計算モデル特論

関数型計算モデル

佐藤一郎
E-mail: ichiro@nii.ac.jp

型なしラムダ計算

1. はじめに
2. 関数と型
3. ラムダ記法
4. ラムダ計算
5. 変換例
6. チャーチロッサ性
7. 正規形の求め方
8. ラムダ計算の計算能力

ラムダ計算(Lambda Calculus)

1930年代に数学基礎論の研究から生まれた (A. Church)

- 一般に数学で扱われる**関数概念**に伴う計算的要素を抽出して作られる計算体系
- 関数型プログラミング言語の基礎理論
  - Lisp、Scheme、ML、Haskellなど
- プログラムの意味論、型理論に関係する

関数と型

関数:与えられた引数に適用して値を得るための操作

f(x): 関数 f を x に適用して得られた値

- x のとりうる値の領域A (定義域と呼ぶ)
- f(x) のとりうる領域B (値域と呼ぶ)
- このような関数の集合は“A→B”と書き、**f の型**と呼ぶ

例: square(x) = x * x

x のとりうる領域は自然数Nのとき、square の型は N→N である。
これを、square∈N→Nとかく

疑問

N→(N×N) はどんな関数か？

1つの自然数を与えると**関数が得られる関数**

$f_x(y)=x\cdot y$で、xをある値kに決めれば、$f_k(y)=k\cdot y$で、N→N の型を持つ関数となる

(N×N)→(N×N) はどんな関数か？

関数を引数として、関数が得られる関数

twice f(x)＝f(f(x)) なる関数 twice を考える

f(x)のところにsquareを引数として与えれば、

twice square(x)＝square(square(x))

として関数を作り出す。

高階関数(higher-order function)

関数を引数とする関数や関数を結果とする関数

例：

twice (f (x)) = f(f(x))

関数を引数とする関数

関数の関数などを取り扱っていくうえで、関数を値と同様に取り扱えると便利

例：関数の関数 twice f(x)＝f(f(x)) を考える

f(x)のところにsquareを引数として与えれば、

twice square(x)＝square(square(x))＝x・x・x・x

従って、値域はNの型を持つ。

f(x)のところに$f_x(y)=x\cdot y$を引数として与えれば、

twice $f_x(y)=f_x f_x(y)=(x\cdot y)\cdot y$

従って、値域はN×Nの型を持つ。

・・・・

ラムダ記法の必要性

関数として計算を扱うため、余計なものは取り除く

例：f(x) = x * x とするとき、f(x)とは

- xを変数とする関数fを表すのか、それとも
- 関数fのxにおける値を表しているのかが不明確

関数自身に名前を付けずに一つのモノ(first class object)として扱う

λx.(x*x)

ここで λx とはxが(x*x)の引数であることを示す

ラムダ記法の例

例：f(x) = x^2+2y+1のラムダ記法

$\lambda x.(x^2+2y+1)$　関数(x^2+2y+1)の引数はxであり、yは固定値と扱う

c.f.

$\lambda y.(x^2+2y+1)$　関数(x^2+2y+1)の引数はyであり、xは固定値と扱う

$\lambda x.(\lambda y.(x^2+2y+1))$ 関数(x^2+2y+1)の引数はxとyであること

ラムダ抽象(Lambda Abstraction)

式Mの中の固定値を表す名前を変数にすること

$\lambda x.M$

Mはxを変数とする関数となる

ラムダ抽象の逆操作

- ラムダ適用、
- 部分計算
- 定数畳み込み

ラムダ適用(Lambda Application)

関数M中の変数 x に値（または関数） d を代入すること

$((\lambda x.M)d)$

例：

$((\lambda x.(x^2+2y+1))3)$ → 3^2+2y+1

$((\lambda y.(x^2+2y+1))4)$ → $x^2+2\cdot 4+1$

$((\lambda x.(\lambda y.(x2+2y+1))4)3)$ →$3^2+2\cdot 4+1$

関数の自己適用

関数twiceのラムダ記法

twice＝λf.（λx. f (f x)）

関数twiceに関数gを引数として適用すると、

twice g n＝(λf.（λx. f (f x)）g) n

＝(λx. g (g x)) n ＝ g (g n)

gをtwiceに置き換えてみる

twice twice g n ＝ ((twice twice) g) n

＝ (twice (twice g)) n＝ (twice g) ((twice g) n)

＝ (g (g ((twice g) n))) ＝g (g (g (g n)))

ラムダ式

BNF文法による定義

$$M \ ::= \ x \ | \ \underbrace{(\lambda x.M)}_{\text{ラムダ抽象}} \ | \ \underbrace{(M_1 \ M_2)}_{\text{ラムダ適用}}$$

