CS 3313
Foundations of Computing:

Deterministic Finite Automata (DFA)

http://gw-cs3313-2021.github.io

Today: Deterministic Finite Automata aka Finite State Machines in Sequential Circuits

- Define Deterministic Finite Automata (DFA) Model
  - Formal definition
  - Model as a graph
  - Acceptance by DFA
  - Examples
- Deterministic: at every step, and for every input, there is exactly one next state (i.e, one decision)
- Non-deterministic Automata
  - “choice” of moves the machine can make
  - View this as exploring several “parallel” options concurrently
  - Machine eventually follows one sequence of options/choices
  - Question: does adding non-determinism add to their “power”?

DFAs and Finite State Machines (FSM)

- How are they different ? Are they different ?...NO
  - DFAs are mathematical model of FSMs, but defined as “acceptors”
    - Final output is a “yes” or “no”
  - FSMs are an implementation of a DFA and can generate different outputs from each state
- Why DFAs (/FSMs)?
  - Control unit of a processor…..yes, they are DFAs!
  - Network protocols, switching circuits,
  - Sometimes an algorithm can be modeled as a DFA
    - Ex: searching for substrings
- Focus of this course = DFAs
  - We won’t discuss the implementation of a DFA in hardware
    - You know this already!!

Recall Definitions and Notations:

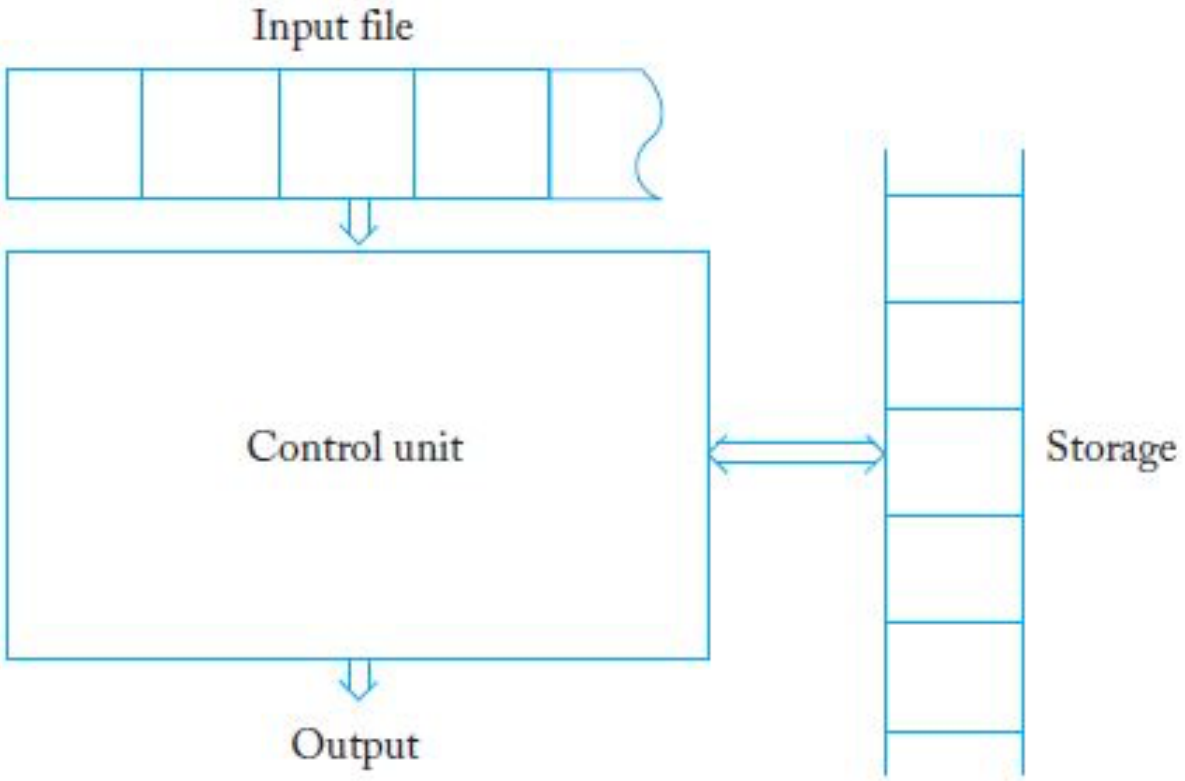
- *Alphabet*: set of symbols, i.e. $\Sigma = \{a, b\}$
- *String*: finite sequence of symbols from Σ
  - Empty string: denoted λ or ϵ
- *Operations on strings*:*Concatenation, Reverse,* ..
- *Length of a string: number of symbols*
- Σ^* = set of all strings formed by concatenating zero or more symbols in Σ
- Σ^+ = set of all non-empty strings formed by concatenating symbols in Σ, i.e., $\Sigma^+ = \Sigma^* - \{ \lambda \}$
- A *formal language* L is any subset of Σ^*

