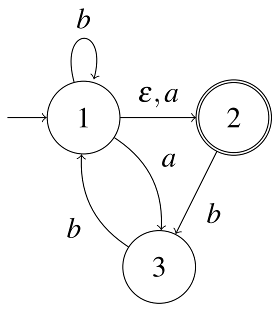
CSE 103 Computational Models
Fall 2021 Prof. Fremont HW 3

Due October 20 at 11:59pm

(6 questions, 230 points total)

1. (40 pts.) Conversion of NFAs to DFAs
Consider the following NFA N over the alphabet $\Sigma = \{a,b\}$:



(a) *(30 pts.)* Use the construction we described in class to build a DFA D equivalent to N. Draw D as a graph, omitting any states which are not reachable from the start state.

Solution: Recall that the states of D will be subsets of states of N, with the start state being the ε-closure of the start state of N, i.e. $E(\{1\}) = \{1,2\}$. On input a, the transition from this state is defined:

$$\delta_D(\{1,2\},a) = \bigcup_{q_N \in \{1,2\}} E(\delta_N(q_n,a)) = E(\delta_N(1,a)) \cup E(\delta_N(2,a)) = E(\{2,3\}) \cup E(\emptyset) = \{2,3\}.$$

Likewise, on input b we have

$$\delta_D(\{1,2\},b) = E(\delta_N(1,b)) \cup E(\delta_N(2,b)) = E(\{1\}) \cup E(\{3\}) = \{1,2\} \cup \{3\} = \{1,2,3\}.$$

We've reached two new states: $\{2,3\}$ and $\{1,2,3\}$. Let's look at the state $\{2,3\}$ first. On input a we have

$$\delta_D(\{2,3\},a) = E(\delta_N(2,a)) \cup E(\delta_N(3,a)) = E(\emptyset) \cup E(\emptyset) = \emptyset,$$

while on input b we have

$$\delta_D(\{2,3\},b) = E(\delta_N(2,b)) \cup E(\delta_N(3,b)) = E(\{3\}) \cup E(\{1\}) = \{3\} \cup \{1,2\} = \{1,2,3\}.$$

We've reached one new state, $\emptyset$. Let's first look at the state $\{1,2,3\}$ we reached earlier. On input a we have

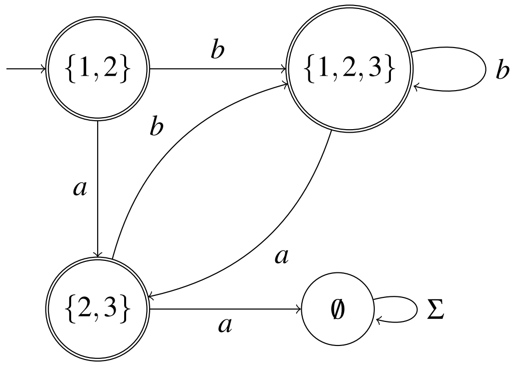
$$\delta_D(\{1,2,3\},a) = E(\delta_N(1,a)) \cup E(\delta_N(2,a)) \cup E(\delta_N(3,a)) = E(\{2,3\}) \cup E(\emptyset) \cup E(\emptyset) = \{2,3\},$$

while on input b we have

$$\delta_D(\{1,2,3\},b) = E(\delta_N(1,b)) \cup E(\delta_N(2,b)) \cup E(\delta_N(3,b)) = E(\{1,3\}) \cup E(\{3\}) \cup E(\{1\}) = \{1,2,3\}.$$

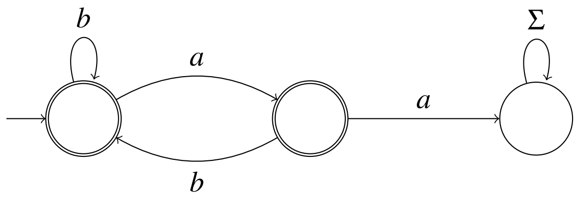
Neither of these transitions reaches new states.

