CS 3313
Foundations of Computing:

Non-Deterministic Finite Automata (NFA)

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

Non-Deterministic Finite Automata (NFA) and Equivalence to DFA

- Define Non-Deterministic Finite Automata (NFA) Model
  - Simplified form first (without transitions on empty string)
  - Model as a graph
  - Acceptance by NFA
  - Examples
- Equivalence with DFAs
- Extend NFA model to include moves on empty string input
  - Note: Book jumps straight to this model of NFA

Recall Definition: DFA

- **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
- Can extend transition function to apply over strings. δ(q,w)
  - Basis: $\delta^*(q, \epsilon) = q$
  - Induction: $\delta^*(q,wa) = \delta(\delta(q,w),a)$
    - w is a string; a is an input symbol, by convention.

Recall Definitions: Language accepted by a DFA

- For a DFA M, L(M) is the set of strings labeling paths from the start state to a final state, i.e.,

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

L(AM = $\{ w \mid \delta(q_0, w) \epsilon F \}$

- 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.

Nondeterministic Finite Accepters

- An automaton is nondeterministic if it has a choice of moves (next state) from current state and input
  - View this as *exploring several parallel options concurrently*
  - Machine *eventually follows one sequence of options/choices*

- Basic differences between deterministic and nondeterministic finite automata:
  - In an nfa, a (state, symbol) combination may lead to several states <u>simultaneously</u>
  - If a transition is labeled with the empty string as its input symbol, the nfa may change states <u>without consuming input</u>
  - an nfa may have <u>undefined transitions</u>
- Question: does adding non-determinism add to their "power"?
  - Power: we can solve a problem using non-determinism which we cannot solve using deterministic automaton
    - Different from efficiency/time

Definition: NFA

- $M = (Q, \Sigma, \delta, q_0, F)$
- 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 $F \subseteq Q$.
- 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, and reads one input symbol at each step.
- 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.**

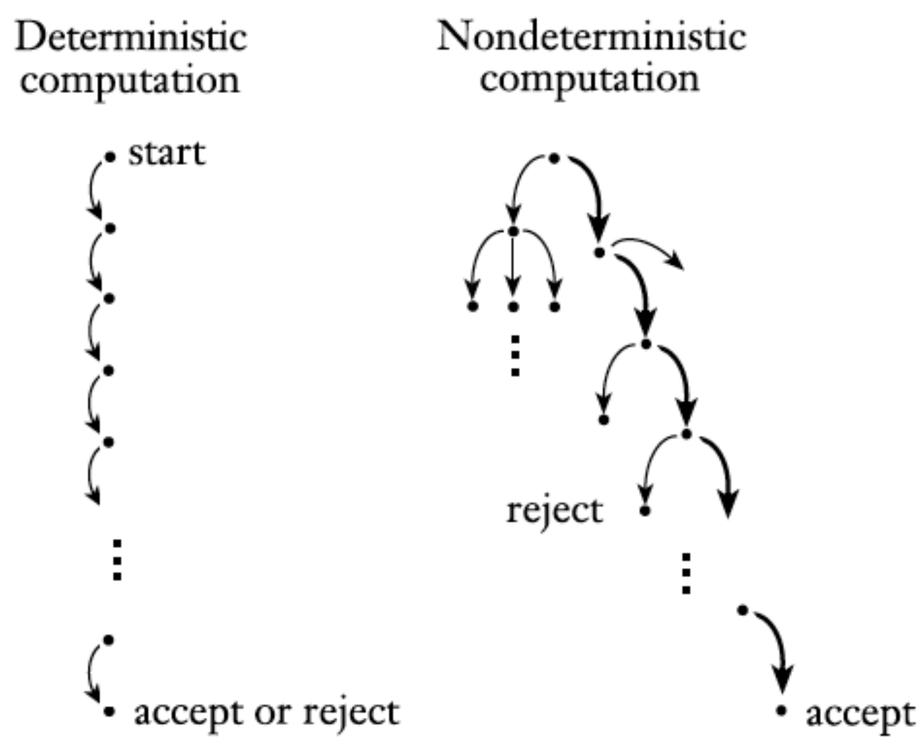
- The intuition is that the NFA is allowed to guess which way to go, but it is able always to guess right, since all the guesses are followed in parallel and the NFA gets credit for the right guesses, no matter how many wrong guesses it also makes.

Non-determinism

- Nondeterminism: useful concept with big impact on theory of computation
- A "parallel" computation wherein multiple independent "*processes*" (or "*threads*") can be running concurrently and

  if at least one of these processes accepts (i.e., computes

  result) then process accepts



Language of an NFA

- A string w is accepted by an NFA if $\delta(q_0, w)$ contains at least one final state.

$$L(M) = \{ w \mid \delta(q_0, w) \cap F \neq \emptyset \}$$

The language of the NFA is the set of strings it accepts.

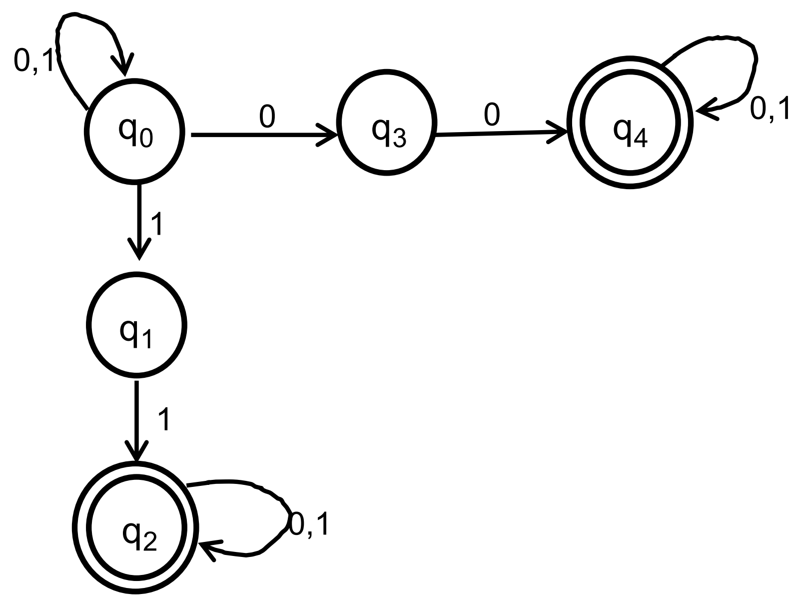
Extended Transition Function of an NFA

- $\delta(q, a)$ is a set of states.
- Extend to strings as follows:
- Basis: $\delta(q, \epsilon) = \{q\}$
- Induction: $\delta(q, wa)$ = the union over all states p in $\delta(q, w)$ of $\delta(p, a)$

$$\delta(q, wa) = \cup_{p \epsilon \delta(q, w)} \delta(p_i , a)$$

NFA Example

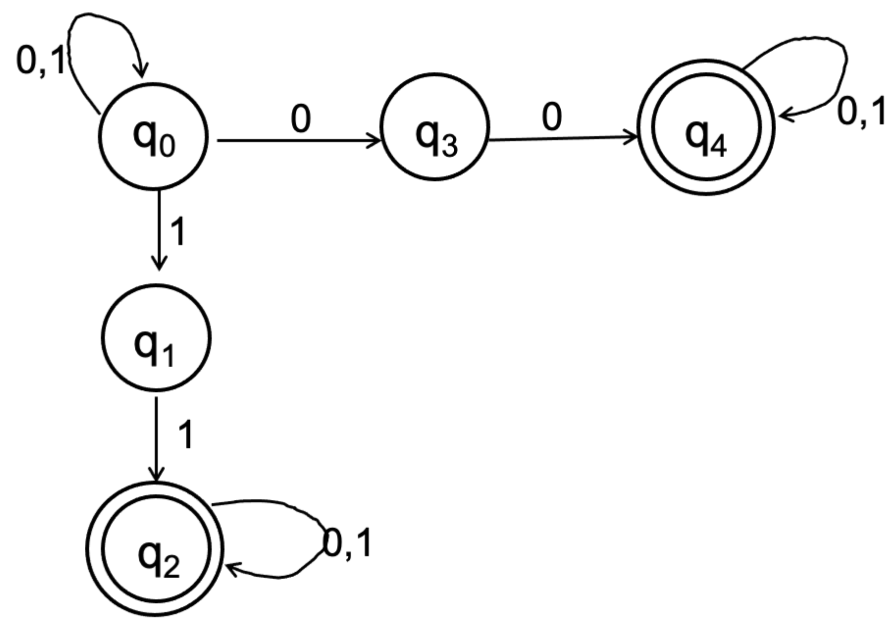
- L = { w | w has two consecutive 0's or w has two consecutive 1's }



