On the Equivalence Between Regular Languages and Regular Expressions

Last Updated: August 13, 2025

1 Introduction

By definition, a language is regular iff it can be recognized by a DFA. Furthermore, in the Nondeterministic Finite Automata lecture we saw that any language recognized by an NFA can also be recognized by a DFA. And since every DFA is an NFA, it follows that a language is regular iff it is accepted by an NFA. In this lecture we show that the set of languages that can be described by a regular expression is equal to the set of regular languages, which justifies the name of such expressions. We accomplish this in two steps.

1. We show that every regular expression E may be converted to an NFA that recognizes the language $L(E)$ that is described by E. We conclude that any language that can be described by a regular expressions must be regular. Mathematically we may write

$$\text{RegEx} \subseteq \text{Regular}.$$

2. Conversely, we then show how to convert any NFA N to a regular expression E such that $L(N) = L(E)$. We then conclude that *every* regular language may be described by a regular expression, i.e.

$$\text{Regular} \subseteq \text{RegEx}.$$

Putting the two statement together yields

$$\text{RegEx} = \text{Regular}.$$

2 RegEx $\subseteq$ Regular

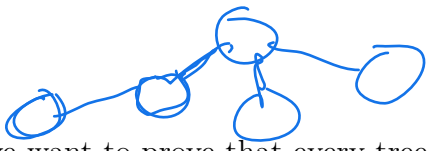
Theorem 1. Let L be a language over some alphabet Σ. If $L = L(E)$ can be described by some regular expression E, then L is regular.

The proof of Theorem 1 uses **structural induction**, a technique that is used for proving that the members of some set of objects all have some property, assuming that the set of objects is recursively defined.

Example 1. A **tree** is a special kind of simple graph which is connected and has no cycles. The following is a recursive definition for the set of all trees.

Base Case A simple graph that consists of a single node is a tree.

Recursive Case If T is a tree, then the graph T' is also a tree, where T' is obtained by adding a vertex v to T as well as the edge (u, v).



Now suppose we want to prove that every tree that has $n \geq 1$ vertices must also have exactly $n - 1$ edges. The following proof of this fact uses structural induction in the following manner.

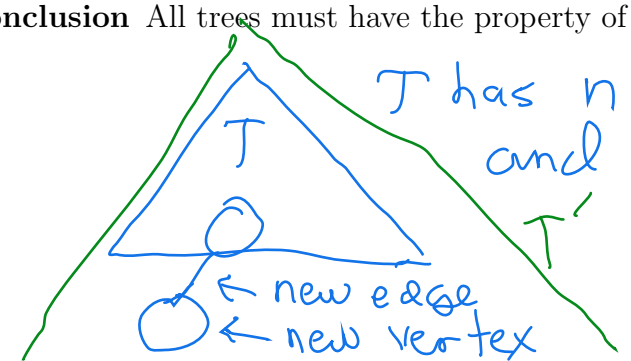
Basis Step: Every Atomic Tree Has the Property If tree T consists of a single node, then it has 1 vertex and $1 - 1 = 0$ edges.



Inductive Step: Every Compound Tree Has the Property

1. Let T be any tree that has $n \geq 1$ vertices and $n - 1$ edges.
2. Let T' be the graph obtained from T by adding a vertex v to T as well as the edge (u, v).
3. Then T' has $n + 1$ vertices and $(n - 1) + 1 = n$ edges. □

Conclusion All trees must have the property of having one fewer number of edges than vertices.



Proof of Theorem 1. Since the set of all regular expressions over some alphabet was recursively defined in the previous lecture, we may prove the theorem using structural induction. The following is an outline of such a proof.

2.1 Every Atomic Regular Expression Describes a Regular Language

Basis Step 1 For each letter $\mathtt{a} \in \Sigma$, $\mathtt{a}$ is a regular expression that represents the regular language $\{\mathtt{a}\}$.

Basis Step 2 ε is a regular expression that represents the regular language $\{\varepsilon\}$.

Basis Step 3 $\emptyset$ is a regular expression that represents the regular language $\emptyset$.

2.2 Every Compound Regular Expression Describe a Regular Language

Inductive Step 1 If R_1 and R_2 are regular expressions and $L(R_i)$ is a regular language, $i = 1, 2$, then $(R_1 \cup R_2)$ is a regular expression that represents the regular language $L(R_1) \cup L(R_2)$.

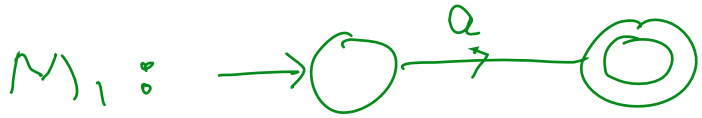
Inductive Step 2 If R_1 and R_2 are regular expressions and $L(R_i)$ is a regular language, $i = 1, 2$, then $(R_1 \circ R_2)$ is a regular expression that represents the regular language $L(R_1) \circ L(R_2)$.

Inductive Step 3 If R is a regular expression $L(R)$ is a regular language, then (R^*) is also a regular expression that represents the regular language $L(R)^*$.

Example 2. Provide NFA's that establish the fact that every atomic regular expression describes a regular language.

$a \in \Sigma$

a describes $\{a\}$

M_1:  $L(M_1) = \{a\}$

ε describes $\{\varepsilon\}$

M_2:  $L(M_2) = \{\varepsilon\}$

$\emptyset$ describes $\emptyset$

M_3:  $L(M_3) = \emptyset$

2.3 Closure properties of regular languages

To finish the proof of Theorem 1, we must now show that

1. The union of any two regular languages is also regular,
2. The concatenation of any two regular languages is also regular, and
3. The star of any regular language is also regular.

We interpret this by saying that "regular languages are *closed* under the operations of union, concatenation, and star".

Closure under union If NFA's are equivalent in power to DFA's, then why bother with them? The reason is because many regular languages are more easily defined using an NFA. Especially those languages that are i) a union of two or more regular languages, a concatenation of two regular languages, or ii) the star of some regular language.

Proposition 1. If A and B are regular, then so is $A \cup B$.

