技術者／プログラマのためのモナドと圏論 第1回

檜山 正幸 (HIYAMA Masayuki)

2009年4月23日 (木曜)
18:00開始

目次

モニャドってにゃんニャ？

Takaiさん、ありがとうございます。

今日の予定（おおよそ）

1. まえおき／まえせつ -- 15分
2. いろいろな図の描き方 -- 15分 (30分)
3. 圏の定義と実例 -- 50分 (80分)
4. 休憩 -- 30分 (110分)
5. 行列の圏で遊ぶ -- 50分 (160分)
6. コレクション・データ -- 20分 (180分) 無理なら次回に

状況により、この予定は（**ときに大幅に**）変更されるかも知れません。

全体（3回）の目標

- 圏論の観点からモナドを理解する（お茶を濁したり、ショートカットしない）。
- モナドを素材に圏論を学ぶ。

1. 圏の基本概念：対象／射／関手／自然変換を学ぶ。
2. モナドの実例を（なるべく）たくさん知る。
3. モナド法則に慣れる。
4. モナドのクライスリ圏の作り方を知る。（ここまではナントカして）
5. モナドがモノイドであることを理解する。（できれば）

もし余裕があれば

余裕があったタメシがないが：

1. テンプレート処理がモナドであると理解する。（流れによっては）
2. モナドの代数について知る。（少しは触れたいが）
3. アイレンベルク／ムーア圏の作り方を知る。（たぶん無理）
4. 複合モナドとは何かを学ぶ。（無理そう）
5. ベックの法則（分配法則）とその役割を知る。（無理だろ）

（フラストレーションになるから書くなよっ＞檜山）

今日の目標

1. 圏の形式的な定義と直感的なイメージの両方を知る。
2. いろいろなお絵描きを実習する。
3. 圏の実例を手でいじり倒して、体になじませる。
4. 横道にそれて、面白そうな話題を紹介する。

少し違った視点で見れば、世界が面白くなる。

7

モナドの意義(1)

計算(プログラミング)の観点から:

- コレクション・データ構造に統一的見方を与える。
- 関数計算を拡張・一般化できる。
  - 部分関数、非決定的関数
  - エラー、例外
  - 状態、副作用(大域変数、破壊的代入、IO、描画など)
- 文や式の構文論を整理できる(項モナドは構文論そのもの)。

8

モナドの意義(2)

圏論の観点から:

- 圏を拡張する標準的方法を与える
  - クライスリ構成
  - アイレンベルク／ムーア構成
- 随伴を理解する枠組みとなる(今回は触れない)
- 関手圏やモノイド概念の有用性の証左

9

モナドの意義(3)

好奇心の観点から:

- 切り上げもモナド
- 多くの閉包演算がモナド
- ヤジロベエと確率遷移系が関係する
- その他横道にそれてもオモロー

10

コース料理から、気まぐれお任せに変更

ちょっとお断りを:

- 系統的なコースにしようかと思ったが
- 体調不良で若干発熱、朦朧とした
- 朦朧でもやれるのは
- やっぱお絵描きだな
- 面白いネタを適当に見つくろってその場で出そう
- そうしよう、そうしよう

となりました。

11

回路とかオダンゴとか

「ラムダ計算」のとき使いましたね。前シリーズ参加した人には復習。

- $f(x, y) = 3x + y$
- $g(z) = z + 1$
- $h(x) = x^2 + x$

- ボックス図(boxes-and-wires)
- オダンゴ図
- ストリング図(string diagram)
- 計算素子からなる回路(circuit diagram)

12

多引数とタプル引数

- f : A, B → C
- f': A × B → C
- g : A × B → C
- `g : A, B → C

掛け算記号（×）はタプル型

- f'(<x, y>) = f(x, y)
- `g(x, y) = g(<x, y>)

フラットケーブル、リボン状ワイヤーを思い起こすといいよ。

13

ストリング図とアロー図

- 通常、図式（diagram）というとアロー図になる
- 最近では、球状図（globular diagram）ということが多い
- ペースティング図（pasting diagram）ともいうね

どこが球なんだよ？ -- 線と点だと球にはならないが、
ストリング図と球状（グロービュラー）図が「双対」となるのだ。
ペーストもグローブも高次圏だから今は別にいいけど。

矢印が異常なまでに多用される（オーバーロードされる）ので、
よほど注意しないと混乱する。「これ、なんて矢印？」

14

矢印で何を表すのか

コンピュータ風の用語なら

- 引数の型
- 戻り値の型
- 関数名（ラベル、目印）
- 関数合成（結合；composition）

圏論は、ものすごく強く型付け（typing）された体系。

型が完全に一致しないと結合は未定義（エラー）となる。

15

左と右：これは頭痛の種！

- (f;g)(x) = g(f(x))
- x.(f;g) = (x.f).g = x.f.g
- <x>(f;g) = <<x>f>g
- x>(f;g) = (x>f)>g = x>f>g
- $x^{fg} = (x^f)^g$

