Coming soon: PS7

Photo: Haris Angelidakis

Class 15: Proof of DFA = Reg-Fun

University of Virginia
cs3120: DMT2
Wei-Kai Lin

Recall: Negation of regular expression

- Given regular expression e.
- Is there a regular expression e' that matches string x if and only if e does not match x?
- Yes! By **Reg-Fun $=$ DFA-Comp**, and by negating DFA.

How to convert?

Reg-Fun $\subseteq$ DFA-Comp

TCS, Section 6.4.2

For each DFA M, there is an equivalent regular expression e

Proof? Generic way to construct such M for all e.

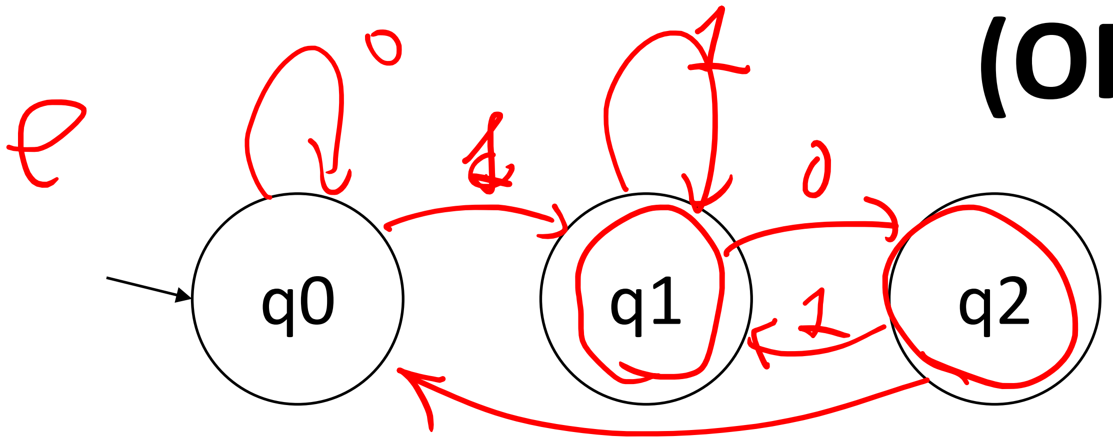
Algorithm: input $\cancel{e}$, output $\cancel{M}$

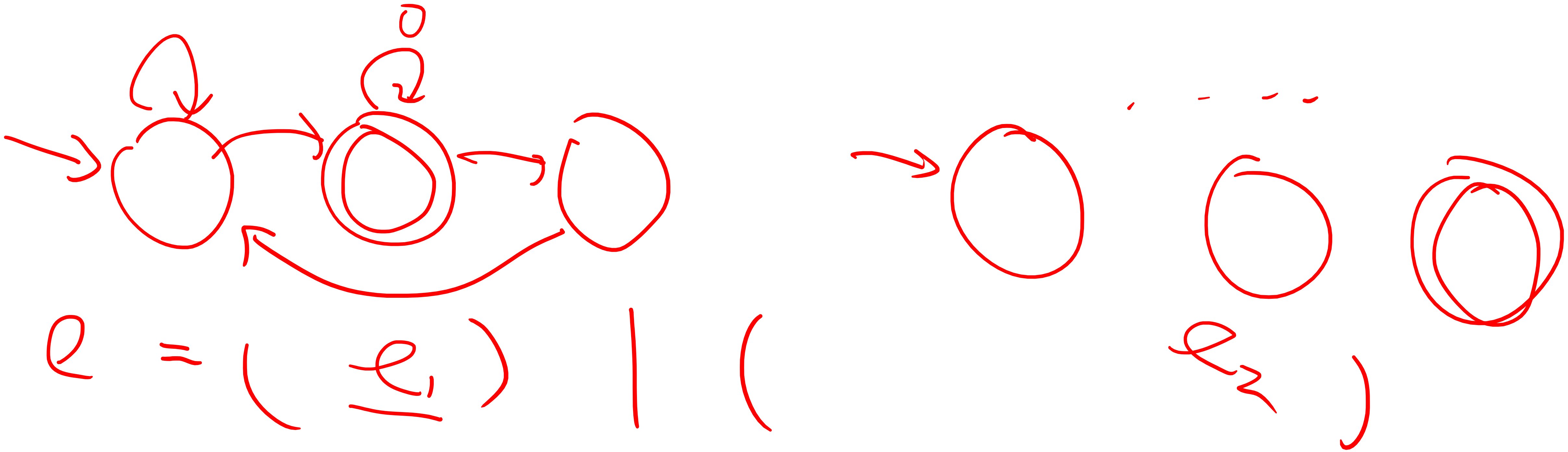
M e

Proof idea 1: Consider only one Accept state (OR later)

Proof idea 2: Induction on subsets of states (see next

Proof idea 1: Consider only one Accept state (OR later)





Proof idea 2: Induction on subsets of states

For any M, let $\{0, 1, 2, \ldots, C-1\}$ = [C] be states in M, let v be the initial and w be the accept state.

