phi_2)(s)\in\Gamma}$$

$$\frac{\vdash (s/y)\phi_2}{\vdash EU_{x,y}(\phi_1,\phi_2)(s)}\ \mathbf{EU}\text{-R}_1 \qquad \frac{\vdash (s/x)\phi_1 \quad \vdash EU_{x,y}(\phi_1,\phi_2)(s')}{\vdash EU_{x,y}(\phi_1,\phi_2)(s)}\ \mathbf{EU}\text{-R}_2{}_{s'\in\text{Next}(s)}$$

图 1.2 SCTL($\mathcal{M}$)

结点到当前叶子节点的分支上的一个节点，使得该节点同样由s所标记。那么，一定存在一棵可能无穷的路径树T'，而且T'的所有叶子节点都由Φ中的某个状态标记，同时用来标记T'节点的状态都用来标记T的节点。

证明. 令T'的根结点为T的根结点，而且对于T的每个节点的标记s来说，如果$s \in \Phi$，那么s标记T'的叶子节点；否则，s的后继节点分别由Next(s)中的每个元素标记。显然，标记T'中节点的状态都标记T中的节点。 □

命题 1.3 (无穷状态序列到有穷状态序列)**.** 给定一个无穷状态序列$s_0, s_1, \ldots$，其中对于任意$i \geq 0$都有$s_i \longrightarrow s_{i+1}$。那么，一定存在一个有穷的状态序列$s'_0, \ldots, s'_n$，对于任意$0 \leq i \leq n-1$，都存在一个$0 \leq p \leq n-1$, 使得$s'_n = s'_p$，而且$s'_0, \ldots, s'_n$中的所有状态都在$s_0, s_1, \ldots$中出现。

证明. 由于Kripke模型的状态集是有穷的，因此在状态序列$s_0, s_1, ...$一定存在$p, n \geq 0$，使得$s_p = s_n$。本命题所述有穷状态序列即为$s_0, ..., s_n$。 □

命题 1.4 (可能无穷的路径树到有穷路径树). 设Φ为一个状态集合；T为一个可能无穷的路径树，其中T的所有叶子节点都由Φ中的某个状态所标记。那么，一定存在一个有穷的路径树T'，使得对于T'的每个叶子节点的标记s，$s \in \Phi$，或者存在从T'的根结点到该叶子节点的分支上的一个节点，该节点同样由s标记。

证明. 由于Kripke模型的状态集是有穷的，因此对于T的每个无穷分支，都存在$0 \leq p < n$，使得$s_p = s_n$。将T的每个这样的无穷分支在s_n处截断，所得到的路径树即为T'。显然，由于T'具有有穷个分支，同时T'的每个分支都是有穷的，因此T'也是有穷的。 □

定理 1.1 (可靠性). 设$\mathcal{M}$为一个*Kripke*模型，ϕ为一个$CTL_P(\mathcal{M})$闭公式。如果相继式$\vdash \phi$具有一个证明，则$\mathcal{M} \models \phi$成立。

证明. 假设相继式$\vdash \phi$具有证明π，以下对证明π的结构做归纳：

- 如果π的最后一条规则为atom-R，那么$\vdash \phi$具有$\vdash P(s_1, ..., s_n)$形式，因此$\mathcal{M} \models P(s_1, ..., s_n)$。
- 如果π的最后一条规则为$\neg$-R，那么$\vdash \phi$具有$\vdash \neg P(s_1, ..., s_n)$形式，因此$\mathcal{M} \models \neg P(s_1, ..., s_n)$。
- 如果π的最后一条规则为$\top$-R，那么$\vdash \phi$具有$\vdash \top$形式，因此$\mathcal{M} \models \top$。
- 如果π的最后一条规则为$\wedge$-R，那么$\vdash \phi$具有$\vdash \phi_1 \wedge \phi_2$形式。根据归纳假设，$\mathcal{M} \models \phi_1$与$\mathcal{M} \models \phi_2$均成立，因此$\mathcal{M} \models \phi_1 \wedge \phi_2$。
- 如果π的最后一条规则为$\vee$-R，那么$\vdash \phi$具有$\vdash \phi_1 \vee \phi_2$形式。根据归纳假设，$\mathcal{M} \models \phi_1$成立或$\mathcal{M} \models \phi_2$成立，因此$\mathcal{M} \models \phi_1 \vee \phi_2$。
- 如果π的最后一条规则为AX-R，那么$\vdash \phi$具有$\vdash AX_x(\phi_1)(s)$形式。根据归纳假设，对于任意$s' \in \mathsf{Next}(s)$，都有$\mathcal{M} \models (s'/x)\phi_1$成立，因此$\mathcal{M} \models AX_x(\phi_1)(s)$。
- 如果π的最后一条规则为EX-R，那么$\vdash \phi$具有$\vdash EX_x(\phi_1)(s)$形式。根据归纳假设，存在$s' \in \mathsf{Next}(s)$，使得$\mathcal{M} \models (s'/x)\phi_1$成立，因此$\mathcal{M} \models EX_x(\phi_1)(s)$。

- 如果π的最后一条规则为**AF**-R_1或**AF**-R_2，那么$\vdash \phi$具有$\vdash AF_x(\phi_1)(s)$形式。根据证明π，我们利用归纳的方式构造一棵路径树$|\pi|$。构造方式如下：
  - 如果π的最后一条规则为**AF**-R_1，而且ρ为$\vdash (s/x)\phi_1$的证明，则路径树$|\pi|$只包含一个节点s；
  - 如果π的最后一条规则为**AF**-R_2，而且$\pi_1, ..., \pi_n$分别为$\vdash AF_x(\phi_1)(s_1), ..., AF_x(\phi_n)(s_n)$的证明，其中$\{s_1, ..., s_n\} = \mathsf{Next}(s)$，那么令$|\pi|$等于$s(|\pi_1|, ..., |\pi_n|)$。

  路径树$|\pi|$的根结点为s，而且对于$|\pi|$的每个叶子节点s'来说，$\vdash (s'/x)\phi_1$都有一个比π小的证明。根据归纳假设，对于$|\pi|$的每个叶子节点s'来说，都有$\mathcal{M} \models (s'/x)\phi_1$成立，因此，$\mathcal{M} \models AF_x(\phi_1)(s)$成立。

- 如果π的最后一条规则为**EG**-R，则$\vdash \phi$具有$\vdash EG_x(\phi_1)(s)$形式。根据证明π，我们归纳构造一个状态序列$|\pi|$。构造方式如下：
  - 如果π的最后一条规则为**EG**-merge，那么$|\pi|$只包含一个单独的状态s；
  - 如果π的最后一条规则为**EG**-R，而且ρ和π_1分别为$\vdash (s/x)\phi_1$和$\Gamma, EG_x(\phi_1)(s) \vdash EG_x(\phi_1)(s')$的证明，其中$s' \in \mathsf{Next}(s)$，那么令$|\pi|$等于$s|\pi_1|$。

  对于状态序列$|\pi| = s_0, ..., s_n$，$s_0 = s$；对于任意$0 \leq i \leq n-1$，$s_i \longrightarrow s_{i+1}$；对于任意$0 \leq i \leq n$，$\vdash (s_i/x)\phi_1$都有一个比$\pi$小的证明；而且存在$p < n$使得$s_n = s_p$。根据归纳假设，对于任意$i \geq 0$，都有$\mathcal{M} \models (s_i/x)\phi_1$成立。由命题1.1可知，存在一个无穷的状态序列$s'_0, s'_1, ...$，其中对于任意$i \geq 0$都有$s'_i \longrightarrow s'_{i+1}$，同时$\mathcal{M} \models (s'_i/x)\phi_1$成立。因此，$\mathcal{M} \models EG_x(\phi_1)(s)$成立。

- 如果π的最后一条规则为**AR**-R_1或**AR**-R_2，那么$\vdash \phi$具有$\vdash AR_x(\phi_1, \phi_2)(s)$形式。根据$\pi$，我们归纳构造一个有穷的路径树$|\pi|$。构造方式如下：
  - 如果π的最后一条规则为**AR**-R_1，而且ρ_1和ρ_2分别为$\vdash (s/x)\phi_1$和$\vdash (s/x)\phi_2$的证明，那么$|\pi|$只包含一个节点s；
  - 如果π的最后一条规则为**AR**-merge，那么$|\pi|$只包含一个节点s；
  - 如果π的最后一条规则为**AR**-R_2，而且$\rho, \pi_1, ..., \pi_n$分别为$\vdash (s/y)\phi_2, \Gamma, AR_{x,y}(\phi_1, \phi_2)(s) \vdash AR_{x,y}(\phi_1, \phi_2)(s_1), ..., \Gamma, AR_{x,y}(\phi_1, \phi_2)(s) \vdash AR_{x,y}(\phi_1, \phi_2)(s_n)$的证明，其中$\{s_1, ..., s_n\} = \mathsf{Next}(s)$，那么令$|\pi|$等于$s(|\pi_1|, ..., |\pi_n|)$。

路径树$|\pi|$以s为根结点，而且对于$|\pi|$的每个节点s'来说，$\vdash (s'/y)\phi_2$都有一个比π小的证明；对于$|\pi|$的任意叶子节点s'来说，$\vdash (s'/x)\phi_1$有一个比π小的证明，或者在从$|\pi|$的根结点到当前叶子节点的分支上存在一个节点，使得s'标记此节点。根据归纳假设，对于$|\pi|$的任意节点s'，$\models (s'/y)\phi_2$成立，而且对于$|\pi|$的任意叶子节点s'，$\models (s'/x)\phi_1$成立，或者在从$|\pi|$的根结点到当前叶子节点的分支上存在一个节点，使得s'标记此节点。根据命题1.2，存在一个可能无穷的路径树T'，使得对于T'的每个节点s'，都有$\models (s'/y)\phi_2$成立，而且对于T'的每个叶子节点s'，都有$\models (s'/x)\phi_1$成立。因此，$\models AR_{x,y}(\phi_1, \phi_2)(s)$成立。

- 如果π的最后一条规则为**EU**-R_1或**EU**-R_2，那么$\vdash \phi$具有$\vdash EU_{x,y}(\phi_1, \phi_2)(s)$形式。根据$\pi$，我们归纳构造一个有穷状态序列$|\pi|$。构造过程如下：
  - 如果π的最后一条规则为**EU**-R_1，那么$|\pi|$只包含一个状态s；
  - 如果π的最后一条规则为**EU**-R_2，而且ρ和π_1分别为$\vdash (s/x)\phi_1$和$\vdash EU_{x,y}(\phi_1, \phi_2)(s')$的证明，那么令$|\pi|$等于$s|\pi_1|$。

  在状态序列$|\pi| = s_0, ..., s_n$中，$s_0 = s$；对于任意$0 \leq i \leq n-1$，$s_i \longrightarrow s_{i+1}$；对于任意$0 \leq i \leq n-1$，$\vdash (s_i/x)\phi_1$有一个比$\pi$小的证明；而且$\vdash (s_n/y)\phi_2$有一个比$\pi$小的证明。根据归纳假设，对任意$0 \leq i \leq n-1$，$\models (s_i/x)\phi_1$和$\models (s_n/y)\phi_2$均成立。因此，$\models EU_{x,y}(\phi_1, \phi_2)(s)$成立。

- π的最后一条规则不能为merge规则。

□

定理 1.2 (完备性). 设ϕ是一个$CTL_P(\mathcal{M})$闭公式。如果$\mathcal{M} \models \phi$，则$\vdash \phi$在$SCTL(\mathcal{M})$中是可证的。

证明. 对ϕ的结构作归纳：

- 如果$\phi = P(s_1, ..., s_n)$，那么由$\mathcal{M} \models P(s_1, ..., s_n)$可知，$\vdash P(s_1, ..., s_n)$是可证的。
- 如果$\phi = \neg P(s_1, ..., s_n)$，那么由$\mathcal{M} \models \neg P(s_1, ..., s_n)$可知，$\vdash \neg P(s_1, ..., s_n)$是可证的。
- 如果$\phi = \top$，那么显然$\vdash \top$是可证的。

- 如果$\phi = \bot$，那么显然$\vdash \bot$是不可证的。
- 如果$\phi = \phi_1 \wedge \phi_2$，那么由于$\mathcal{M} \models \phi_1 \wedge \phi_2$，因此$\mathcal{M} \models \phi_1$和$\mathcal{M} \models \phi_2$均成立。根据归纳假设，$\vdash \phi_1$和$\vdash \phi_2$均可证。因此，$\vdash \phi_1 \wedge \phi_2$是可证的。
- 如果$\phi = \phi_1 \vee \phi_2$，那么由于$\mathcal{M} \models \phi_1 \vee \phi_2$，因此$\mathcal{M} \models \phi_1$或 $\mathcal{M} \models \phi_2$成立。根据归纳假设，$\vdash \phi_1$或$\vdash \phi_2$是可证的。因此，$\vdash \phi_1 \vee \phi_2$是可证的。
- 如果$\phi = AX_x(\phi_1)(s)$，那么由于$\mathcal{M} \models AX_x(\phi_1)(s)$，因此对于任意$s' \in \mathsf{Next}(s)$，都有$\mathcal{M} \models (s'/x)\phi_1$成立。根据归纳假设，对于任意$s' \in \mathsf{Next}(s)$，$\vdash (s'/x)\phi_1$都是可证的。因此，$\vdash AX_x(\phi_1)(s)$是可证的。
- 如果$\phi = EX_x(\phi_1)(s)$，那么由于$\mathcal{M} \models EX_x(\phi_1)(s)$，因此存在$s' \in \mathsf{Next}(s)$使得$\mathcal{M} \models (s'/x)\phi_1$成立。根据归纳假设，$\vdash (s'/x)\phi_1$是可证的，因此$\vdash EX_x(\phi_1)(s)$是可证的。
- 如果$\phi = AF_x(\phi_1)(s)$，那么由于$\mathcal{M} \models AF_x(\phi_1)(s)$，因此存在一棵有穷的路径树$T$，并且$T$以$s$为根结点；对于$T$的每个非叶子节点$s'$，$s'$的后继节点分别由$\mathsf{Next}(s)$中的元素所标记；对于$T$的每个叶子节点$s'$，都有$\mathcal{M} \models (s'/x)\phi_1$成立。根据归纳假设，$\vdash (s'/x)\phi_1$是可证的。然后，对于$T$的每个以子树$T'$（设$T'$的根结点为$s'$），我们归纳构造$\vdash AF_x(\phi_1)(s')$的一个证明$|T'|$。构造过程如下：
  - 如果T'只包含一个节点s'，那么$|T'|$的最后一条规则为$\mathbf{AF}$-R_1，同时$|T'|$中包含$\vdash (s'/x)\phi_1$的证明；
  - 如果$T' = s'(T_1, ..., T_n)$，那么$|T'|$的最后一条规则为$\mathbf{AF}$-R_2，同时$|T'_1|, ..., |T'_n|$分别为$\vdash AF_x(\phi_1)(s_1), ..., \vdash AF_x(\phi_1)(s_n)$的证明，其中$\{s_1, ..., s_n\} = \mathsf{Next}(s)$。

  因此，$|T|$是$\vdash AF_x(\phi_1)(s)$的一个证明。
- 如果$\phi = EG_x(\phi_1)(s)$，那么由于$\mathcal{M} \models EG_x(\phi_1)(s)$，因此存在一个状态序列$s_0, ..., s_n$使得$s_0 = s$，而且对于任意$0 \leq i \leq n$都有$\mathcal{M} \models (s_i/x)\phi_1$成立。根据归纳假设，$\vdash (s_i/x)\phi_1$是可证的。根据命题1.3，存在一个有穷的状态序列$T = s_0, ..., s_n$使得对任意$0 \leq i \leq n-1$，$s_i \longrightarrow s_{i+1}$，同时$\vdash (s_i/x)\phi_1$是可证的，而且存在$p < n$使得$s_n = s_p$。对于$T$的每个后缀$s_i, ..., s_n$，我们归纳构

造$|s_i, ..., s_n|$为$EG_x(\phi_1)(s_0), \ldots, EG_x(\phi_1)(s_{i-1}) \vdash EG_x(\phi_1)(s_i)$的证明。构造方式如下：

- $|s_n|$的最后一条规则为**EG**-merge；
- 如果$i \leq n-1$，根据归纳假设，由于$\vdash (s_i/x)\phi_1$是可证的，而且$|s_{i+1}, ..., s_n|$是$EG_x(\phi_1)(s_0), ... EG_x(\phi_1)(s_{i+1})$的一个证明。因此，$|s_i, ..., s_n|$是$EG_x(\phi_1)(s_0), \ldots, EG_x(\phi_1)(s_{i-1}) \vdash EG_x(\phi_1)(s_i)$的一个证明，而且最后一条规则为**EG**-R。

