Equivalence of DFAs, NFAs & Regular Expressions

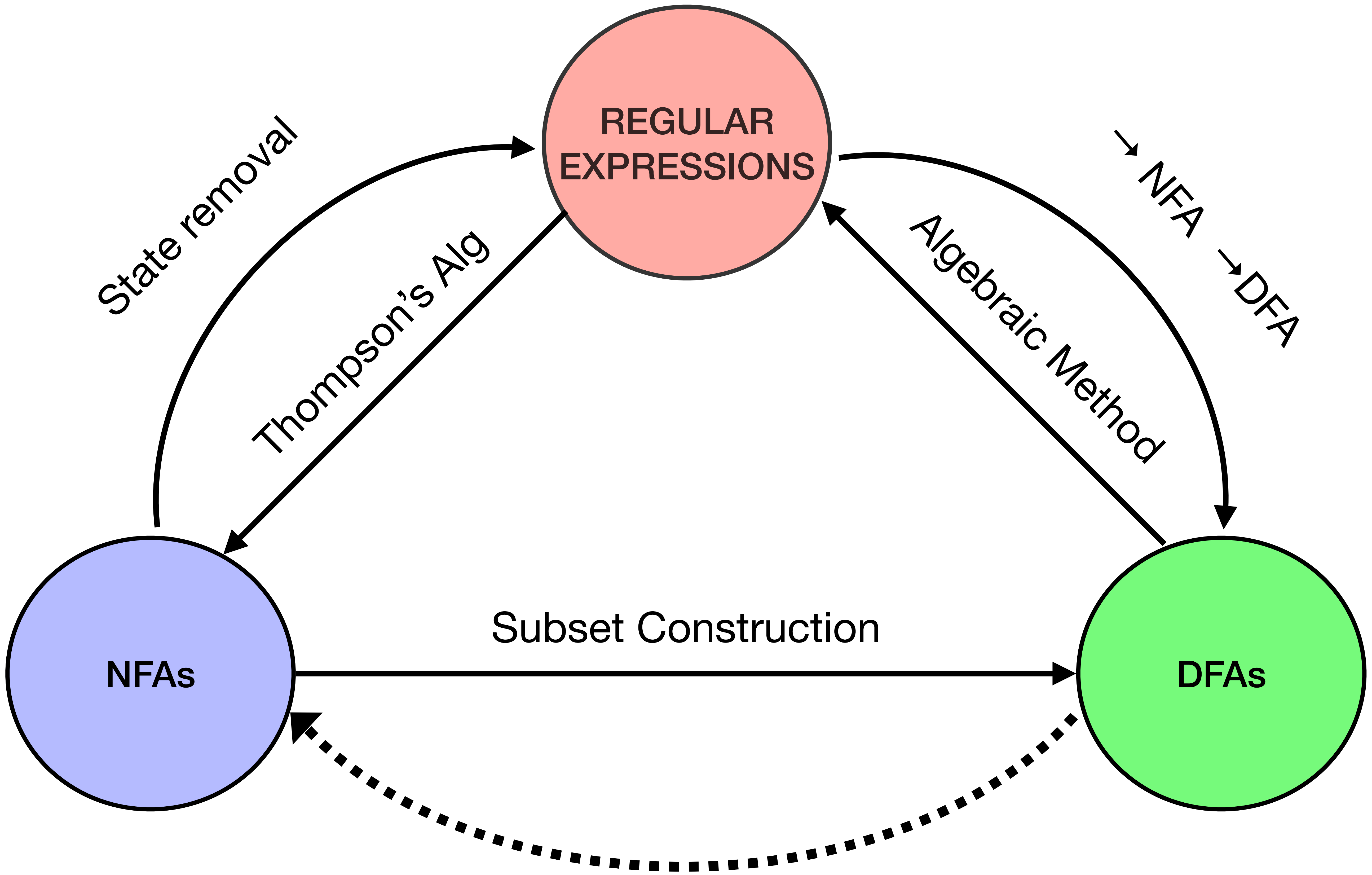
Sides based on material by Profs. Kani, Erickson, Chekuri, et. al.

All mistakes are my own! - Ivan Abraham (Fall 2024)

Image by ChatGPT (probably collaborated with DALL-E)

Goal of lecture

- The point of this lecture is to establish that we gain no additional computational chops by choosing one of DFA/NFA/RegEx (Regular Expressions) over the other.
- They all represent the same class of language - *regular languages.*



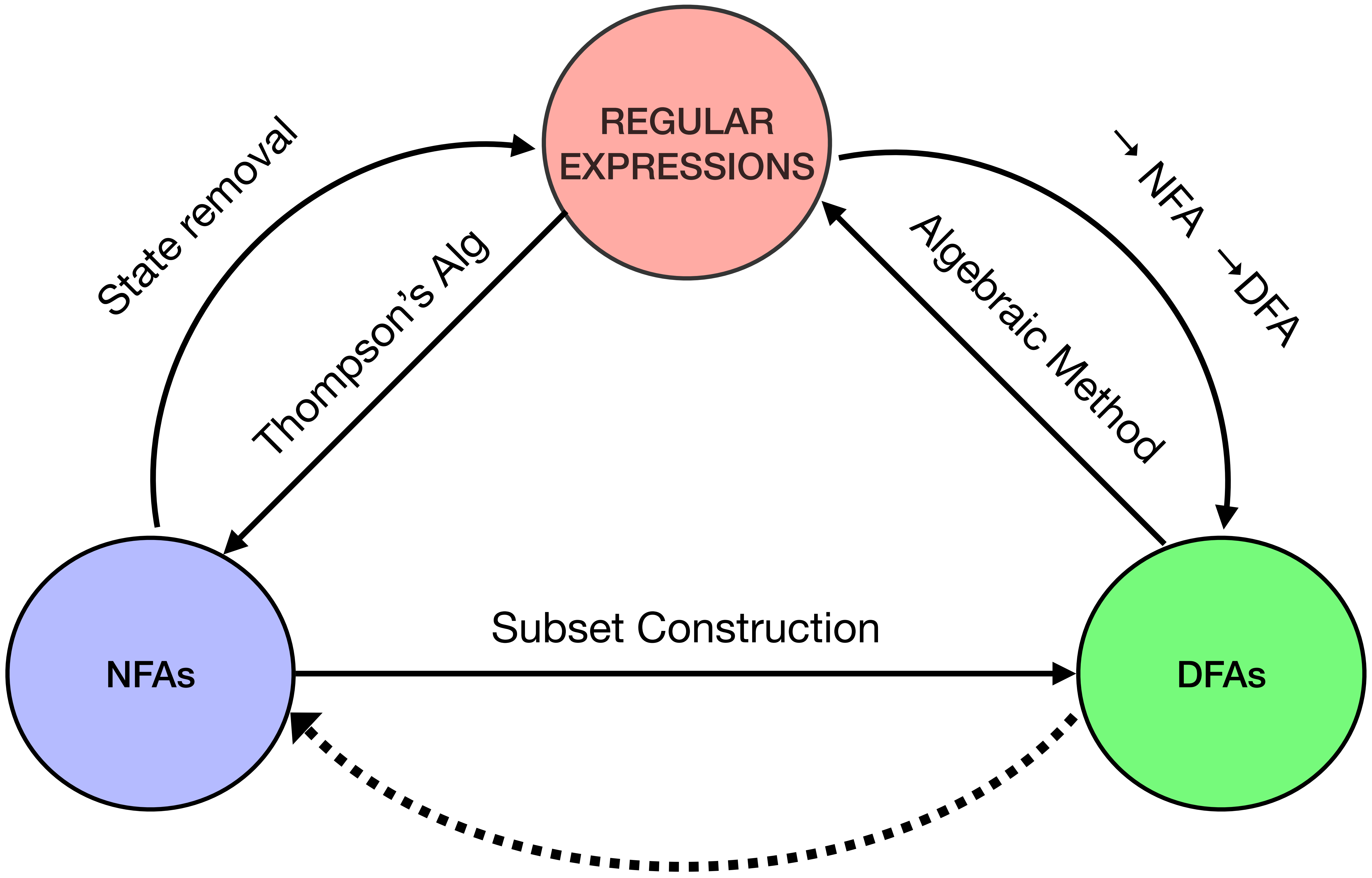
Source: Kani Archive

A language L can be described by a regular expression if and only if L is the language accepted by a DFA.

Kleene's Theorem ~ 1951

Outline of lecture

- Each of the arrows in the figure on the right could be *formally* proved … but
  - We will only look at the *Subset Construction* formally.
  - For the remaining, we will “prove by example.”



Source: Kani Archive

Equivalence of DFAs and NFAs

Formal definitions

Deterministic Finite Automaton

Recall that the formal definition of a DFA is as follows. A DFA is a 5-tuple

$$M = \left(Q, \Sigma, \delta, q_0, F\right)$$

where

- Q is a finite set of *states*,
- Σ is a finite set of tokens/characters called the *alphabet*,
- $\delta : Q \times \Sigma \to Q$ is a *transition function* that encodes state changes when a token from the alphabet is consumed,
- $q_0 \in Q$ is a single distinguished state called the *start state*,
- $F \subseteq Q$ is a set of distinguished states called the *accept* or *final states*.

Formal definitions

Nondeterministic Finite Automaton

Recall that the formal definition of an NFA is as follows. A NFA is a 5-tuple

$$N = \left(Q, \Sigma, \delta, q_0, F\right)$$

where

- Q is a finite set of *states*,
- Σ is a finite set of tokens/characters called the *alphabet*,
- $\delta : Q \times \Sigma \cup \varepsilon \to 2^Q$ is a *transition rule* that encodes state changes when a token from the alphabet is consumed,
- $q_0 \in Q$ is a single distinguished state called the *start state*,
- $F \subseteq Q$ is a set of distinguished states called the *accept* or *final states*.

Equivalence of NFAs and DFAs

Key difference

- NFAs we have introduced allow spontaneous transitions (called ε-transitions)
- NFAs could be in multiple states simultaneously
- NFAs need not spell out every transition

- Therefore, an NFA without any ε-transitions and such that

$$|\delta(q, \sigma)| \leq 1$$

$$\delta(q, \sigma) \neq \emptyset$$

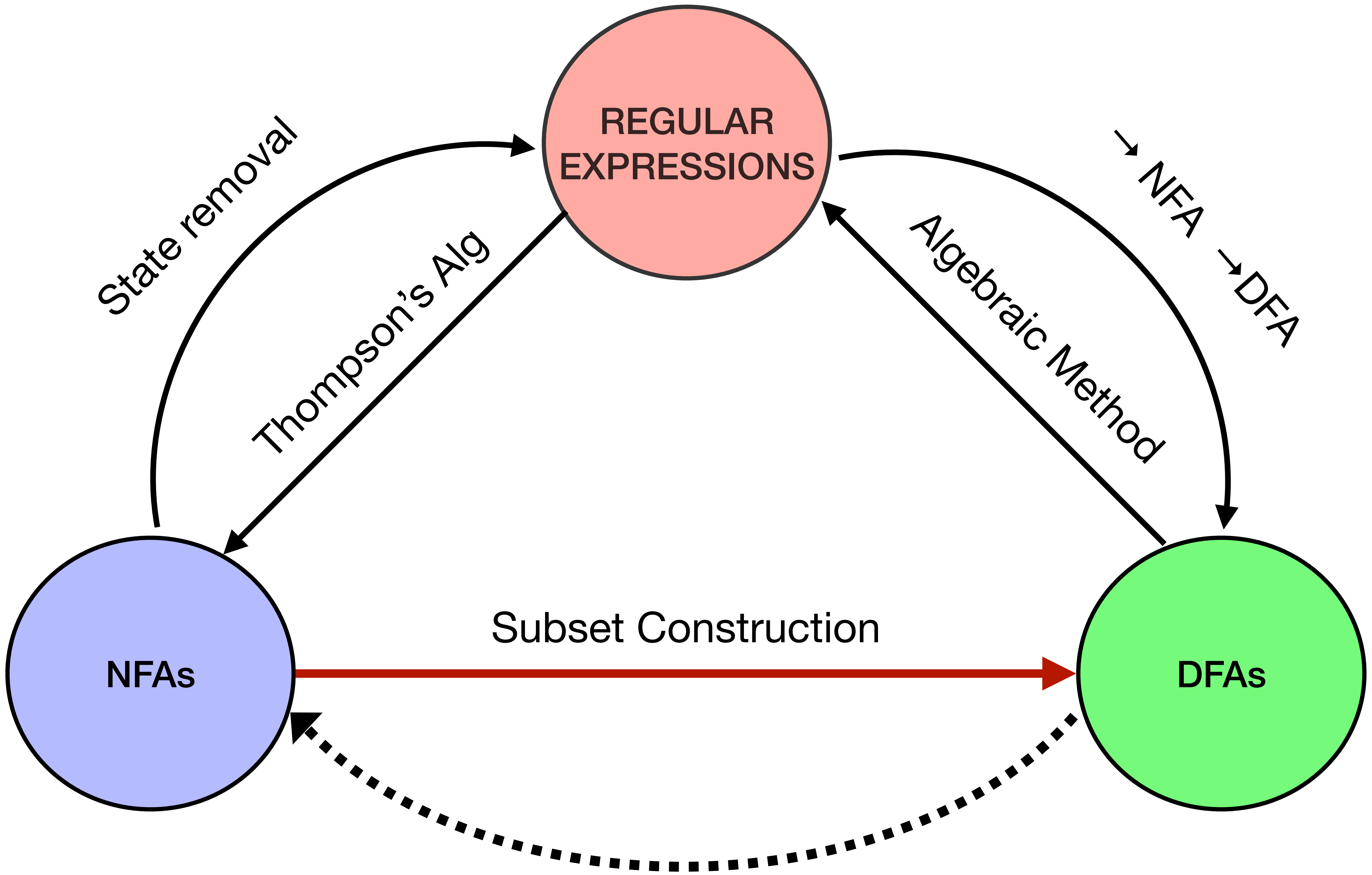
for all $q \in Q, \sigma \in \Sigma$ is a DFA

- In other words, all DFAs are NFAs

Equivalence of NFAs and DFAs

- Thus, we only need to show that for every NFA N, there exists an equivalent DFA M
  - What does it mean for two finite automata to be *equivalent*?
  - Given N, need to show can construct M such that

$$L(M) = L(N)$$



Source: Kani Archive

Equivalence of NFAs and DFAs

Extended transition functions

- For a DFA M we can say M accepts a string w if $\hat{\delta}(q_0, w) \in F$ where $\hat{\delta}_M : Q \times \Sigma^* \to Q$ is the extended transition function defined recursively
  - $\hat{\delta}_M(q, w) = q$ if $w \in \varepsilon$
  - $\hat{\delta}_M(q, w) = \hat{\delta}_M\left(\delta(q, a), x\right)$ if $w = ax$ for some $a \in \Sigma$ and $x \in \Sigma^*$
- What should the extended transition rule for an NFA be?
  - Need to be able to handle those spontaneous ε-transitions

Equivalence of NFAs and DFAs

Extended transition functions

- Define $E(q)$ to be the ε-reach of $q \in Q$. That is, let $E(q)$ be the set of states reachable from q by following zero or more ε arrows.

