Assignment 7: Regular expressions and finite state automata

The assignment

When creating automata, it is often useful to verify them visually. To this end, the *NFA* class has a *toDot()* method. Its output can be pasted on this website (http://sandbox.kidstrythisathome.com/erdos/) or using Graphviz to visualize the automaton.

7.1 Extend the expression language

The project contains a simple 'language' that allows you to create regular expressions in Java. Here are two sample expressions:

```
// (ab|bc)+
Expression ab = Expr.concat(Expr.ch('a'), Expr.ch('b'));
Expression bc = Expr.concat(Expr.ch('b'), Expr.ch('c'));
Expression expr = Expr.plus(Expr.or(ab, bc));

// ((ab|ac)d+)*
Expression ac = Expr.concat(Expr.ch('a'), Expr.ch('c'));
expr = Expr.star(Expr.concat(Expr.or(ab, ac), Expr.plus(Expr.ch('d'))));
```

Read the classes in the *expr* package to get an understanding of how this works. Then, implement the following:

- `Expr.str(String s)` , which should create an expression for recognizing the string that is the argument. This method should be implemented using `Expr.chr(char c)` .

Using this method, you can simplify the fragments above to:

```
// (ab|bc)+
Expression expr = Expr.plus(Expr.or(Expr.str("ab"), Expr.str("bc")));

// ((ab|ac)d+)*
expr = Expr.star(Expr.concat(Expr.or(Expr.str("ab"), Expr.str("ac")),
  Expr.plus(Expr.ch('d'))));
```

7.2 Implement common operations on automata

The *Expression* interface defines a method `NFA compile()` , which converts an expression into a non-deterministic finite state automaton (NFA). The following methods in the `NFAOperations` class are used for this conversion:

```
public static NFA character(char c);
public static NFA concatenate(NFA nfa, NFA nfa2);
public static NFA union(NFA nfa, NFA nfa2);
public static NFA kleeneStar(NFA nfa);
```

- Implement the last three methods.

Note 1: You can check your implementation by using the expression language of the previous assignment and printing the automaton in dot format (see link above). For example:

```
Expression expr = Expr.plus(Expr.or(Expr.str("ab"), Expr.str("bc")));
NFA nfa = expr.compile();

// Paste the output of this line to the website above to inspect the
// automaton.
System.out.println(nfa.toDot());
```

Note 2: If you are modifying existing states in one of the NFAs, you might want to make a copy of the complete NFA. You can use `NFA#clone()` to do so.

7.3 Implement NFA recognition

Implement the method

```
public boolean recognize(String string);
```

in `NFA` . This method should return `true` if the string is in the language of the automaton and `false` otherwise. (Depth-first or breadth-first, it's up to you!)

8.1 Implement the subset construction

Implement the subset construction method to convert a non-deterministic automaton to a deterministic automaton. The `NFAOperations` class provides a start in the following pair of methods:

```
public static DFA determinize(NFA nfa);
private static DFAState determinize(Set<NFAState> stateSet,
  Map<Set<NFAState>, DFAState> stateMapping);
```

`stateMapping` can be used to keep track of which set of states from the NFA corresponds to which state in the DFA. The `DFA` and `DFAState` classes are provided in the project.

8.2 Implement minimization *(Bonus)*

This is a bonus assignment for 10 additional points (the final grade will be *min(100,points)*). As such, submission of assignment 8.2 is not required.

Implement one of the following two minimization algorithms:

1. Hopcroft minimization
2. Brzozowski minimization

If you choose to implement (2), it works best to implement the automaton reversal as a separate operation, which also allows you to test reversal separately.

Submission

Submit the program as follows:

1. Run `mvn clean` in the main directory of the project to remove the compiled classes.
2. The implementation of 7.1, 7.2, 7.3, and 8.1 should be part of the submission. 8.2 is optional.
3. Compress the complete project directory to a `tar.gz` or `zip` archive.