アルゴリズムとデータ構造入門
2.データによる抽象の構築
2.5 Generic Operation System

奥 乃 博
大学院情報学研究科 知能情報学専攻
http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/13/IntroAlgDs/
okuno@i.kyoto-u.ac.jp
池宮 由楽 if mod(学籍番号の下3桁, 3)＝0
坂東 宣昭 if mod(学籍番号の下3桁, 3)＝1
古川孝太郎 if mod(学籍番号の下3桁, 3)＝2

1月7日・本日のメニュー

2.5 Genetic Operation System
2.5.0 Put and Get
2.5.1 Generic Arithmetic Operations
2.5.2 Combining Data of Different Types
2.5.3 Symbolic Algebra

2.4.3 Data-Directed Programming and Additivity

型タグ(type-tag)の問題点

汎用手続き(real-part, imag-part, maginitude, angle)は，異なる表現をすべて知っておく必要がある．

例えば、複素数の新しい表現を作成したら

1. (new-rep? z) を定義
2. 各手続きに new-rep? に関係する処理を追加

```
(define (real-part z)
   (cond ((rectangular? z) … )
         ((polar? z) … )
         ((new-rep? z) … )
         (else … )))
```

⇒ 加法的(additivity)ではない。

複素数の表現法

$$z = x + iy = re^{iA}$$

■ 選択子(selectors)

```
(define (real-part z) (car z))
(define (imag-part z) (cdr z))
(define (magnitude z)
  (sqrt (+ (square (real-part z))
           (square (imag-part z)) )))
(define (angle z)
  (atan (imag-part z) (real-part z)))
```

■ 構築子(constructors)

```
(define (make-from-real-imag x y)
  (cons x y) )
(define (make-from-mag-ang r a)
  (cons (* r (cos a)) (* r (sin a))) )
```

NoStudent Left Behind

Data-Directed Programmingのポイント

NoStudent Left Behind

■ 加法的(additivity)なインタフェースとする為

■ 表を行方向に分割：type-tagでdispatch

演算 (operations) \ 型 (type)	Polar	Rectangular
real-part	real-part-polar	real-part-rectangular
imag-part	imag-part-polar	imag-part-rectangular
magnitude	magnitude-polar	magnitude-rectangular
angle	angle-polar	angle-rectangular

7

Message passing

NoStudent Left Behind

Church numeral と同じ発想

```
(define (make-from-real-imag x y)
  (define (dispatch op)
    (cond ((eq? op 'real-part) x)
          ((eq? op 'imag-part) y)
          ((eq? op 'magnitude)
           (sqrt (+ (square x)
                    (square y) )))
          ((eq? op 'angle) (atan y x))
          (else
           (error "Unknown op –
            MAKE-FROM-REAL-IMAG" op))))
  dispatch)

(define (apply-generic op arg) (arg op))
```

8

Message Passing のポイント

- 表を行方向に分割：type-tagでdispatch
- 表を列方向に分割：データオブジェクトがdispatch

演算 operations	型 (type) Polar	Rectangular
real-part	real-part-polar	real-part-rectangular
imag-part	imag-part-polar	imag-part-rectangular
magnitude	magnitude-polar	magnitude-rectangular
angle	angle-polar	angle-rectangular

9

2.5.0 put と get

NoStudent Left Behind

- Symbol の associative list に値を格納，抽出
- (put <id> <attribute> <value>)
- (get <id> <attribute>)
- 人口DBを put と get で作成．

