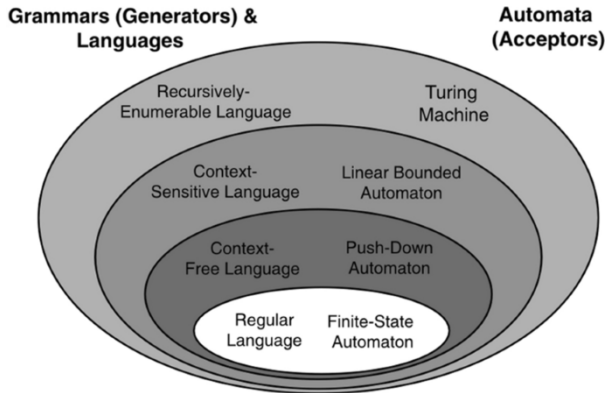


# Finite-State Acceptors



**The Formal Language Hierarchy**

Textbook: 1.1 and 1.2

# Definitions

- ▶ An **automaton** (plural automata) is a self-moving machine. In this context, it just means “small, simple, simulated machine”.
- ▶ An **acceptor** is an automaton which takes in a string and either accepts or rejects it through a series of state transitions.  
**Note:** Many texts fail to distinguish between *automata* and *acceptors*. Acceptors are a proper subset of automata.
- ▶ A **finite-state** automaton has a finite number of states that it can transition through.
- ▶ A **deterministic** automaton has exactly one set of state transitions which follow from a given string. A nondeterministic one, on the other hand, has multiple series of state transitions which may follow.

# Kleene operators

- ▶ For an alphabet  $\Sigma$ , the **Kleene star** is an operator creating the set of all strings of zero or more characters
- ▶ If  $\Sigma^i$  is the set of all strings of length  $i$  over  $\Sigma$ , then  
$$\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$$

- ▶ Ex:

$$\{0, 1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$$

- ▶ The **Kleene plus** gives the set of all strings of *one* or more characters over an alphabet. It is less common

$$\Sigma^* = \Sigma^+ \cup \{\epsilon\}$$

# DFAs

- ▶ A **deterministic finite-state acceptor (DFA)** is an automaton built to accept strings using a finite number of states where each string corresponds to exactly one series of state transitions. Formally:

**Def:** Finite-state acceptor. A finite acceptor is a 5-tuple  $A = (Q, \Sigma, \delta, q_0, F)$  where:

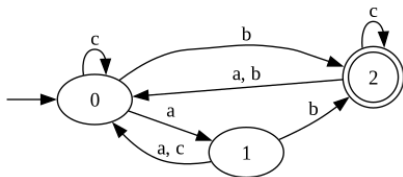
- 1)  $Q$  is a finite set called the **states**
- 2)  $\Sigma$  is a finite set of symbols called the **alphabet**
- 3)  $\delta : (Q \times \Sigma) \rightarrow Q$  is the **transition function**
- 4)  $q_0 \in Q$  is the **start state**
- 5)  $F \subseteq Q$  is the set of acceptance states

**Note:** A DFA must define all transitions! Every state must have a specified target state for every character in the alphabet.

## DFA operation

Consider the DFA  $A$  with states  $Q = \{q_0, q_1, q_2\}$ , starting state  $q_0$ , alphabet  $\Sigma = \{a, b, c\}$ , and acceptance state set  $F = \{q_2\}$ . The following graph demonstrates  $\delta$ .

**Note:** An arrow from nothing means “initial state”. In these diagrams, a double circle means “accepting state”.



This will accept any string of the forms  $abc^i$  or  $bc^i$ , where  $i$  is any nonnegative integer.

## Formal definition of computation

We say that a DFA  $A$  **accepts** string  $w = w_1 w_2 \dots w_n \in \Sigma^*$  if and only if there exists some series of states  $r_0 r_1 r_2 \dots r_n \in Q^+$  such that the following all hold:

- 1)  $r_0 = q_0$
- 2)  $\delta(r_i, w_{i+1}) = r_{i+1}$  for all  $i = 0, \dots, n_1$
- 3)  $r_n \in F$

**Def:** The “language of” operator. For an automaton  $A$  over alphabet  $\Sigma$ , the language of  $A$  is given by the operator  $\mathcal{L}(A) = \{w \in \Sigma^* : A \text{ accepts } w\}$ .

