Jätkuedastusstiilid ja jätkuedastus-denotatsioonisemantika

Riina Kikas

Kontrollkontekst

- Faktoriaali rekursiivne definitsioon

```
fact =  λn.if (n=0) 1 (* n (fact (- n 1)))
```

- `fact` iga järgmine rekursiivne väljakutse toimub üha suuremas kontekstis:

```
fact 4 -> * 4 (fact 3)
 -> * 4 (* 3 (fact 2))
 -> * 4 (* 3 (* 2 (fact 1)))
 -> * 4 (* 3 (* 2 (* 1 (fact 0))))
 -> * 4 (* 3 (* 2 (* 1 1)))
 -> 24
```

Kontrollkontekst

- Faktoriaali saba-rekursiivne definitsioon

```
fact-iter = λn. fact-iter-acc n 1
fact-iter-acc λn a.
    if (n=0) a (fact-iter-acc (- n 1) (* n a))
```

- `fact-iter` kõik rekursiivsed väljakutsed toimuvad samas kontekstis

```
fact-iter 4 -> fact-iter-acc 4 1
-> fact-iter-acc 3 4
-> fact-iter-acc 2 12
-> fact-iter-acc 1 24
-> fact-iter-acc 0 24
-> 24
```

Kontrollkäitumine

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

Jätkud

- Jätk (ingl. k. *continuation*) on kontrollkontekst abstraktsioon
- Avaldise jätk esitab protseduuri, mis saab argumendiks selle avaldise väärtus ja viib arvutuse lõpule

Jätkud

- Näide: `fact` arvutamise samm

  `* 4 (* 3 (* 2 (fact 1)))`, kus

  `fact 1` arvutatakse kontekstis

  `(* 4 (* 3 (* 2 □)))`

- Seda konteksti saame esitada ühe

argumendilise lambda-avaldisena e. jätkuna:

`λv. * 4 (* 3 (* 2 v))`

- Tähistagu k seda lambda-avaldist, siis kehtib

`k (fact 1) -> * 4 (* 3 (* 2 (fact 1)))`

Jätkude kasutamine

- Denotatsioonsemantika stiil (jätkuedastussemantika), et anda tähendus üldistatud hüpetele
- Programmeerimistehika (funktsionaalsetes) keeltes (jätkuedastusstiil)
- Keelekonstruktsioon (funktsionaalsetes) keeltes
- Kompileerimistehnika

Jätkuedastusstiil

- Jätkuedastusstiil (ingl. k. *continuation-passing style* e. *CPS*) on programmeerimisstiil, kus kasutatakse jätke.
- Iga avaldist saab transformeerida ekvivalentseks avaldiseks, millel on iteratiivne kontrollkäitumine (CPS-transformatsioon).

Jätkuedastusstiil - näide

- Näide: CPS-kujul faktoriaal

```
fact-cps = λn k. if (n=0) (k 1)
       (fact-cps (- n 1)(lambda (v) (k (* n v))))
```

- Iga k ja n korral kehtib

  `k (fact n) => fact-cps n k`

  Tõestus induktsiooniga:

  kui `n = 0`: `(k (fact 0)) => (k 1) => (fact-cps 0 k)`

  kui `n > 0`: `(k (fact n)) => (k (* n (fact (- n 1))))`

  `=> ((lambda (v) (k (* n v))) (fact (- n 1)))`

  `=> (fact-cps (- n 1) (lambda (v) (k (* n v))))`

  `=> (fact-cps n k)`

Jätkuedastusstiil - näide

- `fact-cps` on iteratiivse kontrollkäitumisega
- “Kontekstist sõltumatu” faktoriaali saame, kui anname algjätkuks identsusfunktsiooni:

```
fact = λn. fact-cps n (λv. v)
```

Jätkuedastusstiil

- Rekursiivset kontrollkäitumist ei põhjusta protseduuride väljakutsed, vaid argumentide väärtustamine, s.t. kontrollkäitumise määrab alamavaldise positsioon kogu avaldises.
- Üldjuhul ei saa öelda, kas avaldis on iteratiivse kontrollkäitumisega.

