Photo: [Movie Poster]

Coming soon: PS7

Class 16: Non-deterministic Finite Automata

University of Virginia
cs3120: DMT2
Wei-Kai Lin

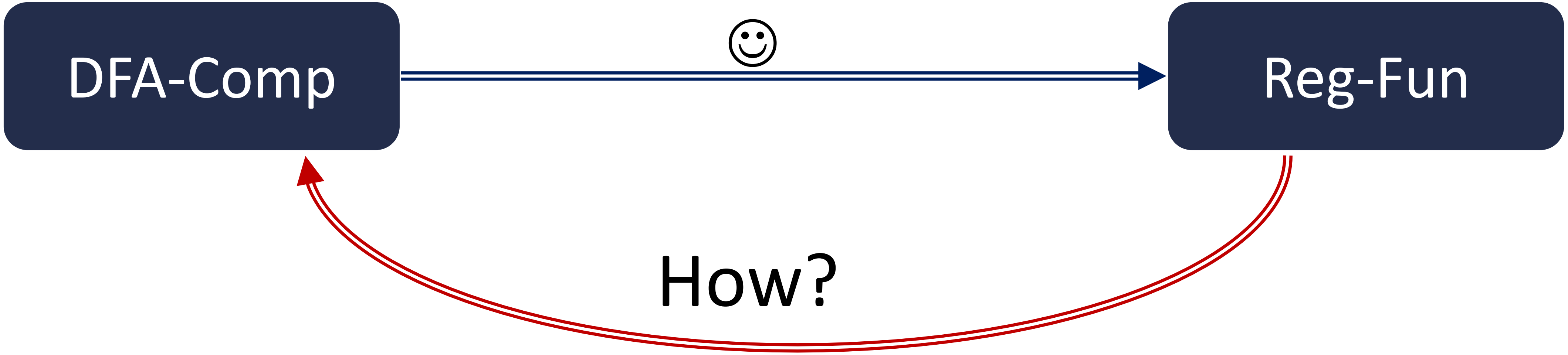
Recall: Proving DFA-Comp = Reg-Fun

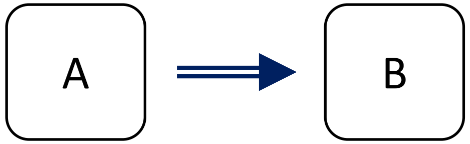
We proved DFA-Comp $\subseteq$ Reg-Fun:
for every DFA M, there is equivalent reg. exp. e

(see TCS Section 6.4.2 for bug-free proof)

Want the other way:
for every reg. exp. e, there is equivalent DFA M

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

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.

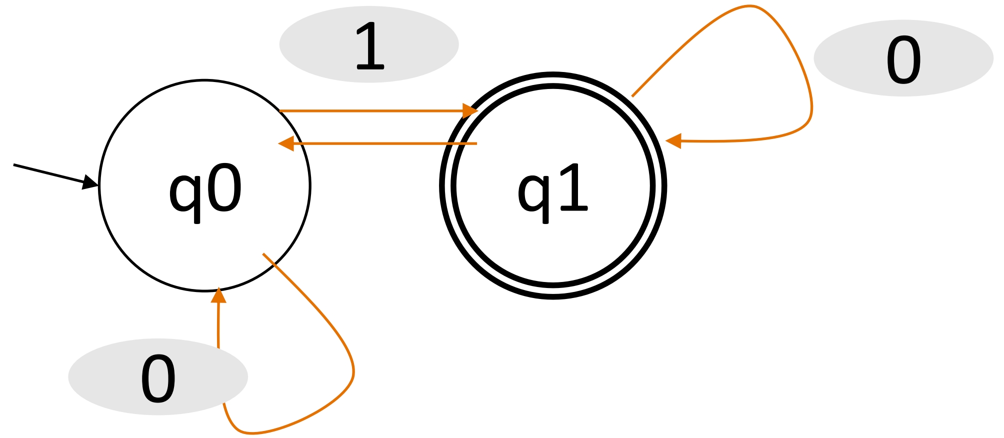
Base cases: easy
Inductive cases: hard

Recall: Kleene Star is Hard

$e = (e_1)^*$

E.g. match $x = 1\ 0\ 0\ 1\ 0$

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

$e = (e_1)^*$

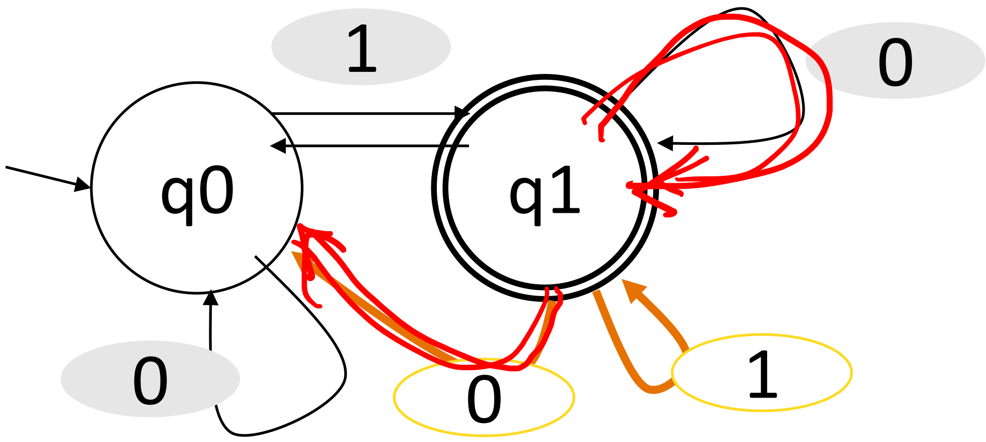
2 choices.