Continuing with state $\emptyset$, on any input $s \in \Sigma$ we have $\delta_D(\emptyset, s) = \cup_{q_N \in \emptyset} E(\delta_N(q_n, s)) = \emptyset$. We have not reached any new states, so the only states reachable from the start state are $\{1,2\}$, $\{2,3\}$, $\{1,2,3\}$, and $\emptyset$. The first three of these contain an accepting state of N, namely 2, and so are accepting states of D. Drawing this as a graph, we get:



(b) *(10 pts.)* Even dropping unreachable states, D is not the smallest DFA equivalent to N. Draw another DFA D' which is equivalent to N and has as few states as possible (you don't need to prove this).

Solution: Observe that the only way for D above to reject is by reaching the state $\emptyset$, since every other state is accepting. The only way to reach $\emptyset$ is to have two as in a row, so the language of D consists of all strings over Σ without two as in a row. We can define another DFA D' recognizing this language with only 3 states:



2. (45 pts.) Intersecting automata

We showed in class that if L_1 and L_2 are regular languages, then so is $L_1 \cap L_2$, since $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ and regular languages are closed under union and complement.

(a) *(10 pts.)* Suppose we have DFAs D_1 and D_2 for L_1 and L_2 with n and m states respectively. Describe the sequence of transformations you would use to turn these into a DFA for $\overline{\overline{L_1} \cup \overline{L_2}}$ (and so for $L_1 \cap L_2$), stating after each step how many states the resulting automaton has.

Solution: First we'll use the DFA complement construction to get DFAs M_1 and M_2 such that $L(M_1) = \overline{L_1}$ and $L(M_2) = \overline{L_2}$. Since the complement construction just swaps accepting/rejecting states, $|M_1| = n$ and $|M_2| = m$ (where the notation $|M|$ measures the size of M as its number of states). Applying the NFA union construction to M_1 and M_2 yields an NFA N with $n+m+1$ states, since we just add one new start state. Finally, we need to complement N, but our complement construction only works for DFAs, so we first have to convert N to an equivalent DFA D. The subset construction produces one DFA state for every possible subset of NFA states, so $|D| = 2^{|N|} = 2^{n+m+1}$. Now we can complement D, and this does not change the number of states, so we will end up with a DFA with 2^{n+m+1} states.

(*Note:* If we had started with NFAs instead of DFAs, we could still use the same procedure, but the complexity would be even worse: in order to do the first complement operations we would need to convert the NFAs to DFAs, which would have sizes 2^n and 2^m. So the final DFA would have $2^{2^n+2^m+1}$ states. Fortunately, the much more efficient product construction in part (b) also works for NFAs.)

(b) *(35 pts.)* A more direct way to build a DFA for $L_1 \cap L_2$ is the *product construction*. The idea is to run the DFAs for L_1 and L_2 in parallel: each time we read an input symbol, we update the states of both automata, and we accept only if both automata end in an accepting state. To simulate running both automata in parallel, we use a DFA whose states are labeled by *pairs* of states, one from each automaton; then the transitions are defined by updating each component of the pair independently. Formally, starting with DFAs $D_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $D_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$, the *product automaton D* is defined as follows:

- $Q = Q_1 \times Q_2$ (all pairs with a state from Q_1 and a state from Q_2)
- $\delta((q_1, q_2), s) = (\delta_1(q_1, s), \delta_2(q_2, s))$
- $q_0 = (q_{01}, q_{02})$
- $F = \{(q_1, q_2) \in Q \mid q_1 \in F_1 \text{ and } q_2 \in F_2\}$.

Prove that the product construction works: $L(D) = L(D_1) \cap L(D_2)$.
(*Hint:* Compare the extended transition functions of D, D_1, and D_2.)

Solution: We'll first prove a lemma that $\hat{\delta}((q_1, q_2), w) = (\hat{\delta}_1(q_1, w), \hat{\delta}_2(q_2, w))$ for all $q_1 \in Q_1$, $q_2 \in Q_2$, and $w \in \Sigma^*$ by induction on the length of w. In the base case $|w| = 0$ we have $w = \varepsilon$, and $\hat{\delta}((q_1, q_2), \varepsilon) = (q_1, q_2) = (\hat{\delta}_1(q_1, \varepsilon), \hat{\delta}_2(q_2, \varepsilon))$ as desired. Now suppose the hypothesis holds for all words of length n, and take any $w \in \Sigma^{n+1}$. Then we can write $w = xs$ with $x \in \Sigma^n$ and $s \in \Sigma$, and by hypothesis we have $\hat{\delta}((q_1, q_2), x) = (\hat{\delta}_1(q_1, x), \hat{\delta}_2(q_2, x))$. Then we have

