Finite Automata

Part Two

Announcements

- Practice midterm solutions available.
- Second practice midterm available soon.
- Problem Set 3 and Problem Set 4 Checkpoints graded; will be returned at end of lecture.

A Friendly Reminder

$\forall$ goes with $\rightarrow$

$\exists$ goes with $\wedge$

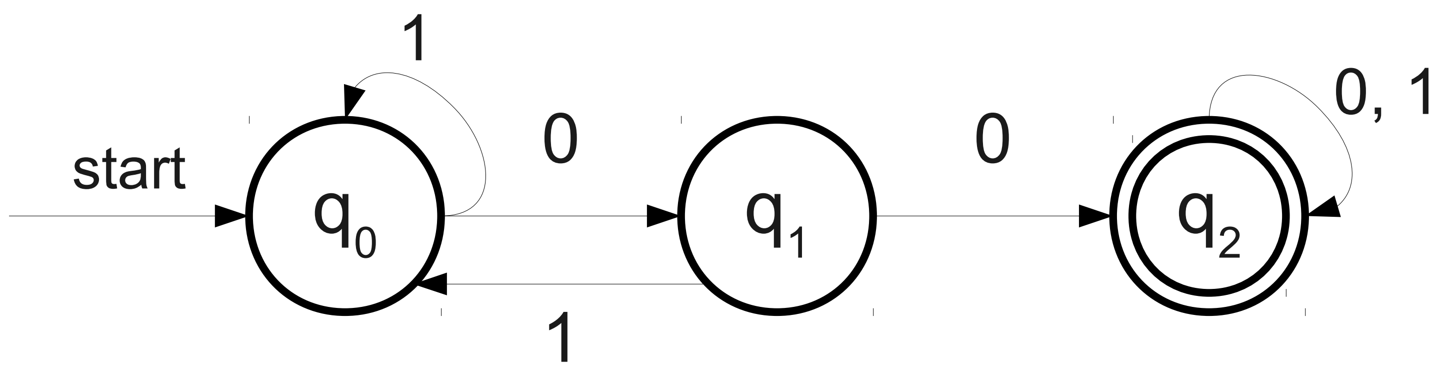
Finite Automata

DFAs, Informally

- A DFA is defined relative to some alphabet Σ.
- For each state in the DFA, there must be **exactly one** transition defined for each symbol in the alphabet.
  - This is the "deterministic" part of DFA.
- There is a **unique** start state.
- There may be multiple accepting states.

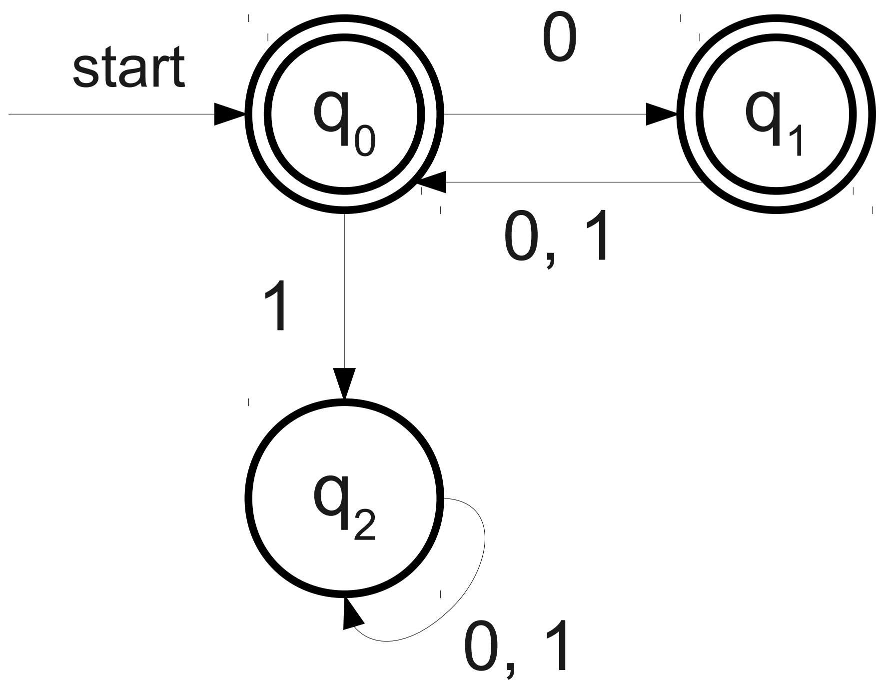
Recognizing Languages with DFAs

$L = \{ w \in \{0, 1\}^* \mid w \text{ contains } 00 \text{ as a substring } \}$



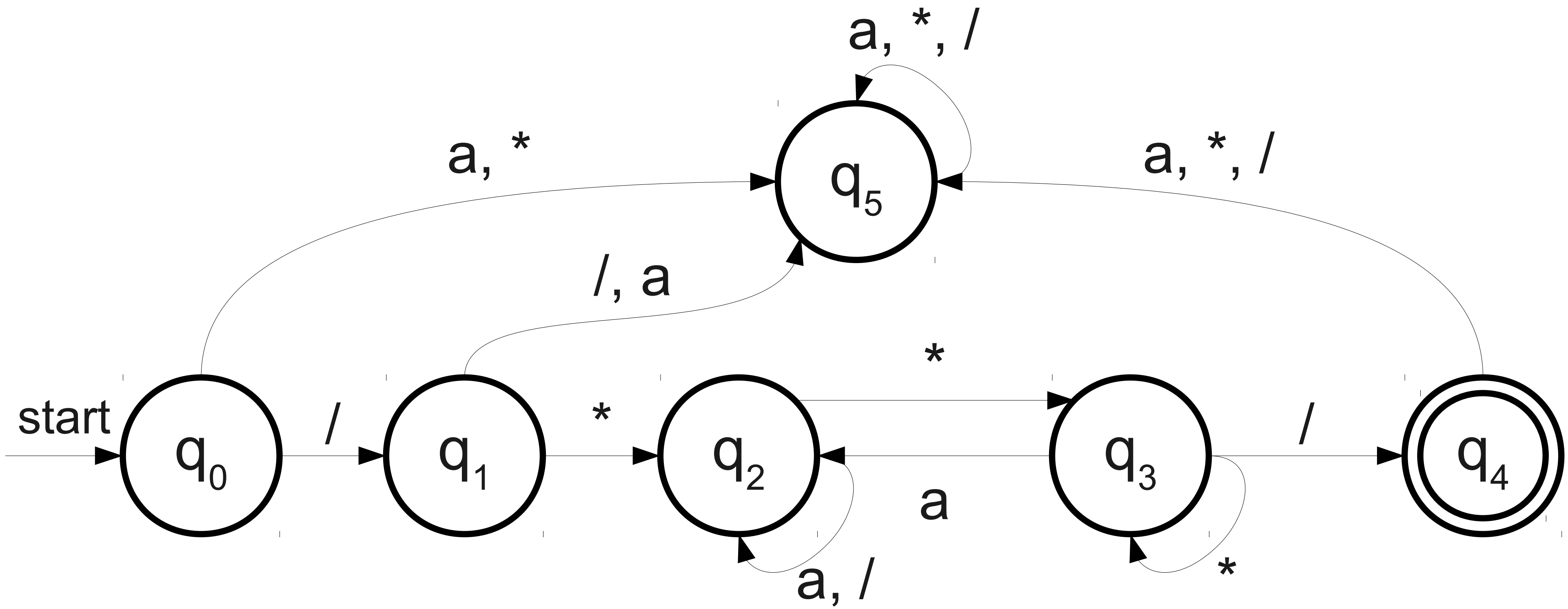
Recognizing Languages with DFAs

$L = \{ w \in \{0, 1\}^* |$ all even-numbered characters of w are 0 $\}$



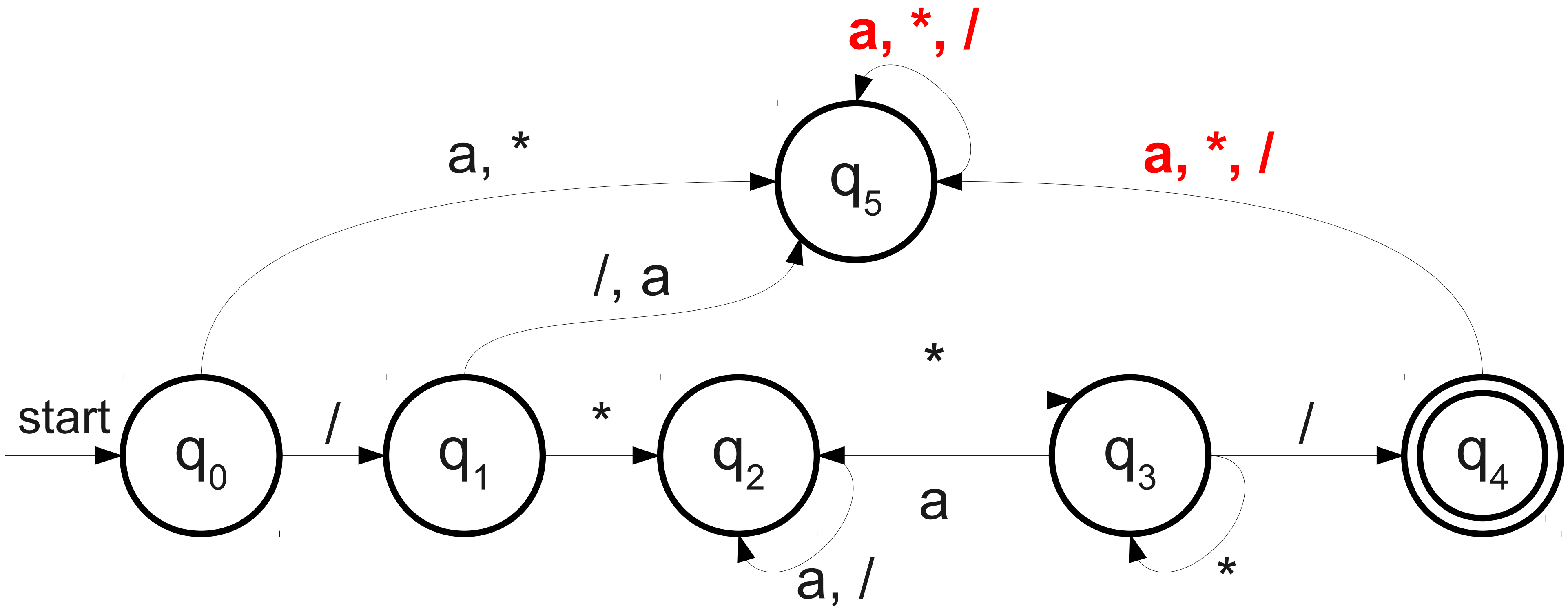
More Elaborate DFAs

$L = \{ w \mid w \text{ is a C-style comment } \}$



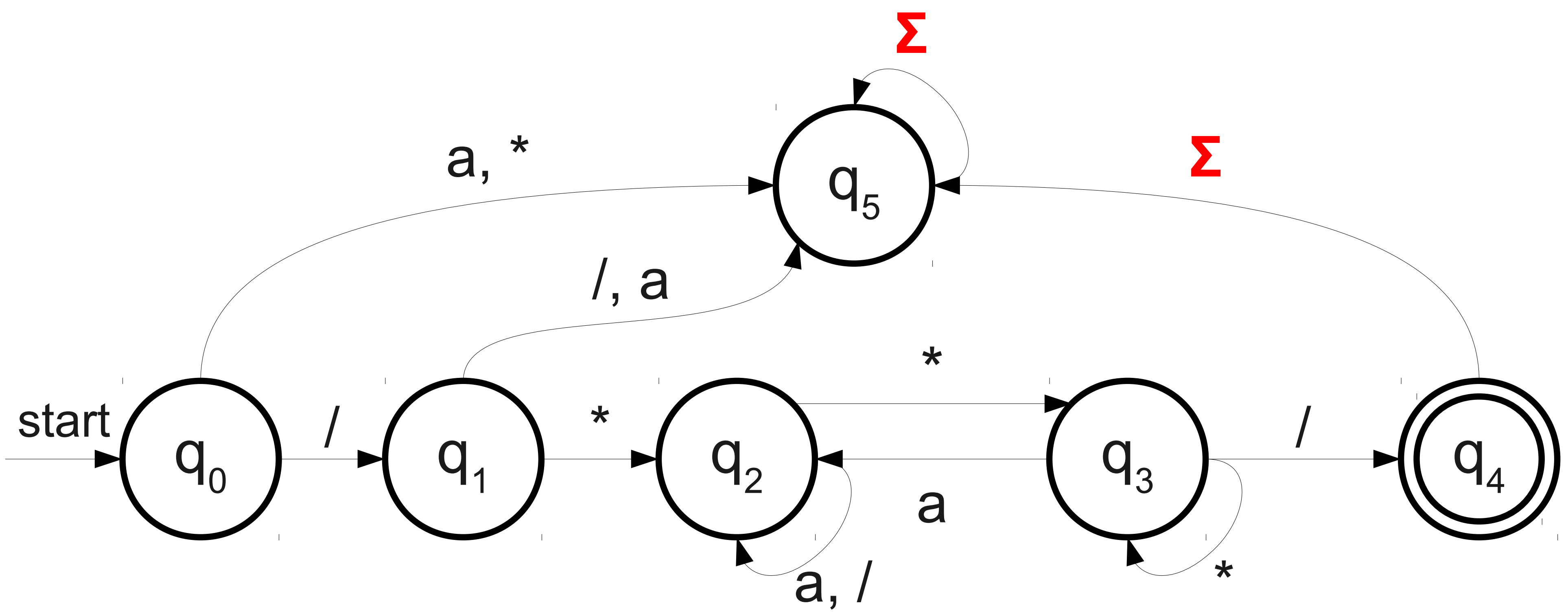
More Elaborate DFAs

$L = \{ \ w \mid w \text{ is a C-style comment } \}$



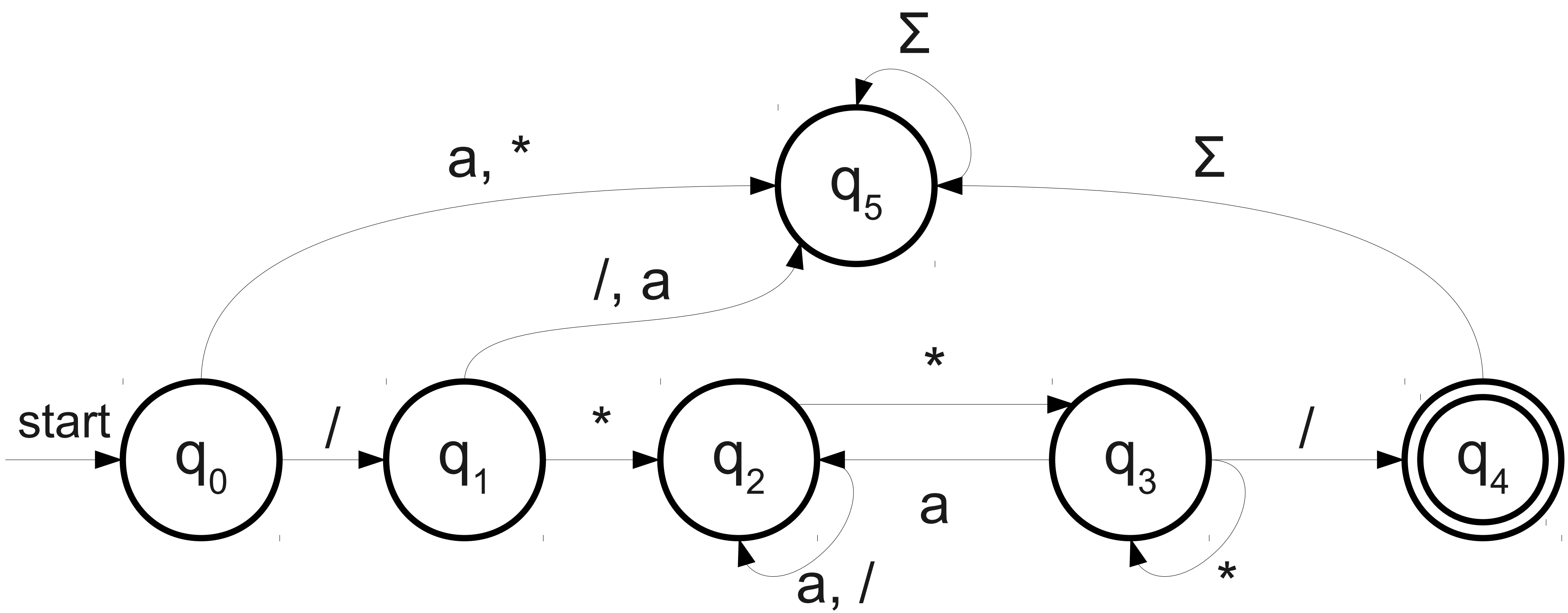
More Elaborate DFAs

$L = \{ \ w \mid w \text{ is a C-style comment } \}$



More Elaborate DFAs

$L = \{ \ w \mid w \text{ is a C-style comment} \ \}$

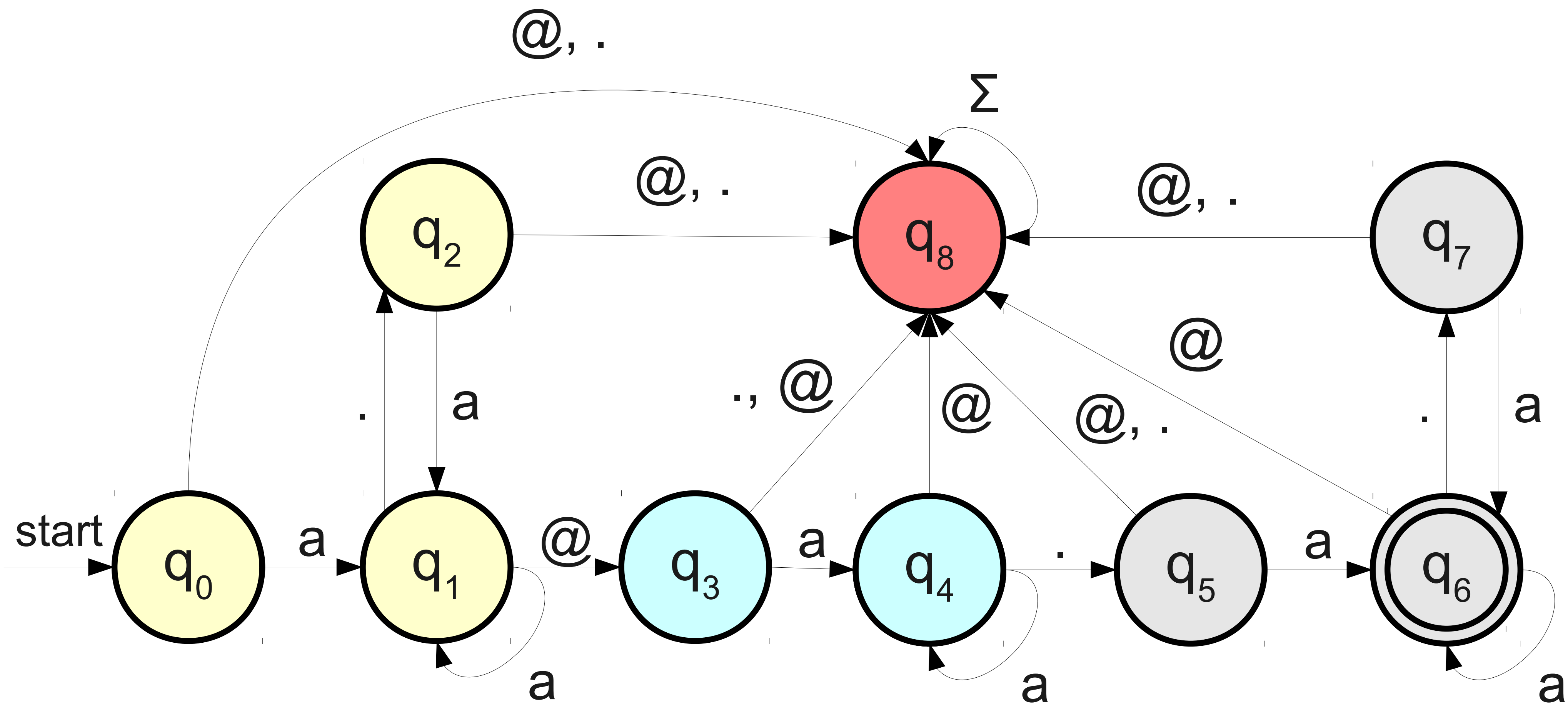


More Elaborate DFAs

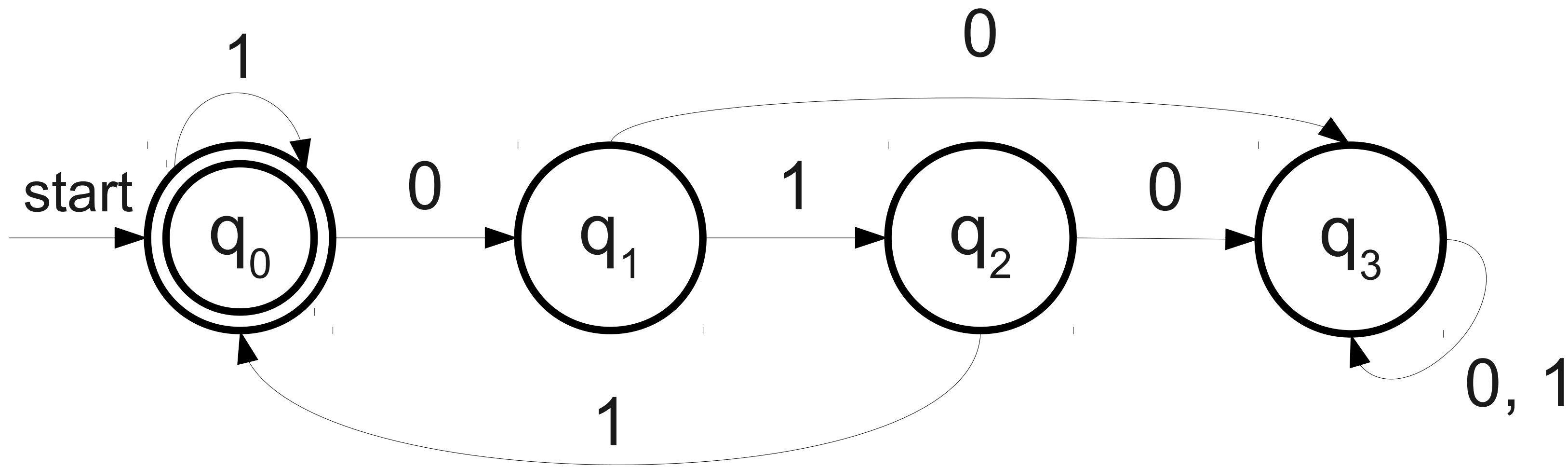
$L = \{ w \mid w \text{ is a legal email address } \}$

More Elaborate DFAs

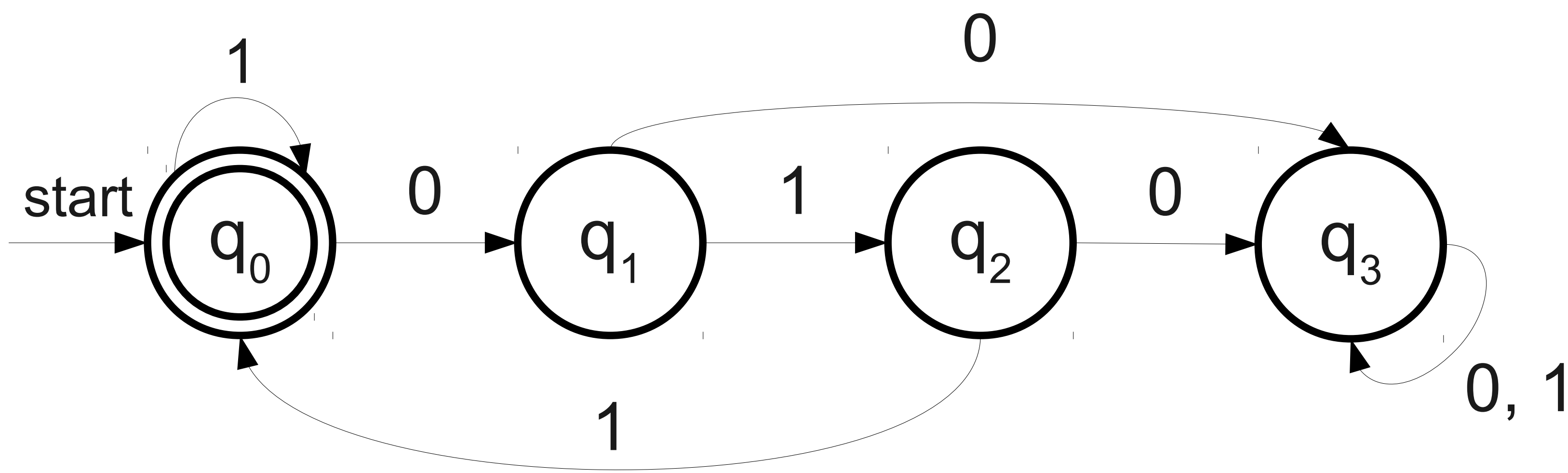
$L = \{ w \mid w \text{ is a legal email address } \}$



Tabular DFAs

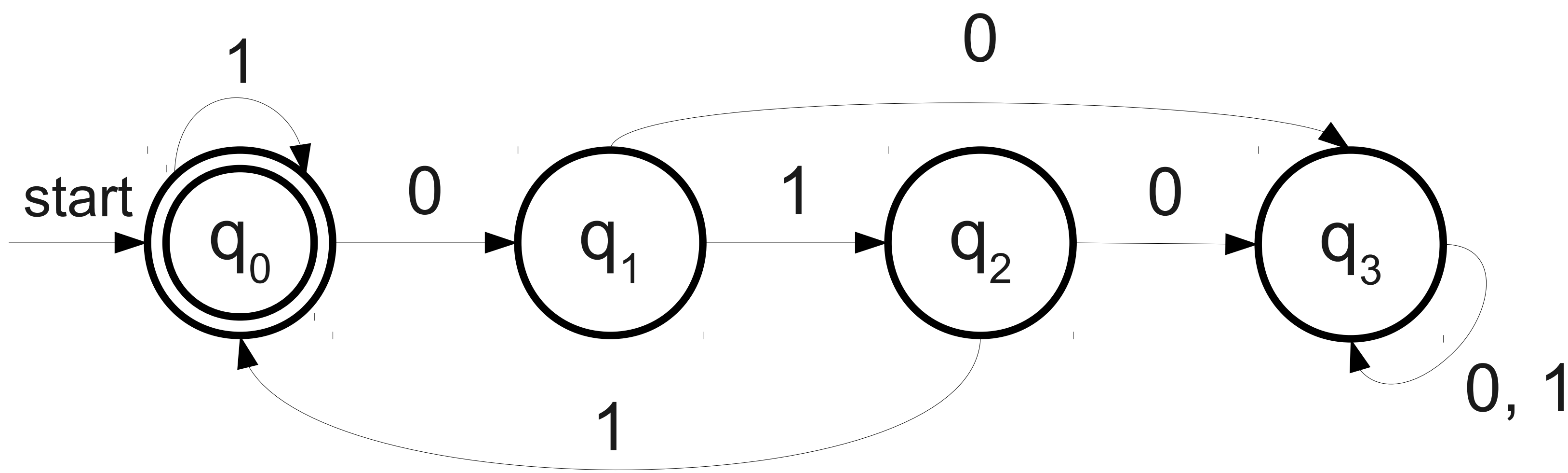


Tabular DFAs



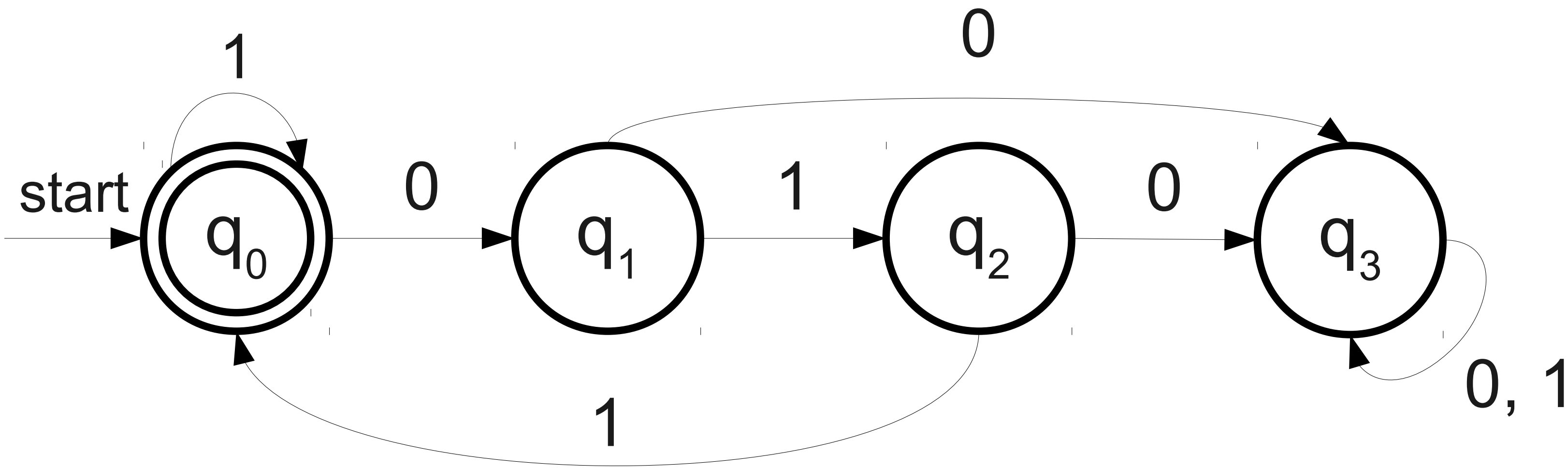
	0	1
q_0		
q_1		
q_2		
q_3		

Tabular DFAs



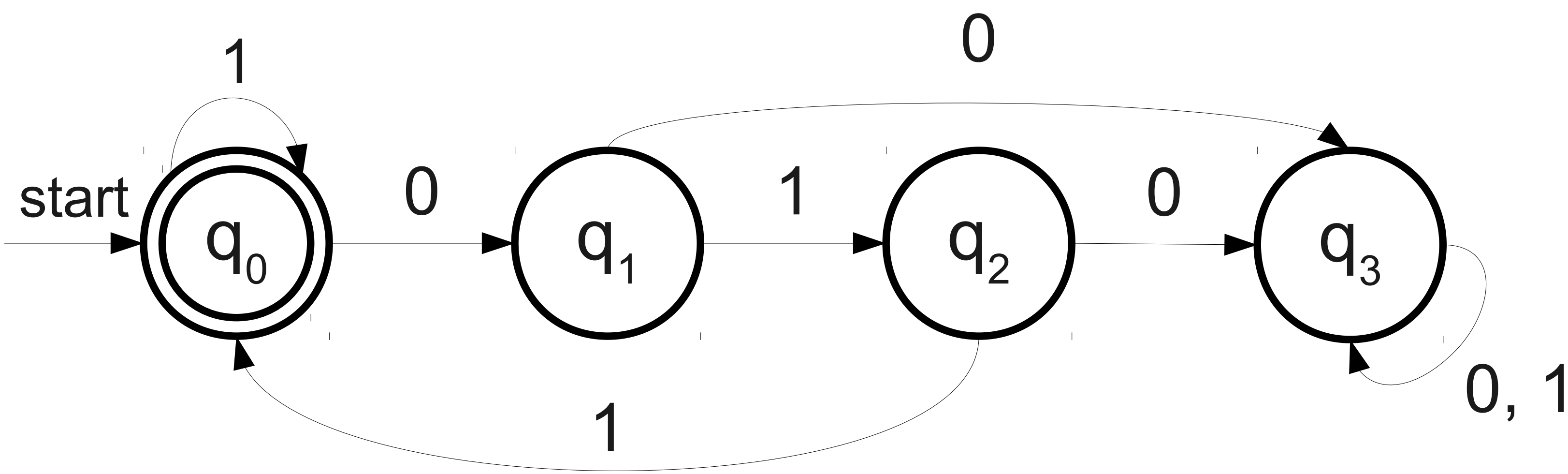
	0	1
q_0	q_1	
q_1		
q_2		
q_3		

Tabular DFAs



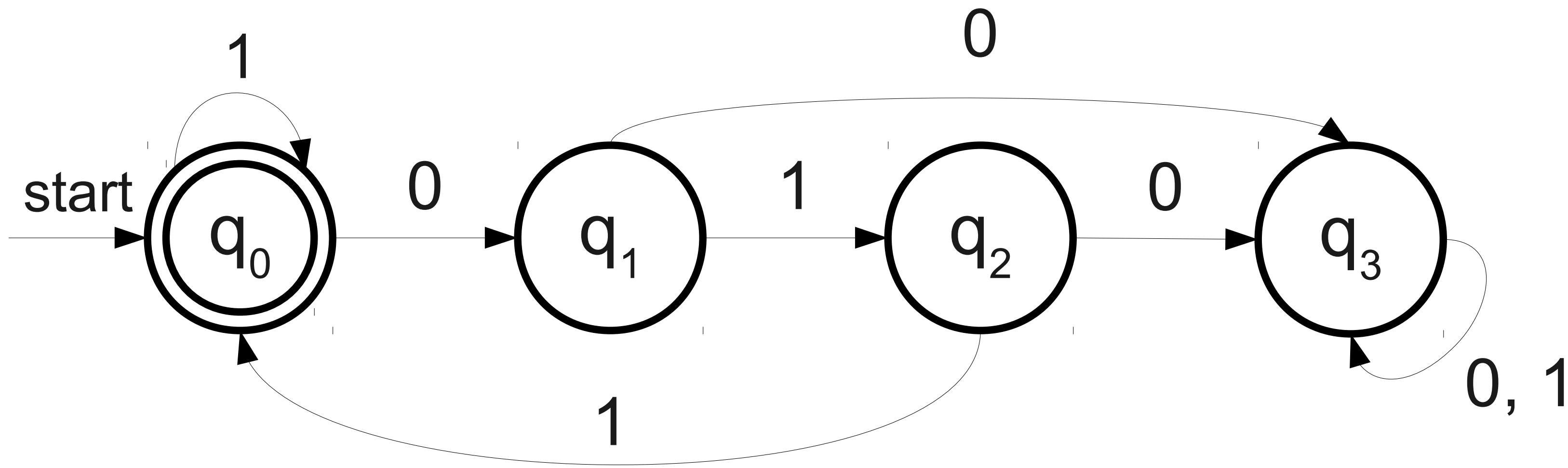
	0	1
q_0	q_1	q_0
q_1		
q_2		
q_3		

Tabular DFAs



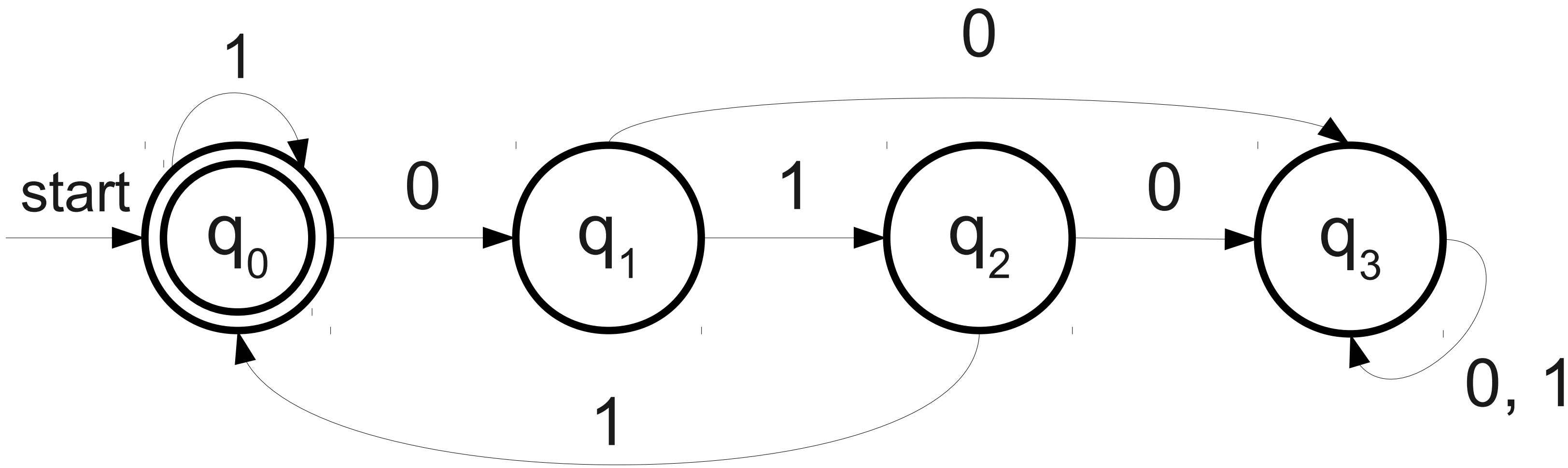
	0	1
q_0	q_1	q_0
q_1	q_3	
q_2		
q_3		

Tabular DFAs



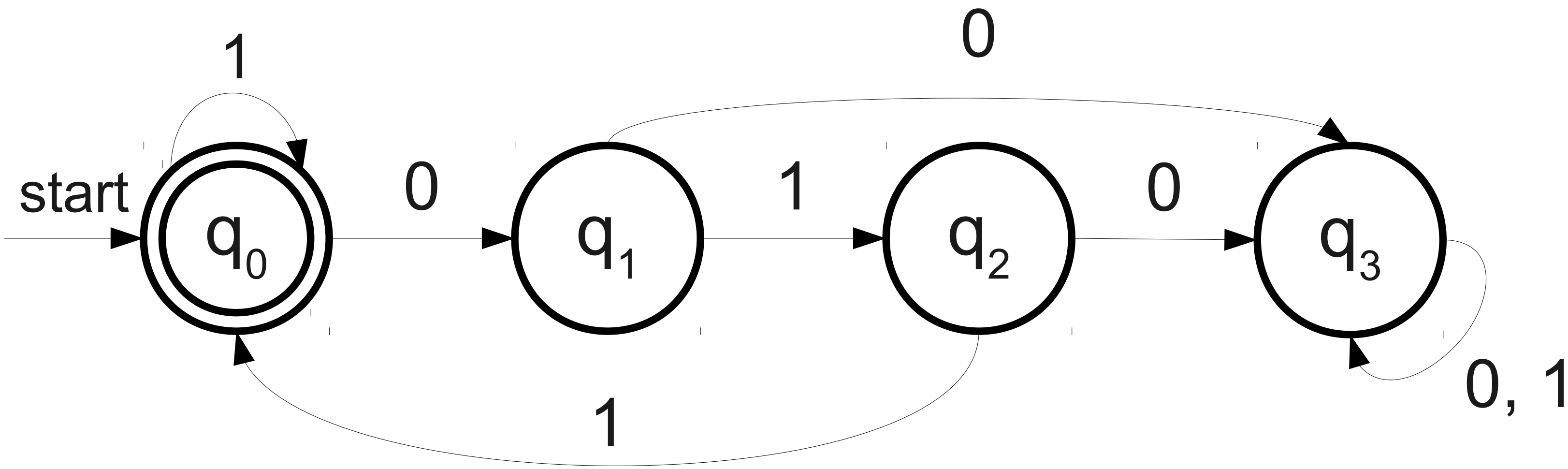
	0	1
q_0	q_1	q_0
q_1	q_3	q_2
q_2		
q_3		

Tabular DFAs



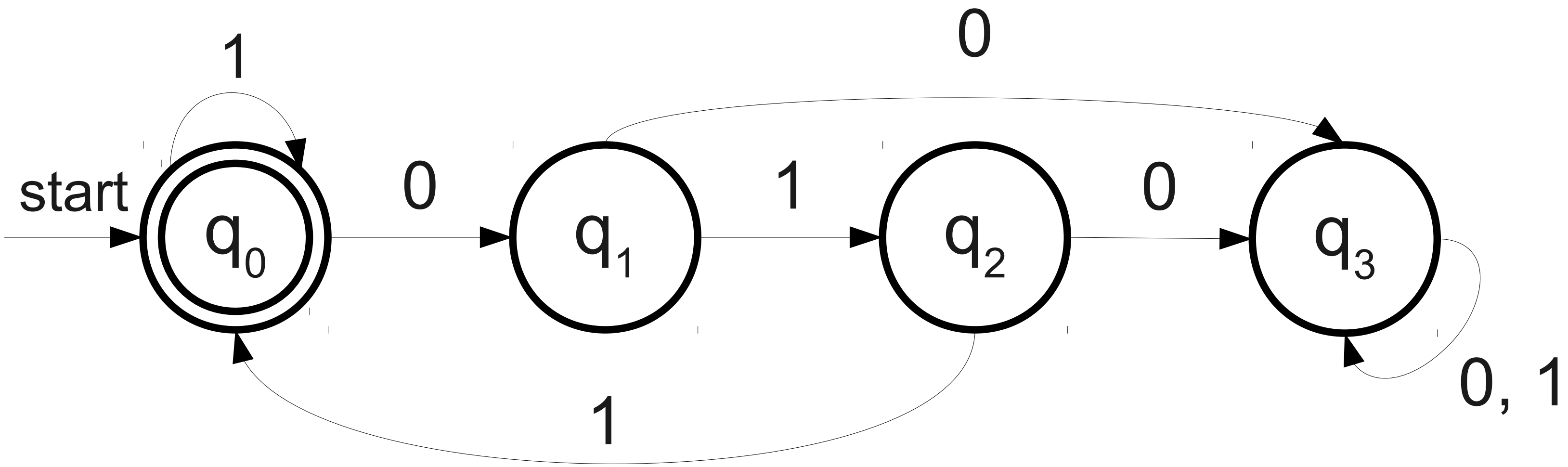
	0	1
q_0	q_1	q_0
q_1	q_3	q_2
q_2	q_3	
q_3		

Tabular DFAs



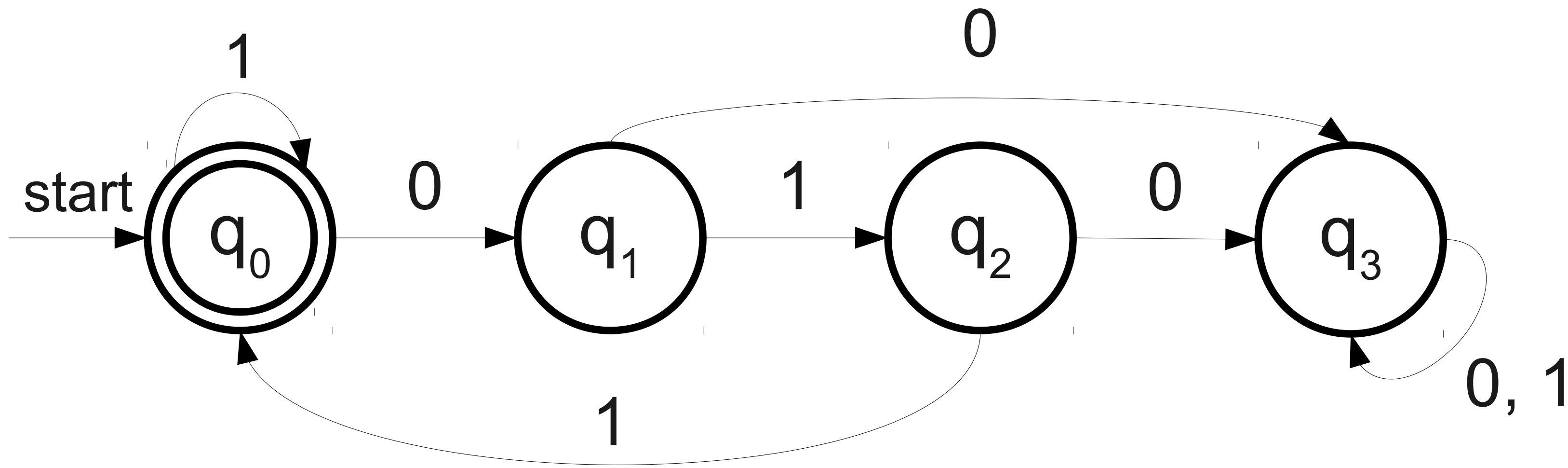
	0	1
q_0	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
q_3		

Tabular DFAs



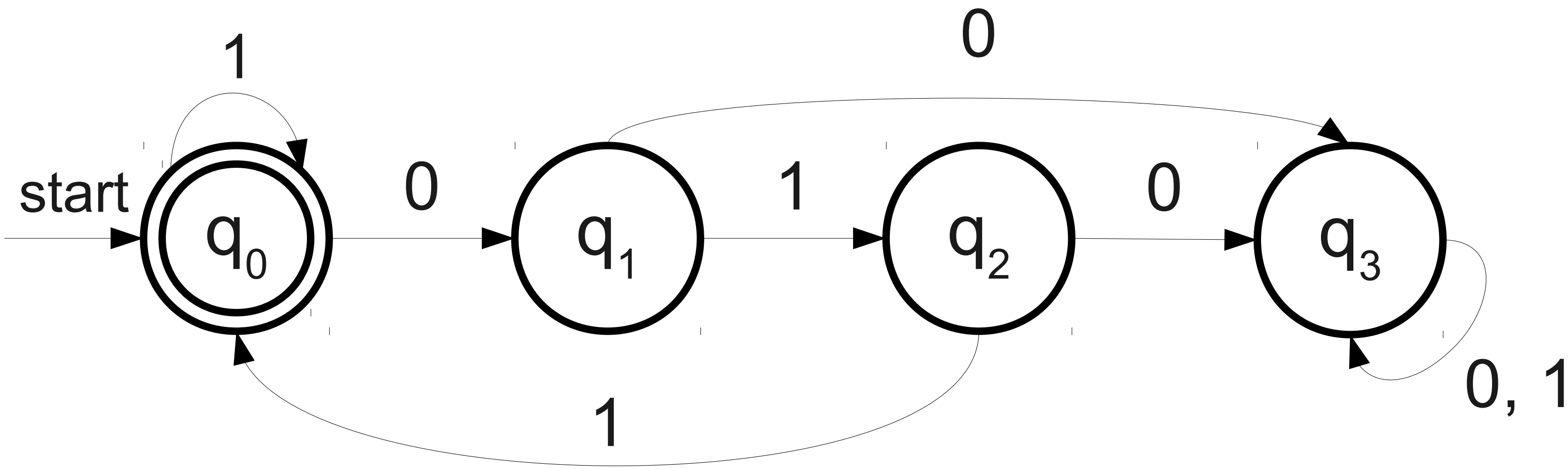
	0	1
q_0	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
q_3	q_3	

Tabular DFAs



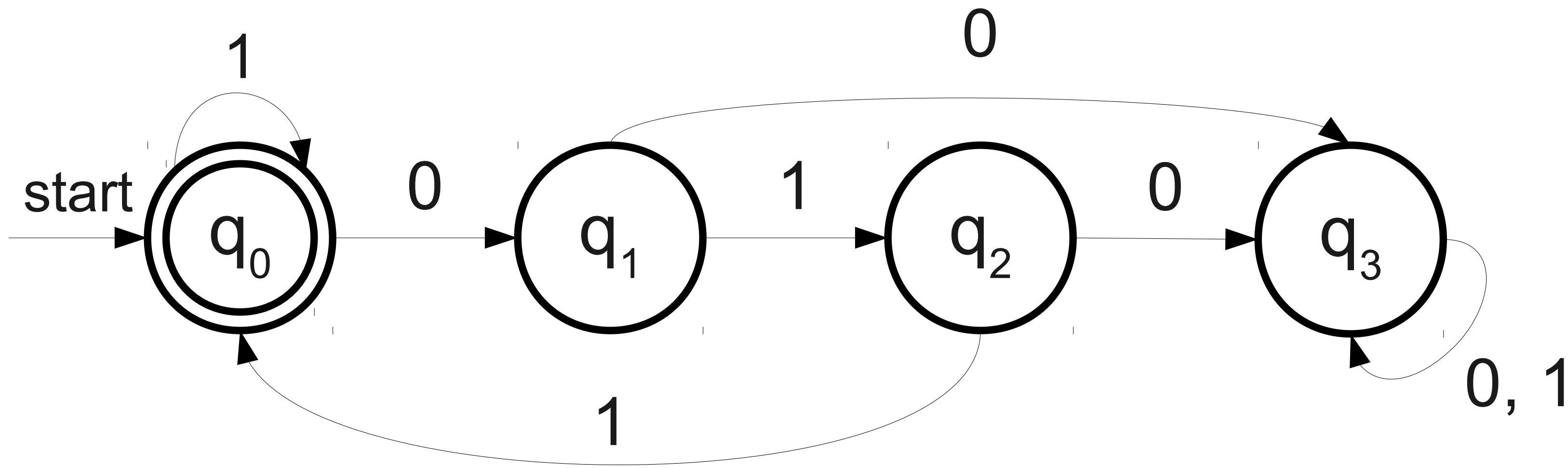
	0	1
q_0	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
q_3	q_3	q_3

Tabular DFAs



	0	1
*q_0	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
q_3	q_3	q_3

Tabular DFAs



	0	1
*q_0	q_1	q_0
q_1	q_3	q_2
q_2	q_3	q_0
q_3	q_3	q_3

The star indicates that this is an accepting state.

Code? In a Theory Course?

```
int kTransitionTable[kNumStates][kNumSymbols] = {
     {0, 0, 1, 3, 7, 1, …},
      …
};
bool kAcceptTable[kNumStates] = {
    false,
    true,
    true,
    …
};
bool SimulateDFA(string input) {
    int state = 0;
    for (char ch: input)
        state = kTransitionTable[state][ch];
    return kAcceptTable[state];
}
```

A language L is called a **regular language** iff there exists a DFA D such that $\mathscr{L}(D) = L$.

The Complement of a Language

- Given a language $L \subseteq \Sigma^*$, the **complement** of that language (denoted $\overline{L}$) is the language of all strings in Σ^* not in L.
- Formally:

$$\overline{L} = \{ \ w \mid w \in \Sigma^* \land w \notin L \ \}$$

The Complement of a Language

- Given a language $L \subseteq \Sigma^*$, the **complement** of that language (denoted $\overline{L}$) is the language of all strings in Σ^* not in L.
- Formally:

$$\overline{L} = \boldsymbol{\{\ w \mid w \in \Sigma^* \land w \notin L\ \}}$$

The Complement of a Language

- Given a language $L \subseteq \Sigma^*$, the **complement** of that language (denoted $\overline{L}$) is the language of all strings in Σ^* not in L.
- Formally:

$$\overline{L} = \boldsymbol{\Sigma^* - L}$$

The Complement of a Language

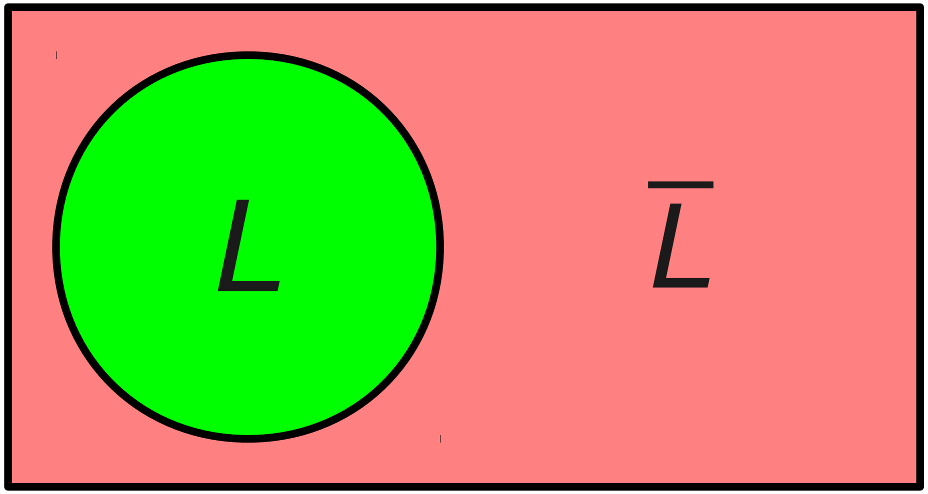
- Given a language $L \subseteq \Sigma^*$, the **complement** of that language (denoted $\overline{L}$) is the language of all strings in Σ^* not in L.
- Formally:

$$\overline{L} = \Sigma^* - L$$

The Complement of a Language

- Given a language $L \subseteq \Sigma^*$, the **complement** of that language (denoted $\overline{L}$) is the language of all strings in Σ^* not in L.
- Formally:

$$\overline{L} = \Sigma^* - L$$

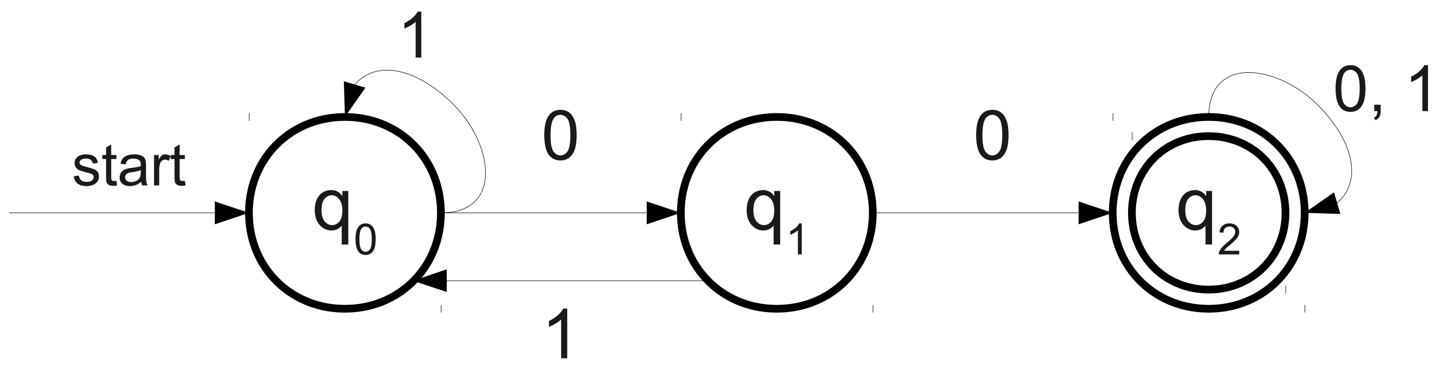


Complementing Regular Languages

- Recall: A **regular language** is a language accepted by some DFA.
- **Question:** If L is a regular language, is $\overline{L}$ a regular language?
- If the answer is "yes," then there must be some way to construct a DFA for $\overline{L}$.
- If the answer is "no," then some language L can be accepted by a DFA, but $\overline{L}$ cannot be accepted by any DFA.

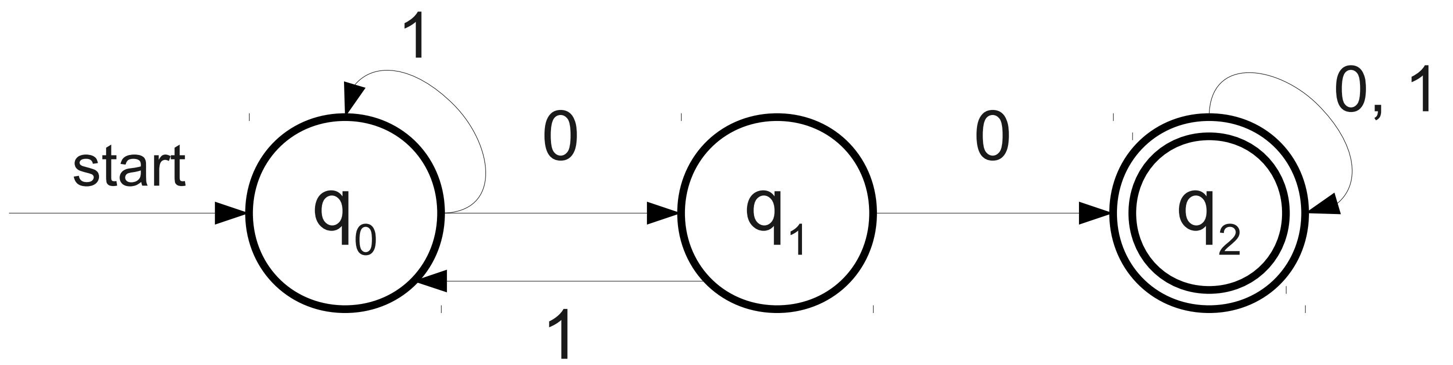
Complementing Regular Languages

$L = \{ \ w \in \{0, 1\}^* \mid w \text{ contains } 00 \text{ as a substring } \}$



Complementing Regular Languages

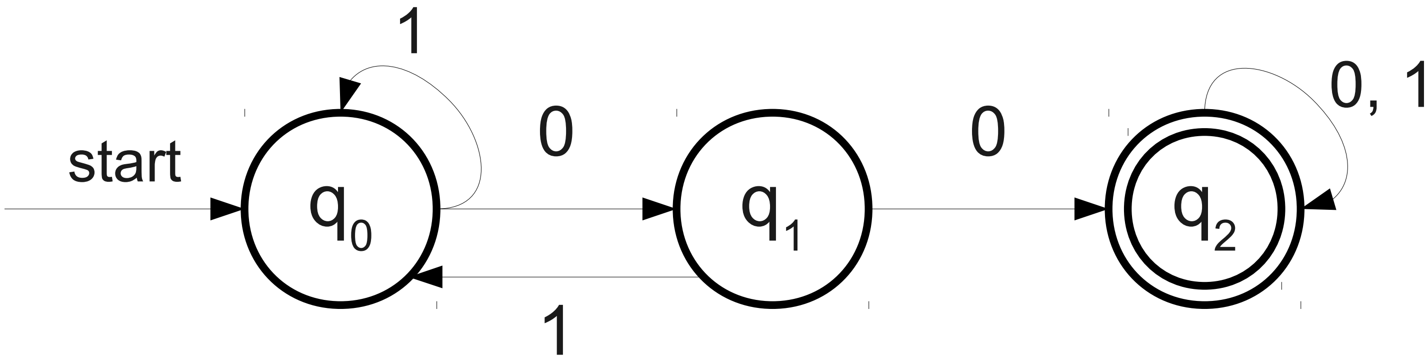
$L = \{ \ w \in \{0, 1\}^* \mid w \text{ contains } 00 \text{ as a substring } \}$



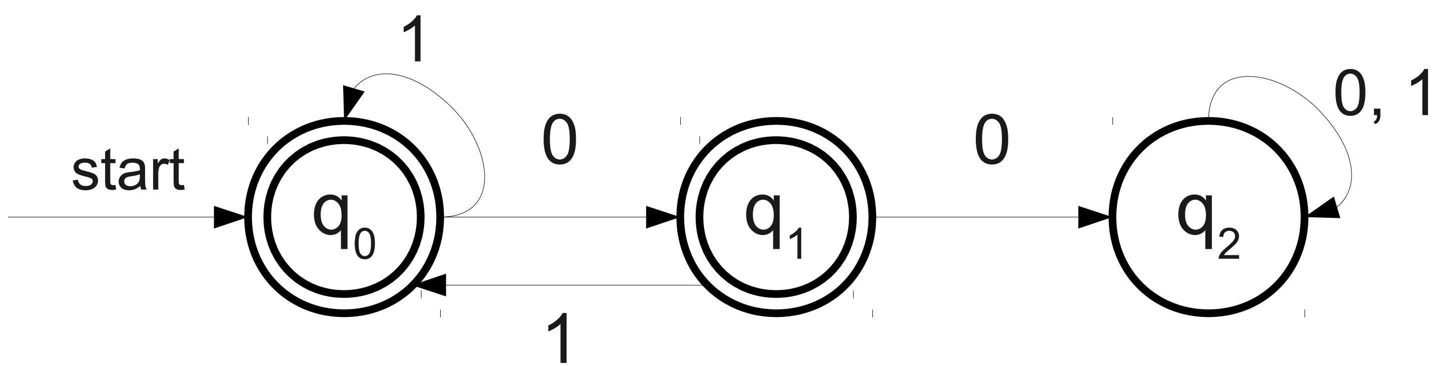
$\overline{L} = \{ \ w \in \{0, 1\}^* \mid w$ **does not contain** 00 as a substring $\}$

Complementing Regular Languages

$L = \{ w \in \{0, 1\}^* \mid w$ contains 00 as a substring $\}$

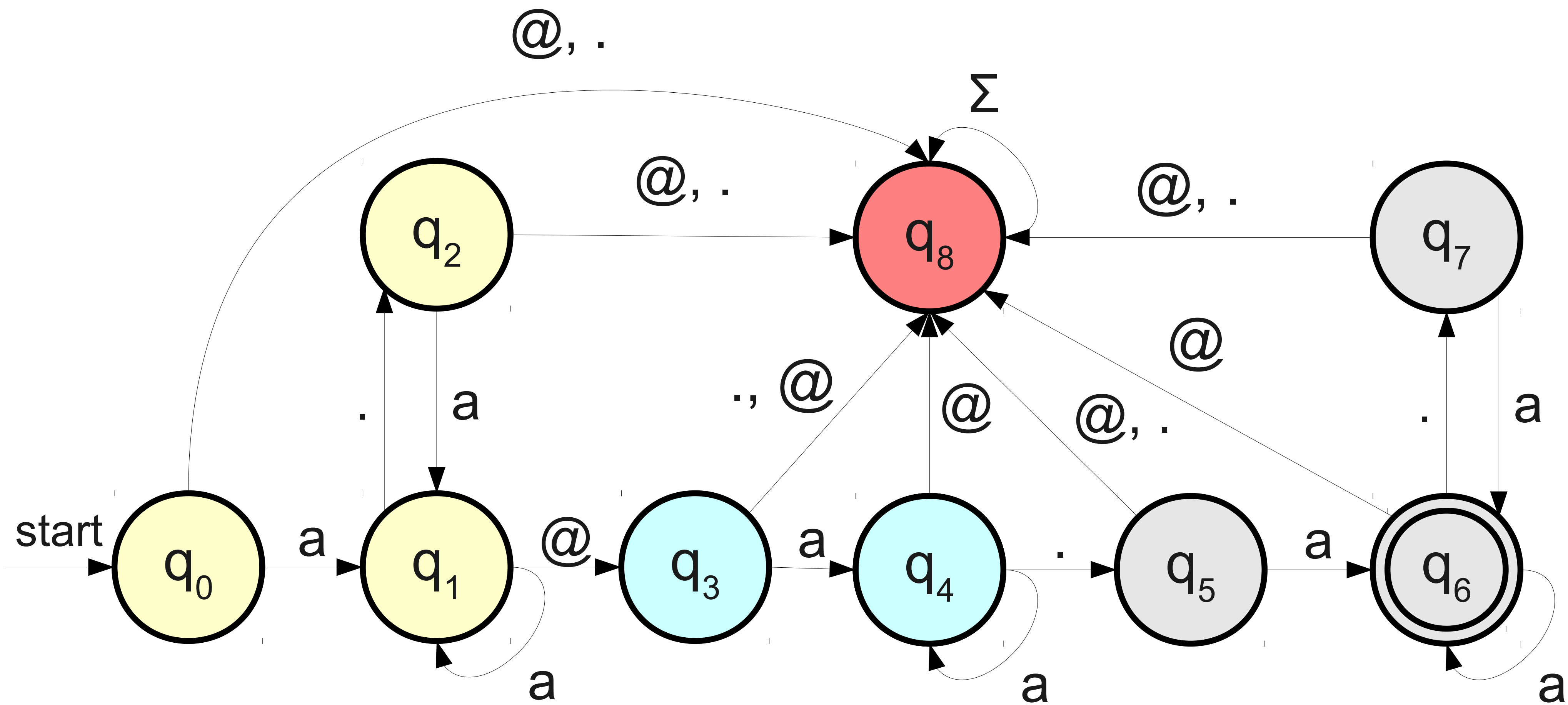


$\overline{L} = \{ w \in \{0, 1\}^* \mid w$ **does not contain** 00 as a substring $\}$



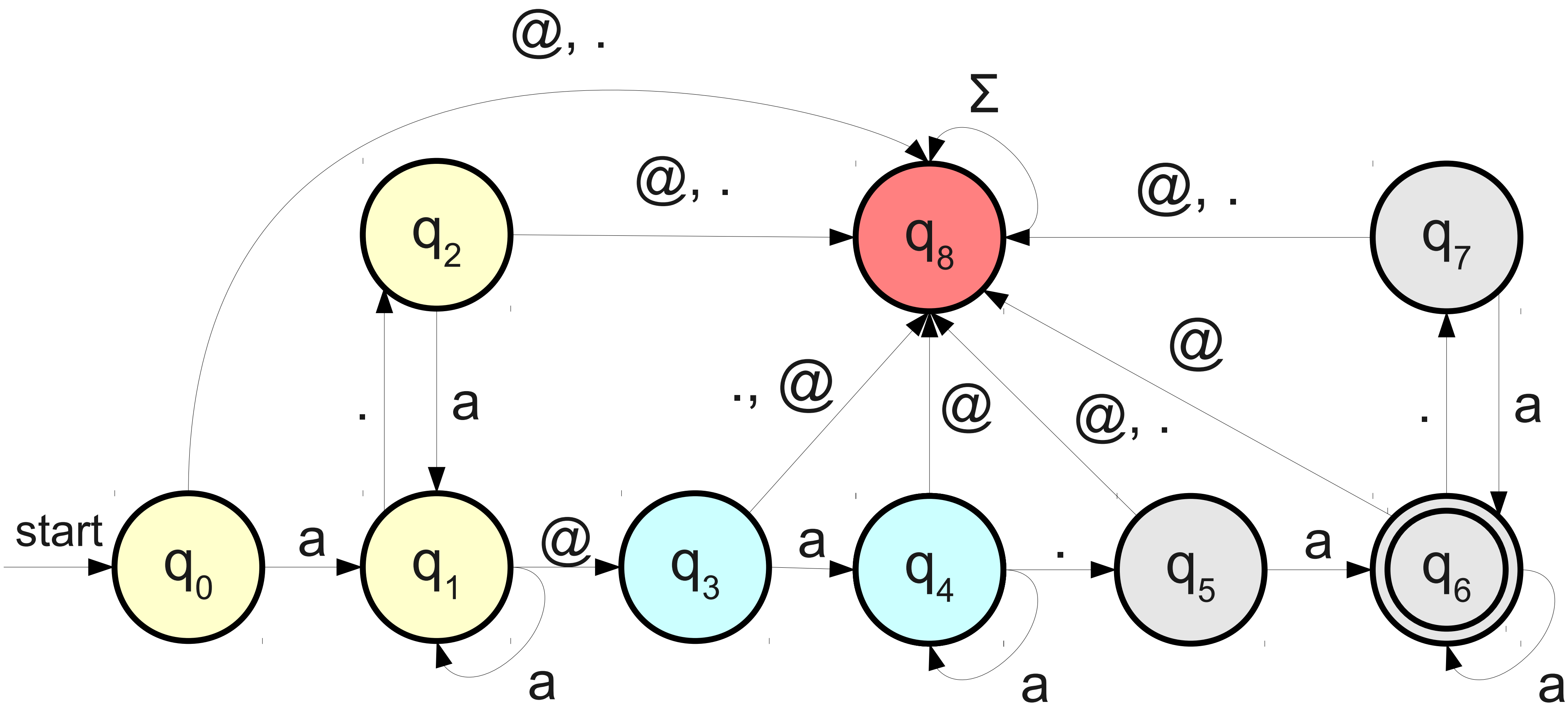
Complementing Regular Languages

$$L = \{ \ w \mid w \text{ is a legal email address } \}$$



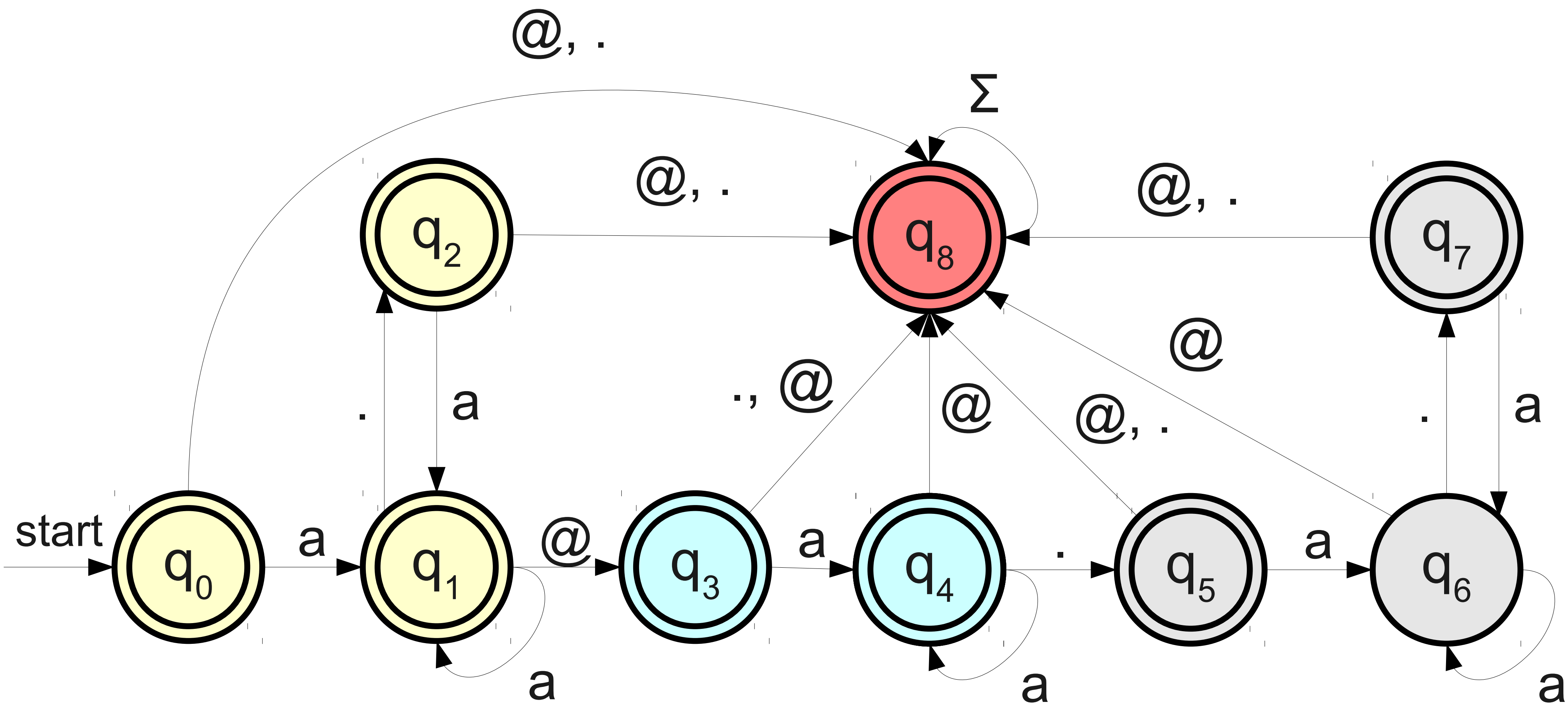
Complementing Regular Languages

$\overline{L} = \{\ w \mid w$ is **not** a legal email address $\}$



Complementing Regular Languages

$\overline{L} = \{ \ w \mid w$ is **not** a legal email address $\}$



Constructions on Automata

- Much of our discussion of automata will consider constructions that transform one automaton into another.
- Exchanging accepting and rejecting states is a simple construction sometimes called the **complement construction**.

Closure Properties

- If L is a regular language, $\overline{L}$ is a regular language.
- If we begin with a regular language and complement it, we end up with a regular language.
- This is an example of a **closure property of regular languages**.
  - The regular languages are **closed under complementation**.
  - We'll see more such properties later on.

NFAs

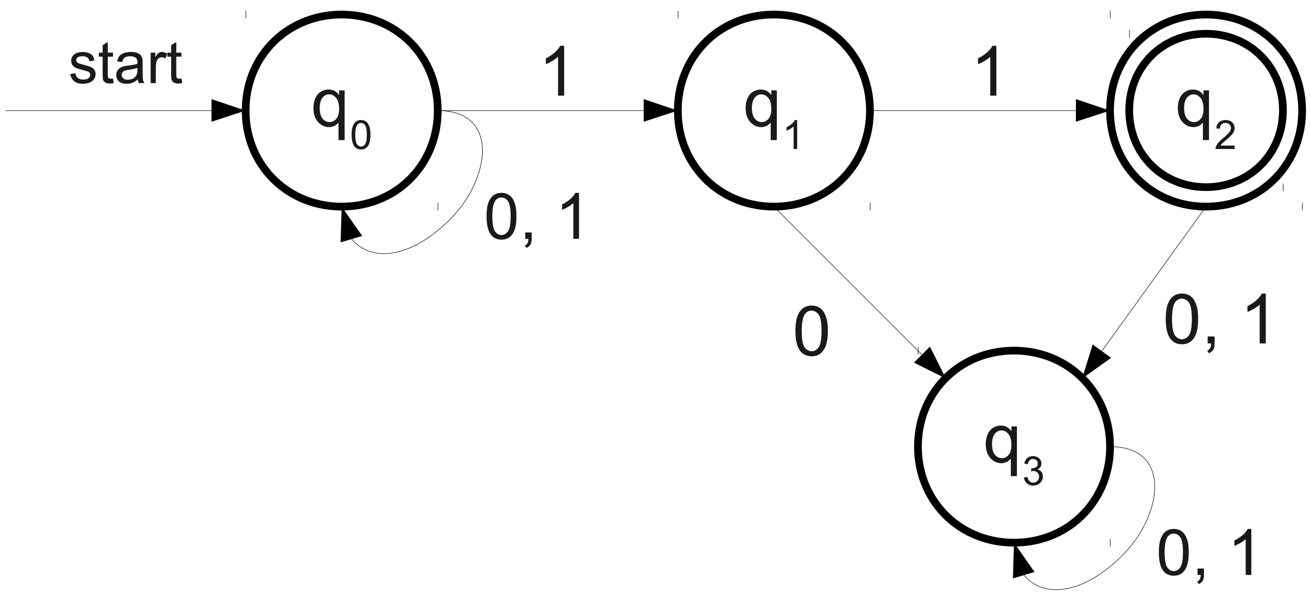
NFAs

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

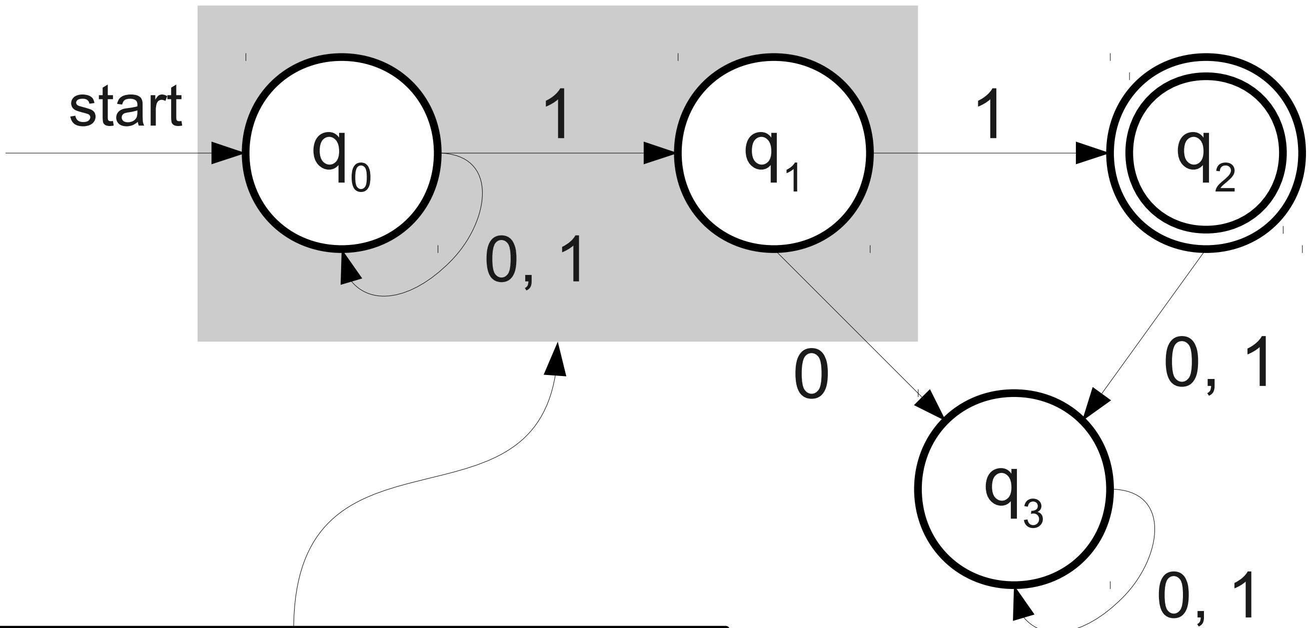
(Non)determinism

- 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

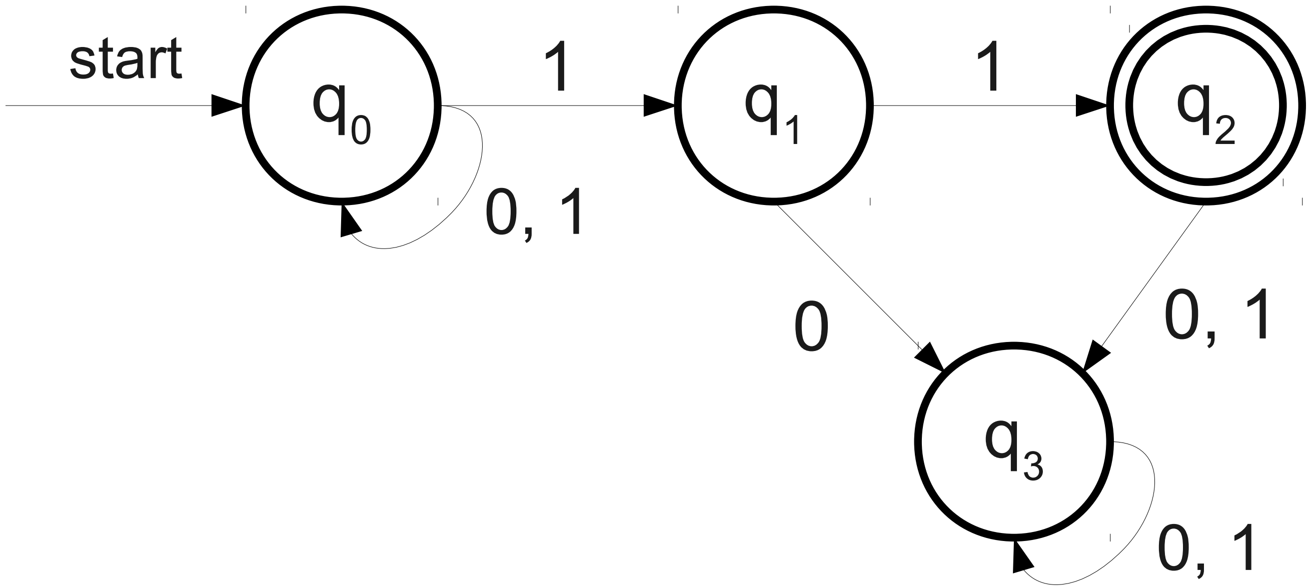


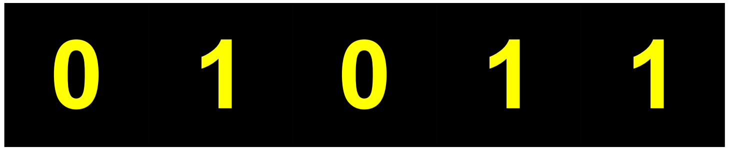
A Simple NFA



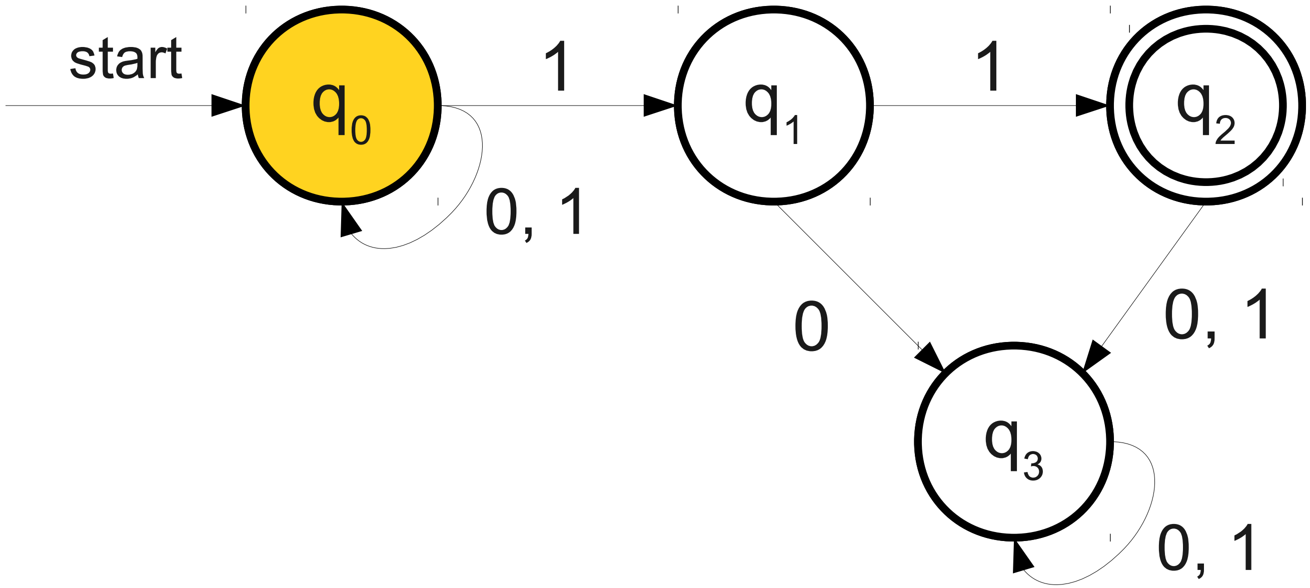
q_0 has two transitions defined on 1!

