RUST PROBLEMS

8. Rust

(a) [3 points] Consider the following code:

```
let a = String::from("cmsc330");
{ let b = a;
   { let c = b;
     let d = &c;
   }
   //HERE
}
```

At the point in the program marked `HERE` what variable, if any, 'owns' the string created on the first line:

Problem 9: Operational Semantics

Consider the following rules for 2 Languages:

Language 1:

$$\frac{}{\text{true} \rightarrow \text{true}}$$

$$\frac{}{\text{false} \rightarrow \text{false}}$$

$$\frac{A(x) = v}{A; x \Rightarrow v}$$

$$\frac{A; e_1 \Rightarrow v_1 \quad A; e_2 \Rightarrow v_2 \quad v_3 = v_1 \, and \, v_2}{A; e_1 \,\&\&\, e_2 \Rightarrow v_3}$$

$$\frac{A; e_1 \Rightarrow v_1 \quad A; e_2 \Rightarrow v_2 \quad v_3 = v_1 \, or \, v_2}{A; e_1 \,||\, e_2 \Rightarrow v_3}$$

$$\frac{A; e_1 \Rightarrow v_1 \quad A, x : v_1; e_2 \Rightarrow v_2}{A; let\, x = e_1 \text{ in } e_2 \Rightarrow v_2}$$

Language 2

$$\frac{}{\text{true} \rightarrow \text{true}}$$

$$\frac{}{\text{false} \rightarrow \text{false}}$$

$$\frac{A(x) = v}{A; x \Rightarrow v}$$

$$\frac{A; e_1 \Rightarrow v_1 \quad A; e_2 \Rightarrow v_2 \quad v_3 = v_1 \, and \, v_2}{A; ((\texttt{fun } x\ y \rightarrow \texttt{if } x \texttt{ then } y \texttt{ else } x)\ e_1\ e_2) \Rightarrow v_3}$$

$$\frac{A; e_1 \Rightarrow v_1 \quad A; e_2 \Rightarrow v_2 \quad v_3 = v_1 \, or \, v_2}{A; ((\texttt{fun } x\ y \rightarrow \texttt{if } x \texttt{ then } x \texttt{ else } y)\ e_1\ e_2) \Rightarrow v_3}$$

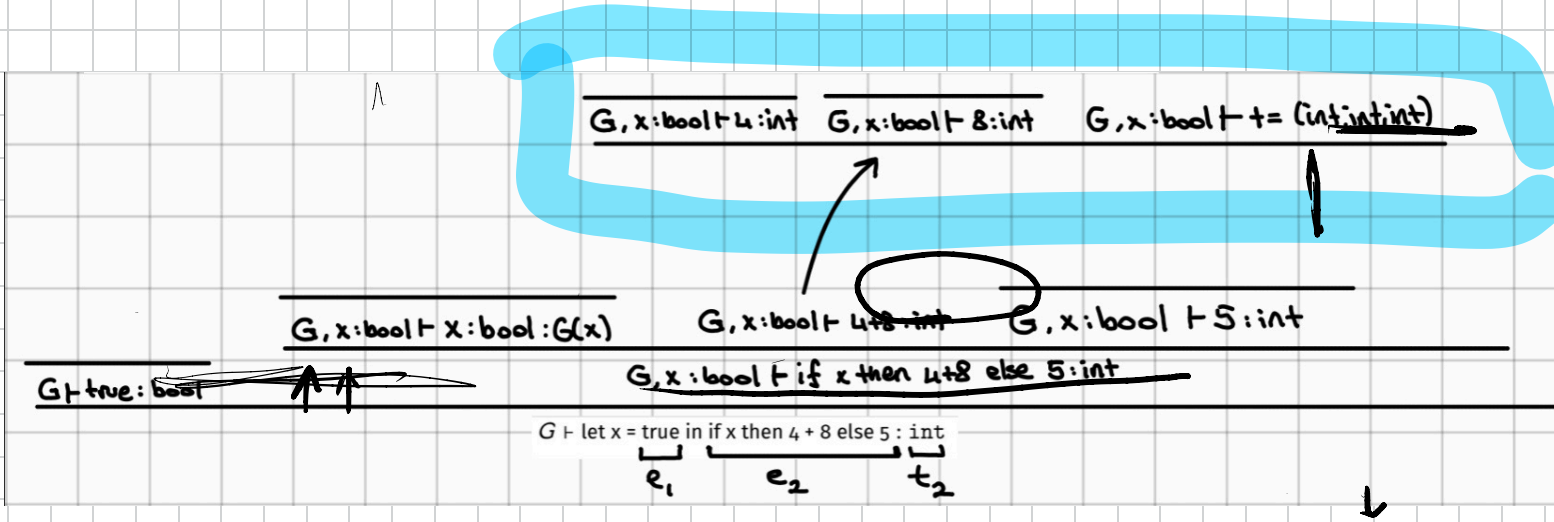
$$\frac{A; e_2 \Rightarrow v_1 \quad A, x : v_1; e_1 \Rightarrow v_2}{A; (fun x \rightarrow e_1)\ e_2 \Rightarrow v_2}$$

