# Computer Programs Et09-12

dr. H. Corporaal
dr.ir. A.J.C. van Gemund
Computer Systems Group

http://cs.et.tudelft.nl/~heco
email: heco@cs.et.tudelft.nl

**Session 5: Data Abstraction II**



## Which problems are we going to solve?

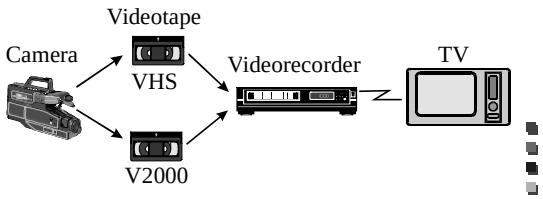
- Various data types have to be manipulated, e.g. look at the number hierarchy:
  - Complex
  - Real
  - Rational
  - Integer
- All these numbers need same kind of operations! (sum, mul, div, square, etc)
  - similar operations could even be performed on 'strange' types like polynomials



## Which problems are we going to solve?

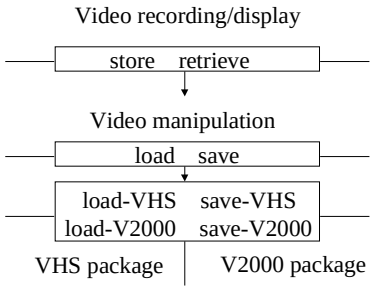
- Data types have multiple representations
  - E.g. complex numbers can be represented as:
    - rectangular format
    - polar format
  - Integers can be represented:
    - in different base (binary, octal, decimal, etc)
    - with different precision (short, long, bignum)
  - Each format may have its advantages and disadvantages
- Having $\nu$ types and $\mu$ representations/type and $\kappa$ operations, how do we deal with the $O(\kappa*\nu*\mu)$ complexity ?

## Example of Multiple data representations




Slide 1.5. 4

## Video data abstraction barriers




Slide 1.5. 5

## Video data abstraction terminology

- LOAD/SAVE are generic operators
- The choice of storage system (VHS/V2000) postponed as long as possible. Principle of least Commitment
- If the type of storage system is encoded in the video data, this is called a manifest type
- If the video data contains the logic to select the type of storage system, message passing is used

Slide 1.5. 6

## Video table of operators

| | Type | |
|---|---|---|
| Operator | VHS | V2000 |
| load | load-VHS | load-V2000 |
| save | save-VHS | save-V2000 |

- The table of operators is used (directly or indirectly) by all programming methods

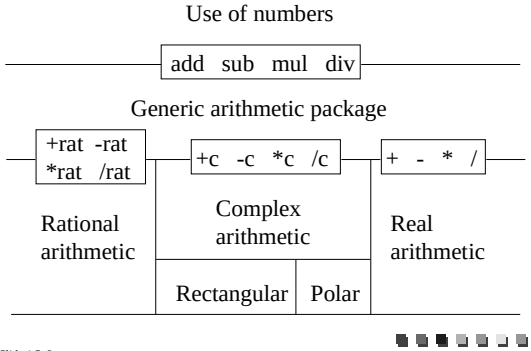
Slide 1.5. 7

## Programming methods

- Procedural
  - type knowledge (usually) within procedure
  - row dispatch on type
- Message passing
  - operator knowledge within object
  - column dispatch on operator
- Data directed
  - table driven dispatch, using both type and operator: (perform load VHS-tape)

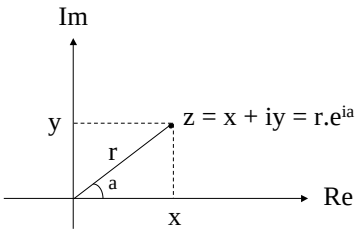
Slide 1.5. 8

## Example: Living by numbers




Slide 1.5. 9

## Complex numbers




Rectangular representation: (x,y)
Polar representations: (r,a)

Slide 1.5. 10

## Complex numbers

Conversion rules: $r = \sqrt{x^2 + y^2}$

$x = r \cos a$

$y = r \sin a$

$a = \arctan \frac{y}{x}$

Note that the scheme (atan y x) function takes signs of y and x into account and returns angle in right quadrant

Slide 1.5. 11

## Complex numbers, constructors/ selectors

- Constructors

  ( make-rectangular <real> <imag> )
  ( make-polar <length> <angle> )

- Selectors ( real-part z )
  ( imag-part z )
  ( magnitude z )
  ( angle z )

  ( make-rectangle (real-part z ) ( imag-part z ))
  =
  (make-polar (magnitude z ) ( angle z ))
  =
  z

Slide 1.5. 12

## Complex number, single rectangular representation

```
(define (make-rectangular x y) (cons x y))

(define (real-part  z) (car  z))

(define (imag-part z) (cdr z))

(define (make-polar  r  a)
  (cons (*  r  (cos  a)) (*  r  (sin  a))))

(define ( magnitude z)
  (sqrt (+ square (car z))(square (cdr z)))))

(define (angle z)
  (atan (cdr  z) (car z)))
```



