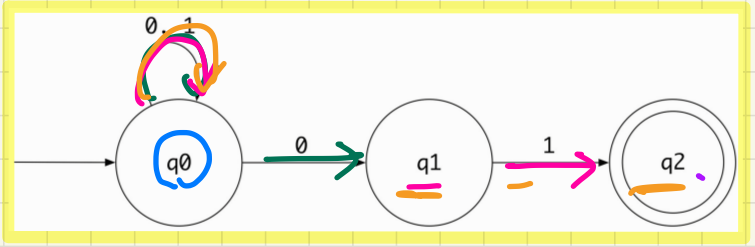
# Lecture 7 - Sep. 29

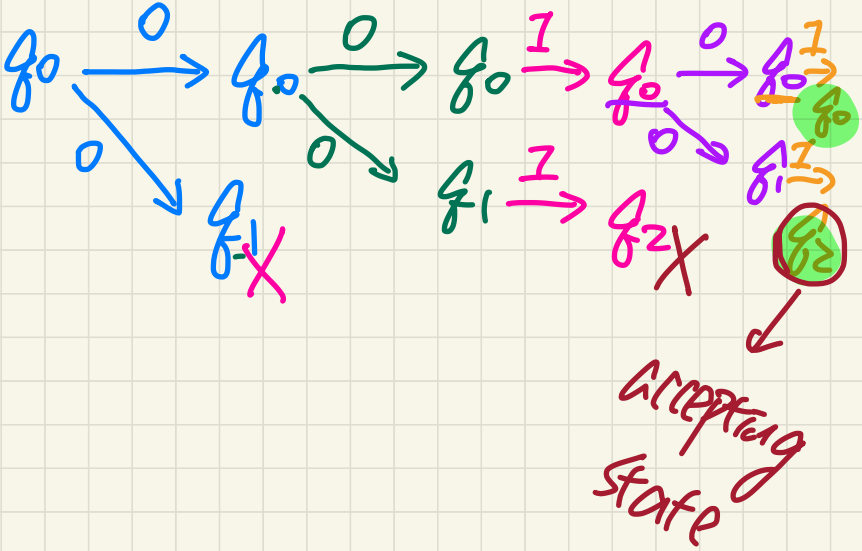
## Lexical Analysis

***NFA: Tracing & Formulation***
***NFA to DFA Conversion***
***ε-NFA: Formulation and ε-Closure***

# NFA Behaviour ≈ Alternative Universe

Obviously the **time continuum** has been disrupted, creating this new temporal event sequence resulting in this **alternate reality**. - Doc Brown



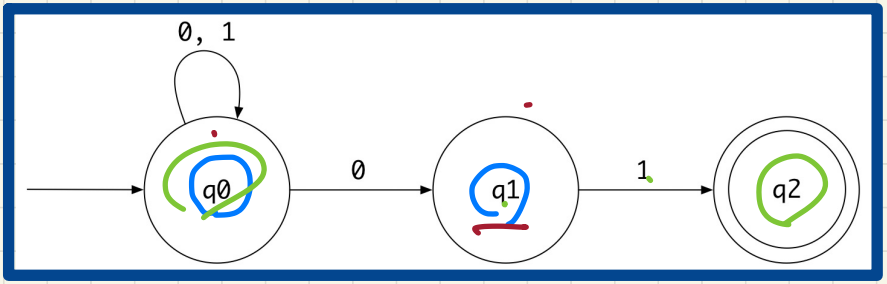




**Trace: 00101**

Try: 00111

# NFA: Processing Strings

How an NFA determines if an input 00101 should be accepted:




Read 0: $\delta(q_0, 0) = \{q_0, q_1\}$

Read 0: $\delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \{\} = \{q_0, q_1\}$

Read 0:

Read 0:

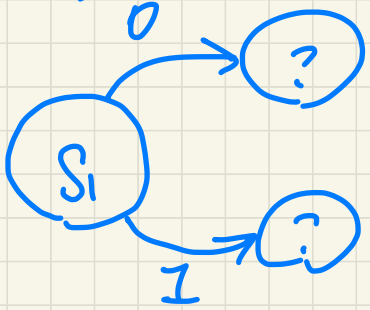
Read 0:

Exercise

$\Sigma = \{0, 1\}$

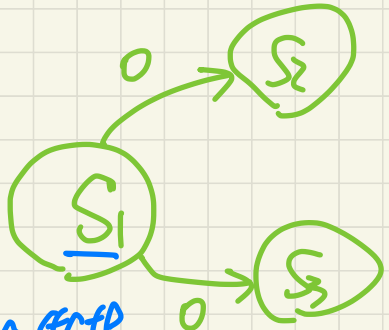
## DFA

$\delta : (Q, \Sigma) \rightarrow Q$




## NFA

$\delta : (Q, \Sigma) \rightarrow \mathcal{P}(Q)$




not necessarily every $(Q, \Sigma)$ has a defined resulting state

$((S_1, 0), \{S_2, S_3\}) \in \delta$

alt. $\delta : (Q, \Sigma) \not\rightarrow Q$ $(S_1, 1)$ undefined $((S_1, 1), \phi) \in \delta$

# NFA: Formulation

## Language of a NFA

A *nondeterministic finite automata (NFA)* is a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

$$\hat{\delta} : (Q \times \Sigma^*) \to \mathbb{P}(Q)$$

We may define $\hat{\delta}$ recursively, using $\delta$!

$$\hat{\delta}(q, \epsilon) = \{q\}$$
$$\hat{\delta}(q, xa) = \bigcup \{\delta(q', a) \mid q' \in \hat{\delta}(q, x)\}$$

where $q \in Q$, $x \in \Sigma^*$, and $a \in \Sigma$

Given an input string 00101:

- **Read 0**: $\delta(q_0, 0) = \{q_0, q_1\}$
- **Read 0**: $\delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \varnothing = \{q_0, q_1\}$
- **Read 1**: $\delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$
- **Read 0**: $\delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0, q_1\} \cup \varnothing = \{q_0, q_1\}$
- **Read 1**: $\delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0, q_1\} \cup \{q_2\} = \{q_0, q_1, q_2\}$

$$L(M) = \{w \mid w \in \Sigma^* \wedge \hat{\delta}(q_0, w) \cap F \neq \varnothing\}$$

Every DFA is an NFA.

