关于无名 λ 演算中 β归约的非传统观点

Rob Nederpelt埃因霍温
理工大学，荷兰埃因霍温 数学与计算机科学系
r.p.nederpelt@tue.nl

Ferruccio Guidi博洛尼亚
大学，意大利博洛尼亚，计算机科学与工程系（DISI）
ferruccio.guidi@unibo.it

λ 演算中的项可以表示为平面树：树上装饰着表示“抽象”和“应用”的符号，并以变量作为叶子。在本文中，我们关注这类树的分支，而不是树本身。我们在这一视角下重新阐述了几种众所周知的 β归约概念。以一种自然的方式，这一重新考察最终导向一种新的 β归约形式；这种归约具有扩展性——也就是说，项 t_1 归约为项 t_2，意味着 t_1 的树是 t_2 的树的子树。

谨以此文献给 Stefano Berardi，以纪念他 64 岁生日，并向他在理论计算机科学领域广泛而多元的研究致敬。

1 初步说明

1.1 引言

众所周知，λ-演算的实际实现会通过用深度索引或层级索引（即正整数）表示约束变量出现，而非使用名称（[8]），将 α-等价转化为句法上的相等。因此，与带名系统相对，这类系统称为 namefree 系统。一般来说， β- 归约会将函数体中受约束变量的出现替换为函数自变量的副本。在这种情况下，为防止发生变量捕获，副本中的索引可能需要更新。经验表明，这种更新按照广为接受的术语称为提升，是一项耗时的操作 [13,附录 A2]，而计算机恰恰力求避免这种操作（[16]）。在一个源自 [11] 和 λσ（[1]）的系统家族中， β-归约不会立即执行更新，而是通过向复制的项添加特定结构，将更新记录在其中。因此，计算可以任意延迟更新，也可以在适当的时候执行更新。

本文介绍的无名系统之一仅基于远距离 β-归约，这是在（[17]）中引入的 β-归约的扩展。这种归约关系例如不仅允许 β-归约 $K \equiv (\lambda xy\,.\,L)MN \to (\lambda y\,.\,L[x := M])N$，也允许其变体 $K \to (\lambda x\,.\,L[y := N])M$。另见（[21, 22]）。

与通常的记法不同，λ 项被表示为属于有根树的完全分支，这些树由顶点和标记边构成。借助这种表示，我们可以清晰地看到抽象与应用的匹配对，而每一对都可能引发远距β归约。这种清晰性极大地促进了我们的讨论。

T. Altenkirch、F. Barbanera、F、Damiani 和 U. de’Liguoro（编）：
《逻辑与类型理论：献给 Stefano Berardi 64岁生日的论文集》，
EPTCS441，2026年，第197–212页，doi:10.4204/EPTCS.441.12

© Rob Nederpelt 和 Ferruccio Guidi
本作品采用知识共享署名许可协议授权。

1.2 关于 λ 项的树结构

这篇论文背后的主要动机纯粹是好奇心。显然，（无类型或有类型的）λ-演算中的项具有树结构，并以抽象符号和应用符号装饰——分别记作 λ 和 @。在带名称版本中，叶子是变量；而在无名情形中，它们是正自然数。无名树对应于（无类型）项的情形见图 1(i)：

$(*)$ $(\lambda((\lambda\lambda\,2\,2)\lambda\,2)1)\lambda\,1$,

即带名称项的无名版本

$(\lambda x.\,(\lambda y.\,\lambda z.\,yy)(\lambda u.\,x)x)\lambda v.\,v.$

在这篇论文中，我们通常研究这种无名树格式（首次见于（[8]）；另见（[9]）。）

我们的工作受到以下问题的启发：

（分支聚焦）。能否通过关注树的分支而非树本身，来描述各种归约关系？

关于图 1 (i) 中所有分支的线性表示，参见例 1.1 左侧。在本文中，我们尝试针对各种知名形式的 β归约回答这个问题。这引出了 β归约的一种新形式，其特点是能够扩展所考察的树，而不会造成任何损失。

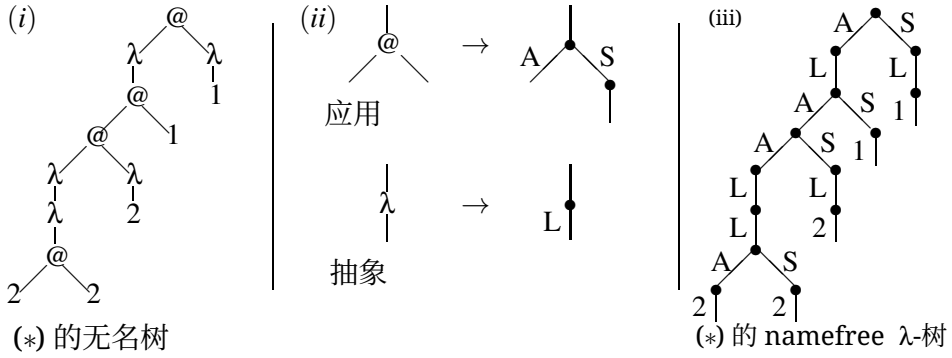


图 1：无名 λ 树；传统形式与改编形式

我们的首要目标是确保整组分支能够表示其所源自的树。为此，需要考虑几个问题：

(i) 可靠的树重构 图 1 (i) 中的树是平面的，也就是说，每个 @ 下方都有一个左分支和一个右分支。然而，在含有 @ 的分支中，没有线索表明原树中的路径是向左还是向右延伸的。

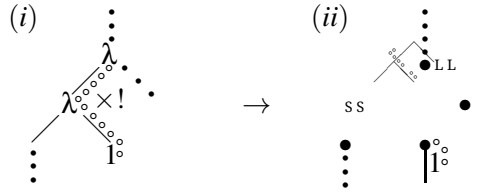
我们选择通过在右分支上方添加一个“标签” S（代表“子项”）来解决这一问题，同时保持左分支不变。参见图 1 (ii)。为便于表示，我们用 A 代替 @，用 L 代替 λ。此后本文均采用这种表示法。

(ii) 可靠的缩约检测 在树中，若某个 @ 的正下方左侧有一个 λ，则可确定存在一个可约式。然而，若某个 @ 的正下方右侧有一个 λ，则不能据此确定存在可约式（参见例 1.1 (3)）。

按照刚才的描述，正确添加标号 S，确保了“向左下降”和“向右下降”之间的区别，取决于 @ 与 λ 之间是否有 S。

（iii）类型准备 在树中检测绑定数值变量 n 的标准方法是：沿着以 n 结尾的分支向上追溯，每遇到一个 λ 就减 1。使 λ 从 1 变为 0 的位置，就是 n 的绑定 λ 。（示例见图 1（i）。）

理想情况下，我们对分支的关注应适合于向类型化 λ 演算的扩展（参见（[6]））。然而，在所述对绑定 λ 的搜索中存在一个令人困扰的异常。确切地说：从 n 到 0、沿分支向上的计数，应绕过每一个位于从右下方接近的分叉处的 λ。原因在于，此类 λ（表示一种类型）正下方的子树并不包含受该 λ 约束的数值变量。这一点已在（[12], 第 2 节。）中明确指出。我们在图 2 (i) 中说明这一异常。