因此，$|s_0, ..., s_n|$是$\vdash EG_x(\phi_1)(s)$的一个证明。

- 如果$\phi = AR_{x,y}(\phi_1, \phi_2)(s)$，那么由于$\mathcal{M} \models AR_{x,y}(\phi_1, \phi_2)(s)$，因此存在一棵以$s$为根节点的可能无穷的路径树，对于该路径树的每个节点s'，都有$\mathcal{M} \models (s'/x)\phi_2$；对于该路径树的每个叶子节点$s'$，都有$\mathcal{M} \models (s'/x)\phi_1$。根据归纳假设，对于该路径树的每个节点$s'$，$\vdash (s'/y)\phi_2$是可证的，而且对于该路径树的每个叶子节点$s'$，$\vdash (s'/y)\phi_1$是可证的。由命题1.4可知，存在一棵有穷的路径树$T$，对于该路径树的每个节点$s'$，$\vdash (s'/y)\phi_2$是可证的；对于该路径树的每个叶子节点$s'$，$\vdash (s'/y)\phi_1$是可证的，或者$s'$为从$T$的根节点到该叶子节点分支上的节点。然后，对于$T$的每个子树$T'$，我们归纳构造$AR_{x,y}(\phi_1, \phi_2)(s_1), ..., AR_{x,y}(\phi_1, \phi_2)(s_m) \vdash AR_{x,y}(\phi_1, \phi_2)(s')$的一个证明$|T'|$，其中$s'$为$T'$的根节点，而且$s_1, ..., s_m$为从$T$的根节点到$T'$的根节点的分支。构造方式如下：
  - 如果T'只包含一个单独的节点s'，同时$\vdash (s'/x)\phi_1$是可证的，那么根据归纳假设，$\vdash (s'/x)\phi_1$和$\vdash (s'/y)\phi_2$皆可证，而且$|T'|$的最后一条规则为**AR**-R_1；
  - 如果T'只包含一个单独的节点，同时s'包含在$s_1, ..., s_m$中，那么$|T'|$的最后一条规则为**AR**-merge；
  - 如果$T' = s'(T_1, ..., T_n)$，那么根据归纳假设，$|T_1|, ..., |T_n|$分别为

$$AR_{x,y}(\phi_1, \phi_2)(s_1), ..., AR_{x,y}(\phi_1, \phi_2)(s_m), AR_{x,y}(\phi_1, \phi_2)(s') \vdash AR_{x,y}(\phi_1, \phi_2)(s'_1)$$

$$\ldots$$

$$AR_{x,y}(\phi_1, \phi_2)(s_1), ..., AR_{x,y}(\phi_1, \phi_2)(s_m), AR_{x,y}(\phi_1, \phi_2)(s') \vdash AR_{x,y}(\phi_1, \phi_2)(s'_n)$$

的证明，同时$|T'|$的最后一条规则为**AR**-R_2，其中$s'_1, ..., s'_n = \mathsf{Next}(s')$。

因此，$|T|$是$\vdash AR_{x,y}(\phi_1,\phi_2)(s)$的一个证明。

- 如果$\phi = EU_{x,y}(\phi_1,\phi_2)(s)$，那么由于$\mathcal{M} \models EU_{x,y}(\phi_1,\phi_2)(s)$，因此存在一个有穷的状态序列$T = s_0,...,s_n$使得$\mathcal{M} \models (s_n/y)\phi_2$成立，而且对于任意$0 \leq i \leq n-1$，$\mathcal{M} \models (s_i/x)\phi_1$成立。根据归纳假设，$\vdash (s_n/y)\phi_2$是可证的，而且对于任意$0 \leq i \leq n-1$，$\vdash (s_i/x)\phi_1$是可证的。然后，对于$T$的每个后缀$s_i,...,s_n$，我们归纳构造$|s_i,...,s_n|$为$\vdash EU_{x,y}(\phi_1,\phi_2)(s_i)$的证明。构造方式如下：

  – $|s_n|$的最后一条规则为**EU**-R_1；

  – 如果$i \leq n-1$，那么根据归纳假设，由于$|s_{i+1},...,s_n|$是$\vdash EU_{x,y}(\phi_1,\phi_2)(s_{i+1})$的证明，而且$\vdash (s_i/x)\phi_1$是可证的，因此，$|s_i,...,s_n|$是$\vdash EU_{x,y}(\phi_1,\phi_2)(s_i)$的证明。

因此，$|s_0,...,s_n|$是$\vdash EU_{x,y}(\phi_1,\phi_2)(s)$的一个证明。

□

1.3 证明搜索策略

在本节，我们介绍一种在SCTL中进行证明搜索的策略，然后，我们证明该证明搜索策略的终止性和正确性，然后，我们提出两种针对该证明搜索策略的优化方法，最后，我们将该证明搜索策略以伪代码的形式给出。

该证明搜素策略如下：首先，对于要证明的相继式，以及SCTL规则将该公式的所有的前提给定一个序；然后，依次对这些前提进行证明搜索。我们将以上证明搜索方法定义成一系列对于连续传递树（定义1.3）的重写规则。下面我们介绍连续传递树的概念。

1.3.0.1 连续传递树

在连续传递树中，连续一个基本的概念。在计算机程序设计语言理论[33, 34]中，连续是计算机程序将要执行的部分的显示表示。

定义 1.3 (连续传递树). 一个连续传递树（*Continuation Passing Tree*，简写为*CPT*）指的是一棵同时满足以下条件的二叉树：

- 每个叶子节点被t或f标记，其中t和f是不同的两个符号；
- 每个非叶子节点都被一个*SCTL*相继式标记。

对于*CPT*的每个非叶子节点来说，它的左子树称之为该节点的t-连续；它的右子树称之为该节点的f-连续。对于一个*CPT* c来说，若c的根节点为$\Gamma \vdash \phi$，以及c的t-连续和f-连续分别为c_1和c_2，那么我们将c记作$cpt(\Gamma \vdash \phi, c_1, c_2)$，或者可以表示为如下形式：

$$\begin{array}{c} \Gamma \vdash \phi \\ \diagup \quad \diagdown \\ c_1 \quad c_2 \end{array}$$

因此，SCTL的该证明搜索策略可总结为：对于给定的SCTL相继式$\vdash \phi$，我们构造一个连续传递树$c = \mathsf{cpt}(\vdash \phi, \mathsf{t}, \mathsf{f})$，然后根据图1.4所示的重写规则将$c$重写到t或f。如果$c$最终重写到t，那么$\vdash \phi$是可证的；如果$c$最终重写到f，那么$\vdash \phi$是不可证的。

在CPT的重写规则中，对一个CPTc的一步重写只需判断c的根节点，而与c的子表达式无关。例如，根据重写规则，$\mathsf{CPTcpt}(\vdash \phi_1 \wedge \phi_2, \mathsf{t}, \mathsf{f})$重写到$\mathsf{cpt}(\vdash \phi_1, \mathsf{cpt}(\vdash \phi_2, \mathsf{t}, \mathsf{f}), \mathsf{f})$，此步重写意味着：如果搜索$\vdash \phi_1$的证明成功，则继续搜索$\vdash \phi_2$的证明；如果搜索$\vdash \phi_1$的证明失败，则直接判定$\vdash \phi_1 \wedge \phi_2$不可证。接下来，根据$\phi_1$的结构，继续对$\mathsf{cpt}(\vdash \phi_1, \mathsf{cpt}(\vdash \phi_2, \mathsf{t}, \mathsf{f}), \mathsf{f})$进行重写。下面，我们用一个完整的例子来说明该证明搜索策略。

例子 1.4. 根据重写规则，我们将例子*1.2*中公式的证明搜索过程表示为图*1.3*中的重写步骤。

下边我们分别解释图*1.3*中的每一步重写。

1. 在这一步中，$\overset{1}{\rightsquigarrow}$左侧的*CPT*中的根节点为$\vdash AF_x(P(x))(a)$。我们暂时还无法判定$\vdash AF_x(P(x))(a)$是否可证，因此根据证明规则，我们需要首先判定$\vdash P(a)$是否可证，如果$\vdash P(a)$可证，则$\vdash AF_x(P(x))(a)$可证；否则，我们需要依次判定$AF_x(P(x))(a) \vdash AF_x(P(x))(b)$和$AF_x(P(x))(a) \vdash AF_x(P(x))(c)$是否可证，如果$AF_x(P(x))(a) \vdash AF_x(P(x))(b)$和$AF_x(P(x))(a) \vdash AF_x(P(x))(c)$均可证，则$\vdash AF_x(P(x))(a)$可证，否则$\vdash AF_x(P(x))(a)$可证不可证。我们将以上介绍的判定$\vdash AF_x(P(x))(a)$是否可证过程编码到$\overset{1}{\rightsquigarrow}$右侧的*CPT*中。

2. 在这一步中，由于$\vdash P(a)$时不可证的，因此将$\overset{2}{\rightsquigarrow}$左侧的*CPT*重写到它的右子树（f-连续），即$\overset{2}{\rightsquigarrow}$右侧的*CPT*。

3. 这一步与第*1*步类似，我们暂时无法判定$AF_x(P(x))(a) \vdash AF_x(P(x))(b)$是否可证，因此根据证明规则，我们需要首先判定$\vdash P(b)$是否可证，如果$\vdash P(b)$可

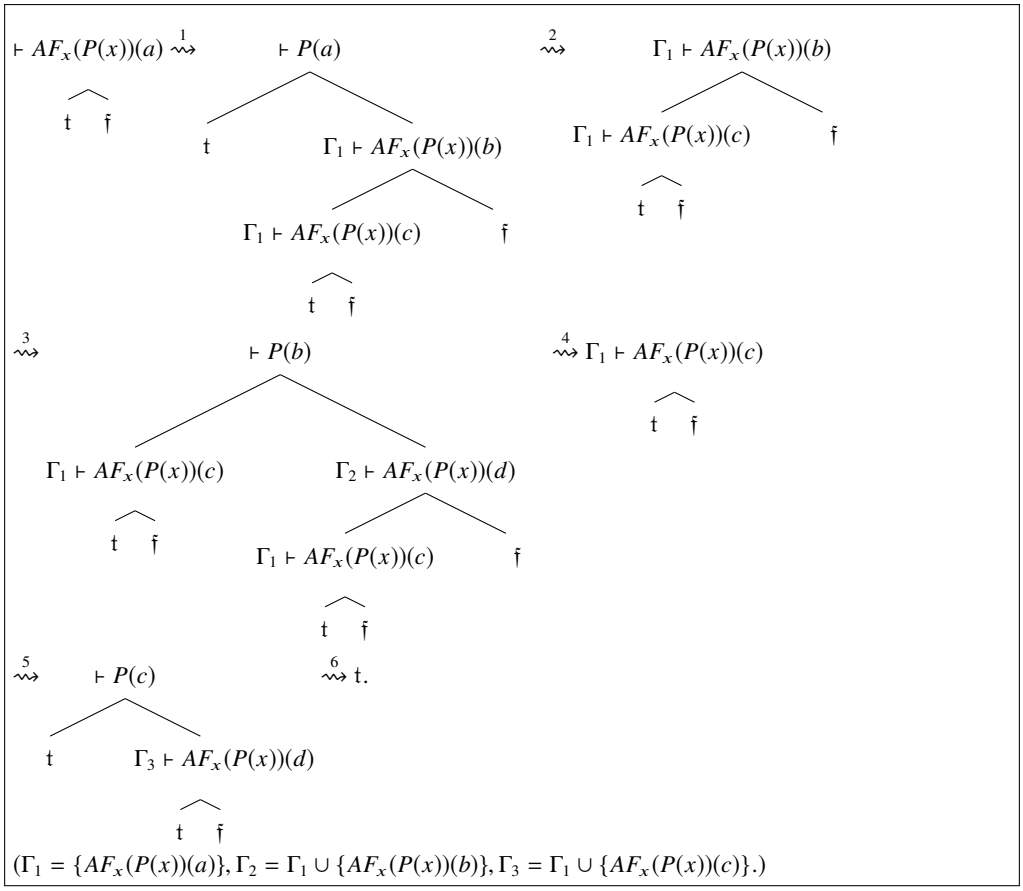


图 1.3 一个重写CPT的例子

证，那么$AF_x(P(x))(a) \vdash AF_x(P(x))(b)$可证，然后接着第2步证明$\overset{3}{\rightsquigarrow}$左侧的*CPT*的左子树；否则，我们需要判定$AF_x(P(x))(a), AF_x(P(x))(b) \vdash AF_x(P(x))(d)$是否可证，若$AF_x(P(x))(a), AF_x(P(x))(b) \vdash AF_x(P(x))(d)$可证，则接着第2步证明$\overset{3}{\rightsquigarrow}$左侧的*CPT*的左子树，若$AF_x(P(x))(a), AF_x(P(x))(b) \vdash AF_x(P(x))(d)$不可证，则$AF_x(P(x))(a) \vdash AF_x(P(x))(b)$不可证。我们将以上介绍的判定 $AF_x(P(x))(a) \vdash AF_x(P(x))(b)$ 是否可证过程编码到$\overset{3}{\rightsquigarrow}$右侧的*CPT*中。

4. 这一步与第2步类似，我们可以判定$\vdash P(b)$可证，因此$\overset{4}{\rightsquigarrow}$左侧的*CPT*重写到它的左子树，即$\overset{4}{\rightsquigarrow}$右侧的*CPT*。

5. 这一步与第1、3步类似，我们暂时无法判定$AF_x(P(x))(a) \vdash AF_x(P(x))(c)$是否可证，因此我们需要首先判定$\vdash P(c)$是否可证，若$\vdash P(c)$可证，则$AF_x(P(x))(a) \vdash AF_x(P(x))(c)$可证；否则，判定$AF_x(P(x))(a), AF_x(P(x))(c) \vdash AF_x(P(x))(d)$是否可证，若$AF_x(P(x))(a), AF_x(P(x))(c) \vdash AF_x(P(x))(d)$可证，则$AF_x(P(x))(a) \vdash AF_x(P(x))(c)$可证。我们将以上介绍的判定 $AF_x(P(x))(a) \vdash AF_x(P(x))(c)$ 是否可证过程编码到$\overset{5}{\rightsquigarrow}$右侧的*CPT*中。

$\mathsf{cpt}(\vdash \top, c_1, c_2) \rightsquigarrow c_1 \quad \mathsf{cpt}(\vdash \bot, c_1, c_2) \rightsquigarrow c_2$
$\mathsf{cpt}(\vdash P(s_1, ..., s_n), c_1, c_2) \rightsquigarrow c_1 \quad [\langle s_1, ..., s_n\rangle \in P]$
$\mathsf{cpt}(\vdash P(s_1, ..., s_n), c_1, c_2) \rightsquigarrow c_2 \quad [\langle s_1, ..., s_n\rangle \notin P]$
$\mathsf{cpt}(\vdash \neg P(s_1, ..., s_n), c_1, c_2) \rightsquigarrow c_2 \quad [\langle s_1, ..., s_n\rangle \in P]$
$\mathsf{cpt}(\vdash \neg P(s_1, ..., s_n), c_1, c_2) \rightsquigarrow c_1 \quad [\langle s_1, ..., s_n\rangle \notin P]$
$\mathsf{cpt}(\vdash \phi_1 \wedge \phi_2, c_1, c_2) \rightsquigarrow \mathsf{cpt}(\vdash \phi_1, \mathsf{cpt}(\vdash \phi_2, c_1, c_2), c_2)$
$\mathsf{cpt}(\vdash \phi_1 \vee \phi_2, c_1, c_2) \rightsquigarrow \mathsf{cpt}(\vdash \phi_1, c_1, \mathsf{cpt}(\vdash \phi_2, c_1, c_2))$
$\mathsf{cpt}(\vdash AX_x(\phi)(s), c_1, c_2) \rightsquigarrow \mathsf{cpt}(\vdash (s_1/x)\phi, \mathsf{cpt}(\vdash (s_2/x)\phi, \mathsf{cpt}(...\mathsf{cpt}(\vdash (s_n/x)\phi, c_1, c_2), ..., c_2), c_2), c_2)$ $[\{s_1, ..., s_n\} = \mathsf{Next}(s)]$
$\mathsf{cpt}(\vdash EX_x(\phi)(s), c_1, c_2) \rightsquigarrow \mathsf{cpt}(\vdash (s_1/x)\phi, c_1, \mathsf{cpt}(\vdash (s_2/x)\phi, c_1, \mathsf{cpt}(...\mathsf{cpt}(\vdash (s_n/x)\phi, c_1, c_2)...)))$ $[\{s_1, ..., s_n\} = \mathsf{Next}(s)]$
$\mathsf{cpt}(\Gamma \vdash AF_x(\phi)(s), c_1, c_2) \rightsquigarrow c_2 \quad [AF_x(\phi)(s) \in \Gamma]$
$\mathsf{cpt}(\Gamma \vdash AF_x(\phi)(s), c_1, c_2) \rightsquigarrow$ $\mathsf{cpt}(\vdash (s/x)\phi, c_1, \mathsf{cpt}(\Gamma' \vdash AF_x(\phi)(s_1), \mathsf{cpt}(...\mathsf{cpt}(\Gamma' \vdash AF_x(\phi)(s_n), c_1, c_2)..., c_2), c_2))$ $[\{s_1, ..., s_n\} = \mathsf{Next}(s), AF_x(\phi)(s) \notin \Gamma, \text{and } \Gamma' = \Gamma, AF_x(\phi)(s)]$
$\mathsf{cpt}(\Gamma \vdash EG_x(\phi)(s), c_1, c_2) \rightsquigarrow c_1 \quad [EG_x(\phi)(s) \in \Gamma]$
$\mathsf{cpt}(\Gamma \vdash EG_x(\phi)(s), c_1, c_2) \rightsquigarrow$ $\mathsf{cpt}(\vdash (s/x)\phi, \mathsf{cpt}(\Gamma' \vdash EG_x(\phi)(s_1), c_1, \mathsf{cpt}(...\mathsf{cpt}(\Gamma' \vdash EG_x(\phi)(s_n), c_1, c_2)...)), c_2)$ $[\{s_1, ..., s_n\} = \mathsf{Next}(s), EG_x(\phi)(s) \notin \Gamma, \text{and } \Gamma' = \Gamma, EG_x(\phi)(s)]$
$\mathsf{cpt}(\Gamma \vdash AR_{x,y}(\phi_1, \phi_2)(s), c_1, c_2) \rightsquigarrow c_1 \quad \left[(AR_{x,y}(\phi_1, \phi_2)(s) \in \Gamma\right]$
$\mathsf{cpt}(\Gamma \vdash AR_{x,y}(\phi_1, \phi_2)(s), c_1, c_2) \rightsquigarrow$ $\mathsf{cpt}(\vdash (s/y)\phi_2, \mathsf{cpt}(\vdash (s/x)\phi_1, c_1, \mathsf{cpt}(\Gamma' \vdash AR_{x,y}(\phi_1, \phi_2)(s_1), \mathsf{cpt}(...\mathsf{cpt}(\Gamma' \vdash AR_{x,y}(\phi_1, \phi_2)(s_n),$ $c_1, c_2)..., c_2), c_2)), c_2) \quad \left[\{s_1, ..., s_n\} = \mathsf{Next}(s), AR_{x,y}(\phi_1, \phi_2)(s) \notin \Gamma, \text{ and } \Gamma' = \Gamma, AR_{x,y}(\phi_1, \phi_2)(s)\right]$
$\mathsf{cpt}(\Gamma \vdash EU_{x,y}(\phi_1, \phi_2)(s), c_1, c_2) \rightsquigarrow c_2 \quad \left[EU_{x,y}(\phi_1, \phi_2)(s) \in \Gamma\right]$
$\mathsf{cpt}(\Gamma \vdash EU_{x,y}(\phi_1, \phi_2)(s), c_1, c_2) \rightsquigarrow$ $\mathsf{cpt}(\vdash (s/y)\phi_2, c_1, \mathsf{cpt}(\vdash (s/x)\phi_1, \mathsf{cpt}(\Gamma' \vdash EU_{x,y}(\phi_1, \phi_2)(s_1), c_1, \mathsf{cpt}(...\mathsf{cpt}(\Gamma' \vdash EU_{x,y}(\phi_1, \phi_2)(s_n),$ $c_1, c_2)...)), c_2)) \quad \left[\{s_1, ..., s_n\} = \mathsf{Next}(s), EU_{x,y}(\phi_1, \phi_2)(s) \notin \Gamma, \text{ and } \Gamma' = \Gamma, EU_{x,y}(\phi_1, \phi_2)(s)\right]$

