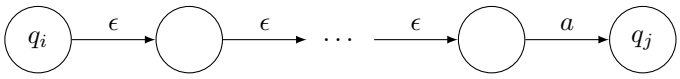
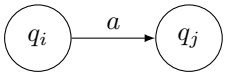
3. Add new transitions to the transition function δ' as follows:

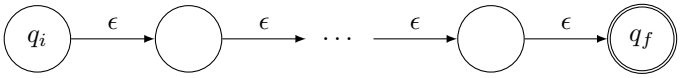
   - If there exists a "chain" of transitions in $\mathcal{M}$ beginning at a state q_i and ending at a state q_j, where all but the last transition is an epsilon transition and the last transition is on some symbol $a \in \Sigma$,

   then replace this "chain" in $\mathcal{M}'$ with a single transition on a between q_i and q_j.

   - If there exists a "chain" of epsilon transitions in $\mathcal{M}$ beginning at a state q_i and ending at a final state $q_f \in F$,

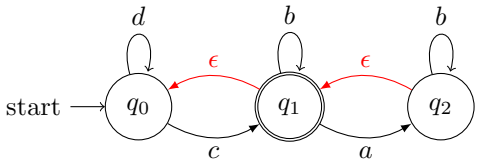
   

   then remove this "chain" from $\mathcal{M}'$ and make q_i a final state.

In this way, we have constructed a nondeterministic finite automaton without epsilon transitions recognizing the same language as the original finite automaton. □

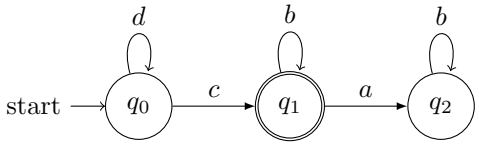
Example 25. Consider the following nondeterministic finite automaton with epsilon transitions (highlighted in red):



We will use our construction to convert this to a nondeterministic finite automaton without epsilon transitions.

1. First, we take our state set Q' and our initial state q_0'. Since there are no states in this finite automaton having *only* incoming epsilon transitions, we don't need to remove any states.

2. Next, we take our transition function δ' with all epsilon transitions removed. We don't need to remove any other transitions from removed states, since we had no such states in the previous step.



3. Now, we add new transitions to δ' by considering any "chains" in the original finite automaton:

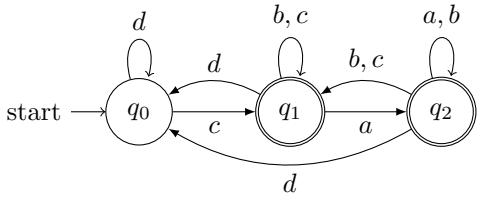
   - For epsilon transition chains ending in a transition on a symbol, we have the following:

     – $q_1 \xrightarrow{\epsilon} q_0 \xrightarrow{c} q_1$ is replaced by $q_1 \xrightarrow{c} q_1$;

- $q_1 \xrightarrow{\epsilon} q_0 \xrightarrow{d} q_0$ is replaced by $q_1 \xrightarrow{d} q_0$;
- $q_2 \xrightarrow{\epsilon} q_1 \xrightarrow{a} q_2$ is replaced by $q_2 \xrightarrow{a} q_2$;
- $q_2 \xrightarrow{\epsilon} q_1 \xrightarrow{b} q_1$ is replaced by $q_2 \xrightarrow{b} q_1$;
- $q_2 \xrightarrow{\epsilon} q_1 \xrightarrow{\epsilon} q_0 \xrightarrow{d} q_0$ is replaced by $q_2 \xrightarrow{d} q_0$; and
- $q_2 \xrightarrow{\epsilon} q_1 \xrightarrow{\epsilon} q_0 \xrightarrow{c} q_1$ is replaced by $q_2 \xrightarrow{c} q_1$.

- For epsilon transition chains ending at a final state, we have the following:
  - $q_2 \xrightarrow{\epsilon} q_1$, so state q_2 becomes a final state.

