2-WAY DFA

E0222 AUTOMATA THEORY AND COMPUTABILITY
26 OCT, 2012

TANUJ AGARWAL
VISHESH GARG
M.Sc.(Engg)
SERC, IISc, Bangalore
AGENDA

- Introduction
- Characterizing a 2-DFA
- Formal Description and Example
- Configuration and Acceptance
- 2-DFA v/s 1-DFA
- Information associated with 2-DFA and building Transition Tables.
- Equivalence with 1-DFA and construction of 1-DFA.
- Advantages.
- Conclusion
INTRODUCTION

- Originated by Rabin and Scott, 1959, “Finite Automata and Seminal works”.
- Generalized version of DFA.
- Process the input in either direction.
  (i) Have a read head, which may move left or right over input string.
  (ii) Revisit the characters already processed.
- Can be seen as Turing machines but
  (i) with finite memory.
  (ii) No work tape, but read only input tapes.
## 2-DFA

- **Input String:**
  1. Symbols occupying cells of finite tape.
  2. One symbol per cell.

```
|   | \( a_1 \) | \( a_2 \) | \( a_3 \) | \( a_4 \) | \( a_5 \) | \( ... \) | \( ... \) | \( a_n \) |   |
```

![Diagram](attachment:image.png)
2-DFA

- **Input String:**
  - (i) Symbols occupying cells of finite tape.
  - (ii) One symbol per cell.
2-DFA

- **Input String**:  
  (i) Symbols occupying cells of finite tape.  
  (ii) One symbol per cell.
2-DFA

- **Input String:**
  (i) Symbols occupying cells of finite tape.
  (ii) One symbol per cell.

- 2-way DFA have a finite set of states $Q$, similar to 1-way DFAs.

<table>
<thead>
<tr>
<th></th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$\ldots$</th>
<th>$\ldots$</th>
<th>$a_n$</th>
<th>$-$</th>
</tr>
</thead>
</table>

$q \in Q$, current state of the automata.
2-DFA

- **Input String:**
  (i) Symbols occupying cells of finite tape.
  (ii) One symbol per cell.
- 2-way DFA have a finite set of states $Q$, similar to 1-way DFAs.
- At any point in time, machine is in state $q$, scans an input symbol $a_i$ or an end marker and moves to its right or left entering new state $p$. 

```plaintext
| - | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | ... | ... | $a_n$ | - |
```

```
q

| - | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | ... | ... | $a_n$ | - |
```

```
p
```
2-DFA

- **Input String:**
  (i) Symbols occupying cells of finite tape.
  (ii) One symbol per cell.

- 2-way DFA have a finite set of states $Q$, similar to 1-way DFAs.

- At any point in time, machine is in state $q$, scans an input symbol $a_i$ or an end marker and moves to its right or left entering new state $p$.

- **Accept and Reject States:**
  (i) **Accept State:** Machine accepts input by entering into an accept state ‘t’.
  (ii) **Reject State:** Machine rejects input by entering into an reject state ‘r’.
2-DFA

- Input String:
  (i) Symbols occupying cells of finite tape.
  (ii) One symbol per cell.

- 2-way DFA have a finite set of states \( Q \), similar to 1-way DFAs.

- At any point in time, machine is in state \( q \), scans an input symbol \( a_i \) or an end marker and moves to its right or left entering new state \( p \).

- Accept and Reject States:
  (i) Accept State: Machine accepts input by entering into an accept state ‘t’.
  (ii) Reject State: Machine rejects input by entering into an reject state ‘r’.

- The transition function \( (\delta) \) is the action of machine on a particular state and input symbol.
FORMAL DEFINITION OF 2-DFA

2-DFA is an octuple

\[ M = (Q, \Sigma, |-, -|, \delta, s, t, r) \]

- \( Q \) : Finite set of states.
- \( \Sigma \) : The input alphabet.
- \( |-, -| \) : Left and Right end markers.
- \( \delta : Q \times (\Sigma \cup \{|-, -|\}) \rightarrow (Q \times \{L, R\}) \), Transition Function, (L,R denote movement towards left or right).
- \( s \in Q \), start state.
- \( t \in Q \), the accept state.
- \( r \in Q \), the reject state.

