效应与重写的结合

Ezra e. k. Cooper

所属机构

摘要

在对以代数方式定义的计算效应进行研究时，通常期望效应符号遵守某些等式。若将这些等式定向，就会得到一个重写系统，它可能成为转换或优化程序中效应的有效方法。为此，我们需要证明该重写系统具有强规范化性质，或称终止性。本文定义了一个用于开展此类证明的框架，并扩展 Dershowitz 著名的递归路径序，以证明某些效应系统的终止性。

2012 ACM 学科分类 计算理论 → 重写系统

关键词与短语 项重写、强规范化、终止性、代数效应、函数式编程

数字对象标识符 10.4230/LIPIcs...

资助 *Ezra e.k.Cooper*：独立

arXiv:2302.02462v1 [cs.PL] 5 Feb 2023

1 引言

Plotkin 和 Power [15] 提出了一种将计算效应视为代数项的观点。其操作语义展示了一个包含效应的源项如何规约为最终效应项，该项表示程序执行的效应轨迹，或者说，所有可能线性轨迹构成的一棵树。这一研究方向探讨了效应算子之间的方程，这些方程定义了它们彼此之间的本质关系。

我们在此基础上提出一个观察：将这些方程定向为效应系统的一组重写规则 后，我们便可以机械地将效应项（作为一条轨迹）规约为更紧凑的形式，即一个状态。在这种情况下，这些重写规则可以被视为取代语言实现，甚至取代使效应得以“生效”的硬件。或者，这类重写规则也可以成为定义和实现含效应程序优化的一种优雅方式。

从含效应项或轨迹得到最终状态，并非只有通用重写系统这一种方式。Ahman 和 Staton [1] 提出了一种求值归一化策略来实现这一点。不过，我们认为研究广义重写系统的行为很有意思。任意应用重写规则的自由度，可能对编译器实现或其他程序变换引擎有所助益。

将一般的项重写应用于代数效应之后，研究者会想知道一些常见性质是否成立，例如合流性和强规范化。虽然可以给出一个特设证明，但更理想的做法是将问题分解，使得只需证明项重写系统具有终止性，就能轻松地将该证明推广为关于该系统在 Moggi 的计算元语言 [13] 上下文中的证明。

如今，在项重写的文献中，有许多证明强规范化的技巧，例如“递归路径序”或 RPO [5]。使用这种技巧时，实践者只需给出函数符号上的一种排序，并证明每条重写规则都以某种方式遵循这一排序。这样，繁复的归纳证明便可由对重写规则进行一些相对简单（尽管是递归的）检查来取代。在“高阶递归路径序”HORPO [9] 的相关文献中，RPO 已扩展到带有 λ-抽象和 β-归约的演算。不过，这不禁让我们思考：那么，带有 let 构造的演算又如何呢?

©Ezra e. k. Cooper；采用知识共享许可协议 CC-BY 4.0 许可 莱布尼茨国际信息学论文集 达格施图尔城堡—莱布尼茨信息学中心，达格施图尔出版社，德国

LIPICS

如果我们能将 RPO 或 HORPO 扩展到计算元语言，那么就能轻松证明某些效应系统的终止性。

本文做出了以下贡献：我们展示了如何将多种计算效应在操作层面上解释为重写规则，而非等式，从而使效应符号的运作更具操作性。我们提出了一种证明代数效应系统强规范化的新方法：将递归路径序提升到 Moggi 的元语言中。我们运用该方法证明了一个全局状态效应系统的终止性，并展示了如何将其轨迹归约为状态。

熟悉递归路径序、强规范化的 Tait-Girard 证明 [7] 以及将 Tait-Girard 方法扩展到计算元语言的 Lindley-Stark 方法 [11]的人，对我们的核心证明不会感到意外。但通过结合这些方法，我们得到了一个有力的结果，可直接用于证明含有代数效应的语言的终止性，其中效应本身受重写规则约束。

2 代数效应

让我们回顾一下代数效应的基本框架。该框架由 Plotkin 和 Power [15]（以及 Bauer [2]）提出，并由 Plotkin 和 Pretnar [17] 通过 let 构造加以扩展。编程语言通过效应符号来定义，每个符号代表一种可执行的原子效应。这些符号建立在两种语法角色之上：参数 p 和参数项a，形式如 $e_{\vec{p}}(\vec{a})$。参数表示效应使用的数据，例如要打印到终端的消息；而参数项则表示可能的延续项，具体取决于效应的结果 。一种从某个输入源读取单个位的效应 $readbit$ 自然会有两个参数项位置，分别代表程序读取 0 或 1 时将采取的行为： $readbit($ $zeroContinuation,\ \ oneContinuation)$。

另一个效应 $print_m$ 可用于打印相应的消息：

$readbit(print_{\texttt{cold}}(30)\ \ print_{\texttt{hot}}(70))$

它会相应地打印 cold 或 hot，然后返回一个数字，例如表示摄氏度数的 30 或 70。作为效应参数项的任何纯（无效应）项，都表示计算最终返回的值 ，具体取决于从根节点到叶节点所经过的路径。

Plotkin 和 Power 语义允许效应符号移出求值上下文，实质上是在求值时将它们浮到项的顶层，因此范式是由效应符号构成的树，表示可能的轨迹，其叶节点则是相应的返回值。因此，我们可以将 λ 项 $\lambda x.x+5$ 应用于上述带效应的项，它将重写如下：

$$
\begin{array}{lll}
 & (\lambda x.x+5)readbit(print_{\texttt{cold}}(30), print_{\texttt{hot}}(70)) & \\
\rightsquigarrow & readbit((\lambda x.x+5)print_{\texttt{cold}}(30), (\lambda x.x+5)print_{\texttt{hot}}(70)) & \text{(eff-assoc)} \\
\rightsquigarrow & readbit(print_{\texttt{cold}}((\lambda x.x+5)\ 30), print_{\texttt{hot}}((\lambda x.x+5)\ 70)) & \text{(eff-assoc)} \\
\rightsquigarrow & readbit(print_{\texttt{cold}}(30+5), print_{\texttt{hot}}(70+5) & \text{(abs-}\beta\text{)} \\
\rightsquigarrow & readbit(print_{\texttt{cold}}(35), print_{\texttt{hot}}(75) & \text{(abs-}\beta\text{)}
\end{array}
$$

最后一行是一个范式，它展示了一棵树： $readbit$ operation可以选择两条路径中的任意一条；在每条路径上都会打印某条特定消息；最后，每条路径都终止于一个值，该值是将 λ-抽象应用于原始副作用表达式的结果后计算得到的。

在当前工作中，我们探究当这些计算（效应）树进一步接受它们自身的重写规则处理时会发生什么；这些重写规则既可以应用于源项，也可以应用于这些最终的计算树。这样的归约会使树失去“轨迹”的性质，但为我们提供了一种模拟效应机制的方法，类似于效应的抽象机。

