Deterministic Finite Automata
DFAs  (also called FSMs)

- A simple(st?) model of what a computer is
- Many devices modeled, programmed as DFAs
  - Vending machines
  - Elevators
  - Digital watch logic
  - Calculators
  - Lexical analysis part of program compilation
- Very limited, but observable universe is finite...
• Start state $q_0$
• Start at left, scan symbol, change state, move right.
• Rules of form “if in state $q$ scanning symbol $s$ then go to state $p$ and move right.”
• Some states (circled) are accepting.
• $M$ accepts the input string if a circled state is reached after scanning the last symbol.
Graphical Representation

- Directed graph with edges labeled with chars in $\Sigma$
- For each state (vertex) $q$ and symbol $a$ in $\Sigma$ there is exactly one edge leaving $q$ labeled with $a$. $q \xrightarrow{a} p$
- Accepting state(s) are double-circled
- Initial state has pointer, or is obviously labeled (0, $q_0$, “start”...)
Graphical Representation

- Where does 001 lead? 10010?
- Which strings end up in accepting state?
- Prove it
- *Every string* has one path that it follows

\[ q \xrightarrow{a} p \text{ versus } q \xrightarrow{w} p \]
Definition

• A DFA $M$ accepts a string $w$ iff the unique path starting at the initial state and spelling out $w$ ends at an accepting state.

• The language accepted (or “recognized”) by a DFA $M$ is denoted $L(M)$ and defined by

$$L(M) = \{ w \mid M \text{ accepts } w \}$$
Warning

• “$M$ accepts language $L$” does not mean simply that $M$ accepts each string in $L$.

• “$M$ accepts language $L$” means

  $M$ accepts each string in $L$ and no others!

• $M$ “recognizes” $L$ is a better term, but “accepts” is widely accepted (and recognized).
Examples: What is $L(M)$?
State = Memory

• The state of a DFA is its entire memory of what has come before
• The state must capture enough information to complete the computation on the suffix to come
• When designing a DFA, think “what do I need to know at this moment?” That is your state.
Construction Challenge

- $L(M) = \{w \mid w \text{ contains } 001 \text{ or } 010\}$
Construction Challenge

- $L(M) = \{ w \mid w \text{ contains } 001 \text{ or } 010 \}$

ends with
Construction Challenge

- $L(M) = \{w \mid w \text{ contains 001 or 010}\}$
Construction Challenge

- $L(M) = \{w \mid w \text{ contains 001 or 010}\} \text{ ends with}$
Construction Challenge

- \( L(M) = \{ w \mid w \text{ contains } 001 \text{ or } 010 \} \)
Construction Challenge

- $L(M) = \{ w \mid w \text{ contains } 001 \text{ or } 010 \}$

ends with
Construction Challenge

- \( L(M) = \{ w \mid w \text{ contains 001 or 010} \} \)
Construction Challenge

- $L(M) = \{ w \mid w \text{ contains } 001 \text{ or } 010 \}$
Construction Challenge

- $L(M) = \{w \mid w \text{ contains 001 or 010}\}$

ends with
Binary #s congruent to 0 mod 5
(assume no leading 0s)

Key Idea
If $w \mod 5 = a$, then:

- $w_0 \mod 5 = 2a \mod 5$
- $w_1 \mod 5 = 2a + 1 \mod 5$

Test: $1101011 = 107 = 2 \mod 5$
A DFA is a quintuple $M=(Q,\Sigma,\delta,q_0,F)$, where:

- $Q$ is a finite set of states
- $\Sigma$ is a finite alphabet of symbols
- $\delta: Q \times \Sigma \rightarrow Q$ is a transition function
- $q_0$ is the initial state
- $F \subseteq Q$ is the set of accepting states
Example

- $Q = \{0, 1, 2, 3\}$
- $\Sigma = \{a, b\}$
- $\delta$ specified at right
- $q_0 = 0$
- $F = \{3\}$
Extending $\delta$

- $\delta(q,a) = p$ means in graph that $q \xrightarrow{a} p$
- But how can we define $\delta(q,w)$ to express $q \xrightarrow{w} p$
- Must extend $\delta$: $Q \times \Sigma^* \rightarrow Q$
  - $\delta(q,\varepsilon) = q$ for every $q$; $\delta(q,a)$ already defined
  - $\delta(q,au) = \delta(\delta(q,a),u)$ for $|u| \geq 1$, all $q, a$

\[ \delta(q,w) = p \text{ corresponds to } q \xrightarrow{w} p \]
Formal definition of $L(M)$

