Compilers

Spring 2022

Lexical Analysis

Sample Exercises and Solutions

Prof. Pedro C. Diniz

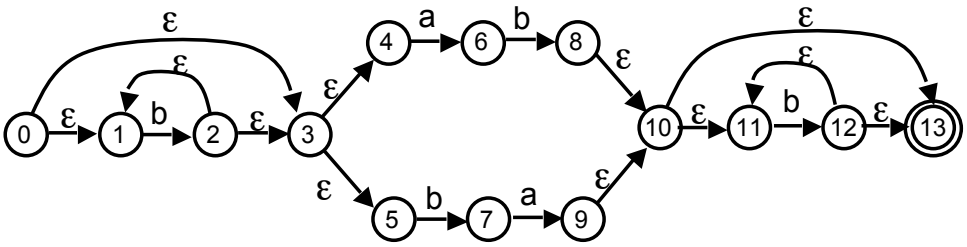
Faculdade de Engenharia da Universidade do Porto
Departamento de Engenharia Informática
pedrodiniz@fe.up.pt

Problem 1

Considering the alphabet Σ = {a,b} derive a Non-Deterministic-Finite Automaton (NFA) using the Thompson construction that is able to recognize the sentences generated by the regular expression b*(ab|ba)b*. Does the sentence w = "abbb" belong to the language generated by this regular expression? Justify.

Solution:

Considering a simplification of the combination of the NFA during the Thompson construction (e.g., sequence of two ε-transitions can be considered as a single ε-transition) a possible NFA is as shown below and where the start state is the state labeled 0.



The word w = "abbb" belongs to the language generated by this RE because there is a path from the start state 0 to the accepting state 13 that spells out the word, respectively the path 0,3,4,6,8,10,11,12,11,12,13.

Problem 2

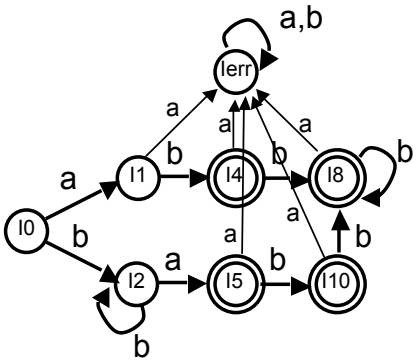
Starting from the NFA you derived in problem 1, convert it to a DFA using the sub-set construction described in class. Verify that the DFA yields the same output as the original NFA for the same input string w.

Solution:

Using the ε-closure and DFAedge computation as described in class, we have the following mapping of set of states of the NFA to sets of states of the DFA:

I0 = ε-closure (0) = {0, 1, 3, 4, 5}
I1 = DFAedge(I0,a) = ε-closure ({0, 1, 3, 4, 5}, a) = {6}
I2 = DFAedge(I0,b) = ε-closure ({0, 1, 3, 4, 5}, b) = {1, 2, 3, 4, 5, 7}
I3 = DFAedge(I1,a) = ε-closure ({6}, a) = Ierr
I4 = DFAedge(I1,b) = ε-closure ({6}, b) = {8, 10, 11, 13}
I5 = DFAedge(I2,a) = ε-closure ({1, 2, 3, 4, 5, 7}, a) = {6, 9, 10, 11, 13}
I6 = DFAedge(I2,b) = ε-closure ({1, 2, 3, 4, 5, 7}, b) = {1, 2, 3, 4, 5, 7} = I2
I7 = DFAedge(I4,a) = ε-closure ({8, 10, 11, 13}, a) = Ierr
I8 = DFAedge(I4,b) = ε-closure ({8, 10, 11, 13}, b) = {11, 12, 13}
I9 = DFAedge(I5,a) = ε-closure ({6, 9, 10, 11, 13}, a) = Ierr
I10 = DFAedge(I5,b) = ε-closure ({6, 9, 10, 11, 13}, b) = {8, 10, 11, 12, 13}
I11 = DFAedge(I8,a) = ε-closure ({11, 12, 13}, a) = Ierr
I12 = DFAedge(I8,b) = ε-closure ({11, 12, 13}, b) = {11, 12, 13} = I8
I13 = DFAedge(I10,a) = ε-closure ({8, 10, 11, 12, 13}, a) = Ierr
I14 = DFAedge(I10,b) = ε-closure ({8, 10, 11, 12, 13}, b) = I8

Effectively, there are only 8 states as shown in the DFA below and this is clearly not the minimal DFA that can recognize this regular expression.



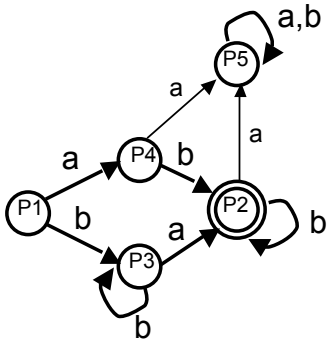
For the input sentence w = "abbb" in his DFA we would reach the state I8, through states I1, I4 and I8 and thus accepting this string.

Problem 3

Starting from the DFA you constructed in the previous problem, convert it to a minimal DFA using the minimization algorithm described in class. Verify that the DFA yields the same output as the original NFA for the same input string w.

Solution:

The first partition would have the two sets of states P1 = {I0, I1, I2, Ierr} and P2 = {I4, I5, I8, I10}. A second partition would maintain P2 and partition P1 into {I0, I1, Ierr} and generate P3 = {I2} since I2 goes to itself on b and to P2 on a. Next the algorithm would generate P4 = {I1) and next generate P5 = {Ierr} leaving P1 = {I0}. The final DFA would have therefore 5 states has shown below.



Again, on the input string w = "abbb" the DFA would traverse P1, P4, P2 and loop in P2 therefore accepting the string.

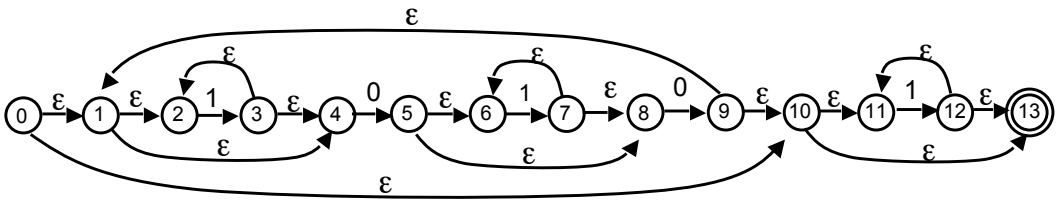
Important Note: If you try to apply the same construction, *i.e.* partitioning the set of states starting with an incomplete DFA where the "trap" state is missing you might be surprised to find that you reach at an incorrect minimal DFA.

Problem 4

Considering the alphabet Σ = {0,1} construct a Non-Deterministic-Finite Automaton (NFA) using the Thompson construction that is able to recognize the sentences generated by the regular expression (1*01*0)*1*. Does the sentence w = "1010" belong to the language generated by this regular expression? Justify.

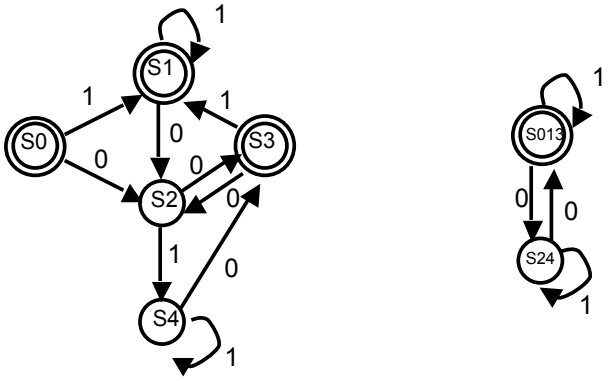
Solution:

The NFA is as shown below and where the start state is state labeled 0.



S0 = ε-closure (0) = {0, 1, 2, 4, 10, 11, 13} – this is a final state because of 13
S2 = DFAedge(S0,0) = ε-closure (S0, 0) = {5, 6, 8}
S1 = DFAedge(S0,1) = ε-closure (S0, 1) = {2, 3, 4, 11, 12, 13} – final state
DFAedge(S1,0) = ε-closure (S1, 0) = {5, 6, 8} = S2
DFAedge(S1,1) = ε-closure (S1, 0) = {2, 3, 4, 11, 12, 13} = S1
S3 = DFAedge(S2,0) = ε-closure (S2, 0) = {1, 2, 4, 9, 10, 11, 13} – final state
S4 = DFAedge(S2,1) = ε-closure (S2, 1) = {6, 7, 8}
DFAedge(S4,0) = ε-closure (S4, 0) = {1, 2, 4, 9, 10, 11, 13} = S3
DFAedge(S4,1) = ε-closure (S4, 1) = {6, 7, 8} = S4
DFAedge(S3,0) = ε-closure (S3, 0) = {5, 6, 8} = S2
DFAedge(S3,1) = ε-closure (S3, 1) = {2, 3, 4, 11, 12, 13} = S1

This results in the DFA shown below with starting state S0.



This DFA can be further minimize by recognizing that S0, S1 and S3 form a self-contained partition and S2 and S4 another partition resulting in the DFA on the right-hand-side.

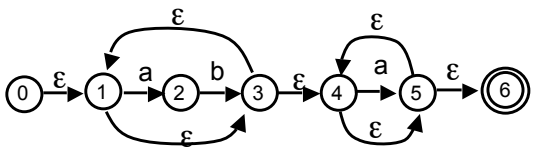
Problem 5

Consider the alphabet Σ = {a,b}.

a. Construct a Non-Deterministic-Finite Automaton (NFA) using the Thompson construction that is able to recognize the sentences generated by the regular expression RE = (ab)*. (a)*.
b. Do the sentences w_1 = "abaa" and w_2 = "aaa" belong to the language generated by this regular expression? Justify.
c. Convert the NFA in part a) to a DFA using the subset construction. Show the mapping between the states in the NFA and the resulting DFA.
d. Minimize the DFA using the iterative refinement algorithm discussed in class. Show your intermediate partition results and double check the DFA using the sentences w_1 and w_2.

Solution:

a) A possible construction (already simplified to have only a single ε-transition between states) would result in the NFA shown below and where the start state is labeled 0.



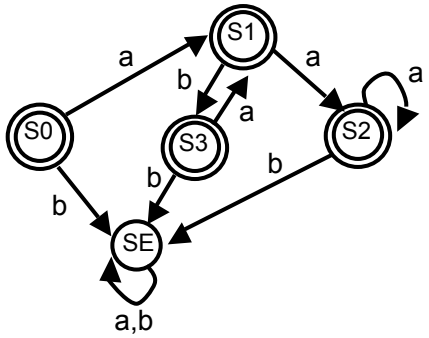
b) Both words are recognized by this NFA. Regarding the word "abaa" there is a path from state 0 to the accepting state 6, namely: 0,1,2,3,4,5,4,5,6. Regarding the word "aaa" the automaton may reach state 6 following the path 0,1,3,4,5,4,5,4,5,6.

c) Using the subset construction, we arrive at the following subsets and transitions.

