射人先射马，擒贼先擒王

——杜甫

第1章　数学基础

1.1　λ　演　算

和组合逻辑一样，λ演算的早期根源也在20世纪20年代，但常被引用的权威文献则是Church在1941年发表的“The Calculi of Lambda Conversion”。许多数学工具都是以集合为基础的，而λ演算与众不同，它是以函数演算为基础的。它允许任意高阶的函数运算，也就是说，一个函数的输入和输出都可以是函数。这个特点使它特别适于用作语义描述的表示方法，尤其是在指称语义描述中更需要用到它。但是请注意，本节给出的实际上是λ演算的操作语义，因此，有的作者在使用λ演算于语义描述之前，首先要定义λ演算本身的指称语义，本节把这一点省略了。

人们通常把函数写为$y=f(x)$的形式，这里涉及到三个因素：函数符号f，变元x和值y。通常，这三者可以属于不同的值域。例如$y=\mathrm{ent}(x)$是定义在实轴上的一个全函数，表示取变元的整数部分，这是程序库中常见的初等函数，其变元的值域（即定义域）自然是实数集，而函数的值域却是整数集。这样，三个因素属于三种不同的值域，不得不分别加以考虑。然而，各式各样的值域是无穷之多的，我们不能统统拿来逐个加以研究，尤其是函数的变元和值都可以又是函数，如

$$g(t)=\int_{-\infty}^{t}f(x)\mathrm{d}x=F(f,t) \tag{1.1.1}$$

就出现了高阶函数，即泛函。在计算机语义的数学表示中，泛函的研究是主要难点之一。所以，我们不仅需要有一种能统一处理各类值域的函数表示方法，而且尤其需要能有效地处理泛函的函数表示方法。人们发现，在逻辑学的库藏中有着现成的处理这类问题的方法，至少λ演算和组合逻辑中使用的表示法是可以考虑的。Church的λ演算因直观、好用而被人们选中。Church用来表示函数的方法叫λ表示法（注意，由于函数和其他类型的值在λ演算中是一视同仁的，我们不再区别函数和泛函这两个概念），它的基本形式是

$$\lambda\langle\text{变元}\rangle.\langle\text{表达式}\rangle \tag{1.1.2}$$

用这种方法表示的函数叫 λ 表达式，如

$$\lambda x.\, x^2+1 \tag{1.1.3}$$

就是一个 λ 表达式，其中 x 是变元，或称变量。函数值的表示方法是：用一对括号把代表函数的 λ 表达式括起来，而把变元值置于 λ 表达式之后。如，取变元值 x 为 1，则相应的函数值是

$$(\lambda x.\, x^2+1)1 \tag{1.1.4}$$

其计算方法是把变元值代入表达式中的变元，去掉前面的 λ〈变元〉，并按通常方式计算此表达式，在上例中

$$(\lambda x.\, x^2+1)1 = 1^2+1 = 2 \tag{1.1.5}$$

变元可以有多个，此时 λ 表达式的形式变元为

$$\{\lambda\langle\text{变元}\rangle.\}_1^n\langle\text{表达式}\rangle \tag{1.1.6}$$

例如，

$$\lambda x.\,\lambda y.\, x^3+y^2 \tag{1.1.7}$$

就是两个变元的 λ 表达式，把它作用于变元值 $x=1, y=2$ 时，得

$$((\lambda x.\,\lambda y.\, x^3+y^2)1)2 = (\lambda y.\, 1+y^2)2 = 5 \tag{1.1.8}$$

从这里可以看出 λ 表示法的一个优点，即函数的值可以是一个函数，由于 $\lambda x.\,\lambda y.\, x^3+y^2$ 本是一个双变元的函数，当变元 x 被具体的值 1 代替时，原来的双变元函数就成了单变元函数 $\lambda y.\, 1+y^2$。因此，用 λ 表示法可以体现函数的层次关系，即函数的函数。

变元的次序会影响函数应用的值，如[①]

$$((\lambda y.\,\lambda x.\, x^3+y^2)1)2 = (\lambda x.\, x^3+1)2 = 9 \tag{1.1.9}$$

