Programmer som Data - Assignment 10

Bastjan Rosgaard Sejberg, Søren Kastrup, Weihao Chen Nyholm-Andersen

November 2023

11.1

(I)

```
let rec lenc xs f =
    match xs with
    | [] -> f 0
    | x::xr -> lenc xr (fun v -> f (1+v));;
```

```
[soer4769@soerthinkpad Fun]$ fsharpi cps.fs

Microsoft (R) F# Interactive version 11.0.0.0 for F# 5.0
Copyright (c) Microsoft Corporation. All Rights Reserved.

For help type #help;;

[Loading /home/soer4769/Desktop/Opgaver/5_semester/programmer_som_data/Fun/cps.fs]
namespace FSI_0002
  val len : xs:'a list -> int
  val lenc : xs:'a list -> f:(int -> 'b) -> 'b
  val leni : xs:'a list -> acc:int -> int
  val rev : xs:'a list -> 'a list
  val revc : xs:'a list -> f:('a list -> 'b) -> 'b
  val revi : xs:'a list -> acc:'a list -> 'a list
  val prod : xs:int list -> int
  val prodc : xs:int list -> f:(int -> 'a) -> 'a
  val prodz : xs:int list -> f:(int -> 'a) -> 'a
  val prodi : xs:int list -> acc:int -> int

> open cps;;
> len [2; 5; 7];;
val it : int = 3

> lenc [2; 5; 7] id;;
val it : int = 3

> lenc [2; 5; 7] (printf "The answer is '% d' \n");;
The answer is ' 3'
val it : unit = ()
```

Refer to the file listmachine.c in the folder ***Exercise_11.1-11.4****.*

(II)

```
> lenc [2; 5; 7] (fun v -> 2*v);;
val it : int = 6
```

When calling the function *lenc* with this kind of lambda method instead, it just doubles the result of the length of the list.

(III)

```
let rec leni xs acc =
    match xs with
    | [] -> acc
    | x::xr -> leni xr (acc+1);;
```

```
> leni [2; 5; 7] 0;;
val it : int = 3
```

The relation between *lenc* and *leni* is that both of them go through each element of the list *xs* to add a one to the next call to itself, however from this part onward it is done differently between the two. CPS includes a continuation argument, which call the continuation instead of returning to the caller. Tail-recursive may as in this case include an accumulator argument, which the last function it calls is itself with an updated accumulator.

11.2

(I)

```
let rec revc xs f =
    match xs with
    | [] -> f []
    | x::xr -> revc xr (fun v -> f (v @ [x]));;
```

```
[soer4769@soerthinkpad Fun]$ fsharpi cps.fs

Microsoft (R) F# Interactive version 11.0.0.0 for F# 5.0
Copyright (c) Microsoft Corporation. All Rights Reserved.

For help type #help;;

[Loading /home/soer4769/Desktop/Opgaver/5_semester/programmer_som_data/Fun/cps.fs]
namespace FSI_0002
  val len : xs:'a list -> int
  val lenc : xs:'a list -> f:(int -> 'b) -> 'b
  val leni : xs:'a list -> acc:int -> int
  val rev : xs:'a list -> 'a list
  val revc : xs:'a list -> f:('a list -> 'b) -> 'b
  val revi : xs:'a list -> acc:'a list -> 'a list
  val prod : xs:int list -> int
  val prodc : xs:int list -> f:(int -> 'a) -> 'a
  val prodz : xs:int list -> f:(int -> 'a) -> 'a
  val prodi : xs:int list -> acc:int -> int

> open cps;;
> rev [2; 5; 7];;
val it : int list = [7; 5; 2]

> revc [2; 5; 7] id;;
val it : int list = [7; 5; 2]
```

Refer to the file listmachine.c in the folder ***Exercise_11.1-11.4****.*

(II)

```
> revc [2; 5; 7] (fun v -> v @ v);;
val it : int list = [7; 5; 2; 7; 5; 2]
```

When calling the function *revc* with this kind of lambda method instead, it simply reverses the list and afterwards joins together with a copy of itself afterwards.

