NON-DETERMINISTIC FSA

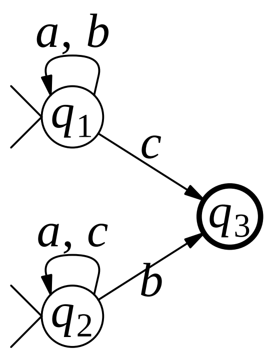
Two types of non-determinism:

(1) *Multiple start-states*; start-states $S \subseteq Q$.

- The language $L(M) = \{x$: x takes M from some start-state to some final-state and all of x is processed$\}$.

The string $x = aac$ is accepted only by starting at state q_1, and $x = aab$ is accepted only by starting state q_2.

Two start-states q_1 and q_2:

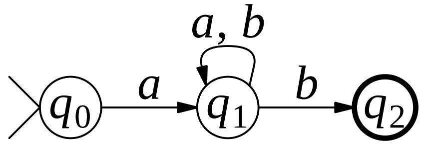


$L(M) = \{\ b, c,$
$ab, ac, bc, cb,$
$\cdots\}$

(2) *Non-unique transitions*; $\delta(q_i, a_j)$ is a set of states $\subseteq Q$.

- The language $L(M) = \{x$: x takes M for some choice of successive transitions from the start-state to some final-state and all of x is processed$\}$.

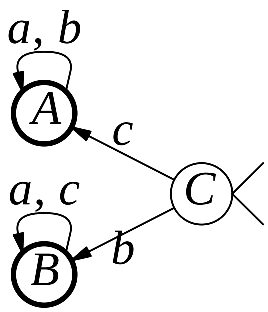
$\delta(q_1, b)$
$= \{q_1, q_2\}$



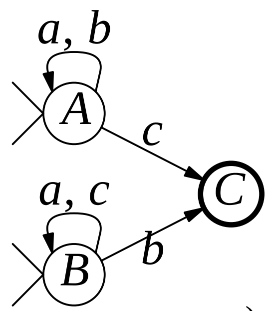
$x = abb$ can be fully processed in only 2 ways, and one of them accepts x.

For each NFSA M, there is an *equivalent* deterministic FSA M' such that $L(M) = L(M')$.

REVERSING AN FSA MAY CREATE A NFSA



M; $L(M)$ =
$b(a+c)^* + c(a+b)^*$



M^r; $L(M^{r)}$ =
$(a+c)^*b + (a+b)^*c$

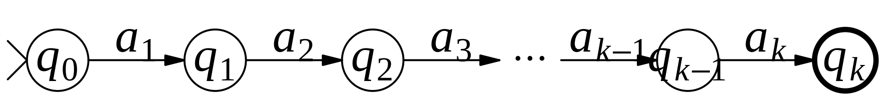
Reversing an FSA:

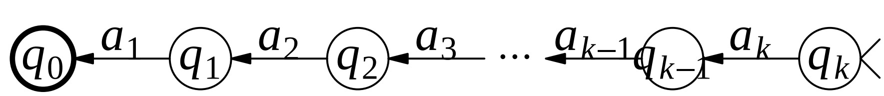
- Reverse direction of each transition (may create non-determinism).
- Make the start-state the final-state.
- Make each final-state a start-state (may create non-determinism).

Reverse of a Language L:

- $L^r = \{x^r : x \in L\}$, where $x^r = a_k a_{k-1} \cdots a_2 a_1$ if $x = a_1 a_2 \cdots a_{k-1} a_k$.

If L is regular, then L^r is also regular.

In M: 

In M^r: 

Question: If M has an error-state, then what will happen to it in M^r? Can the reversal process create unreachable states?