在注意到这一点的前提下，可以把多变元的 λ 表达式简写为

$$\lambda\{\langle\text{变元}\rangle\}_1^n.\langle\text{表达式}\rangle \tag{1.1.10}$$

我们的例子也就成为

$$\lambda xy.\, x^3+y^2 \tag{1.1.11}$$

注意式(1.1.11)只是一种简写，含义未变，它的内层依旧是 $\lambda y.\, x^3+y^2$。在这个子 λ 表达式中，变量 x 未在 λ 部分出现，它有什么意义呢？它叫做自由变量，而在 λ 部分出现的变量，如 y，则叫做约束变量。我们可以按通常程序设计语言中的意义把约束变量理解为局部变量，但如果随之而要把自由变量理解成全局变量，则请谨慎为好！因为我们现在看到的只是一个 λ 表达式

$$\lambda y.\, x^3+y^2$$

① 比较式(1.1.8)。

除此以外我们什么也看不见！不过，大体上可以按数学函数中参数的意义来理解自由变量。

我们再看 $\lambda x.\, x^3+y^2$。在这里，x 和 y 的作用正好和它们在 $\lambda y.\, x^3+y^2$ 中的作用倒了一个个儿。x 成了约束变量，而 y 则是自由变量。如果把前面讲过的也合在一起，则可以看到，同是一个表达式 x^3+y^2，其意义由于加了不同的 λ 首部（$\lambda xy.\,,\lambda yx.\,,\lambda x.\,,\lambda y.\,$）而不一样了。这种在表达式前加上 λ 首部的操作，叫做抽象，是 λ 表示法中一个基本的概念。

实际上，我们刚才还涉及了另外一些基本概念。比如，“应用”就是 λ 演算中（乃至一切函数型系统中）最重要的概念之一，它表示把一个函数应用于一组具体的变元值（如像上例中的 $x=1, y=2$），当然，变元值本身也可以是函数，在下面将要考查的形式系统中，“应用”是唯一的一种操作，一切其他操作都以函数应用的形式出现。

此外，“置换”是又一个重要的基本概念，它不但是实现“应用”的手段，而且是整个 λ 演算的基石。两个 λ 表达式是否相等就是靠“置换”来证明的，也正是依靠它，Church 证明了 λ 演算具有部分递归函数的计算能力，也就是相当于图灵机。

定义 1.1.1（形式的 λ 演算系统） 一个形式的 λ 演算系统由下列几部分组成：

（1）由变量组成的无穷集合 V；

（2）由常量组成的（一般为有穷的）集合 C。集合 V 和集合 C 中的元素统称原子；

（3）由四个特殊符号 λ,.,(,) 组合的集合 S。它们在形式系统中通常称为组合符。集合 $\{V\cup C\cup S\}^+$ 中的任一元素皆能唯一地分解为 V，C 和 S 三个集合中元素的并列（注意，这隐含了 V，C 和 S 三个集合两两相交均为空集）；

（4）以 V，C 和 S 三集合中元素为语法元的一组语法公式；

（5）一组转换规则。 ∎

定义 1.1.2 我们一般以 $x,y,z,\cdots$ 表示 V 中元素，以 $a,b,c,\cdots$ 表示 C 中元素。我们所讨论的形式系统的语法公式是

〈λ表达式〉::=λ〈变量〉.〈λ表达式〉|〈表达式〉

〈表达式〉::=〈应用〉|〈原子〉

〈应用〉::=〈表达式〉〈原子〉

〈原子〉::=〈变量〉|〈常量〉|(〈λ表达式〉) ∎

例 1.1.1 $x, a, (y), xy, (abc\ xyz), x(ab)(yz)c, \lambda x.\, y, \lambda x.\, x, \lambda x.\, \lambda x.\, xx, yx(\lambda x.\, z), (\lambda x.\, y)(\lambda y.\, z)(\lambda z.\, x)$ 等都是合乎定义的 λ 表达式。 ∎

这个语法公式表明，λ 演算中的基本运算是“应用”，每个表达式应用于紧靠着它后边的原子，应用方式取左结合，即

$$abc = (a(b))c$$

这里没有结合律，否则就要出二义性了。并且，本系统的“应用”只适用于单变元。

需要解释一下约束变量的作用域。我们设想把 λ 表达式写成 $\lambda x.(e)$ 的形式，则约束变量 x 的作用域是从 e 的左括号起，到与它配对的右括号止。

