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
2.4 抽象データの多重表現
2.5 汎用演算システム

奥　乃　博
大学院情報学研究科知能情報学専攻
知能メディア講座　音声メディア分野
http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/08/IntroAlgDs/
okuno@i.kyoto-u.ac.jp

https://cms03.media.kyoto-u.ac.jp/webct/logon/343184001

1月6日・本日のメニュー

1. 2.4 Multiple Representations for Abstract Data
2. 2.4.1 Representations for Complex Numbers
3. 2.4.2 Tagged data
4. 2.4.3 Data-Directed Programming and Additivity

jakld.jar の起動（訂正）

1. http://ryujin.kuis.kyoto-u.ac.jp/~yuasa/jakld/index-j.html にあるドキュメント（READMEjp.txt）を見てください．この中に，いくつかの有用な手続きが書かれています．
2. **picture.lsp はjakld に組み込まれました.**
3. **java –jar jakld.jar** を起動
4. **(start-picture)**　で **window** が起動.
5. **READMEjp.txt** を見て，frm1，letterlambdaを定義
6. このファイルを **my-defs.lsp** とすると **(load "my-defs.lsp")**
7. **(letterlambda frm1)** でλが表示される.

1月6日・本日のメニュー

1. 2.4 Multiple Representations for Abstract Data
2. 2.4.1 Representations for Complex Numbers
3. 2.4.2 Tagged data
4. 2.4.3 Data-Directed Programming and Additivity

複素数システムのデータ抽象化の壁

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

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

33

複素数の演算

1. 虚数（imaginary part）

$$z = x + iy \qquad i^2 = -1$$

2. 加算（addition）

$$\mathrm{Real-part}(z_1 + z_2) = \mathrm{Real-part}(z_1) + \mathrm{Real-part}(z_2)$$
$$\mathrm{Imaginary-part}(z_1 + z_2) = \mathrm{Imaginary-part}(z_1) + \mathrm{Imaginary-part}(z_2)$$

3. 乗算（multiplication）

$$\mathrm{Re}(z_1 \cdot z_2) = \mathrm{Re}(z_1)\cdot\mathrm{Re}(z_2) - \mathrm{Im}(z_1)\cdot\mathrm{Im}(z_2)$$
$$\mathrm{Im}(z_1 \cdot z_2) = \mathrm{Re}(z_1)\cdot\mathrm{Im}(z_2) + \mathrm{Im}(z_1)\cdot\mathrm{Re}(z_2)$$

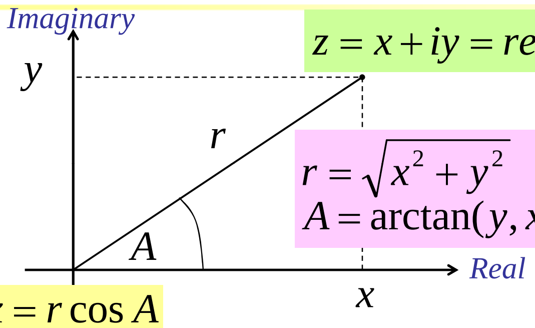
$$\mathrm{Magnitude}(z_1 \cdot z_2) = \mathrm{Magnitude}(z_1)\cdot\mathrm{Magnitude}(z_2)$$
$$\mathrm{Angle}(z_1 \cdot z_2) = \mathrm{Angle}(z_1) + \mathrm{Angle}(z_2)$$

複素数の四則演算

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

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

複素数の2種類の表現法



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

$r = \sqrt{x^2 + y^2}$
$A = \arctan(y, x)$

$x = r\cos A$
$y = r\sin A$

複素数の2種類の表現法の実装

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

```
(make-from-real-imag
   (real-part z) (imag-part z) )
(make-from-mag-ang
   (magnitude z) (angle z) )
```

$x = r\cos A$
$y = r\sin A$

$r = \sqrt{x^2 + y^2}$
$A = \arctan(y, x)$

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))) )
```

38

複素数の表現法(続) $z = x + iy = re^{iA}$

■ 選択子(**selectors**)

