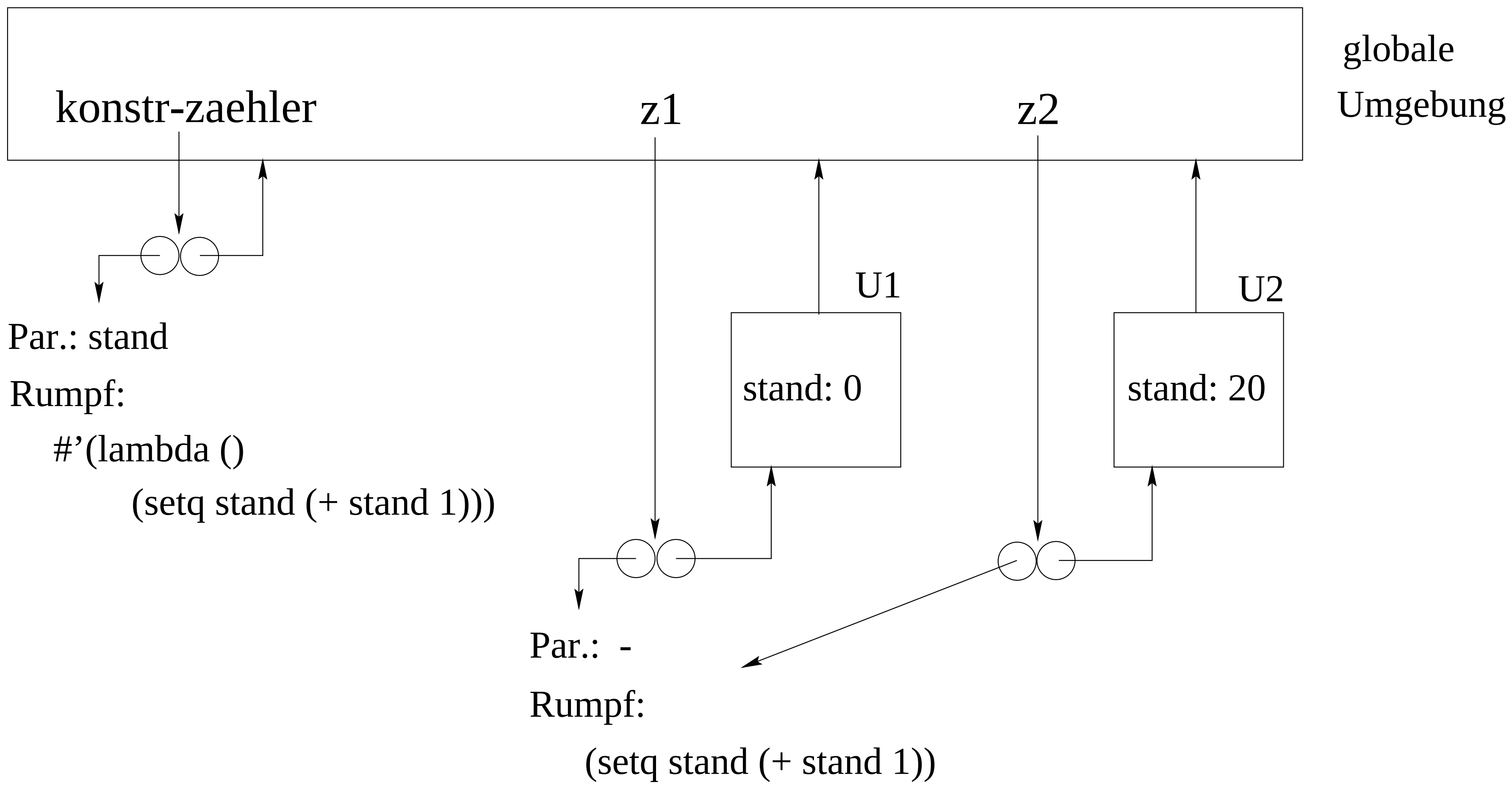
Lokale Zustandsobjekte mit Closures

```
(defun konstr-zaehler (stand)
  #'(lambda () (setq stand (+ stand 1))))
```

```
> (setq z1 (konstr-zaehler 0))
#<Interpreted Closure (:internal konstr-zaehler) @ #x449662>
> (setq z2 (konstr-zaehler 20))
#<Interpreted Closure (:internal konstr-zaehler) @ #x44b6d2>
> (funcall z1)
1
> (funcall z1)
2
> (funcall z1)
3
> (funcall z2)
21
```