定义 1.1.3　令 e_1 和 e_2 皆为表达式，若至少有下列条件之一成立，则称 e_1 在 e_2 中出现：

(1) $e_1 \equiv e_2$；

(2) $e_2 \equiv e_3 e_4$，e_3 为表达式，e_4 为原子，e_1 在 e_3 或 e_4 中出现；

(3) $e_2 \equiv (e_3)$，e_3 为 λ 表达式，e_1 在 e_3 中出现；

(4) $e_2 \equiv \lambda y.e_3$，y 是任一变量，e_3 是 λ 表达式，e_1 在 e_3 中出现。 ∎

例 1.1.2　变量 x 在 x，xy，$\lambda y.x$，$\lambda x.x$，$(\lambda x.x)(\lambda y.x)$ 等 λ 表达式中出现，但不在 $\lambda y.y$，$\lambda x.y$ 中出现。 ∎

对于前面已经提到过的自由变量和约束变量，我们也需要更确切的定义。

定义 1.1.4　令 x 为变量，e_2 为表达式，称 x 在 e_2 中自由出现，若

(1) x 在 e_2 中出现；

(2) 当 x 是按定义 1.1.3(4)的规定在 e_2 中出现时，x 应在表达式 e_3 中自由出现，且 $x \neq y$。 ∎

定义 1.1.5　若变量 x 在表达式 e 中出现，但这个出现不是自由出现，则称 x 在 e 中约束出现。 ∎

请读者注意，我们强调了“这个”两字。因为，同一个变量 x 可在同一个表达式 e 中同时有自由出现和约束出现。例如，x 在 $\lambda y.x(zt)$，$\lambda y.\lambda z.xt$，xzt 中自由出现，在 $\lambda x.x$，$\lambda x.t(\lambda y.x)$ 中约束出现，在 $(\lambda x.x)(\lambda y.x)$ 中既有自由出现，又有约束出现。

定义 1.1.6　若变量 x 在表达式 Y 中仅有自由出现或仅有约束出现，则 x 分别称为该表达式的自由变量或约束变量。 ∎

前面已经提到，置换在 λ 演算中有着特别重要的意义，它的确切定义如定义 1.1.7。

定义 1.1.7　令 x 是变量，M 和 N 是表达式，则置换 $[N/x, M]$ 表示用 N 代替 x 在 M 中的自由出现。置换所得的结果是

(1) M，若 x 不在 M 中自由出现；

(2) N，若 $M = x$；

(3) $[N/x,L][N/x,R]$,若 $M=LR$;

(4) $([N/x,P])$,若 $M=(P)$;

(5) $\lambda y.[N/x,K]$,若 $M=\lambda y.K$,$x\neq y$,y 不在 N 中自由出现;

(6) $\lambda z.[N/x,[z/y,K]]$,若 $M=\lambda y.K$,$x\neq y$,y 在 N 中自由出现,z 为任选变量,$z\neq x$,也不在 M 或 N 中出现。 ∎

在上述定义中,自由出现这个概念非常重要。只有自由出现的变量能够被置换,并且在置换过程中不允许变量的自由出现和约束出现发生混淆。例如 x 在 $\lambda y.x$ 中自由出现,$[y/x,\lambda y.x]=\lambda z.y$,$z$ 是新引进的约束变量。这是一个常函数,因为对所有的变元 a 皆有

$$(\lambda z.y)a = y \tag{1.1.12}$$

但如不把 λy 换成 λz,则结果将会是 $\lambda y.y$,这是一个恒等函数,因为对所有的 a 皆有

$$(\lambda y.y)a = a \tag{1.1.13}$$

定义 1.1.8 一个表达式(λ表达式)称为良构的(又称合式公式),如果它的所有变量或者是自由的,或者是约束的,并且满足如下的递归定义:

(1) 在表达式中自由出现的任意变量 x 是良构的;

(2) 若 M 和 N 是良构的,则 (MN) 也是良构的,在 M 或 N 中自由(或约束)出现的变量在 (MN) 中也是自由(或约束)的;