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA

Then $L(M) = \{w \mid \delta(q_0, w) \in F\}$

We will show later that:

**Theorem**

$L$ is regular if and only if $L = L(M)$ for some DFA $M$
Example use

$L(M) = \{w \mid w \text{ in base } b \text{ is congruent to } k \mod m\}$

- $Q = \{0, 1, \ldots, m-1\}$
- $\Sigma = \{0, 1, \ldots, b-1\}$
- $q_0 = 0$
- $\delta (n, a) = bn + a \mod m$
- $F = \{k\}$
$M$ simulating both $M_1$ and $M_2$

$M_1$ accepts #0 = odd

$M_2$ accepts #1 = odd

Cross-product machine
\( M \) accepting \( L(M_1) \cap L(M_2) \)

\[
Q = Q_1 \times Q_2 \\
q_0 = (q_0^{(1)}, q_0^{(2)}) \\
F = F_1 \times F_2 = \{ (q_1, q_2) \mid q_1 \text{ in } F_1 \text{ and } q_2 \text{ in } F_2 \}
\]

Transition function:
\[
\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))
\]

\[
(q_1, q_2) \xrightarrow{a} (p_1, p_2) \text{ if and only if} \\
\bullet \quad q_1 \xrightarrow{a_1} p_1 \text{ in } M_1 \\
\bullet \quad q_2 \xrightarrow{a_2} p_2 \text{ in } M_2
\]
Proof that simulation is correct

• Induction on what? that what?
• Will need to prove that action of machine is correct starting from any states.

• We know that:

\[(q_1, q_2) \xrightarrow{a} (p_1, p_2) \text{ iff}\]
  \[\bullet q_1 \xrightarrow{a_1} p_1 \text{ in } M_1\]
  \[\bullet q_2 \xrightarrow{a_2} p_2 \text{ in } M_2\]

By definition

Show that:

\[(q_1, q_2) \xrightarrow{w} (p_1, p_2) \text{ iff}\]
  \[\bullet q_1 \xrightarrow{w_1} p_1 \text{ in } M_1\]
  \[\bullet q_2 \xrightarrow{w_2} p_2 \text{ in } M_2\]

Just like definition of \(\delta\), but with \(w\) instead of \(a\)
DEF for M:
\[(q_1, q_2) \xrightarrow{a} (p_1, p_2)\]
means
\[q_1 \xrightarrow{1} p_1 \quad \text{and} \quad q_2 \xrightarrow{2} p_2\]

**BY INDUCTION on \(|w|\)**

\[\text{iff} \quad (q_1, q_2) \xrightarrow{w} (p_1, p_2) \quad \text{and} \quad q_1 \xrightarrow{1} p_1 \quad \text{and} \quad q_2 \xrightarrow{2} p_2\]

---

**Diagram:**

- **Pull apart the computation:**
  - \[(q_1, q_2) \xrightarrow{a} (r_1, r_2)\]
  - \[q_1 \xrightarrow{1} r_1 \quad \text{and} \quad q_2 \xrightarrow{2} r_2\]

- **Apply definition:**
  - \[(r_1, r_2) \xrightarrow{u} (p_1, p_2)\]

- **Apply inductive hypothesis since \(|u| < |w|\):**
  - \[(r_1, r_2) \xrightarrow{u} (p_1, p_2)\]

- **Paste the computations back together:**
  - \[q_1 \xrightarrow{1} p_1 \quad \text{and} \quad q_2 \xrightarrow{2} p_2\]
• We proved:

\[(q_1, q_2) \xrightarrow{w} (p_1, p_2)\]
\[\text{iff}\]
\[q_1 \xrightarrow{1} p_1 \quad \text{and} \quad q_2 \xrightarrow{2} p_2\]

By definition, \(w\) accepted by \(M\)

\[\text{iff} \quad (q_0^{(1)} q_0^{(2)}) \xrightarrow{w} (f_1, f_2) \text{ in } F_1 \times F_2\]

\[\text{iff} \quad q_0^{(1)} \xrightarrow{w} f_1 \text{ in } F_1 \quad \text{AND} \quad q_0^{(2)} \xrightarrow{w} f_2 \text{ in } F_2\]

\[\text{iff} \quad w \text{ in } L(M_1) \text{ AND } w \text{ in } L(M_2)\]
Formal proof that simulation is correct

