Today

• **Finite Automata (FAs)**
  – Our third machine model, after circuits and decision trees.

• Designed to:
  – **Accept** some strings of symbols.
  – **Recognize** a language, which is the set of strings it accepts.

• **FA takes as its input a string of any length.**
  – One machine for all lengths.
  – Circuits and decision trees use a different machine for each length.

• **Today’s topics:**
  – Finite Automata and the languages they recognize
  – Examples
  – Operations on languages
  – Closure of FA languages under various operations
  – Nondeterministic FAs

• **Reading**: Sipser, Section 1.1.

• **Next**: Sections 1.2, 1.3.
Finite Automata and the languages they recognize
Example 1

- An FA diagram, machine M

- Conventions:
  - Start state
  - Accept state
  - Transition from a to b on input symbol 1. Allow self-loops
Example 1

- Example computation:
  - Input word w: 1 0 1 1 0 1 1 1 0
  - States: a b a b c a b c d d
- We say that M accepts w, since w leads to d, an accepting state.
In general...

• A FA M accepts a word w if w causes M to follow a path from the start state to an accept state.

• Some terminology and notation:
  – Finite alphabet of symbols, usually called Σ.
  – In Example 1 (and often), Σ = \{0,1\}.
  – String (word) over Σ: Finite sequence of symbols from Σ.
  – Length of w, |w|
  – ε, placeholder symbol for the empty string, |ε| = 0
  – Σ*, the set of all finite strings of symbols in Σ
  – Concatenation of strings w and x, written w ◦ x or w x.
  – L(M), language recognized by M:
    \{w | w is accepted by M \}.
  – What is L( M ) for Example 1?
Example 1

- What is \(L(M)\) for Example 1?
- \(\{ w \in \{0,1\}^* \mid w \text{ contains 111 as a substring} \}\)
- Note: Substring refers to consecutive symbols.
Formal Definition of an FA

• An FA is a 5-tuple \(( Q, \Sigma, \delta, q_0, F )\), where:
  - \( Q \) is a finite set of states,
  - \( \Sigma \) is a finite set (alphabet) of input symbols,
  - \( \delta: Q \times \Sigma \rightarrow Q \) is the transition function,
  - \( q_0 \in Q \), is the start state, and
  - \( F \subseteq Q \) is the set of accepting, or final states.

\( \delta \) takes a state and an alphabet symbol as arguments and returns a state.
Example 1

• What is the 5-tuple \((Q, \Sigma, \delta, q_0, F)\)?
• \(Q = \{ a, b, c, d \}\)
• \(\Sigma = \{ 0, 1 \}\)
• \(\delta\) is given by the state diagram, or alternatively, by a table:

<table>
<thead>
<tr>
<th>q</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
</tbody>
</table>
Formal definition of computation

- Extend the definition of $\delta$ to input strings and states:
  $\delta^*: Q \times \Sigma^* \rightarrow Q$, state and string yield a state
  $\delta^*( q, w ) = \text{state that is reached by starting at } q \text{ and following } w$.

- Defined recursively:
  $\delta^*( q, \varepsilon ) = q$
  $\delta^*( q, wa ) = \delta( \delta^*( q, w ), a )$

- Or iteratively, compute $\delta^*( q, a_1 a_2 \ldots a_k )$ by:
  $s := q$
  for $i = 1$ to $k$ do $s := \delta( s, a_i )$
Formal definition of computation

• String $w$ is **accepted** if $\delta^*(q_0, w) \in F$, that is, $w$ leads from the start state to an accepting state.
• String $w$ is **rejected** if it isn’t accepted.
• A **language** is any set of strings over some alphabet.
• $L(M)$, language recognized by finite automaton $M = \{ w \mid w$ is accepted by $M \}$.
• A language is **regular**, or **FA-recognizable**, if it is recognized by some finite automaton.
Examples of Finite Automata
Example 2

- Design an FA $M$ with $L(M) = \{ w \in \{0,1\}^* | w$ contains 101 as a substring $\}$.

- Failure from state $b$ causes the machine to remain in state $b$.
Example 3

• $L = \{ w \in \{0, 1\}^* \mid w$ doesn’t contain either 00 or 11 as a substring $\}$. 

• State $d$ is a **trap state** = a nonaccepting state that you can’t leave. 

