HW 3 due this Friday, Feb 13 (10:00pm)

Quiz 4 coming soon

Photo: [Movie Poster]

Class 8: Reg Exp ⇒ DFA (Non-deterministic)

University of Virginia
CS3120: DMT2
https://weikailin.github.io/cs3120-toc
Wei-Kai Lin

Plan

Reg-Fun ⊆ DFA-Comp

Non-deterministic FA

NFA ⊆ DFA-Comp

Today: [Sipser] Section 1.2

- Formal definition of a nondeterministic finite automaton
- Equivalence of NFAs and DFAs

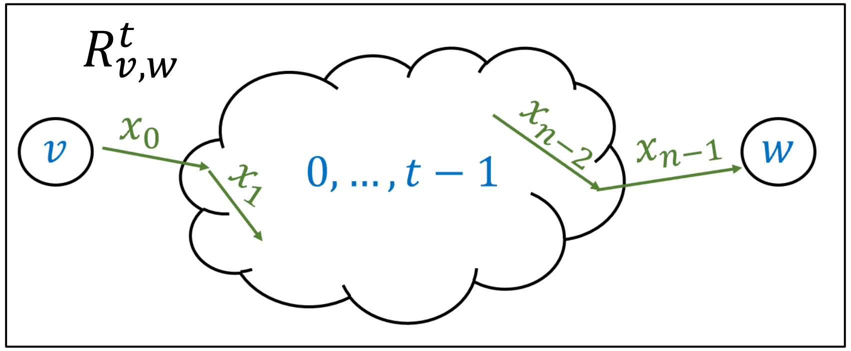
Recap: Reg-Fun ⊇ DFA-Comp

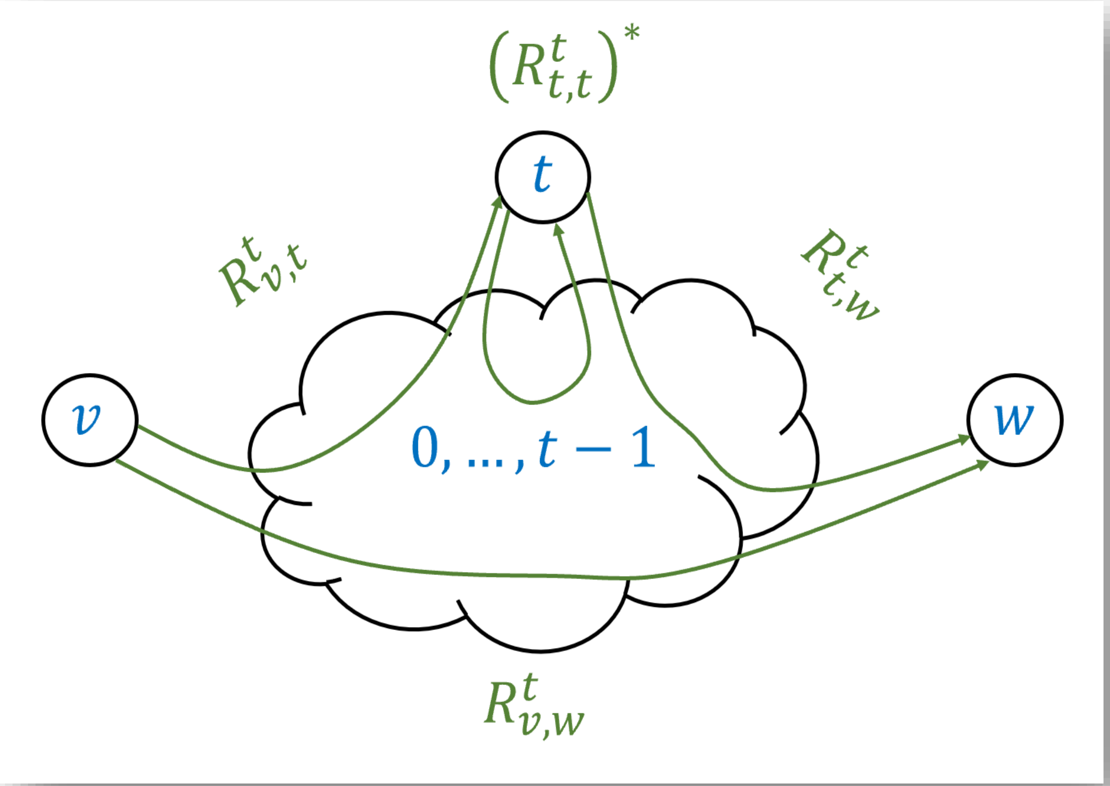
For each DFA $\boldsymbol{M}$, there is an equivalent regular expression $\boldsymbol{e}$.

Proof by induction:

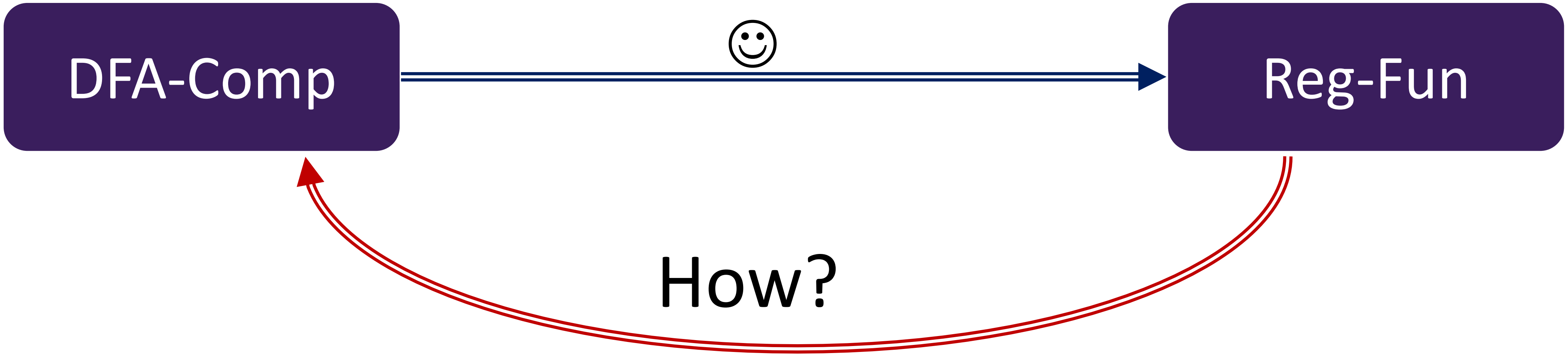
Consider the subset $[t] = \{0,1, \dots, t-1\}$.

