Theory of Computation

Contents

1 Finite Automata

1.1 Deterministic Finite Automata

A deterministic finite automaton[1] (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set whose elements are states.
- Σ is a finite input alphabet, whose elements are input symbols.
- $\delta\colon Q \times \Sigma \to Q$ is the transition function.
- $q_0 \in Q$ is the start state (or initial state).
- $F \subseteq Q$ is the set of accept states (or final states).

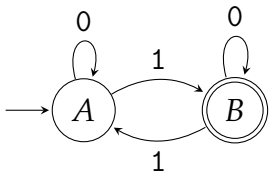
The transition function may be specified using a transition table, which has rows indexed by the states and columns indexed by the input symbols, with the entry in the row of q and column of s being the value of $\delta(q, s)$). See Example 1.1.

A DFA has a graphical representation as a directed graph in which each vertex is a state and each edge is a transition. That is, if $\delta(q_1, s) = q_2$ (where q_1 and q_2 are states and s is an input symbol), then the graph has an edge labelled s from the vertex q_1 to the vertex q_2. It is usual to depict the vertices by circles, with the vertex label (name of the state) written inside the circle, if necessary. The start state is indicated by an arrow without a source entering it, and accept states are indicated by using double-circles. See Example 1.1.

Example 1.1. Consider a DFA $M = (Q, \Sigma, \delta, A, F)$, where $Q = \{A, B, C\}$, $\Sigma = \{0, 1\}$, $F = \{B\}$, and δ is as defined by the transition table given below.

δ	0	1
A	A	B
B	B	A

The graphical representation of M is shown below.



An input to a DFA $M = (Q, \Sigma, \delta, q_0, F)$ is a string over Σ – i.e. a finite sequence of symbols of Σ. If $w = s_1 s_2 \cdots s_k$ is an input string, and $q_1, q_2, \ldots, q_k \in Q$ such that

$$\delta(q_0, s_1) = q_1,\ \delta(q_1, s_2) = q_2,\ \ldots,\ \delta(q_{k-1}, s_k) = q_k,$$

then M is in the state q_k after reading the input w. If $q_k \in F$ (i.e. q_k is an accept state), then M accepts w, otherwise it rejects w. For instance, the DFA M given in Example 1.1 is in the state A after reading the input `0110`, and is in the state B after reading `0100`. Since $A \notin \{F\}$ and $B \in \{F\}$, it rejects `0110` and accepts `0100`.

Given the graphical representation of a DFA M, to check which state M is in after reading an input $w = s_1 \cdots s_k$, simply begin at the start state of M and follow the arrows labelled $s_1, \ldots, s_k$. For instance, in Example 1.1, the input `0110` corresponds to the sequence of states A, A, B, A, A, and the input `0100` corresponds to the sequence of states A, A, B, B, B (note

[1] The plural of *automaton* (ɔːˈtɒmətən or ɔːˈtɒməˌtɒn) is *automata* (ɔːˈtɒmətə).

that in both cases, the first A is the start state, and only the next four states are the results of transitions).

A language over an alphabet Σ is a set of strings over Σ. In particular, the language Σ^* is the set of all strings over Σ. Thus, a language over Σ is any subset of Σ^*. If M is a DFA with input alphabet Σ, then M accepts the language L if L is the set of all strings accepted by M. For example, the DFA M in Example 1.1 accepts the language of all binary strings containing an odd number of `1`s.

Note. Although we say simply that M accepts L, this means not only that every string in L is accepted by M, but also that every string *not* in L is *rejected* by M.

Exercise 1.1. Construct a DFA that accepts all binary strings with an even number of `0`s.

Solution.

Exercise 1.2. Construct a DFA that accepts all strings over the alphabet $\{\texttt{a}, \texttt{b}\}$ that begin in `aab`.

Solution.

Observe that two of the states of the DFA given in the solution of Exercise 1.2 are states that, once entered, cannot be left – i.e. every transition out of such each of these states is into that state itself. Such a state is a trap state. One of these two trap states is an accept states – which indicates that any input that leads to this state is accepted regardless of what the rest of the input contains – while the other trap state is a non-accept state – which indicates that any input leading to this state is *permanently* rejected and that no further "correction" can be made to get the input accepted.

