Compilers

Fall 2009

Lexical Analysis

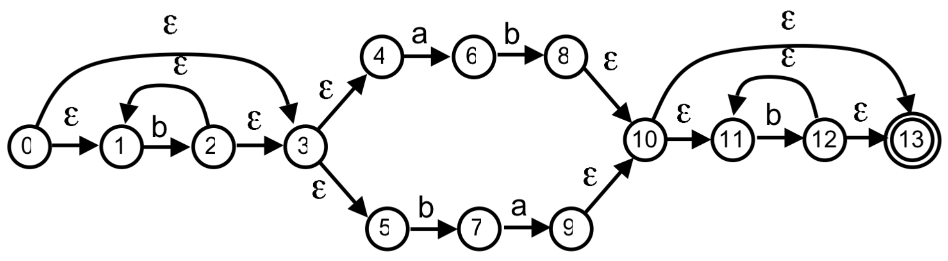
Sample Exercices and Solutions

Prof. Pedro Diniz

Departamento de Engenharia Informática (DEI)
Instituto Superior Técnico / Tagus Park
Av. Prof. Dr. Cavaco Silva
2780-990 Porto Salvo
Portugal

Problem 1: Considering the alphabet Σ = {a,b} construct 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 simplifcation 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 state labeled 0.



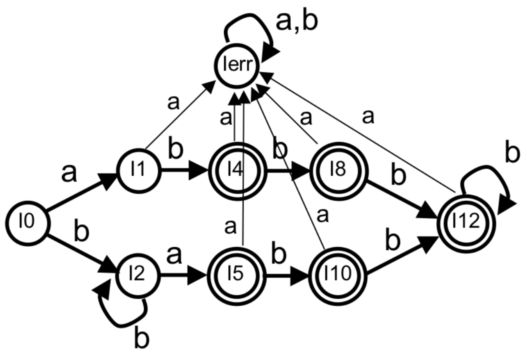
The word w = "abbb" belongs to the language generated by this RE because there is a path from the start state 0 to the accepting state 13 that spells out the word, respectively, 0,3,4,6,8,10,11,12,11,12,13.

Problem 2: Starting from the NFA you constructed 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 NFA for the same input string w.

Solution: Using the ε-closure and DFAedge computation as described in class, we have the following mappings of set of states of the NFA and 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) = {9, 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 ({9, 10, 11, 13}, a) = Ierr
I8 = DFAedge(I4,b) = ε-closure ({9, 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} = I12
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
I15 = DFAedge(I12,a) = ε-closure ({11, 12, 13}, a) = Ierr
I16 = DFAedge(I12,b) = ε-closure ({11, 12, 13}, b) = I12

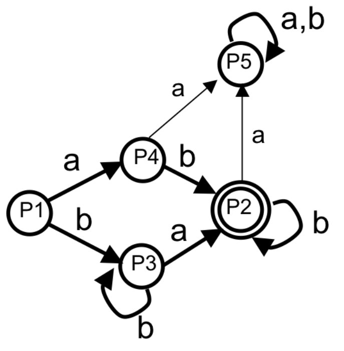
Effectively there are only 9 states as shown in the DFA below and this is clearly not the minimal DFA that can recognize this regular expression.



For the input sentence w = "abbb" we would reach the state I12, 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, I12}. 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.

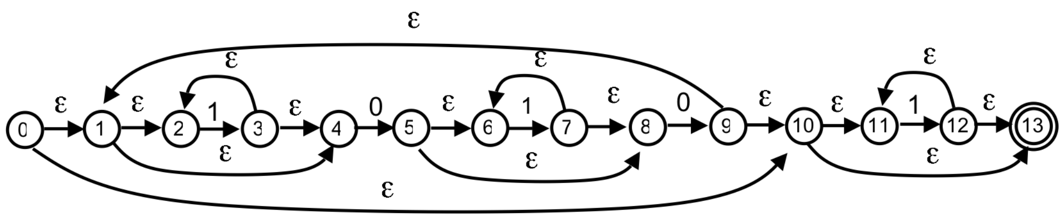


Again on the input string w = "abbb" the DFA would traverse P1, P4, P2 and loop in P2 therefore accepting the string.