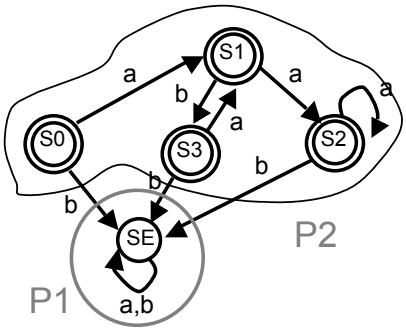
S0 = ε-closure (0) = {0, 1, 3, 4, 5, 6} – this is a final state because of state 6
S1 = DFAedge(S0,a) = ε-closure (goto(S0, a)) = {2, 4, 5, 6} – final state
SE = DFAedge(S0,b) = ε-closure (goto(S0, b)) = { }
S2 = DFAedge(S1,a) = ε-closure (goto(S1, a)) = {4, 5, 6} – final state
S3 = DFAedge(S1,b) = ε-closure (goto(S1, b)) = {3, 4, 5, 6} – final state

DFAedge(S2,a) = ε-closure (goto(S2, a)) = {4, 5, 6} = S2
DFAedge(S2,b) = ε-closure (goto(S2, b)) = { } = SE
DFAedge(S3,a) = ε-closure (goto(S3, a)) = {2, 4, 5, 6} = S1
DFAedge(S3,b) = ε-closure (goto(S3, b)) = { } = SE

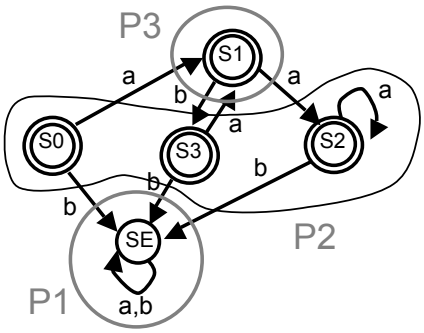
This results in the DFA shown below with starting state S0.



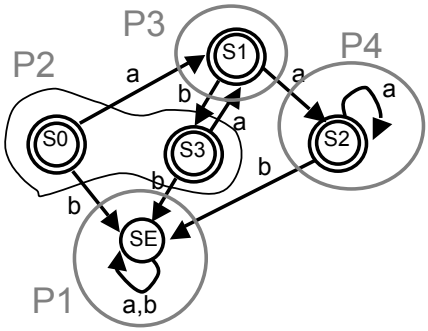
d) This DFA can be further minimize by using the iterative refinement partitioning yeilding the sequence of partitions indicated below.



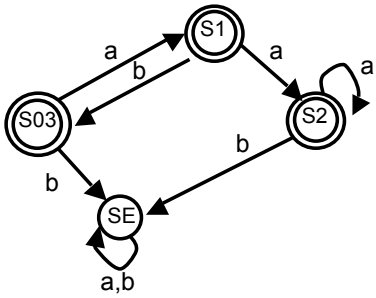
Initial partitioning on final states



Partitioning P2 on b



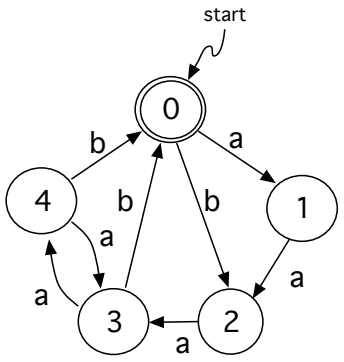
Partitioning P2 on a



final minimal DFA

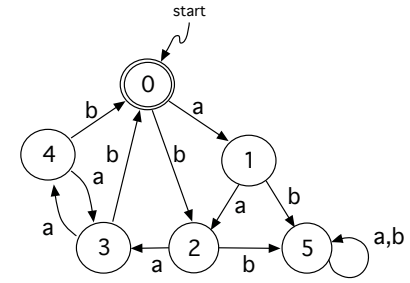
Problem 6

Minimize the FA shown below over the alphabet {a,b}. Make sure to convert it first into a DFA and them apply the iterative refinement algorithm discussed in class.

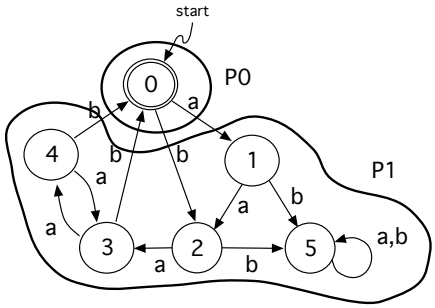


Solution:

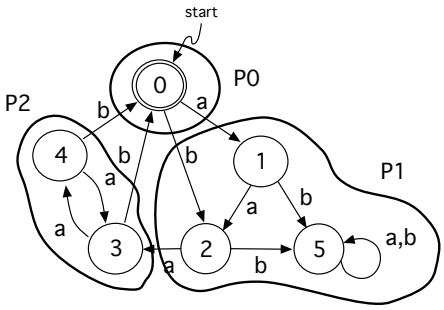
The first observation is that there are some states for which one of the transitions on the alphabet symbols is missing. This is the case with states "1" and "2". As such as need to augment this FA with a "trap-state" that is non-accepting as shown below in a). The remainder DFA result from the application of the iterative partitioning algorithm described in class.



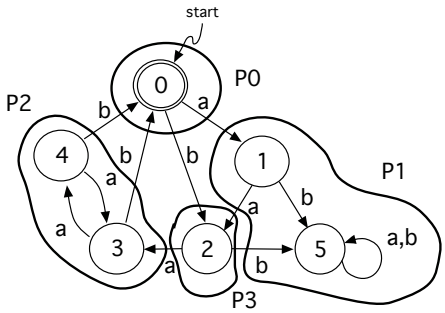
a) Augmented DFA



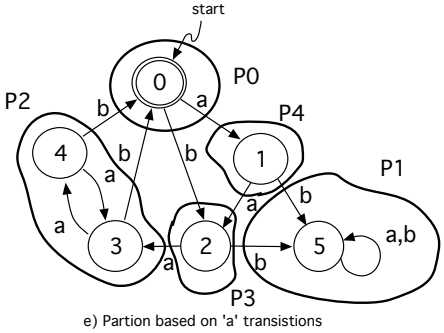
b) Partion based on accepting and non-accepting states



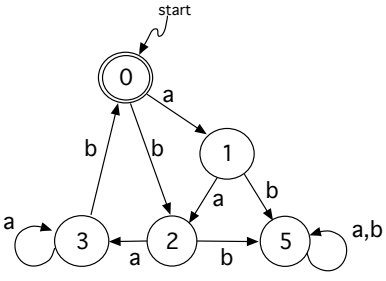
c) Partion based on 'b' transistions



d) Partion based on 'a' transistions



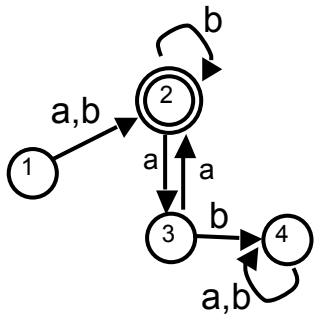
e) Partion based on 'a' transistions



f) Minimized DFA

Problem 7

Consider the DFA below with starting state 1 and accepting state 2:



a) Using the Kleene construction algorithm derive a regular expression.
b) Simplify the expression found in a) above.

Solution:

Before engaging in a very long sequence of concatenation and symbolic operations we make the observation that state 4 is a trap-state. As such we can safely ignore it since it is not an accepting state. What this means is that we can use the algorithm simply with k = 3 and need to use it to compute R^3_{12} as there are paths from the start state 1 to the final state 2 via state 3.

Expressions for k = 0

$R^0_{11} = (\varepsilon)$
$R^0_{12} = (a|b)$
$R^0_{23} = (a)$
$R^0_{32} = (a)$
$R^0_{22} = (\varepsilon|b)$

Expressions for k = 1

$R^1_{11} = R^0_{11} (R^0_{11})^* R^0_{11} \mid R^0_{11} = (\varepsilon)$
$R^1_{12} = R^0_{11} (R^0_{11})^* R^0_{12} \mid R^0_{12} = (a|b)$
$R^1_{23} = R^0_{21} (R^0_{11})^* R^0_{13} \mid R^0_{23} = (a)$
$R^1_{32} = R^0_{31} (R^0_{11})^* R^0_{12} \mid R^0_{32} = (a)$
$R^1_{22} = R^0_{21} (R^0_{11})^* R^0_{12} \mid R^0_{22} = (\varepsilon|b)$

Expressions for k = 2

$R^2_{11} = R^1_{12} (R^1_{22})^* R^1_{21} \mid R^1_{11} = R^1_{11}$ $= (\varepsilon)$
$R^2_{12} = R^1_{12} (R^1_{22})^* R^1_{22} \mid R^1_{12} = (a|b)\ (\varepsilon|b)^*\ (\varepsilon|b) \mid (a|b)$ $= (a|b)\ b^*$
$R^2_{23} = R^1_{22} (R^1_{22})^* R^1_{23} \mid R^1_{23} = (\varepsilon|b)\ (\varepsilon|b)^*\ (a) \mid (a)$ $= b^*(a)$
$R^2_{32} = R^1_{32} (R^1_{22})^* R^1_{22} \mid R^1_{32} = (a)\ (\varepsilon|b)^*\ (\varepsilon|b) \mid (a)$ $= (a)\ b^*$
$R^2_{22} = R^1_{22} (R^1_{22})^* R^1_{22} \mid R^1_{22} = (\varepsilon|b)\ (\varepsilon|b)^*\ (\varepsilon|b) \mid (\varepsilon|b)$ $= b^*$
$R^2_{13} = R^1_{12} (R^1_{22})^* R^1_{23} \mid R^1_{13} = (a|b)\ (\varepsilon|b)^*\ (a)$ $= (a|b)\ b^*\ (a)$
$R^2_{33} = R^1_{32} (R^1_{22})^* R^1_{23} \mid R^1_{33} = (a)\ (\varepsilon|b)^*\ (a)$ $= (a)\ b^*(a)$

Expressions for k = 3

$R^3_{11} = R^2_{13} (R^2_{33})^* R^2_{31} \mid R^2_{11} = R^2_{11} = (\varepsilon)$
$R^3_{12} = R^2_{13} (R^2_{33})^* R^2_{32} \mid R^2_{12} = (a|b)\ b^*\ (a)\ ((a)\ b^*(a))^*\ (a)\ b^* \mid (a|b)b^*$
$R^3_{23} = R^2_{23} (R^2_{33})^* R^2_{33} \mid R^2_{23} = b^*(a)\ ((a)\ b^*(a))^*\ (a)\ b^*(a) \mid b^*\ (a)$
$R^3_{32} = R^2_{33} (R^2_{33})^* R^2_{32} \mid R^2_{32} = (a)\ b^*(a)\ ((a)\ b^*(a))^*\ (a)\ b^* \mid (a)\ b^*$
$R^3_{22} = R^2_{23} (R^2_{33})^* R^2_{32} \mid R^2_{22} = b^*(a)\ ((a)\ b^*(a))^*\ (a)\ b^* \mid b^*$
$R^3_{13} = R^2_{13} (R^2_{33})^* R^2_{33} \mid R^2_{13} = (a|b)\ b^*\ (a)\ ((a)\ b^*(a))^*\ (a)\ b^*(a) \mid\ (a|b)\ b^*\ (a)$
$R^3_{33} = R^2_{33} (R^2_{33})^* R^2_{33} \mid R^2_{33} =\ (a)\ b^*(a)\ ((a)\ b^*(a))^*\ (a)\ b^*(a) \mid (a)\ b^*(a)$

Clearly these expressions have a lot of redundant terms. In fact we are only interested in R^3_{12} = (a|b) b* (a) ((a) b*(a))* (a) b* | (a|b)b* since the starting state is state 1 and the only accepting state is state 2.

b) To simplify the expression found above is very hard, as there are a lot of partial terms in this expression. Instead you follow the edges in the DFA and try to compose the regular expressions (in some sense ignoring what you have done in the previous step).

