

Handout 1: DFAs and Regular Languages

Ruby Chang, Keir Dorchon, Zachary Thayer

Scope. This handout is for review before Quiz 1. On Quiz 1 and HW1, you may use all material from class up to and including the Myhill–Nerode theorem.

1. Review

Definition: Strings and languages

An **alphabet** Σ is a finite, nonempty set of symbols (sometimes called characters), such as $\{0, 1\}$ or $\{a, b, c, d, e\}$.

A **string**, s , over an alphabet, is a finite sequence of characters from that alphabet, e.g. 1000101.

The **length** of a string, denoted $|s|$, is the number of characters in s . Above, $|1000101| = 7$.

The **empty string** is denoted by ε and is defined such that $|\varepsilon| = 0$.

We define the set Σ^k to contain all strings of length k using symbols from Σ , and Σ^* as containing all strings over Σ .

A **language**, L , over Σ is a subset of Σ^* . We often define languages using mathematical notation but it is important to remember that a language is merely a set of strings. e.g., $L = \{010, 11, \varepsilon, 0000\}$ or $L = \{w \in \Sigma^* \mid w \text{ starts with a } 0\}$.

Lastly, the **concatenation** of x and y is denoted $x \circ y$, sometimes simply xy and is the string x followed by y . E.g. $101 \circ 11 = 10111$.

Definition: Deterministic finite automata

A **deterministic finite automaton (DFA)** is given formally by a five-tuple $D = (Q, \Sigma, \delta, q_0, F)$, where:

- Q is a finite set of **states**.
- Σ is the **alphabet** of input symbols.
- $\delta : Q \times \Sigma \rightarrow Q$ is the **transition function**. It takes a state and an input symbol and gives the next state. E.g. $\delta(q_0, 1) = q_1$ means that from state q_0 , reading a 1 takes the DFA to state q_1 .
- $q_0 \in Q$ is the **start state**, where the computation begins.
- $F \subseteq Q$ is the set of **accepting states**.

For every state and every alphabet symbol, there is exactly one next state. We can also specify a DFA using a **state diagram**, with its start state, accepting states, and transitions indicated (see examples in the practice problems below).

Definition: Computations and acceptance

For an input string $w = w_1 \cdots w_n$, each w_i is a character in Σ and $n = |w|$.

The **computation** of D on w is the sequence of states $(r_0, \dots, r_n)$, where:

$r_0 = q_0$: the computation starts in the start state.

$r_i = \delta(r_{i-1}, w_i)$ for $1 \leq i \leq n$: reading the next symbol w_i takes the DFA from state r_{i-1} to state r_i .

The DFA **accepts** w if $r_n \in F$, i.e. the computation ends in an accepting state.

The DFA **rejects** w if $r_n \notin F$, i.e. the computation ends in a nonaccepting state.

On the **empty string** ε , the computation is just (q_0) . Thus D accepts ε exactly when $q_0 \in F$.

Definition: Regular languages

The **language recognized** by D is $L(D) = \{w \in \Sigma^* \mid D \text{ accepts } w\}$, the set of all strings that D accepts.

A language L is **regular** if there exists a DFA D with $L(D) = L$. In other words, D accepts exactly the strings in L and rejects all other strings over Σ .

Theorem: Finite languages are regular

Every finite language is regular.

Note that regular languages can also be infinite. For example, the language $L = \{w \in \{0,1\}^* \mid w \text{ starts with a } 0\}$, is regular and infinite.

Theorem: Closure properties (complement, union, intersection)

Regular languages are **closed under** complement, union, and intersection. By which we mean applying these operations to regular languages gives another regular language.

Let L_1 and L_2 be regular languages over the same alphabet Σ . Then the following languages are regular:

The **complement** $\overline{L_1} = \{w \in \Sigma^* \mid w \notin L_1\}$ (contains all strings over Σ that are not in L_1).

