Finite Automata
Part Two
Recap from Last Time
DFAs

- A **DFA** is a
  - **D**eterministic
  - **F**inite
  - **A**utomaton
- DFAs are the simplest type of automaton that we will see in this course.
DFAs

• A DFA is defined relative to some alphabet $\Sigma$.

• For each state in the DFA, there must be exactly one transition defined for each symbol in $\Sigma$.
  • This is the “deterministic” part of DFA.

• There is a unique start state.

• There are zero or more accepting states.
If $D$ is a DFA, the **language of $D$**, denoted $\mathcal{L}(D)$, is $\{ w \in \Sigma^* | D \text{ accepts } w \}$.

A language $L$ is called a **regular language** if there exists a DFA $D$ such that $\mathcal{L}(D) = L$. 
NFAs

- An **NFA** is a
  - **N**ondeterministic
  - **F**inite
  - **A**utomaton
- Can have missing transitions or multiple transitions defined on the same input symbol.
- Accepts if *any possible series of choices* leads to an accepting state.
New Stuff!
Intuiting Nondeterminism

Nondeterministic machines are a serious departure from physical computers. How can we build up an intuition for them?

There are two particularly useful frameworks for interpreting nondeterminism:

- Perfect positive guessing
- Massive parallelism
Perfect Positive Guessing

(start) $q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3$
Perfect Positive Guessing

$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3$

$\Sigma$

Start

a b a b a b a
Perfect Positive Guessing

The diagram shows a finite automaton with states $q_0, q_1, q_2,$ and $q_3$. The transitions are labeled with symbols $a$ and $b$. The automaton starts at state $q_0$ and transitions as follows:

- From $q_0$ to $q_1$ on input $a$.
- From $q_1$ to $q_2$ on input $b$.
- From $q_2$ back to itself on input $a$.

The input sequence is $a b a b a b a$. The automaton accepts this sequence.
Perfect Positive Guessing

- Start in state $q_0$
- Transition to $q_1$ on input $a$
- Transition to $q_2$ on input $b$
- Transition to $q_3$ on input $a$

Input sequence: $ababaab$
Perfect Positive Guessing

\[
\begin{align*}
\Sigma & \quad a \quad b \\
q_0 & \quad \rightarrow \quad a \quad b \\
q_1 & \quad \rightarrow \quad b \quad a \\
q_2 & \quad \rightarrow \quad a \quad \rightarrow \quad \text{loop} \\
q_3 & 
\end{align*}
\]
Perfect Positive Guessing

\begin{align*}
q_0 & \xrightarrow{a} q_1 \\
q_1 & \xrightarrow{b} q_2 \\
q_2 & \xrightarrow{a} q_3
\end{align*}

\begin{align*}
\Sigma & = \{a, b\}
\end{align*}
Perfect Positive Guessing

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ \Sigma \]

\[ a \ b \ a \ b \ a \ b \ a \]
Perfect Positive Guessing

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ \Sigma \]

\[ \text{a b a b a b a} \]
Perfect Positive Guessing

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ \Sigma \]

\[ \begin{array}{cccccc}
  a & b & a & b & a & a \\
\end{array} \]
Perfect Positive Guessing

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

Input: \( \Sigma \)
Perfect Positive Guessing

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ \Sigma \]

\[ \begin{array}{cccccc}
    a & b & a & b & b & a \\
\end{array} \]

SEAL
OF APPROVAL
Perfect Positive Guessing

• We can view nondeterministic machines as having *Magic Superpowers* that enable them to guess choices that lead to an accepting state.
  • If there is at least one choice that leads to an accepting state, the machine will guess it.
  • If there are no choices, the machine guesses any one of the wrong guesses.
• There is no known way to physically model this intuition of nondeterminism – this is quite a departure from reality!
Massive Parallelism

\[ \sum \]

\[ a \quad b \quad a \quad b \quad a \quad b \quad a \]

Diagram:

- **Start state:** \( q_0 \)
- **Transitions:** 
  - \( q_0 \) to \( q_1 \) on \( a \)
  - \( q_1 \) to \( q_2 \) on \( b \)
  - \( q_2 \) to \( q_3 \) on \( a \)
  - \( q_3 \) is a loop
Massive Parallelism

$\Sigma$  

$\begin{array}{cccccc}
q_0 & \xrightarrow{a} & q_1 & \xrightarrow{b} & q_2 & \xrightarrow{a} & q_3 \\
\text{start} & \quad & \quad & \quad & \quad & \quad & \quad
\end{array}$

\text{a b a b a a
Massive Parallelism

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ \Sigma \]

\[ a \ b \ a \ b \ a \]

\[ \Rightarrow \]
Massive Parallelism

a b a b a b a
Massive Parallelism

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ \Sigma \]

\[ a \quad b \quad a \quad b \quad a \quad b \quad a \]
Massive Parallelism

\[
\begin{align*}
q_0 & \xrightarrow{a} q_1 \\
q_1 & \xrightarrow{b} q_2 \\
q_2 & \xrightarrow{a} q_3
\end{align*}
\]

Input: a b a b a b a
Massive Parallelism
Massive Parallelism

\[ \Sigma \]

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

Input sequence: a b a b a b a
Massive Parallelism

Σ

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

Input sequence: a b a b a b a
Massive Parallelism

\[
\Sigma
\]

\[q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

Input sequence: a b a b a

Diagram:
- Start state: \(q_0\)
- States: \(q_0, q_1, q_2, q_3\)
- Transitions: \(a \rightarrow q_1, b \rightarrow q_2, a \rightarrow q_3\)
- Final state: \(q_3\)
Massive Parallelism
Massive Parallelism

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

Input sequence: \[ a \ b \ b \ a \ b \ a \ a \]
Massive Parallelism

start → $q_0$ → $q_1$ → $q_2$ → $q_3$

$a$ $b$ $a$ $b$ $a$ $b$ $a$
Massive Parallelism
Massive Parallelism

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ \sum \]

\[ \text{start} \quad q_0 \quad q_1 \quad q_2 \quad q_3 \]

\[ a \quad b \quad a \quad b \quad a \]

\[ \uparrow \]
Massive Parallelism

\[ q_0, q_1, q_2, q_3 \]

\[ \Sigma \]