我们的证明中不会用到效应参数，因此不写出它们。

2.1 示例

当我们讨论编程语言的语义时，效应的重写规则可以看作实现了语言实现中的机制，它会将各个效应约化为一个状态，而该状态本身表示为一个效应项的范式。

示例：全局状态

全局状态被建模为一个单一的全局位置，它可以保存某种类型 T 的值。全局状态效应系统的签名为

$$arity(assign_i) = 1$$
$$arity(get) = T$$

请注意， $assign_i$ 由值 i参数化，以便赋值给全局变量。Plotkin 和 Power 区分“参数”和“参数项”。 get 的参数项由存储类型 T的值索引：其“元数”为 T。

如果这些符号不作解释（即不受任何重写约束），那么重写序列的结果就只是一个计算树，它描绘了程序所有可能轨迹构成的树。不过，我们也可以赋予它另一种含义：在存在最终状态的情况下，将其视为这次计算实际得到的最终状态。为此，我们可以指定重写规则（改编自 Plotkin 和 Power [16]），以执行轨迹归约：

$$\begin{aligned} assign_i(get(t_1, \ldots, t_n)) &\rightsquigarrow assign_i(t_i) \\ assign_i(assign_j(s)) &\rightsquigarrow assign_j(s) \\ get(t_1, \ldots, t_i, \ldots, t_n) &\rightsquigarrow get(t_1, \ldots, s_i, \ldots, t_n) \\ &\qquad \text{where } t_i = get(s_1, \ldots, s_n) \end{aligned}$$

这个系统的范式中，不存在相邻的获取-获取 对、不存在相邻的赋值-赋值对，也不存在位于赋值内部的获取，因此它们实际上只是单个赋值或单个获取，其参数是程序最终得到的值。单个赋值 通过其参数告诉我们全局变量的最终状态。单个获取 则表示读取了一个未初始化变量，或进行了非确定性读取；在特定情境下，其他与此无关的机制可能会禁止这种情况。

将这种重写方法与 Johann 等人给出的示例进行对比。 [8] 其中，全局状态计算树通过一个在演算之外定义的函数映射到其最终状态。

示例：非确定性

有一个效应 or，带有 $arity(or) = 2$，以及一条重写规则：

$$or(or(s_1, s_2), s_3) \rightsquigarrow or(s_1, or(s_2, s_3))$$

该规则的目的是将可能计算的分支树规范化为可能结果的扁平列表，因此其行为更像列表。

示例：循环效应

据我们所知，文献中尚未提出这种效应；不过，为了阐明我们工作的动机，我们将探讨一种看起来像“效应”、但具有复杂重写行为的事物。

在使用外部服务（例如数据库服务器或基于 Web 的 API）进行编程时，我们常常希望自己的服务能够在外部服务短暂中断时保持稳健。为此，程序员会在系统中设置有限次重试。如果外部服务在重试期间恢复运行，程序便会继续正常运行；但如果有限次重试全部用尽，就会向用户返回错误。

我们用一对效应来建模这样的系统：请求和重试。程序每次向服务发出请求 时，请求可能会失败（其后续由第一个参数 t 表示），也可能返回一个有意义的值（由一组带索引的参数表示： s_1、.....、 s_n）。因此，我们引入一个效应 $request(t、 s_1、.....、 s_n)$。我们还引入一个效应 $retry(u、 r)$，表示将计算 r 重试 u 所指定的次数。在 u 中，我们会找到一个由重写符号 $zero()$ 和 $succ(u)$ 表示的数，也就是皮亚诺数（我们使用皮亚诺数，使算术运算便于通过重写来处理）。

$$
\begin{aligned}
retry(zero(), request(t, s_1, \ldots, s_n)) &\rightsquigarrow t \\
retry(succ(u), request(t, s_1, \ldots, s_n)) &\rightsquigarrow request(retry(u, t'), s_1, \ldots, s_n) \\
&\qquad \text{where } t' = request(t, s_1, \ldots, s_n)
\end{aligned}
$$

这个效果重写系统产生的结果更像是一条轨迹，而非一个最终状态，因为它会在计算树中将 $request$效应复制 u 次。为了对这条轨迹求值，我们可以选择进一步的重写规则，使 $request$ 的行为类似于全局状态示例中的 get：展平连续的 $request$，并为整个集合选定一个结果。

这或许是一个人为构造的效应重试循环模型，但它展示了我们的技术如何应用于一个比其他示例稍复杂的系统。

示例：并行性

二元效应 par 表示对两个效应流进行并行求值。我们假设它与其他效应结合使用。系统中的每一种其他效应 e 都对应以下重写规则：

$$
\begin{aligned}
par(e(s_1, \ldots, s_n), t) &\rightsquigarrow e(par(s_1, t), \ldots, par(s_n, t)) \\
par(s, e(t_1, \ldots, t_n)) &\rightsquigarrow e(par(s, t_1), \ldots, par(s, t_n))
\end{aligned}
$$

这些规则通常并非合流的，因此从同一个源项可以推导出多个不同的最终状态。当然，这正是并行性的特性。

沿袭分叉-汇合并行的传统，我们还可以添加一个效应 $join$，将两个结果汇集到一个结果项中，以便继续计算。这里的 $\langle \cdot, \cdot \rangle$ 表示将普通数据配对为一个积类型（$S \times T$）：

$$
join(par(v, w)) \rightsquigarrow pure(\langle v, w \rangle)
$$

3 两种带效果元语言

我们必须停下来调和文献中对效应的两种语法处理方式：一种显式标记单子类型，另一种则将其隐去。这两种处理方式都出现在 Moggi 的早期著作中，并且在文献中通常被称为

λ_{ml} 和 λ_c。

第一种方法（λ_{ml}）使用计算类型，而另一种方法（λ_c）则将效应操作视为对类型系统透明。后一种记法在 Plotkin 和 Power [15] 以及其他代数效应研究中更为常见。Sabry 和 Wadler [19] 证明了两者之间存在密切对应关系。

下面给出了每种方法的关键类型规则。我们用 E(T) 表示结果类型为 T 的带效应计算的类型。（单个这样的效应构造子意味着整个系统中的效应共用一个全局单子。）

显式计算类型（λ_{ml}）

$$\frac{\Gamma \vdash t : \mathrm{E}(S) \qquad \Gamma, x : S \vdash u : \mathrm{E}(T)}{\Gamma \vdash \mathsf{let}\ x \Leftarrow t\ \text{于}\ u : \mathrm{E}(T)}$$

作为类型透明结构的效应（λ_c）

$$\frac{\Gamma \vdash t : S \qquad \Gamma, x : S \vdash u : T}{\Gamma \vdash \mathsf{let}\, x \Leftarrow s\ \text{于}\ u : T}$$