类型化抽象

图 2：变量的绑定；传统方式与改编方式

(iv) 跳过无用的 @ 符号 与 (iii) 中类似的说明在无类型情形和类型化情形中均成立，即当从右下方接近的不是 λ，而是一个 @ 时。实际上，在分支的“编码”中也应跳过这样的 @，因为该 @ 没有任何影响。

这有些棘手。对于 (iv)（以及 (iii)！），我们的解决办法是将所有标签 @ 和 λ 下移，也就是说，让它们附着在原标签正下方的边上（而不是顶点上）。此外，我们将新标签 S 附着在分叉正下方的边上。除此之外，我们为每个数值变量添加一条边，并将该变量附着在这条新边上。参见图 1 (ii) 和图 2 (ii)。

关于省略 @ 符号，参见例 1.1 (2)、(3) 和 (4)：比较传统分支与适应后的分支。

例 1.1。图 1 中的树有五个分支。从根到叶，沿着这些分支依次排列的标签列表如下。树的分支按照从左到右的顺序排列。

在传统树中：	在适应树中：
(1) @ λ@ @λ λ@ 2	A L A A L L A 2
(2) @ λ@ @λ λ@ 2	A L A A L L S 2
(3) @ λ@ @λ 2	A L A S L 2
(4) @ λ@ 1	A L S 1
(5) @λ 1	S L 1

注1.2。传统树中的分支 (1) 和 (2) 相同，但在适应树中不同。

注 1.3。树表示的另一个优点是，无需额外的标记或其他技术干预来界定子项；而在对 λ-项的线性表示执行显式替换的过程中，通常需要这样做。例如，考虑一个显式替换算子，比如 Σ，我们想让它在书写成线性形式的 λ-项中向前“推进”一步：$\ldots \Sigma t_1 t_2 \ldots$。我们假设子项 $t_1 t_2$ 写作函数 t_2，其前面是自变量 t_1；参见（*[8]*）。那么， Σ推进后的结果可能是……（$\Sigma t_1 \natural$）Σt_2……，其中插入符号 ♮界定了子项 t_1, ，从而使 Σ 的第一个副本能及时停止（并被擦除）。参见（*[18]*, 第 7 页）和（*[19]*,第 5、6 页）。

1.3 λ-树和路径

我们将图 1（iii）所示适应树中的 λ 项之树称为 λ-树。见下文定义1.4（i）。对于 λ-树，我们用“路径”一词表示一个分支或分支的一部分，并用“数值标签”或“数值变量”表示数值变量。

定义1.4. (i) λ-树是连通无环无向图，记作 t、t′、.....，由下述归纳定义构造。这种树必须非空、有根且边标记的。

设 t、t_1 和 t_2 是 λ-树，n 是一个正自然数。那么以下这些树也都是 λ-树 s:



（注：出于技术上的原因，我们将数值标记边画成一端悬空、没有下方节点的形式。这样便于用一棵新的 λ-树延伸变量标记边，正如我们在第4节中所做的那样。）

(ii) 标签可以是 A、L、S 中的一个，也可以是任意的 $n \in \mathbb{N}^+$，由 ℓ、.....遍历。(iii) 树 t 中的一条路径，是 t 中出现的、向下记录的连通的标记边序列。（一条路径可能只确定某个分支的一部分。）路径用 p 、 q 、 ..... 表示。

引理1.5。在一个 λ-树中，沿不同路径可以找到不同的标签序列。

证明 设 p_1 和 p_2 是 t 中不同的路径。找出两条路径发生偏离的最左位置。这必定是一个分叉。因此，在该位置，一个标签是 A，另一个是 S。由此，沿 p_1 和 p_2 的标签序列也不同。 □

这个引理使我们能够用标签序列来确定一条路径。另见注1.2和定义1.7。我们为特殊类型的路径命名。

定义1.6. 设 t 是一棵 λ-树，p 是 t 中一条非空路径。记号：$p \in t$ 且 $p \neq \varepsilon$。

如果 p 从 t 的根开始，则称 p 是 t 的根路径。记号：$p \in^{\wedge} t$。如果 p 的最终标签是一个叶，则称 p 是叶路径。记号：$p \in_{\vee} t$。如果 p 既是根路径又是叶路径，则称 p 是完整路径。记号：$p \in^{\wedge}_{\vee} t$。

完整路径的例子：见例 1.1 右侧。

定义1.7. 设 $p_1, \ldots, p_n$ 是 λ-树 t 中的所有完整路径。那么我们将 t 与集合$\{p1, \ldots, p_n\}$ 视为相同。

定义1.8. (i) λ-树 t 中路径 p 的长度 $|p|$ 是 p 中标签（包括数值标签）的数量。

(ii) 路径 $p \in t$ 的 L-长度 $\|p\|$ 是 p 中出现的标号 L 的个数。

在 namefree λ 演算中，变量绑定——如带名 λ 演算中所知——由完整路径末端的数值标签 n 的值来表示。该标签 n 表示一个变量。建立标号 L 与 num-变量之间的绑定的过程，已在上一节标题为类型预备性的部分讨论过。现在我们可以给出绑定的一个简单定义。（注：路径 pL q n 是路径 p、标签 L、路径 q 与标签 n 的拼接。）

通常的 n otions“受约束”与“闭合”在 λ 演算中由以下定义涵盖 .

定义 1.9. (i) 设 t 为 λ 树，且 $pL\,q\,n \in^{\wedge}_{\vee} t$ 使得 $n = \|q\|+1$。则该 n 受所述 L 约束。此外，路径 L q n 称为（该次出现的）n 的 L-块。

（ii）如果 t 中所有 num-变量都由某个 $L \in t$ 绑定，则 λ-树 t 是闭合的。

请注意，某个 num-变量 n 的一次出现的绑定符 L 总是出现在以 n 的这次出现为终点的（唯一）完整路径中。

引理 1.10. (i) 某个 $n \in t$ 的 L-块（如果存在）是唯一的。

(ii) 在闭项中，每个数值变量 n 的出现都恰好对应一个 L-块；但即使该项是闭合的，也并非每个 L-块都会绑定某个数值变量。

每棵 λ-树都对应一个定义明确的完整路径集合 $\mathscr{S}$。一个自然的问题是：给定的路径集合 $\mathscr{S}$ 何时定义了一棵可以按照定义1.4构造的 λ-树?

