Tähistatavad ja väljendatavad väärtused

- Traditsioonilistes keeltes on väljendatavate väärtuste (expressed values) hulk tihti piiratud nende väärtustega, mida saab esitada ühe masinsõnaga (arvud, viidadad jne.)
- Keerulisemad väärtused (protseduurid, andmestruktuurid jne.) on "ainult" tähistatavad (denoted values)

$$
\begin{aligned}
\textit{Expressed Value} &= \textit{Number} \\
\textit{Denoted Value} &= \textit{L-Value} + \textit{Procedure} + \textit{Array} \\
\textit{L-Value} &= \textit{Cell}(\textit{Expressed Value}) \\
&+ \textit{Array Element}(\textit{Expressed Value})
\end{aligned}
$$

Tähistatavad ja väljendatavad väärtused

Eemaldame süntaksist järgmised reeglid:

⟨*exp*⟩ ::= `proc` ⟨*varlist*⟩ ⟨*exp*⟩ `proc (formals body)`

⟨*operator*⟩ ::= `(`⟨*exp*⟩`)`

⟨*array-exp*⟩ ::= `(`⟨*exp*⟩`)`

Lisame uued reeglid:

⟨*exp*⟩ ::= `letproc` ⟨*procdecls*⟩ `in` ⟨*exp*⟩

`letproc (procdecls body)`

| `local` ⟨*decls*⟩ `in` ⟨*exp*⟩

`local (decls body)`

Tähistatavad ja väljendatavad väärtused

Aluseks võtame “call-by-reference” interpretaatori; lisame variandid `letproc` ja `local` jaoks:

```
(letproc (procdecls body)
  (let ((vars (map procdecl->var procdecls))
        (closures (map (lambda (decl)
                         (make-closure
                           (procdecl->formals decl)
                           (procdecl->body decl)
                           env))
                       procdecls)))
    (let ((new-env (extend-env vars closures env)))
      (eval-exp body new-env)))))
```

Tähistatavad ja väljendatavad väärtused

```
(local (decls body)
  (let ((vars (map decl->var decls))
        (exps (map decl->exp decls)))
    (let ((new-env (extend-env vars
                               (map (lambda (exp)
                                      (make-cell
                                        (eval-exp exp env)))
                                    exps)
                               env)))
      (eval-exp body new-env))))
```

Tähistatavad ja väljendatavad väärtused

Operaatori väärtustamine:

```
(define eval-rator
  (lambda (rator env)
    (let ((den-val (apply-env env
                              (varref->var rator))))
      (if (closure? den-val)
          den-val
          (denoted->expressed den-val)))))
```

Tähistatavad ja väljendatavad väärtused

Massiivide väärtustamine:

```
(define eval-array-exp
  (lambda (array-exp env)
    (apply-env env (varref->var array-exp))))
```

Massiivide konstrueerimine:

```
(define do-letarray make-array)
```

Parameetrite edastamine nime kaudu

- Call-by-name — parameetrid edastatakse nime kaudu; s.o. protseduuri parameetriga seotakse tegelikku argumenti esitav “külmutatud avaldis”, mis väärtustatakse siis, kui parameetrit tegelikult kasutatakse
  - vastab lambda-arvutuses normaaljärjekorras reduktsioonijadale
  - kui parameetrit ei kasutata, siis ka “külmutatud avaldist” ei väärtustata
  - kui parameetrit kasutatakse mitu korda, siis väärtustatakse “külmutatud avaldis” iga kord uuesti

Parameetrite edastamine nime kaudu

Kõrvalefektidega keeltes on “call-by-name” mõneti sarnane “call-by-reference” parameetrite ülekandmisega

```
let i = 0 in
let p = proc (x)
          begin
            i := 1; x := 2
          end
in letarray a[2]
in begin
     a[0] := 1; p(a[i]); a[1]
   end
```

“Call-by-name” interpretaator

- Aluseks võtame “call-by-reference” interpretaatori
- Operandide BNF ja abstraktne süntaks (sama mis “call-by-reference”):

