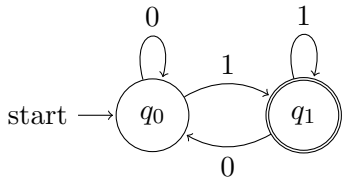
1 DFA Pictionary Challenge

Instruction: Each team selects **three** members as *constructors* to draw DFAs, while the rest are *decoders* who guess the language (either in English or regular expressions) from the state diagram. Each round lasts **3 minutes**. Wrong guesses allow continued drawing and guessing.

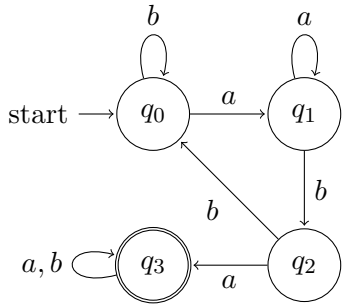
Round 1

The language given to the constructors is: $L = \{b \in \{0,1\}^* : b \text{ ends with a } 1.\}$. The corresponding regular expression is $(0|1)^*1$. One correct DFA that decides the language is



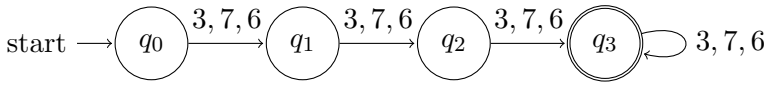
Round 2

The language given to the constructors is $L = \{s \in \{a,b\}^* : s \text{ contains substring "}aba\text{"}\}$. The corresponding regular expression is $(a|b)^*aba(a|b)^*$. One correct DFA that decides the language is



Round 3

The language given to the constructors is $L = \{s \in \{3,7,6\}^* : s \text{ is a string of length } 3\}$. The corresponding regular expression is $(3|7|6)(3|7|6)(3|7|6)$. One correct DFA that decides the language is



2 DFAs and Regular Operations (Optional, but recommended)

In this section, we will highlight some connection between DFA and regular operations on languages. This section may be insightful to construct a "composite" DFA from multiple simpler DFAs that decide some simpler languages.

Definition 1. A language is called a *regular language* if some finite automaton recognizes[1] it.

Definition 2. Let A and B as languages. We define the regular operation *concatenation*, *star*, *union*, and *intersection* as follows:

- Concatenation: $A \circ B = \{xy : x \in A \text{ and } y \in B\}$.
- Star: $A^* = \{x_1 x_2 \cdots x_k : k \geq 0 \text{ and each } x_i \in A\}$.
- Union: $A \cup B = \{x : x \in A \text{ or } x \in B\}$.
- Intersection: $A \cap B = \{x : x \in A \text{ and } x \in B\}$

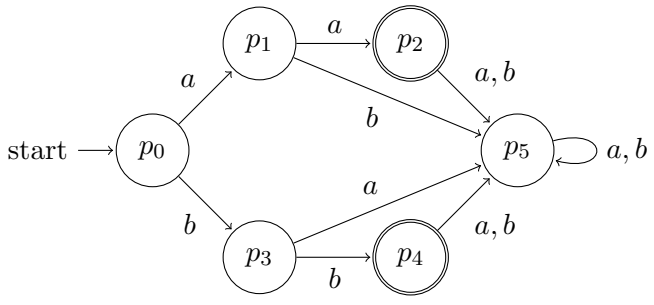
You may start to see the connection between the above operations and some notations we used for regular expressions, such as the bar (|) for "or" and the star (*) for "zero or more". In fact, a language regular if and only if some regular expression describes it[2]. Here, we highlight an important theorem about regular languages.

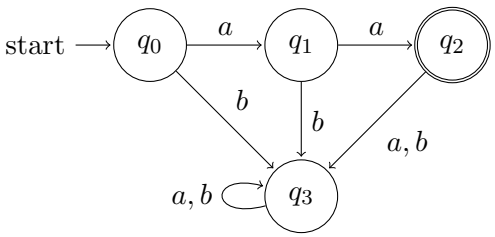
Theorem 3. *The class of regular languages is closed under concatenation, star, union, and intersection.*

For detailed proofs, refer to Section 1.1 in Sipser's *Introduction to the Theory of Computation*. The *product construction method* presented in Section 2.3 and Section 2.4 is both inspired by and justified as a corollary to the proofs.