A Simple NFA



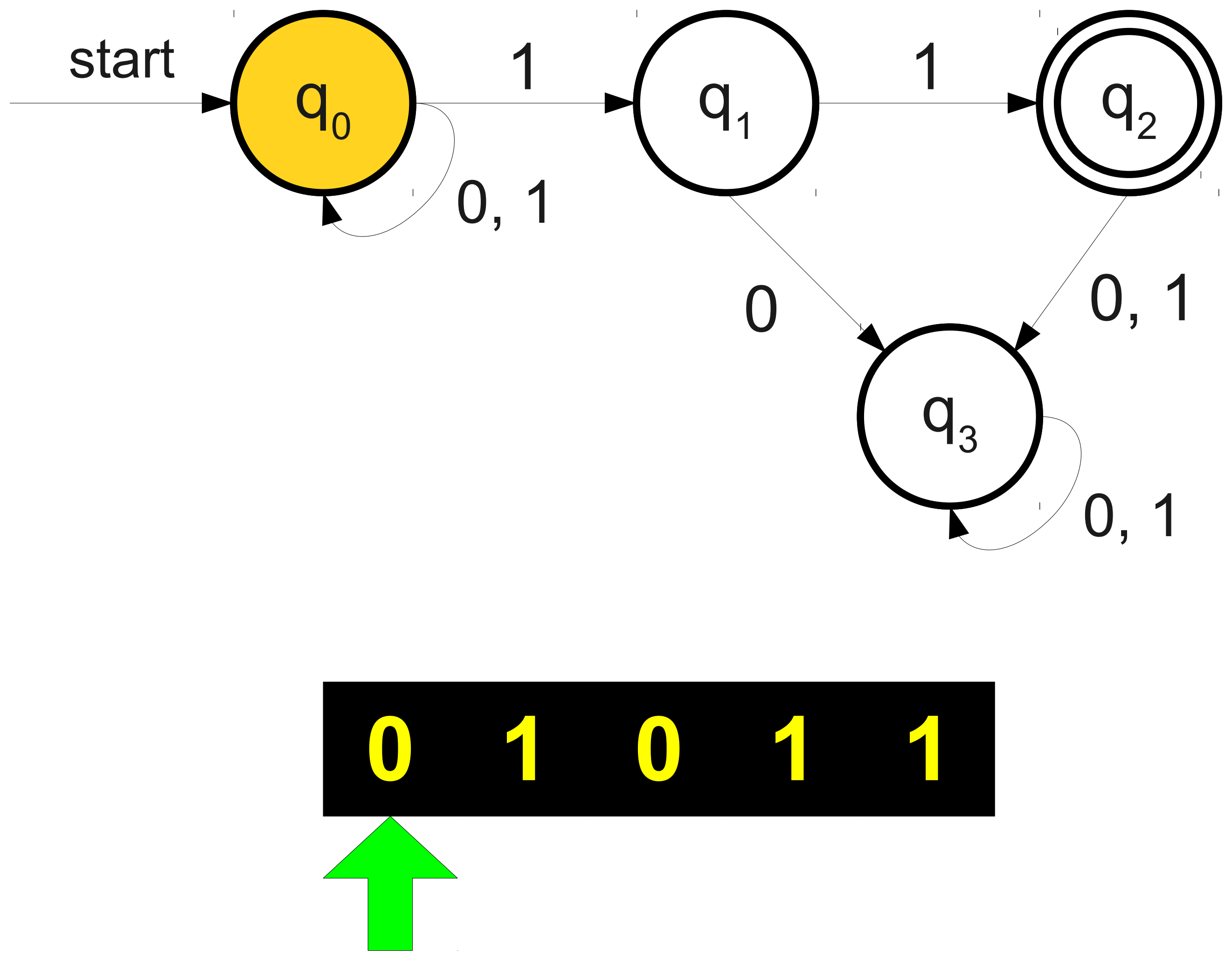


A Simple NFA

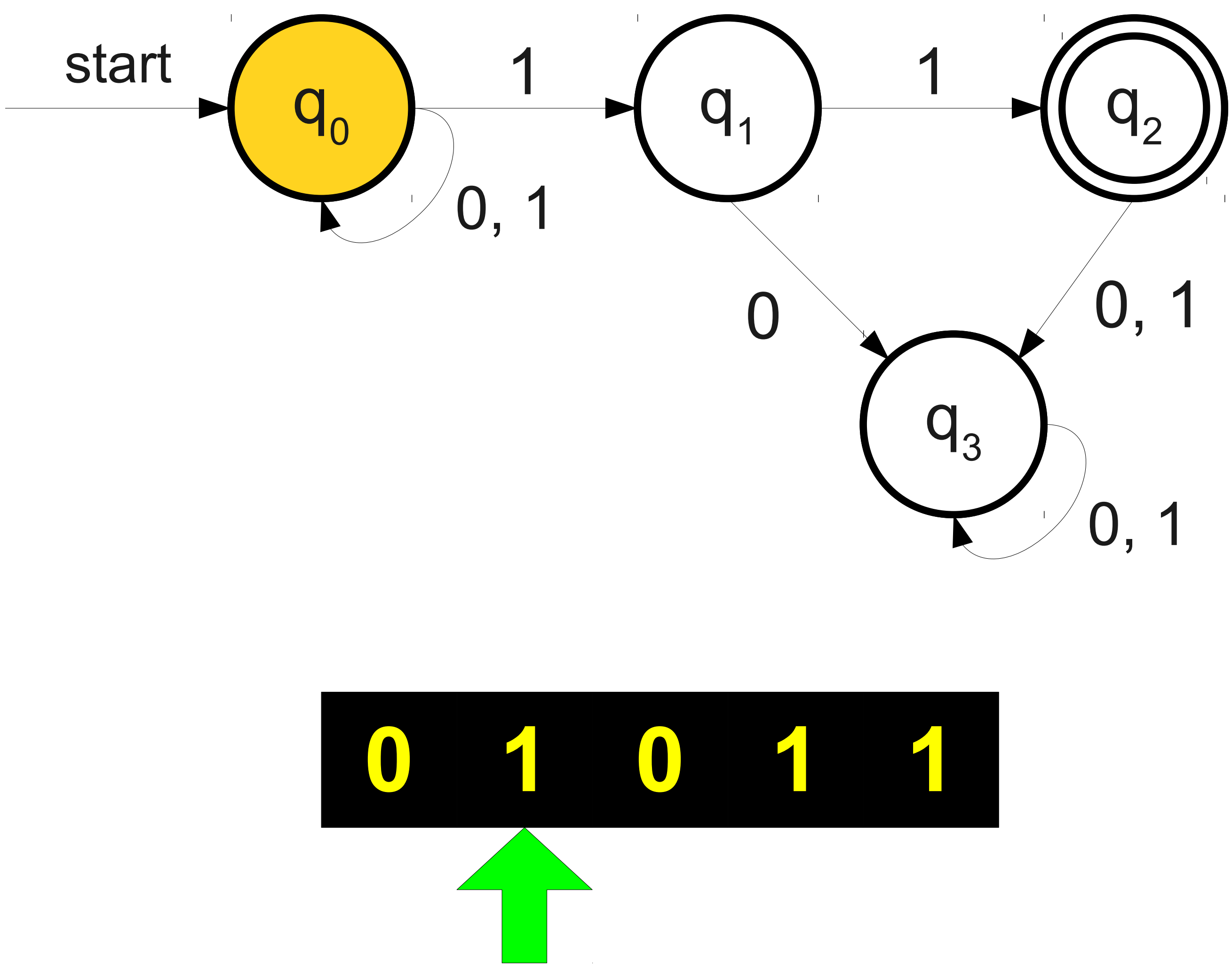


0 1 0 1 1

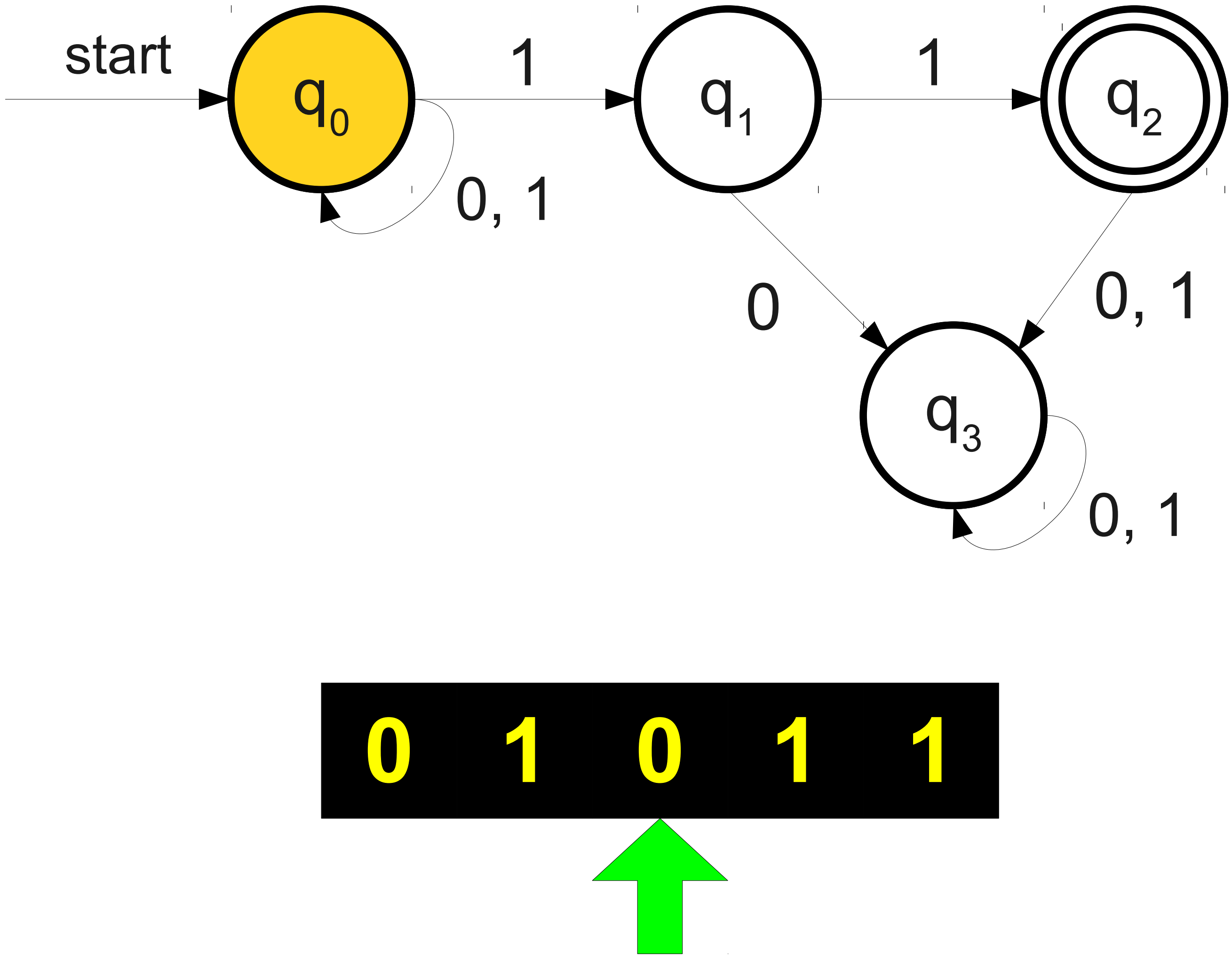
A Simple NFA



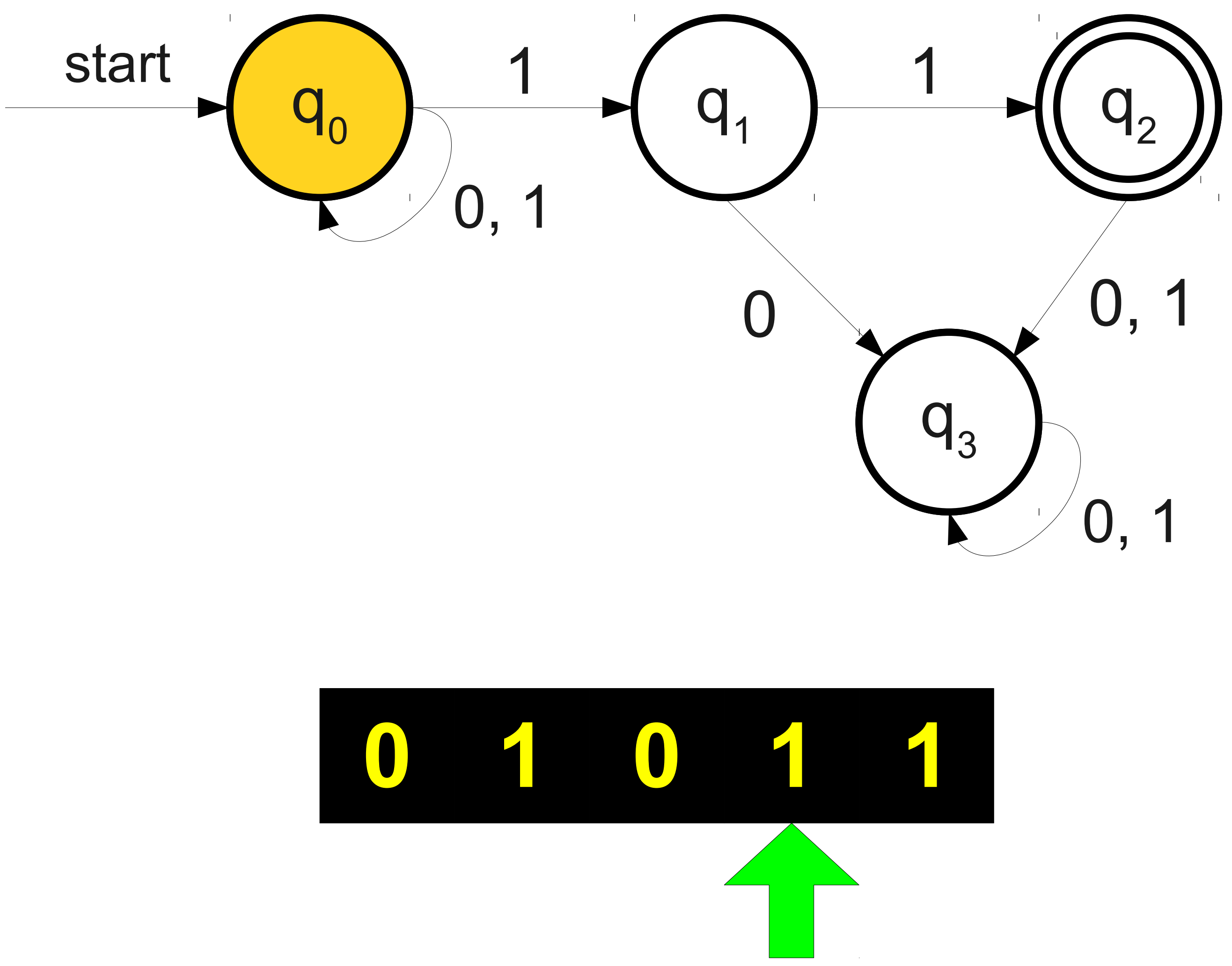
A Simple NFA



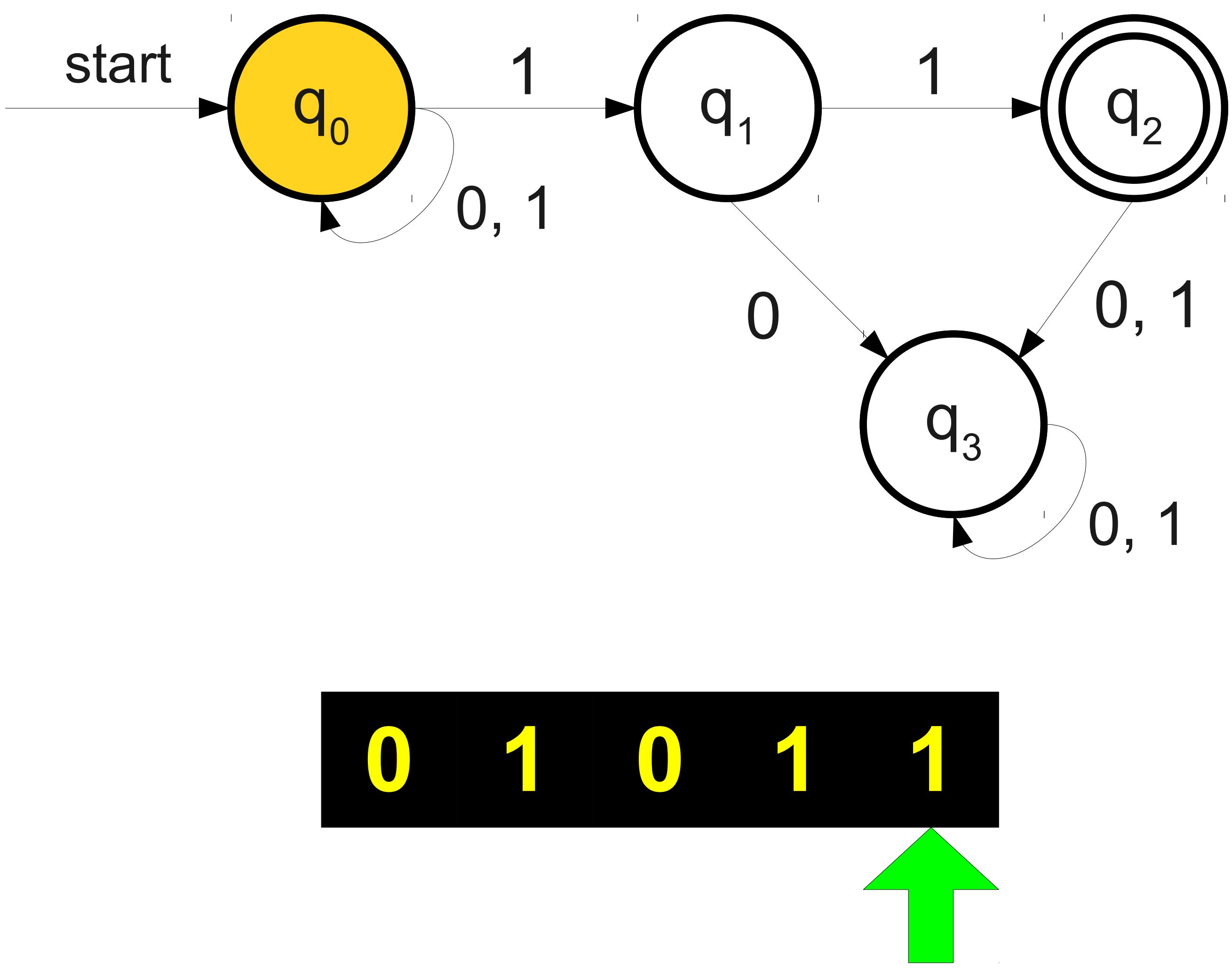
A Simple NFA



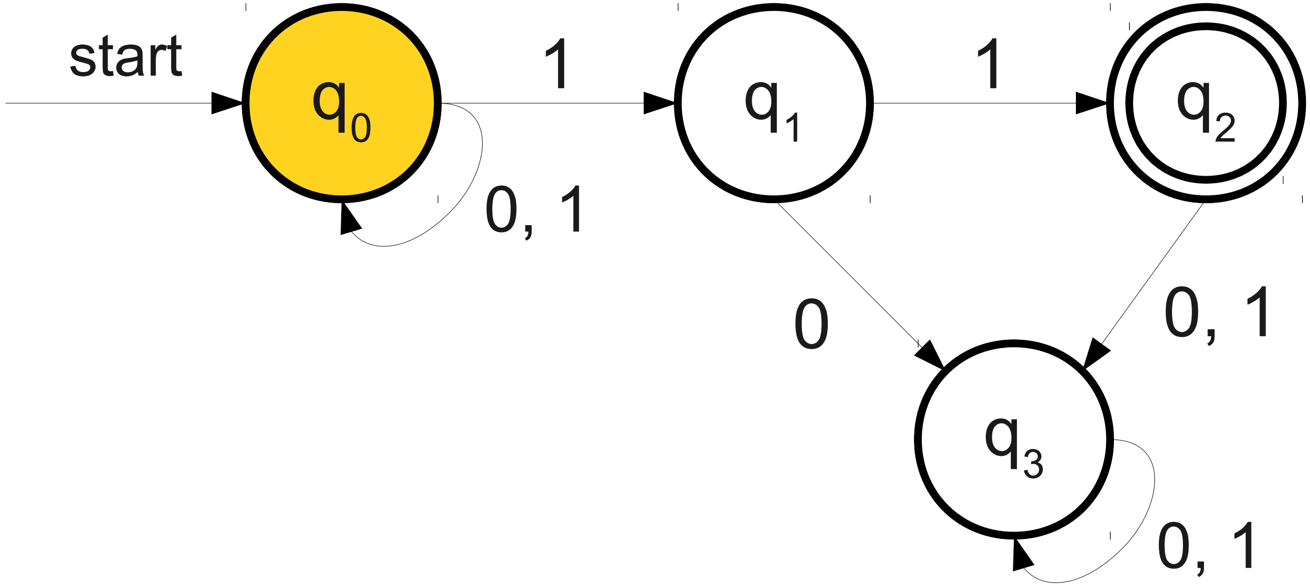
A Simple NFA



A Simple NFA

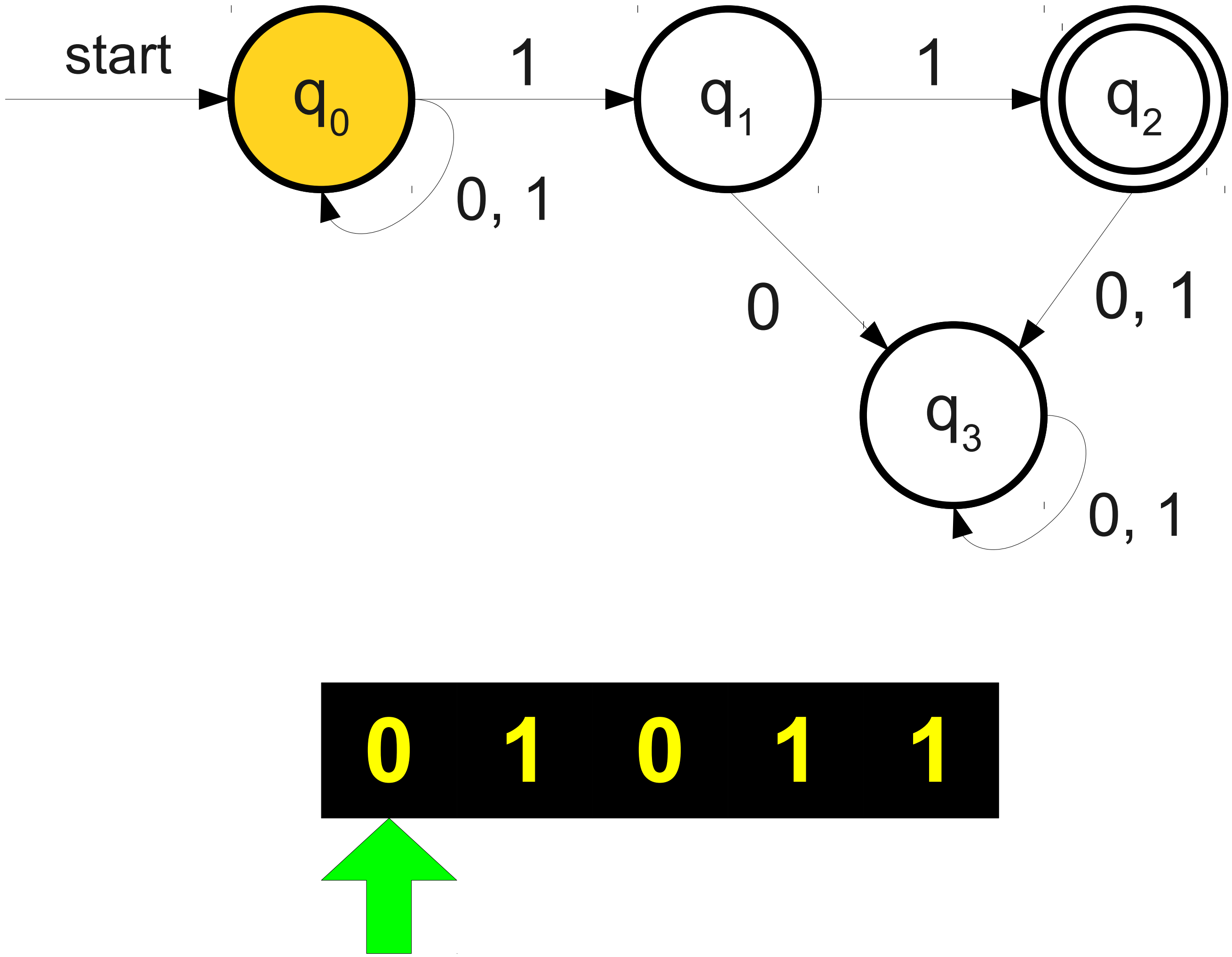


A Simple NFA

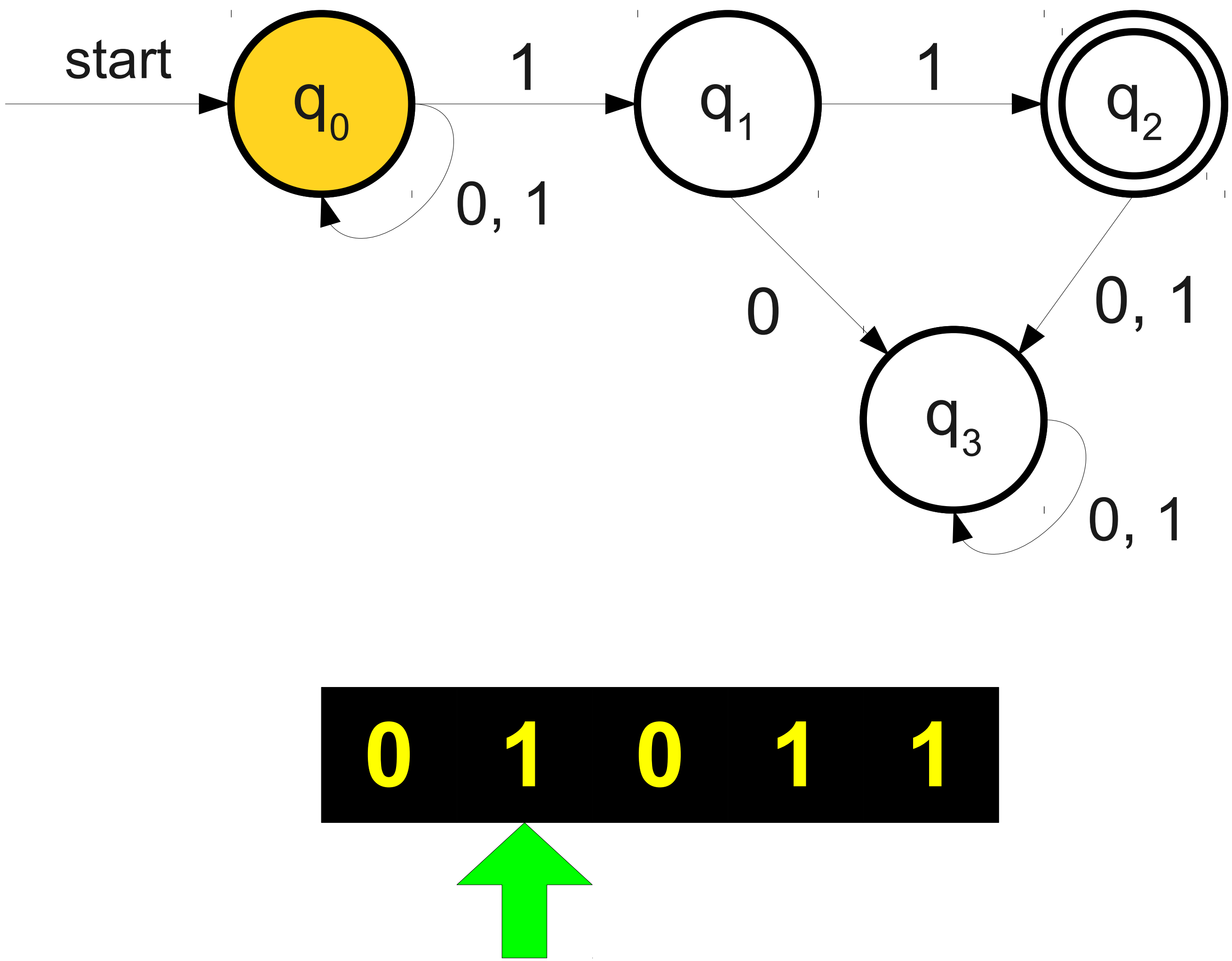


0 1 0 1 1

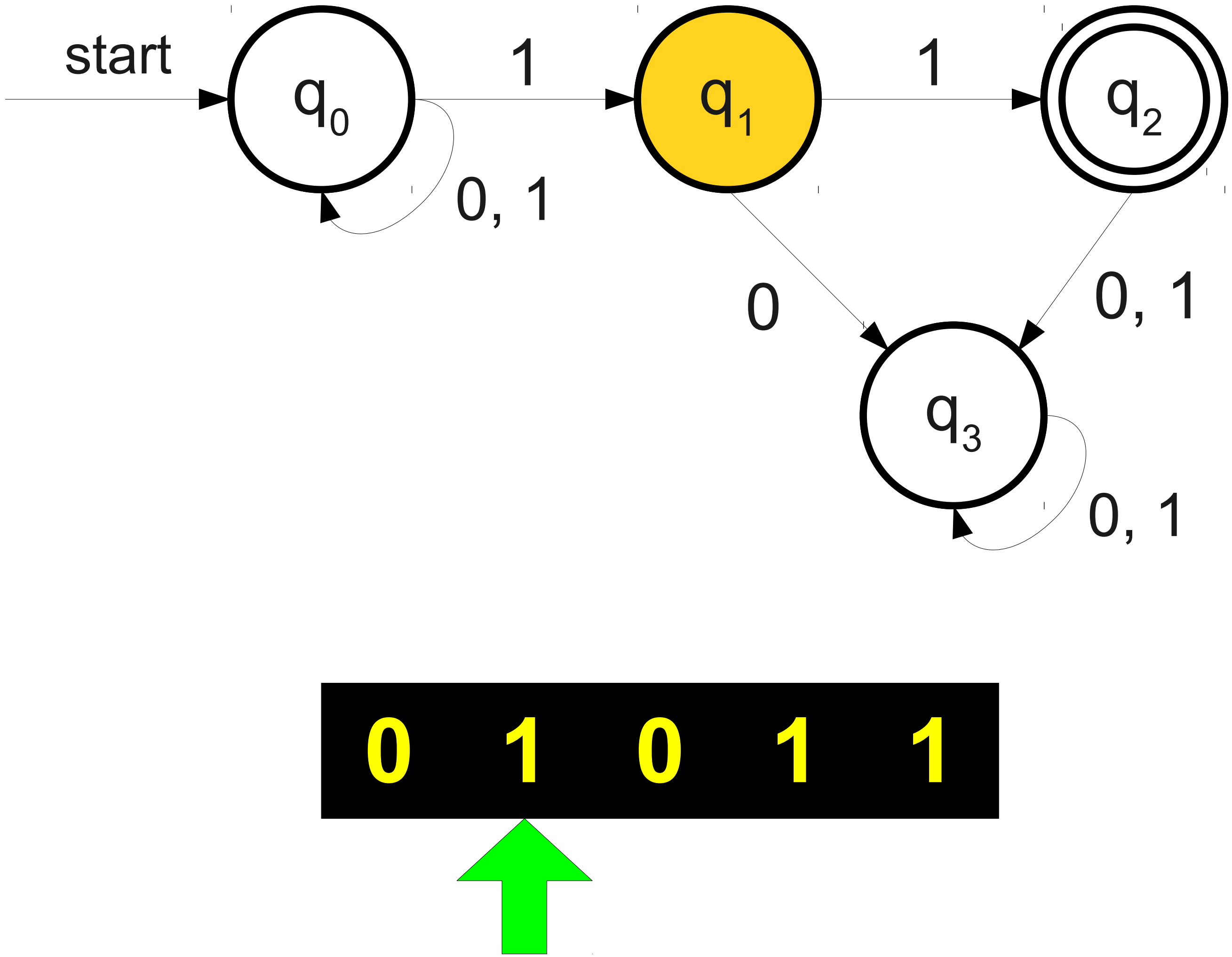
A Simple NFA



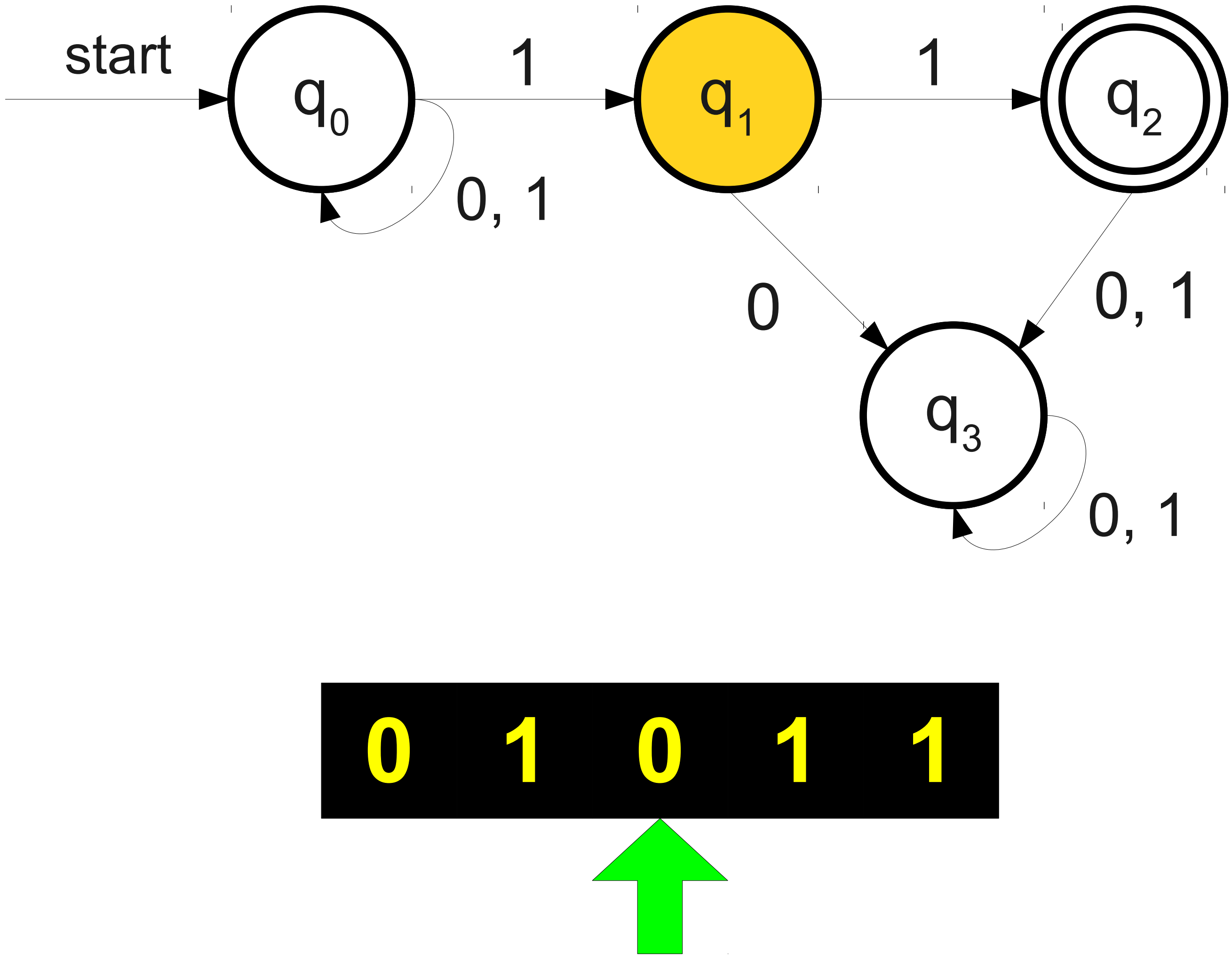
A Simple NFA



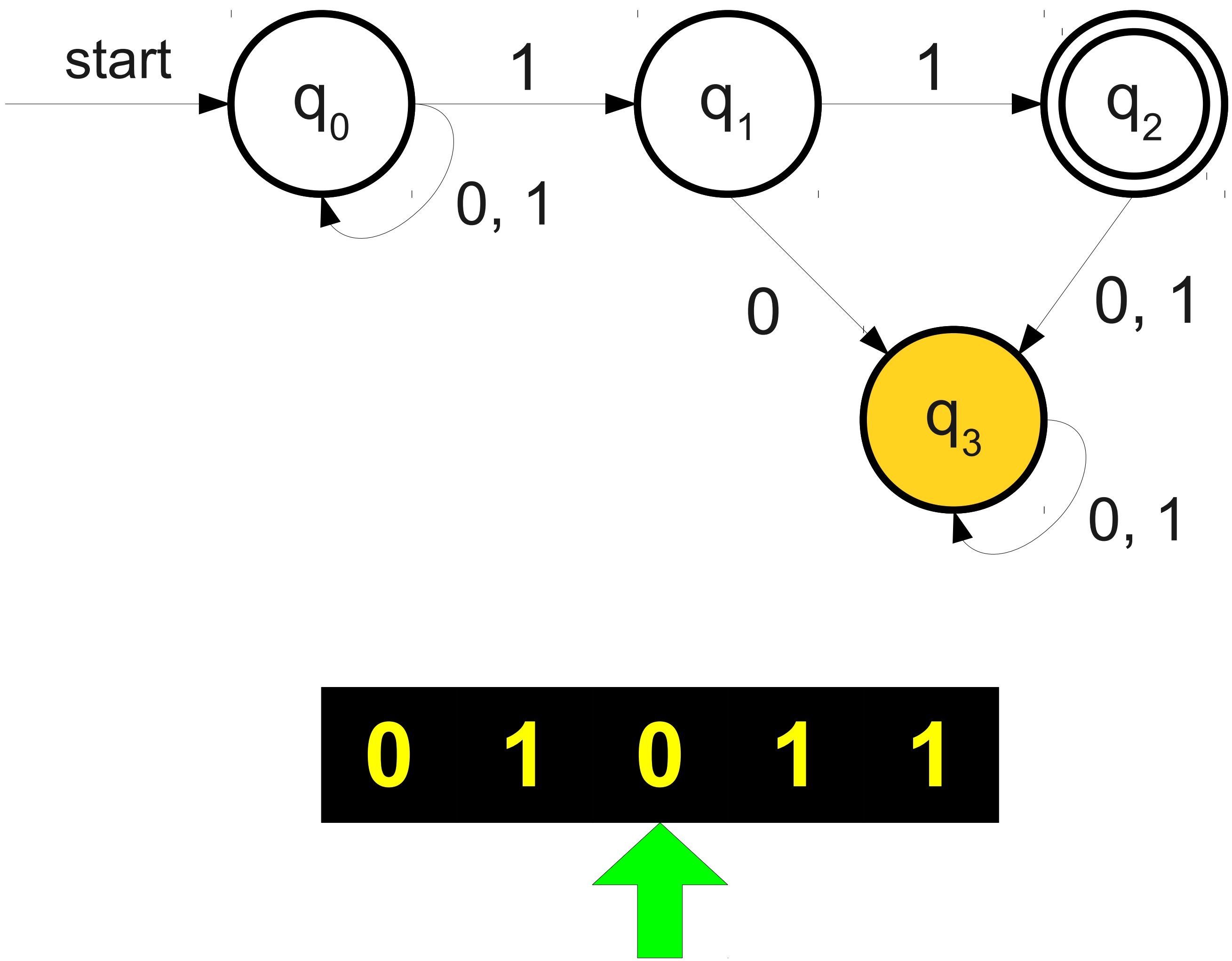
A Simple NFA



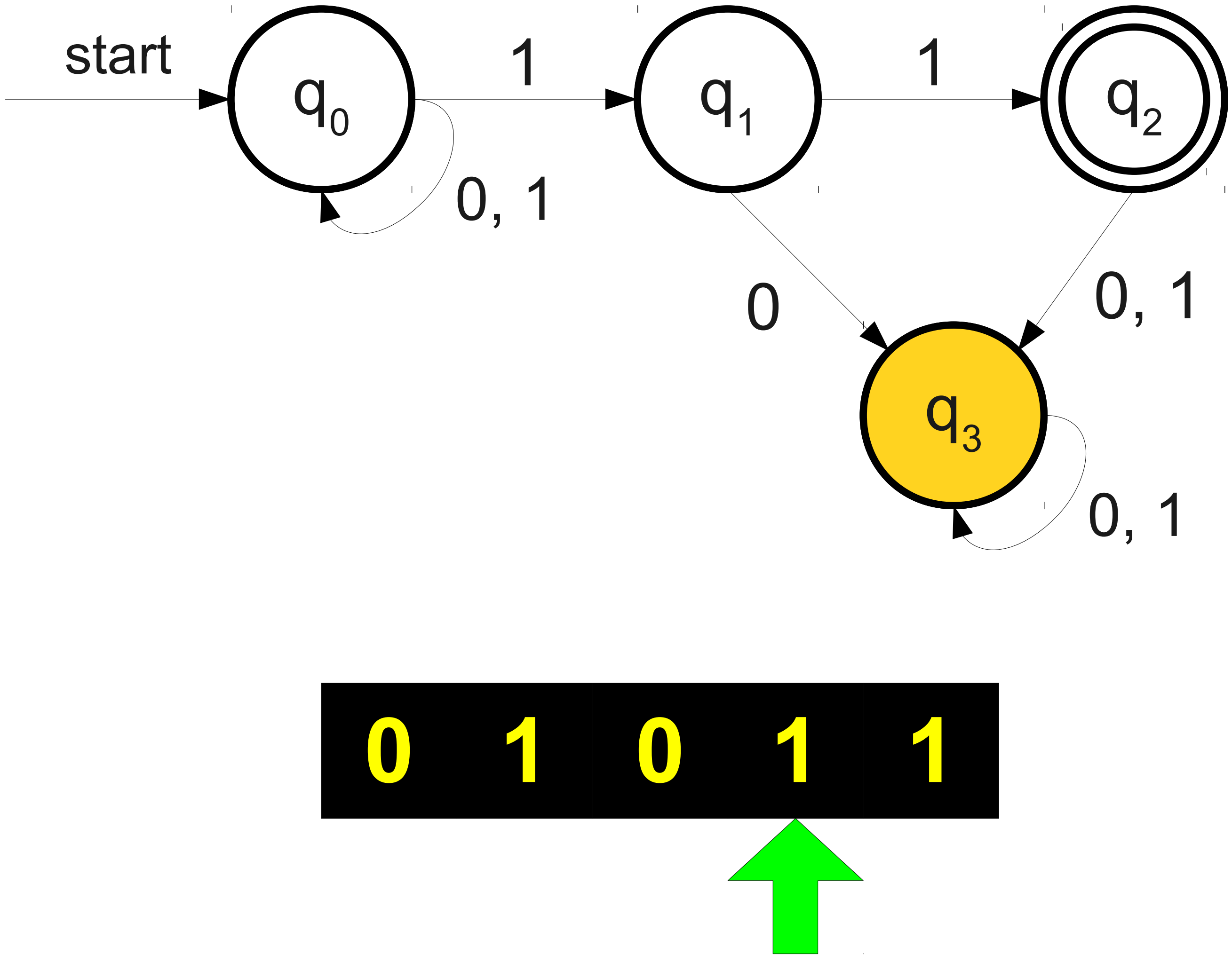
A Simple NFA



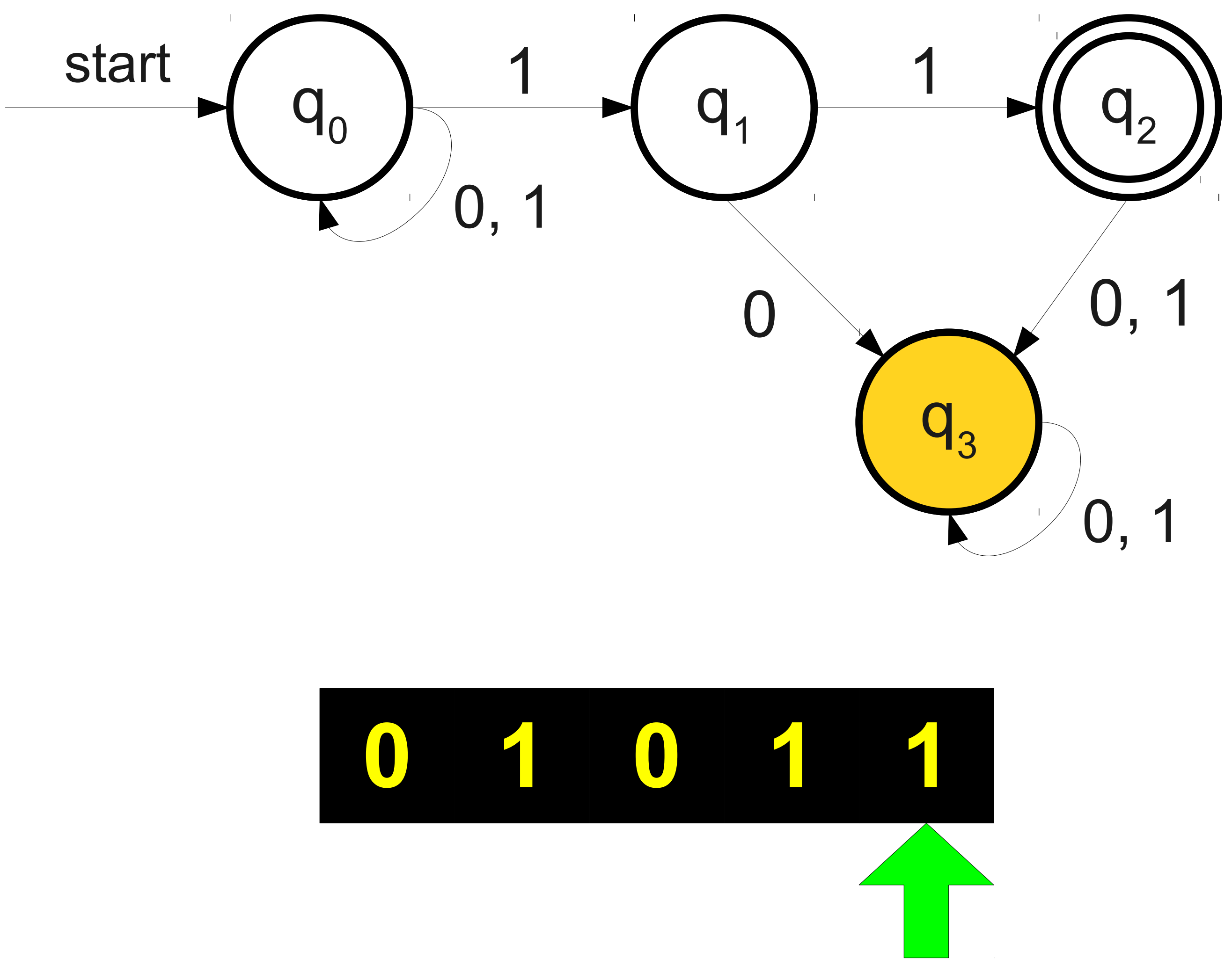
A Simple NFA



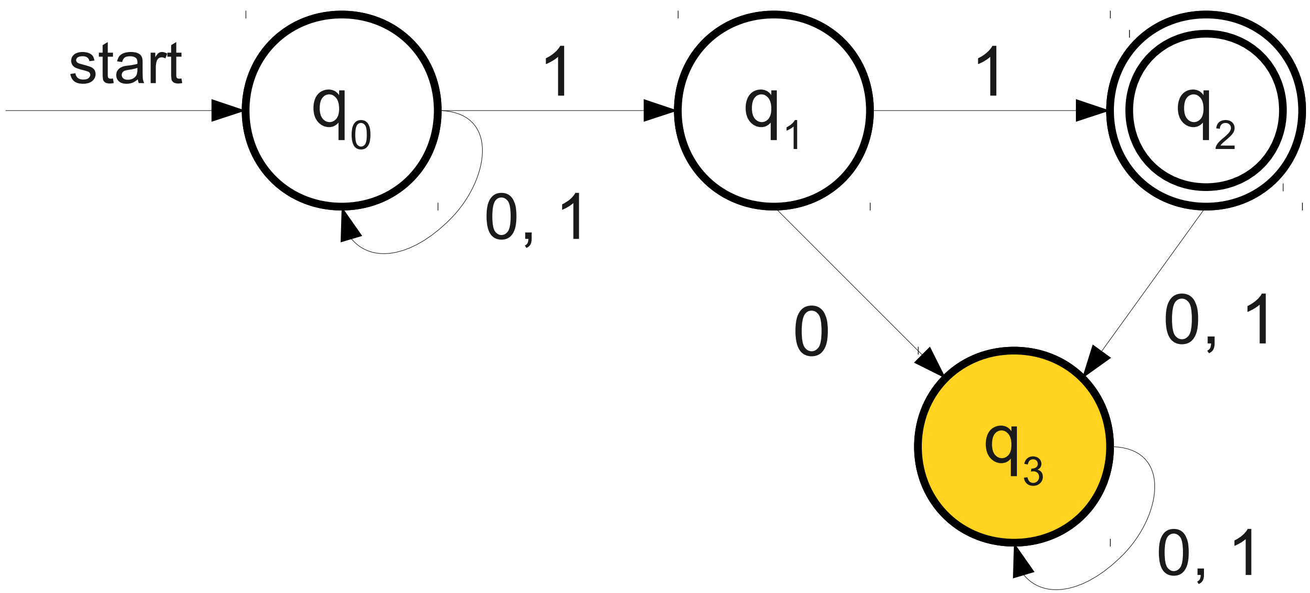
A Simple NFA



A Simple NFA

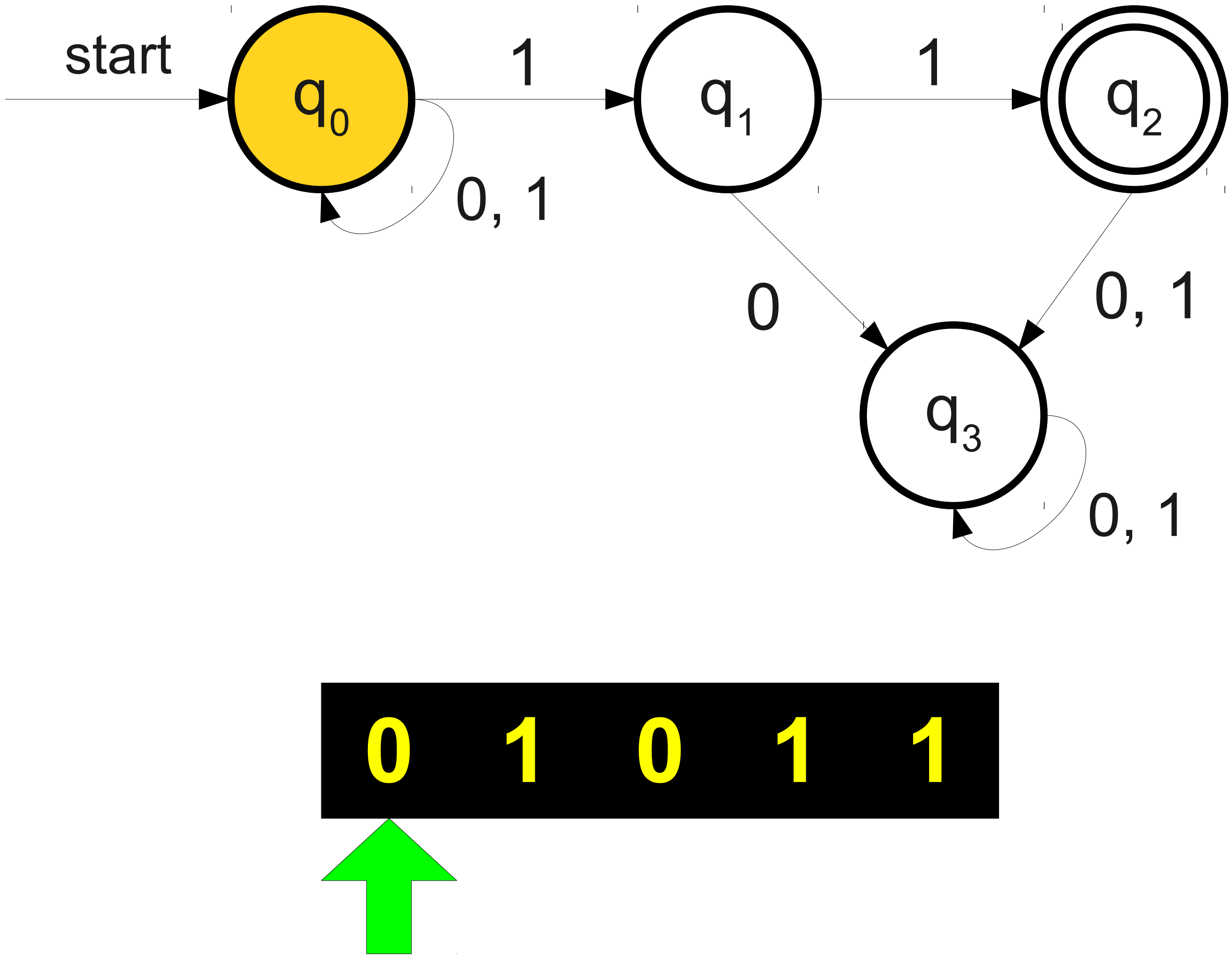


A Simple NFA

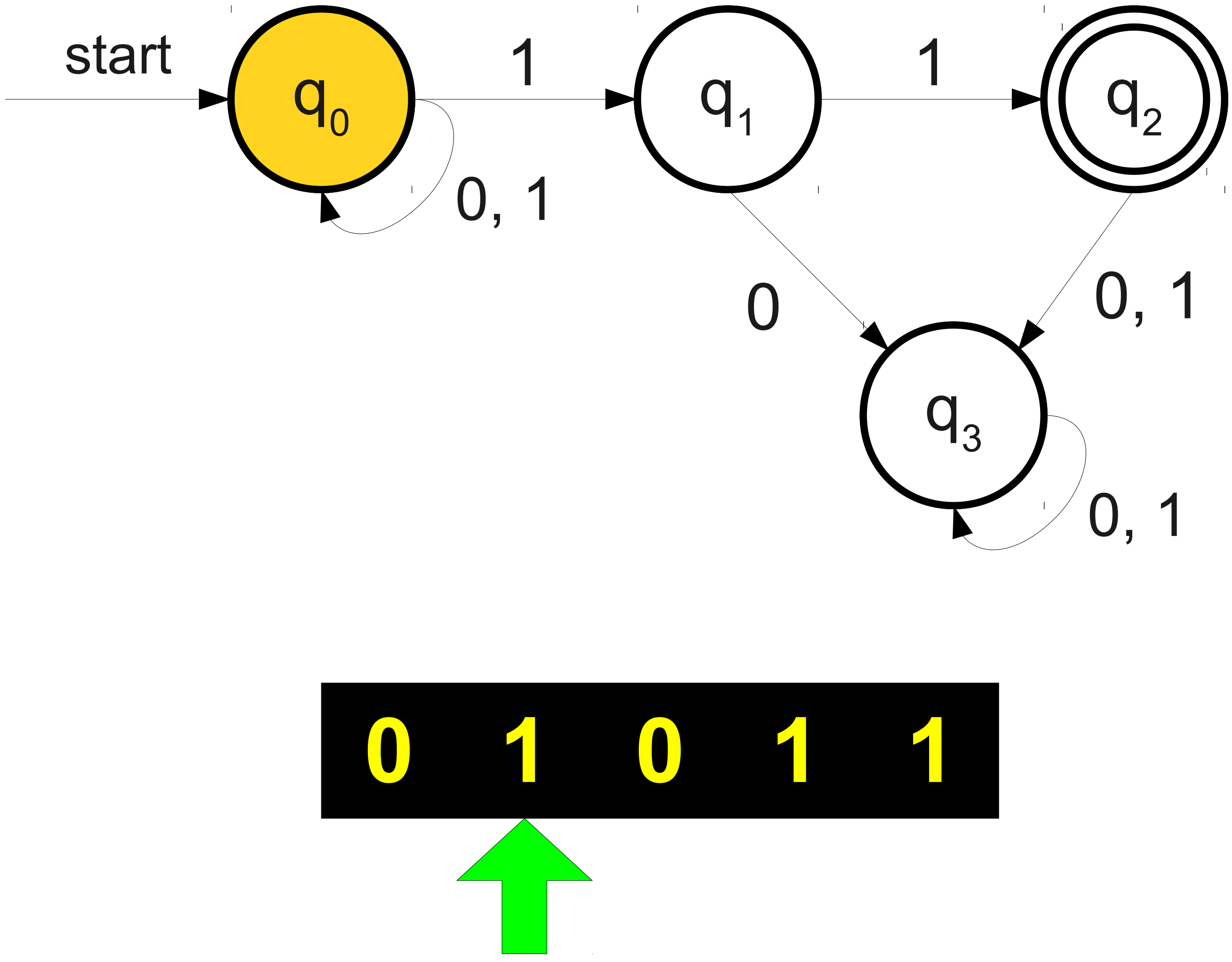


0 1 0 1 1

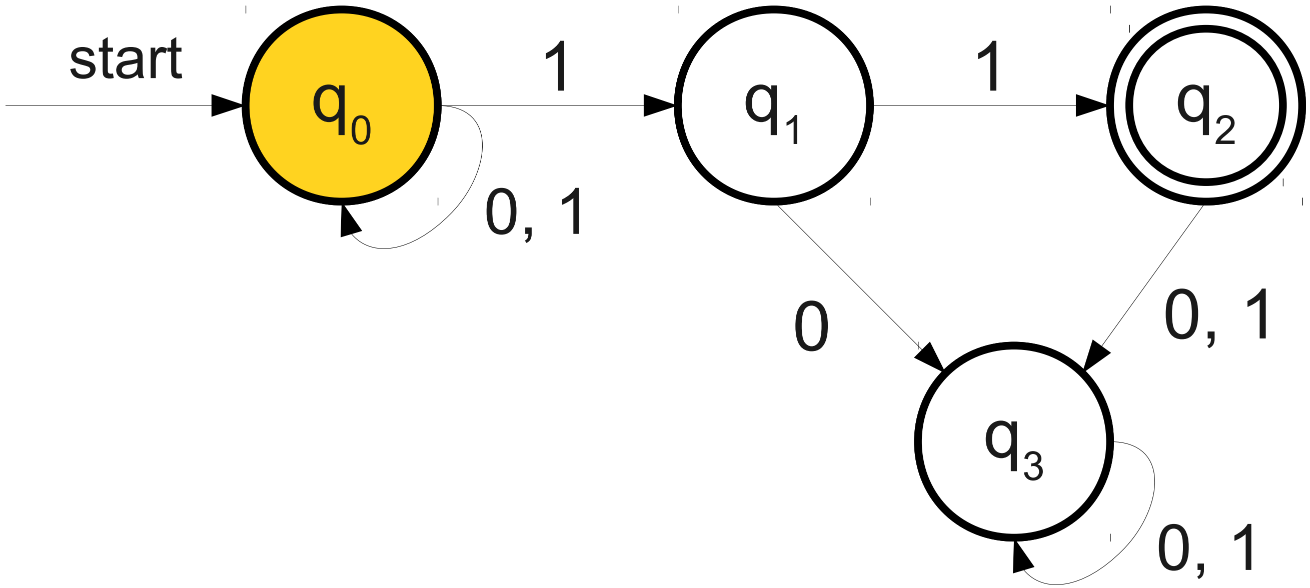
A Simple NFA

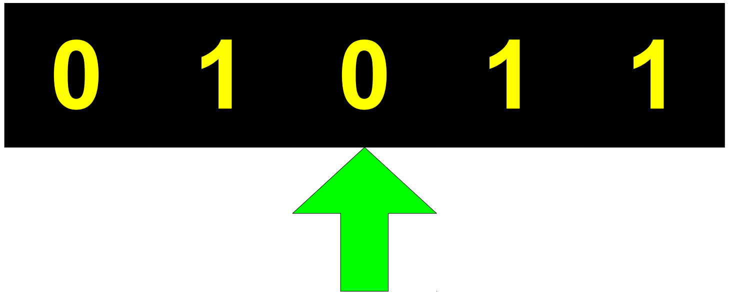


A Simple NFA

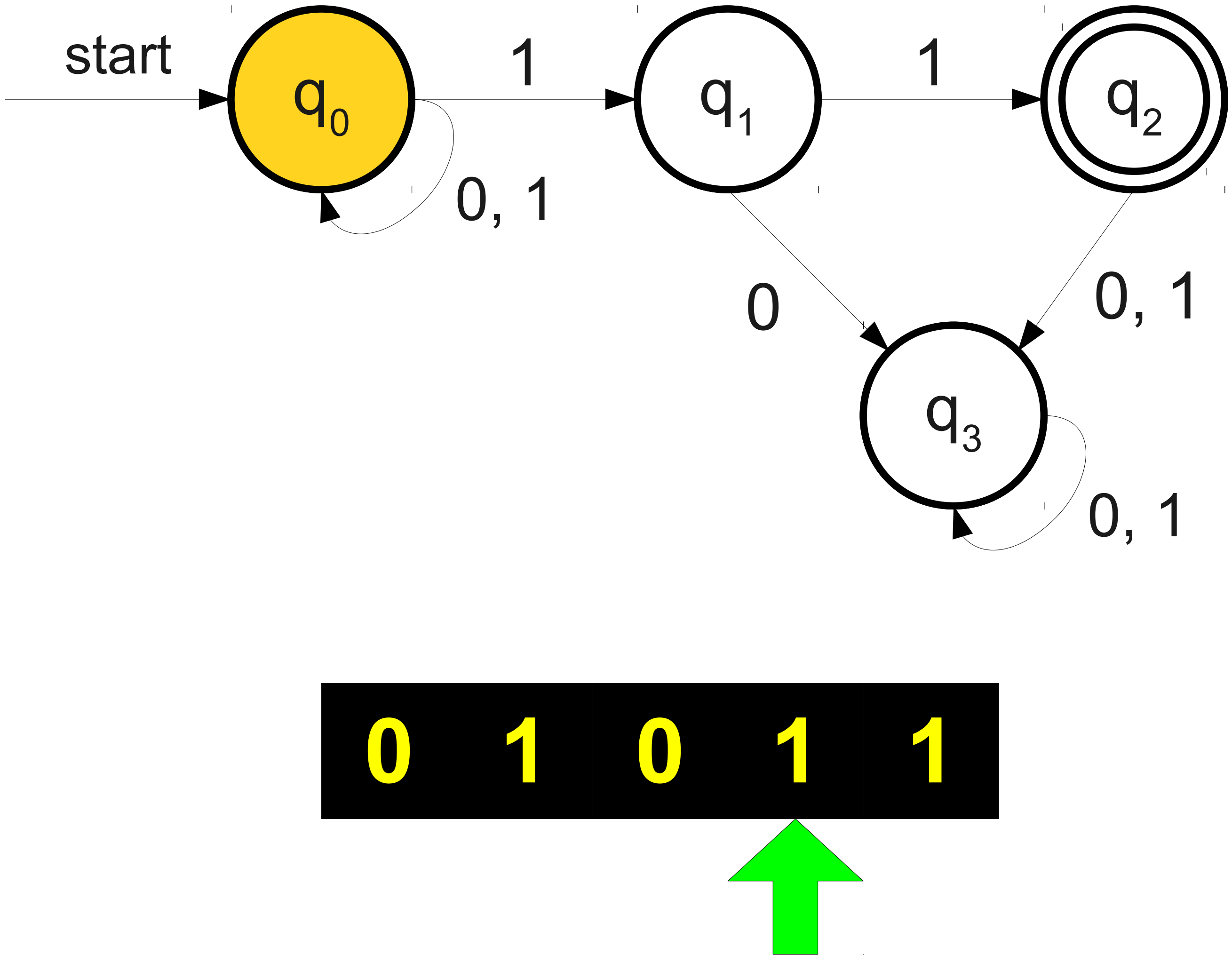


A Simple NFA

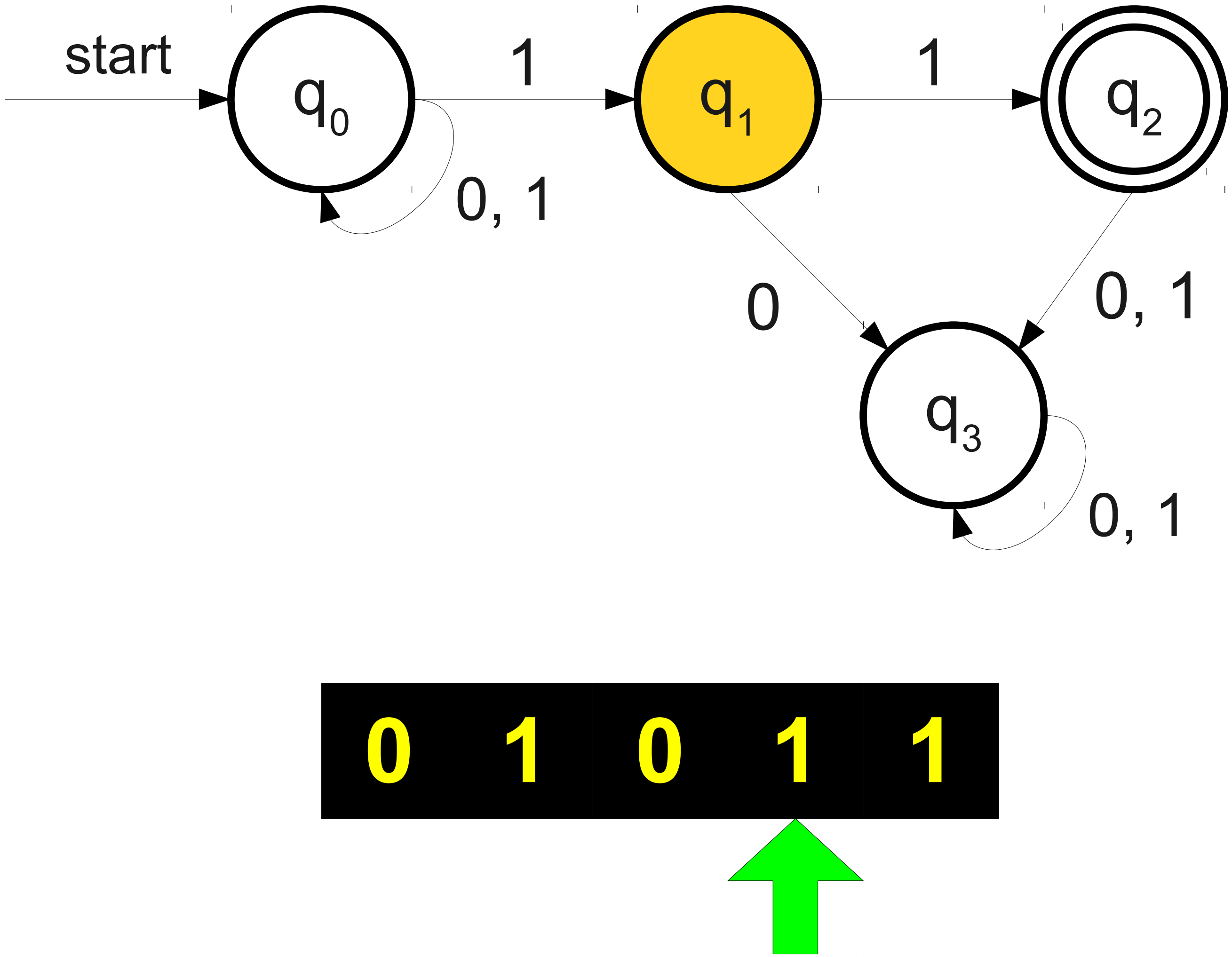




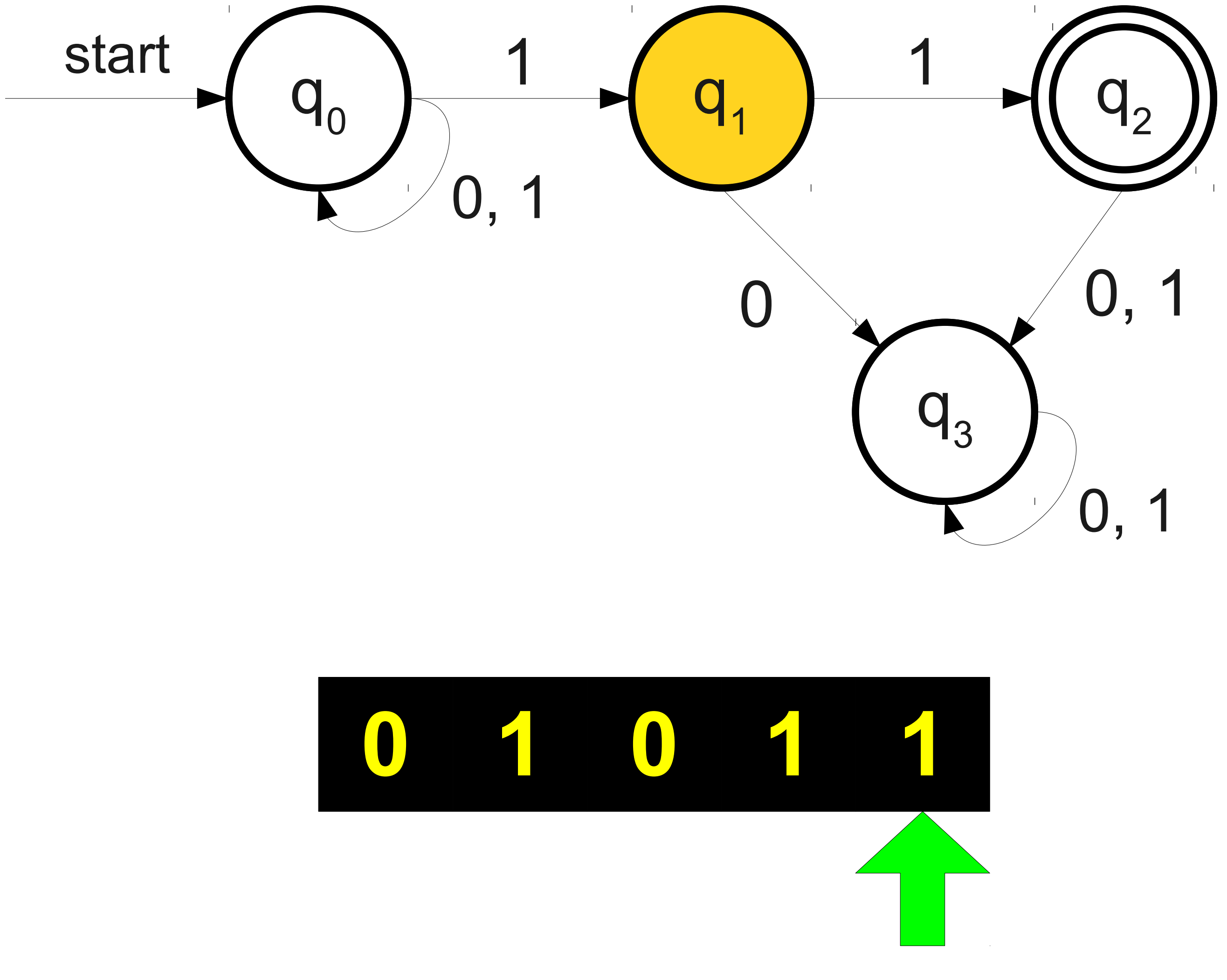
A Simple NFA



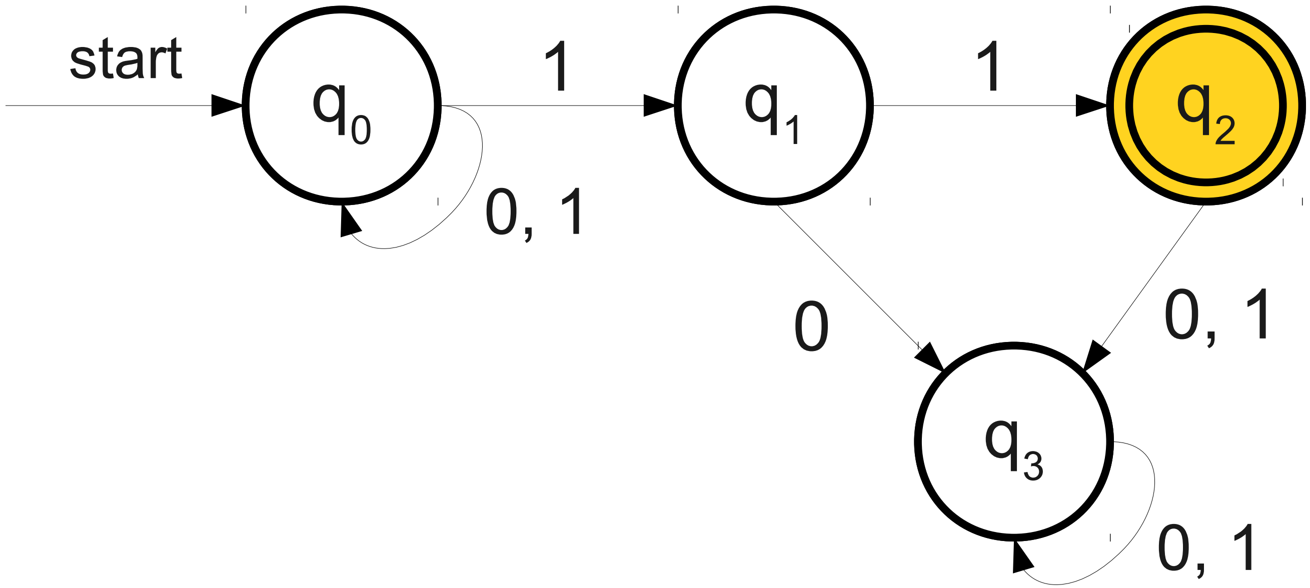
A Simple NFA



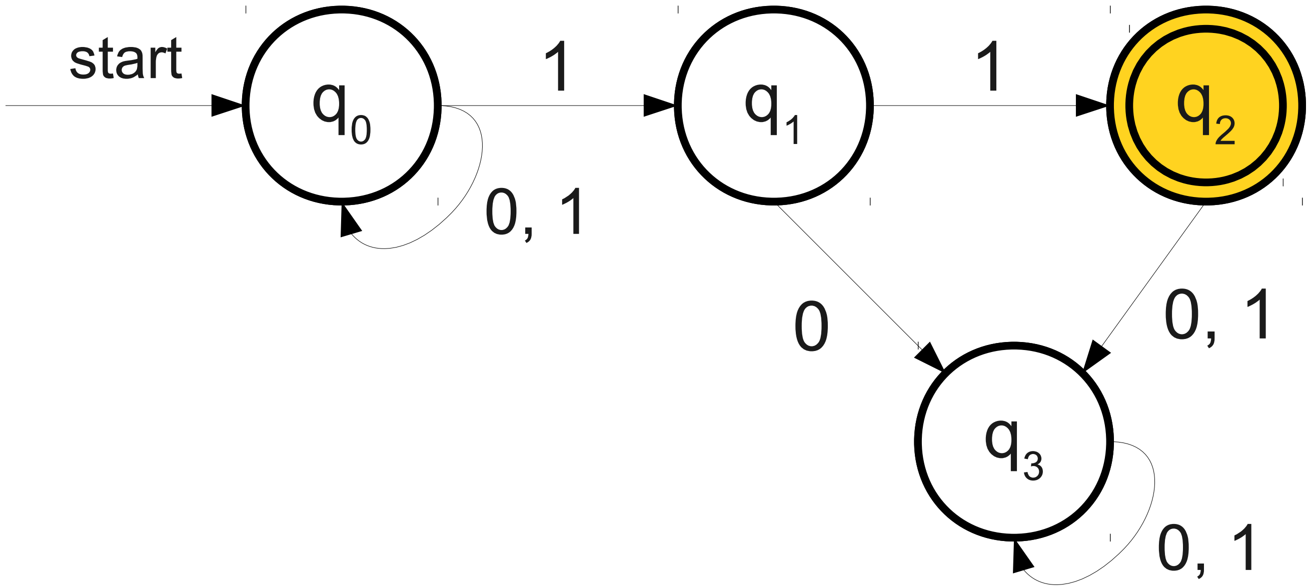
A Simple NFA



A Simple NFA

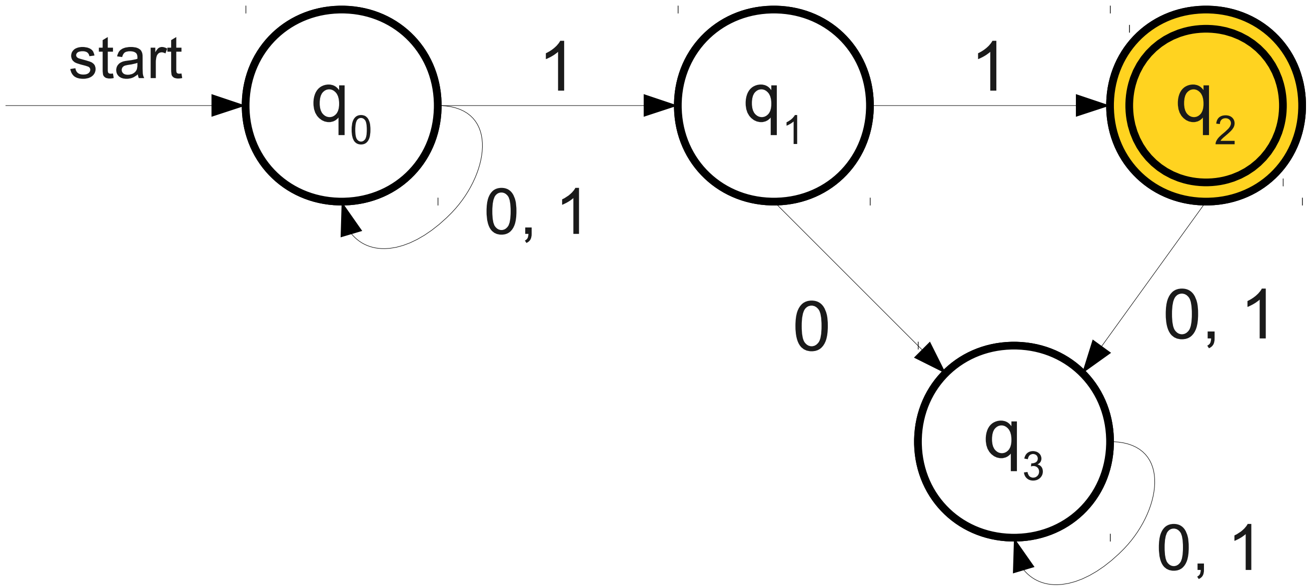


A Simple NFA



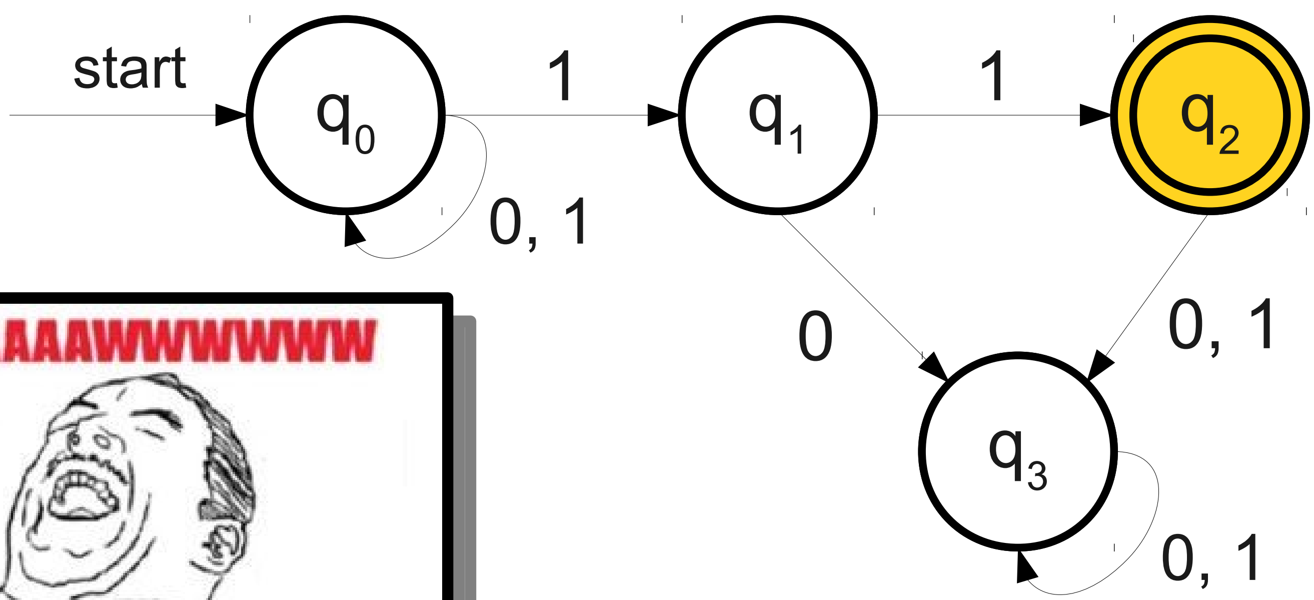
0	1	0	1	1

A Simple NFA



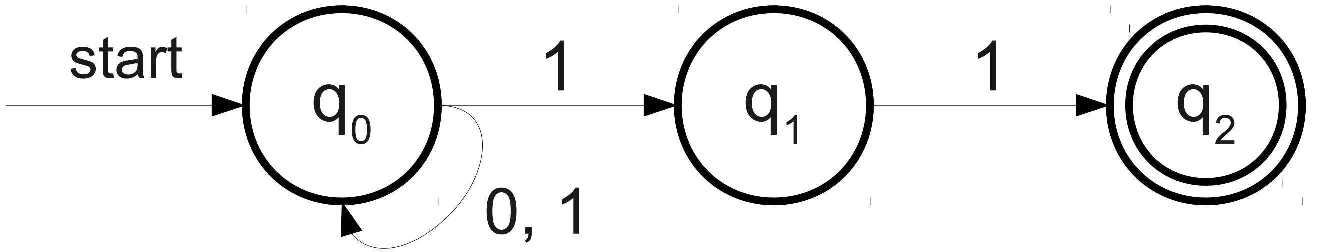
0 1 0 1 1

A Simple NFA

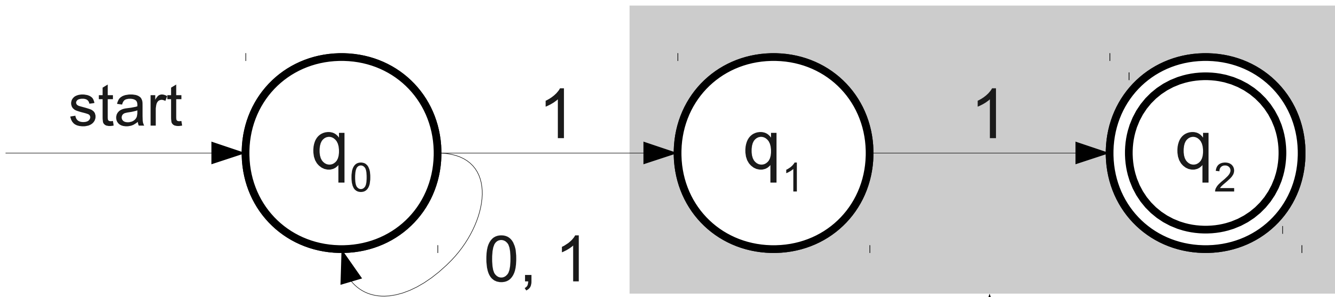


1 0 1 1

A More Complex NFA

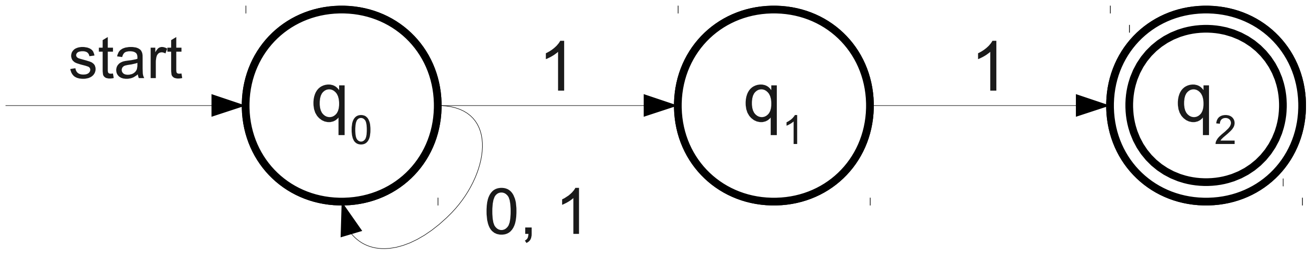


A More Complex NFA



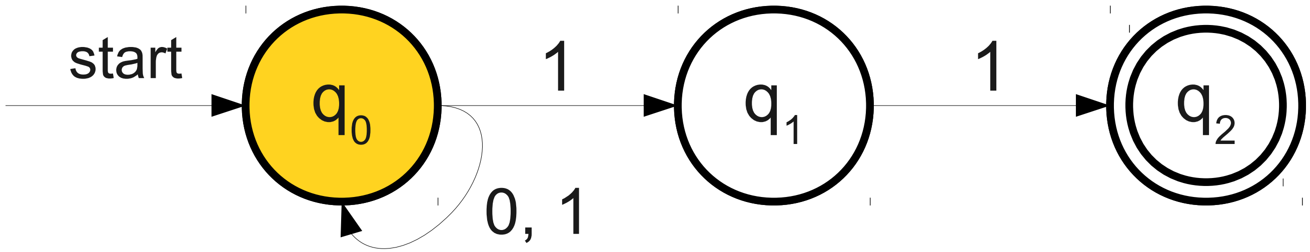
If a NFA needs to make a transition when no transition exists, the automaton dies and that particular path rejects.

