Finite State Automata
Deterministic Finite State Automata

**Deterministic Automata (DFSA)**

- \( M = \{ Q, \Sigma, \delta, q_0, F \} \)
  - \( \Sigma = \text{Symbols (Input)} \)
  - \( Q = \text{States} \)
  - \( q_0 = \text{Initial State} \)
  - \( F = \text{Final (Accepting) States} \subseteq Q \)
  - \( \delta : Q \times \Sigma \rightarrow Q = \text{Transition Functions} \)
Languages and Finite State Automata

Language of a Machine

- \( L(M) = \{ x \in \Sigma^* \mid M \text{ accepts } x \} \).

- \( M \) accepts \( x_1 \cdots x_n \) if there is any sequence of states \( q_0, q_1, \ldots, q_n \) such that
  
  \[
  \begin{align*}
  q_i &= \delta(x_i, q_{i-1}) \\
  q_n &\in F
  \end{align*}
  \]
Applications

String Searching

• Finding Keywords

• Pattern Recognition
Examples and Diagrams

Solving Problems

• Dice
• Parity Checker
• More Examples

Recognizing Languages

• Special Strings from $\sum = \{2,3,4,\ldots,12\}$
• Special Strings from $\sum = \{0,1\}$
• More Examples
**Variations**

*Moore Machines*
- States have Output
- Traffic Lights

*Mealy Machines*
- State–Symbol Pairs have Output
- Coder / Decoder

*Markov Models*
- Transitions Depend on Probabilities
- Computer Games

*Buchi Automata*
- Strings have Infinite Length -- No Halting State
- Air Traffic Control
Moore Machines -- DFSA where States have Output

Moore Machine

- \( M = \{Q, \Sigma, O, \delta, q_0, F\} \)

  - \( \Sigma = \) Input Symbols
  - \( O = \) Output Symbols
  - \( Q = \) States
  - \( q_0 = \) Initial State
  - \( F = \) Final (Accepting) States \( \subseteq Q \)
  - \( \delta : Q \times \Sigma \rightarrow Q = \) Transition Functions
  - \( D : Q \rightarrow O = \) State Output
Mealy Machines -- DFSA where State–Symbol Pairs have Output

Mealy Machine

- \( M = \{ Q, \Sigma, O, \delta, q_0, F \} \)
  - \( \Sigma = \) Input Symbols
  - \( O = \) Output Symbols
  - \( Q = \) States
  - \( q_0 = \) Initial State
  - \( F = \) Final (Accepting) States \( \subseteq Q \)
  - \( \delta : Q \times \Sigma \rightarrow Q \times O = \) Transition Functions
Examples

*Moore Machine*

- Traffic Light

*Mealy Machines*

- Binary Adder
Languages and Finite State Automata

Language of a Machine

- \( L(M) = \{ x \in \Sigma^* \mid M \text{ accepts } x \} \).

- \( M \) accepts \( x_1 \cdots x_n \) if there is any sequence of states \( q_0, q_1, \ldots, q_n \) such that
  
  -- \( q_i = \delta(x_i, q_{i-1}) \)
  
  -- \( q_n \in F \)
Designing Finite State Machines

Clustering
- Two strings with the same future fate must wind up in the same state.

Complementary Machines
- Design a machine to accept the complementary language and then swap accepting and rejecting states.

Examples
- Even number of zeros and even number of ones -- 4 states
- $L = \{0^i 1^j 2^k \}$ -- zeros followed by ones followed by twos
- Strings that do not contain 001
Implementing Finite State Machines

Hardware Constructions

Software Simulations
Non-Deterministic Finite State Automata

Main Idea

- Same Symbol may Transition from One State to Several Different States

Two Interpretations

- Guess the Answer
- Follow all Possible Paths Simultaneously
Examples

1. $\Sigma = \{0,1\}$
   - $L = \{x \mid \text{fifth symbol from end is 1}\}$
   - How many states?

2. $\Sigma = \{0,1\}$
   - $L = \{x \mid \text{contains 101 or 110}\}$
   - How many states?

3. $\Sigma = \{0,1\}$
   - $L = \{x \mid \text{has two zeros separated by a nonempty string of length 4} \}$
   - How many states?
Examples

1. $\Sigma = \{0, 1\}$
   - $L = \{x \mid \text{fifth symbol from end is 1}\}$
   - 6 state machine
     -- Stay in first state until you encounter the correct 1.
     -- Then move through 4 more states.

