Jezyki programowania i kontynuacje

2. Semantyka operatorow kontroli

Dariusz Biernacki

dabi@ii.uni.wroc.pl

Instytut Informatyki

Uniwersytet Wrocławski

Semantyka operacyjna a kontynuacje

- Formalizmy takie jak Semantyka Naturalna czy Strukturalna Semantyka Operacyjna (SOS) nie sa odpowiednie
- Istotna jest informacja o kontekscie (calosci programu), a nie tylko o ewaluowanym podwyrazeniu

Semantyka operacyjna operatorow

- Ewaluator w CPS
  - wyzszego rzedu (impl. semantyki kontynuacyjnej)
  - pierwszego rzedu (impl. maszyny abstrakcyjnej)
- Maszyna abstrakcyjna
  - system przejsc pierwszego rzedu
  - semantyka malych krokow
- Semantyka redukcyjna
  - czysto syntaktyczna, oparta na przepisywaniu calego programu
  - semantyka malych krokow
- Transformacja do CPS + ewaluator

Modelowy jezyk funkcyjny ISWIM

Beztypowy lambda rachunek rozszerzony o stale numeryczne i funkcje nastepnika.

$$e ::= x \mid \lambda x.e \mid e_0\, e_1 \mid \ulcorner m \urcorner \mid succ\ e$$

Program to zamkniete wyrazenie.

Ewaluatory (interpretery definiujace)

- Semantyka jezyka definiowanego (object language) jest definiowana przez interpreter napisany w jezyku definiujacym (metalanguage)

- W przypadku funkcyjnego jezyka definiujacego
  - wartosci funkcyjne jezyka definiowanego moga byc reprezentowane przez funkcje jezyka definiujacego
  - interpreter moze byc wyrazony w CPS
  - interpreter moze byc kompozycyjny (moze implementowac semantyke denotacyjna jezyka)

Ewaluator CBV dla ISWIM (1)

```
type ide = string
datatype exp = VAR of ide
             | LAM of ide * exp
             | APP of exp * exp
             | LIT of int
             | SUCC of exp
signature ENV = sig
  type 'a env
  val mt : 'a env
  val lookup : 'a env * ide -> 'a
  val extend : 'a env * ide * 'a -> 'a env
end
```

Ewaluator CBV dla ISWIM (2)

```
structure Env : ENV = struct
  type 'a env = (ide * 'a) list
  val mt = []
  exception IDENTIFIER_NOT_BOUND
  fun lookup (e, x)
      = let fun walk []
                = raise IDENTIFIER_NOT_BOUND
            | walk ((y,v)::e')
                = if x = y then v else walk e'
        in walk e end
  fun extend (e, x, v) = (x,v)::e
end
```

Ewaluator CBV dla ISWIM (3)

```
datatype value = INT of int
               | FN of value -> value

exception TYPE_ERROR

fun eval (VAR x, env)
    = Env.lookup (env, x)
  | eval (LAM (x,e), env)
    = FN (fn v =>
            eval (e, Env.extend (env, x, v)))
```

Ewaluator CBV dla ISWIM (4)

```
  | eval (APP (e0, e1), env)
    = let val v0 = eval (e0, env)
          val v1 = eval (e1, env)
      in (case v0 of
            FN f => f v1
          | _ => raise TYPE_ERROR) end
  | eval (LIT n, env) = INT n
  | eval (SUCC e, env)
    = let val v = eval (e, env)
      in (case v of INT n => (INT (n+1))
                  | _ => raise TYPE_ERROR) end
fun evaluate e = eval (e, Env.mt)
```

Ewaluator CBN dla ISWIM (1)

```
datatype value = INT of int
               | FN of comp -> value
withtype comp = unit -> value

exception TYPE_ERROR

fun eval (VAR x, env)
    = Env.lookup (env, x) ()
  | eval (LAM (x,e), env)
    = FN (fn c =>
            eval (e, Env.extend (env, x, c)))
```

Ewaluator CBN dla ISWIM (2)

```
  | eval (APP (e0, e1), env)
    = let val v0 = eval (e0, env)
      in (case v0 of
            FN f => f (fn () => eval (e1, env))
          | _ => raise TYPE_ERROR) end
  | eval (LIT n, env) = INT n
  | eval (SUCC e, env)
    = let val v = eval (e, env)
      in (case v of INT n => (INT (n+1))
                  | _ => raise TYPE_ERROR) end

fun evaluate e = eval (e, Env.mt)
```

Maszyny abstrakcyjne

- system przejsc pierwszego rzedu sluzacy do ewaluacji programow
- konfiguracje (stany) maszyny abstrakcyjnej okreslaja stan obliczen na danym etapie
- przejscia sa zdefiniowane przez binarna relacje okreslona na zbiorze konfiguracji
- definiuje semantyke malych krokow
- stanowi etap posredni miedzy np. semantyka redukcyjna a implementacja jezyka (wykorzystuje srodowisko, stos, etc.)

