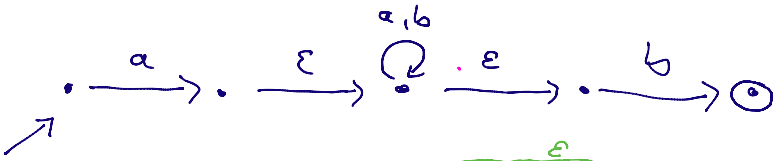
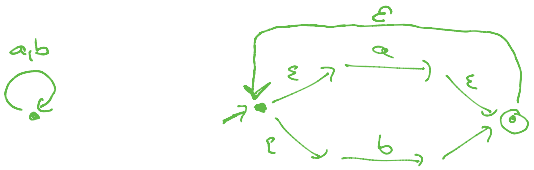
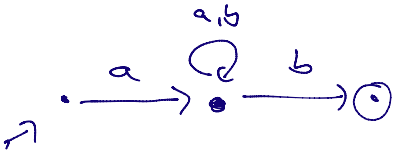
① Write down an ε-NFA for $a(a+b)^* b$



ε-NFA



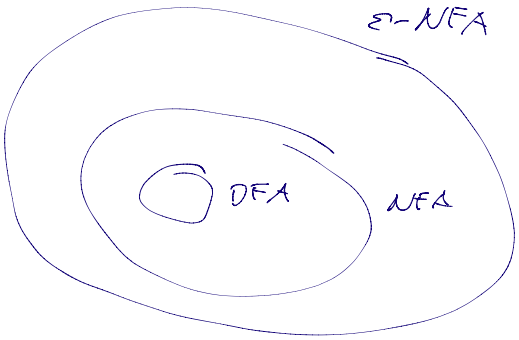
$(a+b)^*$



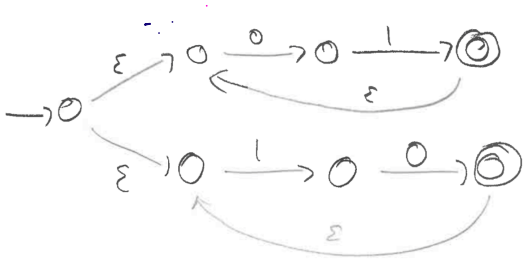
NFA, not DFA

(but also ε-NFA)

Venn Diagram of Automata:



② Consider the following ε-NFA



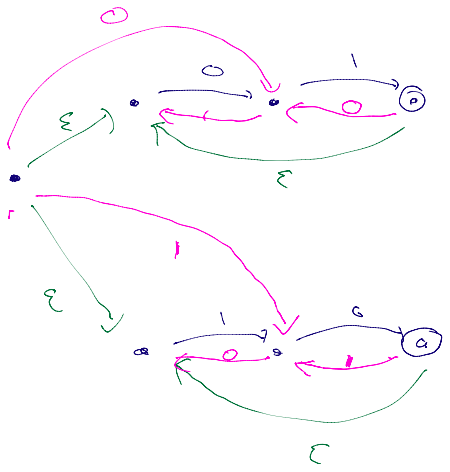
2a) $01(01)^* + 10(10)^*$

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

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

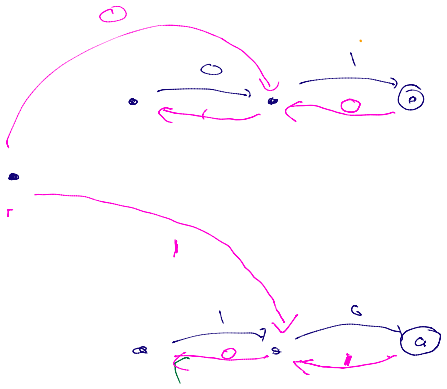
2b) Rules:

