Programmer som Data - Assignment 11

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

November 2023

12.1

Code (changes)[1]**:**

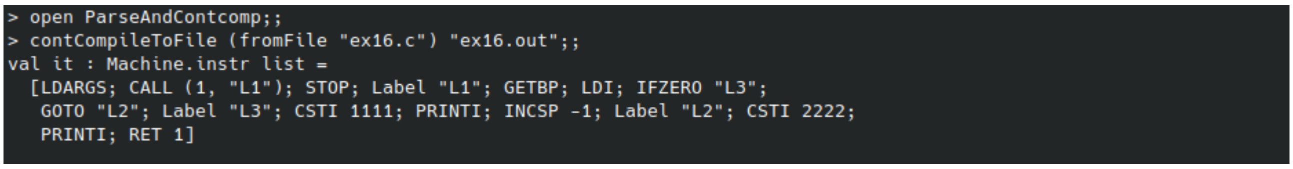
```
let rec addIFZERO lab3 C =
  match C with
  | GOTO lab2 :: Label lab1 :: C1 -> IFNZRO lab2 :: Label lab3 :: C1
  | _ -> IFZERO lab3 :: C

let rec addIFNZRO lab3 C =
  match C with
  | GOTO lab2 :: Label lab1 :: C1 -> IFZERO lab2 :: Label lab3 :: C1
  | _ -> IFNZRO lab3 :: C
...
```

Listing 1: Contcomp.fs

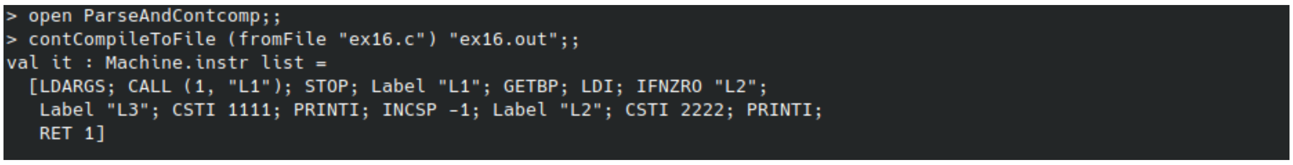
Refer to the files in the folder ***Exercise-12.1****.*

Result (before):



```
> open ParseAndContcomp;;
> contCompileToFile (fromFile "ex16.c") "ex16.out";;
val it : Machine.instr list =
  [LDARGS; CALL (1, "L1"); STOP; Label "L1"; GETBP; LDI; IFZERO "L3";
   GOTO "L2"; Label "L3"; CSTI 1111; PRINTI; INCSP -1; Label "L2"; CSTI 2222;
   PRINTI; RET 1]
```

Result (after):



```
> open ParseAndContcomp;;
> contCompileToFile (fromFile "ex16.c") "ex16.out";;
val it : Machine.instr list =
  [LDARGS; CALL (1, "L1"); STOP; Label "L1"; GETBP; LDI; IFNZRO "L2";
   Label "L3"; CSTI 1111; PRINTI; INCSP -1; Label "L2"; CSTI 2222; PRINTI;
   RET 1]
```

[1]Replecements of *IFZERO labelse::code* to *addIFZERO labelse code* and *IFNZRO labelse::code* to *addINZRO labelse code* throughout the file not shown here.