在 λ_{ml} 中，我们对显式构造的纯主语上的 let 执行 β-归约：

$$\mathsf{let}\, x \Leftarrow \mathit{pure}(t)\, \mathsf{in}\, u \rightsquigarrow u\{t/x\}$$

在 λ_c 中，β-归约由主语位置上值的句法类别触发（假设 v 描述了一种值的句法类别）：

$$\mathsf{let}\, x \Leftarrow v\ \text{在}\ u \rightsquigarrow u\{v/x\}$$

v 中的值由语法定义；该语法禁止 let 和效应应用，至少在它们未嵌入 λ-主体中的情况下如此。

本文的主要证明以 λ_{ml} 为基础语言。

4 具有显式效应的元语言

现在我们定义核心对象：一种用于计算效应的元语言，它基于 Moggi [13] 的基本语法，并采用 Plotkin 和 Pretnar [17] 的代数效应规则。作为这两种语言的融合，它包含显式效应符号和 let 构造。

与一些后续关于代数效应的研究不同，我们没有采用 Levy 等人提出的细粒度值调用 [10], ，实际上也不假定值调用求值顺序，因为我们希望尽可能广泛地涵盖那些能够用我们的方法证明为强正规化的有趣重写系统。事实上，我们的一个动机示例（NRC）就受益于允许在任意位置进行重写，也受益于允许任意的项对项应用（区别于 FGCBV）。

下面是我们元语言的语法：

$$\begin{array}{rcll} s, t, u & ::= & x \mid \lambda x.u \mid \mathit{pure}(t) \mid st \mid \gamma(t_1, ..., t_n) \mid \mathsf{let}\, x \Leftarrow t\, \mathsf{in}\, u & \\ \gamma & ::= & e \mid f & \text{(rewritable symbols)} \\ e, e', e'' & & & \text{(effect symbols)} \\ f, g & & & \text{(function symbols)} \end{array}$$

我们区分两类可重写符号：效应符号和函数符号。这两类符号各自有其类型规则，但在重写理论中通常被同等对待，因此我们用 γ 表示这两类符号，用 e （及其带撇号的形式）表示效应符号，用 f、g、.....表示非效应函数符号。根据 Plotkin-Power 框架的启发，这两个子类都适用一些元语言重写规则。

图 1 类型规则

$$\frac{\Gamma, x : S \vdash u : T}{\Gamma \vdash \lambda x.u : S \to T} \qquad \frac{\Gamma \vdash s : S \to T \qquad \Gamma \vdash t : S}{\Gamma \vdash st : T} \qquad \frac{\Gamma \vdash t : \mathsf{E}(S) \qquad \Gamma, x : S \vdash u : \mathsf{E}(T)}{\Gamma \vdash \mathsf{let}\, x \Leftarrow t \,\mathsf{in}\, u : \mathsf{E}(T)}$$

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash \mathit{pure}(t) : \mathsf{E}(T)} \qquad \frac{\text{for each } t_i, \Gamma \vdash t_i : \mathsf{E}(T) \qquad \Sigma(e) = n}{\Gamma \vdash e(t_1, \ldots, t_n) : \mathsf{E}(T)}$$

$$\frac{\text{for each } s_i, \Gamma \vdash s_i : S_i \qquad \Sigma(f) = S_1 \times \cdots \times S_n \to T}{\Gamma \vdash f(s_1, \ldots, s_n) : T}$$

图 2 元语言的重写规则。

$$\begin{aligned}
(\lambda x.u)t &\rightsquigarrow_{ml} t\{u/x\} && \text{(abs-}\beta\text{)}\\
\mathsf{let}\, x \Leftarrow \mathit{pure}(t)\,\mathsf{in}\, u &\rightsquigarrow_{ml} t\{u/x\} && \text{(let-}\beta\text{)}\\
\mathsf{let}\, y = (\mathsf{let}\, x \Leftarrow t_1\,\mathsf{in}\, t_2)\,\mathsf{in}\, u &\rightsquigarrow_{ml} \mathsf{let}\, x \Leftarrow t_1\,\mathsf{in}\,\mathsf{let}\, y \Leftarrow t_2\,\mathsf{in}\, u \quad \text{when } x \notin FV(u) && \text{(let-assoc)}\\
\mathsf{let}\, x \Leftarrow e(t_1, \ldots, t_n)\,\mathsf{in}\, u &\rightsquigarrow_{ml} e(t_1', \ldots, t_n') && \text{(eff-assoc)}\\
&\quad \text{其中 } t_i' = \mathsf{let}\, x \Leftarrow t_i\,\mathsf{in}\, u
\end{aligned}$$

类型由以下语法定义：

$$\begin{aligned}
S, T \quad &::= \quad B \mid \mathsf{E}(T) \mid S \to T\\
B \quad &\qquad (\text{basic types})
\end{aligned}$$

该语言为效应类型定义了一种特殊的类型构造子：E(T) 是计算的类型，这些计算可能产生效应，并返回类型为 T 的值。函数抽象的类型是$S \to T$。

类型规则见图 1。为了给可重写符号赋予类型，我们假设有一个签名 Σ ，它将每个效应符号映射到一个元数，并将每个函数符号映射到形如 $S_1 \times \cdots \times S_n \to T$ 的类型签名。在本文中，我们将交替使用形如 $s_1, \ldots, s_n$ 的项序列和向量记法 $\vec{s}$。在这样的上下文中，可能存在一个不同的项，称为 s。

元语言的重写规则见图 2。eff-assoc 规则改编自 Plotkin 和 Pretnar [17] 中的一个类似等式。值得注意的是，eff-assoc 不 适用于函数符号，只适用于效应符号。我们省略了 η-约简；Moggi [13]以及许多其他研究中都采用了这种约简，但它对我们的证明和示例没有帮助。

准备好元语言这锅底汤后，接下来便加入更多具有代数性质且针对特定领域的重写规则，制成浓汤，如示例所示。因此，假设我们有一个底层重写系统，它由一个签名 Σ 定义，其中包含具有元数的可重写符号及其重写规则 $l \rightsquigarrow r$。我们要求 l 和 r 是符号项，而非任意项：

$$l, r ::= x \mid \gamma(l_1, l_2, \ldots, l_n)$$

因此，这个符号重写系统并不了解 let 构造、λ 抽象和应用。

重写上下文

在本文中，我们假设所有重写关系（包括标记为 $\rightsquigarrow$、 $\rightsquigarrow_{ml}$、 $\succ$ 和 $\dot{\succ}$ 的关系）都具有相容闭包，因此可以在任意项上下文中应用。这是重写系统中相当标准的假设。

在某些应用领域，可能需要限制允许使用的重写上下文，例如限制为值调用求值顺序。在计算效应研究中，这种做法很常见。但由于允许在任意位置进行重写，我们的结果仍然更具一般性。