The simplest way to look at this problem is to see that there is a prefix (a|b) and then you return to state 2 either by (b*) or (aa), so a compact regular expressions would be

R = (a|b)((b)|(aa))*

a much more compact regular expressions that R^3_{12} above.

Problem 8

Consider the alphabet Σ = {a; b}. Define a short, possibly the shortest, regular expression that generates strings over Σ that contain exactly one "a" and at least one "b".

Solution:

The string must either start with one "b" or an "a", so we can use a "b*" prefix to account for any possible number of leading "b's" before the first "a". The single "a" must have at least one "b" either preceding it or just before it. After this single "a" there can be any number of trailing "b's". As possible (although not necessarily unique) regular expression denoting the sought language is R = b*(ab|ba)b*.

Problem 9

Given an NFA with ε-transitions how do you derive an equivalent NFA without those ε-transitions? Justify.

Solution:

For those states with ε-transitions merge the edges of the states at the end of those ε transitions into the current state. Iterate until there are no ε-transitions left. You still have a NFA but there will be at more multiple edges with the label out of a given state.

Problem 10

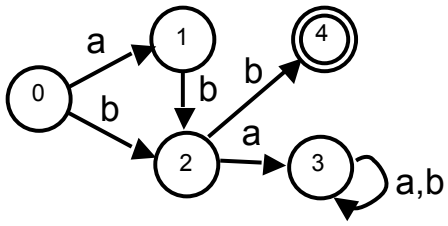
Given an NFA with several accept states, show how to convert it onto a NFA with exactly one start state and one accepting state.

Solution

The initial state is the same as in the original NFA. As to the accepting states just add ε-transitions from the original accepting states to a new accepting state and label that accepting state as the only accepting state of the new NFA.

Problem 11

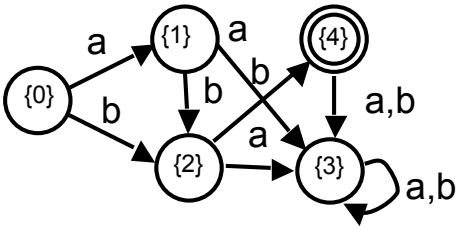
Given the Finite Automaton below with initial state 0 and alphabet {a,b} answer the following questions:



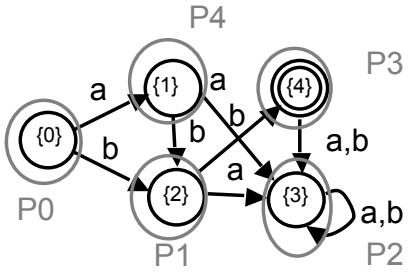
(a) Why is this FA a Non-Deterministic Finite Automaton (NFA)?
(b) Convert this NFA to a DFA using the closure computation.
(c) Minimize the resulting DFA.
(d) What is the Regular Expression matched by this NFA? You are advised not to use the automatic Kleene construction or try to look at the input NFA but rather the *correctly* minimized DFA.

Solution:

(a) This FA is already a DFA, as the only transitions that are missing all go to the trap state 3.
(b) The subset construction, which in this case yields the same original DFA on the left where we have noted the subset each state stands for.



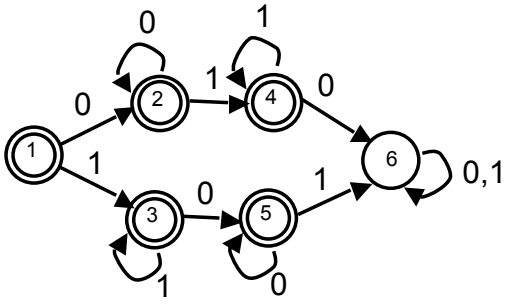
(c) Using the iterative partition refinement, we have the following partitions (below left) and the resulting minimized DFA on the right again with the states relabelled. Notice that this is a special case where the refinement algorithm cannot refine any sub-sets of states given that the original DFA was already minimal.



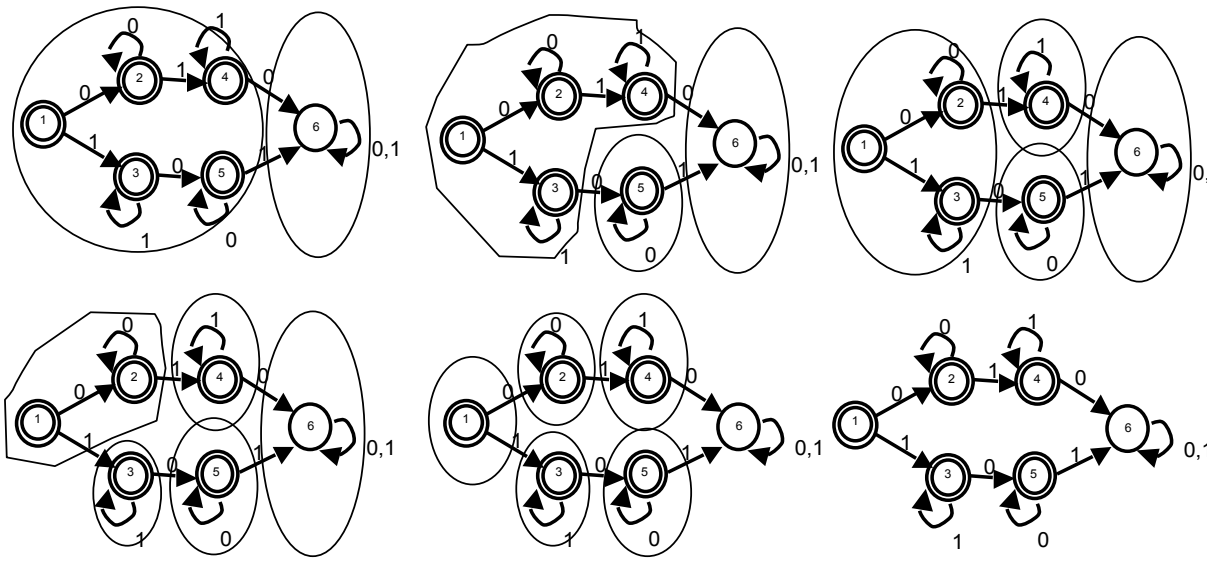
(d) Based on the last DFA the regular expression identified by this FA must have exactly two consecutive "b" symbol possibly preceded by a single "a" symbol. For instance, the string "abb" is in the language but the string "abba" or "ba" are not. The regular expression can be denoted by a*bb.

Problem 12
Draw the smallest DFA that accepts the language derived by the regular expression (0*1*|1*0*).

Solution:
A possible translation guided by the idea of the Thompson construction is shown below.



Using the iterative refinements algorithm for DFA minimization indeed confirms that this is the smallest possible DFA. The sequence of refinements is shown below.



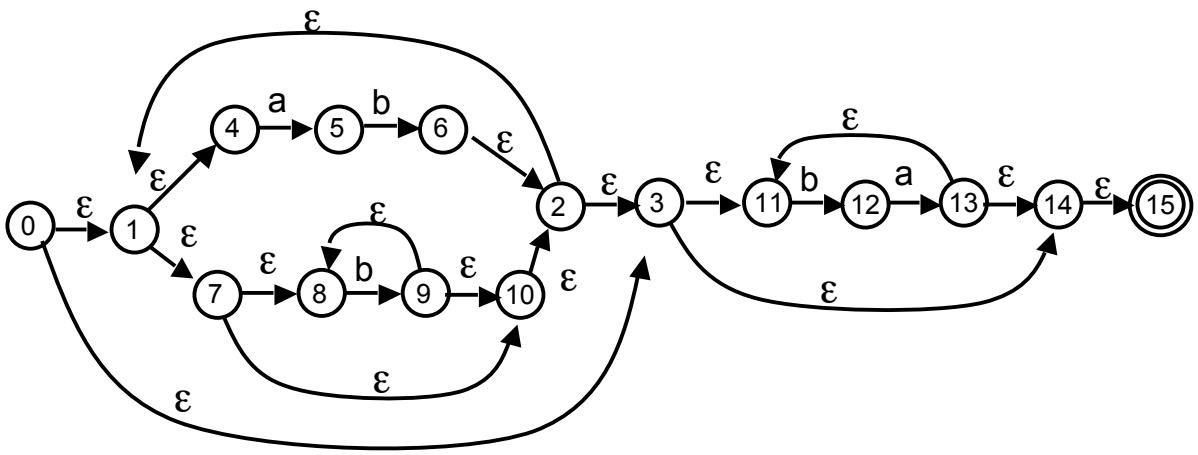
Problem 13

Given the regular expression RE = (ab | b)* . (ba)* over the alphabet Σ = {a,b} do the following transformations:

a. Derive a NFA using the Thompson construction capable of identifying the strings generated by this regular expression.
b. Convert the NFA obtained in a) to a DFA.
c. Minimize the resulting DFA using the iterative refinement algorithm discussed in class.
d. Determine in how many steps is the sequence "ababaaa' processed. Justify.

Solution:

Regarding a) the NFA is given below. Here we made some very simple simplification that two or more sequences of ε-transitions can be converted to a single ε-transitions. This substantially reduces the number of states in the NFA and thus facilitates the construction of the equivalent DFA.

