計算の理論

総合科目F(数理・情報)
火曜5限(17:05-18:35)

後半担当：小林　直樹
理学部情報科学科
大学院情報理工学系研究科コンピュータ科学専攻

後半部分のWebページ：
https://www-kb.is.s.u-tokyo.ac.jp/~koba/class/ComputationTheory/
今後の資料、課題はITC-LMSに掲載

講義計画

今井担当分

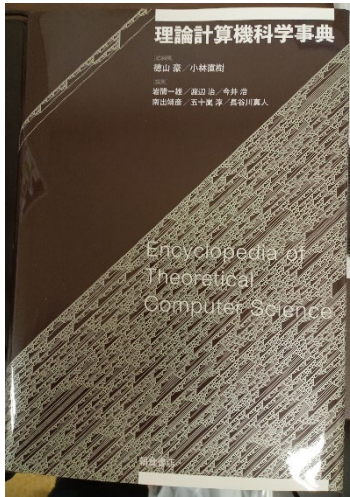
1. 計算とは(アルゴリズム、算術)
2. チューリング・チャーチの
   計算仮説
3. 決定不能な問題
4. 計算困難さ(NP完全問題)
5. 難しい問題をなんとかして解く
6. いろいろな計算パラダイム
(確率的計算など)

小林担当分

7. 計算のモデル1(ラムダ計算)
8. 計算のモデル2(ラムダ計算)
9. 型付きラムダ計算
10. 計算と論理1
   (カリー・ハワード同型)
11. 計算と論理2
   (定理証明支援器を使ってみる)
12. 計算と論理3
   (論理プログラミング:
   時間があれば)

理論計算機科学A分野
アルゴリズムと計算量
(「理論計算機科学事典」
の第一部)

理論計算機科学B分野
計算モデル、プログラム意味論、
計算可能性
(「理論計算機科学事典」
の第二部)



ラムダ計算とは？

- 「関数」だけからなる計算モデル

  $M ::= x \mid \lambda x.M \mid M_1 M_2$

- チューリング機械と同等の表現力
  （計算可能な関数をすべて表現可能。
  cf. チャーチ–チューリングの提唱）
- 近代的プログラミング言語（特に関数型プログラミング言語）の基礎
  - チューリング機械 ≈ 計算機
  - ラムダ計算 ≈ 高級プログラミング言語
- 「型」の概念を加えると、論理学とも密接な対応
  （cf. カリー–ハワード同型対応）

ウォーミングアップ：関数型言語に触れてみよう

- 代表的な関数型プログラミング言語
  - ML (OCaml)
    - https://ocaml.org/
  - Haskell
    - https://www.haskell.org/
  - Scheme
    - https://www.schemers.org/

プログラム例

```
let f x = x+1;;  (* 関数f(x)=x+1 を定義 *)
val f = <fun>  (* 青字はシステムの出力*)
f(3);;    (* f(3)を評価 *)
4
f(f(3));;
5
let rec sum n = if n=0 then 0 else n+sum(n-1)
   (* 1からnまでの和を計算する関数を再帰的に定義 *)
val sum = <fun>
sum(10);;
55
```

プログラム例

（高階関数その1:関数を返す関数）

```
let f x y = x+y;;  (* 2引数関数 *)
val f = <fun>
f 1 2;;
3
let g = f 1;;        (* yをもらって1+yを返す関数 *)
val g = <fun>
g 2;;
3
(* 結局、fはxをもらって、「yをもらってx+yを返す関数」を返す
1引数関数とみることもできる *)
```

プログラム例
（高階関数その2：関数を引数にとる関数）

```
let twice f x = f(f x);;
   (* 関数fをもらってそれを2回適用する関数*)
let succ x = x+1;;
twice succ 3;;

let rec sum n = if n=0 then 0 else n+sum(n-1)
let rec sumsq n = if n=0 then 0 else (n*n) + sumsq(n-1)
let rec sumcube n = if n=0 then 0 else (n*n*n)+sumcube(n-1)
(* 同じような処理を何度も書くのは煩わしい。
  高階関数を用いて一般化すると...*)
let rec gensum f n = if n=0 then 0 else f(n) + gensum f (n-1);;
let sum = let id x = x in gensum id
let sumsq = let sq x = x*x in gensum sq
let sumcube = let cube x = x*x*x in gensum cube
```

