Compiler Construction
Problem Set 1
Alicja Jonczyk

1. Regular languages, NFAs and DFAs

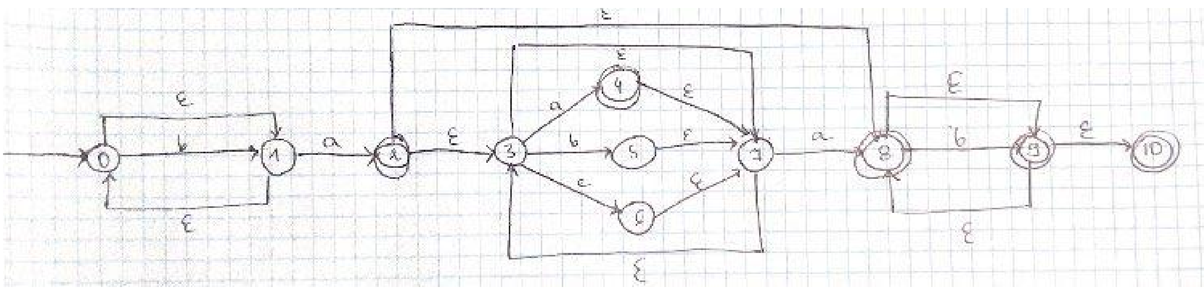
Let the formal language L be all strings over the alphabet {a, b, c}, where there is at least one a, and there are no cs before the first a, nor after the last a. Examples of strings in L include a, babb, abba, acca and babacab. Examples of strings not in L include cab, baac, cbaca and the empty string ϵ.

1.1
Show that L is a regular langauge, by writing a regular expression for it. You only need operators described in slideset 03: |, * and grouping with ( ). You may also use X? as a shorthand for (X|).

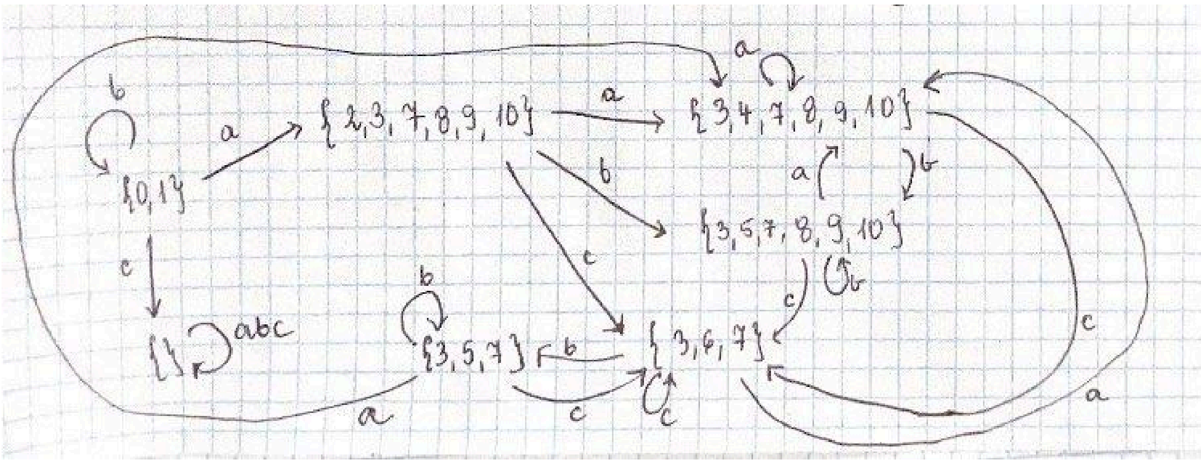
My regular expression:
b*a((a|b|c)*a|)b*