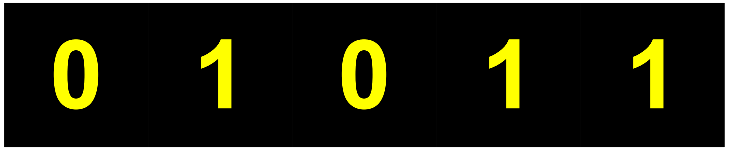
A More Complex NFA



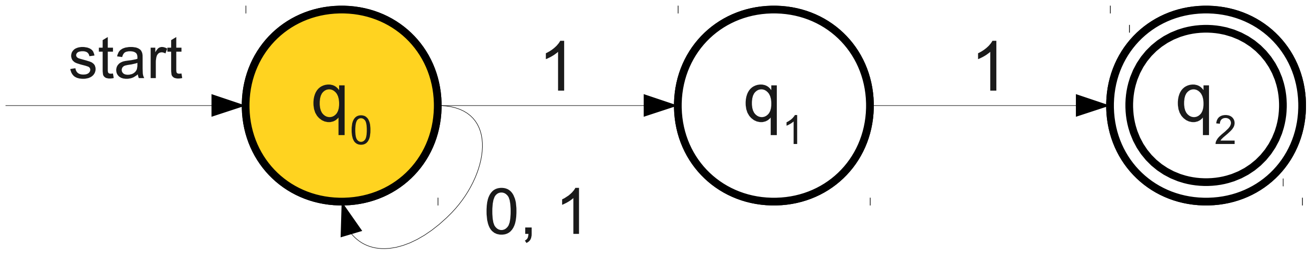
0 1 0 1 1

A More Complex NFA

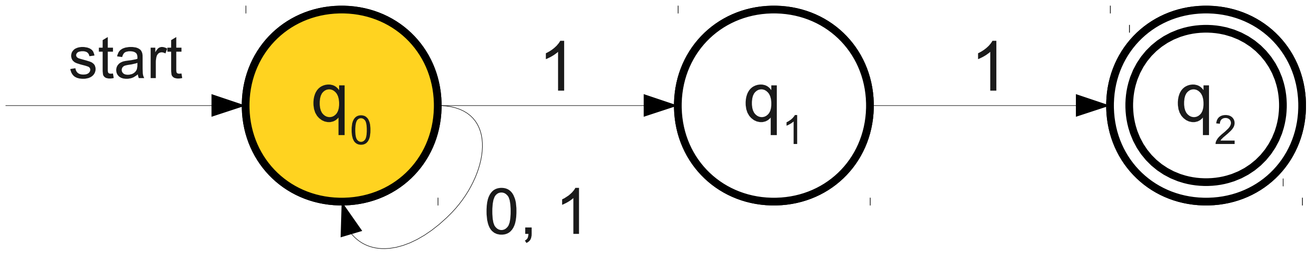


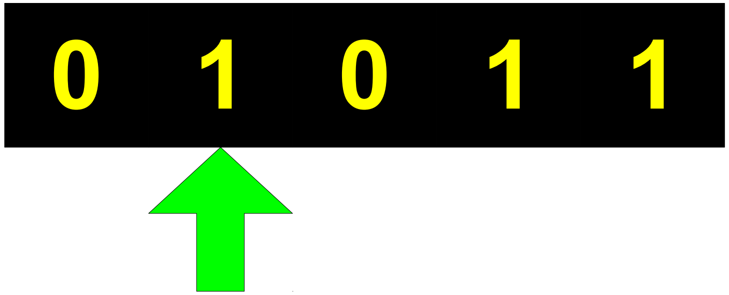


A More Complex NFA

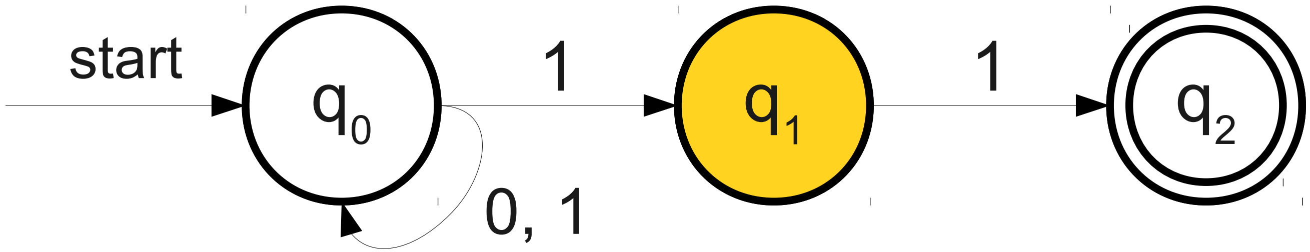


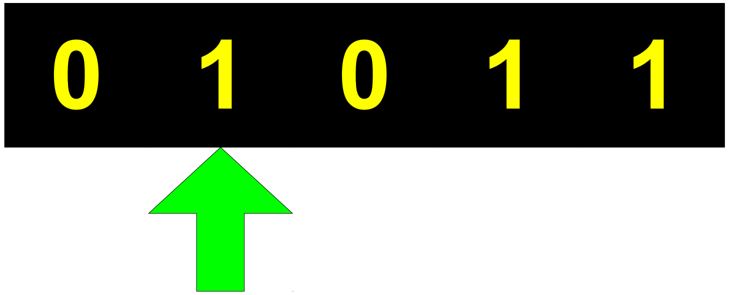
A More Complex NFA



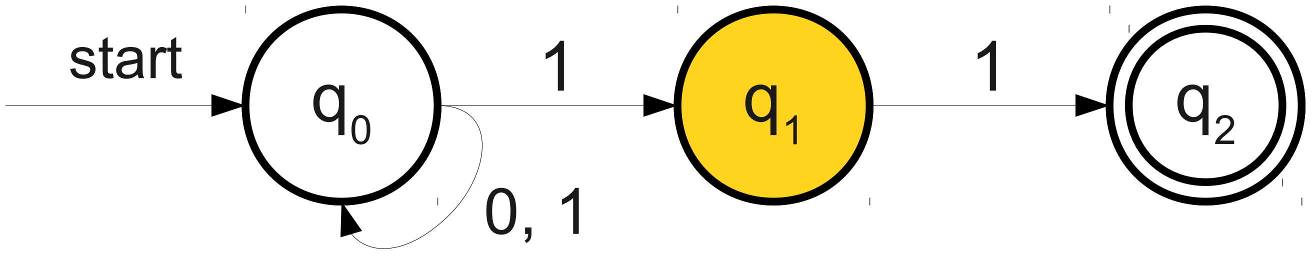


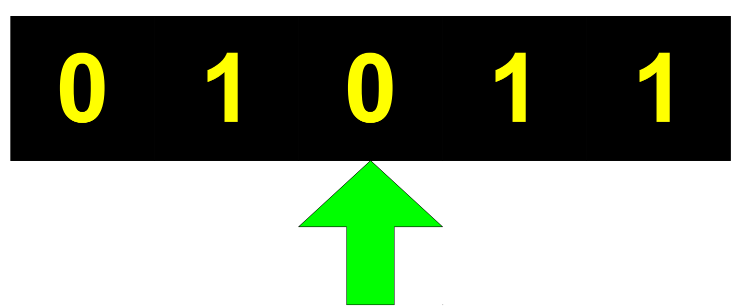
A More Complex NFA



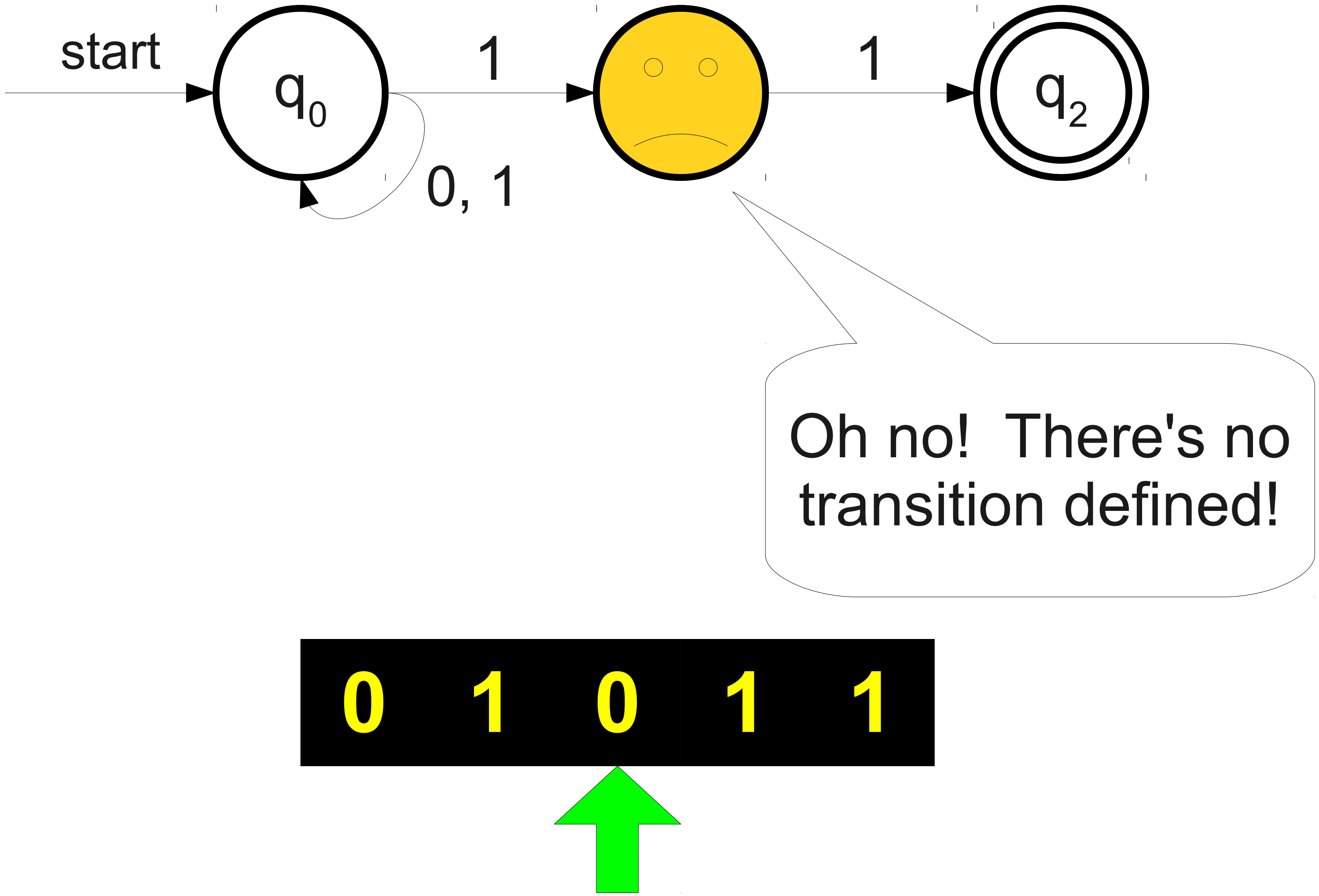


A More Complex NFA

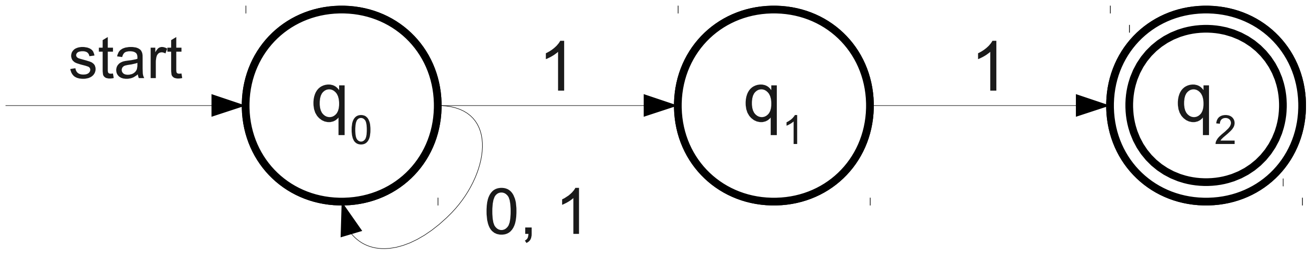


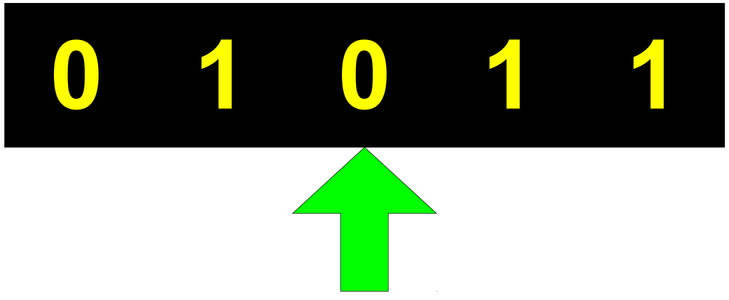


A More Complex NFA

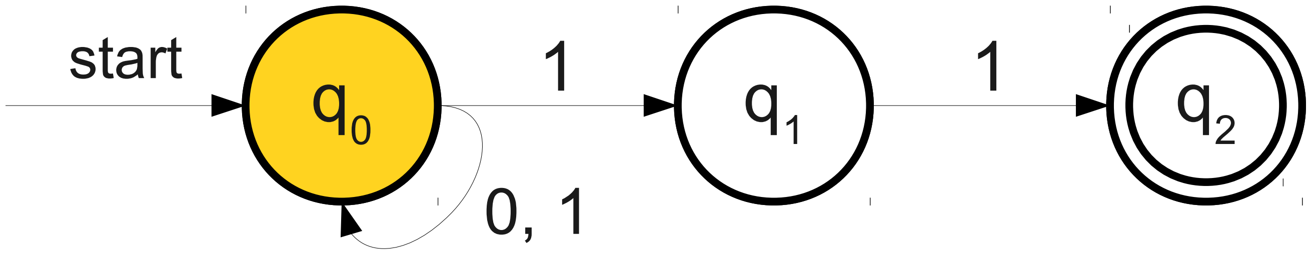


A More Complex NFA

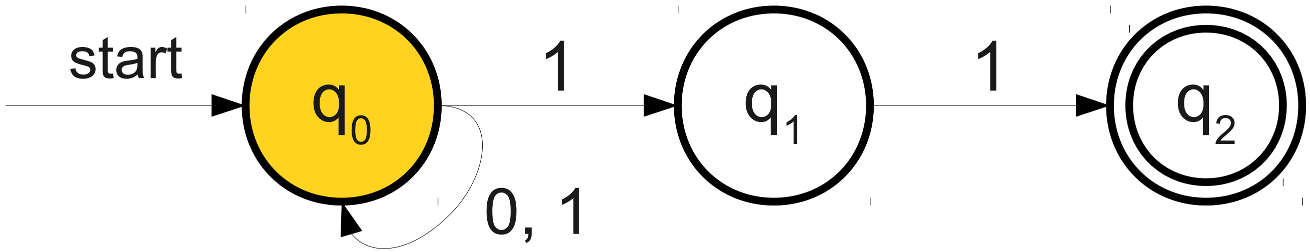


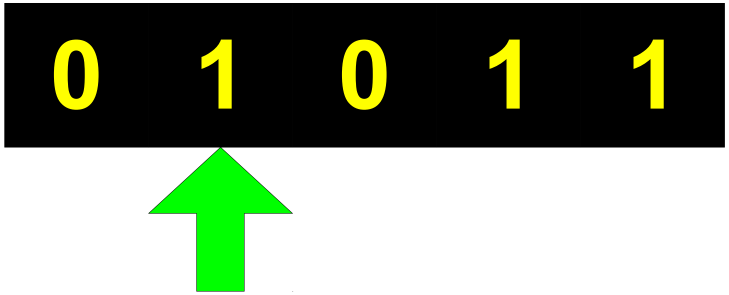


A More Complex NFA

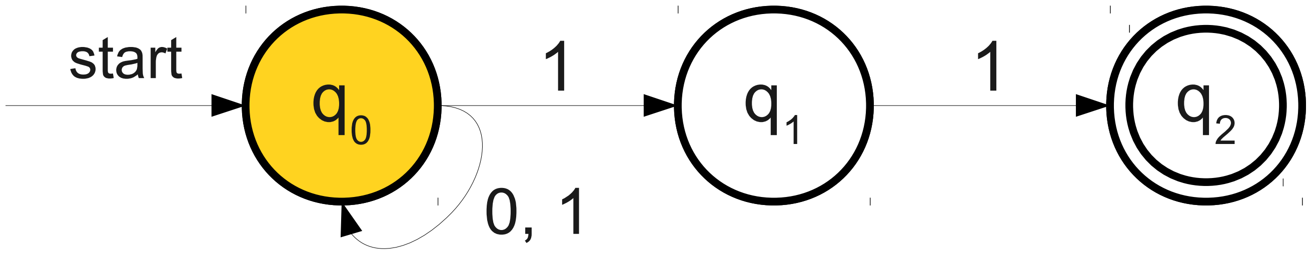


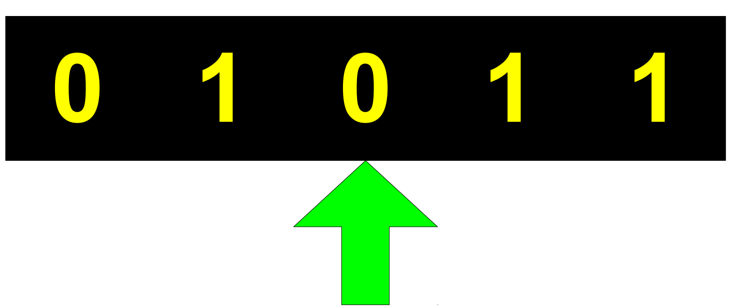
A More Complex NFA



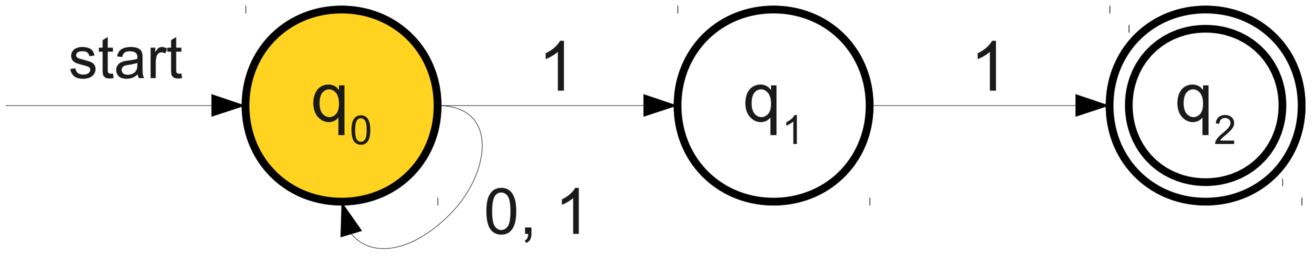


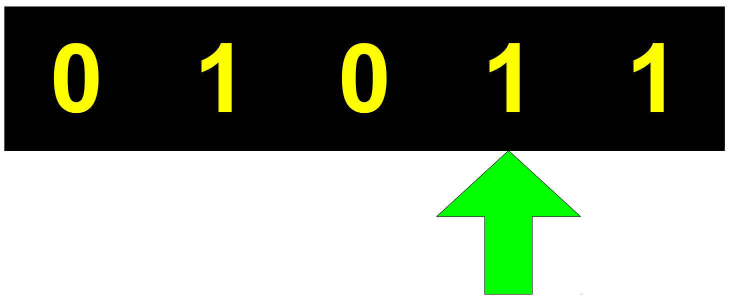
A More Complex NFA



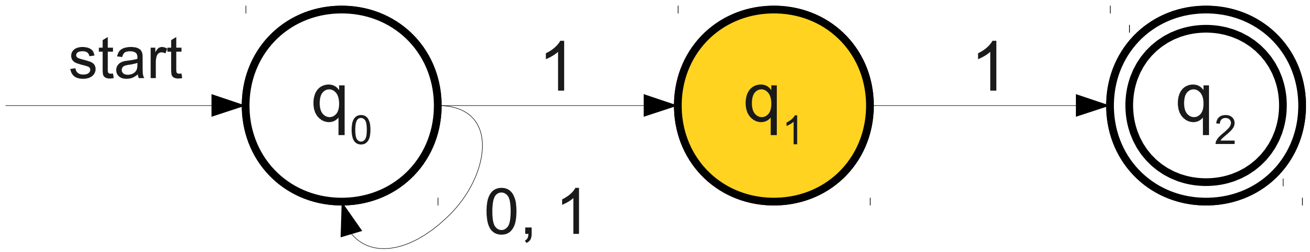


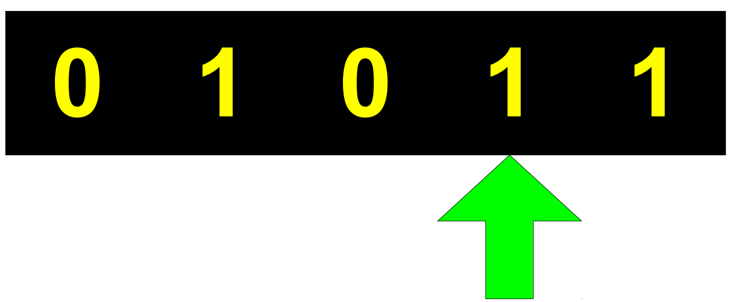
A More Complex NFA



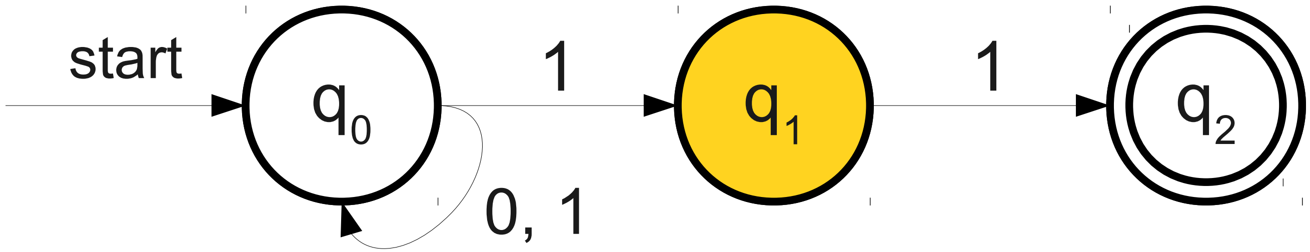


A More Complex NFA

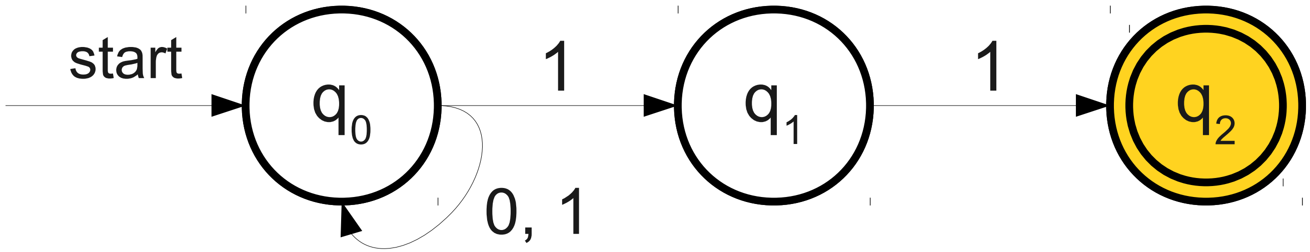




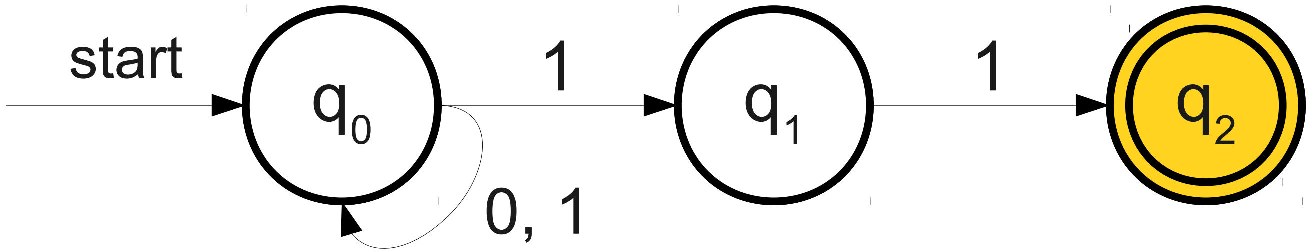
A More Complex NFA



A More Complex NFA

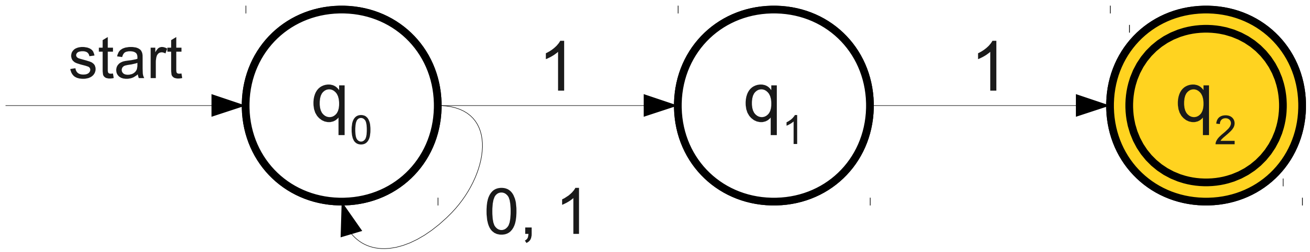


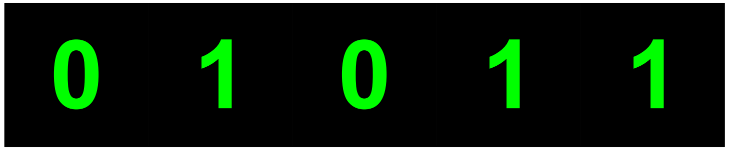
A More Complex NFA



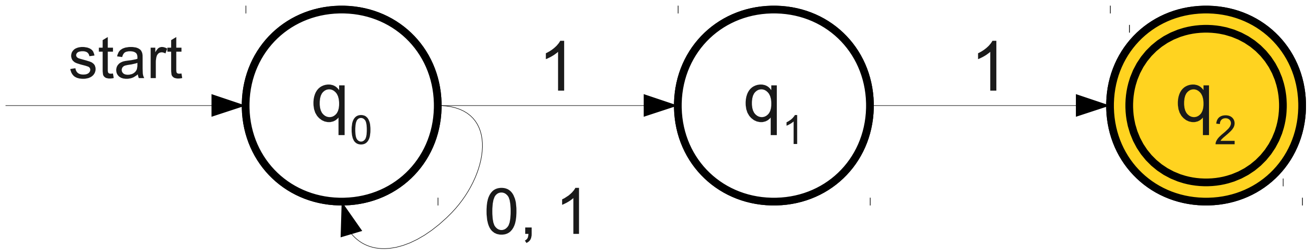
0 1 0 1 1

A More Complex NFA





A More Complex NFA

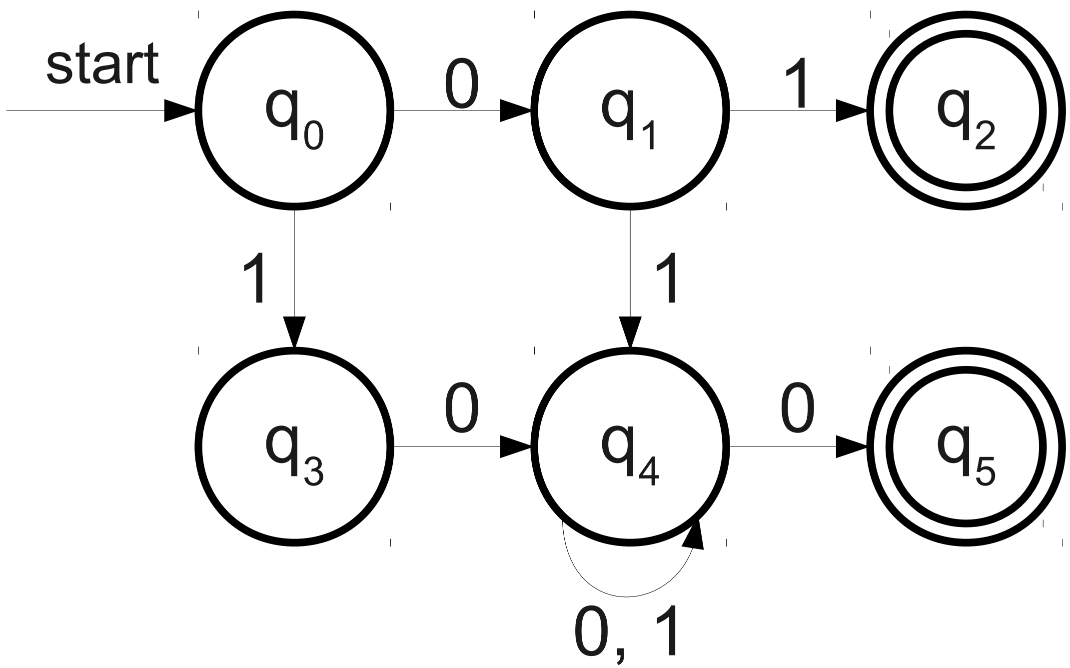


1 0 1 1

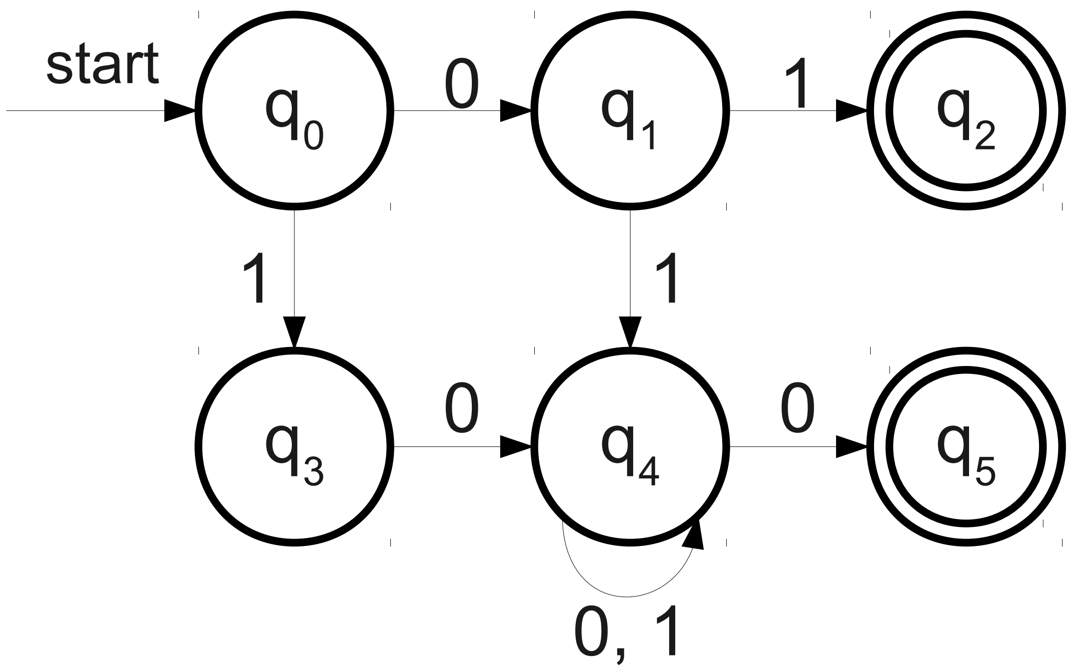
Intuiting Nondeterminism

- 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

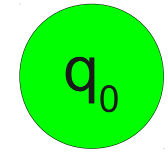


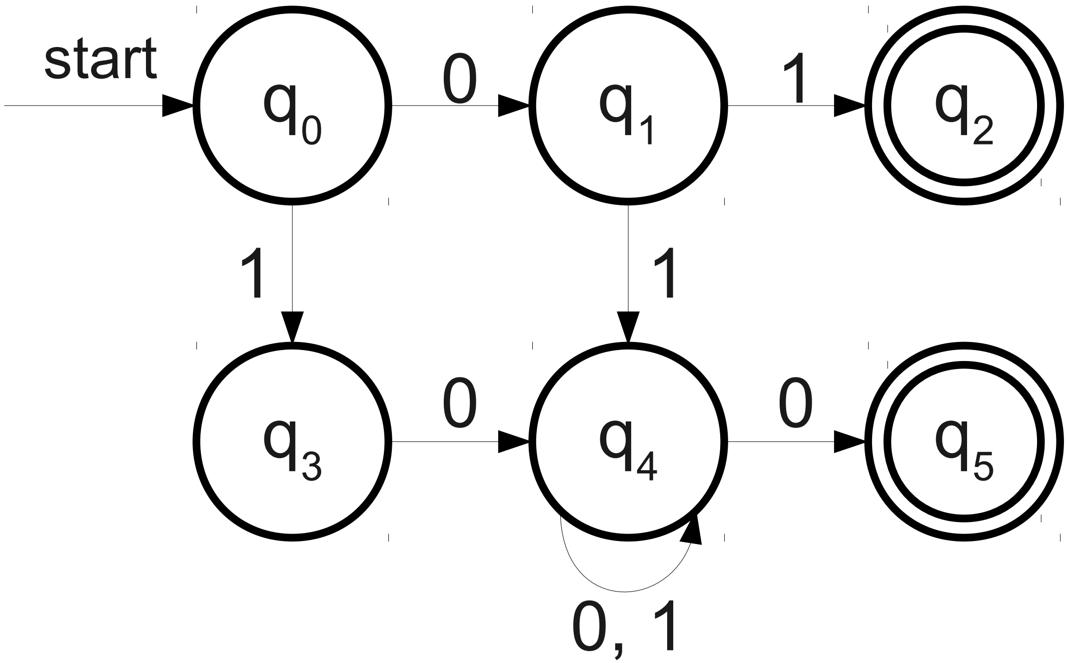
Tree Computation

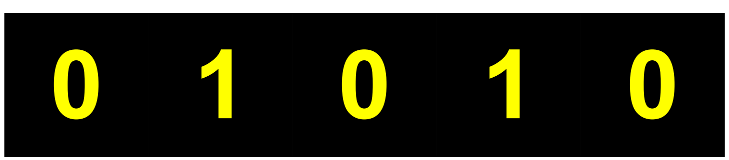


0	**1**	**0**	**1**	**0**

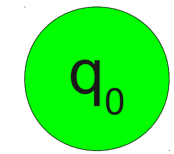
Tree Computation

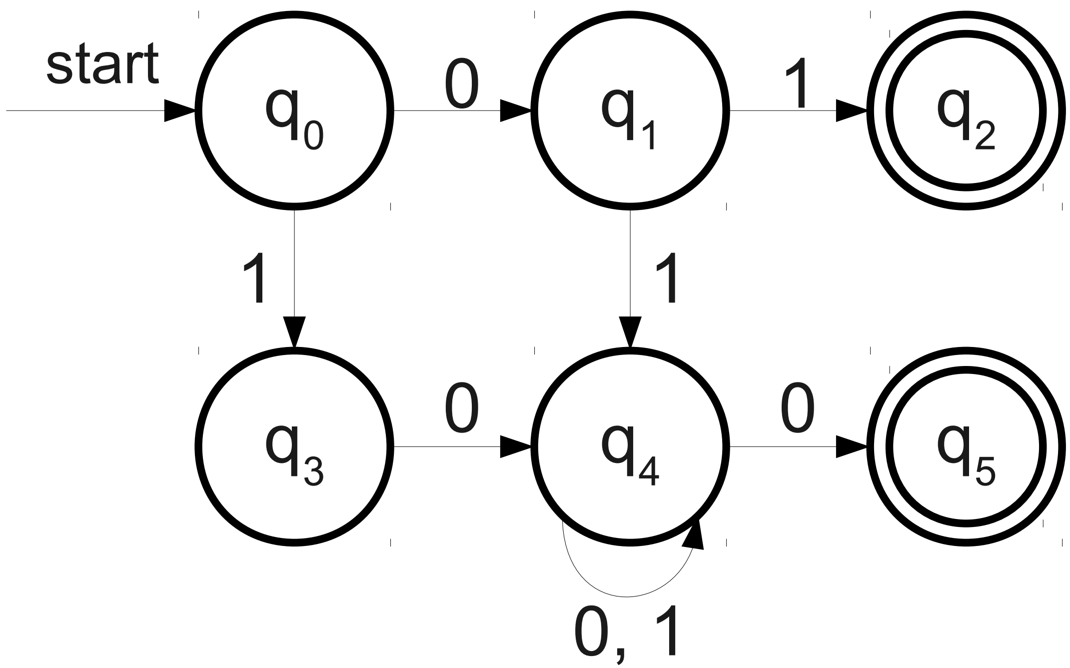




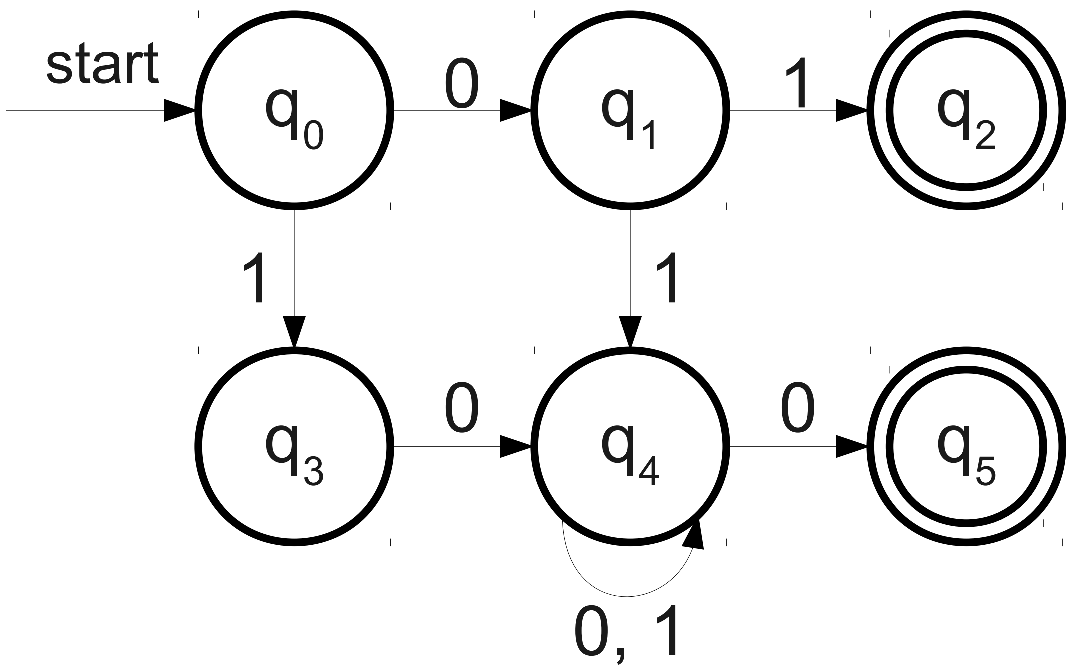


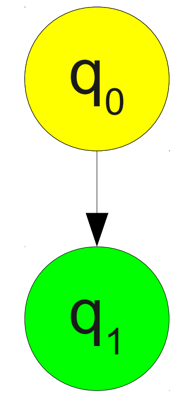
Tree Computation



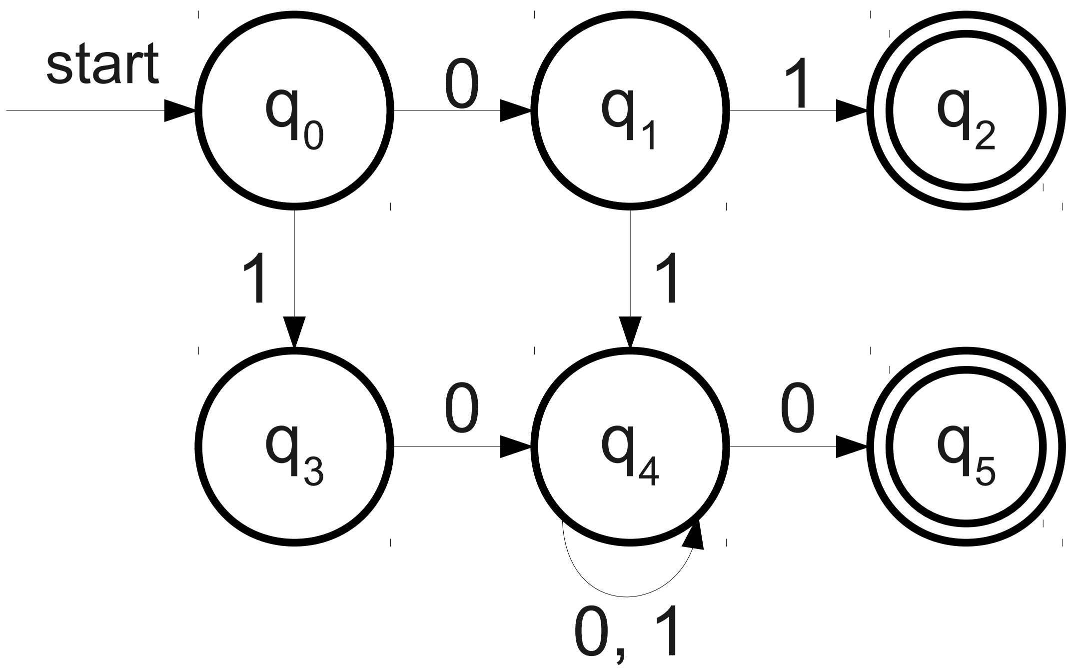


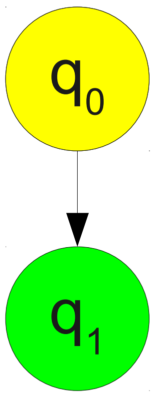
Tree Computation

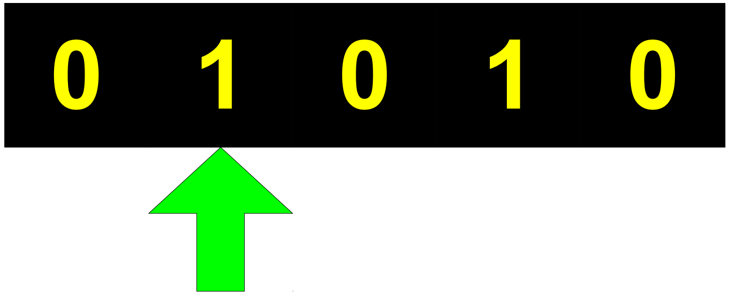




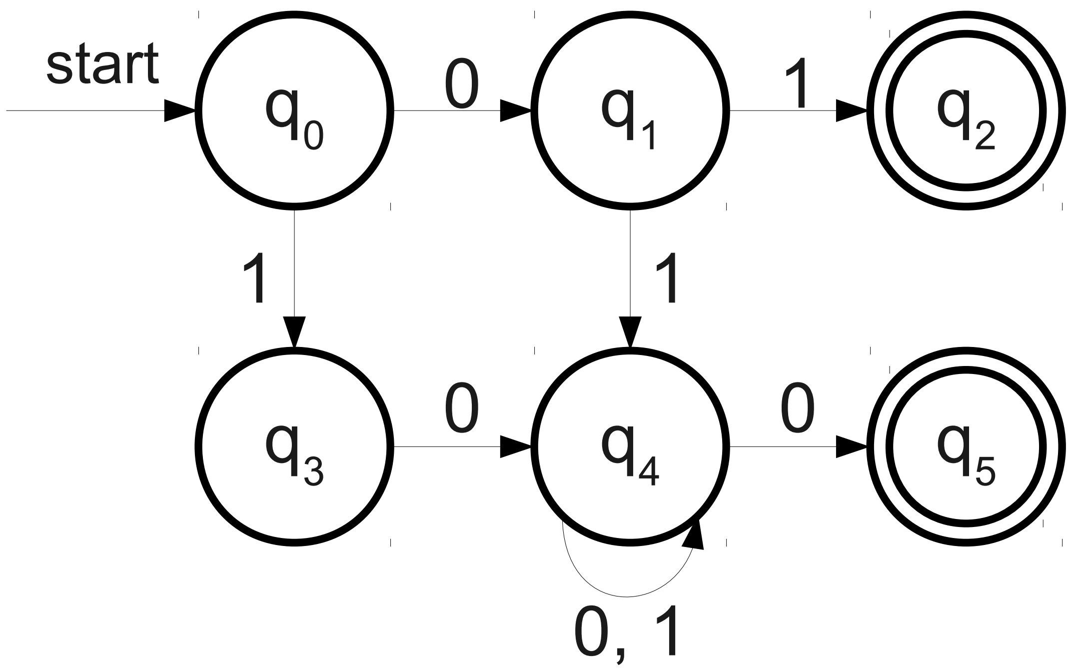
Tree Computation

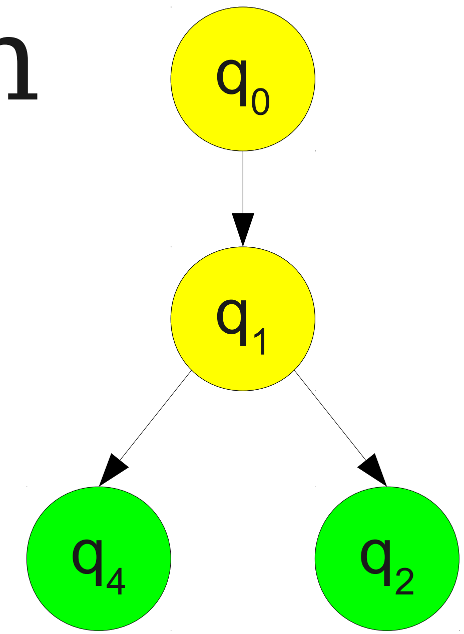


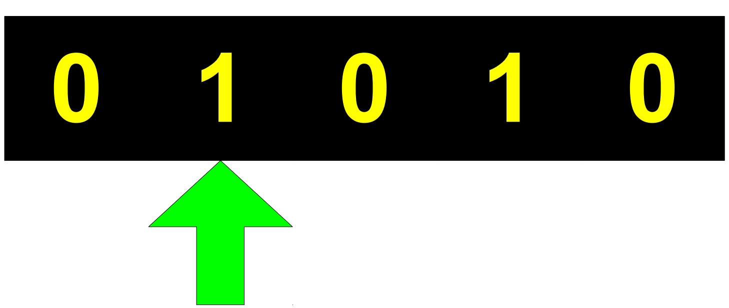




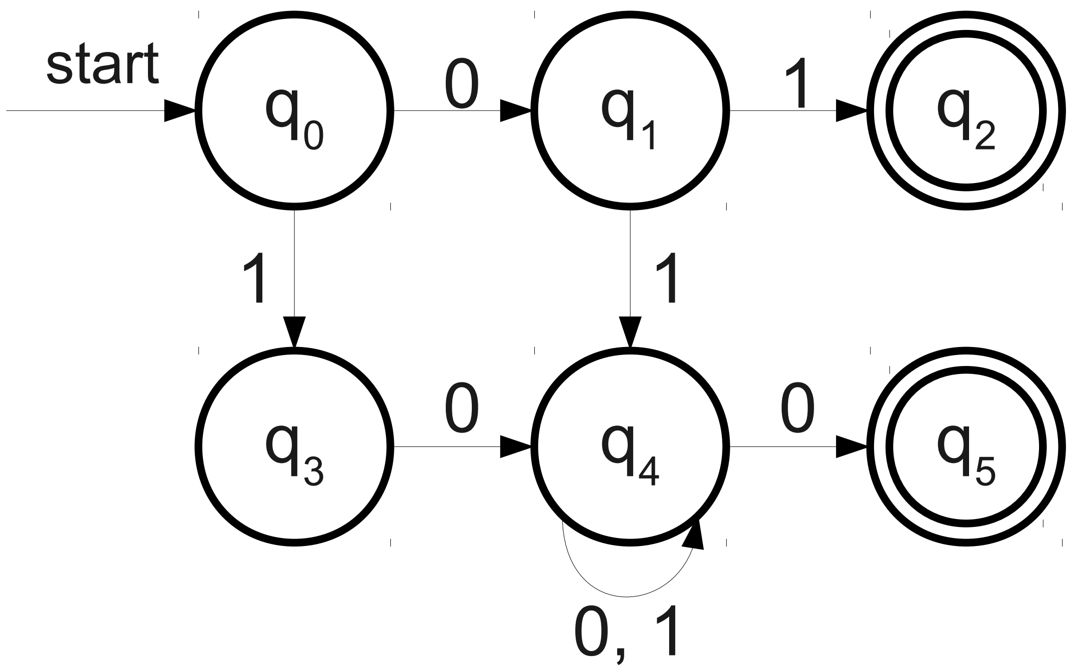
Tree Computation

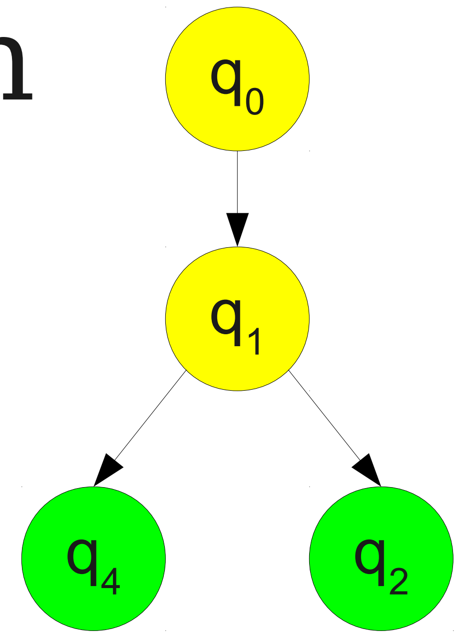


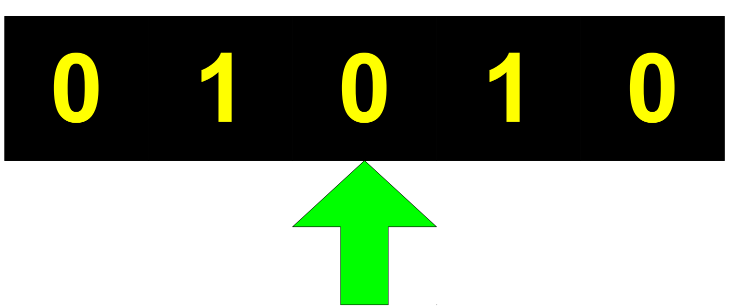




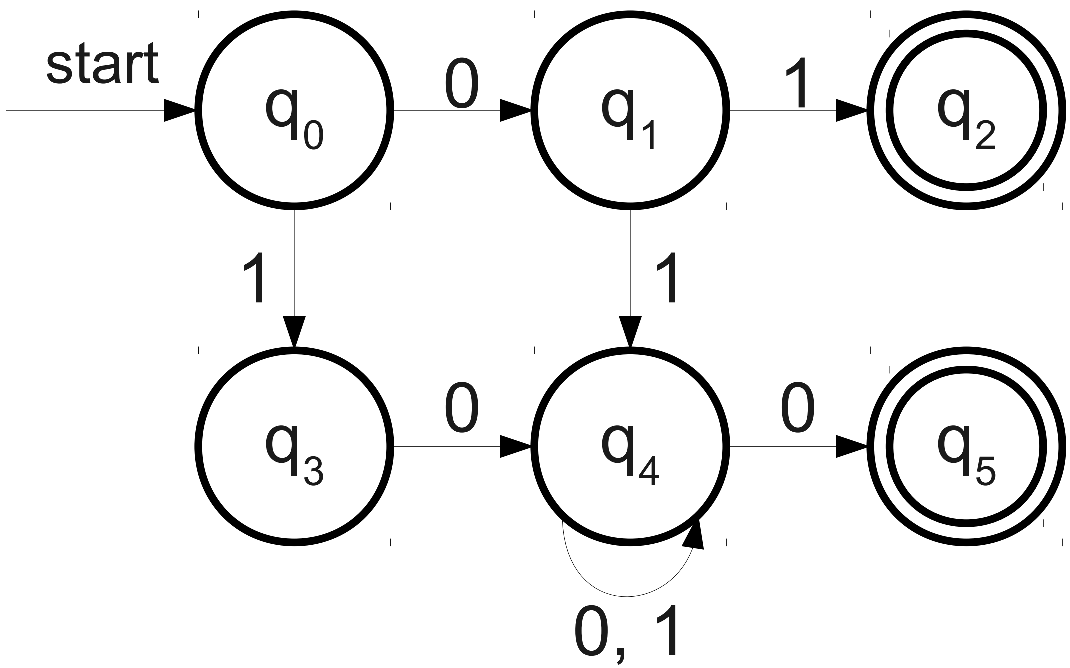
Tree Computation

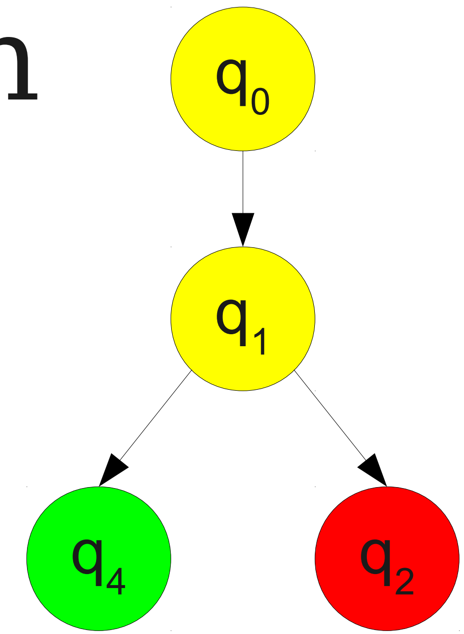


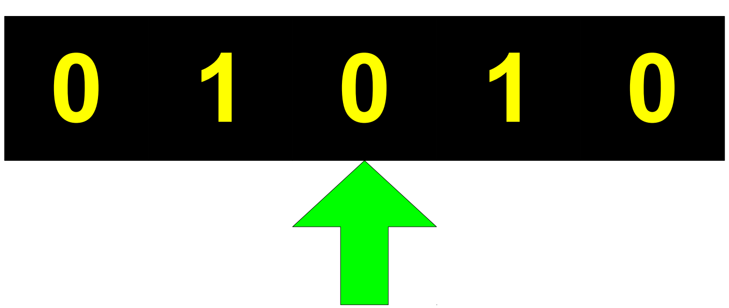




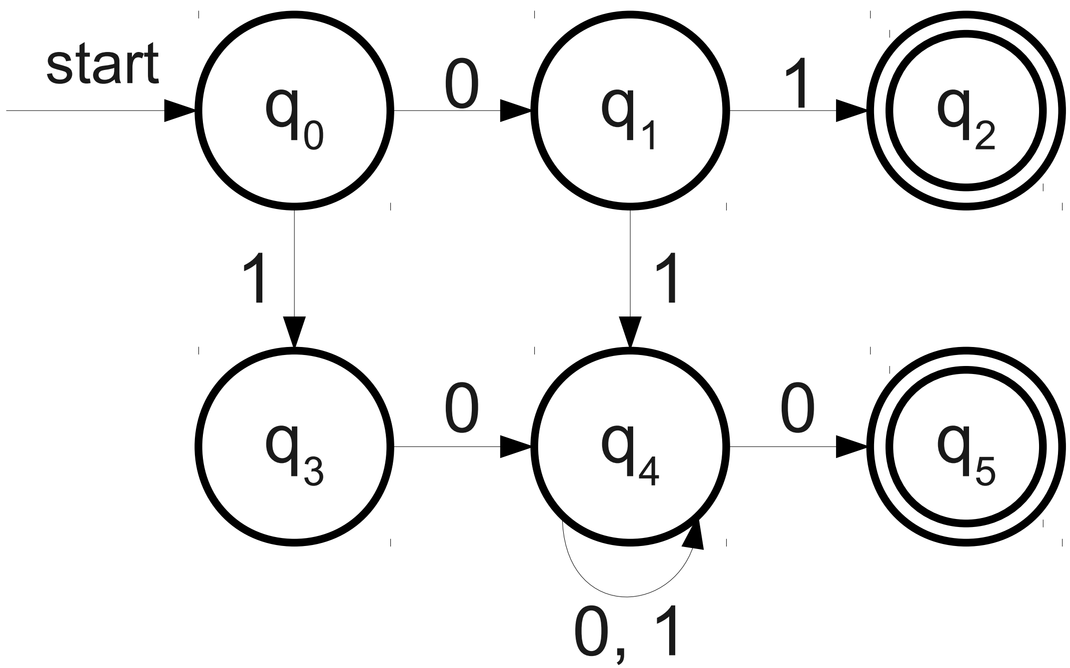
