アルゴリズムとデータ構造入門

2.5 汎用演算システム

Systems with Generic Operations

奥 乃 博

背伸びし合って勁くなる。
自由な学風に甘えて
ぬるま湯につからない。

日経新聞朝刊12月14日1面

12月20日・本日のメニュー

- 2.4 Multiple Representations for Abstract Data
  - 2.4.3 Data-Directed Programming and Additivity
- 2.5 Systems with Generic Operations
  - 2.5.1 Generic Arithmetic Operations
  - 2.5.2 Combining Data of Different Types
  - 2.5.3 Example: Symbolic Algebra
- 配布する用紙に名前を記入して下さい。
- 今週の宿題と共に提出して下さい。

汎用複素数演算システムの構造

複素数を使ったプログラム

プログラム領域での複素数

`add-complex,sub-complex,mul等`

複素数演算パッケージ

`real-part imag-part magnitude angle`

直交座標表現 (Rectangular representation)

極座標表現 (Polar representation)

`cons car cdr + *`

リスト構造と基本マシン算術

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。**

4

Data-Directed Programming (データ駆動型プログラミング)

- 加法的(additivity)なインタフェースとするために、表のようなデータを使用。

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

5

表の操作

- 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)でその処理法を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
- TUT-Scheme(tus2, tustk2)では、
- (define put putprop)
- (define get getprop)

7

表の操作で使うデータ

- 演算に関連する情報<*item*>は、以下では、手続き（ラムダ式）
- <*type*>は、*引数の型のリスト*
- 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）でその処理法を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

9

直交座標のタグ付きデータの表現法

```
(define(install-rectangular-package)
  ;; internal procedures
  (define (real-part z) (car z))
  (define (imag-part z) (cdr z))
  (define (make-from-real-imag x y) (cons x y))
  (define (magnitude z)
    (sqrt (+ (square (real-part z))
             (square (imag-part z)))))
  (define (angle z)
    (atan (imag-part z) (real-part z)))
  (define (make-from-mag-ang r a)
    (cons (* r (cos a)) (* r (sin a))))
  ;; interface to the rest of the system
  (define (tag x) (attach-tag 'rectangular x))
  (put 'real-part '(rectangular) real-part)
  (put 'imag-part '(rectangular) imag-part)
  (put 'magnitude '(rectangular) magnitude)
  (put 'angle '(rectangular) angle)
  (put 'make-from-real-imag 'rectangular
       (lambda (x y) (tag (make-from-real-imag x y))))
  (put 'make-from-mag-ang 'rectangular
       (lambda (r a) (tag (make-from-mag-ang r a))))
  'done)
```

12

極座標のタグ付きデータの表現法

```
(define(install-polar-package)
  ;; internal procedures
  (define (magnitude z) (car z))
  (define (angle z) (cdr z))
  (define (make-from-mag-ang r a) (cons r a))
  (define (real-part z)
    (* (magnitude z) (cos (angle z))))
  (define (imag-part z)
    (* (magnitude z) (sin (angle z))))
  (define (make-from-real-imag x y)
    (cons (sqrt (+ (square x) (square y)))
          (atan y x)))
  ;; interface to the rest of the system
  (define (tag x) (attach-tag 'polar x))
  (put 'real-part '(polar) real-part)
  (put 'imag-part '(polar) imag-part)
  (put 'magnitude '(polar) magnitude)
  (put 'angle '(polar) angle)
  (put 'make-from-real-imag 'polar
       (lambda (x y) (tag (make-from-real-imag x y))))
  (put 'make-from-mag-ang 'polar
       (lambda (r a) (tag (make-from-mag-ang r a))))
  'done)
```

15

generic operation の適用方法

```
(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))
          (error
            "No method for these types –
             APPLY-GENERIC"
            (list op type-tags))))))
  'done)
```

これでgeneric procedure を再定義する。

```
(define (real-part z)
    (apply-generic 'real-part z))
```

generic operation の適用方法

```
(define (real-part z)
   (apply-generic 'real-part z))

(define (imag-part z)
   (apply-generic 'imag-part z))

(define (magnitude z)
   (apply-generic 'magnitude z))

(define (angle z)
   (apply-generic 'angle z))
```

