单子表达式及其导数

Samira Attou[1], Ludovic Mignot[2], Clément Miklarz[2], 和 Florent Nicart[2]

[1] 古斯塔夫·埃菲尔大学，笛卡尔大道5号 — 马恩河谷尚叙马恩77454号，马恩河谷 Cedex 2[2] GR[2]IF，诺曼底鲁昂大学，大学大道，76801 鲁夫赖圣艾蒂安，法国 samira.attou@univ-eiffel.fr, {ludovic.mignot,clement.miklarz1, florent.nicart}@univ-rouen.fr

arXiv:2301.13054v1 [cs.FL] 30 Jan 2023

摘要。 我们提出了对正则表达式中由 Brzozowski、Antimirov 或 Lombardy 和 Sakarovitch 提出的众所周知的导数计算进行另一种解释，以便使用单子的概念抽象底层数据结构（例如集合或线性组合）。作为这一推广优势的示例，我们首先介绍一种基于分次模单子的新推导技术，然后展示如何应用该技术来推广带有捕获组和反向引用的表达式解析。

我们还将定义表达式的运算符扩展到值集上的任意 n-元函数，例如经典运算（如布尔权重上的否定或交）或更为特殊的运算（如有理数权重上的代数平均）。此外，我们还介绍了如何利用 Colcombet 和 Petrisan 对自动机的范畴定义，从这种扩展表达式计算出一个（不一定有限的）自动机。这些范畴论概念使我们能够以统一的方式完成这一构造，而不论底层使用的是哪一种单子。

最后，为了展示我们的工作，我们使用函数式编程的高级技术实现了这些概念的 Haskell 版本，并提供了一个用于操作具体示例的网页界面。

1 引言

本文是 [2] 的扩展版本。

正则表达式是表示单词与值集之间关联的一种经典方式。例如，经典正则表达式表示单词集合，而带重数的正则表达式表示形式级数。对于一个正则表达式，可以通过求解成员关系测试（确定某个单词是否属于所表示的语言）或权重测试（确定某个单词在所表示的形式级数中的权重）来进行处理；根据克莱尼定理 [11,17] ，这可以通过计算一个有限自动机来完成，例如位置自动机 [9,3,5,6]。

另一类解决这些成员关系测试的方法是导数计算，它不需要构造完整的自动机。这些技术的共同点是将对任意字的测试转化为对空字的测试，而后者可以通过纯句法方式轻松解决（即 对表达式的结构进行归纳）。Brzozowski [4] 展示了如何根据正则表达式 E 和字 w计算出一个正则表达式 $d_w(E)$，它表示这样一组字： w' 使得 ww' 属于由 E表示的语言。因此，成员关系测试就转化为判断空字是否属于表达式 $d_w(E)$ 所表示的语言。Antimirov [1]对这种方法进行了改进，使其生成表达式集合而非单个表达式；即 将偏导数$\partial_w(E)$定义为一个表达式集合，其和表示的语言与 $d_w(E)$ 相同。最坏情况下，导数的数量相对于 E 的长度 $|E|$ 呈指数级增长[3], ，而偏导数至多生成相对于 $|E|$呈线性数量增长的表达式。Lombardy 和 Sakarovitch [13] 将这些方法扩展到带重数的表达式。最后，Sulzmann 和 Lu [18] 将这些推导技术应用于 POSIX 表达式的解析。

众所周知，这些方法都基于同一种运算，即语言商。此外，Antimirov 方法可以解释为在布尔半环中对带重数的正则表达式进行推导。然而，Brzozowski 计算并不会产生与 Antimirov 方法相同的表达式（即 语法树上的相等性）。

主要贡献： 本文将范畴论的概念应用于集合范畴，实现了对这些计算的统一，并展示了如何计算 [7], 中定义的范畴自动机，这一过程是通过重新诠释 [15]中开始的工作实现的。我们利用经典单子对众所周知的导数计算进行建模。此外，我们以一种一般化的方式处理扩展表达式：在本文中，表达式可以支持扩展

[3] 就和的结合律、交换律和幂等律规则而言，否则可能是无限的。

运算符，例如补、交，以及任意 n元函数（代数平均、极值乘法、等等）。与 [15] 的主要区别在于，我们以一种关于底层单子的内在方式，形式化地阐述了这些表达式所表示的语言和级数。

更确切地说，本文提出：

– 将表达式扩展到值集上的任意 n 元函数，– 对表达式进行单子化推广，– 为这些表达式的成员关系/权重测试提供一种解决方案，– 计算范畴导数自动机，– 提出一种适用于扩展到 n 元函数的新的单子，– 通过使用高级函数式编程的 Haskell 实现进行示例说明，– 将表达式扩展到捕获组和反向引用。

动机： 统一推导技术本身就是一个目标。此外，用于实现这一统一的形式化工具也很有用：单子兼具理论和实践优势。事实上，从理论角度来看，这些结构能够抽象出各种性质，并聚焦于解决成员关系和权重问题的核心机制。此外，引入一些特殊的单子也有助于研究导出项的有限性。从实践角度来看，单子易于实现（即使是在 Haskell 以外的一些语言中），并能帮助我们编写简洁且安全的代码。最后，借助单子变换器 [10] 等概念，我们可以轻松组合不同的代数结构，或添加一些技术功能（捕获组、日志记录、非确定性等）。本文将探讨这些概念。

本文的结构如下：第 2 节汇集了一些预备材料，例如代数结构或范畴论中的概念。我们还介绍了一些 Haskell 社区熟知的函数，它们能够帮助我们缩减方程的规模。随后，我们在第 3 节中对所处理的表达式、相关级数以及空字的权重检验进行了结构化定义。为了将这一检验扩展到任意字，我们首先在第 4 节中陈述所考虑的单子所需的一些性质。在确定了这个所谓的支撑之后，我们将在第 5 节中说明如何计算导数。第 6 节解释导数自动机的计算。第 7 节给出一个新的单子及其相关的导数计算。第 8 节介绍一个实现。最后，在第 9 节中，我们展示如何（作为 [18] 的替代方式）计算捕获组表达式的导数，并说明对于相同的运算符，无论底层单子是什么，导数公式都是相同的。

2 预备知识

我们用 $S \to S'$ 表示从 Set 范畴 S 到 Set 范畴 S' 的函数集合。记号 $\lambda x \to f(x)$ 是函数 f 的等价记号。

一个 幺半群 是一个 Set 范畴 S，其配备了结合运算和一个单位元。一个 半环 是一个结构 $(S, \times, +, 1, 0)$，满足 $(S, \times, 1)$ 是一个幺半群，$(S, +, 0)$ 是一个交换幺半群，$\times$ 对 $+$ 满足分配律，且 0 是 $\times$的湮灭元。一个 带星半环 是一个半环，并带有一个一元函数 * 使得

$$k^\star = 1 + k \times k^\star = 1 + k^\star \times k.$$

一个 $\mathbb{K}$-级数 定义在与字母表 Σ 相关联的自由幺半群 $(\Sigma^*, \cdot, \varepsilon)$ 上，取值于半环 $\mathbb{K} = (K, \times, +, 1, 0)$，是一个从 Σ^* 到 K 的函数。$\mathbb{K}$-级数 的 Set 范畴可以按如下方式赋予半环结构：

$$1(w) = \begin{cases} 1 & \text{if } w = \varepsilon, \\ 0 & \text{otherwise,} \end{cases} \qquad 0(w) = 0,$$

$$(S_1 + S_2)(w) = S_1(w) + S_2(w), \qquad (S_1 \times S_2)(w) = \sum_{u \cdot v = w} S_1(u) \times S_2(v).$$

此外，如果 $S_1(\varepsilon) = 0$（即 S_1 被称为真正规），则 S_1 的星号 是由下式定义的级数

$$(S_1)^\star(\varepsilon) = 1, \qquad (S_1)^\star(w) = \sum_{n \le |w|, w = u_1 \cdots u_n, u_j \neq \varepsilon} S_1(u_1) \times \cdots \times S_1(u_n).$$

最后，对于 $K^n \to K$ 中的任意函数 f，我们规定：

$$(f(S_1, \ldots, S_n))(w) = f(S_1(w), \ldots, S_n(w)). \tag{1}$$

一个函子[4] F 关联 s 与每个集合 Sa 集合 $F(S)$ 以及每个函数 $f$in $S \to S'$a 函数 $F(f)$ fro m $F(S)$ 到 $F(S')$，使得

$$F(\mathrm{id}) = \mathrm{id}, \qquad F(f \circ g) = F(f) \circ F(g),$$

[4] 更确切地说，是集合范畴的一个子范畴上的函子。

其中 id 是恒等函数，$\circ$ 是经典函数复合。
一个 单子[5] M 是一个函子，带有两族函数

– pure，从集合 S 到 $M(S)$，–bind，将 $S \to M(S')$ 中的任意
函数 f映射到 $M(S) \to M(S')$,

满足以下三个条件：

$$\texttt{bind}(f)(\texttt{pure}(s)) = f(s), \quad \texttt{bind}(\texttt{pure}) = \mathrm{id},$$
$$\texttt{bind}(g)(\texttt{bind}(f)(m)) = \texttt{bind}(\lambda x \to \texttt{bind}(g)(f(x)))(m).$$

例 *1.* Maybe 单子将以下内容关联起来：

– 任意集合 S 与集合 Maybe(S) $=$ {Just(s) $\mid s \in S$} $\cup$ {Nothing}，其中 Just 和 Nothing 是两个语法令牌，可用于为集合添加一个值；
– 任意函数 f 与函数 Maybe(f) 相关联，该函数定义为 Maybe(f)(Nothing) $=$ Nothing
$\texttt{Maybe}(f)(\texttt{Just}(s)) = \texttt{Just}(f(s))$,
– 配备了由下式定义的函数 pure 和 bind：

$$\texttt{pure}(s) = \texttt{Just}(s), \qquad \begin{array}{l}\texttt{bind}(f)(\texttt{Just}(s)) = f(s),\\ \text{bind}(f)(\text{Nothing}) = \text{Nothing}。\end{array}$$

例 *2.* Set 单子关联：

– 对任意集合 S 关联集合 2^S，–对任意函数 f关联由 Set(f) 定义的函数 Set(f)
(R) $= \bigcup_{r \in R}\{f(r)\}$，– 配备了由下式定义的函数 pure 和 bind：

$$\texttt{pure}(s) = \{s\}, \qquad \texttt{bind}(f)(R) = \bigcup_{r \in R} f(r).$$

例 *3.* LinComb($\mathbb{K}$) 单子，对于 $\mathbb{K} = (K, \times, +, 1, 0)$，关联：

– 对任意集合 S，元素 S 的 $\mathbb{K}$-线性组合所构成的集合，其中线性组合是 $K \times S$ 中若干对（记作 $\boxplus$）的有限（形式上的、可交换的）和，满足 $(k, s) \boxplus (k', s) = (k + k', s)$；– 对任意函数 f，函数 LinComb $\mathbb{K}$ 由.....定义
$(\quad)(f)$
$$\texttt{LinComb}(\mathbb{K})(f)(R) = \bigboxplus_{(k,r)\in R} (k, f(r)),$$
– 配备由以下方式定义的函数 pure 和 bind：

$$\texttt{pure}(s) = (1, s), \qquad \texttt{bind}(f)(R) = \bigboxplus_{(k,r)\in R} k \otimes f(r),$$

where $k \otimes R = \bigboxplus_{(k',r)\in R} (k \times k', r).$

为使方程更简洁，对于任意单子 M，我们使用以下运算符：

$$f \texttt{<\$>} s = M(f)(s), \qquad m \texttt{>>=} f = \texttt{bind}(f)(m).$$

如果 <$> 可用于将一元函数提升到单子层级，那么 >>= 和 pure 可用于将任意 n 元函数f在 $S_1 \times \cdots \times S_n \to S$ 中提升，定义一个函数 lift$_n$，它将 $S_1 \times \cdots \times S_n \to S$ 映射为 $M(S_1) \times \cdots \times M(S_n) \to M(S)$，如下所示：

$$\begin{aligned}\texttt{lift}_n(f)(m_1, \ldots, m_n) = & m_1 \texttt{>>=} (\lambda s_1 \to \ldots \\ & m_n \texttt{>>=} (\lambda s_n \to \texttt{pure}(f(s_1, \ldots, s_n))) \ldots)\end{aligned}$$

考虑仅含一个元素的集合 $\mathbb{I} = \{\top\}$。通过先前定义的某些单子对该集合进行映射，所得结果可作为值集进行求值；在经典情形下，这些值集用于为与经典正则表达式相关联的单词赋权。例如，Maybe($\mathbb{I}$) 和 Set 范畴($\mathbb{I}$) 与布尔集合是同构的，而任意集合 LinComb($\mathbb{K}$)($\mathbb{I}$) 都可以转换为 $\mathbb{K}$ 的底层集合。这一性质使我们能够以连贯的方式将经典表达式扩展为单子表达式，其权重类型因此由环境单子决定。

[5] 更准确地说，是集合范畴的某个子范畴上的单子。

3 单子表达式

如上一节所述，对于某些特定的单子， $M(\mathbb{I})$ 中的元素可以作为经典值集进行求值。因此，我们使用这些元素不仅表示表达式赋予单词的权重，也表示作用于所表示级数的元素。