12.2

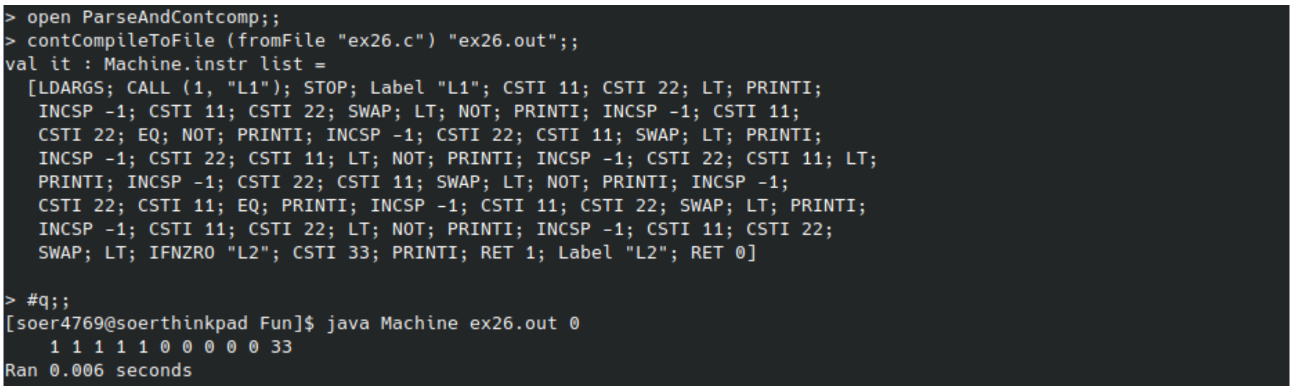
Code (changes):

```
let rec addCST i C =
    match (i, C) with
    | (_, CSTI j :: SWAP :: LT :: NOT :: C1) -> addCST (System.Convert.ToInt32(i<=j)) C1
    | (_, CSTI j :: SWAP :: LT :: C1)        -> addCST (System.Convert.ToInt32(i>j)) C1
    | (_, CSTI j :: EQ :: NOT :: C1)         -> addCST (System.Convert.ToInt32(i<>j)) C1
    | (_, CSTI j :: EQ :: C1)                -> addCST (System.Convert.ToInt32(not (i<>j))) C1
    | (_, CSTI j :: LT :: NOT :: C1)         -> addCST (System.Convert.ToInt32(i>=j)) C1
    | (_, CSTI j :: LT :: C1)                -> addCST (System.Convert.ToInt32(i<j)) C1
    ...
```

Listing 2: Contcomp.fs

Refer to the files in the folder ***Exercise-12.2****.*

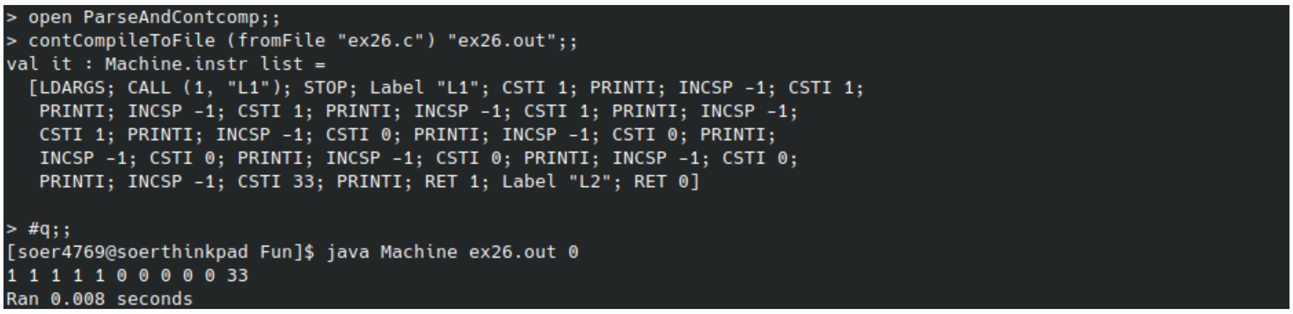
Result (before):



```
> open ParseAndContcomp;;
> contCompileToFile (fromFile "ex26.c") "ex26.out";;
val it : Machine.instr list =
  [LDARGS; CALL (1, "L1"); STOP; Label "L1"; CSTI 11; CSTI 22; LT; PRINTI;
   INCSP -1; CSTI 11; CSTI 22; SWAP; LT; NOT; PRINTI; INCSP -1; CSTI 11;
   CSTI 22; EQ; NOT; PRINTI; INCSP -1; CSTI 22; CSTI 11; SWAP; LT; PRINTI;
   INCSP -1; CSTI 22; CSTI 11; LT; NOT; PRINTI; INCSP -1; CSTI 22; CSTI 11; LT;
   PRINTI; INCSP -1; CSTI 22; CSTI 11; SWAP; LT; NOT; PRINTI; INCSP -1;
   CSTI 22; CSTI 11; EQ; PRINTI; INCSP -1; CSTI 11; CSTI 22; SWAP; LT; PRINTI;
   INCSP -1; CSTI 11; CSTI 22; LT; NOT; PRINTI; INCSP -1; CSTI 11; CSTI 22;
   SWAP; LT; IFNZRO "L2"; CSTI 33; PRINTI; RET 1; Label "L2"; RET 0]

> #q;;
[soer4769@soerthinkpad Fun]$ java Machine ex26.out 0
    1 1 1 1 1 0 0 0 0 0 33
Ran 0.006 seconds
```

