Programmer som data trial exam

Albert Ross Johannessen

December 8, 2024

1

Betragt dette regulære udtryk over alfabetet $\{d,',',\}$, hvor d står for decimalt ciffer og ',' er komma:

$$d+','?d* \tag{1}$$

Ved antagelse af, at d svarer til tallene fra 0 til 9 og ',' er et komma, så beskriver det regulære udtryk kommatal.

1.1

Udtrykket kan beskrive tal i sættet $\mathbb{R}_0^+$, dette vil altså sige vilkårligt store reelle tal.
Følgende eksempler er inkluderet i sættet:

- 0
- $1,1$
- $9,999999999$
- 20
- $123456,0$
- $2,$

1.2

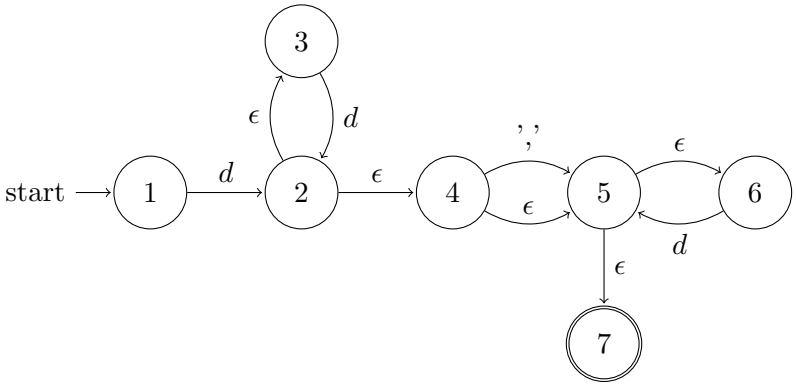

Figure 1: The labeled NFA representing $d+$

1.2.1 Vil tilstandsmaskinen acceptere netop de strenge, som genkendes af det regulære udtryk ovenfor?

Ja man kan dele det regulære udtryk op i diskrete dele. Dette kan vi sammenligne med tilstandsmaskinen angivet nedenfor.

Det er åbenlyst at se at disse alle er dele af den tilstandsmaskine angivet.

Tilstandsmaskinen angivet er ikke deterministisk da man kan være i flere knuder på en gang på grund af eksistensen af epsilon kanter.

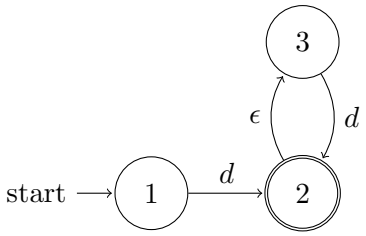

Figure 2: The labeled NFA representing $d+$

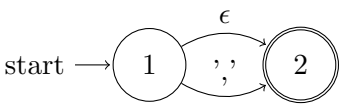

Figure 3: The labeled NFA representing $','?$

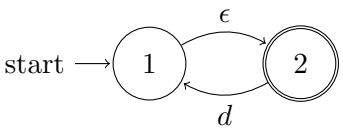

Figure 4: The labeled NFA representing $d*$

1.3

$$(d+','?d*)? \tag{2}$$

1.4

Løsningen kan findes i kommatal.fsl

2

```
let exam2_1 = Every(Write(FromTo(1, 6)));

let exam2_2 = Every(
  Write(
    Prim("+",
      Prim("*", CstI 10, FromTo(3, 6)),
      FromTo(3,4)
    )
  )
)

let exam2_3 = Every(Write(FromToBy(1, 10, 3)))

let exam2_4 = Every(
  Write(
    Prim("+",
      FromToBy(30, 60, 10),
      FromTo(3,4)
    )
  )
```

```
)

let exam2_5 = Every(
  Write(
      FromToBy(10, 11, 0)
  )
)
```

3

```
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
    | Print of expr

let keyword s =
    match s with
    | "else"  -> ELSE
    | "end"   -> END
    | "false" -> CSTBOOL false
    | "if"    -> IF
    | "in"    -> IN
    | "let"   -> LET
    | "not"   -> NOT
    | "then"  -> THEN
    | "true"  -> CSTBOOL true
    | "print" -> PRINT
    | _       -> NAME s

%token ELSE END FALSE IF IN LET NOT THEN TRUE PRINT

Expr:
    AtExpr                              { $1                    }
  | AppExpr                             { $1                    }
  | IF Expr THEN Expr ELSE Expr         { If($2, $4, $6)        }
  | MINUS Expr                          { Prim("-", CstI 0, $2) }
  | Expr PLUS  Expr                     { Prim("+",  $1, $3)    }
  | Expr MINUS Expr                     { Prim("-",  $1, $3)    }
  | Expr TIMES Expr                     { Prim("*",  $1, $3)    }
  | Expr DIV   Expr                     { Prim("/",  $1, $3)    }
  | Expr MOD   Expr                     { Prim("%",  $1, $3)    }
  | Expr EQ    Expr                     { Prim("=",  $1, $3)    }
  | Expr NE    Expr                     { Prim("<>", $1, $3)    }
  | Expr GT    Expr                     { Prim(">",  $1, $3)    }
  | Expr LT    Expr                     { Prim("<",  $1, $3)    }
```

```
  | Expr GE    Expr                    { Prim(">=", $1, $3)     }
  | Expr LE    Expr                    { Prim("<=", $1, $3)     }
  | PRINT      Expr                    { Print($2)              }
;
```

```
| Print(ex) ->
let evaluated = eval ex env
printfn "%A" evaluated
evaluated;;
```

4