Nondeterministic Finite State Machines

Nondeterminism

Imagine adding to a programming language the function \textit{choice} in either of the following forms:

1. \textit{choose} (action 1;;
   action 2;;
   ...
   action \( n \))

2. \textit{choose}(x from S: \( P(x) \))
before the first choice \textit{choose} makes

\begin{itemize}
\item first call to \textit{choose} choice 1
\item first call to \textit{choose} choice 2
\item second call to \textit{choose} choice 1
\item second call to \textit{choose} choice 2
\end{itemize}

\section*{Implementing Nondeterminism}

\section*{Nondeterminism}

- What it means\slash implies
  - We could guess and our guesses would lead us to the answer correctly (if there is an answer).
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

\[
(K \times (\Sigma \cup \{\varepsilon\}) \times K) \times K
\]

\times Cartesian product

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 \( M \).
NDFSM and DFSM

$\Delta$ is the transition relation. It is a finite subset of

$$(K \times (\Sigma \cup \{\varepsilon\})) \times K$$

Recall the definition of 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 $K$

NDFSM and DFSM

$\Delta : (K \times (\Sigma \cup \{\varepsilon\})) \times K$

$\delta : (K \times \Sigma)$ to $K$

Key difference:

- In every configuration, a DFSM can make exactly one move; this is not true for NDFSM

- $M$ may enter a config. from which two or more competing moves are possible. This is due to (1) $\varepsilon$-transition (2) relation, not function
Sources of Nondeterminism

What differ from determinism?

Two approaches:
• Explore a ε
• Follow all paths in parallel

Analyzing Nondeterministic FSMs
Optional Substrings

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

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} q_3 \]

\[ q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{a} \]
Optional Substrings

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

$M = (K, \Sigma, \Delta, s, A) = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \Delta, q_0, \{q_3\})$

where

$\Delta = \{((q_0, a), q_1), ((q_0, \epsilon), q_1), ((q_1, a), q_2), ((q_2, a), q_3), ((q_3, b), q_3)\}$

How many elements does $\Delta$ have?

Multiple Sublanguages

$L = \{ w \in \{a, b\}^* : w = aba \text{ or } |w| \text{ is even}\}.$
Multiple Sublanguages

\[ L = \{ w \in \{a, b\}^* : w = aba \text{ or } |w| \text{ is even}\}. \]
Multiple Sublanguages

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

\[
M = (K, \Sigma, \Delta, s, A) = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{a, b\}, \Delta, q_0, \{q_4, q_5\}), \text{ where} \\
\Delta = \{(q_0, \varepsilon, q_1), (q_0, \varepsilon, q_5), \\
(q_1, a, q_2), (q_2, b, q_3), (q_3, a, q_4), \\
(q_5, a, q_6), (q_5, b, q_6), (q_6, a, q_5), (q_6, b, q_5)\}
\]

Do you start to feel the power of nondeterminism?

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 \( L_{\text{missing}} \)

First develop machine for set of strings containing ALL 4 characters, then reverse.
The Missing Letter Language

$$L = \{w \in \{a, b, c\}^* : \exists x, y \in \{a, b, c\}^* (w = xabcabb y)\}.$$

Pattern Matching

A DFSM:
Pattern Matching

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

A DFSM:

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

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

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

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

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

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

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

$L_4 = L_1 L_2$

$M_1 = \quad M_2 = \quad$

$M_3 = \quad$

$M_4 = \quad$

A “Reel” Example

[Diagram showing a state transition diagram with states such as Hiding, Running, Pick up laser, and outcomes like see enemy, see laser, enemy dies, etc.]
**ε Transitions – eps function**

\[ \text{eps}(q) = \{ p \in K : (q, w) \vdash^*_{M} (p, w) \}. \]

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

How shall we compute \( \text{eps}(q) \)?

It simply means the states reachable without consuming input.

---

**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 \( r \notin \text{result} \) and some transition \( (p, \varepsilon, r) \in \Delta \) do:

Insert \( r \) into \( \text{result} \).

Return \( \text{result} \).
An Example of $\varepsilon$

\begin{align*}
\varepsilon(q_0) &= \\
\varepsilon(q_1) &= \\
\varepsilon(q_2) &= \\
\varepsilon(q_3) &= 
\end{align*}

Simulating a NDFSM

$$\text{ndfsmsimulate}(M: \text{NDFSM}, w: \text{string}) =$$

1. current-state = $\varepsilon(s)$.
2. While any input symbols in $w$ remain to be read do:
   1. $c = \text{get-next-symbol}(w)$.
   2. next-state = $\emptyset$.
   3. For each state $q$ in current-state do:
      For each state $p$ such that $(q, c, p) \in \Delta$ do:
         next-state = next-state $\cup$ $\varepsilon(p)$.
   4. current-state = next-state.
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.

**Proof:** By construction:

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

- \( K' = \mathcal{P}(K) \)
- \( s' = \text{eps}(s) \)
- \( A' = \{ Q \subseteq K : Q \cap A \neq \emptyset \} \)
- \( \delta'(Q, a) = \bigcup \{ \text{eps}(p) : p \in K \text{ and } (q, a, p) \in \Delta \text{ for some } q \in Q \} \)

May create many unreachable states
An Algorithm for Constructing the Deterministic FSM

1. Compute the $\text{eps}(q)$’s.
2. Compute $s' = \text{eps}(s)$.
3. Compute $\delta'$.
4. Compute $K' = \text{a subset of } \mathcal{P}(K)$.
5. Compute $A' = \{Q \in K': Q \cap A \neq \emptyset\}$.

