STALGCM NOTES FOR LE1 

By: Roemer Caliboso

Special Thanks to Anthony Baybayon and Kenneth Go

IMPORTANT REMINDERS:

Reminders for STALGCM Exams:

- TLDR: FOLLOW INSTRUCTIONS
- **DONT** FORGET TO PUT START STATE
- **DONT** EXCEED STATE LIMIT
- ONLY **ONE VALID ANSWER** FOR MACHINES
- PUT **COMPLETE SOLUTION** FOR PARTITIONING
- **DONT** FORGET TO PUT FINAL ANSWER FOR EQUIVALENCE (A $\sim$ J $\therefore$ M1 $\sim$ M2)
- **DONT** FORGET TO PUT L(M) AS YOUR FINAL ANSWER FOR NFA TO REGEX
- NO NEED TO LABEL STATES FOR REGEX TO ϵ / λ - NFA
- **DONT** FORGET TO PUT THE STATE ITSELF IN λ CLOSURE
- FEBRUARY 12, 2025 | 9:00AM - 11:00AM
- BRING LARGE TEST BOOKLET
- 10 ITEMS, 10 POINTS PER ITEM

NOTE THAT SOME OF THESE REMINDERS ARE SPECIFIC TO SR AUSTIN :)

if were wrong,,,,,, sorry

"Turn something you don't know into something that you do know."
- Dr. Jose Tristan Reyes

[1] Mealy and Moore Machines

Mealy Machines (Transition-Assigned Machines)

$M = (Q, S, R, f, g, q_I)$

Q — finite set of states

S — finite set of stimulus symbols

R — finite set of response symbols

$f : Q \times S \mapsto Q$ — transition function

(A state given an input outputs another state)

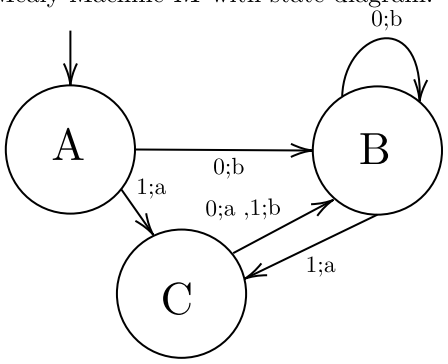
$g : Q \times S \mapsto R$ — output function

(A state given an input outputs a response symbol)

q_I : inital state, $q_I \in Q$

Ex.

Given the Mealy Machine M with state diagram:



$M = (Q, S, R, f, g, q_I)$

$Q = \{A, B, C\}$

$S = \{0, 1\}$

$R = \{a, b\}$

$f(A, 0) = B$	$g(A, 0) = b$
$f(A, 1) = C$	$g(A, 1) = a$
$f(B, 0) = B$	$g(B, 0) = b$
$f(B, 1) = C$	$g(B, 1) = a$
$f(C, 0) = B$	$g(C, 0) = a$
$f(C, 1) = B$	$g(C, 1) = b$

$q_I = A$

Moore Machines (State-Assigned Machines)

$M = (Q, S, R, f, h, q_I)$

Q — finite set of states

S — finite set of stimulus symbols

R — finite set of response symbols

$f : Q \times S \mapsto Q$ — transition function

(A state given an input outputs another state)

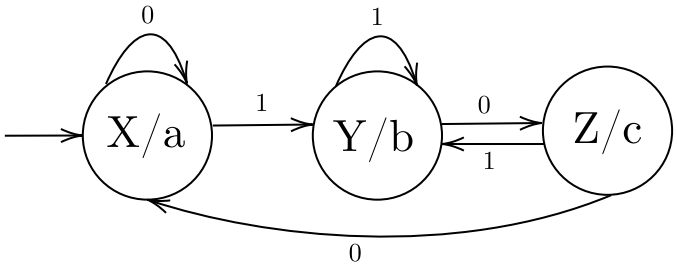
$h : Q \mapsto R$ — output function

(A state given an input outputs a response symbol)

q_I : inital state, $q_I \in Q$

Ex.

Given the Moore Machine M with state diagram:



$M = (Q, S, R, f, g, q_I)$

$Q = \{X, Y, Z\}$

$S = \{0, 1\}$

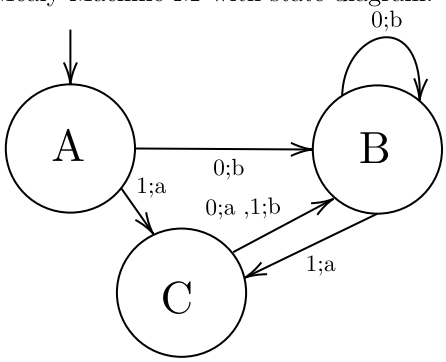
$R = \{a, b, c\}$

$f(X, 0) = X$	$h(X) = a$
$f(X, 1) = Y$	$h(Y) = b$
	$h(Z) = c$
$f(Y, 0) = Z$	
$f(Y, 1) = Y$	
$f(Z, 0) = X$	
$f(Z, 1) = Y$	

$q_I = X$

Mealy and Moore Machines may also be represented by a transition table:

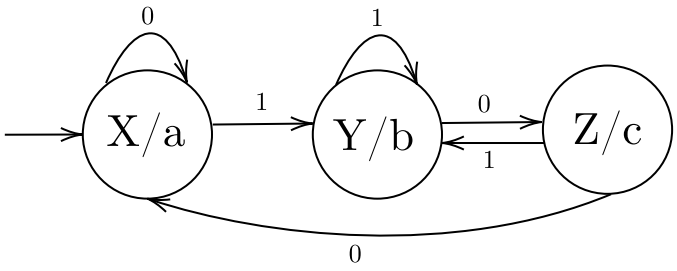
Given the Mealy Machine M with state diagram:



