Induktiivsed spetsifikatsioonid

Induktiivsed spetsifikatsioonid

- Andmetüüp koosneb
  - mingite väärtuste hulgast ja
  - operatsioonidest nendel väärtustel

Induktiivsed spetsifikatsioonid

- Andmetüüp koosneb
  - mingite väärtuste hulgast ja
  - operatsioonidest nendel väärtustel
- Operatsioonide defineerimiseks on vaja enne spetsifitseerida väärtuste hulk.

Induktiivsed spetsifikatsioonid

- Andmetüüp koosneb
  - mingite väärtuste hulgast ja
  - operatsioonidest nendel väärtustel
- Operatsioonide defineerimiseks on vaja enne spetsifitseerida väärtuste hulk.
- Induktiivne spetsifikatsioon
  - määratakse baaselemendid
  - antakse reeglid, kuidas olemasolevatest elementidest moodustada uusi

Induktiivsed spetsifikatsioonid

- Naturaalarvude spetsifikatsioon:
  1. arv 0 on naturaalarv;
  2. kui n on naturaalarv, siis on ka $n+1$ naturaalarv;
  3. reeglid 1 ja 2 määravad kõik naturaalarvud.

Induktiivsed spetsifikatsioonid

- Naturaalarvude spetsifikatsioon:
  1. arv 0 on naturaalarv;
  2. kui n on naturaalarv, siis on ka $n+1$ naturaalarv;
  3. reeglid 1 ja 2 määravad kõik naturaalarvud.
- Arvulistide spetsifikatsioon — on vähim hulk, mis rahuldab järmisi omadusi:
  1. `()` on arvulist;
  2. kui `n` on arv ja `lon` on arvulist, siis `(n . lon)` on arvulist.

Induktiivsed spetsifikatsioonid

- BNF (Backus-Naur Form)
  - terminaalsed sümbolid
  - mitteterminalid (süntaktilised kategooriad)
  - produktsiooni reeglid

Induktiivsed spetsifikatsioonid

- BNF (Backus-Naur Form)
  - terminaalsed sümbolid
  - mitteterminalid (süntaktilised kategooriad)
  - produktsiooni reeglid
- Arvulistid:

  $\langle \textit{list-of-numbers} \rangle$ ::= ()

  | ($\langle \textit{number} \rangle$. $\langle \textit{list-of-numbers} \rangle$)

Induktiivsed spetsifikatsioonid

- BNF (Backus-Naur Form)
  - terminaalsed sümbolid
  - mitteterminalid (süntaktilised kategooriad)
  - produktsiooni reeglid
- Arvulistid:

  $\langle \textit{list-of-numbers} \rangle$ ::= ()
  | ($\langle \textit{number} \rangle$. $\langle \textit{list-of-numbers} \rangle$)
- Binaarsed puud:

  $\langle \textit{binary-tree} \rangle$ ::= $\langle \textit{number} \rangle$
  | ($\langle \textit{symbol} \rangle$ $\langle \textit{binary-tree} \rangle$ $\langle \textit{binary-tree} \rangle$)

Rekursiivsed protseduurid

- Rekursiivselt spetsifitseeritud andmetüüpidel opereerivad protseduurid on (peaaegu alati) defineeritud rekursiivselt

Rekursiivsed protseduurid

- Rekursiivselt spetsifitseeritud andmetüüpidel opereerivad protseduurid on (peaaegu alati) defineeritud rekursiivselt
- Iga rekursiivne protseduur koosneb:
  - ühest või rohkemast baasvariandist;
  - ühest või rohkemast rekursiivsest variandist.

Rekursiivsed protseduurid

- Rekursiivselt spetsifitseeritud andmetüüpidel opereerivad protseduurid on (peaaegu alati) defineeritud rekursiivselt
- Iga rekursiivne protseduur koosneb:
  - ühest või rohkemast baasvariandist;
  - ühest või rohkemast rekursiivsest variandist.
- Rekursiivsed väljakutsed peavad (üldjuhul) olema “väiksemate” argumentidega

Rekursiivsed protseduurid

- Rekursiivselt spetsifitseeritud andmetüüpidel opereerivad protseduuurid on (peaaegu alati) defineeritud rekursiivselt
- Iga rekursiivne protseduur koosneb:
  - ühest või rohkemast baasvariandist;
  - ühest või rohkemast rekursiivsest variandist.
- Rekursiivsed väljakutsed peavad (üldjuhul) olema “väiksemate” argumentidega
- Rekursiivse protseduuri struktuur on sageli sama mis töödeldava andmestruktuuri BNF-spetsifikatsioon

