Turing Machines

Implementation of Computational Models (TC2037)

Edgar Covantes Osuna, PhD
edgar.covantes@tec.mx

Languages

What we have seen so far

A **language** is a set of **words**.

How do we **prove** that a language is regular?

How do we **prove** that a language is context-free?

Languages

What we have seen so far

A **language** is a set of **words**.

How do we **prove** that a language is regular?

How do we **prove** that a language is context-free?

Languages

What we have seen so far

A **language** is a set of **words**.

How do we **prove** that a language is regular?

How do we **prove** that a language is context-free?

The machines and their languages

What we have seen so far

The **Finite Automatons**, the **regular expressions** and the **regular grammars** serve to represent regular languages.

The **Pushdown automatons** and the **context-free grammars** serve to represent context-free languages.

However, there are other languages that we cannot represent with any of these tools.

The machines and their languages

What we have seen so far

The **Finite Automatons**, the **regular expressions** and the **regular grammars** serve to represent regular languages.

The **Pushdown automatons** and the **context-free grammars** serve to represent context-free languages.

However, there are other languages that we cannot represent with any of these tools.

The machines and their languages

What we have seen so far

The **Finite Automatons**, the **regular expressions** and the **regular grammars** serve to represent regular languages.

The **Pushdown automatons** and the **context-free grammars** serve to represent context-free languages.

However, there are other languages that we cannot represent with any of these tools.

The machines and their languages

What we have seen so far

For example, let us try to represent $\{a^n b^n c^n | n \geq 0\}$ with a PA:

- For each a we define a counter in the stack
- For each b we remove a counter from the stack
- We can't count the cs.

Or if we change the order or we use more counters?, it is simply not possible to do it with a PA.

The machines and their languages

What we have seen so far

For example, let us try to represent $\{a^n b^n c^n | n \geq 0\}$ with a PA:

- For each a we define a counter in the stack
- For each b we remove a counter from the stack
- We can't count the cs.

Or if we change the order or we use more counters?, it is simply not possible to do it with a PA.

The machines and their languages

What we have seen so far

For example, let us try to represent $\{a^n b^n c^n | n \geq 0\}$ with a PA:

- For each a we define a counter in the stack
- For each b we remove a counter from the stack
- We can't count the cs.

Or if we change the order or we use more counters?, it is simply not possible to do it with a PA.

The machines and their languages

What we have seen so far

For example, let us try to represent $\{a^n b^n c^n | n \geq 0\}$ with a PA:

- For each a we define a counter in the stack
- For each b we remove a counter from the stack
- We can't count the cs.

Or if we change the order or we use more counters?, it is simply not possible to do it with a PA.

The machines and their languages

What we have seen so far

For example, let us try to represent $\{a^n b^n c^n | n \geq 0\}$ with a PA:

- For each a we define a counter in the stack
- For each b we remove a counter from the stack
- We can't count the cs.

Or if we change the order or we use more counters?, it is simply not possible to do it with a PA.

A better use of the memory

Turing and his machine

Clearly the problem we have is memory related: we do not have how to remember the cs.

In a PA we only read the top of the stack. What would happen if we read any part of it?

What is the stack limit? How many symbols can you save?

A better use of the memory

Turing and his machine

Clearly the problem we have is memory related: we do not have how to remember the cs.

In a PA we only read the top of the stack. What would happen if we read any part of it?

What is the stack limit? How many symbols can you save?

A better use of the memory

Turing and his machine

Clearly the problem we have is memory related: we do not have how to remember the cs.

In a PA we only read the top of the stack. What would happen if we read any part of it?

What is the stack limit? How many symbols can you save?

A new machine

Turing and his machine

A Turing machine (TM) fix these problems, *joining* the *input* and the memory. Now, our `Automaton Reloaded` has the following elements:

- A set of control states that is **finite**.
- An **infinite** tape used as its **memory**.
- A tape head that can **read** and **write** a cell at the time.

In each *step* of the computing, the machine

- Writes a symbol in the cell where the tape head is,
- change of state, and
- moves the tape head.

A new machine

Turing and his machine

A Turing machine (TM) fix these problems, *joining* the *input* and the memory. Now, our `Automaton Reloaded` has the following elements:

- A set of control states that is **finite**.
- An **infinite** tape used as its **memory**.
- A tape head that can **read** and **write** a cell at the time.

In each *step* of the computing, the machine

- Writes a symbol in the cell where the tape head is,
- change of state, and
- moves the tape head.

A new machine

Turing and his machine

A Turing machine (TM) fix these problems, *joining* the *input* and the memory. Now, our `Automaton Reloaded` has the following elements:

- A set of control states that is **finite**.
- An **infinite** tape used as its **memory**.
- A tape head that can **read** and **write** a cell at the time.

In each *step* of the computing, the machine

- Writes a symbol in the cell where the tape head is,
- change of state, and
- moves the tape head.

A new machine

Turing and his machine

A Turing machine (TM) fix these problems, *joining* the *input* and the memory. Now, our `Automaton Reloaded` has the following elements:

- A set of control states that is **finite**.
- An **infinite** tape used as its **memory**.
- A tape head that can **read** and **write** a cell at the time.

In each *step* of the computing, the machine

- Writes a symbol in the cell where the tape head is,
- change of state, and
- moves the tape head.

A new machine

Turing and his machine

A Turing machine (TM) fix these problems, *joining* the *input* and the memory. Now, our `Automaton Reloaded` has the following elements:

- A set of control states that is **finite**.
- An **infinite** tape used as its **memory**.
- A tape head that can **read** and **write** a cell at the time.

In each *step* of the computing, the machine

- Writes a symbol in the cell where the tape head is,
- change of state, and
- moves the tape head.

A new machine

Turing and his machine

A Turing machine (TM) fix these problems, *joining* the *input* and the memory. Now, our `Automaton Reloaded` has the following elements:

- A set of control states that is **finite**.
- An **infinite** tape used as its **memory**.
- A tape head that can **read** and **write** a cell at the time.

In each *step* of the computing, the machine

- Writes a symbol in the cell where the tape head is,
- change of state, and
- moves the tape head.

A new machine

Turing and his machine

A Turing machine (TM) fix these problems, *joining* the *input* and the memory. Now, our `Automaton Reloaded` has the following elements:

- A set of control states that is **finite**.
- An **infinite** tape used as its **memory**.
- A tape head that can **read** and **write** a cell at the time.

In each *step* of the computing, the machine

- Writes a symbol in the cell where the tape head is,
- change of state, and
- moves the tape head.

A new machine

Turing and his machine

A Turing machine (TM) fix these problems, *joining* the *input* and the memory. Now, our `Automaton Reloaded` has the following elements:

- A set of control states that is **finite**.
- An **infinite** tape used as its **memory**.
- A tape head that can **read** and **write** a cell at the time.

In each *step* of the computing, the machine

- Writes a symbol in the cell where the tape head is,
- change of state, and
- moves the tape head.

Terminology and language

Turing and his machine

Let M be a Turing machine:

- M accepts a word w if goes to an acceptance state when it reads w. In this case, M **finishes**.
- M rejects a word w if goes to a rejection state when it reads w. In this case, M **ends**.
- M goes to a loop with a word w when reading w it does not enter neither the acceptance or rejection state. In this case, M **does not ends**.

What can happen if it is *not accepted*? What can happen if *it is not rejected*?

Terminology and language

Turing and his machine

Let M be a Turing machine:

- M accepts a word w if goes to an acceptance state when it reads w. In this case, M **finishes**.
- M rejects a word w if goes to a rejection state when it reads w. In this case, M **ends**.
- M goes to a loop with a word w when reading w it does not enter neither the acceptance or rejection state. In this case, M **does not ends**.

What can happen if it is *not accepted*? What can happen if *it is not rejected*?

Terminology and language

Turing and his machine

Let M be a Turing machine:

- M accepts a word w if goes to an acceptance state when it reads w. In this case, M **finishes**.
- M rejects a word w if goes to a rejection state when it reads w. In this case, M **ends**.
- M goes to a loop with a word w when reading w it does not enter neither the acceptance or rejection state. In this case, M **does not ends**.

What can happen if it is *not accepted*? What can happen if *it is not rejected*?

Terminology and language

Turing and his machine

Let M be a Turing machine:

- M accepts a word w if goes to an acceptance state when it reads w. In this case, M **finishes**.
- M rejects a word w if goes to a rejection state when it reads w. In this case, M **ends**.
- M goes to a loop with a word w when reading w it does not enter neither the acceptance or rejection state. In this case, M **does not ends**.

What can happen if it is *not accepted*? What can happen if *it is not rejected*?

Terminology and language

Turing and his machine

Let M be a Turing machine:

- M accepts a word w if goes to an acceptance state when it reads w. In this case, M **finishes**.
- M rejects a word w if goes to a rejection state when it reads w. In this case, M **ends**.
- M goes to a loop with a word w when reading w it does not enter neither the acceptance or rejection state. In this case, M **does not ends**.

What can happen if it is *not accepted*? What can happen if *it is not rejected*?

Terminology and language

Turing and his machine

The language of a Turing machine M, defined as $L(M)$ o $\mathfrak{L}(M)$ is a set of all the words that M accepts:

$$\mathfrak{L}(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$$

A language is recognizable if and only if it is a language of some Turing machine.

Are there more languages?

Terminology and language

Turing and his machine

The language of a Turing machine M, defined as $L(M)$ o $\mathfrak{L}(M)$ is a set of all the words that M accepts:

$$\mathfrak{L}(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$$

A language is recognizable if and only if it is a language of some Turing machine.

Are there more languages?

Terminology and language

Turing and his machine

The language of a Turing machine M, defined as $L(M)$ o $\mathfrak{L}(M)$ is a set of all the words that M accepts:

$$\mathfrak{L}(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$$

A language is recognizable if and only if it is a language of some Turing machine.

Are there more languages?

Turing and the Entscheidungsproblem

Turing and his machine

In 1928, David Hilbert and Wilhelm Ackermann—both German mathematician-
they proposed a problem that took about 8 years to solve:

The Entscheidungsproblem

Is there any algorithm that takes as *input* a first order logic proposition, and say whether or not is universally valid from its axioms?

Turing and the Entscheidungsproblem

Turing and his machine

In 1928, David Hilbert and Wilhelm Ackermann—both German mathematician-
they proposed a problem that took about 8 years to solve:

The Entscheidungsproblem

Is there any algorithm that takes as *input* a first order logic proposition, and say whether or not is universally valid from its axioms?

Turing and the Entscheidungsproblem

Turing and his machine

In 1936, Alonzo Church—mathematician from United States—publishes the concept of "effective calculability" through lambda calculations. This names the lambda functions in programming languages.

In the same year, Alan Turing—British mathematician—publishes its work *On Computable Numbers, with an application to the Entscheidungsproblem*[1], where he proposes the the TM and delimits everything that can be "computable".

[1] Is *highly* probable that you would need to read the paper for your exam...

Turing and the Entscheidungsproblem

Turing and his machine

In 1936, Alonzo Church—mathematician from United States—publishes the concept of "effective calculability" through lambda calculations. This names the lambda functions in programming languages.

In the same year, Alan Turing—British mathematician—publishes its work *On Computable Numbers, with an application to the Entscheidungsproblem*[1], where he proposes the the TM and delimits everything that can be "computable".

[1] Is *highly* probable that you would need to read the paper for your exam...

Turing and the Entscheidungsproblem

Turing and his machine

In 1936, Alonzo Church—mathematician from United States—publishes the concept of "effective calculability" through lambda calculations. This names the lambda functions in programming languages.

In the same year, Alan Turing—British mathematician—publishes its work *On Computable Numbers, with an application to the Entscheidungsproblem*[1], where he proposes the the TM and delimits everything that can be "computable".

[1] Is *highly* probable that you would need to read the paper for your exam...

What can the Turing machine do?

Turing and his machine

Although two people gave the same answer to the *Entscheidungsproblem*—there is no algorithm to answer—usually we use Turing's version, it is more blunt in defining what an algorithm is:

Theorem 1

If can be done by a Turing machine, then there is an algorithm to do it.

This means that the TM can perform arithmetic operations, work with lists, accept words and perform **functions**.

What can the Turing machine do?

Turing and his machine

Although two people gave the same answer to the *Entscheidungsproblem*—there is no algorithm to answer—usually we use Turing's version, it is more blunt in defining what an algorithm is:

Theorem 1

If can be done by a Turing machine, then there is an algorithm to do it.

This means that the TM can perform arithmetic operations, work with lists, accept words and perform **functions**.

What can the Turing machine do?

Turing and his machine

Although two people gave the same answer to the *Entscheidungsproblem*—there is no algorithm to answer—usually we use Turing's version, it is more blunt in defining what an algorithm is:

Theorem 1

If can be done by a Turing machine, then there is an algorithm to do it.

This means that the TM can perform arithmetic operations, work with lists, accept words and perform **functions**.

What can the Turing machine do?

Turing and his machine

Although two people gave the same answer to the *Entscheidungsproblem*—there is no algorithm to answer—usually we use Turing's version, it is more blunt in defining what an algorithm is:

Theorem 1

If can be done by a Turing machine, then there is an algorithm to do it.

This means that the TM can perform arithmetic operations, work with lists, accept words and perform **functions**.

Formal definition

Formalities and examples

Again, there are many ways to define a formal definition. Thus, we will focus on using the closest thing to what we have used.

First, let us remember that a TM *to recognise words from a language* has

- A set of states.
- an input.
- A tape used as memory.
- An initial state.
- An acceptance state.
- A rejecting state.
- A transition function that defines how to pass from one state to another, given certain conditions.

Formal definition

Formalities and examples

Again, there are many ways to define a formal definition. Thus, we will focus on using the closest thing to what we have used.

First, let us remember that a TM *to recognise words from a language* has

- A set of states.
- an input.
- A tape used as memory.
- An initial state.
- An acceptance state.
- A rejecting state.
- A transition function that defines how to pass from one state to another, given certain conditions.

Formal definition

Formalities and examples

Again, there are many ways to define a formal definition. Thus, we will focus on using the closest thing to what we have used.

First, let us remember that a TM *to recognise words from a language* has

- A set of states.
- an input.
- A tape used as memory.
- An initial state.
- An acceptance state.
- A rejecting state.
- A transition function that defines how to pass from one state to another, given certain conditions.

Formal definition

Formalities and examples

Again, there are many ways to define a formal definition. Thus, we will focus on using the closest thing to what we have used.

First, let us remember that a TM *to recognise words from a language* has

- A set of states.
- an input.
- A tape used as memory.
- An initial state.
- An acceptance state.
- A rejecting state.
- A transition function that defines how to pass from one state to another, given certain conditions.

Formal definition

Formalities and examples

Again, there are many ways to define a formal definition. Thus, we will focus on using the closest thing to what we have used.

First, let us remember that a TM *to recognise words from a language* has

- A set of states.
- an input.
- A tape used as memory.
- An initial state.
- An acceptance state.
- A rejecting state.
- A transition function that defines how to pass from one state to another, given certain conditions.

Formal definition

Formalities and examples

Again, there are many ways to define a formal definition. Thus, we will focus on using the closest thing to what we have used.

First, let us remember that a TM *to recognise words from a language* has

- A set of states.
- an input.
- A tape used as memory.
- An initial state.
- An acceptance state.
- A rejecting state.
- A transition function that defines how to pass from one state to another, given certain conditions.

Formal definition

Formalities and examples

Again, there are many ways to define a formal definition. Thus, we will focus on using the closest thing to what we have used.

First, let us remember that a TM *to recognise words from a language* has

- A set of states.
- an input.
- A tape used as memory.
- An initial state.
- An acceptance state.
- A rejecting state.
- A transition function that defines how to pass from one state to another, given certain conditions.

Formal definition

Formalities and examples

Again, there are many ways to define a formal definition. Thus, we will focus on using the closest thing to what we have used.

First, let us remember that a TM *to recognise words from a language* has

- A set of states.
- an input.
- A tape used as memory.
- An initial state.
- An acceptance state.
- A rejecting state.
- A transition function that defines how to pass from one state to another, given certain conditions.

Formal definition

Formalities and examples

Formal definition of a Turing machine

A Turing machine M to recognise words from a language is a tuple with the form $M = (Q, \Sigma, \Gamma, \delta, q, F)$ where:

- Q is a finite set of **states**,
- Σ is the **input alphabet**, where $\square \notin \Sigma$,
- Γ is the **tape alphabet**, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$,
- $q \in Q$ is the **initial state**,
- $F \subseteq Q$ is the set of **accepting states**,
- δ is the transition function.

Formal definition

Formalities and examples

Formal definition of a Turing machine

A Turing machine M to recognise words from a language is a tuple with the form $M = (Q, \Sigma, \Gamma, \delta, q, F)$ where:

- Q is a finite set of **states**,
- Σ is the **input alphabet**, where $\square \notin \Sigma$,
- Γ is the **tape alphabet**, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$,
- $q \in Q$ is the **initial state**,
- $F \subseteq Q$ is the set of **accepting states**,
- δ is the transition function.

Formal definition

Formalities and examples

Formal definition of a Turing machine

A Turing machine M to recognise words from a language is a tuple with the form $M = (Q, \Sigma, \Gamma, \delta, q, F)$ where:

- Q is a finite set of **states**,
- Σ is the **input alphabet**, where $\square \notin \Sigma$,
- Γ is the **tape alphabet**, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$,
- $q \in Q$ is the **initial state**,
- $F \subseteq Q$ is the set of **accepting states**,
- δ is the transition function.

Formal definition

Formalities and examples

Formal definition of a Turing machine

A Turing machine M to recognise words from a language is a tuple with the form $M = (Q, \Sigma, \Gamma, \delta, q, F)$ where:

- Q is a finite set of **states**,
- Σ is the **input alphabet**, where $\square \notin \Sigma$,
- Γ is the **tape alphabet**, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$,
- $q \in Q$ is the **initial state**,
- $F \subseteq Q$ is the set of **accepting states**,
- δ is the transition function.

Formal definition

Formalities and examples

Formal definition of a Turing machine

A Turing machine M to recognise words from a language is a tuple with the form $M = (Q, \Sigma, \Gamma, \delta, q, F)$ where:

- Q is a finite set of **states**,
- Σ is the **input alphabet**, where $\square \notin \Sigma$,
- Γ is the **tape alphabet**, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$,
- $q \in Q$ is the **initial state**,
- $F \subseteq Q$ is the set of **accepting states**,
- δ is the transition function.

Formal definition

Formalities and examples

Formal definition of a Turing machine

A Turing machine M to recognise words from a language is a tuple with the form $M = (Q, \Sigma, \Gamma, \delta, q, F)$ where:

- Q is a finite set of **states**,
- Σ is the **input alphabet**, where $\square \notin \Sigma$,
- Γ is the **tape alphabet**, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$,
- $q \in Q$ is the **initial state**,
- $F \subseteq Q$ is the set of **accepting states**,
- δ is the transition function.

Formal definition

Formalities and examples

Formal definition of a Turing machine

A Turing machine M to recognise words from a language is a tuple with the form $M = (Q, \Sigma, \Gamma, \delta, q, F)$ where:

- Q is a finite set of **states**,
- Σ is the **input alphabet**, where $\square \notin \Sigma$,
- Γ is the **tape alphabet**, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$,
- $q \in Q$ is the **initial state**,
- $F \subseteq Q$ is the set of **accepting states**,
- δ is the transition function.

Formal definition

Formalities and examples

Some authors define the following special cases for the states:

- **Rejecting state:** Define a specific state for rejection.
- **Accepting state:** Define a unique accepting state.

The outputs *accept* and *reject* are obtained by entering designated accepting and rejecting states. If it doesn't enter an accepting or a rejecting state, it will go on forever, never halting.

Formal definition

Formalities and examples

The transition function δ is a function with the form:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

Example

We can write $q_0 1 \to q_1 1 R$ that means:

- From the state q_0,
- and when reading a 1 in the tape

Then the TM

- change to the state q_1,
- writes a 1 in the current cell, and
- moves the tape head to the right (Right)

Formal definition

Formalities and examples

The transition function δ is a function with the form:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

Example

We can write $q_0 1 \to q_1 1 R$ that means:

- From the state q_0,
- and when reading a 1 in the tape

Then the TM

- change to the state q_1,
- writes a 1 in the current cell, and
- moves the tape head to the right (Right)

Formal definition

Formalities and examples

The transition function δ is a function with the form:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

Example

We can write $q_0 1 \to q_1 1 R$ that means:

- From the state q_0,
- and when reading a 1 in the tape

Then the TM

- change to the state q_1,
- writes a 1 in the current cell, and
- moves the tape head to the right (Right)

Formal definition

Formalities and examples

The transition function δ is a function with the form:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

Example

We can write $q_0 1 \to q_1 1 R$ that means:

- From the state q_0,
- and when reading a 1 in the tape

Then the TM

- change to the state q_1,
- writes a 1 in the current cell, and
- moves the tape head to the right (Right)

Formal definition

Formalities and examples

The transition function δ is a function with the form:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

Example

We can write $q_0 1 \to q_1 1 R$ that means:

- From the state q_0,
- and when reading a 1 in the tape

Then the TM

- change to the state q_1,
- writes a 1 in the current cell, and
- moves the tape head to the right (Right)

Formal definition

Formalities and examples

The transition function δ is a function with the form:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

Example

We can write $q_0 1 \to q_1 1 R$ that means:

- From the state q_0,
- and when reading a 1 in the tape

Then the TM

- change to the state q_1,
- writes a 1 in the current cell, and
- moves the tape head to the right (Right)

Formal definition

Formalities and examples

The transition function δ is a function with the form:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

Example

We can write $q_0 1 \to q_1 1 R$ that means:

- From the state q_0,
- and when reading a 1 in the tape

Then the TM

- change to the state q_1,
- writes a 1 in the current cell, and
- moves the tape head to the right (Right)

Formal definition

Formalities and examples

The truth is that...

...it is easier to think about it as **arrows** that go **from one state** $q0$ **to another state** q_1 with the form $x \to y, D$— if reads x, writes y and move towards D:

Formal definition

Formalities and examples

The truth is that...

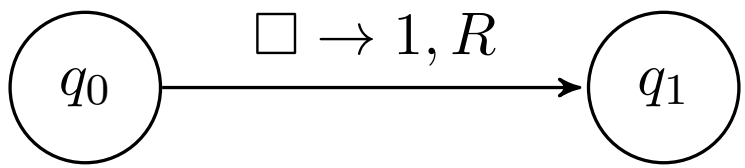
...it is easier to think about it as **arrows** that go **from one state** $q0$ **to another state** q_1 with the form $x \to y, D$— if reads x, writes y and move towards D:

Formal definition

Formalities and examples

The truth is that...

...it is easier to think about it as **arrows** that go **from one state** $q0$ **to another state** q_1 with the form $x \to y, D$— if reads x, writes y and move towards D:



Start configuration

Formalities and examples

Before we start computing, the Turing machine must be in a specific configuration. In this configuration:

1. The tape is empty, this is, it only has $\square$ symbols in it.
2. The input word **is copied** to some place of the tape.
3. The tape head moves to the **beginning** of the input word.
4. Then, it starts computing.

Start configuration

Formalities and examples

Before we start computing, the Turing machine must be in a specific configuration. In this configuration:

1. The tape is empty, this is, it only has $\square$ symbols in it.
2. The input word **is copied** to some place of the tape.
3. The tape head moves to the **beginning** of the input word.
4. Then, it starts computing.

Start configuration

Formalities and examples

Before we start computing, the Turing machine must be in a specific configuration. In this configuration:

1. The tape is empty, this is, it only has $\square$ symbols in it.
2. The input word **is copied** to some place of the tape.
3. The tape head moves to the **beginning** of the input word.
4. Then, it starts computing.

Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

Let us remember the conditions for $\{0^n 1^n\} \mid n \geq 1$.

- **Accepting conditions:**
  - It is a concatenation of 1 or more zeroes, followed by a string of ones with the same length as the zeroes.
  - The order is important. Only zeroes at the beginning followed by a string of ones.

- **Rejecting conditions:**
  - If a zero is found after a one.
  - If the string of ones has a different length than the length of the zeroes.

The **accepting** and **rejecting** states take immediate effect in the TM.

Example for $\{0^n1^n\} \mid n \geq 1$

Formalities and examples

Let us remember the conditions for $\{0^n1^n\} \mid n \geq 1$.

- **Accepting conditions:**
  - It is a concatenation of 1 or more zeroes, followed by a string of ones with the same length as the zeroes.
  - The order is important. Only zeroes at the beginning followed by a string of ones.

- **Rejecting conditions:**
  - If a zero is found after a one.
  - If the string of ones has a different length than the length of the zeroes.

The **accepting** and **rejecting** states take immediate effect in the TM.

Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

Now let us focus just on the tape of the TM.

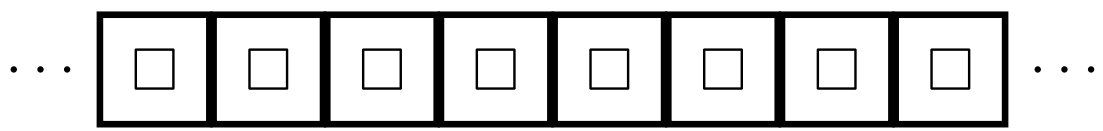
1. The tape is empty, this is, it only has $\square$ symbols in it.
2. The input word **is copied** to some place of the tape.
3. The tape head moves to the **beginning** of the input word.
4. Then, it starts computing.

Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

Now let us focus just on the tape of the TM.

1. The tape is empty, this is, it only has $\square$ symbols in it.
2. The input word **is copied** to some place of the tape.
3. The tape head moves to the **beginning** of the input word.
4. Then, it starts computing.

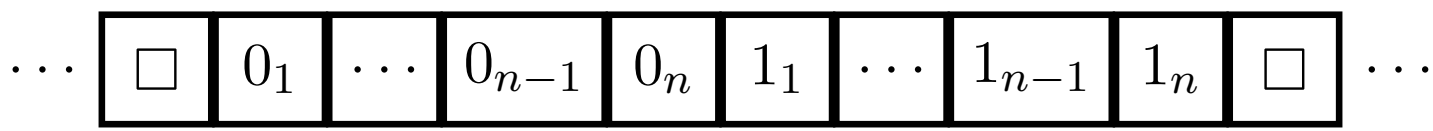


Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

Now let us focus just on the tape of the TM.

1. The tape is empty, this is, it only has $\square$ symbols in it.
2. The input word **is copied** to some place of the tape.
3. The tape head moves to the **beginning** of the input word.
4. Then, it starts computing.

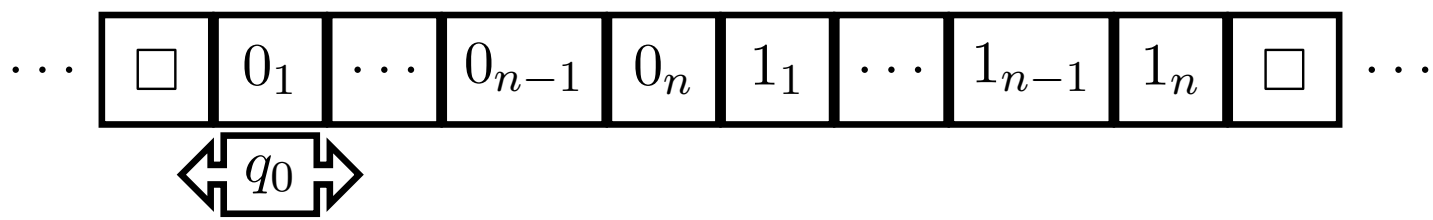


Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

Now let us focus just on the tape of the TM.

1. The tape is empty, this is, it only has $\square$ symbols in it.
2. The input word **is copied** to some place of the tape.
3. The tape head moves to the **beginning** of the input word.
4. Then, it starts computing.



Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

Now let us focus just on the tape of the TM.

1. The tape is empty, this is, it only has $\square$ symbols in it.
2. The input word **is copied** to some place of the tape.
3. The tape head moves to the **beginning** of the input word.
4. Then, it starts computing.

Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

One way to process a string with the form $\{0^n 1^n\}$ is by following the general idea:

- For every 0 found, change it by another symbol in Γ (e.g. X).
- For every 1 found, change it by another symbol in Γ (e.g. Y).

Keep doing this until all 0s and 1s have been changed.

Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

More precisely

1. Starting from the left side (beginning) of the word.
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.
4. Search for a 0 intermediately to the right and, if it is there, repeat the process from step 2.
5. Keep doing this until all 0s and 1s have been changed.

Ending conditions:

- If the input does not have the form 0^*1^*, then the TM won't be able to perform the next action and it will stop without accepting.
- If it ends changing all 0s for Xs in the same iteration in which changes the last 1 for a Y, then the input with the form $0^n 1^n$ will be accepted.

Example for $\{0^n1^n\} \mid n \geq 1$

Formalities and examples

More precisely

1. Starting from the left side (beginning) of the word.
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.
4. Search for a 0 intermediately to the right and, if it is there, repeat the process from step 2.
5. Keep doing this until all 0s and 1s have been changed.

Ending conditions:

- If the input does not have the form 0^*1^*, then the TM won't be able to perform the next action and it will stop without accepting.
- If it ends changing all 0s for Xs in the same iteration in which changes the last 1 for a Y, then the input with the form 0^n1^n will be accepted.

Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

More precisely

1. Starting from the left side (beginning) of the word.
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.
4. Search for a 0 intermediately to the right and, if it is there, repeat the process from step 2.
5. Keep doing this until all 0s and 1s have been changed.

Ending conditions:

- If the input does not have the form 0^*1^*, then the TM won't be able to perform the next action and it will stop without accepting.
- If it ends changing all 0s for Xs in the same iteration in which changes the last 1 for a Y, then the input with the form $0^n 1^n$ will be accepted.

Example for $\{0^n1^n\} \mid n \geq 1$

Formalities and examples

More precisely

1. Starting from the left side (beginning) of the word.
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.
4. Search for a 0 intermediately to the right and, if it is there, repeat the process from step 2.
5. Keep doing this until all 0s and 1s have been changed.

Ending conditions:

- If the input does not have the form 0^*1^*, then the TM won't be able to perform the next action and it will stop without accepting.
- If it ends changing all 0s for Xs in the same iteration in which changes the last 1 for a Y, then the input with the form 0^n1^n will be accepted.

Example for $\{0^n1^n\} \mid n \geq 1$

Formalities and examples

More precisely

1. Starting from the left side (beginning) of the word.
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.
4. Search for a 0 intermediately to the right and, if it is there, repeat the process from step 2.
5. Keep doing this until all 0s and 1s have been changed.

Ending conditions:

- If the input does not have the form 0^*1^*, then the TM won't be able to perform the next action and it will stop without accepting.
- If it ends changing all 0s for Xs in the same iteration in which changes the last 1 for a Y, then the input with the form 0^n1^n will be accepted.

Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

More precisely

1. Starting from the left side (beginning) of the word.
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.
4. Search for a 0 intermediately to the right and, if it is there, repeat the process from step 2.
5. Keep doing this until all 0s and 1s have been changed.

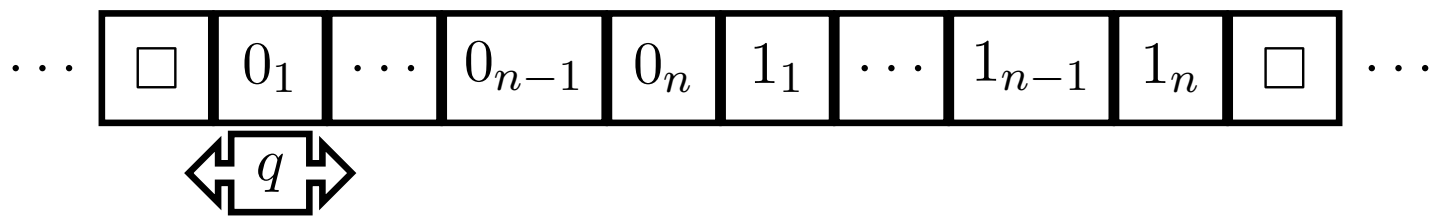
Ending conditions:

- If the input does not have the form 0^*1^*, then the TM won't be able to perform the next action and it will stop without accepting.
- If it ends changing all 0s for Xs in the same iteration in which changes the last 1 for a Y, then the input with the form $0^n 1^n$ will be accepted.

Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

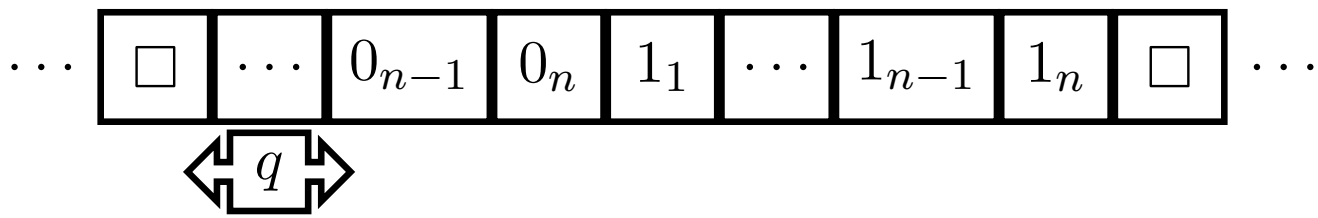
1. Starting from the left side (beginning) of the word.



Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

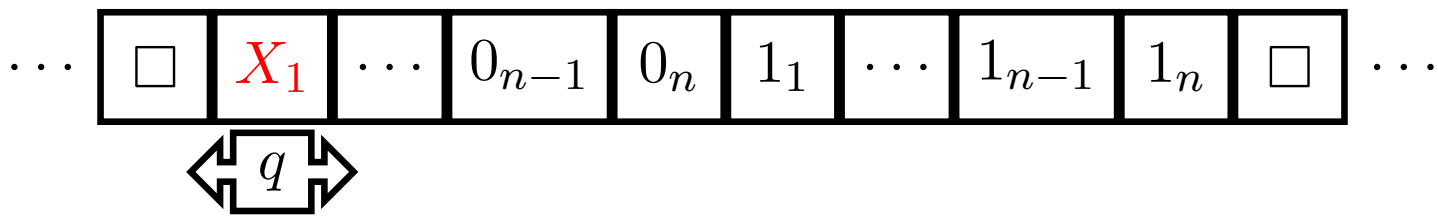
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

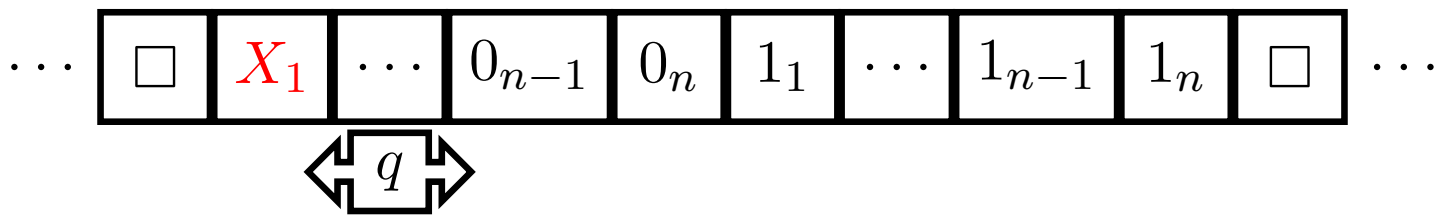
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

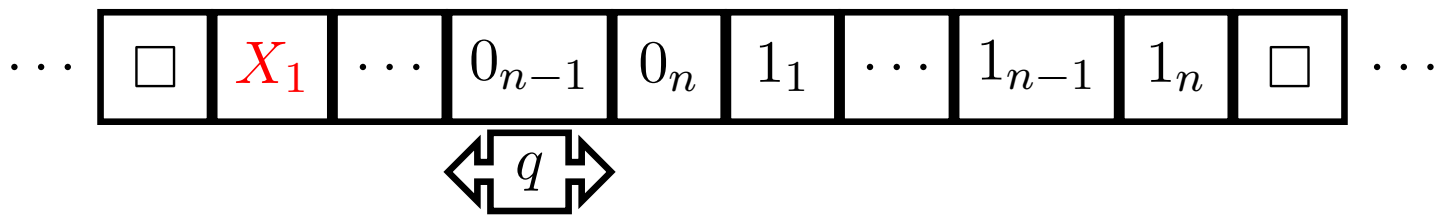
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

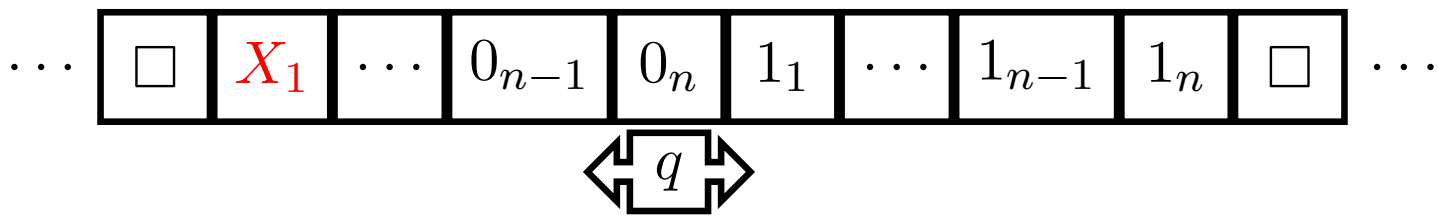
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

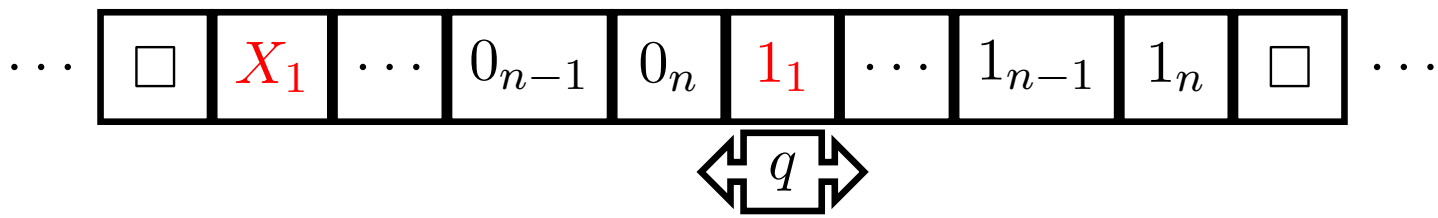
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

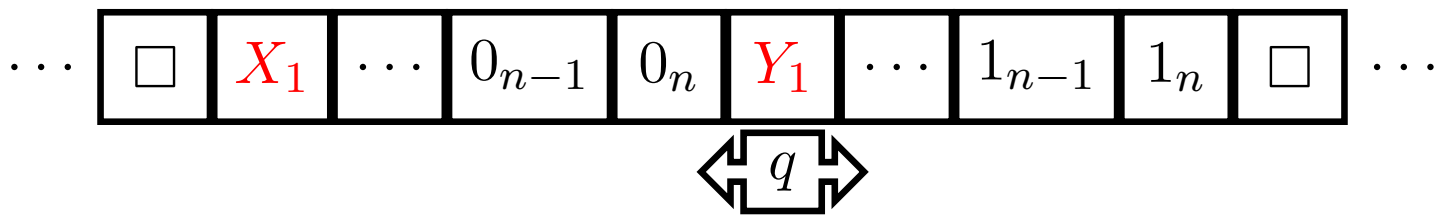
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

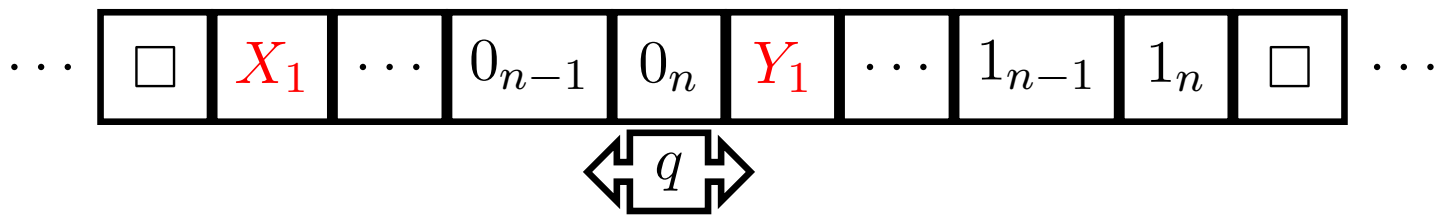
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

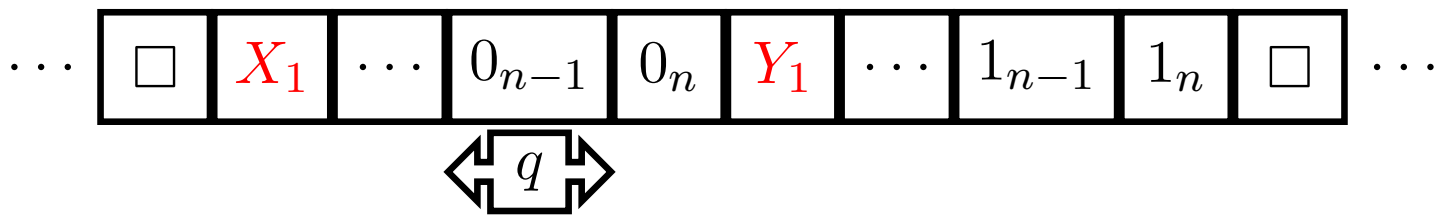
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

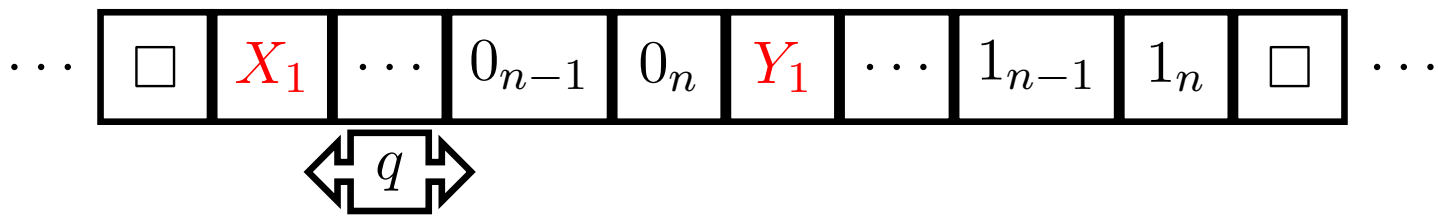
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

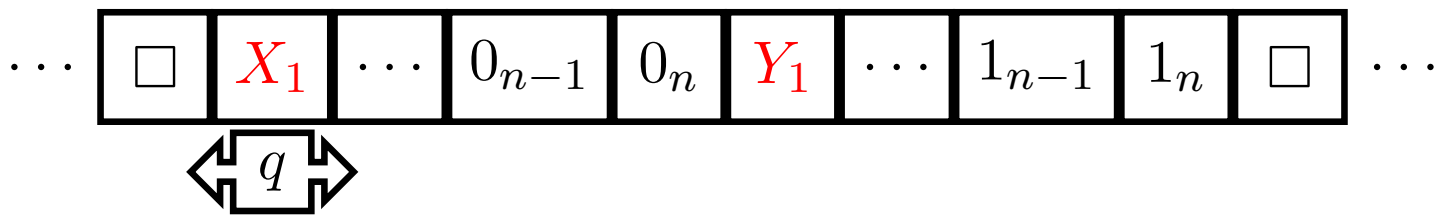
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

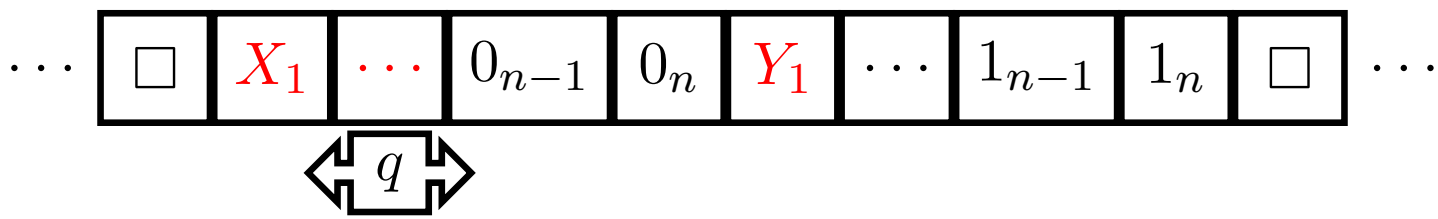
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

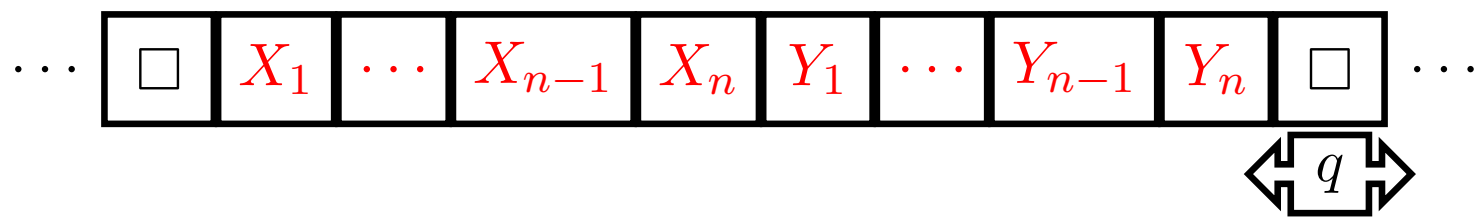
4. Search for a 0 intermediately to the right and, if it is there, repeat the process from step 2.



Example for $\{0^n 1^n\} \mid n \geq 1$

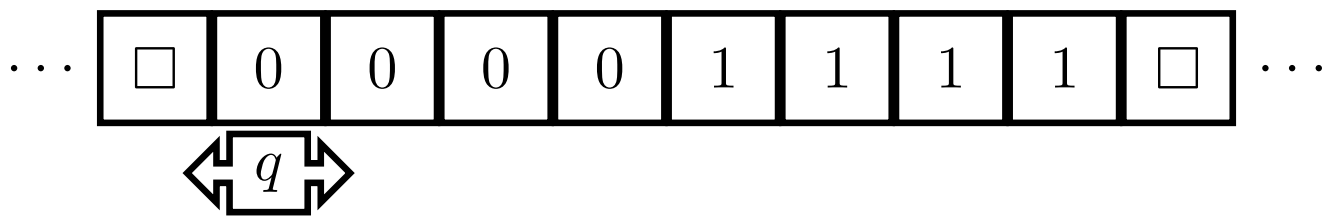
Formalities and examples

5. Keep doing this until all 0s and 1s have been changed.



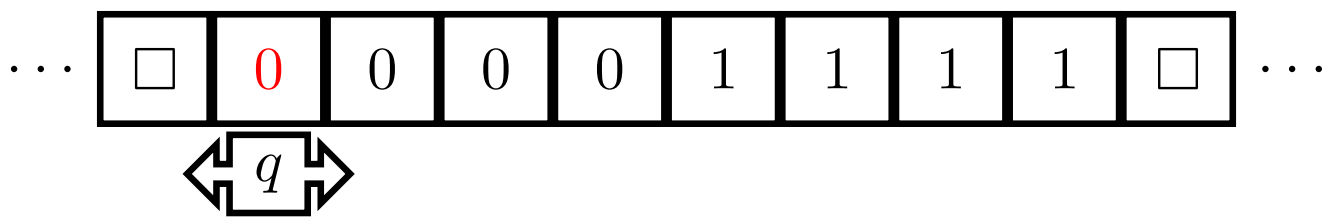
Example for 00001111

1. Starting from the left side (beginning) of the word.



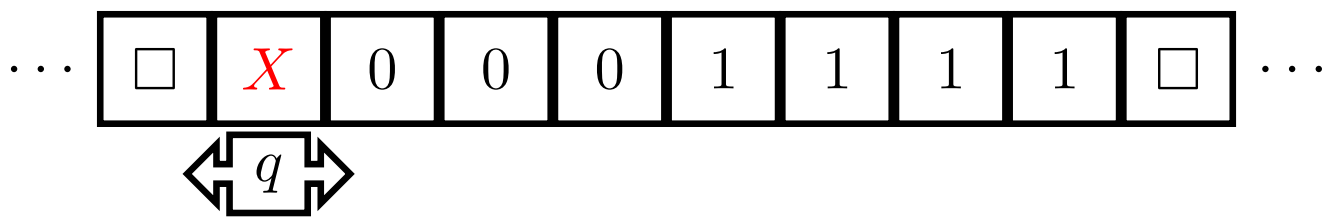
Example for 00001111

2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



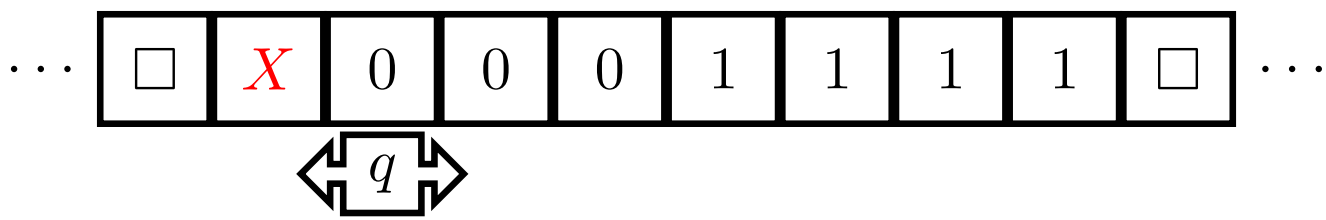
Example for 00001111

2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for 00001111

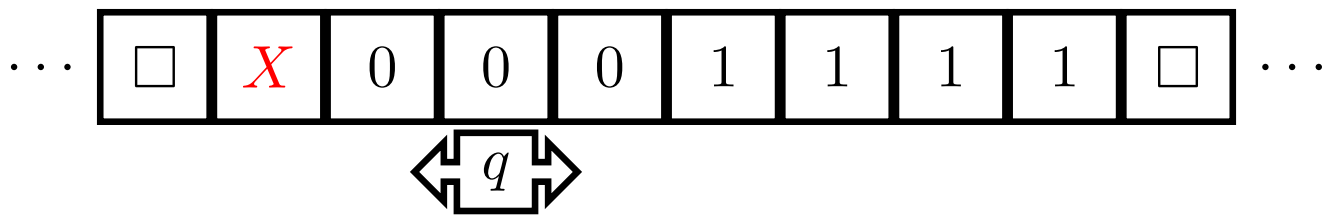
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for 00001111

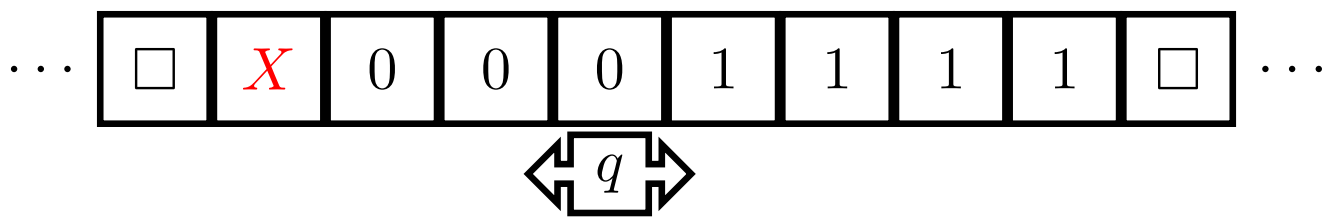
Formalities and examples

2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for 00001111

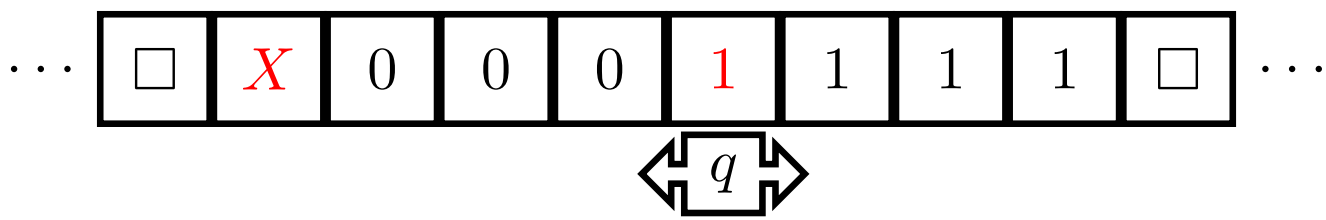
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for 00001111

Formalities and examples

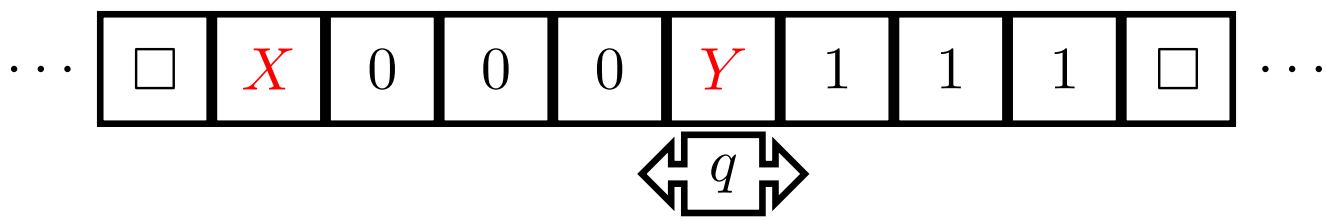
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for 00001111

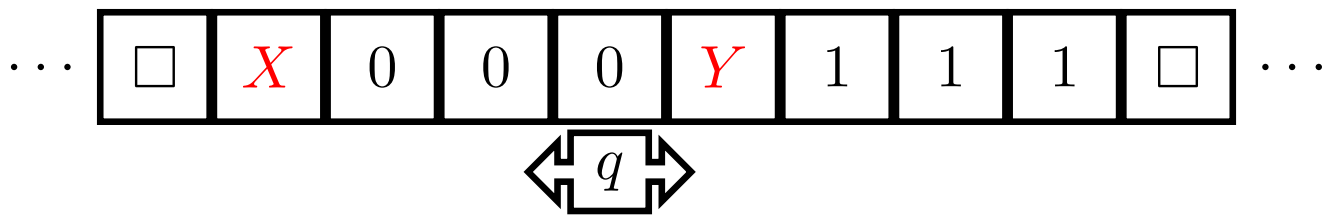
Formalities and examples

3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

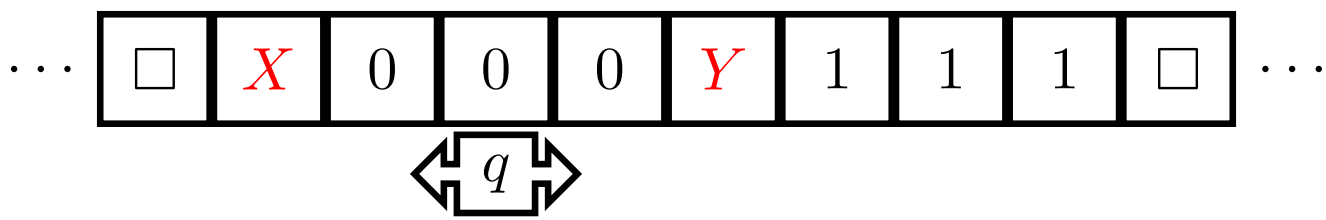
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

Formalities and examples

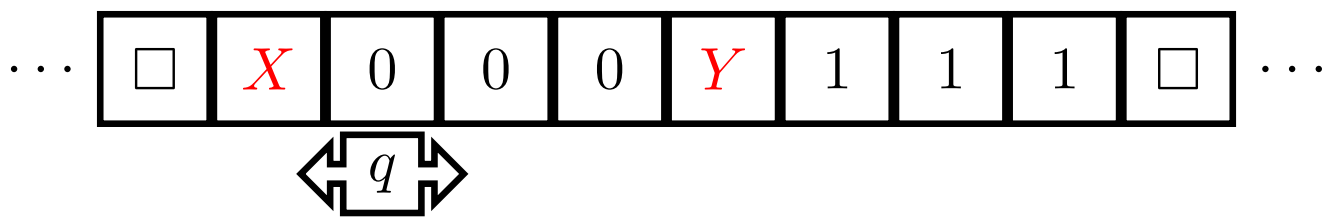
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

Formalities and examples

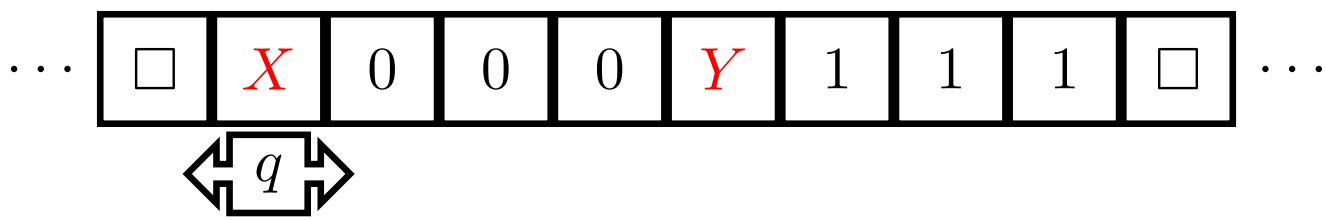
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

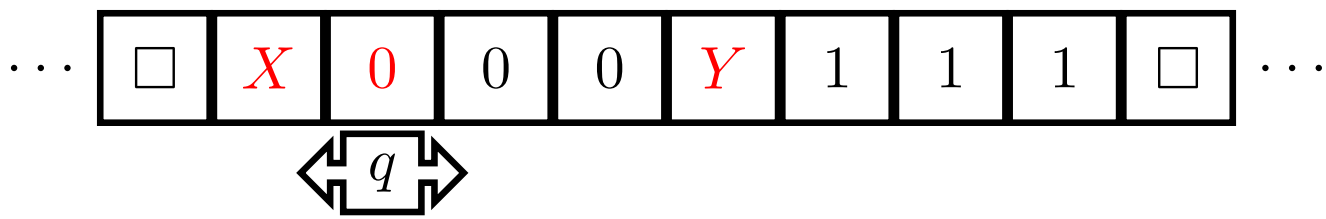
Formalities and examples

3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



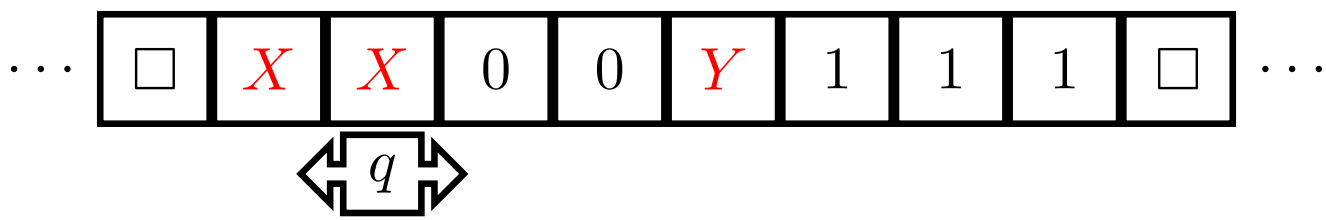
Example for 00001111

4. Search for a 0 intermediately to the right and, if it is there, repeat the process from step 2.



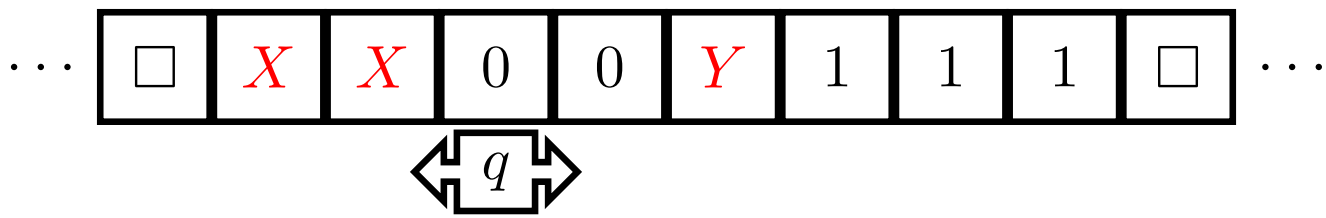
Example for 00001111

2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



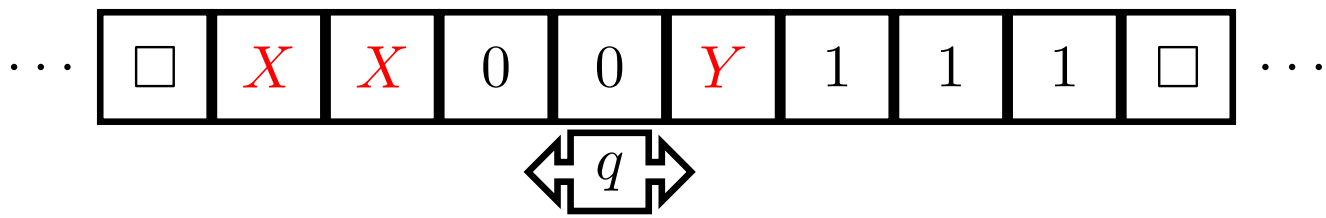
Example for 00001111

2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



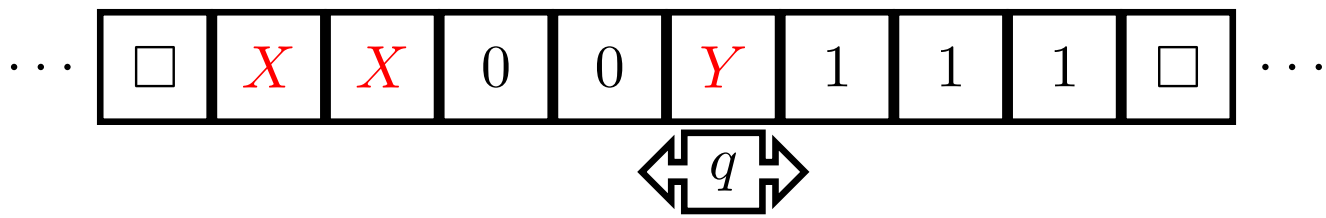
Example for 00001111

2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for 00001111

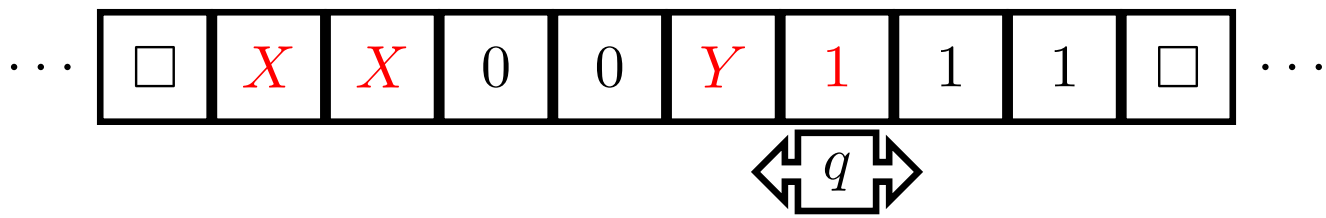
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for 00001111

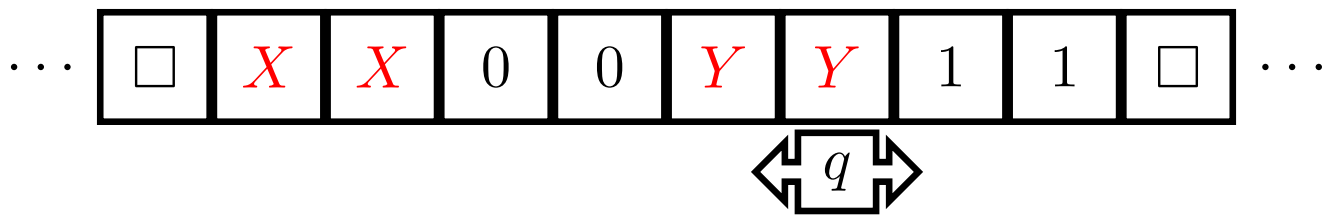
Formalities and examples

2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for 00001111

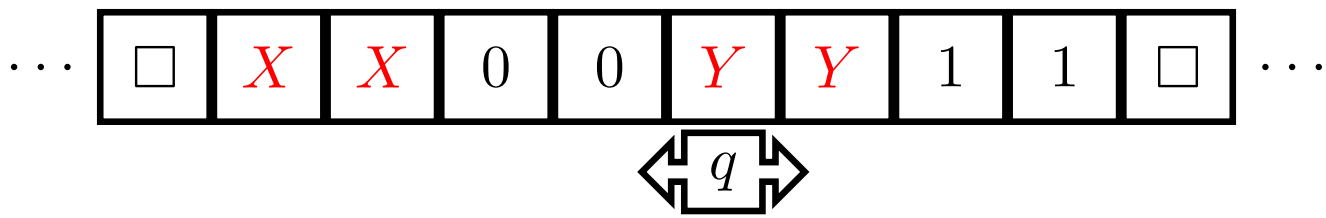
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

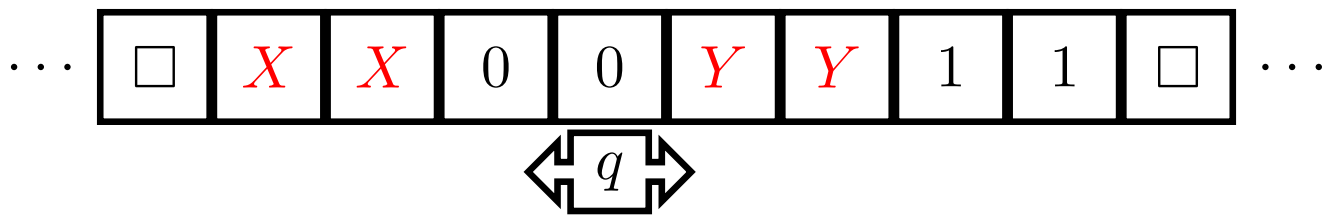
Formalities and examples

3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



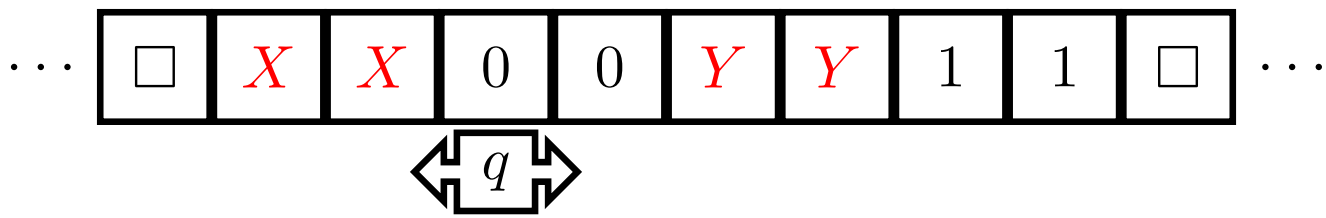
Example for 00001111

3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

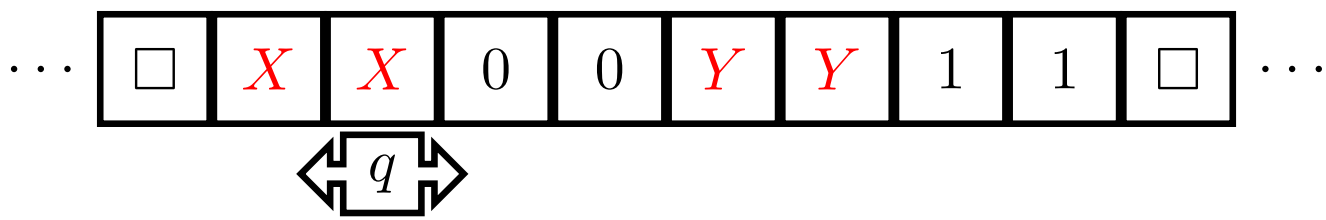
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

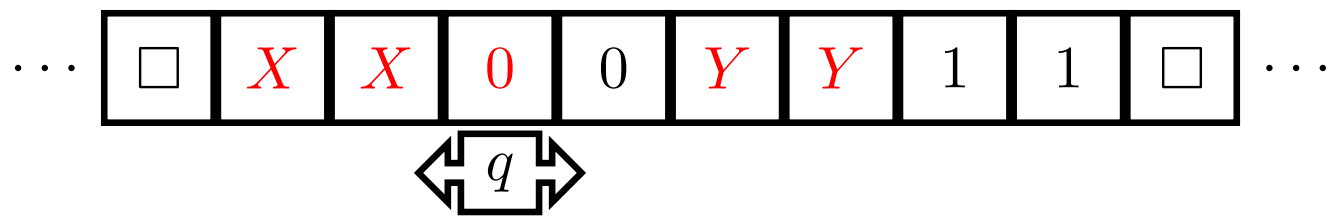
Formalities and examples

3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

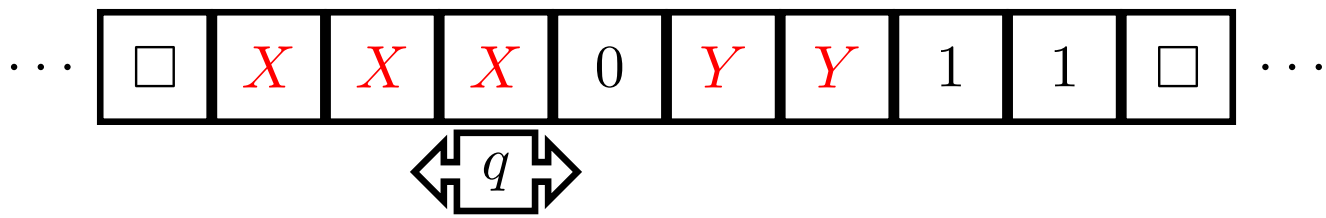
4. Search for a 0 intermediately to the right and, if it is there, repeat the process from step 2.