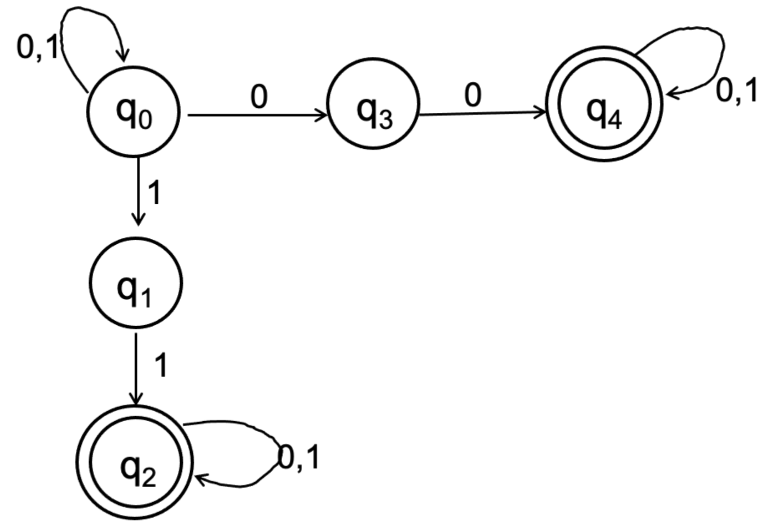
Transition Function for example

Input

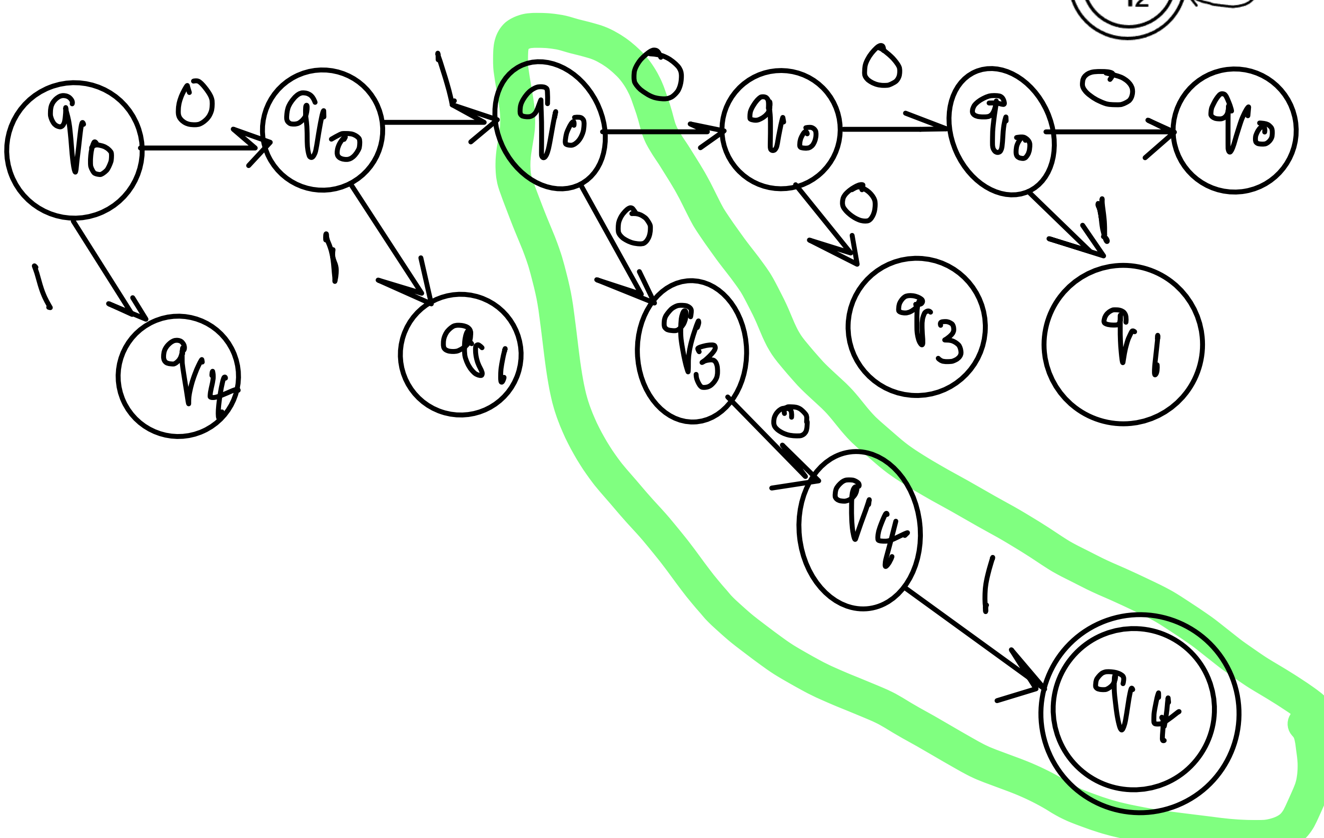
δ	0	1
q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$
q_1	ϕ	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{q_4\}$	ϕ
q_5	$\{q_4\}$	$\{q_4\}$



Moves in NFA



Input: 01001



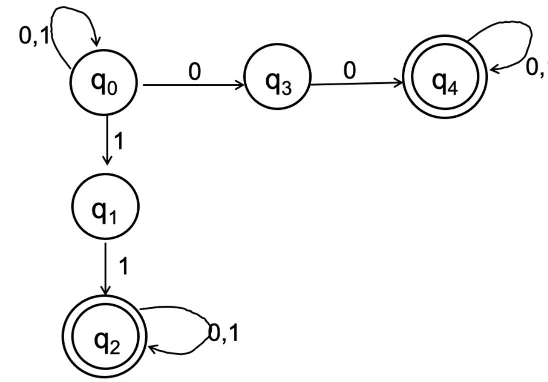
path that leads to a final state

Extended Transition Function of an NFA

- $\delta(q, a)$ is a set of states.
- Extend to strings as follows:
- Basis: $\delta(q, \epsilon) = \{q\}$
- Induction: $\delta(q, wa)$ = the union over all states p in $\delta(q, w)$ of $\delta(p, a)$