Jätkude rakendused - erindid

- Leida arvude listi elementide korrutis:

```
allprod = λ lst. if (lst=[]) 1
                    (* (head lst ) (allprod (tail list)))
```

- Kui argumentlist sisaldab nulli, siis on tulemus alati null.

```
>allprod [1,2,3,0,1,2,3,4,5,6,7]
0
```

- Alati läbitakse kogu list ja korrutatakse kõik elemendid.

Jätkude rakendused - erindid

- Akumulaatoriga versioon:

```
allprod-acc = λ lst acc. if (lst = []) acc
      (if ((head lst) = 0) 0
      (allprod-acc (tail lst) (*(head lst) acc)))
```

- List läbitakse ainult kuni (esimese) nullini.
- Tehakse nii mitu korrutamist, kui mitu elementi läbiti.

Jätkude rakendused - erindid

- Jätkudega versioon:

```
allprod-k = λ lst k. if (lst = []) (k 1)
   (if ((head lst) = 0) 0
   (allprod-k (tail lst) (λ z. k (* (head lst) z))))
```

- List läbitakse ainult kuni (esimese) nullini.
- Kui listis leidub nulle, ei tehta ühtegi korrutamist.

Jätkude rakendused - erindid

```
allprod-k [1,n] (λ x.x)
→  allprod-k (tail [1,n])
   (λ z. (λ x.x) (* (head [1,n]) z))
```

- kui $n = 0$:

```
→  0
```

- kui $n = 3$ ($n > 0$) :

```
→  allprod-k (tail (tail [1,3])) (λ y. (λ z. (λ x.x)
   (* (head [1,3]) z))) (* (head (tail [1,3])) y))))
→  (λ y. (λ z. (λ x.x) (* (head [1,3]) z)))
   (* (head (tail [1,3])) y)) 1
→  (λ y. (λ z. (λ x.x) (* 1 z)) (* 3 y)) 1
```

Jätkude rakendused: mitu resultaati

- Leida listi pikkus ja elementide summa:

```
sum-and-length = λ lst acc-sum acc-len.
    if (lst = []) (cons acc-sum acc-len)
    (sum-and-length (tail lst) (+ acc-sum (head lst))
    (+ acc-len 1))
```

- Ehitab paari, mille järgmine protseduur, mis tahab tulemust kasutada, peab kohe "tükeldama".

```
average = λ lst.let pair = sum-and-length lst 0 0
                in (/ (head pair) (tail pair))
```

Jätkude rakendused: mitu resultaati

- CPS versioon:

```
sum-and-length-k = λ lst acc-sum acc-len k.
   if (lst=[]) (k acc-sum acc-len)
   (sum-and-length-k (tail lst) (+ acc-sum (head
   lst)) (+ acc-len 1))

average-k = λ lst. sum-and-length lst 0 0
   (λ sum len. / sum len)
```

- Resultaadid antakse otse edasi.

Jätkude esitus funktsioonidega

- CPS-kujul remove
- `remove = λ s los. remove-cps s los (λ x.x)`
- `remove-cps = λ s los k. if (los = []) (k [])`
  `(if ((head los) = s) (remove-cps s (tail los) k)`
  `(remove-cps s (tail los) (λ v. k ((head los):v))))`
- `remove = λ s los. remove-cps s los make-final`
- `remove-cps = λ s los k. if (los = []) (apply-cont k [])`
  `(if ((head los) = s) (remove-cps s (tail los) k)`
  `(remove-cps s (tail los) (make-rem1 los k)))`

Jätkude esitus funktsioonidega

- `make-final = λ x. x`
- `make-rem1 = λ los k v. apply-cont k ((head los):v)`
- `apply-cont = λ k v. k v`

Kontrolloperaatorid

- Jätkude kasutamiseks tuuakse kontrolloperaatorid sisse keelekonstruktsioonidena
- Näiteks `goto, call/cc`

`call/cc`

- `call/cc (call-with-current-continuation)` saab argumendina funktsiooni, mille argument seotakse hetkel kehtiva jätkuga. Jätku saab alamavaldises kasutada kui üheargumendilist funktsiooni.
- `call/cc (λ k. E)` väärtustamine:
- kui alamavaldise *E* väärtustamisel jätku *k* ei kasutata, on *E* väärtus kogu avaldise väärtuseks.
- kui *E* väärtustamisel kutsutakse *k* välja argumendiga E_0, siis on E_0väärtus kogu avaldise väärtuseks:

```
+ 2 (call/cc (λ k. * 3 (k 4)))
> 6
```