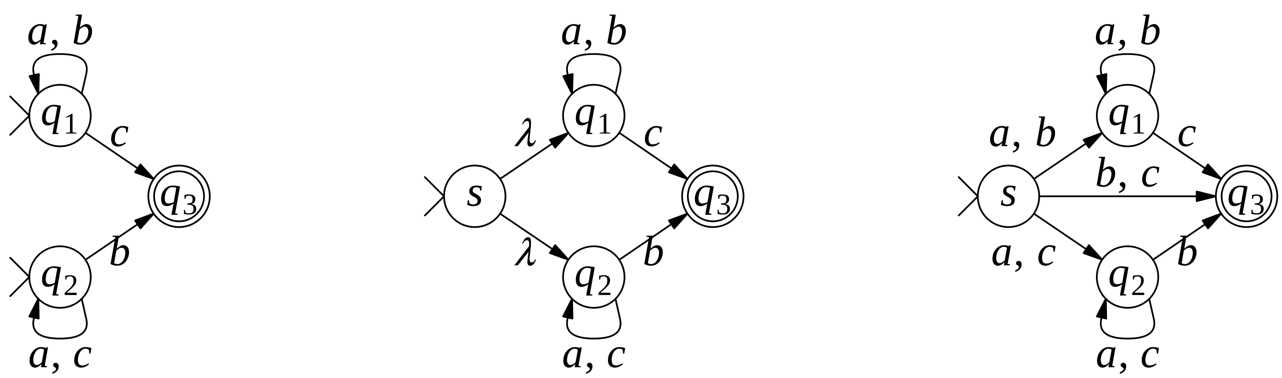
MULTIPLE START-STATES ELIMINATION USING λ-TRANSITIONS

λ-transition:

- An FSA can change state by using a λ-transition and without reading an input symbol.

Elimination of Multiple Start-states:

- Add a new state s and make it the only start-state.
- Add a λ-transition from s to each of the original start-state.
- No change in final-states or other transitions.



(i) Start-states = $\{q_1, q_2\}$. (ii) An equivalent FSA with 1 start-state and λ-moves. (iii) Another equivalent FSA with 1 start-state.

Question:

•? Give an example FSA to show that it is not enough to add a new state s, make it the only start-state, and for each a_j add the following transitions at s:

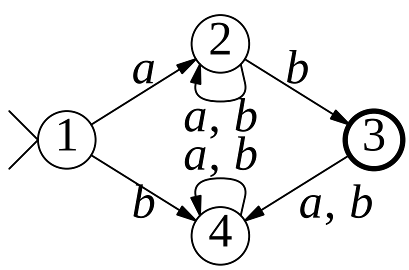
$$\delta(s, a_j) = \bigcup_{q_i} \delta(q_i, a_j), \text{ union over all start-states in } M.$$

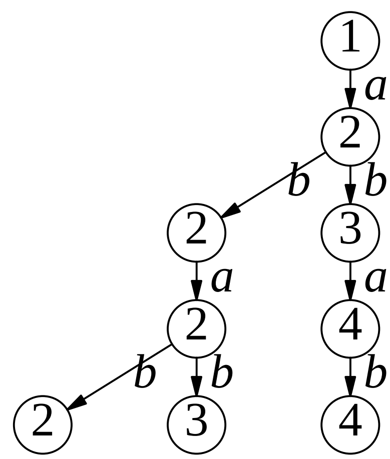
(We have to make the new start-state s also a final-state if one or more the original start-states is a final-state.)

•? Show the resulting FSA when we apply the above construction to the FSA shown at the top left. Does it change the language?

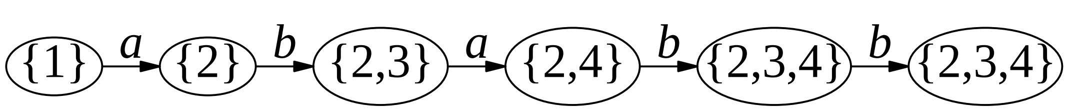
SUBSET-CONSTRUCTION METHOD FOR CONVERTING NFSA TO FSA

- The new FSA M' cannot simulate all alternative paths $\pi(x)$ in the original FSA M for an input string x, because the number of $\pi(x)$ can be exponentially large (in $|x|$) and M' has finitely many states.
- Instead, M' keeps track of the end points $E(x)$ of the paths $\pi(x)$; x is accepted $E(x)$ contains one or more final-states of M.
- The end-points of the paths $\pi(x)$ form a subset of Q in M, and there are only $2^{|Q|}$ many different subsets.
- If $x = a_1 a_2 \cdots a_j$ and $x' = x a_{j+1}$, then $E(x') = \bigcup_{q_i \in E(x)} \delta(q_i, a_{j+1})$.





