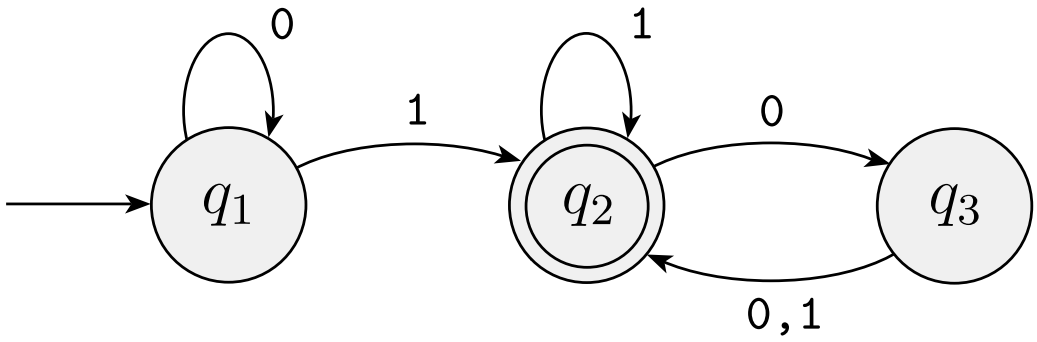
Nondeterministic Finite Automata

Mika Göös

EPFL School of Computer and Communication Sciences

Lecture 2



RECALL: DFAs, REGULAR LANGUAGES

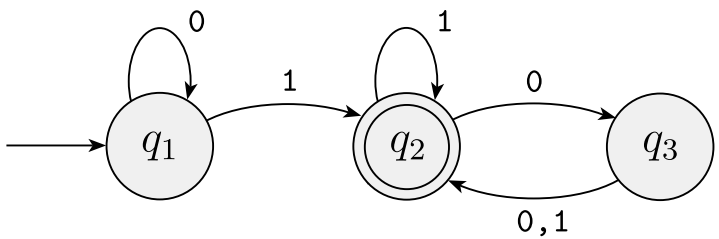
Definitions

A deterministic finite automaton (DFA) M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set called the states,
- Σ is a finite set called the alphabet,
- $\delta : Q \times \Sigma \to Q$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accept states. *(allow $F = \emptyset$)*

- $Q = \{q_1, q_2, q_3\}$,
- $\Sigma = \{0, 1\}$,
- δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,



- q_1 is the start state, and
- $F = \{q_2\}$.

Definitions

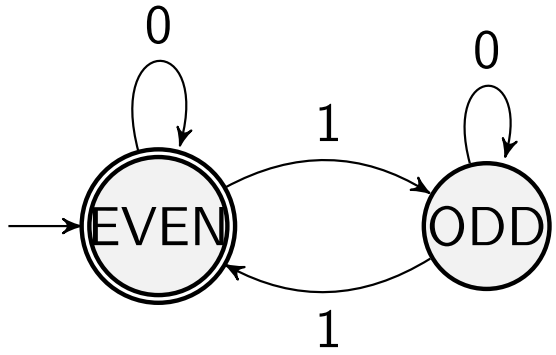
A deterministic finite automaton (DFA) M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set called the states,
- Σ is a finite set called the alphabet,
- $\delta : Q \times \Sigma \to Q$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accept states. *(allow $F = \emptyset$)*

- Σ^* — set of all finite strings composed of symbols from Σ
  - Includes the empty string ε
- $L(M) \subseteq \Sigma^*$ set of strings accepted by M. We say that M recognizes the language $L(M)$.
- $A \subseteq \Sigma^*$ is a regular language if there is a DFA M s.t. $A = L(M)$.

Is $L = \{w \in \{0,1\}^* : w \text{ contains an even number of 1's}\}$ a regular language?

YES because it is recognized by the following DFA



Formally, we prove correctness, i.e., that a DFA accepts a certain language, via induction on the length of the string. It is a good idea to choose a strong induction hypothesis that classifies which strings leads to which states instead of only talking about accepting states

NEW LANGUAGES FROM OLD

- Complement
  - $\bar{L} = \{w \in \Sigma^* : w \text{ is } \mathbf{not} \text{ in } L\}$
- Union
  - $L_1 \cup L_2 = \{w \in \Sigma^* : w \in L_1 \ \mathbf{or}\ w \in L_2\}$
- Intersection
  - $L_1 \cap L_2 = \{w \in \Sigma^* : w \in L_1 \ \mathbf{and}\ w \in L_2\}$
- Concatenation
  - $L_1 \circ L_2 = \{w \in \Sigma^* : w = w_1.w_2,\ w_1 \in L_1 \ \mathbf{and}\ w_2 \in L_2\}$

Complement

If L is regular is its complement $\bar{L} = \{w \in \Sigma^* : w \text{ is } \textbf{not} \text{ in } L\}$ regular?

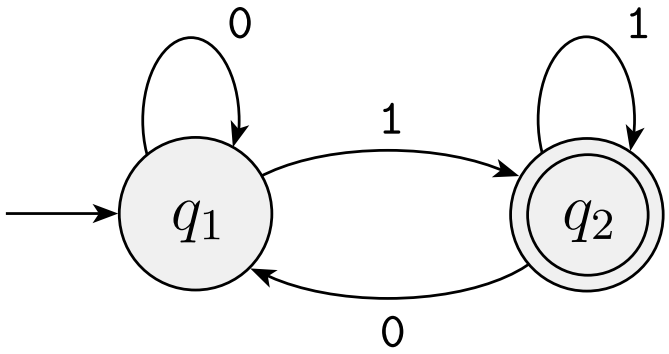
- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that accepts L
- Let $M' = (Q, \Sigma, \delta, q_0, \bar{F} = Q \setminus F)$ be the DFA where accepting states are complemented

- If $w \in L(M)$ then $w \notin L(M')$
- If $w \notin L(M)$ then $w \in L(M')$

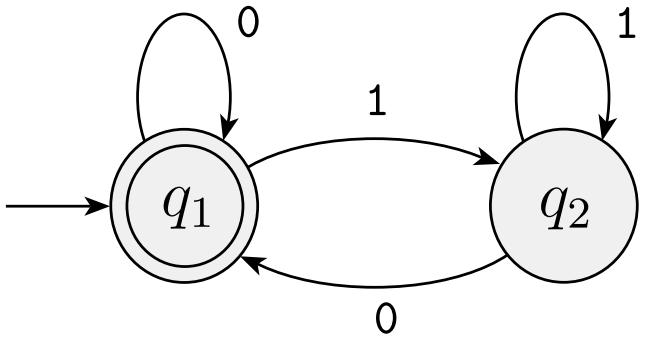
Theorem: $L(M') = \bar{L}$

Hence complement of a regular language is regular

Example



is the complement of



Union

- $L_1 = L(M_1), M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$
- $L_2 = L(M_2), M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

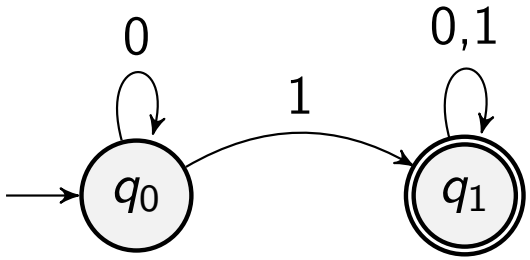
If M_1's alphabet Σ_1 is different from M_2's alphabet Σ_2 then first extend them to the alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$ before taking the union

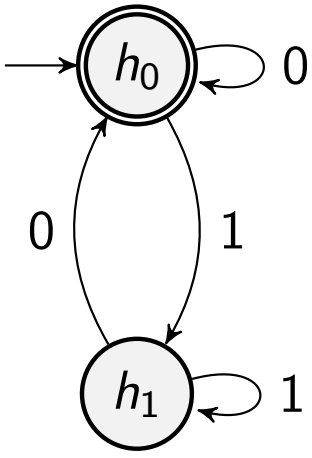
The union is recognized by the DFA $M = (Q, \Sigma, \delta, q_0, F)$ where

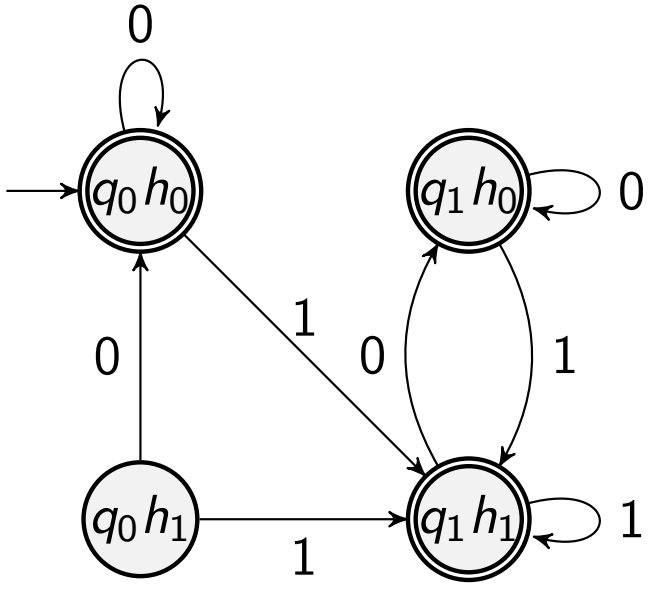
- $Q = Q_1 \times Q_2$
- $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- $q_0 = (q_1, q_2)$
- $F = \{(r_1, r_2) : r_1 \in F_1 \textbf{ or } r_2 \in F_2\}$

Run M_1 and M_2 in parallel and accept if one of them does

Example







The union

Intersection

- $L_1 = L(M_1), M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$
- $L_2 = L(M_2), M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$

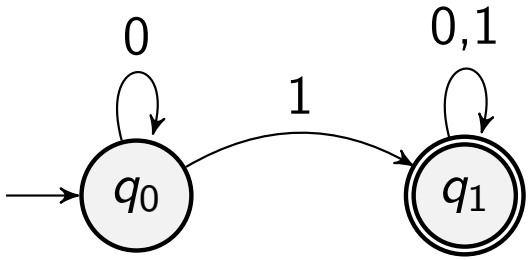
If M_1's alphabet Σ_1 is different from M_2's alphabet Σ_2 then first extend them to the alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$ before taking the intersection

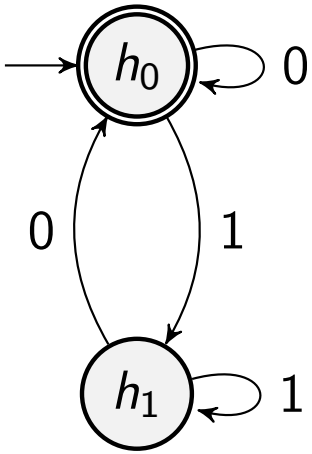
The union is recognized by the DFA $M = (Q, \Sigma, \delta, q_0, F)$ where

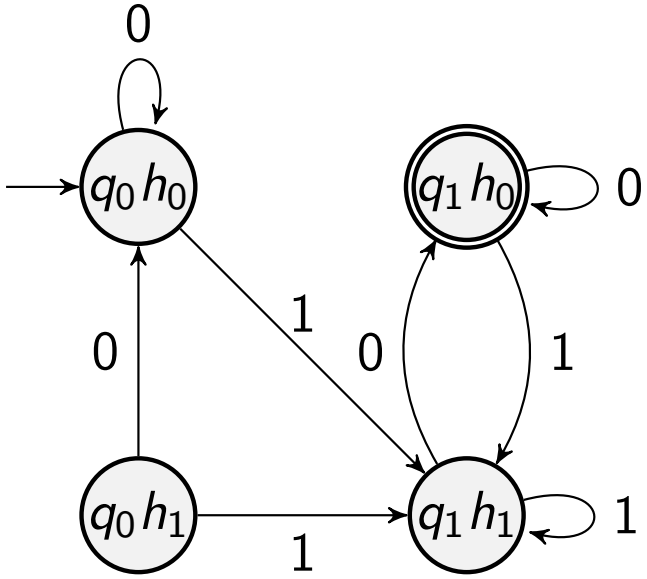
- $Q = Q_1 \times Q_2$
- $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- $q_0 = (q_1, q_2)$
- $F = \{(r_1, r_2) : r_1 \in F_1 \textbf{ and } r_2 \in F_2\}$

Run M_1 and M_2 in parallel and accept if both of them do

Example





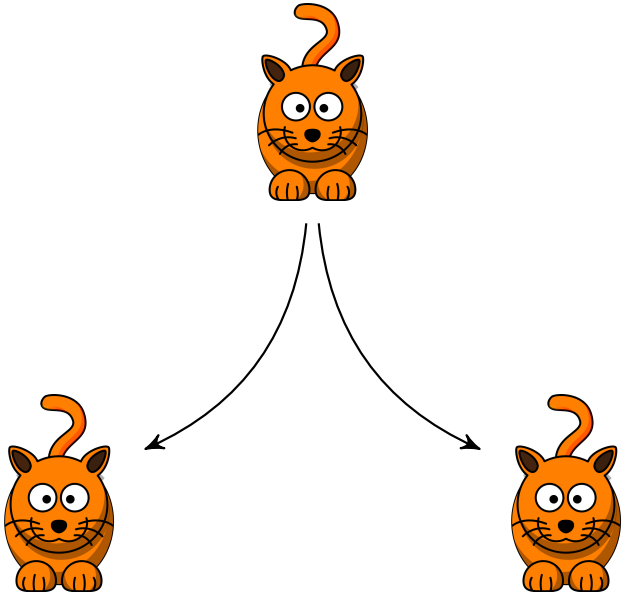


The intersection

Concatenation?

$$L_1 \circ L_2 = \{w \in \Sigma^* : w = w_1.w_2,\ w_1 \in L_1 \textbf{ and } w_2 \in L_2\}$$

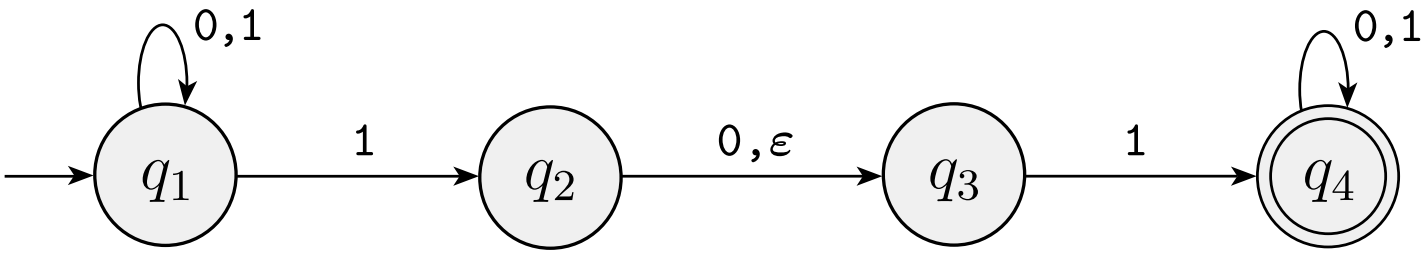
Difficulty to decide whether a string w is in $L_1 \circ L_2$ is that we need to "guess" where to break it into w_1 and w_2



Should I stay or should I go (left or right)

NONDETERMINISM

Nondeterminism vs Determinism



State diagram of a **Nondeterministic Finite Automaton (NFA)**

Differences to DFAs:

1. Ability to transition to more than one state on a given symbol
   Ex: q_1 has two outgoing transitions for symbol 1
2. A state may have no transition on a particular symbol
   Ex: q_3 has no outgoing transitions for symbol 0
3. Ability to take a step without reading any input symbol (ε-transitions)
   Ex: q_2 can transition to q_3 without reading a symbol

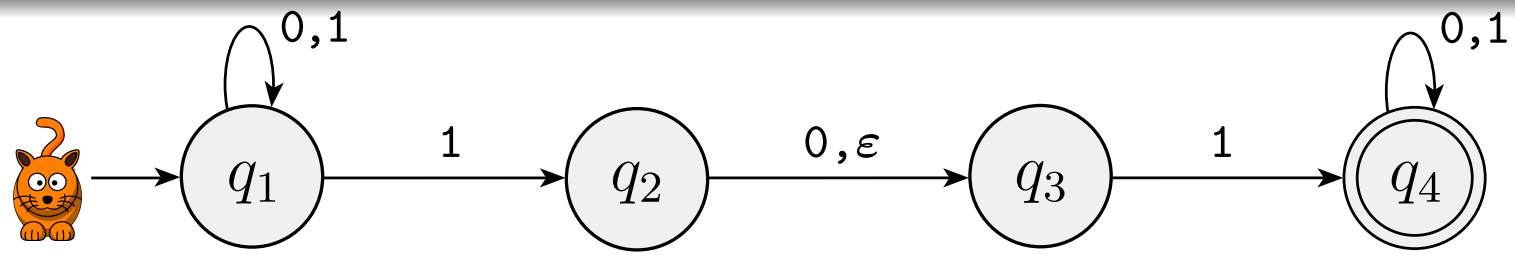
How does an NFA Compute?

