arXiv:2602.09307v1 [cs.LO] 10 Feb 2026

# 关于基于操作语义的程序的参数化动态逻辑理论

Y 张元瑞， 南京航空航天大学软件学院，中国 na

将动态逻辑应用于程序验证是一项挑战，因为其针对正则表达式的公理规则难以适配不同的程序模型。我们提出了一种称为 DLp 的新型动态逻辑，支持基于程序操作语义的推理。对于其迁移行为符合标准语义或自然语义的程序而言，DLp 使验证更加容易，因为无需像大多数其他动态逻辑那样重新设计并验证新的规则，而是可以直接应用程序转换进行推理。DLp 具有参数化特性，提供了一个与模型无关的框架，其中包含一组相对较小的推理规则；这些规则取决于给定的一组可靠的操作语义规则。DLp 的这些特性使多个模型能够在其框架中轻松进行比较，并使其与现有的动态逻辑理论兼容。DLp 支持循环推理，为递归程序提供增量式推导过程，使得无需预先进行程序变换即可更方便地开展推理。我们在一定条件下分析并证明了 DLp 的可靠性与完备性。若干案例研究展示了 DLp 的特性，并充分证明了其潜在的应用价值。

CCS Co ncepts: • **计算理论** → **模态逻辑与时序逻辑**;**霍尔逻辑**; 逻辑与验证; 证明*theo ry*.

附加关键词和短语：动态逻辑、程序演绎、验证框架、循环证明、操作语义、符号执行

**ACM参考文献格式：**
Yuanrui Zhang. 2018. 论面向基于操作语义的程序的参数化动态逻辑理论。载于会议论文集请务必根据您的版权确认邮件输入正确的会议标题（会议缩写'XX）。ACM，纽约，NY，美国，39 页。https://doi.org/XXXXXXX.XXXXXXX

## 1 引言

动态逻辑 [29] 已被证明是一种有价值的程序逻辑，可用于描述和推理不同类型的程序。作为一种将程序与公式整合为一体的"多模态"逻辑，它比传统的霍尔逻辑更具表达力 [31]（参见 [4]）。动态逻辑已成功应用于进程代数等领域 [8], 、编程语言 [7], 、同步系统 [62, 63], 、混合系统 [44, 45] 和概率系统 [35, 43]。这些理论推动了面向安全关键系统的相关验证工具的开发，例如 KIV [50], 、KeY [54], 、KeYmaera [46], 以及 [19, 63]中开发的工具。动态逻辑的近期研究提出了多种扩展，旨在应对现代问题，包括确保区块链协议的正确性 [32], 、形式化超属性 [25], 以及验证量子计算 [19, 58]。此外，动态逻辑作为一种很有前景的错误性推理框架也受到关注，近期研究对此进行了探索 [42, 64]。

作者联系信息： Yuanrui Zhang，yuanruizhang@nuaa.edu.cn，南京航空航天大学软件学院，江苏省南京市，中国。

允许为个人或课堂使用制作本作品全部或部分的数字或纸质副本，且无需付费，但前提是副本不得为营利或商业利益而制作或分发，并且副本须在首页载明本声明及完整引用信息。对于本作品中由作者以外的他人拥有的组成部分，其版权必须得到尊重。允许带署名的摘要。以其他方式复制、再版、在服务器上发布或向列表再分发，均须事先获得特别许可和/或支付费用。请向 permissions@acm.org 申请许可。© 2018 版权由所有者/作者持有。出版权已授权给 ACM。提交给 ACM 的稿件

大多数动态逻辑以及霍尔风格逻辑的理论，都是基于程序的指称语义构建的：程序的行为被解释为一个数学对象（例如轨迹集合），并构造一组推理规则来匹配该对象。将这些理论应用于其他程序可能十分困难。对于 Java、C语言和 Esterel 等编程语言而言尤其如此 [11], ，因为它们的语义非常复杂。为这些语言构建逻辑理论，需要针对其程序领域精心设计大量规则。此外，这些规则往往容易出错，因此需要验证其可靠性（甚至完备性），而这同样可能代价高昂。例如，在 KeY [54], 中，为了将一阶动态逻辑 [49]（FODL）应用于Java程序的验证，针对 Java 的原语提出了 500 多条推理规则（参见 [41]）。其正确性很难得到保证。另一个例子是，工具 Verifiable C [5] 使用近 40,000 行 Rocq 代码，基于分离逻辑定义并验证其针对 C语言的逻辑理论 [51]。

另一个主要问题是，要对某些类型的程序进行推理，必须先将它们转换为某种“标准形式”，以便应用合适的公理规则。这些预先进行的转换并无必要，而且代价可能很高。它们通常意味着破坏原有的程序结构，从而可能导致程序信息丢失。一个典型的例子是 Esterel [11] 或 Quartz [55]等命令式同步编程语言。文献 [24], 展示了同步程序必须如何转换为所谓的“STA程序”，才能对其应用恰当的 Hoare 逻辑规则。

与指称语义不同，结构操作语义 [47]描述程序在某种配置下如何迁移为另一个程序。它是 CCS [38]和 $\pi$-演算 [39]等并发模型的标准语义。对于上述可执行程序，给出操作语义十分直接，因为其执行过程旨在直接转换程序配置。因此，在大多数情况下，操作语义可以直接采信，无需额外验证（例如，参见 [14, 21]）。

**本文**提出了一种基于动态逻辑的理论，旨在简化对操作语义本质上属于操作式的程序进行推理。我们提出了一种所谓的参数化动态逻辑，简称为 DLp [1]。它支持直接基于操作语义进行推理。与以往专注于特定演算的工作（例如 [8, 9]）不同，我们的框架具有参数化特性，可适用于任意程序与公式。我们提出了 DLp 的证明系统，该系统基于循环证明方法（参见 [16]）。它提供了一组用于推导 DLp 公式的“核心规则”，并配有一组假定的程序迁移行为规则。我们在一般设定下研究并证明了 DLp 的可靠性与完备性。

与传统的基于动态逻辑的方法相比，DLp 有以下几项优势：(1) 对于那些易于获得且值得信赖的操作语义的程序，它减轻了目标模型适配以及对不可靠规则进行一致性验证的负担。与以往工作（如 [7]）相比，我们的“核心规则”集合非常精简。(2) 公式参数化提供了一个与模型无关的框架，不同的程序理论可以通过提升过程轻松嵌入其中，多个模型也可以轻松进行比较。(3) 它支持循环推理，可自然地处理递归程序导致的无限符号执行；此外，对于某些类型的“非标准”程序模型，无需预先进行程序变换，即可实现增量式推导。

此前大多数与我们相关的工作都在不同的数理逻辑或理论中探讨了这一问题（例如， [2, 17, 30, 36, 40, 52, 53, 56, 60], 详见第8节的详细比较）。除少数例外， [2, 30, 60], 其中大多数尚未

[1]为避免与著名的命题动态逻辑（PDL）发生冲突 [23]提交给 ACM 的稿件

关注用于推导动态逻辑公式的高效逻辑演算。据我们所知，DLp 是首个为不同程序的直接基于操作语义的推理提供循环验证框架的动态逻辑。

本文对先前的工作进行了非平凡的扩展 [61], ，并采用了截然不同的方法来构建动态逻辑理论p ，该理论基于克里普克结构，且不依赖显式签名。在这项工作中，我们进一步全面分析了动态逻辑的可靠性与完备性p, 并对动态逻辑的不同特性进行了更丰富的案例分析p。

## 2 概述

在动态逻辑中，动态公式形式如下： $[\alpha]\phi$ （参见 [29]），其中 $[\cdot]$ 是模态算子， $\alpha$ 是程序模型（或简称“程序”）， $\phi$ 是逻辑公式。直观地说，这意味着在程序 $\alpha$ 的所有执行过程终止后，公式 $\phi$ 成立。当 $\alpha$ 是确定性的，公式 $\phi \to [\alpha]\psi$ 恰好刻画了霍尔风格逻辑（例如 [31, 51]）中三元组 $\{\phi\}\alpha\{\psi\}$ 的部分正确性。 $\phi \to \alpha\psi$ 则刻画了 $\{\phi\}\alpha\{\psi\}$ 的完全正确性，其中 $\cdot$ 是 $[\cdot]$ 的对偶模态算子。$\alpha\psi$ 的定义使得 $\alpha\psi = \neg[\alpha]\neg\psi$ 表示存在 $\alpha$ 的某个执行过程满足：该过程终止，且终止后公式 $\psi$ 成立。动态公式结合了程序与公式，允许出现多重且嵌套的模态，例如 $[\alpha]\phi \to \beta\psi$、 $[\alpha]\beta\phi$、 $[\alpha](\phi \to \beta\psi)$ 等，因此其表达能力严格强于霍尔逻辑（参见 [4]）。

本节其余部分将概述 DLp 的主要工作，重点通过示例阐释核心思想。本文后续各节将对这些内容作详细介绍。

### 2.1 动态逻辑的标签化与参数化

为了通过程序的操作语义直接对程序进行推理，在 DLp中，我们引入“标签” $\sigma$（定义4.8），用来将显式程序结构表示为符号执行当前的程序配置。 $\sigma$ 将动态公式 $[\alpha]\phi$附加其上，得到如下形式的带标签公式 $\sigma : [\alpha]\phi$。直观地说，这表示在配置 $\sigma$下，程序 $\alpha$ 可以执行，并且在 $\phi$ 的所有终止执行过程之后，公式 $\alpha$成立。引入标签使我们能够通过以下程序转移推导带标签公式 $\sigma : [\alpha]\phi$ :

$$(\alpha, \sigma) \longrightarrow (\alpha', \sigma'),$$

使用概念上可表示为以下形式的推理规则:

$$\frac{\sigma' : [\alpha']\phi, \text{for all } (\alpha', \sigma') \text{ such that } (\alpha, \sigma) \longrightarrow (\alpha', \sigma')}{\sigma : [\alpha]\phi} \ ([\alpha]).$$

这些规则（对应于表（$[\alpha]R$）和（$[\alpha]L$）中的规则2）将 $\sigma : [\alpha]\phi$ 的演绎化简为所有后继公式 $\sigma' : [\alpha']\phi$ 的演绎；这些后继公式对应于单步程序转移。

由于程序转换具有通用形式，该框架适用于任意程序模型和配置。因此，在带标签公式 $\sigma : [\alpha]\phi$（属于 DLp）中，我们对程序 $\alpha$、逻辑公式 $\phi$ 和标签 $\sigma$进行参数化，以允许它们具有任意代数结构。 $\sigma : [\alpha]\phi$ 被证明是比 $[\alpha]\phi$ 更一般的形式。当 $\sigma$ 相对于 $[\alpha]\phi$ 为“free”（参见定义 5.8）时， $\sigma : [\alpha]\phi$ 与 $[\alpha]\phi$ 具有相同的含义。

考虑 FODL [49], 中的公式 $\phi_1 =_{df} (x \geq 0 \to [x := x + 1]x > 0)$，其中 $x$ 是一个取值范围为整数 $\mathbb{Z}$的变量。直观而言，公式 $\phi_1$ 表示：如果 $x \geq 0$ 成立，那么将表达式 $x + 1$ 赋值给 $x$后， $x > 0$ 成立。在 FODL 中，要推导 $\phi_1$，我们应用赋值规则：

$$\frac{\phi[e/x]}{[x := e]\phi} \ (x{:=}e)$$

针对部分 $[x := x+1]x > 0$。它将 $x > 0$ 中的 $x$ 替换为 $x+1$，得到表达式 $x+1 > 0$。推导之后，我们得到公式 $\phi_1' =_{df} (x \geq 0 \rightarrow x+1 > 0)$，它对任意 $x \in \mathbb{Z}$都成立。

另一方面，在 DLp 中，我们可以将 $\phi_1$ 表示为等价的带标签公式：$\psi_1 =_{df} (t \geq 0 \rightarrow \{x \mapsto t\}: [x := x+1]x > 0)$，其中标签 $\{x \mapsto t\}$ 表示变量 $x$ 存储了值 $t$（其中 $t$ 是一个新变量）。由于程序配置显式地呈现出来，为了推导公式 $\psi_1$，我们不再采用上述规则（$[\alpha]$），而是根据程序转换直接应用于 $\{x \mapsto t\}: [x := x+1]x > 0$ 部分

$$(x := x+1, \{x \mapsto t\}) \longrightarrow (\downarrow, \{x \mapsto t+1\}), \quad (op\ x := e)$$

该转换将值赋给 $t+1$，随后 $x$。$\downarrow$ 表示程序终止（参见定义4.1）。推导完成后，我们得到公式 $\psi_1' =_{df} (t \geq 0 \rightarrow \{x \mapsto t+1\}: x > 0)$，其中公式$\{x \mapsto t+1\}: x > 0$恰好表示 $t+1 > 0$，如果我们在公式 $x > 0$中用 $t+1$ 的当前值替换 $x$。因此，由 $\psi_1'$我们得到公式$t \geq 0 \rightarrow t+1 > 0$，它恰好就是公式 $\phi_1'$（忽略自由变量重命名）。

从这个例子可以看出，上述规则（$[\alpha]$）可以直接应用于其他语言（只需选择不同的程序转移集合），而规则（$op\ x := e$）则未必如此。例如，它不能应用于 Java 语句$x := new\ C(...)$, 该语句会创建一个 $C$ 类的新对象（参见 [7]）。在本文中（从例 4.4 到 5.2，以及第 6.1 节和第 6.3 节），我们通过 DLp 的两种实例化——DLp-WP 和 DLp-FODL——展示如何将 DLp 适配到不同的程序理论。第 6.3 节还展示了 DLp 在单一框架中推导多个程序模型的能力。

第 6.4节和附录 C语言 进一步给出两种实例化：DLp-PL 和 DLp-SP，分别用于展示在 DLp 中，不仅可以表达静态性质（即在某个状态下成立的性质），还可以表达更复杂的性质，如时序性质和空间性质。

标记与参数化的整个过程将在第 4 节中完整介绍。在第5.3 节中，我们构建了一个证明系统$Pr_{dlp}$ 用于DLp 。

### 2.2 DLp的提升过程与兼容性

作为一种扩展了额外结构标签的动态逻辑，DLp 与现有动态逻辑理论兼容，其含义是：每条针对不带标签动态公式的推理规则，都可以提升为适用于其在 DLp 中带标签对应形式的规则。第5.5 节详细讨论了这一技术。我们引入了“自由标签”这一概念（定义5.8），并证明给公式附加一个自由标签不会影响该公式的有效性（定理5.10）。

例如，根据上面的规则 $(op\ x := e)$，给每个公式附加标签 $\{x \mapsto t\}$，即可得到一条可靠的提升规则：

$$\frac{\{x \mapsto t\}: \phi[x/e]}{\{x \mapsto t\}: [x := e]\phi}\ (lf\ (x := e)).$$

$\{x \mapsto t\}$ 是自由的，因为 $t$ 是一个新变量。显然，在公式 $[x := e]\phi$ 和 $\phi[x/e]$ 中，将变量 $x$ 的任意自由出现替换为变量 $t$ 不会改变它们的含义。根据上面的公式 $\psi_1$，在 $\{x \mapsto t\}: [x := x+1]x > 0$ 这一部分应用规则 $(lf(x := e))$，即可得到公式 $\psi_1'' =_{df} t \geq 0 \rightarrow \{x \mapsto t\}: x+1 > 0$。如果将 $x+1 > 0$ 的 $x$替换为其当前值 $t$，这正是 $\psi_1'$我们上面看到的内容。

提升过程通过直接利用不同定义域中的特殊规则，提供了一种灵活性。在第6.2节中，我们将详细说明这种技术如何在推导过程中发挥作用。

### 2.3 DL𝔭 公式的循环推理

在通常的演绎过程中，我们通常期望得到一棵有限的证明树。然而，在证明系统 $Pr_{dlp}$ DL𝔭中，证明树的一个分支并不总是会终止，因为通过规则符号执行程序的过程($[\alpha]R$) 或/和规则 ($[\alpha]L$) 可能不会停止。当程序具有可能无限运行的显式/隐式循环结构时，这种情况众所周知。例如，在 DL 的实例化理论 DL𝔭-WP 中𝔭 (参见例 4.4- 5.2)，一个 while 程序

$$W =_{df} \ while\ true\ do\ x := x + 1\ end$$

无限地按以下程序转换继续进行：

$$(W, \{x \mapsto 0\}) \longrightarrow (W, \{x \mapsto 1\}) \longrightarrow ....$$

例如，在推导公式 $\{x \mapsto 1\} : [W]\phi$: 时，会得到以下无限推导分支：

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\{x \mapsto n\} : [W]\phi}}{\vdots}}{\{x \mapsto 2\} : [W]\phi}\ ([\alpha])}{\{x \mapsto 1\} : [W]\phi}\ ([\alpha])}{}.$$

为解决这个问题，我们提出了一个适用于 DL𝔭 （第 5.4 节）的循环证明系统。循环证明方法（参见 [16]）是一种允许某类无限演绎的技术，这类演绎称为“循环证明”（参见第 5.2 节）。循环证明是一棵有限证明树，并附加了一些不终止的叶节点，称为“芽”，这些芽与其某些祖先相同。将一个芽及其某个相同的祖先称为“回链”。

我们提出了一种专用于 DL𝔭 的循环推导方法，主要包含以下两个步骤。

第一步，我们通过识别合适的芽节点和回链来构造一个循环结构，其中最关键的工作是设计标签的代换规则 (*Sub*) (定义 4.12)。例如，对标签 (*Sub*) 应用第 6.1 节给出的规则 $\{x \mapsto 1\}$ 和 $\{x \mapsto t+1\}$，我们可以得到公式$\{x \mapsto 1\}$ 的一个循环推导： $[W]\phi$ 如下左图所示：

$$\dfrac{\dfrac{\dfrac{2 : \{x \mapsto t\} : [W]\phi}{\{x \mapsto t+1\} : [W]\phi\}}\ ([\alpha])}{1 : \{x \mapsto t\} : [W]\phi}\ (Sub)}{\{x \mapsto 1\} : [W]\phi}\ (Sub), \quad \dfrac{\dfrac{\dfrac{\{x \mapsto t\} : \langle W\rangle\phi}{\{x \mapsto t+1\} : \langle W\rangle\phi\}}\ ([\alpha])}{\{x \mapsto t\} : \langle W\rangle\phi}\ (Sub)}{\{x \mapsto 1\} : \langle W\rangle\phi}\ (Sub)$$

其中，节点 2 是一个芽节点，并通过回链指向节点 1。$t$ 是相对于 $x$、$W$ 和 $\phi$ 的新变量。标签 $\{x \mapsto 1\}$ 等于 $\{x \mapsto t\}[1/t]$（即将 $t$ 替换为 1 后得到的标签），而标签 $\{x \mapsto t+1\}$ 等于 $\{x \mapsto t\}[t+1/t]$（即将 $t$ 替换为表达式 $t+1$ 后得到的标签）。

然而，并非所有循环结构都是循环证明。考虑上图右侧公式的循环推导$\{x \mapsto 1\} : \langle W\rangle\phi$。根据模态$\langle\cdot\rangle$的语义（参见第4.1节），$\{x \mapsto 1\} : \langle W\rangle\phi$ 对于任意公式$\phi$ 都无效，因为$W$ 永不终止。因此，在第二步中，我们需要检查这些循环结构是否合法构成循环证明，其中关键步骤是为系统 $Pr_{dlp}$定义合适的专用“渐进推导轨迹”（定义5.5）。

我们分别在第 6.1 节和附录 B 中给出两个示例，说明如何在 DL𝔭 中进行循环推理，以及如何通过循环图，利用对递归程序进行增量推理。特别是，

附录 B 中的示例是一个Esterel程序，由此可以看出，基于操作语义的循环推理如何避免在同步语言中预先进行额外的程序变换。

### 2.4 DLp的可靠性与完备性

我们分析并证明，在某些限制条件（第p 节）下，DL7对于任意程序与公式的可靠性与完备性。

DLp 的可靠性表明，循环证明总能导出有效的结论。在第7.1节中，我们在一个限制程序如何终止的条件（定义7.2）下证明了这一点。尽管受限程序的类型仍然非常丰富，足以涵盖所有确定性编程语言（参见第7.1节）。

证明可靠性的思路是反证法（参见 [15]）。我们假设结论（例如$\{x \mapsto 1\} : [W]\phi$）无效，那么在某个证明分支中就会导出一串无效公式，例如上面推导中的$\{x \mapsto 1\} : [W]\phi$、$\{x \mapsto t\} : [W]\phi$、$\{x \mapsto t+1\} : [W]\phi$，……。这串无效公式会导致违反一个良基集（定义7.4），该集合由一类用于关联这些无效公式的度量（定义7.8）构成。更多细节见第7.1节。

DLp 的完备性指出，对于任意有效的带标签动态公式，都存在一个针对它的循环证明。我们在所谓的“循环程序”（定义7.13、7.14）满足一个充分假设的条件下，证明了 DLp 的完备性。证明的主要思想与细节见第7.2 节和附录A。这一完备性结果很有用，因为限制条件具有一般性：只要 DLp 的程序模型满足此条件，其任意实例化都是完备的。

### 2.5 主要贡献与内容结构

本文的主要贡献可概括如下：

- 我们定义了 DLp 公式的语法与语义。
- 我们构造了一个带标签的证明系统，并为 DLp开发了提升过程。
- 我们提出了一种专为 DLp设计的循环证明方法。
- 我们分析并证明了 DLp 在某些条件下的可靠性与完备性。

本文其余部分安排如下。第 3 节简要介绍理解主要内容所必需的 PDL 和 FODL。第4节定义 DLp的语法和语义。第 5节提出 DLp的循环证明系统。第 6节分析一些案例研究。第 7 节分析 DLp的可靠性与完备性。第 8 节介绍相关工作，而第9节总结全文并讨论未来工作。

## 3 预备知识：PDL 与 FODL

在命题动态逻辑（PDL）中， [23], 公式的语法 $\phi$ 通过对程序与公式进行同时归纳给出，巴科斯-诺尔范式如下：

$$\alpha =_{df} a \mid \phi? \mid \alpha;\alpha \mid \alpha \cup \alpha \mid \alpha^*,$$

$$\phi =_{df} p \mid \neg\phi \mid \phi \wedge \phi \mid [\alpha]\phi.$$

在上述定义中， $\alpha$ 是带测试的正则表达式，通常称为正则程序。 $a \in A$ 是符号集合 $A$中的原子动作。 $\phi?$ 是测试。如果 $\phi$ 为真，则程序继续执行；否则，程序停止； $\alpha;\beta$ 是顺序程序，表示程序 $\alpha$ 终止后， $\beta$ 继续执行。 $\alpha \cup \beta$ 是选择程序，表示 $\alpha$ 或 $\beta$以非确定方式继续执行。提交给 ACM 的稿件

$\alpha^*$ 是星号程序，表示 $\alpha$ 可以执行任意 $n \geq 0$ 次。 $p$ 是原子公式，包括布尔值真。我们将包含模态 $[\cdot]$ 的公式称为动态公式。直观而言，公式 $[\alpha]\phi$ 表示，在程序 $\alpha$的所有执行过程之后，公式 $\phi$ 均成立。

PDL 的语义基于克里普克结构（参见 [29]） $M = (S, \rightarrow, I)$，其中 $S$是一个世界集合；$\rightarrow \subseteq S \times A \times S$是一个由原子程序标记的迁移集合； $I: P \rightarrow \mathcal{P}(S)$ 将集合中的每个原子 PDL 公式解释为 $P$ 一个世界集合。

给定一个克里普克结构 $M$，PDL 的语义基于正则程序的指称语义 $[|\cdot|]$，表示为世界 $w$与 PDL 公式 $\phi$ 之间的满足关系 $M, w \models \phi$。其定义如下，通过对程序与公式进行同时归纳:

a. $[|a|] =_{df} \{(w, w') \mid w \xrightarrow{a} w' 在 M\}$; b. $[|\phi?|] =_{df} \{(w, w) \mid M, w \models \phi\}$; c. $[|\alpha; \beta|] =_{df} \{(w, w') \mid \exists w''.(w, w'') \in [|\alpha|] \wedge (w'', w') \in [|\beta|]\}$; d. $[|\alpha \cup \beta|] =_{df} [|\alpha|] \cup [|\beta|]$; e. $[|\alpha^*|] =_{df} \bigcup_{n=0}^{\infty} [|\alpha^n|]$, 其中 $\alpha^0 =_{df}$真? , $\alpha^n =_{df} \alpha$; $\alpha^{n-1}$对任意 $n \geq 1$。1. $M, w \models p$，若 $w \in I(p)$; 2. $M, w \models \neg\phi$，若 $M, w \not\models \phi$; 3. $M, w \models \phi \wedge \psi$，若 $M, w \models \phi$且 $M, w \models \psi$; 4. $M, w \models [\alpha]\phi$，若对所有 $(w, w') \in [|\alpha|]$, $M, w' \models \phi$。

