Finite State Machines

Chapter 5
Languages and Machines

- Turing Machines
- PDAs
- FSMs
- Regular Languages
- Context-Free Languages
- D Languages
- SD Languages
Regular Languages

Regular Expression $L$

Regular Language

Accepts

Finite State Machine
Finite State Machines

An FSM to accept $.50 in change:
Definition of a DFSM

\[ M = (K, \Sigma, \delta, s, A), \]  where:

- \( K \) is a finite set of states
- \( \Sigma \) is an alphabet
- \( s \in K \) is the initial state
- \( A \subseteq K \) is the set of accepting states, and
- \( \delta \) is the transition function from \((K \times \Sigma)\) to \( K\)
Accepting by a DFSM

Informally, $M$ accepts a string $w$ iff $M$ winds up in some element of $A$ when it has finished reading $w$.

The language accepted by $M$, denoted $L(M)$, is the set of all strings accepted by $M$. 
Configurations of DFSMs

A configuration of a DFSM $M$ is an element of:

$$K \times \Sigma^*$$

It captures the two things that can make a difference to $M$’s future behavior:

- its current state
- the input that is still left to read.

The initial configuration of a DFSM $M$, on input $w$, is:

$$(s_M, w)$$
The Yields Relations

The yields-in-one-step relation $|-_M$:

$$(q, w) |-_M (q', w')$$ iff

- $w = aw'$ for some symbol $a \in \Sigma$, and
- $\delta (q, a) = q'$

$|-_M^*$ is the reflexive, transitive closure of $|-_M$. 
A computation by $M$ is a finite sequence of configurations $C_0, C_1, \ldots, C_n$ for some $n \geq 0$ such that:

- $C_0$ is an initial configuration,
- $C_n$ is of the form $(q, \varepsilon)$, for some state $q \in K_M$,
- $C_0 \mathbin{|-M} C_1 \mathbin{|-M} C_2 \mathbin{|-M} \ldots \mathbin{|-M} C_n$. 


Accepting and Rejecting

A DFSM $M$ accepts a string $w$ iff:

$$(s, w) |-_M^* (q, \varepsilon), \text{ for some } q \in A.$$ 

A DFSM $M$ rejects a string $w$ iff:

$$(s, w) |-_M^* (q, \varepsilon), \text{ for some } q \notin A_M.$$ 

The language accepted by $M$, denoted $L(M)$, is the set of all strings accepted by $M$.

**Theorem:** Every DFSM $M$, on input $s$, halts in $|s|$ steps.
An Example Computation

An FSM to accept odd integers:

On input 235, the configurations are:

\[(q_0, 235) |-_M (q_0, 35) \]
\[ |-_M \]
\[ |-_M \]
\[ |-_M \]

Thus \((q_0, 235) |-_M^* (q_1, \varepsilon)\)
Regular Languages

A language is *regular* iff it is accepted by some FSM.
A Very Simple Example

\[ L = \{ w \in \{a, b\}^* : \text{every } a \text{ is immediately followed by } a \ b \}. \]
A Very Simple Example

\[ L = \{ w \in \{a, b\}^* : \text{every } a \text{ is immediately followed by } a \ b \} . \]
Parity Checking

\[ L = \{ w \in \{0, 1\}^* : w \text{ has odd parity} \}. \]
Parity Checking

\[ L = \{ w \in \{0, 1\}^* : w \text{ has odd parity} \} . \]
No More Than One b

\[ L = \{ w \in \{a, b\}^* : \text{every } a \text{ region in } w \text{ is of even length} \} \]
No More Than One b

$L = \{ w \in \{a, b\}^* : \text{every } a\text{ region in } w \text{ is of even length} \}.$
Checking Consecutive Characters

\[ L = \{ w \in \{a, b\}^* : \text{no two consecutive characters are the same}\}. \]
Checking Consecutive Characters

\[ L = \{ w \in \{a, b\}^* : \text{no two consecutive characters are the same} \}. \]
Dead States

\[ L = \{ w \in \{ a, b \}^* : \text{every } a \text{ region in } w \text{ is of even length} \} \]
Dead States

