

Rekursiivne vs. iteratiivne käitumine

Rekursiivne faktoriaal

$$\text{factR } 0 = 1$$

$$\text{factR } n = n * \text{factR } (n - 1)$$

Rekursiivne vs. iteratiivne käitumine

Rekursiivne faktoriaal

$$\text{factR } 0 = 1$$

$$\text{factR } n = n * \text{factR } (n - 1)$$

NB!

factR iga järgmine rekursiivne väljakutse toimub üha suuremas kontekstis:

$$\begin{aligned}\text{factR } 4 &\Longrightarrow 4 * \text{factR } 3 \\ &\Longrightarrow 4 * (3 * \text{factR } 2) \\ &\Longrightarrow 4 * (3 * (2 * \text{factR } 1)) \\ &\Longrightarrow 4 * (3 * (2 * (1 * \text{factR } 0))) \\ &\Longrightarrow 4 * (3 * (2 * (1 * 1))) \\ &\Longrightarrow 24\end{aligned}$$

Rekursiivne vs. iteratiivne käitumine

Sabarekursiivne (akumulaatoriga) faktoriaal

```
factA = factAcc 1  
  where factAcc a 0 = a  
        factAcc a n = factAcc (a * n) (n - 1)
```

NB!

Kõik väljakutsed toimuvad samas kontekstis:

```
factA 4  $\Rightarrow$  factAcc 1 4  
           $\Rightarrow$  factAcc 4 3  
           $\Rightarrow$  factAcc 12 2  
           $\Rightarrow$  factAcc 24 1  
           $\Rightarrow$  factAcc 24 0  
           $\Rightarrow$  24
```

Rekursiivne vs. iteratiivne käitumine

Iteratiivne kontroll-käitumine

- Funktsioon on **iteratiivse käitumisega** kui ta kasutab $O(1)$ mälu
- Funktsioon on **iteratiivse kontroll-käitumisega** kui ta kasutab $O(1)$ mälu konteksti säilitamiseks
- Funktsioon on **rekursiivse (kontroll-)käitumisega**, kui ta ei ole iteratiivse (kontroll-)käitumisega

Näide

```
reverse = revApp []  
  where revApp as []      = as  
        revApp as (x : xs) = revApp (x : as) xs
```

Rekursiivne vs. iteratiivne käitumine

Küsimus

Millal on funktsioon iteratiivse kontroll-käitumisega?

NB!

Üldjuhul lahendamatu ülesanne!

Näide

```
factQ n = if strangePredicate n  
           then factR n  
           else factA n
```

Rekursiivne vs. iteratiivne käitumine

Sabapositsioonid/-kutsed/-vormid

- Alamavaldis on **sabapositsioonis** (ingl. *tail position*), kui tema väärtustamise korral on ta väärtus “koheselt” kogu avaldise väärtuseks

if $expr_0$ then $expr_1$ else $expr_2$
let $x_1 = expr_1$ in $expr_0$

- Sabapositsioonis olevat funktsiooniaplikatsiooni nimetatakse **sabakutseks** (ingl. *tail call*)
- Avaldis on **sabavormis** (ingl. *tail form*), kui kõik tema mitte-sabapositsioonis olevad alamavaldised on ”lihtsad”

NB!

Sabavormis avaldis on iteratiivse kontroll-käitumisega!

Jätkud

Arvutuskontekstid ja jätkud

- Vaatame *factR* 4 arvutussammu $4 * (3 * (2 * \text{factR } 1))$
- Väljakutse *factR* 1 arvutatakse kontekstis $4 * (3 * (2 * \square))$
- Me saame konteksti (“ühe auguga” avaldist) esitada ühe argumendilise lambda-avaldisena, mida kutsutakse **jätkuks** (ingl. *continuation*) $\lambda v \rightarrow 4 * (3 * (2 * v))$
- Tähistagu *k* seda lambda-avaldist, siis kehtib

$$k (\text{factR } 1) \Longrightarrow 4 * (3 * (2 * \text{factR } 1))$$

- Analoogselt on kõik *factR* arvutamise sammud esitatavad kujul $(k (\text{factR } n))$, kus *k* on vastavat konteksti esitav jätk
- **CPS** = *Continuation Passing Style*

Jätkud

CPS-kujul faktoriaal

$$\mathit{factCPS} \ 0 \ k = k \ 1$$

$$\mathit{factCPS} \ n \ k = \mathit{factCPS} \ (n - 1) \ (\lambda v \rightarrow k \ (n * v))$$