- Convention: we use *w,x,y* to denote strings and *a,b,c* to denote symbols from the alphabet

Recall: Automata Definition

- An *automaton* is an abstract model of a digital computing device
- An automaton consists of
  - An input mechanism
  - A control unit
  - Possibly, a storage mechanism
  - Possibly, an output mechanism
- Control unit can be in any number of internal *states*, as determined by a *next-state* or *transition* function.
- There are a finite number of states
  - *Why ? Because an implementation requires finite devices*

Illustration of a General Automaton



Note: In DFA model, there is no storage device

Deterministic Finite Automata (Accepters)

- **Definition:** A *deterministic finite automaton (DFA)* is defined as M= (Q, Σ, δ, q_0 ,F) where:
  1. Q: a finite set of ***states***
  2. Σ: a set of symbols called the ***input alphabet***
  3. δ: a ***transition function*** from Q X Σ to Q
  4. q_0 : the ***start (initial) state***
  5. F: a subset of Q representing the ***final states***

  Final state also called "accepting" state;

  Start state also called "initial" state

- Example dfa M:

$Q = \{ q_0, q_1, q_2 \}$ $\Sigma = \{ 0, 1 \}$ $F = \{q_0\ q_1 \}$

where the transition function is given by

$\delta(q_0, 0) = q_0$ $\delta(q_0, 1) = q_1$ $\delta(q_1, 0) = q_0$

$\delta(q_1, 1) = q_2$ $\delta(q_2, 0) = q_2$ $\delta(q_2, 1) = q_2$

The Transition Function

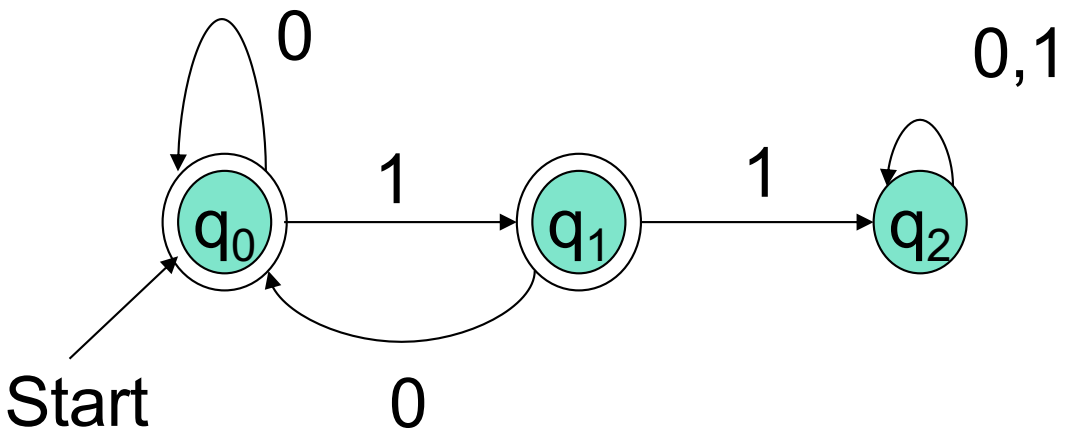
- Takes two arguments: a state and an input symbol.
- δ(q, a) = the state that the DFA goes to when it is in state *q* and input *a* is received.
- Note: always a next state – add a trap state (also called *dead state*) if no transition