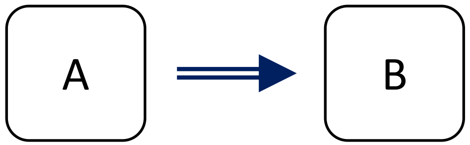
Let $R^t_{v,w}$ be the strings go from v to w **only through nodes in** $[\boldsymbol{t}]$.





High-Level Proof Plan





We can convert every $M_A \in A$ to a $M_B \in B$
Such that for all x, M_A accepts x iff M_B accepts x

Reg-Fun $\subseteq$ DFA-Comp

Introduction to the Theory of Computation
Section 1.2. Michael Sipser.

Recall: Syntax of Regular Expressions

Definition 6.6 (Regular expression)

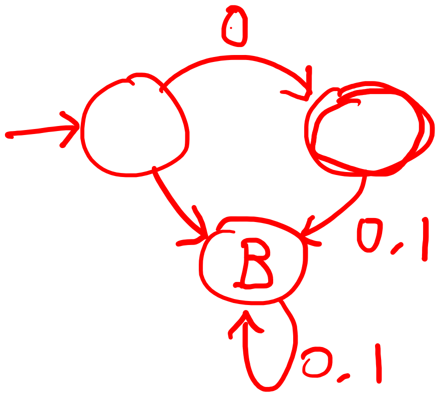
A *regular expression* e over an alphabet Σ is a string over $\Sigma \cup \{(,),|,*,\emptyset,\texttt{""}\}$ that has one of the following forms:

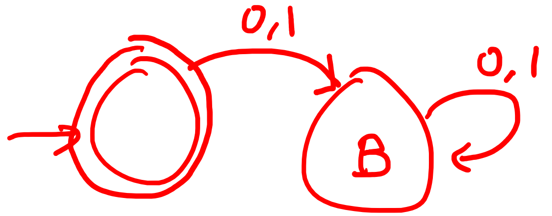
1. $e = \sigma$ where $\sigma \in \Sigma$
2. $e = (e'|e'')$ where e', e'' are regular expressions.
3. $e = (e')(e'')$ where e', e'' are regular expressions. (We often drop the parentheses when there is no danger of confusion and so write this as $e'\ e''$.)
4. $e = (e')^*$ where e' is a regular expression.

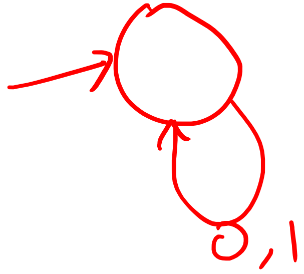
Finally we also allow the following "edge cases": $e = \emptyset$ and $e = \texttt{""}$. These are the regular expressions corresponding to accepting no strings, and accepting only the empty string respectively.