17

目的に合致した複素数表現を選ぶ

- 直交座標表現 if 実数部と虚数部が分かっているとき
- 極座標表現 if 半径と角度が分かっているとき

```
(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))
```

- ほうら、うまく目的に合致した複素数表現が選ばれて、その後、型に対応した手続きが自動的に選択されることがお分かりになったでしょう。

18

Symbolic differentiation

```
(define (deriv exp var)
  (cond ((number? exp) 0)
        ((variable? exp)
         (if (same-variable? exp var) 1 0) )
        ((sum? exp)
         (make-sum (deriv (addend exp) var)
                   (deriv (augend exp) var) ))
        ((product? exp)
         (make-sum
           (make-product (multiplier exp)
                 (deriv (multiplicand exp) var) )
           (make-product
                 (deriv (multiplier exp) var)
                 (multiplicand exp) )))
        <more rules can be added here>
        (else (error "unknown expression type –
                      DERIV" exp ))))
```

19

Symbolic differentiation

- Additivity にする

```
(define (deriv exp var)
   (cond ((number? exp) 0)
         ((variable? exp)
          (if (same-variable? exp var)
              1
              0 ))
         (else
           ((get 'deriv (operator exp))
            (operands exp)
            var ))))

(define (operator exp) (car exp))
(define (operands exp) (cdr exp))
```

20

Data-Directed Programmingのポイント

- 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

21

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

Message passing

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))
```

23

ペア（対、pair）を手続きで実現

```
(define (cons x y)
  (define (dispatch m)
    (cond ((= m 0) x)
          ((= m 1) y)
          (else (error "Argument not 0
                  or 1 -- CONS" m ))))
   dispatch)
(define (car z) (z 0))
(define (cdr z) (z 1))
```

- (define foo (cons 10 25))
- (car foo) ⇒ (foo 0) ⇒ 10
- (cdr foo) ⇒ (foo 1) ⇒ 25

24

12月20日・本日のメニュー

- 2.4 Multiple Representations for Abstract Data
  - 2.4.3 Data-Directed Programming and Additivity
- 2.5 Systems with Generic Operations
  - 2.5.1 Generic Arithmetic Operations
  - 2.5.2 Combining Data of Different Types
  - 2.5.3 Example: Symbolic Algebra
- 配布する用紙に名前を記入して下さい。
- 今週の宿題と共に提出して下さい。

26

汎用複素数演算システム（2.4）

複素数を使ったプログラム

プログラム領域での複素数

`add-complex,sub-complex,mul等`

複素数演算パッケージ

`real-part imag-part magnitude angle`

直交座標表現（Rectangular representation）

極座標表現（Polar representation）

`cons car cdr + *`

リスト構造と基本マシン算術

27

有理数システム（2.1）

有理数を使ったプログラム

プログラム領域での有理数

`add-rat sub-rat mul-等`

分子と分母から構成される有理数

`make-rat numer den`