Rekursiivsed protseduurid

- Näide — defineerida predikaat `list-of-numbers?`, mis kontrollib kas argument on arvulist

```
> (list-of-numbers? '())
#t
> (list-of-numbers? '(1 2 3))
#t
> (list-of-numbers? '(1 two 3))
#f
> (list-of-numbers? '(1 . 2))
#f
```

Rekursiivsed protseduurid

`list-of-numbers?` definitsioon:

```
(define list-of-numbers?
  (lambda (lst)
    ...))
```

Rekursiivsed protseduurid

`list-of-numbers?` definitsioon:

```
(define list-of-numbers?
  (lambda (lst)
    (if (null? lst)
        #t
        ...)))
```

Rekursiivsed protseduurid

`list-of-numbers?` definitsioon:

```
(define list-of-numbers?
  (lambda (lst)
    (if (null? lst)
        #t
        (if (pair? lst)
            ...
            #f))))
```

Rekursiivsed protseduurid

`list-of-numbers?` definitsioon:

```
(define list-of-numbers?
  (lambda (lst)
    (if (null? lst)
        #t
        (if (pair? lst)
            (if (number? (car lst))
                ...
                #f)
            #f))))
```

Rekursiivsed protseduurid

`list-of-numbers?` definitsioon:

```
(define list-of-numbers?
  (lambda (lst)
    (if (null? lst)
        #t
        (if (pair? lst)
            (if (number? (car lst))
                (list-of-numbers? (cdr lst))
                #f)
            #f))))
```

Rekursiivsed protseduurid

- Sümbol-listid

  ⟨*list-of-symbols*⟩ ::= ()
  | (⟨*symbol*⟩ . ⟨*list-of-symbols*⟩)

Rekursiivsed protseduurid

- Sümbol-listid

  ⟨*list-of-symbols*⟩ ::= ()
  | (⟨*symbol*⟩ . ⟨*list-of-symbols*⟩)

- Defineerida protseduurid `remove-first` ja `remove`, mis eemaldavad sümbolite listist ettentud sümboli vastavalt esimese või kõik esinemised:

```
> (remove-first 'b '(a b a b))
(a a b)
> (remove 'b '(a b a b))
(a a)
```

`remove-first` definitsioon

```
(define remove-first
  (lambda (s los)
        ...))
```

`remove-first` definitsioon

```
(define remove-first
  (lambda (s los)
    (if (null? los)
        ...
        ...)))
```

`remove-first` definitsioon

```
(define remove-first
  (lambda (s los)
    (if (null? los)
        '()
        ...)))
```

`remove-first` definitsioon

```
(define remove-first
  (lambda (s los)
    (if (null? los)
        '()
        (if (eq? s (car los))
            ...
            ...))))
```

`remove-first` definitsioon

```
(define remove-first
  (lambda (s los)
    (if (null? los)
        '()
        (if (eq? s (car los))
            (cdr los)
            ...))))
```

`remove-first` definitsioon

```
(define remove-first
  (lambda (s los)
    (if (null? los)
        '()
        (if (eq? s (car los))
            (cdr los)
            (cons (car los)
                  (remove-first s (cdr los)))))))
```

`remove` definitsioon

```
(define remove
  (lambda (s los)
        ...))
```

`remove` definitsioon

```
(define remove
  (lambda (s los)
    (if (null? los)
        '()
        (if (eq? s (car los))
            ...
            (cons (car los)
                  (remove s (cdr los)))))))
```

`remove` definitsioon

```
(define remove
  (lambda (s los)
    (if (null? los)
        '()
        (if (eq? s (car los))
            (remove s (cdr los))
            (cons (car los)
                  (remove s (cdr los)))))))
```

Näide — substitutsioon

- S-listid

$$\begin{aligned}
\langle \textit{s-list} \rangle &::= () \\
&\mid (\langle \textit{symbol-expr} \rangle \,.\, \langle \textit{s-list} \rangle) \\
\\
\langle \textit{symbol-expr} \rangle &::= \langle \textit{symbol} \rangle \\
&\mid \langle \textit{s-list} \rangle
\end{aligned}$$

Näide — substitutsioon

- S-listid

  ⟨*s-list*⟩ ::= ()
  | (⟨*symbol-expr*⟩ . ⟨*s-list*⟩)

  ⟨*symbol-expr*⟩ ::= ⟨*symbol*⟩
  | ⟨*s-list*⟩

- Defineerida protseduur `subst`, mis asendab S-listis kõik etteantud sümboli esinemised teise etteantud sümboliga:

```
> (subst 'a 'b '((b c) ((b) d)))
((a c) ((a) d))
```

`subst` definitsioon (ver. 1)

```
(define subst
 (lambda (new old slst)
      ...))
```