Adding these transitions and final states produces our nondeterministic finite automaton without epsilon transitions:



4.2 NFA = DFA

In our next procedure, we will learn how to "simulate" nondeterminism in a deterministic finite automaton. Recall that, in a nondeterministic finite automaton, the transition function maps state/symbol pairs to an element of $\mathcal{P}(Q)$. We can get around the issue of having multiple transitions from one state on the same symbol not by changing our transitions, but by changing our set of states: we simply need to create one state corresponding to each element of $\mathcal{P}(Q)$!

Theorem 26. *Given a nondeterministic finite automaton $\mathcal{N}$, we can convert it to a deterministic finite automaton $\mathcal{N}'$.*

Proof. Let $\mathcal{N} = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic finite automaton. We assume that $\mathcal{N}$ contains no epsilon transitions; if it does, then use the construction of Theorem 24 to remove the epsilon transitions.

We will construct a deterministic finite automaton $\mathcal{N}' = (Q', \Sigma, \delta', q_0', F')$ in the following way:

1. Take $Q' = \mathcal{P}(Q)$; that is, each state of $\mathcal{N}'$ corresponds to a subset of states of $\mathcal{N}$. Note that our deterministic finite automaton may not need to use all of these states; usually, we omit any inaccessible states to make our diagram easier to follow.
2. For each $q' \in Q'$ and $a \in \Sigma$, take $\delta'(q', a) = \{q \in Q \mid q \in \delta(s, a) \text{ for some } s \in q'\}$.

   (This is perhaps the most difficult step of the construction. Remember that each state q' of $\mathcal{N}'$ corresponds to a subset of states of $\mathcal{N}$. Thus, when $\mathcal{N}'$ reads a symbol a in state q', the transition function δ' takes us to the state corresponding to the subset of states q of $\mathcal{N}$ that we would have transitioned to upon reading a in some state s of $\mathcal{N}$, where s is in the subset corresponding to q'.)
3. Take $q_0' = \{q_0\}$; that is, the initial state of $\mathcal{N}'$ corresponds to the subset containing only the initial state of $\mathcal{N}$.
4. Take $F' = \{q' \in Q' \mid q' \text{ corresponds to a subset containing at least one final state of } \mathcal{N}\}$. In this way, $\mathcal{N}'$ accepts only if $\mathcal{N}$ would be in a final state at the same point in its computation.

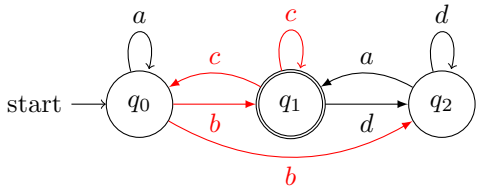
In this way, we have constructed a deterministic finite automaton recognizing the same language as the original finite automaton. □

The procedure allowing us to convert from nondeterministic to deterministic finite automata is known as the *subset construction*, because each state of our deterministic finite automaton corresponds to a subset of states from the original nondeterministic finite automaton.

Step 2 of the subset construction procedure is the most involved step. Fortunately, we can obtain the transition function of our deterministic finite automaton $\mathcal{N}'$ using a tabular method via the following steps:

1. Construct a table where the rows are the states of $\mathcal{N}$ and the columns are the symbols of the alphabet Σ.
2. For each state q_i and symbol a, write the set of states mapped to by $\delta(q_i, a)$ in the corresponding row/column entry.
3. After all entries are filled, take all sets of states listed in the table that don't yet have their own row, and create a new row corresponding to that set of states.
4. Repeat steps 2 and 3 until no new rows can be added to the table.

Example 27. Consider the following nondeterministic finite automaton, with nondeterministic transitions from a state highlighted in red:



We will use our tabular construction method to obtain the transition function of our desired deterministic finite automaton. Our initial table looks like the following:

	a	b	c	d
q_0				
q_1				
q_2				

We fill in the initial table entries by consulting the transition function of $\mathcal{N}$, where — denotes no transition:

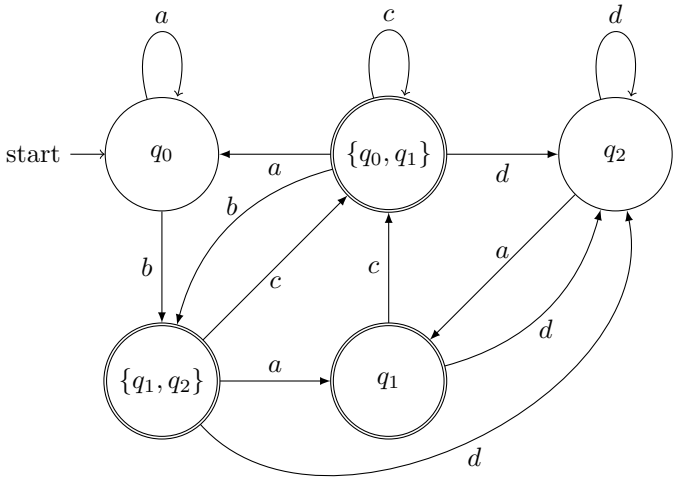
	a	b	c	d
q_0	q_0	$\{q_1, q_2\}$	—	—
q_1	—	—	$\{q_0, q_1\}$	q_2
q_2	q_1	—	—	q_2

Note that there are now two entries in our table without corresponding rows: $\{q_0, q_1\}$ and $\{q_1, q_2\}$. We proceed to add these entries as rows to our table and we fill in the entries for these new rows:

	a	b	c	d
q_0	q_0	$\{q_1, q_2\}$	—	—
q_1	—	—	$\{q_0, q_1\}$	q_2
q_2	q_1	—	—	q_2
$\{q_0, q_1\}$	q_0	$\{q_1, q_2\}$	$\{q_0, q_1\}$	q_2
$\{q_1, q_2\}$	q_1	—	$\{q_0, q_1\}$	q_2

After filling in these new entries, we find that all entries now have corresponding rows, so our table construction is complete. We can now use this table to construct our deterministic finite automaton! Each row of the table corresponds to an accessible state of our deterministic finite automaton, and the table itself specifies our transition function.

Our resultant deterministic finite automaton is the following:



Note that we don't need to come up with procedures for the other directions of conversion: a deterministic finite automaton is a "nondeterministic finite automaton" that doesn't use nondeterminism, and a nondeterministic finite automaton is a "nondeterministic finite automaton with epsilon transitions" that doesn't use any epsilon transitions. Therefore, we can convert in any direction between all three of our finite automaton models!

This, together with the knowledge that any language recognized by a finite automaton is regular, allows us to conclude that all of our finite automaton models are equivalent in terms of recognition power.

4.3 DFA = RE

Let's now turn back to regular expressions. Since regular expressions are entirely symbol-based, it might be easier for us in some cases to represent a regular language using a regular expression. In other cases, it might be easier for us to directly construct a finite automaton that recognizes the language. However, is it always the case that, if we can do one, we can also do the other?

We now know that all models of finite automata are equivalent in terms of their recognition power, so all that remains is for us to discover how we can bring regular expressions under this same umbrella. For this last step, we will devise a procedure—actually, two procedures—that allows us to convert a deterministic finite automaton into a regular expression and vice versa.

One direction of our procedure, taking us from finite automaton to regular expression, will systematically eliminate individual states until the automaton is in a simpler standard form. From this standard form, we can then translate each component of the finite automaton into a component of an equivalent regular expression.

The other direction of our procedure, taking us from regular expression back to finite automaton, will break down a regular expression into its constituent parts and then build up an equivalent finite automaton piece-by-piece.

Theorem 28. *A language A is regular if and only if there exists a regular expression $\boldsymbol{r}$ such that $L(\boldsymbol{r}) = A$.*

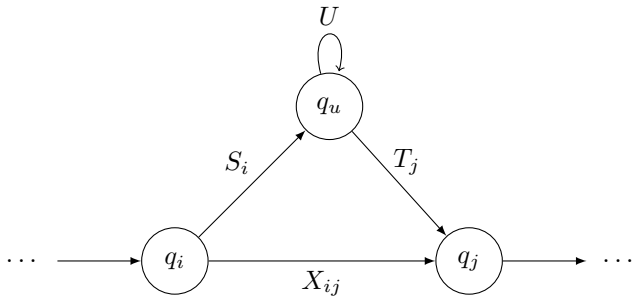
Proof. ($\Rightarrow$): To prove this direction of the statement, we will take a deterministic finite automaton recognizing the language A, and then convert the finite automaton to a regular expression. We will use a *state elimination algorithm* to perform this conversion.