(III)

```
let rec revi xs acc =
    match xs with
    | [] -> acc
    | x::xr -> revi xr (x :: acc);;
```

```
> revi [2; 5; 7] [];;
val it : int list = [7; 5; 2]
```

11.3

```
let rec prodc xs f =
    match xs with
    | [] -> f 1
    | x::xr -> prodc xr (fun v -> f (x*v));;
```

```
[soer4769@soerthinkpad Fun]$ fsharpi cps.fs

Microsoft (R) F# Interactive version 11.0.0.0 for F# 5.0
Copyright (c) Microsoft Corporation. All Rights Reserved.

For help type #help;;

[Loading /home/soer4769/Desktop/Opgaver/5_semester/programmer_som_data/Fun/cps.fs]
namespace FSI_0002
  val len : xs:'a list -> int
  val lenc : xs:'a list -> f:(int -> 'b) -> 'b
  val leni : xs:'a list -> acc:int -> int
  val rev : xs:'a list -> 'a list
  val revc : xs:'a list -> f:('a list -> 'b) -> 'b
  val revi : xs:'a list -> acc:'a list -> 'a list
  val prod : xs:int list -> int
  val prodc : xs:int list -> f:(int -> 'a) -> 'a
  val prodz : xs:int list -> f:(int -> 'a) -> 'a
  val prodi : xs:int list -> acc:int -> int

> open cps;;
> prod [2; 5; 7];;
val it : int = 70

> prodc [2; 5; 7] id;;
val it : int = 70
```

11.4

```
let prodz xs f =
    let rec inner xs2 f2 =
        match xs2 with
        | [] -> f 1
        | x::xr ->
            match x with
            | 0 -> f 0
            | _ -> prodc xr (fun v -> f2 (x*v))
    inner xs f;;

let rec prodi xs acc =
    match xs with
    | [] -> acc
    | x::xr ->
        match x with
        | 0 -> 0
        | _ -> prodi xr (acc*x);;
```

```
[Loading /home/soer4769/Desktop/Opgaver/5_semester/programmer_som_data/Fun/cps.fs]
namespace FSI_0002
  val len : xs:'a list -> int
  val lenc : xs:'a list -> f:(int -> 'b) -> 'b
  val leni : xs:'a list -> acc:int -> int
  val rev : xs:'a list -> 'a list
  val revc : xs:'a list -> f:('a list -> 'b) -> 'b
  val revi : xs:'a list -> acc:'a list -> 'a list
  val prod : xs:int list -> int
  val prodc : xs:int list -> f:(int -> 'a) -> 'a
  val prodz : xs:int list -> f:(int -> 'a) -> 'a
  val prodi : xs:int list -> acc:int -> int

> open cps;;
> prodc [2; 5; 7] id;;
val it : int = 70

> prodc [2; 5; 7] (printf "The answer is '% d' \n");;
The answer is ' 70'
val it : unit = ()

> prodc [2; 0; 5; 7] id;;
val it : int = 0

> prodc [2; 0; 5; 7] (printf "The answer is '% d' \n");;
The answer is ' 0'
val it : unit = ()

> prodi [2; 5; 7] 1;;
val it : int = 70

> prodi [2; 0; 5; 7] 1;;
val it : int = 0
```

11.8

(I)

```
[soer4769@soerthinkpad Fun]$ fsharpi Icon.fs

Microsoft (R) F# Interactive version 11.0.0.0 for F# 5.0
Copyright (c) Microsoft Corporation. All Rights Reserved.

For help type #help;;

[Loading /home/soer4769/Desktop/Opgaver/5_semester/programmer_som_data/Fun/Icon.fs]
namespace FSI_0002
  type expr =
    | CstI of int
    | CstS of string
    | FromTo of int * int
    | Write of expr
    | If of expr * expr * expr
    | Prim of string * expr * expr
    | And of expr * expr
    | Or of expr * expr
    | Seq of expr * expr
    | Every of expr
    | Fail
  type value =
    | Int of int
    | Str of string
  type econt = unit -> value
  type cont = value -> econt -> value
  val write : v:value -> unit
  val eval : e:expr -> cont:cont -> econt:econt -> value
  val run : e:expr -> value
  val ex1 : expr
  val ex2 : expr
  val ex3and : expr
  val ex3or : expr
  val ex3seq : expr
  val ex4 : expr
  val ex5 : expr
  val ex6 : expr
  val ex7 : expr
  val ex8 : expr

> open Icon;;
> run (Every(Write(Prim("+", CstI 1, Prim("*", CstI 2, FromTo(1, 4))))));;
3 5 7 9 val it : value = Int 0

> run (Every(Write(Prim("+", Prim("*", CstI 10, FromTo(2,4)), FromTo(1,2)))));;
21 22 31 32 41 42 val it : value = Int 0
```