$$\begin{aligned}
\hat{\delta}((q_1, q_2), xs) &= \delta(\hat{\delta}((q_1, q_2), x), s) && \text{(definition of } \hat{\delta}) \\
&= \delta\left((\hat{\delta}_1(q_1, x), \hat{\delta}_2(q_2, x)), s\right) && \text{(inductive hypothesis)} \\
&= (\delta_1(\hat{\delta}_1(q_1, x), s), \delta_2(\hat{\delta}_2(q_2, x), s)) && \text{(definition of } \delta) \\
&= (\hat{\delta}_1(q_1, xs), \hat{\delta}_2(q_2, xs)). && \text{(definitions of } \hat{\delta}_1 \text{ and } \hat{\delta}_2)
\end{aligned}$$

So by induction our lemma holds for all words.

Now recall the lemma from class that D accepts $w \in \Sigma^*$ if and only if $\hat{\delta}(q_0, w) \in F$. So noting that $q_0 = (q_{01}, q_{02})$ and using our result above, we have

$$\begin{aligned}
w \in L(D) &\iff \hat{\delta}((q_{01}, q_{02}), w) \in F && \text{(lemma from class)} \\
&\iff (\hat{\delta}_1(q_{01}, w), \hat{\delta}_2(q_{02}, w)) \in F && \text{(lemma we proved above)} \\
&\iff \hat{\delta}_1(q_{01}, w) \in F_1 \text{ and } \hat{\delta}_2(q_{02}, w) \in F_2 && \text{(definition of } F\text{)} \\
&\iff w \in L(D_1) \text{ and } w \in L(D_2) && \text{(lemma from class, twice)} \\
&\iff w \in L(D_1) \cap L(D_2).
\end{aligned}$$

3. (35 pts.) Reading regular expressions

For each of the following pairs of regular expressions over $\Sigma = \{a, b, c\}$, state whether their languages are equal ($=$), one is a proper subset of the other ($\subset$), or if they are incomparable. If $L(R_1) \subset L(R_2)$, give an example of a string in $L(R_2)$ that is not in $L(R_1)$, or vice versa if $L(R_2) \subset L(R_1)$; if the languages are incomparable, give an example string from each language that is not in the other.

(a) $R_1 = a^*b^*$
$R_2 = (a^*b^*)^*$

Solution: $L(R_1)$ consists of all strings with zero or more as followed by zero or more bs, whereas $L(R_2)$ consists of all strings containing only as and bs. So $L(R_1) \subset L(R_2)$, with for example ba being in $L(R_2)$ but not $L(R_1)$.

(b) $R_1 = (a|b|c)^*$
$R_2 = (a^*b^*c^*)^*$

Solution: Both $L(R_1)$ and $L(R_2)$ consist of all strings over Σ, so $L(R_1) = L(R_2)$.

(c) $R_1 = c^*c$
$R_2 = (cc)^* \mid c(cc)^*$

Solution: R_1 matches any string consisting of 1 or more cs. The two terms of R_2 match strings consisting of even and odd numbers of cs respectively, so R_2 is equivalent to c^*. Therefore $L(R_1) \subset L(R_2)$, with ε being the only string in $L(R_2)$ but not $L(R_1)$.

(d) $R_1 = c^*(a|b|\varepsilon)c^*$
$R_2 = c^*ac^* \mid c^*bc^*$

Solution: We have $R_1 = c^*ac^* \mid c^*bc^* \mid c^*c^* = c^*ac^* \mid c^*bc^* \mid c^*$. Therefore $L(R_2) \subset L(R_1)$, with ε and c being examples of strings in $L(R_1)$ but not $L(R_2)$.

(e) $R_1 = \emptyset b \mid aa^*$
$R_2 = a^*$

Solution: Since $\emptyset$ does not match any string, neither does $\emptyset b$ (which matches only the concatenation of something matching $\emptyset$ and something matching b), so $R_1 = aa^*$. Therefore $L(R_1) \subset L(R_2)$, with ε being the only string in $L(R_2)$ but not in $L(R_1)$.

4. (45 pts.) Writing regular expressions

Write regular expressions for the following languages over $\Sigma = \{a, b, c, \dots, z\}$:

(a) Strings containing the substring *dog* followed eventually (not necessarily immediately) by either the substring *cat* or the substring *rat*.

