Deterministic Finite State Automata

Sipser pages 31-46

Formal Definition

- **A D**FA is a quintuple $\mathbf{A} = (\mathbf{Q}, \Sigma, \delta, \mathbf{q_0}, \mathbf{F})$ where
  - $\mathbf{Q}$ is a set of ***states***
  - Σ is the **alphabet** (of *input symbols*)
  - δ: $\mathbf{Q} \times \Sigma \rightarrow \mathbf{Q}$ is the ***transition function***
  - $\mathbf{q_0} \in \mathbf{Q}$ -- the ***start state***
  - $\mathbf{F} \subseteq \mathbf{Q}$ -- ***final states***

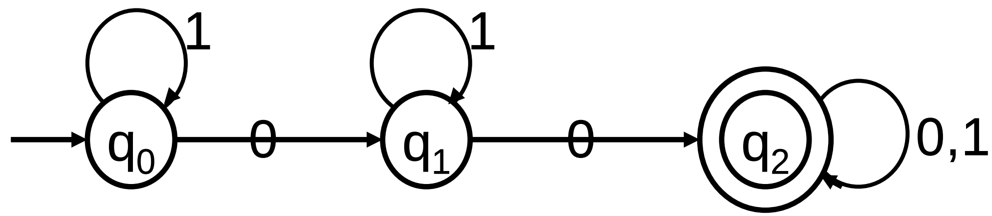
  - ***Page 35 of Sipser***

Example

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

δ is given by 6 equalities

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



Transition Table

- All the information presenting a DFA can be given by a single thing -- its *transition table*:

	0	1
Q_0	Q_1	Q_0
$\rightarrow Q_1$	Q_2	Q_1
$*Q_2$	Q_2	Q_2

- The initial and final states are denoted by $\rightarrow$ and * respectively.

Language of accepted Strings

- A DFA = $(\mathbf{Q}, \Sigma, \delta, \mathbf{q_0}, \mathbf{F})$, *accepts* a string
- $\mathbf{w}$ = "$w_1 w_2 \ldots w_n$" iff

  – There exists a sequence of states $[r_0, r_1, \ldots r_n]$ with 3 conditions

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

  Page 40 in Sisper

Acceptance is about finding a sequence.

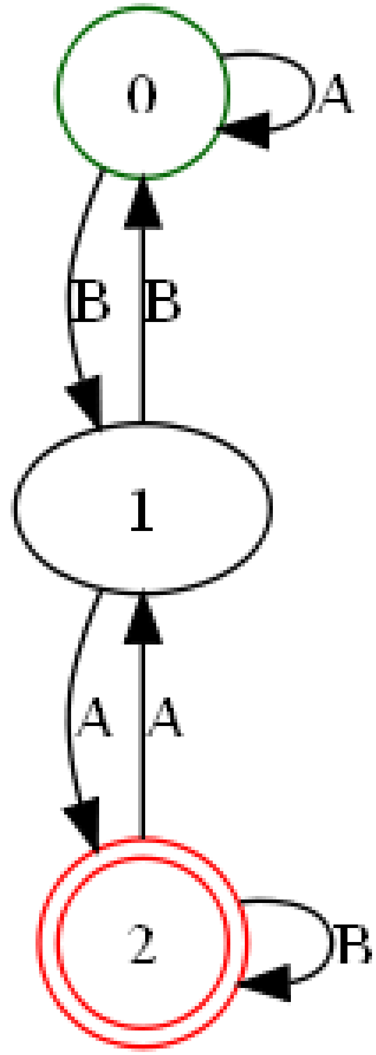
How do we find such a sequence?

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
    - **δ 0 A = 0**
    - **δ 0 B = 1**
    - **δ 1 A = 2**
    - **δ 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 δ 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 $\underline{\delta}$**(q,w)**

- Formally, the function $\underline{\delta} : Q \times \Sigma^* \to Q$
- is defined recursively:

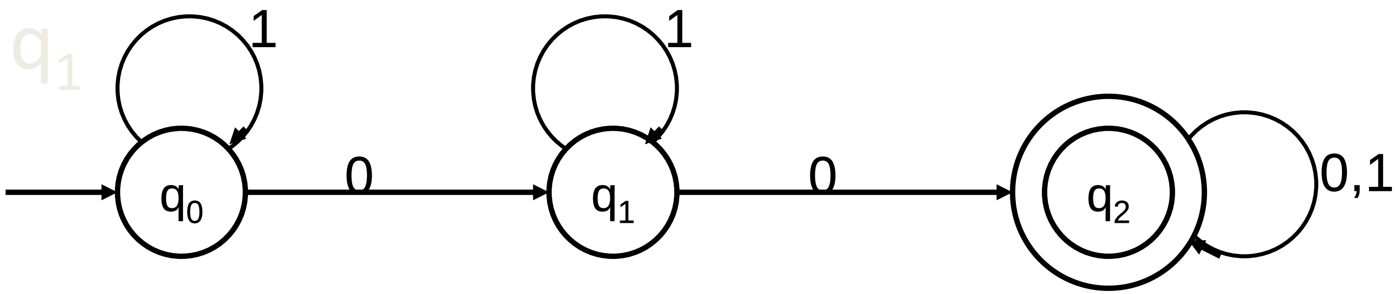
  - $\underline{\delta}(\mathtt{q},\varepsilon)=\mathtt{q}$
  - $\underline{\delta}(\mathtt{q},\mathtt{x:xs}) = \underline{\delta}(\delta(\mathtt{q},\mathtt{x}),\mathtt{xs})$

