Deterministic Finite State Automata

Sipser pages 31-46
Formal Definition

- 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,
- \( \mathcal{Q}=\{q_0, q_1, q_2\} \),
- \( \Sigma=\{0, 1\} \),
- \( q_0=q_0 \),
- \( F=\{q_2\} \),
- and

\( \delta \) is given by 6 equalities

- \( \delta(q_0, 0)=q_1 \),
- \( \delta(q_0, 1)=q_0 \),
- \( \delta(q_2, 1)=q_2 \)
- ...
• 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, Σ, δ, q₀, F), accepts a string

• w = “w₁w₂...wₙ” iff

– There exists a sequence of states [r₀, r₁, ... rₙ] with 3 conditions

1. r₀ = q₀
2. δ(rᵢ, wᵢ₊₁) = rᵢ₊₁
3. rₙ₊₁ ∈ F

Page 40 in Sisper

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:
    • \(\delta 0 A = 0\)
    • \(\delta 0 B = 1\)
    • \(\delta 1 A = 2\)
    • \(\delta 2 B = 2\)

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

• A language is called regular if some finite automaton accepts (i.e. a DFA accepts it)

• Page 40 in Sipser
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:
    
    $\delta(q, \varepsilon) = q$
    $\delta(q, x:xs) = \delta(\delta(q, x), xs)$

• Note that $\delta(q, "a") = \delta(q, a)$ 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 $q_1$. 

\begin{figure}[h]
\centering
\begin{tikzpicture}
% Diagram code here
\end{tikzpicture}
\end{figure}
Implementation and precise arguments need the formal definition.

\[
\delta(q_1, 101) = \delta(\delta(q_1, 1), 01) \\
= \delta(q_1, 01) \\
= \delta(\delta(q_1, 0), 1) \\
= \delta(q_2, 1) \\
= \delta(\delta(q_2, 0), \varepsilon) \\
= \delta(q_2, \varepsilon) \\
= q_2
\]

\[
\delta(q, \varepsilon) = q \\
\delta(q, x:xs) = \delta(\delta(q, x), xs)
\]
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 \texttt{aaa} as a substring.
DFA’s as data structures

```haskell
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

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
Programming for acceptance 2

-- δ = deltaBar

\[
\delta = \deltaBar
\]

\[
\deltaBar :: Eq q \Rightarrow DFA q s \rightarrow q \rightarrow [s] \rightarrow q
\]

\[
\deltaBar dfa q [] = q
\]

\[
\deltaBar dfa q (s:ss) =
\]

\[
\deltaBar dfa (\text{trans} dfa q s) ss
\]

\[
\text{acceptDFA2} dfa w =
\]

\[
\text{elem} (\deltaBar dfa (\text{start} dfa) w)
\]

\[
(\text{accept dfa})
\]
An Example

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

• DFAs are a computation mechanism
  • They compute whether some string is in some language

• Several mechanisms can be defined that describe how they compute. All essentially trace a path through the state diagram.

• DFAs can be represented as a data structure

• Precise algorithms can be defined that implement the computation mechanisms.
Exercises

• Define a DFA for the following languages

• \{w \mid w \text{ has at least 3 a's and at least 2 b's}\}
  • \Sigma = \{a, b\}

• \{w \mid \text{length } w = 3 \}
  • \Sigma = \{m, n, p\}

• \{w \mid w \text{ represents an integer} \}
  • \Sigma = \{a, b, c, 0, 2, 1\}