CSC236H: Introduction to the Theory of Computation

Assignment 3

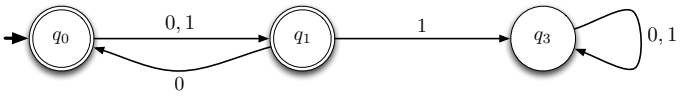
Due on Friday December 4, 2015 before 5pm (submit PDF on Markus)

(Solutions written by Azadeh Farzan)

---

1. **DFAs:** Design a DFA for each of the following languages (all over the alphabet $\{0,1\}$). It suffices to provide a **clean** (readable) drawing of the DFA in each case. Also, formally explain the idea behind your design by providing the state invariants for all the states in your DFA. We don't want you to be verbose. We are not looking for an explanation. One line per state should suffice for you to give a clean invariant of the state. For uniformity, write the invariants in the set form of page 205 of your text. Or, if it is hard to do a big curly bracket in cases, just write them in lines for each state like $q_1 : \{w \in \Sigma^* |\ P(w)\}$ (for the appropriate alphabet Σ).

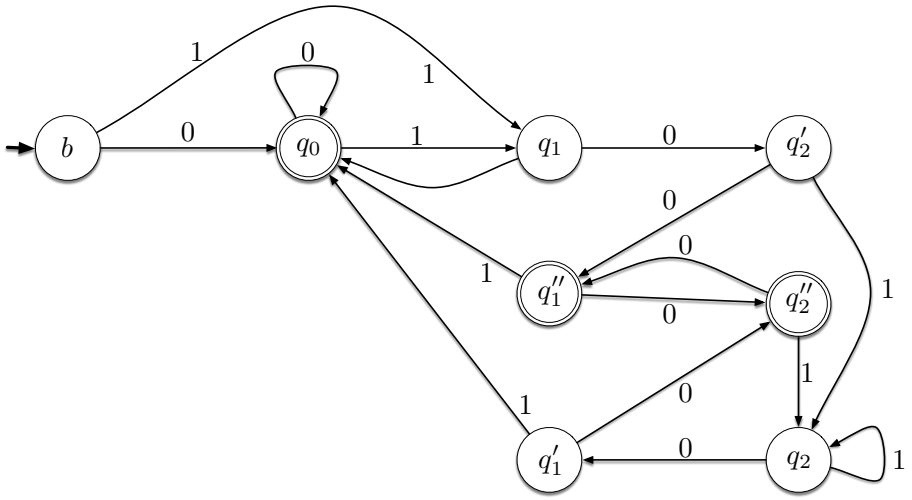
   (a) (5 points) $\{w|$ every odd position of w is a 0$\}$ (the first position is always position 0, similar to an array).
   Solutions:



   Here are the state invariants that explain the above DFA:

   q_0: $\{w \mid w$ has an even length and all its odd positions are 0's $\}$
   q_1: $\{w \mid w$ has an odd length and all its odd positions are 0's$\}$
   q_2: $\{w \mid w$ has a 1 at some odd position $\}$

   (b) (6 points) $\{w|\ |w|$ is divisible by 3 or it ends in 00$\}$
   Solutions:



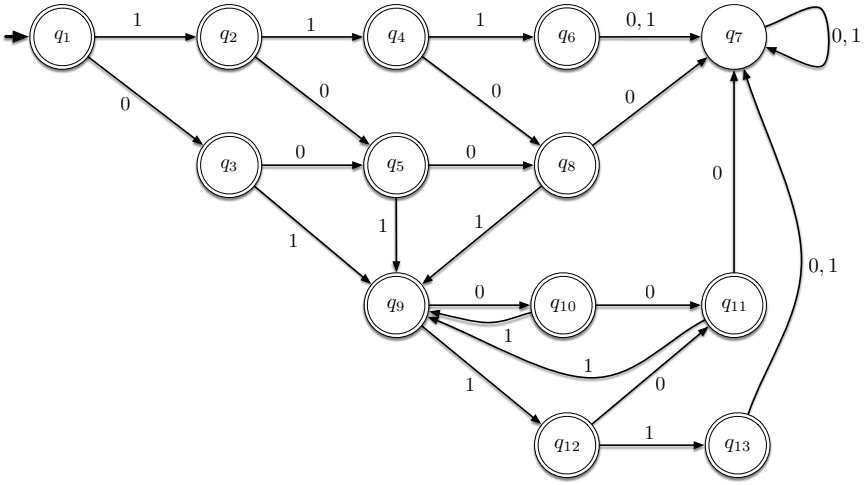
   we use the auxiliary function $\#(w)$ to refer to the number (in base 10) that is represented by the binary string w.

