What is computation?

- the set of strings that lead to "accept" is the language recognized by this machine
- if every other string leads to "reject", then this language is decided by the machine

Terminology

- finite alphabet \( \Sigma \): a set of symbols
- language \( L \subseteq \Sigma^* \): subset of strings over \( \Sigma \)
- a machine takes an input string and either
  - accepts, rejects, or
  - loops forever
- a machine recognizes the set of strings that lead to accept
- a machine decides a language \( L \) if it accepts \( x \in L \) and rejects \( x \notin L \)

What goes inside the box?

- We want the simplest mathematical formalization of computation possible.
- Strategy:
  - endow box with a feature of computation
  - try to characterize the languages decided
  - identify language we "know" real computers can decide that machine cannot
  - add new feature to overcome limits

Finite Automata

- simple model of computation
- reads input from left to right, one symbol at a time
- maintains state: information about what seen so far ("memory")
  - finite automaton has finite # of states: cannot remember more things for longer inputs
- 2 ways to describe: by diagram, or formally

FA diagrams

- read input one symbol at a time; follow arrows; accept if end in accept state
FA operation

• Example of FA operation:

input: 0 1 0 1
not accepted

What language does this FA decide?

$L = \{x : x \in \{0,1\}^*, x_1 = 1\}$

Example FA

• What language does this FA decide?

$L = \{x : x \in \{0,1\}^*, x \text{ has even # of 1s}\}$

• illustrates fundamental feature/limitation of FA:
  - “tiny” memory
  - in this example only “remembers” 1 bit of info.

FA formal definition

A finite automaton is a 5-tuple

$(Q, \Sigma, \delta, q_0, F)$

– $Q$ is a finite set called the states
– $\Sigma$ is a finite set called the alphabet
– $\delta: Q \times \Sigma \rightarrow Q$ is a function called the transition function
– $q_0$ is an element of $Q$ called the start state
– $F$ is a subset of $Q$ called the accept states

• Specification of this FA in formal terms:

  – $Q = \{\text{even, odd}\}$
  – $\Sigma = \{0,1\}$
  – $q_0 = \text{even}$
  – $F = \{\text{even}\}$
  – $\delta$ function:
    - $\delta(\text{even}, 0) = \text{even}$
    - $\delta(\text{even}, 1) = \text{odd}$
    - $\delta(\text{odd}, 0) = \text{odd}$
    - $\delta(\text{odd}, 1) = \text{even}$
Formal description of FA operation

finite automaton
M = (Q, Σ, δ, q₀, F)
accepts a string
w = w₁w₂w₃…wn ∈ Σ*
if ∃ sequence r₀,r₁,r₂,…,rn of states for which
– r₀ = q₀
– δ(rᵢ, wi₊₁) = rᵢ₊₁ for i = 0,1,2,…,n-1
– rn ∈ F

What now?

• We have a model of computation
  (Maybe this is it. Maybe everything we can do with real computers we can do with FA…)

• try to characterize the languages FAs can recognize
  – investigate closure under certain operations
  • show that some languages not of this type

Characterizing FA languages

• We will show that the set of languages recognized by FA is closed under:
  – union "C = (A ∪ B)"
  – concatenation "C = (A ⊕ B)"
  – star " C = A* "

• Meaning: if A and B are languages recognized by a FA, then C is a language recognized by a FA

Characterizing FA languages

• union "C = (A ∪ B)"
  (A ∪ B) = {x : x ∈ A or x ∈ B or both}

• concatenation "C = (A ⊕ B)"
  (A ⊕ B) = {xy : x ∈ A and y ∈ B}

• star " C = A* " (note: ε always in A*)
  A* = {x₁x₂x₃…xₖ : k ≥ 0 and each xᵢ ∈ A}

Concatenation attempt

(A ⊕ B) = {xy : x ∈ A and y ∈ B}

What label do we put on the new transitions?

Concatenation attempt

• Need it to happen "for free": label with ε (?)
• allows construct with multiple transitions with the same label (!?)
Nondeterministic FA

- We will make life easier by describing an additional feature (nondeterminism) that helps us to “program” FAs.
- We will prove that FAs with this new feature can be simulated by ordinary FA—same spirit as programming constructs like procedures.
- The concept of nondeterminism has a significant role in TCS and this course.

NFA diagrams

- At each step, several choices for next state—if possible to reach accept, then input accepted.

NFA diagrams

- May have several with a given label (or none).
- May be labeled with $\varepsilon$.

NFA operation

- Example of NFA operation:

```
input: 0 1 0
not accepted
```

NFA operation

- Example of NFA operation:

```
input: 1 1 0
accepted
```

NFA operation

- One way to think of NFA operation:

```
string $x = x_1 x_2 x_3 \ldots x_n$ accepted if and only if
  there exists a way of inserting $\varepsilon$’s into $x$
  $x_1 \varepsilon x_2 \varepsilon x_3 \ldots \varepsilon x_n$
  so that there exists a path of transitions from
  the start state to an accept state
```

NFA formal definition

A nondeterministic FA $(Q, \Sigma, \delta, q_0, F)$:

- $Q$ is a finite set called the states
- $\Sigma$ is a finite set called the alphabet
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$ is a function called the transition function
- $q_0$ is an element of $Q$ called the start state
- $F$ is a subset of $Q$ called the accept states
• Specification of this NFA in formal terms:
  - \( Q = \{ s_1, s_2, s_3, s_4 \} \)
  - \( \Sigma = \{ 0, 1 \} \)
  - \( q_0 = s_1 \)
  - \( F = \{ s_4 \} \)
  \[
  \begin{align*}
  \delta(s_1, 0) &= \{ s_1 \} \\
  \delta(s_1, 1) &= \{ s_1, s_2 \} \\
  \delta(s_1, \epsilon) &= \{ \} \\
  \delta(s_2, 0) &= \{ s_3 \} \\
  \delta(s_2, 1) &= \{ \} \\
  \delta(s_2, \epsilon) &= \{ s_3 \} \\
  \delta(s_3, 0) &= \{ \} \\
  \delta(s_3, 1) &= \{ s_4 \} \\
  \delta(s_3, \epsilon) &= \{ \} \\
  \delta(s_4, 0) &= \{ s_4 \} \\
  \delta(s_4, 1) &= \{ s_4 \} \\
  \delta(s_4, \epsilon) &= \{ \} 
  \end{align*}
  \]