2.1 Concatenation

In this section, we will discuss how to construct a DFA for a language that is a concatenation of two simpler languages. For example, consider the language over alphabet $\Sigma = \{a, b\}$ described by the regular expression $(aa|bb)aa$. Notice that this language is $A \circ B$, where A is the language described by $(aa|bb)$ while B is the language described by aa. Consider the following two DFAs D_A and D_B that decides A and B respectively:

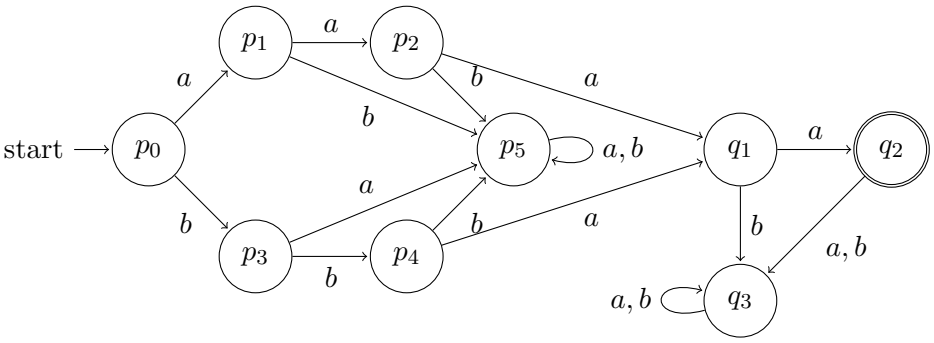




[1] For DFAs, recognizing a language is equivalent to deciding a language.

[2] The full proof of this theorem is out of the scope of this class. If you are interested, check out Section 1.3 in Sipser (listed as an additional resource in the syllabus, contact Eric if you need access to (part of) the book.). Warning: It involves concepts that are *far* beyond the scope of this class, such as nondeterministic finite automata (NFA).

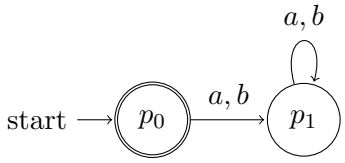
We will now construct a DFA for $(aa|bb)aa$. The idea is as follows: First, run D_A to check the first two characters of the input string. If the first two characters match $(aa|bb)$, then run D_B on the remaining string. This means the accepting states of D_A will merge with the start state of D_B, which gives us the following DFA:



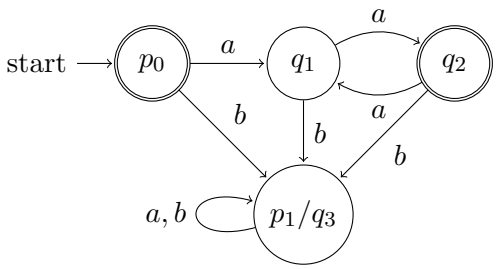
Note that we can further simplify the DFA by combining p_5 and q_3 (so get rid of q_3 and have $\delta(q_1, b) = \delta(q_2, a) = \delta(q_2, b) = p_5$) because they are both "sink states".

2.2 Star

Recall that the star operation indicates "zero or more" occurrences of a string. For example, consider the language over alphabet $\Sigma = \{a, b\}$ described by the regular expression $(aa)^*$. First, we handle the "zero" part, i.e., the language $\{\varepsilon\}$. All we need is a DFA that accepts an empty string in the diagram below:



Next, we handle the "more" part. We have already seen a DFA for aa in Section 2.1. One might be tempted to concatenate infinitely many copies of it, but that approach violates the definition of DFAs having *finitely* many states. We can instead introduce a cycle in the DFA that allows it to recognize repeated occurrences of aa. This gives us the following DFA:



Note that we also merged the two sink states p_1 and q_3.

2.3 Union

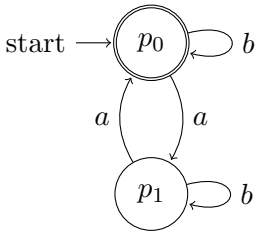
In this section, we will discuss how to construct a DFA for a language that is a union of two simpler languages. For example, consider the following language over alphabet $\Sigma = \{a, b\}$:

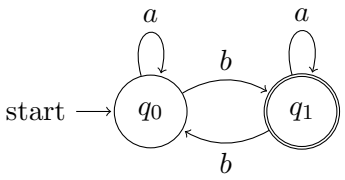
$$L = \{s \in \Sigma^* : s \text{ contains an even number of } a\text{'s } \textbf{or} \text{ an odd number of } b\text{'s}\}.$$

Notice how L is a union of two simpler languages, namely,

$A = \{s \in \Sigma^* : s \text{ contains an even number of } a\text{'s}\}$ and $B = \{s \in \Sigma^* : s \text{ contains an odd number of } b\text{'s}\}$.

which can be decided by the DFAs D_A and D_B below, respectively:





Observe that each state in D_A and D_B correspond to a parity of a's and b's in the string. In particular, p_0 and p_1 corresponds to even and odd number of a, respectively, whereas q_0 and q_1 corresponds to even and odd number of b, respectively. This observation will be useful later.