$L = \{ w \in \{a, b\}^* : \text{every } a \text{ region in } w \text{ is of even length} \}$
Dead States

$L = \{w \in \{a, b\}^* : \text{every } b \text{ in } w \text{ is surrounded by } a's\}$
The Language of Floating Point Numbers is Regular

Example strings:
+3.0, 3.0, 0.3E1, 0.3E+1, -0.3E+1, -3E8

The language is accepted by the DFSM:
A Simple Communication Protocol
Controlling a Soccer-Playing Robot
A Simple Controller

Walking To Unseen Ball
- The Global Map doesn’t know where the ball is.

Turning For Ball
- The robot has turned at least 360 without seeing the ball, and the Global Map knows where the ball is.

Walking To Seen Ball
- Enough time has elapsed without finding the ball.

Turning For Ball
- Ball is seen.

Chin Pinch Turn
- Ball kicked to side or backwards.

Head Scan For Ball
- Ball is true and Chin Pinch is executed.

Near ball is true and Chin Pinch is executed.
- Ball was kicked forwards.

Kicking
- DetermineAndSetKick says Chin Pinch Turn is not necessary.

Robot is facing desired direction.
- Kick is finished.

Recover From Kick
- Ball was kicked forwards.
Programming FSMs

Cluster strings that share a “future”.

Let $L = \{w \in \{a, b\}^* : w$ contains an even number of a’s and an odd number of b’s$\}$
Even $a$’s Odd $b$’s
Vowels in Alphabetical Order

\[ L = \{ w \in \{a-z\}^* : \text{all five vowels, } a, e, i, o, \text{ and } u, \text{ occur in } w \text{ in alphabetical order} \}. \]
Vowels in Alphabetical Order

\[ L = \{ w \in \{a - z\}^* : \text{all five vowels, } a, e, i, o, \text{ and } u, \text{ occur in } w \text{ in alphabetical order} \}. \]
Programming FSMs

\[ L = \{ w \in \{a, b\}^* : w \text{ does not contain the substring } aab \} . \]
Programming FSMs

$L = \{w \in \{a, b\}^* : w \text{ does not contain the substring } aab\}$.

Start with a machine for $\neg L$:

How must it be changed?
A Building Security System

$L = \{\text{event sequences such that the alarm should sound}\}$
FSMs Predate Computers

The Prague Orloj, originally built in 1410.
The Abacus
The Jacquard Loom

Invented in 1801.
Finite State Representations in Software Engineering

A high-level state chart model of a digital watch.
Making the Model Hierarchical

Diagram:

- **alarm-beeping**
  - alarm 1
  - alarm 2
- **any button pushed**
- **displaying**
  - set button pushed
    - done button pushed
- **setting**
  - a_1
  - a_2
The Missing Letter Language

Let $\Sigma = \{a, b, c, d\}$.

Let $L_{\text{Missing}} =$
\[ \{w : \text{there is a symbol } a_i \in \Sigma \text{ not appearing in } w\} \].

Try to make a DFSM for $L_{\text{Missing}}$. 
Definition of an NDFSM

\[ M = (K, \Sigma, \Delta, s, A), \]  where:

- \( K \) is a finite set of states
- \( \Sigma \) is an alphabet
- \( s \in K \) is the initial state
- \( A \subseteq K \) is the set of accepting states, and
- \( \Delta \) is the transition relation. It is a finite subset of

\[ (K \times (\Sigma \cup \{\varepsilon\})) \times K \]
Accepting by an NDFSM

$M$ accepts a string $w$ iff there exists some path along which $w$ drives $M$ to some element of $A$.