Start: \[ q_0 \] to \[ q_1 \] on \[ a \] and \[ b \] to \[ q_2 \] on \[ a \] and \[ q_2 \] to \[ q_3 \] on \[ a \]

Input: \[ a b a b a b a \]
Massive Parallelism

\[ \Sigma \]

\[
\begin{array}{c}
\text{start} \\
q_0 \\
q_1 \\
q_2 \\
q_3
\end{array}
\]

\[
\begin{array}{c}
a \\
b \\
a
\end{array}
\]

\[
\begin{array}{c}
a \\
ba \\
ba \\
ba
\end{array}
\]
Massive Parallelism

\[ \sum \]

\text{start} \quad q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3

\begin{array}{cccccc}
\text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{a}
\end{array}
Massive Parallelism

\[ q_3 \rightarrow q_2 \rightarrow q_1 \rightarrow q_0 \rightarrow \text{start} \]

Input sequence: \[ a \ b \ a \ b \ a \ b \ a \]
Massive Parallelism

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

Input string: \[ a\ b\ a\ b\ a\ b\ a \]
Massive Parallelism

\[ \sum \]

\[
\begin{array}{c}
\text{start} \\
q_0 \\
\rightarrow \\
\rightarrow \\
\rightarrow \\
\rightarrow \\
q_3 \\
\end{array}
\]

\[
\begin{array}{c}
a \\
b \\
a \\
a \\
\end{array}
\]

\[
\begin{array}{c}
a \\
b \\
\end{array}
\]
Massive Parallelism

\[ q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \]

\[ \Sigma \rightarrow a \rightarrow b \rightarrow a \]

Start

a b a b a a
Massive Parallelism

\[ \Sigma \]

\[ q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \]

Input sequence:

\[ a \ b \ a \ b \ a \ b \ a \]
Using the massive parallelism intuition, if we are in the states $q_0$ and $q_2$, what set of states will we be in after reading the character $a$?

Respond at pollev.com/cs103
Massive Parallelism

\[ \Sigma \]

\[ q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \]

\[ a, b, a, b, a \]
Massive Parallelism

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ \Sigma \]

Input sequence: \[ a \ b \ a \ b \ b \ a \ a \]
Massive Parallelism

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

Input sequence: \[ a b a b a b a \]
Massive Parallelism
Massive Parallelism

\[ a \quad b \quad a \quad b \quad a \quad a \]
Massive Parallelism

We’re in at least one accepting state, so there’s some path that gets us to an accepting state.
Massive Parallelism

\[
\begin{align*}
q_0 & \xrightarrow{a} q_1 \\
q_1 & \xrightarrow{b} q_2 \\
q_2 & \xrightarrow{a} q_3
\end{align*}
\]
Massive Parallelism

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ \Sigma \]

\[ a \quad b \quad a \quad b \quad a \quad b \]

\[ \uparrow \]
Massive Parallelism

\[ \Sigma \]

\[ \text{start} \quad q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ \begin{array}{cccc} a & b & a & b \end{array} \]
Massive Parallelism

\[ q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \]

\[ \Sigma \]

\[ a \quad b \quad a \quad a \quad b \quad b \]
Massive Parallelism

\[ q_0 \xrightarrow{\alpha} q_1 \xrightarrow{\beta} q_2 \xrightarrow{\alpha} q_3 \]

\[ \Sigma \]

Input:
- a b a a b b
Massive Parallelism

```
a b a b b
```

\[
\begin{align*}
\text{Start} & \rightarrow q_0 & a & \rightarrow q_1 & b & \rightarrow q_2 & a & \rightarrow q_3
\end{align*}
\]

\[
\Sigma
\]

- Transition from \( q_0 \) to \( q_1 \) on input \( a \)
- Transition from \( q_1 \) to \( q_2 \) on input \( b \)
- Transition from \( q_2 \) to \( q_3 \) on input \( a \)
- Transition from \( q_3 \) to \( q_0 \) on any input \( \Sigma \)
Massive Parallelism

\[ q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \]

Input:\[ a \ b \ b \ a \ b \ b \]
Massive Parallelism
Massive Parallelism

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ \Sigma \]

\[ \text{a b a b b} \]
Massive Parallelism

\[
\begin{align*}
q_0 & \xrightarrow{a} q_1 \\
q_1 & \xrightarrow{b} q_2 \\
q_2 & \xrightarrow{a} q_3
\end{align*}
\]

\(\Sigma\)

a b a b b
Massive Parallelism

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ \Sigma \]

\[ a \quad b \quad a \quad b \quad a \quad b \]

Diagram showing transitions between states with inputs a, b.
Massive Parallelism

\( \Sigma \)

- start \( q_0 \) → a \( q_1 \) → b \( q_2 \) → a \( q_3 \)

- Input: a b a b b
Massive Parallelism

\[ \sum \]

\[
\begin{align*}
\text{start} & \quad q_0 & \quad a & \quad q_1 & \quad b & \quad q_2 & \quad a & \quad q_3 \\
\end{align*}
\]

\[
\begin{array}{cccc}
a & b & a & b \\
\end{array}
\]
Massive Parallelism

\[
\Sigma
\]

\[
\begin{align*}
q_0 & \rightarrow q_1 \\
q_1 & \rightarrow q_2 \\
q_2 & \rightarrow q_3
\end{align*}
\]

Transition:
- \( q_0 \) to \( q_1 \) via \( a \)
- \( q_1 \) to \( q_2 \) via \( b \)
- \( q_2 \) to \( q_3 \) via \( a \)

Input:
- \( a \)
- \( b \)
- \( ab \)
- \( ab \)
- \( b \)
Massive Parallelism

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ \Sigma \]

\[
\begin{array}{c}
| & a & b & a & b & b \\
\hline
\end{array}
\]
Massive Parallelism

\[
\Sigma \quad a \quad b
\]

\[
q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3
\]
Massive Parallelism