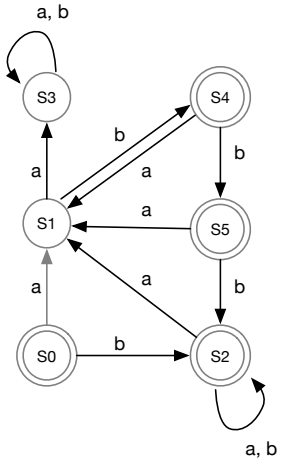


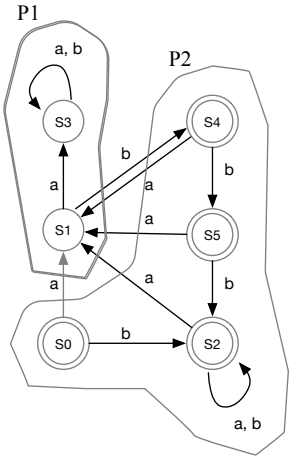
Applying the subset construction, we have the following states

S0 = ε-closure (0) = {0, 1, 4, 7, 8, 10, 2, 3, 11, 14, 15} – this is a final state because of 15
S1 = DFAedge(S0,a) = ε-closure (goto(S0, a)) = {5}
S2 = DFAedge(S0,b) = ε-closure (goto(S0, b)) = {1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 14, 15} – final
S3 = DFAedge(S1,a) = ε-closure (goto(S1, a)) = {}
S4 = DFAedge(S1,b) = ε-closure (goto(S1, b)) = {1, 2, 3, 4, 6, 7, 8, 10, 11, 14, 15} – final
DFAedge(S2,a) = ε-closure (goto(S2, a)) = {5} = S1
DFAedge(S2,b) = ε-closure (goto(S2, b)) = {1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 14, 15} = S2
DFAedge(S3,a) = ε-closure (goto(S3, a)) = {} = S3
DFAedge(S3,b) = ε-closure (goto(S3, b)) = {} = S3
DFAedge(S4,a) = ε-closure (goto(S4, a)) = {5} = S1
S5 = DFAedge(S4,b) = ε-closure (goto(S4, b)) = {1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 14, 15} - final
DFAedge(S5,a) = ε-closure (goto(S5, a)) = {5} = S1
DFAedge(S5,b) = ε-closure (goto(S5, b)) = {1, 2, 3, 4, 7, 8, 9, 10, 11, 14, 15} = S2

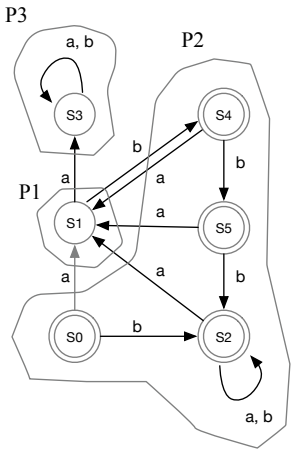
So there are a total of 6 states, S0, S1, S2, S3, S4, S6 and the corresponding DFA is as shown by the table below (bold states are final states) and then in the figure below the table.

	Next State	
Current State	*a*	*b*
S0	S1	S2
S1	S3	S4
S2	S1	S2
S3	S3	S3
S4	S1	S5
S5	S1	S2

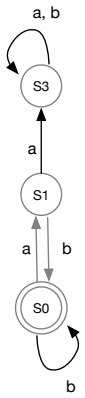




Initial partition based on Acceptance criterion.



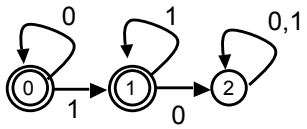
Refinement of P1 based on 'b'.



Final partition and relabelling of states.

Problem 14

Given the DFA below describe in English the set of strings accepted by it.



Solution:

Any string over the alphabet that contains any number of consecutive 1s, including none with a prefix of zero or more consecutive 0s.

Problem 15

Given a regular language L. *i.e.,* a language described by a regular expression, prove that the reverse of L is also a regular language (**Note**: the reverse of a language L is L^R where for each word w in L, w^R is in L^R. Given a word w over the given alphabet, w^R is constructed by spelling w backwards).

Solution:

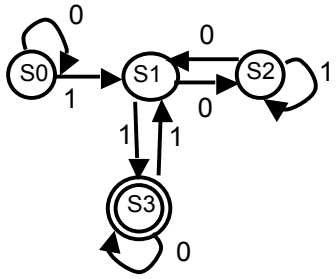
If L is a regular language then there exists a DFA M_1 that recognizes it. Now given M_1 we can construct M_2 that recognizes the reverse of L with respect to the input alphabet. We now describe how to construct M_2. M_2 is a replica of M_1 but reversing all the edges. The final state of M_2 is the state that used to be the start state of M_1. The start state of M_2 is a new state with ε-transitions to the states in M_2 that used to be the final states of M_1. Now because M_1 might have multiple final states M_2 is by construction an NFA. Given the equivalence of NFA and regular expressions we have shown that if L is regular so is L^R.

Problem 16

Draw the DFA capable of recognizing the set of all strings beginning with a 1 which interpreted as the binary representation of an integer (assuming the last digit to be processed is the least significant) is congruent to zero modulo 3 *i.e.*, the numeric value of this binary representation is a multiple of 3.

Solution:

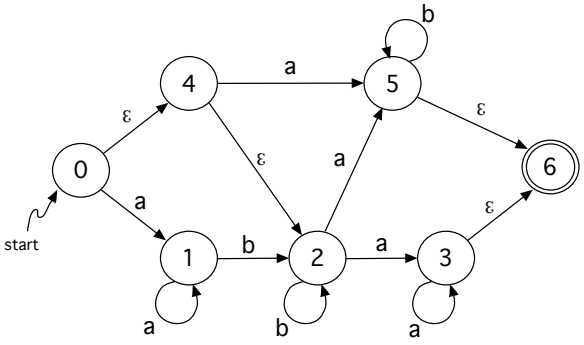
The hard part about this problem is that you need to keep track with the already observed bits what the remainder of the division by 3 is. Given that you have a reminder you would need no more that 2 states, one for each of the remainder values 1 through 2 being the state that represents a remainder of zero the accepting state, in this case state S3. The DFA below accomplishes this. You can verify this DFA by trying the number 12 in binary 1100 or 21 in binary 10101. Notice that in the last state S3 any additional 0 means you are shifting the bits by one bit, *i.e.,* multiplying by 2, hence staying in the same state.



Problem 17

Consider the alphabet Σ = {a,b}.

a. Consider the Non-Deterministic Finite Automaton (NFA) depicted below. Why is this automaton non-deterministic? Explain the various source on indeterminacy.
b. Do the sentences w_1 = "aba" and w_2 = "aab" belong to the language recognized by this FA? Justify.
c. Convert the NFA in part a) to a DFA using the subset construction. Show the mapping between the states in the NFA and the resulting DFA.
d. Minimize the DFA using the iterative refinement algorithm described in class. Show your intermediate partition results and double check the DFA using the sentences w_1 and w_2.



Solution:

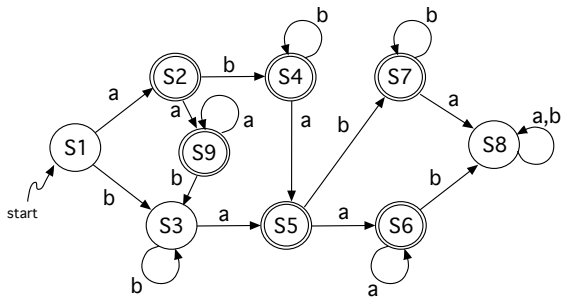
a) This is indeed a NFA for two reasons. First, it includes ε-transitions. Second, in state 2 there are two transitions on the same terminal or alphabet symbol, "a".

b) Regarding the word “aba” there is a path from state 0 to the accepting state 6, namely: 0,4,5,6. Regarding the word “aab” the automaton will never be able to reach the state 6 as in order to spell out the “b” character will necessarily be in state 5 and to reach that state we cannot have two consecutive "a" characters we would be either in state 1 or state 4.

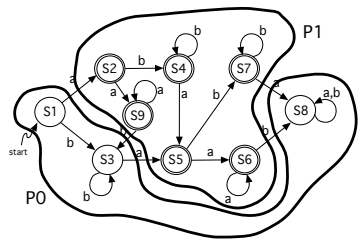
c) Using the subset construction, we arrive at the following subsets and transitions.

S1 = ε-closure ({0}) = {0, 2, 4} – this is not a final state.
S2 = DFAedge(S1,a) = ε-closure (goto(S1, a)) = {1, 3, 5, 6} – final state
S3 = DFAedge(S1,b) = ε-closure (goto(S1, b)) = {2}
S9 = DFAedge(S2,a) = ε-closure (goto(S2, a)) = {1, 3, 6} – final state
S4 = DFAedge(S2,b) = ε-closure (goto(S2, b)) = {2, 5, 6} – final state.
S5 = DFAedge(S3,a) = ε-closure (goto(S3, a)) = {3, 5, 6} – final state.
DFAedge(S3,b) = ε-closure (goto(S3, b)) = {2} = S3
DFAedge(S4,a) = ε-closure (goto(S4, b)) = {3, 5, 6} = S5 – final state.
DFAedge(S4,b) = ε-closure (goto(S4, b)) = {2, 5, 6} = S4 – final state.
S6 = DFAedge(S5,a) = ε-closure (goto(S5, a)) = {3, 6} – final state.
S7 = DFAedge(S5,b) = ε-closure (goto(S5, b)) = {5, 6} – final state.
DFAedge(S6,a) = ε-closure (goto(S6, a)) = {3, 6} = S6 – final state.
S8 = DFAedge(S6,b) = ε-closure (goto(S6, b)) = { }
DFAedge(S7,a) = ε-closure (goto(S7, a)) = { } = S8
DFAedge(S7,b) = ε-closure (goto(S7, b)) = {5, 6} = S7 – final state.
DFAedge(S8,a) = ε-closure (goto(S8, a)) = { } = S8
DFAedge(S8,b) = ε-closure (goto(S8, b)) = { } = S8
DFAedge(S9,a) = ε-closure (goto(S9, a)) = {1, 3, 6} = S9
DFAedge(S9,b) = ε-closure (goto(S9, b)) = {2} = S3

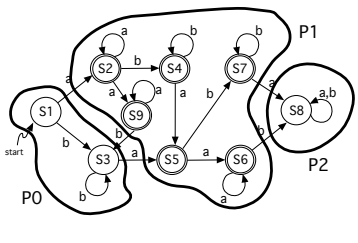
This results in the DFA shown below with starting state S1.



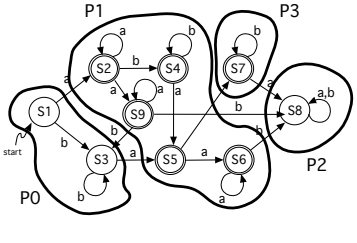
d) We can try to minimize this DFA using the iterative refinement algorithm described in class. The figure below depicts a possible sequence of refinements. For each step we indicate the criteria used to discriminate between states in the previous partition. As can be seen, the algorithm leads to no refinement, so this is a particular case where the sub-set algorithm that help us derive the DFA from its original NFA does yield a minimal DFA.