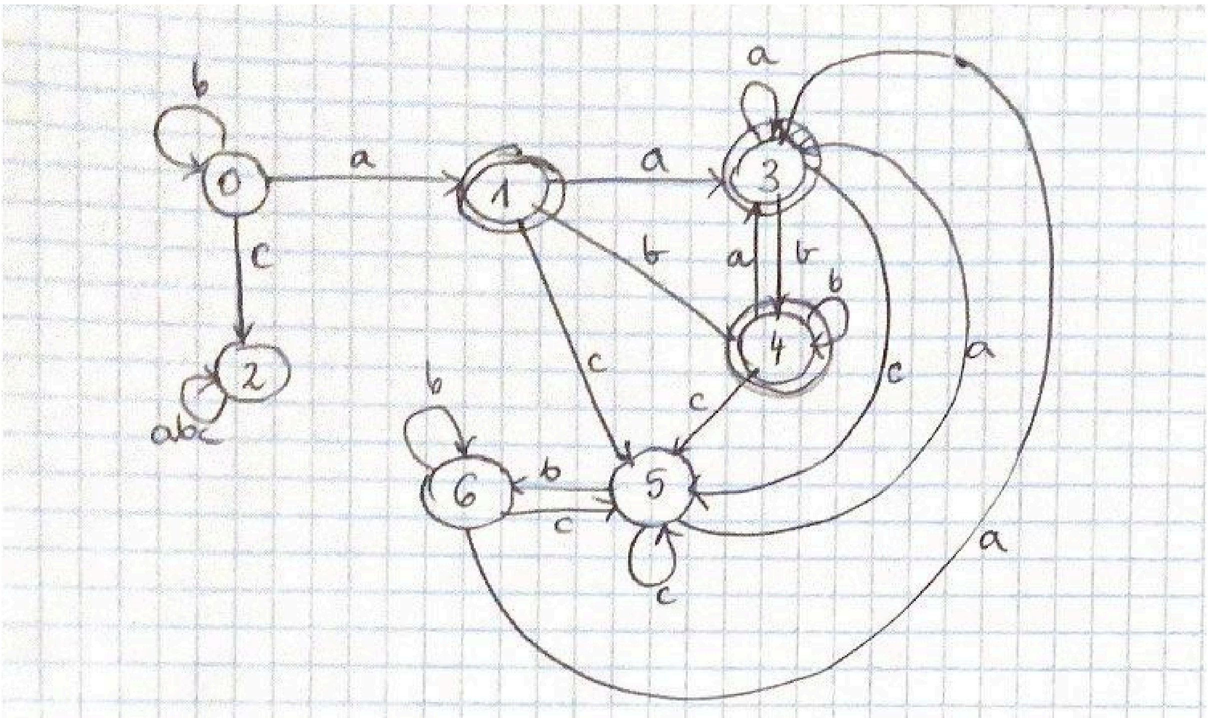
1.2
Convert the regex from 1.1 into a non-deterministic finite automata (NFA) using the McNaughton–Yamada–Thompson algorithm. Remember to number the states, indicate the starting state, and mark states as either accepting or non-accepting.

I marked the accepting states with double circles.



1.3
Convert the NFA from 1.2 into a deterministic finite automata (DFA), using the subset construction method described in slideset 04.Remember that a DFA may not contain ϵ-edges, and that every state in a DFA must have exactly one out-edge per symbol in the alphabet. If a symbol doesn't lead to any states in the NFA, you must create a "dead end" state in the DFA, and direct any lost cause inputs there. Once you have the DFA, give each DFA state a number, independent of the NFA state numbering. Again, remember to indicate the starting state, and which states are accepting.

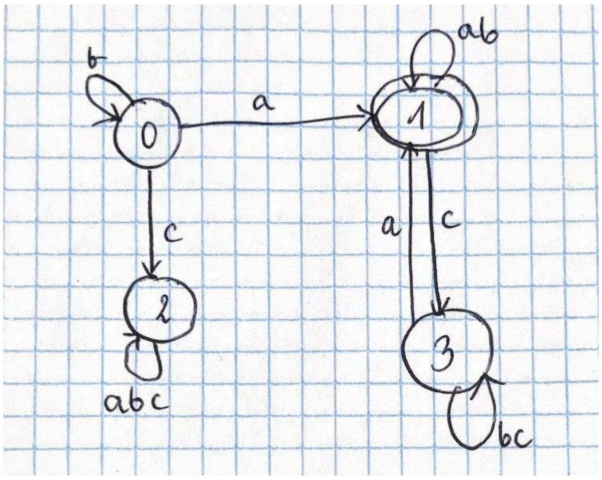




1.4
Minimizing a DFA means creating a new DFA with the minimum number of states that still matches the exact same language.

