Pre-lecture brain teaser

Given $\Sigma = \{0, 1\}$, find the regular expression for the language containing all binary strings with an odd number of 0's

Formulate a **language** that describes the above problem.

ECE-374 B: Lecture 3 - DFAs

Lecturer: Nickvash Kani

January 24, 2023

University of Illinois at Urbana-Champaign

Pre-lecture brain teaser

Given $\Sigma = \{0, 1\}$, find the regular expression for the language containing all binary strings with an odd number of 0's

Formulate a **language** that describes the above problem.

$$(01^*0 + 1)^*0(01^*0 + 1)^*$$

First focus on constraint

$$0(00)^*$$

Then focus on generalizability

$$1^*01^*(01^*01^*)^*$$

A simple program

Program to check if an input string w has odd number of 0's

```
int n = 0
While input is not finished
    read next character c
    If (c ≡'0')
        n ← n + 1
endWhile
If (n is odd) output YES
Else output NO
```

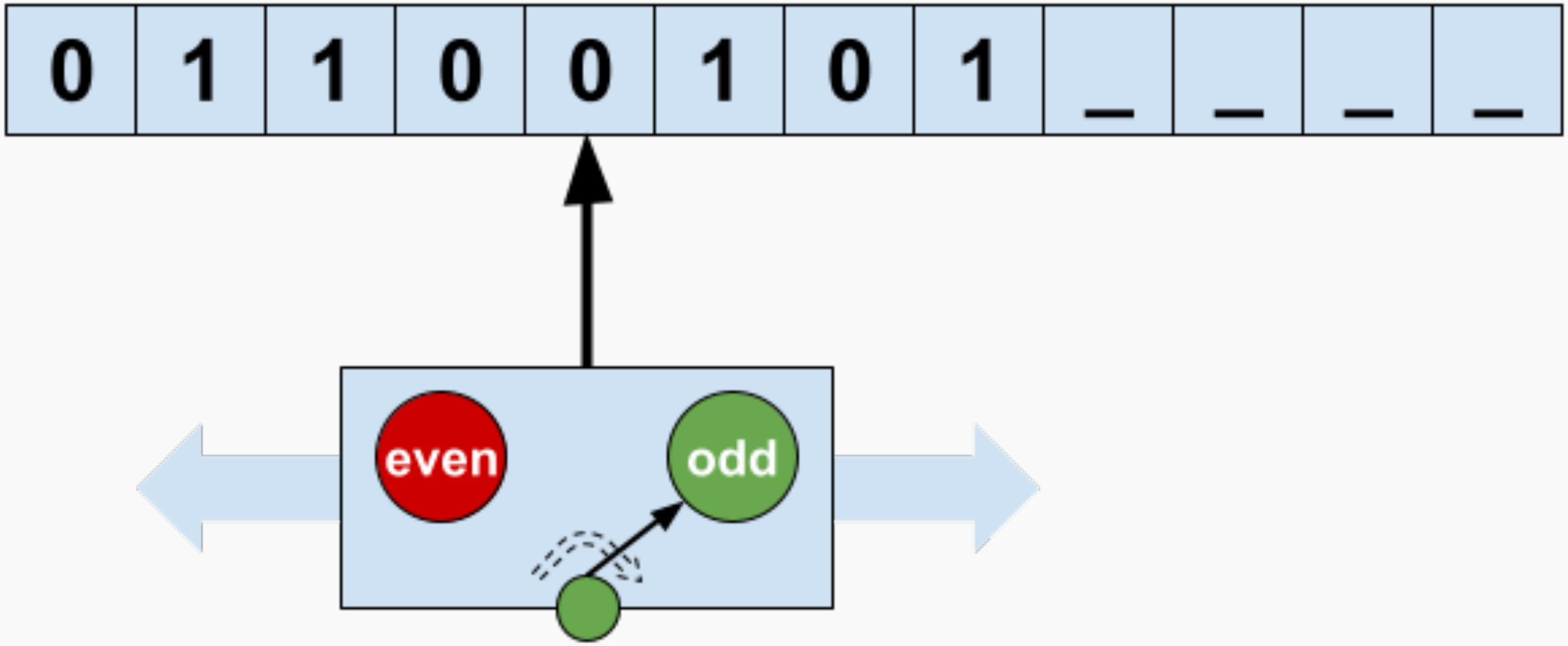
A simple program

Program to check if an input string w has odd number of 0's

```
int n = 0
While input is not finished
    read next character c
    If (c ≡ '0')
        n ← n + 1
endWhile
If (n is odd) output YES
Else output NO
```

```
bit x = 0
While input is not finished
    read next character c
    If (c ≡ '0')
        x ← flip(x)
endWhile
If (x = 1) output YES
Else output NO
```

Another view



- Machine has input written on a read-only tape
- Start in specified start state
- Start at left, scan symbol, change state and move right
- Circled states are accepting
- Machine accepts input string if it is in an accepting state after scanning the last symbol.

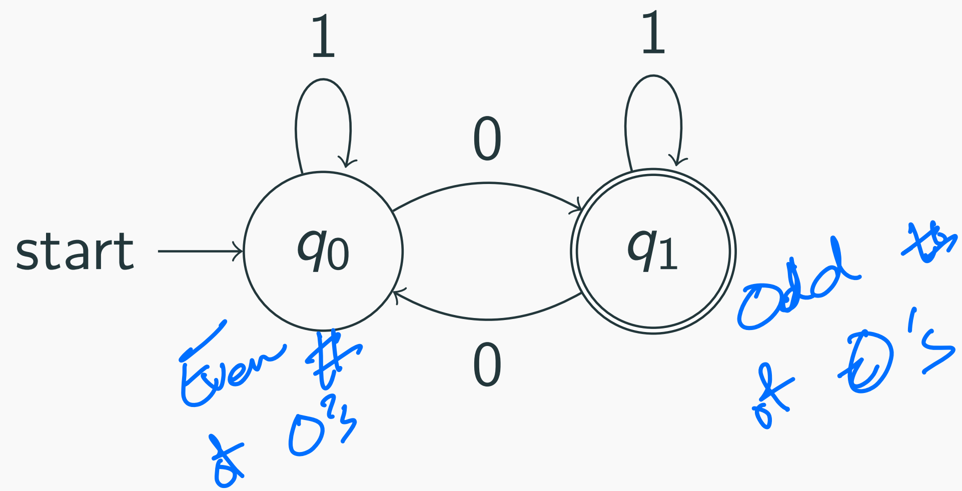
Deterministic-finite-automata (DFA) Introduction

DFAs also called Finite State Machines (FSMs)

- The “simplest” model for computers?
- State machines that are common in practice.
  - Vending machines
  - Elevators
  - Digital watches
  - Simple network protocols
- Programs with fixed memory

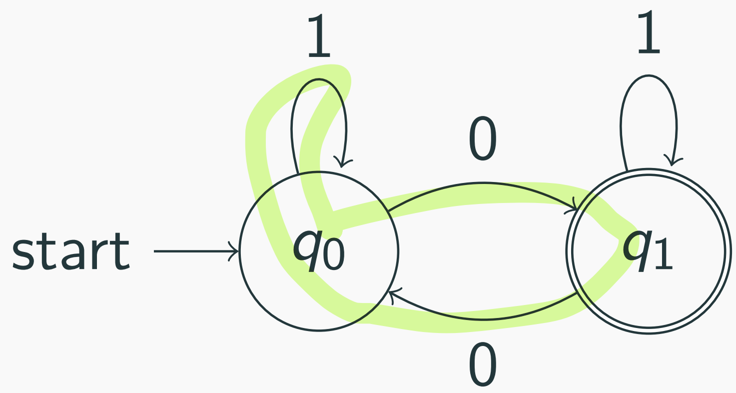
Graphical representation of DFA

Graphical Representation/State Machine



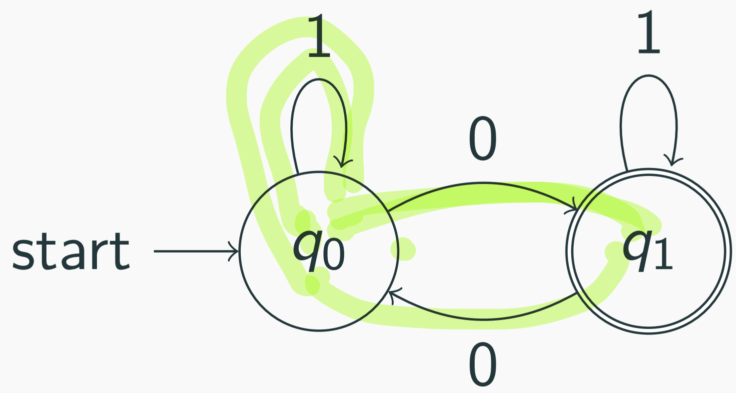
- Directed graph with nodes representing states and edge/arcs representing transitions labeled by symbols in Σ
- For each state (vertex) q and symbol $a \in \Sigma$ there is exactly one outgoing edge labeled by a
- Initial/start state has a pointer (or labeled as s, q_0 or "start")
- Some states with double circles labeled as accepting/final states

Graphical Representation



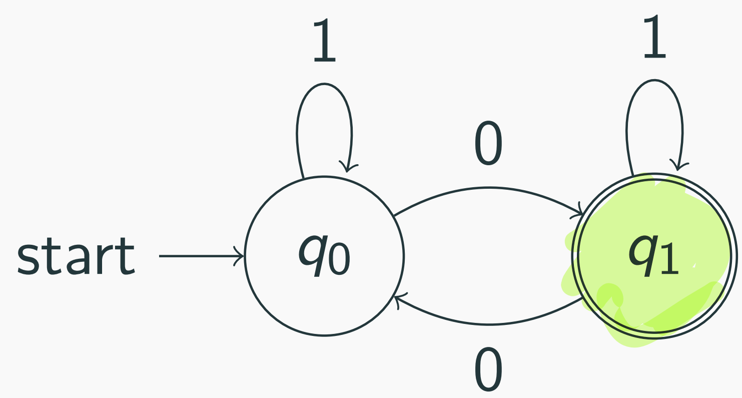
- Where does 001 lead? q_0

Graphical Representation



- Where does 001 lead?
- Where does 10010 lead?