Saturate the automaton by applying the rules

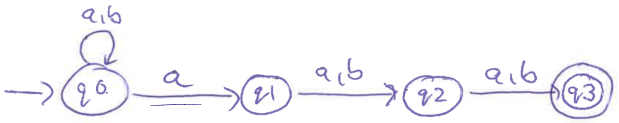


a is either 0 or 1

Eliminating
the ε-transitions



③ Write down a DFA that accepts the same language as



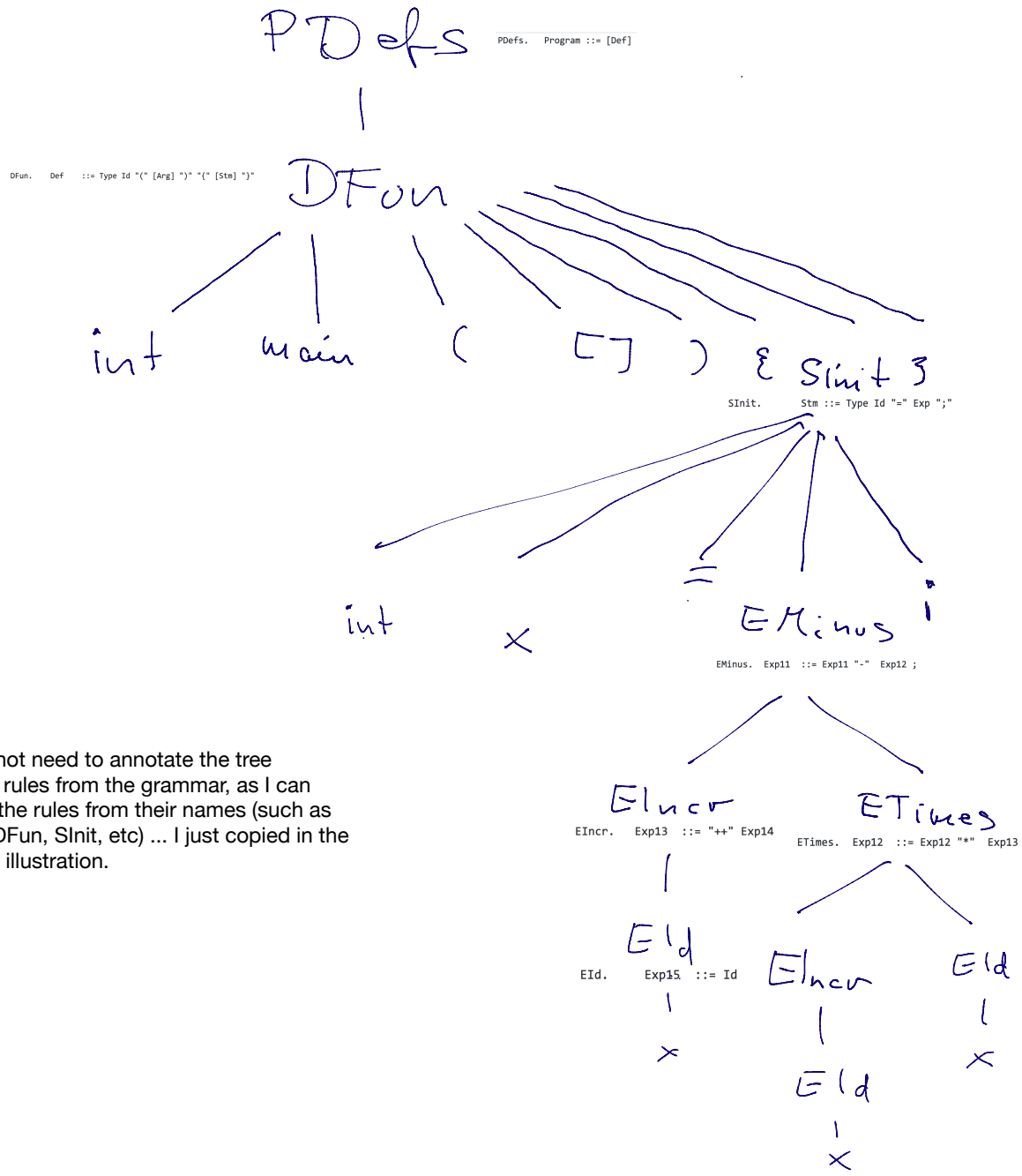
NFA

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

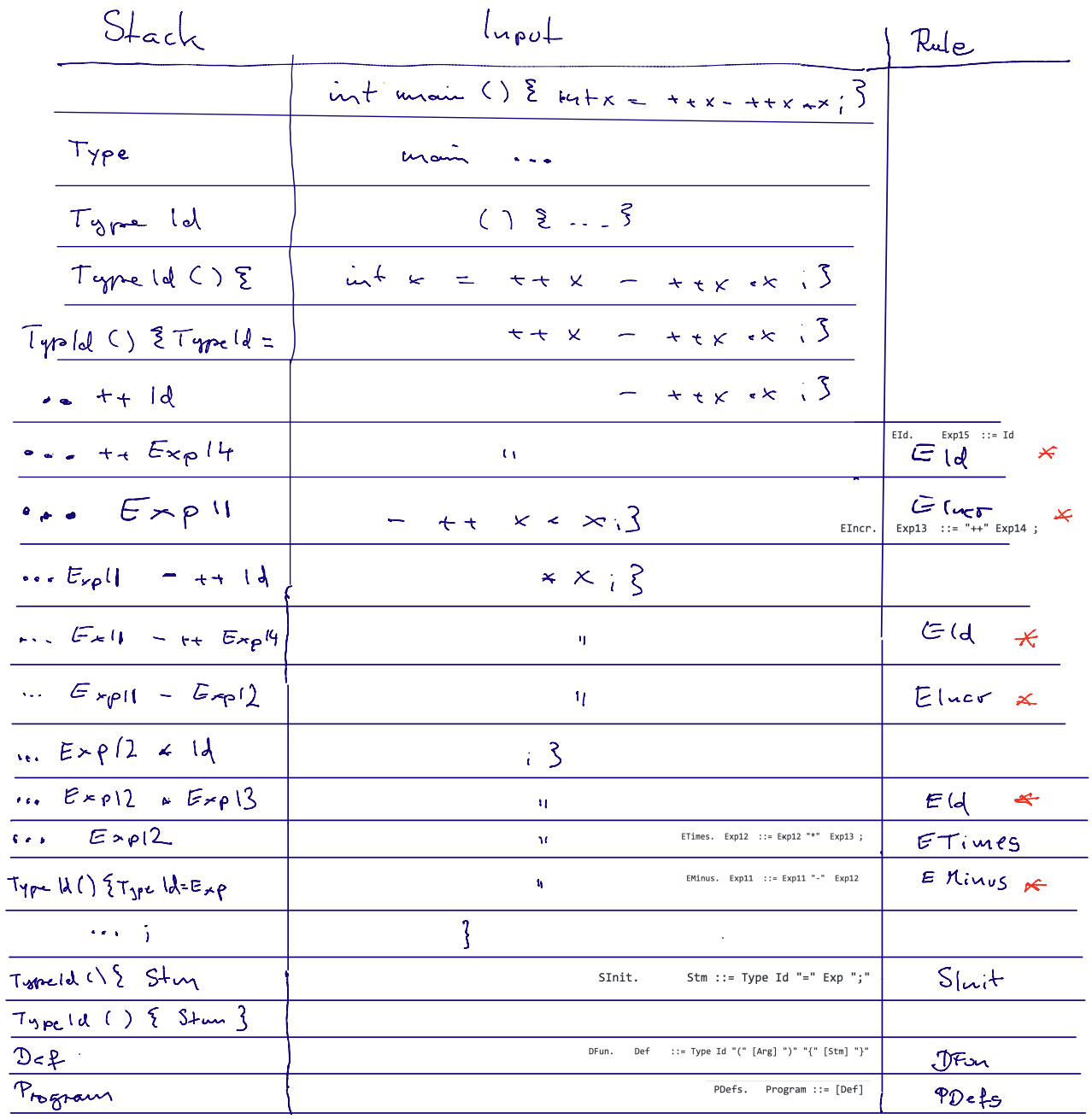
DFA

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

Abstract Syntax Tree:

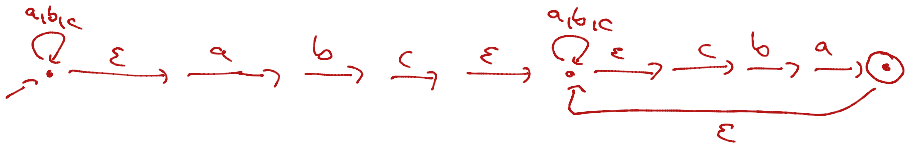


You do not need to annotate the tree with the rules from the grammar, as I can identify the rules from their names (such as PDefs, DFun, SInit, etc) ... I just copied in the rules for illustration.



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 ε-transitions of



results in