Base Cases are Easy

$e =$ 0 1 “” $\emptyset$





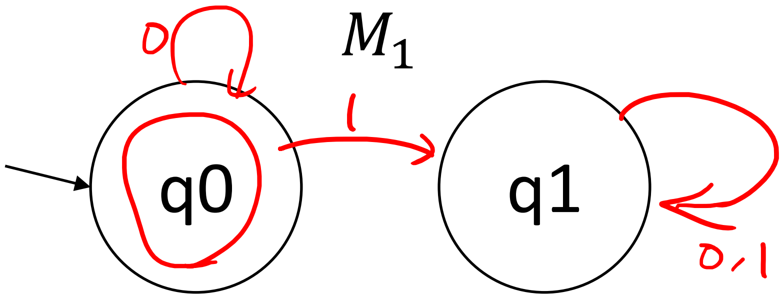


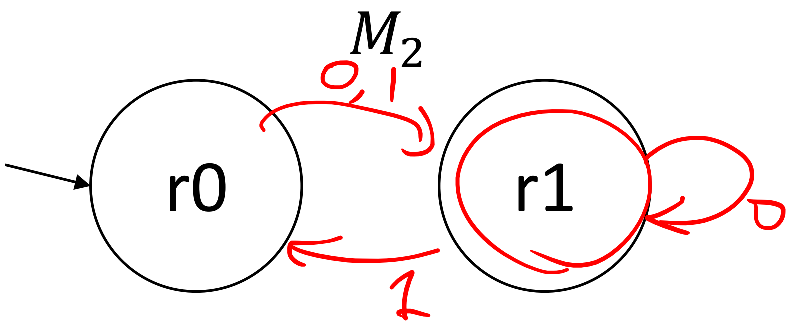
Recursive Case: OR

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

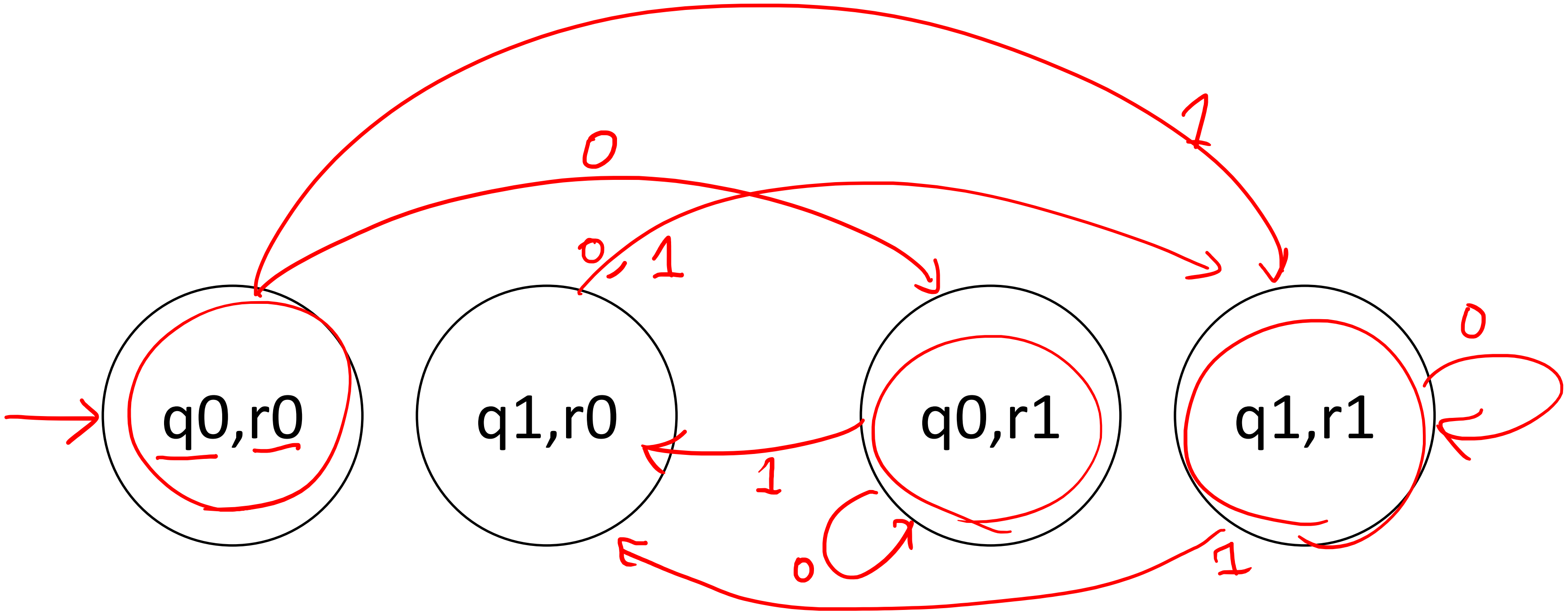
How to make M for e?

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





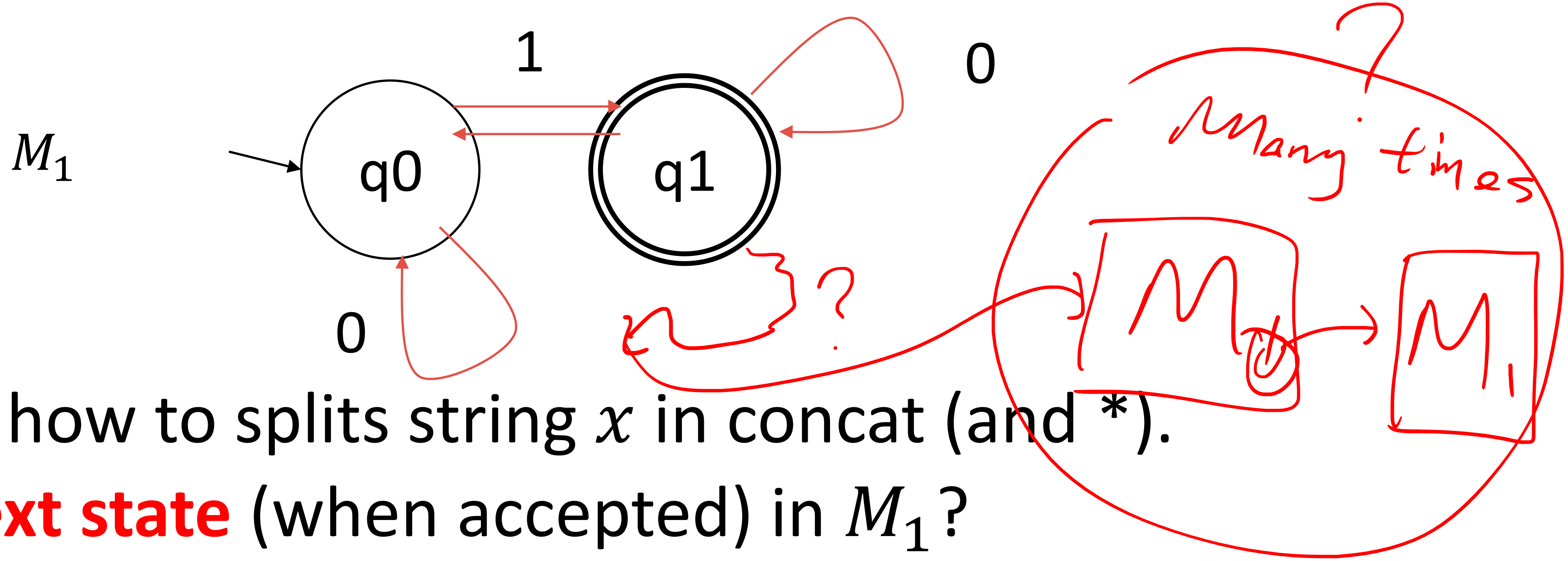
M



Recursive Cases: Kleene Star

$$e = (e_1)^*$$

Suppose M_1 is equivalent to e_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