• Sometimes we’ll omit some arrows; by convention, they go to a trap state.
Example 4

- $L = \{ w \mid \text{all nonempty blocks of 1s in } w \text{ have odd length} \}$.  
- E.g., $\varepsilon$, or 100111000011111, or any number of 0s.  
- Initial 0s don’t matter, so start with:

- Then 1 also leads to an accepting state, but it should be a different one, to “remember” that the string ends in one 1.
Example 4

• $L = \{ w \mid \text{all nonempty blocks of 1s in } w \text{ have odd length} \}$.

• From b:
  – 0 can return to a, which can represent either $\varepsilon$, or any string that is OK so far and ends with 0.
  – 1 should go to a new nonaccepting state, meaning “the string ends with two 1s”.

• Note: c isn’t a trap state---we can accept some extensions.
Example 4

- \( L = \{ w \mid \text{all nonempty blocks of 1s in } w \text{ have odd length} \} \).

- From c:
  - 1 can lead back to b, since future acceptance decisions are the same if the string so far ends with any odd number of 1s.
    - Reinterpret b as meaning “ends with an odd number of 1s”.
    - Reinterpret c as “ends with an even number of 1s”.
  - 0 means we must reject the current string and all extensions.

Diagram:

- States: a, b, c, d
- Transitions:
  - a \( \rightarrow \) b on 1
  - b \( \rightarrow \) c on 1
  - c \( \rightarrow \) a on 0
  - a \( \rightarrow \) d on 0
  - b \( \rightarrow \) c on 0
  - c \( \rightarrow \) d on 0
  - Trap state: d

0

1

0,1

Trap state
Example 4

- \( L = \{ w \mid \text{all nonempty blocks of 1s in } w \text{ have odd length} \} \).

- Meanings of states (more precisely):
  a: Either \( \varepsilon \), or contains no bad block (even block of 1s followed by 0) so far and ends with 0.
  b: No bad block so far, and ends with odd number of 1s.
  c: No bad block so far, and ends with even number of 1s.
  d: Contains a bad block.
Example 5

• $L = EQ = \{ w \mid w \text{ contains an equal number of 0s and 1s} \}$.

• No FA recognizes this language.

• Idea (not a proof):
  – Machine must “remember” how many 0s and 1s it has seen, or at least the difference between these numbers.
  – Since these numbers (and the difference) could be anything, there can’t be enough states to keep track.
  – So the machine will sometimes get confused and give a wrong answer.

• We’ll turn this into an actual proof next week.
Language Operations
Language operations

• Operations that can be used to construct languages from other languages.
• Recall: A language is any set of strings.
• Since languages are sets, we can use the usual set operations:
  – Union, $L_1 \cup L_2$
  – Intersection, $L_1 \cap L_2$
  – Complement, $L^c$
  – Set difference, $L_1 - L_2$
• We also have new operations defined especially for sets of strings:
  – Concatenation, $L_1 \circ L_2$ or just $L_1 L_2$
  – Star, $L^*$
Concatenation

• $L_1 \circ L_2 = \{ x y \mid x \in L_1 \text{ and } y \in L_2 \}$

– Pick one string from each language and concatenate them.

• Example:

$\Sigma = \{ 0, 1 \}$, $L_1 = \{ 0, 00 \}$, $L_2 = \{ 01, 001 \}$

$L_1 \circ L_2 = \{ 001, 0001, 00001 \}$

• Notes:

$| L_1 \circ L_2 | \leq | L_1 | \times | L_2 |$, not necessarily equal.

$L \circ L$ does not mean $\{ x x \mid x \in L \}$, but rather, $\{ x y \mid x \text{ and } y \text{ are both in } L \}$. 
Concatenation

• \( L_1 \circ L_2 = \{ x y \mid x \in L_1 \text{ and } y \in L_2 \} \)

• Example:
  \[
  \Sigma = \{ 0, 1 \}, L_1 = \{ 0, 00 \}, L_2 = \{ 01, 001 \}
  \]
  \[
  L_1 \circ L_2 = \{ 001, 0001, 00001 \}
  \]
  \[
  L_2 \circ L_2 = \{ 0101, 01001, 00101, 001001 \}
  \]