   b: $\{w \mid w = \epsilon\}$
   q_0: $\{w \mid \#(w)$ is divisible by 3$\}$

q_1: $\{w \mid \#(w)$ divided by 3 has remainder 1 and w ends in $1\}$
q_2: $\{w \mid w$ divided by 3 has remainder 2 and w ends in $1\}$
q_1': $\{w \mid \#(w)$ divided by 3 has remainder 1 and w ends in $10\}$
q_2': $\{w \mid w$ divided by 3 has remainder 2 and w ends in $10\}$
q_1'': $\{w \mid \#(w)$ divided by 3 has remainder 1 and w ends in $00\}$
q_2'': $\{w \mid w$ divided by 3 has remainder 2 and w ends in $00\}$

(c) (7 points) $\{w|$ every block of 4 consecutive symbols contains the substring $01\}$ (for example 00100 is in the language and 10001000 is not)

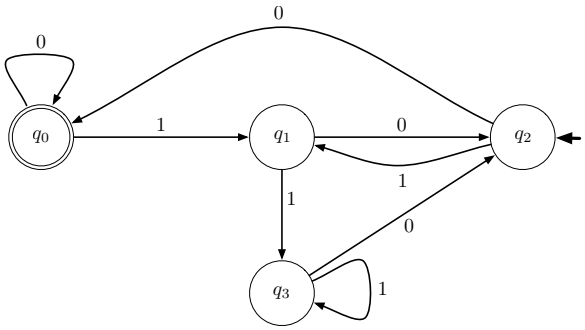
Solutions:



Here are the state invariants that explain the above DFA:

q_1: $\{w \mid w = \epsilon\}$
q_2: $\{w \mid w = 1\}$
q_3: $\{w \mid w = 0\}$
q_4: $\{w \mid w = 11\}$
q_5: $\{w \mid w = 10 \ \vee \ w = 00\}$
q_6: $\{w \mid w = 111\}$
q_7: $\{w \mid w$ has a prefix 1111, 1110, 1100, 1000, or 0000, or a block of 4 consecutive symbols that does not contain the substring 01 $\}$
q_8: $\{w \mid w = 100 \ \vee \ w = 000 \ \vee \ w = 110\}$
q_9: $\{w \mid w$ ends in 01 and has the substring 01 in every block of 4 consecutive symbols $\}$
q_{10}: $\{w \mid w$ ends in 010 and has the substring 01 in every block of 4 consecutive symbols $\}$
q_{11}: $\{w \mid w$ ends in 0100 or 0110 and has the substring 01 in every block of 4 consecutive symbols $\}$
q_{12}: $\{w \mid w$ ends in 011 and has the substring 01 in every block of 4 consecutive symbols $\}$
q_{13}: $\{w \mid w$ ends in 0111 and has the substring 10 in every block of 4 consecutive symbols $\}$

2. **DFA Language Proof:** Prove that the following automata (M) recognizes the set of strings w (over alphabet $\{0, 1\}$) such that w interpreted as a binary number is divisible by 4.



Do this in two steps:

(a) (4 points) Write the state invariants for every state.
Solution: we use the auxiliary function $\#(w)$ to refer to the number (in base 10) that is represented by the binary string w.

$$\delta(q_0, w) = q_0 \iff \#(w) \text{ is divisible by 4} \tag{1}$$
$$\delta(q_1, w) = q_1 \iff \#(w) \text{ divided by 4 has remainder 1} \tag{2}$$
$$\delta(q_2, w) = q_2 \iff \#(w) \text{ divided by 4 has remainder 2} \vee w = \epsilon \tag{3}$$
$$\delta(q_3, w) = q_3 \iff \#(w) \text{ divided by 4 has remainder 3} \tag{4}$$

(b) (7 points) Prove that the invariants are inductive. Or, alternatively, write a full induction proof. Either one is fine, but start your solution by clearly saying which one you are doing, and be careful because if you go off-script in your method of choice, then you will lose marks (even if your proof ends up being closer to the other choice).
Solution: Here is the shorter inductiveness proof (an induction proof would have exactly the same sub-cases, 9 in total):