The language accepted by $M$, denoted $L(M)$, is the set of all strings accepted by $M$. 
Sources of Nondeterminism
Analyzing Nondeterministic FSMs

Two approaches:

• Explore a search tree:

• Follow all paths in parallel
Optional Substrings

$L = \{ w \in \{a, b\}^* : w \text{ is made up of an optional } a \text{ followed by } aa \text{ followed by zero or more } b \text{'s} \}.$
Multiple Sublanguages

\[ L = \{w \in \{a, b\}^* : w = aba \text{ or } |w| \text{ is even}\}. \]
The Missing Letter Language

Let $\Sigma = \{a, b, c, d\}$. Let $L_{\text{Missing}} = \{w : \text{there is a symbol } a_i \in \Sigma \text{ not appearing in } w\}$
The Missing Letter Language
Pattern Matching

\[ L = \{ w \in \{a, b, c\}^* : \exists x, y \in \{a, b, c\}^* (w = x \text{abcabb} y) \}. \]

A DFSM:
Pattern Matching

\[ L = \{w \in \{a, b, c\}^* : \exists x, y \in \{a, b, c\}^* (w = x \text{abcabb} y)\}. \]

An NDFSM:
Pattern Matching with NDFSMs

\[ L = \{ w \in \{a, b\}^* : \exists x, y \in \{a, b\}^* : w = x \text{ aabbb } y \text{ or } w = x \text{ abbab } y \} \]
Multiple Keywords

\[ L = \{ w \in \{a, b\}^* : \exists x, y \in \{a, b\}^* (w = xabbaay \lor w = xbabay) \}. \]
Checking from the End

\[ L = \{ w \in \{a, b\}^* : \text{the fourth to the last character is } a \} \]
Checking from the End

$L = \{ w \in \{a, b\}^* : \text{the fourth to the last character is } a \}$
Another Pattern Matching Example

\[ L = \{ w \in \{0, 1\}^* : w \text{ is the binary encoding of a positive integer that is divisible by } 16 \text{ or is odd} \} \]
Another NDFSM

\( L_1 = \{ w \in \{a, b\}^*: \text{aa occurs in } w \} \)

\( L_2 = \{ x \in \{a, b\}^*: \text{bb occurs in } x \} \)

\( L_3 = \{ y : \in L_1 \text{ or } L_2 \} \)

\( M_1 = \)

\( M_2 = \)

\( M_3 = \)
A “Real” Example

Diagram:
- **Hiding**
  - Found by enemy
  - Found by enemy or Coast clear
  - Enemy killed by brother
  - Dead
    - Get stabbed
    - Enemy dies
    - Enemy dies
    - Become king
- **Running**
  - See enemy
  - See nothing
  - See sword
  - See laser
  - Pick up laser
    - Laser picked up
  - Swing Sword
    - Sword picked up
  - Reach for Sword
Analyzing Nondeterministic FSMs

Does this FSM accept:

\text{baaba}

Remember: we just have to find one accepting path.
Analyzing Nondeterministic FSMs

Two approaches:

• Explore a search tree:

• Follow all paths in parallel
Another Nondeterministic Example

\[ b^* (b(a \cup c)c \cup b(a \cup b) (c \cup \varepsilon))^* b \]
Dealing with $\varepsilon$ Transitions

$\text{eps}(q) = \{ p \in K : (q, w) |-^*_{M} (p, w) \}$. 

$\text{eps}(q)$ is the closure of $\{q\}$ under the relation $\{(p, r) : \text{there is a transition } (p, \varepsilon, r) \in \Delta \}$. 

How shall we compute $\text{eps}(q)$?
An Algorithm to Compute $\text{eps}(q)$

$\text{eps}(q: \text{state}) =$

\[\text{result} = \{q\}.\]

While there exists some $p \in \text{result}$ and some $r \not\in \text{result}$ and some transition $(p, \varepsilon, r) \in \Delta$ do:

Insert $r$ into $\text{result}$.