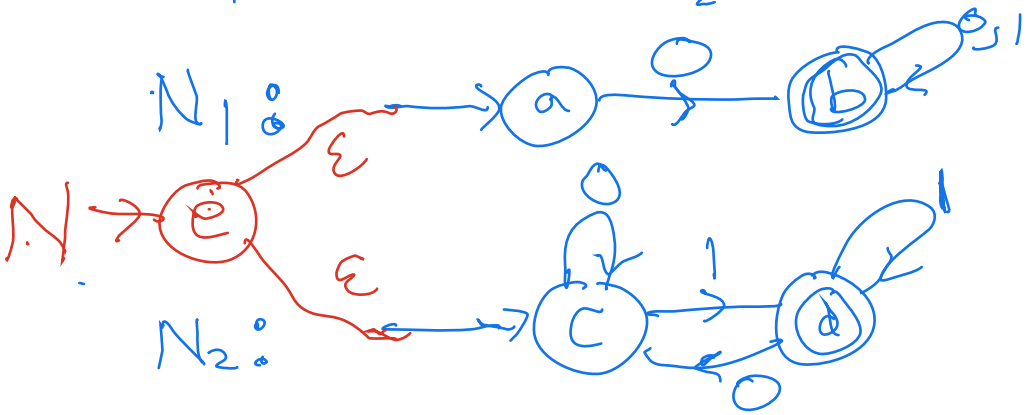
Proof of Proposition 1. Suppose NFA M_1 accepts A and NFA M_2 accepts B. Then to construct a state diagram for an NFA N that accepts $A \cup B$, do the following.

1. Construct the state diagram for M_1, where the node labeled with q_1 is its initial state and F_1 is its set of accepting states.
2. Construct the state diagram for M_2, where the node labeled with q_2 is its initial state and F_2 is its set of accepting states.
3. Add a new node labeled with q_0 which serves as N's initial state.
4. Add ε-labeled edges (q_0, q_1) and (q_0, q_2).
5. Designate any state in $F_1 \cup F_2$ as an accepting state for N.

Then, because of the two ε-edges (q_0, q_1) and (q_0, q_2), a computation of N on input word w results in a parallel computation of both M_1 and M_2 on w, where the computation accepts iff either an accepting state of M_1 or an accepting state of M_2 belongs to the final subset state. In other words, $w \in L(N)$ iff $w \in A$ or $w \in B$. □

Example 3. Use the procedure described in the Proposition to construct an NFA that accepts all binary words that either begin with 0 or end with 1. Show the computation of N on input 101.

$R_1 = 0\{0,1\}^*$ $\quad R_2 = \{0,1\}^*1$



R_1 describes $L(N_1)$

R_2 describes $L(N_2)$

$L(N) = L(N_1) \cup L(N_2)$

ace 1 0 1 c d accept

Closure under concatenation

Proposition 2. If A and B are regular, then so is $A \circ B$.

Proof of Proposition 2. Suppose NFA N_1 accepts A, and NFA N_2 accepts B. Then to construct a state diagram for an NFA N that accepts $A \circ B$, do the following.

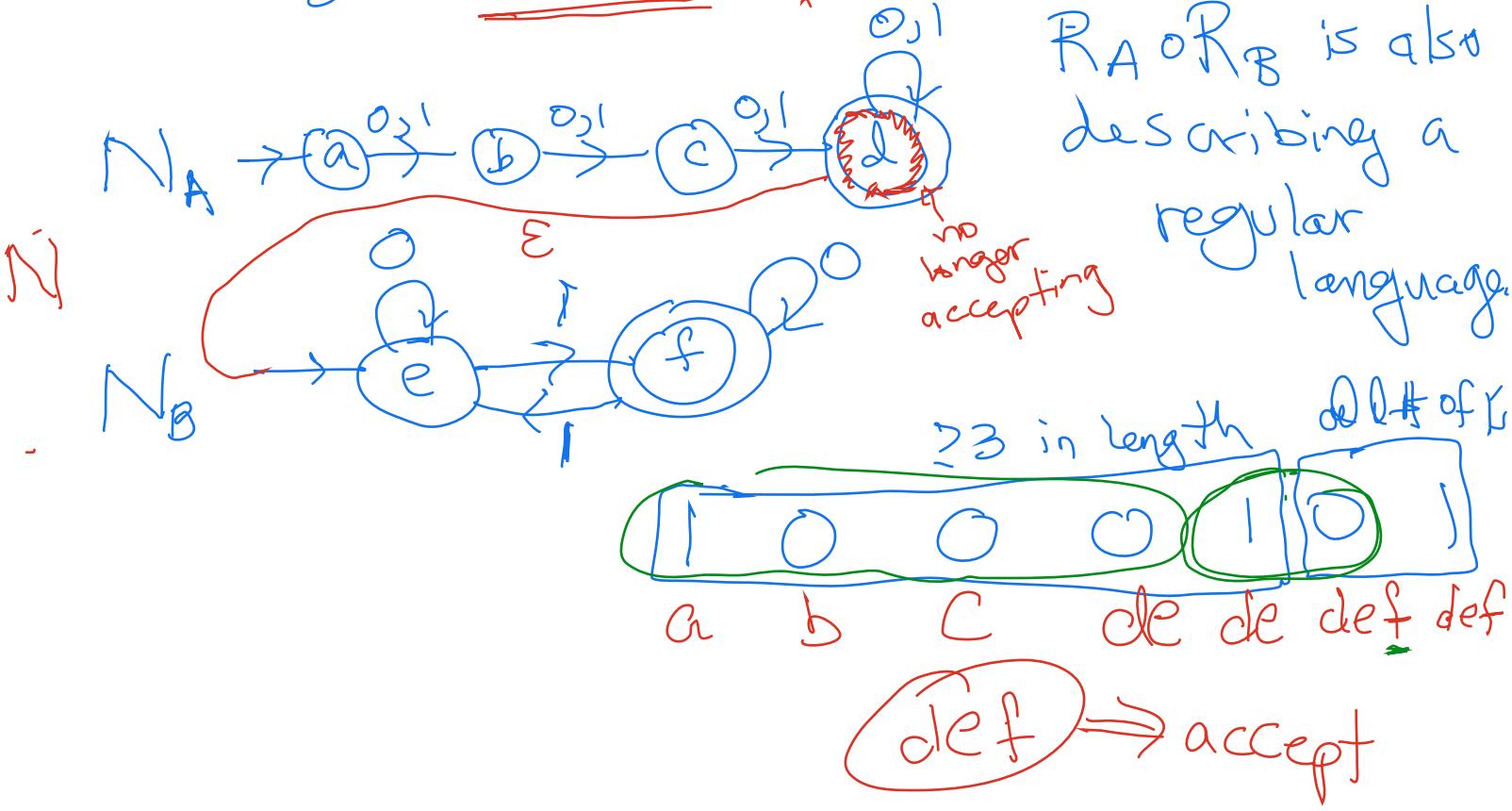
1. Construct the state diagram for N_1, where the node labeled with q_1 is its initial state and F_1 is its set of accepting states.
2. Construct the state diagram for N_2, where the node labeled with q_2 is its initial state and F_2 is its set of accepting states.
3. For each $q \in F_1$ add the ε-labeled edge (q, q_2).
4. Then the initial state for N is q_1, and F_2 is its set of accepting states.