16

圏の例

「はじめての圏論 その第1歩：しりとりの圏」
（http://d.hatena.ne.jp/m-hiyama/20060821/1156120185）をベースに

- H={あ, あ, い, い, ..., ん, ー}（82文字）
- HStr 文字列、例：、"きゅーり", "なす", "びーまん", "はれまけろ", "んじゃびー", "ーょょっん"

結合の例

1. "こぶた";"たぬき" = "こぶたぬき"
2. "すいか";"からす" = "すいからす"
3. "らいおん";"んじゃびー" = "らいおんじゃびー"
4. "あ";"あか" = "あか"

17

圏の例 (続き)

構成要素

1. 対象の集合：H -- ひらがな文字全体の集合
2. 射の集合：HStr -- ひらがな文字列全体の集合（ただし空な列は除く）
3. 域domと余域cod：first, last : HStr → H（最初と最後の文字）
4. 恒等id：unit : H → HStr（1文字からなる文字列）
5. 結合（composition）：しりとり結合s;t は last(s) = first(t) のときだけ定義される

法則

1. $first(unit_x) = last(unit_x) = x$（x∈H）
2. first(s;t) = first(s)、last(s;t) = last(t)
3. (s;t);u = s;(t;u)
4. x = first(s), y = last(s) なら、$unit_x;s = s;unit_y = s$

18

圏に関連する用語と記法

独特な言葉／記号なので繰り返すと:

- 対象（object）、対象の集合、対象類
- 射（morphism）、射の集合、射類
- 域（domain）、余域（codomain）、余域は像ではないので注意
- 恒等（射）（identity morphism）
- ホムセット（homset）

- C : 圏（普通はイタリック）
- Obj(C) = |C| : Cの対象の集合
- Mor(C) = C : Cの射の集合
- C(A, B) : ホムセット

圏全体を表す名前やラベルはイタリックや太字にするのが慣例、そうでないと...

19

絵を描こう

矢印の束

20

触れる例を作る

- [0] = {}
- [1] = {1}
- [2] = {1, 2}
- [3] = {1, 2, 3}
- [4] = {1, 2, 3, 4}
- ...

このような [n] の全体をFO（finite ordinals）と呼ぶ。
FOを対象類とする圏は具体的で扱いやすい。列挙や数え上げができるのがいいね。

21

MapFO

口頭＋ホワイトボード

22

PMapFO

口頭＋ホワイトボード

23

RelFO

口頭＋ホワイトボード

24

圏の包含関係

- MapFO ⊆ PMapFO ⊆ RelFO

これは埋め込み関手の列だね（関手って何よ？）

25

MapFOの部分圏

部分圏て何よ？ -- まー、だいたい言葉通り。

- InjFO 単射（モノ射）の圏
- IsoFO 同型射の圏
- IncFO 包含写像の圏

26

IsoFO = Sym

Sym(3)くらいをいじってみよう。

27

アミダの圏

- |Amida| = Nat
- 圏Amidaの射は図形
- アミダの結果は圏Symに入るだろう
- ゴムだのひもだのを使って全射性を示す

28

ここらで関手

矢印「→」を使うが、関手だよ。関手も射だけど。

- Kata : HShiri → KShiri
- Len : HShiri → 変なNat
- Result : Amida → Sym
- Rev : HShiri → HShiri 反変
- Inv : Sym → Sym 反変
- Inv : Amida → Amida 反変
- Conv : RelFO → RelFO 反変

29

ここらで一段落

進行はどうかな？

時間はどうかな？

30

行列

- 行列は圏の例を与えるが、行列を絵算で計算すると面白い
- 線形（線型）代数は知らなくていい、忘れていていい、思い出さなくていい
- 行列は行列、線形写像じゃないよ
- 作業手順としての行列計算は思い出す

31

行列から回路図へ

口頭＋ホワイトボード

32

行列の二部グラフ

- 方向を持つ
- 係数（成分）が0の場所には、通常何も描かない
- 同じ出発点と到着点に2本以上の矢があってもよい
- そういう矢は足し算で集約できる

33

行列の積と「経路の量」の総和

- (行列の積)[j, i] = (iからjへの経路の量) の全経路に渡る総和

積はいくつの行列を掛け算してもいいよ。

34

正方行列の場合

- (行列Aのn乗)[j, i] = (iからjへの経路の量) の全経路に渡る総和
- (iからjへの経路の量) = 行列Aのグラフ上の経路（道）の量

行列の二部グラフとは別に、折り重ねたグラフを考える。正方行列のときだけ折り重ねができる。

35

グラフの2頂点を結ぶ道の本数

- 道は何本ある？

36

グラフ(ネットワーク)の可達性

- ここからあそこまで行けるのか？

37

ブール係数の行列は関係だ

- Mat_{Bool} = RelFO

38

構文図と正規表現

- (a|b)c*a? はどんな文字列を表すのか？

39

最小コストを求める

- どう行けば安くあがるのかな？

40

氷を手分けして運ぶ問題

- 氷は結局溶けるんだけど…

41

時間があれば

ここから先のスライドは朦朧状態で作ったヤツ。

42

リスト・データ構造

- リスト＝列（シーケンス）＝配列＝ストリング＝ワード
- えっ、違う？ 細かいことは気にしない
- 要素＝項目＝メンバー＝元
- えっ、違う？ 細かいことは気にしない
- 要するに、モノが並んでいる