Example for 00001111

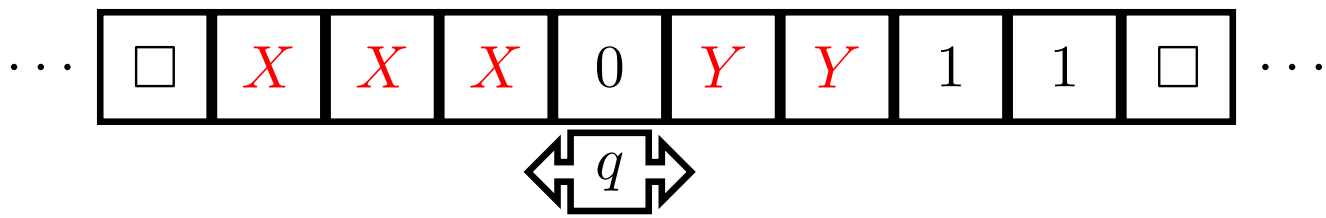
Formalities and examples

2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



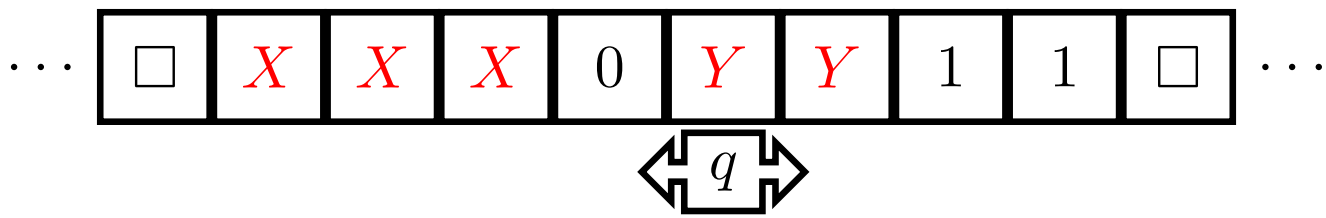
Example for 00001111

2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



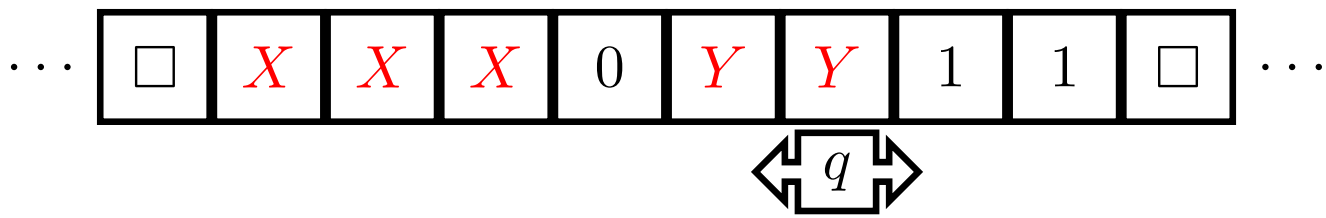
Example for 00001111

2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for 00001111

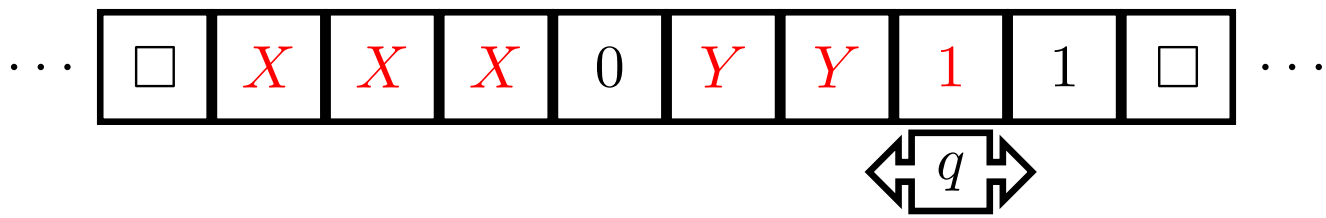
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for 00001111

Formalities and examples

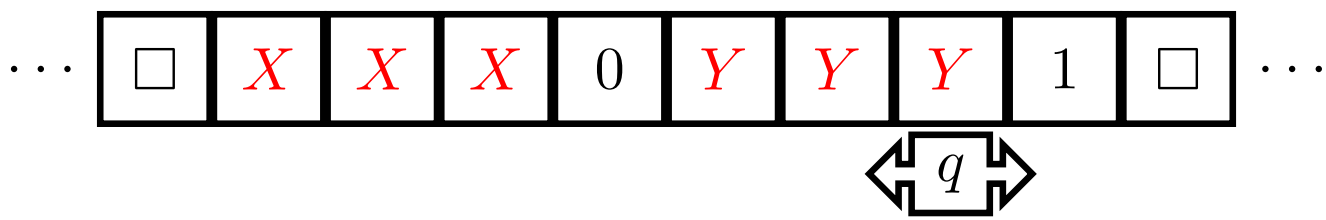
2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for 00001111

Formalities and examples

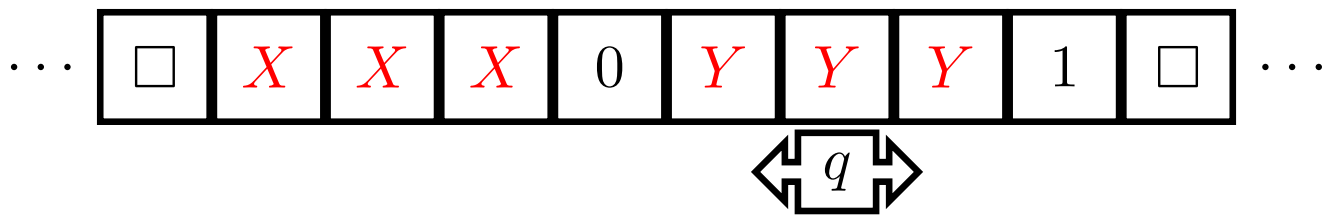
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

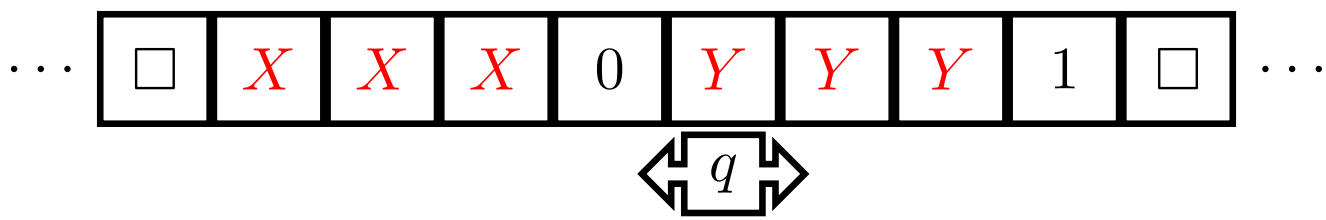
Formalities and examples

3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

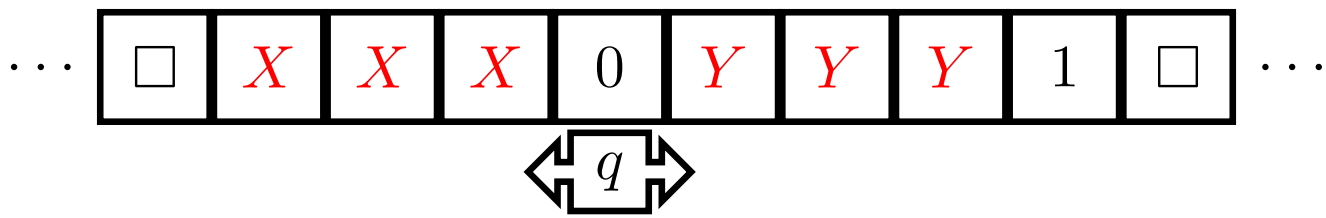
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

Formalities and examples

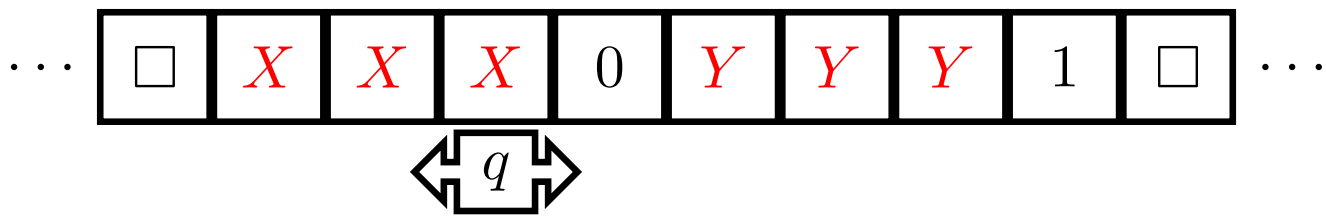
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

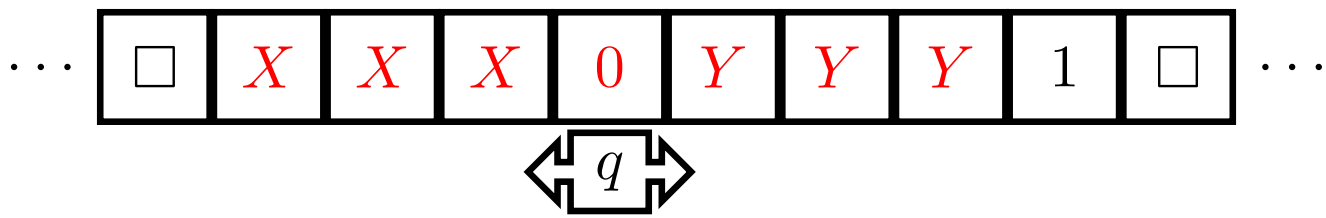
Formalities and examples

3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



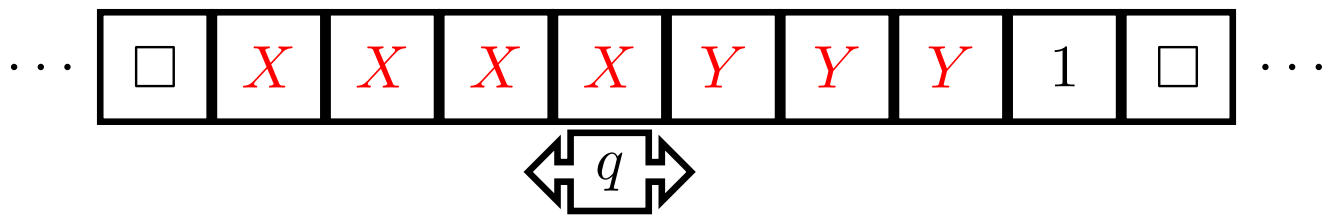
Example for 00001111

4. Search for a 0 intermediately to the right and, if it is there, repeat the process from step 2.



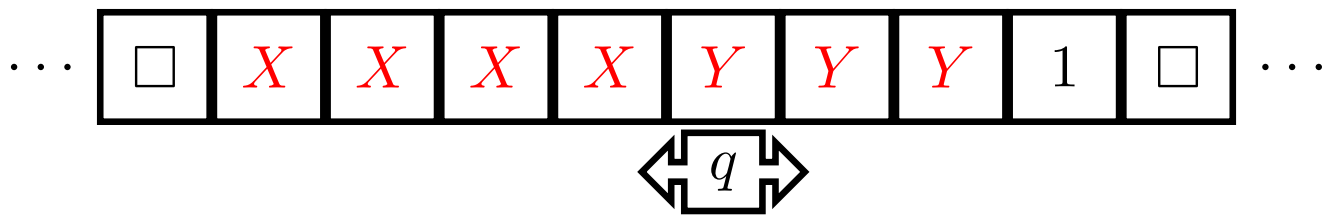
Example for 00001111

2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



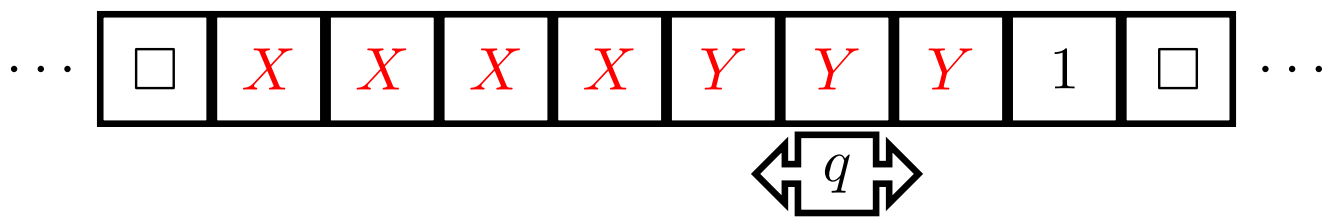
Example for 00001111

2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



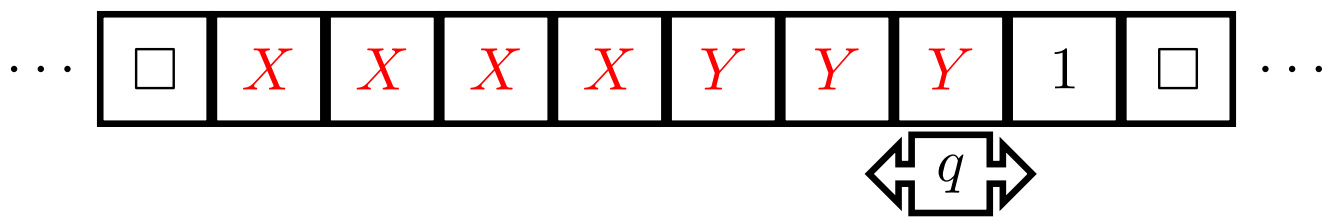
Example for 00001111

2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



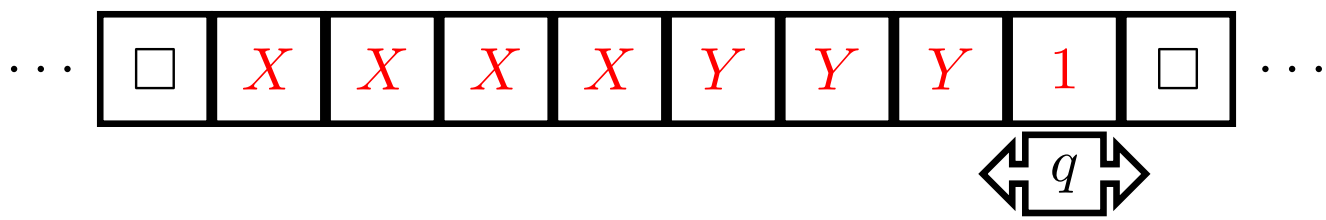
Example for 00001111

2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



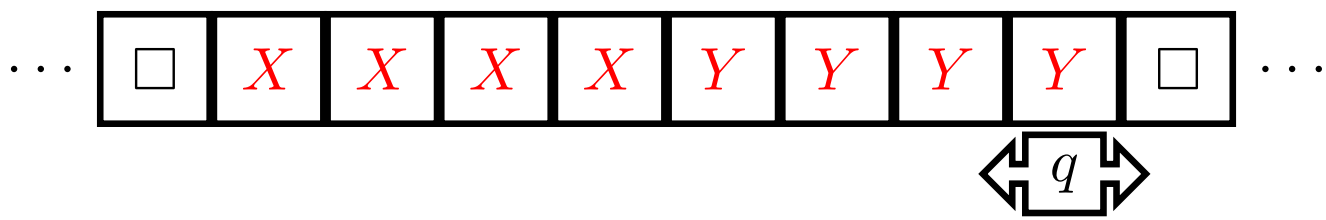
Example for 00001111

2. Change the 0 for an X, then move to the right side of the word (ignoring all 0s, Xs and possible Ys) until a 1 is found.



Example for 00001111

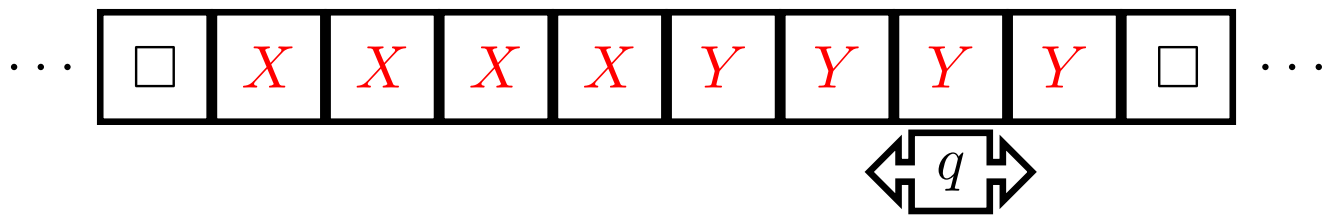
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

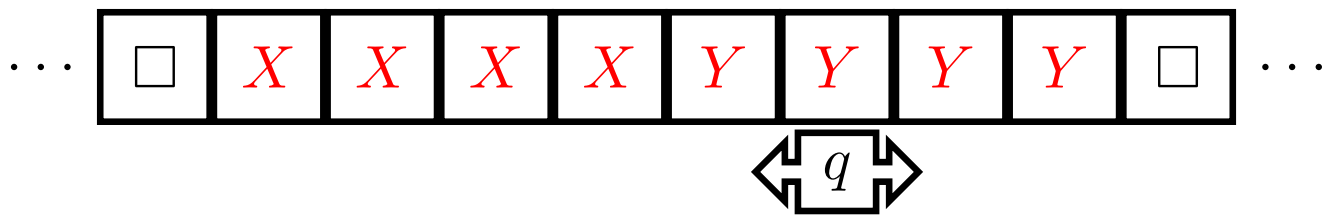
Formalities and examples

3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

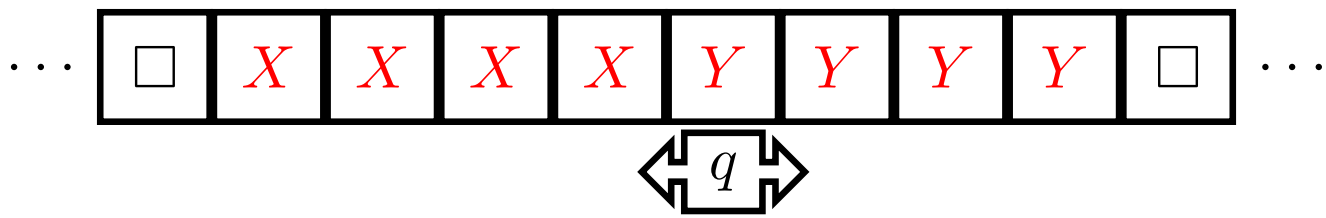
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

Formalities and examples

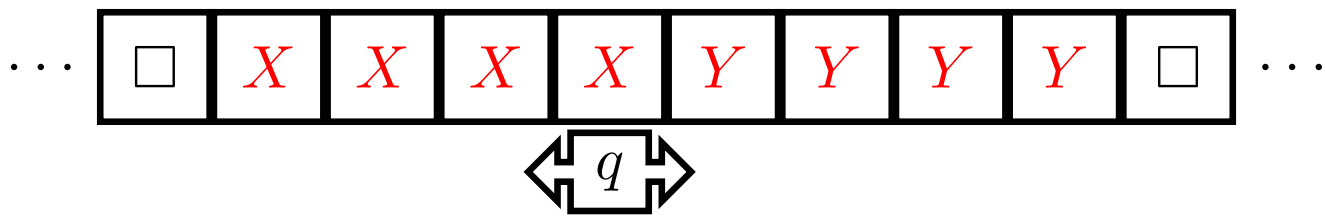
3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



Example for 00001111

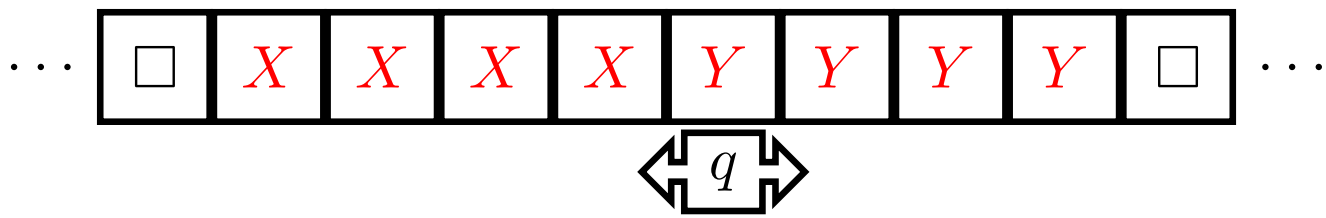
Formalities and examples

3. Change the 1 for a Y, then move to the left side of the word (ignoring all Ys and possible 0s) until an X is found.



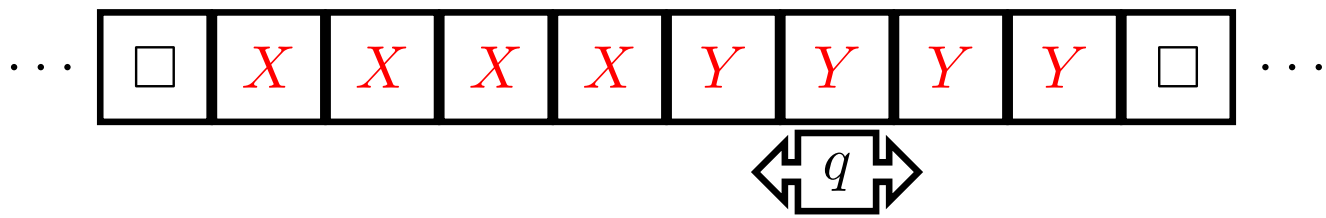
Example for 00001111

4. Search for a 0 intermediately to the right and, if it is there, repeat the process from step 2.



Example for 00001111

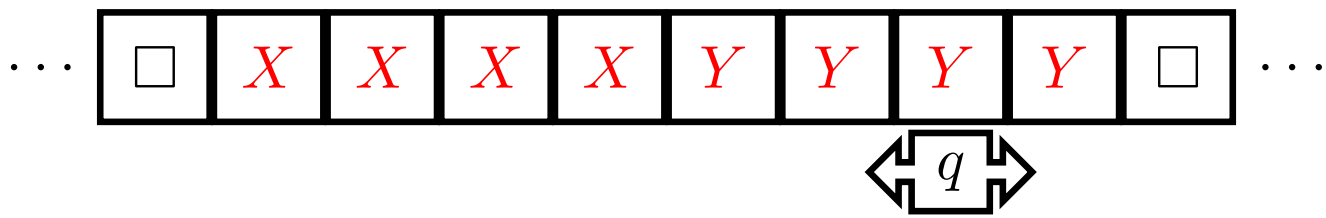
5. Keep doing this until all 0s and 1s have been changed.



Example for 00001111

Formalities and examples

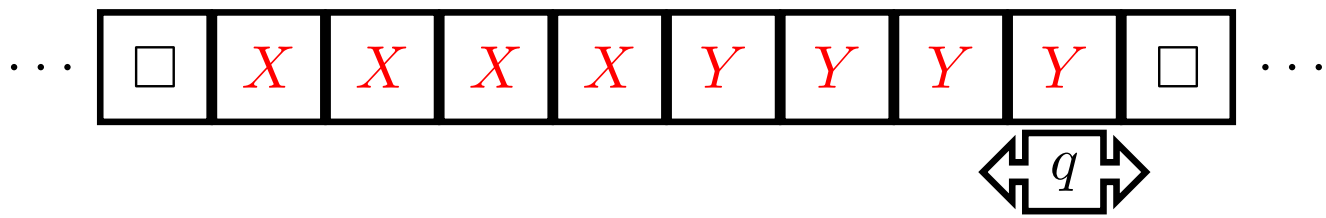
5. Keep doing this until all 0s and 1s have been changed.



Example for 00001111

Formalities and examples

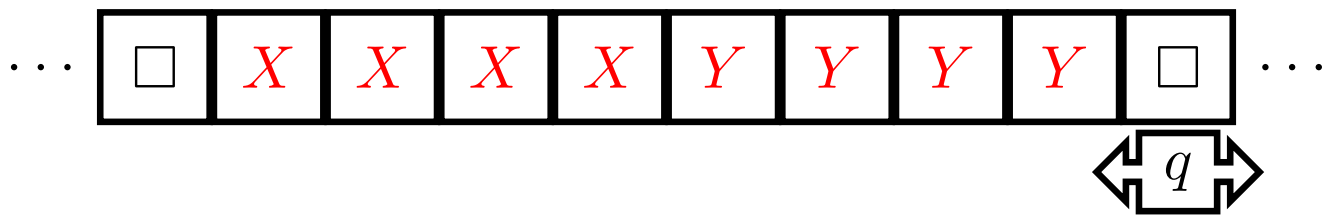
5. Keep doing this until all 0s and 1s have been changed.



Example for 00001111

Formalities and examples

5. Keep doing this until all 0s and 1s have been changed.



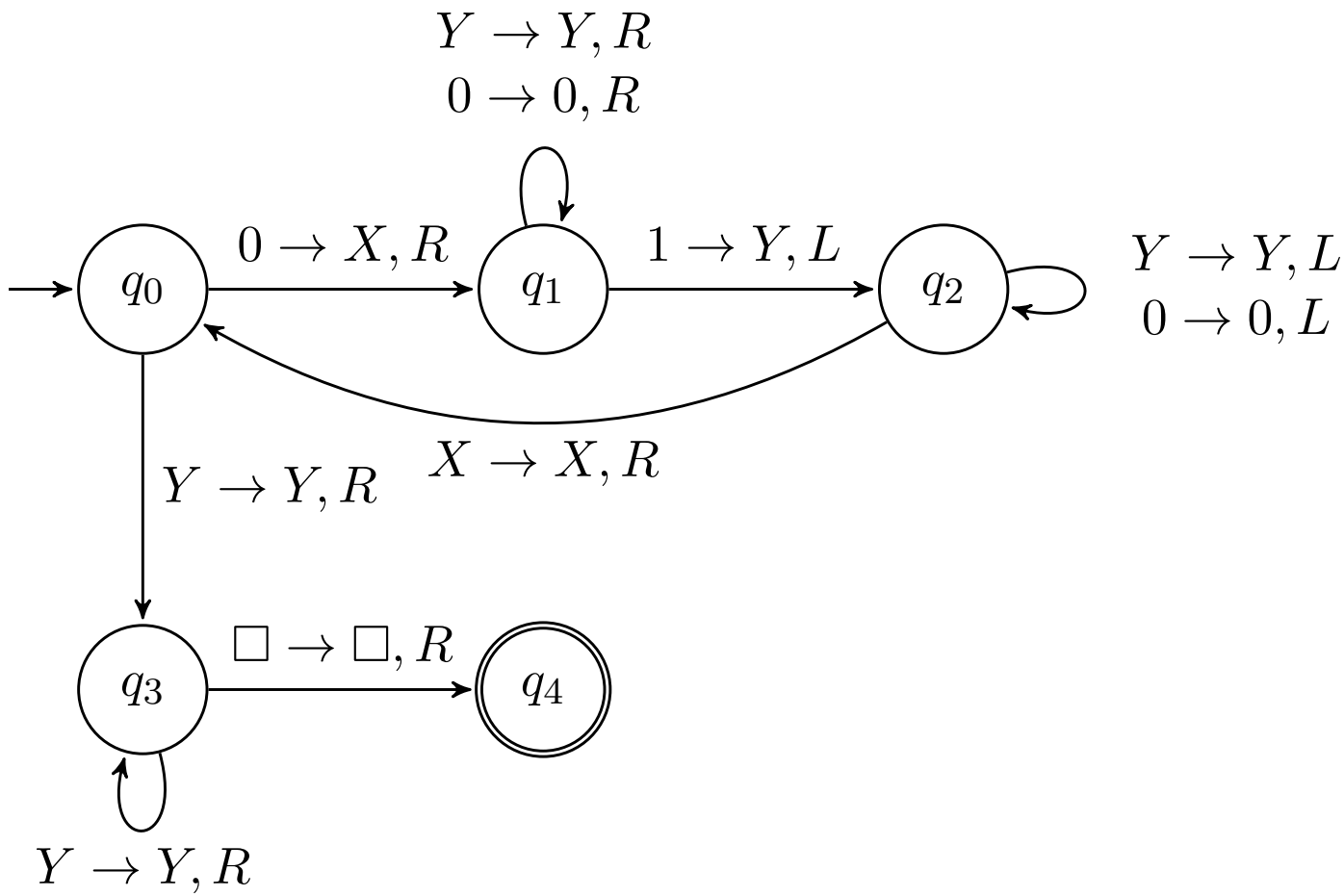
Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

How can we represent a Turing machine using a transition diagram?

Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples



Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

How can we formally represent a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q, F)$?

Example for $\{0^n 1^n\} \mid n \geq 1$

Formalities and examples

How can we formally represent a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q, F)$?

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$,
- $\Sigma = \{0, 1\}$,
- $\Gamma = \{0, 1, X, Y, \square\}$,
- $q = q_0$,
- $F = \{q_4\}$,
- $\delta =$

$$
\begin{aligned}
(q_0, 0) &\to (q_1, X, R) & (q_0, Y) &\to (q_3, Y, R) \\
(q_1, 0) &\to (q_1, 0, R) & (q_1, Y) &\to (q_1, Y, R) \\
(q_1, 1) &\to (q_2, Y, L) & (q_2, 0) &\to (q_2, 0, L) \\
(q_2, Y) &\to (q_2, Y, L) & (q_2, X) &\to (q_1, X, R) \\
(q_3, Y) &\to (q_3, Y, R) & (q_3, \square) &\to (q_4, \square, R).
\end{aligned}
$$

Example for $\{0^n 1^n\} \mid n \geq 1$

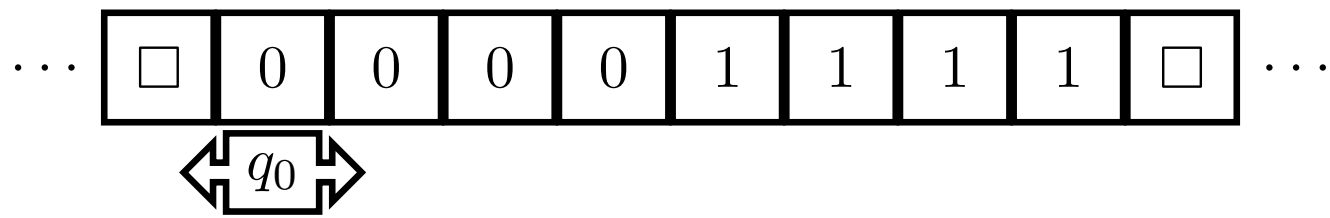
Formalities and examples

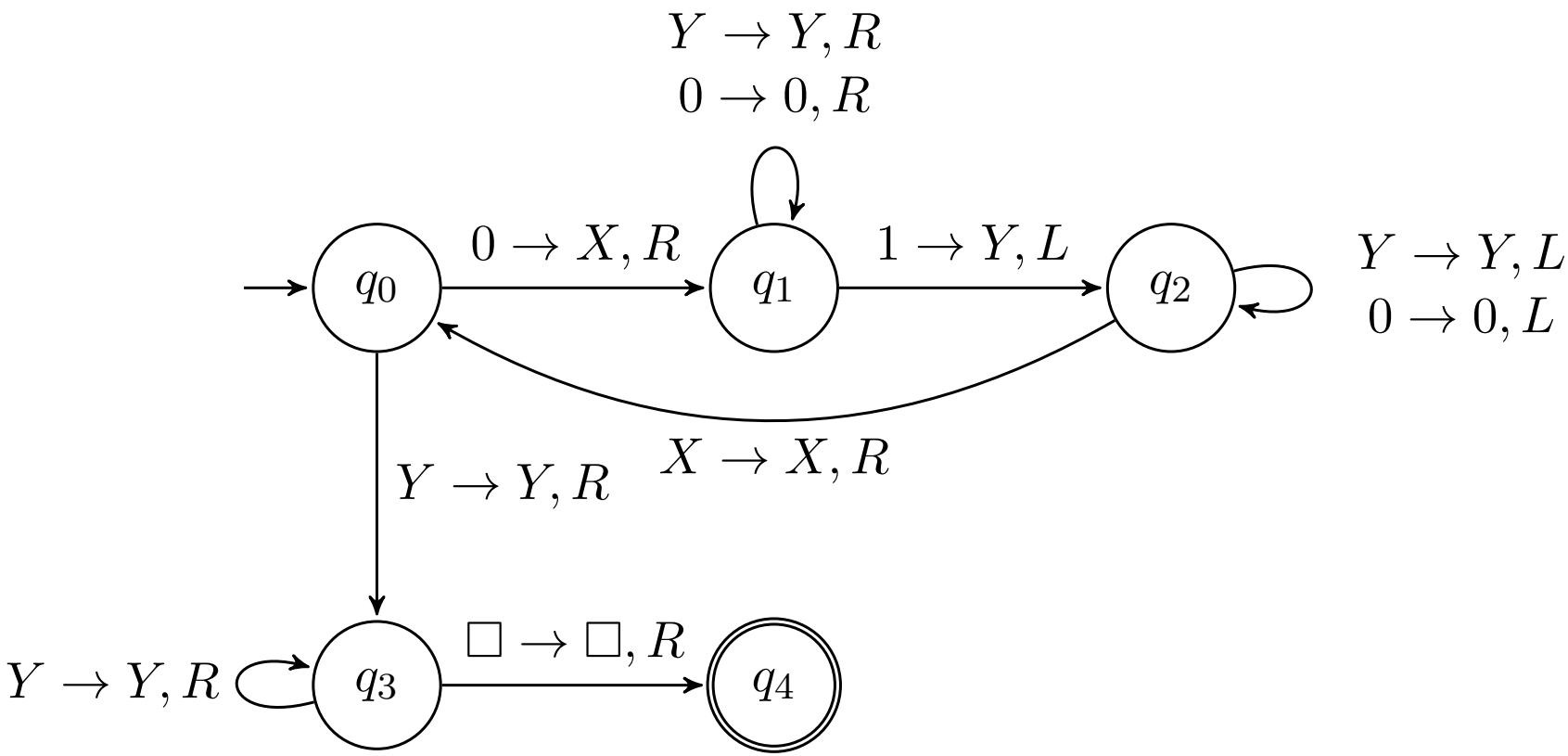
How can we formally represent a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q, F)$?