• Example: \( \emptyset \circ L \)
  \[
  \{ x y \mid x \in \emptyset \text{ and } y \in L \} = \emptyset
  \]

• Example: \( \{ \varepsilon \} \circ L \)
  \[
  \{ x y \mid x \in \{ \varepsilon \} \text{ and } y \in L \} = L
  \]
Concatenation

- \( L_1 \circ L_2 = \{ x \, y \mid x \in L_1 \text{ and } y \in L_2 \} \)
- Write \( L \circ L \) as \( L^2 \),
  \( L \circ L \circ \ldots \circ L \) as \( L^n \), which is \( \{ x_1 \, x_2 \, \ldots \, x_n \mid \text{all } x's \text{ are in } L \} \)

- Example: \( L = \{ 0, 11 \} \)
  \( L^3 = \{ 000, 0011, 0110, 01111, 1100, 11011, 11110, 111111 \} \)

- Example: \( L = \{ 0, 00 \} \)
  \( L^3 = \{ 000, 0000, 00000, 000000 \} \)

- Boundary cases:
  \( L^1 = L \)
  Define \( L^0 = \{ \varepsilon \} \), for every \( L \).
  - Implies that \( L^0 \, L^n = \{ \varepsilon \} \, L^n = L^n \).
  - Special case of general rule \( L^a \, L^b = L^{a+b} \).
The Star Operation

- \( L^* = \{ x \mid x = y_1 y_2 \ldots y_k \text{ for some } k \geq 0, \text{ where every } y \text{ is in } L \} \)
  
  \[ = L^0 \cup L^1 \cup L^2 \cup \ldots \]

- **Note:** \( \varepsilon \) is in \( L^* \) for every \( L \), since it’s in \( L^0 \).

- **Example:** What is \( \emptyset^* \) ?
  
  - Apply the definition:
    
    \[ \emptyset^* = \emptyset^0 \cup \emptyset^1 \cup \emptyset^2 \cup \ldots \]

    The rest of these are just \( \emptyset \).

    This is \( \{ \varepsilon \} \), by the convention that \( L^0 = \{ \varepsilon \} \).

    \[ = \{ \varepsilon \} . \]
The Star Operation

- \( L^* = L^0 \cup L^1 \cup L^2 \cup \ldots \)

- Example: What is \( \{ a \}^* \)?
  - Apply the definition:
    \[
    \{ a \}^* = \{ a \}^0 \cup \{ a \}^1 \cup \{ a \}^2 \cup \ldots \\
    = \{ \varepsilon \} \cup \{ a \} \cup \{ a a \} \cup \ldots \\
    = \{ \varepsilon, a, a a, a a a, \ldots \}
    \]
  - Abbreviate this to just \( a^* \).
  - Note this is not just one string, but a set of strings---any number of \( a \)’s.
The Star Operation

- \( L^* = L^0 \cup L^1 \cup L^2 \cup \ldots \)

- Example: What is \( \Sigma^* \) ?
  - We’ve already defined this to be the set of all finite strings over \( \Sigma \).
  - But now it has a new formal definition:

\[
\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots \\
= \{ \varepsilon \} \cup \{ \text{strings of length 1 over } \Sigma \} \\
\quad \cup \{ \text{strings of length 2 over } \Sigma \} \\
\quad \cup \ldots \\
= \{ \text{all finite strings over } \Sigma \} 
\]

- Consistent.
Summary: Language Operations

- Set operations: Union, intersection, complement, set difference
- New language operations: Concatenation, star
- Regular operations:
  - Of these six operations, we identify three as regular operations: union, concatenation, star.
  - We’ll revisit these next time, when we define regular expressions.
Closure of regular (FA-recognizable) languages under all six operations
Closure under operations

- The set of FA-recognizable languages is closed under all six operations (union, intersection, complement, set difference, concatenation, star).
- This means: If we start with FA-recognizable languages and apply any of these operations, we get another FA-recognizable language (for a different FA).