1.2 Nondeterministic Finite Automata

A nondeterministic finite automaton (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

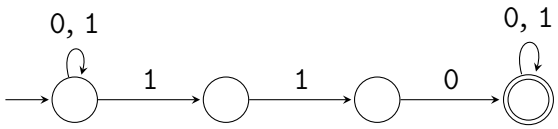
- Q is a finite set of states.
- Σ is a finite input alphabet.
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ is the transition function.
- $q_0 \in Q$ is the start state.
- $F \subseteq Q$ is the set of accept states.

Note that the transition function has codomain 2^Q, the power set of Q. That is, each transition of a nondeterministic finite automaton is from one state into possibly several states or even to no state. The interpretation of this is that at each step, for a given input, the machine may have many options of the state to transition to, and an input string is accepted if there is at least one path from the start state to an accept state with arrows labelled by the input. More formally, an input $s_1 \cdots s_n$ is accepted by an NFA $M = (Q, \Sigma, \delta, q_0, F)$, if and only if there exist states $q_1, \ldots, q_k \in Q$, with $q_k \in F$, such that

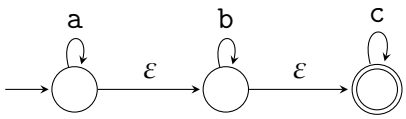
$$q_1 \in \delta(q_0, s_1),\ q_2 \in \delta(q_1, s_2),\ \ldots,\ q_k \in \delta(q_{k-1}, s_n).$$

A transition labelled by ε, i.e. a transition of the form $\delta(q, \varepsilon)$, is an epsilon transition, and does not consume an input symbol. An NFA also has a graphical representation similar to that of a DFA.

Example 1.2. The NFA shown below accepts all binary strings that contain `110` as a (contiguous) substring.



Example 1.3. The NFA shown below accepts all strings over the alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ in which the symbols appear in alphabetical order.



1.3 Conversion of NFA to DFA

Trivially, a deterministic finite automaton can be considered as a nondeterministic finite automaton. That is, NFAs are at least as powerful as DFAs, in the sense that any language accepted by a DFA is also accepted by an NFA. Now we will see that conversely, NFAs are not strictly more powerful than DFAs. Any NFA can be converted to an equivalent DFA[2], by the subset construction. Each transition of an NFA is from one state to a set of states, whereas any transition of a DFA is from one state to a single state. The main idea of the subset construction is to consider each set of states of an NFA as a single state of a DFA. The conversion procedure is described below.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA (without epsilon transitions). Define a DFA $D_M = (Q', \Sigma, \delta', q_0', F')$ as follows:

1. Every set of states of the NFA M becomes a single state of the DFA D_M. That is, $Q' = 2^Q$.
2. The start state of D_M is the singleton set containing the start state of M. That is, $q_0' = \{q_0\}$.
3. The transition function δ' of D_M is given by

$$\delta'(\{q_1, \ldots, q_k\}, s) = \bigcup_{i=1}^{k} \delta(q_i, s).$$

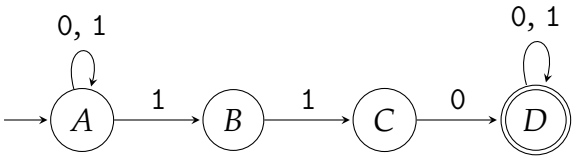
[2]Two finite automata are equivalent if the language accepted by them is the same

That is, given a state q' of D_M, which is a set of states of the NFA M, and an input symbol $s \in \Sigma$, for each state $q \in q'$ $\delta(q, s)$ is some set of states of M. Combine all these sets $\delta(q, s)$ for $q \in q'$ (via union) to get $\delta'(q', s)$.

4. Each state of D_M that contains at least one accept state of M becomes an accept state. That is,
$$F' = \{ q' \in Q' \mid q' \cap F \neq \varnothing \}.$$

If M has epsilon transitions, then $\delta'(\{q_1, \ldots, q_k\}, s)$ will contain, in addition to the states described above, the states of M that can be reached from each state in $\delta(q_i, s)$ via any sequence epsilon transitions. Similarly, q_0' will be the set of all states of M that can be reached from q_0 via (a sequence of) epsilon transitions.