```
(define (real-part z)
  (* (magnitude z) (cos (angle z))) )
(define (imag-part z)
  (* (magnitude z) (sin (angle z))) )
(define (magnitude z) (car z))
(define (angle z) (cdr z))
```

■ 構築子(**constructors**)

```
(define (make-from-real-imag x y)
  (cons (sqrt (+ (square x) (square y)))
        (atan y x)) )
(define (make-from-mag-ang r a)
  (cons r a) )
```

39

図形言語に複素数を導入

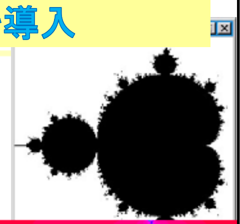
$$z_{n+1} = z_n{}^2 + C$$
$$z_0 = C$$

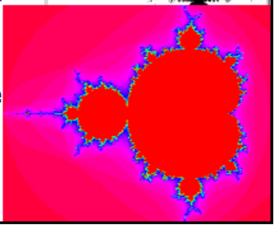
の収束点C= (x, y)の集合

Mandelbrot Set

右上の図は**frame coordinate map**未使用(直接点を描画)

http://mathworld.wolfram.com/MandelbrotSet.html





2.4.2 Tagged data（タグ付きデータ）

- データ抽象化の１つの観点
- *Principle of least commitment*
（最小責任の原則）
- 選択子と構築子を使用した抽象化の壁を使って、データオブジェクトの具体的な表現をできるだけ遅くし、システム設計における柔軟性を最大限にする。
- 本節ではさらに **principle of least commitment** を発展させる。

1. 表現法（選択子と構築子）の設計後でも、表現法の抽象化（曖昧性）を維持。
2. 直交座標と極座標が共用できる仕組みを考える。

43

The Principle of Least Commitmentの例

京都大学は、創立以来築いてきた自由の学風を継承し、発展させつつ、多元的な課題の解決に挑戦し、地球社会の調和ある共存に貢献するため、自由と調和を基礎に、ここに基本理念を定める。

教育

1. 京都大学は、多様かつ調和のとれた教育体系のもと、対話を根幹として自学自習を促し、卓越した知の継承と創造的精神の涵養につとめる。

2. 京都大学は、教養が豊かで人間性が高く責任を重んじ、地球社会の調和ある共存に寄与する、優れた研究者と高度の専門能力をもつ人材を育成する。

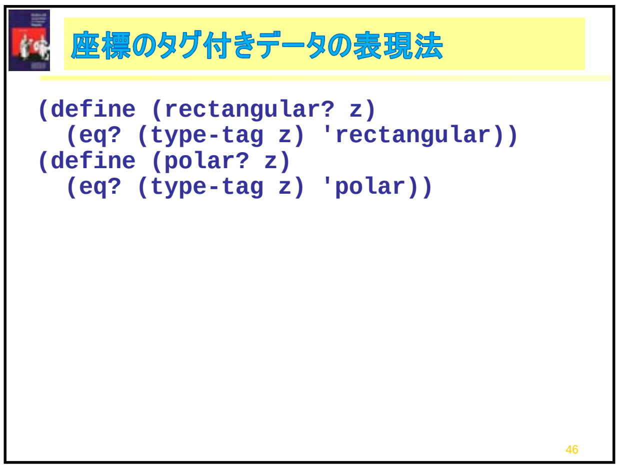
44

タグ付きデータの実装法

- 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 )))
```

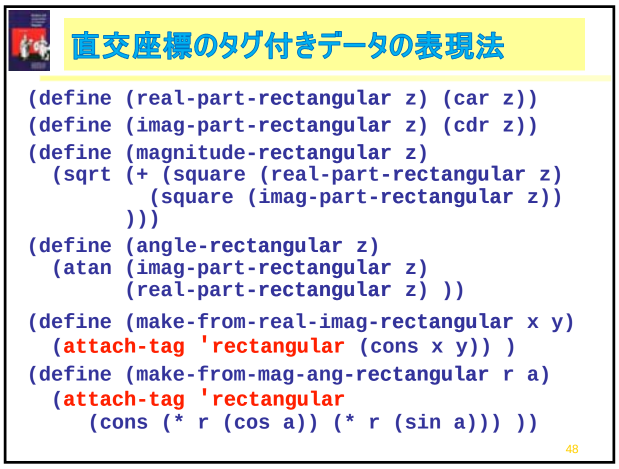
45



座標のタグ付きデータの表現法

```
(define (rectangular? z)
  (eq? (type-tag z) 'rectangular))
(define (polar? z)
  (eq? (type-tag z) 'polar))
```