連想リスト(a-list, associative list)
((<属性> . <値のリスト>)
 (<attribute> . <value-list>)
 ...
)

```
(put 'China 'population '(1285.0 660.5 624.5))
(put 'India 'population '(1025.1 528.5 496.6))
(put 'USA 'population '(285.9 141.0 144.9))
(put 'Indonesia 'population  '(214.8 107.8 107.1))
(put 'Brazil 'population '(172.6 85.2 87.4))
(put 'Pakistan 'population '(145.0 74.5 70.5))
(put 'Russia 'population '(144.7 67.7 77.0))
(put 'Bangradesh 'population '(140.4 72.3 68.0))
(put 'Japan 'population '(127.1 62.2 65.0))
(put 'Nigeria 'population '(116.9 59.0 58.0))
(put 'Mexico 'population '(100.4 49.6 50.7))
```

```
(get 'Japan 'population)
    ⇒ (127.1 62.2 65.0)
(get 'Taiwan 'population)
    ⇒ ()
```

```
(define population
  '((China 1285.0 660.5 624.5)
    (India 1025.1 528.5 496.6)
     ...
    (Japan    127.1 62.2 65.0)
    (Nigeria  116.9 59.0 58.0)
    (Mexico   100.4 49.6 50.7) ))
```

10

表の操作

NoStudent Left Behind

- 表に演算名・型(type)でその処理法をputで付加
- 表から演算名・型(type)で処理法をgetで検索
- (put <*op*> <*type*> <*item*>)
  表に<*op*> <*type*>で索引をつけて<*item*>を登録
- (get <*op*> <*type*>)
  表から<*op*> <*type*>の索引で検索し、あれば、
  <*item*>を抽出
- 演算に関連する情報<*item*>は，手続き(ラムダ式)
- <*type*>は，引数の型のリスト
- TUT-Scheme(tus2, tustk2)では：
  - (define put putprop)
  - (define get getprop)

11

put と get の動き

```
(put 'banana 'price 300)
(put 'banana 'color ' yellow)
(get 'banana 'price)
(put 'Kyoto 'Ja “kyouto”)
(put 'University 'Ja “daigaku”)
(get 'Kyoto 'Ja)
        -> “kyouto”
(map (lambda (x)  (get x 'Ja) )
     '(Kyoto University)  )
        -> (“kyouto” “daigaku”)
(put 'University 'Ge “Universitate”)
```

12

システムの加法性 (addtivity)

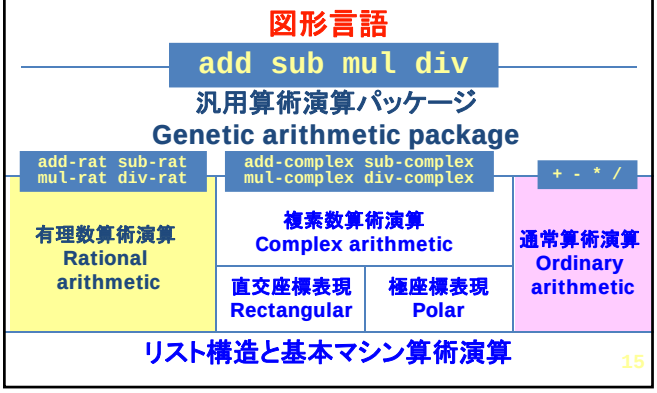
1. Data-directed Programming
2. 型による振分け(dispatching on type) は問題有
3. Additive(新処理を他の処理の変更なしに追加可).
   - (put <*op*> <*type*> <*item*>)
   - (get <*op*> <*type*>)
4. <*type*> は演算のパラメータリスト
5. 【課題】 異なるタイプの組合わせへの対応.
   - 型階層(Tower of types)
   - 強制型変換(coercion)
6. 強制型変換システムもdata-directed programmingで作成.

13

本節の目標: 統一システムの構築

14

汎用演算システムの構造

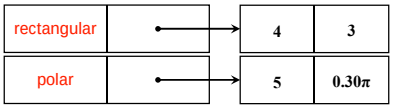


15

汎用演算システム構築のアイデア

Complexで作成した2つのサブタイプ(部分型)
rectangular, polar の構築法を思い出そう

$4+3i$
$=5(\cos 0.30\pi + i\sin 0.30\pi)$
$=5e^{0.30\pi}$




```
(define (make-from-real-imag x y)
  ((get 'make-from-real-imag 'rectangular)
   x y ))
(define (make-from-mag-ang r a)
  ((get 'make-from-mag-ang 'polar)
   r a ))
```