$$e = (e_1)^*$$

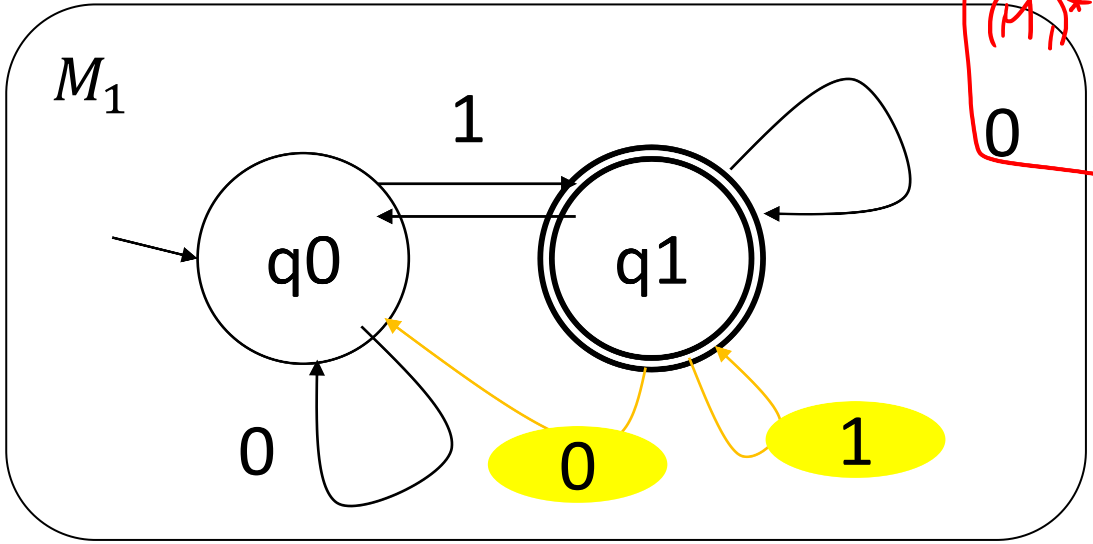
Match $x = 1\ 0\ 0\ 1\ 0$

Suppose M_1 is equivalent to e_1

$M_1 \equiv XOR$

$(M_1)^* \equiv (XOR)^*$

$\equiv (0|1)^*$



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*?

$f: D \to R$

A **(deterministic)** ***finite automaton*** over alphabet $\{0,1\}$ is a tuple (C, T, S) where:

1. C --- the number of *states*
2. $T: [C] \times \{0,1\} \to [C]$ a transition function
3. $S \subseteq [C]$ --- the set of accept states

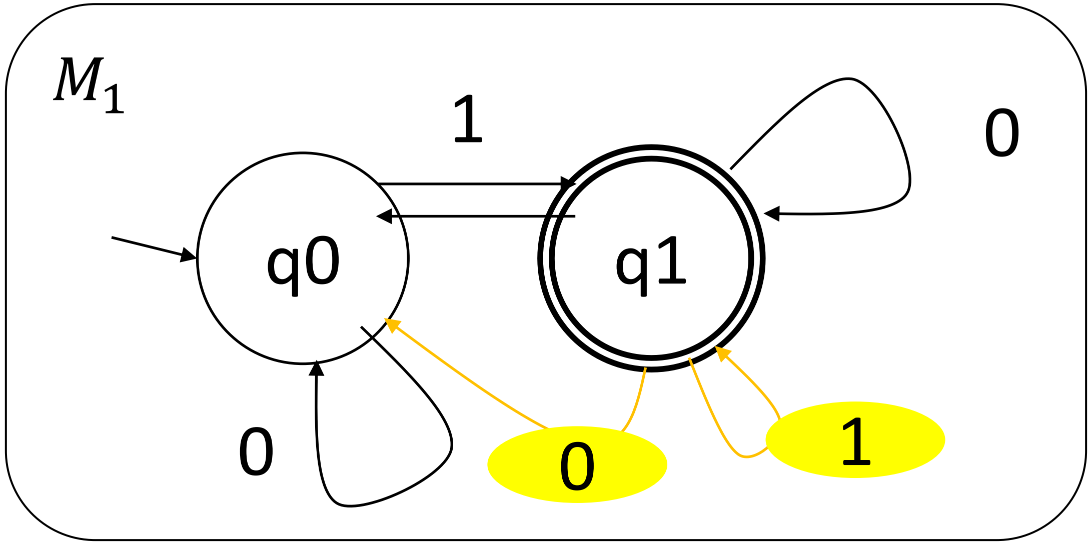
A **Nondeterministic** ***Finite Automaton*** over alphabet $\{0,1\}$ is a tuple (C, T, S) where:

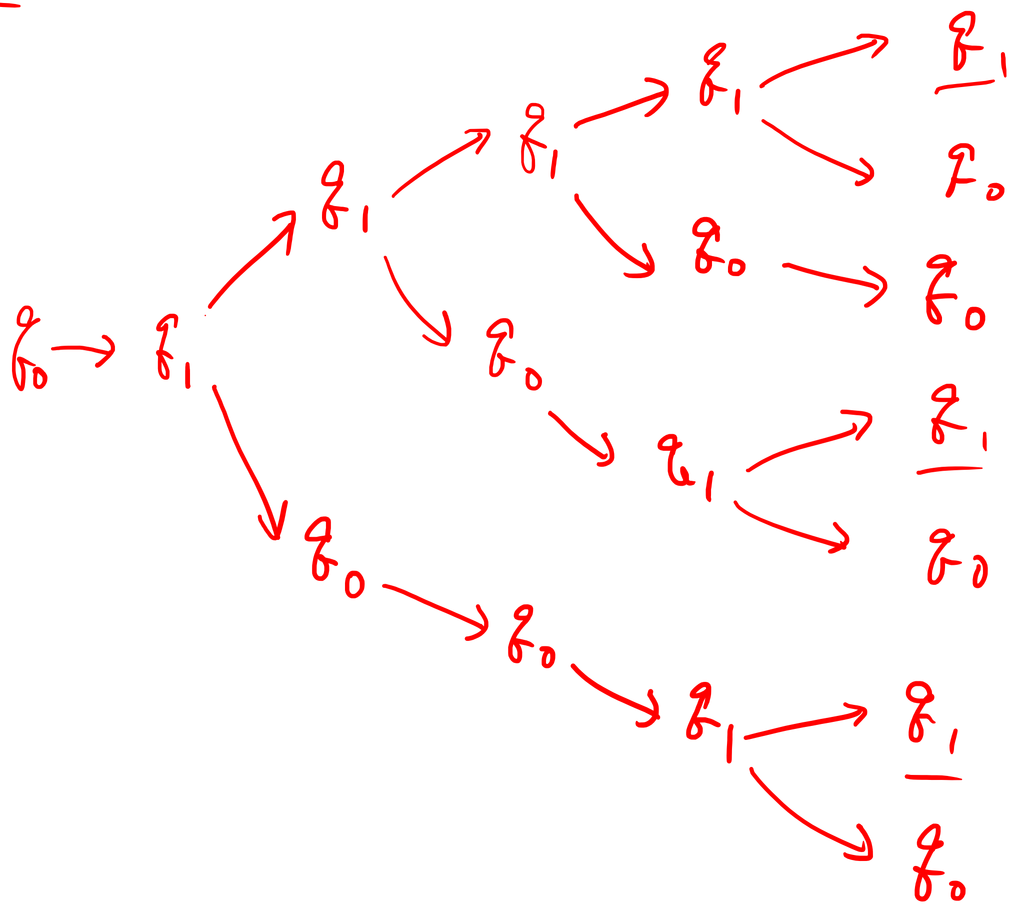
1. C --- the number of *states*
2. $T: [C] \times \{0,1\} \to \boldsymbol{pow}([C])$ a transition function
3. $S \subseteq [C]$ --- the set of accept states

