Homework #4 Solution

For this homework assignment, attach a PDF file containing all answers that involve written text or diagrams. You can provide an electronic version, or a scanned handwritten version.

Question 1 (5 pt.)

Consider the following regular expression matching positive or negative decimal numbers with an optional fraction:

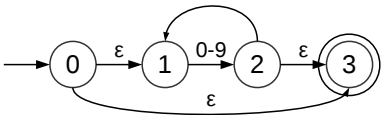
```
[+-]? [0-9]+ (. [0-9]+ )?
```

a) Transform the regular expression to another regular expression that uses only concatenation, union, and Kleene closure operations.

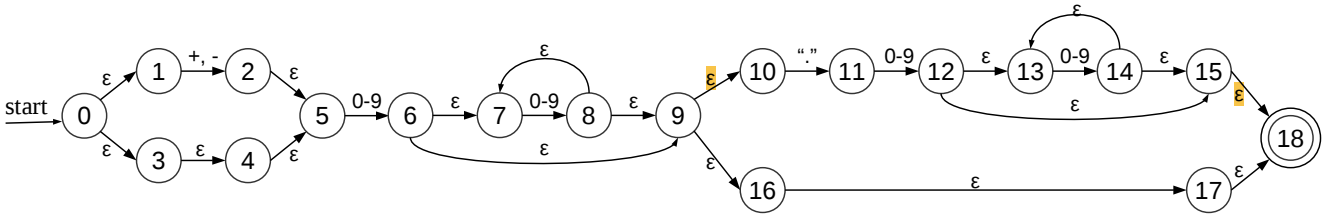
```
([+-] | Ɛ ) [0-9][0-9]* (. [0-9][0-9]* | Ɛ )
```

b) Construct an NFA using the McNaughton-Yamada-Thompson algorithm that accepts the same language as the resulting regular expression, and provide its transition graph. Do not skip any steps of the algorithm, even if they insert redundant states. You can use multiple labels (or label ranges) on state transitions associated with character classes.

Here it is useful to start creating an NFA for sub-expression [0-9]*, and then introduce it in the larger NFA every time it's needed:



The final NFA is:



c) Write a C++ main program that takes a string as an argument and prints a message indicating whether the string matches the regular expression above. Use the implementation of an NFA presented in class, and given in the support material. You can use class `NFA` by including file `NFA.h` in your main program. Attach only the main program in a file named `q1.cc`. This file should compile without modifications, and run correctly on the CoE machines if linked together with file `NFA.cc`.