This may also be represented with the transition table:

	0	1
$\rightarrow A$	$B:b$	$C:a$
B	$B:b$	$C:a$
C	$B:a$	$B:b$

Given the Moore Machine M with state diagram:



This may also be represented with the transition table:

	0	1
$\rightarrow X:a$	X	Y
$Y:b$	Z	Y
$Z:c$	X	Y

[2] Conversions Between Mealy and Moore

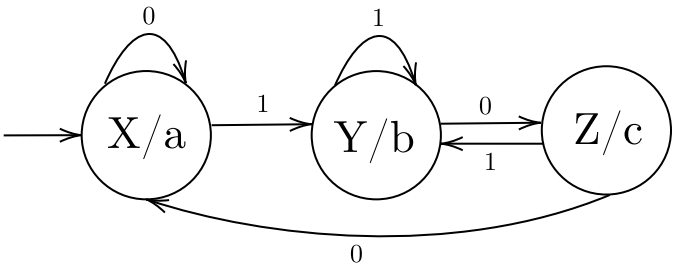
Mealy and Moore machines are **equivalent** because we can convert between them.

Conversion - Moore to Mealy

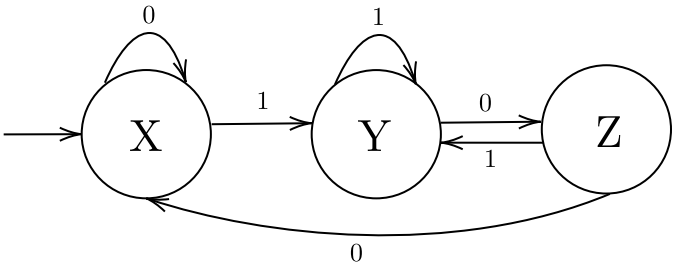
1. Copy all the states (without responses) and all the transitions.
2. For each transition, the response will be the state response.

Ex.

Convert the following Moore Machine into a Mealy Machine.

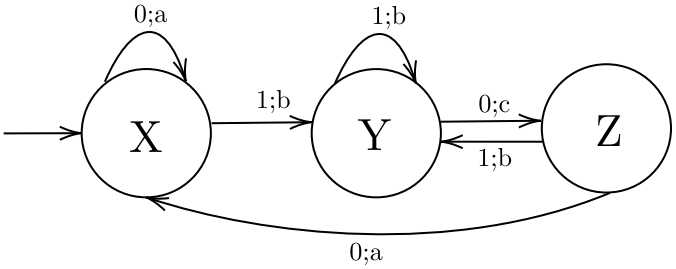


Step 1:



X/a
Y/b
Z/c

Step 2:



X/a
Y/b
Z/c

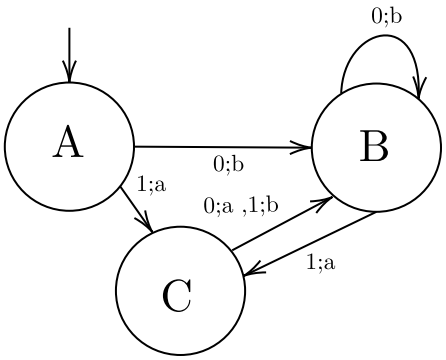
Done!

Conversion - Mealy to Moore

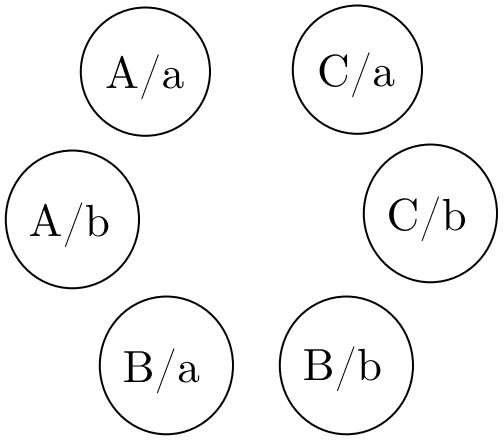
1. For each state duplicate it based on the number of responses in R. The states should have duplicate version for each response.
   (For example, if a machine M has 3 responses, state A will have 3 duplicates based on the responses.)
2. For each transition function of a state I to a state J with output x, each duplicated state I will map to the state J corresponding to output x.
3. To assign the initial state, ANY of the duplicated initial states in the original machine may be assigned the initial state in the converted machine. (but only one initial state should be assigned)

Ex.

Convert the following Mealy Machine into a Moore Machine.

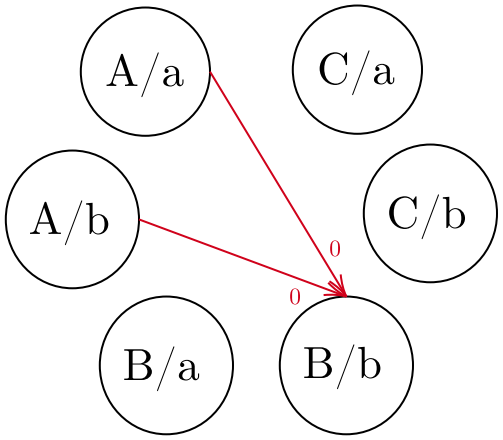


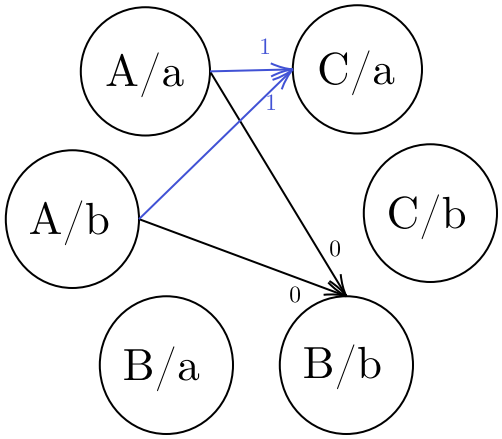
Step 1:

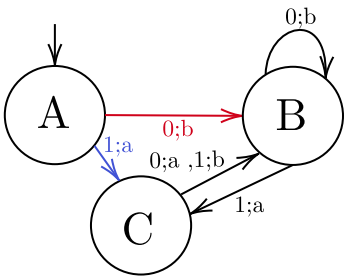


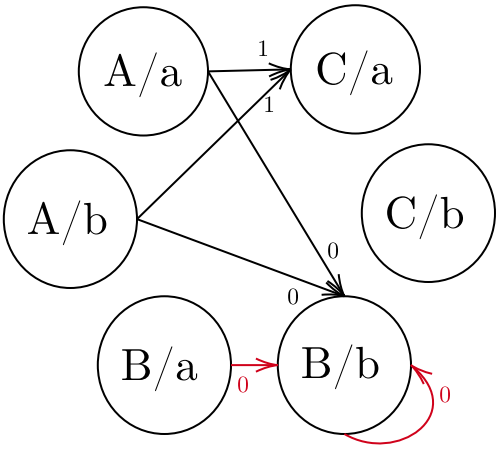
$R = \{a, b\}$

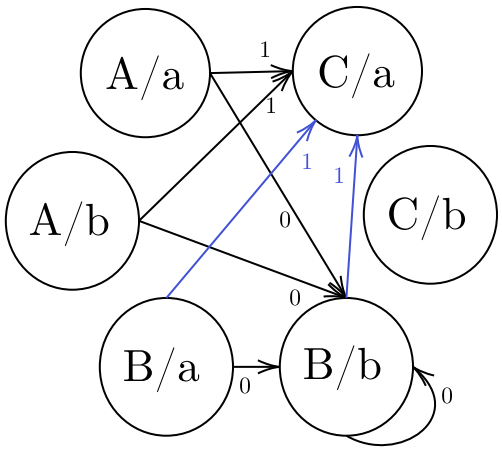
Step 2:

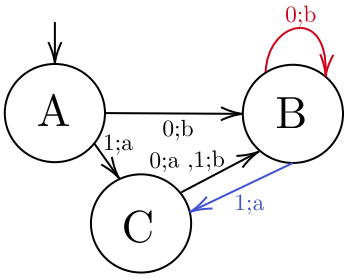


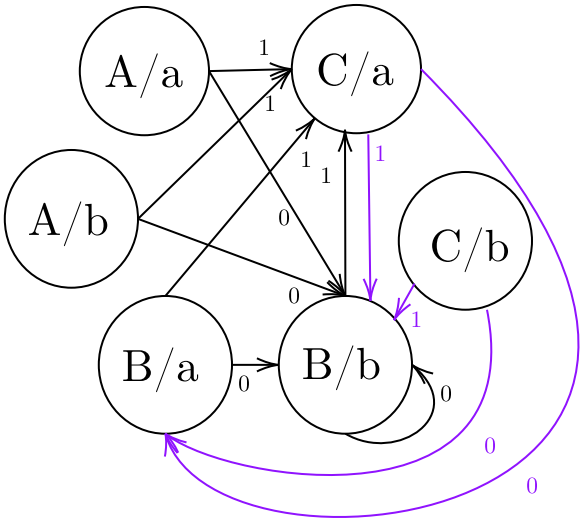


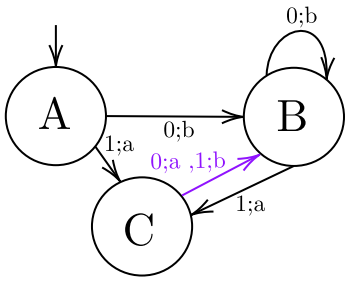




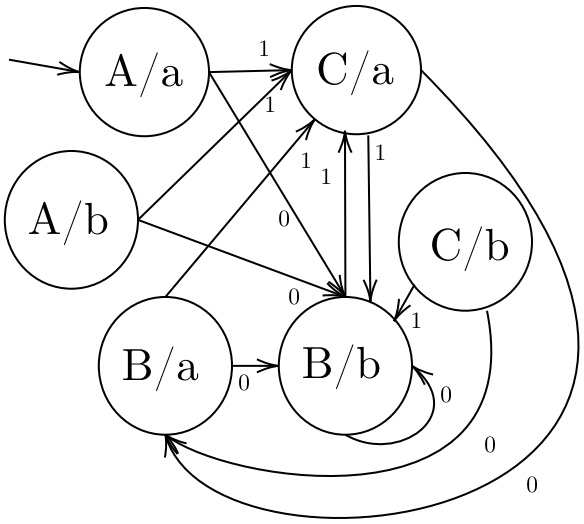








Step 3:



$q_I = A$

Choose A/a or A/b as start state

Done!

