基于正则递归类型的自动摊还资源分析

Jessie Grosen
jgrosen@cs.cmu.edu 卡内基梅隆大学

David M. Kahn
davidkah@andrew.cmu.edu 卡内基梅隆大学

Jan Hoffmann
jhoffmann@cmu.edu 卡内基梅隆大学

arXiv:2304.13627v1 [cs.PL] 26 Apr 2023

摘要——自动资源界分析的目标是静态推断程序求值过程中资源消耗的符号界。自动资源分析长期面临的一项挑战，是推断以复杂自定义数据结构为参数的界。本文在基于类型的自动摊还资源分析（AARA）的基础上应对这一挑战。AARA 基于摊销分析的势能法，将界的推断转化为标准类型推断与额外的线性约束求解，即使在推导非线性界时也是如此。AARA 的一个关键组成部分是资源函数，它们为给定类型的值生成可能界的空间，同时具备必要的闭包性质。

现有关于 AARA 的研究为许多数据结构（例如列表的列表）定义了此类函数，但对于任意数据结构是否存在此类函数这一问题仍未解决。本文通过统一构造由正则递归类型定义的代数数据结构的资源多项式，肯定地回答了这一问题。这些函数概括了此前提出的所有多项式资源函数，也可以被视为给定递归类型的值上的多项式的一般概念。针对 FPC（一种具有递归类型的核心语言）的资源类型系统展示了如何将资源多项式整合进 AARA，同时保留以往技术的全部优势。本文还提出了可用于简洁陈述该类型系统规则并证明其相对于小步成本语义的健全性的新技术。首先，多元势能注释以自由半模的形式陈述，从而大幅抽象了注释表示及其性质证明的细节。其次，一个为资源类型赋予语义含义的逻辑关系，使得只需对类型推导进行一次归纳即可完成健全性证明。

I. 引言

针对静态推导资源（或代价）界的编程语言支持已得到广泛研究。现有技术涵盖手动和自动资源分析，并基于类型系统 [1], [2], [3], 推导并求解递推关系 [4], [5], [6], , 或采用其他静态分析 [7], [8], [9]。这些技术可以推导（最坏情况）上界 [10], [11] 、（最好情况）下界 [12], [13], , 以及两个程序 [14], 在时间或内存等资源上的代价差异的关系界。

大多数自动化技术都专注于整数或简单数据结构（如整数列表）的大小函数界。一个例外是自动摊还资源分析（AARA） [15], [16], [17],, 它可以自动推导复杂数据结构（如列表的列表）的界限，同时考虑内层列表的各自长度。作为

979-8-3503-3587-3/23/$31.00 ©2023 IEEE

例子，考虑函数 sort_lefts_list，该函数从输入列表中提取左注入，并对结果进行排序。假设我们关注的是求值过程中创建的 cons 单元数量。

令 排序_左项_列表(l : (整数 + 布尔类型) 列表) =快速排序 (过滤_映射查找_左项 l)

RaML [18], 是 AARA 的一种实现，能够精确推导 n^2+n cons 单元创建的最坏情况界，其中n 仅表示列表中左注入的数量。这个小例子展现了 AARA 的若干关键特性：它能够对棘手的递归模式进行紧确分析，例如快速排序中出现的递归模式；它具有组合性，能够轻松处理过程间代码；它给出精确的多项式界，而非渐近界；并且能够推导树状数据结构上函数的界，同时考虑数据的形状。

函数式程序的 AARA 基于类型系统，类型推导充当所推导界限的证明证书。类型推断被归约为高效的线性规划，而 AARA 能够自然地推导出非单调资源（如内存）的资源使用峰值界限，因为这些资源可能会在求值过程中重新变得可用。通过线性规划推断非线性界限的关键创新，是使用经过精心选择的一组资源函数，作为物理学家方法中摊销分析所使用的势函数模板。

尽管具有诸多优势，最先进的 AARA 在实际应用中仍存在一些局限，包括不支持通用的正则递归类型。[1] 例如，考察函数 sort_lefts_tree。该函数与上面的函数类似，但将列表替换成了玫瑰树：

let 快速排序 : int 列表 →int 列表 =. . . 类型 'a 树 = Tree of 'a * 'a 树 列表 let filter_map_tree : ('a →'b option) →' a 树 →'b 列表 =. . . let sort_lefts_tree (t : (int + 布尔类型) 树) = 快速排序 (filter_map_tree find_left t)

玫瑰树可以具有任意且可变的分支因子，这是通过以嵌套方式定义树及其子树列表实现的。现有的 AARA 系统无法为此函数推导出界。AARA 无法推导出作为一般代数数据结构函数的界，这构成了一个实际的

[1]我们使用“正则递归类型”一词来指代可能包含非平凡嵌套递归、但所有递归都发生在基础种类上的类型。

缺陷。自 Hoffmann 和 Hofmann 在 2010 [19] 中引入多项式 AARA 以来，如何扩展多项式 AARA 以处理正则递归类型一直是一个悬而未决的问题。核心挑战在于，为这些类型找到一类既具有表达能力、又受到足够约束以确保在类型检查所需操作下保持封闭的势函数。

我们通过为正则递归类型引入一种满足 AARA 要求的资源多项式概念，弥补了这一长期存在的空白。我们借鉴了以往的方法，但最终采用了我们认为更符合类型结构的代数化视角。具体而言，生成基多项式的索引几乎与其分类的值完全对应。我们的资源多项式是此前提出的所有 AARA 多项式资源函数的推广 [16], [19], [17], [18]，也可以视为给定递归类型的值所对应的多项式的一般概念。我们给出了移位和共享这两种构造，分别证明资源多项式对离散差分和乘法封闭；二者结合后，使 AARA 能够仅使用线性规划推断资源界。我们将这些操作及其他操作描述为自由半模上的线性映射，从而抽象掉以往论述中的一些繁琐细节。最后，我们为 FPC 的一个版本构建了一个类型系统（FPC 是一种带有递归类型 [20] 的值调用语言），并在其中加入资源使用，使其能够利用这些资源多项式；随后，我们通过逻辑关系论证证明了该类型系统的可靠性。

II. 概述

首先，我们回顾 AARA（§II-A），详细介绍列表的势函数（§II-B），并阐述我们将其扩展到正则递归类型背后的直觉（§II-C）。

A. AARA 简介

AARA 是一种基于类型的技术，可自动推断操作数据结构的程序的最坏情况成本界。它采用 Tarjan 和 Sleator 提出的物理学家方法的形式化版本 [21]，为数据结构赋予势函数，进而用于摊还分析。随后，在程序执行过程中跟踪给定上下文中可用的势，以确保现有势足以支付下一次状态转换的代价以及转换后状态的势。

为实现物理学家方法的自动化，AARA 为每种类型定义一组固定的势函数。这些势函数必须满足某些（闭包）性质，以便将势的跟踪顺利整合到类型规则中。这种整合是实现自动化的关键，因为势的跟踪可以表示为线性约束，并与类型检查或推断同步生成。随后可由线性规划求解器求解这些约束，得到最终带有资源界注释的类型。

示例：过滤_映射：为了展示 AARA 方法的基础知识，我们逐步构建引言中介绍的动机示例。与之前一样，假设我们关注将创建的 cons 单元数量，并以此作为成本模型。首先，考虑标准列表函数 过滤_映射：$(\tau \to$ 选项$(\sigma)) \to$列表$(\tau) \to$ 列表(σ)，其实现如下：

```
let rec filter_map f l = match l with
  | [] → []
  | x :: l' → match f x with
              | Some y → y :: filter_map f l'
              | None → filter_map f l'
```

对表达式 filter_map f l 的求值会将 f 应用于 l 的每个元素，并将 Some 结果收集到输出列表中。求值的代价取决于高阶参数f的代价。首先假设 f 的代价为 0。那么，filter_mapf l 的代价至多为列表l的长度 $|l|$。以下类型可以表示这一界限。

$$\mathsf{filter_map} : (\langle \mathsf{int}^0 + \mathsf{bool}^0, 0\rangle \to \langle \mathsf{option}^0(\mathsf{int}), 0\rangle) \to \langle \mathsf{list}^1(\mathsf{int}^0 + \mathsf{bool}^0), 0\rangle \to \langle \mathsf{list}^0(\mathsf{int}), 0\rangle$$

高阶参数的类型 $\langle int^0 + bool^0, 0\rangle \to \langle \mathsf{option}^0(int)\ 0\rangle$ 表明，该函数不需要任何输入势能，也不会为其输出分配任何势能。列表类型 $\mathsf{list}^1(int^0 + bool^0)$ 表示列表参数为列表中的每个元素携带一个势能单位，这反映了待证明的界限。在此情况下，输出势能 $\langle \mathsf{list}^0(int)\ 0\rangle$ 为零，但一般而言，它对于分析的组合性十分重要。要了解如何使用结果的势能，请考虑以下类型推导：

$$\mathsf{filter_map} : (\langle \mathsf{int}^1 + \mathsf{bool}^0, 0\rangle \to \langle \mathsf{option}^1(\mathsf{int}), 0\rangle) \to \langle \mathsf{list}^1(\mathsf{int}^1 + \mathsf{bool}^0), 0\rangle \to \langle \mathsf{list}^1(\mathsf{int}), 0\rangle$$

这里，得到的列表中每个元素都携带 1 个势能单位。为了覆盖这部分额外的势，输入列表现在的类型为 $\mathsf{list}^1(int^1 + bool^0)$，这表示每个元素有 1 个势能单位，而每个形式为 $\mathsf{inl}\, n$ 的元素还有 1 个额外的势能单位。高阶参数的类型表示：如果参数的形式为 $\mathsf{inl}\, n$，则需要 1 个势能单位；否则不需要。求值后，如果结果是 Somen，则还剩下 1 个单位；否则剩下 0 个。

filter_map 的正确类型注解取决于函数的使用上下文。一般类型可以用抽象注解和线性约束来描述：

$$\mathsf{filter_map} : (\langle \mathsf{int}^{q_1} + \mathsf{bool}^{q_2}, p_0\rangle \to \langle \mathsf{option}^{q_3}(\mathsf{int}), p_0'\rangle) \to \langle \mathsf{list}^{r_1}(\mathsf{int}^{r_2} + \mathsf{bool}^{r_3}), p_1\rangle \to \langle \mathsf{list}^{r_4}(\mathsf{int}), p_1'\rangle$$

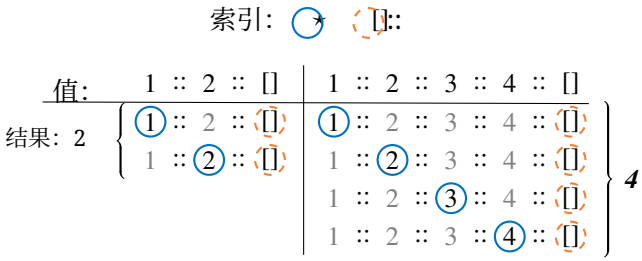
$$r_1 \geq p_0 + 1, r_2 \geq q_1, r_3 \geq q_2, p_1 \geq p_1', q_3 + p_0' \geq r_4$$

需要明确的是，这种符号表示无法在类型系统中表达。然而，作为类型推断的一部分，这种形式会以符号值作为元变量推导出来；随后使用线性规划求解约束，以找到一个解，将该解代入后，便可在类型系统中得到一个具体判断。因此，一个基本要求是，从列表向其头部和尾部转移势的过程能够用线性约束表达。对于线性势函数而言，这一点很直接，因为头部的注释就是元素类型的注释，而尾部的注释就是经过模式匹配的列表的注释。

B. 列表势函数

为了超越线性势，多项式 AARA 将记号 $\langle L^{q_1}(A)\ q_0\rangle$ 扩展为 $L^{\langle q_0,q_1,\ldots,q_m\rangle}(A)$，其中 $\vec{q}$是指定多项式 [19]的系数向量。如何维持前述要求，即对列表进行析构时只产生线性约束，则不那么明确。事实证明，答案在于巧妙地选择基：系数 (q_i) 对应的是二项式系数 $\binom{n}{i}$构成的基，而不是单项式 n^i，这是因为它们具有加法移位函数 $\lhd(q_0,\ldots,q_m) = (q_0+q_1,\ldots,q_{m-1}+q_m,q_m)$。这是一个用于规定如何保持势的线性函数，也就是说，在 n 上计算 $\lhd(\vec{q})$ 等于在 $n+1$上计算 $\vec{q}$。这种线性移位函数的概念最终成为一种关键的指导性抽象，确保模式匹配类型规则的生成过程中只产生线性约束。

将 AARA 扩展到多变量注释时，这一原理依然适用——包括形如 $m{\cdot}n$ 的项，例如计算两个列表的笛卡尔积时可能需要用到这些项——但系数向量表示法却不再适用。为此，多变量 AARA 引入索引来构成势函数的基 [17]。直观来说，这种做法将“单项式”命名的概念推广到了形如 $\binom{n}{2}$ 或 $\binom{n}{3}\binom{m}{2}$ 的项。列表索引的形式为 $[i_1,\ldots,i_n]$，其中每个 i_j 都是列表元素类型的索引。这种列表索引表示对列表中与内部索引相匹配的元素组合进行计数。通过一些示例或许最容易说明这一点；为求简明，我们将只采用单变量示例，但很容易将其扩展到多变量情形。首先，来看列表上的索引 $[\star] = \star :: \text{nil}$，它计算元素 $\star$ 后接 nil 的方式数，也就是列表的长度。直观地看，分别对两个长度不同的列表求其值：



