使用自动资源分析处理异常与效应

ETHAN CHU*，YIYANG GUO*，以及JAN HOFFMANN，美国卡内基梅隆大学

A

通过分析程序的源代码，可以自动导出参数化资源（或代价）界限，已有许多相关技术。这些技术对大量程序和语言特性都行之有效。然而，异常或效应处理器所需的非局部控制转移仍然是一项挑战。

本文提出了首个支持异常或效应与其处理器之间非局部控制转移的自动资源上界分析。该分析扩展了基于类型的自动摊销资源分析（AARA），后者可自动化摊还分析中的势能方法。本文针对一种带有列表和线性势函数的简单函数式语言介绍了该分析。不过，这些思想可直接应用于更丰富的场景，并已针对 Standard ML 和多项式势函数实现。

除异常与效应的新类型系统外，本文的一项主要贡献是提出了一种新颖的句法类型可靠性定理，该定理确立了导出界相对于基于栈的抽象机的正确性。实验评估表明，这种新分析能够分析现有方法无法处理的程序，而且支持异常和效应处理器所带来的效率开销很低。

CCS 概念：• **计算理论** → **控制原语**；**程序分析**；**类型论**；**抽象机**；**操作语义**。

附加关键词和短语：自动资源上界分析、成本分析、类型系统、代数效应、计算效应、效应处理器、异常、自动摊销资源分析、K-Machine

ACM 引用格式：
Ethan Chu、Yiyang Guo 和 Jan Hoffmann。2026。使用自动资源分析处理异常与效应。《美国计算机协会程序设计语言会议论文集》10，OOPSLA1，文章99（2026年4月），41 页。https://doi.org/10.1145/3798207

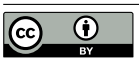
arXiv:2603.02260v1 [cs.PL] 28 Feb 2026

1 引言

自动资源上界分析的目标是静态且自动地推断程序资源使用量的信息。在此语境下，资源是程序执行过程中使用的量，例如时间、能量或内存。此类信息对软件库的用户很有用 [62]，，可用于安排数据中心中的作业，确保在嵌入式系统等资源受限环境中 [20, 107] 以及智能合约中 [6, 32]，保障安全与可靠性，并自动检测性能缺陷 [37]。现有许多推断资源信息的方法，包括类型系统 [10, 48, 60, 101, 104, 109]，、推导并求解递推关系 [4, 28, 42,71, 106]，以及其他静态分析 [12, 19, 46]。大多数技术旨在推导（最坏情况）上界。然而，一些分析关注（最好情况）下界 [8, 40, 85]以及期望

*前两位作者对本文的贡献相同。

作者联系信息：Ethan Chu，ethanchu@cmu.edu；Yiyang Guo，yiyangg@alumni.cmu.edu；Jan Hoffmann，jhoffmann@cmu.edu，卡内基梅隆大学，美国匹兹堡。



本作品采用知识共享署名 4.0 国际许可协议授权。©2026 版权所有归所有者/作者所有。 ACM 2475-1421/2026/4-ART99https://doi.org/10.1145/3798207

代价 [13, 24, 84]。自动资源界限分析是一个不可判定的问题，但该领域的研究仍在不断拓展可推导的界限类别、提高分析效率，并支持更多语言特性。

非局部控制。 自动资源分析中最具挑战性的编程语言特性，有一些涉及非局部控制转移，例如异常处理器、续延 [38,91],和效应处理器 [15, 89]。

异常 是一种常见的语言特性，可以通过动态地将控制转移到异常处理程序来重定向控制流，而该处理程序通常位于先前调用的函数中。推导最坏情况界时，处理不带处理器的错误或异常 [57]相对直接。原因在于，发生错误时提前终止，与无错误情况相比会降低资源开销。异常处理器之所以具有挑战性，是因为资源分析通常围绕结构递归或简单的 for 循环等常规控制流模式设计。据我们所知，目前只有一种资源分析有限支持函数内异常处理程序 [4], ，也就是说，只能捕获在同一次函数调用中抛出的异常。我们尚未发现支持一般异常的资源分析；在一般异常中，抛出的异常无法在静态分析时与同一函数中的处理程序关联起来。

代数效应 以及与之相关的效应处理器编程范式，在过去几年中逐渐受到编程语言社区的关注 [34, 35, 97, 100]。它们可以视为异常的一种推广：用户定义一组操作（或效应），其实现取决于上下文，并由效应处理器提供。与异常处理器相比，效应处理器给资源分析带来了额外的难题：它们可以访问分隔延续 [38, 72] ，以在操作调用处恢复计算，或为延迟的计算提供新的效应处理器。分析带有续延的程序很困难，因为动态控制流难以预测。据我们所知，目前还没有支持续延、分隔延续或效应处理器的资源分析方法。

带异常和效应程序的资源上界。 本文介绍了首个针对计算效应及其处理器的资源上界分析。此外，这也是首个全面支持异常处理器以及能够在不同函数之间转移控制权的异常的资源上界分析。我们扩展了一种基于类型的技术，称为自动摊销资源分析（AARA） [58], ，该技术由 Hofmann 和 Jost [60]提出。在 AARA 中，类型构成资源上界，而类型推导则是关于该上界相对于操作代价语义可靠性的证书。该分析以资源为参数，并能推导非单调资源度量的高水位界限。该分析支持递归类型 [44], 、高阶函数 [63], 、数组和引用 [80]。AARA 可视为对摊还分析中势能方法的自动化，而势函数的使用对于处理异常处理器至关重要。

贡献。 我们保守地扩展 AARA，为异常、计算效应以及支持一次性续延的处理器添加类型规则，从而能够计算取决于非局部控制转移的资源开销。执行异常处理程序或效应处理器的代价由异常或效应载荷的某个函数界定。该分析以载荷的类型为参数。简而言之，其思路是由处理器建立一项契约，其中包含一个势函数，用于指定异常或效应需要提供的势能。本文重点研究一种简单的按值调用语言中的线性最坏情况界，该语言支持计算效应及其处理器、高阶函数、列表、和类型和积类型。不过，将这些技术扩展到更多语言特性和更广泛的界，与本文工作正交。事实上，我们已将该技术实现用于递归类型和多项式

界。我们还展示了如何从效应和效应处理器推导出异常、异常处理程序及其相应的类型规则。 除了设计资源感知型类型系统外，主要技术贡献还包括可靠性证明、实现以及实验评估。

可靠性证明。本工作的一项技术创新是一种新颖的证明技术，用于确立导出界相对于基于栈的抽象机的可靠性，该抽象机规定了程序的代价与行为。几乎所有以往的可靠性证明都基于大步成本语义以及 let 正规形式中的表达式 [54, 57, 60, 67]。本文采用包含进展与保持定理的小步风格可靠性论证。带有抽象机的小步方法支持续延，而续延是对计算效应建模所必需的。我们不使用 let 正规形式，而是以细粒度值调用的方式，在值与表达式之间采用模态分离 [79]。纯值与可能带有效应的计算之间的这种显式分离简化了类型规则、求值规则和证明。Grosen 等人给出了另一种基于小步语义的、针对 AARA（不含异常）的可靠性证明 [44]。区别在于，本文的证明是完全句法的，不依赖逻辑关系，并使用基于栈的抽象机以及表达式与值的模态分离。另一种方法是使用大步成本语义和逻辑关系来证明可靠性 [90]。

实现与实验评估。我们已为 Standard ML (SML) 程序实现了该分析，以评估其有效性、与先前不支持异常处理器或效应的 AARA 实现相比的开销，以及导出界的质量。该实现使用 MLton SML 编译器 [105] 作为前端，将 SML 程序转换为中间表示。与论文中的介绍不同，我们支持以递归类型的值大小为变量的多项式界。我们使用 20 个基准程序评估了该分析的实现，这些程序均使用异常或效应。其中，14 个无法通过现有技术进行分析，因为它们要么处理异常，要么使用效应。我们发现，该实现能高效地为这些基准程序求得紧确的最坏情况界，并且与 RaML [56]等现有 AARA 实现相比，我们的方法开销较低。

2 概述

本节介绍基于类型的自动摊销资源分析（AARA）的基本思想和最新进展，以及阻碍这类系统分析含异常程序的挑战。随后，我们讨论本文的主要技术创新，即（第 2.2 节）对 AARA 的模块化扩展，可与现有类型系统结合，为含异常处理程序的程序导出界；以及（第2.3 节）对效应处理器的支持。

2.1 基于类型的自动摊还资源分析

基于类型的 AARA 由 Hofmann 和 Jost [60] 提出，用于自动导出线性界，以分析一阶函数式程序的堆内存使用量。它的优点是

- 它能为许多典型函数导出紧确的符号化最坏情况界，
- 已证明它相对于代价语义是可靠的，
- 类型推导是导出界的易于检查的证书，
- 类型（因而也包括界）的推断可以归约为线性规划（LP），并且
- 将类型与摊还分析的物理学家方法相结合，使该系统天然具有组合性。

也许令人惊讶的是，基于类型的摊还资源分析的这些性质，在将原始工作保守地扩展到任意用户定义的非线性资源度量时，仍得以保留。

界限 [54, 61, 66], 一般递归类型 [44], 下界 [85], 和高阶函数 [63]。近期的一篇综述对此进行了更全面的讨论 [58]。为了聚焦于主要思想，我们基于一种严格求值的高阶函数式语言（支持列表），以及一种仅编码线性资源界的类型系统，来阐述 AARA 向异常和效应的扩展，并将其称为线性界 AARA。不过，我们认为，只需稍作修改，这种扩展便可应用于更复杂的 AARA 系统，例如能够推导多项式资源界限的系统。本文不讨论这些修改，但举例来说，第 5 节中的评估便使用了扩展了异常和效应的多项式 AARA。

成本模型。针对函数式语言，已有多种成本与资源模型，分别用于描述运行时等单调资源，以及空间使用等非单调资源。为了从特定成本模型中抽象出来，我们沿用先前的资源分析研究 [43, 52]，为语言加入一种计时标记效应——我们可以将表达式 tick q 理解为把固定有理数 q 加到表示已产生代价的某个全局计数器上。尽管计时标记效应接受固定参数，我们仍能模拟非恒定计时标记。例如，我们可以定义一个函数 tick_nat，使得 tick_nat n 最终执行 n 次计时标记。具体做法是将 n 表示为长度为 n 的单元素列表，并对该列表应用递归函数；该函数遍历列表，并在每个元素处执行 tick 1。

tick 资源模型是一种成熟且适用的方法。要为时间（或空间）定义可靠的成本模型，就需要依据某种机器模型对其进行论证。这称为有界实现，由 Blelloch 和 Greiner [17, 18] 首创。众所周知，恒定的计时标记注释对应于在随机存取机器（RAM）上运行解释器，二者相差常数因子。此类成本模型不一定会产生与 β 归约期间替换导致的理论规模增长成比例的代价[29]。相反，我们可以利用一种能更高效地模拟 λ 演算的抽象机模型，例如 Accattoli [1] 所述的显式替换/共享。

在本文的大多数示例中，我们会在代码中插入计时标记，以测量求值的函数调用 以及异常或效应处理器求值的次数。这是一种有趣的资源度量，因为在函数式程序中，函数调用和处理器求值的次数与其渐近运行时间复杂度成正比。此外，这种度量很容易扩展得更贴近实际——已有研究展示了如何选择计时标记所代表的代价，使导出的界对应于执行编译后代码所需的实际时钟周期 [63]。

读者可能会立即对我们选择的资源度量产生疑虑：它假设每次运行处理器的代价都是常数，而非动态变化；但在定位要使用的正确效应处理器时，代价往往会动态变化。我们的类型系统通过两种方式避开了这一问题。首先，我们的处理器要求显式转发 [50, 68], , 也就是说，每个处理器都必须处理被处理的计算可能执行的所有 效应。对于它实际上并不处理的效应，它仍必须显式处理、通过计时标记产生代价，然后重新执行该效应，从而将其转发出去。这样可以确保每当执行某个效应时，控制流总是转移到最近的外层效应处理器，而不必动态搜索能够处理给定效应的处理器。于是，在每个转发点触发的计时标记，实际上模拟了识别合适处理器所需的非恒定代价。其次，通过要求处理器中的分隔延续遵循单次使用约束，我们确保非处理器栈帧不会被执行多次，因此运行时只需存储一个指向最近外层效应处理器的指针，而不必复制或调换栈帧 [96]。综上，每次处理器求值只产生常数代价是合理的。

势能方法。 摊还分析中的势能方法由 [98] 引入，用于分析一系列数据结构操作的代价。其核心思想是为

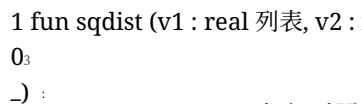


图 1. SML 函数 sqdist 计算两个 n维向量（实数列表）之间欧氏距离的平方。函数 R.tick 用于指定资源使用量。

数据结构以及某项操作的摊销成本。接着，必须证明，对于所有可能的执行，摊销成本与数据结构所携带的势能足以支付实际成本以及所得数据结构的势能。因此，初始势能与摊销成本总和构成程序资源消耗的上界。

资源标注类型。 AARA 基于摊还分析的势能方法。为了自动进行摊还分析，它通过类型固定可能的势函数集合。具体而言，类型会导出势函数，将给定类型的数据结构映射为势能，且势能取非负数。随后，类型规则确保引入或消去这些类型的项，以符合势函数的方式保持势能。请注意，在 AARA 中，摊销成本始终为零，因此程序的代价完全由初始势能限定。

在包含函数、列表、和类型和积类型的简单类型系统之上，AARA 为某些类型添加势注释：

(1) 列表类型会标注一个常数势，该势由列表中的每个元素携带。例如，类型 $L^q(\text{unit})$ 表示一个单元素列表，其中每个元素携带 q 势。换言之，它定义了势函数 $\Phi(v\colon L^q(\text{unit})) = q|v|$。(2) 和类型的每个变体都会标注一个常数势，该势由注入到该变体时携带。例如，类型 $\text{unit}^p + \text{unit}^q$ 表示一个布尔值：为真时携带 p 势，为假时携带 q 势。具体而言，它定义了势函数 $\Phi(v : \text{unit}^p + \text{unit}^q) = p \text{ if } v = \text{inl}() \; q \text{ if } v = \text{inr}()$。(3) 对于函数，参数类型会标注一个常数势，调用函数时必须连同参数一起提供；结果类型则会标注一个常数势，该势会与结果一起返回。例如，类型为 $\tau_1 \to_q^p \tau_2$ 的函数要求连同参数一起提供 p 势，并会连同结果返回 q 势。

示例。 考虑图 sqdist 中的函数 1，它计算两个 n维向量之间欧氏距离的平方，这些向量以实数列表表示（$L(\mathbb{R})$）。它递归遍历两个列表，直到至少一个列表为空；如果第E mis$i$ 个列表较短（ i，则抛出异常 $i = 1,\ 2$）。由于所选资源度量是已求值函数调用的次数，调用 sqdist(v1,v2) 的代价为 min(|v1|, |v2|)，我们希望用 AARA 推导出这一结果。事实上，利用线性界 AARA 简单且局部的类型规则，我们可以为sqdist证明许多可能的类型，例如

$$L^1(\mathbb{R}) \times L^0(\mathbb{R}) \to_0^0 \mathbb{R} \qquad L^0(\mathbb{R}) \times L^1(\mathbb{R}) \to_3^3 \mathbb{R} \qquad L^{0.5}(\mathbb{R}) \times L^{0.5}(\mathbb{R}) \to_5^5 \mathbb{R}$$

给定函数调用 $\text{sqdist}(v_1, v_2)$，这些类型分别表示成本上界 $|v_1|$、 $|v_2| + 3 - 3$ 和 $0.5|v_1| + 0.5|v_2| + 5 - 5$。这三个界都是 $\min(|v1|, |v2|)$ 实际成本的上界。哪个界最合适，取决于调用 $\text{sqdist}(v_1, v_2)$ 所处的上下文。为此，我们的类型系统实际上将函数类型表示为一组可能的类型。在类型推断过程中，这个集合由我们推断出的线性约束决定。例如，sqdist 会被推断为具有以下类型：

$$\text{sqdist} : \{L^{q_1}(\mathbb{R}) \times L^{q_2}(\mathbb{R}) \to_{p'}^{p} \mathbb{R} \mid q_1 + q_2 \geq 1 \wedge p \geq p'\}$$

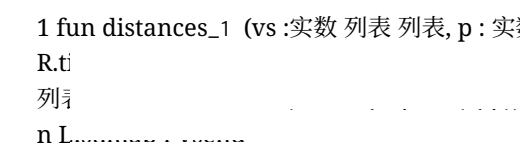


图 2. SML 中的函数 distances_1 和 distances_2 。每当 distances_2 中的异常处理程序被执行时，都会产生 1 的代价。函数调用 List.map f l 会将 f 应用于 l 中的每个元素。

因此，在 sqdist 的每个调用点，线性规划求解器都会为势变量 q_1、q_2、p、p' 选择最佳的具体取值，以最小化调用 sqdist 的较大程序的代价。从这个意义上说，我们的类型系统支持资源多态性。不过，为求简洁，本文中的大多数示例在函数定型时只讨论单态类型签名。

通过为结果类型赋予势能，可以实现组合性。例如，如果我们希望将表达式 sqdist(append(v_1、v_2) v') 定型为 v': $L^0(\mathbb{R})$，那么就必须构造一个将 append 的结果类型定为 $L^1(\mathbb{R})$ 的定型。接着，假设 append 不包含 tick 表达式（因此代价为 0），我们可以为它赋予以下类型：

$$\text{append} : L^1(\mathbb{R}) \times L^1(\mathbb{R}) \to^0_0 L^1(\mathbb{R})$$

此类型体现了这样一点：将两个列表拼接起来，而两个列表中的每个元素都携带一个势能，所得列表中的每个元素也会携带一个势能。

2.2 异常处理

虽然有一些 AARA 系统支持错误（或抛出异常），但目前还不存在支持异常处理的基于类型的 AARA 系统；可以说，异常支持中真正不平凡的部分正是本文所发展的内容。在本小节中，我们将分析图2_1 中的函数distances_2 和distances；它们调用了前文讨论过的sqdist （见图1），并处理它抛出的Emis异常。为简单起见，我们将忽略List.map所引发的函数调用，因此在以下讨论中，我们的成本度量仅为对*sqdist* 的函数调用次数，以及异常处理程序求值次数。

考虑函数 distances_1。它接受一个由向量组成的列表 vs 和一个点 p（一个向量），并调用标准列表映射函数 List.map，使用函数 sqdist 计算 p 到列表中每个向量的平方距离。请回想，我们不计 List.map 所引发的函数应用次数，因此，对 distances_1([$v_1 \ldots, v_n$],p) 求值的代价为$n + \sum_{1 \le i \le n} \min(|v_i|,\ |p|)$。对此，一个有效的上界可由以下 AARA 定型编码

$$\text{distances\_1} : L^1(L^1(\mathbb{R})) \times L^0(\mathbb{R}) \to^0_0 L^0(\mathbb{R})$$

这对应于上界 $n + \sum_{1 \le i \le n} |v_i|$。实际上，我们可以通过部分依赖先前推导出的定型 sqdist : $L^1(\mathbb{R}) \times L^0(\mathbb{R}) \to^0_0 \mathbb{R}$ 得到这一结果。distances_1 会将 sqdist 映射到其第一个参数上，而 sqdist 要求其第一个参数的每个元素具有 1 单位势能，因此，distances_1 要求其第一个参数的每个内层列表元素具有 1 单位势能，这并不令人意外。稍微有趣的是，为了支付 distances_1 中对 sqdist 的首次函数调用（第 2 行 lambda 中的 R.tick 1），disances_1 必须为每个外层列表元素携带 1 单位额外势能。

接下来，考虑函数 distances_2。它执行的计算与 distances_1 相同，但会处理异常 Emis2，而 sqdist 可能会引发该异常。处理程序中的 R.tick 表达式指定：如果执行了处理程序，则额外产生 1 的代价。如果我们只是将处理程序视为顺序计算，就会得出计算 distances_2([$v_1 \ldots, v_n$],p) 的代价界 $n + \sum_{1 \le i \le n}(1 + |v_i|)$，即 $2n + \sum_{1 \le i \le n} |v_i|$。乍看之下，这似乎是一个紧确界。

[1]在分析distances_1 和distances_2 在求值部分5时，我们确实会计算由List.map引发的函数调用。这只是为了概述而做的简化。

然而，本文介绍的类型系统可以通过推导出以下定型来证明更精确的界 $n + \sum_{1 \le i \le n} |v_i|$，其定型与 distances_1的定型相同。[2]

$$\text{distances\_2} : L^1(L^1(\mathbb{R})) \times L^0(\mathbb{R}) \to^0_0 L^0(\mathbb{R})$$

在讨论如何推导出这个紧确界之前，我们先直观地理解它为何正确。首先注意，如果在函数 sqdist 中完整遍历每个内层列表 v_i，那么界 $\sum_{1 \le i \le n} |v_i|$ 是紧确的（我们不妨将这种情况称为最坏情况）。现在考虑在函数 sqdist 中引发异常 Emis2 的情况。在这种情况下，sqdist 的第二个参数比第一个参数短，因此相应的内层列表 v_i 并未被完整遍历。因此，计算处理程序时，我们并不处于前面描述的“最坏情况”：额外代价 1 不会超过完整遍历 v_i 且不处理异常的代价。

我们如何在足够简单、能够支持类型推断的类型系统中表达这一微妙之处？我们的思路是为异常类型关联势注释，从而在引发异常的代码点与处理异常的代码点之间建立契约。势能的大小经过选择，以匹配最内层异常处理程序的代价；势能可以在此处消耗。作为回报，每次引发异常时，都必须提供相同数量的势能。

为了直观地理解这一思路，我们来尝试证明上文所示的距离函数的定型。大体上，我们可以重复为距离函数定型时所采用的步骤，但现在必须决定为异常类型 $\text{Emis1}^p + \text{Emis2}^q$ 标注多少势能 p, q。没有处理异常 Emis1 的处理程序，因此它不携带任何势能，也就是说，我们令 $p = 0$。相比之下，Emis2 的处理程序会执行 R.tick 1，这意味着 Emis2 应携带 1 个单位的势能，也就是说，我们令 $q = 1$。因此，这个异常处理程序是良好定型的。

但从契约的另一端来看，我们必须确保在距离函数中 raise Emis2 的任何一方都必须提供 1 个单位的势能。为此，我们现在必须为 sqdist 构造一个新的定型，具体来说，是为它在距离函数中的调用点构造定型。此前，我们依赖 $\text{sqdist} : L^1(\mathbb{R}) \times L^0(\mathbb{R}) \to^0_0 \mathbb{R}$ 为距离函数定型，因此现在就从这里重新开始。sqdist 的第一个参数 v1 的类型为$L^1(\mathbb{R})$，也就是说，每个元素都携带 1 个单位的势能。接下来请注意，raise Emis2 出现在v_1 匹配 _::_的情况中，这意味着它的长度以及所携带的势能为 ≥ 1，可用于 raise Emis2。因此，我们可以证明 sqdist 具有以下定型，并且现在带有一个带注释的异常类型：

$$\text{sqdist} : L^1(\mathbb{R}) \times L^0(\mathbb{R}) \to^0_{0 \odot 0} \mathbb{R} \odot (\text{Emis1}^0 + \text{Emis2}^1)$$