$$\delta(q, wa) = \bigcup_{p \epsilon \delta(q, w)} \delta(p_i, a)$$

Extended Transition Function



Input: 01

$$\delta^*(q_0, 01) = \delta(\delta^*(q_0, 0), 1)$$

since $\delta^*(q_0, 0) = \{q_0, q_3\}$

$$\delta(\delta^*(q_0, 0), 1) = \delta(\{q_0, q_3\}, 1)$$

$$= \delta(q_0, 1) \cup \delta(q_3, 1)$$

$$= \{q_0, q_1\} \cup \emptyset$$

$$= \{q_0, q_1\}$$

Example

- Design an NFA M which accepts

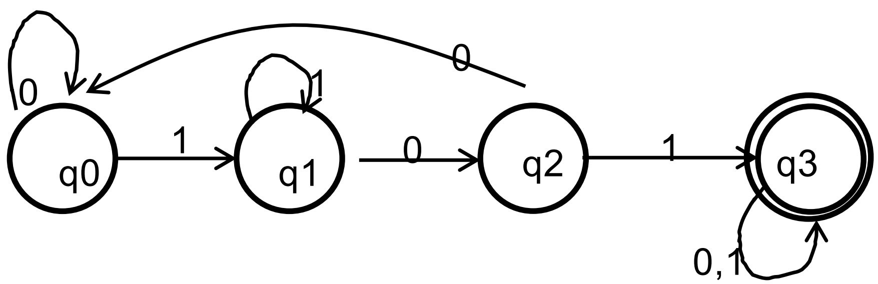
  L(M) = {w | w contains substring 101 or 010 and w is a binary string }

two conditions (1) contains substring 101

OR

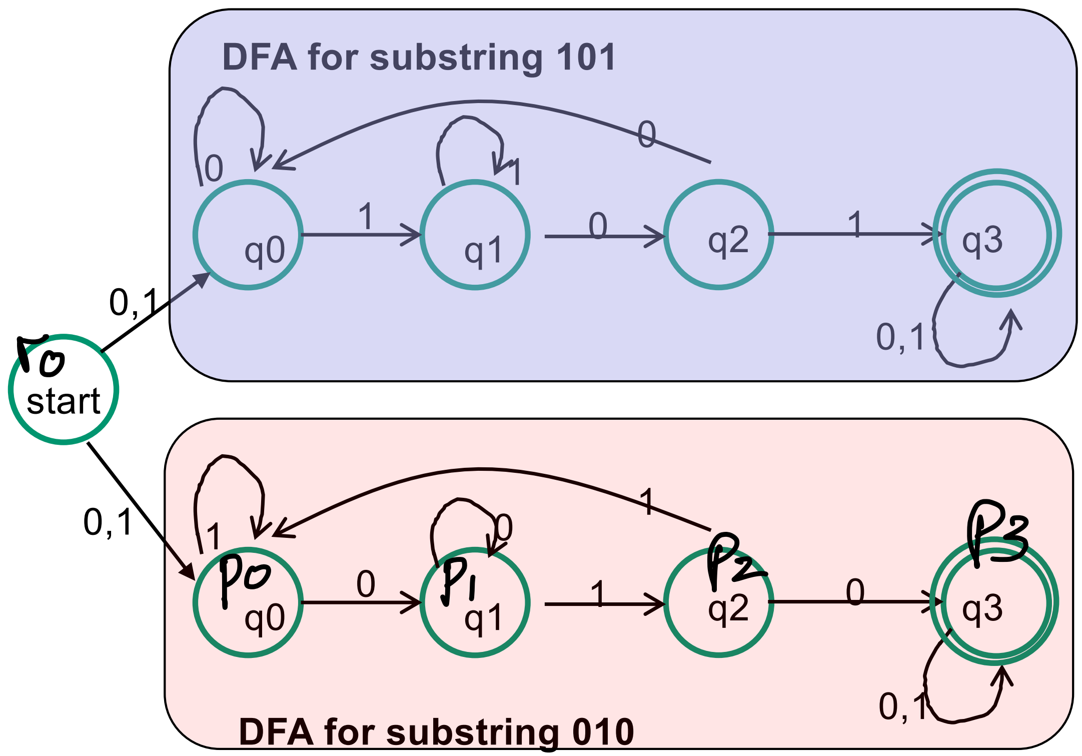
(2) contains substring 010

Example: DFA that recognizes Substring 101



q0: not read first 1 in substring 101
q1: last input read was a 1, could be start of substring 101
q2: last two inputs read were 10 which is part of substring 101
q3: last three inputs read were 101 which means substring 101 is in input

NFA for substring 101 or 010



w is $\in L(M)$

iff.

either

$\delta(r_0, w) = q_3$

or

$\delta(r_0, w) = p_3$

Breakout Group Exercises: Submit with names of all group members on the solution

- Design an NFA N that accepts the language

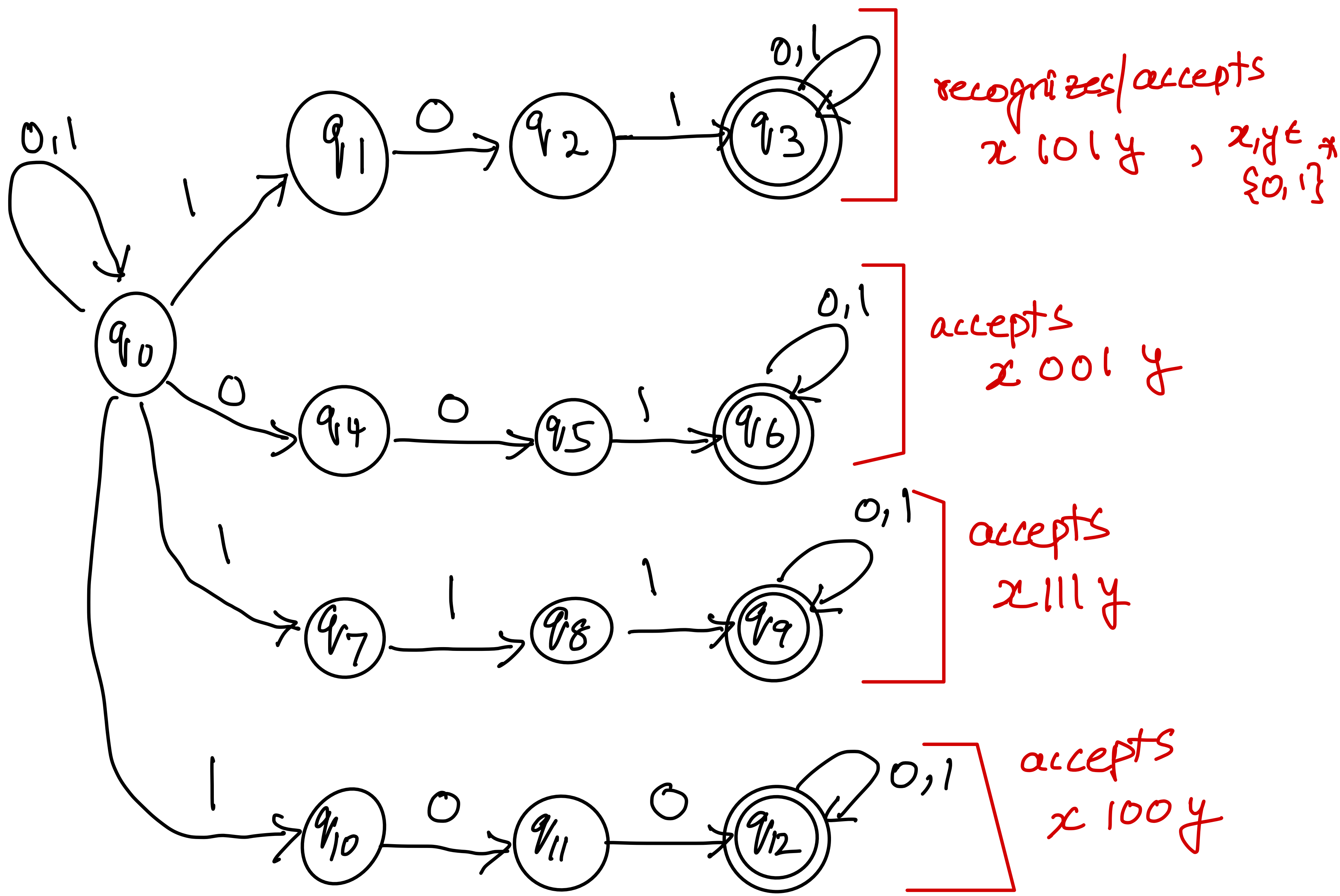
  L = { w | w is a string in $\{0,1\}^*$ and w contains the substring 101 with *at most* 1-bit position of mis-match.

at most one position mismatch $\Rightarrow$

substring can be $\left.\begin{array}{l} 101 \\ \underline{0}01 \\ 1\underline{1}1 \\ 10\underline{0} \end{array}\right\}$ 4 cases.