图 1.4 CPT的重写规则

6. 在这一步中，由于$\vdash P(c)$可证，因此$\overset{6}{\rightsquigarrow}$左侧的*CPT*重写到它的左子树，即t。至此，判定$\vdash AF_x(P(x))(a)$是否可证的过程结束，$\vdash AF_x(P(x))(a)$可证。

1.3.1 证明搜索策略的可终止性

在证明利用图1.3表示的重写规则，使得SCTLProV的证明搜索是可终止的之前，我们需要引入以下定义和命题。

定义 1.4 (字典路径序（lexicographic path ordering）[35, 36])**.** 设$\geq$是函数符号集合F的一个拟序（*quasi-ordering*），其中F的每个符号的元数（*arity*）是固定不变的。集合$T(F)$（由F生成的项的集合）上的字典路径序$\geq_{\mathsf{lpo}}$的归纳定义如下：

$s = f(s_1, ..., s_m) \succeq_{\text{lpo}} g(t_1, ..., t_n) = t$ 当且仅当以下至少一条断言成立：

- 存在$i \in \{1, ..., m\}$，使得$s_i \succeq_{\text{lpo}} t$ 成立。
- 对于任意$j \in \{1, ..., n\}$，$f \succ g$和 $s \succ_{\text{lpo}} t_j$同时成立。
- 对于任意$j \in \{2, ..., n\}, f = g$，$(s_1, ..., s_m) \succeq'_{\text{lpo}} (t_1, ..., t_n)$和 $s \succ_{\text{lpo}} t_j$都成立，其中$\succeq'_{\text{lpo}}$是由$\succeq_{\text{lpo}}$产生的字典序。

命题 1.5 (字典路径序的良基性)**.** 如果$\succeq$是函数符号集合F的一个拟序（*quasi-ordering*），其中F的每个符号的元数（*arity*）是固定不变的，那么根据集合$T(F)$（由F生成的项的集合）上的字典路径序$\succeq_{\text{lpo}}$是良基的当且仅当$\succeq$是良基的。

证明. 证明由Dershowitz提出，参考[35]。 □

定义 1.5 (相继式的权重)**.** 假设一个*Kripke*模型的状态集的基数为n；$\Gamma \vdash \phi$是一个*SCTL*$(\mathcal{M})$相继式；$|\phi|$是公式ϕ的大小；$|\Gamma|$是Γ的基数。相继式$\Gamma \vdash \phi$的权重为

$$w(\Gamma \vdash \phi) = \langle |\phi|, (n - |\Gamma|) \rangle$$

命题 1.6 (可终止性)**.** 假设$\mathcal{M}$是一个*Kripke*模型，ϕ是一个$CTL_P(\mathcal{M})$闭公式，那么*cpt*$(\vdash \phi, \mathrm{t}, \mathrm{f})$能在有限步之内重写到t或f。

证明. 令$F = \{\mathrm{t}, \mathrm{f}, \mathrm{cpt}\} \cup \mathrm{Seq}$，其中 Seq是在cpt$(\vdash \phi, \mathrm{t}, \mathrm{f})$的重写步骤中所出现的相继式的集合；cpt的元数是3，F中其他符号的元数是0。F上的拟序$\succeq$（$\forall f, g \in F$, $f \succ g$是指“$f \succeq g$同时$f \neq g$”）定义如下：

- $\mathrm{cpt} \succ \mathrm{t}$；
- $\mathrm{cpt} \succ \mathrm{f}$；
- 对于每个相继式$\Gamma \vdash \phi$都有$\Gamma \vdash \phi \succ \mathrm{cpt}$；
- $\Gamma \vdash \phi \succ \Gamma' \vdash \phi'$ 当且仅当 $w(\Gamma \vdash \phi) > w(\Gamma' \vdash \phi')$，其中>是自然数对上的字典序。

令$\succeq_{\text{lpo}}$为由$\succeq$生成的关于CPT的字典序。显然，$\succeq$是良基的，因此根据命题1.5可知，$\succeq_{\text{lpo}}$也是良基的。

若要证明重写系统是可终止的，只需证明对于每一步重写$c \rightsquigarrow c'$，都有$c \succ_{\text{lpo}} c'$。下面我们针对重写规则逐条进行分析：

假设c是cpt$(\Gamma \vdash \phi, c_1, c_2)$形式的。

- 如果$\phi = \top, \bot, P(s_1, ..., s_m)$或$\neg P(s_1, ..., s_m)$，那么由于$c_1$和$c_2$是$\mathsf{cpt}(\Gamma \vdash \phi, c_1, c_2)$的子项，因此，$\mathsf{cpt}(\Gamma \vdash \phi, c_1, c_2) \succ_{\mathsf{lpo}} c_1$以及$\mathsf{cpt}(\Gamma \vdash \phi, c_1, c_2) \succ_{\mathsf{lpo}} c_2$。
- 如果$\phi = \phi_1 \wedge \phi_2$, 那么由$\succ_{\mathsf{lpo}}$的定义可知，由于$\vdash \phi_1 \wedge \phi_2 \succ \vdash \phi_1$， $\mathsf{cpt}(\vdash \phi_1 \wedge \phi_2, c_1, c_2) \succ_{\mathsf{lpo}} \mathsf{cpt}(\vdash \phi_2, c_1, c_2)$以及 $\mathsf{cpt}(\vdash \phi_1 \wedge \phi_2, c_1, c_2) \succ_{\mathsf{lpo}} c_2$，因此$\mathsf{cpt}(\vdash \phi_1 \wedge \phi_2, c_1, c_2) \succ_{\mathsf{lpo}} \mathsf{cpt}(\vdash \phi_1, \mathsf{cpt}(\vdash \phi_2, c_1, c_2), c_2)$；
- 如果$\phi = \phi_1 \vee \phi_2$，那么由$\succ_{\mathsf{lpo}}$的定义可知，由于$\vdash \phi_1 \vee \phi_2 \succ \vdash \phi_1$， $\mathsf{cpt}(\vdash \phi_1 \wedge \phi_2, c_1, c_2) \succ_{\mathsf{lpo}} c_1$以及$\mathsf{cpt}(\vdash \phi_1 \vee \phi_2, c_1, c_2) \succ_{\mathsf{lpo}} \mathsf{cpt}(\vdash \phi_2, c_1, c_2)$，因此 $\mathsf{cpt}(\vdash \phi_1 \vee \phi_2, c_1, c_2) \succ_{\mathsf{lpo}} \mathsf{cpt}(\vdash \phi_1, c_1, \mathsf{cpt}(\vdash \phi_2, c_1, c_2))$；
- 如果$\phi = AX_x(\phi_1)(s)$，那么根据$\succ_{\mathsf{lpo}}$的定义可知，由于$\Gamma \vdash AX_x(\phi_1)(s) \succ \vdash (s_i/x)\phi_1$以及 $\mathsf{cpt}(\Gamma \vdash AX_x(\phi_1)(s), c_1, c_2) \succ_{\mathsf{lpo}} \mathsf{cpt}(\vdash (s_i/x)\phi_1, \mathsf{cpt}(...\mathsf{cpt}(\vdash (s_n/x)\phi_1, c_1, c_2)..., c_2), c_2)$，因此$\mathsf{cpt}(\Gamma \vdash AX_x(\phi_1)(s), c_1, c_2) \succ_{\mathsf{lpo}} \mathsf{cpt}(\vdash (s_1/x)\phi_1,$ $\mathsf{cpt}(...\mathsf{cpt}(\vdash (s_n/x)\phi_1, c_1, c_2), ..., c_2), c_2)$，其中$\mathsf{Next}(s) = \{s_1, ..., s_n\}$，而且$i \in \{1, ..., n\}$；
- 对于EX情况的分析与AX类似；
- 如果$\phi = EG_x(\phi_1)(s)$，那么
  - 当$EG_x(\phi_1)(s) \in \Gamma$时，此时与第一种情况类似：$c \succ_{\mathsf{lpo}} c'$；
  - 当$EG_x(\phi_1)(s) \notin \Gamma$时，根据$\succ_{\mathsf{lpo}}$的定义可知，由于$\Gamma \vdash EG_x(\phi_1)(s) \succ \vdash (s/x)\phi_1$以及$\forall i \in \{1, ..., n\}, \Gamma \vdash EG_x(\phi_1)(s) \succ \Gamma' \vdash EG_x(\phi_1)(s_i)$，其中$\mathsf{Next}(s) = \{s_1, ..., s_n\}$以及 $\Gamma' = \Gamma \cup \{EG_x(\phi_1)(s)\}$；
- 对于AF情况的分析与EG类似；
- 如果$\phi = AR_{x,y}(\phi_1, \phi_2)(s)$，那么
  - 当$AR_{x,y}(\phi_1, \phi_2)(s) \in \Gamma$时，此时与第一种情况类似：$c \succ_{\mathsf{lpo}} c'$；
  - 当$AR_{x,y}(\phi_1, \phi_2)(s) \notin \Gamma$时，由$\succ_{\mathsf{lpo}}$的定义可知，由于$\Gamma \vdash AR_{x,y}(\phi_1, \phi_2)(s) \succ \vdash (s/y)\phi_2$， $\Gamma \vdash AR_{x,y}(\phi_1, \phi_2)(s) \succ \vdash (s/x)\phi_1$以及$\forall i \in \{1, ..., n\}, \Gamma \vdash AR_{x,y}(\phi_1, \phi_2)(s) \succ \Gamma' \vdash AR_{x,y}(\phi_1, \phi_2)(s_i)$，因此$c \succ_{\mathsf{lpo}} c'$，其中 $\mathsf{Next}(s) = \{s_1, ..., s_n\}$以及 $\Gamma' = \Gamma \cup \{AR_{x,y}(\phi_1, \phi_2)(s)\}$；
- 对于EU情况的分析与AR类似。

□

1.3.2 证明搜索策略的正确性

证明搜索策略的正确性可由以下命题来表示。

命题 1.7 (证明搜索策略的正确性)**.** 对于给定闭公式ϕ， $cpt(\vdash \phi, \mathrm{t}, \mathrm{f}) \rightsquigarrow^* \mathrm{t}$ 当且仅当$\vdash \phi$是可证的。

证明. 我们证明一个更一般的命题，即：对于一个给定的公式ϕ, 对任意不同的两个CPT c_1和c_2都有$\mathsf{cpt}(\Gamma \vdash \phi, c_1, c_2) \rightsquigarrow^* c_1$当且仅当$\Gamma \vdash \phi$是可证的。

需要注意的是，对任意不同的两个CPT c_1和c_2，$\mathsf{cpt}(\Gamma \vdash \phi, c_1, c_2)$总会在有限步之内重写到$c_1$或$c_2$。这是由于根据命题1.6， $\mathsf{cpt}(\Gamma \vdash \phi, \mathrm{t}, \mathrm{f})$总会在有限步之内重写到 t或f，而且在重写$\mathsf{cpt}(\Gamma \vdash \phi, c_1, c_2)$的过程中;$c_1$和$c_2$的结构都不会影响重写的步骤直到需要重写$c_1$或者$c_2$本身。因此，我们可以在$\mathsf{cpt}(\Gamma \vdash \phi, \mathrm{t}, \mathrm{f})$的重写步骤中将t替换成$c_1$，将f替换成$c_2$并由此得到由$\mathsf{cpt}(\Gamma \vdash \phi, c_1, c_2)$重写到$c_1$或$c_2$的步骤。本证明中需要用到这个性质。

现在，我们通过对$\Gamma \vdash \phi$的权重（见定义1.5）进行归纳分析。在本证明中，我们将CPT c_1无法在有限步之内重写到c_2记作$c_1 \not\rightsquigarrow^* c_2$。

- 如果$\phi = \top$或$\bot$，命题显然成立。
- 如果$\phi = P(s_1, ..., s_n)$，其中$P(s_1, ..., s_n)$是原子公式，那么对任意两个CPT c_1和c_2都有 $\mathsf{cpt}(\vdash P(s_1, ..., s_n), c_1, c_2) \rightsquigarrow c_1$当且仅当$\langle s_1, ..., s_n\rangle \in P$当且仅当$\vdash P(s_1, ..., s_n)$是可证的。
- 如果$\phi = \neg P(s_1, ..., s_n)$，其中$P(s_1, ..., s_n)$是原子命题，那么对任意两个CPT c_1和c_2都有$\mathsf{cpt}(\vdash \neg P(s_1, ..., s_n), c_1, c_2) \rightsquigarrow c_1$当且仅当$\langle s_1, ..., s_n\rangle \notin P$当且仅当$\vdash \neg P(s_1, ..., s_n)$是可证的。
- 如果$\phi = \phi_1 \wedge \phi_2$，那么
  - ($\Rightarrow$) 如果对任意两个不同的CPT c_1和c_2都有$\mathsf{cpt}(\vdash \phi_1 \wedge \phi_2, c_1, c_2) \rightsquigarrow^* c_1$，那么$\vdash \phi_1$和$\vdash \phi_2$都是可证的。否则，如果$\vdash \phi$不可证，那么根据归纳假设，存在两个不相同的CPT c_1和c_2使得 $\mathsf{cpt}(\vdash \phi_1 \wedge \phi_2, c_1, c_2) \rightsquigarrow \mathsf{cpt}(\vdash \phi_1, \mathsf{cpt}(\vdash \phi_2, c_1, c_2), c_2) \rightsquigarrow^* c_2$而且$c_2$不能在有限步内重写到$c_1$；如果$\vdash \phi_1$可证而$\vdash \phi_2$不可证，那么根据归纳假设，存在两个不同的CPT c_1和c_2使得$\mathsf{cpt}(\vdash \phi_1 \wedge \phi_2, c_1, c_2) \rightsquigarrow \mathsf{cpt}(\vdash \phi_1, \mathsf{cpt}(\vdash \phi_2, c_1, c_2), c_2) \rightsquigarrow^* \mathsf{cpt}(\vdash \phi_2, c_1, c_2) \not\rightsquigarrow^* c_1$。因此，由证明系统的规则可知，$\vdash \phi_1 \wedge \phi_2$是可证的。