Maszyna abstrakcyjna CBV dla ISWIM (1)

Kanoniczna maszyna abstrakcyjna CBV zwana CEK:

- Jezyk: $e \quad ::= \quad x \mid \lambda x.e \mid e_0\, e_1 \mid \ulcorner m \urcorner \mid succ\ e$
- Wartosci:

$$v ::= m \mid [x,\ e,\ \rho]$$

- Srodowisko: $\rho ::= \rho_{mt} \mid \rho\{x \mapsto v\}$
- Kontekst (stos):

$$E ::= \mathsf{STOP} \mid \mathsf{ARG}\,((e,\ \rho),\ E) \mid \mathsf{SUCC}\,(E) \mid \mathsf{FUN}\,(v,\ E)$$

Maszyna abstrakcyjna CBV dla ISWIM (2)

- Relacja przejscia:

e	$\Rightarrow$	$\langle e,\ \rho_{mt},\ \mathsf{STOP}\rangle_{eval}$
$\langle x,\ \rho,\ E\rangle_{eval}$	$\Rightarrow$	$\langle E,\ \rho(x)\rangle_{cont}$
$\langle \lambda x.e,\ \rho,\ E\rangle_{eval}$	$\Rightarrow$	$\langle E,\ [x,\ e,\ \rho]\rangle_{cont}$
$\langle e_0\ e_1,\ \rho,\ E\rangle_{eval}$	$\Rightarrow$	$\langle e_0,\ \rho,\ \mathsf{ARG}\ ((e_1,\ \rho),\ E)\rangle_{eval}$
$\langle \ulcorner m \urcorner,\ \rho,\ E\rangle_{eval}$	$\Rightarrow$	$\langle E,\ m\rangle_{cont}$
$\langle succ\ e,\ \rho,\ E\rangle_{eval}$	$\Rightarrow$	$\langle e,\ \rho,\ \mathsf{SUCC}\ (E)\rangle_{eval}$
$\langle \mathsf{ARG}\ ((e_1,\ \rho),\ E),\ v\rangle_{cont}$	$\Rightarrow$	$\langle e_1,\ \rho,\ \mathsf{FUN}\ (v,\ E)\rangle_{eval}$
$\langle \mathsf{SUCC}\ (E),\ m\rangle_{cont}$	$\Rightarrow$	$\langle E,\ m+1\rangle_{cont}$
$\langle \mathsf{FUN}\ ([x,\ e,\ \rho],\ E),\ v\rangle_{cont}$	$\Rightarrow$	$\langle e,\ \rho\{x \mapsto v\},\ E\rangle_{eval}$
$\langle \mathsf{STOP},\ v\rangle_{cont}$	$\Rightarrow$	v

Maszyna abstrakcyjna CBV dla ISWIM (3)

Funkcja ewaluacyjna definiowana przez maszyne abstrakcyjna CEK:

Definicja 1

$$\mathsf{eval}^{\mathsf{v}}_{\mathsf{AM}}(e) \stackrel{\mathrm{def}}{=} v \textit{ wtw } e \Rightarrow^{+} v.$$

Maszyna abstrakcyjna CBN dla ISWIM (1)

Kanoniczna maszyna abstrakcyjna CBN zwana maszyna Krivine'a:

- Jezyk: $e \quad ::= \quad x \mid \lambda x.e \mid e_0\, e_1 \mid \ulcorner m \urcorner \mid succ\ e \mid$
- Wartosci:

$$v ::= m \mid [x,\ e,\ \rho]$$

- Domkniecia:

$$c ::= [e,\ \rho]$$

- Srodowisko:

$$\rho ::= \rho_{mt} \mid \rho\{x \mapsto c\}$$

- Kontekst (stos):

$$E ::= \mathsf{STOP} \mid \mathsf{ARG}\,(c,\ E) \mid \mathsf{SUCC}\,(E)$$

Maszyna abstrakcyjna CBN dla ISWIM (2)

