- You know the drill now: Find students around you to form a ***small group***; use ***all resources*** to help to solve the problems; ***discuss*** your idea with other group member and ***write down*** your own solutions; raise your hand and pull the ***course staffs*** to help; ***submit*** your writeup through Gradescope in *24 hours*.

Our topic for this working session is *regular expression for complements*.

Using subset construction and Kleene's theorem (which resembles the Floyd-Warshall APSP algorithm), theoretically we can construct a regular expression for the complement of any regular language. Given a regular expression E for regular language L, perform the following:

- First, turn E into an equivalent NFA N.
- Next, use subset construction to turn N into DFA M.
- Flip the accepting/non-accepting states of M.
- Finally, turn the DFA M back into a regular expression using Kleene's theorem.

But in practice, how do we deal with the potential double-exponential blowup in size of the regular expression?

We will always use *incremental construction* shown in class in place for the subset construction; often times the resulting DFA is smaller. However, there is no guarantee that the constructed DFA is the *smallest possible*.

There are several algorithms that allow one to minimize a DFA (for the same regular langauge); the best algorithm runs in near-linear time but is quite complicated. Here instead we will introduce a simple algorithm, by Brzozowski:

```
BRZOZOWSKI-MINIMIZATION(N):
  input: NFA N recoginizing language L
  reverse N and obtain N^R, recognizing rev(L)
  turn N^R into DFA M^R
  reverse M^R and obtain N', recognizing L
  turn N' into DFA M
  return M
```

In other words, Brzozowski's algorithm just perform reversal of the NFA follow by subset constrction, twice in a row. We will show in tomorrow's lecture that the final DFA M must be smallest in size for language L. How convenient! Alternatively you may use Moore's algorithm as described in Erickson's notes, Chapter 3.10, but the reading will become easier again after tomorrow since we will be talking about some fine-grainded structure of the DFAs.

Construct a regular expression that represent the *complement* of the language for the following regular expressions. In other words, if the following expression represents language L, then the expression you construct should represent the language $\Sigma^* \setminus L$.

1. $\Sigma^*\texttt{000}\Sigma^*$ (words that contain `000` as substring)
2. $\texttt{0}^*(\texttt{1} + \texttt{01}^*\texttt{0})^*$ (words that are concatenation of `0`s and a string with even number of `0`s)

To think about later: (No submissions needed)

3. $\Sigma^*\texttt{0}\Sigma^n\texttt{0}\Sigma^*$ (words that contain two `0`s that are exactly n places apart)

Conceptual question: Is there an example where the regular expression for the complement of the language $\Sigma^* \setminus L$ is in fact doubly-exponential in length of the regular expression for L?