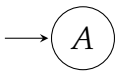
Example 1.4. Consider the NFA given in Example 1.2.



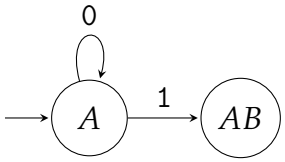
Here, the initial state is $q_0 = A$, and the set of accept states is $F = \{D\}$. Now, the equivalent DFA D_M will $2^4 = 16$ states, namely $\varnothing, \{A\}, \{B\}, \{C\}, \{D\}, \{A, B\}, \{A, C\}, \ldots, \{A, B, C, D\}$. For the sake of simplicity, we will denote these as $\varnothing, A, B, C, D, AB, AC, \ldots, ABCD$.

Note that many of these states may be unreachable – i.e. there will be no path from the start state to them. We will not include them in the graphical representation of D_M, even though, formally, they are part of D_M. To find all the reachable (i.e. not unreachable) states, we will begin with the start state, and iteratively add the other states while computing the transitions from the current states corresponding to each input symbol.

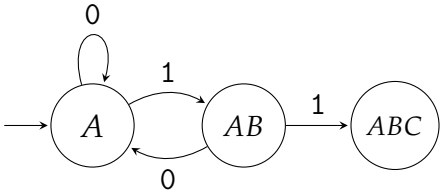
1. Initially, we have only the start state, A.



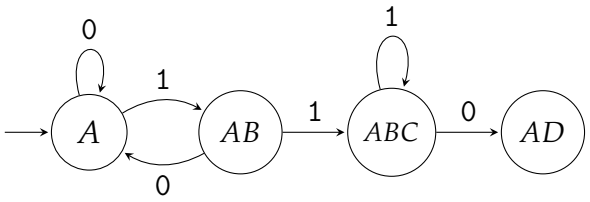
2. Next, $\delta'(A, 0) = \delta(A, 0) = \{A\}$ (written as just A) and $\delta'(A, 1) = \delta(A, 1) = \{A, B\}$ (written as just AB).



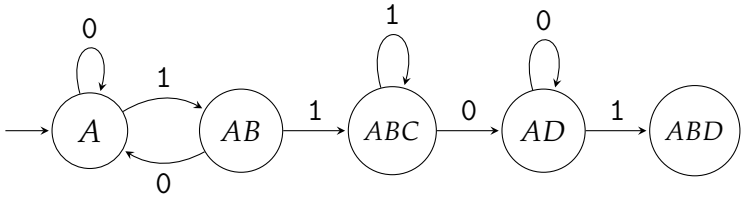
3. Now, $\delta'(AB, 0) = \delta(A, 0) \cup \delta(B, 0) = \{A\} \cup \varnothing = \{A\}$ (i.e. A) and $\delta'(AB, 1) = ABC$.



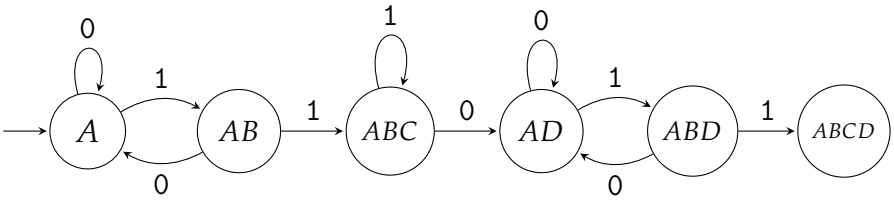
4. Similarly, $\delta'(ABC, 0) = AD$ and $\delta'(ABC, 1) = ABC$.



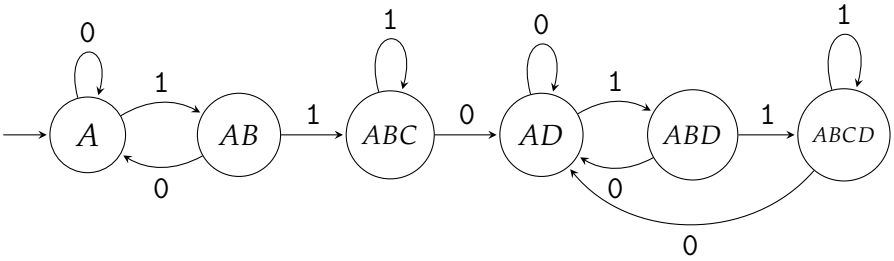
5. Similarly, $\delta'(AD, 0) = AD$ and $\delta'(AD, 1) = ABD$.