$w = x\,101\,y$ or $x\,001\,y$ or $x\,111\,y$ or $x\,100\,y$

$x, y \in \{0,1\}^*$


0,1
1
0
1
q1
q2
q3
0,1
q0
0
0
1
q4
q5
q6
0,1
1
1
1
q7
q8
q9
0,1
1
0
0
q10
q11
q12
0,1
recognizes/accepts
x 101 y , x,y ∈ {0,1}*
accepts
x 001 y
accepts
x 111 y
accepts
x 100 y

Equivalence of DFA's, NFA's

- A DFA can be turned into an NFA that accepts the same language.
- If $\delta_D(q, a) = p$, let the NFA have $\delta_N(q, a) = \{p\}$.
- Then the NFA is always in a set containing exactly one state – the state the DFA is in after reading the same input.

Equivalence – (2)

- Surprisingly, for any NFA there is a DFA that accepts the same language.
- Proof is the *subset construction*.
- The number of states of the DFA can be exponential in the number of states of the NFA.
- Thus, NFA's accept exactly the regular languages.
- Importance of a constructive proof…..
  - The procedure to construct a DFA from the NFA provides us with an algorithm we can use to automate the process!

Subset Construction

- Given an NFA with states Q, inputs Σ, transition function δ_N, state state q_0, and final states F, construct equivalent DFA D with:
  - States 2^Q (Set of subsets of Q).
  - Inputs Σ.
  - Start state $\{q_0\}$.
  - Final states = all those with a member of F.
- The transition function δ_D is defined by:

$\delta_D(\{q_1,\ldots,q_k\}, a)$ is the union over

all $i = 1,\ldots,k$ of $\delta_N(q_i, a)$.

Critical Point

- The DFA states have *names* that are sets of NFA states.
- But as a DFA state, an expression like {p,q} must be understood to be a single symbol, not as a set.
- Analogy: a class of objects whose values are sets of objects of another class.

Proof of Equivalence - 1

NFA $N = (Q, \Sigma, \delta_N, q_0, F)$

define DFA $M = (Q', \Sigma, \delta_D, q_0', F')$

$Q' = 2^Q$ – all subsets of Q

Let's denote element of 2^Q as $[q_{i1}, q_{i2}, \ldots, q_{ik}]$
to denote set $\{q_{i1}, q_{i2}, \ldots, q_{ik}\}$

ex: $[q_0 q_1]$ denotes $\{q_0, q_1\}$

$q_0' = [q_0]$ F' is set of states in Q' that contain a state in F.

Define $\delta_D([q_1, q_2, \ldots, q_i], a) = [p_1, p_2, \ldots, p_j]$
if and only if
$\delta_N(\{q_1, q_2, \ldots, q_i\}, a) = \{p_1, p_2, \ldots, p_j\}$
[applying δ_N to each element in $\{q_1, q_2, \ldots, q_i\}$]

Proof of Equivalence: Subset Construction

- The proof is almost a pun.
- Show by induction on |w| that

$$\delta_N(q_0, w) = \delta_D(\{q_0\}, w)$$

- Basis: $w = \epsilon$: $\delta_N(q_0, \epsilon) = \delta_D(\{q_0\}, \epsilon) = \{q_0\}$.

Proof of Equivalence - 2

Induction - Assume true for $|w| = n$, consider $w = xa$ and $|x| = n$.

Then $\delta_D(q_0', xa) = \delta_D(\delta_D([q_0], x), a)$ by definition of extended δ_D

But by definition of δ_D and Ind. hypothesis

$$\delta_D([p_1, p_2, \ldots, p_j], a) = [r_1, r_2, \ldots, r_k] \text{ iff.}$$

$$\delta_N(\{p_1, p_2, \ldots, p_j\}, a) = \{r_1, r_2, \ldots, r_k\}.$$

Therefore $\delta_D(q_0', xa) = [r_1 r_2 \ldots r_k]$ iff.

$$\delta_N(q_0, xa) = \{r_1, r_2, \ldots, r_k\}$$

Proof of Equivalence - 3

To complete proof of acceptance of same language -

$\delta_D(q_0', w) \in F'$ exactly when

$\delta_N(q_0, w)$ contains state $q \in F$.

$\therefore L(M_{NFA}) = L(M_{DFA})$.

Each state in DFA denotes set of states in NFA

$\Rightarrow$ DFA's state "simulates" all possible states

that the NFA could be in.

Induction

- Assume IH for strings shorter than w.
- Let w = xa; IH holds for x.
- Let $\delta_N(q_0, x) = \delta_D(\{q_0\}, x) = S$.
- Let T = the union over all states p in S of $\delta_N(p, a)$.
- Then $\delta_N(q_0, w) = \delta_D(\{q_0\}, w) = T$.

