ProgAsDataAs4

Mads Ljungberg

September 2021

1 4.1

```
Compiling and loading the micro-ML evaluator and parser (Fun/README.TXT)
-------------------------------------------------------------------------

Archive fun1.zip (lecture 3) contains the files used in points A-C below, and
archive fun2.zip (lecture 4) contains additional files used in points D-F.


A. Loading the micro-ML evaluator, with abstract syntax only

   fsharpi Absyn.fs Fun.fs

   open Absyn;;
   open Fun;;
   let res = run (Prim("+", CstI 5, CstI 7));;
   #q;;


       D:\GitHubRepoes\BPRD\Assignment4\Fun>fsi Absyn.fs Fun.fs

       Microsoft (R) F# Interactive version 12.0.30815.0
       Copyright (c) Microsoft Corporation. All Rights Reserved.

       For help type #help;;

       [Loading D:\GitHubRepoes\BPRD\Assignment4\Fun\Absyn.fs
        Loading D:\GitHubRepoes\BPRD\Assignment4\Fun\Fun.fs]

       namespace FSI_0002
         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
            | Call of expr * expr


namespace FSI_0002
  type 'v env = (string * 'v) list
  val lookup : env:(string * 'a) list -> x:string -> 'a
  type value =
        | Int of int
        | Closure of string * string * Absyn.expr * value env
  val eval : e:Absyn.expr -> env:value env -> int
  val run : e:Absyn.expr -> int
  val ex1 : Absyn.expr
  val ex2 : Absyn.expr
  val ex3 : Absyn.expr
  val rundeep : n:int -> int
  val ex4 : Absyn.expr
  val ex5 : Absyn.expr

> open Absyn;;
> open Fun;;
> let res = run (Prim("+", CstI 5, CstI 7));;

val res : int = 12

> #q;;

D:\GitHubRepoes\BPRD\Assignment4\Fun>
```

