Introduction to the Theory of Computation

Set 4 — Regular Languages (3)

Equivalence of RE's and DFA's

We have seen that every RE has a corresponding NFA

- Therefore, every RE has a corresponding DFA
- Thus every RE describes a regular language

We need to show that every regular language can be described by a RE

Begin by showing how to convert all DFA's into *G*NFA's

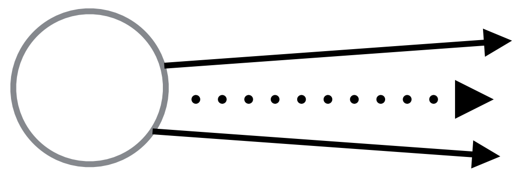
- *Generalized* Nondeterministic Finite Automata

JFLAP uses a *Generalized Transition Graph* (GTG)

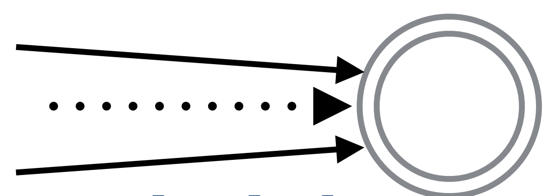
GNFA's

A GNFA is an NFA with the following properties:

- The start state has transition arrows going to every other state, but no arrows coming in from any other state



- There is exactly one accept state and there is an arrow from every other state to this state, but no arrows to any other state from the accept state

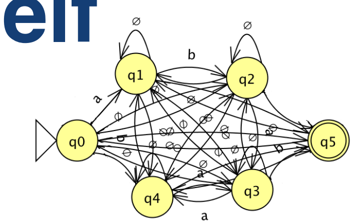


- The start state is not the accept state

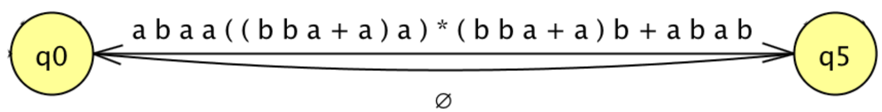
GNFA's (continued)

A GNFA is an NFA with the following properties:

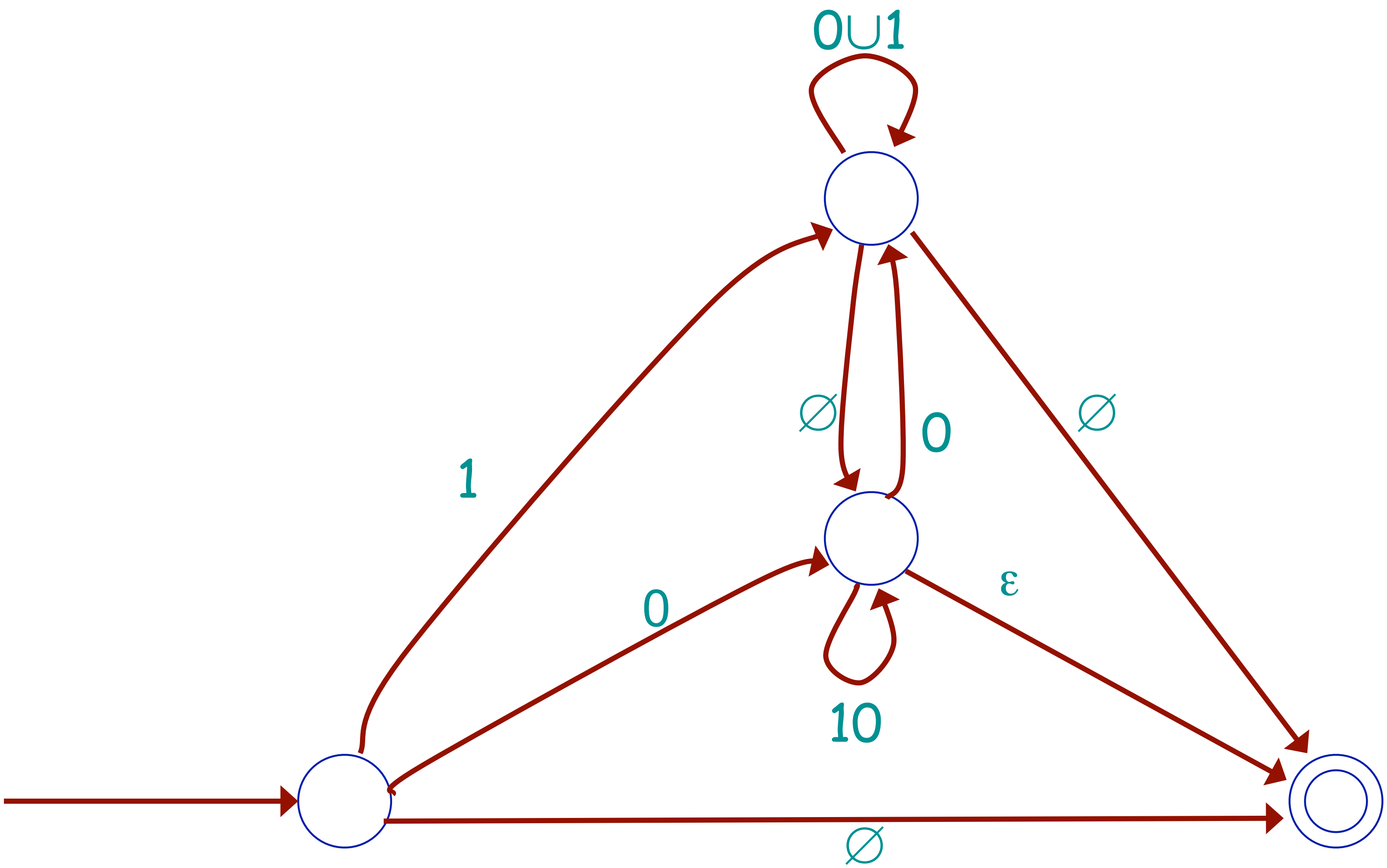
- Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself



- Instead of being labeled with symbols from the alphabet, transitions are labeled with regular expressions



Example GNFA



Equivalence of DFA's and RE's

First show every DFA can be converted into a GNFA that accepts the same language

Then show that any GNFA has a corresponding RE that represents the same language

Converting a DFA into a GNFA

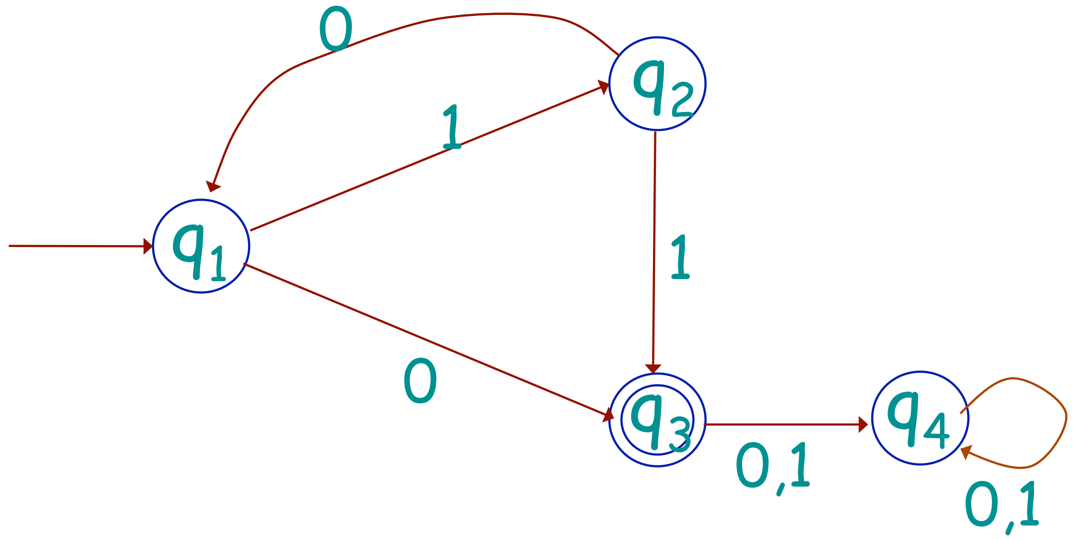
Add two new states

- **New start state** with an ε jump to the original DFA's start state
- **New accept state** with an ε jump from each of the original DFA's accept states
  - This new state will be the *only* accept state

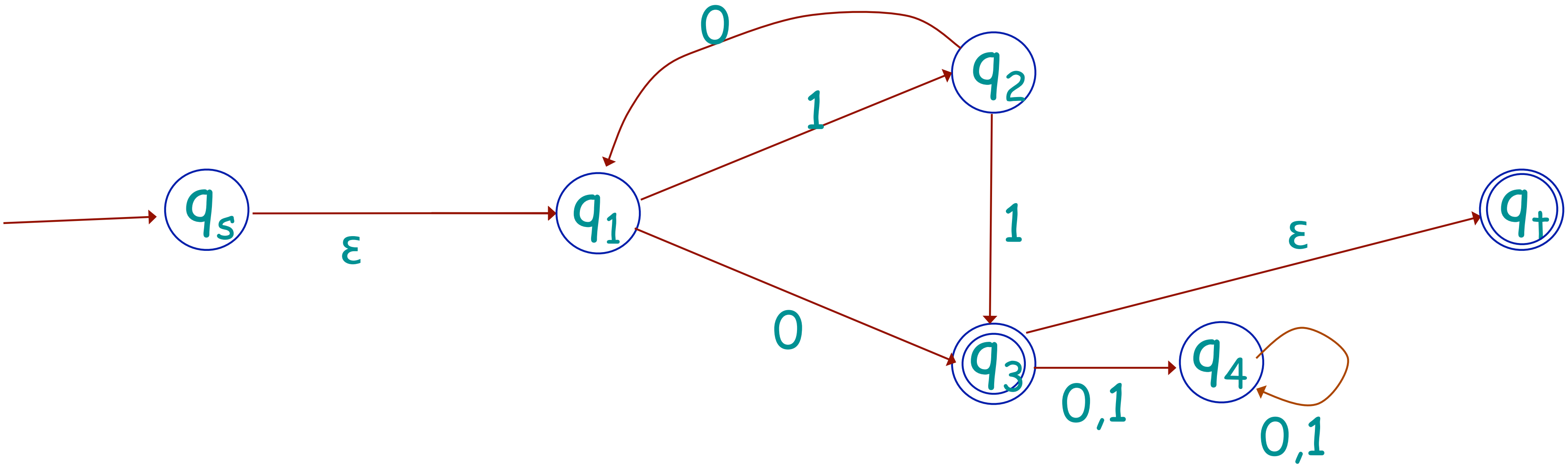
All transitions labeled with multiple labels are relabeled with the *union* of the previous labels