How to evaluate an NFA?
Try all possible "choices"?!

How can we try all possible “executions”?

$e = (e_1)^*$,   Match $x = 1\quad 0\quad 0\quad 1\quad 0$





Defining the NFA Model

A **Nondeterministic** ***Finite Automaton*** over alphabet {0,1} is a tuple (C, T, S) where:

1. C --- the number of *states*
2. $T: [C] \times \{0,1\} \rightarrow \boldsymbol{pow}([C])$
   a transition function
3. $S \subseteq [C]$ --- the set of accept states

Recall our DFA model:

The string $x = b_0 b_1 \dots b_n$ is matched by the DFA $M = (C, T, S)$ iff there are states $s_0, s_1, s_2, \dots, s_n \in [C]$ such that $\boldsymbol{s_{i+1} = T(s_i, b_i)}$ for all $i = 0, \dots, n-1$ and $s_0 = 0$ and $s_n \in S$.

Defining the NFA Model

A **Nondeterministic** ***Finite Automaton*** over alphabet {0,1} is a tuple (C, T, S) where:

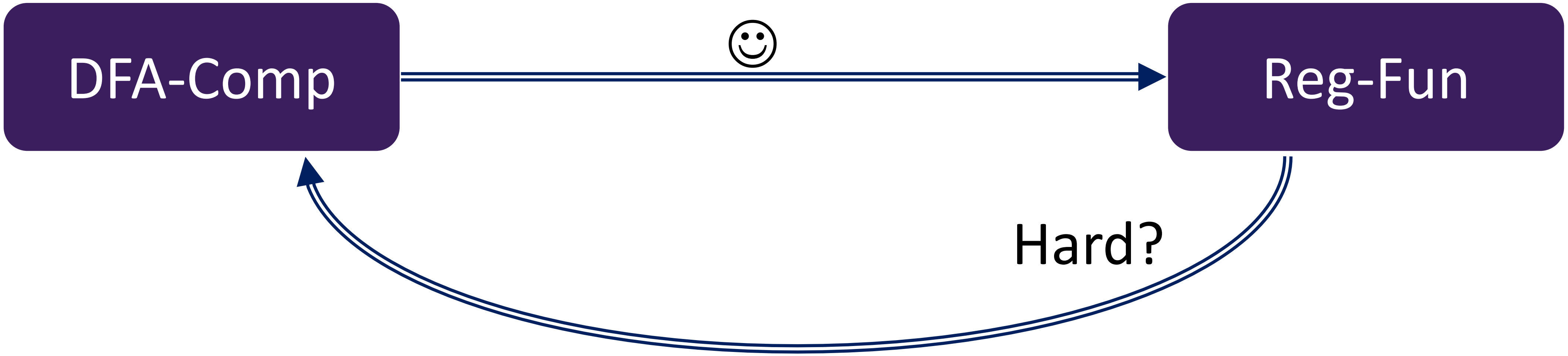
1. C --- the number of *states*
2. $T: [C] \times \{0,1\} \rightarrow \boldsymbol{pow}([C])$
   a transition function
3. $S \subseteq [C]$ --- the set of accept states

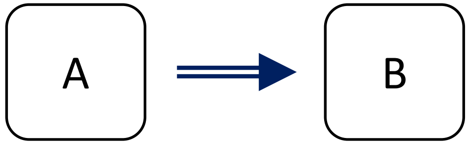
The string $x = b_0 b_1 \ldots b_n$ is matched by the NFA $M = (C, T, S)$ iff there are states $s_0, s_1, s_2, \ldots, s_n \in Q$ such that
$\boldsymbol{s_{i+1}} \in \boldsymbol{T}(\boldsymbol{s_i}, \boldsymbol{b_i})$ for all $i = 0, \ldots, n-1$ and
$s_0 = 0$ and
$s_n \in S$.

Recall our DFA model:

The string $x = b_0 b_1 \ldots b_n$ is matched by the DFA $M = (C, T, S)$ iff there are states $s_0, s_1, s_2, \ldots, s_n \in [C]$ such that $\boldsymbol{s_{\mathrm{i+1}}} = \boldsymbol{T}(\boldsymbol{s_i}, \boldsymbol{b_i})$ for all $i = 0, \ldots, n-1$ and $s_0 = 0$ and $s_n \in S$.