Graph Representation of DFA's

- Nodes = states.
- Edges (arcs) represent transition function.
  - Edge from state p to state q labeled by all those input symbols that have transitions from p to q.
- Arrow labeled "Start" to the start state.
- Final states indicated by double circles.

$\delta(q_0, 0) = q_0$ $\delta(q_0, 1) = q_1$ $\delta(q_1, 0) = q_0$

$\delta(q_1, 1) = q_2$ $\delta(q_2, 0) = q_2$ $\delta(q_2, 1) = q_2$



Processing Input with a DFA

- A DFA starts by processing the leftmost input symbol with its control in state q_0. The transition function determines the next state, based on current state and input symbol
- The DFA continues processing input symbols until the end of the input string is reached
- The input string is *accepted* if the automaton is in a final state after the last symbol is processed. Otherwise, the string is *rejected*.
- For example, the dfa in example 2.1 accepts the string 111 but rejects the string 110

Alternative Representation: Transition Table

Final states starred

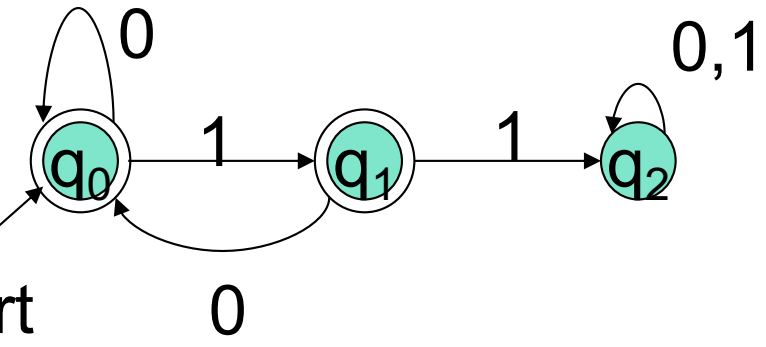
Arrow for start state

Columns = input symbols

	0	1
→ * q_0	q_0	q_1
* q_1	q_0	q_2
q_2	q_2	q_2

Rows = states

Each entry is δ of the row and column.