– ($\Leftarrow$) 如果$\vdash \phi_1 \wedge \phi_2$是可证的，那么由证明系统规则可知，$\vdash \phi_1$和$\vdash \phi_2$都是可证的，那么根据归纳假设，对任意两个不同的CPT c_1和c_2都有$\mathsf{cpt}(\vdash \phi_1 \wedge \phi_2, c_1, c_2) \rightsquigarrow \mathsf{cpt}(\vdash \phi_1, \mathsf{cpt}(\vdash \phi_2, c_1, c_2), c_2) \rightsquigarrow^* \mathsf{cpt}(\vdash \phi_2, c_1, c_2) \rightsquigarrow^* c_1$。

- 如果$\phi = \phi_1 \vee \phi_2$，那么

  – ($\Rightarrow$) 对任意两个不同的CPT c_1和c_2都有$\mathsf{cpt}(\vdash \phi_1 \vee \phi_2, c_1, c_2) \rightsquigarrow^* c_1$，那么$\vdash \phi_1$或$\vdash \phi_2$是可证的。否则，如果$\vdash \phi_1$和$\vdash \phi_2$均不可证，那么根据归纳假设，存在两个不同的CPT c_1和c_2使得 $\mathsf{cpt}(\vdash \phi_1 \vee \phi_2, c_1, c_2) \rightsquigarrow \mathsf{cpt}(\vdash \phi_1, c_1, \mathsf{cpt}(\vdash \phi_2, c_1, c_2)) \rightsquigarrow^* \mathsf{cpt}(\vdash \phi_2, c_1, c_2) \rightsquigarrow^* c_2$，而且$c_2$不能在有限步内重写到$c_1$。因此，由证明系统的规则可知，$\vdash \phi_1 \vee \phi_2$是可证的。

  – ($\Leftarrow$) 如果$\vdash \phi_1 \vee \phi_2$是可证的，那么由证明系统的规则可知，$\vdash \phi_1$或$\vdash \phi_2$是可证的，那么根据归纳假设，如果$\vdash \phi_1$是可证的，那么对任意两个不同的CPT c_1和c_2都有$\mathsf{cpt}(\vdash \phi_1 \vee \phi_2, c_1, c_2) \rightsquigarrow \mathsf{cpt}(\vdash \phi_1, c_1, \mathsf{cpt}(\vdash \phi_2, c_1, c_2)) \rightsquigarrow^* c_1$；如果$\vdash \phi_2$是可证的，那么对任意两个不同的CPT c_1和c_2都有$\mathsf{cpt}(\vdash \phi_1 \vee \phi_2, c_1, c_2) \rightsquigarrow \mathsf{cpt}(\vdash \phi_1, c_1, \mathsf{cpt}(\vdash \phi_2, c_1, c_2)) \rightsquigarrow^* \mathsf{cpt}(\vdash \phi_2, c_1, c_2) \rightsquigarrow^* c_1$或者$\mathsf{cpt}(\vdash \phi_1 \vee \phi_2, c_1, c_2) \rightsquigarrow \mathsf{cpt}(\vdash \phi_1, c_1, \mathsf{cpt}(\vdash \phi_2, c_1, c_2)) \rightsquigarrow^* c_1$。

- 如果$\phi = AX_x(\psi)(s)$，而且$\{s_1, ..., s_n\} = \mathsf{Next}(s)$，那么

  – ($\Rightarrow$) 对任意两个不同的CPT c_1和c_2都有 $\mathsf{cpt}(\vdash AX_x(\psi)(s), c_1, c_2) \rightsquigarrow^* c_1$，那么$\vdash (s_1/x)\psi, ..., \vdash (s_n/x)\psi$都是可证的。否则，如果存在$1 \leq i \leq n$使得对任意$1 \leq j < i$，$\vdash (s_1/x)\psi, ..., \vdash (s_j/x)\psi$都是可证的，而$\vdash (s_i/x)\psi$是不可证的，那么根据归纳假设，存在两个不同的 CPT c_1和c_2使得（我们用$\vdash \psi_{s_i}$表示$\vdash (s_i/x)\psi$）

  $\mathsf{cpt}(\vdash AX_x(\psi)(s), c_1, c_2) \rightsquigarrow$

  $\mathsf{cpt}(\vdash \psi_{s_1}, \mathsf{cpt}(...\mathsf{cpt}(\vdash \psi_{s_n}, c_1, c_2)...), c_2) \rightsquigarrow^*$

  $... \rightsquigarrow^*$

  $\mathsf{cpt}(\vdash \psi_{s_j}, \mathsf{cpt}(...\mathsf{cpt}(\vdash \psi_{s_n}, c_1, c_2)...), c_2) \rightsquigarrow^*$

  $\mathsf{cpt}(\vdash \psi_{s_i}, \mathsf{cpt}(...\mathsf{cpt}(\vdash \psi_{s_n}, c_1, c_2)...), c_2) \not\rightsquigarrow^*$

  c_1。因此，由证明系统的规则可知，$\vdash AX_x(\psi)(s)$是可证的。

  – ($\Leftarrow$) 如果$\vdash AX_x(\psi)(s)$是可证的，那么由证明系统的规则可知，$\vdash (s_1/x)\psi, ..., \vdash$

$(s_n/x)\psi$都是可证的，那么根据归纳假设，对任意两个不同的CPT c_1和c_2都有（我们用$\vdash \psi_{s_i}$表示$\vdash (s_i/x)\psi$）

$\mathsf{cpt}(\vdash AX_x(\psi)(s), c_1, c_2) \rightsquigarrow$

$\mathsf{cpt}(\vdash \psi_{s_1}, \mathsf{cpt}(\ldots\mathsf{cpt}(\vdash \psi_{s_n}, c_1, c_2)\ldots), c_2) \rightsquigarrow^*$

$\ldots \rightsquigarrow^*$

$\mathsf{cpt}(\vdash \psi_{s_n}, c_1, c_2) \rightsquigarrow^* c_1$。

- 如果$\phi = EX_x(\psi)(s)$，而且$\{s_1, \ldots, s_n\} = \mathsf{Next}(s)$，那么
  - ($\Rightarrow$) 对任意两个不同的CPT c_1和c_2都有$\mathsf{cpt}(\vdash EX_x(\psi)(s), c_1, c_2) \rightsquigarrow^* c_1$，那么存在$1 \leq i \leq n$使得$\vdash (s_i/x)\psi$是可证的。否则，如果对任意$1 \leq i \leq n$，$\vdash (s_i/x)\psi$都是不可证的，那么根据归纳假设，存在两个不同的CPT c_1和c_2使得（我们用$\vdash \psi_{s_i}$表示$\vdash (s_i/x)\psi$）

    $\mathsf{cpt}(\vdash EX_x(\psi)(s), c_1, c_2) \rightsquigarrow$

    $\mathsf{cpt}(\vdash \psi_{s_1}, c_1, \mathsf{cpt}(\ldots\mathsf{cpt}(\vdash \psi_{s_n}, c_1, c_2)\ldots)) \rightsquigarrow^*$

    $\ldots \rightsquigarrow^*$

    $\mathsf{cpt}(\vdash \psi_{s_n}, c_1, c_2) \not\rightsquigarrow^* c_1$。因此，由证明系统的规则可知，$\vdash EX_x(\psi)(s)$是可证的。
  - ($\Leftarrow$) 如果$\vdash EX_x(\psi)(s)$是可证的，那么由证明系统的规则可知，存在$1 \leq i \leq n$使得$\vdash (s_i/x)\psi$是可证的。根据归纳假设，对任意两个不同的CPT c_1和c_2都有（我们用$\vdash \psi_{s_i}$表示$\vdash (s_i/x)\psi$）

    $\mathsf{cpt}(\vdash EX_x(\psi)(s), c_1, c_2) \rightsquigarrow$

    $\mathsf{cpt}(\vdash \psi_{s_1}, c_1, \mathsf{cpt}(\ldots\mathsf{cpt}(\vdash \psi_{s_n}, c_1, c_2)\ldots)) \rightsquigarrow^*$

    $\ldots$

    $\mathsf{cpt}(\vdash \psi_{s_{i-1}}, c_1, \mathsf{cpt}(\ldots\mathsf{cpt}(\vdash \psi_{s_n}, c_1, c_2)\ldots)) \rightsquigarrow^*$

    $\mathsf{cpt}(\vdash \psi_{s_i}, c_1, \mathsf{cpt}(\ldots\mathsf{cpt}(\vdash \psi_{s_n}, c_1, c_2)\ldots)) \rightsquigarrow^* c_1$。
- 如果$\phi = AF_x(\psi)(s)$，而且$\{s_1, \ldots, s_n\} = \mathsf{Next}(s)$，那么
  - ($\Rightarrow$) 对任意两个不同的CPT c_1和c_2都有$\mathsf{cpt}(\Gamma \vdash AF_x(\psi)(s), c_1, c_2) \rightsquigarrow^* c_1$，那么$\vdash (s/x)\psi$是可证的，或者$\Gamma, AF_x(\psi)(s) \vdash AF_x(\psi)(s_1)$, $\Gamma, AF_x(\psi)(s) \vdash AF_x(\psi)(s_2)$, ..., $\Gamma, AF_x(\psi)(s) \vdash AF_x(\psi)(s_n)$都是可证的。否则，如果$\vdash (s/x)\psi$是不可证的，而且存在$1 \leq i \leq n$使得$\Gamma, AF_x(\psi)(s) \vdash AF_x(\psi)(s_i)$是不可证的，而且对任意$j < i$，$\Gamma, AF_x(\psi)(s) \vdash AF_x(\psi)(s_j)$是可证的，那

么根据归纳假设，存在两个不同的CPT c_1和c_2使得（令$\Gamma' = \Gamma, AF_x(\psi)(s)$）

$\mathsf{cpt}(\Gamma \vdash AF_x(\psi)(s), c_1, c_2) \rightsquigarrow$
$\mathsf{cpt}(\vdash (s/x)\psi, c_1, \mathsf{cpt}(\Gamma' \vdash AF_x(\psi)(s_1), \mathsf{cpt}(\ldots$
$\mathsf{cpt}(\Gamma' \vdash AF_x(\psi)(s_n), c_1, c_2)\ldots, c_2), c_2)) \rightsquigarrow^*$
$\mathsf{cpt}(\Gamma' \vdash AF_x(\psi)(s_1), \mathsf{cpt}(\ldots\mathsf{cpt}(\Gamma' \vdash AF_x(\psi)(s_n), c_1, c_2)\ldots, c_2), c_2) \rightsquigarrow^*$
$\ldots \rightsquigarrow^*$
$\mathsf{cpt}(\Gamma' \vdash AF_x(\psi)(s_i), \mathsf{cpt}(\ldots\mathsf{cpt}(\Gamma' \vdash AF_x(\psi)(s_n), c_1, c_2)\ldots, c_2), c_2) \rightsquigarrow^* c_2 \not\rightsquigarrow^* c_1$。因此，由证明系统的规则可知，$\Gamma \vdash AF_x(\psi)(s)$是可证的。

- ($\Leftarrow$) 如果$\Gamma \vdash AF_x(\psi)(s)$是可证的，那么由证明系统的规则可知，$\vdash (s/x)\psi$是可证的，或者$\Gamma, AF_x(\psi)(s) \vdash AF_x(\psi)(s_1), \Gamma, AF_x(\psi)(s) \vdash AF_x(\psi)(s_2), \ldots, \Gamma, AF_x(\psi)(s) \vdash AF_x(\psi)(s_n)$都是可证的。因此，根据归纳假设（令$\Gamma' = \Gamma, AF_x(\psi)(s)$），
  - 如果$\vdash (s/x)\psi$是可证的，那么对任意两个不同的CPT c_1和c_2都有
    $\mathsf{cpt}(\Gamma \vdash AF_x(\psi)(s), c_1, c_2) \rightsquigarrow$
    $\mathsf{cpt}(\vdash (s/x)\psi, c_1, \mathsf{cpt}(\Gamma' \vdash AF_x(\psi)(s_1),$
    $\mathsf{cpt}(\ldots\mathsf{cpt}(\Gamma' \vdash AF_x(\psi)(s_n), c_1, c_2)\ldots, c_2),$
    $c_2)) \rightsquigarrow^* c_1$。
  - 如果$\vdash (s/x)\psi$是不可证的，而对任意$1 \leq i \leq n$，$\Gamma, AF_x(\psi)(s) \vdash AF_x(\psi)(s_i)$是可证的，那么根据归纳假设，对任意两个不同的CPT c_1和c_2都有
    $\mathsf{cpt}(\Gamma \vdash AF_x(\psi)(s), c_1, c_2) \rightsquigarrow$
    $\mathsf{cpt}(\vdash (s/x)\psi, c_1, \mathsf{cpt}(\Gamma' \vdash AF_x(\psi)(s_1),$
    $\mathsf{cpt}(\ldots\mathsf{cpt}(\Gamma' \vdash AF_x(\psi)(s_n), c_1, c_2)\ldots, c_2),$
    $c_2)) \rightsquigarrow^*$
    $\mathsf{cpt}(\Gamma' \vdash AF_x(\psi)(s_1), \mathsf{cpt}(\ldots\mathsf{cpt}(\Gamma' \vdash AF_x(\psi)(s_n), c_1, c_2)\ldots, c_2), c_2) \rightsquigarrow^*$
    $\ldots \rightsquigarrow^*$
    $\mathsf{cpt}(\Gamma' \vdash AF_x(\psi)(s_n), c_1, c_2) \rightsquigarrow^* c_1$。

- 如果$\phi = EG_x(\psi)(s)$，而且$\{s_1, \ldots, s_n\} = \mathsf{Next}(s)$，那么
  - ($\Rightarrow$) 如果对任意两个不同的CPT c_1和c_2都有$\mathsf{cpt}(\Gamma \vdash EG_x(\psi), c_1, c_2) \rightsquigarrow^* c_1$，那么 $EG_x(\psi)(s) \in \Gamma$，或者$\vdash (s/x)\psi$是可证的以及存在$1 \leq i \leq n$使

得 $\Gamma, EG_x(\psi)(s) \vdash EG_x(\psi)(s_i)$是可证的。否则（令$\Gamma' = \Gamma, EG_x(\psi)(s)$），

* 如果$EG_x(\psi)(s) \notin \Gamma$，而且$\vdash (s/x)\psi$是不可证的，那么存在不同的两个CPT c_1和c_2使得

  $\mathsf{cpt}(\Gamma \vdash EG_x(\psi)(s), c_1, c_2) \rightsquigarrow^*$

  $\mathsf{cpt}(\vdash (s/x)\psi, \mathsf{cpt}(\Gamma' \vdash EG_x(\psi)(s_1), c_1,$

  $\mathsf{cpt}(...\mathsf{cpt}(\Gamma' \vdash EG_x(\psi)(s_n), c_1, c_2)...)),$

  $c_2) \rightsquigarrow^* c_2 \not\rightsquigarrow^* c_1$。

* 如果$EG_x(\psi)(s) \notin \Gamma$，$\vdash (s/x)\psi$是可证的，而且对任意$1 \leq i \leq n$，$\Gamma, EG_x(\psi)(s) \vdash EG_x(\psi)(s_i)$都是不可证的，那么根据归纳假设，存在不同的两个CPT c_1和c_2使得

  $\mathsf{cpt}(\Gamma \vdash EG_x(\psi)(s), c_1, c_2) \rightsquigarrow^*$

  $\mathsf{cpt}(\vdash (s/x)\psi, \mathsf{cpt}(\Gamma' \vdash EG_x(\psi)(s_1), c_1,$

  $\mathsf{cpt}(...\mathsf{cpt}(\Gamma' \vdash EG_x(\psi)(s_n), c_1, c_2)...)),$

  $c_2) \rightsquigarrow^*$

  $\mathsf{cpt}(\Gamma' \vdash EG_x(\psi)(s_1), c_1, \mathsf{cpt}(...\mathsf{cpt}(\Gamma' \vdash EG_x(\psi)(s_n), c_1, c_2)...)) \rightsquigarrow^*$

  $\ldots$

  $\mathsf{cpt}(\Gamma' \vdash EG_x(\psi)(s_n), c_1, c_2) \not\rightsquigarrow^* c_1$。

因此，由证明系统的规则可知，$\Gamma \vdash EG_x(\psi)(s)$是可证的。

– ($\Leftarrow$) 如果$\Gamma \vdash EG_x(\psi)(s)$是可证的，那么由证明系统的规则可知，$EG_x(\psi)(s) \in \Gamma$，或者$\vdash (s/x)\psi$是可证的以及存在$1 \leq i \leq n$使得$\Gamma, EG_x(\psi)(s) \vdash EG_x(\psi)(s_i)$是可证的。

* 如果$EG_x(\psi)(s) \in \Gamma$，那么对任意两个不同的CPT c_1和c_2都有$\mathsf{cpt}(\Gamma \vdash EG_x(\psi)(s), c_1, c_2) \rightsquigarrow c_1$。

* 如果$\vdash (s/x)\psi$是可证的，以及存在$1 \leq i \leq n$使得$\Gamma, EG_x(\psi)(s) \vdash EG_x(\psi)(s_i)$是可证的，而且对任意$j < i$，$\Gamma, EG_x(\psi)(s) \vdash EG_x(\psi)(s_j)$都不是可证的，那么根据归纳假设，对任意两个不同的CPT c_1和c_2都有（令$\Gamma' = \Gamma, EG_x(\psi)(s)$）

