Computer Science

Dr. Y. Liow (April 5, 2024)

Contents

Chapter 12

Regular Languages

12.1 NFA to Regex

debug: nfa-to-regex.tex

The algorithm for converting an NFA N to a regex r such that

$$L(N) = L(r)$$

is actually pretty simple. Here's the algorithm.

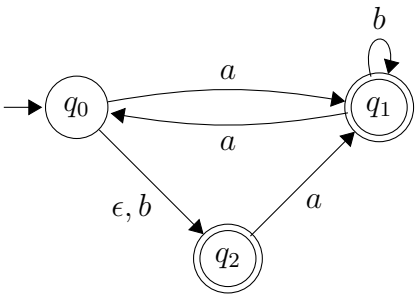
We're going to build a new NFA. Let's call this N'. In fact this NFA will not be exactly an NFA. N' will initially be N. We will make some changes.

First, in N', you create a new node, say i (for initial state) and join i to the start state of N with an ϵ–transition.

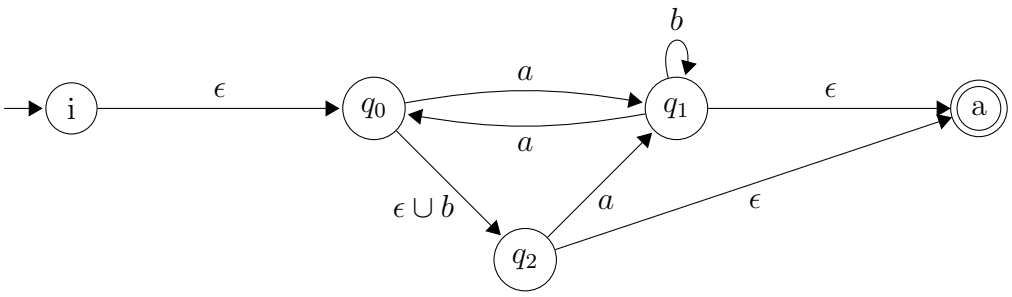
Second, you create a new node, say a (for accept state) and join all accept state of N to a by ϵ–transitions. You then make the accept states from N be non-accept states in N'.

Third, if from two states q, q' we see a, b, we replace this transition label to $a \cup b$, i.e., a regular expression. More generally if you see a list of characters from Σ of N, you replace it with the regular expression of the union of these characters.

We're not done yet, but let's look at an example right away. Consider this NFA N:



The new N' (again, I'm not done changing it yet):



Note that the state diagram is now *not* an NFA since the transitions are labeled with *regular expressions*.

generalized NFA
GNFA

The above is called a **generalized NFA GNFA**. The operation of such animals are obvious: You run them more or less like NFAs. If you read say a character a, then you follow transitions labeled with r if $a \in L(r)$. For instance you are allowed to following transitions labeled a or $a \cup b$ or $a^* \cup b$, etc.

It's clear how acceptance should be defined for such generalized NFAs.

By the way, if there's no transition from say q to q' in N, in the new generalized NFA, we add a transition labeled $\emptyset$ from q to q'. This means that you cannot go from q to q' since the language $L(\emptyset)$ is $\emptyset$.

Now you might ask: Why in the world would you want to make things even more complicated? First we have DFA. Then we have NFA. And now ... *generalized* NFA?!?

The reason is the same reason for generalizing DFA to NFA. The NFA gives us a great deal of flexibility when compared against the DFA. What about from NFA to generalized NFA?

If an NFA has a transition from q to q' labels a, b, then there are actually two transitions, one labeled a and one labeled b. In the case of the corresponding generalized NFA, there is only one transition labeled $a \cup b$.

Here's the point: It allows us to *simplify* NFAs. The above shows us that two transitions become one. Even more importantly, generalized NFA actually also provides us with the flexibility of simplifying states!!!

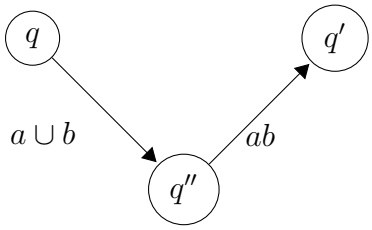
How is that?

Suppose you have a transition from q to q' labeled a and there's a transition from q' to q'' labeled b. Do you see that it's the same as combining the two transitions to one so that there's only one transition from q to q'' labeled ab.

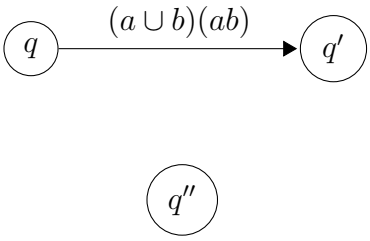
If no other transitions passed through q', do you see that now q' is useless and can be thrown away?

The main idea of the construction is to continually choose a node q in the generalized NFA other than i and a and remove q by "redirecting traffic flow" without changing the language accepted by the new generalized NFA.

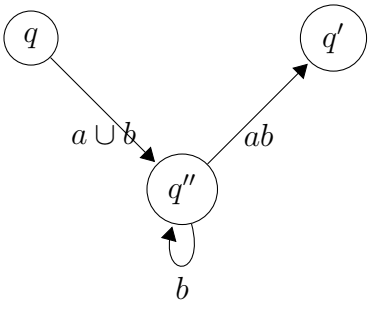
Let's look at an example. Suppose in a generalized NFA we have the following 3 states (there might be more):



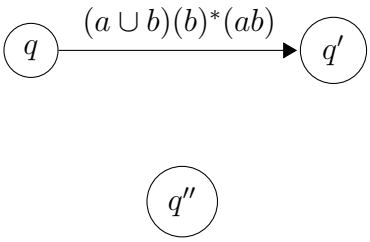
Clearly this can be replaced by



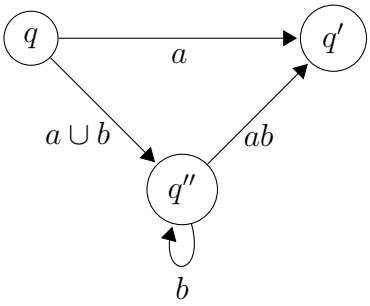
Wait ... what if there's "traffic" going from q'' to q'' , i.e., a loop? Say it's the generalized NFA before traffic redirection is this:



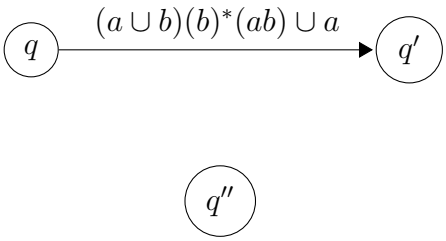
So the total "traffic" going from q to q' through q'' is $(a \cup b)(b)^*(ab)$, allowing the traffic from q'' to itself to loop as many times as we like (include none). And we get this simplification:



Wait a minute ... what if there's already a transition from q to q'? Say



In that case, the combined traffic from q to q' must be



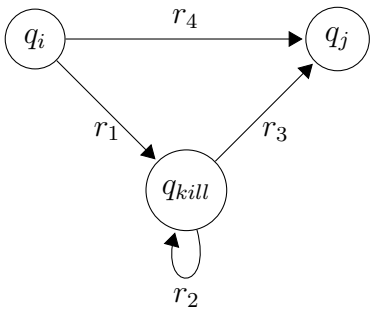
and in this case, the state q'' is redundant since now there is no "traffic" going through q''.

If we do the above for *all* possible q and q', then the resulting generalized NFA would have no transitions running through this lonely and isolated q''. This means that we can remove it. Right?

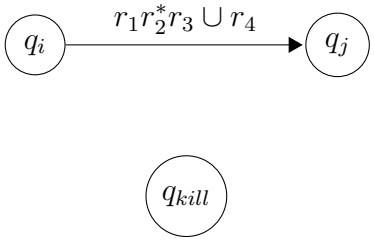
That's the whole idea.

At the end of the "state removal" operation, we would have only the two extra states i and a left. The regular expression from i to a is the required regular expression.

Formally, after the initial construction of the GNFA, you continually pick a state q_{kill} which is not i and not a and for every $q_i \neq q_{kill}$ and $q_j \neq q_{kill}$, with the following (relevant) regular expressions for transitions:

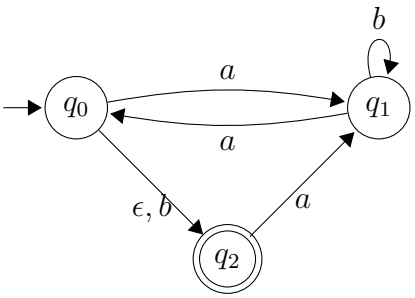


then this part of the GNFA can be replaced by the following without changing the resulting language accepted:



Exercise 12.1.1. Find a regex that accepts the same language as the following NFA:

debug: exercises/nfa-to-regex0/question.tex

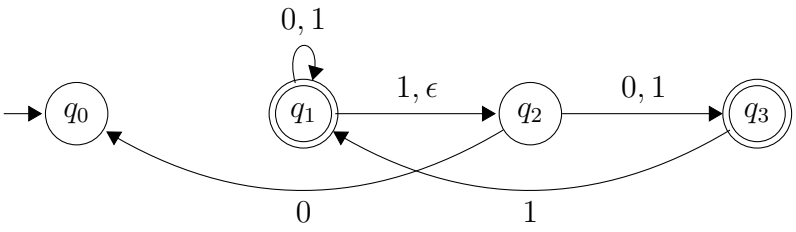


(Go to solution, page 11057) □

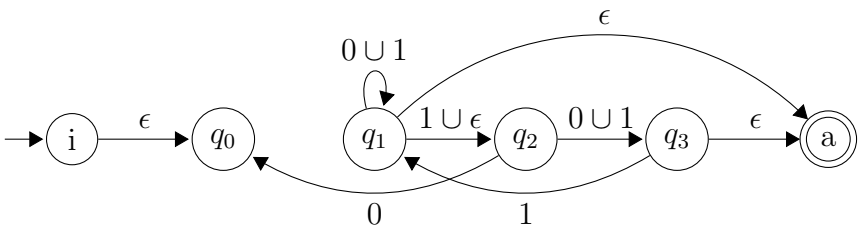
There are two examples on the regex construction for NFAs in the Sipser textbook. Make sure you try both of them.

The following are more examples.