- Relacja przejscia:

$$
\begin{array}{rcl}
e & \Rightarrow & \langle e,\ \rho_{mt},\ \mathsf{STOP}\rangle_{eval} \\
\hline
\langle x,\ \rho,\ E\rangle_{eval} & \Rightarrow & \langle e,\ \rho',\ E\rangle_{eval} \quad \rho(x) = [e,\ \rho'] \\
\langle \lambda x.e,\ \rho,\ E\rangle_{eval} & \Rightarrow & \langle E,\ [x,\ e,\ \rho]\rangle_{cont} \\
\langle e_0\, e_1,\ \rho,\ E\rangle_{eval} & \Rightarrow & \langle e_0,\ \rho,\ \mathsf{ARG}\,([e_1,\ \rho],\ E)\rangle_{eval} \\
\langle \ulcorner m \urcorner,\ \rho,\ E\rangle_{eval} & \Rightarrow & \langle E,\ m\rangle_{cont} \\
\langle succ\ e,\ \rho,\ E\rangle_{eval} & \Rightarrow & \langle e,\ \rho,\ \mathsf{SUCC}\,(E)\rangle_{eval} \\
\hline
\langle \mathsf{ARG}\,(c,\ E),\ [x,\ e,\ \rho]\rangle_{cont} & \Rightarrow & \langle e,\ \rho\{x \mapsto c\},\ E\rangle_{eval} \\
\langle \mathsf{SUCC}\,(E),\ m\rangle_{cont} & \Rightarrow & \langle E,\ m+1\rangle_{cont} \\
\hline
\langle \mathsf{STOP},\ v\rangle_{cont} & \Rightarrow & v
\end{array}
$$

Maszyna abstrakcyjna CBN dla ISWIM (3)

Funkcja ewaluacyjna definiowana przez maszyne abstrakcyjna Krivine'a:

Definicja 2

$$\mathsf{eval}^{\mathsf{n}}_{\mathsf{AM}}(e) \stackrel{\mathrm{def}}{=} v \textit{ wtw } e \Rightarrow^{+} v.$$

Od ewaluatora do MA (1)

Funkcyjna odpowiedniosc miedzy ewaluatorem a maszyna abstrakcyjna:

1. defunkcjonalizacja wartosci funkcyjnych produkowanych przez ewaluator – wprowadzenie domkniec
2. CPS transformacja ewaluatora – sekwencjalizacja obliczen
3. defunkcjonalizacja kontynuacji – wprowadzenie stosu

Od ewaluatora do MA (2)

Przyklady

1. kanoniczny ewaluator CBV —> CEK
2. kanoniczny ewaluator CBN —> maszyna Krivine'a

Konstrukcja ta jest aplikowalna rowniez w przypadku ewaluatorow dla jezykow nalezacych do innych paradygmatow programowania, m.in., logicznego, imperatywnego i obiektowego.

Przesledzmy ja na przykladzie naszego ewaluatora CBV.

1.Zmiana reprezentacji funkcji (1)

Defunkcjonalizujemy przestrzen funkcji `value -> value` w ewaluatorze CBV. Otrzymany ewaluator operuje na domknieciach funkcji (kod funkcji + srodowisko okreslajace wartosc zmiennych wolnych).

```
datatype value = INT of int
               | FN of ide * exp * value Env.env

    ...
  | eval (LAM (x,e), env)
    = FN (x, e, env)
    ...
```

1.Zmiana reprezentacji funkcji (2)

```
| eval (APP (e0, e1), env)
  = let val v0 = eval (e0, env)
        val v1 = eval (e1, env)
    in (case v0 of
          FN (x, e, env) =>
            eval (e, Env.extend (env, x, v1))
        | _ => raise TYPE_ERROR)
    end
```

2.Sekwencjalizacja obliczen (1)

Transformujemy ewaluator do stylu kontynuacyjnego, co uwydatnia porzadek obliczen.

```
datatype value = INT of int
               | FN of ide * exp * value Env.env
withtype cont = value -> value

fun eval (VAR x, env, k)
    = k (Env.lookup (env, x))
  | eval (LAM (x,e), env, k)
    = k (FN (x, e, env))
```