Jätkud

CPS-kujul faktoriaal

$$\text{factCPS } 0 \ k = k \ 1$$

$$\text{factCPS } n \ k = \text{factCPS } (n - 1) (\lambda v \rightarrow k \ (n * v))$$

Väide

Iga k ja n korral

$$k \ (\text{factR } n) = \text{factCPS } n \ k$$

Jätkud

CPS-kujul faktoriaal

$$\mathit{factCPS} \ 0 \ k = k \ 1$$

$$\mathit{factCPS} \ n \ k = \mathit{factCPS} \ (n - 1) \ (\lambda v \rightarrow k \ (n * v))$$

Väide

Iga k ja n korral

$$k \ (\mathit{factR} \ n) = \mathit{factCPS} \ n \ k$$

Tõestus ($n = 0$)

$$k \ (\mathit{factR} \ 0) = k \ 1 = \mathit{factCPS} \ 0 \ k$$

Jätkud

CPS-kujul faktoriaal

$$\text{factCPS } 0 \ k = k \ 1$$

$$\text{factCPS } n \ k = \text{factCPS } (n - 1) (\lambda v \rightarrow k (n * v))$$

Väide

Iga k ja n korral

$$k (\text{factR } n) = \text{factCPS } n \ k$$

Tõestus ($n > 0$)

$$\begin{aligned} k (\text{factR } n) &= k (n * \text{factR } (n - 1)) \\ &= (\lambda v \rightarrow k (n * v)) (\text{factR } (n - 1)) \\ &= \text{factCPS } (n - 1) (\lambda v \rightarrow k (n * v)) \\ &= \text{factCPS } n \ k \end{aligned}$$

Jätkud

NB!

factCPS on iteratiivse kontroll-käitumisega

$$\begin{aligned} & \text{factCPS } 4 \ k \\ \Rightarrow & \text{factCPS } 3 \ (\lambda v \rightarrow k \ (4 * v)) \\ \Rightarrow & \text{factCPS } 2 \ (\lambda v \rightarrow k \ (4 * (3 * v))) \\ \Rightarrow & \text{factCPS } 1 \ (\lambda v \rightarrow k \ (4 * (3 * (2 * v)))) \\ \Rightarrow & \text{factCPS } 0 \ (\lambda v \rightarrow k \ (4 * (3 * (2 * (1 * v))))) \\ \Rightarrow & k \ (4 * (3 * (2 * (1 * 1)))) \\ \Rightarrow & k \ 24 \end{aligned}$$

NB!

“Kontekstist sõltumatu” faktoriaali saame, kui anname algjätkuks identsusfunktsiooni

$$\text{factC } n = \text{factCPS } n \ \text{id}$$

Jätkud

Rekursiivne *length*

$$\begin{aligned} \text{length } [] &= 0 \\ \text{length } (x : xs) &= 1 + \text{length } xs \end{aligned}$$

CPS-kujul *length*

$$\begin{aligned} \text{lengthC } xs &= \text{lengthCPS } xs \text{ id} \\ \textbf{where } \text{lengthCPS } [] \quad k &= k \ 0 \\ \text{lengthCPS } (x : xs) \ k &= \text{lengthCPS } xs \ (\lambda v \rightarrow \\ &\quad k \ (1 + v)) \end{aligned}$$

Jätkud

CPS-teisendus

Üldine meetod avaldise teisendamiseks CPS kujule:

- Identifitseerida mittetriviaalsed vahetulemused
- Järjestikustada nende arvutamine
- Tuua sisse jätkud konteksti esitamiseks

Jätkud

CPS-teisendus

Üldine meetod avaldise teisendamiseks CPS kujule:

- Identifitseerida mittetriviaalsed vahetulemused
- Järjestikustada nende arvutamine
- Tuua sisse jätkud konteksti esitamiseks

Näide: S-kombinaator

$f\ x\ (g\ x)$

let $v1 = f\ x$
 $v2 = g\ x$
 $v3 = v1\ v2$
in $v3$

$\lambda k \rightarrow f\ x\ (\lambda v1 \rightarrow$
 $g\ x\ (\lambda v2 \rightarrow$
 $v1\ v2\ (\lambda v3 \rightarrow$
 $k\ v3)))$

Jätkud

CPS-teisendus

Üldine meetod avaldise teisendamiseks CPS kujule:

- Identifitseerida mittetriviaalsed vahetulemused
- Järjestikustada nende arvutamine
- Tuua sisse jätkud konteksti esitamiseks