Example

DFA $M' = (Q', \Sigma, \delta_D, q_0', F')$

$Q' = 2^Q : \{ \phi, [q_0], [q_1], [q_1 q_2] \}$ $\quad q_0' = [q_0]$

$\delta'([q_0'], 0) = [q_0 q_1]$

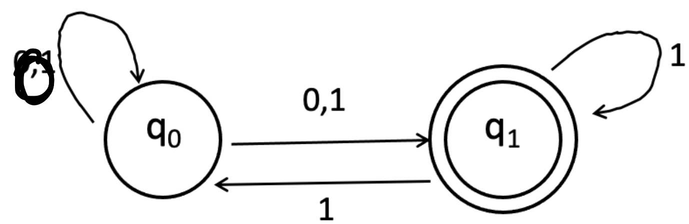
$\delta'([q_0], 1) = [q_1]$

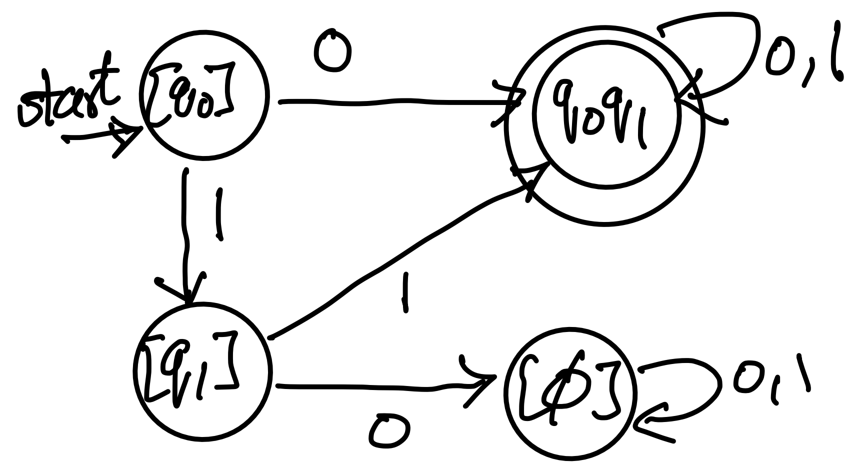
$\delta'([q_1], 0) = \phi$

$\delta'([q_1], 1) = [q_0 q_1]$

$\delta'([q_0 q_1], 0) = [q_0 q_1]$

$\delta'([q_0 q_1], 1) = [q_0 q_1]$

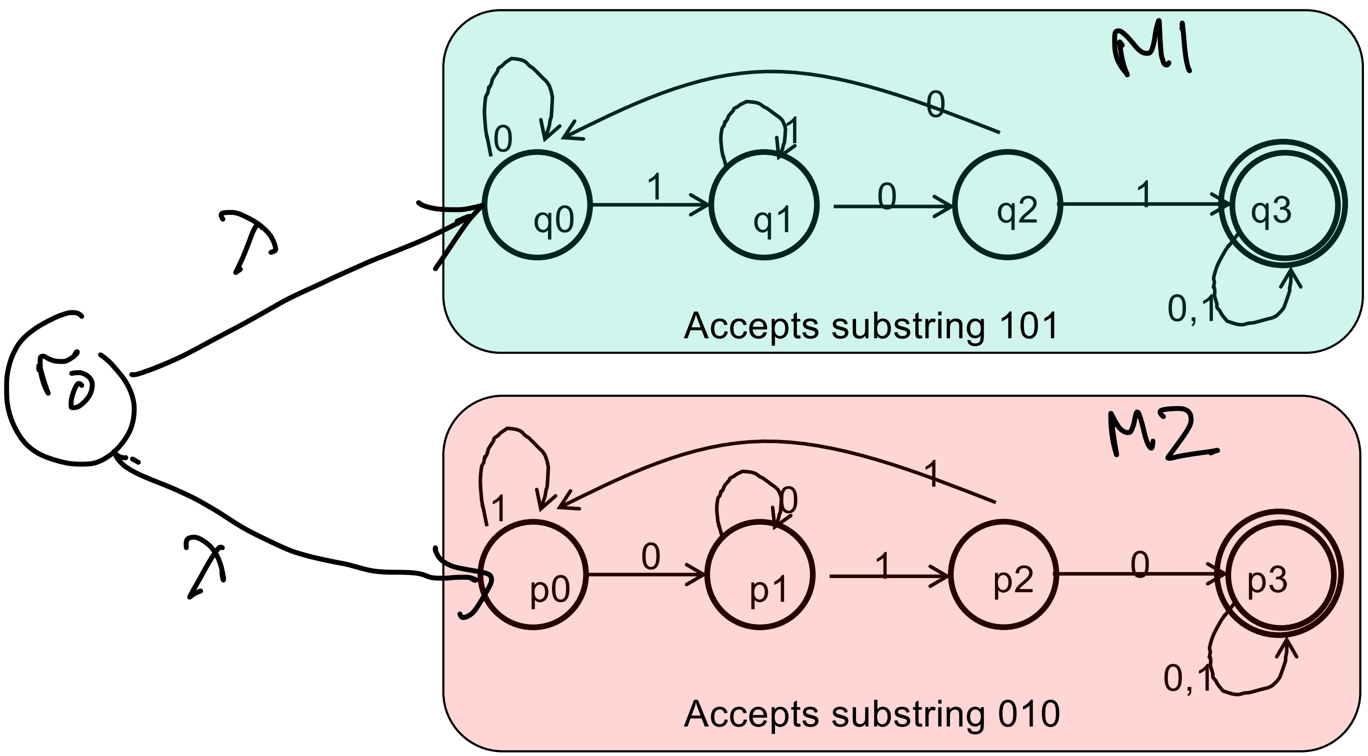




Extending the NFA model: NFA's With ϵ-Transitions

- We allow state-to-state transitions on empty string input λ (also denoted ϵ) .
- These transitions are done spontaneously, without looking at the input string.
- A convenience at times, but still only regular languages are accepted.
  - Allowing λ-transitions can make it easier to define and build the automaton
- Analogous to program going to several next states before reading the next input

Example: recognizes Substring 101 or 010

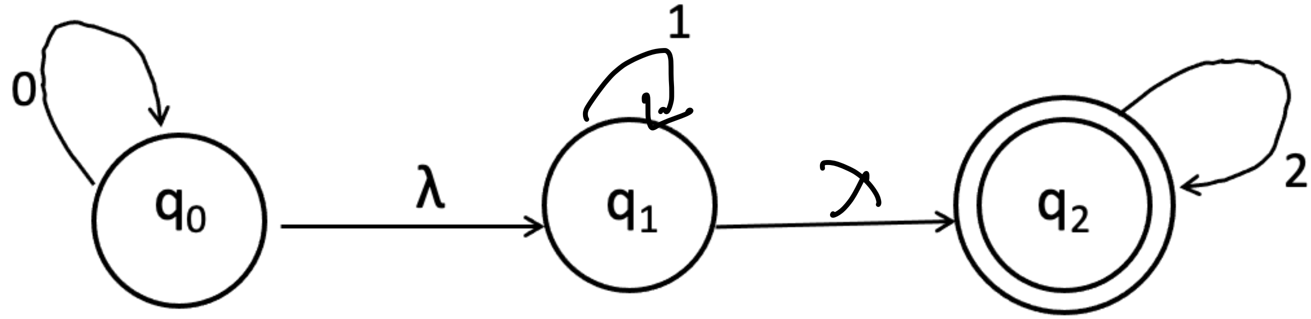


start in r_0, jump to both M_1 and M_2 in parallel

w is accepted in state = q_3 or p_3

Example 1:

- L = {w | w has 0's followed by 1's followed by 2's}



Any advantage to NFA model with empty string input ?

- w is a string in L1 or L2
  - Construct M1 for L1 and M2 for L2, then on E-transition from start go to both M1 and M2

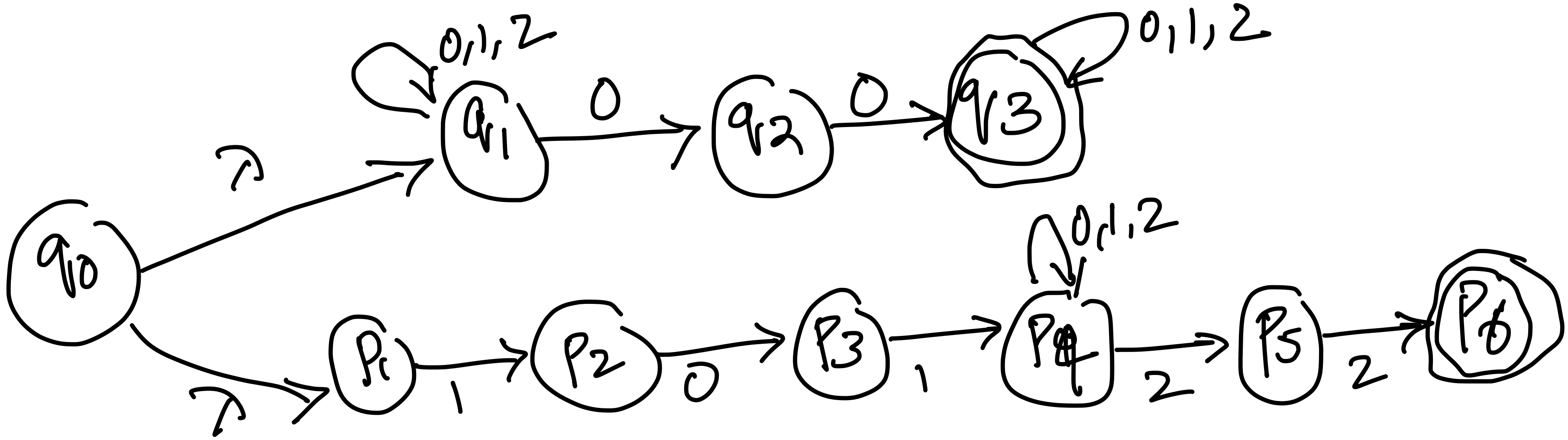
- w is a string x.y where x is in L1 and y is in L2; x has 00 and y has 11
  - Construct M1 for L1 and M2 for L2, start in M1 and if it goes to final state then start M2.