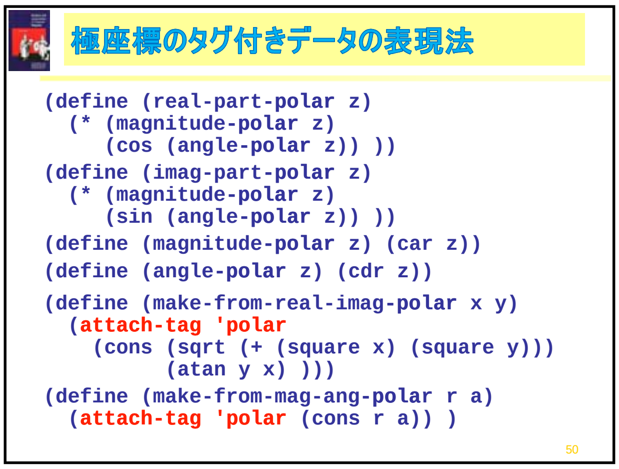
46



直交座標のタグ付きデータの表現法

```
(define (real-part-rectangular z) (car z))
(define (imag-part-rectangular z) (cdr z))
(define (magnitude-rectangular z)
  (sqrt (+ (square (real-part-rectangular z)
           (square (imag-part-rectangular z))
        )))
(define (angle-rectangular z)
  (atan (imag-part-rectangular z)
        (real-part-rectangular z) ))
(define (make-from-real-imag-rectangular x y)
  (attach-tag 'rectangular (cons x y)) )
(define (make-from-mag-ang-rectangular r a)
  (attach-tag 'rectangular
     (cons (* r (cos a)) (* r (sin a))) ))
```

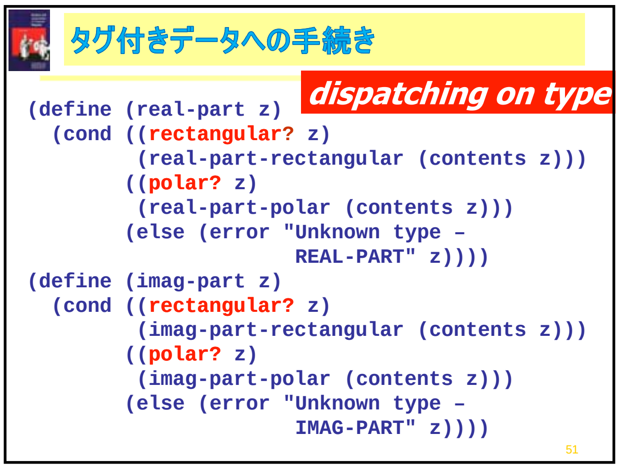
48



極座標のタグ付きデータの表現法

```
(define (real-part-polar z)
  (* (magnitude-polar z)
     (cos (angle-polar z)) ))
(define (imag-part-polar z)
  (* (magnitude-polar z)
     (sin (angle-polar z)) ))
(define (magnitude-polar z) (car z))
(define (angle-polar z) (cdr z))
(define (make-from-real-imag-polar x y)
  (attach-tag 'polar
    (cons (sqrt (+ (square x) (square y)))
          (atan y x) )))
(define (make-from-mag-ang-polar r a)
  (attach-tag 'polar (cons r a)) )
```