5 递归路径序的定义

现在我们定义递归路径序 $\succ$，它关联语言符号部分的项，并且是规范化证明的关键工具。

递归路径排序是相对于可重写符号上的一个序 $>_\Sigma$ 定义的。在上下文明确时，我们将符号序记作$>$。通过记作 $\succ_{lex}$，关系 $\succ$ 可扩展为项序列上的字典序。（（高阶）递归路径序通常允许符号的参数按多重集序或字典序排列。目前我们只定义字典序。）我们用 $\succeq$ 表示 $\succ$与 $=$的并集。

▶ **定义 1**（递归路径排序）。 若以下条件之一成立，则定义 $s \succ t$ 成立：
1. $s = \gamma(s_1, \ldots, s_n)$、 $t = \gamma(t_1, \ldots, t_n)$，且$\vec{s} \succ_{lex} \vec{t}$，并且对所有 j， $s \succ t_j$ *2.*
$s = \gamma(s_1, \ldots, s_m)$、 $t = \gamma'(t_1, \ldots, t_n)$，且 $\gamma > \gamma'$，并且对所有 j， $s \succ t_j$ *3.*
$s = \gamma(s_1, \ldots, s_m)$，且存在某个 i， $s_i \succeq t$.
容易忽略的是，这个关系是归纳定义的，而且在（3）的情形中，也就是 $=$ 关系的 $\succeq$ 部分，隐藏着一个基例。递归路径排序的所有推导最终都会落在叶节点上，而这些叶节点断言右侧项等于左侧项的某个直接子项。

让我们花点时间直观地理解这一步的目的，以及它在整个证明中的作用。 $\succ$ 关系本质上涵盖了一大类终止的重写系统；对于给定的效应签名 Σ 和符号之间的一种排序，都可以定义这类系统。该排序针对具体的重写系统而定，但 $\succ$ 关系对重写规则本身做了些抽象。它通常是我们感兴趣的关系（$\rightsquigarrow$）的超集，因此（$\rightsquigarrow$）$\subseteq$（$\succ$）。用户还必须检查其重写规则（$\rightsquigarrow$）确实符合上述条件（构成一个（$\succ$）关系），但这通常很容易做到，如此一来，便可以说“免费”获得一个强大的终止性证明。

$\succ$ 关系仅对符号改写规则进行抽象；为了将其扩展到元语言中，我们定义

$$(\dot{\succ}) \triangleq (\succ) \cup (\rightsquigarrow_{ml}).$$

而这个 $\dot{\succ}$ 关系正是我们将要证明具有强规范化性质的关系。因此，目标演算中 $(\rightsquigarrow) \cup (\rightsquigarrow_{ml})$ 所对应的关系也必须是强规范化的。

我们也可以通过两条强大的推理规则来介绍递归路径排序：

$$\text{RPO-SUBTERM}\quad \frac{s_i \succeq t \text{ for some } s_i}{\gamma(s_1, \ldots, s_n) \succ t} \qquad \text{RPO-SYMBOL}\quad \frac{(\gamma, \vec{s}) >_{RPO} (\gamma', \vec{t}) \qquad s \succ t_i \text{ for each } t_i}{\gamma(s_1, \ldots, s_n) \succ \gamma'(t_1, \ldots, t_m)}$$

其中， $>_{RPO}$ 排序定义为由 $>_\Sigma$和 $\succ_{lex}$组成的字典序。

在下文中， t 的一个归约项 是指满足 $t \succ t'$的项 t' ； 如果 t 在关系 $\succ$下强规范化，我们记作 SN（t）。当一个项强规范化时，我们可以对其归约树进行归纳（我们只对关系（$\succ$）下的归约树使用这一方法，不用于其他项关系）；调用这一原理时，我们会写作“对 t进行归纳，按 $\succ$排序”。如果手头有多个可正规化的项，我们也可以对它们同时归纳，其中假设当其中任意一项被归约后，该组项仍满足命题。

5.1 延续

证明中的一个关键难点，是要说明在存在 let-assoc 规则时仍然能够强规范化。该规则会以推动进展的方式重组项，却不会使项变小。因此，我们需要一种构造，以便追踪这一进展并据此进行归纳。在 let-assoc 规则中， let $x \Leftarrow$(let $y \Leftarrow s_1$ in s_2) in s_3 的内层左侧有一个 let，我们希望看到它消失。这种形式的归约项 let $y \Leftarrow s_1$ in let $x \Leftarrow s_2$ in s_3，则在内层右侧有 let 形式，更接近范式。我们减少了位于其他 let 形式左侧的 let 形式数量。因此，遵循 Lindley 和 Stark [11], 的做法，我们定义延续，它们是由“错误类型”的 let 上下文构成的栈——也就是上文说明中的 let $x \Leftarrow [\ ]$in s_3。let-assoc 重写会缩小这个栈的大小，因此在归纳证明中，我们便有办法证明进展确实在发生。

定义延续 K 如下:

$$\begin{aligned} K &::= \epsilon \mid K \circ F \\ F &::= \mathsf{let}\, x \Leftarrow [\ ]\, \mathsf{in}\, u \end{aligned}$$

将用 F 中的空位填入 t后得到的项记作 $F[t]$ 。将递归填入空位后得到的结果记作 $K@t$ ，因此 $\epsilon @t = t$且 $K \circ F@t = K@F[t]$。将 K中框架的数量记作$|K|$ ，因此 $|\epsilon| = 0, |K \circ F| = 1 + |K|$。

若对于某个新鲜的 x，有 $K@x \succ K'@x$ ，则记作 $K \succ K'$ 。注意，对于单个框架，唯一的重写形式为 let $x \Leftarrow [\ ]$ in $u \succ$ let $x \Leftarrow [\ ]$ in u'，其中 $u \succ u'$。

6 强规范化

遵循 Lindley-Stark 的做法，定义类型索引的可约性谓词如下:

可约项$_B(t) \triangleq SN(t)$ 可约项$_{S \to T}(s) \triangleq$ 可约项$_T(st)$
对所有 t 其中 可约项$_S(t)$ 可约项$^{\mathsf{E}(T)}(t) \triangleq SN(K@t)$ 对
所有 K其中 可约项$^{\top}_T(K)$ 可约项$^{\top}_T(K) \triangleq SN(K@纯(t))$
对所有 t 其中 可约项 $_T(t)$

为方便行文，我们将等价地写作 $t \in$ 可约项$_T$ 或 可约项$_T(t)$。

将以下各项定义为中性项：一个应用， st。

- 一个变量， x。
-

可约项_ 具有四个标准性质，且都有标准证明。

▶ **引理 2**。 对于任意 T，可约项$_T$ 中都存在某个项。

证明。 对 T 进行归纳。

情形 B。自由变量构成的项满足此条件。情形 $S \to T'$。由归纳假设可得一个项 $t \in$ 可约项$_{T'}$，因此 $\lambda x.t \in$ 可约项$_{S \to T'}$。情形 E(T')。由归纳假设可得一个项 $t \in$ 可约项$_{T'}$，因此 $pure(t) \in$ 可约项$_{\mathsf{E}(T')}$。◀