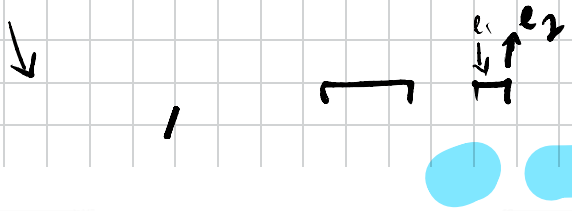
Convert the following language 1 sentence to its language 2 counterpart:

A; ((fun x → [fun x y → if x then y else x) false x)) true

$$\frac{}{A;\ \texttt{let } x = \texttt{true in (false if } x)}$$

TYPE CHECKING PROBLEMS



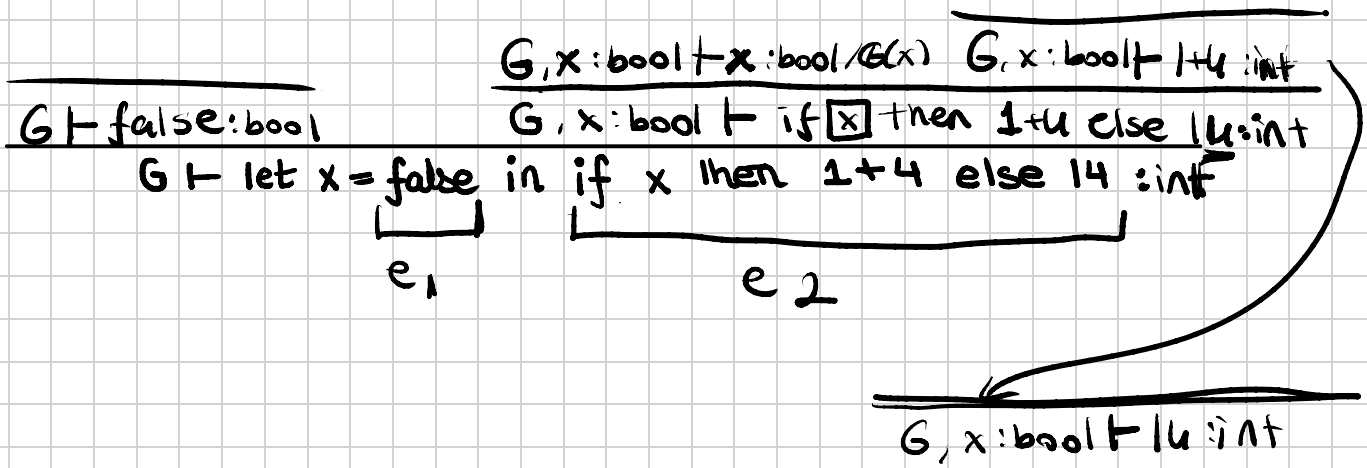


2)