All pairs of states without transitions get a transition labeled ∅

Converting a DFA to a GNFA

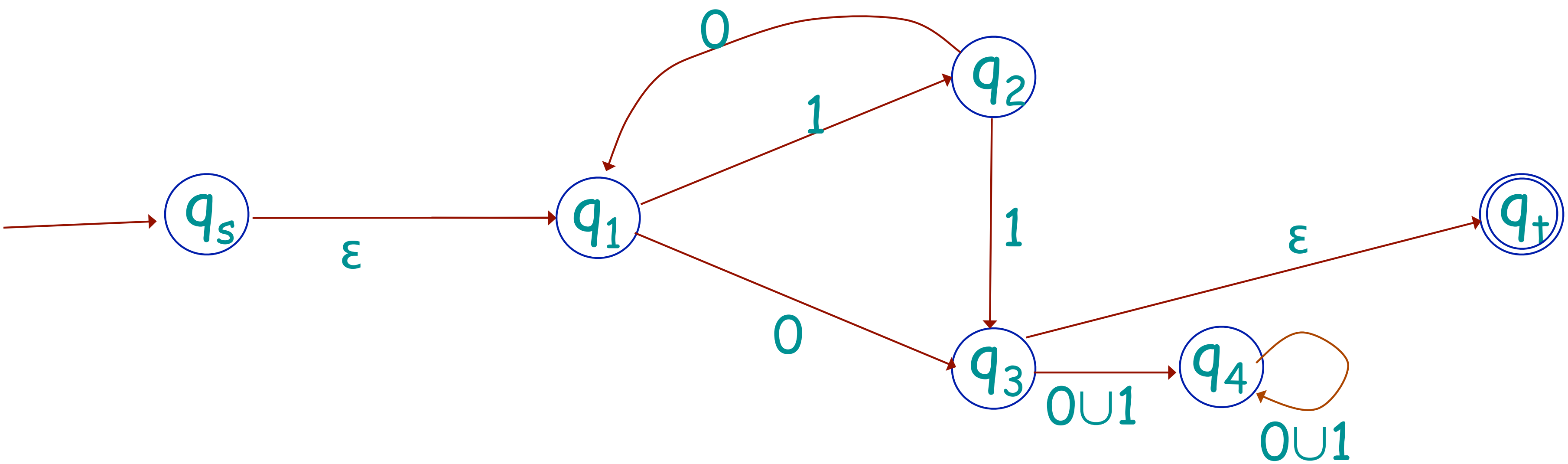


Converting a DFA to a GNFA



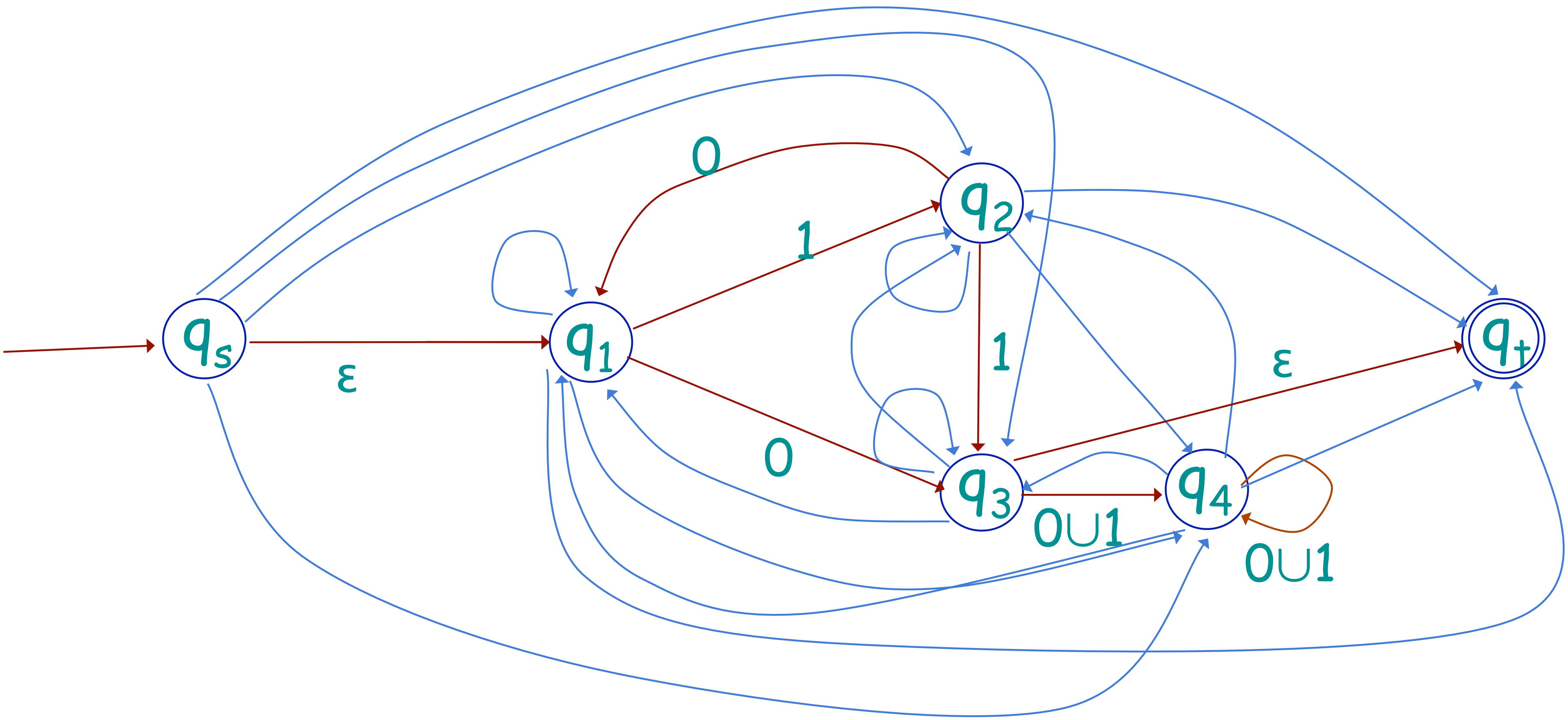