B. Generating and compiling the lexer and parser, and loading them:

```
fsyacc --module FunPar FunPar.fsy
fslex --unicode FunLex.fsl
fsharpi -r ~/fsharp/FsLexYacc.Runtime.dll Absyn.fs FunPar.fs FunLex.fs Parse.fs
// fsharpi should for me be fsi

open Parse;;
let e1 = fromString "5+7";;
let e2 = fromString "let y = 7 in y + 2 end";;
let e3 = fromString "let f x = x + 7 in f 2 end";;


D:\GitHubRepoes\BPRD\Assignment4\Fun>bin\fsyacc --module FunPar FunPar.fsy
building tables
computing first function...time: 00:00:00.0871566
building kernels...time: 00:00:00.0576248
```

```
building kernel table...time: 00:00:00.0146807
computing lookahead relations.....................................................
.........time: 00:00:00.0460465
building lookahead table...time: 00:00:00.0153142
building action table...state 21: shift/reduce error on GT
state 21: shift/reduce error on LT
state 21: shift/reduce error on GE
state 21: shift/reduce error on LE
state 22: shift/reduce error on GT
state 22: shift/reduce error on LT
state 22: shift/reduce error on GE
state 22: shift/reduce error on LE
state 23: shift/reduce error on GT
state 23: shift/reduce error on LT
state 23: shift/reduce error on GE
state 23: shift/reduce error on LE
state 24: shift/reduce error on GT
state 24: shift/reduce error on LT
state 24: shift/reduce error on GE
state 24: shift/reduce error on LE
time: 00:00:00.0610508
building goto table...time: 00:00:00.0018371
returning tables.
16 shift/reduce conflicts
58 states
6 nonterminals
29 terminals
26 productions
#rows in action table: 58

D:\GitHubRepoes\BPRD\Assignment4\Fun>bin\fslex --unicode FunLex.fsl
compiling to dfas (can take a while...)
30 states
writing output

D:\GitHubRepoes\BPRD\Assignment4\Fun>fsi -r bin\FsLexYacc.Runtime.dll Absyn.fs
FunPar.fs FunLex.fs Parse.fs

Microsoft (R) F# Interactive version 12.0.30815.0
Copyright (c) Microsoft Corporation. All Rights Reserved.

For help type #help;;

[Loading D:\GitHubRepoes\BPRD\Assignment4\Fun\Absyn.fs
 Loading D:\GitHubRepoes\BPRD\Assignment4\Fun\FunPar.fs
 Loading D:\GitHubRepoes\BPRD\Assignment4\Fun\FunLex.fs
 Loading D:\GitHubRepoes\BPRD\Assignment4\Fun\Parse.fs]
```

```
namespace FSI_0002
  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
        | Call of expr * expr


namespace FSI_0002
  type token =
        | EOF
        | LPAR
        | RPAR
        | EQ
        | NE
        | GT
        | LT
        | GE
        | LE
        | PLUS
        | MINUS
        | TIMES
        | DIV
        | MOD
        | ELSE
        | END
        | FALSE
        | IF
        | IN
        | LET
        | NOT
        | THEN
        | TRUE
        | CSTBOOL of bool
        | NAME of string
        | CSTINT of int
  type tokenId =
        | TOKEN_EOF
        | TOKEN_LPAR
        | TOKEN_RPAR
        | TOKEN_EQ
        | TOKEN_NE
        | TOKEN_GT
        | TOKEN_LT
```

```
        | TOKEN_GE
        | TOKEN_LE
        | TOKEN_PLUS
        | TOKEN_MINUS
        | TOKEN_TIMES
        | TOKEN_DIV
        | TOKEN_MOD
        | TOKEN_ELSE
        | TOKEN_END
        | TOKEN_FALSE
        | TOKEN_IF
        | TOKEN_IN
        | TOKEN_LET
        | TOKEN_NOT
        | TOKEN_THEN
        | TOKEN_TRUE
        | TOKEN_CSTBOOL
        | TOKEN_NAME
        | TOKEN_CSTINT
        | TOKEN_end_of_input
        | TOKEN_error
type nonTerminalId =
        | NONTERM__startMain
        | NONTERM_Main
        | NONTERM_Expr
        | NONTERM_AtExpr
        | NONTERM_AppExpr
        | NONTERM_Const
val tagOfToken : t:token -> int
val tokenTagToTokenId : tokenIdx:int -> tokenId
val prodIdxToNonTerminal : prodIdx:int -> nonTerminalId
val _fsyacc_endOfInputTag : int
val _fsyacc_tagOfErrorTerminal : int
val token_to_string : t:token -> string
val _fsyacc_dataOfToken : t:token -> System.Object
val _fsyacc_gotos : uint16 []
val _fsyacc_sparseGotoTableRowOffsets : uint16 []
val _fsyacc_stateToProdIdxsTableElements : uint16 []
val _fsyacc_stateToProdIdxsTableRowOffsets : uint16 []
val _fsyacc_action_rows : int
val _fsyacc_actionTableElements : uint16 []
val _fsyacc_actionTableRowOffsets : uint16 []
val _fsyacc_reductionSymbolCounts : uint16 []
val _fsyacc_productionToNonTerminalTable : uint16 []
val _fsyacc_immediateActions : uint16 []
val _fsyacc_reductions : unit -> (Text.Parsing.IParseState -> obj) []
val tables : unit -> Text.Parsing.Tables<token>
val engine :
```

```
                lexer:(Text.Lexing.LexBuffer<'a> -> token) ->
          ----------lexbuf:Text.Lexing.LexBuffer<'a> -> startState:int -> obj
          val Main :
                lexer:(Text.Lexing.LexBuffer<'a> -> token) ->
          ----------lexbuf:Text.Lexing.LexBuffer<'a> -> Absyn.expr


        namespace FSI_0002
          val lexemeAsString : lexbuf:Text.Lexing.LexBuffer<char> -> string
          val commentStart : Text.Lexing.Position ref
          val commentDepth : int ref
          val keyword : s:string -> FunPar.token
          val trans : uint16 [] array
          val actions : uint16 []
          val _fslex_tables : Text.Lexing.UnicodeTables
          val _fslex_dummy : unit -> 'a
----------val Token : lexbuf:Text.Lexing.LexBuffer<char> -> FunPar.token
----------val SkipComment : lexbuf:Text.Lexing.LexBuffer<char> -> unit
----------val _fslex_Token :
----------          _fslex_state:int -> lexbuf:Text.Lexing.LexBuffer<char> -> FunPar.token
----------val _fslex_SkipComment :
----------          _fslex_state:int -> lexbuf:Text.Lexing.LexBuffer<char> -> unit


--------namespace FSI_0002
----------val fromString : str:string -> Absyn.expr
----------val fromFile : filename:string -> Absyn.expr
----------val e1 : Absyn.expr
----------val e2 : Absyn.expr
----------val ex1 : Absyn.expr
----------val ex2 : Absyn.expr
----------val ex3 : Absyn.expr
----------val ex4 : Absyn.expr
----------val ex5 : Absyn.expr

--------> open Parse;;
--------> let e1 = fromString "5+7";;

--------val e1 : Absyn.expr = Prim ("+",CstI 5,CstI 7)

--------> let e2 = fromString "let y = 7 in y + 2 end";;

--------val e2 : Absyn.expr = Let ("y",CstI 7,Prim ("+",Var "y",CstI 2))

--------> let e3 = fromString "let f x = x + 7 in f 2 end";;

--------val e3 : Absyn.expr =
----------Letfun ("f","x",Prim ("+",Var "x",CstI 7),Call (Var "f",CstI 2))
```