2. $\Sigma = \{0, 1\}$
   - $L = \{x \mid \text{contains 101 or 110}\}$
   - 6 state machine

3. $\Sigma = (0 + 1)^*$
   - $L = \{x \mid \text{has two zeros separated by a nonempty string of length 4p}\}$
   - 7 state machine

Observation: Finite State Automata can do Modular Arithmetic.
More Examples

1. Missing Letter

2. Recognizing the Union of Two Languages

3. Recognizing a Specific Pattern or Sequence
Non-Deterministic Finite State Automata

Non Deterministic Automata (DFSA)

- $M = \{Q, \Sigma, \delta, q_0, F\}$
  - $\Sigma$ = Symbols
  - $Q$ = States
  - $q_0$ = Initial State
  - $F$ = Final (Accepting) States $\subseteq Q$
  - $\delta : Q \times \Sigma \rightarrow P(Q)$ = Transition Functions
    - $P(Q)$ = set of all subsets of $Q$
    - Transition to Next State is NOT Unique
    - Empty Transitions Permitted
Finite State Automata

**Deterministic Automata (DFSA)**

- \( M = \{Q, \Sigma, \delta, q_0, F\} \)
  - \( \Sigma = \text{Symbols} \)
  - \( Q = \text{States} \)
  - \( q_0 = \text{Initial State} \)
  - \( F = \text{Final (Accepting) States} \subseteq Q \)
  - \( \delta : Q \times \Sigma \rightarrow Q = \text{Transition Functions} \)

**Non-Deterministic Automata (NDFSA)**

- \( M = \{Q, \Sigma, \delta, q_0, F\} \) -- Same as DFSA
- \( \delta : Q \times \Sigma \rightarrow P(Q) = \text{Non-Determinism of Next State} \)
Meta Theorems

Nondeterminism Does Not Help

- Nondeterminism does NOT add Recognition Power to Finite State Automata
- Any Language that can be Recognized by a Nondeterministic Finite State Automata, can also be Recognized by a Deterministic Finite State Automata.

Nondeterminism Does Help

- Nondeterminism simplifies many proofs about Finite State Automata
- Nondeterministic machines are often simpler (many fewer states) than their deterministic counterparts
- Every Nondeterministic Finite State Machine is equivalent to a Deterministic Finite State Machine.
  -- There is an Algorithm to convert from Nondeterministic Machines to Deterministic Machines
Theory vs. Practice

Nondeterministic Machines are NEVER Built into Hardware

- Theoretical Devise -- Computer Science
- NOT a Practical Machine -- Electrical Engineering
  
  BUT
  
  - Can be Converted into Deterministic Machines
    OR
  
  - Can be Simulated Directly in Software
**DFSA from NDFSA**

**Theorem:** $L$ is accepted by NDFSA, $N$, then exists a DFSA, $M$, that also accepts $L$.

**Proof:**

**Main Idea:** State in $M$ $\iff$ Set of states in $N$.

- $N = \{ Q, \Sigma, \delta, q_0, F \}$
- $M = \{ P(Q), \Sigma, \delta^*, q^*_0, F^* \}$
  
  -- $q^*_0 = \{ q_0 \}$
  
  -- $F^* = \{ R \in P(Q) \mid F \cap R \neq \phi \}$
  
  -- $\delta^*(r, c) = \cup_{q \in R} \delta(q, c)$

**Claim**

By construction

$\delta^*(\{ q_0 \}, w) = R = \text{State in } M \iff \delta(q_0, w) = R = \text{set of reachable states in } N$
Claim

By construction
\[ \delta^* (\{q_0\}, w) = R = \text{State in } M \iff \delta(q_0, w) = R = \text{set of reachable states in } N \]

Proof: By induction on \(|w|\).

Base Case: \(|w| = 1\).
Immediate from the definition of \(\delta^*\).

Inductive Case: \(|w| > 1\).

\[ |w| > 1 \Rightarrow w = u v \quad |u| \geq 1, \quad |v| = 1 \]

By inductive hypothesis

\[ \delta^* (\{q_0\}, u) = \text{set of states } R \text{ in } N \text{ reachable by } u \]

Hence it follows that

\[ \delta^* (\{q_0\}, uv) = \delta^* (R, v) = \text{set of states in } N \text{ reachable by } w \]

Hence \(M\) accepts \(w \iff w\) can reach an accepting state in \(N\).  
QED
**Minimal DFSA**

*Problems*

- Find a Minimum DFSA $M$ that accepts a language $L$.