Let t be natural num. $t \leq C$

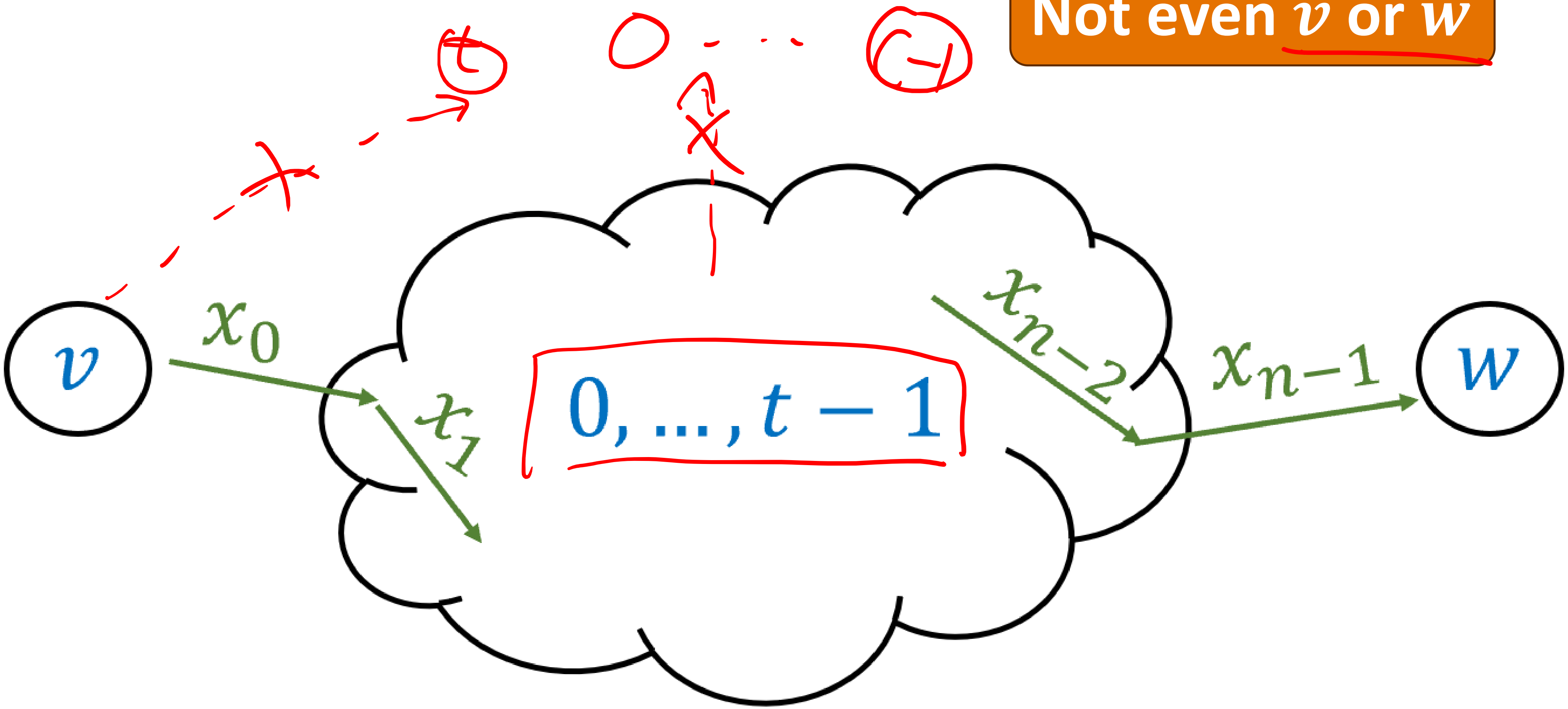
Consider the subset $[t] = \{0,1,\ldots,t-1\}$.

Let L_t be the strings go from v to w **only through** $[t]$.