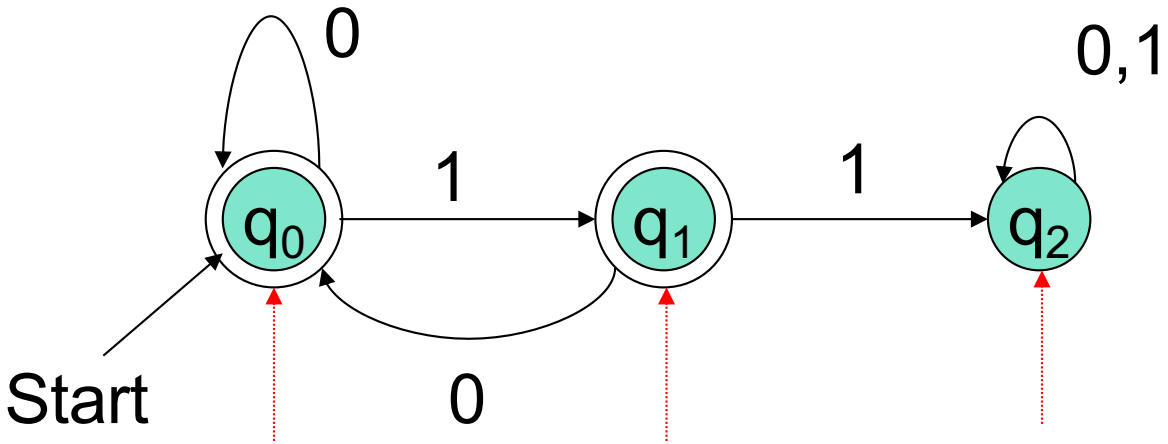
States in a DFA

- Finite number of states Q
- What does a state denote – i.e., design process
  - State summarizes a finite amount of information
  - Summary based on past events
    - In DFA, the past events are inputs read by the automaton until this point in time
    - Ex: input= **bb**a**a**bb
    - After reading bb, DFA is in some state *p* and then reads **a**
      - The current state *p* depends on the input **bb**
      - The next state *q* depends on current state *p* and the input *a*

Example: A Vending Machine

- Accept user input (coins) when total is at least 50 cents
  - Real machine: dispense output (candy) when total is 50 or more
- Input valid coins: alphabet = { 5,10,25}
  - Q (25cents) D (10) or N (5)
- What should it keep track of ?
  - current total
- When it reaches 50 or more: Final state
  - Generate output
- States of the machine ?
  - What should each state capture ?
  - How many states ?

Example: Strings With No 11



String so far has no 11, does not end in 1.

String so far has no 11, but ends in a single 1.

Consecutive 1's have been seen.

Extended Transition Function

- We describe the effect of a string of inputs on a DFA by extending δ to a state and a string.
- Intuition: Extended δ is computed for state q and inputs $a_1a_2\ldots a_n$ by following a path in the transition graph, starting at q and selecting the arcs with labels $a_1, a_2,\ldots, a_n$ in turn.

- Notation: we start by denoting the extended function as δ^* and will drop the * after we define it

Inductive Definition of δ^* – Extended δ

- Induction on length of string.
- Basis: $\delta^*(q, \epsilon) = q$
- Induction: $\delta^*(q,wa) = \delta(\delta(q,w),a)$
  - Remember: w is a string; a is an input symbol, by convention.

Example: Extended Delta

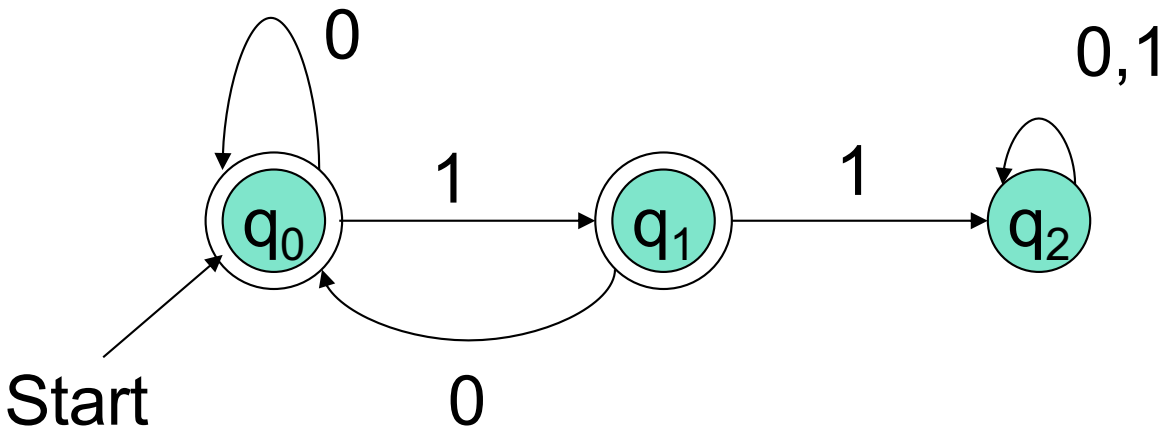
	0	1
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_2	q_2

$$\delta(q_1,011) = \delta(\delta(q_1,01),1) = \delta(\delta(\delta(q_1,0),1),1) =$$

$$\delta(\delta(q_0,1),1) = \delta(q_1,1) = q_2$$

Example: String in a Language

String 101 is in the language of the DFA below.
Start at A.