Note that, for this proof only, we will assume that the transitions of our finite automaton can be labelled by regular expressions and not just symbols.

Suppose that we are given a deterministic finite automaton $\mathcal{M}$ such that $L(\mathcal{M}) = A$. Further suppose, without loss of generality, that there exists at most one transition between any two states of $\mathcal{M}$; we can make this assumption since multiple transitions between two states on symbols $a_1, \ldots, a_n$ can be replaced by the single transition on the regular expression $a_1 + \cdots + a_n$.

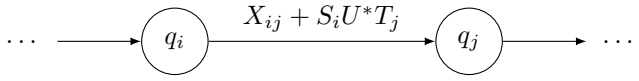
If $\mathcal{M}$ contains no final state, then $A = \emptyset$ and we are done. Otherwise, if $\mathcal{M}$ contains multiple final states, convert them to non-final states and add epsilon transitions from each former final state to a new single final state. If the initial state is also a final state, make a similar change to the initial state.

Now, we eliminate all states q_u of $\mathcal{M}$ that are neither initial nor accepting. Suppose that $\mathcal{M}$ contains the following substructure:



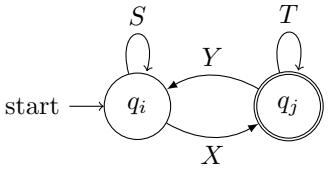
In this substructure, all transitions from states $q_i \neq q_u$ to state q_u are labelled by a regular expression S_i; all transitions from state q_u to states $q_j \neq q_u$ are labelled by a regular expression T_j, and for all such states q_i and q_j the transition between these states is labelled by a regular expression X_{ij}, or $\emptyset$ if no such transition exists. Lastly, any loop from q_u to itself is labelled by a regular expression U, or $\emptyset$ if no loop exists.

We may eliminate state q_u from $\mathcal{M}$ as follows: for each pair of states q_i and q_j, the regular expression X_{ij} on the transition is replaced by $X_{ij} + S_i U^* T_j$.



We then repeat this procedure for all non-initial and non-final states until the only states remaining in the finite automaton are the single initial and final states.

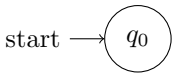
Suppose that, at this stage of the algorithm, our finite automaton is of the following form, where S, T, X, and Y are regular expressions:



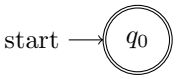
If any of these transitions do not exist, then we simply add them to the finite automaton labelled by $\emptyset$. Then the language recognized by $\mathcal{M}$ is represented by the regular expression $S^* X (T + Y S^* X)^*$.

($\Leftarrow$): To prove this direction of the statement, we will convert a regular expression $\boldsymbol{r}$ to a nondeterministic finite automaton $\mathcal{M}$ using a construction known as the *McNaughton–Yamada–Thompson algorithm*. We consider each of the basic regular expressions:

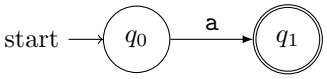
1. If $\boldsymbol{r} = \emptyset$, then $L(\boldsymbol{r}) = \emptyset$ and this language is recognized by the following nondeterministic finite automaton:



2. If $\boldsymbol{r} = \epsilon$, then $L(\boldsymbol{r}) = \{\epsilon\}$ and this language is recognized by the following nondeterministic finite automaton:



3. If $\boldsymbol{r} = a$ for some $a \in \Sigma$, then $L(\boldsymbol{r}) = \{a\}$ and this language is recognized by the following nondeterministic finite automaton:

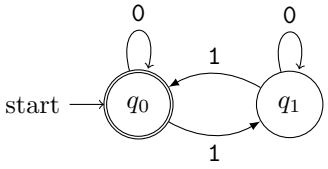


4. If $\boldsymbol{r} = \boldsymbol{r}_1 + \boldsymbol{r}_2$ for some regular expressions $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$, then the corresponding language is recognized by the nondeterministic finite automaton constructed in the proof of Theorem 21.
5. If $\boldsymbol{r} = \boldsymbol{r}_1\boldsymbol{r}_2$ for some regular expressions $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$, then the corresponding language is recognized by the nondeterministic finite automaton constructed in the proof of Theorem 22.
6. If $\boldsymbol{r} = \boldsymbol{r}^*$ for some regular expression $\boldsymbol{r}$, then the corresponding language is recognized by the nondeterministic finite automaton constructed in the proof of Theorem 23.