在此，我们将带注释的函数类型修改为$\tau_1 \to^{\tau_2}_{q \odot q_e} \tau_2 \oplus \tau_e$，其中 τ_e^p是资源带注释异常类型，而 q_e 是随抛出的异常一并返回的额外势能。[3] 因此，该定型表明，在对 sqdist(v1,v2) 求值时，|v1| 足以覆盖和中的 R.tick，并在抛出 Emis2 时提供 1 单位额外势能。

我们的类型系统也能够为携带非平凡载荷（如列表）的异常定型。接下来关于处理计算效应的一节将展示这样一种载荷。

2.3 处理计算效应

本节将展示我们的类型系统如何分析执行并处理效应的程序。我们将考虑配备一组具名效应的程序，每种效应都有输入类型和输出类型。例如，典型的输入输出效应 Print 的输入类型可以是字符串，输出类型可以是单位类型。效应 EName 接收一个属于其输入类型的载荷 x，并通过表达式 do[EName] x 执行，该表达式的类型为该效应的输出类型。例如，do[Print] "Hello World"

[2]在本例中，差异仅为常数因子。然而，此类不精确性在更大的程序中很容易被放大，并导致导出界出现渐近差异。
[3]在我们的大多数示例中，τ_e 是包含程序中所声明的各种异常的和类型，此时由于和类型每个变体上的注释，q_e 是冗余的。不过，我们的类型系统对 τ_e没有任何限制。

1 效应 Insert：整数 列表 $\Rightarrow$ 单位类型 16 fun [illegible]
Ren [illegible] fu [illegible]
类型 [illegible] x::x [illegible]
x = [illegible] ick [illegible]
fun [illegible]
n st [illegible] $\Rightarrow$ [illegible]
(tra [illegible] E [illegible]
$\Rightarrow$ [illegible] 列 [illegible]
fn ([illegible]

图 3. 使用效应处理器实现栈存储的示例。假设可以使用 traverse : $L(\mathbb{Z})$ $\to$ 单位类型。

是一个以字符串 "Hello World" 作为载荷来执行 Print 效应的表达式，其类型为单位类型，因为这就是输出类型。

为了处理效应，我们使用如下形式的效应处理器表达式：e handle return a => e1 | EName x k => e2 |。.. 其中 e 是要处理的效应表达式。当 e 求值为纯值 v时，会将 v 代入 e1 中，以得到最终值。然而，如果执行了某个效应 EName，就会执行处理器分支 e2，并传入该效应的载荷 x，以及一个（定界的）续延 k；该续延允许从执行 EName 的位置恢复 e 的执行。我们采用 Kammar 等人 [68], 提出的深层处理器设定，因此续延 k 会重新安装效应处理器。

请看图 3 中的示例。我们首先定义两种效应：Insert 需要一个整数列表载荷，并返回单位类型；Remove 需要一个单位类型载荷，并返回一个整数列表或 None。两者共同用于实现存储数据结构。函数 hstore 接受一个延迟计算 m，它可能执行 Insert 和 Remove 效应，并对这些效应进行处理。在处理程序的两个分支中，我们使用 R.tick 指定：如果处理程序被求值，则代价为 1，这与处理异常时的做法相同。这个效应处理器使用函数来编码状态，并在过程中传递存储状态。Insert 分支是一个函数：它先在单位类型上调用续延 k，然后将效应负载 n 添加到表头，以此更新存储状态。Remove 分支则根据当前存储状态进行分情况处理。如果存储为空，它会在 None 上调用续延 k，随后将空列表作为状态传递。如果存储非空，它会在 Some x 上调用续延 k，其中 x 是存储的表头，并将状态更新为存储的表尾 xs。

现在我们来看第 26 行要处理的计算：store_lists。给定一个整数列表的列表 l，它首先调用 insert_lists l，该函数对 l 中的每个元素执行 Insert 效应。接下来，它调用 consume ()，反复执行 Remove 效应，直到返回 NONE，这表示存储已清空。重要的是，对于 Remove 输出的每个 SOME x，我们都会调用 traverse x；该函数会消耗 x 中每个元素对应的 1 单位势能，也就是说，我们假设它的定型为 traverse : $L^1(\mathbb{Z})$ $\to_0^0$unit。于是，类型系统能够推导出 store_lists $[v_1, \ldots, v_n]$ 的代价为 $1 + 2n + \sum_{1\leq i\leq n} |v_i|$，也就是以下类型。

$$\text{store\_lists} : L^2(L^1(\mathbb{Z})) \to_0^1 \text{unit}$$

我们首先论证该界限为何正确。在存储_列表中有三处会产生代价：调用 insert_列表 $[v_1, \ldots, v_n]$ 会将每个 v_i恰好插入存储一次，因此产生 n总

代价。调用 consume () 会从存储中移除元素，直到（并包括）其为空，从而产生代价 $n+1$。最终，会对从存储中 x 移除的每个元素调用 traverse x，这些元素是 $v_1,\ldots,v_n$，总共产生的代价为 $\sum_{1\le i\le n}|v_i|$。

但是，我们的类型系统是如何弄清这一点的呢？答案在于为效应使用带注释类型。我们建立在小节 2.2 中所解释的异常类型系统之上。现在，函数定型中的带注释异常类型被替换为一个效应签名，其中包含其函数体可能执行的每个效应的带注释输入类型和输出类型。与带注释异常类型类似，这些资源注释效应签名是客户端代码与处理程序之间的契约。让我们看看示例中 insert_列表 和consume 的定型。

$$\text{insert\_lists} : L^2(L^1(\mathbb{Z})) \to^0_0 \text{unit} \odot \{\text{Insert} : L^1(\mathbb{Z}) \Rightarrow^2_0 \text{unit}\}$$

insert_列表的类型保证，当该函数执行 Insert 效应时，它将提供一个列表载荷：每个元素携带 1 点势能，并额外携带 2 点势能，同时预期处理程序返回 0 点势能。其定型相对直接——其参数 $[v_1,\ldots,v_n]$是一个列表的列表，每个列表携带 $2+|v_i|$ 点势能；它只需通过载荷将该参数转发给 Insert 处理程序。

$$\text{consume} : \text{unit} \to^1_0 \text{unit} \odot \{\text{Remove} : \text{unit} \Rightarrow^1_0 \left(\text{Some}^1(L^1(\mathbb{Z})) + \text{None}^0\right)\}$$

consume 的类型承诺，当函数执行 Remove 效应时提供 1 单位势能，并期望返回时得到 Some x（携带 $1+|x|$ 单位势能）或 None（不携带势能）。它用参数携带的 1 单位势能支付执行 Remove 的开销。随后，对 Remove 的结果进行分情况处理时，第 24 行 Some 分支中的 $|x|$ 单位势能支付调用 traverse x 的开销，而剩余的 1 单位势能则支付递归调用 consume () 的开销。

在契约的另一端，hstore 中的效应处理器必须能够处理上述带注释类型的 Insert 和 Remove 效应，即

$$\{\text{Insert} : L^1(\mathbb{Z}) \Rightarrow^2_0 \text{unit}, \text{Remove} : \text{unit} \Rightarrow^1_0 \left(\text{Some}^1(L^1(\mathbb{Z})) + \text{None}^0\right)\}$$

此外，效应处理器的所有分支都必须具有相同的类型——在这个处理器中，所有分支的类型都是 $L^1(L^1(\mathbb{Z})) \to^0_0$ 单位类型，理由如下。(1) 第 6 行的返回分支是一个不操作其参数的函数，因此显然满足该类型。(2) 第 7 行的 Insert 分支开始时有 $|n|+2$ 单位势能可用，其中 n 是载荷。1 单位势能支付 R.tick 1 的开销。在函数中，store 起初的类型为 $L^1(L^1(\mathbb{Z}))$。通过 n :: store 将 n 和剩余的 $|n|+1$ 单位势能添加到存储中时，这个类型保持不变。(3) 第 9 行的 Remove 分支开始时有 1 单位势能，可用于支付 R.tick 1 的开销。在函数中，我们对 store 进行分情况处理，此时它的类型仍为 $L^1(L^1(\mathbb{Z}))$。如果 store 为空，我们就不携带任何势能地向客户端发送 None。如果 store 非空，即匹配 x::xs，我们就向客户端发送 Some x，并附上该表头元素所提供的 $|x|+1$ 单位势能。

与用于处理异常的**raise** v 类似，我们可以通过载荷 v，将复杂的势能从**do**[EName] v 传递给最近的外层效应处理器。不过，与异常处理器不同，效应处理器还可以通过其定界续延将势能传回带效应计算。

3 类型系统

本节介绍一种简单函数式语言的语法，以及用于推导资源上界的类型规则。我们的语言最初只包含异常及其处理器，之后会将其扩展为也支持效应。我们还会简要讨论类型推断。

3.1 语法

我们基于一种包含（高阶）函数、列表、对和二元和类型的语言构建类型系统。在实现中，我们使用了一种更完整的函数式语言，其中包括正则

递归类型。我们将讨论范围限制在简单的情形，因为它已经呈现了异常处理器（以及效应）带来的所有技术难题。

$$
\begin{array}{ll}
v ::= x & e ::= \text{值}(v) \mid \text{let}(e_1;\ x.e_2) \mid \text{计时标记}\{q\} \mid \text{应用}(v_1;\ v_2) \\
\quad \mid \text{函数}(f.x.e) & \quad \mid \text{二元组分支}(v; x_1.x_2.e) \\
\quad \mid \mid \text{二元组}(v_1; v_2) & \quad \mid \text{空类型分支}(v) \mid \text{和类型分支}(v; x_1.e_1;\ x_2.e_2) \\
\quad \mid \text{左注入}(v) \mid \text{右注入}(v) & \quad \mid \text{列表分支}(v; e_1;\ x_1.x_2.e_2) \\
\quad \mid \text{空列表} \mid \text{cons}(v_1; v_2) & \quad \mid \text{raise}(v) \mid \text{try}(e_1;\ x.e_2)
\end{array}
$$

我们在语法上区分值 v 与表达式/计算 e，采用基于宽松逻辑的模态分离 [33] 以及细粒度值调用 [79]。值包括平凡的单位值、函数、对、注入和列表。由于我们在值调用设定下工作，变量代表值，因此被视为值。在示例和实现中，我们使用 SML 语法，它可以转换为宽松语法。

在不限制表达能力的前提下，表达式的子成分尽可能为值。它们包括函数应用，以及对、注入和列表的模式匹配消去。计算的顺序求值仅发生在let表达式中。值 v 通过语法形式 $\text{val}(v)$ 提升为表达式。语法形式计时标记$\{q\}$指定消耗 $q \in \mathbb{Q}$ 资源。通常，它们可以根据能够建模运行时、内存使用或能量消耗的资源度量，自动插入到更高层代码中。若 $q < 0$，则q 资源变为可用。例如，这用于标记内存被释放的时机。最后，可以使用计算 $\text{raise}(v)$ 引发异常。计算 $\text{try}(e_1;\ x.e_2)$ 定义一个异常处理程序 e_2，用于处理由 e_1引发的异常。

3.2 资源注释类型

资源标注类型定义如下。

$$\tau ::= \mathcal{T} \mid \text{unit} \mid \tau_1 \times \tau_2 \mid \text{void} \mid A_1 + A_2 \mid \text{L}(A) \qquad A, B, C ::= \langle \tau, q \rangle \qquad \mathcal{T} ::= \{A \to B \odot C \mid \Theta\}$$

其核心思想是，在类型 τ 上标注非负数，以定义一个势函数$\Phi(v:\ \tau)$，将类型 τ 的值 v 映射为非负数。下一小节将正式定义势函数。

类型 τ包括箭头类型、列表、积类型和和类型。 A、 B、 C是附带常数势$q \in \mathbb{Q}_{\geq 0}$ 的类型，因此我们可以指定例如 $\text{L}(\tau, q)$，即元素各自携带 q 势能的 τ列表。类似地，我们还可以指定和类型 $\tau_1, q_2 + \tau_2, q_2$，表示左注入除携带类型为 τ_1的载荷外，还会携带 q_1 势能；右注入也同理。最后，函数类型是可能类型的集合 T，其中每个类型的形式均为 $A \to B \odot C$，并受某个谓词 Θ约束；该谓词通常是关于 A、 B、 C中出现的势注释 $q \in \mathbb{Q}_{\geq 0}$的若干线性约束的合取。请注意， B是函数的正常返回类型，而 C 是它可能引发的异常的类型。关于函数类型集合 T 的具体示例，请参阅概述第 2.1 节中 sqdist 的资源多态类型检查。

请注意，在概述第 2.1 节中，我们始终使用更易读的资源标注类型记法。 $L^q(\tau)$ 是 $\text{L}(\tau, q)$ 的简写， $\tau_1^p + \tau_2^q$是 $\tau_1, p\ + \tau_2, q$ 的简写，而 $\tau_1 \to^p_{q \odot q_e} \tau_2 \odot \tau_e$是 $\tau_1, p\ \to \tau_2, q\ \odot \tau_e, q_e$ 的简写。

我们还引入函数 $|\cdot|$，它接受一个资源标注类型，并将正类型中的所有势注释置零。它在图4 中定义，并将有助于定义本文其余部分的各种类型规则。我们也将它扩展到上下文，将 $|\Gamma|$ 定义为 $|\cdot|$ 对 Γ中每个类型的逐点应用。

$\boxed{|\tau| \quad |A|}$ 归零

$|\mathrm{T}| = \mathrm{T}$ |单位类型| =单位类型 $|\tau_1 \times \tau_2| = |\tau_1| \times |\tau_2|$ |空类型| =空类型 $|A_1 + A_2| = |A_1| + |A_2|$

$|\mathrm{L}(A)| = \mathrm{L}(|A|)$ $|\tau, q| = |\tau|, 0$

图 4. 归零资源注释类型

$\boxed{\Gamma; q \vdash v : \tau}$ 值类型判断

T-Var
$$\frac{}{x : \tau; 0 \vdash x : \tau}$$

T-Pair
$$\frac{\Gamma_1; q_1 \vdash v_1 : \tau_1 \qquad \Gamma_2; q_2 \vdash v_2 : \tau_2}{\Gamma_1, \ \Gamma_2; q_1 + q_2 \vdash \text{二元组}(v_1; v_2) : \tau_1 \times \tau_2}$$

T-Inl
$$\frac{\Gamma; p \vdash v : \tau_1}{\Gamma; q_1 + p \vdash \text{左注入}(v) : \tau_1, q_1 + A_2}$$

T-Fun
$$\frac{\Gamma = |\Gamma| \qquad \forall (\tau, q \to B \odot C) \in \mathrm{T} \ . \ \Gamma; f : \mathrm{T} \ x : \tau; q \vdash e : B \odot C}{\Gamma; \ 0 \vdash \mathrm{fun}(f.x.e) : \mathrm{T}}$$

T-Inr
$$\frac{\Gamma; p \vdash v : \tau_2}{\Gamma; q_2 + p \vdash \text{右注入}(v) : A_1 + \tau_2, q_2}$$

T-Unit
$$\frac{}{\cdot; 0 \vdash \ : \text{单位类型}}$$

T-Nil
$$\frac{}{\cdot; 0 \vdash \text{空列表} : \mathrm{L}(A)}$$

T-cons
$$\frac{\Gamma_1; q_1 \vdash v_1 : \tau \qquad \Gamma_2; q_2 \vdash v_2 : \mathrm{L}(A) \qquad A = \tau, p}{\Gamma_1, \ \Gamma_2; \ q_1 + q_2 + p \vdash \mathrm{cons}(v_1; \ v_2) : \mathrm{L}(A)}$$

图 5. 值的类型规则

3.3 值的定型规则

值类型判断 $\Gamma; q \vdash v : \tau$ 表明，在上下文 Γ 下并需要 $q \in \mathbb{Q}_{\geq 0}$ 额外势能时，值 v 具有带注释类型 τ。理解这些规则的一种方式是，Γ 和 q 提供了 $\Phi(v : \tau)$ 势能，即 v 所携带的势能。图 5 包含类型规则，其中上下文 Γ 和势能 q 被线性地处理。我们描述关键规则：

T-Var 所有势能均由上下文 x 提供：τ，且无需额外势能。
T-Pair 上下文 Γ_1, Γ_2 和势能 $q_1 + q_2$ 会被拆分，并且 Γ_i； q_i 用于为 v_i 定型。
T-Inl/T-Inr 势能 $q_i + p$ 会被拆分。上下文 Γ 和势能 p 用于为 v 定型。所需的额外势能 q_i 取决于它属于哪种注入。
T-Cons 上下文 Γ_1, Γ_2 和势能 $q_1 + q_2 + p$ 会被拆分。根据 i， Γ_i； q_i 为列表的表头或表尾定型。需要额外势能 p，因为每个列表元素都必须携带 p 势能。
T-Fun 为函数定型时不会消耗势能。$q = 0$，而前提 $\Gamma = |\Gamma|$ 确保上下文不携带任何势能。这保证了求值函数体时，势能的唯一来源是函数参数。因此，函数是无势的，可以被调用任意次。该规则推导出结果类型 T，其中可能包含无限多个箭头类型 $\tau, q \to B \odot C$；其中，B 是带注释的返回类型，C 是带注释的异常类型，后续计算的类型规则将进一步对此进行说明。每种箭头类型都由以下定型关系所证明：给定参数 $x : \tau$、伴随的常数势能 q、用于递归调用的函数类型集合 T，以及无势闭包 Γ，对函数体 e 进行定型。

对势能进行线性处理，使我们能够确立以下关于闭值的引理，其证明采用对值类型判断的归纳法。

引理 3.1. 若 $\cdot; \ q \vdash v : \ \tau$ 且 $\cdot; \ q' \vdash v : \ \tau$，则 $q = q'$。

由于闭值必须具有唯一的 q 定型，我们可以定义势函数 $\Phi(\_ : \tau)$，将类型为 τ 的值映射为非负有理数，定义为 $\Phi(v : \tau) = q \ \text{if} \ \cdot; q \vdash v : \tau$。

$\boxed{\Gamma; q \vdash e : B \odot C}$计算式类型判定

$$\frac{\Gamma; q \vdash v : \tau}{\Gamma; q + p \vdash \mathsf{val}(v) : \langle \tau, p \rangle \odot C}\ \text{T-Val} \qquad \frac{\Gamma_1; q \vdash e_1 : \langle \tau, q' \rangle \odot C \qquad \Gamma_2, x : \tau; q' \vdash e_2 : B \odot C}{\Gamma_1, \Gamma_2; q \vdash \mathsf{let}(e_1; x.e_2) : B \odot C}\ \text{T-Let}$$

$$\frac{}{\cdot; q + p \vdash \mathsf{tick}\{q\} : \langle \mathsf{unit}, p \rangle \odot C}\ \text{T-Tick}^+ \qquad \frac{}{\cdot; p \vdash \mathsf{tick}\{-q\} : \langle \mathsf{unit}, p + q \rangle \odot C}\ \text{T-Tick}^-$$

$$\frac{\Gamma_1; 0 \vdash v_1 : \mathcal{T} \qquad (\langle \tau, q_1 \rangle \to B \odot C) \in \mathcal{T} \qquad \Gamma_2; q_2 \vdash v_2 : \tau}{\Gamma_1, \Gamma_2; q_1 + q_2 \vdash \mathsf{app}(v_1; v_2) : B \odot C}\ \text{T-App}$$

T-Casevoid
$$\frac{\Gamma; q \vdash v : \mathsf{void}}{\Gamma; q \vdash \mathsf{casevoid}(v) : B \odot C}$$

T-Casepair
$$\frac{\Gamma_1; q_1 \vdash v : \tau_1 \times \tau_2 \qquad \Gamma_2, x_1 : \tau_1, x_2 : \tau_2; q_2 \vdash e : B \odot C}{\Gamma_1, \Gamma_2; q_1 + q_2 \vdash \mathsf{casepair}(v; x_1.x_2.e) : B \odot C}$$

T-Casesum
$$\frac{\Gamma_1; q_1 \vdash v : \tau_1, p_1 + \tau_2, p_2 \qquad \Gamma_2, x_1 : \tau_1; p_1 + q_2 \vdash e_1 : B \odot C \qquad \Gamma_2, x_2 : \tau_2; p_2 + q_2 \vdash e_2 : B \odot C}{\Gamma_1, \Gamma_2; q_1 + q_2 \vdash \text{和类型分支}(v; x_1.e_1; x_2.e_2) : B \odot C}$$

$$\frac{\Gamma_1; q_1 \vdash v : \mathrm{L}(\tau, p)\ \Gamma_2; q_2 \vdash e_1 : B \odot C \qquad \Gamma_2, x : \tau, y : \mathrm{L}(\tau, p); q_2 + p \vdash e_2 : B \odot C}{\Gamma_1, \Gamma_2; q_1 + q_2 \vdash \text{列表分支}(v; e_1; x.y.e_2) : B \odot C}\ \text{T-Caselist}$$

$$\frac{\Gamma; q \vdash v : \sigma}{\Gamma; q + r \vdash \mathsf{raise}(v) : A \odot \langle \sigma, r \rangle}\ \text{T-Raise} \qquad \frac{\Gamma_1; q \vdash e_1 : A \odot \langle \sigma, r \rangle \qquad \Gamma_2, x : \sigma; r \vdash e_2 : A \odot C}{\Gamma_1, \Gamma_2; q \vdash \mathsf{try}(e_1; x.e_2) : A \odot C}\ \text{T-Try}$$

图 6. 计算的定型规则

我们将 q 在 τ 条件下的 v 称为势能。借助 Φ，我们可以陈述一个关于归零函数 $|\cdot|$ 的有用引理；该引理也通过对类型判断进行归纳法证明。它表明，$\tau = |\tau|$ 确保类型为 τ 的值不携带任何势能。我们将这类类型称为无势类型。

引理 3.2. 如果 $\tau = |\tau|$ 和 $\Phi(v : \tau) = q$，那么 $q = 0$。

3.4 计算和异常的类型规则

计算类型判断 $\Gamma; q \vdash e : B \odot C$ 表示在上下文 Γ 下，并要求$q \in \mathbb{Q}_{\geq 0}$ 额外势能时，计算 e 可以求值为带注释类型 B的值和势能，或者抛出带注释类型 C的异常值和势能。图 6 给出了语法导向的计算类型规则，这些规则都将势能视为线性资源。下面介绍一些关键规则：

T-Tick$^+$ 参数为正的计时标记会消耗资源。$q + p$ 输入势能足以支付该计时标记所消耗的 q 势能；该计时标记被定型为带有剩余 p 势能的单位类型。带注释异常类型 C 可以是任意类型。

T-Tick$^-$ 参数为负的计时标记会返还资源。p 输入势能可以是任意值；由于正在返还 q 势能，该计时标记被定型为附带 $p + q$ 势能的单位类型。带注释异常类型 C 可以是任意类型。

T-Val 此规则通过将值作为常量返回，并连同该值传递p 额外势能以及一个任意类型 C 的异常载荷，从而将值的类型判断提升为计算式类型判定。这是合理的，因为值不会引发异常。

T-App 上下文 Γ_1, Γ_2 和势能 $q_1 + q_2$ 都会被拆分。Γ_1 用于为 v_1赋予类型；由于它是一个函数，因此需要 0 额外势能。Γ_2 和 q_2 用于为 v_2（函数参数）赋予类型。

