# 置換と $\alpha$ 同値の基本

## 1 導入

この講義の核心は、「$\beta$ 簡約の実体は置換であり、置換は自由変数を誤って束縛しないように行わなければならない」という点にある。この「誤った束縛」を防ぐ仕組が捕縛回避置換 (capture-avoiding substitution) であり、名前の付け替えが計算の意味を変えないことの保証が $\alpha$ 同値である。

## 2 中心課題

$\lambda x.(\lambda y.x)$ に $y$ を代入しようとすると何が起きるか。なぜ単純な文字列置換では不十分なのか。

## 3 用語

- 置換 (substitution): 項 $t$ の中の変数 $x$ の自由な出現を項 $s$ で置き換える操作。$t[x := s]$ と書く
- 変数捕縛: 置換によって自由変数が意図せず束縛されてしまう現象
- 捕縛回避置換 (capture-avoiding substitution): 変数捕縛を防ぐために、必要に応じて束縛変数を改名してから行う置換
- $\alpha$ 変換 ($\alpha$-conversion): $\lambda x.t$ の $x$ を新しい名前 $y$ に付け替えて $\lambda y.t[x := y]$ にする操作（ただし $y \notin \mathrm{FV}(t)$）
- $\alpha$ 同値 ($\alpha$-equivalence): $\alpha$ 変換で行き来できる項どうしの同値関係

## 4 方針

まず単純な文字列置換がなぜ失敗するかを見る。つぎに捕縛回避置換の定義を BNF の場合分けで書く。最後に $\alpha$ 同値を「名前の違いを無視する 商集合 上の等しさ」として捉える。

## 5 直感的な説明

「$x$ を $s$ に置換する」を「文書の中の単語を検索・置換する」と同じだと思うと問題が起きる。

たとえば $(\lambda y.x)[x := y]$ を「$x$ を $y$ に書き換える」と機械的に行うと $\lambda y.y$ になる。しかし元の意味は「$x$ という外側の値を受け取って、$y$ 無関係に $x$ を返す関数」だった。置換後の $\lambda y.y$ は「$y$ を受け取って $y$ を返す関数」になってしまい、意味が変わっている。

解決策は「置換しようとしている自由変数 $y$ が束縛される 状況 に入ろうとしたら、束縛変数を別名に改名する」ことである。

Display

(問題)

(λy. x)[x := y]

→ λy. y ← y が捕縛された（意味が変わった）
(正しい手順：まず y を z に改名)

λy. x →(α変換)→ λz. x

(λz. x)[x := y] → λz. y ← y は自由のまま保たれた

# 6 厳密な説明

## 6.1 1. 捕縛回避置換の定義

$t[x := s]$ を次の場合分けで定義する。

$$x[x := s] = s$$

$$y[x := s] = y \quad (y \neq x)$$

$$(t_1\ t_2)[x := s] = (t_1[x := s])\ (t_2[x := s])$$

λ 抽象 の場合が重要である。

$$(\lambda x.t)[x := s] = \lambda x.t \quad (x \text{ は [この中/このなか] で [束縛/そくばく] されているので [置換/ちかん] しない})$$

$$(\lambda y.t)[x := s] = \lambda y.(t[x := s]) \quad (y \neq x \text{ かつ } y \notin \mathrm{FV}(s))$$

$$(\lambda y.t)[x := s] = \lambda z.(t[y := z][x := s]) \quad (y \neq x \text{ かつ } y \in \mathrm{FV}(s)\text{、} z \text{ は [新/あたら] しい [名前/なまえ]})$$

最後の行が捕縛回避のための改名である。

## 6.2 2. α同値

α変換 $\lambda x.t \underset{\alpha}{\to} \lambda y.t[x := y]$（ただし $y \notin \mathrm{FV}(t)$）によって行き来できる項は α同値である。

$$\lambda x.x \underset{\alpha}{=} \lambda y.y \underset{\alpha}{=} \lambda z.z$$

記号としての名前は異なるが、「受け取ったものをそのまま返す関数」という意味は同じである。

## 6.3 3. β簡約との関係

β簡約の定義（次の講義）は：

$$(\lambda x.t)\ s \underset{\beta}{\to} t[x := s]$$

右辺の置換は必ず捕縛回避で行われる。α同値を認めることで、$t[x := s]$ は代表元の取り方に依存しない（名前の選び方に依存しない）ことが保証される。

## 7 最小の具体例

### 7.1 例 1: 捕縛回避が不要なケース

$(\lambda x.\lambda y.x)[x := a]$
$a$ は変数名だけとして $a \notin \{y\}$ と仮定すると：
$= \lambda x.\lambda y.x$（外側の $\lambda x$ が $x$ を束縛しているので置換しない）

### 7.2 例 2: 捕縛回避が必要なケース

$(\lambda y.x)[x := y] \to y \in \mathrm{FV}(y)$ なので改名が必要
$z$ を新しい名前として：

$= \lambda z.(x[y := z])[x := y] = \lambda z.y$

## 8 別の見方

$\alpha$ 同値を「商」として見ると、$\lambda$ 項の集合を $\alpha$ 同値類で割った商集合の上で $\beta$ 簡約を定義することになる。あるいはde Bruijn 指標（変数を名前でなく「何番目の $\lambda$ に束縛されているか」という番号で表す方法）
de Bruijn index
を使えば、$\alpha$ 同値な項は同一の表現になり改名が不要になる。

## 9 見分け方

- $t[x := s]$ を計算するとき、$s$ に自由変数が含まれるかを先に $\mathrm{FV}(s)$ で確認する
- $\lambda y....$ に入るとき、$y \in \mathrm{FV}(s)$ なら改名してから置換する
- $\lambda x.x \underset{\alpha}{=} \lambda y.y$ のように束縛変数の付け替えは $\alpha$ 同値の範囲内

## 10 一言でいうと

置換は「$x$ を $s$ に書き換える」だけでなく「自由変数を誤って束縛しないように慎重に行う操作」であり、$\alpha$ 同値は「変数名は本質でない」ことの形式化である。

## 11 関連リンク

→ 講義 束縛と自由変数の基本 lecture information programming-languages
https://study.bem130.com/lecture/information/programming-languages/foundation/束縛と自由変数の基本-講義/

→ 講義 ラムダ計算の基本 lecture information programming-languages
https://study.bem130.com/lecture/information/programming-languages/lambda-calculus/ラムダ計算の基本-講義/