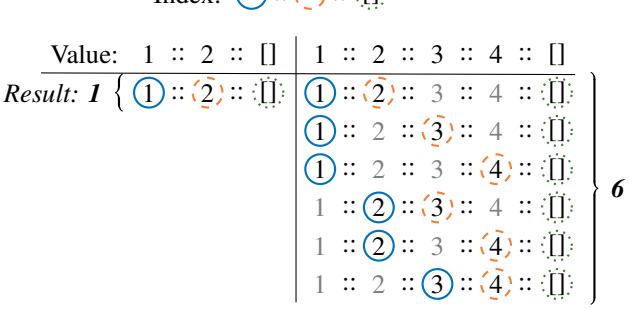
请注意，正如每次求值中的两个圆圈所示，每次求值中都有两个匹配项：一个 Cons 单元，以及结尾的 Nil。列表索引求值的关键在于，它可以完全根据索引和列表元素的表头与表尾在局部进行表述（其中，ϕ 表示对一个值求值索引的函数）：

$$\phi_{i::is}(v :: vs) = \phi_i(v) \cdot \phi_{is}(vs) + \phi_{i::is}(vs)$$

该函数首先统计包含头元素的组合数，然后加上不包含头元素的组合数。从这一表达方式可以直接得到一个类似的移位函数：$\lhd(i :: is) = (i, is) + (\star, i :: is)$，其中结果在 (v, vs) 上求值给定一个列表 $v :: vs$。请注意，这与二项式系数的定义有多么相似！

作为这些索引如何用于类型的示例，让我们回到之前介绍的第二种 filter_map f 类型，即$\langle \text{list}^1(\text{int}^1 + \text{布尔类型}^0)\ 0\rangle \to \langle \text{list}^1(int)\ 0\rangle$。使用索引表示时，该函数要求其参数具有势$2\cdot[\text{左注入}\star] + 1\cdot[\text{右注入}\ \star]$，并返回具有势 $1\cdot[\star]$ 的值。

为了实现对 快速排序进行类型检查的目标，先考虑索引 $[\star;\star]$ 的一些求值：



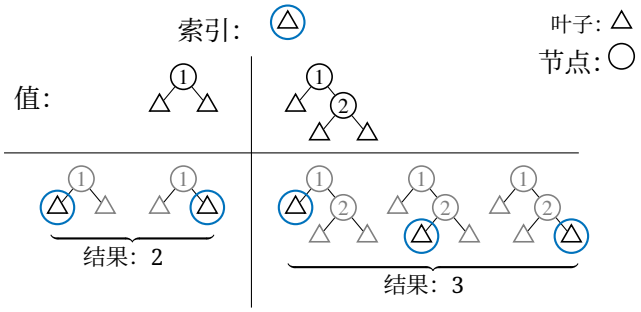
正如预期，我们发现该索引对应于 $\binom{n}{2}$。因此，鉴于我们知道快速排序 的代价为 n^2，我们可以使用索引将其参数所需的势表示为$2\cdot[\star;\ \star]+1\cdot[\star]$。最后，我们可以考察原始函数 排序_左侧列表_列表。这里可以看出，它必须需要输入势 $2\cdot[\text{左注入}\star;\ \text{左注入}\star]+2\cdot[\text{左注入}\ \star]$–过滤_映射消耗其中的$1\cdot[\text{左注入}\star]$ 部分，并将其余部分传递给快速排序。

C. 扩展到正则递归类型

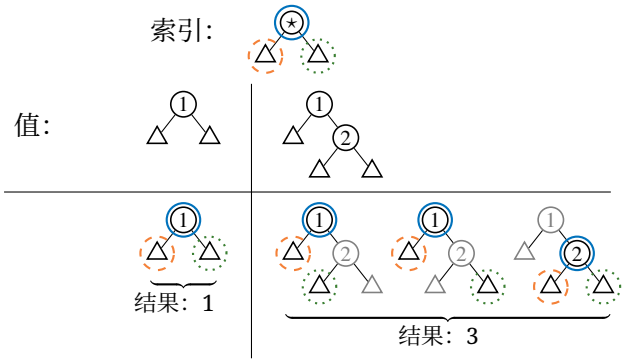
然而，这些索引并不能显然地推广到正则归纳类型。Jost 等人 [16] 处理正则归纳类型上的势，但仅限于非常受限的单变量线性势设定，这实际上只是对构造子的计数。Hoffmann 等人 [17] 及其后续工作处理了表达能力更强的势函数，但不支持正则归纳类型，并且就势而言，甚至将二叉树视为列表。树索引与列表索引完全相同，而树值只是通过先序遍历展平后得到的自身的列表版本。这导致树的组合结构完全丧失。

让我们来探索另一种设计。首先，我们知道必须保留某种线性移位函数。另一个线索来自 Hoffmann 等人 [17], , 他们顺带指出，类型 τ 的索引本质上遵循类型 τ 的值的结构。我们发现，他们的想法确实有道理。我们考虑的索引几乎与其所描述类型的值一一对应。为了建立直觉，我们先针对一些具体的数据类型举例，然后再描述一般情况。

1）垫脚石：二叉树：我们先来考察二叉树的情况。在下面的图示中，树节点用圆形表示，而叶节点用三角形表示。考虑在两棵不同的树上计算“叶子”索引：



这会计算树中叶节点的数量，就像第一个列表索引示例（由一个 cons 节点组成）计算列表中 cons 节点的数量一样。现在，让我们来看下一个最简单的索引，即连接两个叶节点的节点：



右侧的求值起初可能令人困惑——难道不是只有一个与索引匹配的子树吗？答案可以类比此前介绍的列表上的组合求值来理解：在索引施加的顺序约束下，会考虑构造子的所有可能组合。

这些示例都是二叉树规则的实例；这些规则同样完全在局部定义：

$$\phi_{\triangle}(\triangle) = 1$$

$$\phi_{\overset{i}{\underset{i_1}{\triangle}\ \underset{i_2}{\triangle}}}(\triangle) = 0$$

$$\phi_{\triangle}\left(\overset{v}{\underset{t_1}{\triangle}\ \underset{t_2}{\triangle}}\right) = \phi_{\triangle}(t_1) + \phi_{\triangle}(t_2)$$

$$\phi_{\overset{i}{\underset{i_1}{\triangle}\ \underset{i_2}{\triangle}}}\left(\overset{v}{\underset{t_1}{\triangle}\ \underset{t_2}{\triangle}}\right) = \phi_i(v) \cdot \phi_{i_1}(t_1) \cdot \phi_{i_2}(t_2) + \phi_{\overset{i}{\underset{i_1}{\triangle}\ \underset{i_2}{\triangle}}}(t_1) + \phi_{\overset{i}{\underset{i_1}{\triangle}\ \underset{i_2}{\triangle}}}(t_2)$$

2) 尤里卡！：因此，一般情况的关键在于注意到列表规则与二叉树规则之间的对应关系：

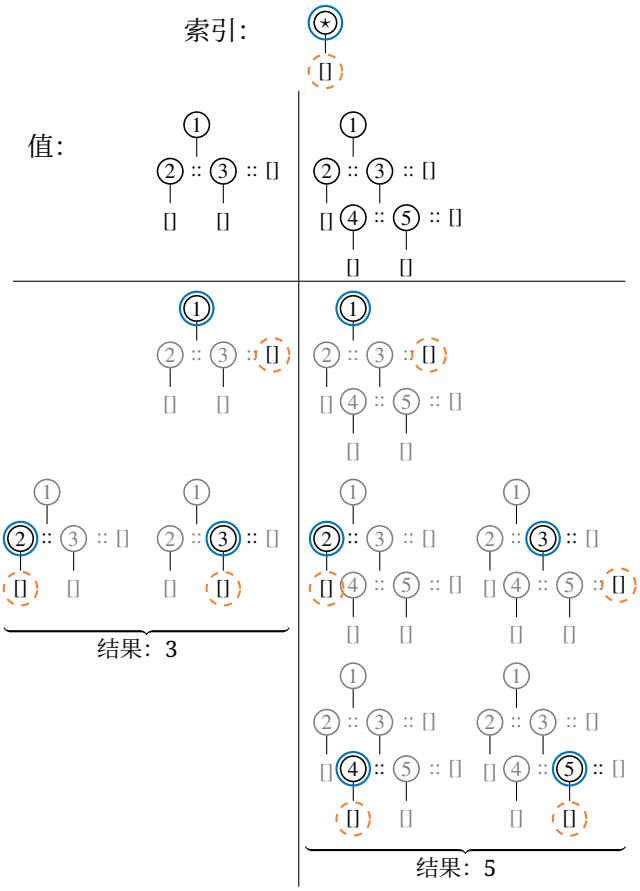
$$\phi_{i::is}(v :: vs) = \phi_i(v) \cdot \phi_{is}(vs) + \phi_{i::is}(vs)$$

$$\phi_{\overset{i}{\underset{i_1}{\triangle}\ \underset{i_2}{\triangle}}}\left(\overset{v}{\underset{t_1}{\triangle}\ \underset{t_2}{\triangle}}\right) = \phi_i(v) \cdot \phi_{i_1}(t_1) \cdot \phi_{i_2}(t_2) + \phi_{\overset{i}{\underset{i_1}{\triangle}\ \underset{i_2}{\triangle}}}(t_1) + \phi_{\overset{i}{\underset{i_1}{\triangle}\ \underset{i_2}{\triangle}}}(t_2)$$

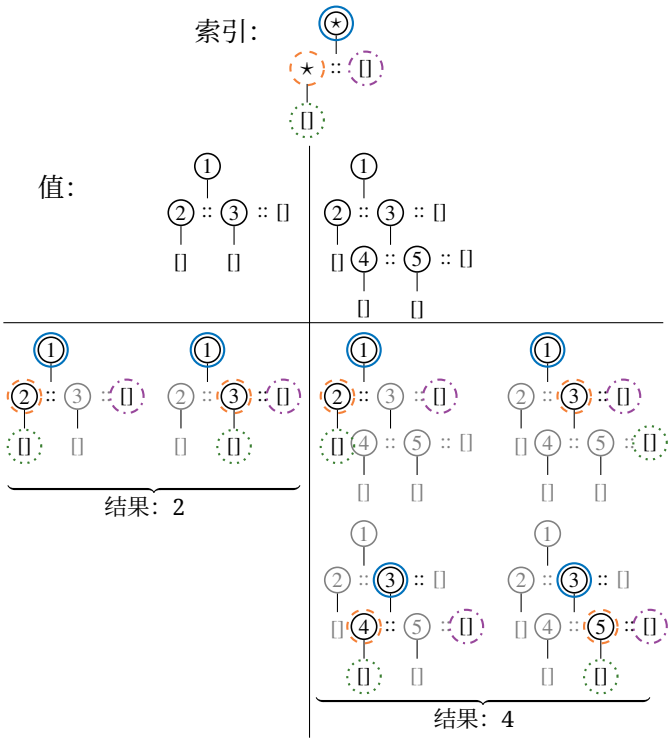
关键直觉。要在构造器处对索引求值，首先在直接构造器处对其求值，然后将该结果加到原始索引在所有直接子节点处的求值结果上。

请注意，它满足我们期望的性质：在直接构造器求值时通过乘法实现多变量；只对直接子节点递归求值，从而实现结构相关；更重要的是，它暗示了一种转换函数，其形式与这一构造完全一致。观察到这一点后，我们将留待 §III-A 对其作形式化定义，但至少会先澄清该规范中的一个歧义：构造器的“直接子节点”是什么？

3) 关键所在：玫瑰树：Cons 单元或树节点的直接子节点显而易见，但对于最初启发我们的数据类型——玫瑰树，其直接子节点就不那么明显了。让我们再次看几个例子，从最简单的索引开始：



以及一个更复杂的索引：



$$
\begin{array}{lrcll}
\text{类型} & \tau & ::= & \alpha & \text{类型变量} \\
 & & \mid & \mathbf{1} & \text{Unit} \\
 & & \mid & \tau_1 \times \tau_2 & \text{积} \\
 & & \mid & \tau_1 + \tau_2 & \text{Sum} \\
 & & \mid & \langle \tau_1 \to \tau_2,\ \Theta,\ \Theta_{\mathsf{cf}} \rangle & \text{箭头} \\
 & & \mid & \mu\alpha.\,\tau & \text{同递归} \\
\text{值} & v & ::= & \mathsf{tt} \mid \text{左注入 } v \mid \text{配对}(v_1;\ v_2) \mid \text{右注入 } v & \\
 & & \mid & \text{函数}(f, x.\, e) \mid \text{折叠 } v &
\end{array}
$$

$$
\frac{}{\mathsf{tt} \in \mathcal{V}(\mathbf{1})} \qquad \frac{}{\text{函数}(f, x.\, e) \in \mathcal{V}(\langle \tau_1 \to \tau_2, \Theta, \Theta_{\mathsf{cf}} \rangle)}
$$

$$
\frac{v_1 \in \mathcal{V}(\tau_1) \qquad v_2 \in \mathcal{V}(\tau_2)}{\text{配对}(v_1;\ v_2) \in \mathcal{V}(\tau_1 \times \tau_2)} \qquad \frac{v_1 \in \mathcal{V}(\tau_1)}{\text{左注入 } v_1 \in \mathcal{V}(\tau_1 + \tau_2)}
$$

$$
\frac{v_2 \in \mathcal{V}(\tau_2)}{\text{右注入 } v_2 \in \mathcal{V}(\tau_1 + \tau_2)} \qquad \frac{v \in \mathcal{V}([\mu\alpha.\,\tau/\alpha]\tau)}{\text{折叠 } v \in \mathcal{V}(\mu\alpha.\,\tau)}
$$

图 1：类型、值及值的类型判断

要指定玫瑰树节点的直接子节点，我们借助列表关于直接子节点的定义：出现在玫瑰树节点的列表中的任意节点，都是它的直接子节点。将外层类型的递归求值问题下推至内层类型，正是解决之道。拟人地说，玫瑰树可以在任意给定的列表节点中识别出唯一可能出现的树（位于 Cons 单元中）；随后，列表便能利用这一信息，遍历列表的递归出现。这一直觉在 §III-A 中再次得到形式化阐述。