$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3$

$\Sigma$

start

a b a b b
Massive Parallelism
Massive Parallelism

\[ \sum \]

Start \rightarrow q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3

\begin{array}{cccc}
 q_0 & a & q_1 & b \\
 \downarrow & \downarrow & \downarrow & \downarrow \\
 q_2 & a & q_3 & \end{array}

\begin{array}{cccc}
 a & b & a & b \\
 \end{array}
Massive Parallelism

States: $q_0, q_1, q_2, q_3$

Transitions:
- $q_0 \xrightarrow{a} q_1$
- $q_1 \xrightarrow{b} q_2$
- $q_2 \xrightarrow{a} q_3$

Input alphabet: $\Sigma = \{a, b\}$

Initial state: $q_0$

Final state: $q_3$

Input sequence: $a b a b b$
Massive Parallelism

\( q_3 \)

\( q_2 \)

\( q_1 \)

\( q_0 \)

\( \Sigma \)

start

\( a \)

\( b \)

\( a \)

\( b \)

\( a \)

\( b \)

\( b \)
Massive Parallelism

Diagram:
- **Start state**: $q_0$
- Transition labels: $a$, $b$
- Possible inputs: $\Sigma$
- States: $q_0$, $q_1$, $q_2$, $q_3$
- Edges:
  - $q_0 \xrightarrow{a} q_1$
  - $q_1 \xrightarrow{b} q_2$
  - $q_2 \xrightarrow{a} q_3$
- Terminal state: $q_3$

Sequence: $a \ b \ a \ b$
Massive Parallelism
Massive Parallelism

\begin{align*}
    & \sum \\
    \text{start} & \rightarrow q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3
\end{align*}

\begin{array}{cccc}
    a & b & a & b
\end{array}
Massive Parallelism

\[ a \quad b \quad a \quad a \quad b \]

We're not in any accepting state, so no possible path accepts.
Massive Parallelism

• An NFA can be thought of as a DFA that can be in many states at once.

• At each point in time, when the NFA needs to follow a transition, it tries all the options at the same time.

• (Here's a rigorous explanation about how this works; read this on your own time).

  • Start off in the set of all states formed by taking the start state and including each state that can be reached by zero or more ε-transitions.

  • When you read a symbol a in a set of states S:
    − Form the set S’ of states that can be reached by following a single a transition from some state in S.
    − Your new set of states is the set of states in S’, plus the states reachable from S’ by following zero or more ε-transitions.
Designing NFAs

- *Embrace the nondeterminism!*
- Good model: *Guess-and-check*:
  - Is there some information that you'd really like to have? Have the machine *nondeterministically guess* that information.
  - Then, have the machine *deterministically check* that the choice was correct.
- The *guess* phase corresponds to trying lots of different options.
- The *check* phase corresponds to filtering out bad guesses or wrong options.
Guess-and-Check

$L = \{ \ w \in \{0, 1\}^* \ | \ w \text{ ends in } 010 \text{ or } 101 \ \}$
Guess-and-Check

\[ L = \{ \ w \in \{0, 1\}^* \mid \text{w ends in 010 or 101} \} \]
Guess-and-Check

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
Guess-and-Check

\[ L = \{ \, w \in \{0,1\}^* \mid w \text{ ends in 010 or 101} \, \} \]

Which of these states should we mark as accepting states?

Respond at pollev.com/cs103
Guess-and-Check

\[ L = \{ w \in \{0, 1\}^* \mid \text{w ends in 010 or 101} \} \]

Nondeterministically guess when the end of the string is coming up.

Deterministically check whether you were correct.
Guess-and-Check

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
Guess-and-Check

\[ L = \{ \ w \in \{0, 1\}^* \mid \text{\textit{w} ends in } 010 \text{ or } 101 \ \} \]
Guess-and-Check

\[ L = \{ \ w \in \{0, 1\}^* \mid \text{w ends in 010 or 101} \ \} \]
Guess-and-Check

\[ L = \{ \ w \in \{0, 1\}^* \mid w \text{ ends in 010 or 101} \} \]
Guess-and-Check

\[ L = \{ \ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
Guess-and-Check

\[ L = \{ \ w \in \{0, 1\}^* \mid w \text{ ends in 010 or 101} \} \]
Guess-and-Check

$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$
Guess-and-Check

\[ L = \{ \, w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \, \} \]
Guess-and-Check

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
Guess-and-Check

$L = \{\ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \ \}$
$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$$
Guess-and-Check

$L = \{ \ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \ \}$
Guess-and-Check

\[ L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]
Guess-and-Check

\[ L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]

Nondeterministically guess which character is missing.

Deterministically check whether that character is indeed missing.
Guess-and-Check

\[ L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]
Guess-and-Check

\[ L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]
Guess-and-Check

\[ L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]
Guess-and-Check

\[ L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]
Guess-and-Check

\[ L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]
Guess-and-Check

\[ L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]
**Guess-and-Check**

\[ L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]
Guess-and-Check

$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$
Time-Out For Announcements!
Midterm Exam on Friday!

- Our midterm exam will be on Friday, July 28th from 4:30 – 7:30 PM in Shriram 104 (our normal lecture room).

- You’re responsible for lectures up to the end of week 3 and topics from PS1 – PS3. Later lectures and problem sets won’t be tested here. Exam problems may build on the written or coding components from the problem sets.

