DFAs and NFAs

Announcements

- Problem Set 4 due last night at 7:00PM.
  - Can submit using a late day up to 7:00PM tonight.
- Problem Set 5 out, due **Friday**, **November 4** at 2:15PM.
  - Specifically designed to make up for the difficulty of that last problem set.
  - Have fun playing around with finite automata!
- Friday Four Square today at 4:15 in front of Gates!

A Formal Definition of DFAs

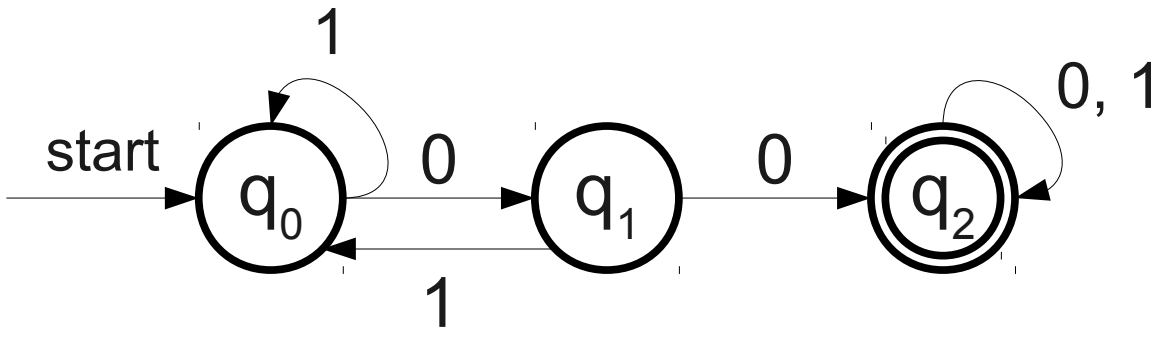
- Formally, a DFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where
  - Q is a set of states.
  - Σ is an alphabet.
  - $\delta : Q \times \Sigma \to Q$ is the **transition function**.
  - $q_0 \in Q$ is the **start state**.
  - $F \subseteq Q$ is a set of **accepting states**.

A Formal Definition of Acceptance

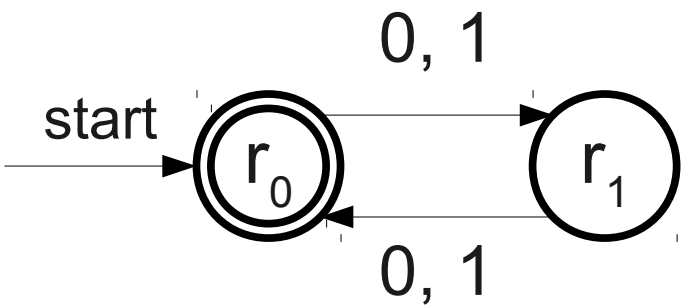
- Given a DFA $(Q, \Sigma, \delta, q_0, F)$, δ^* is defined recursively.
- $\delta^* (\varepsilon) = q_0$
  - Running the automaton on ε ends in the start state.
- $\delta^* (wa) = \delta(\delta^* (w), a)$
  - Running on *w*a is equal to running the automaton on *w*, then following the transition for a.
- $L(D) = \{ w \mid \delta^* (w) \in F \}$
  - The set of strings that end in an accepting state.

Intersecting Regular Languages

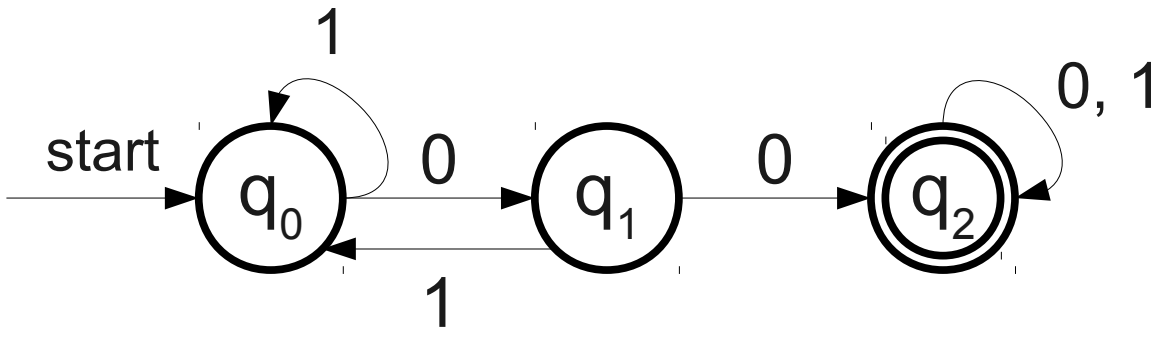
- The **intersection** of two languages L_1 and L_2 is the set $L_1 \cap L_2$.
- Intuitively, the language formed by taking the strings in L_1 and filtering them down by eliminating anything not in L_2.
- **Question:** If L_1 and L_2 are regular languages, is $L_1 \cap L_2$ a regular language?

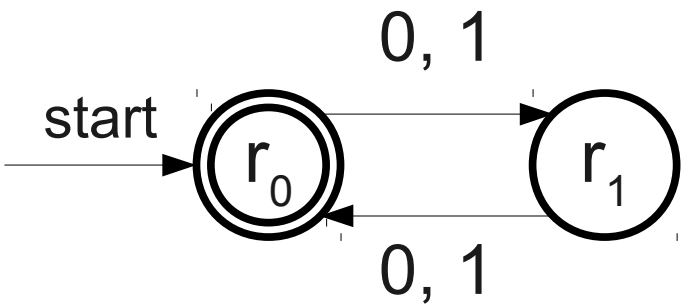


L_1 = { w | w contains 00 as a substring }



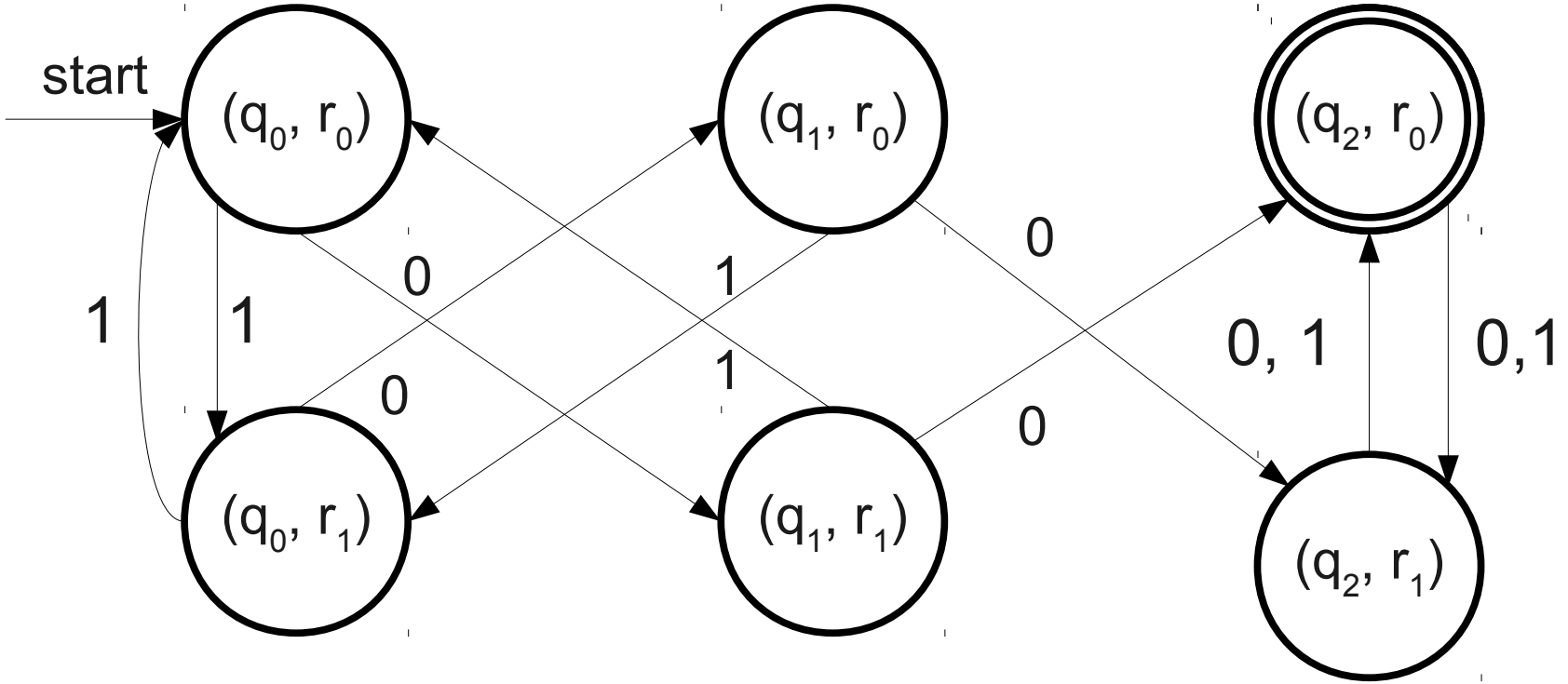
L_2 = { w | w has even length }

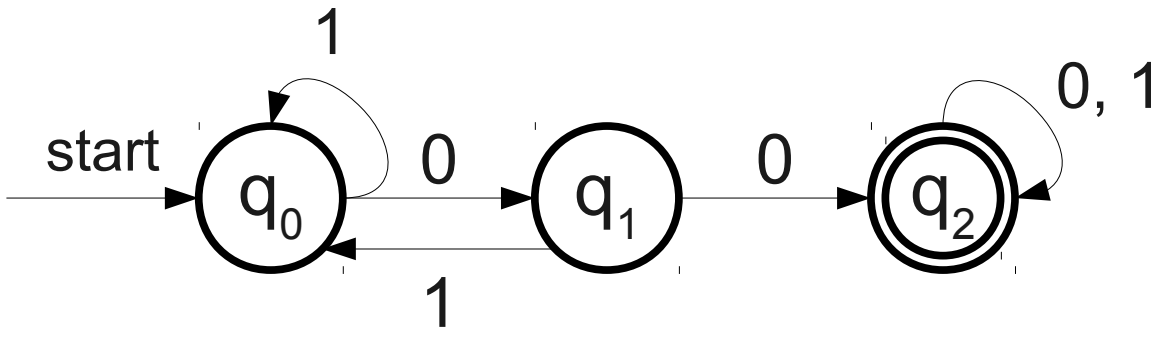


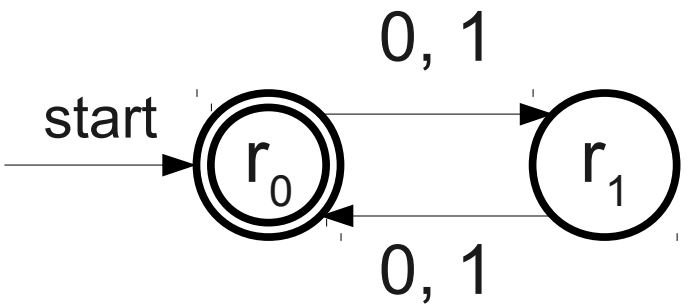


L_1 = { w | w contains 00 as a substring }

L_2 = { w | w has even length }

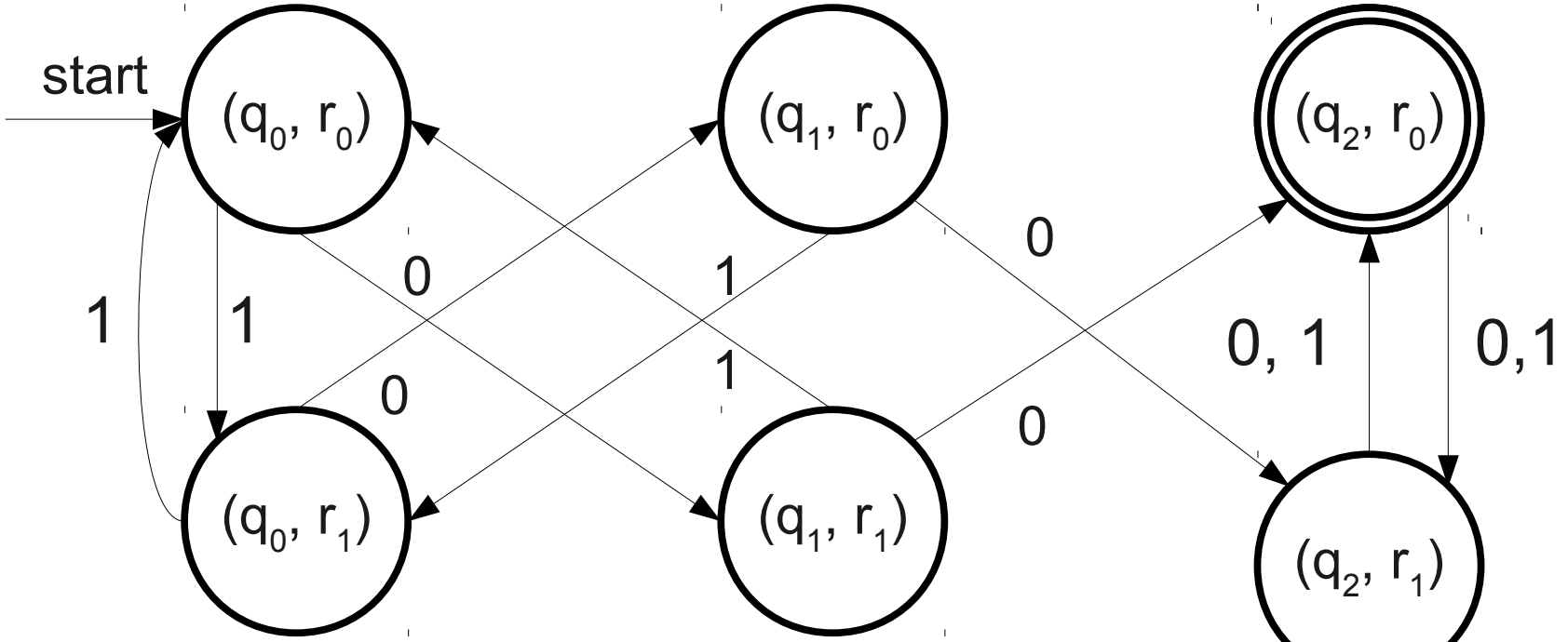




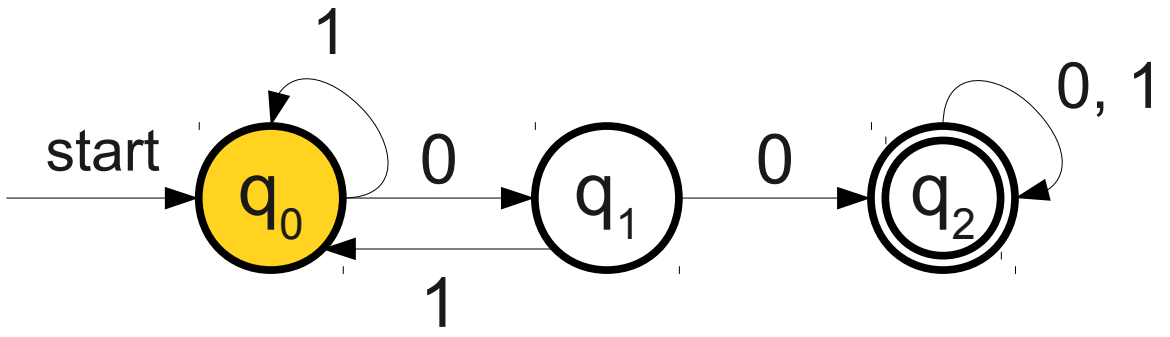


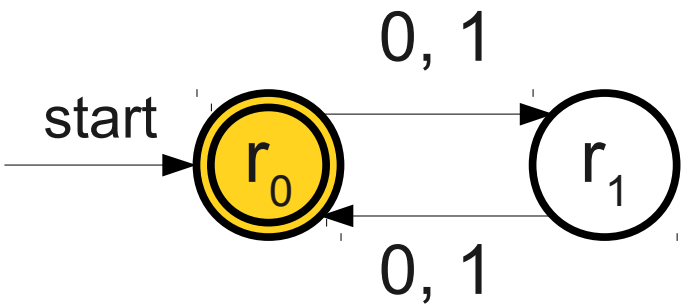
L_1 = { w | w contains 00 as a substring }

L_2 = { w | w has even length }



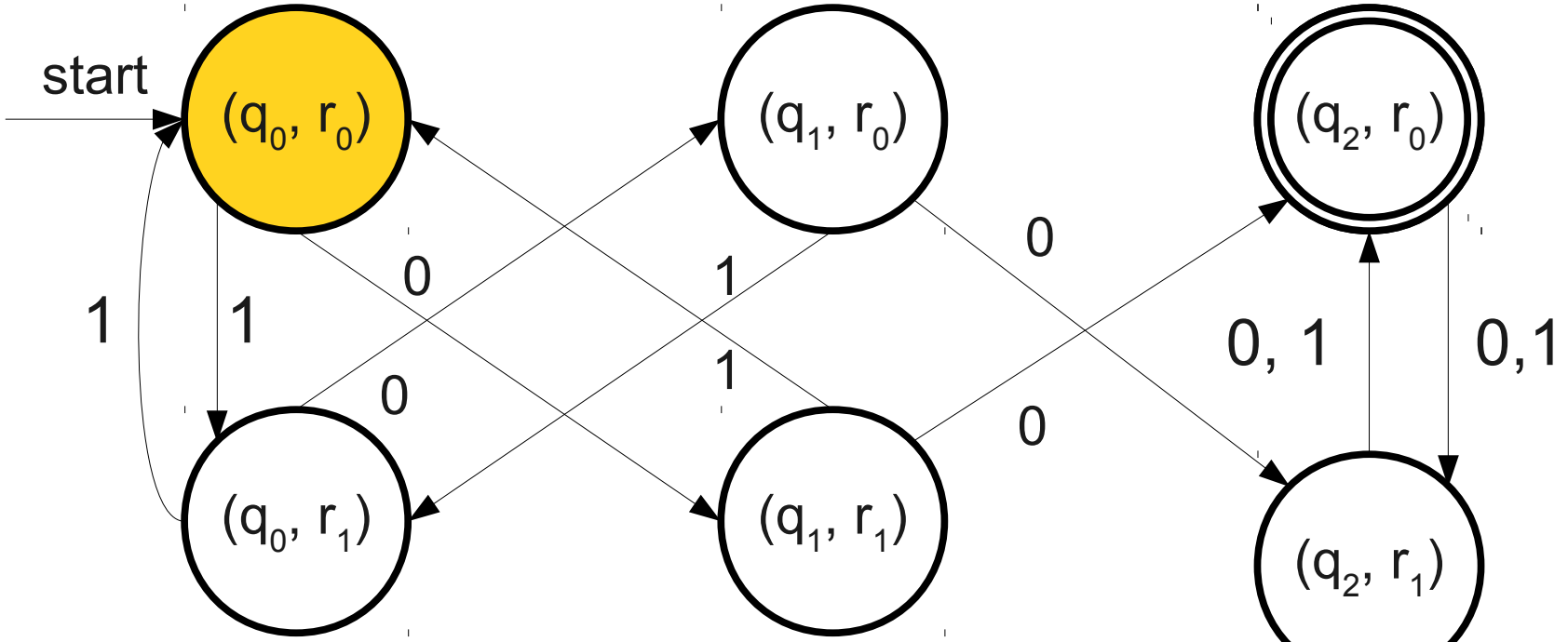
0 1 0 0 1



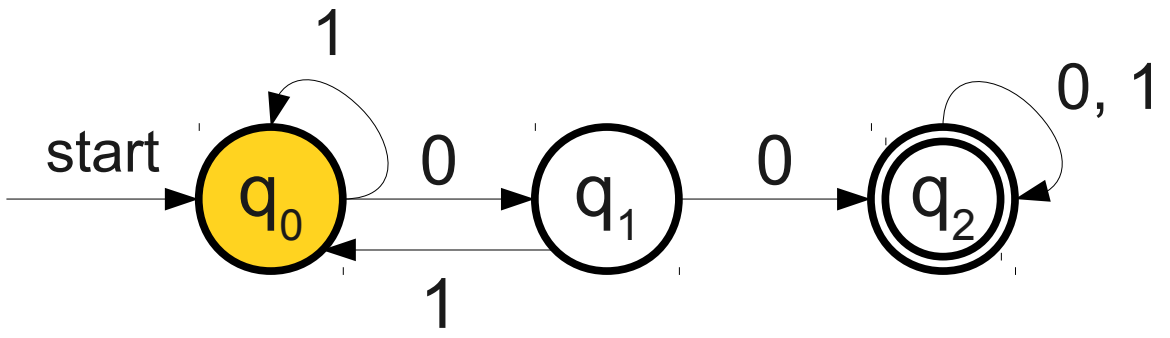


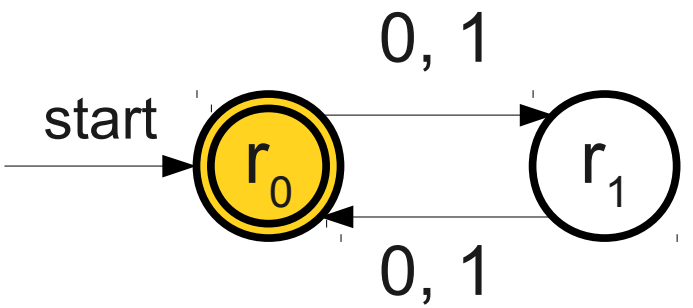
L_1 = { w | w contains 00 as a substring }

L_2 = { w | w has even length }



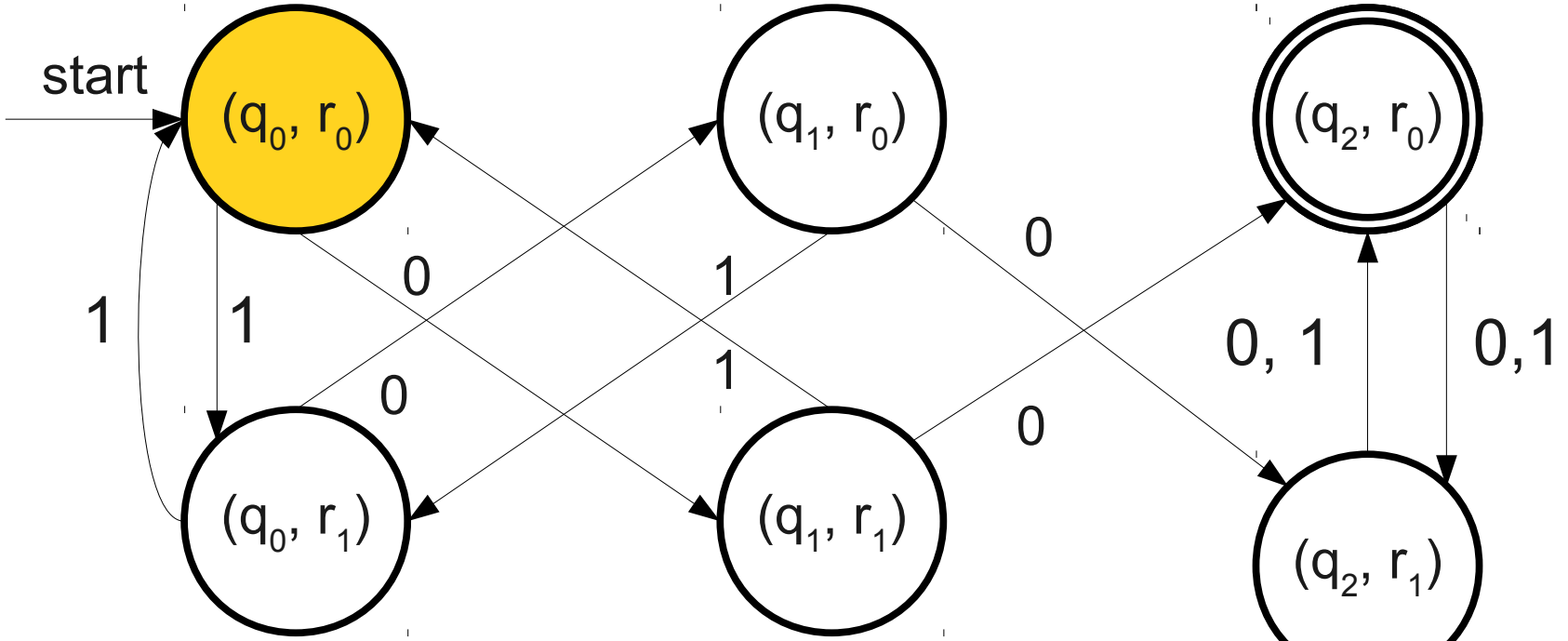
0 1 0 0 1

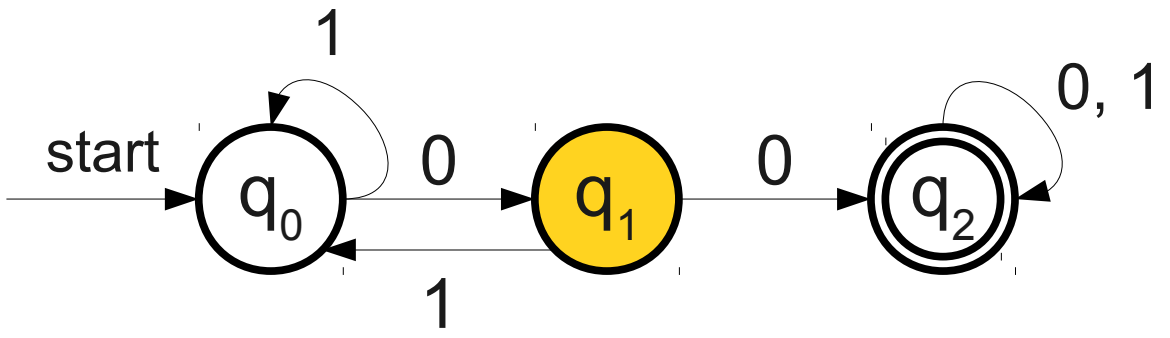


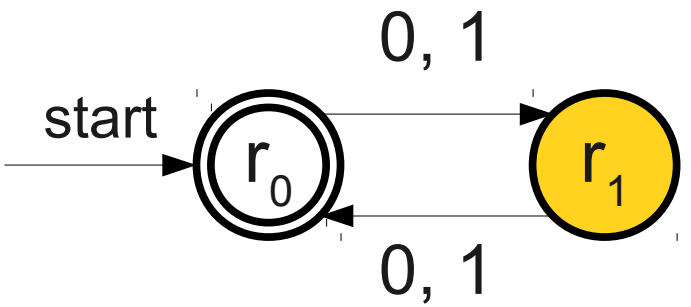


L_1 = { w | w contains 00 as a substring }

L_2 = { w | w has even length }

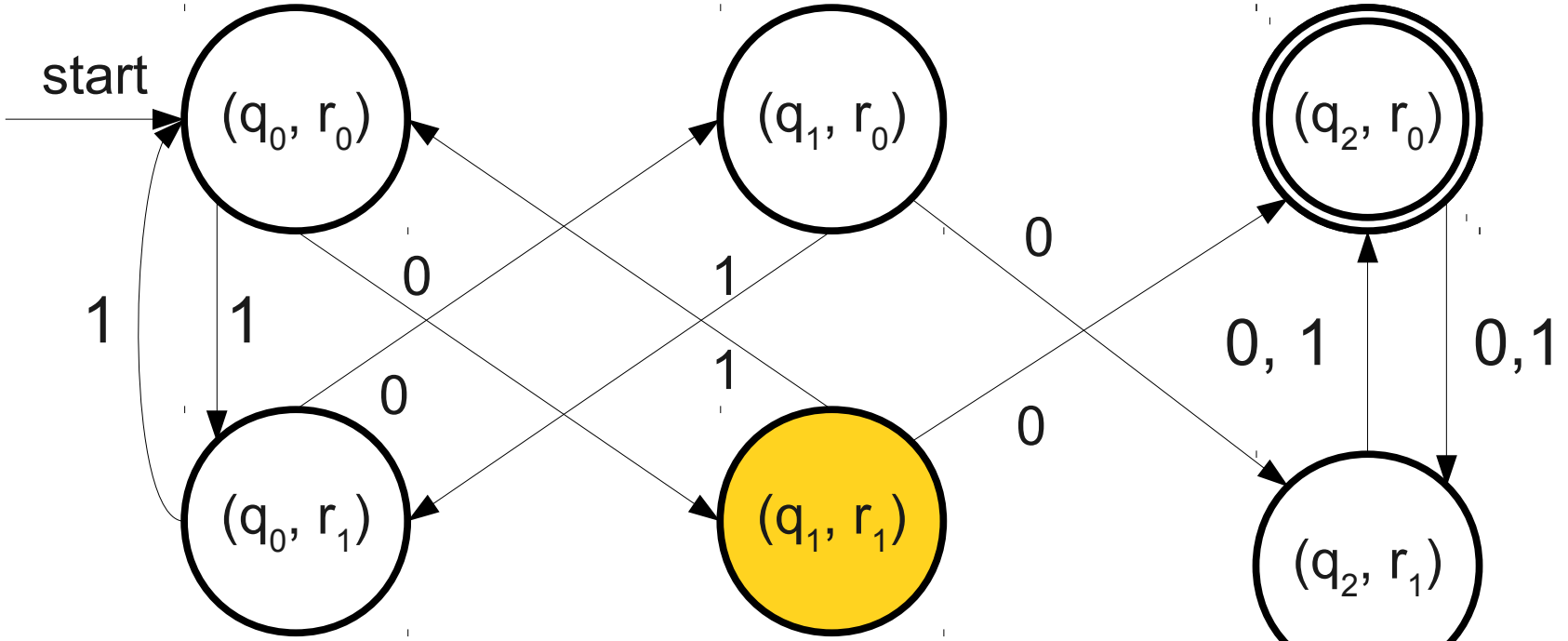


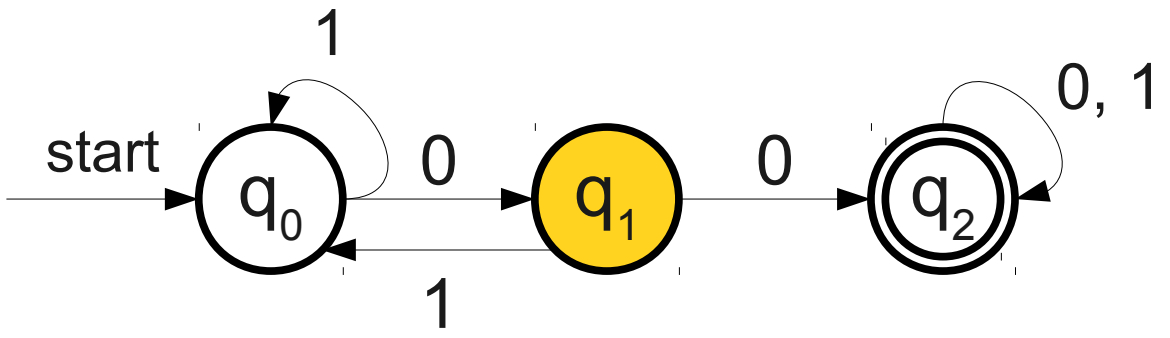


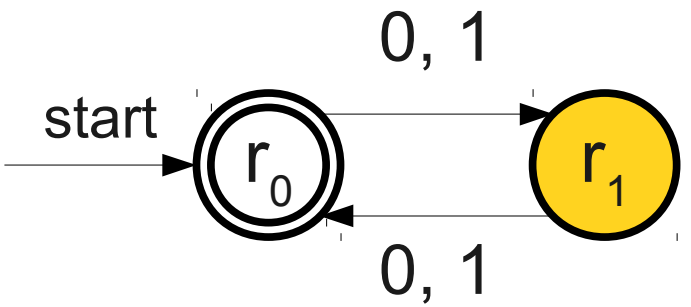


L_1 = { w | w contains 00 as a substring }

L_2 = { w | w has even length }

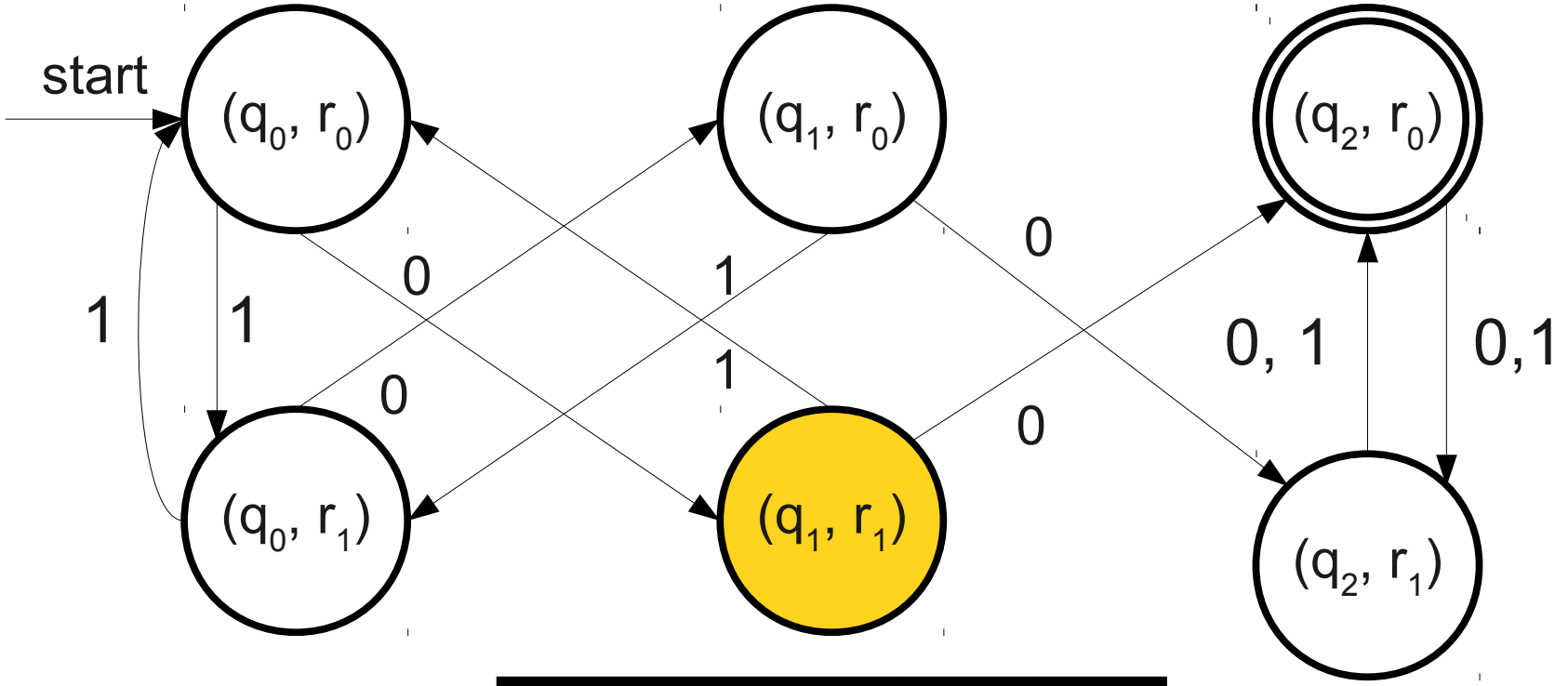


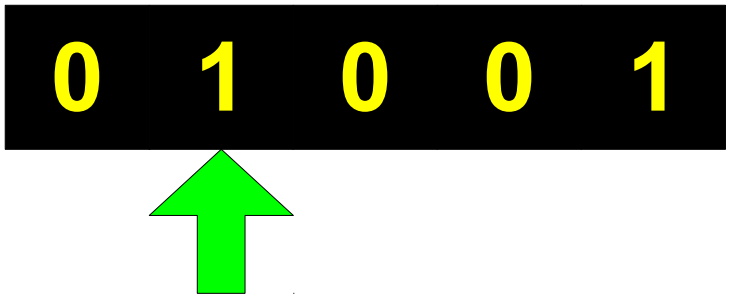


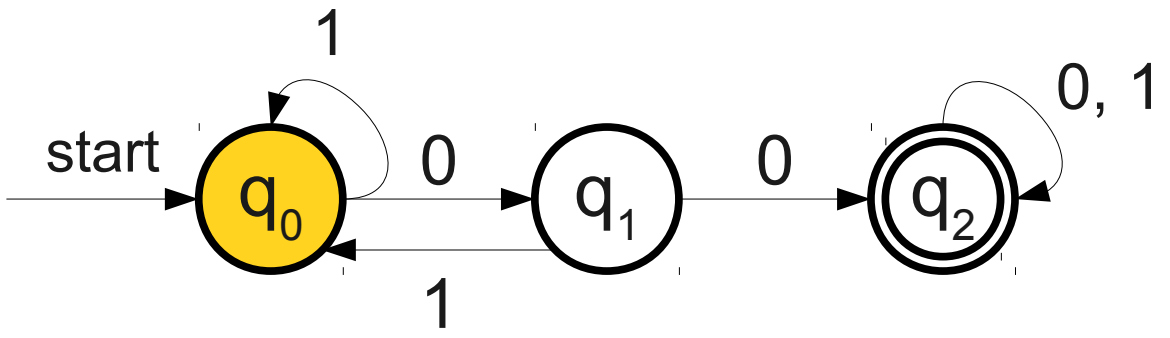


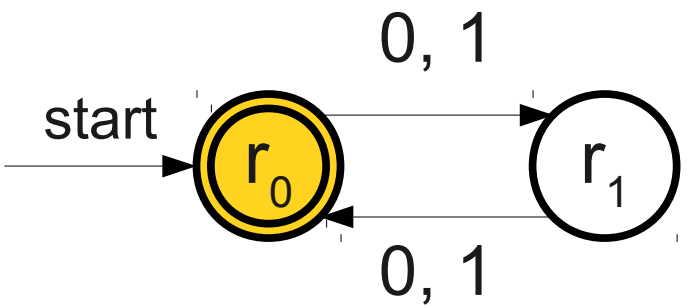
L_1 = { w | w contains 00 as a substring }

L_2 = { w | w has even length }



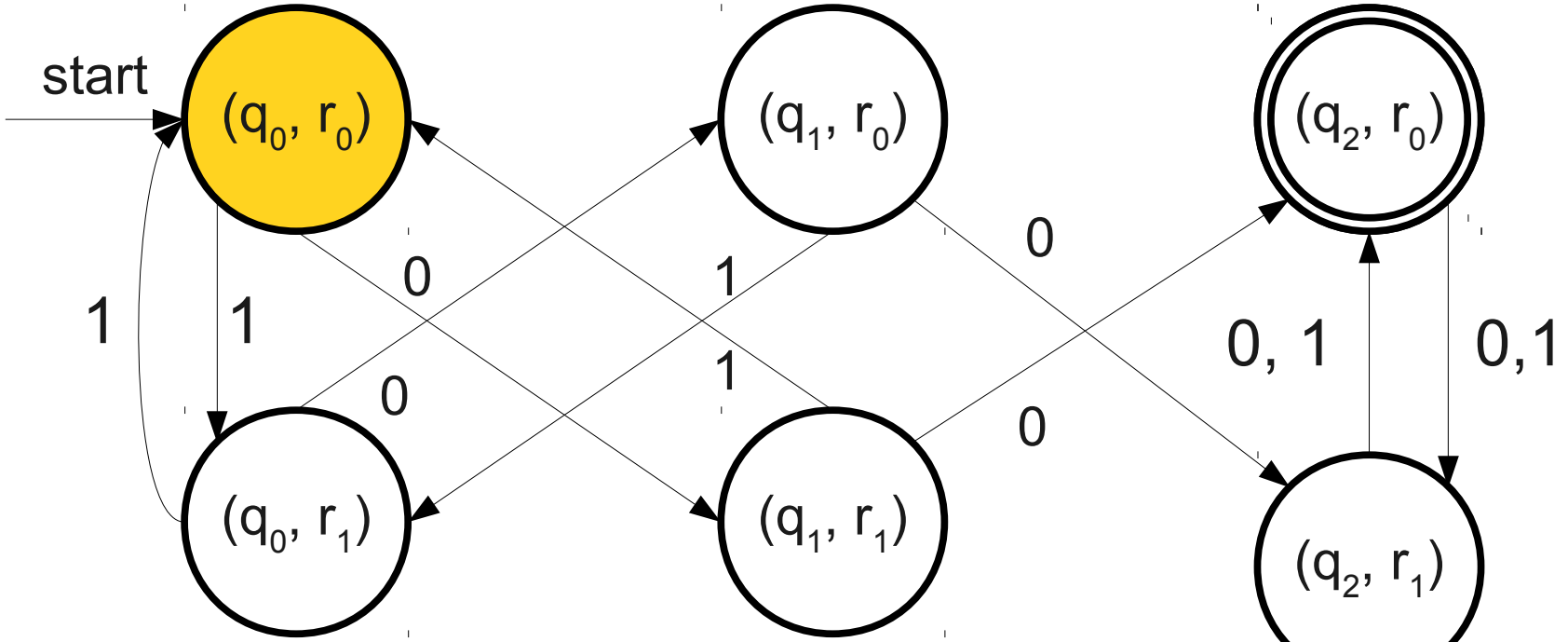


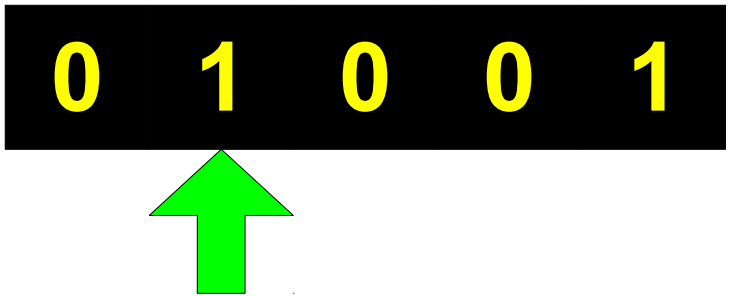


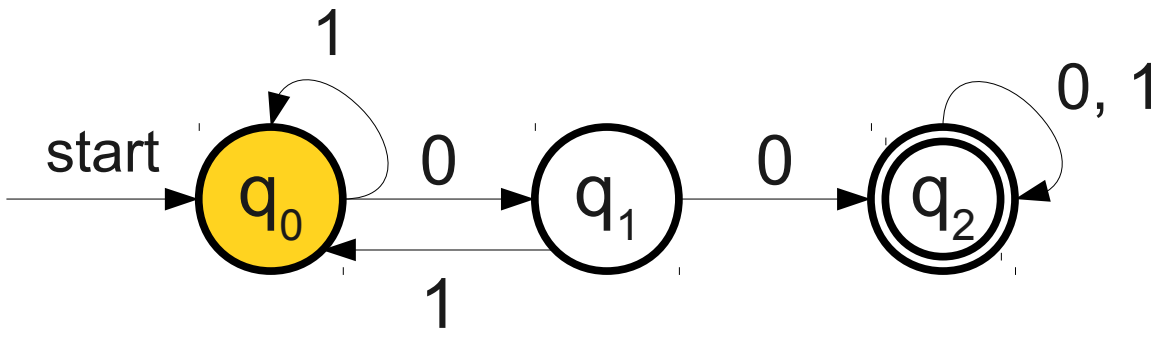


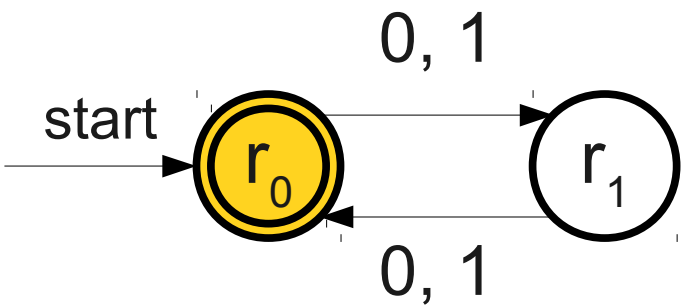
L_1 = { w | w contains 00 as a substring }

L_2 = { w | w has even length }



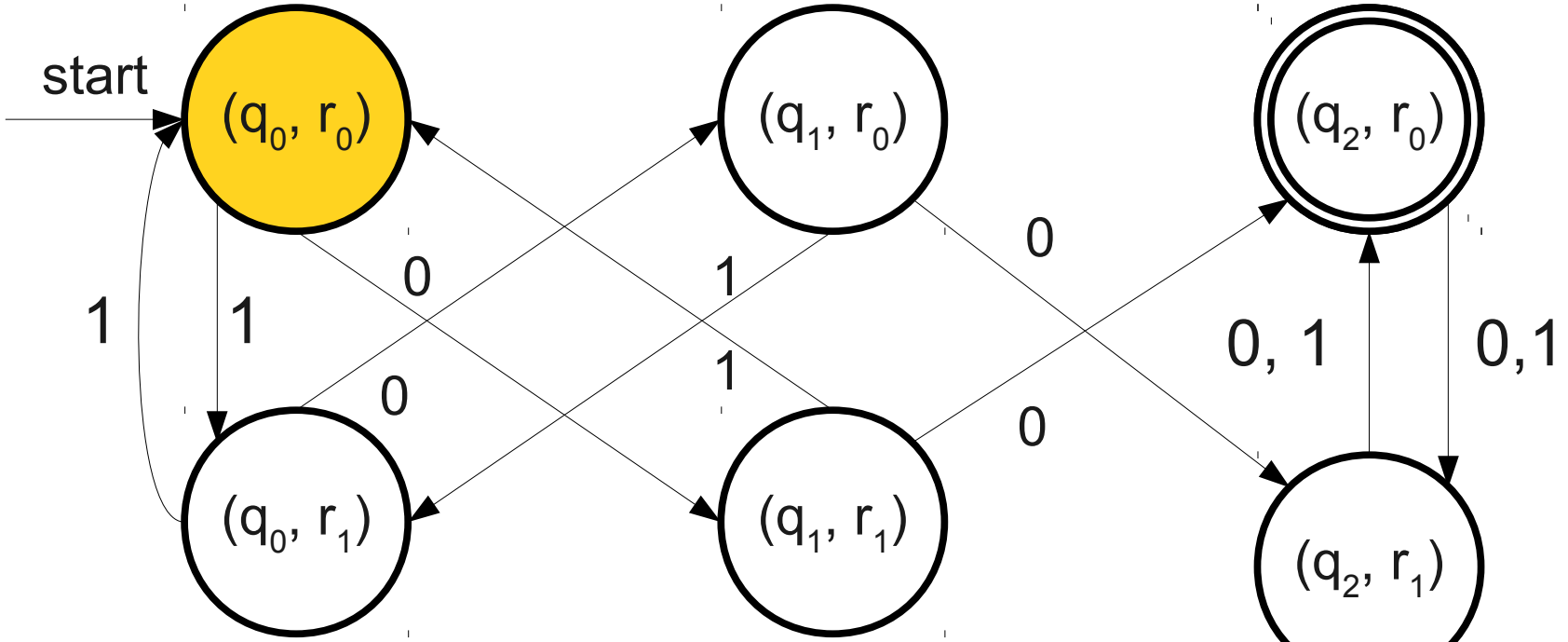


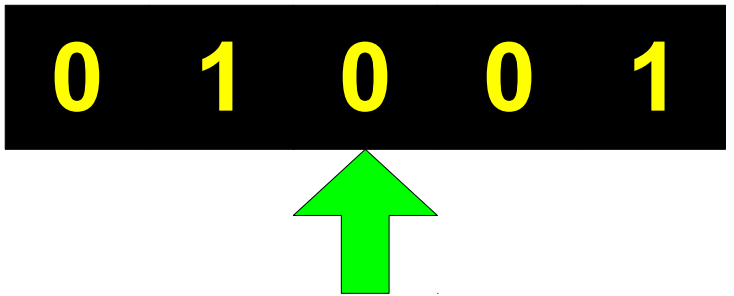


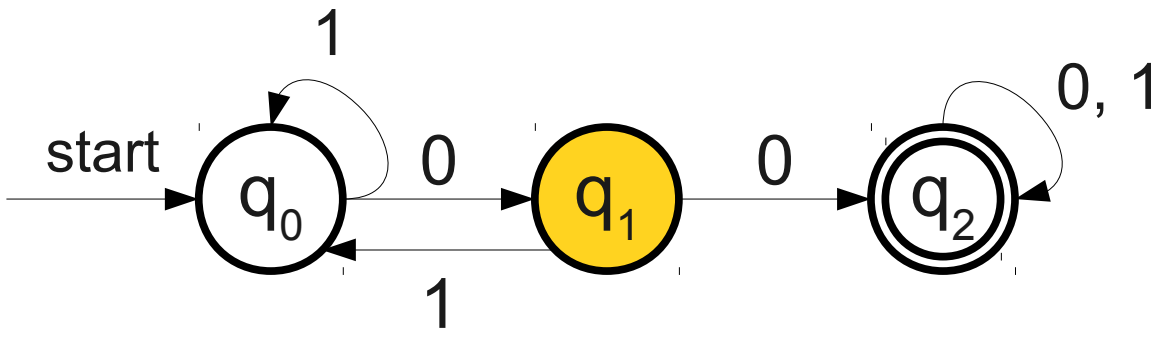


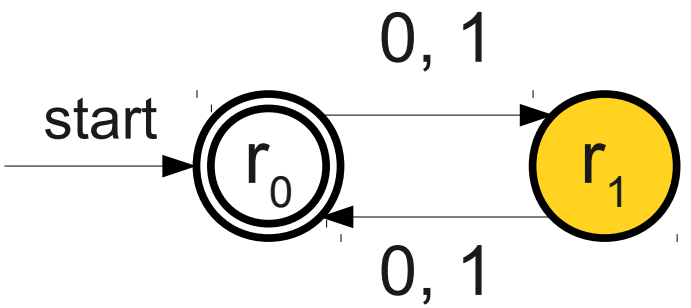
L_1 = { w | w contains 00 as a substring }

L_2 = { w | w has even length }



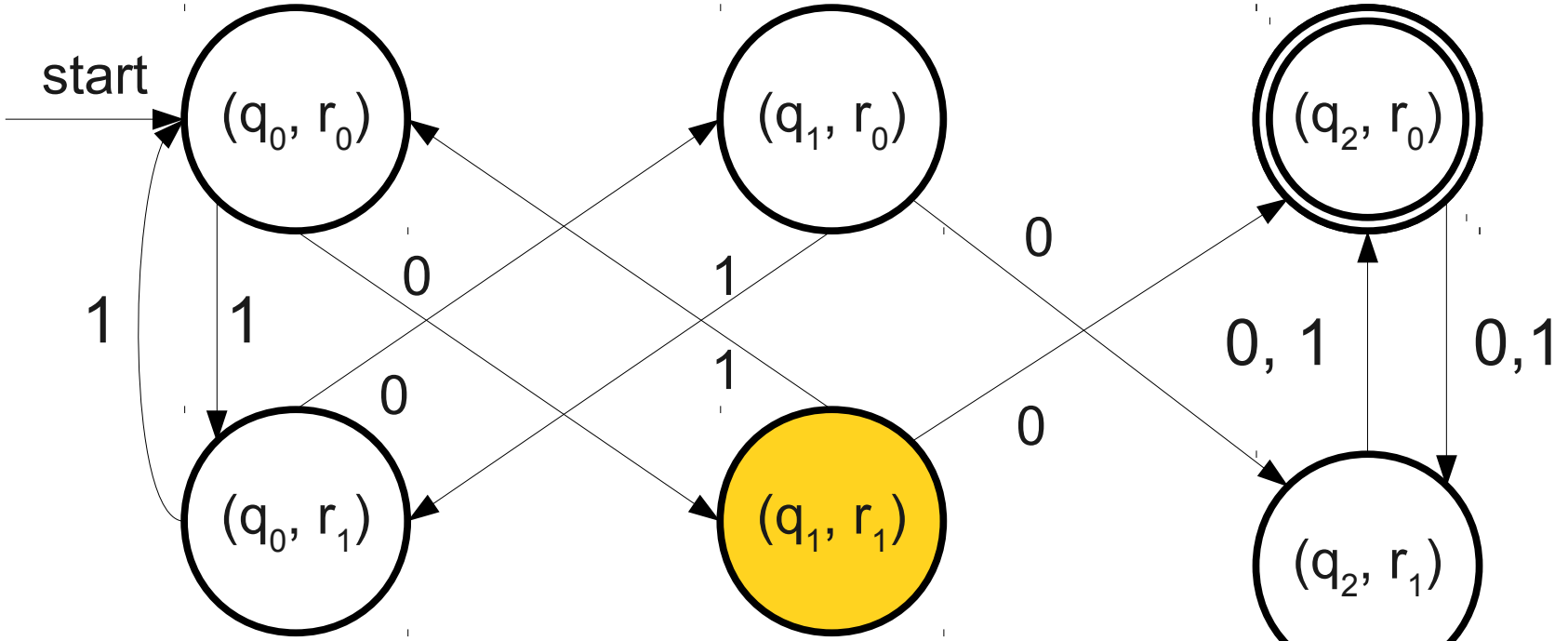


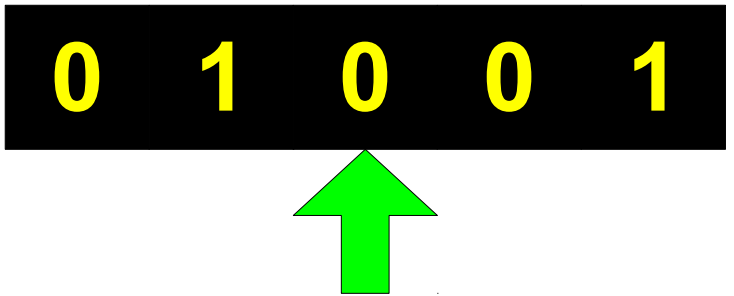


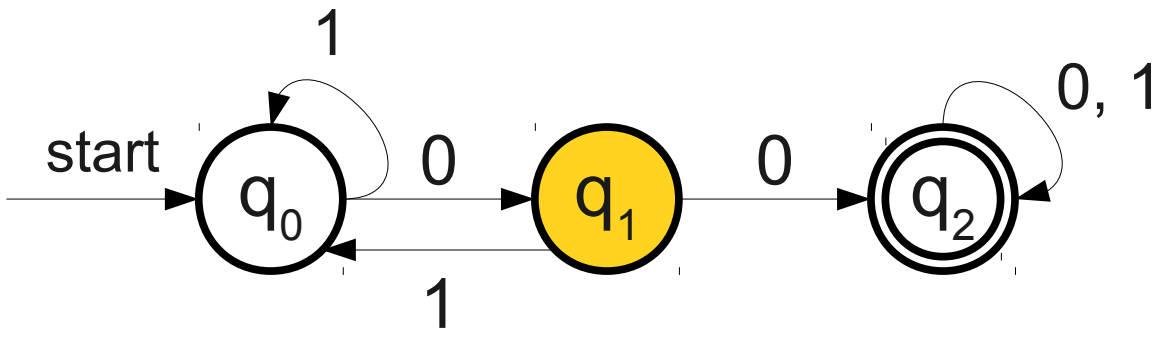


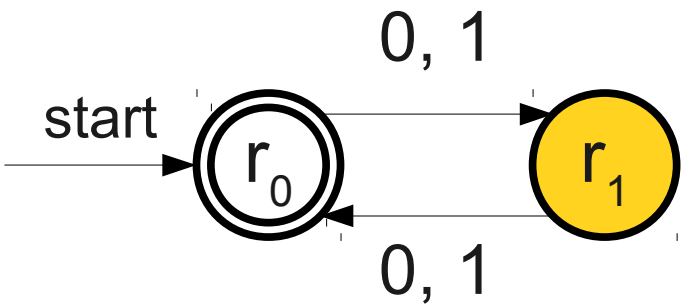
L_1 = { w | w contains 00 as a substring }

