Les 6: Data-abstractie

SESSIE 1

Les 6: Data-abstractie

Uit de inleiding van vorige les weten we al dat data-abstractie een techniek is om die delen van een programma die iets vertellen over hoe en waaruit complexe data-objecten zijn samengesteld te isoleren van de andere delen van het programma die die complexe objecten manipuleren als een geheel. Data-abstractie maakt het ontwerpen, onderhouden en aanpassen van een groot complex programma gemakkelijker.

Overzicht

We introduceren data-abstractie met een zeer eenvoudig voorbeeld: breuken.

Daarna wordt data-abstractie voor het ons zeer bekend concept van een wiskundige verzameling op verschillende manieren uitgewerkt.

We introduceren tussendoor symbolische data en het concept quotatie om wat rijkere voorbeelden dan alleen getallen aan te kunnen.

Data-abstractie:
Breuken als CONS cellen

```
(define (make-rat a b)
  (cons a b))

(define (numer c)
  (car c))

(define (denom c)
  (cdr c))
```

constructor

selectors of accessors

```
> (make-rat 1 2)
(1 . 2)
> (define p (make-rat 2 3))
> p
(2 . 3)
> (numer p)
2
> (denom p)
3
```

Data-abstractie barrières

Programs that use rational numbers

Rational numbers in problem domain

+rat -rat *rat /rat =rat

Rules for rational-number arithmetic

make-rat numer denom

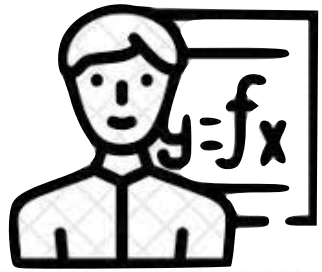
Rational number as pair

cons car cdr

However pairs are implemented

Gebruikt

Ondersteunt



Operaties op breuken

```
(define (rat+ p q)
  (make-rat
    (+ (* (numer p) (denom q))
       (* (numer q) (denom p)))
    (* (denom p) (denom q))))
```

```
(define (rat- p q)
  (make-rat
    (- (* (numer p) (denom q))
       (* (numer q) (denom p)))
    (* (denom p) (denom q))))
```

```
(define (rat* p q)
  (make-rat
    (* (numer p) (numer q))
    (* (denom p) (denom q))))
```

```
(define (rat/ p q)
  (make-rat
    (* (numer p) (numer q))
    (* (denom p) (denom q))))
```

```
> (define p (make-rat 2 3))
> (define q (make-rat 1 4))
> (rat+ p q)
(11 . 12)
> (rat* p q)
(2 . 12)
```

breuk is niet vereenvoudigd

Data-abstractie: Verandering van representatie

```
(define (make-rat2 a b)
  (let ((g (gcd a b)))
    (cons (/ a g) (/ b g))))
```