	S0	S1	S2	S3	S4	S5	S6
S0	—	—	—	—	—	—	—
S1	X	—	—	—	—	—	—
S2	X	X	—	—	—	—	—
S3	X	✓	X	—	—	—	—
S4	X	✓	X	✓	—	—	—
S5	X	X	X	X	X	—	—
S6	X	X	X	X	X	✓	—

$\delta(s0,a) = 1$ $\delta(s0,b) = 0$ $\delta(s0,c) = 2$
$\delta(s2,a) = 2$ $\delta(s2,b) = 2$ $\delta(s2,c) = 2$
$\delta(s3,a) = 3$ $\delta(s3,b) = 4$ $\delta(s3,c) = 5$
$\delta(s4,a) = 3$ $\delta(s4,b) = 4$ $\delta(s4,c) = 5$
$\delta(s5,a) = 3$ $\delta(s5,b) = 6$ $\delta(s5,c) = 5$
$\delta(s6,a) = 3$ $\delta(s6,b) = 6$ $\delta(s6,c) = 5$
$\delta(s1,a) = 3$ $\delta(s1,b) = 4$ $\delta(s1,c) = 5$



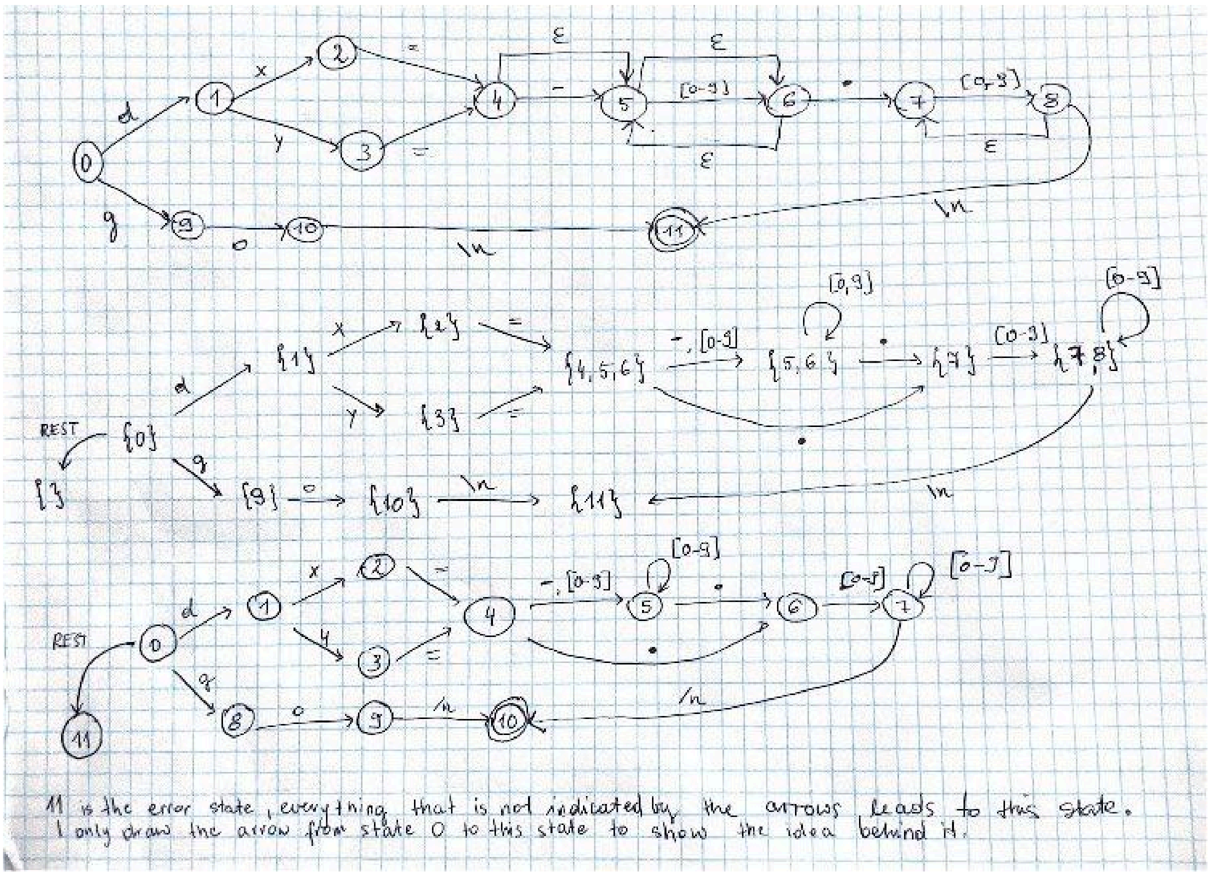
2. **DFA for a small language**

2.1

Write a regular expression matching exactly one statement, including the newline character ('\n') at the end. You can use the shorthand [0-9] to mean "any digit", and the operator X + to mean "one or more repretitions of X ".

