Deterministic Finite Automaton (a simple machine)

- How does it process the string: 0111
- Does it accept the string?
Anatomy of Deterministic Finite Automaton (DFA)

- Circles are states
- Big arrow start state
- Double circles – final states (finish)
- The alphabet is the set of symbols allowed

$L(M) = \text{Language of machine M}$

- For this DFA $M$
  $L(M) =$

- For this DFA $M$
  $L(M) =$
What is $L(M)$?

- $L(M) =$ Strings with an even number of 1’s

Notation

- An *alphabet* $\Sigma$ is a finite set (e.g. $\Sigma = \{0,1\}$)
- A string over $\Sigma$ is a finite-length sequence of elements of $\Sigma$
- For a string $x$, $|x|$ is the length of $x$
- The empty string will be denoted by $\lambda$.
  - $|\lambda| =$
  - Some books use the symbol $\epsilon$
- A language over $\Sigma$ is a set of strings over $\Sigma$
Formal Definition of DFA

- A DFA is a 5-tuple $M=(Q, \Sigma, \delta, q_0, F)$
  - $Q$ is a finite set of states
  - $\Sigma$ is the alphabet, a finite set
  - $\delta: Q \times \Sigma \rightarrow Q$ is the transition function
  - $q_0 \in Q$ is the start state
  - $F \subseteq Q$ is the set of final (or accept) states

$L(M) =$ the language of machine $M$
  - $L(M) =$ the set of all strings machine $M$ accepts

Example: $M=(Q, \Sigma, \delta, q_0, F)$

- $Q =$
- $\Sigma =$
- $\delta: Q \times \Sigma \rightarrow Q$ is the transition function
- Start state is $q_0$
- $F =$

Build a DFA that accepts all and only those strings that contain 001
Build a DFA that accepts all and only those strings that contain 001

Label states with meaning

Regular Language

- A language is regular if it is recognized (accepted) by a DFA
- \(L = \{w \mid w \text{ contains 001}\}\) is regular
- \(L = \{w \mid w \text{ has an even number of 1's}\}\) is regular

Union theorem

- Given two languages \(L_1\) and \(L_2\), define the union of \(L_1\) and \(L_2\) as
  \[L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}\]
- Theorem: The union of two regular languages is also a regular language
  - Proof sketch: Let
    - \(M_1 = (Q_1, \Sigma, \delta_1, s_0, F_1)\) be a DFA for \(L_1\)
    - \(M_2 = (Q_2, \Sigma, \delta_2, p_0, F_2)\) be a DFA for \(L_2\)
    - We want to construct a DFA
    - \(M = (Q, \Sigma, \delta, q_0, F)\) such that \(L = L_1 \cup L_2\)
    - IDEA: Run \(M_1\) and \(M_2\) at the same time
      
      \[Q = \text{pairs of states, one from } M_1 \text{ and one from } M_2\]
      
      \[(s_i, p_j) \mid s_i \in Q_1 \text{ and } p_j \in Q_2\]

Compsci 230 Fall 2013
Theorem: The union of two regular languages is also a regular language

- Proof sketch: Let
  - $M_1 = (Q_1, \Sigma, \delta_1, s_0, F_1)$ be a DFA for $L_1$
  - $M_2 = (Q_2, \Sigma, \delta_2, p_0, F_2)$ be a DFA for $L_2$
  - We want to construct a DFA $M = (Q, \Sigma, \delta, q_0, F)$ such that $L = L_1 \cup L_2$
  - IDEA: Run $M_1$ and $M_2$ at the same time
  - $Q = \text{pairs of states, one from } M_1 \text{ and one from } M_2$
  - $Q = \{(s_i, p_j) \mid s_i \in Q_1 \text{ and } p_j \in Q_2\}$
  - $Q = Q_1 \times Q_2$

Example: Compute the union of:

Result:
Regular operations

- Union: $A \cup B = \{w|w \in A \text{ or } w \in B\}$
- Intersection: $A \cap B = \{w|w \in A \text{ and } w \in B\}$
- Reverse: $A^R = \{w_1 \ldots w_k | w_k \ldots w_1 \in A\}$
- Negation: $\neg A = \{w|w \notin A\}$
- Concatenation: $A \cdot B = \{vw|v \in A \text{ and } w \in B\}$
- Star: $A^* = \{w_1 \ldots w_k | k \geq 0 \text{ and each } w_i \in A\}$

Theorem: The union of two regular languages is also a regular language

- Corollary: Any finite language is regular.
The “Grep” Problem

• Input: Text T of length t, string S of length n
• Problem: Does string S appear inside text T?
• Naïve method:
  \[ a_1, a_2, a_3, a_4, a_5, \ldots, a_t \]
  • Cost: Roughly nt comparisons

Automata Solution

Build a DFA M that accepts any string with S as a consecutive substring
Feed the text to M
Cost:
Real Life Uses of DFAs

- Grep
- Coke machines
- Thermostats (fridge)
- Elevators
- Train track switches
- Lexical Analyzers for Parsers (starting phase of a compiler)

Are all languages regular?

Consider the language

$$L = \{a^n b^n | n > 0\}$$

- A bunch of a’s followed by the same number of b’s.
Consider the language 
$L = \{a^n b^n | n > 0\}$
- A bunch of a’s followed by the same number of b’s.
- No DFA accepts this language!
- Can you prove this?

Another Example
- $L = \{a^n b^n | n > 0\}$ is not regular!
- No DFA has enough states to keep track of the number of a’s it might encounter.
- That is a weak argument!

Another Example
- $L = \text{strings where the number of occurrences of the pattern } ab \text{ is equal to the number of occurrences of the pattern } ba$
- Is this regular?
- Can you build a DFA for this?
- Is this just like the previous problem?
- Does M accept strings only with an equal number of ab’s and ba’s?
- YES
- What is the difference between this and $a_n b_n$?
Let’s look at a formal proof, but first

Pigeonhole Principle
• Given n boxes and m>n objects, at least one box most contain more than one object

Letterbox principle
• If the average number of letters per box is x, then some box will have at least x letters

Theorem: \( \{a_n b_n | n > 0 \} \) is not regular
• Proof (by contradiction):
• Assume L is regular
• \( \exists \) DFA M with k states that accepts L
• For each \( 0 \leq i \leq k \), let \( S_i \) be the state M is in after reading \( a_i \)
• \( \exists i, j \leq k \) such that \( S_i = S_j \), but \( i \neq j \)

A similar problem to
\( \{a^n b^n | n > 0 \} \)
• Given an arithmetic expression, are there an equal number of left and right parentheses?
• \((x + (3 * y + (4 - x) + (y*6)))\)
A similar problem to \( \{a^n b^n | n > 0\} \)
- Given an arithmetic expression, are there an equal number of left and right parentheses?
- \((x + (3 * y + (4 - x) + (y*6)))\)
- Cannot be regular. No DFA has enough states to keep track of the number of each.

Study Bee
- Deterministic Finite Automata (DFA)
  - Definition
  - Testing if they accept a string
  - Building automata
- Regular Languages
  - Definition
  - Closed under Union, Intersection, Negation
  - Using Pigeonhole principle to show language is not regular