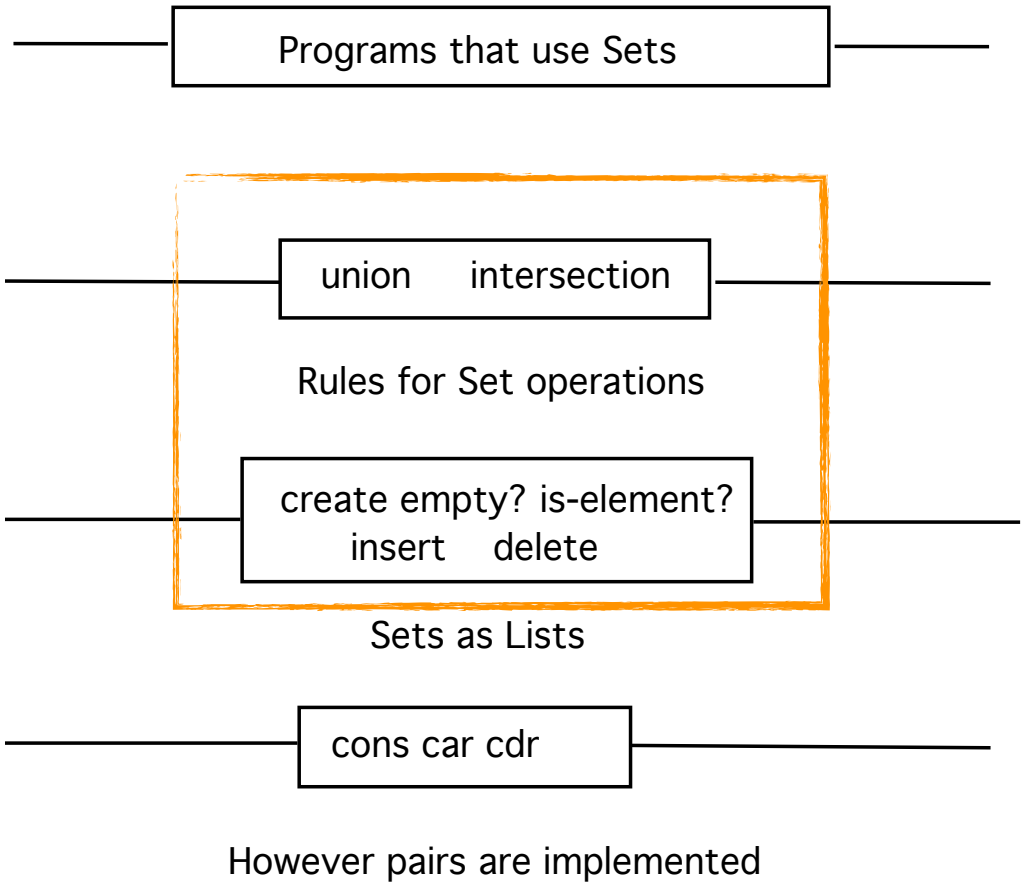
breuken altijd in hun 'vereenvoudigde' vorm opslaan door teller en noemer te delen door hun grootste gemene deler

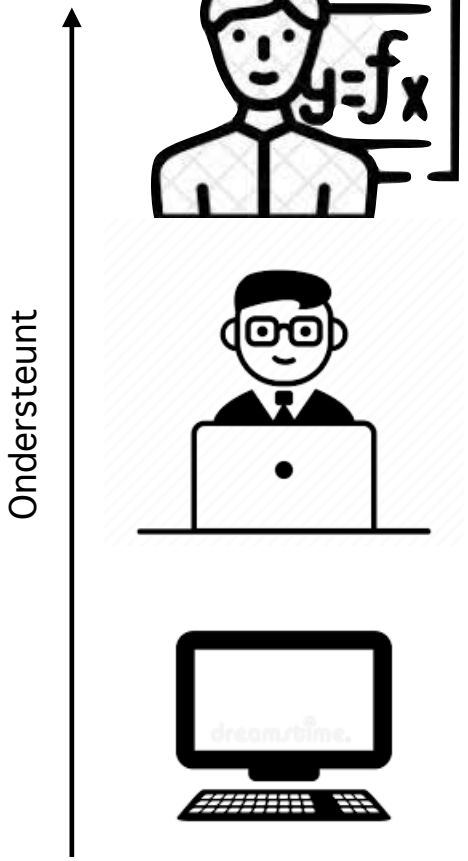
```
> (make-rat 3 12)
(1 . 4)
> (define p (make-rat 2 3))
> (define q (make-rat 3 12))
> (rat+ p q)
(11 . 12)
> (rat* p q)
(1 . 6)
```

operatoren blijven werken ook met deze nieuwe representatie

maar resultaten zijn bij constructie 'vereenvoudigd'

Verzamelingen als ongeordende lijsten





Verzamelingen als ongeordende lijsten

```
(define (create-set) '())

(define (empty? set) (null? set))

(define (element-of? el set)
  (cond
    ((empty? set) #f)
    ((equal? el (car set)) #t)
    (else (element-of? el (cdr set)))))

(define (insert el set)
  (if (element-of? el set)
      set
      (cons el set)))

(define (delete el set)
  (cond
    ((empty? set) set)
    ((equal? el (car set)) (cdr set))
    (else (cons (car set) (delete el (cdr set))))))
```

als een element al in de verzameling zit mag het niet een tweede keer toegevoegd worden