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

1. e_1 e_1 ✓
2. e_1

e_1 ✗

Suppose M_1 is equivalent to e_1



Allow transition to multiple states (clearly, not DFA)

Accept if exist **"a choice" 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

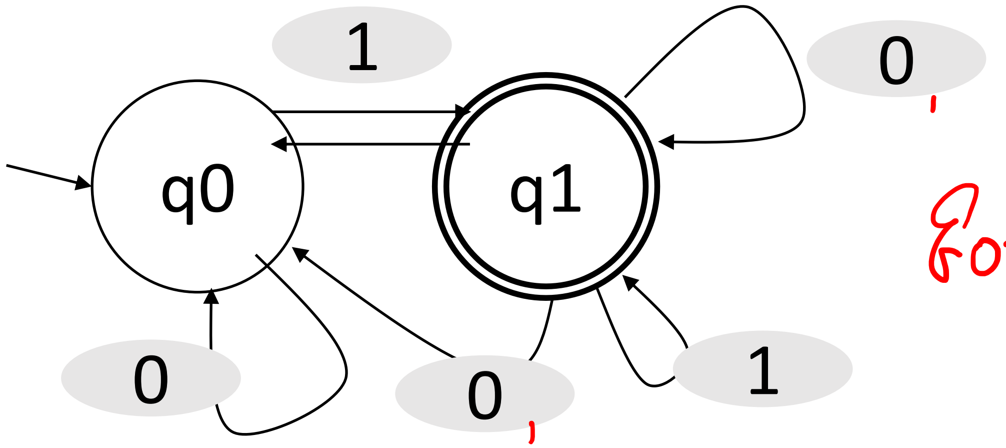
A **Nondeterministic** ***Finite Automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

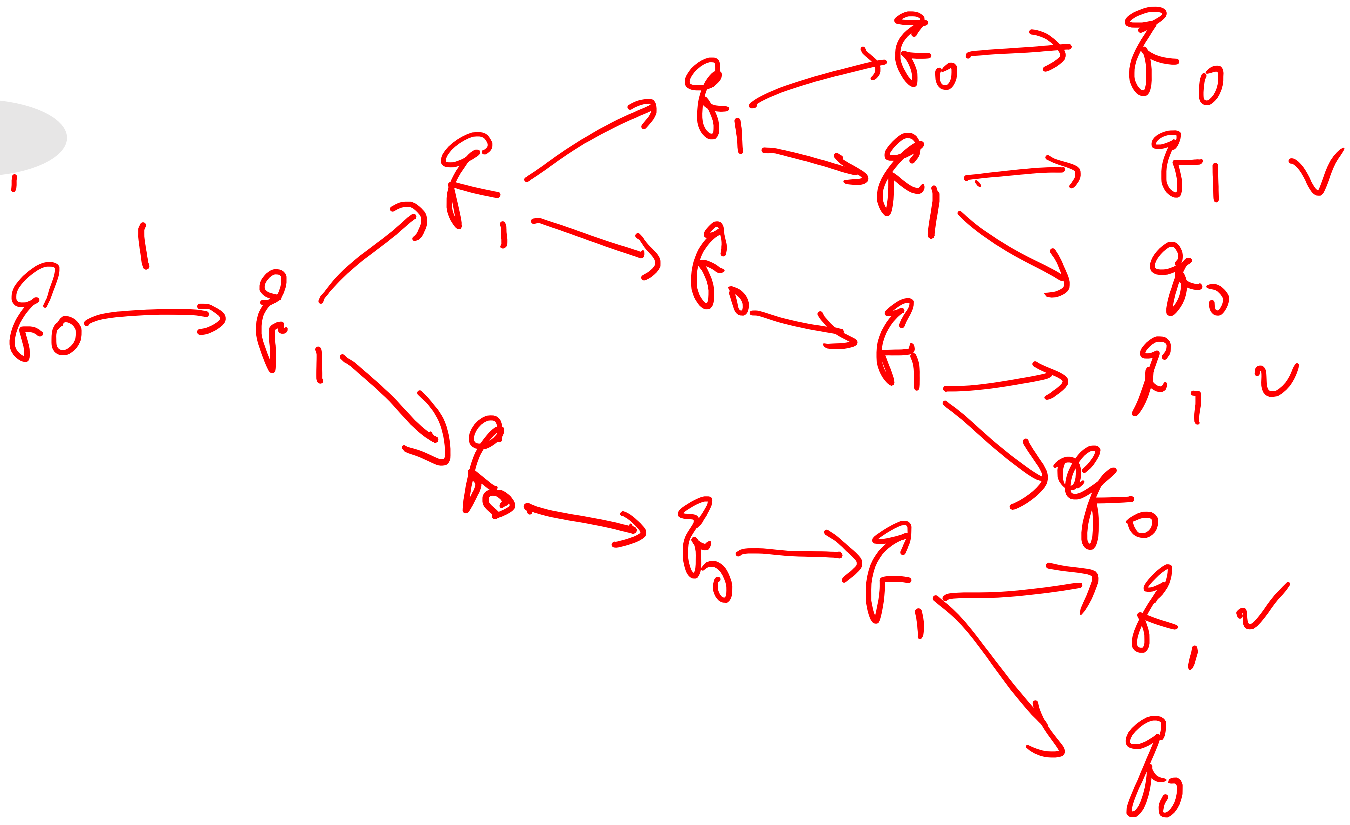
1. Q – a finite set (the *states*)
2. Σ – a finite set (the *alphabet*)
3. $\boldsymbol{\delta: Q \times \Sigma \rightarrow pow(Q)}$
4. $q_0 \in Q$ – the start state
5. $F \subseteq Q$ – the set of accept states