由于这个定义是归纳式的，因此用于判定这个问题的直接过程同样是归纳式的，也就不足为奇了。现在我们用文字描述这样一个过程。

过程 1.11. 首先，我们要求 $\mathscr{S}$ 中任意一条路径都只由 {L、A、S} 中的元素组成，但该路径的最终标签例外，它必须是一个正自然数。我们称这样的路径为合法路径。

因此，我们可以假设 $\mathscr{S}$ 由合法路径组成。进一步假设 $\mathscr{S}$ 是有限的，并且 $\mathscr{S}$ 中的所有路径也都是有限的。为了简化下面给出的过程的描述，我们假设这些路径按照某个基本顺序按字典序排列，例如 $L < A < S < 1 < 2 < \ldots$。相应地，我们将这些路径编号为：$p_1, p2, \ldots, p_n$。

下面给出针对这样一个由合法路径组成的集合 $\mathscr{S}$ 的过程:

情况 1: 设对于某个 q_1，有 $p_1 \equiv L\ q_1$。那么

要求 1 对所有 $1 \le i \le n$: 对于某个 q_i，有 $p_i \equiv L\ q_i$。

从所有这些路径中删去开头的 L；于是得到 $\mathscr{S}' \equiv \{q_1, q2, \ldots q_n\}$。对 $\mathscr{S}'$应用该过程。

情况 2: 设对于某个 q_1，有 $p_1 \equiv A\ q_1$。那么

要求2 必须存在某个 p_i，使得 $p_i \equiv S\ q_i$。

根据字典序，现在必然存在一个 $1 \le m \le n$，使得所有满足 $k \le m$ 的 p_k 都以 A 开头，且所有满足 $k > m$ 的 p_k 都以 S 开头。

将集合 $\mathscr{S}$ 分成两部分:

$\mathscr{S}_1 := \{p_k \in \mathscr{S} \mid p_k$ 以 A 开头$\}$，以及 $\mathscr{S}_2 := \{p_k \in \mathscr{S} \mid p_k$ 以 S 开头$\}$。

跳过 $\mathscr{S}_1$ 中所有路径的前导 A 以及 $\mathscr{S}_2$ 中的前导 S，并将它们收集起来。我们得到 $\mathscr{S}_1'$ 和 $\mathscr{S}_2'$。将该过程应用于 $\mathscr{S}_1'$ 和 $\mathscr{S}_2'$。

情形3: 设对某个正数 n，有 $p_1 \equiv n$。则

要求 3 $\mathscr{S} \equiv \{p_1\}$。

如果有一项要求未得到满足，我们就中止该过程，并给出答案“否”。不难证明，当且仅当原始的 $\mathscr{S}$ 是属于某一特定 λ-树的所有路径的集合时，该过程才会在不中止的情况下结束（答案为“是”）。 □

2 β归约

2.1 namefree β归约中更新的简史

在带名 λ 演算系统中，项 M 中的一个绑定子（例如 λx）以及指向它的变量出现都带有相同的名称，例如 x。相比之下，无名系统使用无名绑定子（例如 λ），并将受约束变量出现 x 替换为一个非负整数索引；该索引表示在将 M 表示为抽象语法树时，相应的 λx 沿连接 x 与 M 的根的路径所处的位置。正如我们在引言中指出的，后者系统的 β 归约步骤要求更新复制的自变量（例如 N）中出现的索引，以维持以下关系:

M 中的受约束变量实例与相应绑定子之间的关系。根据具体系统的不同，如果采用即时更新，则通过对 N 中的索引应用所谓的更新函数来执行更新。相比之下，如果采用延迟更新，则更新函数仅存储在复制的 N 的语法中。

最早采用即时更新的 namefree 系统出现在（[8]）中，其基本更新函数为$\tau_{d,\ h}$类型为 $\mathbb{N}^+ \to \mathbb{N}^+$，其中 d $\in \mathbb{N}$ 且 h $\in \mathbb{N}$。

$$\tau_{d,h} \equiv i \mapsto \begin{cases} i & \text{if } i \le d \\ i+h & \text{if } i > d \end{cases}$$

与（[11]）相关的系统——例如（[9, 10]）中的系统——最早通过引入项节点 ϕ(f) 实现延迟更新，其中 f 是类型为 $\mathbb{N}^+ \to \mathbb{N}^+$ 的任意函数。 ϕ(f) 的原始目的是将替换表示为单个操作，并通过对项的结构进行递归来定义该操作。

同一系列中的其他系统，例如（[18, 19]）、（[15]），采用项节点 μ(d,h) 或 $\phi^{(d,h)}$，其中保存着函数 $\tau_{d,\ h}$。此外，源自（[1]）的系统——例如（[7],1996）中的系统——采用显式替换构造子 id 和 ↑，它们本质上分别保存函数 $\tau_{0,\ 0}$（恒等函数）和$\tau_{0,\ 1}$（后继函数）。

2.2 路径方法中的通常 β归约

我们继续介绍 namefree λ-演算中的一些有用定义，重点讨论路径。

定义 2.1. 设 t 是一棵 λ-树，p 是 t 中一条固定的根路径。

所有满足 p q 是 t 中完整路径的路径 q $\in_\vee$ t 的集合记作 tree(p)。我们称集合{p q | q $\in_\vee$ tree(p)}为 p 在 t 中的嫁接树。

我们注意到，闭合 t 中根路径 p 的嫁接树本身也是“闭合”的，也就是说，intree(p) 中的所有自由变量都受约束于 p。

现在我们将用路径和嫁接树来描述通常的 β-归约。首先介绍众所周知的概念“可约式”（可约表达式）。

定义 2.2。设 t 是一棵 λ-树。设 pA L $\in^\wedge$t。则该路径末端的相邻对 A L 标识出一个可约式。这个可约式由两个元素组成：(1) “函数”L tree(p A L)；(2) “自变量”tree(pS)。

见图3（i）。

现在考虑通常称为 β-约化的关系，并用符号 $\to_\beta$ 表示。这种 β-约化将“把一个函数应用于一个自变量”的操作形式化。在此处研究的无名 λ 演算中， β-约化会对充当数值变量的数字产生重要影响。其中一些数字应在 β-约化之后进行“更新”。

定义 2.3。设 t_1 是一棵 λ-树。假设 pA L $\in^\wedge t_1$。（作为参考，我们将此路径中的 A 和 L 称为枢轴。）考虑相应的嫁接树 pA L tree(pA L)。则 $t^1 \to_\beta t_2$,，其中 t_2 是通过以下方式从 t_1 得到的树：

(i) 将树(p S)代入并更新（见下文）树(p A L)中所有受关键 L 约束的数值变量，(ii) 擦除关键 A-L 对，以及 (ii) 擦除嫁接树 p S tree(p S)中的所有完整路径。

定义2.4. 由于 β-归约而更新 num-变量的过程如图3所示。在该图的图(ii)中，我们根据图(i)区分以下几种情况：

(1) $n < \|q\|+1$, (2) $n > \|q\|+1$, (3) $n = \|q\|+1$, 由子情况 $l \leq \|r\|$ and $l > \|r\|$分隔,

