DDG University

Informal Goals

1. Learn new things related to technology.
2. Learn from each other.
3. Foster inter-team building.
4. To become better engineers.

Search for DDG University in Asana.

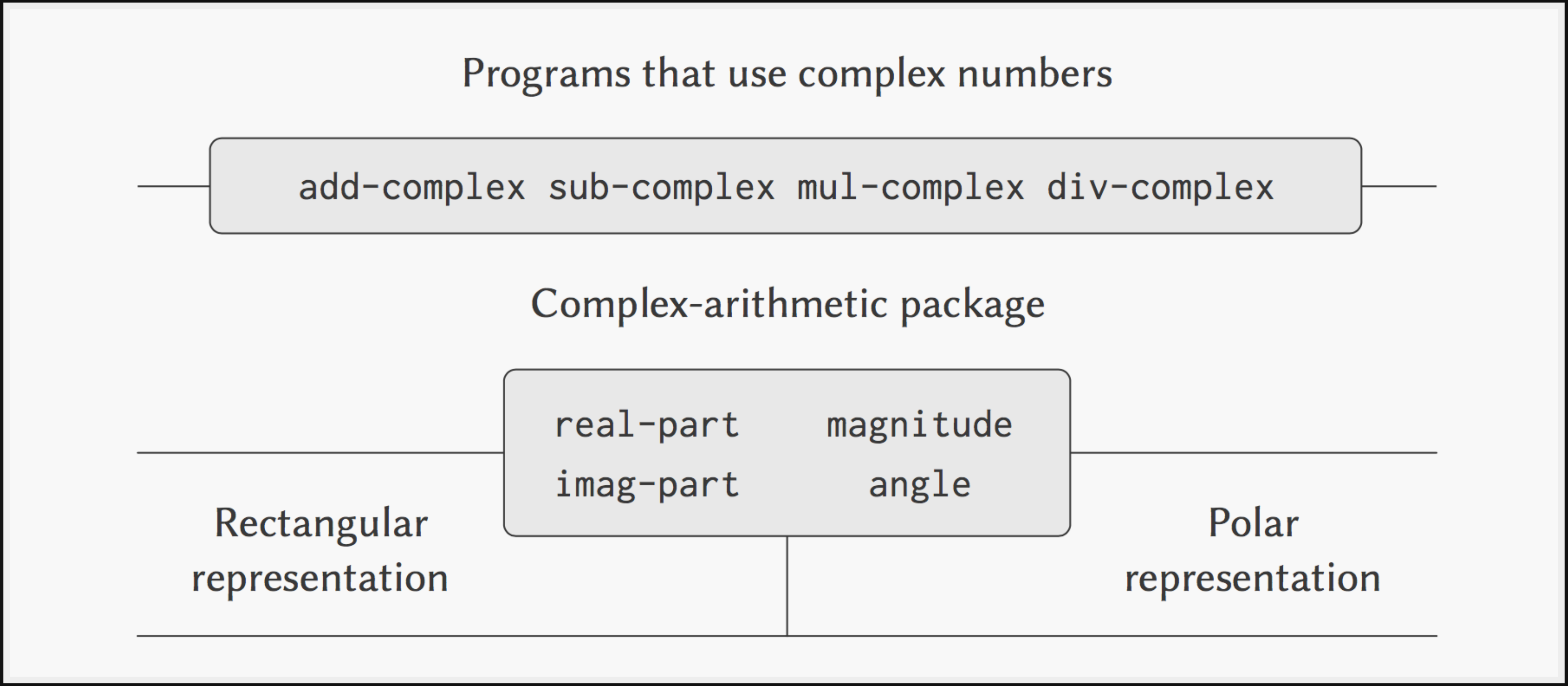
Structure and Interpretation of Computer Programs *(SICP)*

by Harold Abelson and Gerald Jay Sussman

2.4 Multiple Representations for Abstract Data

1. ***Generic operations with explicit dispatch***
2. ***Data-directed programming***
3. ***Message passing***

Complex Numbers API


Programs that use complex numbers
add-complex sub-complex mul-complex div-complex
Complex-arithmetic package
real-part
imag-part
magnitude
angle
Rectangular representation
Polar representation

Complex Numbers API (cont)

```scheme
(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))))
```

We need:

- `make-from-real-imag` | `make-from-mag-ang`
- `real-part` | `imag-part`
- `magnitude` | `angle`

Two Incompatible Implementations

```scheme
;;;A
(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)))

(define (make-from-real-imag x y)
  (cons x y))

(define (make-from-mag-ang r a)
  (cons (* r (cos a)) (* r (sin a))))
```

```scheme
;;;B
(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))

(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))
```

This won't work

- Two different internal representations.
- Naming conflicts.

