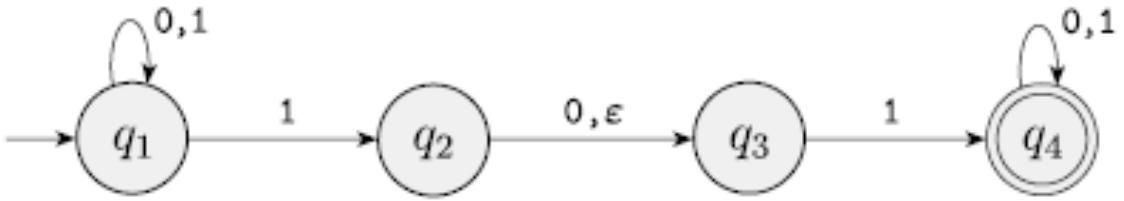
Nondeterministic Finite State Automata (NFA)



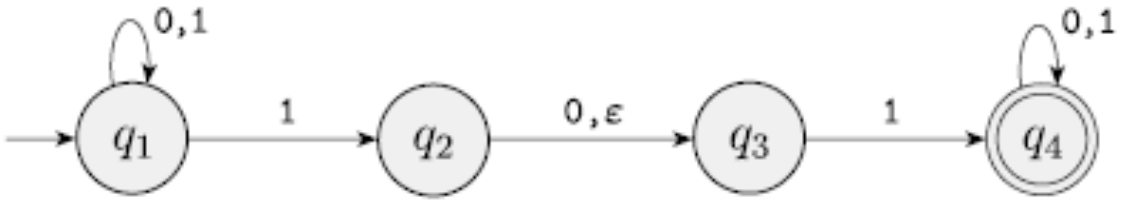
Every state of an NFA is allowed:

- outgoing ϵ transitions
- multiple outgoing transitions with the same label
- zero outgoing transitions for some labels

While processing the input string, the NFA can be in one of many different states

The NFA accepts the input string if one of its possible states is an accept state.

How an NFA Processes a String

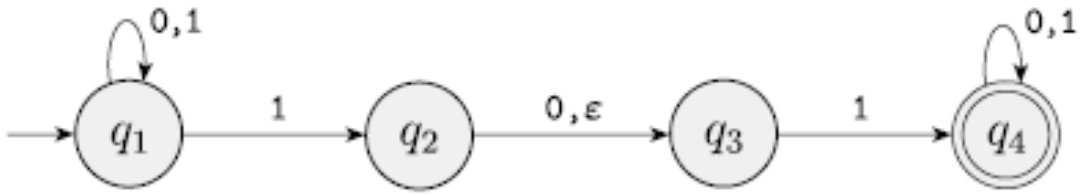


Machine initially in start state
From the start state, clones appear in all states reachable using only ϵ transitions.
Each clone in its current state:

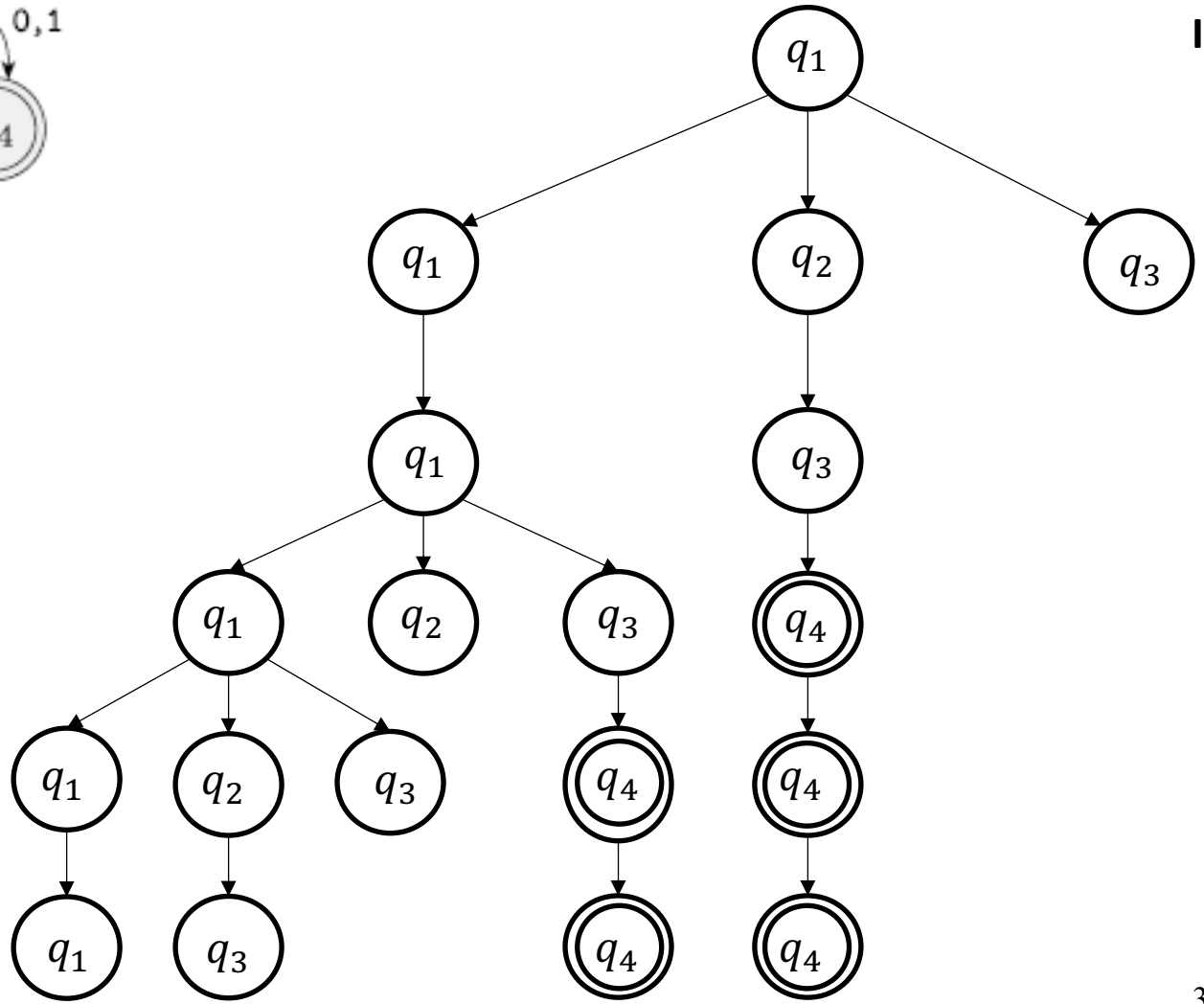
1. Reads the next input symbol a
2. If there is no outgoing transition labeled a : the clone dies
3. For each outgoing transition labeled a: a clone appears at the next state
4. Each clone spawns a clone at every state reachable from its state following only ϵ transitions.

First process the input symbol, then take all free moves thereafter.

NFA Computations



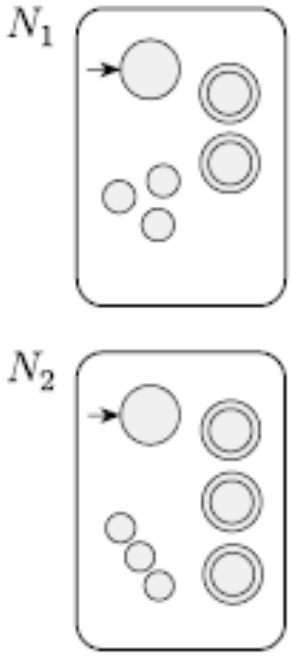
Input string 10110

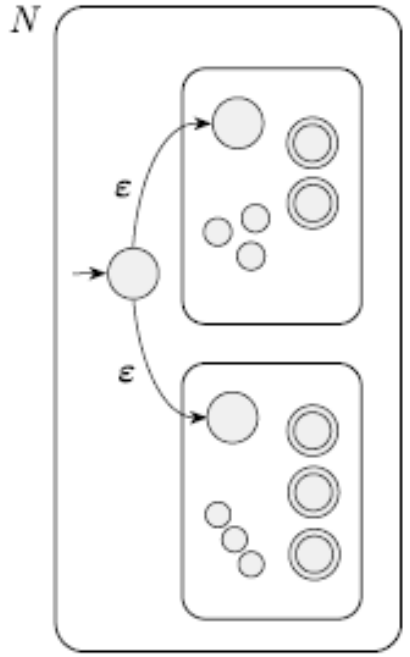


ACCEPTED!