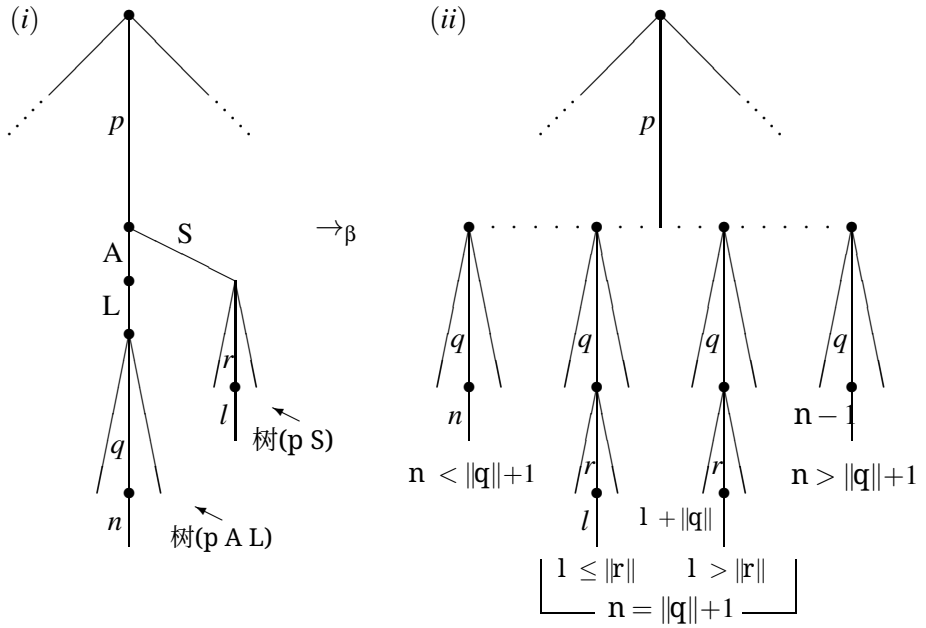


图3：带更新的 namefree β-约简示意图

定理2.5. 更新保留数值变量与其绑定L标签之间的联系。

证明 我们在图3中说明 β-归约时会发生什么。图的第(i)部分展示了可约式的关键部分；同一图的第(ii)部分展示了更新的过程。

我们如下论证更新过程中绑定关系的保留。为便于查阅，我们将 L 绑定 n 记为 L_n，将 L 绑定 l 记为 L_l。我们还以一种不寻常的方式使用符号 $\to_\beta$。我们用 m^{upd} 表示更新后的 m。

只需检查两条代表性路径：

(i) 在 p A L 树(pA L) 中，我们选取 pA L q n，其中 q n 是树(p A L) 中的一条一般路径；
(ii) 在 pS 树(pS) 中，我们选取 pS r l，其中 r l 是树(p S) 中的一条一般路径。

我们按照定义2.4所述，区分 n 的各种情况：

(1) $n < \|q\|+1$。则 $p\,A\,L\,q\,n = p\,A\,L\,q_1 L_n q_2 n \to_\beta p\,q_1 L_n q_2 n^{upd}$，因此 $n^{upd} = n$。(2) $n > \|q\|+1$。则 $p\,A\,L\,q\,n = p_1 L_n p_2 A\,L\,q\,n \to_\beta p_1 L_n p_2 q\,n^{upd}$，因此 $n^{upd} = n - 1$（因为关键 L 已被擦除）。(3) $n = \|q\|+1$。则关键 L 绑定到 n。现在我们必须区分 l 的两种情形：(3a) $l \leq \|r\|$。则 $p\,S\,r\,l = p\,S\,r_1 L_l r_2 l$，因此 $p\,A\,L\,q\,n \to_\beta p\,q\,r_1 L_l r_2 l^{upd}$，所以 $l^{upd} = l$。(3b) $l > \|r\|$。则 $p\,S\,r\,l = p_1 L_l p_2 S\,r\,l$，因此 $p\,A\,L\,q\,n \to_\beta p_1 L_l p_2 q\,r\,l^{upd}$，所以 $l^{upd} = l + \|q\|$，因为 q 现在出现在 L_l 和 r 之间。 □

2.3 比较带名和namefree λ 演算中的β归约

到目前为止，我们所介绍的 λ-演算中的项集合，着重关注路径，记作$\mathscr{T}^{fre}$。在本节中，我们将其与同样着重关注路径的带名 λ-演算进行比较，并将其称为 $\mathscr{T}_{car}$。我们不解释 $\mathscr{T}_{car}$ 中的项长什么样。我们假定读者能够轻松推想出来。与 $\mathscr{T}_{fre}$ 相比，最重要的区别是：

(1) $\mathscr{T}_{car}$ 使用实际变量（如 x、y.....），而非 num-变量。
(2) $\mathscr{T}_{car}$ 中的每个 L标签都以下标变量表示，例如，某个 x 对应的 L_x 。

因此， $\mathscr{T}_{car}$ 中的 L-块表示为 L_x px，而不是 L pn。依此类推。特别是，我们不再重复说明 $\mathscr{T}_{car}$中的绑定方式。

请注意，我们假设 λ-树在 $\mathscr{T}$car 中是闭合的，并且在一个 λ-树 t 中，受约束变量彼此不同。

在本节余下部分，我们将给出一些关于 $\mathscr{T}_{car}$ 和 $\mathscr{T}_{fre}$相关事实的简单结果。

最重要的是， $\mathscr{T}_{car}$ 和 $\mathscr{T}_{fre}$中的 β-归约之间存在一个著名的同构。见下文的引理2.10。首先，我们定义 $\mathscr{T}_{car}$ 与 $\mathscr{T}_{fre}$ 之间的映射，以及反向映射。

过程 2.6。设 s $\in \mathscr{T}_{car}$。则可通过以下方法得到 [s] $\in \mathscr{T}_{fre}$ 。

(i) 设 x 是 s 中的一个变量，它在 $\mathscr{T}_{car}$ 中通过 L-块 L_x px 受约束。将此 x 替换为 $\|p\|+1$。对所有数值标签都执行此操作。

(ii) 删除 s 中标签 L_x 下方的所有下标，例如 $x_\circ$

过程 2.7。设 t $\in \mathscr{T}_{fre}$。然后按如下方式得到 ⟨t⟩ $\in \mathscr{T}_{car}$ 。

对于每个标签 L $\in$t，找出所有受 L 约束的数值变量 $n1, \ldots, n_k$。（如果提到的 L 是路径 pL $\in$^t 中的最终标签，那么受约束的数值变量就出现在树 tree(p L) 中。）

现在用一个新变量 x（即此前在此过程中尚未使用过的变量）将 L 替换为 L_x；并且（如果受约束的 n_i's 的数量不为零）将其中每一个 n_i 替换为 x。

我们将 α-等价扩展到 $\mathscr{T}_{无名}$。我们假定读者理解下文定义和引理中"对应"的含义。

定义2.8。（i）设 t 是 λ-树，属于 $\mathscr{T}^{car}$ 或 $\mathscr{T}^{fre}$。通过去除 t 中所有变量和数值标签（包括标签 $L_x \in \mathscr{T}$ 的下标）来得到 t 的无变量树$_{带名}$。其他所有标签和所有边（包括那些以数值标签为标签的边）均保持不变。

(ii) 设 t $\in \mathscr{T}_{car}$，且 t′ 是 λ-树，属于 $\mathscr{T}_{car}$ 或 $\mathscr{T}_{fre}$。如果 t 和 t′ 的无变量树相同，且 t 中的绑定与 t′ 中的绑定一一对应，那么 t 与 t′ 是 α-等价的（记作：t $\equiv_\alpha$t′）。

Le引理 2.9. (i) 映射 $[-]$和 $\langle - \rangle$ 互为逆映射（模 α-变换）。 .

