1 Exercises: Overview & Lexical Analysis

1. Do you know the following notions: Scanner, parser, compiler, interpreter, token, terminal, non-terminal BNF, EBNF, regular language, regular expression, (deterministic/non-deterministic) finite state automaton (NFA, DFA), start state, final state, production.

2. Give a grammar that describes

   (a) Floating point numbers like 123.345 or 0.54.

   (b) Floating point numbers like 123.345, 1.2e–4, and 0.3e17 that is also including possibly signed exponents.

3. Write a recognizer readA() which tries to read an A from the beginning of the input for

   (a) A = { "a" } ["b"].

   (b) A = "1" { "0" | "1" }.

   You can use ch, nextCh() and error(), as in class.

4. For the following grammars, give an equivalent grammar with one non-recursive EBNF rule:

   (a) A = "a" | A "b".

   (b) A = "a" A | A "b" | "c".

5. For the following expressions give a JLex regular expression

   (a) { "a" } ["b"]

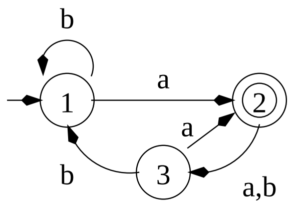
   (b) { "a" "b" }

   (c) A sequence of characters that does not contain $*$ or /.

   (d) One of the first 7 letters of the alphabet, lower or uppercase.

   (e) (hard) A sequence of characters that does not contain the subsequence ab.

6. For the following DFA:



   (a) What is the start state?

   (b) What are the final states?

(c) What sequence of states does it go through on input aabb?

(d) Does it accept aabb?

(e) What is the shortest input accepted?

7. For each of the following regular expressions give an NFA which accepts it.

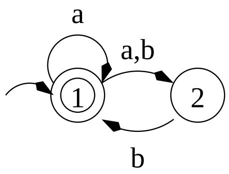
   (a) { "a" }

   (b) ["b"]

   (c) { "0" | "1" } "1"

   (d) "a" { ("b" | { "a" } "c") "x" } | { "x" } "a"

8. Convert the NFA given in the exercise before into DFA's. Convert also the following NFA into a DFA.



9. Construct a parse tree for the given expressions using the grammar

   Expr = Expr "–" Term | Term.
   Term = Term "∗" Factor | Factor.
   Factor = num | "(" Expr ")".

   (a) 3∗4–5

   (b) 3–8∗5∗(3–2)

10. (hard) Extend the grammar

   Expr = Expr "–" Term | Term.
   Term = Term "∗" Factor | Factor.
   Factor = num | "(" Expr ")".

   to allow unary – as for example in 3 – (–4) or –(3 ∗ 4).

2 Solutions

1. –

2. (a)

```
Float  = Number "." Number.
Number = Digit { Digit }.
Digit  = "0" | "1 " | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
```

(b)

```
Float  = Number "." Number [ "e" [ "–" | "+" ] Number ].
Number = Digit { Digit }.
Digit  = "0" | "1 " | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
```

3. (a)

```
void readA() {
    while (ch == 'a') {
        nextCh();
    }
    if (ch =='b') {
        nextCh();
    }
}
```

(b)

```
void readA() {
    if (ch == '1') {
        nextCh();
    } else {
        error();
    }
    while (ch =='0' || ch == '1') {
        nextCh();
    }
}
```

4. (a) A = "a" { "b" }.
   (b) A = { "a" } "c" { "b" }.

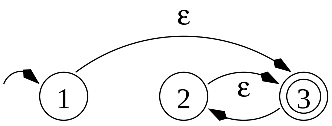
5. (a) a∗b?
   (b) (ab)∗
   (c) [^\∗\/]∗
   (d) [a–gA–G]
   (e) (a∗[^b]|[^a])∗a∗
   (Note that [^ab]∗ is a sequence containing neither a nor b.)

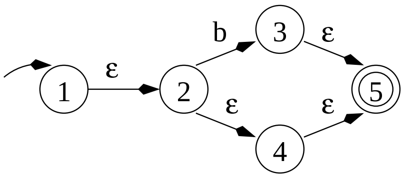
6. (a) State 1.
   (b) Only state 2.
   (c) $\langle 1, 2, 3, 1, 1\rangle$.

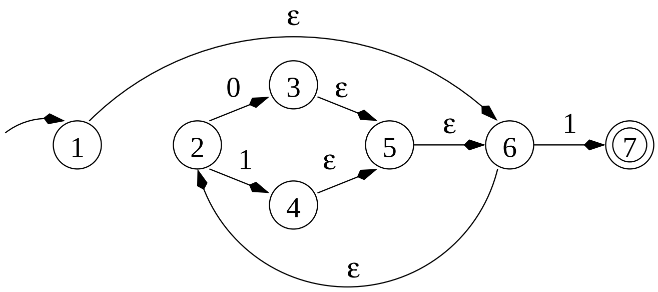
(d) No, state 1 is not a final state.