- The exam is open-book, open-note, and closed-other-humans/AI.
Back to CS103!
Just how powerful are NFAs?
NFAs and DFAs

- Any language that can be accepted by a DFA can be accepted by an NFA.
- Why?
  - Every DFA essentially already is an NFA!
- **Question:** Can any language accepted by an NFA also be accepted by a DFA?
- Surprisingly, the answer is **yes**!
Thought Experiment:
How would you simulate a finite automata in software?
Tabular DFAs

The diagram shows a deterministic finite automaton (DFA) with states $q_0, q_1, q_2, q_3$. The transitions are labeled with inputs 0 and 1, leading to respective states $q_1, q_2, q_3$ for input 0 and $q_0, q_1, q_2, q_3$ for input 1.
Tabular DFAs

Start state: $q_0$

Transition table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_1$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_3$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_3$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_3$</td>
<td>$q_3$</td>
</tr>
</tbody>
</table>
Tabular DFAs

![Diagram of a DFA with states and transitions]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>*q₀</td>
<td>q₁</td>
<td>q₀</td>
</tr>
<tr>
<td>q₁</td>
<td>q₃</td>
<td>q₂</td>
</tr>
<tr>
<td>q₂</td>
<td>q₃</td>
<td>q₀</td>
</tr>
<tr>
<td>*q₃</td>
<td>q₃</td>
<td>q₃</td>
</tr>
</tbody>
</table>
Tabular DFAs

These stars indicate accepting states.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>*q₀</td>
<td>q₁</td>
<td>q₀</td>
</tr>
<tr>
<td>q₁</td>
<td>q₃</td>
<td>q₂</td>
</tr>
<tr>
<td>q₂</td>
<td>q₃</td>
<td>q₀</td>
</tr>
<tr>
<td>*q₃</td>
<td>q₃</td>
<td>q₃</td>
</tr>
</tbody>
</table>
Since this is the first row, it's the start state.
Question to ponder: Why isn’t there a column here for $\Sigma$?
int kTransitionTable[kNumStates][kNumSymbols] = {
    {0, 0, 1, 3, 7, 1, ...},
    ...
};

bool kAcceptTable[kNumStates] = {
    false,
    true,
    true,
    ...
};

bool SimulateDFA(string input) {
    int state = 0;
    for (char ch: input) {
        state = kTransitionTable[state][ch];
    }
    return kAcceptTable[state];
}
Can we do something similar for NFAs?
\( q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \)

\[ \Sigma \]

Start state: \( q_0 \)

Transition:
- \( a \) from \( q_0 \) to \( q_1 \)
- \( b \) from \( q_1 \) to \( q_2 \)
- \( a \) from \( q_2 \) to \( q_3 \)

Transition from \( q_3 \) to \( q_0 \) is labeled with \( \Sigma \)
The diagram illustrates a finite automaton with states $q_0$, $q_1$, $q_2$, and $q_3$. The transitions are labeled with symbols $a$ and $b$. The start state is $q_0$, and there is a transition on symbol $\Sigma$ leading to $q_0$. The sequence $a b a b a a$ is shown below the automaton.
The figure shows a finite automaton with states $q_0, q_1, q_2,$ and $q_3$. The transitions are as follows:

- From $q_0$ on input $a$, move to $q_1$.
- From $q_1$ on input $b$, move to $q_2$.
- From $q_2$ on input $a$, move to $q_3$.
- From $q_3$, there is a loop on any input.

The table below represents the transitions:

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td></td>
</tr>
<tr>
<td>$q_0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_3$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The given image depicts a transition diagram of a finite state machine (FSM). The diagram consists of a start state labeled with $q_0$ and transitions labeled with $a$ and $b$.

The transitions are as follows:
- From $q_0$, on input $a$, move to $q_1$.
- From $q_1$, on input $b$, move to $q_2$.
- From $q_2$, on input $a$, move to $q_3$.
- From $q_3$, on input $a$, return to $q_0$.

The diagram also includes a transition labeled $\Sigma$ from $q_0$ to $q_3$.

The table below the diagram shows the state transitions for inputs $a$ and $b$:

<table>
<thead>
<tr>
<th>State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td></td>
</tr>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td></td>
</tr>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td></td>
</tr>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td></td>
</tr>
</tbody>
</table>
\[ \begin{array}{|c|c|c|}
\hline
\text{state} & \text{a} & \text{b} \\
\hline
\{q_0\} & \{q_0, q_1\} & \text{blank} \\
\text{blank} & \text{blank} & \text{blank} \\
\text{blank} & \text{blank} & \text{blank} \\
\text{blank} & \text{blank} & \text{blank} \\
\hline
\end{array} \]
<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_0}</td>
<td>{q_0, q_1}</td>
<td>{q_0}</td>
</tr>
</tbody>
</table>

Diagram:

- Start state: \(q_0\)
- Transitions:
  - From \(q_0\) on \(a\) to \(q_1\)
  - From \(q_1\) on \(b\) to \(q_2\)
  - From \(q_2\) on \(a\) to \(q_3\)
  - \(q_3\) is a final state
<table>
<thead>
<tr>
<th>State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>$q_1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_3$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:
- Start state $q_0$
- Transitions:
  - $a$ from $q_0$ to $q_1$
  - $b$ from $q_1$ to $q_2$
  - $a$ from $q_2$ to $q_3$
- Final state $q_3$

Input alphabet $\Sigma$
A finite automaton with the following states and transitions:

- **Start state:** $q_0$
- **Final state:** $q_3$
- **Transitions:**
  - From $q_0$ to $q_1$ on input $a$
  - From $q_1$ to $q_2$ on input $b$
  - From $q_2$ to $q_3$ on input $a$

The table below shows the transitions for inputs $a$ and $b$:

<table>
<thead>
<tr>
<th>Current State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${ q_0 }$</td>
<td>${ q_0, q_1 }$</td>
<td>${ q_0 }$</td>
</tr>
<tr>
<td>${ q_0, q_1 }$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Transition Table

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_0}</td>
<td>{q_0, q_1}</td>
<td>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_1}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_2}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_3}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Diagram

- **Start State:** \(q_0\)
- **Final State:** \(q_3\)
- **Transitions:**
  - \(q_0\) on \(a\) to \(q_1\)
  - \(q_1\) on \(b\) to \(q_2\)
  - \(q_2\) on \(a\) to \(q_3\)
  - \(q_3\) on \(\Sigma\) (ε transition) to \(q_3\)
\begin{array}{c|c|c}
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & & \\
& & \\
\end{array}