Example. Here's an NFA:



Here's the initial GNFA:



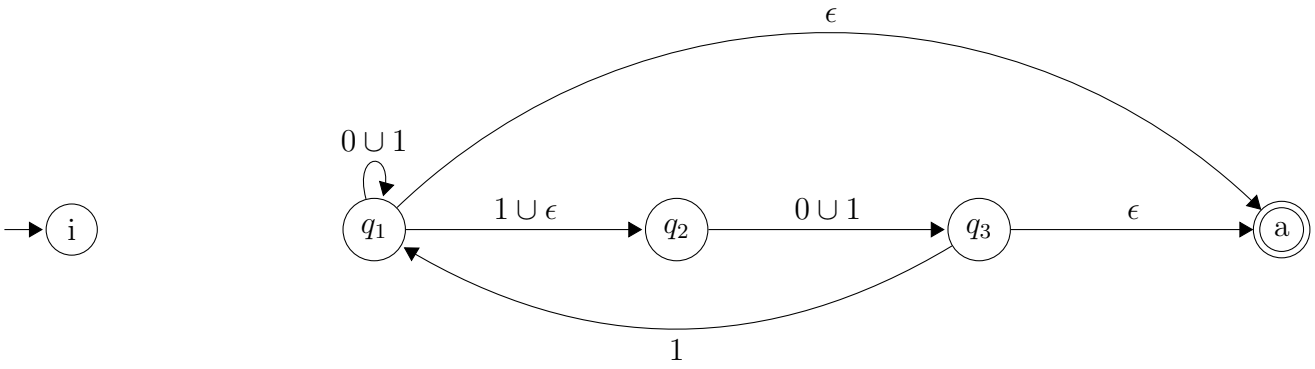
Each table below shows that computation of the resulting transistion after removing a state q_{kill}. In the third, fourth, and fifth columns,

$$r_1 = \delta(q_i, q_{kill})$$
$$r_2 = \delta(q_{kill}, q_{kill})$$
$$r_3 = \delta(q_{kill}, q_j)$$
$$r_4 = \delta(q_i, q_j)$$

Removal of q_{kill}=q_0:

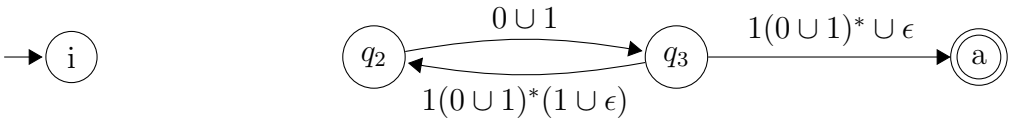
q_i	q_j	r_1	r_2	r_3	r_4	$r_1 r_2^* r_3 \cup r_4$
q_1	q_1	$\emptyset$	$\emptyset$	$\emptyset$	$0 \cup 1$	$0 \cup 1$
q_1	q_2	$\emptyset$	$\emptyset$	$\emptyset$	$1 \cup \epsilon$	$1 \cup \epsilon$
q_1	q_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
q_1	a	$\emptyset$	$\emptyset$	$\emptyset$	ϵ	ϵ
q_2	q_1	0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
q_2	q_2	0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
q_2	q_3	0	$\emptyset$	$\emptyset$	$0 \cup 1$	$0 \cup 1$
q_2	a	0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
q_3	q_1	$\emptyset$	$\emptyset$	$\emptyset$	1	1
q_3	q_2	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
q_3	q_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
q_3	a	$\emptyset$	$\emptyset$	$\emptyset$	ϵ	ϵ
i	q_1	ϵ	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

i	q_2	ϵ	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
i	q_3	ϵ	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
i	a	ϵ	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$



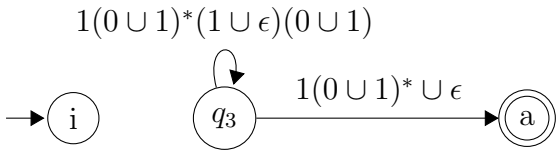
Removal of q_{kill}=q_1:

q_i	q_j	r_1	r_2	r_3	r_4	$r_1 r_2^* r_3 \cup r_4$
q_2	q_2	$\emptyset$	$0 \cup 1$	$1 \cup \epsilon$	$\emptyset$	$\emptyset$
q_2	q_3	$\emptyset$	$0 \cup 1$	$\emptyset$	$0 \cup 1$	$0 \cup 1$
q_2	a	$\emptyset$	$0 \cup 1$	ϵ	$\emptyset$	$\emptyset$
q_3	q_2	1	$0 \cup 1$	$1 \cup \epsilon$	$\emptyset$	$1(0 \cup 1)^*(1 \cup \epsilon)$
q_3	q_3	1	$0 \cup 1$	$\emptyset$	$\emptyset$	$\emptyset$
q_3	a	1	$0 \cup 1$	ϵ	ϵ	$1(0 \cup 1)^* \cup \epsilon$
i	q_2	$\emptyset$	$0 \cup 1$	$1 \cup \epsilon$	$\emptyset$	$\emptyset$
i	q_3	$\emptyset$	$0 \cup 1$	$\emptyset$	$\emptyset$	$\emptyset$
i	a	$\emptyset$	$0 \cup 1$	ϵ	$\emptyset$	$\emptyset$



Removal of q_{kill}=q_2:

q_i	q_j	r_1	r_2	r_3	r_4	$r_1 r_2^* r_3 \cup r_4$
q_3	q_3	$1(0 \cup 1)^*(1 \cup \epsilon)$	$\emptyset$	$0 \cup 1$	$\emptyset$	$1(0 \cup 1)^*(1 \cup \epsilon)(0 \cup 1)$
q_3	a	$1(0 \cup 1)^*(1 \cup \epsilon)$	$\emptyset$	$\emptyset$	$1(0 \cup 1)^* \cup \epsilon$	$1(0 \cup 1)^* \cup \epsilon$
i	q_3	$\emptyset$	$\emptyset$	$0 \cup 1$	$\emptyset$	$\emptyset$
i	a	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$



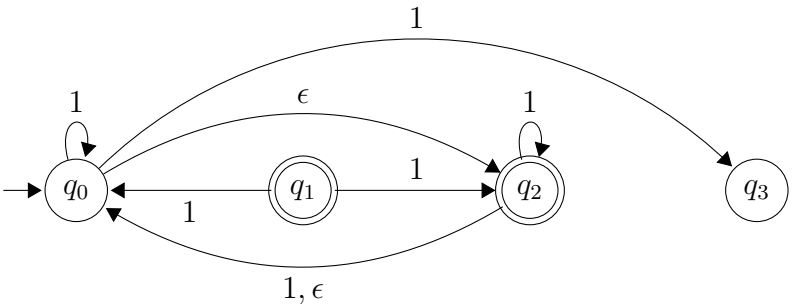
Removal of q_{kill}=q_3:

q_i	q_j	r_1	r_2	r_3	r_4	$r_1 r_2^* r_3 \cup r_4$
i	a	$\emptyset$	$1(0 \cup 1)^*(1 \cup \epsilon)(0 \cup 1)$	$1(0 \cup 1)^* \cup \epsilon$	$\emptyset$	$\emptyset$

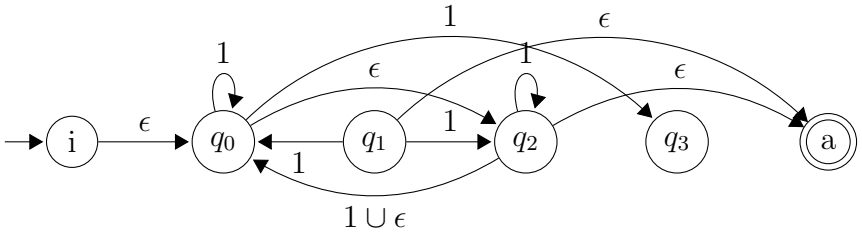
Therefore the regex that generates the same language that is accepted by the given NFA is

$$\emptyset$$

Example. Here's the NFA:



Here's the initial GNFA:



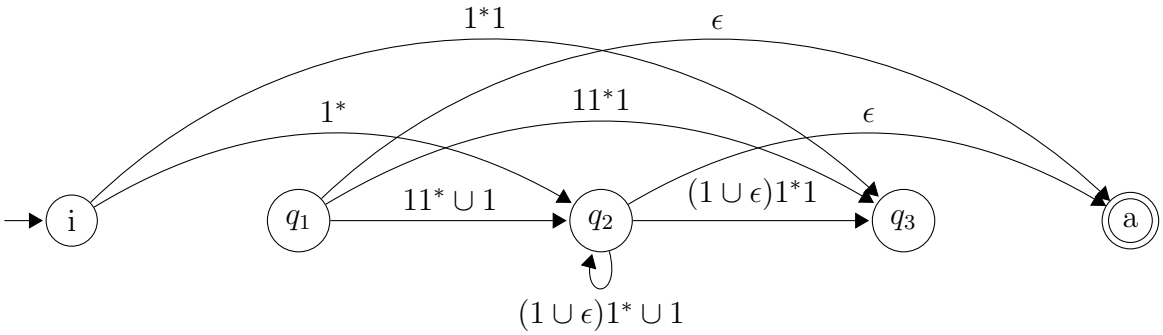
Each table below shows that computation of the resulting transistions after removing a state q_{kill}. In the third, fourth, and fifth columns,

$$r_1 = \delta(q_i, q_{kill})$$
$$r_2 = \delta(q_{kill}, q_{kill})$$
$$r_3 = \delta(q_{kill}, q_j)$$
$$r_4 = \delta(q_i, q_j)$$

Removal of q_{kill}=q_0:

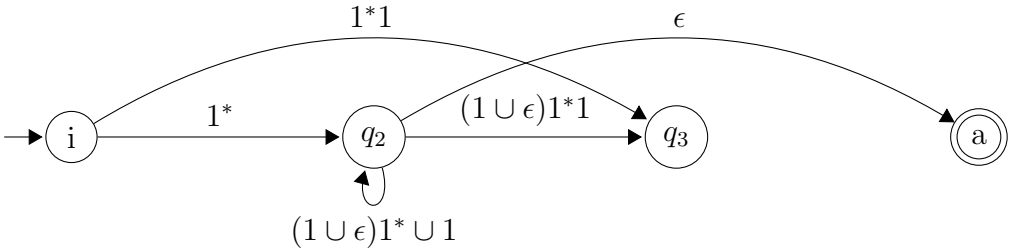
q_i	q_j	r_1	r_2	r_3	r_4	$r_1 r_2^* r_3 \cup r_4$
q_1	q_1	1	1	$\emptyset$	$\emptyset$	$\emptyset$
q_1	q_2	1	1	ϵ	1	$11^* \cup 1$
q_1	q_3	1	1	1	$\emptyset$	11^*1
q_1	a	1	1	$\emptyset$	ϵ	ϵ
q_2	q_1	$1 \cup \epsilon$	1	$\emptyset$	$\emptyset$	$\emptyset$
q_2	q_2	$1 \cup \epsilon$	1	ϵ	1	$(1 \cup \epsilon)1^* \cup 1$
q_2	q_3	$1 \cup \epsilon$	1	1	$\emptyset$	$(1 \cup \epsilon)1^*1$
q_2	a	$1 \cup \epsilon$	1	$\emptyset$	ϵ	ϵ
q_3	q_1	$\emptyset$	1	$\emptyset$	$\emptyset$	$\emptyset$
q_3	q_2	$\emptyset$	1	ϵ	$\emptyset$	$\emptyset$

q_3	q_3	$\emptyset$	1	1	$\emptyset$	$\emptyset$
q_3	a	$\emptyset$	1	$\emptyset$	$\emptyset$	$\emptyset$
i	q_1	ϵ	1	$\emptyset$	$\emptyset$	$\emptyset$
i	q_2	ϵ	1	ϵ	$\emptyset$	1^*
i	q_3	ϵ	1	1	$\emptyset$	1^*1
i	a	ϵ	1	$\emptyset$	$\emptyset$	$\emptyset$



Removal of q_{kill}=q_1:

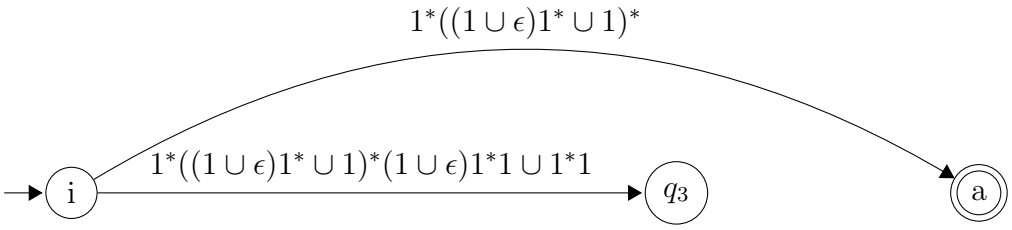
q_i	q_j	r_1	r_2	r_3	r_4	$r_1 r_2^* r_3 \cup r_4$
q_2	q_2	$\emptyset$	$\emptyset$	$11^* \cup 1$	$(1 \cup \epsilon)1^* \cup 1$	$(1 \cup \epsilon)1^* \cup 1$
q_2	q_3	$\emptyset$	$\emptyset$	11^*1	$(1 \cup \epsilon)1^*1$	$(1 \cup \epsilon)1^*1$
q_2	a	$\emptyset$	$\emptyset$	ϵ	ϵ	ϵ
q_3	q_2	$\emptyset$	$\emptyset$	$11^* \cup 1$	$\emptyset$	$\emptyset$
q_3	q_3	$\emptyset$	$\emptyset$	11^*1	$\emptyset$	$\emptyset$
q_3	a	$\emptyset$	$\emptyset$	ϵ	$\emptyset$	$\emptyset$
i	q_2	$\emptyset$	$\emptyset$	$11^* \cup 1$	1^*	1^*
i	q_3	$\emptyset$	$\emptyset$	11^*1	1^*1	1^*1
i	a	$\emptyset$	$\emptyset$	ϵ	$\emptyset$	$\emptyset$