	δ				
Q \ Γ	0	1	X	Y	$\square$
$\rightarrow q_0$	(q_1, X, R)	—	—	(q_3, Y, R)	—
q_1	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)	—
q_2	$(q_2, 0, L)$	—	(q_1, X, R)	(q_2, Y, L)	—
q_3	—	—	—	(q_3, Y, R)	$(q_4, \square, R)$
$*q_4$	—	—	—	—	—

Example for 00001111

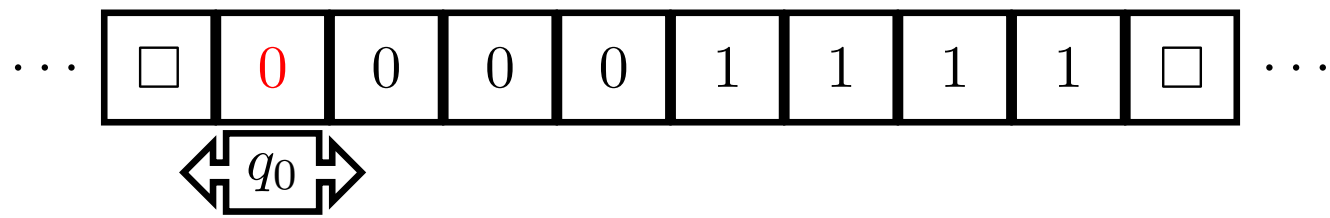
Formalities and examples

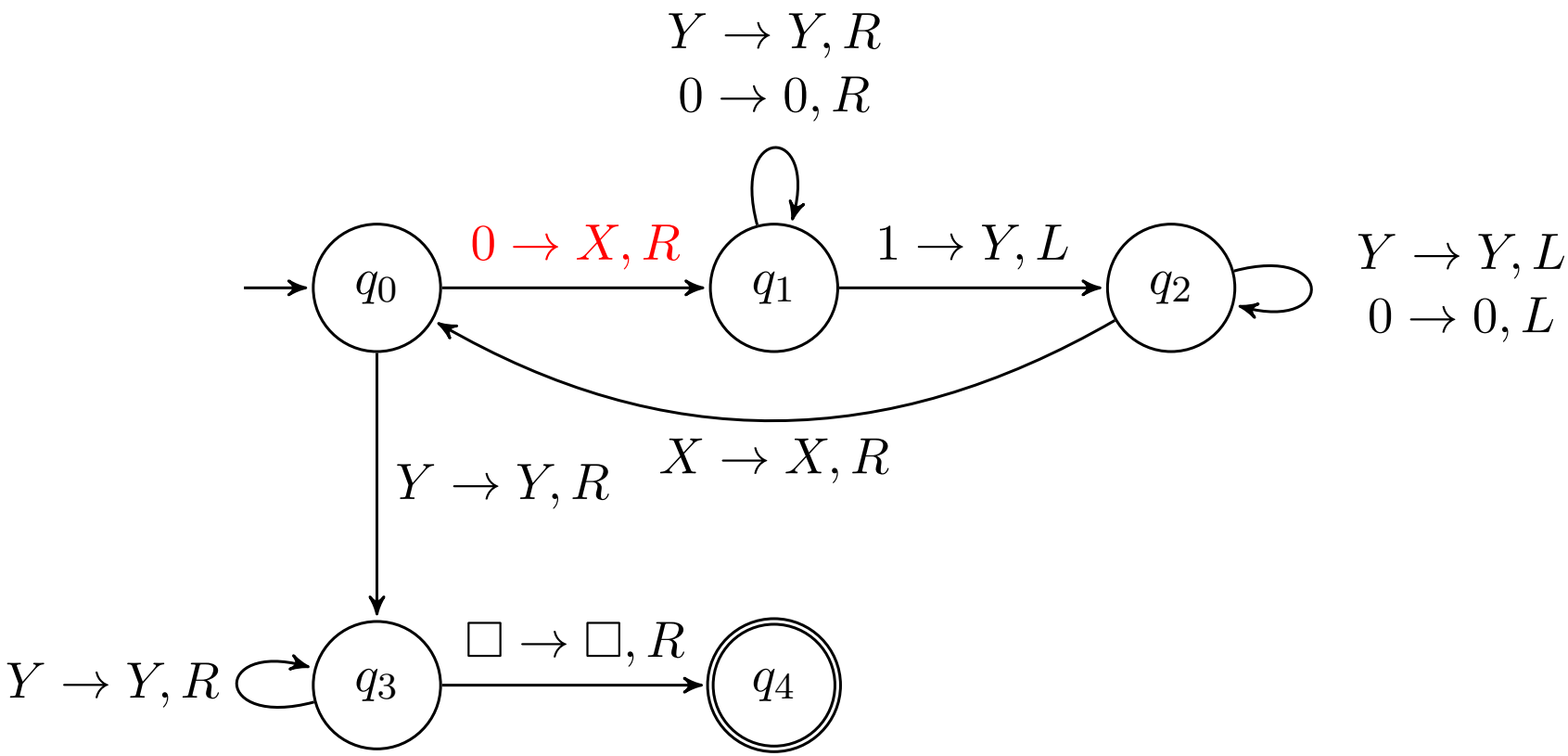




Example for 00001111

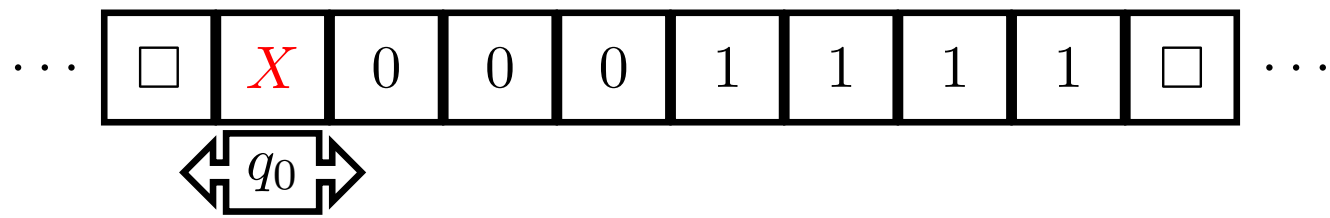
Formalities and examples

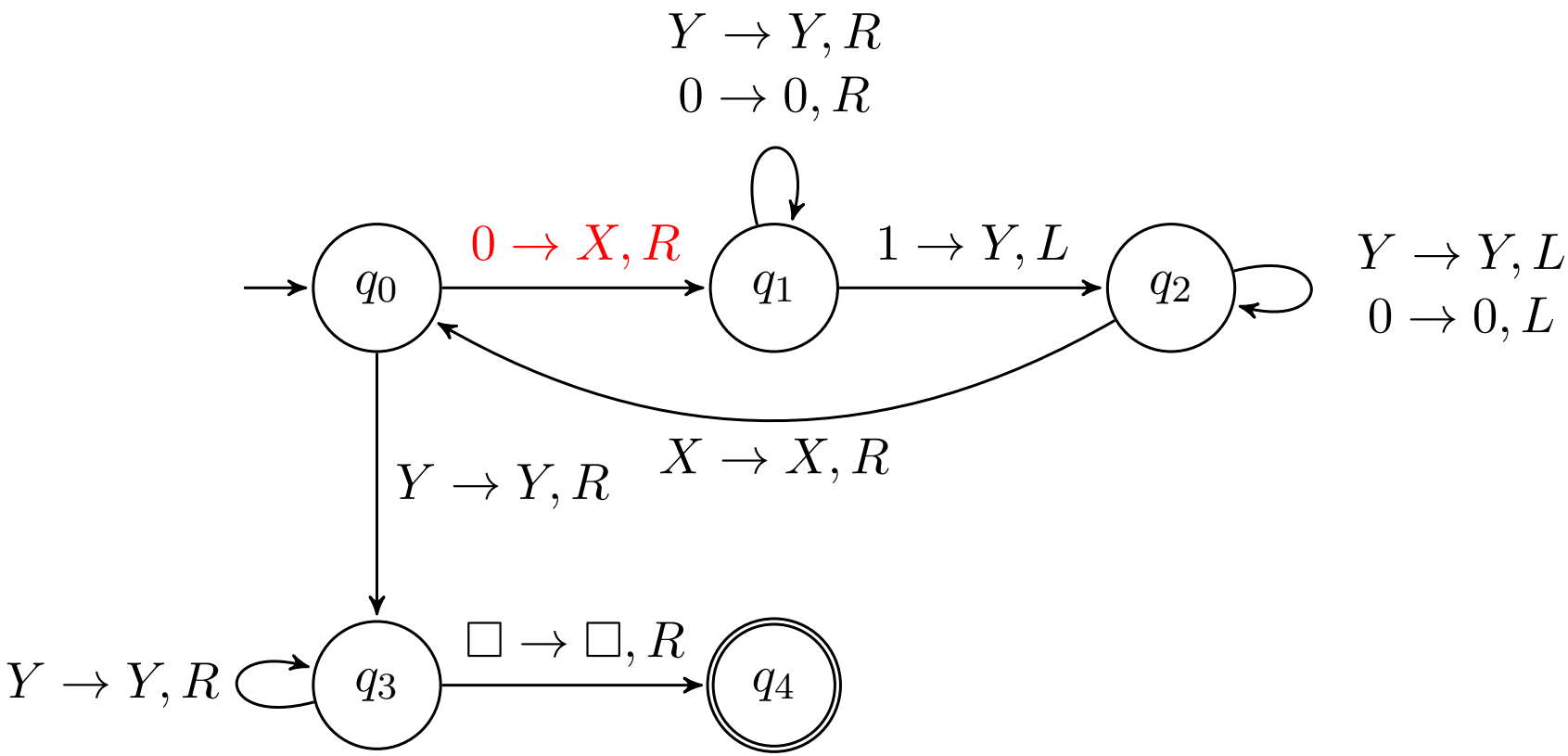




Example for 00001111

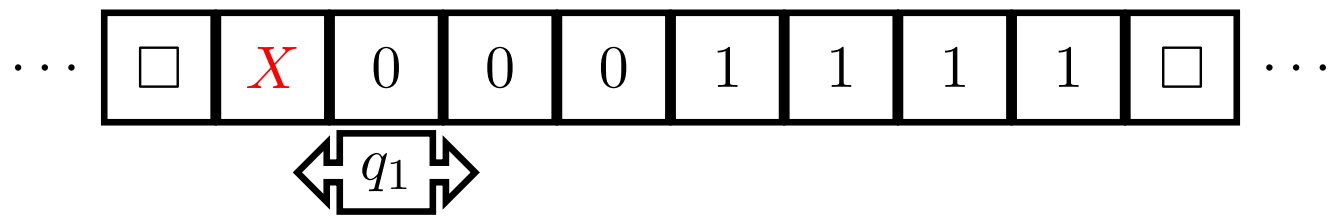
Formalities and examples

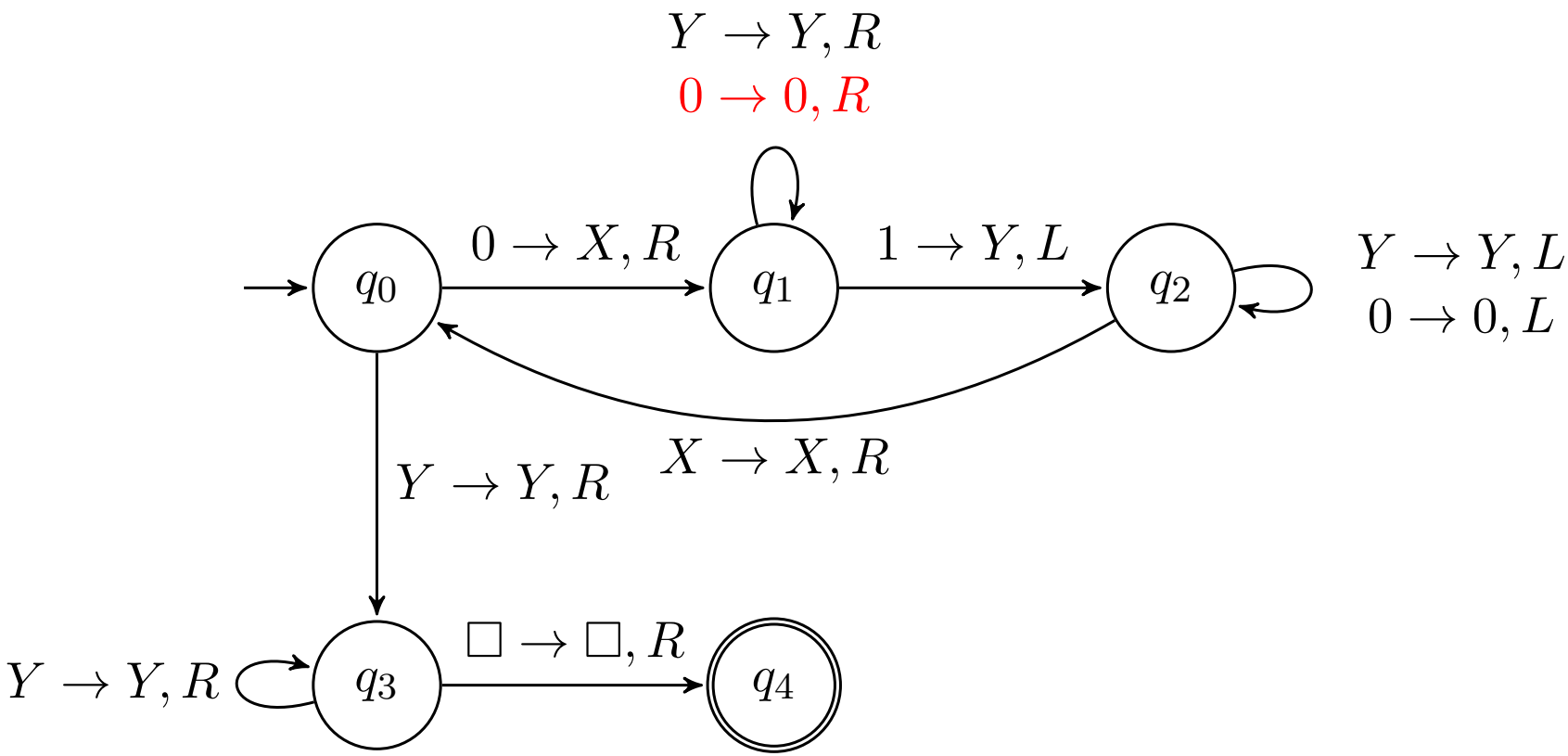




Example for 00001111

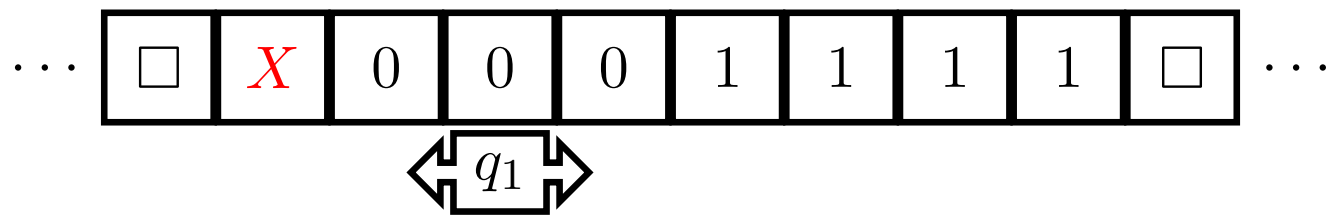
Formalities and examples

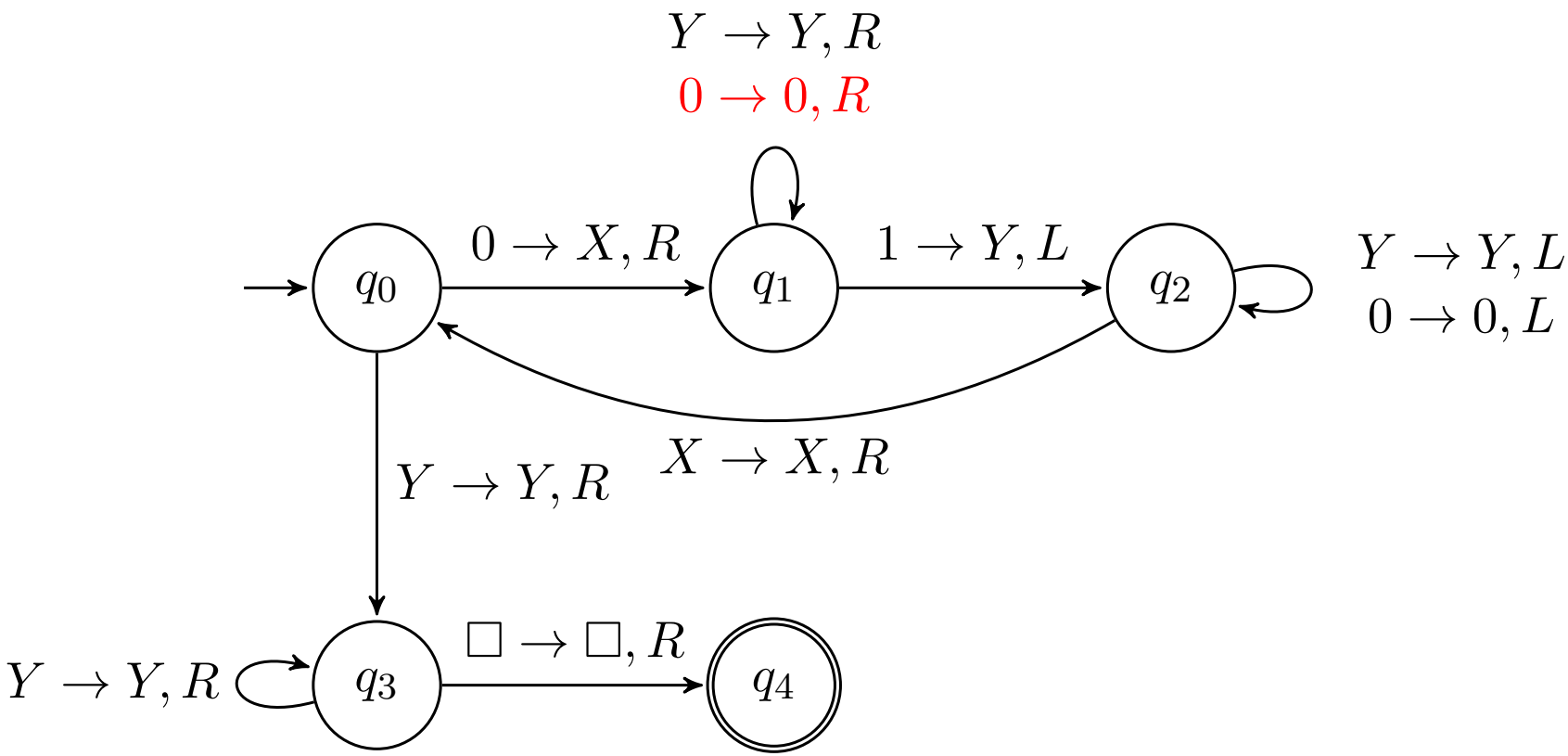




Example for 00001111

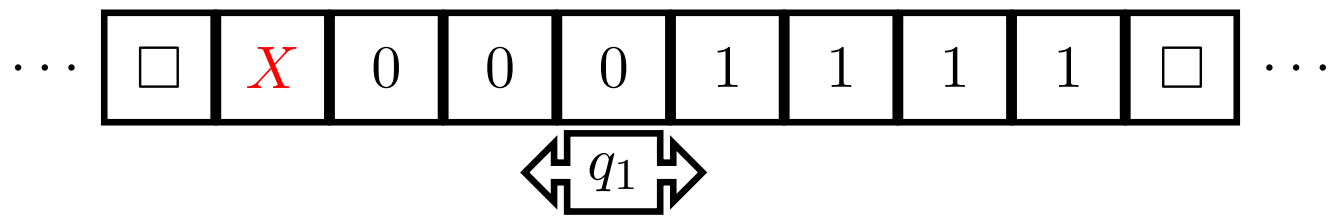
Formalities and examples

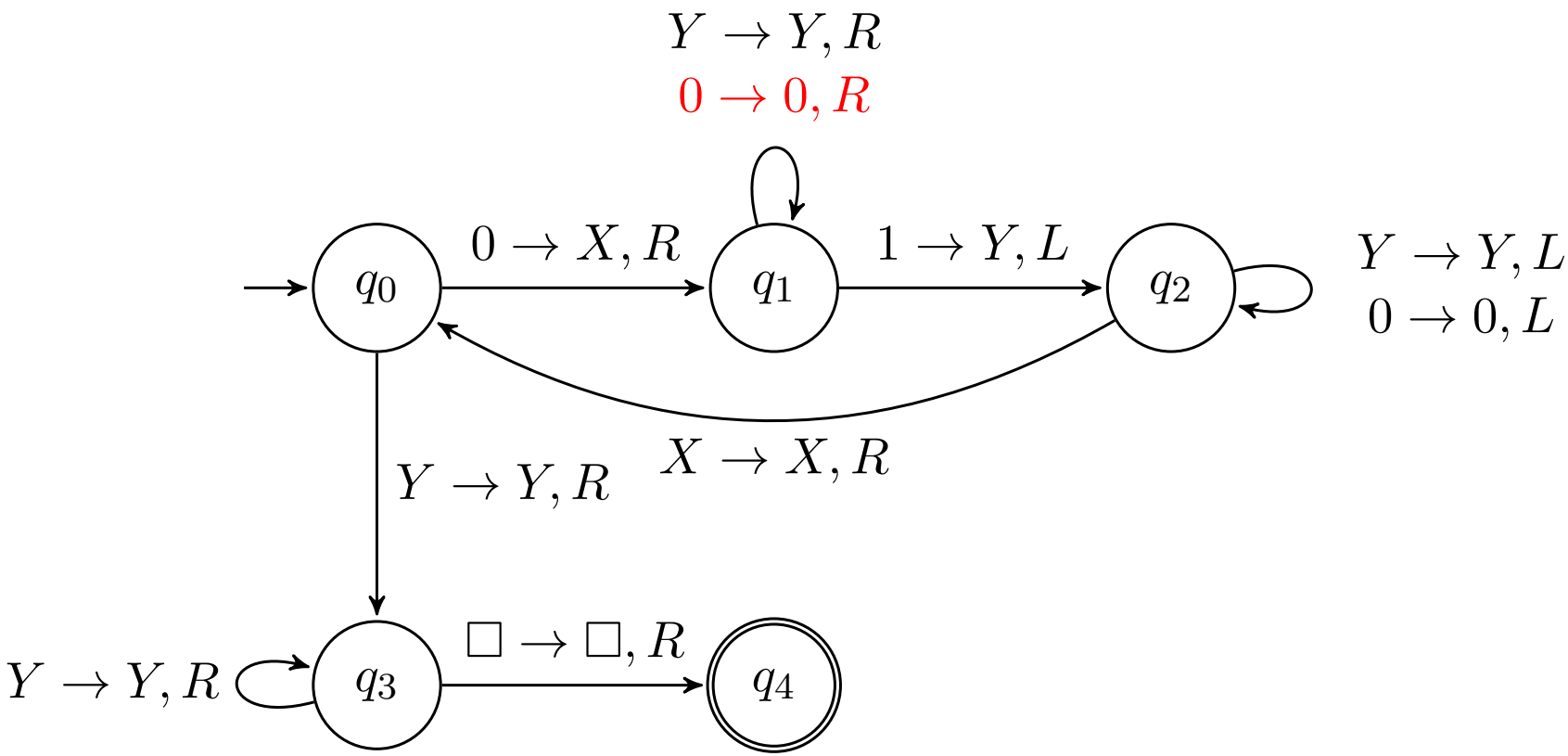




Example for 00001111

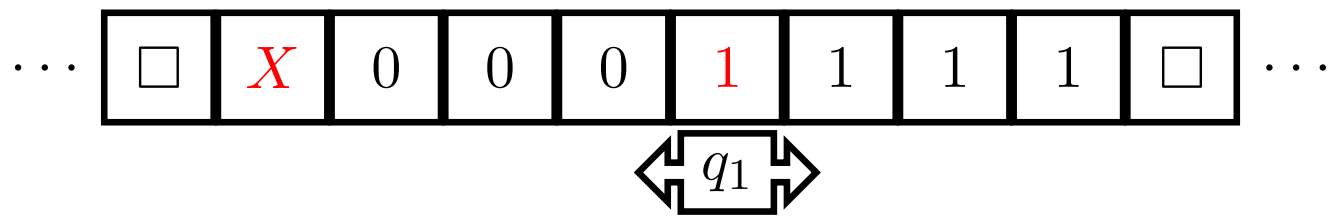
Formalities and examples

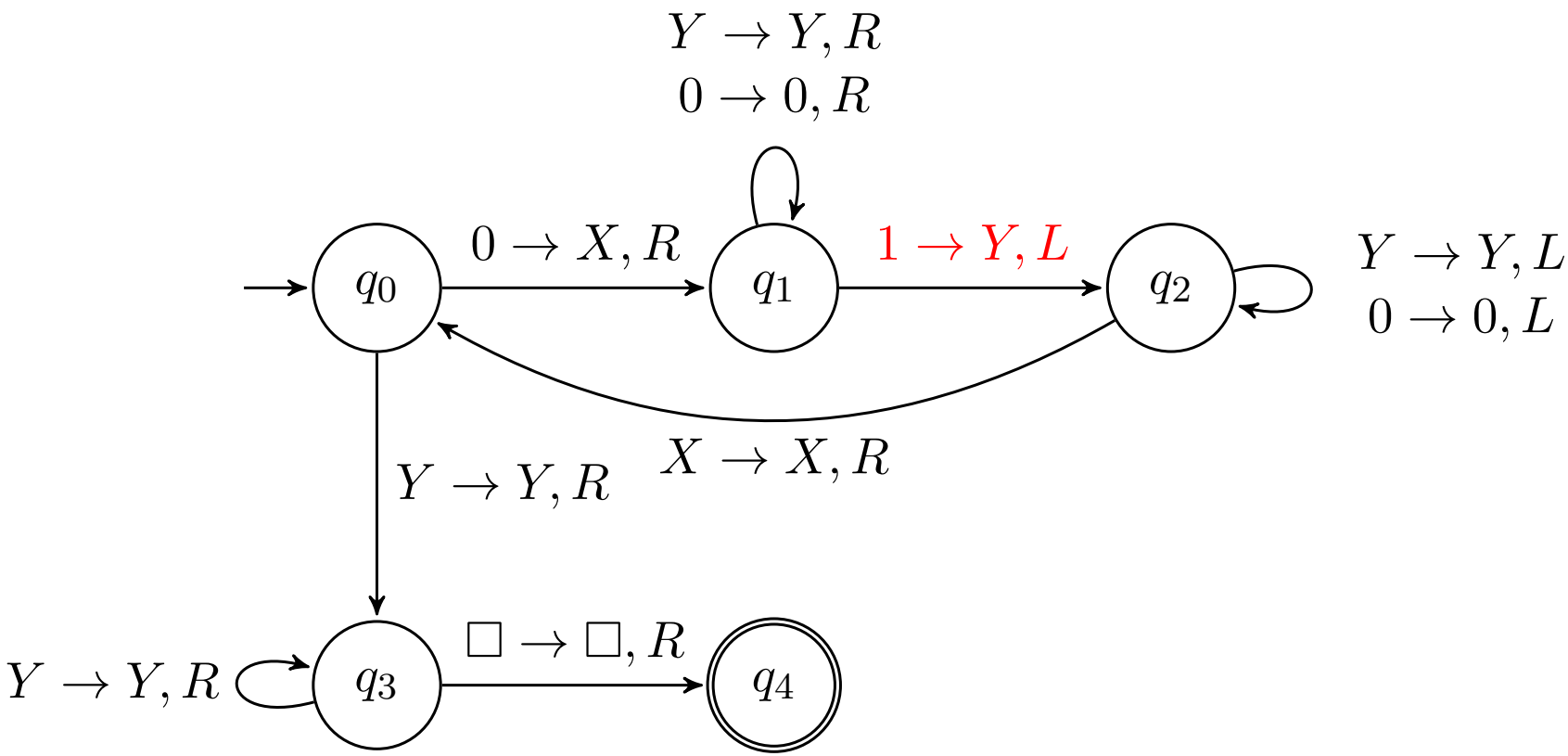