Two different but equivalent viewpoints:

- It is a parallel computer that makes multiple copies of itself and follows *all* possibilities in parallel. If *any one* of these copies is in an accept state at the end, the NFA accepts the input string

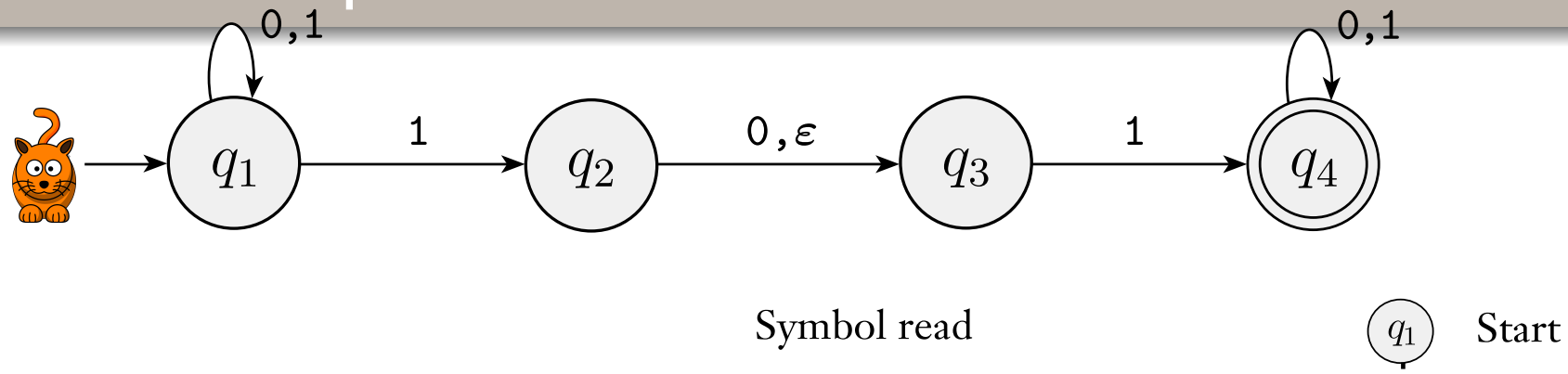
- It is a computer that accepts if there exist guesses/choices that makes it accept

Parallel Viewpoint



Input: 010110

Parallel Viewpoint

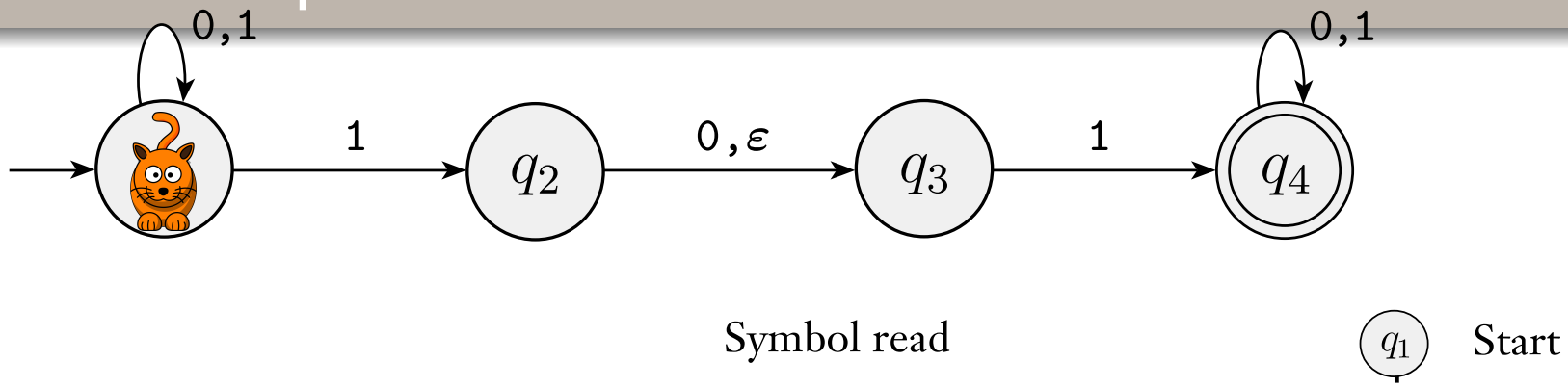


Input: 010110

Let's start

- cat moves to initial state q_1

Parallel Viewpoint

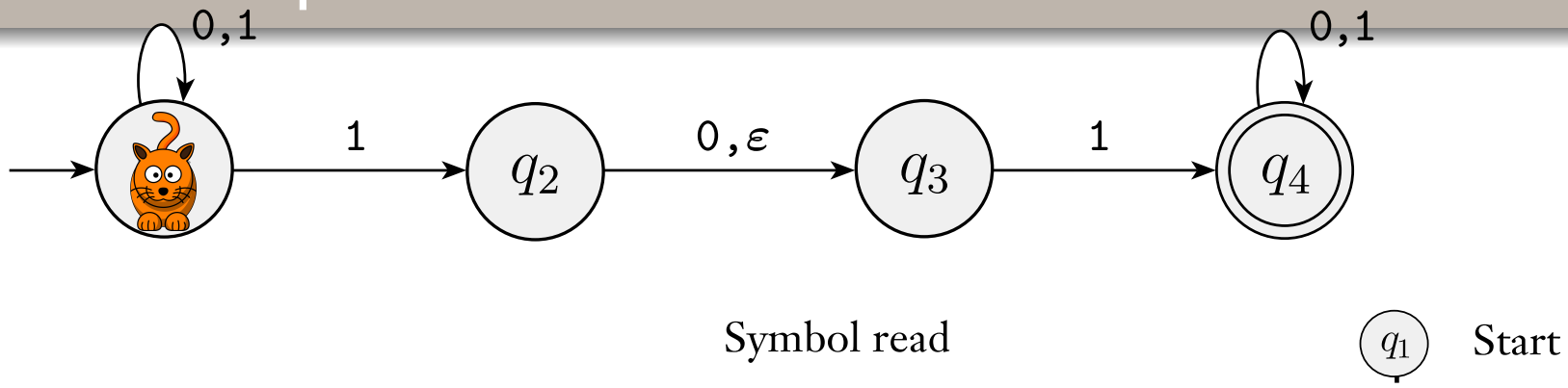


Input: 010110

Let's start

- cat moves to initial state q_1

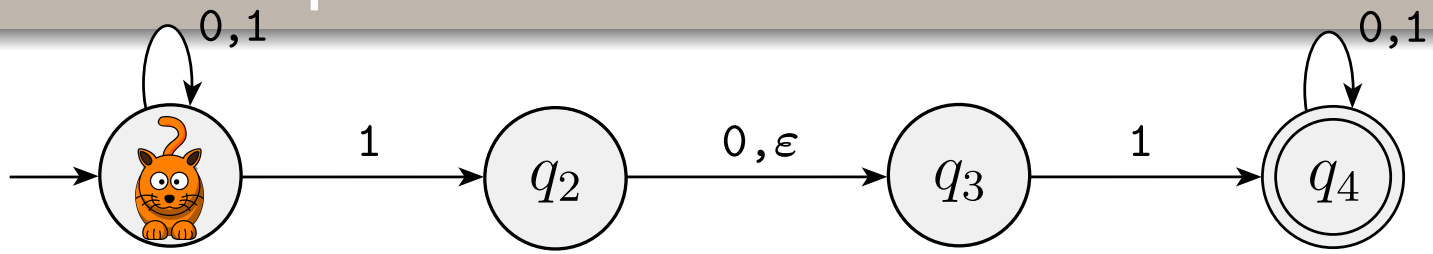
Parallel Viewpoint

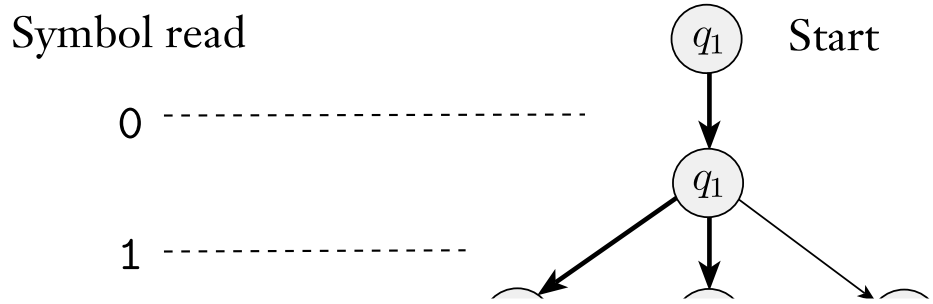


Input: 010110
▲

Symbol read 0

Parallel Viewpoint



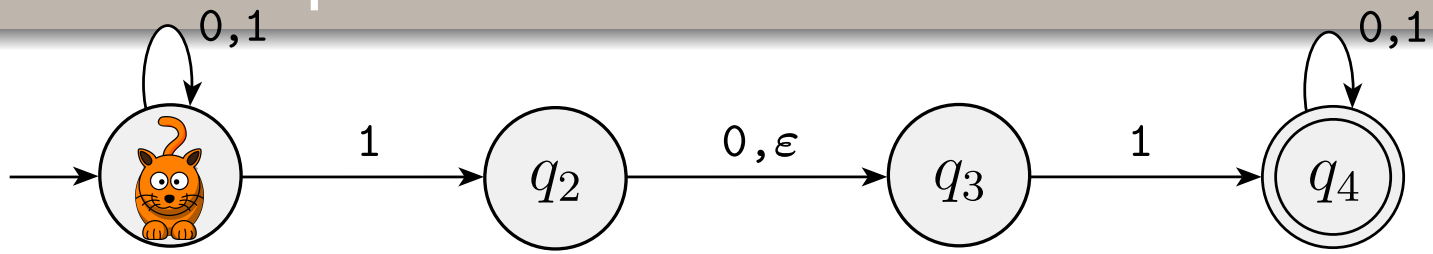


Input: 010110
▲

Symbol read 0

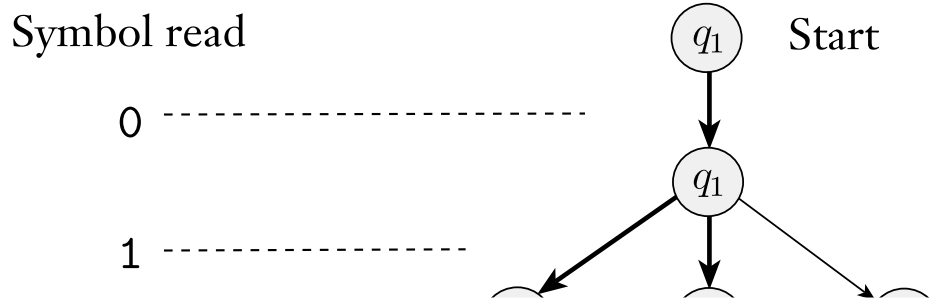
- cat stays in state q_1

Parallel Viewpoint

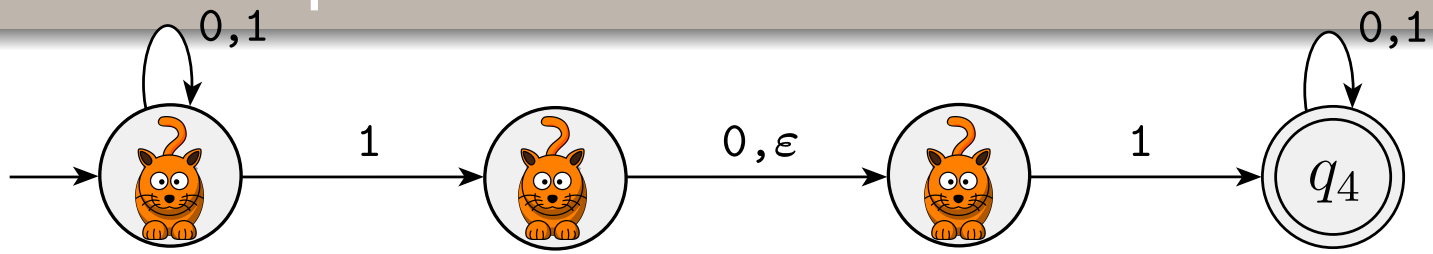


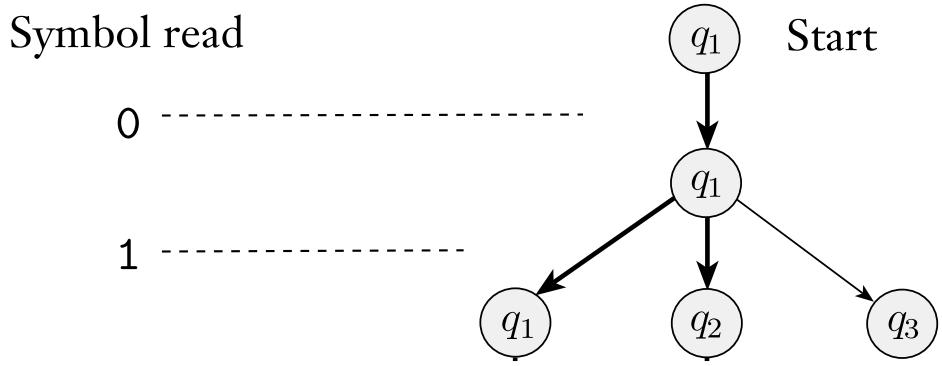
Input: 010110

Symbol read 1



Parallel Viewpoint



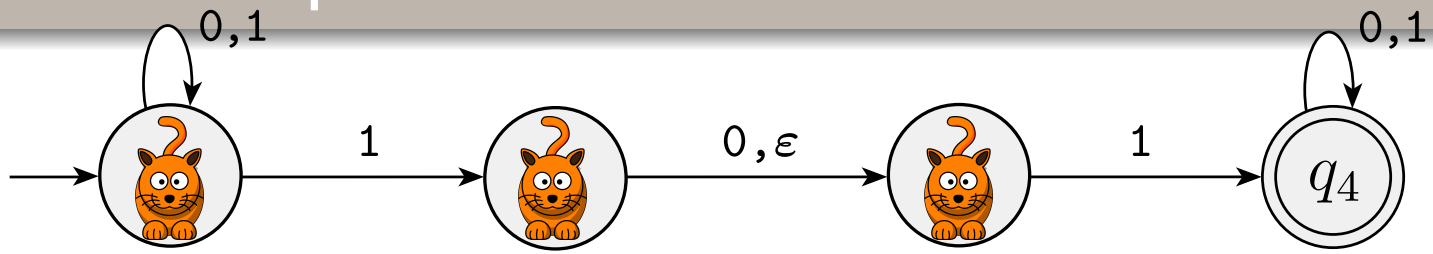


Input: 010110
▲

Symbol read 1

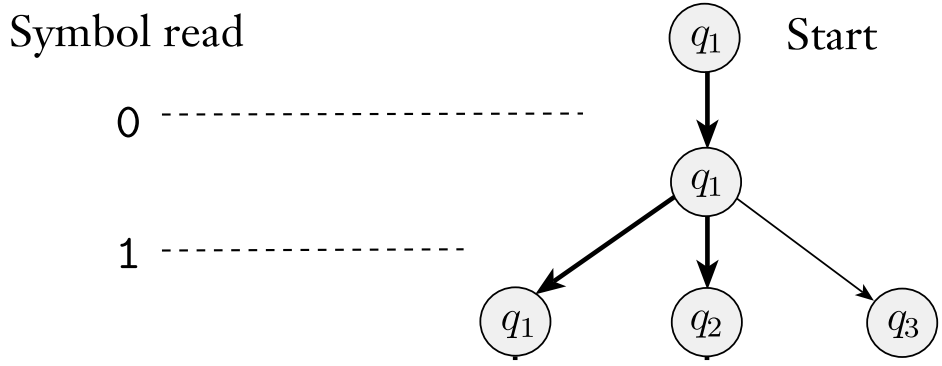
- cat can (i) stay in state q_1, (ii) move to q_2, and (iii) move to q_2 and then to q_3 via the ε-transition
- We make a copy of the process (cat) for each of the three possibilities that we then run in parallel