ラムダ計算とは規則に従って、ラムダ式を順次変形していくこと

ラムダ式

ラムダ式の定義

(1) 変数x，y，z. . . ，定数1，2，3，. . . はラムダ式

(2) Mがラムダ式、xが変数なら、λx. Mもラムダ式

(ラムダ抽象)

(3) M，Nがラムダ式なら、MNもラムダ式

(関数適用)

表記

$\lambda x_1 x_2 \cdots x_n . M = \lambda x_1 . (\lambda x_2 . (\cdots (\lambda x_n . M) \cdots))$

$M_1 M_2 M_3 \cdots M_n = (\cdots((M_1 M_2) M_3) \cdots) M_n$

ラムダ式の例

- x
- $(\lambda x.x)$
- $(\lambda x.y)$
- $(\lambda x.(\lambda y.x))$
- $((\lambda x.(x\ x))\ y)$
- $((\lambda x.(x\ x))\ (\lambda y.y))$

ラムダ式の省略形

省略の規則：

- 一番外側の括弧は外してよい
- $(\lambda x_1.(\lambda x_2...(\lambda x_n.M)...))$は$\lambda x_1 x_2 ... x_n.M$と書いてよい
- $(...(M_1\ M_2)...M_n)$は$M_1\ M_2\ ...\ M_n$と書いてよい

例題：

$(\lambda x.(\lambda y.((xy)(zu))))$

$= \lambda x.(\lambda y.((xy)(zu)))$

$= \lambda x \lambda y.((xy)(zu))$

$= \lambda x \lambda y.xy(zu)$

自由変数

ラムダ式Mに含まれる自由変数の集合FV(M)

$$FV(x) = \{x\}$$

$$FV(M_1\ M_2) = FV(M_1) \cup FV(M_2)$$

$$FV(\lambda x.M) = FV(M) - \{x\}$$

自由変数でない変数を束縛変数

変数

束縛変数

ラムダ式のxを変数としてラムダ抽象

自由変数

式に含まれる変数と抽象化の対象が結びついているかどうか

x (λxy. xyz) xy

①　②③ ④⑤⑥ ⑦⑧

このとき、自由変数は①、⑥、⑦、⑧、束縛変数は、②、③、④、⑤

- ラムダ式M_1, M_2, ･･･M_nで、それらの束縛変数と自由変数が重ならないように置き換えができる。
- 重なりがない状態＝「変数条件を満たす」という

変換規則

α-規則 (束縛変数の名前を置換)

$(\lambda x.M) = (\lambda y.[y/x]M)$

ただし、yがMの自由変数ではないとする

β-規則 (ラムダ式における計算)

$((\lambda x.M)\ N) \rightarrow [N/x]M$

ここで、[b/a]MとはM中の自由変数aをbで置き換える

- β変換によるラムダ式の書き換えをリダクションという。
- リダクションが可能な部分をリデックスと呼ぶ。
- リデックスが含まれていないとき、そのラムダ式は正規形であるという

α変換の例

- $(\lambda x.x) = (\lambda y.y)$
- $((\lambda x.x)\ (\lambda x.xy)) = ((\lambda y.y)\ (\lambda z.zw))$
- $(\lambda x.\ (\lambda x.x)) = (\lambda y.\ (\lambda z.z))$

リダクションの練習問題

(1) (λxy. y) 3 2
(2) (λxy. xy) (λw. w･w) 9
(3) (λxy. x) (λx. xx) (λz. z)
(4) (λxy. y) (λx. xx) (λz. z)
(5) (λx. x (λxy. x)) (λx. x)
(6) (λx. x (λxy. x)) (λx. x (λxy. y)) (λx. x)

リダクションの練習問題 (解答)

(1) (λxy. y) 3 2 → (λy. y) 2 → 2

(2) (λxy. xy) (λw. w･w) 9
→(λy. (λw. w･w) y) 9→(λy. y･y) 9→9･9

(3) (λxy. x) (λx. xx) (λz. z)
→(λy. (λx. xx)) (λz. z)→λx. xx

(4) (λxy. y) (λx. xx) (λz. z)
→(λy. y) (λz. z)→λz. z

リダクションの練習問題 (解答)

(5) (λx. x (λxy. x)) (λx. x)
→(λx. x) (λxy. x)→(λxy. x)

(6) (λx. x (λxy. x)) (λx. x (λxy. y)) (λx. x)
→(λx. x (λxy. y)) (λxy. x) (λx. x)
→(λxy. x) (λxy. y) (λx. x)
→(λy'. (λxy. y)) (λx. x)
→(λxy. y)

変換 (リダクション) の例

(1) 自由変数と束縛変数が衝突する場合は、書き換えておく
(λxy. x) yz → (λy. y) z → z (誤り)

(λxy. x) yz → (λxy'. x) yz → (λy'. y) z→y
(2) リダクションを行うと複雑になってしまう例
(λx. xxx) (λx. xxx)
→(λx. xxx) (λx. xxx) (λx. xxx)
→(λx. xxx) (λx. xxx) (λx. xxx) (λx. xxx)
→････