- We will also allow E to act on a set R:

$$E(R) := \bigcup_{r \in R} E(r)$$

- Then, the extended transition rule $\hat{\delta}_N$ for an NFA can be defined recursively:

$$\hat{\delta}_N\left(q, w\right) = E(q) \quad \text{if } w = \varepsilon$$

$$\hat{\delta}_N\left(q, w\right) = \bigcup_{p \in \hat{\delta}_N\left(q, x\right)} E(\delta(p, a)) \quad \text{if } w = xa \text{ where } a \in \Sigma$$

Equivalence of NFAs and DFAs

Subset construction method

- Now we can say a DFA M and NFA N are equivalent if their extended transitions $\hat{\delta}_M$ and $\hat{\delta}_N$ agree on all words w.
- Given, $N = \left(Q, \Sigma, \delta, q_0, F\right)$ let us try to construct a $M = \left(Q', \Sigma', \delta', q_0', F'\right)$ such that $L(M) = L(N)$.
  - Since they must recognize the same language, $\Sigma' = \Sigma$.
  - Next, an NFA can be in multiple states at once. At each instance, these various states will always be a subset of Q. Thus, we can set $Q' = 2^Q$.

Equivalence of NFAs and DFAs

Subset construction method

- Next, we must define the transition rule for M incorporating those ε-transitions of N.
- From any state R in M (which, remember, is a set of states), if we consume a token a, we need to follow any edges labeled a, and then we need to take any ε-transitions from there. Thus we get:

$$\delta'(R, a) := \bigcup_{q \in R} E\left(\delta(q, a)\right)$$

Equivalence of NFAs and DFAs

Subset construction method

- Finally, it remains to specify the start and accept states q_0' and F' respectively.
- From the start state, we immediately follow all ε-transitions. So set

$$q_0' = E\left(q_0\right)$$

- The final states of M should be the collection of states of N that are final states.

$$F' = \left\{R \in Q' \mid R \cap F \neq \varnothing\right\}$$

Equivalence of NFAs and DFAs

Subset construction method

- That completes the specification of a DFA M mimicking the functioning of an NFA N.

- Is the proof complete?

    - One way to finish the proof is to show $\hat{\delta}_N(q_0, w) = \hat{\delta}_M(q_0', w)$ for ***all*** $w \in \Sigma^*$

    - It can be done using induction on $|w|$ and fair bit of definition chasing.

Example - subset construction

We write software to automate tasks …

…. loops, subroutines and functions to avoid repetition and tedium …

… so why reinvent the wheel?

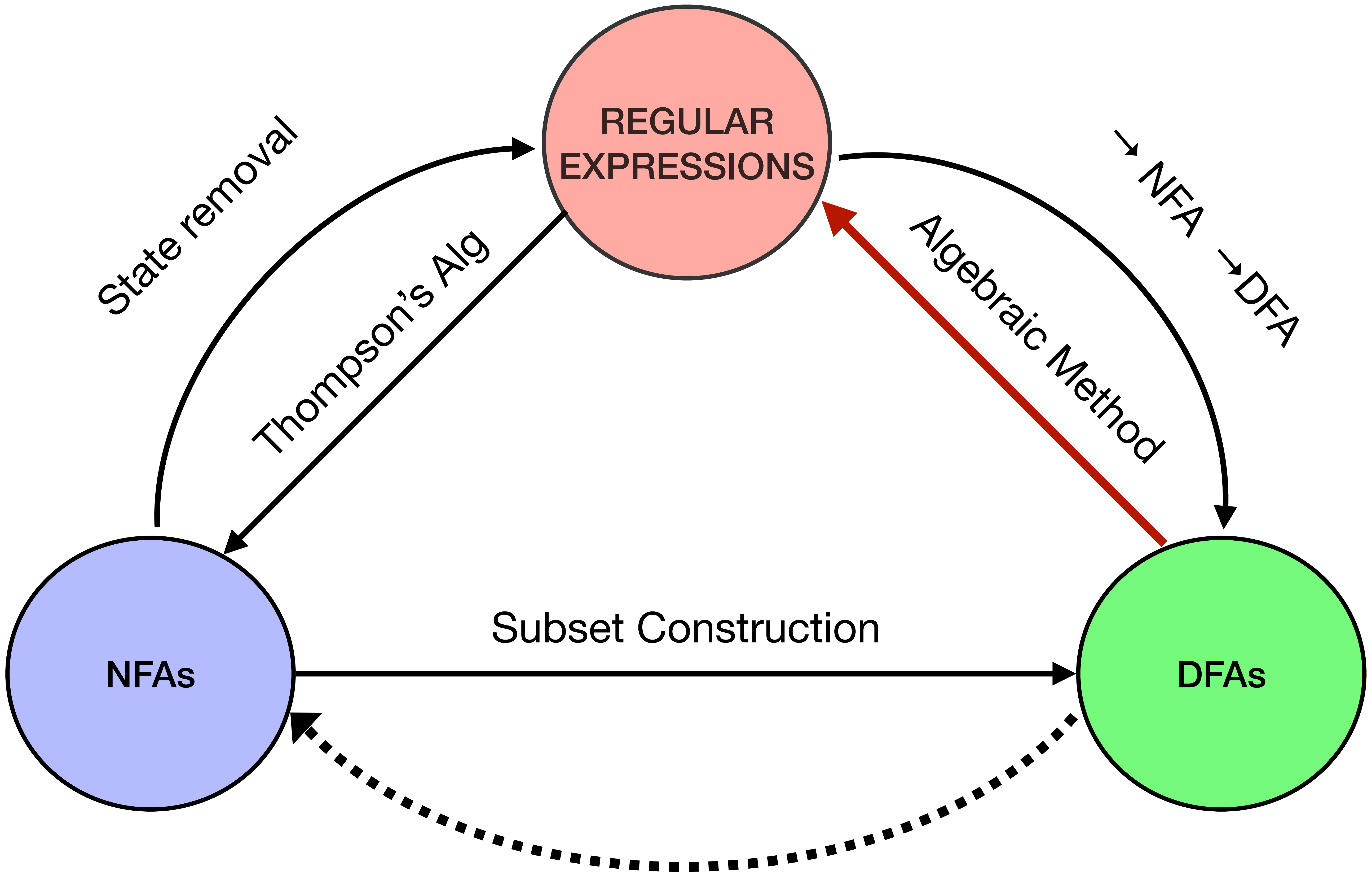
Standford's CS 103 Notes: Guide to the Subset Construction

Equivalence of DFAs and Regular Expressions

Converting a DFA to Regular Expression

Algebraic method

- Next, let us look at how one might construct a **regular expression out of a DFA**:
  - Highlighted red arrow in diagram
- Called *algebraic* because we end up solving a system of equations



Source: Kani Archive

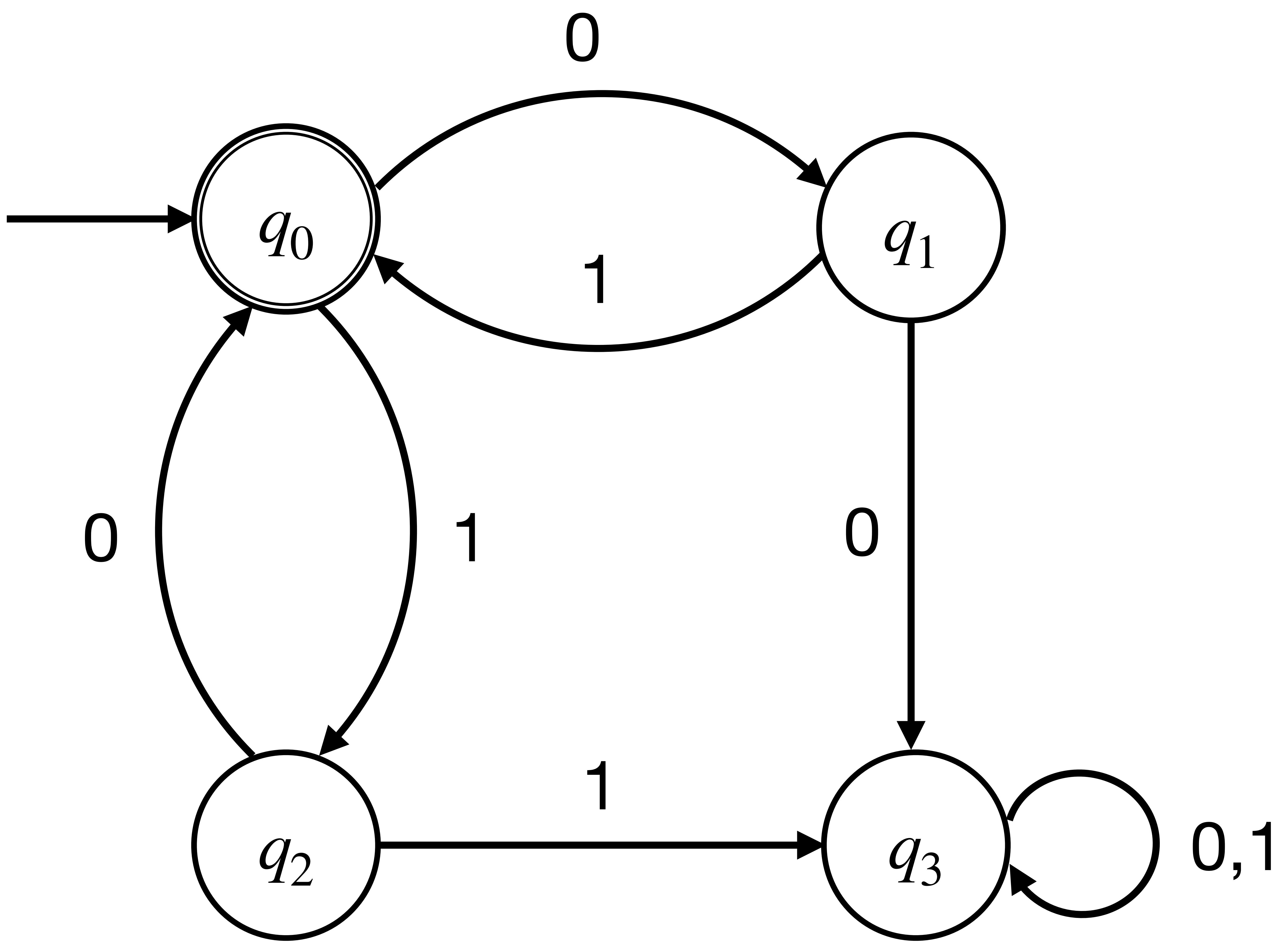
Converting a DFA to Regular Expression

Algebraic method - Example

Key point: We can write a transition to a state as a juxtaposition of the prior state with the consumed token.

Example: The transition to q_1 can be written as

$$q_1 = q_0 \cdot 0$$

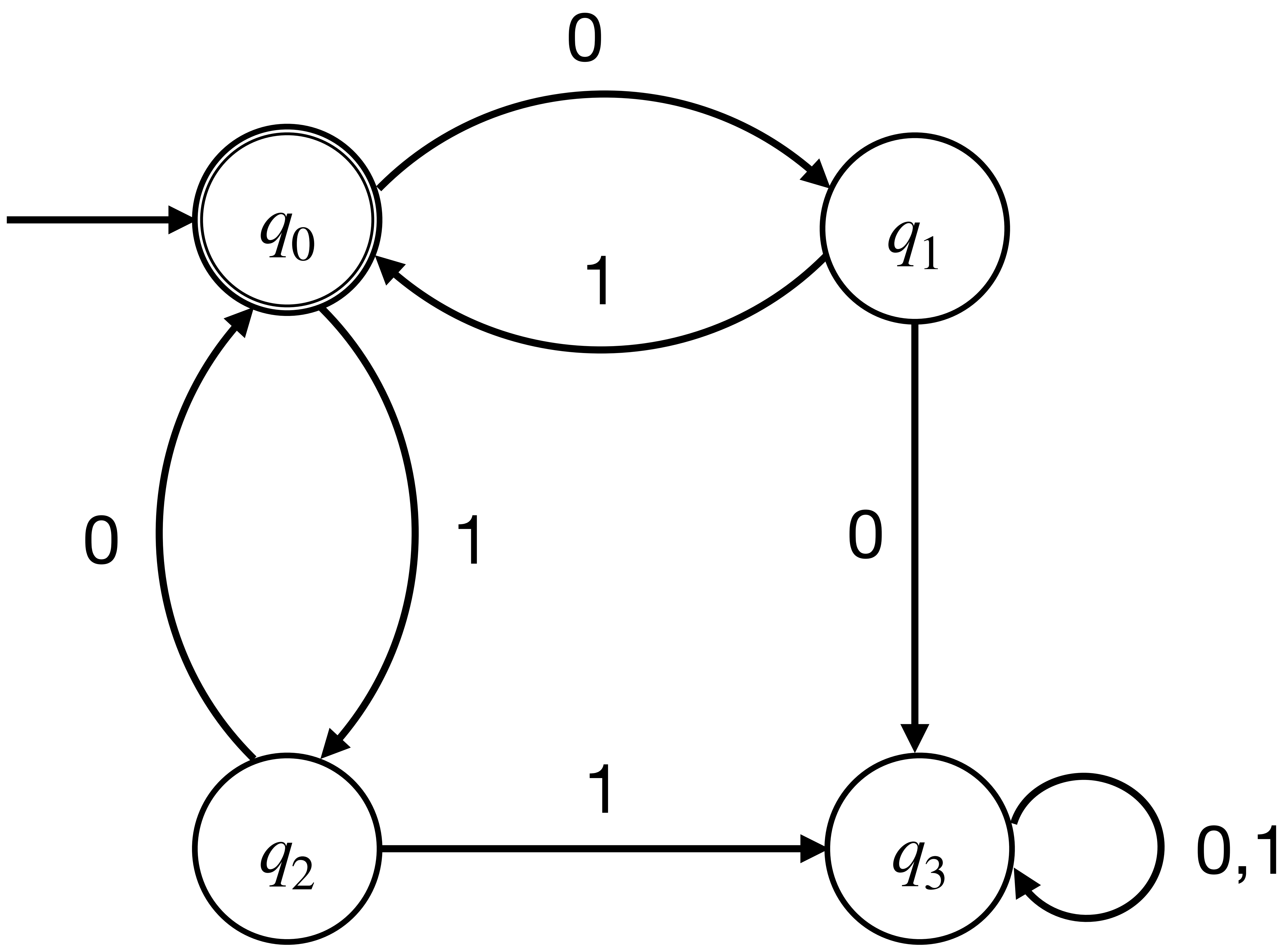


Converting a DFA to Regular Expression

Algebraic method - Example

- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_1 = q_0 0$
- $q_2 = q_0 1$
- $q_3 = q_1 0 + q_2 1 + q_3 (0 + 1)$