- What are we doing here…..simplification of the language/problem

Exercise:

- Provide an NFA (with E moves) that accepts the language L over alphabet {0,1,2} where

L = { w | (a) w=x and x has two consecutive 0's or (b) w=y and y has substring 101 and ends with two 2's }



Equivalence of NFAs and DFAs

- Adding moves on empty string seems to add expressive power

- But the two models are equivalent, L(M) = L(N)

Paths in the NFA and Concept of E-Closure

- A path from state p to state q is labelled with symbols from alphabet OR labeled with empty string
- To transform an NFA with E-moves to an NFA (or DFA) without E-moves, of particular interest are paths labeled with empty string
  - An edge labeled with empty string implies from a state q, we can go to another state p without reading an input
- **Definition:** E-closure of a state = Path where all edges are labeled with empty string

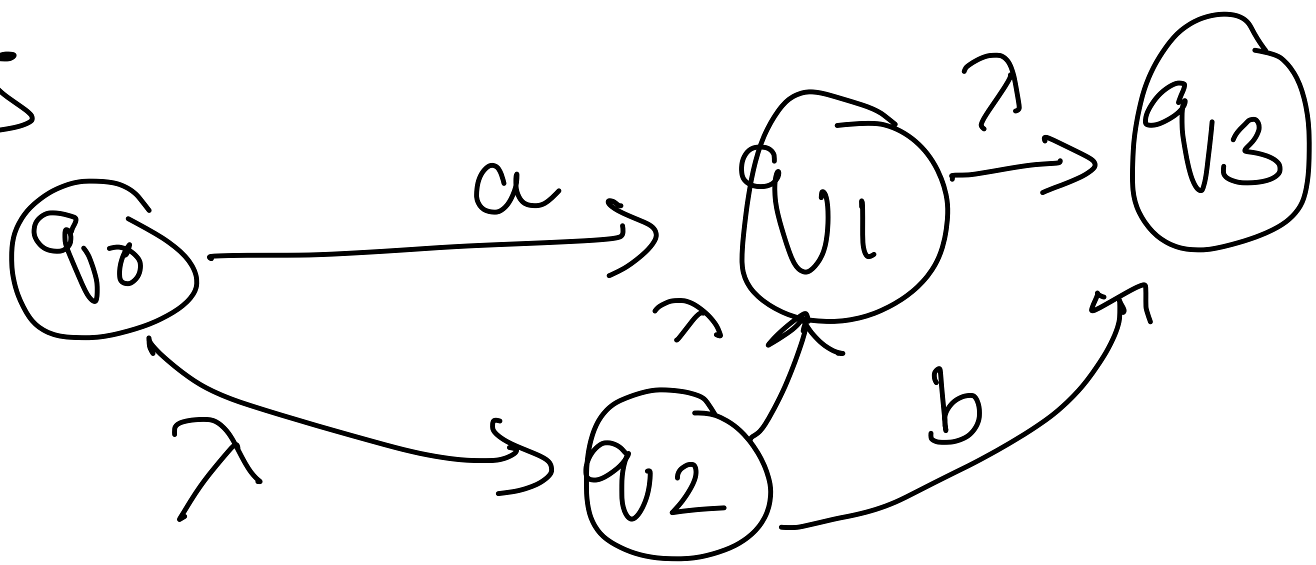
E-Closure: Definition

- E-Closure(q) = set of states p that you can reach from q following only edges labeled with empty string
- Can extend E-Closure to set of states:

For a set of states P,

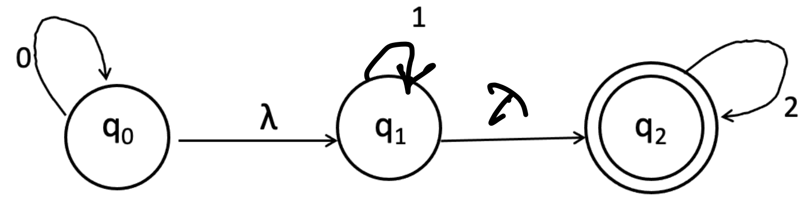
E-Closure(P) = $\bigcup_{q \in P}$ E-closure(q)

$P = \{q_1, q_2\}$



E-Closure: Example

$E\text{-}Closure(q_0) = \{q_0, q_1, q_2\}$



$E\text{-}Closure(q_1) = \{q_1, q_2\}$

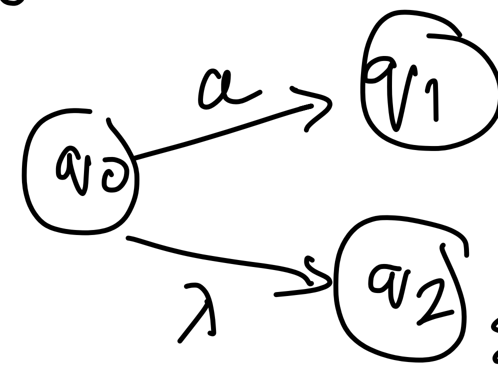
Input : 122

$$q_0 \xrightarrow{\lambda} q_1 \xrightarrow{1} q_1 \xrightarrow{2} q_2 \xrightarrow{2} q_2$$

122 is accepted by M

Extended Delta in NFA with E-moves

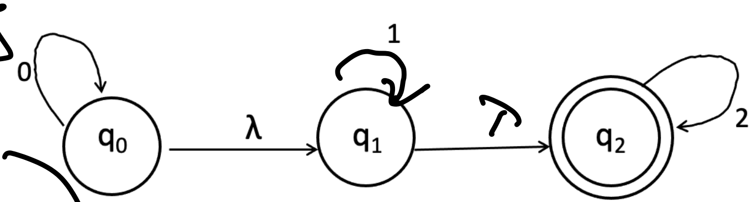
- Intuition: $\hat{\delta}(q, w)$ is the set of states you can reach from q following a path labeled w. $\lambda \cdot w \cdot \lambda = w$
- Basis: $\hat{\delta}(q, \epsilon)$ = CL(q). $|w| = 0$
- Induction: $\hat{\delta}(q, xa)$ is computed by:



$\{q_0, q_2\}$

1. Start with $\hat{\delta}$ (q, x) = S.
2. Take the union of CL(δ(p, a)) for all p in S.

$$\hat{\delta}(q_0, \lambda) = E\text{-}Cl(q_0) = \{q_0 q_1 q_2\}$$



$$\hat{\delta}(q_0, 0) = E\text{-}Cl(\delta(\hat{\delta}(q_0, \lambda), 0))$$

$$= E\text{-}Closure(\delta(q_0, 0), \delta(q_0, 1), \delta(q_0, 2))$$

$$\{q_0, q_1, q_2\} = E\text{-}Closure(q_0 \cup \emptyset \cup \emptyset)$$

Equivalence of NFA and ϵ-NFA

- Every NFA is an ϵ-NFA (It just has no transitions on empty string ϵ)
- Every E-NFA is an NFA: requires us to take an ϵ-NFA and construct an NFA that accepts the same language.
  - We do so by combining ϵ–transitions with the next transition on a real input.
- Start with an ϵ-NFA with states Q, inputs Σ, start state q_0, final states F, and transition function δ_E.
- Construct an "ordinary" NFA with states Q, inputs Σ, start state q_0, final states F', and transition function δ_N.