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*** 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. $\delta: Q \times \Sigma \rightarrow pow(Q)$
4. $q_0 \in Q$ – the start state
5. $F \subseteq Q$ – the set of accept states

Recall our DFA model:

The string $x = b_0 b_1 \ldots b_n$ is matched by the DFA $M = (Q, \Sigma, \delta, q_0, F)$ iff there are states $s_0, s_1, s_2, \ldots, s_n \in Q$ such that $\boldsymbol{s_{i+1} = \delta(s_i, b_i)}$ for all $i = 0, \ldots, n-1$ and $s_0 = q_0$ and $s_n \in F$.

Defining the NFA Model

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

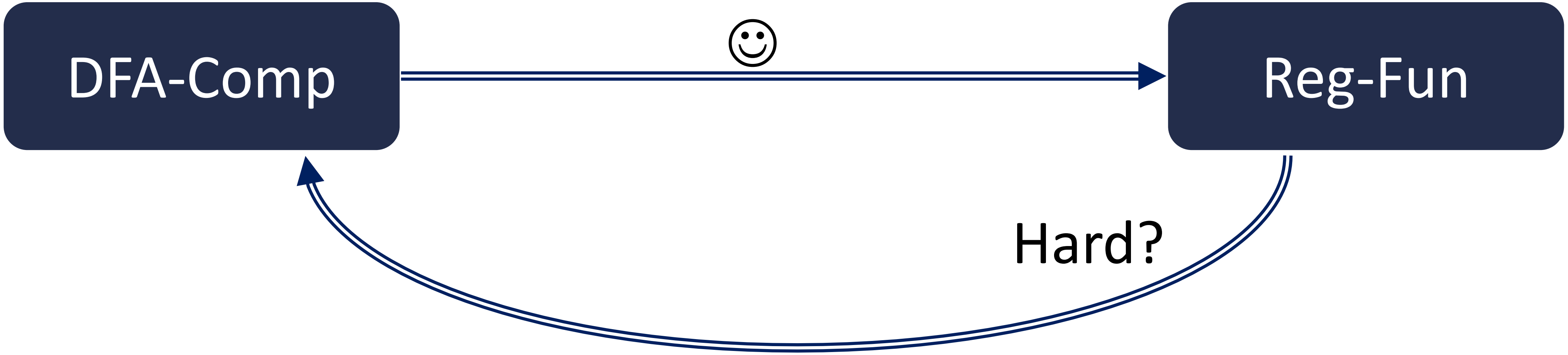
1. Q – a finite set (the *states*)
2. Σ – a finite set (the *alphabet*)
3. $\delta: Q \times \Sigma \rightarrow pow(Q)$
4. $q_0 \in Q$ – the start state
5. $F \subseteq Q$ – the set of accept states

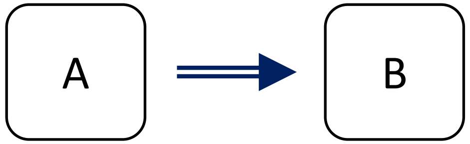
The string $x = b_0 b_1 \dots b_n$ is matched by the NFA $M = (Q, \Sigma, \delta, q_0, F)$ iff there are states $s_0, s_1, s_2, \dots, s_n \in Q$ such that
$\boldsymbol{s_{i+1} \in \delta(s_i, b_i)}$ for all $i = 0, \dots, n-1$ and
$s_0 = q_0$ and
$s_n \in F$.

Recall our DFA model:

The string $x = b_0 b_1 \dots b_n$ is matched by the DFA $M = (Q, \Sigma, \delta, q_0, F)$ iff there are states $s_0, s_1, s_2, \dots, s_n \in Q$ such that $\boldsymbol{s_{i+1} = \delta(s_i, b_i)}$ for all $i = 0, \dots, n-1$ and $s_0 = q_0$ and $s_n \in F$.