Language accepted by a DFA

- Automata of all kinds define languages.
- If A is an automaton, L(A) is its language.
- For a DFA A, L(A) is the set of strings labeling paths from the start state to a final state.
- Formally:

L(A) = the set of strings w such that $\delta(q_0, w)$ is in F.

$L(A) = \{ w \mid \delta(q_0, w) \in F \}$

Definition: Regular Languages

- DFAs accept a family of languages collectively known as *regular languages*.
- A language L is *regular* if and only if there is a DFA M that accepts L.
  - Therefore, to show that a language is regular, one must construct a DFA to accept it, i.e., L=L(M) for some DFA A.
- Regular languages have wide applicability in problems that involve scanning input strings in search of specific patterns.
- Some languages are not regular, i.e., there is no DFA M that accepts the language
  - Intuitively, regular languages cannot “count” to arbitrarily high integers.
  - To show that a language is not regular we need to prove that there is no DFA M that accepts the language

Example 1:

- L = { w | w is a string in $\{0,1\}^*$ and w contains the substring 101}
  - Input is binary string, and DFA M is in final state if input contains 101.
- What should the states keep track of ?
  - Pattern of the input read ‘thus far’
    - All of the input or the relevant part, i.e., the properties of the substring ?

Example 1:

- L = { w | w is a string in $\{0,1\}^*$ and w contains the substring 101}

Example 1:

Exercise 1: Work in breakout groups and submit

- Provide a DFA for L = { w | w is a string in $\{0,1\}^*$ and w contains (a) the substring 101 or (b) substring 010 }
  - Ex: 00101011 is in L, 10110 is in L,
  - Ex:1110 is not in L, 0001100 is not in L

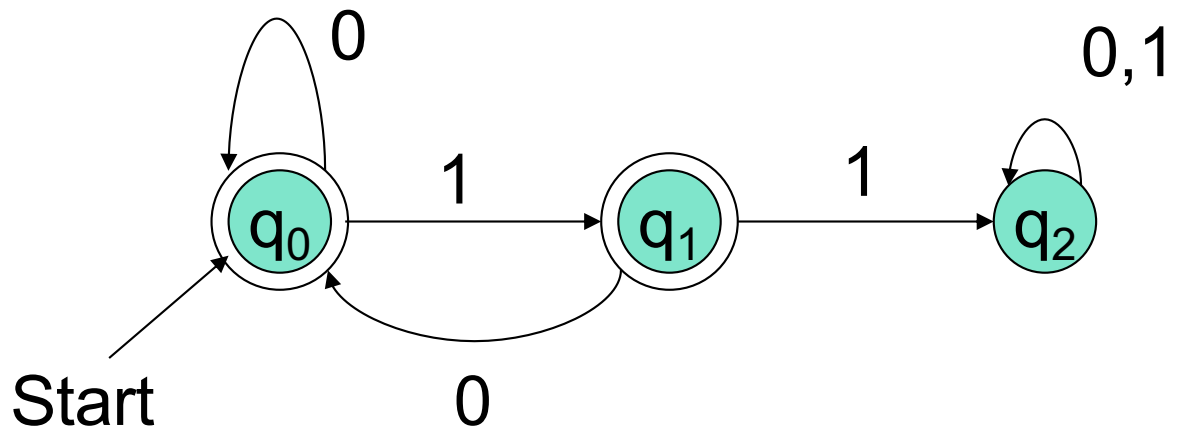
Exercise 1

Example – Proving Property of language accepted by DFA

- The language of our example DFA is:

{w | w is in {0,1}* and w does not have two consecutive 1's}

These conditions about w are true.



Proofs of Set Equivalence

- Often, we need to prove that two descriptions of sets are in fact the same set.
- Here, one set is “the language of this DFA,” and the other is “the set of strings of 0’s and 1’s with no consecutive 1’s.”

