CS3331 - Assignment 1 - 2018

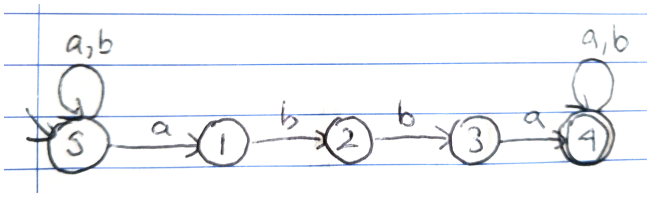
Regular Languages

Due: Tuesday, Oct 16, 2018 (Latest to submit: Friday, Oct 19)

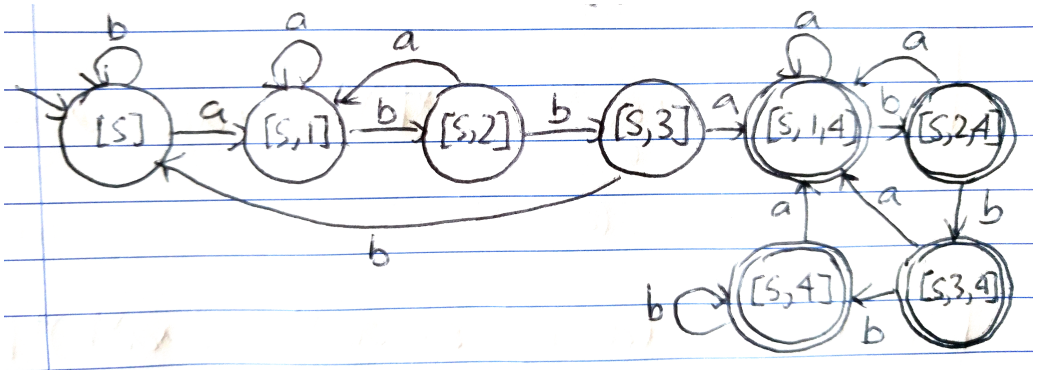
1. (60pt) For each of the following languages, either prove it is not regular or do the following:
   - construct a NDFSM for it
   - convert the NDFSM into a regular expression (horrible regular expressions from JFLAP are accepted only when no obvious ones can be found)
   - convert the NDFSM into a DFSM (Note that you do not have to include trap/dead states)
   - minimize the DFSM

   (a) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* : w$ has `abba` as a substring$\}$.

   For this question we need a machine that will accept provided that `abbb` appears somewhere in the string. This is accomplished with $\Sigma^*\mathtt{abbb}\Sigma^*$, which can be converted into the following NDFSM:



   Since this is non-deterministic, we need to run ndsfmtodfsm to convert it into a deterministic machine:

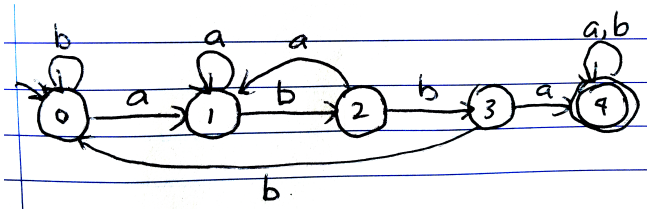


   To minimize it using minDFSM we will rename the states such that the first 6 states on the top row of the deterministic machine $1-6$ in order from left to right, and the last two states 6 and 7 from left to right. Then, applying the minDFSM we get the following

sequence of sets of equivalence classes:

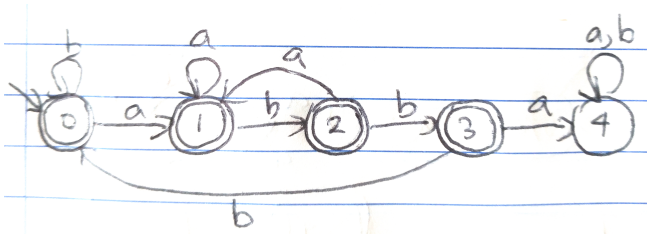
$$\equiv_0 = \{[0,1,2,3],[4,5,6,7]\}$$
$$\equiv_1 = \{[0,1,2],[3],[4,5,6,7]\}$$
$$\equiv_2 = \{[0,1],[2],[3],[4,5,6,7]\}$$
$$\equiv_3 = \{[0],[1],[2],[3],[4,5,6,7]\}$$
$$\equiv_4 = \{[0],[1],[2],[3],[4,5,6,7]\}$$

Letting $0 =_{\text{df}} [0]$**,** $1 =_{\text{df}} [1]$**,** $2 =_{\text{df}} [2]$**,** $3 =_{\text{df}} [3]$**,** $4 =_{\text{df}} [4,5,6,7]$ **this gives the following minimal machine:**



(b) $\{w \in \{\mathtt{a},\mathtt{b}\}^* : w$ does not have `abba` as a substring$\}$.