Removal of q_{kill}=q_2:

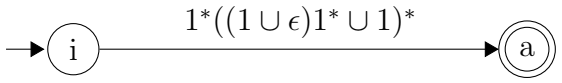
q_i	q_j	r_1	r_2	r_3	r_4	$r_1 r_2^* r_3 \cup r_4$
q_3	q_3	$\emptyset$	$(1 \cup \epsilon)1^* \cup 1$	$(1 \cup \epsilon)1^*1$	$\emptyset$	$\emptyset$

q_3	a	$\emptyset$	$(1 \cup \epsilon)1^* \cup 1$	ϵ	$\emptyset$	$\emptyset$
i	q_3	1^*	$(1 \cup \epsilon)1^* \cup 1$	$(1 \cup \epsilon)1^*1$	1^*1	$1^*((1 \cup \epsilon)1^* \cup 1)^*(1 \cup \epsilon)1^*1 \cup 1^*1$
i	a	1^*	$(1 \cup \epsilon)1^* \cup 1$	ϵ	$\emptyset$	$1^*((1 \cup \epsilon)1^* \cup 1)^*$



Removal of q_{kill}=q_3:

q_i	q_j	r_1	r_2	r_3	r_4	$r_1 r_2^* r_3 \cup r_4$
i	a	$1^*((1 \cup \epsilon)1^* \cup 1)^*(1 \cup \epsilon)1^*1 \cup 1^*1$	$\emptyset$	$\emptyset$	$1^*((1 \cup \epsilon)1^* \cup 1)^*$	$1^*((1 \cup \epsilon)1^* \cup 1)^*$



Therefore the regex that generates the same language that is accepted by the given NFA is

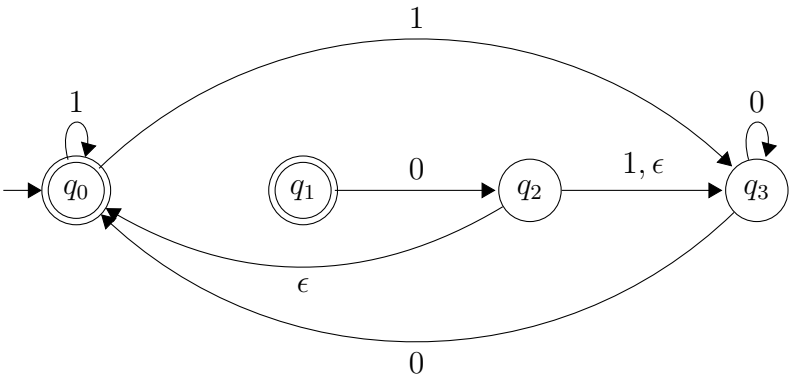
$$1^*((1 \cup \epsilon)1^* \cup 1)^*$$

You can simplify the above regex as follows:

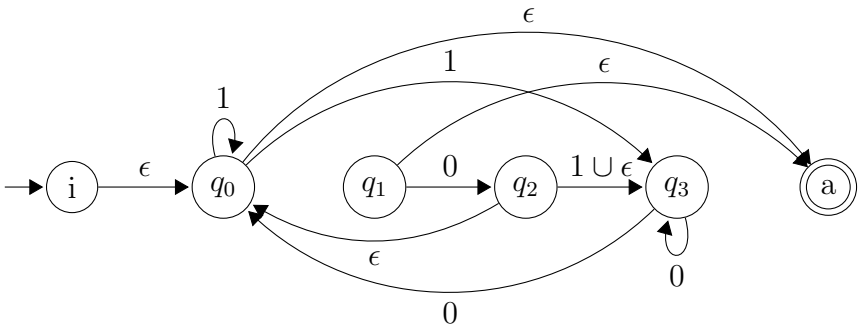
$$\begin{aligned}
1^*((1 \cup \epsilon)1^* \cup 1)^* &= 1^*((11^* \cup \epsilon 1^*)^* \cup 1)^* \\
&= 1^*((11^* \cup 1^*)^* \cup 1)^* \\
&= 1^*((1^*)^* \cup 1)^* \\
&= 1^*(1^* \cup 1)^* \\
&= 1^*(1^*)^* \\
&= 1^*1^* \\
&= 1^*
\end{aligned}$$

If you look at the original NFA, you will see that the language accepted by the NFA is indeed $\{1\}^*$.

Example. Here's the NFA:



Here's the initial GNFA:



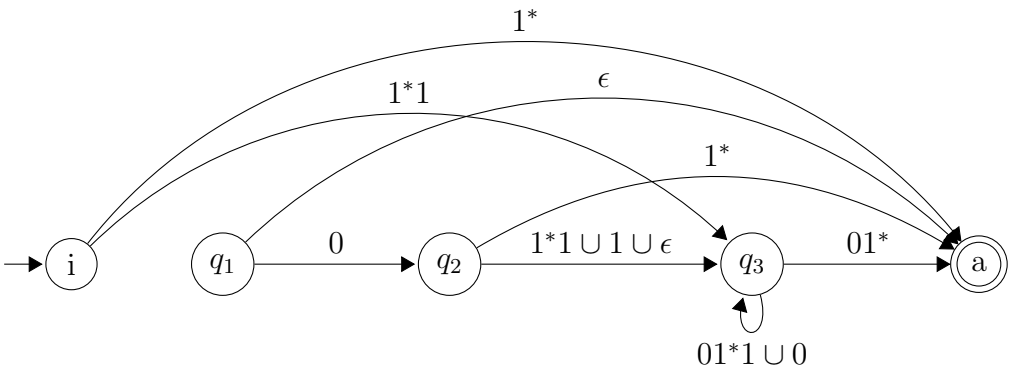
Each table below shows that computation of the resulting transistions after removing a state q_{kill}. In the third, fourth, and fifth columns,

$$r_1 = \delta(q_i, q_{kill})$$
$$r_2 = \delta(q_{kill}, q_{kill})$$
$$r_3 = \delta(q_{kill}, q_j)$$
$$r_4 = \delta(q_i, q_j)$$

Removal of q_{kill}=q_0:

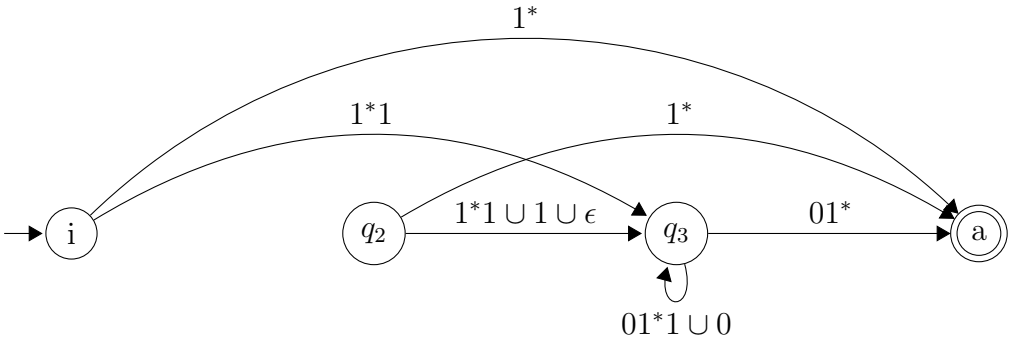
q_i	q_j	r_1	r_2	r_3	r_4	$r_1 r_2^* r_3 \cup r_4$
q_1	q_1	$\emptyset$	1	$\emptyset$	$\emptyset$	$\emptyset$
q_1	q_2	$\emptyset$	1	$\emptyset$	0	0
q_1	q_3	$\emptyset$	1	1	$\emptyset$	$\emptyset$
q_1	a	$\emptyset$	1	ϵ	ϵ	ϵ
q_2	q_1	ϵ	1	$\emptyset$	$\emptyset$	$\emptyset$
q_2	q_2	ϵ	1	$\emptyset$	$\emptyset$	$\emptyset$

q_2	q_3	ϵ	1	1	$1 \cup \epsilon$	$1^*1 \cup 1 \cup \epsilon$
q_2	a	ϵ	1	ϵ	$\emptyset$	1^*
q_3	q_1	0	1	$\emptyset$	$\emptyset$	$\emptyset$
q_3	q_2	0	1	$\emptyset$	$\emptyset$	$\emptyset$
q_3	q_3	0	1	1	0	$01^*1 \cup 0$
q_3	a	0	1	ϵ	$\emptyset$	01^*
i	q_1	ϵ	1	$\emptyset$	$\emptyset$	$\emptyset$
i	q_2	ϵ	1	$\emptyset$	$\emptyset$	$\emptyset$
i	q_3	ϵ	1	1	$\emptyset$	1^*1
i	a	ϵ	1	ϵ	$\emptyset$	1^*



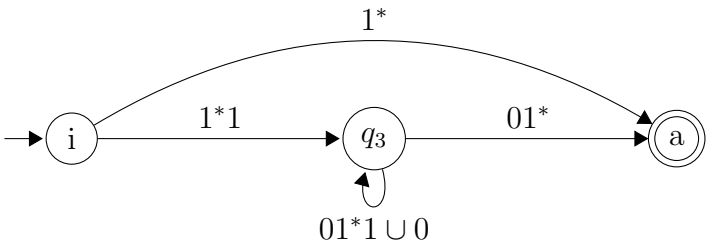
Removal of q_{kill}=q_1:

q_i	q_j	r_1	r_2	r_3	r_4	$r_1 r_2^* r_3 \cup r_4$
q_2	q_2	$\emptyset$	$\emptyset$	0	$\emptyset$	$\emptyset$
q_2	q_3	$\emptyset$	$\emptyset$	$\emptyset$	$1^*1 \cup 1 \cup \epsilon$	$1^*1 \cup 1 \cup \epsilon$
q_2	a	$\emptyset$	$\emptyset$	ϵ	1^*	1^*
q_3	q_2	$\emptyset$	$\emptyset$	0	$\emptyset$	$\emptyset$
q_3	q_3	$\emptyset$	$\emptyset$	$\emptyset$	$01^*1 \cup 0$	$01^*1 \cup 0$
q_3	a	$\emptyset$	$\emptyset$	ϵ	01^*	01^*
i	q_2	$\emptyset$	$\emptyset$	0	$\emptyset$	$\emptyset$
i	q_3	$\emptyset$	$\emptyset$	$\emptyset$	1^*1	1^*1
i	a	$\emptyset$	$\emptyset$	ϵ	1^*	1^*



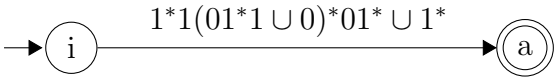
Removal of q_{kill}=q_2:

q_i	q_j	r_1	r_2	r_3	r_4	$r_1 r_2^* r_3 \cup r_4$
q_3	q_3	$\emptyset$	$\emptyset$	$1^*1 \cup 1 \cup \epsilon$	$01^*1 \cup 0$	$01^*1 \cup 0$
q_3	a	$\emptyset$	$\emptyset$	1^*	01^*	01^*
i	q_3	$\emptyset$	$\emptyset$	$1^*1 \cup 1 \cup \epsilon$	1^*1	1^*1
i	a	$\emptyset$	$\emptyset$	1^*	1^*	1^*