Now we simple solve the system of equations for q_0 (accept state)



Converting a DFA to Regular Expression

Algebraic method - Example

- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_1 = q_0 0$
- $q_2 = q_0 1$
- $q_3 = q_1 0 + q_2 1 + q_3 (0 + 1)$

- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_0 = \epsilon + q_0 01 + q_0 10$
- $q_0 = \epsilon + q_0 (01 + 10)$

Apply **Arden's Lemma**

$$R = Q + RP = QP^*$$

Arden's lemma

Proof sketch

- Show that $R = Q + RP = QP*$
- Start with $R = Q + RP$ and repeatedly plug-in the definition recursively
  - Once: $R = Q + \left(Q + RP\right) P$
  - Twice: $R = Q + \left(Q + \left(Q + RP\right) P\right) P$
  - … $R = Q \left(\varepsilon + P + P^2 + P^3 + \dots\right)$

Converting a DFA to Regular Expression

Algebraic method - Example

- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_1 = q_0 0$
- $q_2 = q_0 1$
- $q_3 = q_1 0 + q_2 1 + q_3 (0 + 1)$

$$R = Q + RP = QP^*$$

- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_0 = \epsilon + q_0 01 + q_0 10$
- $q_0 = \epsilon + q_0 (01 + 10)$

Apply **Arden's Lemma**

$$q_0 = \epsilon + q_0 (01 + 10)$$

$$q_0 = \epsilon (01 + 10)^* = (01 + 10)^*$$

Equivalence of NFAs and Regular Expressions - State removal

Converting a DFA to Regular Expression

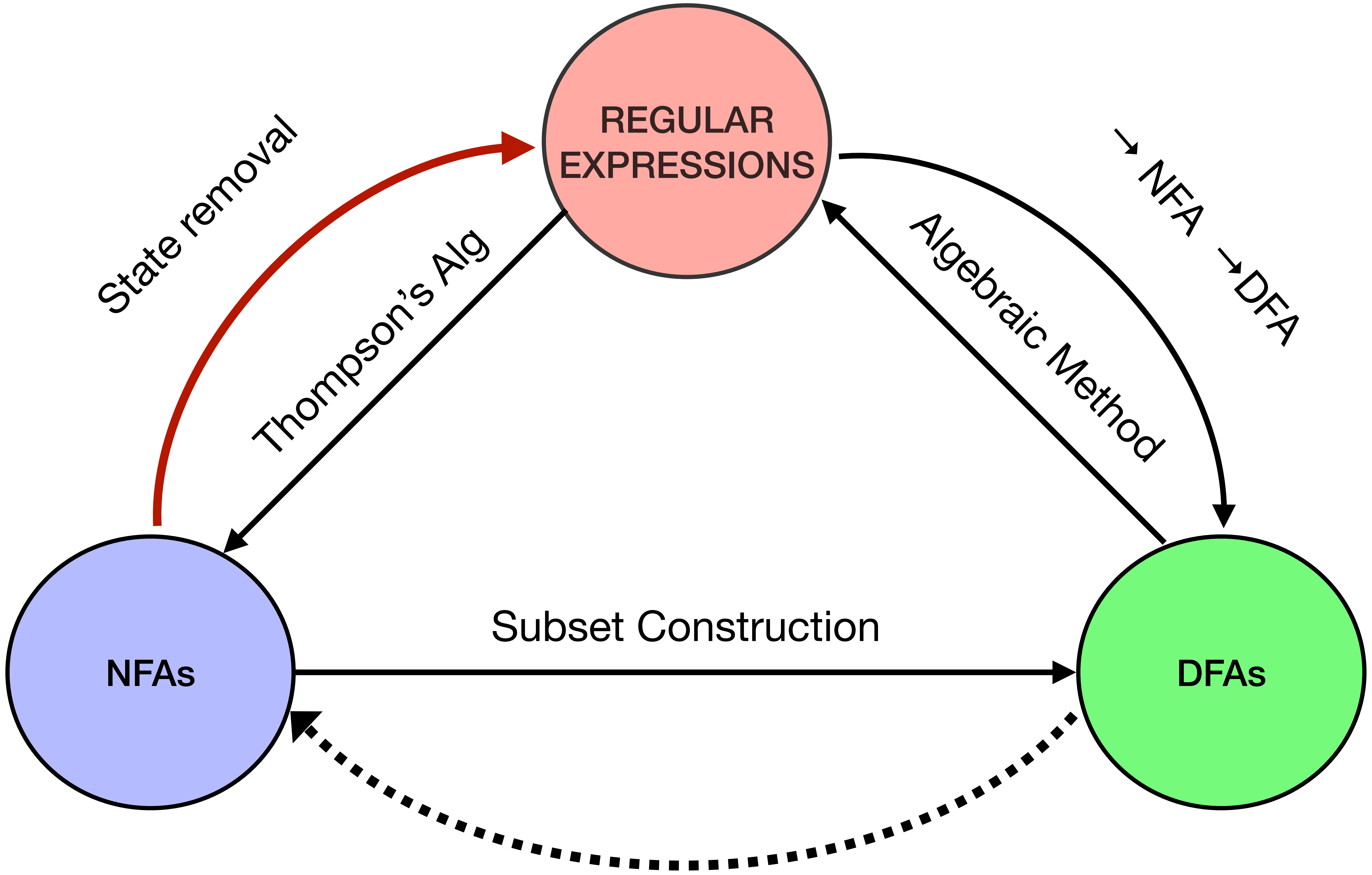
State removal

Key observation

If $q_1 = \delta(q_0, x)$ and $q_2 = \delta(q_1, y)$

then

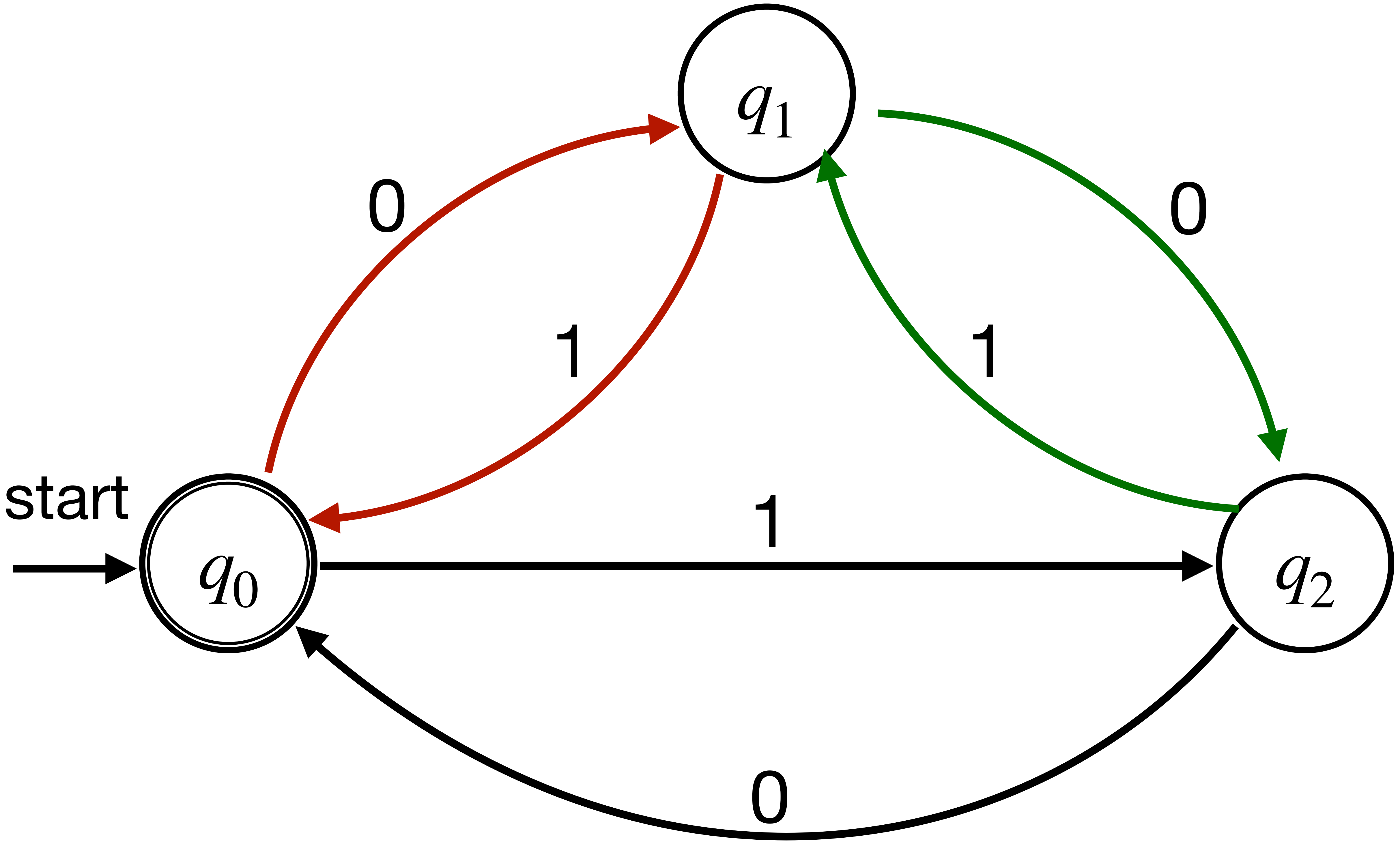
$$q_2 = \delta(q_1, y) = \delta(\delta(q_0, x), y)$$
$$= \delta(q_0, xy)$$



Source: Kani Archive

Converting a DFA to Regular Expression

State removal - example



$q_0 = \delta(q_1, 1)$ $\quad$ $q_1 = \delta(q_0, 0)$

$q_0 = \delta(\delta(q_0, 0), 1)$

$q_0 = \delta(q_0, 01)$

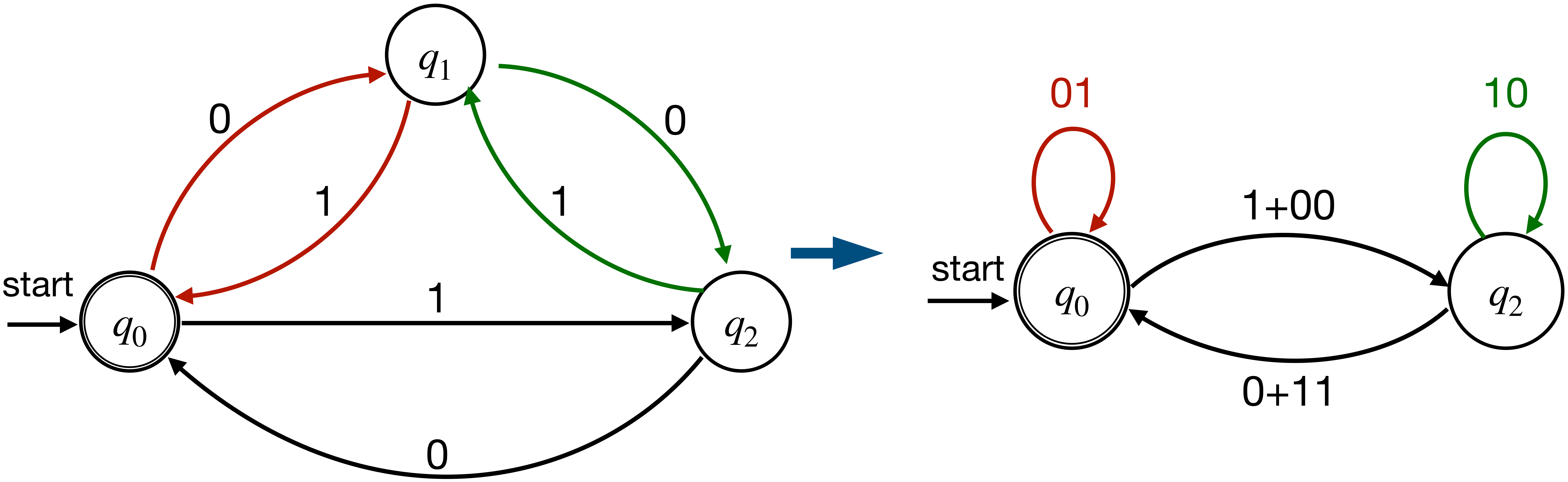
$q_2 = \delta(q_1, 0)$ $\quad$ $q_1 = \delta(q_2, 1)$

