arXiv:2301.09802v2 [cs.LO] 7 Apr 2023

余归纳类型的归纳推理

ALEXANDER BAGNALL，美国俄亥俄大学GORDON STEWART，美国BedRockSystems 公司ANINDYA BANERJEE，西班牙IMDEA 软件研究所

我们介绍 AlgCo（代数余归纳），这是一个实用框架，可用于对常见的余归纳类型进行归纳推理，例如余自然数、流，以及具有有限分支因子的无穷树。其核心思想是利用域理论中的代数完全偏序 概念，通过对元素的“稠密”集合进行原始递归，为余归纳类型定义连续运算，从而能够通过直接的归纳证明，便捷地推理代数余归纳类型。我们在 Coq 中实现了 AlgCo 框架，并通过验证 Eratosthenes 筛法的一个流变体、一个基于形式语言的余归纳Trie编码的正则表达式库，以及随机比特模型中离散概率分布上的余归纳采样过程的期望值语义，展示了该框架的强大能力。

1 引言

作为定义良基数据上的计算并证明其正确性的工具，归纳原理对大多数计算机科学家而言都非常熟悉。因此，Coq 和 Agda 等证明助手为使用归纳进行编程和证明提供了便捷的支持[1, 2]<style id='4'>。而与之对偶的余归纳 [3], 则远未得到同等程度的支持。

余归纳为针对无穷结构进行编程并验证其性质提供了自然的方式，例如余自然数（扩展了“无穷远点”的自然数）、流（无限列表） [4],以及可能不会终止的决策过程（例如，从概率程序编译而来的离散分布采样器 [5]）。然而，余归纳推理往往有悖于直觉，而 Coq 和 Agda 等证明助手的设计明显偏向归纳，这会加剧余归纳本身难以凭直觉理解的问题。因此，在证明助手中使用余归纳，常常会因严格的语法守卫条件（限制了允许使用的余归纳定义范围）以及生成的余归纳假设与自动化策略难以良好配合 [4, 6]）而受到技术障碍的困扰。Hur 等人提出了参数化余归纳，之后 Pous 将其推广 [7] （paco [6]），借助语义上的守卫性概念来解决严格语法检查的问题。虽然 paco 极大地改进了 Coq 中的余归纳，但它并未从根本上脱离原始余归纳。

本文提出了一种替代策略：我们不对 Coq 中的余归纳进行一般化（例如，paco 将语义守卫性一般化），而是考虑一类余归纳类型——即对应于代数 CPO 的类型 [8, 9] ——并针对这些类型采用一种截然不同的策略来推理余归纳程序。这种策略更适合 Coq 等证明助手本质上偏向归纳的特性。

为说明我们的方法，考虑为余归纳流（即无限列表）定义过滤操作的问题。在 Haskell 中，可以根据给定的谓词过滤流——例如，偶数流可表示为 $[n|n \leftarrow [0..], n `mod` 2 =$

0]。我们可以尝试在 Coq 中为流实现类似的过滤操作，如下所示：

```
CoFixpoint filter (P : ℕ → 𝔹) (s : Stream ℕ) ≜
  match s with Cons n s' ⇒
    if P n then Cons n (filter P s') else filter P s'
  end.
```

然而，由于存在未守卫递归调用（它没有被包裹在 **Cons** 构造子中），此定义会被拒绝。事实上，拒绝它是件好事，否则我们就可以通过按 $P := \lambda \_.$ **false**筛选任意流来构造一个发散项（从而使 Coq 的逻辑变得不一致）。虽然在某些特定情况下，我们或许能够证明按给定谓词筛选流是安全的（也就是说，结果流将具有生产性 [4]），但我们无法这样做，因为filter 本身就无法定义。

一种常见的解决方案（例如，Xia 等人在交互树库 [10] 中采用的方案）是在流类型中添加一个构造子，用于表示所谓的“静默步骤”（*Tau* 节点）。Tau 节点可以轻松通过守卫性检查器的检查——只需用 Tau 应用包裹未守卫的共递归调用——但这会使定义和证明需要处理额外的情况，并带来不必要的执行开销。此外，用无限 Tau 序列替代发散点，会让流的消费者承担处理发散的责任，从而使后续的计算和分析变得复杂。

作为 Tau 带来的复杂性示例，可以考虑对一个实数流求无穷和，其中可能出现 Tau 节点。由于缺少针对流的一般归纳原理，我们转而考虑流与其和之间的余归纳关系（其中$\mathbb{R}^{\infty}_{\geq 0}$ 表示扩充了 $+\infty$的非负实数）：

```
CoInductive sum : Stream ℝ∞≥0 → ℝ∞≥0 → ℙ ≜
| sum_tau : ∀ s r, sum s r → sum (Tau s) r
| sum_cons : ∀ s x r, sum s r → sum (Cons x s) (x + r).
```

并尝试证明该关系是函数性的：

```
Lemma sum_functional (s : Stream ℝ∞≥0) (a b : ℝ∞≥0) :
  sum s a → sum s b → a = b.
Proof. … Abort. (* unprovable *)
```

但 sum_函数性无法证明，因为它并不成立。问题在于，余归纳关系被解释为满足其规则的最大关系，因此往往会关联比预期更多的元素对。例如，流 Ω、Tau Ω 通过 sum 与每个 r 都有关联： $\mathbb{R}$，因此 sum 显然不是函数（即确定性）关系，因为例如，sum Ω 0 且 sum Ω 1，但 $0 \neq 1$）。我们可以将引理的适用范围限制为不含无限 Tau 序列的流，但这样一来，我们就得为所有流的构造证明这一附加条件。通过 paco 定义 sum 关系也无济于事，因为它同样存在根本性问题：它被定义为单调泛函的最大不动点。

我们采用一种不同的方法（类似于 [11]）来定义 受域理论启发的 sum。我们首先在列表上定义 sum 的归纳类比：

Fixpointlsum（l：列表 $\mathbb{R}^{\infty}_{\geq 0}$）：
$\mathbb{R}^{\infty}_{\geq 0}$，**匹配**l **与**| nil ⇒ 0| cons x l'
⇒ x + lsuml'**结束**。

接下来，对于 s： 流$\mathbb{R}^{\infty}_{\geq 0}$，我们定义sums： $\mathbb{R}^{\infty}_{\geq 0}$，sup{lsum$l$ | l是 s的有限前缀。这个sum 的定义会将每个流映射到 $\mathbb{R}^{\infty}_{\geq 0}$中的唯一元素，即使存在 Tau 节点也是如此。我们可以进一步问：在什么条件下，才能以这种方式在余归纳类型上定义函数映射？答案要从域理论的一项基本结果（引理12）说起：

- $\mathbb{R}^{\infty}_{\geq 0}$ 是一个 CPO（完备偏序），- 流 $\mathbb{R}^{\infty}_{\geq 0}$ 是一个代数 CPO，- 且 nsum 是单调。

我们称sum为*lsum*的连续扩张。按定义，它是连续的；关于它的许多证明（具体而言，关于连续性质的证明，见第2.5节）都可以归结为对列表进行归纳的简单证明。此外，从流到 $\mathbb{R}^{\infty}_{\geq 0}$ 的每个连续函数（实际上，从任意代数 CPO 到另一个 CPO 的连续函数）都可以通过这种方式得到，也就是说，作为定义在有限元素上的单调函数的连续扩张。

余归纳类型与 CPO 之间的一种基本联系可以这样理解：将余归纳类型解释为给定函子的终余代数 [12], 后，它便凭借终性而具有一种规范的引入原则，用于定义该类型的元素。类似地，CPO 的完备性也可以视为一种引入原则：CPO 中的元素作为近似构成的有向集的上确界引入。因此，余归纳类型自然构成 CPO（例如，流类型可以看作列表类型关于有向集上确界的“完备化”[13]）。此外，利用域的代数性（存在稠密紧子集）以及陪域的完备性（此处分别为流 $\mathbb{R}^{\infty}_{\geq 0}$ 和 $\mathbb{R}^{\infty}_{\geq 0}$），我们能够为代数域提供一种连续消去原则（引理12），用于定义连续函数，例如filter和sum，以及定义其上的连续谓词和关系（第2.5节）。

尽管域理论研究者早已熟知代数 CPO 的理论， [9], 但其与余归纳类型之间联系的实际意义似乎尚未得到足够重视。据我们所知，尚无先前工作明确揭示这种联系，并详细阐述其在定理证明环境中的实际应用。因此，本文的主要贡献是通过提供一个全面的框架（*AlgCo*，代数余归纳的简称），阐明代数 CPO 概念的强大能力与普适性。该框架支持在 Coq 中针对构成代数 CPO 的余归纳类型（包括余自然数、流和无穷树等常用结构）进行连续函数的编程与证明，并通过若干有趣的用例展示其效用，其中包括概率编程语言的语义（第6节）。在第4.1节中，我们将展示上述

filter 操作可以借助 AlgCo 的工具来定义，完全无需使用 Tau 节点。

1.1 贡献

本文作出以下贡献：

概念。 我们介绍 AlgCo（Algebraic Coinductives 的简称）：一个全面的类型论框架，用于在一类称为代数 CPO 的结构上编写连续函数。AlgCo 使得可以将代数 CPO 上的定义和证明，归约为原始递归以及关于有限近似的归纳证明。我们提供了该理论在 Coq 中的实现，称为AlgCo。

技术方面。本文的技术贡献包括：

- 介绍代数余归纳的基本概念和证明原则（第2节）；
- 提供一个用于余自然数的惰性余迭代 框架（第3节）；
- 提供一个用于推理余归纳流的库，并将其应用于形式化验证的埃拉托斯特尼筛法（第 4节）；
- 提供基于余归纳前缀树的、经过验证的正则表达式匹配器（第 5节）；以及
- 提供一个用于推理余归纳二叉树的库，并将其应用于随机比特模型中可能不终止的抽样过程的（实值）期望值语义（第 6节），最终给出一个定理，证明生成的样本相对于该语义均匀分布（定理 69）。

源代码。 PDF 中嵌入的超链接指向相应的 Coq 源代码。

1.2 局限性

诸如 sum : 流 $\mathbb{R}^{\infty}_{\geq 0} \to \mathbb{R}^{\infty}_{\geq 0}$ 这样的连续扩张是通过非构造性的上确界算子定义的（参见第 2.1 节），因此通常不可计算。不过，在某些特殊情况下，包括惰性余迭代（第 3.1 节）和余折叠（第 4.1 节），我们提供了实用的 Haskell 提取原语。第 4.5 节将更详细地讨论可计算性。

本文所述技术仅适用于构成代数 CPO 的类型。虽然这涵盖了许多有用的余归纳类型，例如流和无限二叉树，但也存在许多不适用该技术的有趣类型，例如在深度和广度上都无限的树。

公理基础。 我们通过排中律、构造性任意描述和函数外延性扩展 Coq 的类型论 [15, 16]。我们还使用 Coq 的实数公理库，以及用于余归纳类型的外延性公理（例如，第 1 节 3中的公理；参见附录 B）。

2 代数余归纳

AlgCo 是一个形式化框架，用于定义和推理一类称为代数 CPO 的结构上的 Scott 连续函数（第 2.2节）。代数 CPO 的例子包括余自然数（例 17）、流 [3, 4] （第 4节）、可数集，以及具有固定有限分支因子的无穷树（第 6节）。

代数余归纳框架所依据的大部分基础理论早已为人熟知，可以在域理论的经典著作中找到 [9, 17]。然而，它对于类型论形式验证的一些实际意义仍未得到探索。本节将展示，关于代数 CPO 的一个基本结果（引理12）如何为在 Coq 中使用余归纳类型进行编程提供强大的框架。

2.1 域理论

我们先简要介绍序理论和域理论中的一些定义，本文其余部分都将以此为基础。

定义 1 (有序类型)。 有序类型 (或称偏序)是一种类型 A，带有一个序关系 $\sqsubseteq_A$，该关系具有自反性和传递性。

我们通常将 $\sqsubseteq_A$ 视为一种逼近关系，使得当 a 是 b某种更粗略或定义程度更低的版本时，$a \sqsubseteq b$。若有序类型 A 包含一个底元素 $\bot_A$，且满足 $\forall\, a\colon A,\ \bot_A \sqsubseteq_A a$，则称其为有底的。$A$ 也可能有一个最大元 $\top_A$，且满足 $\forall\, a\colon A,\ a \sqsubseteq_A \top_A$。当有序类型 A 可由上下文明确时，我们通常省略 $\sqsubseteq_A$ 和 $\bot_A$上的下标。

定义 2 (序等价)。 有序类型 x 中的元素 y 和 A 是序等价的，记作 $x \simeq y$，当且仅当 $x \sqsubseteq y$ 且 $y \sqsubseteq x$。

定义 3 (有向集)。 对于指标类型 I和有序类型 A，集合 $U\colon I \to A$是有向的，当且仅当 $\forall\, i\, j\colon\ I,\ \exists\, k\colon\ I,\ U\, i \sqsubseteq U\, k \wedge U\, j \sqsubseteq U\, k$，或者 向下有向，当且仅当 $\forall\, i\, j\colon\ I,\ \exists\, k\colon\ I,\ U\, k \sqsubseteq U\, i \wedge U\, k \sqsubseteq U\, j$。

直观地说，有向集中的所有元素最终都在逼近同一个对象。这引出了一个自然的收敛概念：只要上确界存在，有向集就会“收敛”到其上确界。有向集的一个特例是ω-链：

定义 4 (ω-链)。 对于有序类型 A，集合 $C\colon\ \mathbb{N} \to A$ 是一个 ω-链，当且仅当 $\forall\, i\, j\colon \mathbb{N},\ i \le j \Rightarrow C\, i \sqsubseteq C\, j$。

定义 5 （完备偏序）。 有序类型 A 是完备偏序（CPO），当每个有向 $U\colon \mathbb{N} \to A$ 都有上确界；或者是下完备偏序（lCPO），当每个向下有向 $V\colon\ \mathbb{N} \to A$ 都有下确界。若 A 是d-格，则称其同时是 CPO 和 lCPO。