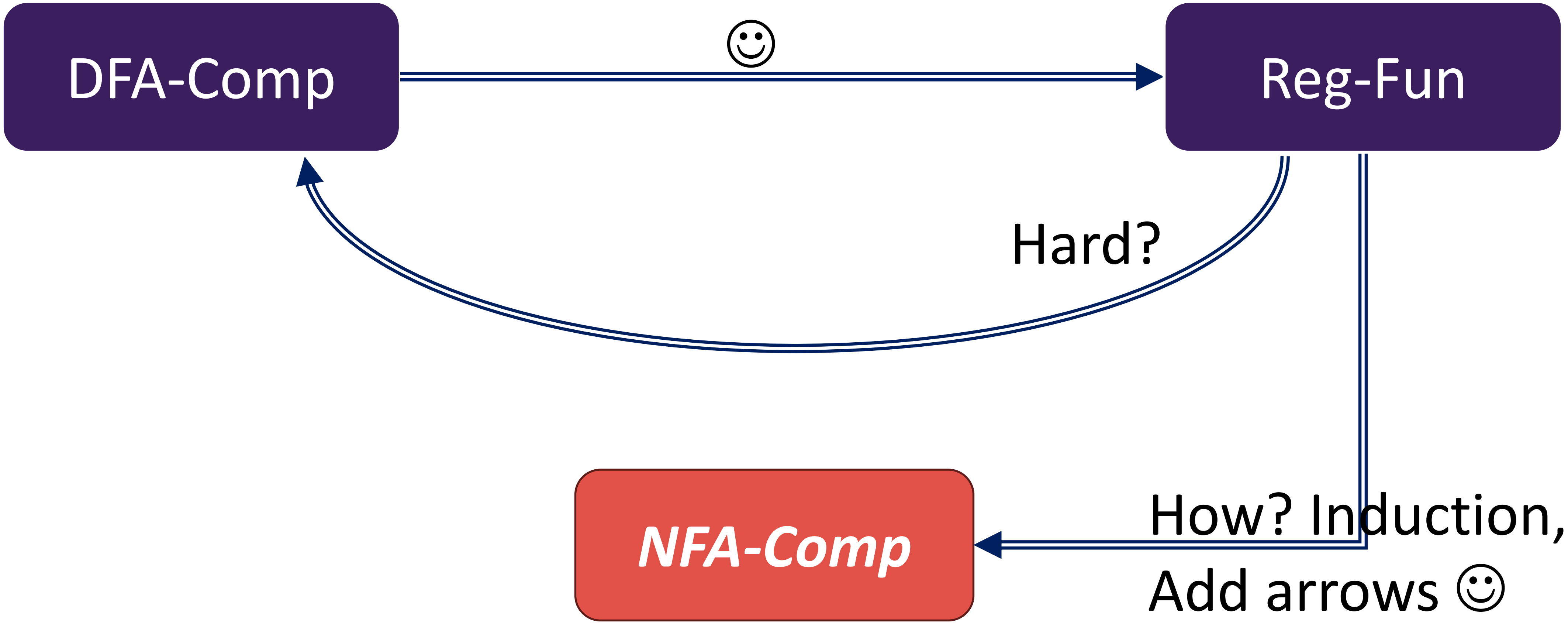
Recalling the High-Level Proof Plan





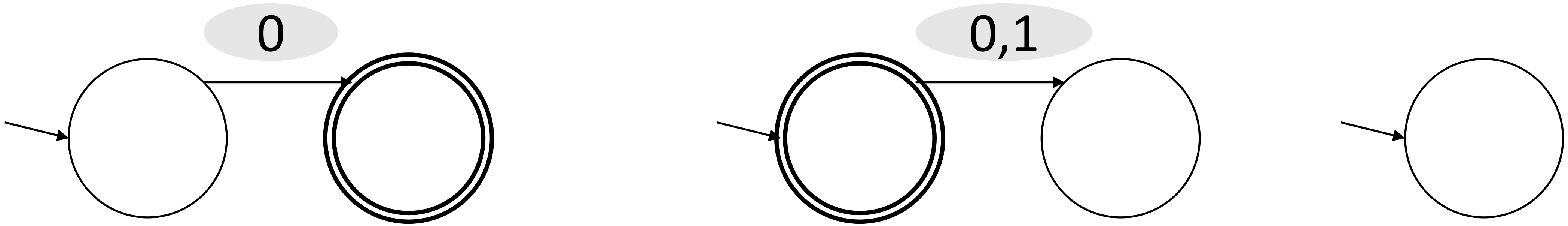
We can convert every $M_A \in A$ to a $M_B \in B$
Such that for all x, M_A accepts x iff M_B accepts x