(3) 若 M 是良构的,并且包含自由变量 x,则 $(\lambda x.M)$ 也是良构的,并且 x 在其中的所有出现都是约束的。 ∎

下面,我们要进入λ演算的"演算"部分了,它以转换的形式出现,而转换的基础又是上面所说的置换。这些转换都是可逆的。

定义 1.1.9(α 转换(换名)) 令 $\lambda x.M$ 为一λ表达式,它也可以是另一表达式的子表达式。x 是任一变量,则 α 转换就是把这个λ表达式变为 $\lambda y.N$,其中

(1) y 是任一不同于 x 的变量;

(2) y 不在 M 中自由出现;

(3) 把 M 中 x 的所有自由出现代之以 y,即得 N。 ∎

其实,α 转换的概念在前面已用到过了,今后用 $A \xrightarrow[\alpha]{} B$ 表示由λ表达式 A 经过 α 转换得到λ表达式 B,例如:

$$\begin{aligned} &\lambda a.ab(\lambda a.a) \xrightarrow[\alpha]{} \lambda a.ab(\lambda b.b) \\ &\lambda a.ab(\lambda a.a) \xrightarrow[\alpha]{} \lambda c.cb(\lambda a.a) \end{aligned} \tag{1.1.14}$$

定义 1.1.10 (β 转换(应用)) 令 $\lambda x.M$ 和 N 为任意λ表达式,则把 $\lambda x.M$ 应用于 N 称为 β 转换,形式为

$$(\lambda x.M)N \xrightarrow[\beta]{} [N/x,M]$$

或

$$(\lambda x.M)(N) \xrightarrow[\beta]{} [N/x,M] \tag{1.1.15}$$

∎

例 1.1.3

$(\lambda x.xy)z \xrightarrow[\beta]{} zy$

$(\lambda x.yz)x \xrightarrow[\beta]{} yz$

$(\lambda x.(\lambda y.yx)z)t \xrightarrow[\beta]{} (\lambda y.yt)z \xrightarrow[\beta]{} zt$

$(\lambda x.(\lambda y.yx)z)t \xrightarrow[\beta]{} (\lambda x.zx)t \xrightarrow[\beta]{} zt$ ∎

有时要首先用 α 转换解决变量名冲突，如

$$(\lambda x.(\lambda y.yx)z)y \xrightarrow[\alpha]{} (\lambda x.(\lambda t.tx)z)y \xrightarrow[\beta]{} (\lambda t.ty)z \xrightarrow[\beta]{} zy \tag{1.1.16}$$

假如不用 α 转换，就会变成

$$(\lambda x.(\lambda y.yx)z)y \xrightarrow[\beta]{?} (\lambda y.yy)z \xrightarrow[\beta]{} zz$$

这是错误的结果。另外，上面的正确推导也可以直接写成

$$(\lambda x.(\lambda y.yx)z)y \xrightarrow[\beta]{} (\lambda t.ty)z \xrightarrow[\beta]{} zy$$

因为 β 转换中可以包含 α 转换。

定义 1.1.11（η 转换（外延）） 令 $\lambda x.Mx$ 为 λ 表达式，x 不在 M 中自由出现，则转换

$$\lambda x.Mx \xrightarrow[\eta]{} M \tag{1.1.17}$$

称为 η 转换。 ∎

作 η 转换的理由是：对任何一个表达式 N，均有（因为 M 中没有 x 的自由出现）

$$(\lambda x.Mx)N \xrightarrow[\beta]{} MN \tag{1.1.18}$$

就是说，由于 $\lambda x.Mx$ 和 M 应用于任意变元值均得相同结果，我们把 $\lambda x.Mx$ 和 M 在 η 转换的意义下看成是同一个函数。函数的定义有内涵和外延之分。函数的结构和特性是它的内涵，函数值和变元值的对应是它的外延。两个函数在什么情况下视为相同？是仅凭它们的外延，还是应根据它们的内涵确定，这在不同的系统中有不同的规定。η 转换在 λ 演算中引进了一种特殊的外延性，即在某些情况下，仅根据两个函数有相同的变元值→函数值对应，就视它们为同一函数。为达到更一般的外延性，我们还需要以下定义。