$q_2 = \delta(\delta(q_2, 1), 0)$

$q_2 = \delta(q_1, 10)$

Converting a DFA to Regular Expression

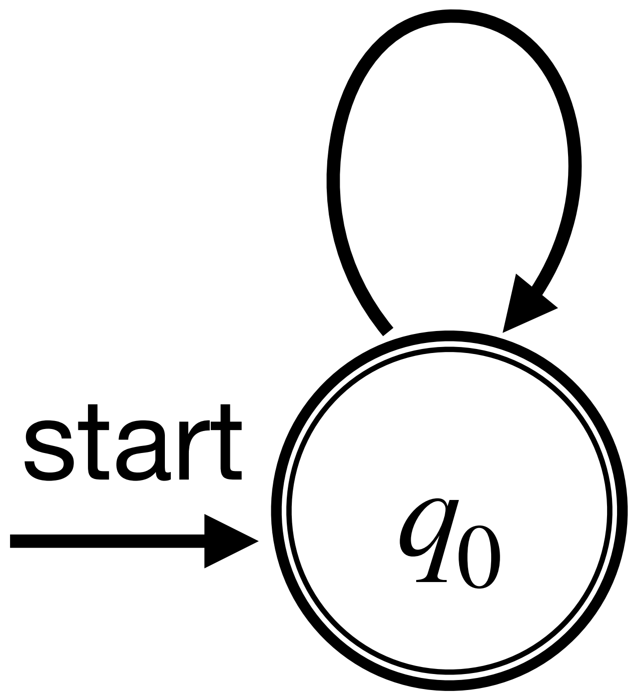
State removal - example

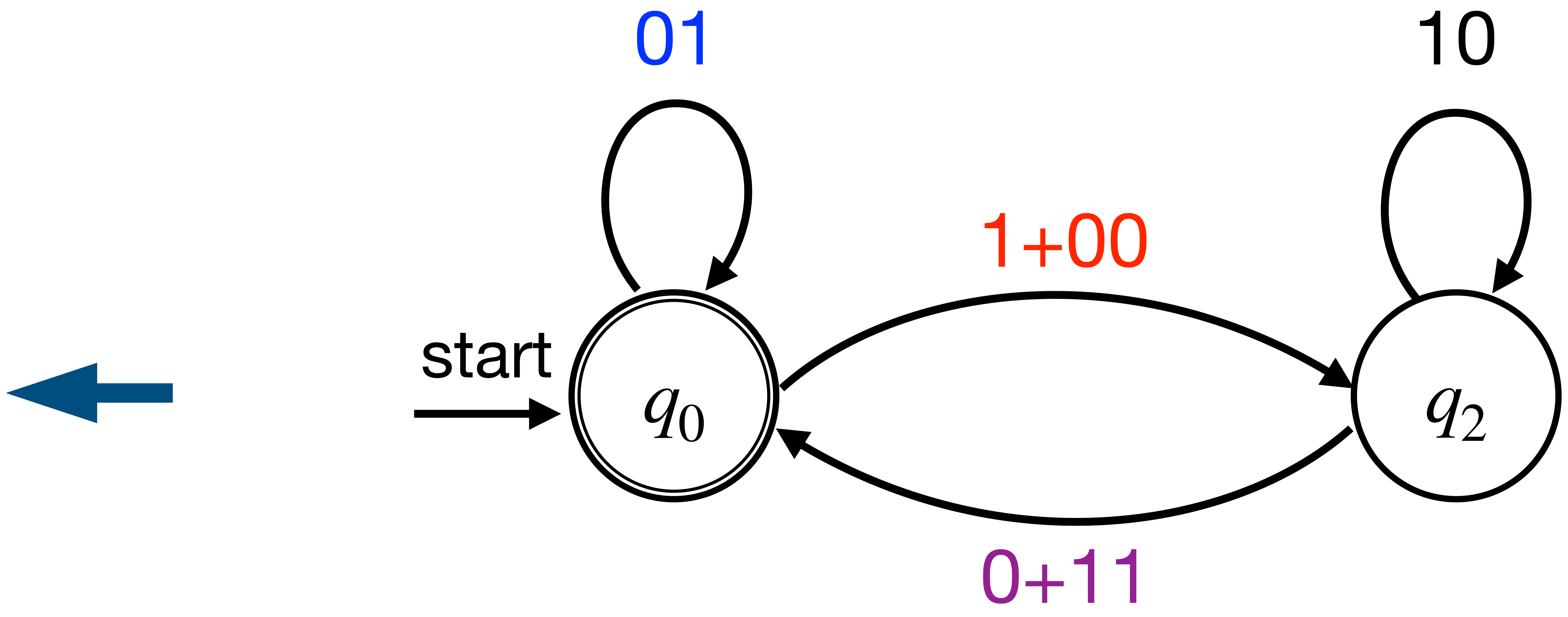


Converting a DFA to Regular Expression

State removal

$$01 + (1 + 00)(10)^*(0 + 11)$$

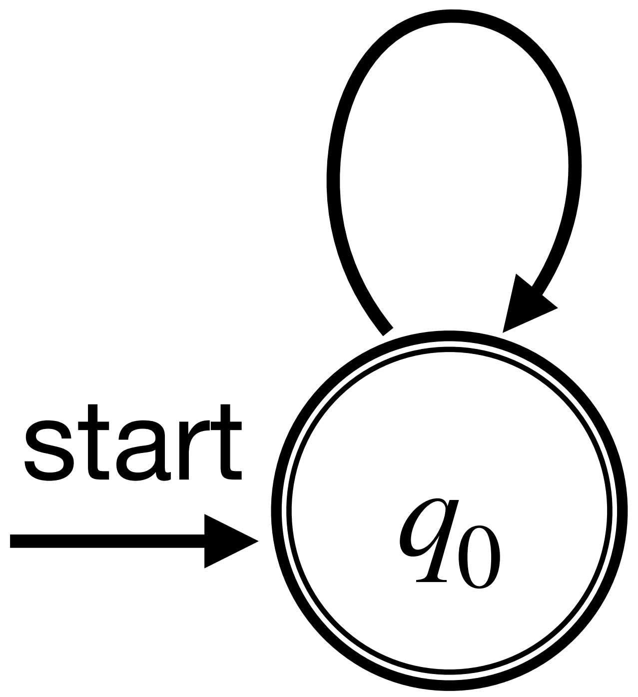




Converting a DFA to Regular Expression

State removal

$01 + (1 + 00)(10)^*(0 + 11)$



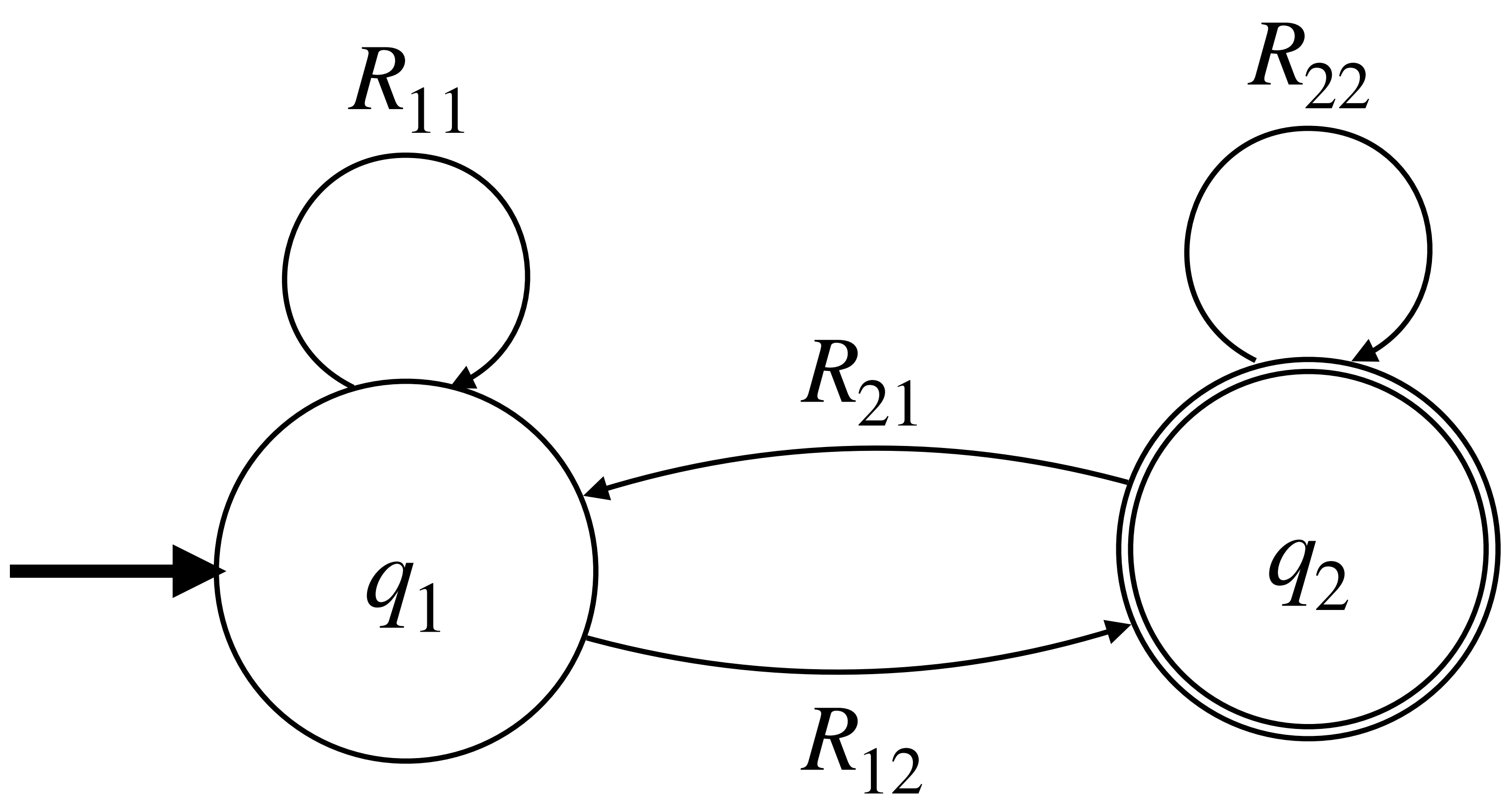
Final expression:

$(01 + (1 + 00)(10)^*(0 + 11))^*$

Converting a NFA to Regular Expression

State removal

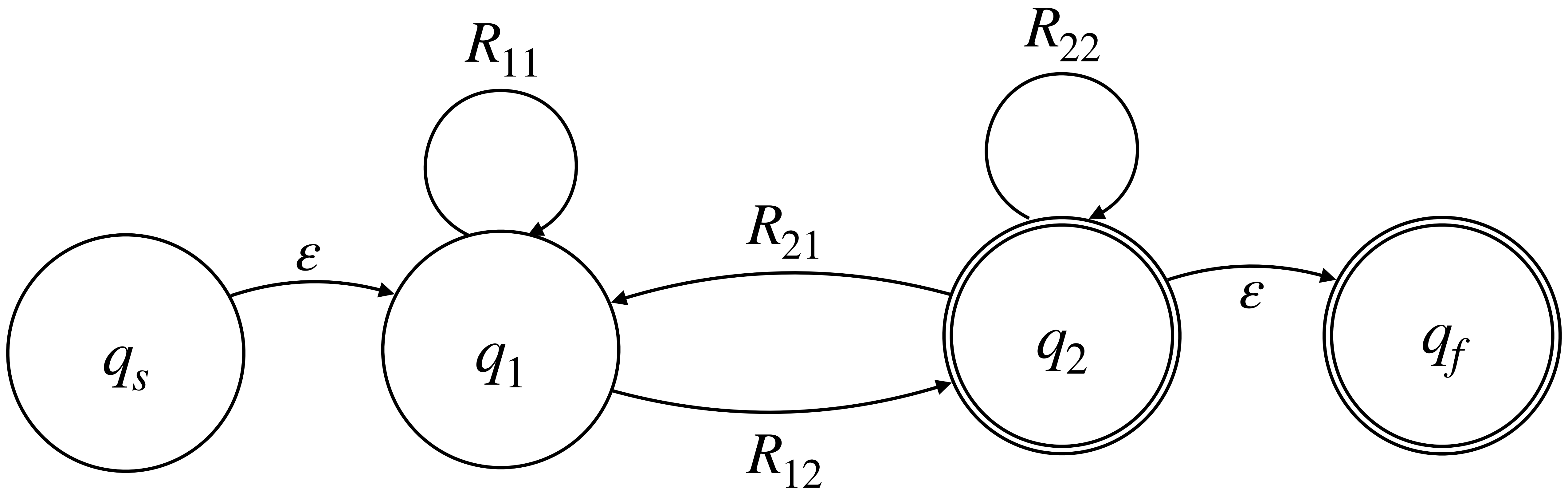
- **Key idea:** We allow for a generalized NFA permitting arbitrary regular expression on the transition arrows.
  - Here R_{11}, R_{12}, R_{21} and R_{22} are valid regular expressions
  - Can we get a clean regular expression from this NFA?