回到最初启发我们的 filter_map_树，像第二个 filter_map 的类型推导中那样，为同一个函数指定所需的势，现在只需写作 $2 \cdot \text{Tree}(\text{左注入 } \star, []) + 1 \cdot \text{Tree}(\text{右注入 } \star, [])$，也就是 $2m + 1n$ 其中 m 是含有整数的节点数，n 是含有布尔值的节点数。对于整体的 sort_lefts_树，同样自然的写法是 $2 \cdot \text{Tree}(\text{左注入 } \star, [\text{Tree}(\text{左注入 } \star, [])]) + 2 \cdot \text{Tree}(\text{左注入 } \star, [])$，原因与列表情形大致相同。令人惊讶的是，这些索引看起来几乎与等价列表函数的索引一样简单；我们认为，这有力地表明了其优雅性。

III. 资源多项式

与之前的 AARA 类型系统一样，资源多项式充当我们语言中为带类型的值分配势的机制。我们对其理论的核心贡献，是将以往系统中的有界分支树类型推广到更一般的代数类型，其中还可能包含相互递归类型。在本节中，我们首先形式化定义这些势函数，然后给出类型系统所必需的相关运算，并继续使用运行示例来阐释这些定义。

A. 资源多项式定义

类型和值：为了说明资源多项式究竟为哪些对象分配势，我们首先在图1中给出定义资源多项式所需的类型和值。除箭头类型外，所介绍的类型都是标准的；箭头类型的具体细节与资源多项式无关，并在第 IV 节中解释。我们还在图1中给出了针对给定类型 τ 的语法上有效的值集合 $\mathcal{V}(\tau)$ 的推理规则。记号 $[\sigma/\alpha]\tau$ 指的是将 σ 在 τ 中替换为 α 时所采用的避免捕获的替换。请注意，这些类型规则并不能保证语言语义所需的任何性质；特别是，这里的函数类型实际上几乎不受限制。相反，它们只是为了确保势函数能够在类型匹配的索引和值上进行求值。

按照贯穿全文的示例，我们可以定义类型 bool $\mathbf{1} + \mathbf{1}$、list(τ) $\mu\alpha.\,\mathbf{1} + \tau \times \alpha$ 和 tree(τ) $\triangleq \mu\beta.\,\tau \times \text{list}(\beta) = \mu\beta.\,\tau \times (\mu\alpha.\,\mathbf{1} + \beta \times \alpha)$，其值构造子分别为 True $\triangleq$ inl tt 和 False $\triangleq$ inr tt、Nil $\triangleq$ fold (inl tt) 和 Cons(h, t) $\triangleq$ fold (inr (pair(h; t))), 以及 Tree(x, t) $\triangleq$fold (pair(x; t))。我们使用记号 $[v_1, \ldots, v_n]$ 表示 Cons(v_1, . . . (Cons(v_n, Nil)))。

索引：资源多项式由带有理系数的“单项式”基多项式之和组成。我们使用索引来命名这些基多项式。图 2 展示了给定类型 τ 的索引集合 $\mathcal{I}(\tau)$ 的推理规则。它们几乎完全对应于语法值 $\mathcal{V}(\tau)$，只是为递归类型增加了一个“end”索引。[2] 对索引的一种直观理解是，将其视为模式匹配中的模式，用于指定值进行比较的形状。

[2]类型系统的若干部分依赖于对常量势的描述；因此，我们加入 end，以便为原本缺少此类索引的递归类型描述常量势。

基多项式索引 $\boxed{i \in \mathcal{I}(\tau)}$

$$\frac{}{\mathsf{tt} \in \mathcal{I}(1)} \qquad \frac{i_1 \in \mathcal{I}(\tau_1) \qquad i_2 \in \mathcal{I}(\tau_2)}{\text{配对}(i_1;\ i_2)\ \in \mathcal{I}(\tau_1 \times \tau_2)}$$

$$\frac{i_1 \in \mathcal{I}(\tau_1)}{\text{左注入}\, i_1 \in \mathcal{I}(\tau_1 + \tau_2)} \qquad \frac{i_2 \in \mathcal{I}(\tau_2)}{\text{左注入}\, i_2 \in \mathcal{I}(\tau_1 + \tau_2)}$$

$$\frac{}{\lambda \in \mathcal{I}(\langle \tau_1 \to \tau_2, \Theta, \Theta_{\mathsf{cf}} \rangle)} \qquad \frac{i \in \mathcal{I}([\mu\alpha.\,\tau/\alpha]\tau)}{\text{折叠}\ i \in \mathcal{I}(\mu\alpha.\,\tau)}$$

$$\frac{}{\mathsf{end}\ \in \mathcal{I}(\mu\alpha.\,\tau)}$$

递归出现索引 $\boxed{\mathcal{M}\{\alpha.\tau\}(i)}$

$$\begin{aligned}
\mathcal{M}\{\alpha.\alpha\}(i) &= \{i\} \\
\mathcal{M}\{\alpha.\top_\mu\}(i) &= \emptyset \\
\mathcal{M}\{\alpha.1\}(i) &= \emptyset \\
\mathcal{M}\{\alpha.\tau_1 + \tau_2\}(i) &= \{\mathsf{inl}\, j \mid j \in \mathcal{M}\{\alpha.\tau_1\}(i)\} \cup \\
&\quad \{\mathsf{inr}\, j \mid j \in \mathcal{M}\{\alpha.\tau_2\}(i)\} \\
\mathcal{M}\{\alpha.\tau_1 \times \tau_2\}(i) &= \{\mathsf{pair}(j;\, c) \mid j \in \mathcal{M}\{\alpha.\tau_1\}(i), \\
&\qquad\qquad c \in \mathcal{C}(\tau_2)\} \cup \\
&\quad \{\mathsf{pair}(c;\, j) \mid c \in \mathcal{C}(\tau_1), \\
&\qquad\qquad j \in \mathcal{M}\{\alpha.\tau_2\}(i)\} \\
\mathcal{M}\{\alpha.\tau_1 \to \tau_2\}(i) &= \emptyset \\
\mathcal{M}\{\alpha.\mu\beta.\,\tau\}(i) &= \{\mathsf{fold}\, j \mid j \in \mathcal{M}\{\alpha.[\top_\mu/\beta]\tau\}(i)\}
\end{aligned}$$

常量索引集 $\boxed{\mathcal{C}(\tau)}$

$$\begin{aligned}
\mathcal{C}(1) &= \{\mathsf{tt}\} \\
\mathcal{C}(\tau_1 \times \tau_2) &= \{\text{配对}(i_1;\ i_2) \mid i_1 \in \mathcal{C}(\tau_1), \\
&\qquad\qquad i_2 \in \mathcal{C}(\tau_2)\} \\
\mathcal{C}(\tau_1 + \tau_2) &= \{\text{左注入}\, i_1 \mid i_1 \in \mathcal{C}(\tau_1)\} \cup \\
&\quad \{\mathsf{inr}\ i_2 \mid i_2 \in \mathcal{C}(\tau_2)\} \\
\mathcal{C}(\tau_1 \to \tau_2) &= \{\lambda\} \\
\mathcal{C}(\mu\alpha.\,\tau) &= \{\mathsf{end}\} \\
\mathcal{C}(\top_\mu) &= \{\mathsf{end}\}
\end{aligned}$$

基础多项式求值 $\boxed{\phi_i(v:\ \tau)}$

$$\begin{aligned}
\phi_{\mathsf{tt}}(\mathsf{tt}:\ 1) &= 1 \\
\phi_{\mathsf{pair}(i_1;\ i_2)}(\text{配对}(v_1;\ \ v_2):\ \ \tau_1 \times \tau_2) &= \phi_{i_1}(v_1:\ \tau_1) \cdot \phi_{i_2}(v_2:\ \tau_2) \\
\phi_{\mathsf{inl}\, i_1}(\text{左注入}\ v_1:\ \ \tau_1 + \tau_2) &= \phi_{i_1}(v_1:\ \ \tau_1) \\
\phi_{\mathsf{inl}\, i_1}(\text{右注入}\ v_2:\ \ \tau_1 + \tau_2) &= 0 \\
\phi_{\mathsf{inr}\, i_2}(\text{左注入}\ v_1:\ \ \tau_1 + \tau_2) &= 0 \\
\phi_{\mathsf{inr}\, i_1}(\text{右注入}\ v_2:\ \ \tau_1 + \tau_2) &= \phi_{i_2}(v_2:\ \ \tau_2) \\
\phi_{\lambda}(\text{函数}(f, x.\, e):\ \ \langle \tau_1 \to \tau_2, \Theta, \Theta_{\mathsf{cf}} \rangle) &= 1 \\
\phi_{\mathsf{fold}\, i}(\text{折叠}\ v:\ \ \mu\alpha.\,\tau) &= \phi_i(v:\ \ [\mu\alpha.\,\tau/\alpha]\tau) \\
&\quad + \textstyle\sum_{k \in \mathcal{M}\{\alpha.\tau\}(\text{折叠}\ i)} \phi_k(v:\ \ [\mu\alpha.\,\tau/\alpha]\tau) \\
\phi_{\mathsf{end}}(\text{折叠}\ v:\ \ \mu\alpha.\,\tau) &= 1
\end{aligned}$$

图 2：基本基多项式索引构造。

然而，匹配模式是一种二元决策，而索引统计的是值中的出现次数。

沿用我们的示例，True 和 False 都是布尔类型的索引，并且分别与这些值完全匹配；Nil 和 end 是列表(τ)的索引，与任意列表值恰好匹配一次；Cons(tt, Nil) 与任意列表(1)值的匹配次数等于列表的长度；Node(tt,Nil) 则与任意树(1)值的匹配次数等于树中的节点数。

常量索引集：图 2 给出了一个函数，它为任意类型 τ 定义索引集合 $\mathcal{C}(\tau)$，使得这些索引在任意类型为 τ 的值上的求值之和恰好为 1。该定义可以直接根据索引求值的定义得到，后文将很快给出后者的定义。（我们还给出了 $\top_\mu$ 的定义；它是一种语法，用于 $\mathcal{M}$ 的求值过程中，在展开递归类型时替代有界类型变量，从而确保每种递归类型只展开一次。）

沿用我们的示例，我们有 $\mathcal{C}$(布尔类型) ={True,False}，它确实涵盖了布尔类型的所有可能值；还有 $\mathcal{C}$(列表(τ)) = $\mathcal{C}$(树(τ)) = {end}，它构成了任意递归类型的常量索引集合。

递归出现索引集：这是实现向更一般的代数类型和互归纳类型扩展的关键洞见。图 2 中定义的函数 $\mathcal{M}\{\alpha.\tau\}(i)$，其中 τ 是一种除了 α 和i 之外不含自由类型变量的类型，而 α 表示该类型的一个索引；该函数返回一个索引集合，这些索引对应于将 i 放置在 τ中 α的每一次出现处。具体而言，该函数的各种情况如下：

α。我们发现了一个 α 的出现位置，因此 i 在此处。$\top_\mu$。这表示在展开该递归类型时，某个递归类型的出现位置被替换了，而该递归类型并非 α 所指的类型。该递归类型中 α 的任何出现位置，都已经由应用于展开位置的折叠处理。1. 此处未找到 α 的出现位置。$\tau_1 + \tau_2$。无论该值最终是左注入还是右注入，其中任一方都可能包含类型为 α 的值，因此我们会考虑两种情况。$\tau_1 \times \tau_2$。在这里， α 可能出现在配对的两个投影中，但我们一次只想考虑一个，因此我们考虑在第一个投影中查找值

而第二个投影包含任意内容，反之亦然。$\tau_1 \to \tau_2$。我们将函数视为不透明的，不包含 α 值。

$\mu\beta.\tau$。下面是处理嵌套递归类型时的关键情形。如 §II-C3 中所述，在此处的索引中引入折叠会导致该递归过程在索引求值期间再次发生，但此时针对的是 β 而不是 α。这种对“递归展开”的延迟，使得嵌套递归求值成为可能，同时又不会让该过程生成无限数量的索引。

沿用前面的示例，我们有

$$\mathcal{M}\{\alpha.\mathsf{bool}\}(i) = \emptyset,$$

因为 α 在 bool 中不是自由的；

$$\mathcal{M}\{\alpha.\mathbf{1} + \mathsf{bool} \times \alpha\}(i) = \{\mathsf{inr}\,(\mathsf{pair}(\mathsf{True};\, i)), \mathsf{inr}\,(\mathsf{pair}(\mathsf{False};\, i))\}$$

（其中 $\mathbf{1} + \mathsf{bool} \times \alpha$ 是移除递归绑定符后的 list(bool)），因为布尔值列表中的所有递归出现都位于 cons 单元的尾部，而布尔值位于头部；并且

$$\mathcal{M}\{\beta.\mathsf{bool} \times \mathsf{list}(\beta)\}(i) = \{\mathsf{pair}(\mathsf{False};\, \mathsf{fold}\,(\mathsf{inr}\,(\mathsf{pair}(i;\, \mathsf{end})))),\ \mathsf{pair}(\mathsf{True};\, \mathsf{fold}\,(\mathsf{inr}\,(\mathsf{pair}(i;\, \mathsf{end}))))\}$$

（其中 bool $\times$ list(β) 是移除递归绑定符后的 tree(bool)），因为布尔值玫瑰树中的所有递归出现都位于子节点列表的某个 cons 单元中。最后一个示例值得进一步仔细考察，以理解这一机制应用于互归纳类型时的直觉：虽然玫瑰树的直接递归出现次数没有上限，因此乍看之下可能需要无限多个索引来表示，但与列表本身对应的折叠能够找到其所有递归出现，从而允许用有限数量的索引捕获任意数量的后代。