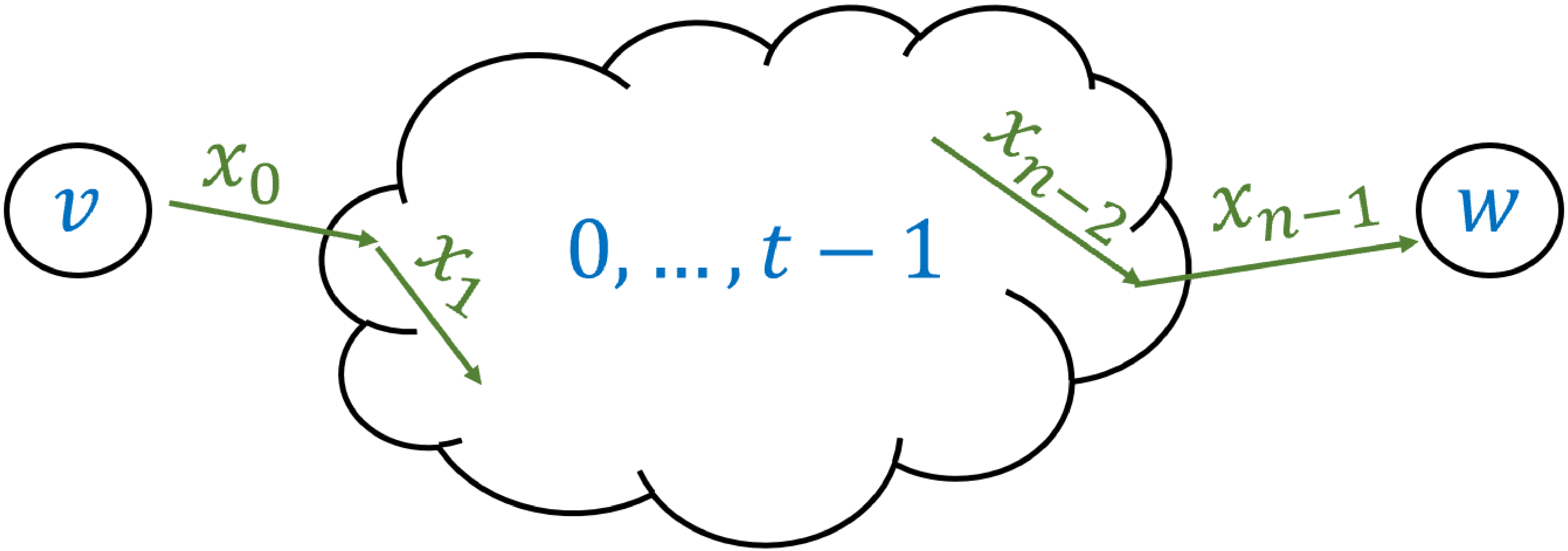
Consider the subset $[t] = \{0,1,\ldots,t-1\}$.

Let $L_t(v,w)$ be the strings go from v to w **only through nodes in $[\boldsymbol{t}]$**.

Not even $\boldsymbol{v}$ or $\boldsymbol{w}$



Induction predicate, P(t):
There exists a regular expression e_t matches $L_t(v, w)$



Induction predicate, P(t):
There exists a regular expression e_t such that matches $L_t(v, w)$

Base case, P(0).

If $v = w$:

- 0 edge

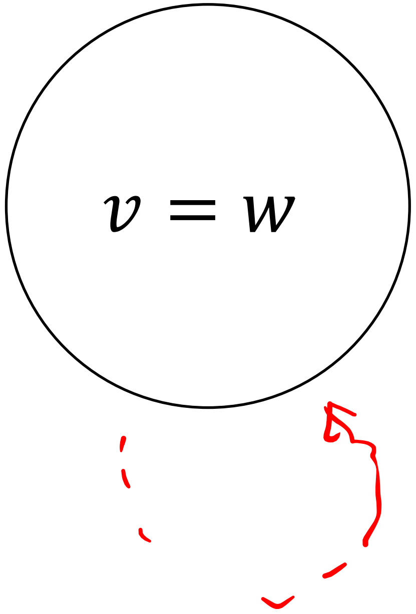
  e_0 : $\emptyset$ ""

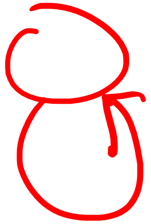
- 1 edge

  e_0 : b

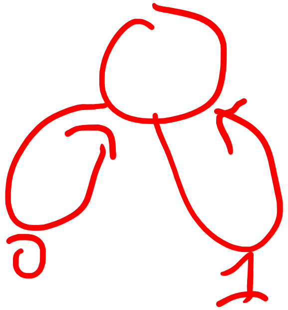
- 2 edges

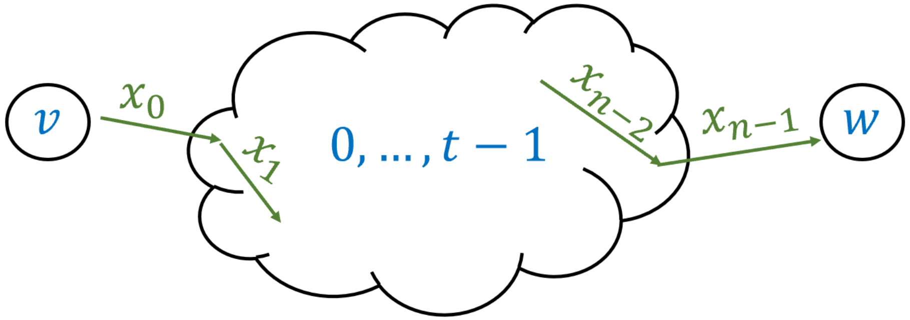
  e_0 : $(0|1)$





$b = 0$ or 1





Induction predicate, P(t):
There exists a regular expression e_t such that matches $L_t(v, w)$

Base case, P(0).