Example for 00001111

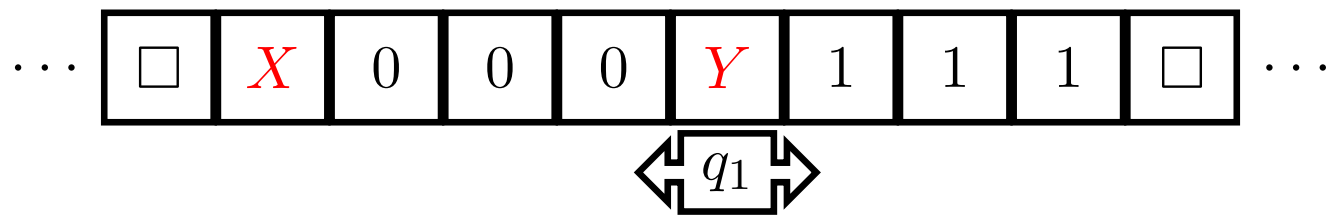
Formalities and examples

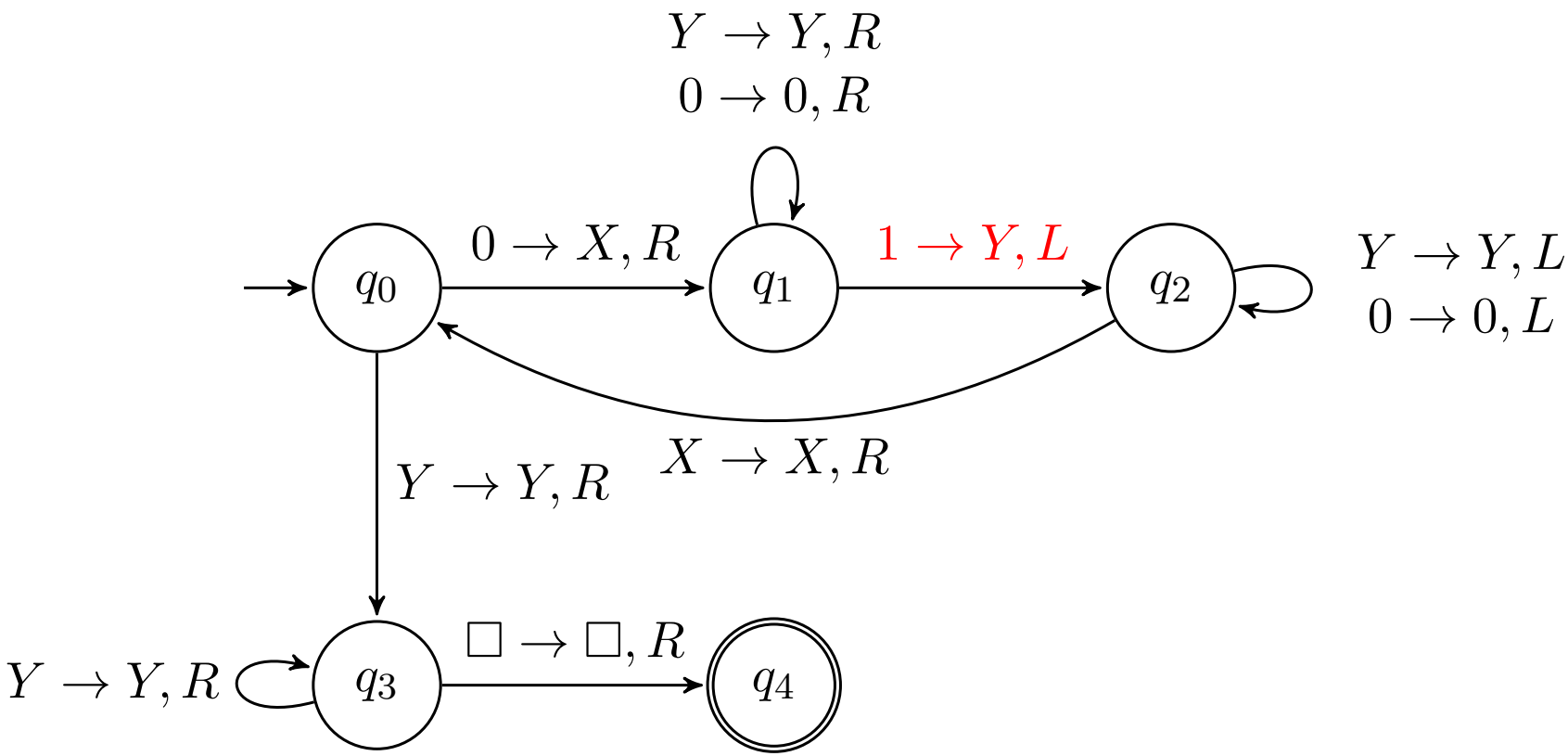




Example for 00001111

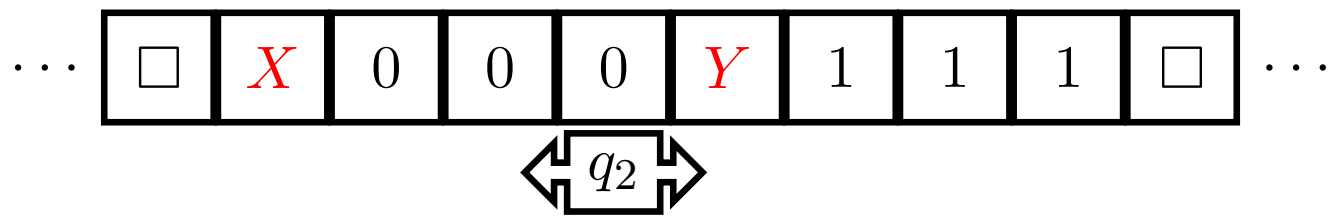
Formalities and examples

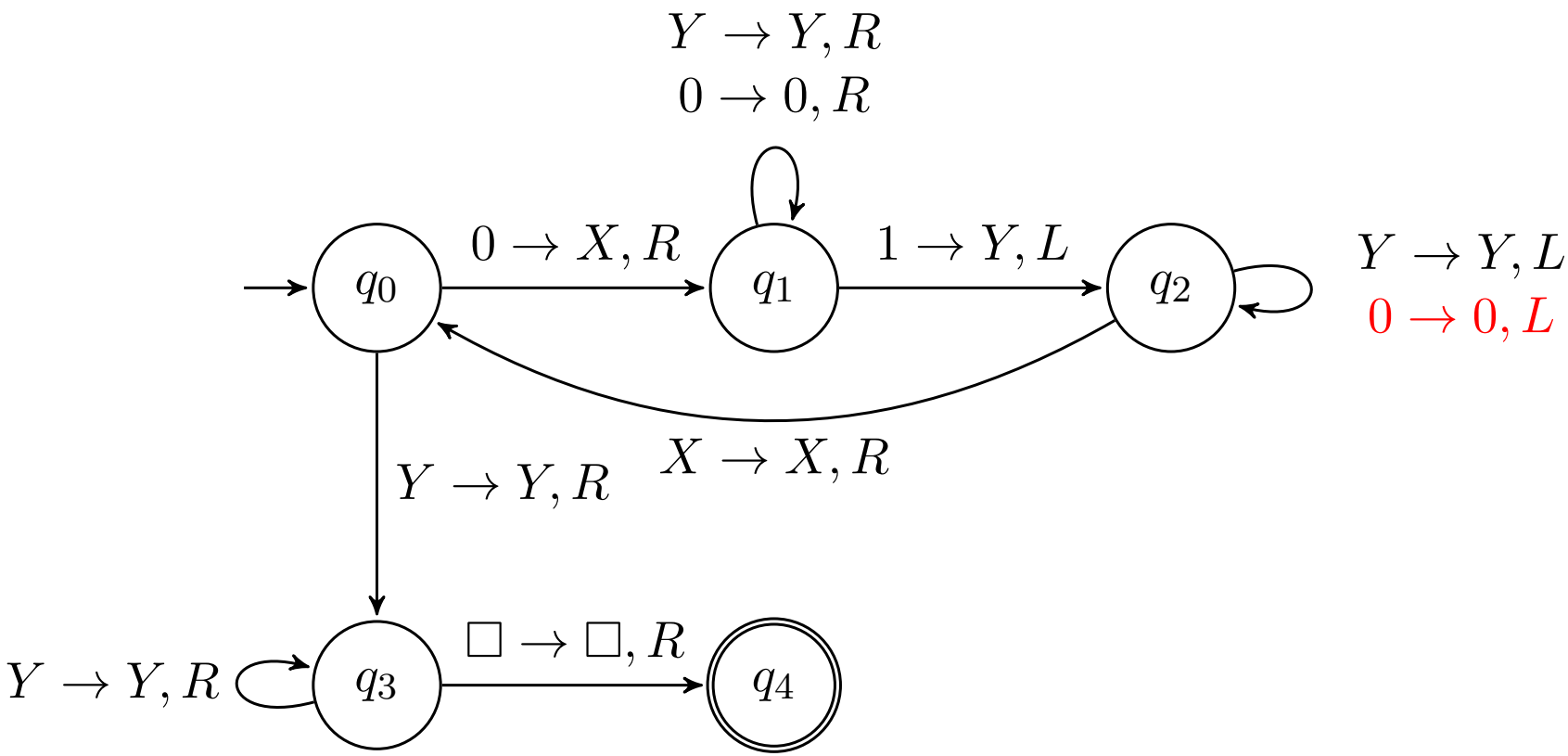




Example for 00001111

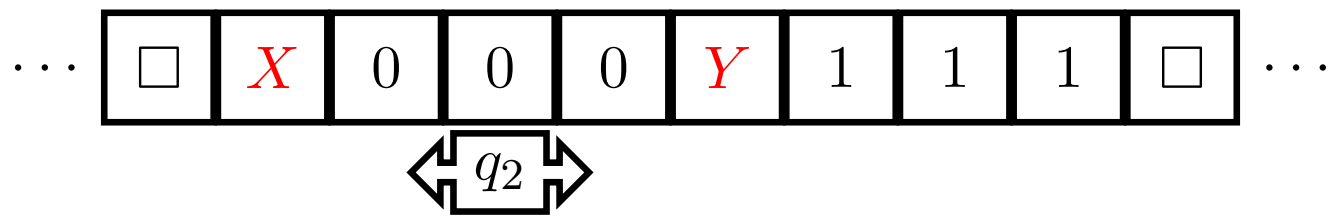
Formalities and examples

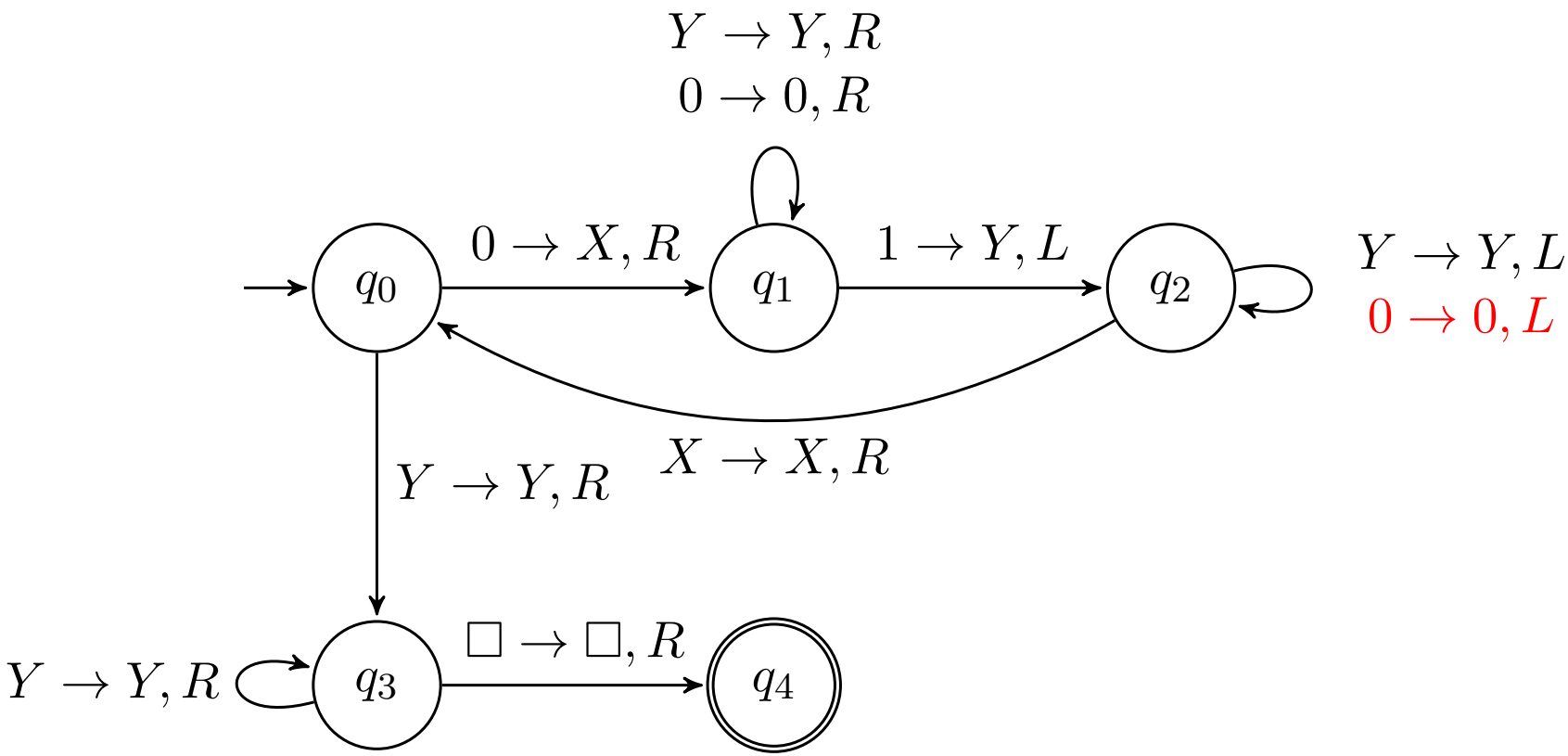




Example for 00001111

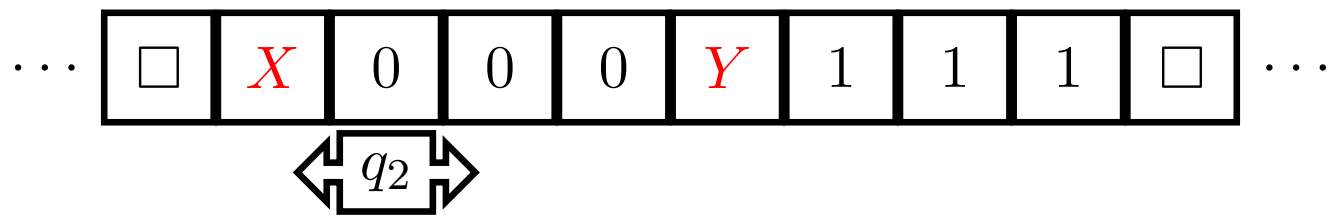
Formalities and examples

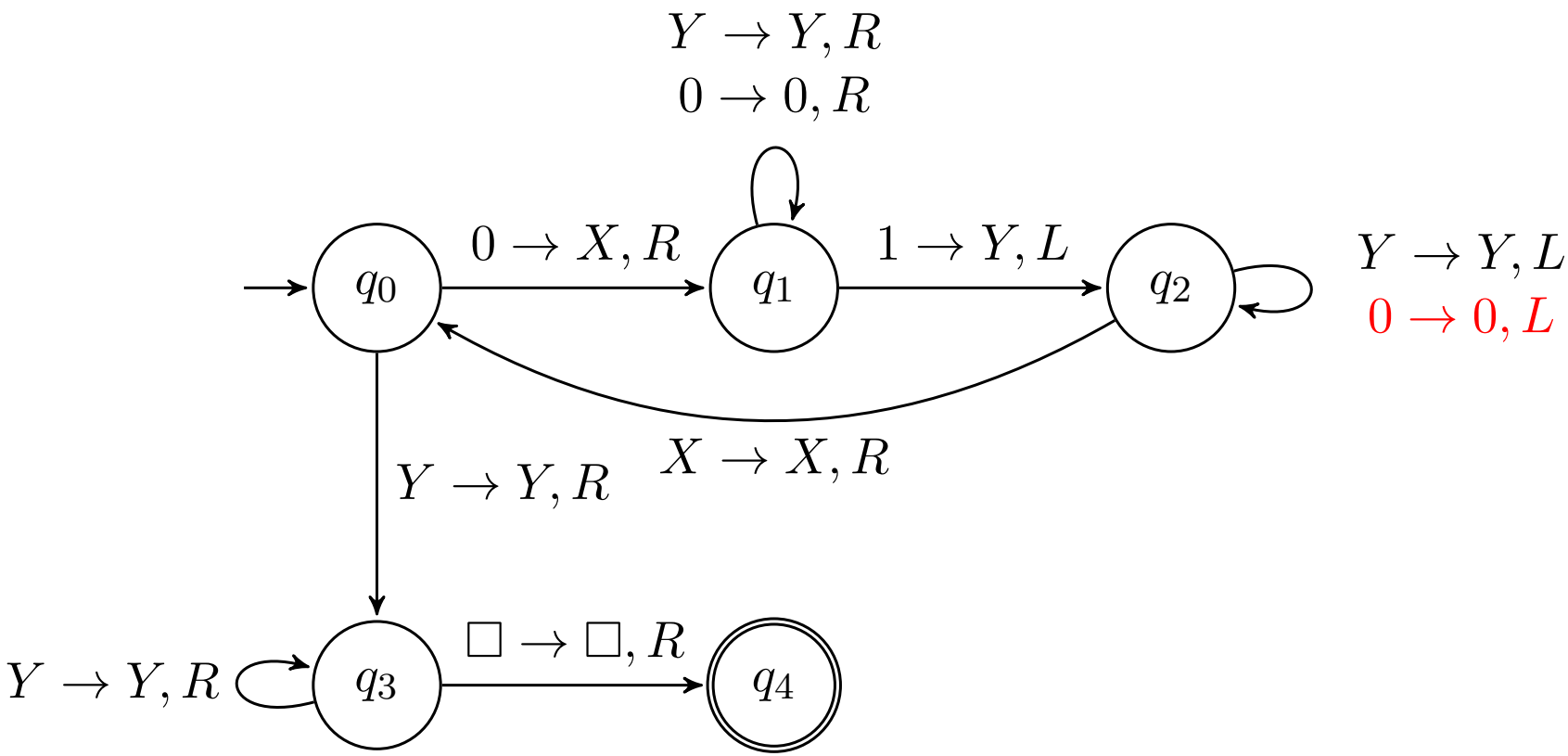




Example for 00001111

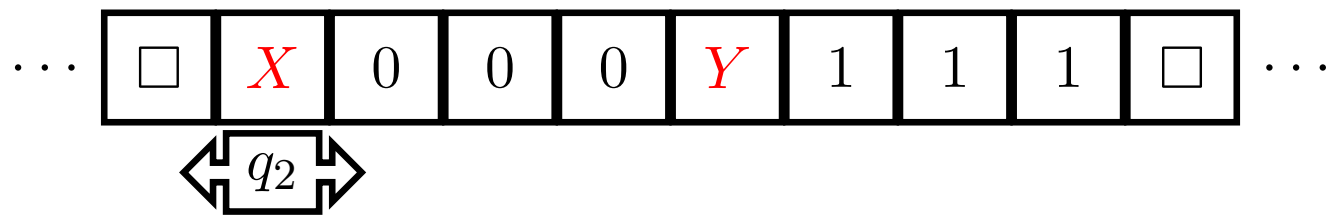
Formalities and examples

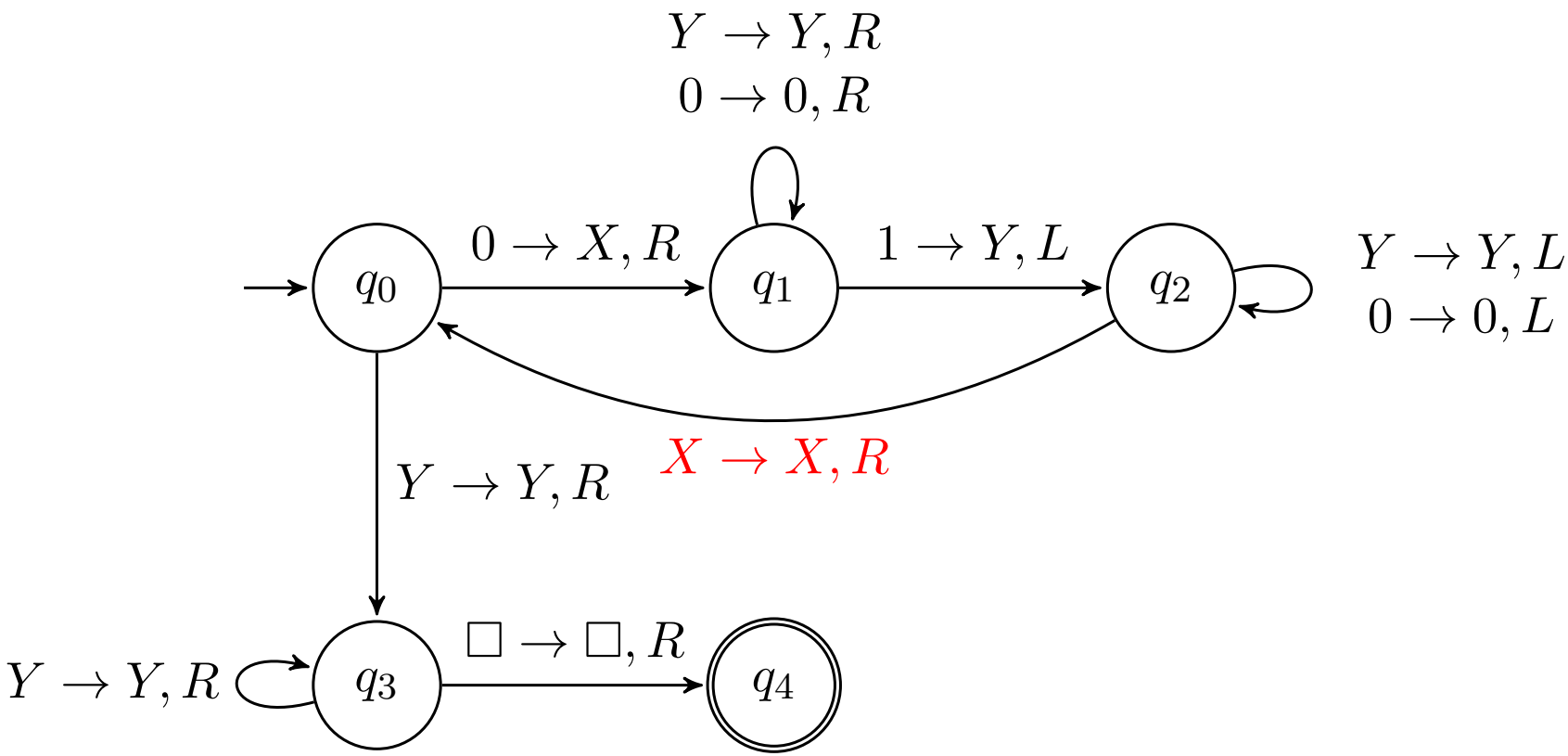




Example for 00001111

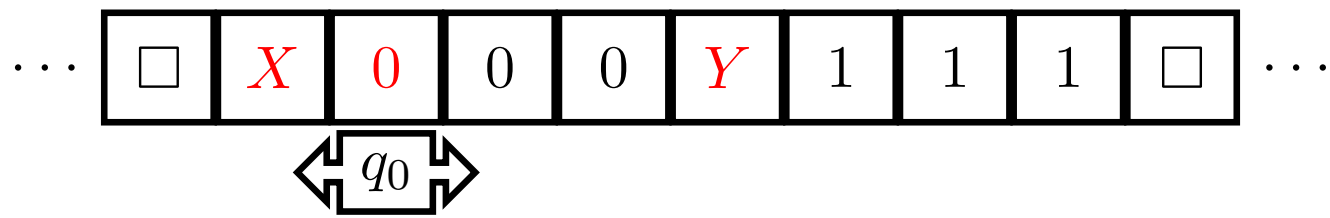
Formalities and examples

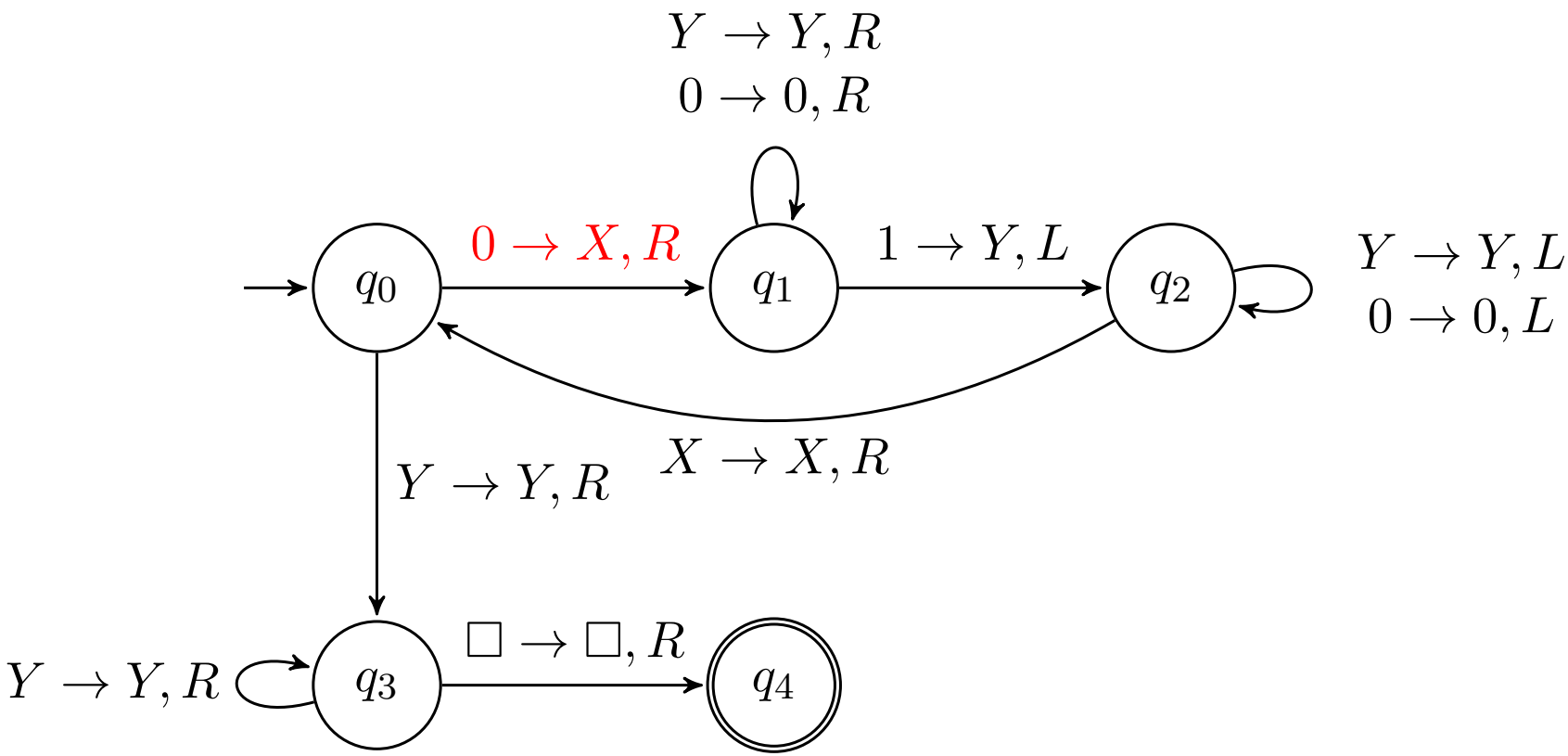




Example for 00001111

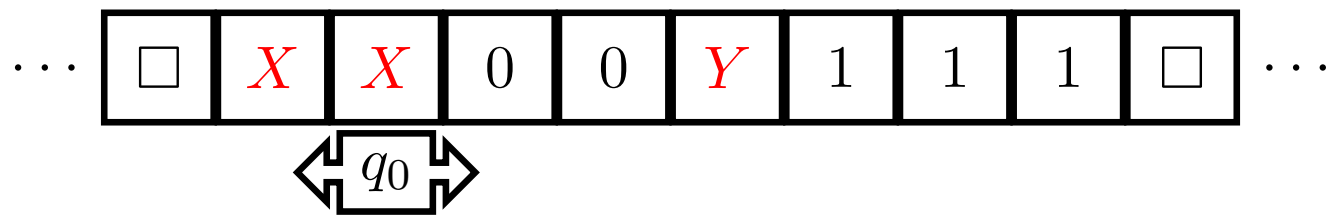
Formalities and examples

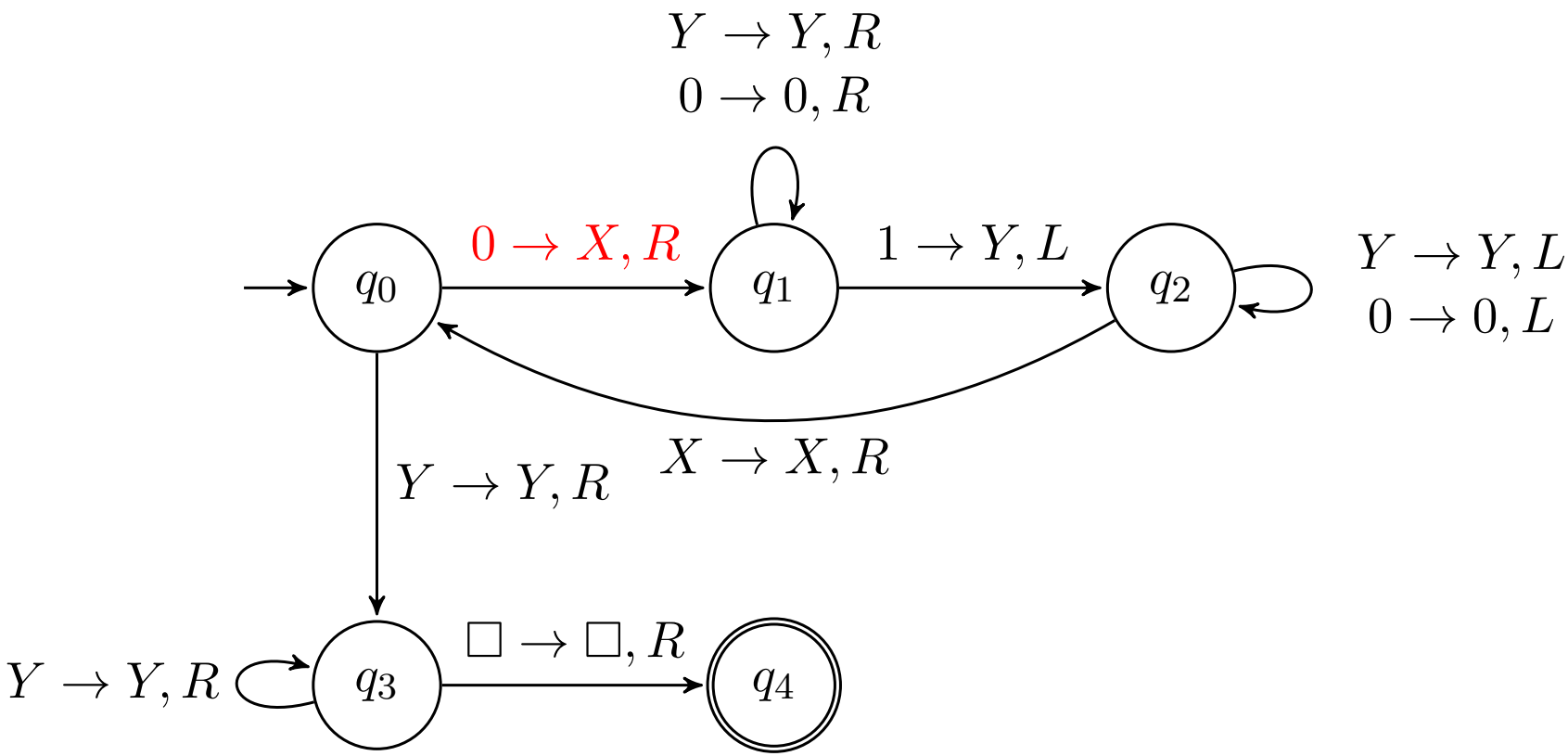




Example for 00001111

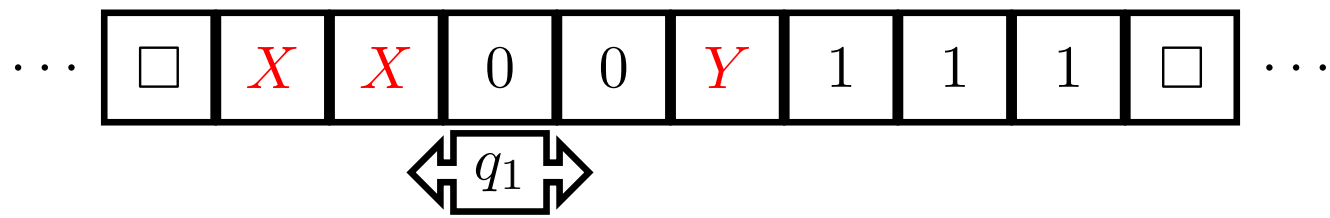
Formalities and examples

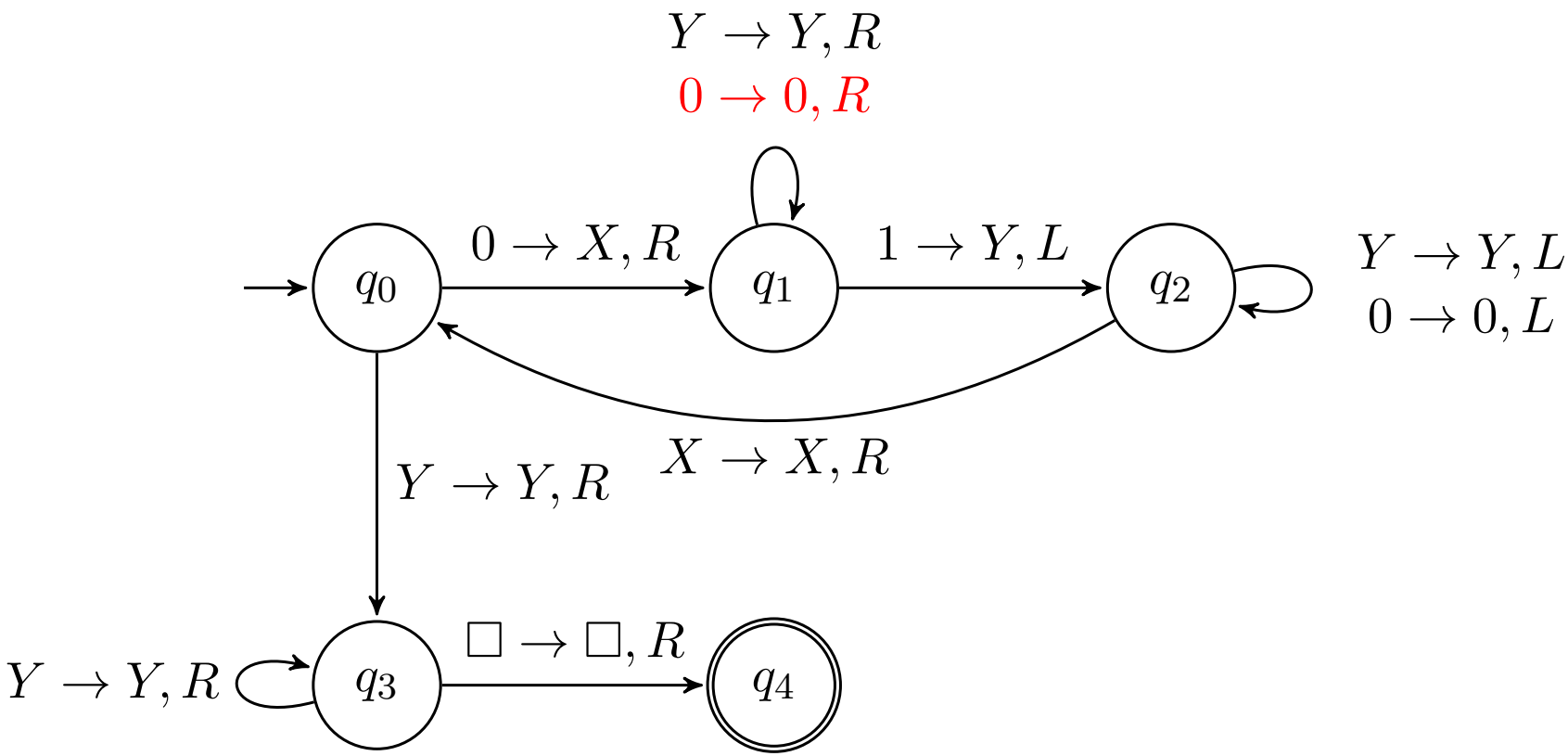