`subst` definitsioon (ver. 1)

```
(define subst
 (lambda (new old slst)
  (if (null? slst)
      ...
      ...)))
```

`subst` definitsioon (ver. 1)

```
(define subst
 (lambda (new old slst)
  (if (null? slst)
      '()
      ...)))
```

`subst` definitsioon (ver. 1)

```
(define subst
 (lambda (new old slst)
  (if (null? slst)
      '()
      (if (symbol? (car slst))
          ...
          ...))))
```

`subst` definitsioon (ver. 1)

```
(define subst
 (lambda (new old slst)
  (if (null? slst)
      '()
      (if (symbol? (car slst))
          (if (eq? (car slst) old)
              ...
              ...
          ...)))))
```

`subst` definitsioon (ver. 1)

```
(define subst
 (lambda (new old slst)
  (if (null? slst)
      '()
      (if (symbol? (car slst))
          (if (eq? (car slst) old)
              (cons new (subst new old (cdr slst)))
              ...
          ...)))))
```

`subst` definitsioon (ver. 1)

```
(define subst
 (lambda (new old slst)
  (if (null? slst)
      '()
      (if (symbol? (car slst))
          (if (eq? (car slst) old)
              (cons new (subst new old (cdr slst)))
              (cons (car slst)
                    (subst new old (cdr slst))))
          ...))))
```

`subst` definitsioon (ver. 1)

```
(define subst
 (lambda (new old slst)
  (if (null? slst)
      '()
      (if (symbol? (car slst))
          (if (eq? (car slst) old)
              (cons new (subst new old (cdr slst)))
              (cons (car slst)
                    (subst new old (cdr slst))))
          (cons (subst new old (car slst))
                (subst new old (cdr slst)))))))
```

`subst` definitsioon (ver. 2)

```
(define subst
  (lambda (new old slst)
    (if (null? slst)
        '()
        (cons (subst-symbol-expr new old (car slst))
              (subst new old (cdr slst))))))
(define subst-symbol-expr
  (lambda (new old se)
    (if (symbol? se)
        (if (eq? se old) new se)
        (subst new old se))))
```

Näide — `flatten`

- Defineerida protseduur `flatten`, mis teisendab S-listi sümbolite listiks (ex.: 2.2.8.4):

Näide — `flatten`

- Defineerida protseduur `flatten`, mis teisendab S-listi sümbolite listiks (ex.: 2.2.8.4):

```
> (flatten '(a b c d))
(a b c d)
> (flatten '((a b) c (((d)) e)))
(a b c d e)
> (flatten ' (a b (() (a))))
(a b a)
```

`flatten` definitsioon

- Kasutame akumuleerivat parameetrit

`flatten` definitsioon

- Kasutame akumuleerivat parameetrit

```
(define flatten-aux
  (lambda (slst los)
        ...))
```

`flatten` definitsioon

- Kasutame akumuleerivat parameetrit

```
(define flatten-aux
  (lambda (slst los)
    (if (null? slst)
        ...
        ...)))
```

`flatten` definitsioon

- Kasutame akumuleerivat parameetrit

```
(define flatten-aux
  (lambda (slst los)
    (if (null? slst)
        los
        ...)))
```

`flatten` definitsioon

- Kasutame akumuleerivat parameetrit

```
(define flatten-aux
  (lambda (slst los)
    (if (null? slst)
        los
        (flatten-sym-expr
              (car slst)
              (flatten-aux (cdr slst) los)))))
```

`flatten` definitsioon

```
(define flatten-sym-expr
  (lambda (sym-expr los)
        ...))
```

`flatten` definitsioon

```
(define flatten-sym-expr
  (lambda (sym-expr los)
    (if (symbol? sym-expr)
        ...
        ...)))
```

`flatten` definitsioon

```
(define flatten-sym-expr
  (lambda (sym-expr los)
    (if (symbol? sym-expr)
        (cons sym-expr los)
        ...)))
```

`flatten` definitsioon

```
(define flatten-sym-expr
  (lambda (sym-expr los)
    (if (symbol? sym-expr)
        (cons sym-expr los)
        (flatten-aux sym-expr los))))
```

`flatten` definitsioon

```
(define flatten-sym-expr
  (lambda (sym-expr los)
    (if (symbol? sym-expr)
        (cons sym-expr los)
        (flatten-aux sym-expr los))))

(define flatten
  (lambda (slst)
    (flatten-aux slst '())))
```

Järgmiseks korraks

- Lugeda läbi EOPL ptk. 2.1 ja 2.2
- Koduülesannete lahendamiseks moodustada rühmad (kuni 5 liiget)
  - esimene komplekt ülesandeid tuleb järgmine kord