L_2 = { w | w has even length }



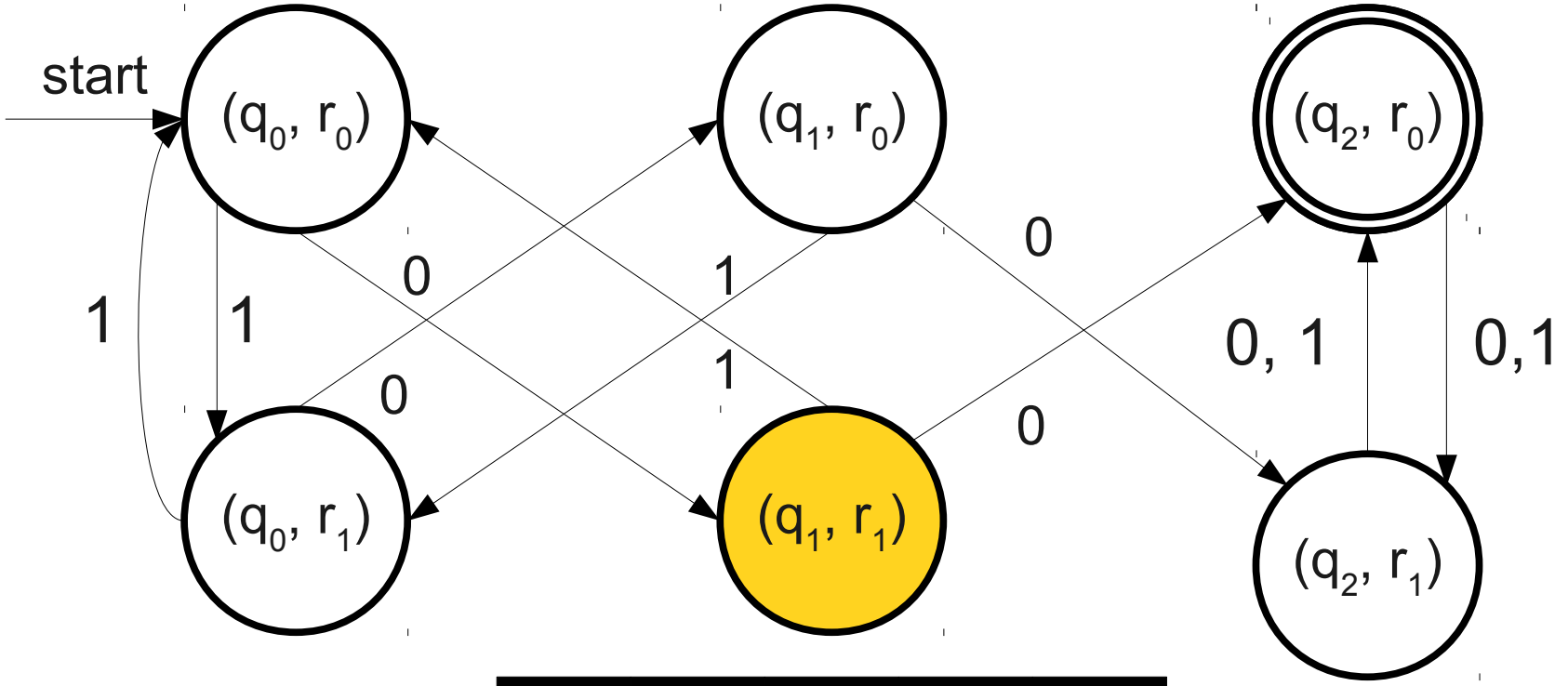


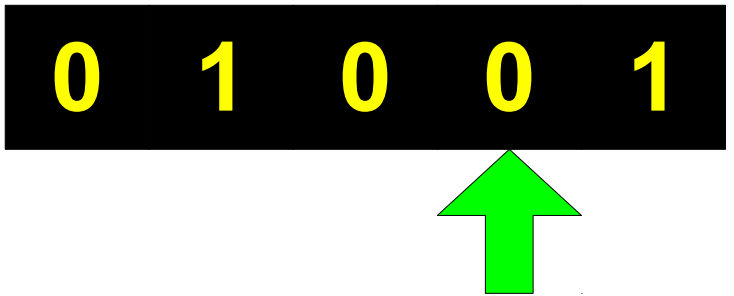


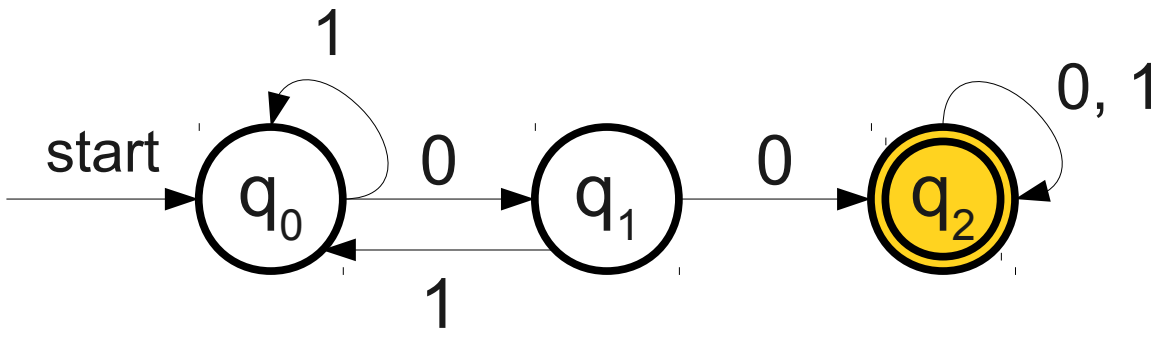


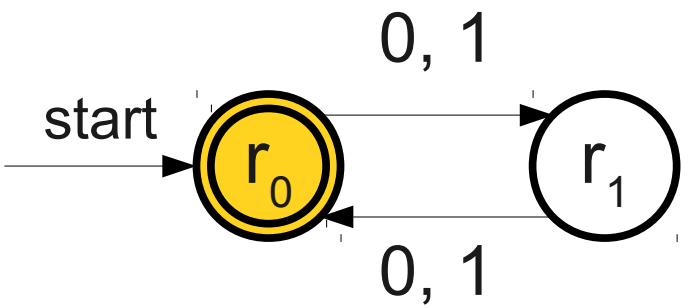
L_1 = { w | w contains 00 as a substring }

L_2 = { w | w has even length }



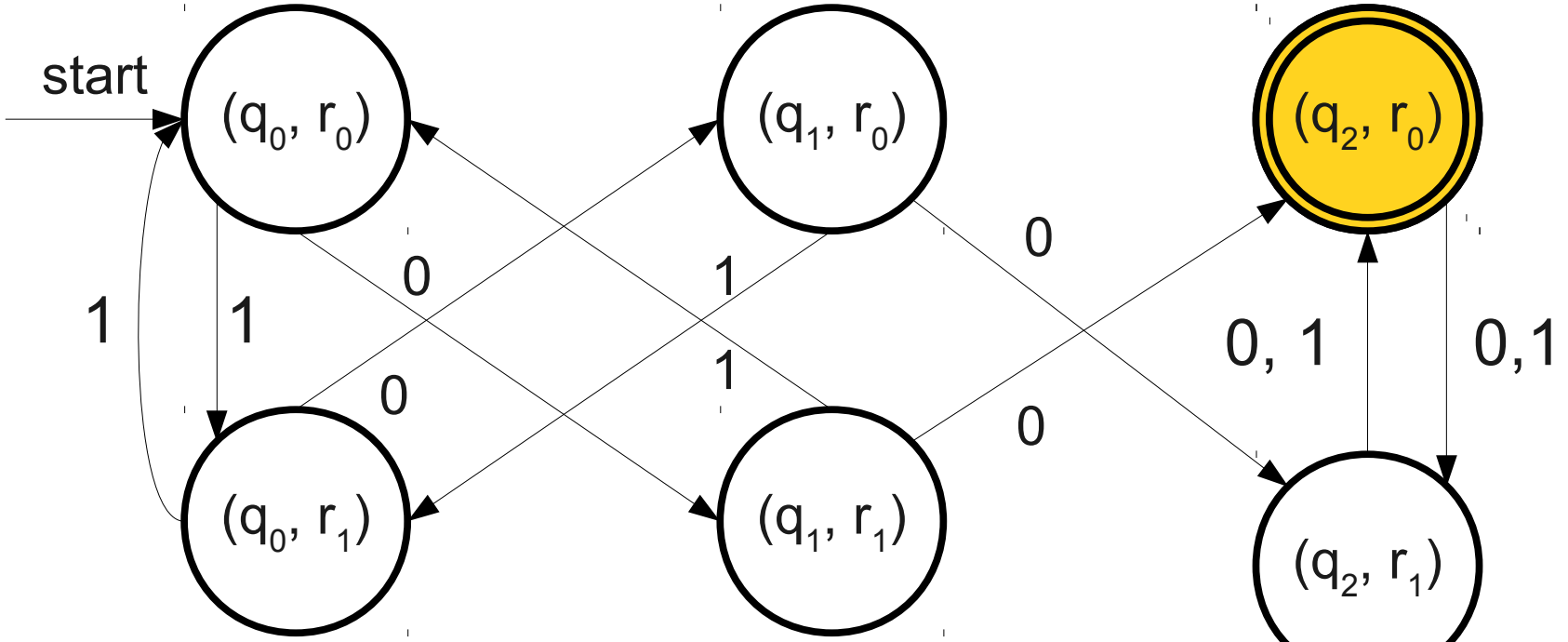


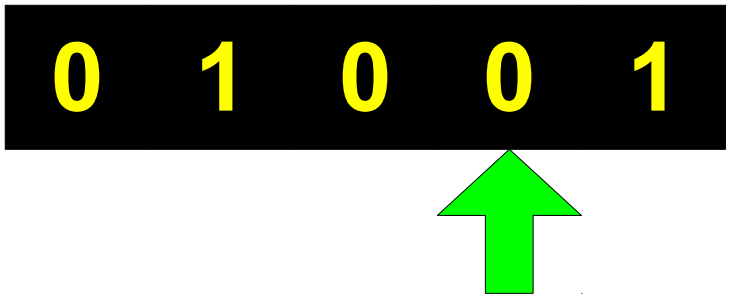


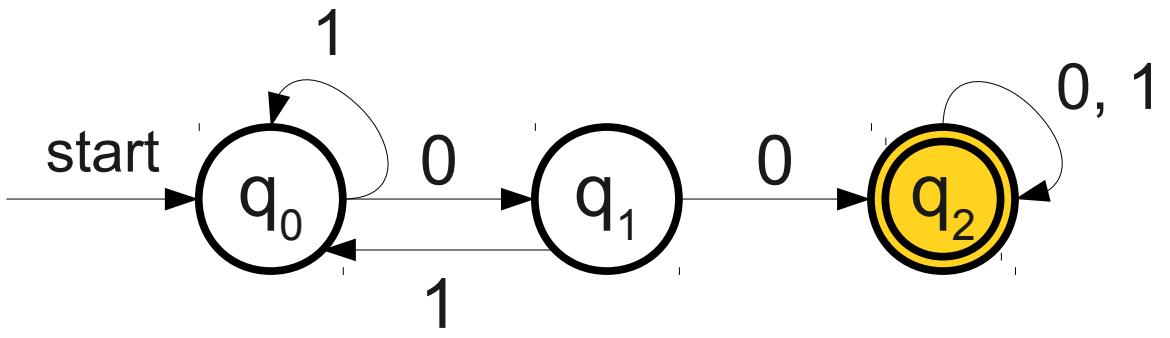


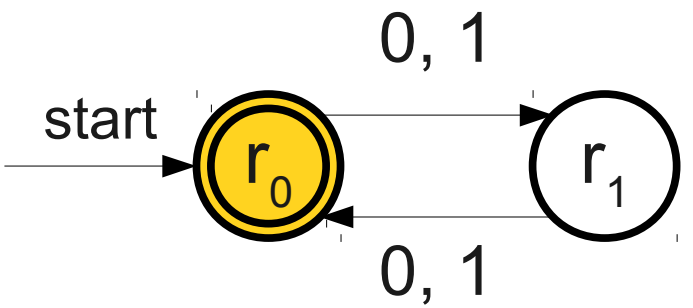
L_1 = { w | w contains 00 as a substring }

L_2 = { w | w has even length }



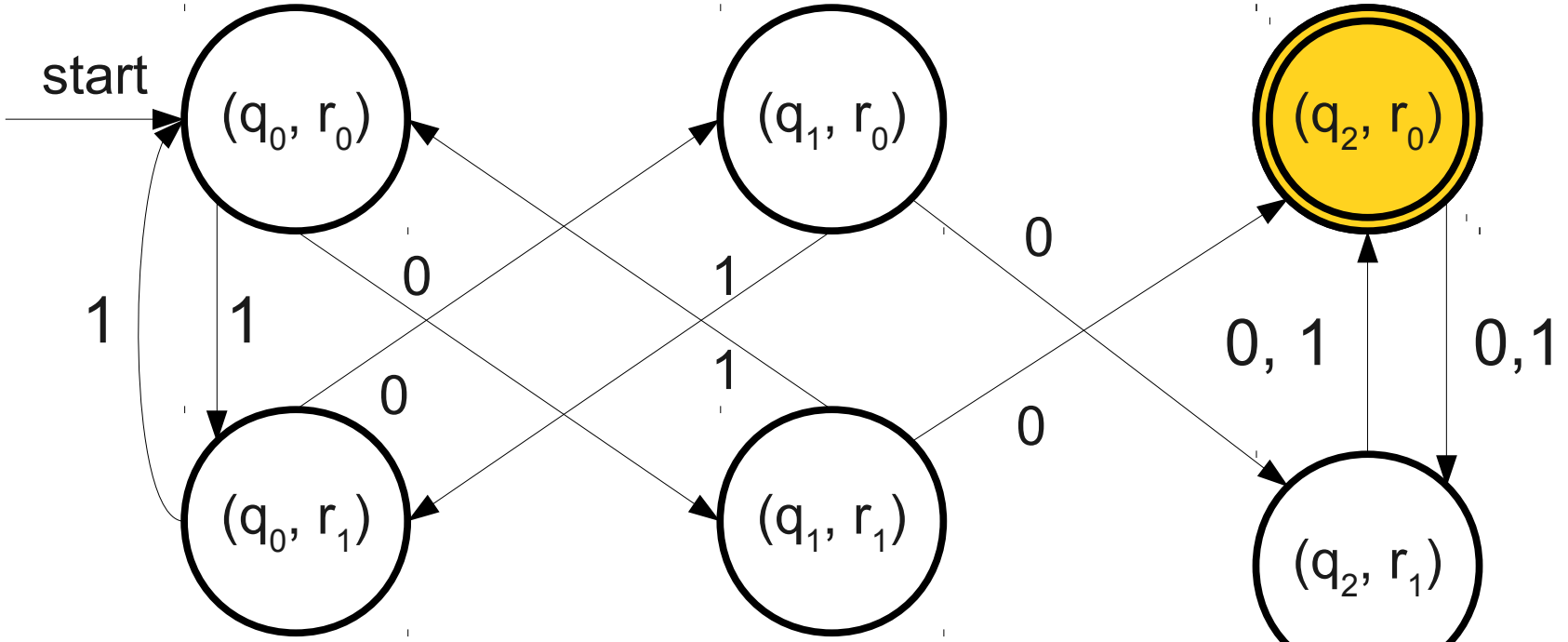


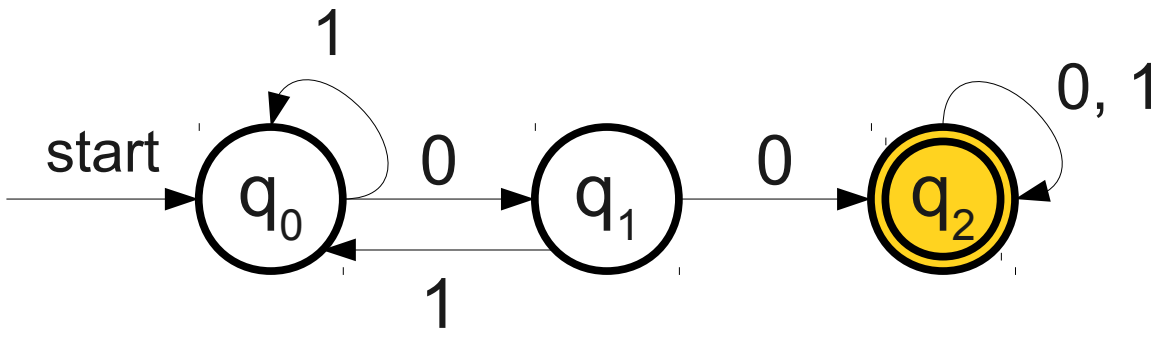


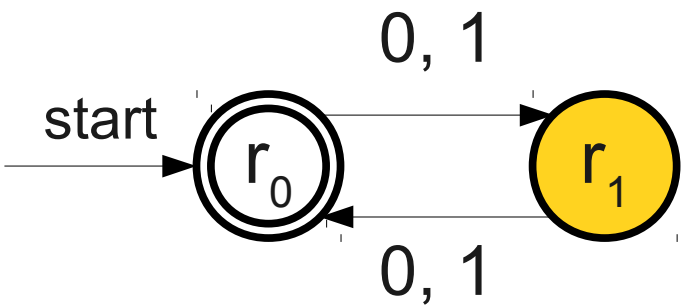


L_1 = { w | w contains 00 as a substring }

L_2 = { w | w has even length }

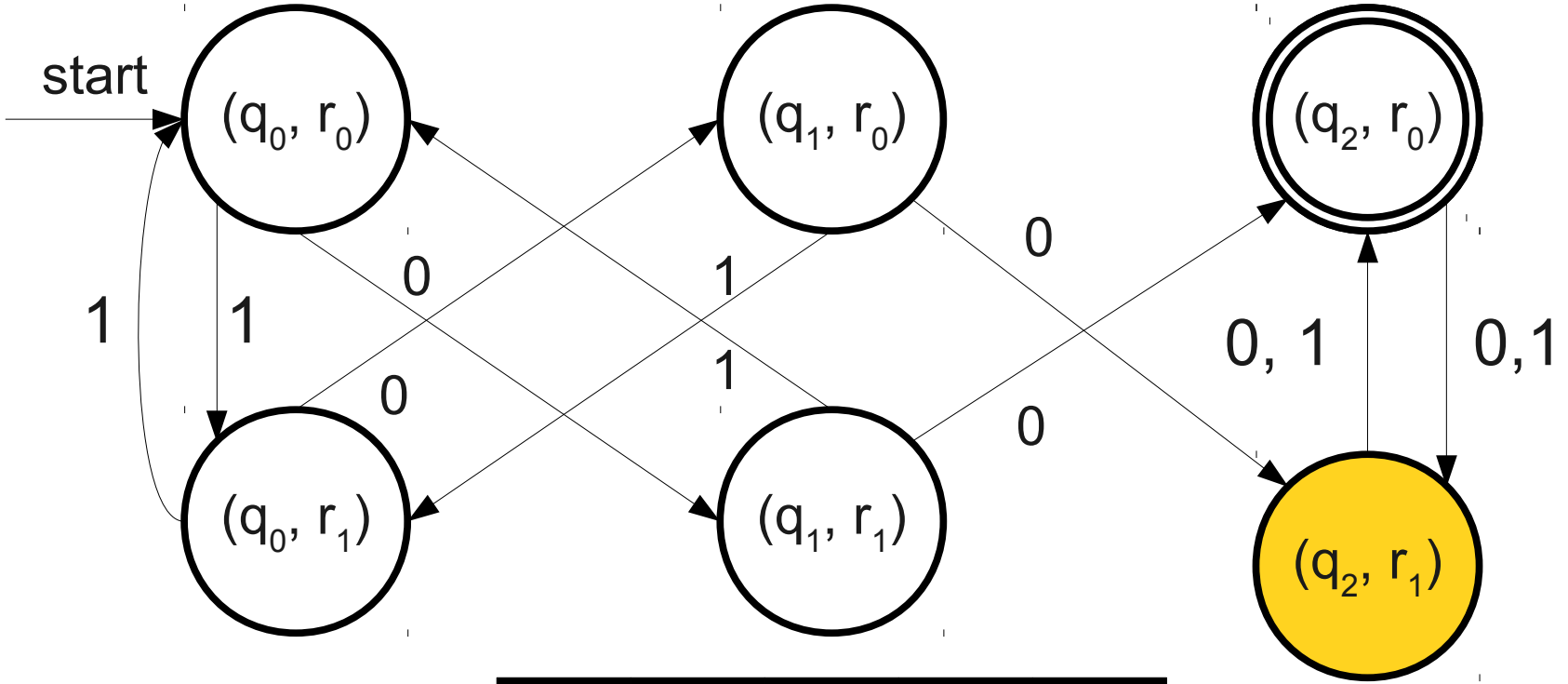


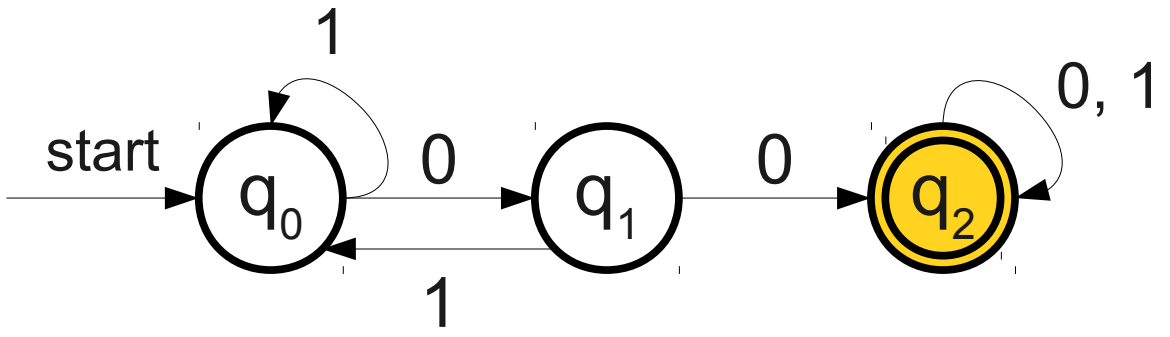


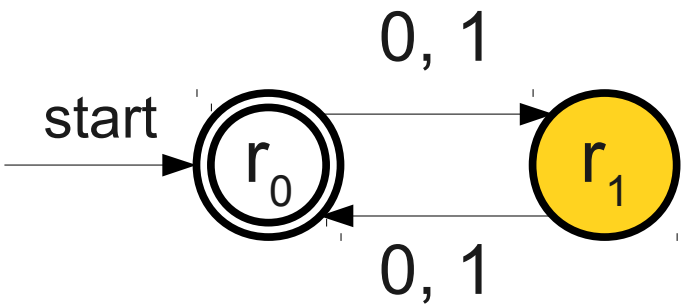


L_1 = { w | w contains 00 as a substring }

L_2 = { w | w has even length }

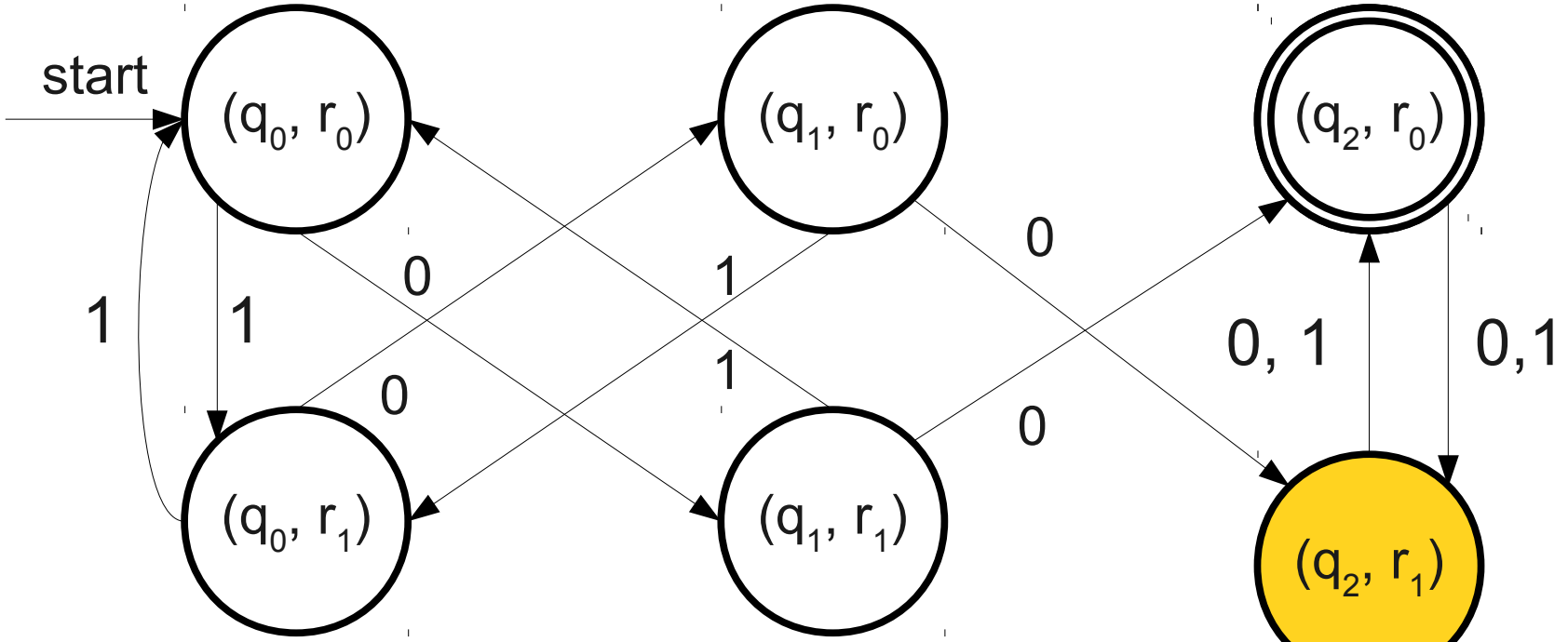




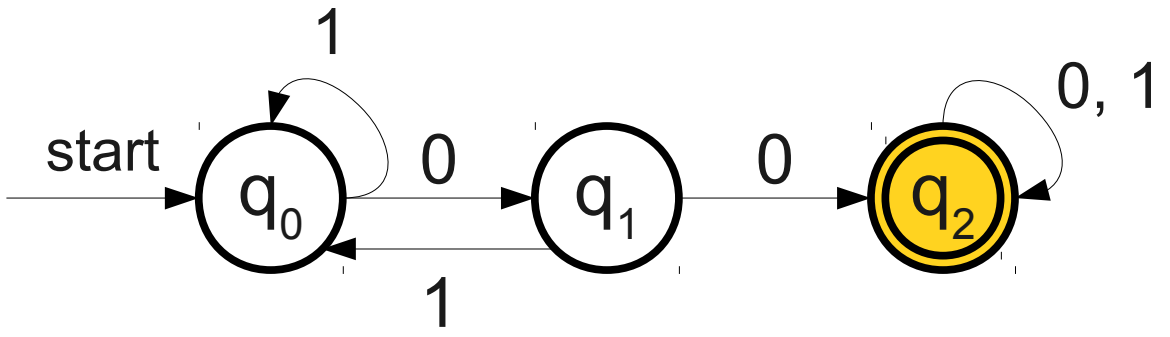


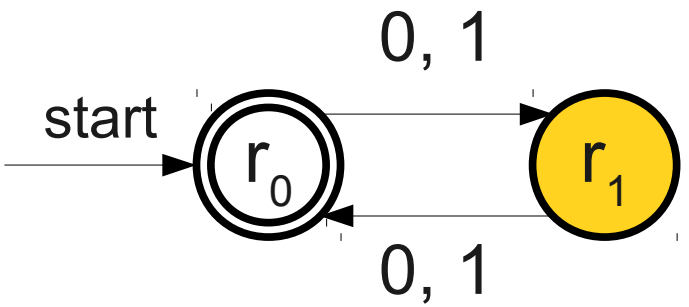
L_1 = { w | w contains 00 as a substring }

L_2 = { w | w has even length }



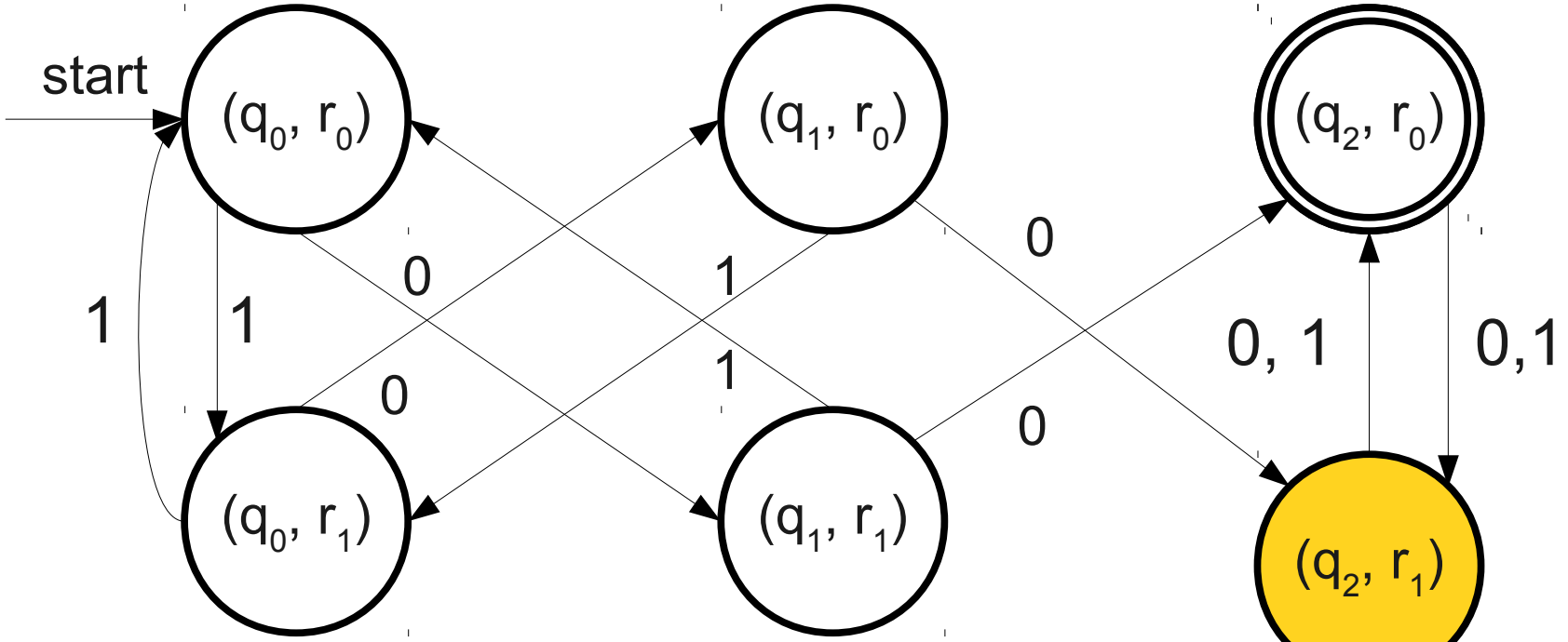
0 1 0 0 1





L_1 = { w | w contains 00 as a substring }

L_2 = { w | w has even length }



0 1 0 0 1

The Product Construction

- Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, the **product construction** constructs a new DFA $A \times B$ that accepts $L(A) \cap L(B)$.
- Formally, $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$, where
  - $Q \times R$ is the Cartesian product of the two sets of states.
  - Σ is the original alphabet.
  - $\delta'((q_i, r_j), a) = (\delta_A(q_i, a), \delta_B(r_j, a))$
  - (q_0, r_0) is the pair of the start states of A and B.
  - $F_A \times F_B$ is the set of pairs of accepting states of A and B.

The Product Construction

- The product construction runs both DFAs concurrently and accepts it both machines accept.
- Intuitively, computes the intersection by only accepting strings accepted by both machines.
- How would we formally prove that it works correctly?

Recall: Formal Acceptance

- Recall: If $D = (Q, \Sigma, \delta, q_0, F)$ is a DFA, then
$$L(D) = \{ w \mid \delta^*(w) \in F \}$$
- Here, δ^* is defined recursively:
  - $\delta^*(\varepsilon) = q_0$
  - $\delta^*(wa) = \delta(\delta^*(w), a)$
- Can we formally prove that $L(A \times B) = L(A) \cap L(B)$ using the definition?
- **Idea**: If we can show that the δ^* function for $A \times B$ correctly simulates A and B, we can conclude that $L(A \times B) = L(A) \cap L(B)$.

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

$$\delta_A^*(\varepsilon) = q_0$$
$$\delta_B^*(\varepsilon) = r_0$$
$$\delta_A^*(wa) = \delta_A(\delta_A^*(w), a)$$
$$\delta_B^*(wa) = \delta_B(\delta_B^*(w), a)$$

$$\delta'((q_i, r_j), a) = (\delta_A(q_i, a), \delta_B(r_j, a))$$

$$\delta'^*(\varepsilon) = (q_0, r_0)$$
$$\delta'^*(wa) = \delta'(\delta'^*(w), a)$$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*.

$$\delta_A^*(\varepsilon) = q_0$$
$$\delta_B^*(\varepsilon) = r_0$$
$$\delta_A^*(wa) = \delta_A(\delta_A^*(w), a)$$
$$\delta_B^*(wa) = \delta_B(\delta_B^*(w), a)$$

$$\delta'((q_i, r_j), a) = (\delta_A(q_i, a), \delta_B(r_j, a))$$

$$\delta'^*(\varepsilon) = (q_0, r_0)$$
$$\delta'^*(wa) = \delta'(\delta'^*(w), a)$$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

$$\delta_A^*(\varepsilon) = q_0$$
$$\delta_B^*(\varepsilon) = r_0$$
$$\delta_A^*(wa) = \delta_A(\delta_A^*(w), a)$$
$$\delta_B^*(wa) = \delta_B(\delta_B^*(w), a)$$

$$\delta'((q_i, r_j), a) = (\delta_A(q_i, a), \delta_B(r_j, a))$$

$$\delta'^*(\varepsilon) = (q_0, r_0)$$
$$\delta'^*(wa) = \delta'(\delta'^*(w), a)$$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε.

$$\delta_A^*(\varepsilon) = q_0$$
$$\delta_B^*(\varepsilon) = r_0$$
$$\delta_A^*(wa) = \delta_A(\delta_A^*(w), a)$$
$$\delta_B^*(wa) = \delta_B(\delta_B^*(w), a)$$

$$\delta'((q_i, r_j), a) = (\delta_A(q_i, a), \delta_B(r_j, a))$$

$$\delta'^*(\varepsilon) = (q_0, r_0)$$
$$\delta'^*(wa) = \delta'(\delta'^*(w), a)$$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε. By definition, $\delta_A^*(\varepsilon) = q_0$ and $\delta_A^*(\varepsilon) = r_0$.

$$\delta_A^*(\varepsilon) = q_0$$
$$\delta_B^*(\varepsilon) = r_0$$
$$\delta_A^*(wa) = \delta_A(\delta_A^*(w), a)$$
$$\delta_B^*(wa) = \delta_B(\delta_B^*(w), a)$$

$$\delta'((q_i, r_j), a) = (\delta_A(q_i, a), \delta_B(r_j, a))$$

$$\delta'^*(\varepsilon) = (q_0, r_0)$$
$$\delta'^*(wa) = \delta'(\delta'^*(w), a)$$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε. By definition, $\delta_A^*(\varepsilon) = q_0$ and $\delta_A^*(\varepsilon) = r_0$. Also by definition, $\delta'^*(\varepsilon) = (q_0, r_0)$.

$$\delta_A^*(\varepsilon) = q_0$$
$$\delta_B^*(\varepsilon) = r_0$$
$$\delta_A^*(wa) = \delta_A(\delta_A^*(w), a)$$
$$\delta_B^*(wa) = \delta_B(\delta_B^*(w), a)$$

$$\delta'((q_i, r_j), a) = (\delta_A(q_i, a), \delta_B(r_j, a))$$

$$\delta'^*(\varepsilon) = (q_0, r_0)$$
$$\delta'^*(wa) = \delta'(\delta'^*(w), a)$$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε. By definition, $\delta_A^*(\varepsilon) = q_0$ and $\delta_A^*(\varepsilon) = r_0$. Also by definition, $\delta'^*(\varepsilon) = (q_0, r_0)$. Thus $\delta'^*(\varepsilon) = (q_0, r_0) = (\delta_A^*(\varepsilon), \delta_B^*(\varepsilon))$.

$$\delta_A^*(\varepsilon) = q_0$$
$$\delta_B^*(\varepsilon) = r_0$$
$$\delta_A^*(wa) = \delta_A(\delta_A^*(w), a)$$
$$\delta_B^*(wa) = \delta_B(\delta_B^*(w), a)$$

$$\delta'((q_i, r_j), a) = (\delta_A(q_i, a), \delta_B(r_j, a))$$

$$\delta'^*(\varepsilon) = (q_0, r_0)$$
$$\delta'^*(wa) = \delta'(\delta'^*(w), a)$$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε. By definition, $\delta_A^*(\varepsilon) = q_0$ and $\delta_A^*(\varepsilon) = r_0$. Also by definition, $\delta'^*(\varepsilon) = (q_0, r_0)$. Thus $\delta'^*(\varepsilon) = (q_0, r_0) = (\delta_A^*(\varepsilon), \delta_B^*(\varepsilon))$.

$\delta'((q_i, r_j), a) = (\delta_A(q_i, a), \delta_B(r_j, a))$

$\delta'^*(\varepsilon) = (q_0, r_0)$

$\delta'^*(wa) = \delta'(\delta'^*(w), a)$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε. By definition, $\delta_A^*(\varepsilon) = q_0$ and $\delta_A^*(\varepsilon) = r_0$. Also by definition, $\delta'^*(\varepsilon) = (q_0, r_0)$. Thus $\delta'^*(\varepsilon) = (q_0, r_0) = (\delta_A^*(\varepsilon), \delta_B^*(\varepsilon))$.

For the inductive step, assume that for some natural number n, for any string *w* of length n, the theorem is true.

$\delta'((q_i, r_j), a) = (\delta_A(q_i, a), \delta_B(r_j, a))$

$\delta'^*(\varepsilon) = (q_0, r_0)$

$\delta'^*(wa) = \delta'(\delta'^*(w), a)$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε. By definition, $\delta_A^*(\varepsilon) = q_0$ and $\delta_A^*(\varepsilon) = r_0$. Also by definition, $\delta'^*(\varepsilon) = (q_0, r_0)$. Thus $\delta'^*(\varepsilon) = (q_0, r_0) = (\delta_A^*(\varepsilon), \delta_B^*(\varepsilon))$.

For the inductive step, assume that for some natural number n, for any string *w* of length n, the theorem is true. We prove that the theorem is true for any string of length n + 1.

$\delta'((q_i, r_j), a) = (\delta_A(q_i, a), \delta_B(r_j, a))$

$\delta'^*(\varepsilon) = (q_0, r_0)$

$\delta'^*(wa) = \delta'(\delta'^*(w), a)$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε. By definition, $\delta_A^*(\varepsilon) = q_0$ and $\delta_A^*(\varepsilon) = r_0$. Also by definition, $\delta'^*(\varepsilon) = (q_0, r_0)$. Thus $\delta'^*(\varepsilon) = (q_0, r_0) = (\delta_A^*(\varepsilon), \delta_B^*(\varepsilon))$.

For the inductive step, assume that for some natural number n, for any string *w* of length n, the theorem is true. We prove that the theorem is true for any string of length n + 1. Consider any string of length n + 1; it must have the form *wa* for some string *w* of length n and some character a.

$\delta'((q_i, r_j), a) = (\delta_A(q_i, a), \delta_B(r_j, a))$

$\delta'^*(\varepsilon) = (q_0, r_0)$

$\delta'^*(wa) = \delta'(\delta'^*(w), a)$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε. By definition, $\delta_A^*(\varepsilon) = q_0$ and $\delta_A^*(\varepsilon) = r_0$. Also by definition, $\delta'^*(\varepsilon) = (q_0, r_0)$. Thus $\delta'^*(\varepsilon) = (q_0, r_0) = (\delta_A^*(\varepsilon), \delta_B^*(\varepsilon))$.

For the inductive step, assume that for some natural number n, for any string *w* of length n, the theorem is true. We prove that the theorem is true for any string of length n + 1. Consider any string of length n + 1; it must have the form *wa* for some string *w* of length n and some character a. Then by definition

$\delta'^*(wa) = \delta'(\delta'^*(w), a)$

$\delta'((q_i, r_j), a) = (\delta_A(q_i, a), \delta_B(r_j, a))$

$\delta'^*(\varepsilon) = (q_0, r_0)$

$\delta'^*(wa) = \delta'(\delta'^*(w), a)$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε. By definition, $\delta_A^*(\varepsilon) = q_0$ and $\delta_A^*(\varepsilon) = r_0$. Also by definition, $\delta'^*(\varepsilon) = (q_0, r_0)$. Thus $\delta'^*(\varepsilon) = (q_0, r_0) = (\delta_A^*(\varepsilon), \delta_B^*(\varepsilon))$.

For the inductive step, assume that for some natural number n, for any string *w* of length n, the theorem is true. We prove that the theorem is true for any string of length n + 1. Consider any string of length n + 1; it must have the form *wa* for some string *w* of length n and some character a. Then by definition

$$\begin{aligned}\delta'^*(wa) &= \delta'(\delta'^*(w), a) \\ &= \delta'((\delta_A^*(w), \delta_B^*(w)), a)\end{aligned}$$

$\delta'((q_i, r_j), a) = (\delta_A(q_i, a), \delta_B(r_j, a))$

$\delta'^*(\varepsilon) = (q_0, r_0)$

$\delta'^*(wa) = \delta'(\delta'^*(w), a)$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε. By definition, $\delta_A^*(\varepsilon) = q_0$ and $\delta_A^*(\varepsilon) = r_0$. Also by definition, $\delta'^*(\varepsilon) = (q_0, r_0)$. Thus $\delta'^*(\varepsilon) = (q_0, r_0) = (\delta_A^*(\varepsilon), \delta_B^*(\varepsilon))$.

For the inductive step, assume that for some natural number n, for any string *w* of length n, the theorem is true. We prove that the theorem is true for any string of length n + 1. Consider any string of length n + 1; it must have the form *wa* for some string *w* of length n and some character a. Then by definition

$$\begin{aligned}\delta'^*(wa) &= \delta'(\delta'^*(w), a) \\ &= \delta'((\delta_A^*(w), \delta_B^*(w)), a) \\ &= (\delta_A(\delta_A^*(w), a), \delta_B(\delta_B^*(w), a))\end{aligned}$$

$\delta'((q_i, r_j), a) = (\delta_A(q_i, a), \delta_B(r_j, a))$

$\delta'^*(\varepsilon) = (q_0, r_0)$

$\delta'^*(wa) = \delta'(\delta'^*(w), a)$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε. By definition, $\delta_A^*(\varepsilon) = q_0$ and $\delta_A^*(\varepsilon) = r_0$. Also by definition, $\delta'^*(\varepsilon) = (q_0, r_0)$. Thus $\delta'^*(\varepsilon) = (q_0, r_0) = (\delta_A^*(\varepsilon), \delta_B^*(\varepsilon))$.

For the inductive step, assume that for some natural number n, for any string *w* of length n, the theorem is true. We prove that the theorem is true for any string of length n + 1. Consider any string of length n + 1; it must have the form *wa* for some string *w* of length n and some character a. Then by definition

$$\begin{aligned}\delta'^*(wa) &= \delta'(\delta'^*(w), a) \\ &= \delta'((\delta_A^*(w), \delta_B^*(w)), a) \\ &= (\delta_A(\delta_A^*(w), a), \delta_B(\delta_B^*(w), a))\end{aligned}$$

$$\boldsymbol{\delta_A^*(\varepsilon) = q_0}$$
$$\boldsymbol{\delta_B^*(\varepsilon) = r_0}$$
$$\boldsymbol{\delta_A^*(wa) = \delta_A(\delta_A^*(w), a)}$$
$$\boldsymbol{\delta_B^*(wa) = \delta_B(\delta_B^*(w), a)}$$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε. By definition, $\delta_A^*(\varepsilon) = q_0$ and $\delta_A^*(\varepsilon) = r_0$. Also by definition, $\delta'^*(\varepsilon) = (q_0, r_0)$. Thus $\delta'^*(\varepsilon) = (q_0, r_0) = (\delta_A^*(\varepsilon), \delta_B^*(\varepsilon))$.

For the inductive step, assume that for some natural number n, for any string *w* of length n, the theorem is true. We prove that the theorem is true for any string of length n + 1. Consider any string of length n + 1; it must have the form *wa* for some string *w* of length n and some character a. Then by definition

$$\begin{aligned}\delta'^*(wa) &= \delta'(\delta'^*(w), a) \\ &= \delta'((\delta_A^*(w), \delta_B^*(w)), a) \\ &= (\delta_A(\delta_A^*(w), a), \delta_B(\delta_B^*(w), a)) \\ &= (\delta_A^*(wa), \delta_B^*(wa))\end{aligned}$$

$$\delta_A^*(\varepsilon) = q_0$$
$$\delta_B^*(\varepsilon) = r_0$$
$$\delta_A^*(wa) = \delta_A(\delta_A^*(w), a)$$
$$\delta_B^*(wa) = \delta_B(\delta_B^*(w), a)$$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε. By definition, $\delta_A^*(\varepsilon) = q_0$ and $\delta_A^*(\varepsilon) = r_0$. Also by definition, $\delta'^*(\varepsilon) = (q_0, r_0)$. Thus $\delta'^*(\varepsilon) = (q_0, r_0) = (\delta_A^*(\varepsilon), \delta_B^*(\varepsilon))$.

For the inductive step, assume that for some natural number n, for any string *w* of length n, the theorem is true. We prove that the theorem is true for any string of length n + 1. Consider any string of length n + 1; it must have the form *w*a for some string *w* of length n and some character a. Then by definition

$$\begin{aligned}
\delta'^*(wa) &= \delta'(\delta'^*(w), a) \\
&= \delta'((\delta_A^*(w), \delta_B^*(w)), a) \\
&= (\delta_A(\delta_A^*(w), a), \delta_B(\delta_B^*(w), a)) \\
&= (\delta_A^*(wa), \delta_B^*(wa))
\end{aligned}$$

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε. By definition, $\delta_A^*(\varepsilon) = q_0$ and $\delta_A^*(\varepsilon) = r_0$. Also by definition, $\delta'^*(\varepsilon) = (q_0, r_0)$. Thus $\delta'^*(\varepsilon) = (q_0, r_0) = (\delta_A^*(\varepsilon), \delta_B^*(\varepsilon))$.

For the inductive step, assume that for some natural number n, for any string *w* of length n, the theorem is true. We prove that the theorem is true for any string of length n + 1. Consider any string of length n + 1; it must have the form *w*a for some string *w* of length n and some character a. Then by definition

$$\begin{aligned}
\delta'^*(wa) &= \delta'(\delta'^*(w), a) \\
&= \delta'((\delta_A^*(w), \delta_B^*(w)), a) \\
&= (\delta_A(\delta_A^*(w), a), \delta_B(\delta_B^*(w), a)) \\
&= (\delta_A^*(wa), \delta_B^*(wa))
\end{aligned}$$

Thus the theorem holds for strings of length n + 1, completing the induction.

Theorem: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$.

Proof: By induction on the length of the string *w*. As our base case, we prove that any string of length 0 satisfies $\delta'^*(w) = (\delta_A^*(w), \delta_B^*(w))$. There is only one string of length 0, which is ε. By definition, $\delta_A^*(\varepsilon) = q_0$ and $\delta_A^*(\varepsilon) = r_0$. Also by definition, $\delta'^*(\varepsilon) = (q_0, r_0)$. Thus $\delta'^*(\varepsilon) = (q_0, r_0) = (\delta_A^*(\varepsilon), \delta_B^*(\varepsilon))$.

For the inductive step, assume that for some natural number n, for any string *w* of length n, the theorem is true. We prove that the theorem is true for any string of length n + 1. Consider any string of length n + 1; it must have the form *wa* for some string *w* of length n and some character a. Then by definition

$$\begin{aligned}
\delta'^*(wa) &= \delta'(\delta'^*(w), a) \\
&= \delta'((\delta_A^*(w), \delta_B^*(w)), a) \\
&= (\delta_A(\delta_A^*(w), a), \delta_B(\delta_B^*(w), a)) \\
&= (\delta_A^*(wa), \delta_B^*(wa))
\end{aligned}$$

Thus the theorem holds for strings of length n + 1, completing the induction. ■

Theorem: $L(A \times B) = L(A) \cap L(B)$

Theorem: $L(A \times B) = L(A) \cap L(B)$

Proof: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction.

Theorem: $L(A \times B) = L(A) \cap L(B)$

Proof: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then we have that

$L(A \times B) = \{ w \mid \delta'^*(w) \in F_A \times F_B \}$

Theorem: $L(A \times B) = L(A) \cap L(B)$

Proof: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then we have that

$$L(A \times B) = \{ w \mid \delta'^*(w) \in F_A \times F_B \}$$
$$= \{ w \mid (\delta_A^*(w), \delta_B^*(w)) \in F_A \times F_B \}$$

Theorem: $L(A \times B) = L(A) \cap L(B)$

Proof: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then we have that

$$\begin{aligned} L(A \times B) &= \{ w \mid \delta'^*(w) \in F_A \times F_B \} \\ &= \{ w \mid (\delta_A^*(w), \delta_B^*(w)) \in F_A \times F_B \} \\ &= \{ w \mid \delta_A^*(w) \in F_A \wedge \delta_B^*(w) \in F_B \} \end{aligned}$$

Theorem: $L(A \times B) = L(A) \cap L(B)$

Proof: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then we have that

$$
\begin{aligned}
L(A \times B) &= \{ w \mid \delta'^*(w) \in F_A \times F_B \} \\
&= \{ w \mid (\delta_A^*(w), \delta_B^*(w)) \in F_A \times F_B \} \\
&= \{ w \mid \delta_A^*(w) \in F_A \wedge \delta_B^*(w) \in F_B \} \\
&= \{ w \mid \delta_A^*(w) \in F_A \} \cap \{ w \mid \delta_B^*(w) \in F_B \}
\end{aligned}
$$

Theorem: $L(A \times B) = L(A) \cap L(B)$

Proof: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then we have that

$$
\begin{aligned}
L(A \times B) &= \{ w \mid \delta'^*(w) \in F_A \times F_B \} \\
&= \{ w \mid (\delta_A^*(w), \delta_B^*(w)) \in F_A \times F_B \} \\
&= \{ w \mid \delta_A^*(w) \in F_A \wedge \delta_B^*(w) \in F_B \} \\
&= \{ w \mid \delta_A^*(w) \in F_A \} \cap \{ w \mid \delta_B^*(w) \in F_B \} \\
&= L(A) \cap L(B)
\end{aligned}
$$

Theorem: $L(A \times B) = L(A) \cap L(B)$

Proof: Given DFAs $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (R, \Sigma, \delta_B, r_0, F_B)$, let $A \times B = (Q \times R, \Sigma, \delta', (q_0, r_0), F_A \times F_B)$ be the DFA produced by the product construction. Then we have that

$$
\begin{aligned}
L(A \times B) &= \{ w \mid \delta'^*(w) \in F_A \times F_B \} \\
&= \{ w \mid (\delta_A^*(w), \delta_B^*(w)) \in F_A \times F_B \} \\
&= \{ w \mid \delta_A^*(w) \in F_A \wedge \delta_B^*(w) \in F_B \} \\
&= \{ w \mid \delta_A^*(w) \in F_A \} \cap \{ w \mid \delta_B^*(w) \in F_B \} \\
&= L(A) \cap L(B) \quad \blacksquare
\end{aligned}
$$

Why These Proofs Matter

- There are too many languages for all of them to be recognized by DFAs (modified proof from first lecture).
- So what can DFAs recognize?
- These sorts of proofs let us learn more about what languages (problems) DFAs are capable of solving based on what problems we already know they can solve.

