

## Addierer-Beschränkung

```
(define (addierer a1 a2 summe)
  (local
    ((define (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))
            (else 'fertig)))

      (define (verarbeite-vergiss-wert)
        (begin (vergiss-wert! summe ich)
                (vergiss-wert! a1 ich)
                (vergiss-wert! a2 ich)
                (verarbeite-neuen-wert))) ; wegen moeglicher anderer Werte
```

1/8

```
(define (ich aufforderung)
  (cond ((eq? aufforderung 'ich-habe-einen-wert)
         verarbeite-neuen-wert)
        ((eq? aufforderung 'ich-verlor-meinen-wert)
         verarbeite-vergiss-wert)
        (else (error 'addierer "Unbekannte Aufforderung"))))

(begin (verbinde a1 ich)
        (verbinde a2 ich)
        (verbinde summe ich)))
```

## Schnittstelle von Beschränkungen

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

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

2/8

## Multiplikator-Beschränkung

```
(define (multiplikator m1 m2 produkt)
  (local
    ((define (verarbeite-neuen-wert)
      (cond ((or (if (hat-wert? m1) (= (get-wert m1) 0) false)
                 (if (hat-wert? m2) (= (get-wert m2) 0) false))
             (set-wert! produkt 0 ich))
            ((and (hat-wert? m1) (hat-wert? m2))
             (set-wert! produkt (* (get-wert m1) (get-wert m2)) ich))
            ((and (hat-wert? produkt) (hat-wert? m1))
             (set-wert! m2 (/ (get-wert produkt) (get-wert m1)) ich))
            ((and (hat-wert? produkt) (hat-wert? m2))
             (set-wert! m1 (/ (get-wert produkt) (get-wert m2)) ich))
            (else 'fertig)))

      (define (verarbeite-vergiss-wert)
        (begin (vergiss-wert! produkt ich)
                (vergiss-wert! m1 ich) (vergiss-wert! m2 ich)
                (verarbeite-neuen-wert)))
```

3/8

```
(define (ich aufforderung)
  (cond ((eq? aufforderung 'ich-habe-einen-wert)
         verarbeite-neuen-wert)
        ((eq? aufforderung 'ich-verlor-meinen-wert)
         verarbeite-vergiss-wert)
        (else (error 'multiplikator "Unbekannte Aufforderung"))))

(begin (verbinde m1 ich)
        (verbinde m2 ich)
        (verbinde produkt ich)))
```

## Konstanten-Beschränkung

```
(define (konstante wert konnektor)
  (local
    ((define (ich aufforderung) (error 'konstante "Aufforderung"))
     (begin (verbinde konnektor ich)
            (set-wert! konnektor wert ich))))
```

4/8

## Sonden zur Beobachtung

```
(define (sonde name konektor)
  (local
    ((define (verarbeite-neuen-wert)
      (begin (newline)
              (display "Sonde: ") (display name) (display " = ")
              (display (get-wert konektor))))
      (define (verarbeite-vergiss-wert)
        (begin (newline)
                (display "Sonde: ") (display name) (display " = ")
                (display "?")))
      (define (ich aufforderung)
        (cond ((eq? aufforderung 'ich-habe-einen-wert)
              verarbeite-neuen-wert)
              ((eq? aufforderung 'ich-verlor-meinen-wert)
              verarbeite-vergiss-wert)
              (else (error 'sonde "Unbekannte Forderung")))))
    (verbinde konektor ich)))
```

5 / 8

## Implementierung von Konnektoren

```
(define (konstr-konnektor)
  (local
    ((define wert empty)
      (define informant empty)
      (define beschraenkungen empty)
      (define (set-mein-wert neuwert von)
        (cond ((not (hat-wert? ich))
              (begin (set! wert neuwert) (set! informant von)
                    (fuer-jeden-ausser von informiere-ueber-wert
                                         beschraenkungen)))
              ((not (= wert neuwert)) (error 'konnektor "Widerspruch"))
              (else 'fertig)))
      (define (vergiss-mein-wert von)
        (if (eq? von informant)
            (begin (set! informant empty)
                  (fuer-jeden-ausser von informiere-ueber-kein-wert
                                         beschraenkungen))
            'fertig))
    'fertig))
```

6 / 8

```
(define (verbinde neue-besch)
  (begin
    (if (not (memq neue-besch beschraenkungen))
        ; (memq a l)=false falls a in Liste l nicht enthalten
        (set! beschraenkungen (cons neue-besch beschraenkungen))
        'fertig)
    (if (hat-wert? ich)
        (informiere-ueber-wert neue-besch)
        'fertig)))

(define (ich aufforderung)
  (cond ((eq? aufforderung 'hat-wert?) (not (empty? informant)))
        ((eq? aufforderung 'wert) wert)
        ((eq? aufforderung 'set-wert!) set-mein-wert)
        ((eq? aufforderung 'vergiss) vergiss-mein-wert)
        ((eq? aufforderung 'verbinde) verbinde)
        (else (error 'konnektor "Unbekannte Operation"))))

ich))
```

7 / 8

```
(define (fuer-jeden-ausser ausnahme prozedur liste)
  (local ((define (schleife l)
            (cond ((empty? l) 'fertig)
                  ((eq? (first l) ausnahme) (schleife (rest l)))
                  (else (begin (prozedur (first l))
                                (schleife (rest l)))))))
    (schleife liste)))
```

## Grundoperationen der Konnektoren

```
(define (hat-wert? konektor) (konnektor 'hat-wert?))
(define (get-wert konektor) (konnektor 'wert))
(define (vergiss-wert! konektor von) ((konnektor 'vergiss) von))
(define (set-wert! konektor neuer-wert informant)
  ((konnektor 'set-wert!) neuer-wert informant))
(define (verbinde konektor beschr) ((konnektor 'verbinde) beschr))
```

8 / 8