$\boxed{\Gamma; q \vdash e : B \odot C}$计算式类型判定

$$\text{T-ContFun}\ \frac{\Gamma, x_1 : \mathrm{T}\ \ x_2 : \mathrm{T}\ ;\ q \vdash e : B \odot C}{\Gamma,\ \ x : \mathrm{T}\ ; q \vdash [x, x/x_1,\ \ x_2]e : B \odot C} \qquad \text{T-WeakVar}\ \frac{\Gamma; q \vdash e : B \odot C}{\Gamma,\ \ x :\ \tau;\ \ q \vdash e :\ \ B \odot C} \qquad \text{T-WeakPot}\ \frac{\Gamma; q' \vdash e : B \odot C \qquad q \geq q'}{\Gamma; q \vdash e : B \odot C}$$

图 7. 计算的结构类型规则

最后，如果箭头类型 $\tau, q_1 \to B \odot C$属于 T，则所得的带注释结果和异常类型 $B \odot C$ 有效。

T-Let 上下文 Γ_1，Γ_2 被拆分。Γ_1；q 为 e_1定型。e_1 的带注释类型中的势能 q'，连同由 x 增强的 Γ_2：τ，用于为 e_2定型，其类型 B 是整体结果类型。对本文而言最重要的是，e_1 和 e_2 必须对带注释异常类型 C达成一致，这样在任一计算中引发异常时，控制流都会以相同方式转向。

T-Caselist 上下文 Γ_1，Γ_2 和势能 $q_1 + q_2$ 都被拆分。Γ_1 和 q_1 用于将 v 定型为列表 $\mathrm{L}(\tau,\ p)$。空分支 e_1 使用 Γ_2 和 q_2进行定型。cons 分支 e_2 也使用 Γ_2 和 q_2，但还加入了绑定变量 x（表头，类型为 τ）和 y（表尾，类型与原列表相同），以及表头提供的额外势能 p。与 T-Casesum 一样，各分支的最终 $B \odot C$必须一致。

T-Raise 势能 $q + r$被拆分。上下文 Γ 与势能 q 共同用于将 v（即异常载荷）定型为某个 σ。随后，抛出表达式具有带注释异常类型 σ, r。其带注释结果类型 A 可以是任意类型，因为它永远不会正常求值。

T-Try 上下文 Γ_1, Γ_2 被拆分。$\Gamma_1; q$ 用于对客户端表达式 e_1进行定型。该定型的带注释异常类型 σ, r用于对“处理程序”表达式 e_2进行定型；定型时使用由绑定变量 x 增强后的 Γ_2：σ 和势能 r。重要的是，主表达式与处理程序表达式的带注释结果类型 A 必须一致。此外，请注意，处理程序表达式本身也可能抛出异常，而其带注释异常类型 C 就是整体异常类型。

请注意 T-Let 与 T-Try 之间的对称性。在 T-Let 中，正常返回在 e_1和 e_2之间组合，而异常流 $\odot C$ 则在它们之间直接传递。在 T-Try 中，正常返回直接传递e_1 和 e_2，而异常流则在它们之间组合。此外，正如概述部分 2所提到的，全局范围内并未强制要求带注释异常类型一致。相反，只有在抛出异常的点与最近的外围处理程序（或函数）之间才强制执行局部契约。这对于提高资源分析的效率至关重要；同一个异常在不同位置被处理时，其携带的势能无需相同。

3.5 计算的结构化定型规则

到目前为止，所介绍的所有类型规则都是语法导向且严格线性的。这样的类型系统基本上会按照我们的预期运行，但有两个主要问题需要解决。一个显而易见的问题是，完全线性的系统要求程序的所有执行路径都消耗相同数量的资源，无论走的是哪条分支（在列表分支、和类型分支和 try 中）。这与我们在程序中添加计时标记来评估函数调用次数等指标的做法相冲突，因为这些指标通常会因执行路径而异。因此，我们通过添加 T-WeakVar 和 T-WeakPot 规则，使类型系统成为仿射而非线性的，从而允许在任意点丢弃变量和势能。

更微妙的问题是，AARA 函数应当是结构性的。回想一下，在讨论 T-Fun 规则时，前提 $\Gamma = |\Gamma|$ 之所以有必要，只是因为我们预期函数会被多次调用，因此它不能从闭包中捕获势能。我们添加定型规则

T-ContFun，以便结构函数可以被任意复制多次。这对于允许递归函数进行多次递归调用也是必要的。

我们对结构规则的阐述与 AARA 以往的表述之间，一个显著且有意的差异在于不再显式使用共享判断。在 AARA 以往的表述中，共享判断 $\tau \curlyvee (\tau_1, \tau_2)$ 关联三个结构相同、仅势注释不同的带注释类型。带注释类型 τ 的值 v 所携带的势能，等于同一个值 v 分别使用注释 τ_1 和 τ_2 时所携带的势能之和。随后使用该判断来约束保持势能不变的收缩规则：

$$\frac{\tau \curlyvee (\tau_1, \tau_2) \qquad \Gamma, x_1 : \tau_1, x_2 : \tau_2; q \vdash e : B \odot C}{\Gamma, x : \tau; q \vdash [x, x/x_1, x_2]e : B \odot C}\ \textsc{T-Cont}$$

我们认为，T-Cont 在我们的类型系统中是不必要的。一般而言，线性类型系统有一个众所周知的事实：纯正类型的收缩是可容许的，即诸如$\mathsf{unit} + \mathsf{unit} \times \mathsf{unit}$ 之类的类型可以在不修改类型规则的情况下进行复制。类似的事实也适用于我们的类型系统[4] ——纯正类型允许共享。我们唯一需要添加的收缩规则是T-ContFun，它用于复制函数。共享的可容许性在形式上由以下引理表达。

引理 3.3. 若 τ、τ_1、τ_2 是类型，且以下陈述成立：

$$\mathit{If}\ \Phi(v : \tau) = q, \mathit{\ then\ } q = \Phi(v : \tau_1) + \Phi(v : \tau_2).$$

那么我们可以构造 $\curlyvee_{\tau,\tau_1,\tau_2} \colon \tau \to (\tau_1 \times \tau_2)$ $\odot$ 空类型，它对共享进行编码。

该引理可通过对类型 τ 进行归纳法证明。void 异常类型确保 $\curlyvee_{\tau,\tau_1,\tau_2}$ 不会通过引发异常来“取巧”，以达到所需的结果类型——由于类型 void 没有闭值，因此不可能引发异常。我们可以轻松实现编译器遍，将完全结构化（所有类型都支持收缩和弱化）的前端语言转换为我们的语言，并适当地插入对 $\curlyvee_{\tau,\tau_1,\tau_2}$、T-WeakVar 和 T-WeakPot 的调用。这样一来，在以仿射方式管理势能的同时，值便可以重复使用或丢弃。

3.6 计算效应的类型规则

事实证明，异常的定型规则可以自然地扩展，以支持效应及其处理器。我们首先从计算语法中移除 raise 和 try，改为用以下形式替代：

$$e ::= \ldots \mid \mathsf{do}[\ell](v) \mid \mathsf{handler}(e; \{\ell : x.c.e_\ell\}_{\ell \in \Delta}; y.e_r)$$

语法形式 $\mathsf{do}[\ell](v)$ 执行效应 ℓ，并携带载荷 v。语法形式 $\mathsf{handler}(e; \{\ell : x.c.e_\ell\}_{\ell \in \Delta}; y.e_r)$ 是一个效应处理器，用于处理在计算 e 执行期间发生的效应签名 Δ 中的任意效应 ℓ。效应 ℓ 由分支 e_ℓ 处理，该分支可以使用载荷 x，也可以使用续延 c，从效应发生的位置恢复计算 e。如果 e 求值得到一个纯值，则该值会在返回分支 e_r 中替换 y。形式上，效应签名 Δ 的语法如下：

$$\Delta ::= \cdot \mid \Delta, \ell : A \Rightarrow B$$

每个效应 ℓ 都有一个类型规范 $A \Rightarrow B$，表明它需要一个带注释类型为 A 的载荷；如果通过处理器可用的续延恢复执行，则它会被一个带注释类型为 B 的值替代。例如，$\mathsf{inc} : \mathsf{int} \Rightarrow \mathsf{int}$ 可以表示一种效应：它将给定载荷加到某个计数器上；若该效应得到处理，则返回计数器更新后的值。

此外，我们需要修改类型判断，以适应效应而非异常。值类型判断保持不变，但新的计算式类型判定

[4]事实上，这对于先前的 AARA 类型系统也同样成立。

$\boxed{\Gamma; q \vdash e : B \odot \Delta}$ 计算式类型判定

$$\text{T-Linapp}\ \frac{\Gamma_1; p \vdash v_1 : \langle \tau, q_1 \rangle \multimap B \odot \Delta \qquad \Gamma_2; q_2 \vdash v_2 : \tau}{\Gamma_1, \Gamma_2; p + q_1 + q_2 \vdash \mathsf{linapp}(v_1; v_2) : B \odot \Delta} \qquad \text{T-Do}\ \frac{\Gamma; p \vdash v : \tau_1 \qquad \ell : \langle \tau_1, q_1 \rangle \Rightarrow \langle \tau_2, q_2 \rangle \in \Delta}{\Gamma; p + q_1 \vdash \mathsf{do}[\ell](v) : \langle \tau_2, q_2 \rangle \odot \Delta}$$

$$\frac{\begin{array}{c}\Gamma_1; p \vdash e : \langle \tau, q \rangle \odot \Delta \qquad \Gamma_3, y : \tau; q \vdash e_r : C \odot \Delta' \\ \Gamma_2 = |\Gamma_2| \qquad \forall \ell : \langle \tau_1, q_1 \rangle \Rightarrow B \in \Delta.\ (\Gamma_2, x : \tau_1, c : B \multimap C \odot \Delta'; q_1 \vdash e_\ell : C \odot \Delta')\end{array}}{\Gamma_1, \Gamma_2, \Gamma_3; p \vdash \mathsf{handler}(e; \{\ell : x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) : C \odot \Delta'}\ \text{T-Handle}$$

$\boxed{\Gamma; q \vdash v : \tau}$ 值类型判断

$$\frac{\Gamma, x : \tau; p + q \vdash e : B \odot \Delta}{\Gamma; p \vdash \mathsf{linfun}(x.e) : \langle \tau, q \rangle \multimap B \odot \Delta}\ \text{T-LinFun}$$

图 8. 效应处理器和线性函数的新类型规则

为 $\boxed{\Gamma; q \vdash e : B \odot \Delta}$，其中效应签名 Δ 包含计算 e 能够执行的所有效应。幸运的是，之前与效应无关的类型规则都可以保留，只需将所有 C 重命名为 Δ，再将所有带注释异常类型重新解释为效应签名即可。这一变化也会影响函数的定型。函数集合 T 的元素原本具有 $A \to B \odot C$ 这种形式，现在则具有 $A \to B \odot \Delta$ 这种形式。

线性函数。 为了使资源分析易于处理，我们通过将续延定型为线性函数，强制要求效应处理器 $\mathsf{handler}(e; \{\ell : x.c.e_\ell\}_{\ell \in \Delta}; y.e_r)$ 中的续延 c 均为一次性的。这就需要在语言中加入线性函数：

$$\tau ::= \ldots \mid A \multimap B \odot \Delta \qquad v ::= \ldots \mid \mathsf{linfun}(x.e) \qquad e ::= \ldots \mid \mathsf{linapp}(v_1; v_2)$$

加入线性函数类型后，我们扩展势能归零函数 $|\cdot|$（定义见图 4）。有趣的是，正确的做法是让 $|\cdot|$ 成为偏函数，并拒绝将线性函数类型作为输入。这样采取了审慎的做法，确保 $A \multimap B \odot \Delta = |A \multimap B \odot \Delta|$ 永不成立；这意味着线性函数永远不会被捕获到结构函数的闭包中，否则它们可能面临被多次（非线性地）调用的风险。

此外，我们的结构函数类型是资源多态集合，而线性函数类型则始终被定义为单一的资源单态类型。这一限制简化了类型推断，并足以满足大多数用例，包括我们评估中的所有程序。

新的定型规则。线性函数和计算效应的新定型规则如图8所示。我们介绍关键规则：

T-LinFun 线性函数抽象使我们能够定义可调用效应处理器所提供的一次性续延，以及其他线性函数的函数；而结构函数 $\mathsf{fun}(f.x.e)$ 则禁止这样做。这一点体现在其定型规则中：对 Γ 和 p 没有限制，表明它可以从其闭包中捕获势能。

T-Do 与 T-Raise 类似，势能 $p + q_1$ 会被拆分。上下文 Γ 和势能 p 用于将效应载荷 v 定型为 τ_1。τ_1, q_1 是此效应 ℓ 的带注释的输入类型，如效应签名中所指定。关键区别在于，如果外围效应处理器调用其续延，计算就可以恢复。因此，它的结果类型不是任意类型，而是 τ_2, q_2，即效应 ℓ 的带注释的输出类型，如效应签名中所指定。

T-Handle 上下文 Γ_1，Γ_2，Γ_3 被拆分。Γ_1 和 p 用于为 e 定型，即正在被处理的带效应计算。它具有一个效应签名 Δ，该签名与处理程序的各分支相匹配。至于处理程序分支，它们与 T-Try 规则有几个关键区别。

首先，用于为每个效应处理程序分支定型的 Γ_2 受到约束： $\Gamma_2 = |\Gamma_2|$。这是因为效应处理程序分支可以执行任意多次，就像结构函数的函数体一样，因此我们必须禁止它捕获 Γ 中的势能或线性函数。T-Try 没有这一限制，因为异常处理程序至多执行一次。其次，除了效应签名 Δ 所指定的效应输入载荷 $x : \tau_1$ 和势能 q_1之外，我们的处理程序分支还可以使用续延 c。该续延具有线性函数类型，以确保遵守单次使用约束。它的参数类型与效应ℓ的输出类型相匹配，因为 do 的最终输出正是我们传递给续延的参数。续延的结果类型 C 和效应签名 Δ' 与最终结果类型一致，因为续延会重新安装效应处理器。在此上下文中，每个效应处理程序分支 e_ℓ都必须具有类型 C，并且效应签名 Δ'也必须一致。

最后，返回分支 e_r 借助 Γ_3 以及绑定到 y的 e 纯结果进行定型。其结果类型也与最终结果类型 C 和效应签名 Δ'一致。

显式转发。 请注意，我们对效应处理器的定型规则T-Handle强制执行显式效应转发 [68]，要求处理程序必须处理 Δ（即正在被处理的计算 e 的效应签名）中的所有效应。那些实质上并未被处理的效应仍必须被显式处理，但其处理器分支 e_l 会转发该效应，从而将其暴露在 Δ'之中。强制执行这一范式的结果是，每当执行某个效应时，它总是由最近的外层处理程序（如果存在）来处理。

具体来说，被显式转发的效应 ℓ_f的处理器分支可以实现为

$$\{\ell_f : x.c.\mathsf{let}(\mathsf{do}[\ell_f](x); y.\mathsf{linapp}(c; y))\}$$

该处理器分支先转发效应 ℓ_f，再通过把分隔延续 c 应用到效应结果 y上，将该效应的结果转发回正在被处理的计算。

涵盖异常。我们当然希望能够使用效应来实现异常。实际上，可以通过定义一种异常效应来实现，其输入类型为异常类型 C，输出类型为空类型。按照设计，异常处理程序无法从异常引发处恢复执行。重用我们在引理3.3中使用的技巧，由于不存在类型为空类型的闭值，将其设为异常效应的输出类型，便可强制施加这一限制，因为效应处理程序分支不可能构造出参数来调用其续延。形式上，下面给出了异常类型以及raise和try 计算转换到新的效应处理器设定后的形式，其中 ℓ_e 是异常效应标签：

$$C \rightsquigarrow \{\ell_e : C \Rightarrow \langle \mathsf{void}, 0\rangle\}$$
$$\mathsf{raise}(v) \rightsquigarrow \mathsf{let}(\mathsf{do}[\ell_e](v); x.\mathsf{casevoid}(x))$$
$$\mathsf{try}(e_1; x.e_2) \rightsquigarrow \mathsf{handler}(e_1; \{\ell_e : x.\_.e_2\}_{\ell_e \in \{\ell_e\}}; y.y)$$

此外，我们将 try 转换为处理器时，无法完全保持势能，因为 try 允许处理程序使用上下文中的线性变量，而处理器则要求处理器分支只能使用结构变量。之所以有此限制，是因为我们的效应处理器旨在处理可多次处理的可恢复效应，而异常处理程序只会运行一次。为解决这一问题，我们可以为从不调用其续延的效应处理程序分支添加特例，但为简洁起见，此处省略。

3.7 类型推断

针对异常和异常处理程序的新规则经过设计，使得 AARA 现有的类型推断算法可以直接应用于我们的类型系统。类型推断的主要思路是将寻找资源注解的问题归约为现成的线性规划求解。

本文以声明式的方式给出这些规则。为了进行类型推断，我们必须先将规则改写为更具算法性的形式。这包括将结构规则整合到语法导向规则中，并为函数类型集合确定有限表示。后者可以通过用一组线性约束表示类型来实现。新规则并未给设计算法化类型规则带来额外挑战。

使用算法化类型规则，AARA 的类型推断包含以下步骤。

(1) 对结构类型执行 Hindley–Milner 类型推断。(2) 在类型推导中添加表示资源注解的未知变量。(3) 根据算法化类型规则生成线性约束。(4) 使用线性规划求解器求解这些约束，并使初始势能最小化。

这种策略适用于函数和闭表达式。对于函数，初始势能就是参数的势能。有关类型推断的更多细节，请参阅文献 [55, 58, 60]。

4 通过 K 机器证明可靠性

本节给出了新颖的可靠性证明，通过进展性和保持性，借助 K 机器（一种基于栈的抽象机）确立导出界的正确性。

4.1 K-Machine 代价语义

为了定义我们语言的代价语义，我们引入一种称为 K-Machine [49] 的抽象机。K-Machine 包含一个显式控制栈，用于记录程序求值过程中执行一步后仍待完成的工作。

栈 k 是由栈帧组成的列表，每个栈帧要么是对计算进行排序的绑定子 $x.e$，要么是等待其计算结果的效应处理程序 $\mathsf{handler}(\_; \{\ell\colon x.c.e_\ell\}_{\ell\in\Delta};\ y.e_r)$。只有两种栈帧，这得益于值与计算的模态分离。程序求值在具有以下几种机器状态的转移系统中进行：

- $k \triangleright e$。这表示正在栈 栈 k 上求值一个闭表达式 e。
- $k \triangleleft v$。这表示闭值 v 正被返回到栈 k。
- $k \blacktriangleleft (\ell,\ v,\ k_r)$。这表示效应 ℓ 正被传播到栈 k，并携带载荷 v 以及分隔延续 k_r 的累加器。

完整语法如下。

Frame f	::=	$x.e$	sequence
	|	$\mathsf{handler}(\_; \{\ell : x.c.e_\ell\}_{\ell\in\Delta}; y.e_r)\Delta$	effect handler
Stack k	::=	ϵ	empty stack
	|	$k; f$	frame sequence
State s	::=	$k \triangleright e$	evaluate expression
	|	$k \triangleleft v$	return value
	|	$k \blacktriangleleft (\ell, v, k')$	propagate effect

我们还为分隔延续引入一种新值：$\mathsf{dcont}(k)$。它是一种动态值，也就是说，它不会出现在我们语言的源程序中，而是在下文将定义的动态语义步骤中产生。它将控制栈 k 具体化为线性函数类型的分隔延续。当我们将某个实参值 v 应用于它时，状态会转移到 $k \triangleleft v$，并返回该步骤最终得到的值。

现在，我们可以通过机器状态之间的转移来定义语言的动态语义。初始状态是在空栈上对表达式求值的状态：$\epsilon \triangleright e; q$。最终状态要么将值返回到空栈 $\epsilon \triangleleft v; q$，要么将未处理效应传播到空栈

$\boxed{s; q \mapsto s'; q'}$ 状态转换

D-let
$$\frac{}{k \triangleright \mathsf{let}(e_1;\ x.e_2);\ q \mapsto k;\ x.e_2 \triangleright e_1;\ q}$$

D-seq
$$\frac{}{k;\ \ x.e \triangleleft v;\ \ q \mapsto k \triangleright [v/x]e;\ \ q}$$

D-tick
$$\frac{p \geq q}{k \triangleright \text{计时标记}\{q\}; p \mapsto k \triangleleft;\ p - q}$$

D-Ret
$$\frac{}{k \triangleright \mathsf{val}(v);\ q \mapsto k \triangleleft v; q}$$

D-Cons
$$\frac{}{k \triangleright \text{列表分支}(\mathsf{cons}(v_1;\ v_2);\ e_0;\ x.y.e_1);\ q \mapsto k \triangleright [v_1, v_2/x,\ \ y]e_1;\ q}$$

D-Dcont
$$\frac{}{k \triangleright \text{线性应用}(\text{定界续延}(k');\ v);\ q \mapsto k @\ k' \triangleleft v; q}$$

D-Capture
$$\frac{}{k;\ \ x.e \blacktriangleleft (\ell,\ \ v,\ \ k');\ \ q \mapsto k \blacktriangleleft (\ell,\ \ v,\ \ x.e @\ k');\ \ q}$$

D-Do
$$\frac{}{k \triangleright \text{执行}[\ell](v);\ q \mapsto k \blacktriangleleft (\ell,\ \ v,\ \ \epsilon);\ q}$$

D-normal
$$\frac{}{k; \text{处理程序}(\_; \{\ell:\ x.c.e_\ell\}_{\ell \in \Delta};\ y.e_r)\ \triangleleft v;\ q \mapsto k \triangleright [v/y]e_r;\ q}$$

D-handle
$$\frac{\ell' \in \Delta \qquad d = \text{定界续延}(\text{处理程序}(\_; \{\ell:\ x.c.e_\ell\}_{\ell \in \Delta};\ y.e_r) @\ k')}{k; \text{处理程序}(\_; \{\ell:\ x.c.e_\ell\}_{\ell \in \Delta};\ y.e_r)\Delta \blacktriangleleft (\ell', v, k');\ q \mapsto k \triangleright [v, d/x, c]e_{\ell'};\ q}$$

图9. 选择状态转移

$\epsilon \blacktriangleleft (\ell,\ \ v,\ \ k);\ \ q$。图9列出了一些用于转移判断的规则（完整规则集见附录B）：$s;\ q \mapsto s';\ q'$，它表示机器状态 s 在给定 $q \in \mathbb{Q}_{\geq 0}$ 资源的情况下，转移到机器状态 s' 及 $q' \in \mathbb{Q}_{\geq 0}$ 剩余资源。该判断使用@ 运算符拼接两个栈，其定义如下，基于第二个栈的结构进行归纳：

$$k @ \epsilon := k \qquad\qquad k @ k'; f := (k @ k'); f$$

这些规则大多是标准规则，机器状态中的资源部分 q 除 D-tick 规则外不会被修改。下面讨论该规则，以及其他关键规则的作用。

D-Tick 要求提供足够的资源 p，以满足 tick$\{q\}$ 中的资源需求 q。因此，结果状态中还剩下 $p - q$ 资源。

D-Do 执行一个效应，其作用与前面介绍的用于处理异常的 D-raise 类似。关键区别在于，它返回的不仅是一个负载值 v，还包括效应标签 ℓ，以及一个空的分隔延续 $\cdot$；随着效应向下传播，该延续会捕获更多栈帧。

D-Capture 展示了传播效应的状态 $k \blacktriangleleft (\ell, v, k')$ 如何捕获连续栈帧，并在效应沿栈向下传播时将它们累积到 k' 中。