Converting a NFA to Regular Expression

State removal

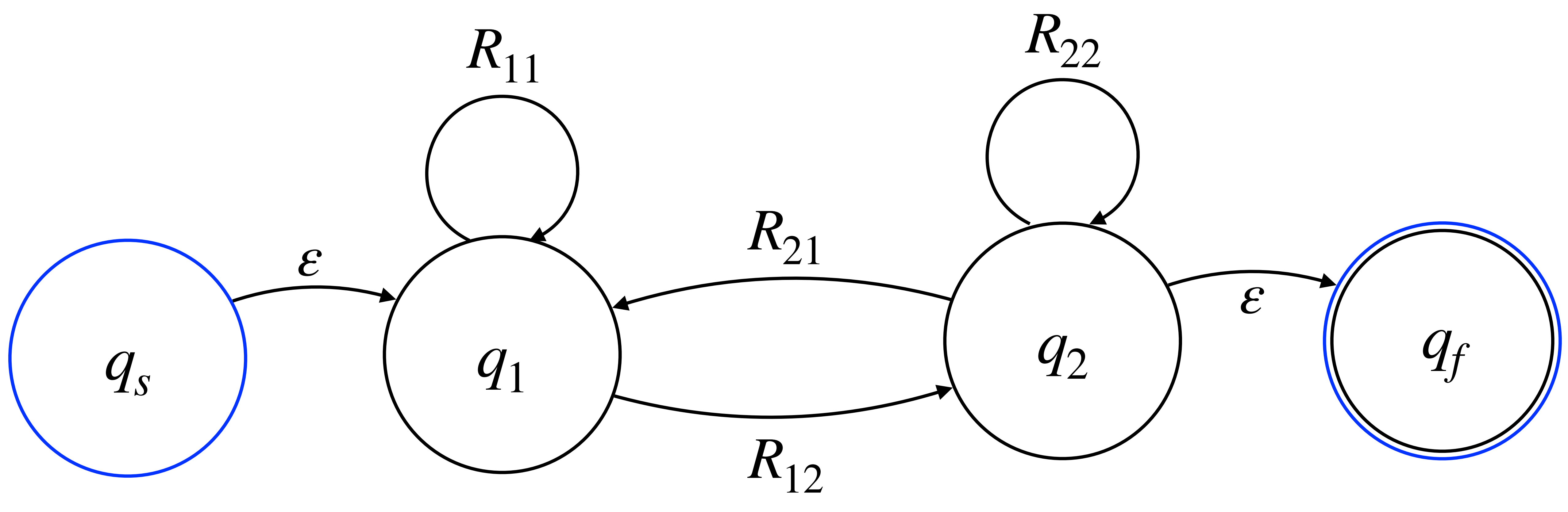
- **Step 1: Normalize**
  - Add a new start state q_s and accept state q_f to the NFA.
  - Add an ε-transition from q_s to the old start state of N.
  - Add ε-transitions from **each** accepting state of N to q_f then mark them as *not accepting.*



Converting a NFA to Regular Expression

State removal

- **Step 2: Remove states**
  - Repeatedly remove states other than q_s and q_f from the NFA by "shortcutting" them until only two states remain: q_s and q_f.
  - The transition from q_s to q_f is then a regular expression for the NFA.



Converting a NFA to Regular Expression

State removal

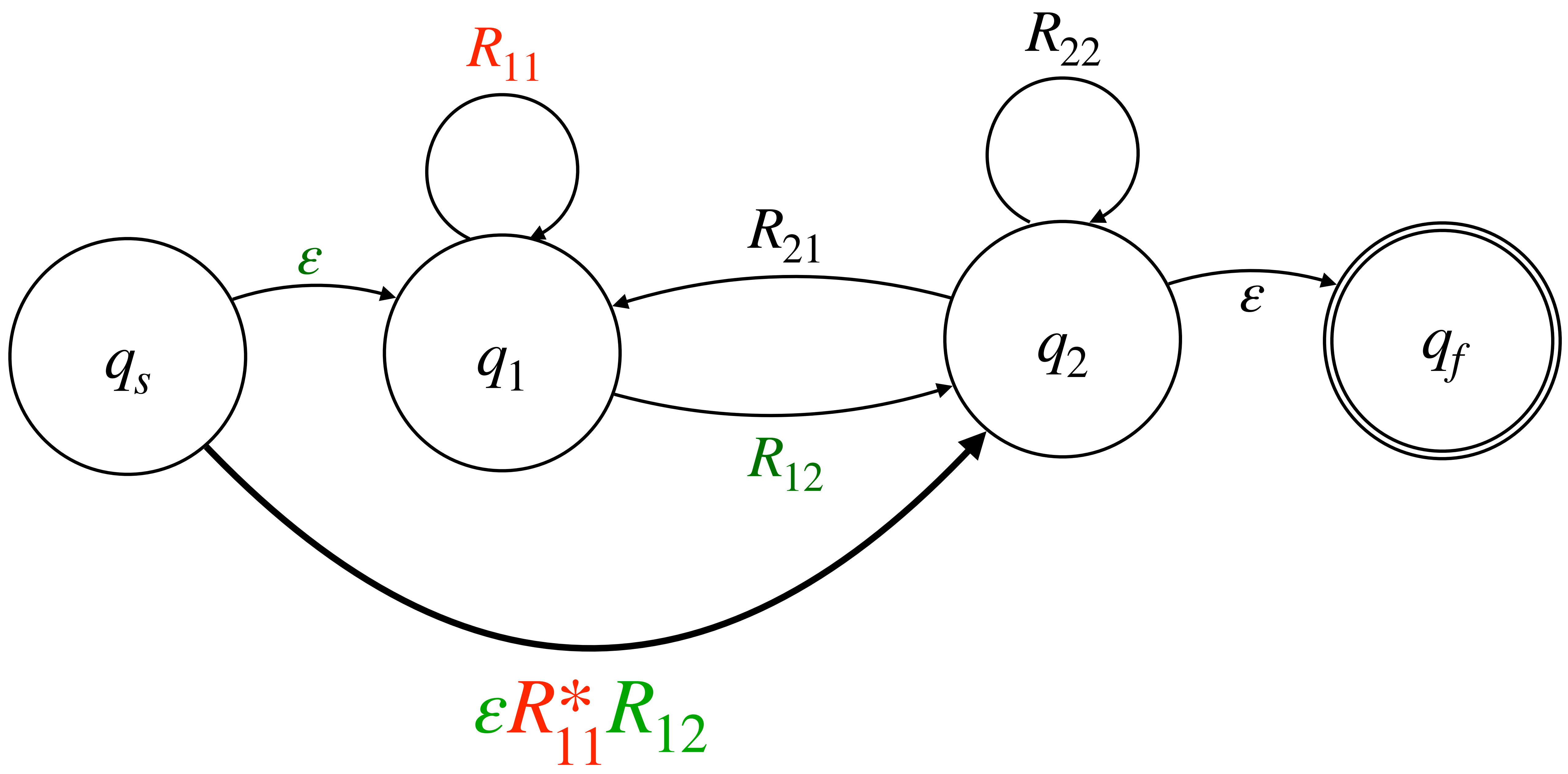
- **Step 2: Details**
  - For each pair (q_1, q_2) such that

$$q_1 \xrightarrow{R_{in}} q, \ \ q \xrightarrow{R_{out}} q_2$$

Add a transition such that

$$q_2 = \delta\left(q_1, R_{in} \cdot R_q^* \cdot R_{out}\right)$$

where R_q is a self-transition (if any)



Converting a NFA to Regular Expression

State removal

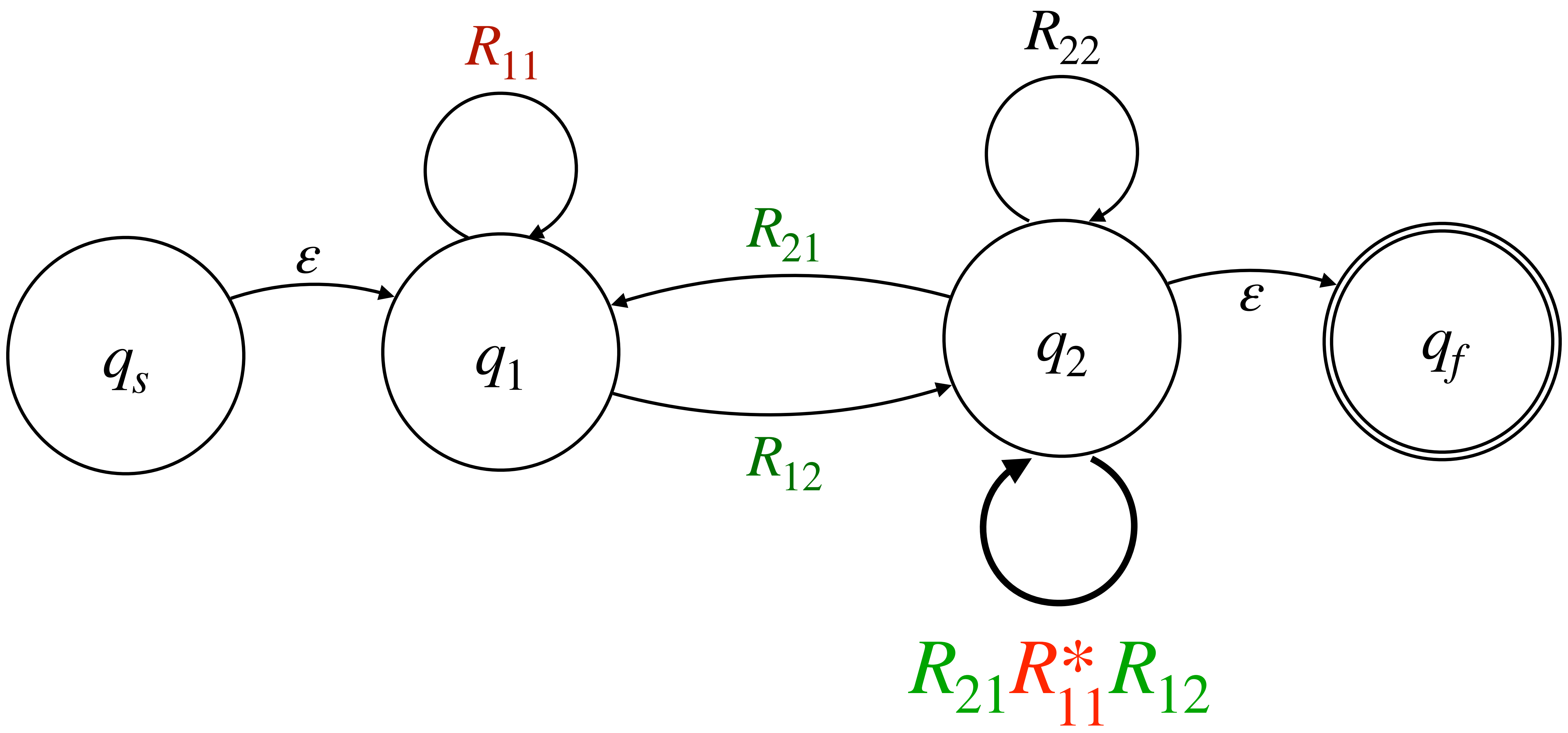
- **Step 2: Details**
  - For each pair (q_1, q_2) such that

$$q_1 \xrightarrow{R_{in}} q, \;\; q \xrightarrow{R_{out}} q_2$$

Add a transition such that

$$q_2 = \delta\left(q_1, R_{in} \cdot R_q^* \cdot R_{out}\right)$$

where R_q is a self-transition (if any)



Converting a NFA to Regular Expression

State removal

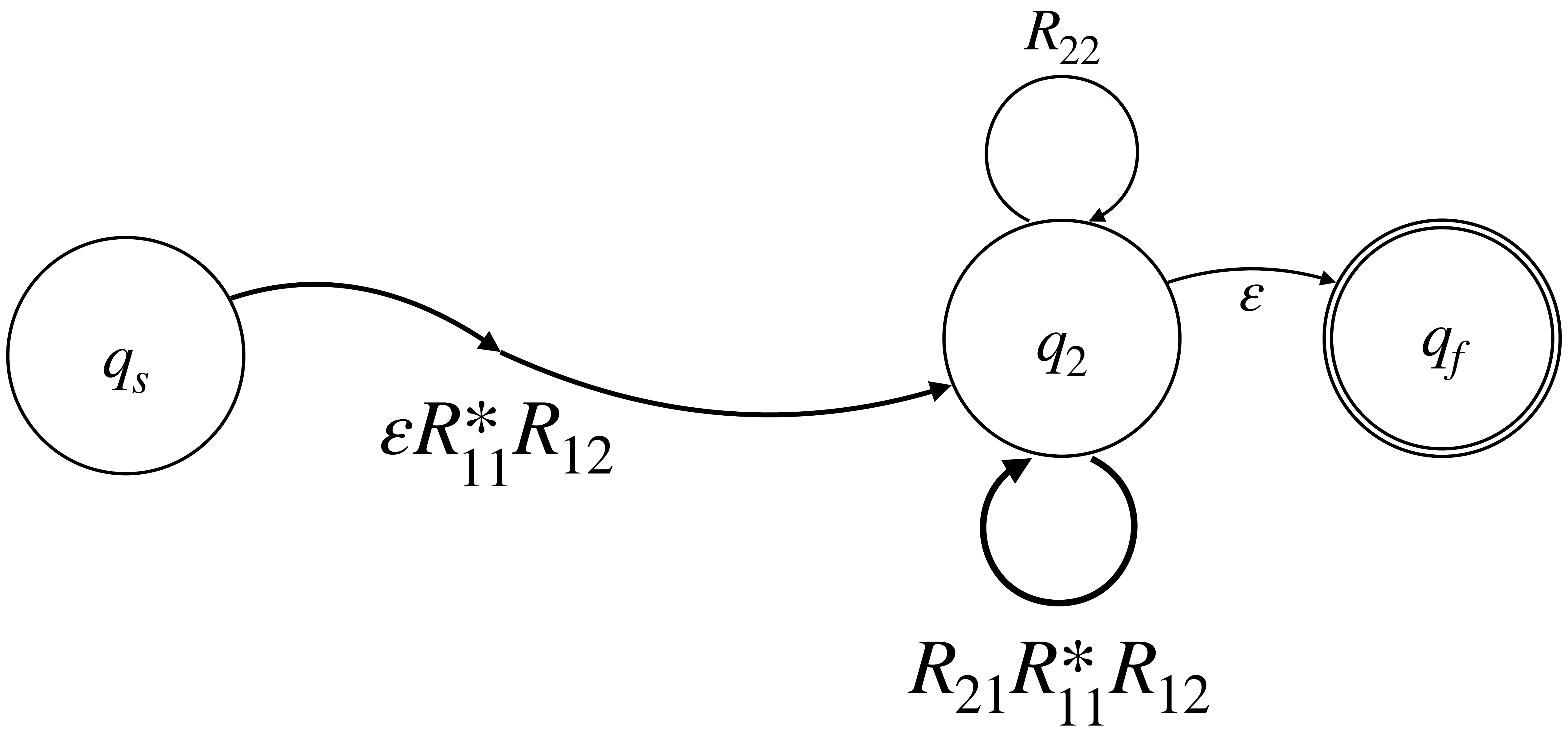
- **Step 2: Details**
  - For each pair (q_1, q_2) such that

$$q_1 \xrightarrow{R_{in}} q, \quad q \xrightarrow{R_{out}} q_2$$

Add a transition such that

$$q_2 = \delta\left(q_1, R_{in} \cdot R_q^* \cdot R_{out}\right)$$

where R_q is a self-transition (if any)