我们选择专门处理可数有向集，这只是形式化过程造成的。我们也可以改用 ω-CPO（其中每个 ω-链都有上确界），但通常构造有向集比构造 ω-链更容易（例如，任何实数集合显然都是有向集）。

注 1 (ℙ是一个 d-格)。 命题 $\mathbb{P}$ 的类型按蕴含关系排序（即，$P \sqsubseteq Q \Longleftrightarrow P \Rightarrow Q$）是一个 d-格，其 底 $\bot$ 和 顶 $\top$ 满足：对于任意 U： $I \to \mathbb{P}$（不一定是（向下）有向的），$\sup(U)$ $\exists\, i.U\ i$ 和 $\inf(U)$ $\forall\, i.U\ i$。

注释 2 ($\mathbb{B}$是一个 d-格)。 类型 $\mathbb{B}$ 的布尔值构成一个 d-格，其底为 **false**，顶为**true**。

注释 3 ($\mathbb{R}^{\infty}_{\geq 0}$ 是一个 d-格)。 类型 $\mathbb{R}^{\infty}_{\geq 0}$ 的非负扩展实数构成一个 d-格，其底为 0，顶为 $+\infty$。

定义 6 (单调)。 对于有序类型 A 和 B，函数 f： $A \to B$ 是单调的 当 $\forall\, x\ y$： A、 $x \sqsubseteq y \Rightarrow f\ x \sqsubseteq f\ y$，或者 反单调的 当 $\forall\, x\ y$: A， $x \sqsubseteq y \Rightarrow f\ y \sqsubseteq f\ x$。

定义 7（连续）。对于有序类型 A 和 B，函数 f: $A \to B$ 在对每个有向集 U: $\mathbb{N} \to A$， $f(\sup U) = \sup(f \circ U)$ 时是连续的；当 $f(\sup U) = \inf(f \circ U)$ 时，则是余连续的。

注意，（余）连续性蕴含（反）单调性。

定义 8（下连续）。对于有序类型 A 和 B，函数 f: $A \to B$在对每个向下有向集 U: $\mathbb{N} \to A$都满足$f(\inf U) = \inf(f \circ U)$ 时，称为下连续（l-连续）；若满足 $f(\inf U) = \sup(f \circ U)$，则称为 l-余连续。

计算上确界。有向集的上确界通常不可计算。不过，我们可以使用不定描述公理和排中律 [15] 来定义一个 sup（inf）算子（希尔伯特 ε 算子的特例 [18]），从而在 Coq [16]的计算上下文中，以经典风格简洁地表达上确界（下确界）。我们定义 $\sup : (\mathbb{N} \to A) \to A$，使得 sup f是任意有向 f: $\mathbb{N} \to A$的上确界；并定义 $\inf : (\mathbb{N} \to A) \to A$，使得 inf f是任意向下有向 f: $\mathbb{N} \to A$的下确界。任何试图使用这些算子进行计算的操作都会受阻 [19],，但在某些特殊情况下，可以通过提取原语将它们实现为 Haskell 中的惰性执行（参见第 4.5 节中的讨论）。

2.2 代数 CPO

传统上，如果一个 CPO 包含一组“基”元素，可用于逼近域 [9] 中的任意元素，我们就称它为代数的 。这些基元素必须是紧致的:

定义 9 (紧元)。 设 A 是一个有序类型。元素 x： A 是紧致的，当且仅当对于每个有向集 U: $\mathbb{N} \to A$，若 $\sup(U) = x$，则有 $\exists\, i,\ U\ i = x$。

一个紧致元素 x： A虽然不一定是有限的，但它是有限可逼近的，也就是说，任何“收敛”到 x 的有向集都包含一个同样收敛到 x的有限子集（参见第5 节，其中有一个代数 CPO，其基元素是紧致的但不是有限的）。若类型 A 的每个 x： A 都是紧致的，则该类型是紧致的。紧致性至关重要，因为它与归纳原理的存在相吻合；本文的一个关键思想是

将关于余归纳类型上连续函数的推理转化为关于基元素的归纳推理（例如，参见引理33、34、65、66 以及定理37、38）。

我们将传统的代数性定义调整到类型论框架中：如果存在一个基类型 B，可以注入到 A 中（因此类似于 A 的子集），并且在 A 中“稠密”，那么 CPO A 就是代数的；也就是说，A 的所有元素都可以表示为 B 中元素的有向集的上确界，这些元素被注入到 A 中。

定义 10 (稠密)。 设 A 是一个 CPO，且 B 是一个有序类型。 B 在 中稠密 于 A 当存在连续运算

$$\mathsf{incl}_{B,A} : B \to A$$
$$\mathsf{idl}_{B,A} : A \to \mathbb{N} \to B$$

使得对于所有 a： A,

$$\mathsf{idl}_{B,A}\ a \text{ is an } \omega\text{-chain, and} \qquad \sup\left(\mathsf{incl}_{B,A} \circ \mathsf{idl}_{B,A}\ a\right) = a.$$

将理想算子（读作“理想”）应用于元素 x： A，会得到一个由基元素构成的 ω-链，这些基元素通过注入映射到 A 后收敛于 x。严格来说，我们只需要求理想算子生成有向集，但为了将来发展出一种关于连续扩张的一般可计算性的实用概念，我们选择将其限制为 ω-链。当上下文明确时，我们省略理想算子和 incl 的下标。

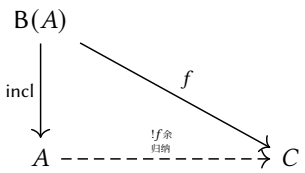
定义 11 (代数 CPO)。 设 A 是一个 CPO，且 B 是一个有序类型。 A 当 是紧致且在 A中稠密时，它是一个带有基 B 的代数 CPO（aCPO）。

我们用 $\mathsf{B}(A)$ 表示代数 CPO A的基。代数 CPO 在构造积和和时是封闭的。类型 $A \to C$ 是代数 CPO，当且仅当 A 是有限的，且 C 是代数 CPO。

2.3 连续扩张

AlgCo 框架的一个关键思想是，将代数 CPO 上的连续函数定义为较简单的、作用于基元素的单调函数的连续扩张；其特征由以下引理给出（基于[9, 引理 5。24]）：

引理 12 (连续扩张)。 设 A 是一个代数 CPO， C 是一个 CPO，且 f: $\mathsf{B}(A)$ $\to$ C 是一个单调函数。那么，存在唯一的连续函数f_{co}： $A \to C$（基函数 f 的连续扩张），使得 $f_{\mathrm{co}} \circ \mathsf{incl} = f$。也就是说，以下图表交换：



引理的主要内容 12 分为两方面：

存在性。任意定义在代数 CPO 的基上的单调函数 $f : \mathrm{B}(A) \to C$ 都可以扩展为连续函数 $f^{\mathrm{co}} : A \to C$，其定义域为整个域 A。

唯一性。该扩张是唯一的。任何能够补全图表的连续函数都必须等于 f^{co}。

的存在性构成了代数 CPO 的连续消去方案：要在代数 CPO A上定义连续函数，只需改为定义一个单调基函数 $f : \mathrm{B}(A) \to C$(通常通过归纳定义，参见第4 节和第6节)，然后将其扩张到 $f^{\mathrm{co}} : A \to C$。

此外，的唯一性意味着每个连续函数 $g : A \to C$都可以表示为连续扩张 $f^{\mathrm{co}} : A \to C$，其中某个单调基函数 $f : \mathrm{B}(A) \to C$对应的扩张即为 $f = g \circ \mathrm{incl}$ (参见推论73)。

引理12告诉我们，代数 CPO 上的连续函数恰好是那些可以“轻松”定义的函数（即基函数的扩张），并由此导出一系列有用的证明原则（例如引理13、定理14以及附录A中的更多内容），可用于对这类函数进行推理。例如，下面的引理常用于证明连续扩张的等价性：

引理 13 (连续扩张的等价性)。设 A 是一个代数 CPO，C是任意 CPO，且 $f : \mathrm{B}(A) \to C$ 和 $g : \mathrm{B}(A) \to C$ 是定义在A的基上的单调函数。那么，

$$f = g \Rightarrow f^{\mathrm{co}} = g^{\mathrm{co}}.$$

引理 13 在实践中极为有用：要证明 $f \text{余} a = g^{\mathrm{co}} a$ 对所有 $a : A$都成立，只需证明 $f\ b = g\ b$ 对所有 $b : \mathrm{B}(A)$ 都成立，而这通常可以通过直接归纳证明。

除了连续扩张是连续的这一事实之外（这一点不应被低估，因为连续性的临时证明往往既困难又耗时），我们还有一个关于连续扩张的融合律：

定理 14 (融合)。设 A 是一个代数 *CPO*，B 和 C 是连续偏序集，$f : \mathrm{B}(A) \to B$ 是一个单调函数，且 $g : B \to C$ 是一个连续函数。那么，$g \circ f^{\mathrm{co}} = (g \circ f)^{\mathrm{co}}$。

2.4 余连续扩张

我们还可以将代数 CPO 上的余连续函数视为反单调函数的余连续扩张
每个关于连续扩张的证明原理，都有一个关于余连续扩张的对应原理。

2.5 （余）连续性质

陪域为 $\mathbb{P}$ 的连续函数称为连续性质。代数 CPO A 上的连续性质可以定义为 $\mathrm{B}(A)$ 上单调性质 的连续扩张。将单调性的定义特化到代数 CPOA 和陪域 $\mathbb{P}$，可知性质 $P : \mathrm{B}(A) \to \mathbb{P}$ 在以下情况下是单调的：

$$x \sqsubseteq y \Rightarrow P\ x \Rightarrow P\ y.$$

也就是说，当某个近似 x 满足性质 P 时，所有对 x 的进一步精化（所有满足 $x \sqsubseteq y$ 的 y）也都满足该性质。那性质连续又是什么意思呢？回想一下，当对每个 ω-链 $C\colon \mathbb{N} \to A$，都有 $f(\sup C) \simeq \sup(f \circ C)$ 时，函数 $f\colon A \to B$ 是 ω-连续的。将其特化为 $P^{\mathrm{co}}\colon A \to \mathbb{P}$：

$$P^{\mathrm{co}}(\sup C) \iff \sup(P^{\mathrm{co}} \circ C)$$

或者等价地（根据定义 10 和注释 1）：

$$P^{\mathrm{co}}\, a \iff \exists i.\, P(\mathrm{idl}\, a\, i). \tag{1}$$

我们看到，性质 P^{co} 在元素 a 上成立：A 当且仅当它在 $\mathrm{B}(A)$ 上的限制对 a 的某个近似成立。这引出了连续性质的定义：

定义 15（连续性质）。设 A 是一个代数 CPO。谓词 $P : A \to \mathbb{P}$ 在 A 上是连续性质，当且仅当对所有 $a :\ A$：

$$P\, a \iff \exists i.\, P(\mathrm{incl}(\mathrm{idl}\, a\, i)).$$

可判定性质的连续扩张是半可判定的。我们还有对偶概念余连续性质：

定义 16（余连续性质）。设 A 是一个代数 CPO。谓词 $P : A \to \mathbb{P}$ 在 A 上是余连续性质，当且仅当对所有 $a :\ A$：

$$P\, a \iff \forall i.\, P(\mathrm{incl}(\mathrm{idl}\, a\, i)).$$

3 余自然数

我们的第一个余归纳代数 CPO 的具体例子是类型 $\mathbb{N}^{\mathrm{co}}$ 余自然数，即扩展了“无穷远点” $\omega_{\mathbb{N}}$ 的自然数。

定义 17 ($\mathbb{N}^{\mathrm{co}}$ (余自然数))。以归纳方式定义类型 $\mathbb{N}^{\mathrm{co}}$ 余自然数，定义规则如下：

$$\frac{}{\mathbf{cozero} :\ \mathbb{N}^{\mathrm{co}}}\ \text{余自然数零} \qquad \frac{n :\ \mathbb{N}^{\mathrm{co}}}{\mathbf{余后继}\ n :\ \mathbb{N}^{\mathrm{co}}}\ \text{余自然数后继}$$

定义18 ($\omega_{\mathbb{N}}$)。将无限余自然数 $\omega_{\mathbb{N}}$ 余归纳地定义如下：

$$\omega_{\mathbb{N}} \triangleq \mathbf{cosucc}\ \omega_{\mathbb{N}}$$

上的序关系是自然数的通常序关系，并将其扩展为使得 $n \sqsubseteq \omega_{\mathbb{N}}$ 对所有 $n :\ \mathbb{N}^{\mathrm{co}}$。为避免过度依赖广义重写 [20],，我们引入一个外延性公理，由序等价推出命题相等。第 4、5和6节分别对流、树和前缀树的余归纳类型作了同样处理（讨论见附录B ）。

公理 1 (余自然数外延性)。 $\forall n\, m :\ \mathbb{N}^{\mathrm{co}},\ \ n \simeq_{\mathbb{N}^{\mathrm{co}}} m \Rightarrow n = m$。

类型 $\mathbb{N}$ 的自然数构成了 $\mathbb{N}^{\mathrm{co}}$的紧致基，其中包含映射$\mathrm{incl}_{\mathbb{N},\ \mathbb{N}^{\mathrm{co}}} :\ \mathbb{N} \to \mathbb{N}^{\mathrm{co}}$ 将自然数单射嵌入到 $\mathbb{N}^{\mathrm{co}}$中，而 $\mathrm{idl}_{\mathbb{N},\ \mathbb{N}^{\mathrm{co}}} :\ \mathbb{N}^{\mathrm{co}} \to \mathbb{N} \to \mathbb{N}$生成余自然数的有限近似所构成的收敛链。

注 4（ℕ是紧致的）。

注记 5（ℕ 在 $\mathbb{N}^{\mathrm{co}}$ 中稠密。 对于所有 $n:\ \mathbb{N}^{\mathrm{co}}$,

$$\mathrm{idl}\ n \text{ is an } \omega\text{-chain, } \textit{and}$$
$$\sup(\mathrm{incl} \circ \mathrm{idl}\ n) = n.$$