Add two new states

Converting a DFA to a GNFA



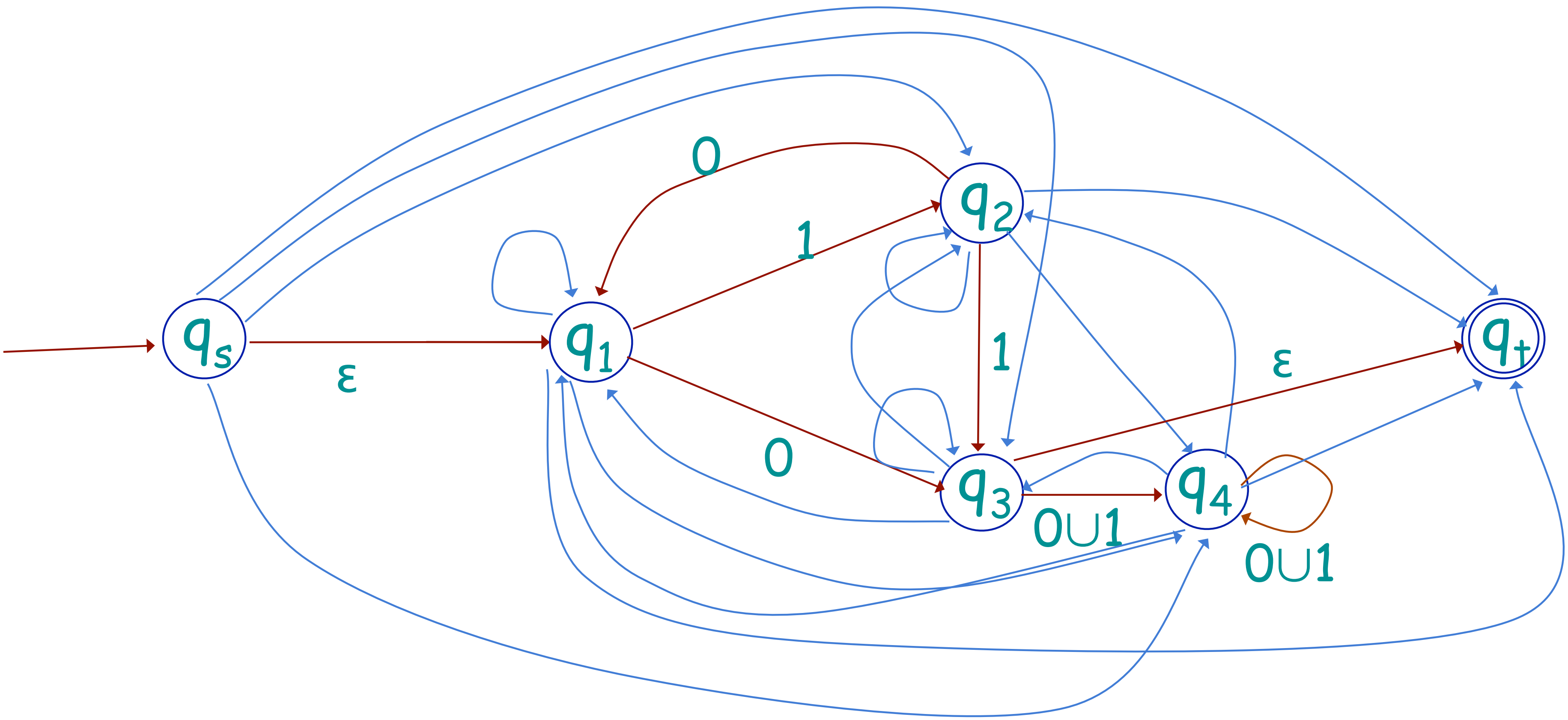
All transitions with multiple labels are relabeled with the union of the previous labels