$$\frac{}{G \vdash true : bool} \qquad \frac{}{G \vdash false : bool} \qquad \frac{}{G \vdash n : int}$$

$$\frac{}{G \vdash x : G(x)} \qquad \frac{G \vdash e_1 : int \quad G \vdash e_2 : int \quad + = (int, int, int)}{G \vdash e_1 + e_2 : int}$$

$$\frac{G \vdash e_1 : t_1 \quad G, x : t_1 \vdash e_2 : t_2}{G \vdash let\ x = e_1\ in\ e_2 : t_2} \qquad \frac{G \vdash e_1 : bool \quad G \vdash e_2 : t \quad G \vdash e_3 : t}{G \vdash if\ e_1\ then\ e_2\ else\ e_3 : t}$$



LAMBDA CALC PROBLEMS

Language

e → |x (a variable
|λx. e (a function)
|ee (an application)

* turing complete

SCOPING

λx. λy. x y → λx. (λy. (x y))
λx. (x a b (λy. y) z λz. (z))
((((((x a) b) (λy. y) z) (λz. z))

EVALUATING

(λx. e1) e2
e e'

(λx. λy. λz. x z) a

λy. λz. a z

(fun x → fun y → fun z → x z) a
fun y → fun z → a z

* call by name: (lazy) don't simplify the argument
* call by value (eager): do simplify

call by name

(λy. y a) ((λz. x (λy. y)) x)

((λz. x (λy. y)) x) a

(x (λy. y)) a

Beta Normal Form

call by value

(λy. y a) ((λz. x (λy. y)) x)
e1 e2

(λy. y a) (x (λy. y))

((x (λy. y)) a)

$(\lambda x.\ b)((\lambda y.\ y\ y)(\lambda z.\ z\ z))$

b

$(\lambda x.\ b)((\lambda y.\ y\ y)(\lambda z.\ z\ z))$

$(\lambda z.\ z\ z)(\lambda z.\ z\ z)$

Renaming bound variables
→ variables that get replaced

$(\lambda x.\ x\ (\lambda y.\ y\ z))$

$(\lambda x.\ x\ y\ (\lambda y.\ y\ x))$

$((\lambda x.\ \lambda y.\ x\ x)(\lambda x.\ x\ y))\ x)$

$((\lambda a.\ \lambda b.\ a\ a)(\lambda c.\ c\ y))x)$
$(\lambda b.\ (\lambda c.\ cy)(\lambda c.\ cy))\ x$
$(\lambda b.(\lambda c.\ cy)y)\ x$
$(\lambda b.\ y\ y)\ x$
$(y\ y)$

representing statements as lambdas

true = $\lambda x.\ \lambda y.\ x$
false = $\lambda x.\ \lambda y.\ y$
if a then b else c = a b c
not = $(\lambda x.\ x$ false true$)$

not true
$(\lambda x.\ x$ false true$)$ true

true false true
$(\lambda x.\ \lambda y.\ x)$ false true

$(\lambda y.$ false$)$ true
$(\lambda y.\ \lambda x.\ \lambda y.\ y)$ true
false

$0 = \lambda f.\ \lambda y.\ y$

$1 = \lambda f.\ \lambda y.\ f\ y$

$2 = \lambda f.\ \lambda y.\ f\ (f\ y)$

$n = \lambda f.\ \lambda y.\ f$, n times to y

$succ = \lambda z.\ \lambda f.\ \lambda y.\ f\ (z\ f\ y)$

succ 0 = 1

succ 0

$(\lambda z.\ \lambda f.\ \lambda y.\ f\ (z\ f\ y))\ (\lambda f.\ \lambda y.\ y)$