Unlike in Section 2.1, we can't simulate them sequentially, because we need to check for *both* A and B from the beginning of the input string. That requires "rewinding the input tape", which is not possible with a DFA. Instead, we need to construct a DFA that simulates both smaller DFAs *simultaneously*, which can be achieved by the *product construction method* below.

Theorem 4 (**Product construction method for the union of languages**). *Suppose we have a DFA $D_A = (Q_A, \Sigma, p_0, F_A, \delta_A)$ that decides A and another DFA $D_B = (Q_B, \Sigma, q_0, F_B, \delta_B)$ that decides B[3]. We can construct a DFA D for $L = A \cup B$ as follows:*

1. *The state space for D is the* Cartesian product $Q_A \times Q_B$. *This means each state of D corresponds to a pair (p, q), where p is a state from D_A and q is a state from D_B.*
2. *If in state (p, q) and the input is x, transition to (p', q'), where $p' = \delta_A(p, x)$ is the state reached from p in D_A on input x, and $q' = \delta_B(q, x)$ is the state reached from q in the DFA for B on input x.*
3. *The start state of D is the state corresponds to (p_0, q_0).*
4. *The set of final states of D is $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$. In other words, the states corresponds to (p, q) are accepting states if either p or q is an accepting state in their respective DFAs.*

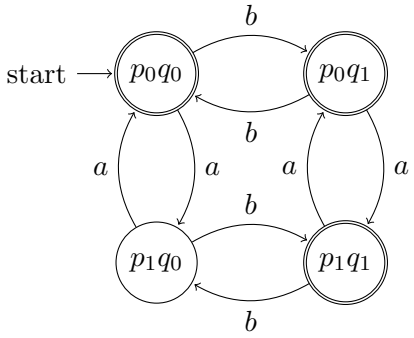
Remark 5. The product construction method *does not* always give the DFA that uses the *least* number of states possible.

We will demonstrate how to apply the method using the example from earlier. Following Item 2, we have the following transition functions:

[3]For simplicity, we assume that A and B are over the same alphabet here, but the method can be extended for languages over different alphabets Σ_A and Σ_B by taking $\Sigma = \Sigma_A \cup \Sigma_B$.

Current	Read a	Read b
(p_0, q_0)	(p_1, q_0)	(p_0, q_1)
(p_0, q_1)	(p_1, q_1)	(p_0, q_0)
(p_1, q_0)	(p_0, q_0)	(p_1, q_1)
(p_1, q_1)	(p_0, q_1)	(p_1, q_0)

The start state of the new DFA is (p_0, q_0), and we have three accepting states: (p_0, q_0), (p_0, q_1), and (p_1, q_1). Putting everything together, we have the following DFA:



Observe that each state corresponding to (p, q) can also be interpreted as the product of what p and q originally represent in their respective DFAs. In particular:

- (p_0, q_0) corresponds to "even a's, even b's"
- (p_0, q_1) corresponds to "even a's, odd b's"
- (p_1, q_0) corresponds to "odd a's, even b's"
- (p_1, q_1) corresponds to "odd a's, odd b's"

With that interpretation, we can quickly verify the correctness of the new DFA: States that correspond to "even a's" or "odd b's" must be accepting, which is true for the DFA we constructed.

2.4 Intersection

In this section, we will discuss how to construct a DFA for a language that is an intersection of two simpler languages.

Theorem 6 (**Product construction method for the intersection of languages**). *Suppose we have a DFA $D_A = (Q_A, \Sigma, p_0, F_A, \delta_A)$ that decides A and another DFA $D_B = (Q_B, \Sigma, q_0, F_B, \delta_B)$ that decides B. We can construct a DFA D for $L = A \cap B$ as follows:*

1. *The state space for D is the* Cartesian product *$Q_A \times Q_B$. This means each state of D corresponds to a pair (p, q), where p is a state from D_A and q is a state from D_B.*
2. *If in state (p, q) and the input is x, transition to (p', q'), where $p' = \delta_A(p, x)$ is the state reached from p in D_A on input x, and $q' = \delta_B(q, x)$ is the state reached from q in the DFA for B on input x.*
3. *The start state of D is the state corresponds to (p_0, q_0).*