Removal of q_{kill}=q_3:

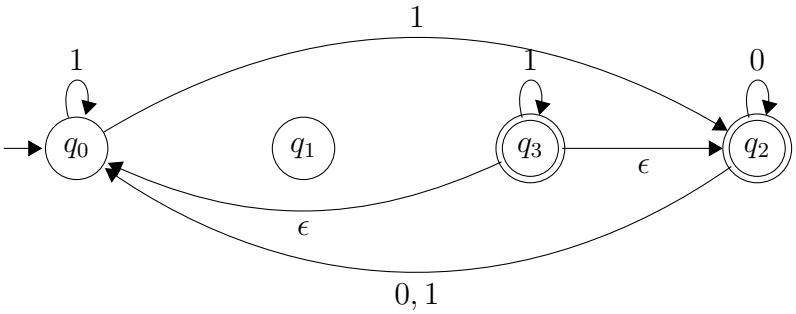
q_i	q_j	r_1	r_2	r_3	r_4	$r_1 r_2^* r_3 \cup r_4$
i	a	1^*1	$01^*1 \cup 0$	01^*	1^*	$1^*1(01^*1 \cup 0)^*01^* \cup 1^*$



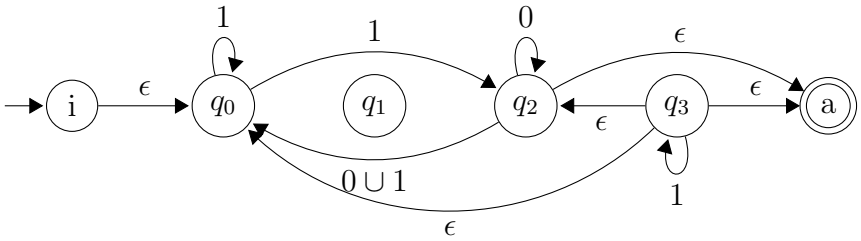
Therefore the regex that generates the same language that is accepted by the given NFA is

$$1^*1(01^*1 \cup 0)^*01^* \cup 1^*$$

Example. Here's the NFA:



Here's the initial GNFA:



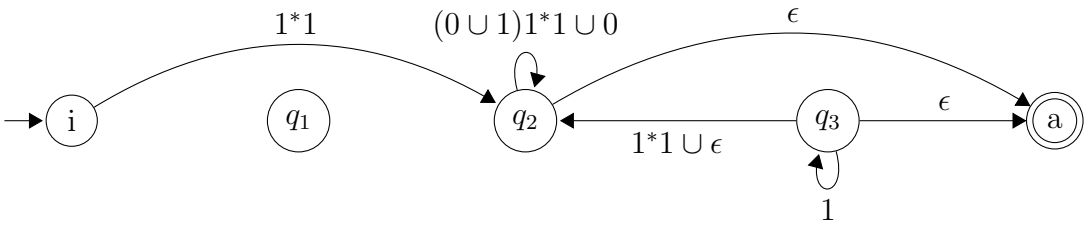
Each table below shows that computation of the resulting transistions after removing a state q_{kill}. In the third, fourth, and fifth columns,

$$r_1 = \delta(q_i, q_{kill})$$
$$r_2 = \delta(q_{kill}, q_{kill})$$
$$r_3 = \delta(q_{kill}, q_j)$$
$$r_4 = \delta(q_i, q_j)$$

Removal of q_{kill}=q_0:

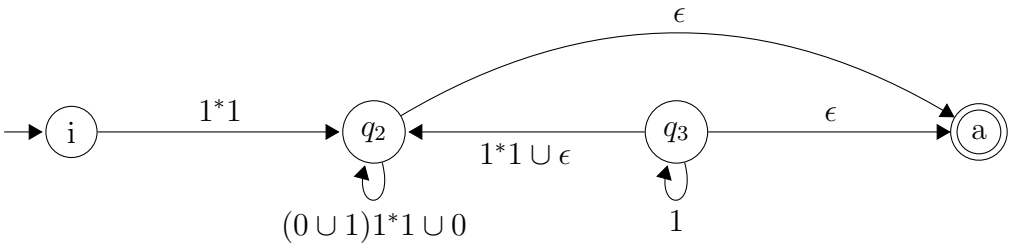
q_i	q_j	r_1	r_2	r_3	r_4	$r_1 r_2^* r_3 \cup r_4$
q_1	q_1	$\emptyset$	1	$\emptyset$	$\emptyset$	$\emptyset$
q_1	q_2	$\emptyset$	1	1	$\emptyset$	$\emptyset$
q_1	q_3	$\emptyset$	1	$\emptyset$	$\emptyset$	$\emptyset$
q_1	a	$\emptyset$	1	$\emptyset$	$\emptyset$	$\emptyset$
q_2	q_1	$0 \cup 1$	1	$\emptyset$	$\emptyset$	$\emptyset$
q_2	q_2	$0 \cup 1$	1	1	0	$(0 \cup 1)1^*1 \cup 0$
q_2	q_3	$0 \cup 1$	1	$\emptyset$	$\emptyset$	$\emptyset$
q_2	a	$0 \cup 1$	1	$\emptyset$	ϵ	ϵ
q_3	q_1	ϵ	1	$\emptyset$	$\emptyset$	$\emptyset$
q_3	q_2	ϵ	1	1	ϵ	$1^*1 \cup \epsilon$

q_3	q_3	ϵ	1	$\emptyset$	1	1
q_3	a	ϵ	1	$\emptyset$	ϵ	ϵ
i	q_1	ϵ	1	$\emptyset$	$\emptyset$	$\emptyset$
i	q_2	ϵ	1	1	$\emptyset$	1^*1
i	q_3	ϵ	1	$\emptyset$	$\emptyset$	$\emptyset$
i	a	ϵ	1	$\emptyset$	$\emptyset$	$\emptyset$



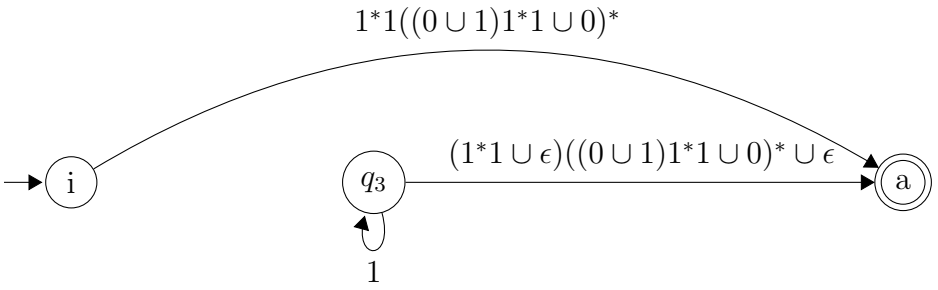
Removal of q_{kill}=q_1:

q_i	q_j	r_1	r_2	r_3	r_4	$r_1r_2^*r_3 \cup r_4$
q_2	q_2	$\emptyset$	$\emptyset$	$\emptyset$	$(0 \cup 1)1^*1 \cup 0$	$(0 \cup 1)1^*1 \cup 0$
q_2	q_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
q_2	a	$\emptyset$	$\emptyset$	$\emptyset$	ϵ	ϵ
q_3	q_2	$\emptyset$	$\emptyset$	$\emptyset$	$1^*1 \cup \epsilon$	$1^*1 \cup \epsilon$
q_3	q_3	$\emptyset$	$\emptyset$	$\emptyset$	1	1
q_3	a	$\emptyset$	$\emptyset$	$\emptyset$	ϵ	ϵ
i	q_2	$\emptyset$	$\emptyset$	$\emptyset$	1^*1	1^*1
i	q_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
i	a	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$



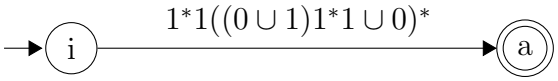
Removal of q_{kill}=q_2:

q_i	q_j	r_1	r_2	r_3	r_4	$r_1r_2^*r_3 \cup r_4$
q_3	q_3	$1^*1 \cup \epsilon$	$(0 \cup 1)1^*1 \cup 0$	$\emptyset$	1	1
q_3	a	$1^*1 \cup \epsilon$	$(0 \cup 1)1^*1 \cup 0$	ϵ	ϵ	$(1^*1 \cup \epsilon)((0 \cup 1)1^*1 \cup 0)^* \cup \epsilon$
i	q_3	1^*1	$(0 \cup 1)1^*1 \cup 0$	$\emptyset$	$\emptyset$	$\emptyset$
i	a	1^*1	$(0 \cup 1)1^*1 \cup 0$	ϵ	$\emptyset$	$1^*1((0 \cup 1)1^*1 \cup 0)^*$



Removal of q_{kill}=q_3:

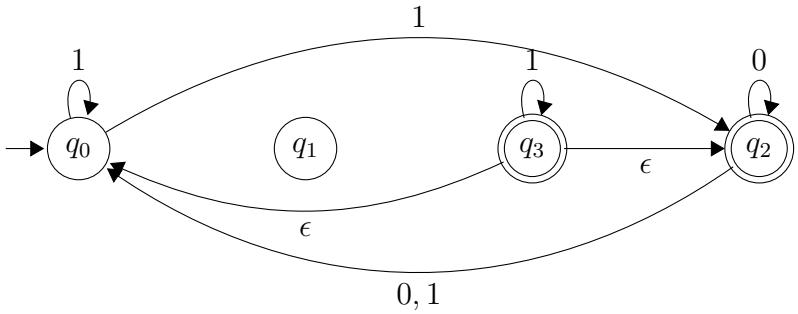
q_i	q_j	r_1	r_2	r_3	r_4	$r_1 r_2^* r_3 \cup r_4$
i	a	$\emptyset$	1	$(1^*1 \cup \epsilon)((0 \cup 1)1^*1 \cup 0)^* \cup \epsilon$	$1^*1((0 \cup 1)1^*1 \cup 0)^*$	$1^*1((0 \cup 1)1^*1 \cup 0)^*$



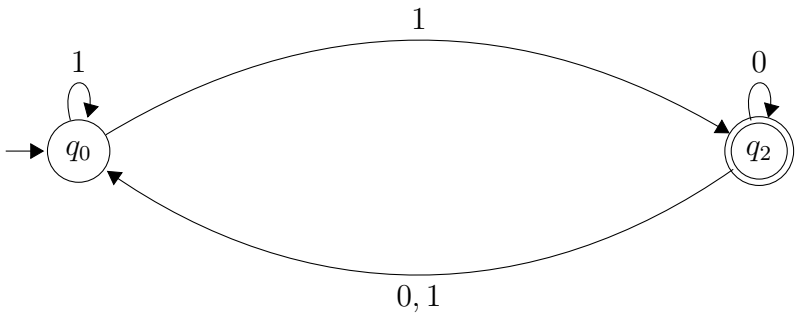
Therefore the regex that generates the same language that is accepted by the given NFA is

$$1^*1((0 \cup 1)1^*1 \cup 0)^*$$

If you look at the original NFA,



and you remove the unreachable states, you would get



You see that the most general path that reaches from q_0 to q_2 without leaving q_2 has a regex of

$$1^*1$$

Intuitively, after that, if you *leave* q_0, you can still get back to q_2, if you do

$$0$$

or

$$(0 \cup 1)1^*1$$

Intuitively, the most general path is then

$$1^*1(0 \cup (0 \cup 1)1^*1)^*$$

which is exactly what we get using the algorithm.

To give a mathematical proof, all you need to do is follow the idea behind the above examples.

First you have to define mathematically the concept of a GNFA, together with language acceptance of an GNFA.