By adding an ε-edge from an accepting state of N_1 to q_2, it allows for N to accept concatenations of A with B. For example, suppose 1001 is in A, and 0001 is in B. Then the computation of N on 1001 will end with a subset state that not only includes an accepting state of N_1, but also includes q_2, the initial state of N_2. This allows the computation to "start over" and next accept 0001. Thus, the entire word 10010001 will be accepted by N. □

Example 4. Use the procedure described in Proposition 2 to construct an NFA that accepts $A \circ B$, where A is the language of all binary words that have a length of at least three, and B is the language of all binary words that have an odd number of 1's. Show the computation of the NFA on input 1000101.

$R_A = \{0,1\}^3 \{0,1\}^*$

$R_B = 0^*10^*(10^*10^*)^*$



Closure under star

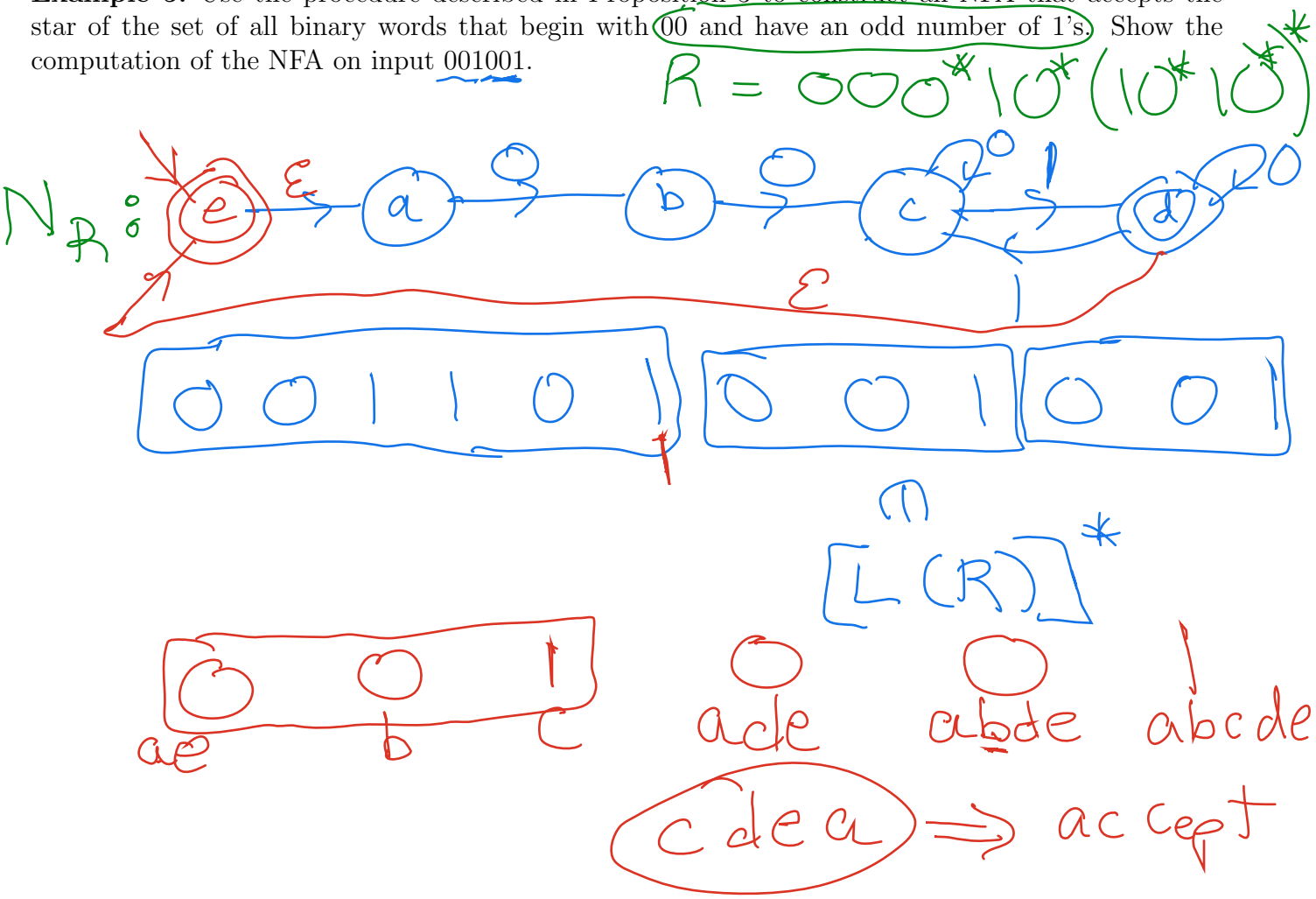
Proposition 3. If A is regular, then so is A^*.

Proof of Proposition 3. Suppose NFA M accepts A. Then to construct a state diagram for an NFA N that accepts A^*, do the following.