注记 6（$\mathbb{N}^{\mathrm{co}}$ 是一个有底元素 **cozero** 且基为 ℕ 的有基点代数连续偏序集。

3.1 无限燃料

当一个函数对于其任何参数的结构都不是归纳定义时，一种常见的技巧是改为对一个单独的 ℕ 参数（即燃料）进行归纳定义，并确保始终提供足够的燃料，使函数能够完成任务。我们通过以下「iter」构造来定义带燃料计算：从初始元素 z开始，反复应用函数 f，直到燃料耗尽。

定义 19（iter）。 对于类型 A, $z:\ A$, $f:\ A \to A$, 以及 $n:\ \mathbb{N}$, 定义 $\mathrm{iter}\ z\ f\ n:\ A$, 通过对 n进行归纳：

$$\begin{array}{l} \mathrm{iter}\ z\ f : \mathbb{N} \to A \\ \hline \mathrm{O} \triangleq z \\ \mathrm{S}\ n \triangleq f\ (\mathrm{iter}\ f\ n) \end{array}$$

通过取带燃料迭代的连续扩张，我们将其域扩展为包含 $\omega_{\mathbb{N}}$，从而可以为它提供无限量的燃料！

定义 20 (coiter)。 对于有底类型 A 和 $f:\ A \to A$, 定义 $\mathrm{coiter}\ f:\ \mathbb{N}^{\mathrm{co}} \to A$如下：

$$\mathrm{coiter}\ f \triangleq (\mathrm{iter}\ \bot_A\ f)^{\mathrm{co}}.$$

我们将这种技术称为惰性余迭代，并在第 5.1 节中用它在正则语言的余归纳编码上实现 Kleene 闭包算子。

4 流

本节将流 （或余列表）定义为余归纳代数 CPO（定义 21），并定义一些重要的运算和谓词（包括常常令人棘手的“filter”运算）（第 4.1 节），说明如何运用 AlgCo 的基本证明原则对流进行推理（第 4.2 节），最后验证余归纳版本的埃拉托斯特尼筛法（第4.4节）。

作为代数 *CPO* 的流。 我们对余归纳列表的定义与标准流定义（例如， [4]）略有不同，因为其中包含一个底元素：

定义 21 (流)。 通过以下形成规则，余归纳地定义元素类型为 A 的流类型L^*_A：

$$\frac{}{\bot_{\mathcal{L}^*_A} : \mathrm{L}^*_A}\ \text{流底} \qquad \frac{a : A \qquad l : \mathrm{L}^*_A}{\textbf{余构造子}\ a\ l : \mathcal{L}^*_A}\ \text{流构造子}$$

我们谨慎地不把 $\bot_{\mathcal{L}_A^*}$ 视为流的一个简单 **nil** 构造子，而是将其视为一个未定义的或发散的流，它会永远循环且不产生任何输出。流按一种直接的结构前缀关系排序：

定义 22 (流序)。 余归纳地定义 $\sqsubseteq_{\mathcal{L}_A^*} : \mathcal{L}_A^* \to \mathcal{L}_A^* \to \mathbb{P}$，其推理规则如下：

$$\frac{l : \mathcal{L}_A^*}{\bot_{\mathcal{L}_A^*} \sqsubseteq_{\mathcal{L}_A^*} l}\ \sqsubseteq_{\mathcal{L}_A^*}\text{-BOT} \qquad \frac{a : A \qquad l_1 \sqsubseteq_{\mathcal{L}_A^*} l_2}{\mathbf{cocons}\ a\ l_1 \sqsubseteq_{\mathcal{L}_A^*} \mathbf{cocons}\ a\ l_2}\ \sqsubseteq_{\mathcal{L}_A^*}\text{-CONS}$$

直观来说，当 l_1 是 $\bot$ 终止的 l_2 的有限近似，或者 l_1 与 l_2 相等时，有 $l_1 \sqsubseteq_{\mathcal{L}_A^*} l_2$。$\mathrm{L}_A^*$ 的一个紧基由列表的归纳类型 L_A 给出（其构造子为 **空列表** 和 **cons**），并带有前缀序：

定义 23 (列表序)。 归纳地定义 $\sqsubseteq_{\mathcal{L}_A} : \mathcal{L}_A \to \mathcal{L}_A \to \mathbb{P}$，其推理规则如下：

$$\frac{l : \mathcal{L}_A}{\mathbf{nil} \sqsubseteq_{\mathcal{L}_A} l}\ \sqsubseteq_{\mathcal{L}_A}\text{-NIL} \qquad \frac{a : A \qquad l_1 \sqsubseteq_{\mathcal{L}_A} l_2}{\mathbf{cons}\ a\ l_1 \sqsubseteq_{\mathcal{L}_A} \mathbf{cons}\ a\ l_2}\ \sqsubseteq_{\mathcal{L}_A}\text{-CONS}$$

备注 7($\mathcal{L}_A$ 是紧致的)。

备注 8 ($\mathcal{L}_A$ 在 $\mathcal{L}_A^*$ 中稠密。 设 A 是一个类型。那么，对于所有的 $a : \mathcal{L}_A^*$，$\mathrm{idl}\, a$ 是一个 ω-链，且 $\sup\,(\mathrm{incl} \circ \mathrm{idl}\ a) = a$。

备注7 和 8 表明 $\mathcal{L}_A$ 是 $\mathcal{L}_A^*$的一个紧基。此外，对于所有的 l，都有: $\mathcal{L}_A^*$，因此：

备注 9（$\mathcal{L}_A^*$ 是一个具有基 $\mathcal{L}_A$的有基点代数连续偏序集。

例 24 （自然数）。 对于 $n :\ \mathbb{N}$，通过余归纳地定义从 自然数 $n :\ \mathcal{L}_{\mathbb{N}}^*$ 开始的自然数流 n：

$$\mathrm{nats}\ n \triangleq \mathbf{cocons}\ n\ (\mathrm{nats}\ (\mathrm{S}\ n))$$

4.1 共折叠

许多在流上的连续扩张都具有共同的计算结构。本节将对这一共同模式（“余折叠”）进行抽象，并利用它推导流上标准运算的定义和计算规则。在第 4.5 节中，我们将为余折叠提供一个提取原语，以便在 Haskell 中进行惰性执行。余折叠的基础是列表上的标准右结合折叠算子：

定义 25 (折叠)。 对于类型 A 和 B，$z :\ B$，$f :\ A \to B \to B$，以及 $l :\ \mathcal{L}_A$，定义折叠 $z\ f\ l :\ B$ 通过对 l 进行归纳：

$$\begin{array}{l|l} \mathrm{fold}\ z\ f : \mathcal{L}_A \to B & \\ \hline \mathbf{nil} & \triangleq\ z \\ \mathbf{cons}\ a\ l & \triangleq\ f\ a\ (\mathrm{fold}\ f\ l) \end{array}$$

形如 折叠 $z\ f$ 的运算通常称为 *catamorphism* [21], , 或简称折叠。我们引入余折叠：定义在形如 (折叠 $\bot\ f)^{\mathrm{co}}$ 的流上的连续函数，其中函数 f 单调；以及 反余折叠 (写作 ^cofold)：形如(折叠 $\top\ f)^{\mathrm{cb}}$ 的余连续函数，其中 f反单调。

定义 26 (cofold)。 对于类型 A、有底类型 B和 f： $A \to B \to B$，定义 cofold f： $\mathcal{L}_A^* \to B$ 如下：

$$\mathrm{cofold}\ f \triangleq (\mathrm{fold}\ \bot_B\ f)^{\mathrm{co}}$$

或者，当 B 有最大元 $\top_B$时，定义 ^cofold f: $\mathrm{L}_A^* \to B$:

$$\mathrm{c\hat{o}fold}\ f \triangleq (\mathrm{fold}\ \top_B\ f)^{\mathrm{cb}}.$$

cofold f在 f单调时是连续的，而且 ^cofold f在 f反单调时是余连续的。

余折叠计算。 以下通用计算引理可用于推导余折叠的计算规则。

引理 27 (cofold计算)。 设 A 是一个类型， B 是一个有序类型，且 f： $A \to B \to B$。那么，如果 B 是一个有底连续偏序，且 $f\ a$ 对每个 a都连续，

$$\mathrm{cofold}\ f\ (\mathbf{cocons}\ a\ l) \simeq f\ a\ (\mathrm{cofold}\ f\ l)$$

或者，如果 B 是一个具有最大元的下连续偏序，且 $f\ a$ 对每个 a都 l-连续，

$$\mathrm{c\hat{o}fold}\ f\ (\mathbf{cocons}\ a\ l) \simeq f\ a\ (\mathrm{c\hat{o}fold}\ f\ l).$$

请记住，$\simeq$表示序等价（定义 2），它通常蕴含命题相等（例如，对于 $\mathbb{R}^{\infty}_{\geq 0}$，依据反对称性；对于 $\mathbb{N}^{\mathrm{co}}$，依据公理 1）。

余折叠示例。这里我们给出一些流上的余折叠示例，并根据引理27推导它们的计算规则。有时我们会为正在扩展的基函数指定明确的名称，以便更易于阐述相关证明（例如，定义 32 和引理 33）。第一个例子是余折叠 长度$\mathcal{L}^*_{\mathrm{A}}$，它将余树映射到 $\mathbb{N}^{\mathrm{co}}$，使得任意无限流的长度都等于 $\omega_{\mathbb{N}}$。

定义 28 (长度$\mathcal{L}^*_{\mathrm{A}}$)。对于类型 A，定义 长度$\mathcal{L}^*_{\mathrm{A}}$: $\mathcal{L}_A^* \to \mathbb{N}^{\mathrm{co}}$，(折叠**cozero**($\lambda$_.**余后继**))^{co}，其计算规则为：计算规则：

$$\mathrm{length}_{\mathcal{L}^*_{\mathrm{A}}}\ (\mathbf{cocons}\ \_\ l) = \mathbf{cosucc}\ (\mathrm{length}_{\mathcal{L}^*_{\mathrm{A}}}\ l).$$

流上的谓词量化。流上的存在量化由以下连续谓词给出：

定义 29 ($\exists_P^{\mathrm{co}}$)。 对于类型 A 和谓词 P： $A \to \mathbb{P}$，定义 $\exists_P^{\mathrm{co}}$: $\mathrm{L}_A^* \to \mathbb{P}$，(折叠 $\bot$ ($\lambda a.\ \lambda Q.\ P\ a \vee Q$))^{co}，并附有导出的引入规则和消去规则：

$$\frac{P\ a \qquad l : \mathcal{L}_A^*}{\exists_P^{\mathrm{co}}\ (\mathbf{cocons}\ a\ l)}\ \exists_P^{\mathrm{co}}\text{-INTRO-1} \qquad \frac{a : A \qquad \exists_P^{\mathrm{co}}\ l}{\exists_P^{\mathrm{co}}\ (\mathbf{cocons}\ a\ l)}\ \exists_P^{\mathrm{co}}\text{-INTRO-2} \qquad \frac{\exists_P^{\mathrm{co}}\ (\mathbf{cocons}\ a\ l)}{P\ a \vee \exists_P^{\mathrm{co}}\ l}\ \exists_P^{\mathrm{co}}\text{-ELIM}$$

以及通过余连续谓词进行全称量化：

定义30 ($\forall^{\hat{}}_{\mathrm{co}P}$). 对于类型 A 和谓词 P: $A \to \mathbb{P}$，定义 $\forall^{\mathrm{co}}_P$: $\mathrm{L}^*_A \to \mathbb{P}$, (折叠 $\top$ ($\lambda a.\, \lambda Q.\, P\ a \wedge Q$))^{co}，并导出引入规则和消去规则：

$$\frac{P\ a \qquad \forall^{\mathrm{co}}_P\ l}{\forall^{\hat{\mathrm{co}}}_{\mathrm{余}P}(\textbf{余构造子}\ a\ l)}\ \forall^{\mathrm{co}}_P\text{-引入} \qquad \frac{\forall^{\hat{\mathrm{co}}}_{\mathrm{余}P}(\textbf{余构造子}\ a\ l)}{P\ a}\ \forall^{\mathrm{co}}_P\text{-消去-1} \qquad \frac{\forall^{\hat{\mathrm{co}}}_{\mathrm{余}P}(\textbf{余构造子}\ a\ l)}{\forall^{\mathrm{co}}_P\ l}\ \forall^{\mathrm{co}}_P\text{-消去-2}$$

定义 29 和 30用于证明定理37 和 38中埃拉托斯特尼筛法正确性的相关性质。

作为连续扩展的序关系。尽管要首先使用 AlgCo 框架的机制，就必须已经定义了流上的原始序关系（定义 22），但我们可以将其重新定义为连续扩张，并证明它与原始定义等价。

定义 31 ($\sqsubseteq^{\hat{}}_{\mathrm{共}\mathcal{L}_A}$)。对于类型 A，定义 $\sqsubseteq^{\hat{}}_{\mathrm{共}\mathcal{L}_A}$: $\mathcal{L}^*_A \to \mathcal{L}^*_A \to \mathbb{P}$，其中 $\sqsubseteq_{\mathcal{L}_A}$: $\mathcal{L}_A \to \mathcal{L}^*_A$，

$$\begin{aligned}&\mathrm{fold}\ (\lambda\_.\ \top)\ (\lambda a.\, \lambda f.\, \lambda l.\ \mathrm{match}\ l\ \mathrm{with}\\ &\qquad\qquad \mid \bot_{\mathcal{L}^*_A} \Rightarrow \bot\\ &\qquad\qquad \mid \textbf{cocons}\ b\ l' \Rightarrow a = b \wedge f\ l'\\ &\qquad\qquad \mathrm{end})\end{aligned}$$

注 10 ($\sqsubseteq^{\hat{}}_{\mathrm{共}\mathcal{L}_A}$ 与 $\sqsubseteq^*_{\mathcal{L}_A}$相同)。设 A 为一种类型。那么，$\forall\ l_1\ l_2 : \mathrm{L}^*_A,\ l_1 \sqsubseteq^{\hat{}}_{\mathrm{共}\mathcal{L}_A} l_2 \iff l_1 \sqsubseteq_{\mathcal{L}^*_A} l_2$.