Umgebungsstrukturen für Zählerbeispiel



Prinzip der Nachrichtenweitergabe

```
(defun konstr-aufadd ()
  (let ((stand 0))
    (labels ((reset () (setq stand 0))
             (add (wert) (setq stand (+ stand wert)))
             (zuteilen (nachricht &rest parameter)
               (case nachricht
                 (reset (apply #'reset parameter))
                 (add (apply #'add parameter))
                 (otherwise 'unbekannte-nachricht))))
      #'zuteilen)))
```

```
(defun reset (aufaddierer)
  (funcall aufaddierer 'reset))

(defun add (aufaddierer wert)
  (funcall aufaddierer 'add wert))
```

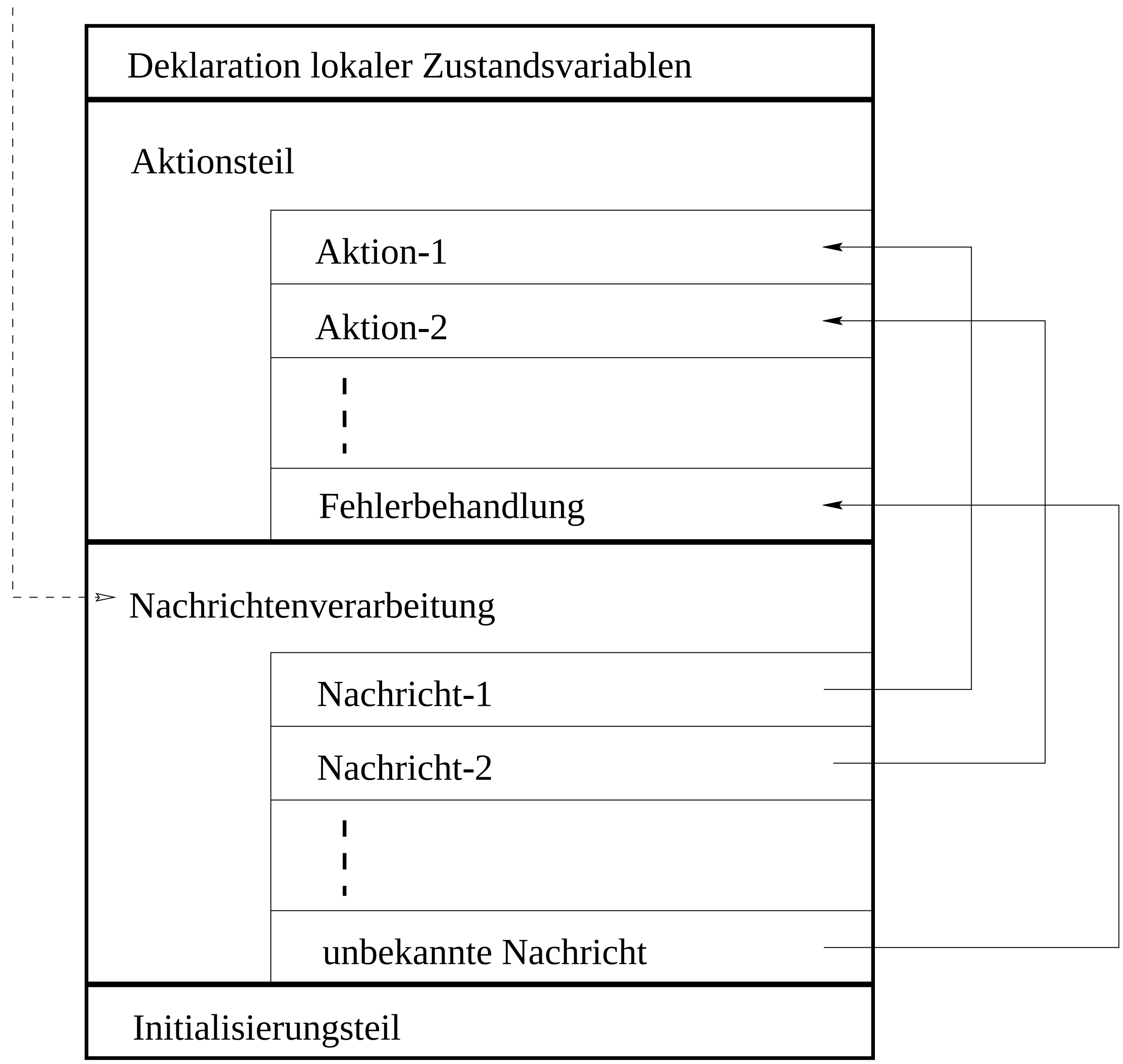
Nachrichtenweitergabe – Beispiel

```
> (setq a1 (konstr-aufadd))
#<Interpreted Closure (labels konstr-aufadd zuteilen) @ #x459432>
> (setq a2 (konstr-aufadd))
#<Interpreted Closure (labels konstr-aufadd zuteilen) @ #x45ce4a>

> (add a1 3)
3
> (add a2 7)
7
> (add a1 2)
5
> (reset a1)
0
> (add a2 3)
10
```