定义 1.1.12(ξ转换(抽象)) 令 M 和 N 为λ表达式,且有

$$M \xrightarrow[\alpha,\beta,\eta]{} N \tag{1.1.19}$$

成立,则下列形式的转换称为 ξ 转换:

$$\lambda x.M \xrightarrow[\xi]{} \lambda x.N \tag{1.1.20}$$

其中 x 为任一变量,$\xrightarrow[\alpha,\beta,\eta]{}$表示它的左边可通过适当选择 α,β,η 转换而变成右边。

■

定义 1.1.13(τ转换(展开)) 任何λ表达式中的 $M[N/x]$(即$[N/x, M]$)都可以用任何$((\lambda x.M)N)$来取代,只要$((\lambda x.M)N)$是良构的,并且 N 中的所有约束变量均不同于 x,也不同于 N 中的任何自由变量。 ■

定理 1.1.1 若(λ)表达式 M 是良构的,则在 α,β,ξ,τ 四种转换之下得到的(λ)表达式 N 仍然是良构的,且 M 和 N 有相同的自由变量。 ■

定义 1.1.14 除λ转换外,不能再做其他任何约简的λ表达式称为已化成标准型的λ表达式。 ■

定义 1.1.15 如果(λ)表达式 M 能够通过 α,β,ξ,τ 四种转换成为标准型 N,则 N 称为是 M 的标准型。 ■

为了在λ演算形式系统的基础之上构造其他形式系统,我们还需要一种手段,它可以实现语言设计者想要的其他形式转换,这就是δ转换。具有δ转换功能的λ演算称为δ演算。

定义 1.1.16 δ演算的语法如下:

(1) δ演算的基本符号是λ演算的所有基本符号加上δ符号和$\{\delta\rho_i \mid i\in I\}$符号,其中$\{\rho_i \mid i\in I\}$是一组专用符号,采用不同的专用符号形成不同的 δ 演算系统;

(2) $\{\rho_i \mid i\in I\}$也可以是空集($I=\varnothing$),此时专用符号的功能由δ符号本身执行,相当于其专用函数功能已经单一化、固定化;

(3) δ演算的良构表达式是δ($\{\delta\rho_i \mid i\in I\}$)加上λ演算的所有良构表达式加上这两者按定义 1.1.8 的所有组合;

(4) 无二义性时简称良构δ表达式为δ表达式;

(5) δ表达式 D 称为是处于δ标准型中,如果它不包含形如的$(\lambda x.P)Q$子表达式,也不包含形如 δRS 的子表达式,其中 R 和 S 不含自由变量。 ■

δ演算的语义可以通过如下的转换规则来刻画:

定义 1.1.17 δ演算转换规则(外部函数应用)的一般形式如下:

δ[函数δ表达式][变元δ表达式] →[结果δ表达式]

■

这里的函数 δ 表达式就是定义 1.1.16 中的专用符号，代表不同的函数计算。由此可见，δ 演算转换规则本质上是把复杂的函数计算用简单的形式表示出来。不同的专用符号加上不同的转换规则进一步形成不同的 δ 演算系统。

例 1.1.4　以 C 表示代表布尔值比较的专用符号，令

$$\delta \mathrm{C}MN \to \mathrm{True}, \quad \text{若 } M \text{ 可以通过 } \alpha \text{ 转换变成 } N;$$

$$\delta \mathrm{C}MN \to \mathrm{False}, \quad \text{若 } M \text{ 不能通过 } \alpha \text{ 转换变成 } N。$$

则得到一个有比较功能的 δ 演算。 ∎

由上述定义可见，δ 转换是一个转换类，真正应用时还需要具体化。下面是 Church 引进的一种具体 δ 转换，其中没有 δ 以外的专用符号。

定义 1.1.18（Church 的 δ 转换）　N 对 M 的取代分下列两种情形：

(1) $\delta MN \xrightarrow[\delta]{} \lambda xy.\, x(xy)$，若 M 可以通过 α 转换变成 N；　(1.1.21)

(2) $\delta MN \xrightarrow[\delta]{} \lambda xy.\, xy$，若 M 不能通过 α 转换变成 N。　(1.1.22)