Example for 00001111

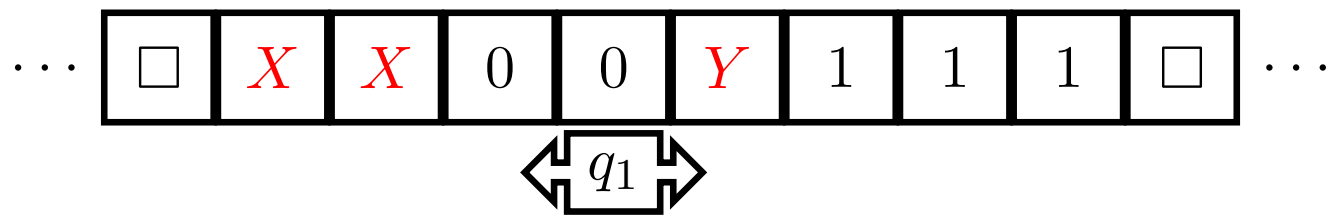
Formalities and examples

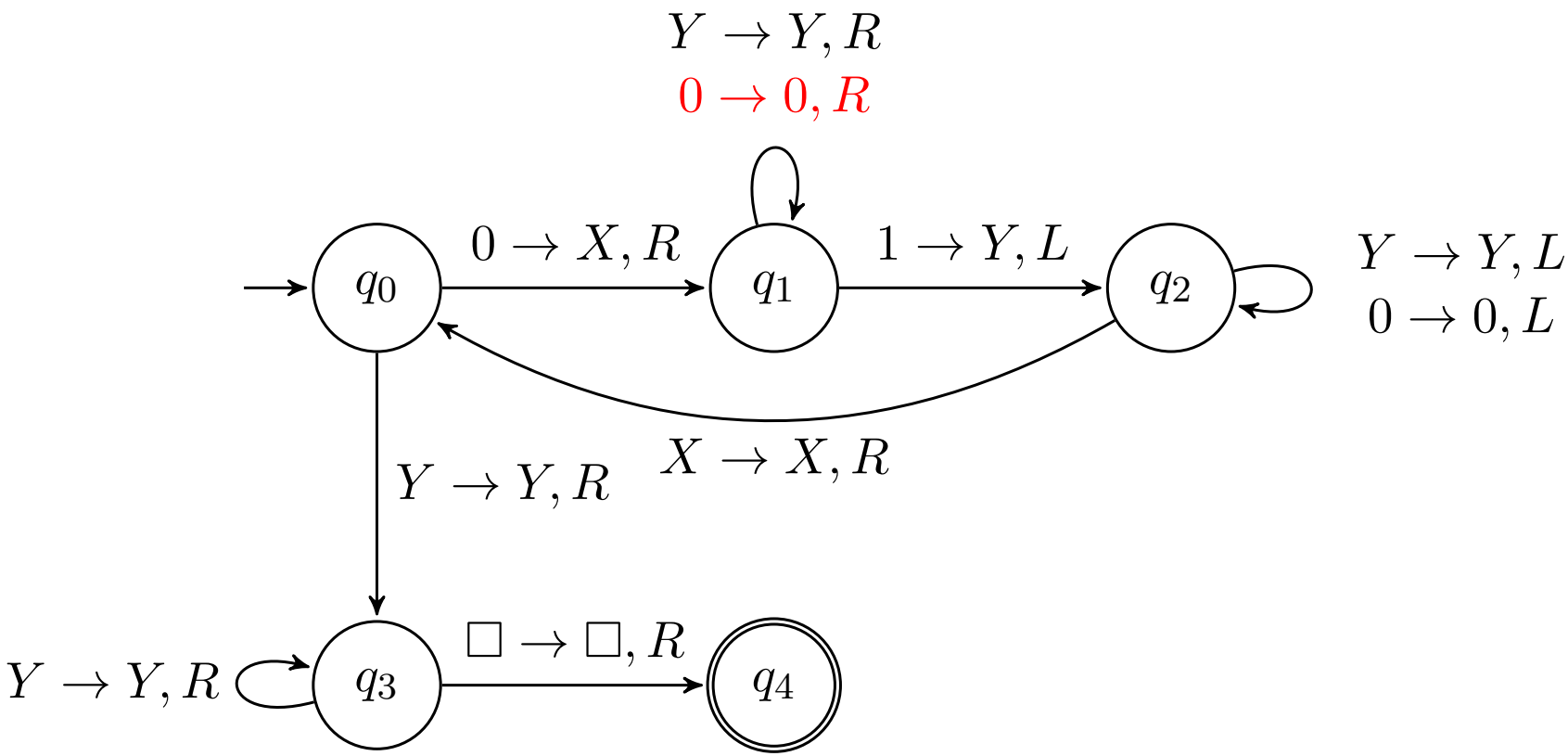




Example for 00001111

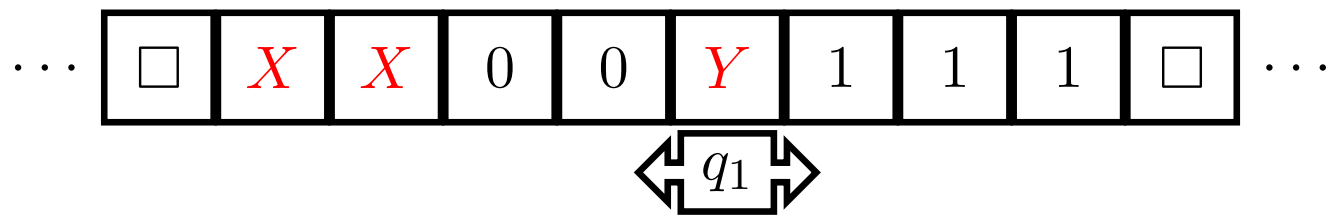
Formalities and examples

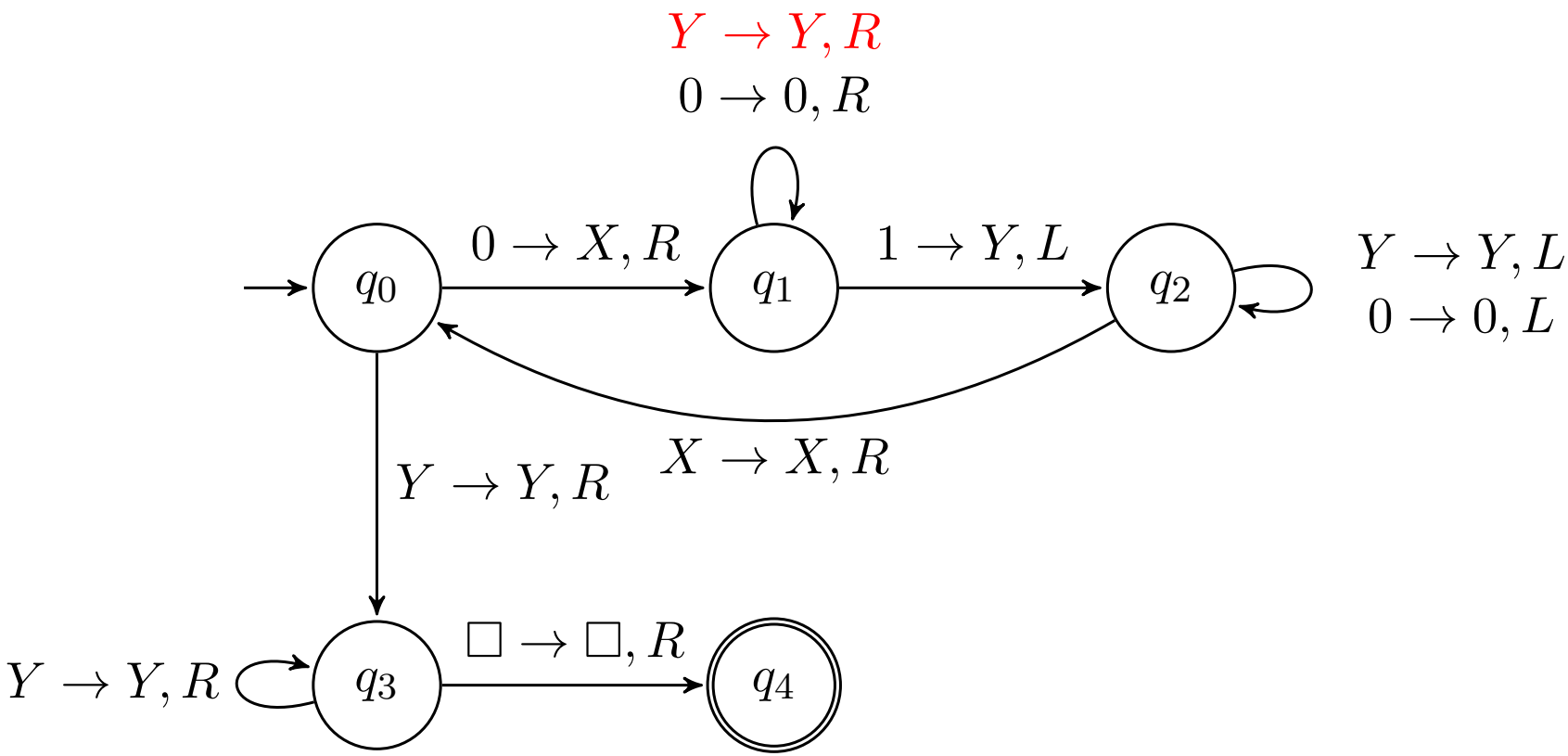




Example for 00001111

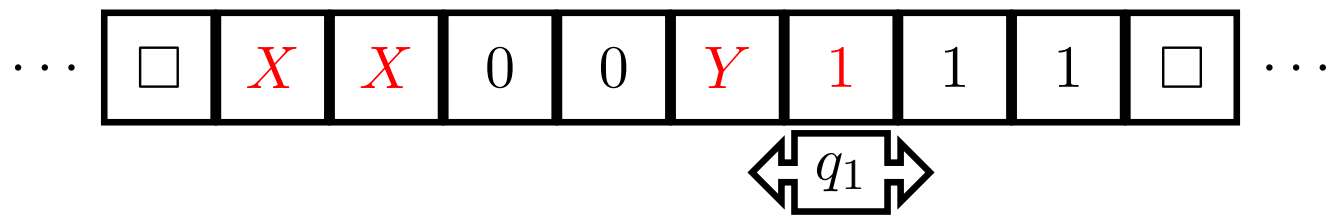
Formalities and examples

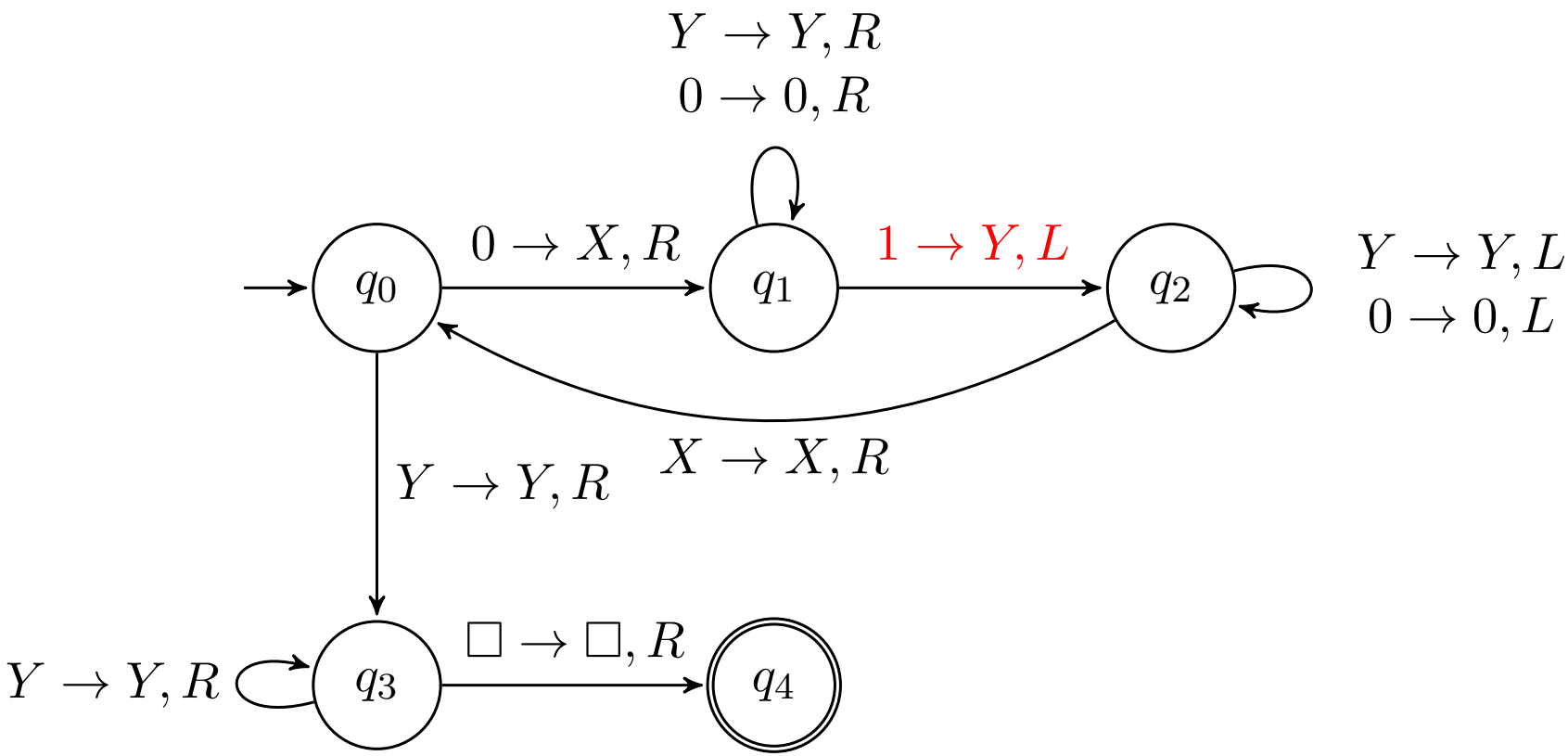




Example for 00001111

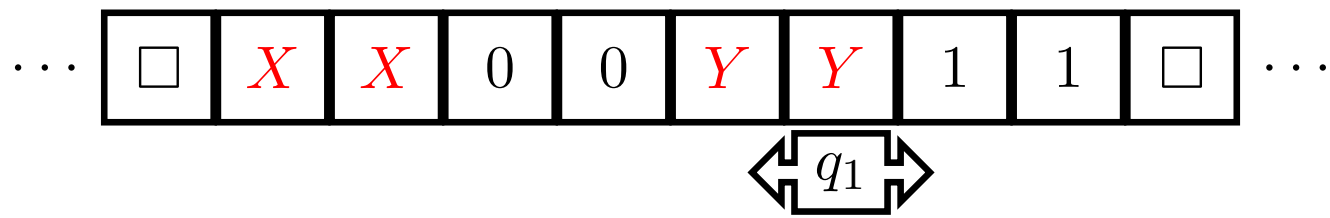
Formalities and examples

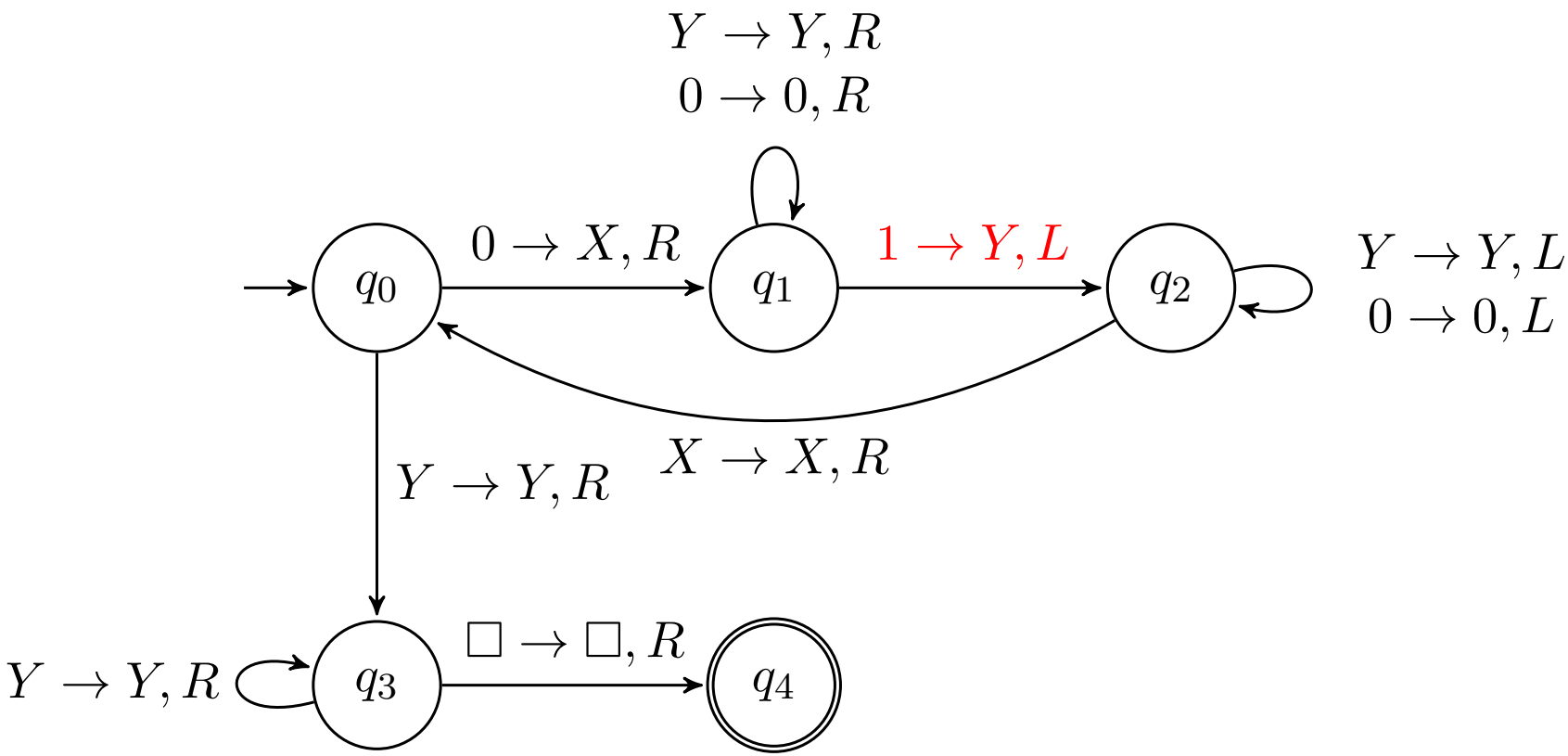




Example for 00001111

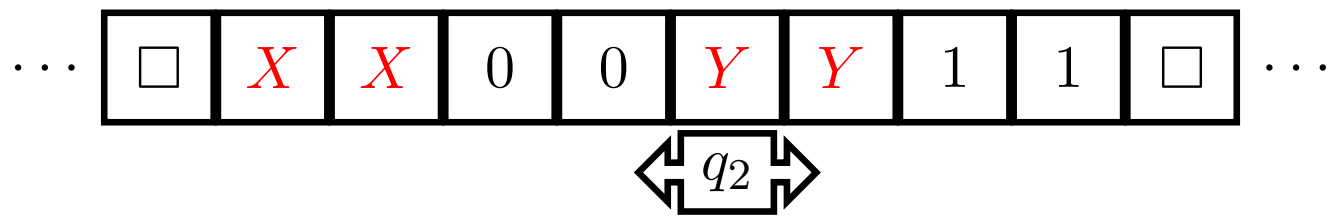
Formalities and examples

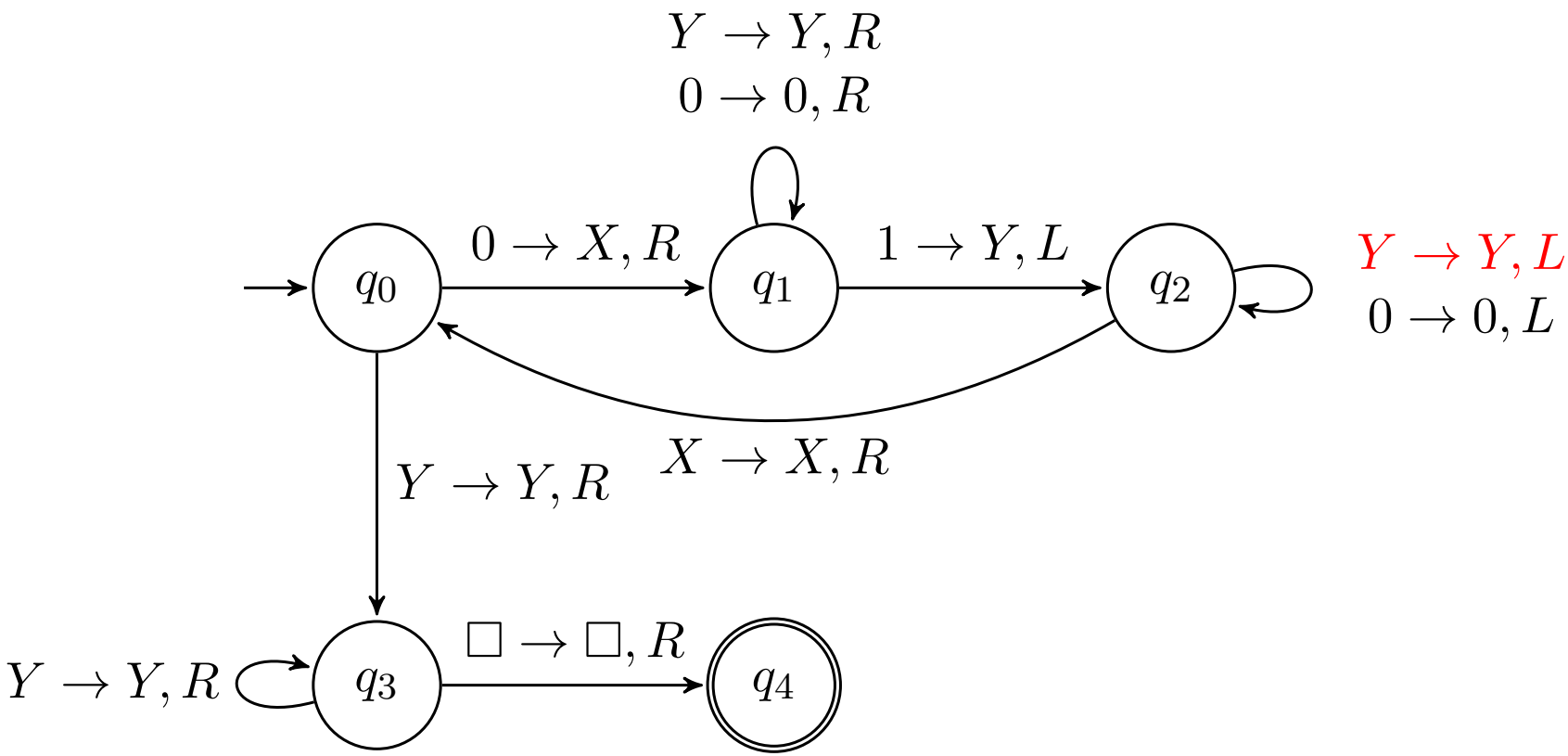




Example for 00001111

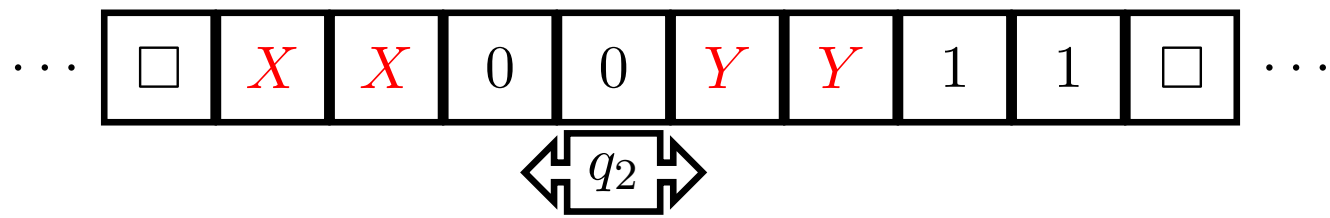
Formalities and examples

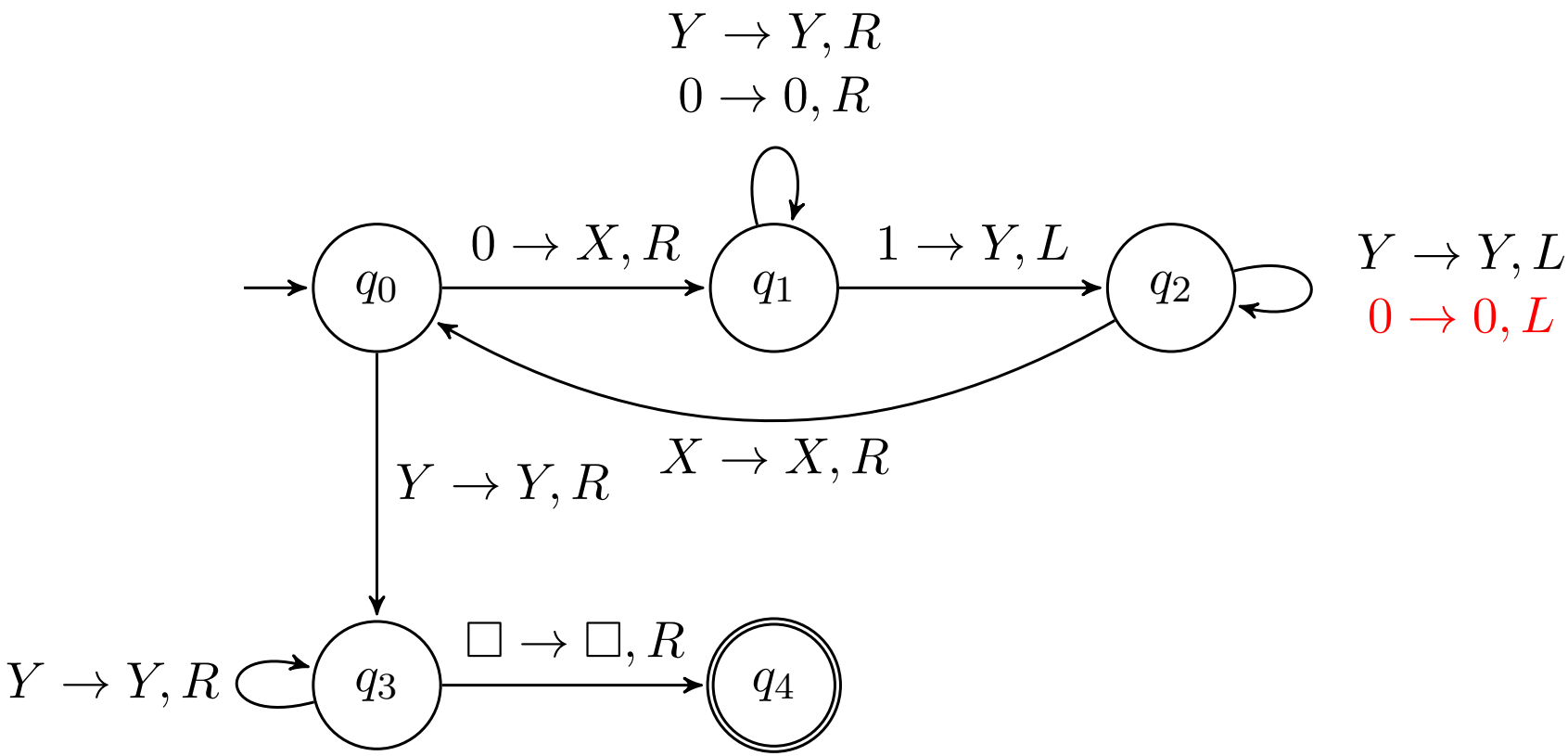




Example for 00001111

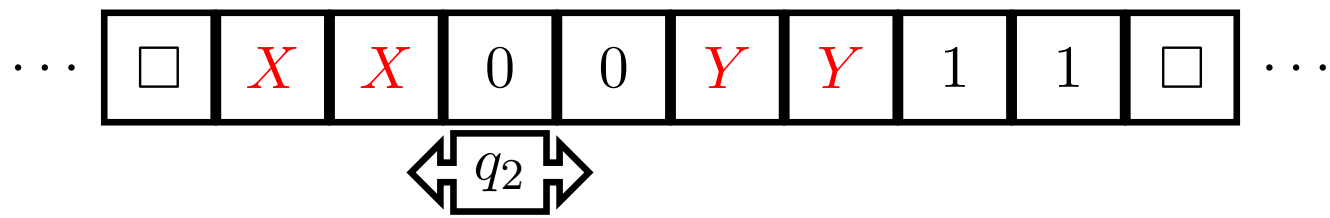
Formalities and examples

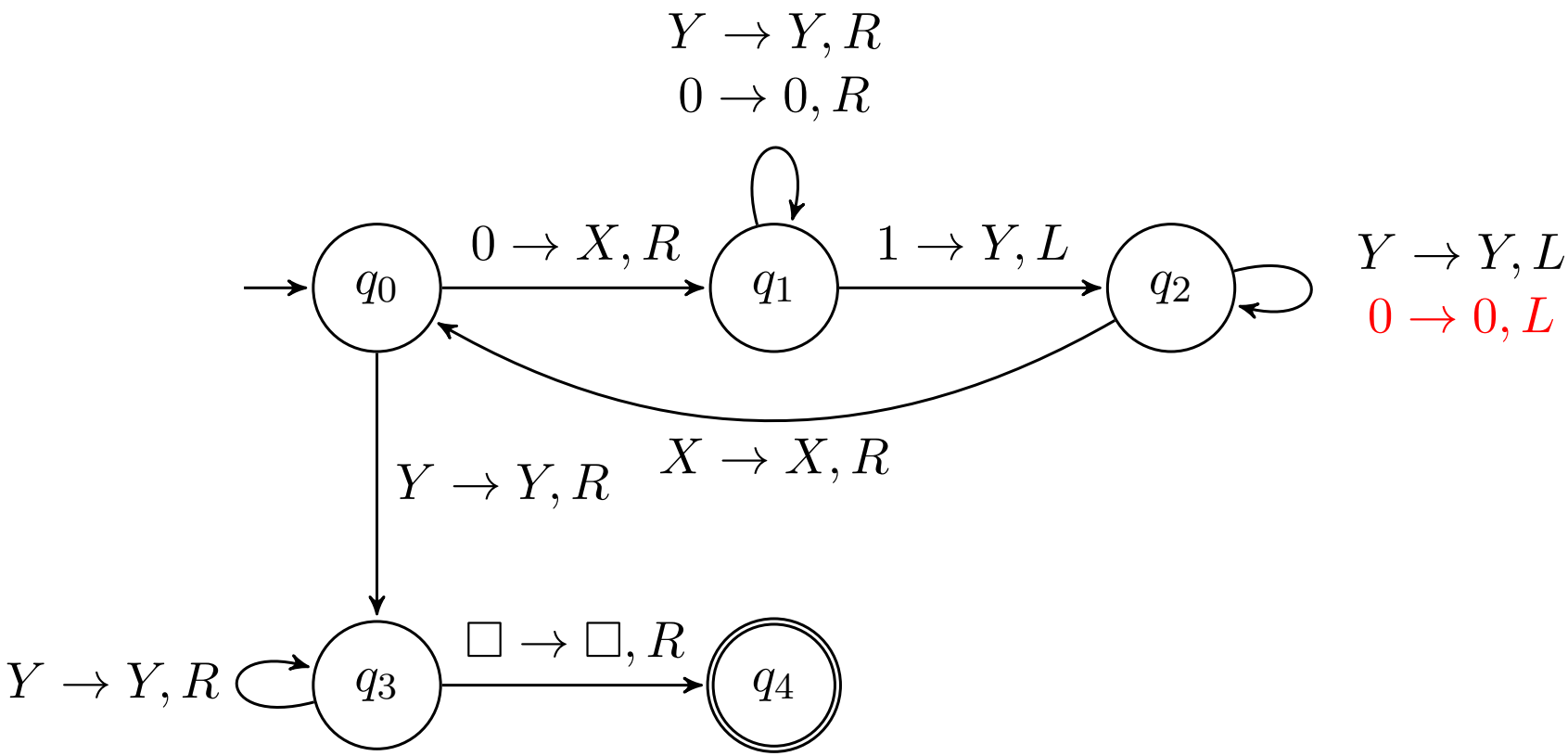




Example for 00001111

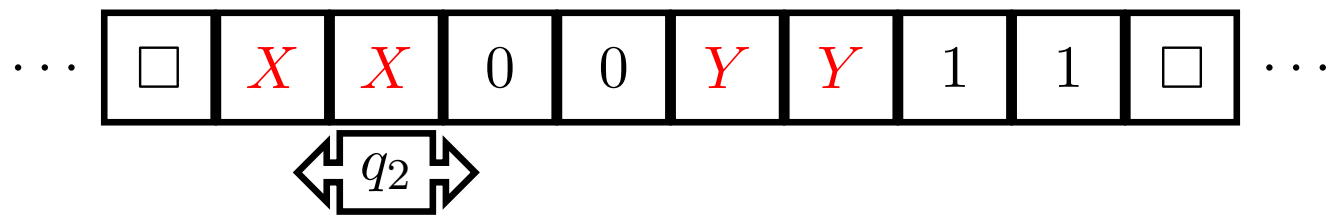
Formalities and examples

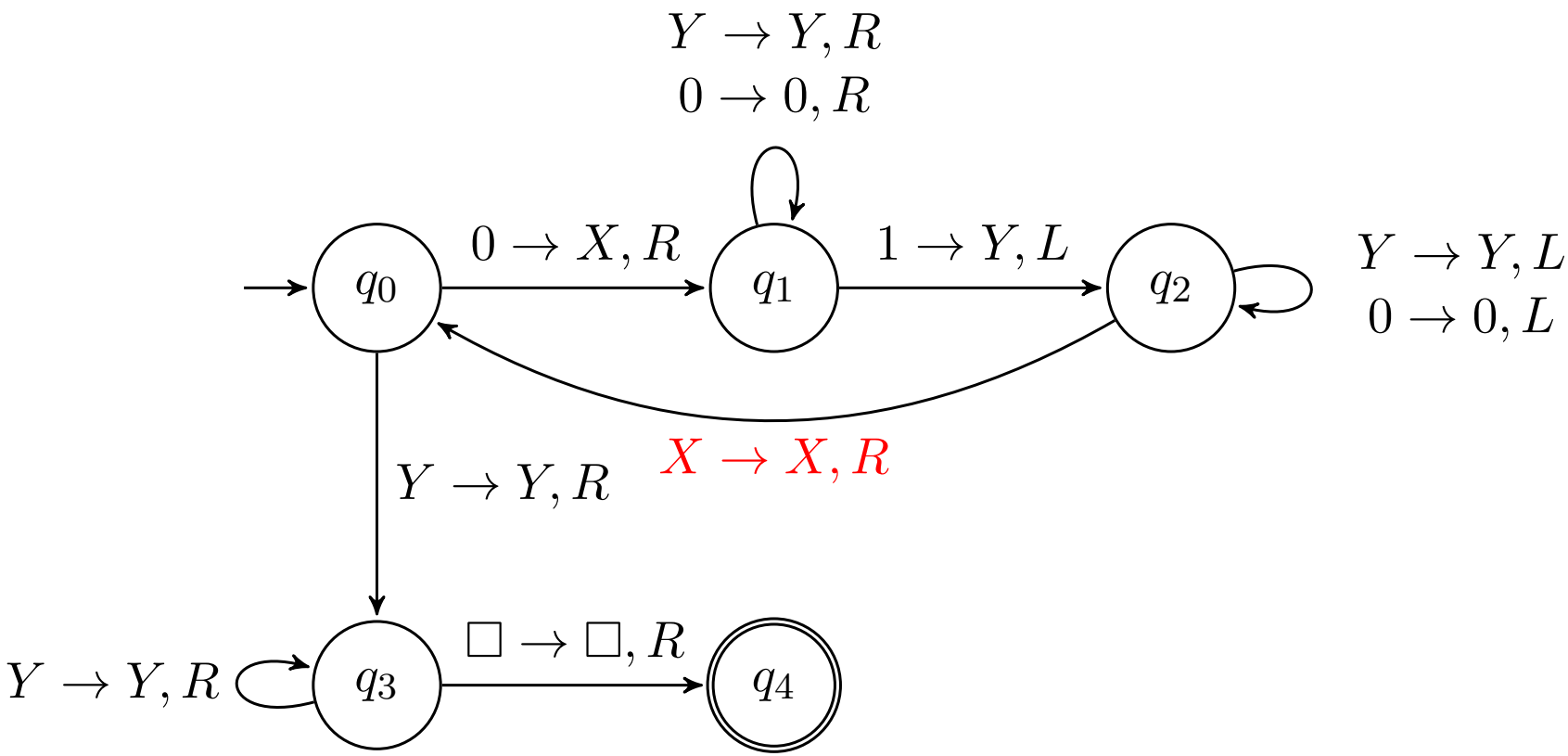