## Complex numbers, using single rectangular representation

```
(define (*c  z1 z2 )
  (make-polar
    (* (magnitude z1) (magnitude z2))
       (+ ( angle z1) (angle z2 ))))
```

- Very elegant definition of <*c>
- However: *Very inefficient when complex numbers are represented as <real,imag>*
- #operations:
  2 * sqrt, 4 * square, 1 * cos, 1 * sin, 2 * atan



## Complex numbers, changing to polar representation

```
(define (make-rectangular x y)
  (cons (sqrt (+ (square x) (square y)))
        (atan y x)))
(define (real-part z)
  (* (car z) (cos (cdr z))))
(define (imag-part x)
  (* (car z) (sin (cdr z))))
(define (make-polar r a) (cons r a))
(define (magnitude z) (car z))
(define (angle z) (cdr z))
```

## Complex numbers, changing to polar representation

```
(define (+c z1 z2)
  (make-rectangular
    (+ (real-part z1) (real-part z2))
    (+ (imag-part z1) (imag-part z2))))
```

- Very elegant definition of <+c>
- However, inefficient when complex number are represented in polar notation as <len,angle>
- Check out the #operations !

Slide 1.5. 16

## Complex numbers, manifest type

- One representation is inefficient, use two concurrent representations
- Add to each complex number a <type>
  - called 'tag'
- Constructors:

```
( define (make-rectangular x y )
         (list 'rectangular x y ))

( define (make-polar r a )
         (list 'polar r a ))
```

Slide 1.5. 17

## Complex numbers, manifest type

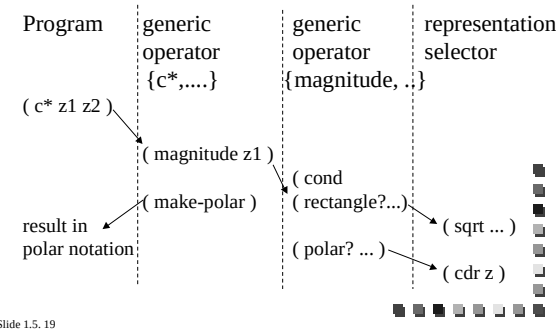
Example generic selector:

```
(define (magnitude z)
  (cond ((rectangle? z)
         (magnitude-rectangular (contents z))
        ((polar? z)
         (magnitude-polar (contents z)))))
```

- The generic operator <*c> does not notice the difference with the single representation
- An appropriate representation can be chosen

Slide 1.5. 18

## Manifest type at work





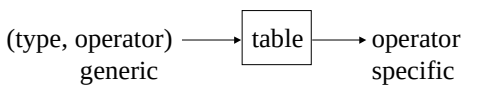

## Procedural programming evaluated

- Observe that procedures like ‘magnitude’ check the type of the object and dispatch on this type to the proper procedure
- Disadvantages:
  - All generic operators (procedures) have to be changed when adding an type or representation of a type
  - possible name conflicts between dispatched procedures



## Data-directed programming

- Table driven
- Explicit selection of representation in the generic selectors makes it hard to extend a package
- Using a data structure called table removes this restriction



# Data-directed programming example

**Types**

| **Operators** | 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 |



# Data-directed programming

- Access table with ‘put’ and ‘get’

```
(put <type> <op> <item>)
(get <type> <op>)        ; returns proc
```

Examples:

```
(put ‘rectangular ‘real-part
     real-part-rectangular)
(put ‘polar ‘real-part real-part-polar)
```



# Data-directed programming

```
(define (operate op obj)
  (let ((proc (get (type obj) op)))    ; get proc from table
    (if (not (null? proc))
        (proc (contents obj))          ; apply procedure
        (error “operator undefined for this type”
               (list op obj) ))))
```

Usage:

```
(define (real-part obj) (operate ‘real-part obj)
(define (angle obj) (operate ‘angle obj)
```

# Message passing

- Send ‘message’ to object, e.g.:
  - `(z ‘real-part)`

```
(define (make-rectangular x y)
  (lambda (mesg)
    (cond ((eq? mesg ‘real-part) x)
          ((eq? mesg ‘imag-part) y)
          ((eq? m ‘magnitude)
              (sqrt (+ (square x)(square y))))
          ((eq? m ‘angle) (atan y x))
          (else (error “unknown message
                       MAKE-RECTANGULAR” m)) )))
```

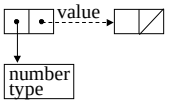
Link with table driven computation:

```
(define (operate op obj) (obj op))
```

Slide 1.5. 25

# Summary

- Complex numbers had two different representations
- Numbers in general have many representations (e.g., number, float, complex)
- A manifest type is used to distinguish between types




- A table is used by the generic operators to select type operator

Slide 1.5. 26

**Mr. Skiem, session 1.5**

**- a. Data and multiple representation**
- binary representation
- decimal representation

**- b. More efficient, using Manifest Types**

Slide 1.5. 27