If $v \neq w$:

- 0 edge

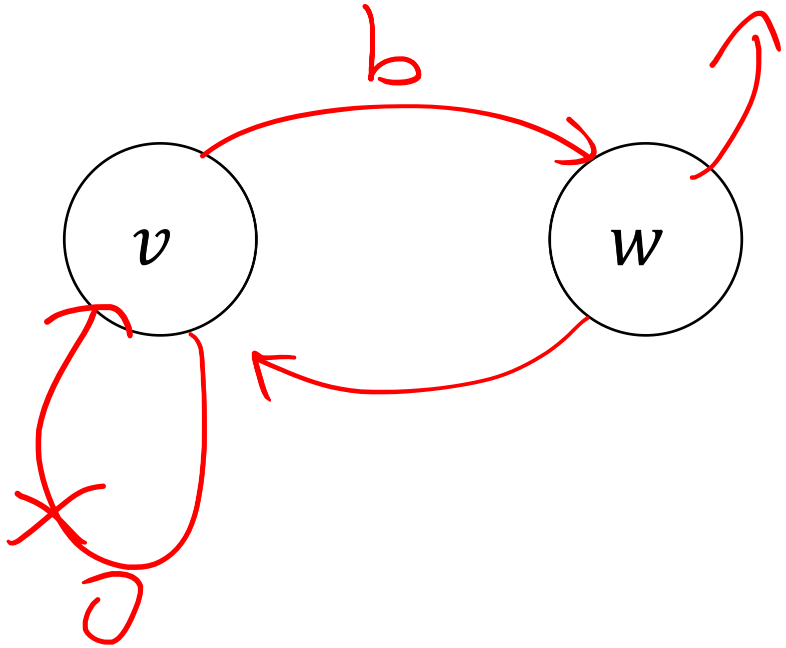
  e_0 : ϕ

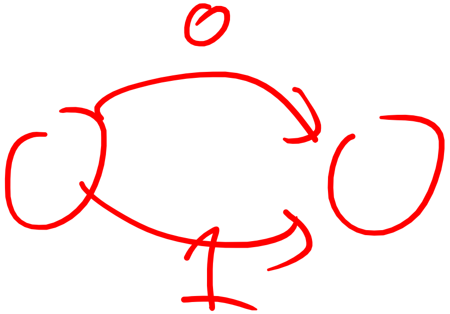
- 1 edge

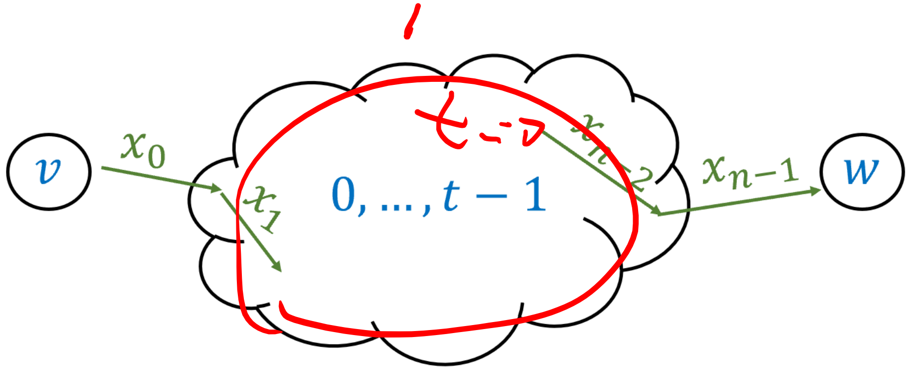
  e_0 : b , $b \in \{0, 1\}$

- 2 edges

  e_0 : $(0|1)$







Induction predicate, P(t):
There exists a regular expression e_t such that matches $L_t(v, w)$

Inductive case, P(t) holds.
Want P(t+1):