2.Sekwencjalizacja obliczen (2)

```
| eval (APP (e0, e1), env, k) =
  eval (e0, env,
   fn v0 =>
    eval (e1, env,
     fn v1 => case v0 of
                FN (x, e, env) =>
                 eval (e,
                       Env.extend (env, x, v1), k)
              | _ => raise TYPE_ERROR))
```

3.Zmiana repr. kontynuacji (1)

```
datatype value = INT of int
               | FN of ide * exp * value Env.env
and cont = STOP
         | ARG of exp * value Env.env * cont
         | FUN of value * cont
         | SUC of cont

fun eval (VAR x, env, k)
    = apply (k, Env.lookup (env, x))
  | eval (LAM (x,e), env, k)
    = apply (k, FN (x, e, env))
  | eval (APP (e0, e1), env, k)
    = eval (e0, env, ARG (e1, env, k))
```

3.Zmiana repr. kontynuacji (2)

```
    | eval (LIT n, env, k)
      = apply (k, INT n)
    | eval (SUCC e, env, k)
      = eval (e, env, SUCC k)
  and apply (STOP, v) = v
    | apply (ARG (e1, env, k), v0)
      = eval (e1, env, (FUN (v0, k)))
    | apply (FUN (FN (x, e, env), k), v1)
      = eval (e, Env.extend (env, x, v1), k)
    | apply (SUC k, INT n)
      = apply (k, INT (n+1))

  fun evaluate e = eval (e, Env.mt, STOP)
```

Uwagi

- Kolejnosc krokow 1. i 2. w transformacji moze byc odwrotna
- W przypadku gdy ewaluator bazowy nie produkuje wartosci funkcyjnych, krok 1. jest pomijany
- W przypadku gdy ewaluator bazowy jest wyrazony w stylu kontynuacyjnym (np. operatory), krok 2. jest pomijany
- Moze zajsc koniecznosc wielokrotnego zastosowania kroku 2. (np. kontynuacje ograniczone)
- **Transformacja jest odwracalna: jezeli dana maszyna abstrakcyjna jest w zdefunkcjonalizowanej formie, to refunkcjonalizacja kontynuacji, a nastepnie transformacja ze stylu kontynuacyjnego prowadzi do ewaluatora**

Semantyka redukcyjna (1)

- czysto syntaktyczna semantyka malych krokow
- oparta na przepisywaniu termow (przepisywany jest tekst programu)
- definiowane sa nastepujace komponenty:
  - termy (wyrazenia), a wsrod nich wartosci – mozliwy wynik obliczen
  - redeksy (potencjalne i faktyczne) – postac wyrazen, ktore moga byc zredukowane
  - konteksty ewaluacyjne – programy z “dziura”, okreslajace gdzie w programie moze nastapic redukcja
  - reguly redukcji (jak redukuje sie redeks danej postaci)

Semantyka redukcyjna (2)

- Lemat o jednoznacznym rozkladzie

  Lemat 1 *Kazdy program albo jest wartoscia, albo mozna go jednoznacznie rozlozyc na potencjalny redeks i kontekst.*

- Jeden krok redukcji polega na
  - dekompozycji termu na kontekst i redeks
  - kontrakcji redeksu
  - wlozeniu wyniku kotrakcji do kontekstu
- Ewaluacja definiowana jest jako zwrotno-przechodnie domkniecie relacji redukcji

Semantyka redukcyjna CBV dla ISWIM (1)

- Wyrazenia i wartosci

$$\begin{array}{lcl} e & ::= & x \mid v \mid e_0\, e_1 \mid succ\ e \\ v & ::= & \lambda x.e \mid \ulcorner m \urcorner \end{array}$$

- Potencjalne i faktyczne redeksy

$$p ::= v_0\, v_1 \mid succ\ v$$

$$r ::= (\lambda x.e)\, v \mid succ\ \ulcorner m \urcorner$$

- Konteksty ewaluacyjne

$$E ::= [\,] \mid E\,[[\,]\,e] \mid E\,[succ\,[\,]] \mid E\,[v\,[\,]]$$

Semantyka redukcyjna CBV dla ISWIM (2)

- Reguły redukcji

$$\begin{array}{lrcl} (\beta_\lambda) & (\lambda x.e)\, v & \rightarrow & e\{v/x\} \\ (succ) & succ\ \ulcorner m \urcorner & \rightarrow & \ulcorner m+1 \urcorner \end{array}$$