Formal Conversion between Mealy and Moore:

Moore → Mealy	**Mealy → Moore**
Given a Moore Machine M_a we convert to a Mealy Machine M_b. $M_a = (Q_a, S_a, R_a, f_a, h, q_{Ia})$ $M_b = (Q_b, S_b, R_b, f_b, g, q_{Ib})$	Given a Mealy Machine M_a we convert to a Moore Machine M_b. $M_a = (Q_a, S_a, R_a, f_a, g, q_{Ia})$ $M_b = (Q_b, S_b, R_b, f_b, h, q_{Ib})$
$Q_b = Q_a$; $S_b = S_a$; $R_b = R_a$; $f_b = f_a$; $g : Q_b \times S_b \mapsto Q_b$ $g(q, s) = h(f_a(q, s))$; $q_{Ib} = q_{Ia}$	$Q_b = Q_a \times R_a$; $S_b = S_a$; $R_b = R_a$; $f_b : Q_b \times S_b \mapsto Q_b$ $f_b((q, r), s) = (f_b(q, s), g(q, s)), q \in Q_b, r \in R_b, s \in R_b$ $h : Q_b \mapsto R_b$ $h((q, r) = r, q \in Q_a, r \in R_b$; and $q_{Ib} = (q_{Ia}, r)$, where r is any $r \in R_b$

[3] Reduction and Equivalence

Machine Equivalence

Two machines are equivalent if they exhibit the same behavior.

- Machines must have the same input and output alphabet.
- Two accepters are equivalent if they accept and reject the same set of strings.
- Two generators are equivalent if they generate the same set of strings.
- Two transducers are equivalent if give the stimulus alphabet S where $\forall \omega \in S^*$, the response of the two machines are the same for any given ω.

Note that machine equivalence is an equivalence relation:

Machines are reflexive: $(M \sim M)$

All machines are equivalent to themselves.

Machines are symmetric: $(M_1 \sim M_2 \ \leftrightarrow M_2 \sim M_1)$

If a machine M_1 is equivalent to a machine M_2, then they are equivalent to each other.

Machines are transitive: $\big(M_1 \sim M_2 \wedge M_2 \sim M_3 \rightarrow M_1 \sim M_3\big)$

If a machine M_1 is equivalent to a machine M_2 and M_2 is equivalent to a machine M_3 then M_1 is equivalent to M_3

State Equivalence

Two states q_a and q_b in a finite-state machine are equivalent if and only if two machines M_a and M_b created with q_a and q_b as respective initial states are equivalent. If q_a and q_b are equivalent, we denotes this as $q_a \sim q_b$.

Two states q_a and q_b are equivalent when:

for Mealy machines:

$\forall s \in S \ (g(q_a, s) = g(q_b, s))$ (All responses are equal).

for Moore machines:

$h(q_a) = h(q_b)$ (All responses are equal).

Conditions for Machine Reduction and Connection

A finite-state machine is reduced if no two distinct states in the machine are equivalent.

A state q in a finite-state machine is accessible if it is the ω successor of the initial state for some $\omega \in S^*$.

($\exists \omega \in S^*$ where $q_I \xrightarrow{\omega} q$ (an input string can successfully reach the state))

A finite machine is connected if all states $q \in Q$ are accessible

K-Distinguishable Machines

Two states q_a and q_b of a Mealy machine are k - distinguishable if there exists at least one $\omega \in S^*$ with $|\omega| \leq k$ where the responses of two machines M_a and M_b differ by at least one symbol.

If q_a and q_b are not k - distinguishable, then they are k - equivalent.

Two states q_a and q_b of a Mealy machine are k - equivalent if and only if they are:

- 1 - equivalent
- For all $s \in S$, their s - successors are $(k-1)$ - equivalent.

Partition of a Set

A partition $P = \{B_1, B_2, ..., B_n\}$ of a set A has the following properties:

$\left(\bigcup_{i=1}^{n} B_i\right) = A$ and for all i (The union of all the sets is the original set); and

$\forall i \forall j \ B_i \cap B_j = \emptyset, 1 \leq i \leq j \leq n$ (All sets are pairwise disjoint) (No two sets contain the same element)

Refinement

Given two partitions P_1 and P_2 if all sets in P_2 are the subset of exactly one set in P_1, P_2 is a refinement of P_1.

This means no trading!

$A = \{\{1,2,3\},\{4,5,6\}\}$
$B = \{\{1,2\},\{3\},\{4,5,6\}\}$ Valid refinement of A

$A = \{\{1,2,3\},\{4,5,6\}\}$ $\times$
$B = \{\{1,2\},\{3,4,5,6\}\}$ Not a refinement of A (3 was given to a diff set)

$A = \{\{1,2,3\},\{4,5,6\}\}$
$B = \{\{1,2,3\},\{4,5,6\}\}$ Valid refinement of A

Reducing a Machine:

1. Form an initial partition P_1 by grouping together states that are 1 - equivalent. (Same outputs for each input symbol).
2. Refine the previous partition given that state q_a and q_b are in the same block of P_{k+1} if and only if:
   a. They are in the same block of P_k (no trading!)
   b. All s - successors are in the same block of P_k.
3. Repeat step 2, until the refinement is the same ($P_{m+1} = P_m$ for some m)

Ex.

Reduce the following Mealy Machine:

	0	1
$\to A$	$B:1$	$D:0$
B	$F:0$	$H:0$
C	$D:1$	$A:0$
D	$C:1$	$A:0$
E	$H:1$	$C:0$
F	$B:1$	$C:0$
G	$A:1$	$G:0$
H	$E:0$	$B:0$

$P1$

0	A	10
	C	00
	D	00
	E	10
	F	10
	G	00
1	B	01
	H	01

$P2$

0	A	21
	E	21
	F	21
1	C	10
	D	10
	G	01
2	B	02
	H	02

$P3$

0	A	31
	E	31
	F	31
1	C	10
	D	10
2	G	02
3	B	03
	H	03

	0	1
$\to S0$	$S3:1$	$S1:0$
$S1$	$S1:1$	$S0:0$
$S2$	$S0:1$	$S2:0$
$S3$	$S0:0$	$S3:0$

note that S0 contains the initial state
$S2$ is inaccessible.

	0	1
$\to S0$	$S3:1$	$S1:0$
$S1$	$S1:1$	$S0:0$
$S3$	$S0:0$	$S3:0$

done!