Using these Results

- Right now, we've shown that if L_1 and L_2 are regular languages, that $\overline{L}_1$ and $L_1 \cap L_2$ are regular languages.
- How about $L_1 - L_2$?

Using these Results

- Right now, we've shown that if L_1 and L_2 are regular languages, that $\overline{L}_1$ and $L_1 \cap L_2$ are regular languages.
- How about $L_1 - L_2$?

$$L_1 - L_2 = L_1 \cap \overline{L}_2$$

Getting More Mileage

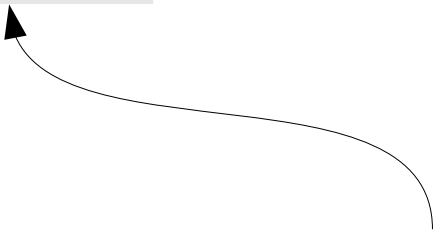
- If L is a regular language, is this language regular?

$$\{ w \mid w \in L \wedge |w| \geq n \}, \text{ where } n \in \mathbb{N}$$

Getting More Mileage

- If L is a regular language, is this language regular?

$$\{ w \mid w \in L \wedge |w| \geq n \}, \text{ where } n \in \mathbb{N}$$



The length of a string is denoted $|w|$

Getting More Mileage

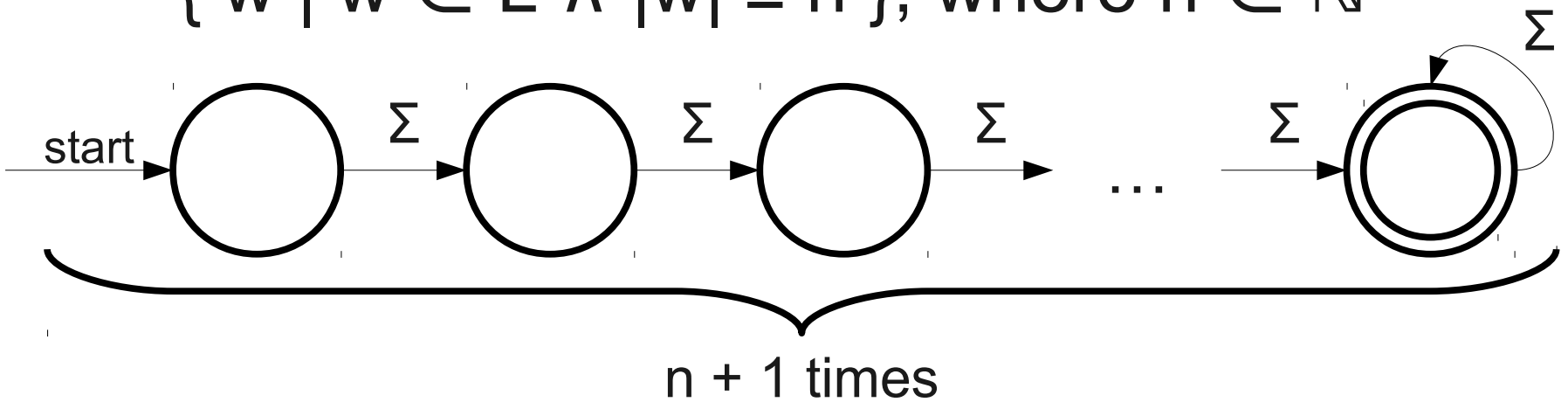
- If L is a regular language, is this language regular?

$$\{ w \mid w \in L \wedge |w| \geq n \}, \text{ where } n \in \mathbb{N}$$

Getting More Mileage

- If L is a regular language, is this language regular?

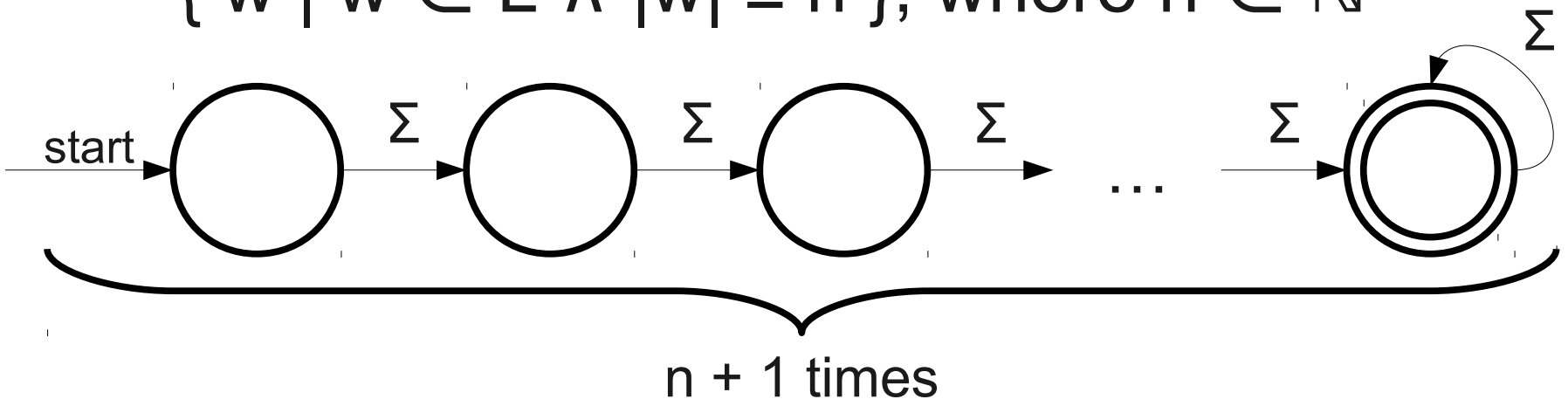
$$\{ w \mid w \in L \wedge |w| \geq n \}, \text{ where } n \in \mathbb{N}$$



Getting More Mileage

- If L is a regular language, is this language regular?

$$\{ w \mid w \in L \wedge |w| \geq n \}, \text{ where } n \in \mathbb{N}$$



- Intersect the DFA for L with this DFA.

Even More Mileage

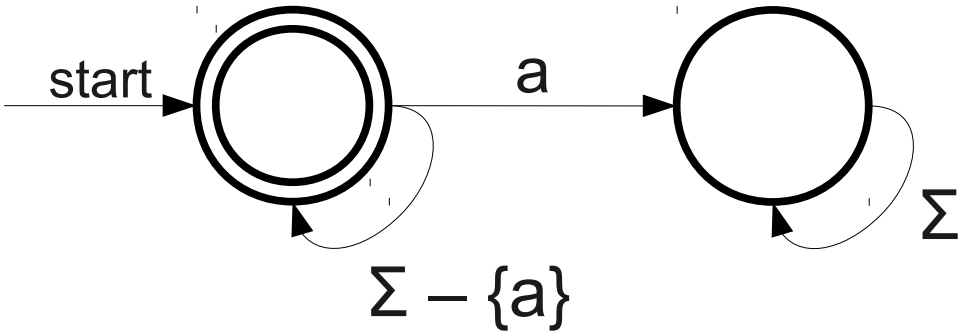
- If L is a regular language, is this language regular?

$$\{ w \mid w \in L \wedge w \text{ does not contain a} \},$$
$$\text{for some choice of } a \in \Sigma$$

Even More Mileage

- If L is a regular language, is this language regular?

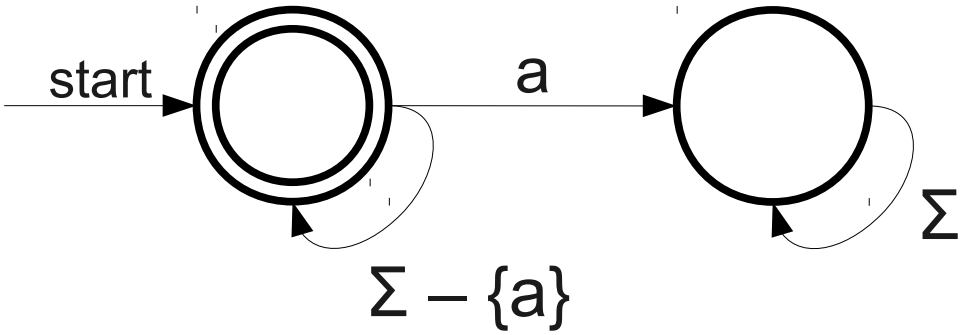
$$\{ w \mid w \in L \wedge w \text{ does not contain a} \},$$
$$\text{for some choice of } a \in \Sigma$$



Even More Mileage

- If L is a regular language, is this language regular?

$$\{ w \mid w \in L \wedge w \text{ does not contain a} \},$$
$$\text{for some choice of } a \in \Sigma$$



Intersect the DFA for L with this DFA.

Summary of DFAs

- Simple but powerful model of computation.
- Gives rise to **regular languages**.
- Constructions let us show what we else we can compute based on what we already know how to do.

NFAs

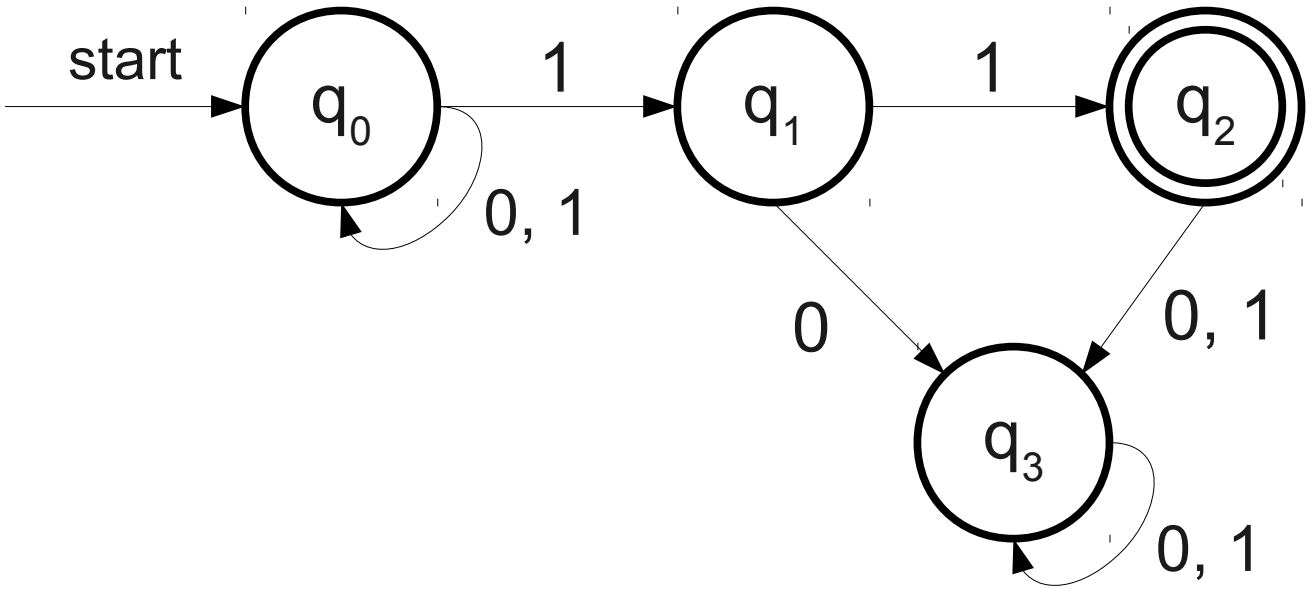
NFAs

- An **NFA** is a
  - **N**ondeterministic
  - **F**inite
  - **A**utomaton
- Conceptually similar to a DFA, but equipped with the power of **nondeterminism**.

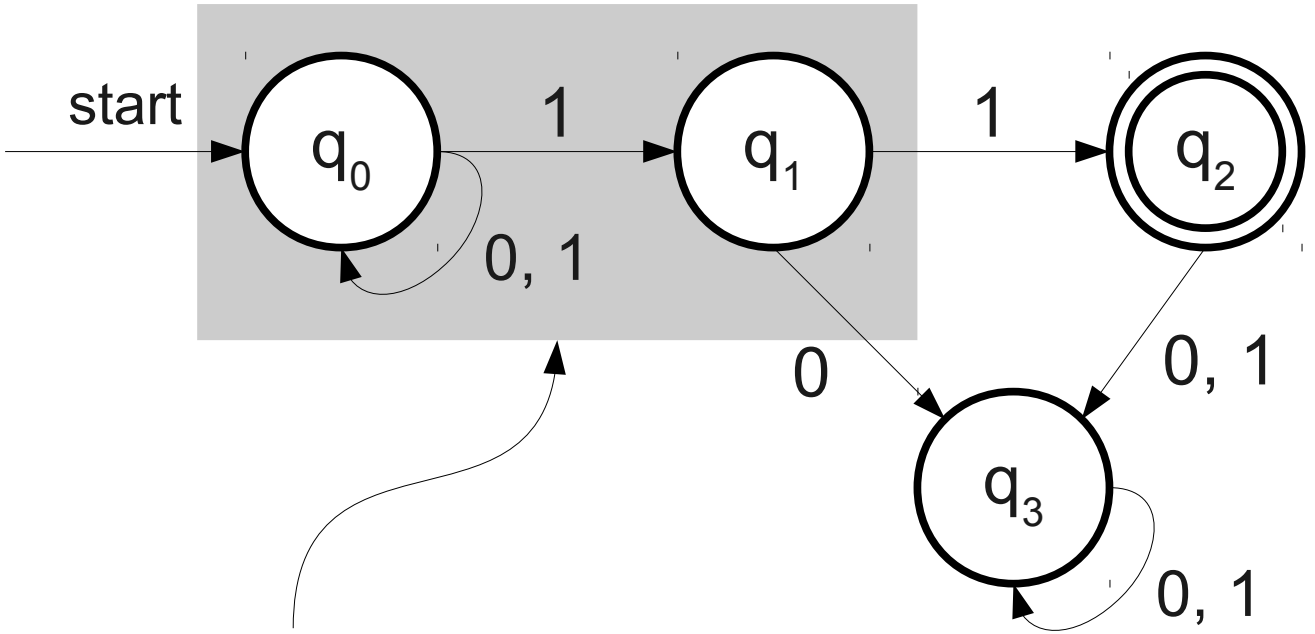
(Non)determinism

- A model of computation is **deterministic** if at every point in the computation, there is exactly one choice that can make.
- The machine accepts if that series of choices leads to an accepting state.
- A model of computation is **nondeterministic** if the computing machine may have multiple decisions that it can make at one point.
- The machine accepts if **any** series of choices leads to an accepting state.

A Simple NFA

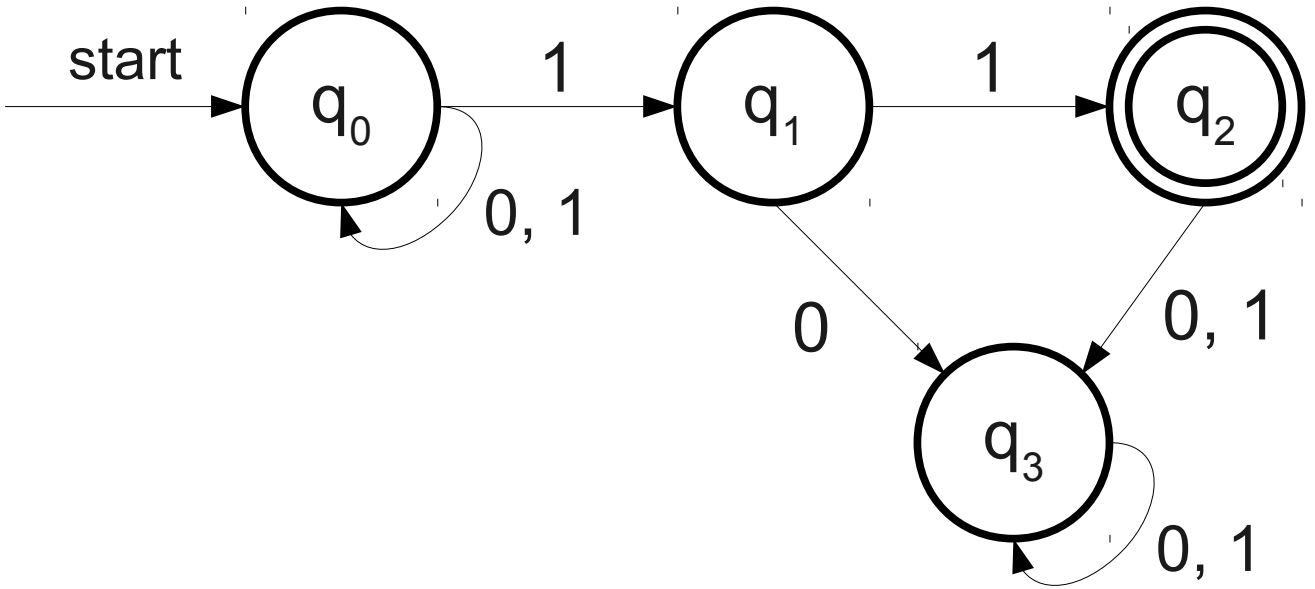


A Simple NFA



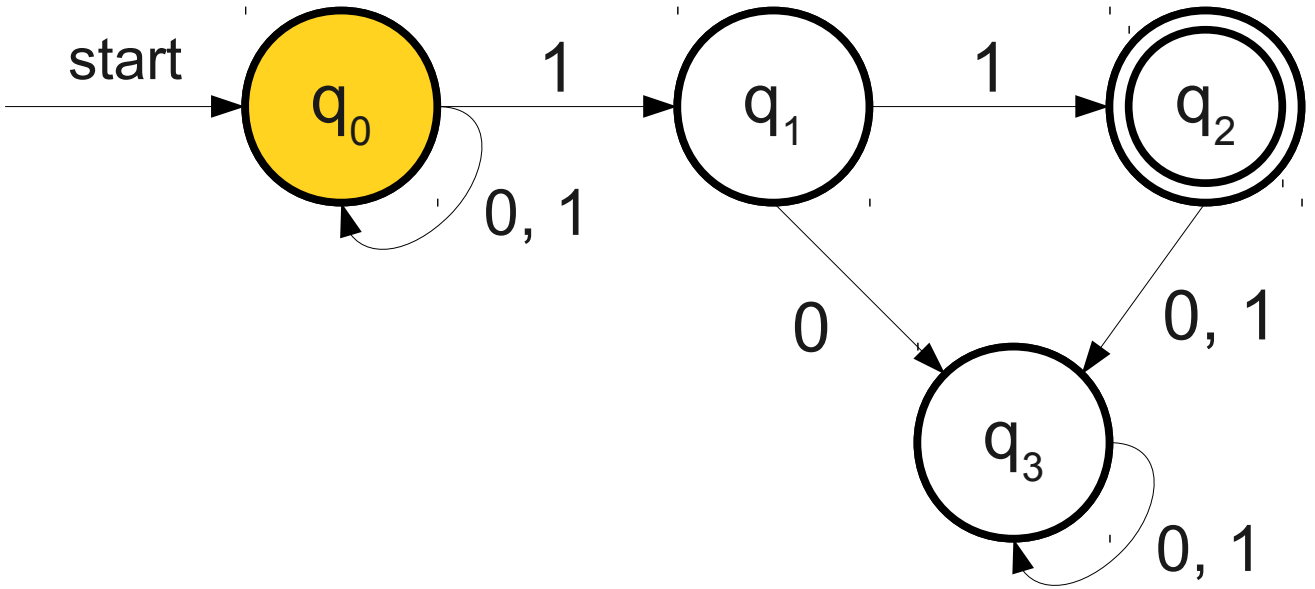
q_0 has two transitions defined on 1!