Creating Types

```
 _____________________
|          |          |
| type tag | contents |
|__________|__________|
```

```
(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)))

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

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

Implementation A

```scheme
(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)))

(define (make-from-real-imag x y)
  (cons x y))

(define (make-from-mag-ang r a)
  (cons (* r (cos a)) (* r (sin a))))
```

```scheme
(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)))))
```

Implementation B

```scheme
(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))

(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))
```

```scheme
(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)))
```

Interfacing with the API

To use these implementations we either have to modify our complex number API or create the following interstitial layer:

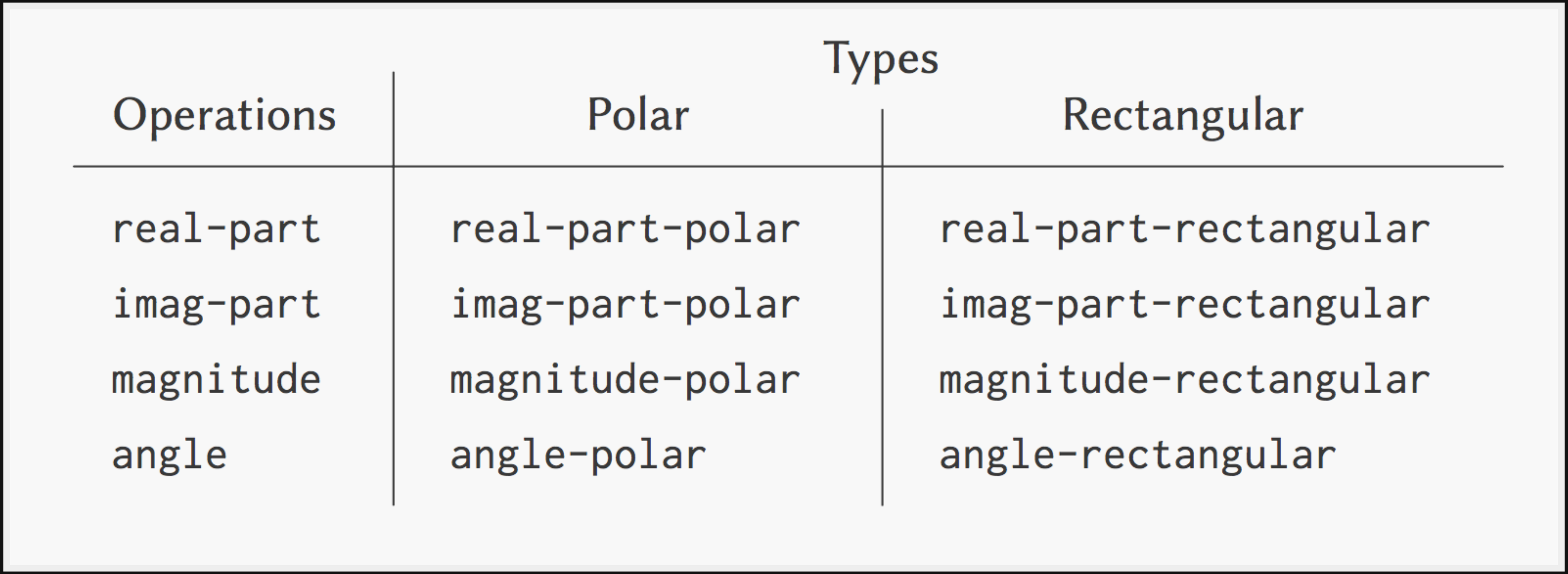
```scheme
(define (real-part z)
  (cond ((rectangular? z)
         (real-part-rectangular (contents z)))
        ((polar? z)
         (real-part-polar (contents z)))
        (else (error "Unknown type: REAL-PART" z))))

(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))))

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

We call this "generic operations with explicit dispatch"

Data-Directed Programming



	Types	
Operations	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`

```
(put ⟨op⟩ ⟨type⟩ ⟨item⟩)

(get ⟨op⟩ ⟨type⟩)
```

The Rectangular Implementation "Package"

```scheme
(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)
```

The Polar Implementation "Package"

```scheme
(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))))
```

Interfacing with the API

```
(define (real-part z)
  ((get 'real-part (type-tag z)) (contents z)))
(define (imag-part z)
  ((get 'imag-part (type-tag z)) (contents z)))
(define (magnitude z)
  ((get 'magnitude (type-tag z)) (contents z)))
(define (angle z)
  ((get 'angle (type-tag z)) (contents z)))

(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))
```

Message Passing

```
(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))
```

Look familiar?

```
(define foo (make-from-real-imag 10 20))
(foo 'real-part)
;10
```

That's all for section 2.4.

Thanks!