- **Theorem 1:** FA-recognizable languages are closed under complement.
- **Proof:**
  - Start with a language $L_1$ over alphabet $\Sigma$, recognized by some FA, $M_1$.
  - Produce another FA, $M_2$, with $L(M_2) = \Sigma^* - L(M_1)$.
  - Just interchange accepting and non-accepting states.
Closure under complement

- **Theorem 1**: FA-recognizable languages are closed under complement.
- **Proof**: Interchange accepting and non-accepting states.
- **Example**: FA for \( \{ w \mid w \text{ does not contain 111} \} \)
  - Start with FA for \( \{ w \mid w \text{ contains 111} \} \):
Closure under complement

• **Theorem 1**: FA-recognizable languages are closed under complement.
  
  • **Proof**: Interchange accepting and non-accepting states.
  
  • **Example**: FA for \( \{ w \mid w \text{ does not contain } 111 \} \)
    
    – Interchange accepting and non-accepting states:

```
0

a 1 b 1 c 1 d
```

0 1 0,1
Closure under intersection

• **Theorem 2:** FA-recognizable languages are closed under intersection.

• **Proof:**
  – Start with FAs $M_1$ and $M_2$ for the same alphabet $\Sigma$.
  – Get another FA, $M_3$, with $L(M_3) = L(M_1) \cap L(M_2)$.
  – Idea: Run $M_1$ and $M_2$ “in parallel” on the same input. If both reach accepting states, accept.
  – Example:
    • $L(M_1)$: Contains substring 01.
    • $L(M_2)$: Odd number of 1s.
    • $L(M_3)$: Contains 01 and has an odd number of 1s.
Closure under intersection

• Example:
  
  $M_1$: Substring 01

  $M_2$: Odd number of 1s

  $M_3$:
Closure under intersection, general rule

• Assume:
  – $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$
  – $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$

• Define $M_3 = (Q_3, \Sigma, \delta_3, q_{03}, F_3)$, where
  – $Q_3 = Q_1 \times Q_2$
    - Cartesian product, $\{(q_1, q_2) \mid q_1 \in Q_1 \text{ and } q_2 \in Q_2\}$
  – $\delta_3 ((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$
  – $q_{03} = (q_{01}, q_{02})$
  – $F_3 = F_1 \times F_2 = \{(q_1, q_2) \mid q_1 \in F_1 \text{ and } q_2 \in F_2\}$
Closure under union

• **Theorem 3:** FA-recognizable languages are closed under union.

• **Proof:**
  – Similar to intersection.
  – Start with FAs $M_1$ and $M_2$ for the same alphabet $\Sigma$.
  – Get another FA, $M_3$, with $L(M_3) = L(M_1) \cup L(M_2)$.
  – **Idea:** Run $M_1$ and $M_2$ “in parallel” on the same input. If either reaches an accepting state, accept.
  – **Example:**
    • $L(M_1)$: Contains substring $01$.
    • $L(M_2)$: Odd number of $1$s.
    • $L(M_3)$: Contains $01$ or has an odd number of $1$s.
Closure under union

- Example:
  - $M_1$: Substring 01
  - $M_2$: Odd number of 1s
  - $M_3$: 1
Closure under union, general rule

• Assume:
  – \( M_1 = ( Q_1, \Sigma, \delta_1, q_{01}, F_1 ) \)
  – \( M_2 = ( Q_2, \Sigma, \delta_2, q_{02}, F_2 ) \)

• Define \( M_3 = ( Q_3, \Sigma, \delta_3, q_{03}, F_3 ) \), where
  – \( Q_3 = Q_1 \times Q_2 \)
    • Cartesian product, \( \{(q_1,q_2) \mid q_1 \in Q_1 \text{ and } q_2 \in Q_2 \} \)
  – \( \delta_3 ((q_1,q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a)) \)
  – \( q_{03} = (q_{01}, q_{02}) \)
  – \( F_3 = \{(q_1,q_2) \mid q_1 \in F_1 \text{ or } q_2 \in F_2 \} \)
Closure under set difference

• **Theorem 4:** FA-recognizable languages are closed under set difference.

• **Proof:**
  – Similar proof to those for union and intersection.
  – Alternatively, since $L_1 - L_2$ is the same as $L_1 \cap (L_2)^c$, we can just apply Theorems 2 and 3.
Closure under concatenation

- **Theorem 5**: FA-recognizable languages are closed under concatenation.