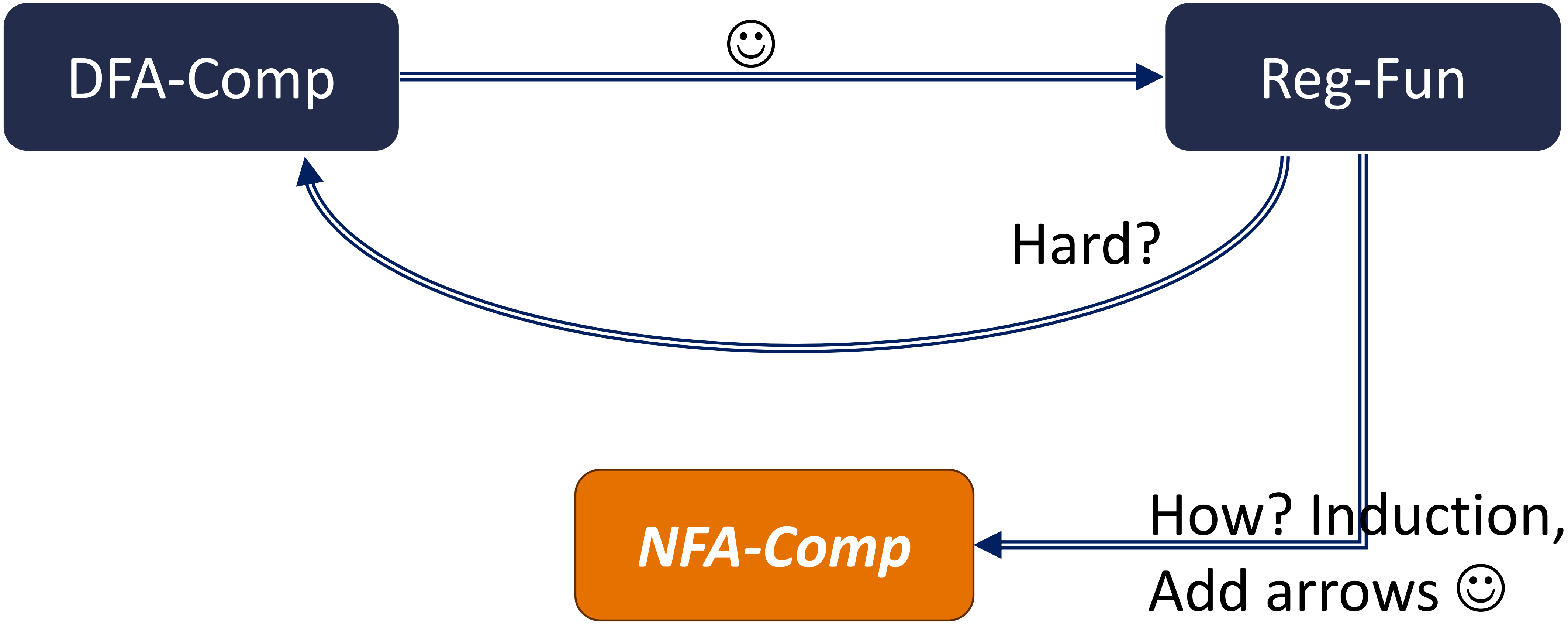
Recalling the High-Level Proof Plan





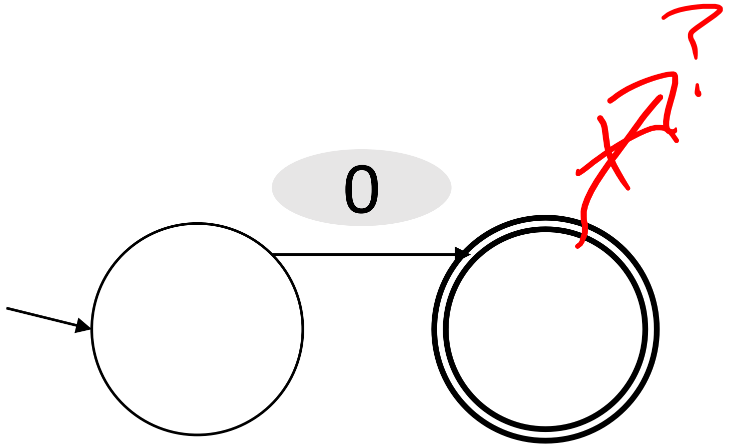
We can convert every $M_A \in A$ to a $M_B \in B$
Such that for all x, M_A accepts x iff M_B accepts x