High-Level Proof Plan



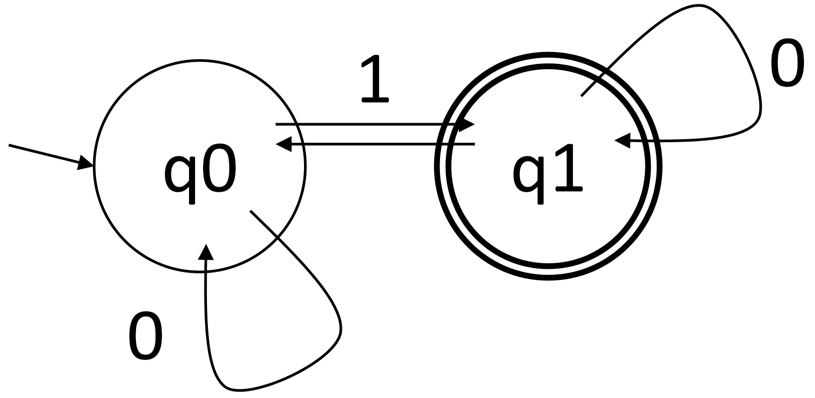
Base Cases are Similar to DFA

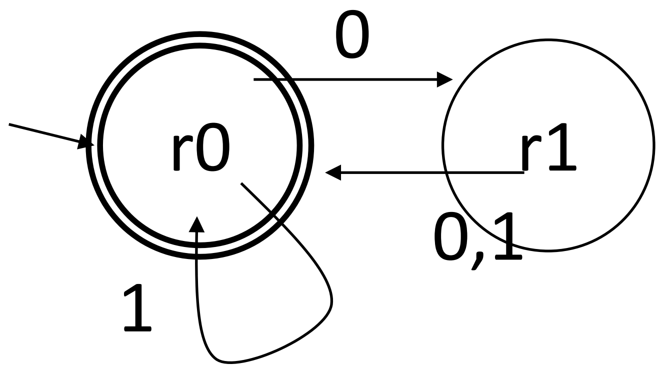
$e =$ 0 1 “” $\emptyset$



Recursive Case: OR

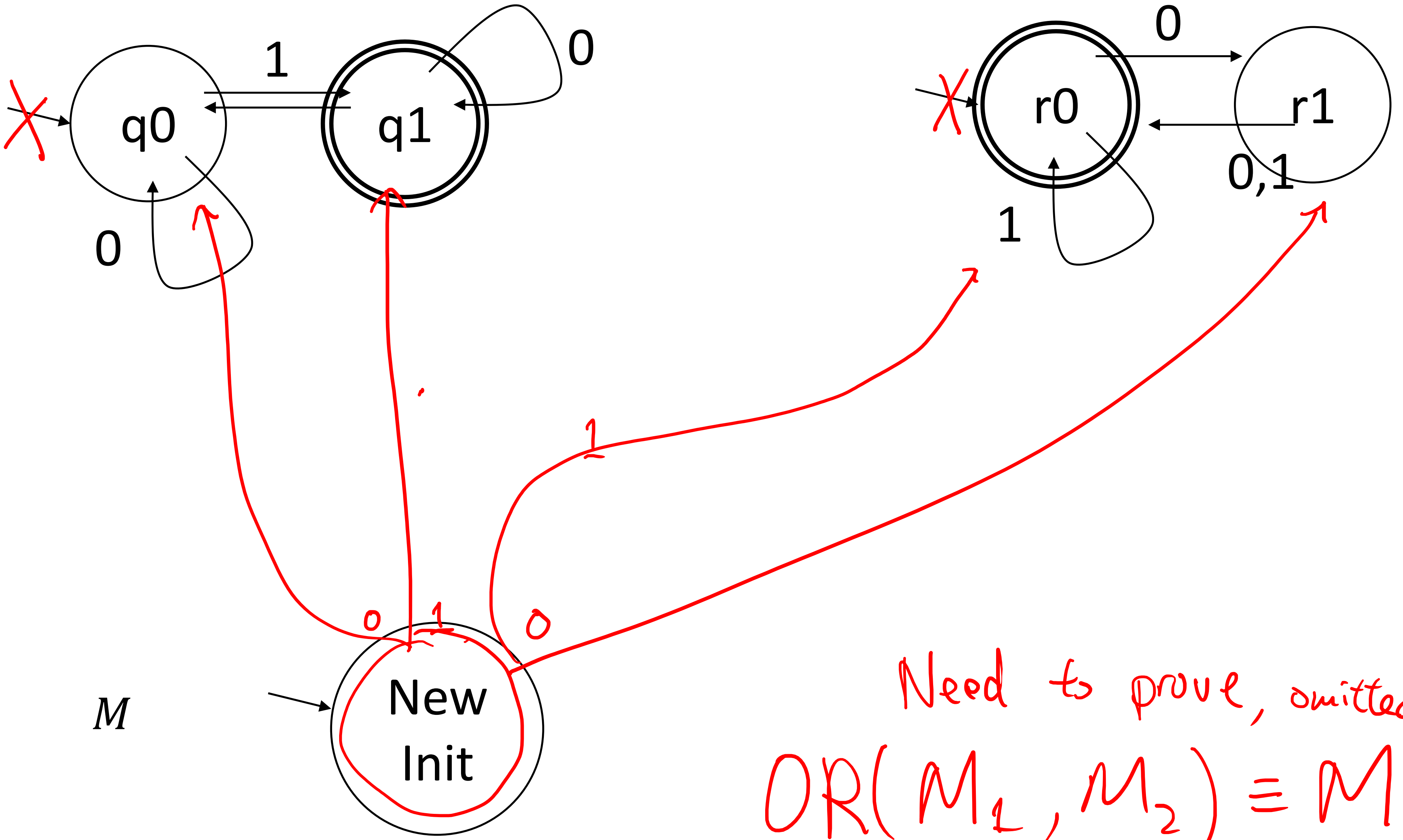
$e = (e_1)|(e_2)$. Suppose we have corresponding NFA M_1 and M_2 for e_1 and e_2.
How to make M for e?





Idea: Add a new init state and new edges

Want: M_1 OR M_2



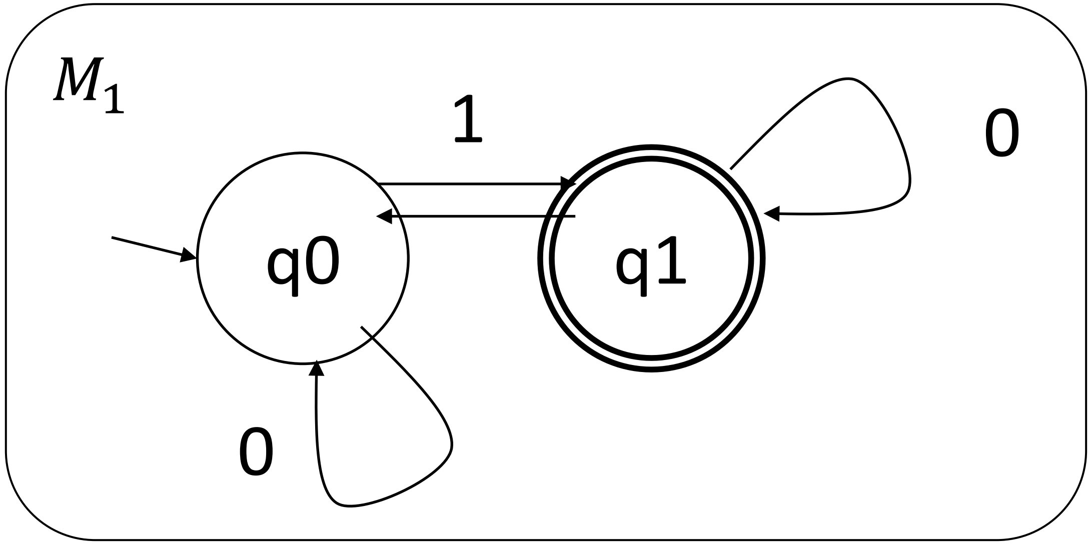
Need to prove, omitted.