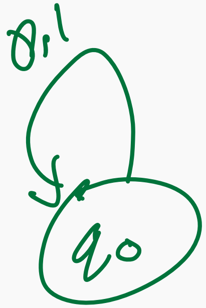
Graphical Representation

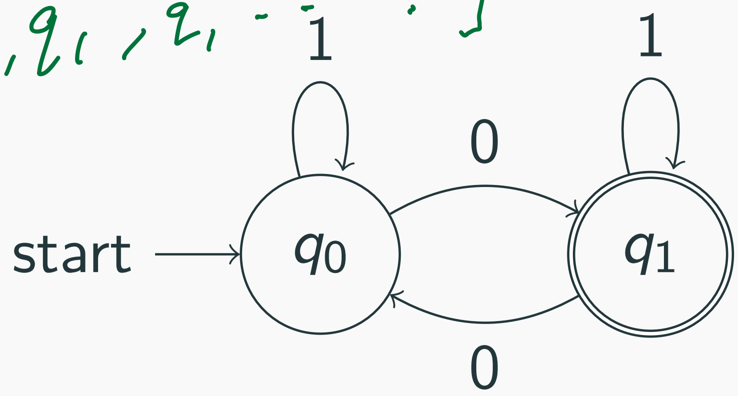


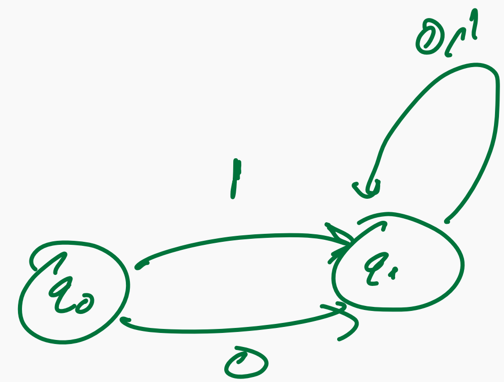
- Where does 001 lead?
- Where does 10010 lead?
- Which strings end up in accepting state? Odd # of 0's

Graphical Representation

walk = [q_0, q_1, q_0, q_1, q_1, ...]

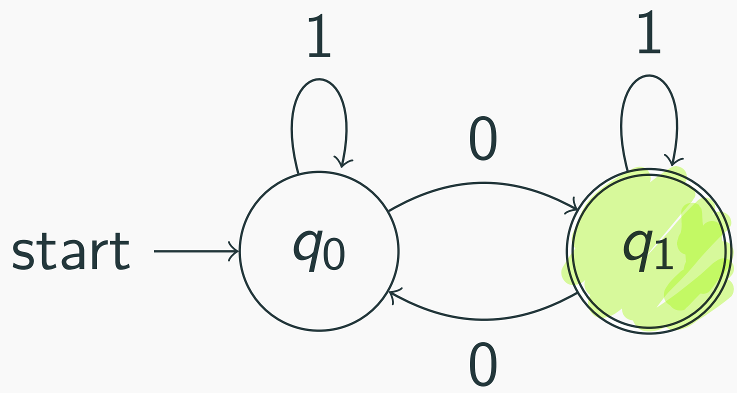






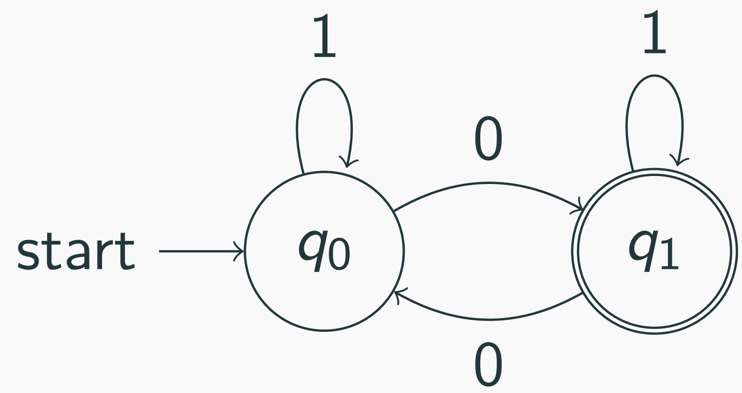
- Where does 001 lead?
- Where does 10010 lead?
- Which strings end up in accepting state?
- Every string w has a unique walk that it follows from a given state q by reading one letter of w from left to right.

Graphical Representation



Definition
A DFA M accepts a string w iff the unique walk starting at the start state and spelling out w ends in an accepting state.

Graphical Representation



Definition
A DFA M accepts a string w iff the unique walk starting at the start state and spelling out w ends in an accepting state.

Definition
The language accepted (or recognized) by a DFA M is denote by $L(M)$ and defined as: $L(M) = \{w \mid M \text{ accepts } w\}$.

$L(r)$

$r \rightarrow$ regular expression

$M \rightarrow$ DFAs $\quad N \rightarrow$ NFAs $\quad TM \rightarrow$ Turing Machines

$G \rightarrow$ grammar

$P \rightarrow$ Pushdown automata

Formal definition of DFA

Formal Tuple Notation

Definition
A deterministic finite automata (DFA) $M = (Q, \Sigma, \delta, s, A)$ is a five tuple where

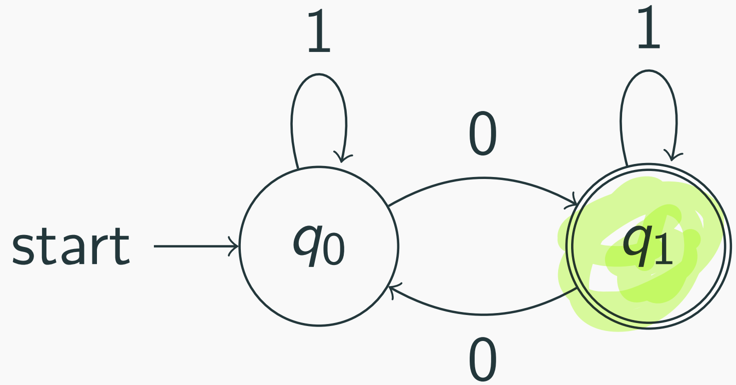
- Q is a finite set whose elements are called states,
- Σ is a finite set called the input alphabet,
- $\delta : Q \times \Sigma \to Q$ is the transition function, $\delta(q_x, c_i) = q_y$
- $s \in Q$ is the start state,
- $A \subseteq Q$ is the set of accepting/final states.

Common alternate notation: q_0 for start state, F for final states.

DFA Notation

$$M = (\overbrace{Q}^{\text{set of states}} , \underbrace{\Sigma}_{\text{alphabet}} , \overbrace{\delta}^{\text{transition function}} , \underbrace{s}_{\text{start state}} , \overbrace{A}^{\text{accepting states}})$$

Example



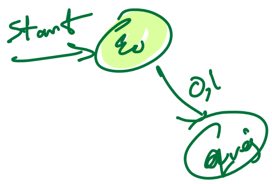
Σ^*

- $Q = \{q_0, q_1\}$
- $\Sigma = \{0, 1\}$
- $\delta =$ $\delta(q_0, 0) = q_1$
  $\delta(q_1, 0) = q_0$
  $\delta(q_0, 1) = q_0$
  $\delta(q_1, 1) = q_1$
- $s = q_0$
- $A = \{q_1\}$

$\delta =$

	0	1
q_0	q_1	q_0
q_1	q_0	q_1

$L = \{\varepsilon\}$



Extending the transition function to strings

Extending the transition function to strings

Given DFA $M = (Q, \Sigma, \delta, s, A)$, $\delta(q, a)$ is the state that M goes to from q on reading letter a

Useful to have notation to specify the unique state that M will reach from q on reading <u>string</u> w

Extending the transition function to strings

Given DFA $M = (Q, \Sigma, \delta, s, A)$, $\delta(q, a)$ is the state that M goes to from q on reading letter a

Useful to have notation to specify the unique state that M will reach from q on reading string w

Transition function $\delta^* : Q \times \Sigma^* \rightarrow Q$ defined inductively as follows:

- $\delta^*(q, w) = q$ if $w = \epsilon$
- $\delta^*(q, w) = \delta^*(\delta(q, a), x)$ if $w = ax$.

