Kontrollstruktuurid

- CPS-kujul interpretaatoris on programmi kontrollvoog (ingl. *control flow*) ilmutatud
- Spetsifitseerib täpselt keelekonstruktsioonide väärtustamise järjekorra
- Võimaldab lihtsasti esitada mitte-lokaalseid kontrolloperaatoreid
  - Programmi töö katkestamine
  - “1.-klassi” jätkud

Kontrollstruktuurid

- Protseduur `abort`
  - katkestab avaldise väärtustamise
  - kogu avaldise väärtus on `abort`'i argumendi väärtus
- Näiteid:

```
--> "+ (3, *(abort(4), 5))"
4
--> "+ (3, *(abort(*(2, abort(+(3,4)))), 5))"
7
--> "+ (abort(1), *(abort(*(2, abort(+(3,4)))), 5))"
1
```

Kontrollstruktuurid

- `abort`'iga interpretaator

```
(define-record abort (exp))

(define eval-exp
  (lambda (exp env k)
    (variant-case exp
      (abort (exp)
        (eval-exp exp env final-valcont))
      ...)))

(define final-valcont (make-final-valcont))
```

“1.-klassi” jätkud

- `letcont` süntaks:

  $\langle exp \rangle$ ::= ...

  | `letcont` $\langle var \rangle$ `in` $\langle exp \rangle$ `letcont (var body)`

- Avaldise `letcont` k `in` E väärtustamisel
  - seotakse muutuja k hetkel kehtiva jätkuga ning seda saab alamavaldises E kasutada kui üheargumendilist protseduuri
  - kui alamavaldise E väärtustamisel jätku k ei kasutata, on E väärtus kogu avaldise väärtuseks
  - kui E väärtustamisel kutsutakse k välja argumendiga E', siis on E' väärtus kogu avaldise väärtuseks

“1.-klassi” jätkud

- Näiteid:

```
--> "+ (2, letcont k in *(3, k(4)))"
6
--> "letcont k in
       proc(x) +( k(proc (x) 5), x)"
#(struct:closure (x) .....
--> "(letcont k in
       proc(x) +( k(proc (x) 5), x))
     (25)"
5
```

“1.-klassi” jätkude interpretaator

```
(define-record continuation (cont))

(define eval-exp
  (lambda (exp env k)
    (variant-case exp
      (letcont (var body)
        (eval-exp body
          (extend-env (list var)
                      (list (make-continuation k))
                      env)
          k))
      ...)))
```

“1.-klassi” jätkude interpretaator

```
(define apply-proc
  (lambda (proc args k)
    (variant-case proc
      (prim-proc (prim-op)
        (apply-continuation k
          (apply-prim-op prim-op args)))
      (closure (formals body env)
        (eval-exp body (extend-env formals args env) k))
      (continuation (cont)
        (apply-continuation cont (car args)))
      (else (error "Invalid procedure:" proc)))))
```

`call-with-current-continuation`

- Scheme's on "1.-klassi" jätkudega manipuleerimiseks protseduur `call-with-current-continuation` ehk lühendatult `call/cc`
- Saab argumendina protseduuri, mille argument seotakse "hetkel kehtiva" jätkuga
- `letcont`'iga analoogne konstruktsioon `let/cc` on defineeritud makrona:

$$\texttt{(let/cc}\ k\ E\texttt{)} \quad \Rightarrow \quad \texttt{(call/cc (lambda (}k\texttt{)}\ E\ \texttt{))}$$

call-with-current-continuation

- Näide: listi elementide korrutis

```
(define allprod
  (lambda (lst)
    (call/cc (lambda (exit)
       (letrec
           ((AllP (lambda (lst)
                    (if (null? lst) 1
                        (if (zero? (car lst)) (exit 0)
                            (* (car lst)
                               (AllP (cdr lst))))))))
         (AllP lst))))))
```

rember-upto-last

- Väljastada listi elemendid vasakult kuni etteantud elemendi esimese esinemiseni

```
(define rember-beyond-first (lambda (a lat)
  (cond ((null? lat) '())
        ((eq? (car lat) a) '())
        (else (cons (car lat)
                    (rember-beyond-first a (cdr lat)))))))
```

```
> (rember-beyond-first 'c '(a b c d c e f g))
(a b)
> (rember-beyond-first 'h '(a b c d c e f g))
(a b c d c e f g)
```

rember-upto-last

- Väljastada kõik listi elemendid pärast etteantud elemendi viimast esinemist

```
> (rember-upto-last 'c '(a b c d e f g))
(d e f g)
> (rember-upto-last 'c '(a b c d e c f g))
(f g)
> (rember-upto-last 'h '(a b c d e f g))
(a b c d e f g)
```

- Listide ümberpööramisega definitsioon:

```
(define rember-upto-last
  (lambda (a lat)
    (reverse (rember-beyond-first a (reverse lat)))))
```

rember-upto-last

Rekursiivne definitsioon

```
(define rember-upto-last (lambda (a lat)
  (letrec ((R (lambda (lat)
                (if (null? lat) (cons #f '())
                    (let ((pair (R (cdr lat))))
                      (cond ((car pair) pair)
                            ((eq? (car lat) a)
                             (cons #t (cdr pair)))
                            (else
                             (cons #f (cons (car lat)
                                            (cdr pair))))))))))
    (cdr (R lat)))))
```

rember-upto-last

call/cc'ga definitsioon

```
(define rember-upto-last
  (lambda (a lat)
    (call/cc (lambda (skip)
       (letrec
          ((R (lambda (lat)
                (cond
                  ((null? lat) '())
                  ((eq? (car lat) a) (skip (R (cdr lat))))
                  (else (cons (car lat) (R (cdr lat)))))))))
            (R lat))))))
```

Järgmiseks korraks

- Lugeda läbi EOPL ptk. 9.3
- “Mängida” interpretaatoritega:
  - `loeng21-1.ss` (protseduurse esitusega jätkud)
  - `loeng21-2.ss` (kirjete abil esitatud jätkud)
- Järgmine loeng toimub 7. mail!