Parallel Viewpoint

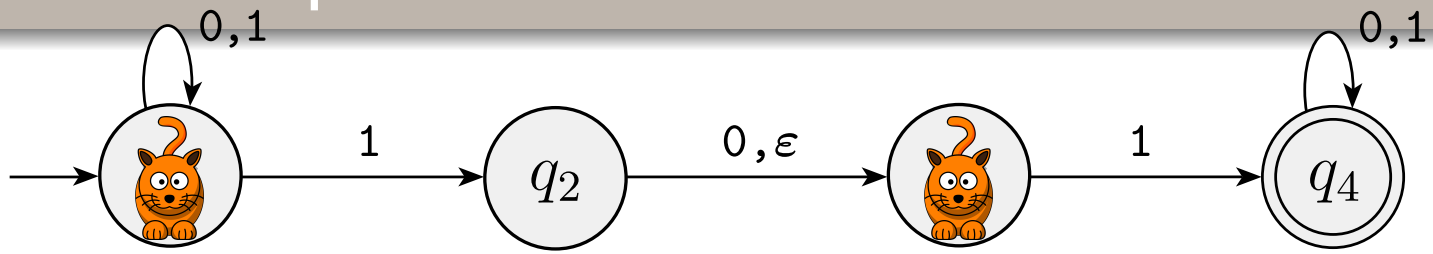


Input: 010110

Symbol read 0



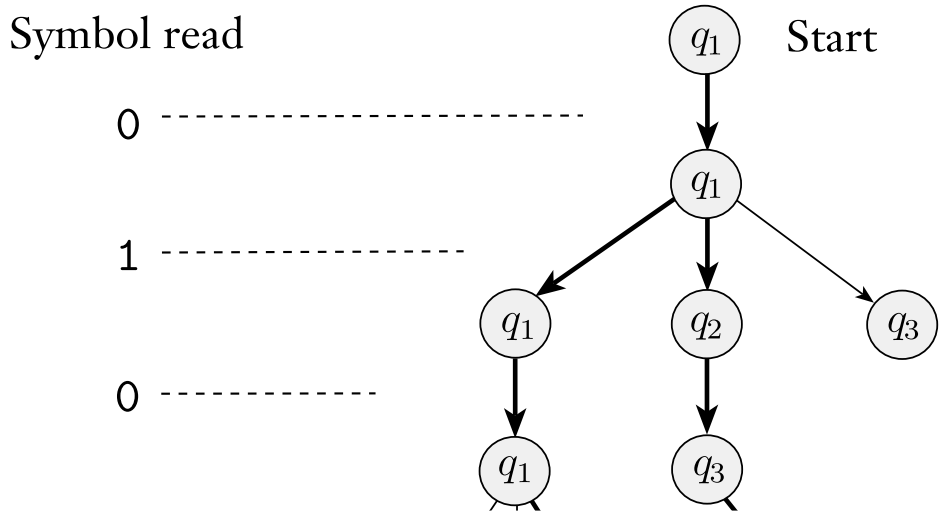
Parallel Viewpoint



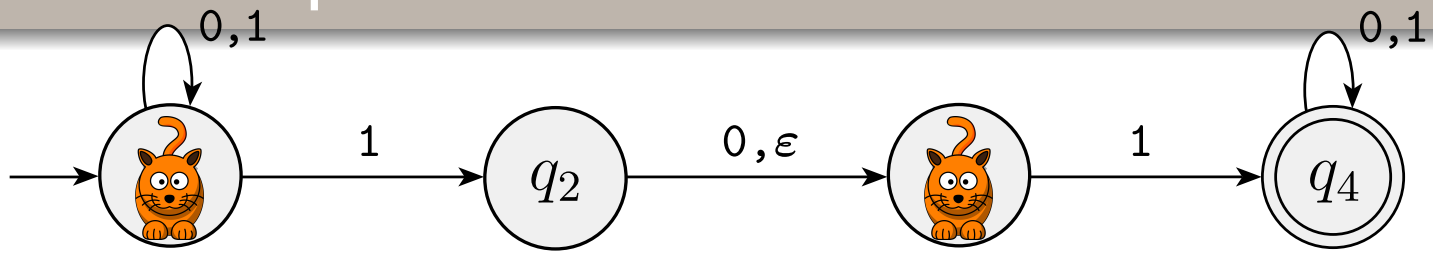
Input: 010110

Symbol read 0

- $q_1 \to q_1$ and $q_2 \to q_3$
- The process/cat previously at q_3 dies since there is no valid 0-transition

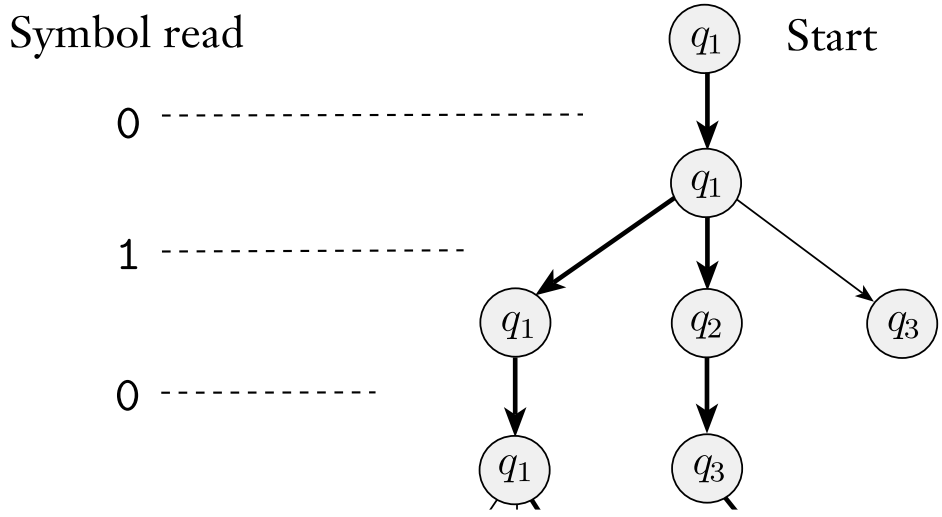


Parallel Viewpoint

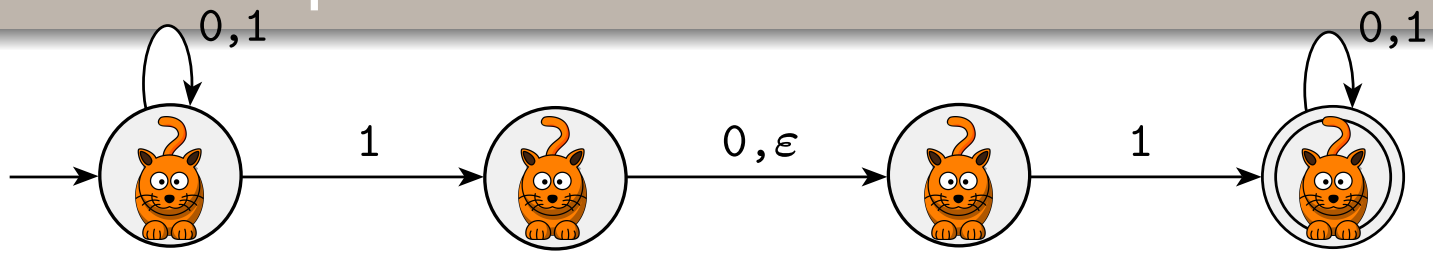


Input: 010110

Symbol read 1



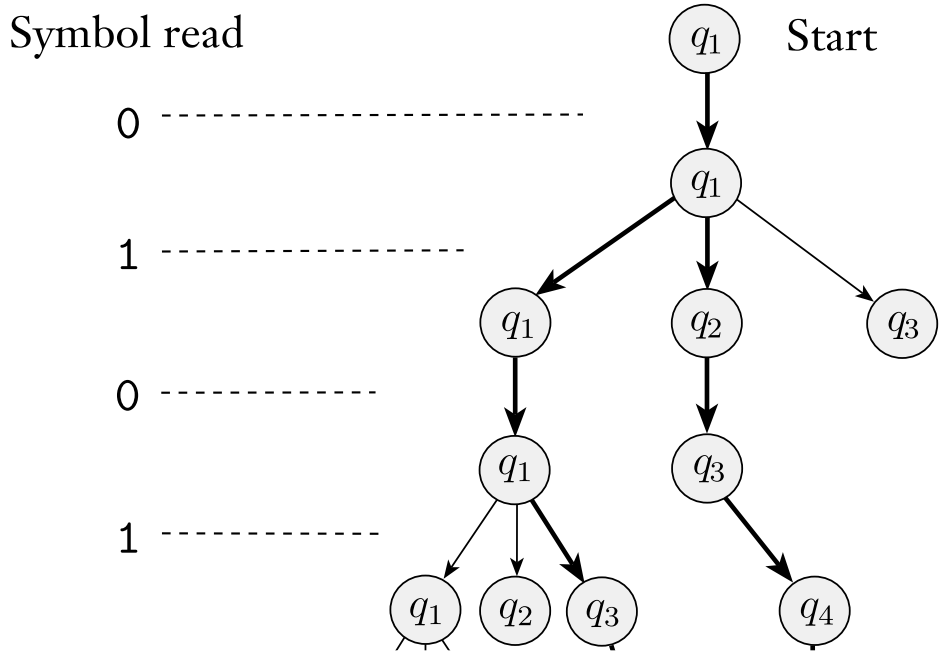
Parallel Viewpoint



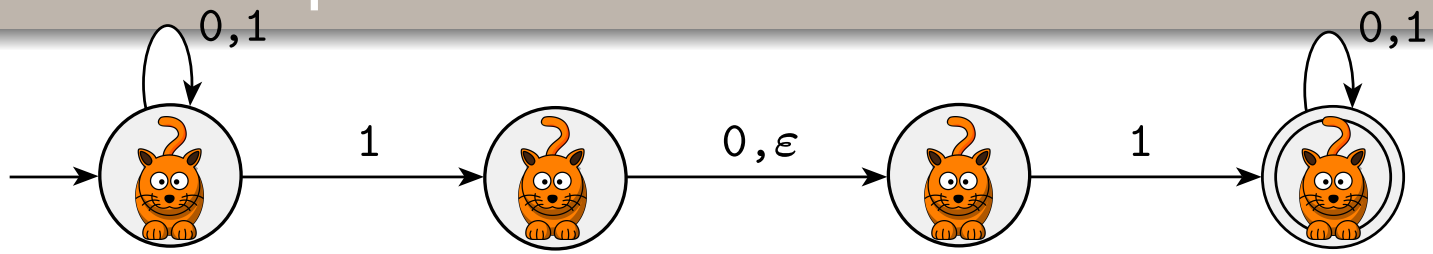
Input: 010110

Symbol read 1

- Cat at q_1 can end up at q_1, q_2 or q_3 so we make three copies
- Cat at q_3 ends up in q_4