- Compute $\delta_N(q, a)$ as follows:
  1. Let S = E-Closure(q).
  2. $\delta_N(q, a)$ is the union over all p in S of $\delta_E(p, a)$.
- F' = the set of states q such that CL(q) contains a state of F.

NFA and E-NFA Equivalence – (2)

- Prove by induction on |w| that

  $$\text{E-Closure}(\delta_N(q_0, w)) = \hat{\delta}_E(q_0, w).$$

- Thus, the ϵ-NFA accepts w if and only if the "ordinary" NFA does.

Construction of DFA from NFA with E-moves

- Direct constructive proof to go from NFA+E-moves to a DFA
  - Extend the idea of E-closures to define transition function for DFA

- Theorem: Given an NFA $M_E = (Q_E, \Sigma, \delta_E, q_0, F_E)$ there is an equivalent DFA $M_D = (Q_D, \Sigma, \delta_D, q_D, F_D)$

  such that $L(M_E) = L(M_D)$

Proof of Equivalence of NFA+E and DFA

$M_D = (Q_D, \Sigma, \delta_D, q_D, F_D)$

$Q_D = 2^{Q_E}$ power set of Q (all subsets of Q_E)

$q_D = \text{E-Closure}(q_0)$ $\quad F_D = \left\{ S \;\middle|\; \begin{array}{l} S \in Q_D \text{ and} \\ S \cap F_E \neq \emptyset \end{array} \right\}$

$\delta_D(S, a)$ for $a \in \Sigma$, $S \in Q_D$

(a) let $S = \{p_1, p_2, \ldots, p_k\}$

(b) Compute $\bigcup_{p_i \in S} \delta_E(p_i, a) = \{r_1, r_2, \ldots, r_m\}$

(c) then $\delta_D(S, a) = \text{E-Closure}\{r_1, r_2, \ldots, r_m\}$

Equivalence of NFA+E and DFA

$$\hat{\delta}(q_0, w) = \delta_D(q_D, w)$$

Ind. on $|w|$

Basis $|w| = 0$ $\quad \hat{\delta}(q_0, w) = E\text{-}Closure(q_0) =$

$$q_D = EClosure(q_0)$$

Ind. $\quad \hat{\delta}_E(q_0, w) \quad w = x \cdot a \quad |x| = n$

$$\hat{\delta}_E(q_0, w) = EClosure(r_1, r_2, \ldots, r_k)$$

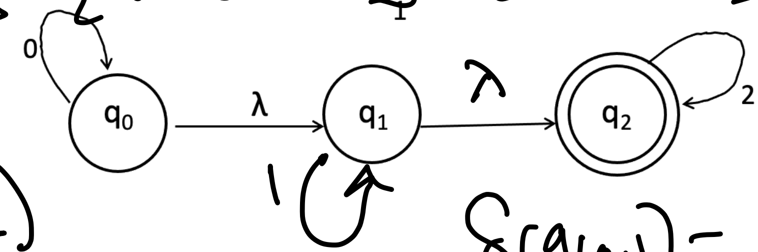
where $\{r_1, r_2, \ldots r_k\} = \bigcup \hat{\delta}_E(p_i, a)$.

I.H. $\rightarrow$ $\hat{\delta}_E(q_0, x) = \{p_1, \ldots, p_m\} = \delta_D(q_0, x)$.

$$= \delta_D(q_0, w).$$

Equivalence of NFA+E and DFA: Example

$\phi \quad \{q_0\} \quad \{q_1\} \quad \{q_2\} \quad \{q_0 q_1\} \quad \{q_1 q_2\} \quad \{q_0 q_1 q_2\} \quad \{q_0 q_2\}$



$q_D = E\text{-Closure}(q_0) = (q_0 q_1 q_2)$

$\delta_D(q_0, 0) = E\text{-Closure}(\{q_0\}) = (q_0 q_1 q_2)$

$\delta(q_0, 1) = \phi$

$\delta_D(q_0, 1) = E\text{-Closure}(\delta_E(q_0, 1)) = \phi = \delta_D(q_0, 2)$

$\delta_D(q_0 q_1, 0) = ECl\{\delta(q_0, 0) \cup \delta(q_1, 0)\} = \{q_0 q_1 q_2\}$

$\cdot\ \delta_D(q_0 q_1, 1) = ECl\{\underbrace{\delta(q_0, 1)}_{\phi} \cup \underbrace{\delta(q_1, 1)}_{\{q_1\}}\} = ECl\{q_1\}$

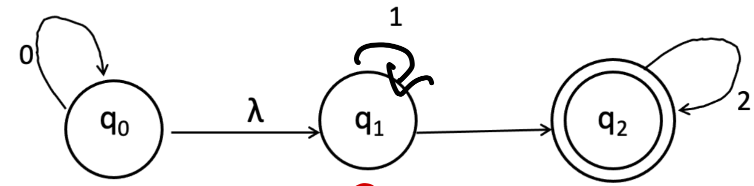
$= \{q_1 q_2\}$

$\delta_D(q_0 q_1, 2) \overset{ECl}{=} \{\delta(q_0, 2) \cup \delta(q_1, 2)\} = ECl(\{q_2\})$

$= \{q_2\}$

Equivalence of NFA+E and DFA: Example

$\delta_D : 2^Q \to 2^Q$



$\delta_D(q_0 q_2, 0) = (q_0 q_1 q_2)$

$\delta_D(q_0, q_2, 1) = \phi$ $\quad$ $\delta_D(q_0, q_2, 2) = (q_2)$ $\quad$ $\delta_D : Q_D \to Q_D$

$\delta_D(q_1 q_2, 0) = \phi$ $\quad$ $\delta_D(q_1 q_2, 1) = (q_1 q_2)$ $\quad$ $Q_D = 2^Q$