Example for 00001111

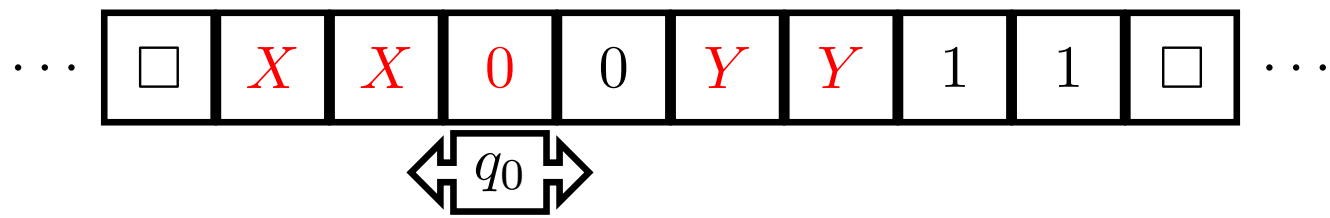
Formalities and examples

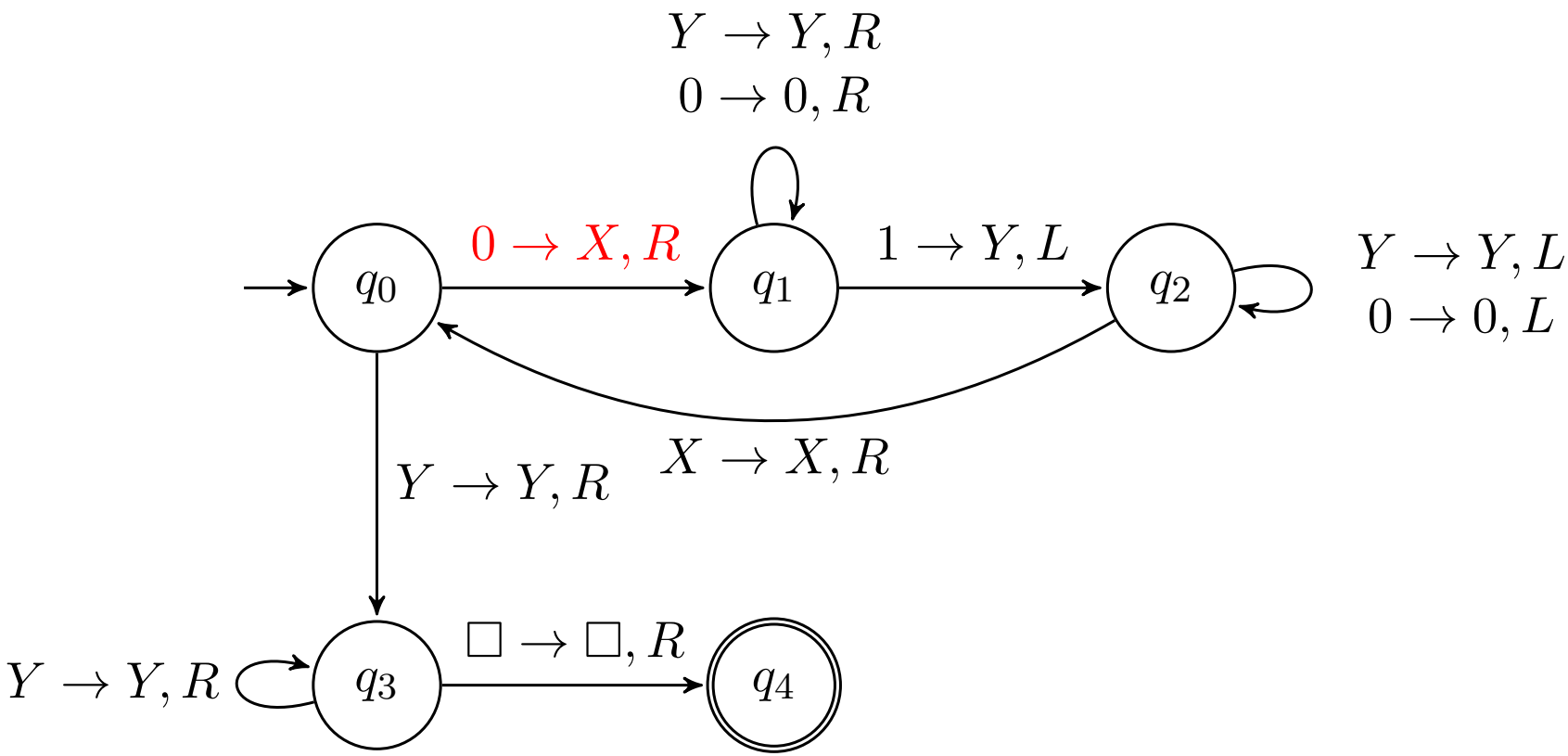




Example for 00001111

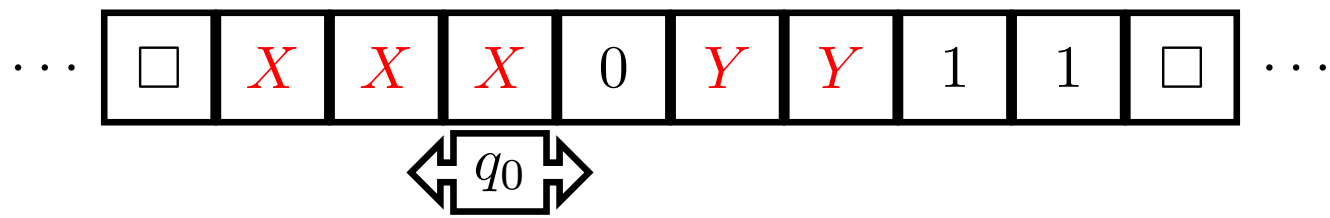
Formalities and examples

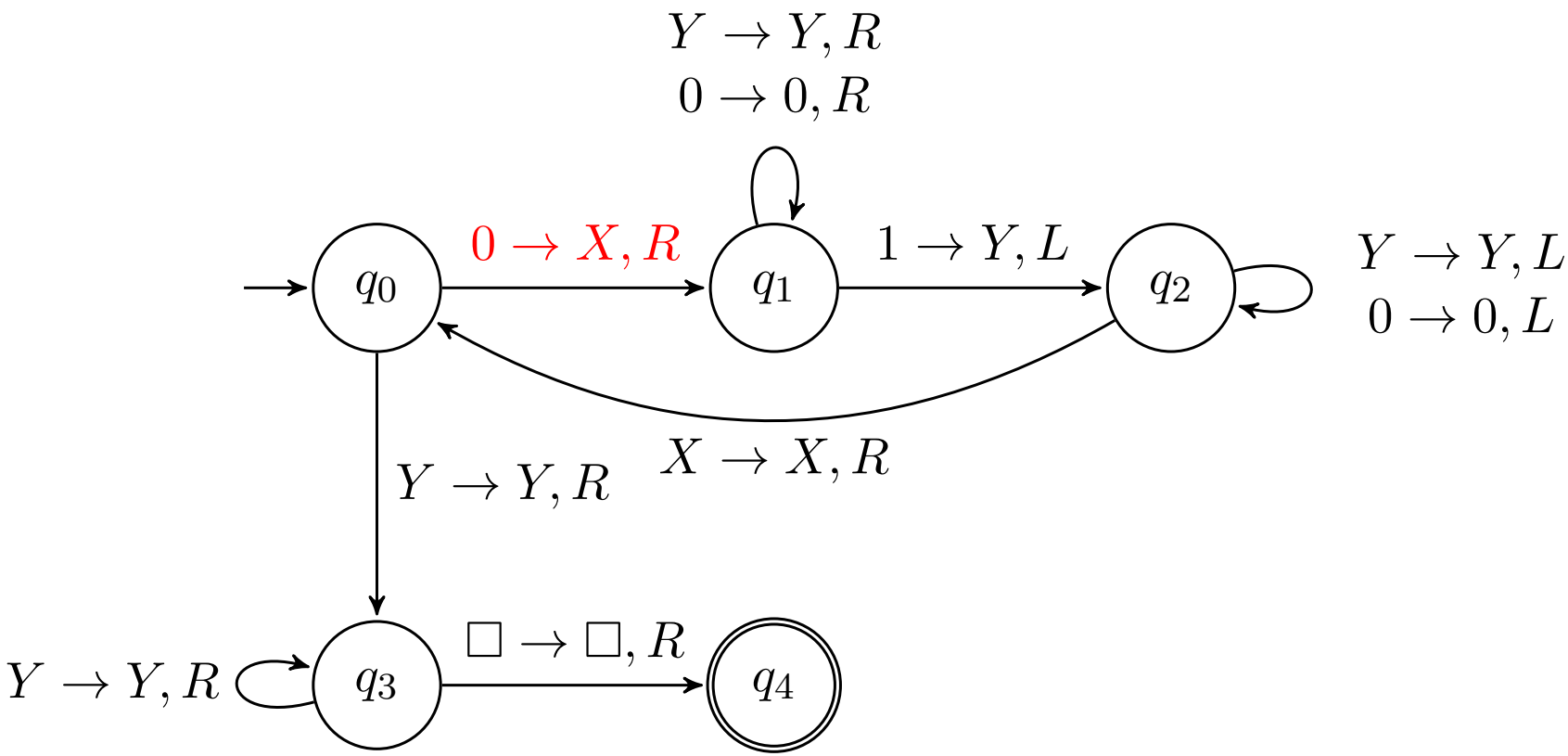




Example for 00001111

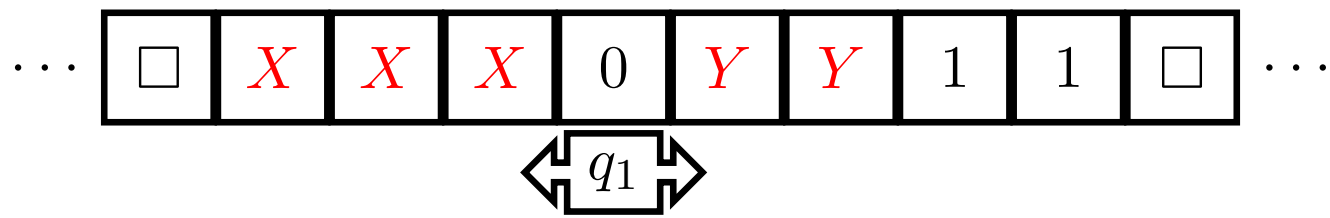
Formalities and examples

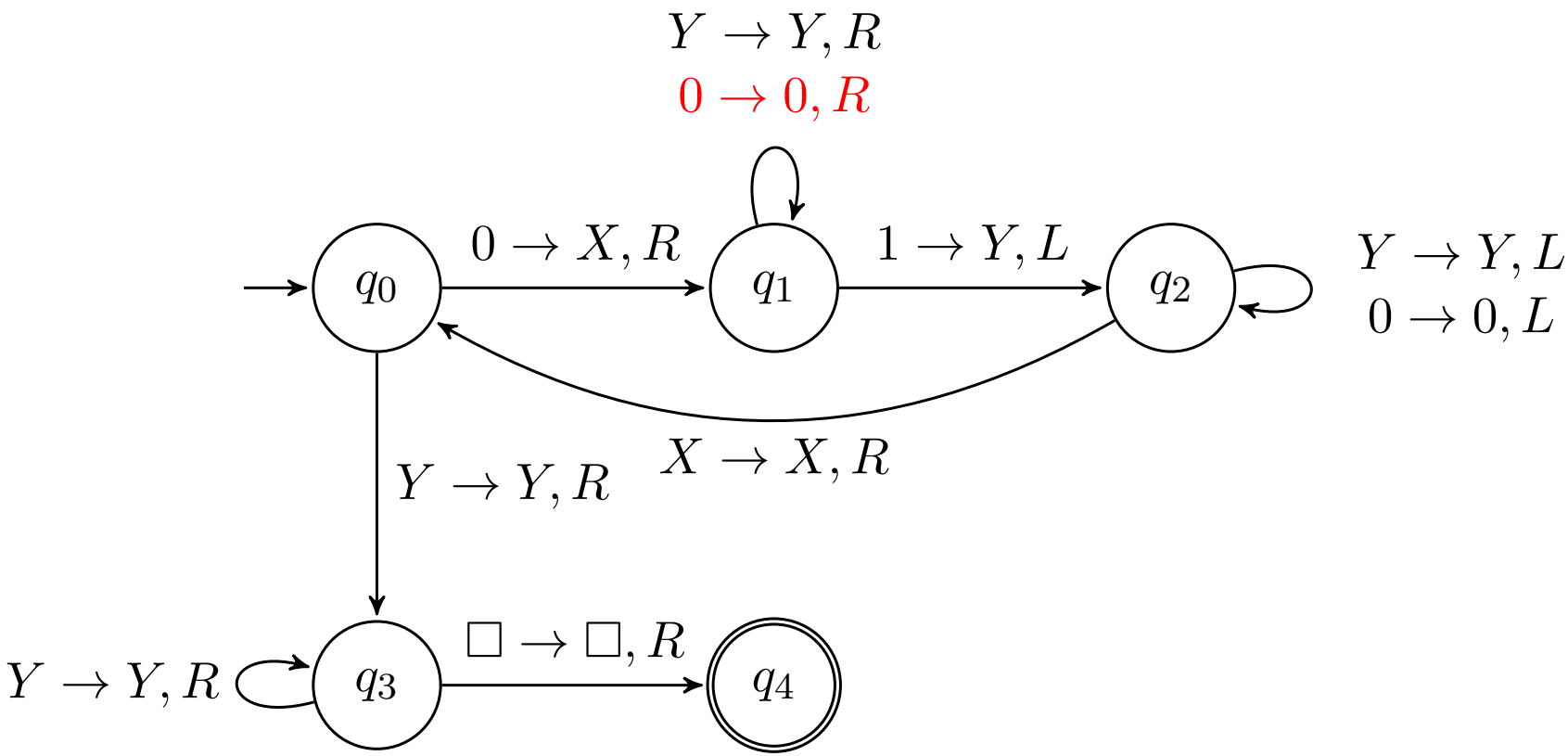




Example for 00001111

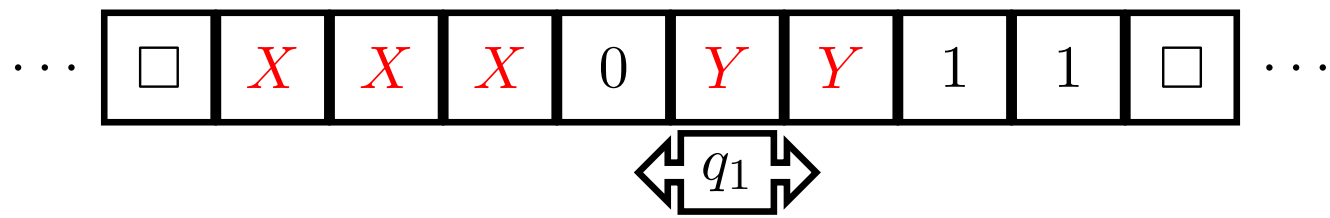
Formalities and examples

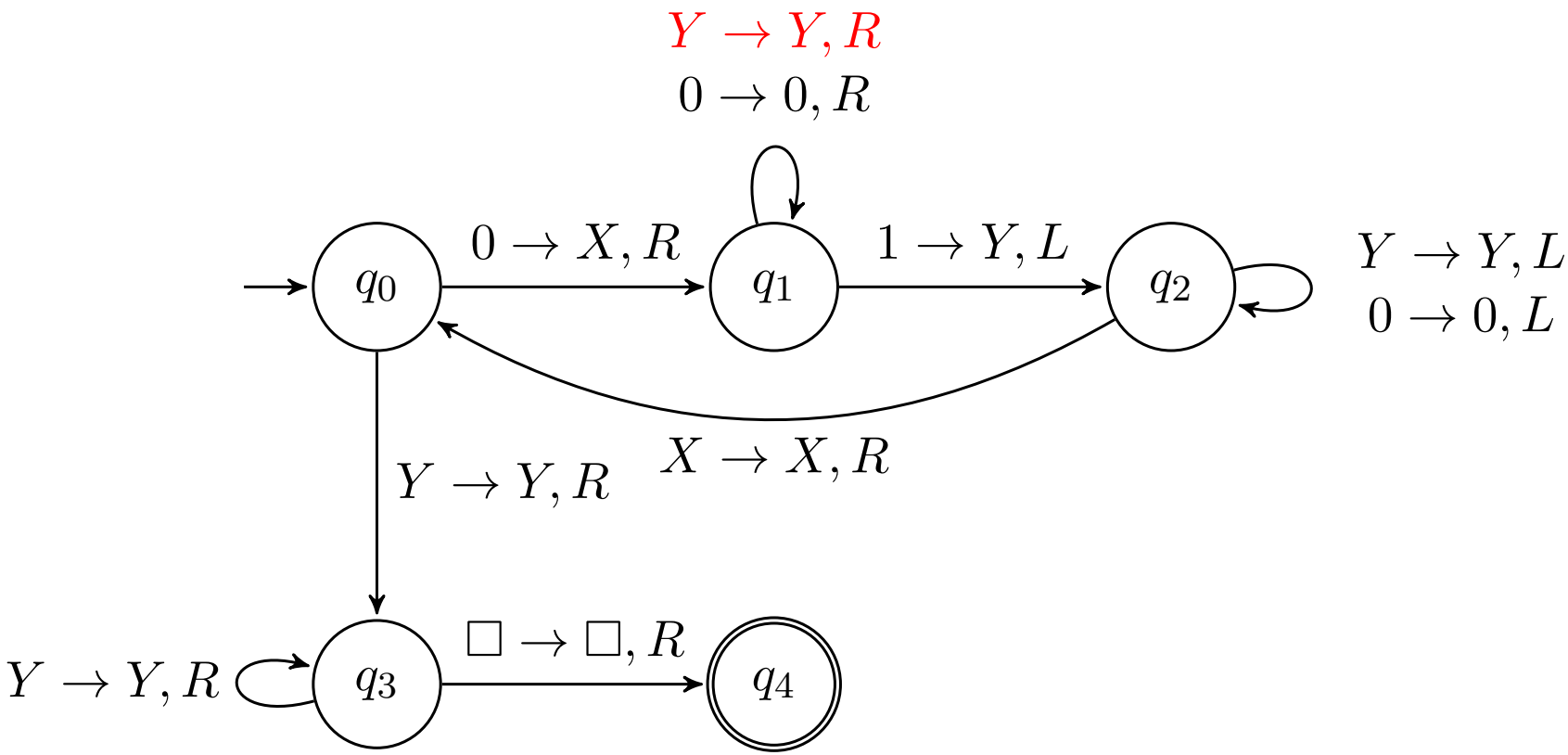




Example for 00001111

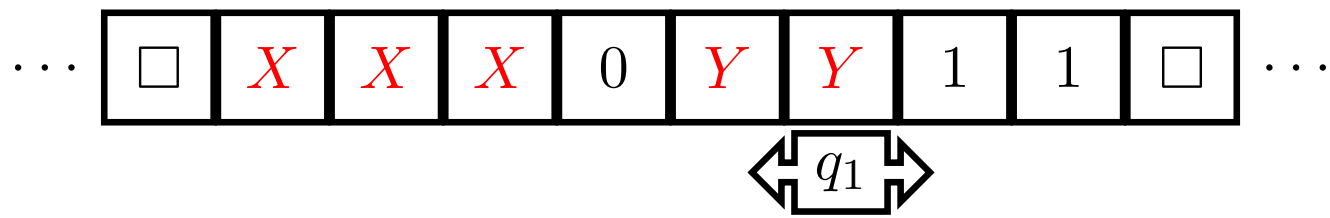
Formalities and examples

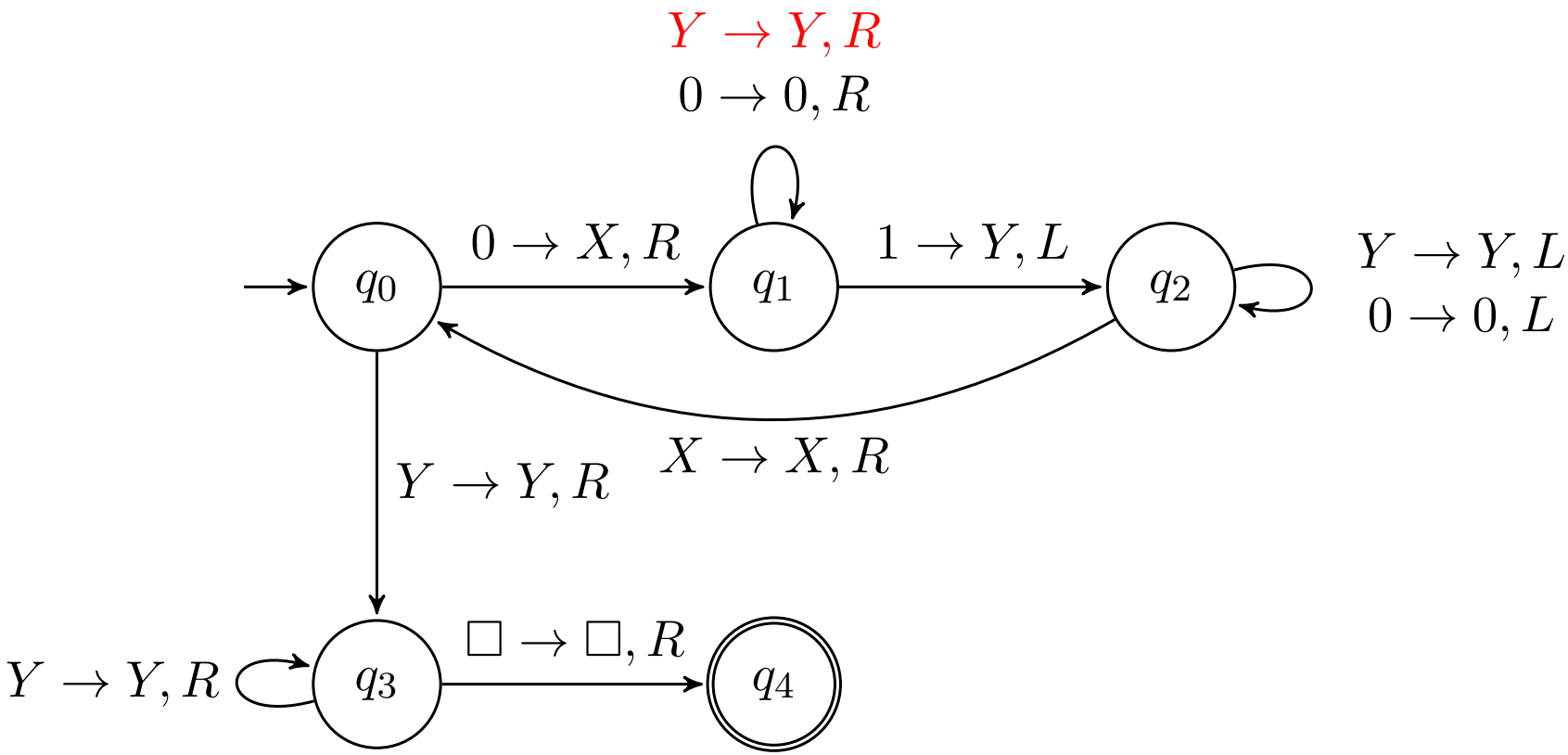




Example for 00001111

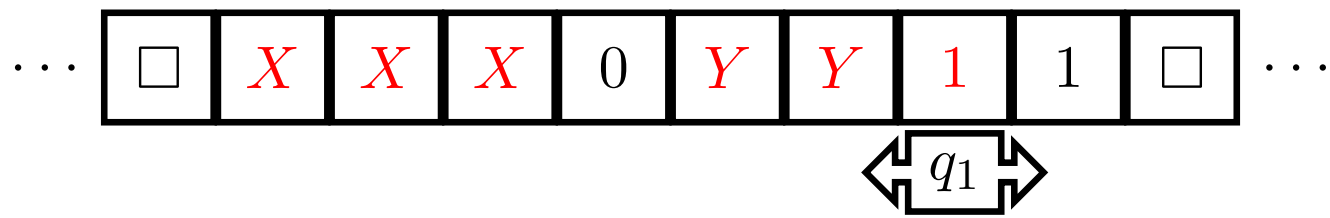
Formalities and examples

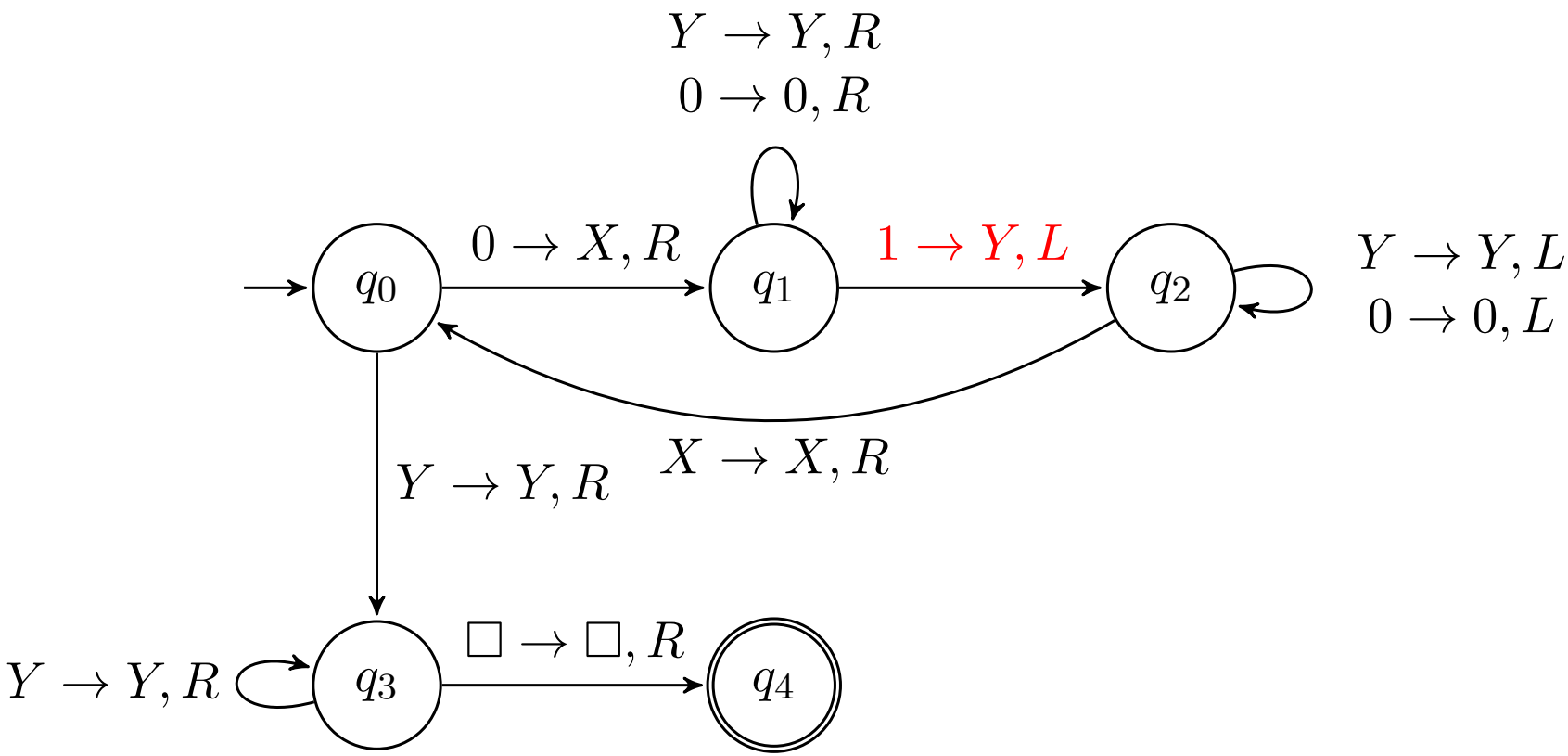




Example for 00001111

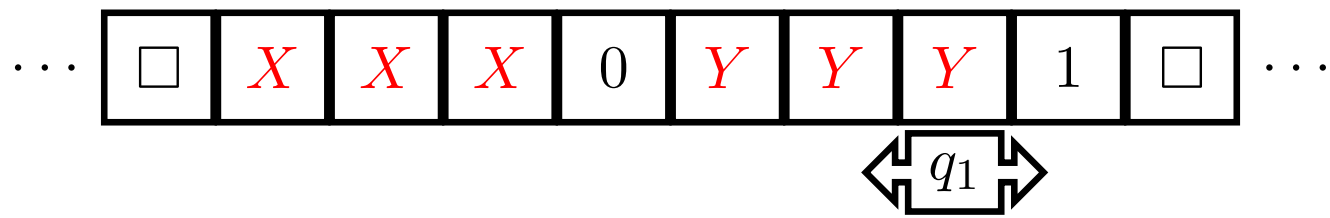
Formalities and examples

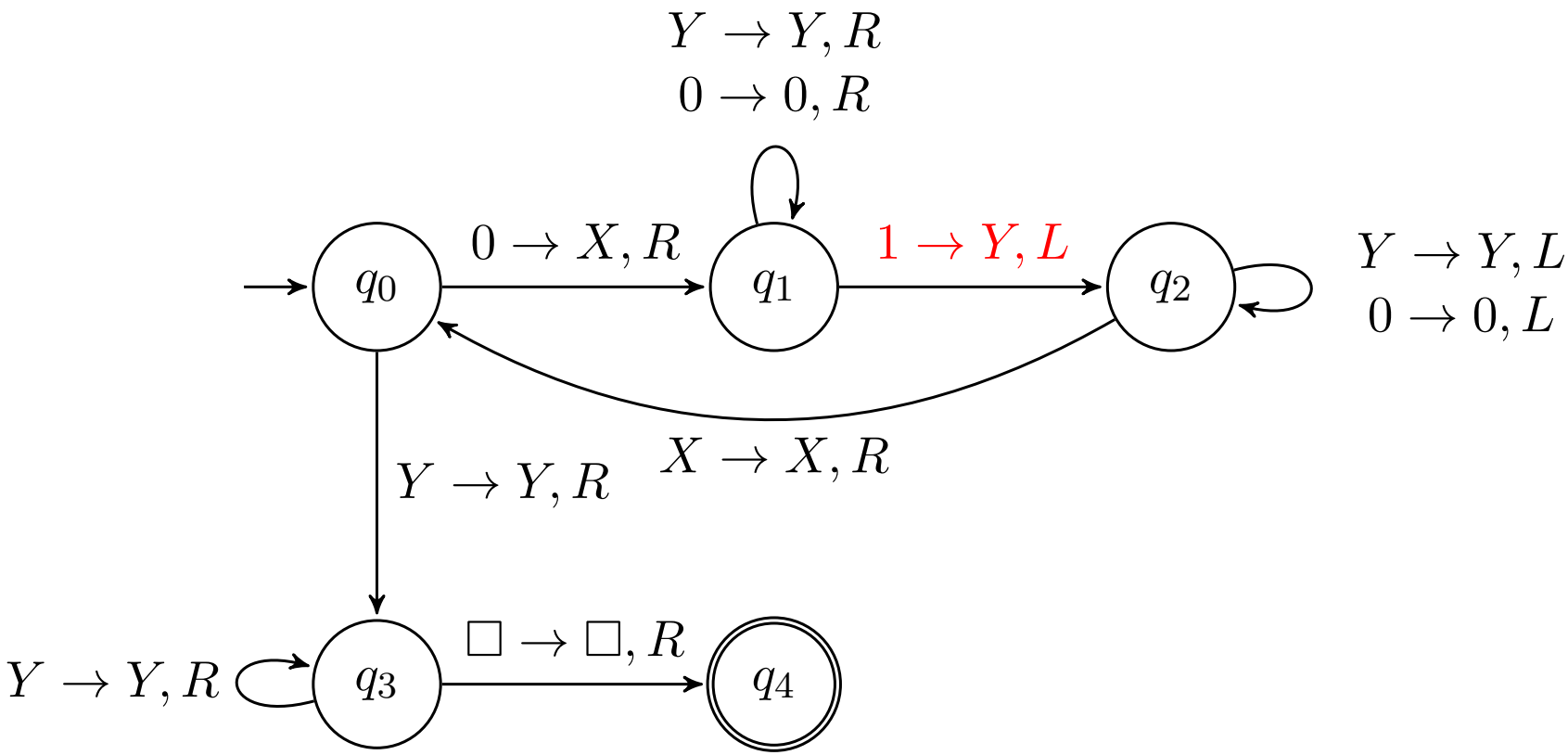




Example for 00001111

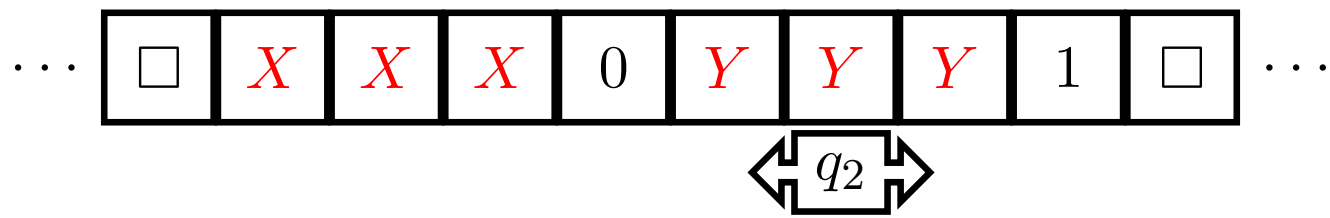
Formalities and examples