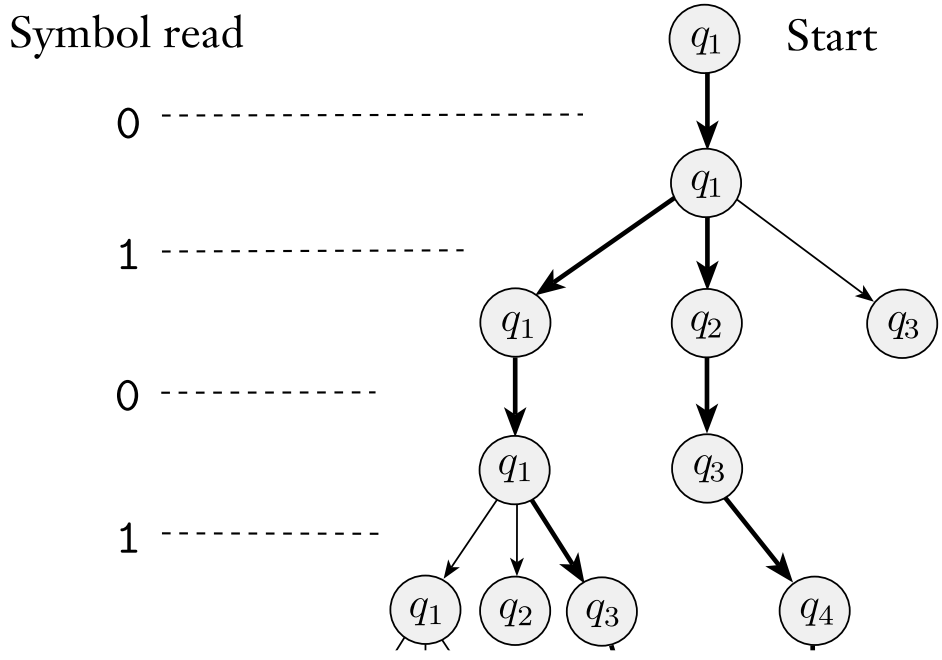


Parallel Viewpoint

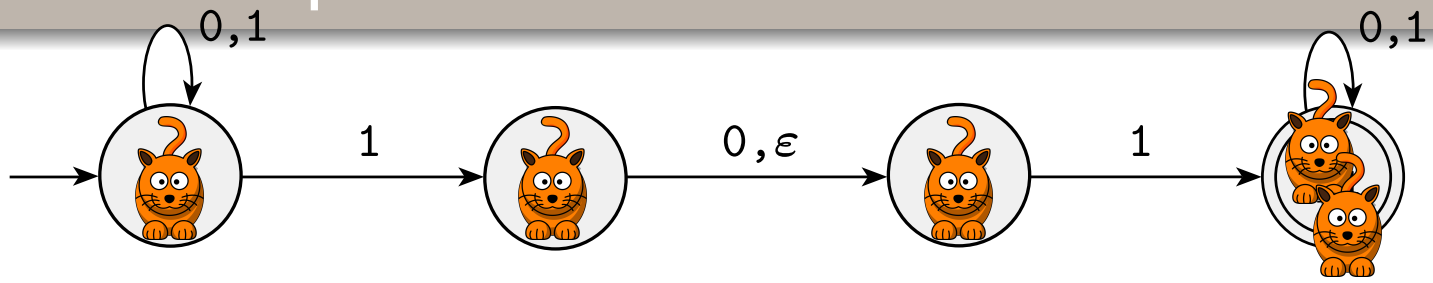


Input: 010110

Symbol read 1



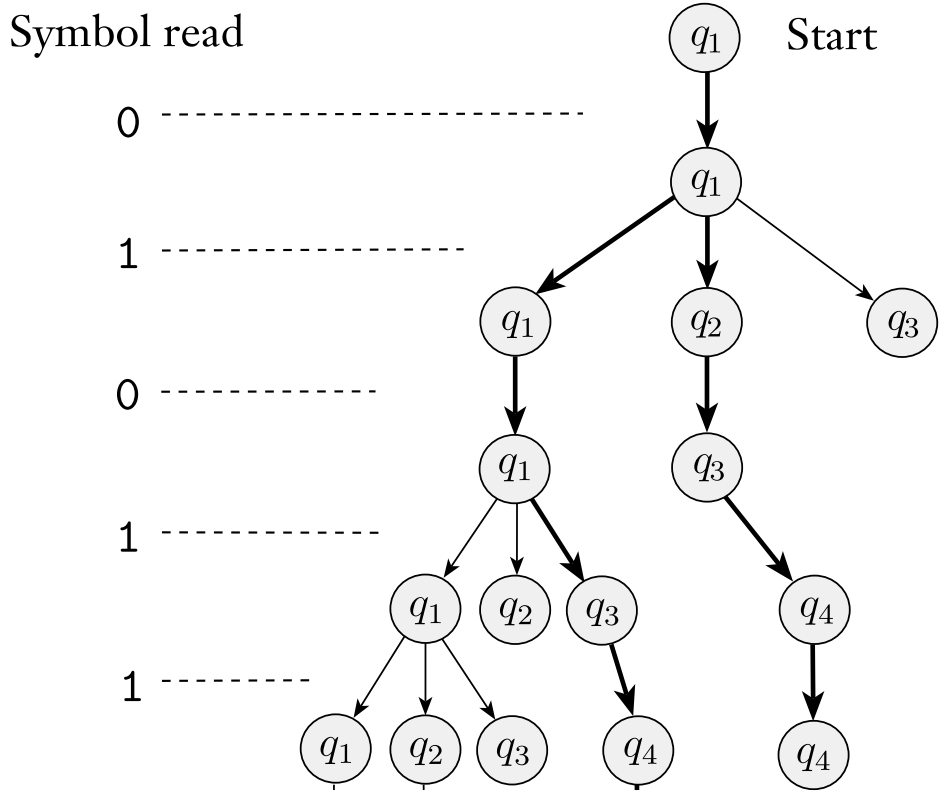
Parallel Viewpoint



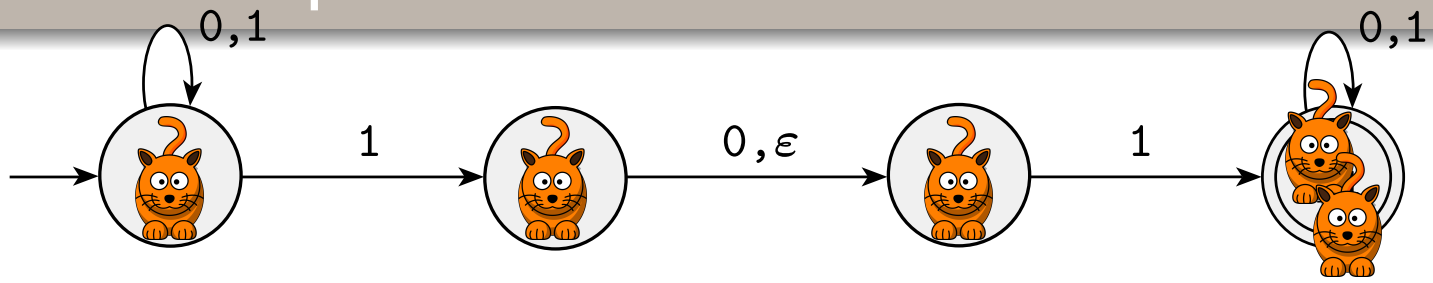
Input: 010110

Symbol read 1

- Cat at q_1 can end up at q_1, q_2 or q_3 so we make three copies
- Cat at q_2 "dies"
- Cat at q_3 ends up in q_4
- Cat at q_4 stays at q_4

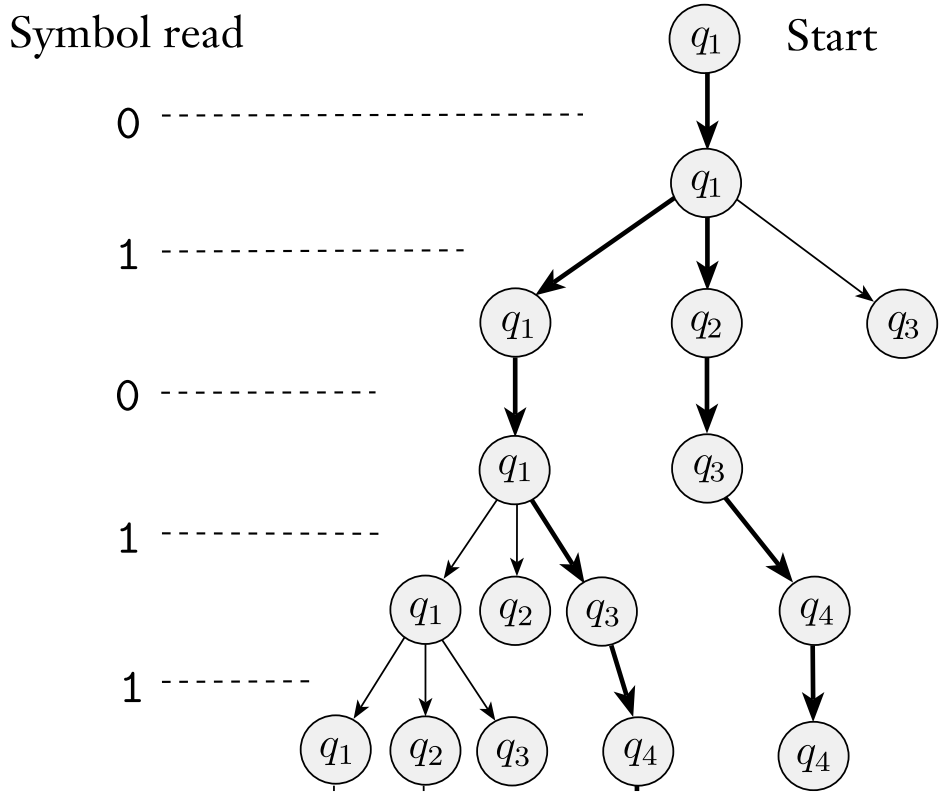


Parallel Viewpoint

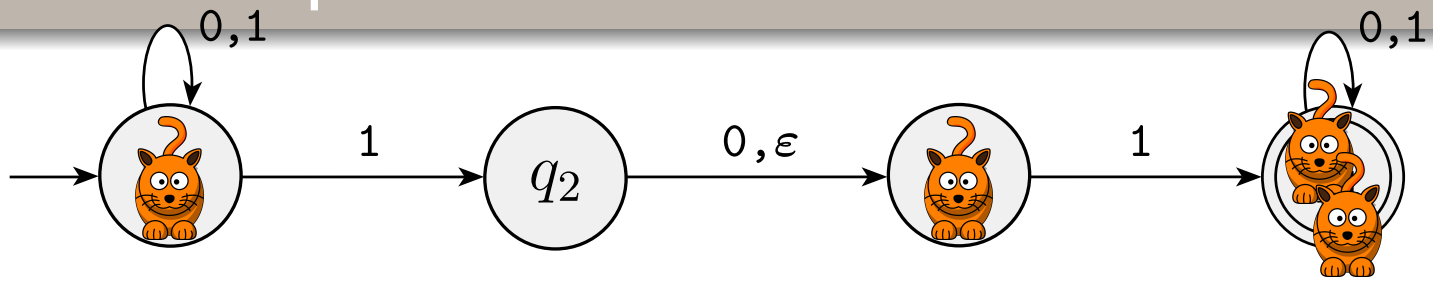


Input: 010110

Symbol read 0



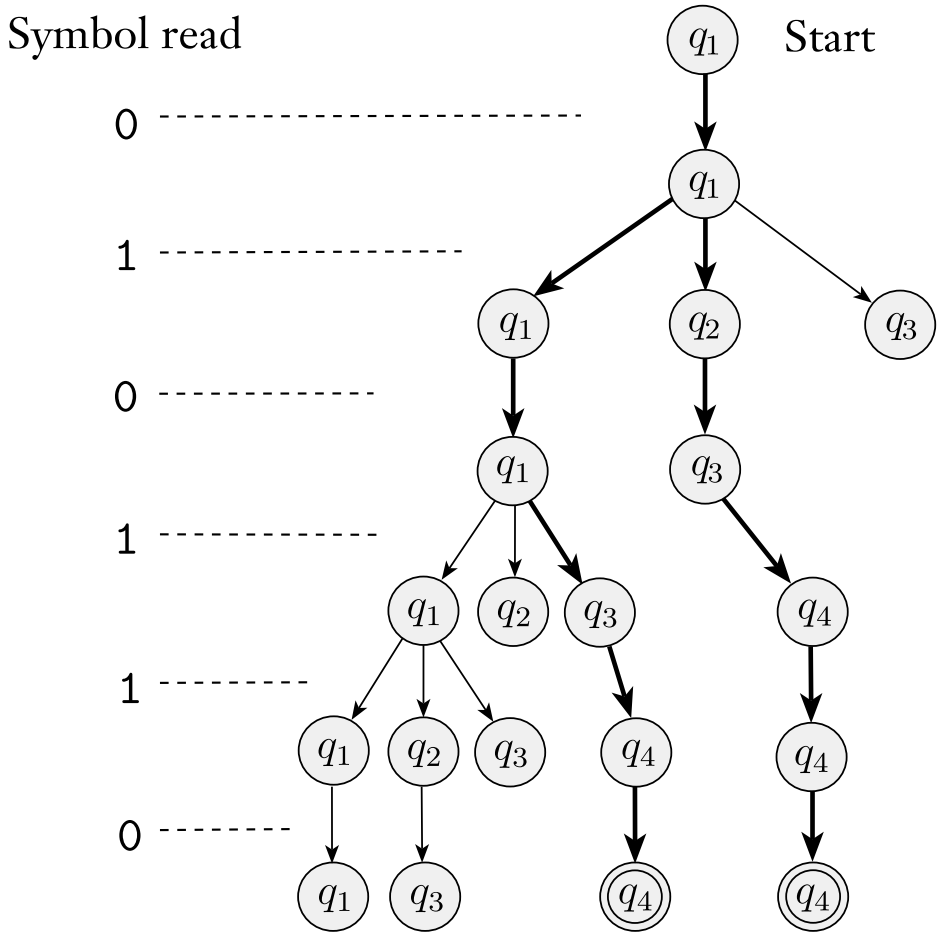
Parallel Viewpoint



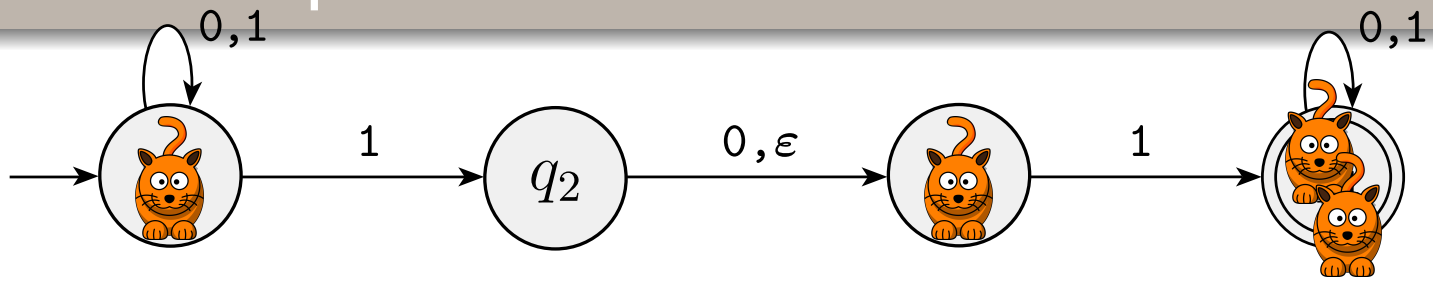
Input: 010110

Symbol read 0

- Cat at q_1 stays at q_1
- Cat at q_2 goes to q_3
- Cat at q_3 dies
- Cats at q_4 stay at q_4



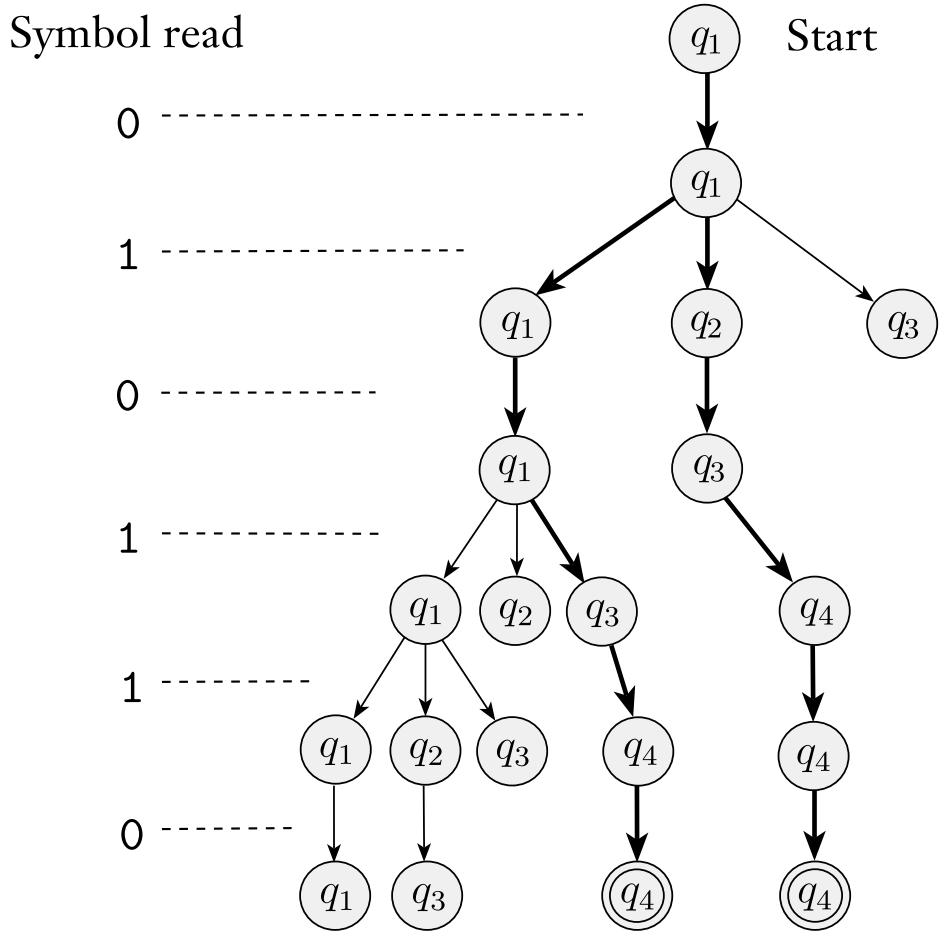
Parallel Viewpoint



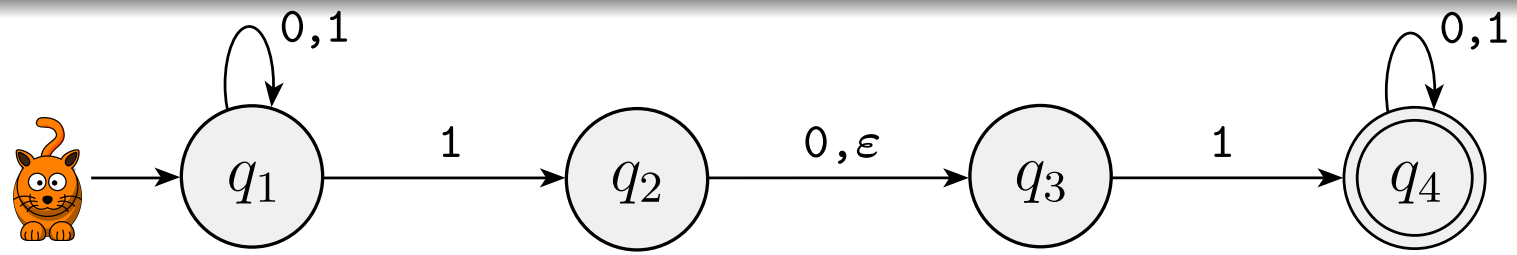
Input: 010110

The input string is accepted if any one of the copies is in an accept state at the end of the input