定义 31 在实践中通常比原始序关系（定义 22）更方便，因为它可以与其他连续扩展融合（由定理 14）。

余滤波。使用给定谓词筛选余归纳流是一项出了名的棘手工作，通常需要为流类型扩展一个特殊构造子，以表示所谓的“静默步”（例如，[10],这会造成大量定义上的繁杂内容和性能开销），或者像 [22] 中那样使用复杂的混合良基归纳—余归纳方案。我们可以轻松地将 filter 定义为 cofold：

定义 32 ($\mathrm{filter}^{\mathcal{L}^*_A}$)。对于类型 A 和 f: $A \to \mathbb{B}$，定义 $\mathrm{filter}_{\mathcal{L}^*_A}\ f$: $\mathcal{L}^*_A \to \mathcal{L}^*_A$，$(\mathrm{filter}_{\mathcal{L}_A}\ f)^{\mathrm{co}}$，其中：

$$\mathrm{filter}_{\mathcal{L}_A}\ f \triangleq \mathrm{fold}\ \bot\ (\lambda a.\, \lambda l.\ \mathrm{if}\ f\ a\ \mathrm{then}\ \textbf{cocons}\ a\ l\ \mathrm{else}\ l)$$

附带计算规则：

$$\begin{aligned}&\mathrm{filter}_{\mathcal{L}^*_A}\ f\ (\textbf{cocons}\ a\ l) =\\ &\quad \mathrm{if} f\ a\ \mathrm{then}\ \textbf{cocons}\ a\ (\mathrm{filter}_{\mathcal{L}^*_A}\ f\ l)\ \mathrm{else}\ \mathrm{filter}_{\mathcal{L}^*_A}\ f\ l.\end{aligned}$$

4.2 利用融合证明

这里展示如何使用 AlgCo 证明 $\text{filter}_{\mathcal{L}_A^*}$ 的交换性。首先，我们使用定理 14 将等式两边融合，然后应用引理13 将证明归约为对基元素的归纳。

引理 33 ($\text{filter}_{\mathcal{L}_A^*}$ 具有交换性)。 设 A 是一个类型，并且 $f, g : A \to \mathbb{B}$。那么，

$$\forall\, l : \mathcal{L}_A^*, \text{filter}_{\mathcal{L}_A^*}\ f\ (\text{filter}_{\mathcal{L}_A^*}\ g\ l) = \text{filter}_{\mathcal{L}_A^*}\ g\ (\text{filter}_{\mathcal{L}_A^*}\ f\ l).$$

证明。展开 $\text{filter}_{\mathcal{L}_A^*}$ 的定义，我们要证明：

$$\text{filter}_{\mathcal{L}_A^*}\ f \circ (\text{filter}_{\mathcal{L}_A}\ g)^{\text{co}} = \text{filter}_{\mathcal{L}_A^*}\ g \circ (\text{filter}_{\mathcal{L}_A}\ f)^{\text{co}}.$$

首先，根据定理14，我们将等式两边分别融合：

$$(\text{filter}_{\mathcal{L}_A^*}\ f \circ \text{filter}_{\mathcal{L}_A}\ g)^{\text{co}} = (\text{filter}_{\mathcal{L}_A^*}\ g \circ \text{filter}_{\mathcal{L}_A}\ f)^{\text{co}}$$

接着由推论 13 可得：

$$\forall\, l : \mathcal{L}_A, \text{filter}_{\mathcal{L}_A^*}\ f\ (\text{filter}_{\mathcal{L}_A}\ g\ l) = \text{filter}_{\mathcal{L}_A^*}\ g\ (\text{filter}_{\mathcal{L}_A}\ f\ l)$$

这可通过对 l 进行直接归纳，并应用 $\text{filter}_{\mathcal{L}_A^*}$ 计算规则（定义 32）得出。 □

4.3 证明（余）连续性质

AlgCo 框架一个令人惊讶却优雅的特点是：将代数 CPO 上的谓词视为连续函数的特例后，我们便能轻松地将它们与连续扩张组合，并通过融合大幅简化证明。下面以此为例，证明对于任意 P 和 Q，$\text{filter}_{\mathcal{L}_A^*}\ P$ 都保持 $\forall_Q^{\text{co}}$：

引理 34 ($\text{filter}_{\mathcal{L}_A^*}$ 保持 $\forall_Q^{\text{co}}$)。 设 A 是一个类型，$P : A \to \mathbb{P}$，$Q : A \to \mathbb{P}$，且 $s : \mathrm{L}_A^*$ 满足 $\forall\, {}^\wedge_{\text{余}Q}\ s$。那么，$\forall_Q^{\text{co}}(\text{filter}_{\mathcal{L}_A^*}\ P\ s)$。

证明。展开 $\text{filter}_{\mathcal{L}_A^*}$ 的定义后，目标变为：

$$\forall_Q^{\text{co}}\ ((\text{filter}_{\mathcal{L}_A}\ P)^{\text{co}}\ s).$$

通过融合 这等价于：

$$(\forall_Q \circ \text{filter}_{\mathcal{L}_A}\ P)^{\text{co}}\ s$$

根据 ^余引入 (定义 16)，可由以下内容推出：

$$\forall i : \mathbb{N},\ \forall_Q\ (\text{filter}_{\mathcal{L}_A}\ P\ (\text{idl}\ s\ i)).$$

固定 i。根据 $\forall l :\ \mathcal{L}_A,\ \forall_Q l \Rightarrow \forall_Q^{\text{co}}\ (\text{filter}_{\mathcal{L}_A} P l)$ （对 l作直接归纳即可），因此只需证明：

$$\forall_Q\ (\text{idl}\ s\ i)$$

由 ^余归纳消去 （定义 16）可得 $\forall_Q^{\text{co}}\ s$. □

4.4 埃拉托斯特尼筛法

埃拉托斯特尼筛法是一种古老的算法，用于生成不超过给定上限的素数序列。筛法的一个无穷版本（有时称为“不忠实筛法” [23]），虽然常用于展示惰性函数式编程的优雅，但由于它在流上使用了 filter 过滤操作，因此难以在可靠的类型论（例如 Coq 的类型论）中复现。我们将筛法定义为列表上的单调函数，并将其连续扩展到流：

定义 35 (筛法)。定义 筛法 : $\mathcal{L}^*_{\mathbb{Z}}$，筛法_aux (自然数2)，其中：

$$\text{sieve\_aux} : \mathcal{L}^*_{\mathbb{Z}} \to \mathcal{L}^*_{\mathbb{Z}} \triangleq \text{sieve\_aux}_{\mathcal{L}_{\mathbb{A}}}{}^{\text{co}}, \textit{ where}$$

$$\begin{aligned}&\text{sieve\_aux}_{\mathcal{L}_{\mathbb{A}}} \triangleq \\ &\text{fold} \perp_{\mathcal{L}^*_{\mathbb{Z}}} (\lambda n.\, \lambda l.\, \textbf{cocons}\ n\ (\text{filter}_{\mathcal{L}^*_{\mathbb{A}}}\ (\lambda m.\, m \bmod n \neq 0)\ l))\end{aligned}$$

带有 计算规则：

$$\begin{aligned}&\text{sieve\_aux}\ (\textbf{cocons}\ n\ l) = \\ &\quad \textbf{cocons}\ n\ (\text{filter}_{\mathcal{L}^*_{\mathbb{A}}}\ (\lambda m.\, m \bmod n \neq 0)\ (\text{sieve\_aux}\ l)).\end{aligned}$$

我们证明 筛法 相对于素数是完备的（定理 37），且是可靠的（定理38），并且按升序生成这些素数，不含重复项（定理 39）。

定义 36 (is_prime)。定义 is_prime : $\mathbb{Z} \to \mathbb{P}$ 如下：

$$\text{is\_prime}\ n \triangleq 1 < n \wedge \forall\, m,\ 1 < m \to n \neq m \to n \bmod m \neq 0.$$

定理 37 (筛法 相对于素数是完备的)。设 n: $\mathbb{Z}$ 满足 is_prime$n$。那么，$\exists^{\text{co}}_{\text{eq}\ n}$ 筛法。也就是说，每个素数 n 都会出现在由 筛法生成的流中。

定理 37说明筛法流具有连续性质（定义 15），因此只需证明对每个 n，它对筛法的某个有限近似成立。

定理 38 （筛法对于素数是可靠的）。$\forall^{\text{co}}_{\text{is\_prime}}$ 筛法。也就是说，筛法生成的流中出现的每个数字筛法都是素数。

定理38 说明筛法流具有余连续性质（定义 16），因此只需证明它对筛法的所有有限近似成立。

定理 39 （筛法已排序且不含重复项）。

定理 39 通过证明流 nats2 严格递增，且连续扩张 sieve_aux(定义 35) 严格保序得证。

```
cofold  =
  \ o p f l ->
    casel  of
      Conil ->bot o p
      对偶构造子 a l' ->  f a (cofold o p f l')
```

图 1. cofold 的 Haskell 提取原语。参数 o 和 p 是 OType 和 PType 陪域序关系的实例字典对象。

4.5 提取筛法

连续扩张不可计算，因为sup (第 2.1节）并非构造性的。尽管如此，仍可在 Haskell 中为流上的余折叠实现提取原语（如图 1所示），以支持惰性执行。余折叠提取原语的正确性由引理 27中的余折叠计算规则保证；该规则的特例可以在准备执行的流上进行显式检验（例如定义 35中的筛法计算规则）。

连续扩张的可计算性。图 1 所示的提取原语为余折叠提供了一种计算解释，但这种解释并不完全正确，因为从连续扩张中提取的程序并不能保证对每个输入都终止（参见 [16] 和 [24]中提取的不动点算子）。这是因为连续谓词仅为半可判定（见第 2.5节），而连续性通常只对应于部分可计算性。例如，连续的布尔值谓词 bad $\triangleq (\lambda\_\ x.\ x)^{\mathrm{co}} : \mathcal{L}^*_{\mathbb{B}} \to \mathbb{B}$ 对每个流都会发散。

不过，我们可以为由连续扩张生成的流定义一种 生产性 概念（对于我们的筛法，其证明可由欧几里得关于 素数无穷性 [25]的定理得出）：

定义 40 (具有生产性)。 对于任意类型 A，若流 $s : \mathcal{L}^*_A$ 被称为（无限）具有生产性 当 长度$\mathcal{L}^*_{\mathrm{A}}\ s = \omega_{\mathbb{N}}$。

我们证明筛法具有生产性：

定理 41 (筛法 具有生产性)。 长度$\mathcal{L}^*_{\mathbb{Z}}$ 筛法 $= \omega_{\mathbb{N}}$。

但一般而言，这种生产性概念不足以绝对保证不会发生发散。为了解原因，请考虑由余归纳定义的流：坏_流 $\triangleq$ **余构造子** (坏 (自然数0)) 坏_流。坏坏_流中的每个元素都可证明等于 **假**，而且 坏_流 具有生产性，但任何尝试提取其元素并进行计算的操作都会立即发散。

5 余归纳前缀树

形式语言通常定义为字母表 Σ 上的字符串集合，也可以用在 Σ 上分支的无限前缀树（余归纳字典树）优雅地表示。形式语言的余归纳字典树表示通过将每个节点标记为接受或拒绝空串，编码一个用于识别该语言中字符串的自动机

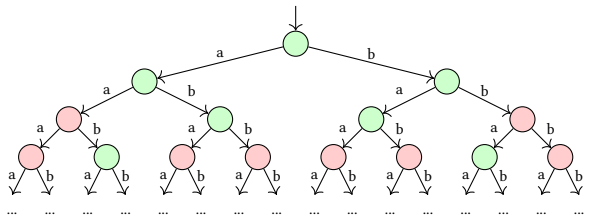


图 2. 字母表 {a, b} 上正则语言 '$\epsilon + a^*b + b^*a$' 的余归纳字典树编码。绿色节点表示编码自动机的接受状态。

ϵ 并通过余归纳延续，直接在节点处编码相对于字母表中每个符号的 Brzozowski 导数 [26]。本节将余归纳字典树定义为代数 CPO 类型，构造其上的正则运算，并证明这些运算满足克莱尼代数的公理，从而能够提取经过验证的 Haskell 正则表达式匹配器（例如，参见文件 *RE_test.v*）。

形式语言作为代数 *CPO*。 余归纳字典树的类型是根据有限字母表类型（用于索引每个节点的子节点）和用于节点标签的带底元有序类型定义的。为便于说明，我们将字母表固定为有限字母表 Σ，标签类型固定为 $\mathbb{B}$。

定义 42 (语言$_\Sigma$)。 通过以下形成规则，定义$_\Sigma$ 为有限字母表 Σ上的语言类型：

$$\frac{b : \mathbb{B} \qquad \forall\, a : \Sigma,\ k\ a : \mathsf{lang}_\Sigma}{\mathbf{lnode}\ b\ k : \mathsf{lang}_\Sigma}\ \textsc{lang-node}$$

定义 43 (语言$_\Sigma$ 序)。 通过以下推理规则，定义 $\sqsubseteq_{\mathsf{lang}_\Sigma}$: 语言$_\Sigma$ → 语言$_\Sigma$ → $\mathbb{P}$：

$$\frac{b_1 \sqsubseteq_{\mathbb{B}} b_2 \qquad \forall\, a : \Sigma,\ f\ a \sqsubseteq_{\mathsf{lang}_\Sigma} g\ a}{\mathbf{lnode}\ b_1\ f \sqsubseteq_{\mathsf{lang}_\Sigma} \mathbf{lnode}\ b_2\ g}\ \sqsubseteq_{\mathsf{lang}_\Sigma}\textsc{-node}$$

与本文中的其他余归纳类型不同，这里没有构造子可用于 $\bot$，实际上也没有任何有限构造子（假设字母表非空）。因此，每棵余归纳字典树都是无限的，且结构上彼此全等，仅节点标签的值有所不同。所有标签均为假的空语言充当底元素：

定义 44 ($\emptyset$语言)。 余归纳地定义空语言 $\emptyset$ 如下： $\emptyset$ ， **lnodefalse** ($\lambda\_.\ \emptyset$)。