16

2.5.1 汎用算術演算手続き

- add sub mul div だけで算術演算を記述
- Generic operation(汎用算術演算)
- 引数のタイプ(型)により適切な演算を行う手続きを適用

```
(define (add x y)
  (apply-generic 'add x y) )
(define (sub x y)
  (apply-generic 'sub x y) )
(define (mul x y)
  (apply-generic 'mul x y) )
(define (div x y)
  (apply-generic 'div x y) )
```

1. データにはそのタイプを表現するタグ(tag)を付与
2. 演算に引数のタイプの組合せとその手続きを付与.

17

Ordinary number パッケージ

```
(define (install-scheme-number-package)
  (define (tag x)
      (attach-tag 'scheme-number x))
  (put 'add '(scheme-number scheme-number)
       (lambda (x y) (tag (+ x y))) )
  (put 'sub '(scheme-number scheme-number)
       (lambda (x y) (tag (- x y))) )
  (put 'mul '(scheme-number scheme-number)
       (lambda (x y) (tag (* x y))) )
  (put 'div '(scheme-number scheme-number)
       (lambda (x y) (tag (/ x y))) )
  (put 'make 'scheme-number
       (lambda (x) (tag x)) )
  'done )
```

演算に引数の型の組合せとその手続きを付与

汎用算術演算システムの設計と課題

1. データにはそのタイプ(型)を表現するタグ(tag)を付与.
2. 演算に引数のタイプの組合せとその手続きを付与
3. 同じタイプ同士の手続きを定義

【課題1】 異なるタイプの組合わせへの対応

- 型階層(Tower of types)
- 強制型変換(coercion)

【課題2】 システム組込みデータタイプへの対応

- 実装ではタグは付けない.

19

Tagとデータ, 演算の関係

```
(define (make-scheme-number n)
   ((get 'make 'scheme-number) n) )
(define foo (make-scheme-number 8))
```

scheme-number	8

が作成される.

- 中身を見る手続きは contents と type-tag
- 汎用演算(add sub mul div) には引数のタイプの組合せに対する手続きの対が並ぶ.

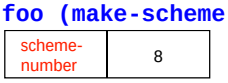
```
((scheme-number scheme-number) . 手続き)
 (rational rational) . 手続き)
 (complex complex) . 手続き)
 ...)
```

- put get の定義を思い出そう (連想リスト)

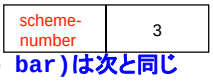
20

Scheme number パッケージの使用法

```
(define (make-scheme-number n)
   ((get 'make 'scheme-number) n) )
(define foo (make-scheme-number 8))
```




```
(define bar (make-scheme-number 3))
```



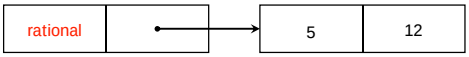
(add foo bar)は次と同じ

```
((get 'add '(scheme-number scheme-number))
  (contents foo) (contents bar) )
(+ 8 3)
```

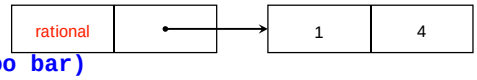
21

Rational number パッケージの使用法

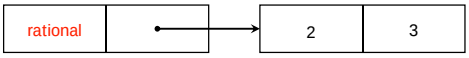
```
(define (make-rational n d)
   ((get 'make 'rational) n d)) )
(define foo (make-rational 5 12))
```




```
(define bar (make-rational 1 4))
```




```
(add foo bar)
((get 'add '(rational rational))
 (contents foo) (contents bar) )
(add-rat (contets foo) (contents bar))
```



25

Complex number パッケージ

Complexには，rectangular と polar というサブタイプ（部分型）があることを思い出そう．

```
(define(install-complex-package)
  ;; imported procedures from rectangular and polar
   packages

   (define (make-from-real-imag x y)
      ((get 'make-from-real-imag
            'rectangular) x y ))
   (define (make-from-mag-ang r a)
      ((get 'make-from-mag-ang 'polar)
       r a ))
```