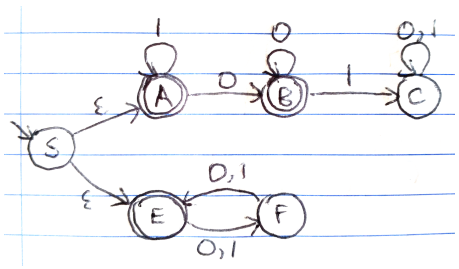
For this case we could design a NDFSM to directly detect those strings that do not contain `abba`**. For simplicity, however, since we can take the complement of the minimal machine from part (a) (by reversing the accepting and non-accepting states), and this is also a NDFSM that accepts the strings that do not contain** `abba`**, we will start with this machine:**



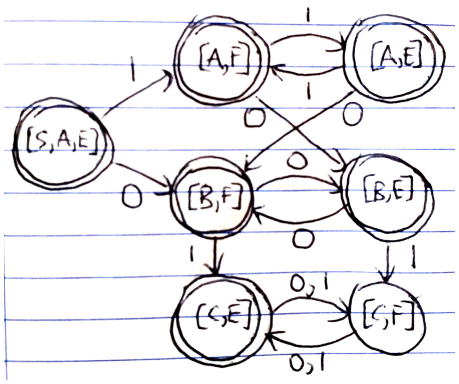
This machine is already deterministic and minimal. The language accepted by this machine is generated by the regular expression
$(\mathtt{b}^* \cup (\mathtt{a}^+\mathtt{b})^+\mathtt{bb})^*(\varepsilon \cup \mathtt{a}^+ \cup (\mathtt{a}^+\mathtt{b})^+(\varepsilon \cup \mathtt{a}^+) \cup (\mathtt{a}^+\mathtt{b})^+\mathtt{b})$**.**

(c) $\{w \in \{\mathtt{0},\mathtt{1}\}^* :$ if w contains the substring `01` then $|w|$ is even$\}$.

We start with the observation that the condition "if w **contains the substring** `01` **then** $|w|$ **is even" is equivalent to "**w **does not contain the substring** `01` **or** $|w|$ **is even." Thus, one strategy is to construct a NDFSM that combines two machines, one that accepts the language that does not contain** `01` **(complement of a DFSM that accepts strings containing** `01`**), and one that accepts** $|w|$ **is even, and join them with** ε**-transitions from a start state** S**:**



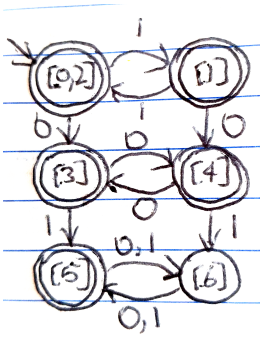
From this machine we can extract the regular expression $1^*(\varepsilon \cup 0^+) \cup ((0 \cup 1)(0 \cup 1))^*$ Converting to a DFSM yields:



Renaming the states 0 for the start state, then 1 and 2 for the top two remaining states, 3 and 4 for the middle two, and 5 and 6 for the bottom two (all from left to right), and running the minimation algorithm we get the sequence:

$$\equiv_0 = \{[6], [0, 1, 2, 3, 4, 5]\}$$
$$\equiv_1 = \{[6], [0, 1, 2, 3], [4, 5]\}$$
$$\equiv_2 = \{[6], [0, 2], [1], [2], [3], [4], [5]\}$$

This yields the machine:

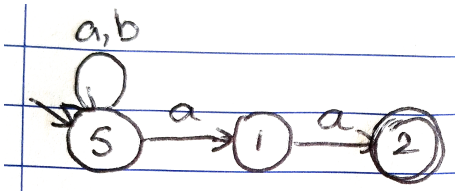


(d) $\{w \in \{\mathtt{0},\mathtt{1}\}^* : w = w^{\mathrm{R}}\}$.