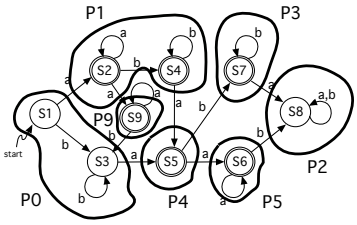
(a) Partition between accepting and non-accepting states



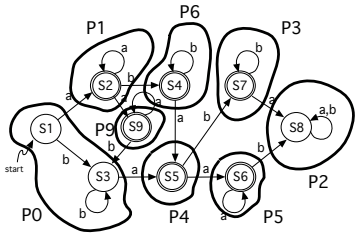
(b) Discrimination based on "a" transitions



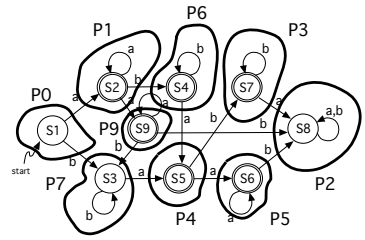
(c) Discrimination based on "a" transitions



(d) Discrimination based on "b" transitions



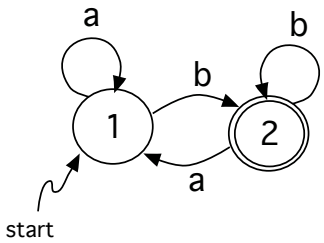
(e) Discrimination based on "a" transitions



(f) Discriminaion based on "a" transitions

Problem 18

Consider the DFA below with starting state 1 and accepting state 2:



a) Describe in English the set of strings accepted by this DFA.
b) Using the Kleene construction algorithm derive the regular expression recognized by this automaton simplifying it as much as possible.

Solution:

a) This automaton recognizes all non-null strings over the {a,b} alphabet that end with "b".
b) The derivations are shown below with the obvious simplification.

Expressions for k = 0
$R^0_{11} = (a \mid \varepsilon)$
$R^0_{12} = (b)$
$R^0_{21} = (a)$
$R^0_{22} = (b \mid \varepsilon)$

Expressions for k = 1

$R^1_{11} = R^0_{11} (R^0_{11})^* R^0_{11} \mid R^0_{11} = (a|\varepsilon) . (a|\varepsilon)^* . (a|\varepsilon) \mid (a|\varepsilon) = a^*$

$R^1_{12} = R^0_{11} (R^0_{11})^* R^0_{12} \mid R^0_{12} = (b|\varepsilon) . (a|\varepsilon)^* . (b) \mid (b) = ba^* \mid ba^*b$

$R^1_{21} = R^0_{21} (R^0_{11})^* R^0_{11} \mid R^0_{21} = (a) . (a|\varepsilon)^* . (b|\varepsilon) \mid (a) = a^+ \mid a^+b$

$R^1_{22} = R^0_{21} (R^0_{11})^* R^0_{12} \mid R^0_{22} = (a) . (a|\varepsilon)^* . (a|\varepsilon) \mid (b \mid \varepsilon) = (a^* \mid b)$

Expressions for k = 2

$R^2_{11} = R^1_{12} (R^1_{22})^* R^1_{21} \mid R^1_{11} = (ba^* \mid ba^*b) . (a^* \mid b)^* . (a^+ \mid a^+b) \mid (a^*)$

$R^2_{12} = R^1_{12} (R^1_{22})^* R^1_{22} \mid R^1_{12} = (ba^* \mid ba^*b) . (a^* \mid b)^* . (a^+ \mid a^+b) \mid (ba^* \mid ba^*b)$

$R^2_{21} = R^1_{12} (R^1_{22})^* R^1_{21} \mid R^1_{21} = (ba^* \mid ba^*b) . (a^* \mid b)^* . (a^+ \mid a^+b) \mid (a^+ \mid a^+b)$

$R^2_{22} = R^1_{12} (R^1_{22})^* R^1_{22} \mid R^1_{22} = (ba^* \mid ba^*b) . (a^* \mid b)^* . (a^+ \mid a^+b) \mid (a^* \mid b)$

$L = R^2_{12} = (ba^* \mid ba^*b) . (a^* \mid b)^* . (a^+ \mid a^+b) \mid (ba^* \mid ba^*b)$

As can be seen the simplification of any of these regular expressions beyond the expresions for k=1 is fairly complicated. This method, although correct by design leads to regular expressions that are far from being a minimal or even the most compact representation of the regular language the DFA recognizes.

Problem 19

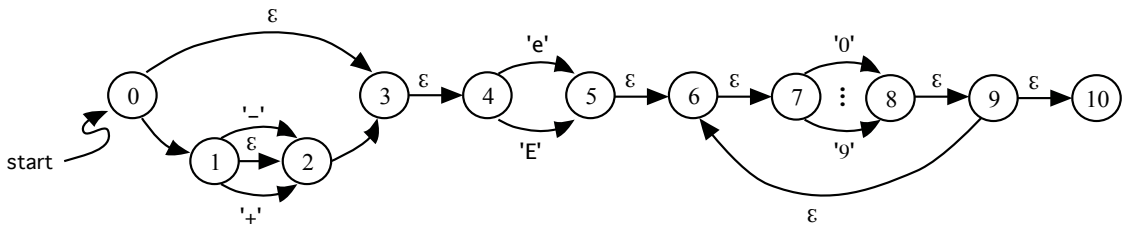
For the regular expression depicted below answer the following questions:

$$RE = (\varepsilon \mid - \mid +)? \,.\, (e \mid E) \,.\, (0\text{-}9)+$$

a) Using the Thompson construction (or a simplification thereof) derive a NFA that recognizes the strings specified by this RE.
b) Convert the NFA in a) to an equivalent DFA using the subset construction.
c) Make sure the DFA M_1 found in b) above is minimal (you do not need to show that it is in fact minimal) and construct the DFA M_2 that accepts the complement of the regular language the first DFA accepts. Show that M_2 does accept the word w = "10e" which is not accepted by the original DFA.

Solution:

a) The figure below depicts the NFA with ε-transitions and where we have simplified the transition on numeric digits.



b) We use the subset construction by tracing for every set of states the other possible states the NFA could be in if it were to traverse ε-edges only. The list below depicts the results of this construction where we also describe the computation of edge DFA edge using the function DFA_edge and then computing the e-closure of the resulting intermediate set of states.

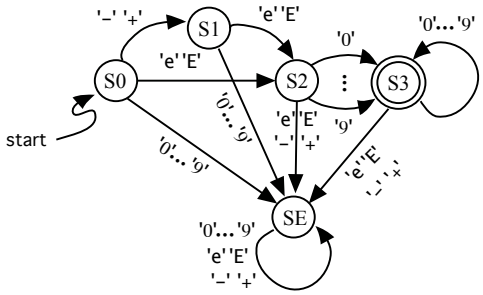
ε-closure({0}) = { 0, 1, 2, 3, 4} = S0
DFA_edge(S0,'-') = ε-closure({1}) = { 2, 3, 4} = S1
DFA_edge(S0,'+') = ε-closure({2}) = { 2, 3, 4} = S1
DFA_edge(S0,'e') = ε-closure({5}) = { 5, 6, 7} = S2
DFA_edge(S0,'E') = ε-closure({5}) = { 5, 6, 7} = S2
DFA_edge(S0,'<digit>') = ε-closure({}) = SE; error state

DFA_edge(S1,'-') = ε-closure({}) = SE; error state
DFA_edge(S1,'+') = ε-closure({}) = SE; error state
DFA_edge(S1,'e') = ε-closure({5}) = { 5, 6, 7 } = S2
DFA_edge(S1,'E') = ε-closure({5}) = { 5, 6 ,7 } = S2
DFA_edge(S1,'<digit>') = ε-closure({}) = SE; error state

DFA_edge(S2,'-') = ε-closure({}) = SE; error state
DFA_edge(S2,'+') = ε-closure({}) = SE; error state
DFA_edge(S2,'e') = ε-closure({}) = SE; error state
DFA_edge(S2,'E') = ε-closure({}) = SE; error state
DFA_edge(S2,'<digit>') = ε-closure({8}) = { 6, 7, 8, 9, 10 } = S3;

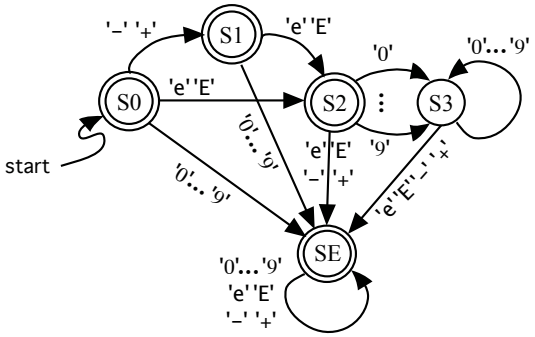
DFA_edge(S3,'-') = ε-closure({}) = SE; error state
DFA_edge(S3,'+') = ε-closure({}) = SE; error state
DFA_edge(S3,'e') = ε-closure({}) = SE; error state
DFA_edge(S3,'E') = ε-closure({}) = SE; error state
DFA_edge(S3,'<digit>') = ε-closure({8}) = { 6, 7, 8, 9, 10 } = S3;

The figure below depicts the DFA resulting from the application of the subset construction. Notice that this is a fully specified DFA where each state includes transition for all the characters in the considered alphabet.



c) Let M_1 be the DFA found in b). This DGA is in fact minimal. Informal proof: All states are required. S0 is the start state. S1 captures the fact that we have observed a sign. State S2 captures to fact we need to observe at least one digit. All the digits are then captured by state S2. State SE is the trap error state.

Let M_1 be this first DFA. We construct M_2 by making all accepting states of M_1 non-accepting states and all non-accepting state s of M_1 as accepting states of M_2. The state start is unchanged. This is a DFA as we only change the state of non-accepting states. The start state is still a singleton and all transitions are clearly identified in the newly created DFA as shown below.

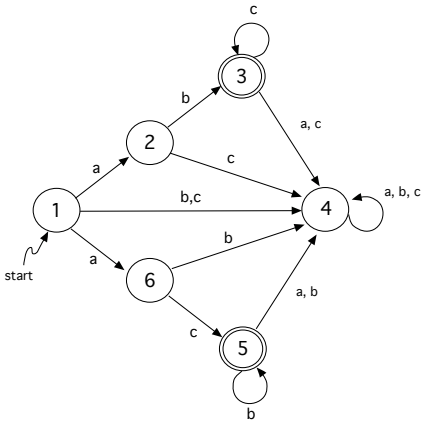


In this newly created DFA the input string w = "10e" takes the DFA to state SE which is an accepting state n this "complement" machine.

Problem 20

Consider the alphabet Σ = {a, b, c}.

a. Consider the Non-Deterministic Finite Automaton (NFA) depicted below. Why is this automaton non-deterministic? Explain the various source on indeterminacy.
b. Do the sentences w_1 = "ab" and w_2 = "bbb" belong to the language generated by this FA? Justify.
c. Convert the NFA in part a) to a DFA using the subset construction. Show the mapping between the states in the NFA and the resulting DFA.
d. By inspection can you derive a regular expression (RE) this NFA detects?