A Simple NFA



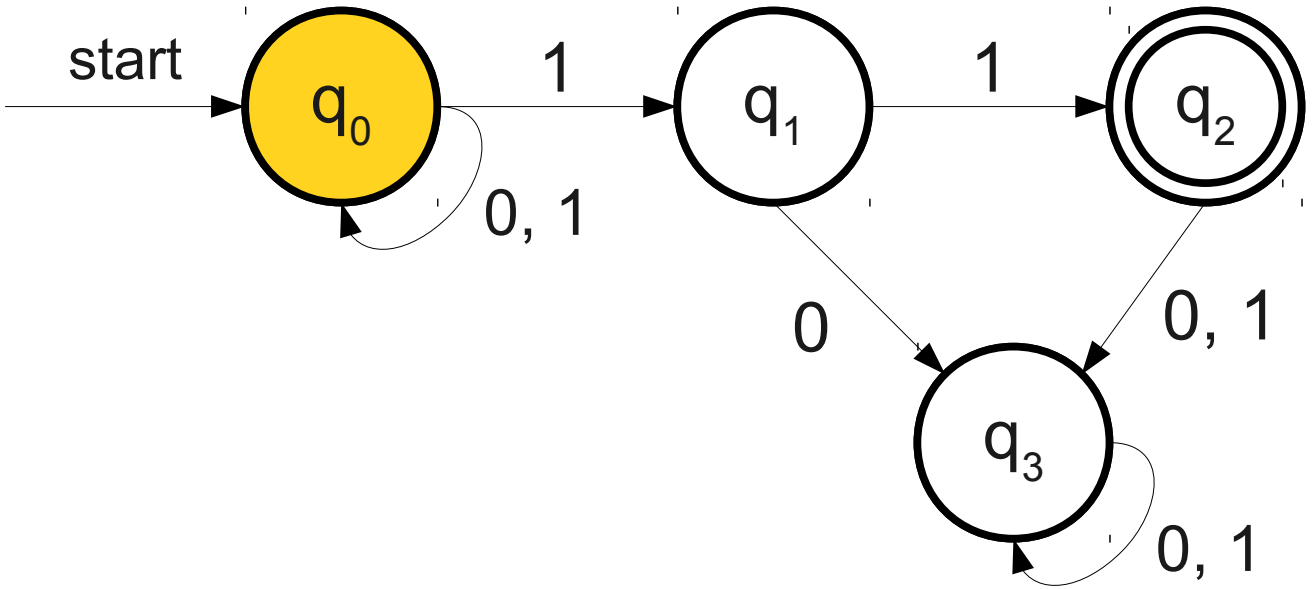
0 1 0 1 1

A Simple NFA

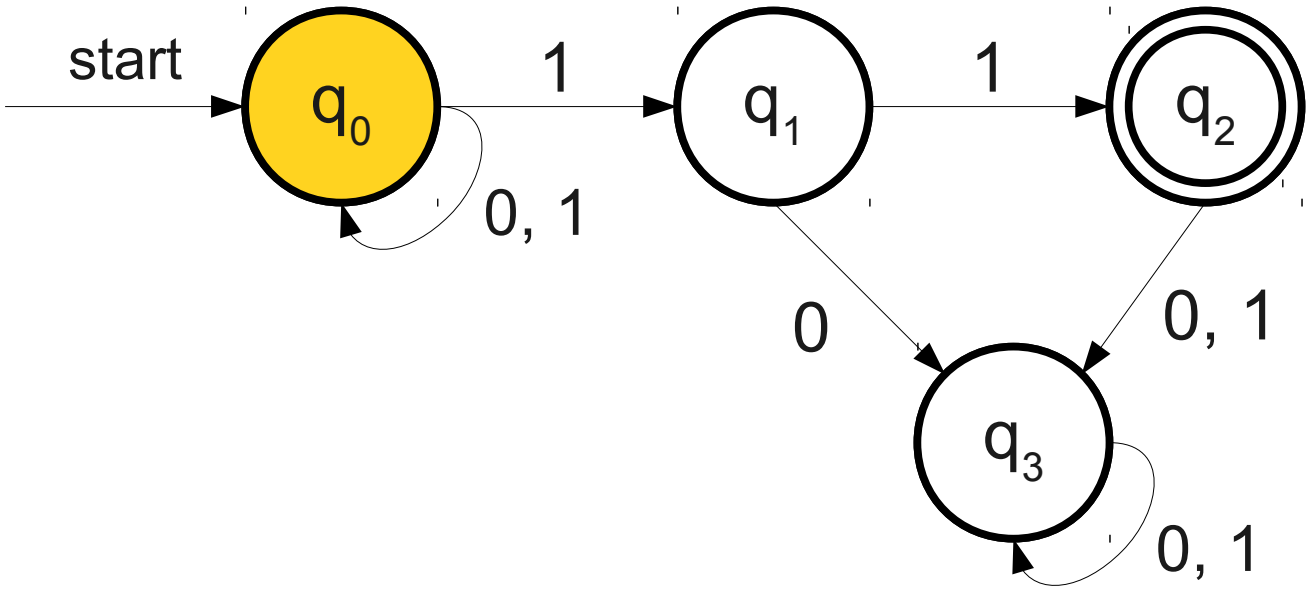


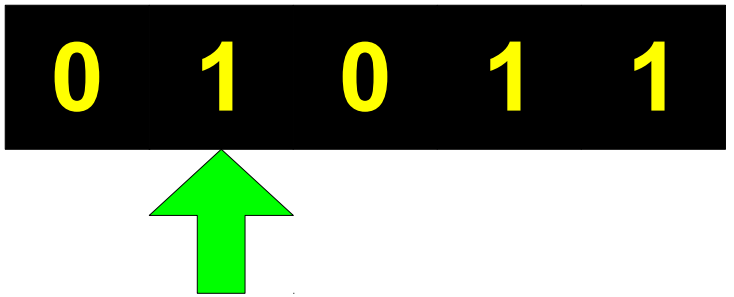
0 1 0 1 1

A Simple NFA

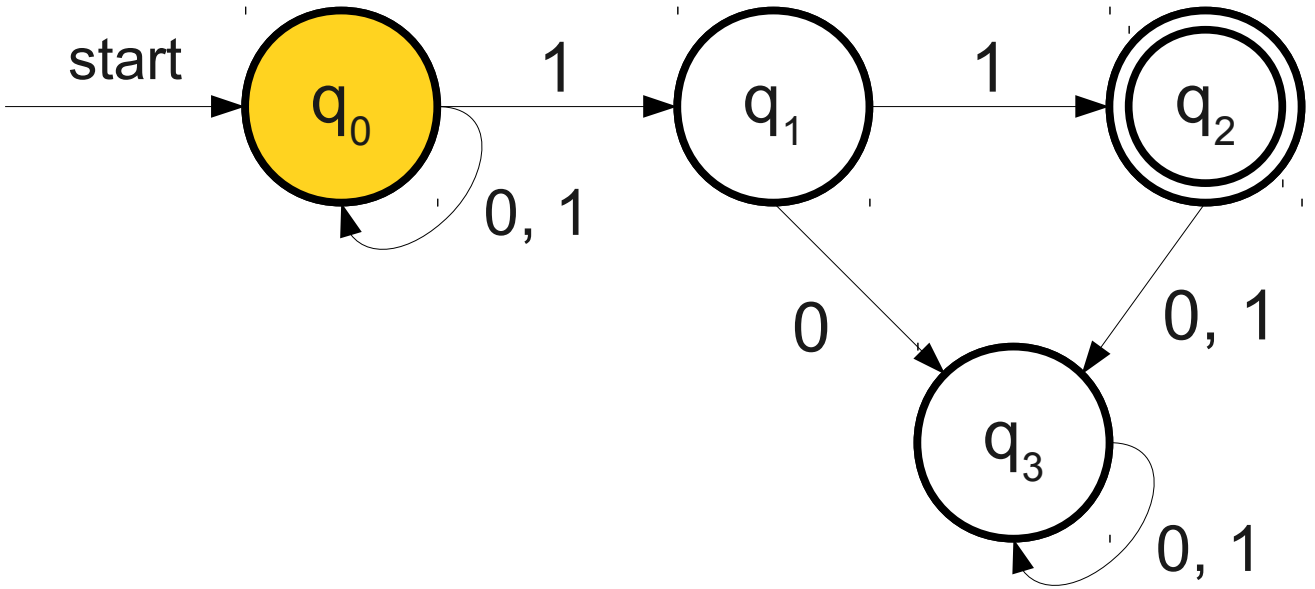


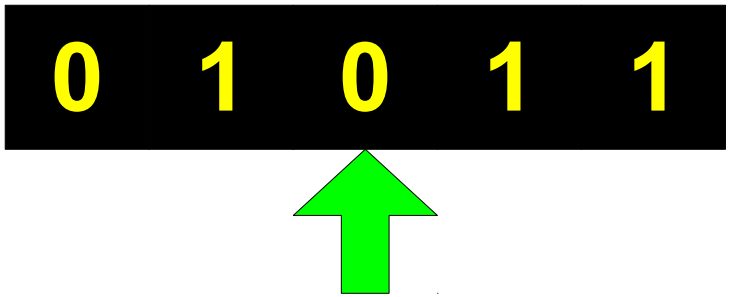
A Simple NFA



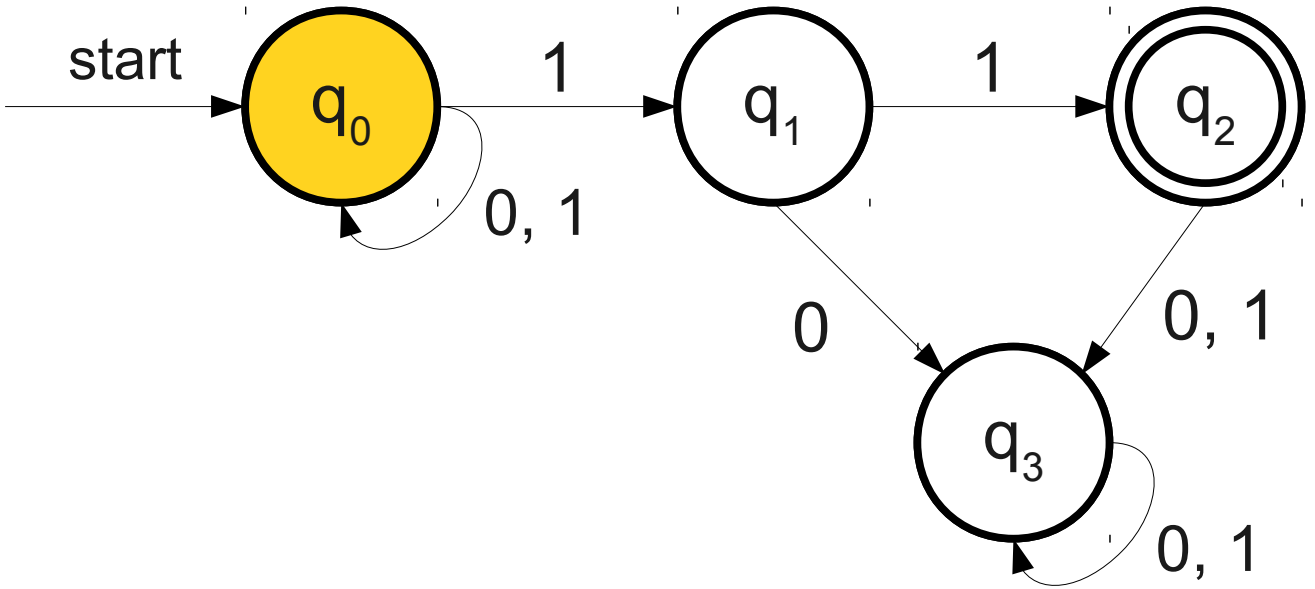


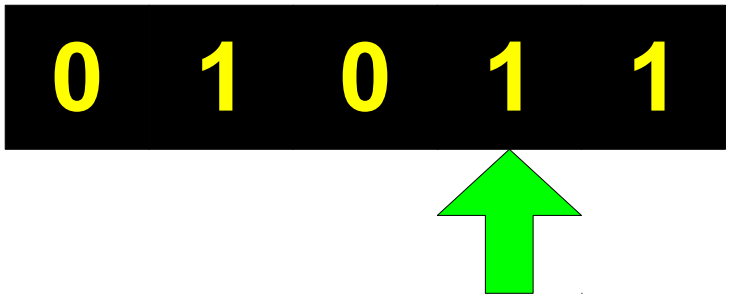
A Simple NFA



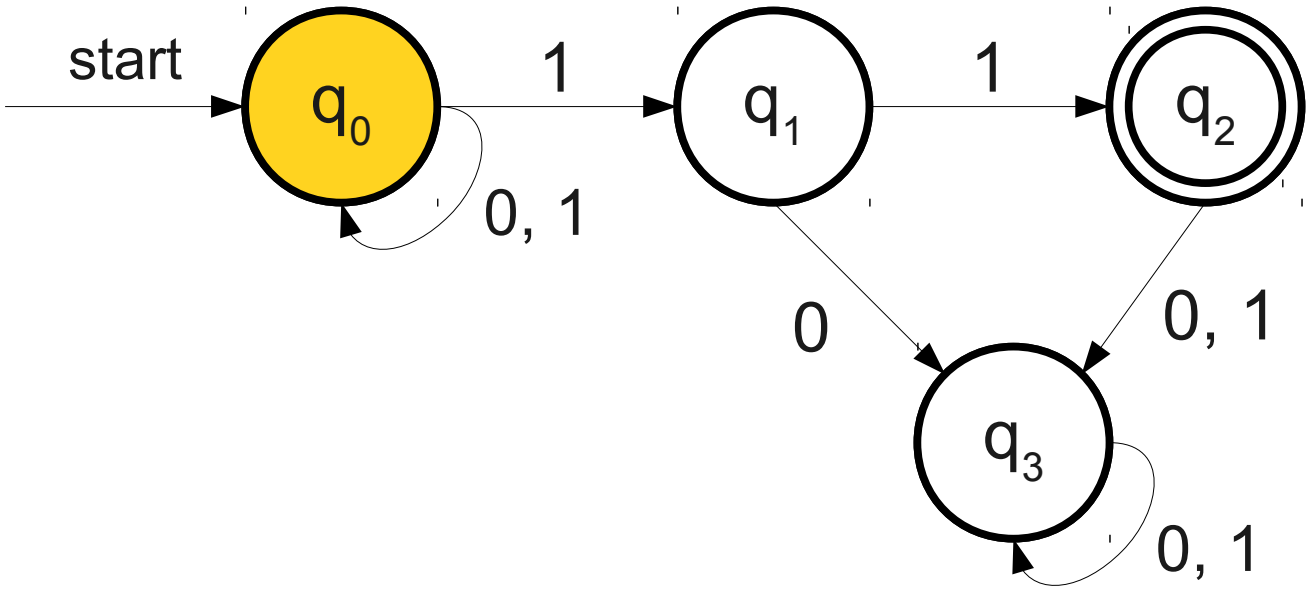


A Simple NFA

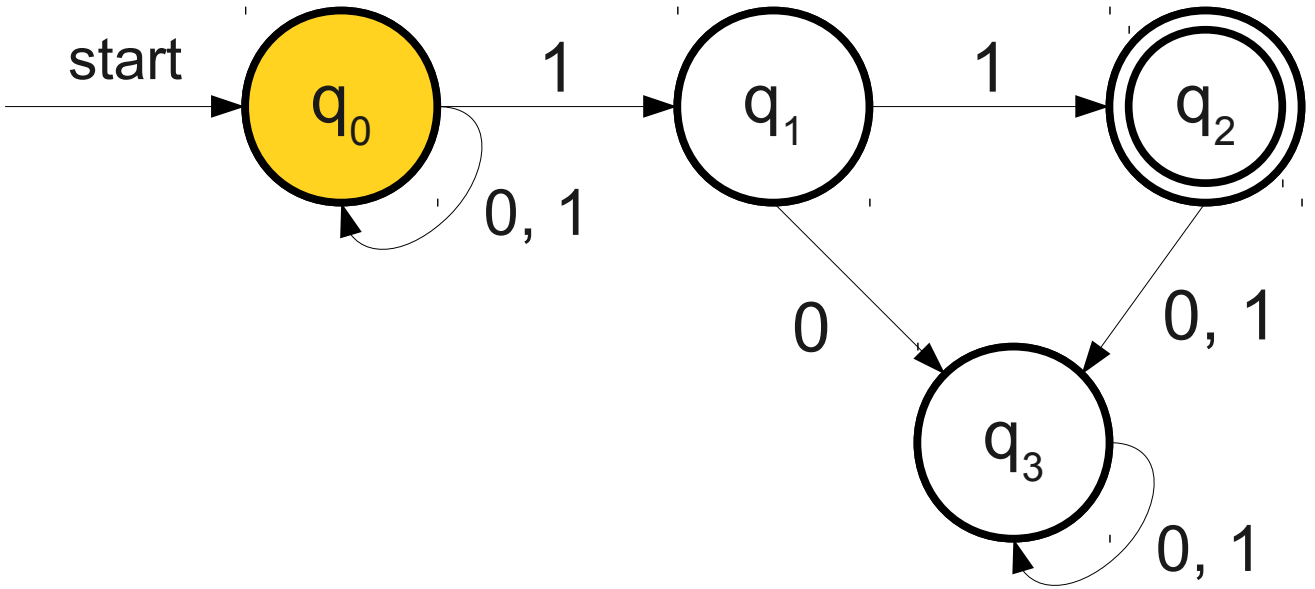




A Simple NFA

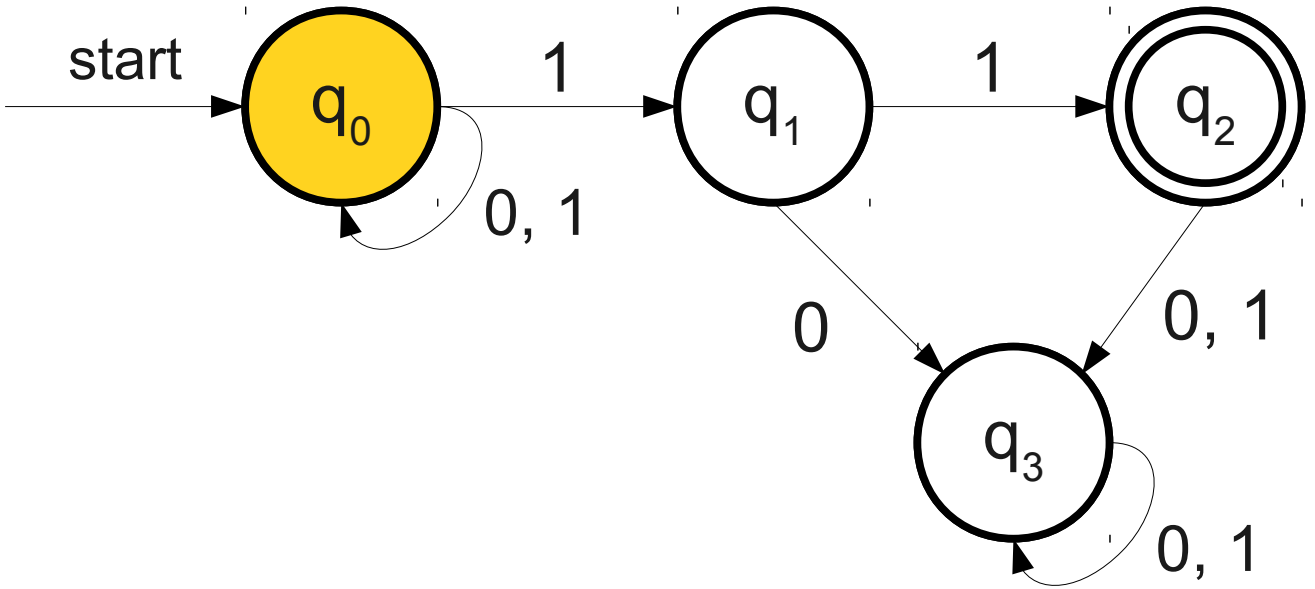


A Simple NFA

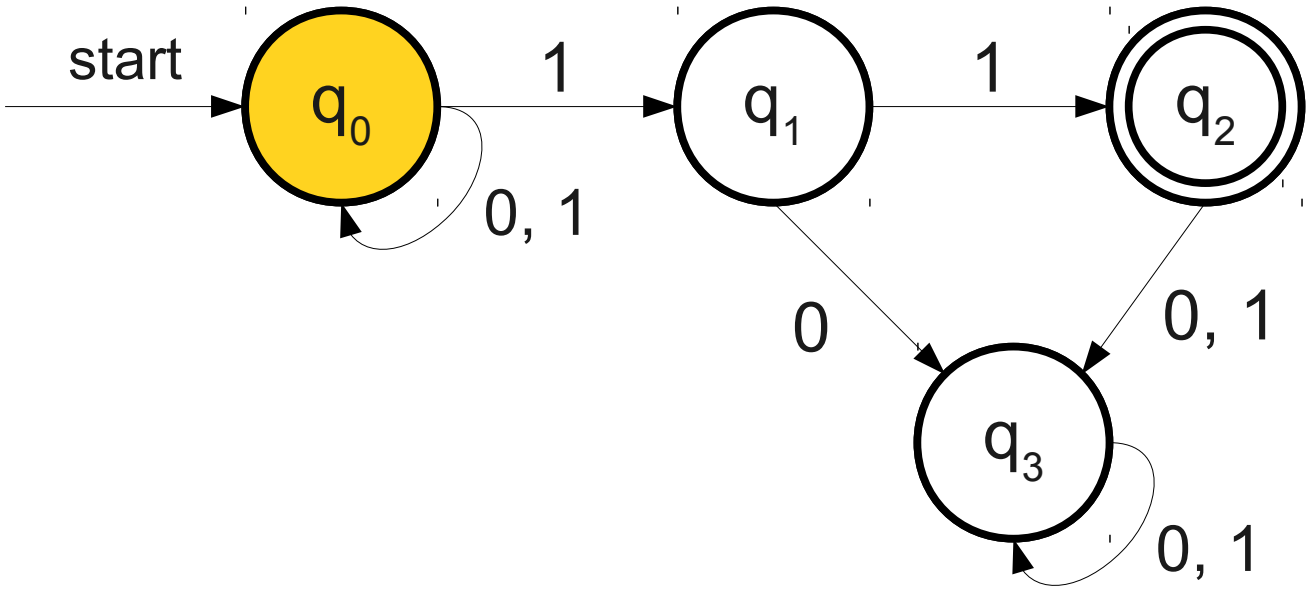


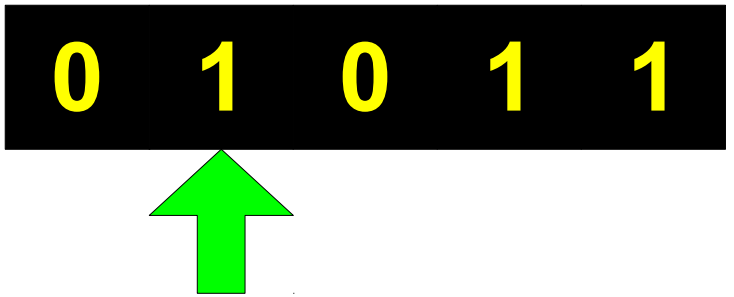
0 1 0 1 1

A Simple NFA

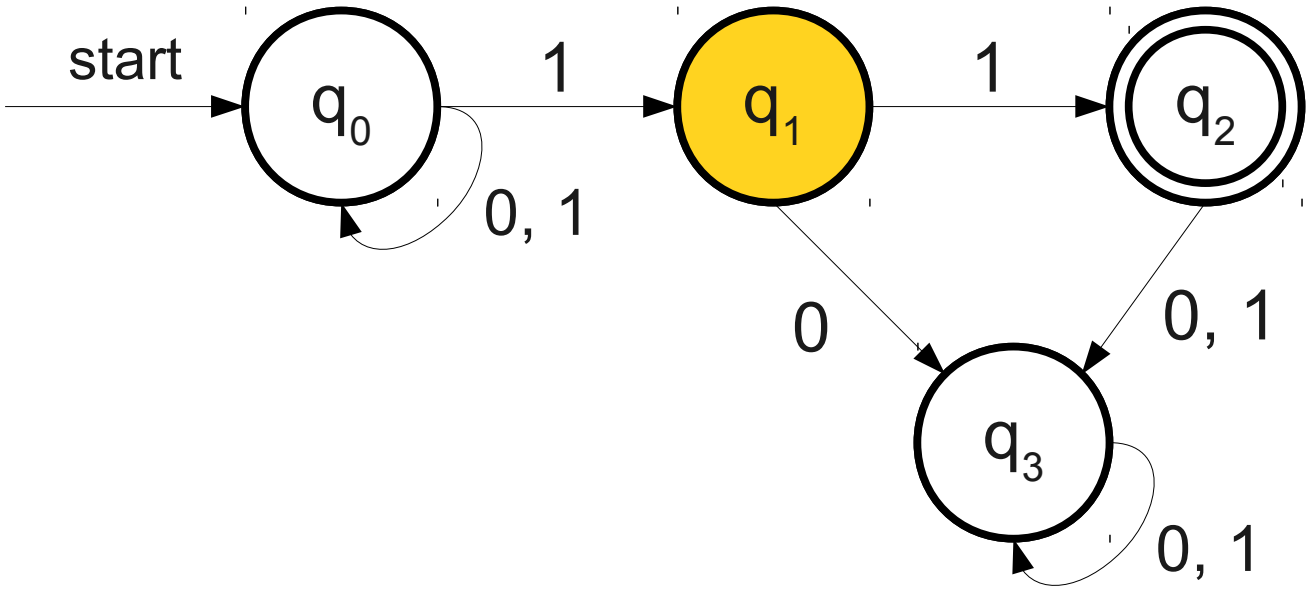


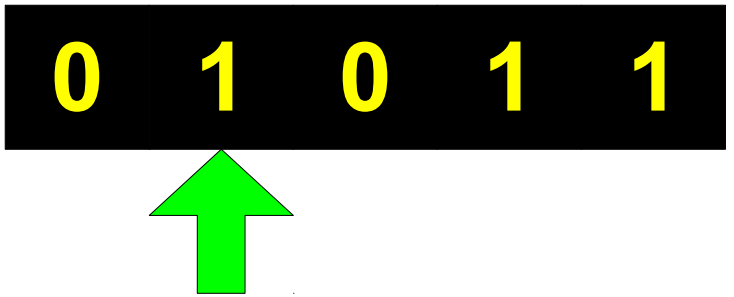
A Simple NFA



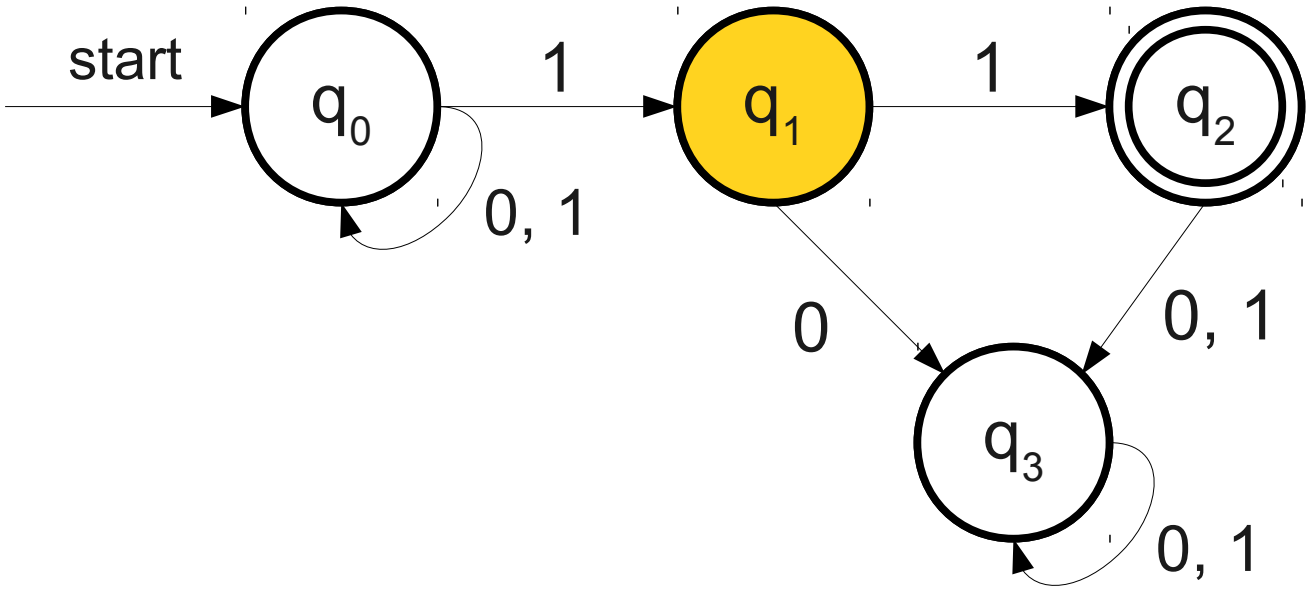


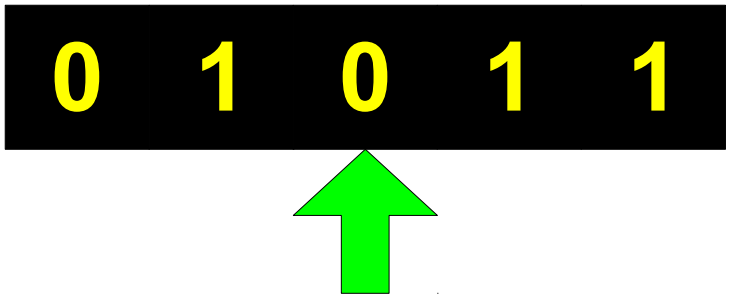
A Simple NFA



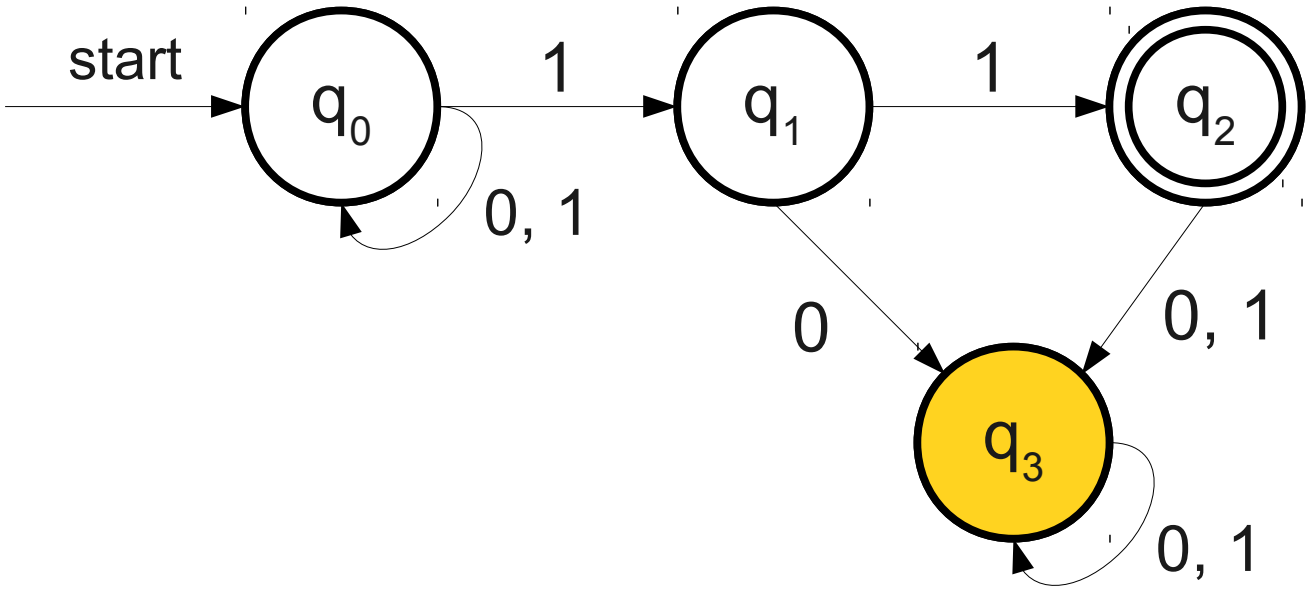


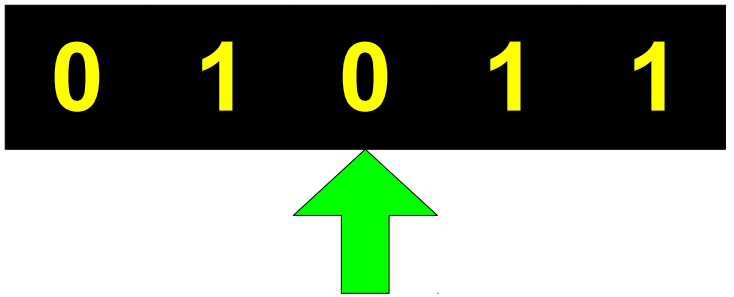
A Simple NFA



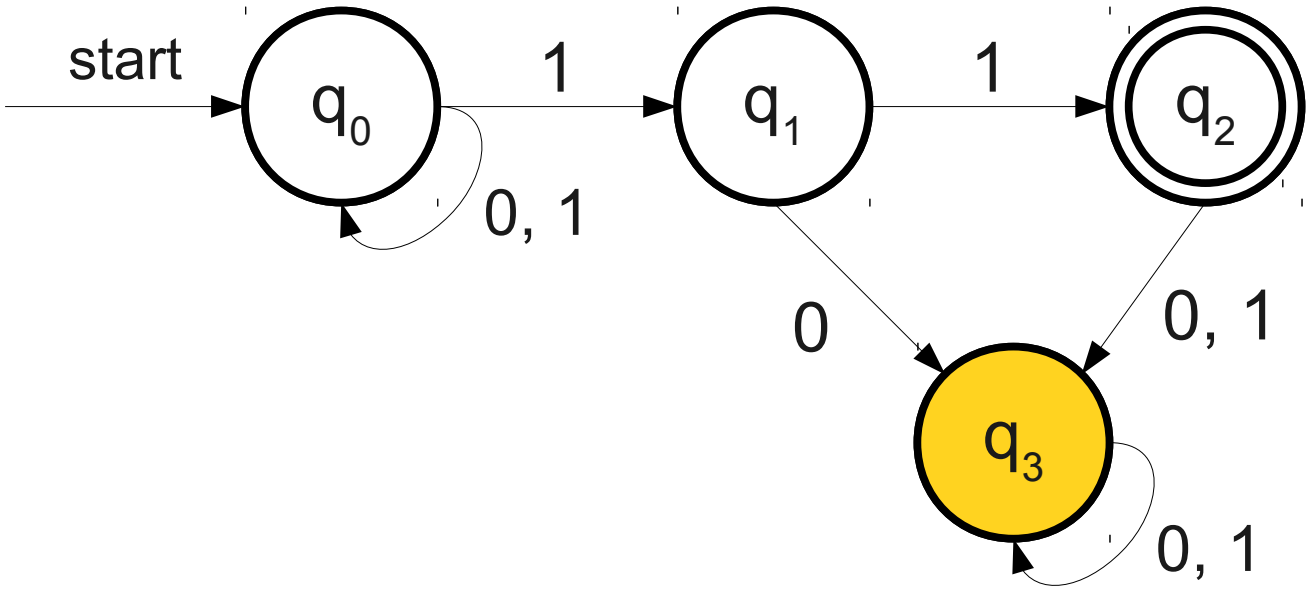


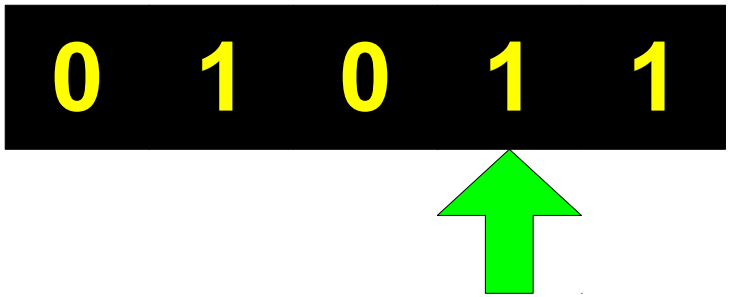
A Simple NFA



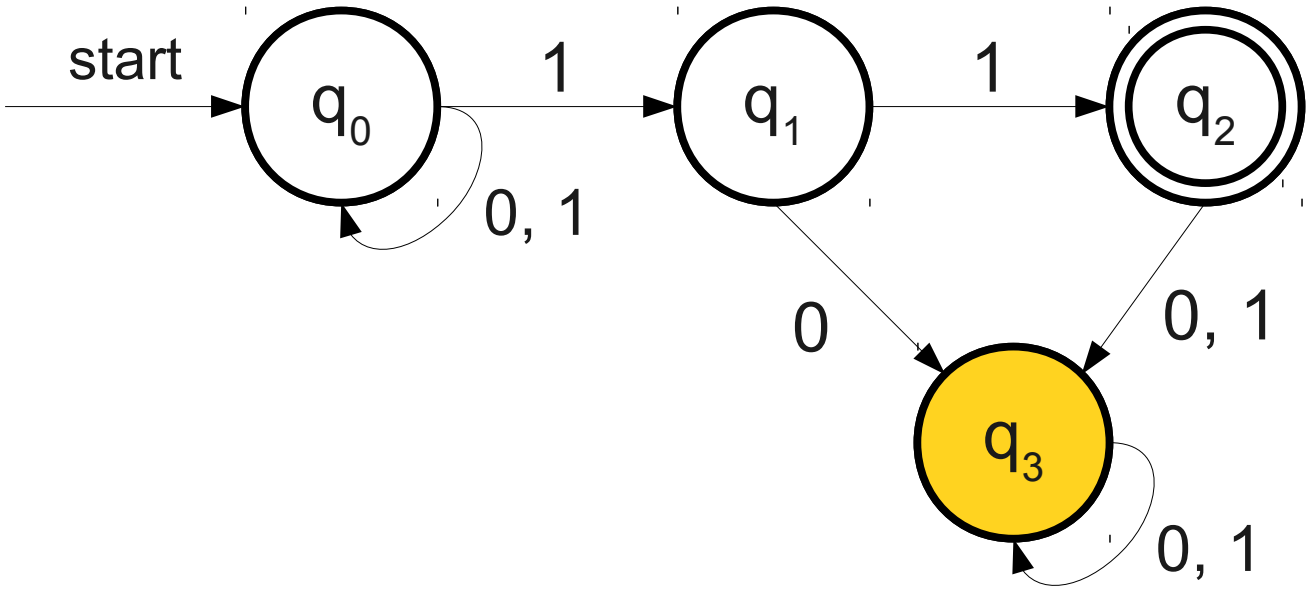


A Simple NFA

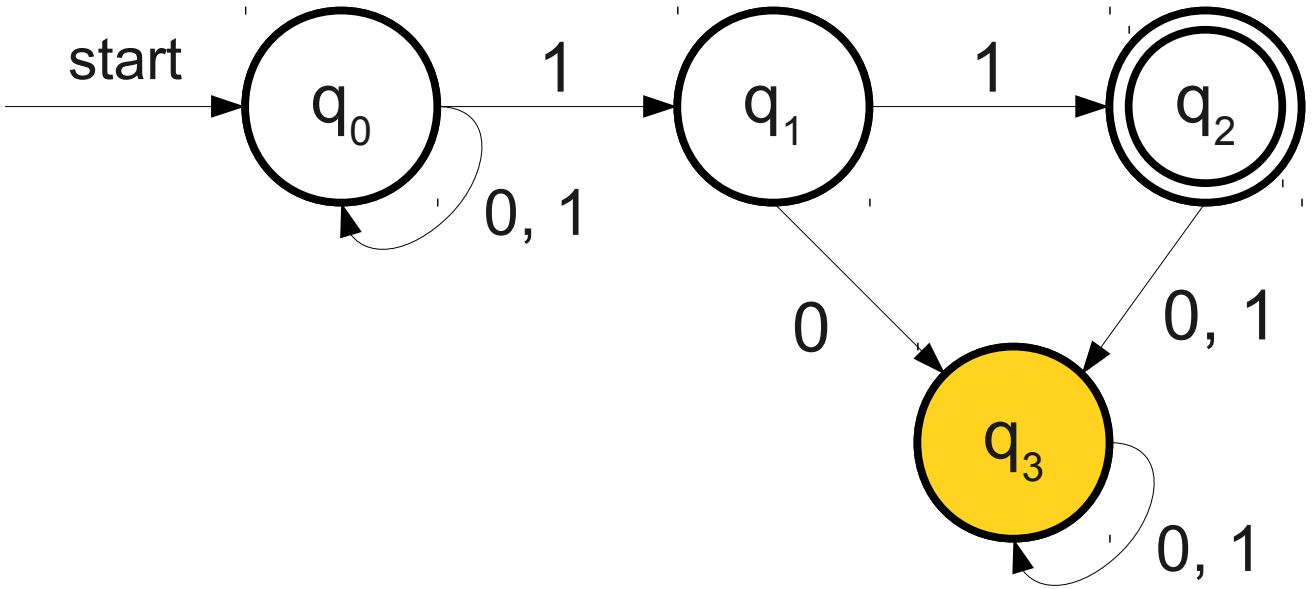




A Simple NFA

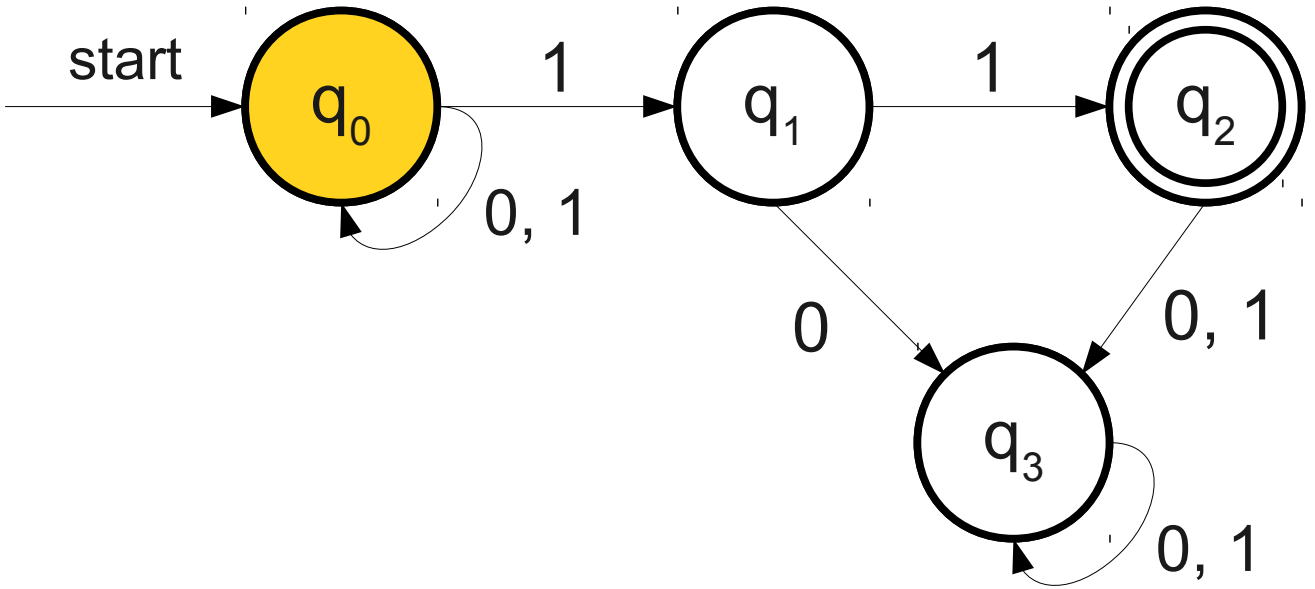


A Simple NFA

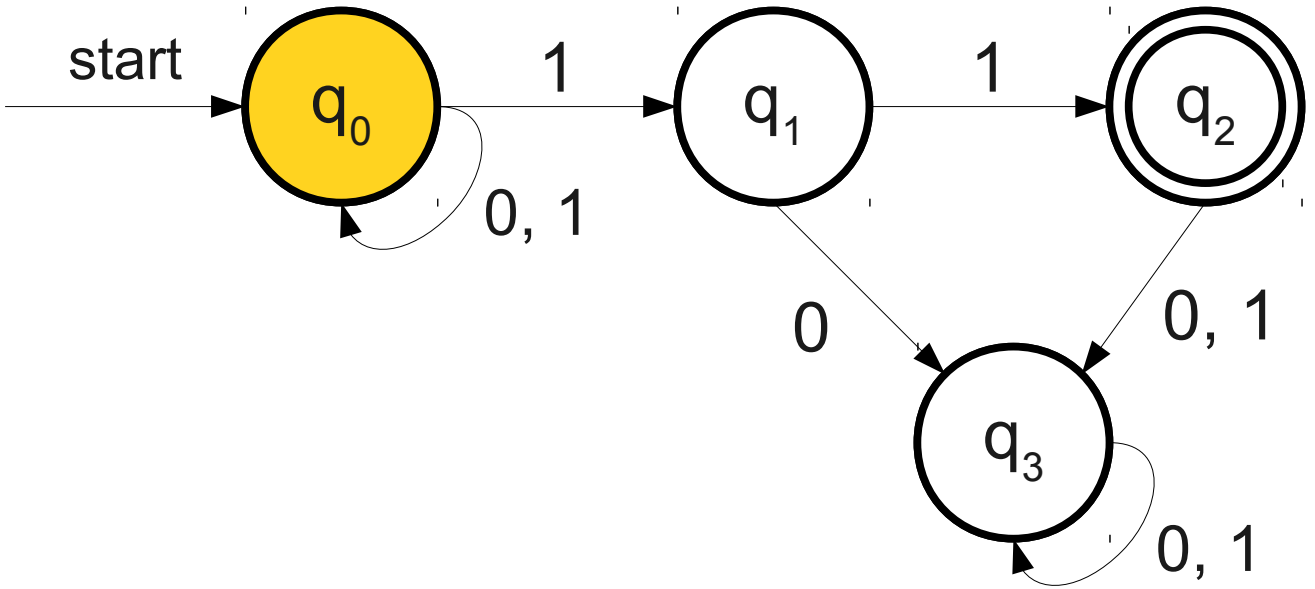


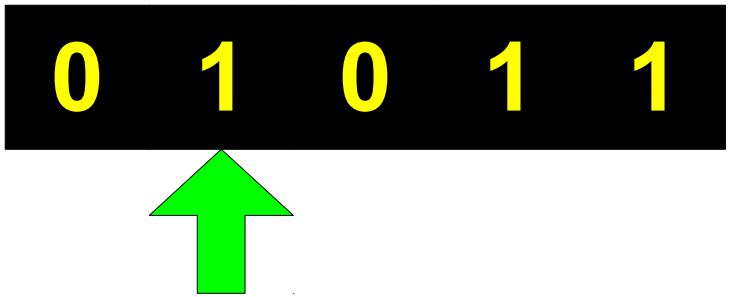
0 1 0 1 1

A Simple NFA

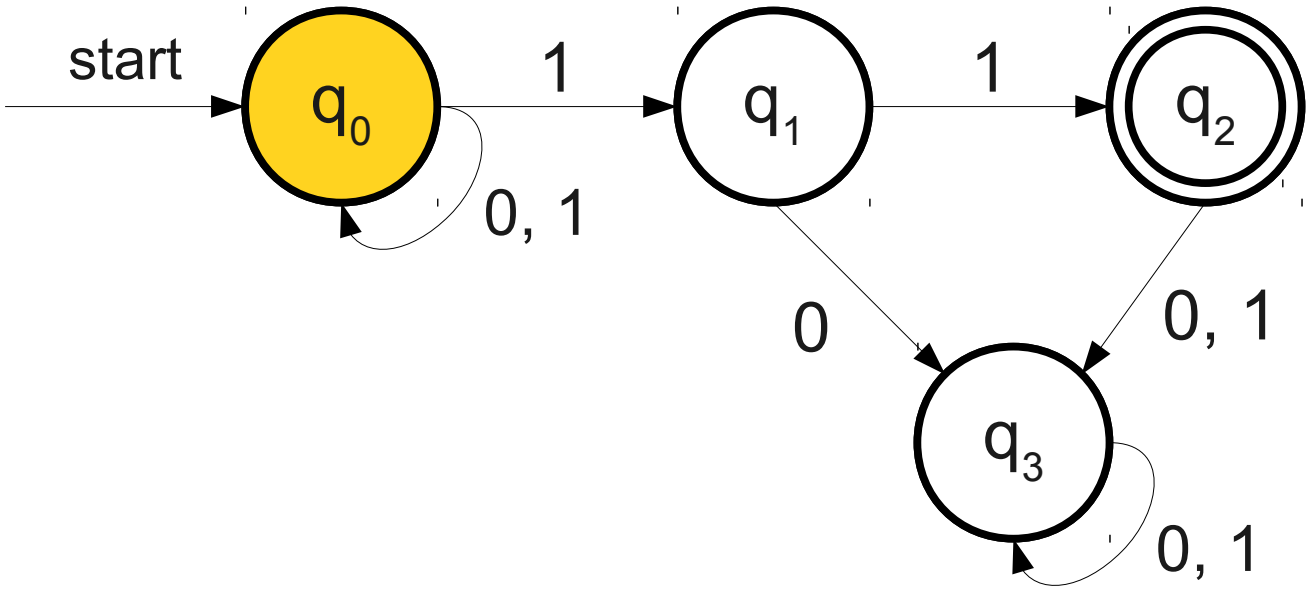


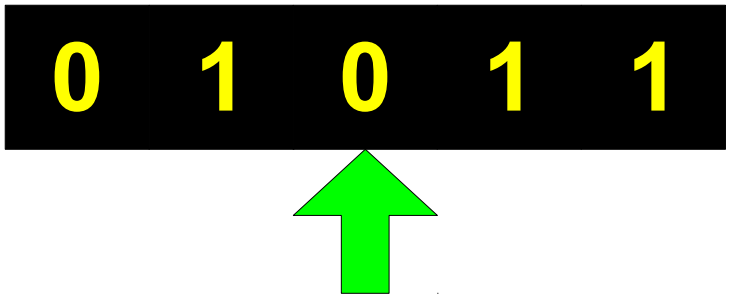
A Simple NFA



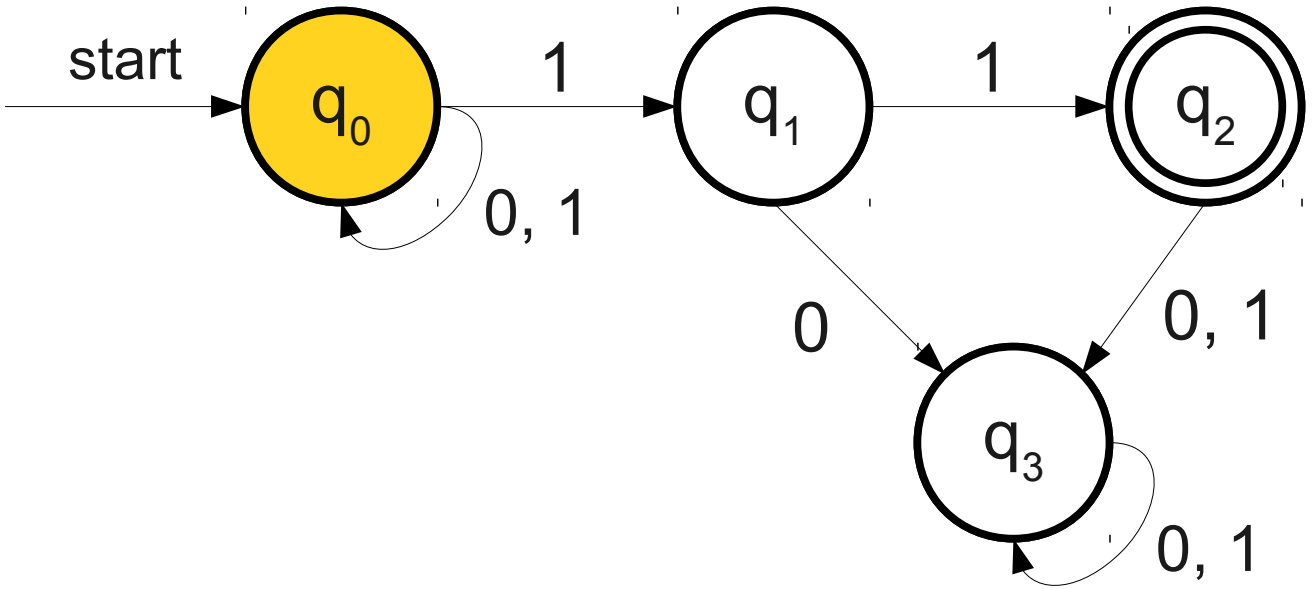


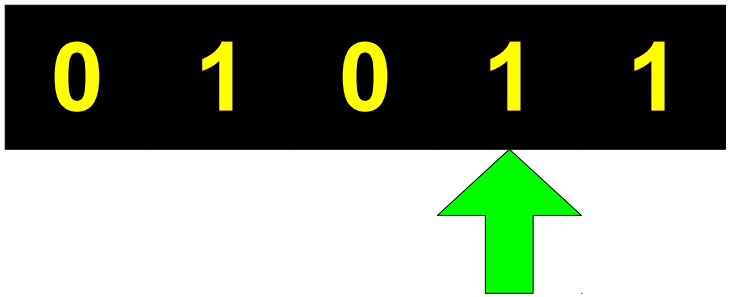
A Simple NFA



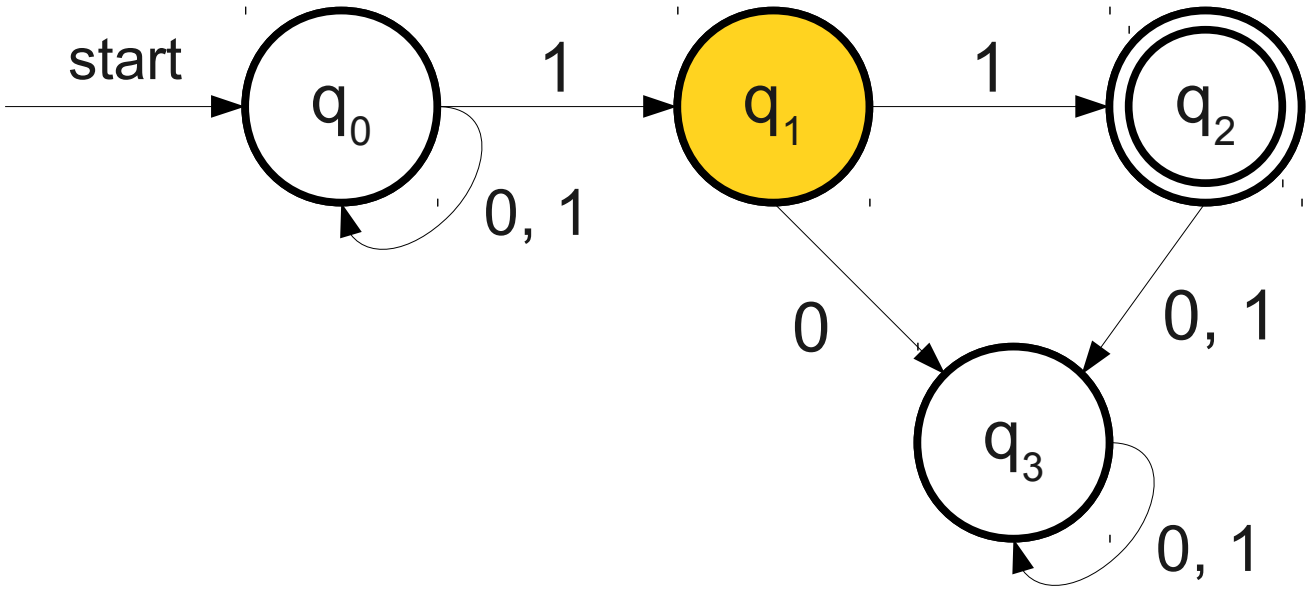


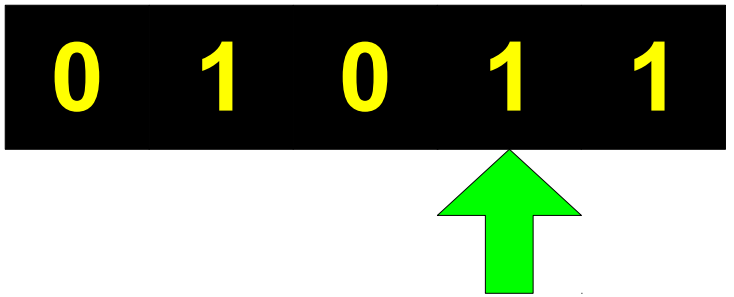
A Simple NFA



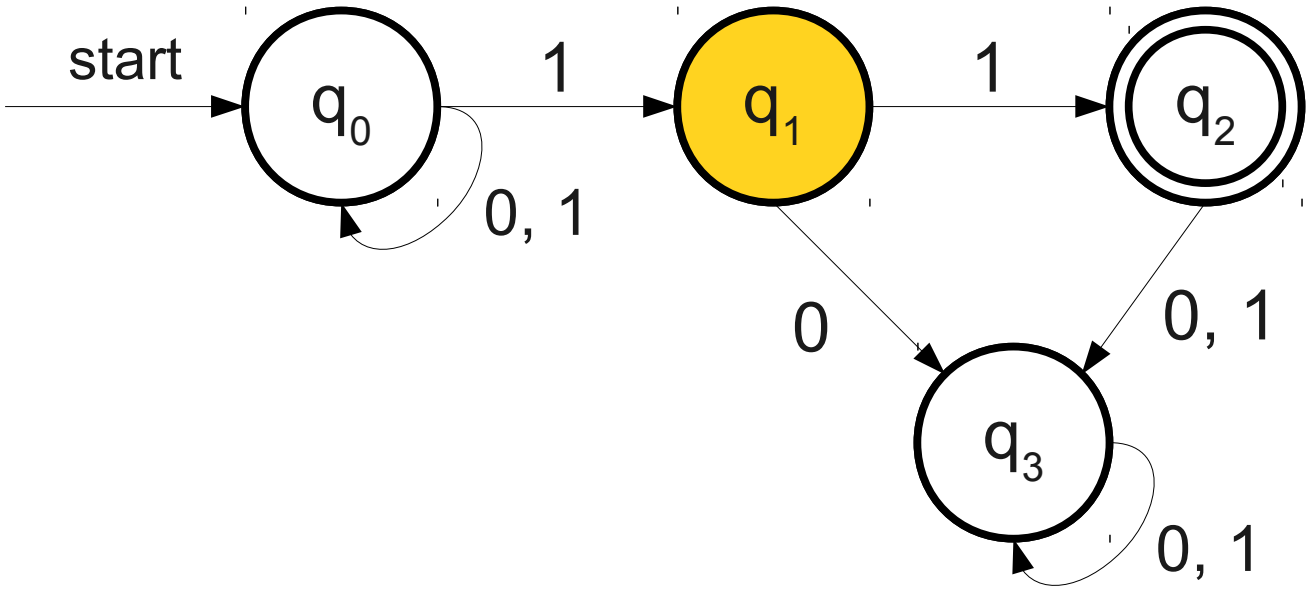


A Simple NFA

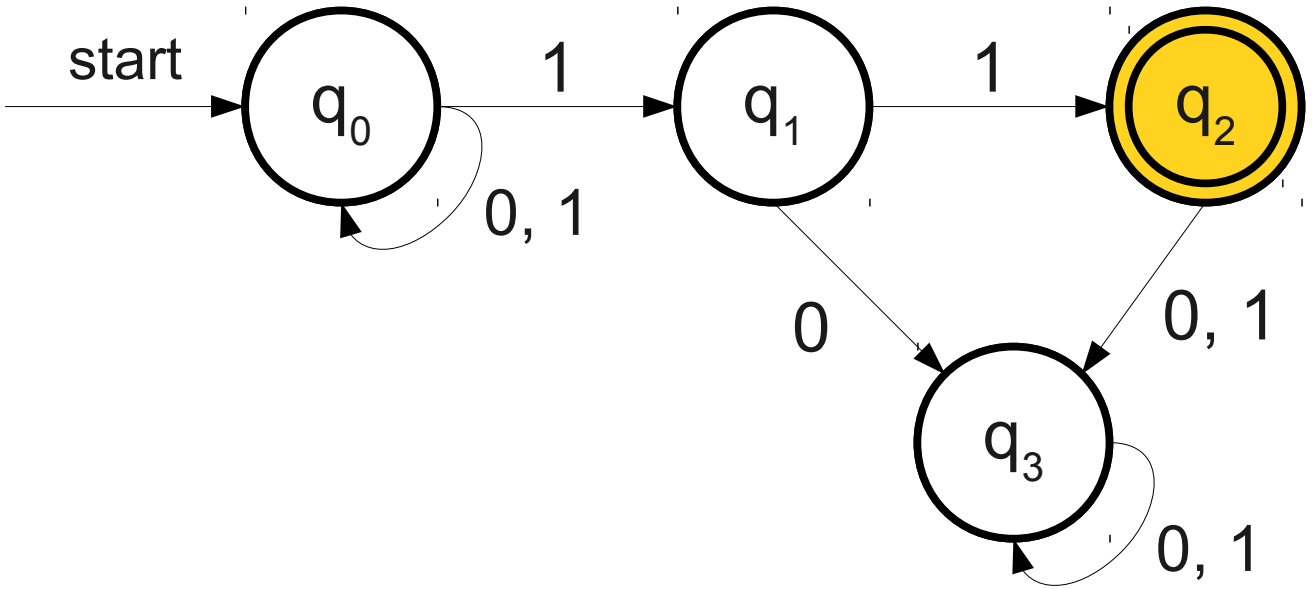




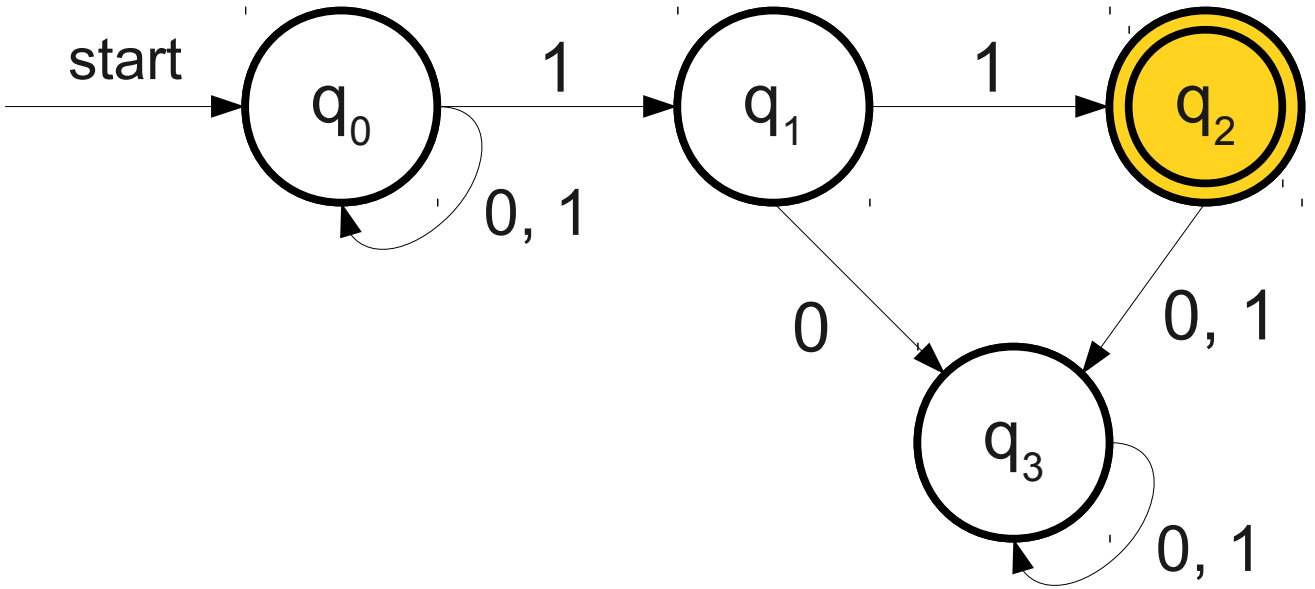
A Simple NFA



A Simple NFA

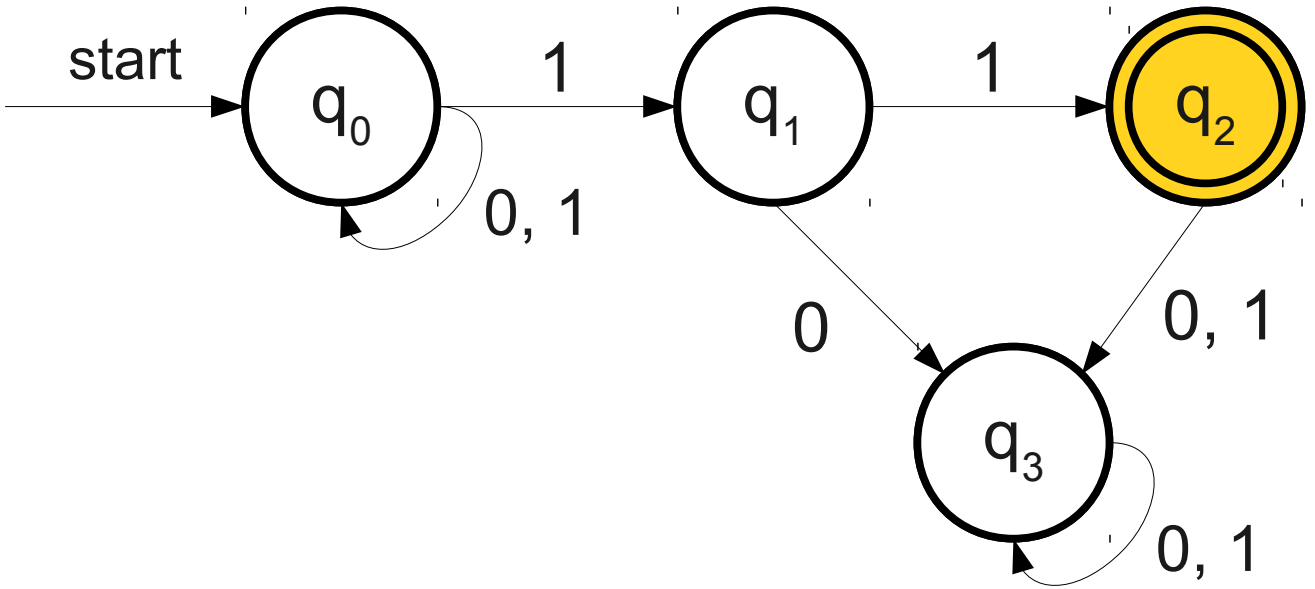


A Simple NFA



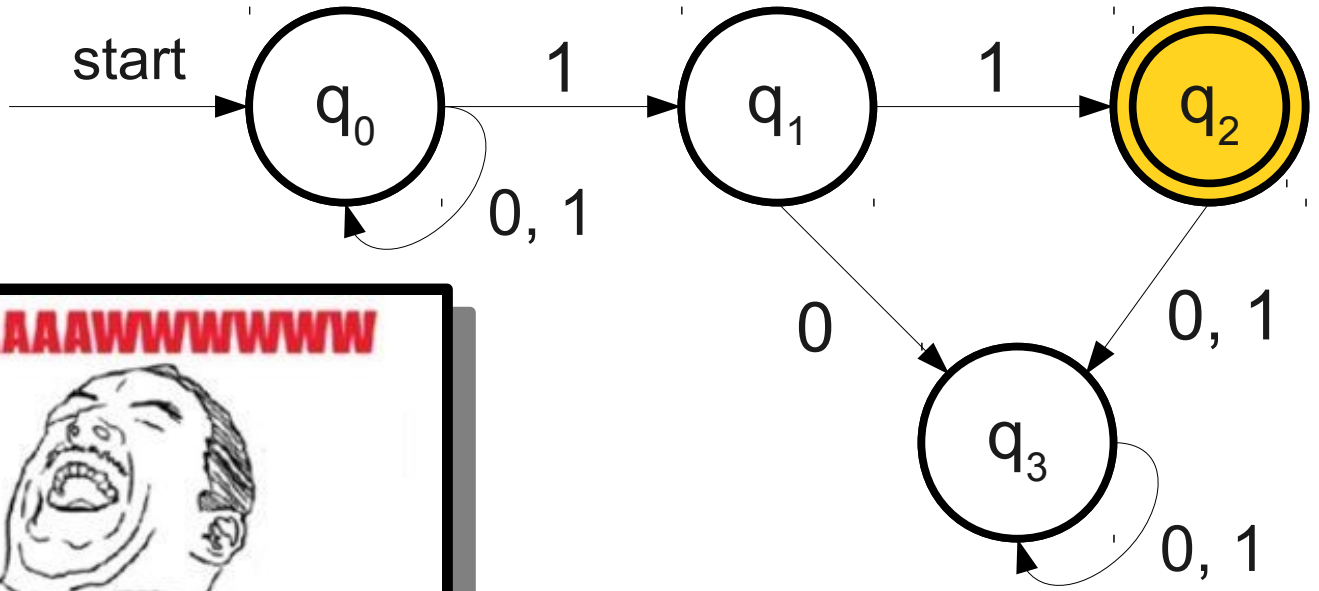
0 1 0 1 1

A Simple NFA



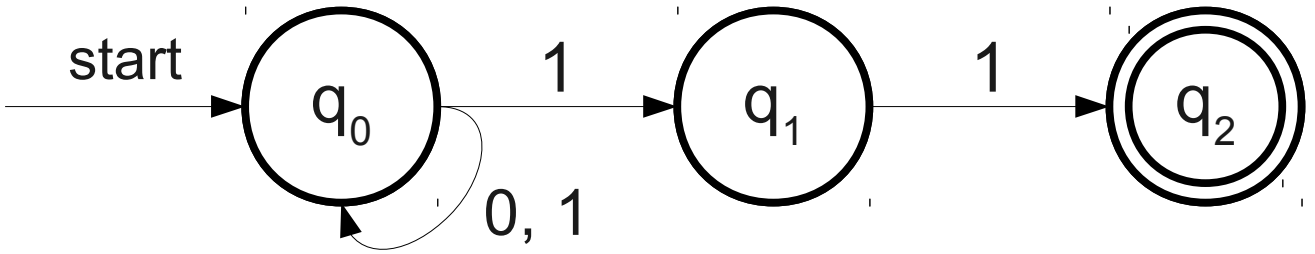
0 1 0 1 1

A Simple NFA

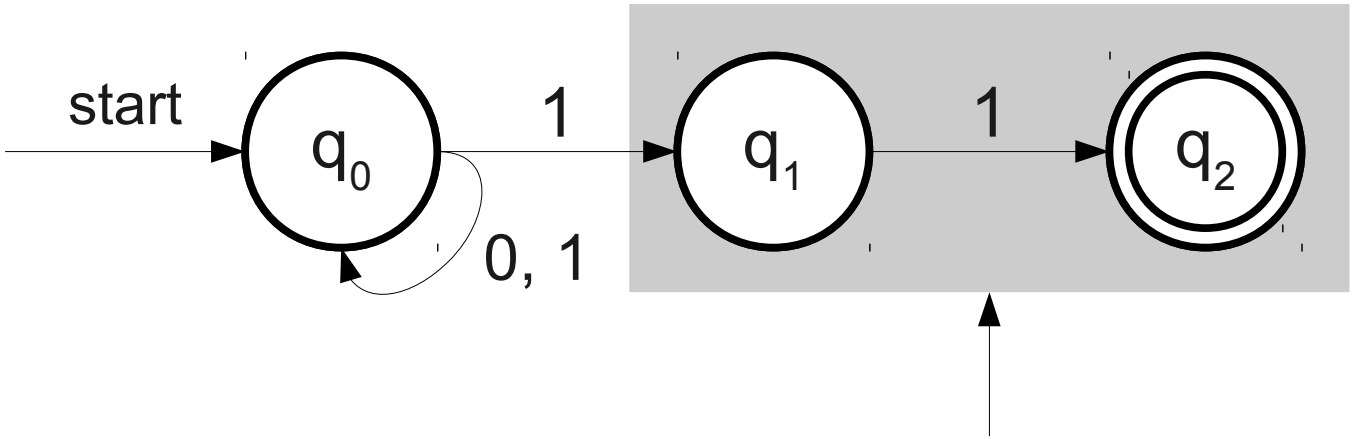


0 1 0 1 1

A More Complex NFA

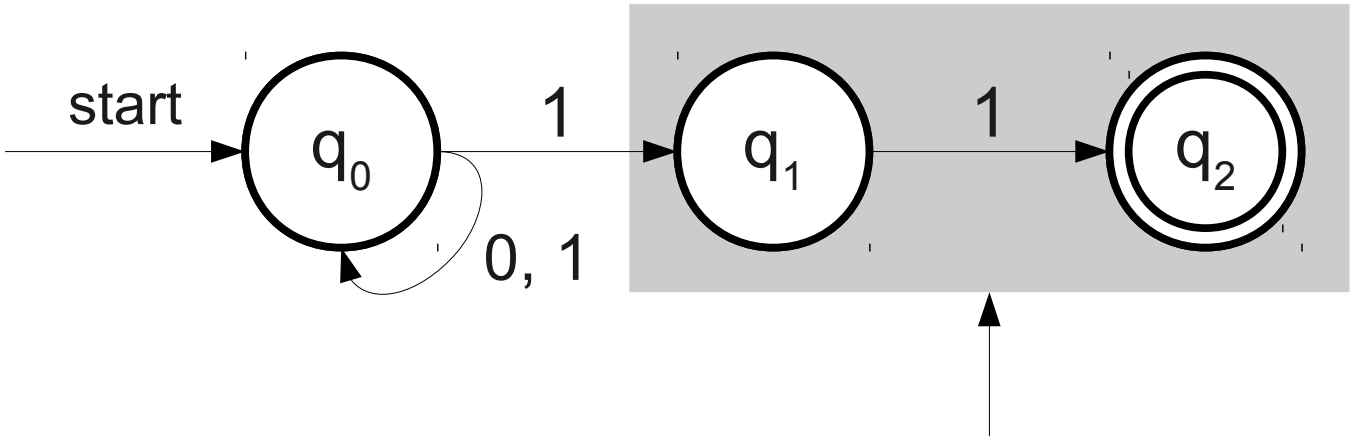


A More Complex NFA



These states don't have transitions defined on all symbols!

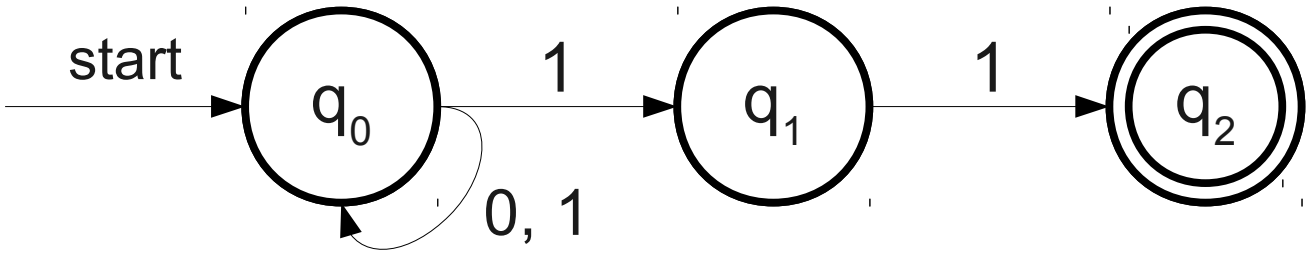
A More Complex NFA



These states don't have transitions defined on all symbols!

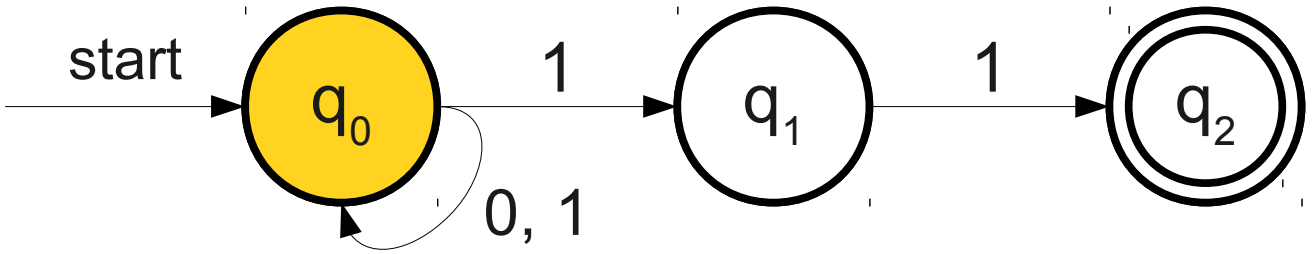
If a NFA needs to make a transition when no transition exists, the automaton dies and that particular path rejects.