- Oznaczamy przez $\mapsto$ domknięcie relacji $\rightarrow$ kompatybilne ze względu na konteksty ewaluacyjne
- Funkcja ewaluacyjna

$$\mathsf{eval_v}(e) \stackrel{\mathrm{def}}{=} v \text{ wtw } e \mapsto^* v.$$

Semantyka redukcyjna CBN dla ISWIM (1)

- Wyrazenia i wartosci

$$\begin{array}{lcl} e & ::= & x \mid v \mid e_0\, e_1 \mid succ\ e \\ v & ::= & \lambda x.e \mid \ulcorner m \urcorner \end{array}$$

- Potencjalne i faktyczne redeksy

$$p ::= v\, e \mid succ\ v$$

$$r ::= (\lambda x.e)\, e' \mid succ\ \ulcorner m \urcorner$$

- Konteksty ewaluacyjne

$$E ::= [\,] \mid E\,[[\,]\,e] \mid E\,[succ\,[\,]]$$

Semantyka redukcyjna CBN dla ISWIM (2)

- Reguły redukcji

$$\begin{array}{lrcl} (\beta_\lambda) & (\lambda x.e)\, e' & \rightarrow & e\{e'/x\} \\ (succ) & succ\ \ulcorner m \urcorner & \rightarrow & \ulcorner m+1 \urcorner \end{array}$$

- Oznaczamy przez $\mapsto$ domknięcie relacji $\rightarrow$ kompatybilne ze względu na konteksty ewaluacyjne
- Funkcja ewaluacyjna

$$\mathsf{eval_n}(e) \stackrel{\mathrm{def}}{=} v \text{ wtw } e \mapsto^* v.$$

Od MA do SR i z powrotem (1)

- Maszyna abstrakcyjna to zoptymalizowana semantyka redukcyjna
  - W SR term jest dekomponowany, nastepuje kontrakcja redeksu, a nastepnie odbudowa termu, po czym ten proces sie powtarza
  - W MA term jest dekomponowany, nastepuje kontrakcja redeksu, a nastepnie, bez odbudowywania calego termu, poszukiwany jest kolejny redeks
  - Dodatkowa optymalizacja jest uzycie srodowiska zamiast podstawienia, ale czasem rozwaza sie MA oparte na podstawieniu

Od MA do SR i z powrotem (2)

- Z danej MA mozna wyekstraktowac SR
  - semantyczne komponenty zastepujemy syntaktycznymi
  - odczytujemy reguly redukcji z przejsc maszyny – redukcja nastepuje tam gdzie lewa i prawa strona reprezentuja rozne termy
- Z danej SR mozna mechanicznie wyprowadzic MA
  - metoda zwana “refocusing”
  - polega na fuzji funkcji wkladajacej term do kontekstu z funkcja dekomponujaca term
  - w celu otrzymania MA ze srodowiskiem, startujemy z SR z jawnym podstawieniem

Od MA do SR i z powrotem (3)

Twierdzenie 1 *Dla kazdego programu* e *oraz literalu* $\ulcorner m \urcorner$*,*

1. $\mathsf{eval}_{\mathsf{v}}(e) = \ulcorner m \urcorner$ *wtw* $\mathsf{eval}^{\mathsf{v}}_{\mathsf{AM}}(e) = \ulcorner m \urcorner$
2. $\mathsf{eval}_{\mathsf{n}}(e) = \ulcorner m \urcorner$ *wtw* $\mathsf{eval}^{\mathsf{n}}_{\mathsf{AM}}(e) = \ulcorner m \urcorner$

Transformacja do CPS formalnie

- Semantyka programow funkcyjnych, a w szczegolnosci tych z operatorami, mozna definiowac poprzez transformacje do stylu kontynuacyjnego

- Istnieje wiele formalnie zdefiniowanych transformacji do CPS

Transformacja CBV do CPS

$$\overline{x} = \lambda k.k\, x$$

$$\overline{\overline{\lambda x.e}} = \lambda k.k\, (\lambda x k.\, \overline{e}\, k)$$

$$\overline{\overline{e_0\, e_1}} = \lambda k.\overline{\overline{e_0}}\, (\lambda v_0.\, \overline{\overline{e_1}}\, (\lambda v_1.\, v_0\, v_1\, k))$$

$$\overline{\overline{\ulcorner m \urcorner}} = \lambda k.k \ulcorner m \urcorner$$

$$\overline{\overline{succ\ e}} = \lambda k.\overline{e}\, (\lambda v.\, k\, (succ\ v))$$