6. Similarly, $\delta'(ABD, 0) = AD$ and $\delta'(ABD, 1) = ABCD$.

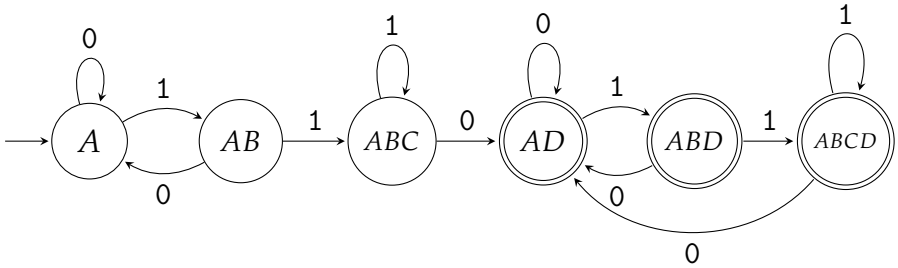


7. Finally, $\delta'(ABCD, 0) = AD$ and $\delta'(ABCD, 1) = ABCD$.

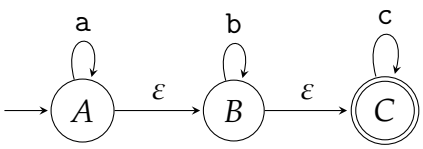


   As no new state is added in this step, all the other states (not shown here) are unreachable.

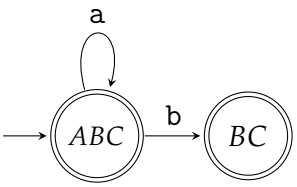
Now, the states AD, ABD, and $ABCD$ all contain D, which is the only accept state of the original NFA. Hence, these states are accept state of the DFA.



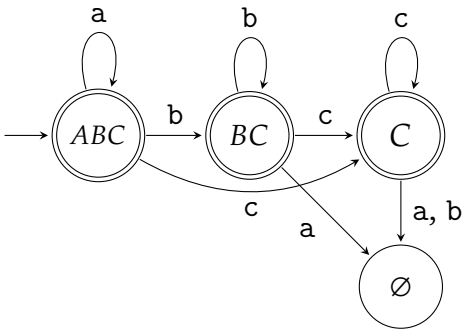
Example 1.5. Consider the NFA given in Example 1.2.



Here, the start state is A, but B can be reached from A via an epsilon transition, and C can be reached from B via an epsilon transition. Hence, the start state of the converted DFA will be ABC, which will also be an accept state, since it contains C, an accept state of the original NFA. Next, $\delta(A, \mathtt{a}) = A$, $\delta(B, \mathtt{a}) = \delta(C, \mathtt{a}) = \varnothing$. But B and C can be reached from A by epsilon transitions. Therefore, $\delta'(ABC, \mathtt{a}) = ABC$. Similarly, $\delta'(ABC, \mathtt{b}) = BC$, which will also be an accept state. Similarly, $\delta'(ABC, \mathtt{c}) = C$, also an accept state.



But now, $\delta(B, \mathtt{a}) = \delta(C, \mathtt{a}) = \varnothing$. Hence, $\delta'(BC, \mathtt{a}) = \varnothing$. Note that $\varnothing$ will be state from which there are no transitions into any other states – i.e. it is a trap state. Hence, $\delta'(\varnothing, \mathtt{a}) = \delta'(\varnothing, \mathtt{b}) = \varnothing$. Proceeding thus, we find that $\delta'(BC, \mathtt{b}) = BC$, $\delta'(BC, \mathtt{c}) = C$, $\delta'(C, \mathtt{a}) = \delta'(C, \mathtt{b}) = \varnothing$, and $\delta'(C, \mathtt{c}) = C$.