Näide: S-kombinaator

$f\ x\ (g\ x)$

let $v1 = f\ x$
 $v2 = g\ x$
 $v3 = v1\ v2$
in $v3$

$\lambda k \rightarrow f\ x\ (\lambda v1 \rightarrow$
 $g\ x\ (\lambda v2 \rightarrow$
 $v1\ v2\ (\lambda v3 \rightarrow$
 $k\ v3)))$

Jätkud

CPS-teisendus

Üldine meetod avaldise teisendamiseks CPS kujule:

- Identifitseerida mittetriviaalsed vahetulemused
- Järjestikustada nende arvutamine
- Tuua sisse jätkud konteksti esitamiseks

Näide: S-kombinaator

$f\ x\ (g\ x)$

let $v1 = f\ x$
 $v2 = g\ x$
 $v3 = v1\ v2$
in $v3$

$\lambda k \rightarrow f\ x\ (\lambda v1 \rightarrow$
 $g\ x\ (\lambda v2 \rightarrow$
 $v1\ v2\ (\lambda v3 \rightarrow$
 $k\ v3)))$

Jätkud

CPS-teisendus

Üldine meetod avaldise teisendamiseks CPS kujule:

- Identifitseerida mittetriviaalsed vahetulemused
- Järjestikustada nende arvutamine
- Tuua sisse jätkud konteksti esitamiseks

Näide: Fibonacci

$$\text{fib } 0 = 0$$

$$\text{fib } 1 = 1$$

$$\text{fib } n = \text{fib } (n - 1) + \text{fib } (n - 2)$$

Jätkud

CPS-teisendus

Üldine meetod avaldise teisendamiseks CPS kujule:

- Identifitseerida mittetriviaalsed vahetulemused
- Järjestikustada nende arvutamine
- Tuua sisse jätkud konteksti esitamiseks

Näide: Fibonacci

$fib\ 0 = 0$

$fib\ 1 = 1$

$fib\ n = \text{let } v1 = fib\ (n - 1)$

$\quad v2 = fib\ (n - 2)$

$\text{in } v1 + v2$

Jätkud

CPS-teisendus

Üldine meetod avaldise teisendamiseks CPS kujule:

- Identifitseerida mittetriviaalsed vahetulemused
- Järjestikustada nende arvutamine
- Tuua sisse jätkud konteksti esitamiseks

Näide: Fibonacci

$$\text{fib } 0 \ k = k \ 0$$

$$\text{fib } 1 \ k = k \ 1$$

$$\begin{aligned} \text{fib } n \ k = & \text{fib } (n - 1) (\lambda v1 \rightarrow \\ & \text{fib } (n - 2) (\lambda v2 \rightarrow \\ & k \ (v1 + v2))) \end{aligned}$$

Jätkud

CPS-teisendus

Üldine meetod avaldise teisendamiseks CPS kujule:

- Identifitseerida mittetriviaalsed vahetulemused
- Järjestikustada nende arvutamine
- Tuua sisse jätkud konteksti esitamiseks

Näide: *map*

$$\begin{aligned} \text{map } f [] &= [] \\ \text{map } f (x : xs) &= f\ x : \text{map } f\ xs \end{aligned}$$

Jätkud

CPS-teisendus

Üldine meetod avaldise teisendamiseks CPS kujule:

- Identifitseerida mittetriviaalsed vahetulemused
- Järjestikustada nende arvutamine
- Tuua sisse jätkud konteksti esitamiseks

Näide: *map*

$$\begin{aligned} \text{map } f [] &= [] \\ \text{map } f (x : xs) &= \text{let } v1 = f x \\ &\quad v2 = \text{map } f xs \\ &\quad \text{in } v1 : v2 \end{aligned}$$

Jätkud

CPS-teisendus

Üldine meetod avaldise teisendamiseks CPS kujule:

- Identifitseerida mittetriviaalsed vahetulemused
- Järjestikustada nende arvutamine
- Tuua sisse jätkud konteksti esitamiseks

Näide: *map*

$$\begin{aligned} \text{map } f [] \quad k &= k [] \\ \text{map } f (x : xs) \, k &= f \, x \, (\lambda v1 \rightarrow \\ &\quad \text{map } f \, xs \, (\lambda v2 \rightarrow \\ &\quad \quad k \, (v1 : v2))) \end{aligned}$$

Jätkud

CPS vahekokkuvõte

CPS kujul programmis on kõik mittetriviaalsed tegevused ning nende tegemise järjekord ilmutatud

- Lihtsustab semantikat ning transleerimist
- Kõrgemat-järku funktsioonide asemel saab jätkude esitamiseks kasutada andmestruktuure (**defunktsionaliseerimine**)
- Jätke võib ka kasutada "mitte-standartselt", võimaldades selliselt modelleerida erinevaid juhtimiskonstruktsioone

Näiteid jätkude rakendustest: erindite töötlus

Leida listielementide korrutis (ver. 1)

$$\begin{aligned} \text{prod } [] &= 1 \\ \text{prod } (x : xs) &= x * \text{prod } xs \end{aligned}$$

NB!