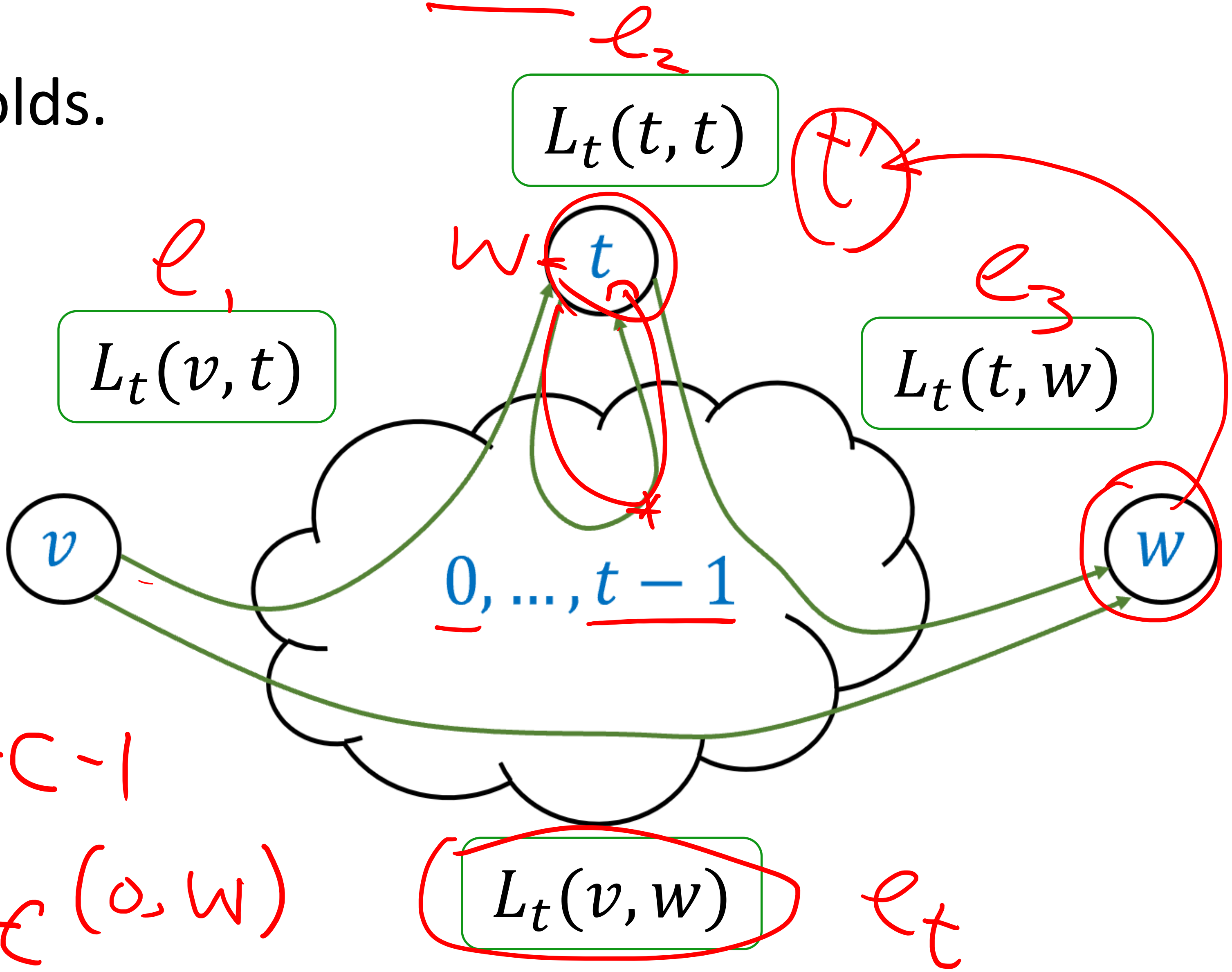
$e_{t+1} =$

$(e_1')(e_2')^*(e_3')$

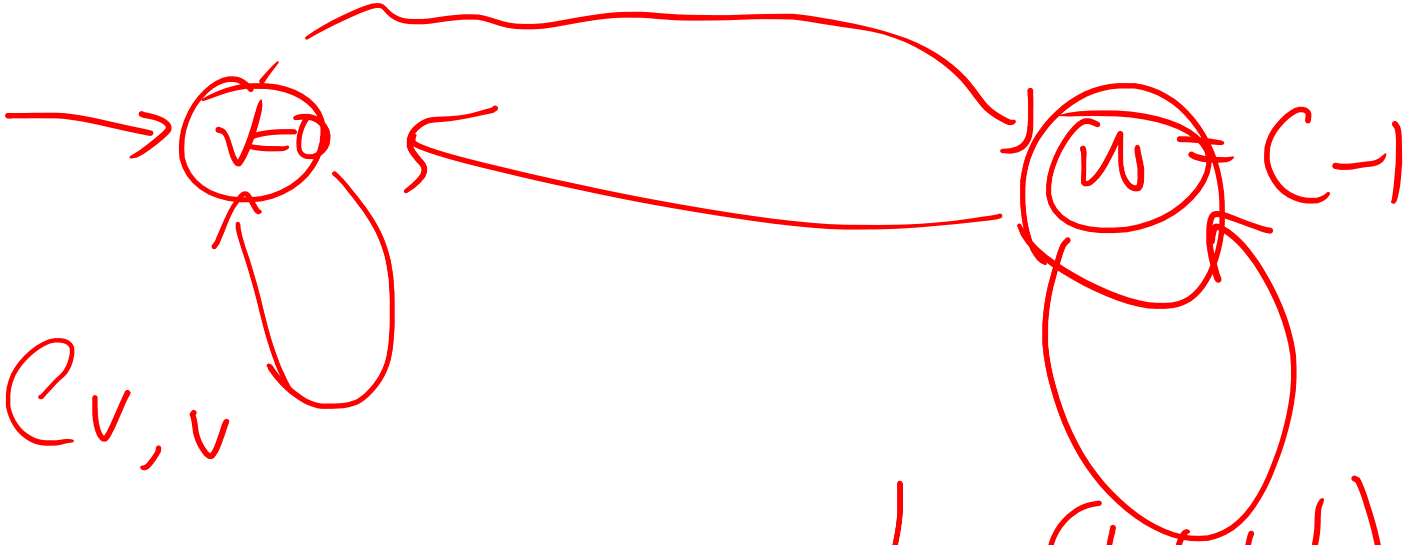
$| \; (e_t)$

$P(c)$: $v=0$, $w=c-1$

$L_c(w, w)$, $L_c(0, w)$



$L_C(v,w) \Rightarrow e_{v,w}$



$L_C(w,w) \Rightarrow e_{w,w}$

$$e = (e_{v,v})^*\left(e_{v,w}\left(\left(e_{w,w} \mid (e_{w,v})(e_{v,v})^*(e_{v,w})\right)^*\right)\right)$$

DFA $\Rightarrow$ Regular Expression

Example?