▶ **引理 3.** 对于任意 T、 t，若 $t \in$ 可约项$_T$，则 t 是强规范化的。

证明。 对 T 作归纳，并依据 可约项$_T$ 定义。在 E(T') 情形中， t 是某个被直接断言为强规范化（SN）的项的子项。在 $T_1 \to T_2$ 情形中，归纳假设给出了一个项的强规范化，而该项以 t 为子项。我们需要证明 可约项$_S$ 有元素， 引理 2 证明了这一点。◀

▶ **引理 4.** 对于任意满足 $s \succ s'$ 的 $s \in$ 可约项$_T$，有 $s' \in$ 可约项$_T$。

证明。 对 T作归纳。

情形 B。 s' 只需是强规范化的；由于它是 s 的归约项，因此确实如此。情形 $T_1 \to T_2$。证明对于任意 $t \in$，若 s_1，则 $s't \in$Red$_{T_2}$。我们有 $st \in$ Red$_{T_2}$。但是 $st \succ s't$，所以由归纳假设可得结论。情形 E(T')。证明当 $K \in$ Red$_T^\top$ 时， $K@s' \in SN$。同样， $K@s$ 归约为 $K@s'$；既然后者是强规范化的，前者也一样。◀

▶ **引理 5。** 给定一个中性项 s，若它的每个归约结果都属于 可约项$_T$，则 s 属于 可约项$_T$。

证明。 对 T 的结构进行归纳。

情形 B 由于这些归约结果属于 Red$_B$，它们也属于强正规化，这符合 Red$_B$ 的定义。情形 $S \to T'$ 要证明对每个 $t \in$， $st \in$ 都属于 Red$_{T'}$。我们知道 t 是强正规化；对 t 的归约树进行归纳。考察 st 的归约。由于 s 是中性项，它不是 λ 抽象，因此头部不存在 β 归约。唯一的归约结果是 $s't$（其中 $s \succ s'$）和 st'（其中 $t \succ t'$）。第一种情形下，引理假设足以完成证明。第二种情形下，内层归纳假设足以完成证明。 情形 E(T') 给定 $K \in$Red$_{T'}^\top$，我们要证明 $SN(K@s)$。对 K 进行归纳。由于 s 是中性项，唯一的归约结果是 $K'@s$ （其中 $K \succ K'$）和 $K@s'$ （其中 $s \succ s'$）。（注意，在元语言中没有规则可以将框架 F 重写为 $F[st]$ 中的应用 st，而且也无法通过符号重写提供这样的规则。）第一种情形下，内层归纳假设足以完成证明。第二种情形下，引理假设足以完成证明。◀

▶ **引理 6。** 如果 $SN(u\{t/x\})$ 那么 $SN(u)$。

证明。 从构造性角度看， u 的归约树中的每个归约，在$u\{t/x\}$的归约树中都有一个对应项。因此， u 的树不会大于另一个项的树；若后者收敛，它也不会发散。◀

现在我们证明，在满足适当条件时，每个项构造子都能构造出一个可归约项。

▶ **引理 7。** 若可约项$_{S \to T}(s)$ 且可约项$_S(t)$，则可约项$_T(st)$。

证明。 由可约项$_{S \to T}$的定义可立即得出。◀

▶ **引理 8。** 若对每个 t，都有可约项$_T(u\{t/x\})$ 属于可约项$_S$，则可约项$_{S \to T}(\lambda x.u)$。

证明。 由于 $(\lambda x.u)t$ 是中性项，因此只需证明它的所有归约结果都是可约的。由于 t 属于 可约项$_S$，我们知道它是强正规化的。根据引理 6， u 是强正规化的，因此我们可以对这两棵重写树应用同时归纳。归纳假设是：对于任意 $(\lambda x.u')t$ 都是可约的；对于任意 $u \succ u'$，$(\lambda x.u)t'$ 也是可约的。现在我们来讨论 $(\lambda x.u)t$ 的归约结果对应的情形。

- 情形 $(\lambda x.u)t \dot{\succ} (\lambda x.u')t$，其中 $u \dot{\succ} u'$；根据归纳假设，这一项是可约
- 的。情形 $(\lambda x.u)t \dot{\succ} (\lambda x.u)t'$，其中 $t \dot{\succ} t'$；根据归纳假设，这一项是
- 可约的。情形 $(\lambda x.u)t \dot{\succ} u\{t/x\}$；根据引理假设，这一项是可约的。 ◀

▶ **引理 9**。如果 $K \dot{\succ} K'$ 那么 $|K| \geq |K'|$。

证明。对 K 作结构归纳。如果 $K = \epsilon$，则没有归约。如果 $K = K_0 \circ F$，我们有归约 $K_0 \dot{\succ} K_0'$ 和 $F \dot{\succ} F'$，它们保持长度不变（前者由归纳假设可得）。

通过（let-结合），我们也有 $K = K_0 \circ F_1 \circ (\mathsf{let}\, x \Leftarrow [\ ]\, \mathsf{in}\ u) \circ K_1$ 和 $K' = K_0 \circ (\mathsf{let}\, x \Leftarrow [\ ]\, \mathsf{in}\ F_1[u]) \circ K_1$。而这少一个框架。◀

▶ **引理 10**。如果 $t \in$ 可约项$_T$ 那么 纯$(t) \in$ 可约项$^{\mathsf{E}(T)}$。

证明。要证明：对于任意 $K@$纯$(t) \in$ 可约项$^{\mathsf{E}(T)}$，$K \in$ 可约项$_T^{\top}$。但这由可约项$_T^{\top}$的定义立即可得。◀

▶ **引理 11.** 若 $s \in SN$ 和 $K@(u\{s/x\}) \in SN$，则 $K@(\mathsf{let}\ x \Leftarrow pure(s)$ 在 $u) \in SN$。

证明。对 $(|K|, (s, u, K))$ 按 $(>, (\dot{\succ}_{lex}))$进行归纳。接下来证明 $K@\,\mathsf{let}\ x \Leftarrow pure(s)$ 在 u 的所有归约结果都属于 SN。

- 情形 $K'@\ \mathsf{let}\ x \Leftarrow pure(s)\ \mathsf{in}\ F[u]$，其中 $K' \circ F = K$，由 let-结合可得。要应用归纳假设（IH），我们需要证明 K 和 $F[u]$ 满足引理的前提，即 $K'@(F[u]\{s/x\}) \in SN$。注意，由 let-结合的侧条件，$K'@F[u] = K@u$ 和 x 不能是 F 中的自由变量，因此 $K'@(F[u]\{s/x\}) = K'@(F[u\{s/x\}])$。此外，$|K'| < |K|$，所以度量值减小。情形 该归约结果为 $K@u\{s/x\}$。根据假设。情形 该归约结果为 $K@\,\mathsf{let}\, x \Leftarrow pure(s')\ \mathsf{in}\ u$，其中 $s \dot{\succ} s'$。由归纳假设。情形 该归约结果为 $K@\,\mathsf{let}\, x \Leftarrow pure(s)\ \mathsf{in}\ u'$，其中 $u \dot{\succ} u'$。由归纳假设。情形 该归约结果为 $K'@\,\mathsf{let}\, x \Leftarrow pure(s)\ \mathsf{in}\ u$，其中 $K \dot{\succ} K'$。由归纳假设。◀