4. *The set of final states of D is $F = F_1 \times F_2$. In other words, the states corresponds to (p, q) are accepting states if both p and q is an accepting state in their respective DFAs.*

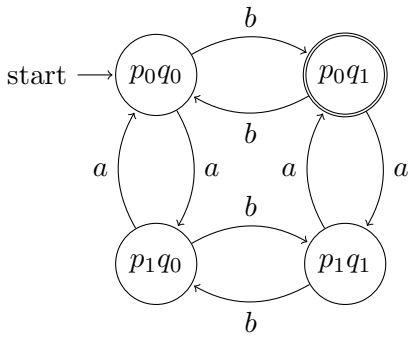
Notice that this is identical to Theorem 4 except in Item 4 we have $F = F_1 \times F_2$ because we need the string to be accepted by *both* D_A and D_B. To illustrate, we will consider a language over $\Sigma = \{a, b\}$ similar to the one we saw in Section 2.3:

$$L = \{s \in \Sigma^* : s \text{ contains an even number of } a\text{'s } \textbf{and} \text{ an odd number of } b\text{'s}\}.$$

Now, L is the intersection of the two languages

$A = \{s \in \Sigma^* : s \text{ contains an even number of } a\text{'s}\}$ and $B = \{s \in \Sigma^* : s \text{ contains an odd number of } b\text{'s}\}$.

Item 1 to Item 3 are identical to what we did earlier, so we only need have one accepting state (p_0, q_1), which results in the following DFA:



Exercises

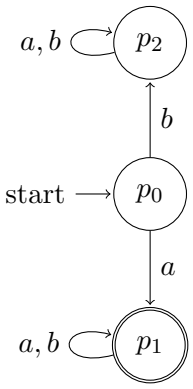
Construct a DFA for each of the following language over alphabet $\Sigma = \{a, b\}$. The solution can be found on the next page.

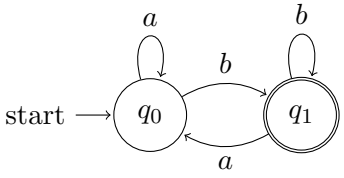
1. $L = \{s \in \Sigma^* : s \text{ starts with an } a \textbf{ or } \text{ends with a } b.\}$
2. $L = \{s \in \Sigma^* : s \text{ has an even number of } a\text{'s } \textbf{and} \text{ each } a \text{ is followed by at least one } b.\}$

Solutions to Exercises

1. $L = \{s \in \Sigma^* : s \text{ starts with an } a \textbf{ or } \text{ends with a } b.\}$

Solution: First, notice that $L = A \cup B$ where $A = \{s \in \Sigma^* : s \text{ starts with an } a\}$ and $B = \{s \in \Sigma^* : s \text{ ends with a } b\}$, which can be decided by the two DFAs below, respectively:

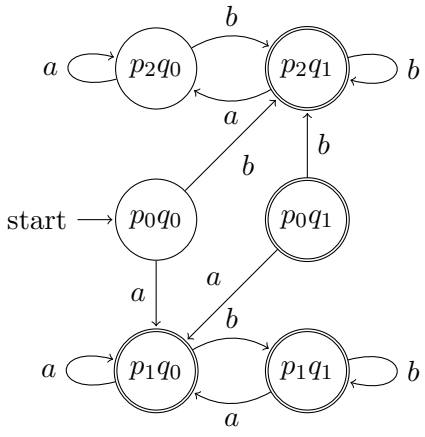




By product construction method, we have the following transition functions:

Current	Read a	Read b
(p_0, q_0)	(p_1, q_0)	(p_2, q_1)
(p_0, q_1)	(p_1, q_0)	(p_2, q_1)
(p_1, q_0)	(p_1, q_0)	(p_1, q_1)
(p_1, q_1)	(p_1, q_0)	(p_1, q_1)
(p_2, q_0)	(p_2, q_0)	(p_2, q_1)
(p_2, q_1)	(p_2, q_0)	(p_2, q_1)

The start state of the new DFA is (p_0, q_0). Since $L = A \cup B$, any state that involve p_1 or q_1 will be an accepting state. Putting everything together, we have the following DFA:

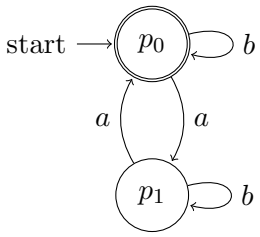


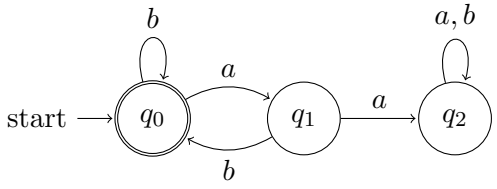
We can verify the correctness of the DFA by inspection. First, if a string starts with an a, then it immediately enters an accepting state p_1q_0, and the only state reachable state

from there is p_1q_1, which is also an accepting state. Second, observe that in the DFA, every state transitions to an accepting state on input b. This ensures that a string ends with a b, the DFA will accept it.

