Addierer-Beschränkung

```
(define (addierer a1 a2 s)
  (define (verarbeite-neuen-wert)
    (cond ((and (hat-wert? a1) (hat-wert? a2))
           (set-wert! s
                      (+ (get-wert a1) (get-wert a2))
                      ich))
          ((and (hat-wert? a1) (hat-wert? s))
           (set-wert! a2
                      (- (get-wert s) (get-wert a1))
                      ich))
          ((and (hat-wert? a2) (hat-wert? s))
           (set-wert! a1
                      (- (get-wert s) (get-wert a2))
                      ich))))

  (define (verarbeite-vergiss-wert)
    (vergiss-wert! s ich)
    (vergiss-wert! a1 ich)
    (vergiss-wert! a2 ich)
    (verarbeite-neuen-wert))

  (define (ich aufforderung)
    (cond ((eq? aufforderung 'ich-habe-einen-wert)
           verarbeite-neuen-wert)
          ((eq? aufforderung 'ich-verlor-meinen-wert)
           verarbeite-vergiss-wert)
```

```
          (else
           (error "Unbekannte Aufforderung"))))

  (verbinde a1 ich)
  (verbinde a2 ich)
  (verbinde s ich)
  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)))
```

Multiplikator-Beschränkung

```
(define (multiplikator m1 m2 p)
  (define (verarbeite-neuen-wert)
    (cond ((or (if (hat-wert? m1)
                   (= (get-wert m1) 0) #f)
               (if (hat-wert? m2)
                   (= (get-wert m2) 0) #f))
           (set-wert! p 0 ich))
          ((and (hat-wert? m1) (hat-wert? m2))
           (set-wert! p
                      (* (get-wert m1) (get-wert m2))
                      ich))
          ((and (hat-wert? p) (hat-wert? m1))
           (set-wert! m2
                      (/ (get-wert p) (get-wert m1))
                      ich))
          ((and (hat-wert? p) (hat-wert? m2))
           (set-wert! m1
                      (/ (get-wert p) (get-wert m2))
                      ich))))

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

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

  (verbinde m1 ich)
  (verbinde m2 ich)
  (verbinde p ich)
  ich)
```

Konstanten-Beschränkung

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

Sonden zur Beobachtung

```
(define (sonde name konnektor)
  (define (verarbeite-neuen-wert)
    (newline)
    (display "Sonde: ")
    (display name)
    (display " = ")
    (display (get-wert konnektor)))

  (define (verarbeite-vergiss-wert)
    (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 "Unbekannte Forderung"))))

  (verbinde konnektor ich)
  ich)
```

Implementierung von Konnektoren

```
(define (konstr-konnektor)
  (let ((wert nil) (informant nil)
        (beschraenkungen '()))
    (define (set-mein-wert neuwert setzender)
      (cond ((not (hat-wert? ich))
             (set! wert neuwert)
             (set! informant setzender)
             (fuer-jeden-ausser
                        setzender
                        informiere-ueber-wert
                        beschraenkungen))
            ((not (= wert neuwert))
             (error "Widerspruch"))))

    (define (vergiss-mein-wert rueckziehender)
      (if (eq? rueckziehender informant)
          (begin (set! informant nil)
                 (fuer-jeden-ausser
                       rueckziehender
                       informiere-ueber-kein-wert
                       beschraenkungen))))
```

```
    (define (verbinde neue-beschr)
      (if (not (memq neue-beschr beschraenkungen))
          (set! beschraenkungen
                (cons neue-beschr beschraenkungen)))
      (if (hat-wert? ich)
          (informiere-ueber-wert neue-beschr)))

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

(define (fuer-jeden-ausser ausnahme prozedur liste)
  (define (schleife l)
    (cond ((null? l) 'fertig)
          ((eq? (car l) ausnahme) (schleife (cdr l)))
          (else (prozedur (car l))
                (schleife (cdr l)))))
  (schleife liste))
```

Grundoperationen der Konnektoren

```
(define (hat-wert? konnektor)
  (konnektor 'hat-wert?))

(define (get-wert konnektor)
  (konnektor 'wert))

(define (vergiss-wert! konnektor rueckziehender)
  ((konnektor 'vergiss) rueckziehender))

(define (set-wert! konnektor neuer-wert informant)
  ((konnektor 'set-wert!) neuer-wert informant))

(define (verbinde konnektor neue-beschraenkung)
  ((konnektor 'verbinde) neue-beschraenkung))
```