NFA defined

NFA

- A Non-deterministic Finite-state Automata (NFA) is a language recognizing system similar to a DFA.
- It supports a level of non-determinism. I.e. At some points in time it is possible for the machine to take on many next-states.
- Non-determinism makes it easier to express certain kinds of languages.

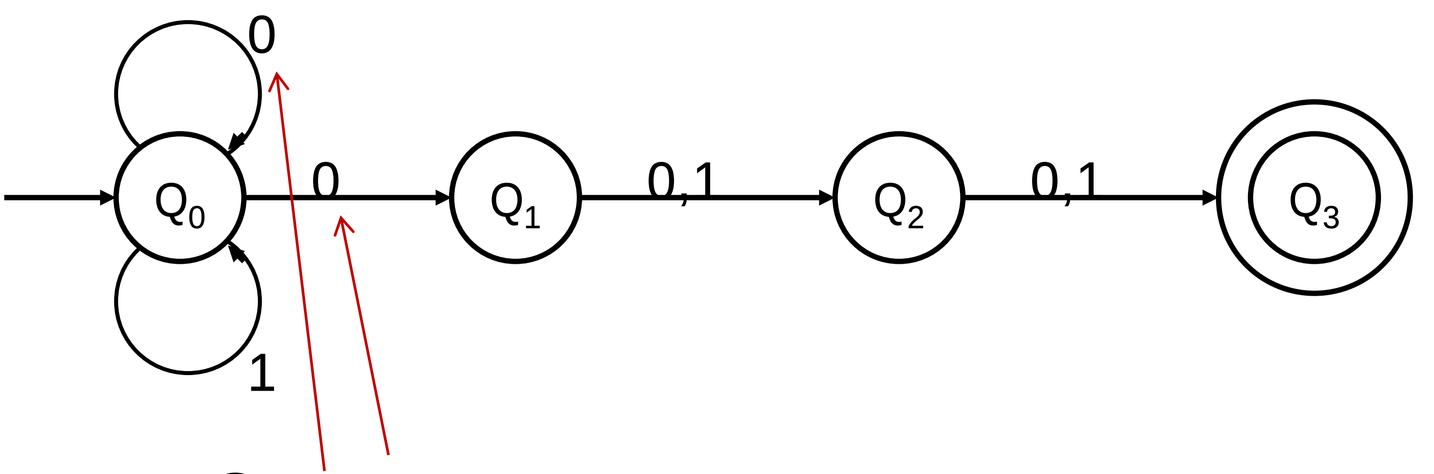
Nondeterministic Finite Automata (NFA)

- When an NFA receives an input symbol a, it can make a transition to zero, one, two, or even more states.
  - each state can have multiple edges labeled with the same symbol.

- An NFA accepts a string w iff there exists a path labeled w from the initial state to one of the final states.
  - In fact, because of the non-determinism, there may be many states labeled with w

Example N1

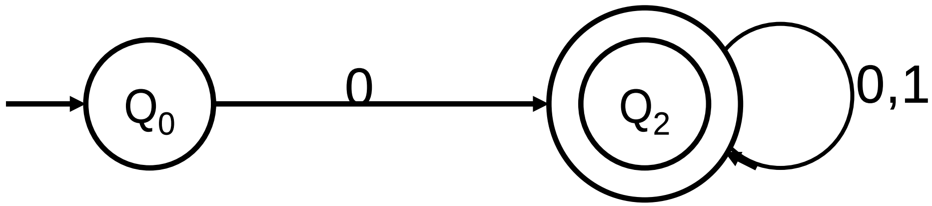
- The language of the following NFA consists of all strings over {0,1} whose 3rd symbol from the right is 0.