`call/cc` - näited

- Näide: listi elementide korrutis

```
allprod = λ lst. call/cc (λ exit.
    let allP = λ lst. if (lst = []) 1
           (if ((head lst) = 0) (exit 0)
           (* (head lst) (allP (tail lst))))
    in allP lst
```

- Näide: väljastada kõik listi elemendid pärast etteantud elemendi viimast esinemist

```
rember-upto-last = λ a lat. call/cc
    (λ skip.let r = λ lat. if (lat = []) []
           (if ((head lat) = a) (skip (r tail lat))
           ((head lat):(r (tail lat))))
    in r lat
```

Ülessuunatud jätkud

- Siiani kõik allasuunatud (i.k. *downward)* jätkud, s.t. jätke kasutati alamavaldises konteksti taastamiseks.
- Ülessuunatud (i.k. *upward)* jätkud – jätku kasutatakse välises kontekstis.

Ülessuunatud jätkud

- `break` – kogu avaldise väärtuseks saab `break` argumendi väärtus ja `resume` väärtuseks saab jätk kohast, kus `break` töö katkestas.
- `resume` – jätk, mida argumendile rakendades saab jätkata programmi tööd `break` poolt katkestatud kohast

Ülessuunatud jätkud - näide

```
> + (break( 1)) (break( 2))
break: 1
> resume(5)
break: 2
> resume(3)
8
```

call-by-value CPS transformatsioon

- CPS transformatsiooni $C\langle\rangle$ reeglid

$$C\langle x\rangle = \lambda k.k\ x$$

$$C\langle \lambda x.E\rangle = \lambda k.k\ (\lambda x.C\langle E\rangle)$$

$$C\langle E_0\ E_1\rangle = \lambda k.C\langle E_0\rangle(\lambda v_0.C\langle E_1\rangle(\lambda v_1.v_0 v_1 k))$$

call-by-value CPS transformatsioon

- CPS transformatsiooni $C\langle\rangle$ reeglid

$$C\langle x\rangle = \lambda k.k\ x$$

$$C\langle \lambda x.E\rangle = \lambda k.k\ (\lambda xk.C\langle E\rangle\ k)$$

$$C\langle E_0\ E_1\rangle = \lambda k.C\langle E_0\rangle(\lambda v_0.C\langle E_1\rangle(\lambda v_1.v_0 v_1(\lambda v.k\ v))$$

call-by-value CPS transformatsioon

- CPS transformatsiooni $C\langle\rangle$ reeglid

$C\langle c\rangle = \lambda k.k\ c$

$C\langle x\rangle = \lambda k.k\ x$

$C\langle \text{if } B\ E_1\ E_2\rangle = \lambda k.C\langle B\rangle(\lambda b.\ \text{if } b\ (C\langle E_1\rangle\ k)\ (C\langle E_2\rangle\ k))$

$C\langle \lambda x.E\rangle = \lambda k.k\ (\lambda x.C\langle E\rangle)$

$C\langle E_0\ E_1\rangle = \lambda k.C\langle E_0\rangle(\lambda v_0.C\langle E_1\rangle(\lambda v_1.v_0 v_1 k))$

$C\langle E_1{+}E_2\rangle = \lambda k.C\langle E_1\rangle(\lambda v_1.C\langle E_2\rangle(\lambda v_2.k\ (v_1{+}v_2))$

CPS transformatsioon

- Kui algjätkuna kasutada identsusfunktsiooni, siis transformatsiooni tulemusena saadud term on ekvivalentne esialges termiga:

$$C\langle E\rangle(\lambda k.k) = E$$

call-by-value CPS transformatsioon - näide

$C\langle$`(λx.x) ((λx.x) 3)`$\rangle$

$\rightarrow$ `λk.`$C\langle$`(λx.x)`$\rangle$`(λu.`$C\langle$`(λx.x) 3`$\rangle$ `(λv.u v k))`

$\rightarrow$ `λk.(λl.l(λx.`$C\langle$`x`$\rangle$`)) (λu.`$C\langle$`(λx.x) 3`$\rangle$ `(λv.u v k))`

$\rightarrow$ `λk.(λl.l(λx.(λm.m x))) (λu.`$C\langle$`(λx.x) 3`$\rangle$
`(λv.u v k))`