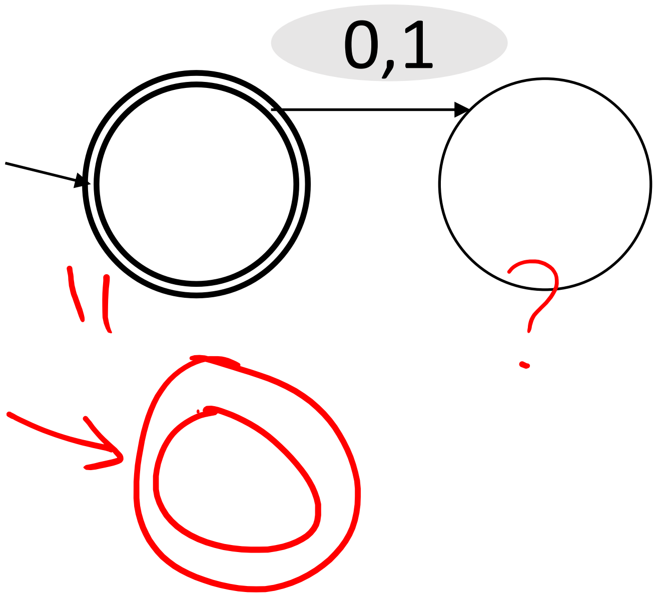
High-Level Proof Plan

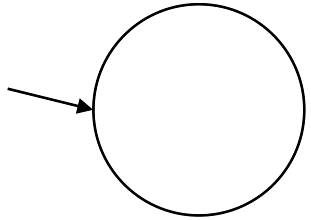


Base Cases are Similar to DFA

$e =$ 0 1 "" $\emptyset$

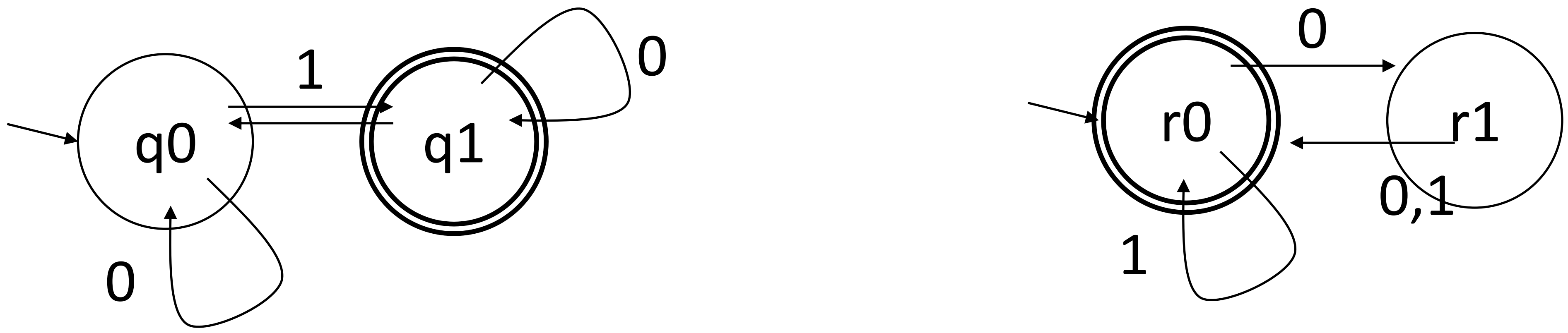






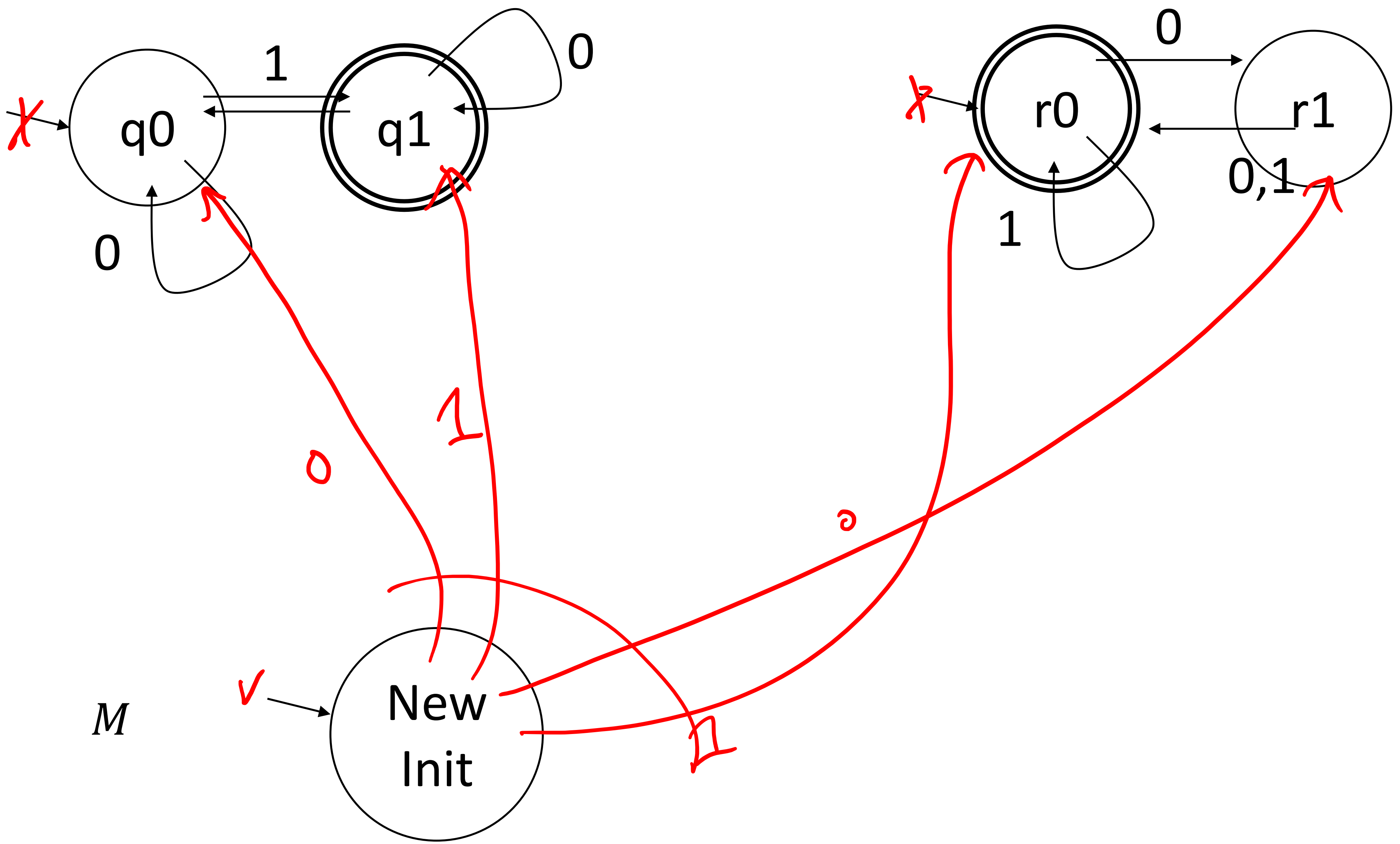
Recursive Case: OR

$e = (e_1)|(e_2)$. 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

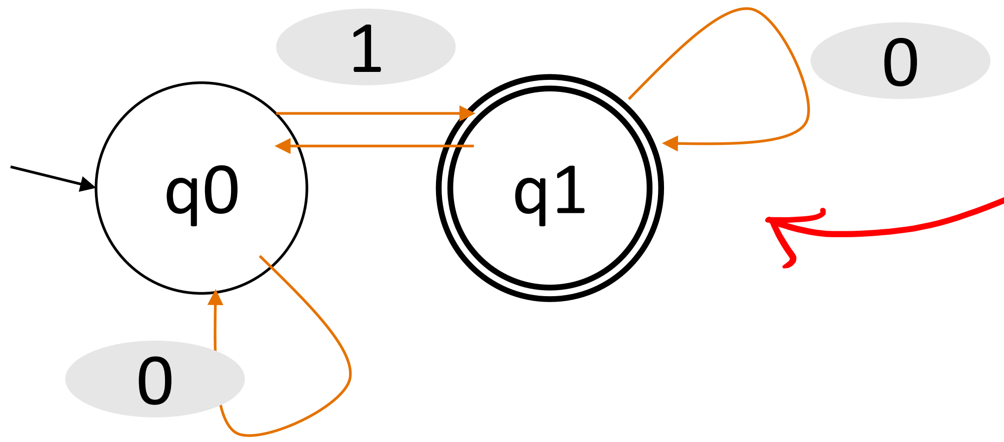