1.4 Product Construction

If L_1 and L_2 and languages over the same alphabet Σ, and M_1 and M_2, respectively, are the NFAs accepting them, then the intersection of L_1 and L_2,

$$L_1 \cap L_2 = \{\, w \in \Sigma^* \mid w \in L_1 \text{ and } w \in L_2 \,\}$$

is accepted by the product of M_1 and M_2, defined as follows: Let $M_1 = (Q_1, \Sigma, \delta_1, q_{10}, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_{20}, F_2)$. Then their product $M_1 \times M_2 = (Q, \Sigma, \delta, q_0, F)$, where

(i) $Q = Q_1 \times Q_2$, the Cartesian product of Q_1 and Q_2. That is, each state of $M_1 \times M_2$ is a pair of states, the first from M_1 and the second from M_2.
(ii) For each $q_1 \in Q_1$, $q_2 \in Q_2$, and $s \in \Sigma$, $\delta((q_1, q_2), s) = (\delta_1(q_1, s), \delta_2(q_2, s))$.
(iii) $q_0 = (q_{10}, q_{20})$.
(iv) $F = F_1 \times F_2$.

2 Regular Expressions

A regular expression (RE) over an alphabet Σ is an expression consisting of one or more of the symbols in Σ or ε or $\varnothing$, one or both of the operators + and *, and parentheses (and), whose syntax is defined recursively as follows:

1. For each $s \in \Sigma$, s is a regular expression, and ε and $\varnothing$ are regular expressions.
2. If r is a regular expression, then so is $(r)^*$.
3. If r_1 and r_2 are regular expressions, then so are $r_1 + r_2$ and $(r_1)(r_2)$.

Note. If r is a single symbol (i.e. $r \in \Sigma$), then $(r)^*$ can also be written as r^*; $(r)(r')$ can also be written as $r(r')$; and $(r')(r)$ can also be written as $(r')r$.

Example 2.1. The following are some regular expressions over $\Sigma = \{0, 1\}$:

$$0 + 1, \quad 0(0 + 1), \quad (0 + 1)^*110(0 + 1)^*, \quad 1^*0^* + 0^*1^*, \quad 0^*10^*(0^*10^*1)^*0^*, \quad 0(0 + 1)^*0 + 1(0 + 1)^*1$$

Each RE represents or generates a particular language. To rigorously define the language generated by an RE, we will first define some operations on languages.

Let L_1 and L_2 be two languages over the same alphabet Σ. Recall that this means L_1 and L_2 are sets of strings of symbols in Σ, i.e. $L_1, L_2 \subseteq \Sigma^*$.

1. The union of L_1 and L_2 is $L_1 \cup L_2$, defined as their union as sets.

$$L_1 \cup L_2 = \{\, w \in \Sigma^* \mid w \in L_1 \text{ or } w \in L_2 \,\}$$

2. The concatenation of L_1 and L_2 is L_1L_2, the language of all strings obtained by concatenating a string of L_1 with a string of L_2.

$$L_1L_2 = \{\, w_1w_2 \mid w_1 \in L_1,\ w_2 \in L_2 \,\}$$

3. The Kleene star[3] of L_1 is L_1^*, the language of all strings obtained by concatenating any finite number of strings of L_1 together.

$$\begin{aligned} L_1^* &= \{\, w_1w_2 \dots w_n \mid w_i \in L_i,\ i = 1, \dots, n,\ n \in \mathbb{N}_0 \,\} \\ &= \{\varepsilon\} \cup L_1 \cup L_1L_1 \cup L_1L_1L_1 \cup \cdots \end{aligned}$$

The language generated by an RE can now be recursively defined as follows:

1. The language generated by the RE s, where $s \in \Sigma$ (a single symbol), is $\{s\}$ and the language generated by the RE ε is ε.
2. The language generated by the RE $\varnothing$ is the empty language $\varnothing = \{\}$. Note that this language does *not* contain the empty string!
3. If r_1 and r_2 are two REs over the same alphabet Σ, that generate languages L_1 and L_2, respectively, then
   (i) the RE $r_1 + r_2$ generates the language $L_1 + L_2$
   (ii) the RE $(r_1)(r_2)$ generates the language L_1L_2
   (iii) the RE $(r_1)^*$ generates the language L_1^*.