The algorithm
1. Proves NDFSM $\equiv$ DFSM
2. Allows us to solve problems using NDFSM then construct equivalent DFSM

The Algorithm $\text{ndfsmtodfsm}$

$\text{ndfsmtodfsm}(M: \text{NDFSM}) =$

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 active-states $= \{s\}$.
   3.2 $\delta' = \emptyset$.
   3.3 While there exists some element $Q$ of active-states for which $\delta'$ has not yet been computed do:
      For each character $c$ in $\Sigma_M$ do:
         new-state $= \emptyset$.
         For each state $q$ in $Q$ do:
            For each state $p$ such that $(q, c, p) \in \Delta$ do:
               new-state $= \text{new-state } \cup \text{eps}(p)$.
            Add the transition $(Q, c, \text{new-state})$ to $\delta'$.
            If new-state $\notin$ active-states then insert it.
4. $K' = \text{active-states}$.
5. $A' = \{Q \in K': Q \cap A \neq \emptyset\}$. 
For each of the following NDFSMs, use ndfsmtodfsm to construct an equivalent DFSM. Begin by showing the value of \( eps(q) \) for each state \( q \):

(a)

(b)
For each of the following NDFSMs, use ndfsmtodfsm to construct an equivalent DFSM. Begin by showing the value of $\text{eps}(q)$ for each state $q$. 

Another Example
An Example – Optional Substrings

\[ L = \{ w \in \{a, b\}^* : w \text{ is made up of 0 to 2 } a\text{'s followed by zero or more } b\text{'s} \}. \]

\[
q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} q_2
\]

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: \(\binom{n}{n-2} = n(n-1)/2\)
No. of new states after 3 chars: \(\binom{n}{n-3} = n(n-1)(n-2)/6\)
Total number of states after \(n\) chars: \(2^n\)
If the Original FSM is Deterministic

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

\( M' = M \)

A Deterministic FSM Interpreter

\( \text{dfsmsimulate}(M: \text{DFSM}, w: \text{string}) = \)
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.
A NDFSM Interpreter

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

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
      \( st1 = \emptyset \).
      For all \( q \in st \) do
         For all \( r \in \Delta(q, c) \) do
            \( st1 = st1 \cup \text{eps}(r) \).
      \( st = st1 \).
      If \( st = \emptyset \) then exit.
   until \( c = \text{end-of-file} \).
5. If \( st \cap A \neq \emptyset \) then accept else reject.

Nondeterministic FSMs as Algorithms

Real computers are deterministic, so we have three choices if we want to execute an 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.
Finite State Transducers

- A finite state transducer (FST) is a finite state machine, that transduces (translates) an input string into an output string.
  - instead of \{0,1\} as in FSMs (acceptors / recognizers)
  - input tape, output tape
  - Moore machine and Mealy machine

- Moore machine: outputs are determined by the current state alone (and do not depend directly on the input)
  - Advantage of the Moore model is a simplification of the behavior

- Mealy machine: output depends on current state and input
**Moore and Mealy**

**Edward F. Moore** (1925 – 2003)
- Professor of Math and CS in UW-madison
- [Memorial resolution](http://boards.ancestry.com/surnames.mealy/56.1.1/mb.ashx)

**George H. Mealy** (1927 – 2010)
worked at the Bell Laboratories in 1950’s and was a Harvard University professor in 1970’s

http://boards.ancestry.com/surnames.mealy/56.1.1/mb.ashx

---

**Moore Machine**

A *Moore machine* $M = (K, \Sigma, O, \delta, D, s, A)$, where:

- $K$ is a finite set of states
- $\Sigma$ is an input alphabet
- $O$ is an output alphabet
- $s \in K$ is the initial state
- $A \subseteq K$ is the set of accepting states, *(not important for some app.)*
- $\delta$ is the transition function from $(K \times \Sigma)$ to $K$,
- $D$ is the output function from $K$ to $O^*$.

$M$ outputs each time it lands in a state.

A Moore machine $M$ computes a function $f(w)$ iff, when it reads the input string $w$, its output sequence is $f(w)$. 
A Simple US Traffic Light Controller

A Mealy machine $M = (K, \Sigma, O, \delta, s, A)$, where:

- $K$ is a finite set of states
- $\Sigma$ is an input alphabet
- $O$ is an output alphabet
- $s \in K$ is the initial state
- $A \subseteq K$ is the set of accepting states (not important for some app.)
- $\delta$ is the transition function from $(K \times \Sigma)$ to $(K \times O^*)$

$M$ outputs each time it takes a transition.

A Mealy machine $M$ computes a function $f(w)$ iff, when it reads the input string $w$, its output sequence is $f(w)$. 
An Odd Parity Generator

After every four bits, output a fifth bit such that each group of five bits has odd parity.

\[0 \ 0 \ 0 \ 0 \quad 1 \ 0 \ 0 \ 0 \quad 1 \ 1 \ 1 \ 1\]

A Bar Code Scanner
A Bar Code Scanner