Want: M_1 OR M_2



Recursive Case: Kleene Star

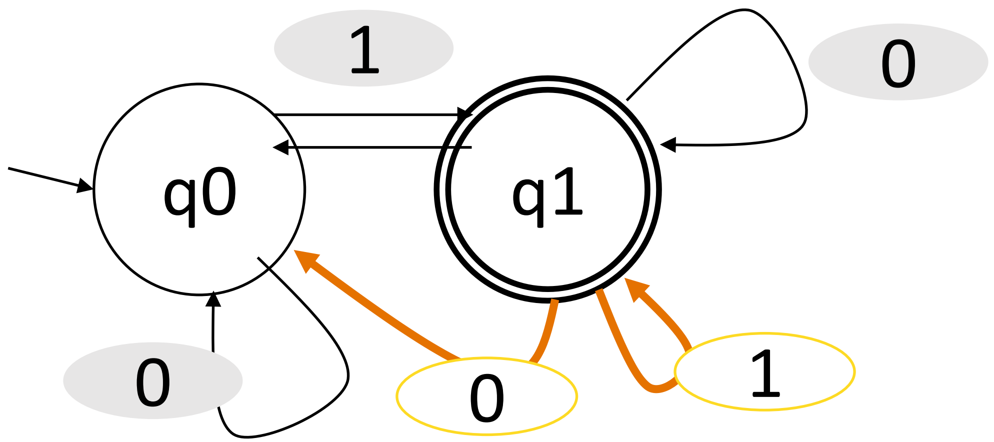
NFA

M_1 is equivalent to e_1

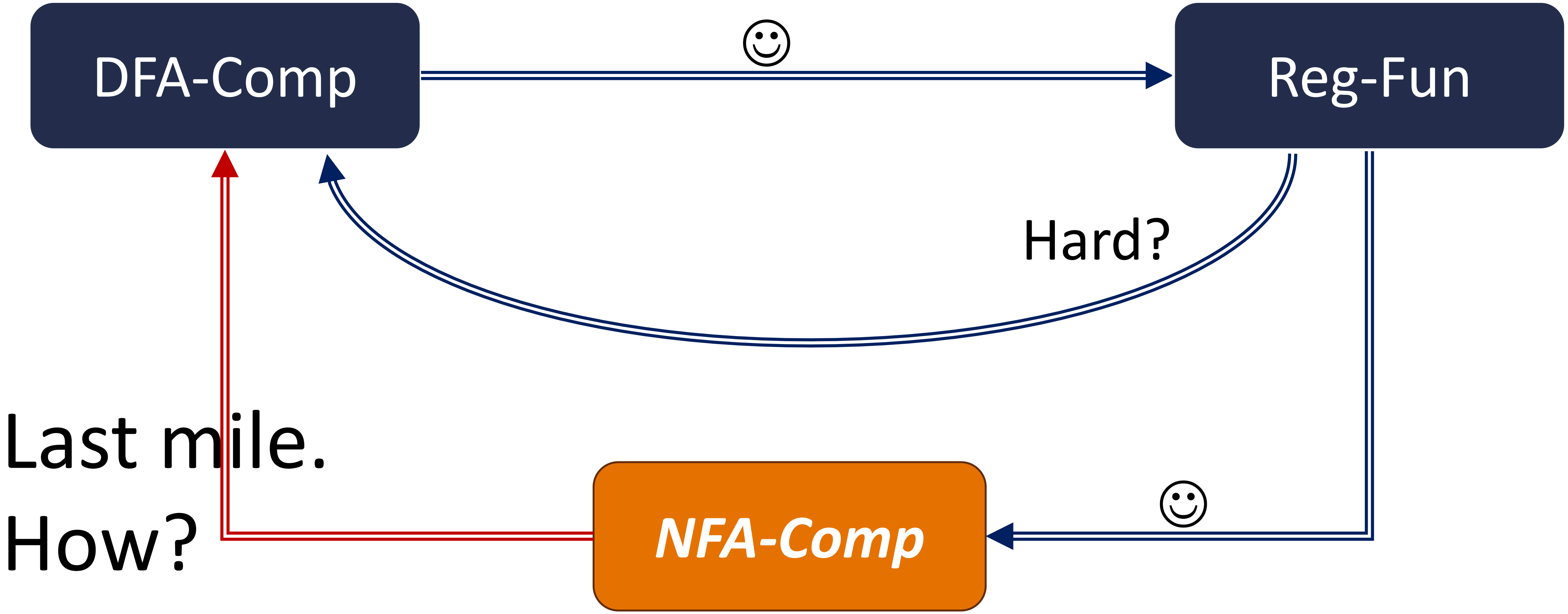


NFA

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



High-Level Proof Plan



Power of NFA/DFA

1. Is there any function a **DFA** can compute that cannot be computed by an **NFA**?