PDL 研究对所有克里普克结构都有效的公式。其证明系统是完备的（参见 [29]）。

一阶动态逻辑（FODL） [49] 是通过在某些特殊论域中专门化 PDL 的原子动作 $a$ 和原子公式$p$ 而从 PDL 得到的。在 FODL 中，原子动作是形如 $x := e$ 的赋值，其中 $x \in \mathrm{Var}^{fodl}$是一个变量， $e$是一个表达式。通常，我们将 $e$ 视为整数域 $\mathbb{Z}$上的算术表达式，例如 $x + 5$和 $x - 2 * y$，其中 $x$、 $y \in \mathrm{Var}^{fodl}$。 $+$、 $-$、 $*$和 / 是常见的算术运算符。原子公式是一个算术关系 $e_1 \bowtie e_2$，其中 $\bowtie \in$属于$=$、 $<$、 $\leq$、 $>$、 $\geq$，例如 $x + 5 < 0$和 $x - 2 * y = 1$。因此，FODL 中的非动态公式就是由逻辑联结词 $\neg$、 $\wedge$和量词 $\forall$连接起来的常见算术一阶公式。

在 FODL 中，状态 $w : \mathrm{Var}_{fodl} \rightarrow \mathbb{Z}$ 将每个变量映射到一个整数。对于表达式 $e$， $w(e)$ 返回这样一个值: 将每个变量 $x$ 在 $e$ 中的所有自由出现替换为值 $w(x)$ 后所得的值。FODL 的克里普克结构$M_{fodl} = (S_{fodl}, \rightarrow_{fodl}, I_{fodl})$ 定义如下： $S_{fodl}$ 是关于 $\mathrm{Var}^{fodl}$ 和 $\mathbb{Z}$ 的所有状态构成的集合。对于每个赋值$x := e$， $w \xrightarrow{x:=e} w'$ 是 $M_{fodl}$ 上的一个关系，当且仅当 $w' = w[x \mapsto e]$，其中 $w[x \mapsto e]$ 返回一个状态，该状态将 $x$ 映射到值 $w(e)$，并将其他变量映射到与 $w$相同的值。 $I_{fodl}$ 将每个原子FODL公式解释为其满足状态所构成的集合。或者，形式化地说，对于状态 $w \in I_{fodl}(e_1 \bowtie e_2)$， $w(e_1) \bowtie w(e_2)$ 为真。基于此，可以用与上文类似的方式定义 FODL 的语义。FODL 的形式化定义可参见 [29] 。

FODL 构成了许多现有动态逻辑理论的语言基础 [7, 8, 22, 35, 43–45, 62, 63] 如第1节所述。有些理论通过添加新的原语扩展 FODL，另一些则可以用 FODL 表示。例如，对于传统的 while 程序 $\alpha$:

$$\alpha =_{df} x := e \mid \alpha; \alpha \mid \mathit{if}\ \phi\ \mathit{then}\ \alpha\ \mathit{else}\ \beta\ \mathit{end} \mid \mathit{while}\ \phi\ \mathit{do}\ \alpha\ \mathit{end},$$

它们的特殊语句可以用 FODL 表示如下（参见 [29]）：

$$\mathit{if}\ \phi\ \mathit{then}\ \alpha\ \mathit{else}\ \beta\ \mathit{end} =_{df} \phi?; \alpha \cup \neg\phi?; \beta,$$

$$\mathit{while}\ \phi\ \mathit{do}\ \alpha\ \mathit{end} =_{df} (\phi?; \alpha)^*; \neg\phi?.$$

## 4 动态逻辑 DLp

### 4.1 DLp 公式的语法和语义

DL 理论p 通过允许程序 $\alpha$ 和公式 $\phi$ 在模态中 $[\alpha]\phi$ 采用任意形式而扩展了 PDL；不过，在第7节讨论其可靠性与完备性时，仍须满足某些限制条件。

在本节定义的关系中，我们用 · 表示一个被忽略的对象，其内容并不重要。例如，我们可以写作 $w \to \cdot$、$w \overset{\alpha}{/\cdot\longrightarrow} w'$、$(\alpha, \sigma) \longrightarrow (\alpha', \cdot)$ 等。

定义 *4.1*（程序与公式）。在 DLp 中，我们假设有两个预先定义且互不相交的集合 **P** 和**F**。**P** 是一个程序集合，其中有一个特殊程序 $\downarrow \in \mathbf{P}$，称为“终止程序”。**F** 是一个公式集合。

定义*4.2*（*DL*p 公式）。关于参数**P** 和**F**的动态逻辑公式 $\phi$（称为“参数化动态逻辑”（DLp）公式）按巴科斯-诺尔范式定义如下：

$$\phi =_{df} F \mid \neg\phi \mid \phi \wedge \phi \mid [\alpha]\phi,$$

其中 $F \in \mathbf{F}, \alpha \in \mathbf{P}$； $[\cdot]$ 是一个新算子，不会出现在**F**的任何公式中。

我们将 DLp 公式的集合记为 $\mathfrak{F}_{dlp}$。

若 DLp 公式中含有模态 $[\cdot]$，则称其为动态公式 。直观而言，公式 $[\alpha]\phi$ 表示程序 $\alpha$的所有执行过程终止后，公式 $\phi$ 成立。· 是 $[\cdot]$的对偶算子。公式 $\alpha\phi$可表示为 $\neg[\alpha]\neg\phi$。其他带有逻辑联结词（如 $\vee$ 和 $\to$）的公式，也可分别用含有 $\neg$和 $\wedge$ 的公式表示。

遵循定义动态逻辑的惯例（参见 [29]），我们引入一种新颖的克里普克结构，以刻画**P**中的参数化程序行为。

定义 *4.3*（程序标记*Kripke*结构）。关于参数**P**和**F** 的“程序标记”克里普克（PLK）结构是一个三元组

$$K(\mathbf{P}, \mathbf{F}) =_{df} (\mathcal{S}, \longrightarrow, \mathcal{I}),$$

其中，S 是世界的集合； $\longrightarrow \subseteq \mathrm{S} \times (\mathbf{P} \times \mathbf{P}) \times \mathrm{S}$ 是由程序对标记的关系集合，其形式为 $w_1 \overset{\alpha}{/\alpha'\longrightarrow} w_2$，其中 $w_1, w_2 \in S, \alpha, \alpha' \in \mathbf{P}$； $\mathrm{I} : \mathbf{F} \to \mathrm{P}(\mathrm{S})$ 是定义在世界幂集上的公式 **F**的解释。此外，$K(\mathbf{P}, \mathbf{F})$ 满足：对于任意 $w \in \mathrm{S}$ 和 $\alpha \in \mathbf{P}$， $w \overset{\downarrow}{/\longrightarrow} \cdot$。

定义 4.3 与 PDL 的克里普克结构 $M$（第 3 节）有以下区别：（1）它引入了如下形式的程序标记关系：$w_1 \overset{\alpha}{/\alpha'\longrightarrow} w_2$；（2）它为终止程序 $\downarrow$引入了一个附加条件。程序标记关系描述程序的迁移行为，这通常由其操作语义刻画（如第 5.3 节所示）。这不同于 $M$中的关系，在那里，一个关系只刻画原子程序的行为。直观地说， $w_1 \overset{\alpha}{/\alpha'\longrightarrow} w_2$ 表示从世界 $w_1$出发，程序 $\alpha$ 迁移为程序 $\alpha'$，并最终到达世界 $w_2$。 $\downarrow$ 的条件准确刻画了程序终止的含义。

本文以下的讨论始终基于一个假定的 PLK结构，即 K(P, F) = (S $\longrightarrow$, I)。从下文的例 4.4 开始，经由例 4.9、4.10 和 5.2，我们逐步将理论 DLp 具体化到特殊理论 FODL 的框架中，并将 FODL 的程序模型限制为一种更简单的程序模型——while 程序。为此，我们明确给出参数 P、F、L、M 以及 DLp 中操作语义的证明系统 $\mathrm{Pr}^{op}$的定义（L、M 和 $\mathrm{Pr}^{op}$ 将在下文介绍）。具体化后的逻辑理论称为 DLp-WP。

例 *4.4*（程序与公式的实例化）。考虑将 **P** 实例化为第 3 节中定义的 while 程序集合$\mathbf{P}_W$。考虑一个程序 $WP$，它属于 $\mathbf{P}_{\mathbb{W}}$

$$WP =_{df} \{while\ (n > 0)\ do\ s := s + n\ ;\ n := n - 1\ end\}.$$

给定变量 $n$和 $s$的初始值，程序 $WP$计算从 $n$到 1 的总和，并将其存储在变量 $s$中。while 程序的 PLK结构 $\mathbf{K}_W = (\mathbf{S}_W, \longrightarrow_W, \mathbf{I}_W)$ 满足 $\mathbf{S}_W = S_{fodl}$和 $\mathbf{I}_W = I_{fodl}$。 $\longrightarrow_W$描述 while 程序的迁移行为，这些行为由 $\mathbf{P}_W$的操作语义捕获（见第5节中的表1）。 $\longrightarrow_W$与原子程序上的 $\rightarrow_{fodl}$一致。例如，关系 $w \xrightarrow{x := x+1} w[x \mapsto w(x) + 1]$ 在 $\mathbf{K}_W$上，当且仅当关系$w \xrightarrow{x := x+1} w[x \mapsto w(x) + 1]$ 在 $M_{fodl}$上。

我们将 **F** 实例化为整数域上的算术一阶公式 $\mathbb{Z}$（第 3节），即 $\mathbf{F}_{afo}$。

定义*4.5*（执行路径）。K 上的“执行路径”是 $\longrightarrow: w_1 \xrightarrow{\alpha_1} \xrightarrow{\beta_1} \ldots \xrightarrow{\alpha_n} \xrightarrow{\beta_n} w_{n+1}\ (n \geq 0)$ 上的一个有限关系序列，满足 $\beta_n \in \{\downarrow\}$，且对所有 $1 \leq i < n$， $\beta_i = \alpha_{i+1} \notin \{\downarrow\}$。

在定义 4.5中，执行路径有时简写为一个世界序列： $w_1 \ldots w_{n+1}$。当 $n = 0$ 时，执行路径是一个单独的世界 $w_1$（不包含 $\longrightarrow$ 上的任何关系）。

给定一条路径 $tr$，我们常用 $tr_b$和 $tr_e$表示其首元素和末元素（若存在）。对于两条路径 $tr_1 =_{df} w_1 \ldots w_n$ 和 $tr_2 =_{df} w'_1 w'_2 \ldots w'_m \ldots$（$n$， $m \geq 0$）， $tr_1$是有限的。拼接 $tr_1 \cdot tr_2$定义为路径： $w_1 \ldots w_n w'_2 \ldots w'_m \ldots$，如果 $w_n = w'_1$成立。我们用关系 $tr_1 \preceq_s tr_2$表示 $tr_1$是 $tr_2$的后缀。若 $tr_1$是 $tr_2$的真后缀，则记作 $tr_1 \prec_s tr_2$。

定义 *4.6*（*DL*$\mathfrak{p}$ 公式的语义）。给定一个 DL$\mathfrak{p}$ 公式 $\phi$，世界 $w \in \mathbf{S}$ 在 K 下对 $\phi$的满足关系，记作 K $w \models \phi$，归纳定义如下：

1. K $w \models F$，其中 $F \in \mathbf{F}$，若 $w \in \mathbf{I}(F)$；2. K $w \models \neg\phi$，若 K $w \not\models \phi$；3. K $w \models \phi \wedge \psi$，若 K $w \models \phi$且 K $w \models \psi$；4. K $w \models [\alpha]\phi$，若对于所有形式如下的执行路径： $w \xrightarrow{\alpha} \cdot \longrightarrow \ldots \cdot \xrightarrow{\downarrow} w'$，对于某个 $w' \in \mathbf{S}$， K $w' \models \phi$。

根据算子 $\cdot$ 的定义，其语义定义为：若存在形式为 $w \xrightarrow{\alpha} \cdot \longrightarrow \ldots \cdot \xrightarrow{\downarrow} w'$ 的执行路径，对于某个 $w' \in \mathbf{S}$，满足 K $w' \models \phi$，则 K $w \models \alpha\phi$。

如果 K $\models \phi$（或简写为 $\models \phi$），则称 DL$\mathfrak{p}$ 公式 $\phi$ 关于 K 是有效的，当且仅当 K $w \models \phi$对于所有 $w \in \mathbf{S}$。

与 PDL 的语义（第 3 节）相比，要刻画正则程序的语义，只需记录起始世界和终止世界；而我们必须记录从起始节点到终止节点的整个执行路径。这是因为 DL$\mathfrak{p}$ 中程序的语义是操作式的，而不是根据其语法结构定义的指称式语义。

例 *4.7*（*DL*$\mathfrak{p}$ 规范）。程序 $WP$（例 4.4）的一个性质可描述为以下公式

$$(n \geq 0 \wedge n = N \wedge s = 0) \rightarrow [WP](s = ((N + 1)N)/2),$$

这意味着给定初始条件 $n$和 $s$，在执行$WP$之后， $s$等于$((N + 1)N)/2$，即 $1 + 2 + \ldots + N$的和，其中 $N$是 $\mathbb{Z}$中的自由变量。我们在 DL$\mathfrak{p}$ 的第6.1节中证明了该公式的一个等价带标签版本。

### 4.2 LabeledDLp 公式

定义 *4.8*（标签与标签映射）。在 DLp 中，我们假定两个预定义的集合 **L** 和 **M**。**L** 是“标签”的集合。$\mathbf{M} \subseteq \mathbf{L} \to \mathrm{S}$ 是标签映射的集合。每个映射 $\mathfrak{m} \in \mathbf{M}$将 **L** 的一个标签映射到集合 S 的一个世界。

标签通常表示捕获程序配置的显式数据结构，例如存储、堆、替换等。标签映射将标签与世界关联起来，充当标签的语义函数。

例 *4.9*（标签的实例化）。对于 while 程序，我们考虑一种标签，称为 $\mathbf{L}^{W}$，它捕获以下形式的程序配置的含义：

$$\{x_1 \mapsto e_1, ..., x_n \mapsto e_n\}\ (n \geq 0)$$

其中，每个变量 $x_i \in Var^W$ 存储一个算术表达式 $e_i\,(1 \leq i \leq n)$的唯一值。为简化起见，我们限制变量 $x_1, ...x_n$ 必须出现在所讨论的程序中，并且 $e_1, ..., e_n$ 中的任何自由变量都不能是$x_1, ..., x_n$中的任何一个。对于任意表达式 $e$，$\sigma(e)$ 都会返回一个表达式：将 $e$ 中的每个自由变量 $x_i$ 替换为其在 $\sigma$ 中对应的表达式 $e_i$。“配置更新”$\sigma_e^x$ 返回一个配置，其中变量 $x$ 存储表达式 $\sigma(e)$ 的值，而其他变量仍存储与 $\sigma$相同的值。

例如，在程序 $WP$(例 4.4) 中，$\{n \mapsto N, s \mapsto 0\}$ 可以是一个将 $n$ 映射到值 $N$(作为自由变量)，并将 $s$ 映射到 0的配置。

例 *4.10*（标签映射的实例化）。对于 while 程序，我们考虑一个由标签映射构成的集合 $\mathbf{M}^{W}$。$\mathbf{M}_W \subseteq \mathbf{L}_W \to \mathrm{S}^W$ $\mathbf{M}^W$中的每个标签映射都与一个世界相关联，记为 $\mathfrak{m}_w$，其中 $w \in \mathrm{S}^W$。给定一个刻画 $\{x_1 \mapsto e_1, ..., x_n \mapsto e_n\}$（$n \geq 1$）含义的配置 $\sigma$，$\mathfrak{m}_w(\sigma)$ 被定义为满足以下条件的世界：

(1) $\mathfrak{m}_w(\sigma)(x_i) = w(e_i)$，对于每个 $x_i (1 \leq i \leq n)$;
(2) 对于其他变量 $y \in Var^W$，$\mathfrak{m}_w(\sigma)(y) = w(y)$。

如第3节所述，$w(e_i)$ 通过将变量 $x$ 在 $e_i$ 中的每个自由出现替换为值 $w(x)$，返回一个值。

例如，设 $w$ 是一个世界，且 $w(N) = 5$，那么对于任意其他变量 $y \notin \{n, s\}$，我们有 $\mathfrak{m}_w(\{n \mapsto N, s \mapsto 0\})(n) = w(N) = 5$、$\mathfrak{m}_w(\{n \mapsto N, s \mapsto 0\})(s) = 0$，以及 $\mathfrak{m}_w(\{n \mapsto N, s \mapsto 0\})(y) = w(y)$。

定义 *4.11*（带标签的 *DL*p 公式）。DLp 中的“带标签公式”属于以下定义的公式类型之一：

$$\phi =_{df} \sigma : \psi \mid (\alpha, \sigma) \longrightarrow (\alpha', \sigma') \mid \sigma \Downarrow \alpha,$$

其中 $\sigma, \sigma' \in \mathbf{L}$、$\alpha, \alpha' \in \mathbf{P}$、$\psi \in \mathfrak{F}_{dlp}$。

我们用 $\mathfrak{F}_{ldlp}$、$\mathfrak{F}_{pt}$ 和 $\mathfrak{F}_{ter}$ 分别表示以下形式的带标签公式集合：$\sigma : \psi$、$(\alpha, \sigma) \longrightarrow (\alpha', \sigma')$ 以及 $\sigma \Downarrow \alpha$。我们通常用 $\tau$ 表示 $\mathfrak{F}_{ldlp} \cup \mathfrak{F}_{pt} \cup \mathfrak{F}_{ter}$ 中的一个带标签公式。

在 DLp 中，关系 $(\alpha, \sigma) \longrightarrow (\alpha', \sigma')$ 称为程序转换，它表示从一个所谓的程序状态 $(\alpha, \sigma)$ 执行到另一个程序状态 $(\alpha', \sigma')$。关系 $\sigma \Downarrow \alpha$ 称为程序终止，它描述程序 $\alpha$ 在标签 $\sigma$下的终止。

定义 *4.12*（标签替换）。“替换” $\eta : \mathbf{L} \to \mathbf{L}$ 是定义在**L** 上的一个函数，满足：对于任意标签映射 $\mathfrak{m} \in \mathbf{M}$，都存在一个标签映射 $\mathfrak{m}'(\mathfrak{m}, \eta)$（仅由 $\mathfrak{m}$ 和 $\eta$决定），使得对于所有标签 $\sigma \in \mathbf{L}$，$\mathfrak{m}'(\sigma) = \mathfrak{m}(\eta(\sigma))$。

定义4.12 用于规则(*Sub*)（表2）以及规则的可靠性证明 $Pr_{ldlp}$ 和 DL 的循环证明系统$\mathfrak{p}$。

定义 *4.13*（带标签 *DL*$\mathfrak{p}$ 公式的语义）。给定一个标签映射 $\mathfrak{m} \in \mathbf{M}$ 和一个带标签公式 $\tau \in \mathfrak{F}_{ldlp} \cup \mathfrak{F}_{pt} \cup \mathfrak{F}_{ter}$，公式 $\tau$关于 K、M 和 $\mathfrak{m}$（简写为 $\mathfrak{m} \models \tau$）的满足关系 $\mathrm{K},\mathbf{M}\ \mathfrak{m} \models \tau$，根据 $\tau$的不同情况定义如下：

1. $\mathrm{K},\mathbf{M}\ \mathfrak{m} \models \sigma$: $\phi$，若 $\mathrm{K}\ \mathfrak{m}(\sigma) \models \phi$；2. $\mathrm{K},\mathbf{M}\ \mathfrak{m} \models (\alpha,\sigma) \longrightarrow (\alpha',\sigma')$，若 $\mathfrak{m}(\sigma) \xrightarrow{\alpha/\alpha'} \mathfrak{m}(\sigma')$ 是 K 上的一个关系；3. $\mathrm{K},\mathbf{M}\ \mathfrak{m} \models \sigma \Downarrow \alpha$，若存在一条执行路径 $\mathfrak{m}(\sigma) \xrightarrow{\alpha} \ldots \xrightarrow{/\cdot\cdot/} \downarrow w$，其位于 K 上，且对应某个世界 $w \in \mathrm{S}$。

公式 $\tau \in \mathfrak{F}_{ldlp} \cup \mathfrak{F}_{pt} \cup \mathfrak{F}_{ter}$有效，记作 $\mathrm{K} \models \tau$（或简记为 $\models \tau$），如果对于所有 $\mathfrak{m} \in \mathbf{M}$，$\mathrm{K},\mathbf{M}\ \mathfrak{m} \models \tau$。

## 5 DL$\mathfrak{p}$ 的循环证明系统

我们为 DL$\mathfrak{p}$提出一个循环证明系统。首先，我们提出一个带标签的证明系统 $Pr_{dlp}$，以支持基于操作语义的推理（第5.3节）。然后，我们为系统 $Pr_{dlp}$构造一个循环证明结构，它支持在某些条件下导出无限证明树（第5.4节）。第5.1节和5.2 节分别介绍带标签序列演算和循环证明的概念。

### 5.1 带标签序列演算

序列是一种逻辑论证，形式为：$\Gamma \Rightarrow \Delta$，其中 $\Gamma$ 和 $\Delta$ 是有限的公式多重集合，分别称为序列的左侧和右侧。当 $\Gamma$ 或 $\Delta$ 为空集合时，我们用点号 $\cdot$ 表示它们。直观地说，序列 $\Gamma \Rightarrow \Delta$ 表示：如果 $\Gamma$ 中的所有公式都成立，那么 $\Delta$ 中至少有一个公式成立。我们用 $\nu$ 表示一个序列。

带标签序列是 $\mathfrak{F}_{ldlp} \cup \mathfrak{F}_{pt} \cup \mathfrak{F}_{ter}$ 中每个公式都是带标签公式的序列。

根据上面对序列的定义，若对于每个 $\mathfrak{m} \in \mathbf{M}$，$\mathfrak{m} \models \tau$对所有$\tau \in \Gamma$都蕴含 $\mathfrak{m} \models \tau'$对某个 $\tau' \in \Delta$，则带标签序列 $\Gamma \Rightarrow \Delta$ 是有效的。对于公式的多重集 $\Gamma$，我们用 $\mathfrak{m} \models \Gamma$ 表示 $\mathfrak{m} \models \tau$对所有 $\tau \in \Gamma$。

### 5.2 证明与预证明与循环证明

一条推理规则的形式为$\nu_1 \quad \frac{\cdots}{\nu}$，其中每个 $\nu$、$\nu_i$（$1 \le i \le n$）也称为一个节点。每个 $\nu_1$，……，$\nu_n$称为该规则的一个前提，而 $\nu$ 称为该规则的结论。该规则的语义是：序列$\nu_1$，……，$\nu_n$的有效性蕴含序列 $\nu$ 的有效性。节点 $\nu$ 的公式 $\tau$称为目标公式，如果除 $\tau$ 外，其他公式在从 $\nu$到某个节点 $\nu_i$（$1 \le i \le n$）的推导中均保持不变。在这种情况下，节点 $\nu$ 中除 $\tau$ 外的其他公式称为 $\nu$ 的上下文。公式对（$\tau_1$，$\tau_2$）中，若 $\tau_1$ 属于 $\nu$，且 $\tau_2$ 属于某个 $\nu_i$，则称其为从 $\nu$到 $\nu_i$ 的推导中的结论—前提（CP）对。

在本文中，我们使用双线推理形式：

$$\frac{\phi_1 \quad \ldots \quad \phi_n}{\phi}$$

来表示这两条规则

$$\frac{\Gamma \Rightarrow \phi_1, \Delta \quad \ldots \quad \Gamma \Rightarrow \phi_n, \Delta}{\Gamma \Rightarrow \phi, \Delta} \quad \text{and} \quad \frac{\Gamma, \phi_1 \Rightarrow \Delta \quad \ldots \quad \Gamma, \phi_n \Rightarrow \Delta}{\Gamma, \phi \Rightarrow \Delta},$$

