Regular languages

Probably, most of you have had a course on regular and context-free languages. As a warm up, we review the theory of finite automata and regular languages. We follow Chapter 1 in the book of Sipser. Except for the definition of a finite automata, no materials in the course depend directly on results in this section. Although, some experience in programming finite state automata might be useful to program Turing machines.

1 Deterministic finite automata

A finite automata is a device whose input are strings over an alphabet A and returns either "accept" or "reject". Typically, they are schematically represented as in figure 1. Initially, a "finger" is placed in the start state. The machine processes it's input string from the left to the right. For each letter, it moves the finger in the next state by following the arrow which is labeled with the same letter. If, after processing the last letter, the finger is in a state marked with a double circle, the machine returns "accept", otherwise, it returns "reject".

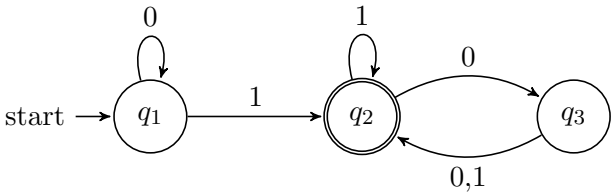


Figure 1: Schematic representation of a DFA. That accepts 1, 01, 11101, but not 0, 10, 11.

Definition 1. *A* deterministic finite automata *or* DFA (Q, A, δ, q_0, F) *is a* 5*-tuple, where*

- Q *is a finite set whose elements are called the* states*,*
- A *is a finite set called the* alphabet*,*
- $\delta : Q \times A \to Q$ *is called the* transition function*,*
- $q_0 \in Q$ *is the* start state*,*
- $F \subseteq Q$ *is the set of* accept states*.*

Let A^* be the set of strings of elements of A. A DFA accepts a string $x = x_1 \dots x_n$ in A^* if there exists a list of states $s_1, \dots, s_{n+1}$ such that $s_1 = q_0$, $s_{n+1} \in F$ and $\delta(s_i, x_i) = s_{i+1}$ for all $i \leq n$.

A DFA recognizes *a language $L \subseteq A^*$ if and only if the set of strings it accepts equals L. A language is* regular *if there is a DFA that recognizes it.*

Example: $(\{q_1, q_2, q_3\}, \{0, 1\}, \begin{array}{c|cc} & 0 & 1 \\ \hline q_1 & q_1 & q_2 \\ q_2 & q_3 & q_2 \\ q_3 & q_2 & q_2 \end{array}, q_1, \{q_2\})$, see figure 1.

Exercises:

- Make a schematic representation of a DFA that recognizes the set of strings in $\{0, 1\}^*$ whose one but last symbol is 0.
- Show that the complement of a regular language is also regular.
- Explain why $L_n = \{x : x \text{ is a multiple of } n \text{ in binary}\}$ is regular for each $n = 0, 1, 2, 3, \ldots$.
- Make a schematic representation of a DFA that recognizes the set of strings in $\{a, b\}^*$ that either have an even number of a's or for which the number of b's is a multiple of 3.

Theorem 1. *If L_1 and L_2 are regular languages over the same alphabet, then $L_1 \cup L_2$ is a regular language.*

Proof. Let k_1 and k_2 be the number of states of the DFA's that recognize L_1 and L_2. If we create a new machine, consists of two DFA's that run in parallel on the same input, this new machine has at most $k_1 \times k_2$ different "states". Hence, we can simulate it with a single DFA's with at most $k_1 \times k_2$ states. The new machine accepts a string if at the end of the computations of both DFA's, at least one of them accepts the string.

More formally, let $(Q_1, A, \delta_1, s_1, F_1)$ and $(Q_2, A, \delta_2, s_2, F_2)$ be DFA's recognizing L_1 and L_2. We construct a new DFA (Q, A, δ, q_0, F):

- $Q = Q_1 \times Q_2$ (i.e. the set of pairs (q_1, q_2) with $q_1 \in Q_1$ and $q_2 \in Q_2$).
- δ is defined by $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$
- $q_0 = (s_1, s_2)$
- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$.

It follows easily that the construction recognizes $L_1 \cup L_2$. □

Note that in a similar way, we can prove that $L_1 \cap L_2$ is also regular. Because regular languages are closed under compliment, it follows that also $L_1 \setminus L_2$ is regular.