Reduce the following Moore Machine:

	0	1	2	3
$\to A:0$	D	B	B	D
$B:1$	F	E	C	D
$C:1$	C	D	E	A
$D:0$	A	F	F	E
$E:0$	E	B	F	A
$F:1$	B	A	C	E

$P1$

0	A	0110
	D	0110
	E	0110
1	B	1010
	C	1000
	F	1010

$P2$

0	A	0110
	D	0110
	E	0110
1	B	1020
	F	1020
2	C	2000

	0	1	2	3
$\rightarrow S0:0$	$S0$	$S1$	$S1$	$S0$
$S1:1$	$S1$	$S0$	$S2$	$S0$
$S2:1$	$S2$	$S0$	$S0$	$S0$

Showing Machine Equivalence

To show machine equivalence, we must perform the partitioning algorithm and keep in mind the initial states of each machine. The transition tables for each machine should be appended to each other (see example).

If at any point during the partitioning where the initial states of each machine get split in a refinement, then the two machines are not equivalent. Otherwise, the two machines are equivalent.

Ex.
Show that the following two Moore machines are equivalent:

M_1

	0	1
$\rightarrow A:1$	B	C
$B:1$	D	F
$C:0$	F	E
$D:1$	B	G
$E:0$	F	C
$F:0$	E	D
$G:0$	F	G

M_2

	0	1
$\rightarrow H:1$	I	J
$I:1$	H	K
$J:0$	K	J
$K:0$	J	H

	0	1
$A:1$	B	C
$B:1$	D	F
$C:0$	F	E
$D:1$	B	G
$E:0$	F	C
$F:0$	E	D
$G:0$	F	G
$H:1$	I	J
$I:1$	H	K
$J:0$	K	J
$K:0$	J	H

$P1$

0	C	00
	E	00
	F	01
	G	00
	J	00
	K	01
1	A	10
	B	10
	D	10
	H	10
	I	10

$P2$

0	C	10
	E	10
	G	10
	J	10
1	F	02
	K	02
2	A	20
	B	21
	D	20
	H	20
	I	21

$P3$

0	C	10
	E	10
	G	10
	J	10
1	F	02
	K	02
2	A	30
	D	30
	H	30
3	B	21
	I	21

Note that the initial states are not in separate sets, so initial states are equivalent.

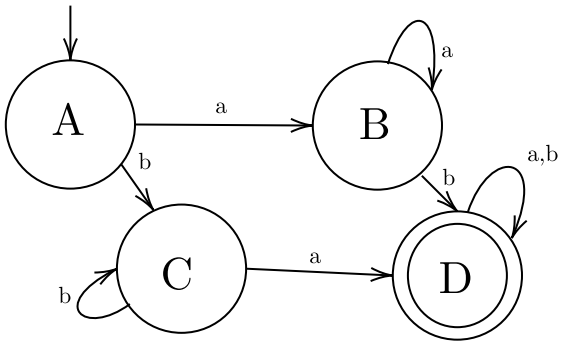
Since $A \sim H$, therefore $M_1 \sim M_2$

[4] DFAs and NFAs

Deterministic Finite Accepters (DFA)
$M = (Q, S, \delta, I, F)$
Q — finite set of states
S — finite set of stimulus symbols
$\delta : Q \times S \mapsto Q$ — transition function
F — finite set of accepting/final states
I — $\{q_I\}$ set of initial states
$F \subseteq Q$

A deterministic finite state accepter is said to accept a string ω if there exists an accepting state $q \in F$, such that q is the ω-successor of the initial state q_I.

Ex.
Design a DFA for all strings $\omega \in \{a, b\}*$ that contain at least one instance of an a followed by a b or a b followed by an a.



$M = (Q, S, \delta, I, F)$
$Q = \{A, B, C, D\}$
$S = \{a, b\}$

$\delta(A, a) = B$
$\delta(A, b) = C$
$\delta(B, a) = B$
$\delta(B, b) = D$
$\delta(C, a) = C$
$\delta(C, b) = D$
$\delta(D, a) = D$
$\delta(D, b) = D$

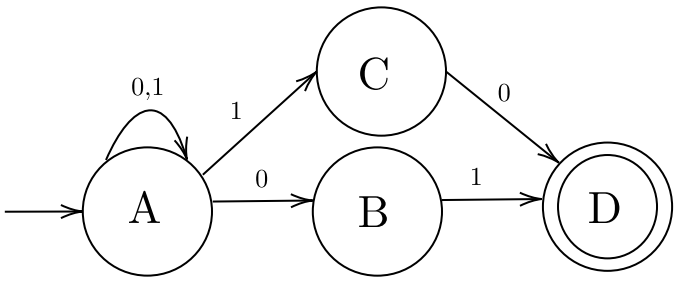
$F = \{D\}$
$I = \{A\}$

This may also be represented with the transition table:

	a	b
$\rightarrow A$	B	C
B	B	D
C	D	C
D^*	D	D

Non-Deterministic Finite Accepters (NFA)
$M = (Q, S, \delta, I, F)$
Q — finite set of states
S — finite set of stimulus symbols
$\delta : Q \times S \mapsto \mathcal{P}(Q)$ — transition function
Each state-symbol pair maps to a subset of Q
F — finite set of accepting/final states
I — $\{q_I\}$ set of initial states
$F \subseteq Q$
A deterministic finite state accepter is said to accept a string ω if there exists an initial state $q \in I$ and an accepting state $q' \in F$ such that q' is a ω - successor of q.

Ex.
Design an NFA that accepts all strings $\omega \in \{0, 1\}$ ending with 01 or 10.