下文中，除经典算子（$+$、 $\cdot$ 和 *）外，我们还记号表示：

– 元素对级数的作用记为 $\odot$，– 函数对自身的应用。

定义 1。 设 M 为一个单子。字母表 Σ 上的 M-单子表达式 E 归纳定义如下： E

$$
\begin{array}{lll}
= a, & E = \varepsilon, & E = \emptyset, \\
E = E_1 + E_2, & E = E_1 \cdot E_2, & E = E_1^*, \\
E = \alpha \odot E_1, & E = E_1 \odot \alpha, & E = f(E_1, \ldots, E_n),
\end{array}
$$

其中， a 是 Σ中的一个符号， （E_1， ... ， E_n） 是 n M-单子表达式，定义在 Σ上； α 是 M （$\mathbb{I}$） 的一个元素； f是从（M （$\mathbb{I}$）） ^{n}到 M （$\mathbb{I}$） 的函数。

我们将字母表 Σ上的单子表达式集合记为 $\mathrm{Exp}(\Sigma)$。

例 *4.* 作为可用于扩展经典算子的函数示例，可以定义函数 $\mathrm{ExtDist}(x_1, x_2, x_3) = \max(x_1, x_2, x_3) - \min(x_1, x_2, x_3)$，其定义域为 $\mathbb{N}^3$，值域为 $\mathbb{N}$。

与经典正则表达式类似，单子表达式为任意一个字赋予一个权重。这种关系可以通过形式级数来表示。不过，在定义这一概念之前，为了简化研究，我们只考虑适当表达式。首先说明如何通过计算可空性值来刻画它们。

定义 2. 设 M是一个单子，且其结构 $(M(\mathbb{I}), +, \times, ^\star, 1, 0)$ 是一个带星半环。字母表 Σ上的一个 M-单子表达式 E 的可空性值 是 $M(\mathbb{I})$ 中的元素 $\mathtt{Null}(E)$，其归纳定义如下：

$$
\begin{array}{rr}
\mathtt{Null}(\varepsilon) = 1, & \mathtt{Null}(\emptyset) = 0, \\
\mathtt{Null}(a) = 0, & \mathtt{Null}(E_1 + E_2) = \mathtt{Null}(E_1) + \mathtt{Null}(E_2), \\
\mathtt{Null}(E_1 \cdot E_2) = \mathtt{Null}(E_1) \times \mathtt{Null}(E_2), & \mathtt{Null}(E_1^*) = \mathtt{Null}(E_1)^\star, \\
\mathtt{Null}(\alpha \odot E_1) = \alpha \times \mathtt{Null}(E_1), & \mathtt{Null}(E_1 \odot \alpha) = \mathtt{Null}(E_1) \times \alpha,
\end{array}
$$

$$
\mathtt{Null}(f(E_1, \ldots, E_n)) = f(\mathtt{Null}(E_1), \ldots, \mathtt{Null}(E_n)),
$$

其中， a是 Σ中的一个符号，$(E_1, \ldots, E_n)$是关于 Σ的 n M-单子表达式， α是 $M(\mathbb{I})$ 中的一个元素，而 f是一个从$(M(\mathbb{I}))^n$到 $M(\mathbb{I})$的函数。

当所考虑的半环不是星号半环时，我们将可空性值的计算限制在这样的表达式上：其中星号子表达式具有空可空性值。为了计算它，我们考虑 Maybe 单子，从而优雅地处理这样的部分函数。

定义 3. 设 M是一个单子，使得结构$(M(\mathbb{I}), +, \times, 1, 0)$构成一个半环。字母表 Σ上某个 M-单子表达式 E的部分可空性值是$\mathtt{Maybe}(M(\mathbb{I}))$ 中的元素$\mathtt{PartNull}(E)$，定义如下：

$$
\begin{array}{c}
\mathtt{PartNull}(\varepsilon) = \mathtt{Just}(1), \quad \mathtt{PartNull}(\emptyset) = \mathtt{Just}(0), \quad \mathtt{PartNull}(a) = \mathtt{Just}(0), \\
\mathtt{PartNull}(E_1 + E_2) = \mathtt{lift}_2(+)(\mathtt{PartNull}(E_1), \mathtt{PartNull}(E_2)), \\
\mathtt{PartNull}(E_1 \cdot E_2) = \mathtt{lift}_2(\times)(\mathtt{PartNull}(E_1), \mathtt{PartNull}(E_2)), \\
\mathtt{PartNull}(E_1^*) = \begin{cases} \mathtt{Just}(1) & \textit{if } \mathtt{PartNull}(E_1) = \mathtt{Just}(0), \\ \mathtt{Nothing} & \textit{otherwise,} \end{cases} \\
\mathtt{PartNull}(\alpha \odot E_1) = (\lambda E \to \alpha \times E) \mathbin{<\$>} \mathtt{PartNull}(E_1), \\
\mathtt{PartNull}(E_1 \odot \alpha) = (\lambda E \to E \times \alpha) \mathbin{<\$>} \mathtt{PartNull}(E_1), \\
\mathtt{PartNull}(f(E_1, \ldots, E_n)) = \mathtt{lift}_n(f)(\mathtt{PartNull}(E_1), \ldots, \mathtt{PartNull}(E_n)),
\end{array}
$$

其中 a是 Σ中的一个符号， $E_1, \ldots, E_n$是 Σ上的 n M-单子表达式， α是 $M(\mathbb{I})$ 的一个元素，而 f是一个从 $(M(\mathbb{I}))$到 $M(\mathbb{I})$的函数。

如果表达式 E 的部分可空性值不为 Nothing，那么它就是真正规的；也就是说，如果它是一个值 Just(v)。在这种情况下， v 就是它的可空性值，记作 Null(E)（滥用记法）。

定义4。 设 M为一个单子，使得结构 $(M(\mathbb{I})\ +,\ \times,\ 1,\ 0)$ 是一个半环，并设 E为字母表 Σ上的一个 M-单子适当表达式。与 E相关联的级数 $S(E)$ 按如下方式归纳定义：

$$S(\varepsilon)(w) = \begin{cases} 1 & \text{if } w = \varepsilon, \\ 0 & \text{otherwise}, \end{cases} \quad S(\emptyset)(w) = 0, \quad S(a)(w) = \begin{cases} 1 & \text{if } w = a, \\ 0 & \text{otherwise}, \end{cases}$$

$$S(E_1 + E_2) = S(E_1) + S(E_2), \quad S(E_1 \cdot E_2) = S(E_1) \times S(E_2), \quad S(E_1^*) = (S(E_1))^\star,$$
$$S(\alpha \odot E_1)(w) = \alpha \times S(E_1)(w), \quad S(E_1 \odot \alpha)(w) = S(E_1)(w) \times \alpha,$$
$$S(f(E_1, \ldots, E_n)) = f(S(E_1), \ldots, S(E_n)),$$

其中， a是 中的一个符号 o， $E_1, \ldots,$ M是 上的 n-单子表达式， α是 $M(\mathbb{I})$ 的一个元素，而 f是从 $(M(\mathbb{I}))^n$ 到 $M(\mathbb{I})$ 的函数。

从现在起，我们将 Σ 上的 M-单子表达式的集合 Exp(Σ) 视为一个半环，并规定表示同一个级数的两个表达式相等。由 E 与 w 在 Σ^*中通过 E 所关联的权重是 $\mathtt{weight}_w(E) = S(E)(w)$ 的值。适当表达式的可空性，是它根据定义3和定义4与 ε 关联的权重。

命题 1. 设 M是一个单子，使得其结构 $(M(\mathbb{I})\ +, \times, 1, 0)$ 是一个半环。设 E是 Σ上的一个 M-单子适当表达式。那么：

$$\mathtt{Null}(E) = \mathtt{weight}_\varepsilon(E).$$

前一个命题意味着，空字的权重可以通过句法方式计算（即 从单子表达式中归纳计算得出）。现在，我们来说明如何通过为单子表达式定义导数计算来扩展这一计算。

4 表达式的单子支撑

$\mathbb{K}$-左半模是指：对于半环 $\mathbb{K} = (K, \times, +, 1, 0)$， 一个交换幺半群$(S, \pm, 0)$配备了一个从 $K \times S$ 到 S 的函数$\triangleright$ ，使得：

$$(k \times k') \triangleright s = k \triangleright (k' \triangleright s), \quad (k + k') \triangleright s = k \triangleright s \pm k' \triangleright s,$$
$$k \triangleright (s \pm s') = k \triangleright s \pm k \triangleright s', \quad 1 \triangleright s = s, \quad 0 \triangleright s = k \triangleright \underline{0} = \underline{0}.$$

$\mathbb{K}$-右半模 可以对称地定义。

一个歌剧 [12,14] 是一个结构$(O, (\circ_j)_{j \in \mathbb{N}}, \mathrm{id})$，其中 O 是一个分次集合（即 $O = \bigcup_{n \in \mathbb{N}} O_n$） ，id是$O_1$的一个元素， $\circ_j$ 是一个针对任意三个整数(i, j, k)定义的函数[6]，其中 $0 < j \leq k$属于 $O_k \times O_i \to O_{k+i-1}$ ，并且满足对任意元素 $p_1 \in O_m$, $p_2 \in O_n$, $p_3 \in O_p$：

$$\forall 0 < j \leq m, \mathrm{id} \circ_1 p_1 = p_1 \circ_j \mathrm{id} = p_1,$$
$$\forall 0 < j \leq m, 0 < j' \leq n, p_1 \circ_j (p_2 \circ_{j'} p_3) = (p_1 \circ_j p_2) \circ_{j+j'-1} p_3,$$
$$\forall 0 < j' \leq j \leq m, (p_1 \circ_j p_2) \circ_{j'} p_3 = (p_1 \circ_{j'} p_3) \circ_{j+p-1} p_2.$$

将这些复合 $\circ_j$ 组合起来，i可以定义一个将 $_k$ ×d fi$_{i_1} \times \cdots \times O_{i_k}$ i映射到 $O_{i_1 + \cdots + i_k}$的复合 $\circ$：对于 $O_k \times O^k$ 中的任意元素 $(p, q_1, \ldots, q_k)$，

$$p \circ (q_1, \ldots, q_k) = (\cdots((p \circ_k q_k) \circ_{k-1} q_{k-1} \cdots) \cdots) \circ_1 q_1.$$

反过来，复合 $\circ$ 可以借助单位元定义复合 $\circ_j$：对于 i$O_k \times O_i$ 中的任意两个元素 (p, q)，对于任意整数 $0 < j \leq k$：

$$p \circ_j q = p \circ (\underbrace{\mathrm{id}, \ldots, \mathrm{id}}_{j-1 \text{ times}}, q, \underbrace{\mathrm{id}, \ldots, \mathrm{id}}_{k-j \text{ times}}).$$

例如，集合上所有 n 元函数的集合，以恒等函数作为单位，h构成 id个歌剧。一个 算子系模（O、$\circ$、id）是一个集合 S，它配备了一个函数 $*$，从 $O_n \times S^n$ 到 S，满足

$$f * (f_1 * (s_{1,1}, \ldots, s_{1,i_1}), \ldots, f_n * (s_{n,1}, \ldots, s_{n,i_n}))$$
$$= (f \circ (f_1, \ldots, f_n)) * (s_{1,1}, \ldots, s_{1,i_1}, \ldots, s_{n,1}, \ldots, s_{n,i_n}).$$

[6] 每一对 (i, k) 都能明确地确定一个函数 $\circ_j$ 的定义域和陪域

导数计算的扩展可以针对任意单子进行。事实上，任何单子都可以用来定义类型良好的辅助函数，以模拟经典计算。然而，为了按照定义4等价地计算权重，还应满足一些性质。因此，在下文中我们考虑一类受限的单子。

一个单子支撑是满足以下条件的结构 $(M, +, \times, 1, 0, \pm, 0, \ltimes, \triangleright, \triangleleft, \circledast)$：

– M 是一个单子，– $\mathbb{R} = (M(\mathbb{I}), +, \times, 1, 0)$ 是一个半环，– $\mathbb{M} = (M(\mathrm{Exp}(\Sigma)), \pm, 0)$ 是一个幺半群，– $(\mathbb{M}, \ltimes)$ 是一个 $\mathrm{Exp}(\Sigma)$-右半模，– $(\mathbb{M}, \triangleright)$ 是一个 $\mathbb{R}$-左半模，– $(\mathbb{M}, \triangleleft)$ 是一个 $\mathbb{R}$-右半模，– $(M(\mathrm{Exp}(\Sigma)), \circledast)$ 是 $M(\mathbb{I})$ 上函数所构成的歌剧的一个模。

一个表达性支撑是一个单子支撑 $(M, +, \times, 1, 0, \pm, 0, \ltimes, \triangleright, \triangleleft, \circledast)$，并配备了一个函数 $\mathtt{toExp}$，从 $M(\mathrm{Exp}(\Sigma))$ 映射到 $\mathrm{Exp}(\Sigma)$，满足以下条件：

$$\mathtt{weight}_w(\mathtt{toExp}(m)) = m \gg\!= \mathtt{weight}_w \tag{2}$$
$$\mathtt{toExp}(m \ltimes F) = \mathtt{toExp}(m) \cdot F, \tag{3}$$
$$\mathtt{toExp}(m \pm m') = \mathtt{toExp}(m) + \mathtt{toExp}(m'), \tag{4}$$
$$\mathtt{toExp}(m \triangleright x) = \mathtt{toExp}(m) \odot x, \tag{5}$$
$$\mathtt{toExp}(x \triangleleft m) = x \odot \mathtt{toExp}(m), \tag{6}$$
$$\mathtt{toExp}(f \circledast (m_1, \ldots, m_n)) = f(\mathtt{toExp}(m_1), \ldots, \mathtt{toExp}(m_n)). \tag{7}$$