2. $L = \{s \in \Sigma^* : s$ has an even number of a's **and** each a is followed by at least one $b.\}$

Solution: First, notice that $L = A \cap B$, where $A = \{s \in \Sigma^* : s$ has an even number of a's$\}$ and $B = \{s \in \Sigma^* :$ each a in s is followed by at least one $b\}$, which can be decided by the two DFAs below, respectively:

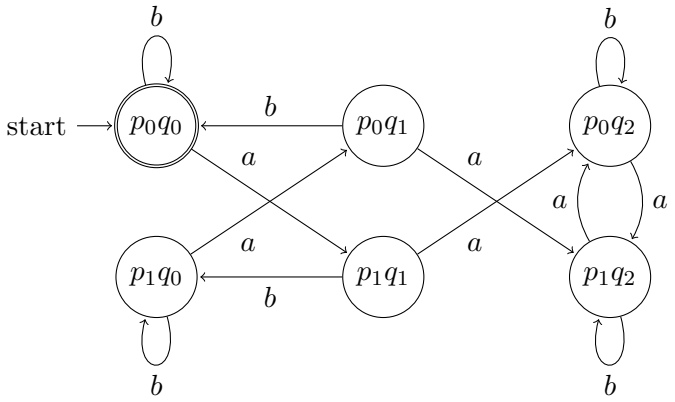




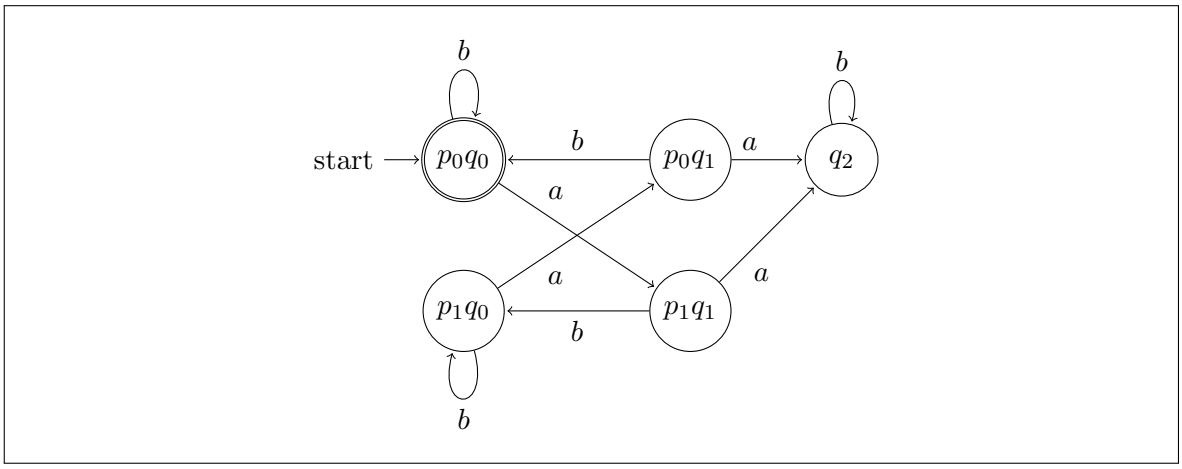
By product construction method, we have the following transition functions:

Current	Read a	Read b
(p_0, q_0)	(p_1, q_1)	(p_0, q_0)
(p_0, q_1)	(p_1, q_2)	(p_0, q_0)
(p_0, q_2)	(p_1, q_2)	(p_0, q_2)
(p_1, q_0)	(p_0, q_1)	(p_1, q_0)
(p_1, q_1)	(p_0, q_2)	(p_1, q_0)
(p_1, q_2)	(p_0, q_2)	(p_1, q_2)

The start state of the new DFA is (p_0, q_0). Since $L = A \cap B$, only states that involve both p_0 and q_0 are accepting states. Putting everything together, we have the following DFA:



We could further simplify the DFA by combining p_0q_2 and p_1q_2:


start
p_0q_0
p_0q_1
q_2
p_1q_0
p_1q_1
b
b
b
a
a
a
a
b
b