- Note Q_0 has multiple transitions on 0

Example N2

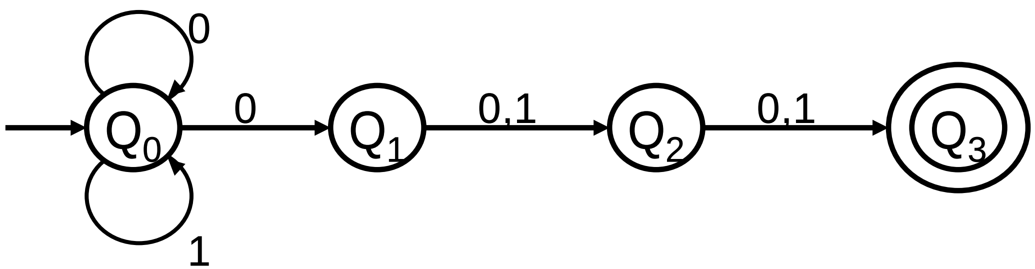
- The NFA N_2 accepts strings beginning with 0.



- Note Q_0 has no transition on 1
  - It is acceptable for the transition function to be undefined on some input elements for some states.

NFA Processing

- Suppose N_1 receives the input string `0011`. There are three possible execution sequences:
- $q_0 \longrightarrow q_0 \longrightarrow q_0 \longrightarrow q_0 \longrightarrow q_0$
- $q_0 \longrightarrow q_0 \longrightarrow q_1 \longrightarrow q_2 \longrightarrow q_3$
- $q_0 \longrightarrow q_1 \longrightarrow q_2 \longrightarrow q_3$



- Only the second finishes in an accept state. The third even gets stuck (cannot even read the fourth symbol).
- As long is there is at least one path to an accepting state , then the string is accepted.

Implementation

- Implementation of NFAs has to be deterministic, using some form of backtracking to go through all possible executions .

- Any thoughts on how this might be accomplished?

Formal Definiton

- An NFA is a quintuple $A=(Q,\Sigma,s,F,T)$, where the first four components are as in a DFA, and the transition function takes values in P(Q) (the power set of Q) instead of Q. Thus
  - $T: Q \times \Sigma \longrightarrow P(Q)$ note that T returns a set of states

- A NFA $A=(\mathbf{Q},\Sigma,\mathbf{s},\mathbf{F},T)$, *accepts* a string $x_1x_2..x_n$ (an element of Σ^*) iff there exists a sequence of states $q_1q_2..q_nq_{n+1}$ such that
  - $q_1 = s$
  - $q_{i+1} \in T(q_i, x_i)$
  - $Q_{n+1} \cap F \neq \emptyset$

Compare with

A DFA $A=(\mathbf{Q},\Sigma,\mathbf{s},\mathbf{F},T)$, *accepts* a string $x_1x_2..x_n$ (an element of Σ^*) iff
There exists a sequence of states $q_1q_2..q_nq_{n+1}$ such that
1. $q_1 = s$
2. $q_{i+1} = T(q_i, x_i)$
3. Q_{n+1} is an element of $\mathbf{F}$

The extension of the transition function

- Let an NFA $A=(Q,\Sigma,s,F,\delta)$

- The extension $\underline{\delta} : Q \times \Sigma^* \longrightarrow P(Q)$ extends δ so that it is defined over a string of input symbols, rather than a single symbol. It is defined by
  - $\underline{\delta}(q,\varepsilon)=\{q\}$
  - $\underline{\delta}(q,ua) = \cup_{p\in \underline{\delta}(q,u)} \delta(p,a)$,

  Compute this by taking the union of the sets $\delta(p,a)$, where p varies over all states in the set $\underline{\delta}(q,u)$

  - First compute $\underline{\delta}(q,u)$, this is a set, call it S.
  - for each element, p in S, compute $\delta(p,a)$,
  - Union all these sets together.

Another NFA Acceptance Definition

- An NFA accepts a string w iff $\underline{\delta}(s,w)$ contains a final state. The language of an NFA N is the set L(N) of accepted strings:

- $L(N) = \{w \mid \underline{\delta}(s,w) \cap F \neq \emptyset\}$

- **Compare this with the 2 definitions of DFA acceptance in last weeks lecture.**