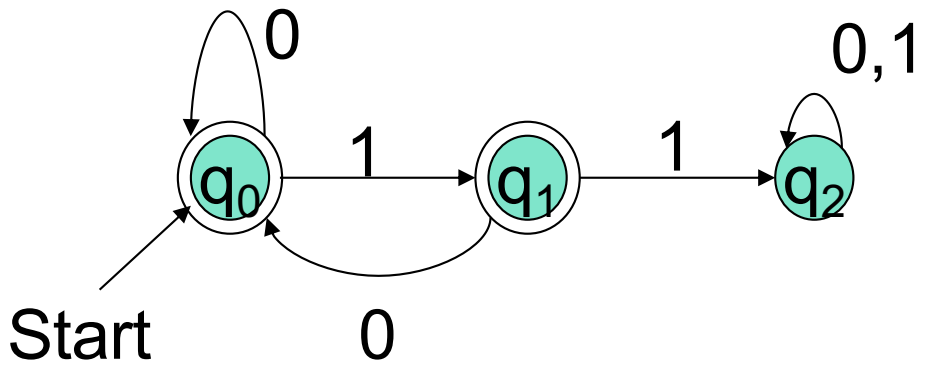
Proofs – (2)

- In general, to prove S = T, we need to prove two parts: S $\subseteq$ T and T $\subseteq$ S. That is:
  1. If w is in S, then w is in T.
  2. If w is in T, then w is in S.
- Here, S = the language of our running DFA, and T = "no consecutive 1's."

Part 1: S ⊆ T

- To prove: if w is accepted by

  then w has no consecutive 1's.
- Proof is an induction on length of w.
- *Important trick*: Expand the inductive hypothesis to be more detailed than the statement you are trying to prove.



The Inductive Hypothesis

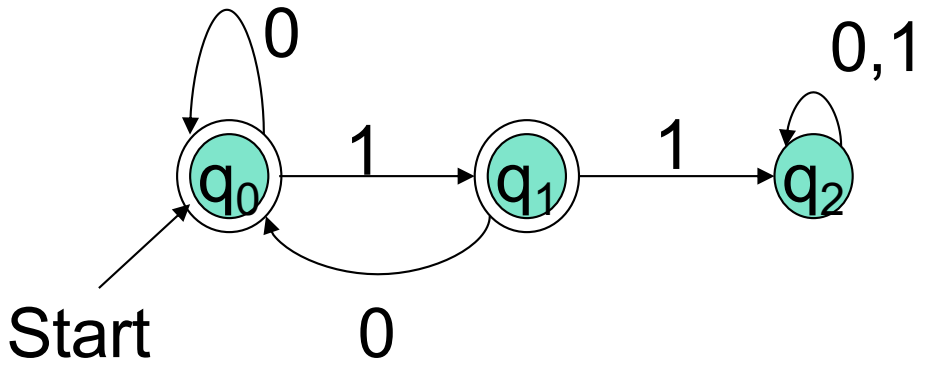
1. If $\delta(q_0, w) = q_0$, then w has no consecutive 1's and does not end in 1.
2. If $\delta(q_0, w) = q_1$, then w has no consecutive 1's and ends in a single 1.

- Basis: $|w| = 0$; i.e., $w = \epsilon$.
  - (1) holds since ϵ has no 1's at all.
  - (2) holds *vacuously*, since $\delta(q_0, \epsilon)$ is not q_1.

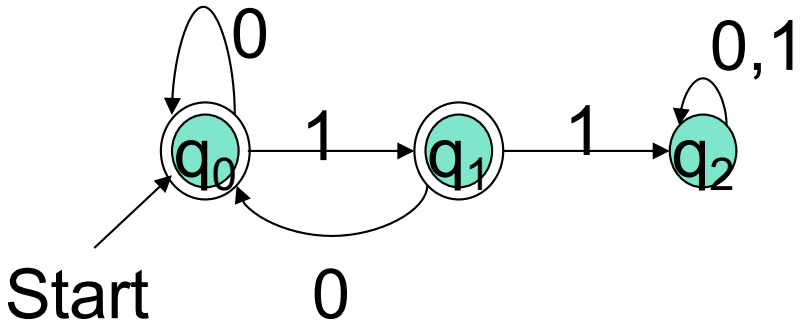
"length of"

Important concept:
If the "if" part of "if..then" is false, the statement is true.