50

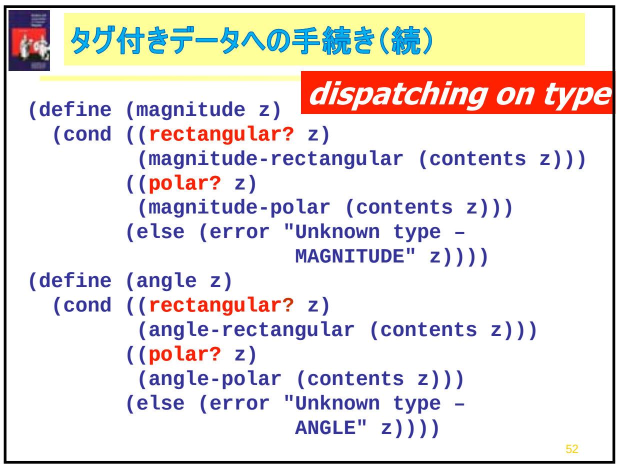


タグ付きデータへの手続き

dispatching on type

```
(define (real-part z)
  (cond ((rectangular? z)
         (real-part-rectangular (contents z)))
        ((polar? z)
         (real-part-polar (contents z)))
        (else (error "Unknown type –
                      REAL-PART" z))))
(define (imag-part z)
  (cond ((rectangular? z)
         (imag-part-rectangular (contents z)))
        ((polar? z)
         (imag-part-polar (contents z)))
        (else (error "Unknown type –
                      IMAG-PART" z))))
```

51

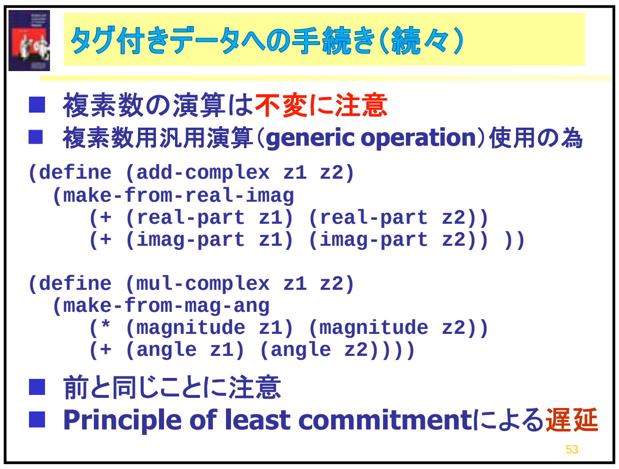


タグ付きデータへの手続き(続)

dispatching on type

```
(define (magnitude z)
  (cond ((rectangular? z)
         (magnitude-rectangular (contents z)))
        ((polar? z)
         (magnitude-polar (contents z)))
        (else (error "Unknown type –
                      MAGNITUDE" z))))
(define (angle z)
  (cond ((rectangular? z)
         (angle-rectangular (contents z)))
        ((polar? z)
         (angle-polar (contents z)))
        (else (error "Unknown type –
                      ANGLE" z))))
```

52



タグ付きデータへの手続き(続々)

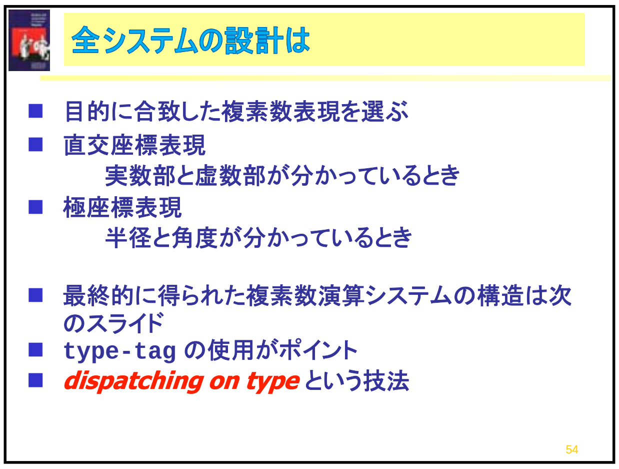
- 複素数の演算は不変に注意
- 複素数用汎用演算(**generic operation**)使用の為

```
(define (add-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))))
```