其中 δMN 还需满足下列条件：

(1) M, N 皆不包含自由变量；

(2) M, N 皆已为 δ 标准型，且均不包含形为 $(\lambda xP)Q$ 或 δRS 的真子表达式，其中 R, S 没有自由变量。 ∎

Church 引进这种特殊的 δ 变换，是为了定义一个类似布尔代数的系统，即只有真（用 2 代表）和假（用 1 代表）两种值的演算系统。在这里，$\lambda xy.\, xy$ 表示 1，$\lambda xy.\, x(xy)$ 表示 2，这是正整数的 λ 表示法。任一正整数 n 可表为

$$\lambda xy.\, \underbrace{x(x(x\cdots(xy)\cdots))}_{n\text{个}x} \tag{1.1.23}$$

采取这种表示法的原因是：对任一原子 f，均有

$$(\lambda xy.\, xy)f = \lambda y.\, f(y)$$

$$\cdots\cdots$$

$$(\lambda xy.\, \underbrace{x(x(x\cdots(xy)\cdots))}_{n\text{个}x})f = \lambda y.\, \underbrace{f(f\cdots f}_{n\text{个}f}(y)\cdots) \tag{1.1.24}$$

即用函数的 n 重嵌套来表示正整数 n。在这个基础上，λ 演算可用来模拟一整套递归函数理论，有兴趣的读者可参看有关著作。

以上讲的这些转换，并不是每个 λ 演算系统都允许的。由于允许转换的种类多寡不同，各系统功能的强弱也不同，得到的有关定理也不一样，因此，在进行讨论时，往往要说明“在允许哪几种转换的前提下”。例如，在几何学中，允许直角坐标变换的是欧氏几何，允许仿射变换的是仿射几何，允许射影变换的是射影几何，如此等等，在我们这里，α 转换属各系统公有，无需指出，其他均需注明，如 $\beta\eta$ 演算表示允许 β 转换和 η 转换的 λ 演算，$\beta\delta$ 演算表示允许 β 转换和 δ 转换的 λ

演算，如此等等，同时允许 α,β,η,δ 四种转换的系统称为完全λ演算系统。

其次，α,β,η,δ 这四种转换，都可以在正反两个方向进行。对于正向进行的转换（如同定义中给出的那样），给一个特殊的名字叫约简。转换前的λ表达式称待约式，转换后的λ表达式称约简式。一个自然的问题是：是否存在一类标准的约简式，使得所有的λ表达式都可以在有限步之内通过约简化成这类形式。这就是标准型问题。

有例子表明，某些λ表达式是不能转换成标准型的，如

$$\begin{aligned}&(\lambda x.\,xx)(\lambda x.\,xx)\\ \xrightarrow[\beta]{}\;&(\lambda x.\,xx)(\lambda x.\,xx)\\ \xrightarrow[\beta]{}\;&\cdots\end{aligned}\tag{1.1.25}$$

即可以对它无穷无尽地实施 β 转换. 更有甚者：

$$\begin{aligned}&(\lambda x.\,xxx)(\lambda x.\,xxx)\\ \xrightarrow[\beta]{}\;&(\lambda x.\,xxx)(\lambda x.\,xxx)(\lambda x.\,xxx)\\ \xrightarrow[\beta]{}\;&\cdots\end{aligned}\tag{1.1.26}$$

即越“约简”，越膨胀。

另一种奇怪的现象是约简的结果似乎与约简的次序有关，请看

$$(\lambda x.\,y)((\lambda x.\,xx)(\lambda x.\,xx))\tag{1.1.27}$$

如果先做后面大括号的 β 转换，则仍将如前例那样无休无止地转换下去，但如先做左面的 β 转换，则立即得到标准型 y。

这就产生了一个问题，如果约简的结果真的与约简的次序有关，那么λ演算就是不确定的了。幸好，我们有：

定理 1.1.2（Church-Rosser 第一定理） 如果λ表达式 A 既能转换为λ表达式 B，又能转换为另一λ表达式 C，则一定存在一个λ表达式 D，使 B 和 C 都能约简为 D。 ∎

定理的证明我们省略了，有兴趣的读者可参见有关著作。请注意，最后的结论是 B 和 C 都能“约简为 D”，而不是“转换为 D”，因为转换是双向的，从而“转换为 D”是显然的。

这条定理保证了：如果一个λ表达式确有相应的标准型，则此标准型在允许 α 转换的意义下是唯一的。

注意，Church-Rosser 第一定理中所说的转换包括 α,β,η,δ 四种转换，这表示该定理对完全λ演算系统成立。

但是，这个定理只能解决存在性的问题，还不能解决构造性的问题，我们已