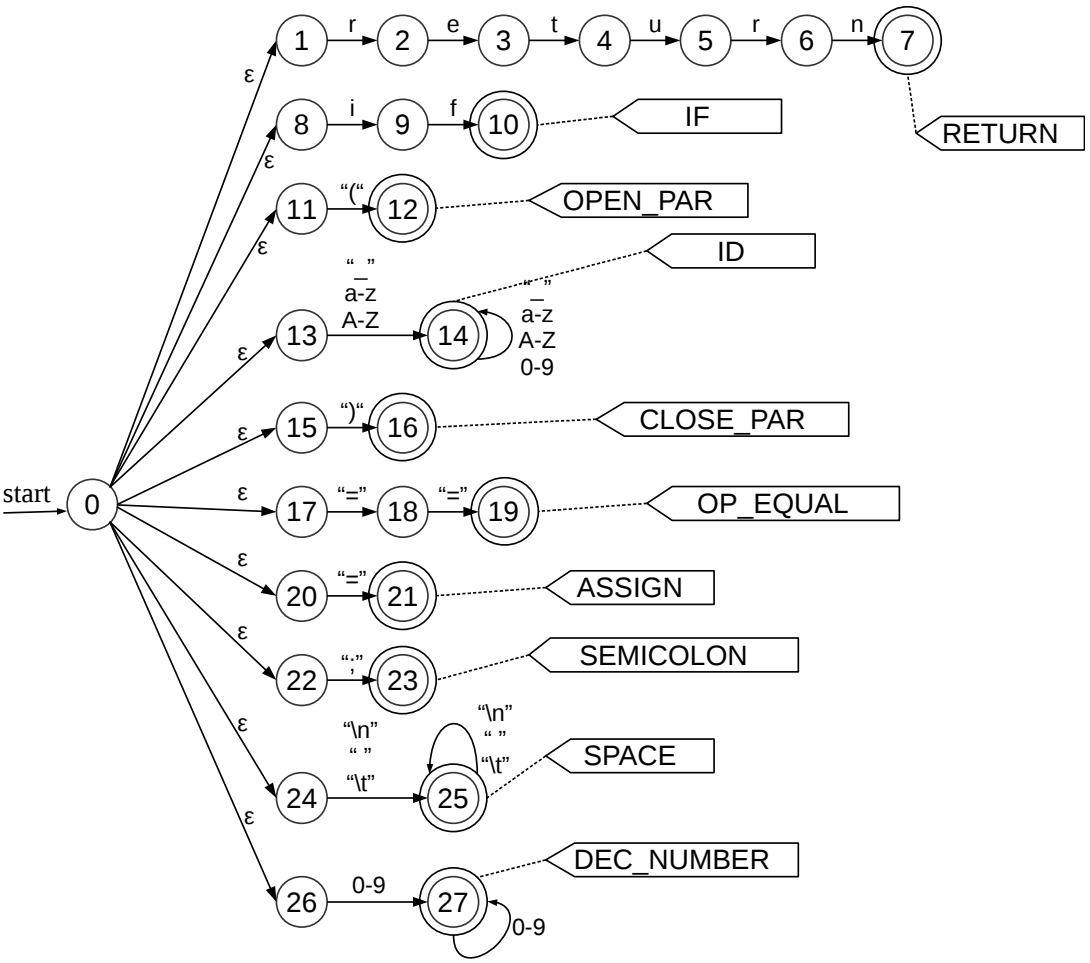
```
#include <iostream>
#include "NFA.h"

int main(int argc, char **argv)
{
	// Syntax
	if (argc != 2)
	{
		std::cerr << "Syntax: ./main <string>\n";
		return 1;
	}

	NFA nfa;
	nfa.setFinalState(18);
	nfa.addTransition(0, 0, 1);
	nfa.addTransition(0, 0, 3);
	nfa.addTransition(1, '+', 2);
	nfa.addTransition(1, '-', 2);
	nfa.addTransition(2, 0, 5);
	nfa.addTransition(3, 0, 4);
	nfa.addTransition(4, 0, 5);
	for (char c = '0'; c <= '9'; c++)
		nfa.addTransition(5, c, 6);
	nfa.addTransition(6, 0, 7);
	nfa.addTransition(6, 0, 9);
	for (char c = '0'; c <= '9'; c++)
		nfa.addTransition(7, c, 8);
	nfa.addTransition(8, 0, 7);
	nfa.addTransition(8, 0, 9);
	nfa.addTransition(9, 0, 10);
	nfa.addTransition(9, 0, 16);
	nfa.addTransition(10, '.', 11);
	for (char c = '0'; c <= '9'; c++)
		nfa.addTransition(11, c, 12);
	nfa.addTransition(12, 0, 13);
	nfa.addTransition(12, 0, 15);
	for (char c = '0'; c <= '9'; c++)
		nfa.addTransition(13, c, 14);
	nfa.addTransition(14, 0, 13);
	nfa.addTransition(14, 0, 15);
	nfa.addTransition(15, 0, 18);
	nfa.addTransition(16, 0, 17);
	nfa.addTransition(17, 0, 18);

	// Parse string
	bool accepted = nfa.accepts(argv[1]);
	std::cout << "String " << (accepted ? "accepted" : "rejected") << "\n";
	return 0;
}
```

Question 2 (5 pt.)