Transition diagram:
- Start state: $q_0$
- Transitions:
  - $q_0 \xrightarrow{a} q_1$
  - $q_1 \xrightarrow{b} q_2$
  - $q_2 \xrightarrow{a} q_3$

Accepting states:
- $q_3$

Input alphabet: $\Sigma$

Start state: $q_0$

Transitions:
- $q_0 \xrightarrow{\Sigma} q_0$
- $q_0 \xrightarrow{a} q_1$
- $q_1 \xrightarrow{b} q_2$
- $q_2 \xrightarrow{a} q_3$
- $q_3$ is a final state.
<table>
<thead>
<tr>
<th>State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\begin{align*}
\begin{array}{c|c|c}
 & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \\
\{q_0, q_1\} & & \\
\end{array}
\end{align*}
The given deterministic finite automaton (DFA) consists of states $q_0$, $q_1$, $q_2$, and $q_3$. The transitions are as follows:

- From $q_0$, on reading $a$ it transitions to $q_1$.
- From $q_1$, on reading $b$ it transitions to $q_2$.
- From $q_2$, on reading $a$ it transitions back to $q_3$.

The initial state is $q_0$, marked as the start state. The transitions are shown with arrows labeled by the input symbols $a$ and $b$.

The corresponding transition table is:

<table>
<thead>
<tr>
<th>Current State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0, q_2}$</td>
</tr>
</tbody>
</table>

The input symbols $a$ and $b$ are shown above the transitions, indicating the actions taken upon reading each symbol from the input.

Additionally, the state $q_3$ is marked as an accept state, denoted by the double circle.
The given automaton is a Deterministic Finite Automaton (DFA) with the following states and transitions:

- **States:** $q_0, q_1, q_2, q_3$
- **Start State:** $q_0$
- **Final State:** $q_3$
- **Alphabet:** $\Sigma = \{a, b\}$

### Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0, q_2}$</td>
</tr>
</tbody>
</table>
\[
\begin{align*}
\Sigma \\
\text{start} \\
q_0 \\
a \rightarrow q_1 \\
b \rightarrow q_2 \\
a \rightarrow q_3
\end{align*}
\]

<table>
<thead>
<tr>
<th>States</th>
<th>\text{a}</th>
<th>\text{b}</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ q_0 }</td>
<td>{ q_0, q_1 }</td>
<td>{ q_0 }</td>
</tr>
<tr>
<td>{ q_0, q_1 }</td>
<td>{ q_0, q_1 }</td>
<td>{ q_0, q_2 }</td>
</tr>
<tr>
<td>{ q_0, q_2 }</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[
\begin{array}{c|c|c}
\text{a} & \text{b} \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & & \\
\end{array}
\]
\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ \Sigma \]

<table>
<thead>
<tr>
<th>State</th>
<th>( a )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( { q_0 } )</td>
<td>( { q_0, q_1 } )</td>
<td>( { q_0 } )</td>
</tr>
<tr>
<td>( { q_0, q_1 } )</td>
<td>( { q_0, q_1 } )</td>
<td>( { q_0, q_2 } )</td>
</tr>
<tr>
<td>( { q_0, q_2 } )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\begin{array}{|c|c|c|}
\hline
& a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \\
\hline
\end{array}
\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

<table>
<thead>
<tr>
<th>State</th>
<th>Transition</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_0}</td>
<td></td>
<td>{q_0, q_1}</td>
<td>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td></td>
<td>{q_0, q_1}</td>
<td>{q_0, q_2}</td>
</tr>
<tr>
<td>{q_0, q_2}</td>
<td></td>
<td>{q_0, q_1, q_3}</td>
<td></td>
</tr>
<tr>
<td>State Set</td>
<td>a</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------</td>
<td>-----------------</td>
<td></td>
</tr>
<tr>
<td>{q_0}</td>
<td>{q_0, q_1}</td>
<td>{q_0}</td>
<td></td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td>{q_0, q_1}</td>
<td>{q_0, q_2}</td>
<td></td>
</tr>
<tr>
<td>{q_0, q_2}</td>
<td>{q_0, q_1, q_3}\</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[\sum \]

\[
\begin{array}{ccc}
\text{start} & \rightarrow & q_0 \\
q_0 & \rightarrow & a \rightarrow q_1 \\
q_1 & \rightarrow & b \rightarrow q_2 \\
q_2 & \rightarrow & a \rightarrow q_3 \\
q_3 & \rightarrow & \text{start}
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\{ q_0 \} & \{ q_0, q_1 \} & \{ q_0 \} \\
\{ q_0, q_1 \} & \{ q_0, q_1 \} & \{ q_0, q_2 \} \\
\{ q_0, q_2 \} & \{ q_0, q_1, q_3 \} & \text{---} \\
\hline
\end{array}
\]
<table>
<thead>
<tr>
<th>State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0, q_2}$</td>
</tr>
<tr>
<td>${q_0, q_2}$</td>
<td>${q_0, q_1, q_3}$</td>
<td>-</td>
</tr>
</tbody>
</table>
\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

**Transition Table**

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_0}</td>
<td>{q_0, q_1}</td>
<td>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td>{q_0, q_1}</td>
<td>{q_0, q_2}</td>
</tr>
<tr>
<td>{q_0, q_2}</td>
<td>{q_0, q_1, q_3}</td>
<td>-</td>
</tr>
</tbody>
</table>
\[
\begin{align*}
\text{start} & \quad \xrightarrow{\Sigma} \quad q_0 \\
q_0 & \quad \xrightarrow{a} \quad q_1 \\
q_1 & \quad \xrightarrow{b} \quad q_2 \\
q_2 & \quad \xrightarrow{a} \quad q_3
\end{align*}
\]

| \{q_0\}  | \{q_0, q_1\} | \{q_0\}  \\
| \{q_0, q_1\} | \{q_0, q_1\} | \{q_0, q_2\}  \\
| \{q_0, q_2\} | \{q_0, q_1, q_3\} | \{q_0\}  \\

| a | b |
|---|---|---|
| \{q_0\} | \{q_0, q_1\} | \{q_0\}  \\
| \{q_0, q_1\} | \{q_0, q_1\} | \{q_0, q_2\}  \\
| \{q_0, q_2\} | \{q_0, q_1, q_3\} | \{q_0\}  \\