Converting a DFA to a GNFA



All pairs of states without transitions get a transition labeled ∅

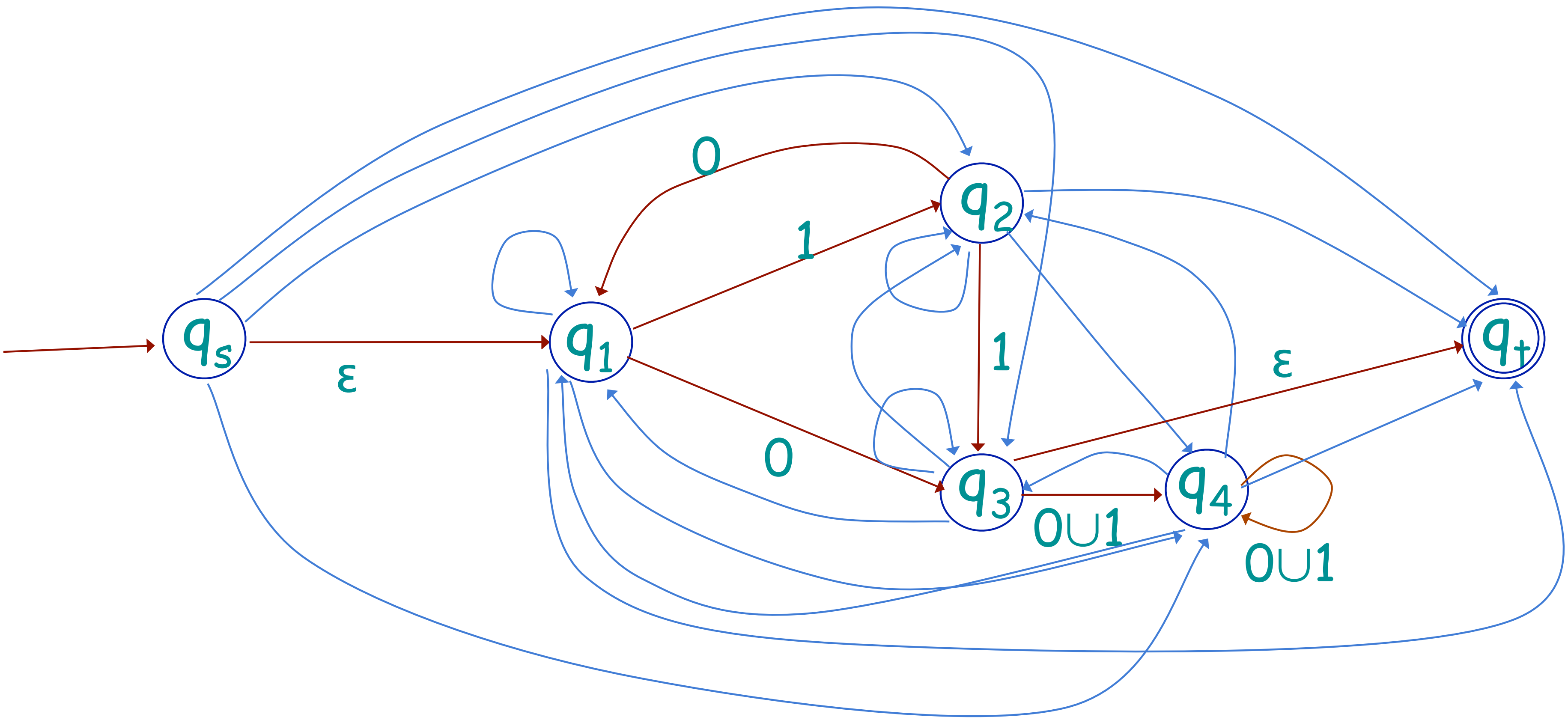
Converting a DFA to a GNFA



The resulting state diagram is a GNFA

- All GNFA properties are satisfied

Converting a DFA to a GNFA



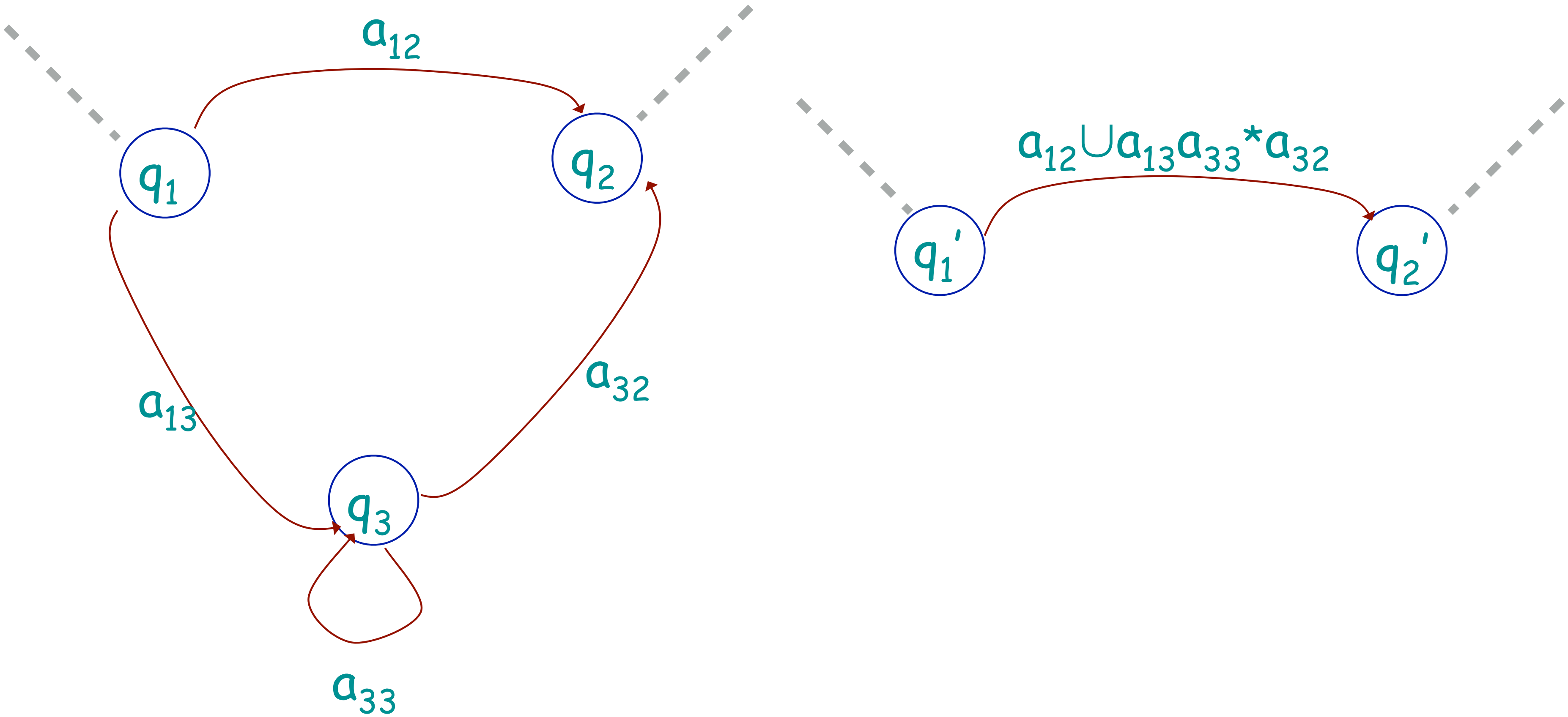
No step changed the strings accepted by the machine