Solution:

a) This FA is non-deterministic as for at least one of its states there are two transitions on the same label to two distinct states. This is the case with state 1 where on 'a' the FA goes to state 2 and 6. Similarly, states 3 and 5 exhibit the same behavior.
b) The sentence w_1 = "ab" belongs to the language accepted by this FA as there is a path through states 1, 2 and 3 that speels out this ord and state 3 is an accepting state. As to the word w_2 = "bbb" any word that begins with 'b' leads to state 4 which is a trap and non-accepting state. As such w_2 does not belong to the language recognized by this FA.
c) A simplified sub-set construction is as shown below:

S1 = ε-closure ({1}) = {1} – this is not a final state.
DFAedge(S1, a) = ε-closure (goto(S1, a)) = {2,6} = S2
DFAedge(S1, b) = ε-closure (goto(S1, b)) = {4} = S3
DFAedge(S1, c) = ε-closure (goto(S1, c)) = {4} = S3

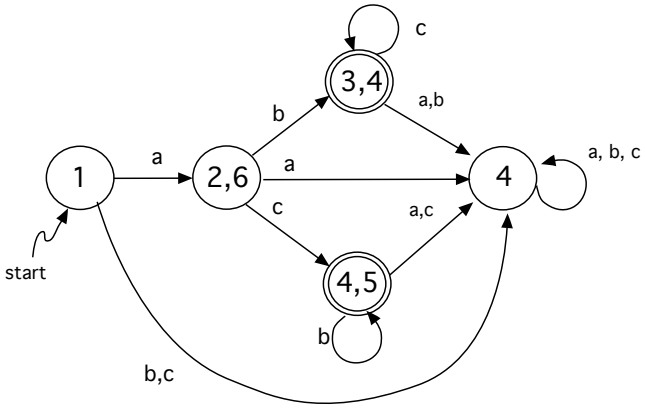
DFAedge(S2, a) = ε-closure (goto(S2, a)) = {4} = S3
DFAedge(S2, b) = ε-closure (goto(S2, b)) = {3,4} = S4
DFAedge(S2, c) = ε-closure (goto(S2, c)) = {4,5} = S5

DFAedge(S3, a) = ε-closure (goto(S3, a)) = {4} = S3
DFAedge(S3, b) = ε-closure (goto(S3, b)) = {4} = S3
DFAedge(S3, c) = ε-closure (goto(S3, c)) = {4} = S3

DFAedge(S4, a) = ε-closure (goto(S4, a)) = {4} = S3
DFAedge(S4, b) = ε-closure (goto(S4, b)) = {4} = S3
DFAedge(S4, c) = ε-closure (goto(S4, c)) = {3,4} = S4

DFAedge(S5, a) = ε-closure (goto(S5, a)) = {4} = S3
DFAedge(S5, b) = ε-closure (goto(S5, b)) = {4,5} = S5
DFAedge(S5, c) = ε-closure (goto(S5, c)) = {4} = S3

In this construction we have simplified the resulting DFA by making the transition between state {2,6} on 'a' directly move to state 4 which is a trap or error state. A more mechanical procedure would have seen the creation of an additional trap state that is equivalent to state 4.



d) By inspection of the two paths from state 1 to either state 3 or state 5 we can immediately see that the language accepted by this FA can be described by the regular expressions RE = abc* | acb*.

Problem 21

Consider the alphabet $\Sigma = \{a, b, c\}$.

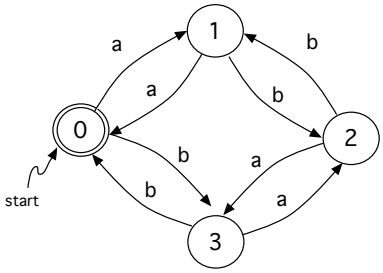
Construct DFAs that accept the following languages. Argue that your DFA is minimal (with respect to the number of states) or use the DFA minimization algorithm to show that that is the case.

a. $\{w \in \{a,b\}^* \mid |w|_a = 2n, |w|_b = 2m, m,n \in N\}$

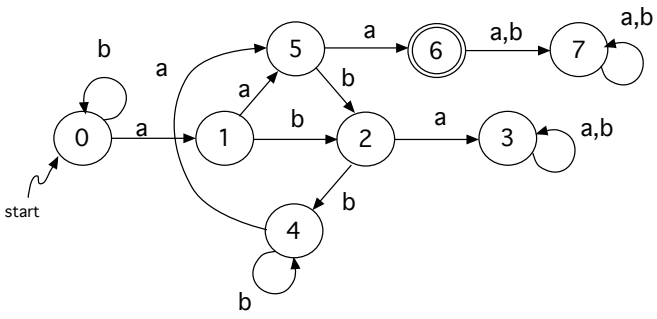
b. $\{w \in \{a, b\}^* \mid |w|_a = 3n, n \in N\}$ and w doesn't contain the substring aba

Solution:

a.



b.



Problem 22

Develop a regular expression (RE) that detects the longest string over the alphabet {a-z} with the following properties:

1. The string begins with an 'a' character and ends with a 'z' character;
2. After the first 'a', the string can include no alternation of two subsequences, namely a sequence that begins with a 'b' character followed by zero or more 's' characters ending with an 'e' character and a subsequence that begins with a 'c' character but does not have any 's' character until a terminating 't' character.
3. If at any point in the subsequence there is a 'x' character, that specific sequence is considered terminated, i.e., the 'x' acts as the 'e' character in the first type of sequence and as the 't' character in the second type of sequence.
4. The two subsequences cannot be nested but can be repeated in any alternating order.

As an example, the string "absssecaaabbbafghtbsez" is to be accepted as well as the string "abssxcfftz"

Questions:

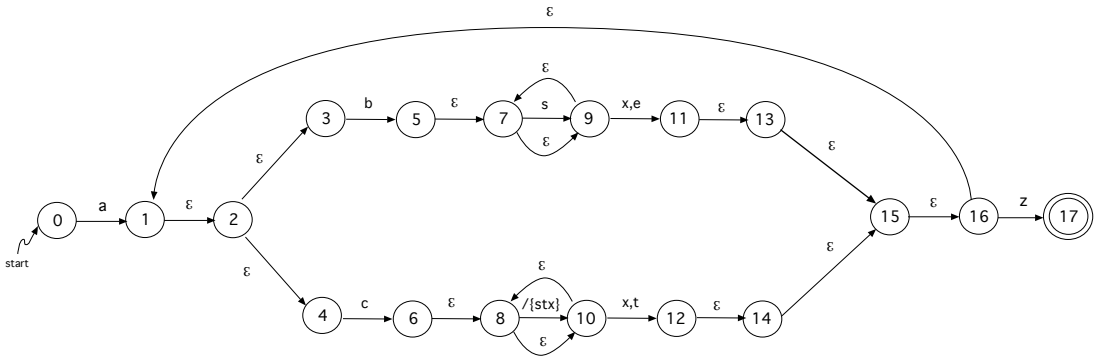
e. Develop a regular expression that captures the structure of the acceptable strings described above. Use short-hands to capture subsets of characters in the alphabet so that your description and the corresponding FA are kept short.
f. Using the regular expression define in section a) devise the corresponding Non-Deterministic Finite Automaton (NFA) using the Thompson construction described in class.
g. Convert the NFA in section b) to a DFA using the subset construction. Show the mapping between the states in the NFA and the resulting DFA.
h. Minimize the DFA derived in section c) (or show it is already minimal) using the iterative refinement algorithm described in class.

Solution:

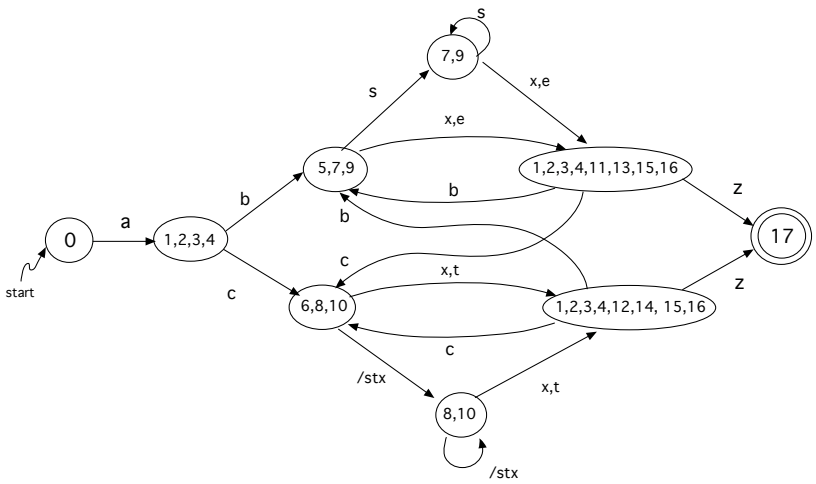
a) Among the many possible solutions for a regular expression the one below seems to be the most intuitive. Here we use the short-hand "notSTX" for all characters in the alphabet except 's' and 't' i.e., notSTX = {a-r, u, v, y, z}.

$$RE = a \,.\, [\, (\, b \,.\, (s^*)\,.\,(x|e)\,) \,|\, (\, c \,.\, (\, notSTX^*\,)\,.\,(x \mid t)\,)\,]^* \,.\, z$$

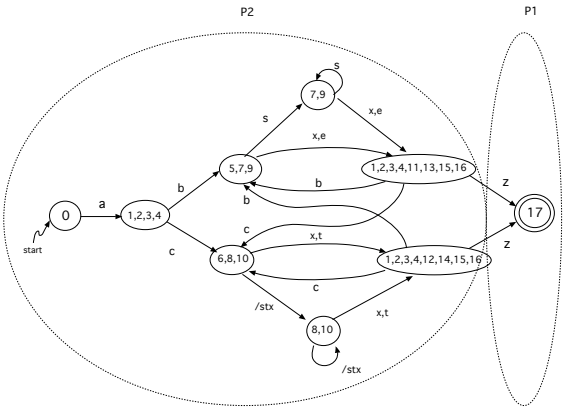
b) The NFA below reflects the structure of the RE presented above where we have also "compressed' some ε-transitions for brevity, in particular the ones that reflect Kleene closure and alternation.

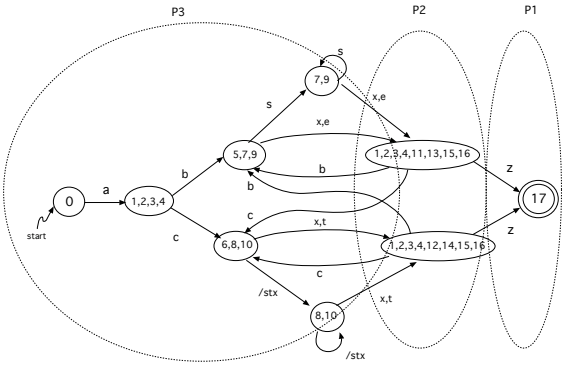


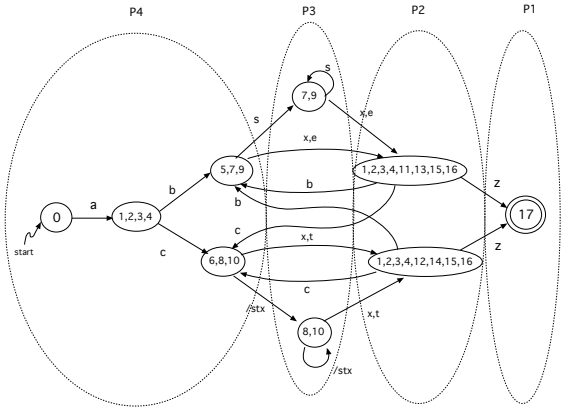
c) The subset construction of this NFA is as shown below where again we have merged transitions on character with analogous behavior.