Designing NFAs

Given NFAs N_1, N_2 for A, B

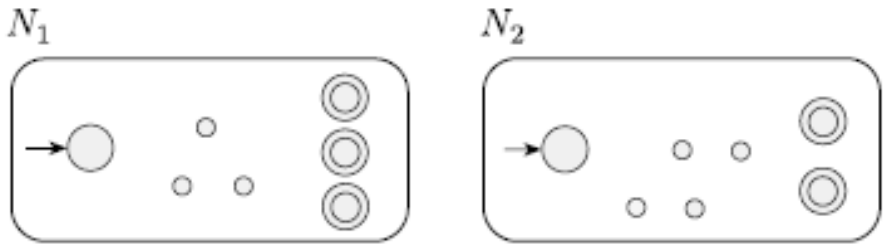


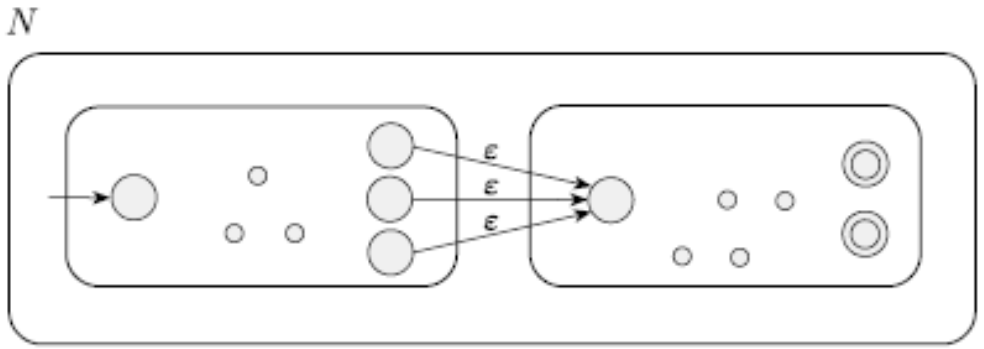


NFA for $A \cup B$

Designing NFAs

Given NFAs N_1, N_2 for A, B

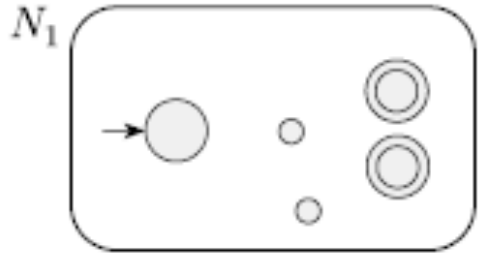


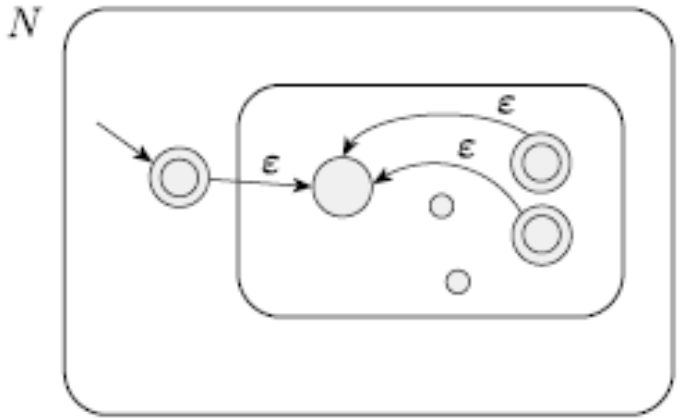


NFA for $A \circ B$

Designing NFAs

Given NFA N_1 for A





NFA for A^*

Are We There Yet?

We've shown is that NFA recognizable languages are closed under regular operations.

……. BUT

We had asked if regular languages are closed under regular operations.

So, what's the connection between regular languages and NFA-recognizable languages?

Can we use NFAs to design DFAs?

Getting Formal

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

Q is the finite set of states

Σ is the finite alphabet of symbols

$\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \wp(Q)$ is the transition ***function***

q_0 is the start state

$F \subseteq Q$ is the set of accept states

Notation: $\wp(Q)$ is the powerset (set of all subsets) of Q

Formal Definition 2

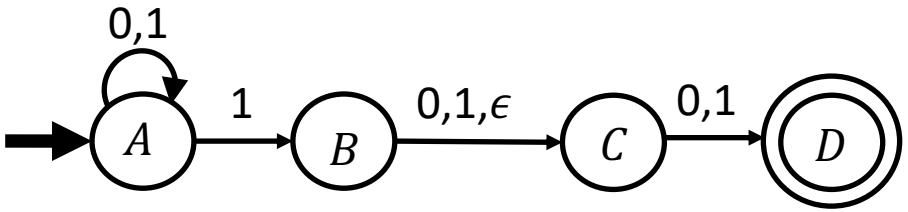
A nondeterministic finite automaton $N = (Q, \Sigma, \delta, q_0, F)$ ***accepts*** a string w if

i. w can be expressed as $w = y_1 y_2 \ldots y_n$, $\forall i: y_i \in \Sigma \cup \{\epsilon\}$, and

ii. there is a sequence $r_0, r_1, \ldots, r_n$ of states in Q such that:

1. $r_0 = q_0$, (start in the start state)
2. $r_{i+1} \in \delta(r_i, y_{i+1})$ for $0 \leq i < n$, and (every transition is legal)
3. $r_n \in F$. (the final state is an accept state)

Example NFA N



What is the language $L(N)$? $\{w: w \text{ has at least 2 symbols and either the second last or third last symbol is } 1\}$

$N = (Q, \Sigma, \delta, q_0, F)$ where:

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

$\Sigma = \{0,1\}$

$q_0 = A$

$F = \{D\}$

$\delta(A, 0) = \{A\}$ $\quad \delta(C, 0) = \{D\}$

$\delta(A, 1) = \{A, B\}$ $\quad \delta(C, 1) = \{D\}$

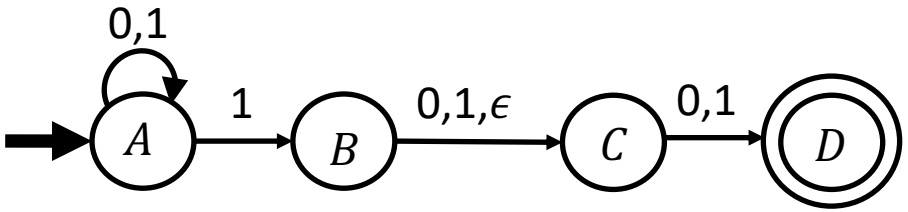
$\delta(A, \epsilon) = \Phi$ $\quad \delta(C, \epsilon) = \Phi$

$\delta(B, 0) = \{C\}$ $\quad \delta(D, 0) = \Phi$

$\delta(B, 1) = \{C\}$ $\quad \delta(D, 1) = \Phi$

$\delta(B, \epsilon) = \{C\}$ $\quad \delta(D, \epsilon) = \Phi$

Designing a DFA Equivalent to NFA N



$N = (Q, \Sigma, \delta, q_0, F)$ where:

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

$\Sigma = \{0,1\}$

$q_0 = A$

$F = \{D\}$

$\delta(A, 0) = \{A\}$
$\delta(A, 1) = \{A, B\}$
$\delta(A, \epsilon) = \Phi$
$\delta(B, 0) = \{C\}$
$\delta(B, 1) = \{C\}$
$\delta(B, \epsilon) = \{C\}$

$\delta(C, 0) = \{D\}$
$\delta(C, 1) = \{D\}$
$\delta(C, \epsilon) = \Phi$
$\delta(D, 0) = \Phi$
$\delta(D, 1) = \Phi$
$\delta(D, \epsilon) = \Phi$

$M = (Q', \Sigma', \delta', q_0', F')$ where

$$\Sigma' = \Sigma$$

What states?

Hint: Where are all the clones?

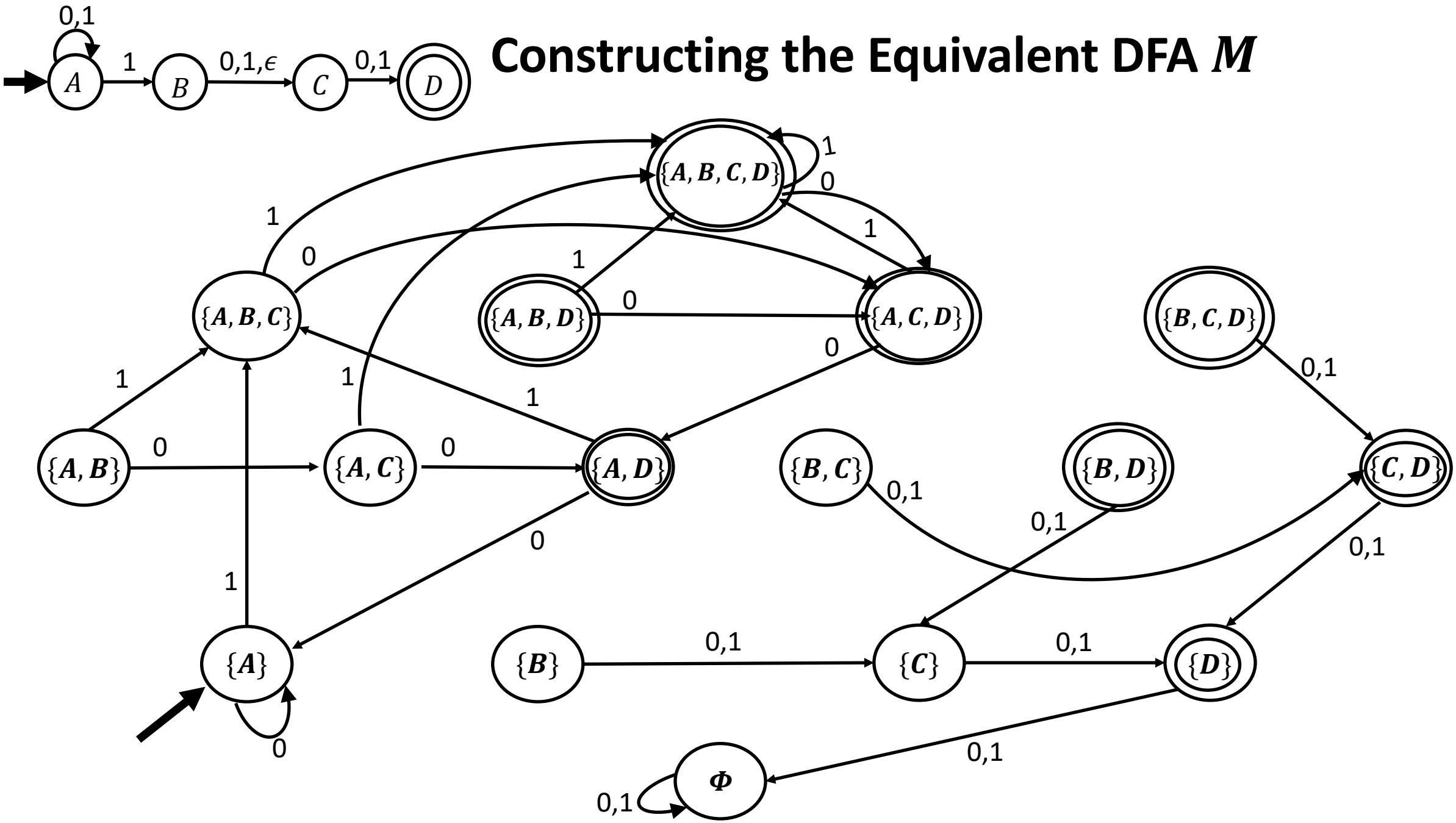
So, make one state in M for each subset of Q:

$$Q' = \wp(Q)$$

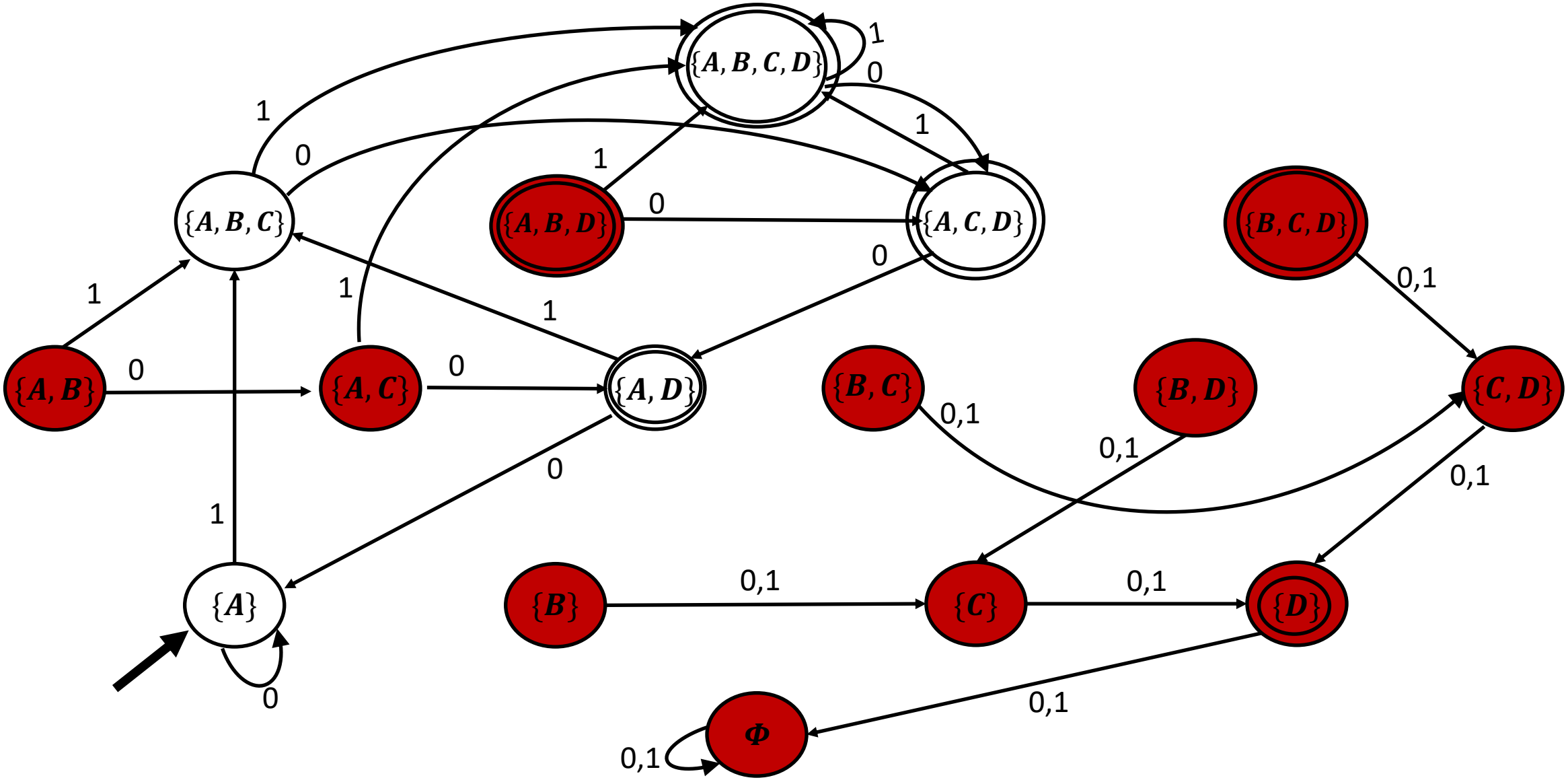
$$q_0' = \{A\}$$

$$F' = \{R \in Q' : R \cap F \neq \Phi\}$$

Constructing the Equivalent DFA M



Removing Redundant States of M



Equivalence of NFA and DFAs

Theorem 1.39. Every NFA has an equivalent DFA.

Proof Sketch: Let NFA $N = (Q, \Sigma, \delta, q_0, F)$.

Define DFA $M = (\wp(Q), \Sigma, \delta', q_0', F')$

$F' = \{R \subseteq Q: R \cap F \neq \Phi\}$ all subsets that contain a final state of N.

From state $r \in Q, a \in \Sigma$:

$\delta(r, a) =$ set of states in N directly following an a-transition from r

$\delta'(R, a) =$ set of states in N reached by following an a-transition from any state in $R \subseteq Q$

$= \bigcup_{r \in R} \delta(r, a)$

But what about freebies: States reached by ϵ-transitions following the a-transition?

Equivalence of NFA and DFAs … 2

Define:

$E(R) = \{q \in N: q$ is reachable from $r \in R$ using a sequence of 0 or more ϵ-transitions$\}$

Now, $\delta'(R, a) = E(\bigcup_{r \in R} \delta(r, a))$

$= \bigcup_{r \in R} E(\delta(r, a)$

Finally, $q_0' = E(\{q_0\})$

Putting it All Together

Definition: Regular Languages are those recognized by DFAs.

Obvious Fact: Every DFA is also an NFA

Last Week: If A, B are each recognized by an NFA then so are $A \cup B, A \circ B, A^*$

Last lecture: every NFA has an equivalent DFA

Therefore, if A, B are each recognized by a DFA then so are $A \cup B, A \circ B, A^*$

In other words, regular languages are closed under regular operations

Moral: Think NFA!

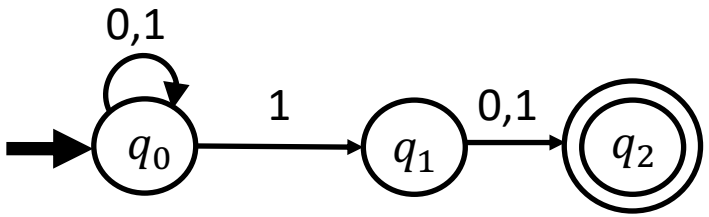
Question: So, is there any sense in which NFAs more powerful than DFAs?

We simulated a k-state NFA by a 2^k-state DFA. Can we do better?

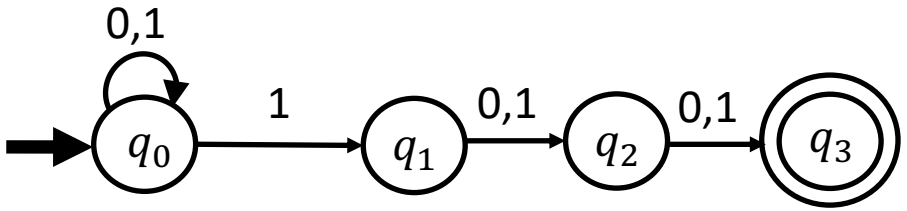
Is an exponential blowup necessary?