2. Is there any function an **NFA** can compute that cannot be recognized by a **DFA**?

Power of NFA/DFA

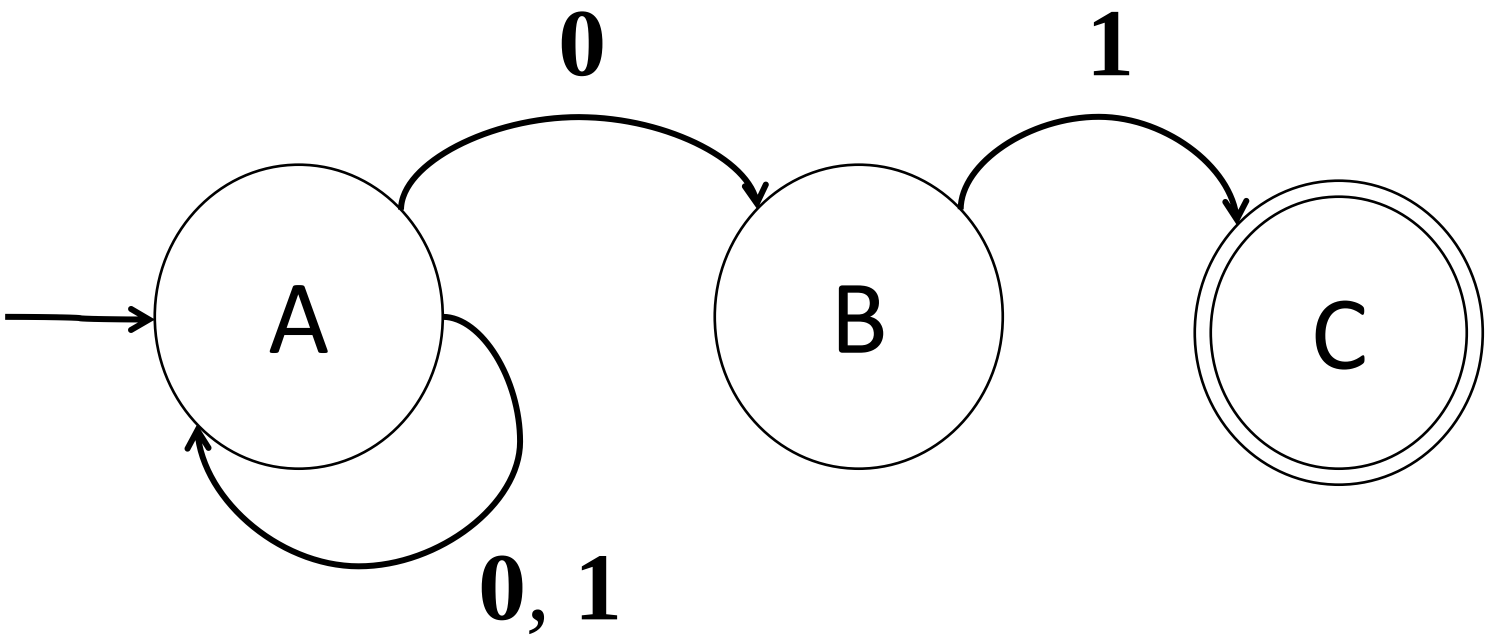
1. Is there any function a **DFA** can compute that cannot be computed by an **NFA**? DFA-Comp ⊆ NFA-Comp

No: NFAs are at least as powerful as DFAs.

2. Is there any function an **NFA** can compute that cannot be recognized by a **DFA**?

NFA–Comp ⊆ DFA–Comp?