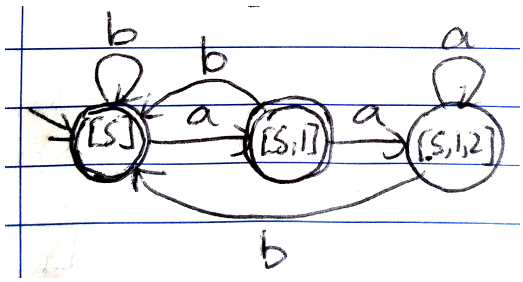
This language, call it L, is not regular, which we prove using a pumping theorem argument (for this language I give a more formal style of proof, this careful and argument is not necessary for you to give). Assume that L is regular. Let $w = \mathtt{a}^k\mathtt{b}^k\mathtt{b}^k\mathtt{a}^k$, then $|w| \geq k$. So, by the pumping theorem there are x, y with $w = xyz$, such that $|xy| \leq k$, $y \neq \varepsilon$ and for any $q \geq 0$, $xy^qz \in L$. Then $y = a^p$ for some $0 < p \leq k$ ($p > 0$ because $y \neq \varepsilon$). If we set $q = 0$ (pump out) then we get $w' = xz = \mathtt{a}^{k-p}\mathtt{b}^k\mathtt{b}^k\mathtt{a}^k$, but this is not in L because $w'^{\mathbf{R}}$ has a different number of initial $\mathtt{a}$'s than w', so $w' \neq w'^{\mathbf{R}}$. Since this contradicts $xy^qz \in L$ for all q, we conclude that our assumption that L is regular is false.

(e) $\{w \in \{\mathtt{a},\mathtt{b}\}^* : w$ does not end in $\mathtt{aa}\}$.

For this problem, it is most natural to start with the NDFSM that accepts strings that end in $\mathtt{aa}$:



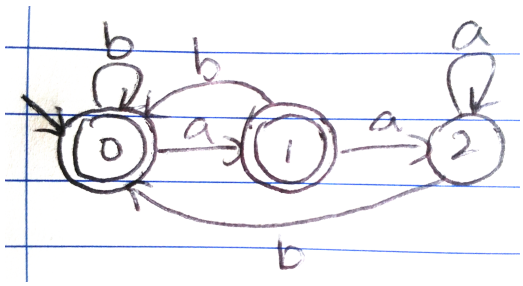
To complement this machine to make a machine that accepts strings that do not end in $\mathtt{aa}$, we must first convert it to a DFSM and then reverse the accepting and non-accepting states:



This machine is now our NDFSM that accepts strings that do not end in $\mathtt{aa}$, because a DFSM is *a fortiori* a NDFSM. This machine is equivalent to the regular expression $(\mathtt{b} \cup \mathtt{ab} \cup \mathtt{aa}^+\mathtt{b})^*(\varepsilon \cup \mathtt{a})$. This machine is already minimal, as we find by numbering the states 0–2 from left to right and producing the sequence:

$$\equiv_0 = \{[2], [0, 1]\}$$
$$\equiv_1 = \{[0], [1], [2]\}$$

For completeness, the minimal machine is therefore:

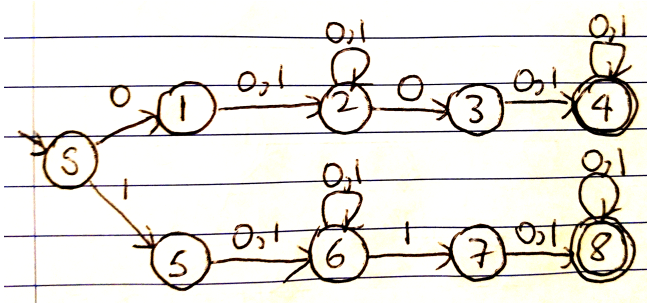


(f) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* : w$ contains exactly three more $\mathtt{a}$'s than $\mathtt{b}$'s$\}$.

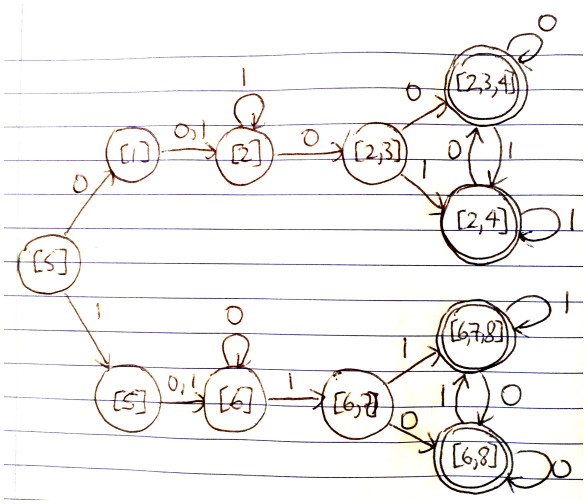
This language is not regular, which can be shown by a pumping theorem argument. Let $w = xyz = \mathtt{a}^{k+3}\mathtt{b}^k$, so that $y = \mathtt{a}^p$ for some $0 < p \leq k$. Then pump in once. The resulting string $w' = \mathtt{a}^{k+p+3}\mathtt{b}^k$ no longer has exactly three more $\mathtt{a}$'s than $\mathtt{b}$'s and is therefore not in the language.