现在，让我们通过三个表达性支撑来说明这一概念，它们使我们能够对众所周知的导数计算进行建模。

例5（函子支撑。 y ）

$$\mathtt{toExp}(\mathtt{Nothing}) = 0, \quad \mathtt{toExp}(\mathtt{Just}(E)) = E,$$
$$\mathtt{Nothing} + m = m, \qquad \mathtt{Nothing} \times m = \mathtt{Nothing},$$
$$m + \mathtt{Nothing} = m, \qquad m \times \mathtt{Nothing} = \mathtt{Nothing},$$
$$\mathtt{Just}(\top) + \mathtt{Just}(\top) = \mathtt{Just}(\top), \qquad \mathtt{Just}(\top) \times \mathtt{Just}(\top) = \mathtt{Just}(\top),$$
$$\mathtt{Nothing} \pm m = m, \quad m \pm \mathtt{Nothing} = m, \quad \mathtt{Just}(E) \pm \mathtt{Just}(E') = \mathtt{Just}(E + E'),$$
$$1 = \mathtt{Just}(\top), \quad 0 = \mathtt{Nothing}, \quad \underline{0} = \mathtt{Nothing},$$
$$m \ltimes F = (\lambda E \to E \cdot F) \mathbin{<\$>} m,$$
$$m \triangleright m' = m \gg\!= (\lambda x \to m'), \quad m \triangleleft m' = m' \gg\!= (\lambda x \to m),$$
$$f \circledast (m_1, \ldots, m_n) = \mathtt{pure}(f(\mathtt{toExp}(m_1), \ldots, \mathtt{toExp}(m_n))).$$

例6（Set 范畴支撑。 p ）

$$\mathtt{toExp}(\{E_1, \ldots, E_n\}) = E_1 + \cdots + E_n,$$
$$+ = \cup, \quad \times = \cap, \quad \pm = \cup, \quad 1 = \{\top\}, \quad 0 = \emptyset, \quad \underline{0} = \emptyset,$$
$$m \ltimes F = (\lambda E \to E \cdot F) \mathbin{<\$>} m,$$
$$m \triangleright m' = m \gg\!= (\lambda x \to m'), \quad m \triangleleft m' = m' \gg\!= (\lambda x \to m),$$
$$f \circledast (m_1, \ldots, m_n) = \mathtt{pure}(f(\mathtt{toExp}(m_1), \ldots, \mathtt{toExp}(m_n))).$$

例7（$\mathtt{LinComb}\ \mathbb{K}$ 支撑。（ ） pp ）

$$\mathtt{toExp}((k_1, E_1) \boxplus \cdots \boxplus (k_n, E_n)) = k_1 \odot E_1 + \cdots + k_n \odot E_n,$$
$$+ = \boxplus, \quad (k, \top) \times (k', \top) = (k \times k', \top), \quad 1 = (1, \top), \quad 0 = (0, \top),$$
$$\pm = \boxplus, \quad \underline{0} = (0, \top),$$
$$m \ltimes F = (\lambda E \to E \cdot F) \mathbin{<\$>} m,$$
$$m \triangleright m' = m \gg\!= (\lambda x \to m'), \quad m \triangleleft k = (\lambda E \to E \odot k) \mathbin{<\$>} m,$$
$$f \circledast (m_1, \ldots, m_n) = \mathtt{pure}(f(\mathtt{toExp}(m_1), \ldots, \mathtt{toExp}(m_n))).$$

5 单子导数

下文中，$(M, +, \times, \mathbf{1}, \mathbf{0}, \pm, \mathbf{0}, \ltimes, \rhd, \lhd, \circledast, \mathrm{toExp})$ 是一种表达性支撑。

定义 5. M-单子表达式 E 关于 Σ 中字母 a 的导数 是 $d_a(E)$中的元素，在 $M(\mathrm{Exp}(\Sigma))$ 中按以下方式归纳定义：

$$d_a(\varepsilon) = \underline{0}, \quad d_a(\emptyset) = \underline{0}, \quad d_a(b) = \begin{cases} \mathtt{pure}(\varepsilon) & \text{if } a = b, \\ \underline{0} & \text{otherwise,} \end{cases}$$
$$d_a(E_1 + E_2) = d_a(E_1) \pm d_a(E_2), \quad d_a(E_1^*) = d_a(E_1) \ltimes E_1^*,$$
$$d_a(E_1 \cdot E_2) = d_a(E_1) \ltimes E_2 \pm \mathtt{Null}(E_1) \rhd d_a(E_2),$$
$$d_a(\alpha \odot E_1) = \alpha \rhd d_a(E_1), \quad d_a(E_1 \odot \alpha) = d_a(E_1) \lhd \alpha,$$
$$d_a(f(E_1, \ldots, E_n)) = f \circledast (d_a(E_1), \ldots, d_a(E_n))$$

其中，b是 Σ中的一个符号，$E_1, \ldots, E_n$是 Σ上的 n 个 M-单子表达式，α是 $M(\mathbb{I})$ 中的一个元素，而 f 是从$(M(\mathbb{I}))^n$到 $M(\mathbb{I})$的函数。

导数与级数之间的联系可表述如下，这也是对经典商的一种替代性描述。

命题 2. 设 E 是字母表 Σ上的一个 M-单子表达式，a 是 Σ 中的一个符号，w 是 Σ^*中的一个字。那么：

$$\mathtt{weight}_{aw}(E) = d_a(E) \gg\!= \mathtt{weight}_w.$$

证明. 我们对 E的结构进行归纳。所有经典情形（即 暂不考虑函数算子）都可以用经典方法（[1,4,13]）证明。因此，我们考虑最后一种情形。d

$$\begin{aligned}
&{}_a(f(E_1, \ldots, E_n)) \gg\!= \text{权重}_w \\
&\quad = \mathtt{weight}_w(\mathtt{toExp}(d_a(f(E_1, \ldots, E_n)))) && \text{(Eq (2))} \\
&\quad = \mathtt{weight}_w(\mathtt{toExp}(f \circledast (d_a(E_1), \ldots, d_a(E_n))) && \text{(Def 5))} \\
&\quad = \mathtt{weight}_w(f(\mathtt{toExp}(d_a(E_1)), \ldots, \mathtt{toExp}(d_a(E_n)))) && \text{(Eq (7))} \\
&\quad = f(\mathtt{weight}_w(\mathtt{toExp}(d_a(E_1))), \ldots, \mathtt{weight}_w(\mathtt{toExp}(d_a(E_n)))) && \text{(Def 4, Eq (1))} \\
&\quad = f(d_a(E_1) \gg\!= \mathtt{weight}_w, \ldots, d_a(E_n) \gg\!= \mathtt{weight}_w) && \text{(Eq (2))} \\
&\quad = f(\mathtt{weight}_{aw}(E_1), \ldots, \mathtt{weight}_{aw}(E_n)) && \text{(归纳假设)} \\
&\quad = \text{权重}_{aw}(f(E_1, \ldots, E_n)) && \text{(Def 4, Eq (1))}
\end{aligned}$$

下面定义如何使用单子函数将导数计算从符号扩展到单词。

定义 6. M-单子表达式 E 在 Σ 上关于 w 中的一个字 Σ^* 的导数 是 $d_w(E)$ 中的元素 $M(\mathrm{Exp}(\Sigma))$，按如下方式递归定义：

$$d_\varepsilon(E) = \mathtt{pure}(E), \qquad d_{a \cdot v}(E) = d_a(E) \gg\!= d_v,$$

其中 a 是 Σ 中的一个符号，v是 Σ^* 中的一个字。

最后，按照命题 2，通过对单词长度进行归纳，可以很容易地证明，导数计算可用于定义一种语法计算，用来计算与某个表达式相关联的单词的权重。

定理 1 . 设 E 是字母表 Σ 上的一个 M-单子表达式，且 w 是 Σ^* 中的一个字。

$$\mathtt{weight}_w(E) = d_w(E) \gg\!= \mathtt{Null}.$$

注意，将单子表达式限制为正则表达式时，

– Maybe 支持导致经典导数 [4]，– Set 支持导致偏导数 [1]，– LinComb 支撑导致带重数的导数 [13]。

例8. 让我们考虑例4中定义的函数 ExtDist，以及 LinComb($\mathbb{N}$)-单子表达式$E = \mathrm{ExtDist}(a^*b^* + b^*a^*, b^*a^*b^*, a^*b^*a^*)$。

$$d_a(E) = \mathtt{ExtDist}(a^*b^* + a^*, a^*b^*, a^*b^*a^* + a^*)$$
$$d_{aa}(E) = \mathtt{ExtDist}(a^*b^* + a^*, a^*b^*, a^*b^*a^* + 2 \odot a^*)$$

$$
\begin{aligned}
d_{aaa}(E) &= \texttt{ExtDist}(a^*b^* + a^*, a^*b^*, a^*b^*a^* + 3 \odot a^*) \\
d_{aab}(E) &= \texttt{ExtDist}(b^*, b^*, b^*a^*) \\
\texttt{weight}_{aaa}(E) &= d_{aaa}(E) \gg\!= \texttt{Null} \\
&= \texttt{ExtDist}(1+1, 1, 1+3) = 4 - 1 = 3 \\
\texttt{weight}_{aab}(E) &= d_{aab}(E) \gg\!= \texttt{Null} = \texttt{ExtDist}(1,1,1) = 0
\end{aligned}
$$

在下一节中　节中，我们展示如何计算与某个表达式相关联的导数自动机　.

6 自动机构造

一个范畴 $\mathcal{C}$ 定义如下：

– 一个对象类 Obj$_\mathcal{C}$；– 对任意两个对象 A 和 B，一个态射集合 Hom$_\mathcal{C}(A,B)$；– 对任意三个对象 A、B和 C，一个结合组合函数 $\circ_\mathcal{C}$，其定义于 Hom$_\mathcal{C}(B,C) \longrightarrow$ Hom$_\mathcal{C}(A,B) \longrightarrow$Hom$_\mathcal{C}(A,C)$ 中；– 对任意对象 A，一个恒等态射 id$_A$，其属于 Hom$_\mathcal{C}(A,A)$，并且对于任意态射 $f \in$ Hom$_\mathcal{C}(A,B)$ 和$g \in$ Hom$_\mathcal{C}(B,A)$，$f \circ_\mathcal{C}$ id$_A = f$ 和 id$_A \circ_\mathcal{C} g = g$。

给定一个范畴 $\mathcal{C}$，一个 $\mathcal{C}$-自动机是一个元组 $(\Sigma, I, Q, F, i, \delta, f)$，其中

– Σ 是一个符号集合（字母表），– I 是初始对象，属于 Obj($\mathcal{C}$)，– Q 是状态对象，属于 Obj($\mathcal{C}$)，– F 是终对象，属于 Obj($\mathcal{C}$)，– i 是初始态射，属于 Hom$_\mathcal{C}(I,Q)$，– δ 是转移函数，属于 $\Sigma \longrightarrow$ Hom$_\mathcal{C}$ (Q,Q)，– f是值态射，属于 Hom$_\mathcal{C}(Q,F)$。

函数 δ可以扩展为从自由幺半群（$\Sigma^*, \cdot, \varepsilon$）到态射幺半群（Hom$_\mathcal{C}(Q,Q)$ $\circ_\mathcal{C}$, id$_Q$）的幺半群态射，从而得到以下权重定义。

$\mathcal{C}$-自动机 $A = (\Sigma, I, Q, F, i, \delta, f)$ 对字 w（属于 Σ^*）赋予的权重，是 Hom$_\mathcal{C}(I,F)$ 中由下式定义的态射 weight(w)：

$$\texttt{weight}(w) = f \circ_\mathcal{C} \delta(w) \circ_\mathcal{C} i.$$

如果所在范畴是集合范畴，并且 $I = \mathbb{I}$，那么一个字的权重等价地是 F 中的一个元素。因此，确定性（完全）自动机等价于一个以 $\mathbb{I}$ 为初始对象、以 $\mathbb{B}$ 为终对象的 Set-自动机。

给定一个单子 M，两个态射 $f \in$Hom$_\mathcal{C}(A,B)$ 和 $g \in$Hom$_\mathcal{C}(B,C)$ 的 Kleisli组合是态射 $(f >\!=\!> g)(x) = f(x) \gg\!= g$，属于 Hom$_\mathcal{C}(A,C)$。此复合定义了一个范畴，称为 M 的 Kleisli范畴 $\mathcal{K}(M)$，其中：

– 对象是集合，– 两个集合 A 和 B 之间的态射是 A 和 M 之间的函数（B），– 恒等态射是纯函数。

考虑以下范畴：

– 确定性自动机等价于一个 $\mathcal{K}$（Maybe）-自动机，– 非确定性自动机等价于一个 $\mathcal{K}$（Set）-自动机，– 半环 $\mathbb{K}$ 上的加权自动机等价于一个 $\mathcal{K}$（LinComb（$\mathbb{K}$））-自动机，