$\rightarrow$ `λk.(λl.l(λx.(λm.m x))) (λu.(λw.`$C\langle$`λx.x`$\rangle$`(λz.`$C\langle$`3`$\rangle$
`(λt.z t w))) (λv.u v k))`

$\rightarrow$ `λk.(λl.l(λx.(λm.m x))) (λu.(λw.(λs.s(λx.`$C\langle$`x`$\rangle$`)))`
`(λz.`$C\langle$`3`$\rangle$`(λt.z t w))) (λv.u v k))`

$\rightarrow$ `λk.(λl.l(λx.(λm.m x))) (λu.(λw.(λs.s(λx.`
`(λq.q x)))(λz.(λr.r 3)(λt.z t w))) (λv.u v k))`

CPS kujul termi redutseerimine - näide

→ (λk.(λl.l(λx.(λm.m x))) (λu.(λw.(λs.s(λx.(λq.q x)))(λz.(λr.r 3)(λt.z t w)))(λv.u v k))) (λx.x)

→ (λl.l(λx.(λm.m x))) (λu.(λw.(λs.s(λx.(λq.q x)))(λz.(λr.r 3)(λt.z t w)))(λv.u v (λx.x))

→ (λu.(λw.(λs.s(λx.(λq.q x)))(λz.(λr.r 3)(λt.z t w))) (λv.u v (λx.x))) (λx.(λm.m x))

→ (λw.(λs.s(λx.(λq.q x)))(λz.(λr.r 3)(λt.z t w))) (λv.(λx.(λm.m x)) v (λx.x))

→ (λs.s(λx.(λq.q x)))(λz.(λr.r 3)(λt.z t (λv.(λx.(λm.m x)) v (λx.x))))

CPS kujul termi redutseerimine – näide (jätkub)

→ (λz.(λr.r 3)(λt.z t (λv.(λx.(λm.m x)) v (λx.x))))
(λx.(λq.q x))

→ (λr.r 3)(λt. (λx.(λq.q x)) t (λv.(λx.(λm.m x)) v
(λx.x)))

→ (λt. (λx.(λq.q x)) t (λv.(λx.(λm.m x)) v (λx.x))) 3

→ (λx.(λq.q x)) 3 (λv.(λx.(λm.m x)) v (λx.x))

→ (λq.q 3) (λv.(λx.(λm.m x)) v (λx.x))

→ (λv.(λx.(λm.m x)) v (λx.x)) 3

→ (λx.(λm.m x)) 3 (λx.x)

→ (λm.m 3) (λx.x)

→ (λx.x) 3

→ 3

CPS kujul termi redutseerimine

- CPS kujul termi redutseerimisel on igal redutseerimissammul täpselt üks reedeks.
- CPS kujul avaldisse on redutseerimisstrateegia sisse kirjutatud.

call-by-name CPS transformatsioon

- *call-by-name* transformatsiooni $D\langle\rangle$ reeglid

$D\langle c\rangle = \lambda k.k\ c$

$D\langle x\rangle = \lambda k.x\ k$

$D\langle \text{if } B\ E_1\ E_2\rangle = \lambda k.\ D\langle B\rangle(\lambda b.\ \text{if } b\ (D\langle E_1\rangle\ k)\ (D\langle\ E_2\rangle\ k))$

$D\langle \lambda x.E\rangle = \lambda k.k\ (\lambda x.\ D\langle E\rangle)$

$D\langle E_0\ E_1\rangle = \lambda k.\ D\langle E_0\rangle(\lambda v_0.\ v_0\ D\langle E_1\rangle\ k)$

call-by-name CPS transformatsioon - näide

$D\langle(\lambda x.x)\ ((\lambda x.x)\ 3)\rangle$

$\rightarrow \lambda k.D\langle(\lambda x.x)\rangle\ (\lambda v.v\ D\langle(\lambda x.x)\ 3\rangle\ k)$

$\rightarrow \lambda k.(\lambda l.l\ (\lambda x.D\langle x\rangle))\ (\lambda v.v\ D\langle(\lambda x.x)\ 3\rangle\ k)$

$\rightarrow \lambda k.(\lambda l.l\ (\lambda x.(\lambda m.x\ m)))\ (\lambda v.v\ D\langle(\lambda x.x)\ 3\rangle\ k)$