26

Ex.2.77 Complex number パッケージ

```
(define (make-complex-from-real-imag x y)
   ((get 'make-from-real-imag 'complex)
    x y ))

(define (make-complex-from-mag-ang r a)
   ((get 'make-from-mag-ang 'complex)
    r a ))
```

■ Complex number の genetic operations

```
(put 'real-part '(complex) real-part)
(put 'imag-part '(complex) imag-part)
(put 'magnitude '(complex) magnitude)
(put 'angle '(complex) angle)
```

31

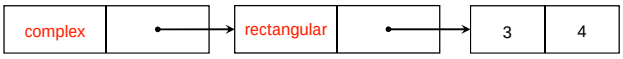
Complex number パッケージの使用法

```
(define (make-complex-from-real-imag x y)
   ((get 'make-from-real-imag 'complex)
    x y ))

(define (make-complex-from-mag-ang r a)
   ((get 'make-from-mag-ang 'complex)
    r a ))

(define foo
   (make-complex-from-real-imag 3 4) )
```

3+4*i*



32

西陣織（体験工房あり）

- http://kyoori.or.jp/

西陣織「紋意匠図」は西陣織を製織するための設計図、あるいは指図（さしず）ともいう。紙紙（けいがみ）という一種の方眼紙へ、現物の図案を引きのばして色別にかき直したもの。方眼紙の桁の一コマが経（たて）糸、緯（よこ）糸の一本にあたる。

紋紙への穴開け作業（ピアノマシン）

紋紙 • http://blog.goo.ne.jp/vitello/

38

1月7日・本日のメニュー

- 2.5 Genetic Operation System
- 2.5.0 Put and Get
- 2.5.1 Generic Arithmetic Operations
- 2.5.2 Combining Data of Different Types
- 2.5.3 Symbolic Algebra

39

2.5.2 Combining Data of Different Types

- 異なるタイプ同士に算術演算を拡張する。
- 引数のタイプにより適切な演算を行う手続きを適用
- 次の案はどうか？

```
;;  to be included in the complex package
(define (add-complex-to-schemenum z x)
   (make-from-real-imag
      (+ (real-part z) x)
      (imag-part z)))
(put 'add '(complex scheme-number)
   (lambda (z x)
     (tag (add-complex-to-schemenum z x))
   ))
```

40

Coercion(強制型変換)

- 異なるタイプ同士に算術演算を拡張
- 引数のタイプにより適切な演算を行う手続きを適用
- 手続きを適用する前に、対応するタイプでない引数についてはそのタイプに変換

```
(+ 3 3.1) ⇒ (+ 3.0 3.1)
(* 3+4i 2) ⇒ (+ 3+4i 2+0i)
```

```
(define (scheme-number->complex n)
   (make-complex-from-real-imag
       (contents n) 0 ))
(put-coercion
    'scheme-number 'complex
    scheme-number->complex )
```

41

Coercion(強制型変換)

```
(define (apply-generic op . args)
 (let ((type-tags (map type-tag args)))
   (let ((proc (get op type-tags)))
     (if proc
        (apply proc (map contents args))
        (if (= (length args) 2)
            (let ((type1 (car type-tags))
                  (type2 (cadr type-tags))
                  (a1 (car args))
                  (a2 (cadr args)) )
              (let ((t1->t2 (get-coercion type1 type2))
                    (t2->t1 (get-coercion type2 type1)) )
                (cond (t1->t2
                       (apply-generic op (t1->t2 a1) a2))
                      (t2->t1
                       (apply-generic op a1 (t2->t1 a2)))
                      (else
                       (error "No method for these types"
                              (list op type-tags) )))))
            (error "No method for these types"
                        (list op type-tags) ))))))
```