其中初始对象和终对象均为 $\mathbb{I}$。

此外，对于给定的表达式 E，如果 $i =$ pure(E)、$\delta(a)(E') =$ d$_a(E')$ 以及 $f =$ Null，我们可以利用此前定义的三个支撑来计算众所周知的导数自动机；对于经典表达式而言，这些自动机的可达部分是有限的 [4,1,13]。

更准确地说，扩展表达式可能导致无限自动机，如下一个例子所示。

例*9*。 考虑例8中的计算，可以证明

$$d_{a^n}(E) = \texttt{ExtDist}(a^*b^* + a^*, a^*b^*, a^*b^*a^* + n \odot a^*).$$

因此，经典导数自动机中的状态数不是有限的。这一无限自动机如图1所示，其中状态的终权用双重边表示。汇状态已省略。

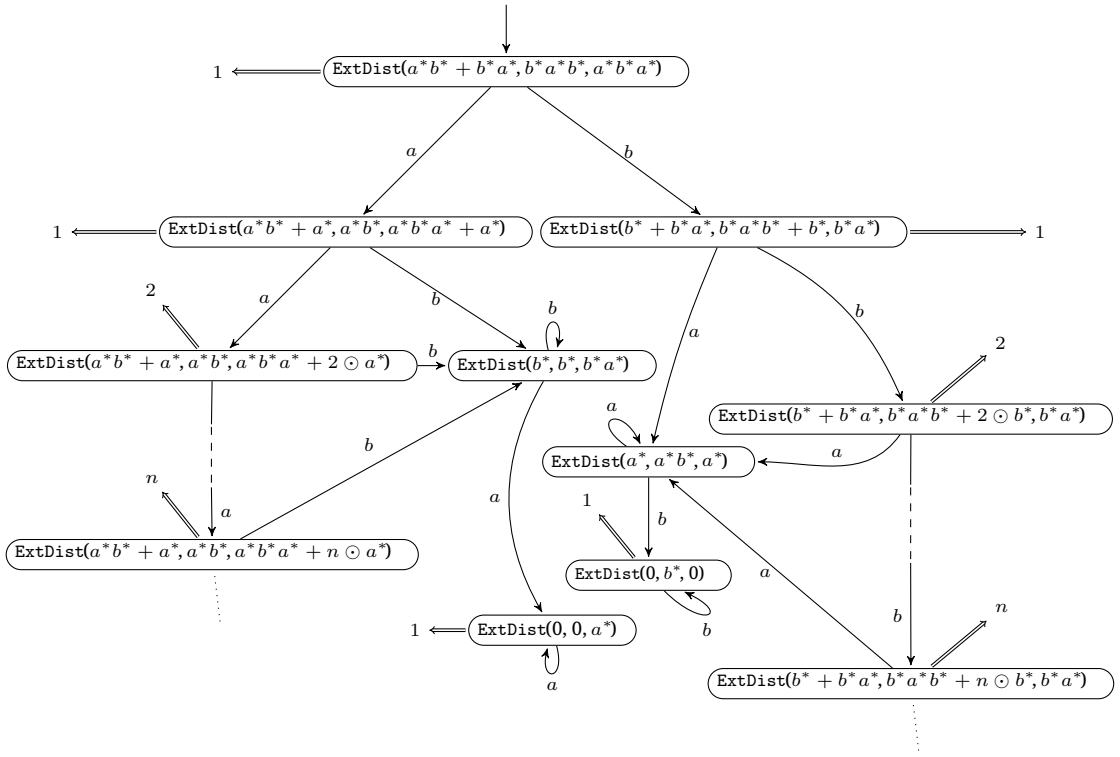

图 1。 与 E相关联的（无限）导数加权自动机。

在下 一节中，我们将展示如何构造一个新的单子来解决这一问题。

7 分次模单子

考虑一个歌剧 $\mathbb{O} = (O, \circ, \mathrm{id})$ 以及将以下内容发送过去的结合：
– 将任意集合 S 映射到 $\bigcup_{n\in\mathbb{N}} O_n \times S^n$，– 将任意 f 中的元素 $S \to S'$ 映射到 $\bigcup_{n\in\mathbb{N}} O_n \times S^n \to \bigcup_{n\in\mathbb{N}} O_n \times S'^n$中的函数 g： $g(o, (s_1, \ldots, s_n)) = (o, (f(s_1) \ldots, f(s_n)))$

可以验证这是一个函子，记作 GradMod($\mathbb{O}$)。此外，考虑以下两个函数，它构成了一个单子：

$$\texttt{pure}(s) = (\mathrm{id}, s),$$
$$(o, (s_1, \ldots, s_n)) >\!\!>\!\!= f = (o \circ (o_1, \ldots, o_n), (s_{1,1}, \ldots, s_{1,i_1}, \ldots, s_{n,1}, \ldots, s_{n,i_n}))$$

其中 $f(s_j) = (o_j, s_{j,1}, \ldots, s_{j,i_j})$。不过，目前尚不清楚 GradMod($\mathbb{O}$) 是否能够轻易地作为值空间进行求值。因此，让我们将其与另一个单子复合。作为示例，考虑一个半环 $\mathbb{K} = (K, \times, +, 1, 0)$ 以及 $\mathbb{O}$ 上的 n 元函数歌剧 $\mathbb{O}$。于是，我们定义函子[7] GradComb($\mathbb{O}, \mathbb{K}$)，它将 S 映射为 GradMod($\mathbb{O}$)(LinComb($\mathbb{K}$)(S))。

[7] 众所周知，两个函子的复合仍是一个函子。

为了证明这个组合是一个单子，我们先定义一个函数 α ，将 GradComb($\mathbb{O},\mathbb{K}$)(S) 映射到 GradMod($\mathbb{O}$)(S)。这可以通过将线性组合转换为拟运算组合，即GradMod $\mathbb{O}$ S 中的一个元素来轻松实现，所用函数如下：toOp

$$\begin{aligned}
&\texttt{toOp}((k_1,s_1)\boxplus\cdots\boxplus(k_n,s_n))\\
&\qquad=(\lambda(x_1,\ldots,x_n)\to k_1\times x_1+\cdots+k_n\times x_n,(s_1,\ldots,s_n)),\\
&\alpha(o,(\mathcal{L}_1,\ldots,\mathcal{L}_n))=(o\circ(o_1,\ldots,o_n),(s_{1,1},\ldots,s_{1,i_1},\ldots,s_{n,1},\ldots,s_{n,i_n}))
\end{aligned}$$

其中 toOp($\mathcal{L}_j$) = o_j, $s_{j,1}$、…、s_{j,i_j}。

因此，我们可以如下定义单子函数:

$$\begin{aligned}
\texttt{pure}(s)&=(\mathrm{id},(1,s)),\\
(o,(\mathcal{L}_1,\ldots,\mathcal{L}_n))\gg\!=f&=\alpha(o,(\mathcal{L}_1,\ldots,\mathcal{L}_n))\gg\!=f
\end{aligned}$$

其中，第二个 $\gg\!=$ 是与单子 GradMod($\mathbb{O}$) 相关联的单子函数。

最后，让我们为这个单子定义一个表达性支撑:

$$\begin{aligned}
\texttt{toExp}(o,(\mathcal{L}_1,\ldots,\mathcal{L}_n))&=o(\texttt{toExp}(\mathcal{L}_1),\ldots,\texttt{toExp}(\mathcal{L}_n)),\\
(o,(\mathcal{L}_1,\ldots,\mathcal{L}_n))+(o',(\mathcal{L}'_1,\ldots,\mathcal{L}'_{n'}))&=(o+o',(\mathcal{L}_1,\ldots,\mathcal{L}_n,\mathcal{L}'_1,\ldots,\mathcal{L}'_{n'}))\\
(o,(\mathcal{L}_1,\ldots,\mathcal{L}_n))\times(o',(\mathcal{L}'_1,\ldots,\mathcal{L}'_{n'}))&=(o\times o',(\mathcal{L}_1,\ldots,\mathcal{L}_n,\mathcal{L}'_1,\ldots,\mathcal{L}'_{n'}))\\
\pm=+,\quad 1=(\mathrm{id},(1,\top)),\quad 0=(\mathrm{id},(0,\top)),\quad \underline{0}&=(\mathrm{id},(0,\top)),\\
m\ltimes F&=\texttt{pure}(\texttt{toExp}(m)\cdot F),\\
(o,(\mathcal{M}_1,\ldots,\mathcal{M}_k))\triangleright(o',(\mathcal{L}_1,\ldots,\mathcal{L}_n))&=(o(\mathcal{M}_1,\ldots,\mathcal{M}_k)\times o',(\mathcal{L}_1,\ldots,\mathcal{L}_n)),\\
(o,(\mathcal{L}_1,\ldots,\mathcal{L}_n))\triangleleft(o',(\mathcal{M}_1,\ldots,\mathcal{M}_k))&=(o\times o'(\mathcal{M}_1,\ldots,\mathcal{M}_k),(\mathcal{L}_1,\ldots,\mathcal{L}_n))\\
f\circledast((o_1,(\mathcal{L}_{1,1},\ldots,\mathcal{L}_{1,i_1})),\ldots,(o_n,(\mathcal{L}_{n,1},\ldots,\mathcal{L}_{n,i_n})))&\\
=(f\circ(o_1,\ldots,o_n),(\mathcal{L}_{1,1},\ldots,\mathcal{L}_{1,i_1},\ldots,\mathcal{L}_{n,1},\ldots,\mathcal{L}_{n,i_n}))&\\
\text{where }(o+o')(x_1,\ldots,x_{n+n'})=o(x_1,\ldots,x_n)+o'(x_{n+1},\ldots,x_{n+n'})&\\
(o\times o')(x_1,\ldots,x_{n+n'})=o(x_1,\ldots,x_n)\times o'(x_{n+1},\ldots,x_{n+n'})&
\end{aligned}$$

例*10.* 考虑 GradComb($\mathbb{O},\mathbb{K}$)(Exp(Σ)) 中的两个元素，如果它们经 toExp 映射后相等，则认为它们相等。考虑例8中的表达式 $E=$ ExtDist($a^*b^*+b^*a^*,b^*a^*b^*,a^*b^*a^*$)。

$$\begin{aligned}
d_a(E)&=\texttt{ExtDist}\circledast((+,(a^*b^*,a^*)),(\mathrm{id},a^*b^*),(+,(a^*b^*a^*,a^*)))\\
&=(\texttt{ExtDist}\circ(+,\mathrm{id},+),(a^*b^*,a^*,a^*b^*,a^*b^*a^*,a^*))\\
d_{aa}(E)&=(\texttt{ExtDist}\circ(+,\mathrm{id},+\circ(+,\mathrm{id})),(a^*b^*,a^*,a^*b^*,a^*b^*a^*,a^*,a^*))\\
&=(\texttt{ExtDist}\circ(+,\mathrm{id},+\circ(\mathrm{id},2\times)),(a^*b^*,a^*,a^*b^*,a^*b^*a^*,a^*))\\
d_{aaa}(E)&=(\texttt{ExtDist}\circ(+,\mathrm{id},+\circ(\mathrm{id},3\times)),(a^*b^*,a^*,a^*b^*,a^*b^*a^*,a^*))\\
d_{aab}(E)&=(\texttt{ExtDist}\circ(+,\mathrm{id},+),(b^*,\emptyset,b^*,b^*a^*,\emptyset))\\
&=(\texttt{ExtDist},(b^*,b^*,b^*a^*))\\
\texttt{weight}_{aaa}(E)&=d_{aaa}(E)\gg\!=\texttt{Null}\\
&=\texttt{ExtDist}\circ(+,\mathrm{id},+)(1,1,1,1,3)\\
&=\texttt{ExtDist}(1+1,1,1+3)=4-1=3\\
\texttt{weight}_{aab}(E)&=d_{aab}(E)\gg\!=\texttt{Null}=\texttt{ExtDist}(1,1,1)=0
\end{aligned}$$

使用这个单子，导出项的数量，也就是相关导数自动机中的状态数量，是有限的。事实上，计算被吸收到转移结构中。该自动机如图2所示。请注意，虚线矩形表示遍历一个字时组合而成的函数。终权用双重边表示。汇状态未显示。为简化表示，状态 b^* 被重复绘制。