変換（リダクション）の例（続き）

（3）自分に戻ってしまうリダクション

$(\lambda x.\ xx)(\lambda x.\ xx) \rightarrow (\lambda x.\ xx)(\lambda x.\ xx)$

（4）異なる部分から始めて同じ結果が出るリダクション

$I \equiv \lambda x.\ x$とする

$I(I\ x)$は二つのリデックス$I(I\ x)$と$I\ x$を持つ

$I(I\ x) \rightarrow I\ x \rightarrow x$

$I(I\ x) \rightarrow I\ x \rightarrow x$

正規形の求め方

正規形を計算する戦略

2つのリデックスがあるとき、どちらを選ぶか？

$M \equiv (\lambda x.\ y)\underbrace{\underbrace{((\lambda x.\ xx)(\lambda x.\ xx))}_{①}}_{②}$

①では、M→M→・・・の無限のリダクションが続く

②では、M→yとなり、1回で完了

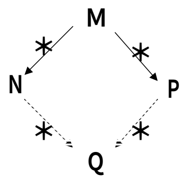
リダクション戦略

（1）ラムダ式がM正規形を持つならば、最も左側のリデックスを常にリダクションすることで、必ず正規形が得られる。

（2）最も左側のリデックスとは、最も外側のリデックスのうちで、最も左側のものであること

チャーチ・ロッサ性

- ラムダ式にリデックスが複数あるとき、そのどれに注目するかにより、何通りかのリダクションの可能性がある。
- 場合によっては、正規形にならないリダクションもある。
- 複数のリダクション列ができるとき、その結果得られる正規形は途中のリダクションの道筋によらず一意に決まる。



$M \xrightarrow{*} N$、$M \xrightarrow{*} P$のとき、

（0回以上のリダクションでMからNに到達する意味）

（MからP）

適当なリダクションで、Qに合流できる。

チャーチ・ロッサ性の利点

- リダクションの順序に気を使う必要がない
- どんな方法でリダクションを行っても、得られた結果（正規形）が唯一であることが保証される

（通常の並列計算や、非決定的な計算では、計算の順序を保つため、同期が必要となる）

ラムダ計算の計算能力

ラムダ計算モデルによるプログラム

各自然数kを正規形のラムダ式「k」で表す。　g：$N^n \rightarrow N$に対して、

$$g(k_1, k_2, \cdots, k_n) = k \Leftrightarrow G「k_1」「k_2」\cdots「k_n」\rightarrow「k」$$

を満たすラムダ式Gを、関数gを計算するためのプログラムとみなす。
「k_1」,「k_2」, ・・・,「k_n」はこのプログラムの入力とみなす。

- このプログラムGを入力「k_1」,「k_2」, ・・・,「k_n」に対して β変換を次々と行うことを、計算過程としてとらえる。
- 変換が終結して「k」が得られたとき、プログラムの計算結果がkであると考える。変換が終結しない場合、プログラムの計算結果は未定義となる。

コード化：論理値

論理値の真（true）と偽（false）は次のようなラムダ式に対応

「true」≡ λxy. x　（Tともかく）
「false」≡ λxy. y　（Fともかく）

条件判定式に対応するラムダ式（AとBはプログラム分に相当）

「cond」≡ λb λA. λB. bAB

例：任意のAとBに対して

Cond true A B →・・・ → A
Cond false A B →・・・ → B

コード化：自然数

自然数nに対応するλ式

- 「0」と「次の自然数」という概念からコード化

  「0」≡ λf. λz. z
  「1」≡ λf. λz. f z
  「2」≡ λf. λz. f (f z)
  ・・・
  「N」 ≡ λf. λz. $f^n z$

- 次の自然数を求める関数のコード化

  Succ≡ λn. λf. λz. f(n f z)

コード化：自然数演算

自然数演算に対応するλ式

- 足し算のコード化

  Add≡ λm. λn. m Succ n

- かけ算のコード化

  Mul≡ λm. λn. m (Add n)「0」

- ゼロ判定関数のコード化

  IsZero≡ λn. n (λn.「false」)「true」

不動点オペレータ

例：

$$Y \equiv \lambda f.(\lambda x.f(x\ x))(\lambda x.f(x\ x))$$

Yを任意のラムダ式Fに適用すると

$$YF \rightarrow (\lambda x.F(x\ x))(\lambda x.F(x\ x))$$

$$\rightarrow F((\lambda x.F(x\ x))(\lambda x.F(x\ x)))$$

$$\leftarrow F(YF)$$

β変換を等式とみなすと、

$F(YF) = YF$ ・・・・ラムダ式Fの不動点はYFとなる

（関数fの不動点とは、$f(x) = x$となるxのこと）