Tree Computation

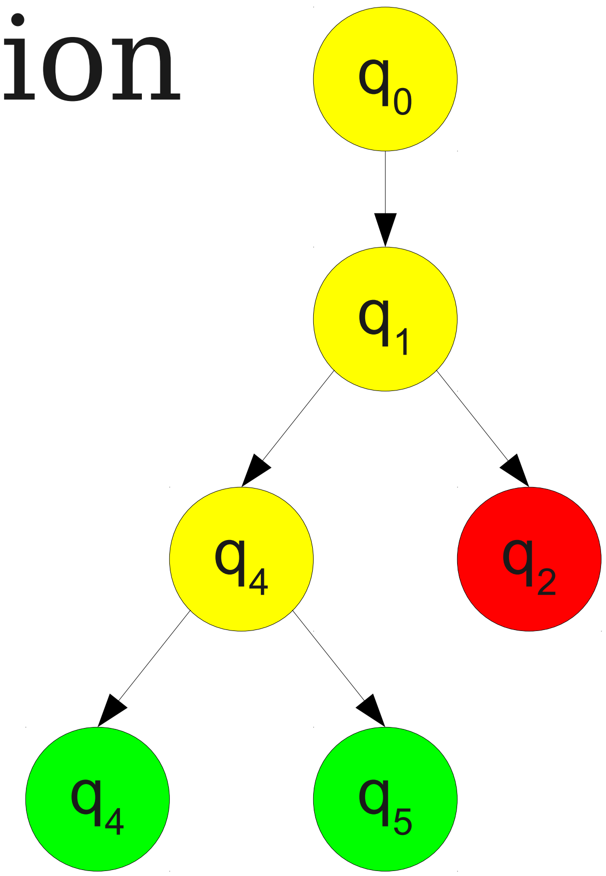


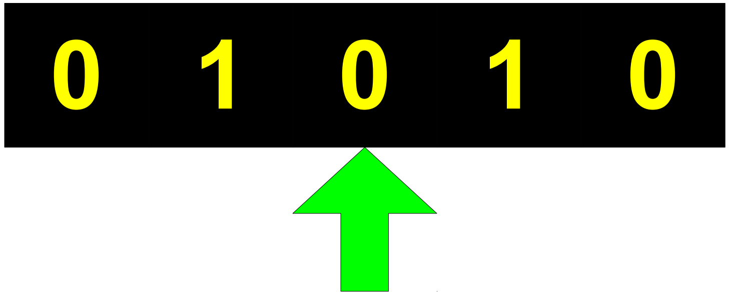




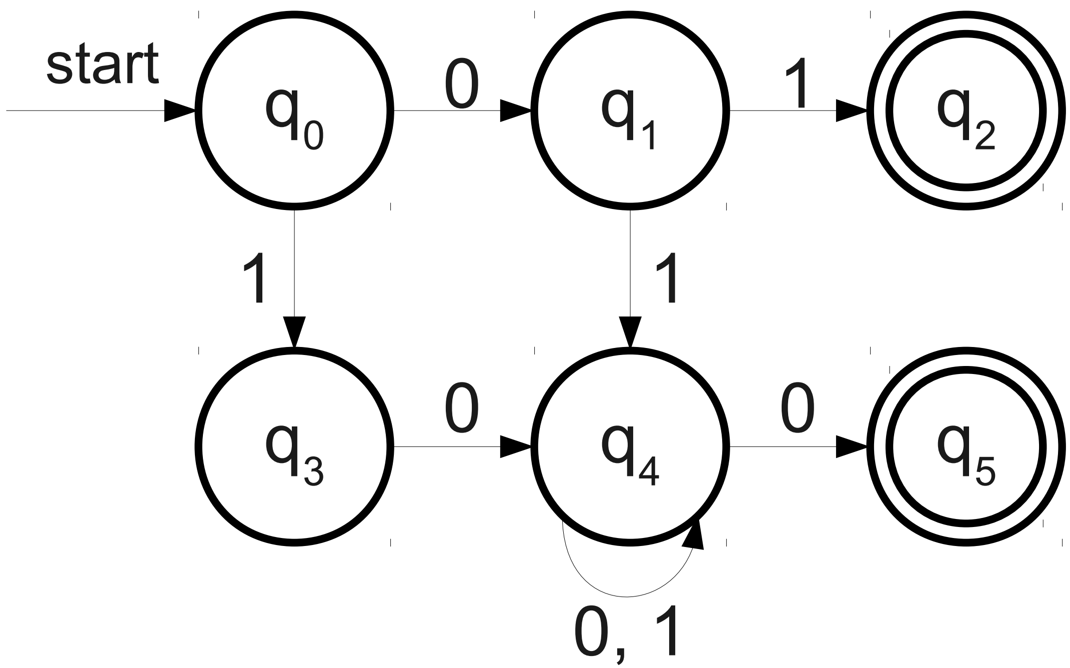
Tree Computation

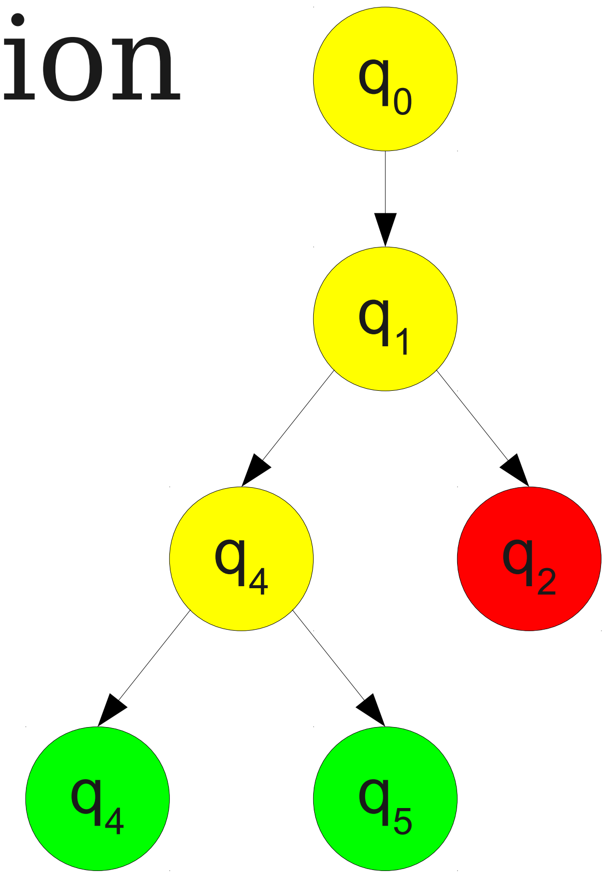


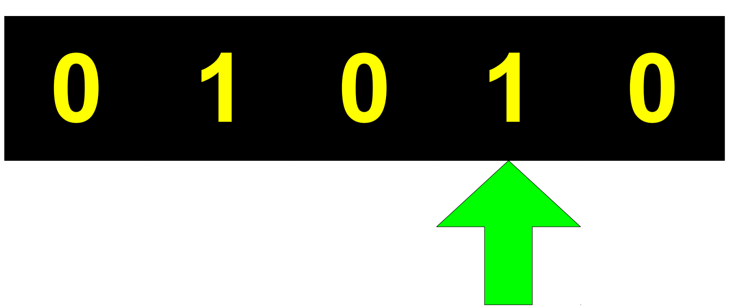




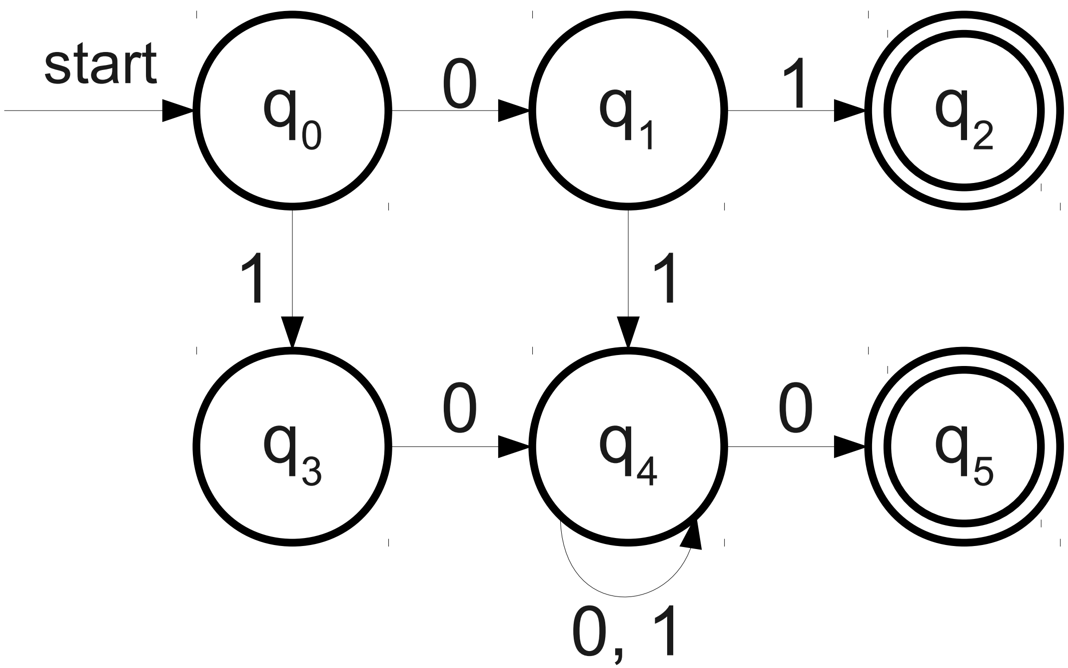
Tree Computation

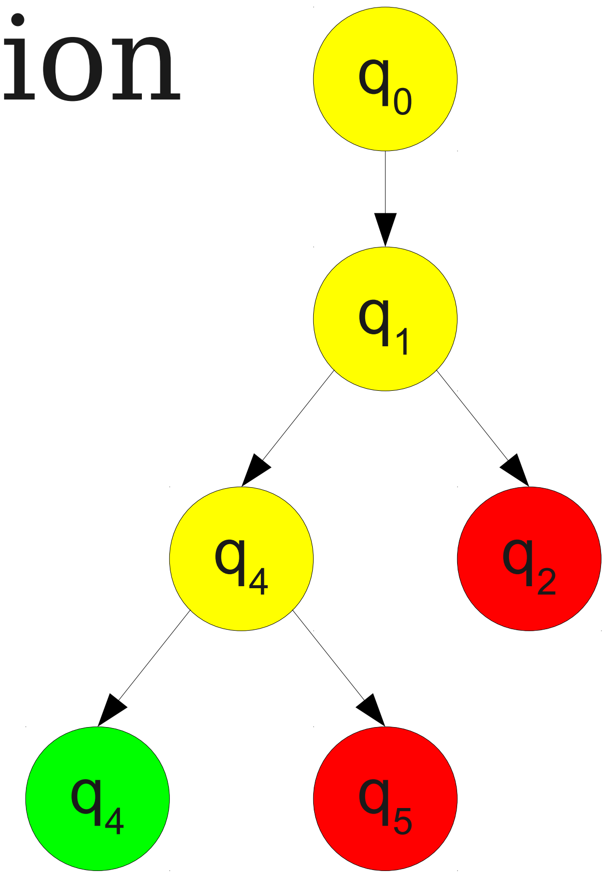


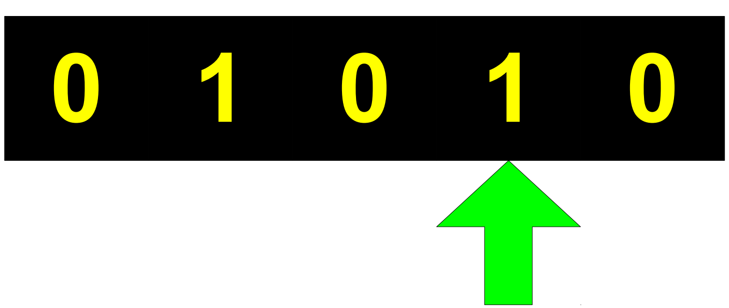




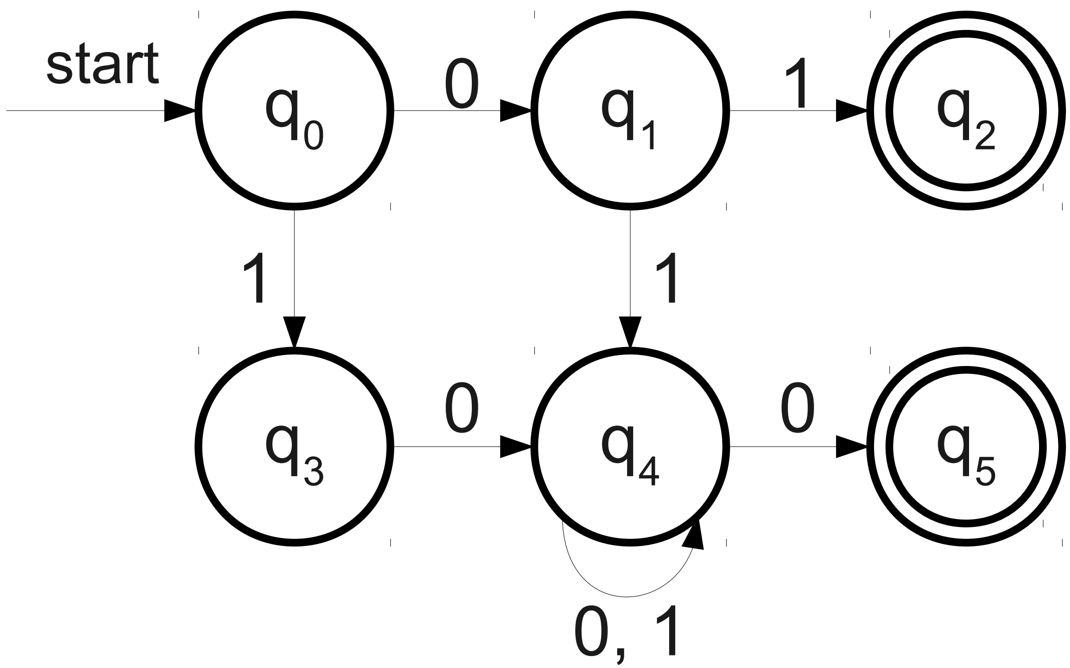
Tree Computation

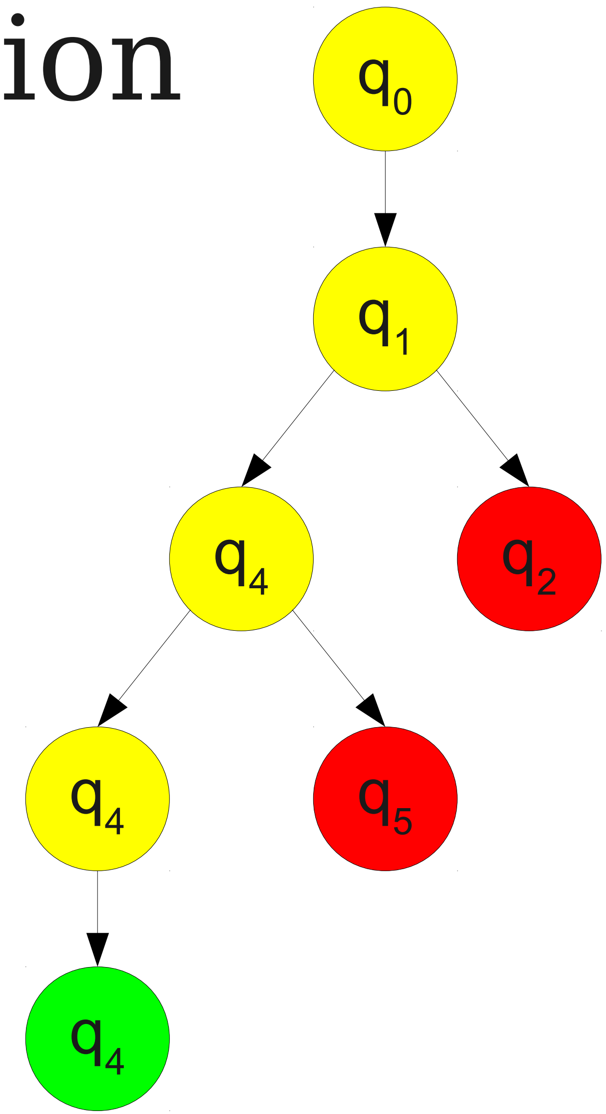


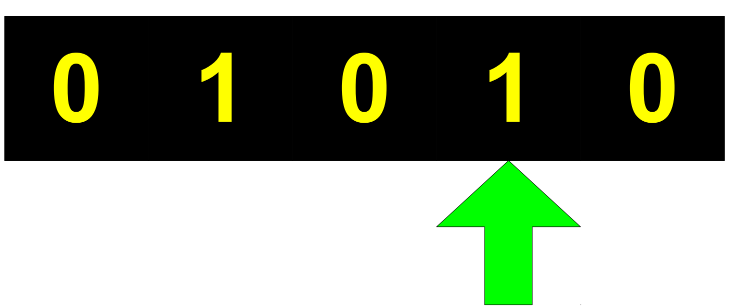




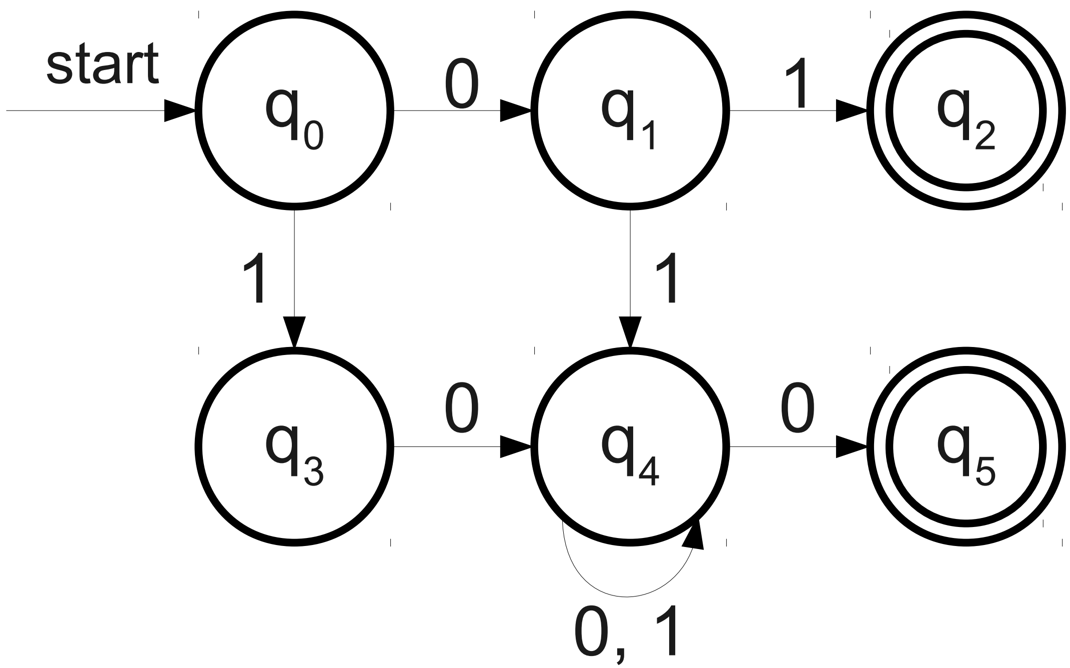
Tree Computation

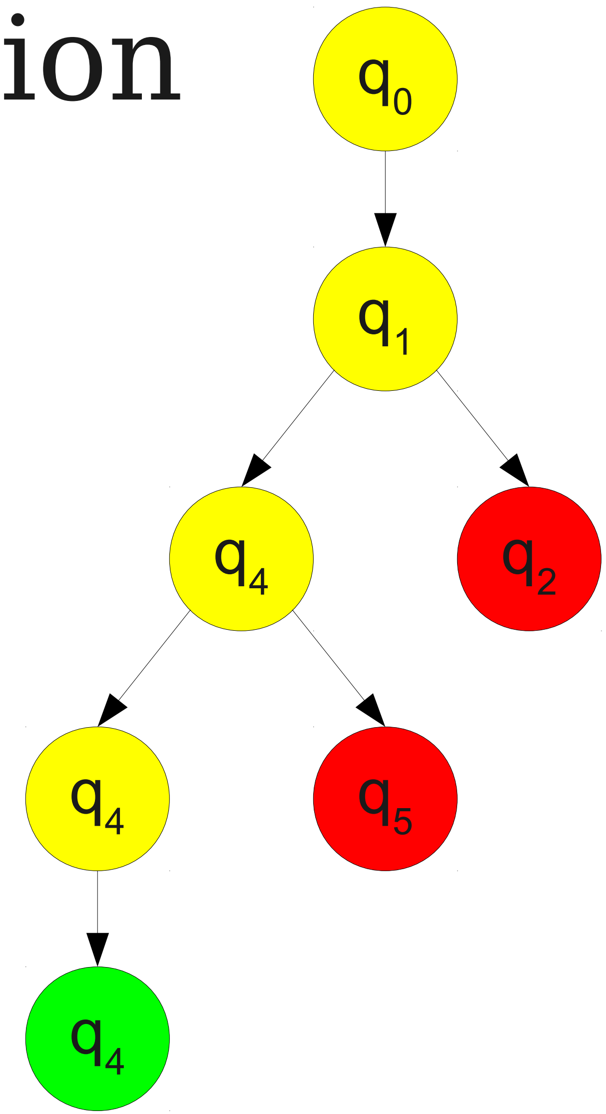




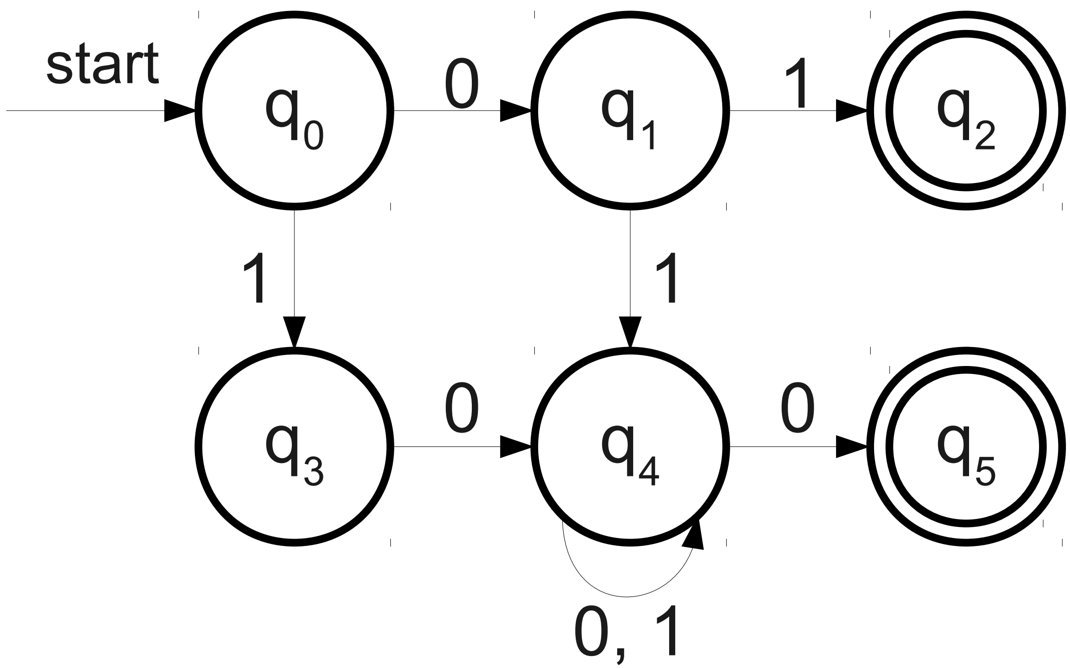


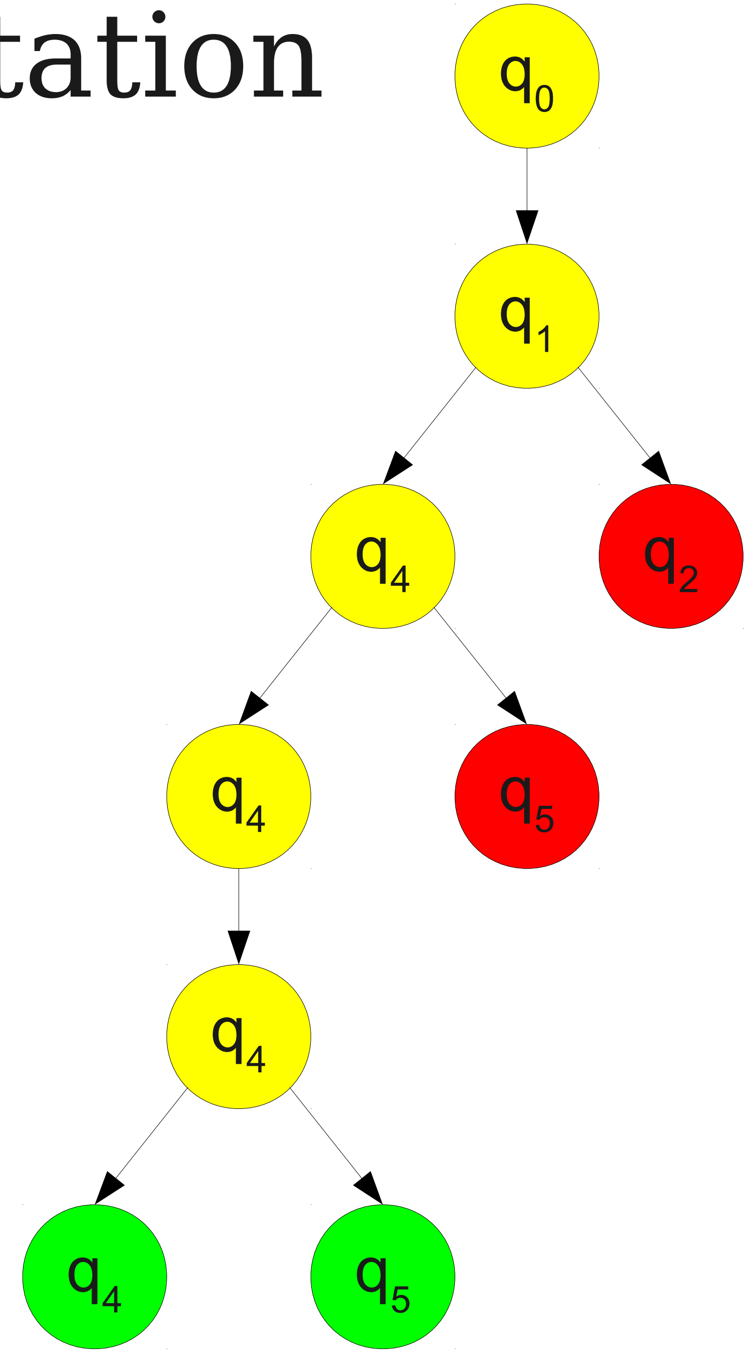
Tree Computation



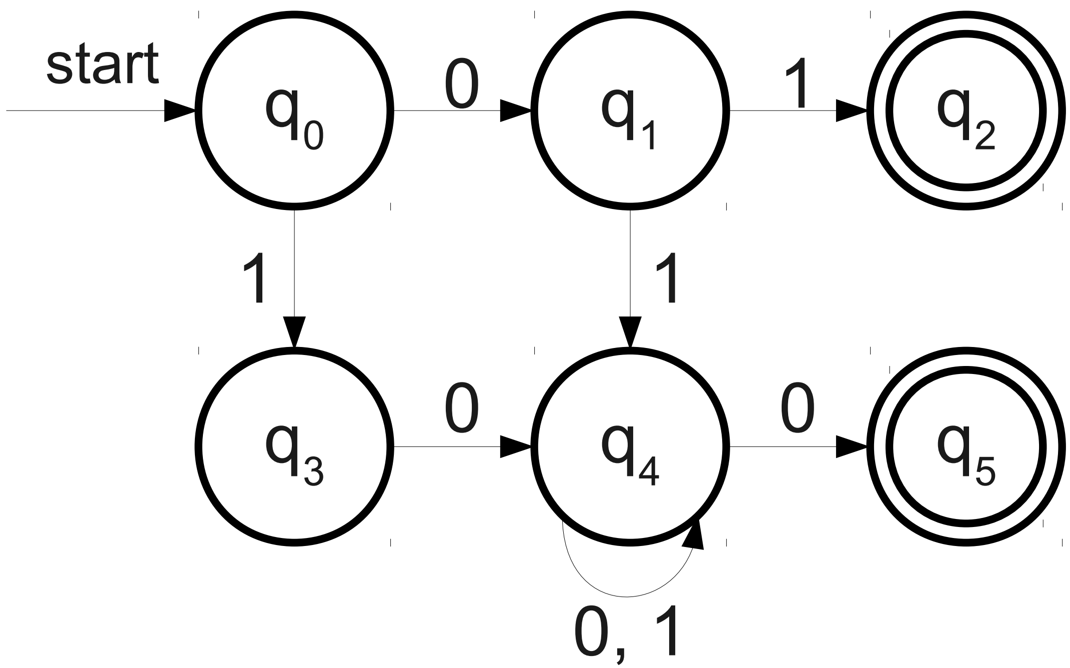


Tree Computation

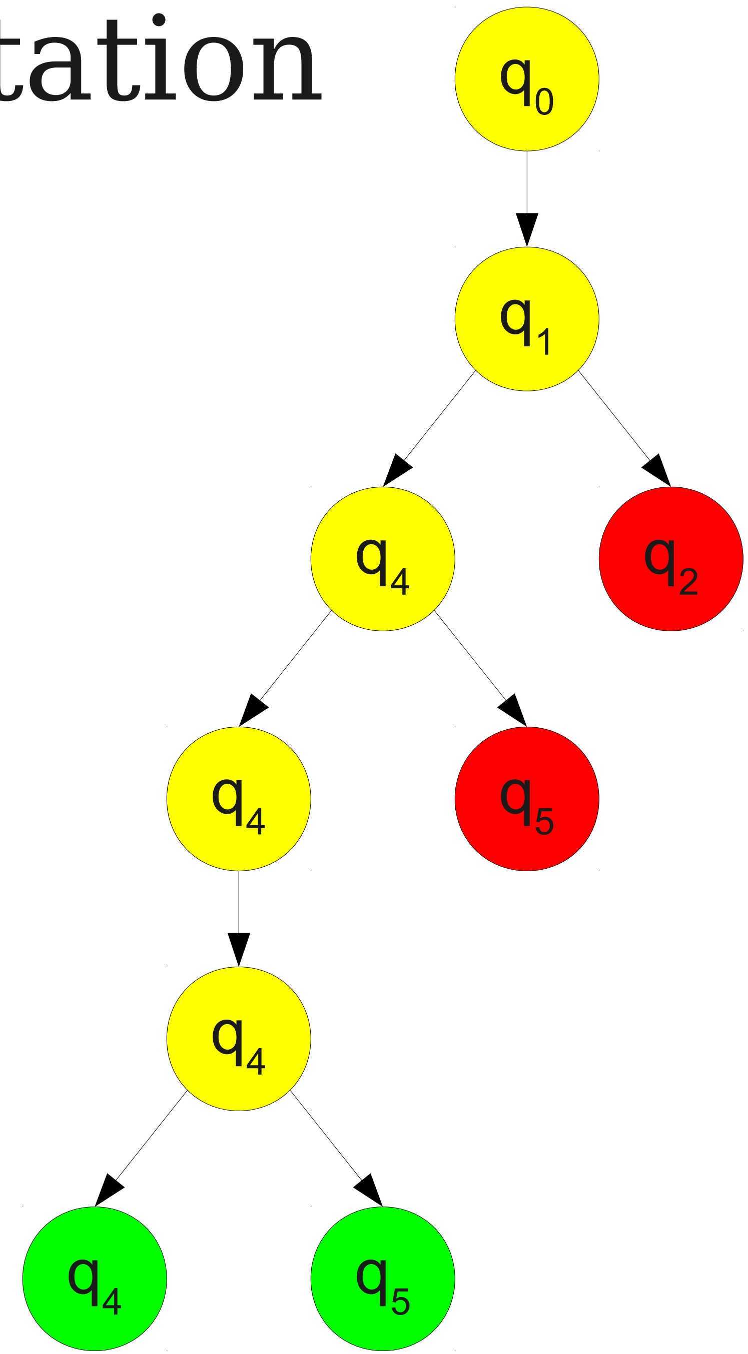




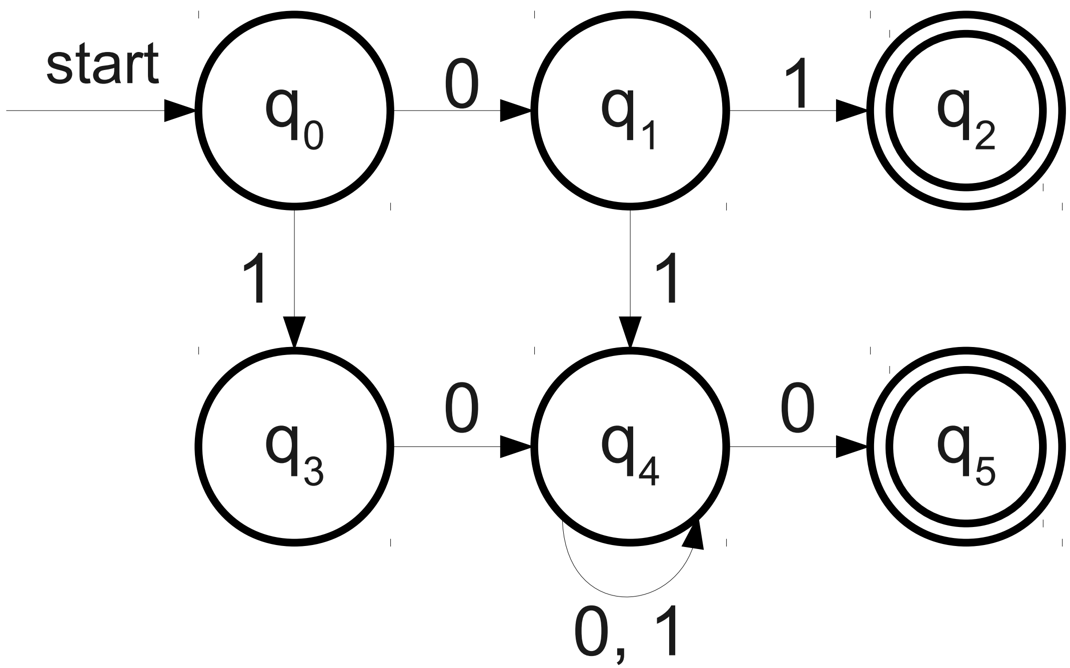
Tree Computation



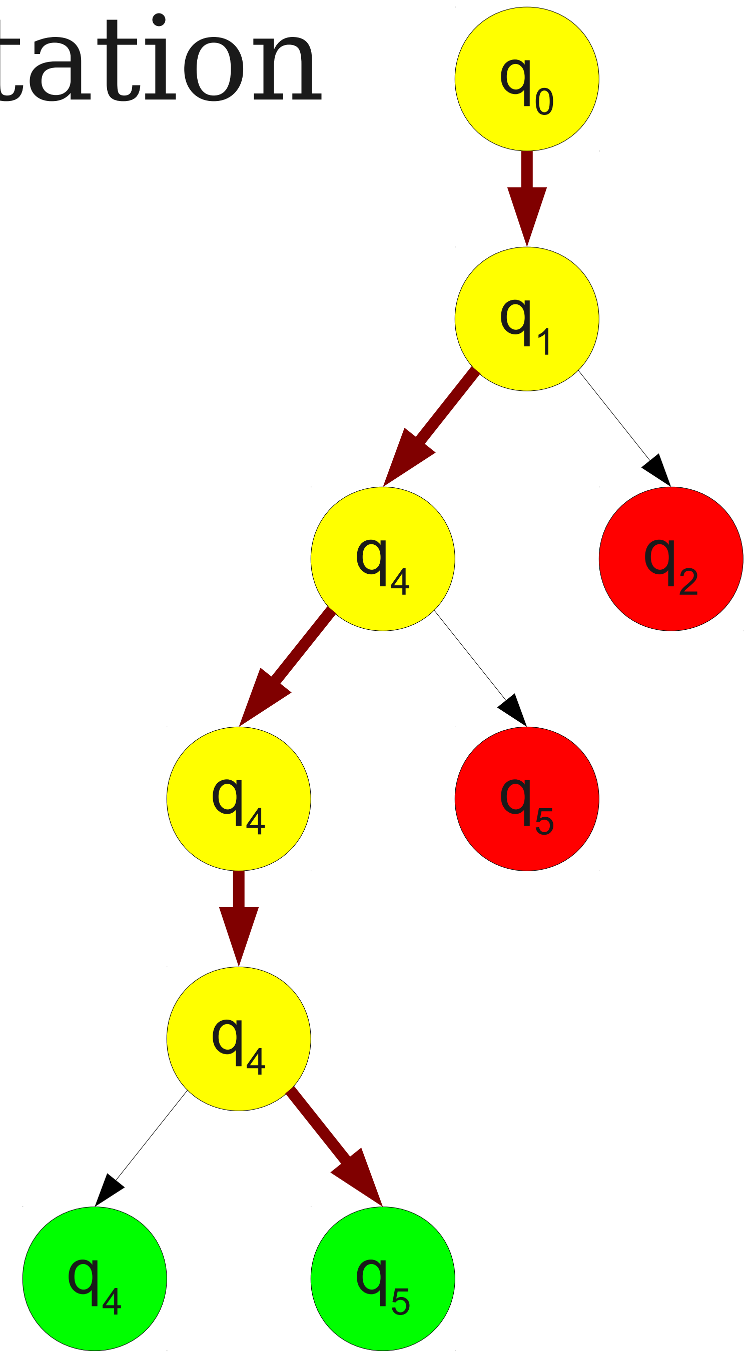
0 1 0 1 0



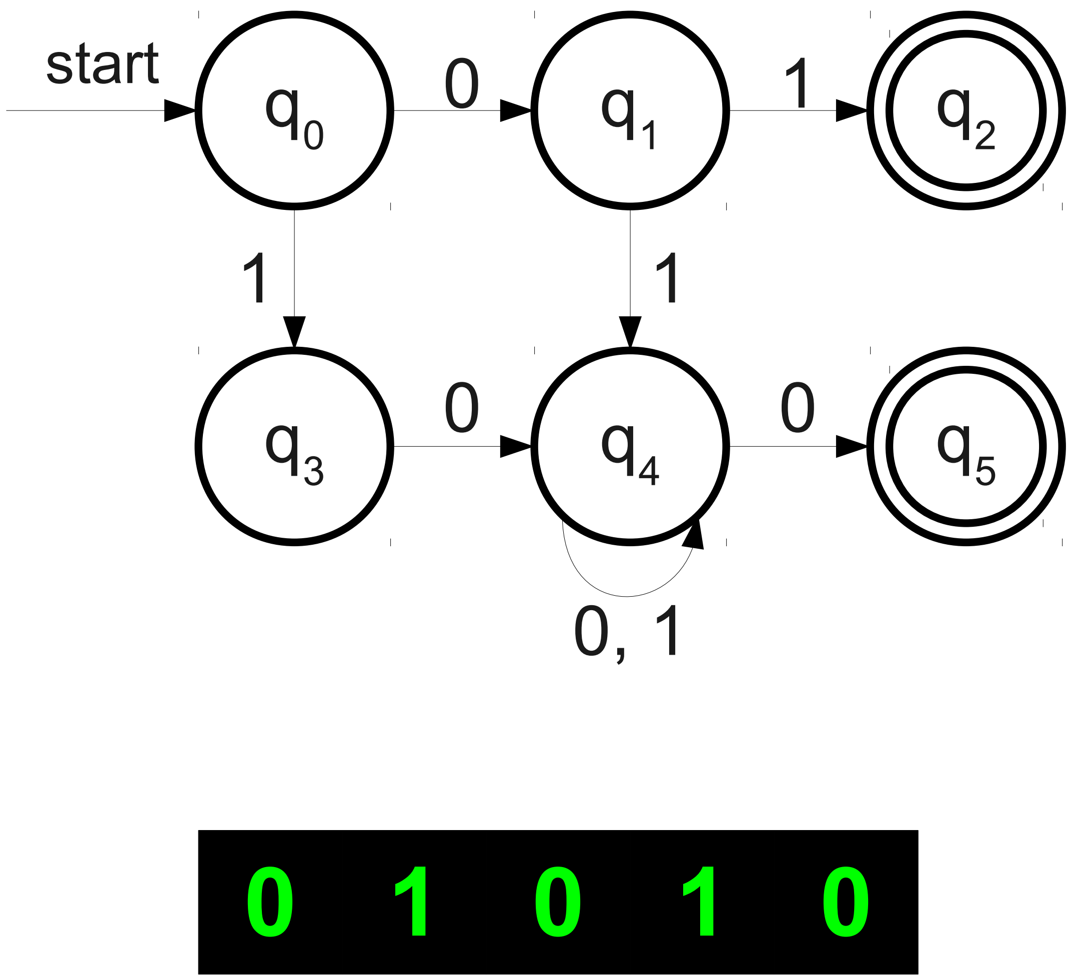
Tree Computation

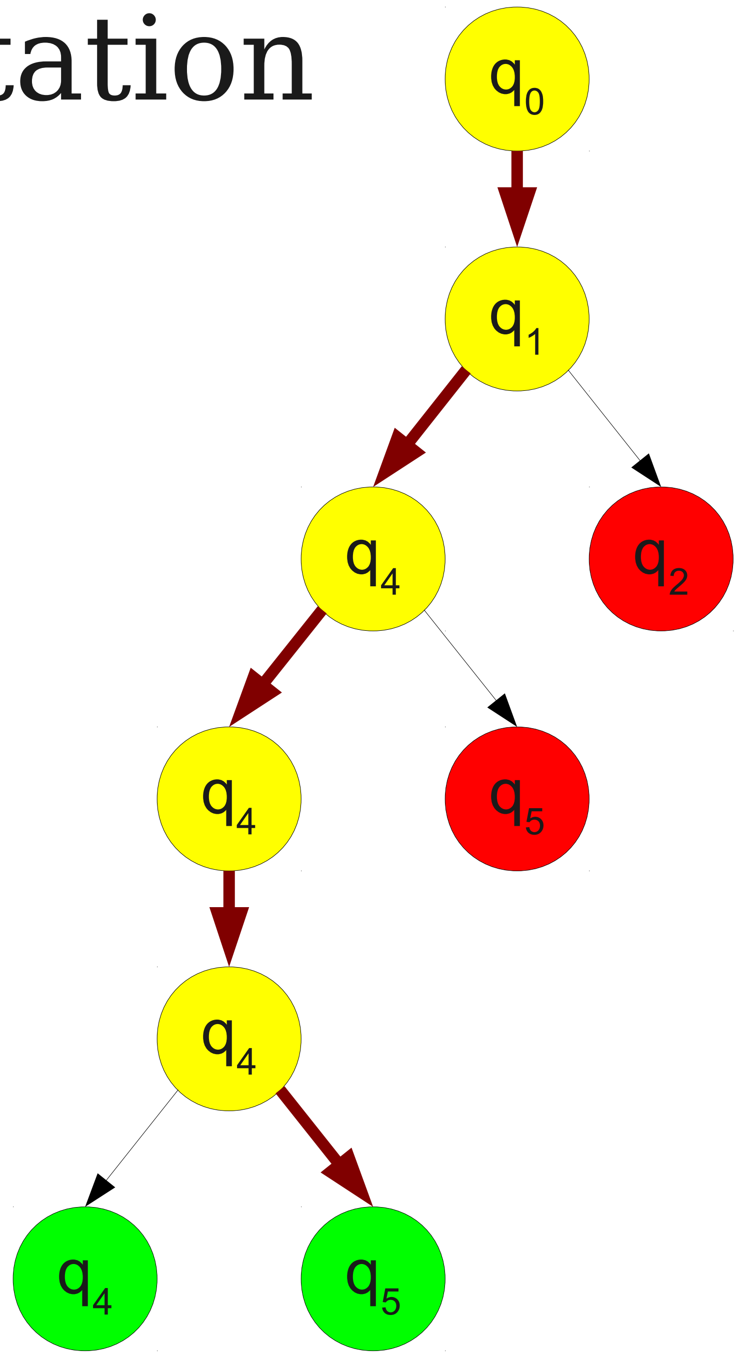


0 1 0 1 0



Tree Computation

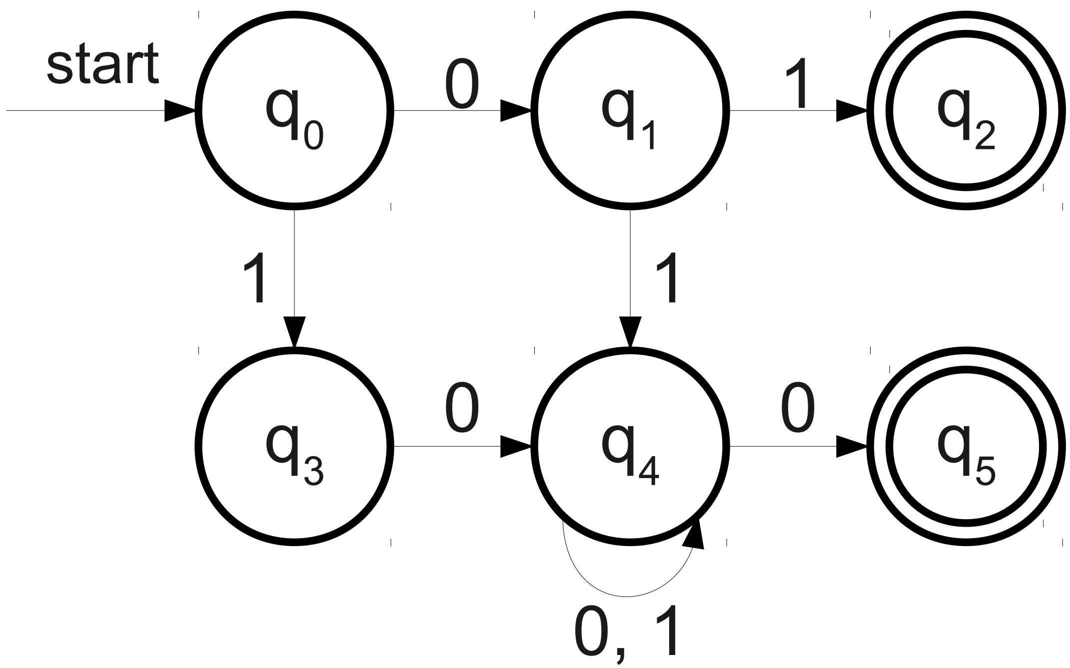




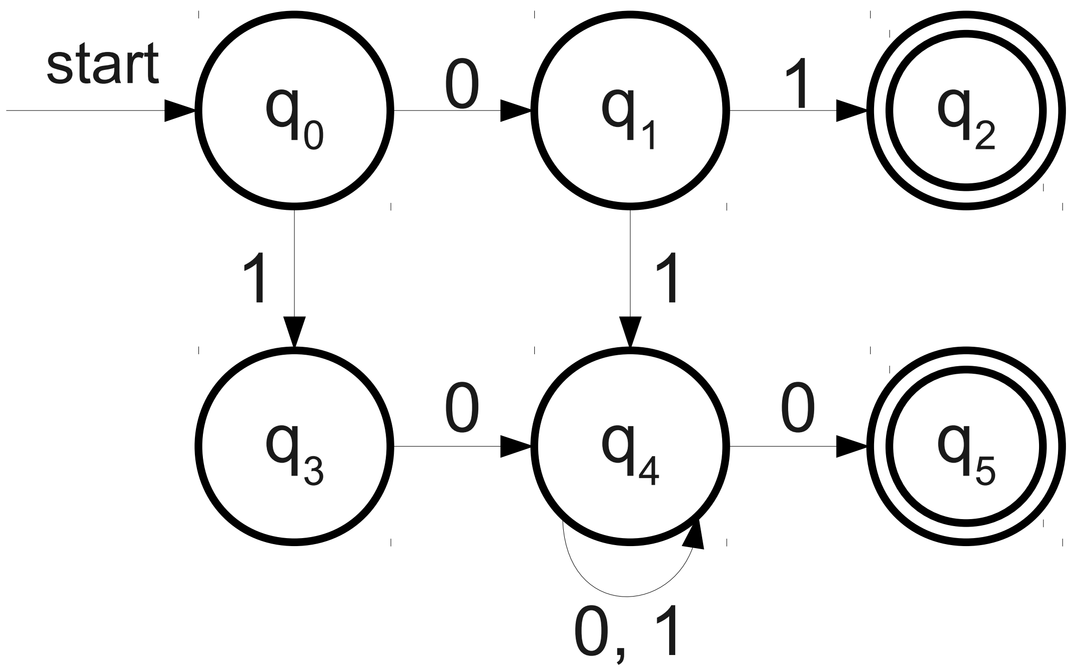
Nondeterminism as a Tree

- 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

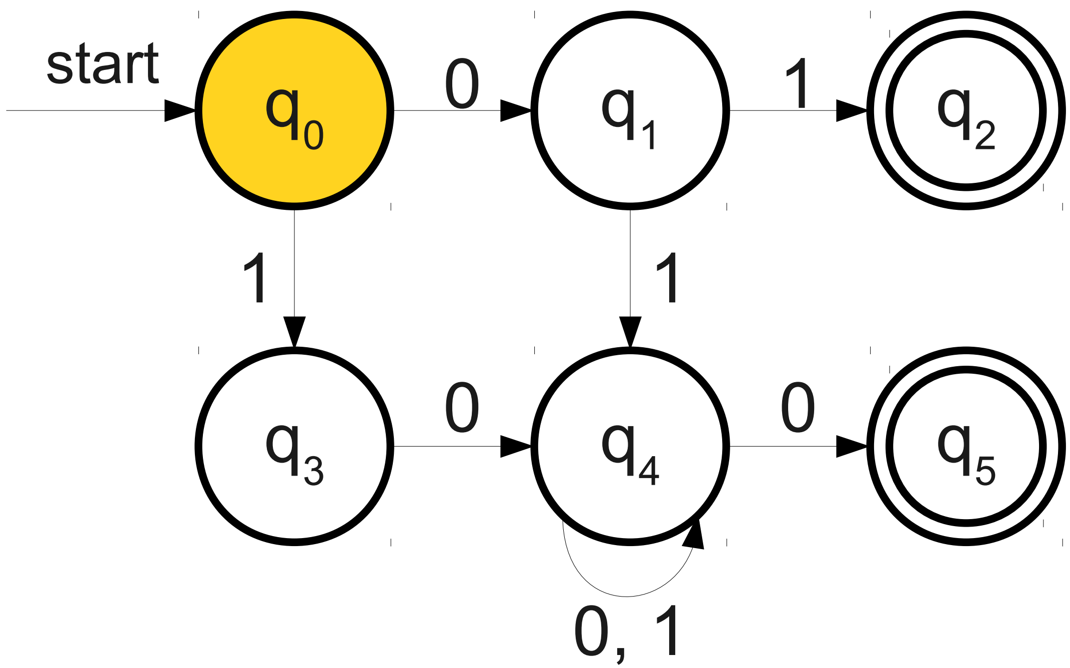


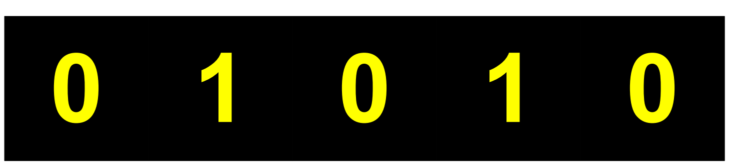
Perfect Guessing



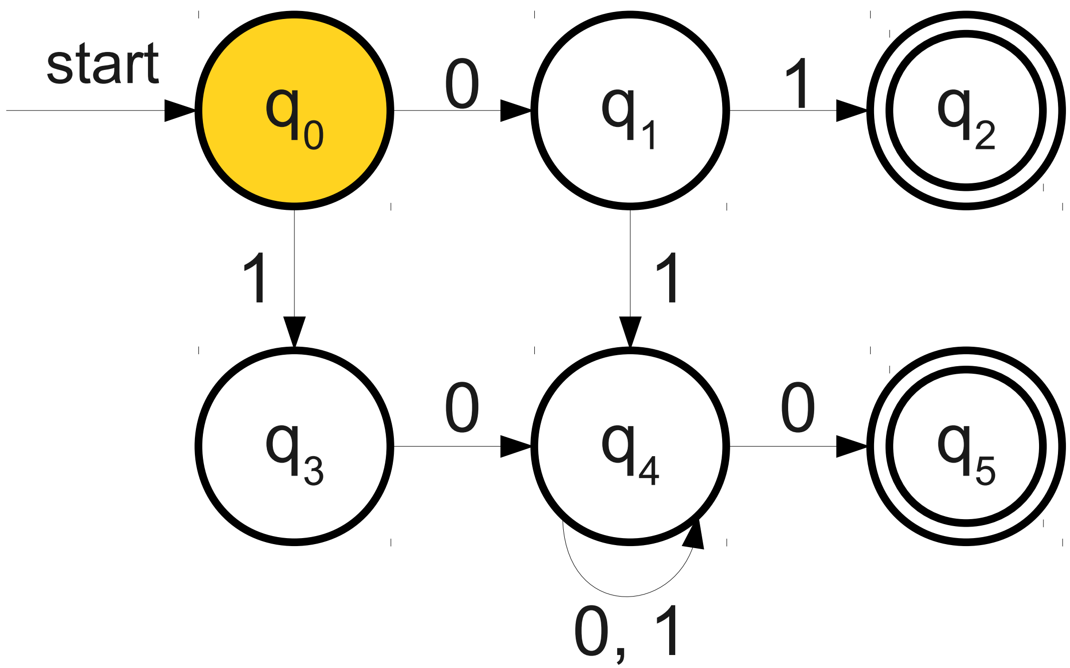
0	1	0	1	0

Perfect Guessing

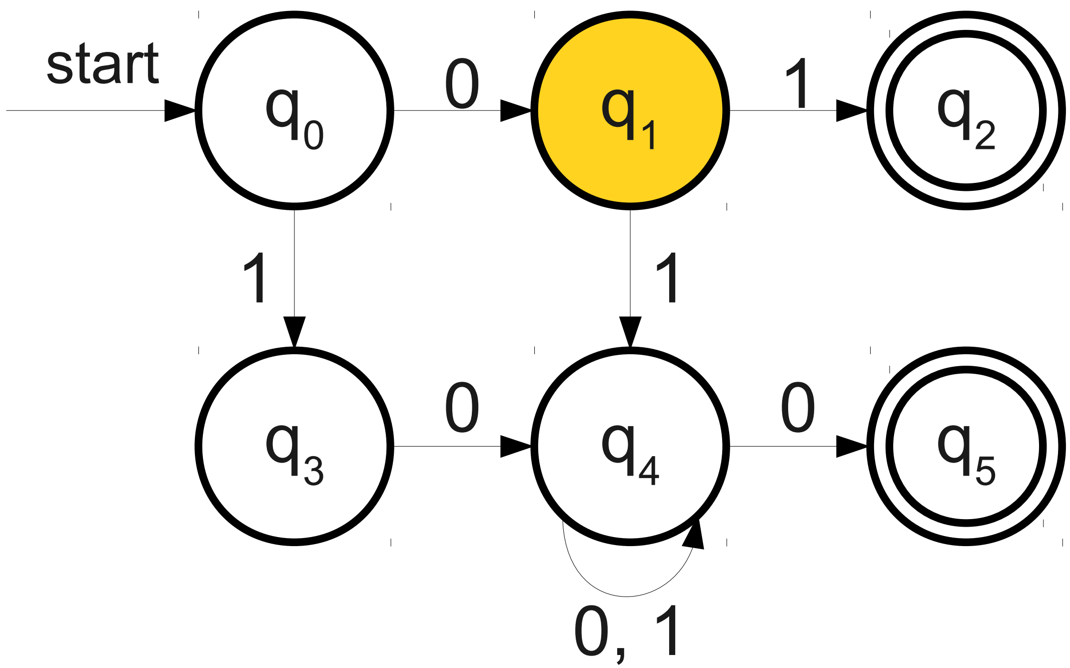




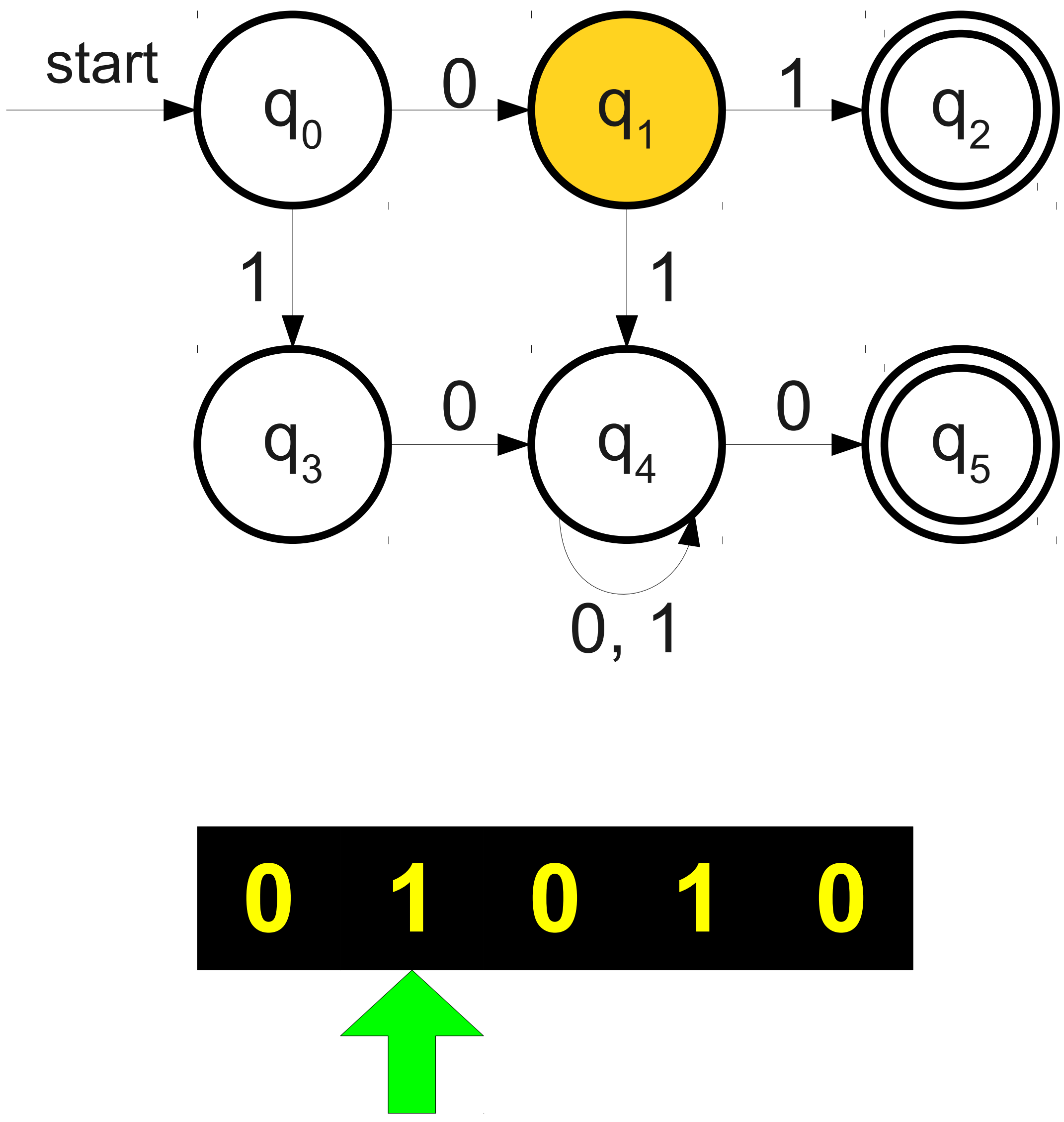
Perfect Guessing



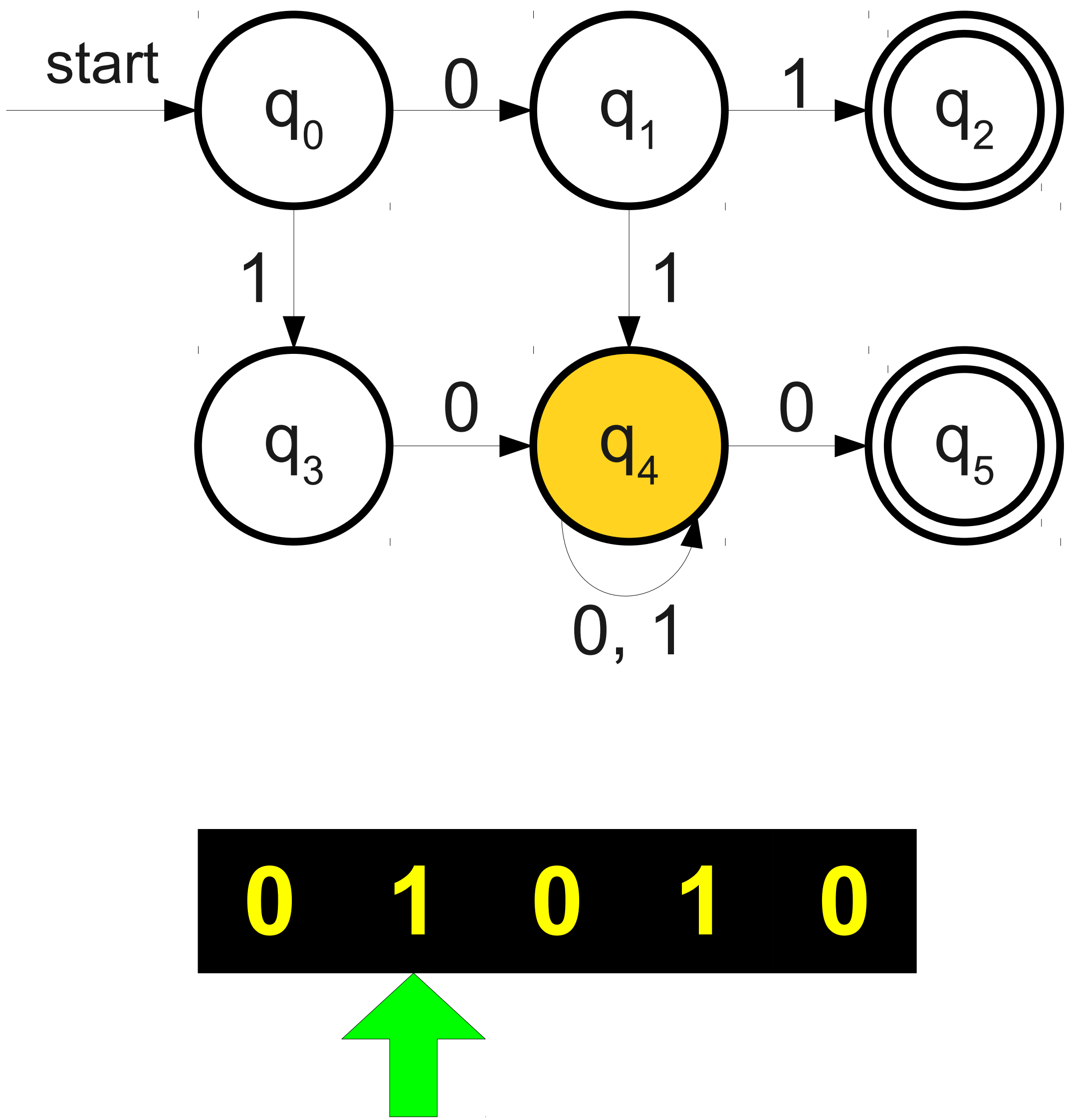
Perfect Guessing



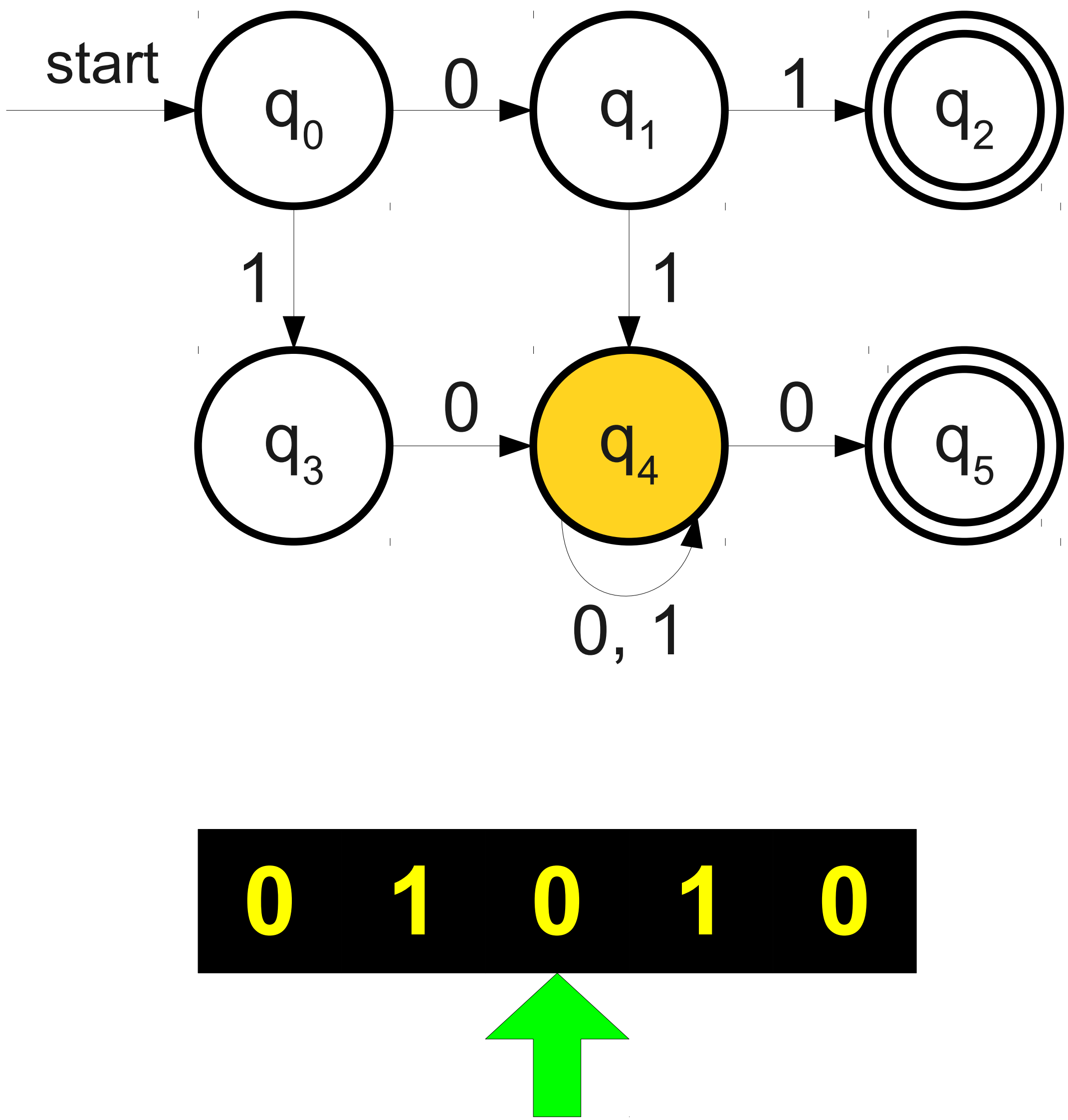
Perfect Guessing



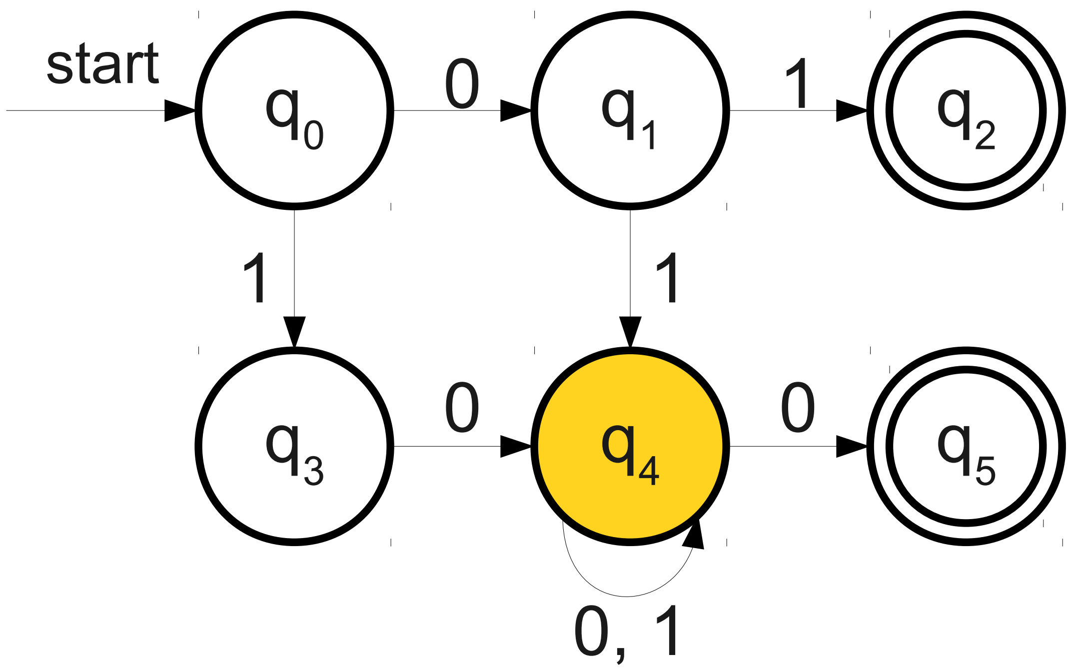
Perfect Guessing

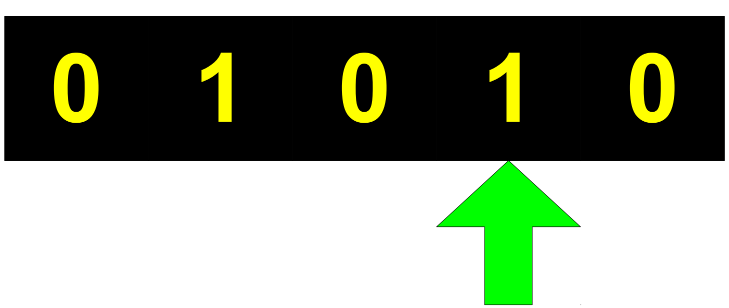


Perfect Guessing

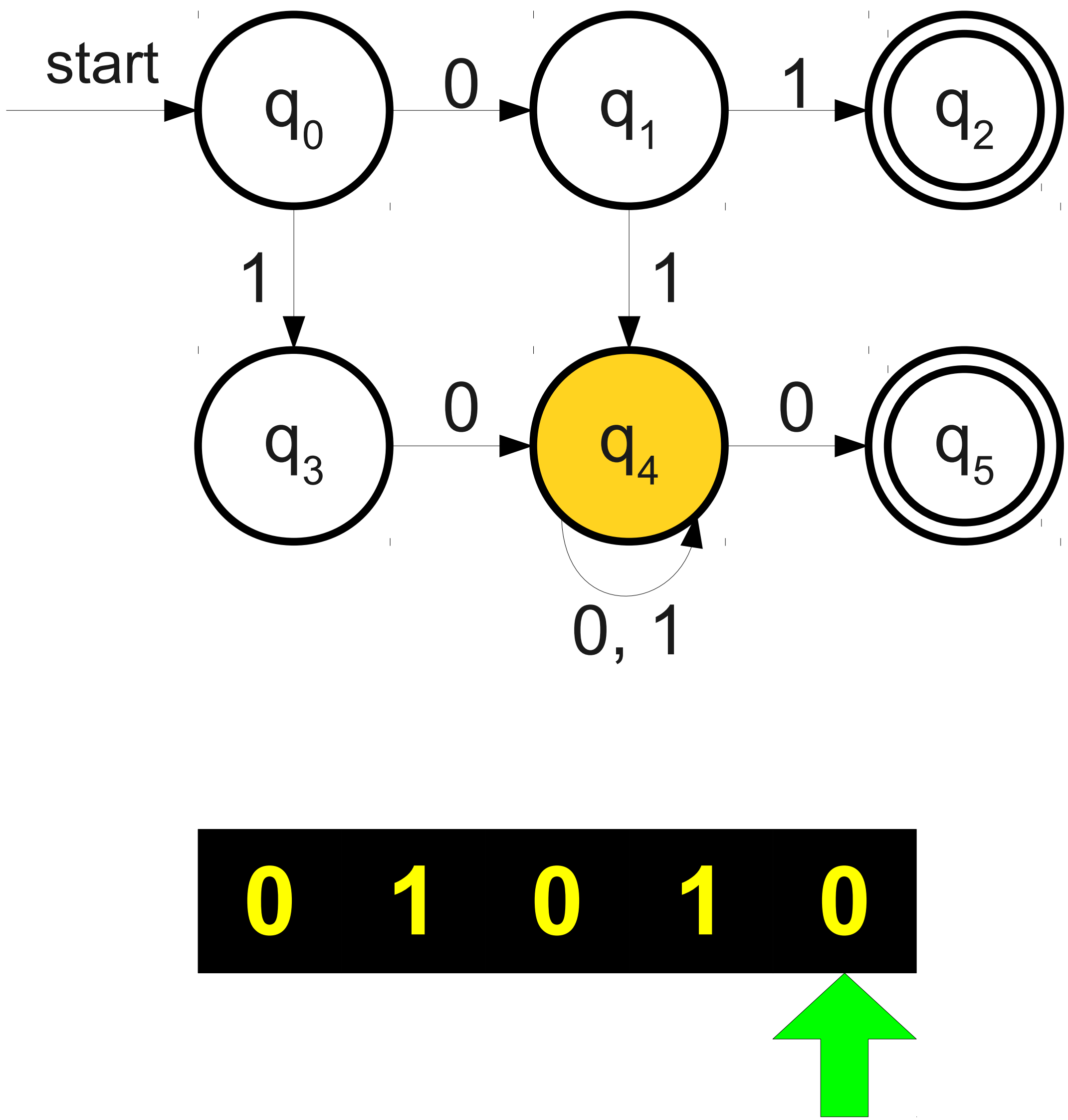


Perfect Guessing

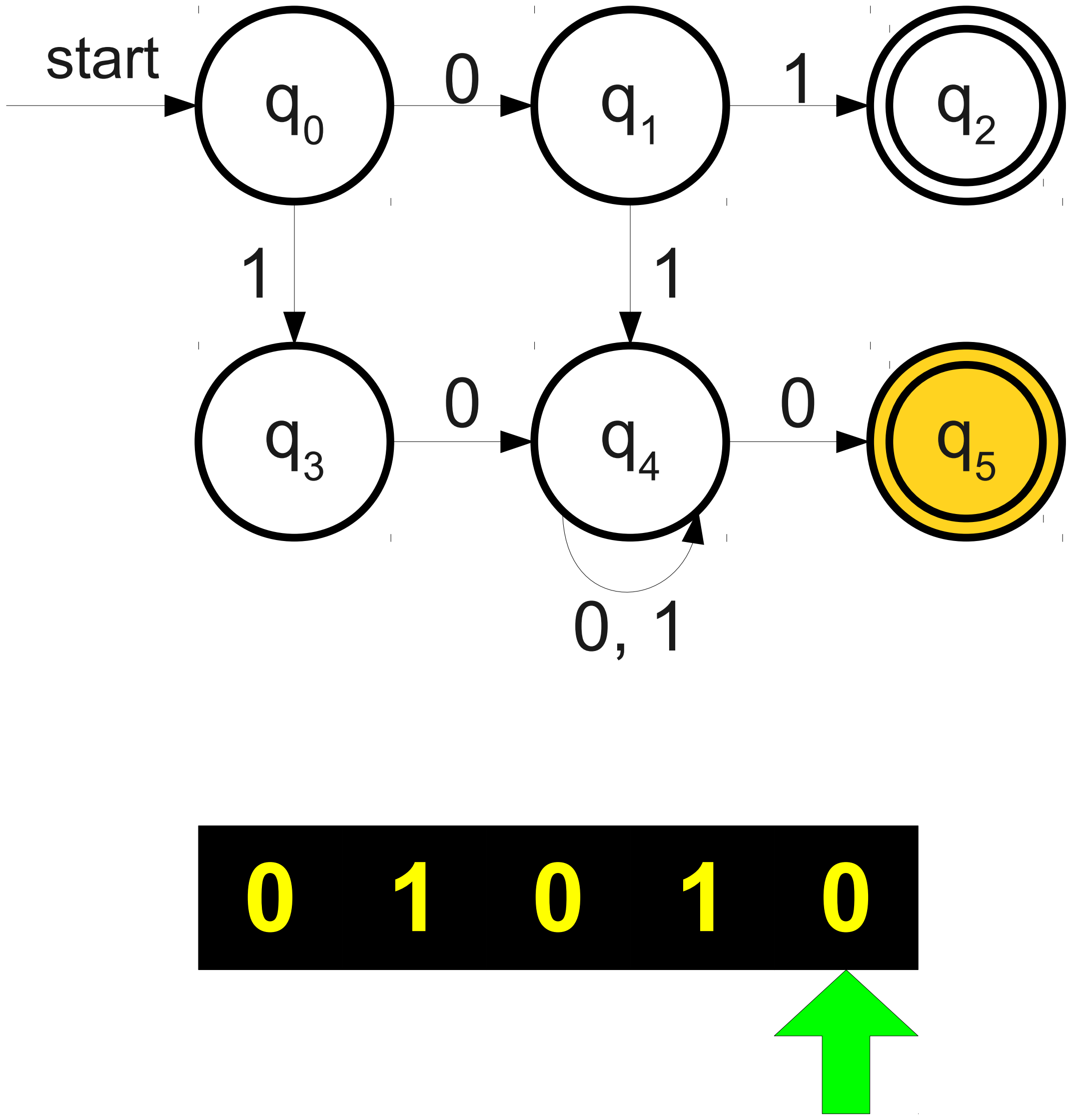




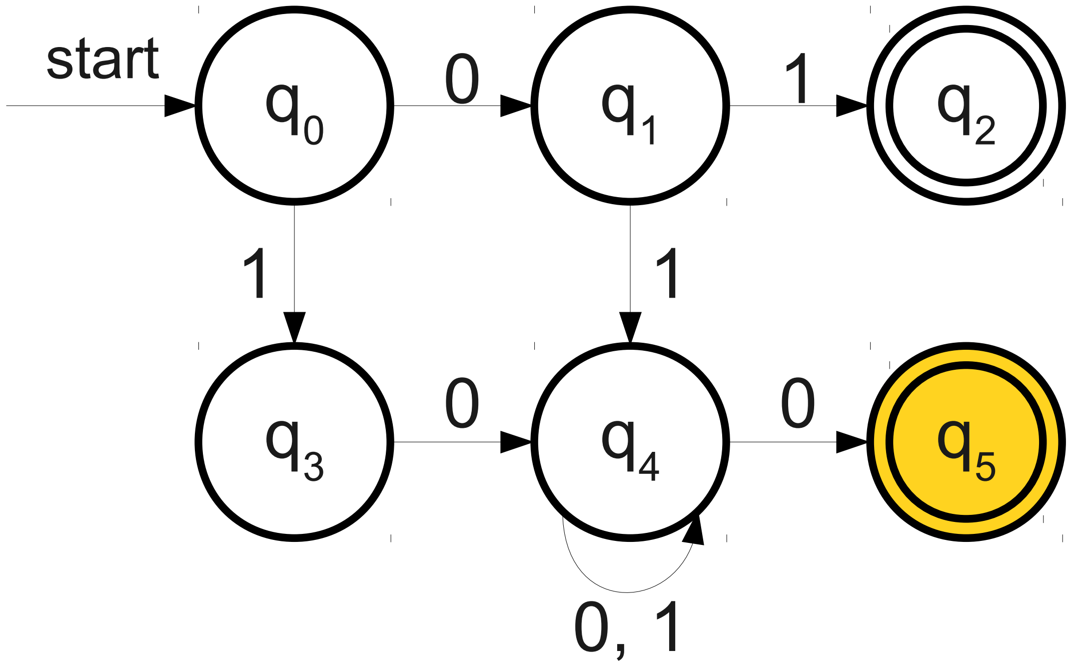
Perfect Guessing



Perfect Guessing

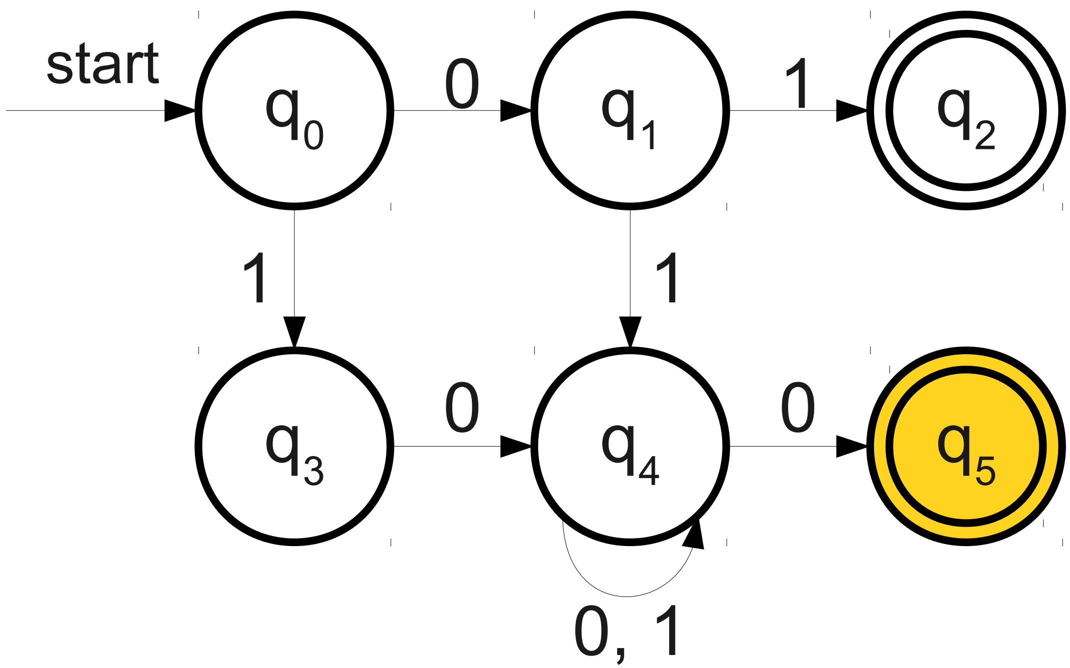


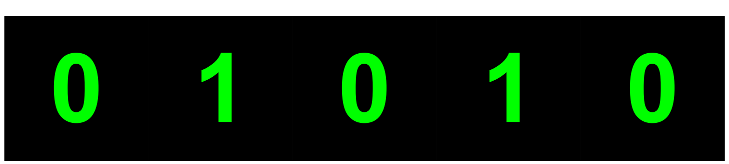
Perfect Guessing



0 1 0 1 0

Perfect Guessing

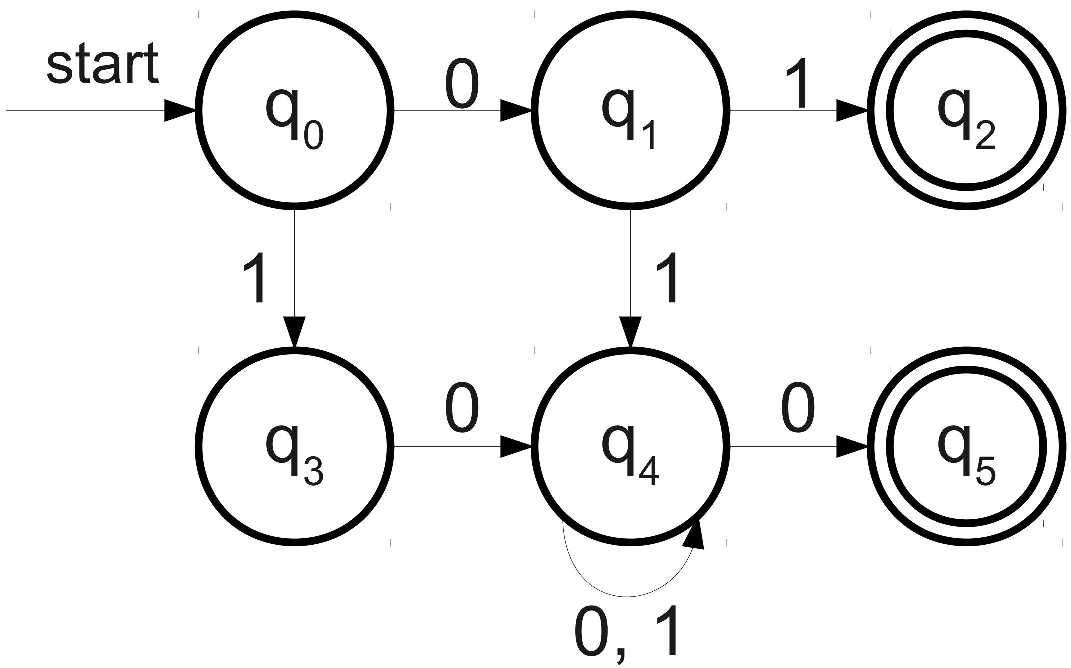


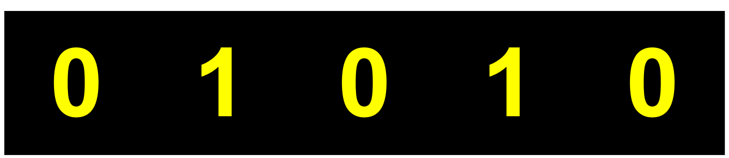


Perfect Guessing

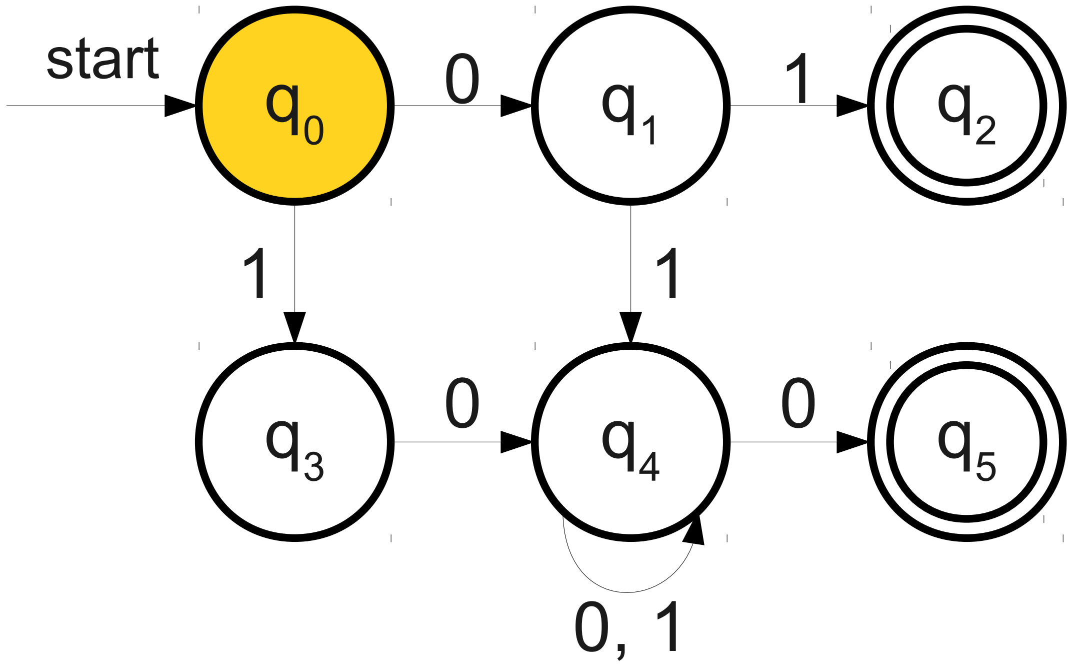
- We can view nondeterministic machines as having **Magic Superpowers** that enable them to guess the correct choice of moves to make.
- Idea: Machine can always guess the right choice if one exists.
- No physical analog for something of this sort.
  - (Those of you thinking quantum computing – this is not the same thing. We actually don't fully know the relation between quantum and nondeterministic computation.)