- $w = \epsilon$: Therefore, $\delta(q_2, w) = q_2$. And, since ϵ is part of the state invariant for q_2, invariant (3) holds since both sides of the $\iff$ are true. All the other invariants hold vacuously since both sides of their $\iff$ are false.
- Let $w = ua$. We have the following 4 cases:
  - $\delta(q_2, u) = q_0$: Therefore, by invariant (1), we know that $\#(u)$ is divisible by 4, or $\exists k \in \mathbb{N}_0 : \#(u) = 4k$.
    Depending on what a is we have two sub-cases:
    - $a = 0$: Then, $\delta(q_2, ua) = \delta(q_0, 0) = q_0$. Also, $\#(ua) = 2 \times \#(u) = 8k$. Therefore, invariant (1) is true since both sides of $\iff$ are true, and all other invariants are vacuously true since both sides of $\iff$ are false.
    - $a = 1$: Then, $\delta(q_2, ua) = \delta(q_0, 1) = q_1$. Also, $\#(ua) = 2 \times \#(u) + 1 = 8k + 1$. Therefore, invariant (2) is true since both sides of $\iff$ are true, and all other invariants are vacuously true since both sides of $\iff$ are false.
  - $\delta(q_2, u) = q_1$: Therefore, by invariant (2), we know that $\#(u)$ divided by 4 has remainder 1, or $\exists k \in \mathbb{N}_0 : \#(u) = 4k + 1$.
    Depending on what a is we have two subcases:
    - $a = 0$: Then, $\delta(q_2, ua) = \delta(q_1, 0) = q_2$. Also, $\#(ua) = 2 \times \#(u) = 8k + 2$. Therefore, invariant (3) is true since both sides of $\iff$ are true, and all other invariants are vacuously true since both sides of $\iff$ are false.
    - $a = 1$: Then, $\delta(q_2, ua) = \delta(q_1, 1) = q_3$. Also, $\#(ua) = 2 \times \#(u) + 1 = 8k + 3$. Therefore, invariant (4) is true since both sides of $\iff$ are true, and all other invariants are vacuously true since both sides of $\iff$ are false.

* $\delta(q_2, u) = q_2$: Therefore, by invariant (3), we know that $\#(u)$ divided by 4 has remainder 2, or $\exists k \in \mathbb{N}_0 : \ \#(u) = 4k + 2$.
  Depending on what a is we have two subcases:
  - $a = 0$: Then, $\delta(q_2, ua) = \delta(q_2, 0) = q_0$. Also, $\#(ua) = 2 \times \#(u) = 8k+4 = 4(2k+1)$. Therefore, invariant (1) is true since both sides of $\iff$ are true, and all other invariants are vacuously true since both sides of $\iff$ are false.
  - $a = 1$: Then, $\delta(q_2, ua) = \delta(q_2, 1) = q_1$. Also, $\#(ua) = 2 \times \#(u) + 1 = 8k + 5 = 4(2k + 1) + 1$. Therefore, invariant (2) is true since both sides of $\iff$ are true, and all other invariants are vacuously true since both sides of $\iff$ are false.
* $\delta(q_2, u) = q_3$: Therefore, by invariant (4), we know that $\#(u)$ divided by 4 has remainder 3, or $\exists k \in \mathbb{N}_0 : \ \#(u) = 4k + 3$.
  Depending on what a is we have two subcases:
  - $a = 0$: Then, $\delta(q_2, ua) = \delta(q_3, 0) = q_2$. Also, $\#(ua) = 2 \times \#(u) = 8k + 6 = 4(2k + 1) + 2$. Therefore, invariant (3) is true since both sides of $\iff$ are true, and all other invariants are vacuously true since both sides of $\iff$ are false.
  - $a = 1$: Then, $\delta(q_2, ua) = \delta(q_3, 1) = q_3$. Also, $\#(ua) = 2 \times \#(u) + 1 = 8k + 7 = 4(2k + 1) + 3$. Therefore, invariant (4) is true since both sides of $\iff$ are true, and all other invariants are vacuously true since both sides of $\iff$ are false.