  $\mathsf{cpt}(\Gamma \vdash EG_x(\psi)(s), c_1, c_2) \rightsquigarrow^*$

  $\mathsf{cpt}(\vdash (s/x)\psi, \mathsf{cpt}(\Gamma' \vdash EG_x(\psi)(s_1), c_1,$

  $\mathsf{cpt}(...\mathsf{cpt}(\Gamma' \vdash EG_x(\psi)(s_n), c_1, c_2)...)),$

  $c_2) \rightsquigarrow^*$

  $\mathsf{cpt}(\Gamma' \vdash EG_x(\psi)(s_1), c_1, \mathsf{cpt}(...\mathsf{cpt}(\Gamma' \vdash EG_x(\psi)(s_n), c_1, c_2)...)) \rightsquigarrow^*$

$$\ldots \rightsquigarrow^*$$

$$\mathsf{cpt}(\Gamma' \vdash EG_x(\psi)(s_i), c_1, \mathsf{cpt}(\ldots\mathsf{cpt}(\Gamma' \vdash EG_x(\psi)(s_n), c_1, c_2)\ldots)) \rightsquigarrow^* c_1。$$

- 如果$\phi = AR_{x,y}(\phi_1, \phi_2)(s)$，由于均为余归纳公式，因此对$AR$公式的分析与$EG$类似。

- 如果$\phi = EU_{x,y}(\phi_1, \phi_2)(s)$，由于均为归纳公式，因此对$EU$公式的分析与$AF$类似。

□

1.3.3 证明搜索策略的优化

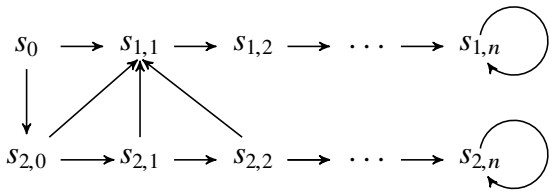
在余归纳公式（尤其是EG和AR公式）的证明搜索中，我们通常利用merge规则来证明某个性质在无穷路径上满足。对于每一个merge规则，上下文Γ（我们称之为merge）中的公式都是一种类型的，而且都与要证明的公式是相同类型的，唯一的不同是公式中相应的状态常量不同。本质上来说，每个merge对应的是无穷路径上的状态，其中所有的状态都满足某个性质。因此，在merge规则的实现中，我们可以对于每一种公式类型，只在Γ中记录状态常量。

需要注意的是，对于归纳公式（尤其是AF和EU公式），虽然在证明规则中没有merge规则，但是在证明搜索中仍然需要merge。对于这些公式，merge的存在可避免无穷的证明搜索。这是由于当归纳公式是不可证的时候，即它们的非（余归纳公式）是可证的时候，我们需要merge来表示一个无穷路径，而且这个无穷路径上的状态都不满足某些性质。对于归纳公式，之所以在证明规则中没有merge，而在证明搜索中需要merge是因为，在证明规则中我们只关心证明树的形状，而不是证明树的构造过程，只有在证明树的构造过程中才需要记录归纳公式的merge。因此，作为对证明搜索策略的另一个优化，我们需要在归纳公式的证明搜索中同样记录merge。

另外，在证明搜索过程中，我们对每个子公式都用用一个全局的数据结构来保存访问过的状态，以避免在证明子公式的时候同一个状态被重复访问。在证明搜索的实际实现中，用来记录状态集合的全局数据结构既可以是哈希表也可以是BDD。两种数据结构各有优缺点：当模型中的状态变量绝大多数为布尔类型的时候，用BDD往往能减少空间的占用；而当模型中包含许多非布尔类型的状态变量时，如果用BDD记录的话则需将所有的非布尔变量转化成布尔变量，这个转化的过程会增多状态变量的个数，从而消耗掉更多的时间与空间，因此

这时用哈希表来记录状态往往效率更高。接下来，我们利用一个例子来说明利用全局数据结构来记录状态之后，验证的效率会提升。

例子 1.5. 考虑一个具有如下状态迁移图的*Kripke*模型，其中$n > 1$，而且在该模型中有集合$P = \{s_0, s_{1,1}, s_{1,2}, ..., s_{1,n-1}, s_{2,0}, ..., s_{2,n}\}$。



在该模型中，我们考虑公式$EG_x(P(x))(s_0)$的验证，为了验证该公式，我们需要重写*CPT* $cpt(\vdash EG_x(P(x))(s_0), \mathrm{t}, \mathrm{f})$。设在集合$Next(s_0)$中，对状态$s_{1,1}$的搜索在状态$s_{2,0}$之前；对于任意的$0 \leq i \leq n-1$，在集合$Next(s_{2,i})$中，对状态$s_{1,1}$的搜索在状态$s_{2,i+1}$之前。重写步骤如图*1.5*所示，其中，令

$$\Gamma_0 = \{EG_x(P(x))(s_0)\},$$

$$\Gamma_{1,i} = \{EG_x(P(x))(s_0), EG_x(P(x))(s_{1,0}), \ldots, EG_x(P(x))(s_{1,i})\},$$

以及

$$\Gamma_{2,i} = \{EG_x(P(x))(s_0), EG_x(P(x))(s_{2,0}), \ldots, EG_x(P(x))(s_{2,i})\}$$

在图*1.5*中，在重写$cpt(\Gamma_{2,0} \vdash EG_x(P(x))(s_{2,1}), \mathrm{t}, \mathrm{f})$之前，路径$\pi = s_{1,1}, s_{1,2}, ..., s_{1,n}$上的所有的状态都已被访问过一遍。而且，对于任意$0 \leq i \leq n-1$，在每次重写$cpt(\Gamma_{2,i} \vdash EG_x(P(x))(s_{2,i+1}), \mathrm{t}, \mathrm{f})$之前，$\pi$上的所有状态都会重新被访问一遍。

当我们用一个全局数据结构M来记录访问在证明EG公式的过程中访问过的状态时，可以避免π被重复访问。在本例中，我们将所有不满足公式$EG_x(P(x))(s)$的状态s存入到M中，而且当重写$c = cpt(\Gamma \vdash EG_x(P(x))(s), c_1, c_2)$的时候，如果$s \in M$，那么直接在当前的重写步骤中将$c$替换成$c_2$。因此在本例中，对于任意的$0 \leq i \leq n-1$，在重写$cpt(\Gamma_{2,i} \vdash EG_x(P(x))(s_{2,i+1}), \mathrm{t}, \mathrm{f})$之前，$\pi$上的所有状态都已记录在$M$中，因此不用重复被访问。

有关全局数据结构的详细解释见下一小节。

$\mathsf{cpt}(\vdash EG_x(P(x))(s_0), \mathsf{t}, \mathsf{f}) \rightsquigarrow^*$

$\mathsf{cpt}(\Gamma_0 \vdash EG_x(P(x))(s_{1,1}), \mathsf{t}, \mathsf{cpt}(\Gamma_0 \vdash EG_x(P(x))(s_{2,0}), \mathsf{t}, \mathsf{f})) \rightsquigarrow^*$

$\mathsf{cpt}(\Gamma_{1,0} \vdash EG_x(P(x))(s_{1,2}), \mathsf{t}, \mathsf{cpt}(\Gamma_0 \vdash EG_x(P(x))(s_{2,0}), \mathsf{t}, \mathsf{f})) \rightsquigarrow^*$

$\ldots \rightsquigarrow^*$

$\mathsf{cpt}(\Gamma_{1,n-1} \vdash EG_x(P(x))(s_{1,n}), \mathsf{t}, \mathsf{cpt}(\Gamma_0 \vdash EG_x(P(x))(s_{2,0}), \mathsf{t}, \mathsf{f})) \rightsquigarrow^*$

$\mathsf{cpt}(\Gamma_0 \vdash EG_x(P(x))(s_{2,0}), \mathsf{t}, \mathsf{f}) \rightsquigarrow^*$

$\mathsf{cpt}(\Gamma_{2,0} \vdash EG_x(P(x))(s_{1,1}), \mathsf{t}, \mathsf{cpt}(\Gamma_{2,0} \vdash EG_x(P(x))(s_{2,1}), \mathsf{t}, \mathsf{f})) \rightsquigarrow^*$

$\ldots \rightsquigarrow^*$

$\mathsf{cpt}(\Gamma_{2,0} \vdash EG_x(P(x))(s_{2,1}), \mathsf{t}, \mathsf{f}) \rightsquigarrow^*$

$\ldots \rightsquigarrow^*$

$\mathsf{cpt}(\Gamma_{2,n-1} \vdash EG_x(P(x))(s_{2,n}), \mathsf{t}, \mathsf{f}) \rightsquigarrow^*$

$\mathsf{cpt}(\Gamma_{2,n} \vdash EG_x(P(x))(s_{2,n}), \mathsf{t}, \mathsf{f}) \rightsquigarrow \mathsf{t}$

图 1.5 cpt($\vdash EG_x(P(x))(s_0), \mathsf{t}, \mathsf{f}$)**的重写步骤**

1.3.4 证明搜索策略的伪代码

在本小节，我们给出以上所述的证明搜索策略的中用到的数据结构及伪代码，如图1.6所示。

在解释该伪代码之前，我们需要介绍公式模式的概念：对于以模态词AF、EG、AR、EU开头的公式ϕ，用_来代替ϕ中的常量后得到即为ϕ的公式模式ϕ^-。比如，$EU_{x,y}(\phi_1, \phi_2)(s)$的模式为$EU_{x,y}(\phi_1, \phi_2)(_)$；$AF_x(\phi')(s')$的模式为$AF_x(\phi')(_)$。一个公式的模式可以看作是当前公式中状态常量的上下文。

在该伪代码中， c是当前需要被重写的CPT；pt和ce是在证明搜索的过程构造的树结构，分别指的是证明树和反例；$\mathsf{M^t}$和$\mathsf{M^f}$记录的是，对于当前要证明的公式的每一个子公式，分别使得该子公式满足与不满足的状态的集合；visited记录的是在对于每个以AF、EG、AR、EU开头的公式的证明过程中访问过的状态集合。

需要注意的是，对于伪代码中重写步骤中的每个CPT c，我们人为关联一个动作集合A，使得当c是当前需要重写的CPT时，执行A中的所有动作。将CPT关联动作集合的目的是为了构造证明树和反例，以及更新$\mathsf{M^t}$和$\mathsf{M^f}$。在伪代码的初始时刻，我们分别给三个CPT $\mathsf{cpt}(\vdash \phi, \mathsf{t}, \mathsf{f})$、 t、f均关联一个空动作集合，其中$\phi$是要验证的公式。

Input: A CPT c.

Output: A pair (r, t), where r is a Boolean, and t is either pt or ce.

```
1:  function ProofSearch
2:      c := cpt^∅(⊢ ψ, t^∅, f^∅)
3:      pt := ce := ⟨tree with a single node: ⊢ ψ⟩
4:      M^t := M^f := visited := ⟨empty hash table⟩
5:      while c = cpt^{A_0}(` ⊢ φ, c_1^{A_1}, c_2^{A_2}) do
6:          ∀a ∈ A_0, perform a
7:          case φ is
8:            ⊤: c := c_1^{A_1}
9:            ⊥: c := c_2^{A_2}
10:           P(s_1, ..., s_n):
11:             if ⟨s_1, ..., s_n⟩ ∈ P then c := c_1^{A_1} else c := c_1^{A_2} end if
12:           ¬P(s_1, ..., s_n):
13:             if ⟨s_1, ..., s_n⟩ ∈ P then c := c_1^{A_2} else c := c_1^{A_1} end if
14:           φ_1 ∧ φ_2 : ProveAnd(⊢ φ_1 ∧ φ_2)
15:           φ_1 ∨ φ_2 : ProveOr(⊢ φ_1 ∨ φ_2)
16:           EX_x(φ_1)(s) : ProveEX(` ⊢ EX_x(φ_1)(s))
17:           AX_x(φ_1)(s) : ProveAX(` ⊢ AX_x(φ_1)(s))
18:           EG_x(φ_1)(s) : ProveEG(` ⊢ EG_x(φ_1)(s))
19:           AF_x(φ_1)(s) : ProveAF(` ⊢ AF_x(φ_1)(s))
20:           EU_{x,y}(φ_1, φ_2)(s) : ProveEU(` ⊢ EU_{x,y}(φ_1, φ_2)(s))
21:           AR_{x,y}(φ_1, φ_2)(s) : ProveAR(` ⊢ AR_{x,y}(φ_1, φ_2)(s))
22:         end case
23:     end while
24:     if c = t^A then
25:         ∀a ∈ A, perform a
26:         return (true, pt)
27:     end if
28:     if c = f^A then
29:         ∀a ∈ A, perform a
30:         return (false, ce)
31:     end if
32: end function
```

图 1.6 证明搜索策略的伪代码

另外需要注意的是，该伪代码中，三个全局变量M^t、M^f、visited的值都是哈希表。实际上，这三个全局变量都是对于每个公式模式记录一个状态集合，因此，M^t、M^f、visited均可看作是以公式模式为键，以状态集合为值的哈希表。在该伪代码中，$M^t_{EG_x(\phi)(_)}$用来表示一个状态集合S，其中$\forall s \in S$，$EG_x(\phi)(s)$是可证

的；$M^{f}_{EG_x(\phi)(_)}$用来表示一个集合S，其中$\forall s \in S$，$EG_x(\phi)(s)$是不可证的；$visited_{EG_x(\phi)(_)}$用来表示一个集合S，其中$\forall s \in S$， s在证明以$EG_x(\phi)(_)$为模式的公式的过程中已被访问过。

该伪代码中的“while”循环的作用是重复重写CPT c，直到重写到t或f。该伪代码的返回值是一个由一个布尔值和一棵树组成的二元组，布尔值表示要验证的公式的满足与否，树则表示此公式的证明树或者反例。

该伪代码中，CPT的重写方式由公式ϕ的形状决定，其中ϕ为原子公式和原子公式的非的情况在伪代码主体（表1.6）中给出，对于ϕ的更复杂的情况，我们在接下来的小节中依次加以讨论。

1.3.4.1 ProveAnd和ProveOr

1: let $A_{12} = A_1 \cup \{ac(pt, \vdash \phi_1 \wedge \phi_2, \{\vdash \phi_1, \vdash \phi_2\})\}$
2: let $A_{21} = A_2 \cup \{ac(ce, \vdash \phi_1 \wedge \phi_2, \{\vdash \phi_1\})\}$
3: let $A_{22} = A_2 \cup \{ac(ce, \vdash \phi_1 \wedge \phi_2, \{\vdash \phi_2\})\}$
4: let $c' = cpt^{\emptyset}(\vdash \phi_1, cpt^{\emptyset}(\vdash \phi_2, c_1^{A_{12}}, c_2^{A_{22}}), c_2^{A_{21}})$
5: c := c′

图 1.7 ProveAnd($\vdash \phi_1 \wedge \phi_2$)

该伪代码中合取公式的证明搜索如图1.7所示，其中ac(t, parent, children)表示将children的每个元素都作为证明树t上parent的子节点。ProveAnd的解释如下：

- 第4行：当从c′重写到c_1时，由于$\vdash \phi_1$和$\vdash \phi_2$均可证，那么将$\vdash \phi_1$和$\vdash \phi_2$都加在证明树中作为$\vdash \phi_1 \wedge \phi_2$的子节点（第1行）。否则，当从c′重写到外部的$c_2$或内部的$c_2$时，由于$\vdash \phi_1$或$\vdash \phi_2$是不可证的，这时将$\vdash \phi_1$或$\vdash \phi_2$加到反例中作为$\vdash \phi_1 \wedge \phi_2$的子节点（第2、3行）。
- 第5行: 将c重写到c′。

ProveOr是ProveAnd的对偶情况，伪代码细节如图1.8所示。

1.3.4.2 ProveEX和ProveAX

ProveEX的伪代码细节如图1.9所示，其中令$\{s_1, ..., s_n\}$ = Next(s)。ProveEX与ProveAnd的分析过程类似，不同点在于第4、5行，当c′重写到第i个c_1的时候，$\vdash (s_i/x)\phi_1$应该作为$\vdash EX_x(\phi_1)(s)$的子节点被加入到证明树中（第3行）；相反，当c′重写到c_2的

1: let $A_{22} = A_2 \cup \{ac(ce, \vdash \phi_1 \vee \phi_2, \{\vdash \phi_1, \vdash \phi_2\})\}$
2: let $A_{11} = A_1 \cup \{ac(pt, \vdash \phi_1 \vee \phi_2, \{\vdash \phi_1\})\}$
3: let $A_{12} = A_1 \cup \{ac(pt, \vdash \phi_1 \vee \phi_2, \{\vdash \phi_2\})\}$
4: let $c' = cpt^{\emptyset}(\vdash \phi_1, c_1^{A_{11}}, cpt^{\emptyset}(\vdash \phi_2, c_1^{A_{12}}, c_2^{A_{22}}))$
5: $c := c'$

图 1.8 ProveOr($\vdash \phi_1 \vee \phi_2$)