索引求值：最后，我们来到了图 2 中索引求值函数 $\phi_i(v :\ \tau)$ 的定义。该函数对值 v 上类型 τ 的索引 i 求值，从而得到对 v 中 i 的匹配次数进行“计数”的结果。除了对索引折叠 i 求值时的情况外，定义都很直接，因此我们只详细解释这条规则。对类型为 $\mu\alpha.\tau$ 的值折叠 v 上的索引折叠 i 求值时，我们要找出 v 中所有可能与 i 匹配的位置。第一个可能出现匹配的位置就是值 v 本身，这由项 $\phi_i(v\colon [\mu\alpha.\tau/\alpha]\tau)$ 处理。不过，我们还要考虑 v 中类型递归位置上的匹配；如上所述，这些位置正是 $\mathcal{M}\{\alpha.\tau\}$ 所标识的。我们要继续在这些位置查找折叠 i 的所有匹配，因此将 $\mathcal{M}\{\alpha.\tau\}$(fold i) 中各个索引的求值结果相加，以统计递归出现次数。

在我们示例类型的几个值上进行索引求值的结果如图 3 所示。

定义了索引求值后，我们可以通过对 τ 进行归纳法来刻画常量索引集的关键性质。

引理 1（常量索引求和）。对于所有类型 τ 和所有值 $v \in \mathcal{V}(\tau)$，$\sum_{i\in\mathcal{C}(\tau)} \phi_i(v\colon\ \tau) = 1.$

Type τ	索引 $i \in \mathcal{I}(\tau)$	值 $v \in \mathcal{V}(\tau)$	结果 $\phi_i(v :\ \tau)$
bool	False	False	1
		True	0
	True	False	0
		True	1
列表(1)	[]	[tt; tt]	1
		[tt; tt; tt; tt]	1
	[tt]	[tt; tt]	2
		[tt; tt; tt; tt]	4
	[tt; tt]	[tt; tt]	1
		[tt; tt; tt; tt]	6
树(1)	end		1
			1
	树(tt, [])		3
			6
	树(tt, [树(tt, [])])		2
			7

图3：索引求值结果示例。

请注意，不同的索引有时可能指向同一个基多项式。例如，类型 list(τ) 的两个索引 end 和 Nil 都表示常值函数。

资源多项式，正式定义：截至目前，我们通过规定其句法指标来描述基多项式；对于类型 τ，资源多项式 $\mathcal{R}(\tau)$ 是带非负有理系数的基多项式的线性组合。

B. 注释

尽管资源多项式是我们在分析中真正关注的对象，但资源多项式最有用的表示形式是我们称为注释 $\mathcal{A}(\tau)$ 的具体化形式。

定义 1（注释）。令 $\mathcal{A}(\tau)$ 表示以 $\mathcal{I}(\tau)$ 为基的自由 $\mathbb{Q}_{\geq 0}$-半模。那么，对于类型τ 而言，注释是 $\mathcal{A}(\tau)$ 中的一个元素。

半模的定义与向量空间相同，区别在于半模定义在半环上，而不是域上。由于这一差异，从一般意义上说，半模缺少向量空间的大部分结构；然而，自由半模具有基，因此可以认为其行为与向量空间相当相似。关于半模的更详细概述见附录 A。

我们将注释表示为 P 或 Q，并将 p_i 定义为与索引 i 对应的系数。这样，我们就可以将资源多项式恢复为势函数

$$\Phi(v : \langle\tau; P\rangle) \triangleq \sum_{i\in\mathcal{I}(\tau)} p_i \cdot \phi_i(v : \tau).$$

请注意，$P \mapsto \Phi(\cdot : \langle \tau;\ P\rangle)$ 是从注释到资源多项式的线性映射，而 $P \mapsto \Phi(v : \langle \tau;\ P\rangle)$ 是一种线性形式。除了半模上的基本运算外，我们还使用注释上的以下概念：

- 将有限指标集的集合强制转换为注释 $B \subseteq \mathcal{I}(\tau)$，其中 $b_i = 1$ 若 $i \in B$，否则为 0。
- 预序 $P \le Q$定义为：对于所有 i，$p_i \le q_i$。注意，$P \le Q$ 等价于扩张序，即存在一个 R 使得 $Q = P + R$，因此 $\Phi(v : \langle \tau;\ \cdot\rangle)$ 保持该序。
- 还要注意，由于 $\mathcal{I}(\tau_1 \times \tau_2) \cong \mathcal{I}(\tau_1) \times \mathcal{I}(\tau_2)$，我们有 $\mathcal{A}(\tau_1 \times \tau_2) \cong \mathcal{A}(\tau_1) \otimes \mathcal{A}(\tau_2)$。于是 $\Phi(\mathsf{pair}(v_1;\ v_2) : \langle \tau_1 \times \tau_2;\ P \otimes Q\rangle) = \Phi(v_1 : \langle \tau_1;\ P\rangle) \cdot \Phi(v_2 : \langle \tau_2;\ Q\rangle)$。我们将使用记号 $\mathsf{pair} : \mathcal{A}(\tau_1) \otimes \mathcal{A}(\tau_2) \to \mathcal{A}(\tau_1 \times \tau_2)$ 和 $\mathsf{pair}^{-1} : \mathcal{A}(\tau_1 \times \tau_2) \to \mathcal{A}(\tau_1) \otimes \mathcal{A}(\tau_2)$，以明确标记该同构的两侧。
- 类似地，我们有线性映射 $\mathsf{inl} : \mathcal{A}\,\tau_1 \to \mathcal{A}\,\tau_1 + \mathcal{A}\,\tau_2$ 及其回缩 $\mathsf{inl} : (\,)\,(\,)$，$\mathsf{inr}$ 也同样如此。具体而言，$\mathsf{inl}^{-1} : \mathcal{A}\,\tau_1 + \mathcal{A}\,\tau_2 \to \mathcal{A}\,\tau_1$ 是满足以下定义方程的线性映射：$\mathsf{inl}^{-1}(\mathsf{inl}\ i) = i$ 以及 $\mathsf{inl}^{-1}(\mathsf{inr}\ i) = 0$。
- 记号 $\mathsf{id}_{\mathcal{A}(\tau)}$ 表示恒等线性映射 $P \mapsto P : \mathcal{A}(\tau) \to \mathcal{A}(\tau)$。

因此，如果我们想要为长度为n的单位列表 ℓ 定义 n^2 势，就可以使用注释 $P = 1 \cdot [\mathsf{tt}] + 2 \cdot [\mathsf{tt};\ \mathsf{tt}]$, 从而

$$\Phi(\ell : \langle \mathsf{list}(\mathbf{1}); P\rangle) = 1 \cdot \phi_{[\mathsf{tt}]}(\ell : \mathsf{list}(\mathbf{1})) + 2 \cdot \phi_{[\mathsf{tt};\mathsf{tt}]}(\ell : \mathsf{list}(\mathbf{1})) = n + 2\binom{n}{2} = n^2.$$

移位：资源多项式的一个关键要求，是能够在保持势相等的同时折叠和展开递归值。构建我们的系统时，保持这一能力是一项关键的设计约束。我们可以通过加性移位算子实现这一点。

定义 2（加性移位算子）。设 $\mu\alpha.\tau$ 是一种递归类型。那么，加性移位算子 $\lhd$ 是线性映射 $\lhd : \mathcal{A}(\mu\alpha.\tau) \to \mathcal{A}([\mu\alpha.\tau/\alpha]\tau)$，它对应于从基元素 $\mathcal{I}(\mu\alpha.\tau)$ 到 $\mathcal{A}([\mu\alpha.\tau/\alpha]\tau)$ 的函数，其定义为

$$\lhd\,\mathsf{end} \triangleq \mathcal{C}([\mu\alpha.\tau/\alpha]\tau) \quad \lhd(\mathsf{fold}\,i) \triangleq i + \mathcal{M}\{\alpha.\tau\}(\mathsf{fold}\,i).$$

换言之，end 是常数索引，而 $\mathsf{fold}\,i$ 指的是对当前构造子（i 项）以及所有直接子节点（$\mathcal{M}\{\alpha.\tau\}(\mathsf{fold}\ i)$ 项）进行求值。我们希望该算子具备的关键性质如下：

定理 1（移位保持势）。对于任意 $P \in \mathcal{A}(\mu\alpha.\tau)$ 和 $v \in \mathcal{V}([\mu\alpha.\tau/\alpha]\tau)$,

$$\Phi(\mathsf{fold}\,v : \langle \mu\alpha.\tau; P\rangle) = \Phi(v : \langle [\mu\alpha.\tau/\alpha]\tau; \lhd P\rangle).$$

证明。这等价于线性形式 $\Phi(\mathsf{fold}\ v;\ \langle \mu\alpha.\tau;\ \cdot\rangle) = \Phi(v;\ \langle [\mu\alpha.\tau/\alpha]\tau;\ \lhd\cdot\rangle)$ 相等的陈述。根据线性性，只需在基元素 $\mathcal{I}(\mu\alpha.\tau)$ 上证明这一点，此时结论可直接得到。 □

此外还可以证明，移位实际上是一个线性同构 $\mathcal{A}(\mu\alpha.\tau) \cong \mathcal{A}([\mu\alpha.\tau/\alpha]\tau)$。

共享：由于我们需要能够多次使用一个值，因此需要能够将其势拆分到多次使用中。虽然这乍听起来很简单，但由于多变量设置，会出现一些微妙之处：如果它们之间的势最终相互交织，该怎么办？为此，我们需要共享算子，这是一种双线性映射 $\curlyvee : \mathcal{A}(\tau) \to \mathcal{A}(\tau) \to \mathcal{A}(\tau)$。与移位类似，只需在基元素上定义它即可。完整定义见附录 B，但我们认为这是本文的一项关键贡献，因此着重说明关键情形下的定义，即共享两个折叠索引：

$$\begin{aligned}\curlyvee_{\mu\alpha.\tau}(\mathsf{fold}\,i, \mathsf{fold}\,j) \triangleq\ & \mathsf{fold}\left(\curlyvee_{[\mu\alpha.\tau/\alpha]\tau}(i,j)\right) \\ & + \mathsf{fold}\left(\curlyvee_{[\mu\alpha.\tau/\alpha]\tau}(\mathcal{M}\{\alpha.\tau\}(\mathsf{fold}\,i), j)\right) \\ & + \mathsf{fold}\left(\curlyvee_{[\mu\alpha.\tau/\alpha]\tau}(i, \mathcal{M}\{\alpha.\tau\}(\mathsf{fold}\,j))\right) \\ & + \mathsf{fold}\left(\mathcal{N}\{\alpha.\tau\}(\mathsf{fold}\,i, \mathsf{fold}\,j)\right)\end{aligned}$$

此处 $\mathcal{N}$ 类似于 $\mathcal{M}$，但会在 α的两次出现处分别放置两个索引。直观地说，这表示一对索引可以在以下四类位置中的任意一类应用于同一个值：两个索引都位于当前值处，左侧索引位于子节点处而右侧索引位于当前值处，左侧索引位于当前值处而右侧索引位于子节点处，或者两个索引都位于子节点处。

共享算子满足以下所述的关键性质：

定理 2（共享保持势）。对于 P、$Q \in \mathcal{A}(\tau)$ 和 $v \in \mathcal{V}(\tau)$，$\Phi(v : \langle \tau;\ P\rangle) \cdot \Phi(v : \langle \tau;\ Q\rangle) = \Phi(v : \langle \tau;\ P \curlyvee Q\rangle)$。

C. 与 Hoffmann 等人 *[17]*的比较

从对势函数的这一描述来看，我们所描述的资源多项式究竟是对 Hoffmann 等人 [17], 所述既有多变量 AARA 势的推广，还是仅仅有所不同，并不完全清楚。以下定理表明，前者成立：

定理 3。Hoffmann 等人 *[17]* 可表示的所有资源多项式，在我们的系统中也都可以表示。

证明概要。由于资源多项式是基多项式的线性组合，因此只需证明它们的基多项式可以表示为我们的注释。我们通过类型归纳来证明这一点；唯一非平凡的情况是二叉树。进一步对作为这类二叉树索引的索引列表长度进行归纳后，我们可以考虑该列表的所有可能拆分，通过归纳法获得每种拆分对应的注释，并构造出两侧带有这些注释的节点。

□

该证明可以扩展到 [18] 中的有限元数树。

IV. 语言与类型系统

我们的语言本质上是急切求值的 FPC [22], [20]，通过 $\mathsf{tick}\{q\}$ 表达式来表示代价，并采用显式的 let 构造。tick 表达式是我们语言中唯一的代价来源，但任何基于句法形式的给定成本度量，都可以转换为一种除显式 tick 表达式外不包含其他代价的语言；此外，这类形式为程序员指定特定类型的代价提供了更大的灵活性。

A. 语义

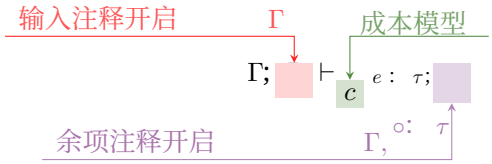
该语言的成本语义是一种标准的小步操作语义。判断 $(e, q) \mapsto (e', q')$ 表示表达式 e 从 $q \geq 0$ 资源开始，经过一步归约转变为表达式 e'，剩余 $q' \geq 0$ 资源。与纯值调用语言相比，唯一新增的归约是针对 tick$\{q\}$ 的归约，如下所示：

$$\frac{q \geq q_0}{(\mathsf{tick}\{q_0\}, q) \mapsto (\mathsf{tt}, q - q_0)}$$

判断 $(e, q) \mapsto^* (e', q')$ 随后定义为单步关系的自反传递闭包，并规定只能考虑非负资源。为了用于无成本推导，我们还定义了"纯"语义$e \to e'$，$\exists q, q'.\, (e, q) \mapsto (e', q')$，并将其自反传递闭包记为 $e \to^* e'$。

B. 类型判断

我们系统中的类型判断如下：



这可以非正式地理解为："在具有 P资源的上下文 Γ 中，根据成本模型 c，e 的类型为 τ，并剩余 Q 资源。"现在我们将详细介绍其中尚未解释的每个构造：上下文注释、余项上下文和成本模型。

