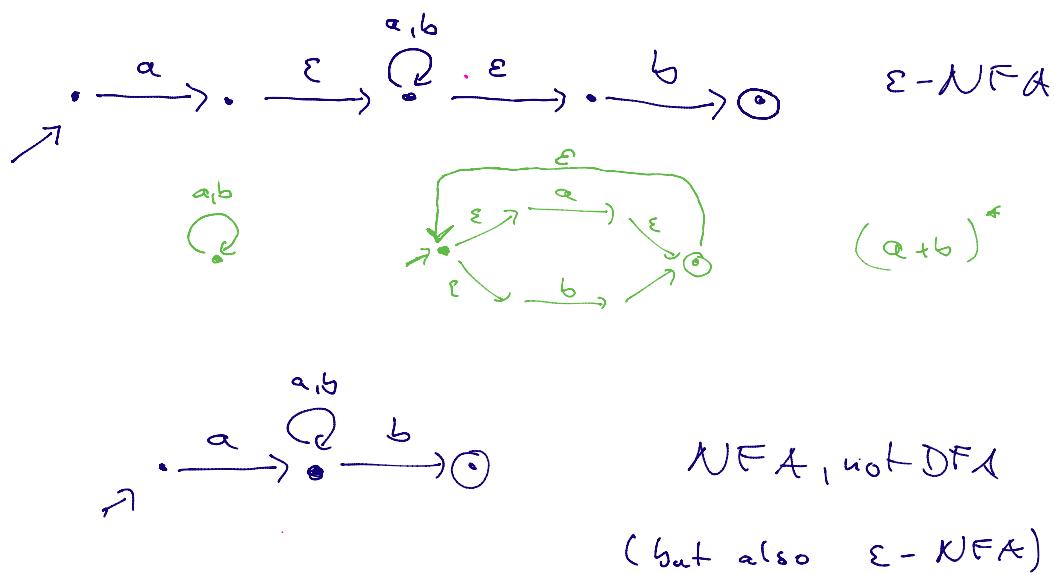
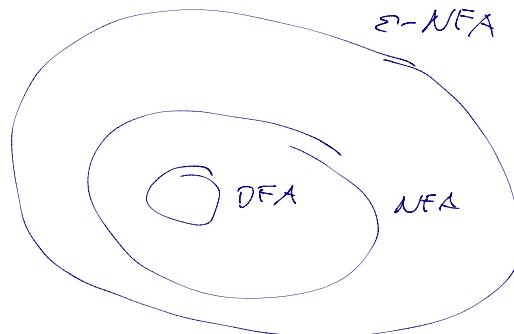


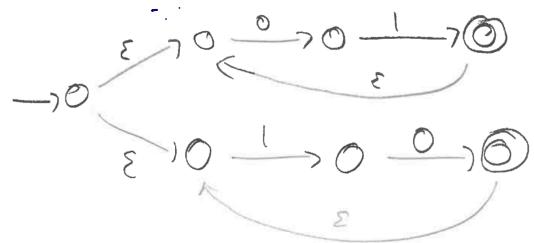
① Write down an  $\epsilon$ -NFA  
for  $a(a+b)^*b$



Venn Diagram  
of Automata:



② Consider the following  $\epsilon$ -NFA



2a)  $01(01)^* + 10(10)^*$

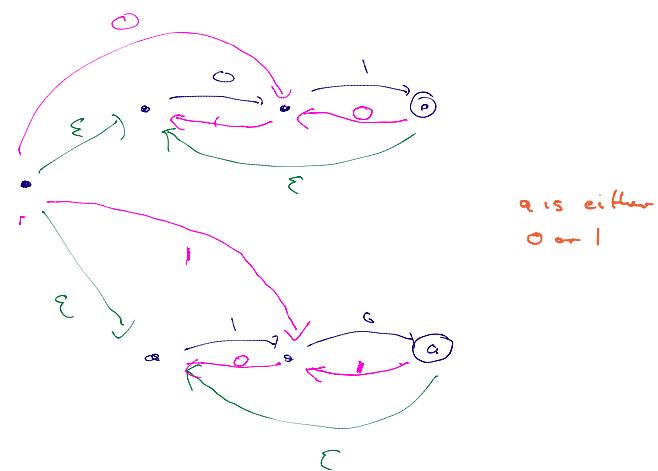
$$(01)^* 01 + (10)^* 10$$

$$(01)^+ + (10)^+$$

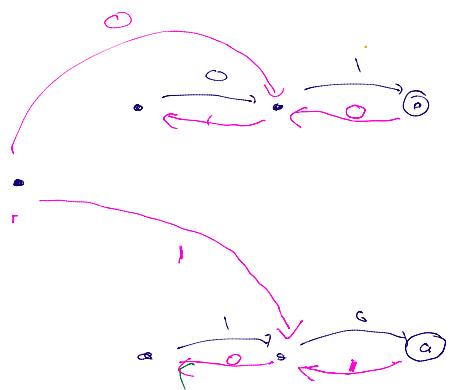
2b) Rules:



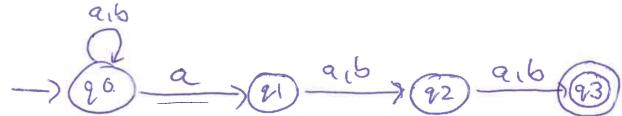
Saturate the  
automaton  
by applying  
the rules



Eliminating  
the  $\epsilon$ -transitions



- ③ Write down a DFA that accepts the same language as



NFA

initial

State	a	b
0	0,1	0
0,1	0,1,2	0,2
0,1,2	0,1,2,3	0,2,3
0,2	0,1,3	0,3
0,1,2,3	0,1,2,3	0,2,3
0,2,3	0,1,3	0,3
0,1,3	0,1,2	0,2
0,3	0,1	0

final

final

final

final

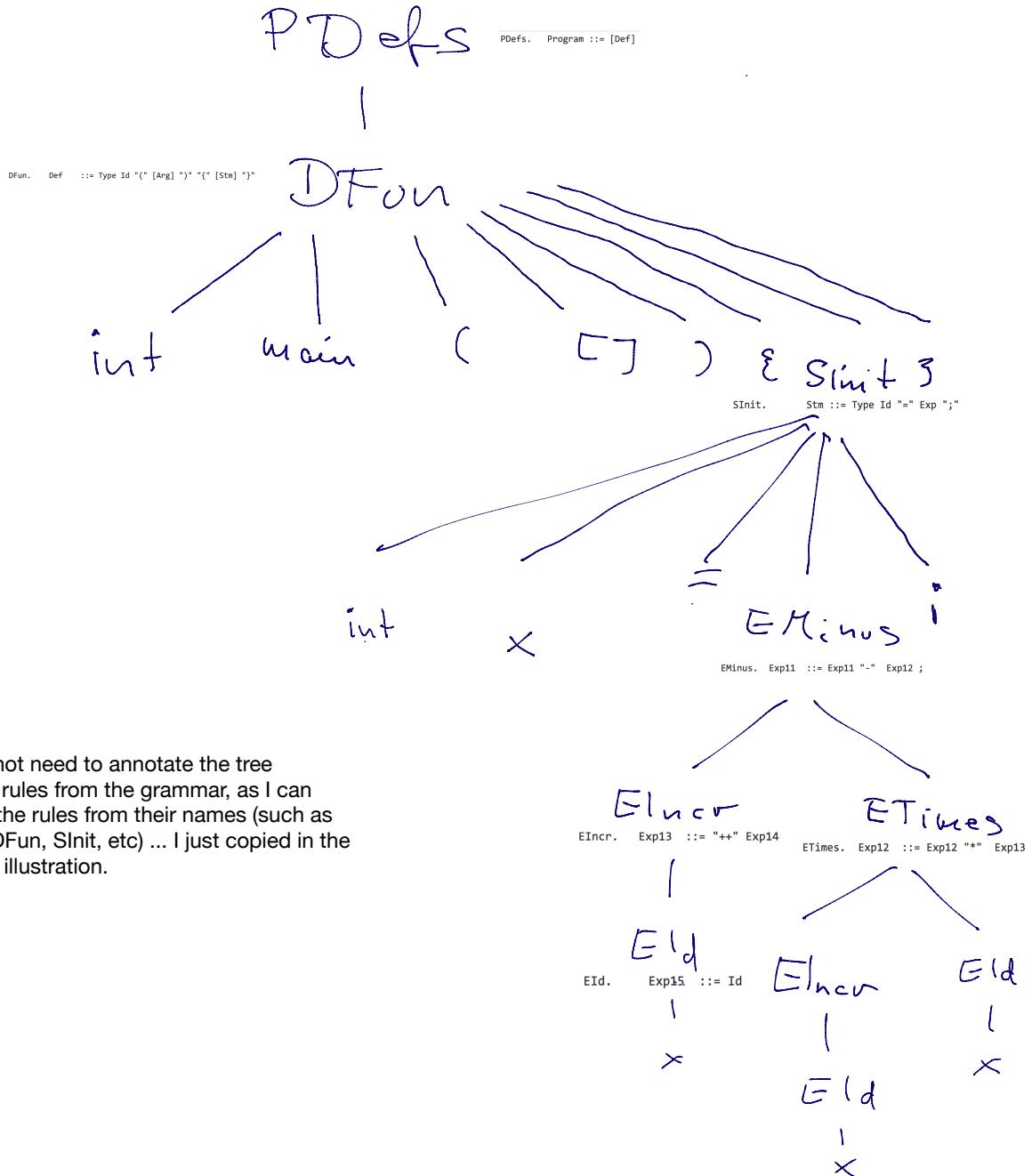
DFA

```

int main(){
    int x = ++x-++x*x ;
}