Remark: It is rather obvious that δ "simulates" the computations of the DFA's in parallel. In case we want to write a very detailed formal proof, we need to use induction (on the length of the computation).

Definition 2. *For $x, y \in A^*$ let $xy \in A^*$ be the concatenation of x and y. For $L, M \subseteq A^*$ let:*
- $LM = \{xy : x \in L \wedge y \in M\}$.
- $L^* = \{x_1 \ldots x_k : x_i \in L \text{ for all } i \leq k \text{ and for } k = 0, 1, \ldots\}$.

Explain why it is not obvious that the concatenation of two regular languages is also regular.

2 Nondeterministic finite automata

In computational complexity, nondeterministic computation plays an important role. We study the corresponding notion for automata. We show that nondeterministic and deterministic automata recognize the same class of languages. This implies that the concatenation of two regular languages is regular.

In a computation on a nondeterministic finite automata, for each state and for each element in the alphabet there might be several subsequent states. Hence, for every input string, there might exist many computation paths. A nondeterministic automata accepts a string if there exists at least one accepting computation path for the string. Let $\mathcal{P}(Q)$ denote the set of subsets of Q.

Definition (Tentative)**.** *A* nondeterministic finite automata (Q, A, δ, q_0, F) *is a* 5*-tuple, where* Q *and* A *are finite sets,* $q_0 \in Q$*,* $F \subseteq Q$ *and*

$$\delta : Q \times A \to \boldsymbol{\mathcal{P}(Q)}.$$

A NFA accepts $x_1 \dots x_n \in A^*$ *if there exists a list of states* $s_1, \dots, s_{n+1}$ *such that* $s_1 = q_0$*,* $s_{n+1} \in F$ *and* $\delta(q_i, x_i) \in q_{i+1}$ *for all* $i \leq n$*.*

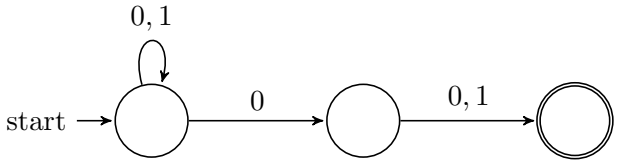


Figure 2: A nondeterministic automata that recognizes strings with a 0 in the one but last position.

To verify whether a string is accepted by such an NFA, we can write down all possible computations for the string in the form of a tree. At the i-th level, this tree contains all the states that appear at the i-th step of possible computations on this input. This tree might have branches for which the computation does not reach the end of the input, this happens if $\delta(q, x_i) = \emptyset$. See figure 2

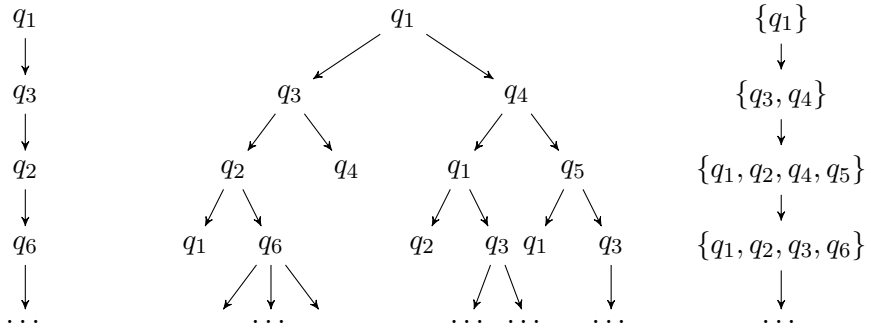


Figure 3: Computation tree corresponding to a DFA and a NFA.

We use an even more general definition of a nondeterministic automata for which a state in a computation step can change without reading a new input letter. Let ε denote the empty string. Note that $x\varepsilon = x$ for all $x \in A^*$.

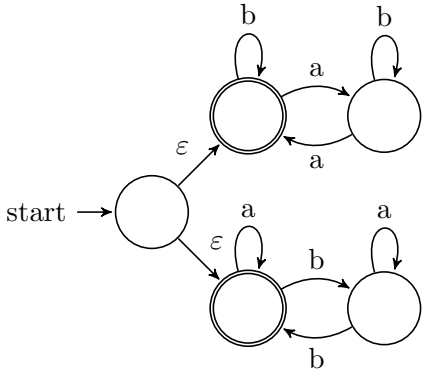


Figure 4: An NFA that recognizes the strings with an even number of a's or an even number of b's. Note that this construction shows that the union of two languages recognized by an NFA can also be recognized by an NFA.