$\delta^*(q_0, \text{"}010\text{"}) = \delta^*(\delta(q_0, 0), \text{"}10\text{"}) = \delta^*(q_1, \text{"}10\text{"})$

Formal definition of language accepted by M

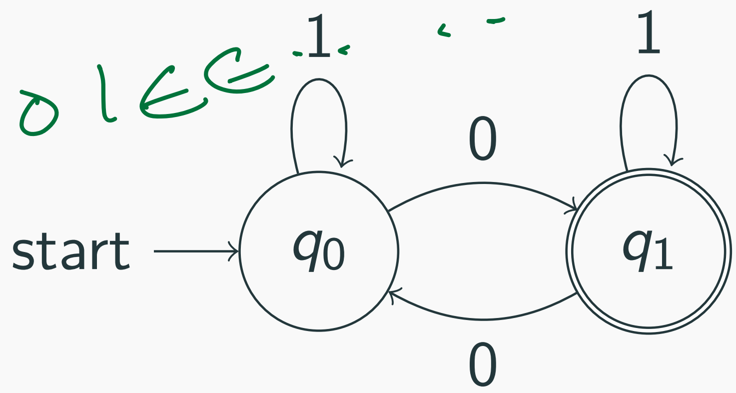
Definition
The language $L(M)$ accepted by a DFA $M = (Q, \Sigma, \delta, s, A)$ is

$$\{w \in \Sigma^* \mid \delta^*(s, w) \in A\}.$$

Example

Input: 0101

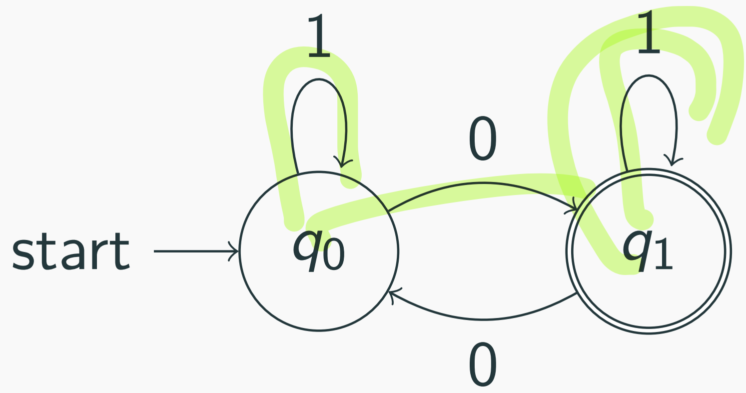
$= 0 \epsilon \epsilon 1 \epsilon 0 1 \epsilon \epsilon$



What is:

- $\delta^*(q_1, \epsilon) =$ q_1

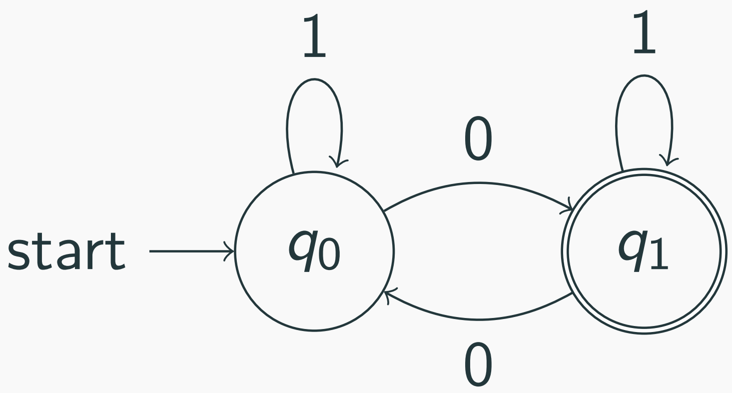
Example



What is:

- $\delta^*(q_1, \epsilon) =$
- $\delta^*(q_0, 1011) =$ q_1

Example



What is:

- $\delta^*(q_1, \epsilon) =$
- $\delta^*(q_0, 1011) =$
- $\delta^*(q_1, 010) =$ q_1

Constructing DFAs: Examples

DFAs: State = Memory

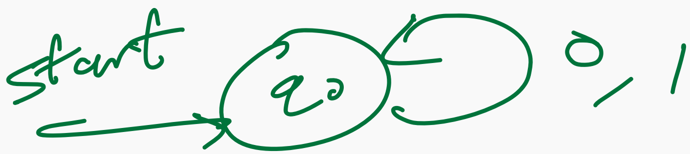
How do we design a DFA M for a given language L? That is $L(M) = L$.

- DFA is a like a program that has fixed number of states regardless of its input size.
- The state must capture enough information from the input seen so far that it is sufficient for the suffix that is yet to be seen (note that DFA cannot go back)

DFA Construction: Example I: Basic languages

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

1. $L = \emptyset$



DFA Construction: Example I: Basic languages

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

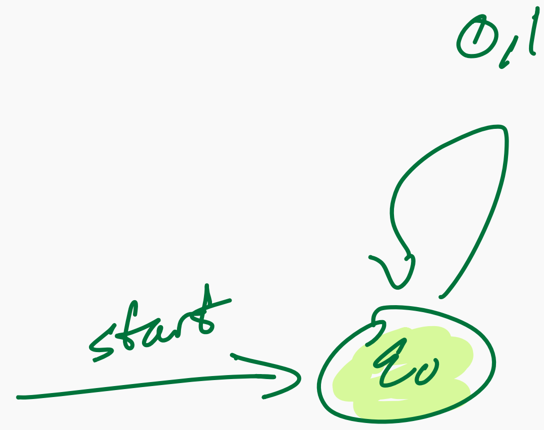
1. $L = \emptyset$

2. $L = \Sigma^*$

DFA Construction: Example I: Basic languages

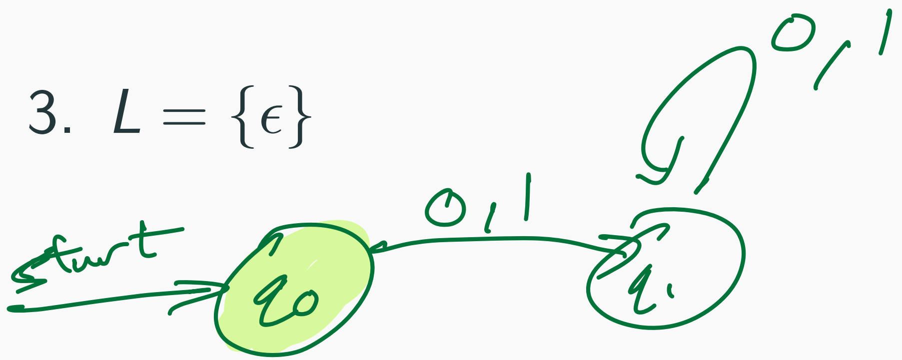
Assume $\Sigma = \{0, 1\}$.

1. $L = \emptyset$



2. $L = \Sigma^*$

3. $L = \{\epsilon\}$

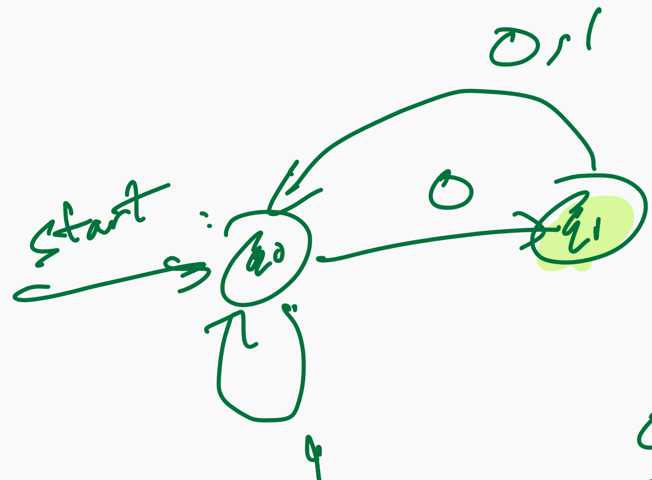


DFA Construction: Example I: Basic languages

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

1. $L = \emptyset$

2. $L = \Sigma^*$



3. $L = \{\epsilon\}$

4. $L = \{0\}$

DFA Construction: Example II: Length divisible by 5

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

$L = \{w \in \{0,1\}^* \mid |w| \text{ is divisible by } 5\}$

DFA Construction: Example III: Ends with 01

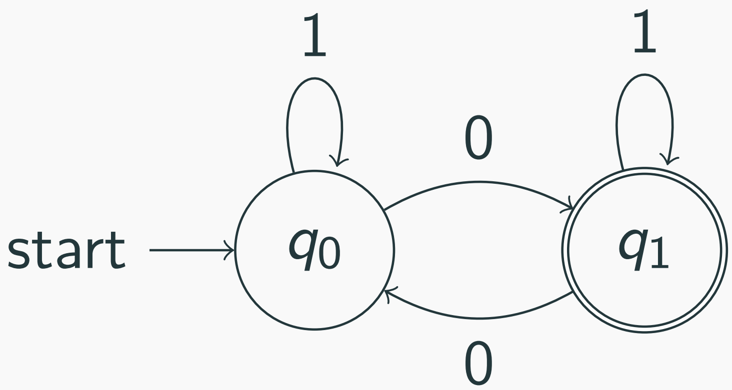
Assume $\Sigma = \{0, 1\}$.