▶ **引理 12**。若 $s \in$ 可约项$^{\mathsf{E}(S)}$ 且 u 满足：对于所有 $s' \in$ 可约项$_S$，都有 $u\{s'/x\} \in$ 可约项$^{\mathsf{E}(T)}$，那么 $\mathsf{let}\ x \Leftarrow s\ \mathsf{in}\ u \in$ 可约项$^{\mathsf{E}(T)}$。

证明。我们证明，对于任意 $K \in$可约项$_T^{\top}$，都有 $SN(K@\ \mathsf{let}\ x \Leftarrow s\ \mathsf{in}\ u)$。首先，我们证明 $K' = K \circ (\mathsf{let}\ x \Leftarrow [\ ]\ \mathsf{in}\ u) \in$可约项$_S^{\top}$，换言之，就是对于任意 $s'' \in$ 可约项$_S$，都有 $SN(K@\ \mathsf{let}\ x \Leftarrow pure(s'')\ \mathsf{in}\ u)$。这由引理 11 得出（根据假设，$u\{s''/x\} \in$可约项$_T$，并且进一步有$K'@(u\{s''/x\}) \in SN$，正如引理 11 所要求的）。现在，根据 $s \in$ 可约项$_S$的定义可知，$K'@s = K@\ \mathsf{let}\ x \Leftarrow s\ \mathsf{in}\ u \in SN$。◀

下一个引理展示了未加点的 $\succ$，也就是原始 RPO 关系的一个性质；在后续的一些归纳证明中，它将作为子程序使用。由于这是对归纳步骤的提取，因此它以“引理假设”的形式陈述；在使用它的情形中，该假设将与某个外层归纳假设相对应。

为了使引理具有适当的普遍性，我们定义 上下文 来涵盖各种可放置项以证明其可约性的环境：

$$C \quad ::= \quad K \mid [\ ]t \mid [\ ]$$

并用 $C[s]$ 表示将项填入上下文：

$$C[s] = \begin{cases} K@s & \text{when } C = K \\ st & \text{when } C = [\ ]t \\ s & \text{when } C = [\ ] \end{cases}$$

▶ **引理 13** (递归路径排序步骤)。 给定某个上下文 C，以及 $\gamma(s_1, \ldots\ldots, s_n) = s \succ t$，其中每个$C[s_i] \in SN$，以及一个“引理假设”，即

- 对于任意一个$(\gamma', \vec{t})$，若它在$(\gamma, \vec{s})$ 之上，且在$(\gamma', \vec{t})$ 之下（按$((>_\Sigma), (\dot{\succ}_{lex}))$的字典序），则有 $C[\gamma'(t_1, \ldots, t_n)] \in SN$,

那么 $C[t] \in SN$。

证明。 我们通过对 $C[t]$ 的大小作归纳，证明其是强正规化的 t，并根据证明 $s \succ t$ 的递归路径排序规则分情况讨论：

- 情形 (1) $t = \gamma(t_1, \ldots, t_n)$。首先证明 $C[t_i]$ 是强正规化的，这由归纳假设得出（注意到 t_i 小于 t）。接着根据引理假设， $C[t] = C[\gamma'(t_1, \ldots, t_n)]$ 是强正规化的。满足引理假设所要求的序关系时， γ保持不变，而递归路径排序规则给出了 $\vec{s} \succ_{lex} \vec{t}$，进而推出 $\vec{s} \dot{\succ}_{lex} \vec{t}$。另请注意，归纳时引理假设本身仍然成立。情形 (2) $t = \gamma'(t_1, \ldots, t_m)$ 和 $\gamma >_\Sigma \gamma'$。首先证明 $C[t_i]$ 是强正规化的，这由归纳假设得出（注意到 t_i 小于 t）。接着根据引理假设，$C[t] = C[\gamma'(t_1, \ldots, t_m)]$ 是强正规化的。 $\gamma >_\Sigma \gamma'$满足引理假设。情形 (3) $s_i \succeq t$。此处 $C[t]$ 位于 $C[s_i]$的归约树中，而我们已假设它是强正规化的，因此 $C[t] \in SN$。◀

▶ **引理 14**。 设 $s = \gamma(s_1, \ldots, s_n)$。给定上下文 C，若每个 s_i都具有 $C[s_i] \in SN$，则 $C[s] \in SN$。

证明。 按 s 的类型分类。在每种情况下，我们考察可能的项 $C[s]$，并证明它们的归约结果全都是强正规化的，因此 $C[s]$ 也是如此。

- 情形 B。必有 $C = [\,]$，因此我们只需考察项 s 本身。对按字典序排列的元组 $(\gamma, \vec{s})$ 进行归纳，排序依据为

  $((>_\Sigma), (\dot{\succ}_{lex}))$。

  按 s 的归约结果分类：
  - 情形 $\gamma(s_1, \ldots, s'_i, \ldots, s_n)$，其中索引为 i，且 $s_i \dot{\succ} s'_i$。由于 γ 保持不变，而 $\vec{s}$ 在 $\dot{\succ}_{lex}$ 下减小，因此适用内部归纳假设。情形 t，其中 $s \succ t$。（在未加点的 $\succ$ 关系下。）适用引理 13。我们的内部归纳假设蕴含引理 13 的“引理假设”。
- 情形 $T_1 \to T_2$。上下文 C 要么是 $[\,]$，要么是 $[\,]s'$。证明 ss' 是强正规化，即可涵盖仅针对 s 的所有情形，因此我们只需证明这些情形。对 $(s', \gamma, \vec{s})$ 按 $(\dot{\succ}, >_\Sigma, \dot{\succ}_{lex})$ 所确定的顺序进行字典序归纳。
  - 情形：一个子项发生归约：要么是某个 $s_i \dot{\succ} s'_i$，要么是被应用项 $s' \dot{\succ} s''$。根据归纳假设，归约序列递减。情形 $s \succ t$。引理 13 适用。我们的归纳假设蕴含“引理假设”。
- 情形 E(T')。这里 $C = K$。由于 $K@s_i \in SN$，我们知道 K 本身是强正规化的。我们对元组进行字典序归纳

  $(|K|, K, \gamma, \vec{s})$ ordered by $(>, \dot{\succ}, >_\Sigma, \dot{\succ}_{lex})$.