1. Construct the state diagram for M, where the node labeled with q_0 is its initial state and F is its set of accepting states.
2. Add a new node labeled with q_0' which serves as N's initial state.
3. Add the ε-labeled edge (q_0', q_0).
4. For each $q \in F$, add the ε-labeled edge (q, q_0).
5. Designate each state in F as an accepting state. Also designate q_0' as an accepting state.

Making q_0' an accepting state takes care of accepting input $\varepsilon \in A^*$. Also, by adding an ε-edge from an accepting state of M to q_0, it allows for N to accept concatenations of words from A. For example, if 1001 and 0001 are both in A. Then the computation of N on 1001 will end with a subset state that not only includes an accepting state of M, but also includes q_0. This allows the computation to "start over" and next accept 0001. Thus, the entire word 10010001 will be accepted by N. More generally, the concatenation of any number of words from A will be accepted by N, and concatenations of words from A are the only words accepted by N (in addition to ε). Therefore, N accepts A^*. $\square$

Example 5. Use the procedure described in Proposition 3 to construct an NFA that accepts the star of the set of all binary words that begin with 00 and have an odd number of 1's. Show the computation of the NFA on input 001001.



Example 6. Provide NFA's that accept the languages described by the regular expressions 000, 01, and $(11)^*$.

$$(000(01 \cup (11)^*)^*)^*.$$

Then use the closure under concatentation and union algorithms to construct a single NFA that accepts the regular language described by

$$000(01 \cup (11)^*.$$

3 All Regular Languages are Described by Regular Expressions

The converse to Theorem 1 is also true. In other words, we now show that

$$\text{Regular} \subseteq \text{RegEx}.$$

Theorem 2. Every regular language L is associated with some regular expression E, for which $L(E) = L$.

Proof of Theorem 1. Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA. The goal is to find a regular expression E such that $L(E) = L(N)$. Without loss of generality, we may assume that i) $|F| = 1$ and there is no transitioning from the sole accepting state q_a, and ii) there is no transitioning to initial state q_0 and $q_0 \neq q_a$. Thus N has at least two states.

It turns out that an NFA is a special case of a type of automaton called a **generalized nondeterministic finite automaton (GNFA)**. The state diagram of a GNFA allows for edges to be labeled with regular expressions (either atomic or compound), whereas a transition edge of an NFA is only allowed to be labeled either a finite subset of symbols in Σ, ε, or $\emptyset$ (being labeled with $\emptyset$ is the same as having no transition edge going from some state to another). As a consequence, a GNFA is capable of transitioning from one state q_1 to another q_2 by reading an entire word $w \in L(E)$ where E is the regular expression that labels the edge (q_1, q_2) going from q_1 to q_2. For this reason, a GNFA computation is similar to that of an NFA computation, with the exception that state transitions can be triggered by an entire word rather than a single symbol.

We now use induction on the number of states of GNFA N to prove that $L(N)$ is described by some regular expression.

Basis step. GNFA N has two states. Then $L(N)$ equals $L(E)$ where E is the label of the edge going from q_0 to q_a, and so $L(N)$ is described by regular expression E.

Inductive step. Assume that every GNFA N having $k \geq 2$ states accepts a language $L(N)$ that is described by some regular expression E. Now consider a GNFA N having $k+1$ states. The idea is to remove one of N's states q ($q \neq q_0$ and $q \neq q_a$) so that we may use the inductive assumption. We may do this so long as we update the regular expressions on each of the remaining edges of our GNFA's state diagram. For example, consider the edge (q_1, q_2) going from q_1 to q_2 before removing q. Let E denote the regular expression that labels this edge. By removing q, we've lost a way to read a word $w \in \Sigma^*$ that allows us to transition from q_1 to q_2. Namely, any word w that belongs to the language associated with the regular expression

$$E_1 \circ E_2^* \circ E_3,$$

where E_1 is the label for edge (q_1, q), E_2 is the label for the loop (q, q), and E_3 is the label for edge (q, q_2) (see diagram on next page). In other words w is any sequence of symbols that allows one to transition first from q_1 to q, then to loop a finite number of times at q, followed by transitioning from q to q_2. Indeed, if q gets removed, these words could be lost. For this reason we replace the expression E that labels (q_1, q_2) with the new expression

$$E \cup E_1 \circ E_2^* \circ E_3.$$

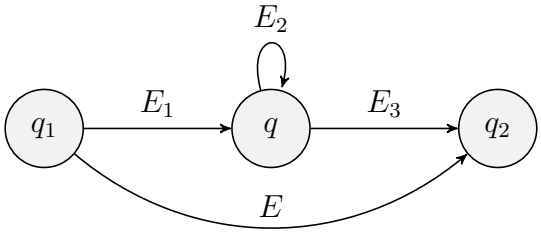
By doing this, we've preserved all the words that allow us to transition from q_1 to q_2. Moreover, after removing q, the new GNFA N' now has k states, but still accepts the same language as N. Thus, by the inductive assumption, we have

$$L(N) = L(N')$$

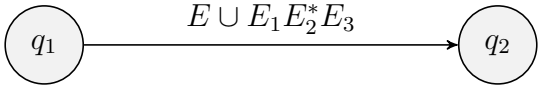
is described by regular expression. □

The case where $q_1 \neq q_2$

Before removing q:

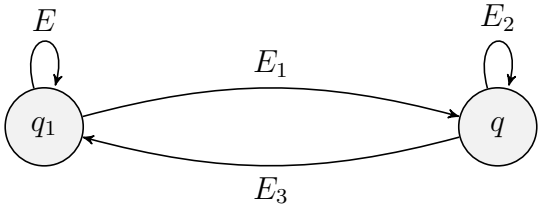


After removing q:

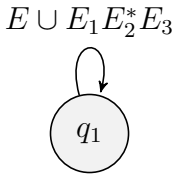


The case where $q_1 = q_2$

Before removing q:



After removing q:



The proof of Theorem 2 suggests an algorithm that starts with the initial NFA N and returns a regular expression E that describes $L(N)$.

Name: `gnfa_to_regex`

Input: GNFA N have at least two states and for which

1. initial state q_a has no edges entering it
2. final state $q_a \neq q_0$ has no edges leaving it
3. between any two (not necessarily different) states q_1 and q_2, N has at most one directed edge (q_1, q_2).