Exercise 12.1.2. Define formally the concept of a GNFA and define what is meant a a word begin accepted by a GNFA. (Go to solution, page 11058) □

debug: exercises/nfa-to-regex1/question.tex

Next, you have to prove that given a GNFA N, the above operation of removing a state to obtain a new GNFA N' does not change the language being accepted, i.e.,

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

You can then use induction on the number of states in a GNFA to prove the original statement above.

Exercise 12.1.3. Prove the main theorem of this section. (Go to solution, page 11059) □

debug: exercises/nfa-to-regex2/question.tex

12.2 Regex to NFA

Now let me talk about converting a regex to an NFA, i.e., if r is a regex, I want to build an NFA N such that

$$L(N) = L(r)$$

This is actually pretty easy.

Let's try an example. Let

$$r = (a^* \cup bba)^*(bab \cup ab)^*$$

Then the language generated by r is

$$\begin{aligned} L(r) &= L((a^* \cup bba)^*(bab \cup ab)^*) \\ &= L((a^* \cup bba)^*) \cdot L((bab \cup ab)^*) \end{aligned}$$

At this point, we see that is we can build an NFA N_1 to accept

$$L((a^* \cup bba)^*)$$

and another NFA N_2 to accept

$$L((bab \cup ab)^*)$$

then, we just use the concatenation construction on N_1 and N_2, i.e., the new NFA will have the combination of N_1 followed by N_2 where the accept states of N_1 is joined to the start state of N_2 with ϵ-transitions.

Focusing on

$$L((a^* \cup bba)^*)$$

we see that

$$L((a^* \cup bba)^*) = L(a^* \cup bba)^*$$

So if we can build an NFA N_3 to accept

$$L(a^* \cup bba)$$

then can use the Kleene star construction on this NFA to get N_1, i.e., N_1 is built from N_3 where the start state of N_3 is changed to an accept state and the original accept state of N_3 have ϵ-transitions to this new start state. Instead

of doing this breakdown one step at a time, I think you see now that

$$L(a^* \cup bba) = L(a)^* \cup L(bba)$$

can be accepted by this NFA:

This NFA accepts

$$L(a^* \cup bba) = L(a)^* \cup L(bba)$$

Now using the Kleene star construction, we get

which accepts

$$L((a^* \cup bba)^*) = L(a^* \cup bba)^*$$

Let me simplify this a little before we continue. First I do this:

then this:

and then this:

Now for

$$L((bab \cup ab)^*) = L(bab \cup ab)^*$$

An NFA accepting this language is

I'm going to simplify it to

and then this

And finally when I concat the two machines I get

Exercise 12.2.1. Construct an NFA N such

debug: exercises/regex-to-nfa0/question.tex

$$L(N) = L(r)$$

where r is the regular expression

$$r = a(ba \cup abb \cup (ab)^*)^*$$

(Go to solution, page 11060) □

The big picture is this: Regular expressions are constructed starting from the symbols in Σ or $\emptyset$ (and you know how to construct NFAs for all these). You combine regular expressions recursively using unions, concatenation, and Kleene star (and you know how to construct the unions, concatenations, and Kleene stars of NFAs). Therefore the languages accepted by regex must be contained in the languages accepted by NFAs.

That's the main idea.

Formally, the right strategy is to proof the following statement

> If r is a regular expression, then there is an NFA N such that $L(N) = L(r)$.

by mathematical induction on the length of the regex (remember: a regex is a string.) In other words you should form this:

> $P(n)$: If r is a regular expression of length n, then there is an NFA N such that $L(N) = L(r)$.

You then prove this by induction. The base case starts at $n = 1$ since a regex have at least one character. The base cases are

$$r = c$$

where $c \in \Sigma$ or

$$r = \emptyset$$

It's easy to construct NFAs for these regex (of course!). So the base case is ... DONE!

Now you assume $P(1), P(2), \ldots, P(n)$ is true for some $n \geq 1$. Now you want to prove $P(n+1)$, i.e., given any regex r of length $n+1$, you want to prove that there is an NFA N such that

$$L(N) = L(r)$$

You can of course assume $n \geq 1$ since we're done with the base case. Now what?

Well, since the length of r is $n + 1 > 1$, r must be either

$$r = s^*$$

where s is a regex, or

$$r = st$$

where s and t are regexes, or

$$r = s \cup t$$

where s and t are regexes. (For simplicity, I'm ignoring the case where the regex has parenthesis, i.e., r is (s) – this is an easy case. So I'll leave that to you.) By definition, there are no other cases. Note that in the first case the length of s is $\leq n$ (in fact exactly equal to n). In the second case, the lengths of s and t are both $\leq n$. Likewise in the third case, the lengths of s and t are both $\leq n$. By induction hypothesis, for all cases, the s or the t (depending on the cases) are accepted by some NFAs N_1 and N_2.

In the first case, since

$$L(N_1) = L(s)$$

we have by definition

$$L(N_1)^* = L(s)^* = L(s^*)$$

We also know that the Kleene star operator is a closed operator. Since $L(N_1)$ is a regular language, this means that $L(N_1)^*$ is also regular. This means that there is some NFA/DFA N_1' such that

$$L(N_1') = L(N_1)^*$$

Altogether we have found some NFA/DFA N_1' such that

$$L(N_1') = L(N_1)^* = L(s^*) = L(r)$$

We have proven $P(n+1)$ for this case. All other cases are similar. (Go over the proof yourself.)

Therefore $P(n+1)$ is true for all cases.

By mathematical induction, $P(n)$ is true for all $n \geq 1$.

Solutions

Solution to Exercise 12.1.1.

debug: exercises/nfa-to-regex0/answer.tex

Solution not provided.

Solution to Exercise 12.1.2.

debug: exercises/nfa-to-regex1/answer.tex

Solution not provided.

Solution to Exercise 12.1.3.

debug: exercises/nfa-to-regex2/answer.tex

Solution not provided.

Solution to Exercise 12.2.1.

debug:
exercises/regex-to-nfa0/answer.tex

Solution not provided.

12.3 Myhill–Nerode theorem for regularity and DFA minimization

debug: minimization.tex

PUT SOMEWHERE: Go ahead and review equivalence relations.

Here's the aim: Let M be a DFA. Can we find another DFA M' such that

- $L(M') = L(M)$
- M' is a DFA with the smallest number of states satisfying the above.

In other words I want to minimize M.

Why would you want to do that? Because a smaller DFA will execute faster and furthermore will require less memory (states are memory) and therefore cheaper to produce. This is the case whether the DFA is a piece of software or a piece of hardware. Therefore DFA minimization improves runtime, space, and cost.

I will say that two automatas are **equivalent** (whether they are DFAs or NFAs) if they accept the same language.

equivalent

Exercise 12.3.1. Create three equivalent DFAs where one has two states, one has 3 states, and the last has 5 states. Prove that the one with two states is the minimal, i.e., you cannot find another DFA that accepts the same language and has 1 state. You have 2 minutes. (Go to solution, page 11061) □

debug: exercises/minimization0/question.tex

The main result is this:

Theorem 12.3.1. *For any DFA, there is a minimum equivalent DFA that is unique up to relabeling of states.*

The result is not just "there is a minimal DFA". There are algorithms for converting a given DFA to a smallest one. I don't have time to talk about all of them. I'll focus on just one. And before explaining the algorithm, I'll do one example first so you get the main idea. Before that, I'll talk about isomorphism of DFAs.

In computer science and math, two objects of the same kind is said to be isomorphic (example: graph isomorphism) if they are essentially the same except for some renaming of the internals of the objects. You can also define isomorphism of DFAs. In that case, you can restate the theorem as saying that the minimal DFA exist and is unique up to isomorphism. Here's the

formal definition of DFA isomorphism: Two DFAs $M = (\Sigma, Q, q_0, F, \delta)$ and $M' = (\Sigma, Q', q_0', F', \delta')$ are **isomorphic** if there is a bijection

$$f : Q \to Q'$$

(f is the renaming) such that

$$f(q_0) = q_0'$$
$$f(F) = F'$$

and

$$\delta(q_i, x) = q_j \iff \delta(f(q_i), x) = f(q_j)$$

You can see that f is just a re-labeling of states so that the "behavior" of the states remain the same. For instance

$$f(q_0) = q_0'$$

means that the start state of M has been renamed as the start state of M'. The condition

$$f(F) = F'$$

means that if a state q in M is an accept state, then after being renamed to $f(q)$ in M', $f(q)$ is also an accept state in M'. The transition function renaming is a little bit more complicated. The condition

$$\delta(q_i, x) = q_j \iff \delta(f(q_i), x) = f(q_j)$$

means that if at state q_i, on reading a character x, you transition to q_j, then after renaming q_i to $f(q_i)$ and q_j to $f(q_j)$, at state $f(q_i)$, on reading x, you also transition to state $f(q_j)$ (and vice versa).

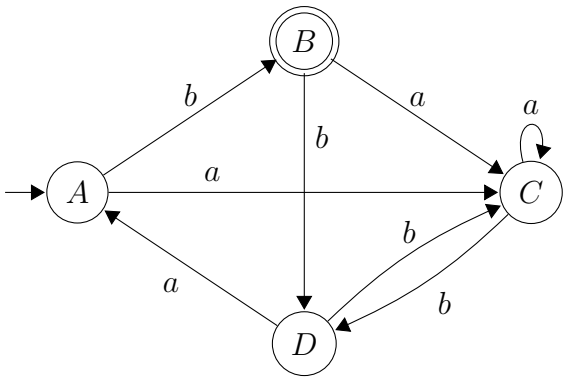
Note that the alphabet is the same for M and M'. You can also talk about relabeling (or renaming) of alphabet too.

The general idea of isomorphism is extremely important in computer science and math (and therefore is also use heavily in physics, chemistry, engineering, etc.)

Note that the above talks about "sameness" in two different ways: equivalence of DFAs and isomorphism of DFAs. Isomorphism of DFAs is stronger.

Exercise 12.3.2. Consider the following two DFAs: debug: exercises/minimization1/question.tex

and



Are they isomorphic? (Go to solution, page 11062) □

Exercise 12.3.3.

debug: exercises/minimization2/question.tex

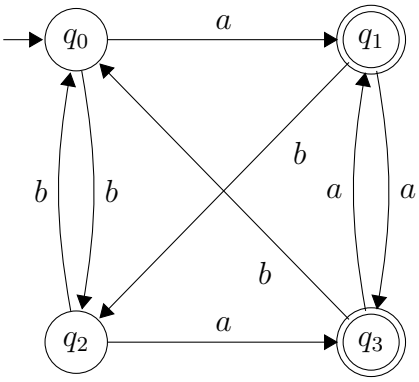
1. True or false:
   a) If M, M' are equivalent DFAs, then they are isomorphic.
   b) If M, M' are isomorphic DFAs, then they are equivalent.
2. Is equivalence of DFAs an equivalent relation?
3. Is isomorphism of DFAs an equivalent relation?