In the example two copies are in the accept state at the end so the input is accepted

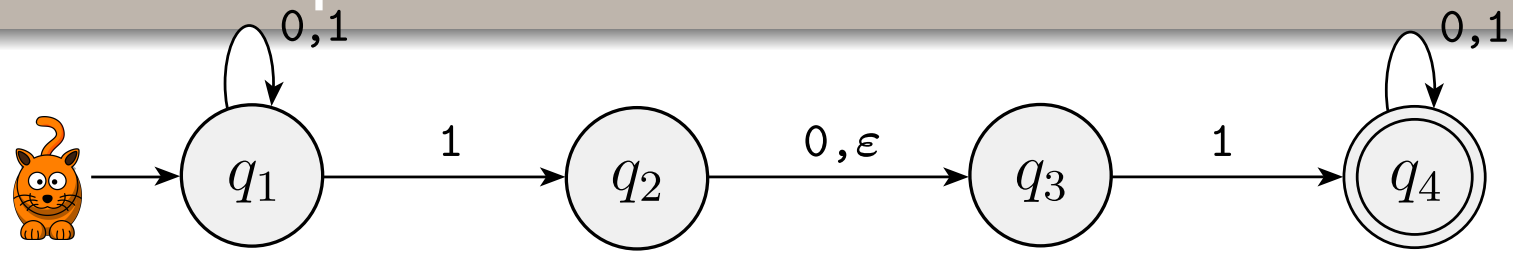


Guess Viewpoint



Input: 010110

Guess Viewpoint



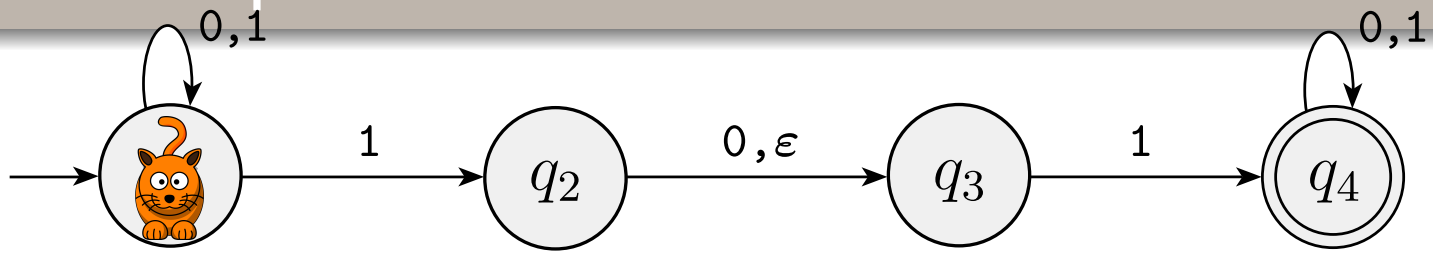
Symbol read q_1 Start

Input: 010110

Let's start

- cat moves to initial state q_1

Guess Viewpoint



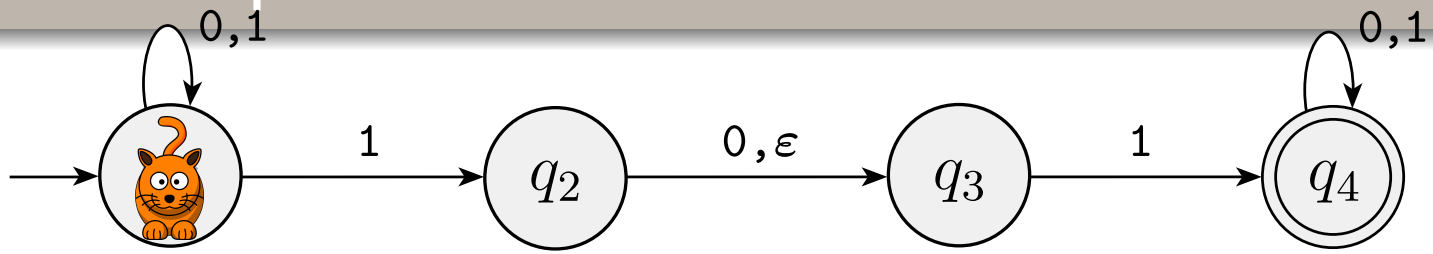
Symbol read q_1 Start

Input: 010110

Let's start

- cat moves to initial state q_1

Guess Viewpoint



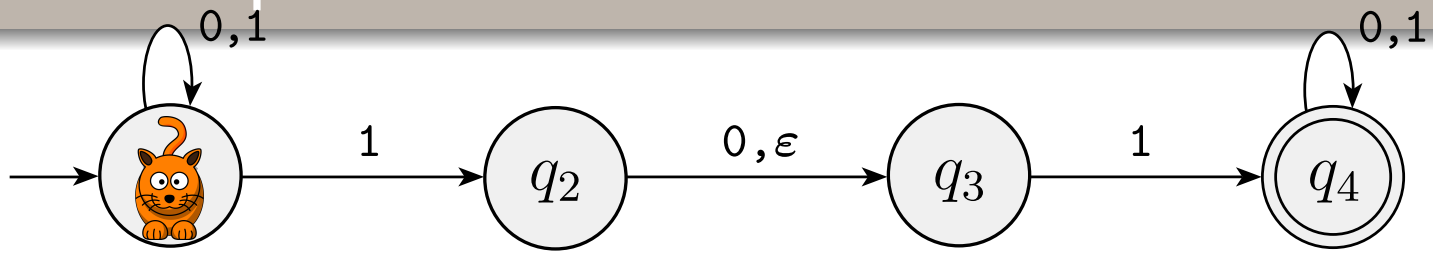
Symbol read

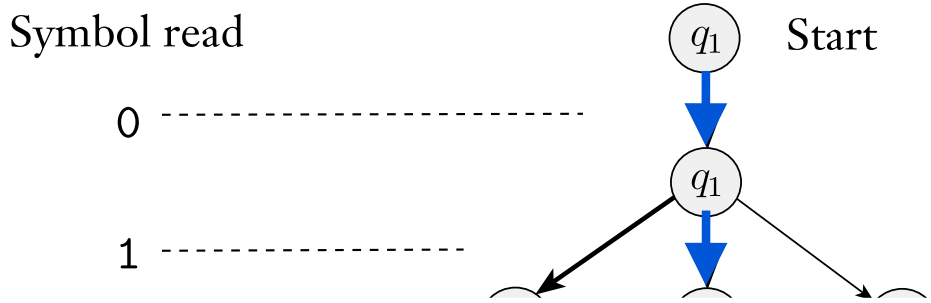
q_1 Start

Input: 010110
▲

Symbol read 0

Guess Viewpoint



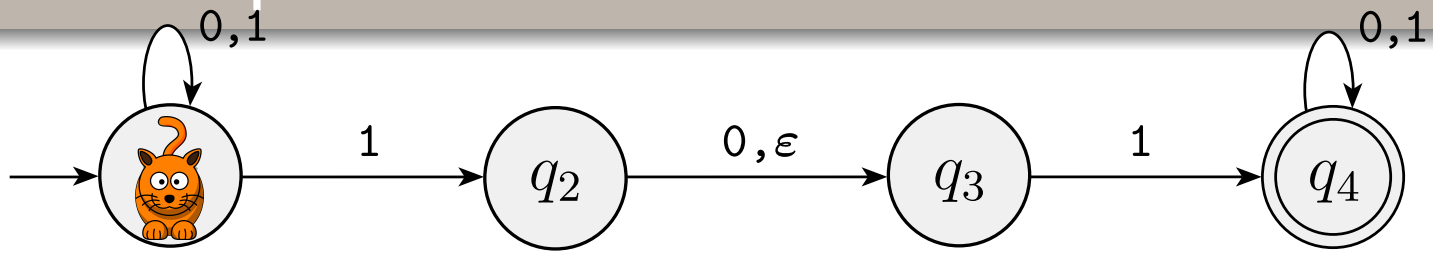


Input: 010110
▲

Symbol read 0

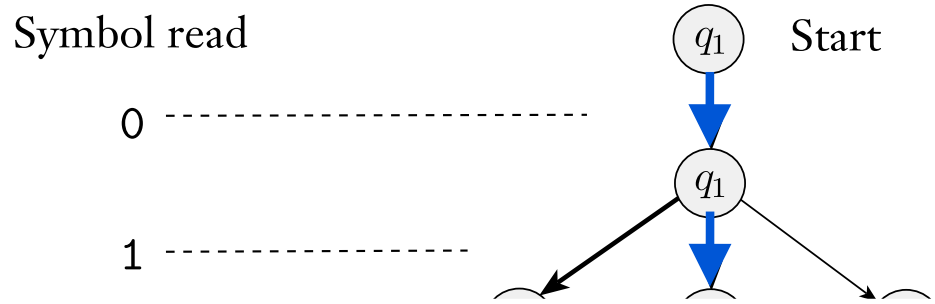
- cat stays in state q_1

Guess Viewpoint

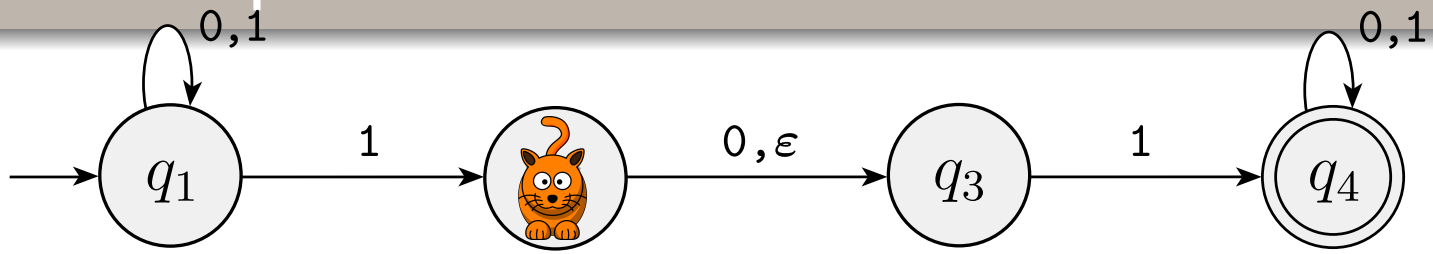


Input: 010110

Symbol read 1



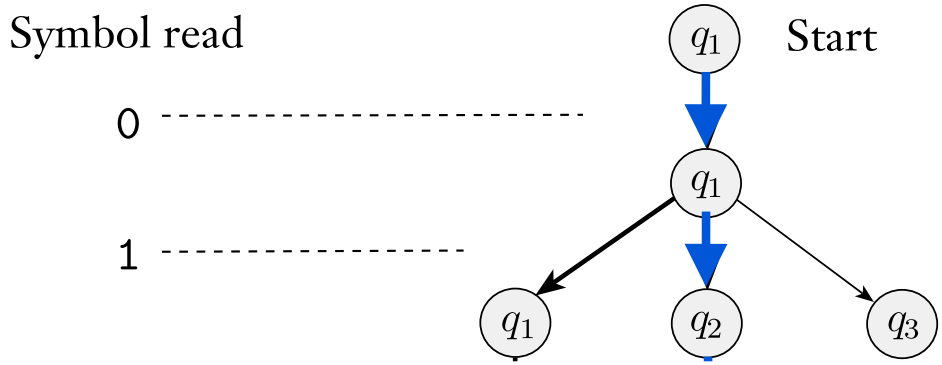
Guess Viewpoint



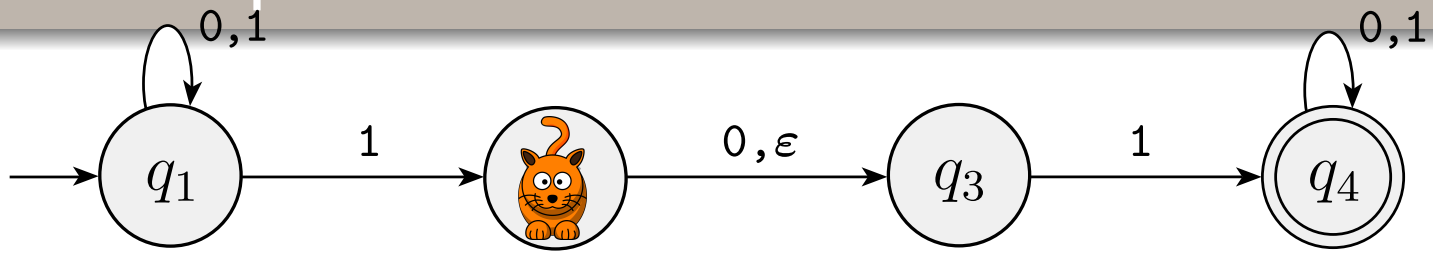
Input: 010110

Symbol read 1

- cat can (i) stay in state q_1, (ii) move to q_2, and (iii) move to q_2 and then to q_3 via the ε-transition
- we make a guess that the cat goes to state q_2