C. Using the lexer, parser and first-order evaluator together:

```
   fsyacc --module FunPar FunPar.fsy
   fslex --unicode FunLex.fsl
   fsharpi -r ~/fsharp/FsLexYacc.Runtime.dll Absyn.fs FunPar.fs FunLex.fs Parse.fs Fun

   open ParseAndRun;;
   run (fromString "5+7");;
   run (fromString "let y = 7 in y + 2 end");;
   run (fromString "let f x = x + 7 in f 2 end");;



        D:\GitHubRepoes\BPRD\Assignment4\Fun>bin\fsyacc --module FunPar FunPar.fsy
        building tables
        computing first function ... time: 00:00:00.0855374
        building kernels ... time: 00:00:00.0564305
        building kernel table ... time: 00:00:00.0147743
        computing lookahead relations .....................................................
        ..... time: 00:00:00.0449269
        building lookahead table ... time: 00:00:00.0146503
        building action table ... state 21: shift/reduce error on GT
        state 21: shift/reduce error on LT
        state 21: shift/reduce error on GE
        state 21: shift/reduce error on LE
        state 22: shift/reduce error on GT
        state 22: shift/reduce error on LT
        state 22: shift/reduce error on GE
        state 22: shift/reduce error on LE
        state 23: shift/reduce error on GT
        state 23: shift/reduce error on LT
        state 23: shift/reduce error on GE
        state 23: shift/reduce error on LE
        state 24: shift/reduce error on GT
        state 24: shift/reduce error on LT
        state 24: shift/reduce error on GE
        state 24: shift/reduce error on LE
        time: 00:00:00.0583091
        building goto table ... time: 00:00:00.0016027
        returning tables.
        16 shift/reduce conflicts
        58 states
        6 nonterminals
        29 terminals
        26 productions
        #rows in action table: 58
```

```
        D:\GitHubRepoes\BPRD\Assignment4\Fun>bin\fslex --unicode FunLex.fsl
        compiling to dfas (can take a while...)
        30 states
        writing output

        D:\GitHubRepoes\BPRD\Assignment4\Fun>fsi -r bin\FsLexYacc.Runtime.dll Absyn.fs
        FunPar.fs FunLex.fs Parse.fs Fun.fs ParseAndRun.fs

        Microsoft (R) F# Interactive version 12.0.30815.0
        Copyright (c) Microsoft Corporation. All Rights Reserved.

        For help type #help;;

        [Loading D:\GitHubRepoes\BPRD\Assignment4\Fun\Absyn.fs
         Loading D:\GitHubRepoes\BPRD\Assignment4\Fun\FunPar.fs
         Loading D:\GitHubRepoes\BPRD\Assignment4\Fun\FunLex.fs
         Loading D:\GitHubRepoes\BPRD\Assignment4\Fun\Parse.fs
         Loading D:\GitHubRepoes\BPRD\Assignment4\Fun\Fun.fs
         Loading D:\GitHubRepoes\BPRD\Assignment4\Fun\ParseAndRun.fs]

        namespace FSI_0002
          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
            | Call of expr * expr


        namespace FSI_0002
          type token =
            | EOF
            | LPAR
            | RPAR
            | EQ
            | NE
            | GT
            | LT
            | GE
            | LE
            | PLUS
            | MINUS
            | TIMES
            | DIV
            | MOD
```

␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣ELSE
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣END
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣FALSE
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣IF
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣IN
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣LET
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣NOT
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣THEN
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TRUE
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣CSTBOOL␣of␣bool
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣NAME␣of␣string
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣CSTINT␣of␣int
␣␣␣␣␣␣␣␣␣␣type␣tokenId␣=
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_EOF
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_LPAR
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_RPAR
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_EQ
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_NE
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_GT
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_LT
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_GE
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_LE
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_PLUS
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_MINUS
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_TIMES
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_DIV
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_MOD
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_ELSE
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_END
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_FALSE
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_IF
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_IN
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_LET
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_NOT
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_THEN
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_TRUE
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_CSTBOOL
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_NAME
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_CSTINT
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_end_of_input
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣TOKEN_error
␣␣␣␣␣␣␣␣␣␣type␣nonTerminalId␣=
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣NONTERM__startMain
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣NONTERM_Main
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣NONTERM_Expr
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣NONTERM_AtExpr
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣NONTERM_AppExpr
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ | ␣NONTERM_Const

```
          val tagOfToken : t:token -> int
          val tokenTagToTokenId : tokenIdx:int -> tokenId
          val prodIdxToNonTerminal : prodIdx:int -> nonTerminalId
          val _fsyacc_endOfInputTag : int
          val _fsyacc_tagOfErrorTerminal : int
          val token_to_string : t:token -> string
          val _fsyacc_dataOfToken : t:token -> System.Object
          val _fsyacc_gotos : uint16 []
          val _fsyacc_sparseGotoTableRowOffsets : uint16 []
          val _fsyacc_stateToProdIdxsTableElements : uint16 []
          val _fsyacc_stateToProdIdxsTableRowOffsets : uint16 []
          val _fsyacc_action_rows : int
          val _fsyacc_actionTableElements : uint16 []
          val _fsyacc_actionTableRowOffsets : uint16 []
          val _fsyacc_reductionSymbolCounts : uint16 []
          val _fsyacc_productionToNonTerminalTable : uint16 []
          val _fsyacc_immediateActions : uint16 []
          val _fsyacc_reductions : unit -> (Text.Parsing.IParseState -> obj) []
          val tables : unit -> Text.Parsing.Tables<token>
          val engine :
                lexer:(Text.Lexing.LexBuffer<'a> -> token) ->
                  lexbuf:Text.Lexing.LexBuffer<'a> -> startState:int -> obj
          val Main :
                lexer:(Text.Lexing.LexBuffer<'a> -> token) ->
                  lexbuf:Text.Lexing.LexBuffer<'a> -> Absyn.expr


        namespace FSI_0002
          val lexemeAsString : lexbuf:Text.Lexing.LexBuffer<char> -> string
          val commentStart : Text.Lexing.Position ref
          val commentDepth : int ref
          val keyword : s:string -> FunPar.token
          val trans : uint16 [] array
          val actions : uint16 []
          val _fslex_tables : Text.Lexing.UnicodeTables
          val _fslex_dummy : unit -> 'a
        val Token : lexbuf:Text.Lexing.LexBuffer<char> -> FunPar.token
        val SkipComment : lexbuf:Text.Lexing.LexBuffer<char> -> unit
        val _fslex_Token :
              _fslex_state:int -> lexbuf:Text.Lexing.LexBuffer<char> -> FunPar.token
        val _fslex_SkipComment :
              _fslex_state:int -> lexbuf:Text.Lexing.LexBuffer<char> -> unit


      namespace FSI_0002
        val fromString : str:string -> Absyn.expr
        val fromFile : filename:string -> Absyn.expr
        val e1 : Absyn.expr
```

```
  val e2 : Absyn.expr
  val ex1 : Absyn.expr
  val ex2 : Absyn.expr
  val ex3 : Absyn.expr
  val ex4 : Absyn.expr
  val ex5 : Absyn.expr


namespace FSI_0002
  type 'v env = (string * 'v) list
  val lookup : env:(string * 'a) list -> x:string -> 'a
  type value =
        | Int of int
        | Closure of string * string * Absyn.expr * value env
  val eval : e:Absyn.expr -> env:value env -> int
  val run : e:Absyn.expr -> int
  val ex1 : Absyn.expr
  val ex2 : Absyn.expr
  val ex3 : Absyn.expr
  val rundeep : n:int -> int
  val ex4 : Absyn.expr
  val ex5 : Absyn.expr


namespace FSI_0002
  val fromString : (string -> Absyn.expr)
  val eval : (Absyn.expr -> Fun.value Fun.env -> int)
  val run : e:Absyn.expr -> int

> open ParseAndRun;;
> run (fromString "5+7");;
val it : int = 12
> run (fromString "let y = 7 in y + 2 end");;
val it : int = 9
> run (fromString "let f x = x + 7 in f 2 end");;
val it : int = 9
```