$\Rightarrow (\lambda f.\ \lambda y.\ f\ ((\lambda f.\ \lambda y.\ y)\ f\ y))$

$\Rightarrow (\lambda f.\ \lambda y.\ f\ ((\lambda y.\ y)\ y))$

$\Rightarrow (\lambda f.\ \lambda y.\ f\ y)$

$= 1$

true = $(\lambda x.\ \lambda y.\ x)$ false = $(\lambda x.\ \lambda y.\ y)$

not = $(\lambda x.\ x\ false\ true)$

xor = $(\lambda x.\ \lambda y.\ x\ (not\ y)\ y)$

xor true false = true

$((\lambda x.\ \lambda y.\ x\ (not\ y)\ y)\ true)\ false$

$\Rightarrow ((\lambda y.\ true\ (not\ y)\ y)\ false)$

$\Rightarrow true\ (not\ false)\ false$

$\Rightarrow ((\lambda x.\ \lambda y.\ x)\ (not\ false))\ false$

$\Rightarrow ((\lambda y.\ (not\ false))\ false)$

$\Rightarrow not\ false$

$\Rightarrow ((\lambda x.\ x\ false\ true)\ false)$

$\Rightarrow false\ false\ true$

$\Rightarrow ((\lambda x.\ \lambda y.\ y)\ false)\ true$

$\Rightarrow ((\lambda y.\ y)\ true)$

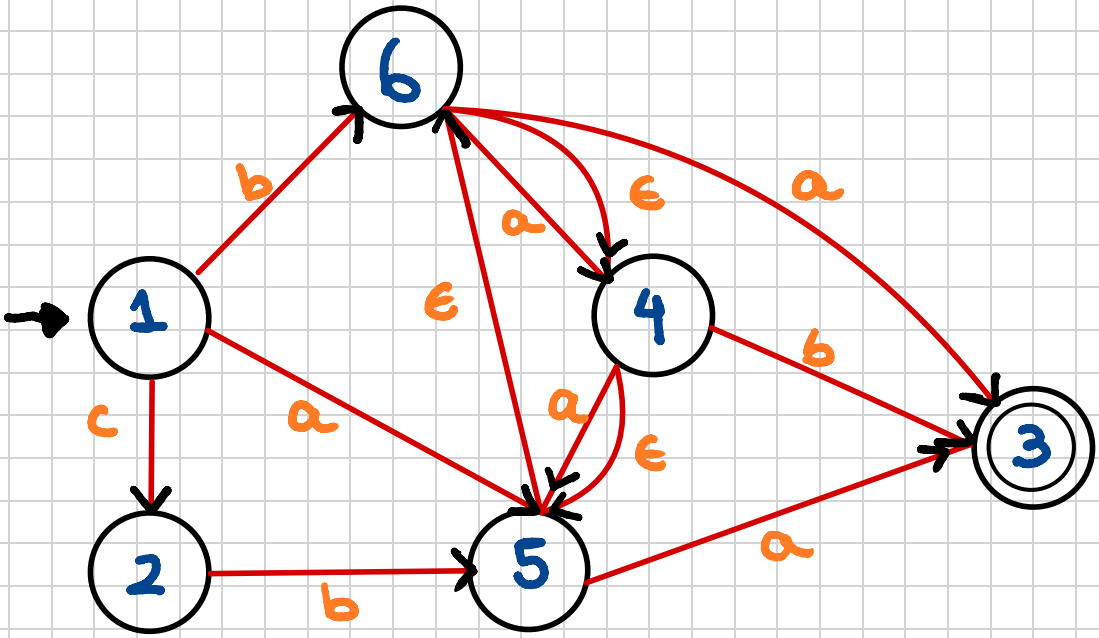
$\Rightarrow true$

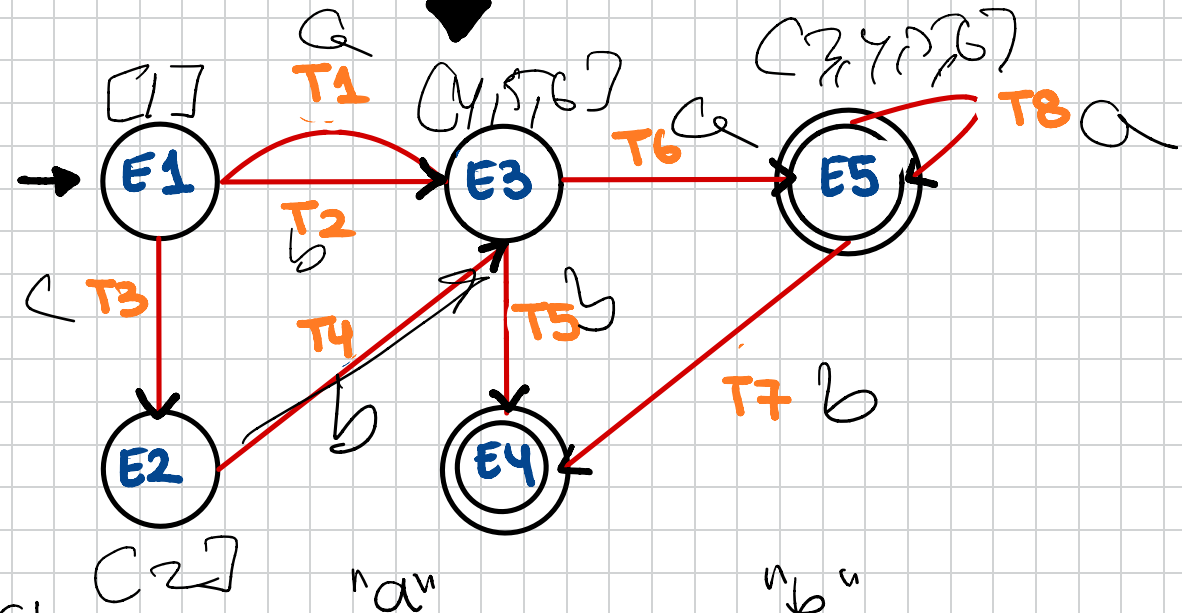
$(\lambda y.\ x)\ c \Rightarrow x$

$(\lambda x.\ x)$

FSM Problems

1) Convert the following NFA to a DFA

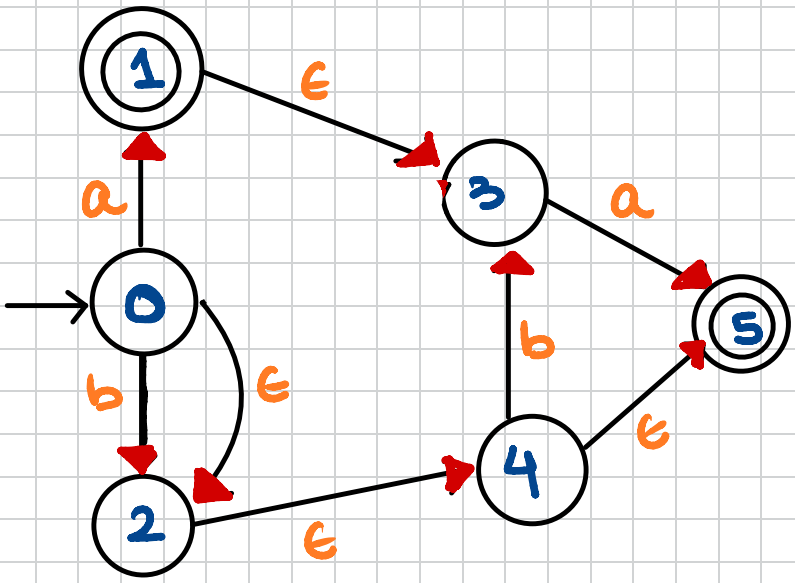




[1] [2]