Consider the language $L_k = \{w: length\ of\ w \geq k\ and\ the\ k^{th}\ last\ symbol\ is\ 1\}$

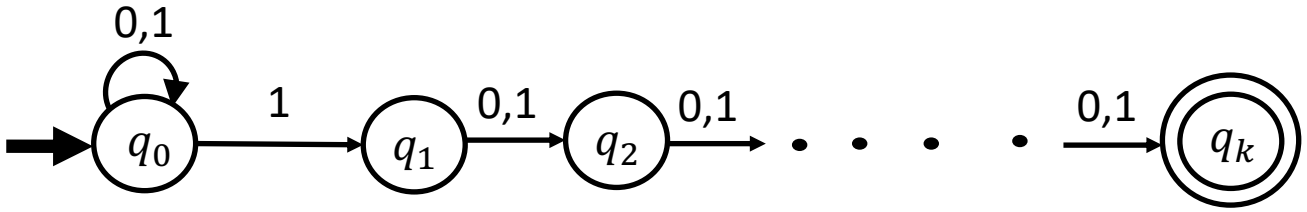
3-state NFA for L_2



4-state NFA for L_3



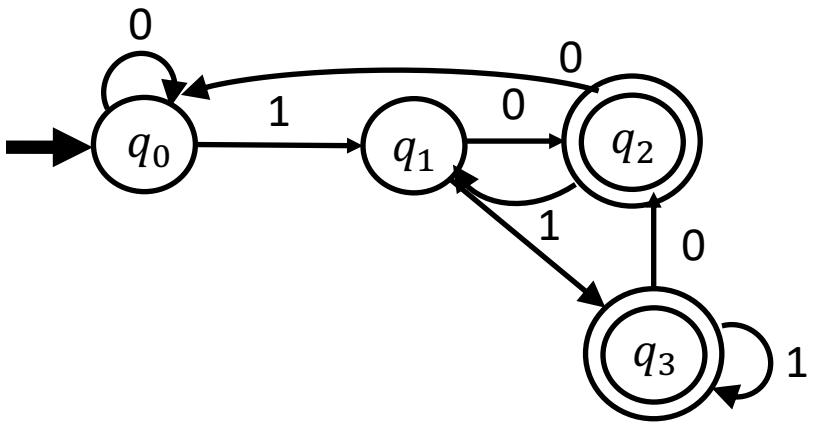
$(k+1)$-state NFA for L_k



How many states must a DFA for L_k have?

