Deterministic Finite State Automata

Sipser pages 31-46
Deterministic Finite Automata (DFA)

- DFAs are easiest to present pictorially:

They are directed graphs whose nodes are states and whose arcs are labeled by one or more symbols from some alphabet $\Sigma$. Here $\Sigma$ is $\{0, 1\}$.

Such a graph is called a state transition diagram.
• One state is *initial* (denoted by a short incoming arrow), and several are *final/accepting* (denoted by a double circle). For every symbol $a \in \Sigma$ there is an arc labeled $a$ emanating from every state.

• Automata are string processing devices. The arc from $q_1$ to $q_2$ labeled 0 shows that when the automaton is in the state $q_1$ and receives the input symbol 0, its next state will be $q_2$. 
Drawing Conventions

• I use some software to draw DFAs, which is somewhat limited. So I use conventions
  1. Initial states are green circles
  2. Final states are double red circles
  3. Other states are oval
  4. If the initial and final states overlap, I use blue double circle.
Missing alphabet

• I sometimes draw a state transition diagram where some nodes do not have an edge labeled with every letter of the alphabet, by convention we add a new (dead) state where all missing edges terminate.
• Every path in the graph spells out a string over $S$. Moreover, for every string $w \in \Sigma^*$ there is a unique path in the graph labelled $w$. (Every string can be processed.) The set of all strings whose corresponding paths end in a final state is the *language of the automaton*.

![Diagram of a DFA]

• In our example, the language of the automaton consists of strings over $\{0, 1\}$ containing at least two occurrences of 0.
• Modify the automaton so that its language consists of strings containing *exactly two* occurrences of 0.
A DFA is a quintuple $A = (Q, \Sigma, \delta, q_0, F)$ where

- $Q$ is a set of states
- $\Sigma$ is the alphabet (of input symbols)
- $\delta: Q \times \Sigma \rightarrow Q$ is the transition function
- $q_0 \in Q$ -- the start state
- $F \subseteq Q$ -- final states

- Page 35 of Sipser
Example

- In our example,
- $Q = \{ q_0, q_1, q_2 \}$,
- $\Sigma = \{ 0, 1 \}$,
- $q_0 = q_0$,
- $F = \{ q_2 \}$,
- and

$\delta$ is given by 6 equalities

- $\delta(q_0, 0) = q_1$,
- $\delta(q_0, 1) = q_0$,
- $\delta(q_2, 1) = q_2$
- ...

Diagram:

- $q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_2 \xrightarrow{0,1}$
All the information presenting a DFA can be given by a single thing -- its *transition table*:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_0$</td>
<td>$Q_1$</td>
<td>$Q_0$</td>
</tr>
<tr>
<td>$Q_1$</td>
<td>$Q_2$</td>
<td>$Q_1$</td>
</tr>
<tr>
<td>$*Q_2$</td>
<td>$Q_2$</td>
<td>$Q_2$</td>
</tr>
</tbody>
</table>

The initial and final states are denoted by $\rightarrow$ and * respectively.
Language of accepted Strings

• A DFA \( (Q, \Sigma, \delta, q_0, F) \), accepts a string
  \[ w = \text{"w}_1\text{w}_1...\text{w}_n" \] iff

  – There exists a sequence of states \([r_0, r_1, ... r_n]\)
    with 3 conditions

  1. \( r_0 = q_0 \)
  2. \( \delta(r_i, w_{i+1}) = r_{i+1} \)
  3. \( r_n \in F \)

Acceptance is about finding a sequence.
How do we find such a sequence?
Example

• Show that “ABAB” is accepted.

• Here is a path $[0,0,1,2,2]$
  – The first node, 0, is the start state.
  – The last node, 2, is in the accepting states
  – The path is consistent with the transition
    • $\delta 0 \text{ A} = 0$
    • $\delta 0 \text{ B} = 1$
    • $\delta 1 \text{ A} = 2$
    • $\delta 2 \text{ B} = 2$

Note that the path is one longer than the string
Definition of Regular Languages

• A language is called regular if it is accepted by some DFA.
Extension of $\delta$ to Strings

- Given a state $q$ and a string $w$, there is a unique path labeled $w$ that starts at $q$ (why?). The endpoint of that path is denoted $\delta(q,w)$.