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.

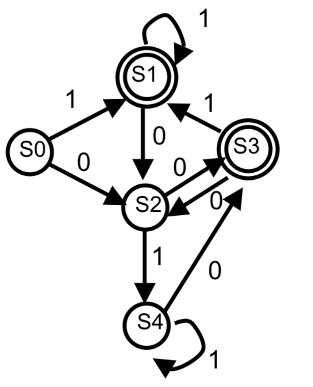
Problem 4: Considering the alphabet Σ = {0,1} construct a Non-Deterministic-Finite Automaton (NFA) using the Thompson construction that is able to recognize the sentences generated by the regular expression (1*01*0)*1*. Does the sentence w = "1010" belong to the language generated by this regular expression? Justify.

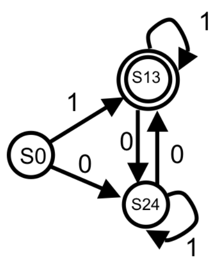
Solution: The NFA is as shown below and where the start state is state labeled 0.



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

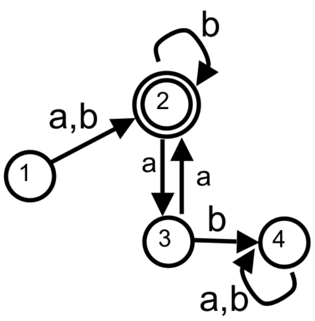
This results in the DFA shown below with starting state S0.





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

Problem 5: Consider the DFA below with starting state 1 and accepting state 2:



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

Solution: Before engaging in a very long sequence of concatenation and symbolic operations we make the observation thaty state 3 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, hat is after the first intialization step we will only have to iterate for k = 1 and k = 2. In addition we are not interested in finding any regular expressions that go from the starting state 0 to the 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

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

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

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

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

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

$R^1_{13} = R^0_{11} (R^0_{11})^*R^0_{13} \mid R^0_{13} = (a).(\varepsilon|b)^*.(a)$

$R^1_{33} = R^0_{31} (R^0_{11})^*R^0_{13} \mid R^0_{33} = (a|b).(b)^*.(a)$

Expressions for k = 2

$R^2_{11} = R^1_{12} (R^1_{22})^*R^1_{21} \mid R^1_{11} = R^1_{11} = (\varepsilon)$

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

$R^2_{23} = R^1_{22} (R^1_{22})^*R^1_{23} \mid R^1_{23} = (\varepsilon|b)^*.(a).((a|b).(b)^*.(a))^*. (\varepsilon|b)^* . (a) \mid (\varepsilon|b)^* . (a)$

$R^2_{32} = R^1_{32} (R^1_{22})^*R^1_{22} \mid R^1_{32} = (a|b).(b)^*.(a).((a|b).(b)^*.(a))^*. (a).(\varepsilon|b)^* \mid (a).(\varepsilon|b)^*$

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

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

$R^2_{33} = R^1_{32} (R^1_{22})^*R^1_{23} \mid R^1_{33} = (a|b).(b)^*.(a) . ((a|b).(b)^*.(a))^* . (a|b).(b)^*.(a) \mid (a|b).(b)^*.(a)$

Clearly these expressions have a lot of redundant terms. In fact we are only interested in R^2_{12} since the starting state is state 0 and the only accepting state is state 1. As such the only regular expression of interest is $R^2_{12} = R^1_{12} (R^1_{22})^*R^1_{22} \mid R^1_{12} = (a).(\varepsilon|b)^*.(a) . ((a|b).(b)^*.(a))^* . (a).(\varepsilon|b)^* \mid (a|b)(\varepsilon|b)+$

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 1 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^2_{01} above.

Problem 6: 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*" preffix to account for any possible number of leading "b's" before the first "a". The single "a" must have at least one "b" either preceeding 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*

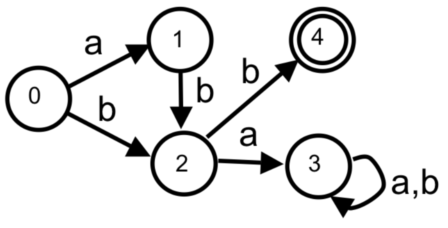
Problem 7: Given an NFA with ε-transitions how do you derive an equivalent NFA without those ε-transitions? Justify.