Converting a NFA to Regular Expression

State removal

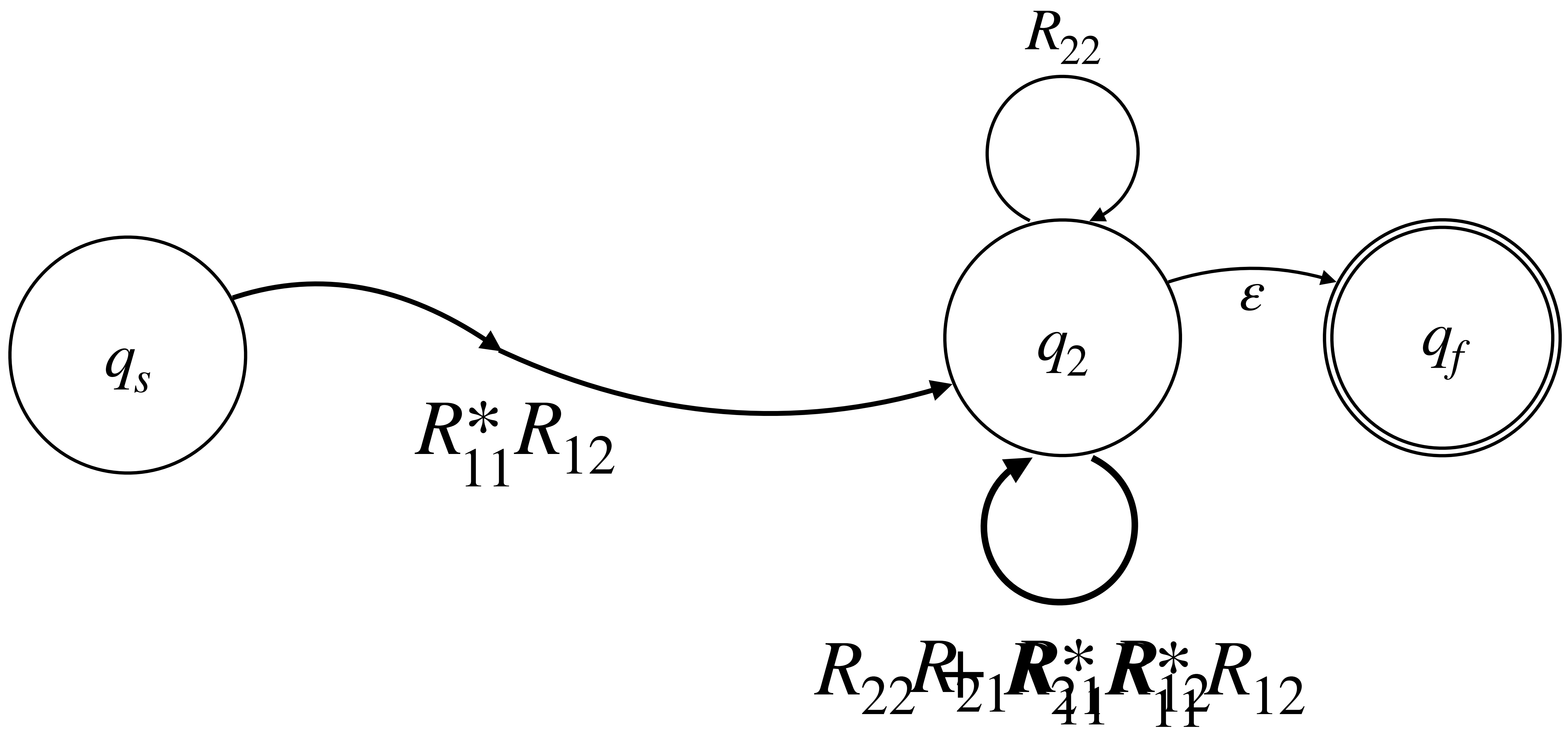
- **Step 2: Details**
  - For each pair (q_1, q_2) such that

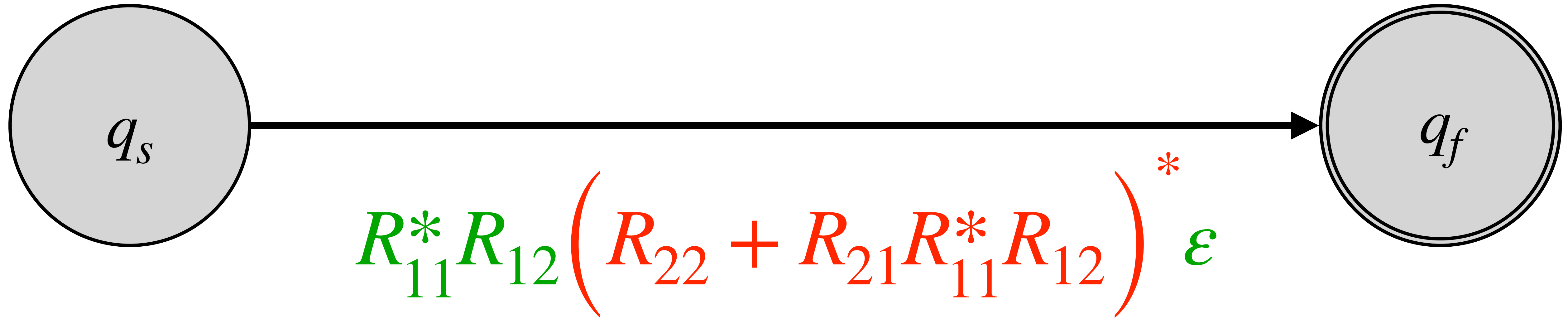
    $$q_1 \xrightarrow{R_{in}} q, \quad q \xrightarrow{R_{out}} q_2$$

    Add a transition such that

    $$q_2 = \delta\left(q_1, R_{in} \cdot R_q^* \cdot R_{out}\right)$$

    where R_q is a self-transition (if any)
  - Use union operation to handle multiple transitions





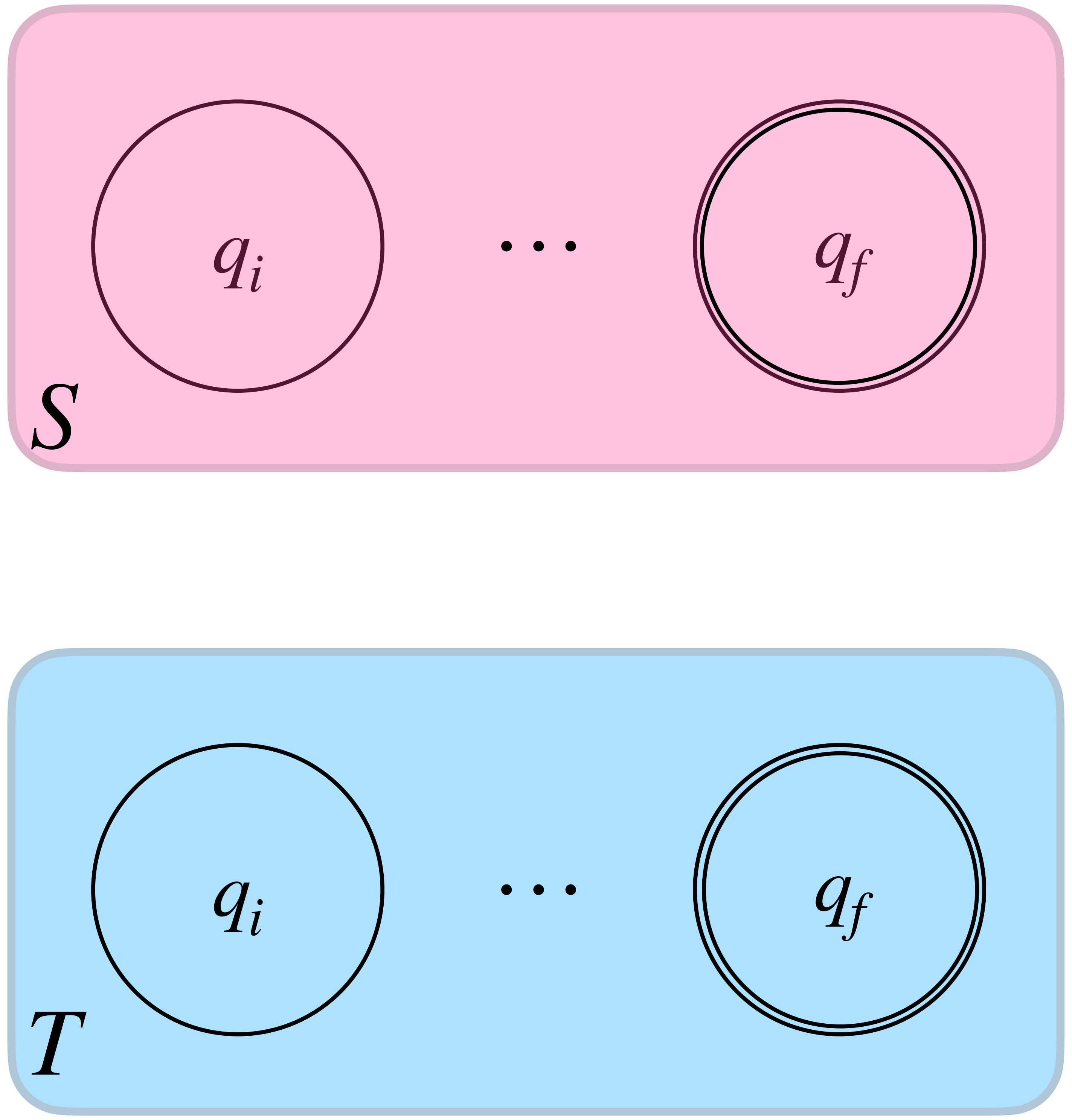
Equivalence of NFAs and Regular Expressions - Thompson's algorithm

NFA from a RegEx

Thompson's algorithm

- **Key idea:** Represent regular operations (Union, Concatenation & Kleene Star) using NFAs.

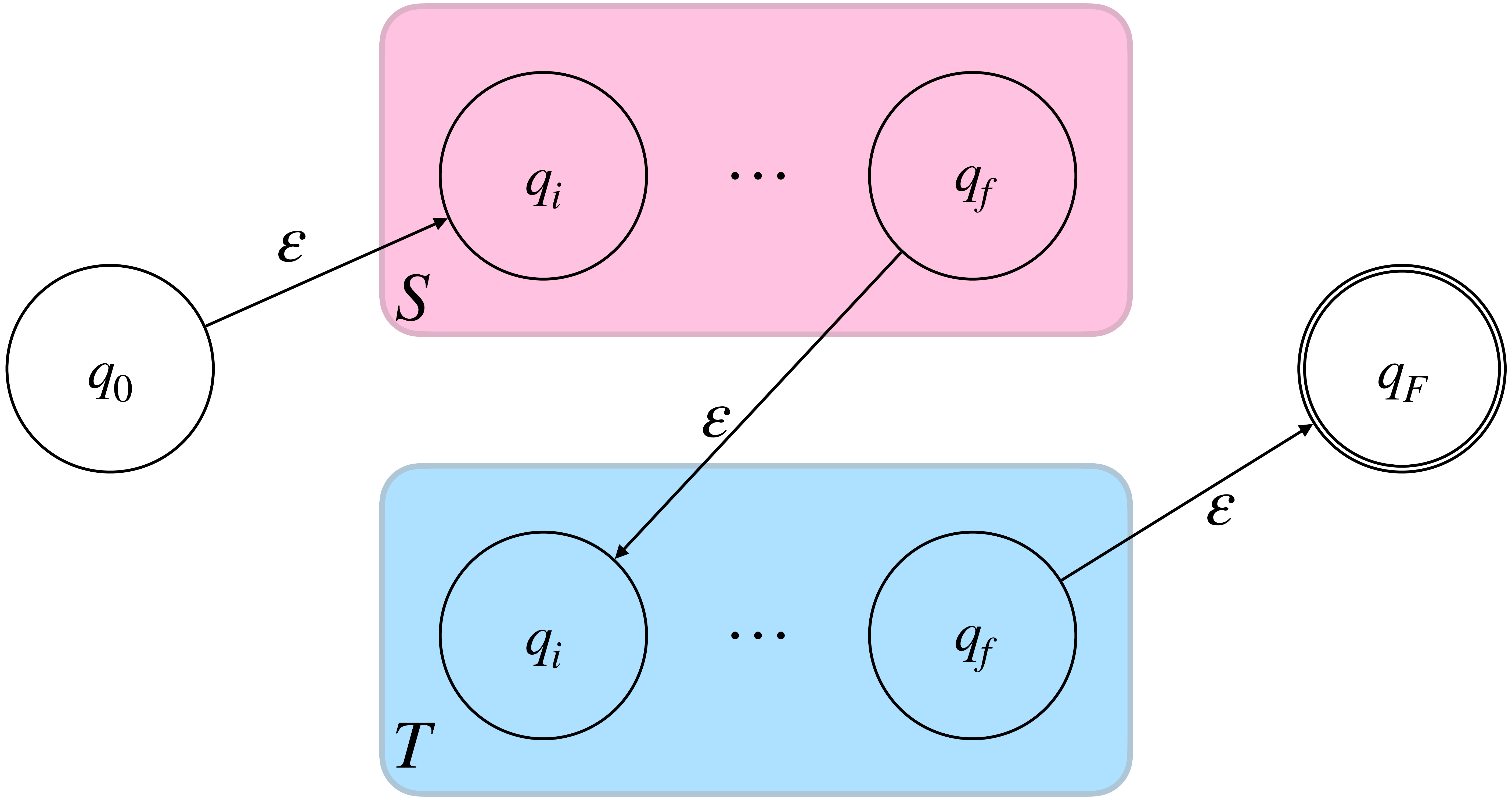
- Given: Two NFAs S and T representing languages L_S and L_T
    - What NFA represents $L_S \cdot L_T$, $L_S + L_T$ and L_S^*