(II)

```
[soer4769@soerthinkpad Fun]$ fsharpi Icon.fs

Microsoft (R) F# Interactive version 11.0.0.0 for F# 5.0
Copyright (c) Microsoft Corporation. All Rights Reserved.

For help type #help;;

[Loading /home/soer4769/Desktop/Opgaver/5_semester/programmer_som_data/Fun/Icon.fs]
namespace FSI_0002
  type expr =
    | CstI of int
    | CstS of string
    | FromTo of int * int
    | Write of expr
    | If of expr * expr * expr
    | Prim of string * expr * expr
    | And of expr * expr
    | Or of expr * expr
    | Seq of expr * expr
    | Every of expr
    | Fail
  type value =
    | Int of int
    | Str of string
  type econt = unit -> value
  type cont = value -> econt -> value
  val write : v:value -> unit
  val eval : e:expr -> cont:cont -> econt:econt -> value
  val run : e:expr -> value
  val ex1 : expr
  val ex2 : expr
  val ex3and : expr
  val ex3or : expr
  val ex3seq : expr
  val ex4 : expr
  val ex5 : expr
  val ex6 : expr
  val ex7 : expr
  val ex8 : expr

> open Icon;;
> run (Every(Write(Prim("<", CstI 50, Prim("*", CstI 7, FromTo(1,8))))));;
56 val it : value = Int 0
```

(III)

```
type expr =
    | ...
    | Prim1 of string * expr

let rec eval (e : expr) (cont : cont) (econt : econt) =
    march e with
    | ...
    | Prim1(ope, e1) ->
      eval e1 (fun v1 -> fun econt1 ->
        match (ope, v1) with
        | ("square", Int i1) ->
            cont (Int(i1*i1)) econt1
        | ("even", Int i1) ->
            if i1 % 2 = 0 then
              cont (Int i1) econt1
            else
              econt1 ()
        | _ -> Str "unknown prim1") econt
```

```
[soer4769@soerthinkpad Fun]$ fsharpi Icon.fs

Microsoft (R) F# Interactive version 11.0.0.0 for F# 5.0
Copyright (c) Microsoft Corporation. All Rights Reserved.

For help type #help;;

[Loading /home/soer4769/Desktop/Opgaver/5_semester/programmer_som_data/Fun/Icon.fs]
namespace FSI_0002
  type expr =
    | CstI of int
    | CstS of string
    | FromTo of int * int
    | Write of expr
    | If of expr * expr * expr
    | Prim of string * expr * expr
    | Prim1 of string * expr
    | And of expr * expr
    | Or of expr * expr
    | Seq of expr * expr
    | Every of expr
    | Fail
  type value =
    | Int of int
    | Str of string
  type econt = unit -> value
  type cont = value -> econt -> value
  val write : v:value -> unit
  val eval : e:expr -> cont:cont -> econt:econt -> value
  val run : e:expr -> value
  val ex1 : expr
  val ex2 : expr
  val ex3and : expr
  val ex3or : expr
  val ex3seq : expr
  val ex4 : expr
  val ex5 : expr
  val ex6 : expr
  val ex7 : expr
  val ex8 : expr

> open Icon;;
> run (Every(Write(Prim1("square", FromTo(3,6)))));;
9 16 25 36 val it : value = Int 0

> run (Every(Write(Prim1("even", FromTo(1,7)))));;
2 4 6 val it : value = Int 0
```

Refer to the file Icon.fs in the folder ***Exercise_11.8****.*

(IV)

The group was unsure how to finish this part of the exercise.