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 label out of a given state.

Problem 8: 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 start state is the same as in the original NFA. As to the accepting states just add ε-transitions from the orignal accepting states to a new accepting state and label that accepting state as the only accepting state of the new NFA.

Problem 9: Given the Finite Automaton below with starting state 0 and alphabet {a,b} answer the following questions:

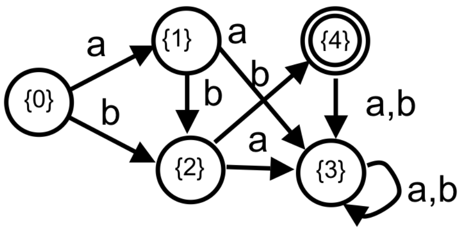


(a) Why is this FA a Non-Deterministic Finite Automaton (NFA)?
(b) Convert this NFA to a DFA using the closure computation.
(c) Minimize the resulting DFA.
(d) What is the Regular Expression matched by this NFA? You are advised not to use the automatic Kleene construction or try to look at the input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

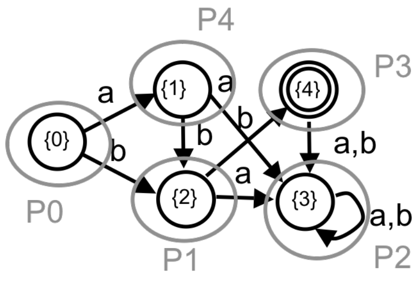
Solution:

(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.



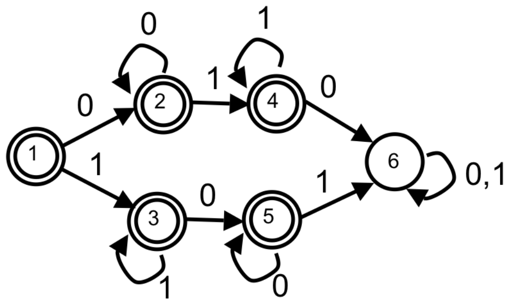
(c) Using the iterative partition refinement we have the following partitions (below left) and the resulting minimized DFA on the right again with the states relabeled. Notice that this is a special case where the refinement algorithm cannot refine any sub-sets of states given that the original DFA was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



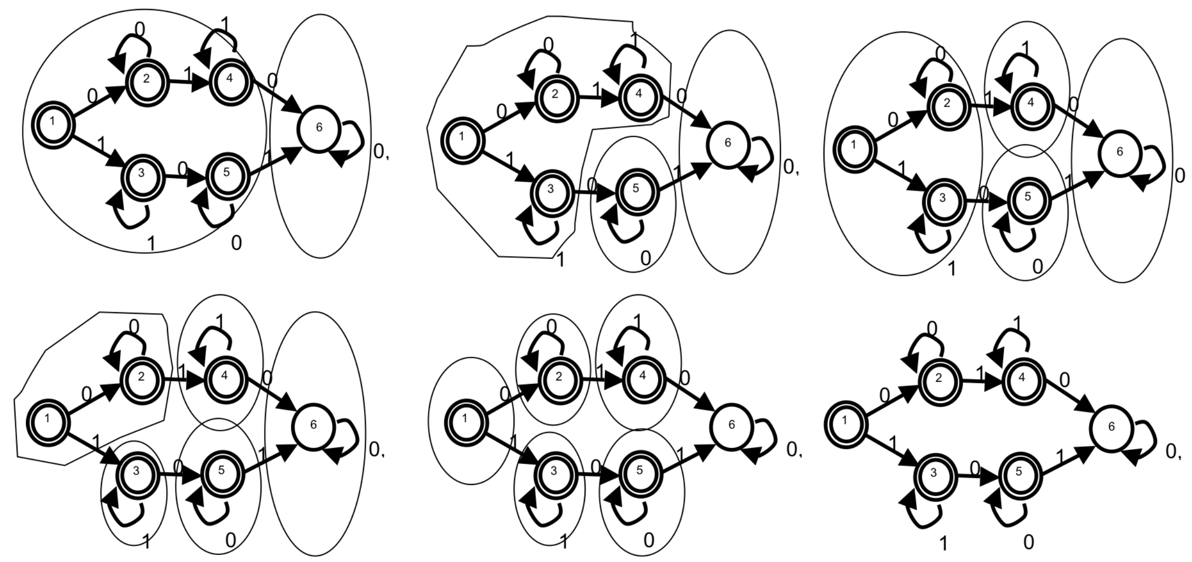
(d) Based on the last DFA the regular expression identified by this FA must have exaclty two consecutive "b" symbol possibly preceeded by a single "a" symbol. For instance, the string "abb" is in the language but the string "abba" or "ba" are not. The regular expression can be denoted by a*bb.