42

Coercion(強制型変換、改)

```
(define (apply-generic op . args)
 (let* ((type-tags (map type-tag args))
        (proc (get op type-tags)) )
   (if proc
       (apply proc (map contents args))
       (if (= (length args) 2)
           (let* ((type1 (car type-tags))
                  (type2 (cadr type-tags))
                  (a1 (car args))
                  (a2 (cadr args))
                  (t1->t2 (get-coercion type1 type2))
                  (t2->t1 (get-coercion type2 type1)) )
             (cond (t1->t2
                    (apply-generic op (t1->t2 a1) a2))
                   (t2->t1
                     (apply-generic op a1 (t2->t1 a2)))
                   (else
                     (error "No method for these types"
                            (list op type-tags) )))))
           (error "No method for these types"
                  (list op type-tags) ))))))
```

43

Hierarchies of types（型階層）

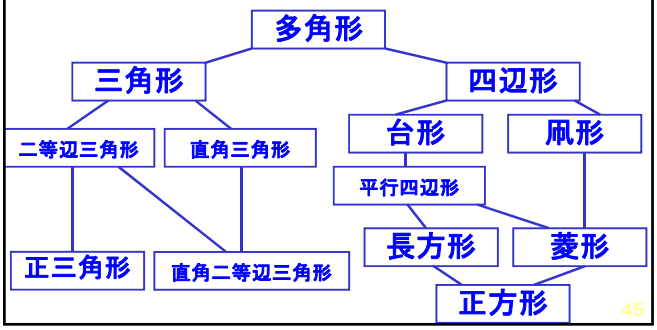
- Coercionではどの型へ変換するかが重要.
- 単純な場合： すぐ上位に変換

- *Tower of types* （型の塔）
- 演算結果の単純化には既約化だけでなく、型階層を低位に簡略化することも含まれる。
- 例： 4.0+3.7i + 5.0-3.7i
- 結果は複素数ではなく、実数

44

Hierarchies of types（型階層）

複雑な場合： 両者に共通の上位に変換



45

Inadequencies of hierachies

- 演算結果の単純化には既約化だけでなく、型階層を低位に簡略化することも含まれる。
- 例： 4.0+3.7i + 5.0-3.7i
- 結果は複素数ではなく、実数

- Ex2.83, 84 `raise` の設計

- Ex2.85 `drop` の設計

47

Abstraction barrier の効用

1. インタフェースの手続き(generic nameで)を定義しておけば，その手続きをどのように実装するかの決定は遅延できる.
2. インタフェースの実装はタイプにより規定.
3. 複数のタイプに対して手続きを適用する前には，対応するタイプに強制型変換を行う.
4. 同じインタフェースを複数のタイプで実装することも可能.

- 汎用演算(generic operations)
- 情報隠蔽(information hiding)

48

12月25日・本日のメニュー

NoStudent Left Behind

2.5 Genetic Operation System
2.5.0 Put and Get
2.5.1 Generic Arithmetic Operations
2.5.2 Combining Data of Different Types
2.5.3 Symbolic Algebra

49

記号代数ー多項式・有理関数

1. 多項式(Arithmetic on polynomials)
2. 多項式の表現(Representation of poly.)
   - 次数をすべて記する
   - 項リスト(term lists)で表現
3. 多項式の型の階層
4. 有理関数への拡張(簡約化が必要)

50

宿題：1月13日24時　締切

1. 教科書 2-3-3 を読み，汎用演算システムの場合を参考にして，多項式システムを設計する上での課題について述べ，あなたの解決方針を提案しなさい．（最低A41枚，12ポイント，single space，marginは1inch以下）
2. ソーティング法を1つ調べて，そのアルゴリズムと計算量についてまとめなさい．（最低A41枚，同様）
3. 上記2つのレポートを SICP-12@zeus.kuis.kyoto-u.ac.jp に送付

- 友達に教えてもらったら，明記すること．
- Webは出展を明記．（otherwise『同じ』回答は減点）

51