

May 15, 06 15:36

condition-locks.scm

Page 1/1

```
;; Condition-Locks

(define-record-type :condition-lock
  (really-make-condition-lock lock condition)
  condition-lock?
  (lock condition-lock-lock)
  (condition condition-lock-condition))

(define (make-condition-lock lock)
  (let ((condition (make-lock)))
    (obtain-lock condition)
    (really-make-condition-lock lock condition)))

(define (wait-condition-lock condition-lock)
  (release-lock (condition-lock-lock condition-lock))
  (obtain-lock (condition-lock-condition condition-lock))
  (obtain-lock (condition-lock-lock condition-lock)))

(define (signal-condition-lock condition-lock)
  (release-lock (condition-lock-condition condition-lock)))
```

May 15, 06 13:05

prodcons-cond-lock.scm

Page

```
;; Producer/Consumer-Puffer mit Condition-Locks

(define-record-type :buffer
  (really-make-buffer data lock available empty)
  buffer?
  (data buffer-data set-buffer-data!)
  (lock buffer-lock)
  (available buffer-available)
  (empty buffer-empty))

(define (make-buffer)
  (let ((lock (make-lock)))
    (really-make-buffer #f
                        lock
                        (make-condition-lock lock)
                        (make-condition-lock lock))))

(define (with-lock lock thunk)
  (obtain-lock lock)
  (let ((value (thunk)))
    (release-lock lock)
    value))

(define (insert buffer value)
  (with-lock (buffer-lock buffer)
    (lambda ()
      (let loop ()
        (cond
         ((buffer-data buffer)
          (wait-condition-lock (buffer-empty buffer))
          (loop))
         (else
          (set-buffer-data! buffer value)
          (signal-condition-lock (buffer-available buffer))))))))

(define (remove buffer)
  (with-lock (buffer-lock buffer)
    (lambda ()
      (let loop ()
        (cond
         ((buffer-data buffer)
          => (lambda (value)
              (set-buffer-data! buffer #f)
              (signal-condition-lock (buffer-empty buffer))
              value))
         (else
          (wait-condition-lock (buffer-available buffer))
          (loop))))))))
```

May 15, 06 13:35

cond-locks-broadcast.scm

Page 1/1

```

;; Condition-Locks mit Broadcast

(define-record-type :condition-lock
  (really-make-condition-lock lock waiting-locks)
  condition-lock?
  (lock condition-lock-lock)
  (waiting-locks condition-lock-waiting-locks
   set-condition-lock-waiting-locks!))

(define (make-condition-lock lock)
  (really-make-condition-lock lock '()))

(define (wait-condition-lock cond-lock)
  (let ((waiting-lock (make-lock)))
    (obtain-lock waiting-lock)
    (set-condition-lock-waiting-locks!
     cond-lock
     (cons waiting-lock
           (condition-lock-waiting-locks cond-lock)))
    (let ((lock (condition-lock-lock cond-lock)))
      (release-lock lock)
      (obtain-lock waiting-lock)
      (obtain-lock lock))))

(define (signal-condition-lock cond-lock)
  (let ((waiting-locks (condition-lock-waiting-locks cond-lock)))
    (if (not (null? waiting-locks))
        (begin
         (set-condition-lock-waiting-locks!
          cond-lock (cdr waiting-locks))
         (release-lock (car waiting-locks))))))

(define (broadcast-condition-lock cond-lock)
  (let ((waiting-locks (condition-lock-waiting-locks cond-lock)))
    (set-condition-lock-waiting-locks! cond-lock '())
    (let loop ((waiting-locks waiting-locks))
      (if (not (null? waiting-locks))
          (begin
           (release-lock (car waiting-locks))
           (loop (cdr waiting-locks)))))))

```

May 15, 06 13:38

prodcons-cond-lock-broadcast.scm

Page

```

;; Producer/Consumer-Puffer mit Lock-Conditions und Broadcasts

(define-record-type :buffer
  (really-make-buffer data lock cond-lock)
  buffer?
  (data buffer-data set-buffer-data!)
  (lock buffer-lock)
  (cond-lock buffer-cond-lock))

(define (make-buffer)
  (let ((lock (make-lock)))
    (really-make-buffer #f
                        lock
                        (make-condition-lock lock))))

(define (with-lock lock thunk)
  (obtain-lock lock)
  (let ((value (thunk)))
    (release-lock lock)
    value))

(define (insert buffer value)
  (with-lock (buffer-lock buffer)
    (lambda ()
      (let loop ()
        (cond
         ((buffer-data buffer)
          (wait-condition-lock (buffer-cond-lock buffer)
                               (loop)))
         (else
          (set-buffer-data! buffer value)
          (broadcast-condition-lock (buffer-cond-lock buffer))))))))

(define (remove buffer)
  (with-lock (buffer-lock buffer)
    (lambda ()
      (let loop ()
        (cond
         ((buffer-data buffer)
          => (lambda (value)
              (set-buffer-data! buffer #f)
              (broadcast-condition-lock (buffer-cond-lock buffer)
                                       value)))
         (else
          (wait-condition-lock (buffer-cond-lock buffer)
                               (loop))))))))

```