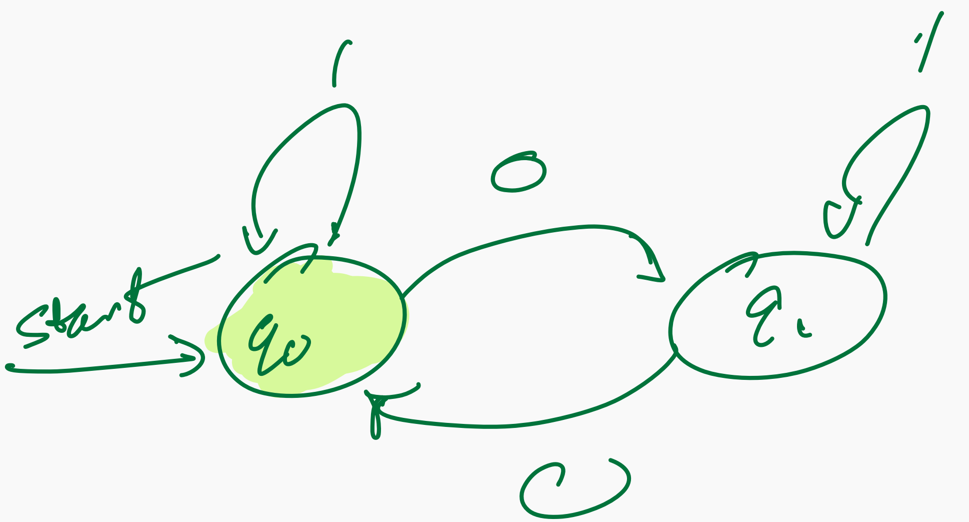
$L = \{w \in \{0,1\}^* \mid w \text{ ends with } 01\}$

Complement language

Complement

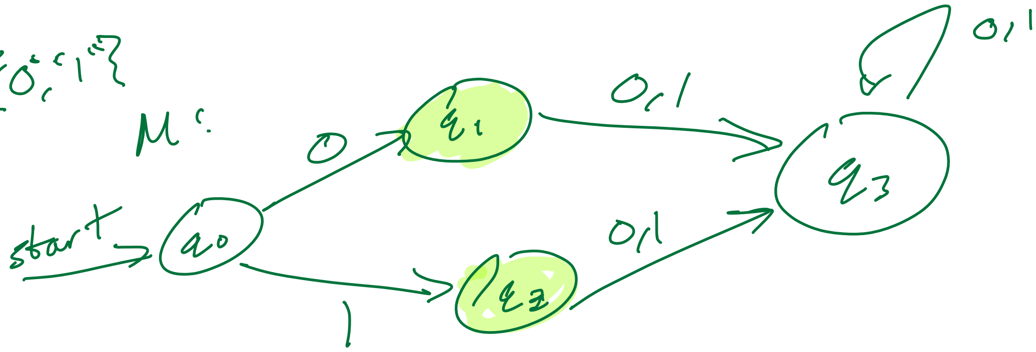
Question: If M is a DFA, is there a DFA M' such that $L(M') = \Sigma^* \setminus L(M)$? That is, are languages recognized by DFAs closed under complement?





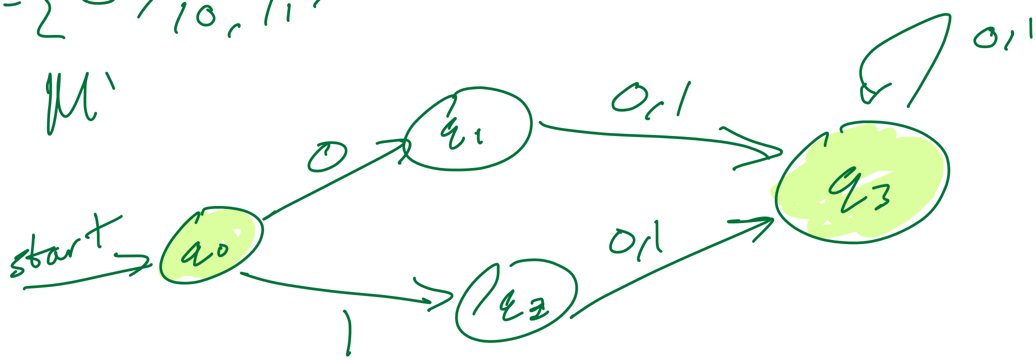
$L(M) = \{"0", "1"\}$

M:



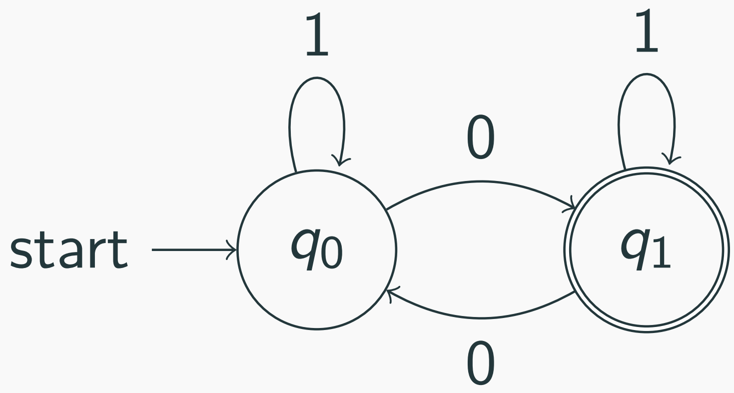
$L(M') = \{\epsilon, 00, 01, 10, 11, \ldots\}$

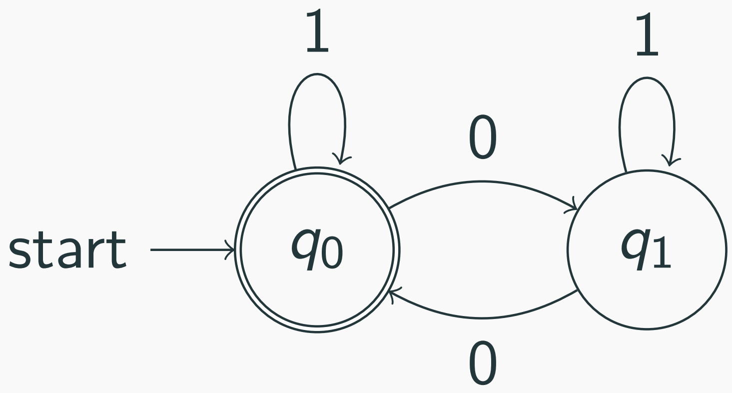
M'



Complement

Just flip the state of the states!





Complement

Theorem
Languages accepted by DFAs are closed under complement.

Complement

Theorem
Languages accepted by DFAs are closed under complement.

Proof.
Let $M = (Q, \Sigma, \delta, s, A)$ such that $L = L(M)$.

Let $M' = (Q, \Sigma, \delta, s, Q \setminus A)$. Claim: $L(M') = \bar{L}$. Why?

$\delta^*_M = \delta^*_{M'}$. Thus, for every string w, $\delta^*_M(s, w) = \delta^*_{M'}(s, w)$.

$\delta^*_M(s, w) \in A \Rightarrow \delta^*_{M'}(s, w) \notin Q \setminus A$.

$\delta^*_M(s, w) \notin A \Rightarrow \delta^*_{M'}(s, w) \in Q \setminus A$. $\square$

Product Construction

Union and Intersection

Are languages accepted by DFAs closed under union? That is, given DFAs M_1 and M_2 is there a DFA that accepts $L(M_1) \cup L(M_2)$?

How about intersection $L(M_1) \cap L(M_2)$?

Union and Intersection

Are languages accepted by DFAs closed under union? That is, given DFAs M_1 and M_2 is there a DFA that accepts $L(M_1) \cup L(M_2)$?

How about intersection $L(M_1) \cap L(M_2)$?

Idea from programming: on input string w

- Simulate M_1 on w
- Simulate M_2 on w
- If both accept than $w \in L(M_1) \cap L(M_2)$. If at least one accepts then $w \in L(M_1) \cup L(M_2)$.