$\rightarrow \lambda k.(\lambda l.l\ (\lambda x.(\lambda m.x\ m)))\ (\lambda v.v\ (\lambda u.D\langle\lambda x.x\rangle$
$(\lambda w.w\ D\langle 3\rangle\ u))\ k)$

$\rightarrow \lambda k.(\lambda l.l\ (\lambda x.(\lambda m.x\ m)))$
$(\lambda v.v\ (\lambda u.(\lambda z.z(\lambda x.D\langle x\rangle))(\lambda w.w\ D\langle 3\rangle\ u))\ k)$

$\rightarrow \lambda k.(\lambda l.l\ (\lambda x.(\lambda m.x\ m)))\ (\lambda v.v\ (\lambda u.(\lambda z.z$
$(\lambda x.\lambda t.x\ t))(\lambda w.w\ (\lambda s.s\ 3)\ u))\ k)$

`call/cc` transformatsiooni-reegel (*call-by-value*)

$$C\langle \texttt{call/cc}\ E\rangle = \lambda k.\ C\langle E\rangle(\lambda f.f\ k\ k)$$

$$C\langle \texttt{throw}\ E_0\ E_1\rangle = \lambda k.\ C\langle E_0\rangle(\lambda k'.\ C\langle E_1\rangle(\lambda v.k'\ v))$$

`call/cc` transformatsioon - näide

$C\langle$`call/cc (λl.2 + (throw l 3))`$\rangle$

$\rightarrow$ `λk.(`$C\langle$`λl.2 + (l 3)`$\rangle$`) (λf.f k k)`

$\rightarrow$ `λk.(λm.m (λl.`$C\langle$`2 + (throw l 3)`$\rangle$`)) (λf.f k k)`

$\rightarrow$ `λk.(λm.m (λl.(λi.`$C\langle$`2`$\rangle$`(λj.(λa.`$C\langle$`l`$\rangle$`(λb.`$C\langle$`3`$\rangle$
`(λc.b c)))(λn.i (j+n))))))(λf.f k k)`

$\rightarrow$ `λk.(λm.m (λl.(λi.(λu.u 2)(λj.(λa. (λz.z l)`
`(λb. (λt.t 3)(λc.b c)))(λn.i (j+n))))))(λf.f k k)`

`call/cc` transformatsioon - näide

```
(λk.(λm.m (λl.(λi.(λu.u 2)(λj.(λa. (λz.z l)(λb. (λt.t 3)(λc.b c)))
   (λn.i (j+n))))))(λf.f k k)) (λ x.x)
→ (λm.m (λl.(λi.(λu.u 2)(λj.(λa. (λz.z l)(λb. (λt.t 3)(λc.b c)))
   (λn.i (j+n))))))(λf.f (λ x.x) (λ x.x))
→ (λf.f (λ x.x) (λ x.x)) (λl.(λi.(λu.u 2)(λj.(λa. (λz.z l)
   (λb. (λt.t 3)(λc.b c)))(λn.i (j+n)))))
→ (λl.(λi.(λu.u 2)(λj.(λa. (λz.z l)(λb. (λt.t 3)(λc.b c)))
   (λn.i (j+n))))) (λ x.x) (λ x.x)
→ (λi.(λu.u 2)(λj.(λa. (λz.z (λ x.x))(λb. (λt.t 3)(λc.b c)))
   (λn.i (j+n)))) (λ x.x)
→ (λu.u 2)(λj.(λa. (λz.z (λ x.x))(λb. (λt.t 3)(λc.b c)))
   λn. (λ x.x)(j+n)))
→ (λj.(λa. (λz.z (λ x.x))(λb. (λt.t 3)(λc.b c)))(λn. (λ x.x)(j+n))) 2
→ (λa. (λz.z (λ x.x))(λb. (λt.t 3)(λc.b c)))(λn.(λx.x)(2+n))
→ (λz.z (λ x.x))(λb. (λt.t 3)(λc.b c))
→ (λb. (λt.t 3)(λc.b c)) (λ x.x)
→ (λt.t 3)(λc. (λ x.x) c)
→ (λc. (λ x.x) c) 3
→ (λ x.x) 3
→ 3
```