Operaties op verzamelingen

insert zal er voor zorgen dan elementen die in beide lijsten voorkomen niet dubbel in het resultaat terecht komen

hier mag cons gebruikt worden ipv insert omdat er geen dubbels in de input zitten en dus ook niet in de doorsnede

```
(define (union set1 set2)
  (if (empty? set1)
      set2
      (insert (car set1)
              (union (cdr set1) set2))))

(define (intersection set1 set2)
  (cond
    ((or (empty? set1) (empty? set2))
     (create-set))
    ((element-of? (car set1) set2)
     (cons (car set1)
           (intersection (cdr set1) set2)))
    (else (intersection (cdr set1) set2))))
```

Verzamelingen aan het werk

```
> (define set1 (insert 5 (insert 2 (create-set))))
> set1
(5 2)
> (define set2 (insert 2 (insert 4 (insert 3 (create-set)))))
> set2
(2 4 3)
> (delete 3 set2)
(2 4)
> set2
(2 4 3)
> (insert 7 set2)
(7 2 4 3)
> set2
(2 4 3)
> (union set1 set2)
(5 2 4 3)
> (intersection set1 set2)
(2)
```

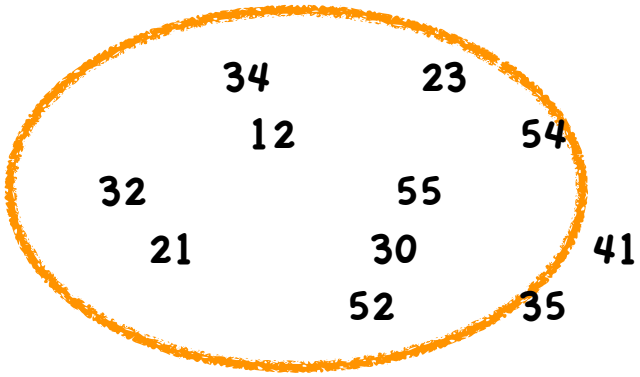
de operatoren over sets zijn functies!

```
> set1
(5 2)
> set2
(2 4 3)
```

Les 6: Data-abstractie

SESSIE 2

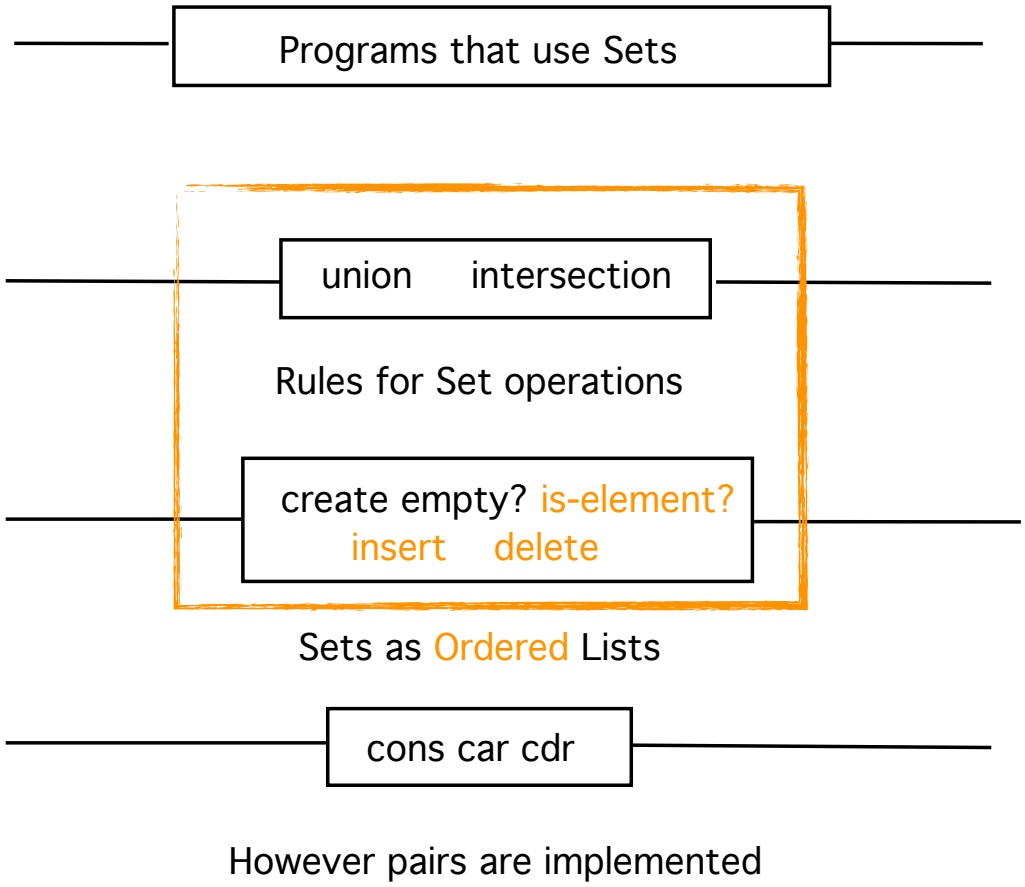
Verzamelingen als geordende lijsten - motivatie