1: */* For notation purpose, here we refer "k" to $EX_x(\phi_1)(\_)$, and "k(s)" to $EX_x(\phi_1)(s)$.*/*
2: $A_2 := A_2 \cup \{ac(ce, \vdash k(s), \{\vdash (s_1/x)\phi_1, ..., \vdash (s_n/x)\phi_1\})\}$
3: $\forall i \in \{1, ..., n\}$, let $A_{1i} = A_1 \cup \{ac(pt, \vdash k(s), \{\vdash (s_i/x)\phi_1\})\}$
4: let $c' = cpt^{\emptyset}(\vdash (s_1/x)\phi_1, c_1^{A_{11}}, cpt^{\emptyset}(...cpt^{\emptyset}(\vdash (s_n/x)\phi_1, c_1^{A_{1n}}, c_2^{A_2})...))$
5: $c := c'$

图 1.9 ProveEX($\vdash EX_x(\phi_1)(s)$)

时候，$\vdash (s_1/x)\phi_1, ..., \vdash (s_n/x)\phi_1$ 都应该作为$\vdash EX_x(\phi_1)(s)$的子节点被加入到反例中（第2行）。

ProveAX是ProveEX的对偶情况，伪代码细节如图1.10所示。

1: */*For notation purpose, here we refer "k" to $AX_x(\phi_1)(\_)$, and "k(s)" to $AX_x(\phi_1)(s)$.*/*
2: $A_1 := A_1 \cup \{ac(pt, \vdash k(s), \{\vdash (s_1/x)\phi_1, ..., \vdash (s_n/x)\phi_1\})\}$
3: $\forall i \in \{1, ..., n\}$, let $A_{2i} = A_2 \cup \{ac(ce, \vdash k(s), \{\vdash (s_i/x)\phi_1\})\}$
4: let $c' = cpt^{\emptyset}(\vdash (s_1/x)\phi, cpt^{\emptyset}(...cpt^{\emptyset}(\vdash (s_n/x)\phi, c_1^{A_1}, c_2^{A_{2n}}), ...), c_2^{A_{21}})$
5: $c := c'$

图 1.10 ProveAX($\vdash AX_x(\phi_1)(s)$)

1.3.4.3 ProveEG and ProveAF

ProveEG的伪代码细节如图1.11所示，其中，states(`)表示`中出现的状态，并且令$\{s_1, ..., s_n\} = Next(s)$。 ProveEG的解释如下：

- 第3、4行：如果已知$EG_x(\phi_1)(s)$是不可证的，那么将c重写到c_2。
- 第5 – 8行：如果已知$EG_x(\phi_1)(s)$是可证的，或者可应用merge规则，那么将c重写到c_1，同时，由于`中的每个公式都是可证的，因此将`中出现的所有状态加入到$M^t_{EG_x(\phi_1)(_)}$中。另外，由于状态的访问是深度优先的，因此将所有的访问过的除了在`出现的状态之外都加入集合$M^f_{EG_x(\phi_1)(_)}$中。

1: */* For notation purpose, here we refer "k" to $EG_x(\phi_1)(\_)$ and "k(s)" to $EG_x(\phi_1)(s)$,*
2: *and let* Γ' *be* $\Gamma \cup \{k(s)\}$.**/*
3: **if** $s \in M_k^f$ **then**
4: $c := c_2^{A_2}$
5: **else if** $s \in M_k^t$ or $k(s) \in \Gamma$ **then**
6: $c := c_1^{A_1}$
7: $M_k^t := M_k^t \cup states(\Gamma)$
8: $M_k^f := M_k^f \cup visited_k \setminus states(\Gamma)$
9: **else**
10: let $A_{20} = A_2 \cup \{ac(ce, \Gamma \vdash k(s), \{\vdash (s/x)\phi_1\})\}$
11: let $A_{2n} = A_2 \cup \{ac(ce, \Gamma \vdash k(s), \{\Gamma' \vdash k(s_1), ..., \Gamma' \vdash k(s_n)\})\}$
12: $\forall i \in \{1, ..., n\}$, let
13: $A_{1i} = A_1 \cup \{ac(pt, \Gamma \vdash k(s), \{\vdash (s/x)\phi_1, \Gamma' \vdash k(s_i)\})\}$
14: **if** $\Gamma = \emptyset$ **then**
15: $visited_k := \{s\}$
16: **else**
17: $visited_k := visited_k \cup \{s\}$
18: **end if**
19: $A_{20} := A_{20} \cup \{M_k^f := M_k^f \cup \{s\}\}$;
20: $A_{2n} := A_{2n} \cup \{M_k^f := M_k^f \cup \{s\}\}$;
21: let $c' = cpt^{\emptyset}(\vdash (s/x)\phi_1, cpt^{\emptyset}(\Gamma' \vdash k(s_1), c_1^{A_{11}}, cpt^{\emptyset}(...cpt^{\emptyset}(\Gamma' \vdash k(s_n), c_1^{A_{1n}}, c_2^{A_{2n}})...)), c_2^{A_{20}})$
22: $c := c'$
23: **end if**

图 1.11 ProveEG($\Gamma \vdash EG_x(\phi_1)(s)$)

- 第21行： CPT c′的构造方式如下：

  1. 如果c′重写到外层的c_2，那么$\vdash (s/x)\phi_1$是不可证的，因此将$\vdash (s/x)\phi_1$作为$\Gamma \vdash EG_x(\phi_1)(s)$的子节点加到反例中（第10行）。另外，如果$\Gamma \vdash EG_x(\phi_1)(s)$是不可证的，那么不存在以$s$开头的无穷路径使得该路径上的所有状态都满足$\phi_1$，在这种情况下，我们将$s$加入到$M^f_{EG_x(\phi_1)(_)}$中（第19、20行）。

  2. 如果c′重写到内层的c_2，那么$\Gamma' \vdash EG_x(\phi_1)(s_1), ..., \Gamma' \vdash EG_x(\phi_1)(s_n)$都是不可证的，因此将其都作为$\Gamma \vdash EG_x(\phi_1)(s)$的子节点加入到反例中（第11行）。

  3. 如果c′重写到第i个c_1，那么$\vdash (s/x)\phi_1$和`′ $\vdash EG_x(\phi_1)(s_i)$都是可证的，因此将其都作为` $\vdash EG_x(\phi_1)(s)$的子节点加入到证明树中（第12、13行）。

1: /* For notation purpose, here we refer "k" to $AF_x(\phi_1)(\_)$ and "k(s)" to $AF_x(\phi_1)(s)$,
2: and let Γ' be $\Gamma \cup \{k(s)\}$..*/
3: **if** $s \in M_k^t$ **then**
4: $c := c_1^{A_1}$
5: **else if** $s \in M_k^f$ or $k(s) \in \Gamma$ **then**
6: $c := c_2^{A_2}$
7: $M_k^f := M_k^f \cup states(\Gamma)$
8: $M_k^t := M_k^t \cup visited_k \setminus states(\Gamma)$
9: **else**
10: let $A_{10} = A_1 \cup \{ac(pt, \Gamma \vdash k(s), \{\vdash (s/x)\phi_1\})\})\}$
11: let $A_{1n} = A_1 \cup \{ac(pt, \Gamma \vdash k(s), \{\Gamma' \vdash k(s_1), ..., \Gamma' \vdash k(s_n)\})\}$
12: $\forall i \in \{1, ..., n\}$, let
13: $A_{2i} = A_2 \cup \{ac(ce, \Gamma \vdash k(s), \{\vdash (s/x)\phi_1, \Gamma' \vdash k(s_i)\})\}$
14: **if** ` $= \emptyset$ **then**
15: $visited_k := \{s\}$
16: **else**
17: $visited_k := visited_k \cup \{s\}$
18: **end if**
19: $A_{10} := A_{10} \cup \{M_k^t := M_k^t \cup \{s\}\}$
20: $A_{1n} := A_{1n} \cup \{M_k^t := M_k^t \cup \{s\}\}$;
21: let $c' = cpt^{\emptyset}(\vdash (s/x)\phi_1, c_1^{A_{10}}, cpt^{\emptyset}(\Gamma' \vdash k(s_1), cpt^{\emptyset}(...cpt^{\emptyset}(\Gamma' \vdash k(s_n), c_1^{A_{1n}}, c_2^{A_{2n}})...), c_2^{A_{21}}))$
22: $c := c'$
23: **end if**

图 1.12 ProveAF($\Gamma \vdash AF_x(\phi_1)(s)$)

- 第22行：将c重写到c′。

ProveAF是ProveEG的对偶情况，伪代码细节如图1.12所示。

1.3.4.4 ProveEU和ProveAR

ProveEU的伪代码细节要比ProveEG更加复杂。原因是，对于后者我们只需找到一个满足某个公式的环即可，而对于前者我们需要找到一条终点状态满足某公式的有穷路径，而且在找到这条路径之前可能已访问若干个环。ProveEU的伪代码细节如图1.13所示。在ProveEG中，我们令$\{s_1, ..., s_n\} = Next(s)$；令reachable(S)表示能经过有穷路径到达S的某个状态的状态集合。ProveEU的解释如下：

- 第3、4行：如果$EU_{x,y}(\phi_1, \phi_2)(s)$是可证的，那么将c重写到$c_1$。
- 第5、6行：如果$EU_{x,y}(\phi_1, \phi_2)(s)$是不可证的，或者$EU_{x,y}(\phi_1, \phi_2)(s) \in \Gamma$，那么将c重写到$c_2$。

1: /* For notation purpose, here we refer "k" to $EU_{x,y}(\phi_1, \phi_2)(\_)$ and "k(s)" to $EU_{x,y}(\phi_1, \phi_2)(s)$,
2: and let Γ' be $\Gamma \cup \{k(s)\}$. */
3: **if** $s \in M_k^t$ **then**
4: $c := c_1^{A_1}$
5: **else if** $s \in M_k^f$ or $k_s \in `$ **then**
6: $c := c_2^{A_2}$;
7: **else**
8: let $A_{10} = A_1 \cup \{$
9: $ac(pt, ` \vdash k(s), \{\vdash (s/y)\phi_2\})$,
10: $M_k^t := M_k^t \cup reachable(states(\Gamma) \cup \{s\})$,
11: $M_k^f := (M_k^f \cup visited_k) \setminus reachable(states(\Gamma) \cup \{s\})$
12: let $A_{20} = A_2 \cup \{ac(ce, \Gamma \vdash k(s), \{\vdash (s/x)\phi_1, \vdash (s/y)\phi_2\})\}$
13: let $A_{2n} = A_2 \cup \{ac(ce, \Gamma \vdash k(s), \{\Gamma' \vdash k(s_1), ..., \Gamma' \vdash k(s_n)\})\}$
14: $\forall i \in \{1, ..., n\}$, let
15: $A_{1i} = A_1 \cup \{$
16: $ac(pt, \Gamma \vdash k(s), \{\Gamma' \vdash k(s_i)\})$,
17: $M_k^t := M_k^t \cup reachable(states(\Gamma') \cup \{s_i\})$,
18: $M_k^f := (M_k^f \cup visited_k) \setminus reachable(states(\Gamma') \cup \{s_i\})\}$
19: **if** $\Gamma = \emptyset$ **then**
20: $visited_k := \{s\}$
21: **else**
22: $visited_k := visited_k \cup \{s\}$
23: **end if**
24: $A_{20} := A_{20} \cup \{M_k^f := M_k^f \cup \{s\}\}$
25: $A_{2n} := A_{2n} \cup \{M_k^f := M_k^f \cup \{s\}\}$
26: let $c' = cpt^{\emptyset}(\vdash (s/y)\phi_2, c_1^{A_{10}}, cpt^{\emptyset}(\vdash (s/x)\phi_1, cpt^{\emptyset}(\Gamma' \vdash k(s_1),$
27: $c_1^{A_{11}}, cpt^{\emptyset}(...cpt^{\emptyset}(\Gamma' \vdash k(s_n), c_1^{A_{1n}}, c_2^{A_{2n}})...)), c_2^{A_{20}}))$
28: $c := c'$
29: **end if**

图 1.13 ProveEU($\Gamma \vdash EU_{x,y}(\phi_1, \phi_2)(s)$)

- 第26、27行：CPT c′的构造方式如下：

    1. 如果c′将来能重写到第一个c_1，那么$\vdash (s/y)\phi_2$是可证的，因此将其作为$\Gamma \vdash EU_{x,y}(\phi_1, \phi_2)(s)$的子节点加到证明树中（第9行）。与此同时，由于$EU_{x,y}(\phi_1, \phi_2)(s)$是可证的，那么对于任意访问过的状态$s'$，如果存在以$s'$为起点的有穷路径而$s$在这条路径上，那么$EU_{x,y}(\phi_1, \phi_2)(s')$也是可证的。因此，可将状态集合$reachable(states(\Gamma) \cup \{s\})$中的所有状态加入到 $M^t_{EU_{x,y}(\phi_1,\phi_2)(_)}$中（第10行）。每个$reachable(states(\Gamma) \cup \{s\})$中的

1: /* For notation purpose, here we refer "k" to $AR_{x,y}(\phi_1, \phi_2)(\_)$ and "k(s)" to $AR_{x,y}(\phi_1, \phi_2)(s)$,
2: and let Γ' be $\Gamma \cup \{k(s)\}$. */
3: **if** $s \in M_k^f$ **then**
4: $c := c_2^{A_2}$
5: **else if** $s \in M_k^t$ or $k(s) \in \Gamma$ **then**
6: $c := c_1^{A_1}$;
7: **else**
8: let $A_{10} = A_1 \cup ac(pt, \Gamma \vdash k(s), \{\vdash (s/x)\phi_1, \vdash (s/y)\phi_2\})\}$
9: let $A_{1n} = A_1 \cup \{ac(pt, \Gamma \vdash k(s), \{ \Gamma' \vdash k(s_1), ..., \Gamma' \vdash k(s_n)\})\}$
10: let $A_{20} = A_2 \cup \{$
11: $ac(ce, \Gamma \vdash k(s), \{\vdash (s/y)\phi_2\})$,
12: $M_k^f := M_k^f \cup reachable(states(\Gamma) \cup \{s\})$,
13: $M_k^t := (M_k^t \cup visited_k) \setminus reachable(states(\Gamma) \cup \{s\})$
14: $\forall i \in \{1, ..., n\}$, let
15: $A_{2i} = A_2 \cup \{$
16: $ac(ce, \Gamma \vdash k(s), \{\vdash (s/x)\phi_1, \Gamma' \vdash k(s_i)\})$,
17: $M_k^f := M_k^f \cup reachable(states(\Gamma') \cup \{s_i\})$,
18: $M_k^t := (M_k^t \cup visited_k) \setminus reachable(states(\Gamma') \cup \{s_i\})\}$
19: **if** ` $= \emptyset$ **then**
20: $visited_k := \{s\}$
21: **else**
22: $visited_k := visited_k \cup \{s\}$
23: **end if**
24: $A_{10} := A_{10} \cup \{M_k^t := M_k^t \cup \{s\}\}$
25: $A_{1n} := A_{1n} \cup \{M_k^t := M_k^t \cup \{s\}\}$
26: let $c' = cpt^{\emptyset}(\vdash (s/y)\phi_2, cpt^{\emptyset}(\vdash (s/x)\phi_1, c_1^{A_{10}}, cpt^{\emptyset}(\Gamma' \vdash k(s_1),$
27: $cpt^{\emptyset}(...cpt^{\emptyset}(\Gamma' \vdash k(s_n), c_1^{A_{1n}}, c_2^{A_{2n}})...), c_2^{A_{21}})), c_2^{A_{20}})$
28: $c := c'$
29: **end if**

图 1.14 ProveAR($\Gamma \vdash AR_{x,y}(\phi_1, \phi_2)(s)$)

状态s′或者在 $states(\Gamma) \cup \{s\}$中，或者在与 $states(\Gamma) \cup \{s\}$重叠的某个环上。因此，通过记录并选取与$states(\Gamma) \cup \{s\}$重叠的所有的环，就可以计算状态集合 $reachable(states(\Gamma) \cup \{s\})$了。然后，我们将所有被访问过的状态加入到$M^f_{EU_{x,y}(\phi_1,\phi_2)(_)}$中，然后去掉可经过有穷路径到达$states(\Gamma) \cup \{s\}$中某个状态的状态（第11行）。

2. 如果c′能重写到第i个其他的c_1，那么$\vdash (s/x)\phi_1$和 $\Gamma' \vdash EU_{x,y}(\phi_1, \phi_2)(s_i)$是可证的，因此将其都作为$\Gamma \vdash EU_{x,y}(\phi_1, \phi_2)(s)$的子节点加入到证明树中（第16行）。与此同时，$reachable(states(\Gamma') \cup \{s_i\})$中的所有状态都应

加到$M^{t}_{EU_{x,y}(\phi_1,\phi_2)(_)}$中（第17行）；所有被访问过的状态都加到$M^{f}_{EU_{x,y}(\phi_1,\phi_2)(_)}$中并且去掉可经过有穷路径到达$states(\Gamma') \cup \{s_i\}$中某个状态的状态（第18行）。

