# Opgave 1

## Udvid lexer og parser

Der er lavet følgende ændringer i `FunLex.fsl`

```
rule Token = parse
  ...
  | "++"             { UNION }
  | '{'              { LBRA }
  | '}'              { RBRA }
  | ','              { COMMA }
  ...
  | _                { failwith "Lexer error: illegal symbol" }
```

Der er lavet følgende ændringer i `FunPar.fsy`

```
%token COMMA LBRA RBRA UNION

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

Expr:
  ...
  | Expr PLUS  Expr                    { Prim("+",  $1, $3)     }
  | Expr UNION  Expr                   { Prim("++",  $1, $3)    }
  ...
  | LBRA Lst RBRA                      { Set($2)                 }
;

Lst:
    Expr                              { [$1] }
  | Expr COMMA Lst                    { $1 :: $3}
;
```

## Angiv evalueringstræ

Nej

## Udvid higherfun

```
type value =
  | Int of int
  | Closure of string * string * expr * value env       (* (f, x, fBody, fDeclEnv) *)
```

```
  | SetV of Set<value> (* Exam *)

let rec eval (e : expr) (env : value env) : value =
    match e with
    | CstI i -> Int i
    | CstB b -> Int (if b then 1 else 0)
    | Var x  -> lookup env x
    | Prim(ope, e1, e2) ->
      let v1 = eval e1 env
      let v2 = eval e2 env
      match (ope, v1, v2) with
      ...
      | ("++",SetV s1, SetV s2) ->
        (* Anvender den eksisterende union operation *)
        SetV(Set.union s1 s2)
      ...
      | ("=", SetV s1, SetV s2) ->
        (* Hvis de begge er et subset af hinanden så må de være det præcis samme sæt *)
        Int (if Set.isSubset s1 s2 && Set.isSubset s2 s1 then 1 else 0)
      ...
      |  _ -> failwith "unknown primitive or wrong type"
    ...
    | Set(lst) ->
      (*
        Da vi skal have et sæt af værdier skal vi evaluerer listen af expressions
        Dette kan vi herefter konvertere til et sæt og smide ind i SetV værdien
      *)
      lst
      |> List.map (fun e -> eval e env)
      |> Set.ofList
      |> SetV
```

Kørsel af programmet

```
   opgave1 git:(main)  dotnet run
"
        let s1 = {2, 3} in
            let s2 = s1++{1, 4} in
                s1 ++ s2 = {2, 4, 3, 1}
            end
        end
    "


Let
  ("s1", Set [CstI 2; CstI 3],
   Let
```

```
("s2", Prim ("++", Var "s1", Set [CstI 1; CstI 4]),
 Prim
   ("=", Prim ("++", Var "s1", Var "s2"),
    Set [CstI 2; CstI 4; CstI 3; CstI 1])))
```

```
Int 1
```

## Udvid typereglerne

$$(++)\frac{\rho \vdash e_1 : t\,\text{set} \quad \rho \vdash e_2 : t\,\text{set}}{\rho \vdash e_1 + + e_2 : t\,\text{set}}$$

$$(=)\frac{\rho \vdash e_1 : t\,\text{set} \quad \rho \vdash e_2 : t\,\text{set}}{\rho \vdash e_1 = e_2 : \text{bool}}$$

Det giver ikke mening at sammenligne eller forene to sæt med forskellige typer, på samme måde som det ikke giver mening at lægge en bool og en int sammen eller at tjekke om en bool er lig en int. Derfor skal begge to være af samme type når vi udfører operationer på dem. Det skal dog siges at der ikke er noget typemæssigt der afholder nogen fra at placere flere forskellige typer i den nuværende implementation.

Derfor skulle et typsystem påkræve at begge sæt er ens fra et type perspektiv. Forenelses operation vil returnere et sæt som et de to sæt forenede, da vi lige har defineret at de skal være af samme type, så vil det nye sæt også være af den type.

Når der laves en sammenligning vil der altid skulle returneres en boolsk værdi, derfor returnere (=) typen *bool*.