The diagram represents a nondeterministic finite automaton (NFA) with states \(q_0, q_1, q_2, q_3\) and transitions labeled with \(a\) and \(b\). The start state is \(q_0\), and \(q_3\) is a final state. The transitions are as follows:

- From \(q_0\) on \(a\) to \(q_1\)
- From \(q_1\) on \(b\) to \(q_2\)
- From \(q_2\) on \(a\) to \(q_3\)

The table specifies the transition function for \(a\) and \(b\) for the given states.
\[
\begin{array}{cccc}
\text{state} & a & b \\
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & & \\
\end{array}
\]
\[
\begin{array}{|c|c|c|}
\hline
q & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & & \\
\hline
\end{array}
\]
\[
\begin{array}{c|cc}
\{ q_0 \} & \{ q_0, q_1 \} & \{ q_0 \} \\
\{ q_0, q_1 \} & \{ q_0, q_1 \} & \{ q_0, q_2 \} \\
\{ q_0, q_2 \} & \{ q_0, q_1, q_3 \} & \{ q_0 \} \\
\{ q_0, q_1, q_3 \} & & \\
\end{array}
\]
\[ \begin{array}{c|cc}
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & & \\
\end{array} \]
<table>
<thead>
<tr>
<th>State Set</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_0}</td>
<td>{q_0, q_1}</td>
<td>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td>{q_0, q_1}</td>
<td>{q_0, q_2}</td>
</tr>
<tr>
<td>{q_0, q_2}</td>
<td>{q_0, q_1, q_3}</td>
<td>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1, q_3}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_0}</td>
<td>{q_0, q_1}</td>
<td>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td>{q_0, q_1}</td>
<td>{q_0, q_2}</td>
</tr>
<tr>
<td>{q_0, q_2}</td>
<td>{q_0, q_1, q_3}</td>
<td>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1, q_3}</td>
<td>{q_0, q_1}</td>
<td></td>
</tr>
</tbody>
</table>
### Automaton

![Automaton Diagram](attachment:automaton.png)

### Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>On $a$</th>
<th>On $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0, q_2}$</td>
</tr>
<tr>
<td>${q_0, q_2}$</td>
<td>${q_0, q_1, q_3}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1, q_3}$</td>
<td>${q_0, q_1}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0, q_2}$</td>
</tr>
<tr>
<td>${q_0, q_2}$</td>
<td>${q_0, q_1, q_3}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1, q_3}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0, q_2}$</td>
</tr>
</tbody>
</table>
start

$\Sigma$

$q_0$ -> $q_1$ (a)
$q_1$ -> $q_2$ (b)
$q_2$ -> $q_3$ (a)

### Transition Table

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0, q_2}$</td>
</tr>
<tr>
<td>${q_0, q_2}$</td>
<td>${q_0, q_1, q_3}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1, q_3}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0, q_2}$</td>
</tr>
</tbody>
</table>

### Diagrams

start

- $b$ from $\{q_0\}$
- $a$ from $\{q_0\}$
- $b$ from $\{q_0, q_2\}$
- $a$ from $\{q_0, q_1\}$
- $a$ from $\{q_0, q_1, q_3\}$
\begin{align*}
\{q_0\} & \quad \{q_0, q_1\} & \quad \{q_0\} \\
\{q_0, q_1\} & \quad \{q_0, q_1\} & \quad \{q_0, q_2\} \\
\{q_0, q_2\} & \quad \{q_0, q_1, q_3\} & \quad \{q_0\} \\
\*\{q_0, q_1, q_3\} & \quad \{q_0, q_1\} & \quad \{q_0, q_2\}
\end{align*}
\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[
\begin{array}{ccccccc}
\text{a} & \text{b} & \text{a} & \text{a} & \text{a} & \text{b} & \text{a} & \text{a} \\
\end{array}
\]

\[
\begin{array}{c}
\{ q_0 \} \\
\{ q_0, q_1 \} \\
\{ q_0, q_2 \} \\
\{ q_0, q_1, q_3 \}
\end{array}
\]
q₀ \rightarrow q₁ \rightarrow q₂ \rightarrow q₃

\Sigma

\{q₀, q₁\}

\{q₀, q₂\}

\{q₀, q₁, q₃\}
The diagram depicts a state transition diagram for a deterministic finite automaton (DFA). The initial state is $q_0$, and the transitions are labeled with symbols $a$ and $b$.

- From $q_0$, on input $a$, move to $q_1$.
- From $q_1$, on input $b$, move to $q_2$.
- From $q_2$, on input $a$, move back to $q_0$.
- From $q_3$, which is a final state, there is a loop labeled with $\Sigma$.

The input sequence presented is "a b a a a b a a."
The Subset Construction

- This procedure for turning an NFA for a language $L$ into a DFA for a language $L$ is called the subset construction.
  - It’s sometimes called the powerset construction; it’s different names for the same thing!
- Intuitively:
  - Each state in the DFA corresponds to a set of states from the NFA.
  - Each transition in the DFA corresponds to what transitions would be taken in the NFA when using the massive parallel intuition.
  - The accepting states in the DFA correspond to which sets of states would be considered accepting in the NFA when using the massive parallel intuition.
The Subset Construction

- In converting an NFA to a DFA, the DFA's states correspond to sets of NFA states.
- **Useful fact:** $|\mathcal{P}(S)| = 2^{|S|}$ for any finite set $S$.
- In the worst-case, the construction can result in a DFA that is exponentially larger than the original NFA.
- **Question to ponder:** Can you find a family of languages that have NFAs of size $n$, but no DFAs of size less than $2^n$?
A language $L$ is called a **regular language** if there exists a DFA $D$ such that $\mathcal{L}(D) = L$. 
An Important Result

**Theorem:** A language $L$ is regular if and only if there is some NFA $N$ such that $\mathcal{L}(N) = L$.

**Proof Sketch:** Pick a language $L$. First, assume $L$ is regular. That means there’s a DFA $D$ where $\mathcal{L}(D) = L$. Every DFA is “basically” an NFA, so there’s an NFA $(D)$ whose language is $L$.