Output: regular expression E for which $L(N) = L(E)$.

If N has exactly two states, then return regular expression E that is the label of edge (q_0, q_a).

$//N$ has at least three states

Select a state $q \notin \{q_0, q_a\}$ to remove from N.

For each edge (q_1, q_2) for which (q_1, q) and (q, q_2) are edges of N,

If expression E is the label of (q_1, q_2), then replace it with

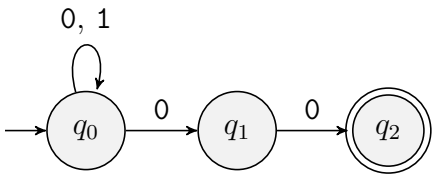
$$E \cup E_1 E_2^* E_3.$$

// E_1 is the label of (q_1, q), E_2 the label of (q, q), and E_3 the label of (q, q_2).

Let N' denote the resulting NFA.

Return gnfa_to_regex(N').

Example 7. Use the proof of Theorem 2 to derive the regular expression that describes the language accepted by the following NFA.



$\{0,1\}^*00$

$\{0,1\}$

1. Create new initial state

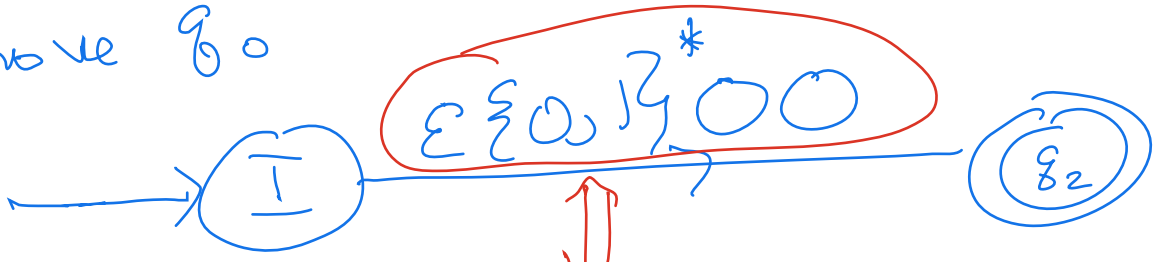


$\{0,1\}$

2. Remove q_1

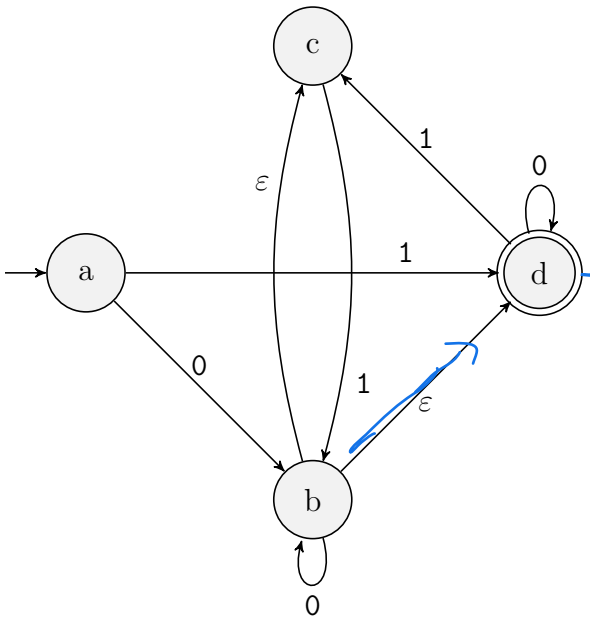


3. Remove q_0

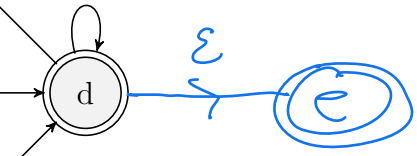


$\{0,1\}^*00$

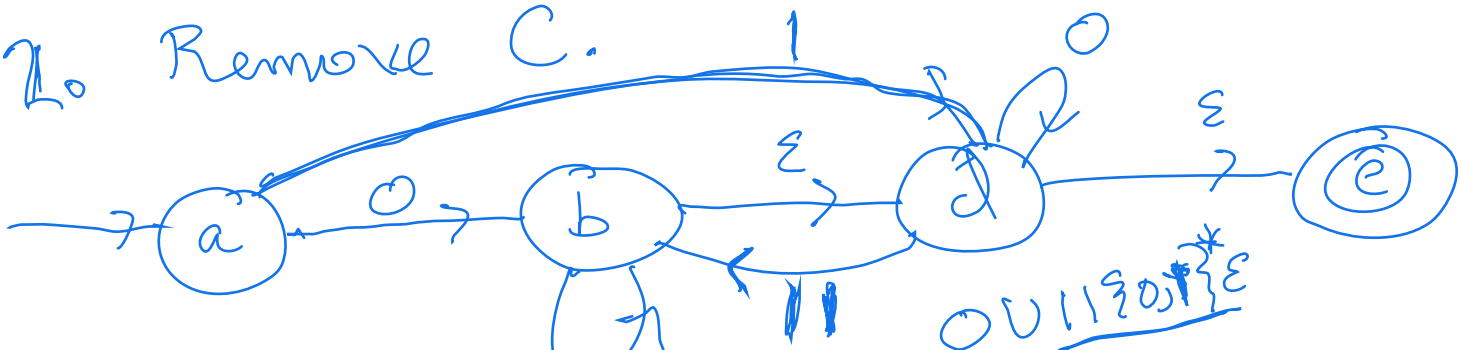
Example 8. Use the proof of Theorem 2 to derive the regular expression that describes the language accepted by the following NFA.