$M = (Q, S, \delta, I, F)$
$Q = \{A, B, C, D\}$
$S = \{a, b\}$

$\delta(A, 0) = \{A, B\}$
$\delta(A, 1) = \{A, C\}$
$\delta(B, 0) = \{\}$
$\delta(B, 1) = \{D\}$
$\delta(C, 0) = \{D\}$
$\delta(C, 1) = \{\}$
$\delta(D, 0) = \{\}$
$\delta(D, 1) = \{\}$

$F = \{D\}$
$I = \{A\}$

This may also be represented with the transition table:

	0	1
$\rightarrow A$	AB	AC
B		D
C	D	
D^*		

Conversion between DFAs to NFAs is trivial, as all DFAs are NFAs by definition.

Converting NFAs to DFAs

1. Write down the transition table of the NFA.

	0	1
$\to A$		

	0	1
$\to A$	AB	AC
B		D
C	D	
D^*		

2a. Starting with the initial state, for each transition, read each state reached (consider each cell as a set) and add each state that has not been read before on the left hand side.

If there are multiple initial states, take the union of initial states.

	0	1
$\to A$	AB	AC
AB		
AC		

	0	1
$\to A$	AB	AC
B		D
C	D	
D^*		

2b. For states that are the union of multiple states, for each transition, read the union of their outputs and add each state that has not been read before on the left hand side.

	0	1
$\to A$	AB	AC
AB	AB	ACD
AC	ABD	AC
ACD		
ABD		

	0	1
$\to A$	AB	AC
B		D
C	D	
D^*		

3. Repeat steps 2a and 2b until no more new states have been discovered.

After no more new states have been discovered, each state including any of the final states in the original NFA will become final states themselves.

In the resulting DFA, the take each state cell as its own state. Note that the final DFA may have unreachable states. States that are unreachable from the start state may be removed (this is a necessary step in connecting the machine).

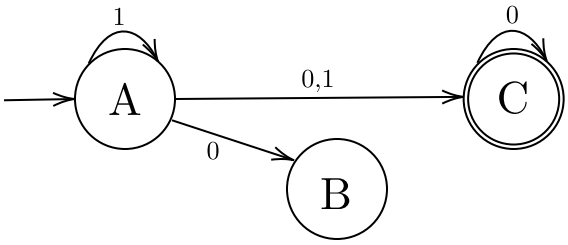
NOTE that if there are any missing transitions in the final DFA, a dead state must be put in its place. This dead state must be added on the left hand side of states. Each transition must lead back to this dead state.

	0	1
$\to A$	AB	AC
AB	AB	ACD
AC	ABD	AC
ACD^*	ABD	AC
ABD^*	AB	ACD

	0	1
$\to A$	AB	AC
B		D
C	D	
D^*		

The partitioning algorithm for DFAs are the same, the initial partition merely depends on the set of final states.

Ex.
Convert the following NFA to a reduced and connected DFA



We first represent the NFA with a transition table:

	0	1
$\to A$	BC	AC
B		
C^*	C	

We then convert it to a DFA:

	0	1
$\to A$	BC	AC

	0	1
$\to A$	BC	AC
BC		
AC		

	0	1
$\to A$	BC	AC
BC	C	
AC	BC	AC
C		

	0	1
$\to A$	BC	AC
BC	C	
AC	BC	AC
C	C	

	0	1
$\to A$	BC	AC
BC^*	C	X
AC^*	BC	AC
C^*	C	X
X	X	X

After getting the equivalent DFA, we partition.

	0	1
$\to A$	BC	AC
BC^*	C	X
AC^*	BC	AC
C^*	C	X
X	X	X

$P1$

0	A	11
	X	00
1	BC	10
	AC	11
	C	10

$P2$

0	A	23
1	X	11
2	BC	21
	C	21
3	AC	23

	0	1
$\to S0$	$S2$	$S3$
$S1$	$S1$	$S1$
$S2^*$	$S2$	$S1$
$S3^*$	$S2$	$S3$

All states are accessible.
NFA conversion and reduction complete.

[5] Regular Expressions

Regular Expressions (Regex)

A sequence of symbols/characters expressing pattern.

Let V be a finite alphabet. A regular expression on V is any finite string of symbols from the set $\{w \mid w \in V\} \cup \Big\{ \cup, {}^*, (,), \lambda, \varnothing \Big\}$ that may be formed according to the following rules:

1. λ is a regex.
2. $\varnothing$ is a regex.
3. $(a \in V) \to a$ is a regex.
4. If α and β are regular expressions, then the following are also regular expressions:
   (a) $\alpha\beta$ (concatenation)
   (b) $\alpha \cup \beta$ (choosing α or β)
   (c) $(\alpha)^*$ (choosing zero to many α)
   (d) (α)

Note that two regexs are equivalent if they describe the same set of strings.

Ex. Provide a regular expression over the alphabet $V = \{0, 1\}$ where the string starts with a 1 and ends with a 1, or the string starts with a 0 and ends with a 0.

$\big(1(0 \cup 1)^*1\big) \cup \big(0(0 \cup 1)^*0\big)$

Properties of Regular Expressions:

$\big(\alpha^*\big)^* = \alpha^*$	$\alpha(\beta \cup \gamma) = \alpha\beta \cup \alpha\gamma$
$\alpha\alpha^* = \alpha^*\alpha$	$\alpha(\beta\alpha)^* = (\alpha\beta)^*\alpha$
$\alpha\alpha^* \cup \ \lambda = \alpha^*$	$\big(\alpha^* \cup \beta^*\big)^* = (\alpha \cup \beta)^* = \big(\alpha^*\beta^*\big)^* = \alpha^*\big(\beta\alpha^*\big)^*$

Ardens Rule:

$A = \alpha A \cup \beta$

$A = \alpha^*\beta$

Converting Finite State Automata to Regex

Given any FSA (DFA or NFA) we can convert it into a regular expression.