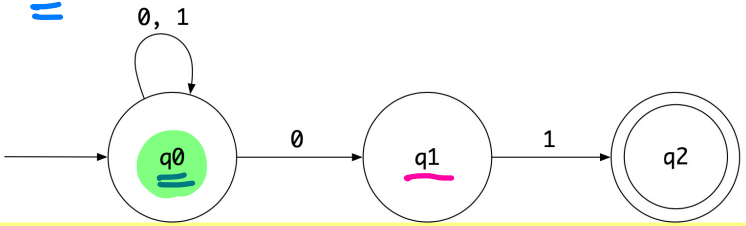
Not necessarily every NFA is a DFA.

$\Sigma$

has some missing transition $c \in \Sigma$

# NFA to DFA: Subset Construction (Lazy Evaluation)

Given an *NFA*:




***Subset construction*** (with ***lazy evaluation***) produces a ***DFA*** with $\delta$ as:

| state \ input | 0 | 1 |
|---|---|---|
| $\{q_0\}$ | $\delta(q_0, 0) = \{q_0, q_1\}$ | $\delta(q_0, 1) = \{q_0\}$ → discovered already. |
| $\{q_0, q_1\}$ | $\delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\}$ | $\delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0, q_2\}$ |
| $\{q_0, q_2\}$ ($\subseteq Q$) | (Exercise) | |

subset state = each state in the DFA corresponds to a set of states in NFA

# Subset Construction: Algorithmic Specification

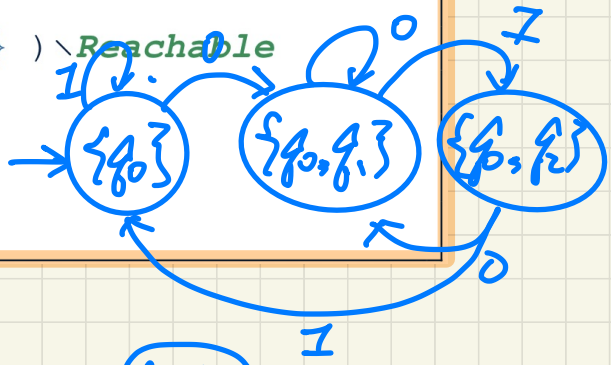
Given an **NFA** $N = (Q_N, \Sigma_N, \delta_N, q_0, F_N)$:

**ALGORITHM:** *ReachableSubsetStates*
**INPUT:** $q_0 : Q_N$ ; **OUTPUT:** `Reachable` $\subseteq \mathbb{P}(Q_N)$
**PROCEDURE:**

```
Reachable := { {q0} }
ToDiscover := { {q0} }
while ( ToDiscover ≠ ∅ ) {
  choose S : ℙ(Q_N) such that S ∈ ToDiscover
  remove S from ToDiscover
  NotYetDiscovered :=
      ( { {δ_N(s,0) | s ∈ S} } ∪ { {δ_N(s,1) | s ∈ S} } ) \ Reachable
  Reachable := Reachable ∪ NotYetDiscovered
  ToDiscover := ToDiscover ∪ NotYetDiscovered
}
return Reachable
```

to determine if the lazy eval. should continue.

| state \ input | 0 | 1 |
|---|---|---|
| $\{q_0\}$ | $\{q_0, q_1\}$ | $\{q_0\}$ |
| $\{q_0, q_1\}$ | $\{q_0, q_1\}$ | $\{q_0, q_2\}$ |
| $\{q_0, q_2\}$ | $\{q_0, q_1\}$ | $\{q_0\}$ |



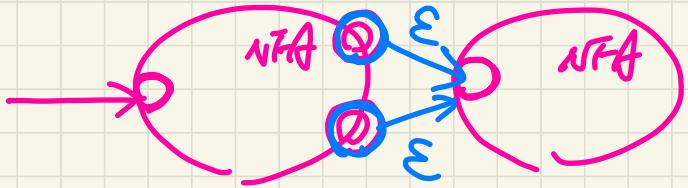

NFA: $S_0, S_1, S_2$ $\quad 2^{|Q|} = 8$

Worst case DFA: $\{\}$ $\{S_0\}$ $\{S_0,S_1\}$ $\{S_1,S_2\}$ ... $\{S_0,S_1,S_2\}$

# epsilon-NFA: Motivation

Draw NFA

$$\left\{ xy \;\middle|\; \begin{array}{ll} & x \in \{0,1\}^* \\ \wedge & y \in \{0,1\}^* \\ \wedge & x \text{ has alternating 0's and 1's} \\ \wedge & y \text{ has an odd \# 0's and an odd \# 1's} \end{array} \right\}$$




$$\left\{ w : \{0,1\}^* \;\middle|\; \begin{array}{l} w \text{ has alternating 0's and 1's} \\ \vee \\ w \text{ has an odd \# 0's and an odd \# 1's} \end{array} \right\}$$