3. 如果c′重写到外层的c_2，那么$\vdash (s/x)\phi_1$和$\vdash (s/y)\phi_2$是不可证的，因此将其都作为$\Gamma \vdash EU_{x,y}(\phi_1, \phi_2)(s)$的子节点加入到反例中（第12行）。
4. 如果c′重写到内层的c_2，那么$\Gamma' \vdash EU_{x,y}(\phi_1, \phi_2)(s_1)$, ..., $\Gamma' \vdash EU_{x,y}(\phi_1, \phi_2)(s_n)$都是不可证的，因此将其全部作为$\Gamma \vdash EU_{x,y}(\phi_1, \phi_2)(s)$的子节点加入到反例中（第13行）。
5. 如果$\Gamma \vdash EU_{x,y}(\phi_1, \phi_2)(s)$是不可证的，那么将$s$加入到$M^{f}_{EU_{x,y}(\phi_1,\phi_2)(_)}$中，并以此在对这个$EU$公式的证明搜索中避免重复访问s（第24、25行）。值得注意的是s可从$M^{f}_{EU_{x,y}(\phi_1,\phi_2)(_)}$被移除（第11 – 18行）。这种情况只当存在Γ′和s′使得$\Gamma' \vdash EU_{x,y}(\phi_1, \phi_2)(s')$是可证的，$s', \vdash (s'/y)\phi_2$是可证的，以及$s \in reachable(states(\Gamma') \cup \{s'\})$的时候发生。

- 第28行：将c重写到c′。

ProveAR是ProveEU的对偶情况，伪代码细节如图1.14所示。

1.4 公平性性质的验证

在这一小节中，我们介绍如何利用以上介绍的证明搜索策略完成公平性性质的验证。

在并发系统的形式化验证中，公平性性质是一类比较重要的性质。在验证公平性性质时，模型检测算法通常会规避系统的一些不切实际的行为[3]，比如在两个进程的互斥模型中，我们通常只考虑没有进程饿死的情况，即没有永远处在等待状态的进程。在这类模型的验证中，我们就可以添加一个公平性限制来规避模型的不公平的迁移（即在该模型中的状态迁移过程中有进程饿死），从而只在对两个进程都公平的前提下进行状态迁移并验证模型的性质。

由于无法描述特定路径的性质，因此带有公平性前提的性质无法用CTL或CTL_P公式来表示。然而，我们仍然可以利用SCTL系统的证明搜索策略来完成公平性性质的验证。本文中公平性的定义与McMillan提出的公平性定义[8]一致，即一个无穷路径是公平的指的是某种性质在该路径上无穷次成立。一个带有公平性前提的性质通常具有$E_C f$或者$A_C f$形式，其中C表示一个CTL_P公式的集合，而f表

示路径的性质。在本文中，$E_C G(\phi)(s)$在给定的Kripke模型中成立当且仅当存在以s为起始状态的一个无穷路径，集合C中的每个公式在该路径上的无穷次成立，而且在该路径上的每个状态上ϕ都成立，而且集合C中的每个公式ϕ'，ϕ'在π上的无穷个状态上都是成立的；$A_C F(\phi)(s)$在给定的Kripke模型中是有效的当且仅当对于所有以s为起始状态的无穷路径，集合C中的每个公式都在每条路径上无穷次成立，而且每条路径上都存在一个状态，使得ϕ在该状态上成立。另外，与McMillan的定义类似，带有别的模态词的公平性性质可用$E_C G$与$A_C F$公式进行定义：

$$E_C X_x(\phi)(t) = EX_x(\phi \wedge E_C G_x(\top)(x))(t)$$

$$A_C X_x(\phi)(t) = AX_x(\phi \vee A_C F_x(\bot)(x))(t)$$

$$E_C U_{x,y}(\phi_1, \phi_2)(t) = EU_{x,y}(\phi_1, \phi_2 \wedge E_C G_z(\top)(y))(t)$$

$$A_C R_{x,y}(\phi_1, \phi_2)(t) = AR_{x,y}(\phi_1, \phi_2 \vee A_C F_z(\bot)(y))(t)$$

根据以上定义可知，要验证公式$E_C G_x(\phi)(s)$，只需找到一条以s为起始状态的公平的无穷路径使得ϕ在这条路径上永远成立；要验证公式$A_C F(\phi)(s)$，只需确定不存在一条以s为其实状态的无穷路径使得ϕ在这条路径上永远不成立。因此，要利用SCTL的证明搜索策略来验证公平性性质，只需要一个机制来判断公平的无穷路径的存在与否。

由命题1.8与命题1.9可知，我们可以在证明搜索中利用merge来判断公平的无穷路径的存在与否：存在一条具有公平性前提C的无穷路径当且仅当存在一个merge，使得C中的每个公式都在该merge上的环中的某个状态上成立。

命题 1.8. *令C为一个CTL_P公式的集合，而且$\sigma = s_0, s_1, \ldots$是一条无穷路径，其中对于任意$i \geq 0$都有$s_i \rightarrow s_{i+1}$。如果$C$中的每个公式都在$\sigma$中无穷次成立，那么一定存在一条有穷路径$\sigma_f = s'_0, s'_1, \ldots, s'_n$，使得对任意$0 \leq i \leq n-1$都有$s'_j \rightarrow s'_{j+1}$而且存在$0 \leq p \leq n-1$使得$s'_n = s'_p$，每个状态$s'_j$都在$\sigma$出现，而且对于$C$中的任意公式，都存在某个状态$s'_q$使得该公式成立，其中$p \leq q \leq n$。*

证明. 由于Kripke模型中的状态是有穷的，因此存在状态集合S使得每个状态$s \in S$都在σ中无穷次出现，而且对于每个公式$f \in C$都存在S中的某个状态使得f成立。否则，如果对于任意状态集合S'中的状态在σ无穷次出现，都存在公式$f \in C$使得f在S'中的所有状态上都不成立，那么f一定在所有在σ无穷次出现的状态

上都不成立，因此f无法在σ中无穷次成立。假设$S = \{s_{i_1}, s_{i_2}, ..., s_{i_k}\}$，而且$i_1 \leq i_2 \leq ... \leq i_k$，那么令$s'_p = s_{i_1}$，$s'_n = s_{i_{k'}}$使得$i_{k'} \geq i_k$以及 $s_{i_{k'}} = s_{i_1}$. □

命题 1.9. 令C一个CTL_P公式的集合，而且$\sigma_f = s_0, s_1, ..., s_n$是一条有穷路径，其中对任意$0 \leq i \leq n-1$都有$s_i \rightarrow s_{i+1}$，而且存在$0 \leq p \leq n$使得$s_p = s_n$，而且对任意$C$中的公式，都存在$s_p$和$s_n$之间的某个状态使得该公式在该状态上成立。那么一定存在一条无穷路径$\sigma = s'_0, s'_1, ...$使得对任意$i \geq 0$都有$s'_i \rightarrow s'_{i+1}$，而且每个状态$s'_j$都在$s_0, s_1, ..., s_n$中出现，而且$C$中的每个公式都在该无穷路径上无穷次成立。

证明. 令$\sigma = s_0, ..., s_{p-1}, s_p, ..., s_{n-1}, ...$ 即可得证。 □

附录 A　SCTLProV的输入语言描述

A.1　词法标记与关键字

输入文件中包含的是有限个依次排列的字符，这些字符通过词法解析器的解析生成有限个词法标记。在词法标记中，

- 一个非负整数表示为有限个连续数字，而一个负整数则表示为在一个非负整数前添加符号“-”；

```
integer ::= [-]{0...9}+
```

- 一个非负浮点数表示为以符号“.”分隔的两个非负整数，其中当“.”后边的非负整数为0时可省略，而一个负浮点数则表示为在一个非负浮点数前添加符号“-”；

```
float ::= [-]{0...9}+.{0...9}*
```

- 一个标识符表示为以字母开头，并且有限个字母、数字、下划线以及符号“-”的组合，而且，在输入语言中区分以大写字母开头的标识符和以小写字母开头的标识符，不同标识符的详细用法可参考下面具体的语法描述。

```
iden    ::=  letter {letter|uletter|0...9|_|-}*
uiden   ::=  uletter {letter|uletter|0...9|_|-}*
letter  ::=  a...z
uletter ::=  A...Z
```

- 空白符和注释被用来增强程序的可读性，且在输入文件的语法分析阶段被忽略掉。其中，空格、制表符、回车符、换行符均为空白符。注释分为单行注释与多行注释，其中单行注释以“//”开头且以换行符结尾，而多行注释分为两种：以“/*”开头且以“*/”结尾，或以“(*”开头且以“*)”结尾。

除了以上语法标记外，输入语言中还包括一系列关键字。关键字是一种特殊的词法标记，也叫做保留字，用于在输入文件中表示特殊的含义。输入语言中的关键字如下：

Model	Var	Define	Init	Transition	Atomic	Fairness	Spec
int	bool	list	array	true	false	TRUE	FALSE
not	AX	EX	AF	AF	EG	AR	EU
datatype	value	function	let	match	with	if	then
else	(	)	[	]	{	}	=
!=	<	<=	>	>=	+	+.	-
-.	*	*.	/	/.	!	|	||
&&	->	:	::	,	;	.	

在输入文件中，关键字不能用来定义值或者函数的名字。关键字的用法在接下来的语法描述中有具体描述。

A.2 值

在输入语言中，值通常被用来定义Kripke结构的状态。一个值通常具有以下几种形式：

- 单位值：单位值`()`通常可看作一个空元组，是最简单的值。
- 布尔值：布尔值有两种：`true`和`false`。
- 数值：数值分为两种：整数值与浮点数值。数值的取值范围与OCaml编译器所能表示的相应值的取值范围一致，即：整数值的取值范围为-2^{30}到$2^{30}-1$（32位）或-2^{62}到$2^{62}-1$（64位）；浮点数值的取值范围符合IEEE 754[1]标准。
- 标量值：标量值也被称为枚举值，标量值通常由编程人员自定义且具有`#iden`形式，即“`#`”符号紧接着一个以小写字母为开头的标识符。
- 数组和列表：与OCaml语言类似，数组在这里具有`[|v1,...,vn|]`形式，其中`v1,...,vn`为$n \geq 0$个值；列表在这里具有`[v1,...,vn]`形式，其中`v1,...,vn`为$n \geq 0$个值。
- 元组：元组具有`v1,...,vn`，其中`v1,...,vn`为$n \geq 1$个值。
- 记录：记录可看作带标签的元组，记录具有`{l1 = v1 ; ... ; ln = vn ;}`形式，其中值`vi`的标签是`li`。

[1] https://en.wikipedia.org/wiki/IEEE_754

- 变体：变体由变体构造子及若干个值构造而成。变体分两种：只包含构造子的变体（构造子的元数为0）以及构造子和元组构造而成的变体（构造子的元数为元组中值的个数）。

值的语法描述如下：

```
value ::=
     ()
   | true | false
   | integer | float
   | #iden
   | [| value ; ... ; value |] | [ value ; ... ; value ]
   | ( value,...,value )
   | { iden = value ; ... ; iden = value ; }
   | uiden | uiden ( value , ... , value )
```

A.3 表达式与模式

Expressions are used to build complex values via evluations. The evaluations of expressions in our input language coincide with that of the corresponding OCaml expressions. Values can then be seen as the normal form of expressions, i.e., expressions that cannot be further evaluated.

Patterns are used to match different cases of expressions. The way that patterns match expressions in our input language coincide with that in OCaml.

```
expr ::=
      value                          (*value*)
    | iden                           (*variable or name of a value*)
    | uiden [(expr,...,expr)]        (*variant expression*)
    | expr . iden                    (*select one field of a record*)
    | expr [ expr ]                  (*select one field of a array*)
    | ! expr                         (*logical negation*)
    | expr && expr                   (*logical and*)
    | expr || expr                   (*logical or*)
    | - expr                         (*integer negation*)
```

```
    | expr + expr                  (*integer addition*)
    | expr - expr                  (*integer subtraction*)
    | expr * expr                  (*integer multiplication*)
    | -. expr                      (*float negation*)
    | expr +. expr                 (*float addition*)
    | expr -. expr                 (*float subtraction*)
    | expr *. expr                 (*float multiplication*)
    | expr = expr                  (*expression equivalence*)
    | expr != expr                 (*expression non-equivalence*)
    | expr < expr                  (*less than*)
    | expr <= expr                 (*less than or equal*)
    | expr > expr                  (*larger than*)
    | expr >= expr                 (*larger than or equal*)
    | ( expr )                     (*expression grouping*)
    | let pattern = expr           (*declare local variables*)
    | if expr then expr            (*if-then expression*)
    | if expr then expr else expr (*if-then-else expression *)
    | expr ; expr                  (*sequence of expressions*)
    | expr <- expr                 (*assignment*)
    | match_expr                   (*pattern matching*)
    | expr with {iden = expr ; ... ; iden = expr [;]}
                            (*a record with changed bindings*)
    | iden (expr , ..., expr)      (*a function call*)

match_expr ::= match expr with {| pattern -> expr}+

pattern ::=
      iden
    | constant
    | pattern :: pattern                    (*list*)
    | ( pattern ,..., pattern )             (*tuple*)
    | _                                     (*match any case*)
```

A.4 类型

附录 B　VMDV与定理证明工具的交互协议

VMDV与不同的定理证明工具的交互是通过互相发送与解析JSON形式的消息来实现的，每个JSON对象代表一个消息，消息的详细说明如下：

- **初始化显示窗口**（定理证明器 ⟶ VMDV）：在VMDV中，我们将每个显示窗口成为一个会话（Session）。当定理证明器需要可视化一个数据结构（比如证明树）的时候，定理证明器发送一个“create_session”消息，指定需要可视化的数据结构的名称（“session_id”）、简介（“session_descr”）及类型（“graph_type”）。其中，定理证明器指定的可视化的数据结构的类型分为两种：树（“Tree”）和有向图（“DiGraph”），前者通常用来表示证明树，后者通常用来表示可能有环的有向图。VMDV接收到该消息之后立即初始化并显示一个窗口，并等待定理证明器的进一步消息。

```
{
"type": "create_session",
"session_id": string,
"session_descr": string,
"graph_type": string
}
```

- **添加节点**（定理证明器 ⟶ VMDV）：向VMDV相应的会话中加入一个节点，并指定节点的标识（“id”）、要显示的字符串（“label”）、状态（“state”）。其中，节点的状态分为两种：已证明（“Proved”）和未证明（“Not_Proved”），不同状态的节点通常用不同的颜色高亮显示。

```
{
"type": "add_node",
"session_id": string,
"node":
{
"id": string,
"label": string,
```

```
"state": string
}
}
```

- **删除节点**（定理证明器 ⟶ VMDV）：在指定的会话中删除一个指定的节点。

```
{
"type": "remove_node",
"session_id": string,
"node_id": string
}
```

- **添加边**（定理证明器 ⟶ VMDV）：在指定的会话中添加一条由“from_id”节点指向“to_id”节点的边，同时指定该条边上要显示的字符串（“label”）。

```
{
"type": "add_edge",
"session_id": string,
"from_id": string,
"to_id": string,
"label": string
}
```

- **删除边**（定理证明器 ⟶ VMDV）：在指定的会话中删除由“from_id”节点指向“to_id”节点的边。

```
{
"type": "remove_node",
"session_id": string,
"node_id": string
}
```

- **高亮节点**（定理证明器 ⟷ VMDV）：高亮节点消息可由定理证明器和VMDV互相发送给对方。当定理证明器向VMDV发送该消息时，VMDV在相应的会

话中将指定的节点用不同于节点当前的颜色高亮显示。当用户在VMDV窗口中执行鼠标选中或者搜索操作时，被选中或者搜索到的节点在窗口中被高亮显示的同时，将所有节点的高亮消息发送到定理证明器。当定理证明器同时在VMDV中可视化多个数据结构时，高亮节点消息的传递过程可用来实现多个数据结构的交互显示。比如，在可视化SCTLProV的证明结果的时候，VMDV通常会初始化两个窗口，分别可视化当前公式的证明树，以及在搜索当前证明树的过程中所访问到的Kripke模型的状态。当用鼠标选中或搜索到证明树的某个或某些节点时，VMDV将这些证明树节点的高亮信息发送给SCTLProV，同时SCTLProV识别相应的证明树节点，并计算出与这些证明树节点相关的状态，然后将这些状态节点的高亮信息发送给VMDV，于是VMDV在状态图窗口中高亮显示相应的状态。

```
{
"type": "remove_node",
"session_id": string,
"node_id": string
}
```

- **取消高亮节点**（定理证明器 ⟷ VMDV）：同高亮节点消息一样，取消高亮节点的消息同样可由定理证明器和VMDV互相发送给对方。

```
{
"type": "unhighlight_node",
"session_id": string,
"node_id": string
}
```

- **取消所有高亮**（定理证明器 ⟷ VMDV）：去掉所有节点的高亮颜色。

```
{
"type": "clear_color",
"session_id": string
}
```

- **删除会话**（定理证明器 ⟶ VMDV）：VMDV收到删除会话消息后，销毁对应的窗口，结束一个数据结构的可视化过程。

```
{
"type": "remove_session",
"session_id": string
}
```

当定理证明器或VMDV收到对方发来的控制消息后，会发送回对方一个反馈消息，用来表明该控制消息是否被成功解析：

- **成功解析**（定理证明器 ⟷ VMDV）：

```
{
"type": "feedback",
"session_id": string,
"status": "OK"
}
```

- **解析失败**（定理证明器 ⟷ VMDV）：

```
{
"type": "feedback",
"session_id": string,
"status": "Fail",
"error_msg": string
}
```

参考文献

[1] CLARKE E M, GRUMBERG O, PELED D. Model checking[M]. Cambridge, MA, USA: MIT Press, 2001.

