Rekursiivsed protseduuurid

- Rekursiivsete protseduuuride BNF:

$$
\begin{array}{rcl}
\langle exp \rangle & ::= & \ldots \\
 & | & \texttt{letrecproc}\ \langle procdecls \rangle\ \texttt{in}\ \langle exp \rangle \\
\langle procdecls \rangle & ::= & \langle procdecl \rangle\ \{\texttt{;}\ \langle procdecl \rangle\}^{*} \\
\langle procdecl \rangle & ::= & \langle var \rangle\ \langle varlist \rangle = \langle exp \rangle
\end{array}
$$

- Rekursiivsete protseduuuride abstraktne süntaks:

```
(define-record letrecproc (procdecls body))
(define-record procdecl (var formals body))
```

Rekursiivsed protseduuurid

- Näited:

```
letrecproc
   fact(n) = if zero(n) then 1
                        else *(n, fact(sub1(n)))
in fact(6)

letrecproc
   even(x) = if zero(x) then 1 else odd(sub1(x));
   odd(x)  = if zero(x) then 0 else even(sub1(x))
in odd(13)
```

Rekursiivsed protseduuurid

- Protseduuri väärtuseks on sulund (*closure*), kus tema definitsioon on “pakendatud” koos kehtiva keskonnaga
- Rekursiivsete protseduuride korral peab see keskond olema tsükliline
- Võimalikud lahendused:
  - Loome algul sulundi ja seejärel muudame keskonna “õigeks” (ei vaatle)
  - Loome algul keskonna ja seejärel asendame seosed “õigeteks” (variant 1)
  - Viivitame sulundi loomisega kuni protseduuri rakendamiseni (variant 2)

Rekursiivsed protseduuurid (variant 1)

```
(define eval-exp
  (lambda (exp env)
    (variant-case exp
      ...
      (letrecproc (procdecls body)
         (let ((names (map procdecl->var procdecls)))
            (let ((new-env (extend-env names
                             (map (lambda (x)
                                    (make-cell '*dummy*))
                                  names)
                             env)))
              ...
```

Rekursiivsed protseduuurid (variant 1)

```
...
(foreach (lambda (procdecl)
   (let ((name    (procdecl->var procdecl))
         (formals (procdecl->formals procdecl))
         (body    (procdecl->body procdecl)))
     (cell-set!
           (apply-env new-env name)
           (make-closure formals body new-env))))
   procdecls)
(eval-exp body new-env))))
...
```

Rekursiivsed protseduuurid (variant 2)

```
(define eval-exp
  (lambda (exp env)
    (variant-case exp
      ...
      (letrecproc (procdecls body)
         (eval-exp body
                   (extend-rec-env procdecls env)))
      ...
```

Rekursiivsed protseduuurid (variant 2)

```
(define-record extended-rec-env (vars vals old-env))

(define extend-rec-env
  (lambda (procdecls env)
    (make-extended-rec-env
      (map procdecl->var procdecls)
      (list->vector
        (map (lambda (procdecl)
               (make-proc (procdecl->formals procdecl)
                          (procdecl->body procdecl)))
             procdecls))
      env)))
```

Rekursiivsed protseduuurid (variant 2)

```
(define apply-env
  (lambda (env var)
    (variant-case env
      (extended-rec-env (vars vals old-env)
        (let ((p (ribassoc var vars vals '*fail*)))
          (if (eq? p '*fail*)
              (apply-env old-env var)
              (make-closure (proc->formals p)
                            (proc->body p)
                            env))))
      ...
```

Rekursiivsed protseduuurid (variant 2)

```
...
(empty-ff ()
  (error "apply-env: no association for var" var))
(extended-ff* (sym-list val-vector ff)
  (let ((val (ribassoc var sym-list
                       val-vector '*fail*)))
    (if (eq? val '*fail*)
        (apply-env ff var)
        val)))
(else (error "Invalid environment" env)))))
```

Dünaamiliselt skoobitud muutujad

- Dünaamilise skoopimise korral viitab muutuja viimasele seosele, mis täitmisajal on talle antud ja on veel kehtiv

```
--> "let a = 3
     in let p = proc (x) +(x,a);
            a = 5
        in *(a, p(2))"
35
--> "let a = 3
     in let p = proc () +(x,a);
            f = proc (x, y) *(p(), y);
            a = 5
        in *(a, f(let a = 2 in a, 1))"
```

Dünaamiliselt skoobitud muutujad

```
(define eval-exp
  (lambda (exp env)
    (variant-case exp
      ...
      (app (rator rands)
        (let ((proc (eval-exp rator env))
              (args (eval-rands rands env)))
          (apply-proc proc args env)))
      ...
      (proc (formals body) exp)
      ...
```

Dünaamiliselt skoobitud muutujad

```
(define apply-proc
  (lambda (proc args current-env)
    (variant-case proc
      (prim-proc (prim-op)
        (apply-prim-op prim-op (map cell-ref args)))
      (proc (formals body)
        (eval-exp body
            (extend-env formals args
                        current-env)))
      (else (error "Invalid procedure:" proc)))))
```

Dünaamiliselt skoobitud muutujad

- Dünaamilise skoopimise plussid:
  - lihtne realiseerida; keskonnana saab kasutada näiteks ühte globaalset magasini (*deep binding*) või iga muutuja jaoks oma magasini (*shallow binding*)
  - rekursioon realisatsioon triviaalne:

```
--> "let fact = proc (n) add1(n)
     in let fact = proc (n)
                     if zero(n)
                     then 1
                     else *(n,(fact(sub1(n))))
      in fact (5)"
120
```

Dünaamiliselt skoobitud muutujad

- Dünaamilise skoopimise miinused:
  - programmid raskesti mõistetavad
  - näiteks võib seotud muutujate ümbernimetamine muuta programmi tähendust:

```
--> "let a = 3
     in let p = proc () +(x,a);
            f = proc (x, a) *(p(), a);
            a = 5
        in *(a, f(let a = 2 in a, 1))"
15
```

Dünaamiline omistamine

- Dünaamilise omistamise BNF ja abstraktne süntaks:

$$\langle exp \rangle \quad ::= \quad \ldots$$
$$\qquad\qquad | \quad \langle var \rangle \texttt{ := } \langle exp \rangle \texttt{ during } \langle exp \rangle$$

- Dünaamilise omistamise BNF ja abstraktne süntaks:

```
(define-record dynassign (var exp body))
```

- Näide:

```
--> "let x = 4
     in let p = proc (y) +(x, y)
        in +(x := 7 during p(1), p(2))"
14
```

Dünaamiline omistamine

```
(define eval-exp
  (lambda (exp env)
    (variant-case exp
      ...
      (dynassign (var exp body)
        (let ((a-cell (apply-env env var))
              (b-cell (make-cell (eval-exp exp env))))
          (cell-swap! a-cell b-cell)
          (let ((value (eval-exp body env)))
            (cell-swap! a-cell b-cell)
            value)))
      ...
```

Järgmiseks korraks

- Lugeda läbi EOPL ptk. 5.6 – 5.8
- “Mängida” interpretaatoritega
  - failis `loeng12-1.ss` olev interpretaator realiseerib rekursiivsed protseduurid (variant 1);
  - failis `loeng12-2.ss` olev interpretaator realiseerib rekursiivsed protseduurid (variant 2);
  - failis `loeng12-3.ss` olev interpretaator realiseerib dünaamilise skoopimise;
  - failis `loeng12-4.ss` olev interpretaator realiseerib dünaamilise omistamise.