Problem 10: 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.



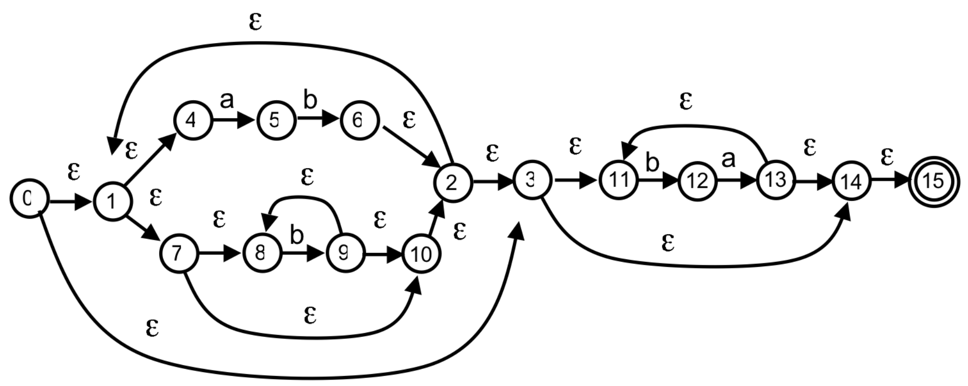
Using the iterative refinements algorithm for DFA minimization indeed confirms that this is the smallest possible DFA. The sequence of refinements is shown below.



Problem 11: Given the regular expression RE = (ab|b*)*(ba*) over the alphabet Σ = {a,b} do the following transformations:

a. construct a NFA using the Thompson construction capabable 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 "abababaa' 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 thius facilitates the construction of the equivalent DFA.

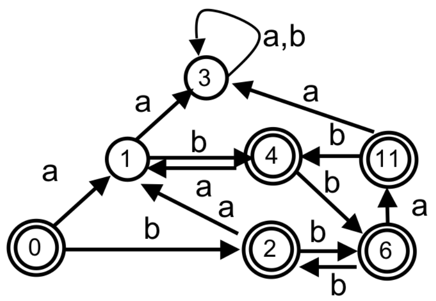


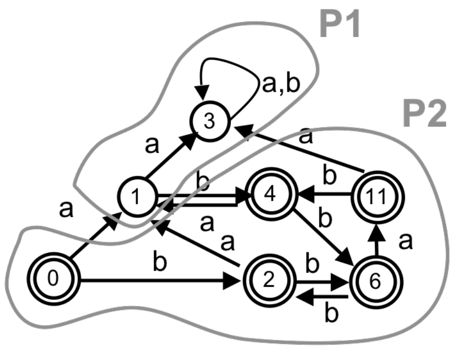
Applying the subset construction we have the following states

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, 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
S5 = DFAedge(S2,a) = ε-closure (goto(S2, a)) = {5} = S1
S6 = DFAedge(S2,b) = ε-closure (goto(S2, b)) = {1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 14, 15} – final
S7 = DFAedge(S3,a) = ε-closure (goto(S3, a)) = {} = S3
S8 = DFAedge(S3,b) = ε-closure (goto(S3, b)) = {} = S3
S9 = DFAedge(S4,a) = ε-closure (goto(S4, a)) = {5} = S1
S10 = DFAedge(S4,b) = ε-closure (goto(S4, b)) = {1, 2, 3, 4, 5, 7, 8, 8, 10, 11, 12, 14, 15} = S6
S11 = DFAedge(S6,a) = ε-closure (goto(S6, a)) = {5, 11, 13, 14, 15} – final
S12 = DFAedge(S6,b) = ε-closure (goto(S6, b)) = {1, 2, 3, 4, 7, 8, 9, 10, 11, 14, 15} = S2
S13 = DFAedge(S11,a) = ε-closure (goto(S11, a)) = {} = S3
S14 = DFAedge(S11,b) = ε-closure (goto(S11, b)) = {1, 2, 3, 4, 6, 7, 8, 10, 11, 14, 15}= S4