D-Handle 处理正在传播的效应。我们的类型系统确保效应标签 ℓ 包含在 Δ 处理器栈帧的签名中。处理该效应意味着我们从传播效应切换到对处理器主体 $e_{\ell'}$ 求值。在此过程中，效应负载 v 会替换 x，更重要的是，分隔延续值会替换 c。关键的是，这个分隔延续是 k'，它由从处理器栈帧（含该帧）开始，直到执行效应的点为止的所有栈帧组成。这种“重新安装”清楚地表明，我们使用的是深处理器。

D-Dcont 表明，在进行线性函数应用时，将分隔延续 dcont(k') 应用于值 v 时，我们只需将其中的栈 k' 追加到当前栈 k，并返回 v 到该栈。最终，当栈的 k' 部分运行结束后，它返回的任何值都将直接返回到原始栈 k，以继续执行。

$\boxed{A_1 \odot \Delta_1 \blacktriangleleft: \ k \blacktriangleleft: \ A_2 \odot \Delta_2}$ 栈定型

$$\frac{}{B \odot \Delta \blacktriangleleft: \epsilon \blacktriangleleft: B \odot \Delta}\ \text{K-Emp} \qquad \frac{A \odot \Delta_1 \blacktriangleleft: k \blacktriangleleft: B \odot \Delta_2 \qquad x : \tau; q \vdash e : B \odot \Delta_2}{A \odot \Delta_1 \blacktriangleleft: k; x.e \blacktriangleleft: \langle \tau, q \rangle \odot \Delta_2}\ \text{K-Bnd}$$

$$\frac{\begin{array}{c} A \odot \Delta_1 \blacktriangleleft: k \blacktriangleleft: B \odot \Delta_3 \qquad y : \tau; q \vdash e_r : B \odot \Delta_3 \\ \forall \ell : \langle \tau_1, q_1 \rangle \Rightarrow C \in \Delta_2.\ x : \tau_1, c : C \multimap B \odot \Delta_3; q_1 \vdash e_\ell : B \odot \Delta_3 \end{array}}{A \odot \Delta_1 \blacktriangleleft: k; \mathsf{handler}(\_; \{\ell : x.c.e_\ell\}_{\ell \in \Delta_2}; y.e_r) \blacktriangleleft: \langle \tau, q \rangle \odot \Delta_2}\ \text{K-Handler}$$

图 10. 栈的类型规则

4.2 类型可靠性

现在，我们通过关于抽象栈机的进展与保持定理，给出我们语言的类型可靠性。这种证明技术是新颖的，也是本文的主要贡献之一。我们从若干引理和附加判断开始。

替换引理。 类型可靠性的一项关键内容是以下替换引理。它指出，如果在一个良类型的值（或计算）中，用值 v_1 替换变量 $x_1 : \tau_1$，那么替换后得到的值（或计算）仍然可以进行类型判断，但必须计入该值的势能$q_1 = \Phi(v_1 : \tau_1)$。

引理 4.1（替换）。若 $\cdot; q_1 \vdash v_1 : \tau_1$:
(1) 若 $\Gamma, x_1 : \tau_1; q_2 \vdash v : \tau$，则 $\Gamma; q_1 + q_2 \vdash [v_1/x_1]v : \tau$。*(2)* 若 $\Gamma, x_1 : \tau_1; q_2 \vdash e : B \odot \Delta$，则 $\Gamma; q_1 + q_2 \vdash [v_1/x_1]e : B \odot \Delta$。

该引理分别通过对判断 $\Gamma, x_1 : \tau_1; q_2 \vdash v : \tau$和 $\Gamma, x_1 : \tau_1; q_2 \vdash e : B \odot \Delta$进行归纳法证明。证明见附录 C。

栈和状态的类型判断。图 10 包含栈类型判断的规则$A_1 \odot \Delta_1 \blacktriangleleft: k \blacktriangleleft: A_2 \odot \Delta_2$，该判断指出，当栈 k接受带注释类型 A_2的值或签名 Δ_2中的效应，并返回带注释类型 A_1的值或签名 Δ_1中的效应时，该栈是良构的。栈类型判断最终使我们能够定义$\mathsf{dcont}(k)$的定型规则：

$$\frac{A_1 \odot \Delta_1 \blacktriangleleft: k \blacktriangleleft: A_2 \odot \Delta_2}{\Gamma; 0 \vdash \mathsf{dcont}(k) : A_2 \multimap A_1 \odot \Delta_1}\ \text{T-Dcont}$$

也就是说，由栈 k 编码的分隔延续是一个线性函数，它接受类型 A_2（即 k所接受的类型），并返回类型 A_1，其效应签名为 Δ_1（即 k 所输出的签名）。

图 11 包含判断 $q \vdash s$ 的规则，这些规则规定，给定 q 势能时，执行状态 s是良构的。

栈类型判断使我们能够陈述以下引理：只有当第二个栈的输出类型和效应签名与第一个栈的输入类型和效应签名一致时，才能将两个栈追加起来。根据 @ 的定义，可以通过对第一个栈的结构进行归纳来证明该引理。

引理 4.2（栈追加类型判断）。 $B_2 \odot \Delta_2 \blacktriangleleft: k_1 @ k_2 \blacktriangleleft: B_1 \odot \Delta_1$ 当且仅当存在 B, Δ，使得 $B_2 \odot \Delta_2 \blacktriangleleft: k_1 \blacktriangleleft: B \odot \Delta$并且 $B \odot \Delta \blacktriangleleft: k_2 \blacktriangleleft: B_1 \odot \Delta_1$

类型可靠性。定义了执行状态的良构性后，我们终于可以陈述可靠性定理：保持性和进展。

$\boxed{q \vdash s}$ 状态类型检查

$$\frac{\cdot; q \vdash e : B \odot \Delta \qquad \_ \triangleleft : k \triangleleft : B \odot \Delta}{q \vdash k \triangleright e}\ \text{ST-Exp} \qquad \frac{\cdot; q_1 \vdash v : \tau \qquad \_ \triangleleft : k \triangleleft : \langle \tau, q_2 \rangle \odot \Delta}{q_1 + q_2 \vdash k \triangleleft v}\ \text{ST-Val}$$

$$\frac{\cdot; q_1 \vdash v : \tau \qquad \ell : \langle \tau, p \rangle \Rightarrow C \in \Delta' \qquad \_ \triangleleft : k \triangleleft : B \odot \Delta' \qquad B \odot \Delta' \triangleleft : k' \triangleleft : C \odot \Delta''}{q_1 + p \vdash k \blacktriangleleft (\ell, v, k')}\ \text{ST-Eff}$$

图 11. 执行状态的类型规则

定理 4.3（保持性）。若 $q \vdash s$ 且 $s; q \mapsto s_0; q_0$ 则 $q_0 \vdash s_0$。

证明。 保持性通过对状态转移判断进行分类讨论来证明 $s; q \mapsto s_0; q_0$。我们展示一个说明正常计算的案例，以及一个说明效应处理的案例。完整证明见附录C。

D-Cons $k \triangleright$列表分支(cons(v_1; v_2); e_0; $x.y.e_1$); $q \mapsto k \triangleright [v_1, v_2/x, y]e_1$; q

根据假设， $_ \triangleleft : k \triangleleft : B \odot \Delta, \cdot; q \vdash$列表分支(cons($v_1$; v_2); e_0; $x.y.e_1$) : $B \odot \Delta$。只需证明 $\cdot; q \vdash [v_1, v_2/x, y]e_1 : B \odot \Delta$。对 $\cdot; q \vdash$列表分支(cons(v_1; v_2); e_0; $x.y.e_1$) : $B \odot \Delta$ 作归纳。只有 T-caselist 和 T-WeakPot 两种情况。后者略去。

根据 T-caselist 的前提， $x : \tau, y : \mathrm{L}(\tau, p); q_2 + p \vdash e_0 : B \odot \Delta$， 其中 $\cdot; q_1 \vdash \mathrm{cons}(v_1; v_2) : \mathrm{L}(\tau, p)$， $q = q_1 + q_2$。对 $\cdot; q_1 \vdash \mathrm{cons}(v_1; v_2) : \mathrm{L}(\tau, p)$ 反演，得到 $q_1 = q_1' + q_2' + p, \cdot; q_1' \vdash v_1 : \tau$，
$\cdot; q_2' \vdash v_2 : \mathrm{L}(\tau, p)$. 接着应用替换引理 C.4， $\cdot; q_2 + p + q_1' + q_2' \vdash [v_1, v_2/x, y]e_1 : B \odot \Delta$， 即 $\cdot; q \vdash [v_1, v_2/x, y]e_1 : B \odot \Delta$。

D-Handle k; 处理程序(_; {ℓ: $x.c.e_\ell$}$_{\ell \in \Delta}$; $y.e_r$) $\blacktriangleleft(\ell', v, k')$; $q \mapsto k \triangleright [v, d/x, c]e_{\ell'}$; q当 $\ell \in \Delta$ 且 $d =$为 dcont(处理程序(_; {ℓ: $x.c.e_\ell$}$_{\ell \in \Delta}$; $y.e_r$) @ k')。

根据左侧 ST-Eff 的前提，我们有 (1) $q = q_1 + p$， (2) $\cdot; q_1 \vdash v : \tau$， (3) $\ell' : \tau, p \Rightarrow C \in \Delta$， (4) $\_ \triangleleft : k$; 处理程序(_; {ℓ: $x.c.e_\ell$}$_{\ell \in \Delta}$; $y.e_r$) $\triangleleft : B' \odot \Delta$， 以及 (5) $B' \odot \Delta \triangleleft : k' \triangleleft : C \odot \Delta''$。利用 K-Handler 对 (4) 进行反演，得到 (6) $\_ \triangleleft : k \triangleleft : B \odot \Delta_3$， (7) 对所有 $\ell : \tau_1, q_1 \Rightarrow C$，若在 Δ 中，则 $x : \tau_1, c : C \multimap B \odot \Delta_3; q_1 \vdash e_\ell : B \odot \Delta_3$， 以及 (8) $y : \tau'; q' \vdash e_r : B \odot \Delta_3$， 使得 $B' = \tau', q'$。接下来，我们使用 K-Emp 、 K-Handler 、 (7) 、 (8) 和引理 C.6 构造栈类型 $B \odot \Delta_3 \triangleleft :$ 处理程序 (_; {ℓ: $x.c.e_\ell$}$_{\ell \in \Delta}$; $y.e_r$) @ $k' \triangleleft : C \odot \Delta''$。然后根据 T-Dcont，有 (9) $\cdot; 0 \vdash d : C \multimap B \odot \Delta_3$。

现在，我们可以在 e_ℓ 中分别对 x 和 c 应用替换引理。对于 $[v/x]$，我们有 (2)。对于 $[\mathrm{dcont}(\ldots)/c]$，我们有 (9)。对于 e_ℓ，我们有 (7)。综合以上结果，我们得到

$$\cdot; p + q_1 + 0 \vdash [v, d/x, c]e' : B \odot \Delta_3$$

最后，利用 (1) 中的 $q = q_1 + p$， 以及 (6) 中的 $\_ \triangleleft : k \triangleleft : B \odot \Delta_3$， 我们可以应用 ST-Exp 对 $\mapsto$ 的整个右侧进行定型。□

定理 4.4（进展）。如果 $q \vdash s$， 那么要么是 s*final*， 要么是 s； $q \mapsto s'$； q'。

证明。进展性通过对状态类型判断 $q \vdash s$进行分类讨论来证明。我们给出传播效应状态的情形。完整证明见附录 C。

ST-Eff $q \vdash k \blacktriangleleft (\ell', v, k')$

对 k 的结构进行归纳。

- $k = \epsilon$: $k \blacktriangleleft (\ell, v, k)$ 是最终的
- $k = k_1$; $x.e$: 根据 D-Capture， k_1; $x.e \blacktriangleleft (\ell, v, k)$; $q \mapsto k_1 \blacktriangleleft (\ell, v, x.e @ k)$; q
- $k = k_1$; handler(_; {ℓ: $x.c.e_\ell$}$_{\ell \in \Delta}$; $y.e_r$): 我们的类型系统保证 $\ell' \in \Delta$。于是，根据 D-handle， k_1; handler(_; {ℓ: $x.c.e_\ell$}$_{\ell \in \Delta}$; $y.e_r$) $\blacktriangleleft (\ell' v, k')$; $q \mapsto k_1 \triangleright [v, d/x, c]e_{\ell'}$; q其中$d =$dcont(handler(_; {ℓ: $x.c.e_\ell$}$_{\ell \in \Delta}$; $y.e_r$) @ k') □

5 评估

本节首先介绍我们如何实现支持异常和效应的 AARA。随后，我们使用 21 个有代表性的基准程序评估该实现的准确性和性能，并将其与先前的 AARA 实现进行比较。我们的实现的源代码和这 21 个基准程序均可在相关制品中获取 [25]。

5.1 实现

我们对含异常与效应的 AARA 的实现建立在一个新颖的框架之上，该框架以 RaML 为蓝本 [53]，并使用 OCaml 实现。本文将其称为 RaML+ 。RaML+ 具有 SML 前端，并会调用 MLton SML 编译器执行一些命令行操作 [105]。因此，本文中的所有基准测试和代码示例均使用 SML 实现。正如第3节所述，RaML+ 中实现的分析比本文的形式化推导更为完整，并支持许多 SML 特性，包括积类型、和类型和递归类型。我们的类型论和 RaML+不支持多态、嵌套模式匹配和模块等 SML 特性；相反，我们依靠 MLton 编译器来消除这些特性，例如通过 MLton 的单态化和模式展平过程。

RaML+ 中针对异常与效应的新类型规则的实现，源自第3.7节（见第3节）所述类型规则的算法版本。这些类型规则会生成势注释之间的线性约束，从而将整个程序的类型推断归约为一个线性优化问题。随后使用 Coin-Or 线性规划求解器 [99]求解该优化问题。

异常前端。SML 原生支持通过 **raise** ExName 引发异常，并通过 e **handle** e’ 处理异常。在为 RaML+ 编写程序时，我们直接使用这些语法形式。随后，为了向 RaML+ 添加异常支持，我们只需将 SML 异常和处理器翻译为中间表示中对应的抽象语法树变体。

由于 MLton 执行全程序编译，我们将所有异常合并为一个大型和类型，该类型作为所有异常的载荷。例如，图 1 中的运行示例声明了两个列表长度不匹配异常，将使用 **数据类型**exn = Emis1**of** 单位类型 | Emis2 **of**单位类型 来表示异常。如第 3.6 节所述，另一种选择是将异常翻译为效应。然而，由于效应处理器分支不得使用线性变量的限制，我们保留了这两种语法形式。

效应前端。 添加效应支持更具挑战性，因为 SML 没有对效应及其处理器的原生支持。用户在数据类型中定义他们希望执行的效应。然后，他们可以使用我们提供的名为 e_do 的 SML 函数来执行效应，并使用 e_handle 来处理效应。这些函数会被翻译为中间表示中显式的 do 和处理程序抽象语法树变体。

成本度量。 如概述中所述，本文选择的成本度量是函数调用和处理器求值的次数。为简化评估，我们在 RaML+ 中添加了命令行标志 -capps 和 -chandles，它们会在运行主分析遍之前，分别在函数调用和处理器旁插入 (R.tick 1)。这样一来，我们就无需在基准程序中写入 R.tick 表达式，并能尽量减少基准测试评估方式不一致的情况。

5.2 目标与方法

我们的评估旨在确定新分析方法的效率、导出界的质量，以及相较于 RaML 的开销。RaML 是一种支持引发异常、但不支持处理异常的 AARA 实现。

为此，我们构建了 21 个基准程序，通过引发和处理异常与效应来产生非平凡的（非局部）控制流。基于这些基准程序，我们

	RaML		RaML+ 与异常		
函数	导出界	时间（秒）	导出界	紧吗?	时间（秒）
sqdist(v1 : L(ℝ), v2 : L(ℝ)) : ℝ	|v2|	0.05	|v1|	✓	0.007
distances_1(vs : L(L(ℝ)), p : L(ℝ)) : L(ℝ)	$1 + 3\|$与$\| + \sum_i \|v_i\|$	0.07	$1 + 3\|$与$\| + \sum_i \|v_i\|$	✓	0.02
distances_2(vs : L(L(ℝ)), p : L(ℝ)) : L(ℝ)	无法求解	–	$1 + 3\|$对阵$\| + \sum_i \|v_i\|$	✓	0.03
最近距离函数(vs : L(L(ℝ)), p : L(ℝ)) : (ℝ, L(L(ℝ)))	无法求解	–	$1 + 3\|$对$\| + \sum_i \|v_i\|$	✓	0.12
zip函数(l1 : L(a), l2 : L(b)) : L((a, b))	无法求解	–	$1 + 2\|$‖2$\|$	✓	0.01
map2(f : (a, b) → c, l1 : L(a), l2 : L(b)) : c	无法求解	–	$1 + 4\|$‖2$\|$	✓	0.01
eval_exp(e : 表达式) : ℤ	无法求解	–	$1 + 2c + 2p + t$	✓	0.13
sort(l : L(ℤ)) : L(ℤ)	无法求解	–	$\|$‖$\| + \binom{\|l\|}{2}$	✓[5]	0.07
zip'(l1 : L(a), l2 : L(b)) : L((a, b))	|l1|	0.03	|l1|	✓	0.01
map2'(f : (a, b) → c, l1 : L(a), l2 : L(b)) : c	$2\|$‖1$\|$	0.03	$2\|$‖1$\|$	✓	0.01
求值_表达式'(e : 表达式) : ℤ	$2p + 2t$	0.04	$c + p + t$	✓	0.04
sort'(l : L(ℤ)) : L(ℤ)	$\binom{\|l\|}{2}$	0.09	$\binom{\|l\|}{2}$	✓	0.06

表 1. RaML+带异常的评估结果

使用支持异常/效应的 RaML+ 自动导出界，并将其与手动导出的界进行比较，以确定其紧致性。除两个基准程序外，其余基准程序所选用的成本度量均衡量函数调用和处理器求值的次数。此外，我们还在 6 个不含异常处理器的示例上测试了实现，以确定 RaML+ 在分析与 RaML 相同的程序时是否会产生显著的性能开销。由于将 AARA 扩展到异常的方式是保守的，我们预计这些基准程序的导出界不会与 RaML 的界有所不同（实验也证实了这一点）。

对于每个基准程序，我们手工分析这些基准测试，以判断 RaML+ 自动得到的导出界是否为紧界。此外，对于异常基准，我们还测量 RaML+ 执行分析所需的时间。我们对每个基准程序运行 10 次试验，并报告其平均耗时。这些实验在一台配备 AMD Ryzen 9 7940HS 处理器和 64GB RAM 的机器上运行。

5.3 异常基准

表 1 包含异常基准的评估结果。我们评估的第一组函数是概述第 2.2节中介绍的运行示例。其中距离 1 和 距离 2 的界与概述中导出的界不同，因为我们现在统计_ _所有函数调用，而不是仅对 sqdist 的调用。在第二组中，我们评估同时包含异常和处理器的其他函数：

最近距离函数 函数 最近距离函数 基于第2.2 节中的示例，并返回 vs 中任意向量与向量 p之间的最短距离，以及 vs 中满足该距离的子集。一个有趣的方面是，最近距离函数 会利用带有效载荷的异常，在最短距离为 0 时提高效率。

zip zip函数会组合两个列表。然而，如果两个列表长度不同，就会抛出异常 UnequalLengths，其载荷为部分压缩结果，即一个 ('a * 'b) 列表。该函数需要反复处理并重新抛出此异常，同时扩大载荷。

map2 类似地，函数 map2 会将一个接收两个输入的函数分别应用于两个列表；如果它们长度不同，同样会抛出该异常。

eval_exp 这是一个简单表达式数据类型的求值器。它使用异常和处理程序，在遇到 0 时跳过乘法运算。

sort 函数 sort 会对输入的整数列表进行一次线性遍历，以检查列表是否已排序。如果尚未排序，就会抛出异常，外围的处理程序则调用快速排序对列表进行排序。

[5]此界限是 ti 这是因为我们使用了快速排序，其最坏情况下的时间复杂度为二次。RaML+ 无法推导出对数界。

函数	RaML+ 与效应导出界	紧确吗?
简单_输入输出(() : 单位类型) : 单位类型	11	✓
存储_列表(vs : L(L(ℤ))) : 单位类型	$6 + 9\lvert\text{与}\rvert + \sum_i \lvert v_i\rvert$	✓
存储_列表_q_naive(vs : L(L(ℤ))) : 单位类型	无法求解	–
存储_列表_q(vs : L(L(ℤ))) : 单位类型	$8 + 13\lvert\text{对比}\rvert + \sum_i \lvert\text{版本}_i\rvert$	✓
存储_后缀_列表(vs : L(ℤ)) : 单位类型	$14 + 10\lvert vs\rvert + \binom{\lvert vs\rvert}{2}$	✓
生成器_到_列表(n : nat) : L(nat)	$1 + 3\lvert n\rvert$	✓
生成器_到_列表_2(n : nat) : L(nat)	$1 + 5\lvert n\rvert + \binom{\lvert n\rvert}{2}$	✓
生成器_计数器(n : nat) : 单位类型	$3 + 11\lvert n\rvert + \binom{\lvert n\rvert}{2}$	✓
前缀_和_列表(ns : L(nat)) : L(nat)	$2 + 12\lvert ns\rvert + \sum_i \lvert ns_i\rvert$	✓

表 2. RaML+效应求值结果

最后一组基准测试只是上一组中的函数，只不过移除了它们的异常处理程序。例如，eval_exp’ 不再使用异常来跳过乘以 0 的运算，而 sort’ 调用快速排序时，不会先执行 sort 中的有序性检查。

5.4 效应基准测试

效应处理器的基准程序列于表2中。由于我们只评估带效应的 RaML+，而不是原始的 RaML（后者无法分析带效应的程序），因此没有测量该基准程序的分析时间，因为没有可供比较的对象。第一个基准程序 simple_IO 执行 3 个输入输出效应。第二组包含使用与图Insert 和 Remove 中概述部分的运行示例相同的效应的函数3：

store_lists_q_naive 是 store_lists（概述示例）的一个版本，只是对处理程序进行了修改。store_lists 中的效应处理器以栈的方式存储和移除元素，而 store_lists_q_naive 则实现了队列。正如名称中的 naive 所示，该队列被朴素地实现为单个列表，Insert 或 Remove 都必须遍历该列表，因此产生与存储大小成正比的代价。这一代价无法在效应签名中捕获，因此该示例有意设计为无法分析。

存储_列表_队列 是一种高效的存储_列表_队列_朴素实现。它采用标准的双列表函数式队列实现，Insert 和 Remove 操作的摊还代价均为常数，现在可以为其赋予类型。

存储_后缀_列表 是对原始存储_列表的一种变体。它的参数 vs 不再是由整数列表组成的列表，而只是一个整数列表。我们将此列表的所有后缀存入存储_。然后，我们从存储中移除所有后缀，并逐一遍历，就像在存储_列表中一样。

最后，我们评估四个基准测试，它们使用文献中生成器所提供的 Yield : nat $\Rightarrow$ 单位类型效应。它们都依赖于数据类型 nat = Zero | Succ of nat。这种表示方式使我们能够将与所表示数字成正比的势能存储在 nat 中。