Example 2.2. Let $\Sigma = \{0, 1\}$.

1. The RE $0 + 1$ generates the language $\{0, 1\}$.
2. The RE $0(0 + 1)$ generates the language $\{00, 01\}$.
3. The RE $(0 + 1)^*110(0 + 1)^*$ generates the language of all binary strings containing 110 as a substring.
4. The RE $0^*1^* + 1^*0^*$ generates the language of all binary strings that are either a string of (zero or more) 0s followed by a string of 1s, or a string of 1s followed by a string of 0s.

[3] Named after the mathematician Stephen Cole Kleene, pronounced `klay-nee` (ˈkleɪni).

Exercise 2.1. Let $\Sigma = \{0, 1\}$.

1. What is the language generated by $0(0 + 1)^*0 + 1(0 + 1)^*1$?
2. What is the language generated by $0^*10^*(0^*10^*1)^*0^*$?

Solution.

2.1 Conversion of NFA to RE

Any nondeterministic finite automaton can be converted to an equivalent[4] regular expression. The conversion procedure involves first reducing the given NFA, by eliminating one state at a time, to intermediate finite automata of a new kind, until just one start state and one accept state remain. A generalised (nondeterministic) finite automaton (GFA or GNFA) is a finite automaton in which the arrows are labelled by REs, and a transition along this arrow is made if the next few characters of the input match the RE. That is, if the arrow from state q_1 to state q_2 is labelled by the regular expression r, and the unread part of the input when q_1 is reached is $s_1 s_2 \cdots s_n$, then a transition can be made to q_2 is $s_1 s_2 \cdots s_i$ is a string generated by r for some $i \leq k$. Trivially, every NFA is a GFA, since any single symbol or ε is an RE, and a transition of an NFA consists of reading one symbol and moving along one arrow, or reading no symbol (i.e. an empty string) and making an epsilon transition.

The procedure for converting an NFA to RE is as follows. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA. Furthermore, assume that q_0 has no arrows entering it and F contains a single state q_f with no arrows leaving it – in both cases, even self-loops are not allowed[5]. Now, iteratively reduce M by eliminating one state other than q_0 and q_f at a time as given below:

1. Let q be any state different from q_0 and q_f, and let r be the label on the self-loop on q.
2. For each state $q_1 \neq q$ with a transition into q labelled r_1, and each state $q_2 \neq q$ into which there is a transition from q labelled r_2, let $r_{12} = r_1 r^* r_2$ (suitably parenthesised), where r is the regular expression on the self-loop on q (if q has no self-loop, then this is just $r_1 r_2$).
3. If there is no arrow from q_1 to q_2, add an arrow labelled r_{12}. If an arrow already exists from q_1 to q_2, replace its label, say r', by $r' + r_{12}$.
4. Delete q.

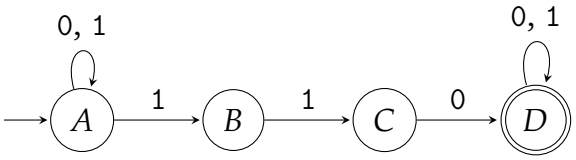
When q_0 and q_f are the only states remaining, the label on the arrow from q_0 to q_f is the RE equivalent to the NFA M.

Note. The order in which the states are eliminated does not matter, although the REs obtained may differ (but be equivalent) according to the order followed.

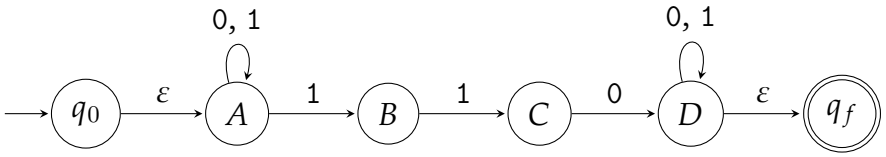
Example 2.3. Consider the NFA given in Example 1.2.

[4] An RE equivalent to an NFA is one that generates the language accepted by the NFA.