1）上下文注释：将 $\mathcal{I}(\Gamma)$ $\prod_{x:\,\tau \in \Gamma} \mathcal{I}(\tau)$、$\mathcal{A}(\Gamma)$ 和 $\mathcal{C}(\Gamma)$ 定义为 $\mathcal{A}(\tau)$ 与$\mathcal{C}(\tau)$ 的相应扩展；注意，$\mathcal{A}(\Gamma) \cong \bigotimes_{x:\,\tau \in \Gamma} \mathcal{A}(\tau)$。对于同构半模之间的强制转换，例如$\mathcal{A}(\Gamma_1, \Gamma_2) \cong \mathcal{A}(\Gamma_1) \otimes \mathcal{A}(\Gamma_2)$ 和 $\mathcal{A}(x:\ \tau) \cong \mathcal{A}(\tau)$，我们可以在记号上省略这些强制转换。其他注释运算（全部都是线性映射）包括：

- "投影算子" $\pi_i :\ \mathcal{A}(\Gamma_1, \Gamma_2) \to \mathcal{A}(\Gamma_1)$，其中 $i \in \mathcal{I}(\Gamma_2)$，它在索引 i 处获取注释的一个切片。$Q = \pi_i(P)$ 定义为 $q_j = p_{(j,i)}$。
- "配对算子" $\mathsf{pair}_y^{x_1\, x_2}$: $\mathcal{A}(\Gamma, x_1:\ \tau_1, x_2 : \tau_2) \to \mathcal{A}(\Gamma, y : \tau_1 \times \tau_2)$，其定义仅通过重新结合实现，因为 $\mathcal{A}(\Gamma, x_1 : \tau_1, x_2 :\ \tau_2) \cong \mathcal{A}(\Gamma) \otimes \mathcal{A}(\tau_1 \times \tau_2) \cong \mathcal{A}(\Gamma, y :\ \tau_1 \times \tau_2)$。
- 将移位算子扩展到上下文注释，$\lhd_x = (\mathsf{id}_{\mathcal{A}(\Gamma)} \otimes \lhd)$: $\mathcal{A}(\Gamma, x :\ \mu\alpha.\, \tau) \to \mathcal{A}(\Gamma, x : [\mu\alpha.\, \tau/\alpha]\tau)$，即对 x 应用移位。
- 将共享算子扩展到上下文注释，${}^{x}\curlyvee_{z}^{y} = (\mathsf{id}_{\mathcal{A}(\Gamma)} \otimes \curlyvee)$: $\mathcal{A}(\Gamma, x : \tau, y :\ \tau) \to \mathcal{A}(\Gamma, z : \tau)$，即变量 x 和 y 一起共享到 z

2）余项上下文：为了更准确地追踪势，我们使用余项上下上的注释。这些注释的灵感来自线性逻辑证明搜索中的 IO 上下文 [23],[24]，但由 Kahn 和 Hoffmann [25] 引入程序化 AARA 设置。余项上下文包含类型上下文 Γ，以及额外的伪变量"$\circ : \tau$"，用于表示结果。余项上下文上的注释随后表示被类型检查的表达式终止后，整个上下文中剩余的势。

在 [25]中指出的余项上下文的许多优点也适用于我们的情形。这些优点包括：函数调用后能够将势返还给参数，以及无需显式共享。

3）成本模型：我们有两种不同的成本模型："付费成本"（记作 cp）和"免费成本"（记作 cf）。前者指实际的、与代价相关的执行（$\mapsto^*$），后者指纯执行（$\to^*$）。虽然我们并不直接关注纯执行，但要跨越抽象边界转换混合势上下文（在我们的情形中，即函数调用），就必须理解纯执行。

C. 类型

我们已在 §III-A 中解释了本语言的大多数类型，但此处将更详细地讨论递归类型和函数类型。

对更一般递归类型的支持，是我们为 AARA 增添的核心创新特性。如第 II-C 节和第 III 节所述，我们在将资源多项式推广到这些类型方面做出了重要贡献。我们选择用"递归"来描述这些类型，因为它们确实是以递归方式构造的。不过，我们想强调，在我们的设定中，递归类型与归纳类型之间的区别并无太大意义：首先，我们的函数内置了一般递归，因此递归类型并不比归纳类型更具表达能力；此外，由于我们对函数值本身只有平凡的（常数）资源多项式，递归类型与归纳类型之间的区别也不会影响资源多项式的范围。

函数类型的形式为 $\langle \tau_1 \to \tau_2, \Theta, \Theta_{\mathsf{cf}} \rangle$，其中$\Theta$，$\Theta_{\mathsf{cf}} \subseteq \mathcal{A}(\tau_1) \times \mathcal{A}(\tau_1 \times \tau_2)$。$\Theta$ 和 Θ_{cf} 是资源规约：它们表示函数参数所需的势，以及在参数和结果上返回的势。为处理资源多态递归，一个函数可以有许多不同的资源规约。直观地说，对于任意规约 $(P, Q) \in \Theta$，要以参数 v 调用该函数，至少需要 $\Phi(v :\ \langle \tau_1;\ P \rangle)$ 资源；当函数返回值 v'时，也会返回 $\Phi(\mathsf{pair}(v;\ v') :\ \langle \tau_1 \times \tau_2;\ Q \rangle)$ 资源。输出势能中提及参数v，是为了支持第 §IV-B2 节讨论的余项上下文。对于第 §IV-B3 节讨论的无成本模型，Θ_{cf} 也起到类似作用：$(P, Q) \in \Theta_{\mathsf{cf}}$ 意味着 $\Phi(\mathsf{pair}(v;\ v') :\ \langle \tau_1 \times \tau_2;\ Q \rangle) \geq \Phi(v :\ \langle \tau_1;\ P \rangle)$，且不涉及执行期间获得或消耗的资源。

D. 类型规则

我们语言的类型规则见图 4。除最后三个规则为结构规则外，所有规则均为语法制导的。所有规则都适用于 let 规范形式的程序，以便更精确地核算势。当然，也可以在类型检查之前添加预处理步骤，从用户的角度隐藏这一限制。

现在概述图 4 中给出的一组具有代表性的有趣类型规则。

$\boxed{\Gamma;\ P \vdash_c e :\ \tau;\ Q}$

T:TICKPOS
$$\frac{q \geq 0 \quad P \otimes \mathcal{C}(\circ : 1) \ = Q + q \cdot \mathcal{C}(\Gamma, \circ : 1)}{\Gamma;\ P \vdash_{\mathsf{cp}} \mathsf{tick}\{q\} :\ 1;\ Q}$$

T:TICKNEG
$$\frac{q \geq 0 \quad P \otimes \mathcal{C}(\circ : 1) + q \cdot \mathcal{C}(\Gamma, \circ : 1) \ = Q}{\Gamma; P \vdash_{\mathsf{cp}} \mathsf{tick}\{-q\} :\ 1;\ Q}$$

T: INR
$$\frac{P = (\mathrm{id}_{\mathcal{A}(\Gamma)} \otimes (\curlyvee \circ (\mathrm{id}_{\mathcal{A}(\tau_r)} \otimes \mathrm{inr}^{-1})))(Q)}{\Gamma, x :\ \tau_r;\ P \vdash_c \mathrm{inr}\ x :\ \tau_l + \tau_r;\ Q}$$

T: TICKFREE
$$\frac{P \otimes \mathcal{C}(\circ :\ 1) \ = Q}{\Gamma; P \vdash_{\mathsf{cf}} \mathsf{tick}\{q\} :\ 1;\ Q}$$

T: 变量
$$\frac{^{\circ}\curlyvee_x^x(Q) = P}{\Gamma,\ x :\ \tau;\ P \vdash_c x :\ \tau;\ Q}$$

T: 单位
$$\frac{P \otimes \mathcal{C}(\circ : 1) \ = Q}{\Gamma;\ P \vdash_c \mathrm{tt} : 1; Q}$$

T: 左注入
$$\frac{P = (\mathrm{id}_{\mathcal{A}(\Gamma)} \otimes (\curlyvee \circ (\mathrm{id}_{\mathcal{A}(\tau_l)} \otimes \text{左注入}^{-1})))(Q)}{\Gamma, x :\ \tau_l;\ P \vdash_c \text{左注入}\, x :\ \tau_l + \tau_r;\ Q}$$

T: 让绑定
$$\frac{\Gamma;\ P \vdash_c e_1 :\ \tau';\ R \qquad (\mathrm{id}_{\mathcal{A}(\Gamma)} \otimes \mathrm{id}_{\mathcal{A}(\tau')})(R) \ = (\mathrm{id}_{\mathcal{A}(\Gamma)} \otimes \mathrm{id}_{\mathcal{A}(\tau')})(S) \qquad \Gamma, x :\ \tau';\ S \vdash_c e_2 :\ \tau;\ Q \otimes \mathcal{C}(x :\ \tau')}{\Gamma;\ P \vdash_c \mathrm{let}(e_1;\ x.e_2) :\ \tau;\ Q}$$

T:案例求和 (x' 新鲜变量)
$$\frac{\begin{array}{c} {}^{x'}\curlyvee_x^x(P') \ = P \qquad {}^{x'}\curlyvee_x^x(Q') = Q \qquad \forall n \in \{\mathrm{l}, \mathrm{r}\}.\ (\\ (\mathrm{id}_{\mathcal{A}(\Gamma)} \otimes \mathrm{in}_n^{-1} \otimes \mathrm{in}_n^{-1})(P') \ = (\mathrm{id}_{\mathcal{A}(\Gamma)} \otimes \mathrm{in}_n^{-1} \otimes \mathrm{id}_{\mathcal{A}(\tau_n)})(R_n) \\ \Gamma, x :\ \tau_l + \tau_r, x_n :\ \tau_n;\ R_n \vdash_c e_n :\ \tau;\ S_n \\ (\mathrm{id}_{\mathcal{A}(\Gamma)} \otimes \mathrm{in}_n^{-1} \otimes \mathrm{id}_{\mathcal{A}(\tau_n)} \otimes \mathrm{id}_{\mathcal{A}(\tau)})(S_n) \ = (\mathrm{id}_{\mathcal{A}(\Gamma)} \otimes \text{在}_n^{-1} \otimes \text{在}_n^{-1} \otimes \mathrm{id}_{\mathcal{A}(\tau)})(R_n)) \end{array}}{\Gamma, x :\ \tau_l + \tau_r;\ P \vdash_c \text{分情况}(x;\ x_l.e_l;\ x_r.e_r) :\ \tau;\ Q}$$

T:乘积
$$\frac{P = (\mathrm{id}_{\mathcal{A}(\Gamma)} \otimes (\text{配对}^{-1} \circ \curlyvee \circ (\text{配对} \otimes \mathrm{id}_{\mathcal{A}(\circ : \tau_1 \times \tau_2)})))(Q)}{\Gamma, x_1 :\ \tau_1, x_2 :\ \tau_2;\ P \vdash_c \text{配对}(x_1;\ x_2) :\ \tau_1 \times \tau_2;\ Q}$$

T:CASE乘积
$$\frac{\begin{array}{c} P = (\mathrm{id}_{\mathcal{A}(\Gamma)} \otimes (\curlyvee \circ (\mathrm{id}_{\mathcal{A}(\tau_1 \times \tau_2)} \otimes \text{配对})))(R) \\ \Gamma, x :\ \tau_1 \times \tau_2, x_1 :\ \tau_1, x_2 :\ \tau_2;\ R \vdash_c e :\ \tau;\ S \\ Q = (\mathrm{id}_{\mathcal{A}(\Gamma)} \otimes (\curlyvee \circ (\mathrm{id}_{\mathcal{A}(\tau_1 \times \tau_2)} \otimes \text{配对})) \otimes \mathrm{id}_{\mathcal{A}(\tau)})(S) \end{array}}{\Gamma, x :\ \tau_1 \times \tau_2;\ P \vdash_c \text{分情况}(x;\ x_1, x_2.e) :\ \tau;\ Q}$$

T:折叠
$$\frac{P = {}^{\circ}\curlyvee_x^x(\triangleleft_{\circ}(Q))}{\Gamma, x :\ [\mu\alpha.\tau/\alpha]\tau;\ P \vdash_c \text{折叠}\ x :\ \mu\alpha.\tau;\ Q}$$

T:展开
$$\frac{P = {}^{\circ}\curlyvee_x^x(\triangleleft_{\circ}^{-1}(Q))}{\Gamma, x :\ \mu\alpha.\tau;\ P \vdash_c \text{展开}\ x :\ [\mu\alpha.\tau/\alpha]\tau;\ Q}$$

T:FUN
$$\frac{\begin{array}{c} \forall (R, S) \in \Theta_{\mathsf{cp}}.\ \Gamma, f :\ \langle \tau_1 \to \tau_2, \Theta_{\mathsf{cp}}, \Theta_{\mathsf{cf}} \rangle, x :\ \tau_1;\ \mathcal{C}(\Gamma) \otimes R \vdash_{\mathsf{cp}} e :\ \tau_2;\ \mathcal{C}(\Gamma) \otimes S \\ \forall (R,\ S) \in \Theta_{\mathsf{cf}}.\ \Gamma,\ f :\ \langle \tau_1 \to \tau_2,\ \Theta_{\mathsf{cp}},\ \Theta_{\mathsf{cf}} \rangle,\ x :\ \tau_1;\ \mathcal{C}(\Gamma) \otimes R \vdash_{\mathsf{cf}} e :\ \tau_2;\ \mathcal{C}(\Gamma) \otimes S \end{array}}{\Gamma;\ P \vdash_c \text{函数}(f,\ x.e) :\ \langle \tau_1 \to \tau_2,\ \Theta_{\mathsf{cp}},\ \Theta_{\mathsf{cf}} \rangle;\ Q}$$

T:WEAKEN
$$\frac{\Gamma;\ R \vdash_c e :\ \tau;\ S \qquad P = R \otimes \mathcal{C}(x :\ \sigma) \qquad Q = S \otimes \mathcal{C}(x :\ \sigma)}{\Gamma、\ x :\ \sigma;\ P \vdash_c e :\ \tau;\ Q}$$