NOTE:
(i) A 2-way DFA needs only a single accept and reject state.
(ii) We can halt immediately as soon as DFA enters one of these states.
2-DFA is an octuple

\[ M = (Q, \Sigma, |-, -|, \delta, s, t, r) \]

Rules for the Transition function:

(i) Input is End-Marker:

\[ \delta(q, |-) = (u, R) \text{ for some } u \in Q. \]
\[ \delta(q, -|) = (v, L) \text{ for some } v \in Q. \]

(ii) Input \( b \in \Sigma \cup \{-|\} \) and for accept and reject states, \( t \) and \( r \),

\[ \delta(t, b) = (t, R) \]
\[ \delta(t, -|) = (t, L) \]
\[ \delta(r, b) = (r, R) \]
\[ \delta(r, -|) = (r, L) \]

(iii) In general,

\[ \delta(p, b) = (q, d) \quad (p, q \in Q, d \in (L,R)) \]
EXAMPLE (Constructing Normal DFA)

Consider the DFA accepting the set
\[ A = \{ x \in \{a, b\}^* \mid \#a(x) \text{ is a multiple of 3 and } \#b(x) \text{ is even} \}. \]

Constructing a normal DFA:

(i) DFA \( M_1 \), accepting \( A_1 = \{ x \in \{a, b\}^* \mid \#a(x) \text{ is a multiple of 3} \} \).
\[ M_1 = \{ Q_1, \Sigma, \delta_1, s_1, F_1 \} \]

(ii) DFA \( M_2 \), accepting \( A_2 = \{ x \in \{a, b\}^* \mid \#b(x) \text{ is even} \} \).
\[ M_1 = \{ Q_2, \Sigma, \delta_2, s_2, F_2 \} \]

(iii) DFA \( M \), accepting \( A \), such that \( M = M_1 \times M_2 \).
\[ M = \{ Q, \Sigma, \delta, s, F \} \]
EXAMPLE (Constructing 2-DFA)

Consider the 2-DFA accepting the set

\[ A = \{ x \in \{a, b\}^* \mid \#a(x) \text{ is a multiple of 3 and } \#b(x) \text{ is even} \}. \]

- Machine starts scanning input from left endmarker.
- Moves left to right, scanning all ‘a’, ignoring ‘b’
  - If \#a(x) is not multiple of 3, rejects the input string, enters ‘r’.
- If \#a(x) is multiple of 3, scans right to left, all ‘b’, ignoring ‘a’ if \#b(x) is even enters accept state ‘t’, else enters reject state ‘r’.
- It can be seen if machine enters ‘r’ in step-2 then it can immediately discard the input string.
EXAMPLE (Formal Description)

\[ Q = \{q_0, q_1, q_2, p_0, p_1, t, r\} \]
\[ \Sigma = \{a, b\} \]

Transition Function Table:

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>a</th>
<th>b</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_0</td>
<td>(q_0, R)</td>
<td>(q_1, R)</td>
<td>(q_0, R)</td>
<td>(p_0, L)</td>
</tr>
<tr>
<td>q_1</td>
<td>-</td>
<td>(q_2, R)</td>
<td>(q_1, R)</td>
<td>(r, L)</td>
</tr>
<tr>
<td>q_2</td>
<td>-</td>
<td>(q_0, R)</td>
<td>(q_2, R)</td>
<td>(r, L)</td>
</tr>
<tr>
<td>p_0</td>
<td>(t, R)</td>
<td>(p_0, L)</td>
<td>(p_1, L)</td>
<td>-</td>
</tr>
<tr>
<td>p_1</td>
<td>(r, R)</td>
<td>(p_1, L)</td>
<td>(p_0, L)</td>
<td>-</td>
</tr>
<tr>
<td>t</td>
<td>(t, R)</td>
<td>(t, R)</td>
<td>(t, R)</td>
<td>(t, L)</td>
</tr>
<tr>
<td>r</td>
<td>(r, R)</td>
<td>(r, R)</td>
<td>(r, R)</td>
<td>(r, L)</td>
</tr>
</tbody>
</table>
CONFIGURATION AND ACCEPTANCE

For an input $x \in \Sigma^*$ say $x = a_1a_2....a_n$. Let $a_0 = \|- \text{ and } a_{n+1} = \|- \text{ so that}

$$a_0a_1a_2... a_na_{n+1} = \|- x - |$$

- Configuration on machine on input $x$ is pair $(q, i)$; $q \in Q$ and $0 \leq i \leq n+1$.
i.e. , current state ‘$q$’ and the position ‘$i$’ of the read head.
e.g. Start configuration is $(s, 0)$.

- A binary relation, Configuration relation is defined on Configurations as

$$\delta(p, a) = (q, L) \Rightarrow (p, i) \xrightarrow{1} (q, i-1)$$
$$\delta(p, a) = (q, R) \Rightarrow (p, i) \xrightarrow{1} (q, i+1)$$

($^1 \rightarrow$ represent one step of machine on input $x$.)