Result (after):



```
> open ParseAndContcomp;;
> contCompileToFile (fromFile "ex26.c") "ex26.out";;
val it : Machine.instr list =
  [LDARGS; CALL (1, "L1"); STOP; Label "L1"; CSTI 1; PRINTI; INCSP -1; CSTI 1;
   PRINTI; INCSP -1; CSTI 1; PRINTI; INCSP -1; CSTI 1; PRINTI; INCSP -1;
   CSTI 1; PRINTI; INCSP -1; CSTI 0; PRINTI; INCSP -1; CSTI 0; PRINTI;
   INCSP -1; CSTI 0; PRINTI; INCSP -1; CSTI 0; PRINTI; INCSP -1; CSTI 0;
   PRINTI; INCSP -1; CSTI 33; PRINTI; RET 1; Label "L2"; RET 0]

> #q;;
[soer4769@soerthinkpad Fun]$ java Machine ex26.out 0
1 1 1 1 1 0 0 0 0 0 33
Ran 0.008 seconds
```

12.3

Code (changes):

```
%token QUESTION COLON
...
%left QUESTION COLON
...
ExprNotAccess:
  ...
  | Expr QUESTION Expr COLON Expr       { Cond($1, $3, $5)    }
;
```

Listing 3: CPar.fsy

```
rule Token = parse
  | '?'             { QUESTION }
  | ':'             { COLON }
  ...
```

Listing 4: CLex.fsl

```
and expr =
  | Cond of expr * expr * expr       (* Ternary Conditional Expr.   *)
  ...
```

Listing 5: Absyn.fs

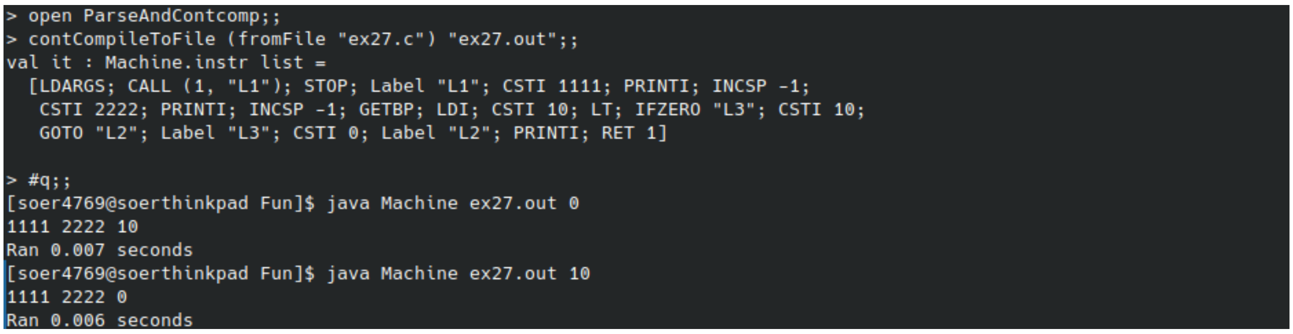
```
and Expr e varEnv funEnv C =
    | Cond(e1, e2, e3) ->
      let le1 = cExpr e1 varEnv funEnv []
      match le1 with
      | CSTI 1 :: _ -> cExpr e2 varEnv funEnv C
      | CSTI 0 :: _ -> cExpr e3 varEnv funEnv C
      | _        -> let (jumpend, C1) = makeJump C
                    let (labelse, C2) = addLabel (cExpr e3 varEnv funEnv C1)

                    cExpr e1 varEnv funEnv (IFZERO labelse
                    :: cExpr e2 varEnv funEnv (addJump jumpend C2))
    ...
```