((d(x|y)=(-|)([0-9]+|).[0-9]+)|go)\n

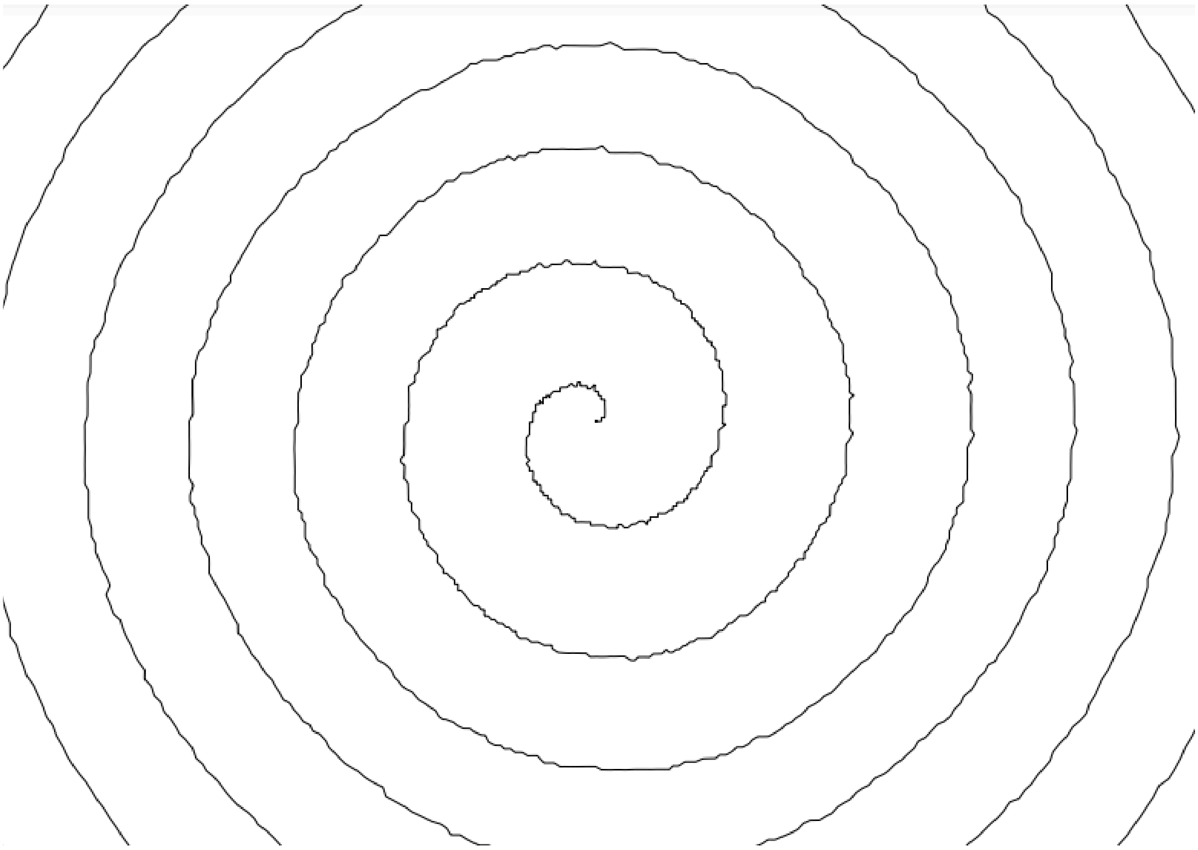
2.2



2.3

```
[100%] Built target scanner
jonczyk@alicja:~/COMPILER_CONSTRUCTION$ cat spiral.txt | ./build/scanner | ps2pdf - spiral.pdf
error: 6038: unrecognized statement: dy=-2.55.
jonczyk@alicja:~/COMPILER_CONSTRUCTION$
```