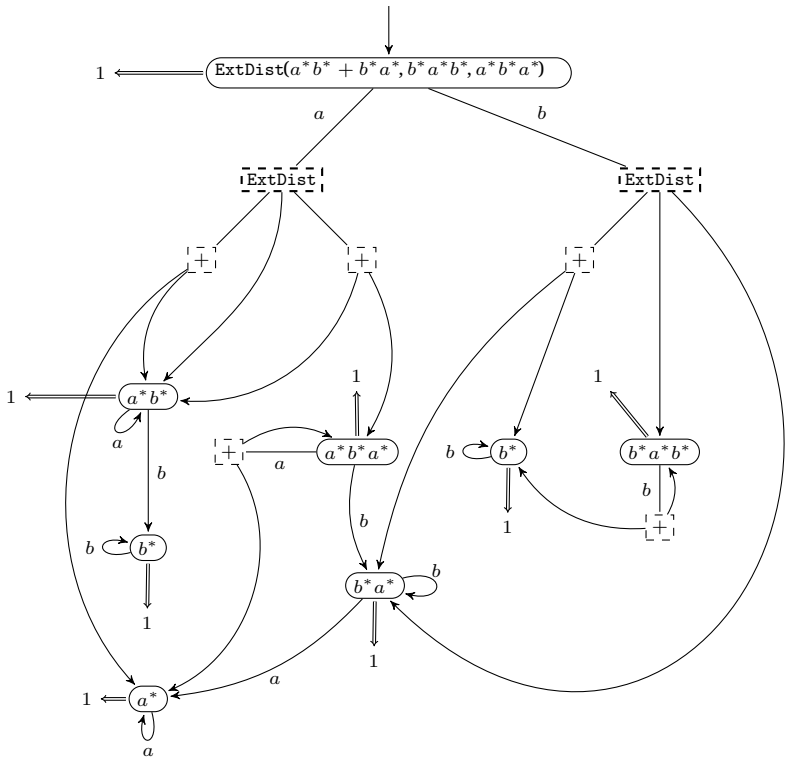

图 2. ExtDist($a^*b^* + b^*a^*, b^*a^*b^*, a^*b^*a^*$) 的关联导数自动机。

不过，请注意，并非每个单子表达式都会产生有限集的导出项，下一例将对此加以说明。

例*11.* 考虑例8中的表达式 E 和表达式 $F = E \cdot c^*$。可以证明

$$\begin{aligned} d_{a^n}(F) &= \mathtt{toExp}(d_{a^n}(E)) \cdot c^* \\ &= \mathtt{ExtDist}(a^*b^* + a^*, a^*b^*, a^*b^*a^* + n \odot a^*) \cdot c^*. \end{aligned}$$

研究能够导出有限项集的单子的充要条件，是我们下一步工作之一。

8 Haskell 实现

否　　前几节所述的内容已在 Haskell 中实现如下：

– 集合子范畴上的单子概念通过一个类型类实现，并使用 *Constraint* 种类 指定子范畴；– n元函数及其运算子结构通过定长向量实现，其大小在编译时使用类型级编程确定；– 分次模的概念通过存在类型实现，以处理未知的元数：其单子结构基于异构列表的一种扩展，即分次向量，并根据其所包含元素的元数列表进行类型化；– 解析器和一些类型级函数基于使用单例类型的依赖类型编程 [8]，例如可以在运行时确定单子的类型或相关函数的元数；– 此处提供了一个应用 [16]，展示这些计算：

- 后端使用 servant 定义应用程序接口；

- 前端使用 Reflex（一个函数式响应式编程引擎）定义，并通过 GHCJS 交叉编译为 JavaScript。

例如，前面示例中的单子表达式可以作为输入 ExtDist(a*.b*+b*.a*,b*.a*.b*,a*.b*.a*) 输入 Web 应用程序。

9 捕获组

捕获组是 POSIX 正则表达式的一项标准功能，其中使用括号记忆所匹配输入字符串的某些部分，以便将其用于替换或匹配。这里给出一个等价定义，以及推导公式和单子定义。该定义的语义与 POSIX 表达式的语义一致。具体而言，当某个捕获组由于带星号的子表达式而被涉及多次时，相应变量的值对应于最后一次捕获。

9.1 带捕获组的表达式语法

在符号字母表 Σ 和变量字母表 Γ（或简写为 Σ,Γ-表达式）上的一个捕获组表达式 E归纳定义如下

$$
\begin{array}{lll}
E = a, & E = \varepsilon, & E = \emptyset, \\
E = F + G, & E = F \cdot G, & E = F^*, \\
E = (F)_x, & & E = x,
\end{array}
$$

其中， F和 G是两个 Σ、Γ-表达式， a是 Σ中的一个符号， u属于 Σ^*，而 x是 Γ中的一个变量。在 POSIX 语法中，捕获组按照左括号的顺序分别隐式映射到变量。这里，通过使用该变量的名称为右括号建立索引，将每个捕获组显式关联到一个变量。

9.2 上下文表达式及其上下文语言

为了定义上下文语言以及捕获组表达式的推导，我们需要扩展表达式的语法，以便将执行过程中捕获的输入字符串当前部分附加到任意捕获组。

一个在符号字母表 Σ 和变量字母表 Γ（或简称为 Σ、Γ-表达式）上的上下文捕获组表达式 E归纳定义如下:

$$
\begin{array}{lll}
E = a, & E = \varepsilon, & E = \emptyset, \\
E = F + G, & E = F \cdot G, & E = F^*, \\
E = (F)^u_x, & & E = x,
\end{array}
$$

其中 F 并且 G是两个 Σ、Γ-表达式， a是 Σ中的一个符号， u属于 Σ^*，而 x是 Γ中的一个变量 .

请注意， Σ、Γ-表达式等价于一个上下文捕获组表达式，其中 $u = \varepsilon$对于每个 出现捕获组的地方。

下面，我们将 上下文 视为从 Γ 到 Maybe(Σ^*) 的函数，用于表示某个变量在解析过程中是否已初始化。上下文集合记为 Ctxt(Γ, Σ)。

利用这些上下文概念，现在让我们解释上下文捕获组表达式的语义。在解析过程中，会构建一个上下文，用于记忆单词到变量的各种赋值。因此，与某个表达式相关联的（上下文）语言，是由一个语言和用于计算该语言的上下文构成的偶对集合。

经典的原子情形（一个符号、空字或空集）很容易定义，同时保留上下文。另一种情形是变量 x：在这里应用上下文来计算与之关联的字（如果存在），并保留该上下文。

递归情形的解释如下:

– 两个表达式之和的上下文语言，是分别独立计算其上下文语言后所得的并集。– 两个表达式 F 和 G 的连接的上下文语言分三步计算。首先，计算 F 的上下文语言。其次，对于该上下文语言中的每一对 (L, ctxt)，将函数 ctxt 视为新的上下文，用于计算 G的上下文语言，从而得到新的二元组 (L', ctxt′)。最后，对于这些组合中的每一个，添加一个二元组 ($L \cdot L'$,ctxt′)，以构成最终的上下文语言。

– 星号表达式的上下文语言通常是幂次上下文语言的无限并集，这些幂次上下文语言通过迭代连接计算得到。

– 捕获表达式的上下文语言（F）$^{u}_{x}$ 分两步计算。首先，计算 F 的上下文语言。然后，对于其中的每一对（L、ctxt），在 L 中选择一个字 w，并且必须相应地更新上下文 ctxt。

更正式地说，上下文语言对于一个与 $\mathsf{Ctxt}(\Gamma,\Sigma)$ 中的上下文 ctxt 相关联的 Σ,Γ-表达式 E，是 $2^{\Sigma^*}\times\mathsf{Ctxt}(\Gamma,\Sigma)$ 的子集 $\mathsf{L}^{\mathrm{ctxt}}(E)$，其归纳定义如下：

$$\mathsf{L}^{\mathrm{ctxt}}(a)=\{(\{a\},\mathrm{ctxt})\},\qquad \mathsf{L}^{\mathrm{ctxt}}(\varepsilon)=\{(\{\varepsilon\},\mathrm{ctxt})\},$$

$$\mathsf{L}^{\mathrm{ctxt}}(\emptyset)=\emptyset,\qquad \mathsf{L}^{\mathrm{ctxt}}(x)=\begin{cases}\emptyset & \text{if } \mathrm{ctxt}(x)=\mathrm{Nothing},\\ \{(\{w\},\mathrm{ctxt})\} & \text{otherwise if } \mathrm{ctxt}(x)=\mathrm{Just}(w),\end{cases}$$

$$\mathsf{L}^{\mathrm{ctxt}}(F+G)=\mathsf{L}^{\mathrm{ctxt}}(F)\cup\mathsf{L}^{\mathrm{ctxt}}(G),\qquad \mathsf{L}^{\mathrm{ctxt}}(F\cdot G)=\bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in\mathsf{L}^{\mathrm{ctxt}}(F),\\(L_2,\mathrm{ctxt}_2)\in\mathsf{L}^{\mathrm{ctxt}_1}(G)}}\{(L_1\cdot L_2,\mathrm{ctxt}_2)\},$$

$$\mathsf{L}^{\mathrm{ctxt}}(F^*)=\bigcup_{n\in\mathbb{N}}(\mathsf{L}^{\mathrm{ctxt}}(F))^{\underline{n}},\qquad \mathsf{L}^{\mathrm{ctxt}}((F)^u_x)=\bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in\mathsf{L}^{\mathrm{ctxt}}(F),\\ w\in L_1}}\{(\{w\},[\mathrm{ctxt}_1]_{x\leftarrow uw})\},$$

其中 F和 G是两个 Σ、Γ-表达式，a是 Σ 中的一个符号，x是 Γ 中的一个变量，u属于 Σ^*，$\mathcal{L}^{\underline{n}}$定义为：对于任意的语言、上下文对集合 $\mathcal{L}$，通过

$$\mathcal{L}^{\underline{n}}=\begin{cases}\displaystyle\bigcup_{(L,\mathrm{ctxt})\in\mathcal{L}}\{(\{\varepsilon\},\mathrm{ctxt})\} & \text{if } n=0,\\ \displaystyle\bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in\mathcal{L},\\(L_2,\mathrm{ctxt}_2)\in\mathcal{L}^{\underline{n-1}}}}\{(L_1\cdot L_2,\mathrm{ctxt}_2)\} & \text{otherwise,}\end{cases}$$

并且 $[\mathrm{ctxt}]_{x\leftarrow w}$ 是上下文定义 y fi d b

$$[\mathrm{ctxt}]_{x\leftarrow w}(y)=\begin{cases}\mathtt{Just}(w) & \text{if } x=y,\\ \mathrm{ctxt}(y) & \text{otherwise.}\end{cases}$$

表达式 E 的上下文语言是从未初始化上下文中得到的偶对集合，其中没有任何变量与任何内容相关联，也就是该集合。

$$\mathsf{L}^{\lambda_\to\mathrm{Nothing}}(E).$$

最后，由表达式 E 表示的语言是忘掉上下文后得到的单词集合，也就是集合

$$\bigcup_{(L,_)\in\mathsf{L}^{\lambda_\to\mathrm{Nothing}}(E)}L.$$

例*12*。考虑以下三个由符号字母表 $\{a,b,c\}$ 和变量字母表 $\{x\}$ 构成的表达式：

$$E=E_1\cdot E_2,\qquad E_1=((a^*)_x bx)^*,\qquad E_2=cx.$$

E_2 表示的语言是空集，因为它是从空上下文计算得出的，其中没有任何内容与 x 关联。然而，解析 E_1 使我们能够计算出定义单词值的上下文，并将这些值赋给 x。因此，我们来说明如何定义 E_1 的上下文语言：

– $(a^*)_x$ 的上下文语言是集合

$$\bigcup_{n\in\mathbb{N}}\{(\{a^n\},\lambda x\to\mathrm{Just}(a^n))\}$$

其中，每个单词 a^n 都记录在一个上下文中；

– $(a^*)_x bx$ 的上下文语言是集合

$$\bigcup_{n\in\mathbb{N}}\{(\{a^nba^n\},\lambda x\to\mathrm{Just}(a^n))\}$$

其中每个字 a^n 都记录在上下文应用中，即用于求值变量 x；h

– E_1 的上下文语言是以下两个集合 S_1 和 S_2的并集：

$$S_1=\{(\{\varepsilon\},\lambda x\to\mathrm{Nothing})\}$$

$$S_2=\{(\{a^nba^n\mid n\in\mathbb{N}\}^*\cdot\{a^mba^m\},\lambda x\to\mathrm{Just}(a^m))\mid m\in\mathbb{N}\}$$

其中，最外层星号的每次迭代都会为上下文中的变量 x 生成一条新的记录；不过请注意，在该过程结束时只会记录最后一条。

最后，通过考虑从解析 E_1 所获得的上下文来计算 E_2中 x 的出现情况，从而得到 E 的语言，最终得到集合

$$\bigcup_{m\in\mathbb{N}} (\{a^n b a^n \mid n \in \mathbb{N}\}^* \cdot \{a^m b a^m c a^m\}).$$

显然，使用这些计算时，一些经典方程仍然成立：

引理 1. 设 E、 F 和 G 是三个 Σ, Γ-表达式，且 上下文 是 $\mathrm{Ctxt}(\Gamma, \Sigma)$ 中的一个上下文。以下两个方程成立：

$$\mathsf{L}^{\mathrm{ctxt}}(E \cdot (F + G)) = \mathsf{L}^{\mathrm{ctxt}}(E \cdot F + E \cdot G)$$
$$\mathsf{L}^{\mathrm{ctxt}}(F^*) = \mathsf{L}^{\mathrm{ctxt}}(\varepsilon + F \cdot F^*)$$