A More Complex NFA



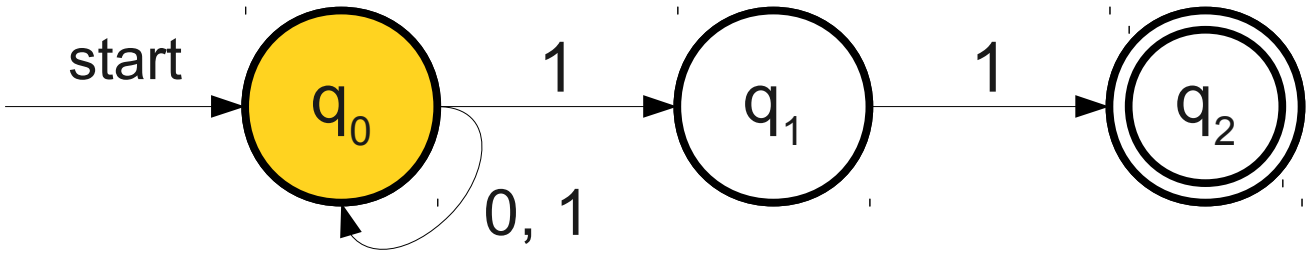
0 1 0 1 1

A More Complex NFA

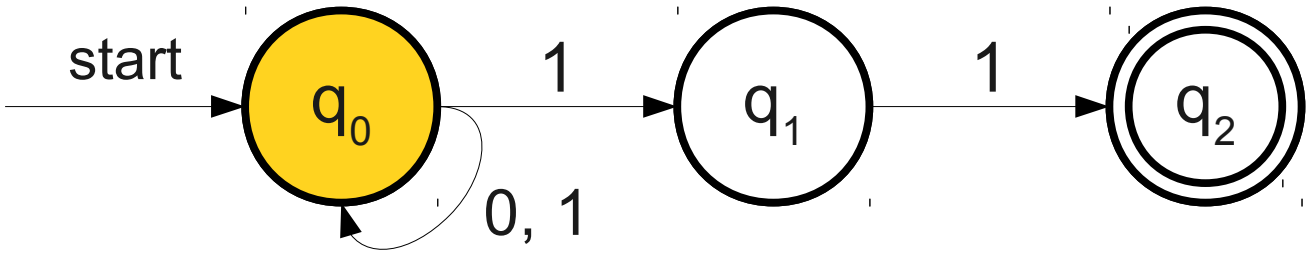


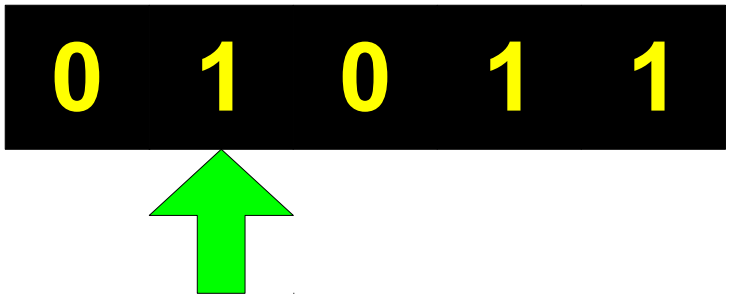
0 1 0 1 1

A More Complex NFA

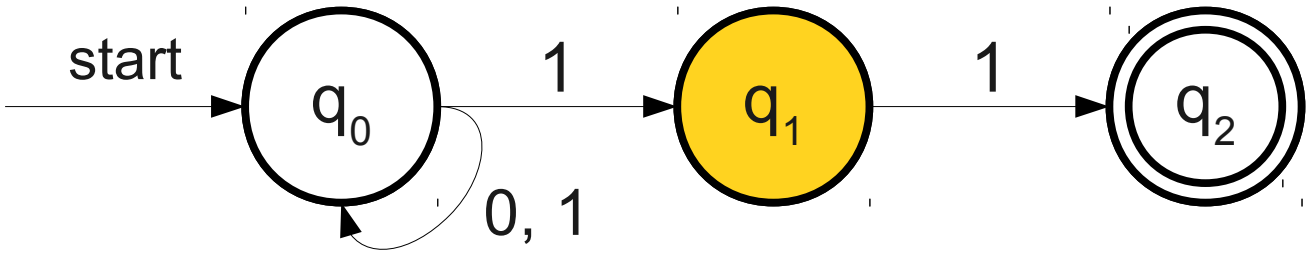


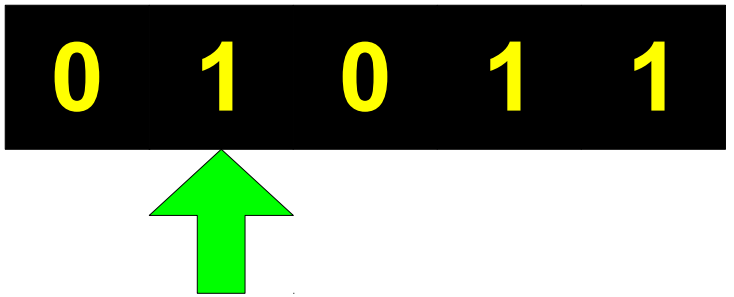
A More Complex NFA



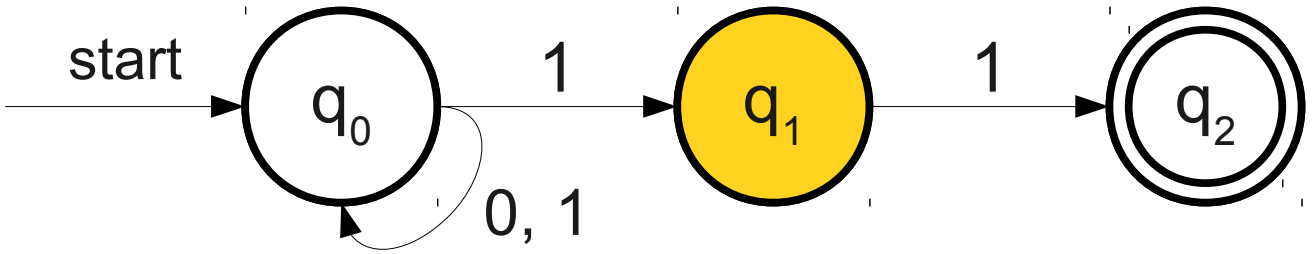


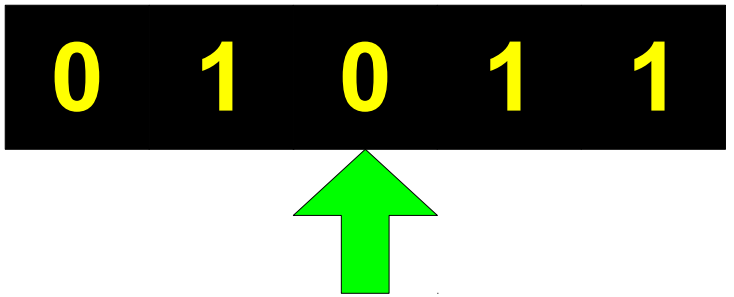
A More Complex NFA



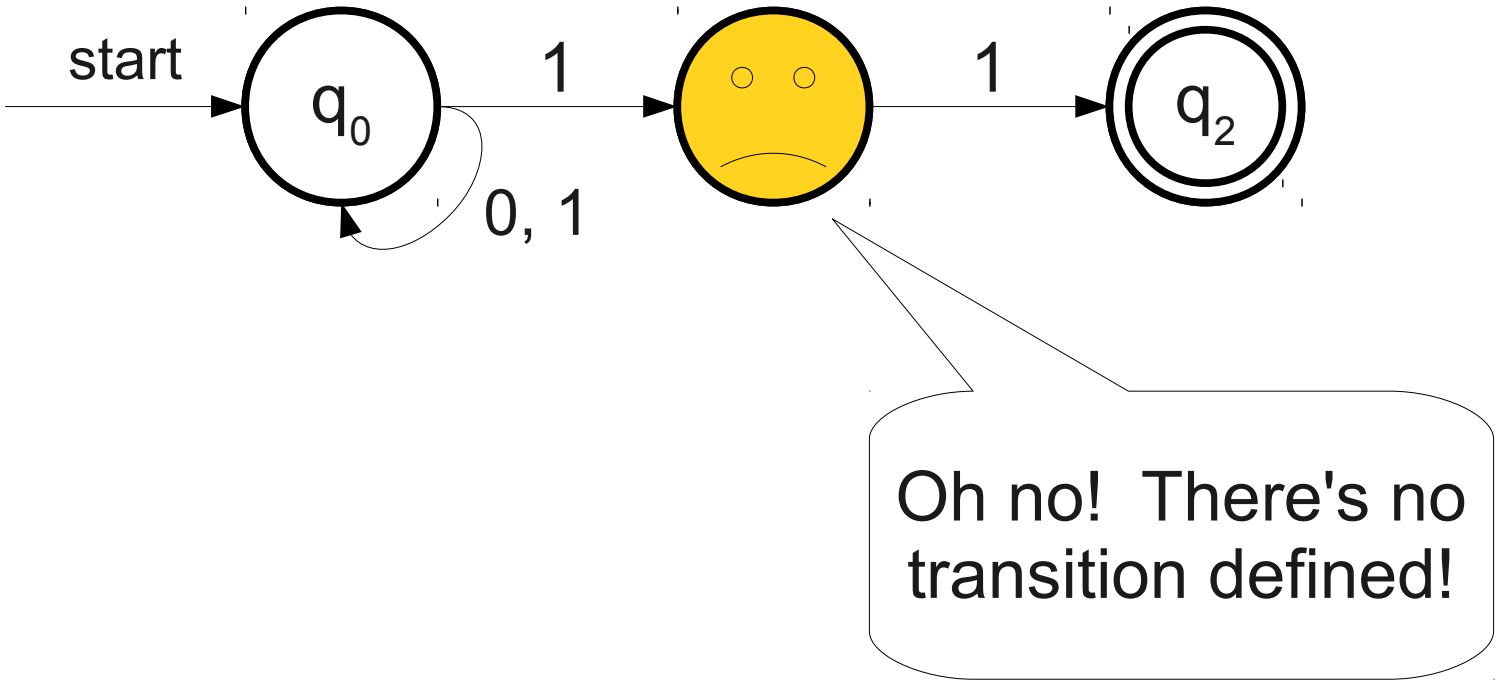


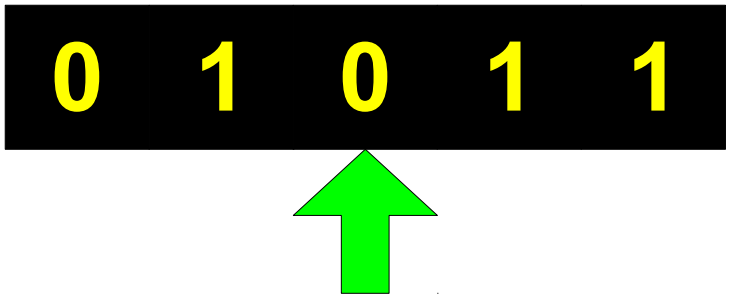
A More Complex NFA



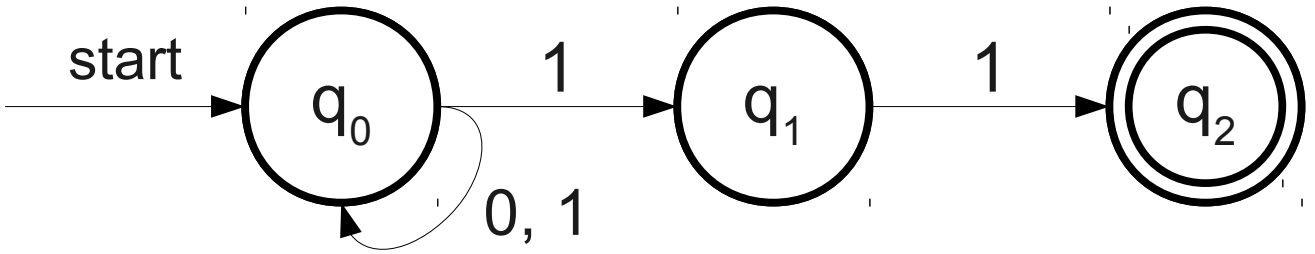


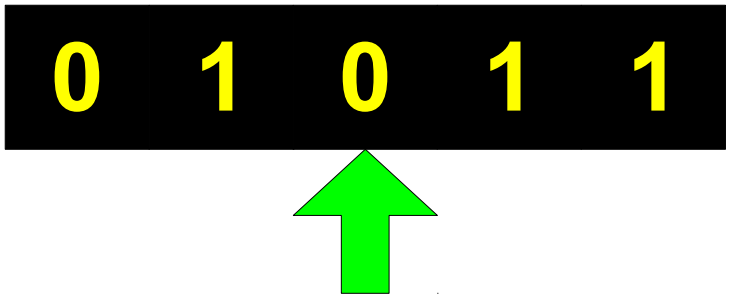
A More Complex NFA



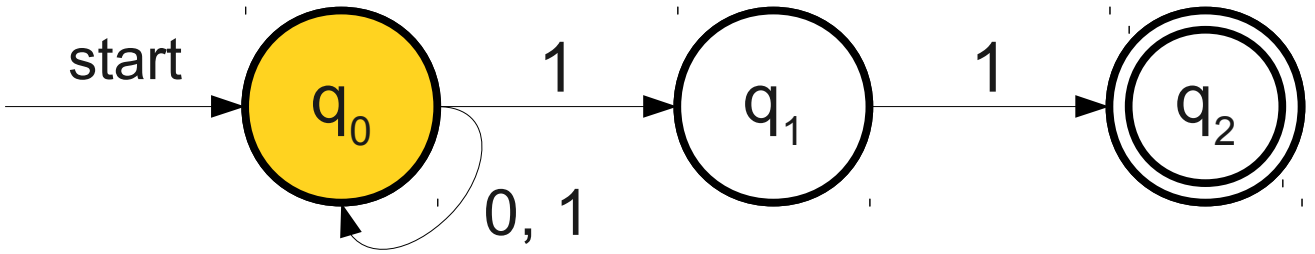


A More Complex NFA

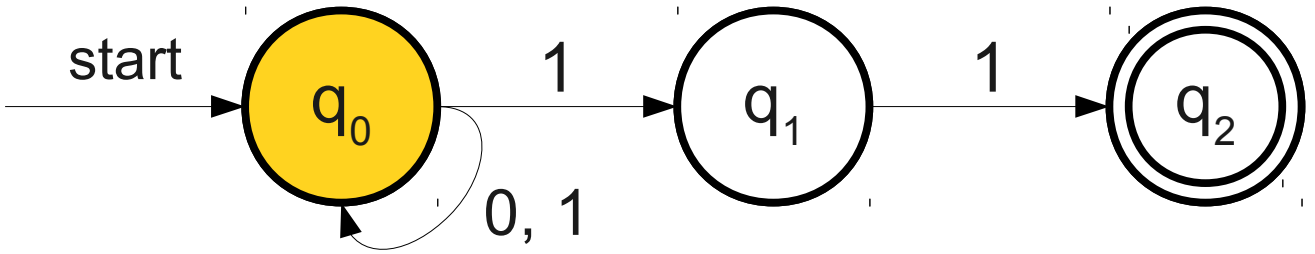


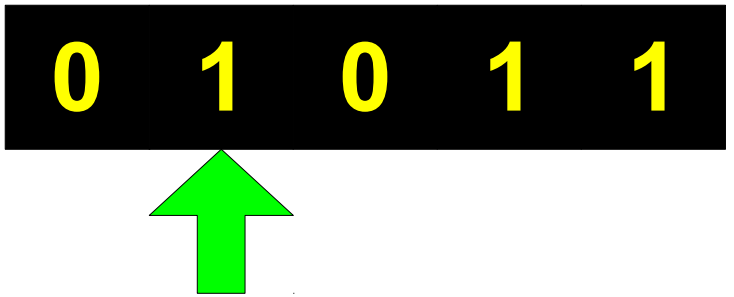


A More Complex NFA

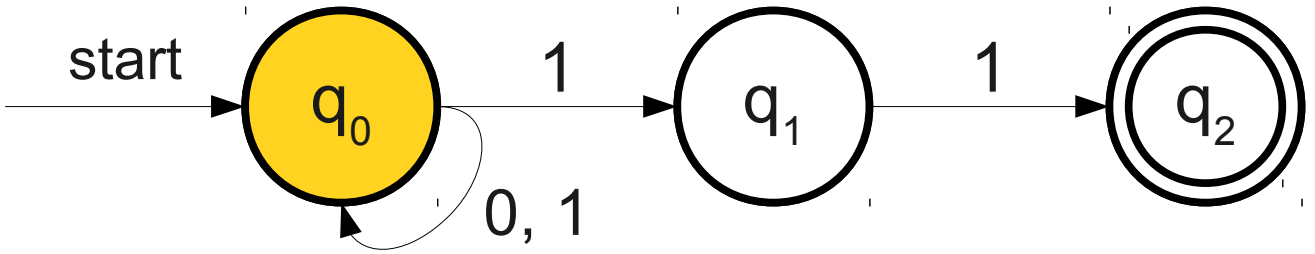


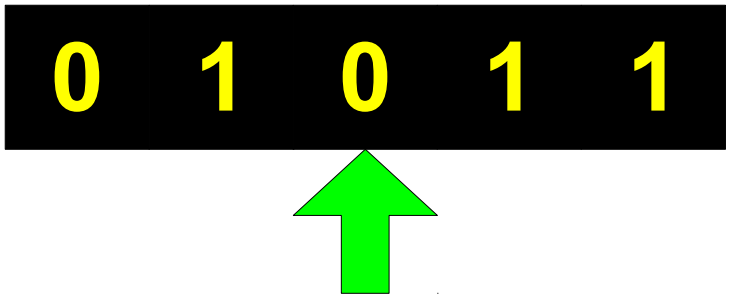
A More Complex NFA



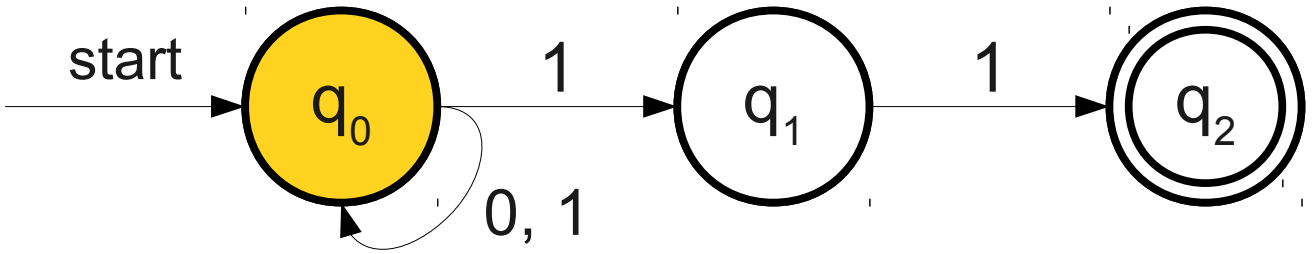


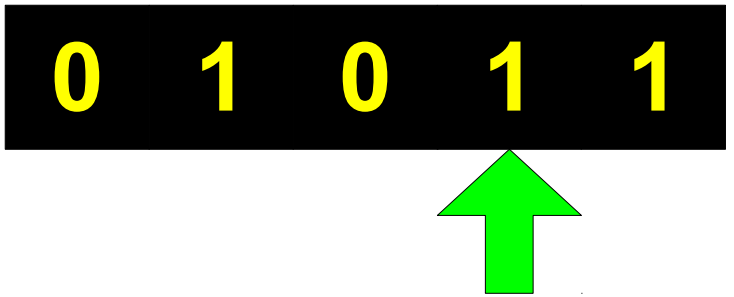
A More Complex NFA



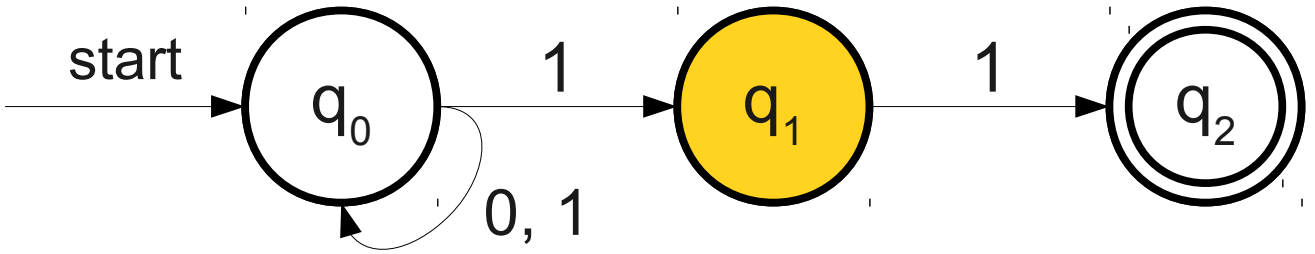


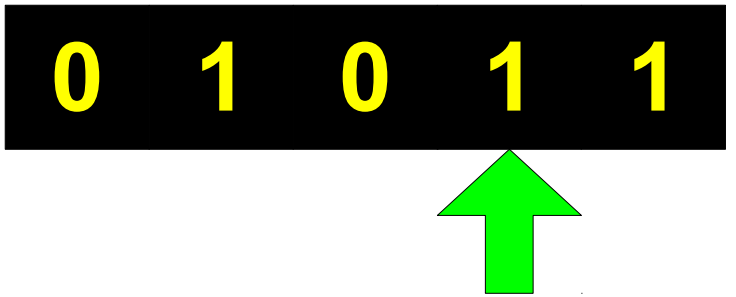
A More Complex NFA



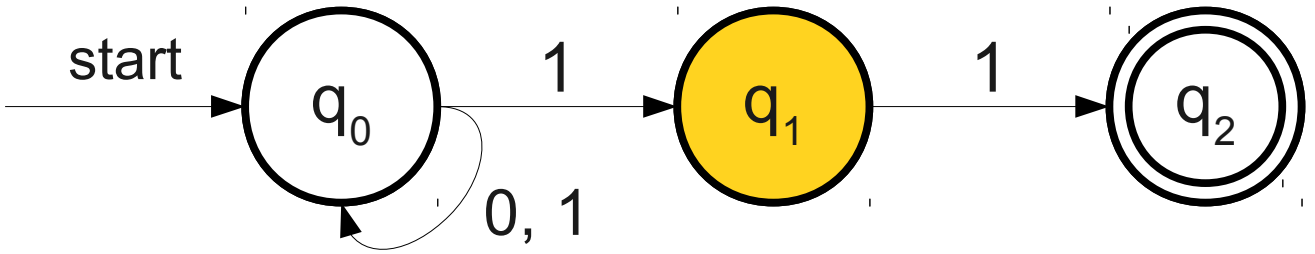


A More Complex NFA

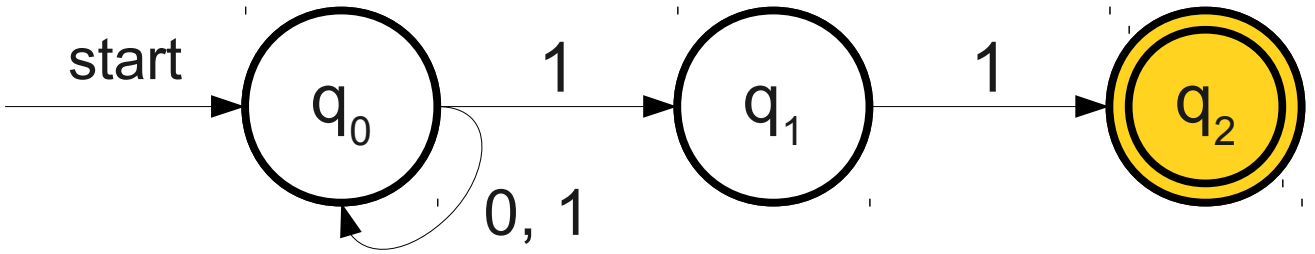




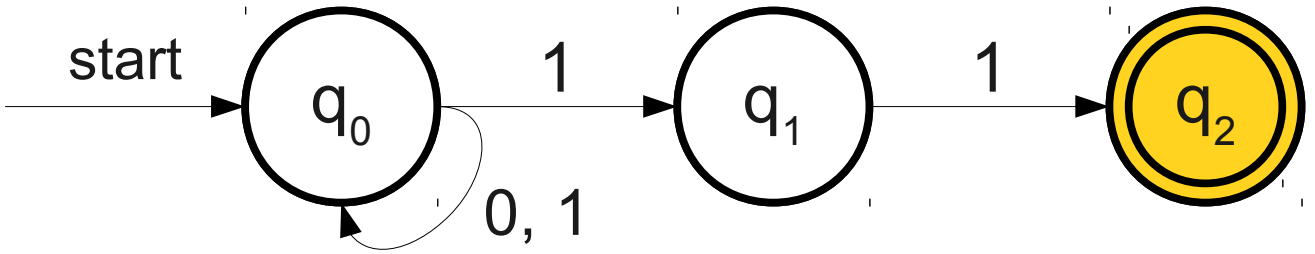
A More Complex NFA



A More Complex NFA

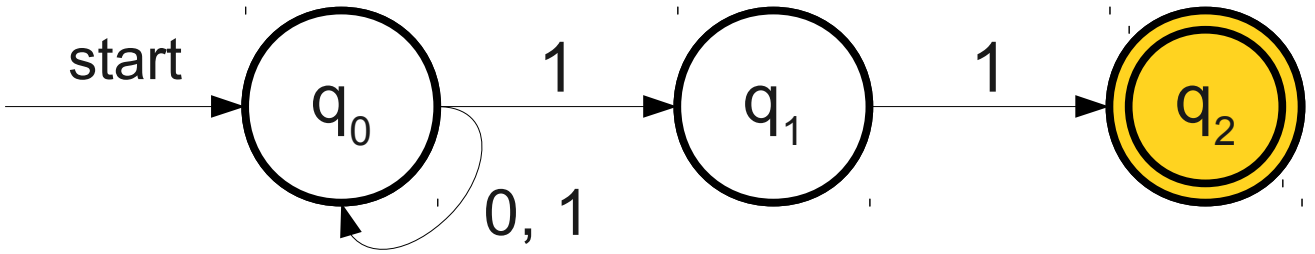


A More Complex NFA



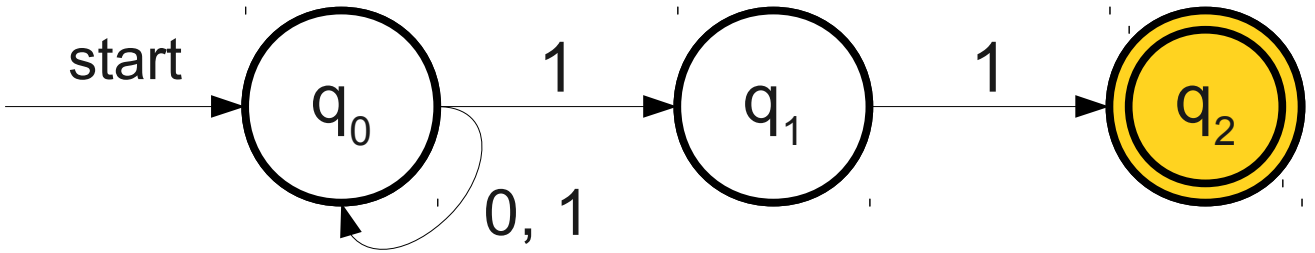
0 1 0 1 1

A More Complex NFA



0 1 0 1 1

A More Complex NFA

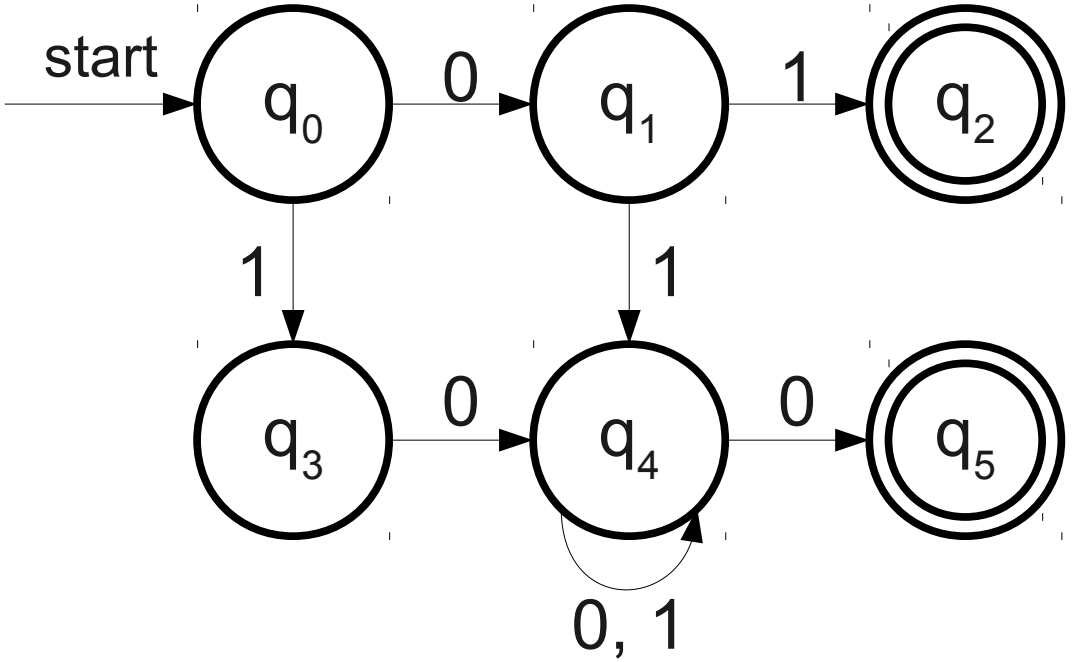


0 1 0 1 1

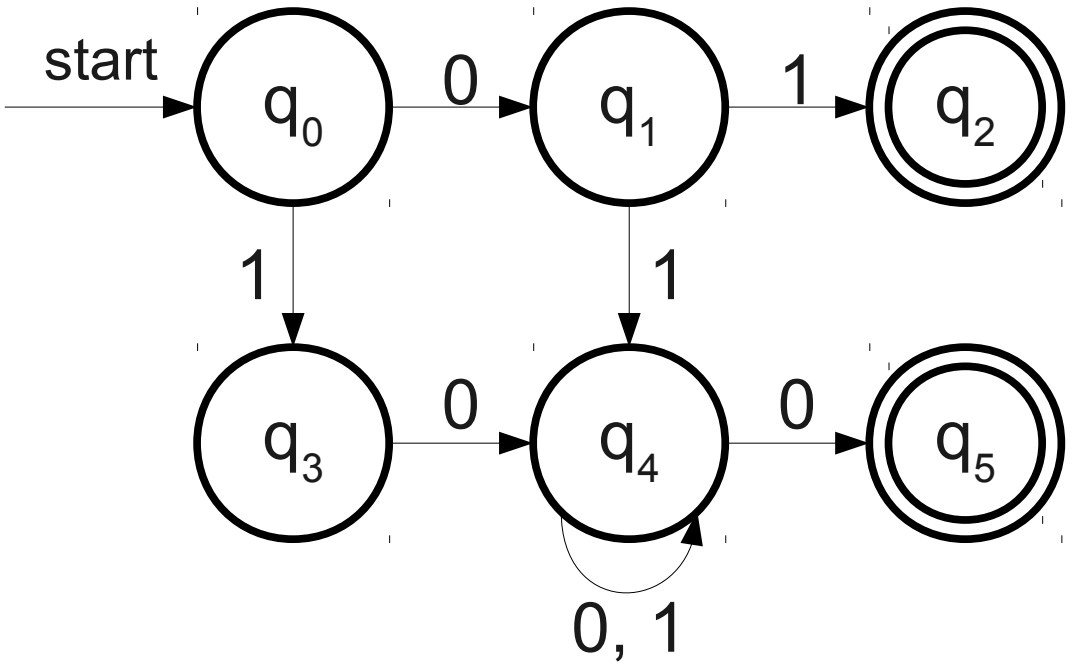
Intuiting Nondeterminism

- Nondeterministic machines are a serious departure from physical computers.
- How can we build up an intuition for them?
- Three approaches:
  - **Tree computation**
  - **Perfect guessing**
  - **Massive parallelism**