After fixing this error by removing the dot at the end of the number, my program generated a spiral.



```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>

// TODO: Update to your state values
#define N_STATES 12
#define START_STATE 0
#define ACCEPT 10
#define ERROR 11
```

```c
#define DASH 45  // '-'
#define DOT 46   // '.'
#define DIGIT_BEG 48  // '0'
#define DIGIT_END 57  // '9'
#define NL 10  // /n

#define D 'd'
#define X 'x'
#define Y 'y'
#define G 'g'
#define O 'o'
#define EQUAL '='

int transition_table[N_STATES][256]; // Table form of the automaton

void initialize_transition_table() {
    // TODO: Fill the transition table with values

    // all values set to error at the beginning
    for (int i = 0; i < N_STATES; i++) {
        for (int j = 0; j<256; j++){
            transition_table[i][j] = ERROR;
        }
    }

    // assigning what symbol will cause a given state to change into
another state
    //0
    transition_table[0][D] = 1;
    transition_table[0][G] = 8;
    //1
    transition_table[1][X] = 2;
    transition_table[1][Y] = 3;
    //2
    transition_table[2][EQUAL] = 4;
    //3
    transition_table[3][EQUAL] = 4;
    //4
    transition_table[4][DASH] = 5;
    transition_table[4][DOT] = 6;
    for (int i = DIGIT_BEG; i < DIGIT_END+1; i++)
        transition_table[4][i] = 5;
    //5
```

```c
    transition_table[5][DOT] = 6;
    for (int i = DIGIT_BEG; i< DIGIT_END+1; i++)
        transition_table[5][i] = 5;
    //6
    for (int i = DIGIT_BEG; i< DIGIT_END+1; i++)
        transition_table[6][i] = 7;
    //7
    transition_table[7][NL] = ACCEPT;
    for (int i = DIGIT_BEG; i< DIGIT_END+1;i++)
        transition_table[7][i] = 7;
    //8
    transition_table[8][O] = 9;
    //9
    transition_table[9][NL] = ACCEPT;
    //10
    // 10 is the ACCEPTING state
    //11
    // 11 is the error state in my algorithm
}

// Driver program's internal state
int state = START_STATE;
float x = 421, y = 298,     // We start at the middle of the page,
      dx = 0, dy = 0;       // and with dx=dy=0

// Used to store the chars of statement we are currently reading
char lexeme_buffer[1024];
int lexeme_length = 0;

// In here we can assume that lexeme_buffer contains a valid statement,
since the DFA reached ACCEPT
void handle_statement() {
    if (strncmp(lexeme_buffer, "go", 2) == 0) {
        x = x + dx;
        y = y + dy;
        printf( "%f %f lineto\n", x, y );
        printf( "%f %f moveto\n", x, y );
    } else if (strncmp(lexeme_buffer, "dx=", 3) == 0) {
        sscanf( lexeme_buffer+3, "%f", &dx );
    } else if (strncmp(lexeme_buffer, "dy=", 3) == 0) {
        sscanf( lexeme_buffer+3, "%f", &dy );
    } else {
        assert(0 && "Reached an unreachable branch!");
```

```c
    }
}

int main() {
    // Setup the DFA transitions as a table
    initialize_transition_table();

    // PostScript preable to create a valid ps-file
    printf ( "<< /PageSize [842 595] >> setpagedevice\n" );
    printf ( "%f %f moveto\n", x, y );

    // Main loop
    int line_num = 1; // Used to report which line an error occured on
    int read;
    while( (read = getchar()) != EOF) {
        // Store the read char in the buffer
        lexeme_buffer[lexeme_length++] = read;
        lexeme_buffer[lexeme_length] = 0; // Add NULL terminator

        // Use the current state and the read char to find the next
state
        state = transition_table[state][read];

        // Check if we reached the ACCEPT or ERROR states
        switch (state) {
            case ACCEPT:
                handle_statement();
                state = START_STATE;
                lexeme_length = 0;
                break;
            case ERROR:
                fprintf(stderr, "error: %d: unrecognized statement:
%s\n", line_num, lexeme_buffer);
                exit( EXIT_FAILURE );
            default: break;
        }

        // If the char was a newline, the next char will be on a new
line!
        if (read == '\n')
            line_num++;
    }
```

```
    if (state != START_STATE) {
        fprintf(stderr, "error: %d: input ended in the middle of a statement: %s\n", line_num, lexeme_buffer);
        exit( EXIT_FAILURE );
    }

    printf ( "stroke\n" );
    printf ( "showpage\n" );
}
```