```

Abstract Syntax Tree :



You do not need to annotate the tree with the rules from the grammar, as I can identify the rules from their names (such as PDefs, DFun, SInit, etc) ... I just copied in the rules for illustration.

Stack	Input	Rule
	int main () { int x = ++x - ++x - x; }	
Type	main ...	
Type Id	( ) ... ;	
Type Id () ;	int x = ++x - ++x - x ;	
Type Id () ; Type Id =	++x - ++x - x ;	
... ++ Id	- ++x - x ;	
... ++ Exp14	"	EId.    Exp15 ::= Id <del>EId</del>
... Exp11	- ++ x < x ;	EIncr.    Exp13 ::= "+" Exp14 ; <del>EIncr</del>
... Exp11 - ++ Id	* x ;	
... Exp11 - ++ Exp14	"	EId <del>EId</del>
... Exp11 - Exp12	"	EIncr <del>EIncr</del>
... Exp12 * Id	; ;	
... Exp12 * Exp13	"	EId <del>EId</del>
... Exp12	"	ETimes.    Exp12 ::= Exp12 "*" Exp13 ; <del>ETimes</del>
Type Id () ; Type Id = Exp	"	EMinus.    Exp11 ::= Exp11 "-" Exp12 <del>EMinus</del>
... ;	{ ; .	
TypeId () { Stmt	SInit.    Stmt ::= Type Id "=" Exp ";" <del>SInit</del>	
TypeId () { Stmt }		
Def	DFun.    Def ::= Type Id "(" [Arg] ")" "(" [Stm] ")" <del>DFun</del>	
Program	PDefs.    Program ::= [Def] <del>PDefs</del>	

The parse was successful since "Program" is on the stack and the input is empty.

In the steps with rules marked with the red \*, I also did some additional conversions on the stack.

Bottom up, conversions decrease the levels, eg from Exp15 to Exp14 in the first application of EId.

These decrements of levels have to be done while the relevant non-terminal is on the top of the stack.

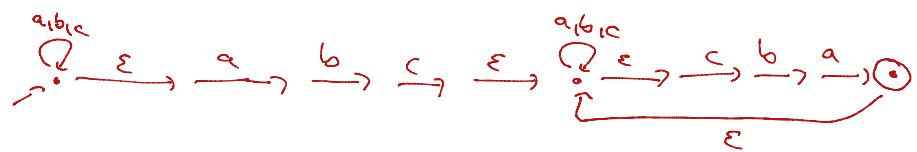
(Recall: Reductions are only allowed at the top (ie on the right in my notation) of the stack.)

You can infer the level to which you have to decrease from the rules that need to be applied in the future.

E.g. the first application of EId reduces to Exp15, which will have to be Exp14 when we come to EIncr.

Similarly, EIncr reduces to Exp13, which will have to be Exp11 when we come to EMinus.

Eliminating  $\epsilon$ -transitions of



results in