Tree Computation

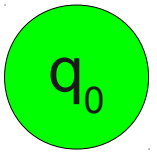


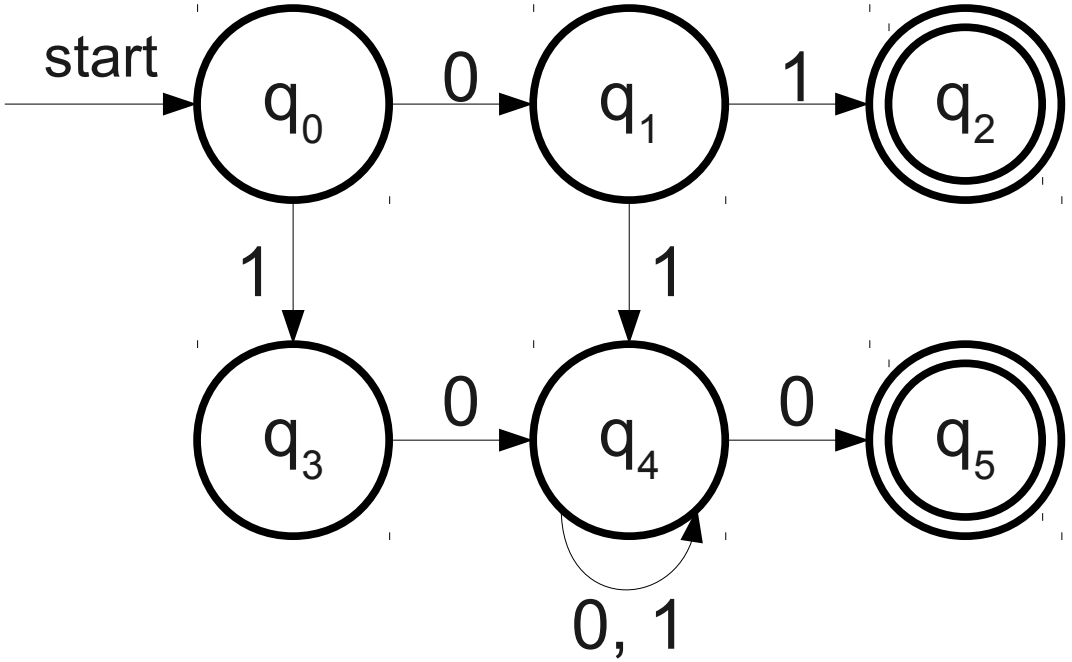
Tree Computation



0 1 0 1 0

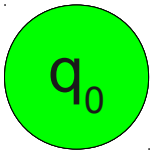
Tree Computation

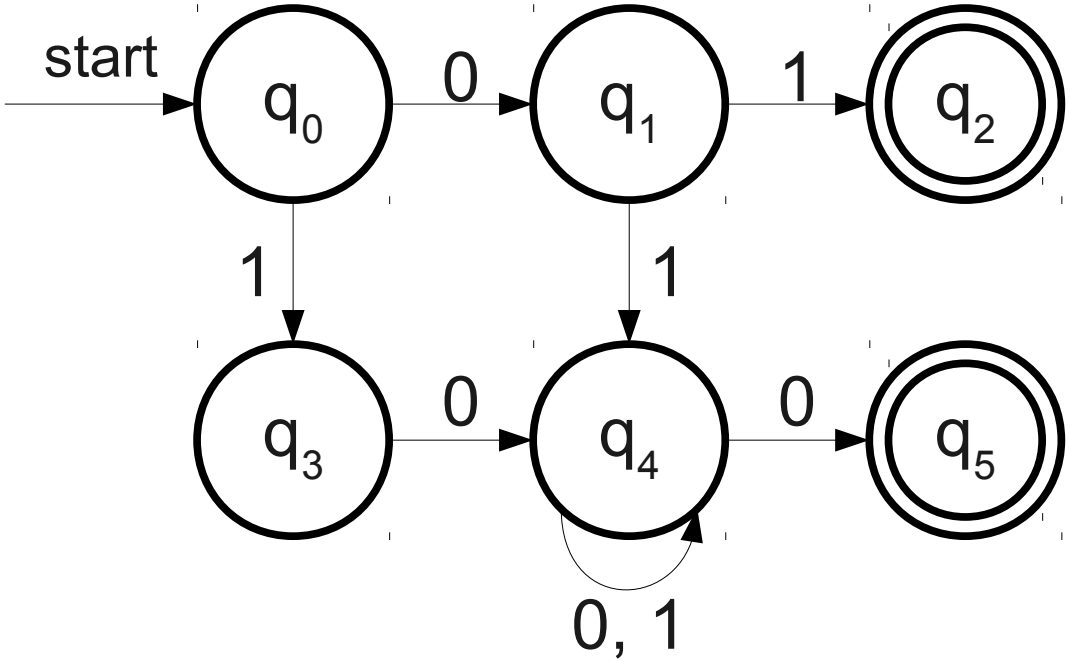




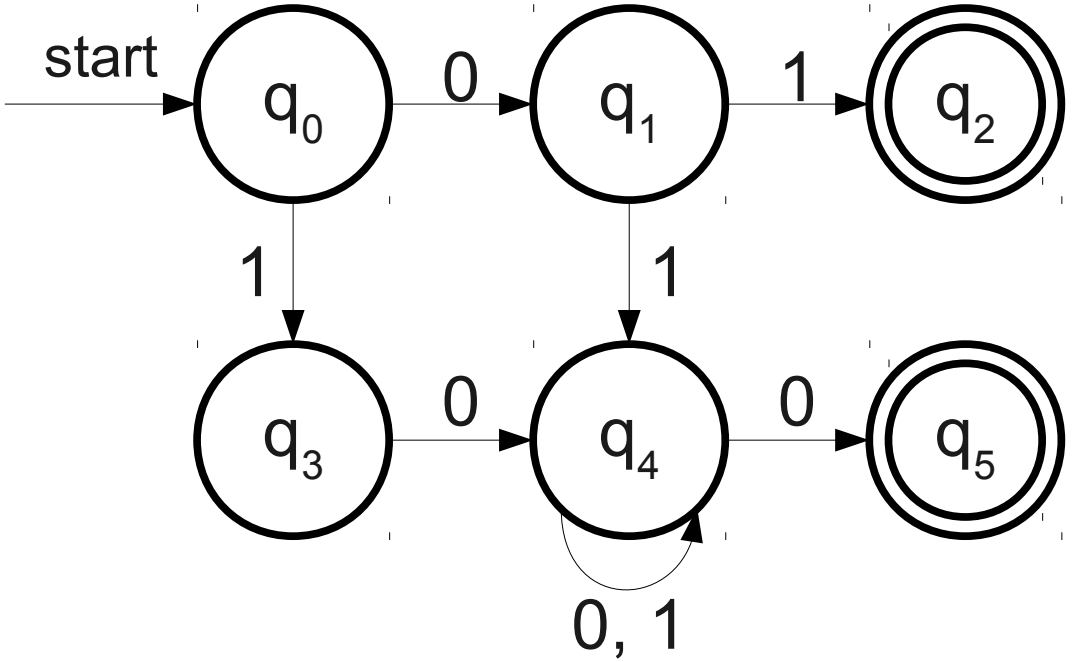
0 1 0 1 0

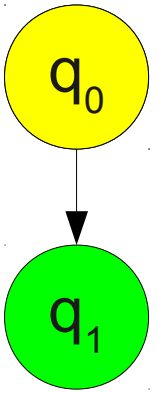
Tree Computation



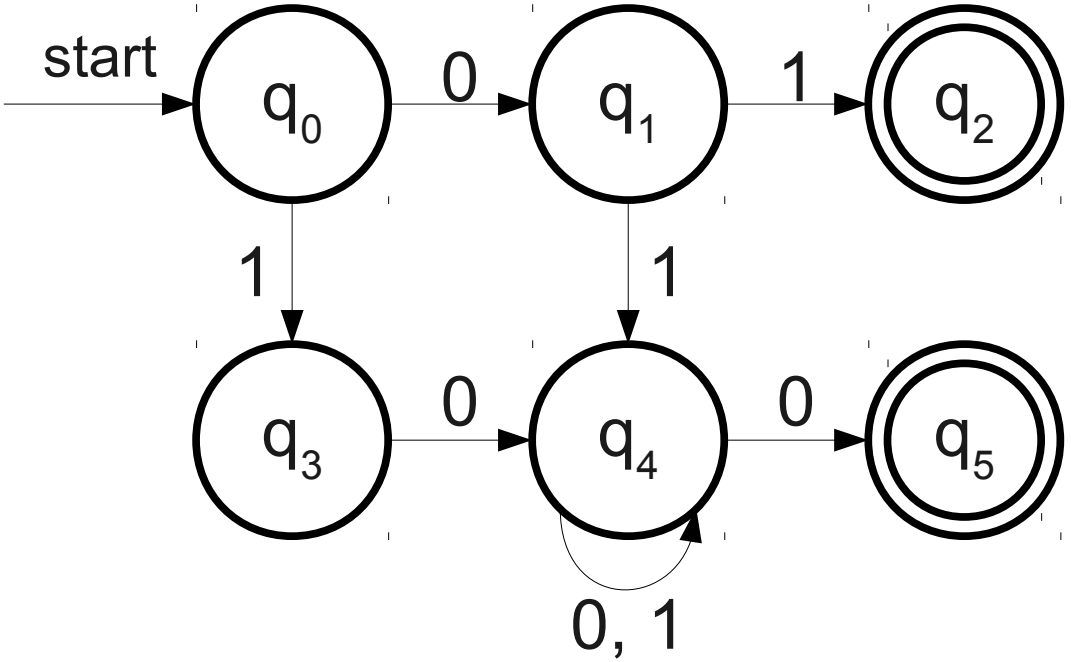


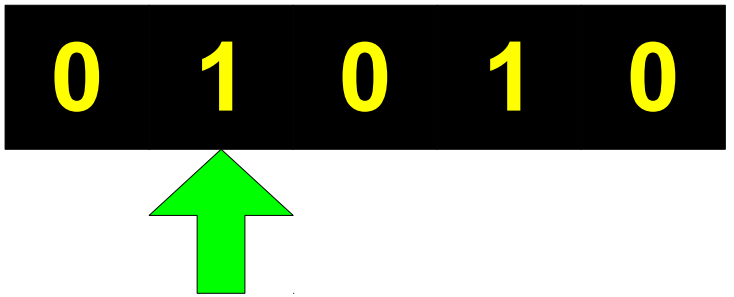
Tree Computation

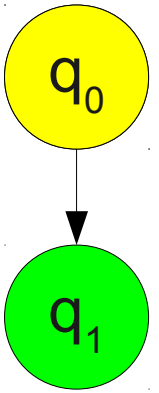




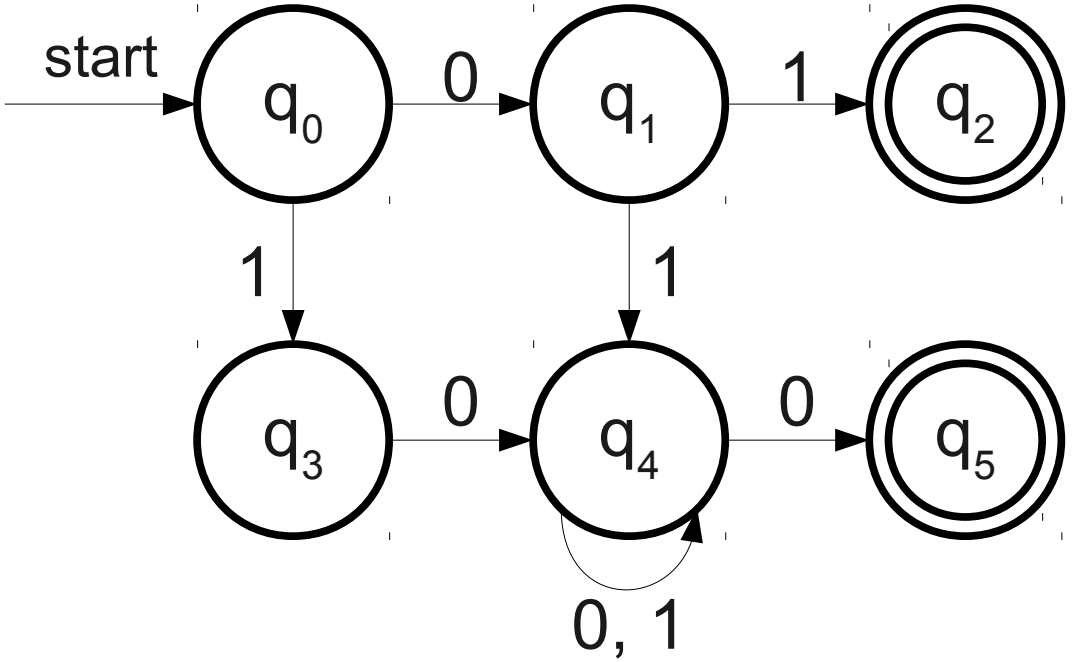
Tree Computation

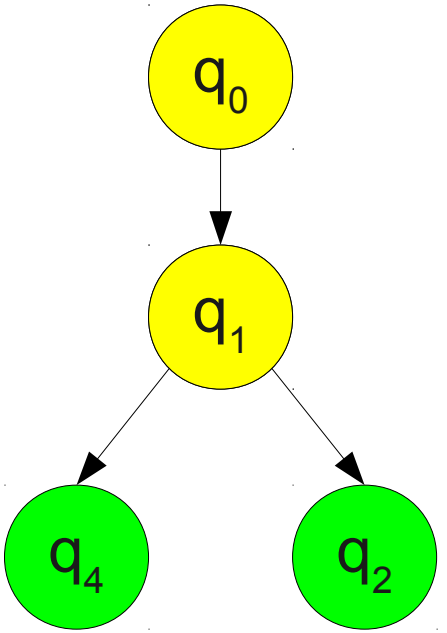


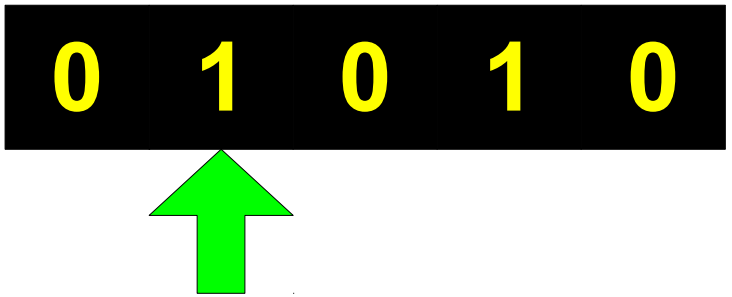




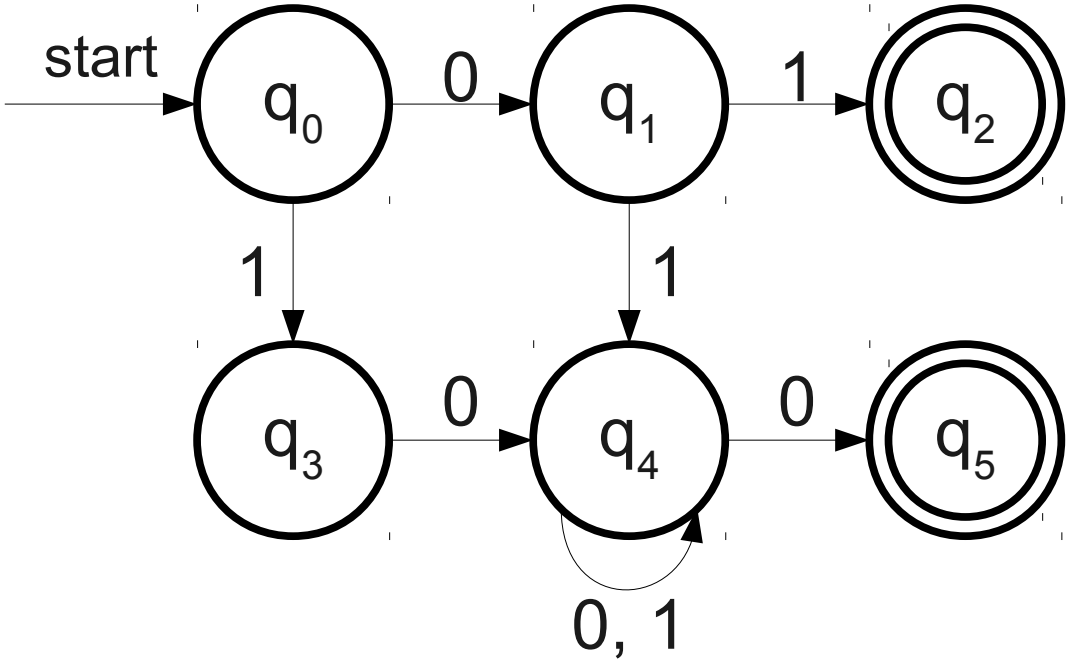
Tree Computation

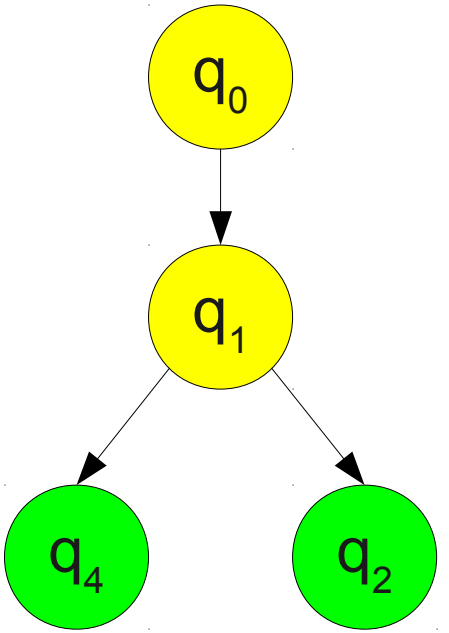


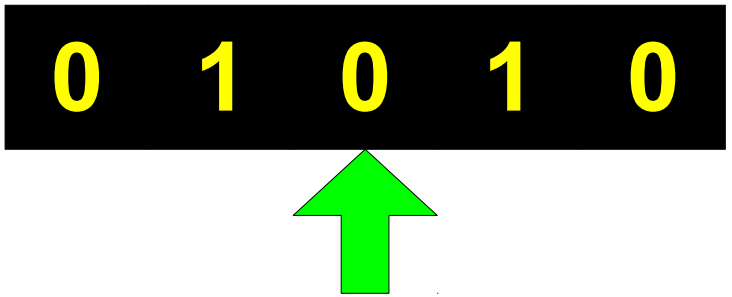




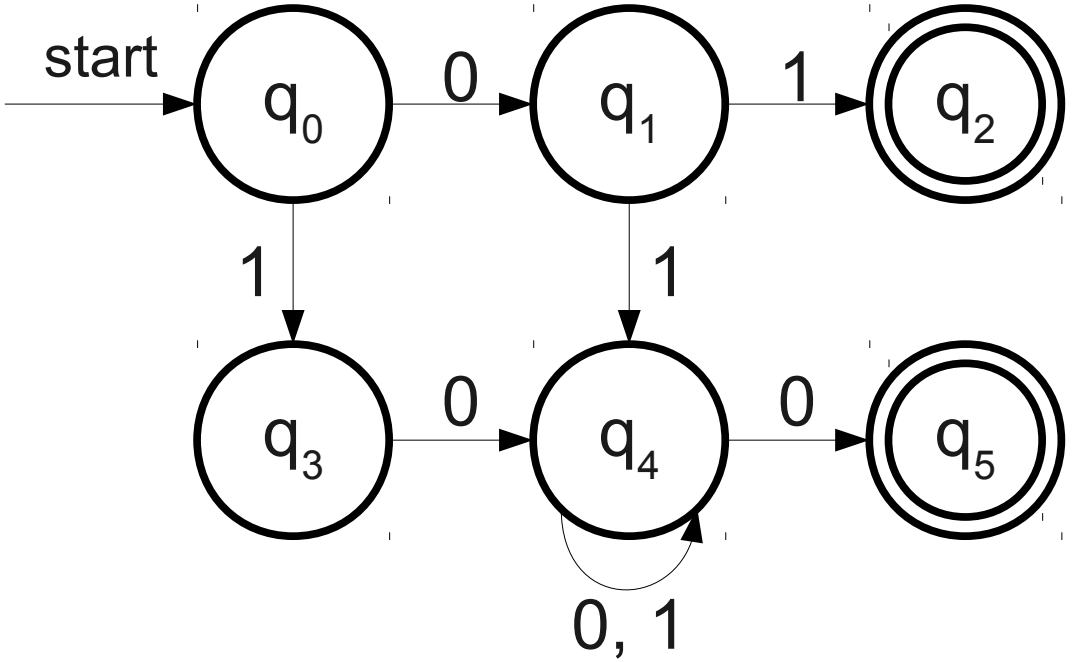
Tree Computation

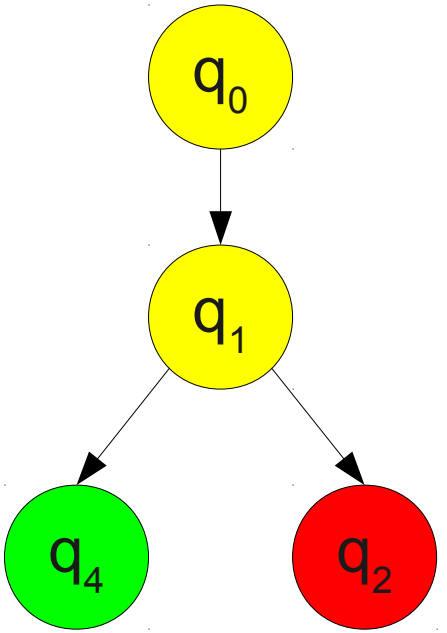


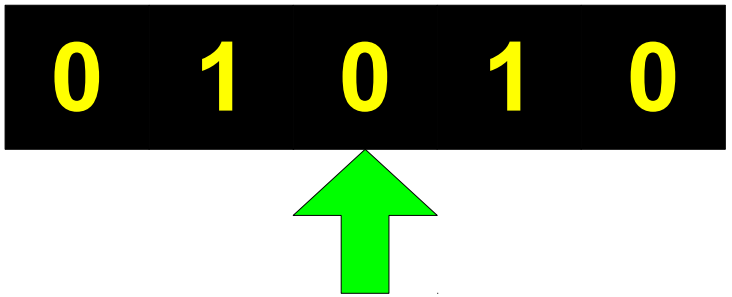




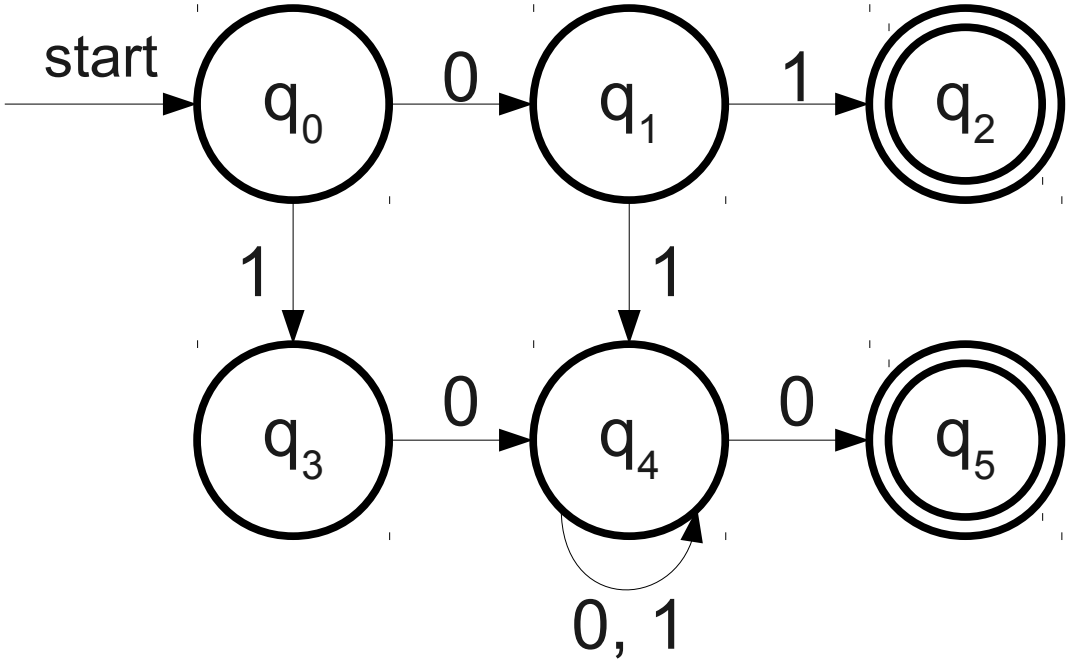
Tree Computation

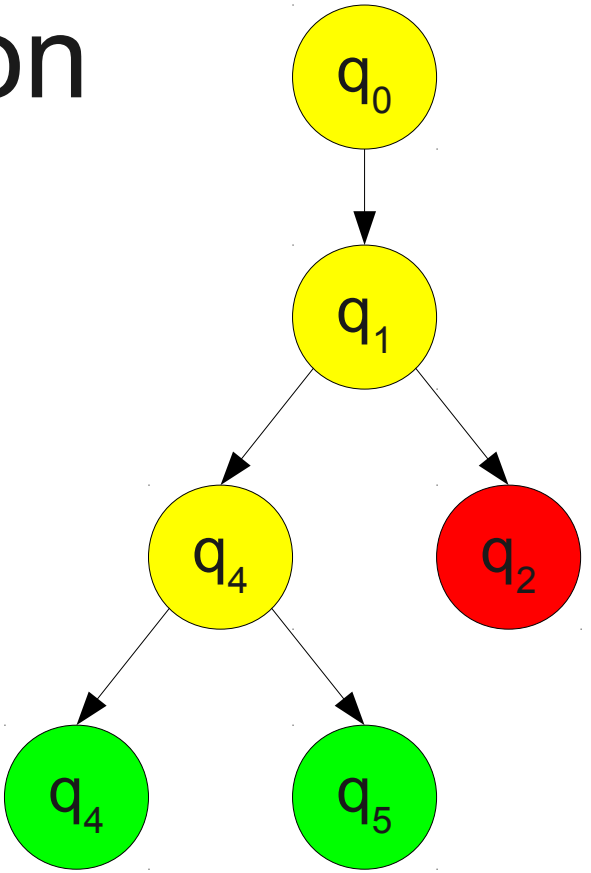


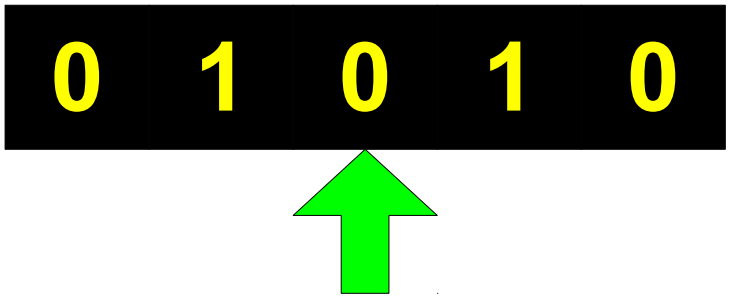




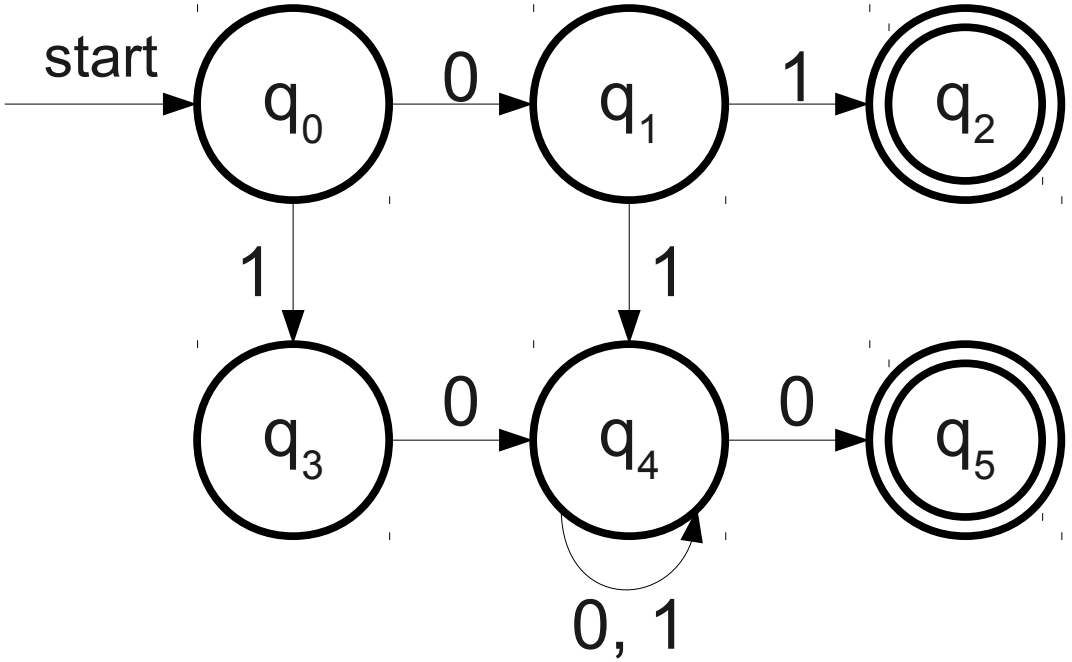
Tree Computation

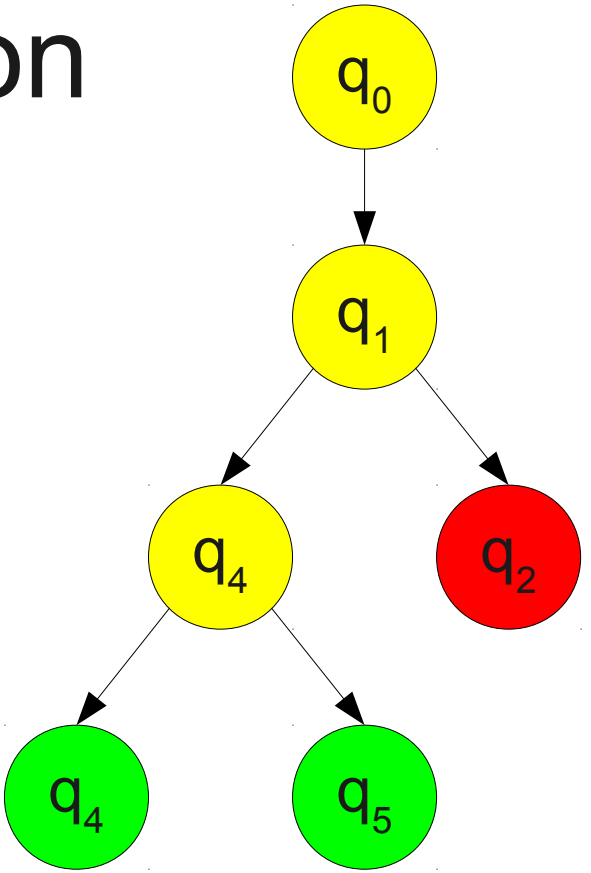


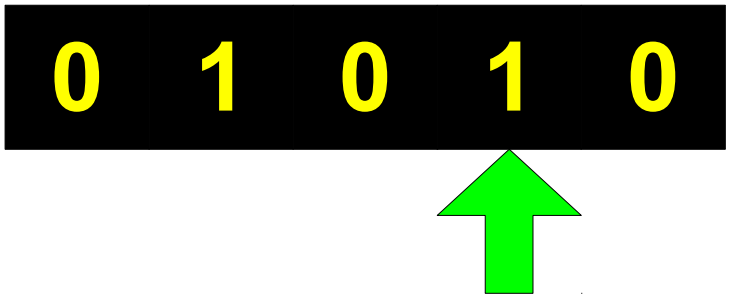




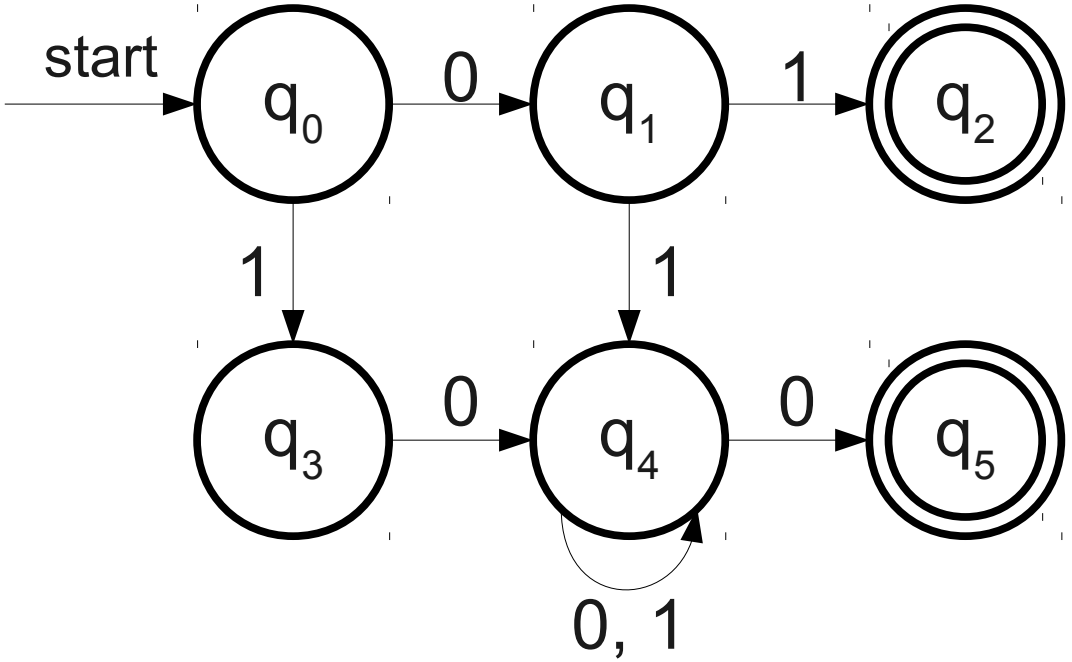
Tree Computation

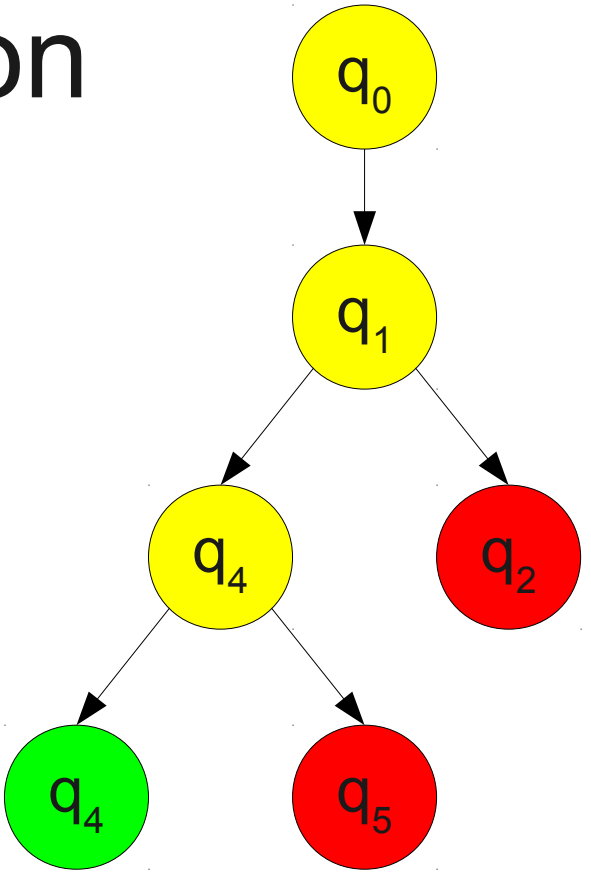


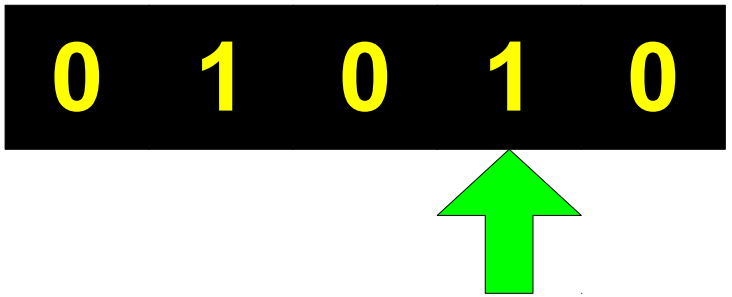




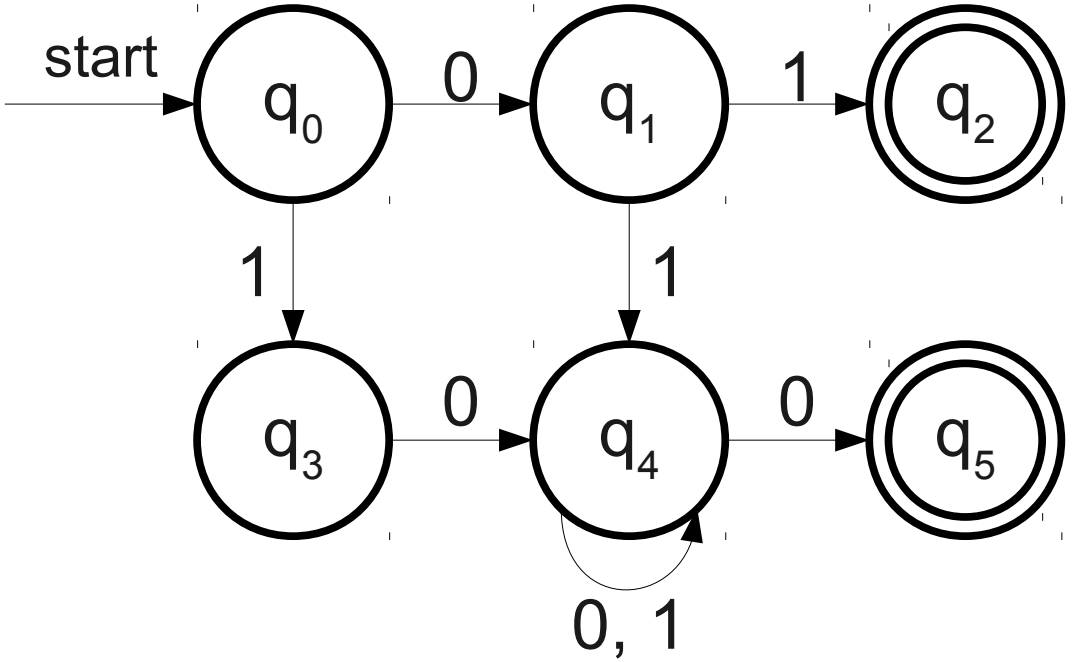
Tree Computation

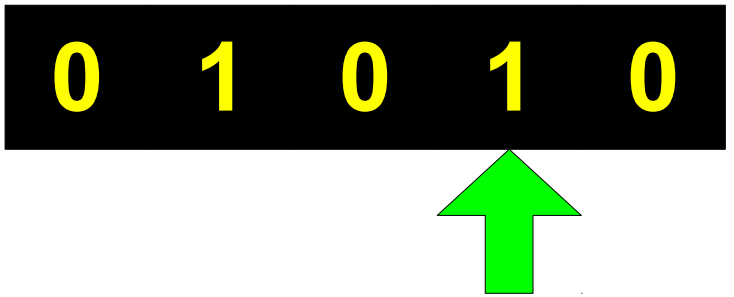


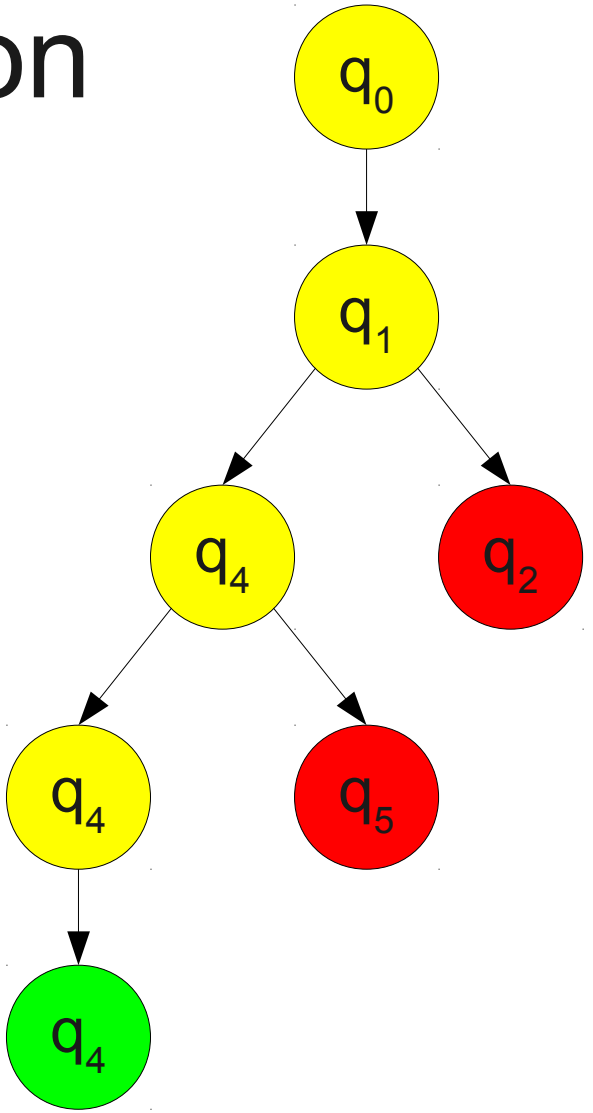




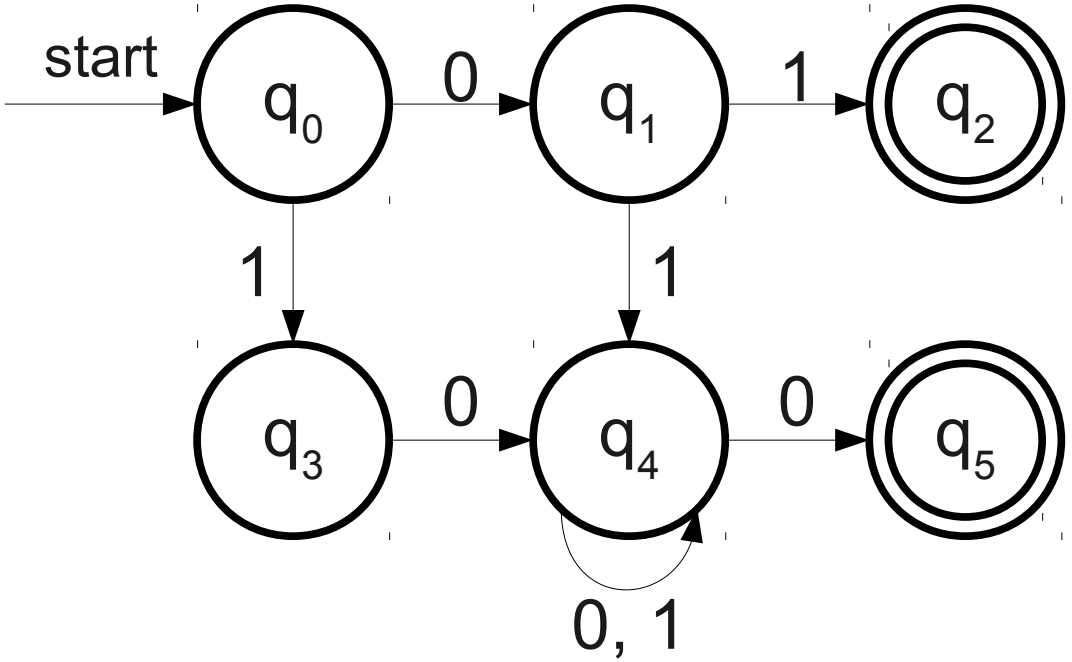
Tree Computation

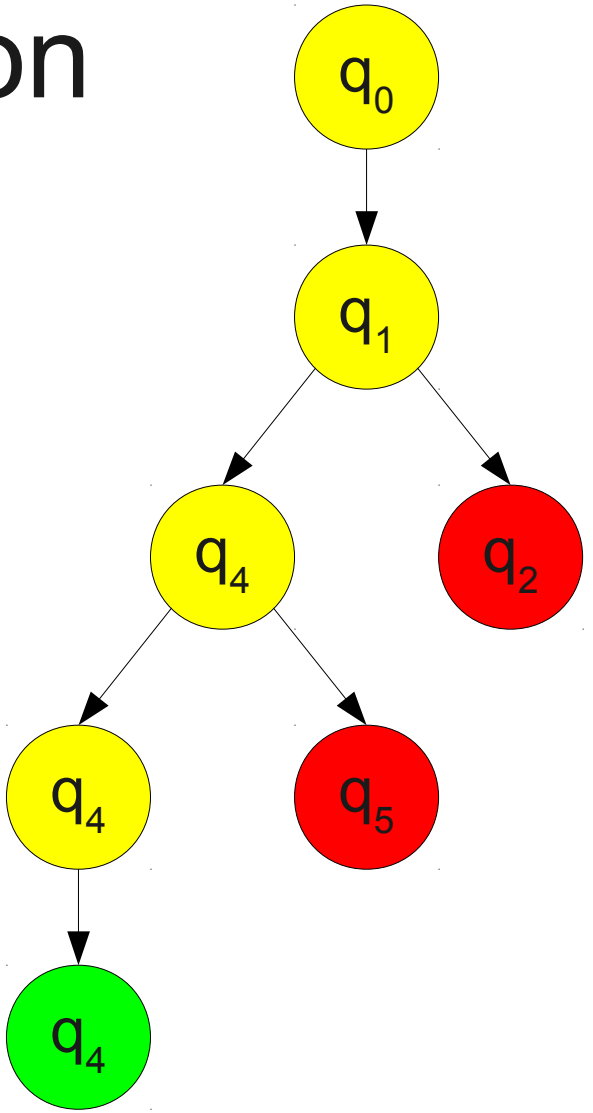




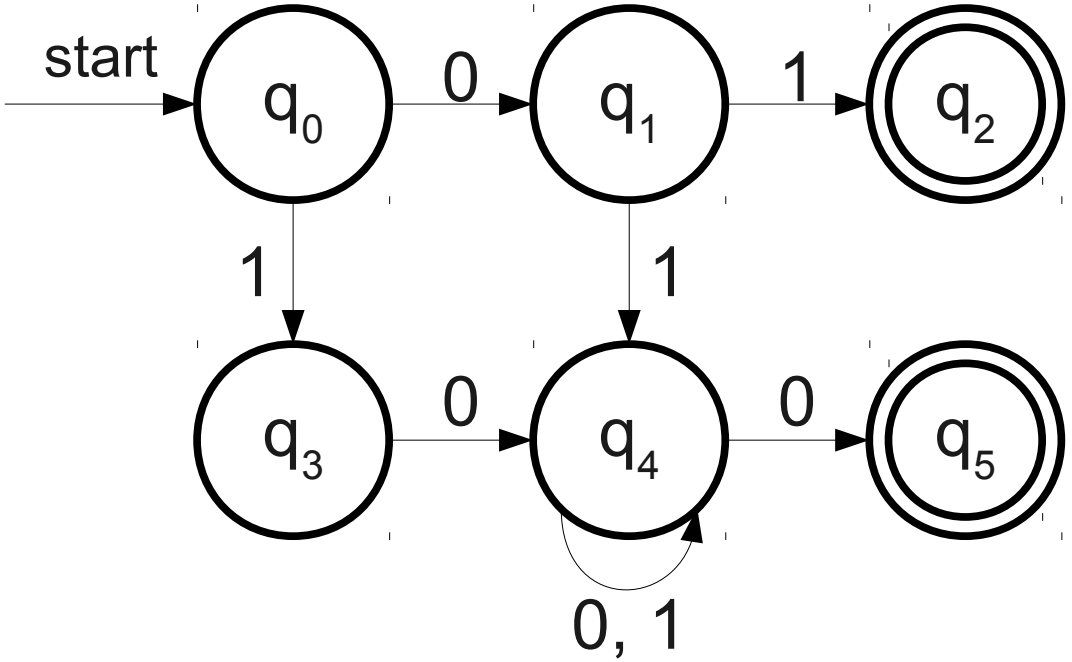


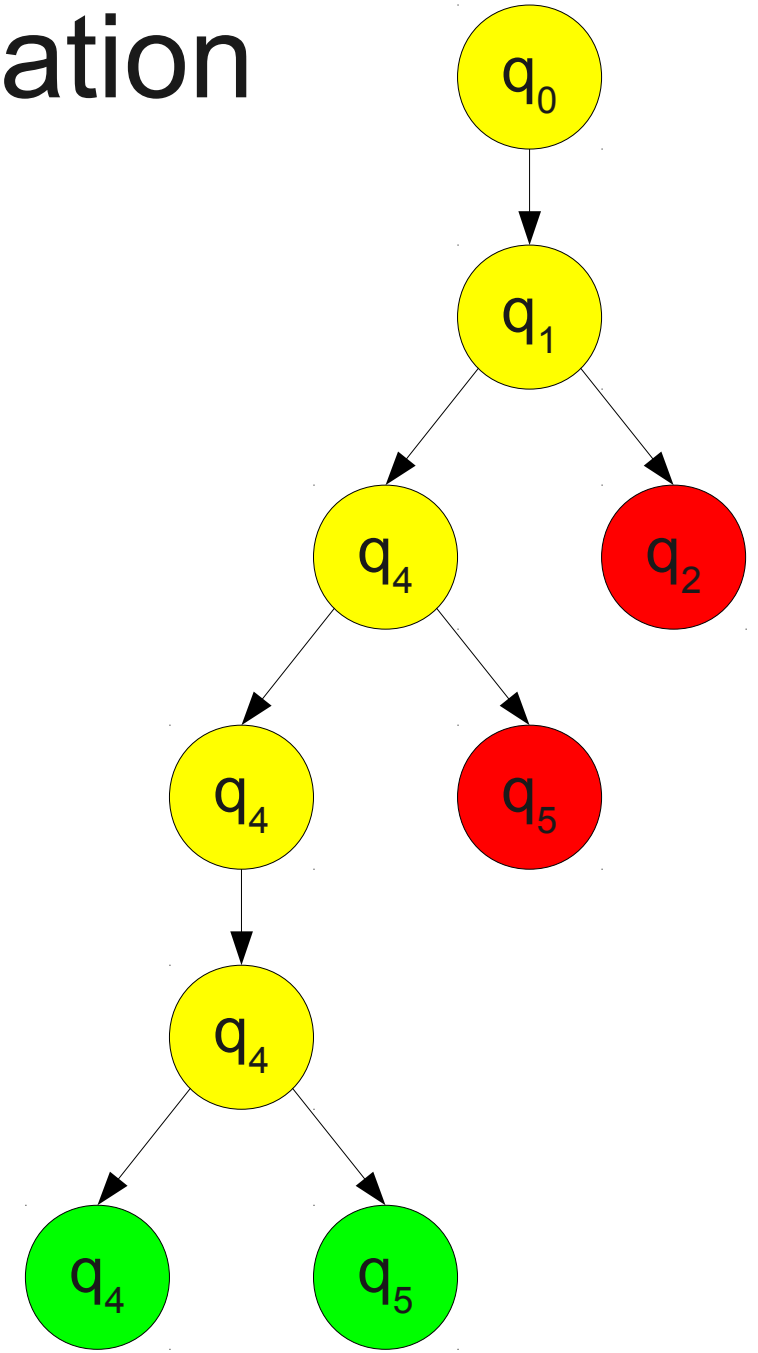
Tree Computation



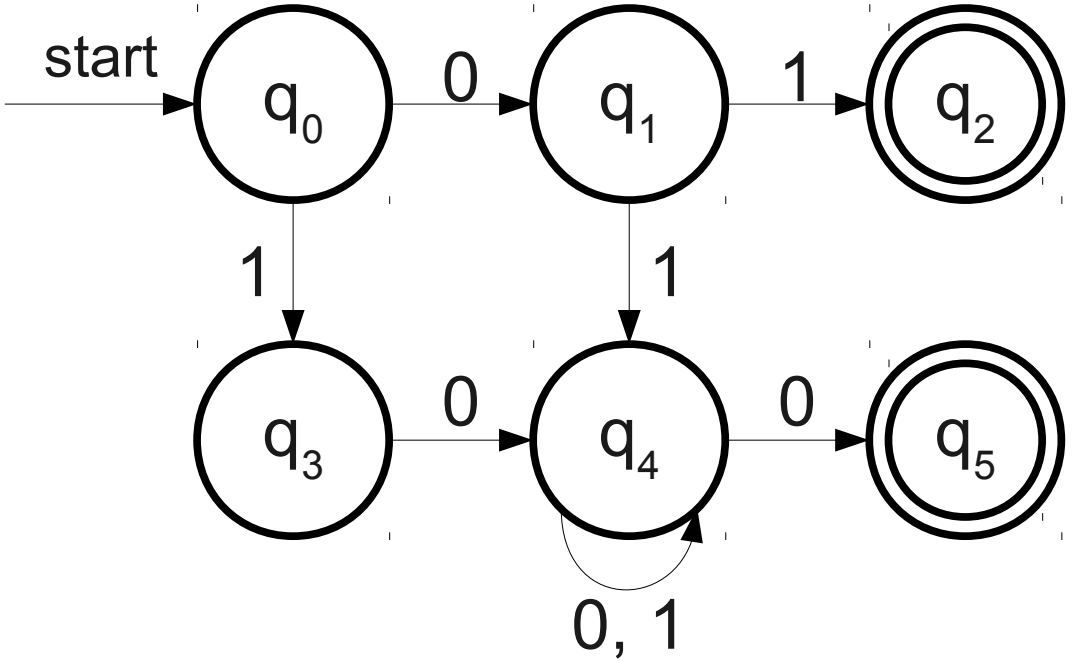


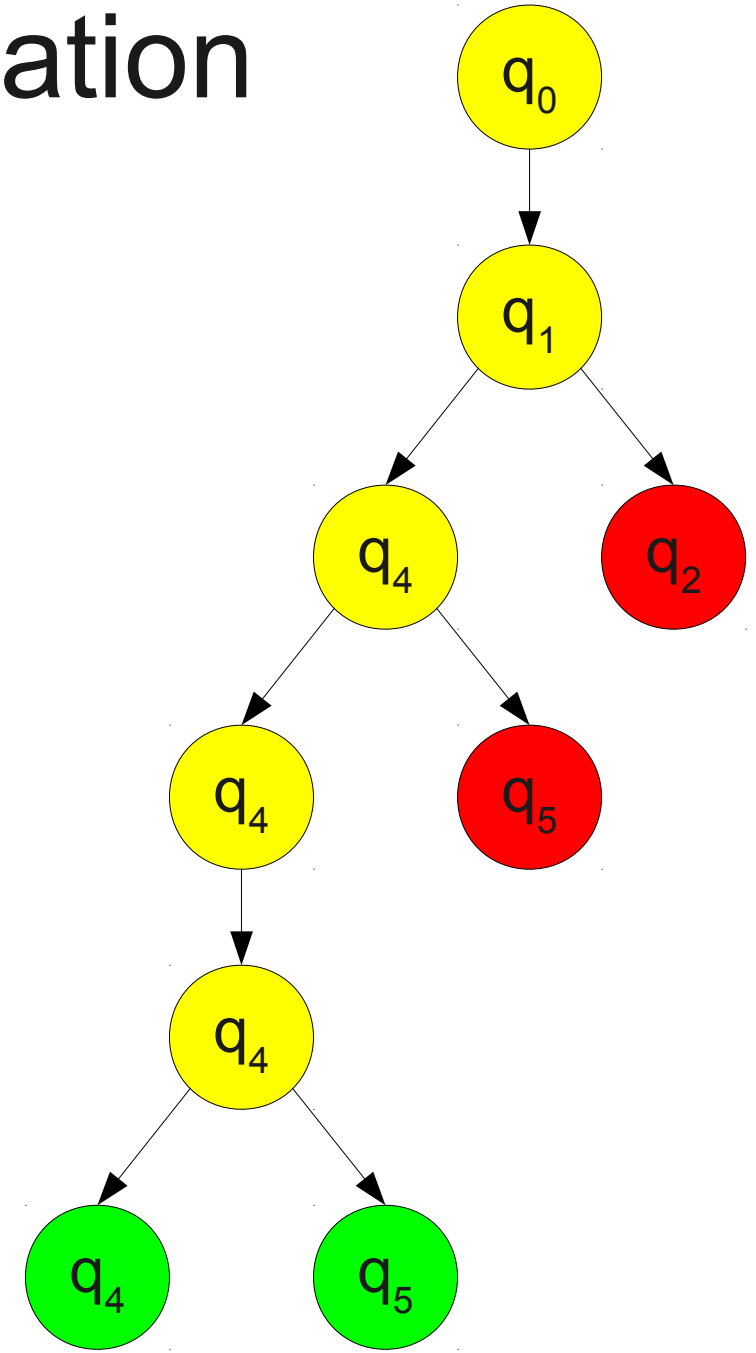
Tree Computation





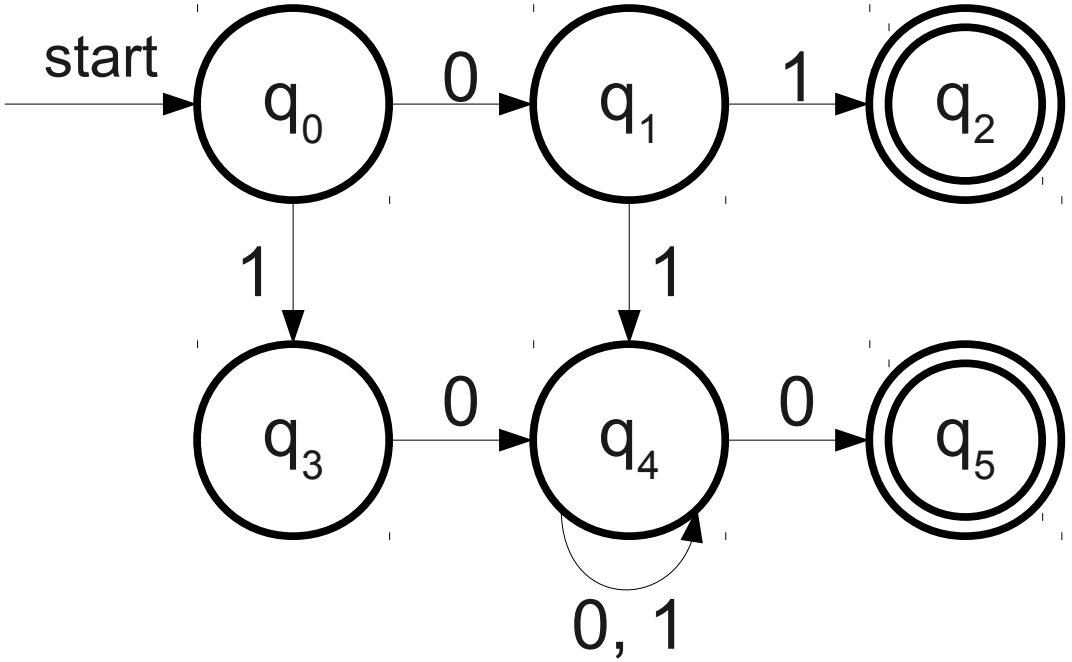
Tree Computation



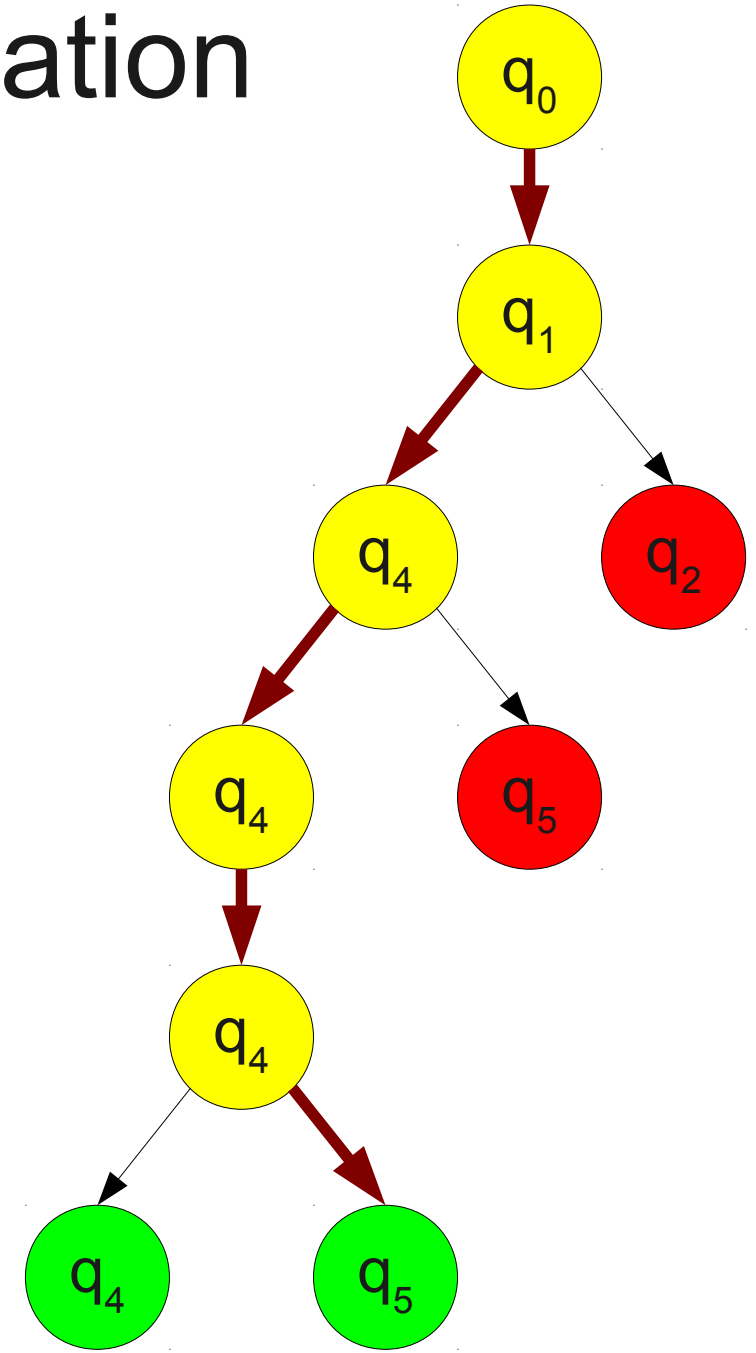


0 1 0 1 0

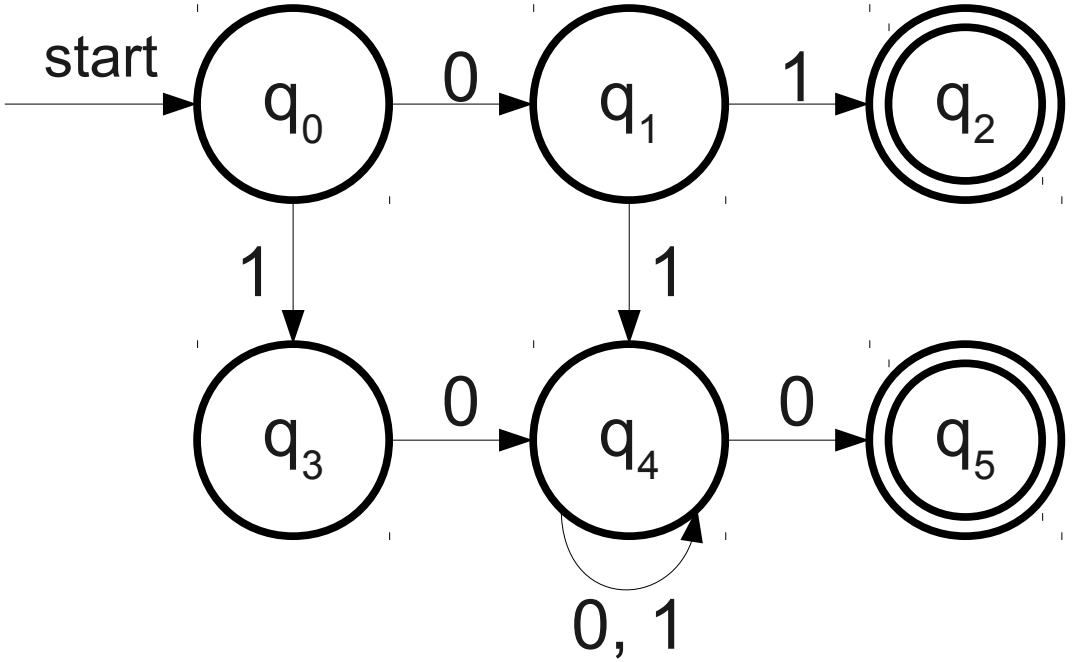
Tree Computation

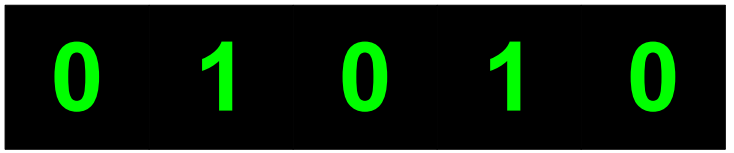


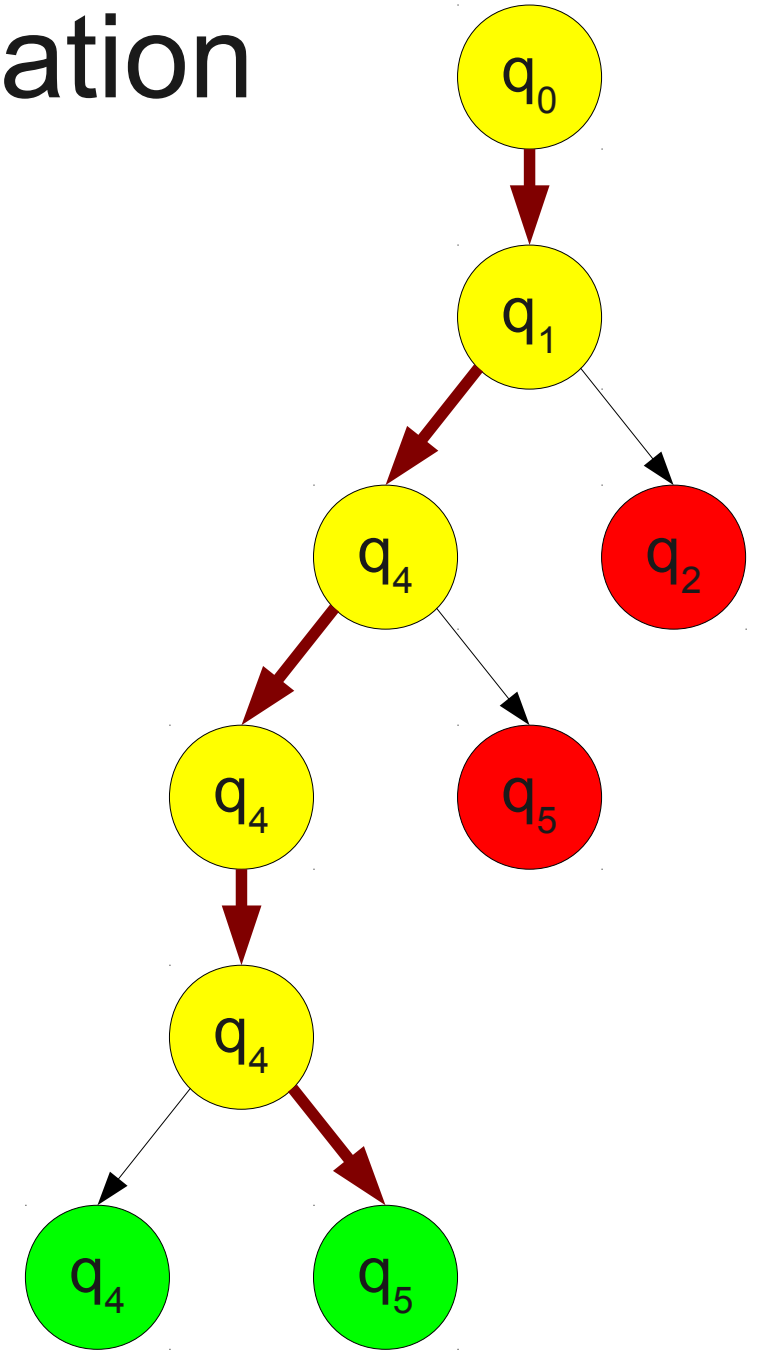
0 1 0 1 0



Tree Computation



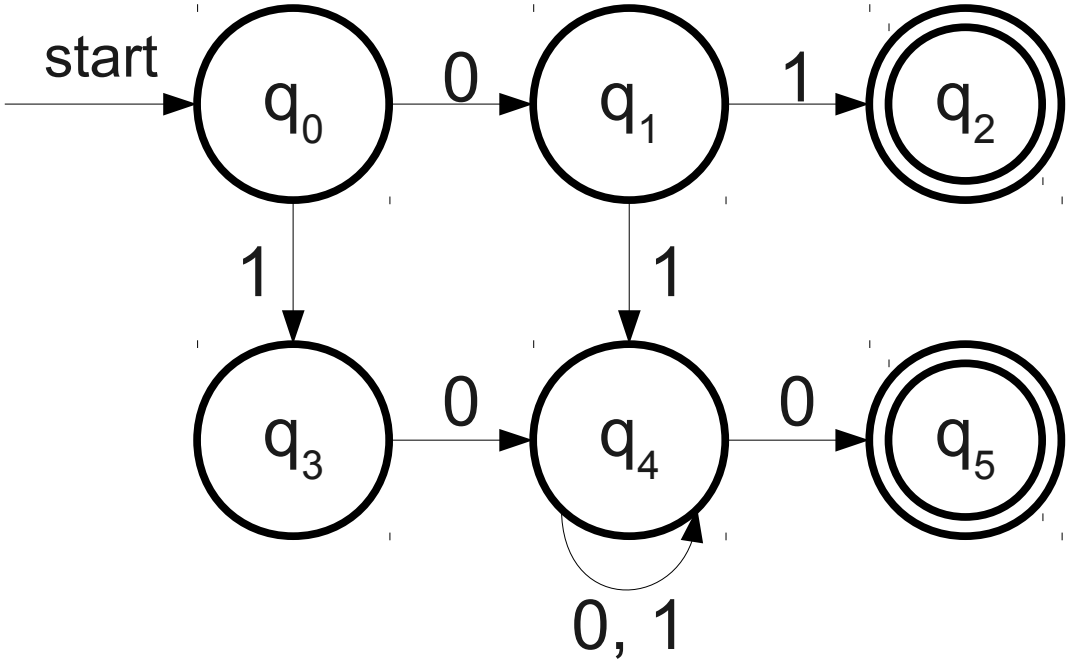




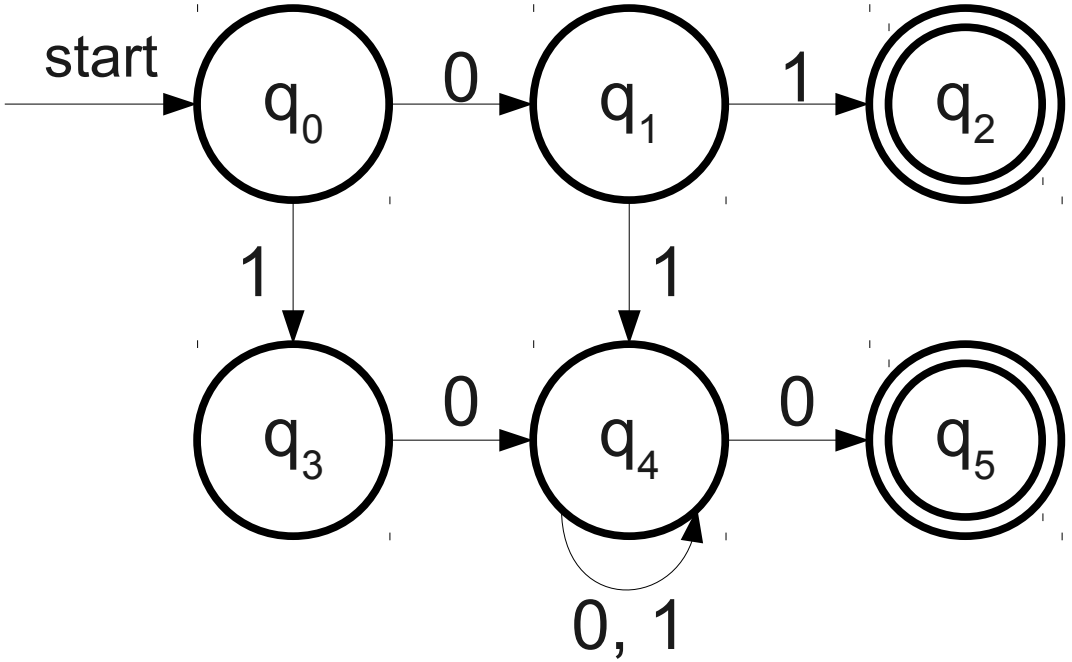
Nondeterminism as a Tree

- At each decision point, the automaton clones itself for each possible decision.
- The series of choices forms a directed, rooted tree.
- At the end, if any active accepting states remain, we accept.