集合Xの要素を並べたリストの全体を List<X> または List(X) と書く。

リストのリテラル表記は、(a, b, c) とか "abc" （項目が文字なら）とか。
<a, b, c>とか[a, b, c]とかもあるけどね。

```
> io:format("~p~n", ["abc"]).
> io:format("~w~n", ["abc"]).
```

43

心構え

このスライドを作っていて、あらためて認識して愕然としたのだが、

- プログラミング言語と圏論の記法はだいぶ違う
- 圏論の記法は、ものっすごく**ズボラ**

```
List.map<X, X>(id<x>) = id<List<X>>
List.map<X, Z>(comp<X, Y, Z>(g, f)) =
  comp<List<X>, List<Y>, List<Z>>(List.map<X, Y>(f),
  List.map<Y, Z>(g))

L(id) = id
L(f;g) = L(f);L(g)
```

テケトーでズボラになれ！ でないとやってられん。

44

リストの実例を作る

リストの項目となるベース（アルファベット）の集合として

1. Nat -- 0を含む自然数
2. Int -- 整数、ただしビット幅とか範囲とかは考えない
3. Alpha -- 普通の英字アルファベット（小文字にしておく）
4. Hira -- ひらがな（長音記号も含めて82文字）
5. Char -- 文字、場合に応じて適当に解釈
6. String -- 文字列、場合に応じて適当に解釈
7. NEString -- 非空（長さ1以上）の文字列

あと、これらの有限部分集合とか。

問題：
1. List<Nat> の例を5つ
2. List<Alpha> の例を5つ
3. List<List<Nat>> の例を5つ

45

リスト、セット、バッグ

ここでは、次の記法で区別する。

- リスト (a, b, c)
- セット {a, b, c}
- バッグ [a, b, c]

- (a, b) ≠ (b, a)、(a, b, b) ≠ (a, b)
- {a, b} ＝ {b, a}、{a, b, b} ＝ {a, b}
- [a, b] ＝ [b, a]、[a, b, b] ≠ [a, b]

46

リスト、セット、バッグ (2)

カンマの代わりに掛け算や足し算で書いてみると気分が変わるよ。

- (a * b) ≠ (b * a)、(a * b * b) ≠ (a * b)
- {a * b} ＝ {b * a}、{a * b * b} ＝ {a * b}
- [a * b] ＝ [b * a]、[a * b * b] ≠ [a * b]

- (a + b) ≠ (b + a)、(a + b + b) ≠ (a + b)
- {a + b} ＝ {b + a}、{a + b + b} ＝ {a + b}
- [a + b] ＝ [b + a]、[a + b + b] ≠ [a + b]

47

map関数(1)

sq(x) = x*x と定義される関数に対して、リストに働く Sq は

- Sq((2, 3, 1)) = (4, 9, 1)

のようになる。伝統的に Sq = map(sq) と書く。
mapは高階関数（関数引数、関数戻り値の関数）。

48

map関数(2)

```
map(F, []) ->
  []; %(1)
map(F, [First|Rest]) ->
  [F(First)|map(F, Rest)]. %(2)

function map(f, list) {
 if (list.length === 0) {
   return []; //(1)
 } else {
   var first = list.shift();
   return Array.prototype.concat.call([f(first)],
  map(f, list)); //(2)
 }
}
```

49

セットやバッグにもmap関数

- map(sq)((4, 2, -1, 2))
- map(sq)({4, 2, -1, 2))
- map(sq)([4, 2, -1, 2])

厳密には、List.map, Set.map, Bag.map のように区別すべき。

区別しないで map と書いたほうが楽。

List, Set, Bag をオーバーロードしてはどうか -- これが圏論の記法

50

圏論の記法

プログラミング言語風	圏論
List<X>	L(X), LX
List.map<X, Y>(f)	L(f), Lf
id<X>	$id_{,,}$ $1_{,,}$, id, 1

型構成子（総称型）とmap関数の組は、関手（functor）の例となる。

51

圏論の記法に慣れよう

1. id
2. sq : Int→Int
3. length : String→Nat
4. first : NEString→Char
5. str_double : String→String

52

連接、合併、結合（ジョイン）

- (a, b) ++ (c, c, b) = (a, b, c, c, b)
- {a, b} ∪ {c, c, b} = {a, b, c}
- [a, b] + [c, c, b] = [a, b, c, c]

++, ∪, + の使い分けに確定した習慣があるわけじゃない。

演算子じゃなくて関数形式でも、concat, union, join などの名前が恣意的に使われる。

面倒だから以下では、こういう区別に神経質にならない。テケトー。気分次第。

「細かいことは気にしない」がモットー。

53

あれ？ モナドは？

次回だね。

54