[5] It is easy to see that any NFA can be converted to an equivalent NFA satisfying this condition by using epsilon transitions. If q_0 has an arrow entering it, then make a new start state q_0' with an epsilon transition into q_0, which will no longer be the start state. Similarly, if q_f has an arrow leaving it, make it a non-accept state and add a new accept state q_f' with an epsilon transition into it from q_f. If M has multiple accept states, make all of them non-accept states, and add a new accept state q_f with an epsilon transition into it from each of the old accept states.

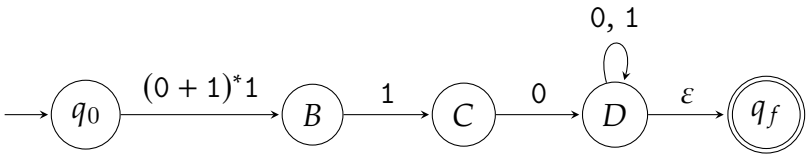


The start state A has two arrows entering it and the accept state D has two arrows leaving it (in each case, self-loops labelled 0 and 1). Therefore, we first add a new start state q_0 with an epsilon transition into A (no longer a start state), and a new accept state q_f with an epsilon transition into it from D, which will be made a non-accept state.

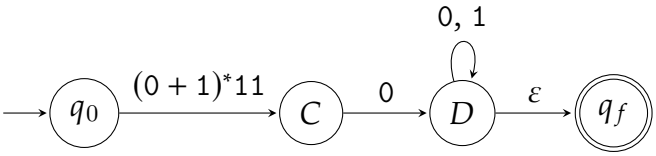


Now we eliminate the non-initial and non-accept states one by one.

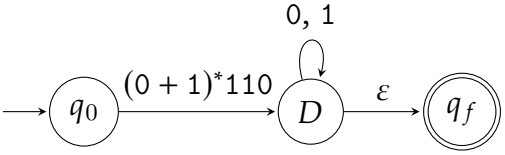
1. If we choose to eliminate A first, we first note that there is one arrow, labelled ε, coming into A from q_0, and one arrow, labelled q, going from A into B. Also, there are self-loops on A labelled 0 and 1. Therefore, we add an arrow from q_0 to B labelled $\varepsilon(0+1)^*1 = (0+1)^*1$, and then delete A.



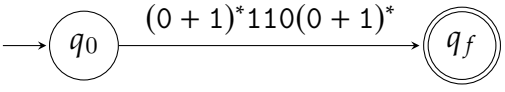
2. Next, if we wish to eliminate B, we replace the arrow from q_0 to B labelled $(0+1)^*1$ and the arrow labelled 1 from B to C by the arrow $(0+1)^*11$ from q_0 to C.



3. Similarly, we may eliminate C next.

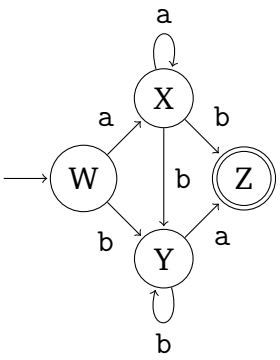


4. Finally, we eliminate D.



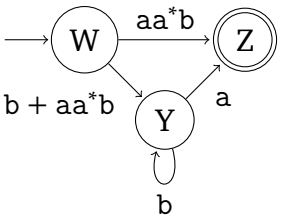
Thus, an RE equivalent to the given NFA is $(0+1)^*110(0+1)^*$. We easily verify that indeed, this is an equivalent RE. The original NFA accepts all binary strings containing the substring 110. Every such string can be written as some binary string (i.e. a string generated by $(0+1)^*$), followed by 110, (generated by 110) followed by any binary string (generated by $(0+1)^*$)).

Example 2.4. Consider the NFA shown below.

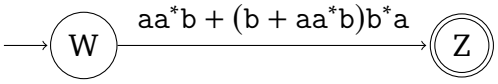


This NFA is already in the form necessary to apply the conversion procedure, as the start state W has no arrows entering it, and there is a unique accept state Z with no arrows leaving it.

1. There is only one arrow from a different state entering X, which is from W, there are two arrows leaving X, which are to Y and Z, and there is a self-loop on X labelled a. On eliminating X, we get the following GFA.



2. Next, we eliminate Y.



Thus, the equivalent RE is $\mathtt{aa^*b + (b + aa^*b)b^*a}$.

Exercise 2.2. Convert the NFA given below to an RE.