Converting a GNFA to a RE

If the GNFA has exactly two states, then the label connecting the states is the RE

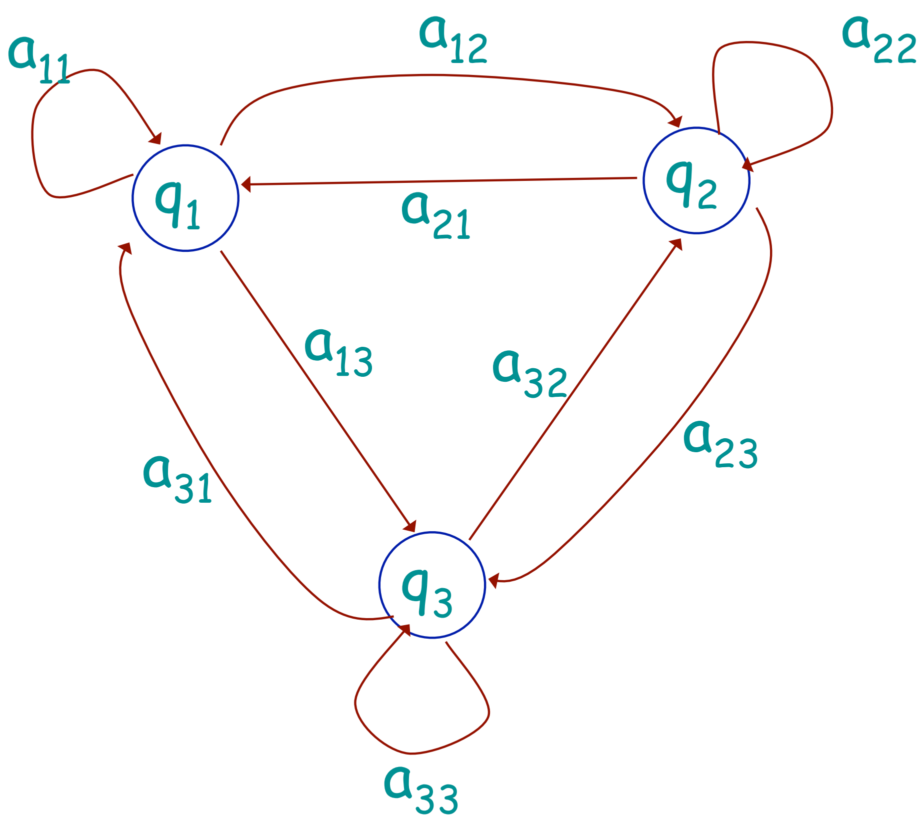
Otherwise, *remove one state at a time* without changing the language accepted by the machine until the GNFA has two states

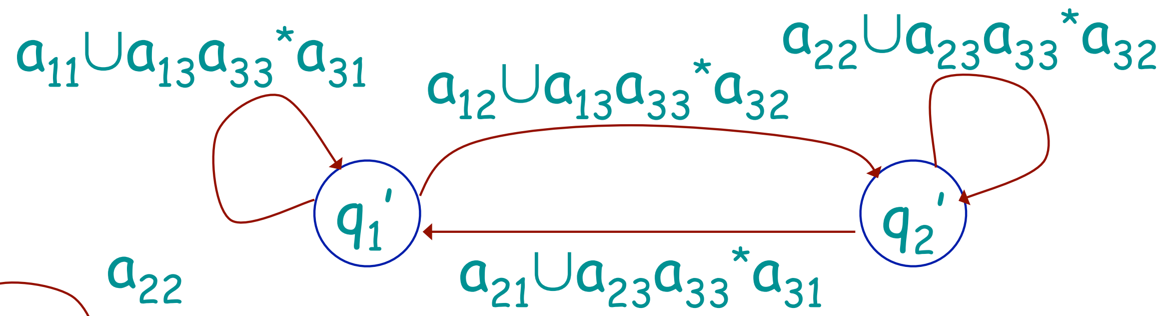
Removing One State From a GNFA



These two portions of GNFA's recognize the same strings

Accounting for Loops





Every DFA has a corresponding RE

Proof: Let M be any DFA and let w be any string in Σ^*. Convert M to G, a GNFA, then convert G to R, a regular expression.

Want to show $w \in L(M) \Leftrightarrow w \in L(R)$.

First show $w \in L(M) \Leftrightarrow w \in L(G)$.

Then show $w \in L(G) \Leftrightarrow w \in L(R)$.

$w \in L(M) \Rightarrow w \in L(G)$

Assume $w \in L(M)$ and $w = w_1 w_2 \ldots w_n$, where each $w_i \in \Sigma$. Then there is a sequence of states $q_1, q_2, \ldots, q_{n+1}$ such that

- $q_1 = q_0$
- $q_{n+1} \in F$
- $q_{i+1} = \delta(q_i, w_i)$ for each $i = 1, 2, \ldots, n$

When w is read by G, the sequence of states $q_s, q_1, q_2, \ldots, q_{n+1}, q_t$ would accept w

- $w \in L(G)$

$w \in L(M) \Leftarrow w \in L(G)$

Assume $w \in L(G)$ and $w = w_1 w_2 \ldots w_n$, where each $w_i \in \Sigma$. Then there is a sequence of states q_s, q_1, q_2, …, q_{n+1}, q_t such that

- $q_1 = q_0$
- $q_{n+1} \in F$
- $q_{i+1} = \delta(q_i, w_i)$ for each $i = 1, 2, \ldots, n$

When w is read by M, the sequence of states q_1, q_2, …, q_{n+1} would accept w

- $w \in L(M)$

$w \in L(G) \Leftrightarrow w \in L(R)$

Prove by induction on number of states in G

Base case: If G has 2 states then clearly $w \in L(G) \Leftrightarrow w \in L(R)$.