生成器_到_列表 接收一个自然数 n，创建一个生成器，使其对从 n 到 0 的每个自然数都执行 Yield，然后使用处理程序将生成的数字收集到一个列表中。

生成器_到_列表_2 与生成器_到_列表相似，但它会对输出列表中的每个元素调用 traverse（针对自然数）。与存储_后缀_列表类似，由于某些原因，它也需要二次势能。

生成器_计数器 创建与前两个基准测试相同的生成器，然后使用来自状态的 Get 和 Put 效应，将生成的数字累加起来。这表明 RaML+ with Effects 可以分析组合效应处理器的代码。

前缀_和_列表依赖于一个前缀_和_生成器，它使用状态效应 Get 和 Put 跟踪列表的前缀和，并在每个前缀处执行 Yield 效应。另一方面，前缀_和_列表会将生成的数字收集回一个列表。它展示了 RaML+ 与

效应可以分析同时执行多种效应（状态与生成器效应）的代码，以及将 Yield 效应显式转发给外层处理程序的代码。

5.5 结果

我们的评估结果汇总于表 1 和表 2中。对于每个基准程序，表中列出了函数名称及其类型、RaML 导出的界+及其紧致性。对于异常基准，我们还提供了 RaML 导出的界，以及 RaML 和 RaML+ 的分析时间（见 时间（秒）列）。如 是否紧致？列中的勾选标记所示，所有导出界不仅渐近紧致，而且常数因子也都是最优的。

RaML 和 RaML+ 导出的界之间存在一些微不足道的差异。对于 sqdist，RaML 导出 $|v2|$，而 RaML+ 导出 $|v1|$。这些界有所不同，但正如概述第 2.1 节所解释的那样，它们都是真实代价 $\min(|v1|, |v2|)$ 的有效且紧致的上界，而我们的类型系统无法直接编码这个上界。在函数 eval_exp 和 eval_exp' 的界中，c、p 和 t 分别表示 Const、Plus 和 Times 节点的数量。对于 eval_exp'，RaML 导出 $2p + 2t$，而 RaML+ 导出 $c + p + t$。乍看之下，这些界似乎相差很大，但实际上它们几乎相等，因为二叉树中的叶节点数量 c 总是等于内部节点数量 $p + t$ 加一。我们不完全确定这两个工具为何会报告不同的界。

我们的发现是，带有异常和效应的 RaML+ 在我们的基准测试上表现优于 RaML。它能够分析更大范围的会 raise 并处理异常和效应的用户函数，而 RaML 只能分析会 raise 异常的函数。此外，通过观察两种实现都能分析的函数所需的分析时间，我们发现，尽管增加了复杂度，我们的实现并未增加开销。事实上，我们的实现始终更快，不过这可以归因于 RaML 会跟踪多元势函数（例如 $|l1||l2|$），而 RaML+ 并不支持这一点。

6 相关工作

自动资源界限分析已从许多不同角度得到研究，但关于非局部控制转移的工作却很少。

递推关系。 自动资源界限分析的一种传统方法最早由 Wegbreit [106], 研究，它是一个由提取再求解递推关系组成的两步过程。该方法已针对函数式程序 [28, 31, 42],逻辑程序 [36, 73],以及 COSTA 项目 [2–4, 7, 8] 中的命令式语言和面向对象语言进行了研究，并使用了抽象解释和符号分析 [39, 70, 71]等技术。

唯一涵盖异常处理器的系统是 COSTA [4, 5]。该分析针对 Java 字节码，可推导出函数的界限，适用于包含异常处理程序、且会捕获在同一函数调用中引发的异常的函数。其思路是将引发异常的代码转换为跳转到处理程序的指令。相比之下，本文关注的是非局部控制转移，即引发异常可能会将控制权转移到另一个函数。

项重写。 最重要的是，项重写系统分析主要关注终止性 [14,41]。不过，资源上界分析也已有研究。两个颇具代表性的实现是 AProVE [19, 40, 88] 和 TCT [12, 51]。可以在项重写中用非确定性对异常进行建模，但这种抽象能否得出令人满意的资源上界尚不明确。使用类型（如本文所采用的方法）是建立处理异常与引发异常之间模块化接口的自然方式。

其他静态分析。 其他针对命令式程序的自动资源分析基于循环分析和插桩 [16, 45–47], 排序函数 [23, 108], 以及类似技术 [95]。与本文不同的是，这些工作无法处理异常处理器所需的非局部控制转移。

通过抽象机证明可靠性。 我们的工作通过考察基于栈的抽象机的语义，首次证明了带效应语言中 AARA 的可靠性；而在更广泛的效应处理器社区中，这种方法已经较为成熟 [82, 92]。与本文最为相关的是 Voigt 等人的工作 [102]，该工作展示了一种效应系统，可以确保资源（如文件句柄）得到安全的获取与释放。

自动摊销资源分析。 与本文最密切相关的是关于 AARA 的研究。Hofmann 和 Jost [60] 提出了这项技术，用于推导一阶函数式程序的线性堆空间界。在过去二十年中，AARA 已扩展到许多编程语言特性 [44, 57, 63, 80], 、非线性界 [54, 61, 66],，以及不追求完全自动化的更高级类型系统 [75, 90]。详情可参阅 Jost 和 Hoffmann 的综述 [58]。许多关于 AARA 的研究考察采用严格求值的（确定性）函数式程序。不过，也有研究探讨并行 [59, 83] 和惰性 [65, 94]求值的界、下界 [85], ，以及概率程序的期望成本界 [103]。

除 Grosen 等人 [44], 的工作外，此前关于 AARA 的研究都是基于大步成本语义，证明导出界的可靠性。Grosen 等人使用小步语义和逻辑关系给出了语义可靠性证明。本文首次使用小步成本语义给出句法可靠性证明。此外，本文也首次证明了基于栈的抽象机以及对计算与值进行模态分离的表达式的可靠性。

基于势能的推理也已与（并发）分离逻辑 [9, 22, 81],和霍尔逻辑 [21] 相结合，用于推理命令式程序。尽管 Resource Aware ML 中的 AARA 实现 [56, 57] 支持异常，但此前关于 AARA 的工作都未考虑异常处理器或效应处理器，甚至未考虑非局部控制转移；而这正是本文的研究重点。

其他基于类型的系统。 其他用于资源界分析的基于类型的方法建立在尺寸类型 [10, 101], 、细化类型 [26, 27, 48, 74, 75, 109], 、索引类型 [104], 、线性依赖类型 [76–78], 以及结构依赖类型 [30, 43, 86, 87]之上。Avanzini 等人 [11] 研究了用于简化高阶函数界推断的代码变换。Sekiyama 等人 [69, 93]使用细化类型研究了带有代数效应和处理器的程序分析，以及带有分隔延续的程序验证。这些工作不包括自动为带有异常处理器或效应处理器的程序推导界的分析。

7 结论与未来工作

本文介绍了将基于类型的 AARA 扩展到异常、效应及其处理器的方法。新颖的类型系统支持非局部控制转移，并通过针对带有控制栈的抽象机所建立的进展与保持定理，证明了其可靠性。这不同于 AARA 传统的可靠性证明方法，后者采用大步成本语义。我们期待这种替代性的可靠性表述能在未来得到进一步应用。

虽然可以用大步语义对异常和效应进行建模，但控制栈能让处理方式更简单、更局部。同样，也可以通过和类型和高阶函数来模拟异常和效应。不过，这需要进行程序变换，在深层子表达式中传播效应和续延，从而增加分析的负担。使用 AARA 支持通用的（callcc 风格）续延似乎需要采用不同的方法，因为续延允许程序多次使用控制栈中的栈帧。未来工作的其他潜在方向包括支持更丰富的效应类型系统（例如采用行多态），或允许多次调用续延。

数据可用性声明

在第 5 节“评估”中，我们将介绍如何把带有异常和效应的 AARA 实现为名为 RaML 的工具 +，并使用 21 个基准程序对其进行评估。我们已将 RaML 的源代码+ 和这 21 个基准程序作为工件提供 [25], ，其中还包含一个可自动构建 RaML 的 Dockerfile+ ，可将其构建为可执行文件。此外，该工件还包含 RaML 的可执行文件 [53], ，以及一个此前实现的 AARA 版本，我们将用它与 RaML+ 进行比较。我们已提交该工件以供评估。

致谢

本材料基于 Jane Street Group, LLC 和美国国家科学基金会资助的研究成果，资助编号为 2311983 和 2525102。本材料中的任何观点、研究发现、结论或建议均属于作者意见，不一定反映 NSF 的观点。

参考文献

[1] Beniamino Accattoli。2019。《重新审视 λ演算》。载于第 4 届国际计算与演绎形式结构会议（FSCD 2019）（莱布尼茨国际信息学会议论文集（LIPIcs），第 131 卷），Herman Geuvers（编）。Schloss Dagstuhl – Leibniz信息学中心, 德国 Dagstuhl，1:1–1:20。doi:10.4230/LIPIcs. FSCD.2019.1[2] Elvira Albert、Puri Arenas、Samir Genaim 和 Germán Puebla。2008。《成本分析中递推关系上界的自动推断》。载于第 15 届静态分析研讨会（SAS’08）。221–237。[3] Elvira Albert、Puri Arenas、Samir Genaim 和 Germán Puebla。2011。《静态成本分析中的闭式上界》。《自动推理杂志》（2011），161–203。[4] Elvira Albert、Puri Arenas、Samir Genaim、Germán Puebla 和 Damiano Zanardini。2007。《Java字节码的成本分析》。载于第 16 届欧洲编程研讨会（ESOP’07）。[5] Elvira Albert、Puri Arenas、Samir Genaim、German Puebla 和 Damiano Zanardini。2012。《面向对象字节码程序的成本分析》。《理论计算机科学》413，1（2012），142–159。[6] Elvira Albert、Jesús Correas、Pablo Gordillo、Guillermo Román-Díez 和 Albert Rubio。2021。《别把燃料耗尽——智能合约的参数化 gas 上界》。《系统与软件杂志》。

176 2021 110923 10 1016 2021 110923 ( ), 。doi: 。/J.JSS. 。

[7] Elvira Albert、Jesús Correas Fernández 和 Guillermo Román-Díez。2015。《非累积资源分析》。载于《系统构造与分析工具和算法——第 21 届国际会议》（TACAS’15）。[8] Elvira Albert、Samir Genaim 和 Abu Naser Masud。2013。《资源使用上界和下界的推断》。《ACM计算逻辑汇刊》14，3（2013）。[9] Robert Atkey。2010。《结合分离逻辑的摊销资源分析》。载于第 19 届欧洲编程研讨会（ESOP’10）。[10] Martin Avanzini 和 Ugo Dal Lago。2017。《用于复杂度分析的尺寸类型推断自动化》。《美国计算机协会程序设计语言会议论文集》。

1, 43 2017 29 10 1145 3110287 ICFP，文章（8 月），页码。doi: 。/ Martin

[11] Avanzini、Ugo Dal Lago 和 Georg Moser。2012。《函数式程序的复杂度分析：高阶遇上一阶》。载于第 29 届国际函数式编程会议（ICFP’15）。[12] Martin Avanzini 和 Georg Moser。2013。《复杂度的组合框架》。载于第 24 届国际重写技术与应用会议（RTA’13）。[13] Martin Avanzini、Georg Moser 和 Michael Schaper。2020。《概率程序的模块化成本分析》。《美国计算机协会程序设计语言会议论文集》。

4, 2020 172 1 172 30 10 1145 3428240

[14] OOPSLA（），：–：。doi: 。/ Franz Baader 和 Tobias Nipkow。1998。《项重写及其他》。剑桥大学出版社。[15] Andrej Bauer 和 Matija Pretnar。2015。《使用代数效应与处理器进行编程》。《逻辑与代数方法程序设计杂志》。

84, 1 2015 108 123 10 1016 2014 02 001 ( ), –。doi: 。/J.JLAMP. 。Régis Blanc、Thomas A.

[16] Henzinger、Thibaud Hottelier 和 Laura Kovács。2010。《ABC：循环的代数界计算》。载于《逻辑、编程、人工智能与推理——第 16 届国际会议》（LPAR’10）。[17] Guy E. Blelloch。1995。《NESL：一种嵌套数据并行语言》（3.1 版）。技术报告 CMU-CS-95-170。卡内基梅隆大学。[18] Guy E. Blelloch 和 John Greiner。1996。《NESL 的可证明时间和空间高效实现》。载于第 1 届国际函数式编程会议（ICFP’96）。[19] Marc Brockschmidt、Fabian Emmes、Stephan Falke、Carsten Fuhs 和 Jürgen Giesl。2014。《整数程序的运行时与规模复杂度交替分析》。载于第 20 届系统构造与分析工具和算法国际会议（TACAS’14）。

[20] Quentin Carbonneaux、Jan Hoffmann、Tahina Ramananandro 和 Zhong Shao。2014。《C 程序栈空间界限的端到端验证》。载于第 35 届编程语言设计与实现大会（PLDI'14）。制品已提交并获批准。[21] Quentin Carbonneaux、Jan Hoffmann 和 Zhong Shao。2015。《组合式认证资源界限》。载于第 36 届编程语言设计与实现大会（PLDI'15）。制品已提交并获批准。[22] Arthur Charguéraud 和 François Pottier。2019。《在分离逻辑中使用时间额度验证并查集实现的正确性与摊还复杂度》。《自动推理杂志》62，3（2019），331–365。doi:10.1007/s10817- 017-9431-7[23] Krishnendu Chatterjee、Hongfei Fu 和 Amir Kafshdar Goharshady。2017。《递归程序的非多项式最坏情况分析》。载于《计算机辅助验证——第 29 届国际会议》（CAV'17）。[24] Krishnendu Chatterjee、Amir Kafshdar Goharshady、Tobias Meggendorfer 和 undefinedorđe Žikelić。2024。《概率程序资源使用的定量界限》。《美国计算机协会程序设计语言会议论文集》8，OOPSLA1，文章 107（4 月 2024 30 10 1145 3649824），页码。doi: 。/ Ethan Chu、Jan Hoffmann 和 Yiyang Guo。2026。[25]《〈使用自动资源分析处理异常与效应〉的制品》。doi: 10 5281 18742777 。/zenodo。Ezgi Çiçek、Deepak Garg 和 Umut A. Acar。2015。《用于增量[26]计算复杂度的细化类型》。载于第 24 届欧洲编程研讨会（ESOP'15）。[27] Karl Crary 和 Stephanie Weirich。2000。《资源界限认证》。载于第 27 届 ACM 编程语言原理研讨会（POPL'00）。184–198。[28] Joseph W. Cutler、Daniel R. Licata 和 Norman Danner。2020。《用于摊还分析的指称递推式提取》。《美国计算机协会程序设计语言会议论文集》。4, 2020 97 1 97 29 10 1145 3408979 ICFP（），：–：。doi: 。/ Ugo Dal Lago 和 Simone Martini[29]。2006。《λ演算的不变成本模型》。载于《计算障碍的逻辑方法》，Arnold Beckmann、Ulrich Berger、Benedikt Löwe 和 John V. Tucker（编）。Springer Berlin Heidelberg，德国柏林，105–114。[30] Nils Anders Danielsson。2008。《纯函数式数据结构的轻量级半形式化时间复杂度分析》。载于第 35 届 ACM 编程语言原理研讨会（POPL'08）。[31] Norman Danner 和 Daniel R. Licata。2022。《指称语义：函数式语言成本递推式提取的基础》。《函数式编程杂志》32（2022），e8。doi:10.1017/S095679682200003X[32] Ankush Das、Stephanie Balzer、Jan Hoffmann、Frank Pfenning 和 Ishani Santurkar。2021。《数字合约的资源感知会话类型》。载于 2021 年 IEEE计算机安全基础研讨会（CSF'21）。[33] Rowan Davies 和 Frank Pfenning。2001。《分阶段计算的模态分析》。《ACM杂志》48，3（2001），555–604。doi:10.1145/382780.382785[34] Paulo Emílio de Vilhena 和 François Pottier。2021。《效应处理器的分离逻辑》。《美国计算机协会程序设计语言会议论文集》。5, 2021 1 28 10 1145 3434314 POPL（），–。doi: 。/ Paulo Emílio de Vilhena 和 François Pottier[35]。2023。《效应处理器与动态标签的类型系统》。载于《编程语言与系统——第 32 届欧洲编程研讨会（ESOP 2023）》，作为欧洲软件理论与实践联合会议 ETAPS 2023 的一部分举行，法国巴黎，2023 年 4 月 22–27 日，论文集（计算机科学讲义，第 *13990* 卷），225 252 10 1007 978 3 031 30044 8_9 。1994。[36] Thomas Wies（编）。施普林格，–。doi: 。/ - - - - Saumya K. Debray、Pedro López-García、Manuel V. Hermenegildo 和 Nai-Wei Lin。《估算逻辑程序的计算成本》。载于《静态分析：第一届国际静态分析研讨会 SAS'94》，比利时那慕尔，1994 年 9 月 28–30 日，论文集（计算机科学讲义，第 864 卷），Baudouin Le Charlier（编）。施普林格，255 265 10 1007 3 540 58485 4_45 –。doi: 。/ - - - Inc. Facebook。《Infer 网站——成本：运行时复杂度分析[37]》。https://fbinfer.com/docs/checker-cost。2024。[38] Matthias Felleisen。1988。《一等提示符的理论与实践》。载于第 15 届 ACM 编程语言原理研讨会会议录，美国加利福尼亚州圣迭戈，1988 年 1 月 10–13 日，Jeanne Ferrante 和 Peter Mager（编）。ACM出版社，180 190 10 1145 73560 73576 –。doi: 。/ 。Antonio Flores-Montoya 和 Reiner Hähnle[39]。2014。《使用成本方程分析复杂程序的资源》。载于《编程语言与系统——第 12 届亚洲研讨会》（APLAS'14）。[40] Florian Frohn、M. Naaf、Jera Hensel、Marc Brockschmidt 和 Jürgen Giesl。2016。《整数程序的运行时下界》。载于《自动推理——第 8 届国际联合会议》（IJCAR'16）。[41] Jürgen Giesl、René Thiemann、Peter Schneider-Kamp 和 Stephan Falke。2004。《使用 AProVE 自动生成终止性证明》。载于《重写技术与应用：第 15 届国际会议 RTA 2004》，德国亚琛，2004 年 6 月 3–5 日，论文集（计算机科学讲义，第 3091 卷），Vincent van Oostrom（编）。施普林格，210–220。doi:10.1007/978-3-540-25979-4 15_[42] Bernd Grobauer。2001。《DML程序的成本递推式》。载于第 6 届国际函数式编程会议（ICFP'01）。253–264。[43] Harrison Grodin、Yue Niu、Jonathan Sterling 和 Robert Harper。2024。《Decalf：有向效应式成本感知逻辑框架》。《美国计算机协会程序设计语言会议论文集》8，POPL（2024），273–301。doi:10.1145/3632852

[44] Jessie Grosen、David Kahn与Jan Hoffmann。2023。带正则递归类型的自动摊销资源分析。载于第38届ACM/IEEE计算机科学逻辑研讨会（LICS' 23）。[45] Sumit Gulwani、Sagar Jain与Eric Koskinen。2009。用于界限分析的控制流细化与进展不变量。载于编程语言设计与实现会议（PLDI'09）。[46] Sumit Gulwani、Krishna K. Mehra与Trishul M. Chilimbi。2009。SPEED：程序计算复杂度的精确高效静态估算。载于第36届ACM编程语言原理研讨会（POPL'09）。[47] Sumit Gulwani与Florian Zuleger。2010。可达性界限问题。载于编程语言设计与实现会议（PLDI'10）。[48] Martin A. T. Handley、Niki Vazou与Graham Hutton。2019。清算你的资产：在Liquid Haskell中推理资源使用。《美国计算机协会程序设计语言会议论文集》 4, 24 2019 27 10 1145 3371092 POPL, [49] 文章（12月），页。2012 doi：。/ Robert Harper。编程语言实践基础。剑桥大学出版社。[50] Daniel Hillerström、Sam Lindley与John Longley。2024。通过效应处理器实现渐近加速。《函数式编程期刊》 34 2024 5 10 1017 0956796824000030 （），e。doi：。/S Nao Hirokawa与Georg [51] Moser。2014 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λ演算（RTA-TLCA'14）。[52] Jan Hoffmann。2011。带势类型：通过自动摊销分析得到多项式资源界限。博士学位论文。慕尼黑路德维希-马克西米利安大学。[53] Jan Hoffmann。2024。RAML网站。http://raml.co/。[54] Jan Hoffmann、Klaus Aehlig与Martin Hofmann。2011。多元摊销资源分析。载于第38届编程语言原理研讨会（POPL'11）。[55] Jan Hoffmann、Klaus Aehlig与Martin Hofmann。2012。多元摊销资源分析。ACM Trans. Program. Lang. Syst.（2012）。[56] Jan Hoffmann、Klaus Aehlig与Martin Hofmann。2012。Resource Aware ML。载于第24届计算机辅助验证国际会议（CAV'12）。[57] Jan Hoffmann、Ankush Das与Shu-Chun Weng。2017。迈向OCaml的自动资源上界分析。载于第44届编程语言原理研讨会（POPL'17）。[58] Jan Hoffmann与Steffen Jost。2022。自动摊销资源分析的二十年。Math. Struct. Comput. Sci.（2022）。[59] Jan Hoffmann与Zhong Shao。2015。并行程序的自动静态成本分析。载于第24届欧洲编程研讨会（ESOP'15）。[60] Martin Hofmann与Steffen Jost。2003。一阶函数式程序堆空间使用的静态预测。载于第30届ACM编程语言原理研讨会（POPL'03）。[61] Martin Hofmann、Lorenz Leutgeb、David Obwaller、Georg Moser与Florian Zuleger。2022。对数摊销复杂度的基于类型的分析。Math. Struct. Comput. Sci. 32, 6 2022 794 826 10 1017 0960129521000232 [62] 2022 *19* （），–。doi：。/S Rishabh Iyer、Katerina Argyraki与George Candea。网络功能的性能接口。载于第届USENIX网络系统设计与实现研讨会（NSDI 22）。USENIX协会，伦顿，WA，567–584。https://www.usenix.org/conference/nsdi22/presentation/iyer [63] Steffen Jost、Kevin Hammond、Hans-Wolfgang Loidl与Martin Hofmann。2010。高阶程序定量资源使用的静态确定。载于第37届ACM编程语言原理研讨会（POPL'10）。[64] Steffen Jost、Hans-Wolfgang Loidl、Kevin Hammond、Norman Scaife与Martin Hofmann。2009。使用摊销分析为资源受限计算提供碳信用额。载于第16届形式方法研讨会（FM'09）。[65] Steffen Jost、Pedro B. Vasconcelos、Mário Florido与Kevin Hammond。2017。惰性函数式语言的基于类型的成本分析。《自动推理杂志》59（2017），87–120。[66] David Kahn与Jan Hoffmann。2020。指数级自动摊销资源分析。载于第23届软件科学与计算结构基础国际会议（FoSSaCS'20）。[67] David Kahn与Jan Hoffmann。2021。采用量子物理学家方法的自动资源分析。载于第26届函数式编程国际会议（ICFP'21）。[68] Ohad Kammar、Sam Lindley与Nicolas Oury。2013。处理器实践。载于ACM SIGPLAN国际函数式编程会议，ICFP'13，美国马萨诸塞州波士顿，2013年9月25日至27日，Greg Morrisett与Tarmo Uustalu（编）。ACM, 145 158 10 1145 2500365 2500590 –。doi：。/。Fuga Kawamata、Hiroshi Unno、Taro Sekiyama [69] 与Tachio Terauchi。2024。答案细化修改：面向代数效应与处理器的细化类型系统。《美国计算机协会程序设计语言会议论文集》8，POPL，文章5（2024年1月），33 10 1145 3633280 页。doi：。/ Zachary Kincaid、Jason Breck、Ashkan Forouhi Boroujeni [70] 与Thomas Reps。2017。组合式递推分析再探。载于编程语言设计与实现大会（PLDI'17）。[71] Zachary Kincaid、John Cyphert、Jason Breck与Thomas Reps。2017。用于不变量综合的非线性推理。《美国计算机协会程序设计语言会议论文集》2，POPL，文章54（2017年12月），33页。doi:10.1145/3158142