  我们证明 $K@s = K@\gamma(s_1, \ldots, s_n)$ 的每个归约结果都是强正规化的，因此该项本身也是强正规化的。按这些归约结果分情况讨论：
  - 情形 $K@\gamma(s_1, \ldots, s'_i, \ldots, s_n)$，其中索引为 i，且 $s_i \dot{\succ} s'_i$。归纳假设适用： K 和 γ 保持不变，而参数项 $\vec{s}$ 在 $\dot{\succ}$ 下按字典序归约。

情形 $K'@s$，其中 $K \dot{\succ} K'$。归纳假设适用，因为续体没有变长（引理 9），且 K' 位于 K 的归约树中。情形 $K'@\gamma(F[s_1], \ldots, F[s_n])$，其中 $K = K' \circ F$，且 γ 是效应符号（效应结合律）。归纳假设适用，因为 K 变短了（如 K'）。我们还需要 $K'@F[s_i]$ 是强正规化的，以满足归纳假设；但 $K'@F[s_i] = K@s_i$ 是我们已知为强正规化的。情形 $K@t$，其中 $s \succ t$。（在无点的 $\succ$ 关系下。）引理 13 适用。◀

▶ **引理 15.** 对于可重写符号 $\gamma\colon S_1 \times \cdots \times S_n \to T$，若每个 $s_i \in$ 可约项$_{S_i}$，则$s = \gamma(s_1, \ldots, s_n) \in$ 可约项 $_T$。

证明。 按 $s.$ 的类型分类讨论。

情形 B。要证明 $s \in$ 可约项$_B$，只需证明 $s \in SN$，而由引理 14 可得这一点。情形 $T_1 \to T_2$。要证明对于任意 $s' \in$ 可约项$_{T_1}$，$ss' \in$ 可约项$_{T_2}$。由于 ss' 是中性项，我们只需证明 ss' 的所有归约结果都可约。对按 $((>_\Sigma, \dot{\succ}_{lex}) \dot{\succ}_{lex})_{lex}$ 排序的 $((\gamma, \vec{s})\ s')$ 作归纳。ss' 的归约只可能发生在 s 中或 s' 中。若发生在 s' 中，归纳假设即可。若发生在 s 中，则有两种可能：情形是该归约具有以下形式 $s = \gamma(s_1, \ldots s_i, \ldots, s_n) \dot{\succ} \gamma(s_1, \ldots s_i', \ldots, s_n)$，其中$s_i \dot{\succ} s_i'$，此时归纳假设即可。情形 $s \succ t$。若它通过 $s \succ t = \gamma'(t_1, \ldots, t_m)$ 归约，且 $(\gamma, \vec{s}) >_{RPO} (\gamma', \vec{t})$，则归纳假设可证得结论。另一方面，若它是 $s_i \succeq t$，则根据 $s_i \in$可约项$_{S_i}$ 的假设，$t \in$可约项$_{S_i}$。

情形 E(T')。要证明对于任意 $K \in$可约项$^{\top}_{T'}$，$K@s \in SN$。由于 $s_i \in$可约项$_{S_i}$，我们有$K@s_i \in SN$，这满足引理 14 的前提，因此 $K@s \in SN$，如所需。◀

记 $s\{\vec{s}/\vec{x}\}$ 表示同时将每个 s_i 代入 s 中的自由变量x_i的操作：$s\{\vec{s}/\vec{x}\} = s\{s_1/x_1, \ldots, s_n/x_n\}$。

▶ **引理 16**（可约性）。给定 $x_1\colon S_1$，……，$x_n\colon S_n$ ‘ $t\colon T$，对于所有 $\vec{s} \in$可约项$_{\vec{S}}$，有$t\{\vec{s}/\vec{x}\} \in$ 可约项$_T$。

证明。 对 t进行结构归纳。

情形 x_i。于是 $S_i = T$。现在 $s_i \in$可约项$_{S_i}$且 $x_i\{s_i/x_i\} = s_i \in$可约项$_{S_i} =$可约项$_T$。

情形 $s't'$。由引理 7 和归纳假设立即可得。情形 $\lambda x.u$。类型推导具有 $\Gamma, x\colon S$ ‘ $u\colon T'$。

设 s' 属于可约项$_S$。根据归纳假设，使用扩展了 s' 的 Γ，我们有 $u\{\vec{s}/\vec{x}, s'/x\} = u\{\vec{s}/\vec{x}\}\{s'/x\} \in$可约项$_T$，进而由引理 8，$\lambda x.u \in$ 可约项$_{S \to T'}$。情形 pure(M)。由引理 10 和归纳假设立即可得。情形 $\gamma(t_1, t_2, ..., t_n)$。由引理 15 和归纳假设立即可得。情形 let $x \Leftarrow t'$ in u。类型推导满足 Γ ‘ $t'\colon S$ 且 $\Gamma, x\colon S$ ‘ $u\colon T$。根据归纳假设，我们有 $t'\{\vec{s}/\vec{x}\} \in$ 可约项$_S$，然后 $u\{t'/x\}\{\vec{s}/\vec{x}\} = u\{\vec{s}/\vec{x}, t'\{\vec{s}/\vec{x}\}/x\} \in$ 可约项$_T$，由此得到引理 15。◀

7 重新审视示例

那么，这一切能带给我们什么？我们能否利用这种技术证明某些有趣演算系统的终止性?

全局状态

回顾一下前面给出的全局状态重写规则：

$$
\begin{aligned}
assign_i(get(t_1, \ldots, t_n)) &\rightsquigarrow assign_i(t_i) \\
assign_i(assign_j(s)) &\rightsquigarrow assign_j(s) \\
get(t_1, \ldots, t_i, \ldots, t_n) &\rightsquigarrow get(t_1, \ldots, s_i, \ldots, t_n) \\
&\qquad \text{where } t_i = get(s_1, \ldots, s_n)
\end{aligned}
$$

Th这些很容易证明是正规化的：每个只需使用一次或两次 RPO(3) 情形 e.

