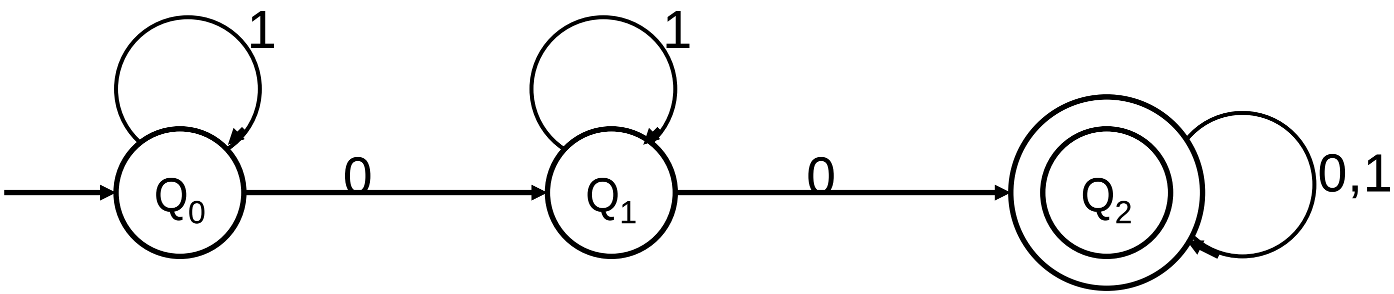
Deterministic Finite State Automata

Sipser pages 31-46

Deterministic Finite Automata (DFA)

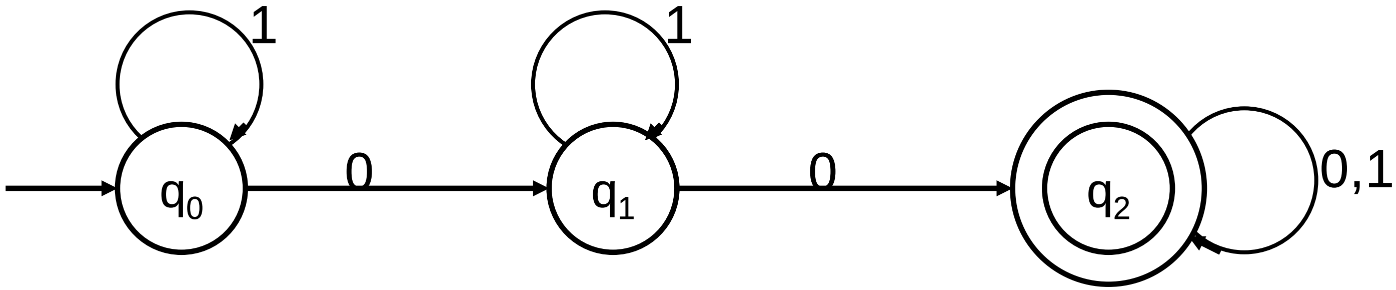
- DFAs are easiest to present pictorially:



They are directed graphs whose nodes are *states* and whose arcs are labeled by one or more symbols from some alphabet Σ.

Here Σ is {0,1}.

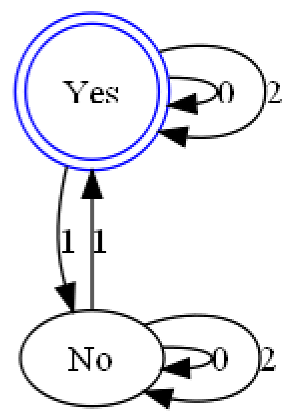
Such a graph is called a state transition diagram.

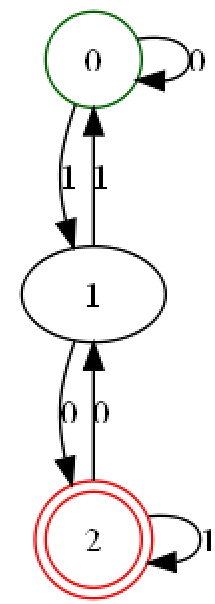
- One state is *initial* (denoted by a short incoming arrow), and several are *final/accepting* (denoted by a double circle). For every symbol $a \in \Sigma$ there is an arc labeled *a* emanating from every state.
- 