Nachrichtenweitergabe
Aufbau der Objekte mit lokalem Zustand

Objekt mit lokalem Zustand



Constraints–I

traditionelle Programmorganisation:

- Berechnungen in einer Richtung
- Bearbeitung von Argumenten $\Rightarrow$ Ergebnisse

Aber:

Modelle von Systemen werden oft als Relationen zwischen den einzelnen Größen angegeben.

Beispiel (aus der Mechanik):

$$\mathbf{d\ A\ E = F\ L}$$

d: Krümmung eines Metallstabs

A: Querschnittsfläche des Stabes

E: Elastizitätsfaktor

F: auf den Stab wirkende Kraft

L: Länge des Stabs

Constraints–II

Lösung bei traditioneller Vorgehensweise:

Implementierung von 5 Berechnungsvorschriften, je nachdem welche 4 Größen gegeben sind (5 Prozeduren oder Fallunterscheidung)

obwohl

eigentlich nur **eine** Berechnungsvorschrift vorliegt.

Abhilfe:

Implementierung als *Beschränkungssystem* bzw. *–netz* (**Constraints**).

Grundbausteine von Beschränkungsnetzen:

- *elementare Beschränkungen*
  legen fest, daß zwischen einzelnen Größen bestimmte Relationen gelten.

  z.B. $a + b = c$ für eine Additionseinheit

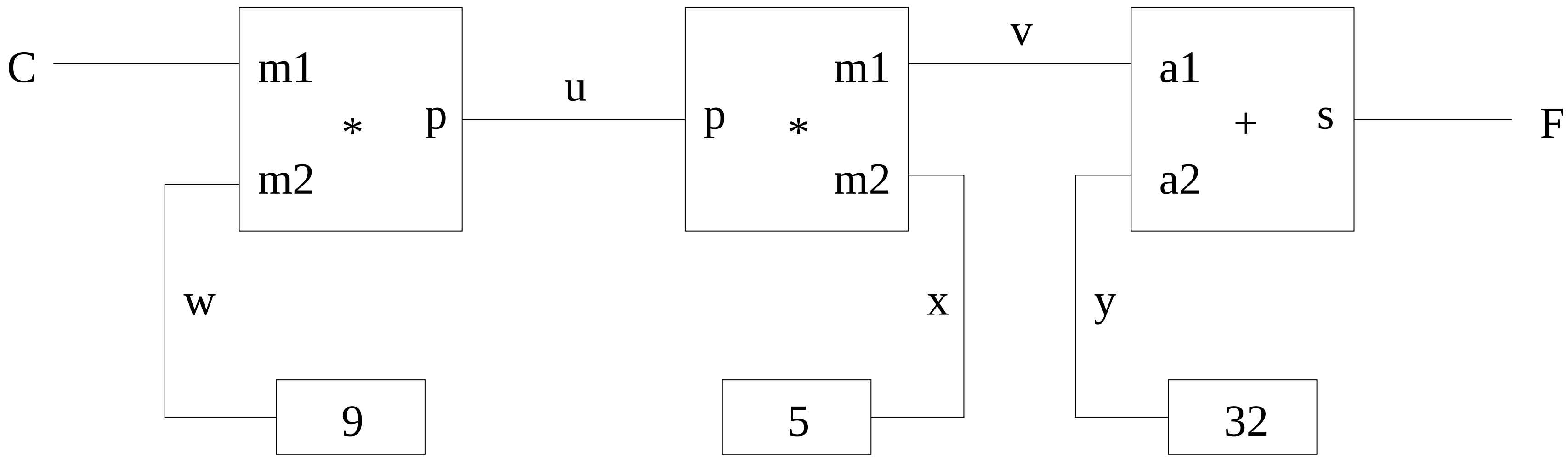
- *Konnektoren*
  verbinden (elementare) Beschränkungseinheiten,
  leiten Werte weiter

Beschränkungsnetze – Beispiel

Die Formel

$$9C = 5(F - 32)$$

zur Umrechnung zwischen Celsius und Fahrenheit als Beschränkungsnetz:



Berechnungsvorgang in einem Beschränkungsnetz

Verhalten der Konnektoren:

Bei Erhalt eines Werts (von einer Beschränkung oder von außen):

- wecke alle angeschlossenen Beschränkungen (außer der Beschränkung, von der der Wert erhalten wurde)
- und informiere sie über den neuen Wert

Verhalten der Beschränkungen:

Prüfe durch Abfragen aller angeschlossenen Konnektoren, ob die Information ausreicht, um einen neuen Wert für einen Konnektor zu bestimmen.