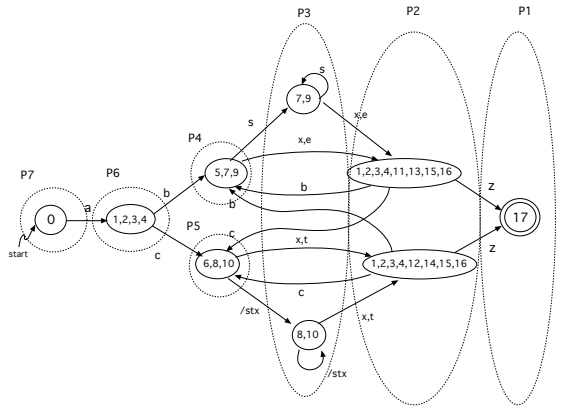


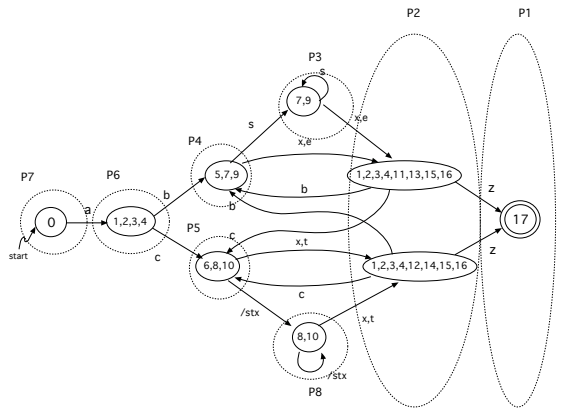
d) The minimization of this DFA using the iterative refinement algorithm is summarizaed below. We begin by a simple partitionbetween the accepting states and the non-accepting states. This immediately alows us to “isolate” the accepting state 17. The second partition used the character ‘z’ as the discriminatig token showing that states “1,2,3,4,11,13,15,16” and “1,2,3,4,12,14,15,16” may be equivalent. The last partition diagrams reflect the fact that all states are disjoint.

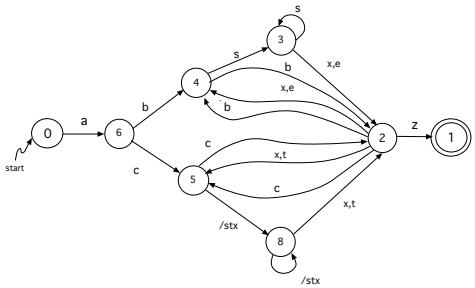

P2
P1
start
0
a
1,2,3,4
b
c
5,7,9
6,8,10
7,9
8,10
s
x,e
x,t
z
17
/stx
1,2,3,4,11,13,15,16
1,2,3,4,12,14,15,16


P3
P2
P1
start
0
a
1,2,3,4
b
c
5,7,9
6,8,10
7,9
8,10
s
x,e
x,t
z
17
/stx
1,2,3,4,11,13,15,16
1,2,3,4,12,14,15,16


P4
P3
P2
P1
start
0
a
1,2,3,4
b
c
5,7,9
6,8,10
7,9
8,10
s
x,e
x,t
z
17
/stx
1,2,3,4,11,13,15,16
1,2,3,4,12,14,15,16


P3
P2
P1
P7
P6
P4
P5
start
0
a
1,2,3,4
b
c
5,7,9
6,8,10
7,9
8,10
s
x,e
x,t
z
17
/stx
1,2,3,4,11,13,15,16
1,2,3,4,12,14,15,16

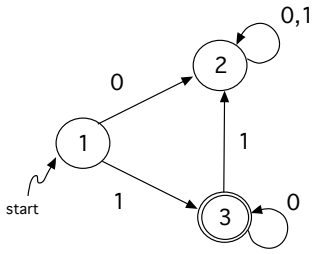

P2
P1
P3
P4
P7
P6
P5
P8
start
0
a
1,2,3,4
b
c
5,7,9
6,8,10
7,9
8,10
s
x,e
x,t
z
17
/stx
1,2,3,4,11,13,15,16
1,2,3,4,12,14,15,16


start
0
a
6
b
c
4
5
3
2
1
8
s
x,e
x,t
z
/stx

Problem 23

Given the DFA below over the alphabet {0,1} determine the following:

a. Use the dynamic-programming Kleene algorithm to derive the regular expression that denotes the language accepted by it. Make sure your labelling of the DFA states is correct and that the DFA is completely specified, i.e., each state has transitions on all the alphabet characters.
b. Describe succintely what are the words accepted by this DFA?



Solution:

a. Use the dynamic-programming Kleene algorithm to derive the regular expression that denotes the language accepted by it. Make sure your labelling of the DFA states is correct and that the DFA is completely specified, i.e., each state has transitions on all the alphabet characters.

Expressions for k = 0

$R^0_{11} = (\varepsilon)$
$R^0_{12} = (0)$
$R^0_{13} = (1)$

$R^0_{21} = \emptyset$
$R^0_{22} = (0 \mid \varepsilon)$
$R^0_{23} = \emptyset$

$R^0_{31} = \emptyset$
$R^0_{32} = (1)$
$R^0_{33} = (0 \mid \varepsilon)$

Expressions for k = 1

$R^1_{11} = R^0_{11} (R^0_{11})^* R^0_{11} \mid R^0_{11} = (\varepsilon)(\varepsilon)^*(\varepsilon) \mid (\varepsilon) = (\varepsilon)$
$R^1_{12} = R^0_{11} (R^0_{11})^* R^0_{12} \mid R^0_{12} = (\varepsilon)(\varepsilon)^*(0) \mid (0) = (0)$
$R^1_{13} = R^0_{11} (R^0_{11})^* R^0_{13} \mid R^0_{13} = (\varepsilon)(\varepsilon)^*(1) \mid (1) = (1)$

$R^1_{21} = R^0_{21} (R^0_{11})^* R^0_{11} \mid R^0_{21} = R^0_{21} = \emptyset$
$R^1_{22} = R^0_{21} (R^0_{11})^* R^0_{12} \mid R^0_{22} = R^0_{22} = (0 \mid \varepsilon)$
$R^1_{23} = R^0_{21} (R^0_{11})^* R^0_{13} \mid R^0_{23} = R^0_{23} = \emptyset$

$R^1_{31} = R^0_{31} (R^0_{11})^* R^0_{31} \mid R^0_{31} = R^0_{31} = \emptyset$
$R^1_{32} = R^0_{31} (R^0_{11})^* R^0_{32} \mid R^0_{32} = R^0_{32} = (1)$
$R^1_{33} = R^0_{31} (R^0_{11})^* R^0_{33} \mid R^0_{33} = R^0_{33} = (0 \mid \varepsilon)$

Expressions for k = 2

$R^2_{11} = R^1_{12}(R^1_{22})^*R^1_{21} \mid R^1_{11} = (0) \,.\, (0 \mid \varepsilon)^*.\, \emptyset \mid (\varepsilon) = (\varepsilon)$
$R^2_{12} = R^1_{12}(R^1_{22})^*R^1_{22} \mid R^1_{12} = (0) \,.\, (0 \mid \varepsilon)^* \,.\, (0 \mid \varepsilon) \mid (0) = (0^+)$
$R^2_{13} = R^1_{12}(R^1_{22})^*R^1_{23} \mid R^1_{13} = (0) \,.\, (0 \mid \varepsilon)^* \,.\, \emptyset \mid (1) = (1)$

$R^2_{21} = R^1_{22}(R^1_{22})^*R^1_{21} \mid R^1_{21} = (0 \mid \varepsilon)\,(0 \mid \varepsilon)^*\,\emptyset \mid \emptyset = \emptyset$
$R^2_{22} = R^1_{22}(R^1_{22})^*R^1_{22} \mid R^1_{22} = (0 \mid \varepsilon)\,(0 \mid \varepsilon)^*\,(0 \mid \varepsilon) \mid (0 \mid \varepsilon) = (0^*)$
$R^2_{23} = R^1_{22}(R^1_{22})^*R^1_{23} \mid R^1_{23} = (0 \mid \varepsilon)\,(0 \mid \varepsilon)^*\,\emptyset \mid \emptyset = \emptyset$

$R^2_{31} = R^1_{32}(R^1_{22})^*R^1_{21} \mid R^1_{31} = (1) \,.\, (0 \mid \varepsilon)^* \,.\, \emptyset \mid \emptyset = \emptyset$
$R^2_{32} = R^1_{32}(R^1_{22})^*R^1_{22} \mid R^1_{32} = (1) \,.\, (0 \mid \varepsilon)^* \,.\, (0 \mid \varepsilon) \mid (1) = (1) \,.\, (0)^*$
$R^2_{33} = R^1_{32}(R^1_{22})^*R^1_{23} \mid R^1_{33} = (1) \,.\, (0 \mid \varepsilon)^* \,.\, \emptyset \mid (0 \mid \varepsilon) = (0 \mid \varepsilon)$

Expressions for k = 3

$R^3_{11} = R^2_{13}(R^2_{33})^*R^2_{31} \mid R^2_{11} = (1) \,.\, (0)^* \,.\, \emptyset \mid (\varepsilon) = \emptyset$
$R^3_{12} = R^2_{13}(R^2_{33})^*R^2_{32} \mid R^2_{12} = (1) \,.\, (0)^* \,.\, (1) \,.\, (0)^* \mid (0^+)$
$R^3_{13} = R^2_{13}(R^2_{33})^*R^2_{33} \mid R^2_{13} = (1) \,.\, (0)^* \,.\, (0 \mid \varepsilon) \mid (1) = (1) \,.\, (0)^*$

$R^3_{21} = R^2_{23}(R^2_{33})^*R^2_{31} \mid R^2_{21} = \emptyset \,.\, (0)^* \,.\, \emptyset \mid \emptyset = \emptyset$
$R^3_{22} = R^2_{23}(R^2_{33})^*R^2_{32} \mid R^2_{22} = \emptyset \,.\, (0)^* \,.\, (1) \,.\, (0)^* \mid (0^*) = (0^*)$
$R^3_{23} = R^2_{23}(R^2_{33})^*R^2_{33} \mid R^2_{23} = \emptyset \,.\, (0)^* \,.\, (0 \mid \varepsilon) \mid \emptyset = \emptyset$

$R^3_{31} = R^2_{33}(R^2_{33})^*R^2_{31} \mid R^2_{31} = (0 \mid \varepsilon) \,.\, (0)^* \,.\, \emptyset \mid \emptyset = \emptyset$
$R^3_{32} = R^2_{33}(R^2_{33})^*R^2_{32} \mid R^2_{32} = (0 \mid \varepsilon) \,.\, (0)^* \,.\, (1) \,.\, (0)^* \mid (1) \,.\, (0)^* = (0)^* \,.\, (1) \,.\, (0)^*$
$R^3_{33} = R^2_{33}(R^2_{33})^*R^2_{33} \mid R^2_{33} = (0 \mid \varepsilon) \,.\, (0)^* \,.\, (0 \mid \varepsilon) \mid (0 \mid \varepsilon) = (0)^*$

The language recognized by this DFA can be expressed by the regular expression R^3_{13}.