- 前と同じことに注意
- **Principle of least commitment**による遅延

53

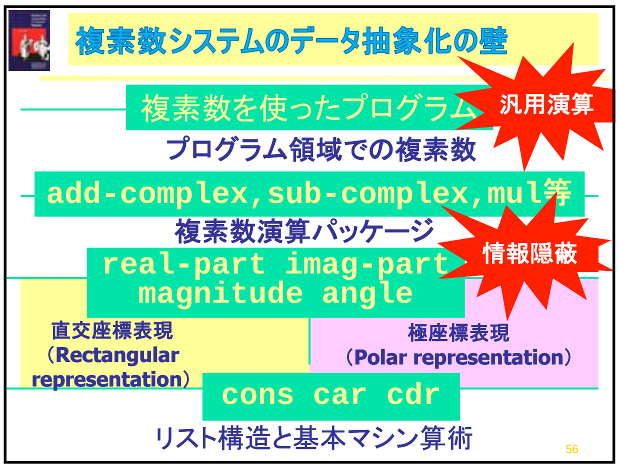


全システムの設計は

- 目的に合致した複素数表現を選ぶ
- 直交座標表現
  - 実数部と虚数部が分かっているとき
- 極座標表現
  - 半径と角度が分かっているとき

- 最終的に得られた複素数演算システムの構造は次のスライド
- `type-tag` の使用がポイント
- *dispatching on type* という技法

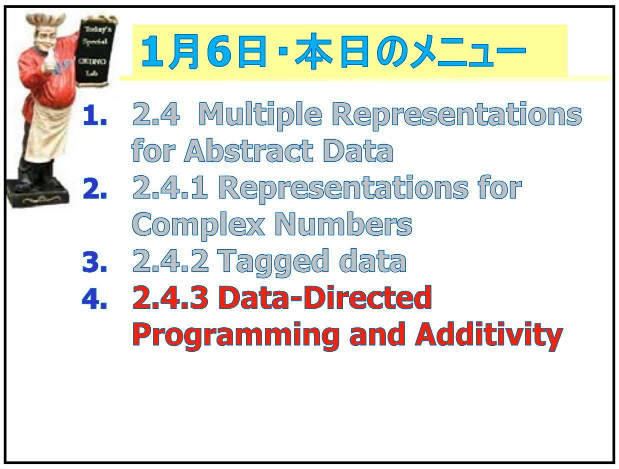
54



複素数システムのデータ抽象化の壁

複素数を使ったプログラム
汎用演算
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
情報隠蔽
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

56



1月6日・本日のメニュー

1. 2.4 Multiple Representations for Abstract Data
2. 2.4.1 Representations for Complex Numbers
3. 2.4.2 Tagged data
4. **2.4.3 Data-Directed Programming and Additivity**

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。

59

Data-Directed Programmingのポイント

- 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	型 (type) 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

60

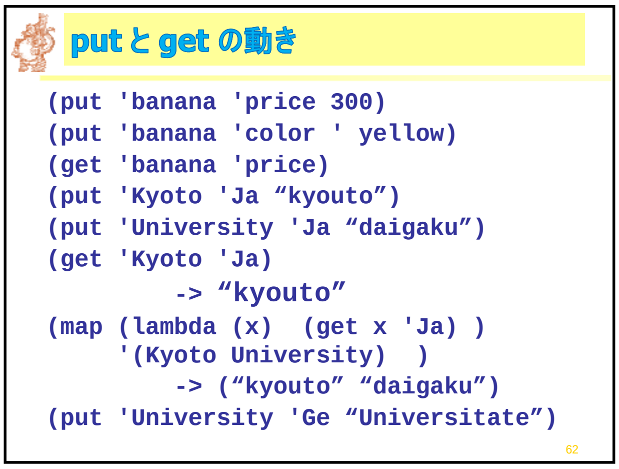
表の操作

- 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)