Induction step:

Assume $w \in L(G) \Leftrightarrow w \in L(R)$

for every G with k-1 states.

Prove $w \in L(G) \Leftrightarrow w \in L(R)$

for every G with k states.

$w \in L(R)$ if $w \in L(G)$

Assume $w \in L(G)$ and an accepting branch of the computation G enters on w is $q_s, q_1, q_2, \ldots, q_t$.

Let G' be the GNFA that results from removing one of G's states, q_{rip}.

There are two possibilities:

Case 1:

q_{rip} is never entered in the computation of w.

Then the same branch of computation exists in G'.

$w \in L(R)$ if $w \in L(G)$

Assume $w \in L(G)$ and an accepting branch of the computation G enters on w is $q_s, q_1, q_2, \ldots, q_t$.

Let G' be the GNFA that results from removing one of G's states, q_{rip}.

There are two possibilities:

Case 2: q_{rip} is entered in the computation of w (bracketed by q_i and q_j).

Then the new transition between q_i and q_j in G' describes the computation that could be done on the computation of w through the branch q_i, q_{rip}, q_j.

$w \in L(R)$ if $w \in L(G)$

Assume $w \in L(G)$ and an accepting branch of the computation G enters on w is $q_s, q_1, q_2, \ldots, q_t$.

Let G' be the GNFA that results from removing one of G's states, q_{rip}.

There are two possibilities:

Case 1:
q_{rip} is never entered in the computation of w.

Case 2: q_{rip} is entered in the computation of w.

So G' accepts w.

By induction, $w \in L(R)$.

$w \in L(G)$ if $w \in L(R)$

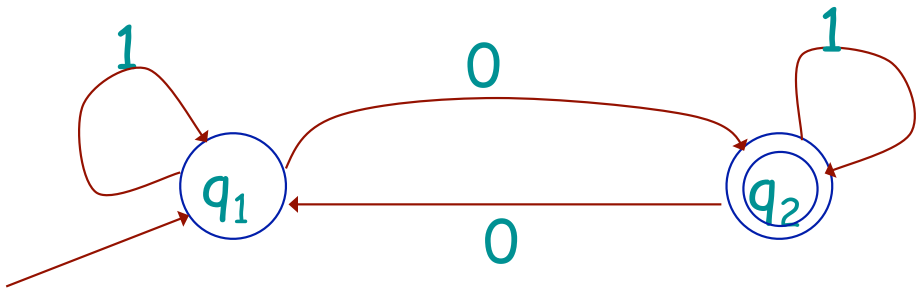
Assume $w \in L(R)$.

By induction hypothesis, $w \in L(G')$, the k-1 state GNFA resulting from removing one state from G.

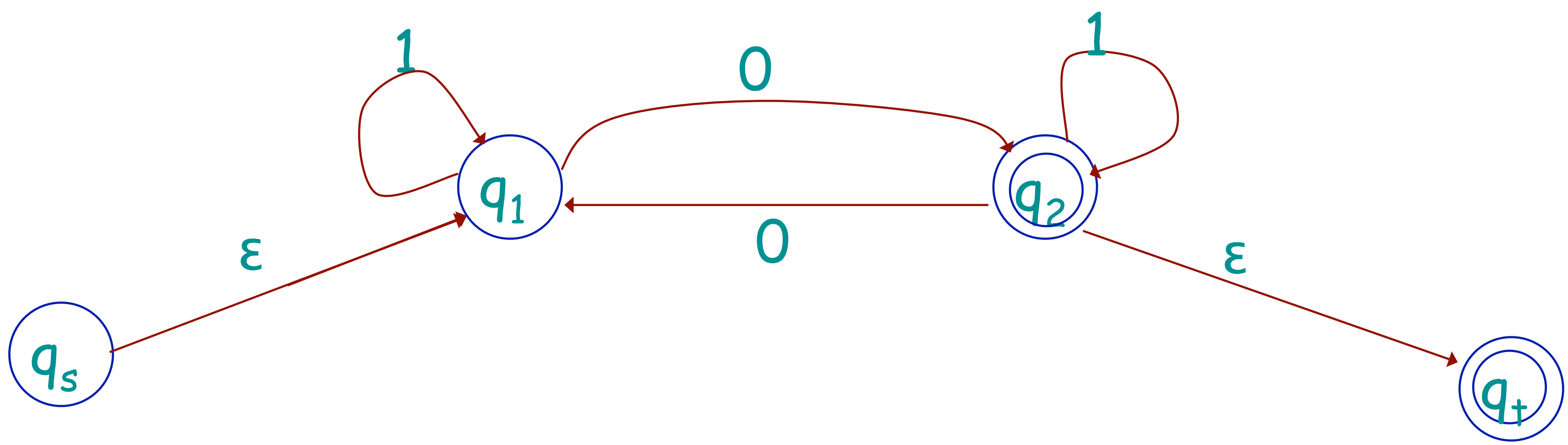
By construction, any computation in G' can also be done in G — possibly going through an extra state q_{rip}.

