

# 情報数学 最終課題

72043913/t20391ks 澤田 開杜

## 概要

情報数学の授業で学んだラムダ計算について下記にまとめて記す。

## テーマを選んだ動機

最近のプログラミング言語では、HaskellやScalaのような関数型言語でなくとも、関数が第一級オブジェクトとなっているような言語が多い。私もこの授業を受ける前までは、λ式というと"無名関数"・"関数オブジェクト"といったような認識でしかいなかった。しかし、学んでみると普通のプログラム言語のような表現力を持っており、奥が深く、とても興味を持ったためである。

## ラムダ計算

### ラムダ記法

ラムダ式とは、いわば名前の無い関数である。例えば+の演算子がすでに定義されているとして、引数を1つ取り、それに1を加える関数を定義すると、 $add1(x) = x + 1$ のように記述するだろう。しかし、関数の名前自体は本質的ではない。そこで、この関数を $\lambda x. x + 1$ と表現することにする。"λ"という記号の直後に引数を記述し、"."の後に関数の本体を記述する。このように仮引数を指定し、式から直接関数を定義する操作をλ抽象という。このように定義したλ式を $add1(3)$ のように関数を値に適用するには、 $(\lambda x. x + 1)(3)$ のように記述する。

### カーリー化

先程は $add1$ という、引数が一つの関数をもとにラムダ式での形を考えた。ここで新しく $add(x, y) = x + y$ という2つの引数を取る関数を考え、それをλ抽象した結果を考えるが、λ式では、一般に $\lambda(x, y). x + y$ のような複数引数を受け取る形では記述しない。そのため、これをλ抽象するにはカーリー化という手法が大切になる。まずは普通の関数をカーリー化するとはどういうことかを考える。関数 $add$ は2つの数を受け取って1つの数を返すため、型は $N \times N \rightarrow N$ である。一言で説明すると、カーリー化とはこの $N \times N \rightarrow N$ のような関数を $N \rightarrow N \rightarrow N$ のような形にすることである。カーリー化した $add$ を $add'$ と表記すると、 $add'$ の例の場合、引数を1つ与えると"引数を1つ受け取り、1つの値を返す $N \rightarrow N$ 型の関数"を返す。つまり、 $add'(1)$ とすると先程の $add1$ と同じ、引数を1つ受け取り、それに1を足した結果を返す関数を作ることができる。よって、 $add$ に対して $add(1)(7)$ のように呼び出せば $1 + 7$ を計算することができる。これをλ式では $\lambda x. (\lambda y. x + y)$ と記述する。更に、λ式では曖昧性がなければカッコやλを省略することができるため、 $\lambda x y. x + y$ と記述するのが一般的である。また、引数に関数を取ったり、戻り値として関数を返したりするような関数のことを、一般に高階関数という。この高階関数はラムダ計算だけでなく、最近のプログラム言語では機能としてサポートされている物が多い。具体例は示さないが、これをうまく使えると美しいプログラムを書くことができる。

## λ式の定義とβ変換

λ式を以下のように定義する。

1. 変数  $x, y, z, x_1, x_2, y', \dots$  はλ式である
2. λ式  $M$  と変数  $x$  に対して、 $(\lambda x. M)$  はλ式である。(λ抽象, *function abstraction*)
3. λ式  $M$  と  $N$  に対して  $(M N)$  はλ式である。(関数適用, *function application*)

λ抽象については前のラムダ記法の部分で記したが、関数適用については深く記していない。しかし、難しいことは全くなく、カリ－化の部分で述べた2つの数を足し算をするλ式で考えると、 $(\lambda x y. x + y)(1)(7)$  とするとラムダ式に1を適用し、後に7を適用している状態になる。この計算の過程を詳しく確認すると、まず  $(\lambda x y. x + y)(1)(7)$  の  $x$  が1に置き換わり、 $(\lambda y. 1 + y)(7)$  となり、次に  $y$  が7に置き換わって  $1 + 7$  となる。そして、この仮引数に実引数に置き換えるという手順のことをβ変換といい、以降 " $\xrightarrow{\beta}$ " と記す。また、カリ－化の部分でも記したが、ラムダ式には省略表現が存在する。

1.  $\lambda x_1 x_2 \dots x_n. M \equiv (\lambda x_1. (\lambda x_2. (\dots (\lambda x_n. M) \dots)))$
2.  $M_1 M_2 M_3 \dots M_n \equiv ((\dots ((M_1 M_2) M_3) \dots) M_n)$

## 束縛変数と自由変数

λ式の中で変数  $x$  が  $\lambda x. \dots x \dots$  のように現れた時、「変数  $x$  は  $\lambda x$  によって束縛されている」といい、これを束縛変数という。逆に、束縛されていない変数のことを自由変数という。また、自由変数を持たないλ式のことを閉式という。λ式  $M$  中の自由変数の全体を  $FV(M)$  で表すとき、閉式であるときは  $FV(M) = \emptyset$  である。

## α変換と代入

束縛された変数を置き換えても意味は変わらない。例えば、 $\lambda x. x$  と  $\lambda y. y$  は同じである。このように束縛された変数を置き換える操作のことをα変換といい、以降、" $\xrightarrow{\alpha}$ " とする。α変換を使うことによって、自由変数と束縛変数が異なるようにすることができる。また、式  $M$  の自由変数  $x$  を式  $N$  に置き換えることを代入といい、以降 " $M[x := N]$ " とする。具体例としては、 $((\lambda x. y x) y)[y := (\lambda z. z)] \equiv (\lambda x. (\lambda z. z) x)(\lambda z. z)$  となる。ただし、変数の束縛関係を変えてはいけないため、 $((\lambda x. y x) y)[y := x] \neq (\lambda x. x x) x$  である。このような場合のときは、 $(\lambda x. y x) y \xrightarrow{\alpha} (\lambda z. y z) y$  のように、α変換してかぶっている変数名を変えるなどする必要がある。