Union and Intersection

Are languages accepted by DFAs closed under union? That is, given DFAs M_1 and M_2 is there a DFA that accepts $L(M_1) \cup L(M_2)$?

How about intersection $L(M_1) \cap L(M_2)$?

Idea from programming: on input string w

- Simulate M_1 on w
- Simulate M_2 on w
- If both accept than $w \in L(M_1) \cap L(M_2)$. If at least one accepts then $w \in L(M_1) \cup L(M_2)$.
- Catch: We want a single DFA M that can only read w once.

Union and Intersection

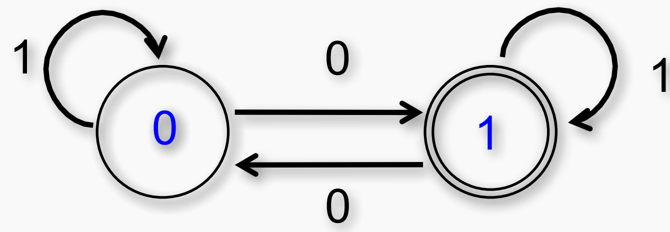
Are languages accepted by DFAs closed under union? That is, given DFAs M_1 and M_2 is there a DFA that accepts $L(M_1) \cup L(M_2)$?

How about intersection $L(M_1) \cap L(M_2)$?

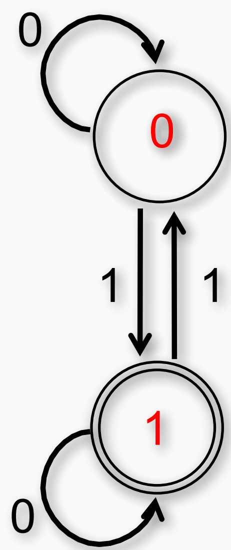
Idea from programming: on input string w

- Simulate M_1 on w
- Simulate M_2 on w
- If both accept than $w \in L(M_1) \cap L(M_2)$. If at least one accepts then $w \in L(M_1) \cup L(M_2)$.
- Catch: We want a single DFA M that can only read w once.
- Solution: Simulate M_1 and M_2 in parallel by keeping track of states of both machines

Example



M_1 accepts #0 = odd

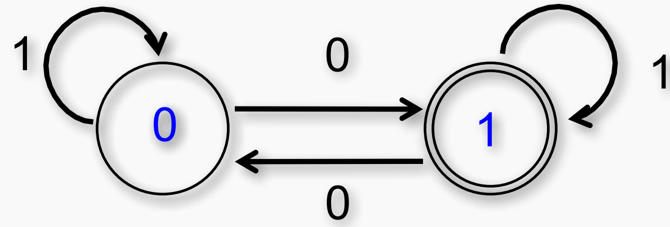


M_2 accepts #1 = odd

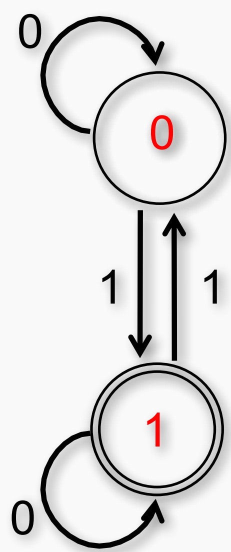
Example

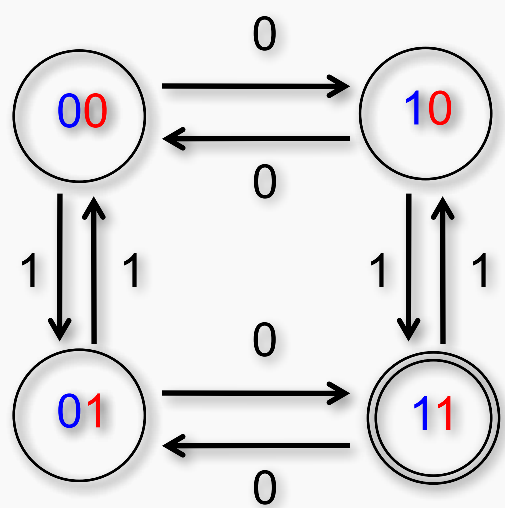
L_1

$L_1 = \{0111, 1011$



M_1 accepts #0 = odd





01

10

111 0

M_2 accepts #1 = odd

Cross-product machine

$L_2 = \{10000$
$1000\}$

10000

Product construction for intersection

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, A_2)$

Theorem
$L(M) = L(M_1) \cap L(M_2)$.

Create $M = (Q, \Sigma, \delta, s, A)$ where

Product construction for intersection

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, A_2)$

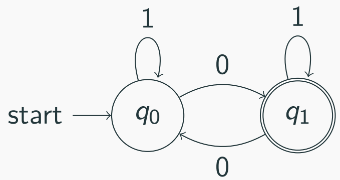
Theorem
$L(M) = L(M_1) \cap L(M_2)$.

Create $M = (Q, \Sigma, \delta, s, A)$ where

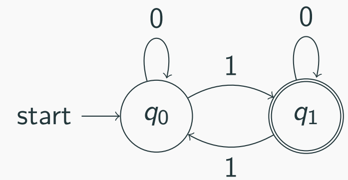
- $Q = Q_1 \times Q_2$
- $s = (s_1, s_2)$
- δ : $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$
- $A = \{(q_1, q_2) \mid q_1 \in A_1 \text{ and } q_2 \in A_2\}$

Intersection vs Union

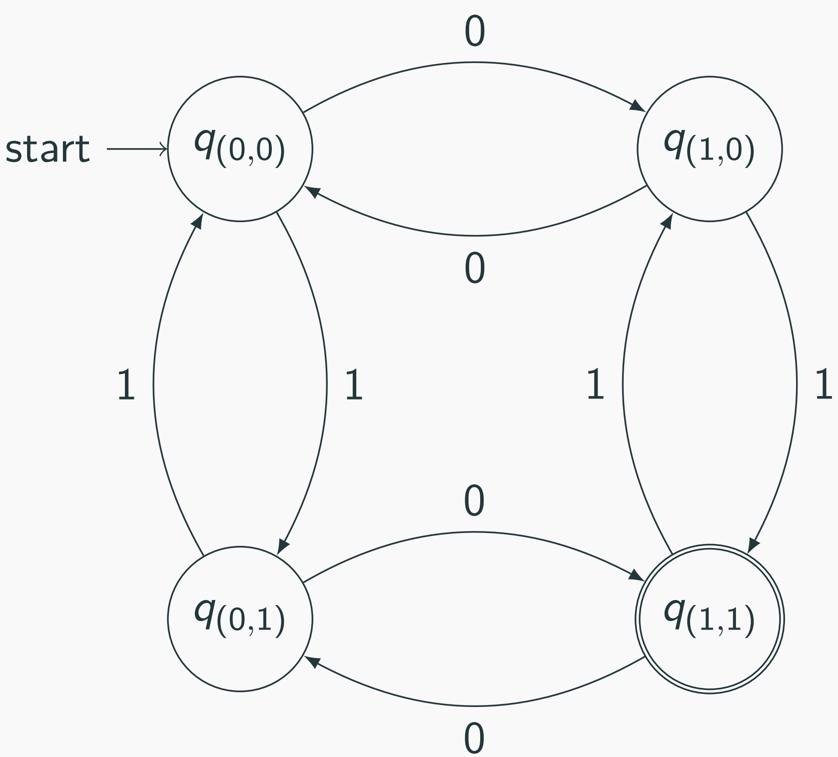
M_1:



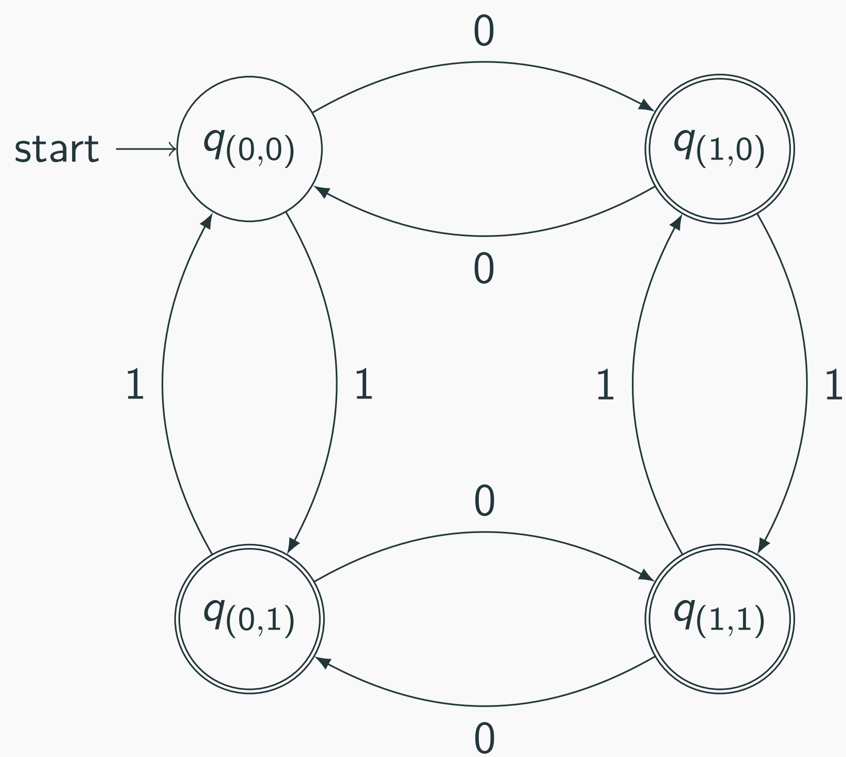
M_2 :



$M_1 \cap M_2$



$M_1 \cup M_2$



Product construction for union

$M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, A_2)$

Theorem
$L(M) = L(M_1) \cup L(M_2)$.