Return $\text{result}$. 
An Example of $\epsilon$

$\epsilon(q_0) =$
$\epsilon(q_1) =$
$\epsilon(q_2) =$
$\epsilon(q_3) =$
Simulating a NDFSM

\[ ndfsmsimulate(M: \text{NDFSM}, w: \text{string}) = \]
\begin{enumerate}
    \item \textit{current-state} = \text{eps(s)}.
    \item While any input symbols in \( w \) remain to be read do:
        \begin{enumerate}
            \item \textit{c} = \text{get-next-symbol}(w).
            \item \textit{next-state} = \emptyset.
            \item For each state \( q \) in \textit{current-state} do:
                \begin{enumerate}
                    \item For each state \( p \) such that \((q, c, p) \in \Delta\) do:
                        \textit{next-state} = \textit{next-state} \cup \text{eps}(p).
                \end{enumerate}
            \item \textit{current-state} = \textit{next-state}.
        \end{enumerate}
    \item If \textit{current-state} contains any states in \( A \), accept. Else reject.
\end{enumerate}
Nondeterministic and Deterministic FSMs

Clearly: \( \{ \text{Languages accepted by a DFSM} \} \subseteq \{ \text{Languages accepted by a NDFSM} \} \)

More interestingly:

**Theorem:**

For each NDFSM, there is an equivalent DFSM.
Nondeterministic and Deterministic FSMs

**Theorem:** For each NDFSM, there is an equivalent DFSM.

**Proof:** By construction:

Given a NDFSM \( M = (K, \Sigma, \Delta, s, A) \), we construct \( M' = (K', \Sigma, \delta', s', A') \), where

\[
K' = P(K) \\
s' = \text{eps}(s) \\
A' = \{Q \subseteq K : Q \cap A \neq \emptyset\} \\
\delta'(Q, a) = \bigcup \{\text{eps}(p) : p \in K \text{ and } (q, a, p) \in \Delta \text{ for some } q \in Q\}
\]
An Algorithm for Constructing the Deterministic FSM

1. Compute the $\epsilon(q)'s$.

2. Compute $s' = \epsilon(s)$.

3. Compute $\delta'$.

4. Compute $K' = \text{a subset of } P(K)$.

5. Compute $A' = \{Q \in K': Q \cap A \neq \emptyset\}$. 
The Algorithm \textit{ndfsmtodfsm}

\textit{ndfsmtodfsm}(M: NDFSM) =

1. For each state \(q\) in \(K_M\) do:
   1.1 Compute \(\text{eps}(q)\).
2. \(s' = \text{eps}(s)\)
3. Compute \(\delta'\):
   3.1 \textit{active-states} = \{s'\}.
   3.2 \(\delta' = \emptyset\).
   3.3 While there exists some element \(Q\) of \textit{active-states} for which \(\delta'\) has not yet been computed do:
      For each character \(c\) in \(\Sigma_M\) do:
         \(\text{new-state} = \emptyset\).
      For each state \(q\) in \(Q\) do:
         For each state \(p\) such that \((q, c, p) \in \Delta\) do:
            \(\text{new-state} = \text{new-state} \cup \text{eps}(p)\).
            Add the transition \((Q, c, \text{new-state})\) to \(\delta'\).
            If \(\text{new-state} \not\in \text{active-states}\) then insert it.
4. \(K' = \text{active-states}\).
5. \(A' = \{Q \in K' : Q \cap A \neq \emptyset\}\).
An Example
The Number of States May Grow Exponentially

$|\Sigma| = n$

- No. of states after 0 chars: $1$
- No. of new states after 1 char: $\binom{n}{n-1} = n$
- No. of new states after 2 chars: $\binom{n}{n-2} = \frac{n(n-1)}{2}$
- No. of new states after 3 chars: $\binom{n}{n-3} = \frac{n(n-1)(n-2)}{6}$

Total number of states after $n$ chars: $2^n$
Another Hard Example

\[ L = \{ w \in \{a, b\}^* : \text{the fourth to the last character is } a \} \]
If the Original FSM is Deterministic

1. Compute the $\epsilon(q)$s:
2. $s' = \epsilon(q_0) =$
3. Compute $\delta'$
   - $\delta'(\{q_0\}, \text{odd}, \{q_1\})$
   - $\delta'(\{q_1\}, \text{odd}, \{q_1\})$
4. $K' = \{q_0\}, \{q_1\}$
5. $A' = \{q_1\}$

$M' = M$
The Real Meaning of “Determinism”

Let $M =$

Is $M$ deterministic?