Solution: We want to match any string with an arbitrary (possibly empty) prefix, then *dog*, then another arbitrary (possibly empty) string, then either *cat* or *rat*, and finally another arbitrary (possibly empty) string. This gives us the regular expression $\Sigma^* dog\, \Sigma^* (cat|rat)\, \Sigma^*$.

(b) Strings containing both x and y, where the first x is followed by a y within 3 symbols (e.g. xy and $xzzy$ are fine but not $xzzzy$).

Solution: To match an x followed by a y within 3 symbols we can use the regular expression $x(y \mid \Sigma y \mid \Sigma\Sigma y)$ (or $x(\varepsilon \mid \Sigma \mid \Sigma\Sigma)y$). To match the *first* x in the string, we can use $(\Sigma - x)^* x$ where $\Sigma - x$ is shorthand for $(a \mid b \mid \dots \mid w \mid y \mid z)$. Putting these together, and allowing an arbitrary suffix, we get the regular expression $(\Sigma - x)^* x(y \mid \Sigma y \mid \Sigma\Sigma y)\Sigma^*$.

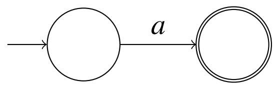
(c) Strings containing an even number of zs.

Solution: First, notice that we can write a regular expression matching any string containing exactly 2 zs by only allowing the symbols before, between, and after the zs to be symbols other than z. Using the notation $\Sigma - z$ for $(a \mid b \mid \dots \mid y)$, this gives us $(\Sigma - z)^* z(\Sigma - z)^* z(\Sigma - z)^*$. Applying the Kleene star to this expression would allow any string which is either empty or contains $2k$ zs for some $k \geq 1$. To allow any string with *no* zs (since 0 is even), we could for example union this with $(\Sigma - z)^*$, obtaining the regular expression $(\Sigma - z)^* \mid [(\Sigma - z)^* z(\Sigma - z)^* z(\Sigma - z)^*]^*$. This is equivalent to the expression $(\Sigma - z)^* [z(\Sigma - z)^* z(\Sigma - z)^*]^*$.

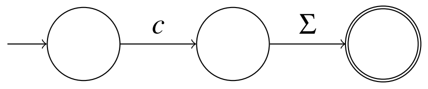
5. (20 pts.) Converting a regular expression to an NFA

Convert the regular expression $(a \mid c\Sigma)^*a$ to an equivalent NFA.

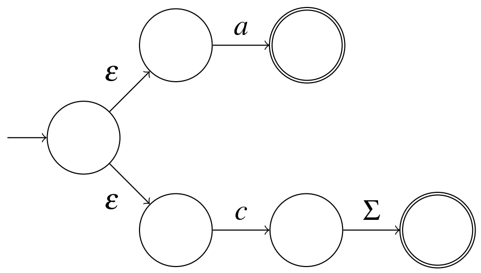
Solution: We will convert the expression from the inside out, applying our NFA constructions for the regular operations. For the primitive subexpression a, we have the NFA



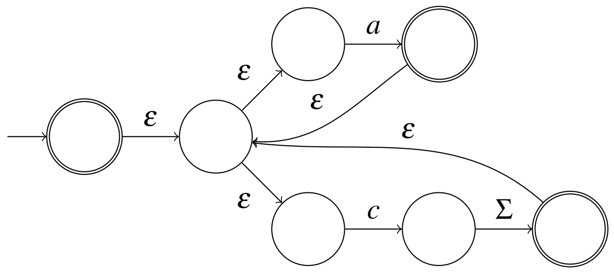
For c and Σ, the NFA is the same except the transition is labeled c or Σ. To get an NFA for $c\Sigma$, we apply the concatenation construction, obtaining the NFA