Create $M = (Q, \Sigma, \delta, s, A)$ where

- $Q = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\}$
- $s = (s_1, s_2)$
- $\delta : Q \times \Sigma \to Q$ where

$$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$$

- $A =$ $\{(q_1, q_2) \mid q_1 \in A_1 \text{ or } q_2 \in A_2\}$

Constructing regular expressions

DFAs to regular expressions

Personal Lemma:
Mastering a concept means being able to do a problem in both direction.

Time to reverse problem direction and find regular expressions using DFAs.

Multiple methods but the ones I'm focusing on:

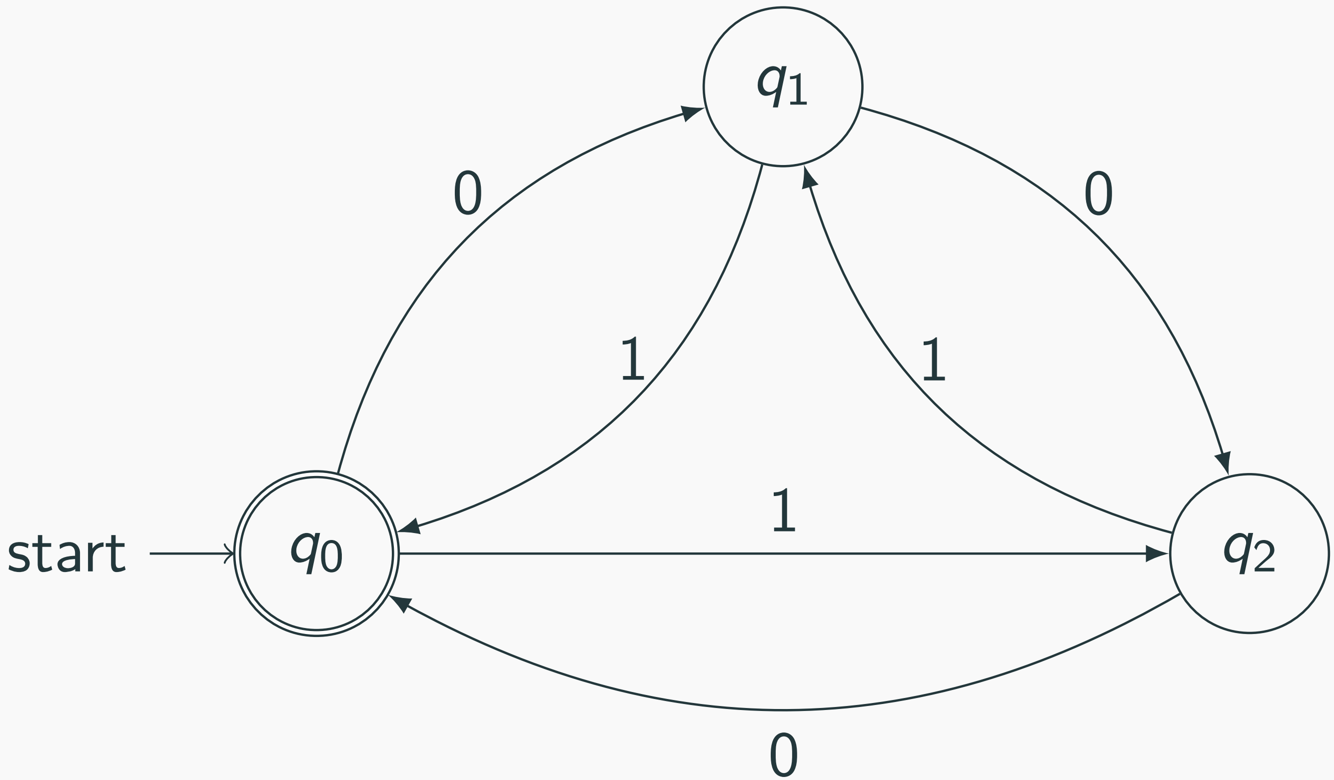
- State removal method
- Algebraic method

State Removal method

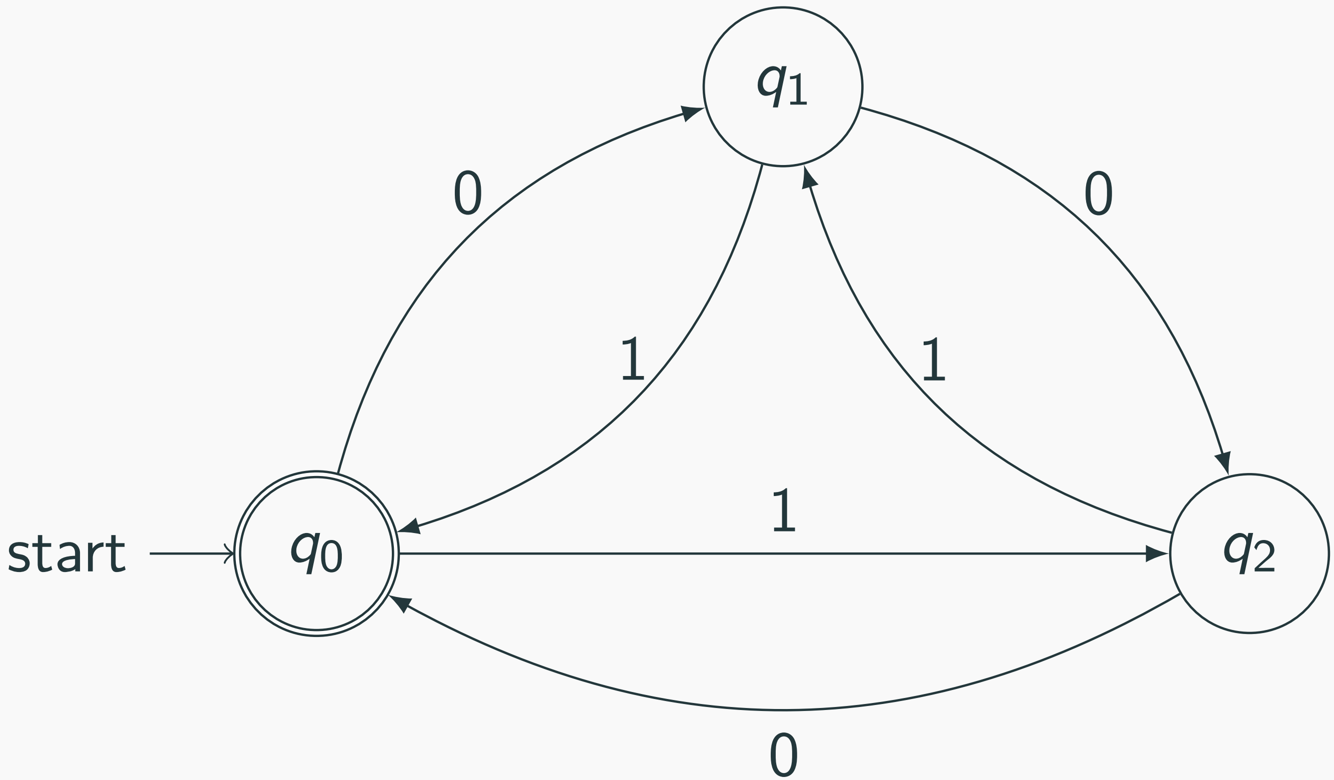
If $q_1 = \delta(q_0, x)$ and $q_2 = \delta(q_1, y)$