(Go to solution, page 11063) □

Exercise 12.3.4. What is the right definition of isomorphism if you want to rename the symbols in alphabets as well? (There's only one reasonable definition.) The definition should look like this: "An isomorphism between two DFAs $M = (\Sigma, Q, q_0, F, \delta)$ and $M' = (\Sigma', Q', q_0', F', \delta')$ is a pair of bijections $f : Q \to Q'$ and $g : \Sigma \to \Sigma'$ such that ..." Complete the definition. (Go to solution, page 11064) □

debug: exercises/minimization3/question.tex

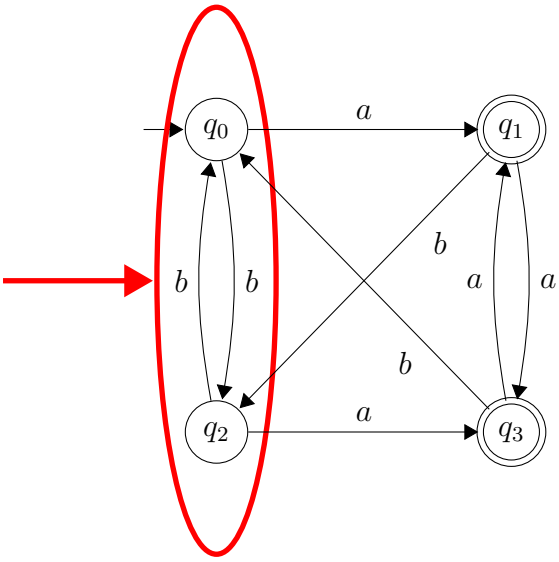
Let's look at this DFA:

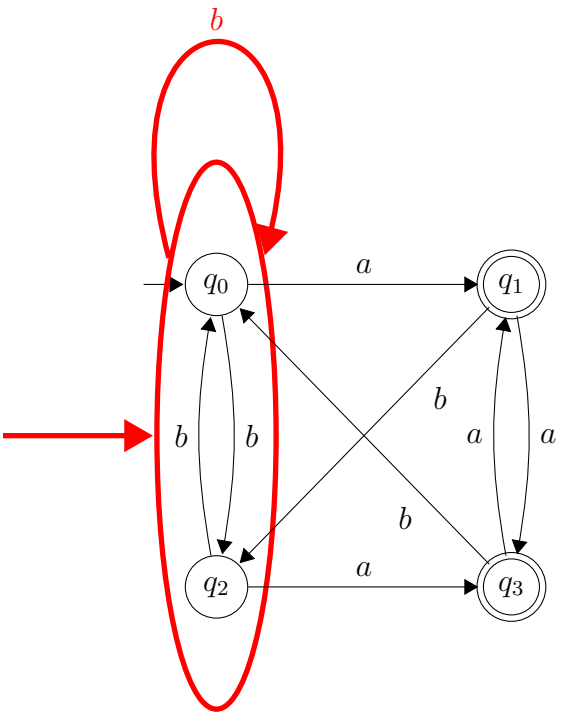


How would you simplify it? (Simplify = remove some state(s).)

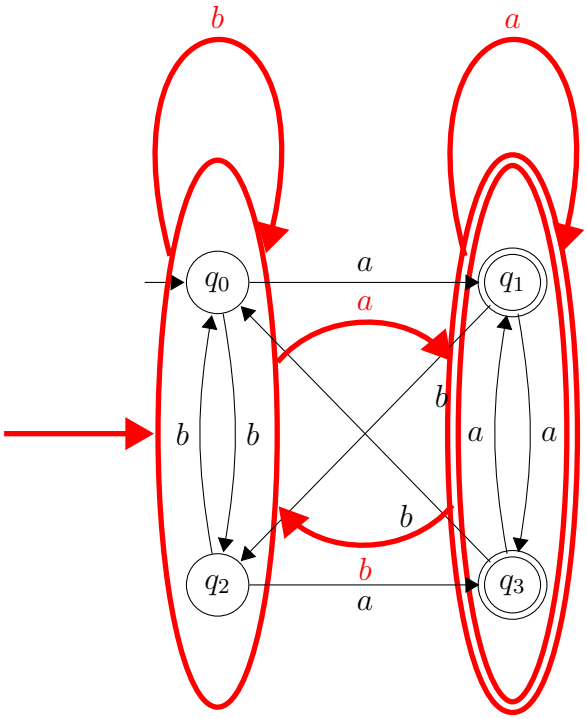
Pause here for 2 minutes and try to do it yourself ...

Now ... you can think of states are dumping ground for strings. For instance q_0 is the dumping ground (or final resting place) of the string ϵ, abb, aab, etc. If so, you can ask if you can "combine" q_0 and q_2 so that strings collected there can be combined into one dumping ground. (Of course it doesn't make sense to combine q_0 and q_1 – you can't put accepted and rejected strings in the same state!)

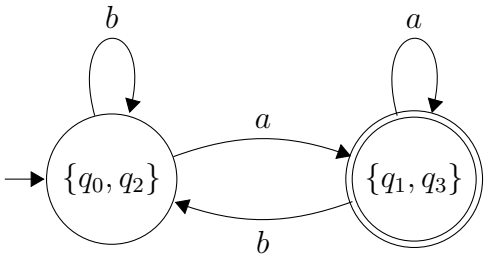




Notice that I have to draw a b–transition from $\{q_0, q_1\}$ to itself because there are b–transitions going between q_0 and q_1 in the original DFA. But notice this: In the original DFA, q_0 goes to q_1 by a and q_1 does to q_3 by a as well. But ... in the new DFA, q_0, q_1 can collapsed into one and are indistinguishable! In the new DFA, I would need to go to $\{q_0, q_1\}$ to q_1 and q_3 through a!!! That's not possible ... unless if q_1 and q_3 becomes one state. See that? At this point, that's possible since q_1 and q_3 are accept states. And I get this:



which gives me this:



As an example execution, look at the diagram with both DFAs and trace the execution of the string *abaa*. In the original DFA, the states visited are

$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \xrightarrow{a} q_1$$

In the minimized DFA, the states are

$$\{q_0, q_2\} \xrightarrow{a} \{q_1, q_3\} \xrightarrow{b} \{q_0, q_2\} \xrightarrow{a} \{q_1, q_3\} \xrightarrow{a} \{q_1, q_3\}$$

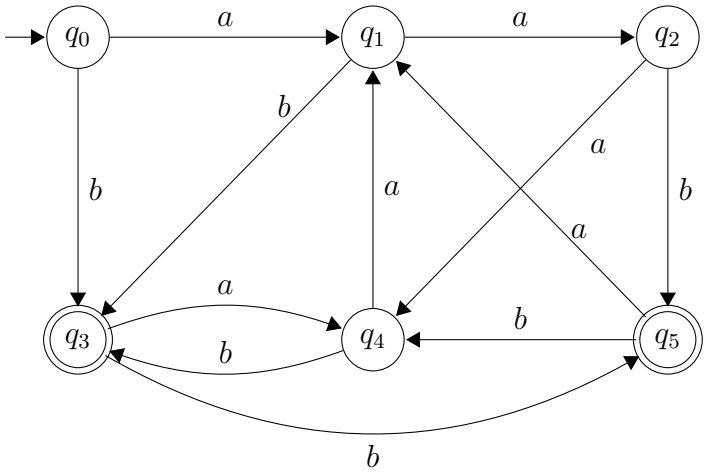
What will prevent us from grouping q_0, q_1 as a single state? Suppose x and y are two string such that x lands in q_0 and y lands in q_1 in the original DFA

M. Suppose I collapse q_0 and q_1 into one state, called it $\{q_0, q_1\}$ and call the new DFA M' which is supposed to minimize M. In this new DFA M', since x, y land in the same state $\{q_0, q_1\}$, the longest strings xz, yz must also land in the same since they are in the same state *before* z. If in the original DFA M, if $xz \in L$ and $yz \notin L$ (i.e., xz lands in an accept state and yz lands in a non-accept state), then the new DFA M' does not work correctly.

The above example of minimizing a 4–state DFA to a 2–state DFA is ad hoc (of course). We now need to create a robust algorithm to minimize *any* DFA. Before that, try to minimize the following DFA without looking at the theory below it.

Exercise 12.3.5. Try to minimize the following DFA (or is it already minimal?)

debug: exercises/minimization4/question.tex



(Go to solution, page 11065) □

Let's fix some terminology and notation. Let me fix a language L (over Σ). Let $x, y \in \Sigma^*$. I will say that L **separates** x and y if

separates

$$x \in L, y \notin L \quad \text{or} \quad x \notin L, y \in L$$

I'll also say in this case that x, y are L–**separable**.

separable

I will say that x, y are L–**distinguishable** if there is *some* $z \in \Sigma^*$ such that xz and yz are L–separable, i.e.,

distinguishable

$$xz \in L, yz \notin L \quad \text{or} \quad xz \notin L, yz \in L$$

(Note the emphasize "there is *some*".) Of course x, y are not L–distinguishable if for all z, xz, yz are not L–separable. If x, y are L–distinguishable, I'll write

$$x \not\equiv_L y$$

$\not\equiv_L$

otherwise I'll write

$$x \equiv_L y$$

$\equiv_L$

(just to save on ink.)

Here's how to think of this concept: All strings in Σ^* will give you a path in a DFA (if there's one) for L. x, y are L–distinguishable, if after you walk along x and your friend walk along y, and then both of you manage to find some extra common steps z and one of you end up being *in* L and the other is *not in* L.

You can also obviously talk about a set of strings (more than two) being **pairwise** L–**distinguishable**, meaninng to say any pair of string in this set is L–distinguishable.

pairwise L–distinguishable

Example 12.3.1. Consider the following example:

$$L_1 = \{a^n b^n \mid n \geq 0\}$$

Let

$$x = a, \;\; y = b$$

Then x, y are L_1–distinguishable. Why? Because if I pick $z = b$, then

$$xz = ab \in L, \;\; yz = bb \notin L$$

However if

$$x = ba, \;\; y = b$$

then x, y are not L_1–distinguishable: Given *any* $z \in \Sigma^*$,

$$xz = baz, \;\; yz = bz$$

are both not not in L_1. □

Exercise 12.3.6. Again let

debug: exercises/minimization5/question.tex

$$L_1 = \{a^n b^n \mid n \geq 0\}$$

1. Are ϵ, a L_1–separable? L_1–distinguishable?
2. Are aa, bb L_1–separable? L_1–distinguishable?
3. Are $ab, aabb$ L_1–distinguishable? L_1–distinguishable?
4. Find an example of x, y such that xz, yz are L_1–separated for z and xz', yz' are not L_1–separated for z'. (So you need to find four strings x, y, z, z'.)
5. Find three strings which are pairwise L_1–distinguishable. How about four? Five?
6. Is this true? Given any positive $n \geq 2$, it's possible to find n strings which are pairwise L_1–distinguishable.

(Go to solution, page 11066) □

Example 12.3.2. Here's another example. Let

$$L_2 = \{w \in \Sigma^* \mid |w|_a, |w|_b \text{ both even}\}$$

where $|w|_a$ is the number of a's in w and $|w|_b$ is the number of b's in w. For example $|abaab|_a = 3$ and $|abaab|_b = 2$. Therefore $abaab \notin L_2$. However $abbaaa \in L_2$.

(a) Are ϵ, a L_2–separable? L_2–distinguishable?
(b) Are ϵ, b L_2–separable? L_2–distinguishable?
(c) Are ϵ, aa L_2–separable? L_2–distinguishable?
(d) Are ϵ, ab L_2–separable? L_2–distinguishable?
(e) Are ϵ, bb L_2–separable? L_2–distinguishable?
(f) Are $abaa, abababa$ L_2–distinguishable?

SOLUTION. (a) $\epsilon \in L_2$ and $a \notin L_2$. Hence ϵ, a are L_2–separable. Choose $z = \epsilon$, $\epsilon z, az$ are L_2–separable. Hence ϵ, a are L_2–distinguishable.

(b) For you.

(c) $\epsilon, aa \in L_2$. Hence ϵ, aa are not L_2–separable. They are also not L_2-distinguishable.

(d) $\epsilon \in L_2$ and $ab \notin L_2$. Hence ϵ, ab are L_2–separable. They are also L_2-distinguishable.