证明思路 让我们按照等式序列进行推导：

$$\begin{aligned}
\mathsf{L}^{\mathrm{ctxt}}(E \cdot (F+G)) &= \bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}}(E),\\ (L_2,\mathrm{ctxt}_2)\in \mathsf{L}^{\mathrm{ctxt}_1}(F+G)}} \{(L_1 \cdot L_2, \mathrm{ctxt}_2)\} \\
&= \bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}}(E),\\ (L_2,\mathrm{ctxt}_2)\in \mathsf{L}^{\mathrm{ctxt}_1}(F)}} \{(L_1 \cdot L_2, \mathrm{ctxt}_2)\} \cup \bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}}(E),\\ (L_2,\mathrm{ctxt}_2)\in \mathsf{L}^{\mathrm{ctxt}_1}(G)}} \{(L_1 \cdot L_2, \mathrm{ctxt}_2)\} \\
&= \mathsf{L}^{\mathrm{ctxt}}(E \cdot F) \cup \mathsf{L}^{\mathrm{ctxt}}(E \cdot G) = \mathsf{L}^{\mathrm{ctxt}}(E \cdot F + E \cdot G)
\end{aligned}$$

$$\begin{aligned}
\mathsf{L}^{\mathrm{ctxt}}(F^*) &= \bigcup_{n\in\mathbb{N}} (\mathsf{L}^{\mathrm{ctxt}}(F))^{\underline{n}} = (\mathsf{L}^{\mathrm{ctxt}}(F))^{\underline{0}} \cup \bigcup_{n\in\mathbb{N}, n\geq 1} (\mathsf{L}^{\mathrm{ctxt}}(F))^{\underline{n}} \\
&= (\mathsf{L}^{\mathrm{ctxt}}(F))^{\underline{0}} \cup \bigcup_{n\in\mathbb{N}} \mathsf{L}^{\mathrm{ctxt}}(F) \cdot (\mathsf{L}^{\mathrm{ctxt}}(F))^{\underline{n}} = (\mathsf{L}^{\mathrm{ctxt}}(F))^{\underline{0}} \cup \mathsf{L}^{\mathrm{ctxt}}(F) \cdot \bigcup_{n\in\mathbb{N}} (\mathsf{L}^{\mathrm{ctxt}}(F))^{\underline{n}} \\
&= L^{\mathrm{ctxt}}(\varepsilon + F \cdot F^*)
\end{aligned}$$

为了对上下文捕获组表达式进行成员关系测试，我们来扩展经典推导方法。不过，首先让我们说明如何扩展可空性谓词；它在整个过程的末尾会用到。

9.3 可空性计算

可空性谓词使我们能够判断空字是否属于某个表达式所表示的语言。对于捕获组而言，需要计算一个上下文。因此，可空性谓词可以表示为一个上下文集合；将该集合应用于表达式后，会得到一个包含空字的语言。

正如我们所见，可空性取决于当前上下文。给定一个表达式和一个上下文 ctxt，可空性谓词是 $2^{\mathrm{Ctxt}(\Gamma,\Sigma)}$ 中的一个集合，计算如下：

$$\mathrm{Null}^{\mathrm{ctxt}}(\varepsilon) = \{\mathrm{ctxt}\} \qquad \mathrm{Null}^{\mathrm{ctxt}}(\emptyset) = \emptyset$$

$$\mathrm{Null}^{\mathrm{ctxt}}(a) = \emptyset \qquad \mathrm{Null}^{\mathrm{ctxt}}(x) = \begin{cases} \{\mathrm{ctxt}\} & \text{if } \mathrm{ctxt}(x) = \mathtt{Just}(\varepsilon) \\ \emptyset & \text{otherwise.} \end{cases}$$

$$\mathrm{Null}^{\mathrm{ctxt}}(E + F) = \mathrm{Null}^{\mathrm{ctxt}}(E) \cup \mathrm{Null}^{\mathrm{ctxt}}(F) \qquad \mathrm{Null}^{\mathrm{ctxt}}(E \cdot F) = \bigcup_{\substack{\mathrm{ctxt}'\in \mathrm{Null}^{\mathrm{ctxt}}(F),\\ \mathrm{ctxt}''\in \mathrm{Null}^{\mathrm{ctxt}'}(G)}} \{\text{上下文}''\}$$

$$\mathrm{Null}^{\mathrm{ctxt}}(E^*) = \{\mathrm{ctxt}\} \qquad \mathrm{Null}^{\mathrm{ctxt}}((E)^u_x) = \bigcup_{\mathrm{ctxt}'\in \mathrm{Null}^{\mathrm{ctxt}}(F)} \{[\mathrm{ctxt}']_{x\leftarrow u}\}$$

其中 E 且 F是两个 Σ, Γ-表达式， a 是 Σ中的一个符号， x 是 Γ中的一个变量，且 u 属于 Σ^*.

例*13.* 让我们考虑例12中的三个表达式：

$$E = E_1 \cdot E_2, \qquad E_1 = ((a^*)_x bx)^*, \qquad E_2 = cx.$$

对于任意上下文 ctxt，

$$\mathrm{Null}^{\mathrm{ctxt}}(E_1) = \{\mathrm{ctxt}\}, \qquad \mathrm{Null}^{\mathrm{ctxt}}(E_2) = \emptyset, \qquad \mathrm{Null}^{\mathrm{ctxt}}(E) = \emptyset.$$

可空性谓词使我们能够确定，在某个表达式的上下文语言中是否存在一对元素，使其第一个分量包含空字。

命题 3。令 E 为一个 Σ，Γ-表达式，并令 ctxt 为 $\mathrm{Ctxt}(\Gamma, \Sigma)$中的一个上下文。则以下两个条件等价：

- $\mathrm{Null}^{\mathrm{ctxt}}(E) \neq \emptyset$,
- $\exists(L, _) \in \mathsf{L}^{\mathrm{ctxt}}(E) \mid \varepsilon \in L$.

证明。对 E 的结构进行归纳：

– 如果 $E = a \in \Sigma$ 或 $E = \emptyset$，则该性质成立，因为 $\mathsf{Null}^{\mathrm{ctxt}}(E)$ 为空，并且 $\mathsf{L}^{\mathrm{ctxt}}(E)$ 中不存在满足 ε 属于 L 的二元组 $(L,$ ctxt′$)$。

– 如果 $E = \varepsilon$，则以下两个条件成立
$$\mathrm{Null}^{\mathrm{ctxt}}(E) = \{\mathrm{ctxt}\}, \qquad \mathsf{L}^{\mathrm{ctxt}}(E) = \{(\{\varepsilon\}, \mathrm{ctxt})\},$$
满足所述条件。

– 如果 $E = F + G$，则以下两个条件成立：
$$\mathsf{Null}^{\mathrm{ctxt}}(F + G) = \mathsf{Null}^{\mathrm{ctxt}}(F) \cup \mathsf{Null}^{\mathrm{ctxt}}(G), \qquad \mathsf{L}^{\mathrm{ctxt}}(F + G) = \mathsf{L}^{\mathrm{ctxt}}(F) \cup \mathsf{L}^{\mathrm{ctxt}}(G)。$$
由于根据归纳假设，以下两个条件成立
$$\mathrm{Null}^{\mathrm{ctxt}}(F) \neq \emptyset \Leftrightarrow \exists(L, \mathrm{ctxt}') \in \mathsf{L}^{\mathrm{ctxt}}(F) \mid \varepsilon \in L,$$
$$\mathrm{Null}^{\mathrm{ctxt}}(G) \neq \emptyset \Leftrightarrow \exists(L, \mathrm{ctxt}') \in \mathsf{L}^{\mathrm{ctxt}}(G) \mid \varepsilon \in L,$$
该命题成立。

– 如果 $E = F \cdot G$，以下两个条件成立：
$$\mathrm{Null}^{\mathrm{ctxt}}(F \cdot G) = \bigcup_{\substack{\mathrm{ctxt}' \in \mathrm{Null}^{\mathrm{ctxt}}(F), \\ \mathrm{ctxt}'' \in \mathrm{Null}^{\mathrm{ctxt}'}(G),}} \{\mathrm{ctxt}''\},$$
$$\mathsf{L}^{\mathrm{ctxt}}(F \cdot G) = \bigcup_{\substack{(L, \mathrm{ctxt}') \in \mathsf{L}^{\mathrm{ctxt}}(F), \\ (L', \mathrm{ctxt}'') \in \mathsf{L}^{\mathrm{ctxt}'}(G),}} \{(L \cdot L', \mathrm{ctxt}'')\}.$$
由于根据归纳假设，以下两个条件成立，f ll i di i h ld
$$\mathrm{Null}^{\mathrm{ctxt}}(F) \neq \emptyset \Leftrightarrow \exists(L, \mathrm{ctxt}') \in \mathsf{L}^{\mathrm{ctxt}}(F) \mid \varepsilon \in L,$$
$$\mathrm{Null}^{\mathrm{ctxt}'}(G) \neq \emptyset \Leftrightarrow \exists(L, \mathrm{ctxt}'') \in \mathsf{L}^{\mathrm{ctxt}'}(G) \mid \varepsilon \in L,$$
该命题成立。

– 如果 $E = F^*$，由于以下两个条件成立
$$\mathrm{Null}^{\mathrm{ctxt}}(F^*) = \{\mathrm{ctxt}\}, \qquad \mathsf{L}^{\mathrm{ctxt}}(F)^{\underline{0}} = \{(\{\varepsilon\}, \mathrm{ctxt})\} \in \mathsf{L}^{\mathrm{ctxt}}(F^*),$$
所述条件成立。 i h ld

– 如果 $E = (F)^u_x$，以下两个条件均成立：
$$\mathrm{Null}^{\mathrm{ctxt}}((F)^u_x) = \bigcup_{\mathrm{ctxt}' \in \mathrm{Null}^{\mathrm{ctxt}}(F)} \{[\mathrm{ctxt}']_{x \leftarrow u}\},$$
$$\mathsf{L}^{\mathrm{ctxt}}((F)^u_x) = \bigcup_{\substack{(L, \mathrm{ctxt}') \in \mathsf{L}^{\mathrm{ctxt}}(F), \\ w \in L}} \{(\{w\}, [\mathrm{ctxt}']_{x \leftarrow uw})\}.$$
然后，根据归纳过程， h th i
$$\mathrm{Null}^{\mathrm{ctxt}}(F) \neq \emptyset \Leftrightarrow \exists(L, \mathrm{ctxt}') \in \mathsf{L}^{\mathrm{ctxt}}(F) \mid \varepsilon \in L,$$
所述条件成立。

– 如果 $E = x$，以下两个条件均成立：
$$\mathrm{Null}^{\mathrm{ctxt}}(x) = \begin{cases} \{\mathrm{ctxt}\} & \text{if } \mathrm{ctxt}(x) = \mathtt{Just}(\varepsilon) \\ \emptyset & \text{otherwise,} \end{cases}$$
$$\mathsf{L}^{\mathrm{ctxt}}(x) = \begin{cases} \emptyset & \text{if } \mathrm{ctxt}(x) = \mathrm{Nothing}, \\ \{(\{w\}, \mathrm{ctxt})\} & \text{otherwise if } \mathrm{ctxt}(x) = \mathrm{Just}(w). \end{cases}$$
因此，命题 o s。 i h ld

9.4 推导公式

与可空性谓词类似，推导计算会在解析表达式的过程中构建上下文。因此，表达式相对于上下文的导数是一个偶对集合（表达式，上下文），其归纳定义如下：对于任意 Σ, Γ-表达式，以及 $\mathsf{Ctxt}(\Gamma, \Sigma)$ 中的任意上下文 ctxt：
$$\mathrm{d}^{\mathrm{ctxt}}_a(\varepsilon) = \emptyset \qquad \mathrm{d}^{\mathrm{ctxt}}_a(\emptyset) = \emptyset$$

$$\mathrm{d}_a^{\mathrm{ctxt}}(b) = \begin{cases} \emptyset & \text{if } a \neq b, \\ \{(\varepsilon, \mathrm{ctxt})\} & \text{otherwise,} \end{cases} \qquad \mathrm{d}_a^{\mathrm{ctxt}}(x) = \begin{cases} \mathrm{d}_a^{\mathrm{ctxt}}(w) & \text{if } \mathrm{ctxt}(x) = \texttt{Just}(w) \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathrm{d}_a^{\mathrm{ctxt}}(F + G) = \mathrm{d}_a^{\mathrm{ctxt}}(F) \cup \mathrm{d}_a^{\mathrm{ctxt}}(G) \qquad \mathrm{d}_a^{\mathrm{ctxt}}(F \cdot G) = \bigcup_{(\mathrm{ctxt}', F') \in \mathrm{d}_a^{\mathrm{ctxt}}(F)} \{(F' \cdot G, \mathrm{ctxt}')\} \cup \bigcup_{\mathrm{ctxt}' \in \mathrm{Null}^{\mathrm{ctxt}}(F)} \mathrm{d}_a^{\mathrm{ctxt}'}(G)$$

$$\mathrm{d}_a^{\mathrm{ctxt}}(F^*) = \bigcup_{(\mathrm{ctxt}', F') \in \mathrm{d}_a^{\mathrm{ctxt}}(F)} \{(F' \cdot F^*, \mathrm{ctxt}')\} \qquad \mathrm{d}_a^{\mathrm{ctxt}}((F)_x^u) = \bigcup_{(\mathrm{ctxt}', F') \in \mathrm{d}_a^{\mathrm{ctxt}}(F)} \{((F')_x^{u \cdot a}, \mathrm{ctxt}')\}$$

其中， F和 G是两个 Σ, Γ-表达式， a是 Σ中的一个符号， x是 Γ中的一个变量，而 u属于 Σ^*。

示例 *14*。 我们考虑示例 12 中的三个表达式：

$$E = E_1 \cdot E_2, \qquad E_1 = ((a^*)_x bx)^*, \qquad E_2 = cx.$$

那么，对于任意上下文 ctxt,

$$\begin{aligned} \mathrm{d}_a^{\mathrm{ctxt}}(E) &= \{((a^*)_x^a bx((a^*)_x bx)^* cx, \mathrm{ctxt})\}, \\ \mathrm{d}_b^{\mathrm{ctxt}}(E) &= \{(x((a^*)_x bx)^* cx, \lambda x \to \varepsilon)\}, \\ \mathrm{d}_c^{\mathrm{ctxt}}(E) &= \{(x, \mathrm{ctxt})\}. \end{aligned}$$