Transformacja CBN do CPS

$$\overline{\overline{x}} = \lambda k.x\ k$$

$$\overline{\overline{\lambda x.e}} = \lambda k.k\ (\lambda x k.\ \overline{\overline{e}}\ k)$$

$$\overline{\overline{e_0\ e_1}} = \lambda k.\overline{\overline{e_0}}\ (\lambda v_0.\ v_0\ \overline{\overline{e_1}}\ k)$$

$$\overline{\overline{\ulcorner m \urcorner}} = \lambda k.k \ulcorner m \urcorner$$

$$\overline{\overline{succ\ e}} = \lambda k.\overline{\overline{e}}\ (\lambda v.\ k\ (succ\ v))$$

Wzajemna symulacja CBV i CBN

Twierdzenie 2 *Dla kazdego programu* e *oraz literalu* $\ulcorner m \urcorner$*,*

1. $\mathsf{eval_v}(e) = \ulcorner m \urcorner$ *wtw* $\mathsf{eval_v}(\overline{e}\,(\lambda x.x)) = \ulcorner m \urcorner$
2. $\mathsf{eval_n}(e) = \ulcorner m \urcorner$ *wtw* $\mathsf{eval_n}(\overline{e}\,(\lambda x.x)) = \ulcorner m \urcorner$

Twierdzenie 3 *Dla kazdego programu* e *oraz literalu* $\ulcorner m \urcorner$*,*

1. $\mathsf{eval_v}(\overline{e}\,(\lambda x.x)) = \ulcorner m \urcorner$ *wtw* $\mathsf{eval_n}(\overline{e}\,(\lambda x.x)) = \ulcorner m \urcorner$
2. $\mathsf{eval_v}(\overline{\overline{e}}\,(\lambda x.x)) = \ulcorner m \urcorner$ *wtw* $\mathsf{eval_n}(\overline{\overline{e}}\,(\lambda x.x)) = \ulcorner m \urcorner$

Modelowy jezyk z operatorami ISWIM$_{\mathcal{C}}$

$$e ::= x \mid \lambda x.e \mid e_0\, e_1 \mid \ulcorner m \urcorner \mid succ\ e \mid \mathcal{K}x.e \mid \mathcal{A}e$$

- $\mathcal{K}$ – call with current continuation
- $\mathcal{A}$ – abort

Ewaluator w CPS dla ISWIM$_C$ (1)

```
datatype exp = ...
             | ABORT of exp
             | CALLCC of ide * exp

datatype value = INT of int
               | FN of value * cont -> value
withtype cont = value -> value
```

Ewaluator w CPS dla ISWIM$_C$ (2)

```
fun eval (VAR x, env, k)
      = k (Env.lookup (env, x))
  | eval (LAM (x,e), env, k)
      = k (FN (fn (v,k) =>
                eval (e, Env.extend (env, x, v), k)))
  | eval (APP (e0, e1), env, k)
      = eval (e0, env,
          fn v0 => eval (e1, env,
                      fn v1 =>
                        case v0 of
                          FN f => f (v1, k)
                        | _ => raise TYPE_ERROR))
```

Ewaluator w CPS dla ISWIM$_C$ (3)

```
| eval (LIT n, env, k) = k (INT n)
| eval (SUCC e, env, k)
  = eval (e, env,
          fn v => case v of
                    INT n => k (INT (n+1))
                  | _ => raise TYPE_ERROR)
| eval (ABORT e, env, k)
  = eval (e, env, fn v => v)
| eval (CALLCC (x, e), env, k)
  = eval (e, Env.extend (env, x,
                         FN fn (v,k') => k v), k)
fun evaluate e = eval (e, Env.mt, fn v => v)
```

Defunkcjonalizacja funkcji (1)

```
datatype value = INT of int
               | FN of ide * exp * value Env.env
               | CONT of cont

  | eval (APP (e0, e1), env, k)
    = eval (e0, env, fn v0 =>
       eval (e1, env, fn v1 =>
        case v0 of
          FN (x, e, env) =>
              eval (e, Env.extend (env, x, v1), k)
        | CONT k' => k' v1
        | _ => raise TYPE_ERROR))
```

Defunkcjonalizacja funkcji (2)

```
| eval (CALLCC (x, e), env, k)
  = eval (e, Env.extend (env, x, CONT k), k)
```