- $(p, i) \xrightarrow{0} (p, i)$
If $(p, i) \xrightarrow{n} (q, j)$ and $(q, j) \xrightarrow{1} (u, k)$ then $(p,i) \xrightarrow{n+1} (u, k)$.

- Therefore, $(p, i) \xrightarrow{*} (q, j) \iff \text{There exists } n \geq 0, (p, i) \xrightarrow{n} (q, j)$.

- Machine accepts input if, $(s, 0) \xrightarrow{*} (t, i)$ for some $i$.

- Machine rejects input if, $(s, 0) \xrightarrow{*} (r, i)$ for some $i$. 
2-DFA v/s Normal DFA
FUNDAMENTAL QUESTION:

Is 2 way DFA more powerful than 1 way DFA?
FUNDAMENTAL QUESTION:

Is 2 way DFA more powerful than 1 way DFA?

“OR”
Does the ability to do multiple scans on the tape bestow 2 way DFA with unprecedented POWER?
FUNDAMENTAL QUESTION:

Is 2 way DFA more powerful than 1 way DFA?

“OR”

Does the ability to do multiple scans on the tape bestow 2 way DFA with unprecedented POWER?

- Not intuitively clear
- Need to develop a formal model
• Right to Left $\rightarrow$ Q
• Left to Right $\rightarrow$ P
• If next time Right to Left Q, then Left to Right will be P
  $T_x(q) = p$
• If never emerges ? Let $T_x(q) = \bot$
• The first time it emerges from x without having been to y - Call the state $T_x(\cdot)$
CONSTRUCTING A TABLE

- **NOTE**: $T_x(q)$ depends only on $x$ and $q$
- Write down $T_x(q)$ for all possible states $q$
  - $T: (Q \cup \{\cdot\}) \rightarrow (Q \cup \{\perp\})$
- Possible such tables assuming $k$ states
  - $(k+1)^{k+1}$
- Hence, a finite information can be passed

<table>
<thead>
<tr>
<th></th>
<th>$q_1$</th>
<th>$q_2$</th>
<th>...</th>
<th>$q_k$</th>
<th>$\perp$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$X$</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_2$</td>
<td></td>
<td>$X$</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>$X$</td>
<td></td>
</tr>
<tr>
<td>$q_k$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$X$</td>
</tr>
<tr>
<td>$\cdot$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$X$</td>
</tr>
</tbody>
</table>
If there are tables $T_x = T_y$ and $M$ accepts $xz$. What can be said about $yz$?

Let $L(M)$ denote the language accepted by the machine.

Define a $\equiv_{L(M)}$ relation on strings $x$ & $y$:
- If $T_x = T_y$, $x \equiv_{L(M)} y$

What are its other properties? (Recall: Right congruence, Finite Index … from previous lectures)
PROPERTIES OF $\equiv_{L(M)}$

- Right congruence
  
  - $x \equiv_{L(M)} y$ then $xa \equiv_{L(M)} ya$

- Refines $L(M)$
  
  - If $T_x = T_y$, either both $x$ and $y$ accepted or rejected

- Finite Index
  
  - $\#\text{Equivalence classes} = \#\text{Tables}$ \{Max : $(k+1)^{k+1}$ \}

Hence by Myhill-Nerode theorem, $L(M)$ is a Regular Language
IMPLICATIONS

- 2 DFA is **NO** more powerful than a normal DFA!
- We can construct a one way DFA equivalent to a 2 DFA
  - Identify equivalence classes
  - Use construction $\equiv \mapsto M_{\equiv}$
CONSTRUCTING EQUIVALENT DFA

- $Q' = \{T : (Q \cup \{\cdot\}) \xrightarrow{a} (Q \cup \{\perp\})\}$
- $S' = T_\epsilon$
- $\delta(T_x, a) = T_{xa}$
- $F' = \{T_x \mid x \in L(M)\}$
ADVANTAGES OF 2-DFA

- **More succinct than Normal DFA’s.**
  e.g. \( L(M) = \{x \in \{a,b\}^* | \#a(x) \text{ is multiple of 7 and } \#b(x) \text{ is multiple of 5}\} \).
  - Normal DFA would have 35 states.
  - 2 DFA can be constructed using 14 states, including the accept and reject states (t, r).

- **Single accept and reject states**, whereas a Normal DFA may have multiple final states.
  e.g. \( L(M) = \{x \in \{a,b\}^* | \#a(x) \text{ is multiple of 7 or } \#b(x) \text{ is multiple of 5}\} \).
  - Normal DFA has 12 final states.
Conclusion

- 2 way DFA is a convenient representation for some regular languages, much like NFA
- Converting it to a normal DFA can be tedious as it involves constructing and identifying identical tables
THANK YOU !!!
QUESTIONS ???