表达式的推导使我们能够以句法方式表达上下文语言中语言分量的商的计算，其中商 $w^{-1}(L)$ 是集合 $\{w' \mid ww' \in L\}$。

命题 4。 令 E 为一个 Σ， Γ-表达式，ctxt 为 Ctxt Γ， Σ且 a 为 Σ中的一个符号。则：

$$\bigcup_{(E', \mathrm{ctxt}') \in d_a^{\mathrm{ctxt}}(E)} \mathsf{L}^{\mathrm{ctxt}'}(E') = \bigcup_{(L', \mathrm{ctxt}') \in \mathsf{L}^{\mathrm{ctxt}}(E)} \{(a^{-1}(L'), \mathrm{ctxt}')\}$$

证明。 对 E 的结构进行归纳，并将 $\emptyset$ 与 $\{(\emptyset, \mathrm{ctxt})\}$ 视为任意上下文 ctxt。

– 如果 $E = \varepsilon$ 或 $E = \emptyset$, 则该性质显然成立。– 如果 $E = b \in \Sigma$,

$$\bigcup_{(E', \mathrm{ctxt}') \in d_a^{\mathrm{ctxt}}(b)} \mathsf{L}^{\mathrm{ctxt}'}(E') = \begin{cases} \emptyset & \text{if } b \neq a, \\ \{(\{\varepsilon\}, \mathrm{ctxt})\} & \text{otherwise,} \end{cases}$$

$$= \{(a^{-1}(\{b\}), \mathrm{ctxt})\} = \bigcup_{(L', \mathrm{ctxt}') \in \mathsf{L}^{\mathrm{ctxt}}(b)} \{(a^{-1}(L'), \mathrm{ctxt}')\}.$$

– 如果 $E = F + G$,

$$\begin{aligned} \bigcup_{(E', \mathrm{ctxt}') \in d_a^{\mathrm{ctxt}}(F+G)} \mathsf{L}^{\mathrm{ctxt}'}(E') &= \bigcup_{(E', \mathrm{ctxt}') \in d_a^{\mathrm{ctxt}}(F) \cup d_a^{\mathrm{ctxt}}(G)} \mathsf{L}^{\mathrm{ctxt}'}(E') \\ &= \bigcup_{(E', \mathrm{ctxt}') \in d_a^{\mathrm{ctxt}}(F)} \mathsf{L}^{\mathrm{ctxt}'}(E') \cup \bigcup_{(E', \mathrm{ctxt}') \in d_a^{\mathrm{ctxt}}(G)} \mathsf{L}^{\mathrm{ctxt}'}(E') \\ &= \bigcup_{(L', \mathrm{ctxt}') \in \mathsf{L}^{\mathrm{ctxt}}(F)} \{(a^{-1}(L'), \mathrm{ctxt}')\} \cup \bigcup_{(L', \mathrm{ctxt}') \in \mathsf{L}^{\mathrm{ctxt}}(G)} \{(a^{-1}(L'), \mathrm{ctxt}')\} \\ &= \bigcup_{(L', \mathrm{ctxt}') \in \mathsf{L}^{\mathrm{ctxt}}(F) \cup \mathsf{L}^{\mathrm{ctxt}}(G)} \{(a^{-1}(L'), \mathrm{ctxt}')\} \\ &= \bigcup_{(L', \mathrm{ctxt}') \in \mathsf{L}^{\mathrm{ctxt}}(F+G)} \{(a^{-1}(L'), \mathrm{ctxt}')\}. \end{aligned}$$

– If $E = F \cdot G$,

$$\begin{aligned} \bigcup_{(E', \mathrm{ctxt}') \in d_a^{\mathrm{ctxt}}(F \cdot G)} \mathsf{L}^{\mathrm{ctxt}'}(E') &= \bigcup_{(\mathrm{ctxt}', F') \in \mathrm{d}_a^{\mathrm{ctxt}}(F)} \mathsf{L}^{\mathrm{ctxt}'}(F' \cdot G) \cup \bigcup_{\substack{\mathrm{ctxt}' \in \mathrm{Null}^{\mathrm{ctxt}}(F), \\ (G', \mathrm{ctxt}'') \in \mathrm{d}_a^{\mathrm{ctxt}'}(G)}} \mathsf{L}^{\mathrm{ctxt}''}(G') \\ &= \bigcup_{\substack{(\mathrm{ctxt}', F') \in \mathrm{d}_a^{\mathrm{ctxt}}(F), \\ (L_1, \mathrm{ctxt}_1) \in \mathsf{L}^{\mathrm{ctxt}}(F'), \\ (L_2, \mathrm{ctxt}_2) \in \mathsf{L}^{\mathrm{ctxt}_1}(G)}} \{(L_1 \cdot L_2, \mathrm{ctxt}_2)\} \cup \bigcup_{\substack{\mathrm{ctxt}' \in \mathrm{Null}^{\mathrm{ctxt}}(F), \\ (G', \mathrm{ctxt}'') \in \mathrm{d}_a^{\mathrm{ctxt}'}(G)}} \mathsf{L}^{\mathrm{ctxt}''}(G') \\ &= \bigcup_{\substack{(L_1, \mathrm{ctxt}_1) \in \mathsf{L}^{\mathrm{ctxt}}(F), \\ (L_2, \mathrm{ctxt}_2) \in \mathsf{L}^{\mathrm{ctxt}_1}(G)}} \{(a^{-1}(L_1) \cdot L_2, \mathrm{ctxt}_2)\} \cup \bigcup_{\substack{\mathrm{ctxt}_1 \in \mathrm{Null}^{\mathrm{ctxt}}(F), \\ (L_2, \mathrm{ctxt}_2) \in \mathsf{L}^{\mathrm{ctxt}_1}(G)}} \{(a^{-1}(L_2), \mathrm{ctxt}_2)\} \end{aligned}$$

$$= \bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}}(F),\\ (L_2,\mathrm{ctxt}_2)\in \mathsf{L}^{\mathrm{ctxt}_1}(G)}} \{(a^{-1}(L_1)\cdot L_2, \mathrm{ctxt}_2)\} \cup \bigcup_{\substack{\exists (L,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}}(F) | \varepsilon\in L,\\ (L_2,\mathrm{ctxt}_2)\in \mathsf{L}^{\mathrm{ctxt}_1}(G)}} \{(a^{-1}(L_2), \mathrm{ctxt}_2)\}$$

$$= \bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}}(F),\\ (L_2,\mathrm{ctxt}_2)\in \mathsf{L}^{\mathrm{ctxt}_1}(G)}} \{(a^{-1}(L_1)\cdot L_2, \mathrm{ctxt}_2)\} \cup \bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}}(F),\\ \varepsilon\in L_1,\\ (L_2,\mathrm{ctxt}_2)\in \mathsf{L}^{\mathrm{ctxt}_1}(G)}} \{(a^{-1}(L_2), \mathrm{ctxt}_2)\}$$

$$= \bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}}(F),\\ (L_2,\mathrm{ctxt}_2)\in \mathsf{L}^{\mathrm{ctxt}_1}(G)}} \{(a^{-1}(L_1\cdot L_2), \mathrm{ctxt}_2)\}$$

$$= \bigcup_{(L',\mathrm{ctxt}')\in \mathsf{L}^{\mathrm{ctxt}}(F\cdot G)} \{(a^{-1}(L'), \mathrm{ctxt}')\}.$$

– If $E = F^*$,

$$\bigcup_{(E',\mathrm{ctxt}')\in d_a^{\mathrm{ctxt}}(F^*)} \mathsf{L}^{\mathrm{ctxt}'}(E') = \bigcup_{(\mathrm{ctxt}',F')\in \mathrm{d}_a^{\mathrm{ctxt}}(F)} \mathsf{L}^{\mathrm{ctxt}'}(F'\cdot F^*)$$

$$= \bigcup_{\substack{(\mathrm{ctxt}',F')\in \mathrm{d}_a^{\mathrm{ctxt}}(F),\\ (L_1,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}}(F'),\\ (L_2,\mathrm{ctxt}_2)\in \mathsf{L}^{\mathrm{ctxt}_1}(F^*)}} \{(L_1\cdot L_2, \mathrm{ctxt}_2)\}$$

$$= \bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}}(F),\\ (L_2,\mathrm{ctxt}_2)\in \mathsf{L}^{\mathrm{ctxt}_1}(F^*)}} \{(a^{-1}(L_1)\cdot L_2, \mathrm{ctxt}_2)\}$$

$$= \bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}}(F),\\ (L_2,\mathrm{ctxt}_2)\in \mathsf{L}^{\mathrm{ctxt}_1}(F^*)}} \{(a^{-1}(L_1\cdot L_2), \mathrm{ctxt}_2)\}$$

$$= \bigcup_{(L',\mathrm{ctxt}')\in \mathsf{L}^{\mathrm{ctxt}}(F\cdot F^*)} \{(a^{-1}(L'), \mathrm{ctxt}')\}$$

$$= \bigcup_{(L',\mathrm{ctxt}')\in \mathsf{L}^{\mathrm{ctxt}}(\varepsilon + F\cdot F^*)} \{(a^{-1}(L'), \mathrm{ctxt}')\}$$

$$= \bigcup_{(L',\mathrm{ctxt}')\in \mathsf{L}^{\mathrm{ctxt}}(F^*)} \{(a^{-1}(L'), \mathrm{ctxt}')\}$$

– If $E = (F)_x^u$,

$$\bigcup_{(E',\mathrm{ctxt}')\in d_a^{\mathrm{ctxt}}((F)_x^u)} \mathsf{L}^{\mathrm{ctxt}'}(E') = \bigcup_{(\mathrm{ctxt}',F')\in \mathrm{d}_a^{\mathrm{ctxt}}(F)} \mathsf{L}^{\mathrm{ctxt}'}((F')_x^{u\cdot a})$$

$$= \bigcup_{\substack{(\mathrm{ctxt}',F')\in \mathrm{d}_a^{\mathrm{ctxt}}(F)\\ (L_1,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}'}(F'),\\ w\in L_1}} \{(\{w\}, [\mathrm{ctxt}_1]_{x\leftarrow uaw})\}$$

$$= \bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}}(F),\\ w\in a^{-1}(L_1)}} \{(\{w\}, [\mathrm{ctxt}_1]_{x\leftarrow uaw})\}$$

$$= \bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}}(F),\\ aw\in L_1}} \{(\{w\}, [\mathrm{ctxt}_1]_{x\leftarrow uaw})\}$$

$$= \bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}}(F),\\ aw\in L_1}} \{(a^{-1}(\{aw\}), [\mathrm{ctxt}_1]_{x\leftarrow uaw})\}$$

$$= \bigcup_{\substack{(L_1,\mathrm{ctxt}_1)\in \mathsf{L}^{\mathrm{ctxt}}(F),\\ w\in L_1}} \{(a^{-1}(\{w\}), [\mathrm{ctxt}_1]_{x\leftarrow uw})\}$$

$$= \bigcup_{(L',\mathrm{ctxt}')\in \mathsf{L}^{\mathrm{ctxt}}((F)_x^u)} \{(a^{-1}(L'), \mathrm{ctxt}')\}$$

– 如果 $E = x$,

$$\bigcup_{(E',\mathrm{ctxt}')\in d_a^{\mathrm{ctxt}}(x)} \mathsf{L}^{\mathrm{ctxt}'}(E') = \begin{cases} \bigcup_{(E',\mathrm{ctxt}')\in d_a^{\mathrm{ctxt}}(w)} \mathsf{L}^{\mathrm{ctxt}'}(E') & \text{if } \mathrm{ctxt}(x) = \mathrm{Just}(w), \\ \emptyset & \text{otherwise,} \end{cases}$$

$$= \begin{cases} \bigcup_{(w,\mathrm{ctxt})\in d_a^{\mathrm{ctxt}}(aw)} \mathsf{L}^{\mathrm{ctxt}}(w) & \text{if } \mathrm{ctxt}(x) = \mathrm{Just}(aw), \\ \emptyset & \text{otherwise,} \end{cases}$$

$$= \begin{cases} \{(\{w\}, \mathrm{ctxt})\} & \text{if } \mathrm{ctxt}(x) = \mathrm{Just}(aw), \\ \emptyset & \text{otherwise,} \end{cases}$$

$$= \begin{cases} \{(a^{-1}(\{aw\}), \mathrm{ctxt})\} & \text{if } \mathrm{ctxt}(x) = \mathrm{Just}(aw), \\ \emptyset & \text{otherwise,} \end{cases}$$

$$= \begin{cases} \{(a^{-1}(\{w\}), \mathrm{ctxt})\} & \text{if } \mathrm{ctxt}(x) = \mathrm{Just}(w), \\ \emptyset & \text{otherwise,} \end{cases}$$

$$= \bigcup_{(L',\mathrm{ctxt}')\in \mathsf{L}^{\mathrm{ctxt}}(x)} \{(a^{-1}(L'), \mathrm{ctxt}')\}$$