Otsese denotatsioonsemantika puudused

- Ei arvesta, et arvutus võib ebaõnnestuda
  - Võimalik semantikareegleid laiendada nii, et arvestatakse ka arvutuse ebaõnnestumist.
  - Tulemuseks väga keerulised reeglid.
- Ei määra reduktsioonijärjekorda:
  - let t=1 div 0 in 2

Otsese denotatsioonsemantika puudused - näide

- $S = \ldots | S_1 \text{ div } S_2$
- $S_d[\![S_1 \text{ div } S_2]\!] s = S_d[\![S_1]\!] s / S_d[\![S_2]\!] s$
- S_2=0?
- Toome hulka Int uue elemendi $\perp$
- $S_d[\![S_1 \text{ div } S_2]\!] s = \text{let } d = S_d[\![S_2]\!] s \text{ in if } d=0 \text{ then } \perp \text{ else } S_d[\![S_1]\!] s / d$
- $\perp$ käsitlemine tuleb sisse viia kõigisse semantikareeglitesse
- Üldisemalt vigade käsitlemiseks (try konstruktsioon) on vaja ka eraldi “error value” sisse tuua ning samuti näidata selle käsitlemine kõigis semantikareeglites.

While keele CPS semantika

- CPS semantikas on jätk osaline funktsioon

$$\mathrm{Cont} = \mathrm{State} \hookrightarrow \mathrm{State}$$

While keele CPS semantika

- Keele While semantiline funktsioon:
  S_{cs}: Stm→(Cont→Cont)

$$S_{cs}[\![x{:=}a]\!]\ c\ s = c\ (s\ [x \mapsto A[\![a]\!]\ s])$$

$$S_{cs}[\![\texttt{skip}]\!] = \texttt{id}$$

$$S_{cs}[\![S_1; S_2]\!] = S_{cs}[\![S_1]\!] \circ S_{cs}[\![S_2]\!]$$

$$S_{cs}[\![\texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2]\!]\ c = \texttt{cond}(B[\![b]\!], S_{cs}[\![S_1]\!]\ c, S_{cs}[\![S_2]\!]\ c)$$

$$S_{cs}[\![\texttt{while}\ b\ \texttt{do}\ S]\!] = \mathrm{FIX}\ G$$

$$\texttt{where}\ (G\ g)\ c = \texttt{cond}(B[\![b]\!], S_{cs}[\![S_1]\!]\ (g\ c), c)$$

While keele CPS semantika - näide

$S_{cs}[\![\texttt{z:=x;x:=y;y:=z}]\!]\ \texttt{id}$

$= (S_{cs}[\![\texttt{z:=x}]\!] \circ S_{cs}[\![\texttt{x:=y}]\!] \circ S_{cs}[\![\texttt{y:=z}]\!])\ \texttt{id}$

$= (S_{cs}[\![\texttt{z:=x}]\!] \circ S_{cs}[\![\texttt{x:=y}]\!])\ g_1$

where $g_1\ s = \texttt{id}(s[\texttt{y}\mapsto(s\ \texttt{z})])$

$= S_{cs}[\![\texttt{z:=x}]\!]\ g_2$

where $g_2\ s = g_1(s[\texttt{y}\mapsto(s\ \texttt{y})])$

$= \texttt{id}(s[\texttt{y}\mapsto(s\ \texttt{y})][\texttt{y}\mapsto(s\ \texttt{z})])$

$= g_3$

where $g_3\ s = g2(s[\texttt{z}\mapsto(s\ \texttt{x})])$

$= \texttt{id}(s[\texttt{z}\mapsto(s\ \texttt{x})][\texttt{y}\mapsto(s\ \texttt{y})][\texttt{y}\mapsto(s\ \texttt{z})])$

While keele CPS semantika

- Semantikareeglite abil defineeritud funktsioon S_{cs} on täielik.