$L = R^3_{13} = (1) \,.\, (0 \mid \varepsilon)^* \,.\, (0 \mid \varepsilon) = (1) \,.\, (0 \mid \varepsilon)^* = 1.0^*$

b. Describe succintely what are the words accepted by this DFA?
This automaton recognizes all binaries strings that denote integers that are non-negative powers of two and as such can be described by the compact regular expression: 10^*.

Problem 24

In this problem, we consider the translation of strings with integer values into a comma-separated-value (csv) format. The input string consists of a sequence of one or more letters or numerical digits (forming an identifier) followed by one or more blank space characters to which we have a set of three integers separated by the '-' character and terminated by a 'newline' ('\n') character. The difficulty arises from the fact that the first integer can be a negative integer and thus begins with the '-' character but the two subsequent integers are assumed to be positive. The output string has all '-' character, except the first, replaced by an opening '(', ':' and a ')' character respectively. Below you can find a couple of examples of the intended translation scheme where the '\n' stands for the new-line character:

Input	Output
`F01 -1-0-1'\n'`	`F01, (-1:0), 1'\n'`
`DX1 1-0-3'\n'`	`DX1, (1:0), 3'\n'`

String that do not comply with the structured described are not to be translated and the resulting translation generates an error condition. We can model this condition by having the translation program emit an error message as described in more detail in section d).

For this translation scheme, answer the following questions:

a) Derive a regular expression RE over the ASCII character alphabet that captures valid strings and thus invalid strings.
b) Convert the RE developed in a) into an NFA using the Thompson construction.
c) Convert the NFA derived in section b) to a DFA using the subset construction. Show the mapping between the states in the NFA and the resulting DFA.
d) Minimize the DFA derived in section c) (or show it is already minimal) using the iterative refinement algorithm described in class.
e) Using the C code skeleton described in class for the implementation of DFA with a table, implement a C code function that can recognize valid strings matching the RE derived in a). Whenever an error condition is detected, your implementation should print the 'error' string.

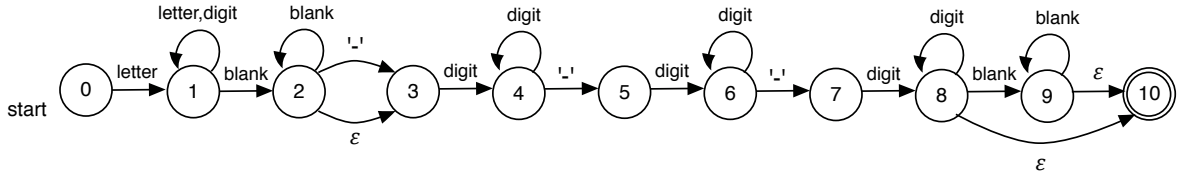
Solution:

a) A regular expression (not necessarily unique) could be described as shown below where *letter* denotes an alphabetic character and *digit* a numerical digit between 0 and 9. Also, *blank* denotes a space character, *int* denotes a string representing an integer (both positive or negative) and *posint* denotes a string exclusively representing a positive integer. Lastly, '.' denotes regular expression concatenation and '|' alternation.

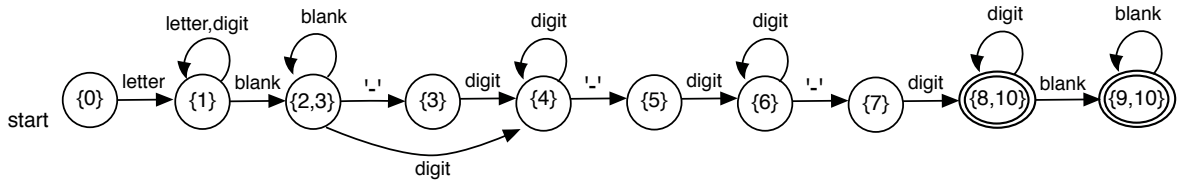
RE = {letter}.({letter} | {digit})*.{blank}+.{int}'-'{posint}'-'{posint}.{blank}*. '\n'

letter = ('a', 'b', … , 'z')
digit = ('0', '1', … , '9')
posint = {digit}+
int = ('-' | ε).{digit}+

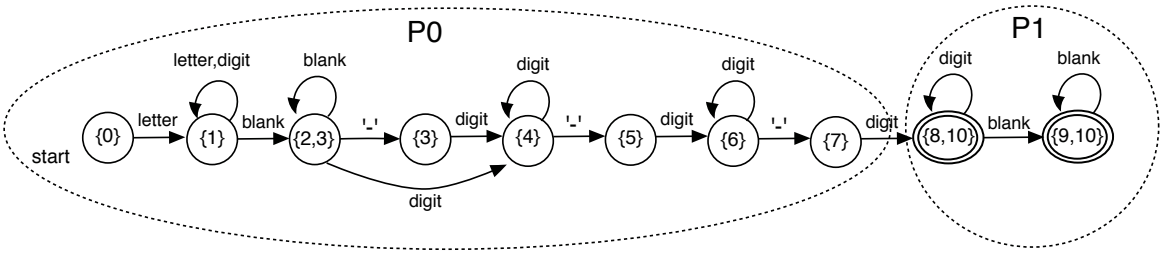
b) A simplified version of the NFA resulting from the use of the Thompson construction for the RE described in a) above is shown below. For compactness, we summarized multiple identical state transitions on various characters, such as transitions on all alphabetic characters as a single state transition. The start state is the state labeled 0. All omitted transitions implicitly denote a transition to an error trap state, say S_{error}.



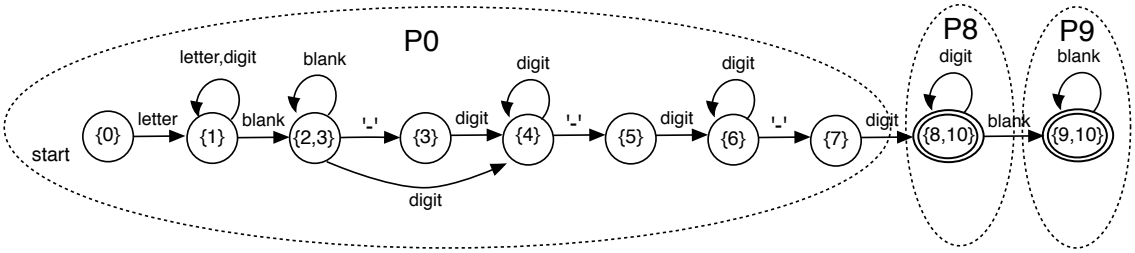
c) The figure below depicts the DFA resulting from the use of the subset construction. Each DFA state is labelled by the corresponding subset of states of the original NFA.



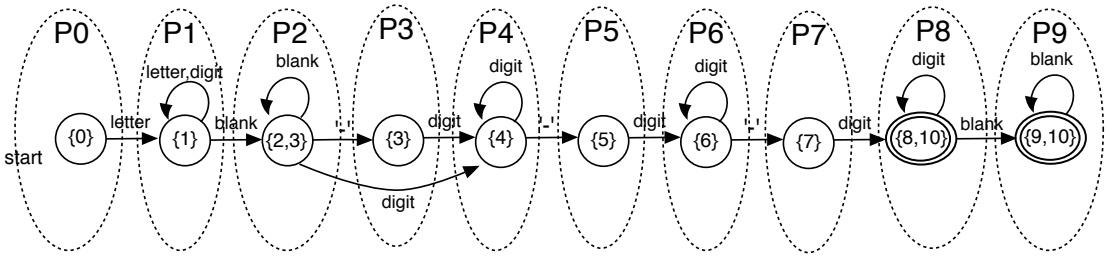
d) The sequence of iterative refinements is shown below.



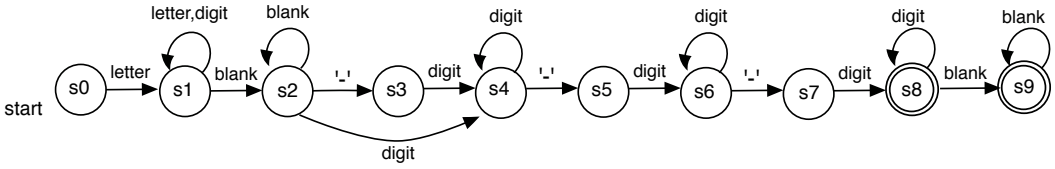
(a) Partition on Accepting vs. Non-Accepting States



(b) Refinement of P1 on digit (note that error state transitions make the difference between states {8,10} and {9,10})



(c) Final refinement.



(d) DFA with relabelled states (trap error state omitted for compactness)

e) The C code is as shown below.

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <ctype.h>

#define error_state -1

int current_state;

int delta(char c){
  switch(current_state){
    case 0:
      if(isalpha(c) != 0){
        printf("%c",c);
        return 1;
      }
      return error_state;
      break;
    case 1:
      if((isalpha(c) != 0) || (isdigit(c) != 0)){
        printf("%c",c);
        return 1;
      }
      if(c == ' '){
        printf("%c, ",c);
        return 2;
      }
      return error_state;
      break;
    case 2:
      if(c == ' '){
        return 2;
      }
      if(c == '-'){
        printf("(-");
        return 3;
      }
      if(isdigit(c) != 0){
        printf("(%c",c);
        return 4;
      }
      return error_state;
      break;
    case 3:
      if(isdigit(c) != 0){
        printf("%c",c);
        return 4;
      }
      return error_state;
      break;
    case 4:
      if(isdigit(c) != 0){
        printf("%c",c);
        return 4;
      }
      if(c == '-'){
        printf(":");
        return 5;
      }
      return error_state;
      break;
   case 5:
      if(isdigit(c) != 0){
        printf("%c",c);
        return 6;
      }
      return error_state;
      break;
    case 6:
      if(isdigit(c) != 0){
        printf("%c",c);
        return 6;
      }
      if(c == '-'){
        printf("), ");
        return 7;
      }
      return error_state;
      break;
    case 7:
      if(isdigit(c) != 0){
        printf("%c",c);
        return 8;
      }
      return error_state;
      break;
    case 8:
      if(isdigit(c) != 0){
        printf("%c",c);
        return 8;
      }
      if(c == ' '){
        printf(" ");
        return 9;
      }
      return error_state;
      break;
    case 9:
      if(c == ' '){
        return 9;
      }
      return error_state;
      break;
    case 10:
      break;
    default:
      return error_state;
  }
  return error_state;
}



int main (int argc, char ** argv){
  char c;

  current_state = 0;
  while(1){
    c = getchar();

    if(c == '\n')
      break;
    current_state = delta(c);

    if(current_state == error_state){
      break;
    }
  }
  if((current_state == 8) || (current_state == 9)){
    printf(" Valid string\n");
  } else {
    printf(" error \n");
  }
  return 0;
}
```