尽管所有余归纳字典树都是无限的，我们可以将那些仅在有限深度以内有意义定义（此后均等于 $\emptyset$）的余归纳字典树子集作为紧基。这类余归纳字典树是有限可逼近的，因此是紧致的，并且可以用一种归纳类型表示，其中一个 $\bot$ 构造子代替 $\emptyset$:

定义45 (tlang$_\Sigma$)。 通过以下形成规则归纳地定义 tlang$_\Sigma$ 上的有限前缀树语言类型，其中字母表为Σ：

$$\frac{}{\bot_{\text{tlang}_\Sigma} : \text{tlang}_\Sigma}\ \text{tlang 底元} \qquad \frac{b : \mathbb{B} \quad \forall\, a : \Sigma,\ k\, a : \text{tlang}_\Sigma}{\textbf{字典树语言节点}\ k : \text{tlang}_\Sigma}\ \text{字典树节点}$$

然而，如果我们按照通常的方式（例如，定义 tlang$_\Sigma$ 22）定义 lang$_\Sigma$ 上的序关系，则映射到其中的包含映射tlang$_\Sigma$ 将不会是单射（在序等价意义下），因为 $\emptyset$ 的多个元素（其所有近似）都会被映射到$\emptyset$。我们通过如下方式定义序关系来解决这一问题：将底余字典树的所有近似压缩为同一个等价类。

定义 46 （tlang$_\Sigma$ 序关系）。 通过以下推理规则归纳地定义 $\sqsubseteq_{\text{tlang}_\Sigma}$： $\text{tlang}_\Sigma \to \text{tlang}_\Sigma \to \mathbb{P}$：

$$\frac{\text{是\_底}\ t_1 \qquad t_2 : \text{tlang}_\Sigma}{t_1 \sqsubseteq_{\text{tlang}_\Sigma} t_2}\ \sqsubseteq_{\text{tlang}_\Sigma}\text{-底} \qquad \frac{b_1 \sqsubseteq b_2 \qquad \forall\, a : \Sigma,\ f\, a \sqsubseteq_{\text{tlang}_\Sigma} g\, a}{\textbf{字典树语言节点}\ b_1\ f \sqsubseteq_{\text{tlang}_\Sigma} g_\Sigma\ \textbf{字典树语言节点}\ b_2\ g}\ \sqsubseteq_{\text{tlang}_\Sigma}\text{-节点}$$

其中 是_底节点： $\text{tlang}_\Sigma \to \mathbb{P}$ 通过归纳定义如下：

$$\frac{}{\text{is\_bot}\ \bot_{\text{tlang}_\Sigma}}\ \text{is\_bot-BOT} \qquad \frac{\forall\, a : \Sigma,\ \text{是\_底节点}\ (k\, a)}{\text{是\_底}(\textbf{字典树语言节点 false}\ k)}\ \text{is\_底节点}$$

注释11（tlang$_\Sigma$ 是紧致的）。

注释12（lang$_\Sigma$ 是一个以 tlang$_\Sigma$ 为基的有基点代数连续偏序集。

5.1 正则语言

定义 47 (lang$_\Sigma$ 成员关系)。 对于 t: lang$_\Sigma$ 和 l: L_Σ，通过对 l 进行归纳，定义 in_lang $t\, l$: $\mathbb{B}$：

$$\begin{array}{l|lll} \text{in\_lang} : \text{lang}_\Sigma \to \mathcal{L}_\Sigma \to \mathbb{B} & & & \\ \hline (\textbf{lnode}\ b\ \_)\ \ \textbf{nil} & & \triangleq & b \\ (\textbf{lnode}\ \_\ k)\ \ (\textbf{cons}\ a\ l) & & \triangleq & \text{in\_lang}\ (k\, a)\ l \end{array}$$

lang 的一个显著性质是lang$_\Sigma$ ，语言等价的语义概念与结构相等完全一致（参见 [27] 的外延前缀树）。这大大简化了余归纳字典树之间方程的证明，因为只需对列表进行直接归纳即可。

引理 48 (语义等价与结构相等一致)。 设 a： 语言$_\Sigma$和 b： 语言$_{\Sigma_\circ}$ $\Sigma_\circ$。那么，$(\forall l :$ $\mathcal{L}_\Sigma$，属于 _ 语言 $a\, l =$ 属于 _ 语言 $b\, l) \iff a = b_\circ$

我们用 ‘o a’ 表示一个布尔值，用于指示 a 是否接受空串；并用 ‘$\delta_a\, x$’ 表示语言 a 关于字符 x 的 Brzozowski 导数（即，对于所有 b 和 $\delta_{(\textbf{lnode}\ \_\text{k})}\ x$，以及对于所有 k 和 x，o (**lnode** b _) b ）。

定义 49（正则语言构造）。 定义包含空串的语言 ϵ：

$$\epsilon \triangleq \mathbf{lnode}\ \mathbf{true}\ (\lambda\_.\ \emptyset)$$

并通过原始余归纳定义语言的并集、交集和补集：

$$\begin{aligned} a + b &\triangleq \mathbf{lnode}\ (\mathrm{o}\ a \vee \mathrm{o}\ b)\ (\lambda x.\ \delta_a\ x + \delta_b\ x) \\ a\ \&\ b &\triangleq \mathbf{lnode}\ (\mathrm{o}\ a \wedge \mathrm{o}\ b)\ (\lambda x.\ \delta_a\ x\ \&\ \delta_b\ x) \\ \neg a &\triangleq \mathbf{lnode}\ (\neg(\mathrm{o}\ a))\ (\lambda x.\ \neg(\delta_a\ x)) \end{aligned}$$

余树语言上的结构序也与克莱尼代数中的标准语义序一致：$a \sqsubseteq b \Longleftrightarrow a + b = a$。 连接和克莱尼星号运算的定义比上述运算更为复杂，因为它们不是原始余递归的。我们将连接定义为对有限前缀树上的单调折叠的连续扩张：

定义 50 (连接)。 对于语言 $a : \mathrm{lang}_\Sigma$ 和 $b : \mathrm{lang}_\Sigma$，定义连接 $a \cdot b : \mathrm{lang}_\Sigma$，$\mathrm{tconcat}^{\mathrm{co}}\ a\ b$，其中：

$$\begin{aligned} \mathrm{tconcat} \triangleq\ & \mathrm{fold}_{\mathrm{tlang}^\cdot}\ (\lambda\_.\ \emptyset) \\ & (\lambda b\ k\ l.\ \mathbf{lnode}\ (b \wedge o\ l) \\ & \quad (\lambda x.\ k\ x\ l + (\mathrm{if}\ b\ \mathrm{then}\ \delta_l\ x\ \mathrm{else}\ \emptyset))) \end{aligned}$$

克莱尼星号是一个惰性余迭代（参见第 3.1节）：

定义 51 (克莱尼星号)。 对于语言 $a : \mathrm{lang}_\Sigma$，定义克莱尼闭包 $a^* : \mathrm{lang}_\Sigma$ 如下：

$$a^* \triangleq \mathrm{coiter}\ (\lambda b.\ \mathbf{lnode}\ \mathbf{true}\ (\delta_a\ x \cdot b))\ \omega_{\mathbb{N}}$$

本节定义的所有运算都可以提取到 Haskell 中，用于判定有限字母表上的字符串是否属于正则语言。最后，我们证明它们满足克莱尼代数公理：

定理 52 (克莱尼代数)。 类型 lang_Σ，其零元素为 $\emptyset$，单位元素为 ϵ，以及+、·和 * 运算构成一个克莱尼代数。

6 余归纳二叉树

本节将有限分支因子的无穷树的余归纳类型（余树）定义为代数 CPO。随后，我们使用余树在随机比特模型中为离散分布编码采样器[28, 29]，并展示 AlgCo 如何支持对它们进行期望值 [14, 30, 31] 推理（第6.1节），最终在定理 69中证明，样本序列与用于生成它们的余树的期望值语义等分布 [32]。本节的结果用于验证由概率程序编译而来的采样器，这些程序运行于Zar系统 [5]中。

余归纳树作为代数 *CPO*。为使表述清晰，我们将余树专门化为指标类型为 $\mathbb{B}$ 的余归纳二叉树（一般而言，指标类型必须是有限的，以确保基元素的紧致性）。

定义 53 (余树)。 通过以下形成规则余归纳地定义元素类型为 $\mathcal{T}_A^*$的余树类型 A：

$$\frac{}{\bot_{\mathcal{T}_A^*} : \mathrm{T}_A^*}\ \text{余树-底} \qquad \frac{a : A}{\mathbf{余叶}\ a : \mathcal{T}_A^*}\ \text{余树-叶} \qquad \frac{\forall b : \mathbb{B},\ k\ b : \mathrm{T}_A^*}{\mathbf{余节点}\ k : \mathcal{T}_A^*}\ \text{余树-节点}$$

以及简单明了的结构前缀关系：

定义 54 （余树序）。 定义 $\sqsubseteq_{\mathcal{T}_A^*} : \mathcal{T}_A^* \to \mathcal{T}_A^* \to \mathbb{P}$ 通过以下推理规则余归纳地定义：

$$\frac{l : \mathrm{T}_A^*}{\bot_{\mathcal{T}_A^*} \sqsubseteq_{\mathcal{T}_A^*} l}\ \sqsubseteq_{\mathcal{T}_A^*}\text{-BOT} \qquad \frac{a : A}{\mathbf{余叶}\ a \sqsubseteq_{\mathcal{T}_A^*} \mathbf{余叶}\ a}\ \sqsubseteq_{\mathcal{T}_A^*}\text{-叶} \qquad \frac{\forall b : \mathbb{B},\quad f\ b \sqsubseteq_{\mathcal{T}_A^*} g\ b}{\mathbf{余节点}\ f \sqsubseteq_{\mathcal{T}_A^*} \mathbf{余节点}\ g}\ \sqsubseteq_{\mathcal{T}_A^*}\text{-节点}$$

余树的一个紧致基由相应的有限二叉树有序类型给出（其顺序与余树的顺序相对应（定义 54），为简洁起见省略）：

定义 55 (有限树)。 通过以下形成规则归纳地定义元素类型为$\mathcal{T}_A$ 的有限二叉树类型 A：

$$\frac{}{\bot_{\mathcal{T}_A} : \mathrm{T}_A}\ \text{树-底} \qquad \frac{a : A}{\mathbf{叶子}\ a : \mathcal{T}_A}\ \text{树-叶} \qquad \frac{\forall b : \mathbb{B},\ k\ b : \mathrm{T}_A}{\mathbf{节点}\ k : \mathcal{T}_A}\ \text{树-节点}$$

注释 13（$\mathcal{T}_A$ 是紧致的）。

注释 14（$\mathcal{T}_A^*$是一个有基点代数连续偏序集，其基为 $\mathcal{T}_A$）。

余树上的余折叠 及其计算规则与流上的情况类似（第 4.1 节），相对于折叠算子推导如下：

定义 56 （折叠$_{\mathcal{T}_A}$）。 对于类型 A、有序类型 B，$z : B$，$f : A \to B$，$g : (\mathbb{B} \to B) \to B$，以及 $t : \mathcal{T}_A$，定义 折叠$_{\mathcal{T}_A}$ $z\ f\ g\ t : B$，通过对 t归纳定义：

$$\begin{array}{|l} \hline \mathrm{fold}_{\mathcal{T}_A}\ z\ f\ g : \mathcal{T}_A \to B \\ \hline \bot_{\mathcal{T}_A} \triangleq z \\ \mathbf{leaf}\ a \triangleq f\ a \\ \mathbf{node}\ k \triangleq h\ (\mathrm{fold}_{\mathcal{T}_A}\ z\ f\ g \circ k) \\ \hline \end{array}$$

6.1 离散分布采样器

一个余树 $t : \mathcal{T}_A^*$可以被解释为随机比特模型中样本空间 A 上的一个采样过程，该过程接收均匀分布的随机比特流，并据此生成样本。事实上，Zar [5]系统采用的正是这种方法：将带有无界循环和条件化的概率程序编译为采样器的余树表示，以表示其后验分布。

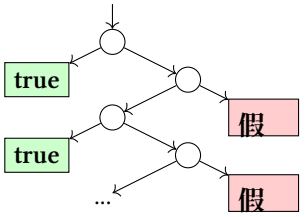


图3. 样本空间 $\mathbb{B}$ 上伯努利分布的余树编码（参数为$\frac{2}{3}$）。

余树 $t : \mathrm{T}_A^*$编码的采样器可以这样理解：从根节点开始，从环境中读取随机比特，以引导对 t的遍历；在每个**余节点**处读取一位，以确定进入左侧还是右侧子树。当到达**余叶** x时（如果最终能够到达），这一过程便会终止，并产生样本$x : A$。

例如，考虑图$t_{\frac{2}{3}}$3 中的余树采样器，它编码了一个伯努利分布，其中$p = \frac{2}{3}$（**true**的概率）。$t_{\frac{2}{3}}$所编码的过程以概率 1 终止（即，它“几乎必然”终止）。然而，尽管我们可以确信它最终会终止，却无法为其所需步数设定任何有限上限，因为理论上确实存在一条由左右交替选择构成的无限执行路径。

为了对余树采样器进行形式化推理，我们定义一种最弱前期期望（wp）语义的变体（最初由 Kozen 提出 [30]），这是根据其在概率守卫命令语言（pGCL）中的应用改编而来 [33]。

一个期望是一个函数 $f : A \to \mathbb{R}^\infty_{\geq 0}$，它将样本空间 A 中的元素映射到非负扩展实数。wp 的用途是计算余树上期望的期望值：

定义 57 ($\mathrm{wp}_{\mathcal{T}^*}$)。对于类型 A 和期望 $f : A \to \mathbb{R}^\infty_{\geq 0}$，定义 $\mathrm{wp}_{\mathcal{T}^*_A} f : \mathcal{T}^*_A \to \mathbb{R}^\infty_{\geq 0}$，$(\mathrm{wp}_{\mathcal{T}_A} f)^{\mathrm{co}}$，其中：

$$\mathrm{wp}_{\mathcal{T}_A} f \triangleq \mathrm{fold}_{\mathcal{T}_A}\ 0\ f\ (\lambda k.\ \frac{k\ \mathbf{true} + k\ \mathbf{false}}{2})$$