Listing 6: Contcomp.fs

*Refer to the files in the folder **Exercise-12.3**.*

Result:



```
> open ParseAndContcomp;;
> contCompileToFile (fromFile "ex27.c") "ex27.out";;
val it : Machine.instr list =
  [LDARGS; CALL (1, "L1"); STOP; Label "L1"; CSTI 1111; PRINTI; INCSP -1;
   CSTI 2222; PRINTI; INCSP -1; GETBP; LDI; CSTI 10; LT; IFZERO "L3"; CSTI 10;
   GOTO "L2"; Label "L3"; CSTI 0; Label "L2"; PRINTI; RET 1]

> #q;;
[soer4769@soerthinkpad Fun]$ java Machine ex27.out 0
1111 2222 10
Ran 0.007 seconds
[soer4769@soerthinkpad Fun]$ java Machine ex27.out 10
1111 2222 0
Ran 0.006 seconds
```

12.4

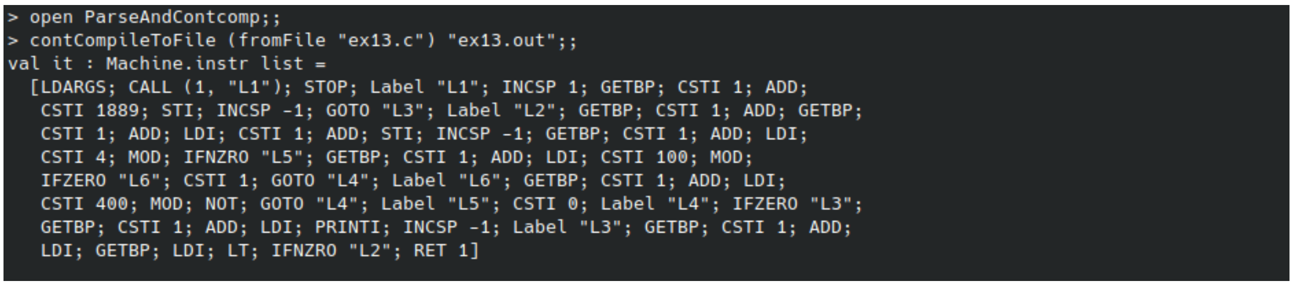
Code (changes):

```
ExprNotAccess:
    ...
    | Expr SEQAND Expr                      { Cond($1, $3, CstI 0)} /* Andalso($1, $3) */
    | Expr SEQOR  Expr                      { Cond($1, CstI 1, $3)} /* Orelse($1, $3)  */
;
```

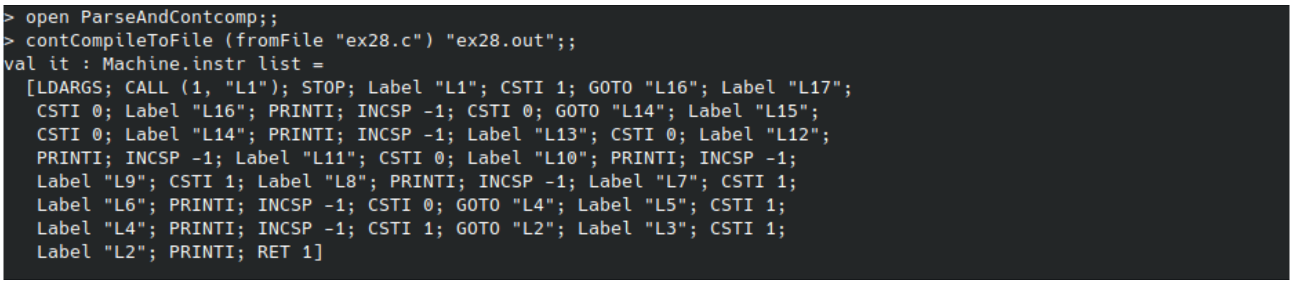
Listing 7: Cpar.fsy

*Refer to the files in the folder **Exercise12.4**.*

Result (before):