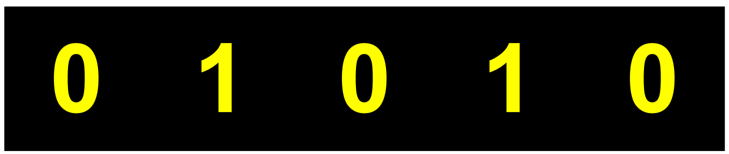
Massive Parallelism



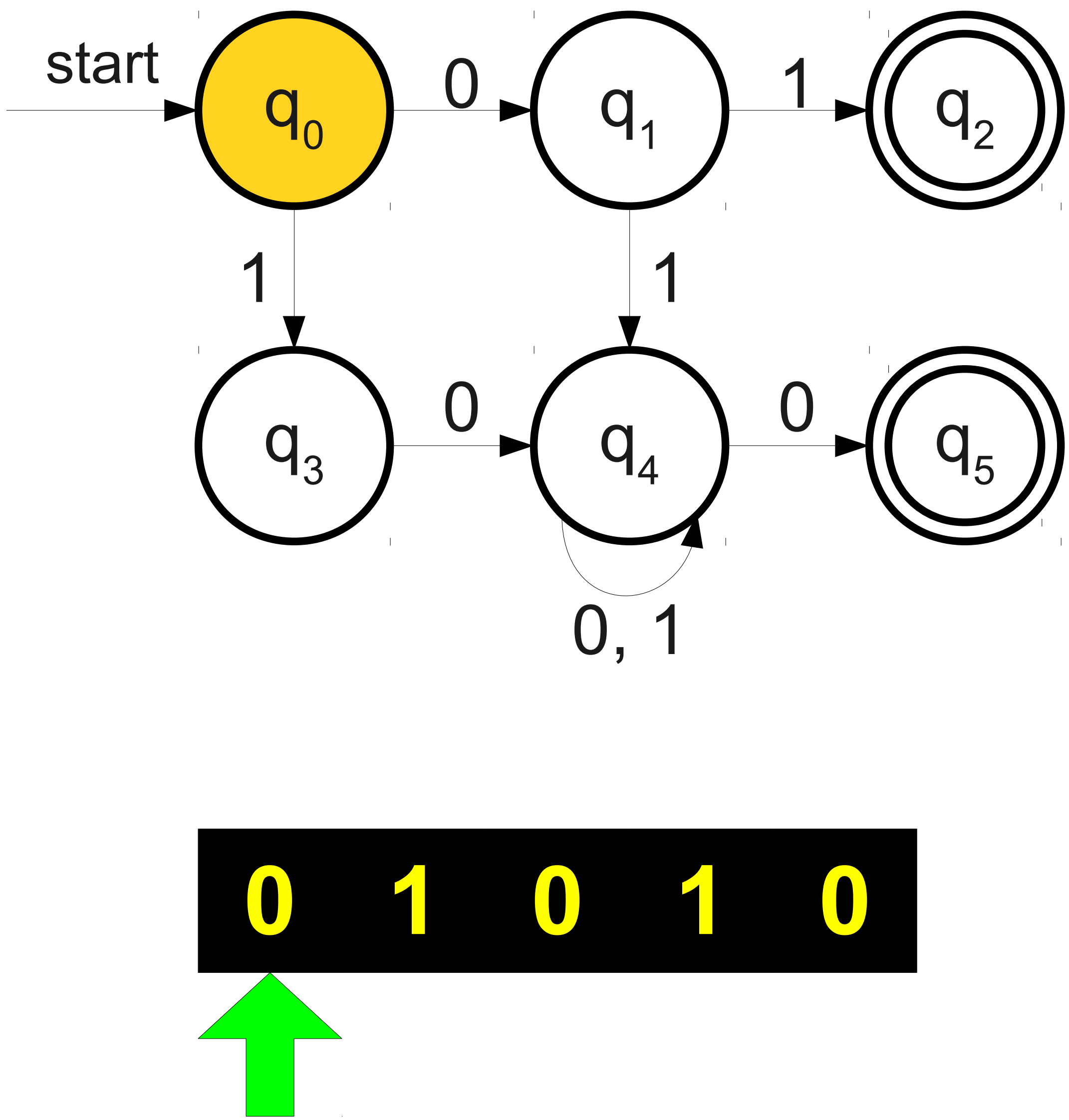


Massive Parallelism

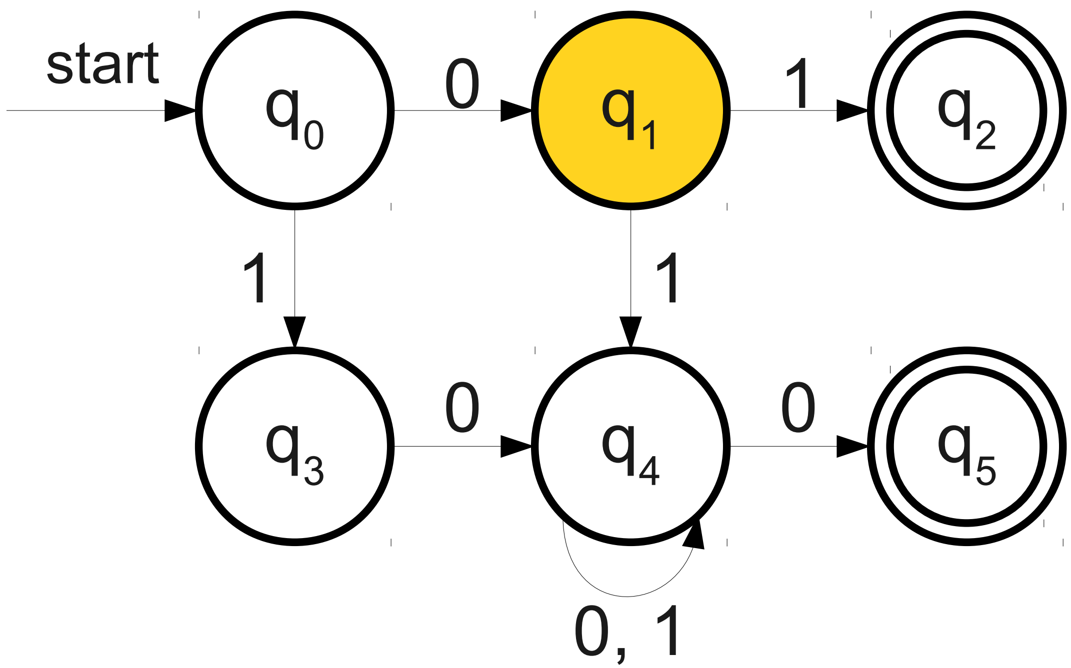




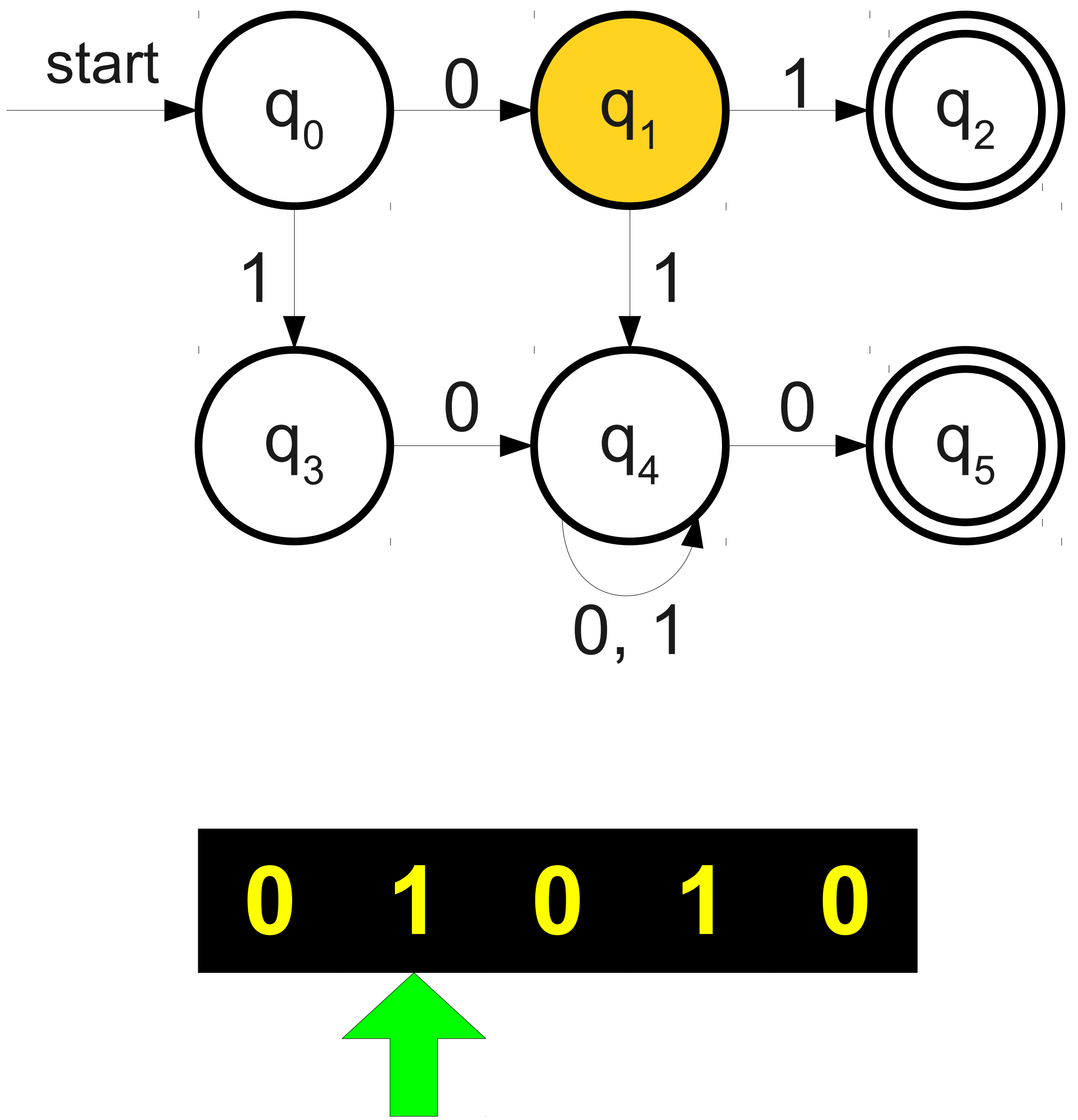
Massive Parallelism



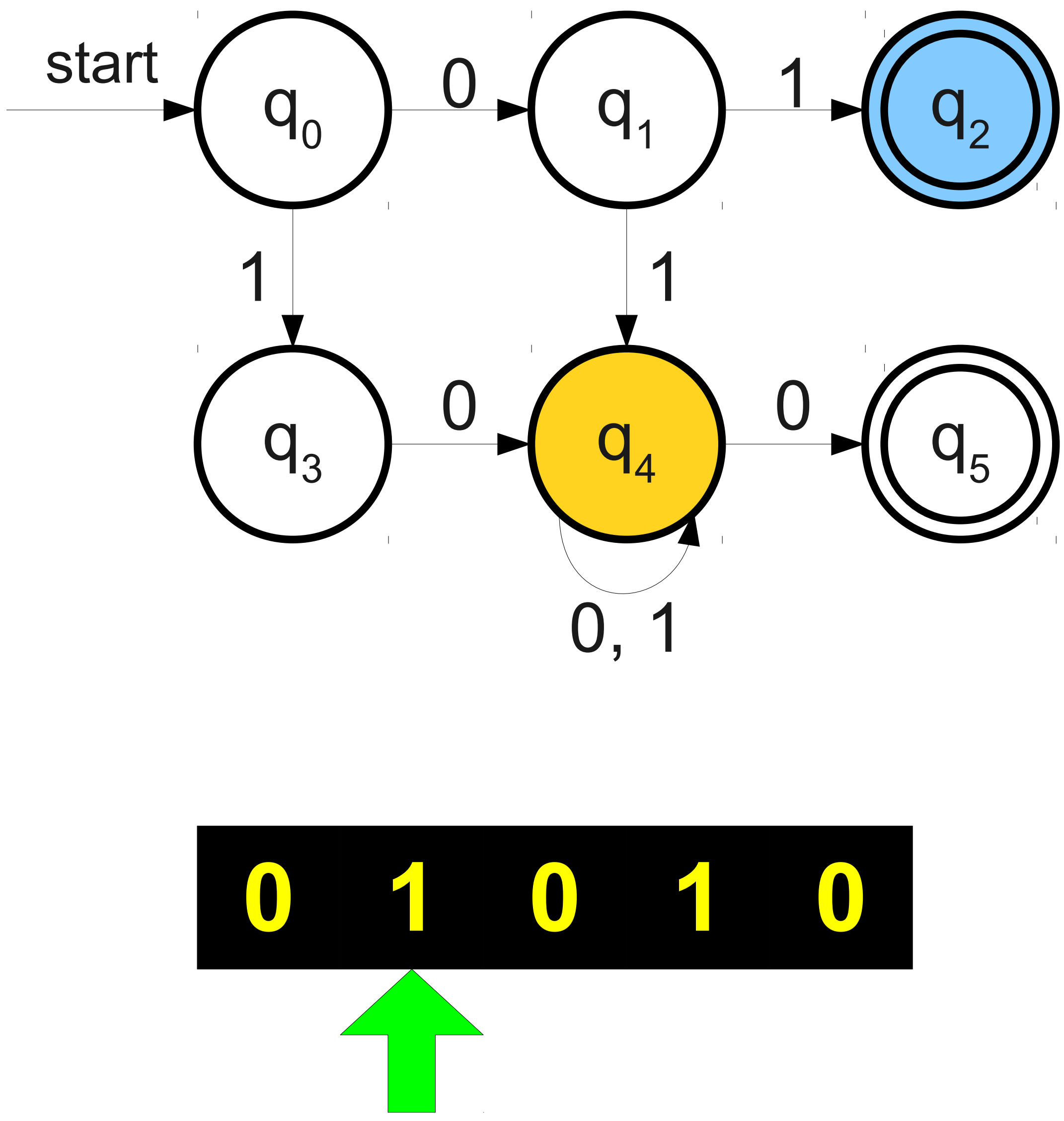
Massive Parallelism



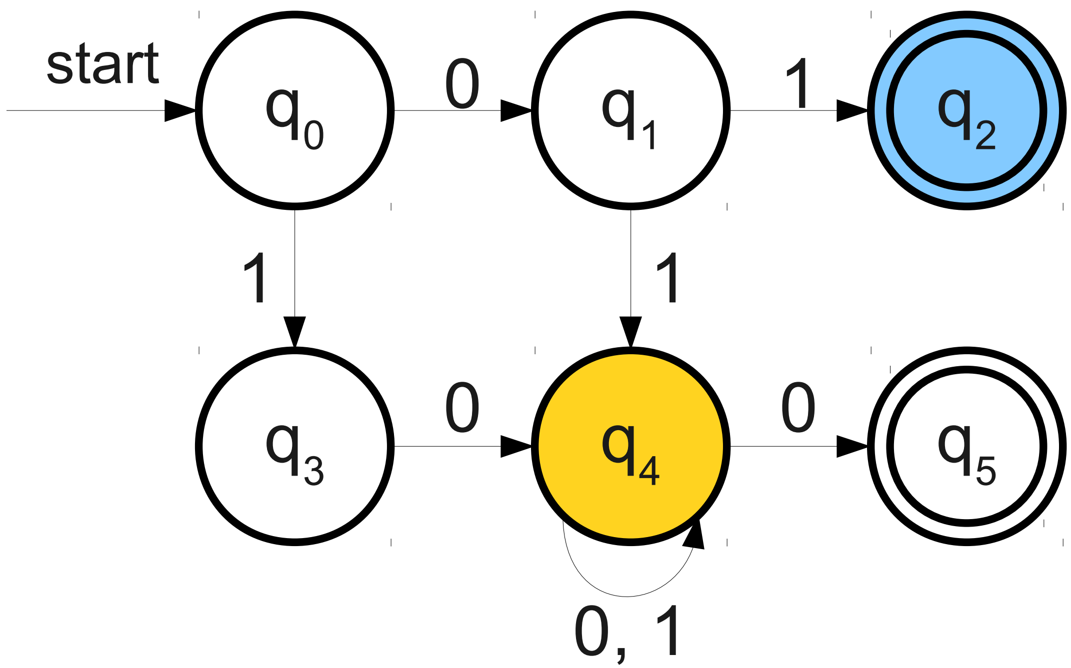
Massive Parallelism

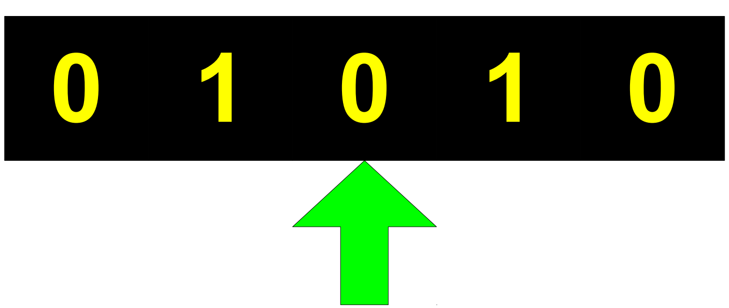


Massive Parallelism

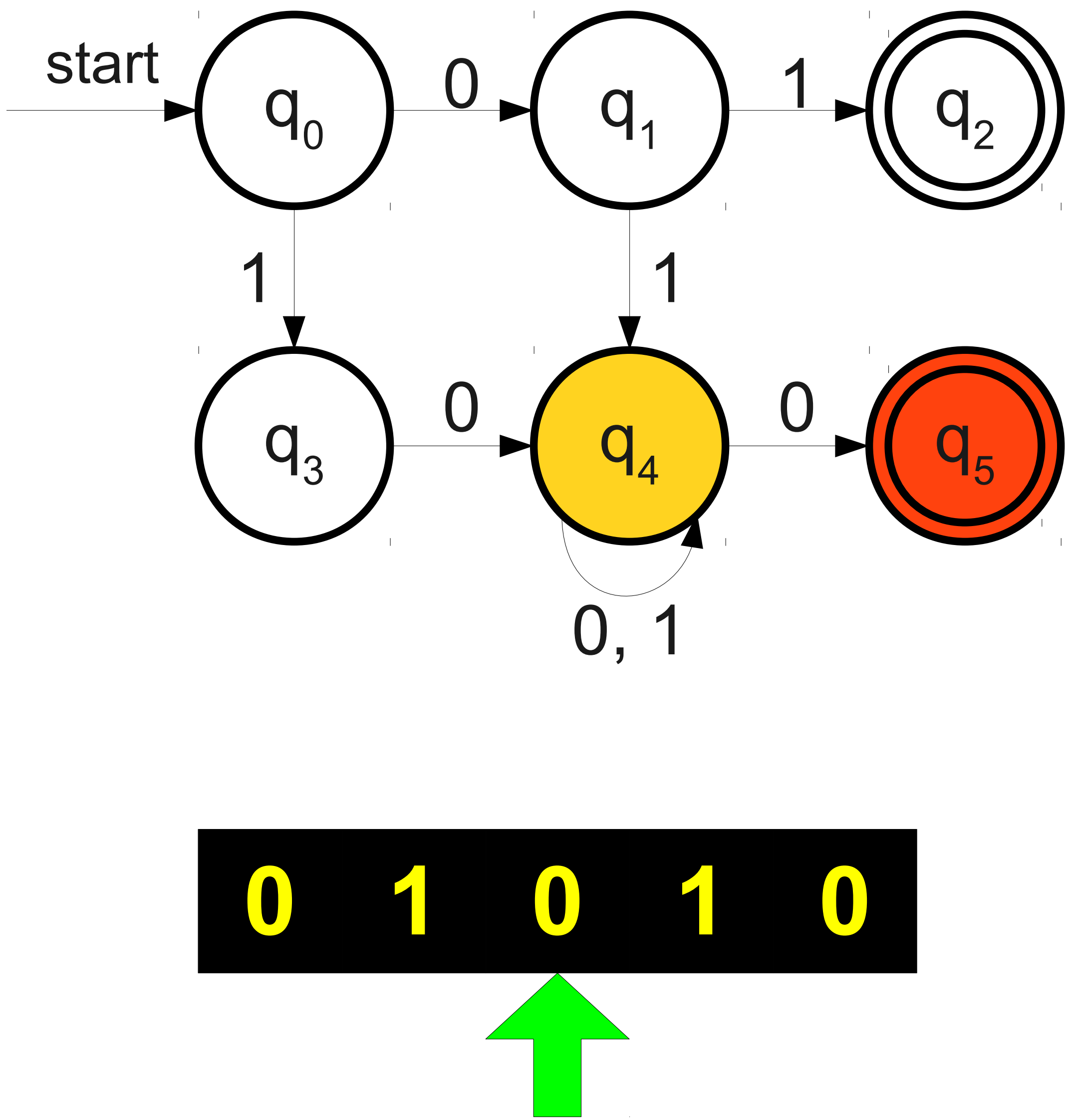


Massive Parallelism

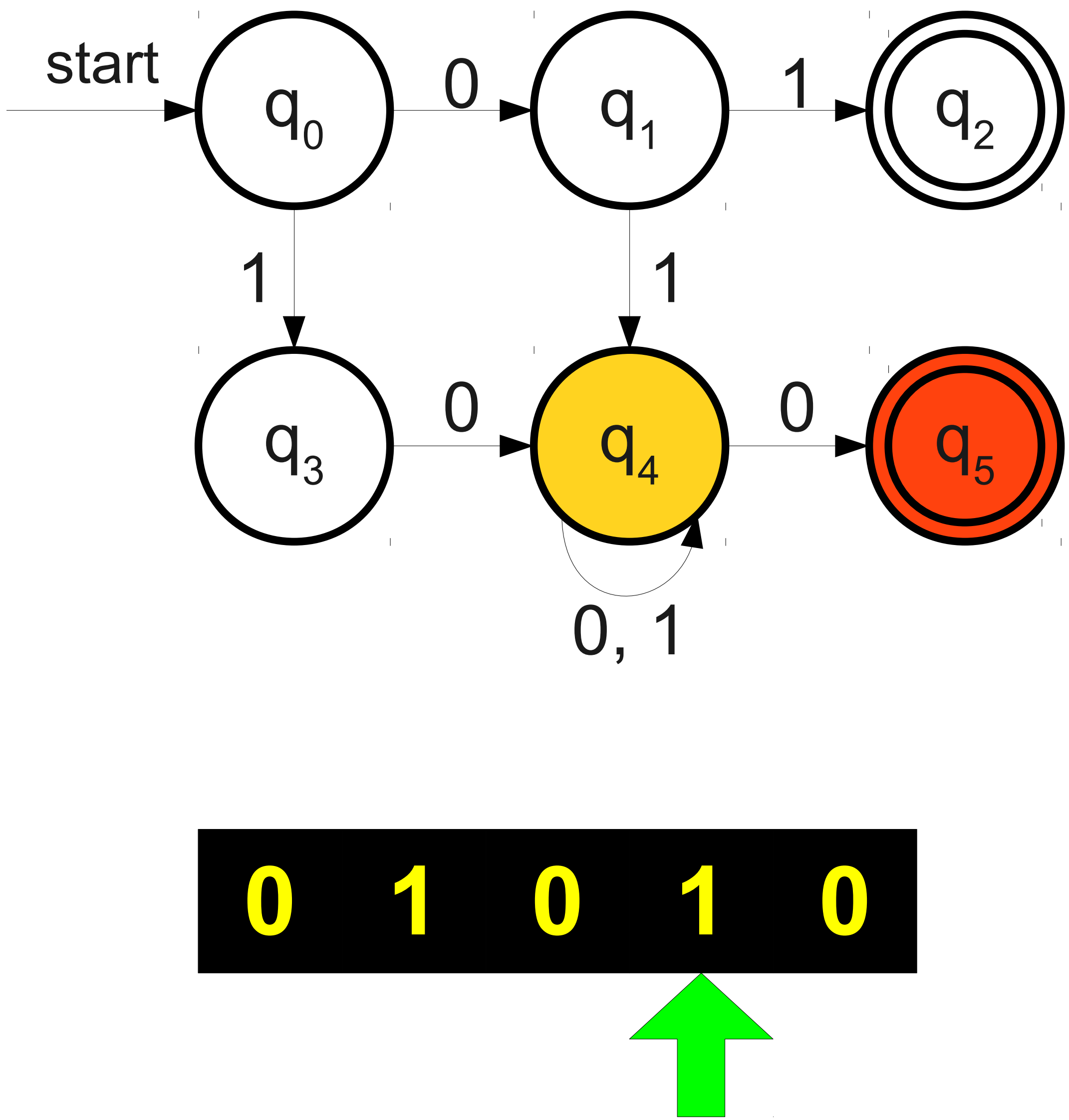




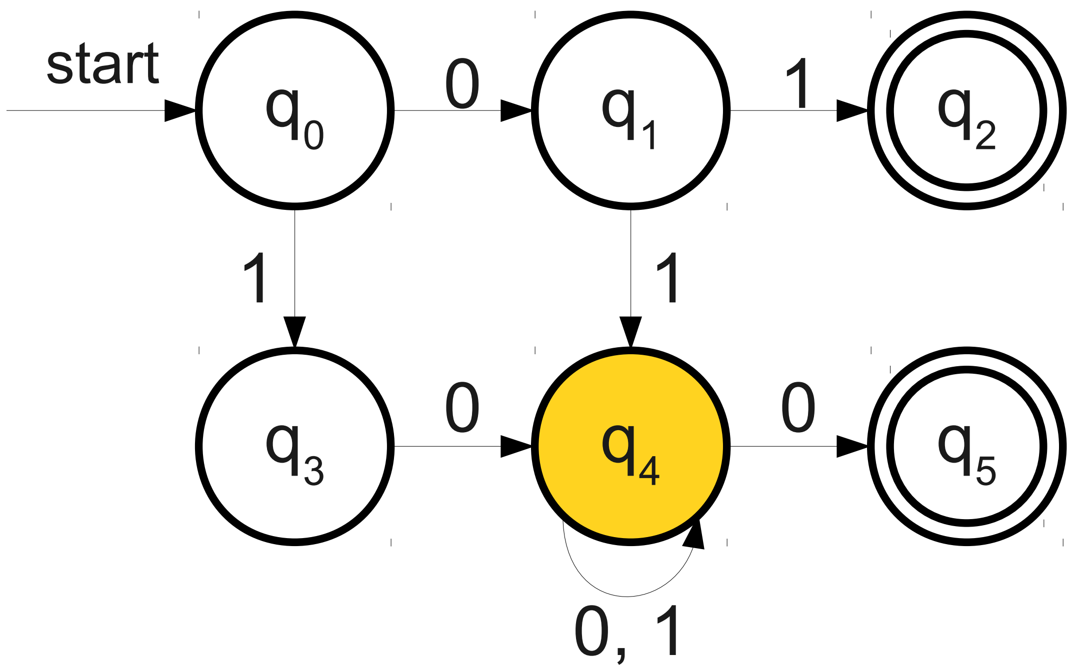
Massive Parallelism

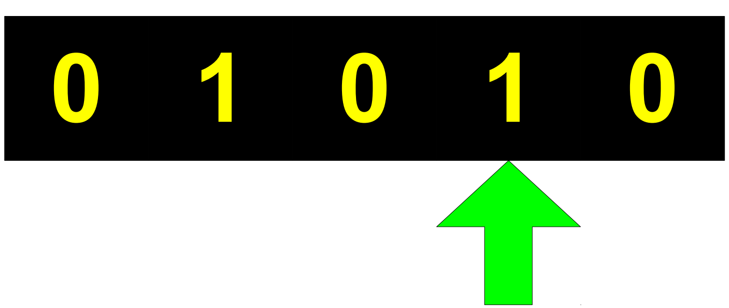


Massive Parallelism

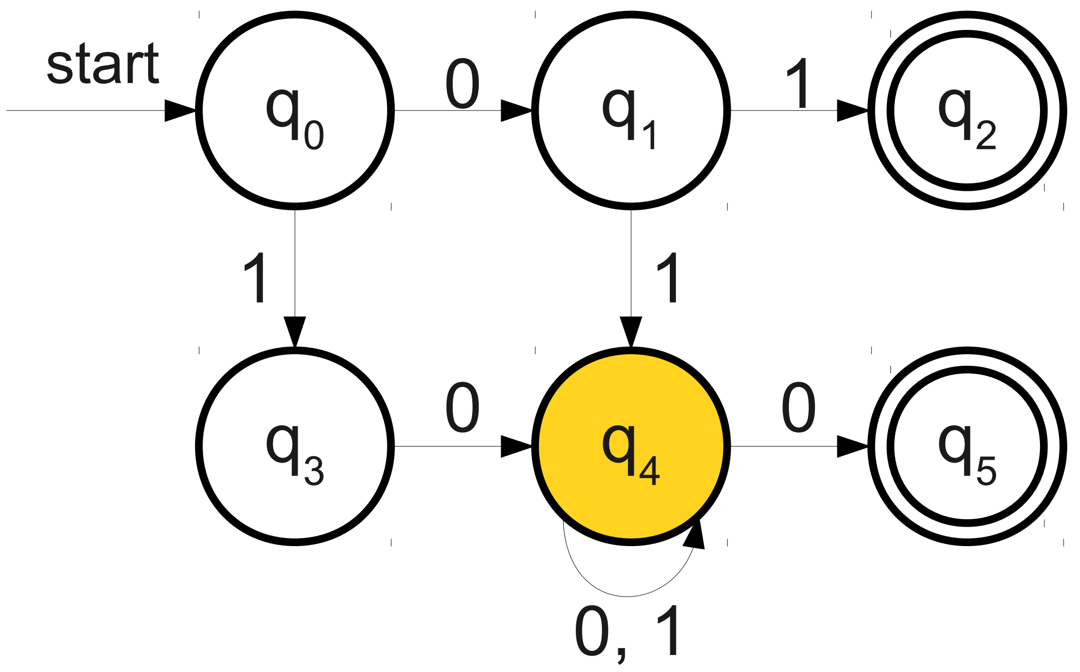


Massive Parallelism

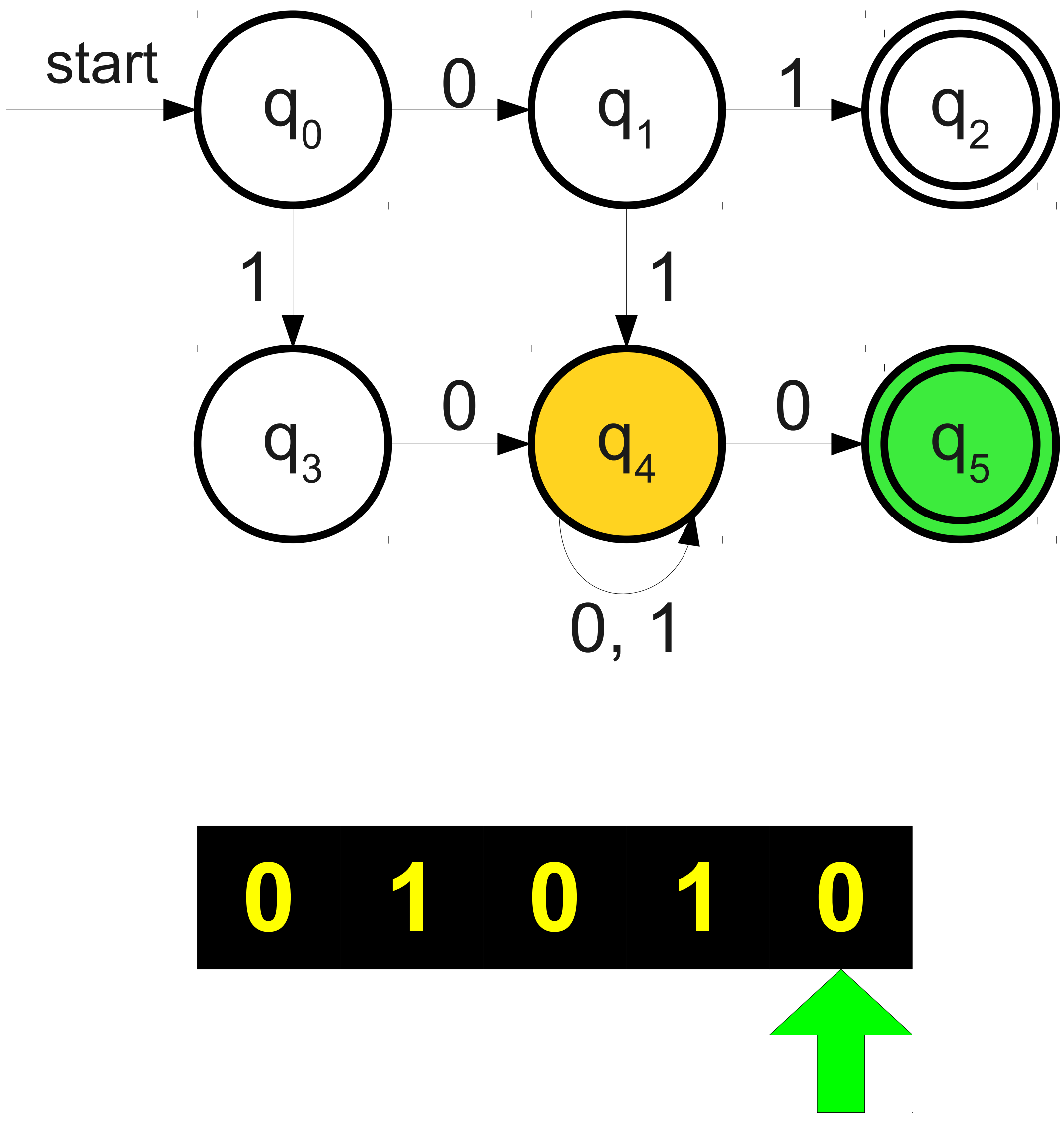




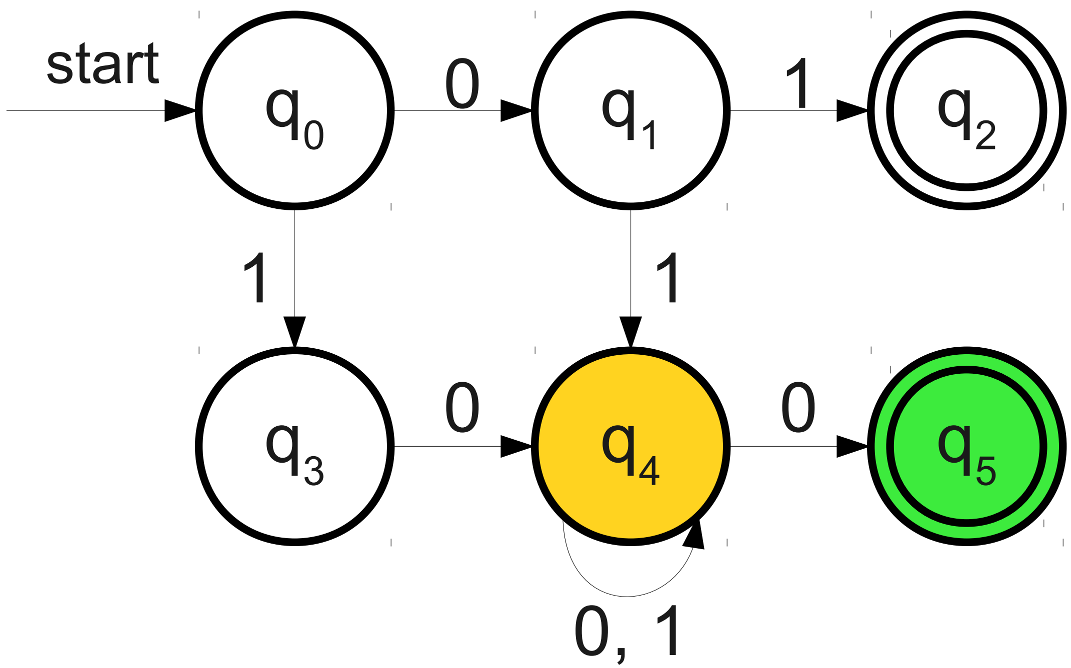
Massive Parallelism

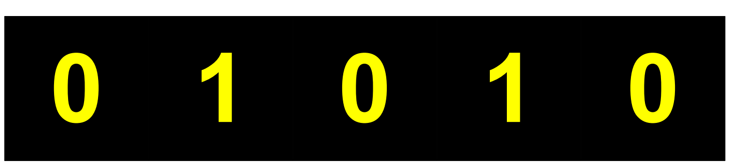


Massive Parallelism

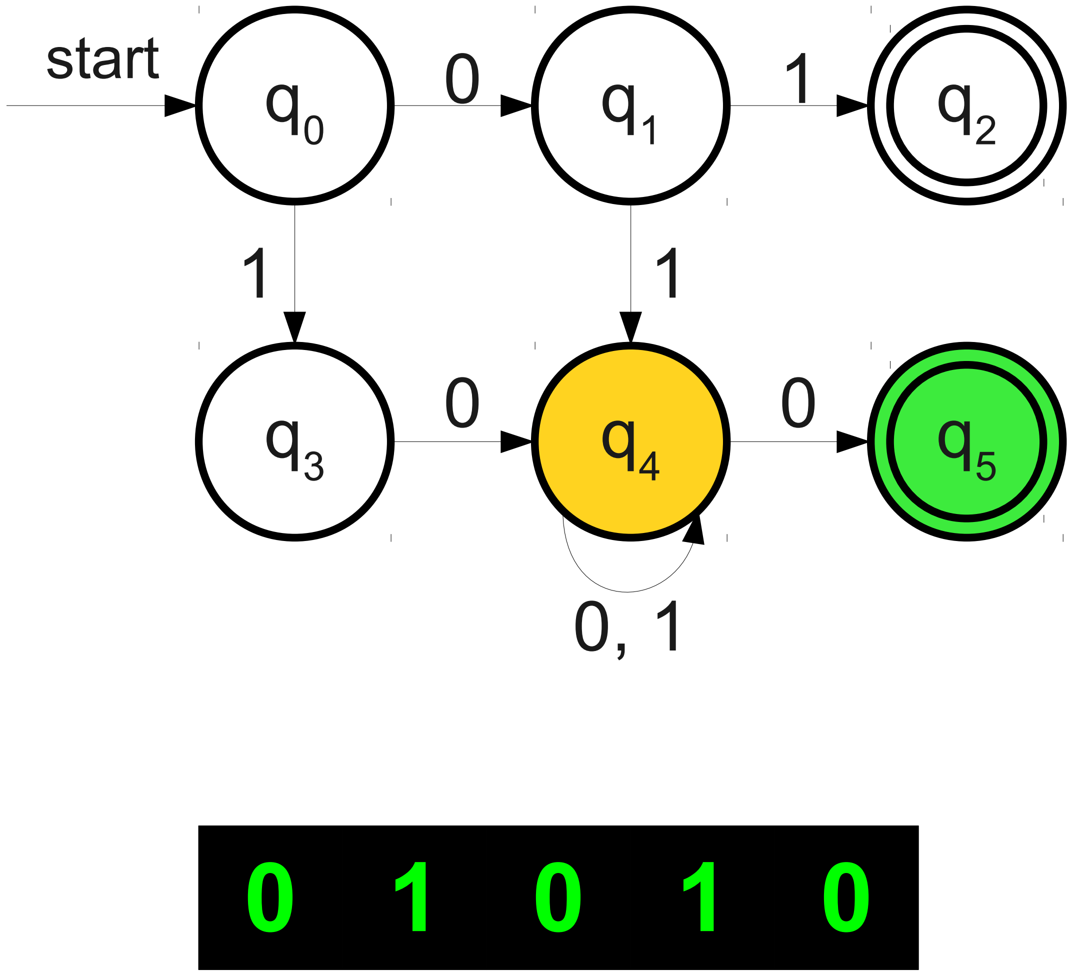


Massive Parallelism





Massive Parallelism



Massive Parallelism

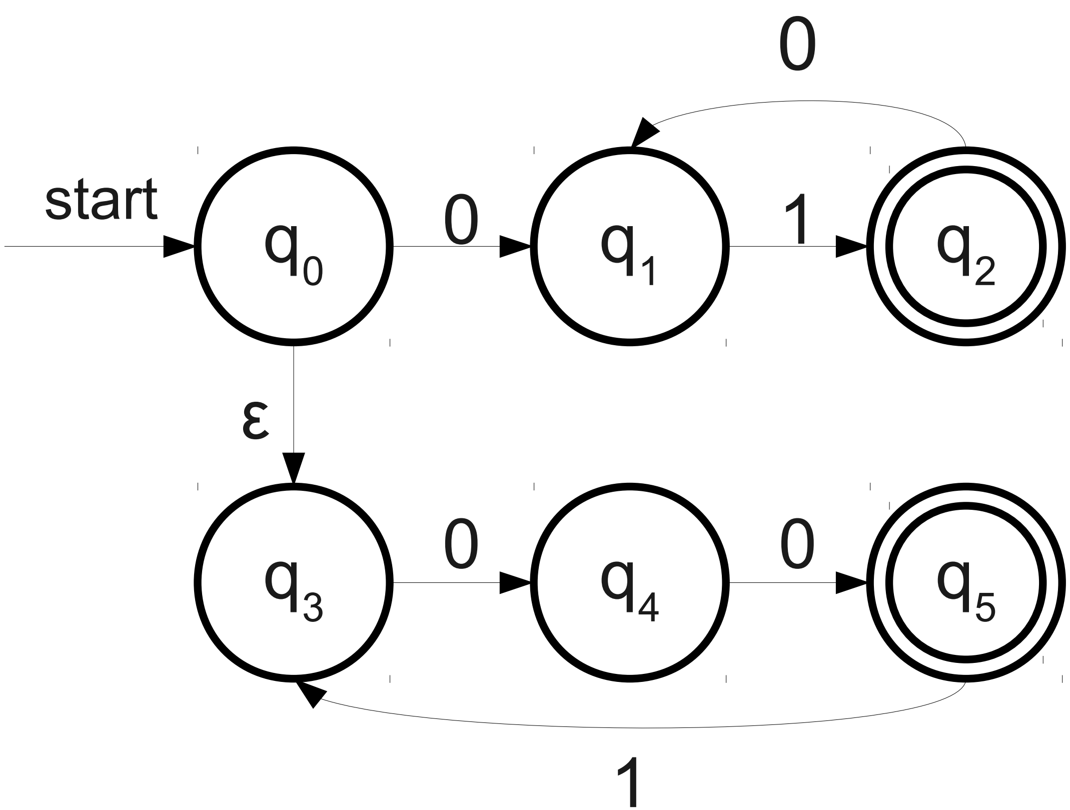
- An NFA can be thought of as a DFA that can be in many states at once.
- Each symbol read causes a transition on every active state into each potential state that could be visited.
- Nondeterministic machines can be thought of as machines that can try any number of options in parallel.
  - No fixed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

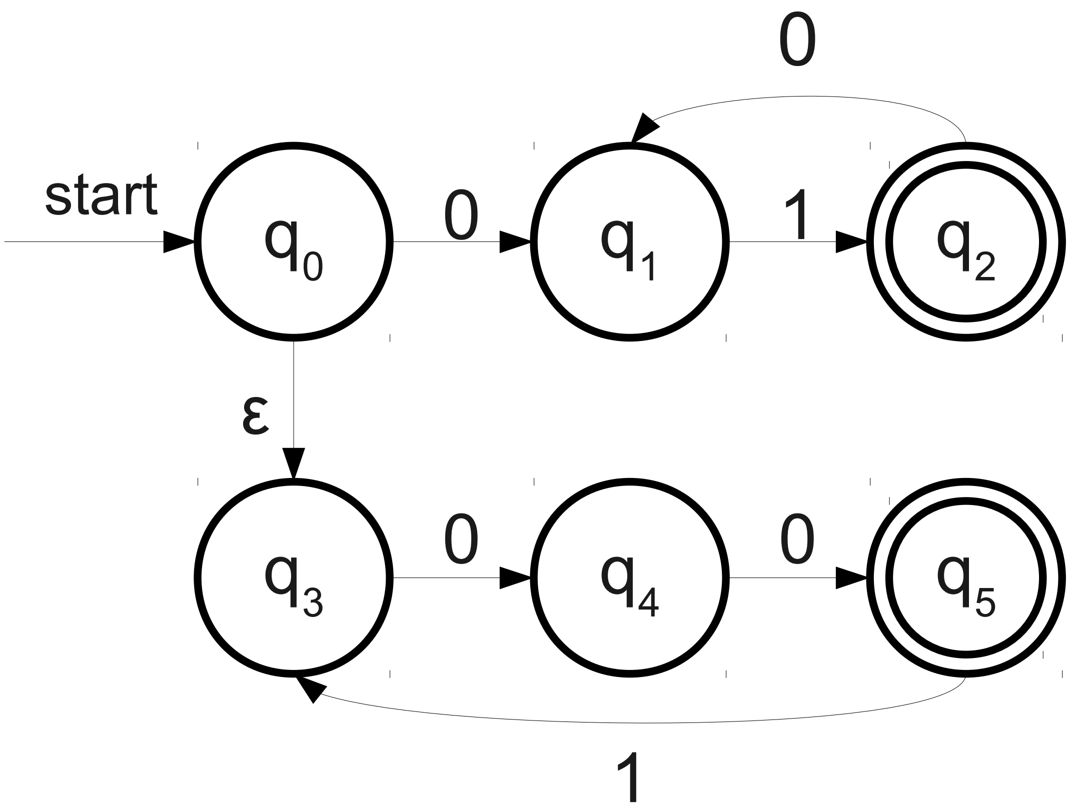
ε-Transitions

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



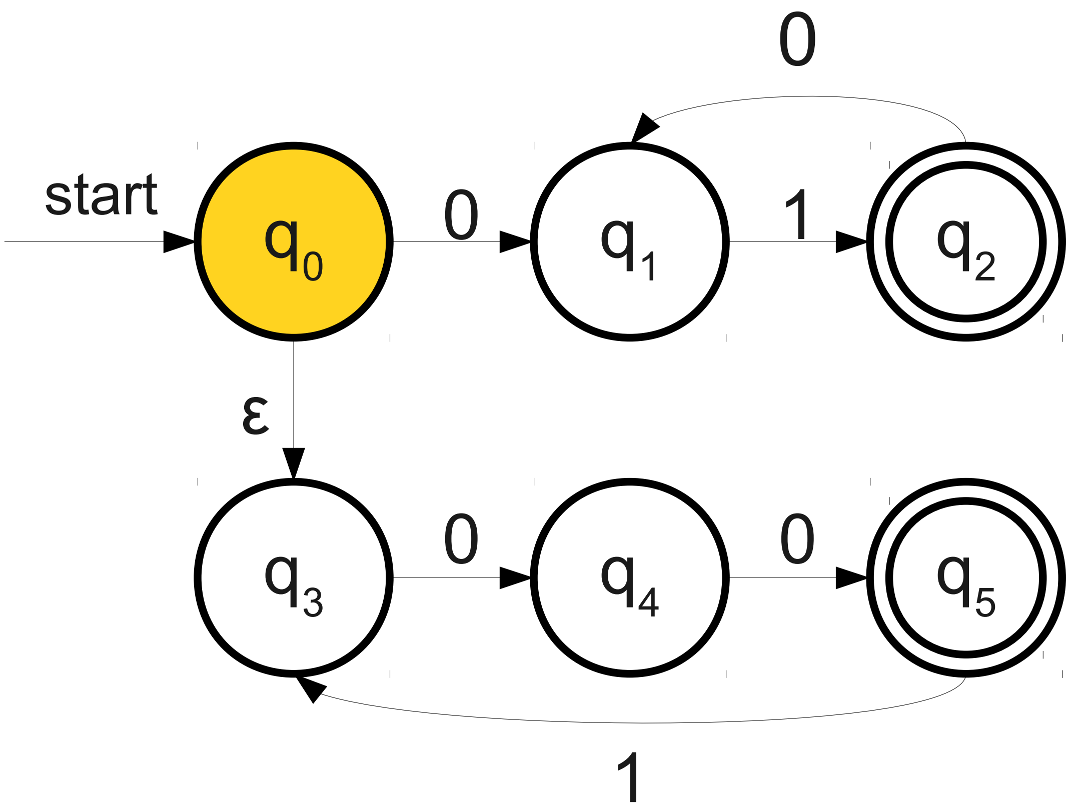
ε-Transitions

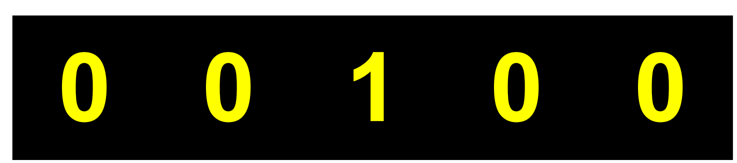
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



ε-Transitions

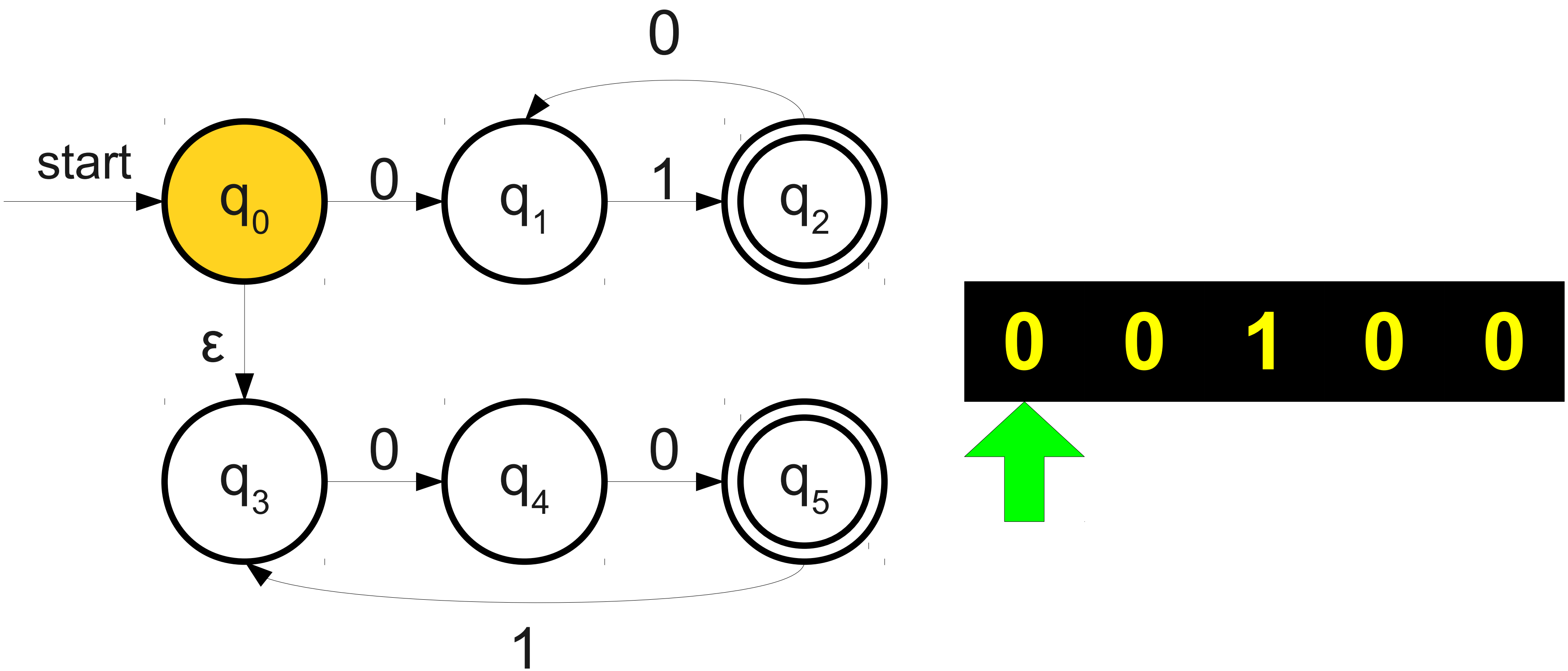
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.





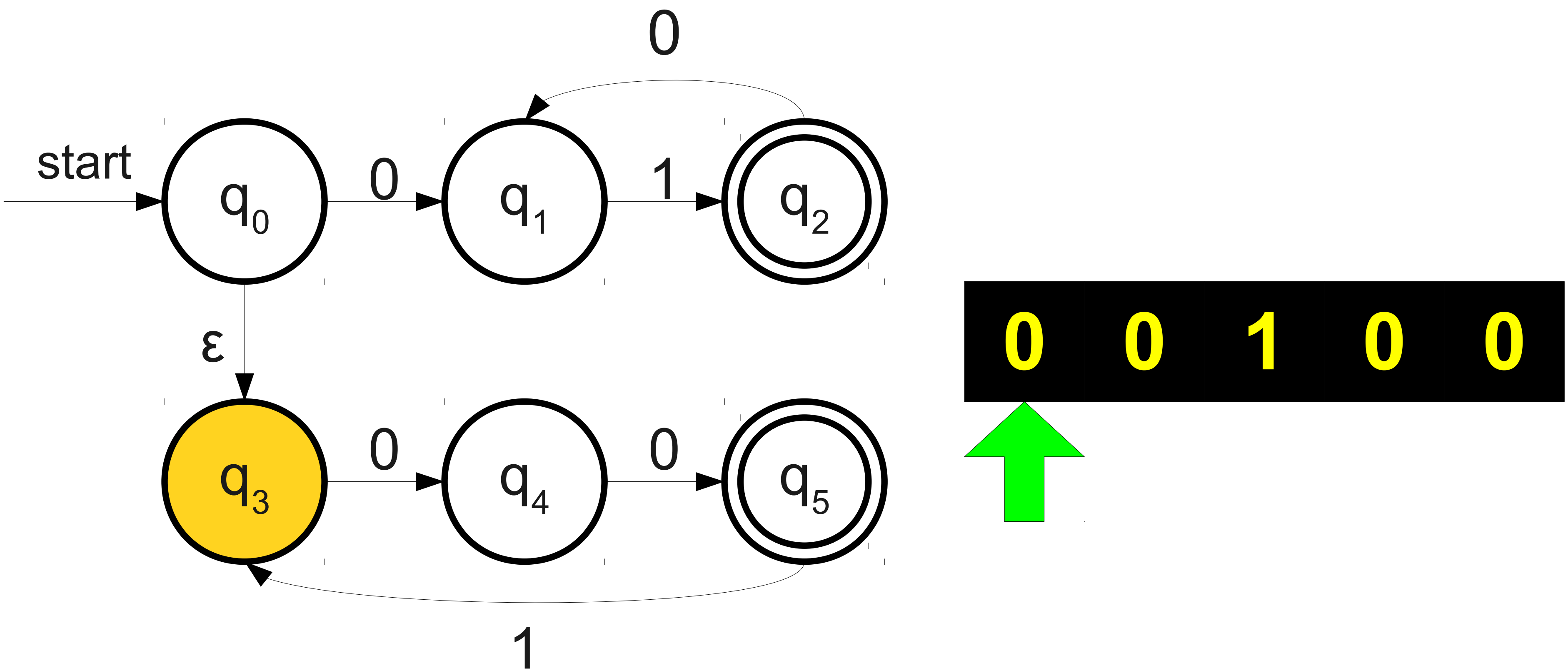
ε-Transitions

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



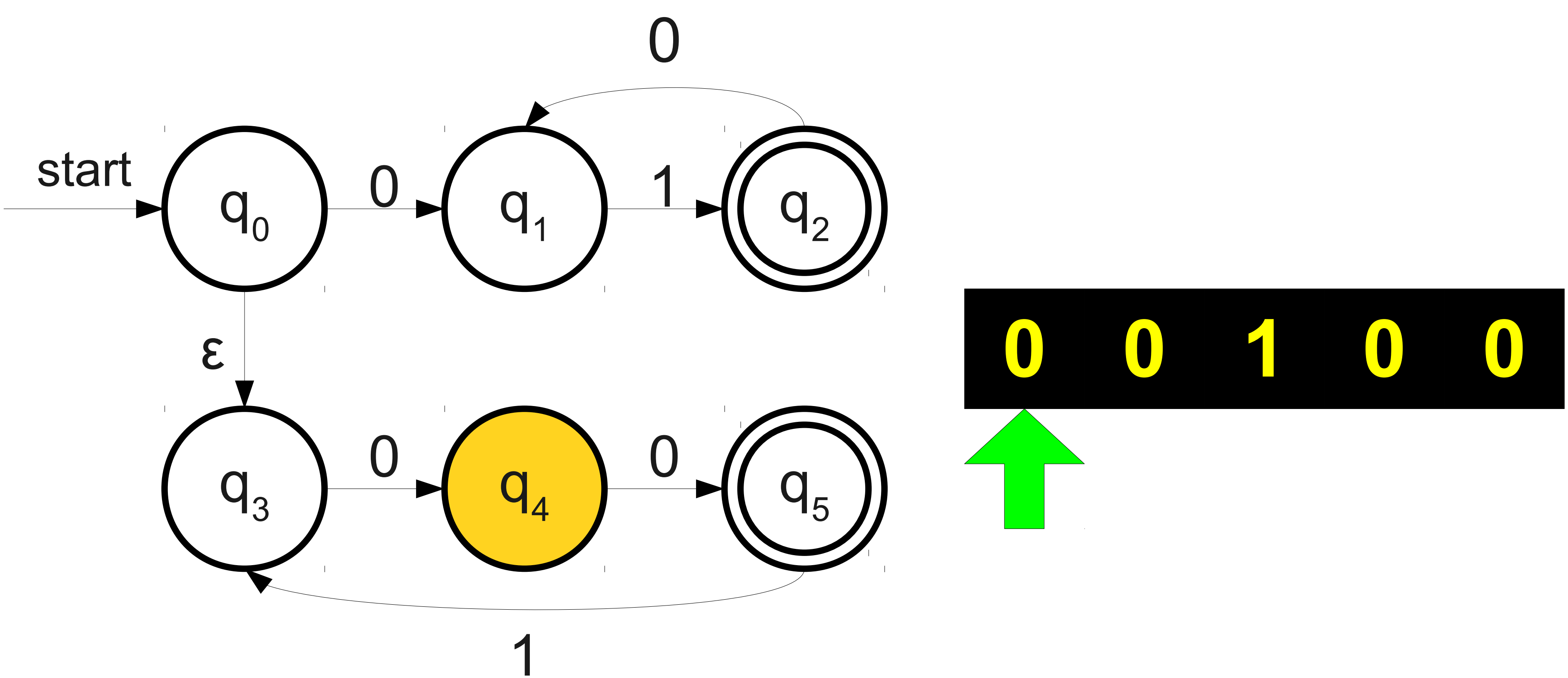
ε-Transitions

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



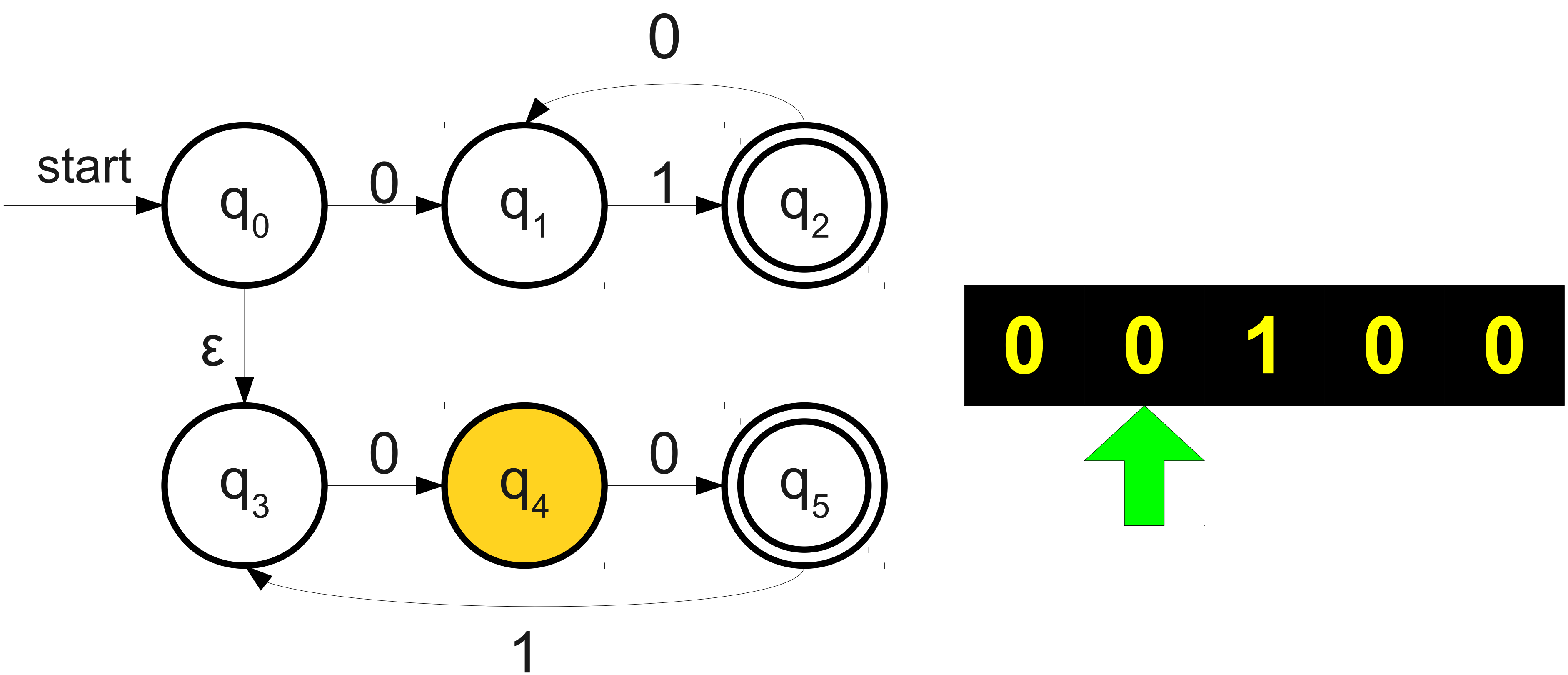
ε-Transitions

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



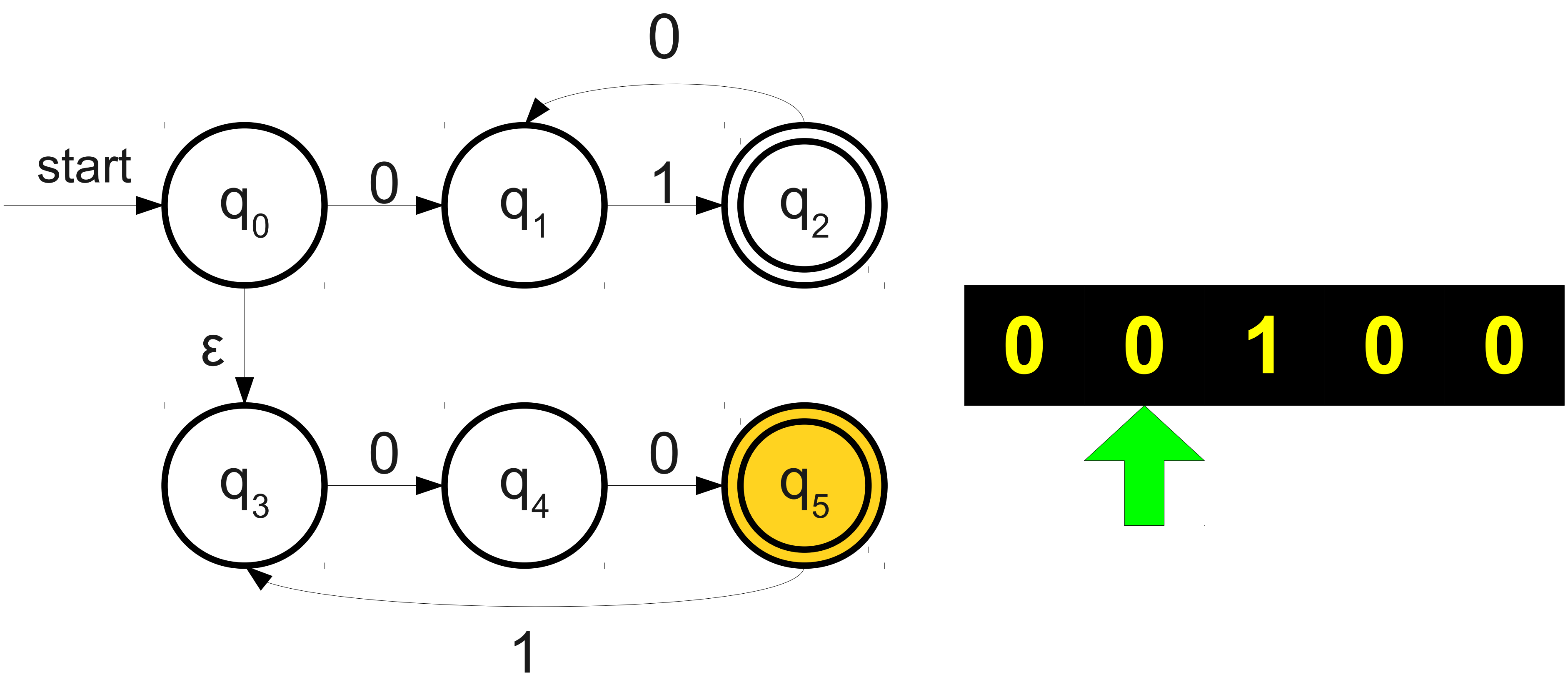
ε-Transitions

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



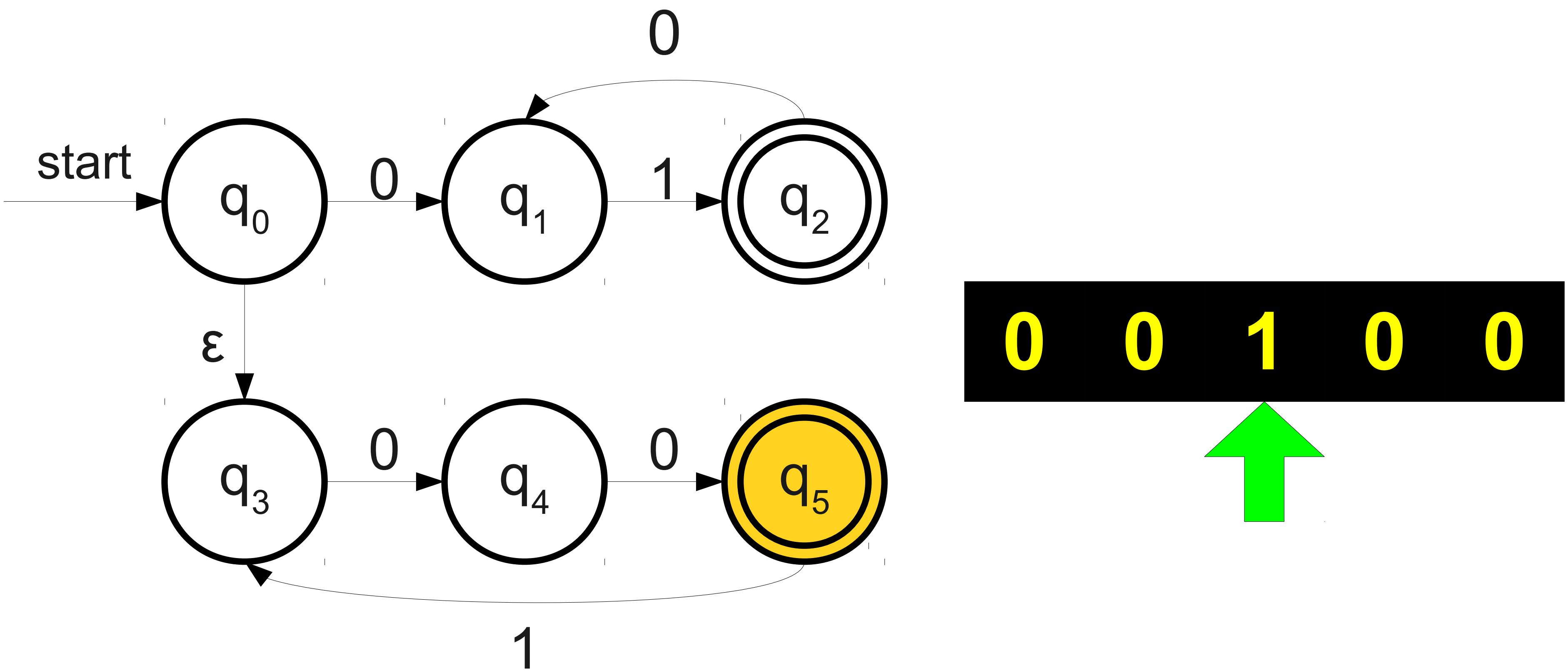
ε-Transitions

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



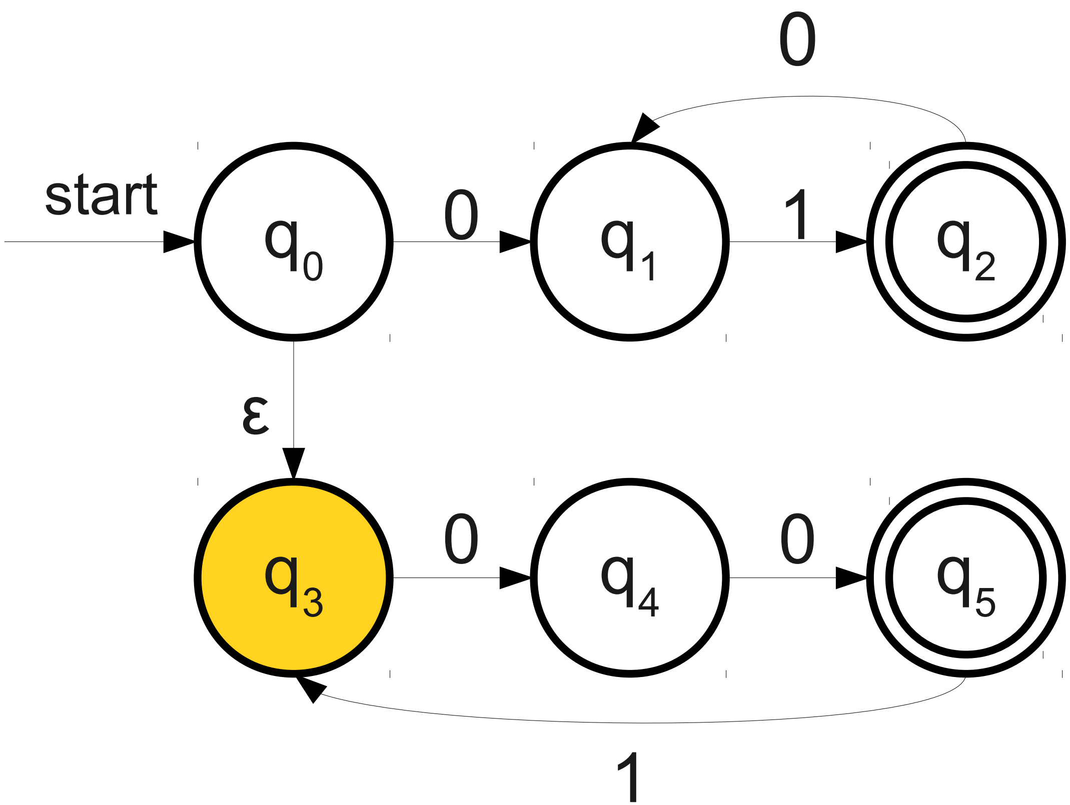
ε-Transitions

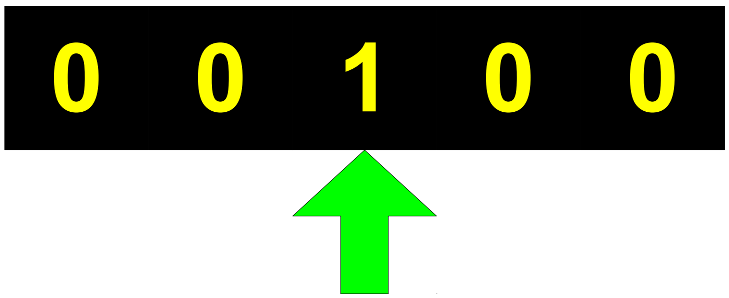
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.



ε-Transitions

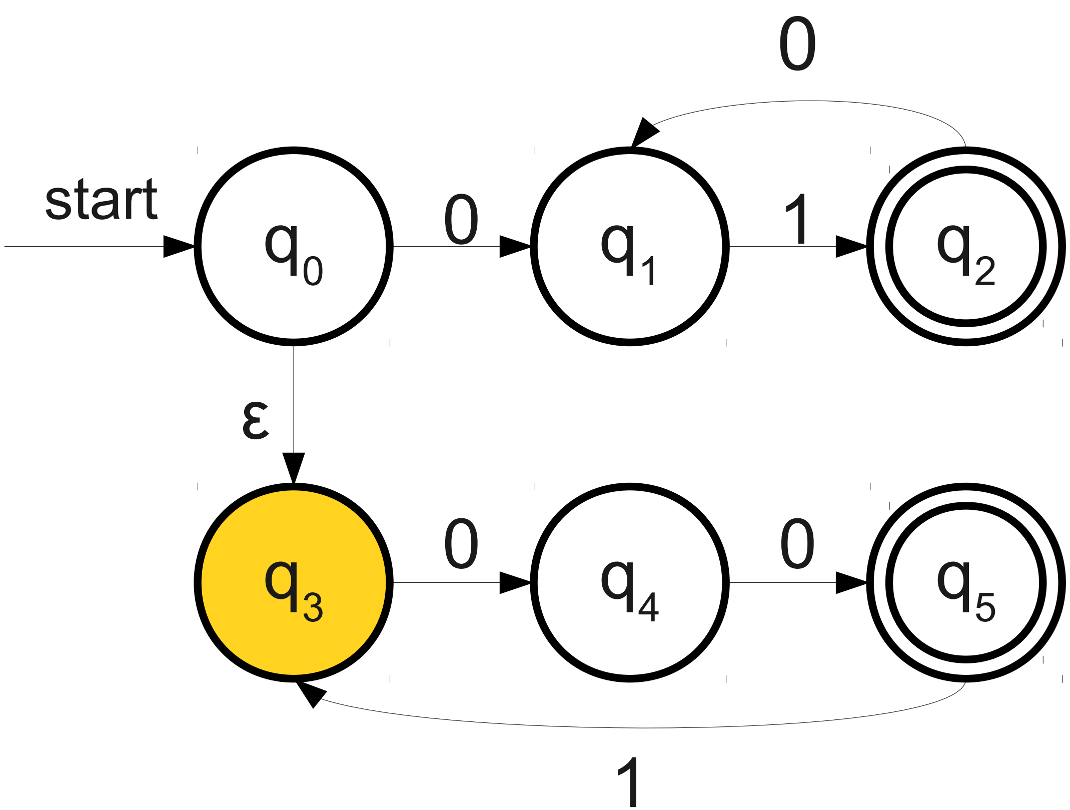
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.

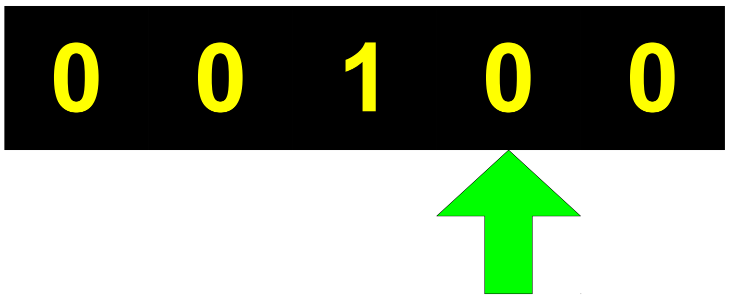




ε-Transitions

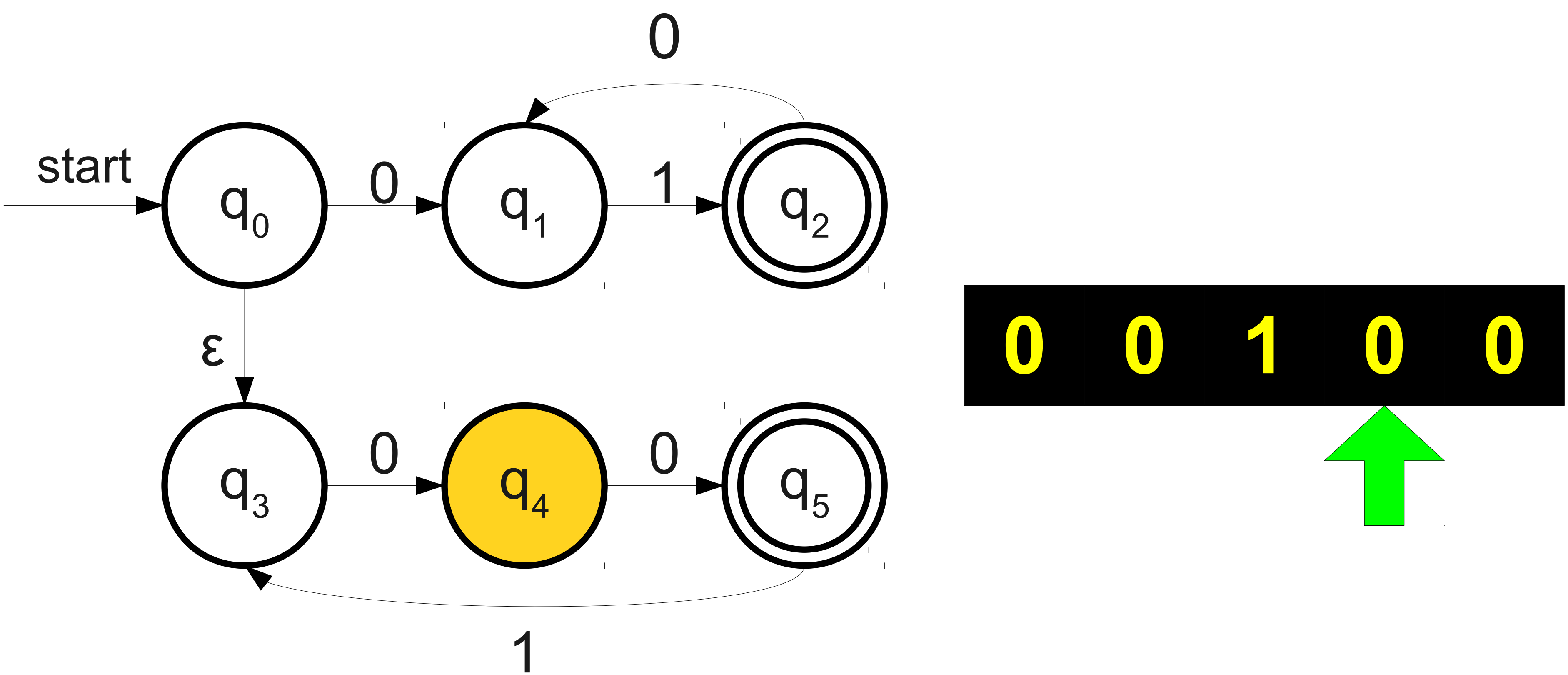
- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.