[72] Oleg Kiselyov与Chung-chieh Shan。2007。用于分隔延续的子结构类型系统。载于类型化 λ 演算及其应用，第8届国际会议，TLCA 2007，法国巴黎，2007年6月26日至28日，会议论文集（计算机科学讲义，第4583卷），Simona Ronchi Della Rocca（编）。施普林格，223–239。doi:10.1007/978-3- 540-73228-0 17_[73] Maximiliano Klemen、Pedro López-García、John P. Gallagher、José F. Morales与Manuel V. Hermenegildo。2019。并行逻辑程序静态成本分析的通用框架。载于基于逻辑的程序合成与转换——第29届国际研讨会，LOPSTR 2019，葡萄牙波尔图，2019年10月8日至10日，修订精选论文（计算机科学讲义，第12042卷），Maurizio Gabbrielli（编）。施普林格，19–35。doi:10.1007/978-3-030-45260-5 2_[74] Tristan Knoth、Di Wang、Jan Hoffmann与Nadia Polikarpova。2019。资源引导的程序合成。载于第40届编程语言设计与实现会议（PLDI'19）。[75] Tristan Knoth、Di Wang、Adam Reynolds、Nadia Polikarpova与Jan Hoffmann。2020。液态资源类型。载于第25届函数式编程国际会议（ICFP'20）。[76] Ugo Dal Lago与Marco Gaboardi。2011。线性依赖类型与相对完备性。载于第26届IEEE计算机科学逻辑研讨会（LICS'11）。[77] Ugo Dal Lago与Giulio Pellitta。2013。存在控制算子和高阶函数时的复杂度分析。载于程序设计、人工智能与推理逻辑——第19届国际会议，LPAR-19，南非斯泰伦博斯，2013年12月14日至19日。会议论文集（计算机科学讲义，第8312卷），Kenneth L. McMillan、Aart Middeldorp与Andrei Voronkov（编）。施普林格，

258 273 10 1007 978 3 642 45221 5_19 –。doi：。/ - - - -

[78] Ugo Dal Lago 与 Barbara Petit。2013。类型的几何。载于第届 ACM 编程语言原理研讨会（ POPL'13 ）。[79] Paul Blain Levy 、John Power与Hayo Thielecke。2003。值调用编程语言中的环境建模。《信息与计算》

185, 2 2003 182 210 10 1016 0890 5401 03 00088 9 （），–。doi：。/S - ( ) -

[80] Benjamin Lichtman与Jan Hoffmann。2017。资源感知ML中的数组与引用。载于第届国际计算与演绎形式结构会议（FSCD'17）。[81] Glen Mével、Jacques-Henri Jourdan与François Pottier。2019。Iris中的时间额度与时间收据。载于编程语言与系统——第28届欧洲编程研讨会，ESOP 2019，作为欧洲软件理论与实践联合会议ETAPS 2019的一部分举行，捷克共和国布拉格，2019年4月6日至11日，会议论文集（计算机科学讲义，第

11423 3 29 10 1007 978 3 030 17184 1_1 卷），Luís

[82] Caires（编）。施普林格，–。doi：。/ - - - - Marius Müller、Philipp Schuster、Jonathan Lindegaard Starup、Klaus Ostermann与Jonathan Immanuel Brachthäuser。2023。从能力到区域：实现词法效应处理器的高效编译。《美国计算机协会程序设计语言会议论文集》 7， 2， 255 2023 30 10 1145 3622831 OOPSLA文章（10月），页。doi：。/ Stefan Muller与Jan Hoffmann。2024。CUDA内核求值成本的建模与分析。《ACM并行计算汇刊》（2024）。[83] [84] Van Chan Ngo、Quentin Carbonneaux与Jan Hoffmann。2018。有界期望：概率程序的资源分析。载于第39届编程语言设计与实现会议（PLDI'18）。[85] Van Chan Ngo、Mario Dehesa-Azuara、Matthew Fredrikson与Jan Hoffmann。2017。使用类型验证与合成恒定资源实现。载于第38届IEEE安全与隐私研讨会（S&P '17）。[86] Yue Niu与Robert Harper。2023。面向成本感知指称语义的元语言。载于LICS。1–14。doi:10. 1109/LICS56636.2023.10175777[87] Yue Niu、Jonathan Sterling、Harrison Grodin与Robert Harper。2022。成本感知逻辑框架。《美国计算机协会程序设计语言会议论文集》 6， 2022 1 31 10 1145 3498670 POPL（），–。doi：。/ Lars Noschinski、

[88] Fabian Emmes与Jürgen Giesl。2013。通过依赖对分析项重写的最内层运行时复杂度。《自动推理杂志》51，1（2013），27–56。[89] Gordon D. Plotkin与Matija Pretnar。2009。代数效应处理器。载于编程语言与系统，第18届欧洲编程研讨会，ESOP 2009，作为欧洲软件理论与实践联合会议ETAPS 2009的一部分举行，英国约克，2009年3月22日至29日。会议论文集（计算机科学讲义，第5502卷），Giuseppe Castagna（编）。施普林格，

80 94 10 1007 978 3 642 00590 9_7 –。doi：。/ - - - - Vineet Rajani、Marco

[90] Gaboardi、Deepak Garg与Jan Hoffmann。2021。高阶（摊销）成本分析的统一类型理论。载于第48届编程语言原理研讨会（POPL'21）。[91] John C. Reynolds。1993。延续的发现。LISP符号计算 6，3-4（1993），233–248。[92] Philipp Schuster、Jonathan Immanuel Brachthäuser、Marius Müller与Klaus Ostermann。2022。词法效应处理器的类型化续延传递转换。载于第43届ACM SIGPLAN编程语言设计与实现国际会议论文集（美国加利福尼亚州圣迭戈）（PLDI 2022）。计算机协会，美国纽约州纽约市，

566 579 10 1145 3519939 3523710 –。doi：。/。

[93] Taro Sekiyama与Hiroshi Unno。2023。带分隔延续的依赖时序类型与效应系统。《美国计算机协会程序设计语言会议论文集》7，POPL，文章71（2023年1月），32页。doi:10.1145/3571264

[94] Hugo R. Simões、Pedro B. Vasconcelos、Mário Florido、Steffen Jost 和 Kevin Hammond。2012。惰性函数式程序动态内存分配的自动摊销分析。载于第17届国际函数式编程会议（ICFP'12）。[95] Moritz Sinn、Florian Zuleger 和 Helmut Veith。2014。一种简单且可扩展的界限分析与摊销复杂度分析方法。载于计算机辅助验证——第26届国际会议（CAV'14）。[96] KC Sivaramakrishnan、Stephen Dolan、Leo White、Tom Kelly、Sadiq Jaffer 和 Anil Madhavapeddy。2021。为 OCaml 加装效应处理器。载于第42届 ACM SIGPLAN 编程语言设计与实现国际会议论文集（线上，加拿大）（PLDI 2021）。计算机协会，纽约州纽约市，206–221。doi:10.1145/3453483.3454039[97] Wenhao Tang、Leo White、Stephen Dolan、Daniel Hillerström、Sam Lindley 和 Anton Lorenzen。2024。模态效应类型。arXiv: 2407 11816 [ ] 2407 11816 。cs.PL https://arxiv.org/abs/ 。Robert Endre Tarjan。 。[98] 1985 摊销计算复杂度。SIAM代数离散方法期刊 6，2（1985），306–318。[99] CLP团队。2024。CLP。https://projects.coin-or.org/Clp。[100] Orpheas van Rooij 和 Robbert Krebbers。2025。Affect：一种仿射类型与效应系统。《美国计算机协会程序设计语言会议论文集》 9， 2025 126 154 10 1145 3704841 POPL（），–。doi: 。/ [101] Pedro B. Vasconcelos、Steffen Jost、Mário Florido 和 Kevin Hammond。2015。惰性函数式语言中互递归的基于类型的分配分析。载于第24届欧洲编程研讨会（ESOP'15）。[102] David Voigt、Philipp Schuster 和 Jonathan Immanuel Brachthäuser。2025。效应处理器的动态风。《美国计算机协会程序设计语言会议论文集》 9， 2， 377 2025 27 10 1145 3763155 OOPSLA文章（10月），页码。doi: 。/ [103] 2020 Di Wang 、 David M. Kahn 和 Jan Hoffmann。。提高预期：用类型自动进行期望成本分析。载于第 25 届函数式编程国际会议（ICFP'20）。[104] Peng Wang、Di Wang 和 Adam Chlipala。2017。TiML：一种用于实用复杂度分析并带有不变量的函数式语言。《美国计算机协会程序设计语言会议论文集》 1， 79 2017 26 10 1145 3133903 OOPSLA，文章（10月），[105] 2006 页码。doi: 。/ Stephen Weeks。 。MLton中的整程序编译。载于ACM ML研讨会论文集，2006年，俄勒冈州波特兰，美国，9月 *16, 2006,* 1 10 1145 1159876 1159877 [106] 1975 Andrew Kennedy 和 François Pottier（编）。ACM， 。doi: 。/ 。Ben Wegbreit。 。机械化程序分析。ACM通讯（），–。[107] Reinhard Wilhelm 等。2008。最坏执行时间问题——方法概述与工具综述。《ACM嵌入式计算系统汇刊》7，3（ 2008）。[108] Florian Zuleger、Moritz Sinn、Sumit Gulwani 和 Helmut Veith。2011。使用大小变化抽象分析命令式程序的界限。载于第18届国际静态分析研讨会（SAS'11）。[109] Ezgi Çiçek、Gilles Barthe、Marco Gaboardi、Deepak Garg 和 Jan Hoffmann。2017。关系成本分析。载于第44届编程语言原理研讨会（POPL'17）。 18, 9 1975 528 539

附录

一种带效应的线性界限 AARA 类型系统

A.1 语法文法

$$v ::= x \qquad e ::= \text{值}(v)\ \text{函数}(f.x.e)\ \text{二元组}(v_1;\ v_2)\ \text{左注入}(v)\ \text{右注入}(v)\ \text{空列表}\ \text{计时标记}\{q\}\ \text{let}(e_1;\ x.e_2)\ \text{应用}(v_1;\ v_2)\ \text{二元组分支}(v;\ x_1.x_2.e)\ \text{空类型分支}(v)\ \text{和类型分支}(v;\ x_1.e_1;\ x_2.e_2)\ \text{cons}(v_1;\ v_2)\ \text{线性函数}(x.e)\ \text{定界续延}(k)\ \text{列表分支}(v;\ e_1;\ x_1.x_2.e_2)\ \text{执行}\ [\ell](v)\ \text{处理程序}(e;\ \{\ell\colon\ x.c.e_\ell\}_{\ell\in\Delta};\ y.e_r)\ \text{线性应用}(v_1;\ v_2)$$

A.2 资源标注类型与效应签名

$$\tau ::= \mathcal{T} \mid \mathsf{unit} \mid \tau_1 \times \tau_2 \mid \mathsf{void} \mid A_1 + A_2 \mid \mathsf{L}(A) \mid A \multimap B \odot \Delta \qquad A, B, C ::= \langle \tau, q \rangle$$

$$\mathcal{T} ::= \{A \to B \odot \Delta \mid \Theta\} \qquad \Delta ::= \cdot \mid \Delta, \ell : A \Rightarrow B$$

A.3 值类型判断

$\boxed{\Gamma; q \vdash v\colon \tau}$

$$\frac{\text{T-FunT-Var}\ \Gamma = |\Gamma| \qquad \forall(\tau, q \to B \odot \Delta) \in \mathsf{T}.\ \Gamma;\ f\colon \mathsf{T}\ x\colon \tau;\ q \vdash e\colon B \odot \Delta x\colon \tau;}{\cancel{0 \vdash x\colon \tau}} \qquad \frac{}{\Gamma;\ 0 \vdash \text{函数}\ (f.x.e) : \text{T}\cancel{\text{T-PairT-Unit}\ \Gamma_1; q_1 \vdash v_1 :}}$$

$$\tau_1 \qquad \Gamma_2; q_2 \vdash v_2\colon \tau_2$$

$$\frac{}{\cdot;\ 0 \vdash : \text{单位类型}\ \Gamma_1, \quad \Gamma_2; q_1 + q_2 \vdash \mathsf{pair}(v_1;\ v_2)\colon \tau_1 \times \tau_2}\text{T-Inl}$$

$$\text{T-Inr}\Gamma; p \vdash v\colon \tau_1 \qquad \Gamma;\ p \vdash v\colon \tau_2\Gamma;\ q_1 + p \vdash \text{左注入}(v) : \tau_1, \quad q_1 + A_2 \qquad \Gamma; q_2 + p \vdash \text{右注入}(v) : A_1 + \tau_2, \quad q_2$$

$$\frac{}{\text{T-cons}\ \text{T-Nil}\ \Gamma_1; q_1 \vdash v_1\colon \tau \qquad \Gamma_2; q_2 \vdash v_2 : \mathsf{L}(A)\ A = \tau, \quad p}$$

$$\frac{}{\cdot;\ 0 \vdash \text{空列表} : \mathsf{L}(A)\ \Gamma_1, \Gamma_2; q_1 + q_2 + p \vdash \mathsf{cons}(v_1;\ v_2) : \mathsf{L}(A)}\text{T-LinFun}$$

$$\text{T-Dcont}\Gamma, x\colon \tau;\ p + q \vdash e\colon B \odot \Delta \qquad A_1 \odot \Delta_1 \lhd\colon k \lhd\colon$$

$$\frac{A_2 \odot \Delta_2\Gamma;\ p \vdash \mathsf{linfun}(x.e) : \tau, q \ \multimap B \odot \Delta}{A_2 \multimap A_1 \odot \Delta_1} \qquad \frac{}{\Gamma;\ 0 \vdash \text{定界续延}(k) :}$$

A.4 计算类型判定

$\boxed{\Gamma; q \vdash e : B \odot \Delta}$

$$\frac{\Gamma; q \vdash v : \tau}{\Gamma; q + p \vdash \mathrm{val}(v) : \tau, p \odot \Delta}\ \text{T-Val} \qquad \frac{\Gamma_1; q \vdash e_1 : \tau, q' \odot C \qquad \Gamma_2,\ x : \tau;\ q' \vdash e_2 : B \odot C}{\Gamma_1,\ \Gamma_2;\ q \vdash \mathrm{let}(e_1;\ x.e_2) : B \odot \Delta}\ \text{T-Let}$$

$$\frac{}{\cdot; q + p \vdash \text{计时标记}\{q\} : \text{单位类型}\ p \odot \Delta}\ \text{T-Tick}^{+} \qquad \frac{}{\cdot; p \vdash \text{计时标记}\{-q\} : \text{单位类型}\ p + q \odot \Delta}\ \text{T-Tick}^{-}$$

T-App
$$\frac{\Gamma_1; 0 \vdash v_1 : \mathrm{T} \qquad (\tau, q_1 \to B \odot \Delta) \in \mathrm{T} \qquad \Gamma_2; q_2 \vdash v_2 : \tau}{\Gamma_1,\ \Gamma_2;\ q_1 + q_2 \vdash \mathrm{app}(v_1;\ v_2) :\ B \odot \Delta}$$

T-casepair
$$\frac{\Gamma_1; q_1 \vdash v : \tau_1 \times \tau_2 \qquad \Gamma_2, x_1 : \tau_1, x_2 : \tau_2; q_2 \vdash e : B \odot \Delta}{\Gamma_1,\ \Gamma_2; q_1 + q_2 \vdash \text{二元组分支}(v;\ x_1.x_2.e) : B \odot \Delta}$$

T-Casevoid
$$\frac{\Gamma; q \vdash v : \text{空类型}}{\Gamma; q \vdash \text{case空类型}(v) : B \odot \Delta}$$

T-Casesum
$$\frac{\Gamma_1; q_1 \vdash v : \tau_1, p_1 + \tau_2, p_2 \qquad \Gamma_2,\ x_1 : \tau_1;\ p_1 + q_2 \vdash e_1 : B \odot \Delta \qquad \Gamma_2,\ x_2 : \tau_2;\ p_2 + q_2 \vdash e_2 : B \odot \Delta}{\Gamma_1, \Gamma_2; q_1 + q_2 \vdash \text{和类型分支}(v;\ x_1.e_1;\ x_2.e_2) : B \odot \Delta}$$

T-Caselist
$$\frac{\Gamma_1; q_1 \vdash v : \mathrm{L}(\tau, p) \qquad \Gamma_2; q_2 \vdash e_1 : B \odot \Delta \qquad \Gamma_2, x : \tau, y : \mathrm{L}(\tau, p);\ q_2 + p \vdash e_2 : B \odot \Delta}{\Gamma_1, \Gamma_2;\ q_1 + q_2 \vdash \text{列表分支}(v;\ e_1;\ x.y.e_2) : B \odot \Delta}$$

T-Linapp
$$\frac{\Gamma_1;\ p \vdash v_1 : \tau, q_1 \multimap B \odot \Delta \qquad \Gamma_2; q_2 \vdash v_2 : \tau}{\Gamma_1, \Gamma_2;\ p + q_1 + q_2 \vdash \text{线性应用}(v_1; v_2) :\ B \odot \Delta}$$

T-Do
$$\frac{\Gamma; p \vdash v : \tau_1 \qquad \ell : \tau_1, q_1 \Rightarrow \tau_2, q_2 \in \Delta}{\Gamma;\ p + q_1 \vdash \text{执行}[\ell](v) : \tau_2, q_2 \odot \Delta}$$

T-Handle
$$\frac{\begin{array}{c}\Gamma_1;\ p \vdash e : \tau, q \odot \Delta \qquad \Gamma_3,\ y : \tau;\ q \vdash e_r : C \odot \Delta' \\ \Gamma_2 = |\Gamma_2| \qquad \forall \ell : \tau_1, q_1 \Rightarrow B \in \Delta. (\Gamma_2,\ x : \tau_1,\ c : B \multimap C \odot \Delta';\ q_1 \vdash e_\ell : C \odot \Delta')\end{array}}{\Gamma_1,\ \Gamma_2,\ \Gamma_3;\ p \vdash \text{处理程序}(e;\ \{\ell : x.c.e_\ell\}_{\ell \in \Delta};\ y.e_r) : C \odot \Delta'}$$

A.5 结构化定型规则

$\boxed{\Gamma; q \vdash e : B \odot \Delta}$

T-ContFun
$$\frac{\Gamma, x_1 : \mathrm{T}\ \ x_2 : \mathrm{T};\ q \vdash e : B \odot \Delta}{\Gamma,\ x : \mathrm{T}; q \vdash [x, x/x_1,\ x_2]e : B \odot \Delta}$$

T-WeakFun
$$\frac{\Gamma; q \vdash e : B \odot \Delta}{\Gamma,\ x : \mathrm{T}; q \vdash e : B \odot \Delta}$$

T-WeakPot
$$\frac{\Gamma; q' \vdash e : B \odot \Delta \qquad q \geq q'}{\Gamma; q \vdash e : B \odot \Delta}$$

B 完整成本语义 B.1 状态转换

$$\boxed{s; q \mapsto s'; q'}$$

D-tick

$$\frac{p \geq q}{k \triangleright \text{计时标记}\{q\}; p \mapsto k \triangleleft ; p - q}$$

D-ret

$$\frac{}{k \triangleright \text{值}(v);\ q \mapsto k \triangleleft v; q}$$

D-fun

$$\frac{}{k \triangleright \text{应用（函数（}f.x.e\text{）；}\ v_2\text{）；}\ q \mapsto k \triangleright [\text{函数（}f.x.e\text{）}\ v_2/f,\ x]e;\ q}$$

D-let

$$\frac{}{k \triangleright \text{let}(e_1;\ x.e_2);\ q \mapsto k;\ x.e_2 \triangleright e_1;\ q}$$

D-seq

$$\frac{}{k;\ x.e \triangleleft v;\ q \mapsto k \triangleright [v/x]e;\ q}$$

D-pair

$$\frac{}{k \triangleright \text{二元组分支}(\text{pair}(v_1;\ v_2);\ x_1.x_2.e);\ q \mapsto k \triangleright [v_1, v_2/x_1,\ x_2]e; q}$$

D-inl

$$\frac{}{k \triangleright \text{和类型分支}(\text{左注入}(v);\ x_1.e_1;\ x_2.e_2); q \mapsto k \triangleright [v/x_1]e_1; q}$$

D-inr

$$\frac{}{k \triangleright \text{和类型分支}(\text{右注入}(v);\ x_1.e_1;\ x_2.e_2); q \mapsto k \triangleright [v/x_2]e_2;\ q}$$

D-nil

$$\frac{}{k \triangleright \text{列表分支}(\text{空列表};\ e_0;\ x.y.e_1);\ q \mapsto k \triangleright e_0;\ q}$$

D-cons

$$\frac{}{k \triangleright \text{列表分支}(\text{cons}(v_1;\ v_2);\ e_0;\ x.y.e_1);\ q \mapsto k \triangleright [v_1, v_2/x,\ y]e_1;\ q}$$

D-try

$$\frac{}{k \triangleright \text{处理程序}(e; \{\ell\colon\ x.c.e_\ell\}_{\ell \in \Delta};\ y.e_r);\ q \mapsto k; \text{处理程序}(\_; \{\ell\colon\ x.c.e_\ell\}_{\ell \in \Delta};\ y.e_r)\ \triangleright e;\ q}$$

D-DO

$$\frac{}{k \triangleright \text{执行}[\ell](v);\ q \mapsto k \blacktriangleleft (\ell,\ v,\ \epsilon);\ q}$$

D-Capture

$$\frac{}{k;\ x.e \blacktriangleleft (\ell, v, k');\ q \mapsto k \blacktriangleleft (\ell, v, x.e@\ k');\ q}$$

D-handle

$$\frac{\ell' \in \Delta}{k;\ \text{处理程序}(\_; \{\ell\colon\ x.c.e_\ell\}_{\ell \in \Delta};\ y.e_r)\ \blacktriangleleft (\ell',\ v,\ k');\ q \mapsto k \triangleright [v, \text{定界续延}(\text{处理程序}(\_; \{\ell\colon\ x.c.e_\ell\}_{\ell \in \Delta};\ y.e_r)\ @\ k')/x,\ c]e_{\ell'};\ q}$$

D-normal

$$\frac{}{k;\ \text{处理程序}(\_; \{\ell\colon\ x.c.e_\ell\}_{\ell \in \Delta};\ y.e_r)\ \triangleleft v;\ q \mapsto k \triangleright [v/y]e_r;\ q}$$

D-Dcont

$$\frac{}{k \triangleright \text{线性应用}(\text{定界续延}(k');\ v);\ q \mapsto k@\ k' \triangleleft v; q}$$

D-linfun

$$\frac{}{k \triangleright \text{线性应用}(\text{linfun}(x.e);\ \ v);\ \ q \mapsto k \triangleright [v/x]e;\ \ q}$$