Next? Regular Expression $\Rightarrow$ DFA

DFA-Comp $\overset{\supseteq}{\cancel{\subseteq}}$ Reg-Fun

Through Non-deterministic Finite Automata (NFA)

Recall: Syntax of Regular Expressions

Definition 6.6 (Regular expression)

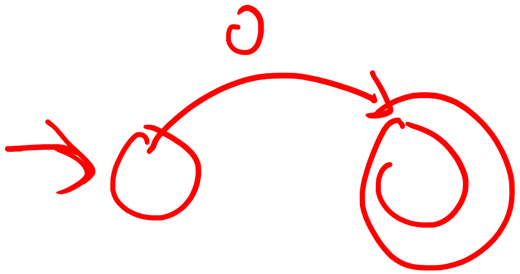
A *regular expression* e over an alphabet Σ is a string over $\Sigma \cup \{(,),|,*,\emptyset,\texttt{""}\}$ that has one of the following forms:

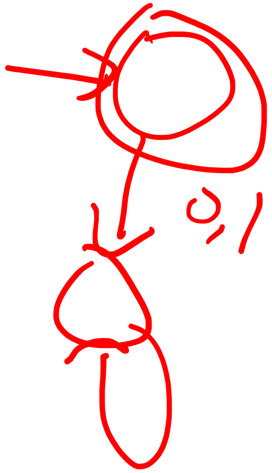
1. $e = \sigma$ where $\sigma \in \Sigma$
2. $e = (e'|e'')$ where e', e'' are regular expressions.
3. $e = (e')(e'')$ where e', e'' are regular expressions. (We often drop the parentheses when there is no danger of confusion and so write this as $e'\ e''$.)
4. $e = (e')^*$ where e' is a regular expression.

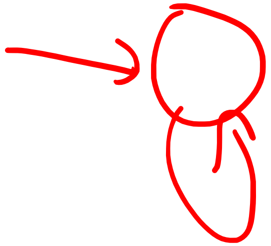
Finally we also allow the following "edge cases": $e = \emptyset$ and $e = \texttt{""}$. These are the regular expressions corresponding to accepting no strings, and accepting only the empty string respectively.