```
(define (make-rat n d)
  (let ((g (gcd n d)))
    (cons (/ n gcd) ( / d gcd)) ))
```

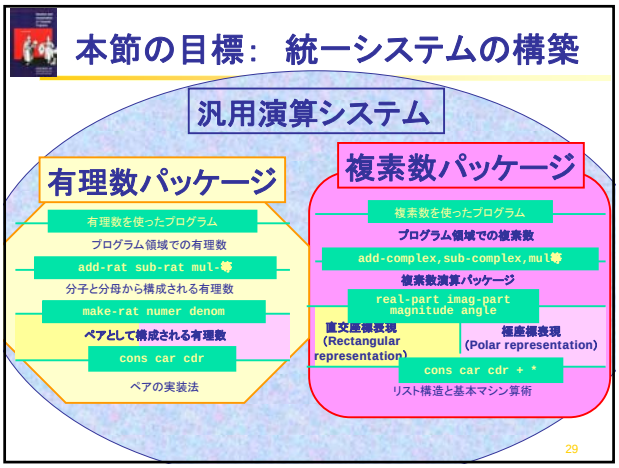
ペアとして構成される有理数

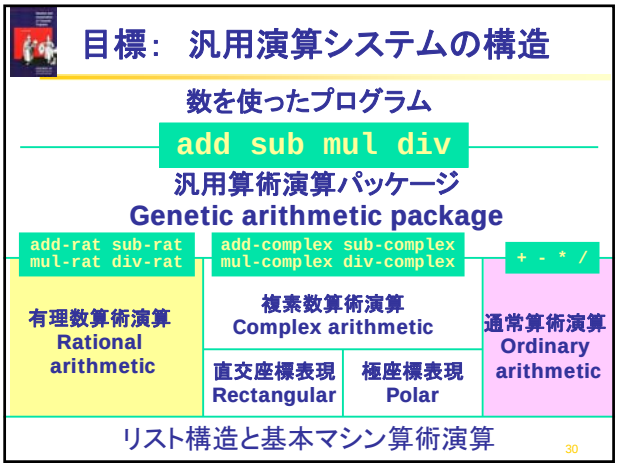
```
(define (make-rat n d)
  (cond n d))
(define (numer x)
  (let ((g (gcd (car x) (cdr x))))
    (/ (car x) g) ))
(define (denom x)
  (let ((g (gcd (car x) (cdr x))))
    (/ (cdr x) g) ))
```

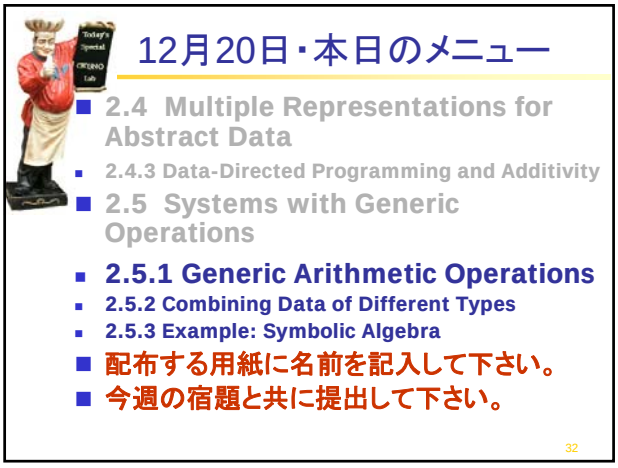
`cons car cdr`

ペアの実装法

28


本節の目標： 統一システムの構築
汎用演算システム
有理数パッケージ
有理数を使ったプログラム
プログラム領域での有理数
add-rat sub-rat mul-等
分子と分母から構成される有理数
make-rat numer denom
ペアとして構成される有理数
cons car cdr
ペアの実装法
複素数パッケージ
複素数を使ったプログラム
プログラム領域での複素数
add-complex,sub-complex,mul等
複素数演算パッケージ
real-part imag-part
magnitude angle
直交座標表現
(Rectangular representation)
極座標表現
(Polar representation)
cons car cdr + *
リスト構造と基本マシン算術
29


目標： 汎用演算システムの構造
数を使ったプログラム
add sub mul div
汎用算術演算パッケージ
Genetic arithmetic package
add-rat sub-rat
mul-rat div-rat
有理数算術演算
Rational
arithmetic
add-complex sub-complex
mul-complex div-complex
複素数算術演算
Complex arithmetic
直交座標表現
Rectangular
極座標表現
Polar
+ - * /
通常算術演算
Ordinary
arithmetic
リスト構造と基本マシン算術演算
30


12月20日・本日のメニュー
2.4 Multiple Representations for Abstract Data
2.4.3 Data-Directed Programming and Additivity
2.5 Systems with Generic Operations
2.5.1 Generic Arithmetic Operations
2.5.2 Combining Data of Different Types
2.5.3 Example: Symbolic Algebra
配布する用紙に名前を記入して下さい。
今週の宿題と共に提出して下さい。
32

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

```
(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) )
```

33

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 )
```

34

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))
```

scheme-number	8

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

scheme-number	3

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

scheme-number	11

35

Rational number パッケージ

```
(define (install-rational-package)
   ;; internal procedures
  (define (numer x) (car x))
  (define (denom x) (cdr x))
  (define (make-rat n d)
    (let ((g (gcd n d)))
      (cons (/ n g) (/ d g))))
  (define (add-rat x y)
    (make-rat (+ (* (numer x) (denom y))
                 (* (numer y) (denom x)) )
              (* (denom x) (denom y)) ))
  (define (sub-rat x y)
    (make-rat (- (* (numer x) (denom y))
                 (* (numer y) (denom x)) )
              (* (denom x) (denom y)) ))
```

Rational number パッケージ（続）