Therefore, $w \in L(G)$.

Example

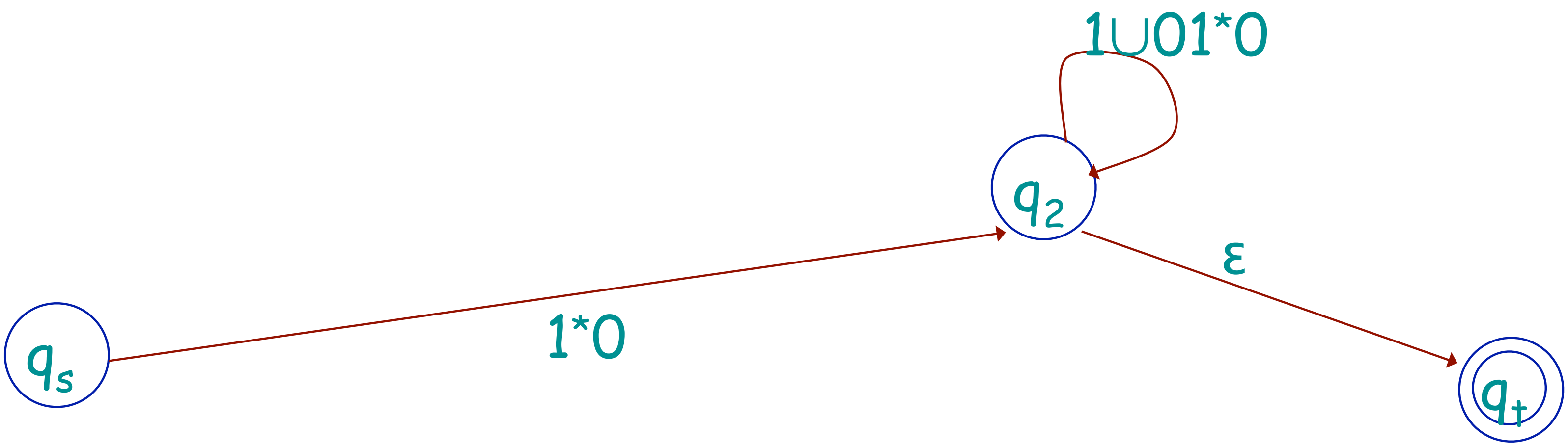


Example



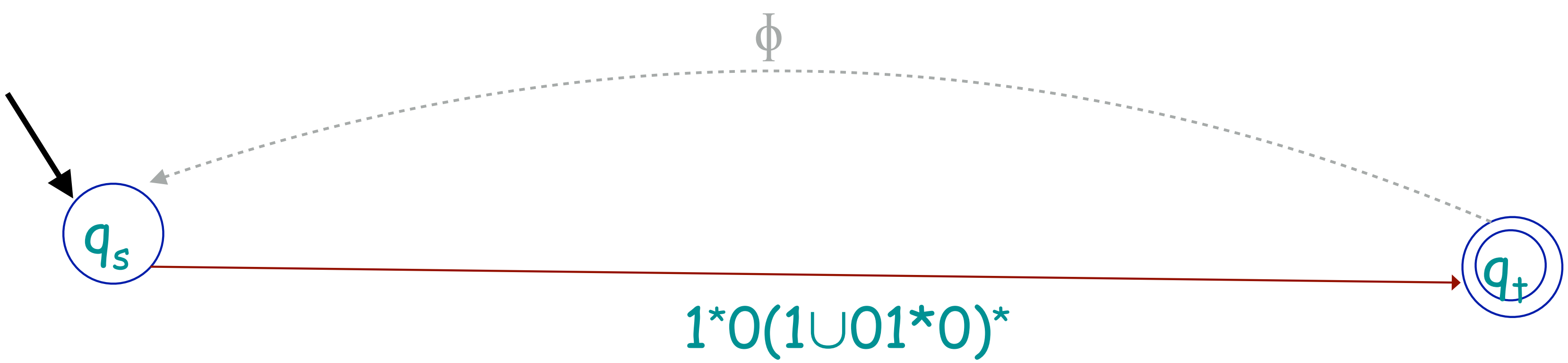
Step 1: Add two new states

Example



Step 2: Remove q_1

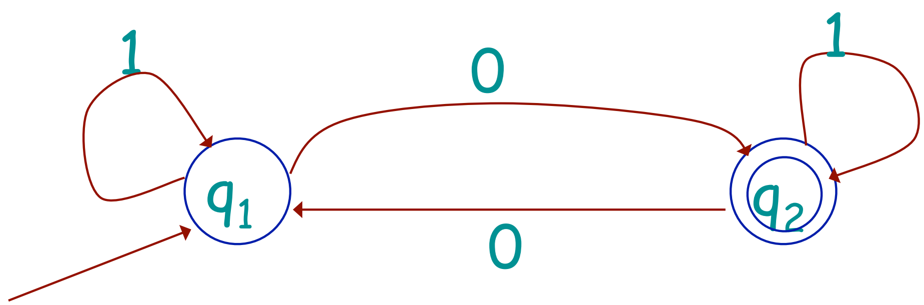
Example



Step 3: Remove q_2

Example

So this DFA



Is equivalent to the regular expression $1^*0(1\cup01^*0)^*$

$1^*0(1\cup01^*0)^*$

Regular Languages

We have explored several ways to identify regular languages

- Deterministic Finite Automata
- Nondeterministic Finite Automata
- Generalized Nondeterminstic Finite Automata
- Regular Grammars
- Regular Expressions

How can we tell that a language is not regular?