Base Cases are Easy

$e =$ 0 1 “” $\emptyset$



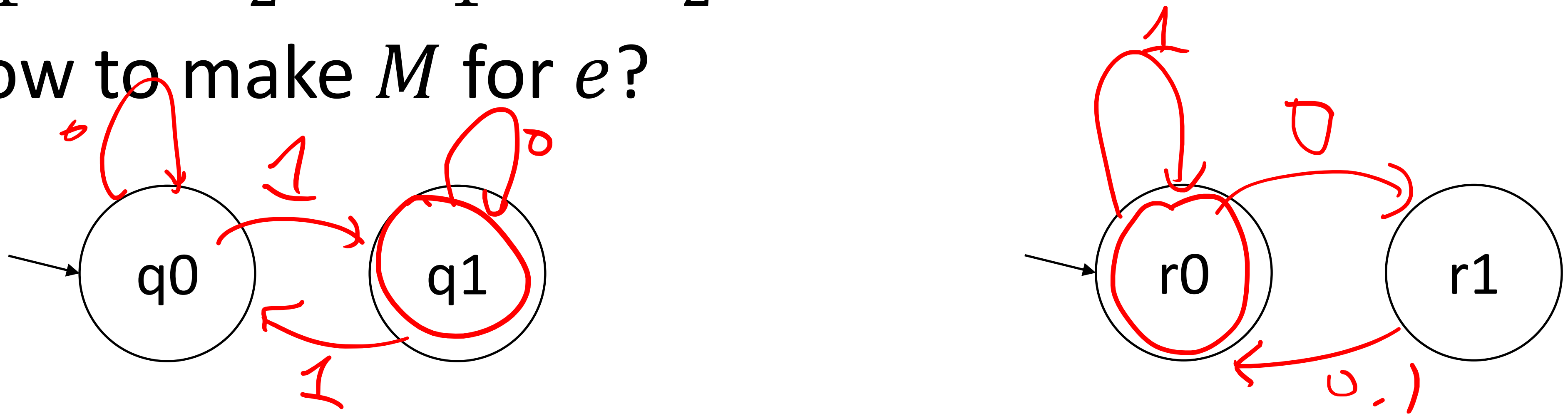




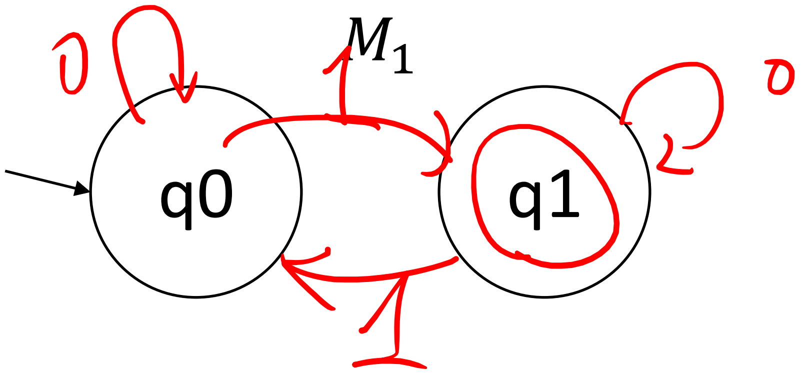
Recursive Case: OR

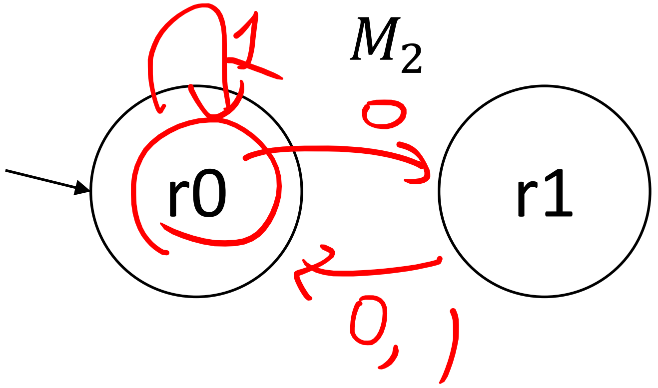
$e = (e_1)|(e_2)$. Suppose we have corresponding DFA M_1 and M_2 for e_1 and e_2.

How to make M for e?