$OR(M_1, M_2) = M$

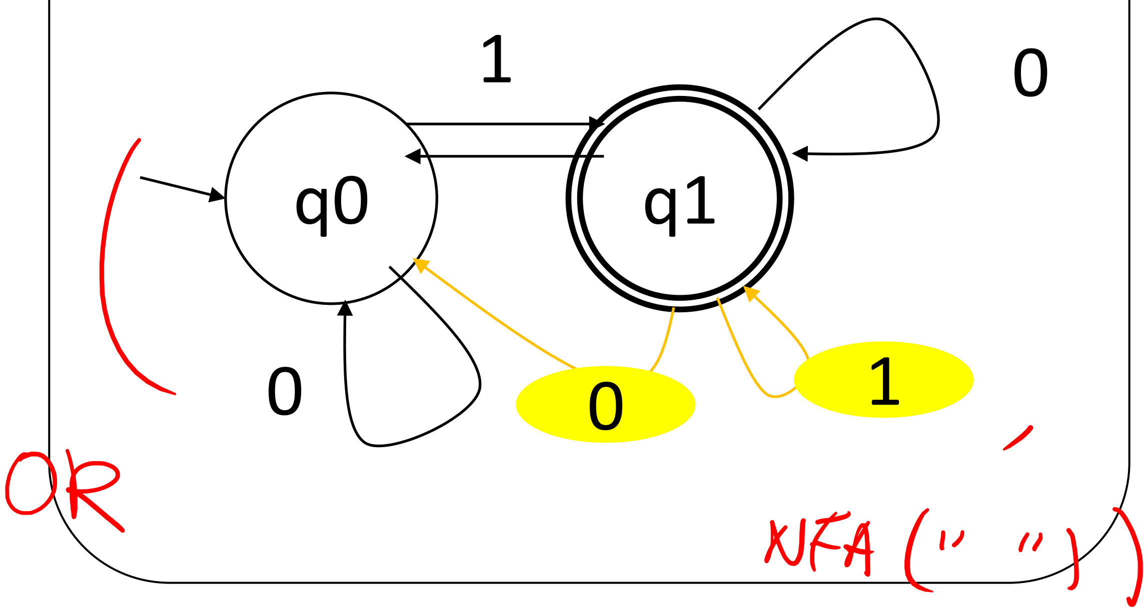
Recursive Case: Kleene Star

X = 1 00 10

M_1 is equivalent to e_1

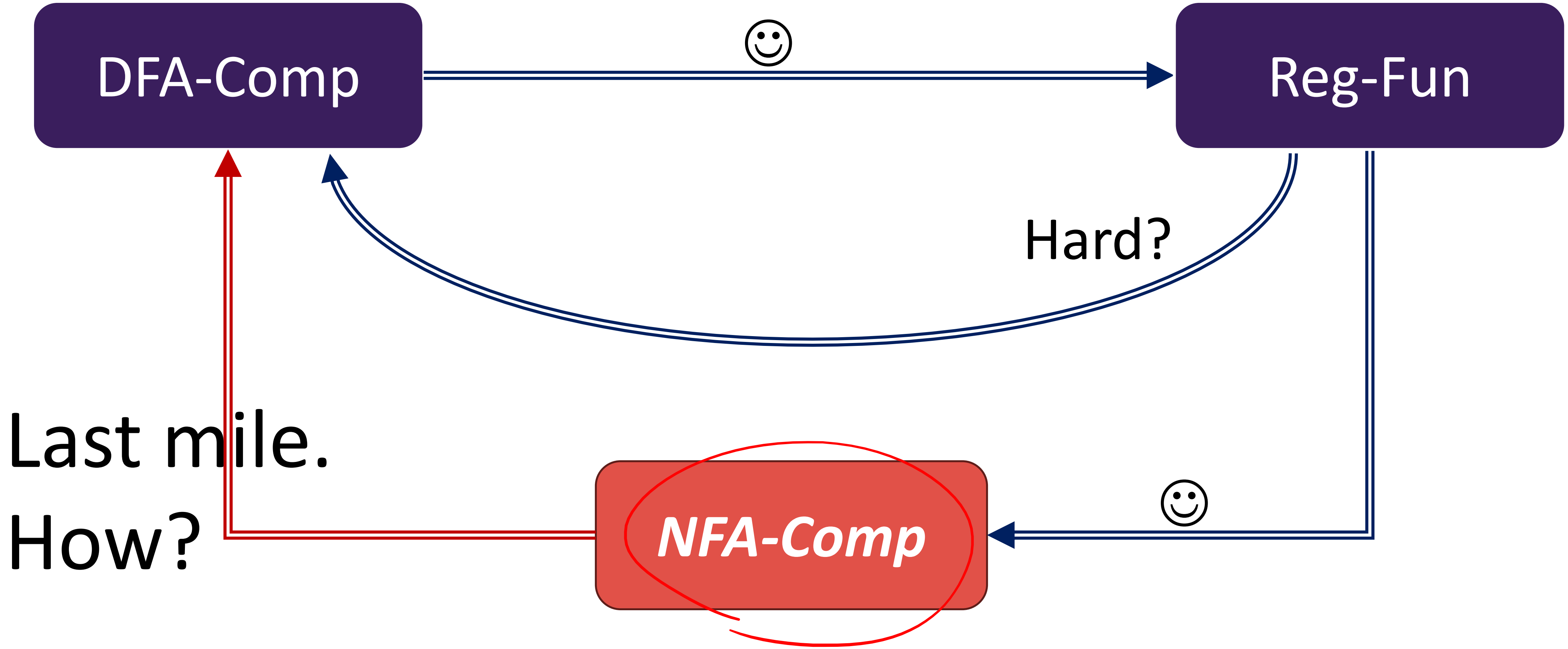


M is equivalent to $e = (e_1)^*$



Idea: In each accept state, "emulate" as the initial state

High-Level Proof Plan



Logistics

- HW 2: Check test cases, resubmit before Feb 11
- Only a few response to “Office Hours” and “Mead Coffee”

HW 3 due this Friday, Feb 13 (10:00pm)

Quiz 4 Coming soon