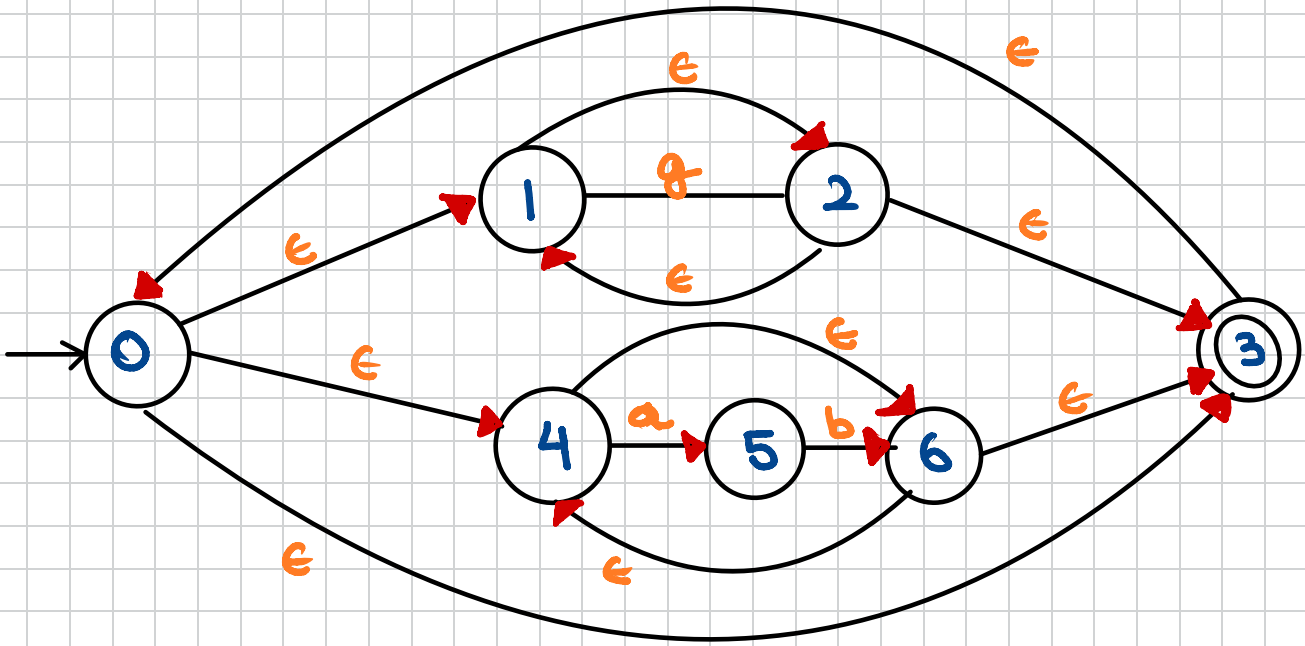
States	"a" m	"a" e	"b" m	"b" e	"c" m	"c" e
[1]	[5]	[4,5,6]	[6]	[4,5,6]	[2]	[2]
[2]	[]		[5]	[4,5,6]	[]	
[4,5,6]	[5,3,4]	[3,4,5,6]	[3]	[3]	[]	
[3,4,5,6]						
[3]						

2) Convert the following NFA to DFA



start	ε - closure	a	b
0	{0,2,4,5}	{1}	{2,3}
{1}	{1,3}	{5}	ϕ
{2,3}	{2,3,4,5}	{5}	{3}
{5}	{5}	ϕ	ϕ
{3}	{3}	{5}	ϕ