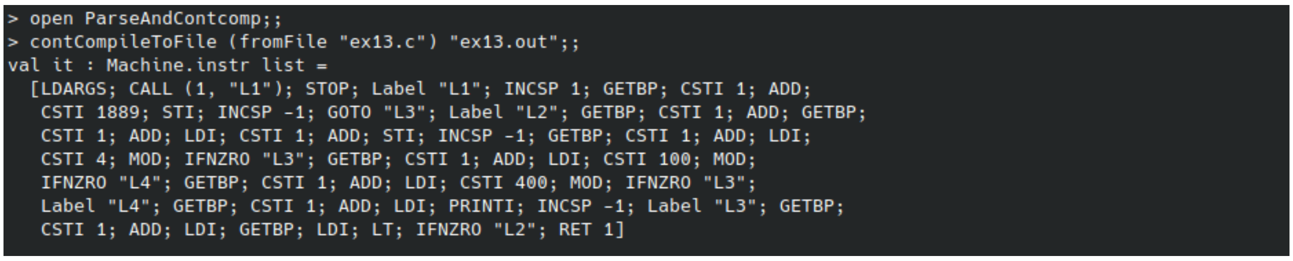


```
> open ParseAndContcomp;;
> contCompileToFile (fromFile "ex13.c") "ex13.out";;
val it : Machine.instr list =
  [LDARGS; CALL (1, "L1"); STOP; Label "L1"; INCSP 1; GETBP; CSTI 1; ADD;
   CSTI 1889; STI; INCSP -1; GOTO "L3"; Label "L2"; GETBP; CSTI 1; ADD; GETBP;
   CSTI 1; ADD; LDI; CSTI 1; ADD; STI; INCSP -1; GETBP; CSTI 1; ADD; LDI;
   CSTI 4; MOD; IFNZRO "L5"; GETBP; CSTI 1; ADD; LDI; CSTI 100; MOD;
   IFZERO "L6"; CSTI 1; GOTO "L4"; Label "L6"; GETBP; CSTI 1; ADD; LDI;
   CSTI 400; MOD; NOT; GOTO "L4"; Label "L5"; CSTI 0; Label "L4"; IFZERO "L3";
   GETBP; CSTI 1; ADD; LDI; PRINTI; INCSP -1; Label "L3"; GETBP; CSTI 1; ADD;
   LDI; GETBP; LDI; LT; IFNZRO "L2"; RET 1]
```

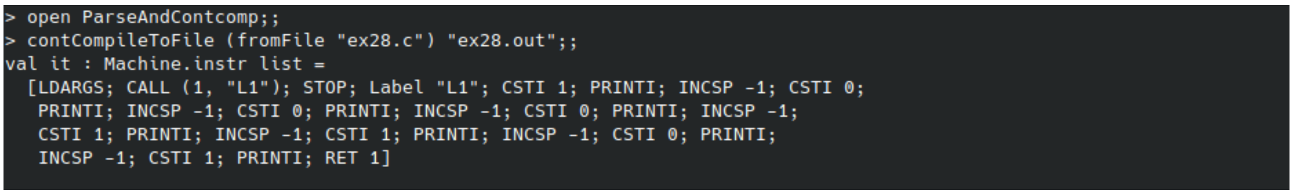


```
> open ParseAndContcomp;;
> contCompileToFile (fromFile "ex28.c") "ex28.out";;
val it : Machine.instr list =
  [LDARGS; CALL (1, "L1"); STOP; Label "L1"; CSTI 1; GOTO "L16"; Label "L17";
   CSTI 0; Label "L16"; PRINTI; INCSP -1; CSTI 0; GOTO "L14"; Label "L15";
   CSTI 0; Label "L14"; PRINTI; INCSP -1; Label "L13"; CSTI 0; Label "L12";
   PRINTI; INCSP -1; Label "L11"; CSTI 0; Label "L10"; PRINTI; INCSP -1;
   Label "L9"; CSTI 1; Label "L8"; PRINTI; INCSP -1; Label "L7"; CSTI 1;
   Label "L6"; PRINTI; INCSP -1; CSTI 0; GOTO "L4"; Label "L5"; CSTI 1;
   Label "L4"; PRINTI; INCSP -1; CSTI 1; GOTO "L2"; Label "L3"; CSTI 1;
   Label "L2"; PRINTI; RET 1]
```