Defunkcjonalizacja kontynuacji (1)

```
datatype value = INT of int
               | FN of ide * exp * value Env.env
               | CONT of cont
and cont = STOP
         | ARG of exp * value Env.env * cont
         | FUN of value * cont
         | SUCC of cont
```

Defunkcjonalizacja kontynuacji (2)

```
| eval (ABORT e, env, k)
  = eval (e, env, STOP)
| eval (CALLCC (x, e), env, k)
  = eval (e, Env.extend (env, x, CONT k), k)

| apply (FUN (CONT k', k), v1)
  = apply (k', v1)
```

CEK dla ISWIM_c

- Rozszerzona kategoria wartosci

$$v ::= \ldots \mid E$$

- Dodatkowe przejscia

$\langle \mathcal{A}e,\ \rho,\ E \rangle_{eval}$	$\Rightarrow$	$\langle e,\ \rho,\ \mathsf{STOP} \rangle_{eval}$
$\langle \mathcal{K}x.e,\ \rho,\ E \rangle_{eval}$	$\Rightarrow$	$\langle e,\ \rho\{x \mapsto E\},\ \mathsf{STOP} \rangle_{eval}$
$\langle \mathsf{FUN}\,(E',\ E),\ v \rangle_{cont}$	$\Rightarrow$	$\langle E',\ v \rangle_{cont}$

Semantyka redukcyjna dla ISWIM$_c$ (1)

- Redukcja wrazliwa na kontekst – kontrakcja redeksu bierze pod uwage zarowno term jak i kontekst
- Sa dwie (rownowazne) mozliwosci reprezentowania "przechwyconego" kontekstu E:
  1. jako funkcje $\lambda y.\mathcal{A}(E[y])$
  2. jako nowy syntaktyczny rodzaj wartosci (ekstrakcja SR z MA prowadzi wlasnie tu)
- Skoncentrujmy sie na pierwszym wariancie

Semantyka redukcyjna dla ISWIM_c (2)

- Wyrazenia i wartosci

$$\begin{array}{rcl} e & ::= & x \mid v \mid e_0\, e_1 \mid succ\ e \mid \mathcal{A}e \mid \mathcal{K}x.e \\ v & ::= & \lambda x.e \mid \ulcorner m \urcorner \end{array}$$

- Potencjalne i faktyczne redeksy

$$p ::= v_0\, v_1 \mid succ\ v \mid \mathcal{A}e \mid \mathcal{K}x.e$$

$$r ::= (\lambda x.e)\, v \mid succ\ \ulcorner m \urcorner \mid \mathcal{A}e \mid \mathcal{K}x.e$$

- Konteksty ewaluacyjne

$$E ::= [\,] \mid E\,[[\,]\,e] \mid E\,[succ\,[\,]] \mid E\,[v\,[\,]]$$

Semantyka redukcyjna dla $\mathrm{ISWIM}_{\mathcal{C}}$ (3)

- Reguły redukcji

$$
\begin{array}{lrcl}
(\beta_\lambda) & E\,[(\lambda x.e)\,v] & \rightarrow & E\,[e\{v/x\}] \\
(succ) & E\,[succ\ \ulcorner m \urcorner] & \rightarrow & E\,[\ulcorner m+1 \urcorner] \\
(\mathcal{A}) & E\,[\mathcal{A}e] & \rightarrow & e \\
(\mathcal{K}_\lambda) & E\,[\mathcal{K}x.e] & \rightarrow & E\,[e\{(\lambda y.\mathcal{A}(E[y]))/x\}]
\end{array}
$$

Semantyka redukcyjna dla ISWIM$_c$ (4)

Wariant drugi – konteksty staja sie czescia skladni:

- Zmiany w skladni i gramatyce kontekstow:

$$\begin{array}{rcl} v & ::= & \ldots \mid E \\ p & ::= & \ldots \mid E\,v \\ r & ::= & \ldots \mid E\,v \end{array}$$

- Regula $(\mathcal{K}_\lambda)$ jest zastapiona przez dwie

$$\begin{array}{lrcl} (\mathcal{K}'_\lambda) & E\,[\mathcal{K}x.e] & \rightarrow & E\,[e\{E/x\}] \\ (\beta_E) & E\,[E'\,v] & \rightarrow & E'\,[v] \end{array}$$

Funkcje ewaluacyjne generowane przez obie semantyki redukcyjne, jak i przez maszyne abstrakcyjne sa rownowazne.