(ii) 设 s $\in \mathscr{T}$car，t $\in \mathscr{T}$fre。那么 s $\equiv_\alpha$ [s]，且 t $\equiv_\alpha$ ⟨t⟩。

现在我们证明，映射 [ ] 从 $\mathscr{T}_{car}$ 到 $\mathscr{T}^{fre}$，会"保留"变量与 L 标签之间的绑定关系。

引理2.10. 设 $s_1 \in \mathscr{T}$car，且 $s^1 \to_\beta s_2$ 是通过枢轴为 L_x 的 β-归约得到的。那么在 $\mathscr{T}$fre 中存在相应的β-归约 $[s_1] \to_\beta [s_2]$，其枢轴相应为 L。

证明 在归约 $s^1 \to_\beta s_2$ 中，绑定得以保留（这是公认的事实）。此外，我们有 $s_1 \equiv_\alpha [s_1]$（引理2.9 (ii)），并且在归约 $[s_1] \to_\beta [s_2]$ 中，绑定同样得以保留（定理2.5）。注意，s_2 和 $[s_2]$ 的无变量树是相同的（这可由 $[s_2]$ 的构造过程推出）。因此，$s_2 \equiv_\alpha [s_2]$。

Accor 相应地，对于倒置情形也有一个引理。我们不再赘述 it.

3 替代 β-归约

3.1 无名 λ 演算中的平衡 β-归约

如果保留原始 λ-演算项中的所有信息更有利，那么 β-归约有一个值得关注的变体。在这种情况下，参数树(pS)应保留在 β-归约后的项中，就像关键 A-L 对（见定义 2.1 和 2.2）一样。我们将这种归约关系称为平衡β-归约。在文献中，它最初以 β_1 ([17]) 之名出现。详情请参阅较新的线性替换演算文献（参见（[2]）和（[3]）），其中这种归约被称为远距 β，符号为 $\rightarrow_{dB}$。另见（[5]）和（[14]）。

我们从 λ-树中平衡路径的定义开始。

定义 3.1. λ-树 t 中的路径 p 称为平衡的，记作 bal(p)，如果它由以下归纳规则构造而成：

(i) bal(ε)，即空串是平衡的；
(ii) 若 bal(p)，则 bal(A p L)；
(iii) 若 bal(p) 且 bal(q)，则 bal(p q)。

在情形 (ii) 中，我们称所提及的 A 与所提及的 L 相匹配。

平衡路径的例子： ε、A L、AA L L、A L AL、AA L A A L L L。

注：平衡路径与（连续的）嵌套括号对之间存在密切对应。注：平衡路径上只出现 A- 与 L 标签，因此不涉及其他标签，例如 S。

现在，如上所述，在 λ-树 t 中维持关键 A-L 对会产生严重后果：它可能阻止在‘正常’ β-归约中出现的其他 β-归约实例。例如，若路径 p A A LL $\in^\wedge$t 中带下划线的对 A L 是枢轴对，则维持该对会阻止另一个 A 与另一个 L~~在~~由 A L 诱导的单步 β-归约之后成为新的枢轴对。在‘正常’的 β-归约中，这种情况不会发生，因为此时 A L 会消失。

使用平衡 β-归约可以避免这种情况。

下面的定义介绍了“平衡归约”这一概念。

定义 3.2。设 t 是一棵 λ-树，b $\in$t 是一条平衡路径，并假设 pA bL $\in^\wedge$t。这条根路径称为活跃的，如果至少存在一条路径 pA bL q n $\in^\wedge_\vee$t，使得 n 在 b 之后受 L 约束。如果不存在这样的路径，则该根路径是非活跃的。

现在我们给出 β-归约的“平衡”变体，其符号为 $\rightarrow_b$。我们回顾一下：“L qn 是一个 L-块”等价于“n 受初始 L 约束”。

定义 3.3。设 t 是一棵 λ-树，b $\in$t 是一条平衡路径，并假设 p A b L $\in^\wedge$t 是一条活跃根路径。令 t′ be t，其中所有满足 L q n 为 L-块的路径 p A b L q n 均被替换为 p A b L q tree(pS)。然后 t $\rightarrow_b$t′。

此定义要求 pA bL 处于活跃状态，是为了避免由所述根路径生成无限约化路径。

定义 3.4. (i) 定义 3.3 中显示的 L 称为关键 L。

(ii) 设 t 是一个 λ-树，且具有 r = pA bL q n $\in$t，其中 b 是一条平衡路径，并且 L $\in$r 绑定到 n。回顾一下，L q n 称为 L-块（定义 1.9）。我们将 A b L q n 称为 A-块。

考虑两棵 λ-树 t 与 t′，使得 t $\rightarrow_b$ t′，如定义 3.3 所述，从而每个受关键 L 约束的 n 都已在树(pA bL)中被树(p S)替换。现在可知，只要省略 t 中所有受关键 L 约束的 num-变量 n，并同时省略相应的边，则 t（几乎）是 t′ 的一棵子树。因此，平衡 β-归约具有如下性质：它扩展了原始底层树 t，只是有若干 num-变量会消失。

3.2 无名 λ 演算中的聚焦 β归约

聚焦 β-归约是平衡 β-归约的一种特殊情形。该归约精确地集中于某棵 λ-树 t 中特定位置上的一个数值变量 n，且该 n 受关键 L 约束。由于 L 是枢轴，必定存在一个与 L“耦合”的 A，因此 n 是某条特定路径 pA bL q n $\in^{\diamondsuit}$t 的最终标签，其中 b 是平衡的。聚焦 β-归约将该 n 替换为连接到枢轴的自变量。见如下定义，其中引入符号 $\to_f$ 表示“聚焦” β-归约。

定义 3.5。设 t 是一个 λ-树，b 是一条平衡路径，且 r = pA bL q n $\in^{\diamondsuit}$t 是 t 中一条固定的完整路径。设 t′与 t 相同，只是将 r 替换为 pA bL q tree(pS)。那么 t $\to_f$t′。

显然，我们在这里需要某种能够保留 A-L对的 β-归约，因为可能还有其他受约束于此 L 的 num- 变量，而且之后或许希望在后续的 $\to_f$- 归约中，将其中一个或多个替换为 tree(pA)。

引入聚焦 β-归约的动机，源于带名 λ-演算中称为定义展开的过程。此时，M 中出现的一个被定义概念（例如 x）会被定义项（例如 P）替换。此操作通常只针对被定义项 x 的某一个实例。因此，人们的目标不是替换 M 中 x 的所有出现，而是精确地替换其中一个出现。

这里，被定义项的“名称”很重要，因为处理一个“无名”的被定义项很困难。尽管如此，我们仍在此讨论 β归约的这一变体，因为无名变体本身很有趣。

具备平衡 β归约的可能性，是处理其他 A-L对所必需的，否则这些 A-L对将无法访问。请看下面的例子。

例 3.6。在带有普通无类型 β归约的 λ演算中，我们有：

(λx . ((λy . M)Q))P $\to_\beta$ (λx . (M[y := Q]))P $\to_\beta$ M[y := Q][x := P]. I

n $f_{\text{专注}}$ d β-reducti于, t his b变为：

$(\lambda x.((\lambda y.M)Q))P \to_f (\lambda x.(\lambda y.M[y_0 := Q])Q)P \to_f$

$(\lambda x.((\lambda y.M[y_0 := Q])Q)[x_0 := P])P.$

这里，y_0 和 x_0 分别是 M 中自由 y 和自由 x 的选定实例。

如果不允许在 λx 与 P 之间存在平衡的 λ-项（λy. . .）Q，那么这两个单步聚焦归约中的第二个就无法进行。

下面的引理显而易见。

引理 3.7. 设 t $\in \mathscr{T}$fre 且 t $\to_b$t′。则 t $\twoheadrightarrow_f$ t′。

3.3 擦除归约

应用平衡或聚焦的 β-归约后，人们还希望进行一种归约，以移除“残留部分”，即嫁接树 pA bL tree(pA bL) 中的 A 和 L，前提是没有 n $\in$tree(pA bL) 受约束于显示的 L。此外，“自变量”树 tree(pS) 必须与所提到的 S 一并移除。我们将相应的归约称为擦除归约，并用符号 $\to_e$ 表示。（文献中也将这种归约称为“垃圾回收”；例如，参见（[23]）。）