#(paths $\pi(x)$ for processing $x = abab^n$) = n+2.



Use the subsets of Q as the states of the new FSA.

THE SUBSET-CONSTRUCTION

Avoid construction of unreachable states:

(1) Choose the set of all start-states in M as the start-state S_0 of the new FSA M'.

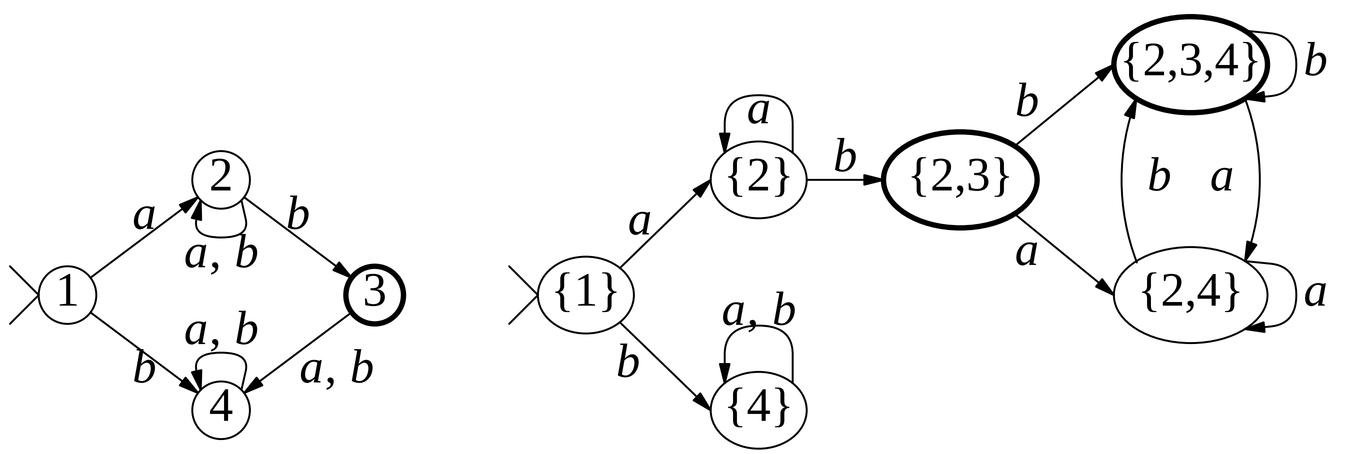
(2) While there is a state S_j for which the transitions have not been determined, do the following:
For each input symbol $a \in \Sigma$ in M,

(i) Let $S = \bigcup_{q_i \in S_j} \delta(q_i, a)$. (It may happen that $S = \emptyset$.)

(ii) If S is not already a state in M', then add it as a new state.

(iii) Add the transition $\delta(S_j, a) = S$ in M'.

(3) Make each state S_j in M' a final-state if it contains one or more final-states of M



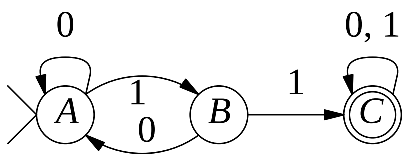
An NFSA The FSA obtained by the subset-construction

Note: If we did not have the dead-state 4 in the above example, then 4 would be removed from all states in the new FSA; the state {4} would now become $\emptyset$.

EXERCISE

1. Complete the partial description of the state A in the finite-state automaton $M_{has\text{-}11}$ below for the language $L_{has\text{-}11}$ (= the binary strings containing "11"), based on the descriptions of states B and C, to justify the transitions to and from A. Note that each state-description is in terms of the "past", i.e., the part of the input which is processed to arrive at the state.

$M_{has\text{-}11}$:



A = have not seen "11" and ⋯
B = have not seen "11" and just seen 1
C = seen "11"

Let $M^r_{has\text{-}11}$ be the non-deterministic automaton obtained by applying the reversal-operation to $M_{has\text{-}11}$; $L(M^r_{has\text{-}11}) = L^r_{has\text{-}11} = L_{has\text{-}11}$.