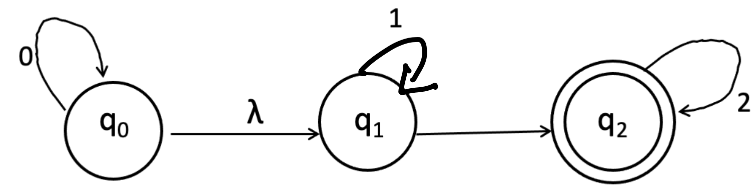
$\delta_D(q_1, q_2, 2) = (q_2)$

$\delta_D(q_0 q_1 q_2, 0) = q_0 q_1 q_2$

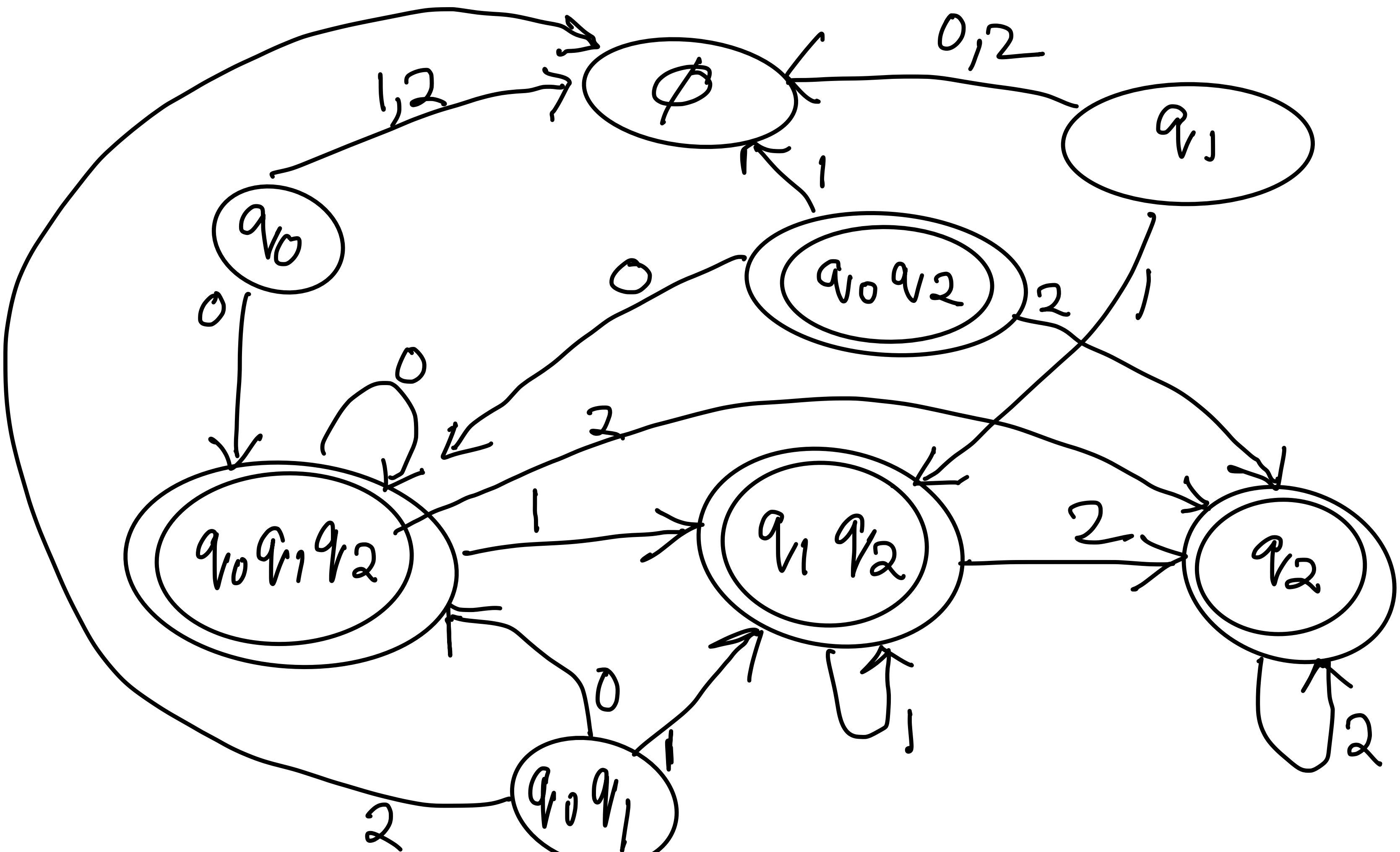
$\delta_D(q_0 q_1 q_2, 1) = E\text{-Closure}(q_1) = q_1 q_2$

$\delta_D(q_0 q_1 q_2, 2) = \phi \cup \phi \cup E\text{-Cl}(q_2) = q_2$

DFA for the example NFA+E

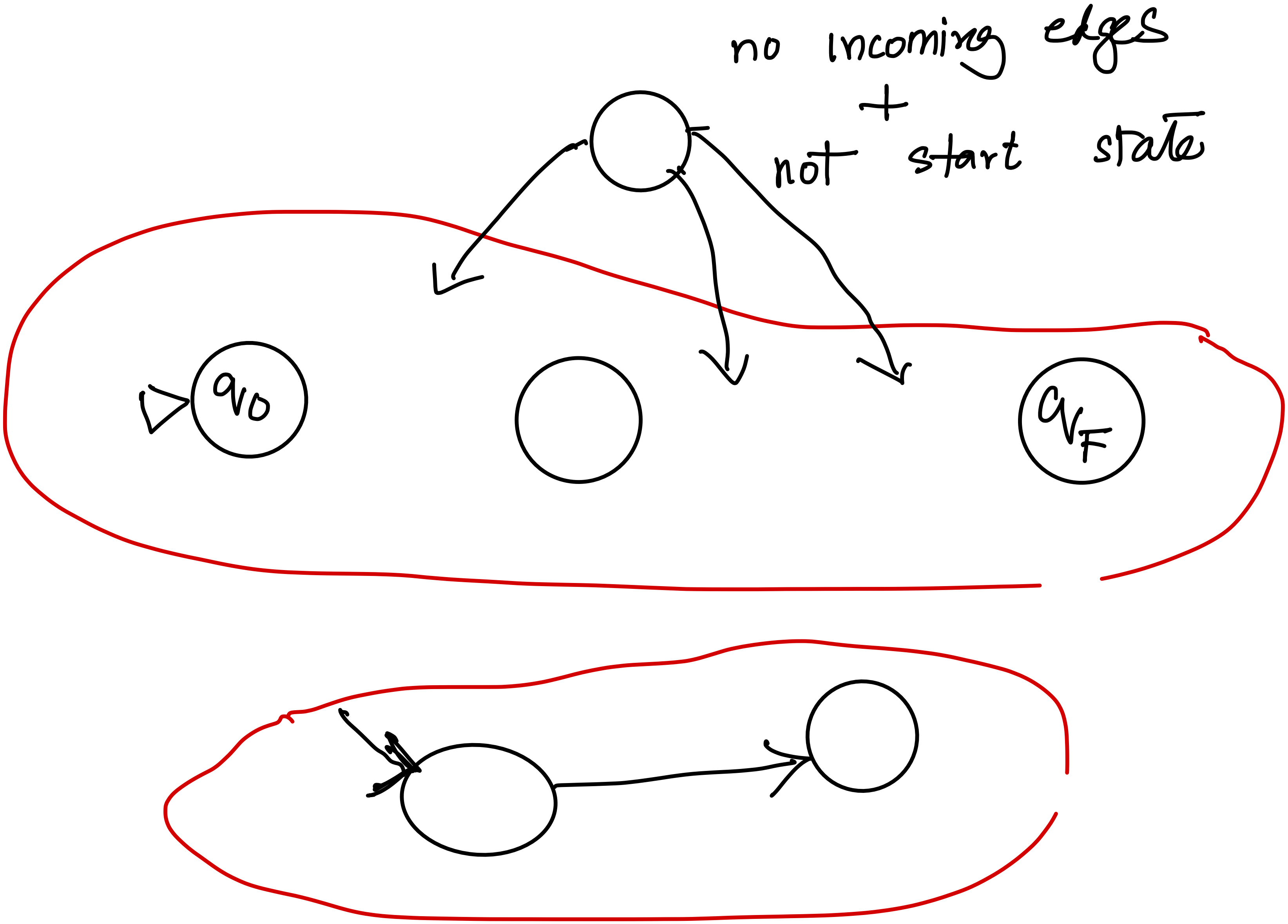


N states $\Rightarrow \lceil \log_2 N \rceil \Rightarrow$ # storage devices



Some Observations

State Minimization



Summary

- DFA's, NFA's, and ϵ–NFA's all accept exactly the same set of languages: the regular languages.
  - NFA = DFA *and* ϵ–NFA = NFA, therefore DFA= ϵ–NFA
- The NFA types are easier to design and may have exponentially fewer states than a DFA.
- But only a DFA can be implemented!

- Algorithms to convert from NFA to DFA…..
  - But could end up with a large number of states….
    - Can we minimize the number of states?

Next…Formal methods to define languages

$(0+1)^* 101 (0+1)^*$

Describe Language

Generate Machine

Machine

- Can we provide formal methods to define a language
  - Instead of defining it as accepted by an automaton ?

- Grammars is one option

- For regular languages, we have a simpler formalism:

Regular Expressions

define a language.

$\{ w \mid w \text{ contains substring } 101 \}$

$\{ w \mid w \text{ starts with } 00 \text{ and ends with } 11 \}$