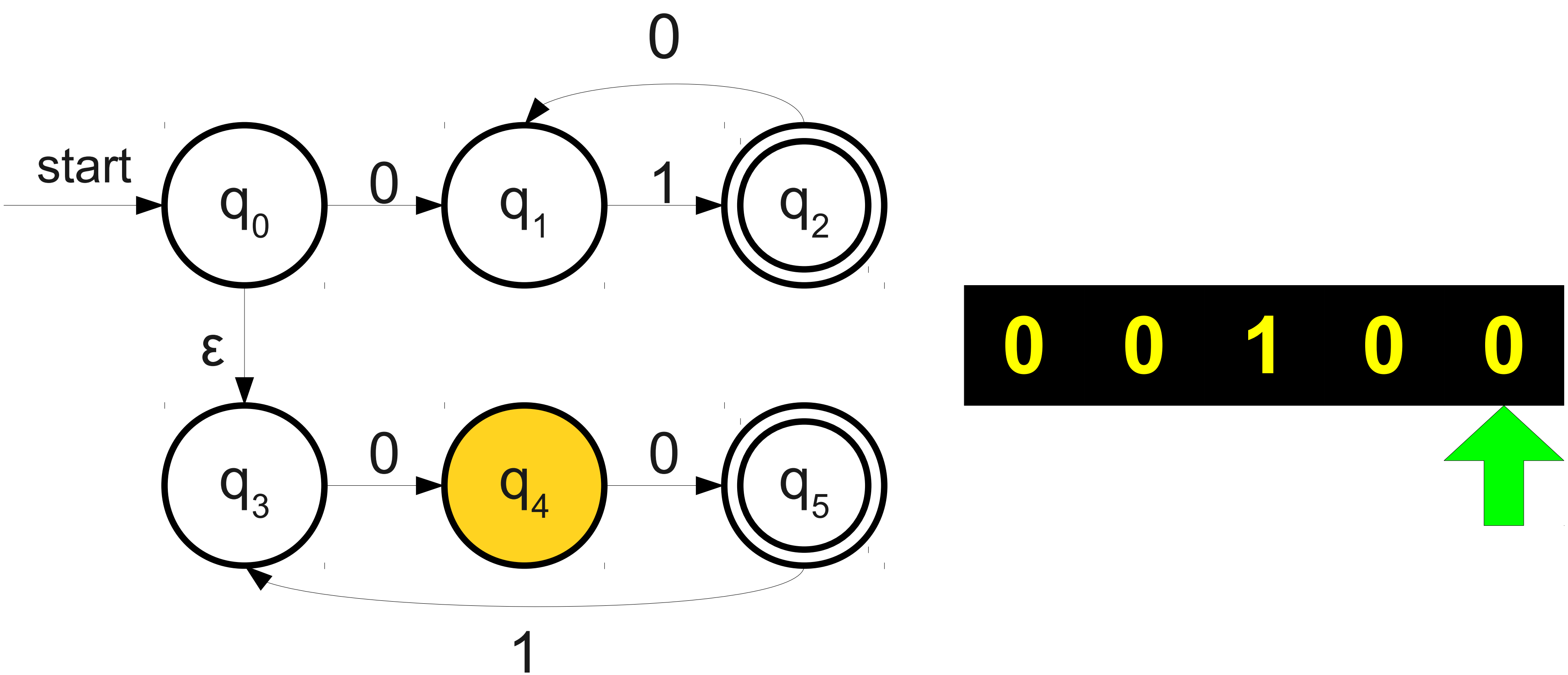
ε-Transitions

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



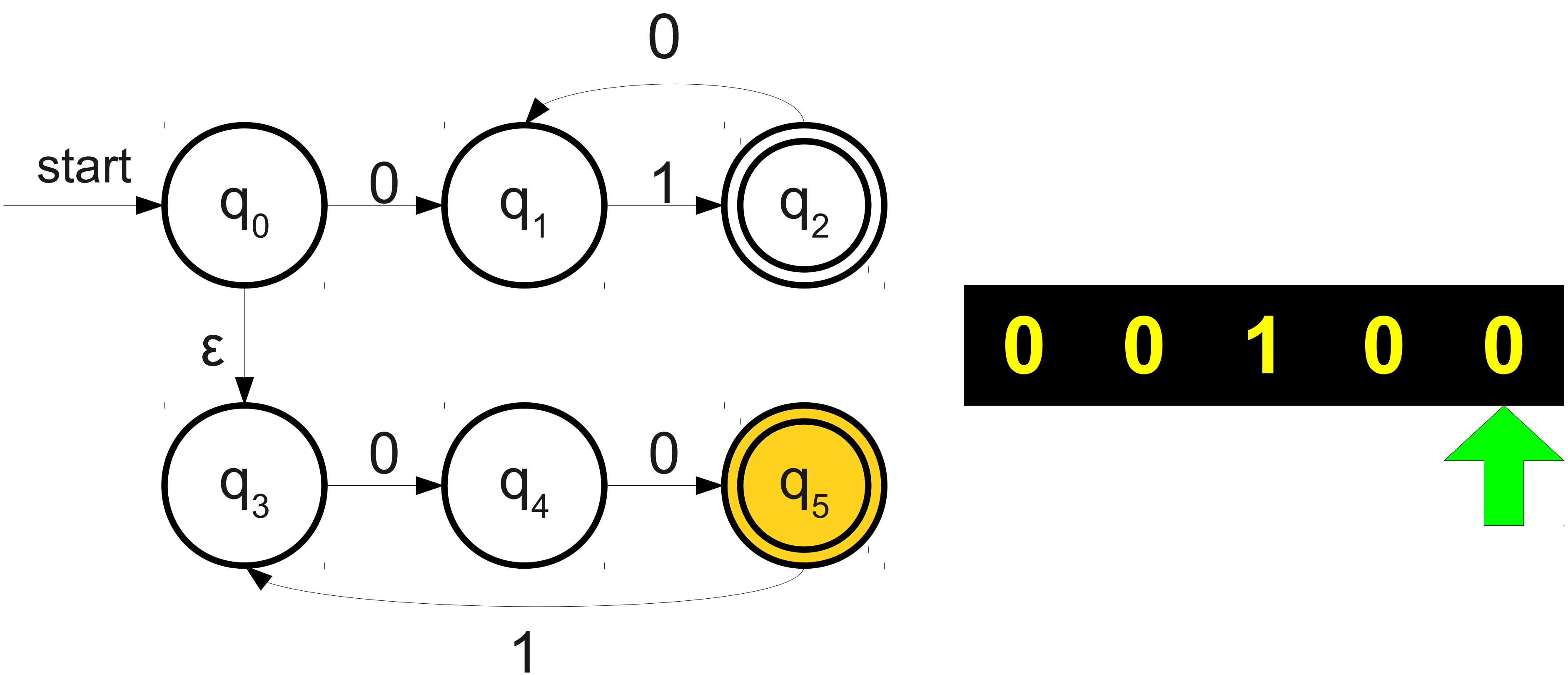
ε-Transitions

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



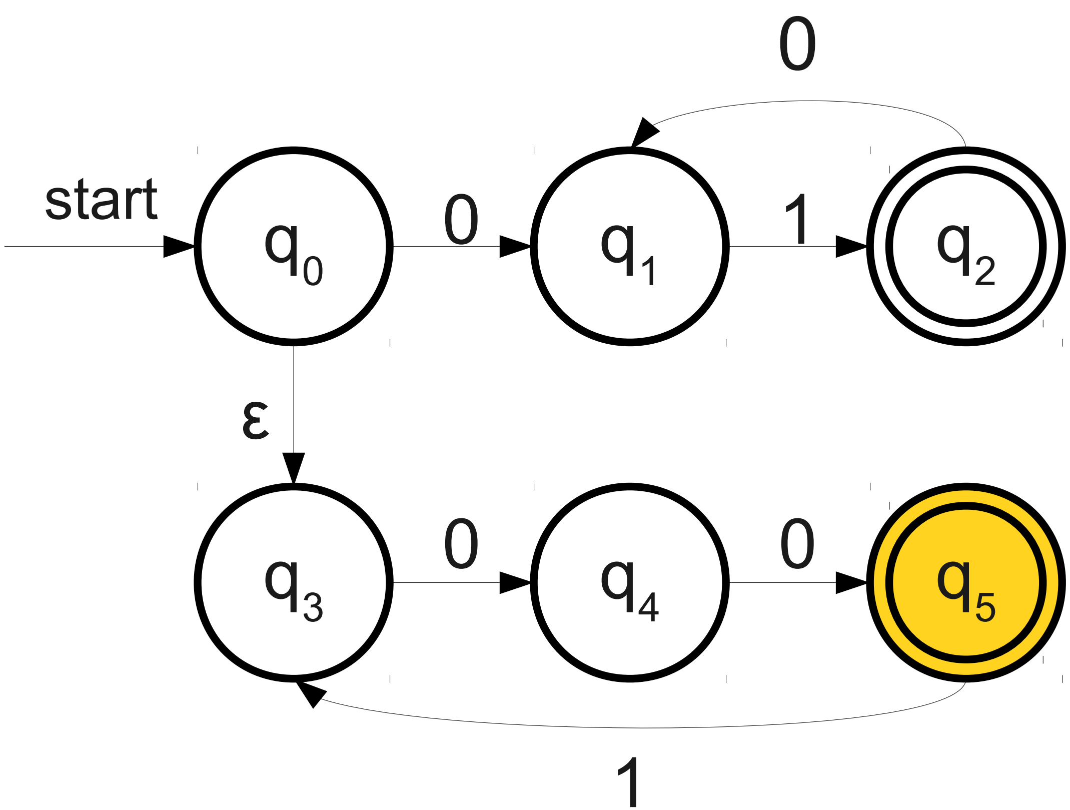
ε-Transitions

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



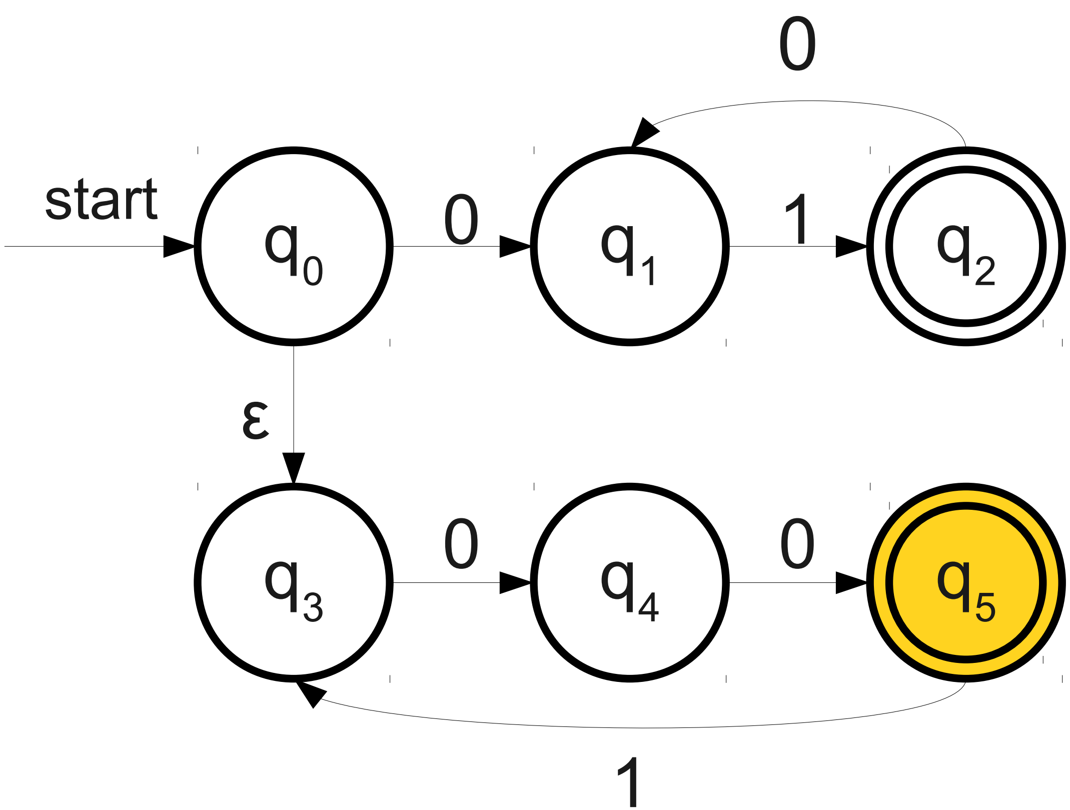
ε-Transitions

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



ε-Transitions

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



0	0	1	0	0

Designing NFAs

Designing NFAs

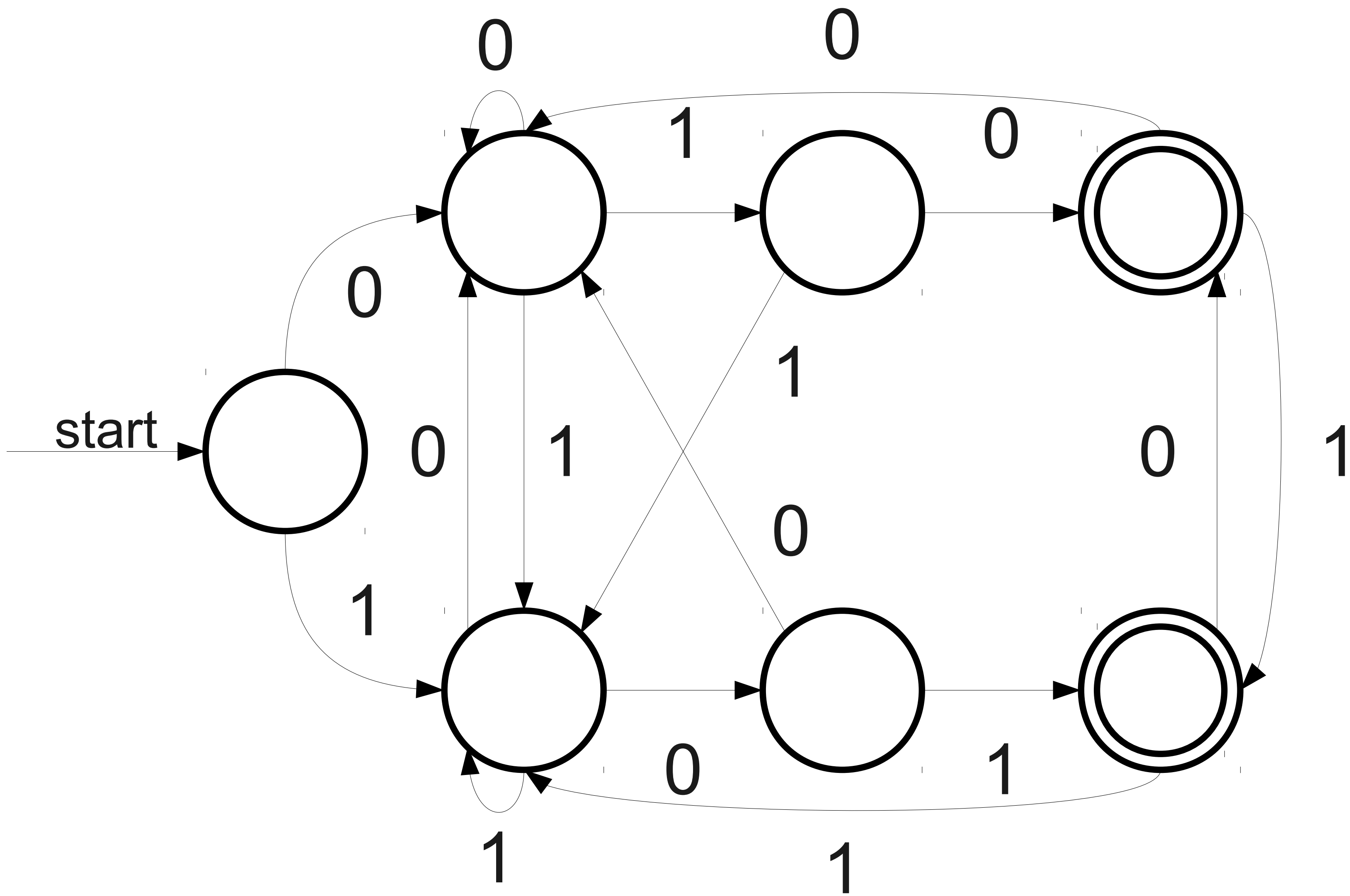
- When designing NFAs, *embrace the nondeterminism!*
- Good model: **Guess-and-check**:
  - Have the machine *nondeterministically guess* what the right choice is.
  - Have the machine *deterministically check* that the choice was correct.

Guess-and-Check

$L = \{ w \in \{\texttt{0}, \texttt{1}\}^* \mid w \text{ ends in } \texttt{010} \text{ or } \texttt{101} \}$

Guess-and-Check

$L = \{ w \in \{\texttt{0}, \texttt{1}\}^* \mid w \text{ ends in } \texttt{010} \text{ or } \texttt{101} \}$

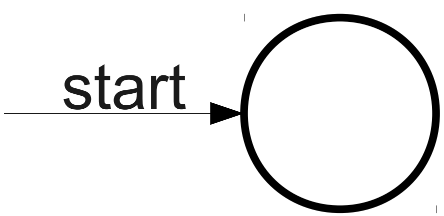


Guess-and-Check

$L = \{ w \in \{\texttt{0}, \texttt{1}\}^* \mid w \text{ ends in } \texttt{010} \text{ or } \texttt{101} \}$

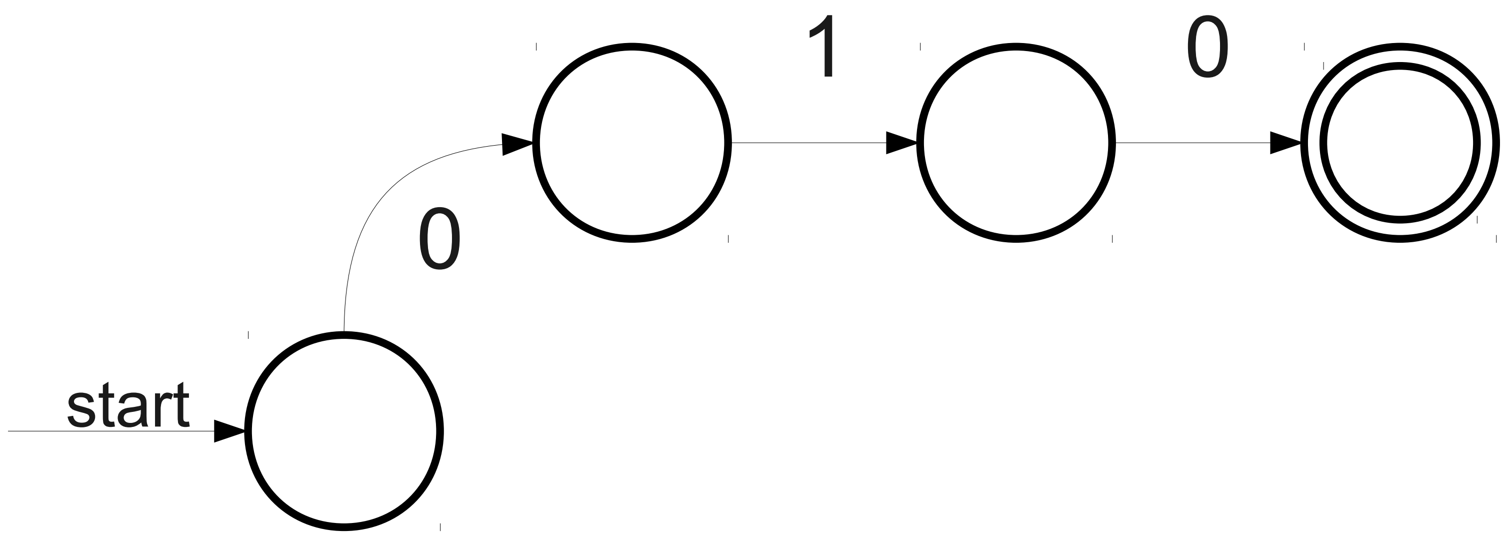
Guess-and-Check

$$L = \{\ w \in \{\mathtt{0}, \mathtt{1}\}^* \mid w \text{ ends in } \mathtt{010} \text{ or } \mathtt{101}\ \}$$



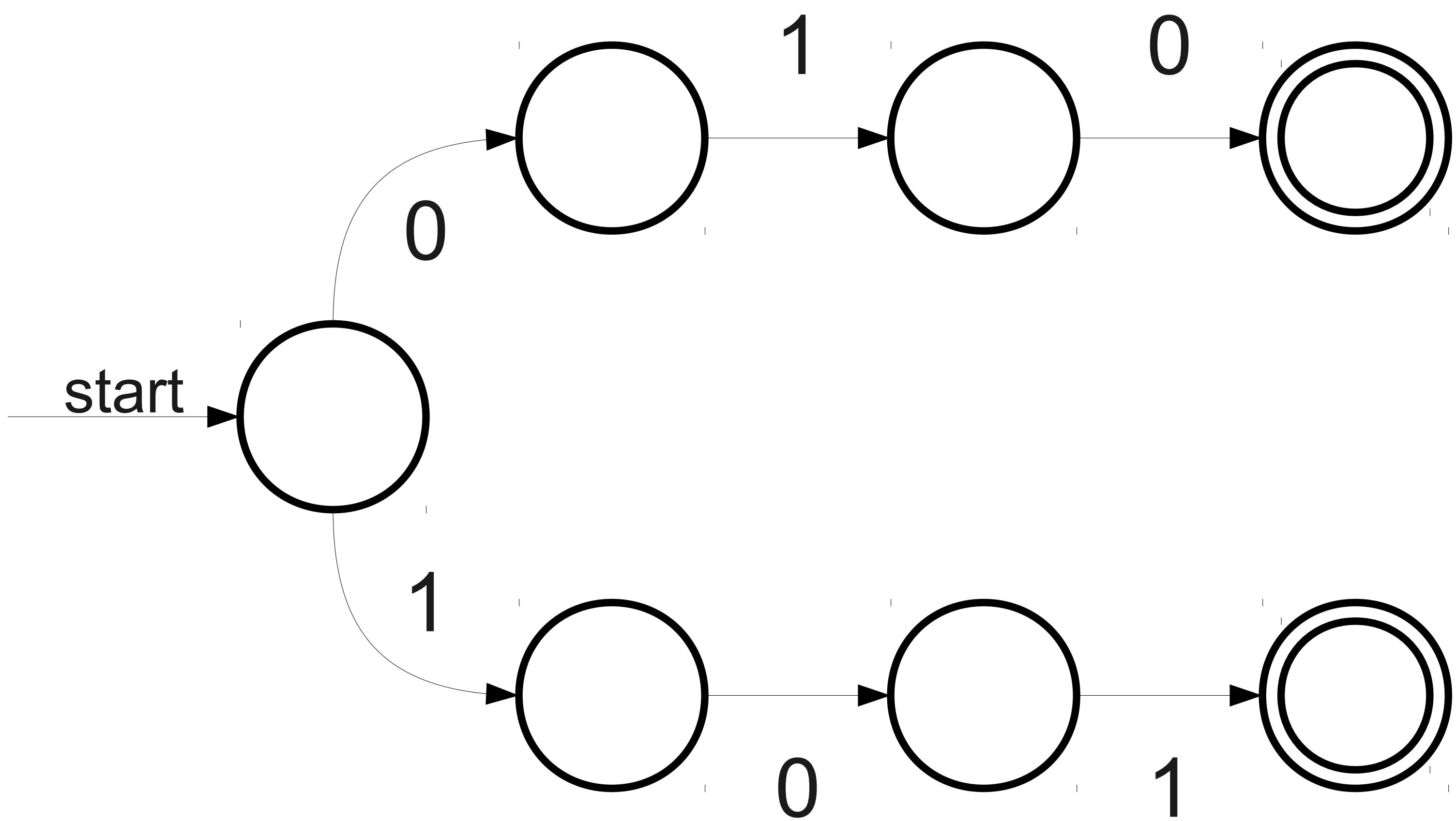
Guess-and-Check

$L = \{ w \in \{\mathtt{0}, \mathtt{1}\}^* \mid w \text{ ends in } \mathtt{010} \text{ or } \mathtt{101} \}$



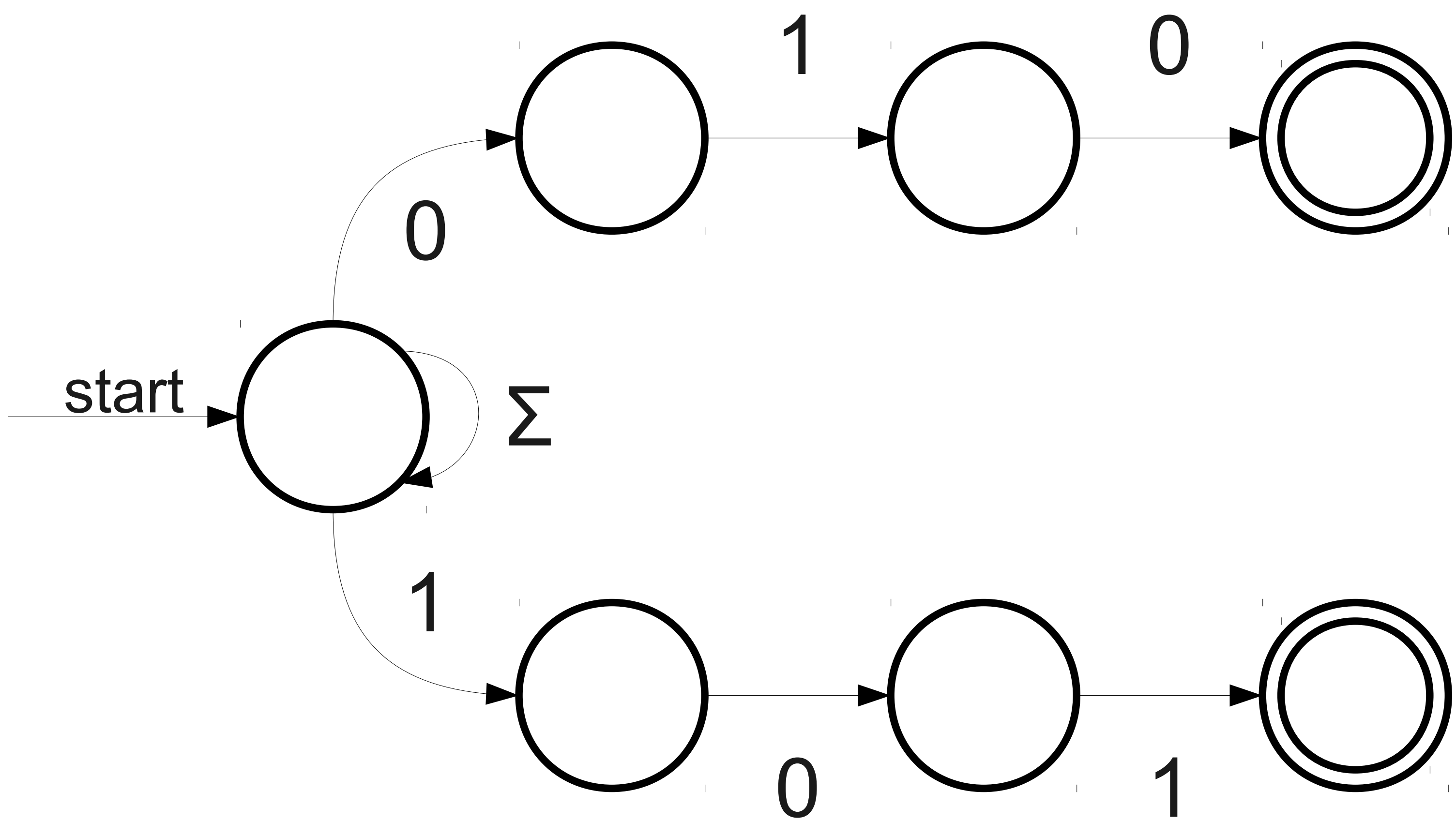
Guess-and-Check

$$L = \{\ w \in \{\texttt{0}, \texttt{1}\}^* \mid w \text{ ends in } \texttt{010} \text{ or } \texttt{101}\ \}$$



Guess-and-Check

$L = \{ w \in \{\texttt{0}, \texttt{1}\}^* \mid w \text{ ends in } \texttt{010} \text{ or } \texttt{101} \}$

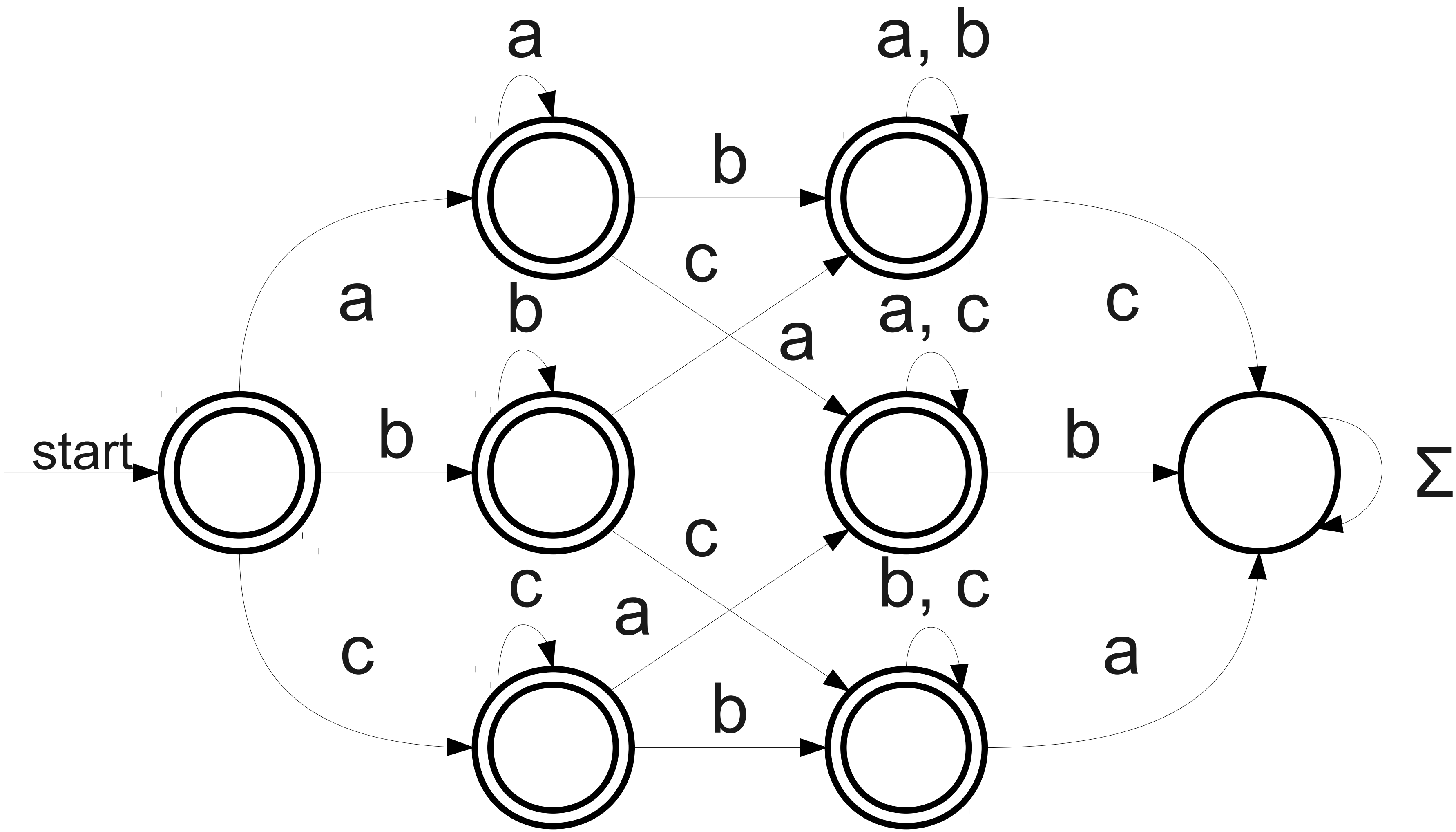


Guess-and-Check

$L = \{ w \in \{\texttt{a}, \texttt{b}, \texttt{c}\}^* \mid \text{at least one of } \texttt{a}, \texttt{b}, \text{ or } \texttt{c} \text{ is not in } w \}$

Guess-and-Check

$L = \{ w \in \{$**a**, **b**, **c**$\}^* \mid$ at least one of **a**, **b**, or **c** is not in w $\}$

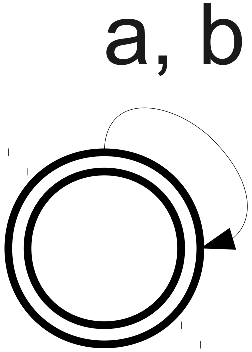


Guess-and-Check

$L = \{ w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid \text{at least one of } \mathtt{a}, \mathtt{b}, \text{ or } \mathtt{c} \text{ is not in } w \}$

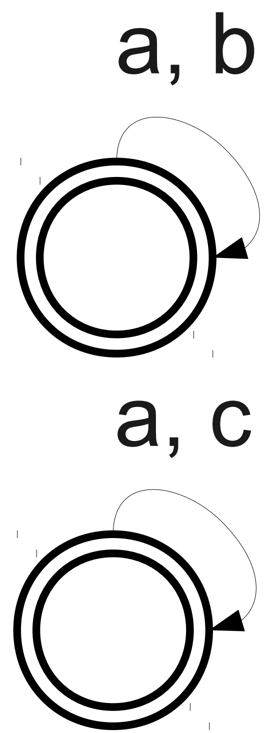
Guess-and-Check

$L = \{\ w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid \text{at least one of } \mathtt{a}, \mathtt{b}, \text{or } \mathtt{c} \text{ is not in } w\ \}$



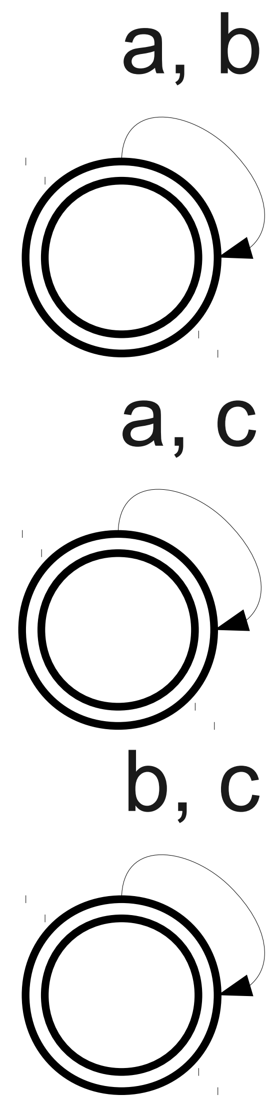
Guess-and-Check

$L = \{ w \in \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}^* \mid \text{at least one of } \mathbf{a}, \mathbf{b}, \text{or } \mathbf{c} \text{ is not in } w \}$



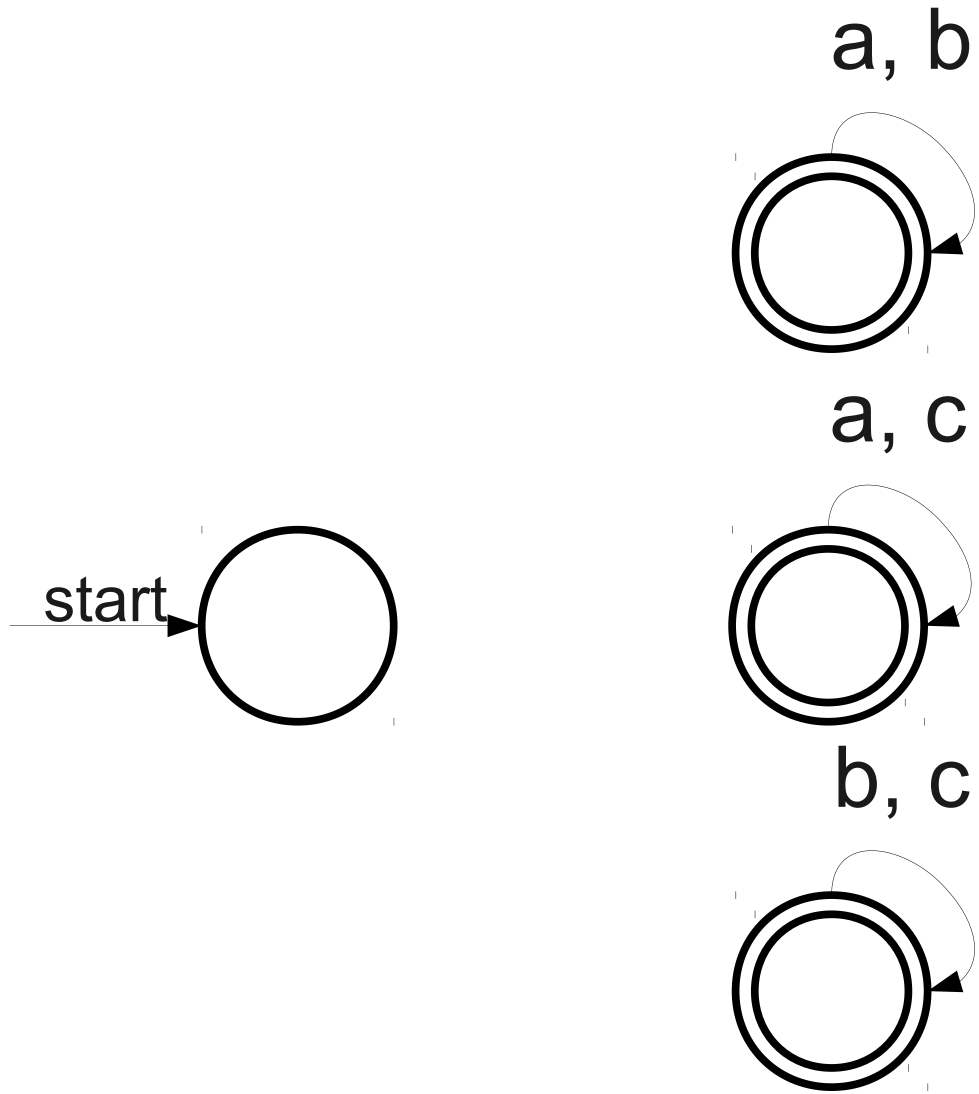
Guess-and-Check

$L = \{ w \in \{\texttt{a}, \texttt{b}, \texttt{c}\}^* \mid \text{at least one of } \texttt{a}, \texttt{b}, \text{ or } \texttt{c} \text{ is not in } w \}$



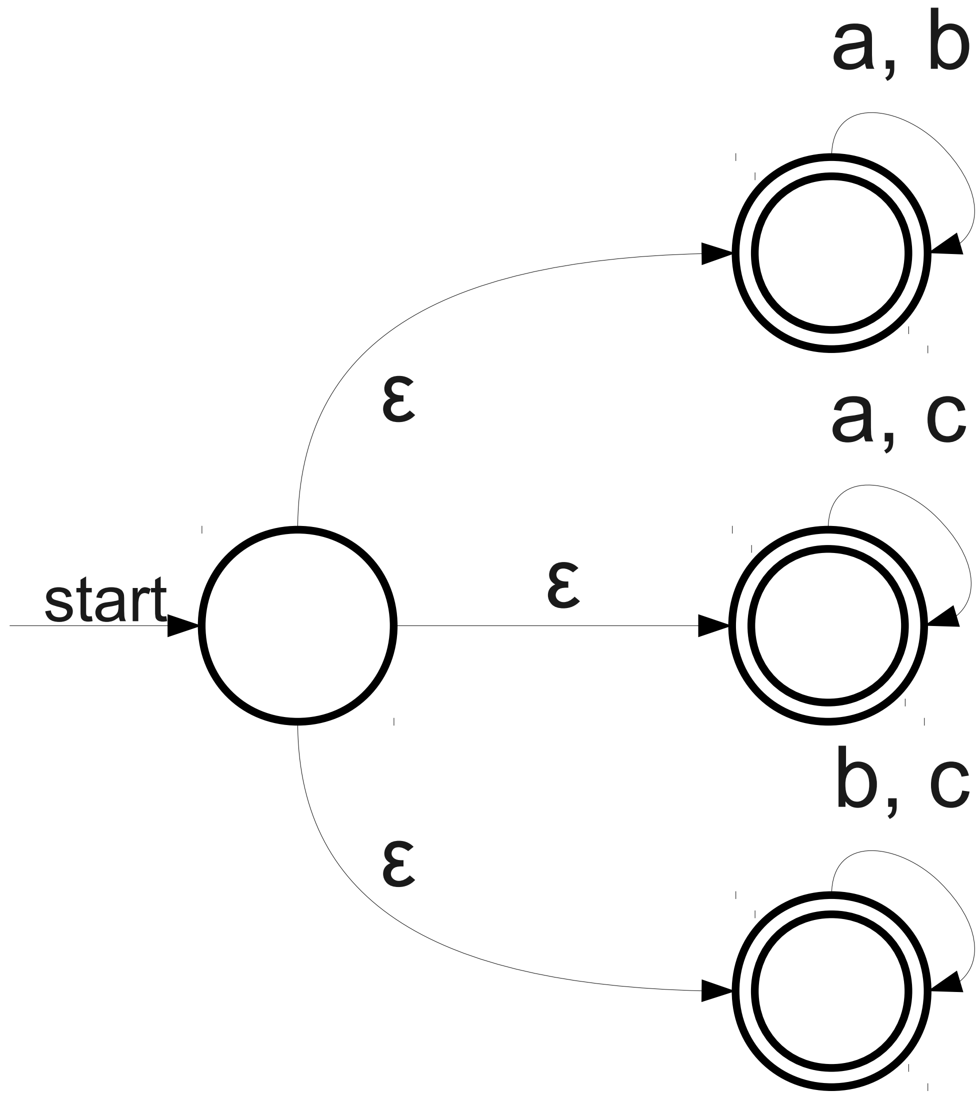
Guess-and-Check

$L = \{ w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid \text{at least one of } \mathtt{a}, \mathtt{b}, \text{ or } \mathtt{c} \text{ is not in } w \}$



Guess-and-Check

$L = \{ w \in \{\texttt{a}, \texttt{b}, \texttt{c}\}^* \mid \text{at least one of } \texttt{a}, \texttt{b}, \text{ or } \texttt{c} \text{ is not in } w \}$



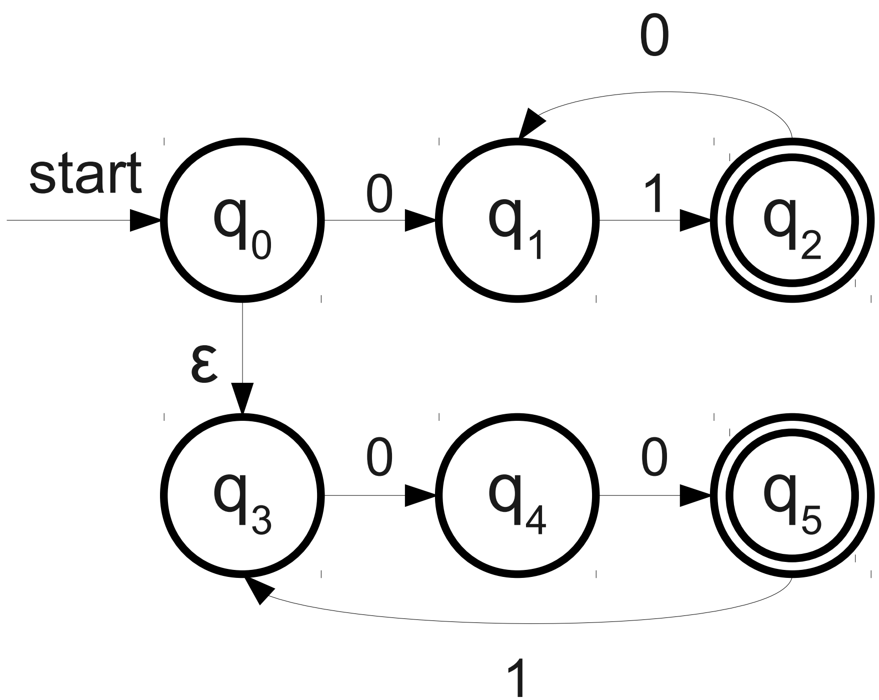
NFAs and DFAs

- Any language that can be accepted by a DFA can be accepted by an NFA.
- Why?
  - Just use the same set of transitions as before.
- **Question**: Can any language accepted by an NFA also be accepted by a DFA?
- Surprisingly, the answer is **yes!**

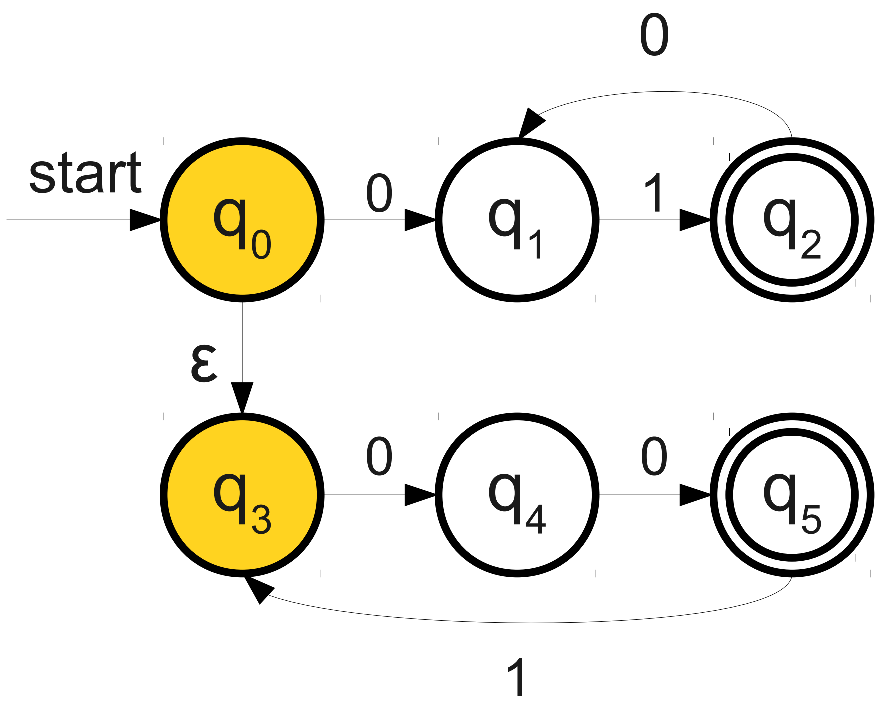
Simulation

- **Simulation** is a key technique in computability theory.
- If we can build an automaton A' whose behavior ***simulates*** that of another automaton A, then we can make a connection between A and A'.
- To show that any language accepted by an NFA can be accepted by a DFA, we will show how to make a DFA that ***simulates*** the execution of an NFA.