(a) Give a suitable description in English for the states of $M^r_{has\text{-}11}$ that would justify its transitions. What is the connection between these descriptions and the previous descriptions?

(b) Show the FSA obtained from $M^r_{has\text{-}11}$ by the subset-construction. Also describe the states of the new FSA in simple English in terms of the descriptions in (a).

2. Remove the redundant state 4 in the NFSA in page 6.4 and then apply the subset-construction. How does the result differ from the FSA shown above; do they accept the same language?

3. Apply the subset-construction for the NFSAs in page 6.1.

4. Consider a deterministic FSA for verifying multiplication of binary numbers by 3, with the usual least significant bit on the right. Also, consider a similar FSA for verifying multiplication by 2. The input alphabet for these machines should be $\{b_0, b_1, b_2, b_3\}$. Now, obtain a non-deterministic FSA for verifying multiplication by either of 2 and 3; convert it to a deterministic form.

PROJECTION OF A LANGUAGE AND λ-TRANSITION IN AN FSM

Projection:

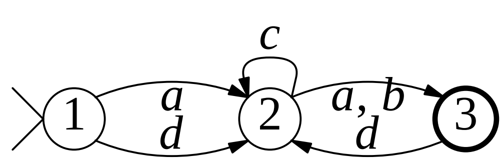
- If $x = x_1 c x_2 c x_3 \cdots c x_k$, where some of x_i's can be λ, none of x_i contains c, and $k \geq 1$, then the projection $\Pi_c(x) = x_1 x_2 \cdots x_k$, which is simply x minus all occurrences of c.
- $\Pi_c(L) = \{\Pi_c(x): x \in L\}$.

Theorem:

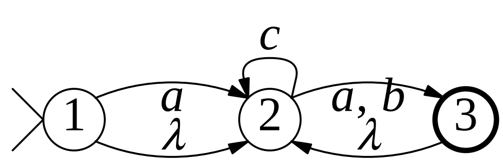
- For any language L and the symbols $a \neq b$, $\Pi_a(\Pi_b(L)) = \Pi_b(\Pi_a(L)))$.
- If L is a regular language, then there is NFSM for $\Pi_c(L)$ containing λ-transitions.

Example.

M:



$\Pi_d(M)$:

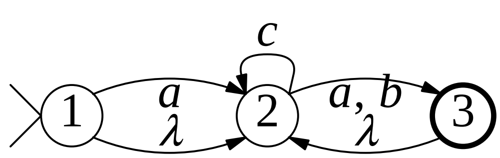


ELIMINATION OF λ-TRANSITIONS

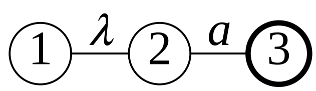
λ-transition:

- The FSA can change its state without reading an input symbol.

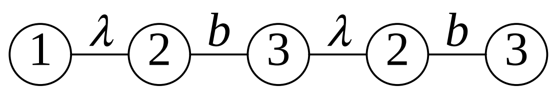
M:



$a \in L(M)$:



$bb \in L(M)$:



Elimination of λ-moves in M gives possibly an NFSA M':

- M and M' have the same states, and the same final-states.
- M' may have multiple start-states (due to λ-transitions from start-state of M) and non-deterministic transitions.

Define: $\lambda(q_i) = \{q_j : q_j$ is reachable from q_i by zero or more λ-transitions$\}$; $q_i \in \lambda(q_i)$.

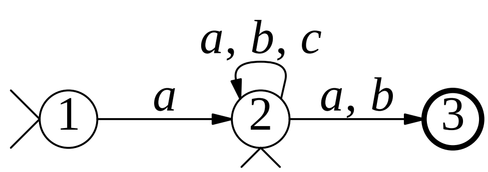
Algorithm:

(1) Make each state in $\lambda(q_0)$ a start-state in M'.

(2) For each $\delta(q_i, a_j) = q_k$ in M for $a_j \neq \lambda$, let $\delta(q_i, a_j) = \lambda(q_k)$ in M'.