```
(define (install-rational-package)
   ;; internal procedures

  (define (mul-rat x y)
    (make-rat (* (numer x) (numer y))
              (* (denom x) (denom y)) ))
  (define (div-rat x y)
    (make-rat (* (numer x) (denom y))
              (* (denom x) (numer y)) ))
```

37

Rational number パッケージ（続々）

```
(define (install-rational-package)
  ;; interface to rest of the system
  (define (tag x) (attach-tag 'rational x))
  (put 'add '(rational rational)
       (lambda (x y) (tag (add-rat x y))))
  (put 'sub '(rational rational)
       (lambda (x y) (tag (sub-rat x y))))
  (put 'mul '(rational rational)
       (lambda (x y) (tag (mul-rat x y))))
  (put 'div '(rational rational)
       (lambda (x y) (tag (div-rat x y))))
  (put 'make 'rational
       (lambda (n d) (tag (make-rat n d))))
  'done )
```

38

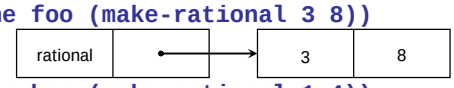
Rational number パッケージ

```
(define (install-rational-package)
  ;; interface to rest of the system
  (define (tag x) (attach-tag 'rational x))
  (put 'add '(rational rational)
       (lambda (x y) (tag (add-rat x y))))
  (put 'sub '(rational rational)
       (lambda (x y) (tag (sub-rat x y))))
  (put 'mul '(rational rational)
       (lambda (x y) (tag (mul-rat x y))))
  (put 'div '(rational rational)
       (lambda (x y) (tag (div-rat x y))))
  (put 'make 'rational
       (lambda (n d) (tag (make-rat n d))))
  'done )
```

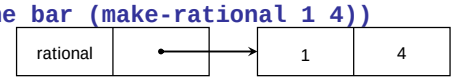
39

Rational number パッケージの使用法

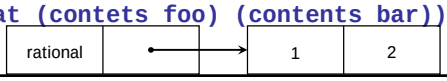
```
(define (make-rational n d)
   ((get 'make 'rational) n d)) )
(define foo (make-rational 3 8))
```




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




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



40

Complex number パッケージ

```
(define(install-complex-package)
  ;; i 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 ))
```

41

Complex number パッケージ(続)

```
(define(install-complex-package)
  ;; internal procedures
  (define (add-complex z1 z2)
     (make-from-real-imag
       (+ (real-part z1) (real-part z2))
       (+ (imag-part z1) (imag-part z2))))
  (define (sub-complex z1 z2)
     (make-from-real-imag
       (- (real-part z1) (real-part z2))
       (- (imag-part z1) (imag-part z2))))
  (define (mul-complex z1 z2)
     (make-from-mag-ang
       (* (magnitude z1) (magnitude z2))
       (+ (angle z1) (angle z2))))
```

42

Complex number パッケージ(続々)

```
(define(install-complex-package)
  ;; internal procedures

  (define (div-complex z1 z2)
     (make-from-mag-ang
       (/ (magnitude z1) (magnitude z2))
       (- (angle z1) (angle z2))))

  ;; internal procedures
  (define (tag z) (attach-tag 'complex z))
  (put 'add '(complex complex)
       (lambda (z1 z2)
          (tag (add-complex z1 z2)) ))
```

43

Complex number パッケージ(4)

```
(define(install-complex-package)
  ;; internal procedures

  (put 'sub '(complex complex)
       (lambda (z1 z2)
          (tag (sub-complex z1 z2)) ))
  (put 'mul '(complex complex)
       (lambda (z1 z2)
          (tag (mul-complex z1 z2)) ))
  (put 'div '(complex complex)
       (lambda (z1 z2)
          (tag (div-complex z1 z2)) ))
```

44

Complex number パッケージ(5)

```
(define(install-complex-package)
  ;; internal procedures

  (put 'make-from-real-imag 'complex
       (lambda (x y)
          (tag (make-from-real-imag x y))
       ))
  (put 'make-from-mag-ang 'complex
       (lambda (r a)
          (tag (make-from-mag-ang r a))))
  'done )
```

45

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)
```