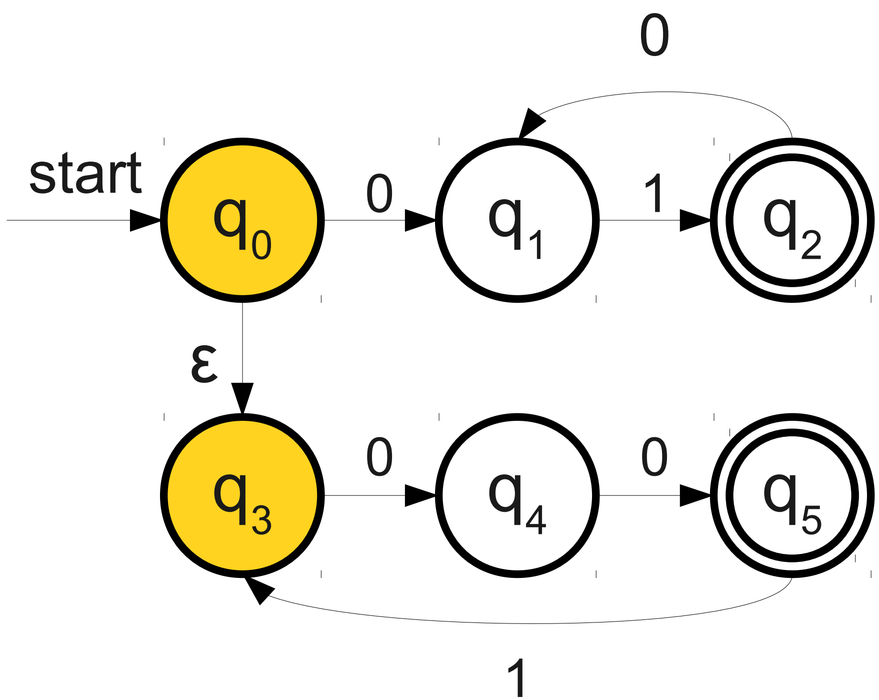
Simulating an NFA with a DFA

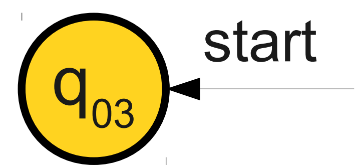


Simulating an NFA with a DFA

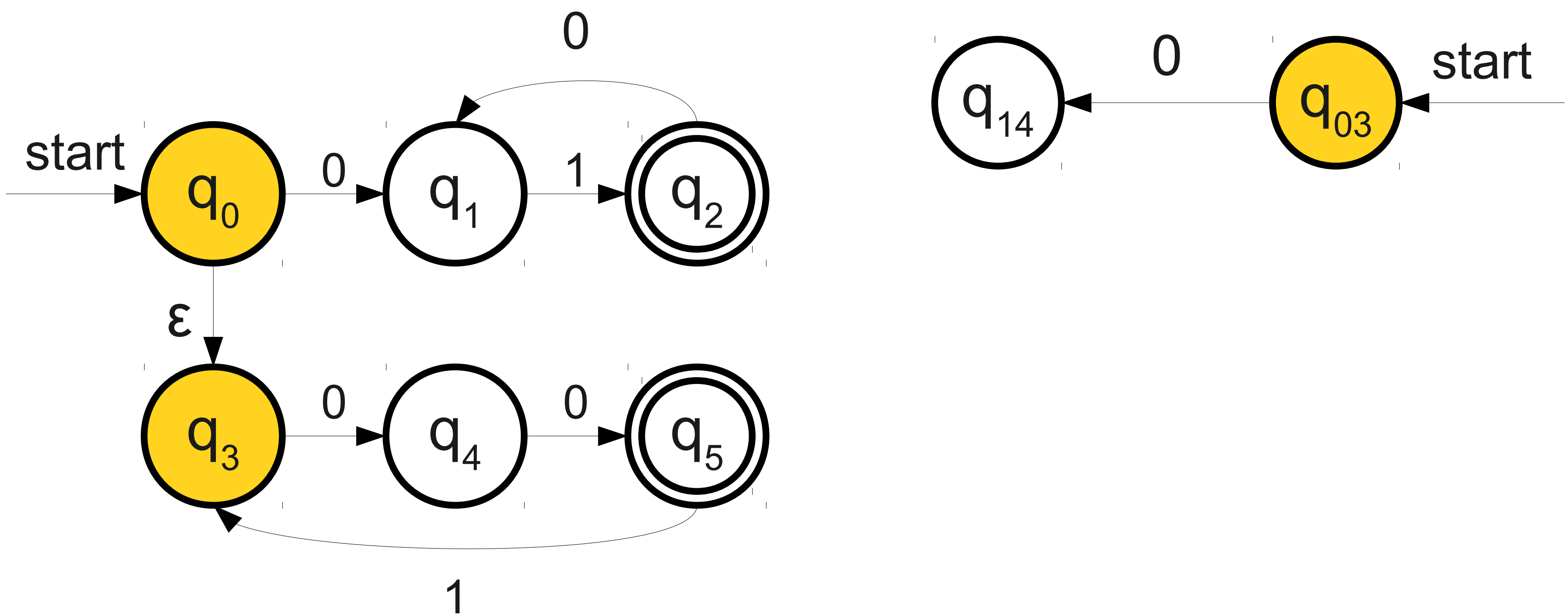


Simulating an NFA with a DFA

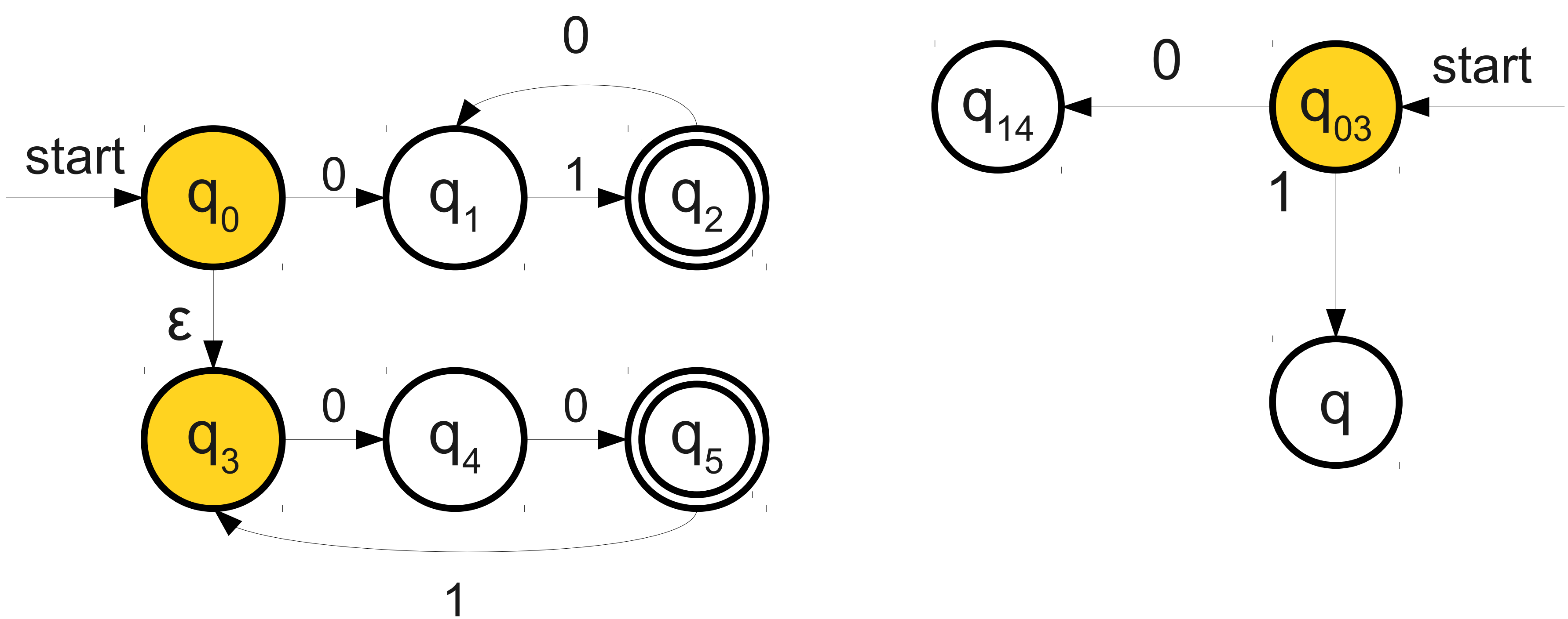




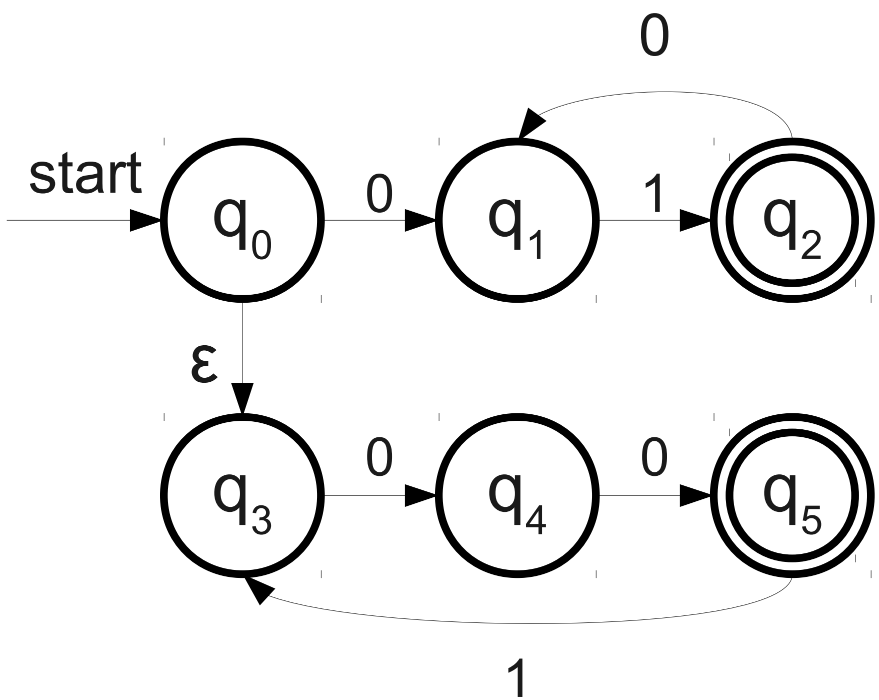
Simulating an NFA with a DFA

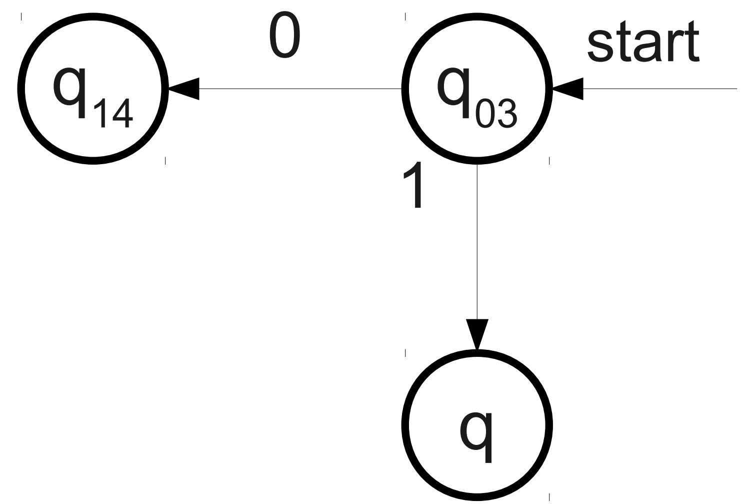


Simulating an NFA with a DFA

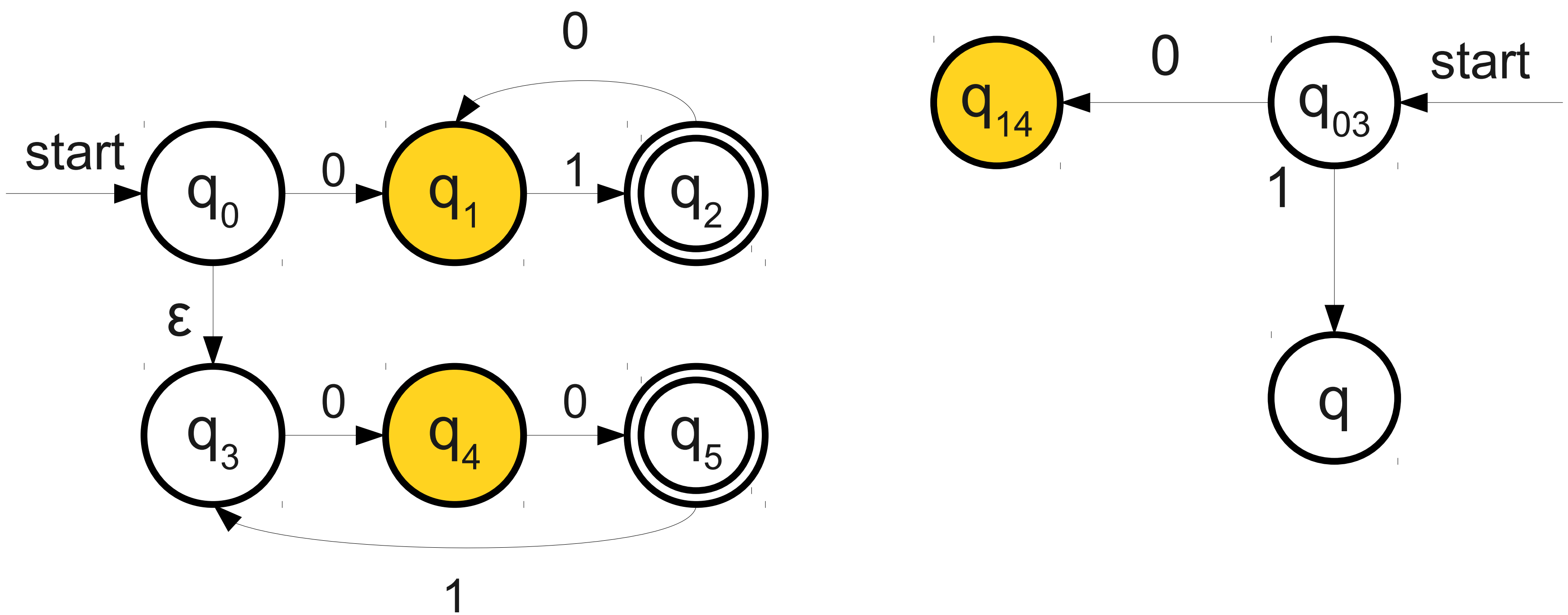


Simulating an NFA with a DFA

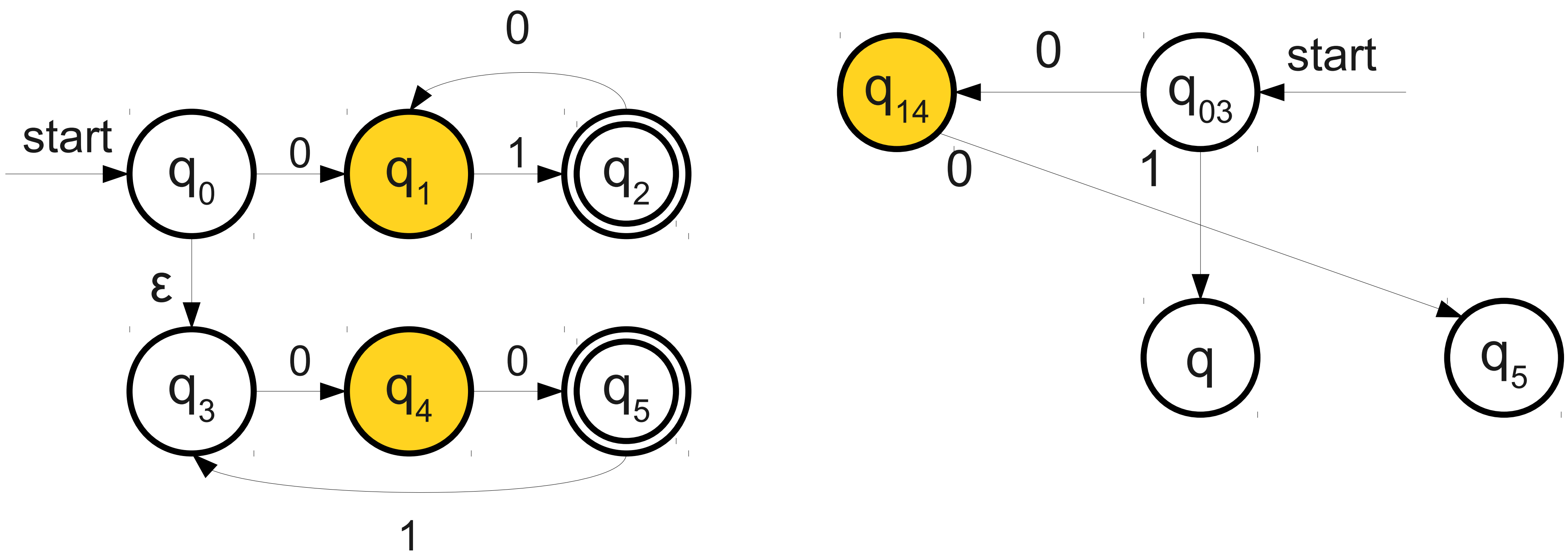




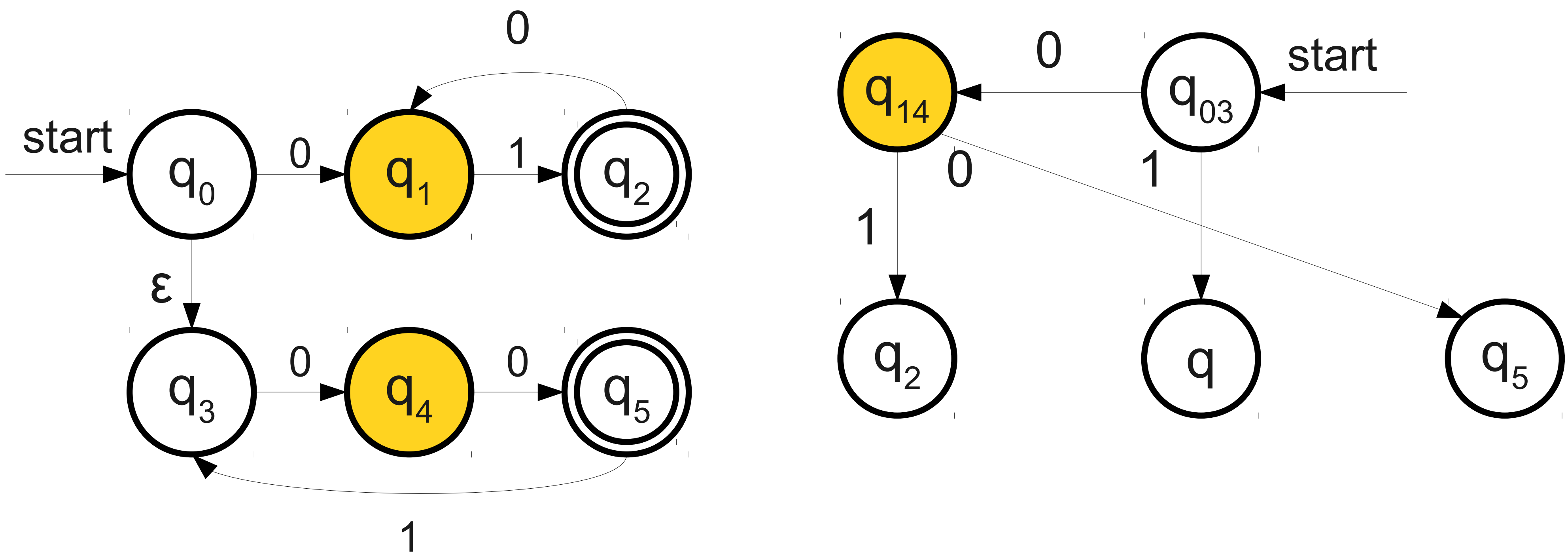
Simulating an NFA with a DFA



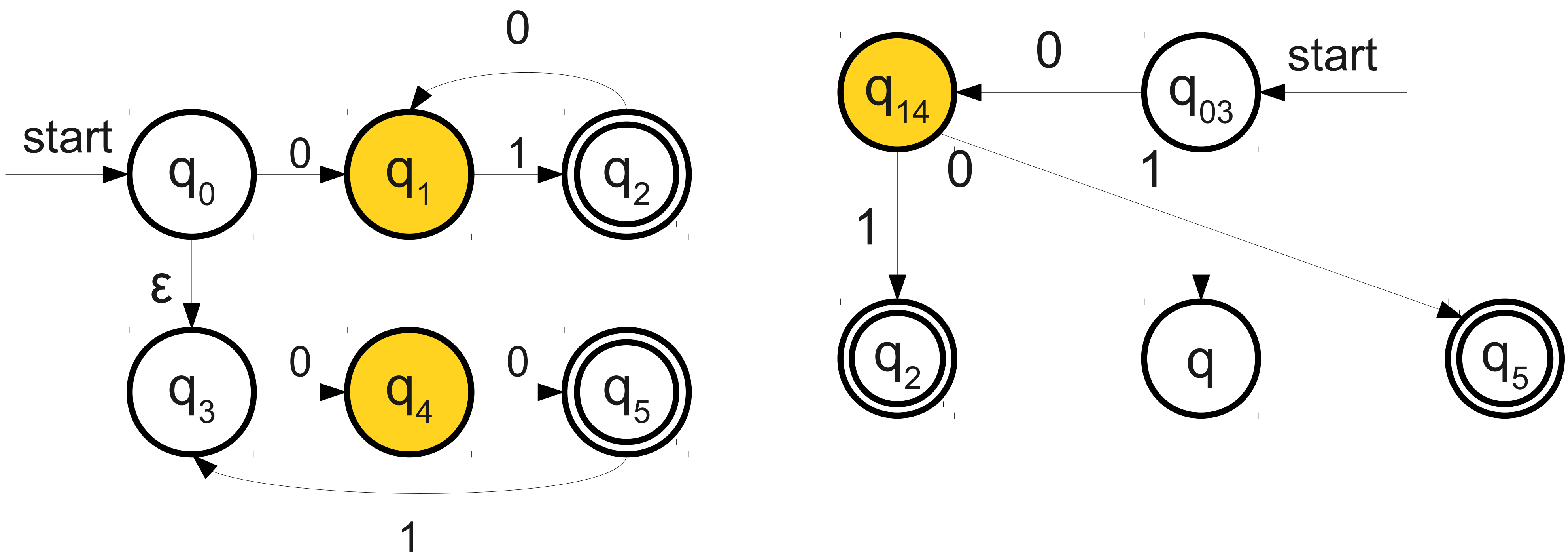
Simulating an NFA with a DFA



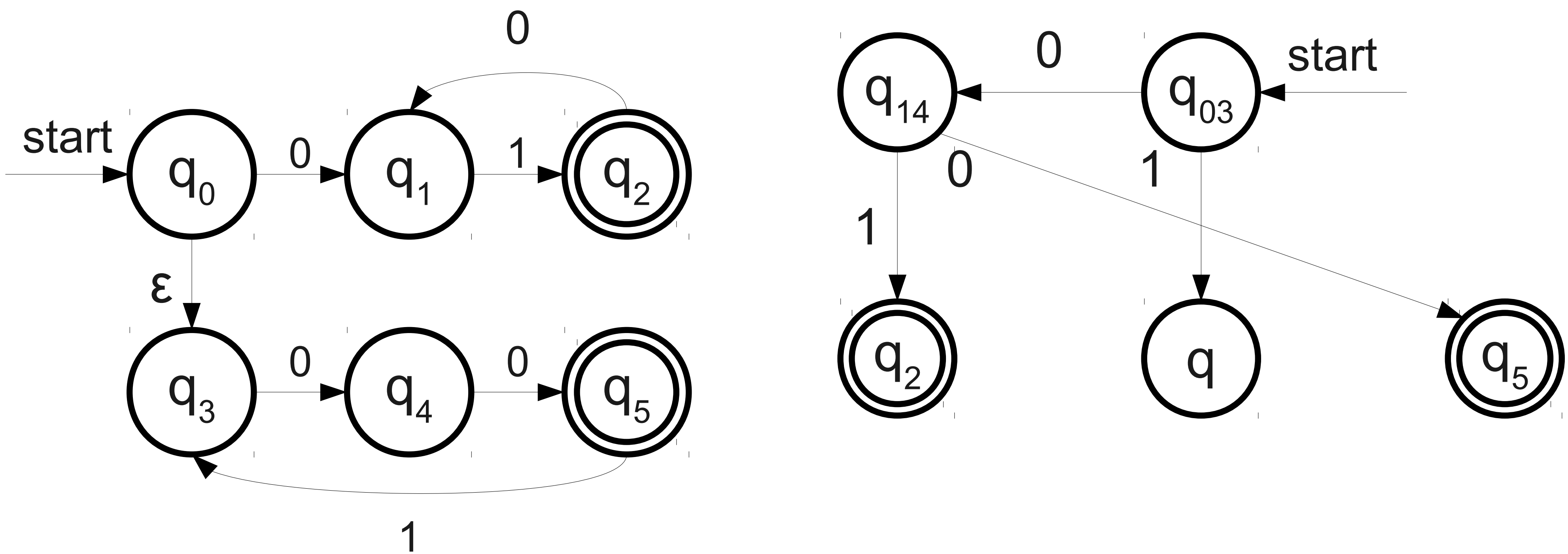
Simulating an NFA with a DFA



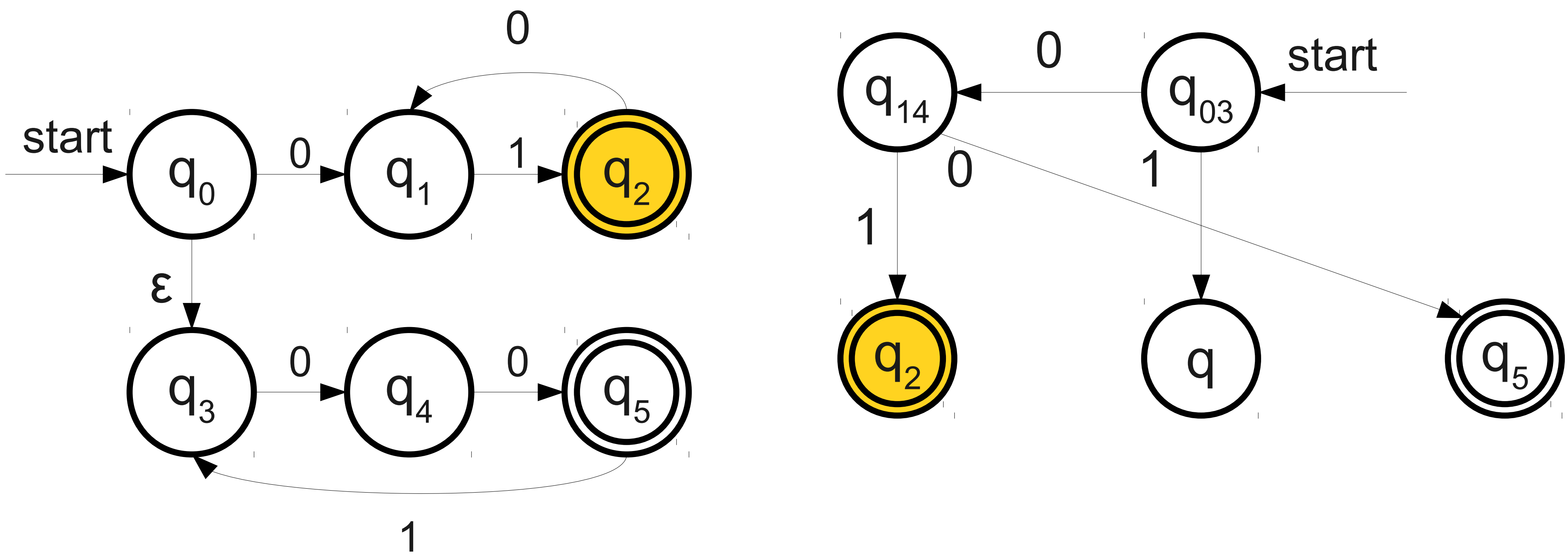
Simulating an NFA with a DFA



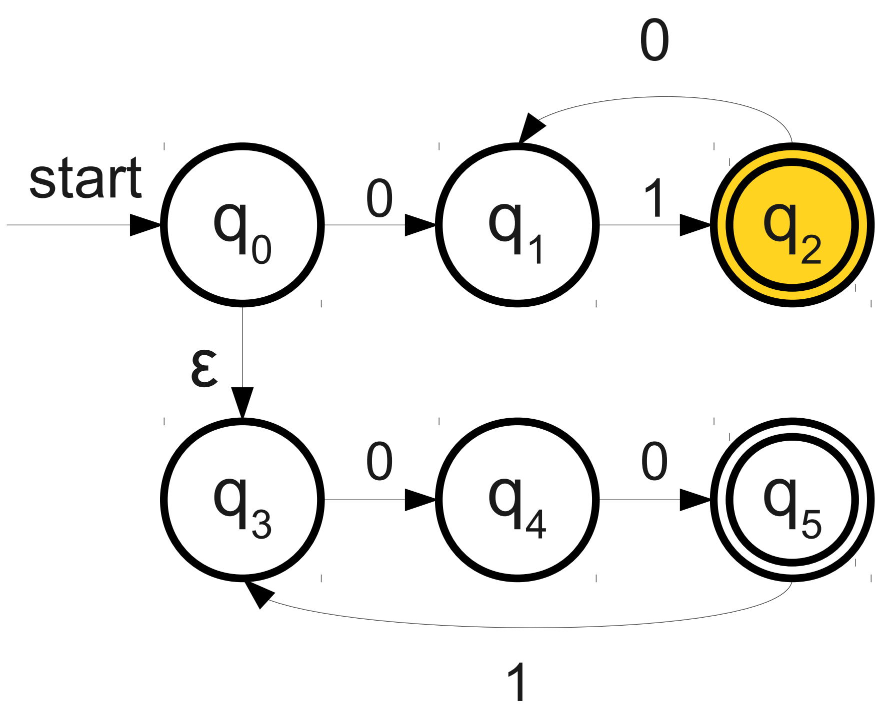
Simulating an NFA with a DFA

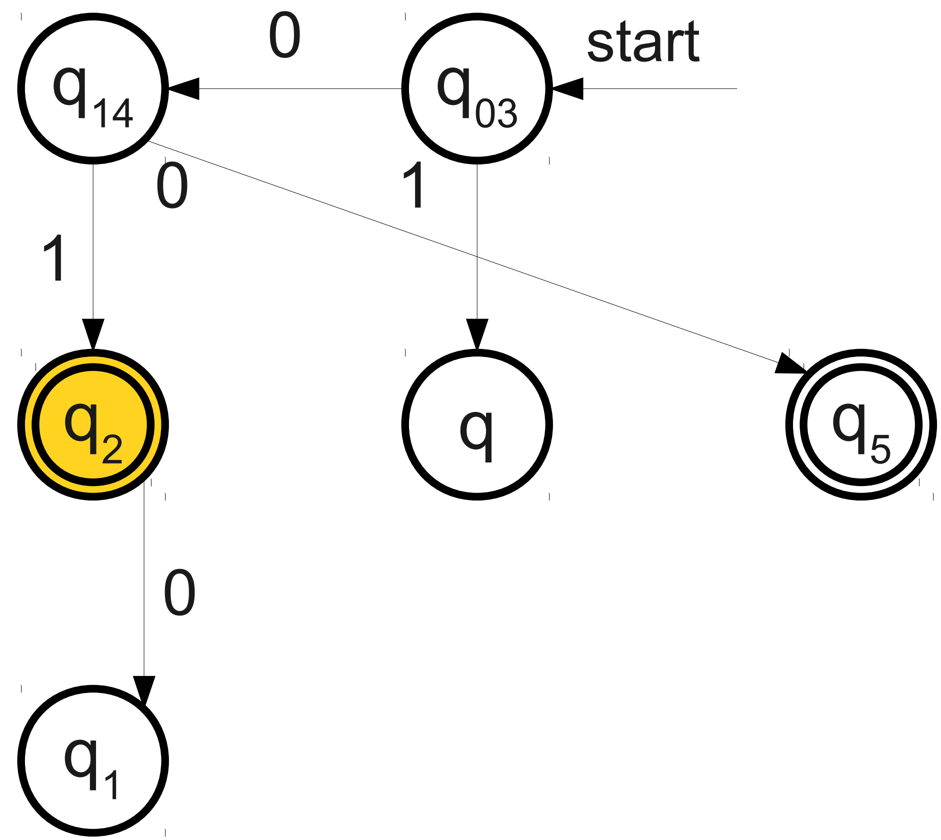


Simulating an NFA with a DFA

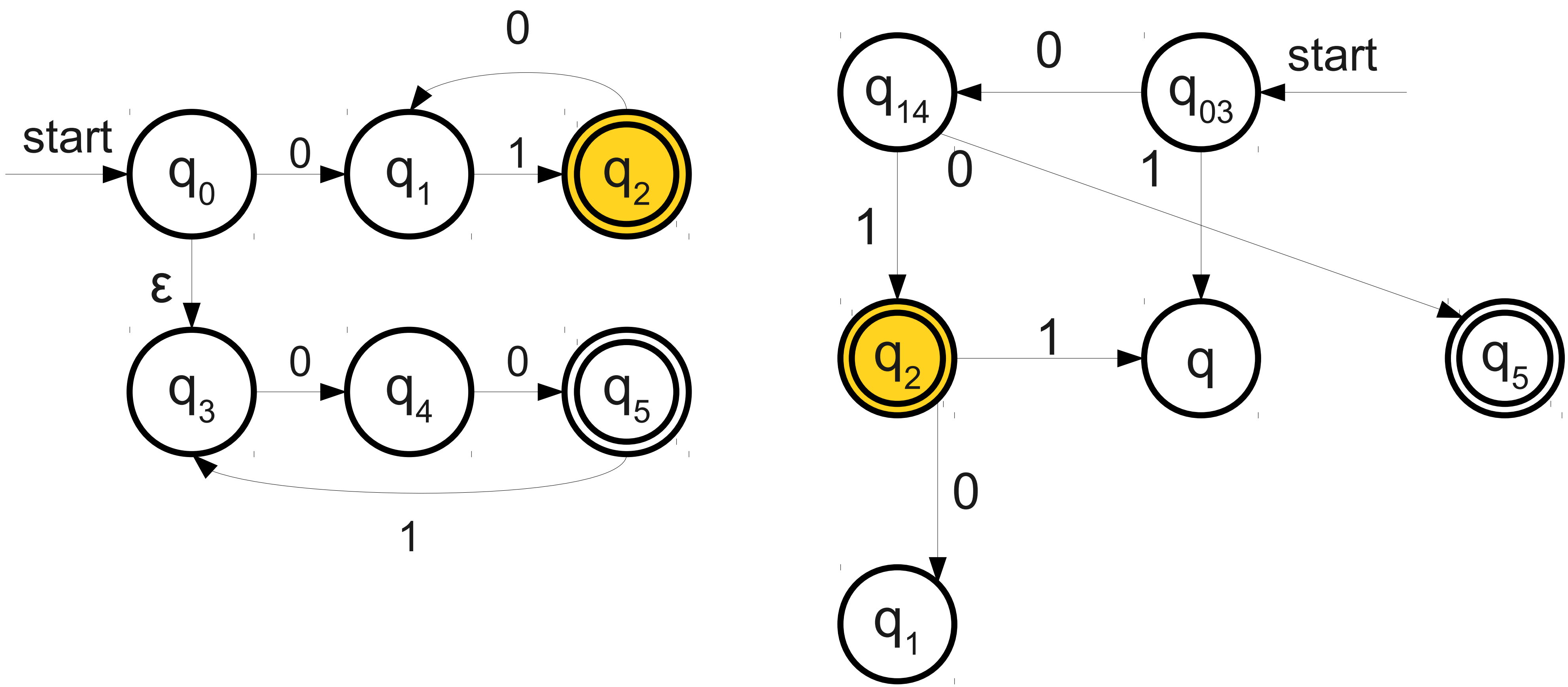


Simulating an NFA with a DFA

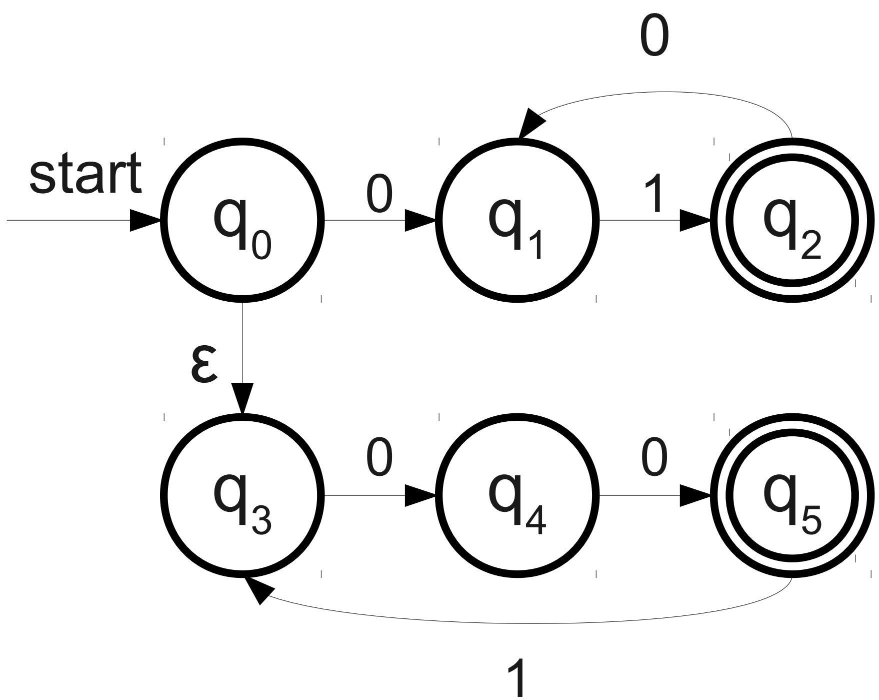


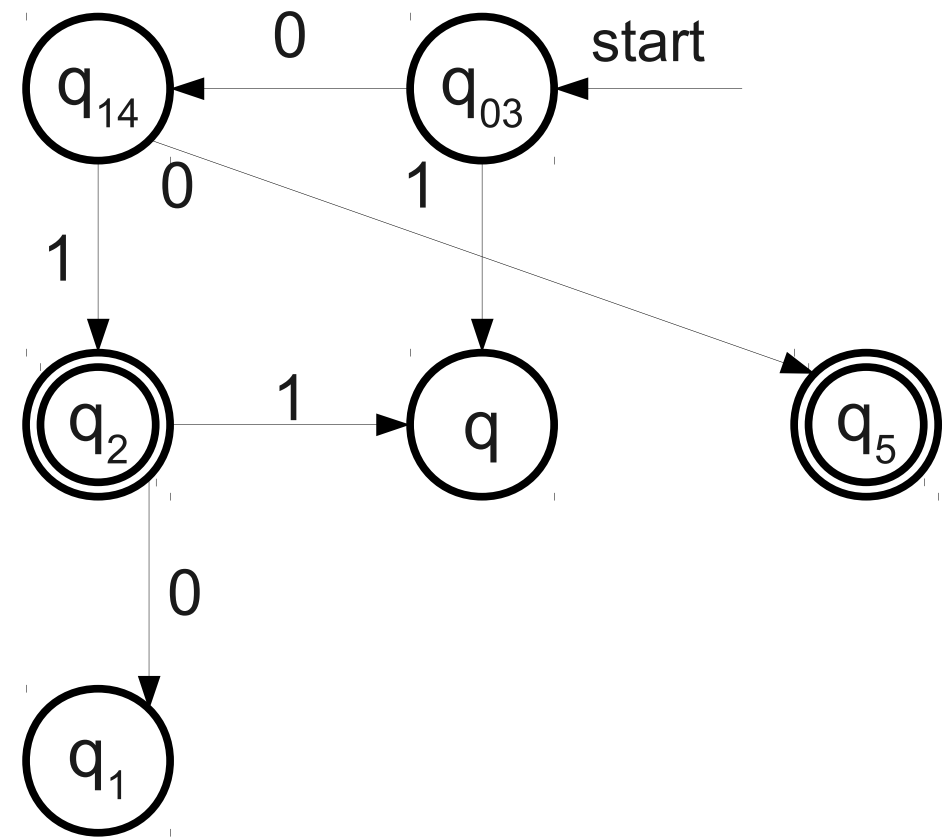


Simulating an NFA with a DFA

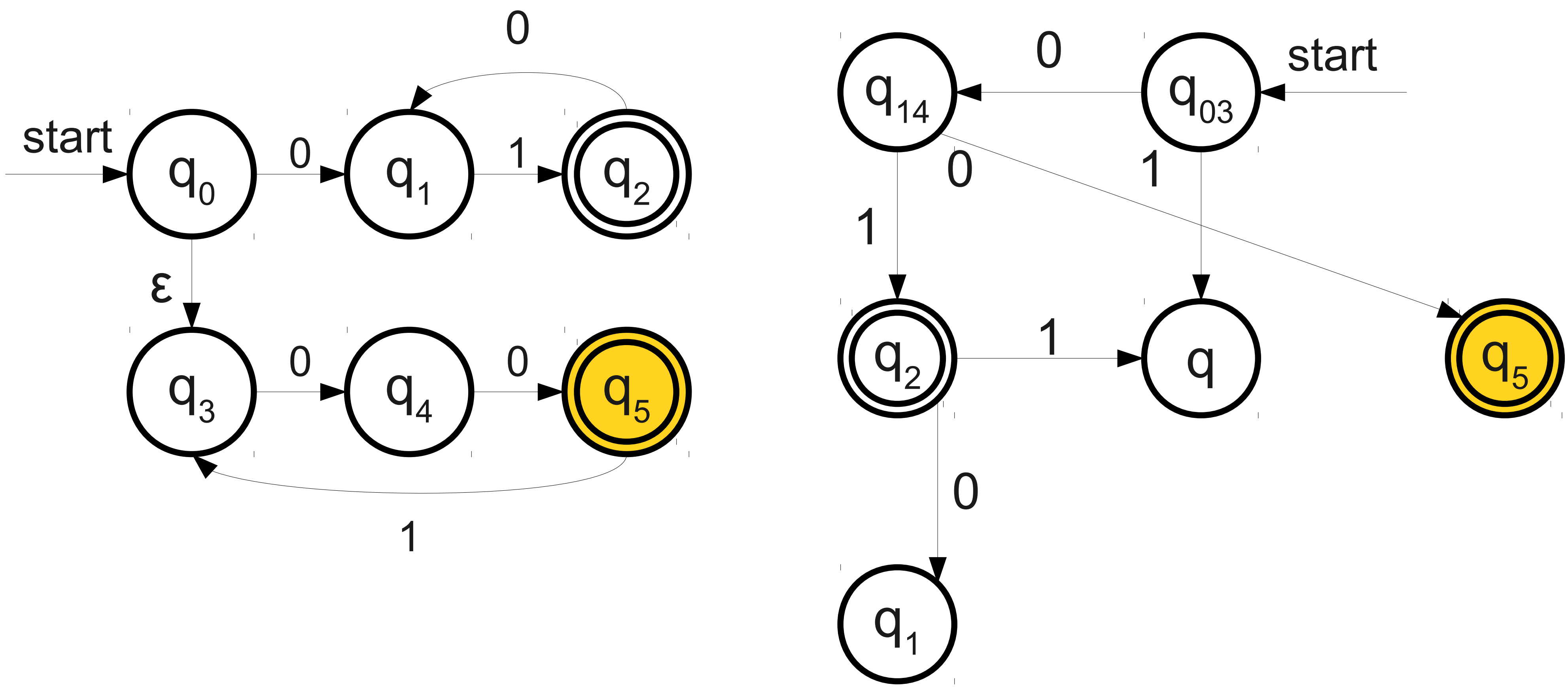


Simulating an NFA with a DFA

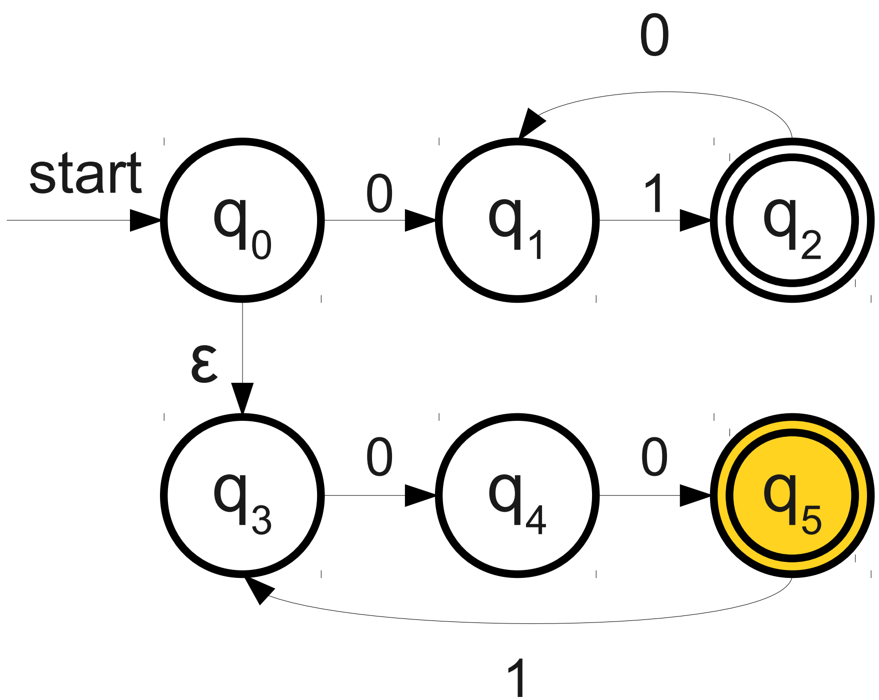


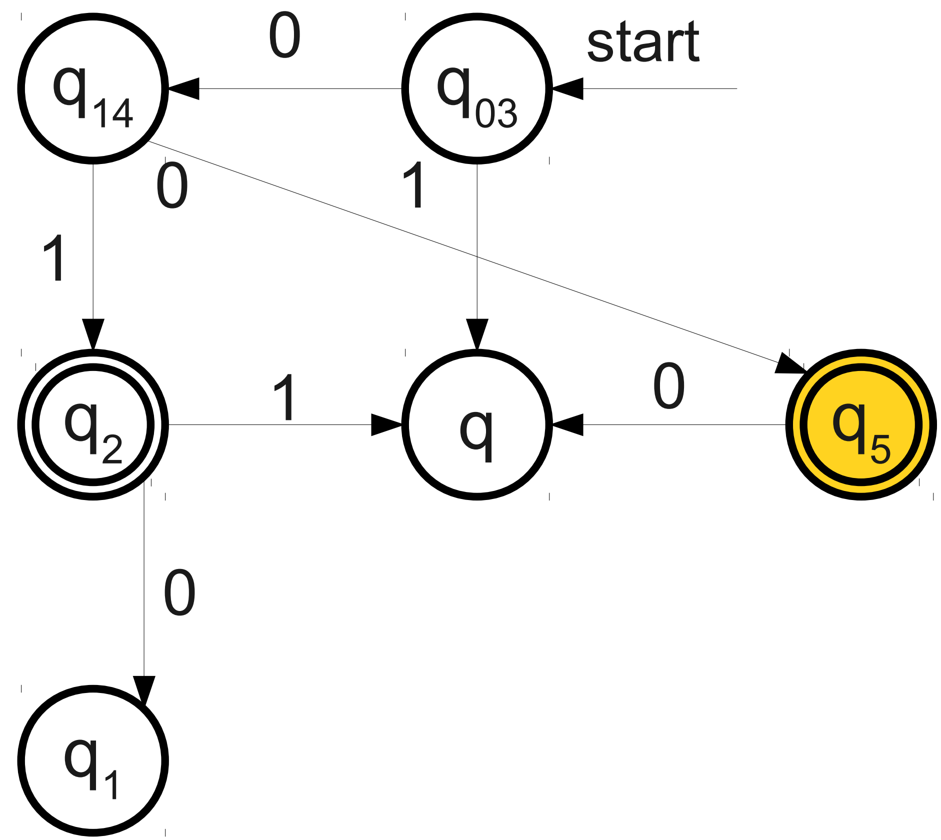


Simulating an NFA with a DFA

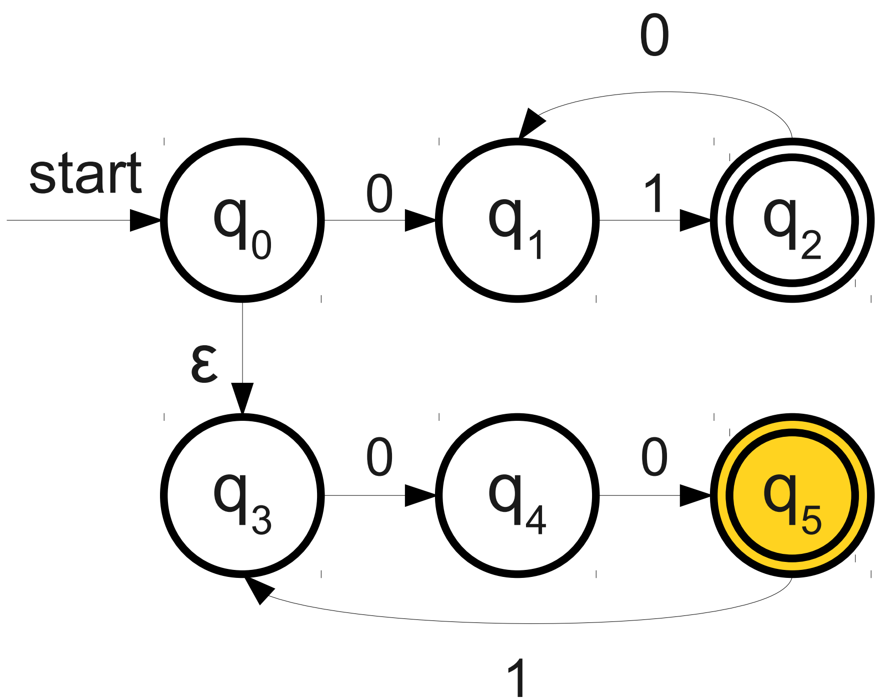


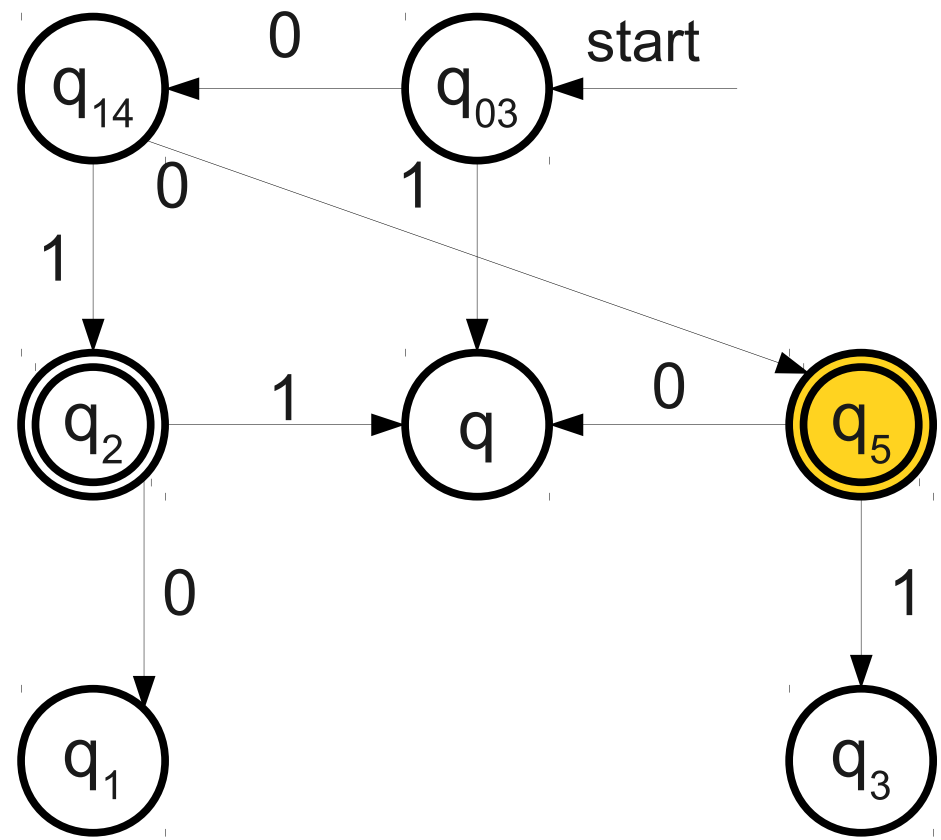
Simulating an NFA with a DFA



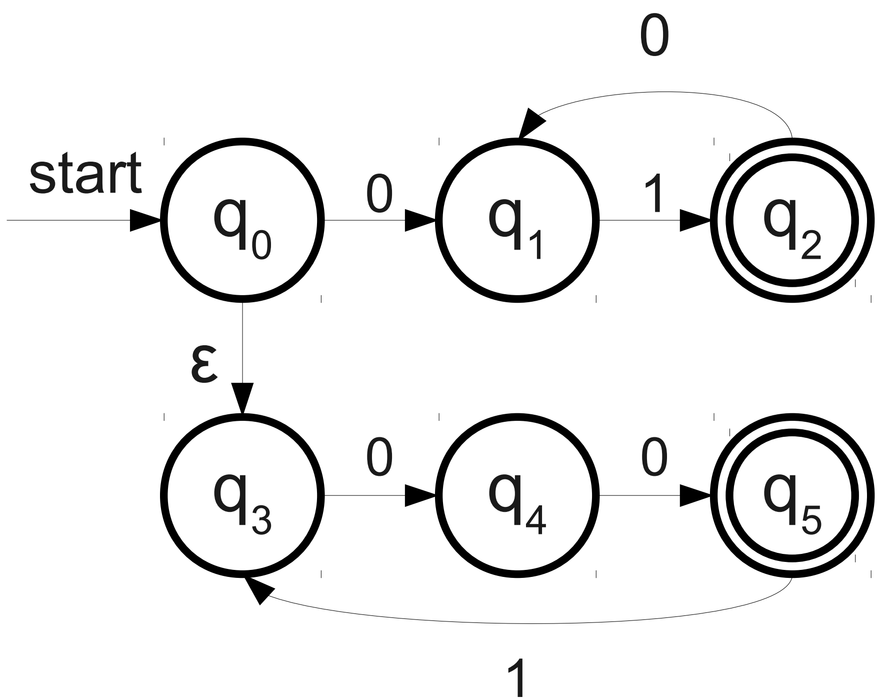


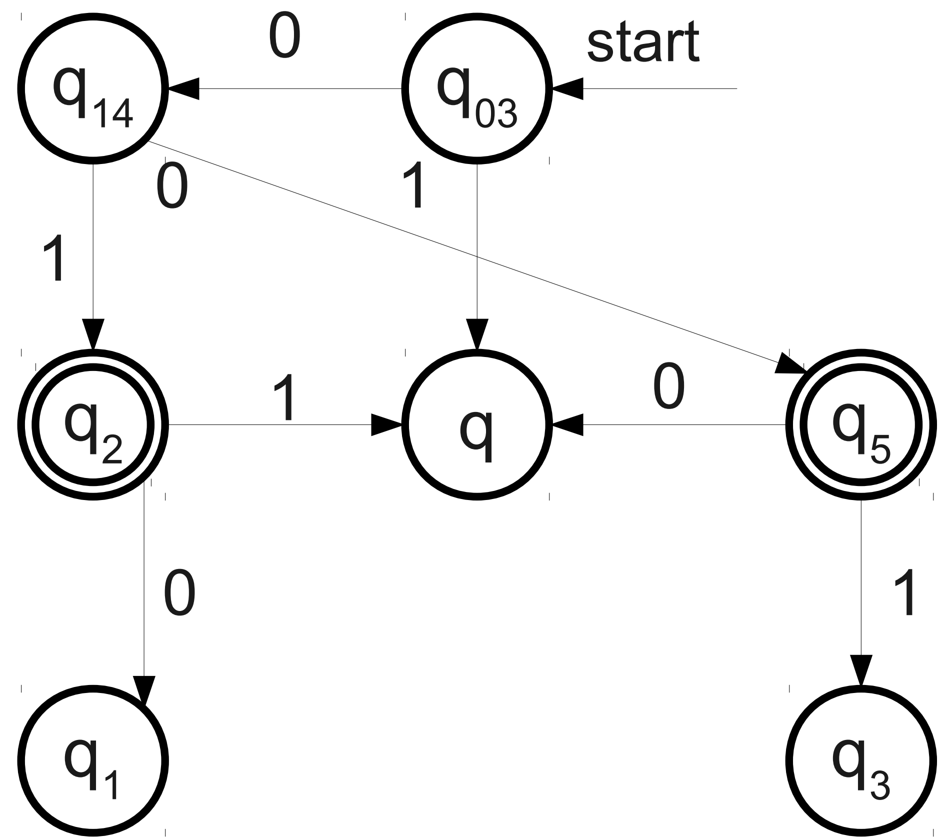
Simulating an NFA with a DFA



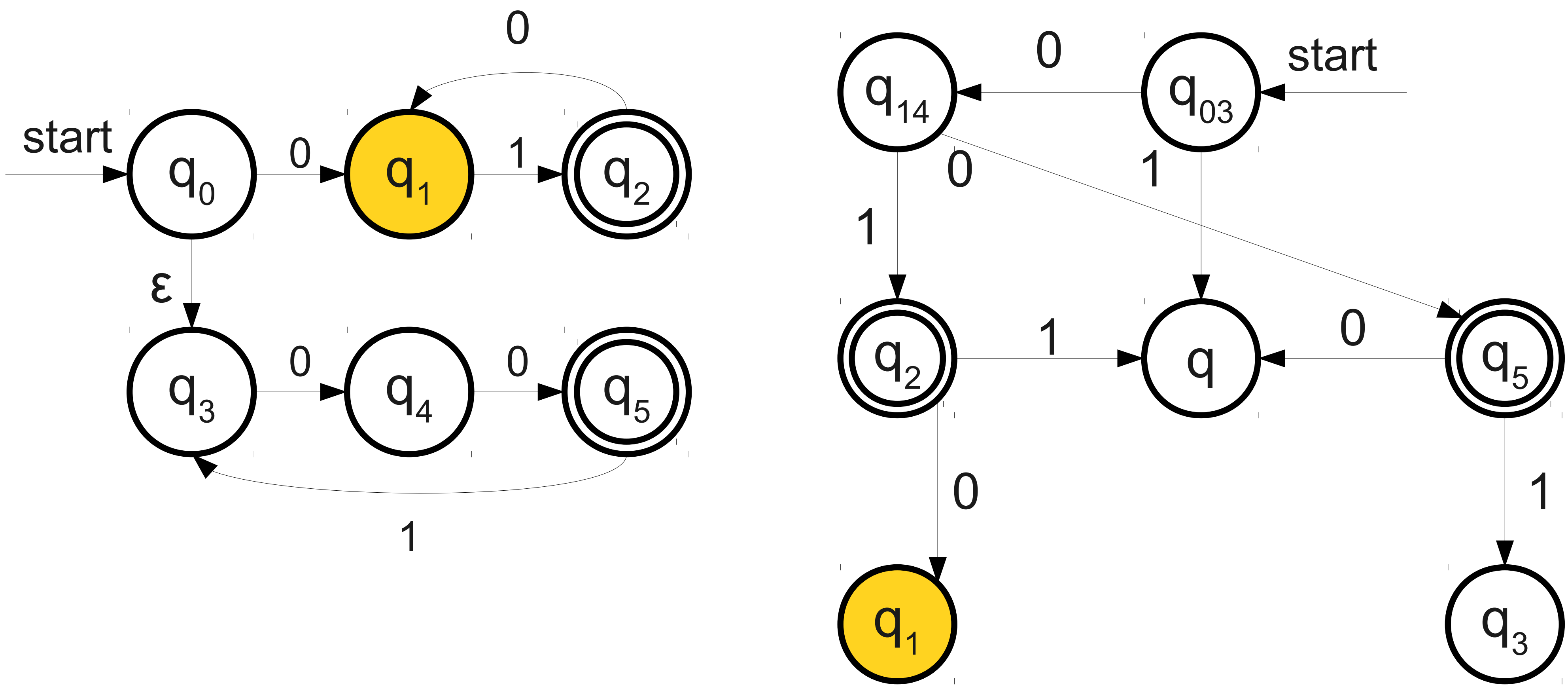


Simulating an NFA with a DFA

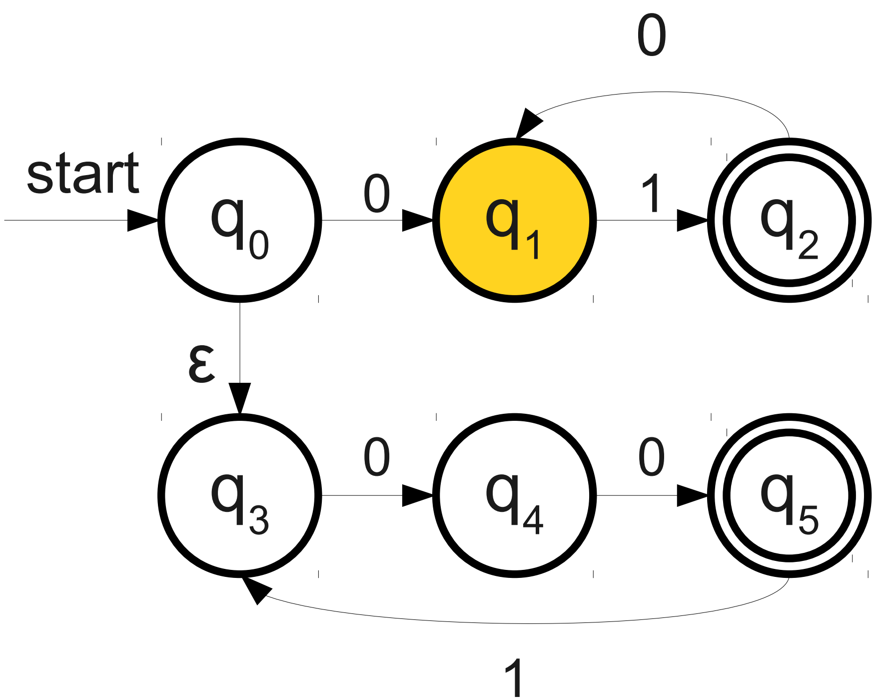


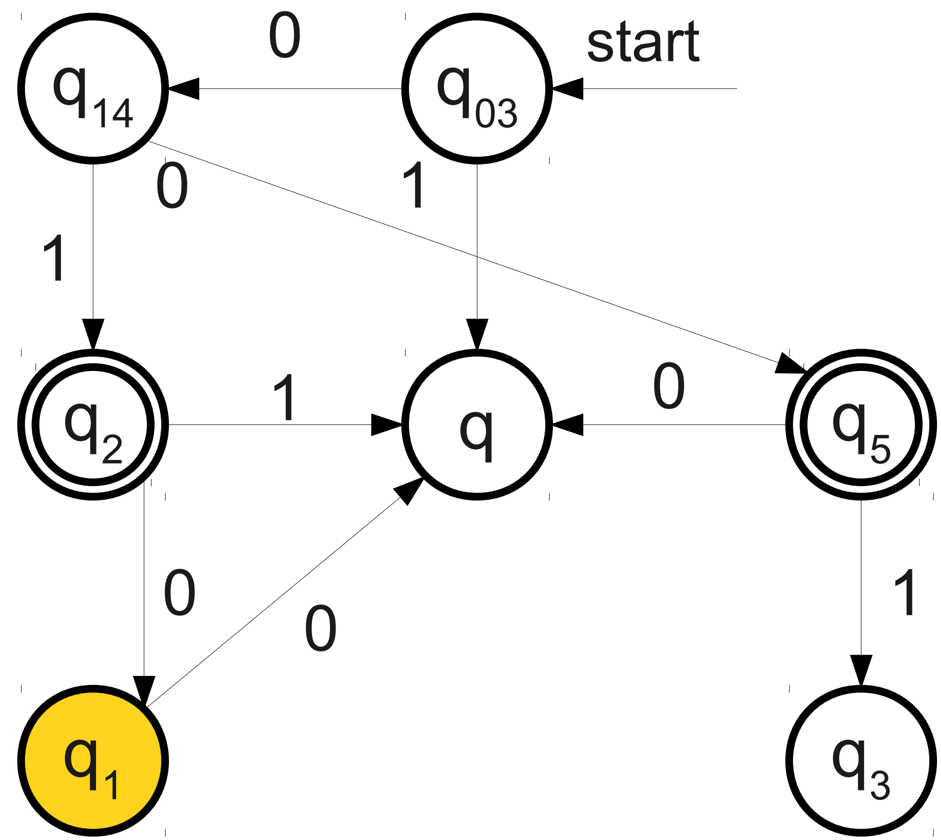


Simulating an NFA with a DFA

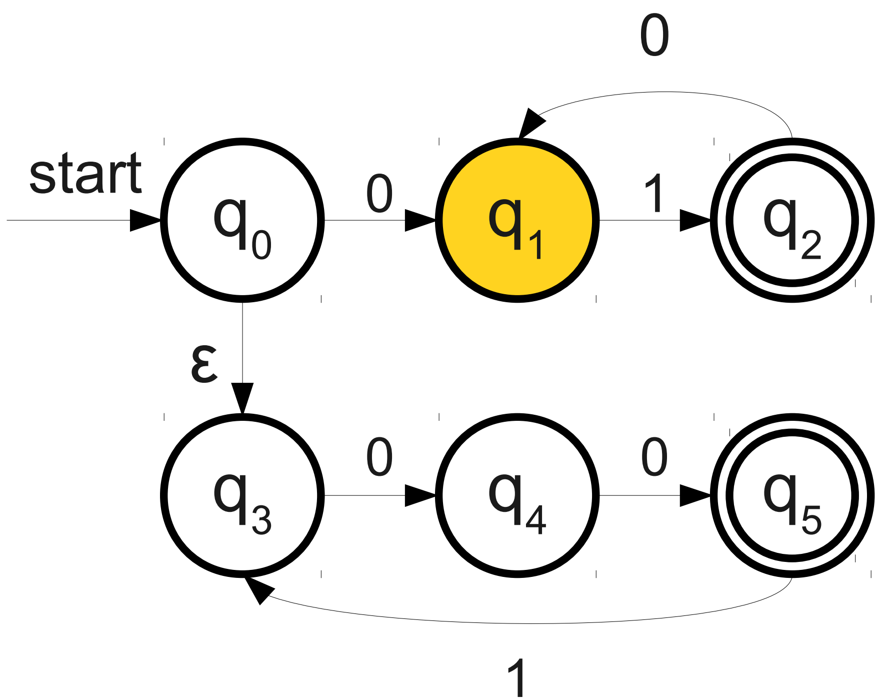


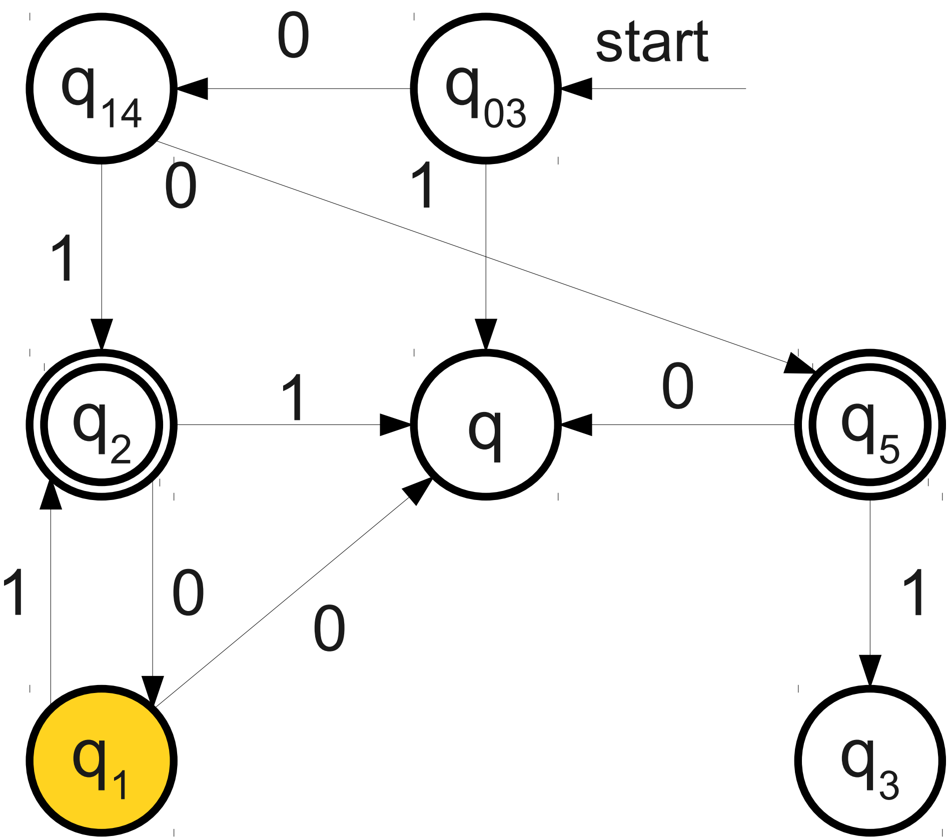
Simulating an NFA with a DFA



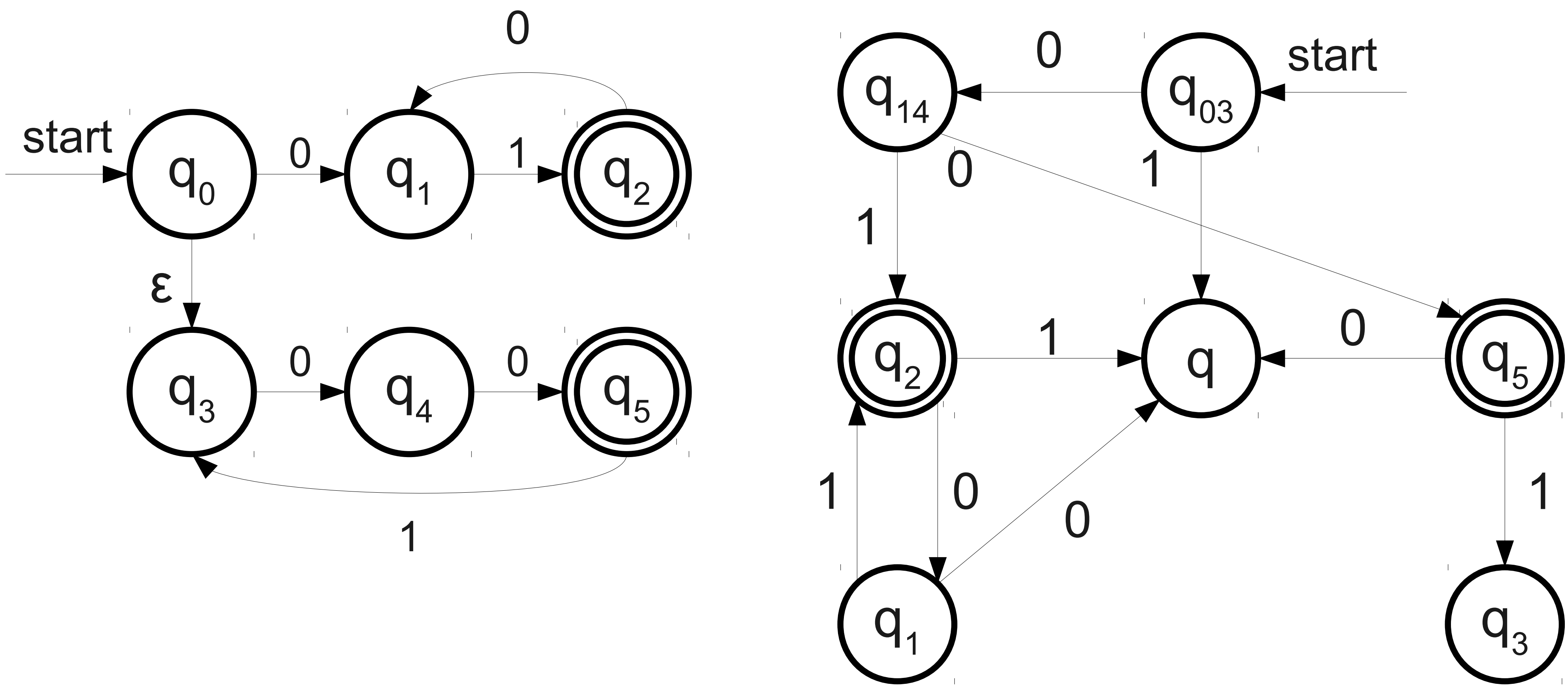


Simulating an NFA with a DFA

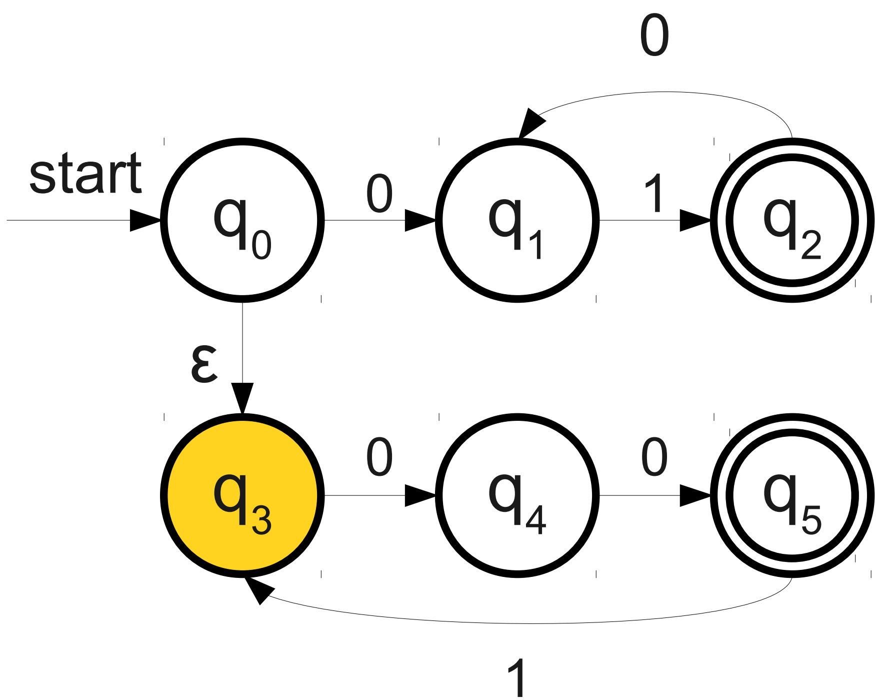


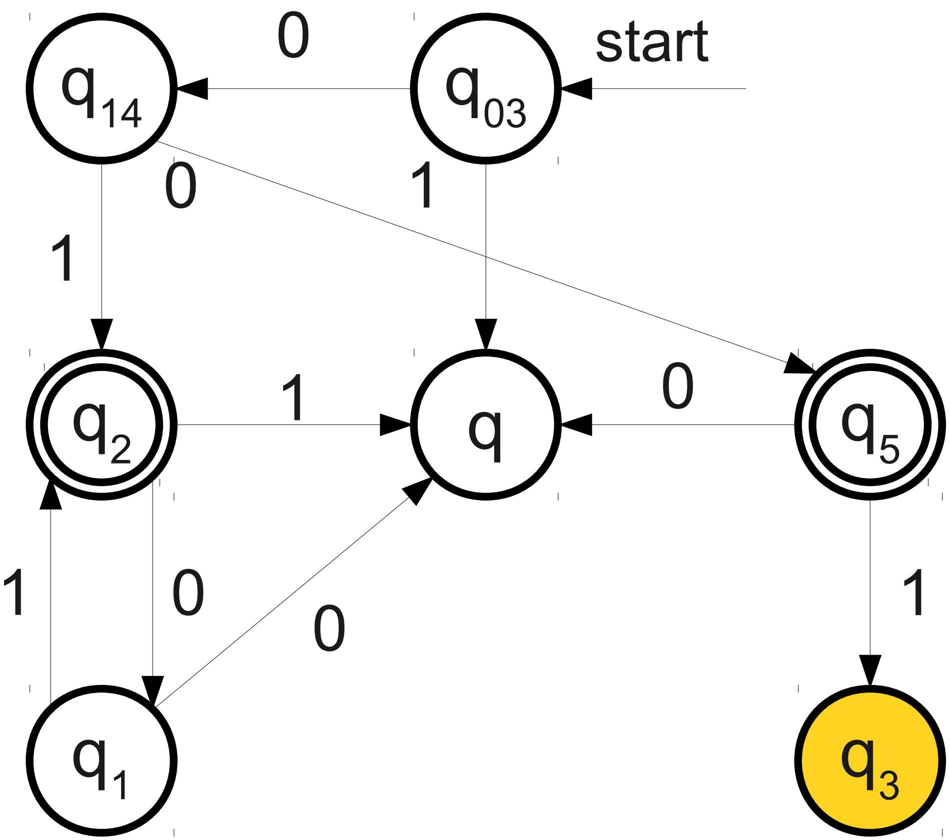


Simulating an NFA with a DFA

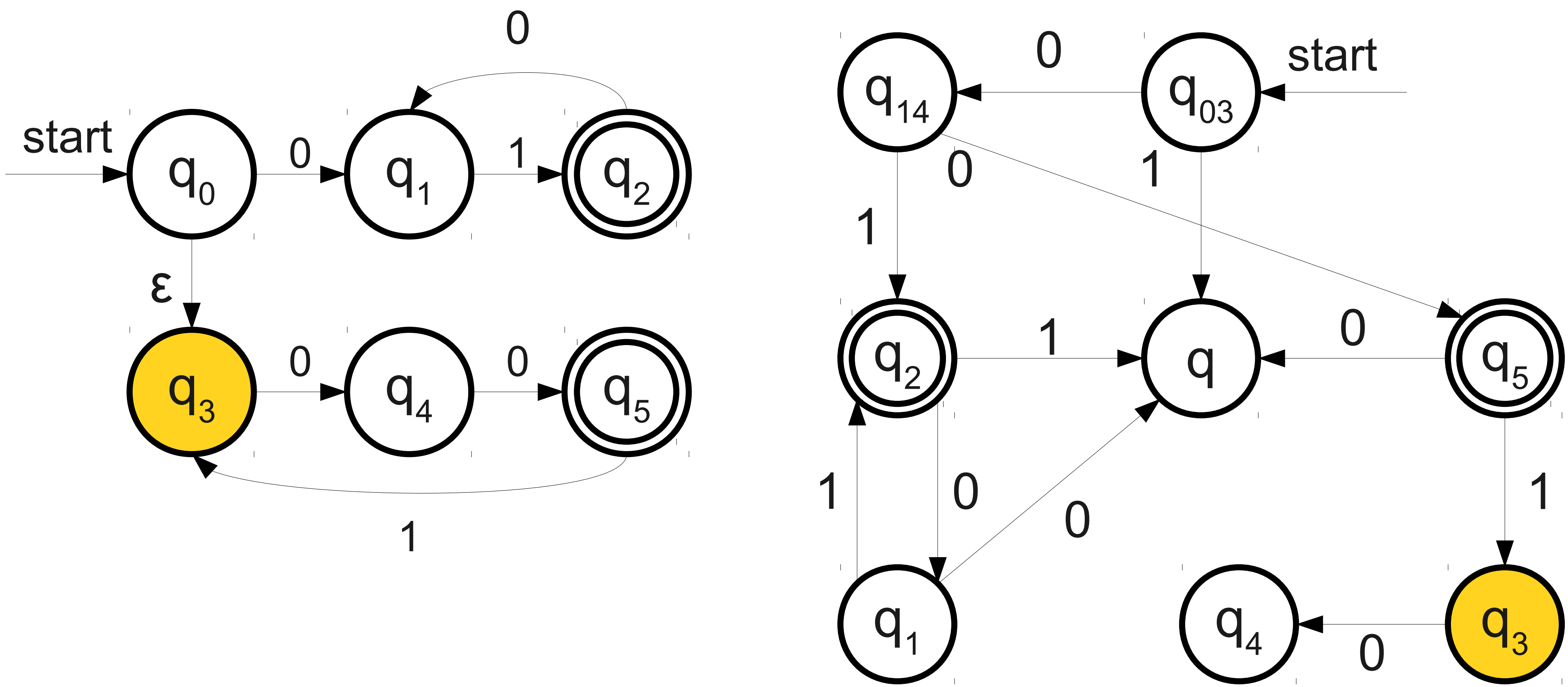


Simulating an NFA with a DFA

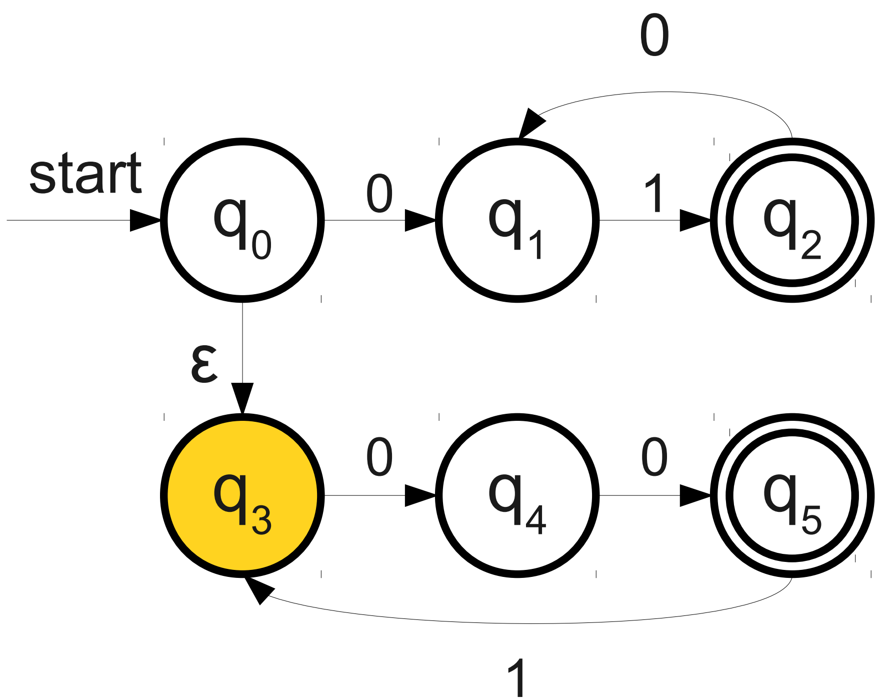


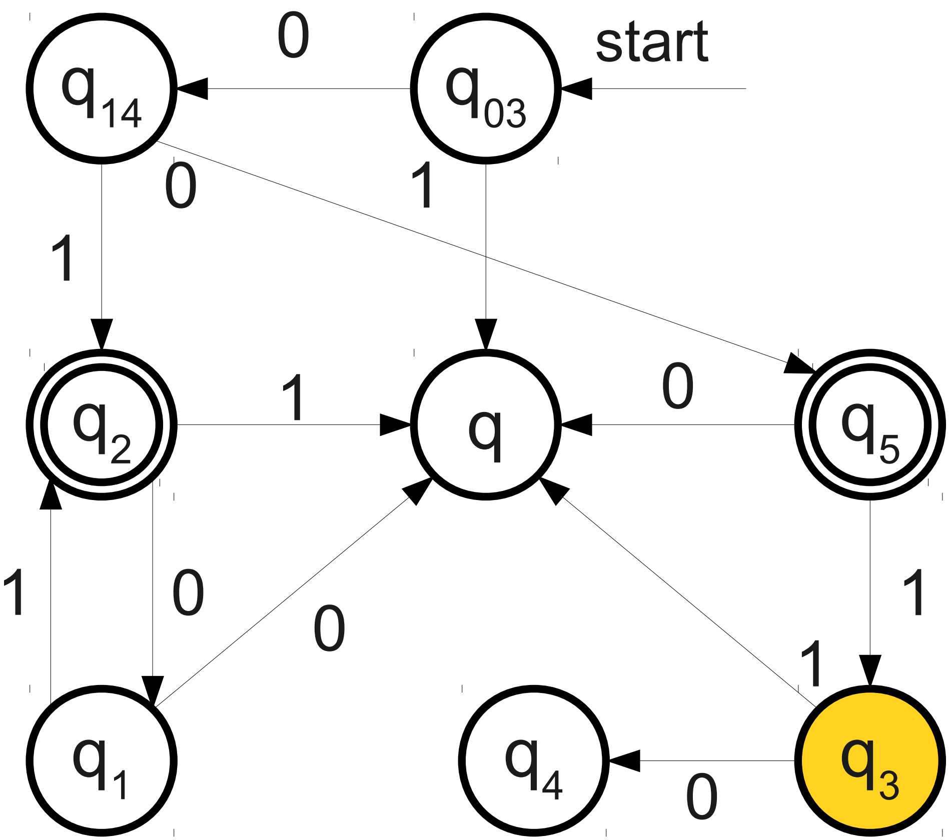


Simulating an NFA with a DFA

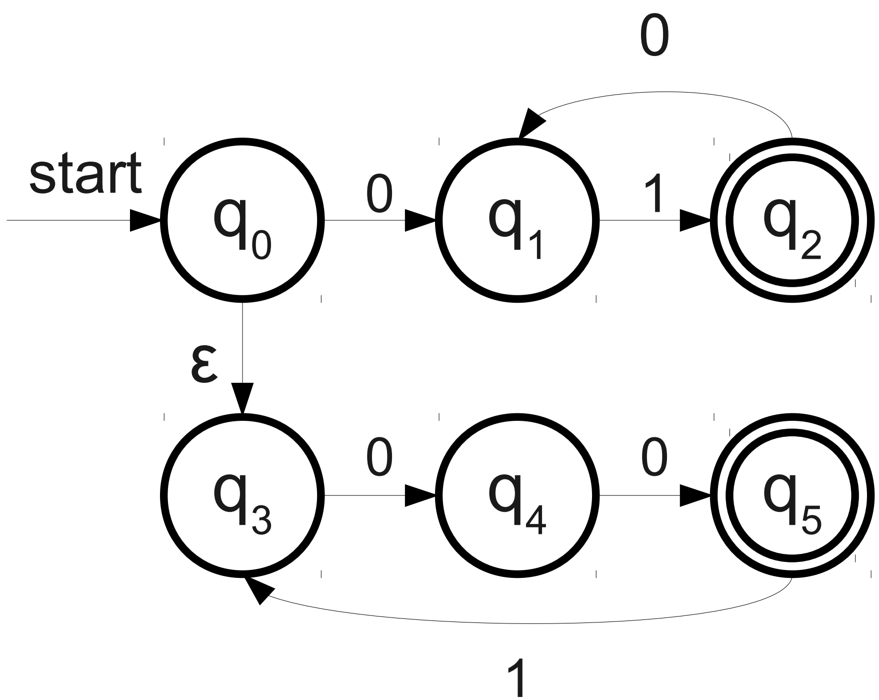


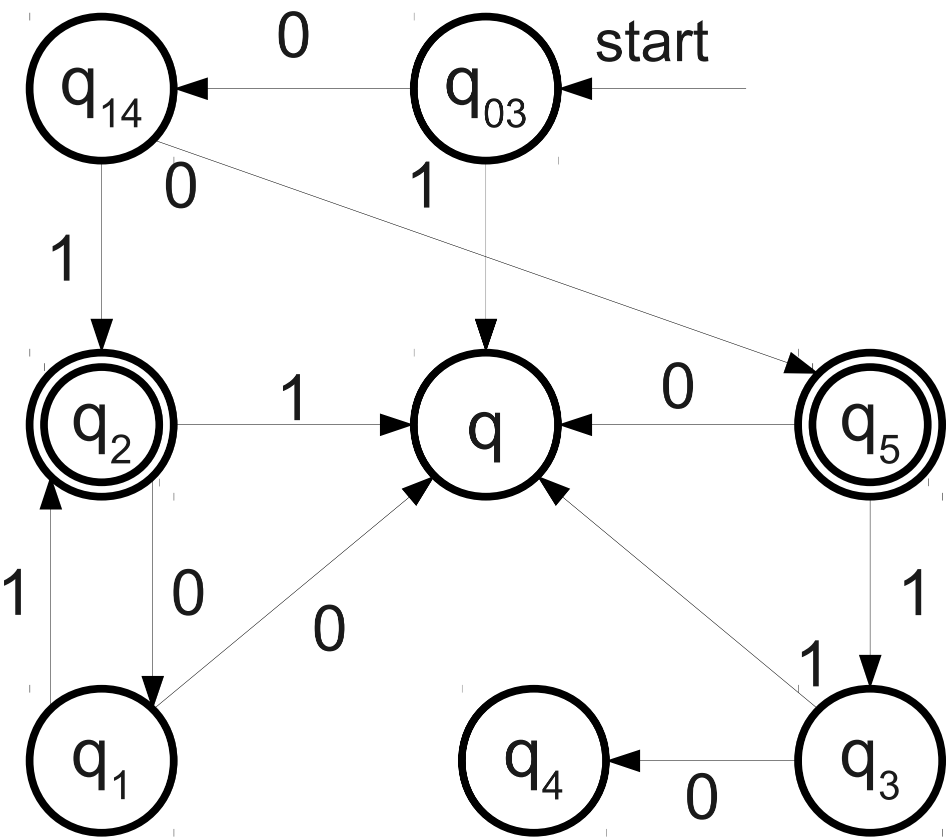
Simulating an NFA with a DFA



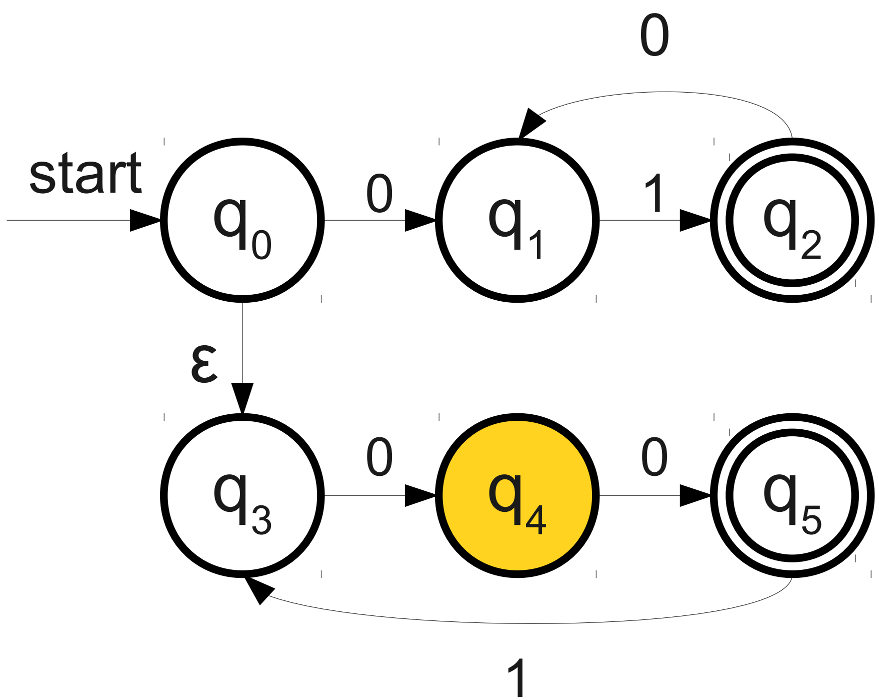


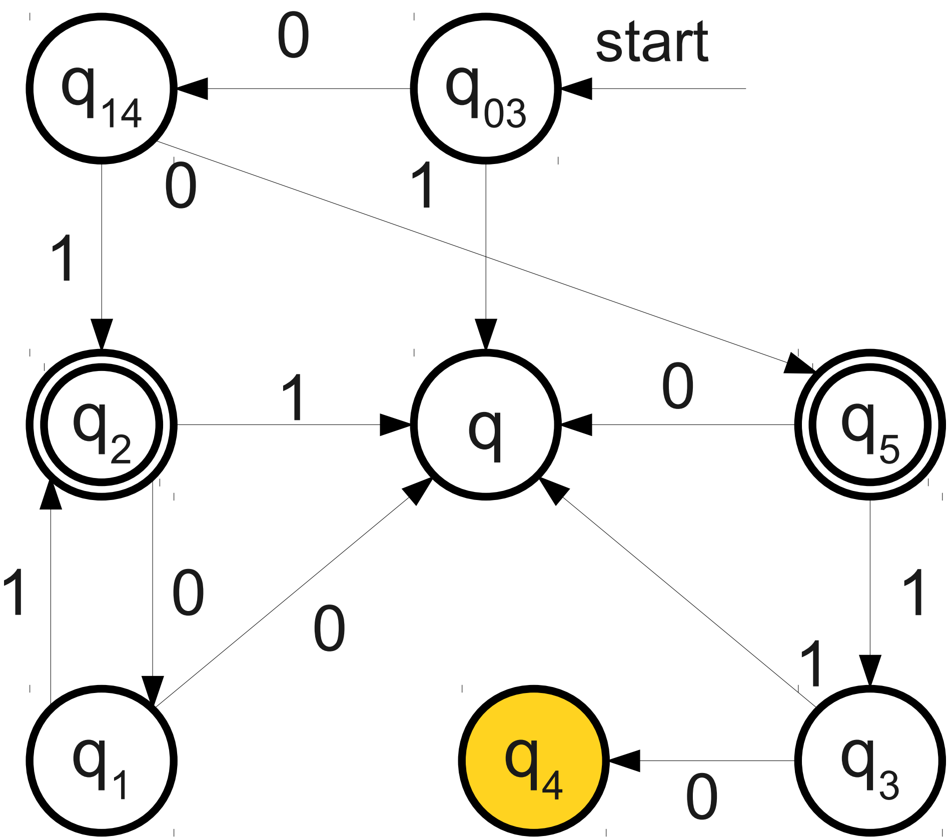
Simulating an NFA with a DFA



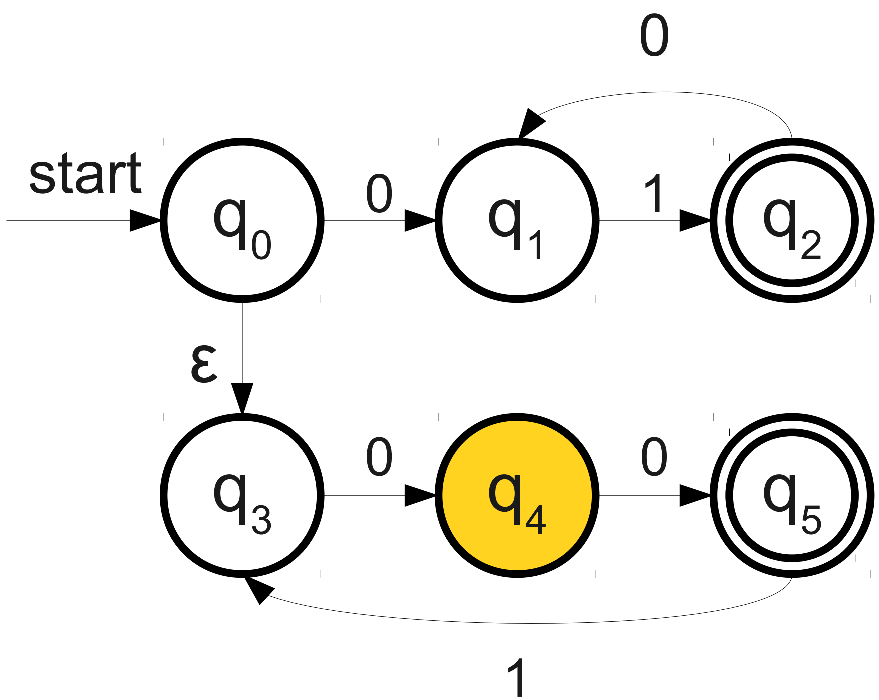


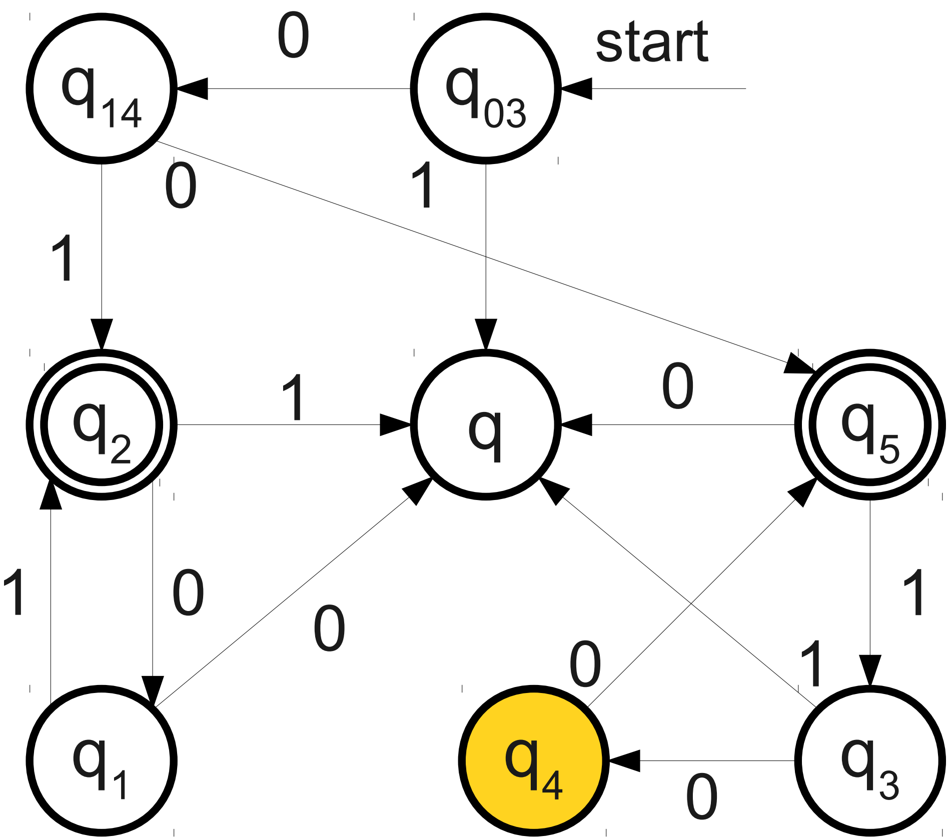
Simulating an NFA with a DFA



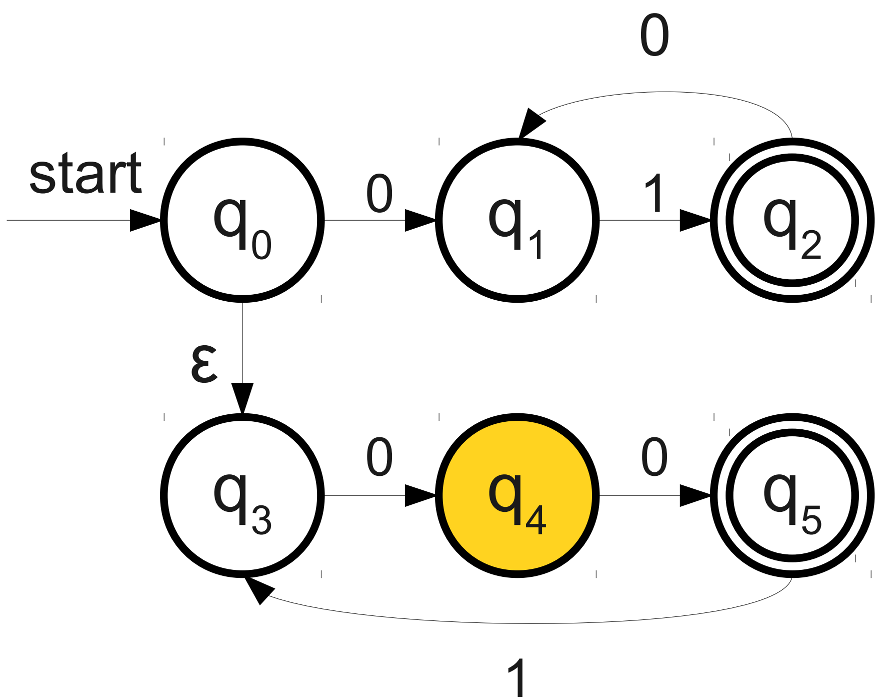


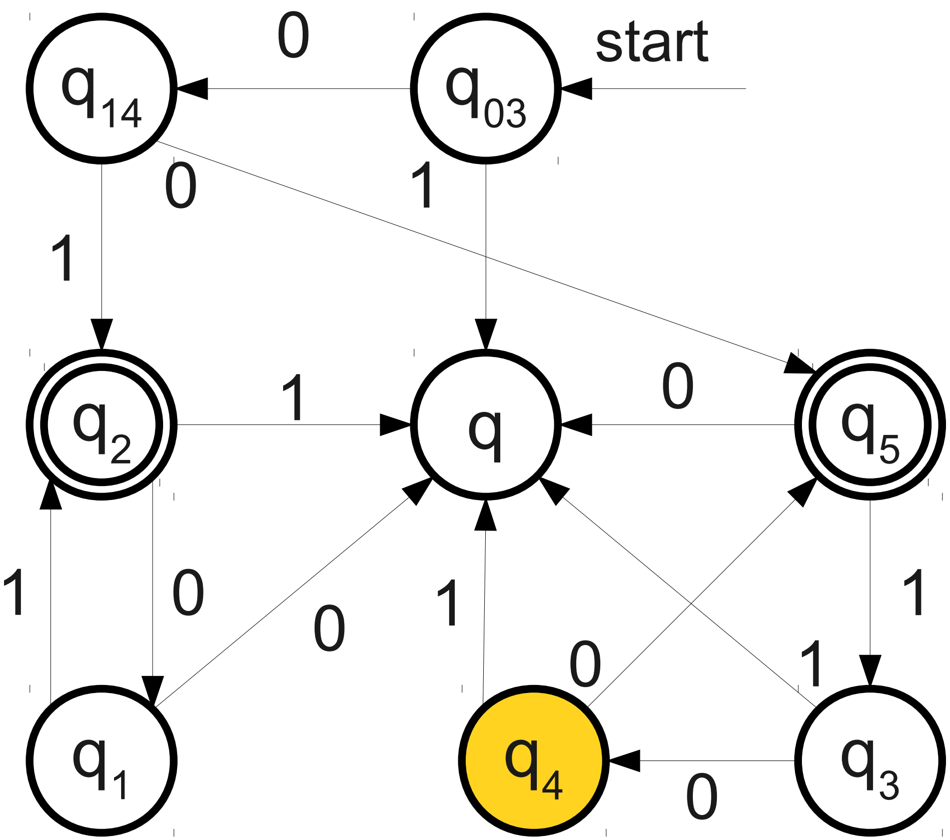
Simulating an NFA with a DFA



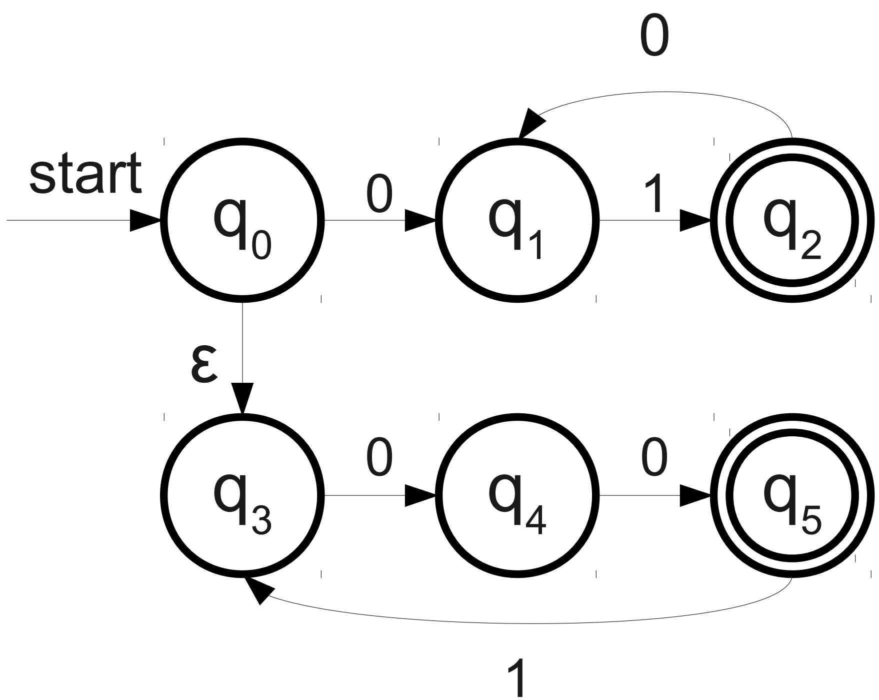


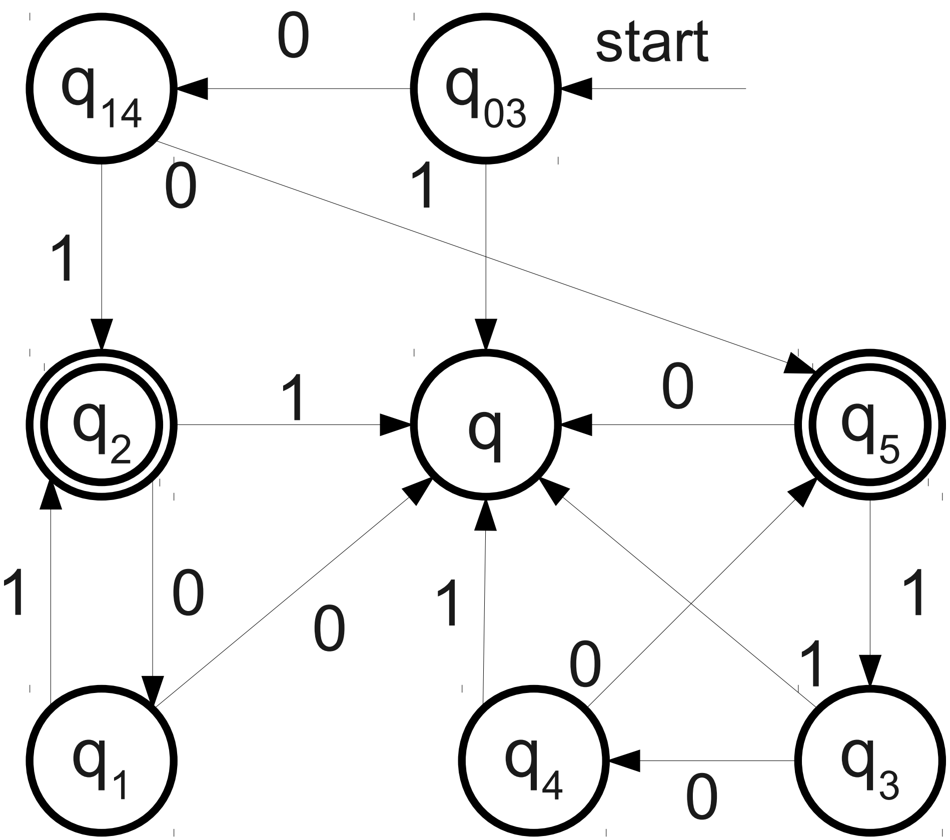
Simulating an NFA with a DFA