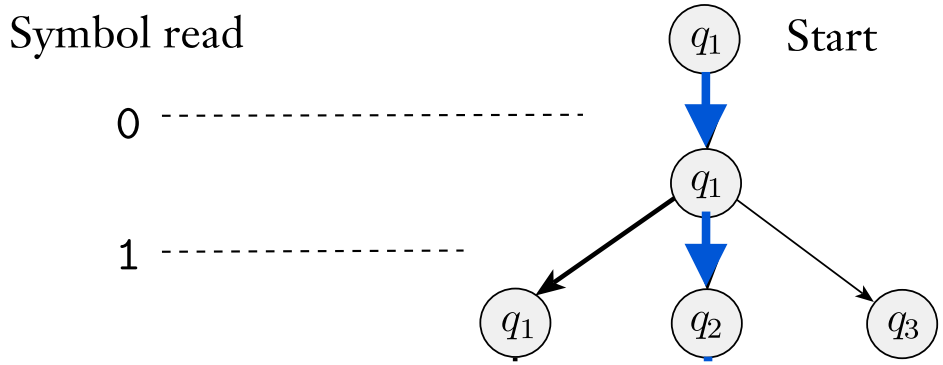


Guess Viewpoint

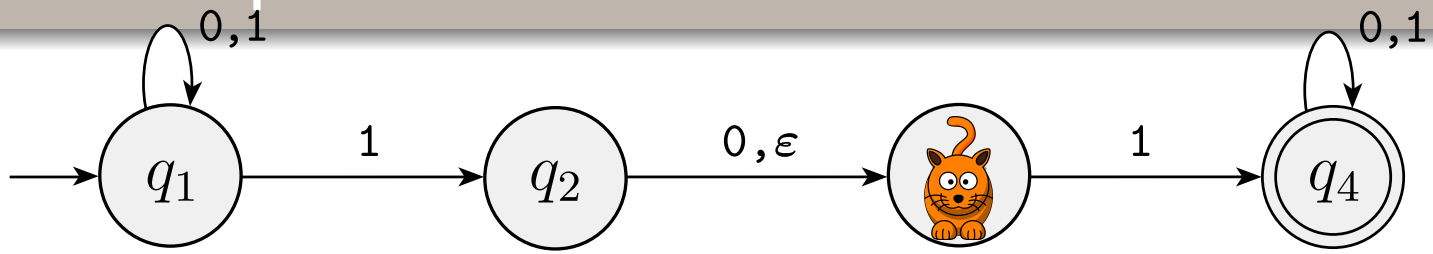


Input: 010110

Symbol read 0



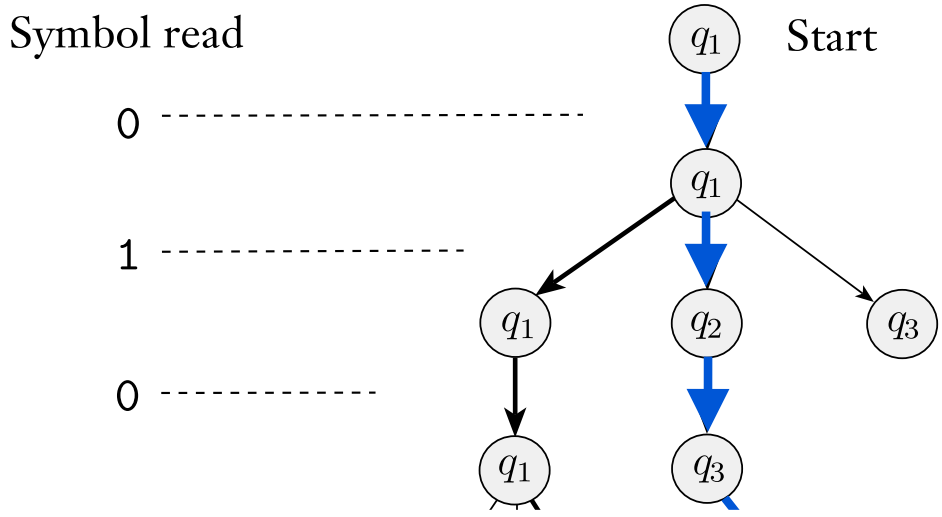
Guess Viewpoint



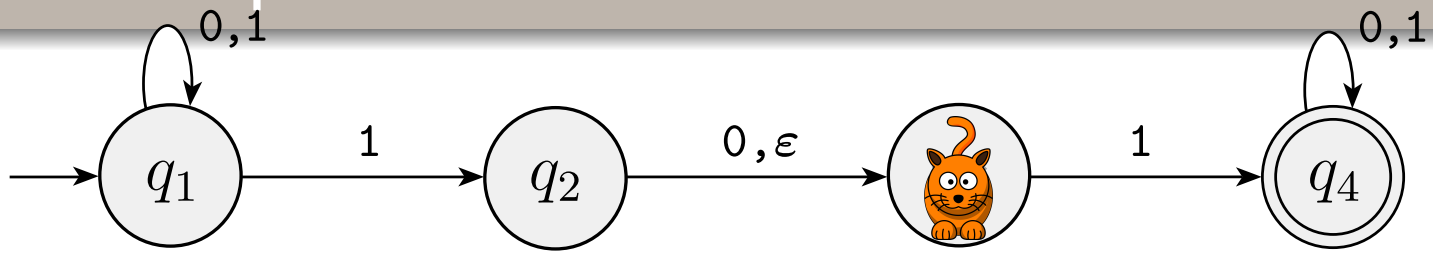
Input: 010110

Symbol read 0

- $q_2 \to q_3$

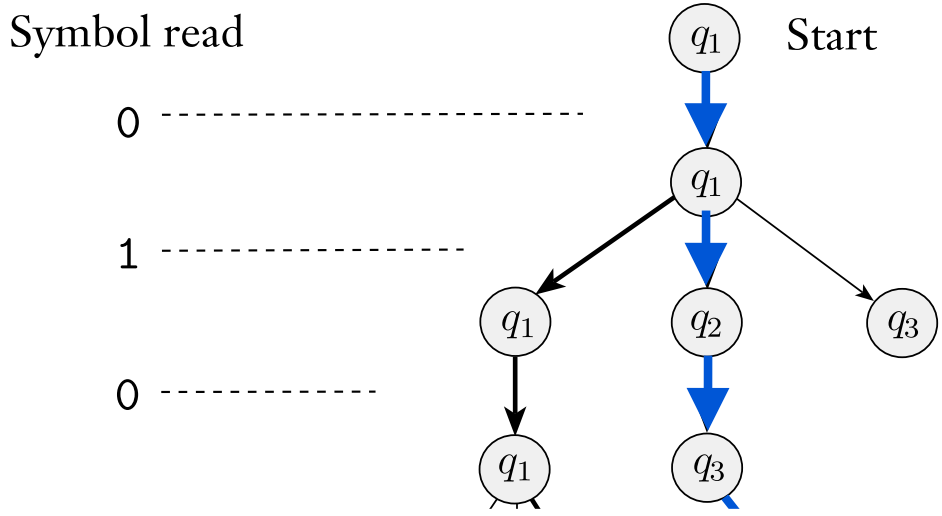


Guess Viewpoint

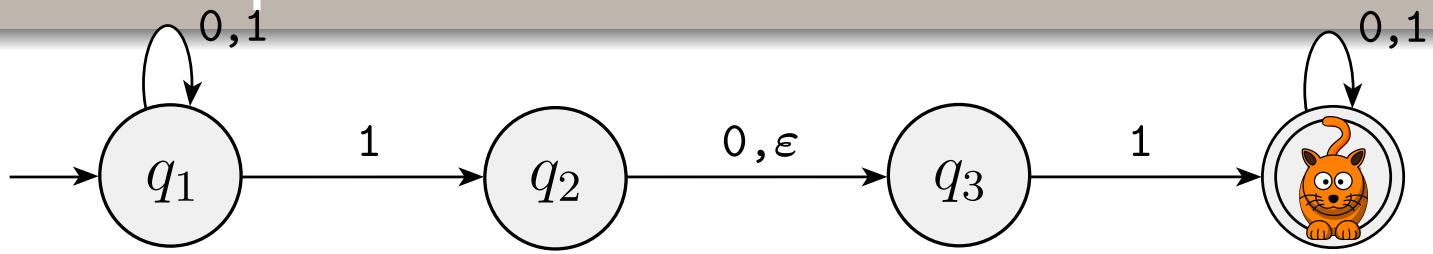


Input: 010110

Symbol read 1



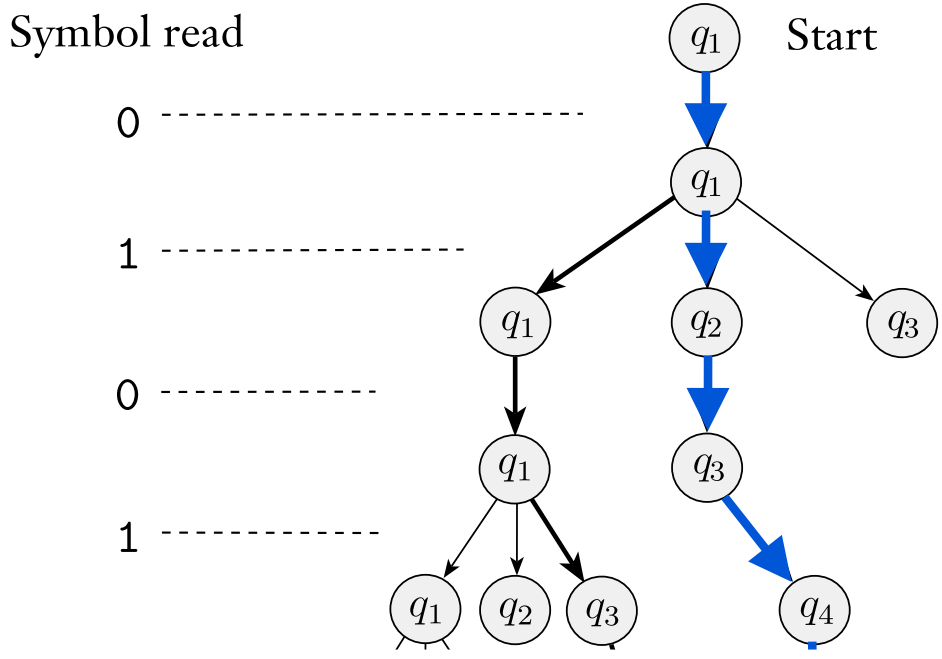
Guess Viewpoint



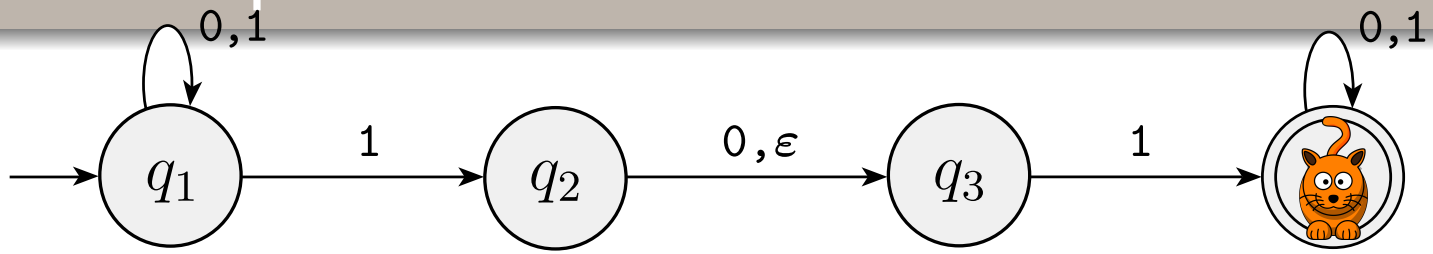
Input: 010110

Symbol read 1

- Cat at q_3 ends up in q_4

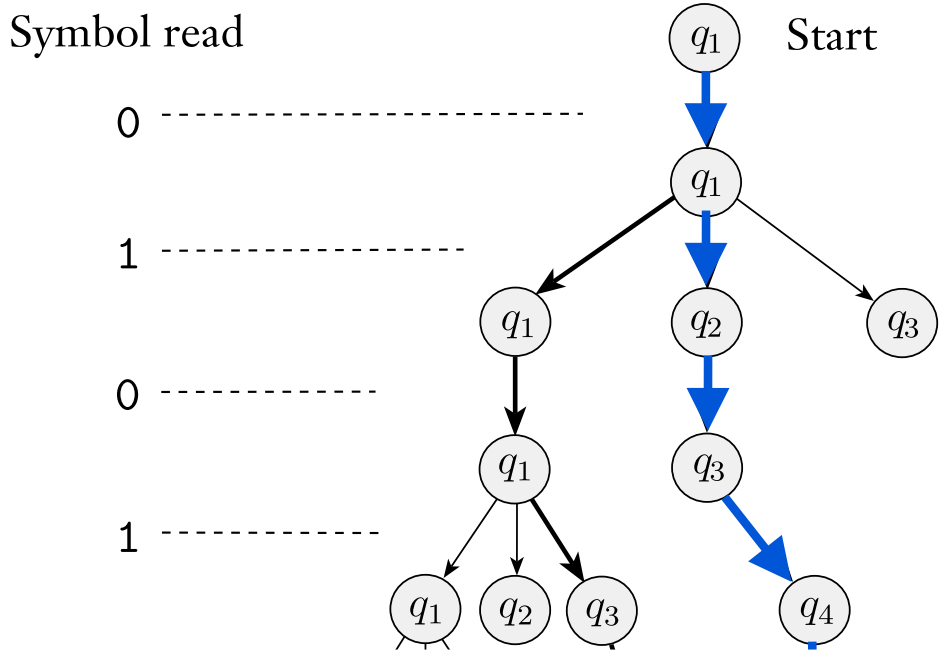


Guess Viewpoint

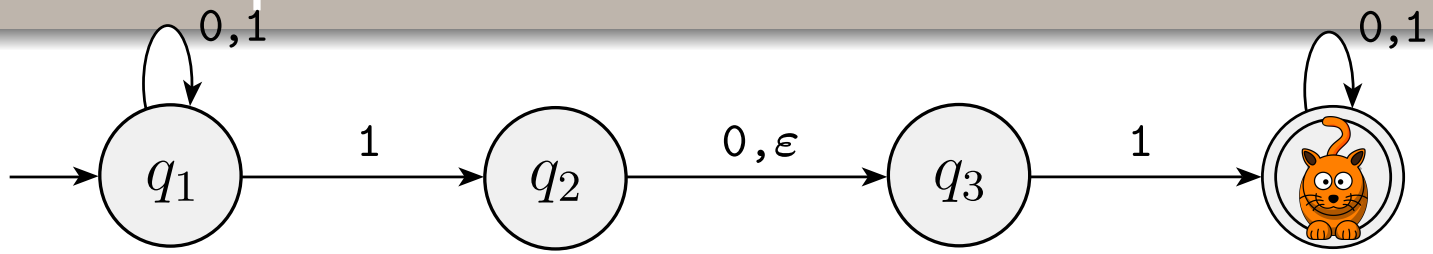


Input: 010110

Symbol read 1



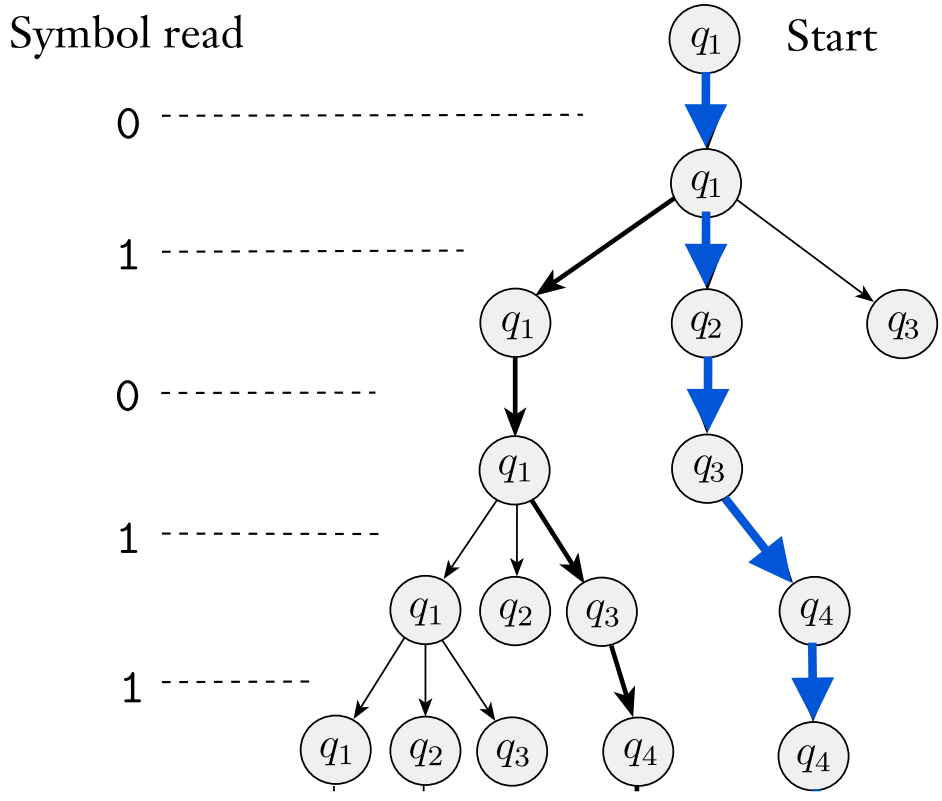
Guess Viewpoint



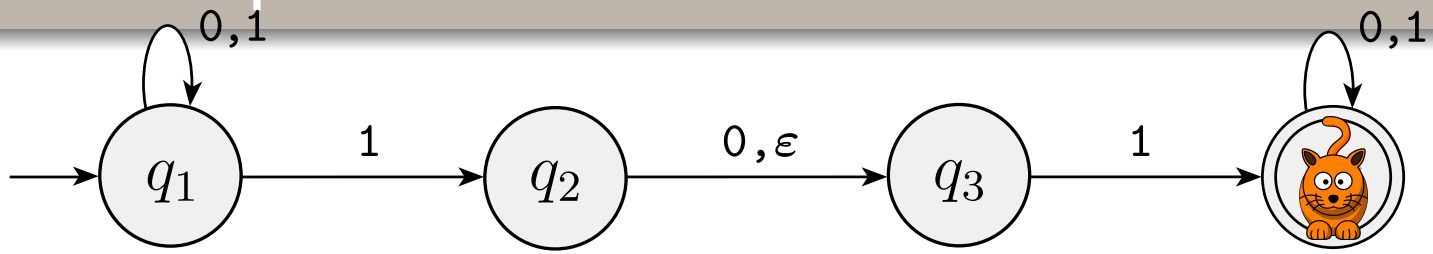
Input: 010110

Symbol read 1

- Cat at q_4 stays at q_4