Definition 3. *A* nondeterministic finite automata *or* NFA (Q, A, δ, q_0, F) *is a* 5*-tuple, where* Q *and* A *are finite sets,* $q_0 \in Q$*,* $F \subseteq Q$ *and*

$$\delta : Q \times (A \cup \{\boldsymbol{\varepsilon}\}) \to \mathcal{P}(Q).$$

An NFA accepts $x \in A^*$ *if there exists a list* $(s_1, y_1), \ldots (s_e, y_e)$ *of elements in* $Q \times (A \cup \{\varepsilon\})$ *such that the concatenation* $y_1 \ldots y_e$ *equals* x*,* $s_1 = q_0$*,* $s_e \in F$*, and* $s_{i+1} \in \delta(s_i, y_i)$ *for all* $i < e$*. An NFA* recognizes *a language in* A^* *if the set of strings it accepts equals the language.*

Note that if an NFA has no ε-arrows leaving from the start state, then it can be replaced by an NFA without ε-arrows. Why?

Exercise: Recall that if we change in a DFA all accepting states to normal states and all normal states to accepting states, then the new DFA recognizes the complement $A^* \setminus L$ of the language L recognized by the original machine. Show that this is not true for NFA's. (Hint: use one of the examples above.)

Theorem 2. *A language is regular if and only if some NFA recognizes it.*

Note that we can simulate an NFA using a schematic representation if we use several fingers. In each computation step i, for each finger in a state q, we place a finger in all possible next states $\delta(q, x_i) \subseteq Q$ (unless there is already a finger). After scanning the input x, we accept if there is at least one finger on an accepting state. There are finitely many configurations possible, hence the process can be simulated using a DFA.

We illustrate the procedure to convert an NFA to a DFA that accepts the same language. Consider the NFA that recognizes the strings with a 0 in the one but last position. The NFA has has states $\{a, b, c\}$, the states of the new machine corresponds to subsets of this set. The start state is $\{a\}$. If the input is 0, we can reach states a and b, hence we place a 0-arrow from $\{a\}$ to $\{a, b\}$. If the input is 1, we can only remain in a, hence we place a 1 arrow from $\{a\}$ to itself. All states can be reached from a or b using 0-arrows, hence we place a 0-arrow from $\{a, b\}$ to $\{a, b, c\}$, etc. In this way we obtain the following picture:

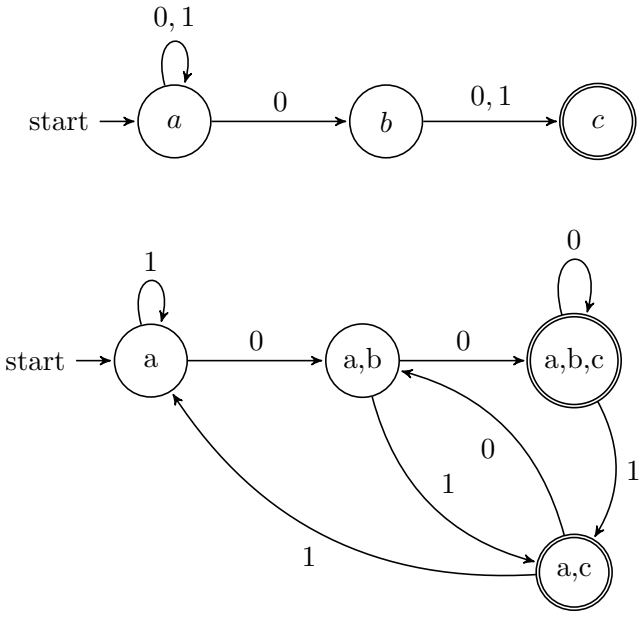


Figure 5: Transformation of an NFA to a DFA.

Proof. We need to show that for every NFA (Q, A, δ, q_0, F) there exists a DFA that recognizes the same language. First assume the NFA has no ε-arrows. In this case the proof follows by observing that the above procedure can be executed by a DFA $(Q', A, \delta', q_0', F)$:

- $Q' = \mathcal{P}(Q)$, (every state is a subset of Q),
- $\delta(q', a) = \bigcup_{q \in q'} \delta(q, a)$,
- $q_0' = \{q_0\}$,
- F' is the set of all states q' that contain an element of F.