D. Loading the evaluator for a higher-order functional language (same abstract syntax as the first-order language):

```
fsharpi Absyn.fs HigherFun.fs

open HigherFun;;
eval ex1 [];;
open Absyn;;
run (Letfun ("twice", "f",
             Letfun ("g", "x", Call (Var "f", Call (Var "f", Var "x")), Var "g"),
```

```
                        Letfun ("mul3", "z", Prim ("*", Var "z", CstI 3),
                                Call (Call (Var "twice",Var "mul3"),CstI 2))));;
#q;;
```

(The above abstract syntax term corresponds to the concrete syntax term shown in point E below).

E. Using the lexer, parser and higher-order evaluator together:

```
fsyacc --module FunPar FunPar.fsy
fslex --unicode FunLex.fsl
fsharpi -r ~/fsharp/FsLexYacc.Runtime.dll Absyn.fs FunPar.fs FunLex.fs Parse.fs Hig

open ParseAndRunHigher;;
run (fromString @"let twice f = let g x = f(f(x)) in g end
                    in let mul3 z = z*3 in twice mul3 2 end end");;
#q;;
```

F. Using the lexer, parser and polymorphic type inference together:

```
fsyacc --module FunPar FunPar.fsy
fslex --unicode FunLex.fsl
fsharpi -r ~/fsharp/FsLexYacc.Runtime.dll Absyn.fs FunPar.fs FunLex.fs Parse.fs Typ

open ParseAndType;;
inferType (fromString "let f x = 1 in f 7 + f false end");;
#q;;
```

2 4.2

```
1) Summation of 1000 to 1..
> let sumt = run (fromString "let sum n = if n = 0 then n else sum (n-1) + n in sum 10
val sumt : int = 500500
This wont do negative numbers.

2) Computing 3^8 in a naive way. with recursion.
> let facttt = run (fromString "let fact n = if 1 = n then 3 else fact (n-1) * 3 in fa
val factt : int = 6561

Since the function only takes one argument 3 has to be hardcoded.

3)
> let sumfact = run (fromString "let fact n = if 1 = n then 3 else fact (n-1) * 3 in l
val sumfact : int = 265720

4)
```

```
> let sum10fact8 = run (fromString "let sum a = if 0 < a then let pows y = if 0 < y th

val sum10fact8 : int = 167731334
```

3 4.3

```
  //4.3
  | Letfun of string * string list * expr * expr    (* (f, x, fBody, letBody) *)
  | Call of expr * expr list

    //4.3 in Fun.fs
    | Letfun(f, x, fBody, letBody) ->
      let bodyEnv = (f, Closure(f, x, fBody, env)) :: env
      eval letBody bodyEnv
    | Call(Var f, eArg) ->
      let fClosure = lookup env f
      match fClosure with
      | Closure (f, x, fBody, fDeclEnv) ->
        (*let rec xValList args =
          match args with
          | [] -> failwith " Empty param"
          | e::rest -> Int(eval e env) :: xValList rest*)
        let values = List.map (fun e -> Int(eval e env)) eArg//xValList eArg
        let fBodyEnvZip = List.zip x values
        let fBodyEnv = fBodyEnvZip @ (f, fClosure) :: fDeclEnv
        eval fBody fBodyEnv
      | _ -> failwith "eval Call: not a function"
```

4 4.4

```
//4.4 FunPar.fsy
MulArgs:
    AtExpr                              { [$1]      }
  | AtExpr MulArgs                      { $1 :: $2  }
;

AppExpr:
    AtExpr MulArgs                     { Call($1, $2)          }
;
```

5 4.5

```
  //4.5 FunLex.fsl
  | "&&"            { AND } // 4.5
  | "||"            { OR } // 4.5
```

```
//4.5 FunPar.fsy
| Expr AND Expr                    { If($1, $3 , CstB false) } // 4.5 FunPar.fsy
| Expr OR Expr                     { If($1, CstB true , $3)  } // 4.5 FunPar.fsy
```