定义 $\to_e$ 的定义并不容易，因为擦除作用于原始树 t 的不同部分。

定义 3.8。设 t 是一棵 λ-树，并假设 t 中存在某个 p A b L $\in^{\wedge}$，其中 b 是平衡的。此外，假设 tree(pA bL) 中没有数值变量受约束于上述 L。

于是 t $\to_e$t′，其中 t′是这样一个 t：移除了 S tree(p S)，并将 tree(p) 替换为 tree(pA)，而其中的 tree(pA b) 又被替换为{q n $\in^{\diamondsuit}$tree(pA bL) | n；如果 n > ||q||}，则将其替换为 n − 1。

当然，必须将 n 替换为 n – 1，是由于擦除了上述的 L。反复应用 $\to_e$ 将产生一个没有垃圾的λ-树。

3.4 定理

引理 3.9。归约 $\to_e$ 是强正则化的，并且具有唯一的范式。

我们将归约 $\to_i$ 的自反传递闭包记作 $\twoheadrightarrow_i$。任意的归约序列 $\to_i$ 和 $\to_j$ 记作 $\twoheadrightarrow_{i,j}$。

定理 3.10。设 t 和 t′ 是 λ-树。

(i) 如果 $t \twoheadrightarrow_\beta t'$，那么 $t \twoheadrightarrow_{b,e} t'$。(ii)（将 $\to_e$ 推迟到 $\to_b$ 之后）如果 $t \twoheadrightarrow_{b,e} t'$，那么存在 t″ 使得 $t \twoheadrightarrow_b t'' \twoheadrightarrow_e t'$。(iii)（将 $\to_e$ 推迟到 $\to_f$ 之后）如果 $t \twoheadrightarrow_{f,e} t'$，那么存在 t″ 使得 $t \twoheadrightarrow_f t'' \twoheadrightarrow_e t'$。

证明 (i) 显然。(ii)（[17], 第 48 页，定理 6.19）。(iii) 类似。□

定理 3.11。$\to_b$、$\to_f$ 和 $\to_e$ 是合流的。

证明 关于 $\to_b$ 和 $\to_e$，参见（[17], 定理 6.38 和 6.42）。关于 $\to_f$，参见（[3]）。

4 一种新的无损 β-归约

4.1 扩展 β-归约

我们将要介绍的系统采用了一种比第 2.1 节中提到的方法更简单的方案。其语法包含一个项节点 m，我们称之为内部数值标签，其中 $m \in \mathbb{N}^+$。一个活跃的 m 持有函数 $\tau_{0,\ m}$，而一个被动的（即存在但被忽略的）m 持有函数 $\tau_{0,\ 0}$。从某种意义上说，我们希望表明，支持函数 $\tau_{0,\ h}$ 就足以在基本的 namefree λ-演算中实现延迟更新。

在第 3.1 节末尾，我们提到，当 $t \to_b t'$ 时，λ-树 t 几乎是 t′ 的子树。“几乎”指的是：在平衡归约中，受关键 $L \in t$ 约束的数值变量会被移除，因此它们不会在 t′ 中重新出现。

在本节中，我们探究如果将受 L 约束的数值变量留在原处，会发生什么。在这种情况下，t 就成为 t′ 的真子树。我们可以说，所得归约具有这样的性质：从 t 到 t′ 的归约没有丢失 t 中的任何信息。

为了实现这一点，我们必须扩展“路径”的概念：现在，数值变量可以出现在路径中的任何位置，而不只是末尾。

定义 4.1。扩展路径是由 L、A、S 标签和任意数值标签组成的有限串，并以一个数值标签结尾。

例：L L 2A A 1A L L 2。

现在，数值标 号分为两类：位于路径内部或位于末端。我们还得到一种新型的 λ-树 s.

定义 4.2。（i）不是终端标签的数值变量称为内部数值标签。作为终端标签（叶子）的数值变量称为外部数值标签。

（ii）允许存在内部变量的 λ-树称为扩展 λ-树。

因此，平衡路径的定义（定义 3.1）也必须相应调整，使其允许 A和L 的字符串中包含内部数值标签：从现在起，“平衡路径”这一概念均指“扩展的”平衡路径。

在本节余下部分，我们假设这些关于路径、平衡路径和λ-树的新定义是有效的。此外，对于新的路径和 λ-树，我们将省略“扩展”一词。

我们回顾第2.3节，符号 $\mathscr{T}_{fre}$ 涉及无名闭树的集合（不含内部变量）。允许包含内部变量的无名闭（扩展） λ-树集合记为 $\mathscr{T}_{exp}$。

显然， $\mathscr{T}_{fre} \subseteq \mathscr{T}_{exp}$。

由于 β-类归约是对含内部变量情形进行扩展的结果，我们称之为扩展β-归约。同样，这种归约有两种显而易见的形式：平衡归约或聚焦归约。从现在起，我们将专注于聚焦归约，因为这种归约可用于模拟平衡归约（参见定理3.10 (ii)）。

我们用符号‘$\rightarrow_{ef}$’表示扩展聚焦 β-归约。其定义如下。

定义4.3。设 t $\in \mathscr{T}_{\text{表达式}}$，设 b $\in$t 是一条平衡路径（现在可能包含内部变量），并假设 r $=$p A b L q n$\in\!\Diamond$t 是 t 中一条固定的完整路径，其中 n 受所示 L 约束。令 t′与 t 相同，只是将 r 替换为 p A b L q n tree(p S)。那么 t $\rightarrow_{ef}$t′。

$\rightarrow_{ef}$-归约的效果是，终端标签 n 和标记为 n 的边保持原位，而 tree(pS) 则直接附接到这条边上（回想定义1.4 (i) 后的注）。我们将看到，标签 n 仍然保留，这使得稍后能够进行更新。

图示见图4。注：如果 tree(pS) 仅由一条标记为数值变量的边构成，那么这条边就直接附接到标记为 n 的边上。

我们定义（扩展的） λ-树的包含关系的含义。

定义4.4. 设 t 与 t′ 为（扩展的）λ-树。则若 p$\in\!\Diamond$ t 蕴含 p $\in\!\wedge$ t′，则 t $\subseteq$t′。此外，若 t$\subseteq$ t′ 且 t $\neq$ t′，则 t $\subset$t′。

定理 4.5. 设 t、t′ $\in \mathscr{T}$exp。则 t $\rightarrow_{ef}$t′ 蕴含 t $\subset$ t′。

证明 显然。 □

引理 4.6. 设 t $\in \mathscr{T}$exp 且 p n $\in\!\wedge$ t。假设 n 是内部数值变量。则存在绑定该 n 的 L $\in$ p，以及一个匹配的 A $\in$ p。