B.2 初始/最终状态

$\boxed{s \quad \text{初始} \qquad s \quad \text{最终}}$

$$\frac{}{\epsilon \triangleright e; q}\ \text{初始}\ (\text{D-初始}) \qquad \frac{}{\epsilon \triangleleft v; q}\ \text{最终}\ (\text{D-最终}) \qquad \frac{}{\epsilon \blacktriangleleft (\ell, v, k); q}\ \text{最终}\ (\text{D-最终-异常})$$

B.3 栈类型

$\boxed{A_1 \odot \Delta_1 \lhd\colon k \lhd\colon A_2 \odot \Delta_2}$

$$\frac{}{B \odot \Delta \lhd\colon \epsilon \lhd\colon B \odot \Delta}\ (\text{K-Emp}) \qquad \frac{A \odot \Delta_1 \lhd\colon k \lhd\colon B \odot \Delta_2 \qquad x\colon \tau; q \vdash e\colon B \odot \Delta_2}{A \odot \Delta_1 \lhd\colon k;\ x.e \lhd\colon \tau, q\ \odot \Delta_2}\ (\text{K-Bnd})$$

$$\frac{\begin{array}{c} A \odot \Delta_1 \lhd\colon k \lhd\colon B \odot \Delta_3 \\ \forall \ell : \tau_1, q_1 \implies C \in \Delta_2.\ x\colon \tau_1, c\colon C \multimap B \odot \Delta_3; q_1 \vdash e_\ell\colon B \odot \Delta_3 \qquad y\colon \tau; q \vdash e_r\colon B \odot \Delta_3 \end{array}}{A \odot \Delta_1 \lhd\colon k; \text{处理程序}(\_; \{\ell\colon x.c.e_\ell\}_{\ell \in \Delta_2};\ y.e_r) \lhd\colon \tau,\ \ q\ \odot \Delta_2}\ (\text{K-Handler})$$

B.4 状态类型检查

$\boxed{q \vdash s}$

$$\frac{\because; q \vdash e\colon B \odot \Delta \qquad \lhd\colon k \lhd\colon B \odot \Delta\_}{q \vdash k \triangleright e}\ (\text{ST-Exp}) \qquad \frac{\because; q_1 \vdash v\colon \tau \qquad \_ \lhd\colon k \lhd\colon \tau, q_2\ \odot \Delta}{q_1 + q_2 \vdash k \lhd v}\ (\text{ST-Val})$$

$$\frac{\because; q_1 \vdash v\colon \tau \qquad \ell : \tau, p \implies C \in \Delta' \qquad \lhd\colon k \lhd\colon B \odot \Delta'\_ \qquad B \odot \Delta' \lhd\colon k' \lhd\colon C \odot \Delta''}{q_1 + p \vdash k \blacktriangleleft (\ell, v, k')}\ (\text{ST-Eff})$$

C 完整可靠性证明

引理 C.1. 若 $\tau = |\tau|$ 和 $\Phi(v : \tau) = q$，则 $q = 0$。

引理 C.2（共享）。若 $\tau \curlyvee (\tau_1, \tau_2)$ 且 $\cdot; q \vdash v : \tau$，则 $\cdot; q_1 \vdash v : \tau_1$，$\cdot; q_2 \vdash v : \tau_2$，其中 $q = q_1 + q_2, q_1 \geq 0, q_2 \geq 0$。

证明。对类型 τ，τ_1，τ_2 进行同步归纳。 □

Le引理 C.3（势能放宽）。若 Γ，$q \vdash e : \tau$，$p \odot \Delta$，则 Γ，$q + r \vdash e : \tau$，$p + r \odot \Delta$。

证明。对计算类型规则进行归纳。 □

引理 C.4（替换）。若 $\cdot; q_1 \vdash v_1 : \tau_1$：
(1) 若 $\Gamma, x_1 : \tau_1; q_2 \vdash v : \tau$，则 $\Gamma; q_1 + q_2 \vdash [v_1/x_1]v : \tau$。*(2)* 若 $\Gamma, x_1 : \tau_1; q_2 \vdash e : B \odot \Delta$，则 $\Gamma; q_1 + q_2 \vdash [v_1/x_1]e : B \odot \Delta$。

证明。对 $\cdot; q_1 \vdash v_1 : \tau_1$: 进行归纳。

- T-var: /
- T-triv、T-fun、T-Pair、T-inl、T-inr、T-nil、T-cons、T-linfun、T-dcont：(A)：对 Γ, x_1 进行归纳：$\tau_1; q_2 \vdash v : \tau$– T-var：$q_2 = 0, v = x_1, \Gamma = \cdot\ \tau_1 = \tau$。于是 $[v_1/x_1]x_1 = v_1$，且 $\cdot; q_1 \vdash v_1 : \tau_1$。– T-triv：/ – T-fun：$q_2 = 0, v = \mathrm{fun}(f.x.e)(x_1 \neq x)\ \tau = \mathrm{T}$.$[v_1/x_1]\mathrm{fun}(f.x.e) = \mathrm{fun}(f.x.[v_1/x_1]e)$。根据假设，$\forall(\tau_x, q' \to B \odot \Delta) \in \mathrm{T}$，我们有 $\Gamma, x_1 : \tau_1, f : \mathrm{T}\ x : \tau_x; q' \vdash e : B \odot \Delta$ 和 $|\Gamma, x_1 : \tau_1| = \Gamma, x_1 : \tau_1$（因此 $|\tau_1| = \tau_1$）。然后根据引理 C.1，$q_1 = 0$。利用归纳假设(B)，$\forall(\tau_x, q' \to B \odot \Delta) \in \mathrm{T}$，我们有 $\Gamma, f : \mathrm{T}\ x : \tau_x; q' + q_1 \vdash [v_1/x_1]e : B$，其中 $q_1 = 0, |\Gamma| = \Gamma$。根据 T-fun，$\Gamma; 0 \vdash \mathrm{fun}(f.x.[v_1/x_1]e) : \tau$。

– T-Pair：

$$\textsc{T-Pair}\ \frac{\Gamma_1; q_1' \vdash v_1' : \tau_1' \qquad \Gamma_2; q_2' \vdash v_2' : \tau_2'}{\underbrace{\Gamma_1, \Gamma_2}_{\Gamma, x_1:\tau_1}; \underbrace{q_1' + q_2'}_{q_2} \vdash \underbrace{\mathrm{pair}(v_1'; v_2')}_{v} : \underbrace{\tau_1' \times \tau_2'}_{\tau}}$$

∗若 $x_1 \in \Gamma_1$，对第一个前提应用 IH(A)，得到 $\Gamma_1 \setminus x_1; q_{\prime 1} + q_1 \vdash [v_1/x_1]v_{\prime 1} : \tau_{\prime 1}$。
对该结果和第二个前提应用 T-Pair，得到 $\Gamma_1 \backslash x, \Gamma_2; q_1 + q_1' + q_2' \vdash$ 二元组$([v_1/x_1]v_1'; v_2') : \tau_1' \times \tau_2'$，证毕。

∗若 $x_2 \in \Gamma_2$，也可以进行对称论证，先对第二个前提应用 IH(A)。– T-inl：

$$\textsc{T-Inl}\ \frac{\Gamma'; p \vdash v' : \tau_1'}{\underbrace{\Gamma'}_{\Gamma, x_1:\tau_1}; \underbrace{q_1' + p}_{q_2} \vdash \underbrace{\mathrm{inl}(v')}_{v} : \underbrace{\langle \tau_1', q_1' \rangle + A_2}_{\tau}}$$

对前提应用 IH(A)，$\Gamma; p + q_1 \vdash [v_1/x_1]v' : \tau_1$。由此应用 T-inl，$\Gamma; p + q_1 + q_1' \vdash$ 左注入 $([v_1/x_1]v') : \tau_1'$，$q_1' + A_2$，即可得到所需结果。– T-inr：与 T-inl 对称– T-nil：/

– T-cons: $v = \text{cons}(v_1'; v_2')$, $\tau = \text{L}(\tau_0, p)$, $\Gamma, x_1 : \tau_1 = \Gamma_1, \Gamma_2$。
根据假设 $q_2 = p + q_1' + q_2'$, $\Gamma_1; q_1' \vdash v_1' : \tau_0$, 以及 $\Gamma_2; q_2' \vdash v_2' : \text{L}(\tau_0, p)$。
则 $x_1 : \tau_1 \in \Gamma_1$ xor $x_1 : \tau_1 \in \Gamma_2$。
∗如果 $x_1 : \tau_1 \in \Gamma_1$, 利用归纳假设(A), $\Gamma_1 \setminus (x_1 : \tau_1); q_{\prime 1} + q_1 \vdash [v_1/x_1]v_{\prime 1} : \tau_0$。
根据 T-cons, $\Gamma; q_2 + q_1 \vdash \text{cons}([v_1/x_1]v_1'; v_2') : \text{L}(\tau_0, p)$。
∗否则, $x_1 : \tau_1 \in \Gamma_2$, 根据归纳假设(A), $\Gamma_2 \setminus (x_1 : \tau_1); q_{\prime 2} + q_1 \vdash [v_1/x_1]v_{\prime 2} : \text{L}(\tau_0, p)$。
根据 T-cons, $\Gamma; q_2 + q_1 \vdash \text{cons}(v'; [v_1/x_1]v') : \text{L}(\tau_0, p)$。
– T-linfun: $v = \text{linfun}(x.e)(x \neq x_1)$ 和 $\tau = \tau', p \multimap B \odot \Delta.[v_1/x_1]\text{linfun}(x.e) = \text{linfun}(x.[v_1/x_1]e)$。根据假设, 我们有 $\Gamma, x_1 : \tau_1, x : \tau'; q_2 + p \vdash e : B \odot \Delta$。使用 IH(B), $\Gamma, x : \tau'; q_2 + p + q_1 \vdash [v_1/x_1]e : B \odot \Delta$。
由 T-linfun, $\Gamma; q_2 + q_1 \vdash [v_1/x_1]\text{linfun}(x.e) : \tau', p \multimap B \odot \Delta$。

– T-dcont: $v = \text{dcont}(k)$ 没有自由变量, 因此该条件空成立。

(B): 对 $\Gamma, x_1 : \tau_1; q_2 \vdash e : B \odot \Delta$ 作归纳。– T-val: $e = \text{val}(v')$ $B = \tau, p$。根据假设, $\Gamma, x_1 : \tau_1; q_2 - p \vdash v' : \tau$。使用归纳假设(A), $\Gamma; q_2 - p + q_1 \vdash [v_1/x_1]v' : \tau$。根据 T-val, $\Gamma; q_2 + q_1 \vdash [v_1/x_1]\text{val}(v') : \tau, p \odot \Delta$。– T-let: $\Gamma, x_1 : \tau_1 = \Gamma_1, \Gamma_2$ 和 $e = \text{let}(e_1; x.e_2)$。根据假设, $\Gamma_1; q_2 \vdash e_1 : \tau, q_3 \odot \Delta$ 和 $\Gamma_2, x : \tau_1; q_3 \vdash e_2 : B \odot \Delta(x \neq x_1)$ 然后 $x_1 : \tau_1 \in \Gamma_1$ 异或 $x_1 : \tau_1 \in \Gamma_2$。

∗若 $x_1 : \tau_1 \in \Gamma_1$, 由归纳假设(B), $\Gamma_1 \setminus (x_1 : \tau_1); q_2 + q_1 \vdash [v_1/x_1]e_1 : \tau, q_3 \odot \Delta$。根据 T-let, $\Gamma; q_2 + q_1 \vdash \text{let}([v_1/x_1]e_1; x.e_2) : B \odot \Delta$。
∗否则, $x_1 : \tau_1 \in \Gamma_2$, 那么由归纳假设(B), $\Gamma_2 \setminus (x_1 : \tau_1)\ x : \tau; q_3 + q_1 \vdash [v_1/x_1]e_2 : B \odot \Delta$。
根据 T-LET 和势能放宽引理 C.3, $\Gamma; q_2 + q_1 \vdash \text{let}(e_1; x.[v_1/x_1]e_2) : B \odot \Delta$。
– T-app: $\Gamma, x_1 : \tau_1 = \Gamma_1, \Gamma_2$ 并且 $e = \text{app}(v_1'; v_2')$。根据假设, $q_2 = p_1 + p_2 + p$。$\Gamma_1; p \vdash v_1' : \text{T}, \tau, p_1 \to B \odot \Delta \in \text{T}$ 和 $\Gamma_2; p_2 \vdash v_2' : \tau$。那么 $x_1 : \tau_1 \in \Gamma_1$ 异或 $x_1 : \tau_1 \in \Gamma_2$。

∗若 $x_1 : \tau_1 \in \Gamma_1$, 由归纳假设(A), $\Gamma_1 \setminus (x_1 : \tau_1)$; $q_1 + p \vdash [v_1/x_1]v_{\prime 1} : \text{T}$。由 T-app, $\Gamma; q_1 + q_2 \vdash \text{app}([v_1/x_1]v_{\prime 1}; v_{\prime 2}) : B \odot \Delta$。
∗否则 $x_1 : \tau_1 \in \Gamma_2$, 则由归纳假设(A), $\Gamma_2 \setminus (x_1 : \tau_1)$; $p_2 + q_1 \vdash [v_1/x_1]v_{\prime 2} : \tau$。
由 T-app, $\Gamma; q_2 + q_1 \vdash \text{app}(v_{\prime 1}; [v_1/x_1]v_{\prime 2}) : B \odot \Delta$。
– T-tick: /–
T-casepair:

$$\text{T-CASEPAIR}\quad \frac{\Gamma_1; q_1' \vdash v : \tau_1' \times \tau_2' \qquad \Gamma_2, x_1 : \tau_1', x_2 : \tau_2'; q_2' \vdash e' : B \odot \Delta}{\underbrace{\Gamma_1, \Gamma_2}_{\Gamma, x_1:\tau_1}; \underbrace{q_1' + q_2'}_{q_2} \vdash \underbrace{\text{casepair}(v; x_1.x_2.e')}_{e} : B \odot \Delta}$$

∗若 $x \in \Gamma_1$, 由第一前提上的归纳假设(A), $\Gamma_1 \setminus x$; $q_{\prime 1} + q_1 \vdash [v_1/x_1]v : \tau_{\prime 1} \times \tau_{\prime 2}$。由此及第二前提上的 T-casepair, $\Gamma_1 \setminus x$, Γ_2; $q_{\prime 1} + q_1 + q_{\prime 2} \vdash \text{casepair}([v_1/x_1]v; x_1.x_2.e_\prime) : B \odot \Delta$, 证毕。

∗若 $x \in \Gamma_2$, 根据第二个前提上的归纳假设(B), $\Gamma_2 \setminus x$, $x_1 : \tau_{\prime 1}$, $x_2 : \tau_{\prime 2}$; $q_{\prime 2} + q_1 \vdash [v_1/x_1]e_\prime : B \odot \Delta$。根据此处和第一个前提上的 T-casepair, $\Gamma_1 \setminus x$, $\Gamma_2; q_{\prime 1} + q_1 + q_{\prime 2} \vdash$ 二元组分支$(v; x_1.x_2.[v_1/x_1]e_\prime) : B \odot \Delta$ 如所求。

– T-casesum：类似于 T-caseprod 情形。
– T-caselist：$\Gamma, x_1 : \tau_1 = \Gamma_1, \Gamma_2$和 $e =$列表分支$(v;\ e_0;\ x.y.e_1)$。
根据假设，$q_2 = q_1' + q_2'$, $\Gamma_1; q_1' \vdash v : \mathrm{L}(\tau_0, r)$, $\Gamma_2; q_2' \vdash e_0 : B \odot \Delta$, $\Gamma_2, x : \tau_0, y : \mathrm{L}(\tau_0, r); q_2' + r \vdash e_1 : B \odot \Delta$。
那么 $x_1 : \tau_1 \in \Gamma_1$ 异或 $x_1 : \tau_1 \in \Gamma_2$。
∗若 $x_1 : \tau_1 \in \Gamma_1$，根据归纳假设(A)，$\Gamma_1 \setminus (x_1 : \tau_1)$；$q_1 + q_{'1} \vdash [v_1/x_1]v : \mathrm{L}(\tau_0, r)$。
根据 T-caselist，Γ；$q_1 + q_2 \vdash$列表分支$([v_1/x_1]v;\ e_0;\ x.y.e_1) : B \odot \Delta$。
∗否则 $x_1 : \tau_1 \in \Gamma_2$，那么根据归纳假设(B)，$\Gamma_2 \setminus (x_1 : \tau_1)$；$q_{'2} + q_1 \vdash [v_1/x_1]e_0 : B \odot \Delta$，并且 $\Gamma_2 \setminus (x_1 : \tau_1)\ x : \tau_0, y : \mathrm{L}(\tau_0, r); q_2' + q_1 + r \vdash [v_1/x_1]e_1 : B \odot \Delta$。
根据 T-caselist，Γ；$q_1 + q_2 \vdash$列表分支$(v;\ [v_1/x_1]e_0;\ x.y.[v_1/x_1]e_1) : B \odot \Delta$。
– T-do：根据前提，$\Gamma,\ x_1 : \tau_1; p \vdash v : \tau_0$其中 $\ell : \tau_0,\ q_0 \Rightarrow B \in \Delta$, $q_2 = p + q_0$。使用归纳假设(A)，$\Gamma; p + q_1 \vdash [v_1/x_1]v : \tau_0$。根据 T-do，$\Gamma$; $q_2 + q_1 \vdash [v_1/x_1]\mathrm{do}[\ell](v) : B \odot \Delta$.– T-handle：

T-Handle

$$\frac{\Gamma_1; q_2 \vdash e : \langle \tau, q \rangle \odot \Delta \qquad \Gamma_2 = |\Gamma_2| \qquad \forall \ell : \langle \tau_1, q_l \rangle \Rightarrow B \in \Delta.\ (\Gamma_2, x : \tau_1, c : B \multimap C \odot \Delta'; q_l \vdash e_\ell : C \odot \Delta') \qquad \Gamma_3, y : \tau; q \vdash e_r : C \odot \Delta'}{\Gamma_1, \Gamma_2, \Gamma_3; q_2 + q_1 \vdash \mathsf{handler}(e; \{\ell : x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) : C \odot \Delta'}$$

令 $\Gamma_1,\ \Gamma_2,\ \Gamma_3 = \Gamma,\ x_1 : \tau_1$。由于 $\Gamma_2 = |\Gamma_2|$，只有两种情况：
∗若 $x_1 \in \Gamma_1$，根据第一个前提上的归纳假设(B)，$\Gamma_1 \setminus x_1;\ q_2 + q_1 \vdash [v_1/x_1]e : \tau,\ q \odot \Delta$。
根据 T-handle，我们得到了所需结论。
∗如果 $x_1 \in \Gamma_2$，根据最后一个前提上的归纳假设(B)，$\Gamma_2 \setminus x_1,\ y : \tau;\ q + q_1 \vdash [v_1/x_1]e_r : C \odot \Delta_r$。根据 T-handle 以及第一个前提上的势能放宽引理 C.3，可得所需结论 .
– T-ContFun、T-weekfun、T-weakpot：对前提应用归纳假设即可得出。

□

引理 C.5（状态放宽）。如果 $q \vdash s$，那么 $q' \vdash s$对于 $q' \geq q$。

证明。根据状态和栈类型规则进行归纳。 □

引理 C.6（栈追加类型判断）。 $B_2 \odot \Delta_2 \triangleleft : \ k_1 @ k_2 \triangleleft : \ B_1 \odot \Delta_1$当且仅当存在 B, Δ使得 $B_2 \odot \Delta_2 \triangleleft : \ k_1 \triangleleft : \ B \odot \Delta$且 $B \odot \Delta \triangleleft : \ k_2 \triangleleft : \ B_1 \odot \Delta_1$

定理 C.7（保持性）。如果 $q \vdash s$并且 $s; q \mapsto s_0; q_0$那么 $q_0 \vdash s_0$。

证明。对 $s; q \mapsto s_0; q_0$ 进行归纳。
• D-tick：$k \triangleright$计时标记$\{r\}$；$q \mapsto k \triangleleft$；$q_0$和 $q \geq r, q_0 = q - r.q \vdash s$，所以 $\cdot;\ q \vdash$计时标记$\{r\}$：$B \odot \Delta$和 $\_ \triangleleft$：$k \triangleleft$：$B \odot \Delta$。我们要证明 $q - r \vdash k \triangleleft$。令 $B = \tau, x$。对 $\cdot;\ q \vdash$计时标记$\{r\}$：$\tau, x \odot \Delta$ 进行归纳，以证明 $q - x \geq r$和 $\tau =$单位类型。只有两种可能的情况：– T-tick$^+$：那么 $r \geq 0, q = r + x, \tau =$ 单位类型，$x = 0$。成立。– T-tick$^-$：那么 $r \leq 0$, $q = r + x, \tau =$单位类型，$x = 0$。成立。– T-WeakPot：根据假设，$\cdot;\ q' \vdash$计时标记$\{r\}$：$\tau, x \odot \Delta$，其中 $q \geq q'$。利用内部的归纳假设 $q' - x \geq r$，可得 $q' - x \geq r$。根据 T-Unit，$\cdot;\ 0 \vdash :$ 单位类型。我们有 $\_ \triangleleft$：$k \triangleleft$：$\tau, x \odot \Delta$，继而根据 ST-Val，$x \vdash k \triangleleft$. $x \leq q - r$。根据状态放宽引理 C.5，$q - r \vdash k \triangleleft$。

• D-fun: $k \triangleright \text{app}(\text{fun}(f.x.e); v_2); q \mapsto k \triangleright [\text{fun}(f.x.e)\ v_2/f,\ x]e;\ q_0$
$q \vdash s,$ 因此 $\_ \triangleleft : k \triangleleft : B \odot \Delta', ' ; q \vdash \text{app}(\text{fun}(f.x.e); v_2) : B \odot \Delta$
。只需证明 $\cdot;\ q \vdash [\text{fun}(f.x.e)\ v_2/f, x]e : B \odot \Delta_0$。对 $\cdot;\ q \vdash \text{app}(\text{fun}(f.x.e); v_2) : B \odot \Delta$ 作归纳。只有两种情况：– T-app：根据假设，$\cdot;\ 0 \vdash \text{fun}(f.x.e) : \text{T},\ \cdot;\ q_2 \vdash v_2 : \tau, q = q_1 + q_2, \tau, q_1 \to B \odot \Delta \in \text{T}$。反演可得，对于 $\tau, q_1 \to B \odot \Delta \in \text{T}$，有 $f: \text{T}\ x: \tau; q_1 \vdash e: B \odot \Delta_0$。对 f, x 两次使用替换引理 C.4，得到 $\cdot;\ q_1 + q_2 \vdash [\text{fun}(f.x.e)\ v_2/f, x]e : B \odot \Delta_0$。– T-WeakPot：根据假设，$\cdot;\ q' \vdash \text{app}(\text{fun}(f.x.e); v_2) : B \odot \Delta$ 且 $q \geq q'_0$。利用内部归纳假设，$\cdot;\ q' \vdash [\text{fun}(f.x.e)\ v_2/f, x]e : B \odot \Delta_0$。根据 T-WeakPot，$\cdot;\ q \vdash [\text{fun}(f.x.e)\ v_2/f, x]e : B \odot \Delta_0$。