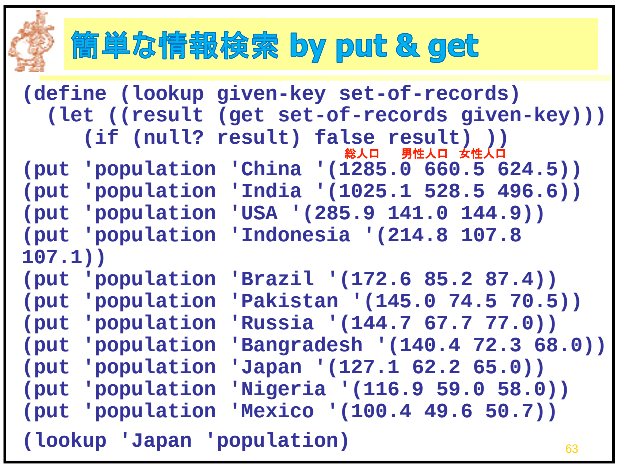
61



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)
         -> “kyoutoˮ
(map (lambda (x)  (get x 'Ja) )
     '(Kyoto University)  )
         -> (“kyouto” “daigaku”)
(put 'University 'Ge “Universitate”)
```

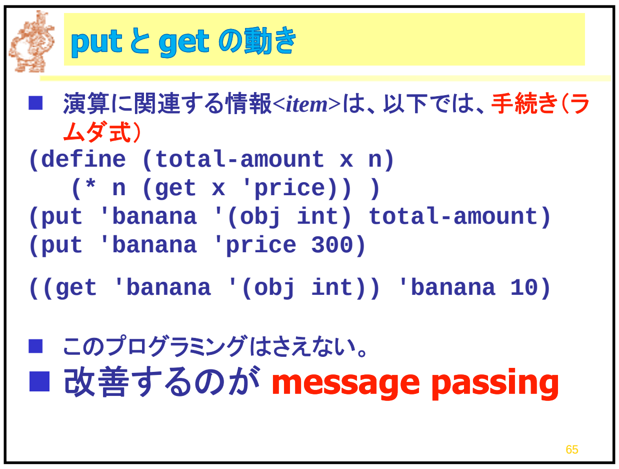
62



簡単な情報検索 by put & get

```
(define (lookup given-key set-of-records)
  (let ((result (get set-of-records given-key)))
     (if (null? result) false result) ))
                               総人口   男性人口  女性人口
(put 'population 'China '(1285.0 660.5 624.5))
(put 'population 'India '(1025.1 528.5 496.6))
(put 'population 'USA '(285.9 141.0 144.9))
(put 'population 'Indonesia '(214.8 107.8
107.1))
(put 'population 'Brazil '(172.6 85.2 87.4))
(put 'population 'Pakistan '(145.0 74.5 70.5))
(put 'population 'Russia '(144.7 67.7 77.0))
(put 'population 'Bangradesh '(140.4 72.3 68.0))
(put 'population 'Japan '(127.1 62.2 65.0))
(put 'population 'Nigeria '(116.9 59.0 58.0))
(put 'population 'Mexico '(100.4 49.6 50.7))
(lookup 'Japan 'population)
```

63



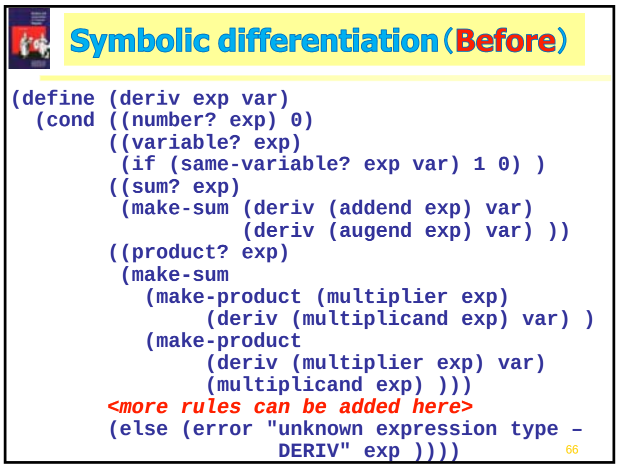
put と get の動き

- 演算に関連する情報<*item*>は、以下では、手続き(ラムダ式)