Since we have now proved all the invariants, we know specifically that of the accept state (state q_0) is true. Therefore,

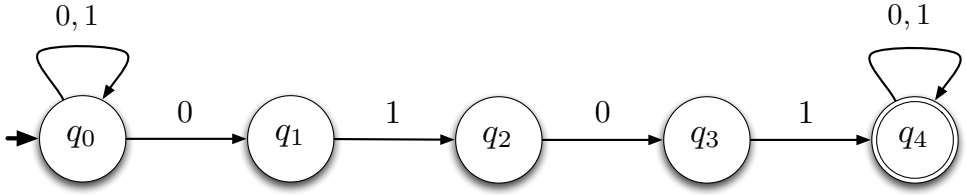
$$L(M) = \{w \in \{0,1\}^* | \ \#(w) \text{ is divisible by } 4\}$$

Note: without this short summary, your proof is incomplete! Alternatively, you could have started with this summary, similar to what we did in lecture 19, by starting with this. But, it is important to note that one of these two (intro or summary) is necessary to link the proved invariants to the language of the DFA.

3. **NFAs:** Give NFAs with the specified number of states recognizing each of the following languages (no ϵ transitions).

   (a) (5 points) The language $\{w \in \{0,1\}^* \mid w$ contains the substring 0101, i.e., $w = x0101y$ for some $x, y \in \{0,1\}^*\}$ with five states or fewer.

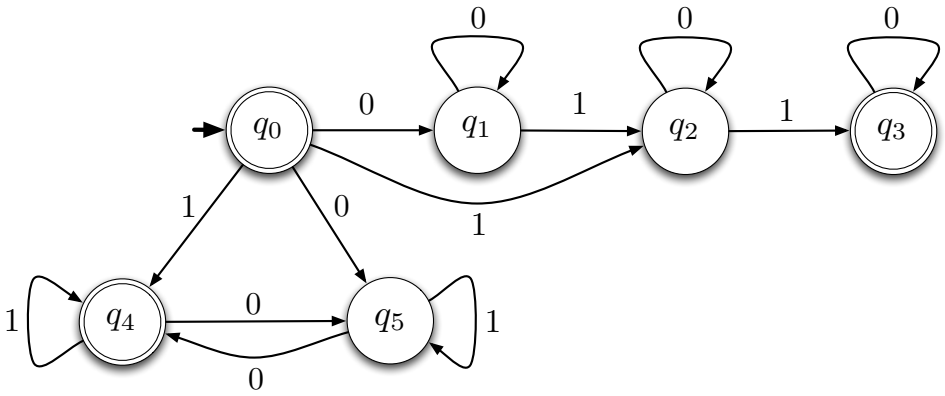
   Solutions: Here is the solution:



   The NFA basically keeps making a guess on seeing a "0" at the beginning that it is the beginning of the substring "0101". If this guess is correct (i.e. the following 3 characters are exactly "101"), then one of the guess will be correct and the string is accepted, no matter what follows. If the string does not contain this substring, then all such guess will fail, and it is rejected.

   (b) (5 points) The language $\{w \in \{0,1\}^* \mid w$ contains an even number of 0s, or exactly two 1s$\}$ with six states or fewer.

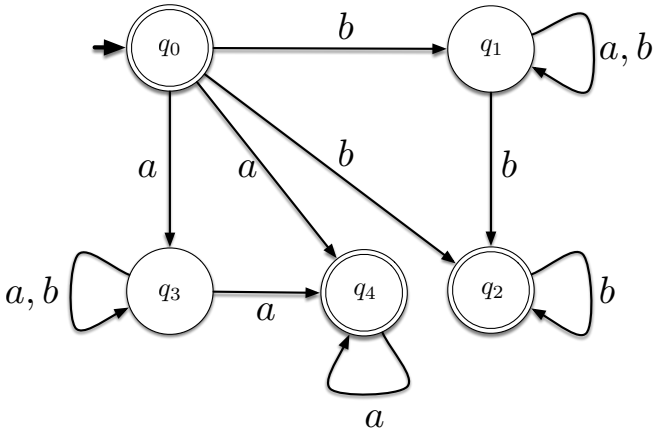
   Solutions: Here is the solution:



The NFA basically consists of two parts which it chooses nondeterministically to go into: (1) the part with q_1, q_2, and q_3 which checks if the string has exactly two 1's, and (2) the part with q_4 and q_5 that checks if the number of 0's in the string is even.