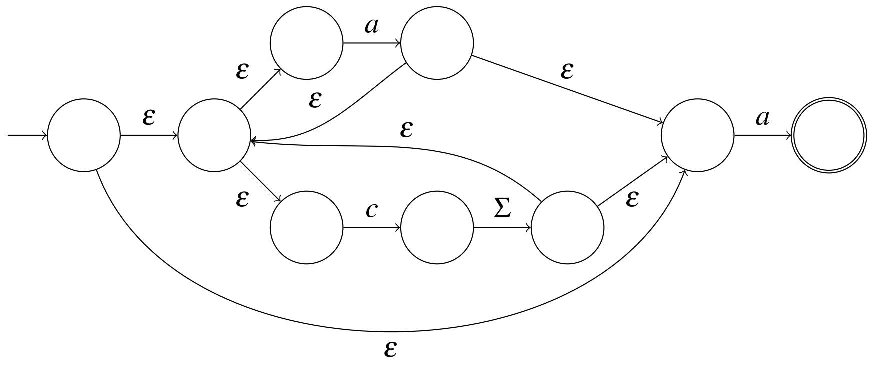
To get an NFA for $a \mid c\Sigma$, we apply the union construction, getting:



To get an NFA for $(a \mid c\Sigma)^*$, we next apply the Kleene star construction, getting:



Finally, we apply the concatenation construction once more to get an NFA for $(a \mid c\Sigma)^*a$:



(Note that this NFA could be significantly simplified: our constructions don't necessarily yield an optimal NFA, but they do give one which is equivalent to the original RE.)

6. (45 pts.) Properties of regular languages
Prove each of the following facts about regular languages. Give a short argument using results from class and/or simple constructions of automata or regular expressions (no need for proofs by induction).

(a) Every finite language (i.e., a language consisting of finitely many strings) is regular.

Solution: First, note that the empty language is regular (consider a DFA with 1 state that rejects everything), and that any language $\{w\}$ consisting of a single word is also regular: we can make a DFA D_w with a chain of states such that each each successive symbol w_i moves us along the chain, any other symbol moves us to a rejecting sink state, and the end of the chain is an accepting state. Then we can build an NFA for any finite language L by taking the DFAs D_w for each $w \in L$ and combining them into a single NFA using the union construction.

Alternate Solution: For any word $w \in \Sigma^*$, the regular expression consisting of the same symbols $w_1 \dots w_{|w|}$ matches only that word. If we enumerate the words of L as $w_1, \dots, w_k$, then $w_1 | \dots | w_k$ is a regular expression for L (if L is empty, then use the expression $\emptyset$ instead).

(b) If I, T, and E are regular languages over Σ, then so is the language

$$\textsc{If-Then-Else}(I,T,E) = \{xy \in \Sigma^* \mid x \in I \to y \in T \text{ and } x \notin I \to y \in E\}.$$

Solution: Strings in $\textsc{If-Then-Else}(I,T,E)$ are of the form xy with either $x \in I$ and $y \in T$ or $x \notin I$ and $y \in E$. So $\textsc{If-Then-Else}(I,T,E) = IT \cup \bar{I}E$. Since I, T, and E are regular, and regular languages are closed under concatenation, union, and complement, this shows that $\textsc{If-Then-Else}(I,T,E)$ is regular.

(c) If $s \in \Sigma$, and L and R are regular languages over Σ, then so is the language $\textsc{Replace}(L,s,R)$ obtained by taking every string in L and replacing each occurrence of the symbol s by any string in R.
(For example, if $L = \{a, aa, aba\}$, $s = a$, and $R = \{c, dd\}$, then $\textsc{Replace}(L,s,R)$ is $\{c, dd, cc, cdd, ddc, dddd, cbc, cbdd, ddbc, ddbdd\}$.)

Solution: Since L and R are regular, there are regular expressions E_L and E_R such that $L(E_L) = L$ and $L(E_R) = R$. Replacing every occurrence of the symbol s in E_L by a copy of E_R gives a regular expression for $\textsc{Replace}(L,s,R)$. Since regular expressions define regular languages, $\textsc{Replace}(L,s,R)$ is regular.

Alternate Solution: Since L and R are regular, there are NFAs N_L and N_R for them. Take every transition on the symbol s in N_L and replace it with a copy of N_R; more precisely, for every transition in N_L from state x to state y on input s:

(1) delete that transition;
(2) create a new copy N of N_R;
(3) add ε-transitions from x to the start state of N and from every accepting state of N to y;
(4) make every state of N non-accepting.

Then any accepting path in N_L that used the transition on s from x to y must now instead enter the corresponding copy of N_R and trace out an accepting path of N_R in order to reach y. So the modified version of N_L recognizes $\textsc{Replace}(L,s,R)$.