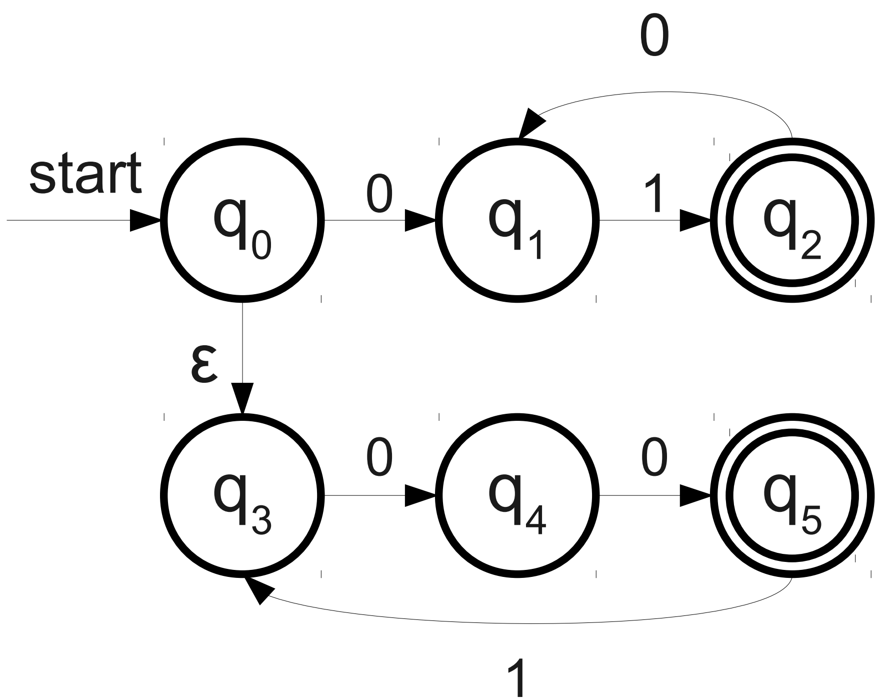


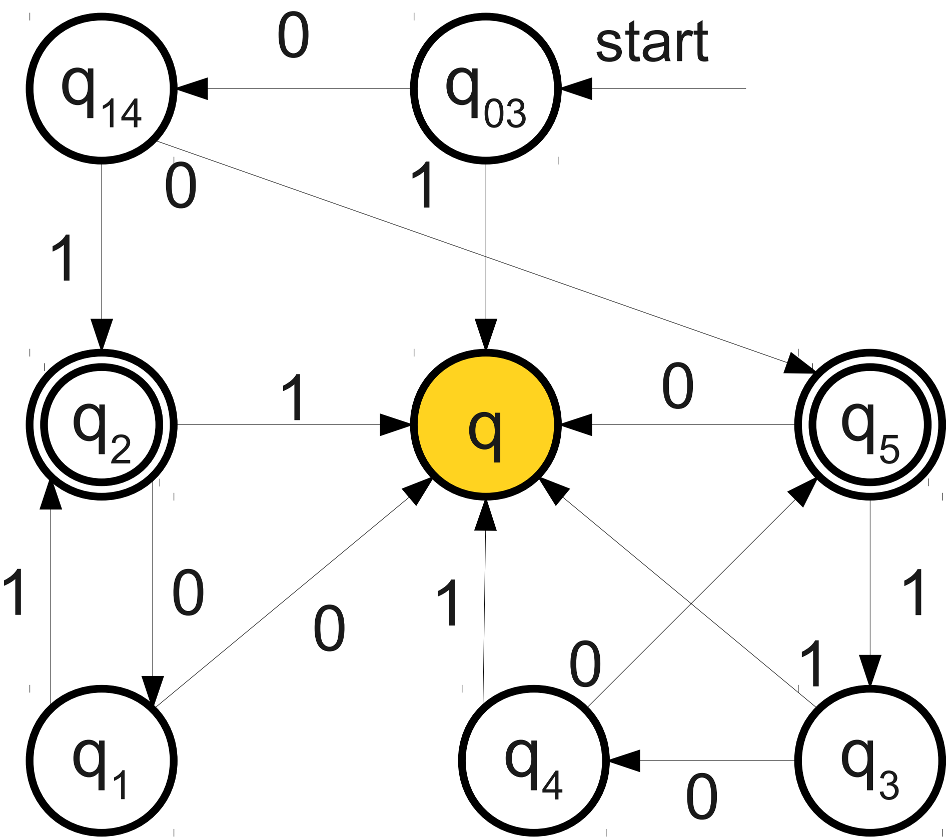
Simulating an NFA with a DFA



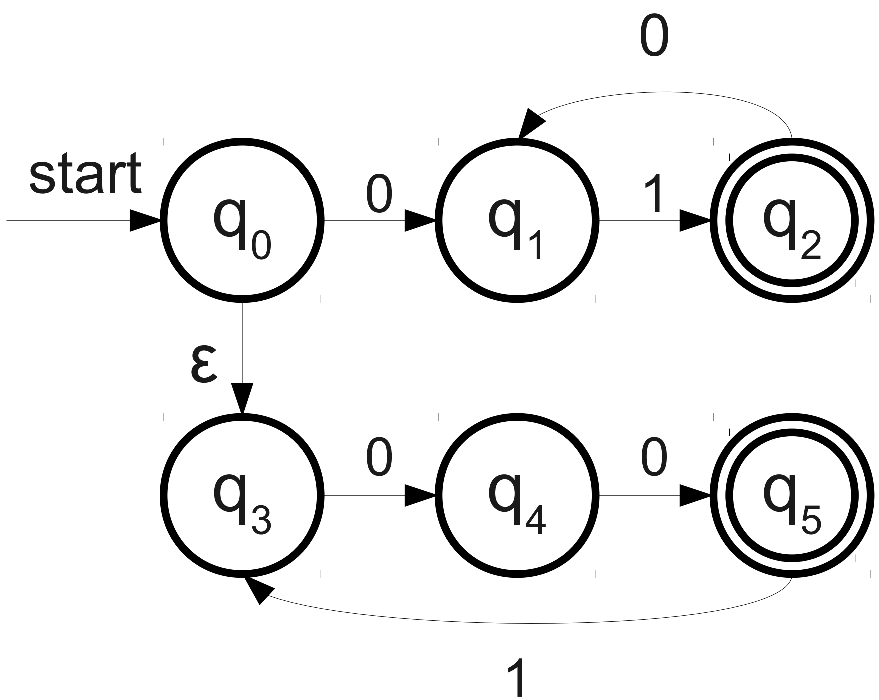


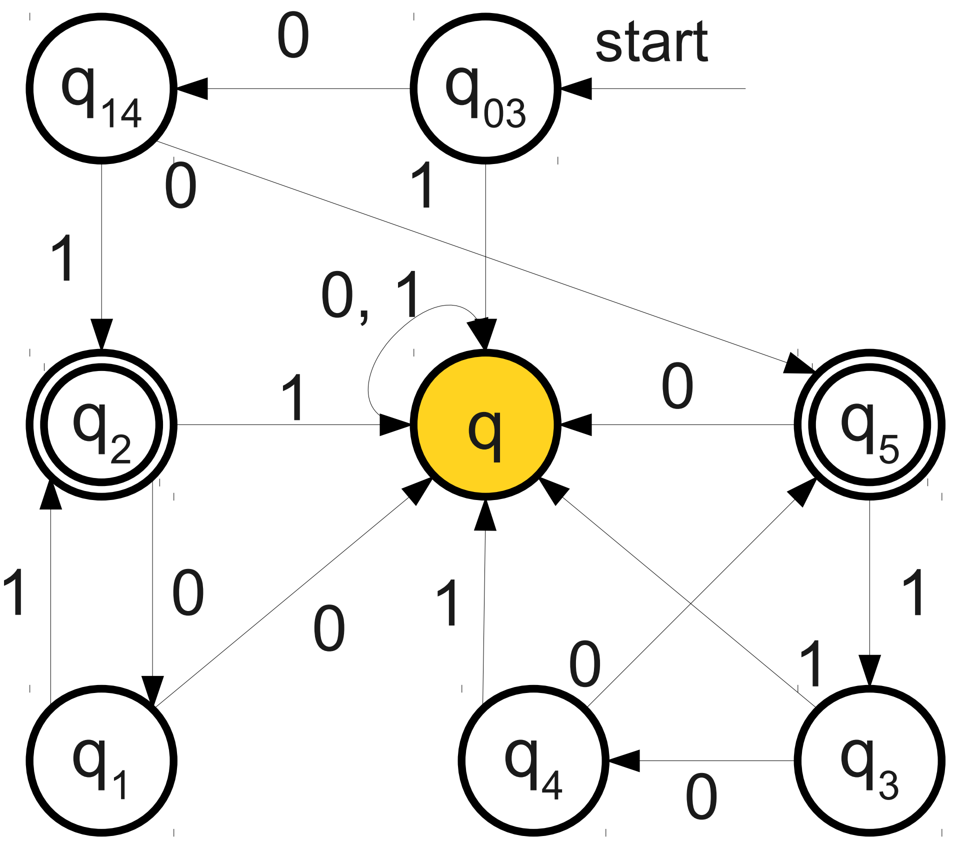
Simulating an NFA with a DFA



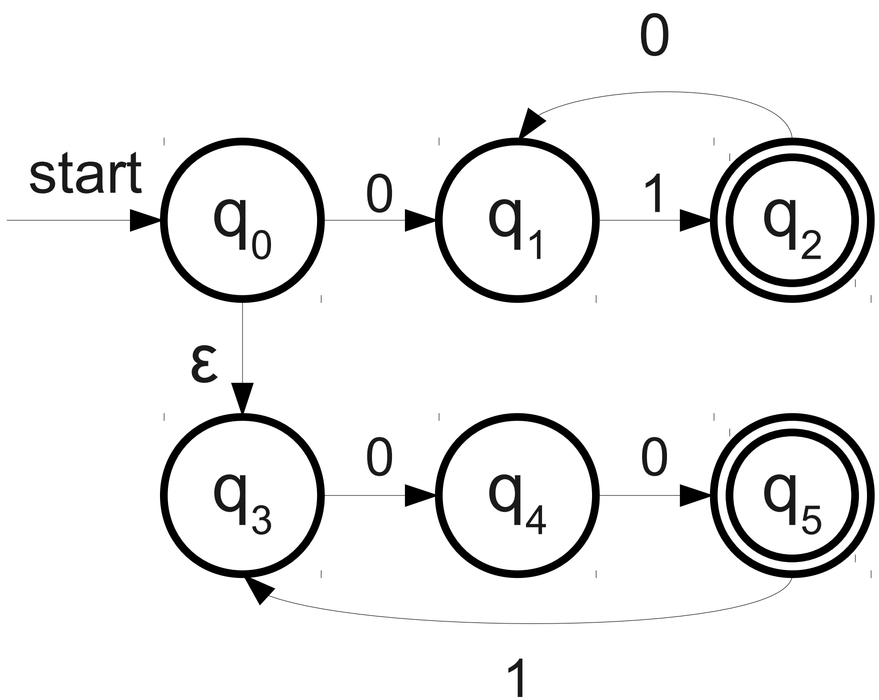


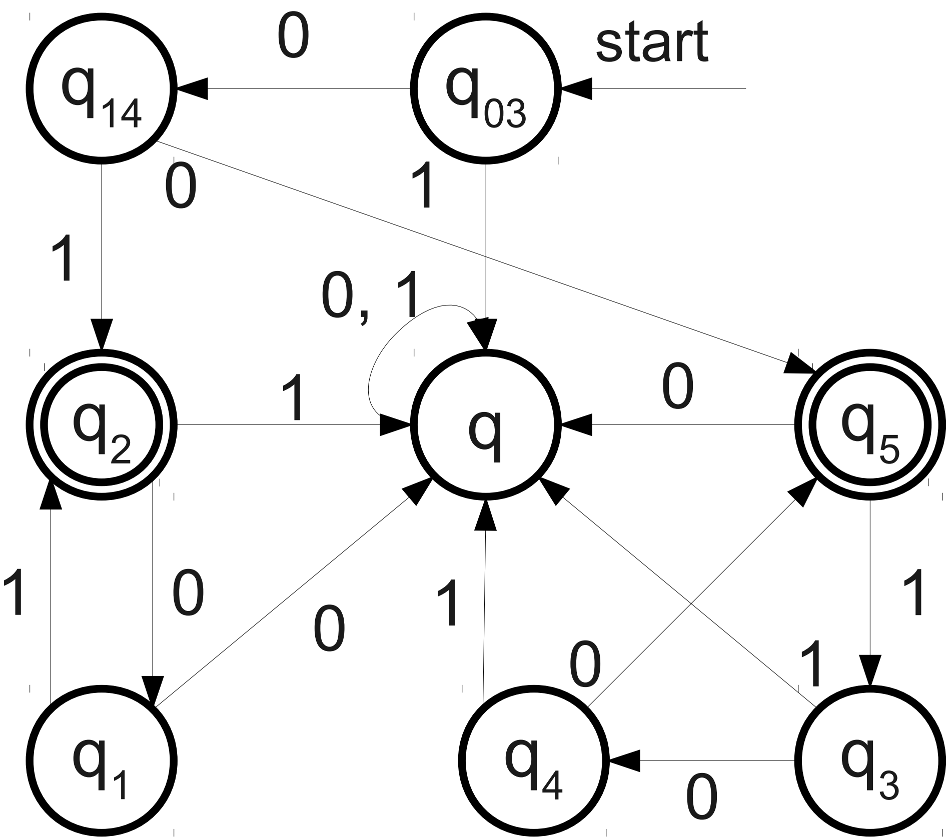
Simulating an NFA with a DFA



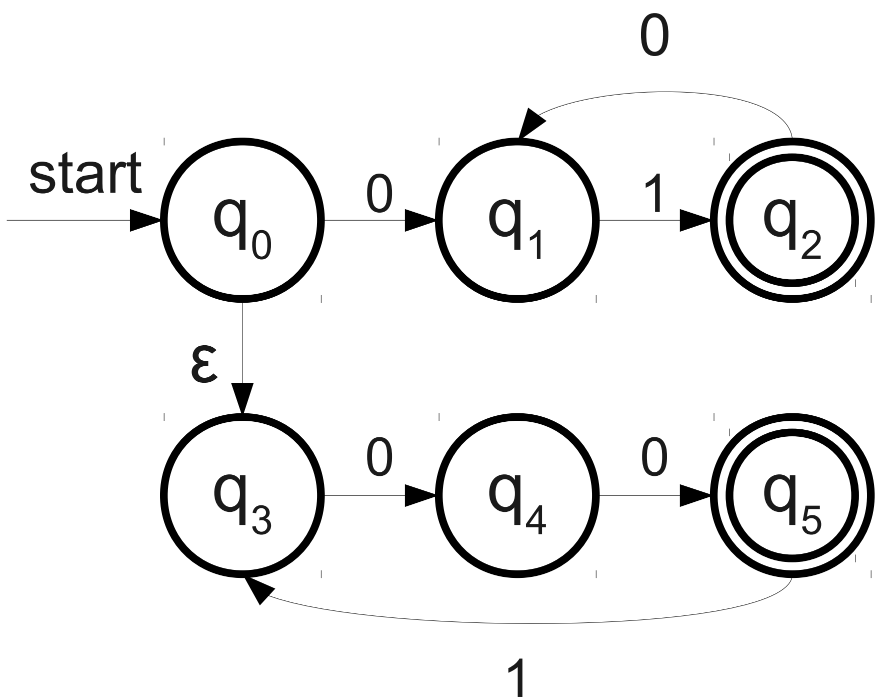


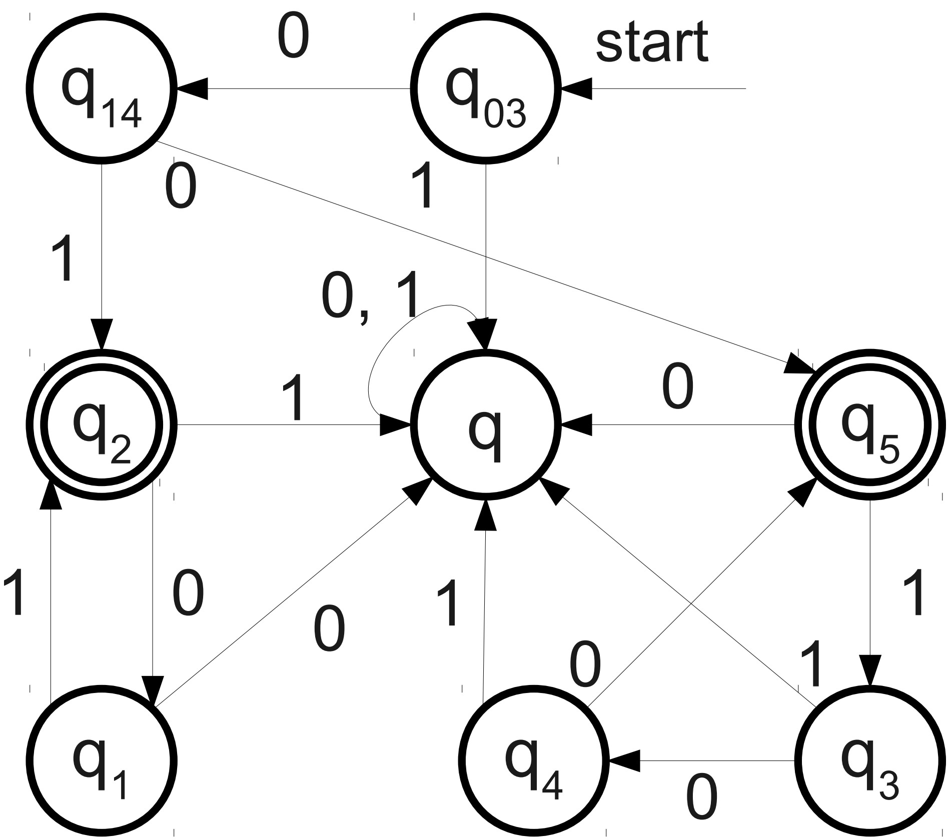
Simulating an NFA with a DFA



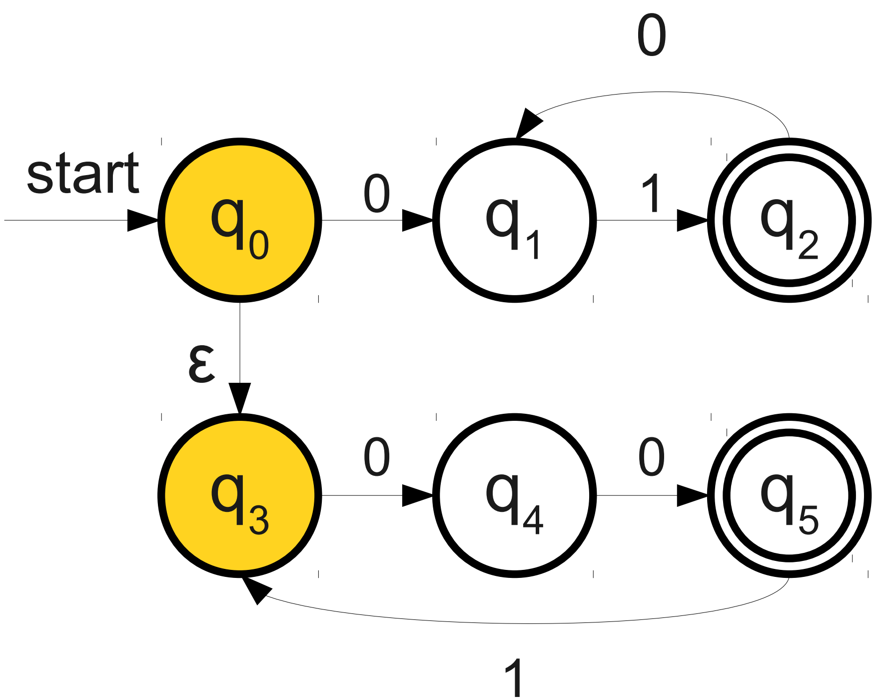


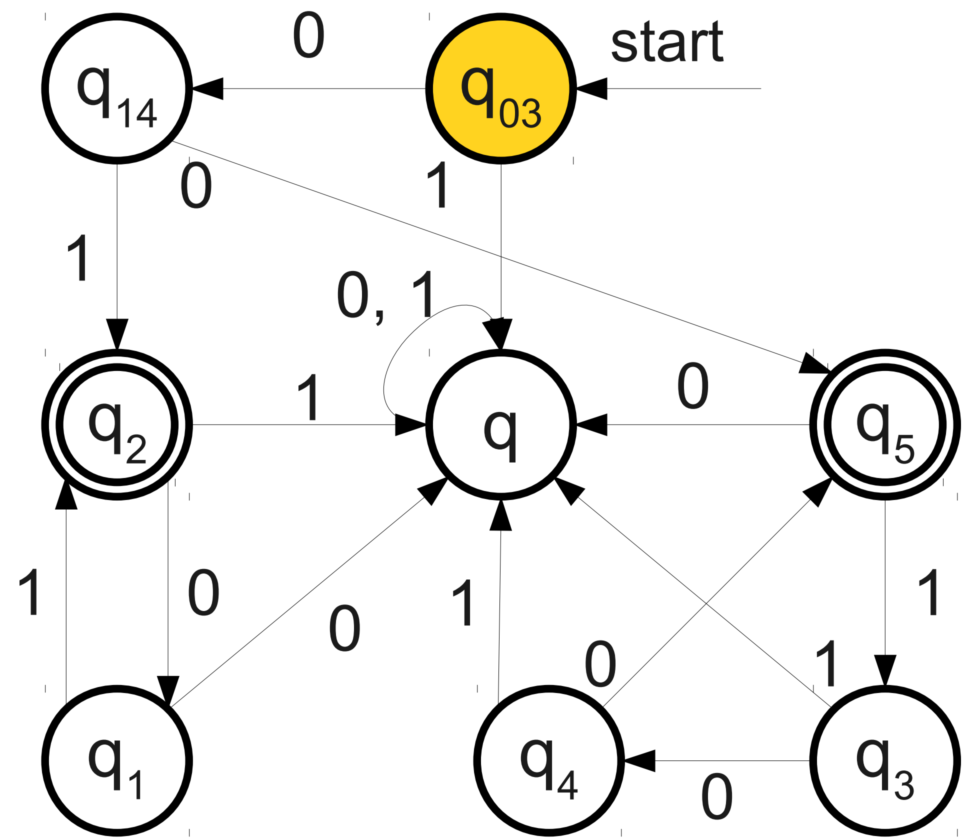
Simulating an NFA with a DFA



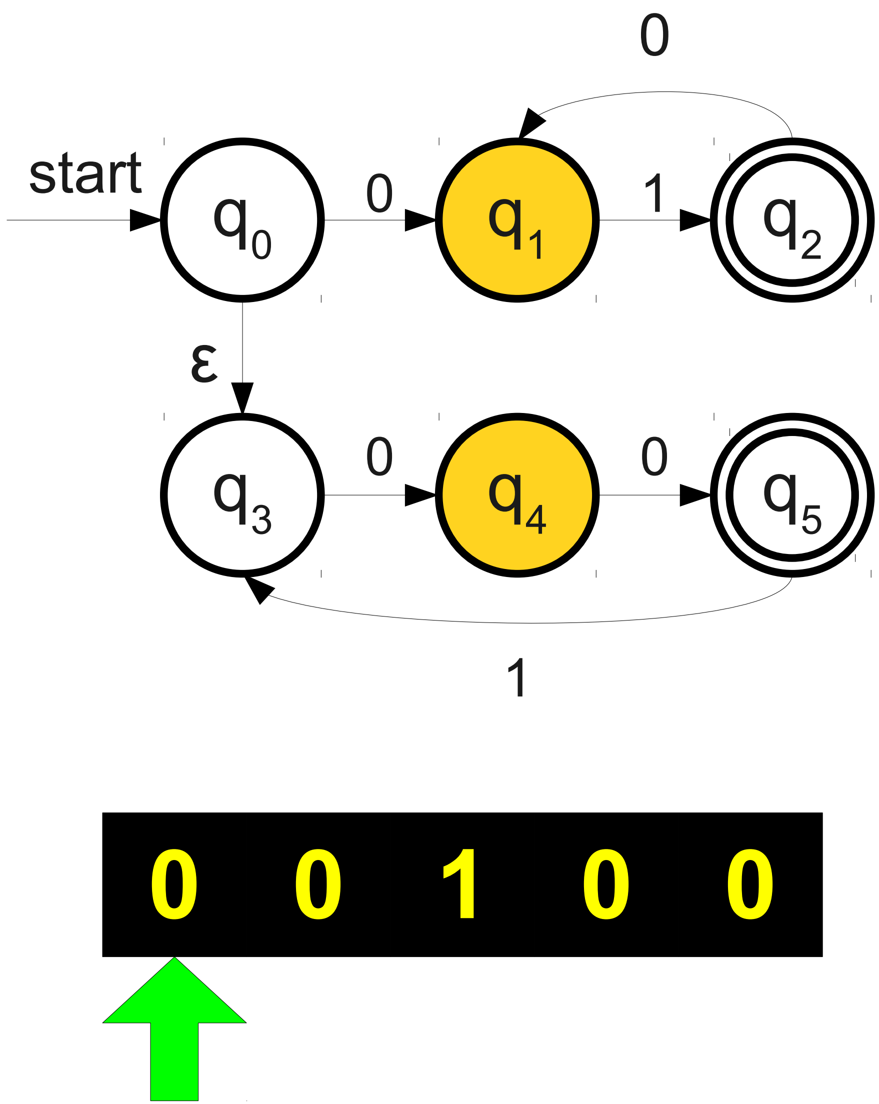


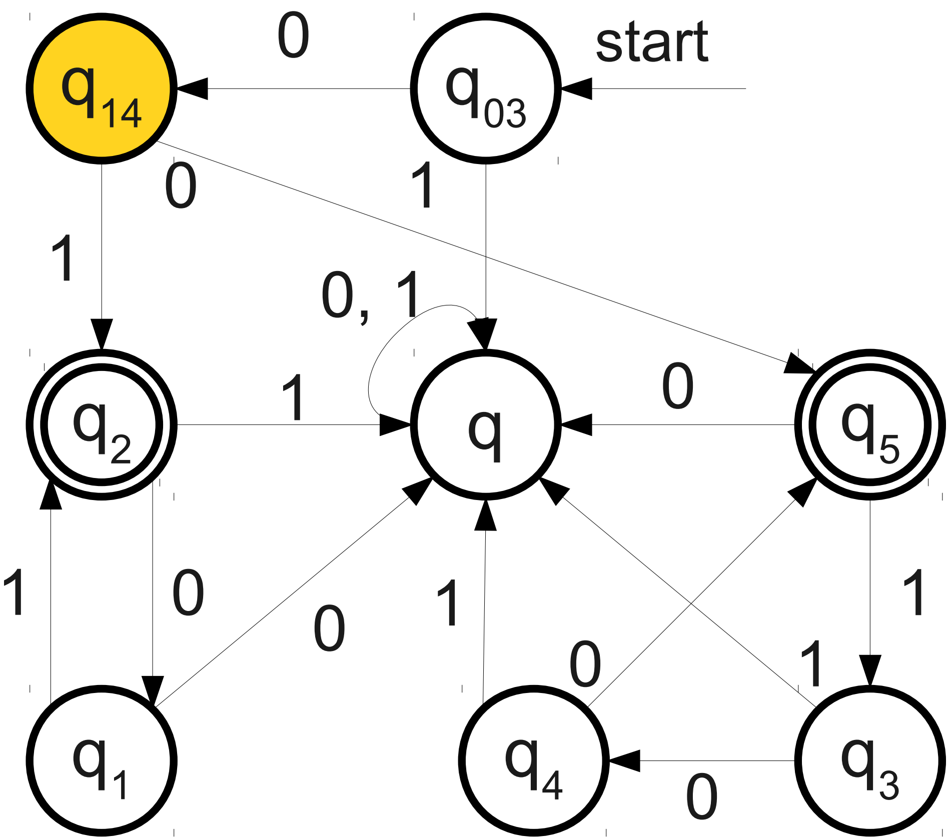
Simulating an NFA with a DFA



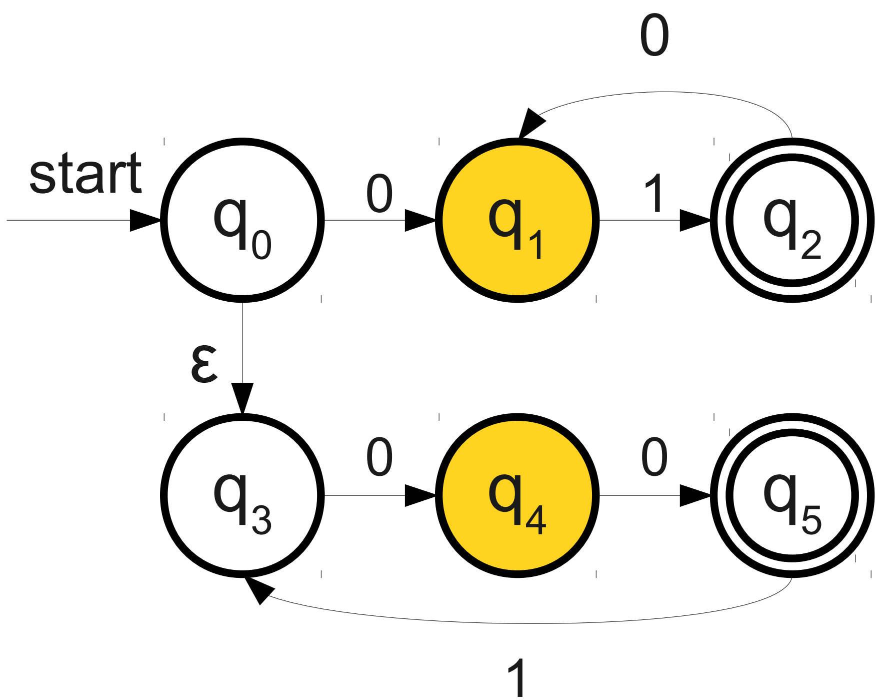


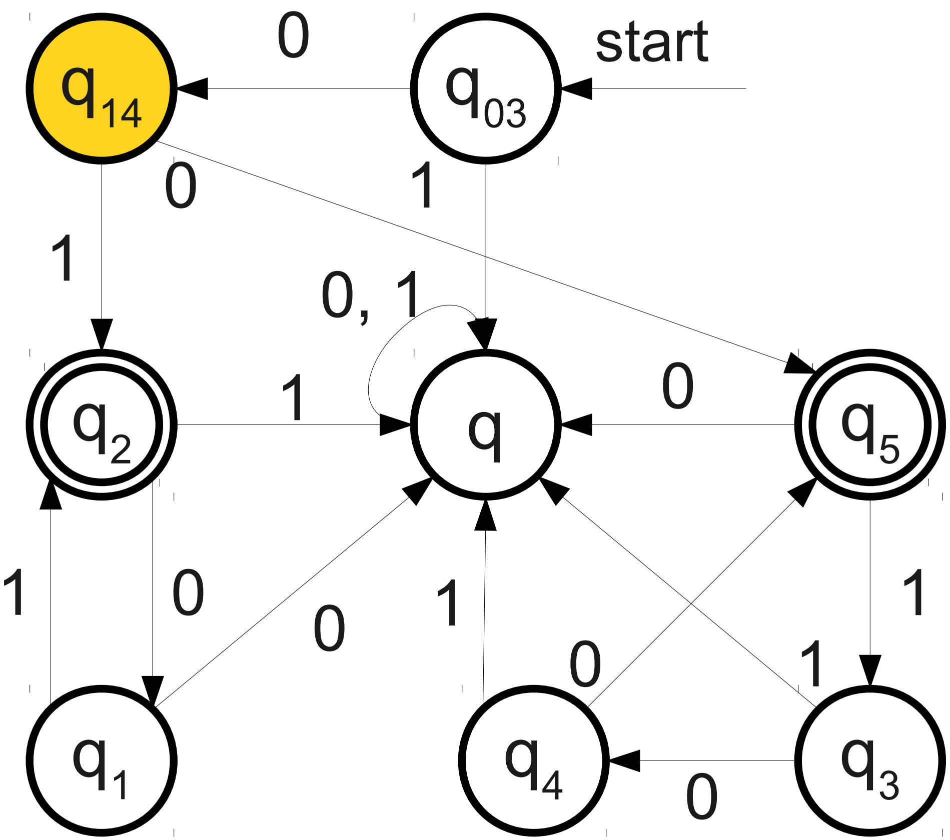
Simulating an NFA with a DFA

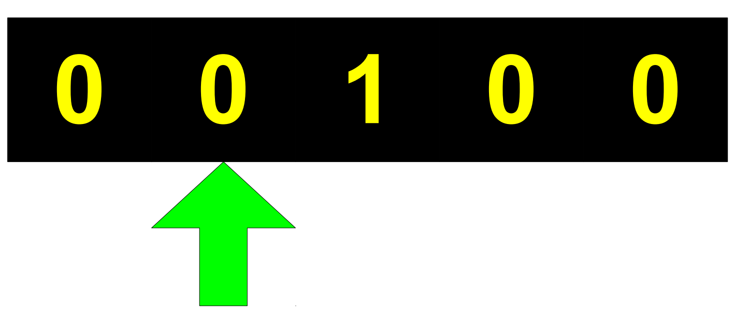




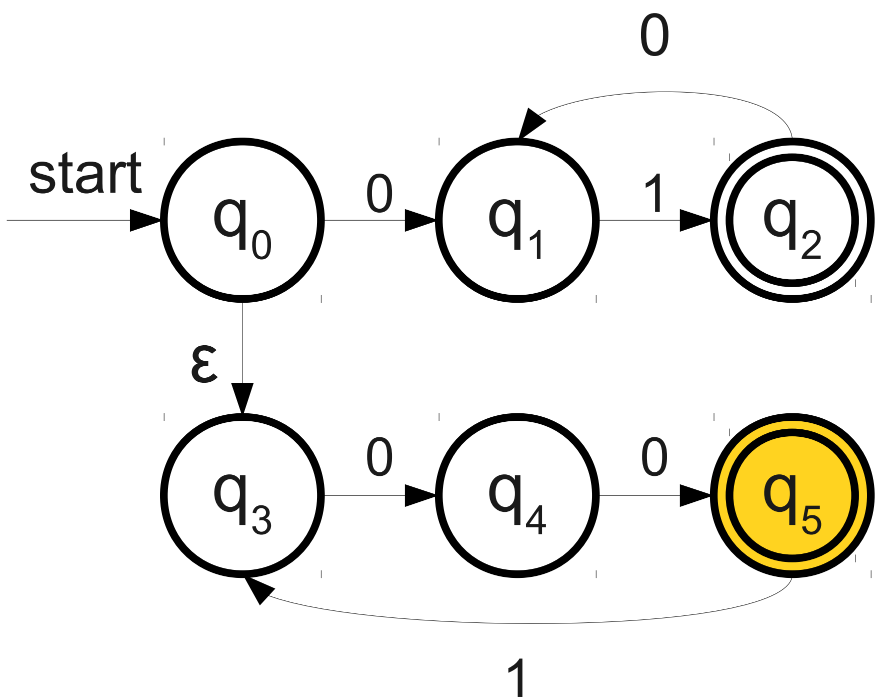
Simulating an NFA with a DFA

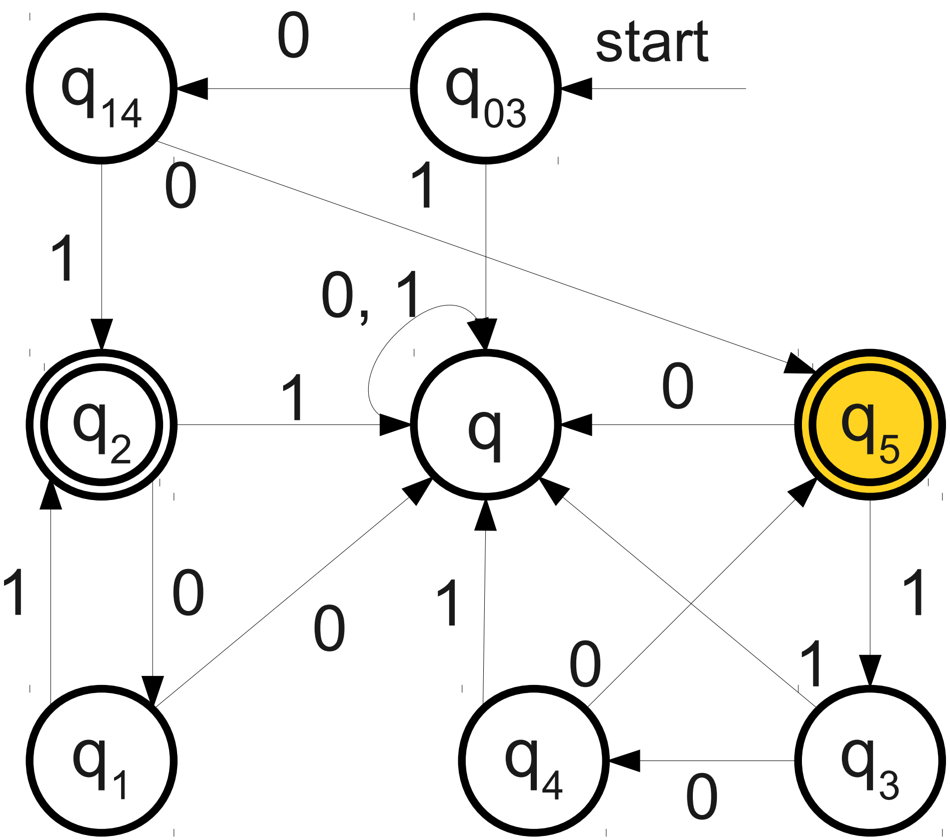


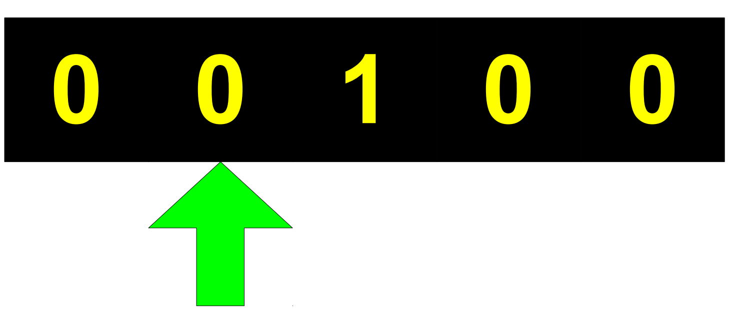




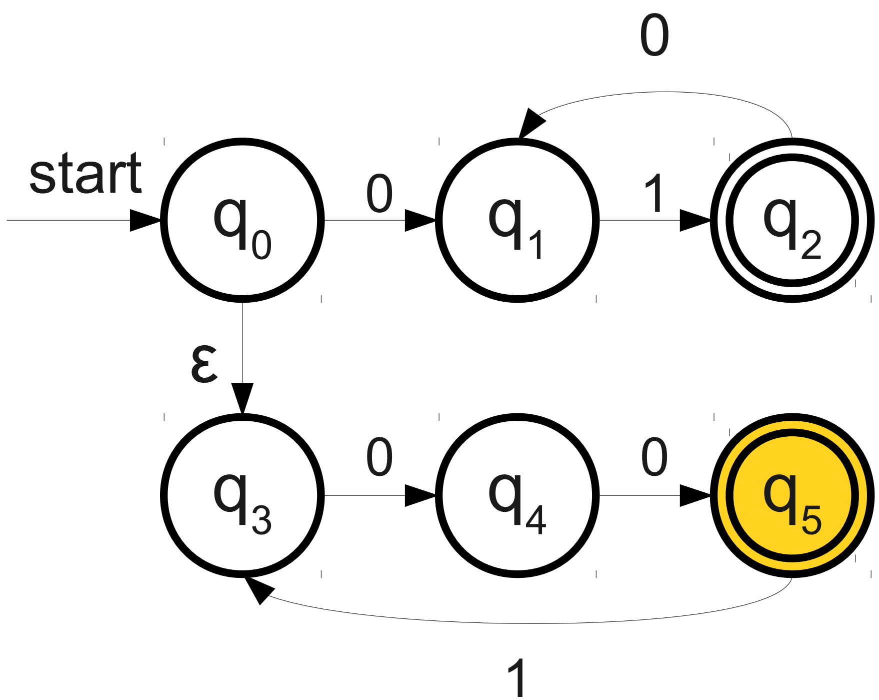
Simulating an NFA with a DFA

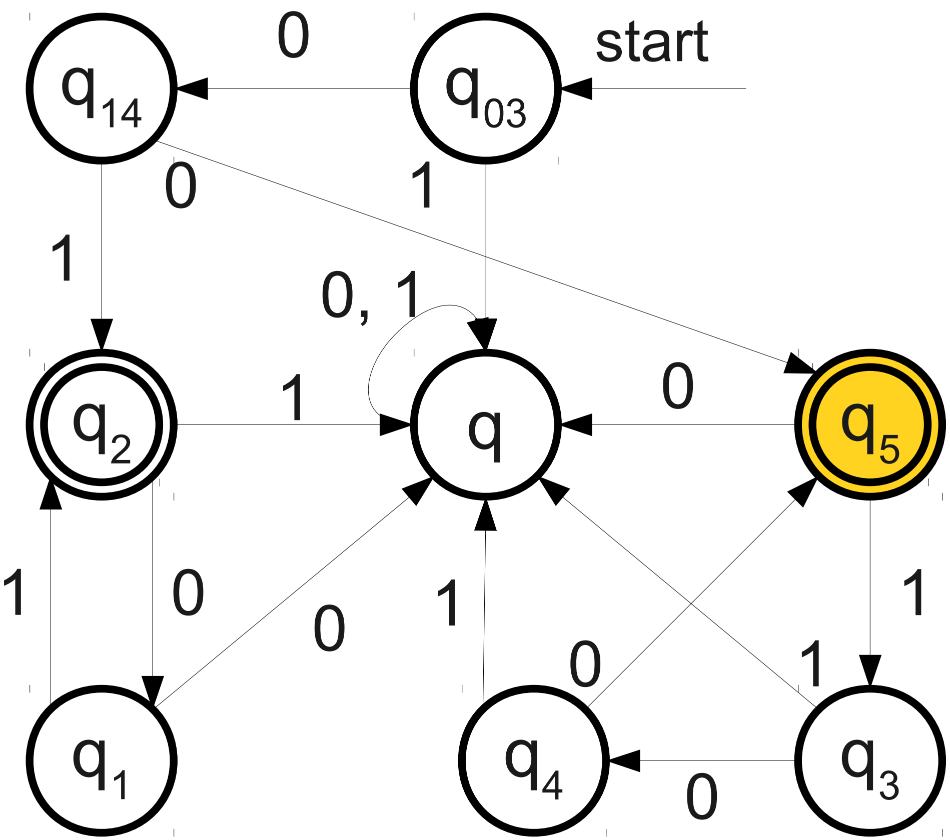


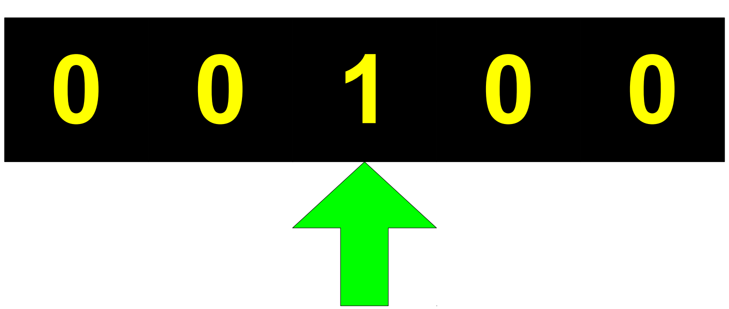




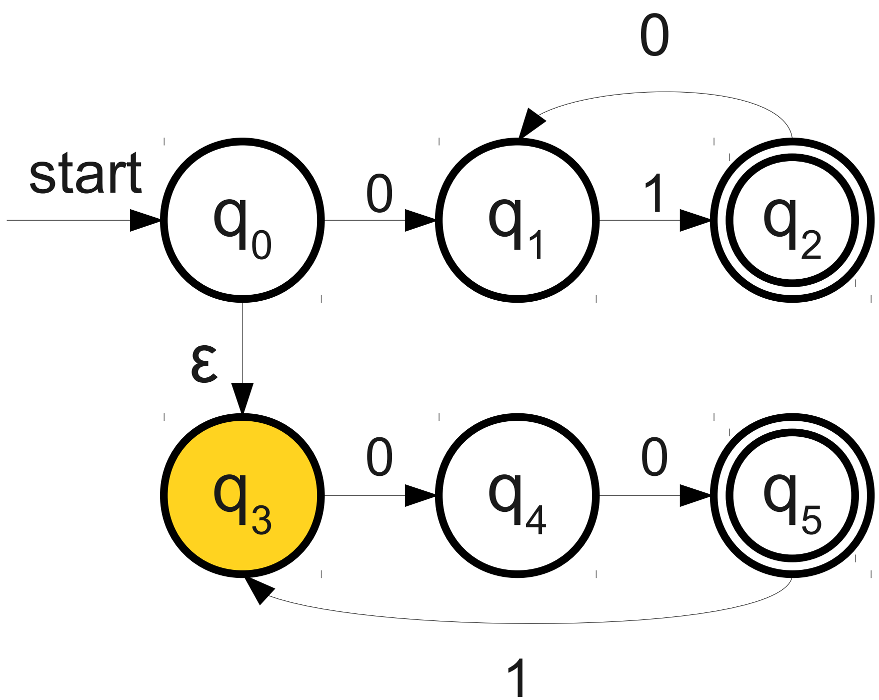
Simulating an NFA with a DFA

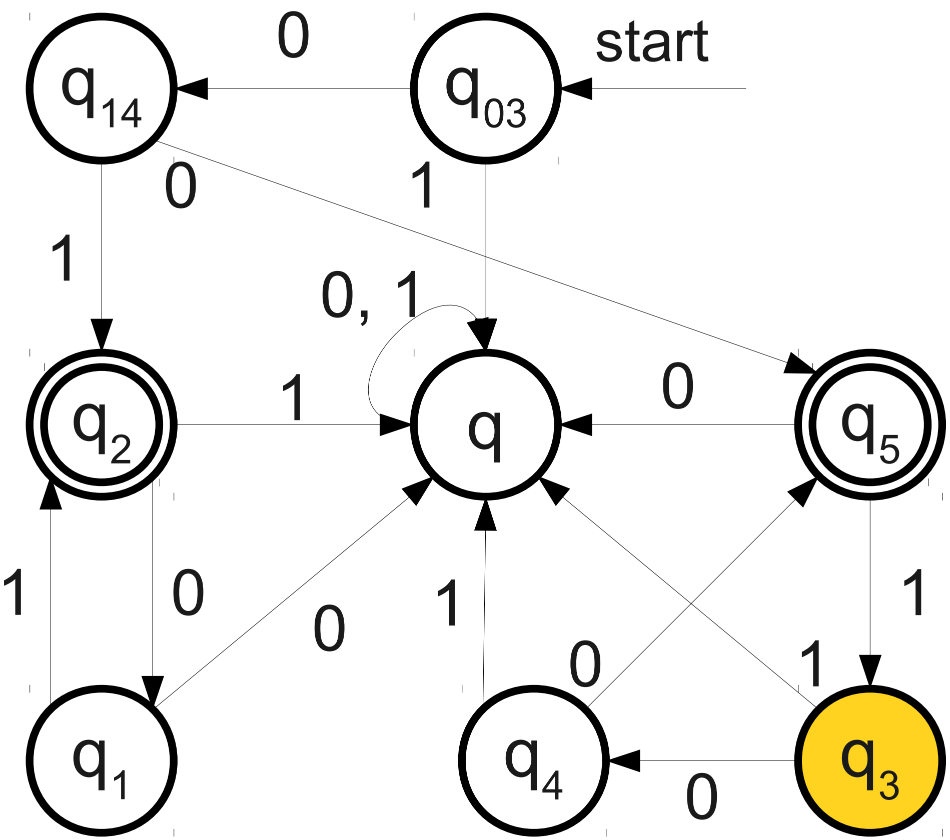


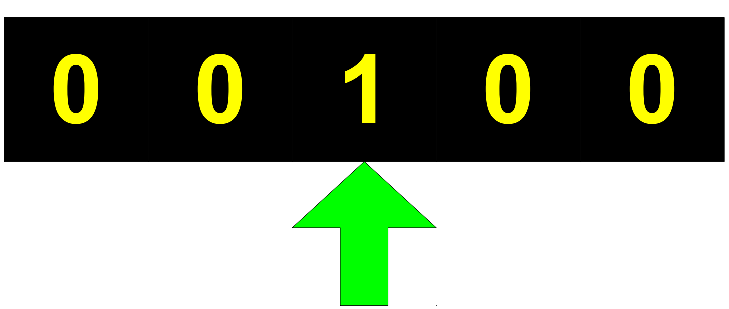




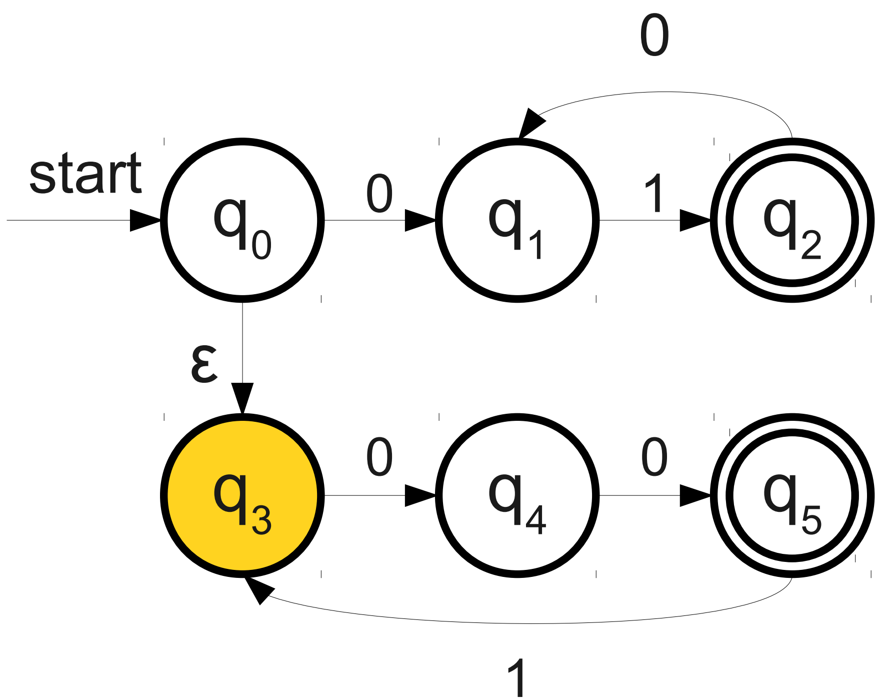
Simulating an NFA with a DFA

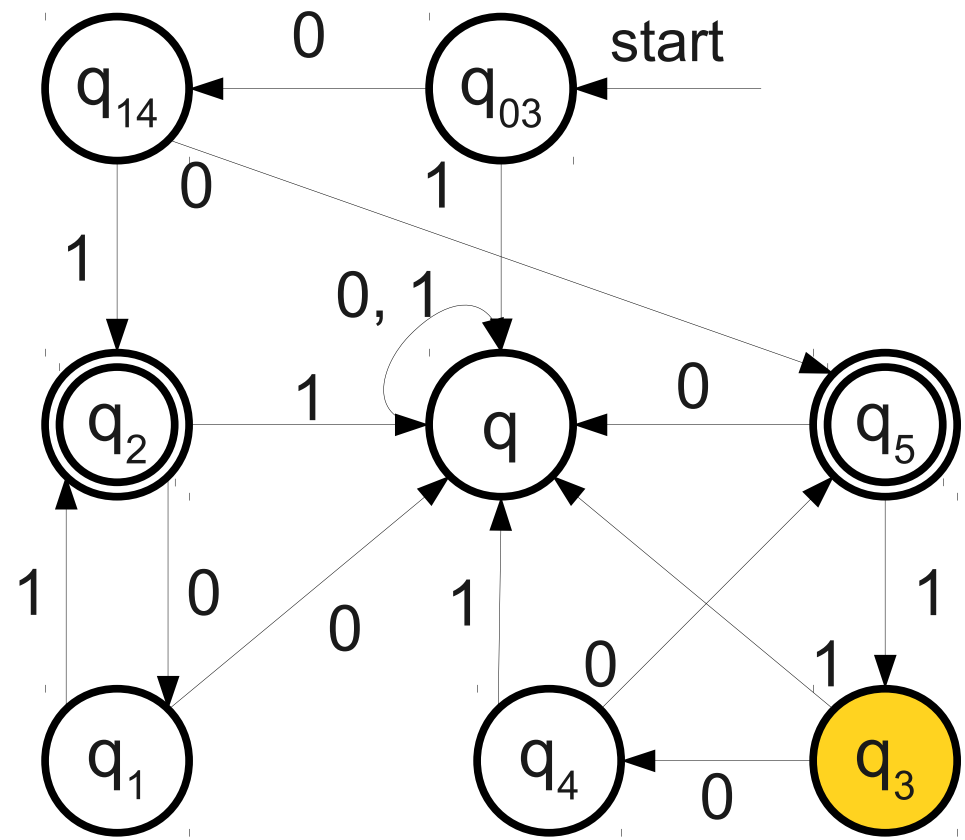


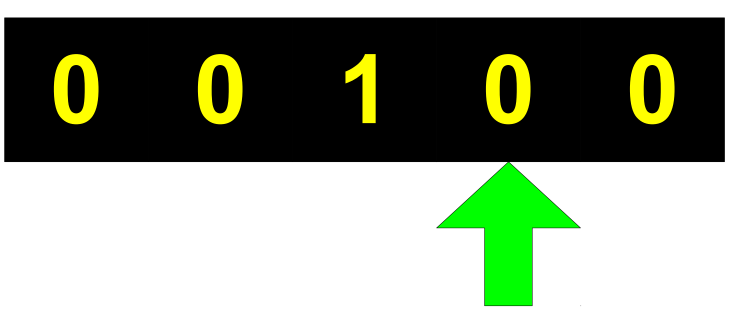




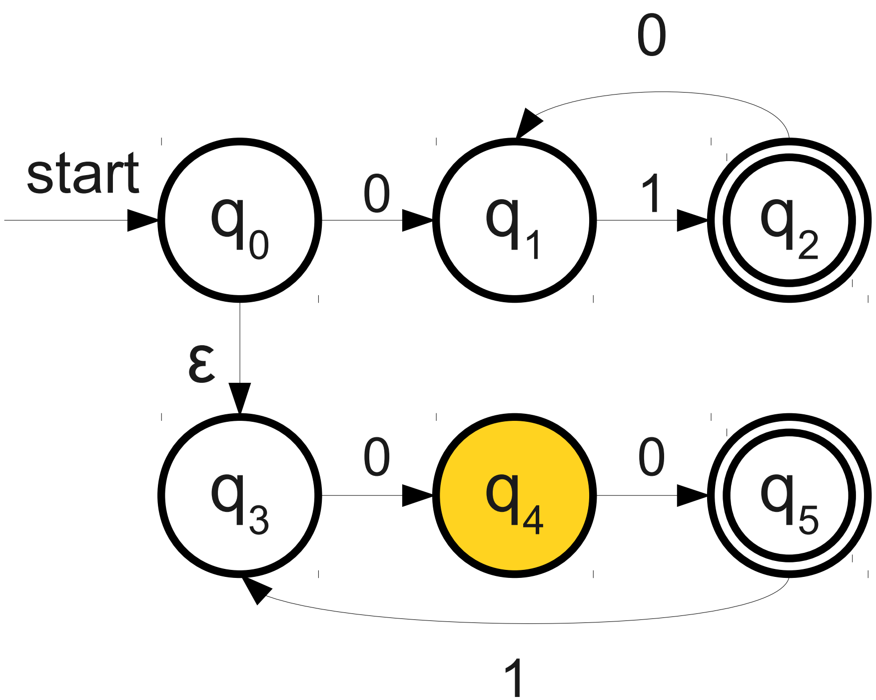
Simulating an NFA with a DFA

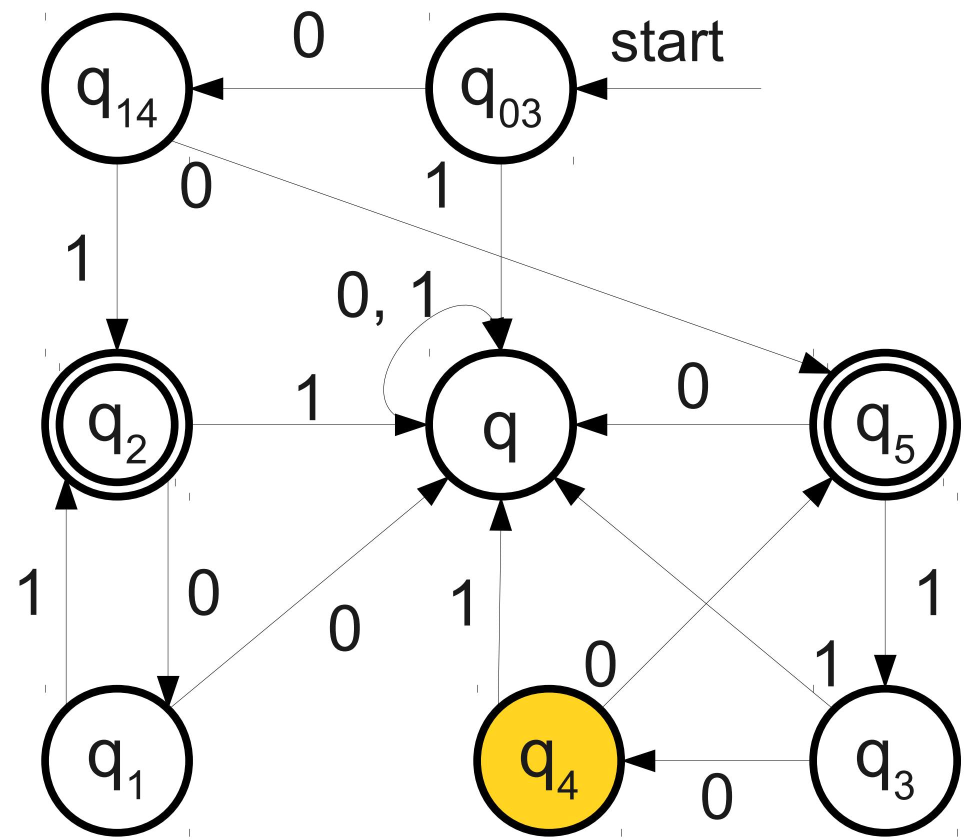


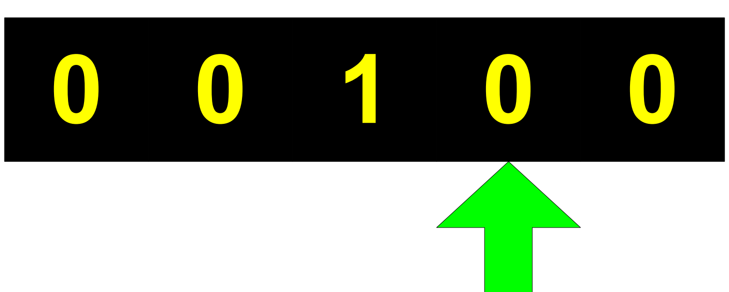




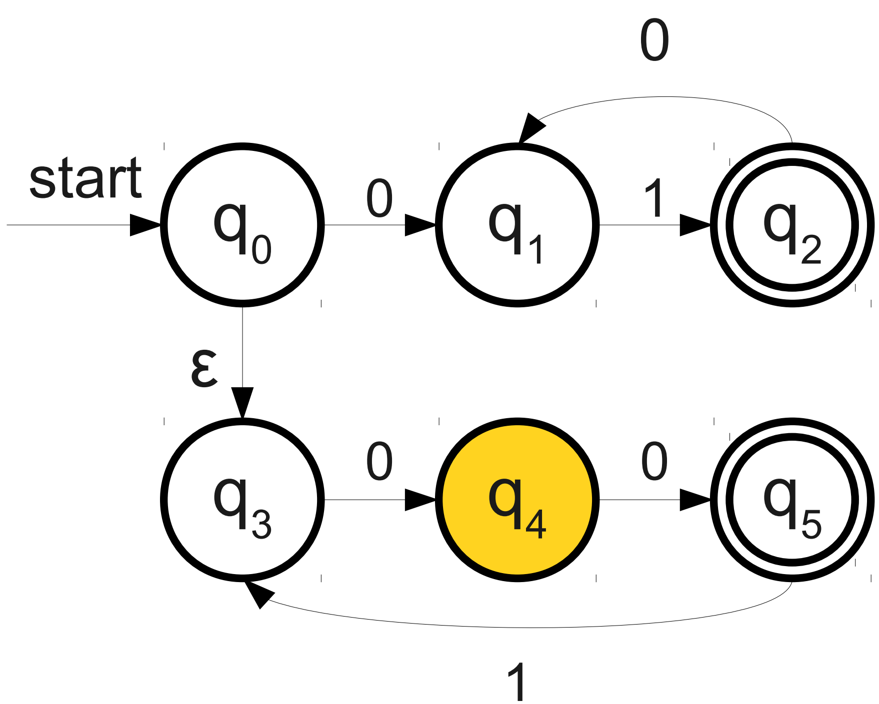
Simulating an NFA with a DFA

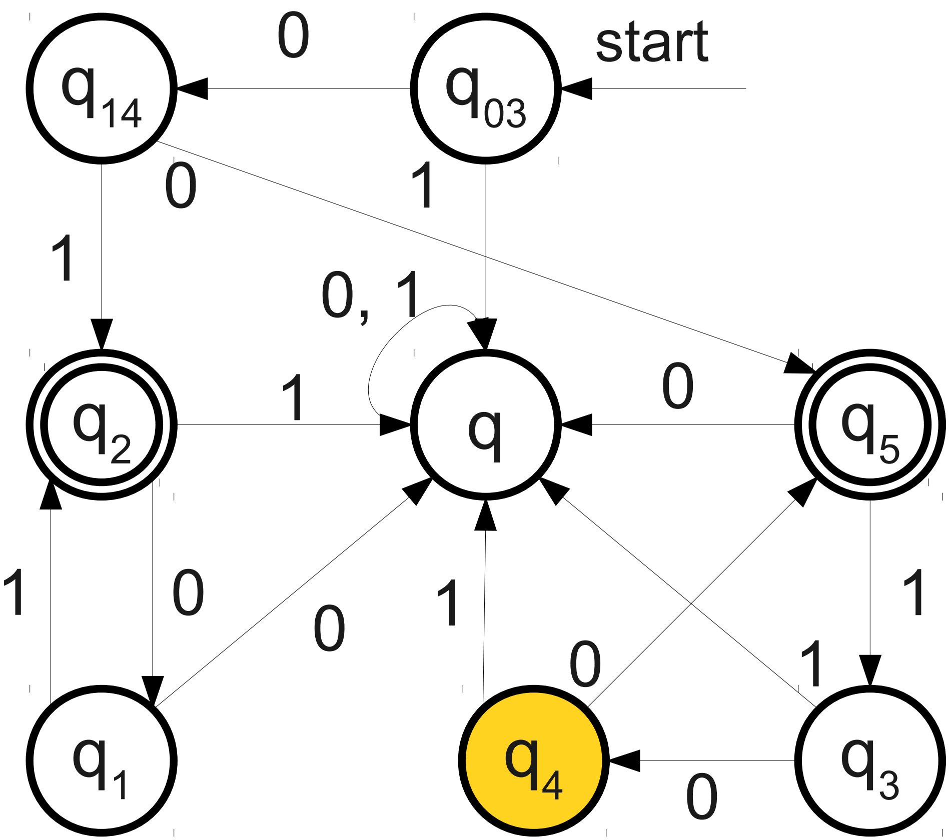




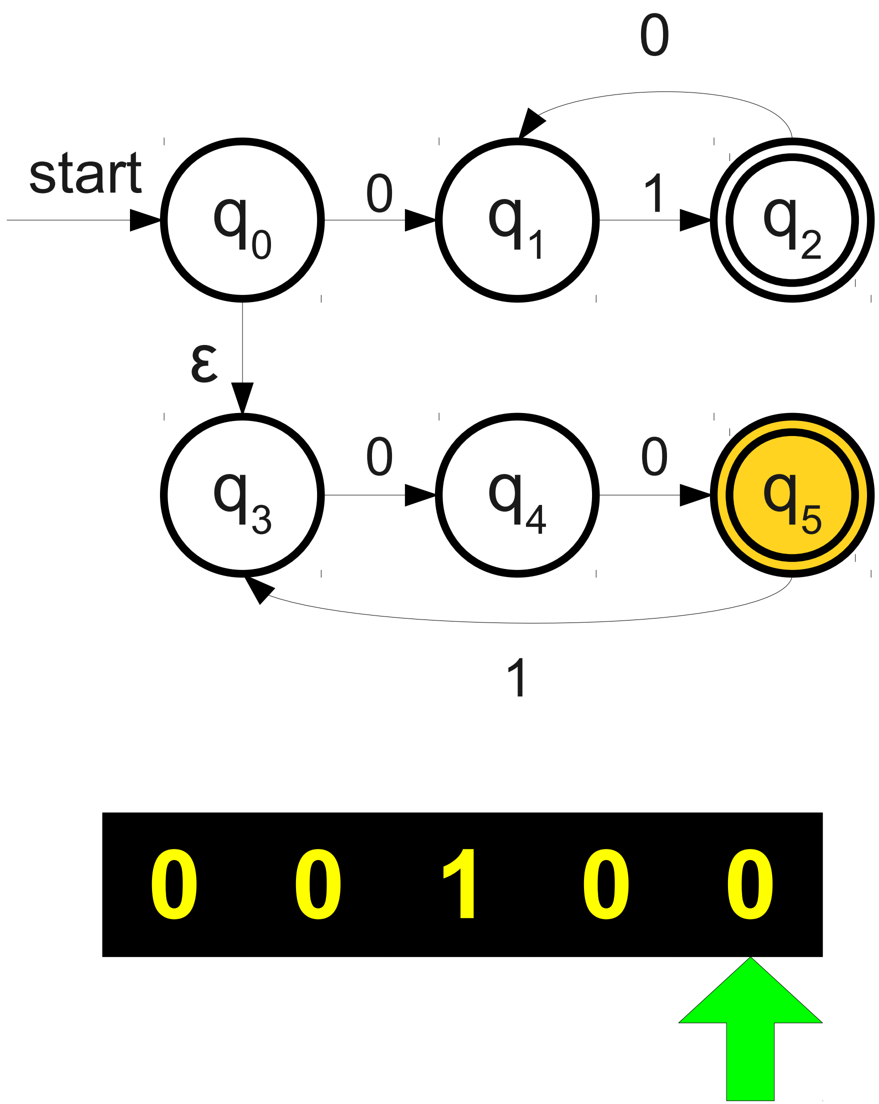


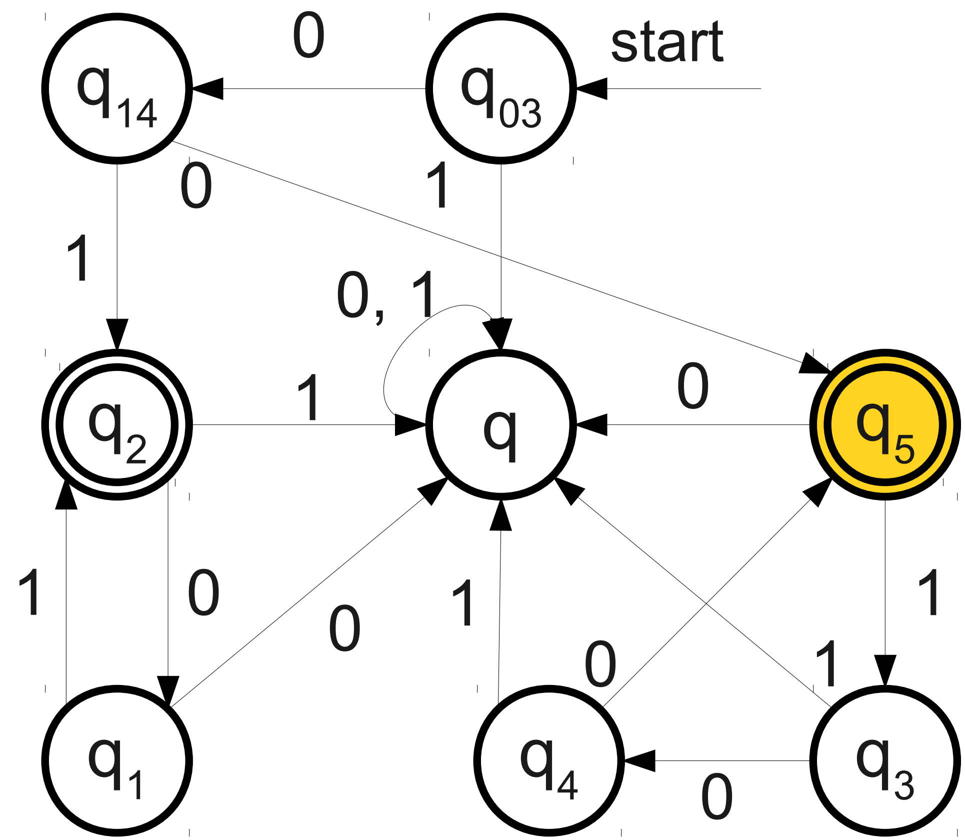
Simulating an NFA with a DFA



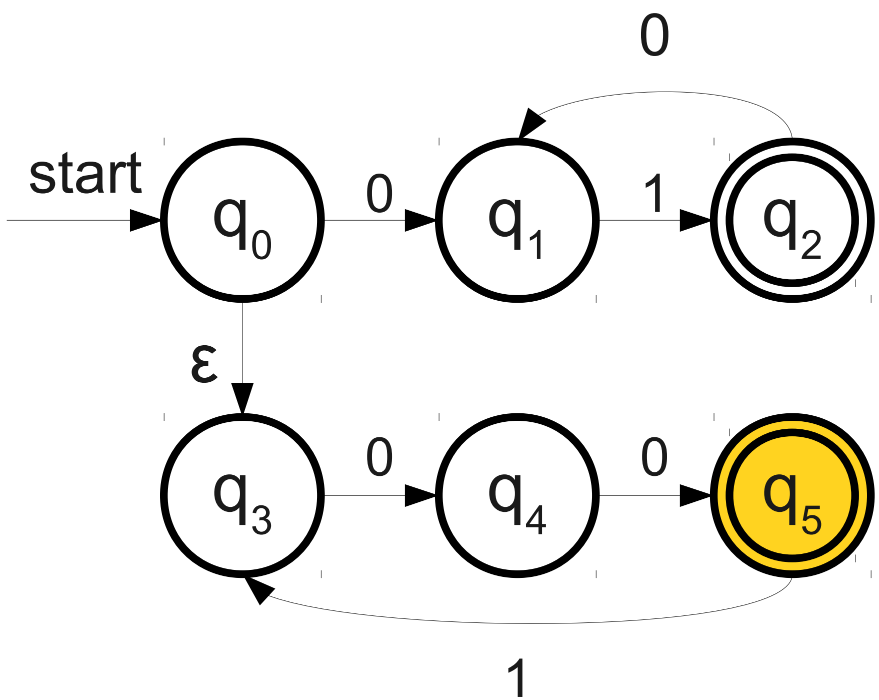


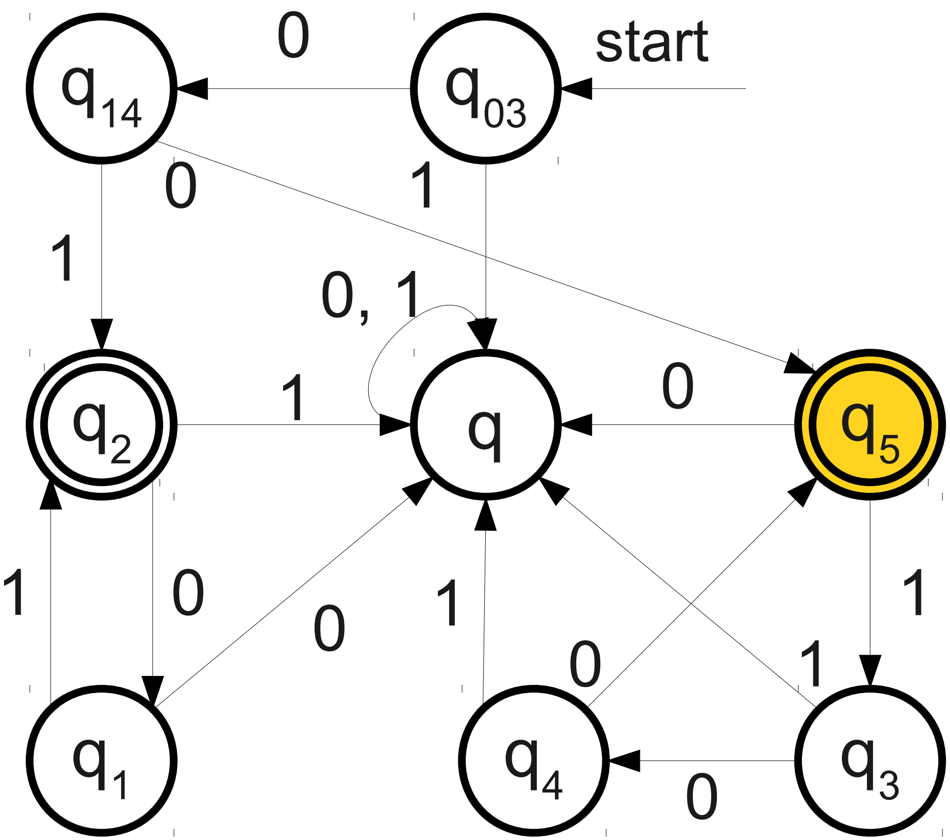
Simulating an NFA with a DFA





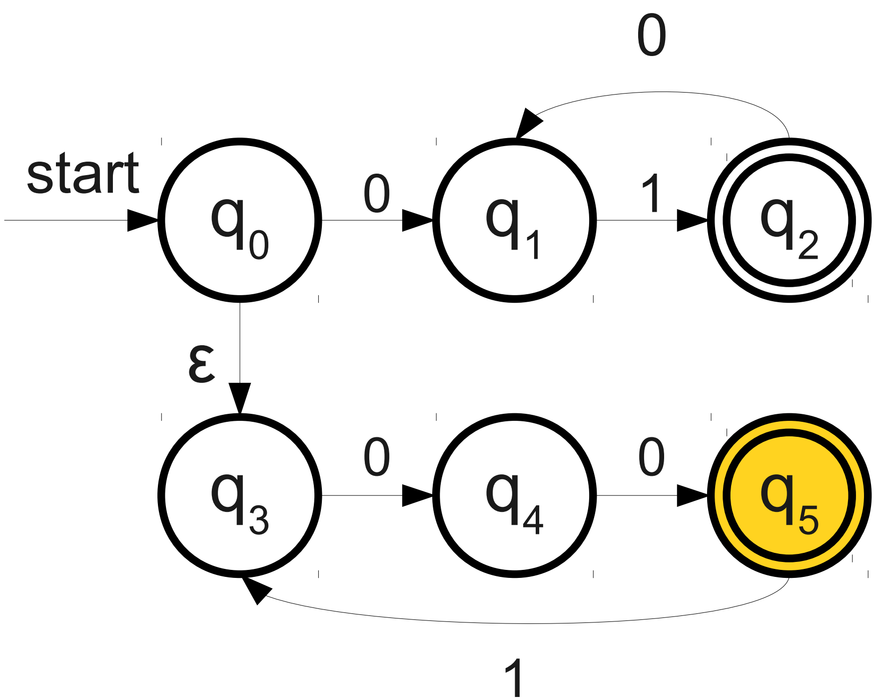
Simulating an NFA with a DFA

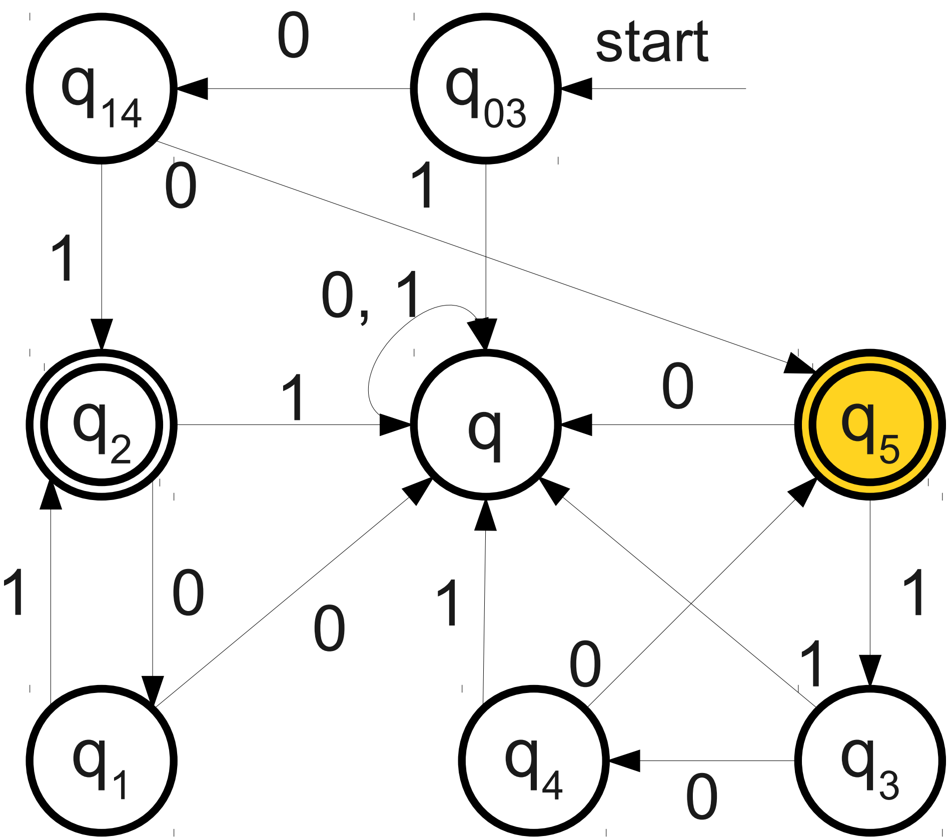




0 0 1 0 0

Simulating an NFA with a DFA





0 0 1 0 0

The Subset Construction

- This construction for transforming an NFA into a DFA is called the **subset construction** (or sometimes the **powerset construction**).
- Intuitively:
  - States of the new DFA correspond to **sets of states** of the NFA.
  - The initial state is the start state, plus all states reachable from the start state via ε-transitions.
  - Transition on state S on character a is found by following all possible transitions on a for each state in S, then taking the set of states reachable from there by ε-transitions.
  - Accepting states are any set of states where *some* state in the set is an accepting state.
- **Read Sipser for a formal account.**