- **Proof:**
  - Start with FAs $M_1$ and $M_2$ for the same alphabet $\Sigma$.
  - Get another FA, $M_3$, with $L(M_3) = L(M_1) \circ L(M_2)$, which is
    \[ \{ x_1 x_2 \mid x_1 \in L(M_1) \text{ and } x_2 \in L(M_2) \} \]
  - Idea: ???
    - Attach accepting states of $M_1$ somehow to the start state of $M_2$.
    - But we have to be careful, since we don’t know when we’re done with the part of the string in $L(M_1)$---the string could go through accepting states of $M_1$ several times.
Closure under concatenation

- **Theorem 5**: FA-recognizable languages are closed under concatenation.

- **Example**:
  - $\Sigma = \{ 0, 1\}$, $L_1 = \Sigma^*$, $L_2 = \{0\} \{0\}^*$ (just 0s, at least one).
  - $L_1 L_2 = \text{strings that end with a block of at least one } 0$
  - $M_1$:
    - How to combine?
    - We seem to need to “guess” when to shift to $M_2$.
    - Leads to our next model, NFAs, which are FAs that can guess.
Closure under star

• **Theorem 6:** FA-recognizable languages are closed under star.

• **Proof:**
  – Start with FA $M_1$.
  – Get another FA, $M_2$, with $L(M_2) = L(M_1)^*$.
  – Same problems as for concatenation---need guessing.
  – …
  – We’ll define NFAs next, then return to complete the proofs of Theorems 5 and 6.
Nondeterministic Finite Automata
Nondeterministic Finite Automata

• Generalize FAs by adding nondeterminism, allowing several alternative computations on the same input string.
• Ordinary deterministic FAs follow one path on each input.
• Two changes:
  – Allow $\delta(q, a)$ to specify more than one successor state:
    
    ![Diagram of multiple successors](image)

  – Add $\varepsilon$-transitions, transitions made “for free”, without “consuming” any input symbols:
    
    ![Diagram of $\varepsilon$-transitions](image)

• Formally, combine these changes:
Formal Definition of an NFA

- An NFA is a 5-tuple (Q, Σ, δ, q₀, F), where:
  - Q is a finite set of states,
  - Σ is a finite set (alphabet) of input symbols,
  - δ: Q × Σₑ → P(Q) is the transition function,
  - q₀ ∈ Q, is the start state, and
  - F ⊆ Q is the set of accepting, or final states.

The arguments are a state and either an alphabet symbol or ε. Σₑ means Σ ∪ {ε}.

The result is a set of states.
Formal Definition of an NFA

• An NFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\), where:
  – \(Q\) is a finite set of states,
  – \(\Sigma\) is a finite set (alphabet) of input symbols,
  – \(\delta: Q \times \Sigma_\varepsilon \rightarrow P(Q)\) is the transition function,
  – \(q_0 \in Q\), is the start state, and
  – \(F \subseteq Q\) is the set of accepting, or final states.

• How many states in \(P(Q)\)?
  \(2^{|Q|}\)

• Example: \(Q = \{a, b, c\}\)
  \(P(Q) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a,b,c\}\}\)
NFA Example 1

Q = { a, b, c }
Σ = { 0, 1 }
q₀ = a
F = { c }

δ:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>ε</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>{a,b}</td>
<td>{a}</td>
<td>∅</td>
</tr>
<tr>
<td>b</td>
<td>∅</td>
<td>{c}</td>
<td>∅</td>
</tr>
<tr>
<td>c</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>
NFA Example 2

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>ε</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>{a}</td>
<td>{a}</td>
<td>{b,c}</td>
</tr>
<tr>
<td>b</td>
<td>{c}</td>
<td>Ø</td>
<td>Ø</td>
</tr>
<tr>
<td>c</td>
<td>Ø</td>
<td>{d}</td>
<td>Ø</td>
</tr>
<tr>
<td>d</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
</tr>
<tr>
<td>e</td>
<td>Ø</td>
<td>{f}</td>
<td>Ø</td>
</tr>
<tr>
<td>f</td>
<td>{g}</td>
<td>Ø</td>
<td>Ø</td>
</tr>
<tr>
<td>g</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
</tr>
</tbody>
</table>
Next time…

• NFAs and how they compute
• NFAs vs. FAs
• Closure of regular languages under languages operations, revisited
• Regular expressions
• Regular expressions denote FA-recognizable languages.

• Reading: Sipser, Sections 1.2, 1.3