证明 内部数值标签 n 只能出现在 $\rightarrow_{ef}$-归约中，此时 L 绑定 n 在该归约中为枢轴，且该 L 与一个 A 相匹配（见定义4.3）。这一情形由 $\rightarrow_{ef}$-归约的扩展性所保持（定理 4.5）。 □

4.2 在扩展 β-归约中追踪绑定子

设 $t_0 \in \mathscr{T}_{exp}$ 且 $t_0 \twoheadrightarrow_{ef} t$，因此 $t \in \mathscr{T}_{exp}$ 是一系列扩展的聚焦归约的结果。这些归约可能引入内部变量，因此（内部或外部）变量对应的绑定符是什么并不立即清楚。在本节中，我们研究如何确定 t 中某个数值变量的绑定子。

设 p n $\in\!\wedge$ t。此处，n 可以是内部或外部数值标签。我们描述一个下推自动机 $\mathscr{P}$，它找出 n 的 L 绑定子，即绑定 n 的标签 L $\in$ p （该标签始终存在，因为 $\mathscr{T}_{exp}$ 只包含闭项）。

我们现在介绍并解释下推自动机 $\mathscr{P}$ 的动作。设 t $\in \mathscr{T}_{\text{表达式}}$ 且 pn $\in\!\wedge$ t。假设我们希望应用算法 $\mathscr{P}$ 来找出 n 的 L 绑定子。

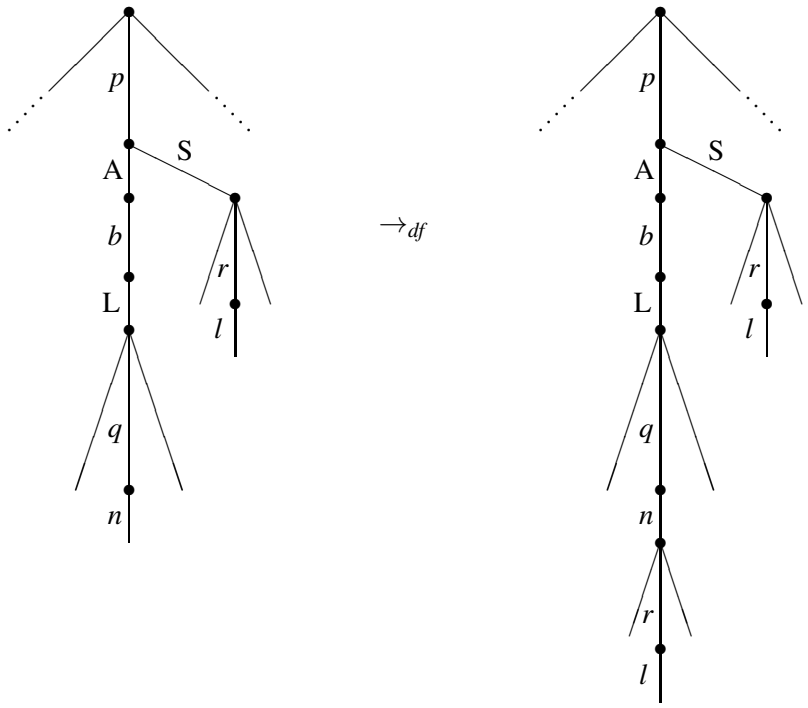


图4：namefree 的扩展 β-约简示意图

注4.7。预备说明。

在算法 $\mathscr{P}$ 中，我们使用由自然数对 (k, l) 组成的状态。首先，在字符串 p n 中，将数对 (n, i) 插入 p 和 n 之间。这里，i 最初为 0 或 1。自动机每次移动一步，使该数对向左穿过 p，依次越过 p 中的标号，同时调整数对中的数字。

自动机有一个外部栈，其中会存放某些状态；这些状态被压入栈顶。栈顶的状态也可以弹出并再次插入路径 p 中。

转换在定义4.8中描述。可能的一步转换用符号 $\rightarrow$ 表示。

该过程可能因多次递归调用而变得复杂。每次递归调用中，算法都会从路径 p 上一个“新的”数值变量 j 开始。随后，它不仅会找到与 j 绑定的 L 绑定，还会找到与 j 匹配的 A。

$\mathscr{P}$ 的形式化描述如下。

过程 4.8。准备：将 p n 变换为 p(n, i) n，其中若目标是找到 L 绑定子，则 $\mathrm{i} = 0$；在递归中，当同时检测到 n 的 L 绑定子与匹配的 A 时，$\mathrm{i} = 1$。

现在开始 $\mathscr{P}$ 采用以下转移规则。

(1) 第一步：栈 = /0 (2) p L (m,k) q $\rightarrow$ p (m–1,k) L q，当 $\mathrm{m} > 0$(3) p A (m,k) q $\rightarrow$ p (m,k) A q，当 $\mathrm{m} > 0$(4) p S (m,k) q $\rightarrow$ p (m,k) S q，当 $\mathrm{m} > 0$(5) p j (m,k) q $\rightarrow$ p (j,1) j q，当 $\mathrm{m} > 0$；压入 (m, k)

(6) p L (0,l) q $\to$ p (0,l+1) L q，当 l >0(7)
p A (0,l) q $\to$ p (0,l–1) A q，当 l >0(8) p j
(0,l) q $\to$ p (0,l) j q，当 l >0(9a) p (0,0) q
$\to$ p 弹出 q，当栈 $\neq$ /0 (9b) p (0,0) q $\to$ 停止，当栈 $=$ /0。

引理 4.9。设 t $\in \mathscr{T}$exp 且 p n $\in$^t。将 $\mathscr{P}$ 应用于 p(n, i) n，其中 i $=0$ 或 1。

(i) 如果 $\mathscr{P}$ 在 p′(0, 0) q′n 处停止，且栈为空（参见规则 9b），则所有可能的递归都已结束。此外，q′n $\equiv$L q″n，且提及的 L 绑定了 n。

(ii) 如果 $\mathscr{P}$ 在 p′(0, 0)q′n 处停止，且栈非空（参见规则 9a），则已找到与 n 的绑定符 L 相匹配的 A，栈顶元素被弹出，而 $\mathscr{P}$ 从递归步骤之前停止的位置继续。

证明：见第 5 节。

例 4.10。我们要在路径 A L L L A L S 1L 3 中找出最终数值标签（即 3）的 L 绑定子。

因此，我们从 $\mathscr{P}$ 开始，在倒数第二个标签和最终标签之间插入 (3,0)。接下来，我们得到以下步骤序列。（箭头上的上标指明了所用规则的编号。）

A L L L A L S 1 L (3, 0) 3（栈 $=$/0） $\to^{(2)}$A
L L L A L S 1(2, 0) L 3（压入 (2, 0)） $\to^{(5)}$
A L L L A L S (1, 1) 1 L 3 $\to^{(4)}$A L L L A L
(1, 1) S 1 L 3 $\to^{(2)}$A L L L A (0, 1) L S 1 L
3 $\to^{(7)}$A L L L (0, 0) A L S 1 L 3（弹出 (2, 0)）
$\to^{(9a)}$A L L L (2, 0) A L S 1 L 3 $\to^{(2)}$A L L
(1, 0) L A L S 1 L 3 $\to^{(2)}$A L (0, 0) L L A L S
1 L 3 $\to^{(9b)}$（栈 $=$/0：停止）L-块 3

