Januar 07-08 2016 BPRD Eksamen

Anders Wind Steffensen: awis@itu.dk

Jeg erklærer hermed at jeg selv har lavet hele denne eksamensbesvarelse uden hjælp fra andre.

Opgave 1

1.1)

- kls
- khs
- kvs
- klhvs
- kvhls
- klls
- khhlls
- khwvhhs

Udtrykket matcher sprog som starter med k. Efter k skal et vilkårligt antal på mindst 1, af enten l, v eller h optræde. Sproget skal afsluttes med et s.

1.2)

Jeg har baseret løsningen ud fra metoden beskrevet i Basics of Compiler Design. Muligvis kan man ikke lave denne ræcise opbygning for sammenkædede | konstruktioner, og de kunne være blevet splitet ud således at a|b hvor b = c|d. Jeg har taget udgangspunkt i at + svarer overens til: aa*

Den udarbejdede NFA kan ses på figur x.

1.3)

Jeg har benyttet mig af algoritemn beskrevet i Basics of Compiler Design. Algoritmens resultat kan ses i tabellen herunder

State	k	l	v	h	s	NFA states
S1	S2	Ø	Ø	Ø	Ø	1
S2	Ø	S3	S3	S3	Ø	2, 3, 4, 5
S3	Ø	S4	S4	S4	S5	6, 7, 8, 9, 10, 11, 13
S4	Ø	S4	S4	S4	S5	7, 8, 9, 10, 11, 12, 13
S5	Ø	Ø	Ø	Ø	Ø	14

På figur x ses den udarbejdede DFA.

1.4)

Jeg har udarbejdet følgende regex: (l|hl|vl)+ Den matcher på alle de strenge der er givet i opgave beskrivelsen, samt undgår at matche på de: lv, lh, lw, lvh, lhv og lhh.

Opgave 2

2.1)

Ændringer i Absyn:

```
type expr =
  | CstI of int
  | CstB of bool
  | Var of string
  | Let of string * expr * expr
  | Prim of string * expr * expr
  | If of expr * expr * expr
  | Letfun of string * string * expr * expr    (* (f, x, fBody, letBody) *)
  | Call of expr * expr
  | Ref of expr
  | Deref of expr
  | UpdRef of expr * expr
```

2.2)

Ændringer i HigherFun type value = | Int of int | Closure of string * string * expr * value env (* (f, x, fBody, fDeclEnv) *) | Refvalue of value ref

2.3)

Ændringer i HigherFun

```
let rec eval (e : expr) (env : value env) : value =
    match e with
    ...
    | Ref e             -> RefVal (ref (eval e env))
    | Deref e           -> let value = eval e env
                           match value with
```

```
                        | RefVal refv -> !refv
                        | _ -> failwith "Deref did not get a refvalue"
    | UpdRef (e1,e2)    -> let value1 = eval e1 env
                           match value1 with
                           | RefVal refv -> let value2 = eval e2 env
                                            refv := value2
                                            value2
                           | _ -> failwith "Updref did not get a refvalue"
```

Der er altså tilføjet 3 nye matches. Disse matches håndterer hver af de nye expr typer. Ref e: returnerer en RefVal med resultatet af den evaluerede e i det nuværende environment refereret til med en F# reference. Deref e: evaluerer e i det nuværende environment og matcher det med en RefVal. I dette tilfælde benyttes F# deref notationen !ref til at returnere værdien i RefVal uden at det er en reference. UpdRed (e1, e2): Først tjekkes det at e1 er en RefVal type ligesom i Deref. Herefter evalueres e2 i det nye environment, og værdien i e1 referencen updateres med F# update ref notation :=.

2.4)

```
> run (Letfun ("f", "x", Deref(Var "x"), Call(Var "f", Ref(CstI 1))));;
val it : HigherFun.value = Int 1

> run (Deref(Ref(UpdRef(Ref(UpdRef(Ref(CstI 1),CstI 2)),CstI 3))));;
val it : HigherFun.value = Int 3

> run (Let("x", Ref(CstI 1), UpdRef(Var "x", CstI 2)));;
val it : HigherFun.value = Int 2

> run (Prim("=", CstI 2, Deref(Ref(Prim("+", CstI 1, CstI 1)))));;
val it : HigherFun.value = Int 1
```

2.5)

Ændringer i FunLex.fsl

```
let keyword s =
    match s with
    ...
    | "ref"   -> REF
    | _       -> NAME s

rule Token = parse
    ...
    | '!'             { EXMARK }
    | ":="            { UPDOP }
    | eof             { EOF }
    | _               { failwith "Lexer error: illegal symbol" }
```

Ændringer i FunLex.fsy

```
%token UPDOP EXMARK REF

%left ELSE              /* lowest precedence  */
%left EQ NE UPDOP
%right REF EXMARK
%nonassoc GT LT GE LE
%left PLUS MINUS
%left TIMES DIV MOD
%nonassoc NOT           /* highest precedence  */

Expr:
...
    | Ref UPDOP Expr                    { UpdRef($1, $3)         }
;

AtExpr:
      Const                             { $1                     }
    | NAME                              { Var $1                 }
    | REF Expr                          { Ref $2                 }
    | EXMARK Expr                       { Deref $2               }
    | LET NAME EQ Expr IN Expr END      { Let($2, $4, $6)        }
    | LET NAME NAME EQ Expr IN Expr END { Letfun($2, $3, $5, $7) }
    | LPAR Expr RPAR                    { $2                     }
;
```

2.6)

Eksempler fra sektion 2.3

```
> fromString "let x = ref 1 in if !x=1 then x:= 2 else 42 end";;
val it : Absyn.expr = Let ("x",Ref (CstI 1), If (Prim ("=",Deref (Var "x"),CstI 1),UpdRef (Var "x",CstI 2),CstI 42)
> run it;;
val it : HigherFun.value = Int 2

> fromString "let x = ref 2 in (x:=3) + !x end";;
```

```
val it : Absyn.expr = Let ("x",Ref (CstI 2),Prim ("+",UpdRef (Var "x",CstI 3),Deref (Var "x")))
> run it;;
val it : HigherFun.value = Int 6
```

Egne eksempler fra sektion 2.4

```
> fromString "let f x = !x in f ref 1 end";;
val it : Absyn.expr = Letfun ("f","x",Deref (Var "x"),Call (Var "f",Ref (CstI 1)))
> run it;;
val it : HigherFun.value = Int 1

> fromString "!(ref ((ref ((ref 1):=2)):=3))";;
val it : Absyn.expr = Deref (Ref (UpdRef (Ref (UpdRef (Ref (CstI 1),CstI 2)),CstI 3)))
> run it;;
val it : HigherFun.value = Int 3

> fromString "let x = ref 1 in x:=2 end";;
val it : Absyn.expr = Let ("x",Ref (CstI 1),UpdRef (Var "x",CstI 2))
> run it;;
val it : HigherFun.value = Int 2

> fromString "2=!(ref (1+1))";;
val it : Absyn.expr = Prim ("=",CstI 2,Deref (Ref (Prim ("+",CstI 1,CstI 1))))
> run it;;
val it : HigherFun.value = Int 1
```

2.7)

Det udarbejdede typetræ kan ses på figur x

- Typen c = int
- Typen b = c ref
- Typen a = int

Altså ses det at den endelige type er: int. Der er også overenstemmelse mellem de værdier der udledes af regel 1 og de værdier x bliver slået op til at være.

Opgave 3

Opgave 4