给定任意上下文 $\Gamma$ 和 $\Delta$。

一棵证明树（或证明）是从根节点向后进行推导而形成的有限树结构。在证明树中，如果一个节点是某个公理的结论，则称其为终结节点。

在循环证明方法中（参见 [16]），预证明 是一棵无穷证明树（即其中某些推导包含无穷多个节点），且存在称为芽节点的非终结叶节点。每个芽节点都与树中的某个祖先相同。芽节点及其相同的祖先合称为回链。预证明中的推导路径 是一个从根节点 $\nu_1\nu_2...\nu_m...$（$m \geq 1$）出发的无限节点序列，其中每对节点 （$\nu_i$， $\nu_{i+1}$）（$i \geq 1$）都是某条规则的 CP 对。如果预证明中存在一条“渐进推导轨迹”（其定义取决于具体的逻辑理论；对于 DL，见后文定义 5.5 ）$\mathfrak{p}$，且该轨迹覆盖每条推导路径，则称该证明树为循环的。

一个证明系统 $Pr$ 由有限条推理规则组成。如果可以构造一棵证明树（以 $\nu$ 为根节点），通过应用 $Pr$中的规则从 $Pr$推导出节点 $\nu$，记作$Pr \vdash \nu$，且该证明树满足以下条件之一：(1) 其所有叶节点均为终结节点；或 (2) 它是一个循环证明。

### 5.3 DL$\mathfrak{p}$的证明系统

带标签的证明系统 $Pr_{dlp}$ 面向DL$\mathfrak{p}$，由两部分组成：一个有限集合 $Pr_{ldlp}$，其中的核心规则用于推导DL$\mathfrak{p}$ 公式，如表2所列，以及一个有限集合$\mathbf{Pr}^{op}$ 中的规则，作为DL$\mathfrak{p}$的一个参数，用于刻画$\mathbf{P}$中程序的操作语义。 $Pr_{ldlp}$ 中的规则并不依赖于$\mathbf{P}$中程序的显式结构，而是依赖于根据$\mathbf{Pr}^{op}$ 对程序转换所作的推导作为其侧条件。 $Pr_{dlp}$ 提供了一个通用逻辑框架，但模不同的程序理论，即$\mathbf{Pr}^{op}$。

$\mathbf{Pr}^{op}$ 的内容依赖于$\mathbf{P}$ 中程序的显式结构，并可因情形而异。通过$\mathbf{Pr}^{op}$中的规则，在系统 $Pr_{dlp}$ 中我们可以分别推导出程序转换 $\mathfrak{F}_{pt}$ 和终止 $\mathfrak{F}_{ter}$。然而，为了与PLK结构K（$\mathbf{P}$，$\mathbf{F}$）相一致，并满足DL$\mathfrak{p}$的可靠性与完备性，我们需要对$\mathbf{Pr}^{op}$ 作出如下假设，如下一定义所述。

定义 *5.1*（关于集合 $\boldsymbol{Pr}^{op}$ *(*$Pr_{dlp}$*)* 的假设。参数 $\mathbf{Pr}^{op}$ （作为 $Pr_{dlp}$的一部分）满足以下条件

1. 与 K (P, F) 的一致性。对于任意 $\mathfrak{m} \in$M、 $\Gamma$，使得 $\mathfrak{m} \models \Gamma$，以及任意 $\sigma \in$L，若 $\mathfrak{m}(\sigma) \alpha/\alpha' \longrightarrow w$是 K 上关于某些 $\alpha$、 $\alpha' \in$ P 和 $w \in$ S 的关系，则存在一个标签 $\sigma' \in$ L，使得 $\mathfrak{m}(\sigma') = w$ 且$\models (\Gamma \Rightarrow (\alpha, \sigma) \longrightarrow (\alpha', \sigma'))$. 2. 关于 $\mathfrak{F}_{pt}$和 $\mathfrak{F}_{ter}$的可靠性。对于系统 $Pr_{dlp}$中的任意推导 $Pr_{dlp} \vdash (\Gamma \Rightarrow (\alpha, \sigma) \longrightarrow (\alpha', \sigma'))$（或 $Pr_{dlp} \vdash (\Gamma \Rightarrow \sigma \Downarrow \alpha)$）， $\models (\Gamma \Rightarrow (\alpha, \sigma) \longrightarrow (\alpha', \sigma'))$（或 $\models (\Gamma \Rightarrow \sigma \Downarrow \alpha)$）。3. 关于 $\mathfrak{F}_{pt}$和 $\mathfrak{F}_{ter}$的完备性。对于任意有效序列 $\Gamma \Rightarrow (\alpha, \sigma) \longrightarrow (\alpha', \sigma')$（或 $\Gamma \Rightarrow \sigma \Downarrow \alpha$）， $Pr_{dlp} \vdash (\Gamma \Rightarrow (\alpha, \sigma) \longrightarrow (\alpha', \sigma'))$（或 $Pr_{dlp} \vdash (\Gamma \Rightarrow \sigma \Downarrow \alpha)$）。4. 简单条件。对于任意有效序列 $\Gamma \Rightarrow (\alpha, \sigma) \longrightarrow (\alpha', \sigma')$，存在一个上下文 $\Gamma'$，其中所有公式均为非动态公式，使得 $\models (\Gamma' \Rightarrow (\alpha, \sigma) \longrightarrow (\alpha', \sigma'))$ 且 $\models (\Gamma \Rightarrow \Gamma')$。

直观而言，与 $\mathcal{K}(\mathbf{L}, \mathbf{F})$ 的一致性$\mathfrak{p}$表明：一旦世界的起始节点可以用标签表示，世界之间在$\mathcal{K}$ 上的关系就可以表示为 DL 的带标签公式中的程序转移。 $Pr_{ldlp}$ 这对于证明规则的可靠性至关重要（见附录A中定理5.3 的证明）。相对于$\mathfrak{F}_{pt}$ 和 $\mathfrak{F}_{ter}$ 的可靠性与完备性是证明定理 5.3、7.3 和7.15所必需的。可靠性条件表明，证明系统$\mathbf{Pr}^{op}$ 推导出的内容不会超出$\mathcal{K}$上有效的程序转移；而完备性条件则表明，$\mathbf{Pr}^{op}$ 足以推导出所有这些行为。“简单条件”这一假设是我们证明 DL$\mathfrak{p}$ 条件完备性（定理7.15）所必需的。它意味着程序转移的条件不取决于其他程序的行为。实际中的程序语言通常都是如此。然而，这篇提交给 ACM 的稿件

$$\frac{}{\Gamma \Rightarrow (x := e, \sigma) \longrightarrow (\downarrow, \sigma_e^x), \Delta}\ (x{:=}e) \qquad \frac{\Gamma \Rightarrow (\alpha_1, \sigma) \longrightarrow (\alpha_1', \sigma'), \Delta}{\Gamma \Rightarrow (\alpha_1; \alpha_2, \sigma) \longrightarrow (\alpha_1'; \alpha_2, \sigma'), \Delta}\ (;)$$

$$\frac{\Gamma \Rightarrow (\alpha_1, \sigma) \longrightarrow (\downarrow, \sigma'), \Delta}{\Gamma \Rightarrow (\alpha_1; \alpha_2, \sigma) \longrightarrow (\alpha_2, \sigma'), \Delta}\ (;\downarrow) \qquad \frac{\Gamma \Rightarrow (\alpha_1, \sigma) \longrightarrow (\alpha_1', \sigma'), \Delta \quad \Gamma \Rightarrow \sigma : \phi, \Delta}{\Gamma \Rightarrow (\mathit{if}\ \phi\ \mathit{then}\ \alpha_1\ \mathit{else}\ \alpha_2\ \mathit{end}, \sigma) \longrightarrow (\alpha_1', \sigma'), \Delta}\ (\mathit{ite1})$$

$$\frac{\Gamma \Rightarrow (\alpha_2, \sigma) \longrightarrow (\alpha_2', \sigma'), \Delta \quad \Gamma \Rightarrow \sigma : \neg\phi, \Delta}{\Gamma \Rightarrow (\mathit{if}\ \phi\ \mathit{then}\ \alpha_1\ \mathit{else}\ \alpha_2\ \mathit{end}, \sigma) \longrightarrow (\alpha_2', \sigma'), \Delta}\ (\mathit{ite2})$$

$$\frac{\Gamma, \sigma : \phi \Rightarrow (\alpha, \sigma) \longrightarrow (\alpha', \sigma'), \Delta \quad \Gamma \Rightarrow \phi : \sigma, \Delta}{\Gamma \Rightarrow (\mathit{while}\ \phi\ \mathit{do}\ \alpha\ \mathit{end}, \sigma) \longrightarrow (\alpha';\ \mathit{while}\ \phi\ \mathit{do}\ \alpha\ \mathit{end}, \sigma'), \Delta}\ (\mathit{wh1})$$

$$\frac{\Gamma, \sigma : \phi \Rightarrow (\alpha, \sigma) \longrightarrow (\downarrow, \sigma'), \Delta \quad \Gamma \Rightarrow \sigma : \phi, \Delta}{\Gamma \Rightarrow (\mathit{while}\ \phi\ \mathit{do}\ \alpha\ \mathit{end}, \sigma) \longrightarrow (\mathit{while}\ \phi\ \mathit{do}\ \alpha\ \mathit{end}, \sigma'), \Delta}\ (\mathit{wh1}\downarrow)$$

$$\frac{\Gamma \Rightarrow \sigma : \neg\phi, \Delta}{\Gamma \Rightarrow (\mathit{while}\ \phi\ \mathit{do}\ \alpha\ \mathit{end}, \sigma) \longrightarrow (\downarrow, \sigma), \Delta}\ (\mathit{wh2})$$

表1. while 程序迁移的部分规则（$\mathbf{Pr}^{op}$）$_W$

$$\frac{\{\Gamma \Rightarrow \sigma' :\ [\alpha']\phi,\ \Delta\}_{(\alpha',\sigma') \in \Phi}}{\Gamma \Rightarrow \sigma :\ [\alpha]\phi,\ \Delta}\ 1\ ([\alpha]R)\ \text{,其中}\ \Phi =_{df} \{(\alpha',\ \sigma')\ |\ Pr_{dlp} \vdash (\Gamma \Rightarrow\ (\alpha, \sigma)\ \rightarrow (\alpha', \sigma')\ \Delta)\}$$

$$\frac{\Gamma, \sigma' :\ [\alpha']\phi \Rightarrow \Delta}{\Gamma, \sigma :\ [\alpha]\phi \Rightarrow \Delta}\ 1\ ([\alpha]L),\quad \text{如果}\ Pr_{dl}{}_{p} \vdash (\Gamma \Rightarrow\ (\alpha, \sigma)\ \rightarrow (\alpha', \sigma')\ \Delta)$$

$$\frac{\sigma : \phi}{\sigma : \{\!\downarrow\!\}\phi}\ (\{\downarrow\}) \quad \left|\ \frac{}{\Gamma \Rightarrow \Delta}\ {}^{2}(\mathit{Ter}) \quad \right| \quad \frac{\Gamma \Rightarrow \Delta}{\mathit{Sub}(\Gamma)\ \Rightarrow \mathit{Sub}\ (\Delta)}\ {}^{3}(\mathit{Sub}) \quad \left|\ \frac{}{\Gamma,\ \sigma :\ \phi \Rightarrow \sigma :\ \phi,\ \Delta}\ (\mathit{ax})\right.$$

$$\frac{\Gamma \Rightarrow \sigma : \phi \quad \Gamma,\ \sigma :\ \phi \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}\ (\mathit{Cut}) \quad \left|\ \frac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \sigma :\ \phi,\ \Delta}\ (\mathit{WkR}) \quad \right| \quad \frac{\Gamma \Rightarrow \Delta}{\Gamma,\ \sigma :\ \phi \Rightarrow \Delta}\ (\mathit{WkL}) \quad \left|\ \frac{\sigma :\ \phi,\ \sigma :\ \phi}{\sigma : \phi}\ (\mathit{Con})\right.$$

$$\frac{\Gamma,\ \sigma :\ \phi \Rightarrow \Delta}{\Gamma \Rightarrow \sigma :\ \neg\phi,\ \Delta}\ (\neg R) \quad \left|\ \frac{\Gamma \Rightarrow \sigma :\ \phi,\ \Delta}{\Gamma,\ \sigma :\ \neg\phi \Rightarrow \Delta}\ (\neg L) \quad \right| \quad \frac{\Gamma \Rightarrow \sigma :\ \phi,\ \Delta \quad \Gamma \Rightarrow \sigma :\ \psi,\ \Delta}{\Gamma \Rightarrow \sigma :\ \phi \wedge \psi,\ \Delta}\ (\wedge R) \quad \left|\ \frac{\Gamma,\ \sigma :\ \phi,\ \sigma :\ \psi \Rightarrow \Delta}{\Gamma,\ \sigma :\ \phi \wedge \psi \Rightarrow \Delta}\ (\wedge L)\right.$$

[1] $\alpha \notin \{\downarrow\}$。 [2] 对每个 $\sigma :\ \phi \in \Gamma \cup \Delta, \phi \in \mathbf{F}$; 序列 $\Gamma \Rightarrow \Delta$ 有效。 [3] *Sub* 由定义 4.12给出。

表 2. DLp证明系统的规则 $Pr_{ldlp}$

一项假设使 DLp 无法实例化具有“丰富测试”的动态逻辑（参见 [29]），其中测试本身可以是动态公式，例如测试 $[x := e]\phi?$ 。

例 *5.2*（$\mathbf{Pr}^{op}$的一个实例化）。表 1 列出了 $(\mathbf{Pr}^{op})_W$的一组部分规则，用于描述 while 程序的操作语义，其中省略了推导程序终止的规则。在表 1中， $\sigma_e^x$在例 4.9中定义。

在实践中，我们通常认为规则 $\mathbf{Pr}^{op}$ 忠实地体现了程序的操作语义。因此，我们直接信任定义 5.1 中的假设 1、2 和 3 ，而不加以证明。

通过 $Pr_{ldlp}$中的规则，带标签的 DLp 公式可以转换为非动态公式形式的证明义务，随后可以对其进行编码，并通过例如 SAT/SMT 检查过程加以验证。对于其他算子（如 ∨、 → ），可以相应地利用表 2中的规则推导出相应规则。

表中每条规则的说明2 如下。

规则 ($[\alpha]R$) 与 ($[\alpha]L$) 对带标签的 DLp 公式的动态部分进行推理。这两条规则都依赖于旁支推导："$Pr_{dlp} \vdash (\Gamma \Rightarrow (\alpha,\sigma) \rightarrow (\alpha',\sigma') \Delta)$"，将其作为程序转换的子证明过程。在规则 ($[\alpha]R$) 中，$\{...\}_{(\alpha',\sigma')\in\Phi}$表示所有程序状态 $(\alpha',\sigma') \in \Phi$的前提集合。由系统 $Pr_{dlp}$ 的有限性可知，集合 $\Phi$ 必须是有限的（因为只能导出有限多个形式 $(\alpha',\sigma')$）。因此，规则 ($[\alpha]R$) 只有有限个前提。当 $\Phi$ 为空时，结论终止。与规则 ($[\alpha]R$) 相比，规则 ($[\alpha]L$) 对某个程序状态 $(\alpha',\sigma')$ 仅有一个前提。

规则 ($[\downarrow]$) 处理程序为终结程序时的情形 $\downarrow$。其可靠性可由 $\downarrow$在定义 4.3中的语义直接得出。

规则 ($Ter$) 表明，当序列 $\Gamma \Rightarrow \Delta$ 有效，且其中所有带标签公式均为非动态公式时，一条证明分支即告终止。

规则 (Sub) 描述了带标签动态公式的特化过程。对于带标签公式的集合 $A$，$\mathrm{Sub}(A) =_{df} \{\mathrm{Sub}(\tau) \mid \tau \in A\}$，其中 Sub 是一个替换（定义 4.12）。直观地说，如果序列 $\Gamma \Rightarrow \Delta$ 有效，那么它的某个特例 $\mathrm{Sub}(\Gamma) \Rightarrow \mathrm{Sub}(\Delta)$ 也有效。规则 (Sub) 在构造循环证明结构中的芽时起着重要作用（第 5.4 节）。更多细节请参见第 6.1 节。

从（公理）到（$\wedge L$）的规则是继承自传统一阶逻辑的相应规则的"带标签版本"。它们的含义是经典的，此处我们省略对其的讨论。

定理 5.3。 表 $Pr_{ldlp}$ 中的每条规则*2* 都是可靠的。

根据上述说明，定理 5.3 可依据带标签 DLp 公式的语义，在定义 5.1的假设下予以证明。更多详情请参见附录 A 。

### 5.4 为 DLp 构造循环证明结构

我们构建一个用于 DLp的循环带标签证明系统，以识别并接纳潜在的无限推导，如第 2 节所示的示例。基于预证明的概念（第 5.2 节），我们为系统$Pr_{dlp}$构建循环证明结构，其中关键在于引入 DL 中渐进推导轨迹的概念p （定义 5.5）。

接下来，我们先介绍 DLp中渐进推导轨迹的概念，然后将 DLp 的循环证明结构定义为第 5.2 节中已给出概念的一个特殊情形。

定义 *5.4*（推导轨迹）。 推导路径 $\mu_1\mu_2...\mu_k\nu_1\nu_2...\nu_m...$ $(k \geq 0, m \geq 1)$ 上的"推导轨迹"是一个无限公式序列 $\tau_1\tau_2...\tau_m...$，其中每个公式 $\tau_i (1 \leq i \leq m)$ 都位于节点 $\nu_i$ 中。推导 $(\nu_i, \nu_{i+1})$ 的每个 CP 对 $(\tau_i, \tau_{i+1})$ $(i \geq 1)$，根据 $(\nu_i, \nu_{i+1})$ 是 $Pr_{ldlp}$ 中规则的不同实例，满足如下特殊条件：

1. 如果 $(\nu_i,\ \nu_{i+1})$ 是规则 ($[\alpha]R$)、($[\alpha]L$)、($[\downarrow]$)、($\neg R$)、($\neg L$)、($\wedge R$) 或 ($\wedge L$) 的一个实例，那么要么 $\tau_i$ 是目标公式，且 $\tau_{i+1}$ 是通过规则应用对其进行的替换，要么 $\tau_i = \tau_{i+1}$；2. 如果 $(\nu_i,\ \nu_{i+1})$ 是规则 ($Sub$) 的一个实例，那么对于某个 $\sigma \in \mathbf{L}$ 和 $\phi \in \mathfrak{F}_{dlp}$，$\tau_i = Sub(\sigma) : \phi$ 且 $\tau_{i+1} = \sigma : \phi$；3. 如果 $(\nu_i,\ \nu_{i+1})$ 是其他规则的一个实例，那么 $\tau_i = \tau_{i+1}$。

在表达式 $n :: O$ 中，表示我们使用名称 $n$ 来指代对象 $O$。

定义 *5.5*（渐进推导轨迹）。在系统 $Pr_{dlp}$ 的预证明中，给定一条推导轨迹 $\tau_1\tau_2...\tau_m...$，它沿着一条从节点 $\nu_1$ 中的 $\tau_1$ 开始的推导路径 $...\nu_1\nu_2...\nu_m...$ $(m \geq 1)$，推导 $(\nu_i, \nu_{i+1})$ 的一个 CP 对 $(\tau_i, \tau_{i+1})$ $(1 \leq i \leq m)$

如果 $(\tau_i, \tau_{i+1})$ 是规则 $([\alpha]R)$ 的某个实例的以下 CP 对，则称其为一个“渐进步骤”：

$$\frac{\ldots \quad \nu_{i+1} :: (\Gamma \Rightarrow \tau_{i+1} :: (\sigma' : [\alpha']\phi), \Delta) \quad \ldots}{\nu_i :: (\Gamma \Rightarrow \tau_i :: (\sigma : [\alpha]\phi), \Delta),} \ ([\alpha]R);$$

或规则（$[\alpha]$）的某个实例对应的以下 CP 对 $L$):

$$\frac{\nu_{i+1} :: (\Gamma, \tau_{i+1} :: (\sigma' : [\alpha']\phi) \Rightarrow \Delta)}{\nu_i :: (\Gamma, \tau_i :: (\sigma : [\alpha]\phi) \Rightarrow \Delta)} \ ([\alpha]L),$$

并附有一个额外的旁推导 $Pr_{dlp} \vdash (\Gamma \Rightarrow \sigma \Downarrow \alpha, \ \Delta)$。

如果一条推导轨迹包含无限多个渐进步骤，我们就称该轨迹是“渐进的”。

规则 （$[\alpha]L$）的实例的额外附加条件是证明引理 7.9中相应情形的关键（见附录 A）。

定理 5.3 表明 $Pr_{dlp}$ 的每条规则都是可靠的。但这并不意味着证明系统 $Pr_{dlp}$ 是可靠的，因为我们需要确保每个循环证明都能得出有效结论。系统 $Pr_{dlp}$的可靠性将在第 7节中全面讨论。

### 5.5 从动态逻辑理论中提升规则

我们介绍一种技术，可将特定动态逻辑理论（例如 FODL）中的规则提升为 DL 中带标签的规则 [49], $\mathfrak{p}$。借助该技术，可以将现有的动态逻辑理论嵌入 DL$\mathfrak{p}$ ，同时保留基于程序语法结构进行推导的能力。反过来，这也有助于利用特殊推理规则，在特定程序领域中推导 DL$\mathfrak{p}$ 公式。下文将针对满足特定标签条件的一般推理规则，提出一种提升过程（命题5.10）。本技术的应用示例见第6.2节。

我们首先介绍自由标签的概念，作为标签能够执行提升的充分条件。然后，我们在命题5.10中介绍提升过程。

定义 *5.6*（效应等价）。两个世界 $w, \ w' \in \mathbf{S}$ 关于一组无标签公式$A \subseteq \mathfrak{F}_{dlp}$“具有相同效应”，记作 $w =_A w'$，当且仅当对于任意 $\phi \in A, w \models \phi$，iff $w' \models \phi$。

例 *5.7*。 在 DL$\mathfrak{p}$-WP 中，考虑两个世界 $w, \ w' \in \mathbf{S}^W$（它们是从 $Var^{fodl}$ 到 $\mathbb{Z}$的两个映射），满足 $w(x) = 1, \ w(y) = 1$以及 $w'(x) = 2, \ w'(y) = 1$，其中 $x, \ y \in Var^{fodl}$。令公式 $\phi = (x + y > 1) \in \mathbf{F}^{afo}$。那么 $w =_{\{\phi\}} w'$，尽管 $w \neq w'$。

定义 *5.8*（自由标签）。 若标签 $\sigma \in \mathbf{L}$ 相对于公式集合 $A$ 满足以下条件，则称其为“自由”的：对于任意世界 $w \in \mathbf{S}$，都存在一个标签映射 $\mathfrak{m} \in \mathbf{M}$ 使得

$$w =_A \mathfrak{m}(\sigma).$$

我们将相对于 $A$ 的所有自由标签构成的集合记为 $free(\mathbf{L}, A)$。

直观而言，标签 $\sigma$相对于公式集合 $A$ 的自由性意味着 $\sigma$足够通用，因此不会影响 $A$中公式的有效性。

例 5.9. 在 DL$\mathfrak{p}$-WP 中，设 $\sigma = \{x \mapsto t + 1\} \in \mathbf{L}^W$（其中 $t$ 是一个变量）。对于 $\phi = (x + y > 1)$， $\sigma$ 相对于 $\{\phi\}$ 是自由的（例 5.7）。因为对于任意世界 $w \in \mathbf{S}^W$，令 $w' =_{df} w[t \mapsto x - 1]$，则有 $w =_{\{\phi\}} m_{w'}(\sigma)$，因为$w(x) = \mathfrak{m}_{w'}(\sigma)(x)$，且 $w(y) = \mathfrak{m}_{w'}(\sigma)(y)$。另一方面，令 $\sigma' = \{x \mapsto 0, y \mapsto 0\}$，则 $\sigma'$ 相对于 $\{\phi\}$ 不是自由的。因为对于任意 $w \in \mathbf{S}^W$， $\mathfrak{m}_w(\sigma')(x) = \mathfrak{m}_w(\sigma')(y) = 0$，所以 $\mathfrak{m}_w(\sigma') \not\models \phi$。