(g) $\{w = yxyz : x, y, z \in \{\mathtt{0}, \mathtt{1}\}^+\}$.

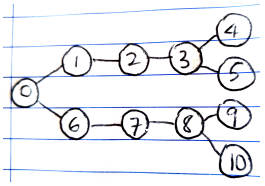
Even though the repeated y's are suggestive of non-regularity, the key in this case is that it is possible to choose y to be a single character, $\mathtt{0}$ or $\mathtt{1}$. This includes the cases where the repeated part can be a more complex string, such as $\mathtt{01100111}$, where $\mathtt{011}$ is repeated, because we can still take y be be the first character of the repeated part, i.e., $\mathtt{0}$, and let x and z absorb the rest of the string, i.e., $x = \mathtt{110}$ and $z = \mathtt{111}$. Thus, the language is just that generated by the regular expression $(\mathtt{0}(\mathtt{0} \cup \mathtt{1})^+\mathtt{0}(\mathtt{0} \cup \mathtt{1})^+) \cup (\mathtt{1}(\mathtt{0} \cup \mathtt{1})^+\mathtt{1}(\mathtt{0} \cup \mathtt{1})^+)$. This is accepted by the following NDFSM:



Converting it to a DFSM yields the machine:



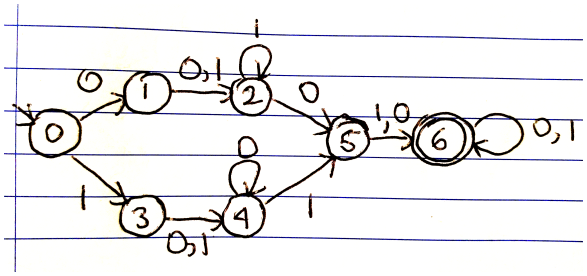
To minimize the machine we will relabel the states according to the pattern:



Minimizing then yields the sequence:

$$\equiv_0 = \{[0,1,2,3,6,7,8],[4,5,9,10]\}$$
$$\equiv_1 = \{[0,1,2,6,7],[3,8],[4,5,9,10]\}$$
$$\equiv_2 = \{[0,1,6],[2],[7],[3,8],[4,5,9,10]\}$$
$$\equiv_3 = \{[0],[1],[6],[2],[7],[3,8],[4,5,9,10]\}$$
$$\equiv_4 = \{[0],[1],[2],[6],[7],[3,8],[4,5,9,10]\}$$

This gives rise to the minimal machine (relabeling the states again according to the order in $\equiv_4$):



(h) $\{w \in \Sigma^* : w$ is a Java comment$\}$, where Σ is the ASCII alphabet and Java comments are of two types: `/* ... comment ... */`; `// ... comment ... \n`.

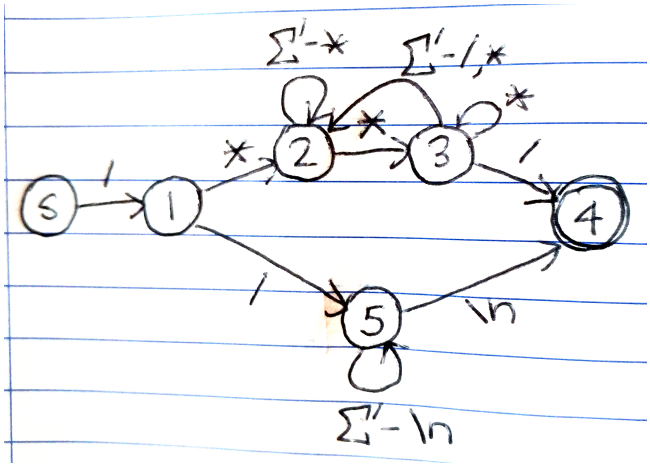
For this question, we begin with the observation that a Java comment cannot contain characters from the entire ASCII alphabet, because control characters, except `\n`, which is the newline character, cannot appear in comments. Thus, we let Σ' be the subset of Σ consisting of printing characters together with `\n`. All other control characters will lead immediately to a dead state, which we omit.

To create a machine to accept Java comments we observe that all comments start with a single /, followed immediately by either a second / or a $*$ for the two kinds of comments.

For the case where the second character is /, the interior of the comment can contain anything other than a newline character, and once the newline character appears we go to an accepting state.