• We know by definition that:
  – for all $q_1 \text{ in } Q_1$, for all $q_2 \text{ in } Q_2$
  – for all characters $a$
    \[ \delta( (q_1, q_2), a ) = (\delta_1(q_1,a),\delta_2(q_2,a)) \]

• We prove by induction on $|w|$ that:
  – for all $q_1 \text{ in } Q_1$, for all $q_2 \text{ in } Q_2$
  – for all strings $w$
    \[ \delta( (q_1, q_2), w ) = (\delta_1(q_1,w),\delta_2(q_2,w)) \]

Looks just like definition of $\delta$, but with $w$ instead of $a$
To prove: \( \delta((q_1, q_2), w) = (\delta_1(q_1, w), \delta_2(q_2, w)) \)

Induction on \(|w|\)

- **Base Case:** \(|w| = 0\), so \(w = \varepsilon\).
  \[ \delta((q_1, q_2), \varepsilon) = (q_1, q_2) = (\delta_1(q_1, \varepsilon), \delta_2(q_2, \varepsilon)) \]

- **Assume true for strings** \(u\) **of length** \(< n\).

- **Let** \(w = au\) be an arbitrary string of length \(n\).

- **\(\delta((q_1, q_2), au)\)**
  \[
  \begin{align*}
  &= \delta(\delta((q_1, q_2), a), u) & \text{defn of } \delta \text{ extension} \\
  &= \delta(\delta_1(q_1, a), \delta_2(q_2, a)), u) & \text{by defn of } \delta \\
  &= \delta((r_1, r_2), u) & \text{define } r's \text{ to simplify} \\
  &= (\delta_1(r_1, u), \delta_2(r_2, u)) & \text{by induction } (|u| < n) \\
  &= (\delta_1(\delta_1(q_1, a), u), \delta_2(\delta_2(q_2, a), u)) & \text{get rid of } r's \\
  &= (\delta_1(q_1, au), \delta_2(q_2, au)) & \text{unsplitting} \\
  &= (\delta_1(q_1, w), \delta_2(q_2, w))
  \end{align*}
\]
Properties of Regular languages

• We’ve shown how to accept intersection of two regular languages
• What about union?
• If $L$ is accepted by a DFA, what about $\overline{L}$?
• What about concatenation, and Kleene $*$?
• Is there a DFA for $L_1 - L_2$ given $M_1$ and $M_2$?

The answer to all of these questions, and more, is “Yes.”
DFA for $L_n = n^{th}$ char from end is 1

- Example: $L_2 = \{w | \text{next to last char is "1"} \}$
- What needs to be remembered?
- Previous character?

State name = last two characters seen
DFA for \( L_n = n^{th} \) char from end is 1

- \( Q = \) set of \( n \)-bit strings
- \( \Sigma = \{0,1\} \)
- \( q_0 = 000....0 \) (why?)
- \( \delta (b_1b_2b_3...b_n, 0) = b_2b_3...b_n 0 \)
- \( \delta (b_1b_2b_3...b_n, 1) = b_2b_3...b_n 1 \)
- \( F = \{1b_2b_3...b_n | b_i \text{ in } \{0,1\}\} \)
Any DFA for $L_n$ needs $\geq 2^n$ states

Proof by contradiction
• If not, then two different $n$-bit strings $u$ and $v$ must lead to the same state of $M$.

Proof:
- Consider two $n$-bit strings $u = \ldots 0 \ldots 1 \ldots$ and $v = \ldots 1 \ldots 0000\ldots$.
- These two strings differ by their $(n-k)$th bit, where $k$ is the index of the differing bit.
- Add $k$ zeros to both strings to form $u' = 0000\ldots 0000\ldots$ and $v' = 0000\ldots 0000\ldots$.
- If $M$ is in the same state for both $u'$ and $v'$, then $M$ must have accepted or rejected one of the strings.

Diagram:

- State $p$ is in the same state for $u'$ and $v'$.
- State $q$ is in the same state for $u'$ and $v'$.

Is state $p$ accepting or not accepting?
Things to Know: 2-way DFA

• Why are DFAs required to only move forward?
• Why not allow DFA to scan back and forth?
• “2-way DFA” is such a model.
  – As long as it isn’t allowed to write, 2-way DFAs accept only the regular languages, thus are no more powerful than regular DFAs.
  – Proof is a very tricky simulation of 2-way DFA by a (1-way) NFA.