プログラム例（名前なし関数）

fun x-> e:
xを引数として受け取り、eの値を返す関数

```
let rec gensum f n = if n=0 then 0 else f(n) + gensum f n;;
let sum = gensum (fun x -> x)
let sumsq = gensum (fun x -> x*x)
let sumcube = gensum (fun x -> x*x*x)
```

xをもらってx*x*xを返す関数

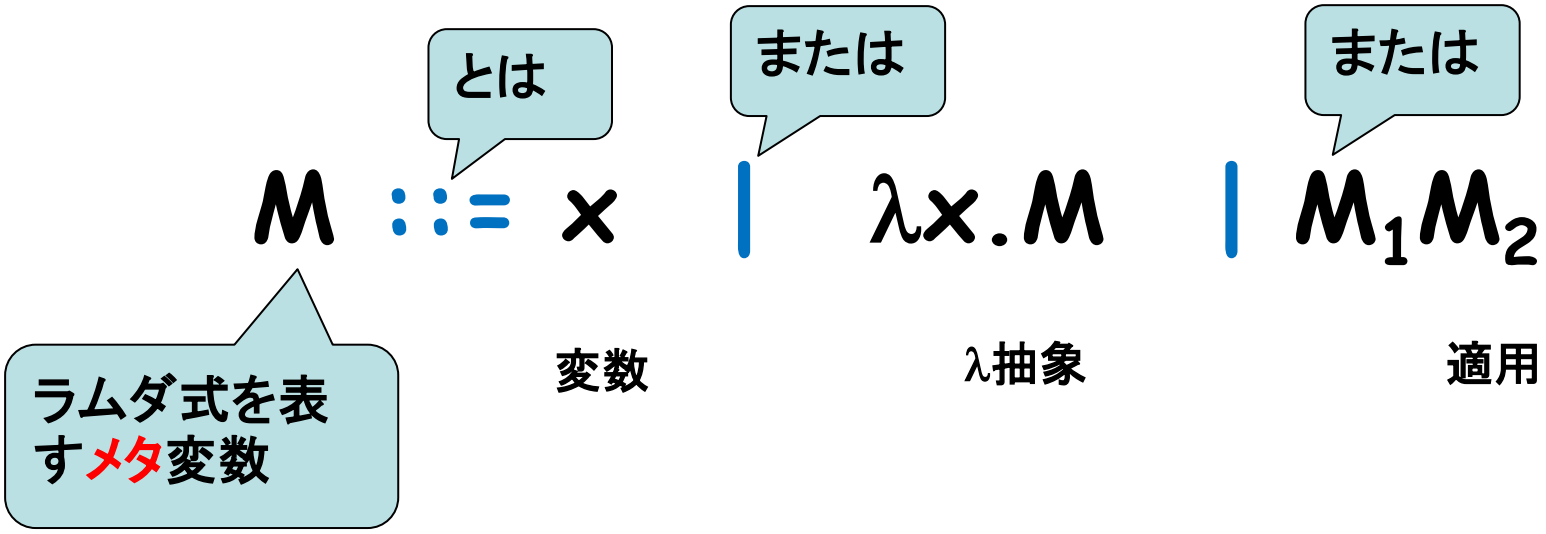
ラムダ計算とは？(再掲)

- 「関数」だけからなる計算モデル

  $M ::= x \mid \lambda x.M \mid M_1 M_2$

- チューリング機械と同等の表現力
  (計算可能な関数をすべて表現可能。
  cf. チャーチ-チューリングの提唱)
- 近代的プログラミング言語(特に関数型プログラミング言語)の基礎
  - チューリング機械 ≈ 計算機
  - ラムダ計算 ≈ 高級プログラミング言語
- 「型」の概念を加えると、論理学とも密接な対応
  (cf. カリー-ハワード同型対応)

ラムダ式の構文



上の定義は
「ラムダ式とは、変数または、λx.Mという形の式（でMの部分はラムダ式）、またはM_1M_2という形の式（でM_1, M_2の部分はラムダ式）のいずれかである」
と読む

記法上の約束

- 関数適用は左結合
  - 例：x y z は、x(y z)ではなく、(x y) zの意味
- ラムダ抽象の範囲はなるべく広くとる
  - 例：λx. x y は (λx. x) yではなく、λx. (x y)の意味