T:APP
$$\frac{\begin{array}{c} \forall i \in \mathcal{C}(\Gamma)_{\circ}\ (\pi_{\Gamma \mapsto i}(P)\ \ \pi_{\Gamma \mapsto i}(Q)) \ \in \Theta_{\mathsf{cp}} \\ \forall i \notin \mathcal{C}(\Gamma)_{\circ}\ (\pi_{\Gamma \mapsto i}(P)\ \ \pi_{\Gamma \mapsto i}(Q)) \ \in \Theta_{\mathsf{cf}} \end{array}}{\Gamma、\ f :\ \langle \tau_1 \to \tau_2、\ \Theta_{\mathsf{cp}}、\ \Theta_{\mathsf{cf}} \rangle、\ x :\ \tau_1;\ P \vdash_c \mathrm{app}(f;\ x) :\ \tau_2;\ Q}$$

T:放松
$$\frac{\Gamma;\ R \vdash_c e :\ \tau';\ S \qquad R \leq P \qquad Q \leq S}{\Gamma;\ P \vdash_c e :\ \tau;\ Q}$$

T:增补
$$\frac{\Gamma;\ R \vdash_c e :\ \tau;\ S \qquad P = R + T \qquad Q = S + T \otimes \mathcal{C}(\tau)}{\Gamma;\ P \vdash_c e :\ \tau;\ Q}$$

图 4：资源类型推导规则。关于所选规则的解释，请参见 §IV-D。

T：让绑定：与以往的研究相比，这条规则之所以尤其值得关注，主要是因为它非常简单。在此前的多变量 AARA 工作中，为了处理结果与上下文中其他变量之间的混合势，这条规则必须考虑所绑定项的免费成本推导 [17]。然而，我们的余项上下文消除了这一要求，因为余项注释可以直接提及这些混合势。这一改进并非免费成本；相反，函数需要免费成本推导，但我们认为这构成了更恰当的抽象边界。

T:PAIR：一些规则使用了看起来复杂的张量构造，但实际上只是繁琐记账的受害者；此规则便是其中之一。换言之，该条件表示，Q其实就是 P，只是将 x_1 和 x_2 共享后，再打包到 $\circ$中。

T:FOLD 和 T:UNFOLD：这些规则是基于势注释的类型系统的核心，利用了 §III-B 中加性移位算子的定义。

T:APP：此函数应用规则实质上要求参数具有足够的势，才能使用函数的已付成本注释该函数随后会被返回到余项注释中，然后所有混合势项都必须根据该函数的免费成本注释进行转换。

E. 类型检查示例

现在给出一个使用该类型系统可以进行的新颖类型检查示例。我们使用一种表层语言来呈现代码，并加入了一些轻量语法糖，为递归类型的构造子提供名称。这里的成本模型仍然是所创建的 cons 单元数量。

设想实现一个模拟文件系统。每个节点要么是文件，由其名称和非结构化内容组成；要么是目录，由其名称以及其中包含的节点列表组成：

```
类型 文件系统 = 文件由 字符串 * 字符串| 目录由
字符串 * 文件系统 列表
```

函数 attach 接收一个名称和一个文件系统，并返回一个配对列表，其中第一个元素是给定的名称，第二个元素是文件系统中某个目录或文件的名称。

令递归函数 attach dname (acc, fs) = 匹配 fs | 文件 (fname, _) → [(dname, fname)] | 目录 (subdname, fss) →(dname, subdname) :: 左折叠函数(attach dname) (acc, fss)

（这里 foldl: ('b × 'a → 'b) → 'b ×list('a) → 'b 是列表上的左折叠函数。）不考虑资源使用时，可以为该函数赋予类型 attach: $\mathbb{S} \to \text{list}(\mathbb{S} \times \mathbb{S}) \times \text{filesystem} \to \text{list}(\mathbb{S} \times \mathbb{S})$，其中 $\mathbb{S}$ 是字符串类型的缩写。该函数创建的 cons 单元数量等于文件系统中的节点数。因此，我们可以为 attach 赋予一种类型，该类型要求其第一个参数不具有势，

$$1 \cdot ([], \text{File}(\star, \star)) + 1 \cdot ([], \text{Dir}(\star, [\text{end}]))$$

其第二个参数具有势（其中 $\star$ 表示字符串上的常量势），并且其输出不返回势。

基于 attach，第二个示例函数 trans 接受文件系统的表示，并计算出一个包含每个配对 (d, s) 的列表，使得 d 是名为 s 的文件或子目录的祖先目录名称。[3]

递归定义 trans (acc, fs) = 匹配 fs 其中| File (_, _) → [] | Dir(dname, fss) →左折叠函数trans ((左折叠函数 (attach dname) (acc, fss)), fss)

不考虑资源使用时，可以为该函数指定类型 trans : list($\mathbb{S} \times \mathbb{S}$) ×filesystem → list($\mathbb{S} \times \mathbb{S}$)。这会创建数量与文件系统中节点数的平方成正比的 cons 单元。使用我们的类型系统，可以说 trans 的参数需要具有以下势：

$$1 \cdot ([], \text{Dir}(\star, [\text{File}(\star, \star)])) + 1 \cdot ([], \text{Dir}(\star, [\text{Dir}(\star, [\text{end}])]))$$

而其输出不返回任何势。参数注释中两层嵌套出现的 Dir 构造器表明了二次势使用。

F. 健全性

此处的健全性意味着，在闭上下文中，若资源充足，则良类型项可以安全执行：

定理 4（健全性）。假设 $\cdot;\ p \vdash_{\mathsf{cp}} e : \tau;\ Q$。对于任意 $r \geq 0$ 以及执行 $(e, p + r) \mapsto^* (e', q)$，我们有以下情况之一：

- $e' \in \mathcal{V}(\tau)$ *and* $q \geq r + \Phi(e' : \langle \tau; Q \rangle)$, *or*
- *there exists some* e'' *and* q' *such that* $(e', q) \mapsto (e'', q')$.

证明。我们构造一个步索引逻辑关系；唯一的非标准情况是箭头类型：从下一步开始，类型的所有注释都必须在具有足够资源的所有状态中得到满足。完整定义见附录 C。随后，我们通过对类型推导进行归纳，证明逻辑关系基本定理，并证明逻辑关系中的可居住性蕴含该定理的结论。

□

[3]此例改编自 Hoffmann [26] 给出的示例。

V. 相关工作

我们的工作建立在自动摊销资源分析（AARA）类型系统的相关研究之上。AARA 最初由 [15], 使用势能法 [21] 推理引入，用于自动推导与数据结构大小呈线性关系的堆空间界。AARA 后来得到扩展，以支持以下形式的界：多项式 [19], 指数函数 [27],对数函数 [28], 以及最大值 [25], [29]。然而，这些工作都将其大小参数和资源函数内置于树和列表（或带标签树 [18]）的归纳数据类型中。唯一的例外是 Schopenhauer 语言[16], , 它支持推导使用嵌套递归类型的程序的界，但仅限于非常受限的线性函数类。我们的索引为一般递归类型的资源函数和大小参数的自动构造提供了第一种方法。由此得到的系统保守地扩展了多变量多项式系统 [17], [18], 给出的界，其中界可能取决于数据结构大小的乘积。尽管我们的工作没有沿这些方向展开，AARA 的分析还可以涵盖其他计算模型和分析，包括命令式编程 [30], 面向对象 [31], [32], 概率式 [33], [34], 以及并行计算 [35], 、数字合约协议[36], 和下界成本 [37]。组合物种 [38]也可能与 AARA 的资源多项式相关。

除 AARA 外，其他基于类型的系统也被用于分析程序的资源使用。这些系统包括线性依赖类型 [2], [1], 精化类型 [14], [10], 模态类型 [3], 大小类型 [39], [40], 基于注解的系统 [41], [42], 以及其他系统。这些类型系统使用各种不同的思想来记录代价，但它们都享有类型系统所提供的高度可组合性，通常也像 AARA 一样采用某些线性特性和成本约束。然而，与 AARA 不同的是，许多系统会以一定程度的自动化能力为代价换取更丰富的特性——在与 AARA 相反的极端方向上，可以找到 [43], 中的基于类型的证明逻辑；这类逻辑需要用户投入大量工作来证明成本界。

项重写领域提供了一些与我们工作相当的研究。具体而言， [44] 中的系统使用树自动机，将多变量势函数推广到任意类型。我们推测，该系统足够通用，可以包含由我们的方法生成的资源函数。然而，他们的工作留下了如何选择适当的自动机，以及如何求解由此产生的约束这两个开放问题。已有大量工作 [45] 甚至在多变量情形之外求解更简单的情形。我们的工作可以被视为朝向这一自动化方向迈出的一步。其他使用项重写进行资源分析的工作包括 [46], [8], [47], [48]。

递推关系是资源分析的另一种常见方法，尤其适用于函数式环境 [11], [49], [5]。一些近期研究采用基于势能的推理来进行摊还分析 [6]。通常，这些方法首先从代码中提取递推关系，然后求解这些递推关系。虽然这可能比求解通过 AARA 提取的线性约束更加困难，但它能够对以下界限的表达式进行

函数，而 AARA 目前尚无法支持这些函数。

命令式编程 [7], [50] 和逻辑 [51] 程序分析中的技术也可以根据数据结构大小的一般概念来推理代价。然而，这些工作通过手动定义的大小概念进行分析，并将其归约为数值分析。虽然这种数值分析在成本分析中很常见，但它并不关注我们工作所涉及的数据结构的那些内在特征。

成本分析的其他方法包括抽象解释、 [4], [52], [53], 循环分析、 [54], [55], 关系成本分析、 [56], [57], 秩函数 [9], 和程序逻辑[58], [59], [60]。该领域涵盖范围广泛，融合了许多方法。特别是，所引用的程序逻辑采用了类似 AARA 的基于势能的推理。

VI. 结论与未来工作

本工作的贡献包括将多变量资源多项式扩展到正则递归数据类型，并引入半模作为系统规范中有用的形式化工具。扩展后的资源多项式使得对使用复杂嵌套数据结构（如玫瑰树）的程序进行资源分析成为可能。这些成果通过数据类型的结构组合学，向可自动化的资源分析迈出了重要一步。未来的工作将包括在完全自动化的资源分析类型检查器（如 RaML）中实现这些资源函数 [18]。此外，我们希望扩展生成资源多项式的方法，使其也涵盖资源指数 [27],，而目前多变量层面尚不支持资源指数。

致谢

本文基于以下研究成果：该研究由 Algorand 基金会管理的 Algorand 卓越中心计划，以及美国国家科学基金会通过项目编号 1801369、1845514 和 2007784 提供支持。本文所载的任何意见、发现和结论均为作者观点，不一定反映资助机构的立场。

参考文献

[1] U. Dal Lago 和 M. Gaboardi，“线性依赖类型与相对完备性”，载于《2011 IEEE 第 26 届计算机科学逻辑年度研讨会论文集》。IEEE，2011 年，第 133–142 页。[2] U. Dal Lago 和 B. Petit，“类型几何”，《ACM SIGPLAN Notices》，第 48 卷，第 1 期，第 167–178 页，2013 年。[3] V. Rajani、M. Gaboardi、D. Garg 和 J. Hoffmann，“高阶（摊销）成本分析的统一类型理论”，《ACM 编程语言论文集》，第 5 卷，POPL 期，第 1–28 页，2021 年。[4] E. Albert 、 P. Arenas 、 S. Genaim 、 G. Puebla 和 D. Zanardini，“Costa：面向 Java 字节码的成本与终止性分析器的设计与实现”，载于《组件与对象形式化方法国际研讨会论文集》。Springer，2007 年，第 113–132 页。[5] G. Kavvos、E. Morehouse、D. R. Licata 和 N. Danner，“通过调用按推值为函数式程序进行递推提取”，《ACM 编程语言论文集》，第 4 卷，POPL 期，第 1–31 页，2019 年。[6] J. W. Cutler、D. R. Licata 和 N. Danner，“用于摊销分析的指称递推提取”，《ACM 编程语言论文集》，第 4 卷，ICFP 期，第 1–29 页，2020 年。[7] S. Gulwani、K. K. Mehra 和 T. Chilimbi，“Speed：精确而高效的程序计算复杂度静态估算”，《ACM SIGPLAN Notices》，第 44 卷，第 1 期，第 127–139 页，2009 年。