CPS transformacja operatorow kontroli

$$\overline{\mathcal{A}e} = \lambda k.\overline{e}\,(\lambda v.\, v)$$

$$\overline{\mathcal{K}x.e} = \lambda k.\overline{e}\{(\lambda v k'.\, k\, v)/x\}\, k$$

Mozemy zdefiniowac semantyke jezyka ISWIM$_c$ uzywajac funkcji ewaluacyjnej dla ISWIM (jest ona rownowazna funkcjom generowanym przez SR i MA):

Definicja 3

$$\mathsf{eval}_{\mathsf{CPS}}(e) \stackrel{\mathrm{def}}{=} \mathsf{eval}_{\mathsf{v}}(\overline{e}\,(\lambda x.x)).$$

Operator $\mathcal{C}$

- Operator $\mathcal{C}$

$$\overline{\mathcal{C}x.e} = \lambda k.\bar{e}\{(\lambda v k'.\, k\, v)/x\}\,(\lambda x.\, x)$$

- $\mathcal{C}$ jest silniejszy niz $\mathcal{K}$ i $\mathcal{A}$

$$\mathcal{K}x.e = \mathcal{C}x.x\, e$$

$$\mathcal{A}e = \mathcal{C}d.e, \quad \text{gdzie } d \notin FV\,(e)$$

$$\mathcal{C}x.e = \mathcal{K}x.\mathcal{A}e$$

λ-rachunek i logika minimalna

- Termy (dowody): $e ::= x \mid \lambda x.e \mid e_0\, e_1$
- Typy (formuły): $\phi, \psi ::= \bot \mid A \mid \phi \to \psi$
- Konteksty typowania: $\Gamma ::= \bullet \mid \Gamma, x : \phi$
- Reguły wnioskowania:

$$\frac{}{\Gamma, x : \phi \vdash x : \phi}(\mathsf{AX}) \qquad \frac{\Gamma, x : \phi \vdash e : \psi}{\Gamma \vdash \lambda x.e : \phi \to \psi}(\to_i)$$

$$\frac{\Gamma \vdash e_0 : \phi \to \psi \quad \Gamma \vdash e_1 : \phi}{\Gamma \vdash e_0\, e_1 : \psi}(\to_e)$$

- Definiujemy $\neg\phi \;=\; \phi \to \bot$

Kontynuacje i logika klasyczna (1)

- Logika intuicjonistyczna = Logika minimalna + Ex Falso Quodlibet

$$\frac{\Gamma \vdash e : \bot}{\Gamma \vdash \mathcal{A}e : \phi}(\text{Ex Falso Quodlibet})$$

- Minimalna logika klasyczna = Logika minimalna + Prawo Peirce'a

$$\frac{\Gamma, x : \phi \to \psi \vdash e : \phi}{\Gamma \vdash \mathcal{K}x.e : \phi}(\text{Prawo Peirce'a})$$

Kontynuacje i logika klasyczna (2)

- Logika klasyczna =
  Logika intuicjonistyczna + Prawo Peirce'a =
  Logika minimalna + Eliminacja podwojnego zaprzeczenia (DNE)

$$\frac{\Gamma, x : \neg\phi \vdash e : \bot}{\Gamma \vdash \mathcal{C}x.e : \phi}(\text{DNE})$$

Kontynuacje i logika klasyczna (3)

- Definiujemy transformacje na typach:

$$\overline{\bot} = \bot, \quad \overline{A} = A, \quad \overline{\phi \to \psi} = \overline{\phi} \to \neg\neg\overline{\psi}$$

$$\overline{\bullet} = \bullet, \ \overline{\Gamma, x : \phi} = \overline{\Gamma}, x : \overline{\phi}$$

- **Twierdzenie 4** *Jezeli* $\Gamma \vdash e : \phi$ *w logice klasycznej, to* $\overline{\Gamma} \vdash \overline{e} : \neg\neg\overline{\phi}$ *w logice minimalnej.*

Literatura

- Definitional Interpreters for Higher-Order Programming Languages, J.Reynolds
- Programming Languages and Lambda Calculi, M.Felleisen, M.Flatt
- A Functional Correspondence between Evaluators and Abstract Machines, M.Ager et al.
- Refocusing in Reduction Semantics, O.Danvy, L.Nielsen
- Call-by-name, Call-by-value and the λ-calculus, G.Plotkin
- Representing Control: A Study of the CPS Transformation, O.Danvy, A.Filinski
- Formulae as Types Notion of Control, T.Griffin