- 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

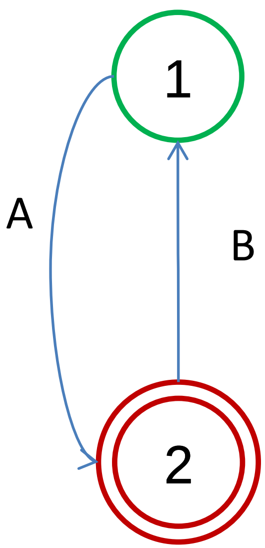
- I use some software to draw DFAs, which is somewhat limited. So I use conventions
  1. Initial states are green circles
  2. Final states are double red circles
  3. Other states are oval
  4. If the initial and final states overlap, I use blue double circle.

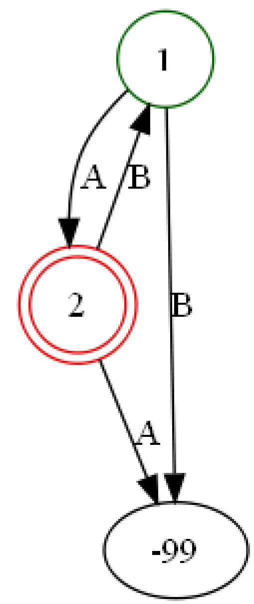




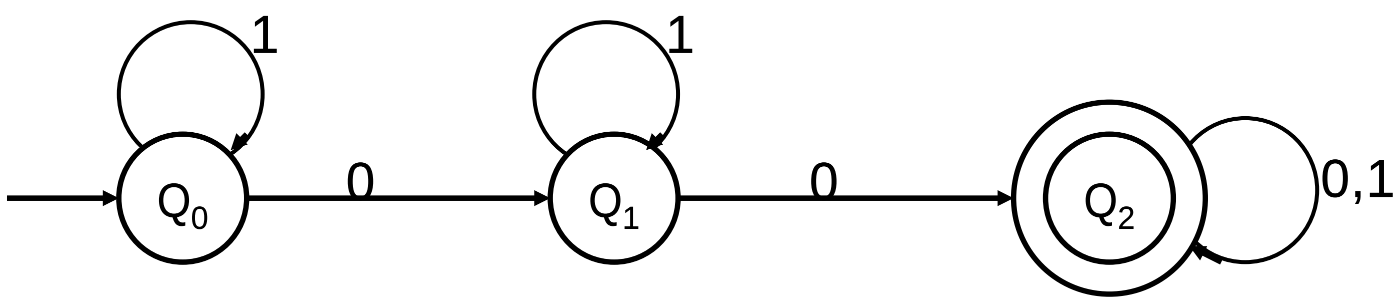
Missing alphabet

- 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*.



- 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.
-

Formal Definition

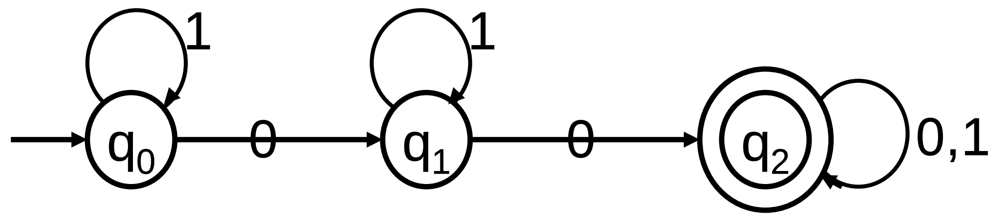
- **A D**FA is a quintuple $\mathbf{A} = (\mathbf{Q}, \Sigma, \delta, \mathbf{q_0}, \mathbf{F})$ where
  - $\mathbf{Q}$ is a set of ***states***
  - Σ is the **alphabet** (of *input symbols*)
  - δ: $\mathbf{Q} \times \Sigma \to \mathbf{Q}$ is the ***transition function***
  - $\mathbf{q_0} \in \mathbf{Q}$ -- 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
- $\mathbf{w}$ = "$w_1 w_1 \ldots w_n$" 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. $r_n \in F$

Acceptance is about finding a sequence.