Perfect Guessing

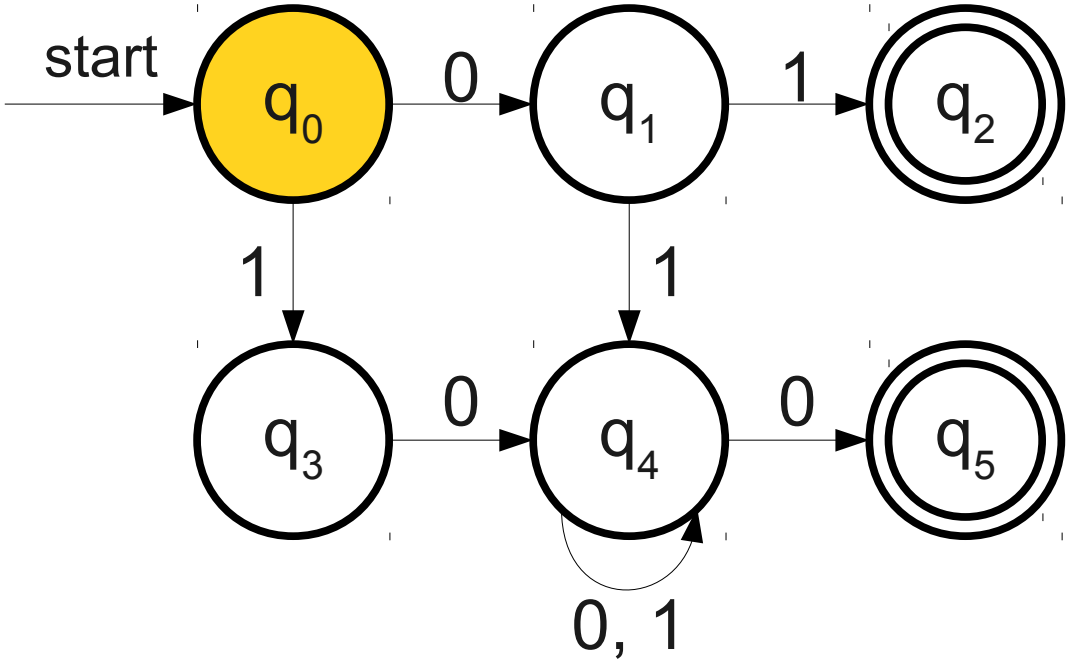


Perfect Guessing



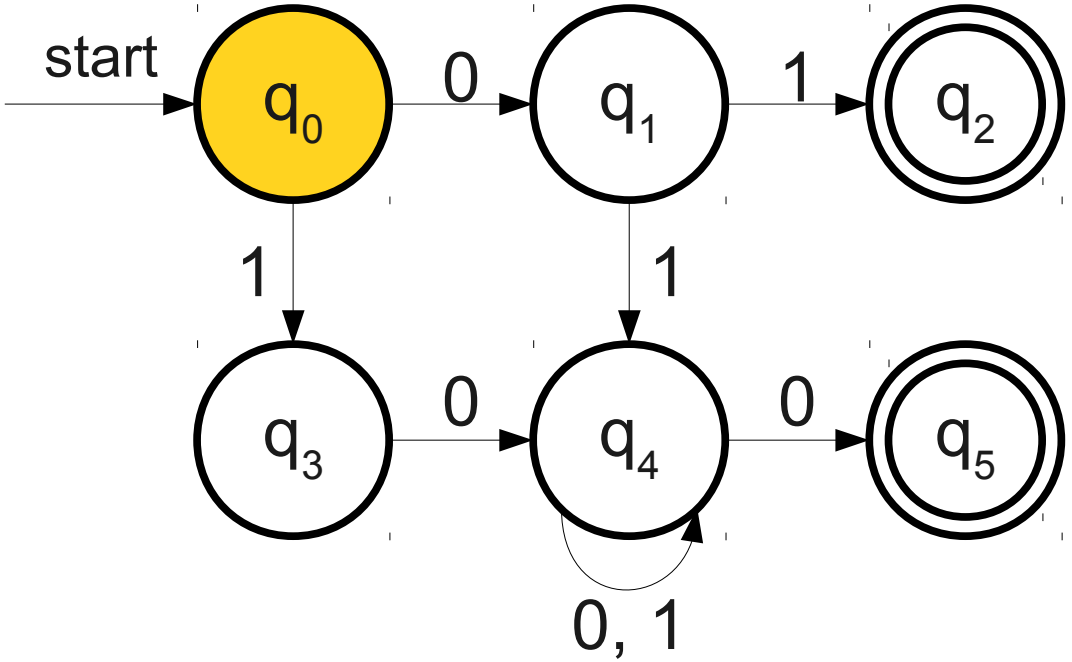
0 1 0 1 0

Perfect Guessing

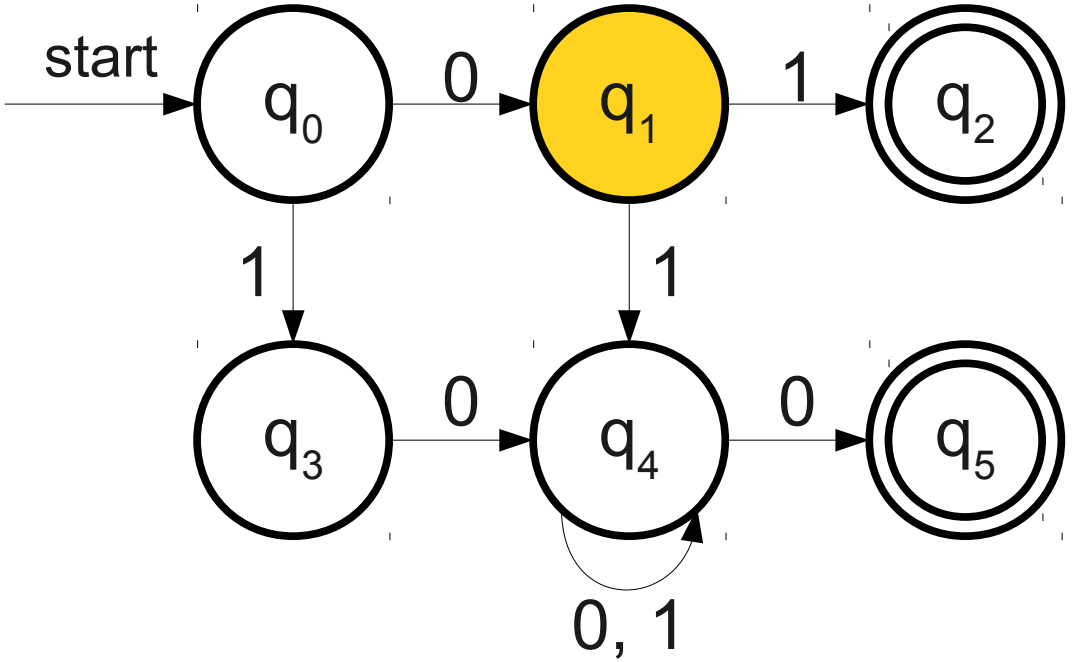


0 1 0 1 0

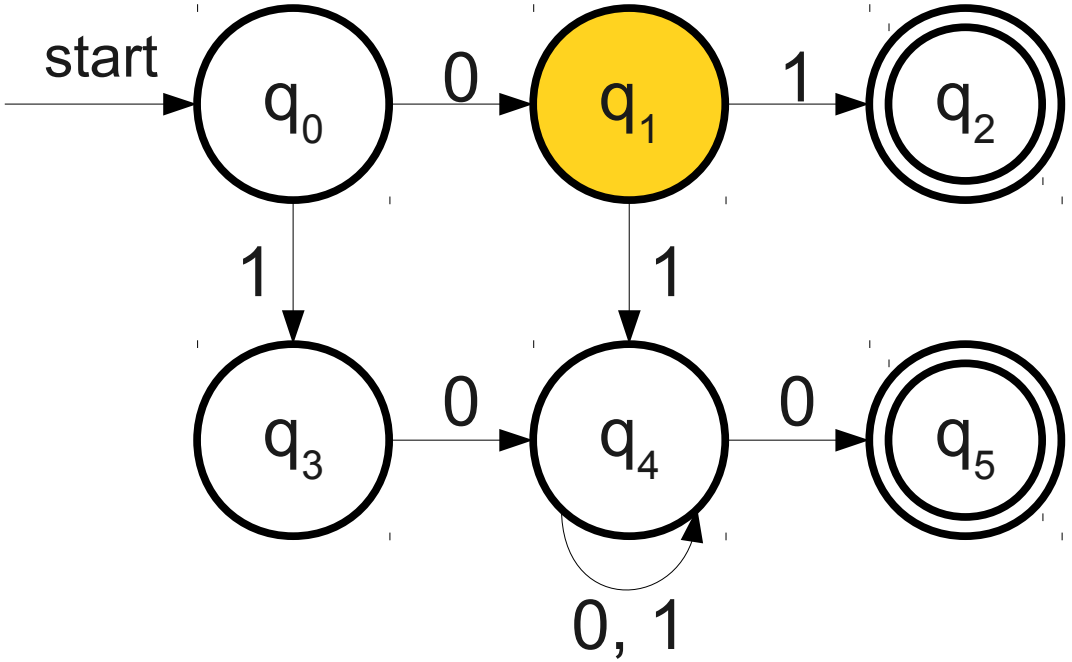
Perfect Guessing

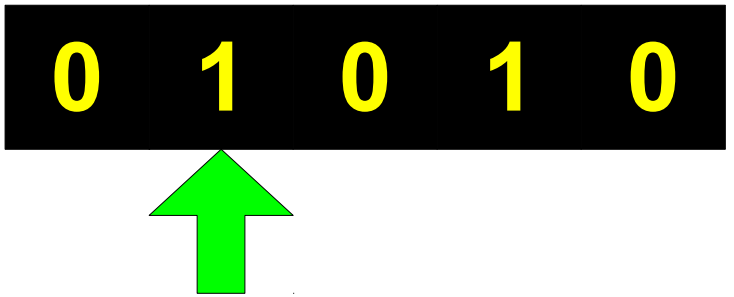


Perfect Guessing

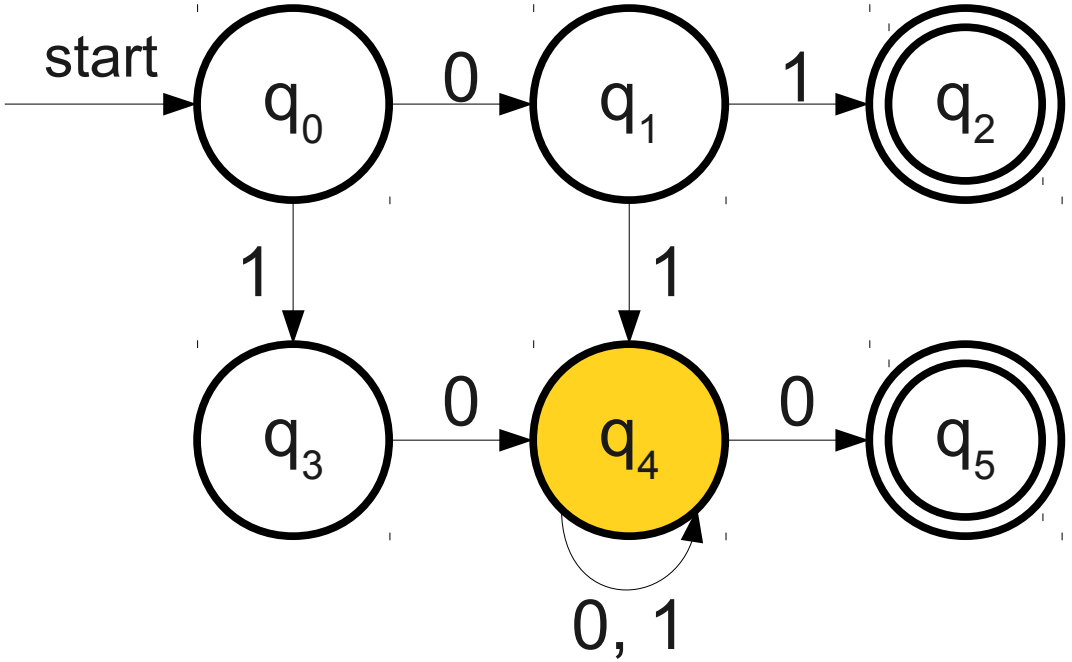


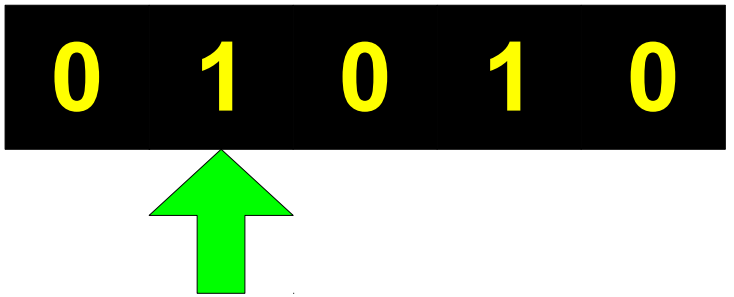
Perfect Guessing



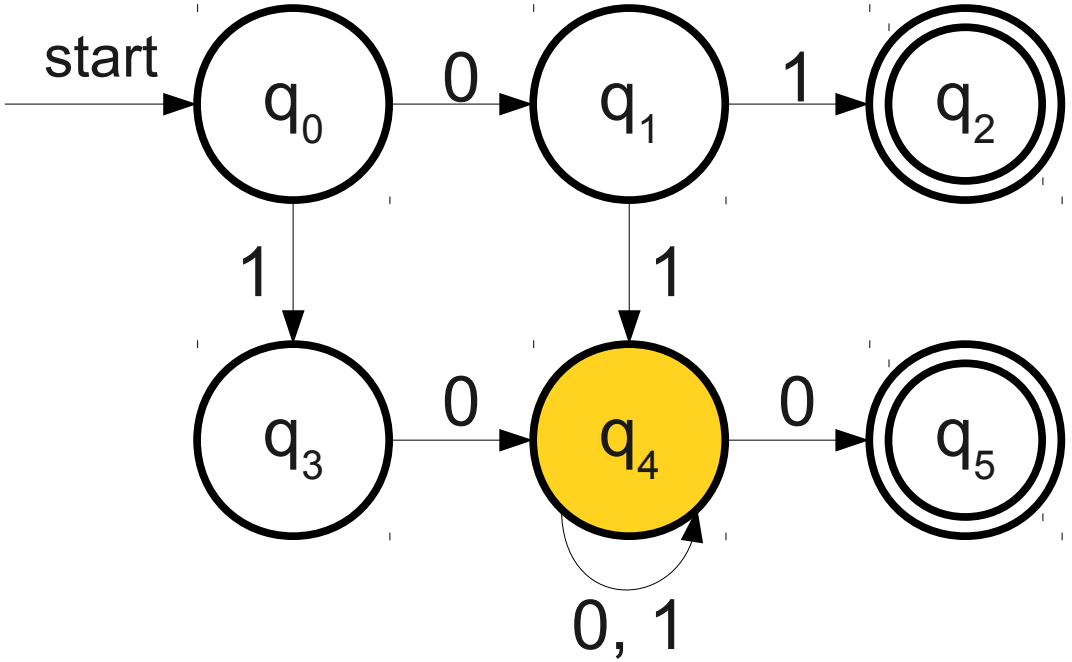


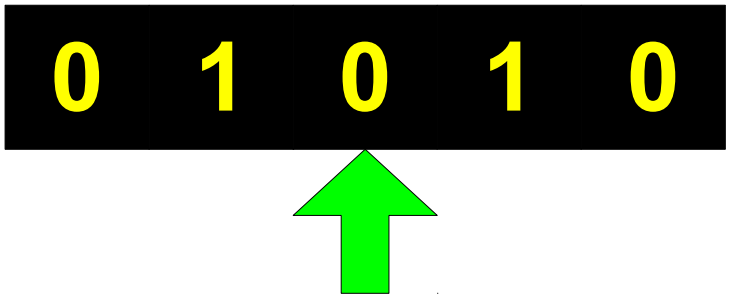
Perfect Guessing



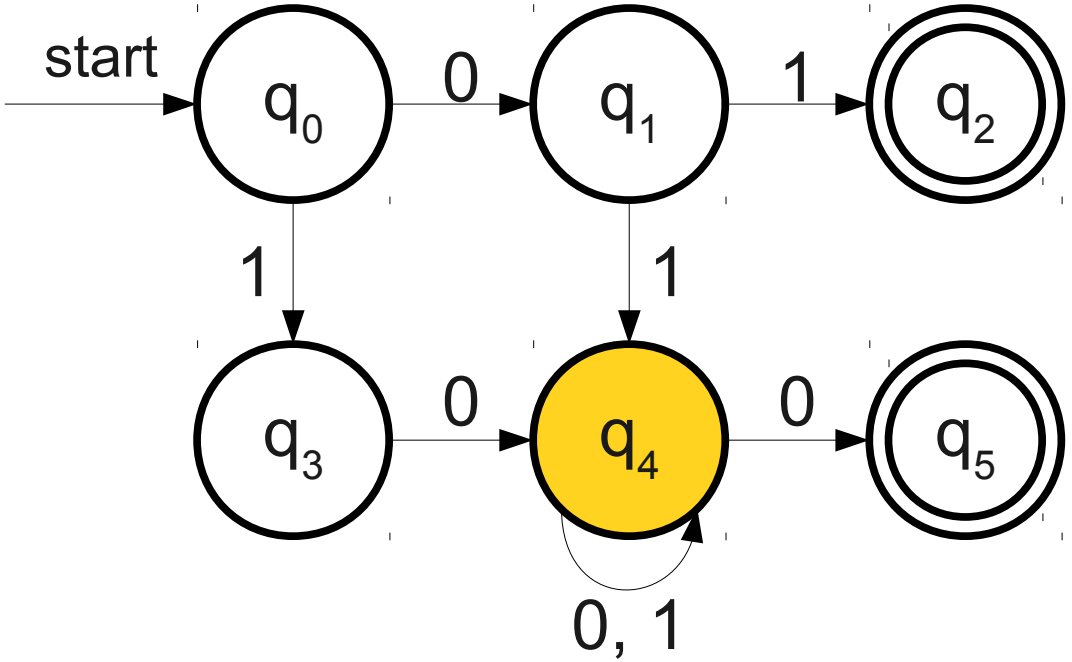


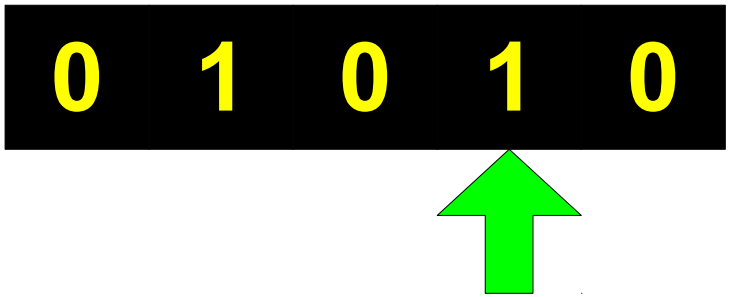
Perfect Guessing



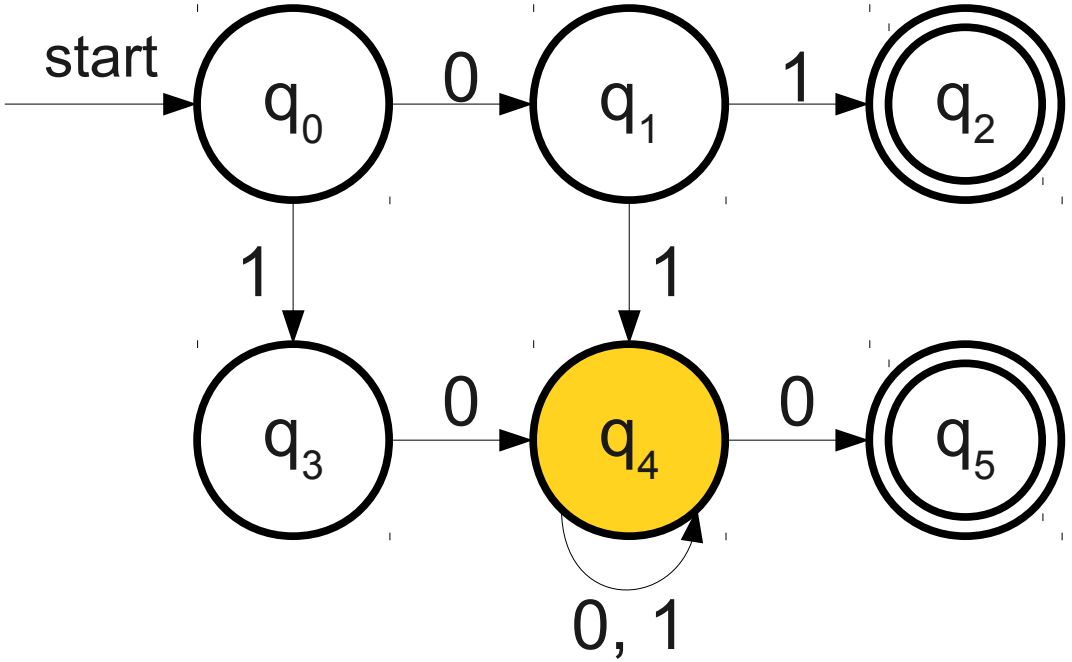


Perfect Guessing

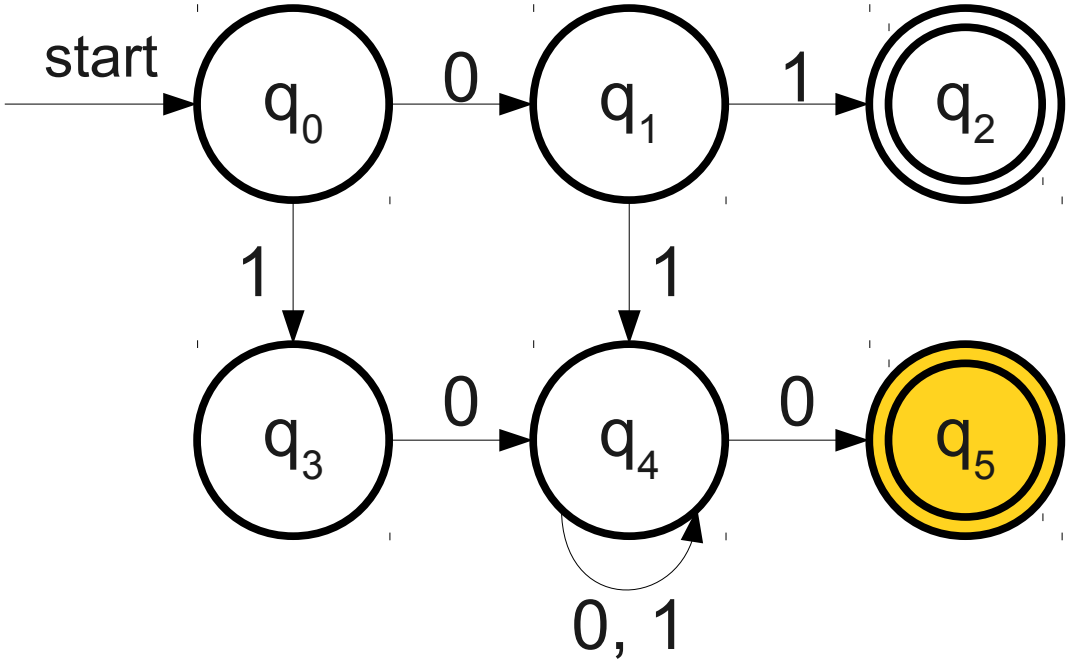




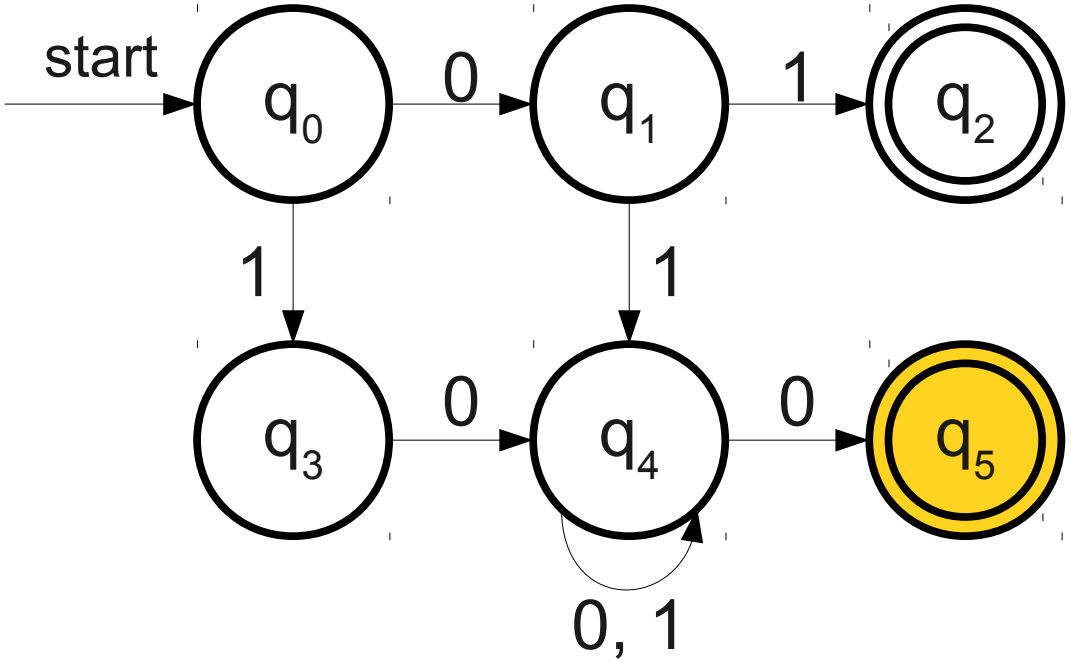
Perfect Guessing



Perfect Guessing

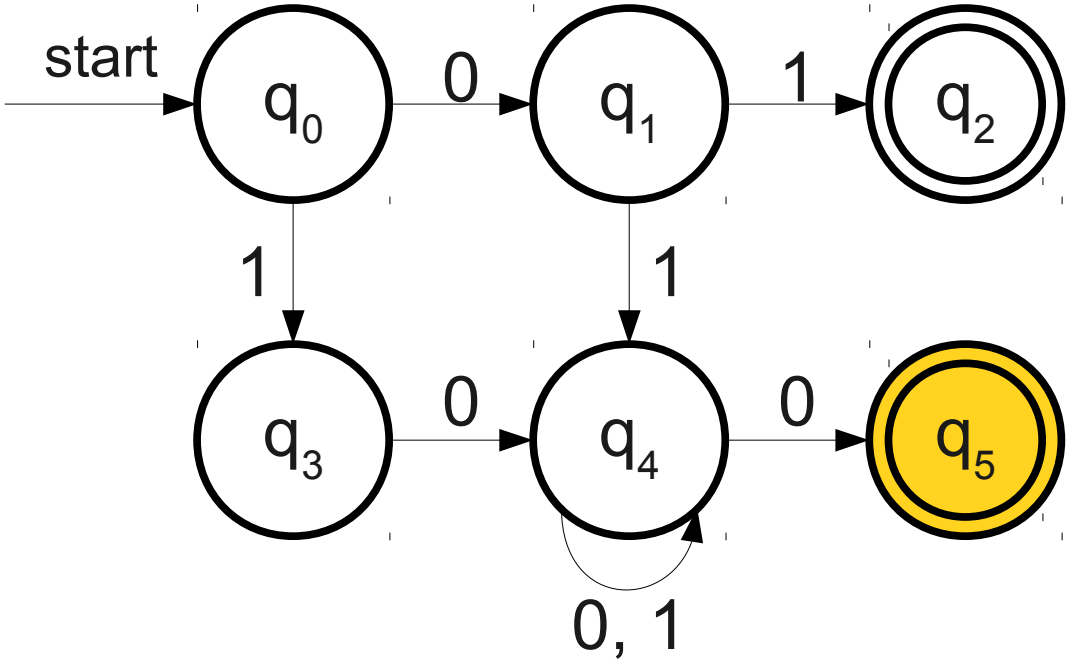


Perfect Guessing



0 1 0 1 0

Perfect Guessing

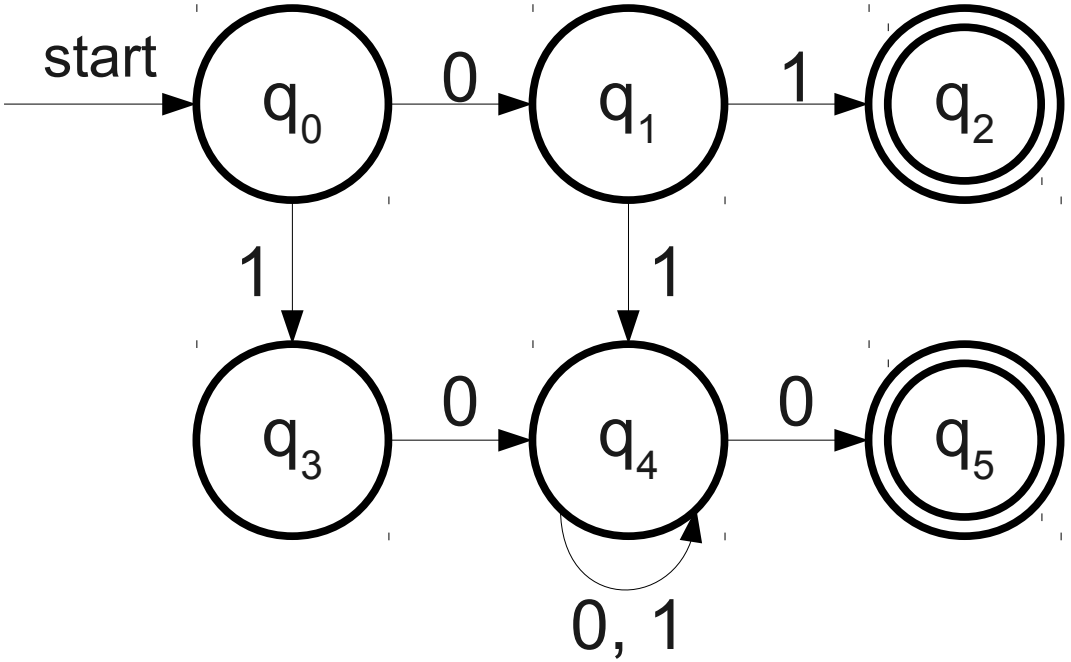


0 1 0 1 0

Perfect Guessing

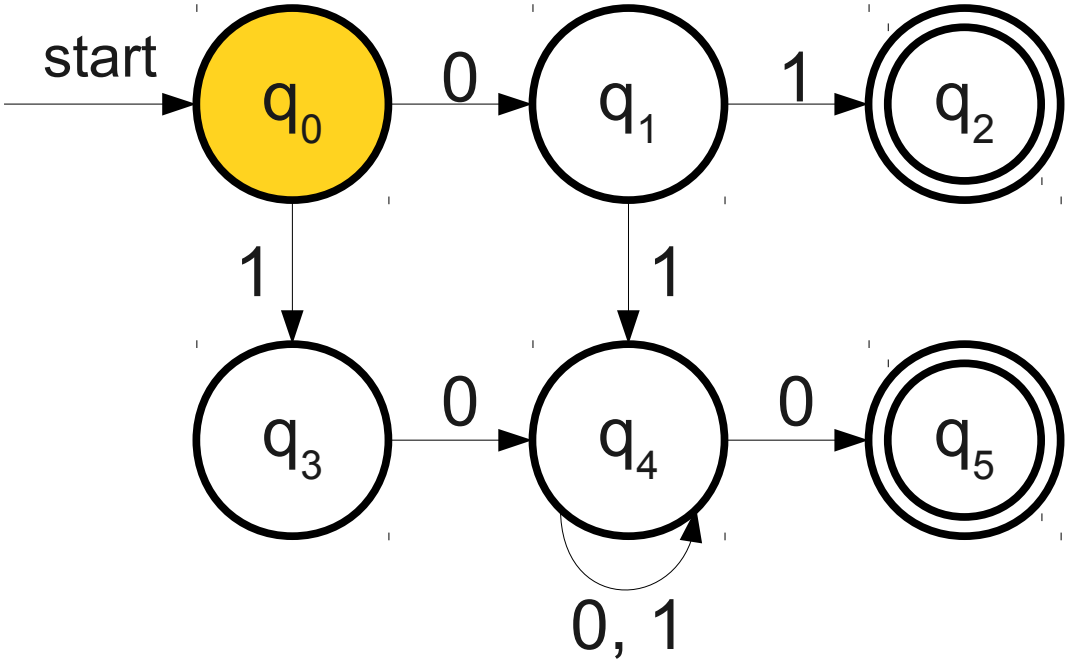
- We can view nondeterministic machines as having Magic Superpowers that enable them to guess the correct choice of moves to make.
- Idea: Machine can always guess the right choice if one exists.
- No physical analog for something of this sort.
  - (Those of you thinking quantum computing – we'll talk about that a bit later. The short version: nondeterminism is more powerful than quantum computation.)

Massive Parallelism



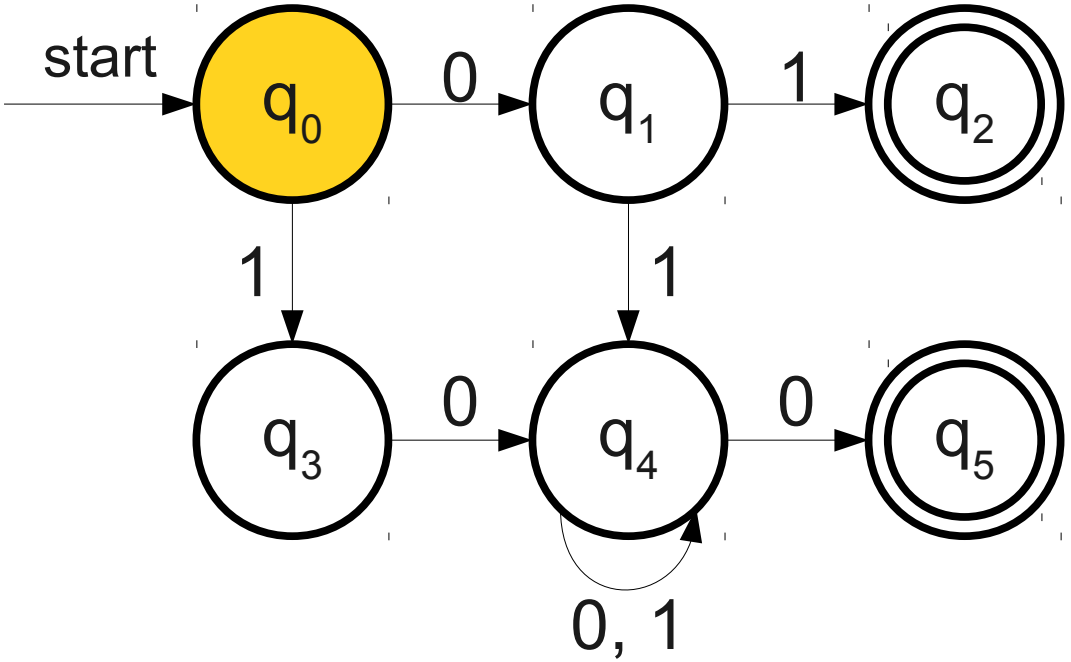
0 1 0 1 0

Massive Parallelism

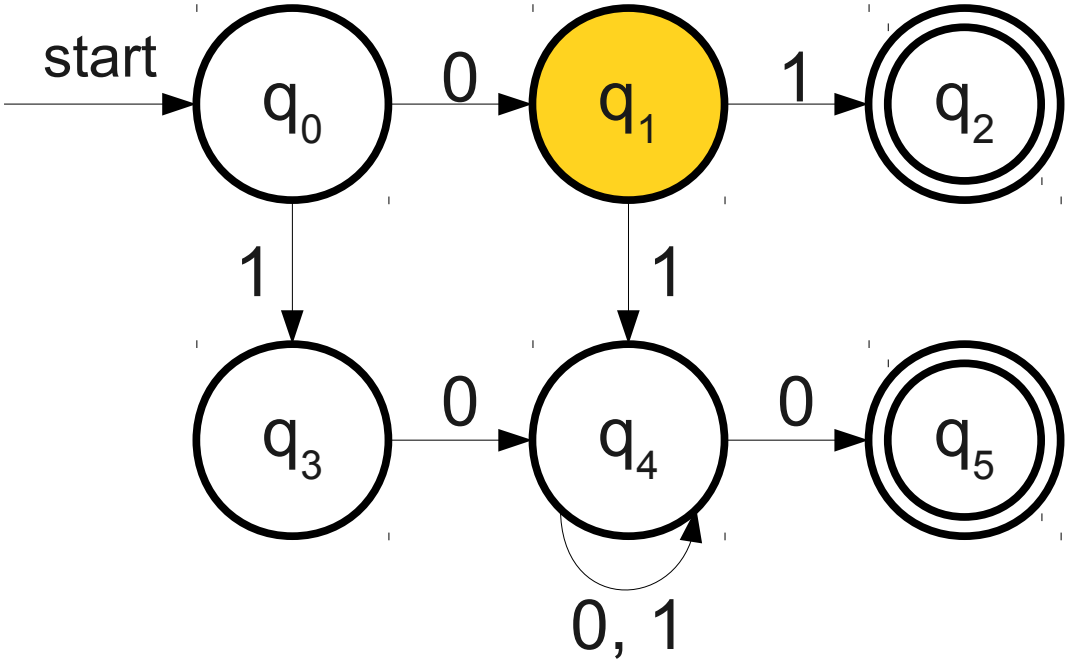


0 1 0 1 0

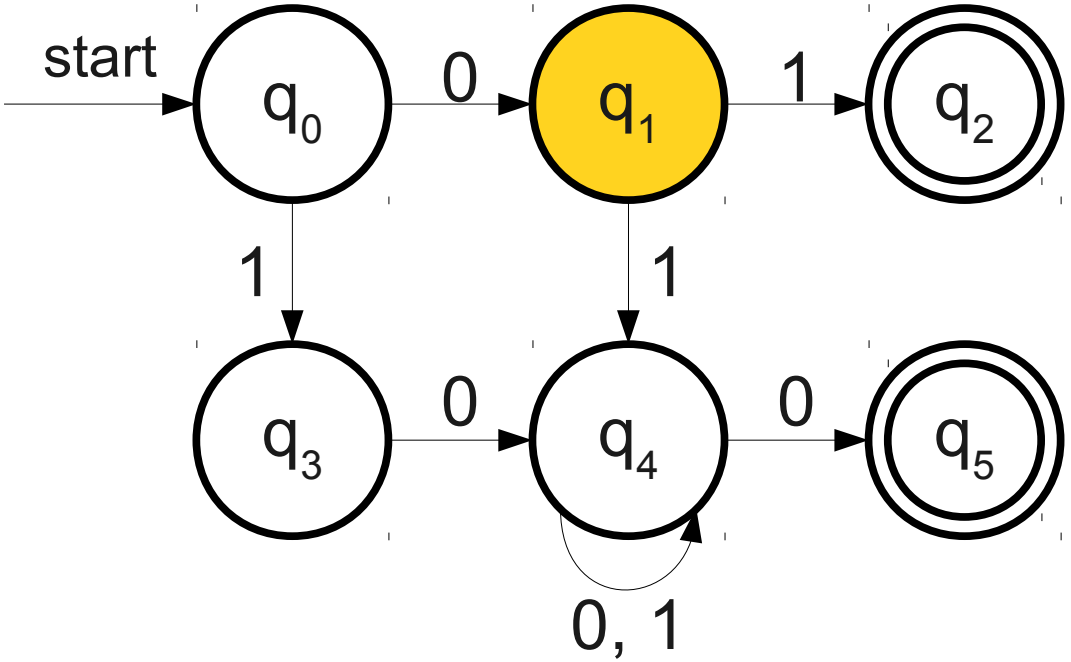
Massive Parallelism

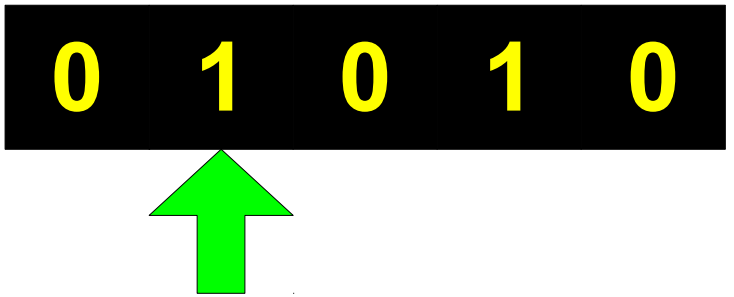


Massive Parallelism

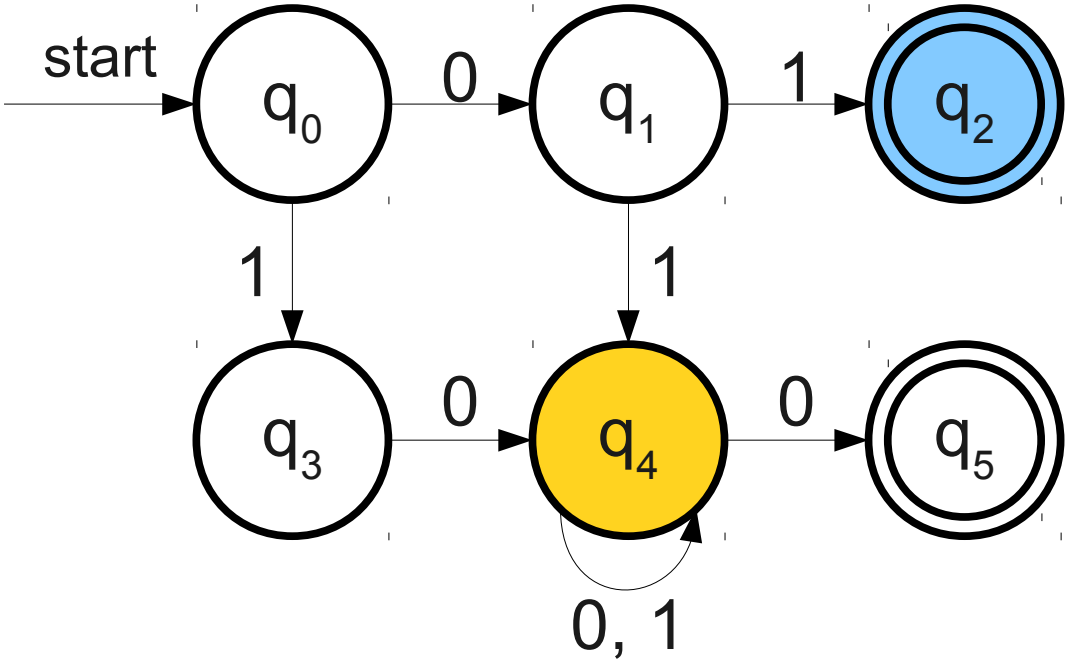


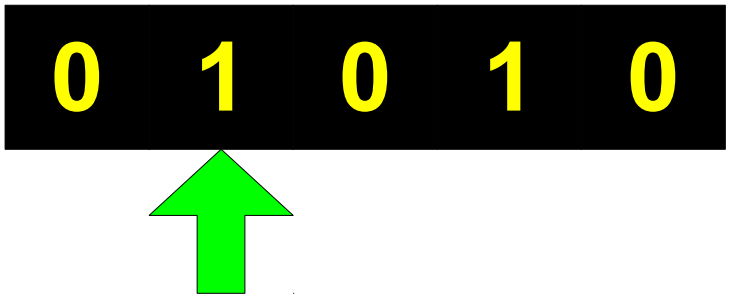
Massive Parallelism



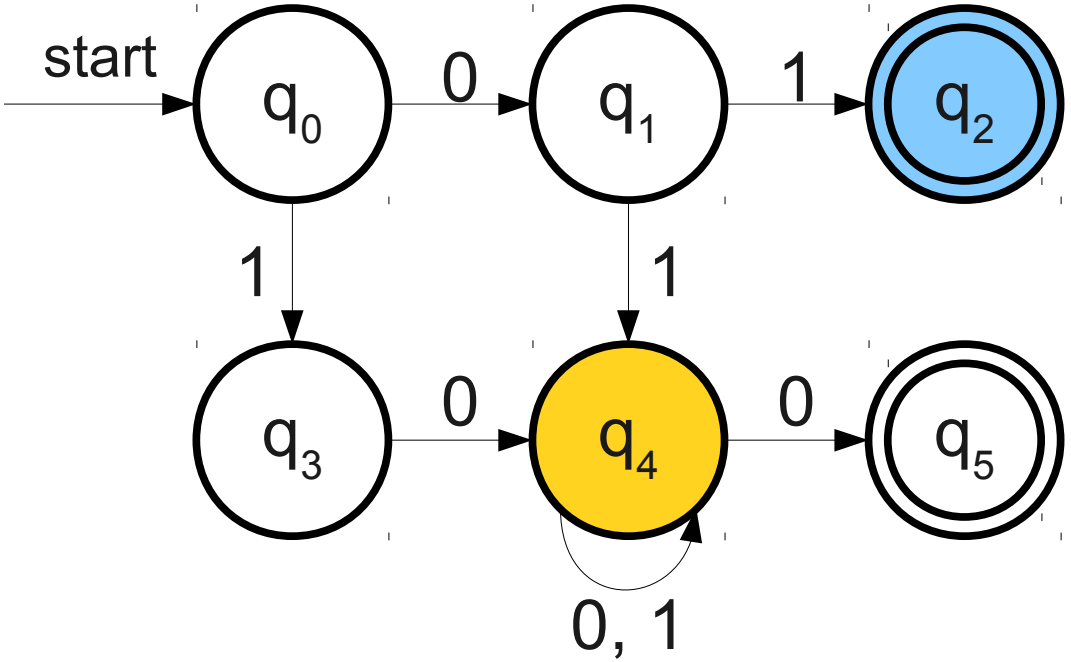


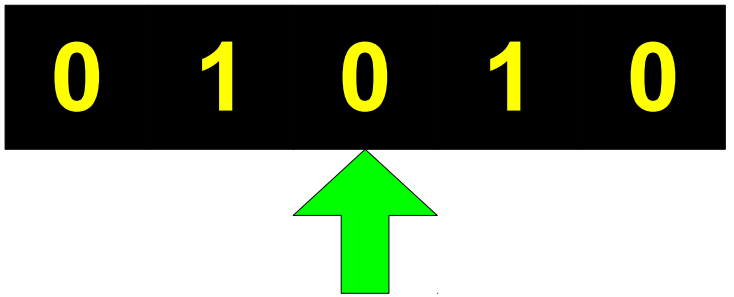
Massive Parallelism



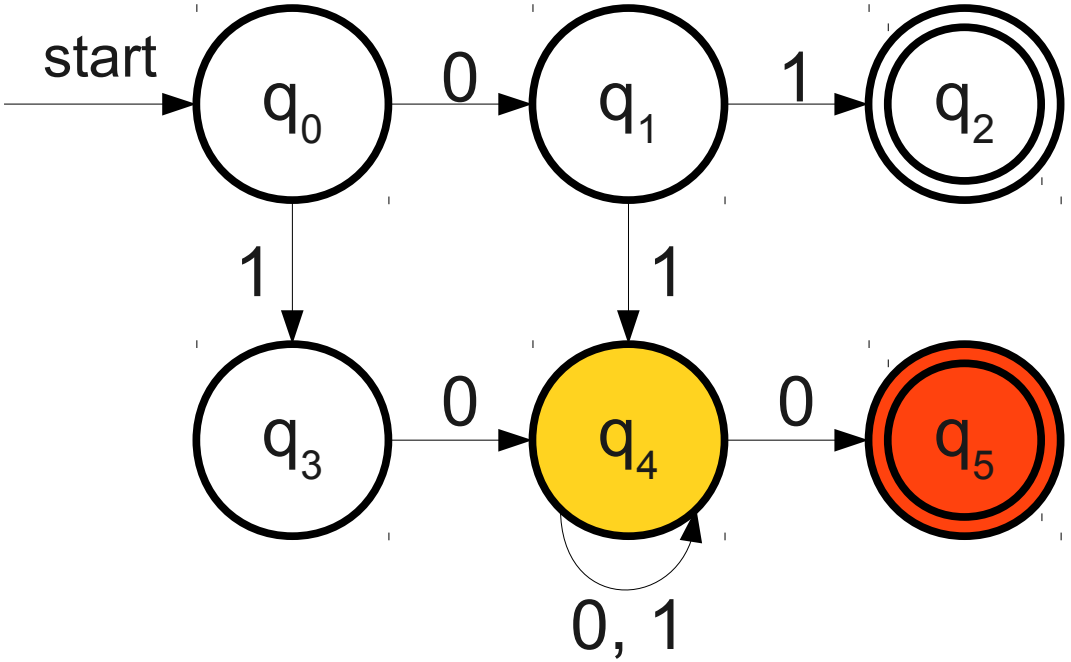


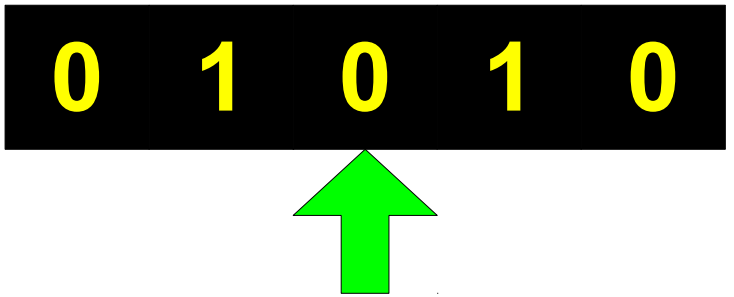
Massive Parallelism



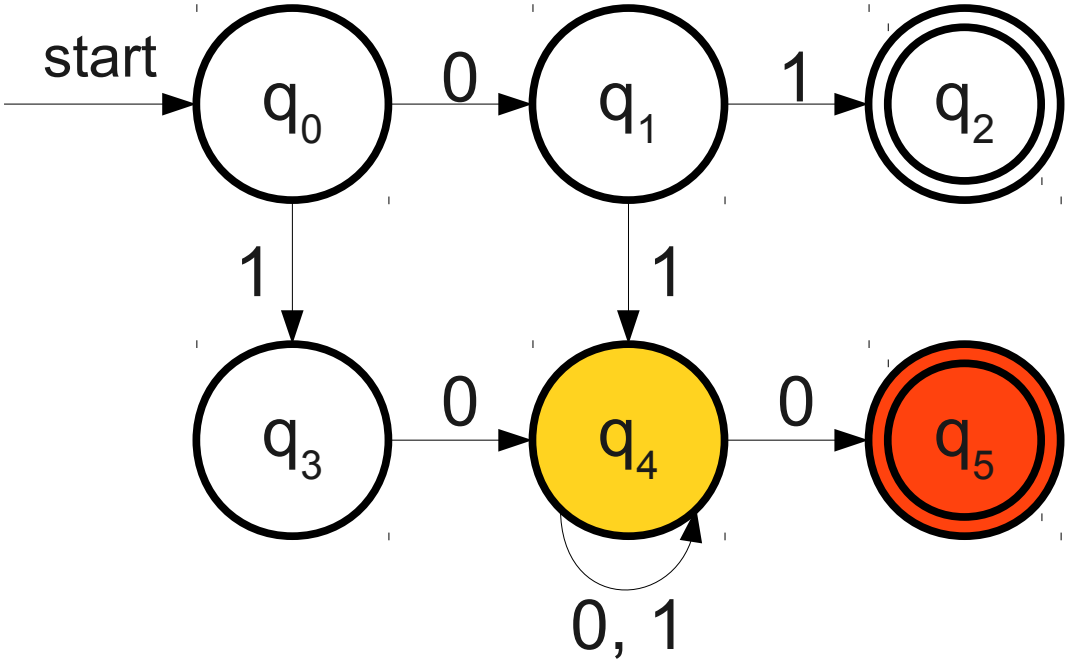


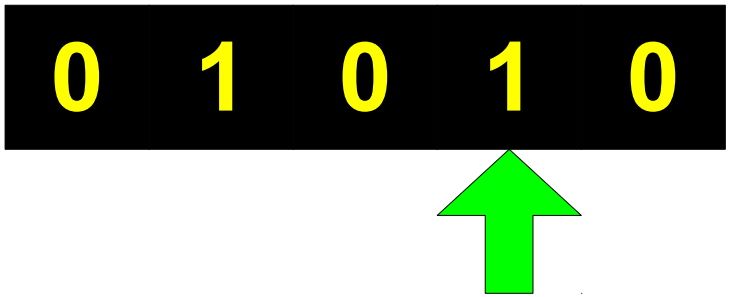
Massive Parallelism



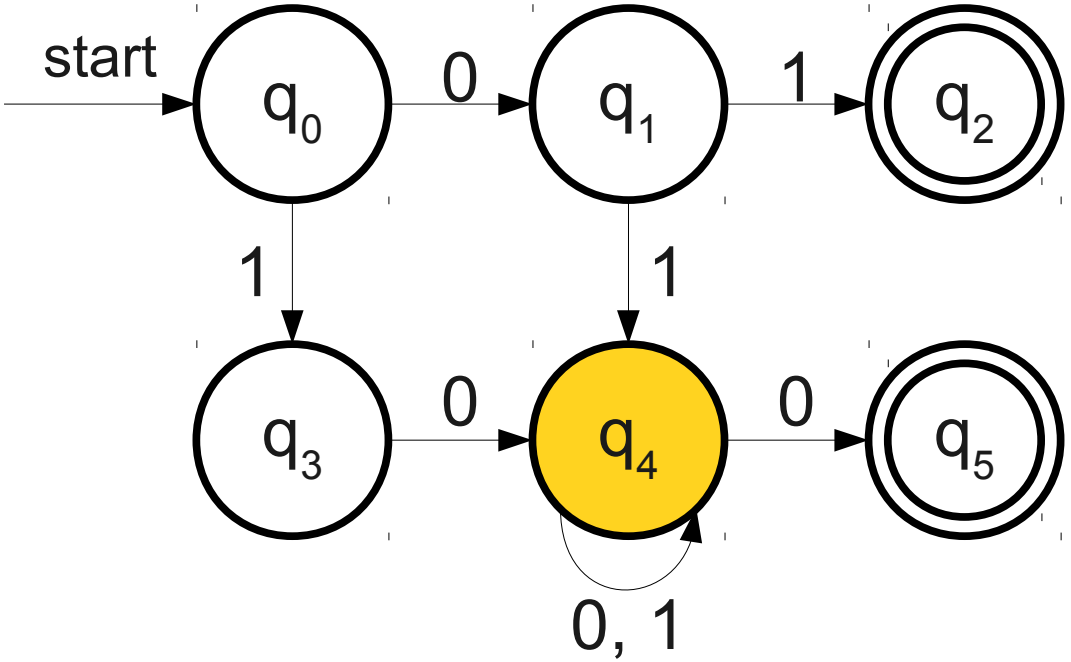


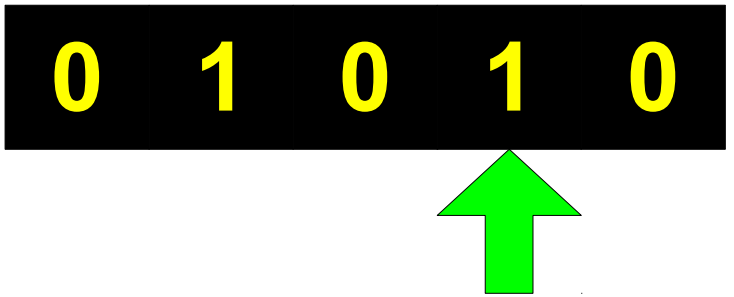
Massive Parallelism



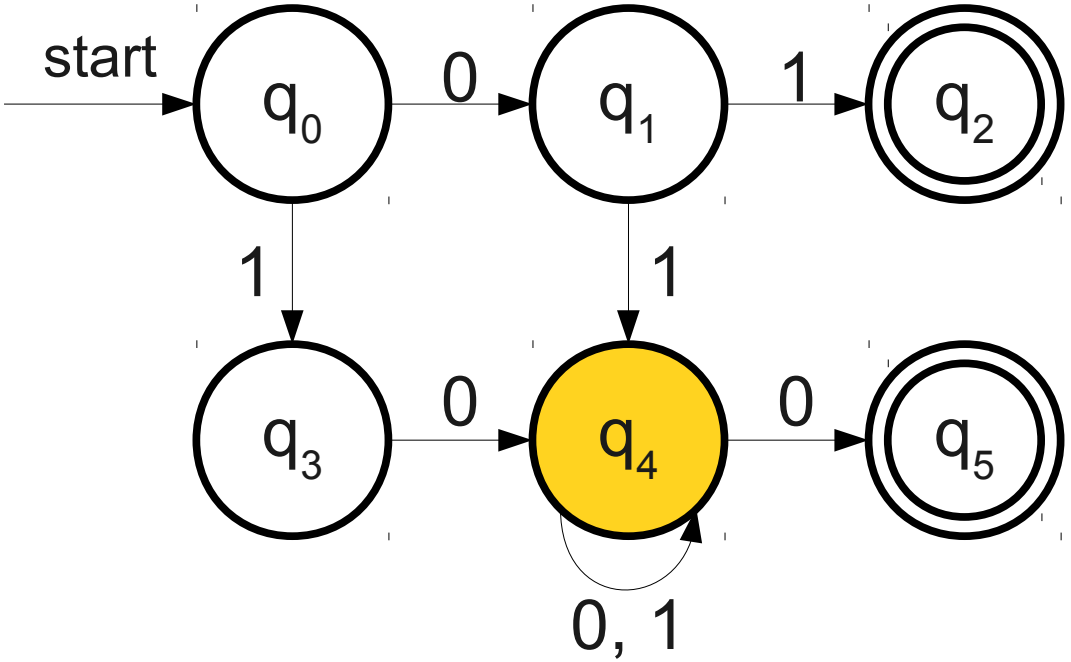


Massive Parallelism

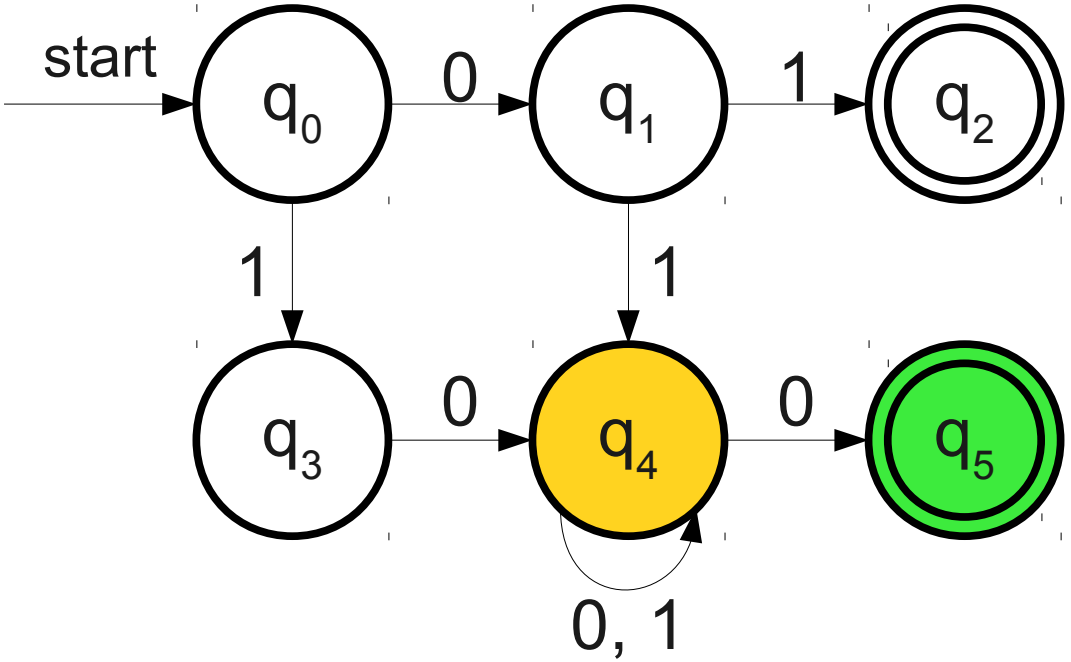




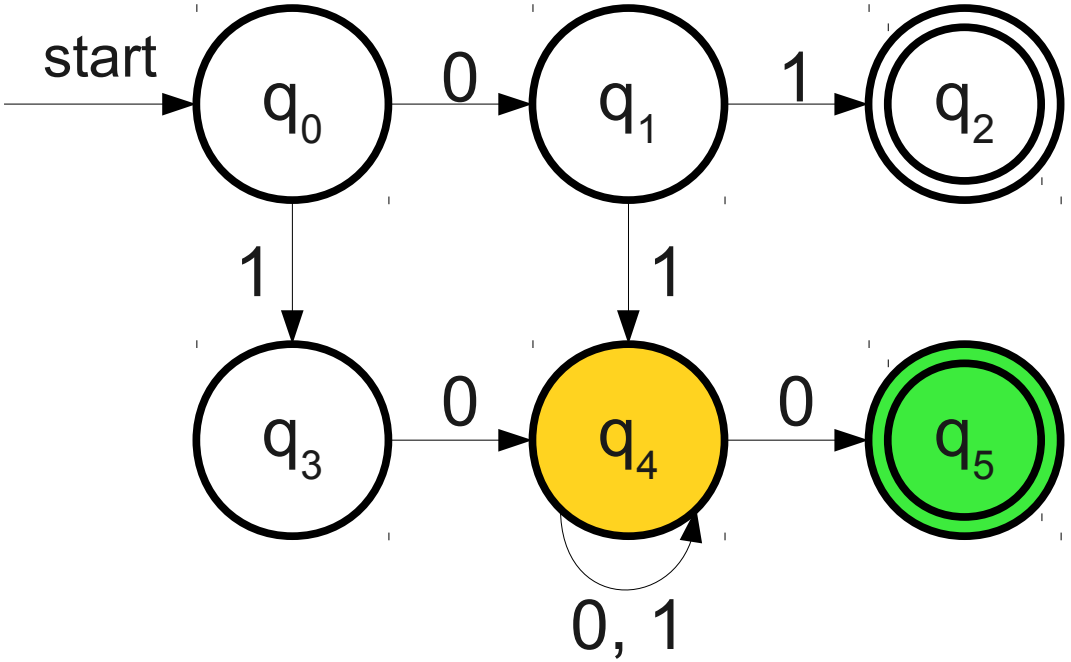
Massive Parallelism



Massive Parallelism

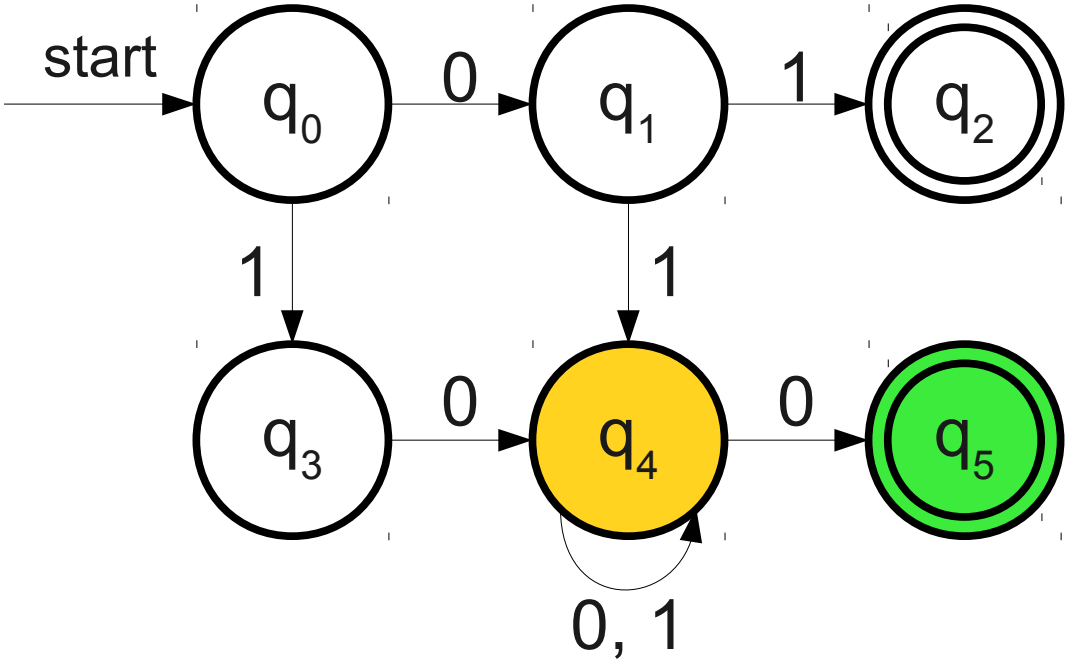


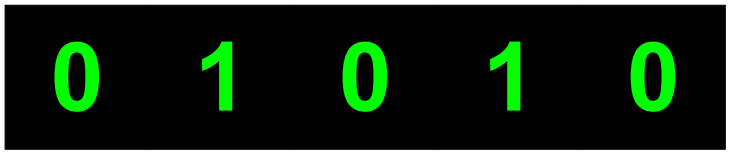
Massive Parallelism



0 1 0 1 0

Massive Parallelism





Massive Parallelism

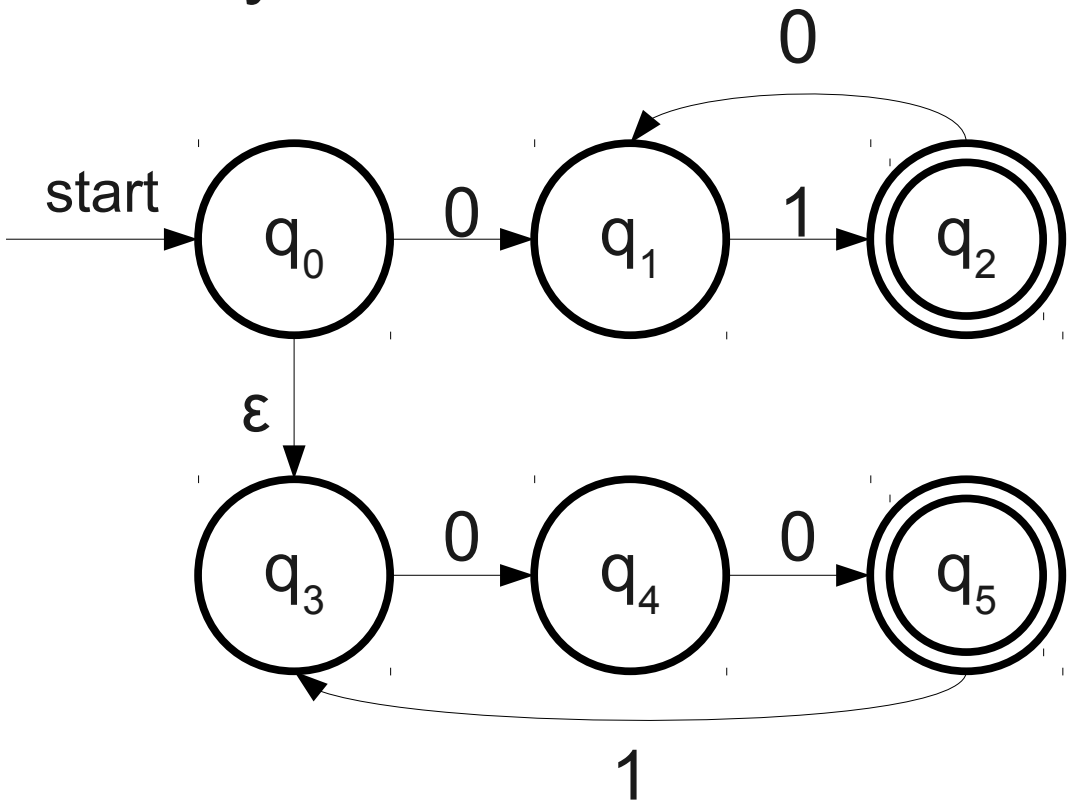
- An NFA can be thought of as a DFA that can be in many states at once.
- Each symbol read causes a transition on every active state into each potential state that could be visited.
- Nondeterministic machines can be thought of as machines that can try any number of options in parallel.
  - No fixed limit on processors; makes multicore machines look downright wimpy!