[8] M. Avanzini、U. Dal Lago 和 G. Moser，《分析函数式程序的复杂度：高阶与一阶的结合》，载于 ACM SIGPLAN 函数式编程国际会议第 20 届会议论文集，2015 年，第 152–164 页。[9] K. Chatterjee、H. Fu 和 A. K. Goharshady，《递归程序的非多项式最坏情况分析》，《ACM 程序设计语言汇刊》（TOPLAS），第 41 卷第 4 期，2019 年，第 1–52 页。[10] P. Wang、D. Wang 和 A. Chlipala，《Timl：一种利用不变量进行实用复杂度分析的函数式语言》，《ACM 编程语言论文集》，第 1 卷，OOPSLA 专刊，2017 年，第 1–26 页。[11] Z. Kincaid、J. Breck、A. F. Boroujeni 和 T. Reps，《重新审视组合递推分析》，《ACM SIGPLAN Notices》，第 52 卷第 6 期，2017 年，第 248–262 页。[12] E. Albert 、 S. Genaim 和 A. N. Masud，《资源使用上下界的推断》，《ACM 计算逻辑汇刊》，第 14 卷第 3 期，2013 年，第 22:1–22:35 页。 [在线]。可访问：https://doi.org/ 10 1145 2499937 2499943 。/。V. C. Ngo、[13] M. Dehesa-Azuara、M. Fredrikson 和 J. Hoffmann，《使用类型验证和综合常量资源实现》，载于第 38 届 IEEE 安全与隐私研讨会（S&P ’17），2017 年。[14] I. Radi˘cek、G. Barthe、M. Gaboardi、D. Garg 和 F. Zuleger，《用于关系成本分析的单子精化》，《ACM 编程语言论文集》，第 2 卷，POPL 专刊，2017 年，第 1–32 页。[15] M. Hofmann 和 S. Jost，《一阶函数式程序堆空间使用量的静态预测》，《ACM SIGPLAN Notices》，第 38 卷第 1 期，2003 年，第 185–197 页。[16] S. Jost、K. Hammond、H.-W. Loidl 和 M. Hofmann，《高阶程序定量资源使用量的静态确定》，载于第 37 届 ACM SIGPLAN-SIGACT 编程语言原理研讨会年度会议论文集，2010 年，第 223–236 页。[17] J. Hoffmann、K. Aehlig 和 M. Hofmann，《多变量摊销资源分析》，载于第 38 届 ACM SIGPLAN-SIGACT 编程语言原理研讨会年度会议论文集，2011 年，第 357–370 页。[18] J. Hoffmann、A. Das 和 S.-C. Weng，《迈向 OCaml 的自动资源界分析》，载于第 44 届 ACM SIGPLAN 编程语言原理研讨会，2017 年，第 359–373 页。[19] J. Hoffmann 和 M. Hofmann，《具有多项式势的摊销资源分析》，载于《欧洲编程研讨会》。Springer，2010 年，第 287–306 页。[20] M. P. Fiore 和 G. D. Plotkin，《FPC 的计算充分域理论模型的公理化》，载于第九届逻辑与计算机科学年度研讨会（LICS ’94），法国巴黎，1994 年 7 月 4–7 日。IEEE 计算机学会，1994 年，第 92–102 页。 [ ] 10 1109 1994 316083 在线。可访[21] 问：https://doi.org/。/LICS。R. E. Tarjan，《摊销计算复杂度》，《SIAM 代数与离散方法杂志》，第 6 卷第 2 期，1985 年，第 306–318 页。[22] R. Harper，《编程语言实用基础》（第 2 版）。剑桥大学出版社，2016 年。 [在线]。可访问：https://www.cs.cmu.edu/%7Erwh/pfpl/index.html [23] I. Cervesato、J. S. Hodas 和 F. Pfenning，《线性逻辑证明搜索的高效资源管理》，《理论计算机科学》，第 232 卷第 1–2 期，2000 年，第 133–163 页。[24] J. S. Hodas 和 D. Miller，《直觉主义线性逻辑片段中的逻辑编程》，《信息与计算》，第 110 卷第 2 期，1994 年，第 327–365 页。[25] D. M. Kahn 和 J. Hoffmann，《使用量子物理学家方法的自动摊还资源分析》，《ACM 编程语言论文集》，第 5 卷，ICFP 专刊，2021 年，第 1–29 页。[26] J. Hoffmann，《带势的类型：通过自动摊还分析获得多项式资源界》，博士学位论文，慕尼黑大学，2011 年。 [在线]。可访问：https://edoc.ub.uni-muenchen.de/id/eprint/ 15935 [27] D. M. Kahn 和 J. Hoffmann，《指数型自动摊还资源分析》，载于《软件科学与计算结构基础国际会议》。Springer，Cham，2020 年，第 359–380 页。[28] M. Hofmann、L. Leutgeb、D. Obwaller、G. Moser 和 F. Zuleger，《基于类型的对数摊销复杂度分析》，《计算机科学中的数学结构》，2021 年，第 1–33 页。[29] B. Campbell，《利用数据结构深度进行摊销内存分析》，载于《欧洲编程研讨会》。Springer，2009 年，第 190–204 页。

[30] Q. Carbonneaux、J. Hoffmann 和 Z. Shao，《组合式认证资源界》，载于 ACM SIGPLAN 编程语言设计与实现会议第 36 届会议论文集，2015 年，第 467–478 页。[31] M. Hofmann 和 S. Jost，《基于类型的摊销堆空间分析》，载于《欧洲编程研讨会》。Springer，2006 年，第 22–37 页。[32] M. Hofmann 和 D. Rodriguez，《摊销堆空间分析的自动类型推断》，载于《欧洲编程研讨会》。Springer，2013 年，第 593–613 页。[33] V. C. Ngo、Q. Carbonneaux 和 J. Hoffmann，《有界期望：概率程序的资源分析》，《ACM SIGPLAN Notices》，第 53 卷第 4 期，2018 年，第 496–512 页。[34] D. Wang、D. M. Kahn 和 J. Hoffmann，《提高期望：利用类型自动化期望成本分析》，《ACM 编程语言论文集》，第 4 卷，ICFP 专刊，2020 年，第 1–31 页。[35] J. Hoffmann 和 Z. Shao，《并行程序的自动静态成本分析》，载于《欧洲编程语言与系统研讨会》。Springer，2015 年，第 132–157 页。[36] A. Das、S. Balzer、J. Hoffmann、F. Pfenning 和 I. Santurkar，《数字合同的资源感知会话类型》，载于 2021 年 IEEE 第 34 届计算机安全基础研讨会（CSF）。IEEE，2021 年，第 1–16 页。[37] M. Dehesa-Azuara、M. Fredrikson、J. Hoffmann 等，《使用类型验证和综合常量资源实现》，载于 2017 年 IEEE 安全与隐私研讨会（SP）。IEEE，2017 年，第 710–728 页。[38] F. Bergeron、G. Labelle 和 P. Leroux，《组合物种与树状结构》，《数学及其应用百科全书》丛书。剑桥大学出版社，1997 年。[在线]。可访问：https://www.cambridge.org/core/books/combinatorial-species-and-treelike-structures/D9941...

[53] P. Lopez-G[39] P. B. Vasconcelos，《使用尺寸类型进行空间代价分析》，博士学位论文，圣安德鲁斯大学，2008 年。[40] A. Serrano、P. López-García 和 M. V. Hermenegildo，《通过使用尺寸类型的抽象解释分析逻辑程序的资源使用》，《逻辑编程理论与实践》，第 14 卷，第 4–5 期，第 739–754 页，2014 年。[41] K. Crary 和 S. Weirich，《资源界认证》，载于第 27 届 ACM SIGPLAN-SIGACT 编程语言原理研讨会论文集，2000 年，第 184–198 页。[42] N. A. Danielsson，《纯函数式数据结构的轻量级半形式化时间复杂度分析》，ACM SIGPLAN Notices，第 43 卷，第 1 期，第 133–144 页，2008 年。[43] Y. Niu、J. Sterling、H. Grodin 和 R. Harper，《成本感知逻辑框架》，载于 ACM 编程语言论文集，第 6 卷，POPL，第 1–31 页，2022 年。[44] M. Hofmann 和 G. Moser，《项重写系统的多变量摊销资源分析》，载于第 13 届类型化兰姆达演算与应用国际会议（TLCA 2015）。Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik，2015 年。[45] S. Bauer，《面向对象程序资源分析的线性树约束可判定性》，博士学位论文，慕尼黑大学，2019 年。[46] M. Avanzini 和 G. Moser，《复杂度组合框架》，《信息与计算》，第 248 卷，第 22–55 页，2016 年。[47] N. Hirokawa 和 G. Moser，《基于上下文敏感重写的自动化复杂度分析》，载于《重写与类型化兰姆达演算》。Springer，2014 年，第 257–271 页。[48] M. Naaf、F. Frohn、M. Brockschmidt、C. Fuhs 和 J. Giesl，《基于整数转移系统的项重写复杂度分析》，载于组合系统前沿国际研讨会。Springer，2017 年，第 132–150 页。[49] N. Danner、D. R. Licata 和 R. Ramyaa，《含归纳类型函数式语言的指称成本语义》，载于第 20 届 ACM SIGPLAN 函数式编程国际会议论文集，2015 年，第 140–151 页。[50] S. Gulwani，《Speed：符号复杂度界分析》，载于计算机辅助验证国际会议。Springer，2009 年，第 51–62 页。[51] J. Navas、E. Mera、P. López-García 和 M. V. Hermenegildo，《逻辑程序的用户可定义资源界分析》，载于逻辑编程国际会议。Springer，2007 年，第 348–363 页。[52] E. Albert、J. C. Fernández 和 G. Román-Díez，《非累积资源分析》，载于系统构造与分析工具和算法国际会议。Springer，2015 年，第 85–100 页。

M. V. Her
翻译
理论与
2018 年第 496
程序编程
第 105–11 页。
[53] Z. Kincaid,
[在线]。可访问：
语言，
[58] P. Çiçek, G
代价分析
2017。[57] W. Qu、M.
functional-i
第 21 卷，20
《逻辑编程
浙近
欧洲
编程研讨会
[60] G. Mével, J
3–29。

arcia、L. Darmawan、M. Klemen、U. Liqat、F. Bueno 和 menegildo，“基于区间的资源使用验证：通过为 Horn子句及其在能耗中的应用，”《逻辑编程实践》，第18卷，第2期，第167–223页，

A. Henzinger, T. Hottelier 和 L. Kovács，“Abc：代数算循环”，载于《逻辑国际会议》ng 人工智能与推理。Springer，2010年，。

J. Cyphert、J. Breck 和 T. Reps，“非线性推理”，ACM《程序设计学报》会议论文集，第 2 卷，第 POPL 期，第 1–33 页，2017 年。

. Barthe、M. Gaboardi、D. Garg 和 J. Hoffmann，“关系”，ACM SIGPLAN Notices，第 52 卷，第 1 期，第 316–329 页，

Gaboardi 和 D. Garg，《关系成本分析》，《函数式编程杂志》，21.

、A. Charguéraud 和 F. Pottier，《一把美元：形式化复杂度主张的演绎式程序验证》，程序设计研讨会。Springer，2018年，第533–

分离逻辑中的摊销资源分析”，载于欧洲。Springer，2010年，第85–103页。

.-H. Jourdan 和 F. Pottier，“时间信用与时间收据欧洲程序设计研讨会。Springer，2019 年，第

2877BDE63FF0C9EDE2F9788A8

附录 A 半模概览

本节简要概述半模，以及本文所涉及的半模上的运算和性质。

R-半模 M 是向量空间的一种推广，它在半环 R 上运作，而不是在域上运作。在本文中，我们将假设 R 是交换的，以避免到处重复使用“交换的”一词。与向量空间一样，其中存在加法运算 $+\colon\ M \times M \to M$，该运算构成一个交换幺半群，以及一个标量乘法运算 $\cdot\colon\ R \times M \to M$，该运算满足各种分配律和单位律。$R$-半模最简单的例子就是$R$ 本身，其半模运算继承自环结构。

R-半模之间的线性映射（也称为同态） M和 M'是从 M到 M'的函数，且保持加法和标量乘法不变。半模之间的同构是它们之间的双射线性映射。双线性映射 $M \times M' \to M''$是一个对两个参数都具有线性的函数。

自由的 R-半模是指在半模运算下由一个基 E 有限生成的半模。等价地，由 E 生成的自由 R-半模是从 E 到 R中非零元素的有限映射集合，其加法和标量乘法按逐点方式定义。（与向量空间不同，并非每个半模都同构于自由半模，但这与本文无关。）

若 f、$g\colon\ M \to M'$ 是线性映射，其中 M 是由 E生成的自由R-半模，则 $f = g$ 当且仅当对于所有 $e \in E$，$f(e) = g(e)$；这可以利用线性映射的性质，通过对 M-元素的有限生成进行归纳而得到。

两个 R-半模 M_1 和 M_2 的张量积是半模 $M_1 \otimes M_2$，带有典范线性映射 $\otimes\colon\ M_1 \times M_2 \mapsto m_1 \otimes m_2$，满足如下泛性质：所有双线性映射 $f\colon\ M_1 \times M_2 \to M'$ 都存在唯一的线性映射 $\tilde{f}\colon\ M_1 \otimes M_2 \to M'$，使得 $f = \tilde{f} \circ \otimes$。（其存在性可以通过商构造证明。）张量积具有对称幺半结构，因此 $M_1 \otimes M_2 \cong M_2 \otimes M_1$ 且$M_1 \otimes (M_2 \otimes M_3) \cong (M_1 \otimes M_2) \otimes M_3$。对于函数 $f\colon\ M_1 \to M_1'$和 $g\colon\ M_2 \to M_2'$，可以定义函数上的张量运算$f \otimes g$，使得对于任意 $m_1 \in M_1$ 和 $m_2 \in M_2$，$(f \otimes g)(m_1 \otimes m_2) = f(m_1) \otimes g(m_2)$。按照约定，对于函数$f\colon\ M_1 \to M_1'$，记号 $f \otimes M_2$ 表示映射 $f \otimes g\colon\ M_1 \otimes M_2 \to M_1' \otimes M_2$，其中 g是 M_2上的恒等映射。

附录 B 共享算子

参见图 5。

附录 C 可靠性细节

为陈述我们的语义良类型性概念，我们首先在图 6 中定义表达式上的最弱前置条件一元关系 $\mathcal{WP}$，以及值上的类型索引的一元逻辑关系 $\mathcal{V}$ 二者都由成本模型 c 进行索引，具体说明见 §IV-B3。

我们的最弱前置条件关系 $\mathcal{WP}$ 由后置条件 Ψ 参数化，并接受表达式 e 和初始资源 q 作为参数。直观地说，如果 e 从 q 资源开始执行，在少于 n 步内“不会出错”，且对于任何由此产生的值 v、剩余步数 m 和剩余资源 q'，$\Psi(m, v, q')$ 均成立，那么 $\mathcal{WP}_c[\Psi]_n(e, q)$ 就成立。