我们可以看到，与 $\sigma$相比， $\sigma'$ 过于“显式”，因此会影响公式 $\phi$的有效性。

对于一组无标签公式 $A$，我们将 $\sigma$ 写作： $A$，表示带标签公式的集合 $\{\sigma\colon \phi \mid \phi \in A\}$。

命题 5.10（提升过程）。给定一个如下形式的可靠规则

$$\frac{\Gamma_1 \Rightarrow \Delta_1 \quad \ldots \quad \Gamma_n \Rightarrow \Delta_n}{\Gamma \Rightarrow \Delta}\ , n \geq 1,$$

其中所有公式均未标注，则规则

$$\frac{\sigma : \Gamma_1 \Rightarrow \sigma : \Delta_1 \quad \ldots \quad \sigma : \Gamma_n \Rightarrow \sigma : \Delta_n}{\sigma : \Gamma \Rightarrow \sigma : \Delta}$$

对于任意标签 $\sigma \in \mathit{free}(\mathbf{L}\ \Gamma \cup \Delta \cup \Gamma_1 \cup \Delta_1 \cup \ldots \cup \Gamma_n \cup \Delta_n)$ 都是可靠的。

命题5.10的证明见附录A，证明基于上文定义的自由标签概念。

## 6 案例研究

本节通过几个实例展示 DLp 的潜在用途。

我们首先展示如何根据表2中提出的循环证明系统推导 DLp 中的带标签动态公式。我们以一个 while 程序为例说明演绎过程（第 6.1 节）。附录 B 简要介绍了 DLp 对同步语言 Esterel [12] 的另一种实例化，并展示了一个 Esterel 程序的循环推导。第二个例子更能凸显 DLp 的优势，因为某些 Esterel 程序的循环结构是隐式的。

其次，我们以 FODL 中的规则为例，说明如何在 DLp 中进行规则提升（第 6.2 节）。这展示了 DLp 与现有动态逻辑理论的兼容性，使这些理论能够在 DLp中重用。正如我们所见，通过采用特殊论域中的组合规则，也有助于提高 DLp 中推导的效率。

在第 4 节和 5 节中，我们已经了解了 DLp-WP 这一 DLp实例化理论。第 6.3 节和 6.4 节进一步介绍了 DLp的更多复杂实例化。

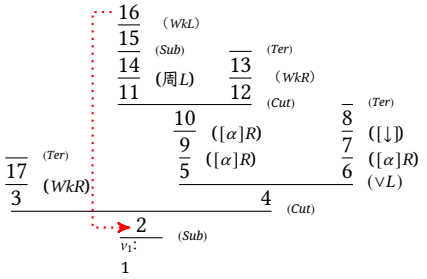
在第 6.3 节中，我们将 FODL 理论嵌入 DLp。此例展示了 DLp 在实践中对不同程序模型的潜在用途，因为 FODL 是许多动态逻辑变体（例如 [7, 43, 44, 62]）所依据的基础理论。通过同时对 while 程序和正则程序进行推理的示例，我们还展示了 DLp的异质性：不同程序模型可以借助不同的操作语义轻松进行比较。

在第 6.4 节中，我们进一步展示了更复杂的标签和公式编码。我们提出将过程逻辑的一阶版本 [28] 编码到 DLp中。过程逻辑提供了一个逻辑框架，不仅可以对程序终止后的程序性质进行推理，还可以对执行过程中的性质进行推理。从这个示例中我们可以看到，在 DLp 中，标签和公式比传统的霍尔风格逻辑更具表达力；在传统的霍尔风格逻辑中，只能对前后性质进行推理。在附录 C语言 中，我们通过将分离逻辑 [51]编码到 DLp中，给出另一个示例来说明这一点。这提供了一种新颖的方式，可直接通过符号执行对分离逻辑公式进行推理。

### 6.1 一个 while 程序的无环推导

我们根据表 4.7 中的规则证明例 2中的性质。该性质可由以下等价的带标签序列表示

$$v_1 =_{df} \sigma_1 : n \geq 0 \Rightarrow \sigma_1 : [WP](s = ((N+1)N)/2),$$




其他符号的定义：

$WP =_{df} \{$当 $(n>0)$ 时执行 $s := s+n;\ n := n-1$ 结束 $\}$

$\alpha_1 =_{df} s := s+n;\ n := n-1$

$\phi_1 =_{df} (s = ((N+1)N)/2)$

$\sigma_1 =_{df} \{n \mapsto N, s \mapsto 0\}$

$\sigma_2 =_{df} \{n \mapsto N-m, s \mapsto (2N-m+1)m/2\}$

$\sigma_3 =_{df} \{n \mapsto N-m, s \mapsto (2N-(m+1)\ +1)(m+1)/2\}$

$\sigma_4 =_{df} \{n \mapsto N-(m+1)\ s \mapsto (2N-(m+1)\ +1)(m+1)/2\}$

| | | | |
|---|---|---|---|
| 1: | $\sigma_1 : n \geq 0$ | $\Rightarrow$ | $\sigma_1 :$ [当 $(n>0)$ 满足条件时执行 $\alpha_1$ *end* $]\phi_1$ |
| 2: | $\sigma_2 : n \geq 0$ | $\Rightarrow$ | $\sigma_2 :$ [当 $(n>0)$ 满足条件时执行 $\alpha_1$ *end* $]\phi_1$ |
| 3: | $\sigma_2 : n \geq 0$ | $\Rightarrow$ | $\sigma_2 :$ [当 $(n>0)$ 执行 $\alpha_1$ 结束 $]\phi_1, \sigma_2 : (n>0 \vee n \leq 0)$ |
| 17: | $\sigma_2 : n \geq 0$ | $\Rightarrow$ | $\sigma_2 : (n>0 \vee n \leq 0)$ |
| 4: | $\sigma_2 :\ n \geq 0,\ \sigma_2 :\ (n>0 \vee n \leq 0)$ | $\Rightarrow$ | $\sigma_2 :$ [当 $(n>0)$ 执行 $\alpha_1$ 结束 $]\phi_1$ |
| 5: | $\sigma_2 :\ n \geq 0, \sigma_2 :\ n>0$ | $\Rightarrow$ | $\sigma_2 :$ [当 $(n>0)$ 时执行 $\alpha_1$ 结束 $]\phi_1$ |
| 9: | $\sigma_2 :\ n \geq 0,\ \sigma_2 :\ n>0$ | $\Rightarrow$ | $\sigma_3 :$ [$n :=n-1$; 当 $(n>0)$ 时执行 $\alpha_1$ 结束 $]\phi_1$ |
| 10: | $\sigma_2 :\ n \geq 0,\ \sigma_2 :\ n>0$ | $\Rightarrow$ | $\sigma_4 :$ [当 $(n>0)$ 时执行 $\alpha_1$ 结束 $]\phi_1$ |
| 11: | $\sigma_2 :\ n \geq 0,\ \sigma_2 :\ n>0,\ \sigma_4 :\ n \geq -1,\ \sigma_4 :\ n \geq 0$ | $\Rightarrow$ | $\sigma_4 :$ [当 $(n>0)$ 时执行 $\alpha_1$ 结束 $]\phi_1$ |
| 14: | $\sigma_4 :\ n \geq -1,\ \sigma_4 :\ n \geq 0$ | $\Rightarrow$ | $\sigma_4 :$ [当 $(n>0)$ 时执行 $\alpha_1$ 结束 $]\phi_1$ |
| 15: | $\sigma_2 :\ n \geq -1,\ \sigma_2 :\ n \geq 0$ | $\Rightarrow$ | $\sigma_2 :$ [当 $(n>0)$ 时执行 $\alpha_1$ 结束 $]\phi_1$ |
| 16: | $\sigma_2 :\ n \geq 0$ | $\Rightarrow$ | $\sigma_2 :$ [当 $(n>0)$ 时执行 $\alpha_1$ 结束 $]\phi_1$ |
| 12: | $\sigma_2 :\ n \geq 0,\ \sigma_2 :\ n>0$ | $\Rightarrow$ | $\sigma_4 :$ [当 $(n>0)$ 时执行 $\alpha_1$ 结束 $]\phi_1,\ \sigma_4 :\ n \geq -1,\ \sigma_4 :\ n \geq 0$ |
| 13: | $\sigma_2 :\ n \geq 0,\ \sigma_2 :\ n>0$ | $\Rightarrow$ | $\sigma_4 :\ n \geq -1,\ \sigma_4 :\ n \geq 0$ |
| 6: | $\sigma_2 :\ n \geq 0,\ \sigma_2 :\ n \leq 0$ | $\Rightarrow$ | $\sigma_2 :$ [当 $(n>0)$ 时 $\alpha_1$ 结束 $]\phi_1$ |
| 7: | $\sigma_2 :\ n \geq 0,\ \sigma_2 :\ n \leq 0$ | $\Rightarrow$ | $\sigma_2 :$ $[\downarrow]\phi_1$ |
| 8: | $\sigma_2 :\ n \geq 0,\ \sigma_2 :\ n \leq 0$ | $\Rightarrow$ | $\sigma_2 :\ (s = ((N+1)N)/2)$ |

表3. 性质 $\nu_1$ 的推导

其中 $\sigma_1 =_{df} \{n \mapsto N,\ \ s \mapsto 0\}$ 描述了 $WP$ 的初始配置。

表3 展示了其推导。对于使用表（$[\alpha]R$）中的推理规则推导出的规则实例，我们省略所有旁支推导作为子证明过程。表1中的非基本规则（$\vee L$）可以通过以下 $\neg$ 和 $\wedge$ 的规则推导得到：

$$\dfrac{\dfrac{\Gamma, \phi \Rightarrow \Delta}{\Gamma \Rightarrow \neg\phi, \Delta}\ (\neg R) \qquad \dfrac{\Gamma, \psi \Rightarrow \Delta}{\Gamma \Rightarrow \neg\psi, \Delta}\ (\neg R)}{\dfrac{\Gamma \Rightarrow (\neg\phi) \wedge (\neg\psi), \Delta}{\Gamma, \phi \vee \psi \Rightarrow \Delta}\ (\neg L)}\ (\wedge R).$$

从序列1到序列2的推导（从14到15的推导亦然）依据规则（$Sub$）：

$$\dfrac{\Gamma \Rightarrow \Delta}{\Gamma[e/x] \Rightarrow \Delta[e/x]}\ (Sub),$$

其中，函数$(\cdot)[e/x]$ 是定义4.12中定义的抽象替换的一个实例化。对于任意标签 $\sigma$，$\sigma[e/x]$ 通过将 $x$ 中的每个自由变量 $\sigma$ 替换为项 $e$，返回相应的标签。我们注意到 $\sigma_1 = \sigma_2[0/m]$，因此，序列1是通过替换$(\cdot)[0/m]$得到的序列2的一个特例。直观地说，标签 $\sigma_2$ 捕获了之后的程序配置

程序 $WP$ 的第 $m$个循环（$m \geq 0$）。这一步至关重要，因为从序列 2 开始，我们可以找到一个与节点 2 相同的芽节点 — 16。

从序列2到{3, 4}的推导给出了一个引理： $\sigma_2 : (n > 0 \vee n \leq 0)$，它显然有效。序列16表示程序$WP$的第$(m+1)$个循环结束。从节点10到16，我们将左侧的公式变换为一个平凡的逻辑等价形式，以便应用从序列14到15的规则（$Sub$）。序列14是序列15的一个特例，因为 $\sigma_4 = \sigma_2[m+1/m]$。

整个证明树是循环的，因为唯一的推导路径为：2，4，5，9，10，11，14，15，16，2，.....具有一条渐进推导轨迹，其元素在表3中带有下划线。

上述演绎过程的一个特点是，while 程序$WP$（即$while...do...end$）的循环结构体现在循环推导树本身中。要对$WP$进行推理，无需使用用于分解循环结构的推理规则。这在循环结构通常是隐式的程序模型中尤为有用，例如类 CCS 进程代数 [38, 39] 和命令式同步语言 [12, 55]。DLp 提供了增量推理，使我们能够避免像文献 [9, 55] 中的工作那样预先进行程序变换。

### 6.2 从 FODL 提升规则

下面给出 DLp-WP 中的两个示例，以说明如何应用 FODL 理论中的现有推理规则（参见 [29]），通过第 5.5 节定义的提升过程，推导组合式 while 程序。

在 FODL 中，考虑以下规则

$$\frac{\Gamma \Rightarrow [\alpha][\beta]\phi, \Delta}{\Gamma \Rightarrow [\alpha;\beta]\phi, \Delta}\ ([;])\,,$$

这意味着，要证明公式 $[\alpha;\ \beta]\phi$，我们只需证明公式 $[\alpha][\beta]\phi$，其中程序 $\alpha$先与程序 $\beta$分离证明。该规则源自有效公式 $[\alpha][\beta]\phi \rightarrow [\alpha;\ \beta]\phi$，并作为组合规则出现在许多基于 FODL 的动态逻辑演算中（例如 [7]）。根据命题5.10，在 DLp-WP 中，我们可以将([;  ]) 提升为一条规则

$$\frac{\sigma : \Gamma \Rightarrow \sigma : [\alpha][\beta]\phi, \sigma : \Delta}{\sigma : \Gamma \Rightarrow \sigma : [\alpha;\beta]\phi, \sigma : \Delta}\ (\sigma[;])\,,$$

其中 $\sigma$是 $free(\Gamma \cup \Delta \cup \{[\alpha][\beta]\phi, [\alpha;\ \beta]\phi\})$ 中的一个自由配置。作为一条附加规则，在系统 $Pr_{dlp}$ 中，$(\sigma[;])$ 为顺序程序提供组合推理。在验证诸如 $\Gamma,\ \sigma' : [\beta]\phi \not\Rightarrow \sigma : [\alpha;\beta]\phi,\ \Delta$ 这样的性质时，这条规则很有用；此时，我们或许只需对程序 $\alpha$ 进行符号执行即可完成证明，如下所示：

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{\Gamma, \sigma' : [\beta]\phi \Rightarrow \sigma' : [\beta]\phi, \Delta}\ (ax)}{\Gamma, \sigma' : [\beta]\phi \Rightarrow \sigma' : [\downarrow][\beta]\phi, \Delta}\ ([\downarrow])}{\Gamma, \sigma' : [\beta]\phi \Rightarrow \sigma' : [\alpha]R, \Delta}\ ([\alpha]R)}{\vdots}}{\Gamma, \sigma' : [\beta]\phi \Rightarrow \sigma : [\alpha][\beta]\phi, \Delta}\ ([\alpha]R)}{\Gamma, \sigma' : [\beta]\phi \Rightarrow \sigma : [\alpha;\beta]\phi, \Delta}\ (\sigma[;])\,,$$

尤其是在验证程序 $\beta$时，成本可能非常高。

另一个例子是以下规则

$$\frac{\phi \Rightarrow \psi}{[\alpha]\phi \Rightarrow [\alpha]\psi}\ ([Gen])$$

$$\frac{}{\Gamma \Rightarrow (x := e, \sigma) \longrightarrow (\downarrow, \sigma_e^x), \Delta}\ (x{:=}e) \qquad \frac{\Gamma \Rightarrow \sigma : \phi, \Delta}{\Gamma \Rightarrow (\phi?, \sigma) \longrightarrow (\downarrow, \sigma), \Delta}\ (\phi?)$$

$$\frac{\Gamma \Rightarrow (\alpha, \sigma) \longrightarrow (\alpha', \sigma'), \Delta}{\Gamma \Rightarrow (\alpha;\beta, \sigma) \longrightarrow (\alpha';\beta, \sigma'), \Delta}\ 1\ (;) \qquad \frac{\Gamma \Rightarrow (\alpha, \sigma) \longrightarrow (\downarrow, \sigma'), \Delta}{\Gamma \Rightarrow (\alpha;\beta, \sigma) \longrightarrow (\beta, \sigma'), \Delta}\ (;\downarrow)$$

$$\frac{}{\Gamma \Rightarrow (\alpha \cup \beta, \sigma) \longrightarrow (\alpha, \sigma), \Delta}\ (\cup 1) \qquad \frac{}{\Gamma \Rightarrow (\alpha \cup \beta, \sigma) \longrightarrow (\beta, \sigma), \Delta}\ (\cup 2)$$

$$\frac{}{\Gamma \Rightarrow (\alpha^*, \sigma) \longrightarrow (\alpha;\alpha^* \cup \mathit{true}?, \sigma), \Delta}\ 2\ (*)$$

表 4. 正则程序程序转换的部分规则（$\mathbf{Pr}^{op}$）$^{fodl}$

用于生成模态 $[\cdot]$,该规则曾用于推导 FODL 中星号正则程序的结构规则（参见 [29]）。根据命题5.10，我们将 $([Gen])$ 提升为以下规则：

$$\frac{\sigma : \phi \Rightarrow \sigma : \psi}{\sigma : [\alpha]\phi \Rightarrow \sigma : [\alpha]\psi}\ (\sigma[Gen]),$$

其中 $\sigma \in \mathit{free}(\mathbf{L}, \{[\alpha]\phi,\ [\alpha]\psi,\ \phi,\ \psi\})$。例如，在推导性质 $\sigma\colon [\alpha]\beta\phi \Rightarrow \sigma\colon [\alpha]\beta'\psi$ 时，这很有用，因为我们可以按如下方式跳过程序 $\alpha$ 的推导：

$$\frac{\sigma : \langle\beta\rangle\phi \Rightarrow \sigma : \langle\beta'\rangle\psi}{\sigma : [\alpha]\langle\beta\rangle\phi \Rightarrow \sigma : [\alpha]\langle\beta'\rangle\psi}\ (\sigma[Gen]),$$

并直接专注于推导程序 $\beta$ 和 $\beta'$。

从这两个示例可以看出，在实际推导中，提升过程可用于减轻某些验证的负担。

### 6.3 FODL 理论中 DLp 的实例化

我们用 FODL 理论实例化 DLp 。由此得到的理论，即 DLp-FODL，提供了另一种通过符号执行正则程序来推理 FODL 公式的方法。

实例化过程主要遵循我们在示例 4.4、4.9、4.10 和 5.2中看到的 while 程序的实例化过程，唯一的区别是：**(1)** 参数 **P** 被实例化为正则程序的集合（第 3 节），记为$\mathbf{P}^{fodl}$；**(2)** 程序行为由一组正则程序的规则来描述，记为 $(\mathbf{Pr}^{op})^{fodl}$，其中关于正则程序程序转换部分的规则见表 4。我们省略了关于正则程序程序终止部分的规则。

将我们针对 FODL 的证明系统：$(Pr_{dlp})_{fodl} =_{df} Pr_{ldlp} \cup (\mathbf{Pr}^{op})^{fodl}$ 与 FODL 的传统证明系统（参见 [29]）进行比较很有意思。通过简单的观察可以看出，对于非星号正则程序，我们的证明系统能够做到传统证明系统所能做到的事情。例如，一个 FODL 公式

$$[(\alpha;\beta) \cup \alpha]\phi,$$

其中令 $\alpha =_{df} (x := x+1)$ 和 $\beta =_{df} (y := 0)$，在 FODL 的传统证明系统中可以通过使用某些规则按以下过程推导得出：

$$\dfrac{\dfrac{\dfrac{x \geq 0 \Rightarrow [\beta]x+1>0}{x \geq 0 \Rightarrow [\alpha][\beta]x>0}\,(x:=e)}{x \geq 0 \Rightarrow [\alpha;\beta]x>0}\,(;) \qquad x \geq 0 \Rightarrow [\alpha]x>0}{x \geq 0 \Rightarrow [(\alpha;\beta)\cup\alpha]x>0}\,(\cup).$$

在 $(Pr_{dlp})^{fodl}$ 中，相应地，我们可以找到一个逻辑等价的标记版本：

$$\{x \mapsto t\} : x \geq 0 \Rightarrow \{x \mapsto t\} : [(\alpha;\beta)\cup\alpha]x>0$$

并通过应用规则 $([\alpha]R)$，以及使用 $(\mathbf{Pr}^{op})^{fodl}$ 中相应的操作规则（下文未列出），得到以下推导：

$$\dfrac{\dfrac{\{x \mapsto t\} : x \geq 0 \Rightarrow \{x \mapsto t+1\} : [\beta]x>0}{\{x \mapsto t\} : x \geq 0 \Rightarrow \{x \mapsto t\} : [\alpha;\beta]x>0}\,([\alpha]R) \qquad \{x \mapsto t\} : x \geq 0 \Rightarrow \{x \mapsto t\} : [\alpha]x>0}{\{x \mapsto t\} : x \geq 0 \Rightarrow \{x \mapsto t\} : [(\alpha;\beta)\cup\alpha]x>0}\,([\alpha]R).$$

我们尤其关注 $(Pr_{dlp})^{fodl}$（关于合适的 $(\mathbf{Pr}^{op})^{fodl}$，此处仅列出部分规则）是否像 FODL 的传统证明系统一样，相对于整数算术理论是完备的。正如我们现在所看到的，证明这一点的一种方法是应用第 7.2 节提出的 DL$\mathfrak{p}$ 条件完备性结果，其中关键步骤是证明证明系统 $(Pr_{dlp})^{fodl}$ 中的正则程序行为良好（定义 7.14）。未来工作中将详细讨论这些方面。

DL 验证框架的异质性$\mathfrak{p}$ 可以从这个例子中看出。尽管在我们的讨论中，while 程序是正则程序的子集（见第 3 节），但 while 程序有自己的一套程序迁移推理规则（表 1），这与正则程序的规则（表 4）不同。DL$\mathfrak{p}$ 公式提供了一种便捷的方式来比较不同模型的行为。例如，给定一个正则程序 $WP_r$，它在语法上等价于 $WP$（例 4.4）：

$$WP_r =_{df} ((n>0)?;s := s+n;n := n-1)^*;\neg(n>0)?,$$

我们希望验证，根据各自的操作语义，它们的行为是否会导向相同的结果。这一性质可以用如下 DL$\mathfrak{p}$ 公式描述：

$$\cdot \Rightarrow \sigma : [WP][WP'_r](s=s' \wedge n=n'),$$

其中 $\sigma = \{s \mapsto t, n \mapsto N, s' \mapsto t, n' \mapsto N\}$；$WP'_r$是通过将$WP_r$中所有变量$s,n$替换为对应的新变量 $s',n'$得到的，以避免变量冲突；$t,N$是不同于 $s,n,s',n'$的新变量。直观地说，该公式表示：如果$WP$ 和$WP_r$ 的输入相同，那么分别运行它们（不进行交互）后，它们的输出也相同。

### 6.4 在 DL$\mathfrak{p}$ 中对过程逻辑进行编码

DL$\mathfrak{p}$ 允许使用更复杂的标签和公式：标签可以不再局限于简单的程序状态，而公式也可以不再局限于简单的静态公式（即在某个世界中非真即假的公式）。下面，我们简要介绍一种可能的方法，将过程逻辑 [28] (PL) 的一阶版本编码到 DL$\mathfrak{p}$ 中，即 DL$\mathfrak{p}$-PL。DL$\mathfrak{p}$-PL 能够使用时态公式来描述并推理程序的渐进行为。与 Hoare 稿件相比，提交给 ACM 的稿件

逻辑而言，它更适用于反应式系统 [26], , 在这类系统中，程序可能永远不会终止，而我们更关注某个性质是否在中间状态成立。

过程逻辑可以看作一类动态逻辑，其中公式的语义依据路径而非世界来定义。PL 动态公式的形式 $[\alpha]\phi$ 继承自 PDL 的形式，其中 $\alpha$ 与 PDL 中一样是正则程序。公式 $\phi$ 是按下述方式定义的时态公式：(1) 任一原子命题 $p$ 都是时态公式；(2) 若 $\phi$ 和 $\psi$ 是时态公式，则$\mathbf{f}\ \phi$ 和 $\phi\,\mathbf{suf}\ \psi$ 是时态公式；(3) 若 $\phi$ 和 $\psi$ 是时态公式，则 $\neg\phi$、$\phi \wedge \psi$ 是时态公式。因此，正则程序和时态公式的语义由世界组成的路径给出。对于正则程序，其语义对应于其执行路径的集合。对于时态公式，给定路径 $tr$，其语义定义如下：