- Formally, the function $\delta : Q \times \Sigma^* \rightarrow Q$
- is defined recursively:

  - $\delta(q,\epsilon) = q$
  - $\delta(q,x:xs) = \delta(\delta(q,x),xs)$

- Note that $\delta(q,\text{"a"}) = \delta(q,a)$ for every $a \in \Sigma$;
- so $\delta$ does extend $\delta$. 

Example trace

- Diagrams (when available) make it very easy to compute $\delta(q, w)$ --- just trace the path labeled $w$ starting at $q$.

- E.g. trace 101 on the diagram below starting at $q_1$. 

![Diagram showing states $q_0$, $q_1$, and $q_2$ with transitions labeled 0 and 1, and an accepting state $q_2$ labeled 0,1.]
Implementation and precise arguments need the formal definition.

$$\delta(q_1, 101) = \delta(\delta(q_1, 1), 01)$$

$$= \delta(q_1, 01)$$

$$= \delta(\delta(q_1, 0), 1)$$

$$= \delta(q_2, 1)$$

$$= \delta(\delta(q_2, 0), \varepsilon)$$

$$= \delta(q_2, \varepsilon)$$

$$= q_2$$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rightarrow q_0$</td>
<td>$q_1$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_2$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$*q_2$</td>
<td>$q_2$</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>
A DFA \( (Q, \Sigma, \delta, q_0, F) \) accepts a string \( w \) iff \( \delta(q_0, w) \in F \)

The language of the automaton \( A \) is
\[
L(A) = \{ w \mid A \text{ accepts } w \}.
\]

More formally
\[
L(A) = \{ w \mid \delta(\text{Start}(A), w) \in \text{Final}(A) \}
\]

Example:
Find a DFA whose language is the set of all strings over \( \{a, b, c\} \) that contain \( \text{aaa} \) as a substring.
DFA’s as Programs

data DFA q s =

    DFA [q]               -- states
    [s]                   -- symbols
    (q -> s -> q)         -- delta
    q                     -- start state
    [q]                   -- accept states

Note that the States and Symbols can be any type.
Programming for acceptance 1

```haskell
path:: Eq q => DFA q s -> q -> [s] -> [q]
path d q [] = [q]
path d q (s:ss) = q : path d (trans d q s) ss

acceptDFA1 :: Eq a => DFA a t -> [t] -> Bool
acceptDFA1 dfa w = cond1 p && cond2 p && cond3 w p
    where p = path dfa (start dfa) w

cond1 (r:rs) = (start dfa) == r
cond1 [] = False

cond2 [r] = elem r (accept dfa)
cond2 (r:rs) = cond2 rs
cond2 _ = False

cond3 [] [r] = True
cond3 (w:ws) (r1:(more@(r2:rs))) =
    (trans dfa r1 w == r2) && (cond3 ws more)
cond3 _ _ _ = False
```

Iff there exists a sequence of states 
\[ [r_0, r_1, \ldots, r_n] \]

1. \( r_0 = q_0 \)
2. \( \delta(r_i, w_{i+1}) = r_{i+1} \)
3. \( r_n \in F \)
Programming for acceptance 1

-- \( \delta = \text{deltaBar} \)

deltaBar :: Eq q => DFA q s -> q -> [s] -> q
deltaBar dfa q [] = q
deltaBar dfa q (s:ss) =
    deltaBar dfa (trans dfa q s) ss

acceptDFA2 dfa w =
    elem (deltaBar dfa (start dfa) w)
    (accept dfa)
d1 :: DFA Integer Integer

d1 = DFA states symbol trans start final

where states = [0,1,2]
    symbol = [0,1]
    trans p a = (2*p+a) `mod` 3
    start = 0
    final = [2]
\[ d1 = \text{DFA states symbol trans start final} \]
\[
\text{where states} = [0,1,2] \\
\text{symbol} = [0,1] \\
\text{trans } p \ a = (2*p+a) \mod 3 \\
\text{start} = 0 \\
\text{final} = [2] \\
\]