Example. For above M, $\lambda(1) = \{1, 2\}$, $\lambda(2) = \{2\}$, and $\lambda(3) = \{2, 3\}$.

M':



THE EFFECT OF INTRODUCING ERRORS IN A REGULAR LANGUAGE

Language L **modified by** *one replacement error:*

- $RE_1(L)$ = {x: x differs from some $y \in L$ in one position}.
- L and $RE_1(L)$ have the same alphabet.

$$\boxed{\text{If } L \text{ is regular, then } RE_1(L) \text{ is also regular.}}$$

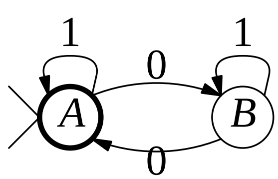
Example. If $L = L_{0\text{-}div\text{-}2} = L_{0\text{-}even}$, then $RE_1(L) = L_{0\text{-}odd}$.

$L_{0\text{-}even}$ = {λ, 1, 00, 11, 001, 010, 100, 111, ⋯}
$L_{0\text{-}odd}$ = { 0, 01, 10, 000, 011, 101, 110, ⋯}

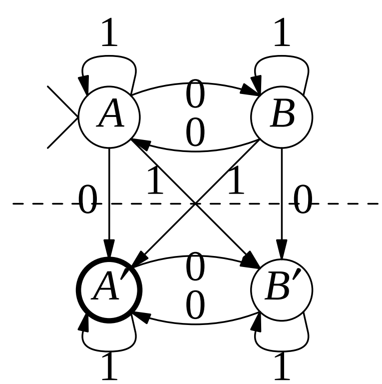
Building an $M(RE_1(L))$ **from** $M(L)$**:**

- The construction below applies to any FSA.

$M_{0\text{–}even}$



NFSA for $RE_1(L_{0\text{-}div\text{-}2})$

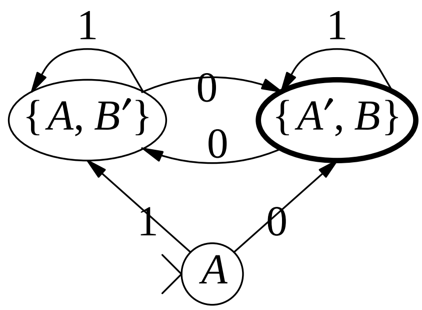


transitions before error

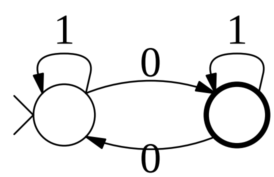
error-transitions

transitions after error

$M(RE_1(L_{0\text{-}even}))$:



reduced $M(RE_1(L_{0\text{-}even}))$:



(A merged with AB')

EXERCISE

1. Apply the above method to obtain an FSA for $RE_1(L_{has\text{-}11})$. Show all details of coversion of NFSA to FSA and the details of state-minimization.

2. How will you generalize the above construction for exactly k (≥ 2) replacement errors? Illustrate the construction using $k = 2$ and $M_{0\text{-}div\text{-}2}$. (The generalization to $\geq$1 errors is also easy.)

3. Show that $RE_{L'}(L) = \{uv'w\colon v' \in L', |v| = |v'|, \text{ and } uvw \in L\}$ is regular if both L and L' are regular. Note that v may equal v'. (Hint: an NFSA for $RE_{L'}(L)$ will have three phases: for the part u (before the error), v', and w (after the error).)

4. Let $DE_1(L) = \{xy\colon xay \in L$ for some x, a, and $y\}$ = the set of strings obtained by *deletion* of a symbol from strings in L. One can show that $DE_1(L)$ is regular by giving a method for the construction of a NFSA for $DE_1(L)$ from an FSA for L where the deletion operation is modeled by λ-transitions. Illustrate your method by using $M_{has\text{-}11}$ as an example; show the NFSA after the introduction of λ-transitions (keep the states "before deletion" distinct from those "after deletion" similar to that for the case of $RE_1(L)$).

5. A similar result holds for the *insertion* error. State the result clearly.