Falls ja: setze den Konnektor auf diesen Wert.

Falls der Konnektor bereits einen Wert hat, dann prüfe auf Widerspruch.

Aufbau des Beschränkungsnetzes

Vorhanden seien Konstruktoren für Konnektoren und elementare Beschränkungen.

Erzeuge Konnektoren C und F und binde sie in ein geeignetes Netz:

```
(setq C (konstr-konnektor))
(setq F (konstr-konnektor))
(konstr-celsius-fahrenheit-konverter C F)
```

mit folgender Konstruktionsfunktion für den Konverter:

```
(defun konstr-celsius-fahrenheit-konverter (c f)
  (let ((u (konstr-konnektor))
        (v (konstr-konnektor))
        (w (konstr-konnektor))
        (x (konstr-konnektor))
        (y (konstr-konnektor)))
    (konstr-multiplikator c w u)
    (konstr-multiplikator v x u)
    (konstr-addierer v y f)
    (konstr-konstante 9 w)
    (konstr-konstante 5 x)
    (konstr-konstante 32 y)))
```

Verwendung des Beschränkungsnetzes

Einsatz von Sonden zum "Abhören" ausgewählter Konnektoren:

```
(sonde "Celsius-Temp." C)
(sonde "Fahrenheit-Temp." F)
```

die Sonden melden neue Werte der betreffenden Konnektoren.

Beispielablauf:

```
> (set-wert! C 25 'benutzer)
Sonde: Celsisus-Temp = 25
Sonde: Fahrenheit-Temp = 77
fertig

> (set-wert! F 212 'benutzer)
Error: Widerspruch (77 212)

> (vergiss-wert! C 'benutzer)
Sonde: Celsisus-Temp = ?
Sonde: Fahrenheit-Temp = ?
fertig

> (set-wert! F 212 'benutzer)
Sonde: Fahrenheit-Temp = 212
Sonde: Celsisus-Temp = 100
fertig
```

Implementierung–I

Grundoperationen für Konnektoren:

`(hat-wert? <konnektor>)` prüft Konnektor auf Wert

`(get-wert <konnektor>)` liefert aktuellen Konnektorwert

`(set-wert! <konnektor> <neuer-wert> <informant>)` teilt Konnektor mit, daß Informant (Beschränkung) Wert neu setzen möchte

`(vergiss-wert! <konnektor> <rückziehender>)` teilt Konnektor mit, daß "Rückziehender" ihn seinen Wert vergessen lassen möchte

`(verbinde <konnektor> <beschränkung>)` verbinde Konnektor mit Beschränkung

Kommunikation der Konnektoren mit Beschränkungen:

`(informiere-ueber-wert <beschränkung>)` teile Beschränkung mit, daß Konnektor einen Wert hat

`(informiere-ueber-kein-wert <beschränkung>)` teile Beschränkung mit, daß Konnektor seinen Wert verloren hat

Implementierung–II

Addiererbeschränkung:

```
(defun konstr-addierer (a1 a2 summe)
  (labels ((verarbeite-neuen-wert ()
             ...      ;; versucht neuen Wert von unbesetzten Konnektoren
                 )    ;; zu bestimmen
           (verarbeite-vergiss-wert ()
             ...      ;; teilt angeschlossenen Konnektoren mit, dass
                 )    ;; sie ihren Wert vergessen sollen

           (ich (nachricht &rest par)
             (case nachricht
               (ich-habe-einen-wert (apply #'verarbeite-neuen-wert par))
               (ich-verlor-meinen-wert (apply #'verarbeite-vergiss-wert par))
               (otherwise
                    (error "Unbekannte Nachricht ~A -- ADDIERER" nachricht)))))

    (verbinde a1 #'ich)     ;; Konnektoren anschliessen
    (verbinde a2 #'ich)
    (verbinde summe #'ich)
    #'ich))                 ;; liefert Objekt mit lokalem Zustand (Closure)
```