Now we have to adapt the procedure for the case there are ε-arrows in the NFA. We adapt the simulation such that each time a finger arrives in a new state from which ε-arrows leave, the finger is split in several fingers which follow these arrows. To present a formal definition of the construction, we we use the E operator which maps a set $S \subseteq Q$ to the set $E(S) \supseteq S$ of all states that can be reached from S following any number of ε-arrows. It suffices to adapt the definition of δ and q_0':

- $\delta(q', a) = \bigcup_{q \in q'} E(\delta(q, a))$,
- $q_0' = E(\{q_0\})$,

□

Theorem 3. *If L and M are regular, then LM and L^* are regular.*

Exercise: proof this theorem.

3 Regular expressions

3.1 Intermezzo

To better understand the proof of the main result, we first consider a similar problem in algorithms.

Imagine there are n cities which are all connected by direct roads. However, some roads are slow: to travel from one city to another, it might be faster to travel through some well connected cities. More generally, suppose we are given a graph with n nodes labelled by $1, 2, \ldots, n$. There is a directed arrow from every node $i \leq n$ to any node $j \leq n$. Each such arrow has a weight which we call the distance from i to j and which we represent as $d_0(i, j)$. (In general it might happen that $d_0(i, j) \neq d_0(j, i)$.) In the example, the weight represents the travelling time from one city to another along the direct road. We want to compute a table of indirect distances, i.e., the minimal weight of a directed path connecting two nodes. In our example this reflects the minimal travelling times between cities.

We show that the table can be constructed using $O(n^3)$ arithmetic operations. For each $S \subset \{1, 2, \ldots, n\}$, Let $d(i, j; k)$ denote the shortest distance from i to j using a path that has only nodes in $\{1, 2, \ldots, k\}$. Let $d(i, j, 0) = d_0(i, j)$. For $k = 1, \ldots, n-1$ let

$$d(i, j; k) \leftarrow \min\{d(i, j; k-1), d(i, k; k-1) + d(k, j; k-1)\}.$$

Note that $d(i, j; n)$ indeed corresponds to the minimal distance of a path to go from node i to node j.

3.2 Definition

Regular expressions are often used in the design of compilers. They are defined by induction.

Definition 4. *A is a regular expression R if and only if*

- *$R = a$ for some $a \in A$,*
- *$R = \varepsilon$,*
- *$R = \emptyset$,*
- *$R_1 \cup R_2$ where R_1 and R_2 are regular expressions,*
- *$R_1 R_2$ where R_1 and R_2 are regular expressions,*
- *R_1^* where R_1 is a regular expression.*

With a regular expression we associate a language over A; which is obtained by induction: $a \leftrightarrow \{a\}$, $\varepsilon \leftrightarrow \{\varepsilon\}$, and the other cases are straightforward.

Examples:

1. $0^*10^* \leftrightarrow$ strings with exactly one 1,
2. $A^*1A^* \leftrightarrow$ strings with at least one 1,
3. $(AA)^* \leftrightarrow$ strings of even length,
4. $01 \cup 10 \leftrightarrow \{01, 10\}$,

5. $(0 \cup \varepsilon)(1 \cup \varepsilon) \leftrightarrow \{\varepsilon, 0, 1, 01\}$,

6. $1^*\emptyset \leftrightarrow \emptyset$,

7. $\emptyset^* \leftrightarrow \{\varepsilon\}$.

If two regular expressions R and L are associated with the same set, we say that they are equal. Note that $R \cup \emptyset = R$, $R\varepsilon = R$, but $R\emptyset = \emptyset$. The $\cup$ operator has lowest priority, the star operater has priority over concatenation, for example: $a \cup bc^* = \{a, b, bc, bcc, bccc, \dots\}$.

3.3 Equivalence with regular languages

Theorem 4. *A language is regular if and only if it is equivalent with a regular expression.*

Proof. It is easy to show by induction that the language associated with a regular expression is recognized by an NFA. Indeed, it is straightforward to write machines that recognize $R = \varepsilon$, $R = \emptyset$ and $R = \{a\}$ for some $a \in A$. The induction step easily follows from the results above: the regular languages are closed under concatenation, union and the star operation.