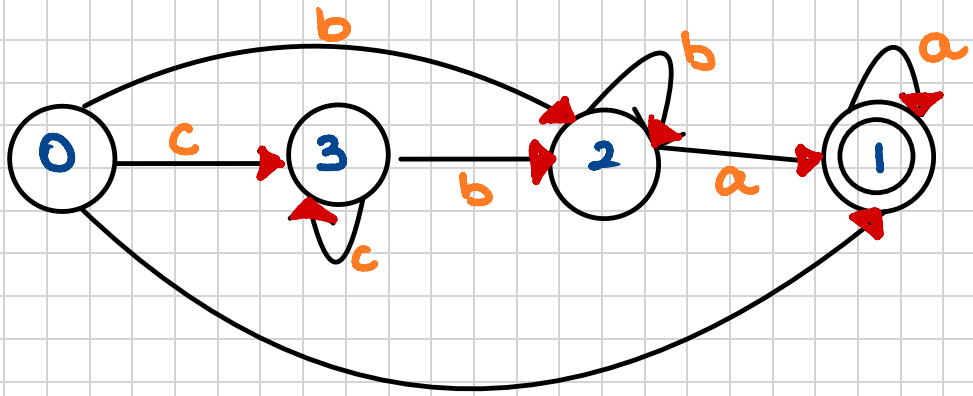
3)



What is the regular expression for the above FSM?

((g)*|(ab)*)*

4) Find the regex for the corresponding DFA



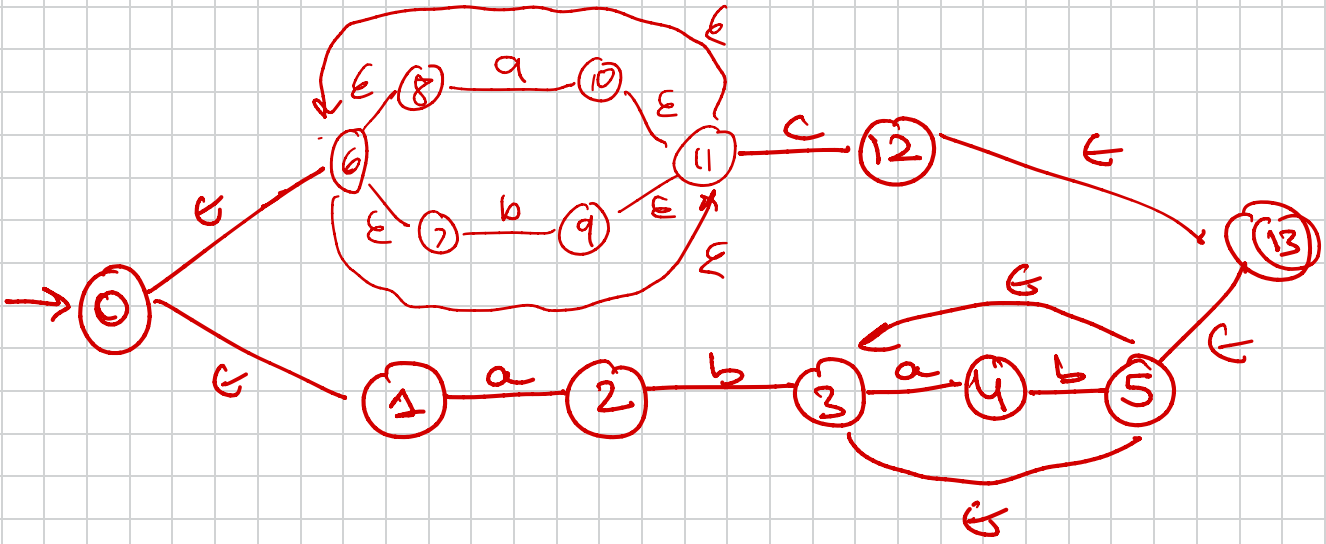
bottom: $a\,a^*$ → $a+$

top: bb^*aa^* → $b+a+$

mid: $a+b+a+$

$$a+ \mid b+a+ \mid a+b+a+$$

5) Convert the following Regex to NFA: $(a|b)^*c \mid (ab)+$



PL & OCAML TYPING PROBLEMS

error

→ let f g = (g 1, g "hi") in f (fun x → x)

= let f = (let g = fun x → x in (g 1, g "hi"))

let f = (let m = ref 0 in (fun n → m := n; !m + n))

>>> int → int

('a * 'b) → ('a → 'b → 'c) → 'c → bool list

let f (a,b) g c = [(g a b) = c]

let f g a c = g a