关于一个字的推导通常是关于一个符号的推导的迭代应用，递归定义如下：对于任意 Σ, Γ-表达式 E、任意上下文 $\mathsf{ctxt} \in \mathsf{Ctxt}(\Gamma, \Sigma)$、任意符号 $a \in \Sigma$ 以及任意字 $v \in \Sigma^*$:

$$d_\varepsilon^{\mathrm{ctxt}}(E) = \{(E, \mathrm{ctxt})\}, \qquad d_{a\cdot v}^{\mathrm{ctxt}}(E) = \bigcup_{(E',\mathrm{ctxt}')\in d_a^{\mathrm{ctxt}}(E)} d_v^{\mathrm{ctxt}'}(E').$$

示例 *15*。 考虑示例 14 中的三个表达式:

$$E = E_1 \cdot E_2, \qquad E_1 = ((a^*)_x bx)^*, \qquad E_2 = cx.$$

于是，对于任意上下文 ctxt,

$$\begin{aligned} \mathrm{d}_{ab}^{\mathrm{ctxt}}(E) &= \mathrm{d}_b^{\mathrm{ctxt}}((a^*)_x^a bx((a^*)_x bx)^* cx) \\ &= \{(x((a^*)_x bx)^* cx, \lambda x \to a)\} \\ \mathrm{d}_{aba}^{\mathrm{ctxt}}(E) &= \mathrm{d}_a^{\lambda x\to a}(x((a^*)_x bx)^* cx) \\ &= \{(((a^*)_x bx)^* cx, \lambda x \to a)\} \\ \mathrm{d}_{abac}^{\mathrm{ctxt}}(E) &= \mathrm{d}_c^{\lambda x\to a}(((a^*)_x bx)^* cx) \\ &= \{(x, \lambda x \to a)\} \\ \mathrm{d}_{abaca}^{\mathrm{ctxt}}(E) &= \mathrm{d}_a^{\lambda x\to a}(x) \\ &= \{(\varepsilon, \lambda x \to a)\} \end{aligned}$$

这一操作使我们能够在语法上计算商。

命题 5。 设 E 是一个 Σ, Γ-表达式，ctxt 是上下文 $\mathrm{Ctxt}(\Gamma, \Sigma)$ 中的一个上下文，且 w 是 Σ^* 中的一个字。则:

$$\bigcup_{(E',\mathrm{ctxt}')\in d_w^{\mathrm{ctxt}}(E)} \mathsf{L}^{\mathrm{ctxt}'}(E') = \bigcup_{(L',\mathrm{ctxt}')\in \mathsf{L}^{\mathrm{ctxt}}(E)} \{(w^{-1}(L'), \mathrm{ctxt}')\}$$

证明。 通过对单词结构进行直接归纳。

最后，可以像通常一样，通过先计算关于 w 的推导，然后确定是否存在可空导数，来执行单词 w 的成员关系测试；这是命题 3 和命题 5 的直接推论。

定理 2。 令 E 为一个 Σ, Γ-表达式，上下文为 $\mathrm{Ctxt}(\Gamma, \Sigma)$ 中的上下文，且 w 为 Σ^* 中的一个字。则以下两个条件等价:

– $\exists(L, _) \in \mathsf{L}^{\mathrm{ctxt}}(E) \mid w \in L$, – $\exists(E', \text{上下文}') \in d_w^{\mathrm{ctxt}}(E) \mid \mathsf{Null}^{\mathrm{ctxt}'}(E') \neq \emptyset$。

我们已经展示了如何以经典方式计算导数并解决成员关系测试。下面我们来展示如何将上下文计算嵌入一个便捷的单子中，以便将这些定义推广到集合以外的其他结构。

9.5 StateT 单子变换器

一般来说，单子并不能很好地组合。不过，我们可以考虑这些对象的某些特定组合。其中，众所周知的模式包括单子变换器，例如 StateT 单子变换器 [10]。这种组合使我们能够以函数式的方式模拟全局变量的使用。在我们的场景中，它使我们能够优雅地嵌入上下文计算。

设 S 是一个集合， M 是一个单子。我们将以下映射记为 StateT(S, M)：

$$\texttt{StateT}(S, M)(A) = S \to M(A \times S).$$

换言之，StateT$(S, M)(A)$ 是从 S 到基于笛卡尔积 $(A \times S)$ 中元素构成的单子结构 $M(A \times S)$ 的函数集合。

映射 StateT(S, M) 可以赋予函子结构：对于任意从集合 A 到集合 B 的函数 f，定义如下：

$$\texttt{StateT}(S, M)(f)(\mathrm{state})(s) = M(\lambda(a, s) \to (f(a), s))(\mathrm{state}(s)).$$

它还可以赋予单子结构，定义如下：对于任意从集合 A 到集合 B 的函数 f：

$$\begin{aligned}\texttt{pure}(a) &= \lambda s \to \texttt{pure}(a, s)\\ \texttt{bind}(f)(\mathrm{state})(s) &= \mathrm{state}(s) \gg\!= \lambda(a, s') \to f(a)(s')\end{aligned}$$

9.6 单子定义

之前与捕获组表达式相关的定义，可以使用专门化为 Set 单子的 StateT 单子变换器进行等价重述。

我们首先考虑以下断言，其中 $M =$ StateT(Ctxt(Γ, Σ), Set)，从而使 M 与先前的单子支撑概念更加接近：

- $\mathbb{R} = (M(\mathbb{I}), +, \times, 1, 0)$ 通过设置以下内容构成一个半环：
$$\begin{aligned} f_1 + f_2 &= \lambda s \to f_1(s) \cup f_2(s), & f_1 \times f_2 &= f_1 \gg\!= \lambda_ \to f_2,\\ 1 &= \lambda s \to \{(\top, s)\} = \mathrm{pure}(\top), & 0 &= \lambda s \to \emptyset, \end{aligned}$$
- $\mathbb{M} = (M(\mathrm{Exp}(\Sigma)), \pm, \underline{0})$ 通过设置以下内容构成一个幺半群：
$$\begin{aligned} \pm &= +, & \underline{0} &= 0, \end{aligned}$$
- $(\mathbb{M}, \ltimes)$ 通过如下设定构成一个 Exp(Σ)-右半模：
$$f \ltimes F = \lambda s \to \bigcup_{(E, \mathrm{ctxt}) \in f(s)} \{(E \cdot F, \mathrm{ctxt})\},$$
- $(\mathbb{M}, \triangleright)$ 通过如下设定构成一个 $\mathbb{R}$-左半模：
$$f_1 \triangleright f_2 = f_1 \gg\!= \lambda_ \to f_2.$$

于是，可空谓词公式可以等价地重述为 StateT(Ctxt(Γ, Σ), Set)$(\mathbb{I})$ 中的一个元素；根据定义，这等于 Ctxt$(\Gamma, \Sigma) \to$ Set$(\mathbb{I} \times$ Ctxt$(\Gamma, \Sigma))$，并且同构于 Ctxt$(\Gamma, \Sigma) \to$ Set(Ctxt$(\Gamma, \Sigma))$。它可以归纳计算如下：

$$\begin{aligned} \mathrm{Null}(\varepsilon) &= 1 & \mathrm{Null}(\emptyset) &= 0 & \mathrm{Null}(a) &= 0\\ \mathrm{Null}(E + F) &= \mathrm{Null}(E) + \mathrm{Null}(F) & \mathrm{Null}(E \cdot F) &= \mathrm{Null}(E) \times \mathrm{Null}(F) & \mathrm{Null}(E^*) &= 1 \end{aligned}$$

$$\mathrm{Null}(x)(\mathrm{ctxt}) = \begin{cases} \mathrm{pure}((\top, \mathrm{ctxt})) & \text{if } \mathrm{ctxt}(x) = \texttt{Just}(\varepsilon),\\ \emptyset & \text{otherwise,} \end{cases}$$

$$\mathrm{Null}((E)_x^u)(\mathrm{ctxt}) = \text{Set 范畴}(\lambda(\top, \mathrm{ctxt}') \to (\top, [\mathrm{ctxt}']_{x \leftarrow u}))(\mathrm{Null}(F)(\mathrm{ctxt})),$$

其中， E和 F是两个 Σ, Γ-表达式， a是 Σ中的一个符号， x是 Γ中的一个变量，而 u属于 Σ^*。请注意，就经典算子而言，这些公式与定义 2 中的公式相同，并且这些公式可以轻松推广到 Set 范畴以外的其他合适的单子。

此外，一个表达式的导数是 StateT(Ctxt(Γ, Σ), Set 范畴)(Exp(Σ, Γ))中的一个元素：

$$\mathrm{d}_a(\varepsilon) = \underline{0} \qquad \mathrm{d}_a(\emptyset) = \underline{0} \qquad \mathrm{d}_a(b) = \begin{cases} \underline{0} & \text{if } a \neq b,\\ \texttt{pure}(\varepsilon) & \text{otherwise,} \end{cases}$$

$$\mathrm{d}_a(E + F) = \mathrm{d}_a(E) \pm \mathrm{d}_a(F) \quad \mathrm{d}_a(E \cdot F) = \mathrm{d}_a(E) \ltimes F + \mathrm{Null}(E) \triangleright \mathrm{d}_a(F) \quad \mathrm{d}_a(E^*) = \mathrm{d}_a(E) \ltimes E^*$$

$$\mathrm{d}_a((E)_x^u) = \texttt{StateT}(\mathrm{Ctxt}(\Gamma, \Sigma), \mathrm{Set})(\lambda F \to (F)_x^{ua})(\mathrm{d}_a(E))$$

$$\mathrm{d}_a(x)(\mathrm{ctxt}) = \begin{cases} \mathrm{pure}((w, \mathrm{ctxt})) & \text{if } \mathrm{ctxt}(x) = \texttt{Just}(aw),\\ \emptyset & \text{otherwise,} \end{cases}$$

其中， E和 F是两个 Σ, Γ-表达式， a是 Σ中的一个符号， x是 Γ中的一个变量，而 u属于 Σ^*。再次注意，就经典算子而言，这些公式与定义 5 中的公式相同；并且这些公式可以很容易地推广到 Set 范畴以外的其他适用单子。

最后，关于一个字的推导如前几节所述以单子方式定义：$d_\varepsilon(E) = \mathsf{pure}(E)$

$$d_{av}(E) = d_a(E) \gg\!= d_v,$$

而一个字的成员关系测试 w 可以等价地改写如下：$(d_w(E) \gg\!= \mathsf{Null})(\lambda_ \to \mathsf{Nothing}) \neq \emptyset.$

10 结论与展望

在本文中，我们完成了统一单词表达式上的导数计算这一计划的第一步。单子确实是抽象底层计算结构的有用工具，因此可以让我们考虑其他一些功能，例如通过广为人知的 StateT 单子变换器 [10]实现捕获组。我们旨在研究使导出项构成有限集的单子所需满足的条件，并使用富化范畴将该方法扩展到树表达式。最后，我们计划将单子导数扩展到用于捕获组的其他底层单子，例如线性组合。

参考文献

1. Antimirov, V.M.：正则表达式的偏导数与有限自动机构造。理论计算机科学**155**319 2 2 1996年 291 ( ) ( ) –。Attou, S.、Mignot, L.、Miklarz, C.、Nicart, F.：单子表达式及其导数。载于：NCMA。EPTCS 第367卷（2022年 64 3 49）–。Berry, G.、Sethi, R.：从正则表达式到确定自动机。理论计算机科学 **48** (1986年) 117–126 4 494 **5114** 1964年 481。Brzozowski, J.A.：正则表达式的导数。ACM 学报 ( ) ( ) –。Caron, P.、Flouret, M.：从 Glushkov 加权有限自动机到 k-表达式。Fundam. Informaticae **109**25 6 1 2011年 1( ) ( ) –。Champarnaud, J.、Laugerotte, É.、Ouardi, F.、Ziadi, D.：从正则加权表达式到有限自动机。Int. J. Found. Comput. Sci. **15**700 7 5 2004年 687 ( ) ( ) –。Colcombet, T.、Petrisan, D.：自动机与最小化。SIGLOG News **4**27 8 2 2017年 4( ) ( ) –。Eisenberg, R.A.、Weirich, S.：使用单例类型进行依赖类型编程。载于：Haskell，ACM（2012年 130 9 **117165** 1961年）–。Glushkov, V.M.：自动机抽象理论。Russian Mathematical Surveys ( ) ( ) 1 10。Jones, M.P.：带重载和高阶多态的函数式编程。载于：Adv. Func. Prog. 计算机科学讲义第925卷，施普林格（136 11 1995年 97）–。Kleene, S.：神经网络和有限自动机中的事件表示。Automata Studies **Ann. Math. Studies 34** (1956年) 3–41 Princeton U. Press。12. Loday, J.L.、Vallette, B.：代数歌剧。第346卷。施普林格科学与商业媒体（2012年）13. Lombardy, S.、Sakarovitch, J.：带重数的有理表达式的导数。理论计算机科学 **332**(1-3) (2005年) 141–177 14. May, J.P.：迭代环空间的几何。第271卷。施普林格（2006年）15. Mignot, L.：使用富化范畴实现自动机结构、表达式及其相关算法的一项方案（法文）。诺曼底鲁昂大学，研究指导资格论文（2020年12月）212页。16. Mignot, L.：单子导数。https://github.com/LudovicMignot/MonadicDerivatives (2022年) 17 270 1842 3 1961年 245。Schützenberger, M.P.：论自动机族的定义。Inf. Control. (-) ( ) –。Sulzmann, M.、Lu, K.Z.M.：使用导数进行 POSIX 正则表达式解析。载于：FLOPS。计算机科学讲义第8475卷，施普林格（2014年）203–220