In each case, we can convert the basic regular expression to a nondeterministic finite automaton, and we can then determinize the overall finite automaton using our procedure from Theorem 26. We therefore end up with a deterministic finite automaton recognizing the same language represented by the original regular expression, and any language recognized by a finite automaton is regular by Definition 14. □

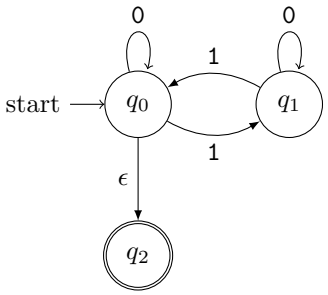
As an illustration of the state elimination algorithm we used in one direction of our proof, let us consider a small example of converting a deterministic finite automaton to a regular expression.

Example 29. Consider the following deterministic finite automaton $\mathcal{M}$:

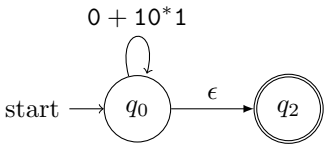


This finite automaton recognizes the language $L(\mathcal{M}) = \{w \mid w \text{ contains an even number of } \texttt{1}\texttt{s}\}$.

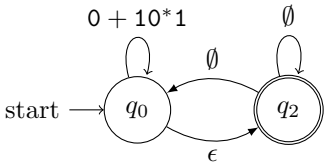
Since the initial state of $\mathcal{M}$ is also a final state, we begin by creating a new final state, converting the initial state to be nonaccepting, and adding an epsilon transition from the initial state to our new final state.



We now use our state elimination algorithm to remove q_1, which is the only state that is neither initial nor accepting. There exists a single transition from q_0 to q_1 and a single transition from q_1 to q_0. Let $S_0 = \mathtt{1}$, $T_0 = \mathtt{1}$, $X_{00} = \mathtt{0}$, and $U = \mathtt{0}$. We can then eliminate the state q_1 by relabelling the loop on q_0 to use the regular expression $X_{00} + S_0 U^* T_0 = \mathtt{0} + \mathtt{10^*1}$.



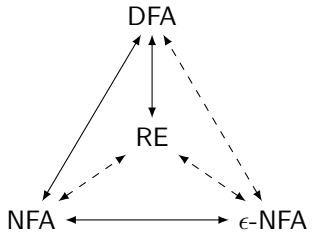
We add the missing transitions to obtain a finite automaton of the form specified in the proof of Theorem 28:



Consequently, the regular expression corresponding to this finite automaton is $(\mathtt{0}+\mathtt{10^*1})^*\epsilon(\emptyset+\emptyset(\mathtt{0}+\mathtt{10^*1})^*\epsilon)^*$. Using our rules for operations applied to empty words and empty languages, this regular expression simplifies to $(\mathtt{0} + \mathtt{10^*1})^*$.

4.4 Kleene's Theorem

Let's now review all that we've done by drawing a *Scutum Fidei*-esque diagram connecting each of our models of computation:



In our diagram, a solid line indicates that we have a method of directly converting between two models of computation, while a dashed line indicates that we have an indirect method—say, by performing two consecutive conversion steps.

All of our conversions considered apart may seem like nothing more than mechanical procedures or unimportant intermediate steps that we can employ in some larger system. However, taken together as we did in our diagram, these conversions reveal what might reasonably be called the most important theorem in the entire study of regular languages.

Theorem 30 (Kleene's theorem)**.** *A language R is regular if it satisfies any of the following equivalent properties:*

1. *There exists a deterministic finite automaton $\mathcal{M}_D$ with $L(\mathcal{M}_D) = R$;*
2. *There exists a nondeterministic finite automaton $\mathcal{M}_N$ with $L(\mathcal{M}_N) = R$; or*
3. *There exists a regular expression $\boldsymbol{r}$ with $L(\boldsymbol{r}) = R$.*

Note that we don't need to prove anything here—the proof of Kleene's theorem is baked into the descriptions of each of our conversion procedures!