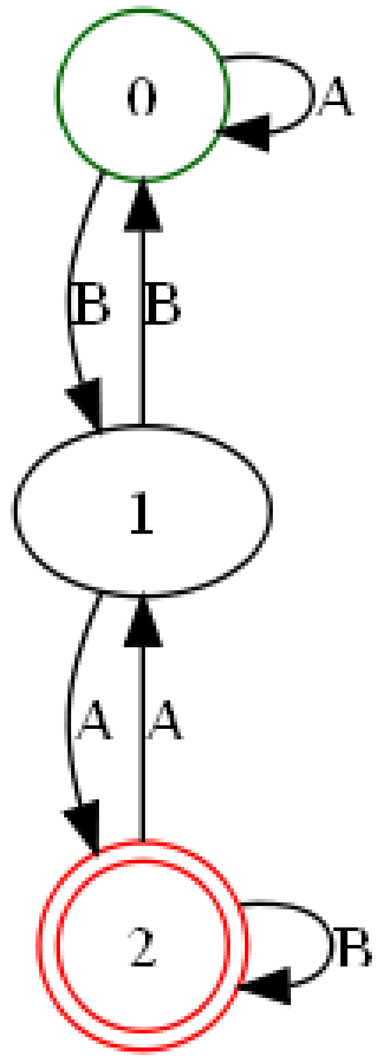
How do we find such a sequence?

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
    - **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 it is accepted by some DFA.

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 $\underline{\delta} : Q \times \Sigma^* \rightarrow Q$
- is defined recursively:

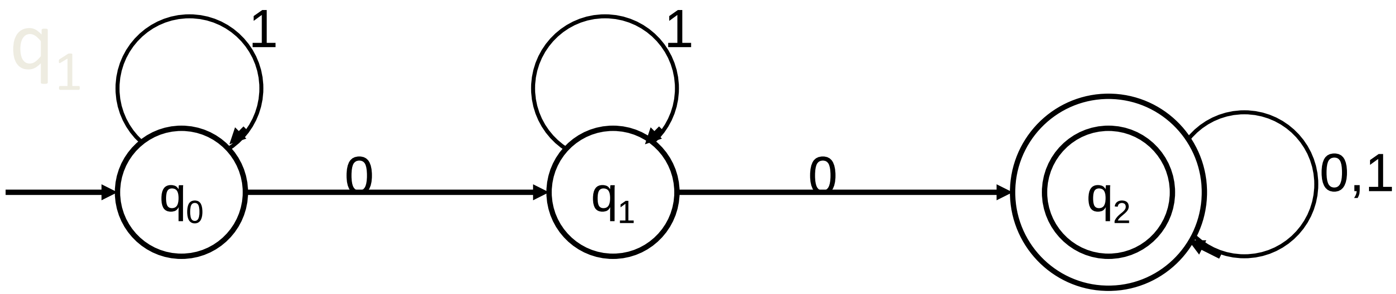
  - $\underline{\delta}(\mathbf{q},\varepsilon)=\mathbf{q}$
  - $\underline{\delta}(\mathbf{q},\mathbf{x:xs})= \underline{\delta}(\delta(\mathbf{q},\mathbf{x}),\mathbf{xs})$

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

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 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

```
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 1

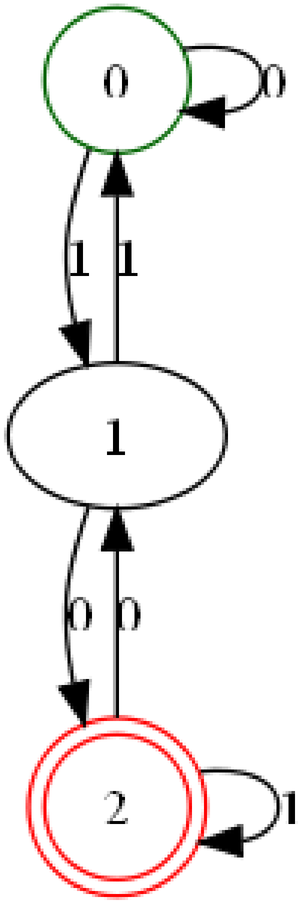
```
-- δ = deltaBar
deltaBar :: Eq q => DFA q s -> q -> [s] -> q
deltaBar dfa q [] = q
deltaBar dfa q (s:ss) =
      deltaBar dfa (trans dfa q s) ss


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}**