非确定性

$$or(or(s,t),u)) \rightsquigarrow or(s, or(t,u))$$

只有一个效应符号时， $>_\Sigma$ 关系为空。但我们会检查唯一规则是否满足递归路径排序条件。由于函数符号匹配，我们使用 RPO(1)。我们必须检查参数项的字典序： $or(s,\ t) \succ s$ （根据 RPO(3)），因此无需检查第二个参数。然后，右侧的每个参数项都必须小于左侧的整个项，这很容易做到（递归应用 RPO(3) 或 RPO(1)）。请注意，字典序对于确保这条规则推动终止性至关重要。

并行性

对于并行性，令 $par >_\Sigma e$ 对每个其他效应符号 e 成立。这样便可以进行递归比较。例如请注意，在 RPO(1) 下， $par(e(s_1、\ldots\ldots、\ s_n)\ t) \succ par(s_i、\ t)$，参数项按字典序递减，并且 s_i 和 t 均可作为子项找到。

请求重试

重述重写规则：

$$
\begin{aligned}
retry(zero(), request(t, s_1, \ldots, s_n)) &\rightsquigarrow t \\
retry(succ(u), request(t, s_1, \ldots, s_n)) &\rightsquigarrow request(retry(u, t'), s_1, \ldots, s_n) \\
&\qquad \text{where } t' = request(t, s_1, \ldots, s_n)
\end{aligned}
$$

令 重试 $>_\Sigma$请求。第一条规则的左侧包含 t 作为子项，因此它属于 RPO。第二条规则包含 重试 $>_\Sigma$ 请求，接下来我们需要证明

$$retry(succ(u), request(t, s_1, \ldots, s_n)) \succ retry(u, t')$$

并且每个 s_i 都有

$$retry(succ(u), request(t, s_1, \ldots, s_n)) \succ s_i.$$

后者很容易通过子项规则得出。对于前者，首符号相匹配，而直接子项 u 和 t' 都能在左侧找到，作为其子项。

8 相关工作

Johann 等人 [8] 提出了“一种适用于代数效应的通用操作元理论”。作者采用计算树或轨迹，这类对象是在没有方程时代数效应系统的范式。针对每种效应系统，都定义了计算树之间的一种等价关系（实际上是一种预序）；不过，这种关系是通过单独的定义给出的，该定义模拟了每种效应在单独的状态表示（一种抽象机）上的运作。相比之下，我们探讨了当效应可以通过对效应符号本身进行重写来定义时会发生什么。Gavazzo 和 Faggian [6]阐释了重写系统中的单子效应。这项工作将重写关系本身 视为单子的／带效应的，因此，例如，重写关系的可能右侧项可以带有概率分布。

我们的目标是证明，一组重写规则可以按任意顺序应用，并且仍然能够正规化。但也可以选择一种特定的重写或正规化策略。求值归一化就是这样一种方法，Ahman 和 Staton [1] 已经展示了如何在带有代数效应和顺序形式（类似于我们的 let）的演算上执行求值归一化。

我们借鉴重写规则序的悠久历史来证明终止性。Dershowitz [5]给出了最初的递归路径序以及终止性证明。该证明与可约性方法完全不同；由于 let-结合规则带来的困难，后者是必需的。Okada [14] 是我们当前工作的关键人物，因为这是第一篇展示一般性证明的论文：任意强正规化重写系统与简单类型λ演算的语法结合后，仍然具有强正规化性。展示重写与其他语法特征之间的这种正交性，正是本工作的精神所在。

9 未来工作

到目前为止，我们只对现有的强正规化系统做出了有限的改进。系统中的符号各自有自己的重写规则，但它们与 let 的交互方式仅限于一种：从主语位置移出。我们希望能为这样的系统给出类似的强正规化证明：其中重写系统可以规定与 let 的更多交互方式（尽管可能仍需遵守一些限制）。我们为这项技术设定的宏大“测试用例”是 Cooper [3] 以及 Ricciotti 和 Cheney [18] 的系统：如果我们只通过符号序就能证明这些系统是强正规化的，就算取得了成功。

致谢

感谢 Sam Lindley、Matija Pretnar 和 Wilmer Ricciotti 提供宝贵意见，帮助塑造了这项工作。

参考资料

1 Danel Ahman 和 Sam Staton。求值归一化与代数效应。《电子理论计算机科学笔记》，298:51–69，2013年。2 Andrej Bauer。代数效应与处理器中的“代数”指的是什么？2018年。URL： https://arxiv.org/abs/1807.05923， doi:10.48550/ARXIV.1807.05923。3 Ezra Cooper。脚本编写者的梦想：如何编写出色的 SQL，并确保它能成功运行。国际数据库编程语言研讨会，2009年。4 Ezra Cooper、Sam Lindley、Philip Wadler 和 Jeremy Yallop。Links：无分层的 Web 编程。载于 国际组件与对象形式化方法研讨会，第 266–296 页。施普林格，2007年。

5 Nachum Dershowitz。项重写系统的序。理论计算机科学，17(3):279–301，1982年。6 Francesco Gavazzo 和 Claudia Faggian。单子重写系统的关系理论，第一部分。载于 2021年第36届 ACM/IEEE计算机科学逻辑研讨会（LICS），第 1–14 页。IEEE，2021年。7 Jean-Yves Girard、Paul Taylor 和 Yves Lafont。证明与类型。剑桥大学出版社，1989年。8 Patricia Johann、Alex Simpson 和 Janis Voigtländer。代数效应的通用操作元理论。载于 2010年第25届 IEEE计算机科学逻辑研讨会，第 209–218 页。IEEE，2010年。9 Jean-Pierre Jouannaud 和 Albert Rubio。高阶递归路径序。载于《计算机科学逻辑研讨会论文集》第14届会议，第 402–411 页。IEEE，1999年。10 Paul Blain Levy、John Power 和 Hayo Thielecke。按值调用编程语言中的环境建模。信息与计算，185(2):182–210，2003年。11 Sam Lindley 和 Ian Stark。可约性与计算类型的 ⊤⊤-提升。载于《类型化λ演算及其应用国际会议》，第 262–277 页。施普林格，2005年。12 Erik Meijer、Brian Beckman 和 Gavin Bierman。LINQ：在 .NET 框架中协调对象、关系和 XML。载于 2006年 ACM SIGMOD数据管理国际会议论文集，第 706–706 页，2006年。13 Eugenio Moggi。计算λ演算与单子。爱丁堡大学计算机科学系计算机科学基础实验室，1988年。14 Mitsuhiro Okada。类型化λ演算与任意收敛项重写系统组合系统的强正规化。载于 ACM-SIGSAM 1989年国际符号与代数计算研讨会论文集，第 357–363 页，1989年。15 Gordon Plotkin 和 John Power。代数效应的充分性。载于《软件科学与计算结构基础国际会议》，第 1–24 页。施普林格，2001年。16 Gordon Plotkin 和 John Power。计算概念决定单子。载于《软件科学与计算结构基础国际会议》，第 342–356 页。施普林格，2002年。17 Gordon Plotkin 和 Matija Pretnar。代数效应逻辑。载于 2008年第23届 IEEE计算机科学逻辑研讨会，第 118–129 页。IEEE，2008年。18 Wilmer Ricciotti 和 James Cheney。强正规化的高阶关系查询。载于 Zena Ariola（编），第5届《计算与演绎的形式结构》国际会议，第 28:1–28:22 页。达格施图尔城堡，2020年。19 Amr Sabry 和 Philip Wadler。对按值调用的反思。ACM编程语言与系统汇刊，19(6):916–941，1997年。20 Limsoon Wong。集合类型上查询语言的范式与保守性。载于第十二届 ACM SIGACT-SIGMOD-SIGART数据库系统原理研讨会论文集，第 26–36 页，1993年。