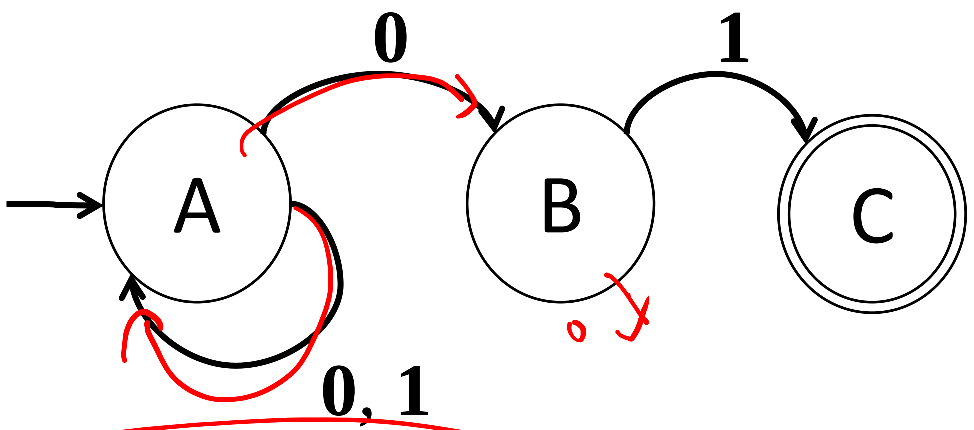
NFA–Comp $\subseteq$ DFA–Comp



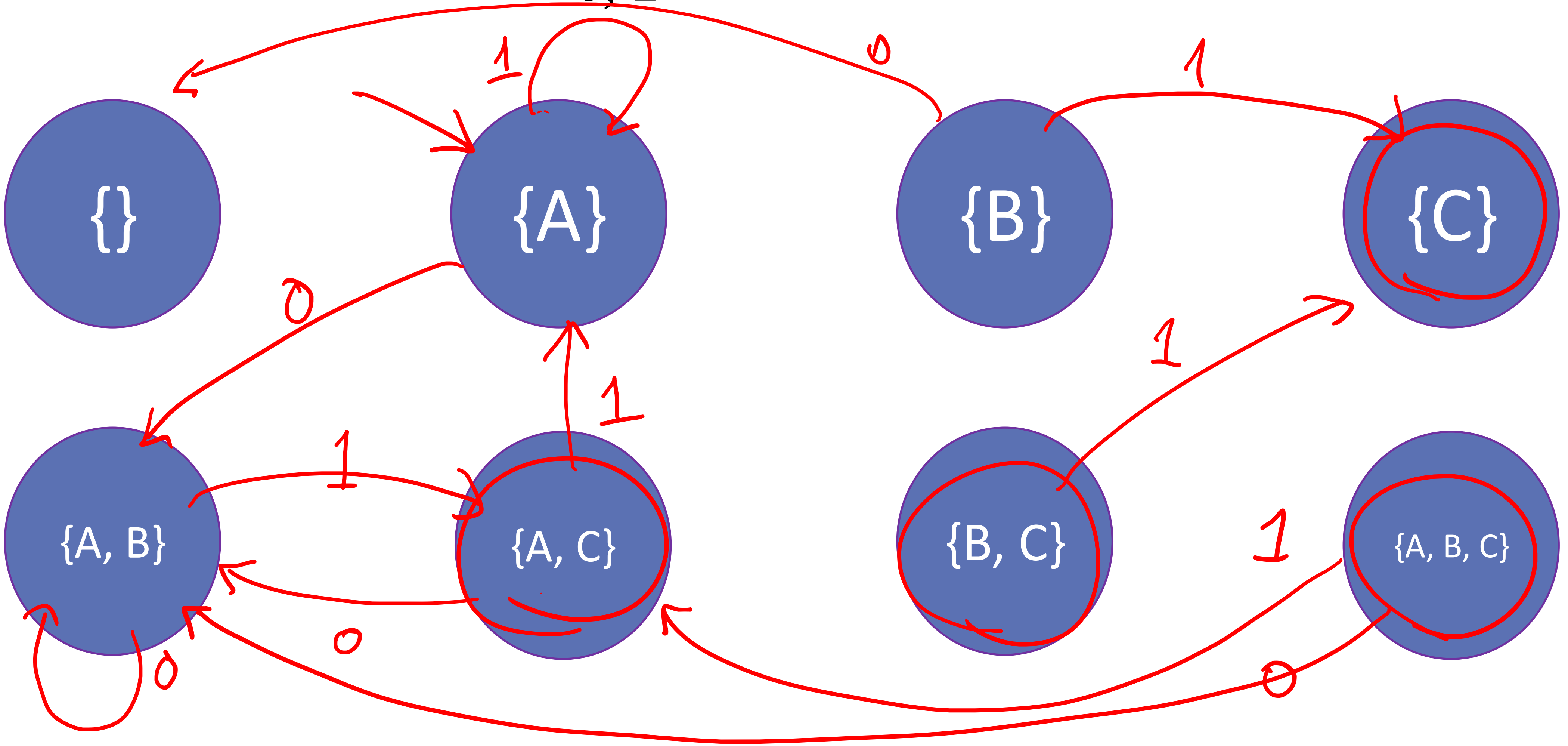
Can we construct a DFA that computes the same function as this NFA?

Idea: the states of DFA is the power set of NFA

NFA



DFA



From NFA to DFA

$Q, \Sigma, q_0, \delta, F$

- States $Q' = pow(Q)$
- Alphabet $\Sigma' = \Sigma$
- Init state $q_0' = \{q_0\}$
- Transition $\delta'(S \subseteq Q, b \in \Sigma') =$ $\bigcup_{a \in S} \delta(a, b)$
- Accept states $F' = \{S \subseteq Q:$ $\exists a \in F$ s.t $a \in S$ $\}$

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

$S \in Q'$

$S = \{A, B\}$

$\delta(A, b) \cup \delta(B, b)$

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. $\delta: Q \times \Sigma \to pow(Q)$
4. $q_0 \in Q$ – the start state
5. $F \subseteq Q$ – the set of accept states

Charge

Nondeterministic Finite Automata

From regular expressions to NFA

From NFA to DFA

Coming soon: PS7, PRR8