Ex: If  $A$  recognizes all nonnegative even integers, then  $\mathcal{L}(A) = \{0, 2, 4, 6, 8, 10, \dots\}$

# NFAs

- ▶ A **nondeterministic finite-state acceptor (NFA)** is a DFA that can follow multiple series of state transitions
- ▶ The NFA accepts iff any of its branches do
- ▶ The only formal difference is in the transition function:  $\delta$  **can output any number of output states**

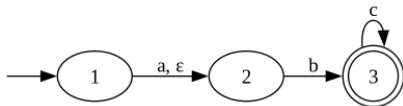
**Def:** Nondeterministic finite-state acceptor (NFA). An NFA is a 5-tuple  $A = (Q, \Sigma, \delta, q_0, F)$  where:

- 1)  $Q$  is a finite set called the states
- 2)  $\Sigma$  is a finite set of symbols called the alphabet
- 3)  $\delta : (Q \times \{\Sigma \cup \epsilon\}) \rightarrow \mathcal{P}(Q)$  is the **nondeterministic transition function**
- 4)  $q_0 \in Q$  is the start state
- 5)  $F \subseteq Q$  is the set of acceptance states

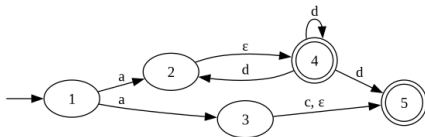
**Note:** If an NFA can leave transitions undefined, unlike a DFA. If it comes to a fork in the road, it duplicates and goes down both paths.

## NFA operation

This NFA accepts the same language as our previous DFA:



This NFA has more “weird” branches:

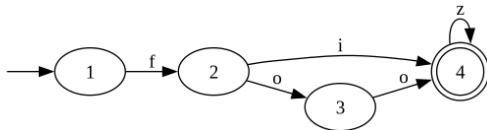




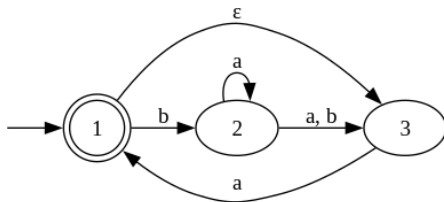
## Practice problems

Do these problems on your own, then we will do them on the board.

- 1) Does the following DFA accept fizz? foo? buzz?



- 2) Describe the set of all strings accepted by the this NFA.



- 3) Construct an NFA **and** a DFA to recognize the set of all strings consisting of repetitions of abc.

## Automaton equivalence

**Def:** Automaton equivalence. Automata  $A$  and  $B$  are equivalent if and only if they recognize the same language: That is,  
 $(A = B) \iff (\mathcal{L}(A) = \mathcal{L}(B)).$

Ex: Let  $A$  be such that

$$\mathcal{L}(A) = \{w : w \text{ is an even nonnegative integer}\}$$

and  $B$  be such that

$$\mathcal{L}(B) = \{w : w \text{ is a non-empty string of numbers ending in } 0,\}$$

Since any nonnegative even integer is a non-empty string of numbers and ends in 0, 2, 4, 6, or 8, all items recognized by  $A$  are recognized by  $B$ . Since all non-empty strings of numbers ending in 0, 2, 4, 6, or 8 are nonnegative even integers, all items recognized by  $B$  are recognized by  $A$ . Therefore  $\mathcal{L}(A) = \mathcal{L}(B)$  and  $A = B$ .

## DFA-NFA equivalence

- ▶ Counterintuitively, **NFA and DFA are equivalent**: Every DFA has an equivalent NFA and vice versa
- ▶ To simulate an NFA, you can keep track of all the states currently held by any branch
- ▶ To advance to the next step, let the new states be the union of all valid transitions from existing states
- ▶ It turns out the set of all possible combinations of NFA states is also a valid DFA state set! We will use this in our proof

## DFA-NFA equivalence proof pt. 1

We will first prove that all DFA are NFA, then prove that all NFA have equivalent DFA.