(c) (5 points) $\{w \in \{a,b\}^* \mid w$ contains an equal number of occurrences of the substrings ab and ba$\}$. This means that aba is in the language, because aba contains a single occurrence of ab and a single occurrence of ba. On the other hand, $abab$ is not, as it contains two ab's and one ba.

Solutions: The key to this solution is to realize that it suffices to capture strings that begin and end with the same letter. For example, if we start with an a and see no b's, then we are OK. If we see a b, then we expect to see at least one more a after the sequence of b's, and so on. The NFA below captures strings that start and end with the same symbol (a or b).



4. **Closure Properties of Regular Languages:** Prove that if L is a regular language over alphabet Σ, then so is $L' = \{xyz |\ x, y, z \in \Sigma^* \ \wedge\ x, z \notin L \ \wedge\ y \in L\}$. We want you to make this argument in two different ways:

(a) (3 points) Using closure properties of regular languages (and only that) make the argument. Note that proofs that are not based on closure properties of regular languages will get no credit.

Solutions: By definitions of concatenation and complementation of languages, we can define L' based on as follows:

$$L' = \overline{L}L\overline{L}$$

The argument for regularity goes as follows:

- Since L is regular, so is $\overline{L}$ (closed under complementation).
- Therefore, so is $\overline{L}L$ (closed under concatenation).
- Therefore, so is $\overline{L}L\overline{L}$ (closed under concatenation).
- Therefore, so is L'.

(b) (6 points) Let $M = (Q, \Sigma, \delta, q_0, F)$ be the DFA that accepts L (i.e. $L = L(M)$). Prove L' is regular by defining the components below, define a finite automaton $M' = (Q', \Sigma', \delta', q_0', F')$ that accepts L'. This finite automaton need not be a DFA; an NFA is equally fine.

Solutions: Let $\overline{M}$ be the DFA that recognizes $\overline{L}$. We know that $\overline{M} = (Q, \Sigma, \delta, q_0, Q - F)$. We know that we want to effectively concatenate a $\overline{M}$ with an M and then with another $\overline{M}$. But, to mathematically define this, we need to create three different copies of this DFA. We do that with the help of labeling the two new copies with tags 1 and 2 using the cartesian product.

- $Q' = Q \cup Q \times 1 \cup Q \times 2$
  where Q is the copy of the states for the first $\overline{M}$, $Q \times \{1\}$ is the copy of the states for M, and $Q \times \{2\}$ is the copy of the states for the second $\overline{M}$.
- $q_0' = q_0$
- $F' = (Q - F) \times 2$ i.e. the accept states for the second $\overline{M}$
- Finally for δ':

$$
\begin{aligned}
\delta' = \delta \cup & && \text{For the first } \overline{M} \\
& \{(\langle q, 1\rangle, a, \{q'\}) |\ q, q' \in Q, a \in \Sigma, \delta(q, a) = q'\} \cup && \text{For } M \\
& \{(\langle q, 2\rangle, a, \{q'\}) |\ q, q' \in Q, a \in \Sigma, \delta(q, a) = q'\} \cup && \text{For the second } \overline{M} \\
& \{(q, \epsilon, \{\langle q_0, 1\rangle\}) |\ q \in Q - F\} \cup && \text{For concatenating the first } \overline{M} \text{ to } M \\
& \{(\langle q, 1\rangle, \epsilon, \{\langle q_0, 2\rangle\}) |\ q \in F\} && \text{For concatenating } M \text{ to the second } \overline{M}
\end{aligned}
$$

5. **Regular Expressions:** for each answer in this part, briefly explain your expression by saying what type of strings are generate by each significant component of it. This helps us give you partial credit if your answer is not perfect. Unless otherwise specified, the alphabet is $\Sigma = \{0, 1\}$.

(a) (5 points) All strings with no more than three 0's.

Solutions:

$$1^* + 1^*01^* + 1^*01^*01^* + 1^*01^*01^*01^*$$

The the four terms respectively represent all strings with exactly zero, one, tow, or three 0's.

(b) (5 points) All strings that contain both 00 and 11 as substrings.

Solutions:

$$[(0+1)^*00(0+1)^*11(0+1)^*] + [(0+1)^*00(0+1)^*11(0+1)^*]$$

The two main terms represent all strings that both have substrings 00 and 11, in the two possible orders for the substring to appear in the string.