1. $tr \models p$，若 $tr_b \models p$；2. $tr \models \mathbf{f}\ \phi$，若 $tr_b \models \phi$；3. $tr \models \phi\mathbf{suf}\psi$，若存在一条路径 $tr'$ 使得 (i) $tr' \prec_s tr$ 且 $tr' \models \psi$，并且 (ii) 对于所有满足 $tr' \prec_s tr'' \prec_s tr$ 的路径 $tr''$，$tr'' \models \phi$；4. $tr \models \neg\phi$，若 $tr \not\models \phi$，以及 $tr \models \phi \wedge \psi$，若 $tr \models \phi$ 且 $tr \models \psi$。

注意，算子$\mathbf{f}$ 与$\mathbf{suf}$ 足以表达通常时态算子的含义（参见 [28]），例如“下一”算子：$\mathbf{n}\,\phi =_{df} false\,\mathbf{suf}\ \phi$，“将来”算子：$\Diamond\phi =_{df} \phi \vee (true\,\mathbf{suf}\ \phi)$，等等。基于时态公式的语义，动态PL公式 $[\alpha]\phi$ 相对于路径 $tr$ 的语义定义如下：

$$tr \models [\alpha]\phi, \text{ if for all execution paths } tr' \text{ of } \alpha,\ tr' \cdot tr \models \phi.$$

我们的实例化过程与DLp-FODL几乎相同，不同之处在于我们选择$\mathbf{F}$作为上文引入的时态公式集合，记为$\mathbf{F}^{pl}$。并且我们选择一个不同的标签集合，命名为$\mathbf{L}^{pl}$，其中每个标签被定义为一种形式，用以刻画$\mathbf{L}_W$中配置序列的含义：

$$\sigma_1\sigma_2...\sigma_n\ (n \geq 1),\ \sigma_i \in \mathbf{L}_W\ (1 \leq i \leq n).$$

此外，类似于对$(\mathbf{Pr}^{op})^{fodl}$的选择，我们提出一个$(\mathbf{Pr}^{op})^{pl}$ 规则集合，用于正则程序。在$(\mathbf{Pr}^{op})^{pl}$中，程序转换规则的形式与$(\mathbf{Pr}^{op})^{fodl}$ 中的相同（见表4），不同之处在于：（1）我们将每个配置替换为$\mathbf{L}^{pl}$中的配置，并且（2）我们对赋值有如下规则：

$$\frac{}{\Gamma \Rightarrow (x := e, l) \longrightarrow (\downarrow, l(\sigma_n)^x_e), \Delta}\ {}^{(x:=e)}, \text{ where } l = \sigma_1\sigma_2...\sigma_n\ (n \geq 1)$$

其中，我们将当前结果 $(\sigma_n)^x_e$追加到 $\mathbf{L}_W$中配置的当前序列 $l$的尾部。

考虑一个程序的时序性质示例：

$$\sigma : [\alpha\,;\alpha]\Diamond x > 0,$$

其中 $\sigma = \{x \mapsto -1\}$，$\alpha = (x := x + 1)$。这表示沿着 $\alpha\,;\alpha$的执行路径，$x > 0$ 最终成立。根据 $(\mathbf{Pr}^{op})^{pl}$中的规则，我们得到以下推导：

$$\dfrac{\dfrac{\dfrac{}{\cdot \Rightarrow \sigma\sigma'\sigma'' : \Diamond x > 0}\ ([\downarrow])}{\cdot \Rightarrow \sigma\sigma'\sigma'' : [\downarrow]\Diamond x > 0}\ ([\alpha]R)}{\cdot \Rightarrow \sigma\sigma' : [\alpha]\Diamond x > 0}\ ([\alpha]R),$$

其中 $\sigma' = \{x \mapsto 0\}$, $\sigma'' = \{x \mapsto 1\}$。

$$\frac{l_b : \phi}{l : \mathbf{f}\,\phi}\ (\mathbf{f}) \qquad \frac{\Gamma \Rightarrow l :\ \phi, \Delta \quad \Gamma \Rightarrow l :\ \phi \mathbf{suf} \psi,\ \Delta}{\Gamma \Rightarrow \sigma l :\ \phi \mathbf{suf} \psi,\ \Delta}\ (\text{后缀 } R1) \qquad \frac{\Gamma \Rightarrow l :\ \psi, \Delta}{\Gamma \Rightarrow \sigma l :\ \phi \mathbf{suf} \psi,\ \Delta}\ (\text{后缀 } R2)$$

$$\frac{\Gamma,\ l :\ \phi,\ l :\ \phi \text{后缀}\ \psi \Rightarrow \Delta \quad \Gamma,\ l :\ \psi \Rightarrow \Delta}{\Gamma,\ \sigma l :\ \phi \mathbf{suf}\ \psi \Rightarrow \Delta}\ (\text{后缀 } L)$$

在上述规则中 $, l \in \mathbf{L}_W \mathbf{L}_W^*, \sigma \in \mathbf{L}_W$.

表5：DLp-PL 中带标签时序公式的规则

不同于 $\mathbf{F}^{afo}$ 在 DLp-WP 和 DLp-FODL 中的公式，在 DLp-PL 中，我们可以使用表 5 中所示的以下附加规则，进一步推导出带标签的时态公式（例如 $\sigma\sigma'\sigma'' :\ \ \Diamond x > 0$）。这些规则直接依据算子 **f** 和 **suf**的语义。

然而，对于迭代程序（例如 $\alpha^*$）在 DLp-PL 中的循环推导，需要采用高阶标签结构，并对定义 4.12 中所定义的抽象替换进行适当的实例化。未来工作将对此作进一步讨论，并分析 DLp-PL 的（相对）完备性，以及将其与 PL 的传统理论进行比较。

## 7 DLp的可靠性与完备性分析

本节分析证明系统 $Pr_{dlp}$的可靠性与完备性。目前，我们考虑在对**P** 的程序行为施加限制（定义7.2）的情况下的可靠性。不过，正如下文的详细分析所示，满足该限制的程序集合仍然相当丰富。至于完备性，由于 DLp 并非特定逻辑，通常而言，若不对 DL 的参数施加限制，就无法讨论其完备性。我们转而研究在何种条件下（定义7.13 和7.14）能够得到相对于带标签的非动态公式的完备性结果。

第 7.1 节讨论 $Pr_{dlp}$的可靠性，而第 7.2 节讨论其完备性 ss.

### 7.1 DLp 的条件可靠性

我们首先介绍最小执行路径的概念。

定义 *7.1*。 如果一条执行路径（定义 4.5） $w_1 \ldots w_n (n \geq 1)$满足：对于某个 $1 \leq i < j < n$，不存在两条关系 $w_i \xrightarrow{\alpha_i} \cdot$ 和 $w_j \xrightarrow{\alpha_j} \cdot$ 使得 $w_i = w_j$且 $\alpha_i = \alpha_j$，则称其为“最小”的。

直观地说，在最小执行路径中，不存在两条从同一世界和程序出发的关系。限制条件在以下定义中给出。

定义 *7.2*（终止有限性）。 从世界 $w \in \mathrm{S}$ 和程序 $\alpha \in \mathbf{P}$出发，最小执行路径的数量（形式为： $w \xrightarrow{\alpha} \ldots$）有限。

满足终止有限性的程序实际上构成一个丰富的集合，例如包括所有行为确定的程序，如本文讨论的 while 程序，以及 Esterel、C语言、Java 等编程语言。也存在显然属于这一类别的非确定性程序。例如，具有非确定性转换但状态数量有限的自动机。关于这一限制的更多内容将在我们的未来工作中讨论。

定理 7.3（DL 的条件可靠性p）。如果P 中的程序满足终止有限性性质，那么对于任意带标签公式 $\sigma:\ \phi \in \mathfrak{F}_{ldlp}$, $Pr_{dlp} \vdash (\cdot \Rightarrow \sigma:\ \ \phi)$，均有 $\models \sigma:\ \ \phi$。

**证明可靠性的主要思路**。我们遵循 [15]背后的主要思路，通过反证法证明定理7.3 。关键在于，如果循环证明的结论无效，那么根据 $Pr_{ldlp}$中的所有规则的可靠性（定理 5.3），必定存在一条无效推导路径，其中每个节点均无效，并且其一条渐进轨迹会导出某个良基集（下文将介绍）中的无限下降序列，这违反了良基性的定义（参见 [18]）本身。

下面我们先介绍所依赖的良基关系 $\preceq_m$，然后聚焦于证明定理7.3的主要框架。其他证明细节见附录A。

**良基性与关系** $\preceq_m$。给定一个集合 $S$ 以及其上的偏序关系 $\preceq S$，如果对于任意元素 $a$ 属于 $S$，都不存在无限下降序列：$a \succ a_1 \succ a_2 \succ \ldots$ 属于 $S$，则称 $\preceq$ 是 $S$上的良基关系。集合 $S$ 称为相对于 $\preceq$的良基集。

定义 7.4（关系 $\preceq_m$）。给定由有限执行路径组成的两个有限集$C_1$ 和$C_2$，若满足以下任一条件，则$C_1 \preceq_m C_2$ 有定义：(1) $C_1 = C_2$；或 (2) 集合$C_1$ 可由$C_2$ 通过将$C_2$ 中的一个或多个元素分别替换为有限个元素得到，并且对于每个被替换元素 $tr$，其在$C_1$ 中的替换项 $tr_1, \ldots, tr_n\ (n \geq 1)$ 均为 $tr$ 的真后缀。

在定义7.4中，请注意，我们可以用一个$C$语言$_2$ 元素替换为空执行路径，其长度为0。这样做等价于从$C$语言$_2$中移除一个元素。

命题 7.5. $\preceq_m$ 是一个偏序关系。

命题 7.5 的证明见附录 A。

例 7.6。令 $C_1 = \{tr_1, tr_2, tr_3\}$，其中 $tr_1 =_{df} ww_1w_2w_3w_4, tr_2 =_{df} ww_1w_5w_6w_7$和 $tr_3 =_{df} ww_8$； $C_2 = \{tr'1, tr'_2\}$，其中 $tr'_1 =_{df} w_1w_2w_3w_4, tr'_2 =_{df} w_1w_5w_6w_7$。我们看到， $tr'1$ 是 $tr_1$ 的真后缀，而 $tr'2$ 是 $tr_2$ 的真后缀。 $C_2$ 可以通过将 $C_1$ 中的 $tr_1$ 和 $tr_2$ 分别替换为 $tr'_1$ 和 $tr'2$，并移除 $tr_3$ 得到。因此$C_2 \preceq_m C_1$。由于 $C_1 \neq C_2, C_2 \prec_m C_1$。

命题 7.7. 关系 $\preceq_m$ 是一个良基关系 关系。

我们略去命题 7.7的证明。关系 $\preceq_m$ 只是文献 [18],中引入的“多重集序”的一个特例，并且已在其中证明其为良基关系。直观地说，对于两个集合$C_1$ 和$C_2$，若$C_1 \prec_m C_2$,，则对于集合 $D_{tr}$ 中每条路径$C_1$ 替换 $tr$ 中的一个元素$C_2$,，元素 $D_{tr}$ 的最大长度严格小于 $tr$。由于$C_2$ 是有限的，我们可以看出，这样的替换会减少集合$C_2$中长度达到最大值的路径数量。

**定理** 7.3的证明概要。下面我们略去引理7.9证明的细节，给出证明的主要概要；完整证明见附录A。

遵循上述主要思路，我们首先引入“动态 DLp 公式的执行路径”这一概念。它们是良基关系 $\preceq_m$中的元素。接下来，我们提出引理7.9，它在随后的定理7.3的证明中起关键作用。

定义 7.8（动态公式的执行路径）。给定一个世界 $w \in$ S 和一个动态公式 $\phi$， $\phi$关于 $w$的执行路径 $EX(w, \phi)$ 根据 $\phi$的结构归纳定义如下：

1. $EX(w,[\alpha]F) =_{df} \text{mex}(w,\alpha)$，其中 $F \in \text{F}$；2. $EX(w,[\alpha]\phi_1) =_{df} \text{mex}(w,\alpha) \cup \{tr_1 \cdot tr_2 \mid tr_1 \in \text{mex}(w,\alpha)\ tr_2 \in EX((tr_1)_e,\phi_1)\}$；3. $EX(w,\neg\phi_1) =_{df} EX(w,\phi_1)$；
4. $EX(w,\phi_1 \wedge \phi_2) =_{df} EX(w,\phi_1) \cup EX(w,\phi_2)$。

其中 $mex(w,\alpha) =_{df} \{w...w' \mid w \xrightarrow{\alpha} ... \xrightarrow{\downarrow} w'$ 是某个 $w' \in \text{S}$ 的最短执行路径}是从世界 $w$出发的 $\alpha$ 的所有最短路径的集合。

在定义 7.8中，一个动态公式的执行路径可以由模态序列中属于不同程序的若干执行路径串接而成。正如引理 7.9 （附录 A）的证明所示，这一考虑是必要的，因为动态公式可能包含多个模态（例如 $[\alpha][\beta]\phi$）。这也是我们的证明与 [61]给出的证明之间的主要区别之一。

在下文中，若 $\mathfrak{m} \in \mathbf{M}$ 使节点 $v$ $v$ 无效，则称其为该节点的反例映射。

引理 *7.9.* 在一个循环证明中（其中至少有一条推导路径），设 $(\sigma: \phi, \sigma': \phi')$ 是无效推导路径的某个推导 $(v, v')$ 上推导轨迹的一步，其中 $\phi, \phi' \in \mathfrak{F}_{dlp}$。对于相对于 $\mathfrak{m}$ （$v$的反例映射）的任意集合 $EX(\mathfrak{m}(\sigma)\ \phi)$（其中包含 $\sigma: \phi$），存在 $v'$ 的一个反例映射 $\mathfrak{m}'$ 以及一个集合 $EX(\mathfrak{m}'(\sigma')\ \phi')$（其中包含 $\sigma': \phi'$），使得 $EX(\mathfrak{m}'(\sigma')\ \phi') \preceq_m EX(\mathfrak{m}(\sigma)\ \phi)$。此外，如果 $(\sigma: \phi, \sigma': \phi')$ 是一个渐进步骤，则 $EX(\mathfrak{m}'(\sigma')\ \phi') \prec_m EX(\mathfrak{m}(\sigma)\ \phi)$。

直观地说，引理 7.9 帮助我们发现由良基关系所施加的合适执行路径集合 $\preceq_m$，该关系存在于无效推导路径中的这些集合之间。

根据命题 7.7 和引理7.9，我们给出定理 7.3 的证明如下。

定理 7.3 的证明。令 $v = (\cdot \Rightarrow \sigma: \phi)$。反证法，假设 $\not\models \sigma: \phi$，即 $v$ 无效。那么根据 $Pr_{dlp}$ 中每条规则的可靠性（定理 5.3），存在一条来自 $v$ 的无效推导路径 $p$（其中每个序列均无效）。由于 $Pr_{dlp} \vdash v$（也就是说，形成了一棵循环证明树来证明结论 $v$），令 $\tau_1\tau_2...\tau_k...$ 为 $p$ 上的一条渐进轨迹，其形式为：$v...v_1v_2...v_k...\ (k \geq 1)$，其中每个公式 $\tau_i$ 都属于 $v_i\ (i \geq 1)$。令 $\tau_i =_{df} \sigma_i: \phi_i$。

由于 $v_1$ 无效，设 $\mathfrak{m}_1$ 是它的一个反例映射。根据引理 7.9，从 $EX(\mathfrak{m}_1(\sigma_1)\ \phi_1)$ 出发，存在一个无限集合序列 $EX1, ..., EX_k, ...\ (k \geq 1)$，其中每个 $EX^i =_{df} EX(\mathfrak{m}_i(\sigma_i)\ \phi_i)\ (i \geq 1)$，且 $\mathfrak{m}_i$ 是节点 $v_i$的一个反例映射，并且满足 $EX_1 \succeq_m ... \succeq_m EX_k \succeq_m ...$。此外，由于轨迹 $\tau_1\tau_2...\tau_k...$ 是渐进的（定义 5.5），必定存在无限多个 $j \geq 1$ 使得 $EX_j \succ_m EX_{j+1}$。因此，这就形成了关于 $\prec_m$ 的无限下降序列，违背了关系 $\preceq_m$ 的良基性（命题 7.7 ）。

□

### 7.2 DLp的条件完备性

我们提出两个足以保证 DLp相对完备性的条件：1）DLp 的程序模型始终具有有限表达式（定义 7.13）；2）在推理过程中，它们的循环程序始终行为良好（定义7.14）。循环程序是指在带有标签的符号执行推理序列中最终会到达自身的程序。

下面我们首先介绍这些条件，然后在这些条件下证明 DL 的相对完备性p。我们仅给出证明概要，技术细节见附录A。

定义 *7.10*（程序序列）。给定上下文 $\Gamma$、程序 $\alpha \in \mathbf{P}$ 和标签 $\sigma \in \mathbf{L}$，若存在如下推导序列，则（可能无限的）序列：$\Gamma: (\alpha_1, \sigma_1, \Gamma_1)(\alpha_2, \sigma_2, \Gamma_2)...(\alpha_n, \sigma_n, \Gamma_n)...\ (1 \leq n < \infty, \alpha_1 = \alpha, \sigma_1 = \sigma)$，称为“$\alpha$序列”，是有定义的。提交给 ACM 的稿件

在系统 $Pr_{dlp}$ 中的推导序列如下： $Pr_{dlp} \vdash (\Gamma_1 \Rightarrow (\alpha_1,\ \sigma_1) \longrightarrow (\sigma_2,\ \sigma_2))$, $Pr_{dlp} \vdash (\Gamma_2 \Rightarrow (\alpha_2,\ \sigma_2) \longrightarrow (\sigma_3,\ \sigma_3))$, ..., $Pr_{dlp} \vdash (\Gamma_{n-1} \Rightarrow (\alpha_{n-1},\ \sigma_{n-1}) \longrightarrow (\alpha_n,\ \sigma_n))$, ...，其满足对于所有 $n \geq 2$, $\models (\Gamma_1 \Rightarrow \Gamma)$ 和 $\models (\Gamma_n \Rightarrow \Gamma_{n-1})$ 均成立。

我们称序列 $\Gamma : (\alpha_1, \sigma_1)(\alpha_2, \sigma_2)...(\alpha_n, \sigma_n)...$ 为“核心 $\alpha$ 序列”，如果存在 $\Gamma_1, \Gamma_2, ..., \Gamma_n, ...$ 使得 $\Gamma : (\alpha_1, \sigma_1, \Gamma_1)(\alpha_2, \sigma_2, \Gamma_2)...(\alpha_n, \sigma_n, \Gamma_n)...$ 是一个 $\alpha$ 序列。

直观地说，从 $(\alpha, \sigma)$ 在上下文 $\Gamma$ 下，一个 $\alpha$ 序列是一个推导序列，其中每一步的上下文只能从 $\Gamma$ 得到加强。根据 $Pr_{dlp}$ 相对于 $\mathfrak{F}_{pt}$ 的可靠性（见定义 5.1），一个 $\alpha$ 序列的每个推导实际上都是该程序的一个符号执行步骤。

定义 *7.11*（程序循环序列）。程序 $\alpha \in \mathbf{P}$ 的“$\alpha$-循环序列”是一个 $\alpha$ 序列： $\Gamma : (\alpha_1, \sigma_1, \Gamma_1)...(\alpha_n, \sigma_n, \Gamma_n)$（$n \geq 1$），使得 $\alpha_1 = \alpha_n = \alpha$，并且 $\alpha_i \neq \alpha_j$ 对于任何其他 $\alpha_i$ 和 $\alpha_j$，带有 $1 \leq i < j \leq n$。

例 7.12. 在第 6.3 节定义的实例化理论 DLp-FODL 中，设 $\alpha =_{df} (x := x + 1)$、$\beta =_{df} (y := 0)$、$\sigma = \{x \mapsto t\}$，则对于程序 $\alpha^*; \beta$，我们有一个 $\alpha$ 序列： $\emptyset : (\alpha^*; \beta, \sigma, \emptyset)((\alpha; \alpha^* \cup \mathrm{true}?); \beta, \sigma, \emptyset)(\alpha; \alpha^*; \beta, \sigma, \emptyset)(\alpha^*; \beta, \sigma', \emptyset)$，其中 $\sigma' = \{x \mapsto t + 1\}$。它对应于以下推导：

$$(Pr_{dlp})_{fodl} \vdash (\cdot \Rightarrow (\alpha^*; \beta, \sigma) \longrightarrow ((\alpha; \alpha^* \cup true?), \sigma),$$
$$(Pr_{dlp})_{fodl} \vdash (\cdot \Rightarrow ((\alpha; \alpha^* \cup true?); \beta, \sigma) \longrightarrow (\alpha; \alpha^*; \beta, \sigma)),$$
$$(Pr_{dlp})_{fodl} \vdash (\cdot \Rightarrow (\alpha; \alpha^*; \beta, \sigma) \longrightarrow (\alpha^*; \beta, \sigma'))$$

根据表 4 中相应的规则 $\alpha$ 该序列也是一个循环序列。

定义 *7.13*（表达式有限性性质）。对于程序 $\alpha \in \mathbf{P}$，存在一个自然数 $N_\alpha$，使得在标签 $\sigma \in \mathbf{L}$ 和上下文 $\Gamma$ 下的每个 $\alpha$ 序列中： $\Gamma : (\alpha_1,\ \sigma_1,\ \Gamma_1)...(\alpha_n,\ \sigma_n,\ \Gamma_n)...\ (1 \leq n < \infty, \alpha_1 = \alpha, \sigma_1 = \sigma)$， $\alpha_1, ..., \alpha_n, ...$ 中的不同程序数量不超过 $N_\alpha$。

定义 7.13 意味着程序在运行过程中，经过有限步后最终会达到其自身的极限形式。实际中的大多数程序模型都满足这一性质。但也有例外，例如$\pi$–带有复制算子的演算中的程序（参见 [39]）。

定义 *7.14*（良态循环程序）。如果存在某个标签和上下文对应的“$\alpha$-循环序列”，则称程序 $\alpha \in \mathbf{P}$ 为“循环程序”。

若对于任意标签 $\sigma \in \mathbf{L}$ 和上下文 $\Gamma$，都存在标签 $\sigma' \in \mathbf{L}$、上下文 $\Gamma'$ 以及替换 $\eta$，使得 $\mathbf{L} \to \mathbf{L}$ 满足以下条件，则称循环程序 $\alpha$“行为良好”：

1. $\sigma = \eta(\sigma')$ 和 $\Gamma = \eta(\Gamma')$；2. $\models (\Gamma \Rightarrow \sigma \Downarrow \alpha)$ 蕴含 $\models (\Gamma' \Rightarrow \sigma' \Downarrow \alpha)$；3. 对于每个 $\alpha$-循环序列： $\Gamma' : (\alpha, \sigma', \Gamma_1)...(\alpha, \sigma'', \Gamma_n)$ $(n \geq 1)$，都存在上下文 $\Gamma''$ 和替换 $\xi$，使得 $\mathbf{L} \to \mathbf{L}$，且 $\Gamma'' = \xi(\Gamma')$、$\sigma'' = \xi(\sigma')$ 和 $\models (\Gamma_n \Rightarrow \Gamma'')$。

良态性质描述了循环程序 $\alpha$ 沿其符号执行过程形成回链的“能力”。更具体地说，对于标签 $\sigma \in \mathbf{L}$，可以找到一个合适的标签 $\sigma'$，通过替换可得到 $\sigma$；并且对于每个从 $(\alpha, \sigma')$ 开始的符号执行过程，都可以通过替换回到 $(\alpha, \sigma')$。这里的 $\sigma'$ 在程序逻辑的常规演绎方法中，起着与循环不变式相同的作用。

定理 7.15（动态逻辑的条件完备性p）。若**P** 中的程序满足表达式有限性性质，且其中所有循环程序的行为均良好，则对于任意带标签公式 $\sigma:\ \phi \in \mathfrak{F}_{ldlp}$, $\models \sigma:\ \phi$ 蕴含$Pr_{dlp} \vdash (\cdot \Rightarrow \sigma:\ \phi)$。