NFA from a RegEx

Regular operation rules

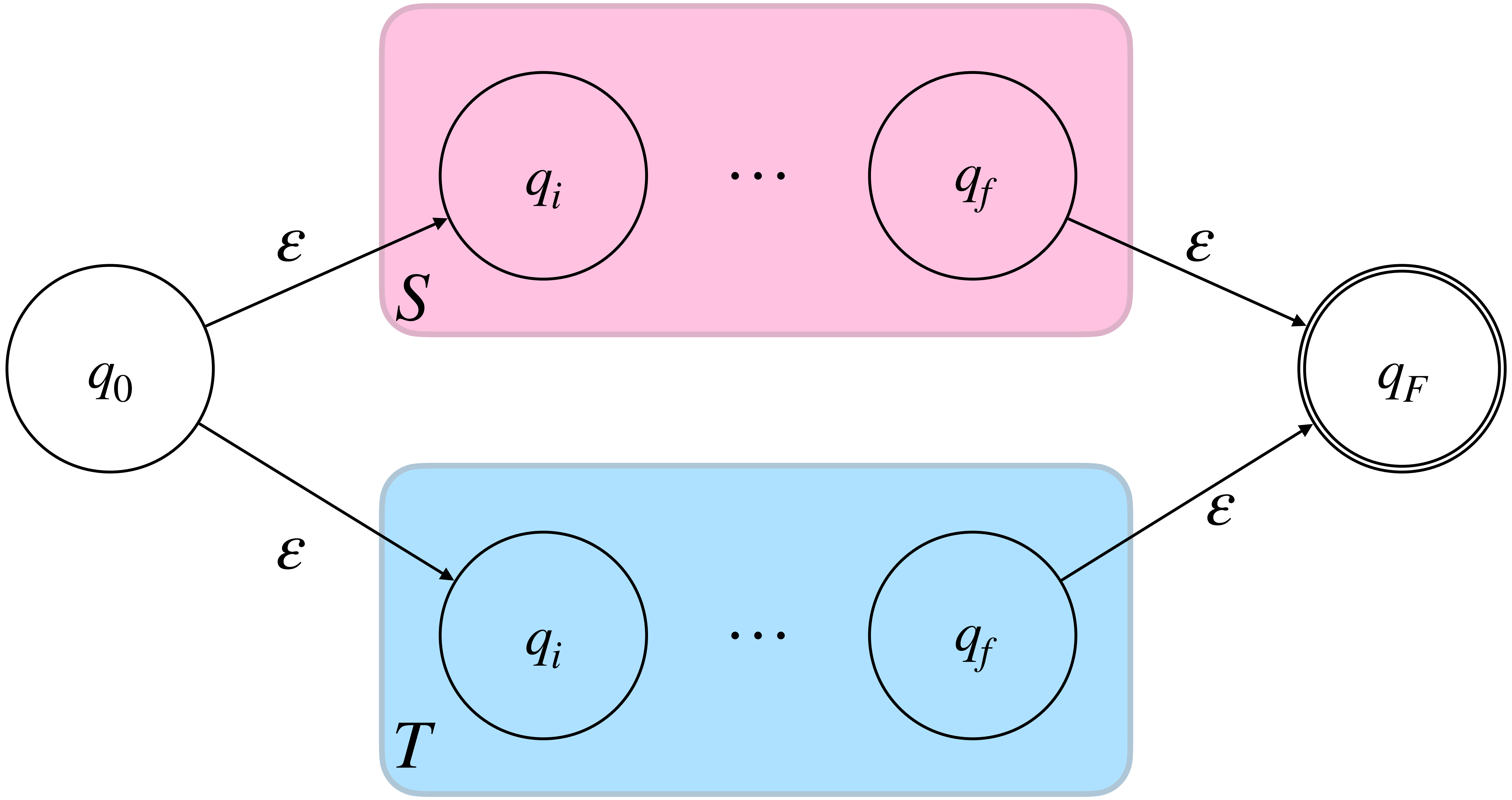
- **Concatenation**
- $\mathbf{L} = L_s \cdot L_t$
  - "Series connection"



NFA from a RegEx

Regular operation rules

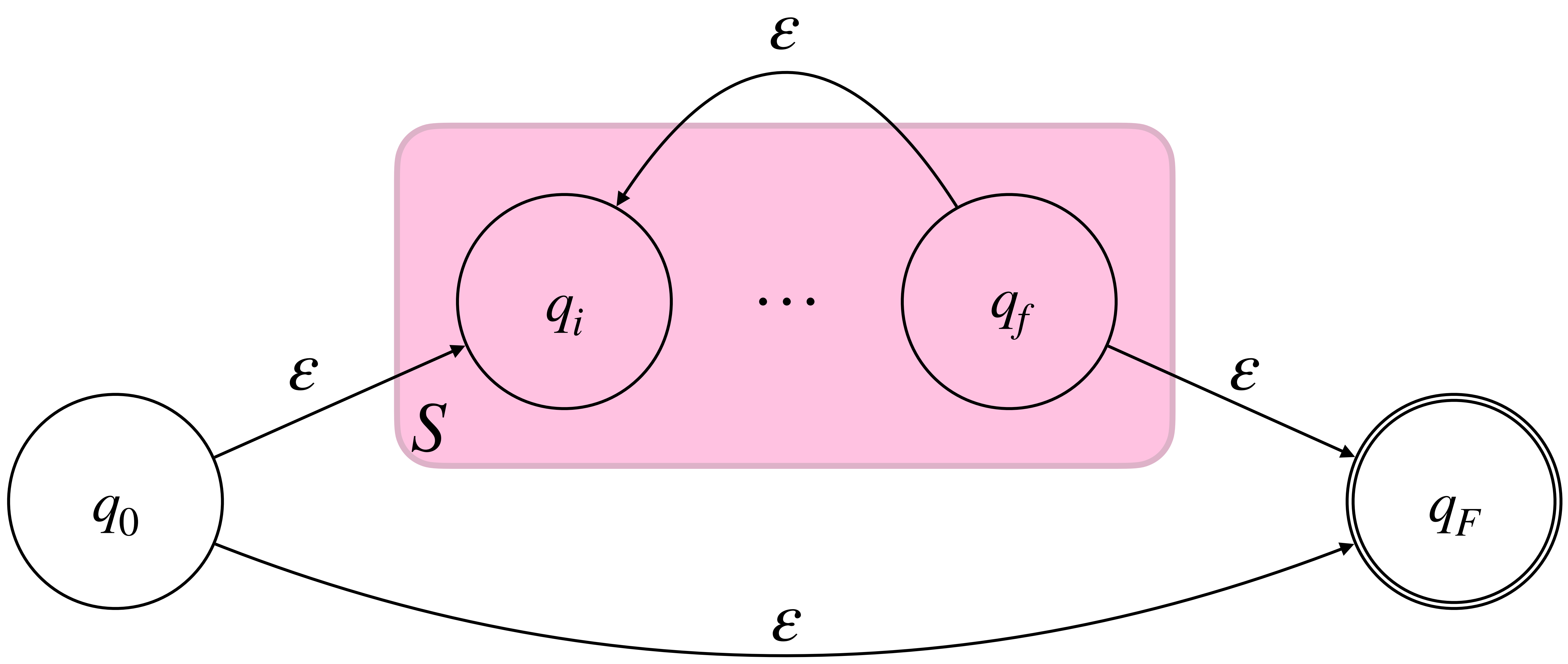
- **Union**
- $\mathbf{L} = L_S + L_T$
  - "Parallel connection"



NFA from a RegEx

Regular operation rules

- **Kleene star**
- $\mathbf{L} = L_s^*$
  - Need to allow the empty string
  - Need to allow multiple copies of any $w \in L_S$



NFA from a RegEx

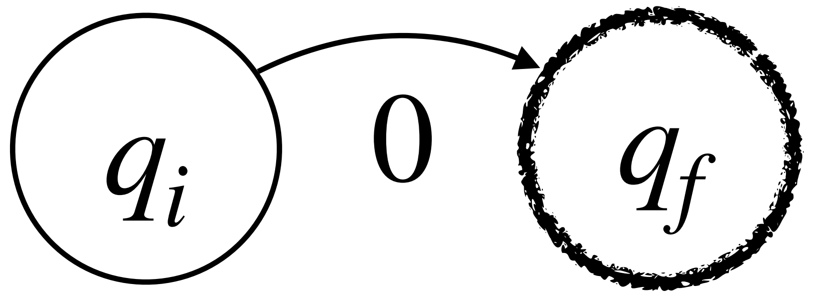
Example

- **Find an NFA for** $(0+1)^*(101+010)(0+1)^*$

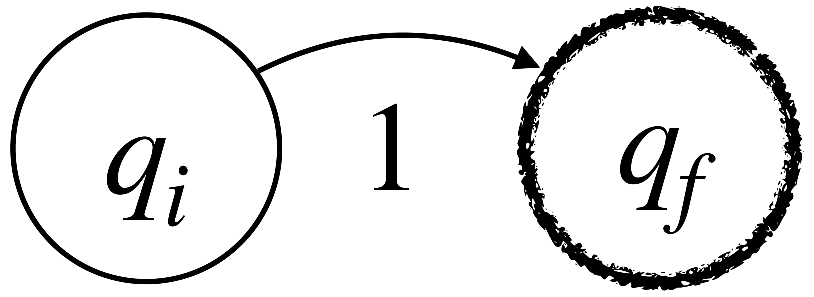
- Rewrite:

$$\underbrace{(0+1)^*}_{N_A} \cdot \underbrace{(101+010)}_{N_B} \cdot \underbrace{(0+1)^*}_{N_A} = \underbrace{(0+1)^*}_{(N_0+N_1)^*} \cdot (\underbrace{101}_{N_C} + \underbrace{010}_{N_D}) \cdot \underbrace{(0+1)^*}_{(N_0+N_1)^*}$$

