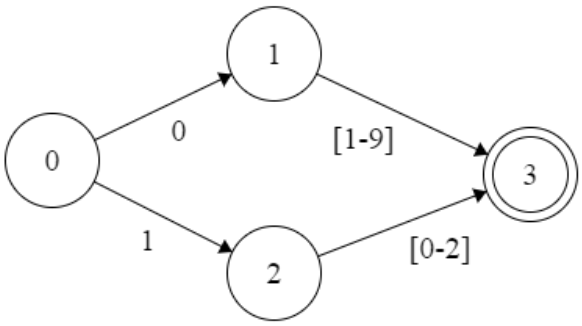
COMP 150-06
Problem Set 1

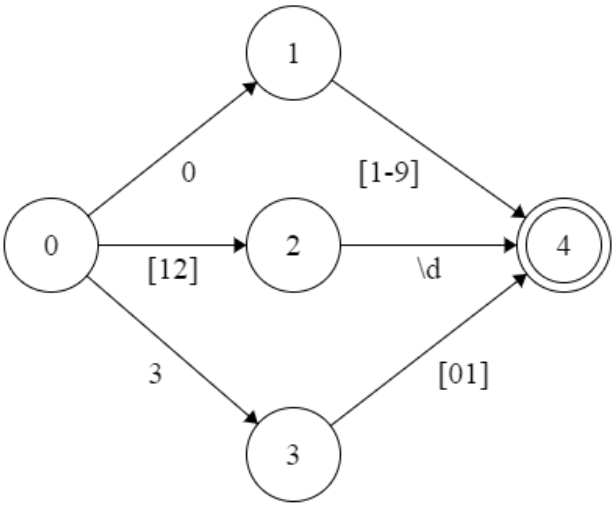
Weitong Ruan

1. Regular expressions:
    1) MM component: /0[1-9] | 1[0-2]/

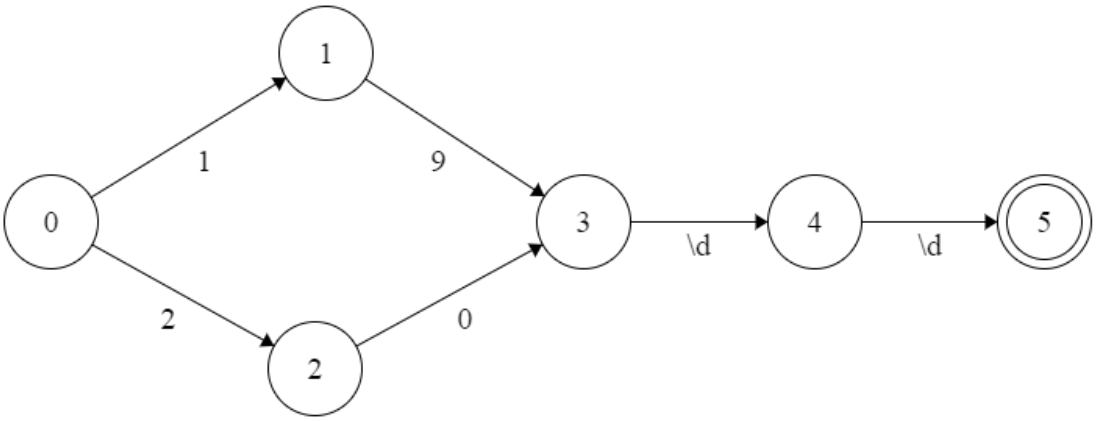
       The first digit can be either 0 or 1, if 0, the second digit can be anything from [1-9], if 1, the second can only be from [0-2]. The idea applies to the rest.
    2) DD component : /0[1-9] | [12] \d | 3[01]/
    3) YYYY component: /19 | 20 \d{2}/
    4) Separator component: /[/-]/
2. Draw FSA:
    1) MM:



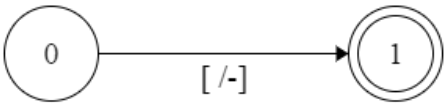
    2) DD:



    3) YYYY:



4) Separators:



3. I follow the prototype when implementing the FSA class with an extra function called “AddTransList” which adds a list of transitions. Basically, transitions is implemented using a dictionary with keys being a pair of current state and input symbol ((input_symbol, current_state)) and the corresponding value is the next state.
4. This part basically follows the algorithm in the book.
5. The main idea for this algorithm is loop over each input FSAs, then for each FSA, check if the input passes the FSA. The first issue becomes how to tell the inputs for each FSA. Basically, what I did was to use a index to mark where we are on the input tape and by checking the FSA that we are currently using to decide how many input symbols we are going to use for checking. For example, if the FSA is months, we take the next two symbols, or if the FSA is separators, we take only the next symbol. Then I have a flag which I call “temp” in the codes. The default is true, then we update it after each FSA test, if it goes to false, then the algorithm stops and return false. Otherwise, it keeps going until we reach the end of the input and the end of FSA list.
6. To implement this, I take the first way, implement NDRecognize and NDRecognizeMulti. The difference between deterministic and nondeterministic are that, first, we add an “epsilon” transition when checking MM and DD and second, we need to use a stack for backtracking. For both MM and DD components, what I did is add an “epsilon” transition from 0 state to 1 in the above graph. In NDRecognizeMulti, compared with DRecognizeMulti, the changes lie in the checking of MM and DD components. For single input digits like “5”, we can’t take the first two input symbols as we did in DRecognizeMulti, hence this case has to be separated from

the main case. For input with multiple symbols, this one digit also needs to be dealt with, what I did is first take two symbols, check wether the second symbol is a number, if so, then the two symbol combined represent a MM or DD component, if not, we only take the first symbol.