The **union** $L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \text{ or } w \in L_2\}$ (contains strings in either language (or both)).

The **intersection** $L_1 \cap L_2 = \{w \in \Sigma^* \mid w \in L_1 \text{ and } w \in L_2\}$ (contains strings that are in both).

Complement construction

If $D = (Q, \Sigma, \delta, q_0, F)$ recognizes L_1 , then $D' = (Q, \Sigma, \delta, q_0, Q \setminus F)$ recognizes $\overline{L_1}$. Here, $Q \setminus F$ denotes the states in Q that are not in F .

Notice that only the accepting states changed, and in particular $Q \setminus F$ is the set of old rejecting states.

Intersection construction

Given DFAs $D_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ for L_1 and $D_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ for L_2 , a DFA for $L_1 \cap L_2$ is $D = (Q_1 \times Q_2, \Sigma, \delta, (q_1, q_2), F_1 \times F_2)$, where:

$Q_1 \times Q_2$ is the set of states. A state (r_1, r_2) records the state r_1 of D_1 and the state r_2 of D_2 .

Σ is the same alphabet as in both original DFAs.

$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$ is the transition function: both DFAs move on the same symbol a .

(q_1, q_2) is the start state: both DFAs begin in their own start states.

$F_1 \times F_2 = \{(r_1, r_2) \mid r_1 \in F_1 \text{ and } r_2 \in F_2\}$ is the set of accepting states: a pair is accepting exactly when both its states are accepting.

Thus D simulates “running both DFAs in parallel” and accepts a string exactly when both accept it.

Union proof

Using DeMorgan’s law we can rewrite the union of two languages using only complements and intersection: $L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$. By the previous two properties we know applying complement and intersection preserve regularity, thus if L_1 and L_2 are regular, so is $\overline{\overline{L_1} \cap \overline{L_2}}$, which equals their union.

Note we could also have given a construction exactly like the one above but with the sole difference that the accepting condition is when either state were an accepting state of their respective DFA. That is, the accepting states are $\{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$ (and the rest of the construction is the same as the intersection one).

Definition: Distinguishability

Let L be a language over Σ .

Two strings $x, y \in \Sigma^*$ are **distinguishable by L** if there exists $z \in \Sigma^*$ such that precisely one of xz and yz is in L .

Such a z is a **distinguishing extension**.

Otherwise, x and y are **indistinguishable by L** , written $x \sim_L y$. This means that for every $z \in \Sigma^*$, either both xz and yz are in L or neither is.

The relation $\sim_L$ is an **equivalence relation**, meaning it is reflexive, symmetric, and transitive.

The **equivalence class** of x is $[x] = \{y \in \Sigma^* \mid x \sim_L y\}$, the set of all strings indistinguishable from x by L . These classes partition Σ^* : every string belongs to exactly one class.

Theorem: Myhill–Nerode

A language L is **regular if and only if** $\sim_L$ has finitely many equivalence classes.

If L is regular, the **number of states in a smallest DFA** recognizing L equals the number of equivalence classes of $\sim_L$.

State lower bounds

Given a set S of strings we say that strings in S are **pairwise distinguishable by L** if for every $x, y \in S$ ($x \neq y$) we have that x and y are distinguishable by L .

If there is a DFA recognizing L , its computation on distinguishable strings must end in **different states**. Thus, if we have a set of k pairwise distinguishable strings, we know that any DFA recognizing L requires **at least k states**.

If we find an **infinite set** of pairwise distinguishable strings, we can conclude that there is no DFA recognizing L , so L is **not regular**.

2. Proof templates

Proving that a language is regular

One approach is to construct a DFA. Another is to express the language using languages already known to be regular and the closure properties above.