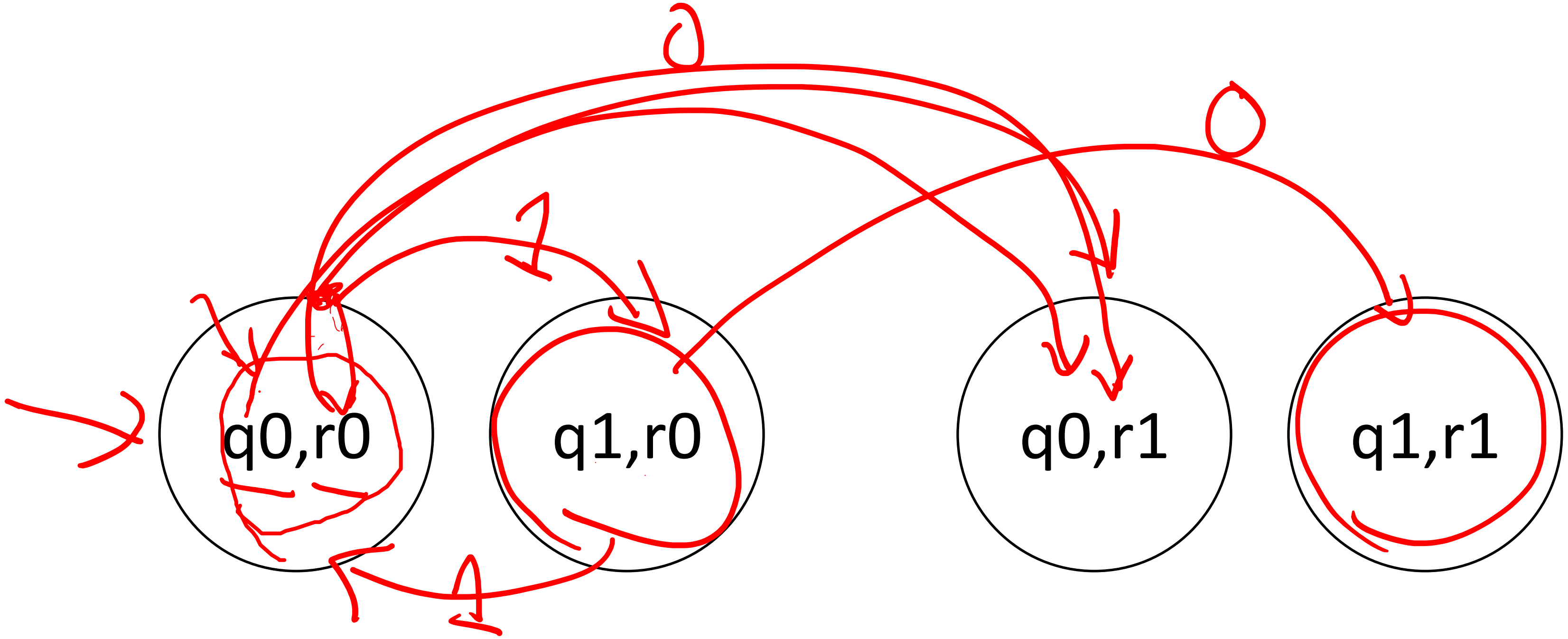


Idea: the states of M is the product set of M_1 and M_2





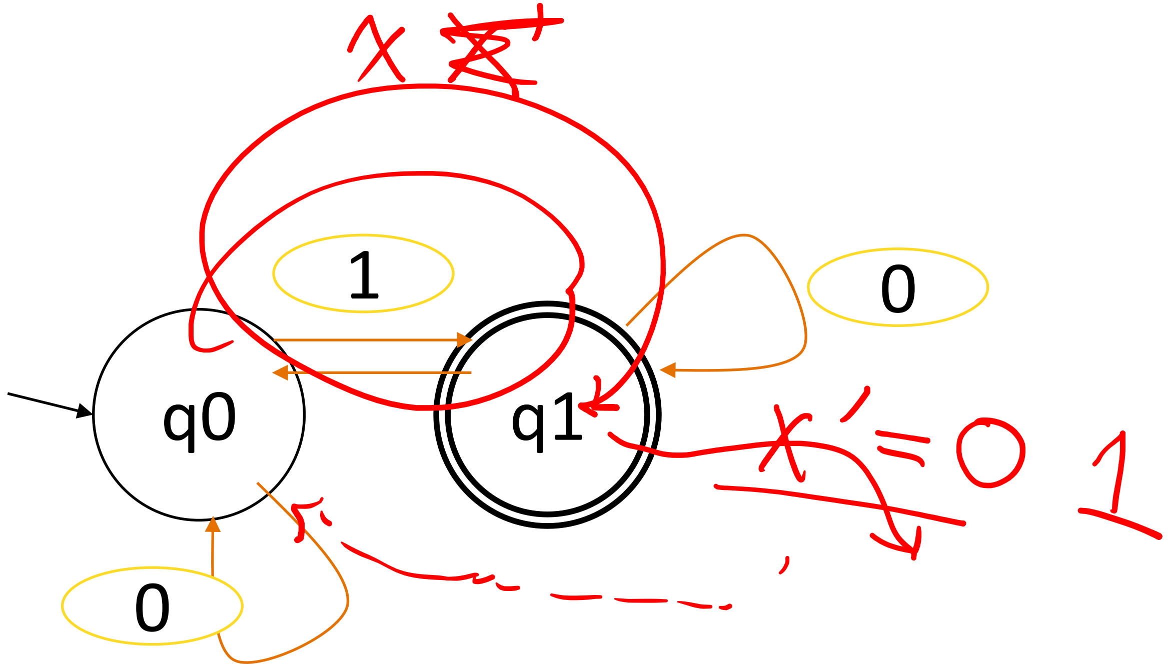
M



Recursive Cases: Kleene Star

$e = (e_1)^*$

M_1



Hard: unclear how to splits string x in concat (and *).

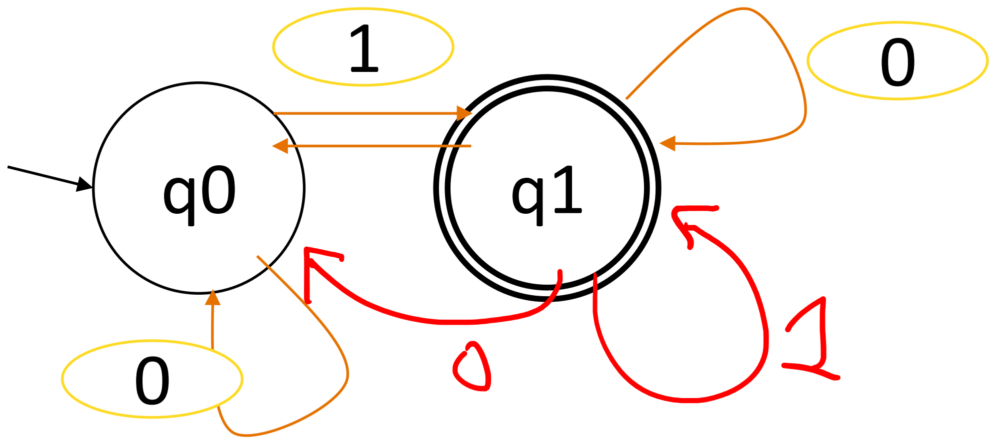
What's the **next state** (when accepted) in M_1?

Concat is same.

Big Idea: Non-deterministic

$e = (e_1)^*$

M_1



Allow transition to **multiple states** (clearly, not DFA)
Accept if exist a **path to accept**

How should we change our DFA description to allow for *choices*?

A **(deterministic)** ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

1. Q – a finite set (the *states*)
2. Σ – a finite set (the *alphabet*)
3. $\delta: Q \times \Sigma \rightarrow Q$ – transition function
4. $q_0 \in Q$ – the start state
5. $F \subseteq Q$ – the set of accept states

A **nondeterministic** ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

1. Q – a finite set (the *states*)
2. Σ – a finite set (the *alphabet*)
3. $\boldsymbol{\delta: Q \times \Sigma \rightarrow pow(Q)}$
4. $q_0 \in Q$ – the start state
5. $F \subseteq Q$ – the set of accept states