To show that there exists a regular expression for the language recognized by a DFA, we transform an automaton in small steps to a regular expression. For this we need a hybrid form of regular expressions and automata. A generalized nondeterministic finite automaton or GNFA is schematically represented in the same way as an NFA, but now each arrow is labeled by a regular expression. If we go from one state to another, we might use several letters from the input; the string of used letters should satisfy the regular expression. The automaton is nondeterministic in the sense that it might happen that a string can be processed in several ways. Note that any NFA is a GNFA because every $a \in A$ can be viewed as a regular expression.

A GNFA accepts its input if it can be processed such that the last state is an accepting state. For technical reasons, we assume that all states are connected by an arrow in both directions (possibly containing the $\emptyset$ expression), except for the start state, which has no incoming arrows, and the accept state, which has no outgoing arrows.

Definition. *A* GNFA $(Q, A, \delta, q_{start}, q_{accept})$ *where Q and A are finite sets, $q_{start}, q_{accept} \in Q$ and*

$$\delta : (Q \setminus q_{accept}) \times (Q \setminus q_{start}) \to \text{ the set of regular expressions over } A.$$

A GNFA accepts *a string $w \in A^*$ if and only if there exist*

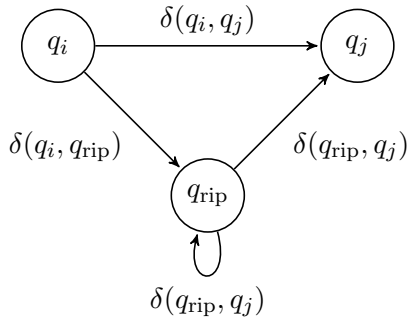
- *a sequence $q_1, \dots, q_{k+1}$ in Q with $q_1 = q_{start}$ and $q_{k+1} = q_{accept}$,*
- *a sequence $w^1, \dots, w^k$ in A^* such that $w^1 w^2 \dots w^k = w$ and each w_i, $i = 1, \dots, k$ is modelled by the regular expression $\delta(q_i, q_{i+1})$.*

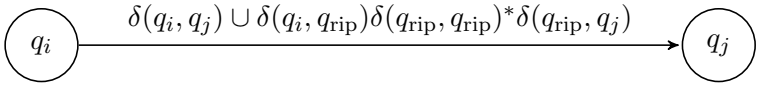
From a GNFA $(Q, A, \delta, q_{\text{start}}, q_{\text{accept}})$ we can remove a state $q_{\text{rip}} \in Q \setminus \{q_{\text{start}}, q_{\text{accept}}\}$ without changing the recognized language. The new GNFA is given by $(Q \setminus q_{\text{rip}}, A, \delta', q_{\text{start}}, q_{\text{accept}})$. where

$$\delta'(q_i, q_j) = \delta(q_i, q_j) \cup \delta(q_i, q_{\text{rip}})\delta(q_{\text{rip}}, q_{\text{rip}})^*\delta(q_{\text{rip}}, q_j),$$

see figure 3.3.

Lemma 5. *After this transformation, both GNFA's accept the same strings.*





Proof. Suppose that the original machine accepts w using an accepting sequence

$$\overset{q_{\text{start}}}{w_1} \cdots \overset{q}{w_k} q_{\text{accept}}$$

that does not contain q_{rip}, then it is also accepted by the transformed machine. Otherwise, concatenate all strings $w_{\ell-1}, w_\ell, \ldots, w_{\ell+e}$ corresponding to a consecutive run of states q_{rip} to obtain $w'_{\ell-1}$. For example, for $e = 1$ the accepting sequence

$$\overset{q_{\text{start}}}{w_1} \cdots \overset{q_i}{w_{\ell-1}} \overset{q_{\text{rip}}}{w_\ell} \overset{q_{\text{rip}}}{w_{\ell+1}} \overset{q_j}{w_{\ell+2}} \cdots \overset{q}{w_k} q_{\text{accept}}$$

is transformed to

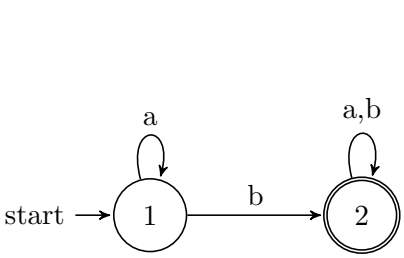
$$\overset{q_{\text{start}}}{w_1} \ldots \left(w_{\ell-1} \overset{q_i}{w_\ell} w_{\ell+1}\right) \overset{q_j}{w_{\ell+2}} \cdots \overset{q}{w_k} q_{\text{accept}}$$