[2] BOUAJJANI A, JONSSON B, NILSSON M, et al. Regular model checking[C]//Proceedings of Computer Aided Verification, 12th International Conference, CAV 2000. Chicago, IL, USA: Springer-Verlag, Berlin, 2000: 403–418.

[3] BAIER C, KATOEN J. Principles of model checking[M]. USA: MIT Press, 2008.

[4] CIMATTI A, CLARKE E M, GIUNCHIGLIA F, et al. Nusmv: A new symbolic model verifier [C]//Proceedings of CAV'99. Trento, Italy: Springer-Verlag, Berlin, 1999: 495–499.

[5] ZHANG W. QBF Encoding of Temporal Properties and QBF-based Verification[C]// Proceedings of IJCAR 2014. Vienna: Springer-Verlag, Berlin, 2014: 224–239.

[6] HOLZMANN G J. The model checker SPIN[J]. IEEE Trans. Software Eng., 1997, 23(5): 279–295.

[7] GARAVEL H, LANG F, MATEESCU R, et al. CADP 2011: a toolbox for the construction and analysis of distributed processes[J]. STTT, 2013, 15(2): 89–107.

[8] MCMILLAN K L. Symbolic model checking[M]. USA: Springer, 1993.

[9] BRYANT R E. Graph-based algorithms for boolean function manipulation[J]. IEEE Trans. Computers, 1986, 35(8): 677–691.

[10] BIERE A, CIMATTI A, CLARKE E, et al. Symbolic model checking without BDDs[C]// CLEAVELAND W R. LNCS: volume 1579 Proceedings of TACAS'99. Amsterdam, the Netherlands: Springer, USA, 1999: 193–207.

[11] FITTING M. Graduate texts in computer science: First-order logic and automated theorem proving, second edition[M]. New York: Springer-Verlag, 1996.

[12] LOVELAND D W. Automated theorem proving: A logical basis (fundamental studies in computer science)[M]. Amsterdam: Elsevier, 1978.

[13] BUREL G. Automating theories in intuitionistic logic[C]//Proceedings of Frontiers of Combining Systems, 7th International Symposium, FroCoS 2009. Trento, Italy: Springer-Verlag, Berlin, 2009: 181–197.

[14] GORDON M. From LCF to HOL: a short history[C]//Proof, Language, and Interaction, Essays in Honour of Robin Milner. [S.l.: s.n.], 2000: 169–186.

[15] NIPKOW T. Interactive proof: Introduction to isabelle/hol[M]//Software Safety and Security - Tools for Analysis and Verification. [S.l.: s.n.], 2012: 254–285.

[16] OWRE S, RUSHBY J M, SHANKAR N. PVS: A prototype verification system[C]//Automated Deduction - CADE-11, 11th International Conference on Automated Deduction, Saratoga Springs, NY, USA, June 15-18, 1992, Proceedings. [S.l.: s.n.], 1992: 748–752.

[17] BERTOT Y, CASTÉRAN P. Texts in theoretical computer science. an EATCS series: Interactive theorem proving and program development - coq'art: The calculus of inductive constructions[M]. [S.l.]: Springer, 2004.

[18] ALLEN S F, CONSTABLE R L, EATON R, et al. The nuprl open logical environment[C]// Automated Deduction - CADE-17, 17th International Conference on Automated Deduction, Pittsburgh, PA, USA, June 17-20, 2000, Proceedings. [S.l.: s.n.], 2000: 170–176.

[19] KAUFMANN M, MOORE J S. An ACL2 tutorial[C]//Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings. [S.l.: s.n.], 2008: 17–21.

[20] PAULSON L C. Isabelle: The next 700 theorem provers[J]. CoRR, 1993, cs.LO/9301106.

[21] RAJAN S, SHANKAR N, SRIVAS M K. An integration of model checking with automated proof checking[C]//Computer Aided Verification, 7th International Conference, Liège, Belgium, July, 3-5, 1995, Proceedings. [S.l.: s.n.], 1995: 84–97.

[22] CAVALLI A R, DEL CERRO L F. A decision method for linear temporal logic[C]//7th International Conference on Automated Deduction, Napa, California, USA, May 14-16, 1984, Proceedings. [S.l.: s.n.], 1984: 113–127.

[23] VENKATESH G. A decision method for temporal logic based on resolution[C]//Foundations of Software Technology and Theoretical Computer Science, Fifth Conference, New Delhi, India, December 16-18, 1985, Proceedings. [S.l.: s.n.], 1985: 272–289.

[24] FISHER M, DIXON C, PEIM M. Clausal temporal resolution[J]. ACM Trans. Comput. Log., 2001, 2(1): 12–56.

[25] ZHANG L, HUSTADT U, DIXON C. A resolution calculus for the branching-time temporal logic CTL[J]. ACM Trans. Comput. Log., 2014, 15(1): 10:1–10:38.

[26] JI K. CTL Model Checking in Deduction Modulo[C]//Proceedings of Automated Deduction - CADE-25. Berlin: Springer International Publishing, Switzerland, 2015: 295–310.

[27] DOWEK G, HARDIN T, KIRCHNER C. Theorem proving modulo[J]. J. Autom. Reasoning, 2003, 31(1): 33–72.

[28] DOWEK G. Polarized resolution modulo[C]//Theoretical Computer Science - 6th IFIP TC 1/WG 2.2 International Conference, TCS 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings. [S.l.: s.n.], 2010: 182–196.

[29] EMERSON E A, CLARKE E M. Using branching time temporal logic to synthesize synchronization skeletons[J]. Sci. Comput. Program., 1982, 2(3): 241–266.

[30] EMERSON E A, HALPERN J Y. Decision procedures and expressiveness in the temporal logic of branching time[J]. J. Comput. Syst. Sci., 1985, 30(1): 1–24.

[31] CRAIG J J. Introduction to robotics - mechanics and control (2. ed.)[M]. USA: Prentice Hall, 1989.

[32] PARTOVI A, LIN H. Assume-guarantee cooperative satisfaction of multi-agent systems[C]//

Proceedings of American Control Conference, ACC 2014. USA: IEEE, USA, 2014: 2053–2058.

[33] APPEL A W. Compiling with continuations (corr. version)[M]. UK: Cambridge University Press, 2006.

[34] SESTOFT P. Undergraduate topics in computer science: volume 50 programming language concepts[M]. Switzerland: Springer International Publishing, 2012.

[35] DERSHOWITZ N. Termination of rewriting[J]. J. Symb. Comput., 1987, 3(1/2): 69–116.

[36] SARNAT J, SCHÜRMANN C. Lexicographic path induction[C]//Proceedings of Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009. Brasilia, Brazil: Springer, Berlin, 2009: 279–293.

[37] CAVADA R, CIMATTI A, DORIGATTI M, et al. The nuxmv symbolic model checker[C]// Proceedings of Computer Aided Verification - 26th International Conference, CAV 2014. Vienna, Austria: Springer International Publishing, Switzerland, 2014: 334–342.

[38] VERGAUWEN B, LEWI J. A linear local model checking algorithm for CTL[C]//Proceedings of CONCUR '93, 4th International Conference on Concurrency Theory. Hildesheim, Germany: Springer-Verlag, Berlin, 1993: 447–461.

[39] BHAT G, CLEAVELAND R, GRUMBERG O. Efficient on-the-fly model checking for ctl^* [C]//Proceedings of LICS'95. San Diego, California, USA: IEEE Computer Society, USA, 1995: 388–397.

[40] REYNOLDS J C. The discoveries of continuations[J]. Lisp and Symbolic Computation, 1993, 6(3-4): 233–248.

[41] PETERSON G L. Myths about the mutual exclusion problem[J]. Inf. Process. Lett., 1981, 12 (3): 115–116.

[42] MUÑOZ C A, DOWEK G, CARREÑO V. Modeling and verification of an air traffic concept of operations[C]//Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2004. Boston, Massachusetts, USA: ACM, USA, 2004: 175–182.

[43] NASA/TM-2004-213006. Abstract model of sats concept of operations: Initial results and recommendations[M]. USA: NASA, 2004.

[44] BYRNES J, BUCHANAN M, ERNST M, et al. Visualizing proof search for theorem prover development[J]. Electronic Notes in Theoretical Computer Science, 2009, 226: 23–38.

[45] LIBAL T, RIENER M, RUKHAIA M. Advanced proof viewing in prooftool[C]//Proceedings Eleventh Workshop on User Interfaces for Theorem Provers, UITP 2014, Vienna, Austria, 17th July 2014. [S.l.: s.n.], 2014: 35–47.

[46] SAKURAI K, ASAI K. Mikibeta: A general GUI library for visualizing proof trees[M]// Logic-Based Program Synthesis and Transformation. [S.l.]: Springer, 2011: 84–98.

[47] STEEL G. Visualising first-order proof search[C]//Proceedings of User Interfaces for Theorem Provers: volume 2005. [S.l.: s.n.], 2005: 179–189.

[48] FARMER W M, GRIGOROV O G. Panoptes: An exploration tool for formal proofs[J]. Electr. Notes Theor. Comput. Sci., 2009, 226: 39–48.

[49] BAJAJ C, KHANDELWAL S, MOORE J, et al. Interactive symbolic visualization of semi-automatic theorem proving[M]. [S.l.]: Computer Science Department, University of Texas at Austin, 2003.

[50] 张敏, 霍朝光, 霍帆帆, et al. 国际信息可视化知识族群：演化、聚类及迁徙研究[J]. 情报科学, 2016, 4: 13–17.

[51] 冯艳林. 城市道路交通信息可视化及其应用研究[J]. 黑龙江科技信息, 2015, 17: 180.

[52] MATHIEU J, TOMMASO V, SEBASTIEN H, et al. Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software[J]. PLOS ONE, 2014, 9(6): e98679.

[53] MARTIN S, BROWN W M, KLAVANS R, et al. Openord: an open-source toolbox for large graph layout[J]. SPIE Proceedings, 2011, 7868.

[54] HU Y. Efficient and high quality force-directed graph drawing[J]. The Matematical Journal, 2005, 10: 149–160.

[55] BIERE A, CIMATTI A, CLARKE E M, et al. Bounded Model Checking[J]. Advances in Computers, 2003, 58: 117–148.

[56] BOUAJJANI A, ESPARZA J, MALER O. Reachability analysis of pushdown automata: Application to model-checking[C]//Proceedings of CONCUR'97: Concurrency Theory. Warsaw, Poland: Springer-Verlag, Berlin, 1997: 135–150.

[57] GIRARD J Y. Proofs and types[M]. [S.l.]: Cambridge University Press, UK, 1989.

[58] KLEENE S C. Introduction to metamathematics[M]. [S.l.]: Ishi Press, California, 2009.

[59] BOLLIG B. On the OBDD complexity of the most significant bit of integer multiplication[J]. Theoretical Computer Science, 2011, 412(18): 1686 – 1695.

[60] GABBAY D M, PNUELI A. A sound and complete deductive system for ctl* verification[J]. Logic JOURNAL of the IGPL, 2008, 16(6): 499–536.

[61] REYNOLDS M. An axiomatization of full computation tree logic[J]. J. Symb. Log., 2001, 66 (3): 1011–1057.

[62] FRIEDMANN O. A proof system for ctl^*[D]. [S.l.]: University of Munich, 2008.

[63] HOLZMANN G J. The SPIN model checker - primer and reference manual[M]. [S.l.]: Addison-Wesley, 2004.

[64] BUREL G. Experimenting with deduction modulo[C]//SOFRONIE-STOKKERMANS V, BJØRNER N. Proceedings of CADE 2011. Wroclaw, Poland: Springer-Verlag, Berlin, 2011: 162–176.

[65] DAVIS M, PUTNAM H. A computing procedure for quantification theory[J]. J. ACM, 1960, 7(3): 201–215.

[66] PNUELI A, KESTEN Y. A deductive proof system for CTL[C]//Proceedings of CONCUR 2002. Brno, Czech Republic: Springer-Verlag, Berlin, 2002: 24–40.

[67] DOWEK G, JIANG Y. A logical approach to CTL[EB/OL]. 2013. http://www.lsv.ens-cachan.fr/~dowek/Publi/ctl.pdf.

[68] BIERE A, CIMATTI A, CLARKE E M, et al. Bounded model checking[J]. Advances in Computers, 2003, 58: 117–148.

[69] HE F, SONG X, HUNG W N N, et al. Integrating evolutionary computation with abstraction refinement for model checking[J]. IEEE Trans. Computers, 2010, 59(1): 116–126.

[70] CRESPI V, GALSTYAN A, LERMAN K. Top-down vs bottom-up methodologies in multi-agent system design[J]. Auton. Robots, 2008, 24(3): 303–313.

[71] LANGE M, STIRLING C. Model checking games for branching time logics[J]. J. Log. Comput., 2002, 12(4): 623–639.

[72] BRÜNNLER K, LANGE M. Cut-free sequent systems for temporal logic[J]. J. Log. Algebr. Program., 2008, 76(2): 216–225.

[73] DERSHOWITZ N, MANNA Z. Proving termination with multiset orderings[J]. Commun. ACM, 1979, 22(8): 465–476.

[74] JOUANNAUD J, LESCANNE P. On multiset orderings[J]. Inf. Process. Lett., 1982, 15(2): 57–63.

[75] DERSHOWITZ N. Orderings for term-rewriting systems[J]. Theor. Comput. Sci., 1982, 17: 279–301.

[76] TUFTE E R, GRAVES-MORRIS P. The visual display of quantitative information: volume 2 [M]. [S.l.]: Graphics press Cheshire, CT, 1983.

[77] DOWEK G, JIANG Y. Cut-elimination and the decidability of reachability in alternating pushdown systems[J/OL]. arXiv preprint arXiv:1410.8470, 2014. https://who.rocq.inria.fr/Gilles.Dowek/Publi/reachability.pdf.

[78] BOUAJJANI A, ESPARZA J, MALER O. Reachability analysis of pushdown automata: Application to model-checking[M]//CONCUR'97: Concurrency Theory. [S.l.]: Springer, 1997: 135–150.

[79] BERTOT Y, CASTÉRAN P. Interactive theorem proving and program development: Coq'art: The calculus of inductive constructions[M]. [S.l.]: Springer Science & Business Media, 2013.

[80] FREEMAN E, ROBSON E, BATES B, et al. Head first design patterns[M]. [S.l.]: "O'Reilly Media, Inc.", 2004.

[81] CLARKE E M, GRUMBERG O, PELED D. Model checking[M]. [S.l.: s.n.], 1999.

[82] BEDERSON B B, SHNEIDERMAN B. The craft of information visualization: readings and reflections[M]. [S.l.]: Morgan Kaufmann, 2003.

[83] TRAC S, PUZIS Y, SUTCLIFFE G. An interactive derivation viewer[J]. Electronic Notes in Theoretical Computer Science, 2007, 174(2): 109–123.

[84] CLARKE E M, GRUMBERG O, MCMILLAN K L, et al. Efficient generation of counterexamples and witnesses in symbolic model checking[C]//DAC. [S.l.: s.n.], 1995: 427–432.

[85] BAIER C, KATOEN J. Principles of model checking[M]. [S.l.]: MIT Press, 2008.

[86] MAZZA R. Introduction to information visualization[M]. [S.l.]: Springer, 2009.

[87] SPENCE R. Information visualization: Design for interaction (2nd edition)[M]. [S.l.]: Prentice Hall, 2007.

发表学术论文

学术论文

[1] Jian Liu, Ying Jiang, Yanyun Chen. VMDV: A 3D Visualization Tool for Modeling, Demonstration, and Verification. TASE 2017. accepted.

[2] Ying Jiang, Jian Liu, Gilles Dowek, Kailiang Ji. SCTL: Towards Combining Model Checking and Proof Checking. The Computer Journal. submitted.

项目资助情况

1. 中法合作项目VIP（项目编号GJHZ1844）

2. 中法合作项目LOCALI (项目编号NSFC 61161130530 和 ANR 11 IS02 002 01)

简 历

基本情况

刘坚，男，山东省茌平县人，1989 年出生，中国科学院软件研究所博士研究生。

教育背景

- 2011年9月至今：中国科学院软件研究所，计算机软件与理论，硕博连读
- 2007年9月至2011年7月：山东农业大学，信息与计算科学，本科

联系方式

通讯地址：北京市海淀区中关村南四街4号，中国科学院软件研究所，5号楼3层计算机科学国家重点实验室

邮编：100090

E-mail: liujian@ios.ac.cn

致 谢

值此论文完成之际，谨在此向多年来给予我关心和帮助的老师、学长、同学、朋友和家人表示衷心的感谢!