**Thm:** All DFA have equivalent NFA. **Pf:** The set of all DFA is a subset of the set of all NFA. Therefore, all DFA are trivially NFA. End of proof.

**Thm:** All NFA have equivalent DFA. **Pf:** By construction. DFA have  $\delta_{\text{DFA}} : (Q \times \Sigma) \rightarrow Q$ , while NFA have  $\delta_{\text{NFA}} : (Q \times \{\Sigma \cup \epsilon\}) \rightarrow \mathcal{P}(Q)$ . We will show that all  $\delta_{\text{NFA}}$  have an equivalent modification in the form of  $\delta_{\text{DFA}}$ , proving that all NFA are equivalent to DFA. Let  $A$  be an arbitrary NFA  $= (Q, \Sigma, \delta, q_0, F)$ .

- 1)  $Q$  vs  $\mathcal{P}(Q)$ . Since  $Q$  is a finite set,  $\mathcal{P}(Q)$  is a finite set of size  $2^{|Q|}$ . Any finite set can be used as a set of DFA states, so  $\mathcal{P}(Q)$  works. **Let**  $Q' = \mathcal{P}(Q)$ .

## DFA-NFA equivalence proof pt. 2

- 2)  $\epsilon$  transitions. We must eliminate any nonconsumptive transitions in order to be a valid DFA. Let  $E : Q \rightarrow \mathcal{P}(Q)$  be the  $\epsilon$ -**closure** function mapping a state to the set all all states reachable by  $\epsilon$  transitions from it. Note that  $E : Q \rightarrow Q'$ . **Let**  $\delta'(q, \sigma) = E(\delta_{\text{NFA}}(E(q), \sigma))$ . Note that  $\delta' : (Q' \times \Sigma) \rightarrow Q'$ .
- 3) Starting states. The starting state may have outgoing  $\epsilon$  transitions which must be accounted for. Thus, **let**  $q'_0 = E(q_0)$ .
- 4) Accepting states. An NFA's  $F$  is a subset of the set of its states ( $F \subset Q$ ). However, it is not a subset of  $Q'$  yet: Thus, we must adjust it. **Let**  
 $F' = \{R \in Q' : R \text{ contains an accept state from } F\}$ .

Now, for an arbitrary NFA  $A = (Q, \Sigma, \delta, q_0, F)$ , let  $A' = (Q', \Sigma, \delta', q'_0, F')$ . By the above rules,  $A'$  is a valid DFA equivalent to  $A$ . End of proof.

## Operation closure

**Def:** Closure under an operation. A set is **closed** under an operation iff that operation maps back to the set. We will study closure under unary and binary relations, but the concept generalizes to  $k$ -ary relations.

Ex: Integers. Since, for all integers  $a, b$  the results  $a + b$  and  $ab$  are integers, the set of all integers is closed under addition and multiplication. Since there exist integers  $b, c$  such that  $b/c$  is not an integer, the set of all integers is *not* closed under division.

Ex: Derived closure. Since  $-1$  is an integer, then  $a, b$  being arbitrary integers implies that  $a - b = a + (-1 * b)$  is also an integer. Since addition and multiplication were previously shown to be closed, we can now say subtraction is as well.

# The regular operations

- ▶ Just like in numeric math we have standard operations like  $+$  and  $\times$ , in the study of regular languages we have the following “**regular**” operations

**Def:** The regular operations. The following three operations are known as the regular operations.

- ▶ **Union:**  $A \cup B = \{x : (x \in A) \vee (x \in B)\}$
- ▶ **Concatenation:**  $A \circ B = \{xy : (x \in A) \wedge (y \in B)\}$
- ▶ **Kleene (“KLAY-nee”) Star:**  
 $A^* = \{x_1x_2 \dots x_k : k \geq 0, \forall i[x_i \in A]\}$

**Note:** Kleene plus. If a set  $A$  is closed under concatenation and Kleene star, then  $A^+ = A \circ A^*$  is also closed.

## State machine closure under regular operations

**Thm:** Finite-state-acceptor recognizable languages are closed under the regular operations: Union, concatenation, and Kleene star.