So What?

- We will turn to these three intuitions for nondeterminism more later in the quarter.
- Nondeterministic machines may not be feasible, but they give a great basis for interesting questions:
  - Can any problem that can be solved by a nondeterministic machine be solved by a deterministic machine?
  - Can any problem that can be solved by a nondeterministic machine be solved **efficiently** by a deterministic machine?
- The answers vary from automaton to automaton.

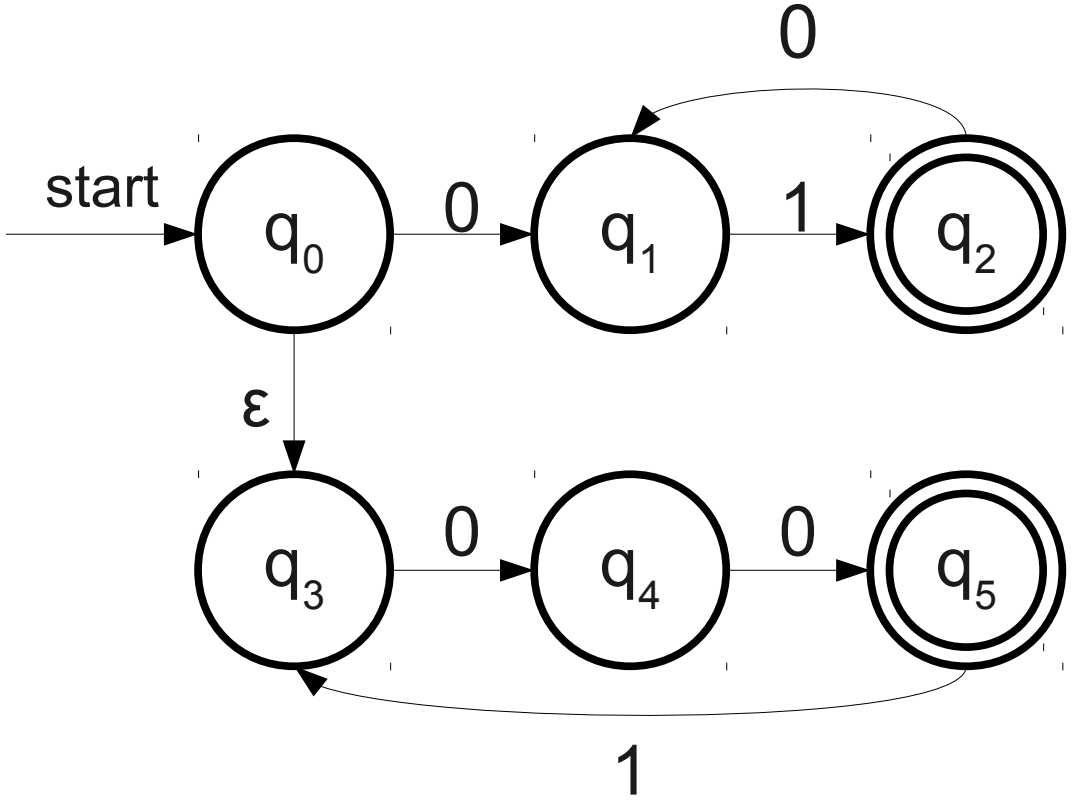
ε-Transitions

- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



ε-Transitions

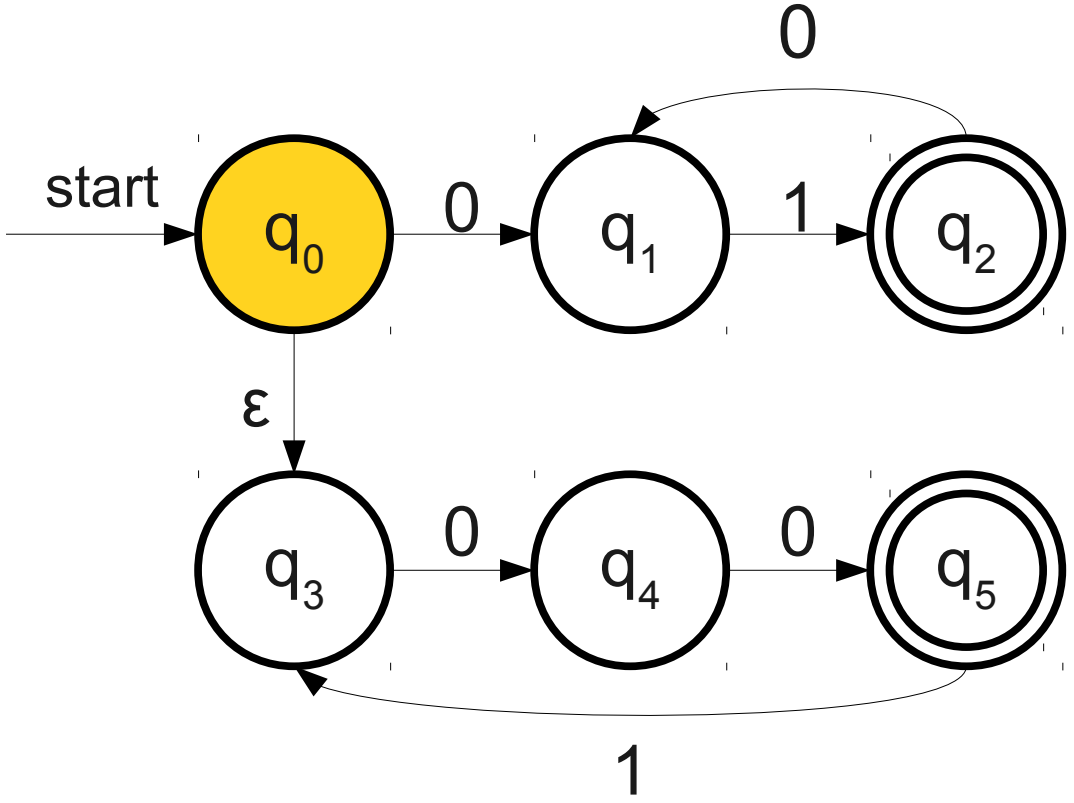
- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



0 0 1 0 0

ε-Transitions

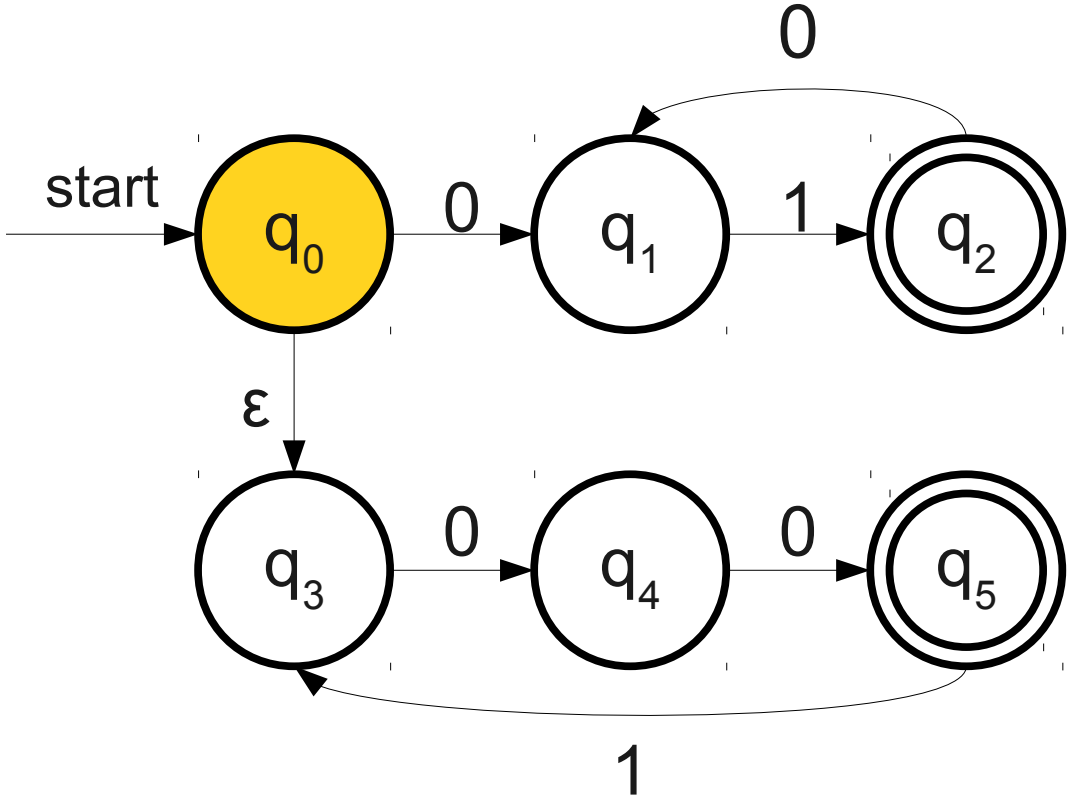
- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



0 0 1 0 0

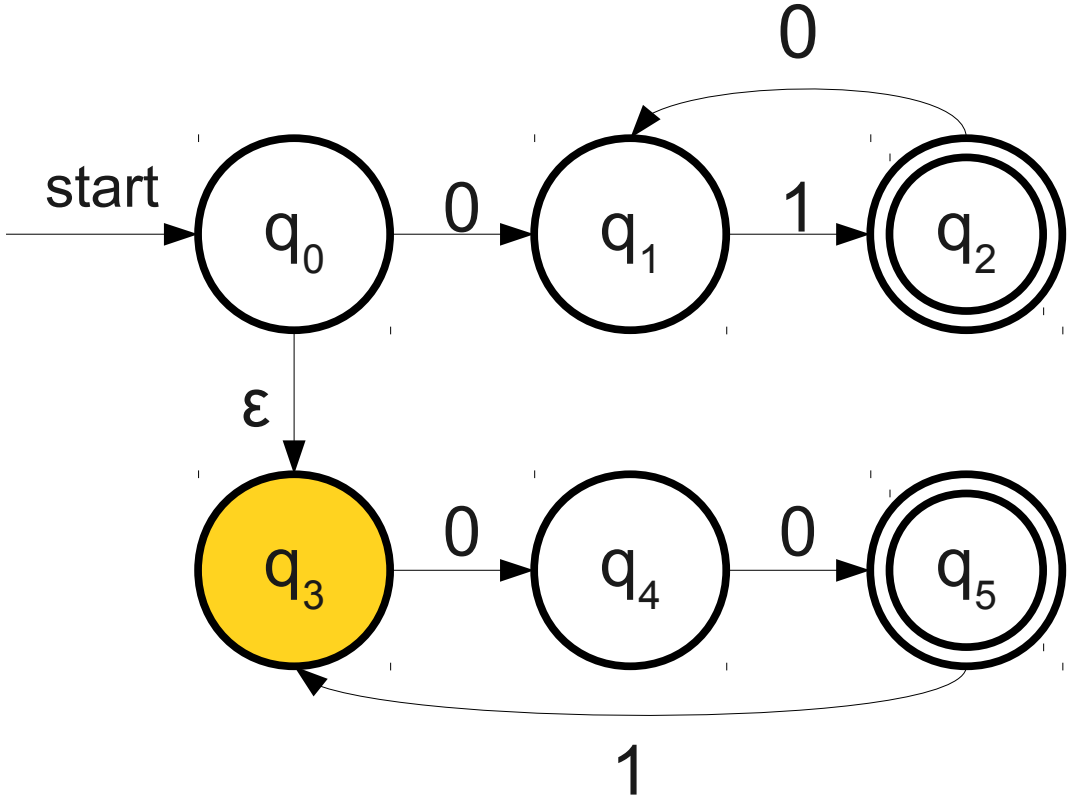
ε-Transitions

- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



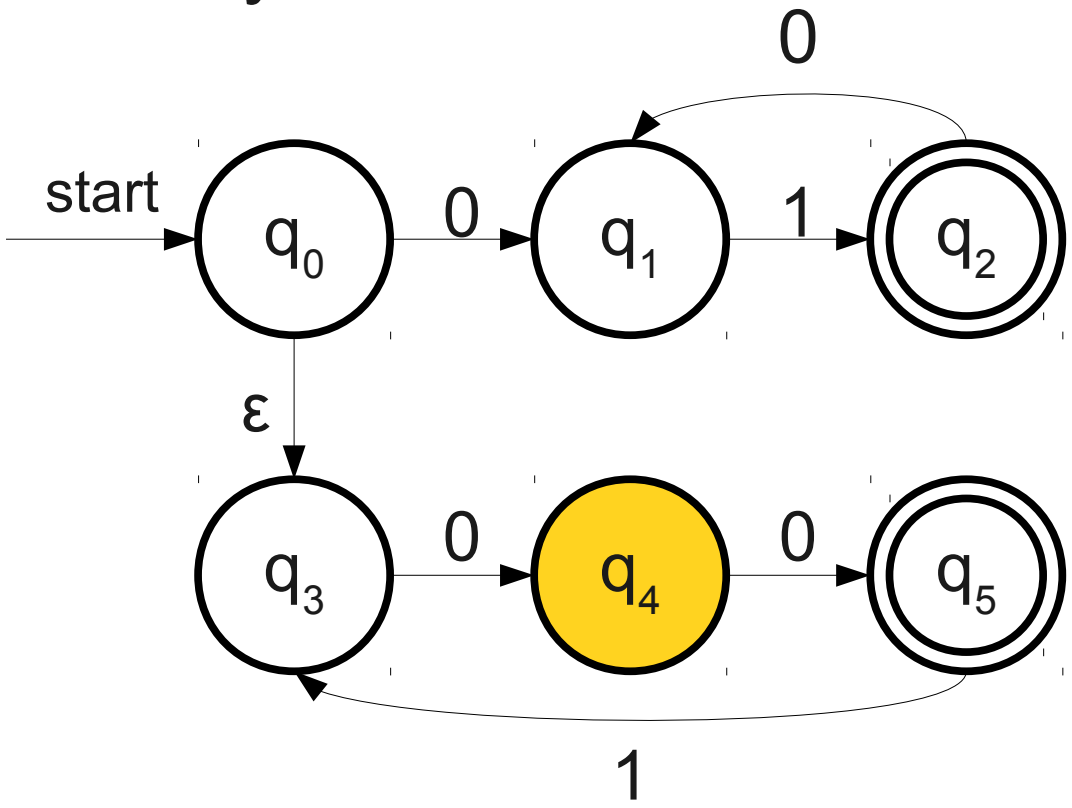
ε-Transitions

- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



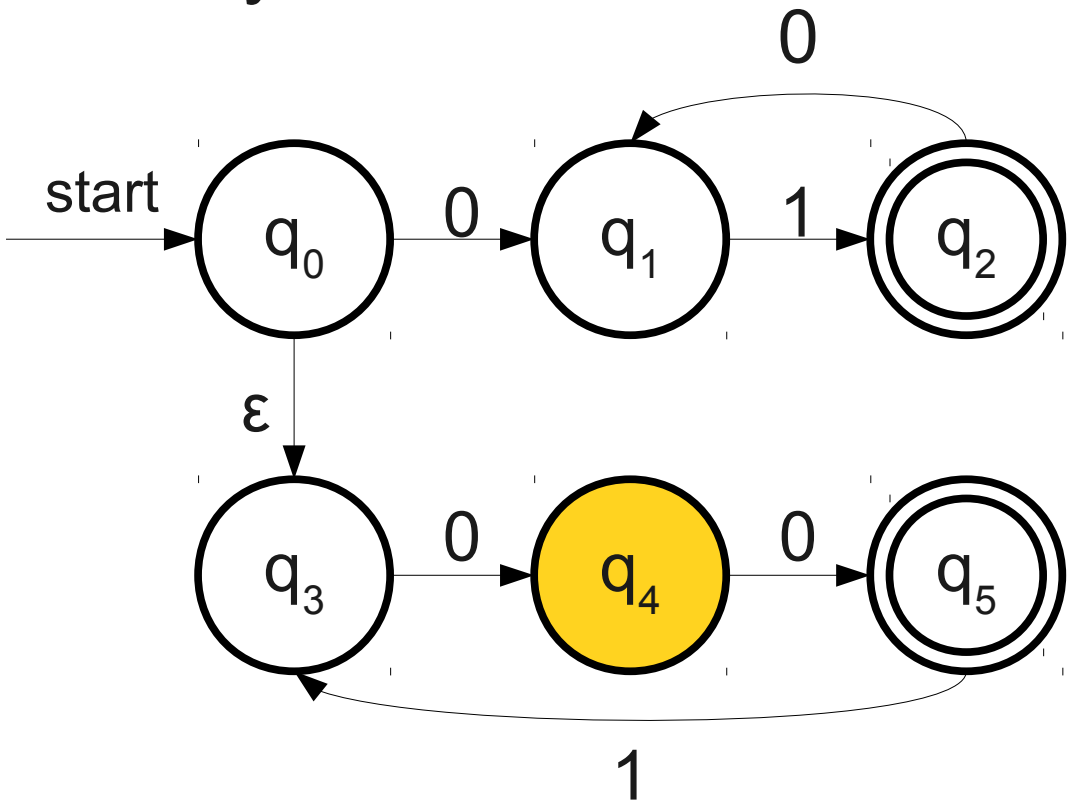
ε-Transitions

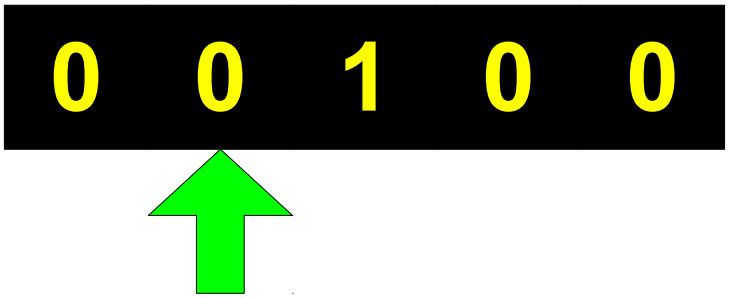
- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



ε-Transitions

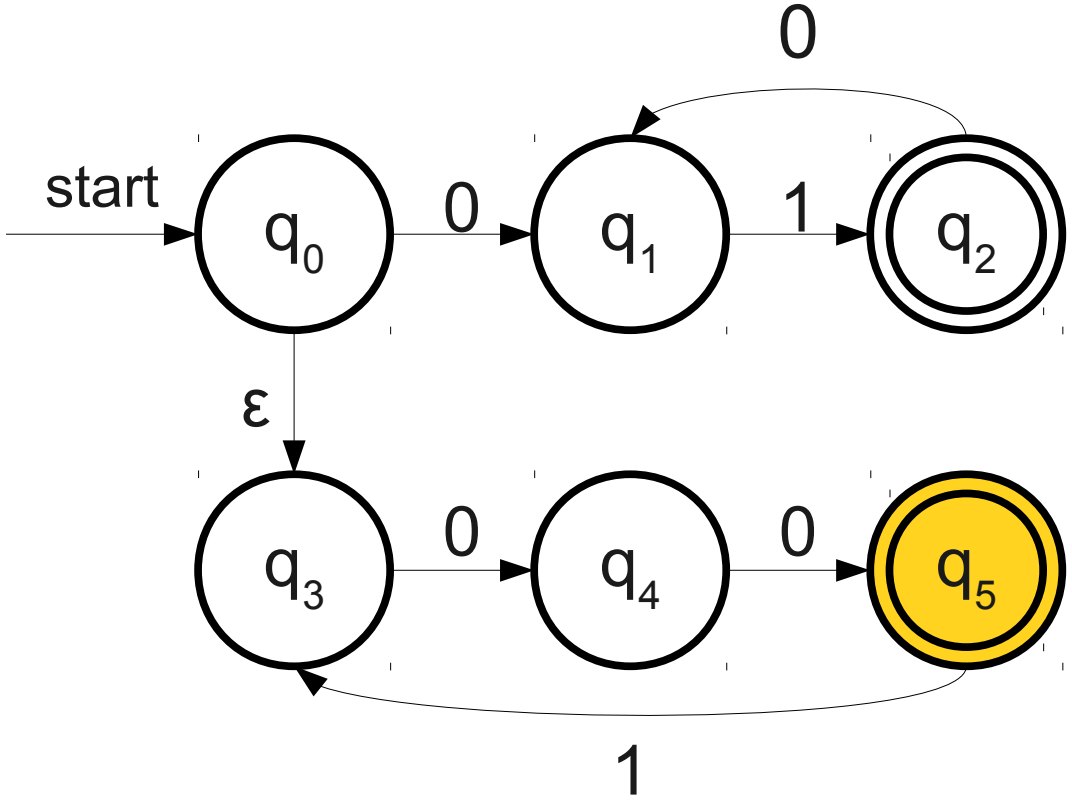
- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

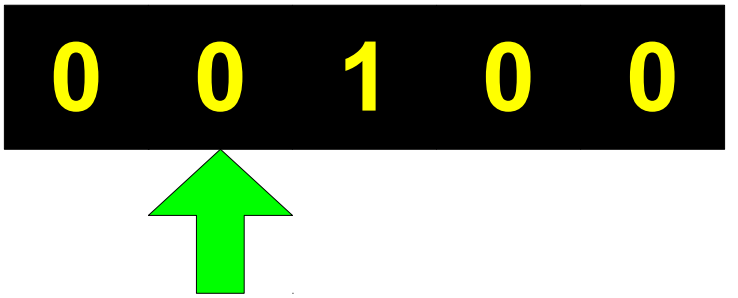




ε-Transitions

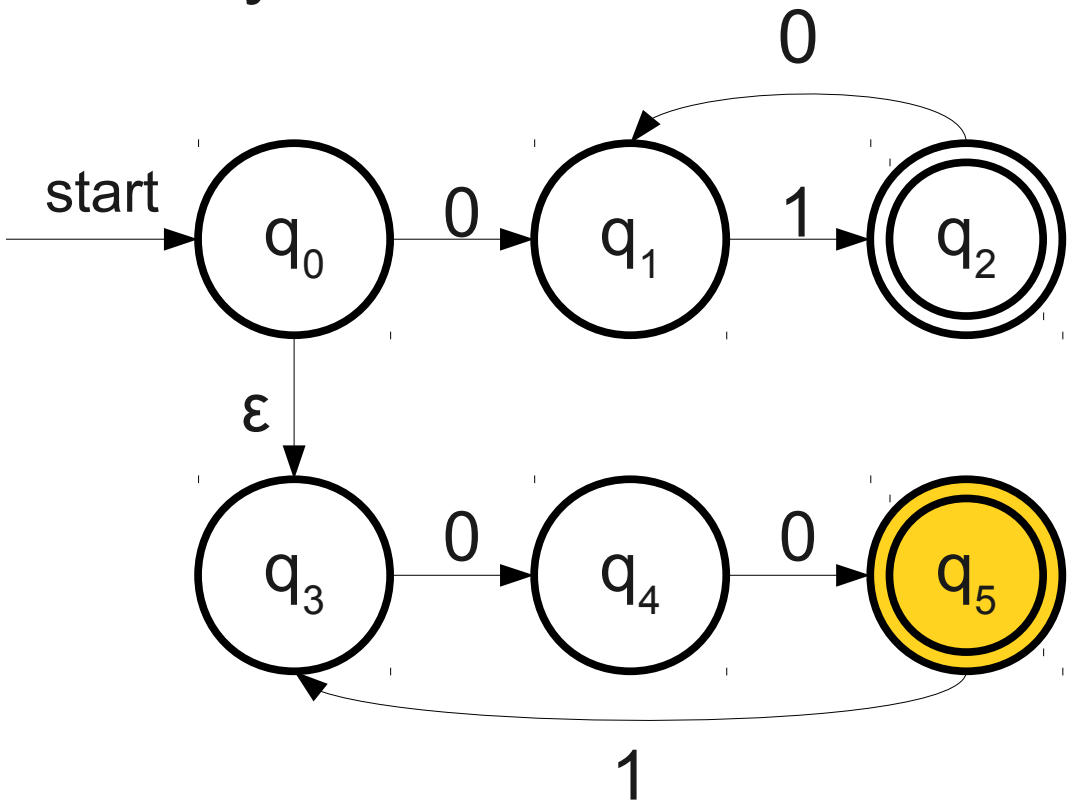
- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

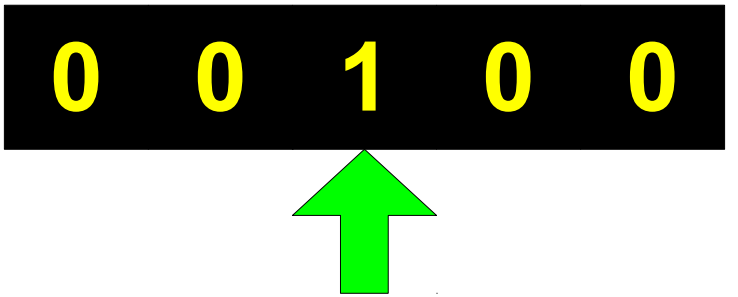




ε-Transitions

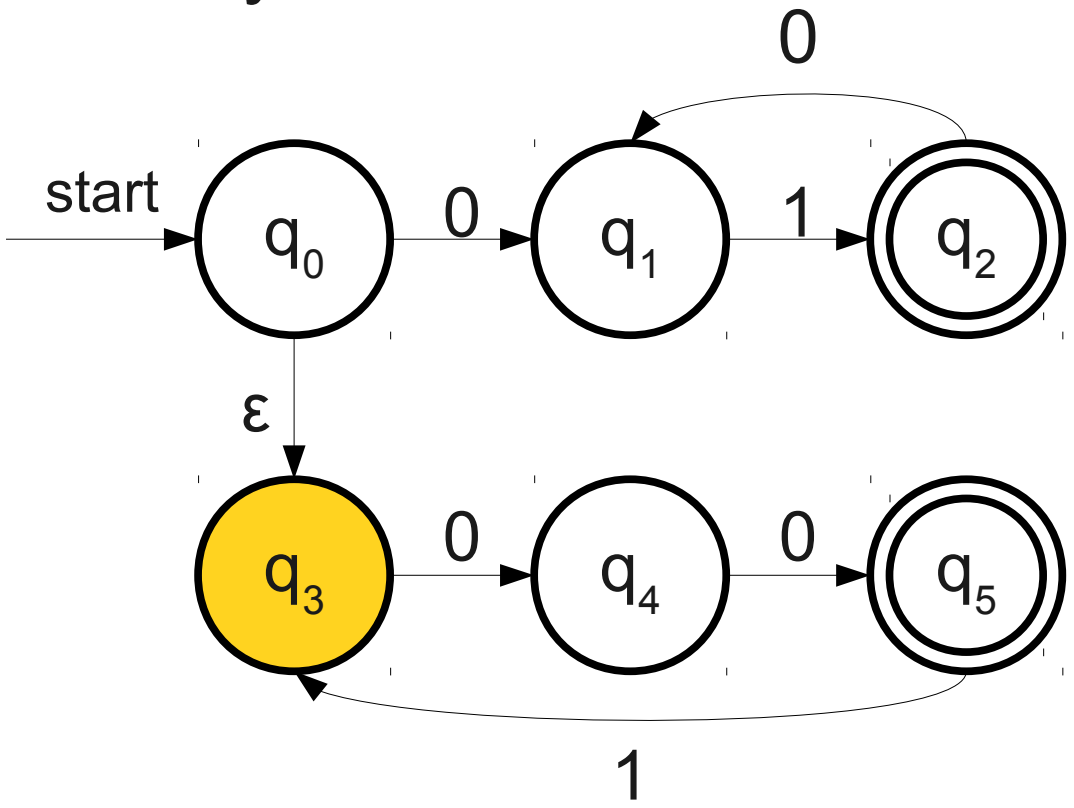
- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

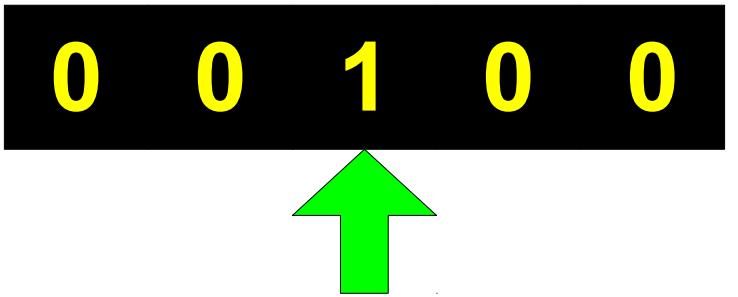




ε-Transitions

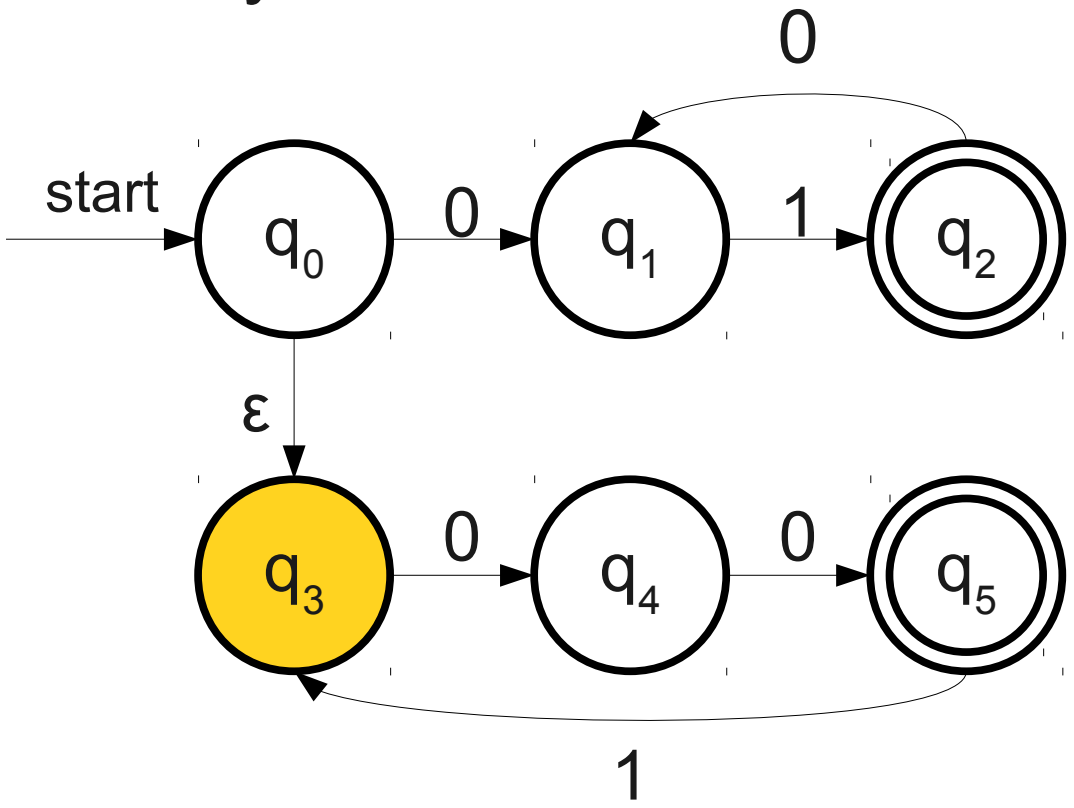
- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

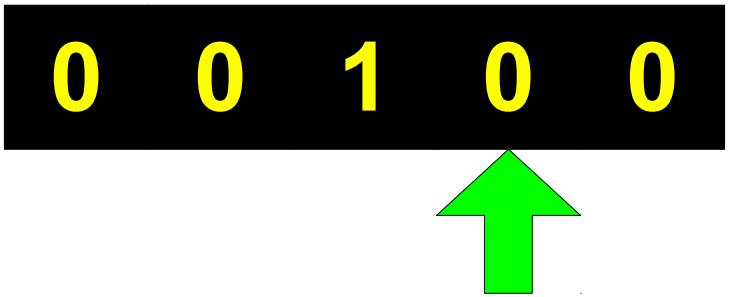




ε-Transitions

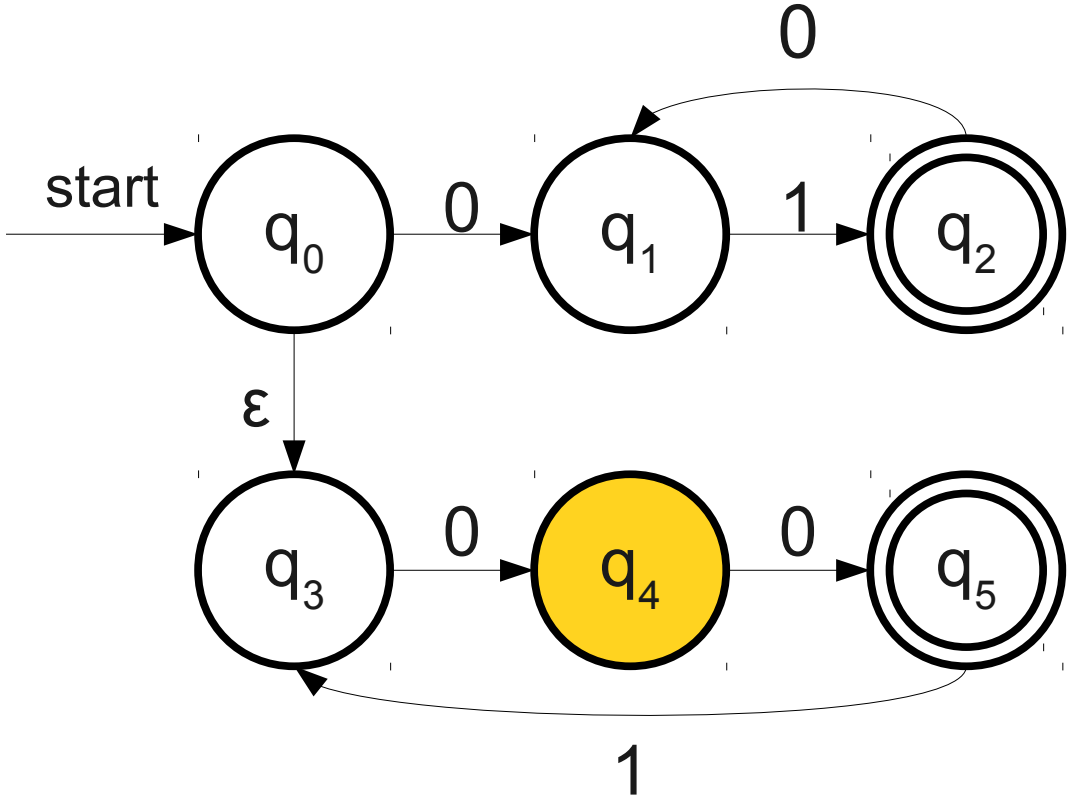
- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

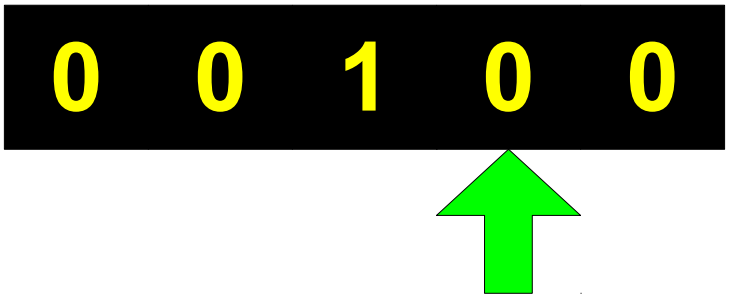




ε-Transitions

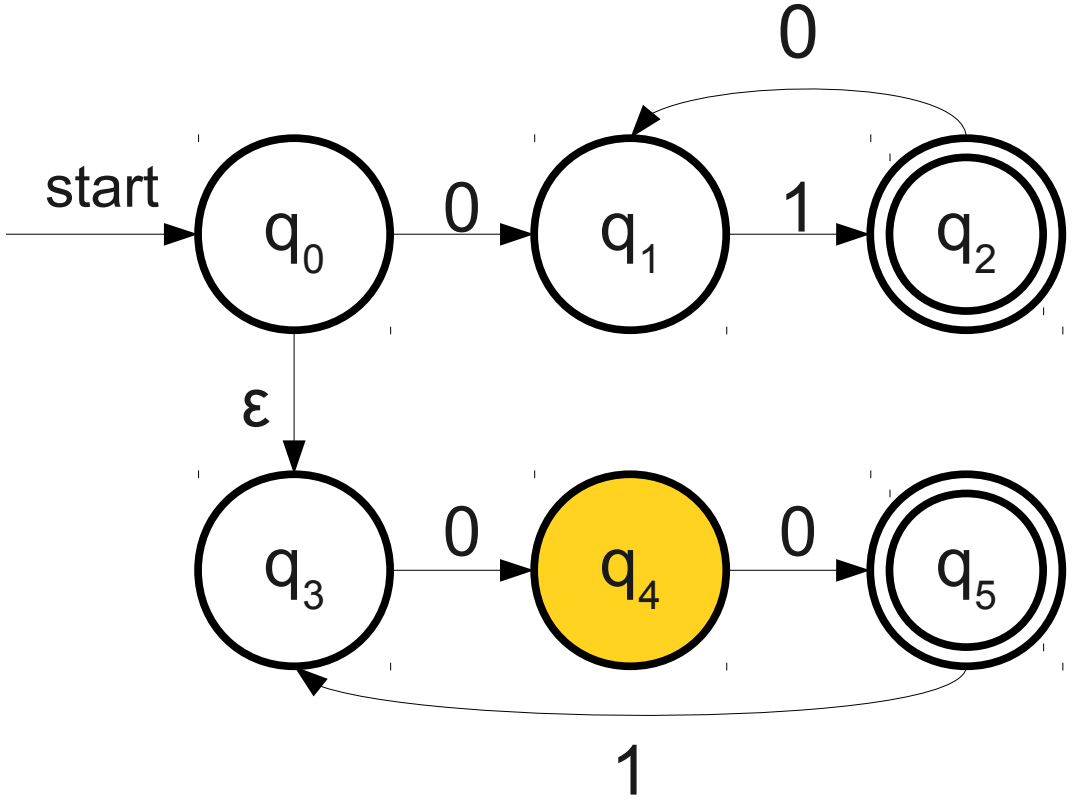
- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.





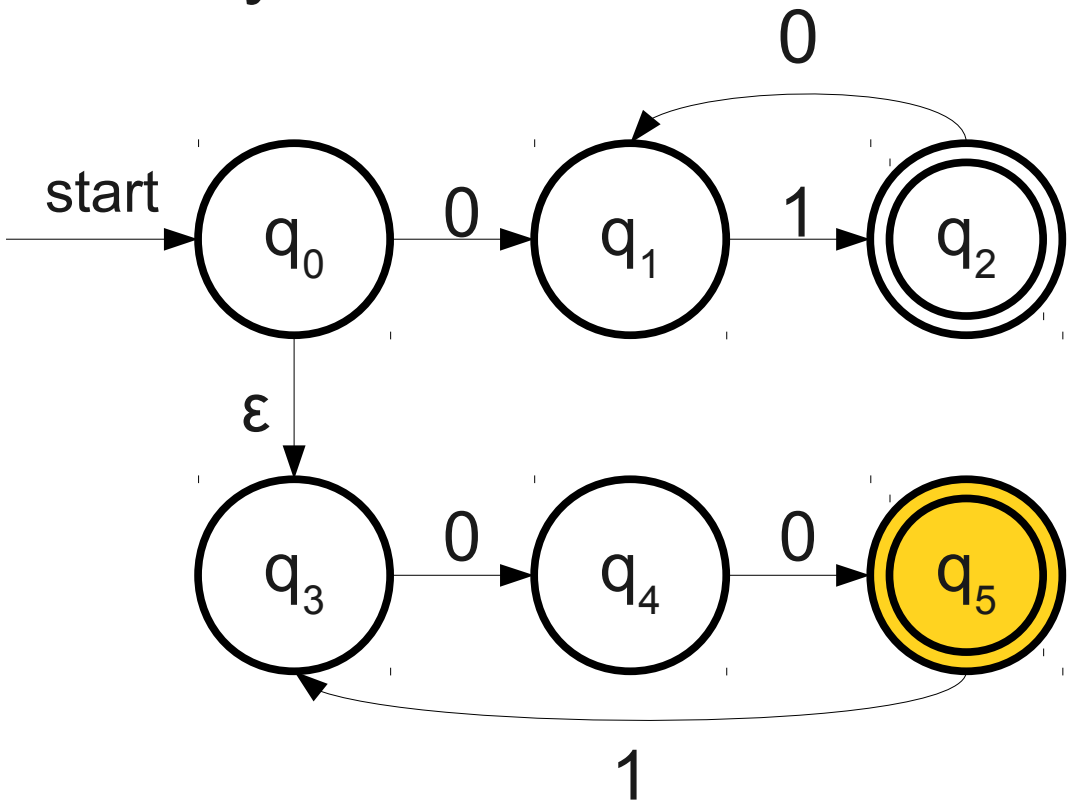
ε-Transitions

- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



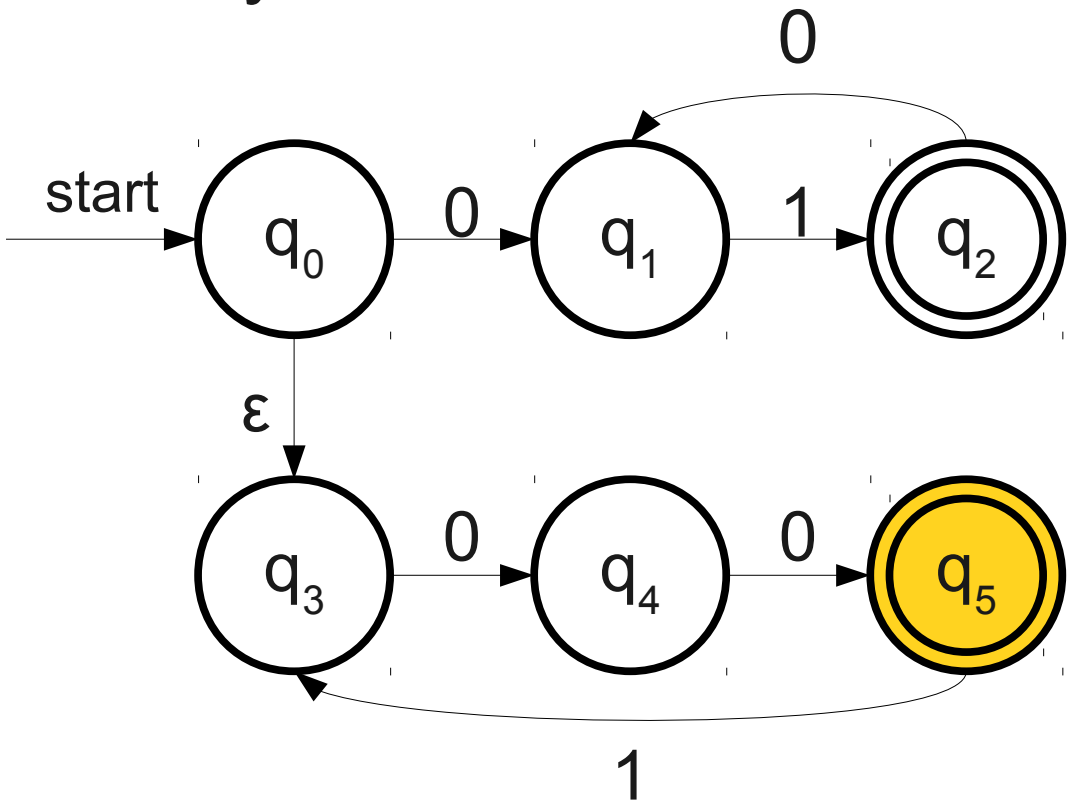
ε-Transitions

- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



ε-Transitions

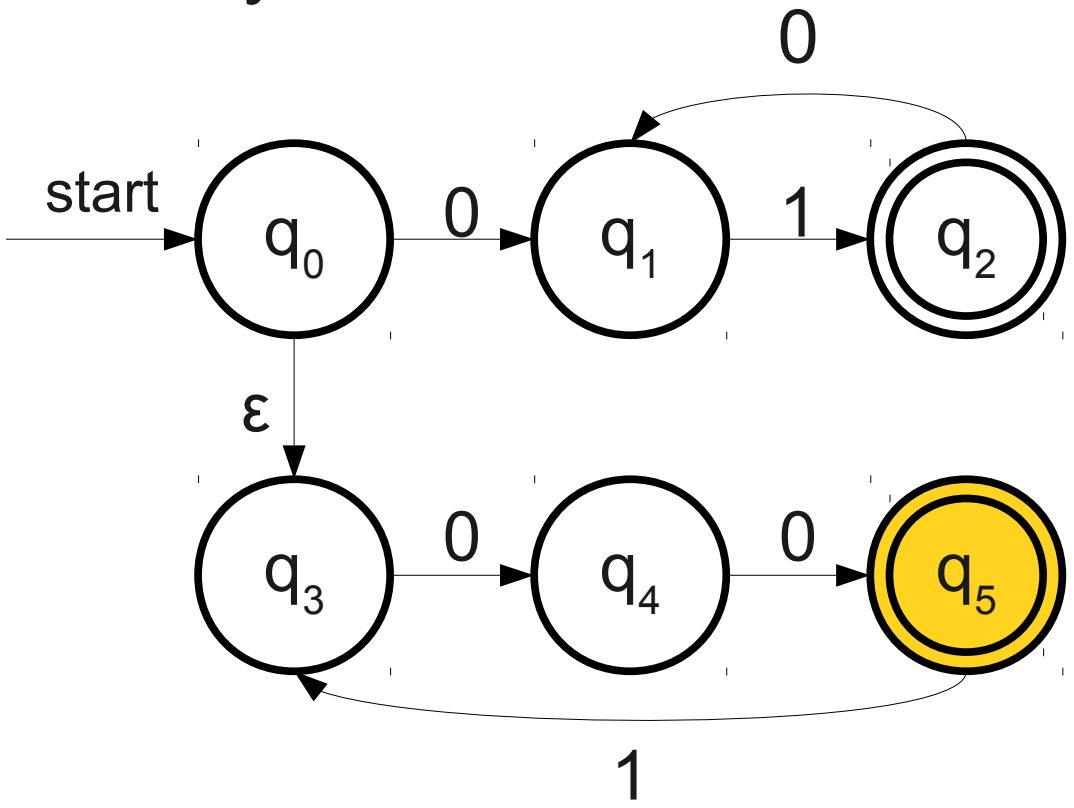
- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



0 0 1 0 0

ε-Transitions

- NFAs can have a special type of transition called an **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



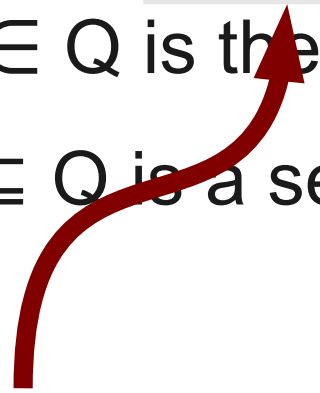
0 0 1 0 0

A Formal Definition of NFAs

- Formally, an NFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where
  - Q is a set of states.
  - Σ is an alphabet.
  - $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to P(Q)$ is the **transition function**.
  - $q_0 \in Q$ is the **start state**.
  - $F \subseteq Q$ is a set of **accepting states**.

A Formal Definition of NFAs

- Formally, an NFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where
  - Q is a set of states.
  - Σ is an alphabet.
  - $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to P(Q)$ is the **transition function**.
  - $q_0 \in Q$ is the **start state**.
  - $F \subseteq Q$ is a set of **accepting states**.



Note the domain allows for ε-moves

A Formal Definition of NFAs

- Formally, an NFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where
  - Q is a set of states.
  - Σ is an alphabet.
  - $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to P(Q)$ is the **transition function**.
  - $q_0 \in Q$ is the **start state**.
  - $F \subseteq Q$ is a set of **accepting states**.

Note the domain allows for ε-moves

Note that the codomain is sets of states to allow for multiple transitions.