Result (after):



```
> open ParseAndContcomp;;
> contCompileToFile (fromFile "ex13.c") "ex13.out";;
val it : Machine.instr list =
  [LDARGS; CALL (1, "L1"); STOP; Label "L1"; INCSP 1; GETBP; CSTI 1; ADD;
   CSTI 1889; STI; INCSP -1; GOTO "L3"; Label "L2"; GETBP; CSTI 1; ADD; GETBP;
   CSTI 1; ADD; LDI; CSTI 1; ADD; STI; INCSP -1; GETBP; CSTI 1; ADD; LDI;
   CSTI 4; MOD; IFNZRO "L3"; GETBP; CSTI 1; ADD; LDI; CSTI 100; MOD;
   IFNZRO "L4"; GETBP; CSTI 1; ADD; LDI; CSTI 400; MOD; IFNZRO "L3";
   Label "L4"; GETBP; CSTI 1; ADD; LDI; PRINTI; INCSP -1; Label "L3"; GETBP;
   CSTI 1; ADD; LDI; GETBP; LDI; LT; IFNZRO "L2"; RET 1]
```



```
> open ParseAndContcomp;;
> contCompileToFile (fromFile "ex28.c") "ex28.out";;
val it : Machine.instr list =
  [LDARGS; CALL (1, "L1"); STOP; Label "L1"; CSTI 1; PRINTI; INCSP -1; CSTI 0;
   PRINTI; INCSP -1; CSTI 0; PRINTI; INCSP -1; CSTI 0; PRINTI; INCSP -1;
   CSTI 1; PRINTI; INCSP -1; CSTI 1; PRINTI; INCSP -1; CSTI 0; PRINTI;
   INCSP -1; CSTI 1; PRINTI; RET 1]
```

How does the code quality compare to the existing complicated compilation of && and ||?

The changes in the amount of code can be seen in two scenarios, one where the answer is already known at compilation time and one where it is not. When the answer is already known ahead of time, code in *ex28.fs* for *print (true && true);* changes from *CSTI 1; GOTO "L16"; LABEL "L17"; CSTI 0; LABEL "L16"; PRINTL;* to *CSTI 1; PRINTI;* which fully removes any unnecessary code from what was an half-optimized check.

When the answer is not known ahead of time the quality only changes in certain situations, code in *ex13.fs* in the while loop *while (y < n)* for *GETBP; CSTI 1; ADD; LDI; GETBP; LDI, LT; IFNZRO "L2";* is identical. However for the if statement in the middle the individual checks are a bit shorter which is because in the original generated code if *y % 4 == 0* is false it jumps to the label islands *LABEL "L5"; CSTI 0; LABEL "L4"; IFZERO "L3";* and then continues the loop, which is also the case if *y % 100 != 0* is true it continues to *CSTI 1; GOTO "L4";*. In the optimised version this is not present and instead the code for each check is almost identical like so *GETBP; CSTI 1; ADD; LDI; CSTI 100; MOD; IFNZRO "L4";*, where if the first or third check fails they jump to the while loop check label *L3* but if the second is succeed it automatically jumps to the print variable y to screen label *L4*.