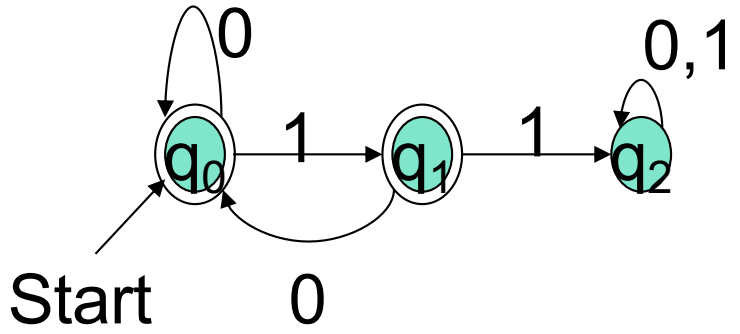
Inductive Step



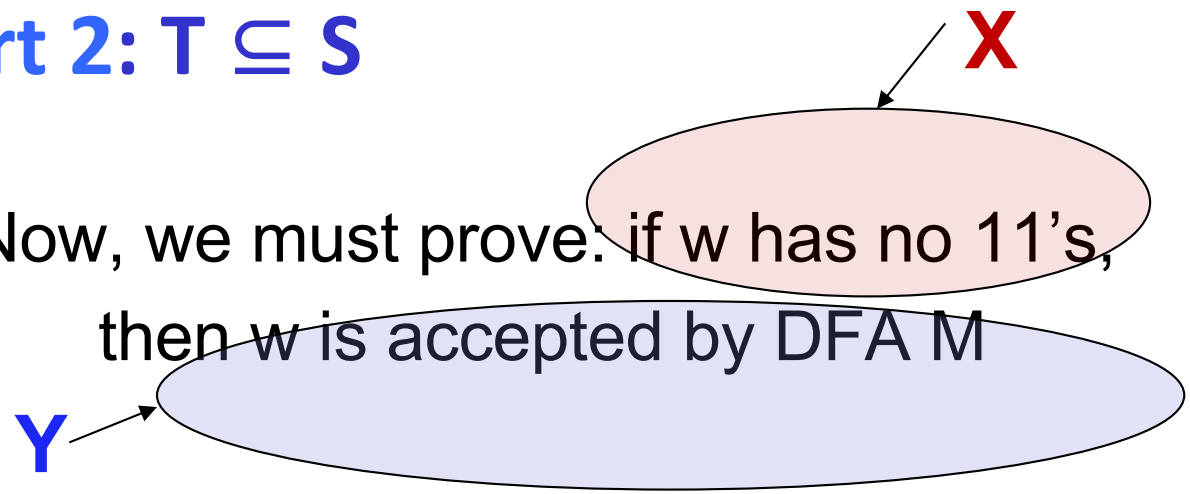
Inductive Step – (2)



Inductive Step – (3)