```
(define (total-amount x n)
   (* n (get x 'price)) )
(put 'banana '(obj int) total-amount)
(put 'banana 'price 300)
((get 'banana '(obj int)) 'banana 10)
```

- このプログラミングはさえない。
- 改善するのが **message passing**

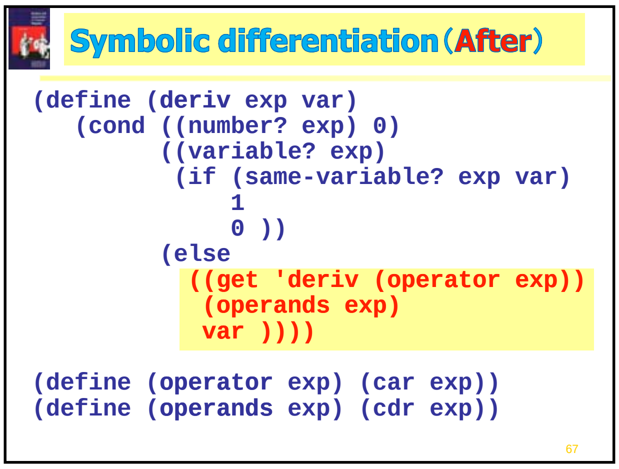
65



Symbolic differentiation（Before）

```
(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 ))))
```

66

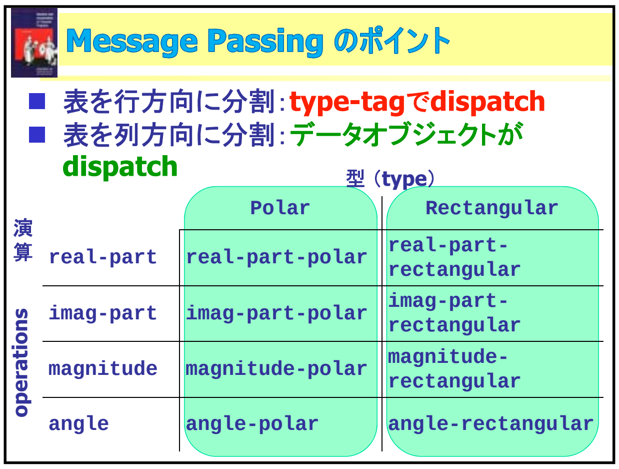


Symbolic differentiation（After）

```
(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))
```

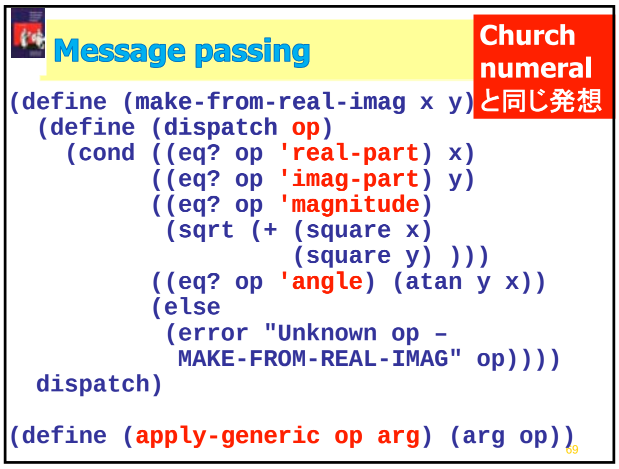
67



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

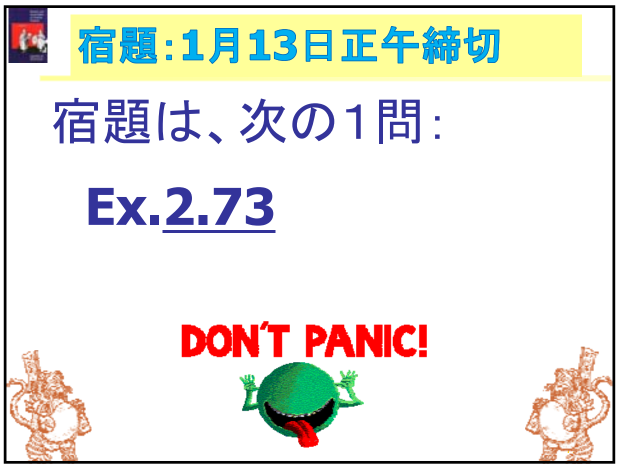


Message passing

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))
```



宿題:1月13日正午締切

宿題は、次の1問:

Ex.2.73