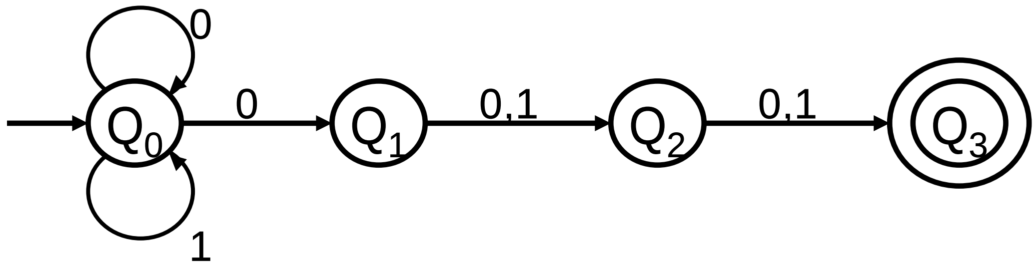
A DFA $A = (Q, \Sigma, s, F, T)$, *accepts* a string $x_1x_2..x_n$ (an element of Σ^*) iff there exists a sequence of states $q_1q_2..q_nq_{n+1}$ such that

1. $q_1 = s$
2. $q_{i+1} = T(q_i, x_i)$
3. Q_{n+1} is an element of F

$L(A) = \{w \mid \underline{T}(s,w) \in F\}$

compute $\underline{\delta}(q_0,000)$

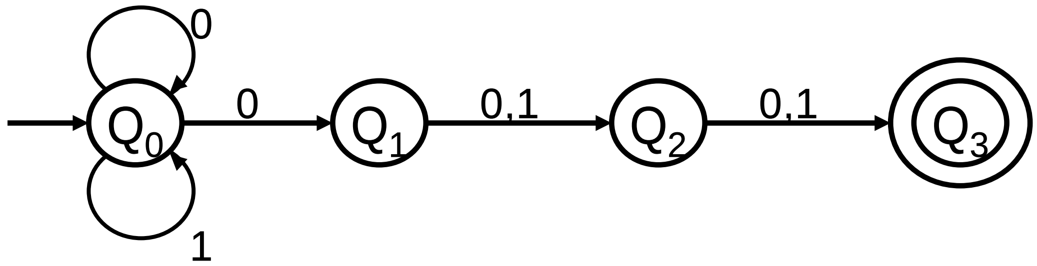
- $\underline{\delta}(q,ua) = \cup_{p\in\ \underline{\delta}(q,u)} \delta(p,a)$



- $\underline{\delta}(q_0,000) = \cup_{x\in\underline{\delta}(q0,00)} \delta(x,0)$
- $\underline{\delta}(q_0,00) = \cup_{y\in\underline{\delta}(q0,0)} \delta(y,0)$
- $\underline{\delta}(q_0,0) = \delta(q_0,0) = \{q_0,q_1\}$
- $\underline{\delta}(q_0,00) = \cup_{y\in\{q0,q1\}} \delta(y,0)$
- $\underline{\delta}(q_0,00) = \{q_0,q_1\} \cup \{q_2\} = \{q_0,q_1,q_2\}$
- $\underline{\delta}(q_0,000)= \cup_{x\in\{q0,q1,q2\}} \delta(x,0)$
- $\underline{\delta}(q_0,000)=\{q_0,q_1\} \cup \{q_2\} \cup \{q_3\}$
- $\underline{\delta}(q_0,000)=\{q_0,q_1,q_2,q_3\}$

Intuition

- At any point in the walk over a string, such as “000” the machine can be in a set of states.
- To take the next step, on a character ‘c’, we create a new set of states. Those reachable from the old set on a single ‘c’



	0	**1**
{Q0}	{Q0,Q1}	{Q0}
{Q0,Q1}	{Q0,Q1,Q2}	{Q0,Q2}
{Q0,Q2}	{Q0,Q1,Q3}	{Q0,Q3}
{Q0,Q1,Q3}	{Q0,Q1,Q2}	{Q0,Q2}
{Q0,Q3}	{Q0,Q1}	{Q0}
{Q0,Q1,Q2}	{Q0,Q1,Q2,Q3}	{Q0,Q2,Q3}
{Q0,Q1,Q2,Q3} **{Q0,Q2,Q3}**	{Q0,Q1,Q2,Q3} {Q0,Q1,Q3}	{Q0,Q2,Q3} {Q0,Q3}