Next, assume there’s an NFA $N$ such that $\mathcal{L}(N) = L$. Using the subset construction, we can build a DFA $D$ where $\mathcal{L}(N) = \mathcal{L}(D)$. Then we have that $\mathcal{L}(D) = L$, so $L$ is regular. ■-ish
Why This Matters

• We now have two perspectives on regular languages:
  • Regular languages are languages accepted by DFAs.
  • Regular languages are languages accepted by NFAs.
• We can now reason about the regular languages in two different ways.
Properties of Regular Languages
The Complement of a Language

- Given a language $L \subseteq \Sigma^*$, the complement of that language (denoted $\overline{L}$) is the language of all strings in $\Sigma^*$ that aren't in $L$.
- Formally:
  
  $$\overline{L} = \Sigma^* - L$$
The Complement of a Language

- Given a language $L \subseteq \Sigma^*$, the **complement** of that language (denoted $\overline{L}$) is the language of all strings in $\Sigma^*$ that aren't in $L$.
- Formally:

$$\overline{L} = \Sigma^* - L$$
The Complement of a Language

- Given a language $L \subseteq \Sigma^*$, the *complement* of that language (denoted $\overline{L}$) is the language of all strings in $\Sigma^*$ that aren't in $L$.
- Formally:
  \[ \overline{L} = \Sigma^* - L \]
The Complement of a Language

- Given a language $L \subseteq \Sigma^*$, the complement of that language (denoted $\overline{L}$) is the language of all strings in $\Sigma^*$ that aren't in $L$.
- Formally:

$$\overline{L} = \Sigma^* - L$$
The Complement of a Language

- Given a language $L \subseteq \Sigma^*$, the \textit{complement} of that language (denoted $\overline{L}$) is the language of all strings in $\Sigma^*$ that aren't in $L$.

- Formally:

  $$\overline{L} = \Sigma^* - L$$

Good proofwriting exercise: prove $\overline{\overline{L}} = L$ for any language $L$. 
Complementing Regular Languages

\[ L = \{ \ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring} \} \]

\[ \bar{L} = \{ \ w \in \{a, b\}^* \mid w \text{ does not contain } aa \text{ as a substring} \} \]
Complementing Regular Languages

\( L = \{ w \in \{a, *, /\}^* \mid w \text{ represents a C-style comment} \} \)
Complementing Regular Languages

\[ L = \{ w \in \{ a, *, / \}^* \mid w \text{ doesn't represent a C-style comment} \} \]
Complementing Regular Languages

\[ \overline{L} = \{ \ w \in \{a, *, /\}^* \ | \ w \text{ doesn't represent a C-style comment} \} \]
Closure Properties

- **Theorem:** If $L$ is a regular language, then $\overline{L}$ is also a regular language.
- As a result, we say that the regular languages are **closed under complementation**.

Question to ponder: are the nonregular languages closed under complementation?
The Union of Two Languages

- If $L_1$ and $L_2$ are languages over the alphabet $\Sigma$, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If $L_1$ and $L_2$ are regular languages, is $L_1 \cup L_2$?
The Union of Two Languages

- If $L_1$ and $L_2$ are languages over the alphabet $\Sigma$, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If $L_1$ and $L_2$ are regular languages, is $L_1 \cup L_2$?
The Union of Two Languages

- If $L_1$ and $L_2$ are languages over the alphabet $\Sigma$, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If $L_1$ and $L_2$ are regular languages, is $L_1 \cup L_2$?
The Union of Two Languages

- If $L_1$ and $L_2$ are languages over the alphabet $\Sigma$, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If $L_1$ and $L_2$ are regular languages, is $L_1 \cup L_2$?
The Union of Two Languages

- If $L_1$ and $L_2$ are languages over the alphabet $\Sigma$, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If $L_1$ and $L_2$ are regular languages, is $L_1 \cup L_2$?
The Intersection of Two Languages

- If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the language of strings in both $L_1$ and $L_2$.

- Question: If $L_1$ and $L_2$ are regular, is $L_1 \cap L_2$ regular as well?
The Intersection of Two Languages

- If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the language of strings in both $L_1$ and $L_2$.
- Question: If $L_1$ and $L_2$ are regular, is $L_1 \cap L_2$ regular as well?
The Intersection of Two Languages

- If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the language of strings in both $L_1$ and $L_2$.

- Question: If $L_1$ and $L_2$ are regular, is $L_1 \cap L_2$ regular as well?
The Intersection of Two Languages

• If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the language of strings in both $L_1$ and $L_2$.

• Question: If $L_1$ and $L_2$ are regular, is $L_1 \cap L_2$ regular as well?
The Intersection of Two Languages

- If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the language of strings in both $L_1$ and $L_2$.

- Question: If $L_1$ and $L_2$ are regular, is $L_1 \cap L_2$ regular as well?

Hey, it's De Morgan's laws!
Concatenation
String Concatenation

- If $w \in \Sigma^*$ and $x \in \Sigma^*$, the **concatenation** of $w$ and $x$, denoted $wx$, is the string formed by tacking all the characters of $x$ onto the end of $w$.

- Example: if $w = \text{quo}$ and $x = \text{kka}$, the concatenation $wx = \text{quokka}$.

- This is analogous to the + operator for strings in many programming languages.

