

7 Datenstrukturen mit Varianten

- Jeder Wert von Scheme (`scheme-value`) trägt seinen Typ mit sich.
- Der Typ kann mit den Typprädikaten (vom Typ `scheme-value` -> `boolean`) getestet werden.

`number?` ; #t gdw Argument ist Zahl

`boolean?` ; #t gdw Argument ist #t oder #f

`struct?` ; #t gdw Argument ist durch `define-struct` definiert

- Der Typtest kann Teil der Funktionsdefinition werden:

⇒ Neue Art von Definition für einen Datentyp: *Varianten*

7.1 Beispiel: Zahlen mit $+/-$ unendlich

Definition Ein Wert vom Typ `number*` ist entweder

1. vom Typ `number` oder
2. vom Typ `symbol` und ist dann entweder `+infty` (positiv unendlich) oder `-infty` (negativ unendlich).

Schreibweise: `number*` = `number` + `symbol`

7.2 Nachfolgeroperation auf `number*`

Signatur:

`number*-succ` : `number* -> number*`

Erklärung: Nachfolgeroperation für `number*`; der Nachfolger von \pm unendlich ist jeweils \pm unendlich.

Beispiel:

```
(number*-succ '+infty)      ; => '+infty
(number*-succ 10)          ; => 11
(number*-succ '-infty)     ; => '-infty
```

Definition:

```
(define number*-succ
  (lambda (x)
    (cond
      ((number? x) (+ x 1))
      ((symbol? x) x))))
```

7.3 Typische Musterdefinition

Falls die Variante $v = t_1 + t_2 + \dots + t_n$ definiert ist, so ist eine typische Musterdefinition

```
(define f
  (lambda (v)
    (cond ((t1? v) e1)
          ((t2? v) e2)
          ⋮
          ((tn? v) en))))
```

- In e_i wird der Fall behandelt, dass v vom Typ t_i ist.

7.4 Beispiel: Geometrische Figuren

Definition Eine `line` ist eine Struktur `(make-line p1 p2)`,
wobei `p1, p2 : posn`.

Repräsentiert eine Gerade von `p1` nach `p1 + p2`.

`(define-struct line (origin extent))`

Definition Ein `shape` ist entweder

1. eine Struktur `(make-circle p n)` mit `p : posn` und `n : number` oder
2. eine Struktur `(make-rectangle p1 p2)` mit `p1, p2 : posn` oder
3. eine Struktur `(make-line p1 p2)` mit `p1, p2 : posn`

D.h. `shape = circle + rectangle + line`

7.5 Fläche einer geometrischen Figur

Signatur:

`shape-area` : `shape` -> `number`

Erklärung: (`shape-area` `s`) berechnet den Flächeninhalt von `s`

Beispiele: siehe `circle-area`, `rectangle-area`

Definition:

```
(define shape-area
  (lambda (s)
    (cond
      ((circle? s)
       (circle-area s))
      ((rectangle? s)
       (rectangle-area s))
      ((line? s)
       (line-area s))))))
```

7.6 Berechnungsprozess mit shape-area

```
(shape-area (make-circle (make-posn 0 0) 10))
```

=>

```
(cond  
  ((circle? (make-circle (make-posn 0 0) 10))  
   (circle-area (make-circle (make-posn 0 0) 10)))  
  ((rectangle? (make-circle (make-posn 0 0) 10))  
   (rectangle-area (make-circle (make-posn 0 0) 10)))  
  ((line? (make-circle (make-posn 0 0) 10))  
   (line-area (make-circle (make-posn 0 0) 10))))
```

7.7 Berechnungsprozess mit shape-area (Forts.)

=>

```
(cond
  (#t
   (circle-area (make-circle (make-posn 0 0) 10)))
  ((rectangle? (make-circle (make-posn 0 0) 10))
   (rectangle-area (make-circle (make-posn 0 0) 10)))
  ((line? (make-circle (make-posn 0 0) 10))
   (line-area (make-circle (make-posn 0 0) 10))))
```

=>

```
(circle-area (make-circle (make-posn 0 0) 10))
```

=>

```
(* pi (square (circle-radius (make-circle (make-posn 0 0) 10))))
```

=>

```
314.1592653589793
```

7.8 Zusammenfassung: Variante Datenstrukturen

- Informelle Definition $v = t_1 + t_2 + \dots + t_n$ mit Erklärung
- Ausnutzung der Typinformation (Typprädikate)
- Definitionsmuster

```
(define f
  (lambda (v)
    (cond ((t1? v) e1)
          ((t2? v) e2)
          :
          ((tn? v) en))))
```