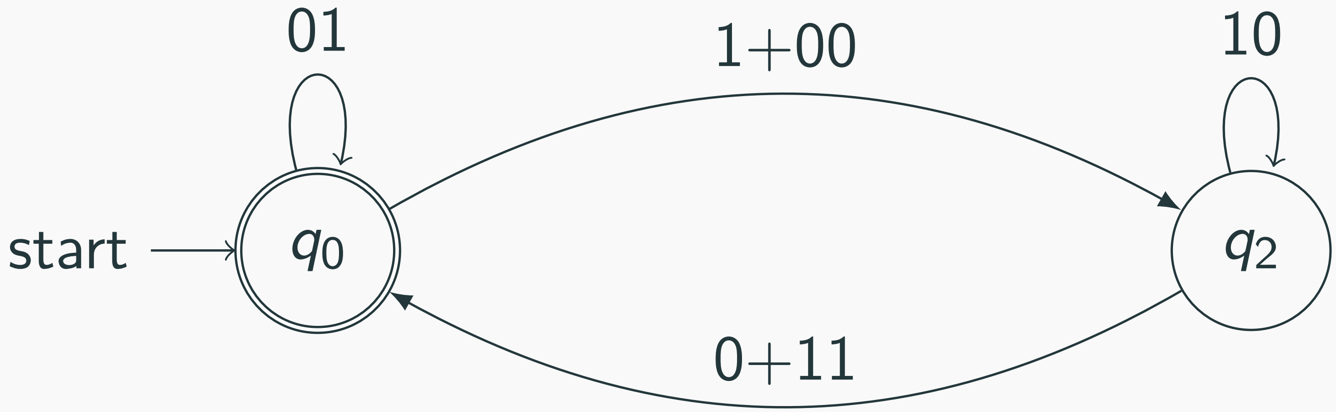
then $q_2 = \delta(q_1, y) = \delta(\delta(q_0, x), y) = \delta(q_0, xy)$