- Some facts about concatenation:
  - The empty string $\varepsilon$ is the **identity element** for concatenation:
    
    \[
    w\varepsilon = \varepsilon w = w
    \]
  
  - Concatenation is **associative**:
    
    \[
    wxy = w(xy) = (wx)y
    \]
Concatenation

- The *concatenation* of two languages $L_1$ and $L_2$ over the alphabet $\Sigma$ is the language

$$L_1L_2 = \{ wx \in \Sigma^* \mid w \in L_1 \land x \in L_2 \}$$
Concatenation Example

Let \( \Sigma = \{ a, b, \ldots, z, A, B, \ldots, Z \} \) and consider these languages over \( \Sigma \):

- **Noun** = \{ Puppy, Rainbow, Whale, \ldots \}
- **Verb** = \{ Hugs, Juggles, Loves, \ldots \}
- **The** = \{ The \}
- **The language** **TheNounVerbTheNoun** is
Concatenation

• The *concatenation* of two languages $L_1$ and $L_2$ over the alphabet $\Sigma$ is the language

$$L_1L_2 = \{ wx \in \Sigma^* | w \in L_1 \land x \in L_2 \}$$

• Two views of $L_1L_2$:
  • The set of all strings that can be made by concatenating a string in $L_1$ with a string in $L_2$.
  • The set of strings that can be split into two pieces: a piece from $L_1$ and a piece from $L_2$. 
Concatenating Regular Languages

- If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?
- Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?
Concatenating Regular Languages

- If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?
- Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?

\[ \text{Machine for } L_1 \quad \text{start} \]
\[ \text{Machine for } L_2 \quad \text{start} \]
Concatenating Regular Languages

- If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?
- Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?

```
bookkeeper
```
Concatenating Regular Languages

- If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?
- Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?

![Machine for $L_1$](image1)

![Machine for $L_2$](image2)

bookkeeper
Concatenating Regular Languages

• If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?
• Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?

Machine for $L_1$

Machine for $L_2$
Concatenating Regular Languages

- If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?
- Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?
- **Idea:**
  - Run a DFA/NFA for $L_1$ on $w$.
  - Whenever it reaches an accepting state, optionally hand the rest of $w$ to a DFA/NFA for $L_2$.
  - If the automaton for $L_2$ accepts the rest, $w \in L_1L_2$.
  - If the automaton for $L_2$ rejects the remainder, the split was incorrect.
Concatenating Regular Languages
Concatenating Regular Languages

Machine for $L_1$
Concatenating Regular Languages

Machine for $L_1$

Machine for $L_2$
Concatenating Regular Languages

Machine for $L_1$

Machine for $L_2$
Concatenating Regular Languages

Machine for $L_1$

Machine for $L_2$
Concatenating Regular Languages

Machine for $L_1$

Machine for $L_2$

Machine for $L_1L_2$
Lots and Lots of Concatenation

• Consider the language \( L = \{ \text{aa, b} \} \)

• \( LL \) is the set of strings formed by concatenating pairs of strings in \( L \).

  \[
  \{ \text{aaaa, aab, baa, bb} \}
  \]

• \( LLL \) is the set of strings formed by concatenating triples of strings in \( L \).

  \[
  \{ \text{aaaaaa, aaaaab, aabaa, aabb, baaaa, baab, bbbaa, bbb} \}
  \]

• \( LLLLL \) is the set of strings formed by concatenating quadruples of strings in \( L \).

  \[
  \{ \text{aaaaaaaa, aaaaaaab, aaaaaabaa, aaaaabb, aabaaaa, aabaab, abbaaa, aabbaab, aabbb, baaaaaa, baaaaab, baabaa, baabb, bbaaaa, bbaab, bbbaa, bbb} \}
  \]
Language Exponentiation

• We can define what it means to “exponentiate” a language as follows:

• $L^0 = \{\varepsilon\}$
  • Intuition: The only string you can form by gluing no strings together is the empty string.
  • Notice that $\{\varepsilon\} \neq \emptyset$. Can you explain why?

• $L^{n+1} = LL^n$
  • Idea: Concatenating $(n+1)$ strings together works by concatenating $n$ strings, then concatenating one more.

• **Question to ponder:** Why define $L^0 = \{\varepsilon\}$?

• **Question to ponder:** What is $\emptyset^0$?
The Kleene Star
The Kleene Closure

• An important operation on languages is the **Kleene Closure**, which is defined as

\[ L^* = \{ w \in \Sigma^* \mid \exists n \in \mathbb{N}. w \in L^n \} \]

• Mathematically:

\[ w \in L^* \iff \exists n \in \mathbb{N}. w \in L^n \]

• Intuitively, \( L^* \) is the language all possible ways of concatenating zero or more strings in \( L \) together, possibly with repetition.

• **Question to ponder:** What is \( \emptyset^* \)?
The Kleene Closure

If $L = \{ a, bb \}$, then $L^* = \{$

$\varepsilon,$

$a, bb,$

$aa, aabb, abba, aabbb, bbbaa, bbabb, bbbba, bbbbbbb,$

$\ldots$\}$

Think of $L^*$ as the set of strings you can make if you have a collection of stamps – one for each string in $L$ – and you form every possible string that can be made from those stamps.
Idea: Can we convert an NFA for language $L$ to an NFA for language $L^*$?
The Kleene Star

Machine for $L$
The Kleene Star
The Kleene Star

Machine for $L$
The Kleene Star
The Kleene Star

Machine for $L$
The Kleene Star

Machine for $L$

Machine for $L^*$
The Kleene Star

Machine for $L$

Machine for $L^*$

Question: Why add the new state out front? Why not just make the old start state accepting?
Closure Properties

- **Theorem:** If $L_1$ and $L_2$ are regular languages over an alphabet $\Sigma$, then so are the following languages:
  - $\overline{L_1}$
  - $L_1 \cup L_2$
  - $L_1 \cap L_2$
  - $L_1 L_2$
  - $L_1^*$

- These properties are called **closure properties of the regular languages**.
Next Time

• *Regular Expressions*
  • Building languages from the ground up!

• *Thompson’s Algorithm*
  • A UNIX Programmer in Theoryland.

• *Kleene’s Theorem*
  • From machines to programs!