Proof Template: Constructing a DFA

1. Define the DFA $D = \underline{\hspace{2cm}}$ by a state diagram or a five-tuple. Specify the start state, accepting states, and all transitions.
2. Explain what each state represents. Use useful state names, and state what must be true of a string w for the computation of D on w to end in each state. Explain why the start state and transitions correctly track this information.
3. As a check on your construction, convince yourself that both directions below hold (if a formal proof is requested, write these out):
 - If $w \in L$, the computation of D on w ends in an accepting state because $\underline{\hspace{2cm}}$.
 - If $w \notin L$, the computation of D on w ends in a rejecting state because $\underline{\hspace{2cm}}$.
4. Together, these two directions show that $L(D) = L$, so L is regular.

What to check for after your DFA proof: Every state needs exactly one transition for each symbol in Σ . Check the empty string and a few short strings, both ones that should be accepted and ones that should be rejected, but remember that testing a few strings does not prove $L(D) = L$ (it is a great way to catch bugs in your DFA though).

Proof Template: Regularity Using Closure Properties

1. Consider the languages $L_1 = \underline{\hspace{2cm}}$, $L_2 = \underline{\hspace{2cm}}$, and any (finitely many) others as needed. Each is regular because $\underline{\hspace{2cm}}$. Cite a result established in class or prove its regularity.
2. Express L as $\underline{\hspace{2cm}}$, using only complement, union, and intersection on those regular languages. Explain why this expression gives exactly L .
3. Since regular languages are closed under complement, union, and intersection, the language given by this expression is regular. Therefore L is regular. $\square$

What to check for after your closure proof: State why the languages you start with are regular. All languages in the expression must be over the same alphabet. Make sure you are using the closure properties in the correct direction: we only proved that starting from regular languages and applying intersection/union/complement, we get a regular language.

Proving that a language is not regular

To prove nonregularity, use Myhill–Nerode or derive a contradiction using closure properties. Saying that you cannot find a DFA, or that a DFA “would need to remember too much,” is not a proof (though it may be a good intuitive starting point).

Proof Template: Using the Myhill–Nerode Theorem

1. Consider the set of strings $S = ______ \subseteq \Sigma^*$. Show that S contains infinitely many distinct strings.
2. Let x and y be **any two distinct** strings from S . Choose $z = ______ \in \Sigma^*$ and show that

$$xz \in L \text{ because } ______, \quad yz \notin L \text{ because } ______,$$
or vice versa. Thus z is a distinguishing extension of x and y .
3. Since every pair of distinct strings in S is distinguishable by L , each string in S belongs to a different equivalence class under $\sim_L$. Since S is infinite, there are infinitely many equivalence classes. By the Myhill–Nerode theorem, L is not regular. $\square$

What to check for after your Myhill–Nerode proof: The z you append to x and y has to be the same z , though the choice of it can depend on the pair x, y . Your proof of distinguishability needs to cover any two distinct strings in S .

Proof Template: Nonregularity Using Closure Properties

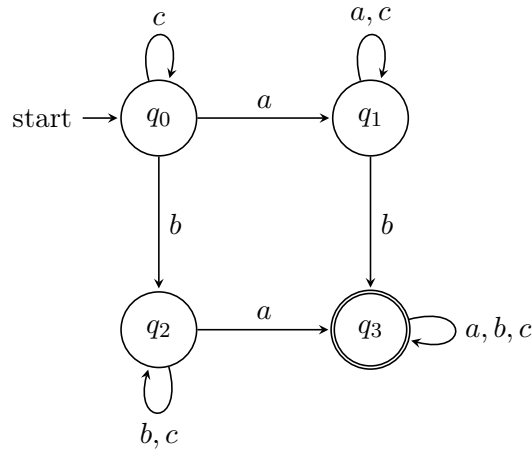
1. Assume for the sake of contradiction that L is regular.
2. Consider the regular languages $L_1 = ______$, $L_2 = ______$, and any (finitely many) others as needed. We know these languages are regular because $______$.
3. Let $L' = ______$ be a language already known to be not regular. Show that L' equals the expression $______$, which applies complement, union, and intersection to L and the other regular languages above. Under our assumption, L' must be regular by these closure properties.
4. But we already proved that L' is not regular (cite that result or earlier proof), which is a contradiction. Thus L is not regular. $\square$

What to check for after your nonregularity-by-closure proof: The nonregularity of L' must be established independently of the claim that L is not regular. Explain the equality in Step 3. The contradiction is that the same language L' would be both regular and not regular.