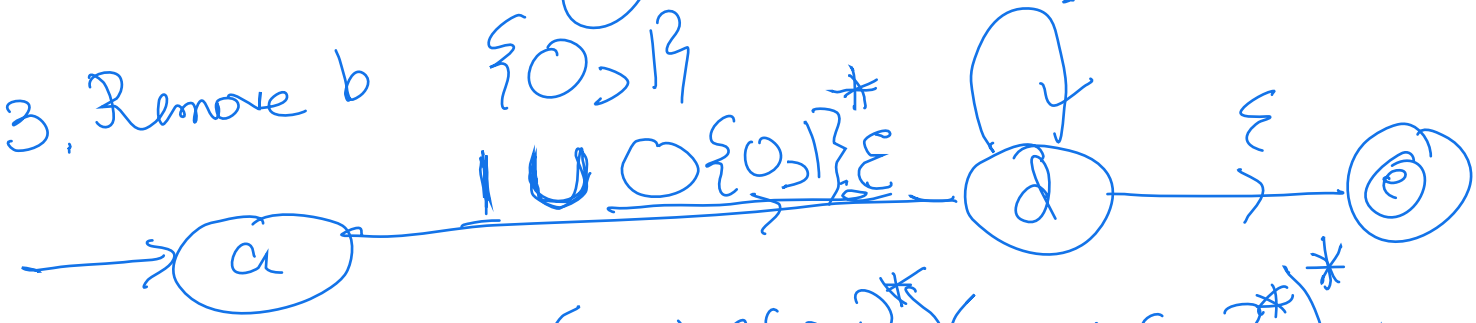
1. Create new accept state



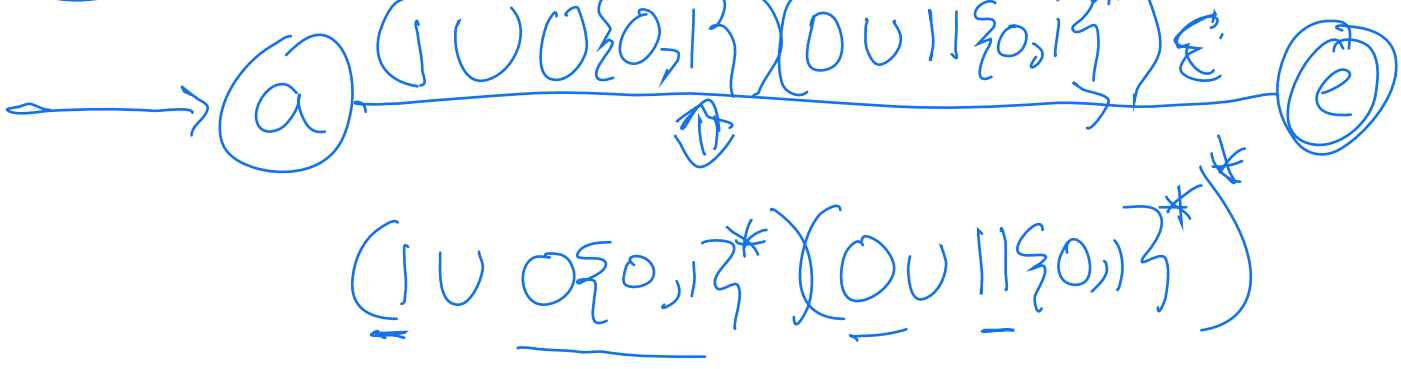
2. Remove c.



3. Remove b



4.



$(1 \cup 0\{0,1\}^*)(0 \cup 1\{0,1\}^*)^*$

Equivalence Core Exercises

1. For each of the following regular expressions, provide an NFA that accepts the language described by the regular expression. Do so by following the provided sequence of steps that is provided with each expression. When constructing NFA's that accept unions, concatenations, and stars of NFA's **that you have already constructed**, then use the techniques provided in this lecture and do *not* simplify the resulting NFA. Box and label the NFA and use it for any subsequent steps if necessary. When concatenating two NFA's make sure to "X out" the inner circle of any accepting state that is no longer accepting. Please do not erase the inner circle. Leave it as evidence that the state was previously an accepting state.

   (a) $a^* \cup (ab)^*$ using the following steps. Construct an NFA that accepts (the language described by)

      i. a^*
      ii. ab
      iii. $(ab)^*$
      iv. $a^* \cup (ab)^*$

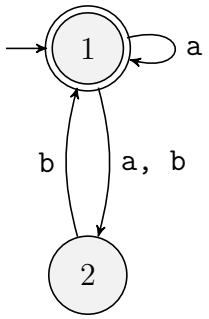
   (b) $a(abb)^* \cup b$ using the following steps. Construct an NFA that accepts (the language described by)

      i. a
      ii. $(abb)^*$
      iii. $a(abb)^*$
      iv. b
      v. $a(abb)^* \cup b$

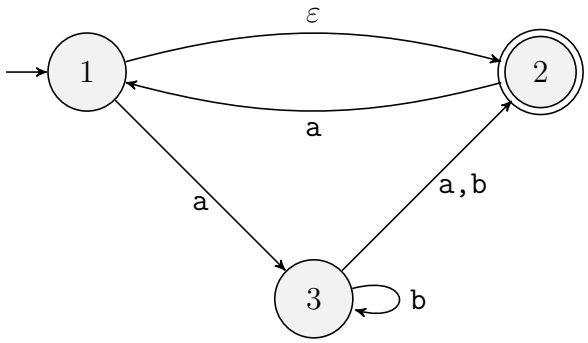
   (c) $((a \cup b^*)a^*)^*$ using the following steps. Construct an NFA that accepts (the language described by)

      i. a
      ii. b^*
      iii. $(a \cup b^*)$
      iv. a^*
      v. $(a \cup b^*)a^*$
      vi. $((a \cup b^*)a^*)^*$

2. Provide a DFA M that accepts the same language as that accepted by the NFA N that is defined in the following state diagram. Do so by defining the states of M to be subsets of the states of N.