(c) (5 points) All strings of a's and b's such that all blocks of a's are of odd length (a block of a's is maximal sequence of consecutive a's). That is, strings such as *bbbaaabaaaaa* and *aaaaabb* are in the language; *abbabaa* and *aaaa* are not.

Solutions:

$$a(aa)^* + (a(aa)^*b + b)^*(\epsilon + a(aa)^*)$$

Consier all odd-length blocks of a's. There is either only 1 and that is the entire string (when the string is captured by the first term). Or, there is more than one, in which case, these blocks are separated from each other by one or more b's (at least one b). In that case, we can represent them by odd-length blocks of a's ending in a single b (i.e. $a(aa)^*b$) and any strings of just b's interleaved (i.e. $(a(aa)^*b + b)^*$). But, these strings always end in b's. We need to make sure that there is the possibility of one more odd-length block of a's to appear at the end; hence the last $(\epsilon + a(aa)^*)$ term concatenated to the end.

(d) (5 points) All strings where every odd-length block of 0's is immediately followed by an odd-length block of 1's, and every even-length block of 0's is immediately followed by an even-length block of 1's.

Solutions:

$$1^*[(00)^+(11)^+ + (00)^*0(11)^*1]^*$$

Short description: keep in mind that there are no obligations for blocks of 1's to be followed by anything. All blocks of 0's however, should be matched with a block of 1's that follows them that has the same parity.

1^* generates all strings that have no 0's at all (the second term can produce just ϵ trivially). We need this, because as long as there are no 0's we should accept the string.

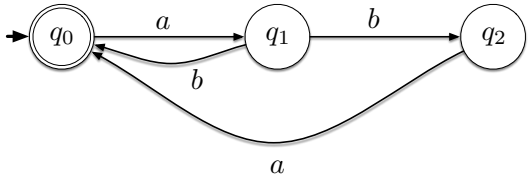
If the string has any 0's in it, then $(00)^+(11)^+$ covers all even-length blocks of 0's that must be followed by even-length blocks of 1's, and $(00)^*0(11)^*1$ covers all odd-length blocks of 0's that must be followed by odd-length blocks of 1's. Therefore, $1^*[(00)^*0(11)^+ + (00)^+(11)^*1]^*$ generates all arrangements of all such blocks of zeros and ones in any order.

Keep in mind that it is OK here for the string here to always end in 1. It cannot end in a 0, because whatever that last block of 0's is has to be followed by a parity-matching block of 1's.

6. **Subset Construction:** Consider the regular expression $(ab + aba)^*$.

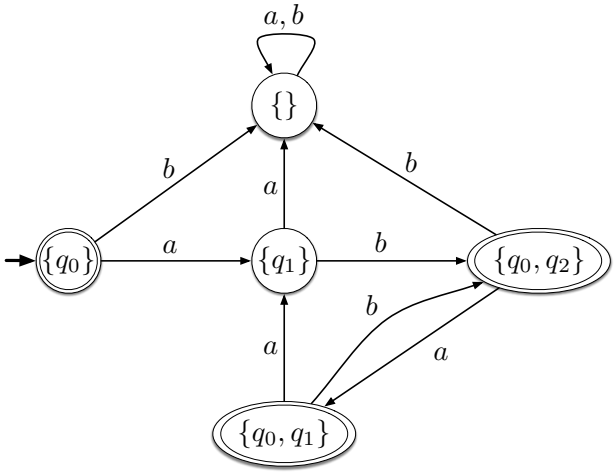
(a) (3 points) Give a 3-state NFA that recognizes the same language.

Solutions: Here is the NFA:



(b) (4 points) Give a DFA that recognizes the same language by converting your NFA from (a) to a DFA through subset construction.

Solutions: Here is the DFA acquired by subset construction:



7. (9 points) The operand $\emptyset$ is not really required for the construction of regular expressions, except that without it, we could not write a regular expression whose language is the empty set. Call a regular expression $\emptyset$-free if it contains no occurrences of $\emptyset$. Prove, by induction, that a regular expression r is either equivalent to the regular expression $\emptyset$ or some $\emptyset$-free regular expression.

Solution: Let $\mathcal{R}$ to be the set of all regular expressions (including those with the $\emptyset$ operand).

Statement: $P(r) \equiv\ (L(r) \neq \emptyset) \Rightarrow (\exists r' :\ r' \text{ is } \emptyset\text{-free } \wedge\ L(r) = L(r'))$

Goal: prove $\forall r \in \mathcal{R},\ P(r)$.