其 计算规则：

$$\begin{aligned}
&\mathrm{wp}_{\mathcal{T}^*_A} f \perp_{\mathcal{T}^*_A} = 0\\
&\mathrm{wp}_{\mathcal{T}^*_A} f\ (\mathbf{coleaf}\ a) = f\ a\\
&\mathrm{wp}_{\mathcal{T}^*_A} f\ (\mathbf{conode}\ k) = \frac{\mathrm{wp}_{\mathcal{T}^*_A} f\ (k\ \mathbf{true}) + \mathrm{wp}_{\mathcal{T}^*_A} f\ (k\ \mathbf{false})}{2}.
\end{aligned}$$

$\mathrm{wp}^{\mathcal{T}^*_A}$可用于表示给定事件 $Q : A \to \mathbb{P}$ 在余树 $t : \mathrm{T}_A^*$所编码的分布下的概率，该概率通过 $\mathrm{wp}_{\mathcal{T}^*_A}\ [Q]\ t$给出的示性函数$[Q]$的期望值表示。例如，$t_{\frac{2}{3}}$产生值**true**的概率由 $\mathrm{wp}\ [\lambda x.\ x = \mathbf{true}]\ t_{\frac{2}{3}} = \frac{2}{3}$给出。从技术上说，$\mathrm{wp}_{\mathcal{T}^*_A}\ [Q]\ t$表示

终止并产生满足 Q的样本的概率，不考虑那些完全没有产生样本的执行。

下面的引理揭示了$\mathrm{wp}_{\mathcal{T}^*}$与单子绑定（在$\mathcal{T}^*$ 单子中）之间的一个基本联系；在推理由带有顺序命令的概率程序编译而来的余树采样器时，这一联系尤为重要：

引理 58 ($\mathrm{wp}_{\mathcal{T}^*}$绑定)。 设 A和 B是类型， $f\colon B \to \mathbb{R}^{\infty}_{\geq 0}$， $t\colon \mathcal{T}_A^*$，且 $k\colon A \to \mathcal{T}_B^*$。则，

$$\mathrm{wp}_{\mathcal{T}_B^*}\, f\, (t \gg\!= k) = \mathrm{wp}_{\mathcal{T}_A^*}\, (\mathrm{wp}_{\mathcal{T}_B^*} f \circ k)\, t.$$

我们还通过证明若干健康性条件（包括马尔可夫不等式 $\mathrm{wp}_{\mathcal{T}^*}$），进一步验证 [14]。所有证明均借助第 2 节中的 AlgCo 工具完成。

定理 59 (马尔可夫不等式)。 设 A是一个类型， $f\colon A \to \mathbb{R}^{\infty}_{\geq 0}$， $t\colon \mathcal{T}_A^*$，且 $a\colon \mathbb{R}_{\geq 0}$。则，

$$\mathrm{wp}_{\mathcal{T}_A^*}\, [f \geq a]\, t \leq \frac{\mathrm{wp}_{\mathcal{T}_A^*}\, f\, t}{a}$$

6.2 余归纳测度

余树采样器也可以视为一种偏函数的编码，该偏函数将无限比特流（坎托尔空间的元素 *Cantor space* 2^{ω}）映射到样本。给定 $t\colon \mathrm{T}_A^*$，令 $f_t\colon 2^{\omega} \rightharpoonup A$ 表示由 t 诱导的函数：它要么在给定比特流上发散，要么从 t 的根节点出发，利用流中的比特引导遍历（例如，遇到 1 时进入左子树，遇到 0 时进入右子树），直到到达包含 x 的叶节点，从而产生样本 x。

事件 Q 在采样器函数 f_t 下的原像 $f_t^{-1}(Q)$，是 2^{ω} 中被 f_t 发送到 Q 中样本的比特流子集。我们用类型为 $\mathrm{T}\ {}^*\mathrm{L}_{\mathbb{B}}$ 的余树表示 2^{ω} 的子集，以编码 2^{ω} 中基本集合的可数并。基本集合由有限比特串编码，其中比特串 b 表示所有共享前缀 b 的比特流所构成的集合 $\{s\colon 2^{\omega} \mid b \sqsubseteq s\}$。我们进一步要求，余树集合中出现的所有比特串两两不可比较，即彼此不相交。我们用 Σ^0_1 表示基本集合的可数并这一类，并展示如何通过定义 62 中的连续扩张，计算事件的 Σ^0_1 原像的余树表示。

在这一视角下，我们将推断事件 Q相对于采样器 t的概率这一任务，重新表述为计算测度的 $f_t^{-1}(Q) \subseteq 2^{\omega}$，其中比特串 bs的测度等于 $\frac{1}{2^{(length\ bs)}}$，而余树（见定义 63）的测度是其组成比特串的测度之和（前提是它们两两不相交）。我们证明，根据采样器的 $\mathrm{wp}^{\mathcal{T}_A^*}$语义，任意事件 Q 的概率等于其在 t下的原像的测度（引理 67），然后利用这一点证明，由 t生成的样本序列相对于其 $\mathrm{wp}^{\mathcal{T}_A^*}$语义是等分布的（定理 69）。

计算原像。 余树的原像由 $\mathrm{lang}^{\mathcal{T}_A^*}$ 和 $\mathrm{filter}^{\mathcal{T}_A^*}$ 两个连续扩张复合而成。我们先从 $\mathrm{lang}^{\mathcal{T}_A^*}$开始，它不加区分地（不依据任何谓词）计算由 sampler t 发送到 2^{ω} 的子集，发送到 任意

叶子（即整个样本空间 A 的原像 $f_t^{-1}(A)$，或 t 的语言）。

定义 60 ($\mathrm{lang}^{\mathcal{T}^*_A}$). 对于类型 A，定义 $\mathrm{lang}^{\mathcal{T}^*_A}\colon \mathrm{T}^*_A \to \mathrm{T}\ {}^*\mathcal{L}_{\mathbb{B}}$、$\mathrm{lang}^{\mathcal{T}_{A\mathrm{co}}}$，其中：

$$\begin{aligned}\mathrm{lang}_{\mathcal{T}_A} \triangleq \mathrm{fold}_{\mathcal{T}_A}\ \bot_{\mathcal{T}^*_{\mathcal{L}_{\mathbb{B}}}}\ (\lambda\_.\ \mathbf{coleaf}\ \mathbf{nil})\\ (\lambda g.\mathbf{conode}\ (\lambda b.\ \mathrm{map}_{\mathcal{T}^*_A}\ (\mathbf{cons}\ b)\ (g\ b)))\end{aligned}$$

其 计算规则为：

$$\begin{aligned}&\mathrm{lang}_{\mathcal{T}^*_A}\ f\ (\mathbf{coleaf}\ a) = \mathbf{coleaf}\ \mathbf{nil}\\ &\mathrm{lang}_{\mathcal{T}^*_A}\ f\ (\mathbf{conode}\ k) =\\ &\quad \mathbf{conode}\ (\lambda b.\ \mathrm{map}_{\mathcal{T}^*_A}\ (\mathbf{cons}\ b)\ (\mathrm{lang}_{\mathcal{T}^*_A}\ (k\ b))).\end{aligned}$$

接下来，我们定义一个用于筛选的连续扩张：

定义 61 ($\mathrm{filter}^{\mathcal{T}^*_A}$). 对于类型 A 和谓词 $P\colon A \to \mathbb{B}$，定义 $\mathrm{filter}_{\mathcal{T}^*_A}\ P : \mathcal{T}^*_A \to \mathcal{T}^*_A \triangleq (\mathrm{filter}_{\mathcal{T}_A}\ P)^{\mathrm{co}}$，其中：

$$\begin{aligned}&\mathrm{filter}_{\mathcal{T}_A}\ P \triangleq\\ &\quad \mathrm{fold}_{\mathcal{T}_A}\ \bot_{\mathcal{T}^*_A}\ (\lambda a.\ \mathrm{if}\ P\ a\ \ \mathrm{then}\ \mathbf{coleaf}\ a\ \mathrm{else}\ \bot_{\mathcal{T}^*_A})\ \mathbf{conode}\end{aligned}$$

其 计算规则为：

$$\begin{aligned}&\mathrm{filter}_{\mathcal{T}^*_A}\ P\ (\mathbf{coleaf}\ a) = \mathrm{if}\ P\ a\ \mathrm{then}\ \ \mathbf{coleaf}\ a\ \mathrm{else}\ \bot_{\mathcal{T}^*_A}\\ &\mathrm{filter}_{\mathcal{T}^*_A}\ P\ (\mathbf{conode}\ k) = \mathbf{conode}\ (\mathrm{filter}_{\mathcal{T}^*_A}\ P \circ k).\end{aligned}$$

请注意，与 $\mathrm{filter}^{\mathcal{L}^*_A}$ (定义 32) 不同，$\mathrm{filter}^{\mathcal{T}^*_A}$移除的元素会被直接替换为 $\bot_{\mathcal{T}^*_A}$，而不是通过重构树来彻底消除它们。这没有问题，因为我们从不对原像集合进行计算——它们只是最终等分布定理（定理 69）的正确性规范的一部分。

谓词 Q 相对于采样器 t 的原像，便是 t经过 Q过滤后得到的语言。

定义 62 (原像$^{\mathcal{T}^*_A}$)。 对于类型 A和谓词$Q\colon A \to \mathbb{B}$，定义原像$^{\mathcal{L}^*_A}\colon \mathrm{T}^*_A \to \mathrm{T}\ {}^*\mathcal{L}_{\mathbb{B}}$如下：

$$\mathrm{preimage}_{\mathcal{L}^*_A} \triangleq \mathrm{lang}_{\mathcal{T}^*_A} \circ \mathrm{filter}_{\mathcal{T}^*_A}\ Q$$

我们通过将求和$^{\mathcal{T}^*_A}$算子应用于余树上的函数求和，来定义所需的Σ^0_1集合上的测度。

定义 63 (和$^{\mathcal{T}^*_A}$)。 对于类型 A和 $f\colon A \to \mathbb{R}^\infty_{\geq 0}$，定义 和$_{\mathcal{T}^*_A}\ f\colon \mathcal{T}^*_A \to \mathbb{R}^\infty_{\geq 0}$，和$_{\mathcal{T}_A}\ f^{\mathrm{co}}$，其中：

$$\mathrm{sum}_{\mathcal{T}_A} \triangleq \mathrm{fold}_{\mathcal{T}_A}\ 0\ f\ (\lambda g.\ g\ \mathbf{true} + g\ \mathbf{false})$$

带有计算规则：

$$\begin{aligned}&\mathrm{sum}_{\mathcal{T}_A^*}\ f\ (\mathbf{coleaf}\ a) = f\ x\\&\mathrm{sum}_{\mathcal{T}_A^*}\ f\ (\mathbf{conode}\ k) =\\&\qquad \mathrm{sum}_{\mathcal{T}_A^*}\ f\ (k\ \mathbf{true}) + \mathrm{sum}_{\mathcal{T}_A^*}\ f\ (k\ \mathbf{false})\end{aligned}$$

定义 64 ($\mu^{\mathcal{T}^*_{\mathcal{L}_\mathbb{B}}}$)。 定义 $\mu^{\mathcal{T}^*_{\mathcal{L}_\mathbb{B}}}\colon \mathrm{T}\ {}^*\mathrm{L}_\mathbb{B} \to \mathbb{R}^\infty_{\geq 0}$, 和$^{\mathcal{T}^*_{\mathcal{L}_{\mathrm{bool}}}}(\lambda l.\ \frac{1}{2^{(\mathrm{length}_{\mathcal{L}_\mathbb{B}}\ l)}})$

使用 *AlgCo* 证明均匀分布。在本节中，我们使用 AlgCo 证明，余树采样器 t 生成的任意样本序列，相对于 t的 $\mathrm{wp}^{\mathcal{T}_A^*}$ 语义都是等分布的。从较高层面来看，我们的策略是将随机比特源服从均匀分布这一假设通过采样器“向前推”。首先，我们需要以下两个引理：

引理 65 ($\mathrm{wp}^{\mathcal{T}_A^*}$ $\mathrm{filter}^{\mathcal{T}_A^*}$)。 设 A 是一个类型，且 $Q\colon A \to \mathbb{B}$。那么，$\mathrm{wp}_{\mathcal{T}_A^*}\ [Q] = \mathrm{wp}_{\mathcal{T}_A^*}\ \mathbf{1} \circ \mathrm{filter}_{\mathcal{T}_A^*}\ Q$。

引理 66 ($\mathrm{wp}_{\mathcal{T}_A^*}\ \mathbf{1}$ 等于 $\mathrm{mu}^{\mathcal{T}_A^*} \circ \mathrm{lang}^{\mathcal{T}_A^*}$)。 设 A 是一个类型，且 $Q\colon A \to \mathbb{B}$。那么，$\mathrm{wp}_{\mathcal{T}_A^*}\ \mathbf{1} = \mathrm{mu}^{\mathcal{T}_A^*} \circ \mathrm{filter}_{\mathcal{T}_A^*}\ Q$。

引理65 和66 的证明都通过将右端融合为单个余态射，并应用引理13 ，将目标化简为对有限树的直接归纳。由此立即可得，按照采样器的 wp 语义，任意事件 Q 的概率$\mathrm{wp}^{\mathcal{T}_A^*}$与采样器 t 的原像的测度相同 Q在 f_t下：

引理 67 ($\mathrm{wp}^{\mathcal{T}_A^*}$是原像的测度)。 设 A 是一个类型， $Q\colon\ A \to \mathbb{B}$，且 $t\colon\ \mathcal{T}_A^*$。那么，$\mathrm{wp}_{\mathcal{T}_A^*}\ [Q]\ t = \mathrm{mu}^{\mathcal{T}^*_{\mathcal{L}_\mathbb{B}}}\ (\text{原像}_{\mathcal{T}_A^*}\ Q\ t)$。