由此可知，路径 A L L L A L S 1L 3 上的最终标签 3 受约束于左侧第二个标签 L。

5 后续工作与致谢

关于第4.1节所介绍的扩展 β-归约概念，还有许多内容值得探讨。我们计划在即将发表的论文中对此作进一步阐述，其中将包括常规证明、关于扩展β-归约的定理，以及该归约的弱正规化与强正规化之间的联系。

自 2021 年起，作者之一 Ferruccio Guidi 便使用 Matita 定理证明器 ([4]) 对所有证明进行形式化并加以检验。

我，Rob Nederpelt，特别感谢 Vincent van Oostrom 对本论文预印本的关注。我也感谢 Herman Geuvers 的鼓励性意见，以及他对本论文早期版本所做的细致审阅。我，Ferruccio Guidi，谨将本书页中呈现的成果以及未来的成果献给 Anyelis Marielbys Parra Pire——一位特别的朋友，她始终陪伴在我身边，伴我完成这项工作的开展。两位作者感谢匿名审稿人认真审阅本文，并提出修改和改进意见。

参考文献

[1] M. Abadi、L. Cardelli、P.-L. Curien 和 J.-J. L´evy（1991）：显式替换。《函数式编程杂志》1(4)，第 375–416 页，doi:10.1017/S0956796800000186。[2] B. Accattoli 和 D. Kesner（2010）：结构λ演算。载于 A. Dawar 和 H. Veith（编）：《计算机科学逻辑》，LNCS6247，Springer 柏林、海德堡，第 381–395 页，doi:10.1007/978-3-642-15205-4 30。第 24 届国际研讨会 CSL 2010暨EACSL第19届年会，2010年8月23—27日，捷克共和国布尔诺，论文集。[3] B. Accattoli 和 D. Kesner（2012）：置换 λ演算。载于 N. Bjørner 和 A. Voronkov（编）：《程序设计、人工智能与推理逻辑》，LNCS 7180，Springer 柏林、海德堡，第 23–36 页，doi:10.1007/978-3-642-28717-6 5。第18届国际会议 LPAR-18，2012年3月11—15日，委内瑞拉梅里达，论文集。[4] A. Asperti、W. Ricciotti、C. Sacerdoti Coen 和 E. Enrico Tassi（2011）：Matita交互式定理证明器。载于 N. Bjørner 和 V. Sofronie-Stokkermans（编）：《自动演绎——CADE-23》，LNCS6803，Springer 柏林、海德堡，第 64–69 页，doi:10.1007/978-3-642-22438-6 7。第23届自动演绎国际会议，2011年7月31日至8月5日，波兰弗罗茨瓦夫，论文集。[5] P. Barenbaum 和 E. Bonelli（2017）：最优性与线性替换演算。载于 D. Miller（编）：《第2届计算与演绎形式结构国际会议（FSCD 2017）》，信息学莱布尼茨国际会议论文集（LIPIcs）84，达格施图尔宫 – 莱布尼茨信息中心，pp. 9:1–9:16，doi:10.4230/LIPIcs.FSCD.2017.9。[6] H.P. Barendregt（1992）：带类型λ演算。载于 S. Abramsky、D.M. Gabbay 和 T. Maibaum（编）：《背景：计算结构》，《计算机科学逻辑手册》2，Clarendon Press，第 117–309 页，doi:10.1093/oso/9780198537618.003.0002。[7] P.-L. Curien、Th. Hardin 和 J.-J. L´evy （ 1996 ）：显式替换弱演算与强演算的合流性性质。《美国计算机学会杂志》 43(2)，第 362–397 页，doi:10.1145/226643.226675。[8] N.G. de Bruijn（1972）：使用无名哑元的 λ 演算记法：一种自动公式处理工具及其在丘奇-罗瑟定理中的应用。《数学研究》75(5)，第 381–392 页，doi:10.1016/1385-7258(72)90034-0。另见 [20]。[9] N.G. de Bruijn（1977）：一种无名称λ演算，其公式包含表示引用变换映射的符号。备忘录 1977-10，埃因霍温理工大学数学系。另见 Automath档案 AUT 050，www.win.tue.nl>Automath。[10] N.G. de Bruijn（1978）：一种支持表达式和片段内部定义的无名称λ演算。EUT报告 78-WSK-03，埃因霍温理工大学。另见 Automath档案 AUT 059，www.win.tue.nl>Automath。[11]N.G. de Bruijn（1978）：使用无名称公式的 λ 演算记法，其中包含表示引用变换映射的符号。《数学研究》81(1)，第 348–356 页，doi:10.1016/1385-7258(78)90052-5。另见 Automath档案 AUT 055，www.win.tue.nl>Automath。[12] N.G. de Bruijn（1987）：用 λ 类型化 λ 演算推广 Automath。载于 D.W. Kueker、E.G.K. Lopez-Escobar 和 C.H. Smith（编）：《数理逻辑与理论计算机科学》，《纯数学与应用数学讲义》106，Marcel Dekker Inc.，第 71–92 页。另见 [20]。[13] F. Guidi（2009）：Landau 的《分析基础》：从 Automath 到 λ-δ。技术报告 UBLCS 2009-16，博洛尼亚大学。[14]F. Kamareddine 和 R. Bloo（2005）：德布鲁因语法与 λ 项的归约行为：无类型情形。《逻辑与代数编程杂志》62(1)，第 109–131 页，doi:10.1016/j.jlap.2004.01.001。[15] F.D. Kamareddine 和 R.P. Nederpelt（1993）：论逐步显式替换。《国际计算机科学基础杂志》4(3)，第 197–240 页，doi:10.1142/S0129054193000146。

[16] W. Kluge（2005）：抽象计算机——λ演算视角。理论计算机科学丛书：EATCS系列，Springer 柏林、海德堡，doi:10.1007/b138965。[17] R.P. Nederpelt（1973）：具有λ结构类型的类型化λ演算中的强正规化。博士论文，埃因霍温理工大学。另见 [20]。[18] R.P. Nederpelt（1979）：具有类型化和缩写功能的λ演算系统，第一部分：非形式化介绍。备忘录 1979-02，埃因霍温理工大学数学系。Automath档案 AUT 068，www.win.tue.nl>Automath。[19] R.P. Nederpelt（1980）：具有类型化和缩写功能的λ演算系统，第二部分：形式化描述。备忘录 1980-11，埃因霍温理工大学数学系。Automath档案 AUT 075，www.win.tue.nl>Automath。[20] R.P. Nederpelt、J.H. Geuvers 和 R.C. de Vrijer 编（1994）：Automath论文选集。逻辑与数学基础研究 133，北荷兰出版公司。[21] L. Regnier（1992）：λ演算与网络。博士论文，巴黎第七大学。[22] L. Regnier（1994）：λ项上的一种 ´ 等价关系。理论计算机科学 126(2)，第 281–292 页，doi:10.1016/0304-3975(94)90012-4。[23] K.H. Rose（1993）：显式循环替换。载于 M. Rusinowitch 和 J.-L. R´emy 编：《条件项重写系统》，LNCS 656，Springer 柏林、海德堡，第 36–50 页，doi:10.1007/3-540-56393-8 3。第三届国际研讨会，CTRS-92，法国蓬塔穆松，1992年7月8日至10日。会议论文集。

-