State Removal method - Example

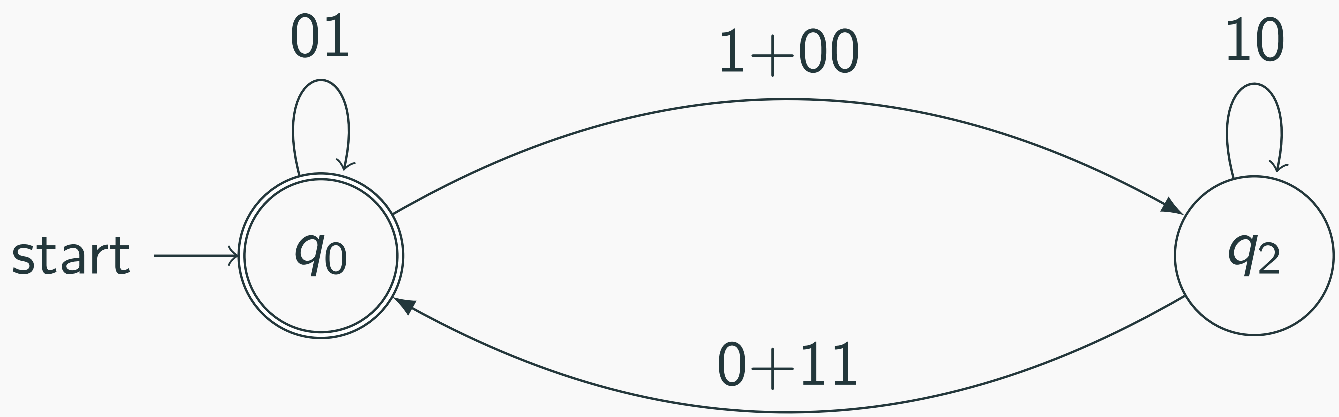


State Removal method - Example

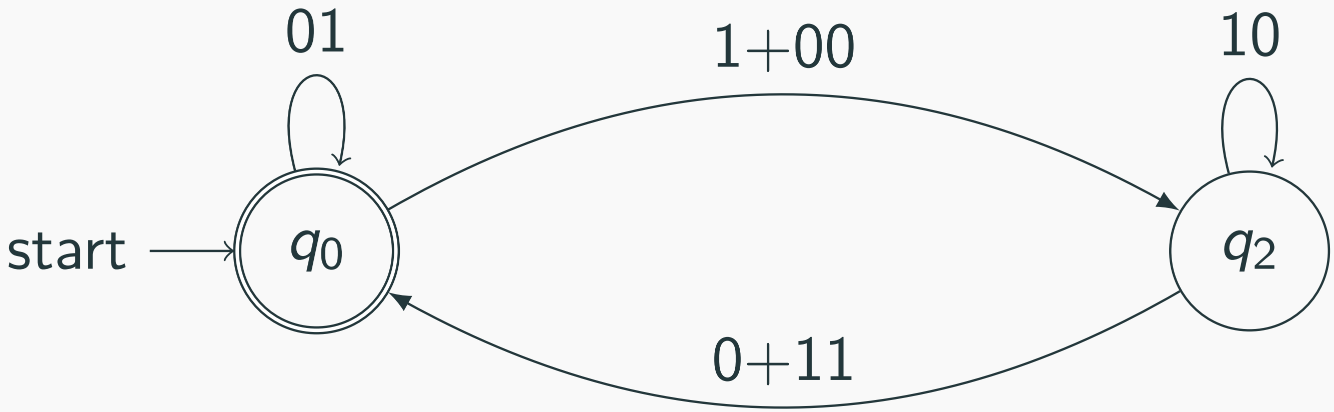




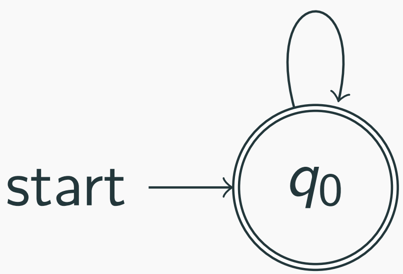
State Removal method - Example



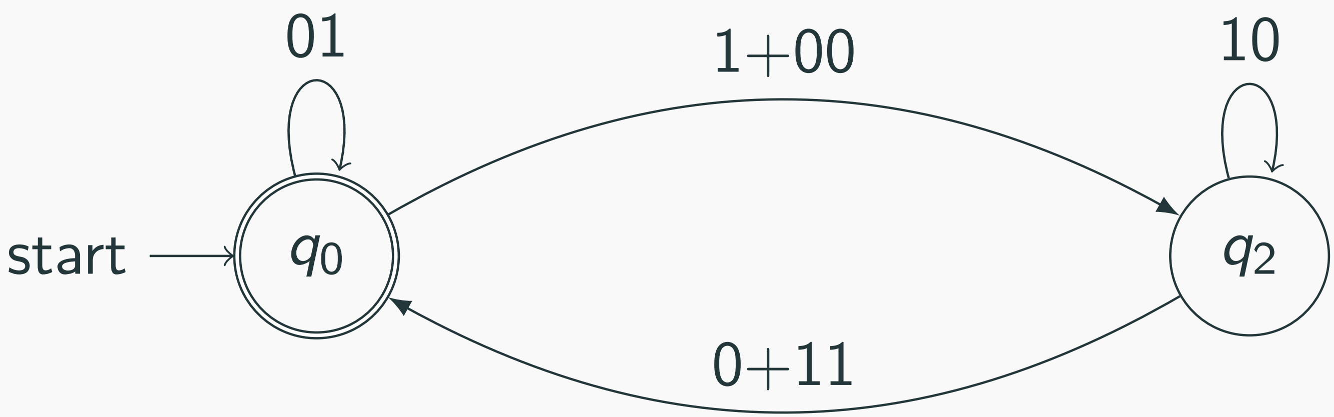
State Removal method - Example



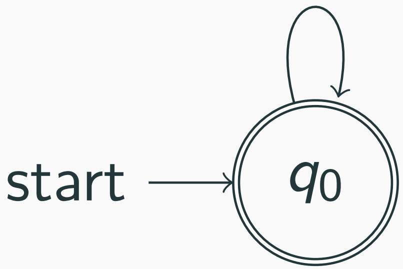
$$01 + (1 + 00)(10)^*(0 + 11)$$



State Removal method - Example



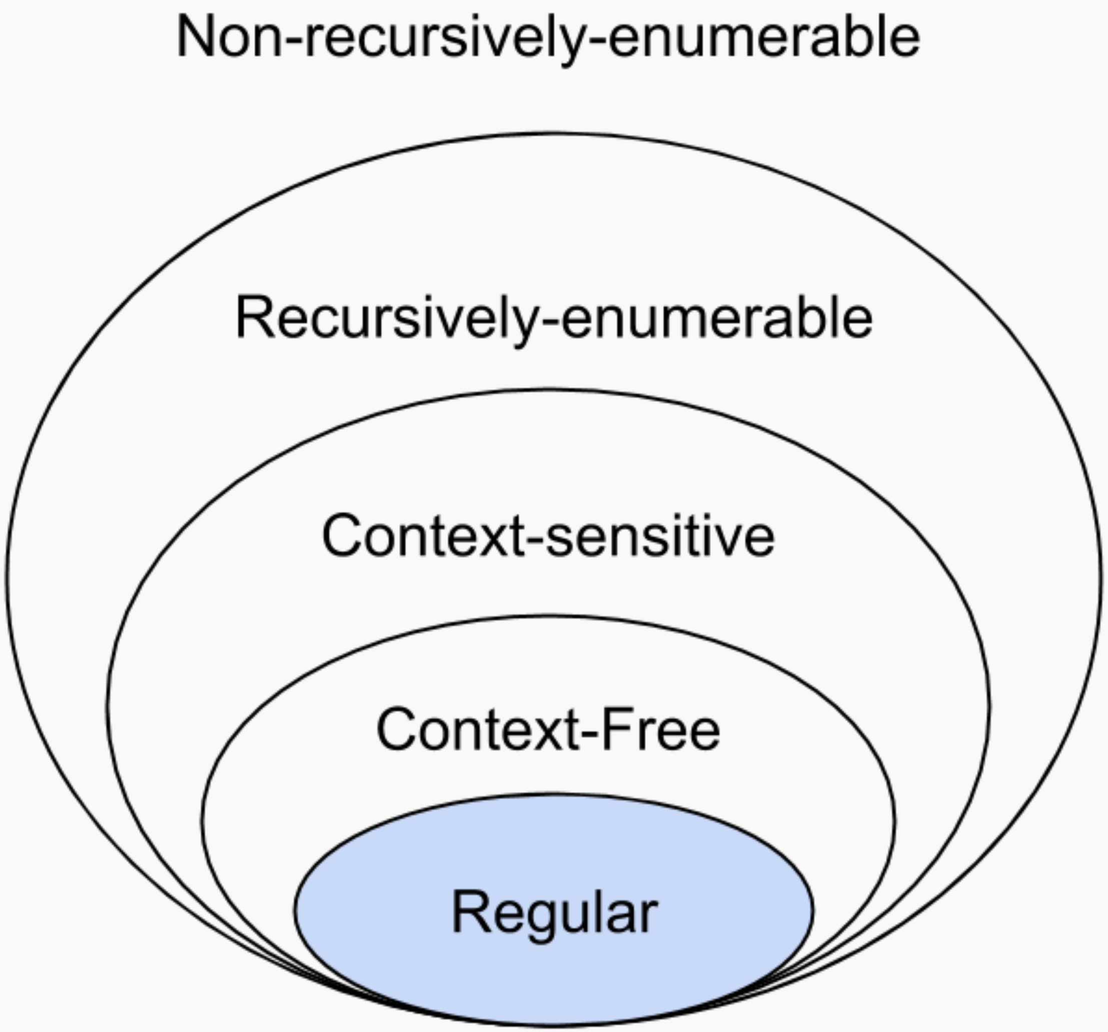
$01 + (1 + 00)(10)^*(0 + 11)$



$(01 + (1 + 00)(10)^*(0 + 11))^*$

DFAs and regular expressions

The thing to know right now is that DFAs and regular expressions represent the same set of languages!



The End - Reminders

- HW 1 has been assigned. Will be due next week.
- Lab tomorrow will go over DFAs

Extra Slides

Algebraic method

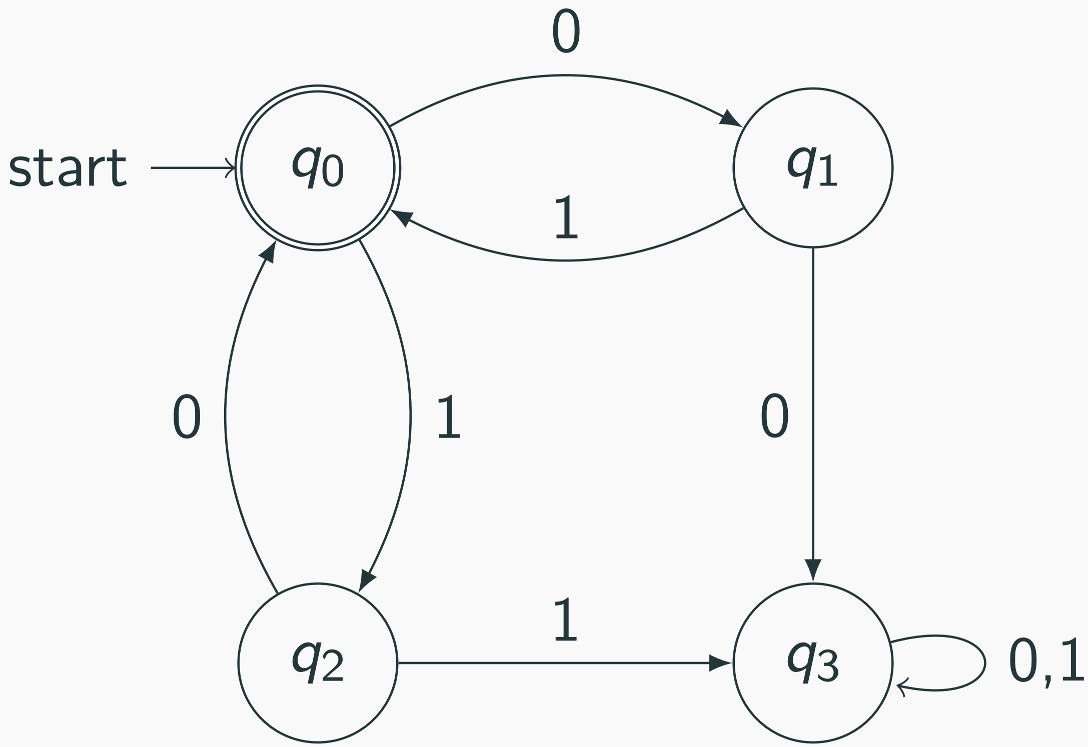
Transition functions are themselves algebraic expressions!

Demarcate states as variables.

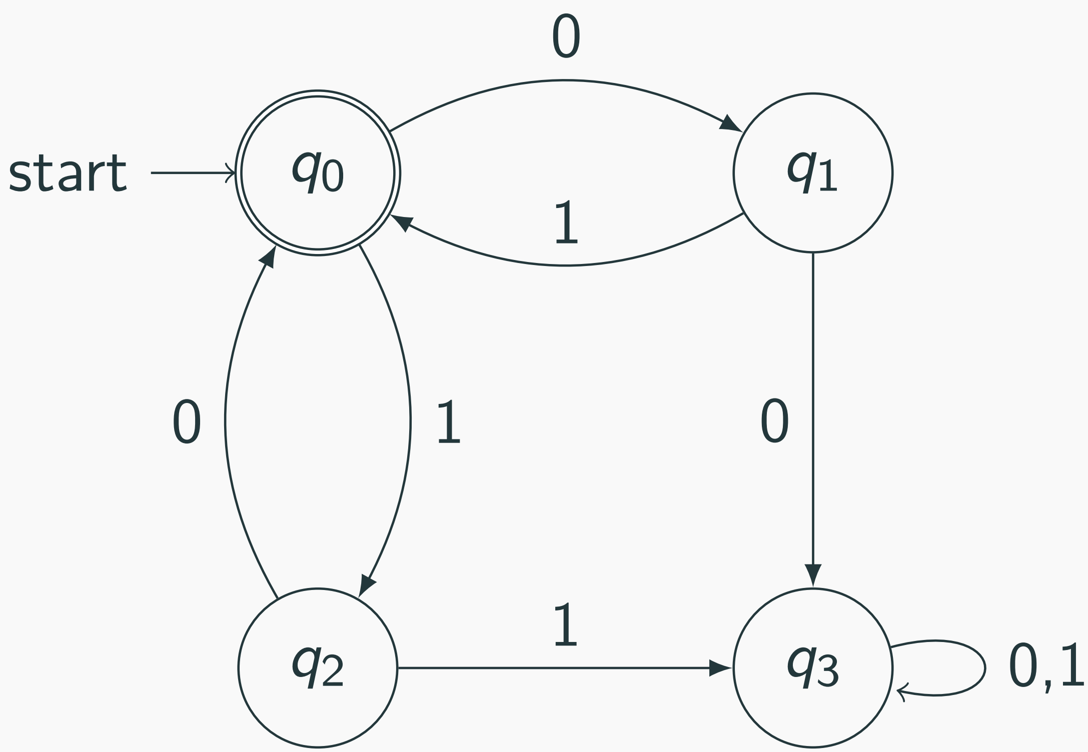
Can rewrite $q_1 = \delta(q_0, x)$ as $q_1 = q_0\mathsf{x}$

Solve for accepting state.

Algebraic method - Example



Algebraic method - Example



- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_1 = q_0 0$
- $q_2 = q_0 1$
- $q_3 = q_1 0 + q_2 1 + q_3 (0 + 1)$

Algebraic method - Example

- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_1 = q_0 0$
- $q_2 = q_0 1$
- $q_3 = q_1 0 + q_2 1 + q_3 (0 + 1)$

Now we simple solve the system of equations for q_0:

- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_0 = \epsilon + q_0 01 + q_0 10$
- $q_0 = \epsilon + q_0 (01 + 10)$

Theorem (Arden's Theorem)
$R = Q + RP = QP^*$

Algebraic method - Example

- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_1 = q_0 0$
- $q_2 = q_0 1$
- $q_3 = q_1 0 + q_2 1 + q_3 (0 + 1)$

Now we simple solve the system of equations for q_0:

- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_0 = \epsilon + q_0 01 + q_0 10$
- $q_0 = \epsilon + q_0 (01 + 10)$
- $q_0 = \epsilon (01 + 10)^* = (01 + 10)^*$