(e), (f) For you. □

Let

$$x = abb, \quad y = ababa$$

Then x, y are not L_2–separable because the number of a's in both are odd and the number of b's in both are odd:

$$|x|_a \equiv |y|_a \equiv 1 \pmod 2$$
$$|x|_b \equiv |y|_b \equiv 1 \pmod 2$$

(Any one of the above condition is actually enough.) If z has an even number of a's and an even number of b's:

$$|z|_a \equiv 0 \pmod 2$$
$$|z|_b \equiv 0 \pmod 2$$

Then

$$|xz|_a = |x|_a + |z|_a \equiv 1 + 0 \equiv |y|_a + |z|_a = |yz|_a \pmod 2$$

and so xz, yz will both have an odd number of a's. Yes? And so again xz, yz are not L_2–separable.

Exercise 12.3.7.

debug: exercises/minimization6/question.tex

(a) Suppose x, y are L_2–separable and $z \in \Sigma^*$. What can you tell me about xz, yz?
(b) Suppose x, y are not L_2–separable and $z \in \Sigma^*$. What can you tell me about xz, yz?
(c) Suppose x, y are not L_2–separable and $z \in \Sigma^*$. What can you tell me about L_s–distinguishability of xz, yz?
(d) Suppose x, y are not L_2–separable and $z \in \Sigma^*$. What can you tell me about L_s–distinguishability of xz, yz?

(Go to solution, page 11067) □

Exercise 12.3.8. Again let

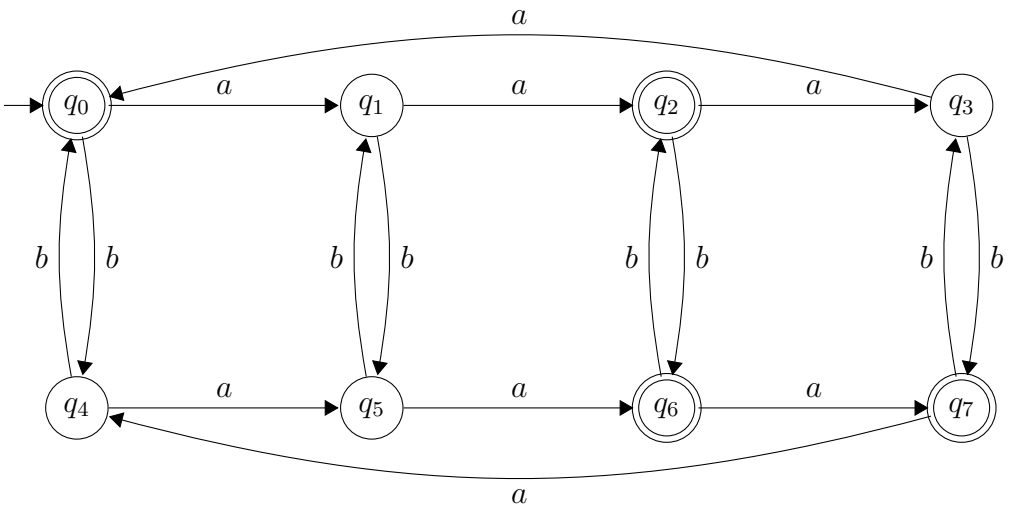
debug: exercises/minimization7/question.tex

$$L_2 = \{w \in \Sigma^* \mid |w|_a, |w|_b \text{ both even}\}$$

(a) Describe x, y if they are not L_2–separable.
(b) If x, y are not L_2–separable, what is the simplest z to make x, y L_2–separable.
(c) If x, y are L_2–separable, what is the simplest z to make not x, y L_2–separable.
(d) Show that ϵ, a are L_2–distinguishable.
(e) If ϵ, a, b pairwise L_2–distinguishable?
(f) Find a set of 4 words which are L_2–distinguishable.
(g) Try (if possible) to find a set of 5 words which are L_2–distinguishable.

(Go to solution, page 11068) □

Example 12.3.3. Let M be the following DFA:



Let $L = L(M)$.

(a) Are ϵ, a^4 L-distinguishable?
(b) Are ϵ, a L-distinguishable?
(c) Are ϵ, a^2 L-distinguishable?
(d) Are ϵ, a^3 L-distinguishable?
(e) Are ϵ, b L-distinguishable?
(f) Are ϵ, ba L-distinguishable?
(g) Are ϵ, ba^2 L-distinguishable?
(h) Are ϵ, ba^3 L-distinguishable?

Note that L–distinguishability does not depend on the concept of DFA. The following is a pretty obvious statement on the relationship between L–distinguishability and DFAs. Let $\text{state}_M(x)$ denotes the state that x lands in if M computes with x, i.e.,

$$\text{state}_M(x) = \delta^*(q_0, x)$$

where q_0 is the start state and δ is the transition function of M.

Proposition 12.3.1. *Let M be a DFA and $L = L(M)$. Let $x, y \in \Sigma^*$ and p, q be the respective states reached when x, y are computed with M. If x, y are L–distinguishable, then $\text{state}_M(x) \neq \text{state}_M(y)$.*

Proof. The proof is obvious. Suppose x and y lands in the same state. Then for any $z \in \Sigma^*$, xz and yz will both also land in a common state. Of course the state is either an accept state of non–accept state, which means that xz and yz cannot be separated by L. This contradicts the L–distinguishability of x, y. □

Proposition 12.3.2. *Let L be a language. If $L = L(M)$ where M is a DFA and L has k pairwise L–distinguishable strings, then M has at least k states.*

Proof. Let $x_0, \ldots, x_{k-1}$ be a set of k pairwise L–distinguishable strings. Suppose M has less than k states. Running M on the strings $x_0, \ldots, x_{k-1}$, we will obtain k states. By the pigeonhole principle, since M has less than k states, two of the strings in $x_0, \ldots, x_{k-1}$ must land in the same state. But this is a contradiction (by the previous statement) since L–distinguishable strings cannot land in the same state. □

Exercise 12.3.9.

debug: exercises/minimization8/question.tex

1. Let $L_1 = \{a^n b^n \mid n \geq 0\}$. Prove that L_1 is not a regular language.
2. Let $L_2 = \{w \in \Sigma^* \mid |w|_a, |w|_b \text{ both even}\}$. What is the size of the smallest DFA? Construct a minimal DFA for L_2. Note that the results above only tells you a lower bound on the number of states of a DFA for L_2. It does not say that the number stated is attained. The results above also does not tell you how to construct a DFA.

(Go to solution, page 11069) □

Note that the previous statements tell you when you have the smallest DFA for a language. But suppose a language has 5 distinguishable strings. At this point, you only know that a DFA has at least 5 states. Does the minimal DFA have *exactly* 5? The above statement does not address this question.

Another thing to note is that we already have many techniques for designing DFAs. What we need is to use those techniques to build a DFA, and *then* minimize *that* DFA just built.

So here we go ...

First, recall that I have a relation $\equiv_L$ on strings in Σ^*. Recall that $\not\equiv_L$ means L–distinguishability. In other words

$$x \not\equiv_L y$$

if there is some $z \in \Sigma^*$ such that L separates xz, yz. In other words

$$x \equiv_L y$$

if for all $z \in \Sigma^*$, L does not separate xz, yz.

Proposition 12.3.3. *$\equiv_L$ is an equivalence relation on Σ^*. In other words:*

(a) *Reflexivity: For all $x \in \Sigma^*$, $x \equiv_L x$.*
(b) *Symmetry: For all $x, y \in \Sigma^*$, if $x \equiv_L y$, then $y \equiv_L x$.*
(c) *Transitivity: For all $x, y, w \in \Sigma^*$, if $x \equiv_L y, y \equiv_L w$, then $x \equiv_L w$.*

Proof. Easy exercise. □

This means that Σ^* is partitioned into a collection of disjoint sets of strings. The fancy term for such a set is **equivalence class**. In other words the collection of equivalence classes look like $[x_0], [x_1], [x_2], ..., [x_{k-1}]$ where

equivalence class

$$[x_i] = \{x \in \Sigma^* \mid x_i \equiv_L x\}$$

Here's I'm assuming the number of equivalence classes is finite. It can be infinite – see exercise on showing $L_1 = \{a^n b^n \mid n \geq 0\}$ is not regular.

Remember what I said: If there are k equivalence classes, it means that a DFA for L (if there's one at all) has at least k states. It turns out the k *is* actually the correct number of states for the smallest DFA:

Theorem 12.3.2. Myhill–Nerode (1958) *Let L be a language (over Σ^*).* Myhill–Nerode (1958)

(a) *If $\equiv_L$ (over Σ^*) has infinitely many equivalence classes then L is not regular.*

(b) *If $\equiv_L$ (over Σ^*) has finitely many equivalence classes, say k, then L is regular and there is a DFA M with exactly k states, where each state correspond to an equivalence class of $\equiv_L$. Specifically if $[x_0], [x_1], [x_2], ..., [x_{k-1}]$ where $x_0 = \epsilon$, are the equivalence classes of $\equiv_L$, then we define $M = (\Sigma, Q, q_0, F, \delta)$ to be*

a) The set of states Q is $\{[x_0], [x_1], [x_2], ..., [x_{k-1}]\}$.
b) The start state is $[x_0]$.
c) The set of final states F consists of those $[x_i]$ where $x_i \in L$.
d) The transition function $\delta : Q \times \Sigma \to Q$ is defined as follows. If $c \in \Sigma$,

$$\delta([x_i], c) = [x_i c]$$

then $L(M) = L$.

Before proving Myhill–Nerode's theorem, let me use it ...

Example 12.3.4. Let's try to compute the minimal DFA for

$$L_2 = \{w \in \Sigma^* \mid |w|_a, |w|_b \text{ both even}\}$$

SOLUTION. Recall from the earlier exercises, there are exactly four L_2–distinguishable strings:

$$\epsilon, a, b, ab$$

Hence $\equiv_{L_2}$ partitions Σ^* into four equivalence classses:

$$[\epsilon], [a], [b], [ab]$$

where

$$\begin{aligned}
[\epsilon] &= \{x \in \Sigma^* \mid \epsilon \equiv_L x\} = \{x \in \Sigma^* \mid |x|_a, |x|_b \equiv 0, 0 \pmod 2\} \\
[a] &= \{x \in \Sigma^* \mid a \equiv_L x\} = \{x \in \Sigma^* \mid |x|_a, |x|_b \equiv 1, 0 \pmod 2\} \\
[b] &= \{x \in \Sigma^* \mid b \equiv_L x\} = \{x \in \Sigma^* \mid |x|_a, |x|_b \equiv 0, 1 \pmod 2\} \\
[ab] &= \{x \in \Sigma^* \mid ab \equiv_L x\} = \{x \in \Sigma^* \mid |x|_a, |x|_b \equiv 1, 1 \pmod 2\}
\end{aligned}$$

The start state is $[\epsilon]$. The set of accept states is

$$F = \{[\epsilon]\}$$

since among the string ϵ, a, b, ab, ϵ is the only string in L. The transition function δ is given by the following:

- $\delta([\epsilon], a) = [\epsilon a] = [a]$.
- $\delta([\epsilon], b) = [\epsilon b] = [b]$.
- $\delta([a], a) = [aa] = [\epsilon]$. This is because $aa \equiv_{L_2} \epsilon$.
- $\delta([a], b) = [ab]$.
- $\delta([b], a) = [ba] = [ab]$. This is because $ba \equiv_{L_2} ab$.
- $\delta([b], b) = [bb] = [\epsilon]$. This is because $bb \equiv_{L_2} \epsilon$.
- $\delta([ab], a) = [aba] = [b]$. This is because $aba \equiv_{L_2} b$.
- $\delta([ab], b) = [abb] = [a]$. This is because $abb \equiv_{L_2} a$.

The above describes the DFA completely. Here's the diagram:

Now let's prove Myhill–Nerode's theorem.

Proof. (a) If $\equiv_L$ has infinitely many equivalence classes, that means that there are infinitely many L–distinguishable strings and we have already mentioned that this means that for every $k > 0$, a DFA for L (if it exists) must have at least k states. But this means that M has arbitrarily many states. That's impossible since M must have a finite number of states by definition. (I actually already talked about this earlier.)