• D-ret: $k \triangleright \text{val}(v);\ q \mapsto k \triangleleft v;\ q.q \vdash k \triangleright \text{val}(v)$，因此 $\_ \triangleleft : k \triangleleft : B \odot \Delta \text{ and } \cdot;\ q \vdash \text{val}(v) : B \odot \Delta_0$。令 $B = \tau, r_0$。对 $\cdot;\ q \vdash \text{val}(v) : \tau, r \odot \Delta$ 进行归纳，以证明存在 q_1，其中 $\cdot;\ q_1 \vdash v : \tau$ 且 $q - r \geq q_{1_0}$。只有两种情况：– T-val: $B = \tau, r$，且假设给出 $\cdot;\ q_1 \vdash v : \tau$，其中 $q_1 + r = q_0$。– T-WeakPot：根据假设，$\cdot;\ q \vdash \text{val}(v) : B \odot \Delta$，且 $q \geq q'_0$。利用内部归纳假设，我们有 $q - r \geq q' - r \geq q_1$，其中 $\cdot;\ q_1 \vdash v : \tau_0$。根据 ST-val，$q'_1 + r \vdash k \triangleleft v.q'_1 + r \leq q - r + r = q$，因此由状态放宽引理 C.5 得到 $q \vdash k \triangleleft v_0$。

• D-let: $k \triangleright \text{let}(e_1; x.e_2); q \mapsto k; x.e_2 \triangleright e_1; q_0$。根据假设，$q \vdash s$，因此 $\cdot;\ q \vdash \text{let}(e_1; x.e_2) : B \odot \Delta, \_ \triangleleft : k \triangleleft : B \odot \Delta_0$。对 $\cdot;\ q \vdash \text{let}(e_1; x.e_2) : B \odot \Delta$ 作归纳，以证明：存在某个 τ_1, q_1，使得 $\cdot; q \vdash e_1 : \tau_1, q_1 \odot \Delta$ 且 $x: \tau_1; q_1 \vdash e_2 : B \odot \Delta_0$。只有 2 种可能的情况：– T-let：由假设立即得出。– T-WeakPot：根据假设，$\cdot;\ q' \vdash \text{let}(e_1; x.e_2) : B \odot \Delta$，其中 $q \geq q'_0$。利用内部归纳假设，我们有 τ_1, q_1，使得 $\cdot; q' \vdash e_1 : \tau_1, q_1 \odot \Delta, x: \tau_1; q_1 \vdash e_2 : B \odot \Delta_0$。根据 T-WeakPot，$\cdot; q \vdash e_1 : \tau_1, q_1 \odot \Delta, x: \tau_1; q_1 \vdash e_2 : B \odot \Delta_0$。根据 K-bnd，$k; x.e_2 \triangleleft : \tau_1, q_1 \odot \Delta_0$。我们有 $\cdot; q \vdash e_1 : \tau_1, q_1 \odot \Delta_0$。根据 ST-exp，$q \vdash k; x.e_2 \triangleright e_{1_0}$。

• D-seq: $k; x.e \triangleleft v;\ q \mapsto k \triangleright [v/x]e;\ q_0$。根据假设，$q \vdash k; x.e \triangleleft v_0$。反演可得 $q = q_1 + q_2, \cdot;\ q_1 \vdash v : \tau$ 和 $\_ \triangleleft : k; x.e \triangleleft : \tau, q_2 \odot \Delta_0$。反演可得，$x: \tau; q_2 \vdash e : B \odot \Delta$ 和 $\_ \triangleleft : k \triangleleft : B \odot \Delta_0$。根据替换引理 C.4，$\cdot;\ q_2 + q_1 \vdash [v/x]e : B \odot \Delta_0$。根据 ST-exp，$q_2 + q_1 \vdash k \triangleright [v/x]e\,(q = q_1 + q_2)_0$。
• D-nil: $k \triangleright$ 列表分支(nil; e_0; $x.y.e_1$); $q \mapsto k \triangleright e_0; q_0$。根据假设，$\_ \triangleleft : k \triangleleft : B \odot \Delta, \cdot;\ q \vdash$ 列表分支(nil; e_0; $x.y.e_1$): $B \odot \Delta_0$。只需证明 $\cdot;\ q \vdash e_0 : B \odot \Delta_0$。对 $\cdot;\ q \vdash$ 列表分支(nil; e_0; $x.y.e_1$): $B \odot \Delta$ 进行归纳。只有 2 种情况可能发生：– T-caselist：根据假设，$\cdot;\ q_2 \vdash e_0 : B \odot \Delta, \cdot;\ q_1 \vdash \text{nil}: \text{L}(A)$，其中 $q = q_1 + q_{2_0}$。然后根据 T-WeakPot，$\cdot;\ q \vdash e_0 : B \odot \Delta_0$。– T-WeakPot：与其他情况类似。

• D-cons: $k \triangleright$ 列表分支(cons(v_1; v_2); e_0; $x.y.e_1$); $q \mapsto k \triangleright [v_1, v_2/x, y]e_1$; q。根据假设, $\_ \triangleleft$: $k \triangleleft$: $B \odot \Delta$, $\cdot$; $q \vdash$ 列表分支(cons(v_1; v_2); e_0; $x.y.e_1$): $B \odot \Delta$。只需证明 $\cdot$; $q \vdash [v_1, v_2/x, y]e_1$: $B \odot \Delta$。对 $\cdot$; $q \vdash$ 列表分支(cons(v_1; v_2); e_0; $x.y.e_1$): $B \odot \Delta$ 进行归纳。只有 2 种情况可能发生: – T-caselist: 根据假设, x: τ, y: L(τ, p); $q_2 + p \vdash e_0$: $B \odot \Delta$, 其中 $\cdot$; $q_1 \vdash$ cons(v_1; v_2): L(τ, p), $q = q_1 + q_2$。对 $\cdot$; $q_1 \vdash$ cons(v_1; v_2): L(τ, p) 进行逆向推导, 得到 $q_1 = q'_1 + q'_2 + p$, $\cdot$; $q'_1 \vdash v_1$: τ, $\cdot$; $q'_2 \vdash v_2$: L(τ, p)。使用替换引理 C.4, $\cdot$; $q_2 + p + q'_1 + q'_2 \vdash [v_1, v_2/x, y]e_1$: $B \odot p_0$, 也就是

$\cdot$; $q \vdash [v_1, v_2/x, \ y]e_1$: $B \odot \Delta_\circ$ –

T-WeakPot: 与其他情况类似。

• D-pair: $k \triangleright$ 二元组分支(pair(v_1; v_2); $x_1.x_2.e$); $q \mapsto k \triangleright [v_1, v_2/x_1, x_2]e$; q。根据 ST-exp 的前提, $k \triangleleft$: $B \odot \Delta$, $\cdot$; $q \vdash$ 二元组分支(pair(v_1; v_2); $x_1.x_2.e$): $B \odot \Delta$。只需证明 $\cdot$; $q \vdash [v_1, v_2/x_1, x_2]e$: $B \odot \Delta$。对 $\cdot$; $q \vdash$ 二元组分支 (pair(v_1; v_2); $x_1.x_2.e$): $B \odot \Delta$ 进行归纳。只有两种可能的情况: – T-casepair: 根据前提, x_1: τ_1, x_2: τ_2; $q_2 \vdash e$: $B \odot \Delta$, $\cdot$; $q_1 \vdash$ pair(v_1; v_2): $\tau_1 \times \tau_2$, and $q = q_1 + q_2$。对 $\cdot$; $q_1 \vdash$ pair(v_1; v_2): $\tau_1 \times \tau_2$ 进行逆推, 可得 $q_1 = q'_1 + q'2$, $\cdot$; $q'_1 \vdash v_1$: τ_1, and

$\cdot$; $q'_2 \vdash v_2$: τ_2。利用替换引理 C.4, $\cdot$; $q_2 + q'_1 + q'_2 \vdash [v_1, v_2/x_1, x_2]e$: $B \odot \Delta$ 即为所求, 因为 $q = q_2 + q'_1 + q'_2$。– T-WeakPot: 与其他情形类似。

• D-inl: $k \triangleright$ 和类型分支(左注入(v); $x_1.e_1$; $x_2.e_2$); $q \mapsto k \triangleright [v/x_1]e_1$; q。根据 ST-exp 的前提, $k \triangleleft$: $B \odot \Delta$, $\cdot$; $q \vdash$ 和类型分支(左注入(v); $x_1.e_1$; $x_2.e_2$): $B \odot \Delta$。只需证明 $\cdot$; $q \vdash [v/x_1]e_1$: $B \odot \Delta$。对 $\cdot$; $q \vdash$ 和类型分支(左注入(v); $x_1.e_1$; $x_2.e_2$): $B \odot \Delta$ 进行归纳。仅有 2 种可能情形: – T-Casesum: 根据前提, x_1: τ_1, ; $p_1 + q_2 \vdash e_1$: $B \odot \Delta$, $\cdot$; $q_1 \vdash$ 左注入(v): $\tau_1, p_1 + A_2$, 以及 $q = q_1 + q_2$。对 $\cdot$; $q_1 \vdash$ 左注入(v): $\tau_1, p_1 + A_2$ 作反演, 得到 $q_1 = p_1 + q'1$ 以及 $\cdot$; $q'_1 \vdash v$: τ_1。利用替换引理 C.4, $\cdot$; $p_1 + q_2 + q'_1 \vdash [v/x_1]e_1$: $B \odot \Delta$ 即为所求, 因为 $q = p_1 + q_2 + q'1$。– T-WeakPot: 与其他情形类似。

• D-inr: 与上文对称

• D-do: $k \triangleright$ do[ℓ](v); $q \mapsto k \blacktriangleleft (\ell, v, \epsilon)$; q 根据左侧 ST-Exp 的前提, $\cdot$; $q \vdash$ do[ℓ](v): $B \odot \Delta$ 和 $\_ \triangleleft$: $k \triangleleft$: $B \odot \Delta_\circ$ 对 $\cdot$; $q \vdash$ do[ℓ](v): $B \odot \Delta$ 进行归纳, 以证明 $\cdot$; $p \vdash v$: τ_1 with ℓ: $\tau_1, q_1 \Rightarrow \tau_2, q_2 \in \Delta$ and $p \leq q - q_1$。只有 2 种可能情况: – T-do: 由其前提直接得出, 且 $q = p + q_1$。– T-WeakPot: 与其他情况类似。令 $k' = \epsilon$。由 K-Emp 和 ST-Eff, $q \vdash k \blacktriangleleft (l, v, \epsilon)$, 如所求。

• D-Capture: k; $x.e \blacktriangleleft (\ell, v, k')$; $q \mapsto k \blacktriangleleft (\ell, v, x.e@\ k')$; q 根据左侧 ST-Eff 的前提, 我们有 – $q = q_1 + p$– $\cdot$; $q_1 \vdash v$: τ– $\Delta' = \Delta$, ℓ: $\tau, p \Rightarrow C$– $\_ \triangleleft$: k; $x.e \triangleleft$: $B \odot \Delta'$ – $B \odot \Delta' \triangleleft$: $k' \triangleleft$: $C \odot \Delta''$ 前 3 个前提可以保持不变。对于第 4 个前提, 使用 K-Bnd 进行逆向推导, 得到 $\_ \triangleleft$: $k \triangleleft$: $B_2 \odot \Delta'$, 其中 x: τ_2; $q_2 \vdash e$: $B_2 \odot \Delta'$, 以及 $B = \tau_2, q_2$。对于第 5 个前提, 我们首先使用 K-Emp

和 K-Bnd 来构造单帧栈 $x.e$具有定型 $B_2 \odot \Delta' \triangleleft: \quad x.e \triangleleft: \quad B \odot \Delta'$，然后应用栈之间的 引理 C.6 $x.e$和 k'来得到 $B_2 \odot \Delta' \triangleleft: \quad x.e @ k' \triangleleft: \quad C \odot \Delta''$。

– $q = q_1 + p$– $\cdot; q_1 \vdash v : \tau$–

$\Delta' = \Delta, \ell : \tau, p \Rightarrow C$– $\_ \triangleleft: k \triangleleft:$

$B_2 \odot \Delta'$– $B_2 \odot \Delta' \triangleleft: x.e @$

$k' \triangleleft: C \odot \Delta''$

现在有了这 5 个前提，我们可以应用 ST-Eff 为右侧定型。

• D-Handle: k; 处理程序$(_; \{\ell: x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) \blacktriangleleft (\ell, v, k_\prime); q \mapsto k \triangleright [v, d/x, c]e_{\ell'}; q$当 $\ell \in \Delta$ 且其中 $d =$定界续延(处理程序$(_; \{\ell: x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) @ k_\prime)$。

由左侧 ST-Eff 的前提可知: (1) $q = q_1 + p$, (2) $\cdot; q_1 \vdash v : \tau$, (3) $\ell' : \tau, p \not\in \in \Delta$, (4) $\_ \triangleleft: k$; handler$(_; \{\ell: x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) \triangleleft: B' \odot \Delta$, 以及 (5) $B' \odot \Delta \triangleleft: k' \triangleleft: C \odot \Delta''$。使用 K-Handler 对 (4) 进行反演，得到 (6) $\_ \triangleleft: k \triangleleft: B \odot \Delta_3$, (7) 对所有 $\ell : \tau_1, q_1 \Rightarrow C$in $\Delta, x : \tau_1, c : C \multimap B \odot \Delta_3; q_1 \vdash e_\ell : B \odot \Delta_3$, 以及 (8) $y : \tau'; q' \vdash e_r : B \odot \Delta_3$，使得 $B' = \tau', q'$。接下来，我们使用 K-Emp、K-Handler、(7)、(8) 和引理 C.6 构造栈类型$B \odot \Delta_3 \triangleleft:$ handler$(_; \{\ell: x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) @ k' \triangleleft: C \odot \Delta''$。然后根据 T-Dcont，可得 (9)

$\cdot; 0 \vdash d : C \multimap B \odot \Delta_3$.

现在，我们可以在 e_ℓ 中对 x 和 c 应用替换引理。对于 $[v/x]$，我们有 (2)。对于 [定界续延(. . .)/c]，我们有 (9)。对于 e_ℓ，我们有 (7)。综合起来，我们得到

$$\cdot; p + q_1 + 0 \vdash [v, d/x, c]e' : B \odot \Delta_3$$

最后，利用 (1) 中的 $q = q_1 + p$，以及 (6) 中的 $\_ \triangleleft: k \triangleleft: B \odot \Delta_3$，我们可以应用 ST-Exp 为整个右侧表达式定型。

• D-normal: k; 处理程序$(_; \{\ell: x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) \triangleleft v; q \mapsto k \triangleright [v/y]e_r; q$。根据假设，$q = q_1 + q_2, \cdot; q_1 \vdash v : \tau, \_ \triangleleft: k$; 处理程序$(_; \{\ell: x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) \triangleleft: \tau, q_2 \odot \Delta$。反演，得到 $\_ \triangleleft: k \triangleleft: \tau, q_2 \odot \Delta$。根据 ST-val, $q_1 + q_2 = q \vdash k \triangleleft v$。

• D-try: $k \triangleright$处理程序 $(e; \{\ell : x.c.e_\ell\}_{\ell \in \Delta}; y.e_r); q \mapsto k$; 处理程序$(_; \{\ell : x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) \triangleright e; q$。根据假设，$_ \triangleleft: k \triangleleft: B \odot \Delta$和 $\cdot; q \vdash$处理程序$(e; \{\ell: x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) : B \odot \Delta$。对 $\cdot; q \vdash$处理程序$(e; \{\ell: x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) : B \odot \Delta$进行归纳，以证明存在 Δ', q'使得$q \geq q'$，且 $\cdot; q' \vdash e_1 : B \odot \Delta'$，并且 $\forall \ell : \tau_1, q_1 \Rightarrow \tau_2, q_2 \in \Delta', x : \tau_1, c : \tau_2, q_2 \multimap A \odot \Delta; q_1 \vdash e_\ell : A \odot \Delta$，且 $y : \tau \vdash e_r : B \odot \Delta$

只有 2 种情况是可能的:

– T-Handle：当上下文为空时，由假设直接得出，并选择 $q' = q$。

– T-WeakPot：根据假设，对于某个 q'''，使得 $q \geq q'''$，我们有 $\cdot; q''' \vdash$处理程序$(e; \{\ell : x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) : B \odot \Delta$。然后应用内部归纳假设（IH），得到存在 Δ', q''，使得 $q''' \geq q''$，且 $\cdot; q' \vdash e_1 : B \odot \Delta'$，以及 $\forall \ell : \tau_1, q_1 \Rightarrow \tau_2, q_2 \in \Delta', x : \tau_1, c : \tau_2, q_2 \multimap A \odot \Delta; q_1 \vdash e_\ell : A \odot \Delta$，且 $y : \tau \vdash e_r : B \odot \Delta$。现在选取 $q' = q''$，则由传递性可得$q \geq q'$。其余条件立即成立。

然后根据 K-Handler、ST-exp 和 T-WeakPot, $q \vdash k$; 处理程序$(_; \{\ell: x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) \triangleright e$。

• D-Dcont: $k \triangleright$app(定界续延$(k_\prime); v); q \mapsto k @ k_\prime \triangleleft v; q$

根据左侧 ST-Exp 的前提，$\cdot; q \vdash$app(定界续延$(k'); v) : B \odot \Delta$，其中 $\_ \triangleleft: k \triangleleft: B \odot \Delta$。我们要证明 $\cdot; q_1 \vdash v : \tau$且 $\_ \triangleleft: k @ k' \triangleleft: \tau, q_2 \odot \Delta'$，其中 $q = q_1 + q_2$。

对 $\cdot; q \vdash$app(dcont$(k'); v) : B \odot \Delta$进行归纳。只有两种可能的情况:

– T-linapp：根据前提，我们有 $\cdot; 0 \vdash$dcont$(k') : \tau, q_2 \multimap B \odot \Delta$，以及 $\cdot; q_1 \vdash v : \tau$，其中 $q = q_1 + q_2$。对第一个前提应用 T-Dcont 进行求逆，可得 $B \odot \Delta \triangleleft: k' \triangleleft: \tau, q_2 \odot \Delta'$。然后，根据C.6引理，在 k和 k'的定型之间，我们有 $\_ \triangleleft: k @ k' \triangleleft: \langle \tau, q_2 \rangle \odot \Delta'$。

– T-WeakPot：与其他情况类似。

由 ST-Val 可得所需的右侧。

□

引理 C.8（规范形式）。若$\cdot; q \vdash v : \tau$，且

- $\tau = \mathsf{L}(A)$，则以下二者之一成立：$v =$ 空列表，或 $v = \mathsf{cons}(v_1; v_2)$。
- $\tau = \tau_1 \times \tau_2$，则 $v =$ 二元组$(v_1; v_2)$
- $\tau = A_1 + A_2$，则以下二者之一成立：$v =$ 左注入(v_1) 或 $v =$ 右注入(v_2)
- $\tau = \mathcal{T}$，那么是 $v =$ 函数$(f.x.e_\prime)$、线性函数$(x.e)$，或 定界续延(k)。

推论 C.9. 若 $\cdot; q \vdash e : B \odot \Delta$，并且

- $e =$ 列表分支$(v;\ e_0;\ x.y.e_1)$，那么要么是 $v =$ 空列表，要么是 $v = \mathsf{cons}(v_1;\ v_2)$。
- $e =$ 二元组分支$(v; x_1.x_2.e_\prime)$，那么 $v =$ 二元组$(v_1;\ v_2)$
- $e =$ 和类型分支$(v; x_1.e_1;\ x_2.e_2)$，那么要么是 $v =$ 左注入(v_1)，要么是 $v =$ 右注入(v_2)
- $e =$ 应用$(v_1;\ v_2)$，那么是 $v_1 =$ 函数$(f.x.e_\prime)$、线性函数$(x.e_\prime)$，或 定界续延(k)。

定理 C.10（进展）。若 $q \vdash s$，则要么 s 最终，要么 $s; q \mapsto s'; q'$。

证明。对 $q \vdash s$ 进行归纳。

- ST-val：$q \vdash k \triangleleft v$。按 k 的结构进行归纳。– $k = \epsilon$：$k \triangleleft v$ 是最终的。– $k = k_\prime; x.e$：根据 D-seq，$k_\prime; x.e \triangleleft v; q \mapsto k_\prime \triangleright [v/x]e; q$。– $k = k_\prime; \mathsf{handler}(\_; \{\ell: x.c.e_\ell\}_{\ell \in \Delta}; y.e_r);$：根据 D-normal，$k_\prime; \mathsf{handler}(\_; \{\ell: x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) \triangleleft v; q \mapsto k_\prime \triangleright [v/y]e_r; q$。

- ST-Eff：$q \vdash k \blacktriangleleft (\ell' v, k')$ 按 k 的结构进行归纳。– $k = \epsilon$：$k \blacktriangleleft (\ell' v, k')$ 是最终的 – $k = k_1; x.e$：根据 D-Capture，$k_1; x.e \blacktriangleleft (\ell' v, k'); q \mapsto k_1 \blacktriangleleft (\ell' v, x.e @\ k'); q$– $k = k_1; \mathsf{handler}(\_; \{\ell: x.c.e_\ell\}_{\ell \in \Delta}; y.e_r)$：我们的类型系统保证 $\ell' \in \Delta$。因此，根据 D-handle，$k_1; \mathsf{handler}(_; \{\ell: x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) \blacktriangleleft (\ell' v, k'); q \mapsto k_1 \triangleright [v, $定界续延$(\mathsf{handler}(_; \{\ell: x.c.e_\ell\}_{\ell \in \Delta}; y.e_r) @\ k')/x, c]e_\ell'; q$

- ST-exp：$q \vdash k \triangleright e$。根据前提，$\cdot; q \vdash e : B \odot \Delta$ 且 $\_ \triangleleft : k \triangleleft : B \odot \Delta$。对 e 的结构进行归纳。– $e = \mathsf{caselist}(v;\ e_0;\ x.y.e_1)$ 根据类型逆推和典范形式引理，要么是 $v = \mathsf{nil}$，要么是 $v = \mathsf{cons}(v_1; v_2)$。
  - $v = \mathsf{nil}$：根据 D-nil，$k \triangleright \mathsf{caselist}(\mathsf{nil};\ e_0;\ x.y.e_1); q \mapsto k \triangleright e_0; q$
  - $v = \mathsf{cons}(v_1;\ v_2)$：根据 D-cons，$k \triangleright \mathsf{caselist}(\mathsf{cons}(v_1;\ v_2);\ e_0;\ x.y.e_1); q \mapsto k \triangleright [v_1, v_2/x, y]e_1; q$– e是其他某个非计时标记$\{r\}$表达式。这些情况与 $e = \mathsf{caselist}(v;\ e_0;\ x.y.e_1)$ 类似。对于每种情况，先应用典范形式引理/推论，然后应用以下规则之一：D-ret、D-fun、D-let、D-seq、D-pair、D-inl、D-inr、D-do、D-try、D-linfun、D-Dcont。最终可得 $k \triangleright e; q \mapsto s_\prime; q$，其中某个 $s_\prime$– $e = \mathsf{tick}\{r\}$。对 $\cdot; q \vdash \mathsf{tick}\{r\} : B \odot \Delta$ 进行归纳，以证明 $q \geq r$。只有两种可能的情况：
    - T-tick：$q = r$。
    - T-WeakPot：$q \geq r$ 根据归纳假设。$q \geq r$然后根据 D-tick，$k \triangleright$计时标记$\{r\};\ q \mapsto k \triangleleft;\ q - r$。

□

收稿日期：2025年10月9日；录用日期：2026年2月17日