An FSM is deterministic, in the most general definition of determinism, if, for each input and state, there is at most one possible transition.

- DFSMs are always deterministic. Why?
- NDFSMs can be deterministic (even with $\varepsilon$-transitions and implicit dead states), but the formalism allows nondeterminism, in general.
- Determinism implies uniquely defined machine behavior.
Deterministic FSMs as Algorithms

$L = \{w \in \{a, b\}^* : w \text{ contains no more than one } b\}$:
Deterministic FSMs as Algorithms

until accept or reject do:
  S:  \( s = \text{get-next-symbol} \)
      if \( s = \text{end-of-file} \) then accept
      else if \( s = a \) then go to \( S \)
      else if \( s = b \) then go to \( T \)
  T:  \( s = \text{get-next-symbol} \)
      if \( s = \text{end-of-file} \) then accept
      else if \( s = a \) then go to \( T \)
      else if \( s = b \) then reject
      end
Deterministic FSMs as Algorithms

until accept or reject do:
  S:  \( s = \text{get-next-symbol} \)
      if \( s = \text{end-of-file} \) then accept
      else if \( s = a \) then go to \( S \)
      else if \( s = b \) then go to \( T \)
  T:  \( s = \text{get-next-symbol} \)
       if \( s = \text{end-of-file} \) then accept
       else if \( s = a \) then go to \( T \)
       else if \( s = b \) then reject
end

Length of Program: \(|K| \times (|\Sigma| + 2)\)
Time required to analyze string \( w \): \( \mathcal{O}(|w| \times |\Sigma|) \)

We have to write new code for every new FSM.
A Deterministic FSM Interpreter

dfsmsimulate(M: DFSM, w: string) =
1. \( st = s. \)
2. Repeat
   2.1 \( c = \text{get-next-symbol}(w). \)
   2.2 If \( c \neq \text{end-of-file} \) then
      2.2.1 \( st = \delta(st, c). \)
      until \( c = \text{end-of-file}. \)
3. If \( st \in A \) then accept else reject.

Input: aabaa
Nondeterministic FSMs as Algorithms

Real computers are deterministic, so we have three choices if we want to execute a NDFSM:

1. Convert the NDFSM to a deterministic one:
   • Conversion can take time and space $2^{|K|}$.
   • Time to analyze string $w$: $O(|w|)$

2. Simulate the behavior of the nondeterministic one by constructing sets of states "on the fly" during execution
   • No conversion cost
   • Time to analyze string $w$: $O(|w| \times |K|^2)$

3. Do a depth-first search of all paths through the nondeterministic machine.
A NDFSM Interpreter

\[\text{ndfsmsimulate}(M = (K, \Sigma, \Delta, s, A) : \text{NDFSM}, \ w : \text{string}) =\]

1. Declare the set \( st \).
2. Declare the set \( st1 \).
3. \( st = \text{eps}(s) \).
4. Repeat
   4.1 \( c = \text{get-next-symbol}(w) \).
   4.2 If \( c \neq \text{end-of-file} \) then do
      4.2.1 \( st1 = \emptyset \).
      4.2.2 For all \( q \in st \) do
         4.2.2.1 For all \( r \in \Delta(q, c) \) do
            4.2.2.1.1 \( st1 = st1 \cup \text{eps}(r) \).
      4.2.3 \( st = st1 \).
   4.2.4 If \( st = \emptyset \) then exit.
   until \( c = \text{end-of-file} \).
5. If \( st \cap A \neq \emptyset \) then accept else reject.