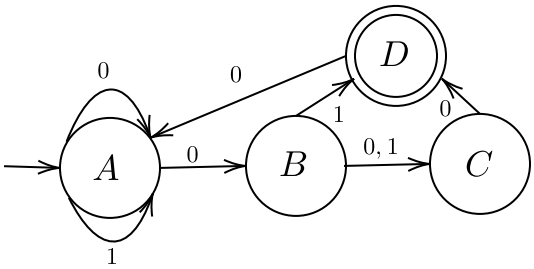
End Set: Set of all strings starting from a state that end on an accepting state.

Steps:

1. Get the transition table of the accepter you want to convert:
2. Write down the language of the machine $L(M)$ and the corresponding states. Write them in end state notation.
   2a. For $L(M)$ list the union of all start states.
   2b. For each of the states, for every transition, list the union of all the items.
   2c. If the state is a final state, add an extra union with λ (empty strings on a final state are accepted).
3. Substitute each of the states in the other states to simplify them and write them in terms of Regex
4. Apply Arden's Rule to recursive equations.
5. Substitute into $L(M)$.

Ex.

Convert the following NFA to a regular expression.



	0	1
$\to A$	AB	A
B	C	CD
C	D	
D^*	A	

Write the End Set Notation:

$L(M) = A$
$A = 0A \cup 0B \cup 1A$
$B = 0C \cup 1C \cup 1D$
$C = 0D$
$D = 0A \cup \lambda$

$L(M) = A$
$A = 0A \cup 0B \cup 1A$
$B = 00D \cup 10D \cup 1D$
$C = 0D$
$D = 0A \cup \lambda$

$L(M) = A$
$A = 0A \cup 0(00D \cup 10D \cup 1D) \cup 1A$
$B = 00D \cup 10D \cup 1D$
$D = 0A \cup \lambda$

$L(M) = A$
$A = 0A \cup (000 \cup 010 \cup 01)D \cup 1A$
$D = 0A \cup \lambda$

$L(M) = A$
$A = 0A \cup (000 \cup 010 \cup 01)(0A \cup \lambda) \cup 1A$
$D = 0A \cup \lambda$

$L(M) = (0 \cup 1)^*(000 \cup 010 \cup 01)$

$L(M) = A$
$A = 0A \cup (000 \cup 010 \cup 01)(0A \cup \lambda) \cup 1A$
$A = 0A \cup 0000A \cup 0100A \cup 010A \cup 000 \cup 010 \cup 01 \cup 1A$
$A = 0A \cup 0000A \cup 0100A \cup 010A \cup 1A \cup 000 \cup 010 \cup 01$
$A = (0 \cup 0000 \cup 0100 \cup 010 \cup 1)A \cup 000 \cup 010 \cup 01$
$A = (0 \cup 0000 \cup 0100 \cup 010 \cup 1)^*(000 \cup 010 \cup 01)$
$A = (0 \cup 1)^*(000 \cup 010 \cup 01)$

[6] λ-NFAs and Conversions

A λ - NFA is an NFA that defines the transition λ, meaning the machine can change state without reading any input provided that transition exists that leads to the other state.

λ - NFA

$M = (Q, \Sigma, \delta, q_I, q_F)$

Q — finite set of states

Σ — finite input alphabet

$\delta : Q \times (\Sigma \cup \{\lambda\}) \mapsto \mathcal{P}(Q)$ (transition function)

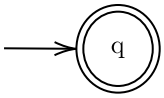
q_I — initial state

q_F — final state

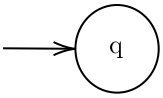
NOTE THAT THIS MAY BE CALLED
ϵ — NFAs in other sections

Building Blocks of Regular Expressions:

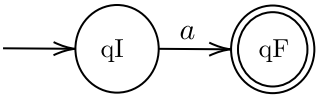
Base Case: λ



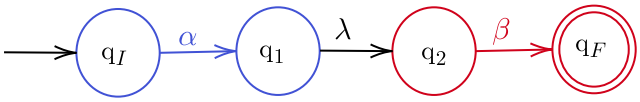
Base Case: $\varnothing$



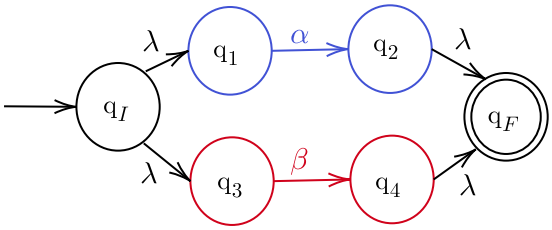
Base Case: a where $a \in V$



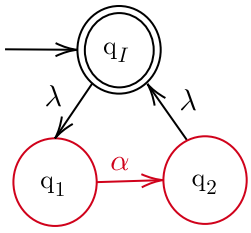
$\alpha\beta$



$\alpha \cup \beta$

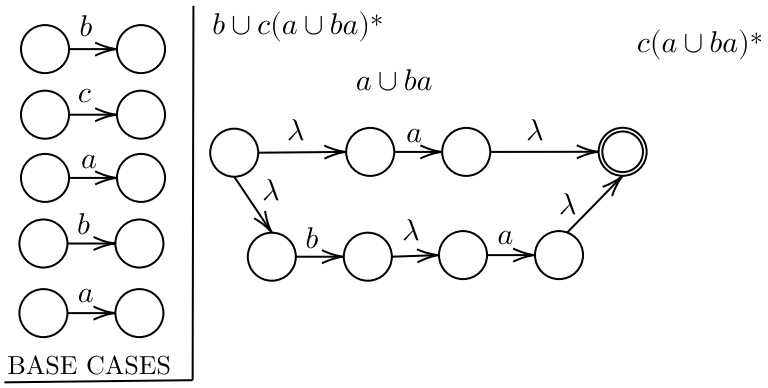


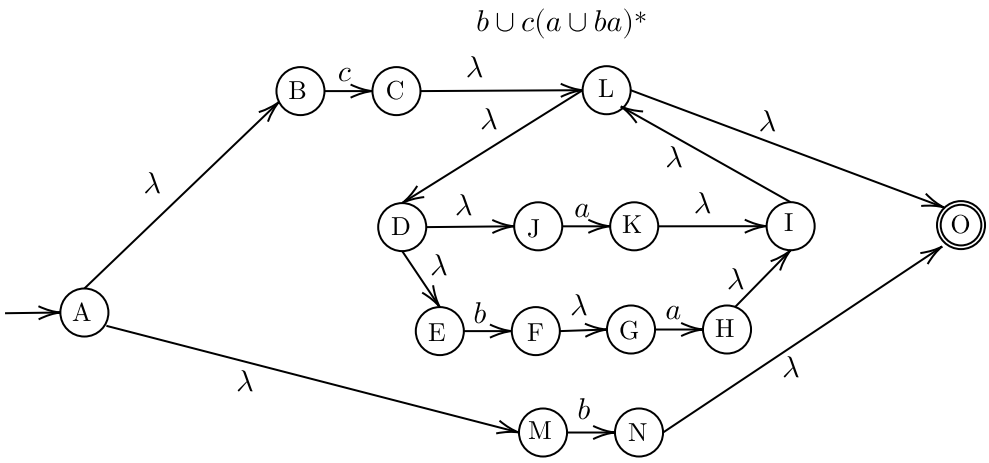
$(\alpha)^*$



Ex.

Given the regular expression $b \cup c(a \cup ba)^*$, construct the λ — NFA





Converting λ - NFA to DFA

To convert a λ - NFA to DFA, we must first define the λ - Closure.

λ - Closure of a state $q \in Q$ is the set of all states that are reachable from q using only λ - transitions.

λ - closure$(q) = \left\{ q' \mid q \xrightarrow{\lambda} q' \right\}$

The λ - closure of a SET of states Q is the set of ALL states reachable from any state in Q using only λ - transitions.

λ - closure$(Q) = \left\{ q' \mid \exists q \left(q \xrightarrow{\lambda} q \right), q \in Q \right\}$

Given a λ - NFA to an equivalent DFA

$M = (Q, \Sigma, \delta, q_I, q_F)$ convert to $M' = (Q', \Sigma, \delta', q_I', F)$

$Q' = \mathcal{P}(Q)$

$\delta'(q, s) = \bigcup_{q' \in q} \lambda$ - Closure$(\delta(q', s))$

$q_I' = \lambda$ - Closure(q_I)

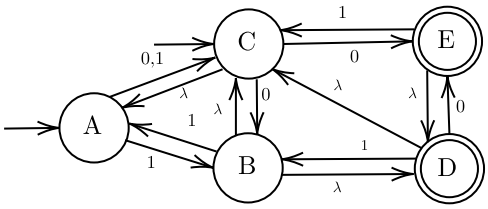
$F = \{q \in Q' \mid q_F \in q\}$

Steps for converting λ - NFA to DFA:

1. Write down the λ - Closure column and for each state, follow the steps.
    1a. Write down the initial state in the cell. Union the states in the λ column.
    1b. For each state in the λ column added to the cell, union to the original cell.
2. In the transition table for the DFA, the start state will be the union of the λ closures of all initial states.
3. For each state, and for each transition, take the union of the states accessed by those states from the λ - NFA
    3a. After you take the union, take the union of the λ closures of those states. This is what you will write the in cell in the DFA.
    3b. If the written down state has not been seen before, introduce it as a new state and write it on the left hand side.
4. Repeat Step 3 until no more new states have been introduced. Final states will be states containing final states in the λ - NFA.

Ex.

Convert the following λ - NFA to a DFA.



	0	1	λ
$\to A$	C	BC	
B		A	CD
$\to C$	BE		A
D^*	E	B	C
E^*		C	D

	0	1	λ	λ - Closure
$\to A$	C	BC		A
B		A	CD	
$\to C$	BE		A	
D^*	E	B	C	
E^*		C	D	

	0	1	λ	λ - Closure
$\to A$	C	BC		A
B		A	CD	$ABCD$
$\to C$	BE		A	
D^*	E	B	C	
E^*		C	D	

	0	1	λ	λ - Closure
$\to A$	C	BC		A
B		A	CD	$ABCD$
$\to C$	BE		A	AC
D^*	E	B	C	ACD
E^*		C	D	$ACDE$

	0	1
$\to AC$	$(CBE)ABCDE$	

	0	1	λ	λ - Closure
$\to A$	C	BC		A
B		A	CD	$\boxed{ABCD}$
$\to C$	BE		A	$\boxed{AC}$
D^*	E	B	C	ACD
E^*		C	D	$\boxed{ACDE}$

	0	1
$\to AC$	$(CBE)ABCDE$	$(BC)ABCD$
$ABCDE$	$(CBE)ABCDE$	$(ABC)ABCD$
$ABCD$	$(CBE)ABCDE$	$(ABC)ABCD$

	0	1
$\to AC$	$ABCDE$	$ABCD$
$ABCDE^*$	$ABCDE$	$ABCD$
$ABCD^*$	$ABCDE$	$ABCD$

D and E are final states.

Done!