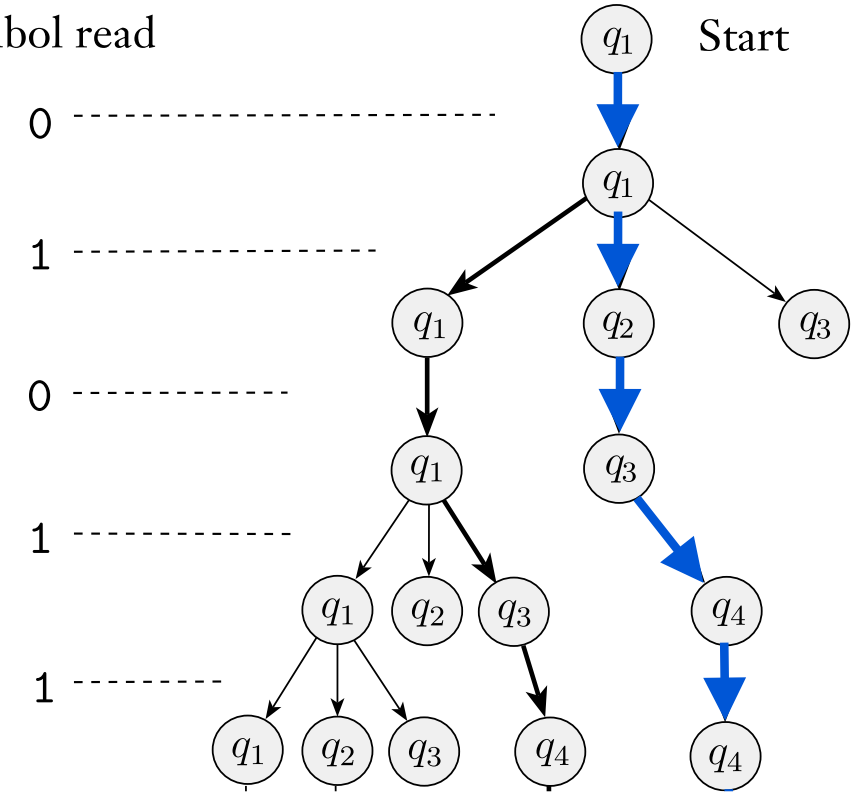


Guess Viewpoint

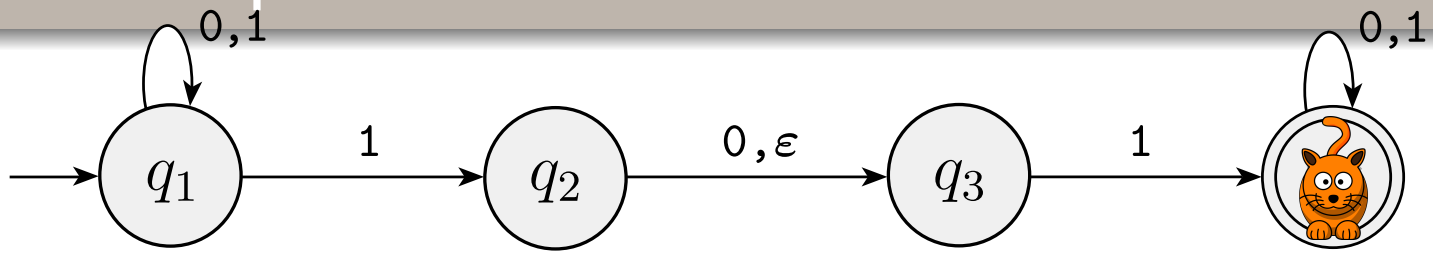


Input: 010110

Symbol read 0



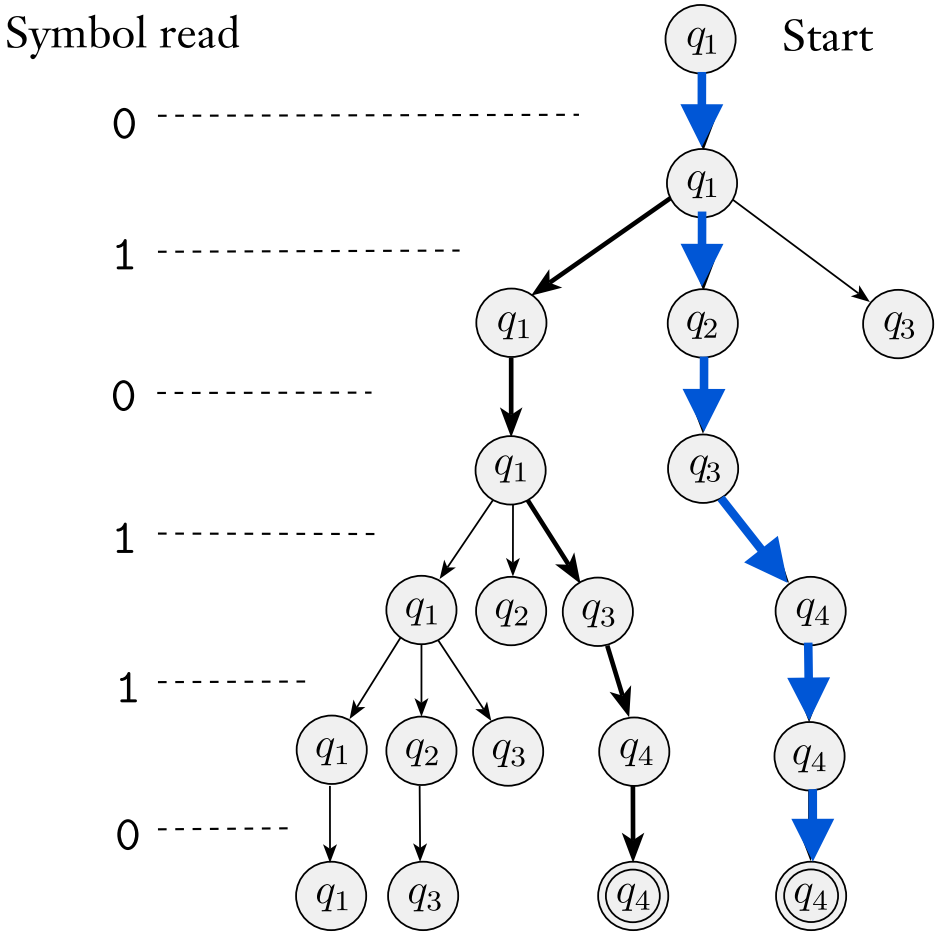
Guess Viewpoint



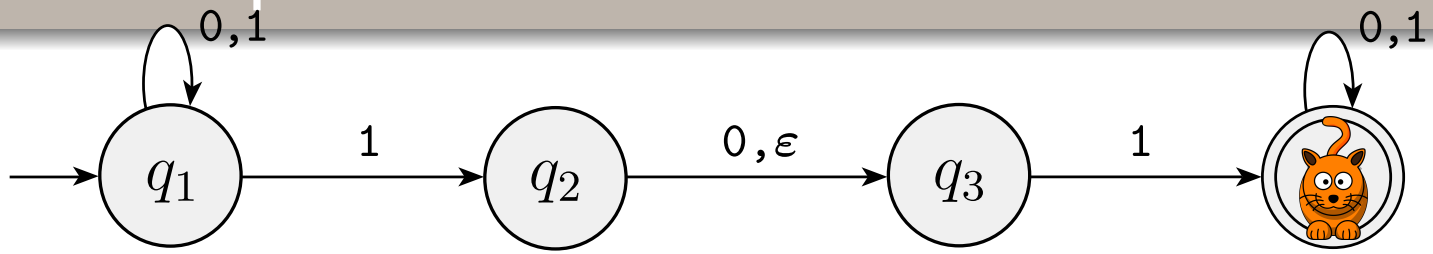
Input: 010110

Symbol read 0

- Cats at q_4 stay at q_4



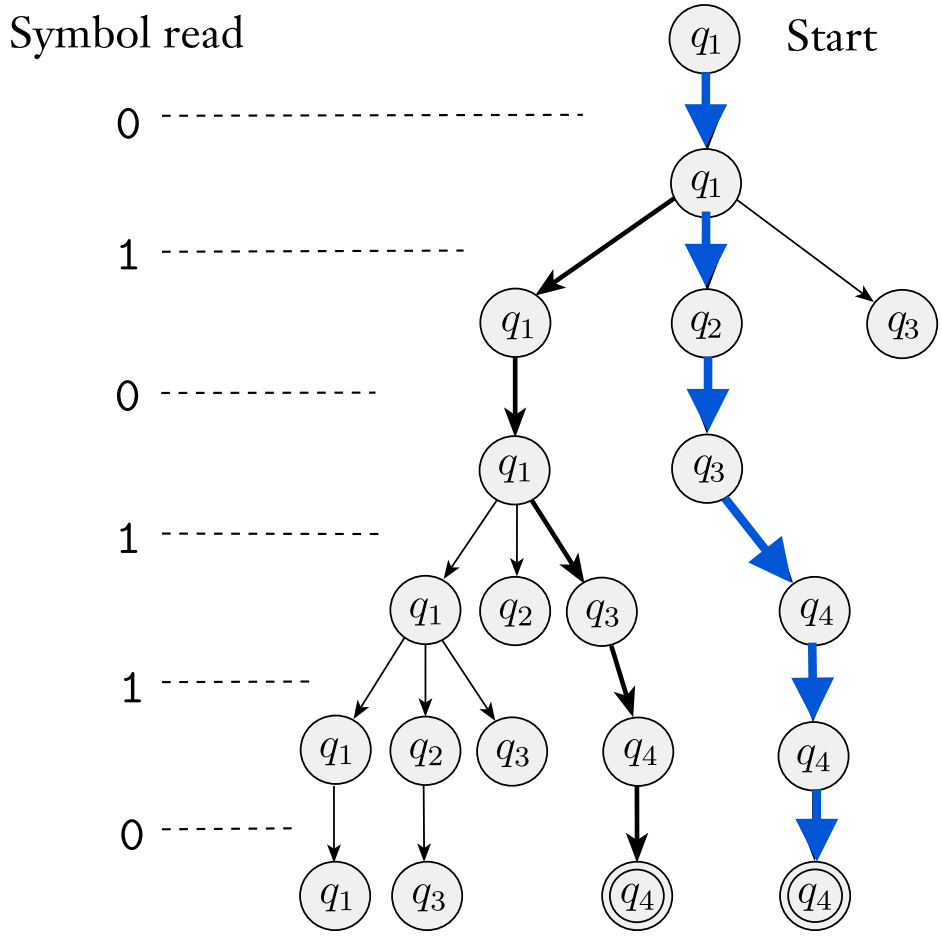
Guess Viewpoint



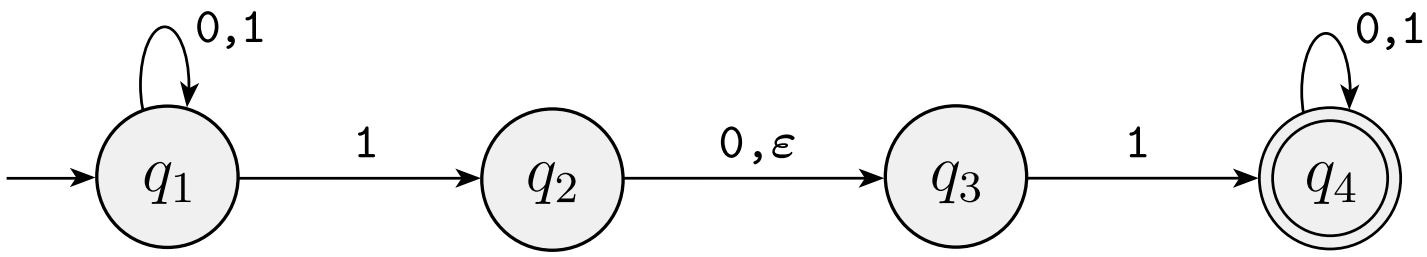
Input: 010110

The input string is accepted if there exist guesses that makes the process end in the accept state at the end of the input

Equivalently, if there exists at least one path that ends at an accept state (Guess and Verify!)



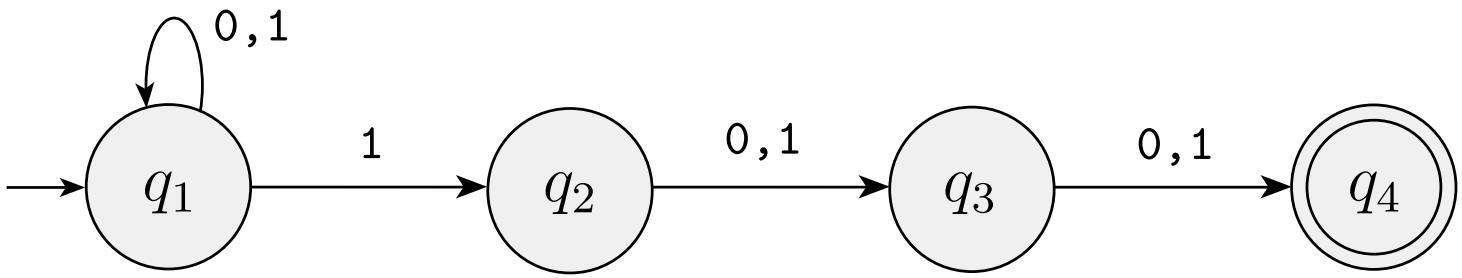
Example 1



What language does the NFA recognize?

$$L = \{w \in \{0,1\}^* \mid w \text{ contains 11 or 101 as a substring}\}$$

Example 2

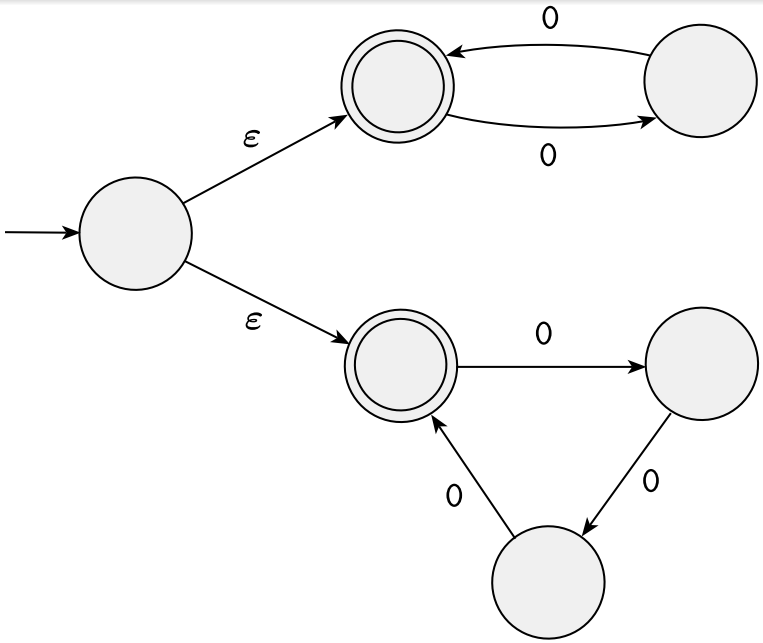


What language does the NFA recognize?