DFAs for L_k

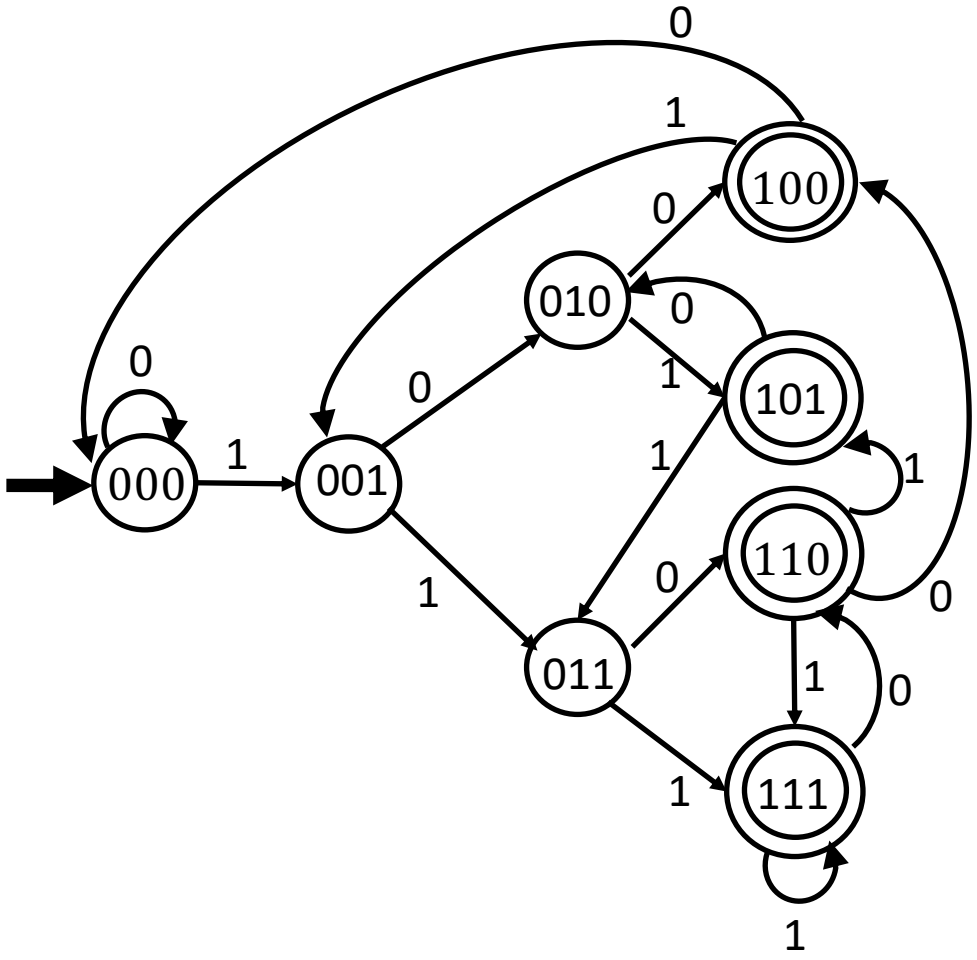
4-state DFA for L_2



Each state tracks the last two symbols: $q_0\ (00)$; $q_1\ (01)$; $q_2\ (10)$; $q_3\ (11)$

DFA for L_3

Each state tracks the last 3 bits of the input



The DFAs Grow Exponentially!

This strategy uses 8 states for L_3
16 states for L_4
…
2^k states for L_k

The NFA for L_k needed only $k + 1$ states while this DFA needs 2^k states!

Is there a better method that requires fewer states for a DFA?
Question: do we ***have to*** remember each of the last k bits of the input?

Intuition: If the i^{th} last bit seen thus far eventually turns out to be the k^{th} last bit of the entire input string, we must remember it – otherwise we won't know whether or not to accept the input!

Every DFA for L_k has $\geq 2^k$ States!

Theorem. Every DFA for L_k has $\geq 2^k$ states.

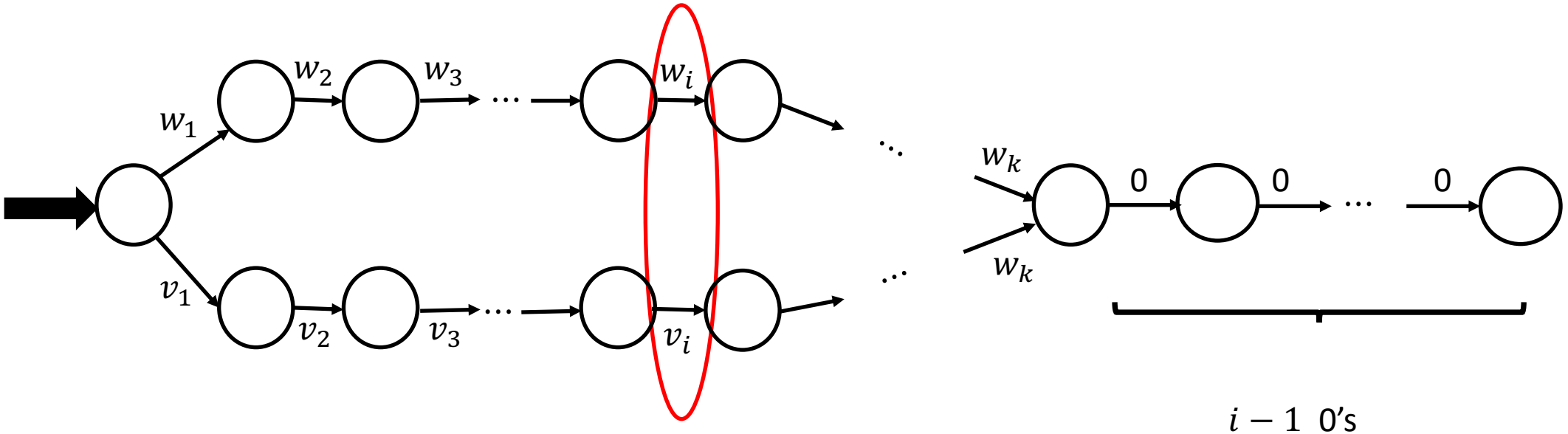
Proof: (by contradiction).

- Suppose that DFA M with fewer states recognizes L_k.
- There are 2^k strings of length k, but M has strictly less than 2^k states.
- By the pigeonhole principle, $\exists w, v$ both of length k that end up in the same state of M.
- Let i be the last (rightmost) bit that strings w, v differ in.

$$w = w_1 w_2 \cdots w_i w_{i+1} \cdots w_k$$

$$v = \quad v_1 v_2 \cdots v_i w_{i+1} \cdots w_k$$

No DFA with less than 2^k states for L_k



$w' = w0^{i-1}$ and $v' = v0^{i-1}$ both end in the same final state. Both are either accepted or both are rejected.

But w', v' differ in the k^{th} last bit. So one is in L_k and the other is not.

This means the DFA with less than 2^k states does not recognize L_k.