46

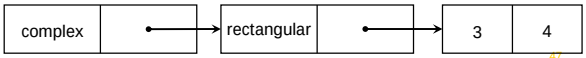
Complex number パッケージの使用法

```
(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+4i



47

Ex.2.78 Ordinary number パッケージ

- Scheme-number を効率化したい
- 基本手続きは、内部でタイプチェックをしている
- symbol? number? pair? などを使用
- scheme-number では、タイプチェックをシステムに任せて、高速化したい。

48

タグ付きデータの実装法（再掲）

- 手続きは type-tag （型タグ）で処理を区別する。
- type-tag はデータに付与されている。

```
(define (attach-tag type-tag contents)
  (cons type-tag contents))
(define (type-tag datum)
  (if (pair? datum)
      (car datum)
      (error "Bad tagged datum –
              TYPE-TAG" datum)))
(define (contents datum)
  (if (pair? datum)
      (cdr datum)
      (error "Bad tagged datum –
              CONTENTS" datum)))
```

49

scheme-number の効率化

```
(define (attach-tag type-tag contents)
  (if (eq? type-tag 'scheme-number)
      contents
      (cons type-tag contents) ))
(define (type-tag datum)
  (if (pair? datum)
      (car datum)
      'scheme-number ))
(define (contents datum)
  (if (pair? datum)
      (cdr datum)
      datum ))
```

- タイプ（型）チェックはシステムに任せた！

50

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 )
```

51

12月20日・本日のメニュー

- 2.4 Multiple Representations for Abstract Data
  - 2.4.3 Data-Directed Programming and Additivity
- 2.5 Systems with Generic Operations
  - 2.5.1 Generic Arithmetic Operations
  - **2.5.2 Combining Data of Different Types**
  - 2.5.3 Example: Symbolic Algebra
- 配布する用紙に名前を記入して下さい。
- 今週の宿題と共に提出して下さい。

53

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
- 案1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))
   ))
```

54

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
- 案2はどうか？

  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する。

強制型変換（*Coercion*）と呼ぶ。

```
(+ 3 3.1)

(* 3+4i 2)
```

55

Coercion（強制型変換）

- 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
- 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する。

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

56

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) ))))))
```

57

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) ))))))
```

58

Hierarchies of types（型階層）

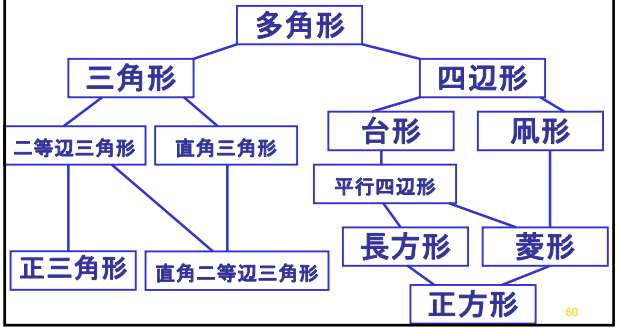
- Coercionではどの型へ変換するかが重要。
- 単純な場合：

- *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

59

Hierarchies of types（型階層）

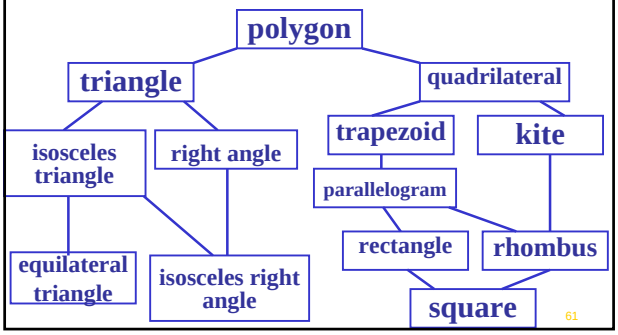
- 複雑な場合：



60

Hierarchies of types（型階層）

- 複雑な場合：



61

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

62

宿題：配布用紙と共に提出すること

- 宿題は、次の計9問：
- **Ex.2.75, 2.76, 2.77,**
- **Ex.2.78, 2.79, 2.80,**
- **Ex.2.83, 2.84, 2.85**

1月10日正午締切

よいお年をお迎えください