Proof by structural induction on $\mathcal{R}$.

Basis: there are three possibilities for r in the basis:

- $r = \epsilon$: we have $L(r) = \{\epsilon\} \neq \emptyset$. And, ϵ is $\emptyset$-free, therefore $P(\epsilon)$ holds.
- $r = a$ (where $a \in \Sigma$): we have $L(r) = \{a\} \neq \emptyset$. And, a is $\emptyset$-free, therefore $P(a)$ holds.
- $r = \emptyset$: we have $L(r) = \emptyset$, therefore $P(\emptyset)$ vacuously holds.

Induction step: there are three construction rules in the recursive definition of $\mathcal{R}$. In each case, we assume that the components satisfy the property P, and prove the property holds for the result of the construction step:

- $r = r_1 + r_2$: We assume $P(r_1)$ holds and $P(r_2)$ holds. We have $L(r) = L(r_1) \cup L(r_2)$. We look at two possible subcases:
  - $L(r_1) = \emptyset \wedge L(r_2) = \emptyset$: In this case, $L(r) = \emptyset \cup \emptyset = \emptyset$, and therefore $P(r)$ holds vacuously.
  - $L(r_1) \neq \emptyset$: which implies $L(r) \neq \emptyset$. Since $P(r_1)$ is true, we know $\exists r_1' = r_1$ where r_1' is $\emptyset$-free. There are two possibilities with respect to r_2:
    - $L(r_2) \neq \emptyset$: Since $P(r_2)$ is true, we know $\exists r_2' = r_2$ where r_2' is $\emptyset$-free. Let $r' = r_1' + r_2'$. Since r_1' is $\emptyset$-free and r_2' is $\emptyset$-free, we know r' is $\emptyset$-free. And, since $r_1' = r_1$ and $r_2' = r_2$, we have $r = r_1 + r_2 = r_1' + r_2' = r'$. Therefore $P(r)$ holds.
    - $L(r_2) = \emptyset$: Then, $r = r_1 + r_2 = r_1'$. Since r_1' is $\emptyset$-free, $P(r)$ holds.
- $r = r_1 r_2$: We assume $P(r_1)$ holds and $P(r_2)$ holds. We have $L(r) = L(r_1)L(r_2)$. We look at two possible sub-cases:
  - $L(r_1) = \emptyset$: We have $L(r) = \emptyset L(r_2) = \emptyset$. Therefore, $P(r)$ holds vacuously.
  - $L(r_1) \neq \emptyset$: There are two possibilities for r_2:
    - $L(r_2) = \emptyset$: We have $L(r) = L(r_1)\emptyset = \emptyset$. Therefore, $P(r)$ holds vacuously.
    - $L(r_2) \neq \emptyset$: Since $P(r_1)$ is true, we know $\exists r_1' = r_1$ where r_1' is $\emptyset$-free. And, since $P(r_2)$ is true, we know $\exists r_2' = r_2$ where r_2' is $\emptyset$-free. Let $r' = r_1' r_2'$. Since r_1' and r_2' are both $\emptyset$-free, we can conclude that r' is also $\emptyset$-free. Also, $r = r_1 r_2 = r_1' r_2' = r'$. Therefore, $P(r)$ holds.
- $r = r_1^*$: We look at two possible sub-cases:
  - $L(r_1) = \emptyset$: We have $L(r) = (\emptyset)^* = \{\epsilon\}$. Since $L(\epsilon) = \{\epsilon\}$, by letting $r' = \epsilon$, we have $P(r)$ is true.
  - $L(r_1) \neq \emptyset$: Since $P(r_1)$ is true, we know $\exists r_1' = r_1$ where r_1' is $\emptyset$-free. Therefore, we can conclude that $(r_1')^*$ is also $\emptyset$-free. We have $r = r_1^* = (r_1')^*$. Therefore, $P(r)$ holds.

8. For a language L, let $\textsc{Init}(L) = \{x \mid xy \in L \text{ for some } y \in \Sigma^*\}$. Let r, s, r_I and s_I be regular expressions for the languages R, S, $\textsc{Init}(R)$, and $\textsc{Init}(S)$ respectively. Using only these regular expressions and the operations $+$, concatenation, and $*$, give expressions for the following languages and very briefly justify your answers:

   Solution: (informal justifications suffice for marking, but the formal justifications are given to you as extra examples of proper formalism!)