Consider the following code:

```
my_2nd_var = 2;
if (x)
    return a == 1;      ( ← TAB character before “return”)
```

a) Give the transition graph for a complete scanner NFA that recognizes all tokens present in this code. Add a label to each final state indicating which token it recognizes.



b) Store this code in a file named `input.txt`. Write a C++ main program that opens this file with an `std::ifstream` object, passes it directly to the constructor of `InputBuffer`, and scans its content with the NFA obtained before. The output of the program should be a list with the identifier of each token, followed by the associated lexeme. Attach only the main program in a file named `q2.cc`, which should compile without modifications on the CoE machines when linked together with `InputBuffer.cc` and `NFA.cc`.

```
#include <fstream>
#include <iostream>

#include "InputBuffer.h"
#include "NFA.h"

const int TokenReturn = 1;
const int TokenIf = 2;
const int TokenOpenPar = 3;
const int TokenId = 4;
const int TokenClosePar = 5;
const int TokenOpEqual = 6;
const int TokenAssign = 7;
const int TokenSemicolon = 8;
const int TokenSpace = 9;
const int TokenDecConst = 10;

int main()
{
      NFA nfa;

      // Final states
      nfa.setFinalState(7, TokenReturn);
      nfa.setFinalState(10, TokenIf);
      nfa.setFinalState(12, TokenOpenPar);
      nfa.setFinalState(14, TokenId);
      nfa.setFinalState(16, TokenClosePar);
      nfa.setFinalState(19, TokenOpEqual);
      nfa.setFinalState(21, TokenAssign);
      nfa.setFinalState(23, TokenSemicolon);
      nfa.setFinalState(25, TokenSpace);
      nfa.setFinalState(27, TokenDecConst);

      // Transitions from state 0
      nfa.addTransition(0, 0, 1);
      nfa.addTransition(0, 0, 8);
      nfa.addTransition(0, 0, 11);
      nfa.addTransition(0, 0, 13);
      nfa.addTransition(0, 0, 15);
      nfa.addTransition(0, 0, 17);
      nfa.addTransition(0, 0, 20);
      nfa.addTransition(0, 0, 22);
      nfa.addTransition(0, 0, 24);
      nfa.addTransition(0, 0, 26);

      // Transitions for TokenReturn
      nfa.addTransition(1, 'r', 2);
      nfa.addTransition(2, 'e', 3);
```

```
nfa.addTransition(3, 't', 4);
nfa.addTransition(4, 'u', 5);
nfa.addTransition(5, 'r', 6);
nfa.addTransition(6, 'n', 7);

// Transitions for TokenIf
nfa.addTransition(8, 'i', 9);
nfa.addTransition(9, 'f', 10);

// Transitions for TokenOpenPar
nfa.addTransition(11, '(', 12);

// Transitions for TokenId
nfa.addTransition(13, '_', 14);
for (char c = 'a'; c <= 'z'; c++)
      nfa.addTransition(13, c, 14);
for (char c = 'A'; c <= 'Z'; c++)
      nfa.addTransition(13, c, 14);
nfa.addTransition(14, '_', 14);
for (char c = 'a'; c <= 'z'; c++)
      nfa.addTransition(14, c, 14);
for (char c = 'A'; c <= 'Z'; c++)
      nfa.addTransition(14, c, 14);
for (char c = '0'; c <= '9'; c++)
      nfa.addTransition(14, c, 14);

// Transitions for TokenClosePar
nfa.addTransition(15, ')', 16);

// Transitions for TokenOpEqual
nfa.addTransition(17, '=', 18);
nfa.addTransition(18, '=', 19);

// Transitions for TokenAssign
nfa.addTransition(20, '=', 21);

// Transition for TokenSemicolon
nfa.addTransition(22, ';', 23);

// Transitions for TokenSpace
nfa.addTransition(24, ' ', 25);
nfa.addTransition(24, '\t', 25);
nfa.addTransition(24, '\n', 25);
nfa.addTransition(25, ' ', 25);
nfa.addTransition(25, '\t', 25);
nfa.addTransition(25, '\n', 25);

// Transitions for TokenDecConst
for (char c = '0'; c <= '9'; c++)
      nfa.addTransition(26, c, 27);
for (char c = '0'; c <= '9'; c++)
      nfa.addTransition(27, c, 27);

// Open file
std::ifstream f("input.txt");

// Create input buffer
InputBuffer input_buffer(f);
while (true)
```

```
	{
		std::pair<int, std::string> pair = input_buffer.getToken(nfa);
		if (!pair.first)
			break;

		// Token found
		std::cout << "Token " << pair.first << ": '" << pair.second << "'\n";
	}
	return 0;
}
```