⟨*operand*⟩ ::= ⟨*varref*⟩

| ⟨*array-exp*⟩ `[`⟨*exp*⟩`]`

`arrayref (array index)`

| ⟨*exp*⟩

"Call-by-name" interpretaator

- Andmemudel:

$$
\begin{array}{lcl}
\textit{Expressed Value} & = & \textit{Number} + \textit{Procedure} + \textit{Array} \\
\textit{L-Value} & = & \textit{Cell}(\textit{Expressed Value}) \\
 & + & \textit{Array Element}(\textit{Expressed Value}) \\
\textit{Denoted Value} & = & \textit{L-Value} + \textit{Thunk} \\
\textit{Thunk} & = & () \to \textit{L-Value}
\end{array}
$$

- "Külmutatud avaldised"

```
(define-record thunk (exp env))
```

“Call-by-name” interpretaator

Argumentide “külmutamine”:

```
(define eval-rands
  (lambda (rands env)
    (map (lambda (rand)
           (variant-case rand
             (varref (var)
               (let ((den-val (apply-env env var)))
                 (if (thunk? den-val)
                     den-val
                     (make-thunk rand env))))
             (else (make-thunk rand env))))
         rands)))
```

“Call-by-name” interpretaator

“Külmutatud avaldiste” väärtustamine

```
(define denoted->L-value
  (lambda (den-val)
    (if (thunk? den-val)
        (eval-rand (thunk->exp den-val)
                   (thunk->env den-val))
        den-val)))
```

“Call-by-name” interpretaator

```
(define denoted->expressed
  (lambda (den-val)
    (let ((l-val (denoted->L-value den-val)))
      (cond
        ((cell? l-val) (cell-ref l-val))
        ((ae? l-val) (array-ref (ae->array l-val)
                                (ae->index l-val)))
        (else (error "Can't dereference"
                     "denoted value:" l-val))))))
```

“Call-by-name” interpretaator

```
(define denoted-value-assign!
  (lambda (den-val exp-val)
    (let ((l-val (denoted->L-value den-val)))
      (cond
        ((cell? l-val) (cell-set! l-val exp-val))
        ((ae? l-val) (array-set! (ae->array l-val)
                                 (ae->index l-val) exp-val))
        (else (error "Can't assign"
                     "to denoted value:" l-val))))))
```

“Call-by-need” interpretaator

- Call-by-need — erineb nime kaudu edastamisest selle poolest, et pärast “külmutatud avaldise” väärtustamist, peetakse tema väärtus meeles (memoiseeritakse) ja järgnevatel kordadel kasutatakse seda väärtust
  - parameeter väärtustatakse maksimaalselt üks kord
- Andmemudel:

$$\begin{aligned} \mathit{Denoted\ Value} &= \mathit{L{-}Value} + \mathit{Memo} \\ \mathit{Memo} &= \mathit{Cell}(\mathit{Thunk} + \mathit{L{-}Value}) \end{aligned}$$

- Memoiseeritud väärtused

```
(define-record memo (cell))
```

“Call-by-need” interpretaator

```
(define eval-rands (lambda (rands env)
  (map (lambda (rand)
         (variant-case rand
          (varref (var)
            (let ((den-val (apply-env env var)))
              (if (memo?  den-val)
                  den-val
                  (make-memo (make-cell
                              (make-thunk rand env))))))
          (else (make-memo (make-cell
                              (make-thunk rand env))))))
       rands)))
```

“Call-by-need” interpretaator

```
(define denoted->L-value (lambda (den-val)
  (if (memo? den-val)
      (let ((cell (memo->cell den-val)))
        (let ((contents (cell-ref cell)))
          (if (thunk? contents)
              (let ((l-val (eval-rand
                            (thunk->exp contents)
                            (thunk->env contents))))
                (cell-set! cell l-val)
                l-val)
              contents)))
      den-val)))
```

Järgmiseks korraks

- Lugeda läbi EOPL ptk. 6.4 – 6.7
- “Mängida” interpretaatoritega
  - `loeng14-1.ss` — expressed = number;
  - `loeng14-2.ss` — “call-by-name”;
  - `loeng14-3.ss` — “call-by-need”.
- 2. komplekt ülesandeid:
  - EOPL 5.5.5, 5.5.7, 5.7.8, 6.1.1, 6.3.1, 6.6.1, 6.6.3
  - tähtaeg kolmapäev 25. aprill, kell 11.59 !!