为说明随机比特源的均匀分布，我们采用经典“模 1 均匀分布”概念的一种变体 [32, 34] ，并将其推广到 Σ^0_1的子集 $2^{\mathbb{N}}$。

定义 68 (Σ^0_1-均匀分布)。 若比特流序列$\{x_i\}$ 是Σ^0_1-均匀分布的(Σ^0_1-均匀分布)，则对每个 $U\colon\ \Sigma^0_1$，$\lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n} [\exists^{\mathrm{co}}_{=x_i}\ U] = \mathrm{mu}_{\mathcal{T}^*_{\mathcal{L}_\mathbb{B}}}\ U$。

换句话说，当样本数量趋于无穷时，如果每个 Σ^0_1集合都获得了“应有份额”的样本，那么比特流序列就是均匀分布的。

定理 *69*（样本的等分布）。设 A是一个类型， $t\colon T_A^*, Q\colon A \to \mathbb{B}$， $\{x_n\}$ 是一个Σ^0_1-*u.d.* 比特流序列，而 $\{f_t(x_n)\}$ 是通过将f_t 映射到 $\{x_n\}$ 上得到的样本序列。那么，$\{f_t(x_n)\}$ 关于 t 是 $wp^{\mathcal{T}_A^*}$-等分布的：

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n} [Q\ (f_t(x_i))] = \mathrm{wp}_{\mathcal{T}_A^*}\ t\ [Q]\ \sigma$$

证明。根据引理67改写右端，目标变为：

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n} [Q\ (f_t(x_i))] = \mathrm{mu}_{\mathcal{T}^*_{\mathcal{L}_\mathbb{B}}}\ (\mathrm{preimage}_{\mathcal{T}_A^*}\ Q\ t).$$

然后在 Σ_1^0-*u.d.* 的假设中令 $U = \text{原像}_{\mathcal{T}_A^*}\ Q\ t$，得到：

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n} [\exists^{\text{co}}_{=x_i}\ \text{preimage}_{\mathcal{T}_A^*}\ Q\ t] = \text{mu}_{\mathcal{T}^*_{\mathcal{L}_{\mathbb{B}}}}\ \text{preimage}_{\mathcal{T}_A^*}\ Q\ t$$

由此目标立即可得，因为 $Q(f_t(x_i)) \Longleftrightarrow \exists^{\text{co}}_{=x_i} \text{原像}_{\mathcal{T}_A^*}\ Q\ t.$ □

7 相关工作

参数化余归纳（*paco*）。paco 库[6] 为 Coq 提供了原始余归纳最广为人知的一般化形式。Paco 用语义条件替代了余归纳证明的语法受限条件。与 Coq 内置的cofix机制相比，这提高了复合性，并缓解了一些常见陷阱，例如误用余归纳假设，而这类错误直到证明完成时才会被检查。然而，使用 paco 定义的余归纳性质仍被定义为单调泛函的最大不动点，因此，在尝试定义关于余归纳结构的函数映射时，仍会遭遇引言中所述的问题。借助 AlgCo，我们采取了完全不同的方法：划定一类余归纳类型的子集（即构成代数 CPO 的类型），对于这类类型，可以用归纳推理替代余归纳推理。Pous 的cawu 框架 [7] 将参数化余归纳理论一般化，并简化了对双模拟“up-to” [35, 36] 增强技术的处理，但从根本上说，它与本文的关系和paco 相同。

余模式。 Abel 等人 [37] 提出了一种类型论基础，用于通过观测（即 余模式 匹配）对流等无限数据进行编程。作为归纳模式匹配的范畴对偶，归纳模式匹配为归纳数据提供了一般性的 消去方案，余模式匹配则为余归纳数据提供了一般性的引入方案。AlgCo 通过为代数余归纳提供 连续消去方案 （引理12），填补了两者之间的空缺。例如，函数sum : 流 $\mathbb{R}^{\infty}_{\geq 0} \to \mathbb{R}^{\infty}_{\geq 0}$ 无法像引言中那样通过余模式定义，因为其陪域不是余归纳类型；但由于其域是代数的，因此可以将其定义为连续扩张。

正则余归纳对象。Jeannin 等人 [38] 扩展了 OCaml，使其支持“正则”（可有限表示的）余归纳类型，并为它们配备了一般性的消去方案，该方案以用户指定的求解器为参数，使程序员能够定义从正则余归纳结构到 CPO 的映射（例如，通过不动点求解器将正则无限项映射为其自由变量集合）。然而，我们感兴趣的许多余归纳结构，例如第 6.1 节所述由概率程序生成的余树采样器，并不总是可以有限表示。代数 CPO 的概念推广到更广泛的一类余归纳结构，同时仍提供了一种通用的（尽管较难计算的）消去原则。

友好函数与大小类型。 在 Isabelle/HOL 证明助手中，通过所谓的

“友好函数” [39],来放宽对原始余递归定义的语法守卫条件，因为它们能保持其参数的生产性。因此，余递归调用可以出现在友好函数的应用之下。大小类型 [40] 已在 Agda 证明助手中实现 [2]，以发挥类似作用，扩大允许使用的余递归定义范围；它们也已用于实现余归纳前缀树上的连接和闭包算子 [26]，如第 5.1 节所示。

连续扩张。 Lochbihler 和 Hölzl [41] 运用域理论和拓扑学中的概念，将流上的 filter 运算定义为有限列表上一个函数的连续扩张（即“消费者视角”），并将相关证明简化为对列表进行归纳。AlgCo 并不明确要求使用任何拓扑学概念，并将范围从流转换器推广到余归纳结构上的各种连续映射，包括实值语义（第 6.1 节）和命题函数（第 2.5 节）。

Rusu 和 Nowak 采用了类似的方法，根据 CPO 元素的有限近似定义余递归函数，从而分别在流和玫瑰树上定义 filter 和镜像操作。我们指出了代数 CPO 的基础抽象，并围绕 AlgCo 框架中此类函数的复合、验证和执行，给出了系统而连贯的阐述。

8 结论

本文介绍了 AlgCo（代数余归纳），这是一个形式化的类型论框架，用于推理由构成代数 CPO 的余归纳结构类上的连续函数。我们介绍了该框架的基本概念，并在AlgCo库中提供了 Coq 实现。我们通过实现并验证若干示例来展示其用途，包括埃拉托斯特尼筛法的流变体、基于 Brzozowski 导数的正则表达式框架，以及在 randombit 模型中对离散分布采样器进行期望值推理。

参考文献

[1] INRIA。（2022）Coq 参考手册。 [在线]。网址：https://coq.inria.fr/refman/# [2] Agda 团队。（2022）Agda 用户手册。[在线]。网址：https://agda.readthedocs.io/en/v2.6.2.2.20221128/ [3] D. Kozen 和 A. Silva，“实用余归纳”，《数学结构与计算机科学》，第 27， 7， 1132 1152, 卷，第 期，第 页。在 2017 [ ] 10 1017 0960129515000493 线。网址：https://doi.org/ 。/S[4] A. Chlipala，《依赖类型的认证编程：Coq 证明助手的实用入门》。麻省理工学院出版社，2022 年。[5] A. Bagnall、G. Stewart 和 A. Banerjee，“通过带循环和条件化的概率程序构造形式化验证的采样器”，2022 [ ] 2211 06747 。在线。网址：[6] https://arxiv.org/abs/ 。C. Hur、G. Neis、D. Dreyer 和 V. Vafeiadis，“余归纳证明中的参数化力量”，载于《第 40届 ACM SIGPLAN-SIGACT 编程语言原理年度研讨会》，POPL ’13，意大利罗马，2013 年 1 月 23—25 日，R. Giacobazzi 和 R. Cousot 编。美国计算机协会，2013 年，第 193–206 页。 [ ] 10 1145 2429069 2429093 在线。网址：https://doi.org/ 。/。D. Pous，“一路向上 [7] 的余归纳”，载于《第 届 ACM/IEEE 计算机科学逻辑研讨会会议论文集》，LICS ’16，美国纽约州纽约，2016 年 7 月 5—8 日，M. Grohe、E. Koskinen 和 N. Shankar 编。美国计算机协会，2016， 307 316 [ ] 10 1145 2933575 2934564 第 – 页。在线。网址：[8] https://doi.org/ 。/。A. Jung，“代数连续偏序集的笛卡尔闭范畴”，《理论计算机科学》，第 卷，第 2 期，第 233–250 页，1990 年。 [在线]。网址：https://doi.org/10.1016/0304-3975(90)90124-Z [9] C. A. Gunter，《编程语言语义：结构与技术》。麻省理工学院出版社，1992 年。[10] L. Xia、Y. Zakowski、P. He、C. Hur、G. Malecha、B. C. Pierce 和 S. Zdancewic，“交互树：在 Coq 中表示递归和非纯程序”，《ACM 程序设计语言会议论文集》，第 4 卷，POPL 期，第 51 1 51 32, 2020 [ ] 10 1145 3371119 页：–：。在线。网址：[11] https://doi.org/ 。/ V. Rusu 和 D. Nowak，“在 Coq 中使用近似定义余递归函数”，载于《第36届欧洲面向对象编程会议，ECOOP 2022》，2022 年 6 月 6—10 日，德国柏林，丛书：LIPIcs，K. Ali 和 J. Vitek 编，第 222 卷。达格施图尔城堡—莱布尼茨信息学中心，2022， 12 1 12 24 [ ] 10 4230 2022 12 第： –： 页。在线。网址：[12] https://doi.org/ 。/LIPIcs.ECOOP. 。T. Hagino，“ML 中的余数据类型”，《符号计算杂志》，第 卷，第 期，第 – 页。在线。网址：https://doi.org/ 10 1016 0747 7171 89 80065 3 。/S - ( ) - J. Adámek，“代数余完备性与有限函子”，《逻辑方法与计算机科学》，第 17 卷，第 2 期，2021 年，第 期。 [13] [ ] 7524 在线。网址：https://lmcs.episciences.org/ B. L. Kaminski，“概率程序的高级最弱前置条件演算”，博士论文，亚琛工业大学，德国，2019 年。[在线]。网址：[14] http://publications.rwth-aachen.de/record/ 754405 [15] 2017 A. Charguéraud。（）Coqandaxioms。[在线]。网址：https://github.com/coq/coq/wiki/CoqAndAxioms [16] A. Charguéraud，“最优不动点组合子”，载于《交互式定理证明：第一届国际会议，ITP 2010》，英国爱丁堡，2010 年 7 月 11—14 日。会议论文集，丛书：计算机科学讲义，M. Kaufmann 和 L. C. Paulson 编，第 6172 卷。施普林格，2010 年，第 195–210 页。 [ ] 10 1007 978 3 642 14052 5_15 在线。网址：https://doi.org/ 。/- - - - S. Abramsky 和 A. Jung，“域理论”，载于《计算机科学逻辑手册》，S. Abramsky 、 D. M. Gabbay 和 T. S. E. Maibaum 编。克拉伦登出版社，1994 年，第 3 卷，第 1–168 页。[18] （2023）库 coq.logic.classicalepsilon。[在线]。网址：https://coq.inria.fr/library/Coq.Logic.ClassicalEpsilon.html [19] X. Leroy。（2015）尽情使用 Coq 的求值机制。 [在线]。网址：http://gallium.inria.fr/blog/coq-eval/ [20] M. Sozeau，“类型论中广义重写的新视角”，《形式化推理杂志》，第 2 卷，第 1 期，第 41 62, 2009 [ ] 10 6092 1972 5787 1574 – 页。在线。网址：https://doi.org/ 。/ [17] [21] issn. - / E. Meijer、M. M. Fokkinga 和 R. Paterson，“使用香蕉、镜头、信封和带刺铁丝网进行函数式编程”，载于《函数式编程语言与计算机体系结构：第5届 ACM 会议》，美国马萨诸塞州剑桥，1991 年 8 月 26—30 日。会议论文集，丛书：计算机科学讲义，J. Hughes 编，第 523 卷。施普林格，1991 年，第 124–144 页。 [在线]。可访问：