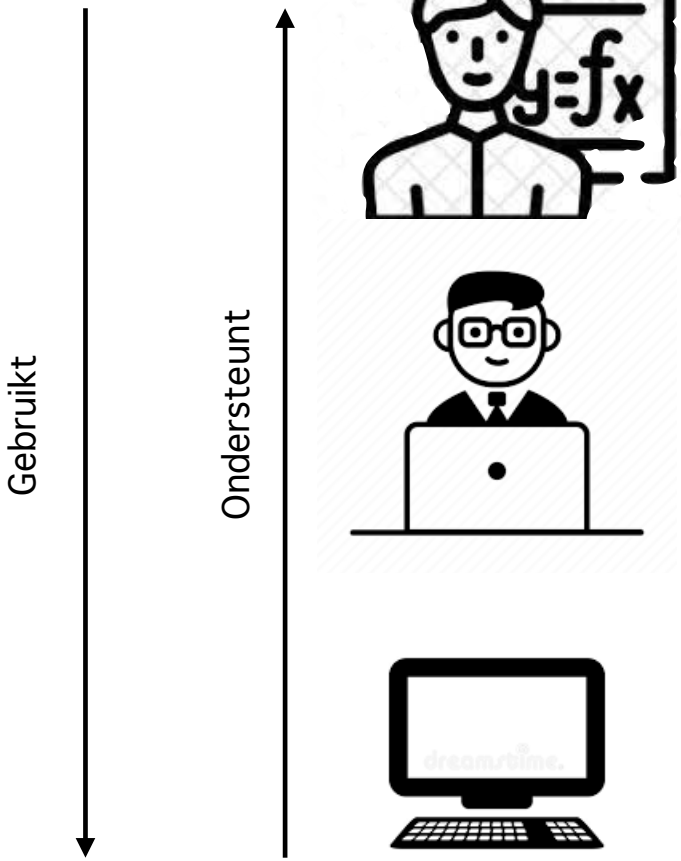


34 23 12 54 32 55 43 21 30 41 52 35

12 21 23 30 32 34 35 41 43 52 54 55

Verzamelingen als geordende lijsten





Verzamelingen als geordende lijsten (1)

zorg dat een element dat wordt toegevoegd op de juiste plaats terecht komt

```
(define (insert el set)
  (cond
    ((empty? set) (list el))
    ((= el (car set)) set)
    ((< el (car set)) (cons el set))
    (else (cons (car set)
                (insert el (cdr set))))))
```

```
(define (insert el set)
  (if (element-of? el set)
      set
      (cons el set)))
```

```
> (define set1 (insert 5 (insert 2 (create-set))))
> set1
(2 5)
> (define set2 (insert 2 (insert 4 (insert 3 (create-set)))))
> set2
(2 3 4)
```

geordende lijsten

Verzamelingen als geordende lijsten (2)

```
(define (element-of? el set)
  (cond
    ((empty? set) #f)
    ((= el (car set)) #t)
    ((< el (car set)) #f)
    (else (element-of? el (cdr set)))))

(define (delete el set)
  (cond
    ((empty? set) set)
    ((= el (car set)) (cdr set))
    ((< el (car set)) set)
    (else (cons (car set)
                (delete el (cdr set))))))
```

