Lambda-arvutus

- Lambda-avaldiste süntaks

  ⟨*expr*⟩ ::= ⟨*variable*⟩
  | (lambda (⟨*variable*⟩) ⟨*expr*⟩)
  | (⟨*expr*⟩ ⟨*expr*⟩)

- Näiteid lambda-termidest:

```
(lambda (x) x)
(lambda (x) y)
((lambda (x) x) x)
```

Vabad ja seotud muutujad

- Definitsioonid:
  - muutuja on avaldises seotud, kui ta viitab mingile formaalsele parameetrile

    ```
    ((lambda (x) x) y)
    ```

  - muutuja on avaldises vaba, kui ta ei ole seotud

    ```
    ((lambda (x) x) y)
    ```

  - avaldist, milles ei leidu ühtegi vaba muutujat, nimetatakse kombinaatoriks

    ```
    (lambda (f) ((lambda (x) (f x))))
    ```

Vabad muutujad

- Muutuja x on termis E vaba, kui
  1. E on muutuja x; või
  2. E on kujul $(E_1\ E_2)$ ja x on vaba kas termis E_1 või termis E_2; või
  3. E on kujul $(\texttt{lambda}\ (y)\ E')$, kus $y \neq x$ ja x on vaba termis E'.
- Predikaat `free?`

```
> (free? 'x '(lambda (x) x))
#f
> (free? 'x '((lambda (x) x) x))
#t
```

`free?` definitsioon

```
(define free?
  (lambda (x expr)
    (if (symbol? expr)
        (eq? x expr)
        (if (eq? (car expr) 'lambda)
            (free-in-lambda? x (caadr expr)
                               (caddr expr))
            (free-in-application? x (car expr)
                                    (cadr expr))))))
```

`free?` definitsioon

```
(define free-in-application?
  (lambda (x expr1 expr2)
    (or (free? x expr1) (free? x expr2))))

(define free-in-lambda?
  (lambda (x y expr)
    (if (eq? x y)
        #f
        (free? x expr))))
```

Seotud muutujad

- Muutuja x on seotud termis E, kui
  1. E on kujul $(E_1\ E_2)$ ja x on seotud kas termis E_1 või termis E_2; või
  2. E on kujul $(\texttt{lambda}\ (y)\ E')$ ja x on seotud termis E' või $x = y$ ja y on vaba termis E'.
- Predikaat `bound?`

```
> (bound? 'x '((lambda (x) x) x))
#t
> (bound? 'x '((lambda (x) y) x))
#f
```

bound? definitsioon

```
(define bound?
  (lambda (x expr)
    (if (symbol? expr)
        #f
        (if (eq? (car expr) 'lambda)
            (bound-in-lambda? x (caadr expr)
                                (caddr expr))
            (bound-in-application? x (car expr)
                                     (cadr expr))))))
```

bound? definitsioon

```
(define bound-in-application?
  (lambda (x expr1 expr2)
    (or (bound? x expr1) (bound? x expr2))))

(define bound-in-lambda?
  (lambda (x y expr)
    (or (bound? x expr)
        (and (eq? x y) (free? y expr)))))
```

Skoop

- Treminoloogiat:
  - muutuja skoop on programmi osa, kus antud muutuja on nähtav
  - kui muutuja skoop on määratud staatiliselt, siis on keel leksiliselt skoobitud
  - kui muutujate skoobid võivad olla üksteisesse sisestatud, siis on keel plokkstruktuuriga
  - kui välimises plokis olev muutuja ei ole sisemises plokis nähtav kuna seal on see muutuja üledeklareeritud, siis sisemine deklaratsioon varjutab välimise

Leksiline aadress

- Muutuja leksiline aadress on arv mis näitab mitu plokki väljaspool see muutuja on defineeritud

```
(lambda (x) (x : 0))
(lambda (x) (lambda (y) ((x : 1) (y : 0))))
```

- Leksiline aadress identifitseerib unikaalselt muutuja; seega pole rangelt võttes muutjatel nimesid vaja

```
(lambda 0)
(lambda (lambda (1 0))
```

Leksiline aadress

- Kui lambda-abstraktsioonis võib olla mitu parameetrit, siis on leksilisel aadressil teine komponent — antud muutuja positsioon parameetrite listis

```
(lambda (x y)
  ((lambda (a)
     ((x : 1 0) ((a : 0 0) (y : 1 1))))
   (x : 0 0)))
```

  või ilma muutujate nimedeta

```
(lambda 2
  ((lambda 1
     ((1 0) ((0 0) (1 1)))) (0 0)))
```

Leksiline aadress

- Leksilise aadressi leidmine

```
> (lexical-address '(lambda (x) x))
(lambda (x) (x : 0))
> (lexical-address '((lambda (x) x) x))
((lambda (x) (x : 0)) x)
> (lexical-address '((lambda (x) y) x))
((lambda (x) y) x)
```

- Definitsioon

```
(define lexical-address
  (lambda (expr)
    (lexical-address-aux expr '())))
```

Leksiline aadress

```
(define lexical-address-aux
  (lambda (expr va-list)
    (if (symbol? expr)
        (lexical-var expr va-list)
        (if (eq? (car expr) 'lambda)
            (lexical-lam (caadr expr)
                         (caddr expr)
                        va-list)
            (lexical-app (car expr)
                         (cadr expr)
                        va-list)))))
```

Leksiline aadress

```
(define lexical-var
  (lambda (x va-list)
    (if (null? va-list)
        x
        (if (eq? x (caar va-list))
            (list x ': (cadar va-list))
            (lexical-var x (cdr va-list))))))
```

Leksiline aadress

```
(define lexical-app
  (lambda (expr1 expr2 va-list)
    (list (lexical-address-aux expr1 va-list)
          (lexical-address-aux expr2 va-list))))

(define lexical-lam
  (lambda (x expr va-list)
    (list 'lambda
          (list x)
          (lexical-address-aux expr
                               (new-block x va-list)))))
```

Leksiline aadress

```
(define new-block
  (lambda (x va-list)
    (cons (list x 0) (inc-addr va-list))))

(define inc-addr
  (lambda (va-list)
    (if (null? va-list)
        '()
        (cons (list (caar va-list)
                    (+ 1 (cadar va-list)))
              (inc-addr (cdr va-list))))))
```

Koduülesanded

- Järgmiseks korraks
  - lugeda läbi EOPL ptk. 2.3 ja 2.4
- 1. komplekt ülesandeid:
  - EOPL 2.2.7.5, 2.2.9.4, 2.3.1, 2.3.10, 2.3.13, 2.3.14
  - tähtaeg kolmapäev 14. märts, kell 11.59 !!
  - ülesannete lahendamiseks moodustada rühmad (kuni 5 liiget)