- Show that the minimal DFSA is unique.

- Determine whether a particular machine $M$ is minimal.

- Find a minimal machine that accepts the same language as a machine $M$. 
Indistinguishable Elements

Equivalence Relation

- \( x \approx_L y \) if and only if for ALL \( w \), \( xw \) and \( yw \) are either both in \( L \) or both not in \( L \)
- \( x \approx_L y \iff \forall w \ (xw \in L \iff yw \in L) \)

- \( x \approx_L y \) is an equivalence relation
  - Reflexive: \( x \approx_L x \)
  - Symmetric: \( x \approx_L y \) implies \( y \approx_L x \)
  - Transitive: \( (x \approx_L y \text{ and } y \approx_L z) \) implies \( x \approx_L z \)

Equivalence Classes

- \([x] = \) the equivalence class of \( x \)
- If \([x] = [y] \), then \( x \) and \( y \) are **indistinguishable** in \( L \)
Examples

Equivalent Classes

• Even number of zeros and even number of ones -- 4 equivalence classes

• Strings where fifth symbol from end is a 1 -- 6 state machine

• Strings with zeros followed by same number of ones -- ? equivalence classes

• Strings that do not contain 001 -- ? equivalence classes
Lower Bound on Number of States

Theorem

Let $M = \text{DFSA}$ that Accepts $L$

Then

Number of States of $M \geq$ Number of Equivalence Classes of $L$

Proof: Pigeonhole Principle
Non-Recognizable Languages

Corollary 1

There is no DFSA that recognizes:

Strings with zeros followed by same number of ones.

Corollary 2

DFSA cannot count!

Corollary 3

There must be stronger computers that DFSA -- Later
**Existence of Minimal DFSA**

*Theorem*

If the number of equivalence classes of $L$ is finite, then there exists a DFSA $M$, where

\[
\text{number of states of } M = \text{number of equivalence classes of } L
\]

Therefore $M$ is minimal.

*Proof:*

$M = \{Q, \Sigma, \delta, q_0, F\}$

-- $Q = \text{Equivalence Classes } [x] \text{ of } L = \text{States}$

-- $q_0 = [\varepsilon] = \text{Initial State}$

-- $\delta([x], y) = [xy] = \text{Transition Functions}$

-- $[x] \in F \iff x \in L = \text{Accepting States}$
Myhill-Nerode Theorem

Corollary

There exists a DFSA $M$ that accepts a language $L$ if and only if the number of equivalence classes of $L$ is finite.

Proof

$\Rightarrow$: Number of equivalence classes of $L \leq$ Number of states of $M$

$\Leftarrow$: By previous theorem
Minimizing Existing DFSA

Splitting Algorithm

• Initialize $Q = 2$ States:
  -- Accepting = List of all Accepting States
  -- Rejecting = List of all Rejecting States

• Split the States
  -- If there is a symbol $c$ for which two or more elements in a state transition to different states
    $$\delta(p,c) \neq \delta(q,c)$$
    then split the state into new equivalence classes of states

• Continue until no further splitting occurs
Minimizing Existing DFSA

**Reasons**

- Existing DFSA may be our only description of $L$.
- Two DFSA are equivalent -- recognize the same language -- if and only if they have the same minimal DFSA.

**Example**

- See Page 93.
- See tennis example (Rosen?).
**Minimizing Existing DFSA**

*Theorems*

- The machine generated by the algorithm accepts the same language as the original machine.

  -- A string $s$ winds up in a state $\{q_1, \ldots, q_n\}$ in the new machine if and only if $s$ winds up in one of the states $q_1, \ldots, q_n$ in the original machine -- induction on the length of the string.

  -- A string winds up in an accepting state in the new machine if and only if the string winds up in an accepting state in the original machine.

- The algorithm generates a minimal DFSA

  -- number of states of $M = \text{number of equivalence classes of } L$

  -- $x \approx_L y \iff \forall w (xw \in L \iff yw \in L) \iff \delta(\text{start, } x) = \delta(\text{start, } y)
Construction of Minimal DFSA

Steps

1. Construct a NDFSA

2. Construct a corresponding DFSA

3. Construct the Minimal DFSA