请注意，完备性结果相对于带标签的非动态公式的相对性体现在规则 (*Ter*)(表 2) 中。

完备性证明的主要思路。为证明定理 7.15，我们首先将其归约为如下特殊情形：推导形如 $\Gamma \Rightarrow [\alpha]\phi$ 的序列，如引理 7.16 所示。对于这一步，我们采用与 [27] 类似的方法。主要的技术部分是推导序列 $\Gamma \Rightarrow \sigma:\ [\alpha]\phi$。我们对模态数量以及在推导该序列期间可能出现的程序形式的最大数量进行同时归纳。关键的观察是，由于执行 $\alpha$ 时的有限性（引理 7.17）以及其满足良态性质（定义 7.14），从 $\Gamma \Rightarrow \sigma:\ [\alpha]\phi$ 出发的每个非终端推导分支都能够形成回链；进而，这表明 $\Gamma \Rightarrow \sigma:\ [\alpha]\phi$ 的整个推导能够形成循环证明。

引理的证明7.16 的详细内容见附录A 。

引理 7.16。 在与定理*7.15*相同的条件下，对于任意以下形式的有效序列：$\Gamma \Rightarrow \sigma:\ [\langle\alpha\rangle]\phi, Pr_{dlp} \vdash (\Gamma \Rightarrow \sigma:\ [\langle\alpha\rangle]\phi)$。

在引理7.16中，“$[\langle\alpha\rangle]$”仅表示 $[\alpha]$ 或 $\langle\alpha\rangle$。

引理 7.17。 从程序 $\alpha \in$ **P** 及标签 $\sigma \in$ **L** 出发，在上下文 Γ下，核心$\alpha$-循环序列只有有限多个。

证明。由于集合 $\mathbf{Pr}^{op}$ 的有限性，固定程序 $\beta$ 后，从 $(\beta, l)$ 出发、形式为 $\Gamma \Rightarrow (\beta, l) \longrightarrow \ldots$ 的迁移数量存在最大值，对所有标签 $l \in$**L**及上下文 Γ 均如此。根据表达式有限性性质以及程序循环序列的特征，不难得出结论。□

为结束本节，我们给出定理 7.15 的证明。

定理 7.15 的证明。对于带标签公式 $\sigma:\ \phi$，$\phi$ 在语义上等价于合取范式：$C_1 \wedge \ldots \wedge C_n$ $(n \geq 1)$。每个子句 $C_i$ $(1 \leq i \leq n)$ 都是文字的析取：$C_i = l_{i,1} \vee \ldots \vee l_{i,m_i}$，其中 $l_{i,j}$ $(1 \leq i \leq n,\ 1 \leq j \leq m_i)$ 是原子 DLp 公式或其否定。根据表 2 中带标签命题逻辑公式的规则，要证明公式 $\sigma:\ \phi$，只需证明对于每个子句 $C_i$，$\models \sigma:\ C_i$ 蕴含 $Pr_{dlp} \vdash \sigma:\ C_i$。不失一般性，令 $C_i = \psi \vee [\alpha]\phi$。于是，只需证明 $Pr_{dlp} \vdash (\sigma:\ \neg\psi \Rightarrow \sigma:\ [\alpha]\phi)$。但这正是引理 7.16 的一个特例。□

## 8 相关工作

**匹配逻辑及其变体**。基于程序的操作语义对程序进行推理，这一思想并不新鲜。过去十年中的先前研究（如 [17, 52, 53, 56] ）使用基于重写逻辑 [37] 的理论探讨了这一问题。匹配逻辑 [52] 以模式和模式匹配为基础。其基本形式——可达性规则 $\varphi \Rightarrow \varphi'$（此处的含义与本文中的用法 $\Rightarrow$ 不同）——刻画了在给定的模式可达性系统中，模式 $\varphi'$ 是否可从模式 $\varphi$ 到达。基于匹配逻辑，通过增强可达性规则的表达能力，发展出了单路径和全路径可达性逻辑 [53, 56] 。通过在匹配逻辑中加入最小不动点 $\mu$绑定符，提出了一种更强大的匹配 $\mu$-逻辑 [17] 。提交给 ACM 的稿件

在这些理论中，“模式”是更一般的结构。因此，要编码 DL $[\alpha]\phi$ 的动态形式$\mathfrak{p}$，还需要额外的工作和程序变换。另一方面，像 DL$\mathfrak{p}$ 这样的动态逻辑通过其模态 $[\cdot]$ 和$\langle\cdot\rangle$，提供了一种更直接的方式来表达和推理复杂的前后状态性质及程序时序性质。在表达能力方面，当程序具有非确定性时（即从某个世界和程序出发存在多条执行路径），匹配逻辑和单路径可达性逻辑无法刻画动态逻辑中模态 $[\cdot]$ 的语义。我们推测，匹配 $\mu$-逻辑可以编码 DL$\mathfrak{p}$，因为已有说法称其能够编码传统动态逻辑（参见 [17]）。

**基于集合论的一般框架**。 [40] 提出了一种基于余归纳的一般程序验证框架。按照本文的术语，程序规约 $\sigma$： $[\alpha]\phi$ 可以表示为一对（（$\alpha$， $\sigma$） $P$（$\phi$））， 其中 [40], 中的 $P$（$\phi$）是一个程序状态集合，用于刻画公式 $\phi$的语义。研究者设计了一种方法，根据（$\alpha$， $\sigma$）的操作语义，以余归纳的方式推导程序规约。继 [40], 之后，[36] 也提出了一个用于程序推理的一般框架，但采用的是大步操作语义。与 [40] 和 [36] 中直接建立在数学集合论之上的框架不同，DL$\mathfrak{p}$ 采用逻辑形式，并基于循环演绎方法，而非余归纳。从表达能力来看，DL$\mathfrak{p}$ 中模态 $\cdot$ 的含义无法在 [40]的框架中表达。

**更新**。文献 [6, 7, 44] 中采用的“更新”结构是变量和项的“延迟替换”。事实上，通过相应地选择合适的标签映射，它们可以定义为 DL$\mathfrak{p}$ 中更一般的标签结构的一个特例。

**基于循环证明方法的逻辑**。DL$\mathfrak{p}$ 的证明系统依赖于循环证明理论，该理论最初出现于 [57]，后来在不同的逻辑中得到发展，例如 [15, 16], ，以及较新的工作，如 [3, 33, 59]。 [34] 为 $\mu$-演算提出了一个完备的循环证明系统，其表达能力涵盖了 PDL [23] 。在 [20], 中，作者为 PDL 提出了一个完备的带标签的循环证明系统。 [20, 34] 中的两种逻辑都是命题逻辑的，无法用于证明特定定义域中的许多有效公式，例如我们示例中所示的数论算术一阶公式。DL$\mathfrak{p}$ 公式 $\sigma$ 的带标签形式 $[\alpha]\phi$ 受到 [20], 的启发，其中标签只是传统克里普克结构中世界的一个变量。另一方面，DL$\mathfrak{p}$ 中的标签允许使用来自实际程序配置的任意项。

**动态逻辑的推广**。近期有一些关于推广动态逻辑理论的研究 [2,30, 60]。 [30] 提出了一种通用动态逻辑，允许 PDL 的程序模型采用任意形式，而不局限于正则程序。程序的语义由一组所谓的“基于交互”的行为给出，这与此处的程序转移非常相似。然而，该研究仅关注逻辑理论的构建，并未提出相应的证明系统。在 [2], 中，研究了 PDL 的一个操作式版本（即 OPDL）。其中，如果 $\alpha \xrightarrow{a} \beta$ 是通过执行动作 $a$ 得到的一次迁移，那么动态公式 $[\alpha]\phi$ 的证明可以化归为公式 $[a][\beta]\phi$ 的证明。与 [20], 类似，研究者为 OPDL 构建了一个完备的非良基证明系统。虽然 [2] 声称 OPDL 可以适用于任意程序模型，但其理论分析仅在命题层面进行，且只针对正则程序。[60] 提出了异构动态逻辑（HDL），这是一个可对不同动态逻辑理论进行比较并加以联合使用的理论框架。与 [60] 对不同理论的集成进行系统分析不同，我们的工作（以及 [61]）旨在促进针对不同程序的基于操作语义的推理。这促成了标签的引入，以及循环推理在 DL$\mathfrak{p}$ 中的发展，二者构成了关键贡献；而提升过程则作为一种“辅助技术”，用于弥补核心证明系统的不足。 [60] 的成果为我们未来在 DL$\mathfrak{p}$ 中分析提升后理论的完备性及其他性质提供了详尽指南。

## 9 结论与未来工作

本文提出了一种基于动态逻辑的全新验证框架，用于根据程序的操作语义对程序进行推理。我们主要构建了 DLp 的理论，并分析了其在特定条件下的可靠性与完备性。通过示例和案例研究，我们展示了这种形式化方法在程序推理不同方面的潜在用途。

未来工作将聚焦于两个方面。在理论方面，我们关注能否进一步放宽条件，以证明 DLp的可靠性与完备性。这对于了解我们的框架能否适用于更复杂的模型（如混合系统或概率系统）非常重要。我们将进一步研究实例化理论 DLp-PL，这是迄今构建的颇具前景的过程逻辑一阶版本；同时也将研究 DLp-SP。在实践方面，我们正在 Rocq 中对 DLp 进行全面的机械化 [13]。目前，我们已成功深度嵌入 DLp (参见 [1])。为探索 DLp 在实践中的潜在应用，我们正在尝试将 DLp 用于对不同的程序模型（如 Esterel、Rust 等）进行规范描述和推理。

致谢。 本工作部分得到了南京航空航天大学新教师启动基金（编号：90YAT24003）和国家自然科学基金委员会面上项目（编号：62272397）的资助。

## 参考文献

[1] [n.d.]。https://github.com/yrz5a/Coq-DLp.git。[2] Matteo Acclavio、Fabrizio Montesi 和 Marco Peressotti。2024。关于命题动态逻辑与并发。arXiv *[cs.LO]*（2024年3月）。[3] Bahareh Afshari、Sebastian Enqvist 和 Graham E Leigh。2022。一阶 μ演算的循环证明。IGPL 逻辑期刊 32，1（2022年08月），1–34。arXiv: https://academic.oup.com/jigpal/article-pdf/32/1/1/56586573/jzac053.pdf[4] Wolfgang Ahrendt、Bernhard Beckert、Richard Bubel、Reiner Hähnle 和 Mattias Ulbrich。2025。动态逻辑的多种用途。载于《计算机科学讲义》。Springer Nature Switzerland, Cham，56–82。[5] Andrew W. Appel、Robert Dockins 等。2014。经认证的编译器的程序逻辑。剑桥大学出版社。[6] Bernhard Beckert 和 Daniel Bruns。2013。带轨迹语义的动态逻辑。载于 CADE 2013。Springer Berlin Heidelberg，315–329。[7] Bernhard Beckert、Vladimir Klebanov 和 Benjamin Weiß。2016。Java 动态逻辑。Springer International Publishing，49–106。[8] Mario R.F. Benevides 和 L. Menasché Schechter。2010。基于 $\pi$演算的并发程序命题动态逻辑。载于 M4M 2009。爱思唯尔，49–64。[9] Mario R. F. Benevides 和 L. Menasché Schechter。2008。用于 CCS 程序的命题动态逻辑。载于《逻辑、语言、信息与计算》。Springer Berlin Heidelberg，柏林，海德堡，83–97。[10] Gérard Berry。1989。使用 Esterel v3 编写数字手表程序。技术报告 RR-1032。[11] Gérard Berry 和 Laurent Cosserat。1985。ESTEREL 同步编程语言及其数学语义。载于《并发研讨会》。Springer Berlin Heidelberg，柏林，海德堡，389–448。[12] Gérard Berry 和 Georges Gonthier。1992。Esterel 同步编程语言：设计、语义与实现。计算机程序科学 19，2（1992），87–152。[13] Yves Bertot 和 Pierre Castéran。2004。交互式定理证明与程序开发——Coq'Art：归纳构造演算。施普林格。1–472页。[14] Denis Bogdanas 和 Grigore Roşu。2015。K-Java：Java 的完备语义。载于第42届ACM SIGPLAN-SIGACT编程语言原理年会论文集（印度孟买）（POPL '15）。计算机协会，纽约，纽约州，美国，445–456。https://doi.org/10.1145/2676726.2676982[15] James Brotherston、Richard Bornat 和 Cristiano Calcagno。2008。分离逻辑中的程序终止循环证明。SIGPLAN 通讯 43，1（2008），101–112。[16] James Brotherston 和 Alex Simpson。2007。归纳与无穷递降的完备序列演算。载于 LICS 2007。51–62。[17] X. Chen 和 G. Rosu。2019。匹配 *mu*逻辑。载于 LICS 2019。IEEE 计算机学会，1–13。[18] Nachum Dershowitz 和 Zohar Manna。1979。使用多重集序证明终止性。载于《自动机、语言与编程》。Springer Berlin Heidelberg，柏林，海德堡，188–202。[19] Canh Minh Do、Tsubasa Takagi 和 Kazuhiro Ogata。2024。基于并发动态量子逻辑的自动量子协议验证。ACM 软件工程与方法学汇刊（2024年12月）。已正式接收。[20] Simon Docherty 和 Reuben N. S. Rowe。2019。用于命题动态逻辑的非良基标记证明系统。载于 TABLEAUX 2019。Springer International Publishing，335–352。提交给 ACM 的稿件

[21] Chucky Ellison 和 Grigore Rosu。2012。C语言的可执行形式语义及其应用。载于第39届ACM SIGPLAN-SIGACT编程语言原理年会论文集（美国宾夕法尼亚州费城）（POPL '12）。计算机协会，纽约，纽约州，美国，533–544。https://doi.org/10.1145/2103656.2103719[22] Yishai A. Feldman 和 David Harel。1984。一种概率动态逻辑。计算机系统科学杂志 28，2（1984），193–215。[23] Michael J. Fischer 和 Richard E. Ladner。1979。正则程序的命题动态逻辑。计算机系统科学杂志 18，2（1979），194–211。[24] Manuel Gesell 和 Klaus Schneider。2012。同步语言验证的霍尔演算。载于 PLPV 2012（美国宾夕法尼亚州费城）。计算机协会，37–48。[25] Jens Oliver Gutsfeld、Markus Müller-Olm 和 Christoph Ohrem。2020。用于超性质的命题动态逻辑。载于第31届国际并发理论会议（CONCUR）。50:1–50:22。[26] Nicolas Halbwachs、Fabienne Lagnier 和 Pascal Raymond。1993。同步观察器与反应式系统的验证。载于第三届方法论与软件技术国际会议：代数方法论与软件技术（AMAST '93）论文集。施普林格出版社，柏林，海德堡，83–96。[27] David Harel。1979。一阶动态逻辑。《计算机科学讲义》（LNCS），第68卷。施普林格。[28] David Harel、Dexter Kozen 和 Rohit Parikh。1982。过程逻辑：表达能力、可判定性与完备性。计算机系统科学杂志 25，2（1982），144–170。[29] David Harel、Dexter Kozen 和 Jerzy Tiuryn。2000。动态逻辑。麻省理工学院出版社。[30] Rolf Hennicker 和 Martin Wirsing。2019。通用动态逻辑及其在基于交互的系统中的应用。Springer International Publishing，172–187。[31] C. A. R. Hoare。1969。计算机编程的公理基础。ACM 通讯 12，10（1969），576–580。[32] Benjamin Icard。2024。智能领域的信息评估动态逻辑。arXiv:2405.19968 [cs.LO] https://arxiv.org/abs/2405.19968[33] Eddie Jones、C.-H. Luke Ong 和 Steven Ramsay。2022。CycleQ：循环等式推理的高效基础。载于第43届ACM SIGPLAN编程语言设计与实现国际会议论文集（美国加利福尼亚州圣迭戈）（PLDI 2022）。计算机协会，纽约，纽约州，美国，395–409。[34] Natthapong Jungteerapanich。2009。模态 $\mu$演算的表列系统。载于 TABLEAUX 2009。Springer Berlin Heidelberg，220–234。[35] Dexter Kozen。1985。一种概率 PDL。计算机系统科学杂志 30，2（1985），162–178。[36] Ximeng Li、Qianying Zhang、Guohui Wang、Zhiping Shi 和 Yong Guan。2021。基于大步语义统一推理迭代与递归。载于 SETTA 2021。Springer International Publishing，61–80。[37] José Meseguer。2012。重写逻辑的二十年。逻辑与代数编程杂志 81，7（2012），721–781。[38] R. Milner。1982。通信系统演算。施普林格出版社，柏林，海德堡。[39] Robin Milner、Joachim Parrow 和 David Walker。1992。移动进程演算，I。信息与计算 100，1（1992），1–40。[40] Brandon Moore、Lucas Peña 和 Grigore Rosu。2018。通过余归纳进行程序验证。载于 ESOP 2018。Springer International Publishing，589–618。[41] Oleg Mürk、Daniel Larsson 和 Reiner Hähnle。2007。KeY-C：用于验证 C语言程序的工具。载于《自动演绎——CADE-21》。Springer Berlin Heidelberg，柏林，海德堡，385–390。[42] Peter W. O'Hearn。2019。错误逻辑。ACM 程序设计语言会议录 4，POPL，文章10（2019），32页。[43] Raúl Pardo、Einar Broch Johnsen 等。2022。概率守卫命令语言中程序的规约逻辑。载于 ICTAC 2022（格鲁吉亚第比利斯）。施普林格出版社，369–387。[44] André Platzer。2007。用于验证参数化混合系统的微分动态逻辑。。载于 TABLEAUX 2007（2007-09-20）。Springer Berlin Heidelberg，216–232。[45] André Platzer。2018。信息物理系统的逻辑基础。施普林格。[46] André Platzer 和 Jan-David Quesel。2008。KeYmaera：用于混合系统的混合定理证明器（系统描述）。载于《自动推理》。Springer Berlin Heidelberg，柏林，海德堡，171–178。[47] G. D. Plotkin。1981。操作语义的结构化方法。技术报告 DAIMI FN-19。奥胡斯大学。[48] Dumitru Potop-Butucaru、Stephen A. Edwards 和 Gérard Berry。2010。编译 Esterel。Springer New York，纽约州纽约。[49] Vaughan R. Pratt。1976。关于弗洛伊德-霍尔逻辑的语义考量。载于 IEEE计算机科学基础年会（FOCS）。IEEE 计算机学会，109–121。[50] Wolfgang Reif。1995。Kiv方法的软件验证。Springer Berlin Heidelberg，柏林，海德堡，339–368。[51] J.C. Reynolds。2002。分离逻辑：一种用于共享可变数据结构的逻辑。载于第17届IEEE计算机科学逻辑年会论文集。55–74。[52] Grigore Roşu 和 Andrei Ştefănescu。2012。迈向操作语义与公理语义的统一理论。载于 ICALP 2012。Springer Berlin Heidelberg，351–363。[53] Grigore Rosu、Andrei Stefanescu、Stefan Ciobâcá 和 Brandon M. Moore。2013。单路径可达性逻辑。载于 LICS 2013。358–367。[54] K. Rustan 和 M. Leino。2007。面向对象软件的验证。KeY方法。《计算机科学讲义》（LNCS），第4334卷。施普林格。[55] Klaus Schneider 和 Jens Brandt。2017。Quartz：用于基于模型的反应式嵌入式系统设计的同步语言。施普林格荷兰，Dordrecht，1–30。提交给 ACM 的稿件

[56] Andrei Ştefănescu、Ştefan Ciobâcă 等。2014。全路径可达性逻辑。载于 RTA-TLCA 2014。Springer International Publishing, 425–440。[57] Colin Stirling 和 David Walker。1991。模态 μ 演算中的局部模型检测。Theor. Comput. Sci. 89, 1 (1991), 161–177。[58] Tsubasa Takagi、Canh Minh Do 和 Kazuhiro Ogata。2024。在动态量子逻辑中自动化量子程序验证。载于《动态逻辑：新趋势与应用》。Springer Nature Switzerland, Cham, 68–84。[59] Gadi Tellez 和 James Brotherston。2020。通过循环证明自动验证指针程序的时序性质。自动推理杂志 64, 3 (2020), 555–578。[60] Samuel Teuber、Mattias Ulbrich、André Platzer 和 Bernhard Beckert。2025。异构动态逻辑：模程序理论可证性。arXiv [cs.LO]（2025年7月）。[61] Yuanrui Zhang。2025。参数化动态逻辑——迈向通用程序规约与验证的循环逻辑框架。arXiv:2404.18098 [cs.LO] https://arxiv.org/abs/2404.18098[62] Yuanrui Zhang、Frédéric Mallet 和 Zhiming Liu。2022。一种基于定理证明、用于验证同步模型的动态逻辑。Front. Comput. Sci. 16, 4 (2022), 3页。[63] Yuanrui Zhang、Hengyang Wu、Yixiang Chen 和 Frédéric Mallet。2021。一种基于时钟的动态逻辑，用于验证同步系统中的 CCSL 规范。计算机程序科学 203 (2021), 102591。[64] Noam Zilberstein、Derek Dreyer 和 Alexandra Silva。2023。结果逻辑：正确性与错误性推理的统一基础。Proc. ACM Program. Lang. 7, OOPSLA1, 文章93 (2023年4月), 29页。

## A 其他命题与证明

引理 A.1. 给定带标签公式 $\tau_1$，……，$\tau_n$，$\tau$，对任意满足 $\mathfrak{m} \models \Gamma$ 的标签映射 $\mathfrak{m} \in \mathbf{M}$，如果 $\mathfrak{m} \models \tau_1$ 且…… 且 $\mathfrak{m} \models \tau_n$ 蕴含 $\mathfrak{m} \models \tau$，则规则

$$\frac{\Gamma \Rightarrow \tau_1, \Delta \quad \ldots \quad \Gamma \Rightarrow \tau_n, \Delta}{\Gamma \Rightarrow \tau, \Delta}$$

对任意上下文 Γ、Δ 都是可靠的。

引理 A.1的证明。 假设 $\Gamma \Rightarrow \tau_1$，$\Delta$，……，$\Gamma \Rightarrow \tau_n$，$\Delta$ 是有效的，对任意 $\mathfrak{m} \in \mathbf{M}$ 满足 $\mathfrak{m} \models \Gamma$，令 $\mathcal{K}$ $\mathfrak{m} \not\models \tau'$ 对所有$\tau' \in \Delta$，我们需要证明 $\mathfrak{m} \models \tau$。由假设可得 $\mathfrak{m} \models \tau_1$，……，$\mathfrak{m} \models \tau_n$。于是 $\mathfrak{m} \models \tau$ 是直接结果。 □

引理 A.2. 给定带标签公式 $\tau_1$，……，$\tau_n$，$\tau$，对任意满足 $\mathcal{K}$ $\mathfrak{m} \models \Gamma$ 的标签映射 $\mathfrak{m} \in \mathbf{M}$，如果 $\mathfrak{m} \models \tau$ 蕴含 $\mathfrak{m} \models \tau_1$ 或 …… 或 $\mathfrak{m} \models \tau_n$ 之一，则规则

$$\frac{\Gamma, \tau_1 \Rightarrow \Delta \quad \ldots \quad \Gamma, \tau_n \Rightarrow \Delta}{\Gamma, \tau \Rightarrow \Delta}$$

对任意上下文 Γ、Δ 都是可靠的。

引理的证明 A.2。 假设 Γ、$\tau_1 \Rightarrow \Delta$、……、Γ、$\tau_n \Rightarrow \Delta$ 均有效。对于任意 $\mathfrak{m} \in \mathbf{M}$ 满足 $\mathfrak{m} \models \Gamma$，若 $\mathfrak{m} \models \tau$，根据假设，$\mathfrak{m} \models \tau_i$ 对于某个 $1 \le i \le n$ 成立。根据 Γ、$\tau_i \Rightarrow \Delta$、$\mathfrak{m} \models \Delta$ 的有效性。由于 $\mathfrak{m}$ 的任意性，我们知道Γ、$\tau \Rightarrow \Delta$ 有效。 □

**定理 5.3的内容：** 表 $Pr_{ldlp}$ 中的每条规则2 都是可靠的。

下面我们只证明规则 ($[\alpha]R$)、($[\alpha]L$) 和 ($Sub$)的可靠性。其他规则可根据 DLp 公式的语义（定义 4.6）以类似方式证明。