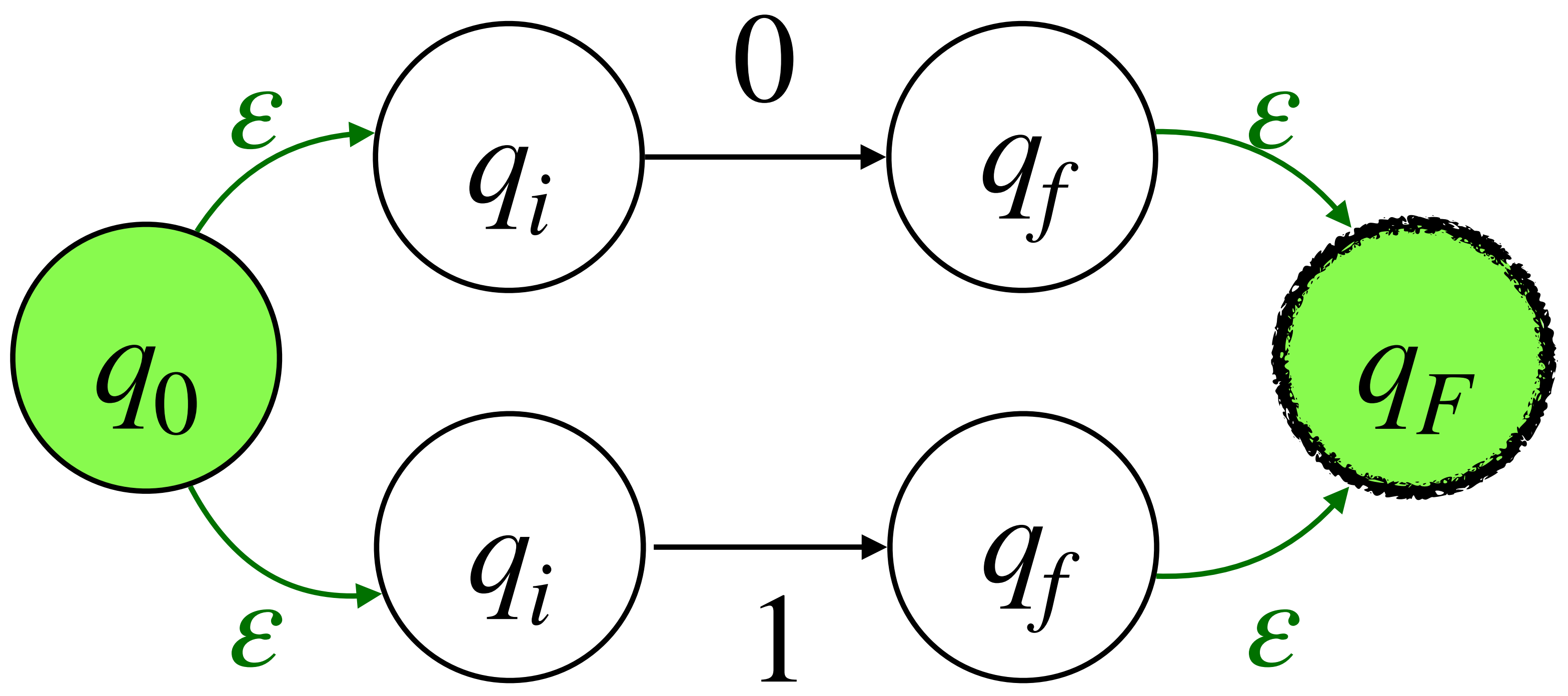
N_0



N_1



$N_0 + N_1$

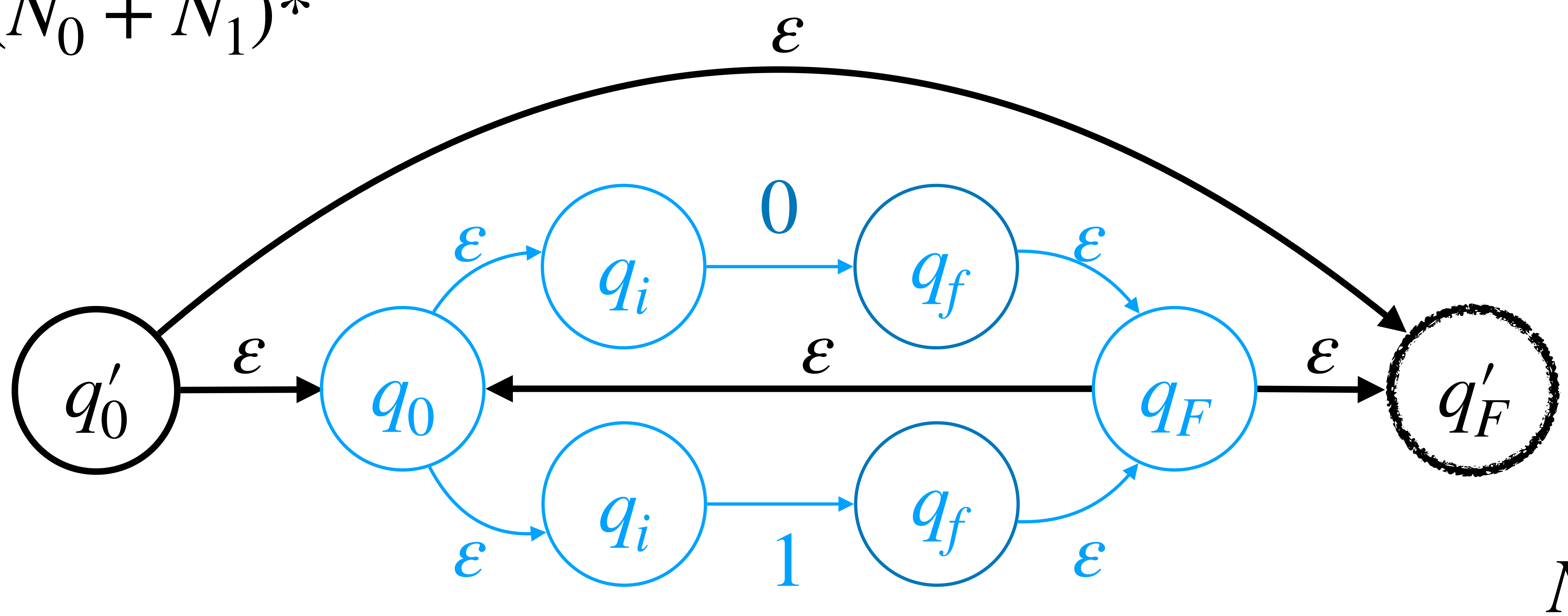


NFA from a RegEx

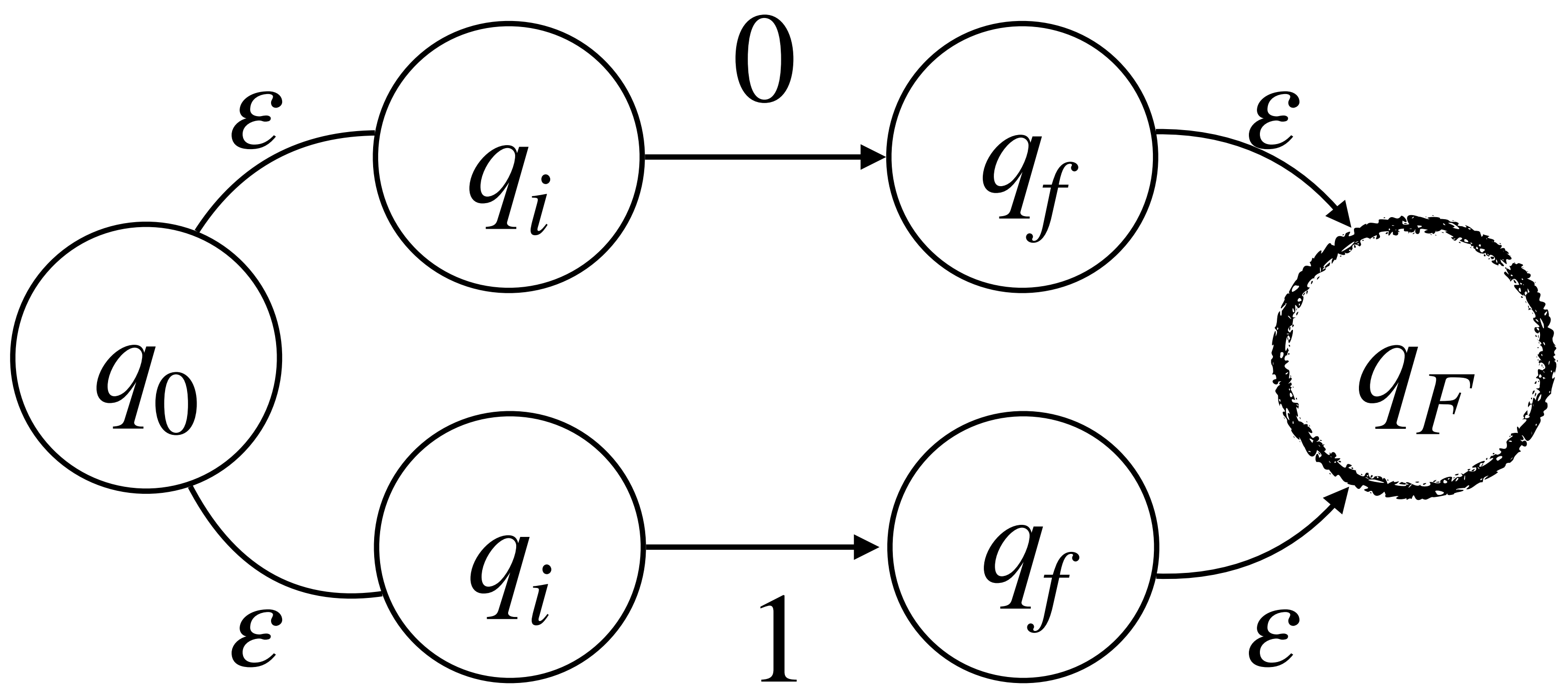
Example

$$\underbrace{(0+1)^*}_{N_A} \cdot \underbrace{(101+010)}_{N_B} \cdot \underbrace{(0+1)^*}_{N_A} = \underbrace{(0+1)^*}_{(N_0+N_1)^*} \cdot (\underbrace{101}_{N_C} + \underbrace{010}_{N_D}) \cdot \underbrace{(0+1)^*}_{(N_0+N_1)^*}$$

$(N_0 + N_1)^*$



$N_0 + N_1$

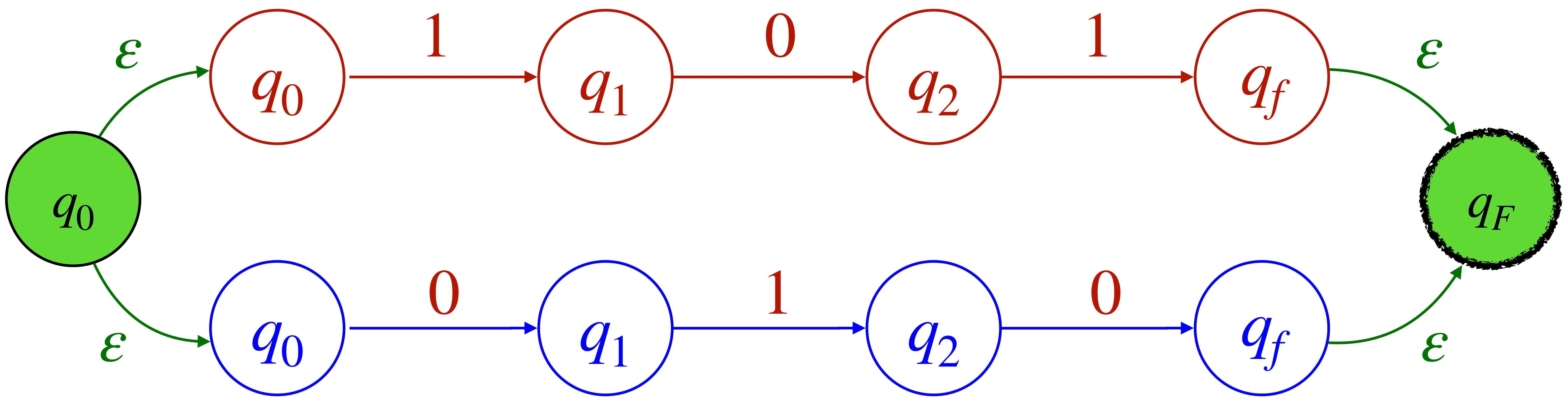


NFA from a RegEx

Example

$$\underbrace{(0+1)^*}_{N_A} \cdot \underbrace{(101+010)}_{N_B} \cdot \underbrace{(0+1)^*}_{N_A} = \underbrace{(0+1)^*}_{(N_0+N_1)^*} \cdot (\underbrace{101}_{N_C} + \underbrace{010}_{N_D}) \cdot \underbrace{(0+1)^*}_{(N_0+N_1)^*}$$

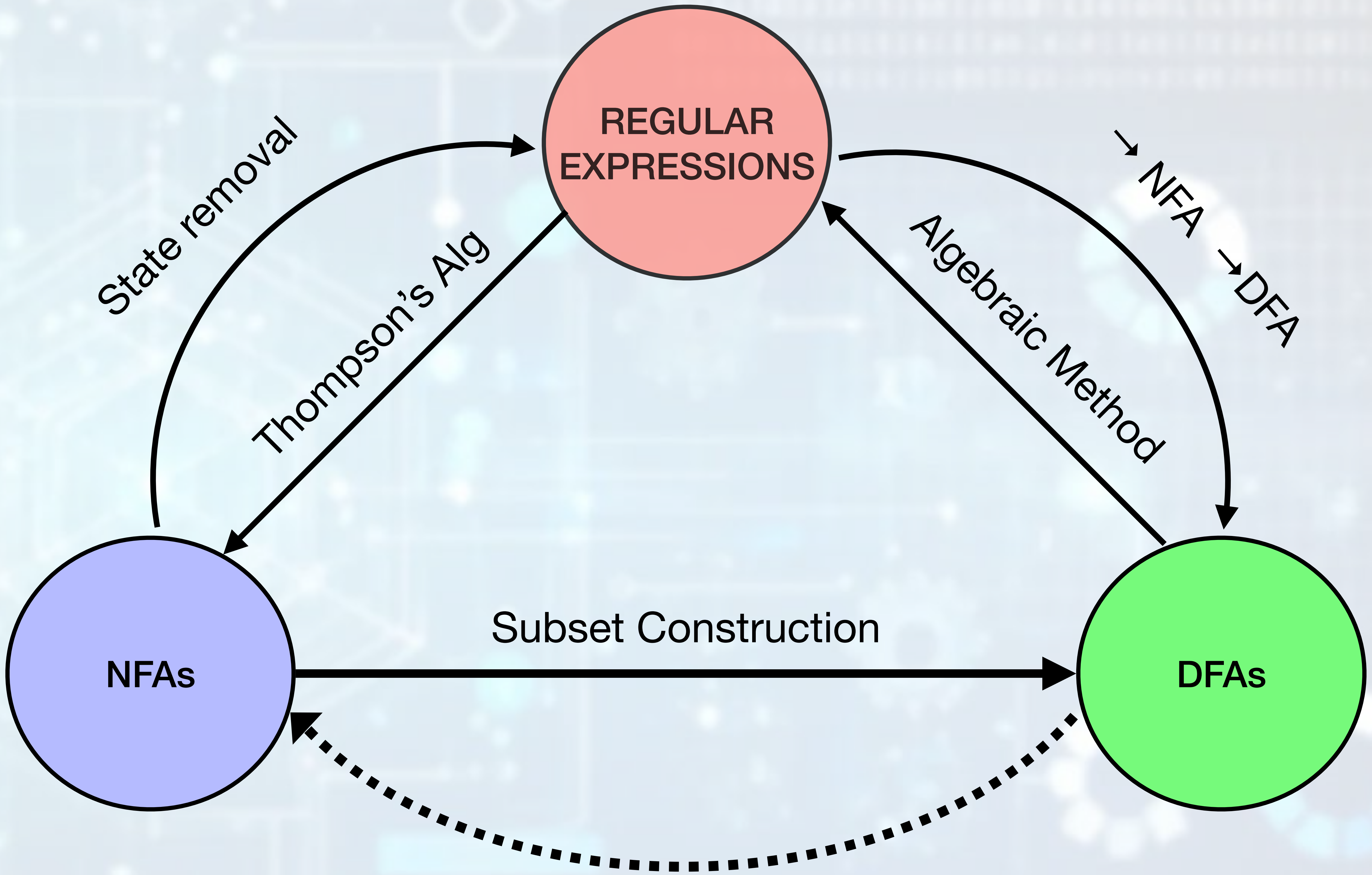
$(N_C + N_D)$



Regular Expression to DFA - Brzozowski's algorithm

Skipped - see Kani Archive for more information

Figure from Kani Archive



Summary

Next class: Languages that are not regular

Image by ChatGPT (probably collaborated with DALL-E)