$L = \{w \in \{0,1\}^* \mid w \text{ contains a 1 in the third position from the end}\}$

Remark: If in the computation path you visit an accept state but don't end there, then that path is not accepting. E.g. 1000

Example 3



What language does the NFA recognize?

$$L = \{w \in \{0,1\}^* \mid w = 0^k \text{ where } k \text{ is divisible by 2 or 3}\}$$

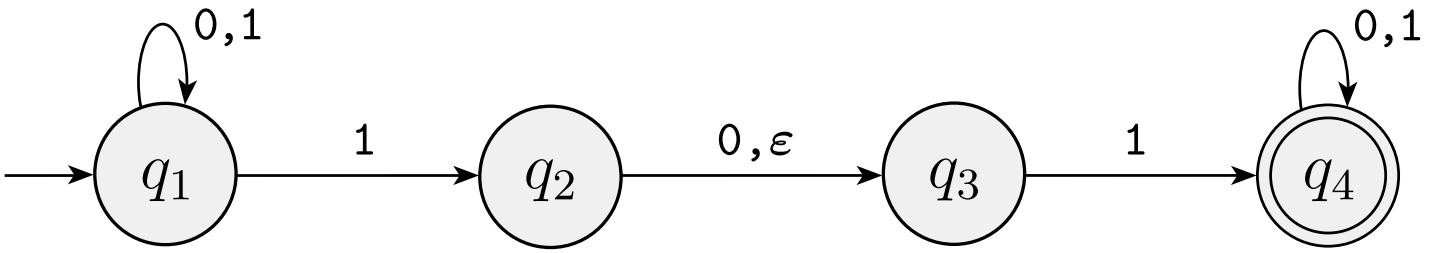
Formal definitions

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

- Q is a finite set called the states,
- Σ is a finite set called the alphabet,
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accept states. *(allow $F = \emptyset$)*

- Here 2^Q denotes the power set of Q. Ex: $Q = \{q_1, q_2\}, 2^Q = \{\emptyset, \{q_1\}, \{q_2\}, \{q_1, q_2\}\}$
- An input is accepted if there exists at least one path that ends at an accepting state
- An input is rejected if no computation path end at an accepting state

Example



The formal description of N_1 is $(Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. δ is given as

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$,

4. q_1 is the start state, and
5. $F = \{q_4\}$.

Why Study Nondeterminism?

If there is no "real-world" analog of nondeterminism construct, namely, make the choice that will lead to success, why bother?

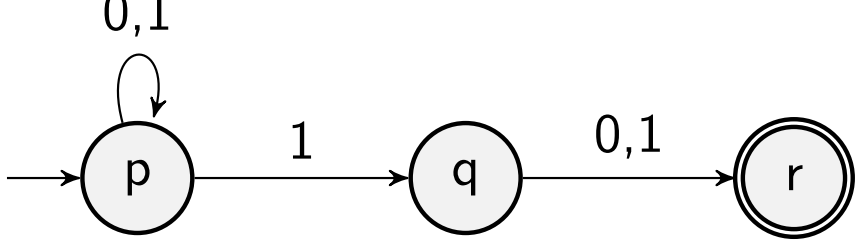
- Mathematically well-defined and valid question to study its impact on computational power
- Useful in translation of regular languages:
  - As we will show, every NFA can be converted into an equivalent DFA
  - So a language is regular iff it is recognized by an NFA
  - Constructing NFAs is sometimes easier than directly constructing DFAs (doing concatenation with NFA is an exercise)
  - NFA may be much (exponentially) smaller than its deterministic counterpart, or its functioning may be easier to understand
- Nondeterminism is also critical to theory of NP-complete problems which includes a large number of natural optimization problems

Theorem. Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

Corollary. A language is regular if and only if some nondeterministic finite automaton recognizes it.

- We prove the theorem for NFAs without ε-transitions
- Exercises
  - Change the proof to work in the presence of ε-transitions (see book)
  - Given an NFA N construct an NFA N' with $L(N') = L(N)$ and no ε-transitions in N' (see Exercise Set 2)

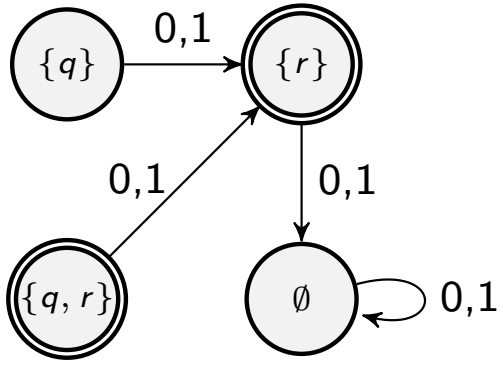
Example

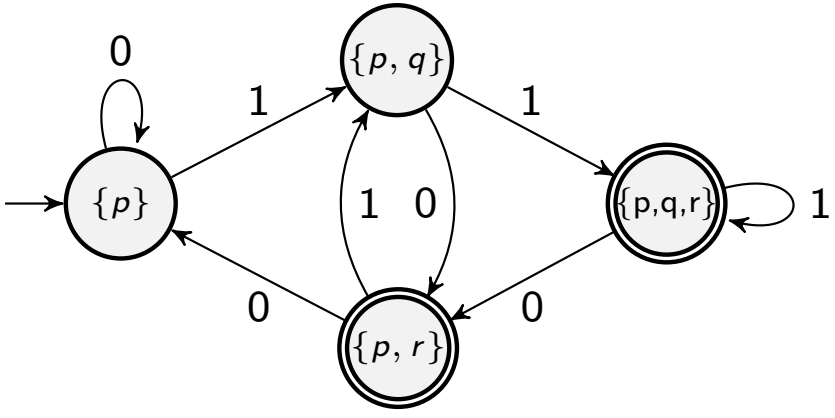


Define $\tilde{\delta} : 2^Q \times \Sigma \to 2^Q$ be function that maps possible states A to the possible states after reading one more symbol a:

$$\tilde{\delta}(A, a) = \bigcup_{q \in A} \delta(q, a)$$

$\tilde{\delta}$	**0**	**1**
$\emptyset$	$\emptyset$	$\emptyset$
$\{p\}$	$\{p\}$	$\{p, q\}$
$\{q\}$	$\{r\}$	$\{r\}$
$\{r\}$	$\emptyset$	$\emptyset$
$\{p, q\}$	$\{p, r\}$	$\{p, q, r\}$
$\{p, r\}$	$\{p\}$	$\{p, q\}$
$\{q, r\}$	$\{r\}$	$\{r\}$
$\{p, q, r\}$	$\{p, r\}$	$\{p, q, r\}$





Extended Transition Function

For a DFA: A state and a sequence of symbols maps to a state by sequentially applying transit function, by reading the input sequence letter by letter

$$M = (Q_M, \Sigma, \delta_M, q_0, F_M) \qquad \Delta_M : Q_M \times \Sigma^* \to Q_M$$

$$\Delta_M(A, \varepsilon) = A$$
$$\Delta_M(A, xa) = \delta_M(\Delta_M(A, x), a)$$

For an NFA: A set of states and a sequence of symbols maps to a set of states by sequentially applying transit function, by reading the input sequence letter by letter. A state with null input maps to the same state (no ε-transitions)

$$N = (Q_N, \Sigma, \delta_N, q_0, F_N) \qquad \Delta_N : 2^{Q_N} \times \Sigma^* \to 2^{Q_N}$$

$$\Delta_N(A, \varepsilon) = A$$
$$\Delta_N(A, xa) = \cup_{q \in \Delta_N(A,x)} \delta_N(q, a)$$

Lemma: For all $x, y \in \Sigma^*$, $\Delta_N(A, xy) = \Delta_N(\Delta_N(A, x), y)$

- Proof via induction on $|xy|$ — exercise

Theorem. Every nondeterministic finite automaton has an equivalent deterministic finite automaton

Proof

Take any NFA:

$$N = (Q_N, \Sigma, \delta_N, q_0, F_N)$$

Extended transition function:

$$\Delta_N : 2^{Q_N} \times \Sigma^* \to 2^{Q_N}$$
$$\Delta_N(A, \varepsilon) = A$$
$$\Delta_N(A, xa) = \cup_{q \in \Delta_N(A,x)} \delta_N(q, a)$$

We need to construct a DFA M that accepts the same language

$$M = (Q_M, \Sigma, \delta_M, q_0, F_M)$$

The Subset Construction:

$$Q_M = 2^{Q_N}$$
$$\delta_M(A, a) = \Delta_N(A, a)$$
$$F_M = \{A \subseteq Q_N : A \cap F_N \neq \emptyset\}$$

Proof of Theorem (continued)

Lemma. For all $A \subseteq Q_N, x \in \Sigma^*$, $\Delta_M(A, x) = \Delta_N(A, x)$

Proof of Lemma: Induction on |x|
Base case: $x = \varepsilon$

$$\Delta_M(A, \varepsilon) = A = \Delta_N(A, \varepsilon)$$

Induction Step: Let $x = ya$ where $a \in \Sigma$ and $a \neq \emptyset$

$$\begin{aligned}
\Delta_M(A, ya) &= \delta_M(\Delta_M(A, y), a) && \text{(Definition of } \Delta_M\text{)} \\
&= \delta_M(\Delta_N(A, y), a) && \text{(Induction as } |y| < |x|\text{)} \\
&= \Delta_N(\Delta_N(A, y), a) && \text{(By subset construction)} \\
&= \Delta_N(A, ya) && \text{(Definition of } \Delta_N\text{)}
\end{aligned}$$

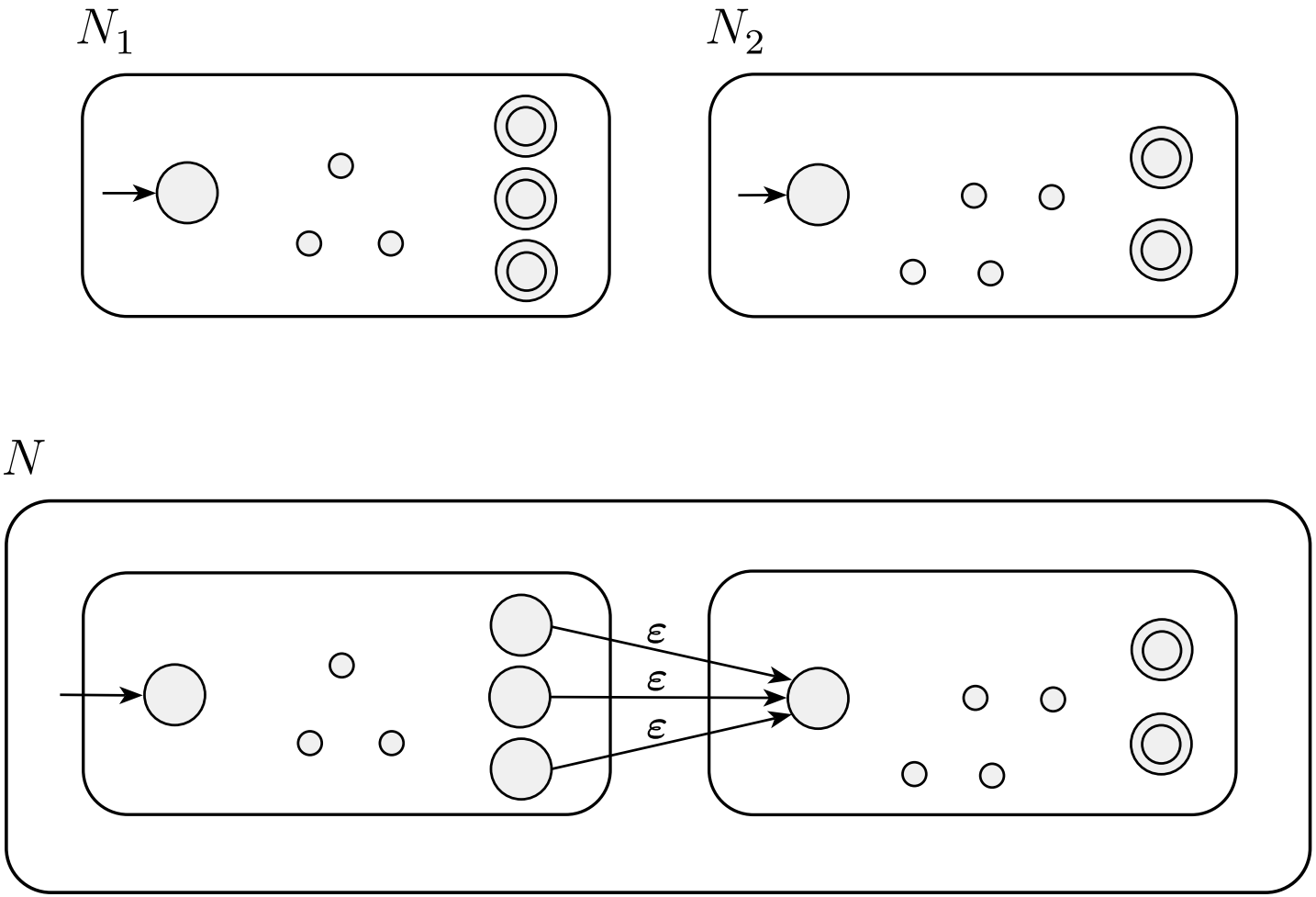
Lemma implies Theorem:

$$x \in L(M) \Leftrightarrow \Delta_M(q_0, x) \in F_M \Leftrightarrow \Delta_M(q_0, x) \cap F_N \neq \emptyset$$
$$\underbrace{\Longleftrightarrow}_{\text{Lemma}} \Delta_N(q_0, x) \cap F_N \neq \emptyset \Leftrightarrow x \in L(N)$$

Back to Concatenation

$$L_1 \circ L_2 = \{w \in \Sigma^* : w = w_1.w_2,\ w_1 \in L_1 \textbf{ and } w_2 \in L_2\}$$

Concatenation



Formal construction

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1,
and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$

1. $Q = Q_1 \cup Q_2$ The states of N are all the states of N_1 and N_2
2. The state q_1 is the same as the start state of N_1
3. The accept states F_2 are the same as the accept states of N_2
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma \cup \{\varepsilon\}$,

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q, a) \cup \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\ \delta_2(q, a) & q \in Q_2 \end{cases}$$

Is every Language Regular?