Kui argumentlist sisaldab nulli, siis on tulemus alati null!

$$\text{prod } [1, 2, 3, 0, 1, 2, 3, 4, 5, 6, 7] \implies 0$$

NB!

Toodud definitsiooni korral läbitakse kogu list ja korrutatakse kõik elemendid!

Näiteid jätkude rakendustest: erindite töötlus

Leida listielementide korrutis (ver. 2)

$$\begin{aligned} \text{prod } [] &= 1 \\ \text{prod } (0 : xs) &= 0 \\ \text{prod } (x : xs) &= x * \text{prod } xs \end{aligned}$$

NB!

Läbitakse listi kuni esimese nullini!

NB!

Tehakse nii mitu korrutamist, kui mitu elementi läbiti!

Näiteid jätkude rakendustest: erindite töötlus

Jätkudega versioon

```
prod xs = prodC xs id
  where prodC []      k = k 1
        prodC (0 : xs) k = 0
        prodC (x : xs) k = prodC xs (\v → k (x * v))
```

NB!

List läbitakse ainult kui (esimese) nullini!

NB!

Nulli korral ei tehta ühtegi korrutamist!

Näiteid jätkude rakendustest: mitu resultaati

Leida listi pikkus ja elementide summa

```
sumLength = sumLen 0 0  
  where sumLen s l []      = (s, l)  
        sumLen s l (x : xs) = sumLen (s + x) (l + 1) xs
```

NB!

Ehitab paari, mille järgmine funktsioon, mis tahab tulemust kasutada, peab kohe “tükeldama”!

```
average xs = let (s, l) = sumLength xs  
               in s 'div' l
```

Näiteid jätkude rakendustest: mitu resultaati

Jätkudega versioon

```
sumLengthC xs k = sumLen 0 0 xs
  where sumLen s l []      = k s l
        sumLen s l (x : xs) = sumLen (s + x) (l + 1) xs
averageC xs = sumLengthC xs (\s l → s 'div' l)
```

NB!

Resultaadid antakse otse edasi!

Jätkude esitamine andmestruktuuridena

CPS-kujul faktoriaal

```
factC n = factCPS n id  
  where factCPS 0 k = k 1  
        factCPS n k = factCPS (n - 1)  
                          (λv → k (n * v))
```

Jätkude esitamine andmestruktuuridena

Jätkuesitusest sõltumatu CPS-kujul faktoriaal

```
factC n = factCPS n mkFinalCont  
  where factCPS 0 k = applyCont k 1  
        factCPS n k = factCPS (n - 1)  
                      (mkNewCont k n)
```

Jätkude esitamine funktsioonidena

```
mkFinalCont      = id  
mkNewCont k n =  $\lambda v \rightarrow \text{applyCont } k (n * v)$   
applyCont k v   = k v
```

Jätkude esitamine andmestruktuuridena

Jätkude esitamine andmestruktuuridena

```
data FactCont = FinalFactCont
              | NewFactCont Int FactCont

mkFinalCont    = FinalFactCont
mkNewCont k n  = NewFactCont n k
applyCont k v  = case k of
    FinalFactCont    → v
    NewFactCont n k  → applyCont k (n * v)
```

NB!

Tüüp *FactCont* on isomorfne täisarvu listidega!

Jätkude esitamine andmestruktuuridena

Jätkude esitamine andmestruktuuridena

```
mkFinalCont      = []  
mkNewCont k n = n : k  
applyCont k v   = foldl ( $\lambda v\ n \rightarrow n * v$ ) v k
```

NB!

Asendame need definitsioonid tagasi *factC* definitsiooni ...

Jätkude esitamine andmestruktuuridena

CPS-kujul faktoriaal — jätkud listidena

```
factC n = factCPS n []  
  where factCPS 0 k = foldl ( $\lambda v\ n \rightarrow n * v$ ) 1 k  
        factCPS n k = factCPS (n - 1) (n : k)
```