**Pf:** By construction in 3 parts. This can be done using only DFA (as in book), but for simplicity and without loss of generality we will instead use NFA here.

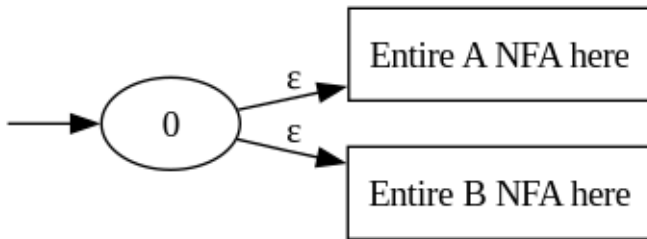
- 1) Union. Given NFA  $A, B$ , we want to construct some NFA representing  $A \cup B$ .
- 2) Concatenation. Given NFA  $A, B$ , we want to construct some NFA representing  $A \circ B$ .
- 3) Kleene star. Given NFA  $A$ , we want to construct some NFA representing  $A^*$ .

This proof will rely on graph representations over formal definitions.



## 1: Closure under union

Let  $Q_A$  and  $Q_B$  be the state sets  $Q$  from  $A$  and  $B$ , respectively. They are relabeled this way to avoid collisions. Let  $a_0$  and  $b_0$  be the initial states of their corresponding NFAs. To create an NFA for  $A \cup B$ , we just need to add a new initial state  $q'_0$  with outgoing  $\epsilon$  transitions to  $a_0$  and  $b_0$ . After this, our new state set  $Q'$ , accepting state set  $F'$ , and  $\delta'$  are just trivial unions of  $A$  and  $B$ 's.

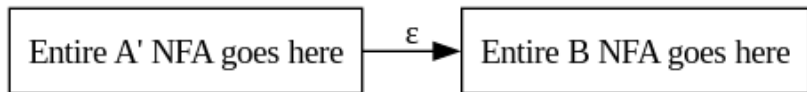


$$\delta'(q_0, \epsilon) = \{a_0, b_0\}$$

This NFA accepts a string matching  $A$  or  $B$ , therefore NFA are

## 2: Closure under concatenation

Let us define  $A'$  as a modification of  $A$  where all accepting states are no longer accepting and have outgoing  $\epsilon$  transitions to the initial state of  $B$ . Let all states of  $A$  and  $B$  be relabelled so they do not conflict. This is represented in the diagram below.

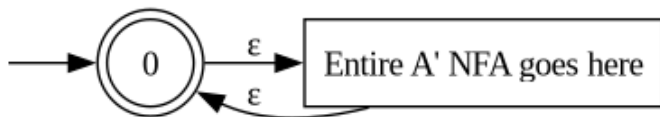


Our new NFA has the initial state of  $A$ , the state transition map of  $A'$  merged with  $B$ , the state set of relabelled states from both, and the final states of  $B$ .

This NFA accepts a string composed of a substring matching  $A$  followed by a substring matching  $B$ , therefore NFA are closed under concatenation.

### 3: Closure under Kleene star

Let us define a new non-conflicting initial state  $q_0$ . All states of  $A$  will be relabelled to avoid a collision with  $q_0$ . Let  $A'$  be the modification of  $A$  such that all accepting states are no longer accepting and have an outgoing  $\epsilon$  transition to  $q_0$ .  $q_0$ , in turn, has an outgoing  $\epsilon$  transition to the old starting node of  $A$ . The below diagram demonstrates this.



This NFA accepts zero or more instances of a substring matching  $A$ , therefore NFA are closed under Kleene star.

End of proof

Since NFA are closed under union, concatenation, and Kleene star, DFA are too. Thus, finite-state machines are closed under the regular operations. End of proof.

## Next up: Regular languages and expressions

“The syntactic component of a grammar must specify, for each sentence, a deep structure that determines its semantic interpretation and a surface structure that determines its phonetic interpretation.” - **Noam Chomsky**

“Give Orange Me Give Eat Orange Me Eat Orange Give Me Eat Orange Give Me You” - **Nim Chimpsky**