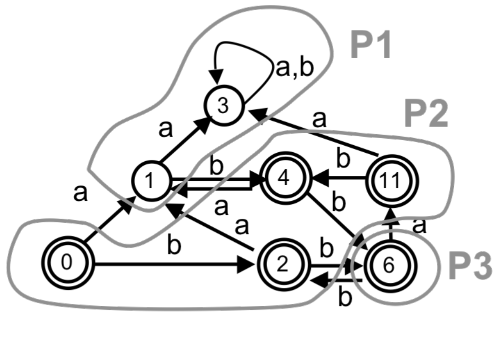
So there are a total of 7 states, S0, S1, S2, S3, S4, S6, S11 and the corresponding DFA is as shown by the table below (bold states are final states) and then in the figure below the table.

	Next State	
Current State	*a*	*b*
S0	S1	S2
S1	S3	S4
S2	S1	S6
S3	S3	S3
S4	S1	S6
S6	S11	S2
S11	S3	S4

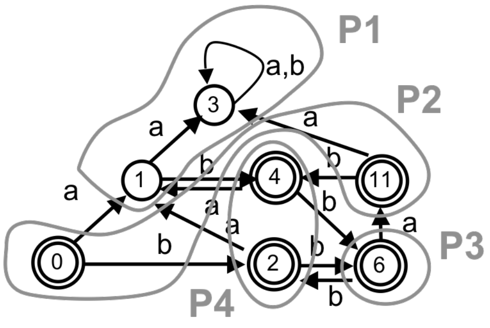




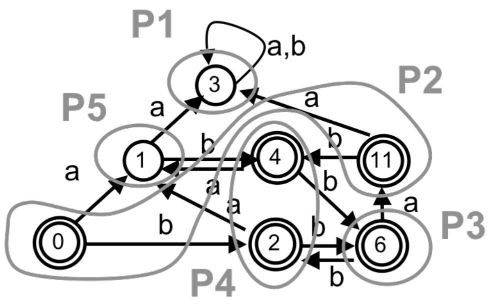
initial refinement on acceptance criteria



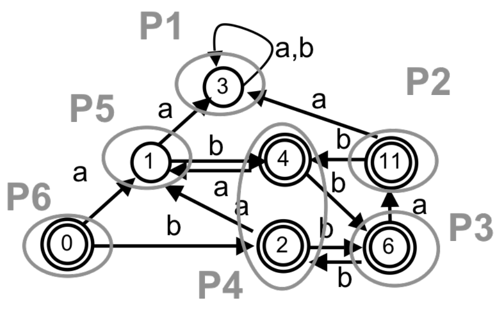
refinement of P2 on a



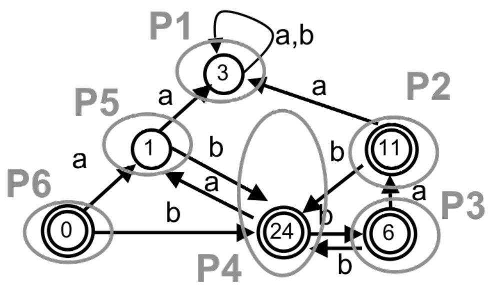
refinement of P2 on b



refinement of P1 on b

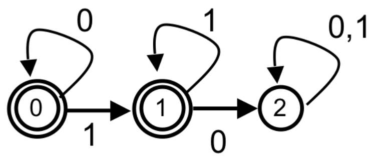


refinement of P2 on a



final refinement with 6 states

Problem 12: 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 with a preffix of zero or more consecutive 0s.