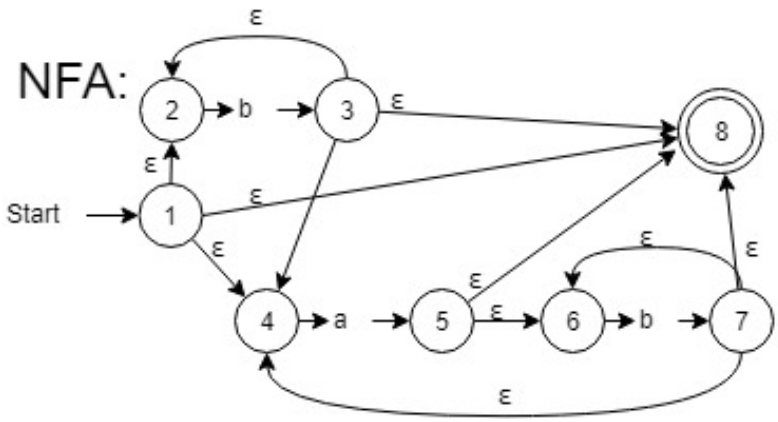
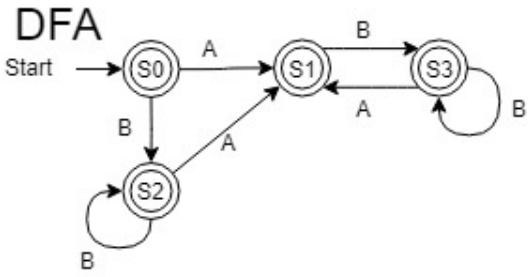
3.2

- Regular expression: b*|b*a(bb*a)*

NFA			
	Move(a)	Move(b)	ε*
1	-	-	1,2,4,8
2	-	3	2
3	-	-	2,3,4,8
4	5	-	4
5	-	-	5,6,8
6	-	7	6
7	-	-	4,6,7,8
8	-	-	8



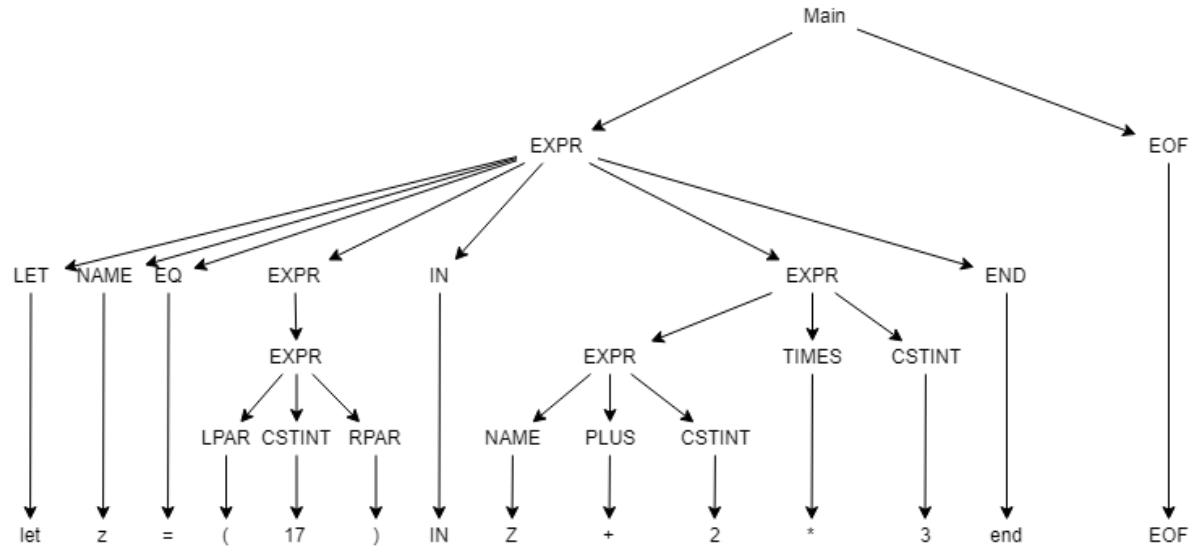
dFA				
State:	a	b	NFA states	Accepting
S0	S1	S2	1,2,4,8	Y
S1	-	S3	5,6,8	Y
S2	S1	S2	2,3,4,8	Y
S3	S1	S3	4,6,7,8	Y



3.3)

- let z = (17) in z + 2 * 3 end EOF
- Converting to EXPR:
  (E to convert Let, B to convert names, C to convert ints, G to convert times, H for plus, E for parentheses)
  LET (NAME “z”) EQ (LPAR (CSTINT 17) RPAR)
  IN (
  ((NAME “z”) PLUS (CSTINT 2))
  TIMES (CSTINT 3)
)
  END EOF

3.4) Draw the above derivation as a tree.



3.5

Loading expression abstract syntax, the lexer and parser modules, and expression interpreter and compilers into an interactive F# session:

```
        fslex --unicode ExprLex.fsl
        fsyacc --module ExprPar ExprPar.fsy
        fsi -r bin/FsLexYacc.Runtime.dll Absyn.fs Expr.fs ExprPar.fs ExprLex.fs Parse.fs
        build bat
        run.bat
open Parse;;
fromString "1 + 2 * 3";;
        val it : Absyn.expr = Prim ("+",CstI 1,Prim ("*",CstI 2,CstI 3))
fromString "1 - 2 - 3";;
        val it : Absyn.expr = Prim ("-",Prim ("-",CstI 1,CstI 2),CstI 3)
fromString "1 + -2";;
        val it : Absyn.expr = Prim ("+",CstI 1,CstI -2)
fromString "x++";;
        System.Exception: parse error near line 1, column 3
fromString "1 + 1.2";;
        System.Exception: Lexer error: illegal symbol near line 1, column 6
fromString "1 + ";;
        System.Exception: parse error near line 1, column 4
fromString "let z = (17) in z + 2 * 3 end";;
        val it : Absyn.expr = Let ("z",CstI 17,Prim ("+",Var "z",Prim ("*",CstI 2,CstI 3)))
fromString "let z = 17) in z + 2 * 3 end";;
        System.Exception: parse error near line 1, column 11
fromString "let in = (17) in z + 2 * 3 end";;
        System.Exception: parse error near line 1, column 6
fromString "1 + let x=5 in let y=7+x in y+y end + x end";;
        val it : Absyn.expr =
          Prim
            ("+",CstI 1,
             Let
               ("x",CstI 5,
                Prim
                  ("+",Let ("y",Prim ("+",CstI 7,Var "x"),Prim ("+",Var "y",Var "y")),
                   Var "x")))
```

3.6

Use the expression parser from Parse.fs and the compiler scomp
(from expressions to stack machine instructions) and the associated datatypes from
Expr.fs, to define a function compString : string -> sinstr list
that parses a string as an expression and compiles it to stack machine code.

```
        open Parse;;
        compString "1 + let x=5 in let y=7+x in y+y end + x end";;
                val it : Expr.sinstr list =[SCstI 1; SCstI 5; SCstI 7; SVar 1; SAdd; SVar 0; SVar 1; SAdd;
                SSwap; SPop; SVar 1; SAdd; SSwap; SPop; SAdd]
```

3.7

- add IfElse
- Not needed to fix eval