例題：以下の式に意味が変わらないように括弧をつけよ

λx.x x λx.x x λx.x

記法上の約束

- 関数適用は左結合
  - 例：x y z は、x(y z)ではなく、(x y) zの意味
- ラムダ抽象の範囲はなるべく広くとる
  - 例：λx. x y は (λx. x) yではなく、λx. (x y)の意味

例題：以下の式に意味が変わらないように括弧をつけよ

答え： λx.((x x)(λx.((x x)(λx.x))))

計算(β簡約)規則

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

Mの中のx をNにおきかえたもの
(M中のxへのNの代入: 定義は後述)

例: $(\lambda x.x+1)2 \rightarrow 2+1$

$$(\lambda x.x+1)((\lambda y.y+1)2) \rightarrow (\lambda x.x+1)(2+1) \rightarrow (2+1)+1$$

2番目の例のように、$(\lambda x.M)N$の形をしている任意の部分式を$[N/x]M$におきかえてよい。
2番目の式は次のようにも簡約できる。

$$(\lambda x.x+1)((\lambda y.y+1)2) \rightarrow (\lambda y.y+1)2 +1 \rightarrow (2+1)+1$$

どのような順序で簡約しても結果には影響がない
(後述するチャーチ-ロッサーの定理)

代入について

β簡約規則：

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

Mの中のx をNにおきかえたもの
（M中のxへのNの代入）

$(\lambda x.(\lambda x.x)(x+2))1 \rightarrow [1/x](\lambda x.x)(x+2) = (\lambda 1.1)(1+2)??$

約束1：λx のxには代入は施さない

$(\lambda x.(\lambda x.x)(x+2))1 \rightarrow [1/x](\lambda x.x)(x+2) = (\lambda x.1)(1+2)??$

約束2：内側にλx.L がある場合にはL中のxには
代入を施さない

正解：$(\lambda x.(\lambda x.x)(x+2))1 \rightarrow [1/x](\lambda x.x)(x+2) = (\lambda x.x)(1+2)$

同じ名前の変数でも、出現位置によって区別

代入について

β簡約規則:

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

Mの中のx をNにおきかえたもの
(M中のxへのNの代入)

$(\lambda y.(\underline{\lambda x.\lambda y.x})y)1\ 2 \rightarrow (\lambda y.[y/x]\lambda y.x)1\ 2 = (\lambda y.\lambda y.y)1\ 2$??

正解: $(\lambda y.(\underline{\lambda x.\lambda y.x})y)1\ 2 \rightarrow (\lambda y.[y/x]\lambda y.x)1\ 2$
$= (\lambda y.[y/x]\lambda z.x)1\ 2$
$= (\lambda y.\lambda z.y)1\ 2$

約束3: 名前が衝突する場合にはλyで「束縛」されている側の変数名を付け替え(α変換)

代入の定義

$[N/x]\ x = N$

$[N/x]\ y = y \quad (x \neq y)$

$[N/x]\ (M_1\ M_2) = ([N/x]M_1\ )\ ([N/x]M_2\ )$

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

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

（$x \neq y$かつyがNに自由に出現*しない場合）

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

（$x \neq y$かつyがNに自由に出現する場合。

zは新しい変数名）

- 「xがMに自由に出現する」とは、
  λxの内側でない位置にxが出現すること

代入の際の注意

- 結合関係が崩れないように、適宜括弧を挿入すること。
  - 例:

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

$$= (\lambda x.x)\ z$$

括弧を付け忘れると $\lambda x.x\ z = \lambda x.(x\ z)$ になってしまう

cf. 連立方程式 y=x+1, x=3-y でyを消去すると

$$x = 3 - (x+1)$$

括弧を付け忘れると $x = 3 - x + 1$

練習問題

- 以下の式をそれ以上簡約できない形にまで簡約せよ
  - $(\lambda x.\ x\ y)y$
  - $(\lambda x.\lambda y.x\ y)\ y$
  - $(\lambda x.x(\lambda x.x\ y))\ y$
  - $(\lambda f.\lambda x.f\ (f\ x))\ (\lambda x.x)$
  - $(\lambda f.f\ \ y\ x)\ (\lambda x.\lambda y.x)$
  - $(\lambda x.y)\ ((\lambda x.x\ x)\ (\lambda x.x\ x))$