   (a) (4 points) $\textsc{Init}(R \cup S) = r_I + s_I$. Informal justification: since $\textsc{Init}(R \cup S)$ is the set of all prefixes of all the strings in $R \cup S$, and any string that is in $(R \cup S)$ is either in R or in S, that same prefix will be either in $\textsc{Init}(R)$ or in $\textsc{Init}(S)$. Therefore, it is in $\textsc{Init}(R) \cup \textsc{Init}(S) = r_I + s_I$.
   The formal version of the same argument:

$$
\begin{aligned}
u \in \textsc{Init}(R \cup S) &\iff \exists v \in \Sigma^* :\ uv \in R \cup S \\
&\iff \exists v \in \Sigma^* :\ uv \in R \vee uv \in S \\
&\iff (\exists v \in \Sigma^* :\ uv \in R) \vee (\exists v \in \Sigma^* :\ uv \in S) \\
&\iff u \in \textsc{Init}(R) \vee u \in \textsc{Init}(S) \\
&\iff u \in \textsc{Init}(R) \cup \textsc{Init}(S) \\
&\iff u \in L(r_I + s_I)
\end{aligned}
$$

(b) (4 points) $\textsc{Init}(RS) = r_I + rs_I$.

Informal justification: $\textsc{Init}(RS)$ is the set of all prefixes of all the strings in RS. Any string that is in $\textsc{Init}(RS)$ is of the form uv where $u \in R$ and $v \in S$. A prefix of uv is either a prefix of u or it is a concatenation of a u with a prefix of v. All the prefixes of all such u's is captured by r_I. All prefixes of all v's is captured by s_I. Therefore, $r_I + rs_I$ captures the two categories of the strings.

The formal version of the same argument:

$$
\begin{aligned}
u \in \textsc{Init}(RS) &\iff \exists v \in \Sigma^* :\ uv \in RS \\
&\iff \exists v \in \Sigma^* :\ (\exists x, y \in \Sigma^* :\ uv = xy \wedge x \in R \wedge y \in S) \\
&\iff \exists v \in \Sigma^* :\ [(\exists s, t, w \in \Sigma^* :\ u = st \wedge v = w \wedge s \in R \wedge tw \in S) \\
&\qquad \vee (\exists s, t, w \in \Sigma^* :\ u = s \wedge v = tw \wedge st \in R \wedge w \in S)] \\
&\iff \exists v \in \Sigma^* :\ [(\exists s, t :\ s \in R \wedge t \in \textsc{Init}(S) \wedge u = st) \vee (u \in \textsc{Init}(R))] \\
&\iff u \in R\textsc{Init}(S) \vee u \in \textsc{Init}(R) \\
&\iff u \in R\textsc{Init}(S) \cup \textsc{Init}(R) \\
&\iff u \in L(rs_I + r_I)
\end{aligned}
$$

(c) (4 points) $\textsc{Init}(R^*) = r^* r_I$.

Informal justification: $\textsc{Init}(R^*)$ is the set of all prefixes of all the strings in R^*. Any string that is in $\textsc{Init}(R^*)$ is of the form $u^* v$ where $u \in R$ and v is a prefix of a string in R, and hence $v \in \textsc{Init}(R)$. Therefore, $r^* r$ captures all such strings.

$$
\begin{aligned}
u \in \textsc{Init}(R^*) &\iff \exists v \in \Sigma^* :\ uv \in R^* \\
&\iff \exists v \in \Sigma^* :\ (\exists x_1, \ldots, x_k \in \Sigma^* :\ uv = x_1 \ldots x_k \wedge \forall i :\ x_i \in R) \\
&\iff \exists v \in \Sigma^* :\ [(\exists i \exists s, t \in \Sigma^* :\ u = x_1 \ldots x_i s \wedge v = t x_{i+2} \ldots x_k \wedge x_{i+1} = st \wedge \forall i :\ x_i \in R) \\
&\iff [(\exists i \exists s, t \in \Sigma^* :\ u = x_1 \ldots x_i s \wedge \forall i :\ x_i \in R \wedge s \in \textsc{Init}(R)) \\
&\iff u \in R^* \textsc{Init}(R) \\
&\iff u \in L(r^* r_I)
\end{aligned}
$$