https://doi.org/10.1007/3540543961_7

[22] Y. Bertot, 《余归纳流生的 filter：在埃拉托斯特尼筛法中的应用》，载于《类型化 Lambda 演算与应用国际会议》。施普林格，2005年，第102–115页。[23] M. E. O'Neill, 《真正的埃拉托斯特尼筛法》，《函数式编程杂志》，第19卷，第1期，第95–106页，2009年。 [ ] 10 1017 0956796808007004 在线。网址：https://doi.org/ . /S Y. Bertot 和 V. Komendantsky, 《Coq 中的不动点语义与部分递归》，载于《第十届国际 ACM SIGPLAN 声明式编程原理与实践会议论文集》，2008年7月15日至17日，西班牙瓦伦西亚，S. Antoy 和 E. Albert 编。美国计算机协会，2008年，第89–96页。 [ ] 10 1145 1389449 1389461 在线。网址：https://doi.org/ . / . T. L. Heath 等, 《欧几里得〈几何原本〉十三卷》。Courier Corporation，1956年。[26] D. Traytel, 《正则语言符号决策过程的形式化》，博士论文，慕尼黑工业大学，2015年。 [在线]。网址：https://nbn-resolving.org/urn:nbn:de:bvb: 91 20151016 1273011 1 9 -diss- - - - A. W. Appel 和 X. Leroy, 《高效的外延二叉字典树》，CoRR，第abs/2110.05063卷，2021年。 [ ] 2110 05063 在线。网址：https://arxiv.org/abs/ . J. 冯·诺伊曼, 《与随机数字有关的各种技术》，Appl. Math Ser，第12卷，第36-38期，第5页，1951年。[29] F. A. Saad、C. E. Freer、M. C. Rinard 和 V. K. Mansinghka, 《从离散概率分布中进行最优近似采样》，Proc. ACM Program. Lang.，第4卷，POPL期，第36:1–36:31页，2020年。 [ ] 10 1145 3371104 在线。网址：https://doi.org/ . / D. Kozen, 《概率命题动态逻辑》，J. Comput. Syst. Sci.，第30卷，第2期，第162–178页，1985年。 [在线]。网址：https://doi.org/ 10 1016 0022 0000 85 90012 1 . / - ( ) - C. Morgan、A. McIver 和 K. Seidel, 《概率谓词变换器》，ACM Trans. Program. Lang. Syst.，第 18, 3, 325 353, 1996 [ ] 10 1145 229542 229547 卷，第期，第–页。在线。网址：https://doi.org/ . / . V. Becher 和 S. Grigorieff, 《随机性与模一均匀分布》，Inf. Comput.，第 285, 104857, 2022 [ ] 10 1016 2021 104857 卷，第Part期，第页。在线。网址：https://doi.org/ . /j.ic. . A. McIver 和 C. Morgan, 《概率系统的抽象、精化与证明》，计算机科学专著丛书。施普林格, 2005 [ ] 10 1007 138392 。在线。网址：https://doi.org/ . /b L. Kuipers 和 H. Niederreiter, 《序列的均匀分布》。Courier Corporation，2012年。[35] R. Milner, 《通信与并发》。Prentice Hall，恩格尔伍德克利夫斯，1989年，第84卷。[36] D. Sangiorgi, 《关于双模拟证明方法》，《计算机科学中的数学结构》，第8卷，第5期，第447–479页，1998年。[37] A. Abel、B. Pientka、D. Thibodeau 和 A. Setzer, 《余模式：通过观测编程无限结构》，载于《第40届 ACM SIGPLAN-SIGACT 编程语言原理年度研讨会》，POPL '13，意大利罗马，2013年1月23日至25日，R. Giacobazzi 和 R. Cousot 编。美国计算机协会, 2013, 27 38 [ ] 10 1145 2429069 2429075 第–页。在线。网址：https://doi.org/ . / . J. Jeannin、D. Kozen 和 A. Silva, 《Cocaml：使用正则余归纳类型进行函数式编程》，Fundam. Informaticae，第150卷，第3-4期，第347–377页，2017年。 [在线]。网址：https://doi.org/ 10 3233 2017 1473 . /FI- - J. C. Blanchette、A. Bouzy、A. Lochbihler、A. Popescu 和 D. Traytel, 《互利共赢——在基础证明助手中实现余递归》，载于《编程语言与系统——第26届欧洲编程研讨会（ESOP 2017）》，作为《欧洲软件理论与实践联合会议》（ETAPS 2017）的一部分举行，乌普萨拉，瑞典，2017年4月22日至29日，会议论文集，计算机科学讲义丛书，H. Yang 编，第10201卷。施普林格，2017年，第111–140页。 [在线]。网址：https://doi.org/ 10 1007 978 3 662 54434 1_5 . / - - - - J. Hughes、L. Pareto 和 A. Sabry, 《使用大小类型证明反应式系统的正确性》，载于《POPL'96 会议记录：第23届 ACM SIGPLAN-SIGACT 编程语言原理研讨会》，研讨会论文，佛罗里达州圣彼得堡海滩，美国，1996年1月21日至24日，H. Boehm 和 G. L. Steele Jr. 编。ACM出版社，1996年，第410–423页。 [在线]。网址：https://doi.org/ 10 1145 237721 240882 . / . A. Lochbihler 和 J. Hölzl, 《通过域和拓扑在惰性列表上定义递归函数》，载于《交互式定理证明——第五届国际会议（ITP 2014）》，作为维也纳

逻辑之夏，*VSL 2014*，奥地利维也纳，*2014*年*7*月*14*日至*17*日。会议论文集，计算机科学讲义丛书，G. Klein 和 R. Gamboa 编，第8558卷。施普林格，2014年，第341–357页。 [在线]。网址：https://doi.org/10.1007/978-3-319-08970-6_22

[42] S（借助原法模型扩展类型设计模型上展类型法国南特高等矿业电信学院布列塔尼卢瓦尔河地区分校，J. Boulier，《用语法模型扩展类型论》 2018 [ ] 02007839。在线。可访问：

[43] 2023 [ ] 69905520 https://tel.archives-ouvertes.fr/tel-J. Gross。（）余归纳外延性。在线。可访问：https://stackoverflow.com/a/

A 代数 CPO 证明原则

连续扩张的主要定义原则：

定理70（连续扩张的存在性）。设 A 是一个代数 *CPO*， C 是任意 *CPO*，且 f： $\mathrm{B}(A) \to C$ 是一个单调函数。定义

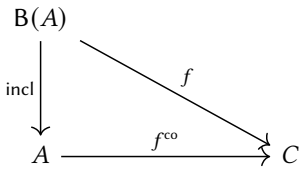
$$f^{\mathrm{co}}\, a \triangleq \sup\,(f \circ \mathrm{idl}\, a).$$

那么，

$$f^{\mathrm{co}} \circ \mathrm{incl} = f. \tag{2}$$

也就是说，下图交换：

$$\begin{tikzcd} \mathrm{B}(A) \arrow[d, "\mathrm{incl}"'] \arrow[dr, "f"] & \\ A \arrow[r, "f^{\mathrm{co}}"] & C \end{tikzcd}$$



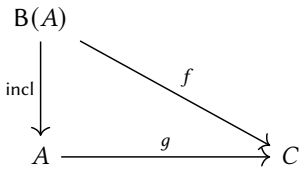
连续扩张的唯一性给出了基本的证明 原理：

定理 71 (连续延拓的唯一性). 设 A 是一个代数 *CPO*， C是任意 *CPO*， $f\colon \mathrm{B}(A) \to C$ 是定义在 A的基上的单调函数，并且 g： $A \to C$ 是定义在 A 上的连续函数，满足：

$$g \circ \mathrm{incl} = f,$$

也就是说，下图交换：

$$\begin{tikzcd} \mathrm{B}(A) \arrow[d, "\mathrm{incl}"'] \arrow[dr, "f"] & \\ A \arrow[r, "g"] & C \end{tikzcd}$$



那么， $g = f^{\mathrm{co}}$。

以下这一等价原理的广义逐点形式比引理 13 更强，因为它保留了前提中的基元素是目标中 CPO 元素的近似这一信息；当所证明的等式取决于某个前提条件时，这一点可能很重要。

引理 72 (连续扩张的广义等价性)。 设 A 和 B 为代数 CPO， C 为任意 CPO， $f : \mathrm{B}(A) \to C$， $g : \mathrm{B}(B) \to C$ 为单调函数， $a : A$且 $b : B$。于是，

$$\forall\, i : \mathbb{N},\ f\ (\mathrm{idl}\ a\ i) = g\ (\mathrm{idl}\ b\ i) \Rightarrow f^{\mathrm{co}}\ a = g^{\mathrm{co}}\ b.$$

推论 73 (连续函数的表示)。 设 A 是一个代数 CPO，C 是任意 CPO，且 $g : A \to C$ 是一个连续函数。那么，

$$g = (g \circ \mathrm{incl})^{\mathrm{co}}.$$

推论 73是定理 71的一个相当直接的推论，它表明每个连续函数 $g : A \to C$ 都可以写成如下连续扩张 $(g \circ \mathrm{incl})^{\mathrm{co}}$。也就是说，每个连续函数 $g : A \to C$ 都可以表示为某个单调基函数的连续扩张。因此，任何两个连续函数在基上的限制相等时，它们就相等：

推论 74 (连续函数的等价性)。 设A是一个基为 B 的代数 CPO， C是任意 CPO，且 f， $g : A \to C$是连续函数。那么，

$$f \circ \mathrm{incl} = g \circ \mathrm{incl} \Rightarrow f = g$$

B 余归纳外延性

Coq 中通常使用的命题（莱布尼茨）相等概念，太弱了，无法证明诸如 $\mathbb{N}^{\mathrm{co}}$ (定义 17) 这样的余归纳类型上的等式。例如，假设我们定义一个函数 coplus : $\mathbb{N}^{\mathrm{co}} \to \mathbb{N}^{\mathrm{co}} \to \mathbb{N}^{\mathrm{co}}$ 用于计算两个余自然数之和。尝试证明交换性等基本性质时，我们很快就会陷入困境：$\forall n\ m,$ coplus $n\ m =$ coplus $m\ n$。通常，证明会对 n 或 m 进行归纳，但这里这两个项都不是归纳类型。我们也无法使用余归纳，因为目标并非余归纳的。

另一种方法是定义一种余归纳双模拟关系，当且仅当n 和 m 在结构上完全相同时，该关系成立：

定义 75 ($\mathbb{N}^{\mathrm{co}}$ 等价关系)。 按以下 $=_{\mathbb{N}^{\mathrm{co}}}$： $\mathbb{N}^{\mathrm{co}} \to \mathbb{N}^{\mathrm{co}} \to \mathbb{P}$ 推理规则余归纳地定义：

$$\frac{}{\mathbf{cozero} =_{\mathbb{N}^{\mathrm{co}}} \mathbf{cozero}}\ =_{\mathbb{N}^{\mathrm{co}}}\text{-ZERO} \qquad \frac{n =_{\mathbb{N}^{\mathrm{co}}} m}{\mathbf{cosucc}\ n =_{\mathbb{N}^{\mathrm{co}}} \mathbf{cosucc}\ m}\ =_{\mathbb{N}^{\mathrm{co}}}\text{-SUCC}$$

现在，我们可以通过余归纳证明交换性（相差 $=_{\mathbb{N}^{\mathrm{co}}}$）： $\forall n\, m,$ coplus $n\, m =_{\mathbb{N}^{\mathrm{co}}}$ coplus $m\ n$。不过，这种方法的问题在于，如果我们想用这类方程进行改写，就必须明确证明，关于 $\mathbb{N}^{\mathrm{co}}$的所有运算对于 $=_{\mathbb{N}^{\mathrm{co}}}$都是良定义的。但请注意， $=_{\mathbb{N}^{\mathrm{co}}}$ 经过精心设计，恰好与莱布尼茨相等完全一致。虽然无法在 Coq 中证明这一点，我们可以通过外延性公理（公理 1）断言这一事实：由余自然数上的互模拟证明可推出莱布尼茨相等（因此，无需任何 Proper 实例，就能轻松地据此改写）。

为了增强对公理 1可靠性的信心，请注意， $\mathbb{N}^{\mathrm{co}}$ 模$=_{\mathbb{N}^{\mathrm{co}}}$ 与类型$\mathbb{N}+1$同构，因为每个余自然数要么是有限自然数，要么是无限余自然数 $\omega_{\mathbb{N}}$
令 $\mathsf{sect} : \mathbb{N}^{\mathrm{co}} \to \mathbb{N}+1$ 和 $\mathsf{retr} : \mathbb{N}+1 \to \mathbb{N}^{\mathrm{co}}$ 见证这个同构。我们可以将余自然数外延性推导为一个定理，由该同构的一侧得出： $\forall n : \mathbb{N}^{\mathrm{co}},\ \mathsf{retr}(\mathsf{sect} n) = n$。也就是说，余自然数外延性可以由这样一个事实推导得出：将一个余自然数注入 $\mathbb{N}+1$ 后再将其投影回来，会还原出原来的余自然数。

对于流、余归纳字典树和余树，也可以进行类似的论证。参见 [42, 第 2.2 节。2] 以及 [43]，了解更多关于余归纳类型外延性公理之可靠性的讨论。

C 交互树与余树的关联

Zar系统[5] 不会直接提取余树以供执行。相反，它会将概率程序编译为交互树 [10]（itrees），以便在 OCaml 中进行采样执行，并将其与等价的余树构造关联起来；由此借助 AlgCo 获得的正确性保证（例如，定理69）得以传递。交互树包含Tau 节点（如本文引言所述），以保证生产性，但代价是可能出现无限的Tau序列（这些序列可能出现在余树表示中的任意位置$\bot_{\mathcal{T}_A^*}$）。因此，Zar 为了便于将余树与相应的交互树关联起来，在余树的定义中也包含了Tau 节点。

将这两种表示联系起来的主要挑战，是为余树提供一种迭代构造，以匹配 itree 库中的 ITree.iter 组合子：

定义 76（$\mathsf{iter}^{\mathcal{T}_A^*}$）。对于类型 I和 A, $f\colon I \to \mathrm{T}^*_{I+A}$，以及 $z\colon I$，定义 $\mathsf{iter}_{\mathcal{T}_A^*}\ f\ z\colon \mathrm{T}^*_A$, $\mathsf{sup}\ (\mathrm{F}^n(\lambda\_.\,\mathsf{cobot}))\ z$，其中 $\mathrm{F} : (I \to \mathrm{T}^*_A) \to I \to \mathrm{T}^*_A$ 由下式给出：

$$\begin{aligned}\mathrm{F}\ g\ i \triangleq f\ i \gg\!= \lambda lr.\ &\mathsf{match}\ lr\ \mathsf{with}\\ &\mid \mathsf{inl}\ j \Rightarrow g\ j\\ &\mid \mathsf{inr}\ x \Rightarrow \mathbf{coleaf}\ x\\ &\mathsf{end}.\end{aligned}$$

图 4 展示了一个用于 $\mathsf{iter}^{\mathcal{T}_A^*}$ 的 Haskell 提取原语，其依据是以下计算规则：

引理 77 ($\mathsf{iter}^{\mathcal{T}_A^*}$计算规则)。设 I和 A为类型， $f\colon I \to \mathcal{T}^*_{I+A}$，且 $i\colon I$。那么，

$$\begin{aligned}\mathsf{iter}_{\mathcal{T}_A^*}\ f\ i = f\ i \gg\!= \lambda lr.\ &\mathsf{match}\ lr\ \mathsf{with}\\ &\mid \mathsf{inl}\ j \Rightarrow \mathsf{iter}_{\mathcal{T}_A^*}\ f\ j\\ &\mid \mathsf{inr}\ x \Rightarrow \mathbf{coleaf}\ x\\ &\mathsf{end}.\end{aligned}$$

有关交互树与余树关系的更多细节，可参阅 Zar 系统的源代码。

```
余树 iter  =_\ f i ->余树_bind(f i) (\ lr ->匹配lr 取值Inl j ->  余树_iter f j Inr x ->  余叶 x)
```

图 4. Haskell 的 iter 提取原语$\mathcal{T}_{\mathrm{A}}^{*}$。