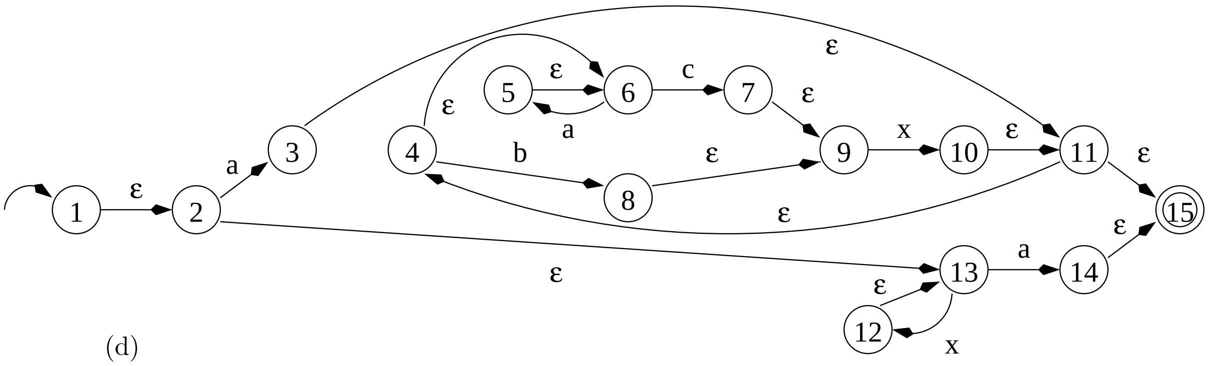
(e) a. the states are $\langle 1, 2\rangle$, and 2 is a final state.

7. The NFAs:

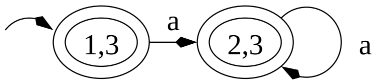
(a)


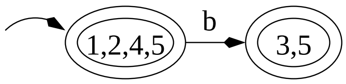
(b)


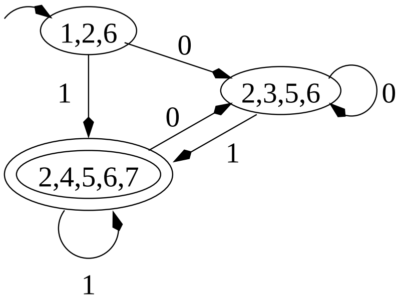
(c)


(d)


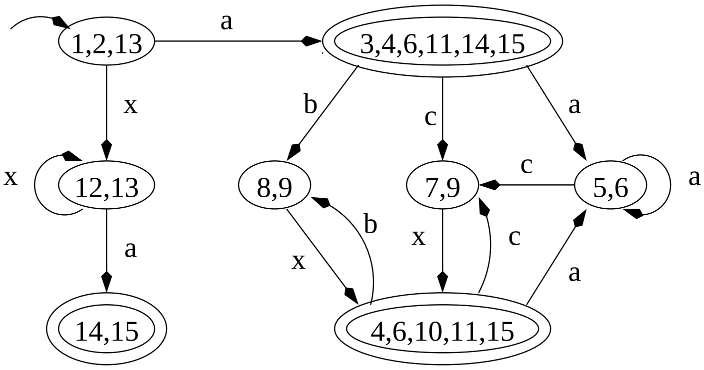
8. The DFAs:

(a)


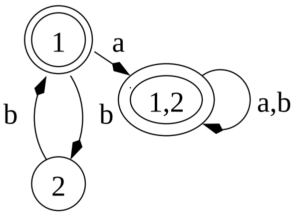
(b)




(c)

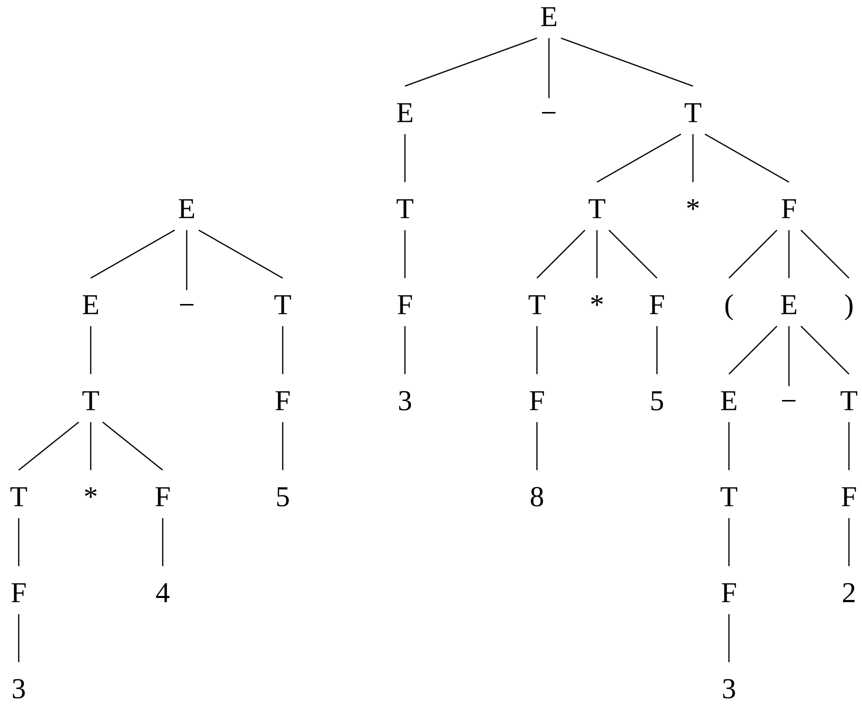


(d)



(e)

9. The parse trees:



10.

Expr	= Expr "–" Term | Term.
Term	= Term "∗" Factor | Factor.
Factor	= num | "(" Expr ")" | "–" Factor.