3. Practice problems

Problem 1

Let D be the following DFA over the alphabet $\Sigma = \{a, b, c\}$.



- Write out the formal definition of D (as a 5-tuple). Describe the transition function in a table.
- What is the language recognized by D ? Give your answer using set notation (such as $L(D) = \{x \in \{a, b\}^* \mid x \text{ starts and ends with an } a\}$).

Problem 2

Give a DFA D recognizing the language

$$L = \{w \in \{0, 1\}^* \mid \text{the number of 1's in } w \text{ is divisible by 3}\}$$

over $\Sigma = \{0, 1\}$. For instance, 000 has zero 1's and is in the language, and 111011 has five 1's and is not in the language, while 111 has three 1's and is in the language.

Draw the state diagram. Give your states useful names, and for each state, write down what must be true of a string for its computation to end in that state.

Problem 3

Draw a state diagram for a DFA that recognizes the following language.

$$L = \{w \in \{a, b\}^* \mid w \text{ has even length and an odd number of } a\text{'s}\}.$$

Problem 4

Prove that the following languages are not regular.

- (a) $L = \{w \in \{a, b\}^* \mid w \text{ is a palindrome, } |w| > 0\}$
- (b) $L = \{xy \in \{0, 1\}^* \mid |x| = |y|, x \text{ has an even number of 1's, } y \text{ has an even number of 0's}\}$ (Challenge)

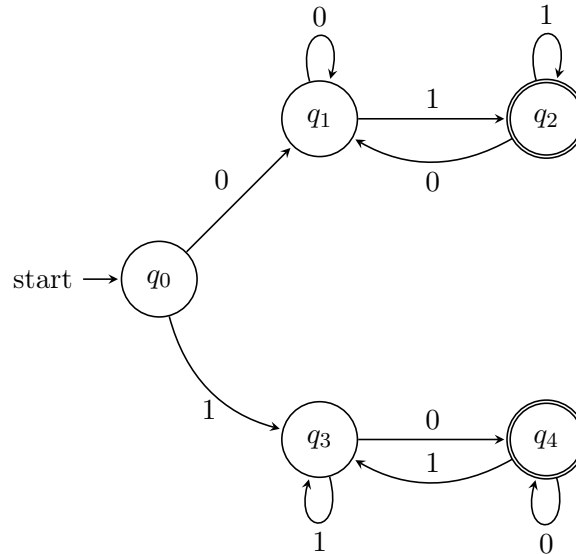
Problem 5

Let $\Sigma = \{a, b\}$. For each part, give an example of two languages $L, L' \subseteq \Sigma^*$ satisfying the conditions below. Don't forget to prove your claims.

- (a) L is regular, L' is not regular, but $L \cup L'$ is regular.
- (b) L is regular, L' is not regular, and $L \cup L'$ is not regular.

Extra Practice Problem 1

Let D' be the following DFA over the alphabet $\Sigma = \{0, 1\}$.



What is the language recognized by D' ? Give your answer in set notation, then a short English description explaining your answer.

Extra Practice Problem 2

Give a DFA D recognizing the language

$$L = \{w \in \{0, 1, \dots, 9\}^+ \mid w \text{ is the decimal representation of a number divisible by 3}\}$$

(we allow leading zeros; for instance 012 represents the number 12; note w must be nonempty). For instance, 12 and 102 are in the language, but 13 is not.

Draw the state diagram. Give your states useful names, and for each state, write down what must be true of a string for its computation to end in that state.