(b) First let me show that δ is well–defined, i.e., I need to show that if $x_i \equiv_L x$, then

$$\delta([x_i], c) = [x_i c] = [x'c] = \delta([x], c)$$

In other words, I need to show

$$x_i c \equiv_L x' c$$

Again, I need to show

$$x_i \equiv_L x \implies x_i c \equiv_L x' c$$

$x_i \equiv_L x$ means that for all $z \in \Sigma^*$, L does not separate $x_i z, xz$. Let $z' \in \Sigma^*$. Then $x_i(cz'), x(cz)$ are not separated by L. Hence $(x_i c)z', (xc)z$ are not separated by L, which implies that $x_i c, xc$ are not L–distinguishable, i.e., $x_i c \equiv_L xc$.

Clearly M is a DFA. Now I need to show $L(M) = L$. Let $x \in \Sigma^*$.

$$x \in L(M) \iff \delta^*([\epsilon], x) \in F$$

Note that

$$\delta^*([\epsilon], x) = [x]$$

for any $x \in \Sigma^*$. (This can be proven by induction.) Hence

$$\begin{aligned} x \in L(M) &\iff [x] \in F \\ &\iff \exists x_i \in L \text{ such that } [x] = [x_i] \\ &\iff \exists x_i \in L \text{ such that } x \equiv_L x_i \\ &\iff \exists x_i \in L \text{ such that } \forall z, L \text{ does not separate } xz, x_i z \end{aligned}$$

In particular, if $x \in L(M)$, then there is some $x_i \in L$ such that for $z = \epsilon$, $xz = x$ and $x_i z = x_i$ are not L separable. Since $x_i \in L$ and L does not separate x, x_i, $x \in L$. I have just shown

$$x \in L(M) \implies x \in L$$

Hence $L(M) \subseteq L$. Now let $x \in L$, say $[x] = [x_i]$.

$$\delta^*([\epsilon], x) = [x] = [x_i]$$

This implies that for all z, L does not separate $xz, x_i z$. In particular when $z = \epsilon$, L does not separate x, x_i. Since $x \in L$, then $x_i \in L$. Hence

$$\delta^*([\epsilon], x) = [x] = [x_i] \in F$$

Therefore $x \in L$. Hence $L \subseteq L(M)$. Altogether I have shown $L = L(M)$. $\square$

The easiest thing to do is to remove states which are not reachable from the start state. Simply perform a breadth-first traversal. For instance (this is not optimized):

```
Let S be the set of reachable states. Initialize is with the start state.

Repeatedly, put all states reachable from S into S.
Repeat the above until S is not changed.
```

After the above is done, only the states in S need to be considered. The states in $Q - S$ are thrown away. Of course F is replaced by $F - S$ and δ is updated so that only states in $Q - S$ are considered part of δ.

First we'll find a smaller DFA. Later we'll prove that it's the smallest using Myhill-Nerode's Theorem.

Throughout this section, $M = (\Sigma, Q, q_0, F, \delta)$ will be a DFA where $Q = \{q_0, \ldots, q_{n-1}\}$

The idea is to collapse states ... equivalence relations to the rescue!

Let L be a language (over Σ). Given two strings x, y in Σ^*, I will say that L separates x, y if there is some $z \in \Sigma^*$ such that either $xz \in L, yz \notin L$ or $xz \notin L, yz \in L$. I will further write

$$x \equiv_L y$$

if there is no z such that L separates xz, yz. In other words

$$x \equiv_L y$$

iff

$$\text{for all } z \in \Sigma^*, xz \in L \iff yz \in L$$

Proposition 12.3.4. *$\equiv_L$ is an equivalence relation on Σ^**

Proof. Exercise. □

What is the intuition here?

Consider the example of

$$L = \{x \in \{a, b\}^* \mid |x| \equiv 1 \bmod 8\}$$

Let M be the DFA with 8 states that accepts $w \in \Sigma^* = \{a, b\}^*$ where the number of a's in w is $\equiv 1$ or 5 (mod 8). We do know that L is regular with a DFA with 8 states. Notice that the string Σ^* will land in the states. So we can think of classes of strings as being same as states, i.e., each state can be thought of as a collection of strings from Σ^*. For instance if $|x| = 6$ and $|y| = 8k + 6$, then

$$x \equiv_L$$

Hence Σ^* is partitioned into 8 sets of strings in the obvious way. The partitions are

- $[\epsilon]$
- $[a] = [b]$
- $[a^2] = [b^2] = [ab] = [ba]$
- $[a^3] = [b^3]$
- $[a^4]$
- $[a^5]$
- $[a^6]$
- $[a^7]$

Exercise 12.3.10. (Go to solution, page 11070) □ debug: exercises/minimization9/question.tex

$\equiv_M$ is more than just an equivalence relation. Note that if $x \equiv_M y$, then they remain equivalence (wrt $\equiv$) even when you extend them "on their right":

Exercise 12.3.11. Let M be a DFA. Prove that $\equiv_M$ is right invariant. (Go to solution, page 11071) □ debug: exercises/minimization10/question.tex

The main idea is that the equivalence classes of L under $\equiv_M$ will be the states. [?]

Now we define an equivalence relation on *any* language L. Note that L need not be regular:

Definition 12.3.1. Let L be any language. Define the following relation on Σ^*. For $x, y \in \Sigma^*$,

$$x \equiv_L y \quad \text{if} \quad \text{for all } z,\ xz \in L \iff yz \in L$$

This is the same as saying that $x \equiv_L y$ if for all $z \in \Sigma^*$, either

- xz, yz both in L, or
- xz, yz both not in L.

We will write $[x]_L$ for the equivalence class of x under $\equiv_L$.

Example 12.3.5. Let $\Sigma = \{a, b\}$. For simplicity we will write $|w|_a$ for the number of a's in w and $|w|_b$ for the number of b's in w. Let $L = \{w \in \Sigma^* \mid |w|_a, |w|_b \text{ are even}\}$.

Look at the conditions: $|w|_a$ is even and $|w|_b$ is even. Notice that if you define the following sets

1. $L_{ee} = \{w \in \Sigma^* \mid |w|_a \text{ even}, |w|_b \text{ even}\}$
2. $L_{eo} = \{w \in \Sigma^* \mid |w|_a \text{ even}, |w|_b \text{ odd}\}$
3. $L_{oe} = \{w \in \Sigma^* \mid |w|_a \text{ odd}, |w|_b \text{ even}\}$
4. $L_{oo} = \{w \in \Sigma^* \mid |w|_a \text{ even}, |w|_b \text{ even}\}$

Obviously $L_{ee}, L_{eo}, L_{oe}, L_{oo}$ forms a partition of L (Recall: this means they are pairwise disjoint and their union is L.)

When we look the equivalence relation on L, we see that

1. $L_{ee} = [\epsilon]$
2. $L_{eo} = [b]$
3. $L_{oe} = [aaabb]$
4. $L_{oo} = [baaabb]$

Not only that, we can actually use these equivalence classes of strings to create a DFA in the following way. If $x \in \Sigma^*$ and $a \in \Sigma$, we just define

$$\delta([x]_L, a) = [xa]_L$$

This is the smallest (in terms of number of states) DFA that accepts L.

Example 12.3.6. Now consider $L = \{w \in \Sigma \,|\, w \text{ does not contain aab }\}$ where $\Sigma = \{a, b\}$.

- Is $ba \equiv_L baa$?
- Suppose w, w' both contain aab. Is it true that $w \equiv_L w'$?

OK. Here's the big theorem of this section:

Theorem 12.3.3. *(Myhill-Nerode) Let L be a language over Σ. The following are equivalent:*

(a) L is regular
(b) There is a right invariant equivalence relation $\equiv$ on L such that $\equiv$ has finite index and L is the union of some equivalence classes of $\equiv$.
(c) $\equiv_L$ has finite index.

I will not give a proof. However the important thing is that the proof of (c) $\implies$ (a) is constructive. In particular, the DFA constructed uses equivalence classes of $\equiv_L$ as states.

The important thing you should realize by reading this theorem is that this is another way of characterizing regular languages. This characterization does not rely on the definition of DFA. In other words you can now re-define what is meant by regular in terms of $\equiv_L$ and the finiteness of the equivalence classes of $\equiv_L$. Theorems like these are extremely important because they reveal connections between different concepts and reveal different points of view. Note also

that if you can write down infinitely many strings which are not equivalent to each other, then L is not regular.

Theorem 12.3.4. *(DFA Minimization) If L is regular and M is a DFA with the minimum number of states, then M is the "same" as the DFA constructed in the Myhill-Nerode theorem.*

Here "same" can be formalize in terms of isomorphisms of DFA. It means "the same up to renaming the states". For instance if you take a DFA with states labeled $q_0, q_1, \ldots, q_{n-1}$ and relabel them as $s_0, s_1, \ldots, s_{n-1}$, then obviously the way they both operate is exactly the same except for the names of the states.

The important thing for us is how do we simplify a DFA? First of all the simplest thing to do first is to remove states that cannot be reached from the initial state. For a human being, if the DFA is simple, then we can do it visually. In terms of an algorithm, you need to process the states in the following manner. First you need a list of states called B. This will be the list of states you will visit. You also need a list of visited states; let's call this A. You initialize B with the initial state and initialize A to be empty. Now in a while loop, take a state q out of B, visit all the states you can go to from q and put them into B, and put q into A. Repeat this loop as long as B is not empty. When you're done, A contains all the states that can be reached from the initial state.

The other question is how do we merge the visited states?

We will define another relation on states $\equiv$ so that states that should merge are equivalence to each other. Let p, q be states. Then we say that p and q are indistinguishable, and we write

$$p \equiv q$$

if there is some z such that either

- $\delta(p, z) \in F$ and $\delta(q, z) \notin F$, or
- $\delta(p, z) \notin F$ and $\delta(q, z) \in F$

All you need to do is to merge indistinguishable states into a new state.

Here's a systematic way of telling if pairs of states are distinguishable or not:

[insert]

There is a more efficient algorithm (I won't do it here) that will build the minimum DFA with time complexity $O(n \lg n)$ where $n = |Q|$.

Solutions

Solution to Exercise 12.1.1.

debug: exercises/nfa-to-regex0/answer.tex

Solution not provided.

Solution to Exercise 12.1.2.

debug: exercises/nfa-to-regex1/answer.tex

Solution not provided.

Solution to Exercise 12.1.3.

debug: exercises/nfa-to-regex2/answer.tex

Solution not provided.

Solution to Exercise 12.2.1.

debug:
exercises/regex-to-nfa0/answer.tex

Solution not provided.

Solution to Exercise 12.3.1.

debug: exercises/minimization0/answer.tex

Solution not provided.

Solution to Exercise 12.3.2.

debug: exercises/minimization1/answer.tex

Solution not provided.

Solution to Exercise 12.3.3.

debug: exercises/minimization2/answer.tex

Solution not provided.

Solution to Exercise 12.3.4.

debug: exercises/minimization3/answer.tex

Solution not provided.

Solution to Exercise 12.3.5.

debug: exercises/minimization4/answer.tex

Solution not provided.

Solution to Exercise 12.3.6.

debug: exercises/minimization5/answer.tex

Solution not provided.

Solution to Exercise 12.3.7.

debug: exercises/minimization6/answer.tex

Solution not provided.

Solution to Exercise 12.3.8.

debug: exercises/minimization7/answer.tex

Solution not provided.

Solution to Exercise 12.3.9.

debug: exercises/minimization8/answer.tex

Solution not provided.

Solution to Exercise 12.3.10.

debug: exercises/minimization9/answer.tex

Solution not provided.

Solution to Exercise 12.3.11.

debug: exercises/minimization10/answer.tex

Solution not provided.

Index