zoeken in een geordende lijst kan efficiënter

ook delete kan nu efficiënter

Verzamelingen als geordende lijsten aan het werk

```
> (define set1 (insert 5 (insert 2 (create-set))))
> set1
(2 5)
> (define set2 (insert 2 (insert 4 (insert 3 (create-set)))))
> set2
(2 3 4)
> (union set1 set2)
(2 3 4 5)
> (intersection set1 set2)
(2)
```

Efficiëntiewinst voor INTERSECTION mogelijk

```
(define (intersection set1 set2)
  (cond
    ((or (empty? set1) (empty? set2))
     (create-set))
    ((= (car set1) (car set2))
     (cons (car set1)
       (intersection (cdr set1) (cdr set2))))
    ((< (car set1) (car set2))
     (intersection (cdr set1) set2))
    (else
     (intersection set1 (cdr set2)))))
```

2 3 4 5 6 8 9
2 4 7 8

Ook efficiëntiewinst voor UNION mogelijk

```
(define (union set1 set2)
  (cond
    ((empty? set1) set2)
    ((empty? set2) set1)
    ((= (car set1) (car set2))
     (cons (car set1)
        (union (cdr set1) (cdr set2))))
    ((< (car set1) (car set2))
     (cons (car set1)
         (union (cdr set1) set2)))
    (else
      (cons (car set2)
         (union set1 (cdr set2))))))
```

1 3 4 6 8 9
2 3 5 7 8

Quoting

```
> (1 2 3)
⊗⊗ application: not a procedure;
expected a procedure that can be applied to arguments
given: 1
arguments...:
2
3
> '(1 2 3)
(1 2 3)
> (quote (1 2 3))
(1 2 3)
> ()
⊗⊗ #%app: missing procedure expression;
probably originally (), which is an illegal empty application in:
(#%app)
> '()
()
> (quote ())
()
```

Quoting and Symbols

```
> a
⊗⊗ a: undefined;
cannot reference undefined identifier
> 'a
a
> (quote a)
a

> (define a 5)
> a
5
> 'a
a
```

```
> 'viviane
viviane
> (list 'olga 'wolf 'viviane)
(olga wolf viviane)
> '(olga wolf viviane)
(olga wolf viviane)

> (define viviane 'goed)
> viviane
goed
> 'viviane
viviane
```

Gelijkheid (=, eq?, equal?)

```
> (= 5 5)
#t
> (= 5 5.0)
#t
> (= 5 7)
#f
> (eq? 5 5)
#t
> (eq? 5 5.0)
#f
```

```
> (eq? 'a 'a)
#t
> (eq? 'a 'b)
#f
> (eq? #t #t)
#t
> (eq? #t #f)
#f
> (eq? '() '())
#t
```

```
> (eq? (cons 1 2) (cons 1 2))
#f
> (eq? '(1 2 3) '(1 2 3))
#f
> (equal? (cons 1 2) (cons 1 2))
#t
> (equal? '(1 2 3) '(1 2 3))
#t
```

Lidmaatschap (is geen predicaat)

```
> (member 'apple '(pear apple banana ananas))
(apple banana ananas)

> (member 'orange '(pear apple banana ananas))
#f

> (member (list 3 4) '((1 2) (2 3) (3 4) (4 5)))
((3 4) (4 5))

> (memq (list 3 4) '((1 2) (2 3) (3 4) (4 5)))
#f
```

Associatielijsten

```
> (define alist '((an 123) (ben 248) (cie 300)))
> alist
((an 123) (ben 248) (cie 300))
> (assq 'ben alist)
(ben 248)
> (assq 'viv alist)
#f

> (define alist '(((an v) 123) ((ben m) 248) ((cie m) 300)))
> alist
(((an v) 123) ((ben m) 248) ((cie m) 300))
> (assq '(cie m) alist)
#f
> (assoc '(cie m) alist)
((cie m) 300)
```

Les 7: Bomen als abstracte datatypes