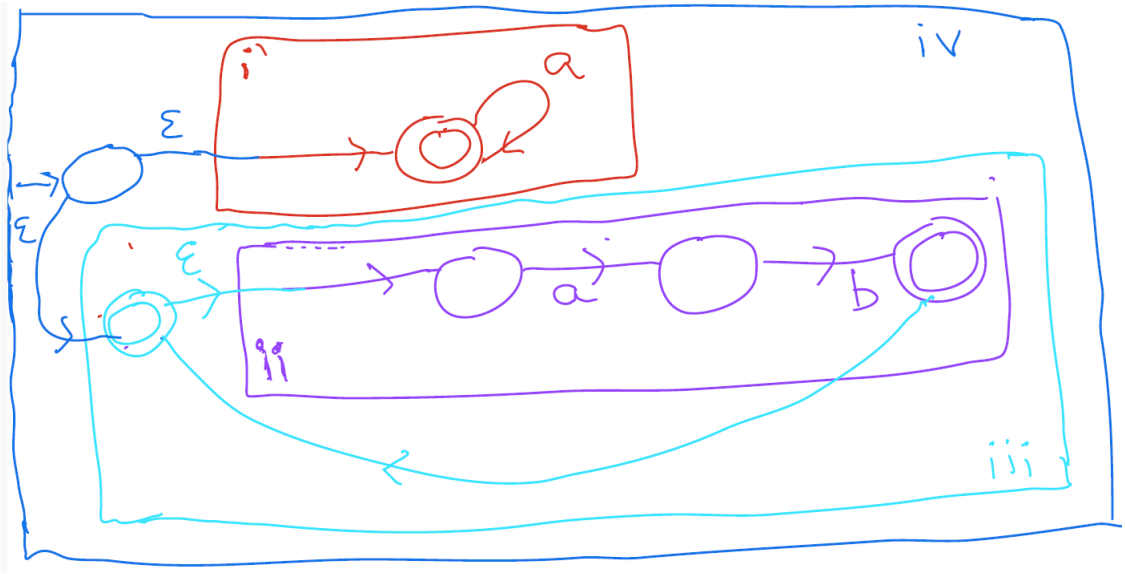
3. Provide a DFA M that accepts the same language as that accepted by the NFA N that is defined in the following state diagram. Do so by defining the states of M to be subsets of the states of N.



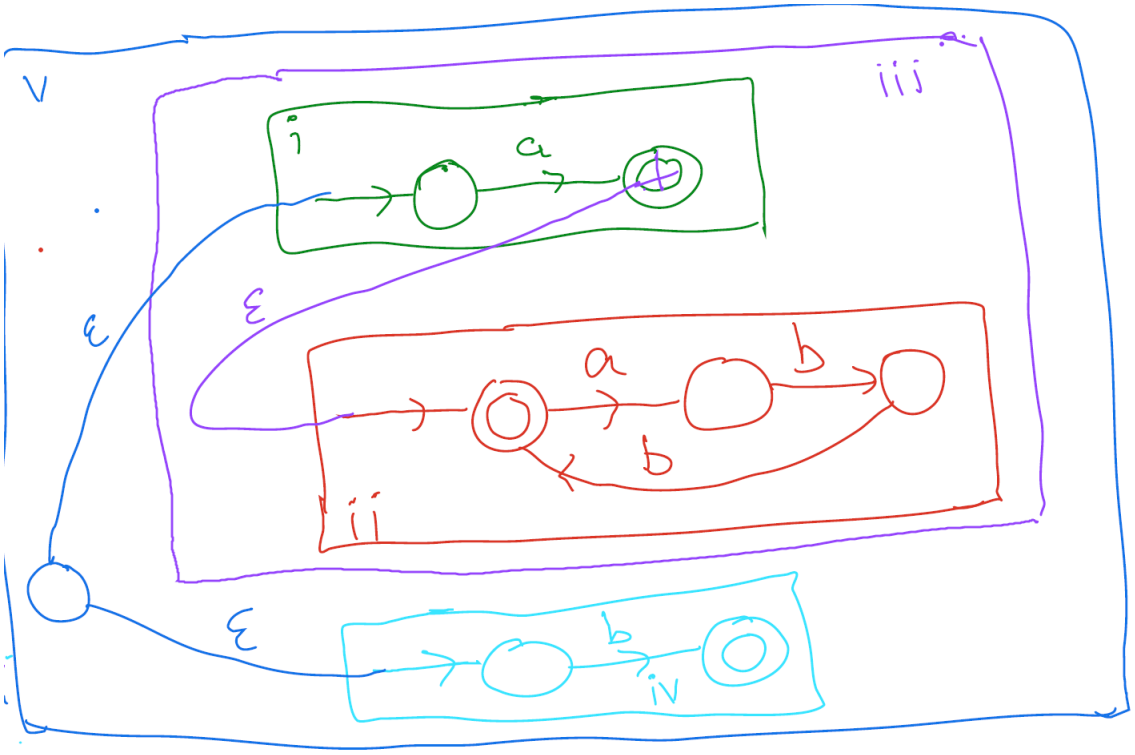
Solutions to Equivalence Core Exercises

1. We have the following NFA's.

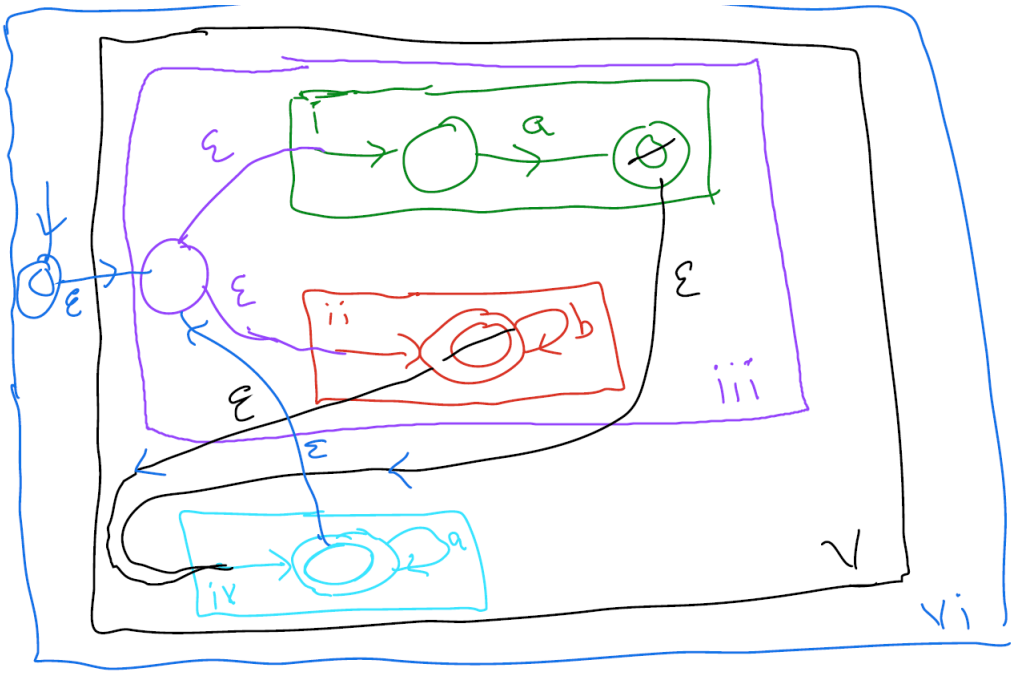
   (a) We have the following NFA.