Implementierung–III

Implementierung der lokalen Addiererfunktionen:

```
(verarbeite-neuen-wert ()
   (cond ((and (hat-wert? a1) (hat-wert? a2))
             (set-wert! summe
                        (+ (get-wert a1) (get-wert a2))
                        #'ich))
         ((and (hat-wert? a1) (hat-wert? summe))
             (set-wert! a2
                        (- (get-wert summe) (get-wert a1))
                        #'ich))
         ((and (hat-wert? a2) (hat-wert? summe))
             (set-wert! a1
                        (- (get-wert summe) (get-wert a2))
                        #'ich))))


(verarbeite-vergiss-wert ()
   (vergiss-wert! summe #'ich)  ;; wirkt nur auf die Konnektoren,
   (vergiss-wert! a1 #'ich)     ;; die "ich" gesetzt hat.
   (vergiss-wert! a2 #'ich)
   (verarbeite-neuen-wert))     ;; berechne neue Werte
```

Implementierung–IV

spezielle Beschränkungen:

```
(defun konstr-konstante (wert konnektor)
  (labels ((ich (nachricht &rest par)
              (error "Unbekannte Nachricht ~A -- KONSTANTE" nachricht)))
    (verbinde konnektor #'ich)
    (set-wert! konnektor wert #'ich)
    #'ich))

(defun sonde (name konnektor)
  (labels ((verarbeite-neuen-wert ()
              (format t "~%Sonde: ~A = ~A" name (get-wert konnektor)))
            (verarbeite-vergiss-wert ()
              (format t "~%Sonde: ~A = ?" name))
            (ich (nachricht &rest par)
              (case nachricht
                (ich-habe-einen-wert (apply #'verarbeite-neuen-wert par))
                (ich-verlor-meinen-wert (apply #'verarbeite-vergiss-wert par))
                (otherwise ... ))))
    (verbinde konnektor #'ich)
    #'ich))
```

Implementierung–V

Implementierung der Konnektoren:

```
(defun konstr-konnektor ()
  (let ((wert nil)
        (informant nil)
        (beschraenkungen '()))
    (labels ((set-mein-wert (neuwert setzender)
               ... )
             (vergiss-mein-wert (rueckziehender)
               ... )
             (verbinde (neue-beschraenkung)
               ... )
             (ich (nachricht &rest par)
               (case nachricht
                 (hat-wert? (not (null informant)))
                 (wert wert)
                 (set-wert! (apply #'set-mein-wert par))
                 (vergiss (apply #'vergiss-mein-wert par))
                 (verbinde (apply #'verbinde par))
                 (otherwise ... ))))
      #'ich)))
```

Implementierung–VI

Implementierung der lokalen Konnektorfunktionen:

```
(set-mein-wert (neuwert setzender)
  (cond ((not (hat-wert? #'ich))
           (setq wert neuwert)
           (setq informant setzender)
           (fuer-jeden-ausser setzender
                              #'informiere-ueber-wert
                              beschraenkungen))
        ((not (= wert neuwert))
           (error "Widerspruch ~A" (list wert neuwert)))))

(vergiss-mein-wert (rueckziehender)
  (when (equal rueckziehender informant)
    (setq informant nil)
    (fuer-jeden-ausser rueckziehender
                       #'informiere-ueber-kein-wert
                       beschraenkungen)))
```

Implementierung–VII

Implementierung der lokalen Konnektorfunktionen:

```
(verbinde (neue-beschraenkung)
  (when (not (member neue-beschraenkung beschraenkungen
                     :test #'equal))
    (setq beschraenkungen (cons neue-beschraenkung beschraenkungen)))
  (when (hat-wert? #'ich)
    (informiere-ueber-wert neue-beschraenkung)))
```

Hilfsfunktion:

```
(defun fuer-jeden-ausser (ausnahme prozedur liste)
  (dolist (element liste 'fertig)
    (when (not (equal element ausnahme))
      (funcall prozedur element))))
```

Implementierung–VIII

Schnittstellen zu Konnektoren:

```
(defun hat-wert? (konnektor)
  (funcall konnektor 'hat-wert?))

(defun get-wert (konnektor)
  (funcall konnektor 'wert))

(defun vergiss-wert! (konnektor rueckziehender)
  (funcall konnektor 'vergiss rueckziehender))

(defun set-wert! (konnektor neuer-wert informant)
  (funcall konnektor 'set-wert! neuer-wert informant))

(defun verbinde (konnektor neue-beschraenkung)
  (funcall konnektor 'verbinde neue-beschraenkung))
```

Schnittstellen zu Beschränkungen:

```
(defun informiere-ueber-wert (beschraenkung)
  (funcall beschraenkung 'ich-habe-einen-wert))

(defun informiere-ueber-kein-wert (beschraenkung)
  (funcall beschraenkung 'ich-verlor-meinen-wert))
```