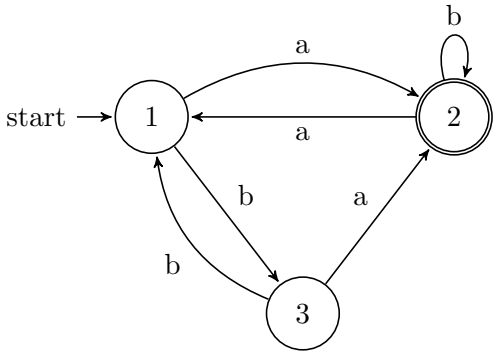
Vice versa, an accepting sequence for the transformed machine can be changed into an accepting sequence for the new machine. If for a pair $\overset{q_i}{w_\ell} \overset{q_j}{w_{\ell+1}}$ the string is modelled by $\delta(q_i, q_j)$ then the accepting sequence need not to be adapted in this position, otherwise, it is modelled by $\delta(q_i, q_{\text{rip}})\delta(q_{\text{rip}}, q_{\text{rip}})^*\delta(q_{\text{rip}}, q_j)$ we can now insert the appropriate amount of states q_{rip} and corresponding strings w to obtain an accepting sequence. □

It remains to explain how an NFA (Q, A, δ, q_0, F) is embedded in a GNFA $(Q', A, \delta', q_{\text{start}}, q_{\text{accept}})$. We add the start and accept states: $Q' = Q \cup \{q_{\text{start}}, q_{\text{accept}}\}$. We connect q_{start} to q_0 with an ε-arrow. We connect all accept states in F to q_{accept} with ε-arrows. Obviously, all a-arrows with $a \in A$ can be interpreted as regular expressions. All missing arrows between states $Q \setminus q_{\text{accept}}$ and $Q \setminus q_{\text{start}}$ are filled with $\emptyset$ arrows. On can now easily observe that this GNFA accepts the same strings as the original NFA.

Using the procedure, we remove states until only the start and accept state remain. For this machine there can be at most one arrow, and this arrow contains the regular expression that is equivalent to the language recognized by the NFA. □

Exercise. Apply the transformation in the proof to the following two automata to obtain a regular expression for the language they recognize.





4 The pumping lemma

The following result is useful to prove that a language is not regular. For any string y, let $y^0, y^1, y^2, \ldots$ be $\varepsilon, y, yy, \ldots$.

Theorem 6 (Pumping lemma)**.** *If L is a regular language, then there exists a number p (the pumping length) such that for all $w \in L$, w can be split in 3 parts $w = xyz$ such that $y \neq \varepsilon$, $|xy| \leq p$, and $xy^iz \in L$ for all $i = 0, 1, 2, \ldots$.*

Note that we cannot omit the condition $y \neq \varepsilon$, otherwise the theorem is trivial.

Proof. If an NFA has processed ℓ letters of its input, the corresponding computation uses exactly $\ell+1$ states. If the automaton has at most ℓ states, this means that one state has been visited at least twice. y now corresponds to the letters from the input that were red after the first time this state was visited until the next time this state was visited. This y satisfies the conditions of the theorem.

More formally, let (Q, A, δ, q_0, F) be the NFA that recognizes L. Let $p = |Q|$. Let $w = w_1 \ldots w_n \in L$ with $n \geq p$, and let $s_1 \ldots s_{p+1}$ be the first $p+1$ states in the sequence of an accepting computation of w (they correspond to the input $w_1 \ldots w_p$). Clearly, one state appears twice. Let $k < \ell \leq p+1$ such that $s_k = s_\ell$. Let $y = w_k \ldots w_{\ell-1}$ and let x and z be the pieces of w before and after y. Note that y must contain at least one letter and $|xy| \leq p$. Finally, observe that also xy^iz is accepted by the NFA. □

Exercises: Show that the following languages over $\{0, 1\}$ are not regular:

- $\{0^n1^n | n \geq 0\}$
- $\{w | w$ has an equal number of 0's and 1's $\}$
- $\{1^{n^2} | n \geq 0\}$
- $\{0^i1^j | i > j\}$ (hint: use "pumping down").