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

Example trace

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

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



Implementation and precise arguments need the formal definition.

$$\begin{aligned}
\underline{\delta}(q_1,101) &= \underline{\delta}(\ \delta(q_1,1)\ ,01) \\
&= \underline{\delta}(q_1,01) \\
&= \underline{\delta}(\ \delta(q_1,0)\ ,1) \\
&= \underline{\delta}(q_2,1) \\
&= \underline{\delta}(\ \delta(q_2,0)\ ,\varepsilon) \\
&= \underline{\delta}(q_2,\varepsilon) \\
&= q_2
\end{aligned}$$

	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_2	q_1
$*q_2$	q_2	q_2

$\underline{\delta}(q,\varepsilon)=q$

$\underline{\delta}(q,x:xs)= \underline{\delta}(\delta(q,x),xs)$

Language of accepted strings - take 2

A DFA $=(\mathbf{Q}, \Sigma, \delta, \mathbf{q_0}, \mathbf{F})$ *accepts* a string $\mathbf{w}$ iff $\underline{\delta}(\mathbf{q_0}, w) \in \mathbf{F}$

The language of the automaton A is

$L(A)=\{w \mid A \text{ accepts } w\}.$

More formally

$L(A)=\{w \mid \underline{\delta}(\mathrm{Start}(A),w) \in \mathrm{Final}(A)\}$

Example:

Find a DFA whose language is the set of all strings over $\{a,b,c\}$ that contain aaa as a substring.

DFA’s as data structures

```
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
```

w =“$w_1 w_1 \ldots w_n$”

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 2

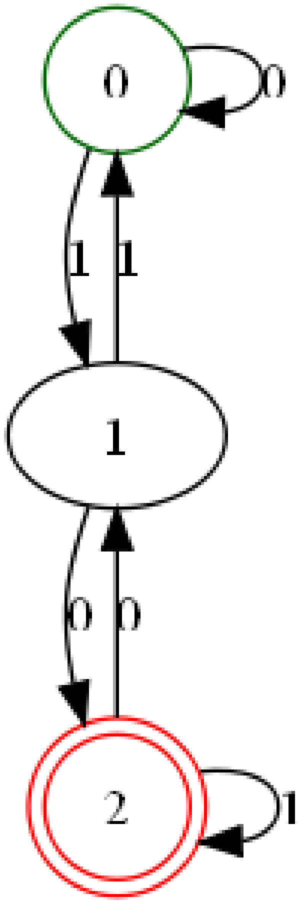
```
-- δ = deltaBar
deltaBar :: Eq q => DFA q s -> q -> [s] -> q
deltaBar dfa q [] = q
deltaBar dfa q (s:ss) =
      deltaBar dfa (trans dfa q s) ss


acceptDFA2 dfa w =
   elem (deltaBar dfa (start dfa) w)
        (accept dfa)
```

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]
```

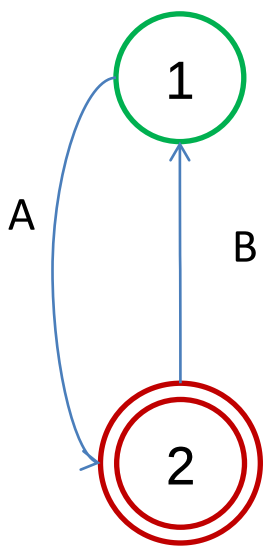
```
d1 = DFA states symbol trans start final
  where states = [0,1,2]
        symbol = [0,1]
        trans p a = (2*p+a) `mod` 3
        start = 0
        final = [2]
```

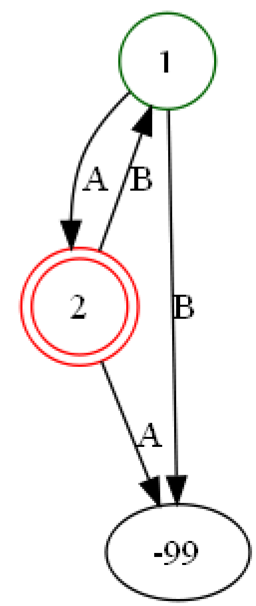
DFA	Q	{0, 1, 2}
	Sigma	{0, 1}
	Delta	0 0 -> 0
		0 1 -> 1
		1 0 -> 2
		1 1 -> 0
		2 0 -> 1
		2 1 -> 2
	q0	0
	Final	{2}



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 | w has at least 3 a's and at least 2 b's}
  - $\Sigma = \{a,b\}$

- {w | length w = 3 }
  - $\Sigma = \{m,n,p\}$

- {w | w represents an integer }
  - $\Sigma = \{a,b,c,0,2,1\}$