定理 5.3 的证明。对于规则 ($[\alpha]R$)，根据引理 A.1，只需证明：对于任意满足 $\mathfrak{m} \models \Gamma$ 的 $\mathfrak{m} \in \mathrm{M}$，若 $\mathfrak{m} \models \sigma' : [\alpha']\phi$ 对所有 $(\alpha', \sigma') \in \Phi$ 都成立，则 $\mathfrak{m} \models \sigma : [\alpha]\phi$。对于某个 $\alpha_0 \in \mathrm{P}$ 和 $w_0 \in \mathrm{S}$ 的任意关系 $\mathfrak{m}(\sigma) \xrightarrow[\alpha_0]{\alpha} w_0$，根据定义 5.1 的第 1 项，存在一个标签 $\sigma_0 \in \mathrm{L}$，使得 $\mathfrak{m}(\sigma_0) = w_0$，且 $\models (\Gamma \Rightarrow (\alpha, \sigma) \longrightarrow (\alpha_0, \sigma_0))$。因此，$Pr_{dlp} \vdash (\Gamma \Rightarrow (\alpha, \sigma) \longrightarrow (\alpha_0, \sigma_0))$（根据定义 5.1 的第 3 项）。于是 $(\alpha_0, \sigma_0) \in \Phi$，根据假设，K $\mathfrak{m} \models \sigma_0 : [\alpha_0]\phi$。由于 $\alpha_0$ 和 $\sigma_0$ 的任意性，根据公式 $[\alpha]\phi$ 的语义（定义 4.6），我们有 $\mathfrak{m} \models \sigma : [\alpha]\phi$。提交给 ACM 的稿件

对于规则 ( $[\alpha]L$ 2, $\mathfrak{m} \in$ $\mathfrak{m} \models \Gamma$ $\mathfrak{m} \models \sigma$ $[\alpha]\phi$ $\mathfrak{m} \models$ ), $\sigma'$ 根据引理 A.，只需证明：对于任意满足 的 M，$\mathfrak{m}$ $\sigma'$ $\xrightarrow{\alpha}$ ... $\xrightarrow{\downarrow}$ $w$ $\alpha'$ $w \in$ 若：，则：$[\alpha']\phi$。对于某个 S 的任意执行路径 ( ) $'/\cdot\cdot/$，根据旁推导 $Pr_{dlp} \vdash (\Gamma \Rightarrow (\alpha,$ $\sigma)$ $\longrightarrow (\alpha', \sigma')$ $\Delta)$ 的可靠性， $\mathfrak{m}(\sigma)$ $\alpha/\alpha'\longrightarrow$ $\mathfrak{m}(\sigma')$ 是 K 上的一个关系，因此 $\mathfrak{m}(\sigma)$ $\alpha/\alpha'\longrightarrow$ $\mathfrak{m}(\sigma')$ $\xrightarrow{\alpha}$ ... $'/\cdot\cdot/\xrightarrow{\downarrow}$ $w$ 是 $\alpha$ 的一条执行路径。根据动态公式的语义（定义 4.6），可得所需结果。

对于规则 (Sub)，我们需要证明，$\Gamma \Rightarrow \Delta$ 的有效性蕴含 $\mathrm{Sub}(\Gamma) \Rightarrow \mathrm{Sub}(\Delta)$ 的有效性。对于任意满足 $\mathfrak{m} \models \mathrm{Sub}(\Gamma)$ 的 $\mathfrak{m} \in \mathrm{M}$，根据定义 4.12，存在一个 $\mathfrak{m}'$，使得对于任意公式 $\sigma$： $\phi \in \Gamma \cup \Delta, \mathfrak{m}'(\sigma) = \mathfrak{m}(\mathrm{Sub}(\sigma))$。因此 K $\mathfrak{m}' \models \Gamma$。由于 $\Gamma \Rightarrow \Delta$ 有效，K $\mathfrak{m}' \models \sigma_0$：对于某个 $\sigma_0$， $\phi_0$： $\phi_0 \in \Delta$。由 $\mathfrak{m}'(\sigma_0) = \mathfrak{m}(\mathrm{Sub}(\sigma_0))$, $\mathfrak{m} \models \mathrm{Sub}(\sigma_0)$： $\phi_0$，且 $\mathrm{Sub}(\sigma_0)$： $\phi_0 \in \mathrm{Sub}(\Delta)$。由 $\mathfrak{m}$ 的任意性可知，$\mathrm{Sub}(\Gamma) \Rightarrow \mathrm{Sub}(\Delta)$ 有效。

□

**命题内容7.5：** $\preceq_m$ 是一个偏序关系。

在证明命题 7.5 之前，我们先回顾偏序关系的概念。集合 $S$ 上的关系 $\preceq$ 是偏序关系，当且仅当满足以下性质：(1) 自反性。 $t \preceq t$ 对每个 $t \in S$均成立。(2) 反对称性。对于任意 $t_1$、 $t_2 \in S$， 如果 $t_1 \preceq t_2$ 且 $t_2 \preceq t_1$， 那么 $t_1$ 和 $t_2$ 是 $S$中的同一个元素，记作 $t_1 = t_2$。(3) 传递性。对于任意 $t_1$、 $t_2$、 $t_3 \in S$， 如果 $t_1 \preceq t_2$ 且 $t_2 \preceq t_3$， 那么 $t_1 \preceq t_3$。 $t_1 \prec t_2$ 定义为 $t_1 \preceq t_2$ 和 $t_1 \neq t_2$。回想一下，我们用 $\preceq_s$ 表示执行路径之间的后缀关系。 $\preceq_s$ 显然是一种偏序关系。

命题传递证明由定义 7.45中所证性质的以及关系 $\prec_s$ 的传递性证明。下面我们只证明反对称性。

对于任意有限集 $D_1$， $D_2$ 的有限路径，若 $D_1 \preceq_m D_2$ 但 $D_1 \neq D_2$， 令 $f_{D_1,\ D_2}$： $D_1 \to D_2$ 为如下定义的函数：对于任意 $tr \in D_1$，或者（1）$f_{D_1,\ D_2}(tr) \approx_s tr$；或者（2） $tr$是被替换元素 $f_{D_1,\ D_2}(tr)$ 在$D_2$ 中的一个替换项，且 $tr \prec_s f_{D_1,\ D_2}(tr)$。

为证明反对称性，假设 $\mathrm{C}_1 \preceq_m \mathrm{C}_2$ 且 $\mathrm{C}_2 \preceq_m \mathrm{C}_1$， 但 $\mathrm{C}_1 \neq \mathrm{C}_2$。令 $tr \in \mathrm{C}_1$ 但 $tr \notin \mathrm{C}_2$。由 $\mathrm{C}_1 \preceq_m \mathrm{C}_2$, 可得 $tr \prec_s f_{C_1}$, C2 $(tr)$。若 $f_{C_1}$， C2 $(tr)$ $\in \mathrm{C}_1$,，则必有 $f_{C_1}$， C2 $(tr) \prec_m f_{C_1}$， C2 $(f_{C_1}$， C2 $(tr))$，因为 $f_{C_1}$， C2 $(tr)$ 已经是 $\mathrm{C}_2$ 中的一个被替换元素。若 $f_{C_1}$， C2 $(tr)$ $\notin \mathrm{C}_1$,，则由 $\mathrm{C}_2 \preceq_m \mathrm{C}_1$, 可得 $f_{C_1}$， C2 $(tr) \prec_s f_{C_2}$， C1 $(f_{C_1}$， C2 $(tr))$。通过继续考察 $f_{C_1}$， C2 $(f_{C_1}$， C2 $(tr))$ 或 $f_{C_2}$， C1 $(f_{C_1}$， C2 $(tr))$ 以及更远的元素，我们实际上可以构造出一个如下的无限下降序列：$tr \prec_s f_{C_1}$， C2 $(tr) \prec_s f_{C_1}$， C2 $(f_{C_1}$， C2 $(tr)) \prec_s \ldots$，相对于关系 $\preceq_s$而言，这违反了其良基性。因此，唯一可能的情况是 $\mathrm{C}_1 = \mathrm{C}_2$。

□

引理 7.9 的内容：在循环证明中（其中至少存在一条推导路径），设 $(\sigma$： $\phi, \sigma'$： $\phi')$ 是无效推导路径的推导 $(v, v')$ 上推导轨迹的一步，其中 $\phi, \phi' \in \mathfrak{F}_{dlp}$。对于相对于 $v$ 的反例映射 $\mathfrak{m}$ 的任意 $\sigma$: $\phi$ 集合 EX$(\mathfrak{m}(\sigma)$ $\phi)$, 存在 $v'$ 的反例映射 $\mathfrak{m}'$ 以及一个 $\sigma'$: $\phi'$ 集合 EX$(\mathfrak{m}'(\sigma')$ $\phi')$，使得 EX$(\mathfrak{m}'(\sigma')$ $\phi')$ $\preceq_m$EX$(\mathfrak{m}(\sigma)$ $\phi)$。此外，若 $(\sigma$: $\phi, \sigma'$: $\phi')$ 是渐进步骤，则 EX$(\mathfrak{m}'(\sigma')$ $\phi')$ $\prec_m$EX$(\mathfrak{m}(\sigma)$ $\phi)$。

引理的证明 7.9。 考虑从节点 $v$出发的规则应用，我们仅考虑它是规则 $([\alpha]R)$、规则 $([\alpha]L)$、规则 $(Sub)$以及规则 $(\wedge R)$的实例，且所考虑的 CP 对的首元素是它们的目标公式的情形。

规则（$[\alpha]R$）的情形：如果从节点 $v$应用规则（$[\alpha]R$），其中$\tau =_{df} \sigma$： $[\alpha]\phi$目标公式，令 $\tau'$ =（$\sigma'$： $[\alpha']\phi$）对于某个 $\alpha'$和 $\sigma'$成立，则 $v$ =（$\Gamma \Rightarrow \tau, \Delta$）以及 $v'$ =（$\Gamma \Rightarrow \tau', \Delta$）。（在此情形下，（$v, v'$）已经是一个渐进步骤。）由于 $\mathfrak{m}$是 $v$ 的反例， $\mathfrak{m} \not\models \tau$，因此 mex（$\mathfrak{m}$（$\sigma$） $\alpha$） $\neq \emptyset$。于是 EX（$\mathfrak{m}$（$\sigma$） $[\alpha]\phi$） $\neq \emptyset$。根据规则（$[\alpha]R$）的可靠性以及 $Pr_{dlp} \vdash \Gamma \Rightarrow$（$\alpha, \sigma$） $\to$（$\alpha', \sigma'$） $\Delta$这一假设，对于每条路径 $tr = \mathfrak{m}$（$\sigma'$）$s_1 \ldots s_n \in$EX（$\mathfrak{m}$（$\sigma'$） $[\alpha']\phi$）（$n \geq 0$），

路径 $\mathfrak{m}(\sigma)\ tr \in \mathrm{EX}(\mathfrak{m}(\sigma)\ [\alpha]\phi)$ 将 $tr$ 作为其真后缀。根据定义 7.2，$\mathrm{EX}(\mathfrak{m}(\sigma)\ [\alpha]\phi)$ 和 $\mathrm{EX}(\mathfrak{m}(\sigma')\ [\alpha']\phi)$ 也都是有限的。因此 $\mathrm{EX}(\mathfrak{m}(\sigma')\ [\alpha']\phi) \prec_m \mathrm{EX}(\mathfrak{m}(\sigma)\ [\alpha]\phi)$。

规则 $([\alpha]L)$ 的情形：若从节点 $\nu$ 应用规则 $([\alpha]L)$，并附带 $\tau =_{df} \sigma\colon\ [\alpha]\phi$ 目标公式，令 $\tau' = (\sigma' :[\alpha']\phi)$，其中 $\alpha'$ 和 $\sigma'$，于是 $\nu = (\Gamma \Rightarrow \tau, \Delta)$ 且 $\nu' = (\Gamma \Rightarrow \tau', \Delta)$。根据规则 $([\alpha]L)$ 的可靠性，以及假设 $Pr_{dlp} \vdash \Gamma \Rightarrow (\alpha, \sigma) \rightarrow (\alpha', \sigma')\ \Delta$，对于每条路径 $tr = \mathfrak{m}(\sigma')s_1 \ldots s_n \in \mathrm{EX}(\mathfrak{m}(\sigma')\ [\alpha']\phi)\ (n \geq 0)$，路径 $\mathfrak{m}(\sigma)tr \in \mathrm{EX}(\mathfrak{m}(\sigma)\ [\alpha]\phi)$ 都以 $tr$ 作为其真后缀。根据定义 7.2，$\mathrm{EX}(\mathfrak{m}(\sigma)\ [\alpha]\phi)$ 和 $\mathrm{EX}(\mathfrak{m}(\sigma')\ [\alpha']\phi)$ 也都是有限的。因此，$\mathrm{EX}(\mathfrak{m}(\sigma')\ [\alpha']\phi) \preceq_m \mathrm{EX}(\mathfrak{m}(\sigma)\ [\alpha]\phi)$，其中等价关系 $=$ 仅在 $\mathrm{EX}(\mathfrak{m}(\sigma)\ [\alpha]\phi) = \emptyset$ 时成立。当 $(\tau, \tau')$ 是渐进的，也就是说，我们还有推导 $Pr_{dlp} \vdash (\Gamma \Rightarrow \alpha \Downarrow \sigma, \Delta)$，则 $\mathrm{mex}(\mathfrak{m}(\sigma)\ \alpha) \neq \emptyset$。这意味着 $\mathrm{EX}(\mathfrak{m}(\sigma)\ [\alpha]\phi) \neq \emptyset$。因此，$\mathrm{EX}(\mathfrak{m}(\sigma')\ [\alpha']\phi) \prec_m \mathrm{EX}(\mathfrak{m}(\sigma)\ [\alpha]\phi)$。

规则 (Sub) 的情形：若从节点 $\nu$ a应用替换规则 (Sub)，令 $\tau = \mathrm{Sub}(\sigma)\colon\ \phi$ 为$\nu$的目标公式，则 $\tau' = \sigma\colon\ \phi$。根据定义 4.12，对于标签映射 $\mathfrak{m}$，存在一个 $\mathfrak{m}'$ 使得 $\mathfrak{m}'(\sigma) = \mathfrak{m}(\mathrm{Sub}(\sigma))$。因此，$\mathrm{EX}(\mathfrak{m}(\mathrm{Sub}(\sigma))\ \phi) = \mathrm{EX}(\mathfrak{m}'(\sigma)\ \phi)$。

规则 $(\wedge R)$ 的情形：若从节点 $\nu$ 应用规则 $(\wedge R)$，令 $\tau = \sigma\colon\ \phi \wedge \psi$ 为 $\nu$的目标公式，且 $\tau' = \sigma\colon\ \phi$。根据 EX 的定义（定义 7.8），我们有 $\mathrm{EX}(\mathfrak{m}(\sigma)\ \phi) \subseteq \mathrm{EX}(\mathfrak{m}(\sigma)\ \sigma\colon\ \phi \wedge \psi)$，因此 $\mathrm{EX}(\mathfrak{m}(\sigma)\ \phi) \preceq_m \mathrm{EX}(\mathfrak{m}(\sigma)\ \sigma :\phi \wedge \psi)$。□

从上述证明中规则 $([\alpha]L)$ 的情形可见，推导 $Pr_{dlp} \vdash (\Gamma \Rightarrow \alpha \Downarrow \sigma, \Delta))$ 要求 $EX(m(\sigma)\ [\alpha]\phi) \neq \emptyset$，这是证明 $EX(\mathfrak{m}(\sigma')\ [\alpha']\phi)$ 与 $EX(\mathfrak{m}(\sigma)\ [\alpha]\phi)$ 之间严格关系 $\prec_m$ 的关键。

**命题的内容 5.10：** 给定如下形式的一个可靠规则

$$\frac{\Gamma_1 \Rightarrow \Delta_1 \quad \ldots \quad \Gamma_n \Rightarrow \Delta_n}{\Gamma \Rightarrow \Delta}\ ,\ n \geq 1,$$

其中所有公式均未标注标签，则规则

$$\frac{\sigma : \Gamma_1 \Rightarrow \sigma : \Delta_1 \quad \ldots \quad \sigma : \Gamma_n \Rightarrow \sigma : \Delta_n}{\sigma : \Gamma \Rightarrow \sigma : \Delta}$$

对任意标签 $\sigma \in free(\mathbf{L}\ \Gamma \cup \Delta \cup \Gamma_1 \cup \Delta_1 \cup \ldots \cup \Gamma_n \cup \Delta_n)$ 都是可靠的。

命题 5.10 的证明。令 $A = \Gamma \cup \Delta \cup \Gamma_1 \cup \Delta_1 \cup \ldots \cup \Gamma_n \cup \Delta_n$。假设序列 $\sigma\colon\ \Gamma_i \Rightarrow \sigma\colon\ \Delta_i\ (1 \leq i \leq n)$ 有效，我们需要证明序列 $\sigma\colon\ \Gamma \Rightarrow \sigma\colon\ \Delta$ 有效。首先，注意到每个序列 $\Gamma_i \Rightarrow \Delta_i\ (1 \leq i \leq n)$ 都有效。因为由于 $\sigma \in \mathrm{free}(\mathbf{L}\ A)$，对于任意满足 $s \models \Gamma_i$ 的 $s \in \mathbf{S}$，都存在一个记作 $\mathfrak{m}_s \in \mathbf{M}$ 的标签映射，使得$s =_A \mathfrak{m}_s(\sigma)$，并且 $\mathfrak{m}_s(\sigma) \models \Gamma_i$(定义 5.6)，因此 $\mathfrak{m}_s \models \sigma\colon\ \Gamma_i$(根据定义 4.13)。根据序列$\sigma\colon\ \Gamma_i \Rightarrow \sigma\colon\ \Delta_i$的有效性，对于某个 $\phi \in \Delta_i$，有 $\mathfrak{m}_s \models \sigma\colon\ \phi$，这意味着 $\mathfrak{m}_s(\sigma) \models \phi$。因此 $s \models \phi$。根据 $\Gamma_i \Rightarrow \Delta_i$ 的有效性，我们得到 $\Gamma \Rightarrow \Delta$ 有效。对于任意 $\mathfrak{m} \in \mathbf{M}$，若 $\mathfrak{m} \models \sigma\colon\ \Gamma$，则 $\mathfrak{m}(\sigma) \models \Gamma$(定义 4.13)。根据 $\Gamma \Rightarrow \Delta$ 的有效性，对于某个 $\phi \in \Delta$，有 $\mathfrak{m}(\sigma) \models \phi$。但这恰好意味着 $\mathfrak{m} \models \sigma\colon\ \phi$。因此，我们已经证明序列 $\sigma\colon\ \Gamma \Rightarrow \sigma\colon\ \Delta$ 有效。

□

**引理内容7.16：** 在与定理 7.15 相同的条件下，对于任意如下形式的有效序列：$\Gamma \Rightarrow \sigma\colon\ [\langle\alpha\rangle]\phi$，$Pr_{dlp} \vdash (\Gamma \Rightarrow \sigma\colon\ [\langle\alpha\rangle]\phi)$。

引理 7.16 的证明。令$^{\nu} =_{df} (\Gamma \Rightarrow \sigma\colon\ [\langle\alpha\rangle]\phi)$。我们对模态 $M_\nu$ 的数量 $[\cdot]$ 或 $\langle\cdot\rangle$ 以及 $\nu$ 中沿着以 $N_\alpha$ 为起点的程序序列所包含的不同程序的最大数量 $\alpha$ 进行同时归纳，其中涵盖所有标签和上下文（见定义 7.13）。

**基本情形**。 $M_\nu = N_\alpha = 1$，因此在序列 $\nu$ 中只有一个模态，即 $[\alpha]$,，而程序 $\alpha$ 要么是 $\downarrow$，要么是循环程序 $\neq\downarrow$，且只能执行如下形式的转移：$(\alpha,\ \sigma) \longrightarrow (\alpha,\ \sigma')$，其中某个 $\alpha'$ 处于某个上下文之下。对于 $\alpha = \downarrow$ 的情形，结论显然成立，因为应用规则 ($[\downarrow]$) 或规则 ($\downarrow$) 以及规则 (*Ter*)，即可直接得到结果。当 $\alpha \neq\downarrow$ 是循环程序时，该证明只是步骤情形证明的一个特例，如下所示。

**步骤情形**。此情形分为两部分。第一部分描述如何构造序列 $\nu$（下文称为"Proc"）的推导；第二部分进一步证明该推导是循环的。

第一部分：不失一般性，假设 $\alpha$ 是一个循环程序。对于标签 $\sigma \in \mathbf{L}$ 和上下文 $\Gamma$，根据定义 7.14，存在一个标签 $\sigma'$、一个上下文 $\Gamma'$ 以及一个替换 $\eta$，满足

(a) $\sigma = \eta(\sigma')$ 和 $\Gamma = \eta(\Gamma')$；(b) $\models (\Gamma \Rightarrow \sigma \Downarrow \alpha)$ 蕴含 $\models (\Gamma' \Rightarrow \sigma' \Downarrow \alpha)$；(c) 对于每个 $\alpha$-循环序列：$\Gamma'$：$(\alpha,\ \sigma',\ \Gamma_1)...(\alpha,\ \sigma'',\ \Gamma_n)$ $(n \geq 1)$，存在一个上下文 $\Gamma''$ 和一个替换 $\xi$，使得$\Gamma'' = \xi(\Gamma')$、 $\sigma'' = \xi(\sigma')$ 和 $\models (\Gamma_n \Rightarrow \Gamma'')$。

由 $\Gamma \Rightarrow \sigma : [\alpha]\phi$，可得到以下名为"Proc"的推导：

$$\dfrac{\dfrac{\dfrac{

(2) 每个子过程 1 都是一个预证明分支。一方面，根据 $\xi(\Gamma') = \Gamma''$（见上文 (c)），$\Gamma''$ 中的模态数量与 $\Gamma'$中的相同。另一方面，根据定义 5.1的第 4项，在从节点 3 推导到节点 4 的过程中强化上下文 $\Gamma'$时，我们可以确保与 $\Gamma'$ 相比，$\Gamma_1,...,\Gamma_n$中不会添加更多动态公式。因此，序列 $\Gamma_n \Rightarrow \Gamma''$ 中的模态数量等于序列 $\nu$中的模态数量，因而严格小于 $M_\nu$。所以，根据归纳假设，$Pr_{dlp} \vdash (\Gamma_n \Rightarrow \Gamma'')$。

(3) 每个子过程 2 都是一个循环证明分支。对于从节点 9 到 10 的这类推导步骤，由于在 $\Lambda$ 下无法到达来自 $(\beta', \varsigma')$ 的程序 $\alpha$，显然有 $N_{\beta'} < N_\alpha$，因为除 $\alpha$ 本身外，$\beta'$ 能到达的任何程序都可以由 $\alpha$ 到达（经过序列 $\alpha: \Gamma': (\alpha, \sigma', \cdot)...(\beta, \varsigma, \Lambda)(\beta', \varsigma', \Lambda)$）。因此，根据归纳假设，$Pr_{dlp} \vdash (\Lambda \Rightarrow \varsigma': [\beta']\phi)$。

接下来要证明的是，在 Proc 中，沿着每条推导路径，都存在一条渐进推导轨迹。注意到在 Proc 中，每条推导路径都必须无限次经过一个预证明分支子过程 1。因此，只需证明每个子过程 1 都有一条渐进推导轨迹。在一个子过程 1 中，考虑两种情况:

(i) 如果模态为 $[\alpha]$，根据定义 5.5，在规则 $([\alpha]R)$ 的每次推理中，两个序列右侧的 CP 对都是一个渐进步骤。(ii) 如果模态为 $\alpha$，由于 $\models (\Gamma \Rightarrow \sigma : \alpha\phi)$，$\models (\Gamma \Rightarrow \sigma \Downarrow \alpha)$。根据上文的 (b)，$\models (\Gamma' \Rightarrow \sigma' \Downarrow \alpha)$。根据相对于 $\mathfrak{F}_{ter}$ 的完备性（定义 5.1 的第 3 项），$Pr_{dlp} \vdash (\Gamma' \Rightarrow \sigma' : \alpha\phi)$。因此，根据定义 7.2，在规则 $(\alpha R)$ 的第一次推理中，两个序列右侧的 CP 对都是一个渐进步骤。