While keele CPS semantika

- Ülesanne. Laiendada While keele CPS semantikat uue konstruktsiooniga `repeat` S `until` b ja lisada uus (kompositsiooniline) semantikareegel.
- $S_{cs}[\![\texttt{repeat}\ S\ \texttt{until}\ b]\!] = S_{cs}[\![S]\!] \circ S_{cs}[\![\texttt{while}\ b\ \texttt{do}\ S]\!]$

Ext keele CPS semantika

- Laiendatud While keel Ext erinditega:

  $S = x := a \mid$ `skip` $\mid S_1 ; S_2 \mid$ `if` b `then` S_1 `else` $S_2 \mid$ `while` b `do` $S \mid$ `begin` S_1 `handle` e`:` S_2 `end`

  $\mid$ `raise` e
- Muutuja e kuulub süntaktilisse kategooriasse Exception
- Näide:

```
begin while true do if x≤0
                    then raise exit
                    else x:=x-1
     handle exit: y:=7
end
```

Ext keele CPS semantika

- Erindite käsitlemiseks toome sisse erindite keskkonna:

$$\mathrm{Env}_E = \mathrm{Exception} \rightarrow \mathrm{Cont}$$

- Semantiline funktsioon

$$S_{cs}\colon \mathrm{Stm} \rightarrow \mathrm{Env}_E \rightarrow (\mathrm{Cont} \rightarrow \mathrm{Cont})$$

Ext keele CPS semantika

- Uued semantikareeglid

$$S_{cs}[\![x := a]\!]\, env_E\, c\, s = c\,(s\,[x \mapsto A[\![a]\!]\, s])$$

$$S_{cs}[\![\texttt{skip}]\!]\, env_E = \texttt{id}$$

$$S_{cs}[\![S_1; S_2]\!]\, env_E = (S_{cs}[\![S_1]\!]\, env_E) \circ (S_{cs}[\![S_2]\!]\, env_E)$$

$$S_{cs}[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2]\!]\, env_E\, c$$
$$= \texttt{cond}(B[\![b]\!], S_{cs}[\![S_1]\!]\, env_E\, c, S_{cs}[\![S_2]\!]\, env_E\, c)$$

$$S_{cs}[\![\texttt{while } b \texttt{ do } S]\!]\, env_E = \text{FIX } G$$
$$\texttt{where } (G\, g)\, c = \texttt{cond}(B[\![b]\!], S_{cs}[\![S_1]\!]\, env_E\, (g\, c), c)$$

$$S_{cs}[\![\texttt{begin } S_1 \texttt{ handle } e\texttt{: } S_2 \texttt{ end}]\!]\, env_E\, c =$$
$$S_{cs}[\![S_1]\!]\, (env_E\, [e \mapsto S_{cs}[\![S_2]\!]\, env_E\, c])\, c$$

$$S_{cs}[\![\texttt{raise } e]\!]\, env_E\, c = env_E\, e$$

Ext keele CPS semantika - näide

$S_{cs}[\![$`begin while true do if x≤0 then raise exit else x:=x-1 handle exit: y:=7 end`$]\!]\ env_E$ `id`

$= (\mathrm{FIX}\ G)$ `id`

where $G\ g\ c\ s =$ `cond(`$B[\![$`true`$]\!]$`, cond(`$B[\![$`x≤0`$]\!]$`,` c_{exit}`,`
$S_{cs}[\![$ `x:=x-1`$]\!]\ env_E$ `[exit↦`$\ c_{\mathrm{exit}}$`]` $(g\ c))$, $c)\ s$

$$= \begin{cases} c_{\mathrm{exit}}\ s & \text{if } s\ \texttt{x≤0} \\ (g\ c)\ (s[\texttt{x}\mapsto (s\ \texttt{x}\ ) - 1]) & \text{if } s\ \texttt{x>0} \end{cases}$$

$c_{\mathrm{exit}}\ s =$ `id` $(s[\texttt{y}\mapsto 7]) = s[\texttt{y}\mapsto 7]$

$$(\mathrm{FIX}\ G)\ \texttt{id}\ \ s = \begin{cases} s[\texttt{y}\mapsto 7] & \text{if } s\ \texttt{x≤0} \\ s[\texttt{x}\mapsto 0][\texttt{y}\mapsto 7] & \text{if } s\ \texttt{x≤0} \end{cases}$$

Ext keele CPS semantika

- Ülesanne. Oletame, et meil on muutuja `out`, mille väärtus on kogu programmi väärtuseks. Siis saame defineerida

$$\text{Cont} = \text{State} \hookrightarrow Z$$

  Defineerida algjätk $c_0 \in \text{Cont}$. Millised muudatused tuleb teha keskkonnas Env_E, S_{cs} funktsionaalsuses ja semantikareeglites?