PROBLEM SET #1

MARIA MOYA

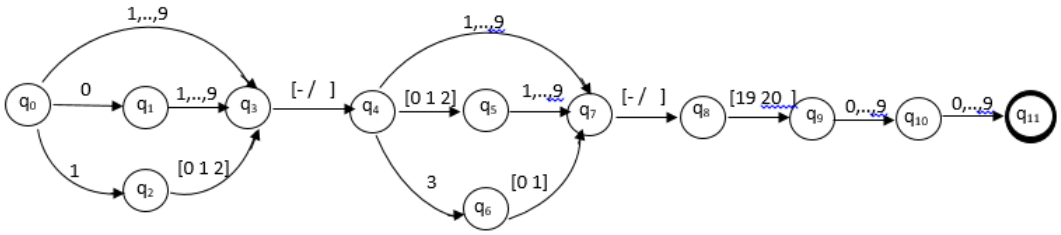
1. Create regular expressions for each component of the date expression

Consider the following regular expression:

$$\wedge([1-9]|0[1-9]|1[012])[-/]([1-9]|0[1-9]||[12][0-9]|3[01])[-/](19|20)\setminus d\setminus d\$$$

The expression is anchored by $\wedge$ and \$. Note that for months, $([1-9]|0[1-9]|1[012])$ our first component(before the pipeline) accounts for months $1-9$. Our second component denotes months $01-09$ and the last portion,1[012], represents months 10, 11 and 12. Each component is separated by a a pipeline which is equivalent to saying months "1-9 or 01-09 or 10-12". The $[-/]$ represents our separators, - / as well as spaces. In regards to the days,$([1-9]|0[1-9]||[12][0-9]|3[01])$, the first two components are the same as the month category. However, the third component, $[12][0-9]$ takes days 10-19 or 20-29. The fourth component, 3[01] accounts for days 30 and 31. Finally for years, this takes $19\setminus d\setminus d$ or $20\setminus d\setminus d$ implies years 1900-1999 or 2000-2099.

2. Draw individual FSA corresponding to each component



The FSA starts at q_0. From there it moves to state q_3 if the month is denoted as 1-9. In the event that the month starts with a zero, we transition to state 3. If the month begins with a 1 (hence 10,11,12) we transition to q_2 then back to q_3. Afterwards we want to account for the seperator, thus we transition to q_4. If the day starts without a zero in the beginning of it and it's from 1-9 we transition over to state 7. If the day is from 01-29 we go to state 5 and then back to q_7. For days 30 and 31 we go to state 6 to 7. We want to account for our second separator and transition onward to state 8. As far as our years, they will start with 19 or 20. Thus we move from state 8 to 9. Then we want to consider the last two digits of the year so we go to state 10 for the

third digit and state 11 for last digit. Finally, state 11 denotes the final state in which our FSA accepts a string of the form mm/dd/yyyy , mm-dd-yyyy or mm dd yyyy. All other strings that do not satisfy that format or get stuck mid point are rejected.

3. Build a nondeterministic recognizer for each component

For the following problem, I used a test function that takes the string and then finds out if it is acceptable by the constructed FSA. This generates the following:

Test Months FSA	Test Days FSA	Test Years FSA	Test Separators FSA
'' False	'' False	'' False	'' False
'0' False	'0' False	'1899' False	',' False
'1' True	'1' True	'1900' True	' ' True
'9' True	'9' True	'1901' True	'-' True
'10' True	'10' True	'1999' True	'/' True
'11' True	'11' True	'2000' True	'//' False
'12' True	'21' True	'2001' True	':' False
'13' False	'31' True	'2099' True	
	'32' False	'2100' False	

I had to play around with different transitions and final states in order to satisfy my FSA which can be found in lines 93 to 122. For each component, I start at an initial state 0. For the months, days and years our alphabet, $\sum = \{0, 1, ..., 9\}$. For the separators we have an alphabet of \ , – and a space.

4. Build a nondeterministic recognizer for entire dates

Lastly, in concatenation, we concatenate 2 constituent FSA to give new FSA. The self.constituent[] holds the FSA that constitute the new FSA, so that it is possible to check, if the concaenated string satisfies the new FSA. This generates the following output:

```
Test Date Expressions FSA
''      False
'12 31 2000'    True
'12/31/2000'    True
'12-31-2000'    True
'12:31:2000'    False
'1 2 2000'      True
'1/2/2000'      True
'1-2-2000'      True
'1:2:2000'      False
'00-31-2000'    False
'12-00-2000'    False
'12-31-0000'    False
'12-32-1987'    False
'13-31-1987'    False
'12-31-2150'    False
'1-2-1988'      True
```