在上述两种情况下，渐进推导轨迹为：$\sigma': [\alpha]\phi$ 在节点 3，...，$\sigma'': [\alpha]\phi$ 在节点 4，$\sigma'': [\alpha]\phi$，$\sigma'': [\alpha]\phi$ 在节点 6，$\sigma': [\alpha]\phi$ 在节点 7，...，如 Proc 中所示。

□

## B Esterel 程序的循环演绎

Esterel [12] 是一种用于反应式系统的同步编程语言。下面我们先介绍 Esterel 程序的语义，然后说明为什么在传统验证框架中需要额外进行程序变换。最后，我们解释如何在 DLp 中表示 Esterel 程序，并给出该程序的循环演绎。
请注意，下面的介绍并不正式，也未涵盖 Esterel 语义的所有方面，但足以阐明我们的观点。

### B.1 DLp 中的 Esterel 程序

我们考虑一个由以 Esterel 语言 [12] 编写的程序的实例化 $\mathbf{P}_E$ 所构成的同步程序 $E$：

$$E =_{df} \{trap\ A \parallel B\ end\},$$

其中

$$A =_{df} \{loop\ (emit\ S(0)\ ;\ x := x - S\ ;\ if\ x = 0\ then\ exit\ end\ ;\ pause)\ end\}$$

$$B =_{df} \{loop\ (emit\ S(1)\ ;\ pause)\ end\}.$$

同步程序的行为由一系列实例 刻画。在每个实例中，程序可能会发生多个（原子）执行过程。每个变量在一个实例中的值都是唯一的。当多个程序并行运行时，它们在某个实例中的执行过程被认为是同时发生的。由此，并行同步程序的行为具有确定性。

在这个示例中，程序 $E$ 的行为说明如下：提交给 ACM 的稿件

| 世界 $w$ | $w(x)$ | $w(S)$ | 第 $n$ 个实例 |
|---|---|---|---|
| $w_1$ | 3 | 1 | 1 |
| $w_2$ | 2 | 1 | 2 |
| $w_3$ | 1 | 1 | 3 |
| $w_4$ | 0 | 1 | 4 |

表 6. 程序 $E$ 的迁移行为（起始 Fr） om $w_1$

- $x$、 $S$ 是两个变量。 $x$ 是一个局部变量，其定义域为 $\mathbb{Z}$； $S$ 是一个“信号”，其定义域为 $\mathbb{Z} \cup \{\perp\}$，其中 $\perp$ 表示没有信号。
- 关键字暂停 标志着一个实例的结束，此时所有信号都被设置为 $\perp$，表示“缺失”状态。
- 信号发射发出 $S(e)$表示将表达式 $e$ 的值赋给信号 $S$ 并广播该值。$x := x - S$ 是与FODL中相同的常规赋值。

- 在每个实例中，程序 $A$ 首先以值0发出信号 $S$，并将 $x$ 减去 $S$的当前值；然后检查 $x = 0$是否成立。而程序 $B$ 以值1发出信号 $S$。信号 $S$ 在一个实例中的值应为不同当前程序所发出的全部值之和。因此 $S$ 的值应为 $1 + 0 = 1$。
- 整个程序 $E$ 持续执行，直到条件 $x = 0$ 得到满足，此时退出 通过跳出陷阱 语句来终止整个程序。

从初始世界开始 $w_1$ 具有 $w_1(x) = 3$ 和 $w_1(S) = 1$，我们有一条执行路径 $w_1w_2w_3w_4$ 属于程序 $E$如表6所说明，其中我们省略了执行过程中程序的中间形式。

### B.2 Esterel 程序中的先前程序变换

在 Esterel 中，并行程序的行为通常并非真交错。其进程之间存在数据依赖。例如，在上述程序 $E$ 中，赋值 $x := x - S$ 只有在程序 $A$ 和 $B$ 中 $S$ 的所有值都被收集之后才能执行。换言之， $x := x - S$ 只有在 发出 $S(0)$ 和 发出 $S(1)$ 之后才能执行。对于这样的同步程序，额外的程序变换是必需的（参见 [24]）。我们需要首先对程序 $E$ 进行变换，例如将其变换为如下顺序程序：

$$E' = \{trap\ C\ end\},$$

其中

$$C =_{df} \{loop\ emit\ S(0)\ ;\ emit\ S(1)\ ;\ x := x - S\ ;\ if\ x = 0\ then\ exit\ end\ ;\ pause)\ end\}.$$

在 $C$中，我们按照正确的顺序，收集实例中来自 $A$ 和 $B$的所有微步。在 [24]中， $E'$ 称为 STA程序。请注意，这种预先转换可能非常耗费资源，因为可以想象，并行 Esterel 程序的行为可能非常复杂（例如 [10]）。

### B.3 Esterel 程序中 DLp 的实例化

实例化过程与 while 程序类似。令$\mathbf{P}_E$为 Esterel 程序的集合，$Var_E$为 Esterel 中局部变量和信号的集合。假设对于$\mathbf{P}_E$，存在 PLK结构 $\mathrm{K}_E = (\mathrm{S}_E, \longrightarrow, \mathrm{I}_E)$，其中每个世界 $w \in \mathrm{S}_E$都是一个映射 $w$： $Var_E \to (\mathbb{Z} \cup \{\perp\})$，它将每个局部变量映射为一个整数，并将每个信号映射为 $\mathbb{Z} \cup \{\perp\}$中的一个值。$\mathbf{P}_E$中的程序配置 $\sigma \in \mathbf{L}_E$ 作为一个标签，其定义旨在刻画该结构的含义

$$\{x_1 \mapsto e_1 \mid \ldots \mid x_n \mapsto e_n\}\ (n \geq 1),$$

其中，与 while 程序相比，唯一的区别在于它是一个栈（ $x_n \mapsto e_n$ 为栈顶元素），因此允许存在多个同名局部变量。例如，配置 $\{x \mapsto 5 \mid y \mapsto 1 \mid y \mapsto 2\}$ 中有两个不同的局部变量 $y$，它们分别存储值 1 和 2。

给定一个世界 $w \in \mathrm{S}_E$，标签映射 $\mathfrak{m}_w \in \mathbf{M}_E$（相对于 $w$）定义如下：对于任意形如 $\{x_1 \mapsto e_1 \mid \ldots \mid x_n \mapsto e_n\}$ 的配置 $\sigma \in \mathbf{L}_E$， $\mathfrak{m}_w(\sigma)$ 是满足以下条件的一个世界：

(1) $\mathfrak{m}_w(\sigma)(x_i) = w(e_j)$，其中 $n \geq j \geq i \geq 1$ 和 $j$ 是 $x_i \mapsto e_j$ 在 $\sigma$ 中的最大索引（即变量$x_i$最右侧的值）；
(2) 对于其他变量 $y \in Var_E$， $\mathfrak{m}_w(\sigma)(y) = w(y)$,

其中， $w(e)$ 对于表达式 $e$，其定义方式与 while 程序中的类似（参见例4.10）。例如，对于任意$\mathfrak{m}_w(\{x \mapsto 5 \mid y \mapsto 1 \mid y \mapsto 2\})(y) = w(2) = 2$。 $\mathfrak{m}_w$。
由于规则过于复杂（参见 [48]），我们省略了 Esterel 程序操作语义的规则集合（$\mathbf{Pr}^{op}$）$_E$的细节。

### B.4 程序的循环演绎 $E$

在 DLp 中，无需额外的程序变换即可直接对程序 $E$ 进行推理。这是因为 DLp 支持直接基于操作语义进行循环推理。在接下来的推导中，我们会看到， $E$ （经过变换后位于 $C$ 上方）的外部循环结构实际上由循环推导树本身体现出来。

我们证明如下性质：

$$v_2 =_{df} \sigma_1 : x > 0 \Rightarrow \sigma_1 : \langle E \rangle true,$$

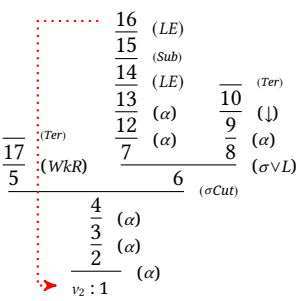
该性质表明，在配置 $\sigma_1 = \{x \mapsto v,\ S \mapsto \perp\}$ 下，若 $v$ 是表示 $x$ 初始值的新变量，且$x > 0$，则 $E$ 最终可以终止。
.....的推导 $v_2$ 如表7所示。程序 $E$ 的符号执行依赖于规则

$$\frac{\Gamma \Rightarrow \sigma' : \langle \alpha' \rangle \phi, \Delta}{\Gamma \Rightarrow \sigma : \langle \alpha \rangle \phi, \Delta}\ (\langle \alpha \rangle),\ \text{if } Pr_{dlp} \vdash (\Gamma \Rightarrow (\alpha, \sigma) \longrightarrow (\alpha', \sigma'), \Delta),$$

可由表($[\alpha]L$)、($\neg R$) 和 ($\neg L$) 中的规则2推导得出。我们省略了规则实例($\langle \alpha \rangle$)中程序转换和终止的所有旁支推导。

从节点 2 到节点 3 是一个渐进步骤。直观地说，要看出程序 $\sigma_2 \Downarrow$ 陷阱 $((x := x - S; A');\ A) \parallel B$ 结束 会终止，我们观察到变量 $x$ 的值在每次循环中（执行 $x := x - S$）都会减 1，因此最终会到达语句退出 。从节点 13 到节点 14，以及从节点 15 到节点 16，规则

$$\frac{\Gamma, \phi' \Rightarrow \Delta}{\Gamma, \phi \Rightarrow \Delta}\ (LE),\ \text{if } \phi \to \phi' \in \mathbf{F} \text{ is valid}$$




其他符号的定义：
$A =_{df}$循环 (发出 $S(0)$； $x := x - S$; 如果 $x = 0$ 则退出 结束 ; 暂停) 结束
$B =_{df}$循环 (发出 $S(1)$ ; 暂停) 结束
$A' =_{df}$(若 $x = 0$ 则退出 结束 ; 暂停)
$\sigma_1 =_{df} \{x \mapsto v \mid S \mapsto \bot\}$
$\sigma_2 =_{df} \{x \mapsto v \mid S \mapsto 0\}$
$\sigma_3 =_{df} \{x \mapsto v \mid S \mapsto 1\}$
$\sigma_4 =_{df} \{x \mapsto v - 1 \mid S \mapsto 1\}$
$\sigma_5 =_{df} \{x \mapsto v - 1 \mid S \mapsto \bot\}$

| | | | |
|---|---|---|---|
| 1: | $\sigma_1 : x > 0$ | $\Rightarrow$ | $\sigma_1 : \langle$陷阱 $A \parallel B$ 结束$\rangle$真 |
| 2: | $\sigma_1 : x > 0$ | $\Rightarrow$ | $\sigma_2 :$ 陷阱 $((x := x - S; A')\,;\ A) \parallel B)$ 结束真 |
| 3: | $\sigma_1 : x > 0$ | $\Rightarrow$ | $\sigma_3 : \langle$陷阱 $((x := x - S; A')\,;\ A) \parallel ($暂停 ; $B)$ 结束$\rangle$真 |
| 4: | $\sigma_1 : x > 0$ | $\Rightarrow$ | $\sigma_4 : \langle$陷阱 $(A'\,;\ A) \parallel ($暂停; $B)$ 结束$\rangle$真 |
| 5: | $\sigma_1 : x > 0$ | $\Rightarrow$ | $\sigma_4 : \langle$陷阱 $(A'\,;\ A) \parallel ($暂停 ; $B)$ 结束$\rangle$真 $\sigma_1 : (x - 1 \neq 0 \vee x - 1 = 0)$ |
| 17: | $\sigma_1 : x > 0$ | $\Rightarrow$ | $\sigma_1 : (x - 1 \neq 0 \vee x - 1 = 0)$ |
| 6: | $\sigma_1 :\ x > 0, \sigma_1 :\ (x - 1 \neq 0 \vee x - 1 = 0)$ | $\Rightarrow$ | $\sigma_4 :\ \langle$陷阱 $(A'\,;\ A) \parallel ($暂停; $B)$ 结束$\rangle$真 |
| 7: | $\sigma_1 :\ x > 0, \sigma_1 :\ x - 1 \neq 0$ | $\Rightarrow$ | $\sigma_4 :\ \langle$陷阱 $(A'\,;\ A) \parallel ($暂停; $B)$ 结束$\rangle$真 |
| 12: | $\sigma_1 :\ x > 0,\ \sigma_1 :\ x - 1 \neq 0$ | $\Rightarrow$ | $\sigma_4 :\ \langle$陷阱 (暂停; $A) \parallel ($暂停; $B)$ 结束$\rangle$真 |
| 13: | $\sigma_1 : x > 0,\ \sigma_1 : x - 1 \neq 0$ | $\Rightarrow$ | $\sigma_5 : \langle$陷阱 $A \parallel B$ 结束$\rangle$真 |
| 14: | $\sigma_5 :\ x + 1 > 0, \sigma_5 :\ x \neq 0$ | $\Rightarrow$ | $\sigma_5 : \langle$陷阱 $A \parallel B$ 结束$\rangle$真 |
| 15: | $\sigma_1 :\ x + 1 > 0, \sigma_1 :\ x \neq 0$ | $\Rightarrow$ | $\sigma_1 : \langle$陷阱 $A \parallel B$ 结束$\rangle$真 |
| 16: | $\sigma_1 : x > 0$ | $\Rightarrow$ | $\sigma_1 : \langle$陷阱 $A \parallel B$ 结束$\rangle$真 |
| 8: | $\sigma_1 :\ x > 0, \sigma_1 :\ x - 1 = 0$ | $\Rightarrow$ | $\sigma_4 : \langle$陷阱 $(A'\,;\ A) \parallel ($暂停; $B)$ 结束$\rangle$真 |
| 9: | $\sigma_1 :\ x > 0, \sigma_1 :\ x - 1 = 0$ | $\Rightarrow$ | $\sigma_5 :\ \langle\downarrow\rangle$真 |
| 10: | $\sigma_1 :\ x > 0, \sigma_1 :\ x - 1 = 0$ | $\Rightarrow$ | $\sigma_5 :\ $ 真 |

表 7. 性质 $v_2$ 的推导

已应用，可通过以下推导得出：

$$\dfrac{\dfrac{\Gamma, \phi' \Rightarrow \Delta}{\Gamma, \phi, \phi' \Rightarrow \Delta}\ (WkL) \qquad \dfrac{}{\Gamma, \phi \Rightarrow \phi', \Delta}\ (Ter)}{\Gamma, \phi \Rightarrow \Delta}\ (Cut)$$

从节点 14 到 15，应用规则（*Sub*）

$$\dfrac{\Gamma \Rightarrow \Delta}{\Gamma[e/x] \Rightarrow \Delta[e/x]}\ (Sub)$$

被应用，其中 $(\cdot)[e/x]$ 是对替换 *Sub* 标签（定义 4.12）的实例化。其定义与第 6.1 节示例中的定义相同。注意，序列 14 可以写成：

$$\sigma_1[v - 1/v] : x + 1 > 0, \sigma_1[v - 1/v] : x \neq 0 \Rightarrow \sigma_1[v - 1/v] : \langle trap\ A \parallel B\ end \rangle true.$$

序列16是一个回链到序列1的芽。整个预证明是循环的，因为它只有一条推导路径：1，2，3，4，6，7，12，13，14，15，16，1，.....其中包含一条渐进轨迹，其元素在表7中以带下划线的形式标出。

### C 分离逻辑在 DL 中的编码ꝑ

我们实例化 DLꝑ 来表达分离逻辑 [51] ——这是霍尔逻辑的一种扩展，用于推理共享可变程序数据结构。实例化后的理论称为 DLꝑ-SP。下面我们只讨论分离逻辑的一部分，但这已足以阐明我们的观点。

**分离逻辑**。下文假设读者熟悉分离逻辑，我们仅对其语义作非正式说明。为简单起见，我们只介绍部分分离逻辑的原语：公式的原子公式 $e \dashrightarrow e'$ 和关键算子 $*$，以及程序的原子语句 $x := \mathbf{cons}(e)$、$x := [e]$、$[e] := e'$ 和$\mathbf{disp}(e)$。我们省略公式中的另一个关键算子 $-\!\!*$，以及因具体情况而异的组合程序。

下面约定一些记号：给定一个偏函数 $f\colon A \rightharpoonup B$，我们用 $\mathrm{dom}(f)$ 表示 $f$ 的定义域。对于一个集合 $C$，偏函数 $f|_C\colon A \rightharpoonup B$ 是将函数 $f$限制在定义域 $\mathrm{dom}(f) \cap C$ 上得到的函数。$f[x \mapsto e]$表示这样一个偏函数：它将 $x$映射到 $e$，并将其定义域中的其他变量映射到与 $f$ 相同的值。令 $\mathbb{V} = \mathbb{Z} \cup \mathrm{Addr}$ 为值的集合，其中 Addr 是地址集合。我们假设 Addr 可用一个无限的整数集合表示。在分离逻辑中，存储 $s\colon V \to \mathbb{V}$ 是一个将每个变量映射到 $\mathbb{V}$ 中某个值的函数；堆 $h : \mathrm{Addr} \rightharpoonup \mathbb{V}$ 是一个将地址映射到 $\mathbb{V}$ 中某个值的偏函数，用以表示该值存储在该地址所指示的内存中。$\mathrm{dom}(h)$ 是 Addr 的有限子集。状态是一个存储-堆对 $(s, h)$。当 $\mathrm{dom}(h_1) \cap \mathrm{dom}(h_2) = \emptyset$ 时，不相交关系 $h_1 \perp h_2$ 有定义。

这里非正式地解释每个原语的语义。给定状态 $(s, h)$，语句 $x := \mathrm{cons}(e)$ 会在 $h$ 中分配一块由新整数 $n$ 寻址的内存，用于存储表达式 $e$ 的值（从而得到新堆 $h \cup \{(n, s(e))\}$，其中 $n \notin \mathrm{dom}(h)$），并将 $n$ 赋给 $x$。语句 $x := [e]$ 将 $h$ 中地址 $e$ 的值（即 $h(s(e))$）赋给变量$x$。$[e] := e'$ 表示将值 $e'$ 赋给 $h$ 中地址 $e$ 所指向的内存（即得到新堆 $h[s(e) \mapsto s(e')]$）。$\mathrm{disp}(e)$ 表示释放堆中地址 $e$ 所指向的内存（即得到新堆 $h|_{dom(h)\setminus\{s(e)\}}$）。公式$e \dashrightarrow e'$ 表示值 $e'$ 存储在地址 $e$ 所指向的内存中。给定状态 $(s, h)$，若$h(s(e)) = s(e')$，则 $s, h \models e \dashrightarrow e'$ 有定义。对于任意分离逻辑公式 $\phi$ 和 $\psi$，若存在堆 $h_1$、$h_2$，使得 $h = h_1 \cup h_2, h_1 \perp h_2$，且 $s, h_1 \models \phi$ 和 $s, h_2 \models \psi$，则有 $s, h \models \phi * \psi$。

示例 *C.1.* 设 $(s, h)$ 是一个状态，满足 $s(x) = 3, s(y) = 4$和 $h = \emptyset$，则下表展示了在执行以下程序的过程中，各状态中焦点变量和地址的信息：

$$(s, h) \xrightarrow{x:=\mathbf{cons}(1)} (s_1, h_1) \xrightarrow{y:=\mathbf{cons}(1)} (s_2, h_2) \xrightarrow{[y]:=37} (s_3, h_3) \xrightarrow{y:=[x+1]} (s_4, h_4) \xrightarrow{\mathbf{disp}(x+1)} (s_5, h_5).$$

| | 存储 | | Heap |
|---|---|---|---|
| $s$ | $x : 3, y : 4$ | $h$ | 空 |
| $s_1$ | $x : 37, y : 4$ | $h_1$ | $37 : 1$ |
| $s_2$ | $x : 37, y : 38$ | $h_2$ | $37 : 1, 38 : 1$ |
| $s_3$ | $x : 37, y : 38$ | $h_3$ | $37 : 1, 38 : 37$ |
| $s_4$ | $x : 37, y : 37$ | $h_4$ | $37 : 1, 38 : 37$ |
| $s_5$ | $x : 37, y : 37$ | $h_5$ | $37 : 1$ |

设 $\phi =_{df} (x \dashrightarrow 1 * y \dashrightarrow 1)$、$\psi =_{df} (x \dashrightarrow 1 \wedge y \dashrightarrow 1)$，则有 $s_2, h_2 \models \phi$以及 $s_2, h_2 \models \psi$、$s_5, h_5 \models \psi$，但 $s_5, h_5 \not\models \phi$，因为$x$和 $y$ 都指向存储值 1 的同一块内存。

$$\frac{}{\Gamma \Rightarrow (x{:=}\,\mathbf{cons}(e),\ (s,\ h)) \longrightarrow (\downarrow,\ (s[x \mapsto n],\ h \cup \{(n,\ s(e))\}))\ \Delta}\ {}^{1\,(\mathbf{cons})}$$

，其中 $n$ 相对于 $h$ 是新的

$$\frac{}{\Gamma \Rightarrow (x{:=}\,[e],\ (s,\ h)) \longrightarrow (\downarrow,\ (s[x \mapsto h(s(e))],\ h))\ \Delta}\ (x{:=}[e])$$

$$\frac{}{\Gamma \Rightarrow ([e]{:=}\,e',\ (s,\ h)) \longrightarrow (\downarrow,\ (s,\ h[s(e) \mapsto s(e')]))\ \Delta}\ ([e]{:=}e')$$

$$\frac{}{\Gamma \Rightarrow (\mathbf{显示}(e),\ (s,\ h)) \longrightarrow (\downarrow,\ (s,\ h|_{dom(h)\setminus\{s(e)\}}))\ \Delta}\ (\mathbf{显示})$$

表8. 分离逻辑中原子语句程序转换的部分规则（$\mathbf{Pr}^{op})_{SP}$

DLp 中分离逻辑的编码。在 DLp 中，令 $\mathrm{P}_{SP}$ 和 $\mathrm{F}_{SP}$ 分别为分离逻辑的程序集合与公式集合。在分离逻辑的 PLK结构 $\mathrm{K}_{SP} = (\mathrm{S}_{SP}, \longrightarrow_{SP}, \mathrm{I}_{SP})$ 中，$\mathrm{S}_{SP} = \{(s,h) \mid s\colon V \to \mathbb{V}, h\colon \mathrm{Addr} \rightharpoonup \mathbb{V}\}$，$\mathrm{I}_{SP}$ 将 $\mathrm{F}_{SP}$ 中的每个公式解释为如上所述。我们直接选取存储-堆对作为分离逻辑的配置，记作 $\mathrm{L}_{SP}$。在这种情况下，我们简单地令 $\mathrm{M} =_{df} \{\tau\}$，其中 $\tau\colon \mathrm{L}_{SP} \to \mathrm{S}_{SP}$ 是一个常值映射，满足对于任意 $(s,h) \in \mathrm{S}_{SP}$，$\tau((s,h)) =_{df} (s,h)$。表8列出了 $(\mathbf{Pr}^{op})_{SP}$ 中原子语句的程序转换规则。

为了进一步将 $\mathbf{F}_{SP}$ 中类似 $\phi * \psi$ 的公式推导为更简单的形式，需要提出除 $Pr_{ldlp}$ 之外的其他规则。例如，我们可以提出一条规则

$$\frac{\Gamma \Rightarrow h_1 \perp h_2, \Delta \quad \Gamma \Rightarrow (s,h_1) : \phi, \Delta \quad \Gamma \Rightarrow (s,h_2) : \psi, \Delta}{\Gamma \Rightarrow (s, h_1 \cup h_2) : \phi * \psi, \Delta}\ (\sigma *)$$

分解堆 $h_1 \cup h_2$ 和公式 $\phi * \psi$，或应用框架规则

$$\frac{\Gamma \Rightarrow (s,h) : \phi, \Delta}{\Gamma \Rightarrow (s,h) : \phi * \psi, \Delta}\ (\sigma Frm),\ \text{if no variables of } dom(h) \text{ appear in } \psi,$$

只需分解公式 $\phi * \psi$。这些规则的灵感来自分离逻辑中针对程序的对应规则。实际上，标签可以采用比此处所示的存储-堆对更明确的结构。相应地也可以得到类似的编码。从这个例子中，我们设想整个分离逻辑理论都可以嵌入 DLp，其中上述这类附加规则支持对分离逻辑公式进行“配置级推理”。