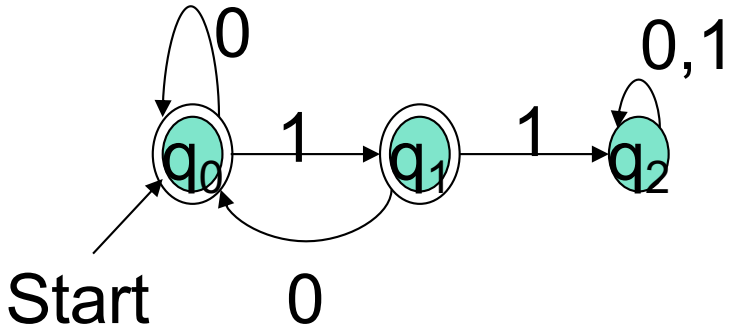


Part 2: T ⊆ S



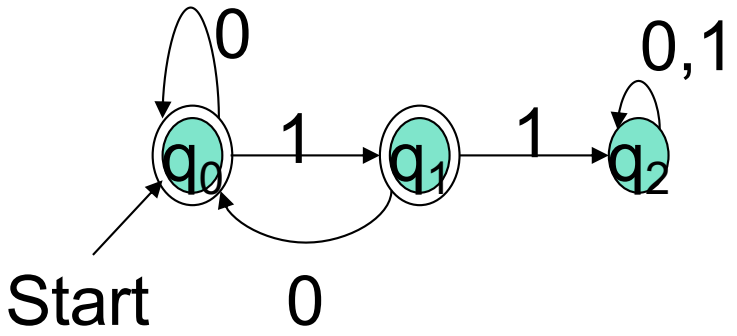
- Now, we must prove: if w has no 11's, then w is accepted by DFA M

- *Contrapositive* : If w is **not** accepted by M then string w has 11

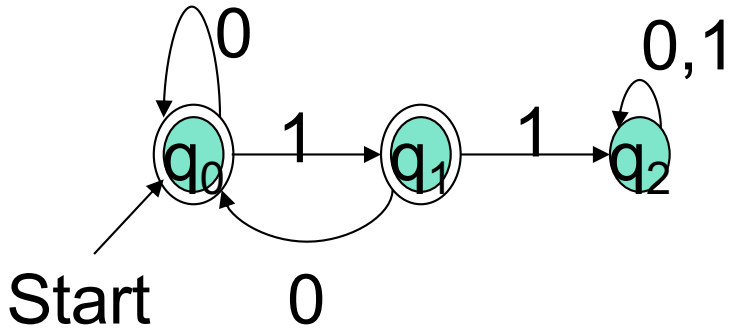


Key idea: contrapositive of "if X then Y" is the equivalent statement "if not Y then not X."

Using the Contrapositive



Using the Contrapositive – (2)



Nondeterministic Finite Accepters

- An automaton is nondeterministic if it has a choice of actions for given conditions
- Basic differences between deterministic and nondeterministic finite automata:
  - In an nfa, a (state, symbol) combination may lead to several states simultaneously
  - If a transition is labeled with the empty string as its input symbol, the nfa may change states without consuming input
  - an nfa may have undefined transitions

Definition: NFA

- M= (Q, Σ, δ, q_0 ,F)
- A finite set of states, typically Q.
- An input alphabet, typically Σ.
- A transition function, typically δ from Q X Σ to 2^Q
- A start state in Q, typically q_0.
- A set of final states F ⊆ Q.
- Difference with DFAs: transition function reads input a in state q and goes to a subset of states in Q
  - Ex: $Q = \{ q_0, q_1, q_2 \}$ $\Sigma = \{ 0, 1 \}$ $F = \{q_0\ q_1 \}$
    where the transition function is given by
    $\delta(q_0, 0) = \{q_0 \}$ $\delta(q_0, 1) = \{q_0 ,q_1 \}$
    ……

DFA and NFAs

- Like the DFA, a nondeterministic finite automaton, or NFA, has one start state, where computation begins.
- The NFA can have any number of final states, and **an input is accepted if any sequence of choices leads from the start state to some final state.**

- The intuition is that the NFA is allowed to guess which way to go, but it is able always to guess right, since all the guesses are followed in parallel and the NFA gets credit for the right guesses, no matter how many wrong guesses it also makes.

Example

- Design an NFA M which accepts

  L(M) = {w | w contains substring 101 or 010 and w is a binary string }

NFA for L= { w | w contains 010 or 101 }

Next…and to-do

- NFAs and Regular expressions
- Install JFLAP and start using it.
- HW1 will be posted Friday.