For the case where the second character is $*$, the comment must end with a $*$/, so we need to track whether a second $*$ appears, and if so, whether a / appears directly after it (indicating the end of the comment and a move to an accepting state). Otherwise we check if we read another $*$, in which case a single / still takes us to the accepting state, or anything else and we go back to looking for an appearance of $*$.

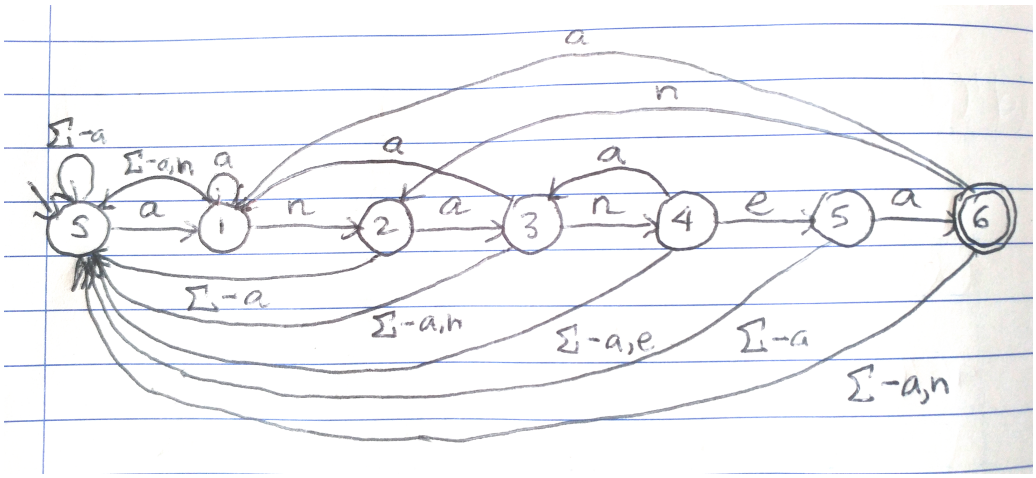
Then by recognizing that nothing can come after the accepting state for either type of comment, we only need one accepting state, with no transitions out of it. This gives rise to the following machine, which is already minimal:



Converting this to a regular expression, which is much easier to do with JFLAP, we obtain $[/*(\Sigma' - *)^**(*\cup(\Sigma' - \{/, \backslash n\})(\Sigma' - *)^**)^*/] \cup [//(\Sigma' - \backslash n)^*\backslash n]$, where I have used square brackets to mark the regular expressions for the two types of comments. Note the difference between the character $*$ and the Kleene star *.

2. (20pt) The Pattern Searching problem is: Given two strings $p, T \in \Sigma^*$ (the pattern and the text), find all occurrences of p in T. It can be solved in time $\mathcal{O}(|T|)$ by constructing a DFSM for the language $\Sigma^* p$ and then run the text T through it; every time the machine is in an accepting state, we report the end of an occurrence of the pattern. Construct the minimal DFSM for the pattern $p =$ `ananea`. (Show also your NDFSM.)

For this problem, once we have recognized the pattern of machines that accept patterns it is possible to construct the DFSM directly, which is what we do here. It is also possible to work with a NDFSM that contains a path for strings that end with `ananea`, which we would then convert to a DFSM and minimize. The result is the machine:



3. (20pt) Show that the following problem is decidable: Given a FSM M and a regular expression α, it is true that both $L(M)$ and $L(\alpha)$ are finite and α generates at least one string that M does not accept?

Given an FSM M and a regular expression α, use the procedure regextofsm(α : regex) **to produce an FSM M' such that $L(M') = L(\alpha)$. Then, use the procedure** infiniteFSM(M : **FSM**) **to check the sizes of $L(M)$ and $L(M')$.**

If infiniteFSM(M) **returns** true**, meaning $L(M)$ is infinite, then return** false**. If** infiniteFSM(M') **returns** true**, meaning that $L(M') = L(\alpha)$ is infinite, then return** false**. Otherwise, continue.**

Since M or M' can be nondeterministic they may not halt on every input, so we need to use the procedure ndfsmtodfsm(M : **NDSFM**) **or** ndfsmsimulate(M : **NDFSM**, w : **string**) **to ensure that the machines halt.**

Let $M'' =$ ndfsmtodfsm(M') **and let k be the number of states of M''. Run M'' on all of the**

strings of length less than k. If M'' accepts a string w, meaning that α generates w, run ndfsmsimulate(M, w)**; if it rejects w, return** true**, meaning that α generates a string that M does not accept. If** ndfsmsimulate(M, w) **accepts all the strings w accepted by M'', return** false**.**