根据应用规则的需要，联系付费成本和免费成本最弱前置条件的核心定理指出，我们可以将二者合并为一个付费成本最弱前置条件；形式化地说：

引理 2（最弱前置条件合并）。如果 $\mathcal{WP}_{\mathsf{cp}}[\Psi_p](e, q_p)$ 和 $\mathcal{WP}_{\mathsf{cf}}[\Psi_f](e, q_f)$ 均成立，那么 $\mathcal{WP}_{\mathsf{cp}}[\lambda\ v\ q'\ .\ \exists q_p'\ q_f'.\, q' = q_p' + q_f' \wedge \Psi_p\ v\ q_p'\ \wedge \Psi_f\ v\ q_f'\ ]\ e\ q_p + q_f$ 也成立。

证明。通过对步骤进行归纳，并使用→的定义。 □

除函数类型外，所有类型的值关系都十分标准，因此我们只详细说明箭头类型。如 §IV-C 所述，箭头类型的形式为 $\langle \tau_1 \to \tau_2, \Theta, \Theta_{\mathsf{cf}} \rangle$ where $\Theta, \Theta_{\mathsf{cf}} \subseteq \mathcal{A}(\tau_1) \times \mathcal{A}(\tau_1 \times \tau_2)$，它们是一些规范的集合，用于描述函数所需和返回的势。因此，我们的解释直观地表明，对于每个注释规范 $(P, Q) \in \Theta$（分别为 Θ_{cf}），对于所有资源状态 q 和参数 v'，假设 v' 在语义上具有类型 τ_1，则使用 q 资源以及 P 要求由 v提供的额外势来运行该函数，其结果（若终止）在语义上具有类型 τ_2，且剩余资源至少为 q 加上Q 所保证的资源。这里，对所有更小的步数索引 m 和资源 q 的量化，起到了对未来状态进行 Kripke 式量化的作用。

有了这些逻辑关系，我们定义语义资源类型判断：

定义 3（语义资源类型）。

$$\begin{aligned}&\Gamma; P \vDash_c e : \tau; Q \triangleq\\ &\quad \forall \gamma, n, q \geq 0.\ \mathcal{V}[\![\Gamma]\!]_n(\gamma) \Rightarrow\\ &\qquad \mathcal{WP}_c[\lambda(m, v, q').\ \mathcal{V}[\![\tau]\!]_m(v) \wedge\\ &\qquad\qquad q' \geq q + \Phi(\gamma[\circ \mapsto v] : \langle \Gamma, \circ : \tau; Q \rangle)]\\ &\qquad (e, q + \Phi(\gamma : \langle \Gamma; P \rangle))\end{aligned}$$

也就是说，在成本模型 c 和带注释上下文 $\langle \Gamma$ 下；$P\rangle$，表达式 e 在 τ 处具有语义良好类型，并带有余项注释 Q，当且仅当对于所有语义上具有良好类型的闭替换上下文 γ，如果终止运行，则 e 永不崩溃，并产生一个语义上具有良好类型的余项值上下文，其势至少与根据 Q 所规定的一样多；前提是运行开始时拥有至少与 P 所说明的 γ 应有资源一样多的资源。

此时，我们希望证明逻辑关系基本定理：

引理 3（基本定理）。如果 $\Gamma;\ P \vDash_c e\colon\ \tau;\ Q$，那么$\Gamma;\ P \vDash_c e\colon\ \tau;\ Q$。

$\mathcal{N}\{\alpha[\tau].\sigma\}(i_1,i_2) \in \mathcal{A}([\mu\alpha.\,\tau/\alpha]\sigma)\mathcal{N}\{\alpha[\tau].\mathbf{1}\}(i,j) = 0\mathcal{N}\{\alpha[\tau].\top_\mu\}(i,j) = 0\mathcal{N}\{\alpha[\tau].\sigma+\rho\}(i,j) =$ 左注入 $(\mathcal{N}\{\alpha[\tau].\sigma\}(i,j)) +$右注入 $(\mathcal{N}\{\alpha[\tau].\rho\}(i,j))\mathcal{N}\{\alpha[\tau].\sigma\times\rho\}(i,j) =$配对$(\mathcal{N}\{\alpha[\tau].\sigma\}(i,j);\ \mathcal{C}(\rho)) +$ 配对$(\mathcal{C}(\sigma);\ \mathcal{N}\{\alpha[\tau].\rho\}(i,j))+$ 配对$(\mathcal{M}\{\alpha.\sigma\}(i);\ \mathcal{M}\{\alpha.\rho\}(j)) +$配对$(\mathcal{M}\{\alpha.\sigma\}(j);\ \mathcal{M}\{\alpha.\rho\}(i))$ $\mathcal{N}\{\alpha[\tau].\mu\beta.\,\sigma\}(i,j) =$折叠 $(\mathcal{N}\{\alpha[\tau].[\top_\mu/\beta]\sigma\}(i,j))+$折叠 $(\mathcal{M}\{\beta.[\mu\alpha.\,\tau/\alpha]\sigma\}(\mathcal{M}\{\alpha.[\top_\mu/\beta]\sigma\}(i))$ $\curlyvee_{[\mu\tau.\,\alpha/\alpha][\mu\beta.\,\sigma/\beta]\sigma}\mathcal{M}\{\alpha.[\top_\mu/\beta]\sigma\}(j))+$折叠 $(\mathcal{M}\{\alpha.[\top_\mu/\beta]\sigma\}(i)\ \curlyvee_{[\mu\tau.\,\alpha/\alpha][\mu\beta.\,\sigma/\beta]\sigma}\mathcal{M}\{\beta.[\mu\alpha.\,\tau/\alpha]\sigma\}($ $\mathcal{M}\{\alpha.[\top_\mu/\beta]\sigma\}(j)))+$折叠 $(\mathcal{N}\{\beta[\sigma].[\mu\alpha.\,\tau/\alpha]\sigma\}(\mathcal{M}\{\alpha.[\top_\mu/\beta]\sigma\}(i)\ \mathcal{M}\{\alpha.[\top_\mu/\beta]\sigma\}(j)))$

$$
\begin{aligned}
i_1 \curlyvee_\tau i_2 &\in \mathcal{A}(\tau)\\
\mathsf{tt} \curlyvee_{\mathbf{1}} \mathsf{tt} &= \mathsf{tt}\\
(\mathsf{inl}\, i) \curlyvee_{\tau+\sigma} (\mathsf{inl}\, j) &= \mathsf{inl}\, i \curlyvee_\tau j\\
(\mathsf{inr}\, i) \curlyvee_{\tau+\sigma} (\mathsf{inr}\, j) &= \mathsf{inr}\, i \curlyvee_\sigma j\\
\mathsf{pair}(i_1;\, i_2) \curlyvee_{\tau\times\sigma} \mathsf{pair}(j_1;\, j_2) &= \mathsf{pair}(i_1 \curlyvee_\tau j_1;\, i_2 \curlyvee_\sigma j_2)\\
\mathsf{end} \curlyvee_{\mu\alpha.\,\tau} \mathsf{end} &= \mathsf{end}\\
\mathsf{end} \curlyvee_{\top_\mu} \mathsf{end} &= \mathsf{end}\\
(\mathsf{fold}\, i) \curlyvee_{\mu\alpha.\,\tau} \mathsf{end} &= \mathsf{fold}\, i\\
\mathsf{end} \curlyvee_{\mu\alpha.\,\tau} (\mathsf{fold}\, i) &= \mathsf{fold}\, i\\
(\mathsf{fold}\, i) \curlyvee_{\mu\alpha.\,\tau} (\mathsf{fold}\, j) &= \mathsf{fold}\, (i \curlyvee_{[\mu\alpha.\,\tau/\alpha]\tau} j)\\
&\quad + \mathsf{fold}\, (\mathcal{M}\{\alpha.\tau\}(i) \curlyvee_{[\mu\alpha.\,\tau/\alpha]\tau} j)\\
&\quad + \mathsf{fold}\, (i \curlyvee_{[\mu\alpha.\,\tau/\alpha]\tau} \mathcal{M}\{\alpha.\tau\}(j))\\
&\quad + \mathsf{fold}\, (\mathcal{N}\{\alpha[\tau].\tau\}(i,j))\\
i \curlyvee_\tau j &= 0 \qquad \text{otherwise}
\end{aligned}
$$

图 5：共享算子的定义

证明概要。鉴于我们的设置方式，对于几乎所有规则而言，这实际上都很容易证明，因为该陈述归结为势评估相等，而利用 §III 和 §IV-B1 中的定理即可轻松证明这一点。对于不满足这一条件的规则，我们给出证明概要：

- T：让绑定。对 e_1 的执行推导进行归纳，并对 let 使用求值上下文步进规则。
- T：展开。势相等性立即成立；唯一的类型检查问题来自 μ类型中的"later"，但通过将展开（折叠 v）转换为 v步即可消除这一问题。
- T:App。首先选择要在 Θ 中使用的一个已付成本注释，然后对具有非零系数的非恒定索引集合进行归纳，使用引理 2 为函数应用创建一个大的最弱前置条件。
- T：放松。由遵循注释顺序的注释求值直接得到。

□

现在我们可以陈述我们的充分性结果：

引理 4（充分性）。假设 $\cdot;\ p \vDash_{\mathsf{cp}} e :\ \tau;\ Q$。对于任意 $r \geq 0$ 和执行 $(e, p+r) \mapsto^* (e', q)$，以下两种情况之一成立：

- $e' \in \mathcal{V}(\tau)$ 且 $q \geq r + \Phi(e' :\ \langle\tau;\ Q\rangle)$，或
- 存在某个 e'' 和 q'，使得 $(e',\ q) \mapsto (e'',\ q')$。

最弱前置条件关系 $\boxed{\mathcal{WP}_c[\Psi]_n(e,q)}$

$$\mathcal{WP}_{\mathsf{cp}}[\Psi]_0(e,q) \triangleq \top$$

$$\begin{aligned}\mathcal{WP}_{\mathsf{cp}}[\Psi]_{n+1}(e,q) \triangleq\ & (e\ \mathrm{val} \wedge \Psi(n+1,e,q)) \vee \\ & ((\exists e', q'.\ (e,q) \mapsto (e',q')) \wedge \forall e', q'.\ (e,q) \mapsto (e',q') \Rightarrow \mathcal{WP}_{\mathsf{cp}}[\Psi]_n(e',q'))\end{aligned}$$

$$\begin{aligned}\mathcal{WP}_{\mathsf{cf}}[\Psi]_{n+1}(e,q) \triangleq\ & (e\ \mathrm{val} \wedge \Psi(n+1,e,q)) \vee \\ & ((\exists e'.\ e \to e') \wedge \forall e'.\ e \to e' \Rightarrow \mathcal{WP}_{\mathsf{cf}}[\Psi]_n(e',q))\end{aligned}$$

值逻辑关系 $\boxed{\mathcal{V}[\![\tau]\!]_n(v)}$

$$\mathcal{V}'[\![\tau]\!]_0(v) \triangleq \top$$

$$\begin{aligned}
\mathcal{V}'[\![\mathbf{1}]\!]_{n+1}(v) &\triangleq v = \mathsf{tt} \\
\mathcal{V}'[\![\tau_1 \times \tau_2]\!]_{n+1}(v) &\triangleq \exists v_1, v_2.\ v = \mathsf{pair}(v_1;\ v_2) \wedge \mathcal{V}'[\![\tau_1]\!]_{n+1}(v_1) \wedge \mathcal{V}'[\![\tau_2]\!]_{n+1}(v_2) \\
\mathcal{V}'[\![\tau_1 + \tau_2]\!]_{n+1}(v) &\triangleq (\exists v_1.\ v = \mathsf{inl}\, v_1 \wedge \mathcal{V}'[\![\tau_1]\!]_{n+1}(v_1)) \vee (\exists v_2.\ v = \mathsf{inr}\, v_2 \wedge \mathcal{V}'[\![\tau_2]\!]_{n+1}(v_2)) \\
\mathcal{V}'[\![\langle \tau_1 \to \tau_2, \Theta_{\mathsf{cp}}, \Theta_{\mathsf{cf}} \rangle]\!]_{n+1}(v) &\triangleq \exists f, x, e.\ v = \mathsf{fun}(f, x.\, e) \wedge \forall m \le n. \\
&\quad (\forall (P,Q) \in \Theta_{\mathsf{cp}}.\ \forall q \ge 0, v'.\ \mathcal{V}[\![\tau_1]\!]_m(v') \Rightarrow \\
&\qquad \mathcal{WP}_{\mathsf{cp}}[\lambda(m', v'', q').\ \mathcal{V}[\![\tau_2]\!]_{m'}(v'') \wedge \\
&\qquad\quad q' \ge q + \Phi(\mathsf{pair}(v';\ v'') : \langle \tau_1 \times \tau_2; Q \rangle)]_m \\
&\qquad ([v'/x, v/f]e, q + \Phi(v' : \langle \tau_1; P \rangle))) \wedge \\
&\quad (\forall (P,Q) \in \Theta_{\mathsf{cf}}.\ \forall q \ge 0, v'.\ \mathcal{V}[\![\tau_1]\!]_m(v') \Rightarrow \\
&\qquad \mathcal{WP}_{\mathsf{cf}}[\lambda(m', v'', q').\ \mathcal{V}[\![\tau_2]\!]_{m'}(v'') \wedge \\
&\qquad\quad q' \ge q + \Phi(\mathsf{pair}(v';\ v'') : \langle \tau_1 \times \tau_2; Q \rangle)]_m \\
&\qquad ([v'/x, v/f]e, q + \Phi(v' : \langle \tau_1; P \rangle))) \\
\mathcal{V}'[\![\mu\alpha.\, \tau]\!]_{n+1}(v) &\triangleq \exists v'.\ v = \mathsf{fold}\, v' \wedge \mathcal{V}'[\![[\mu\alpha.\, \tau/\alpha]\tau]\!]_n(v')
\end{aligned}$$

$$\mathcal{V}[\![\tau]\!]_n(v) \triangleq v \in \mathcal{V}(\tau) \wedge \mathcal{V}'[\![\tau]\!]_n(v)$$

Fig. 6. Weakest precondition relation and logical relation on values.

定理 4 是上述两个引理的推论。