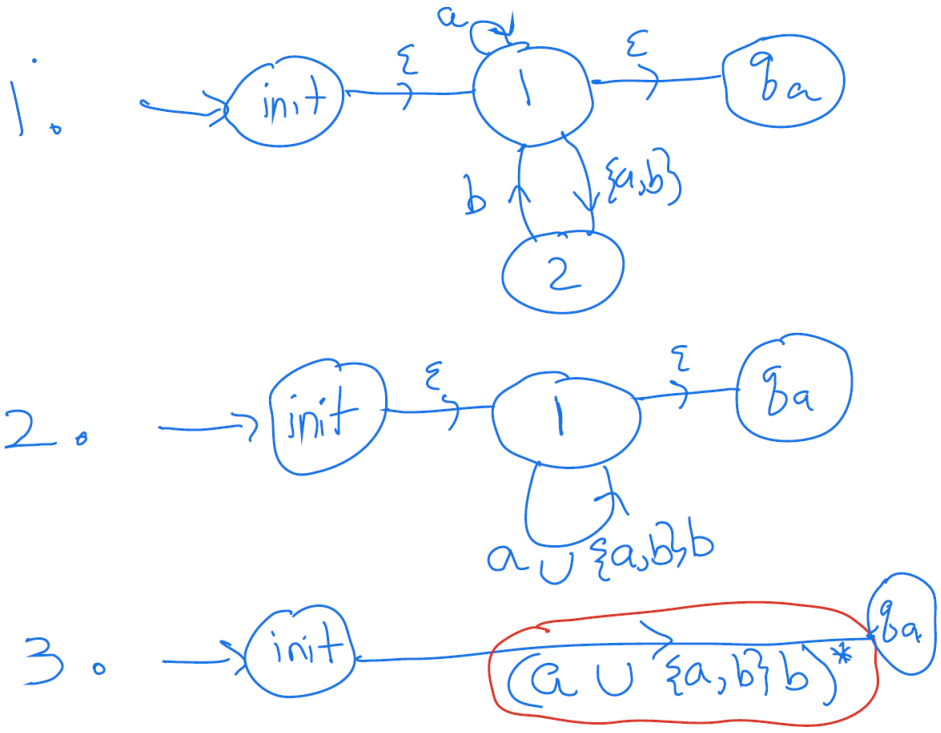
   (b) We have the following NFA.



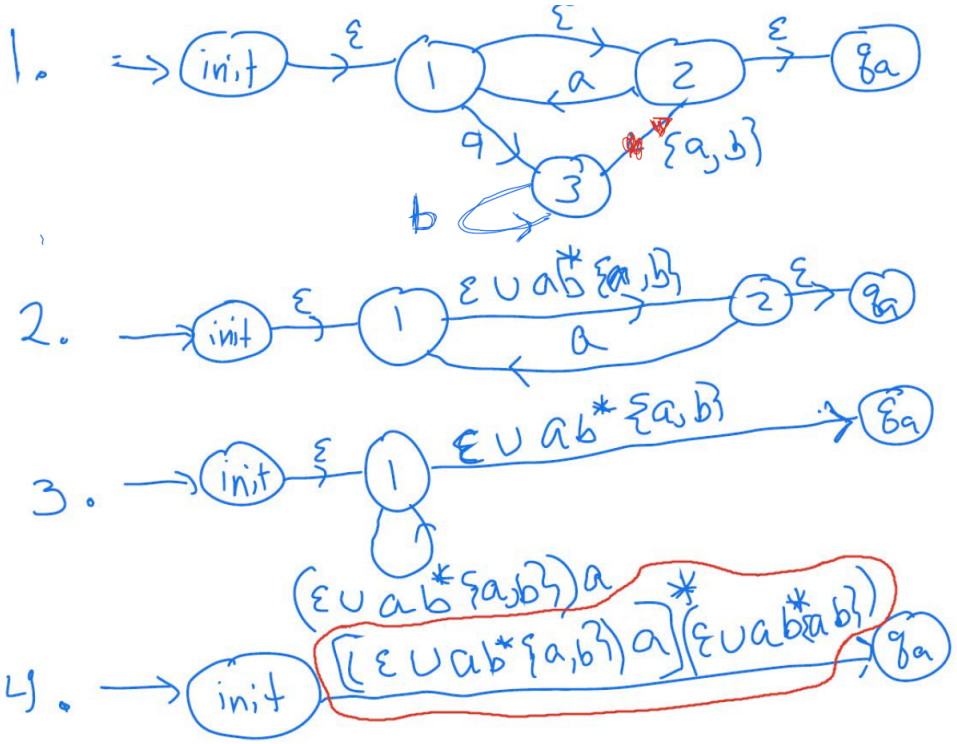
   (c) We have the following NFA.



2. We have the following sequence of GNFA transformations.



3. We have the following sequence of GNFA transformations.


1.
init
ε
1
ε
a
2
ε
qa
a
{a,b}
3
b
2.
init
ε
1
ε ∪ ab* {a,b}
a
2
ε
qa
3.
init
ε
1
ε ∪ ab* {a,b}
qa
4.
init
(ε ∪ ab* {a,b}) a
[(ε ∪ ab* {a,b}) a]*
(ε ∪ ab* {a,b})
qa