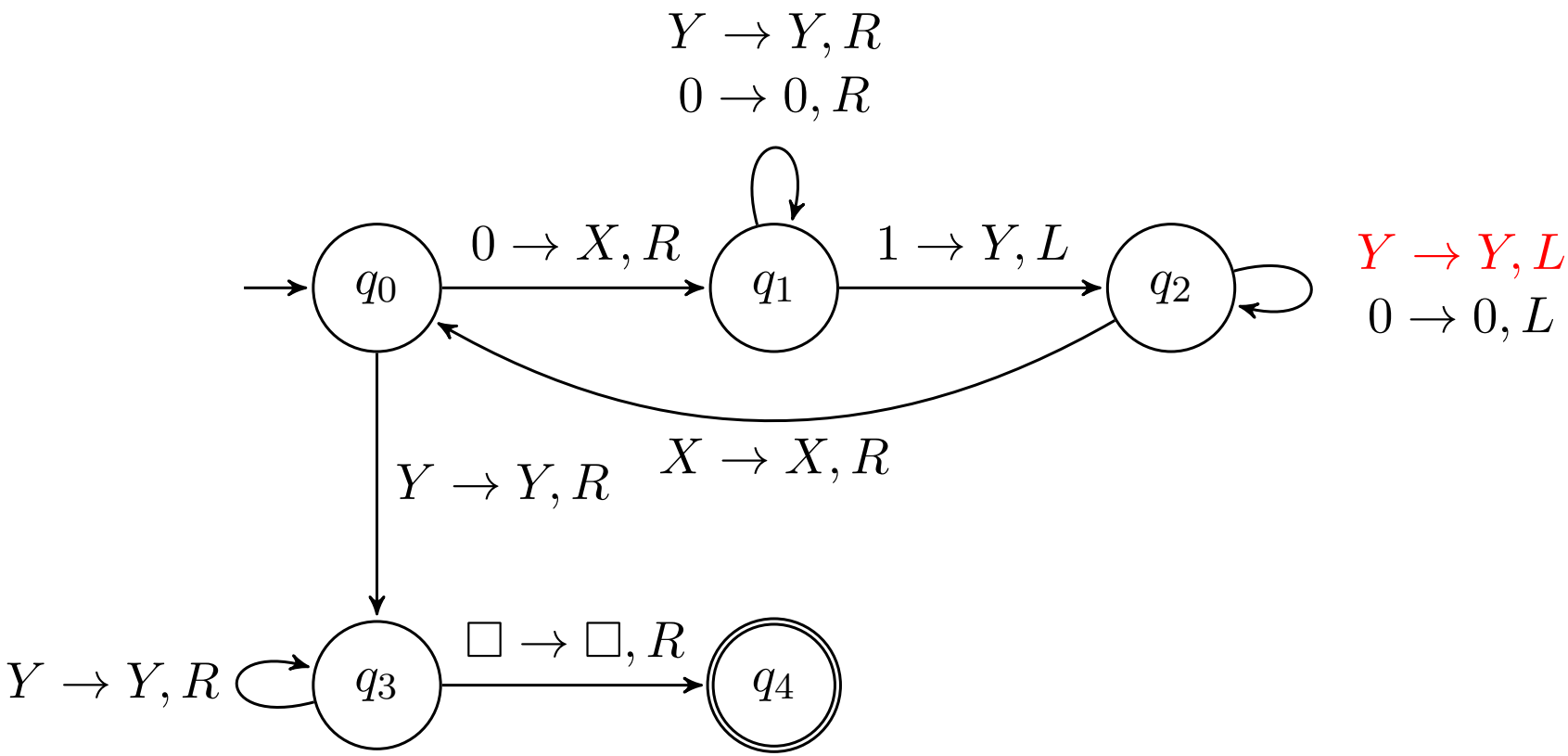




Example for 00001111

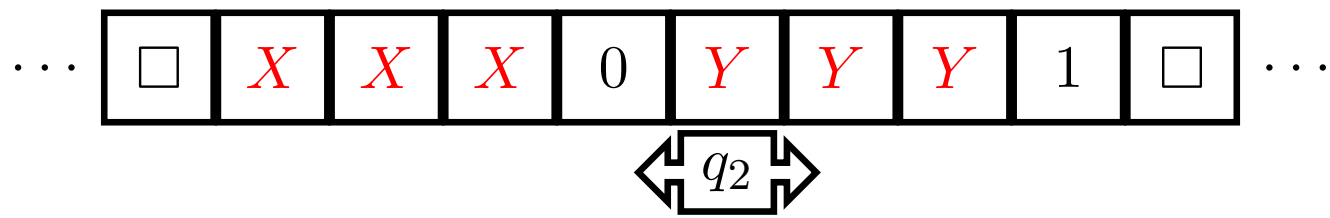
Formalities and examples

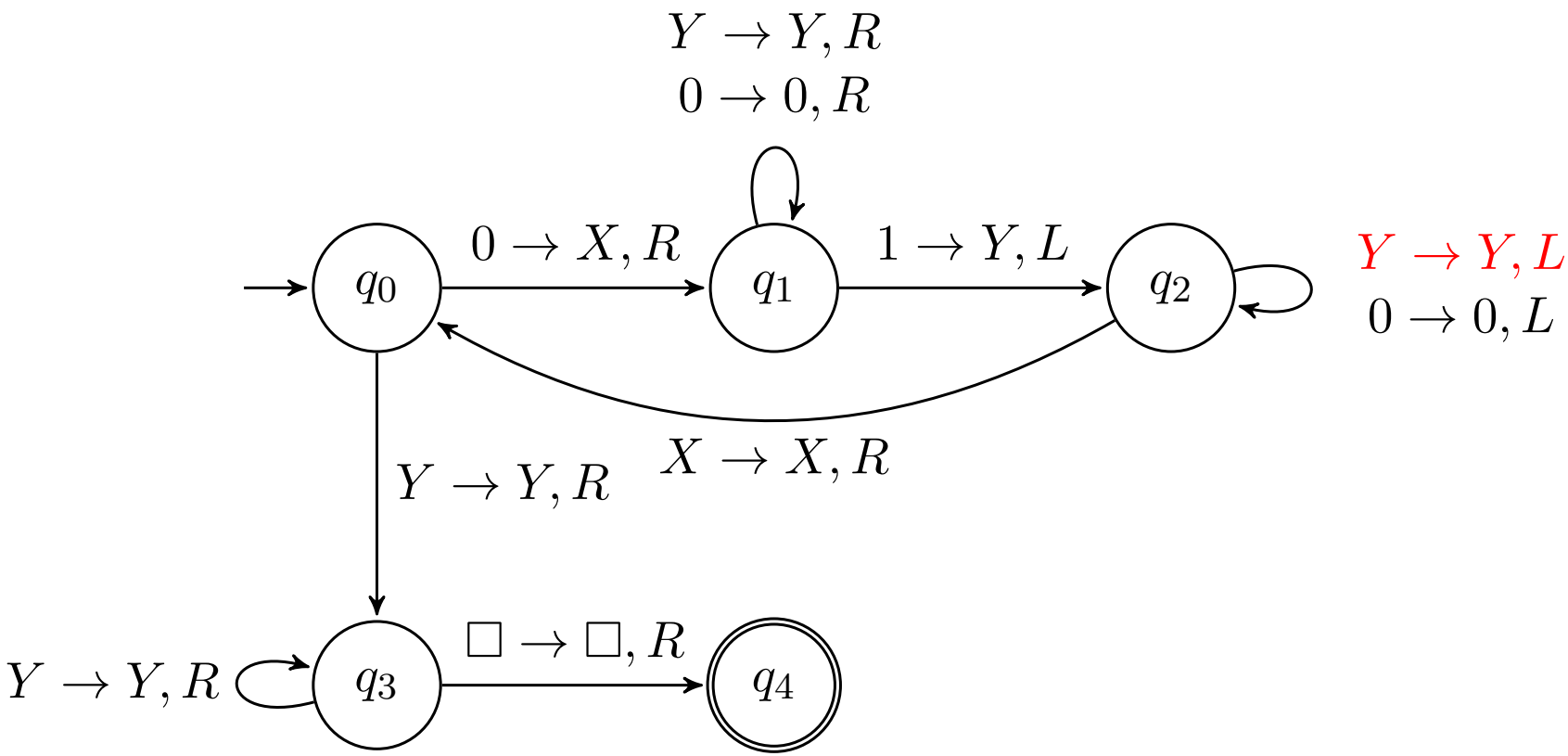




Example for 00001111

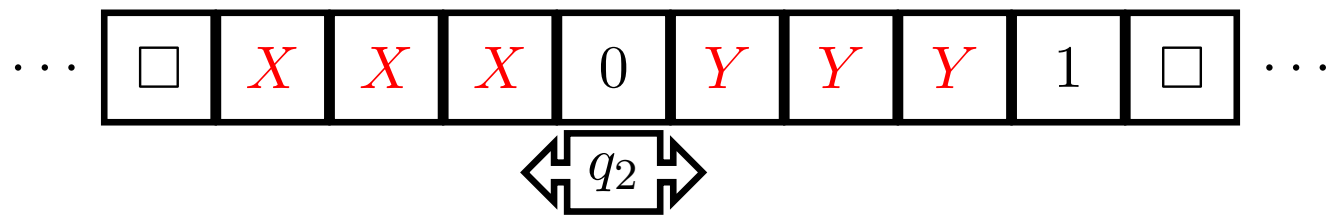
Formalities and examples

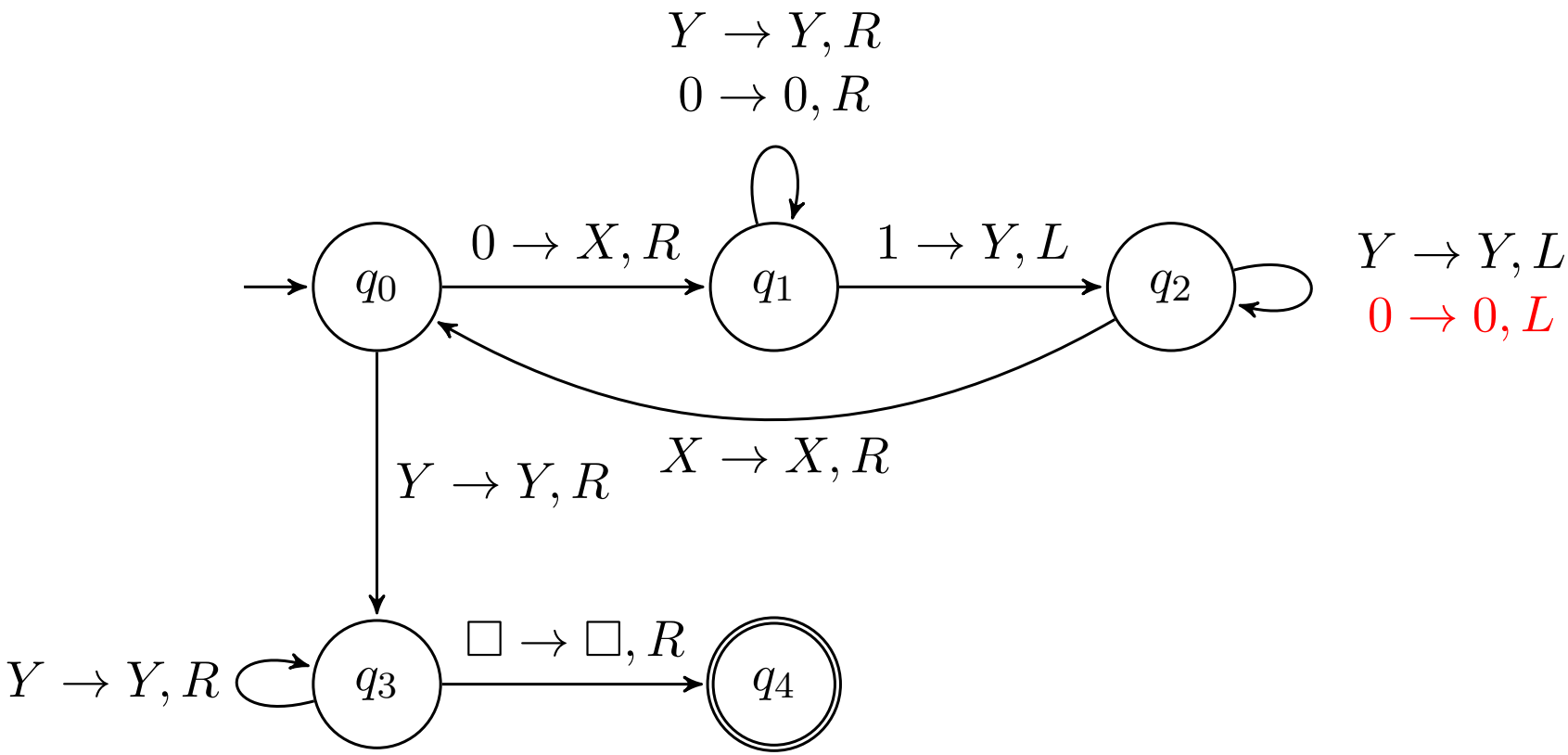




Example for 00001111

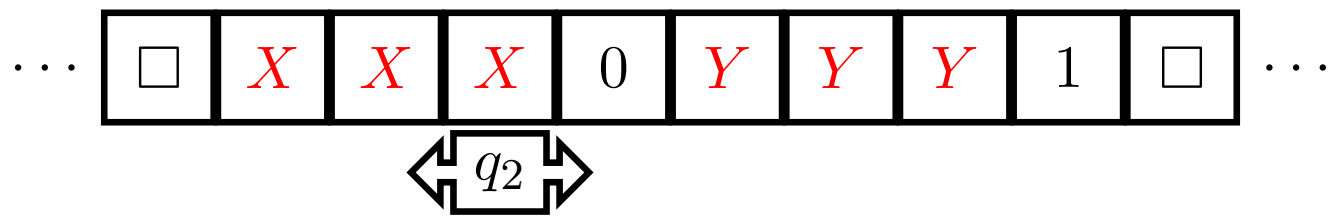
Formalities and examples

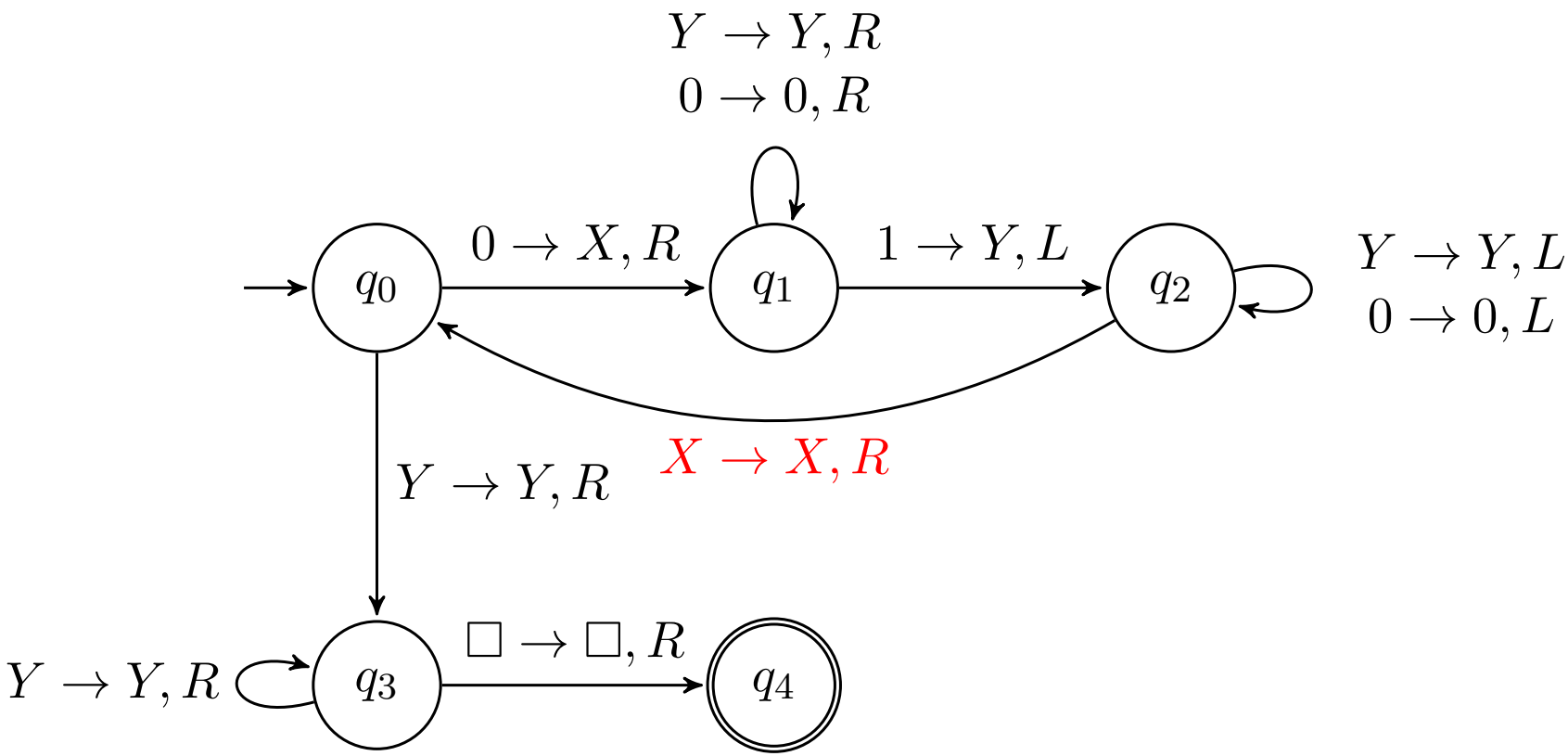




Example for 00001111

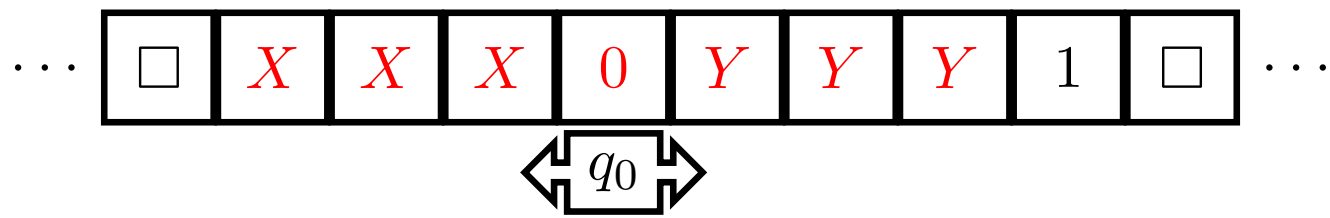
Formalities and examples

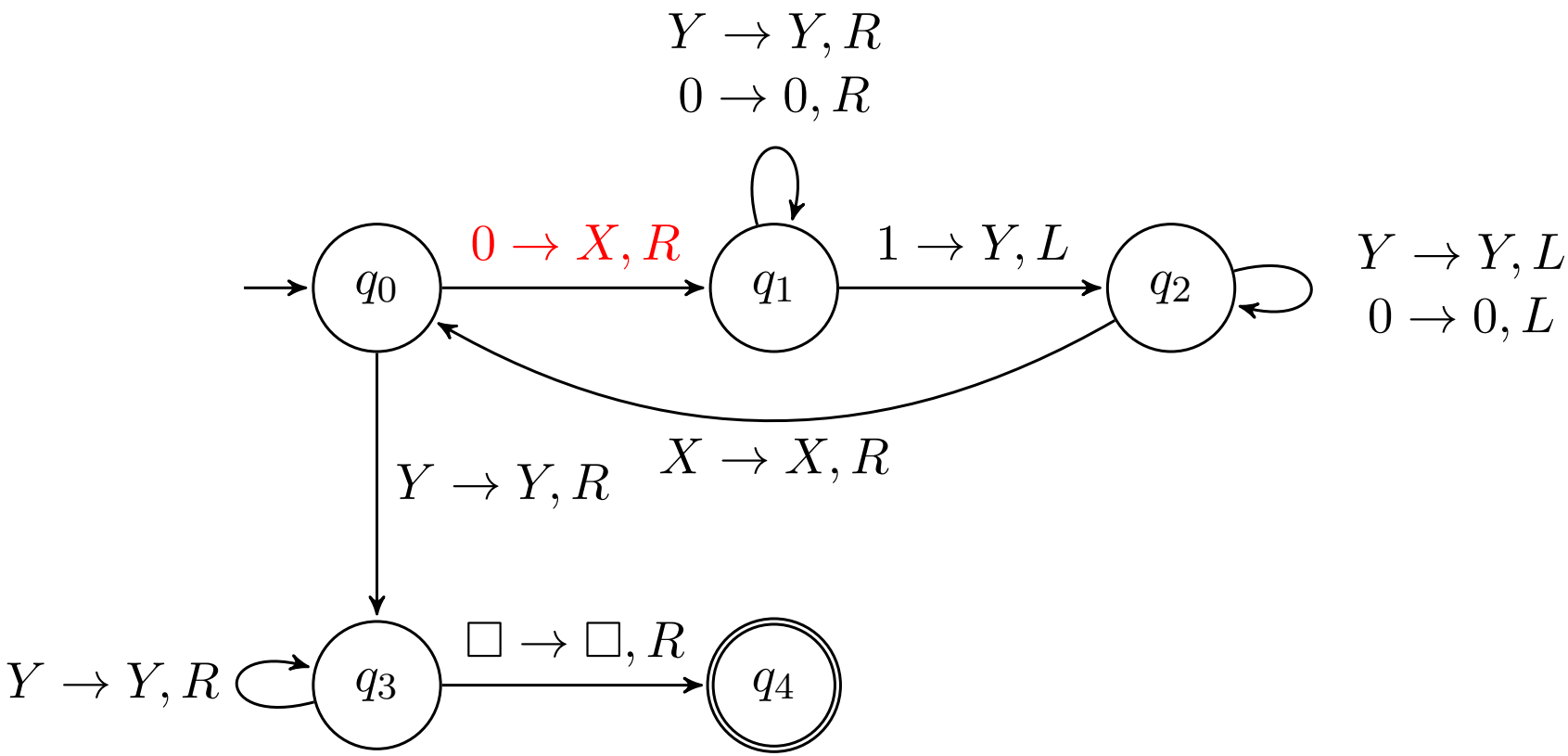




Example for 00001111

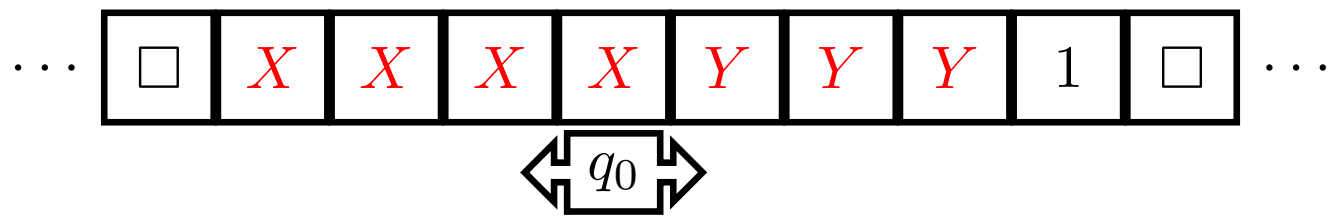
Formalities and examples

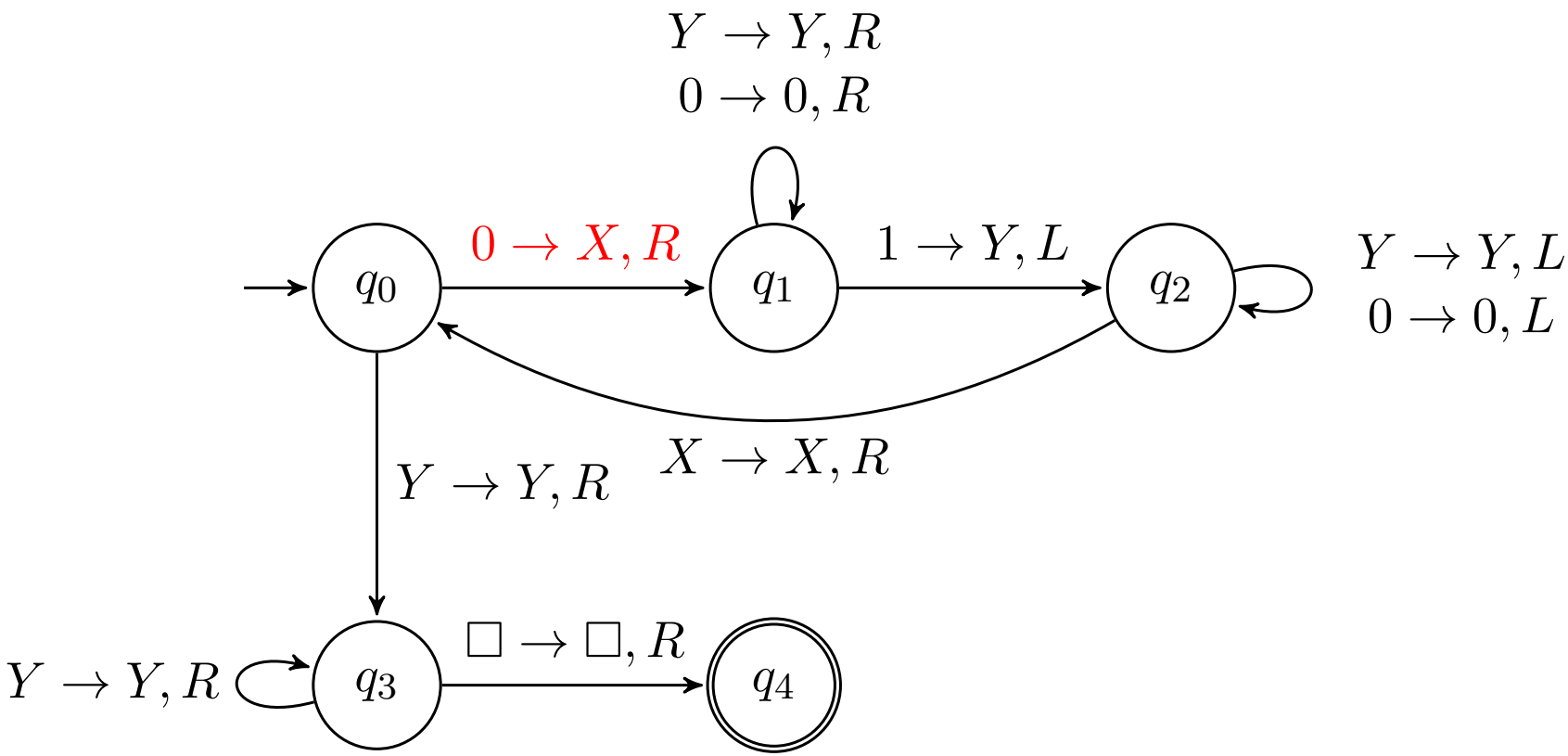




Example for 00001111

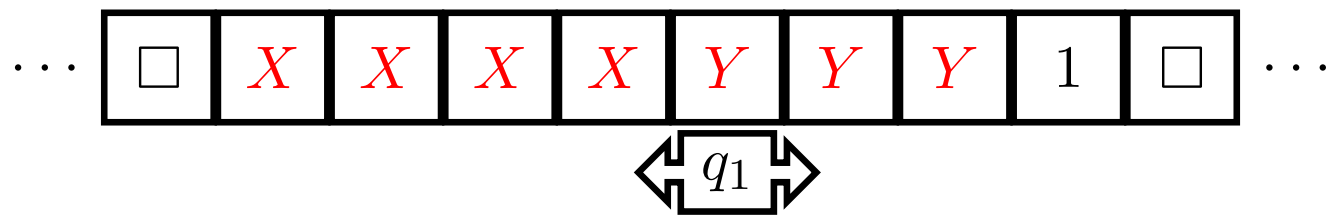
Formalities and examples

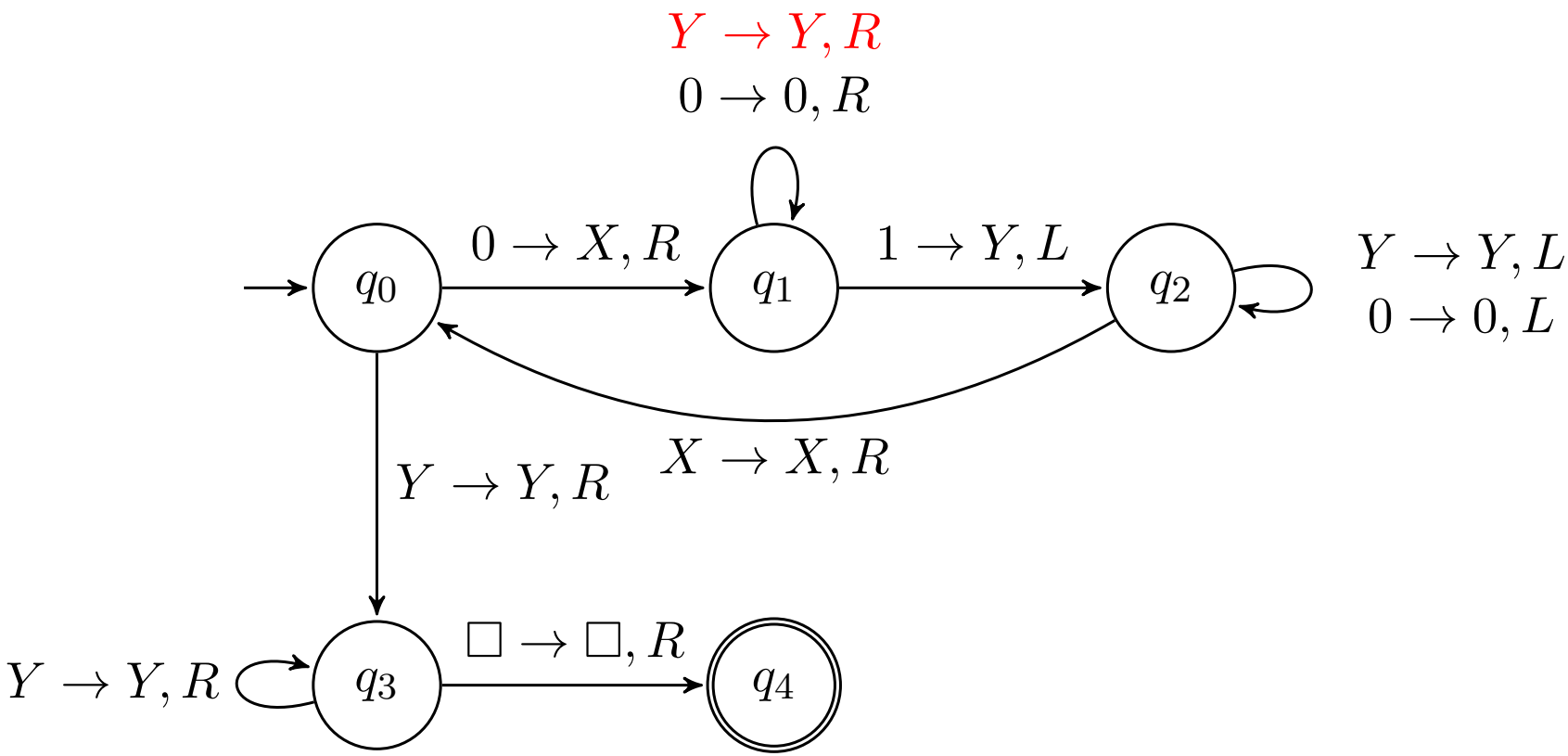




Example for 00001111

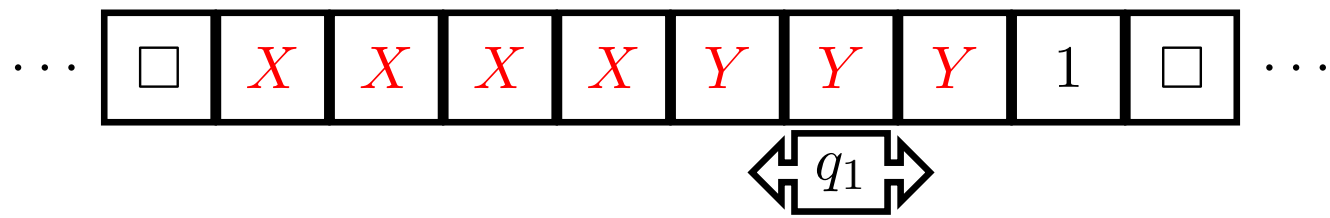
Formalities and examples

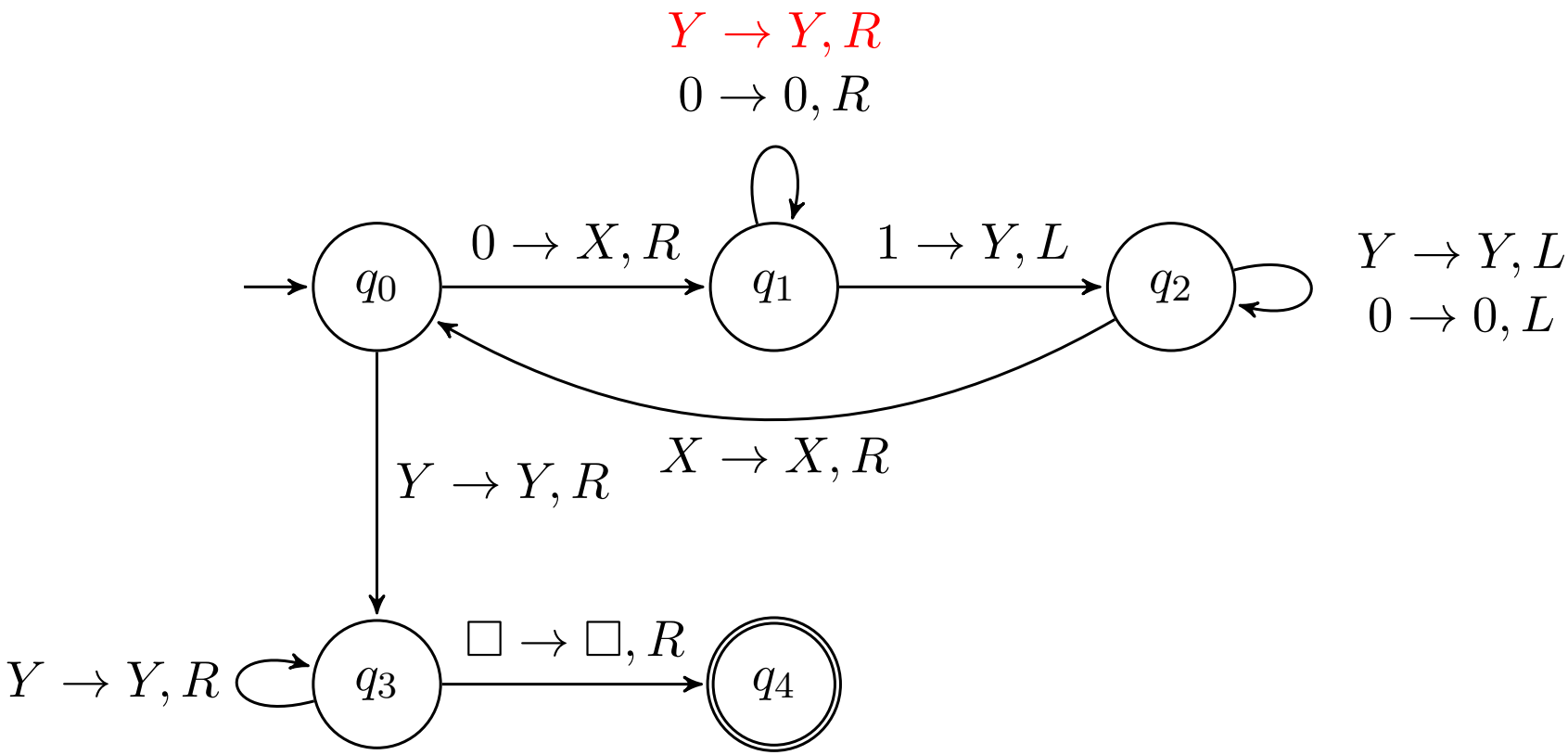




Example for 00001111

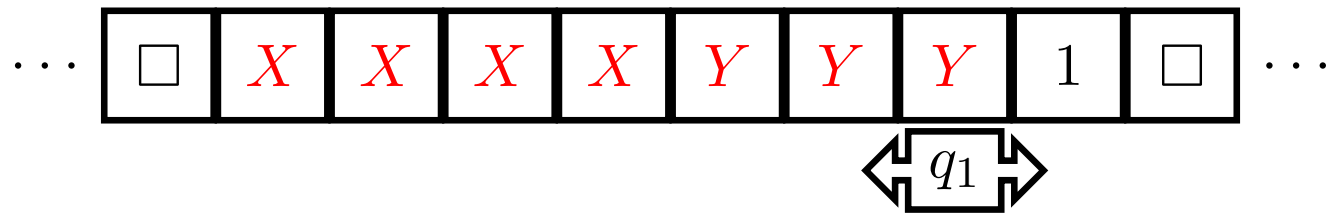
Formalities and examples

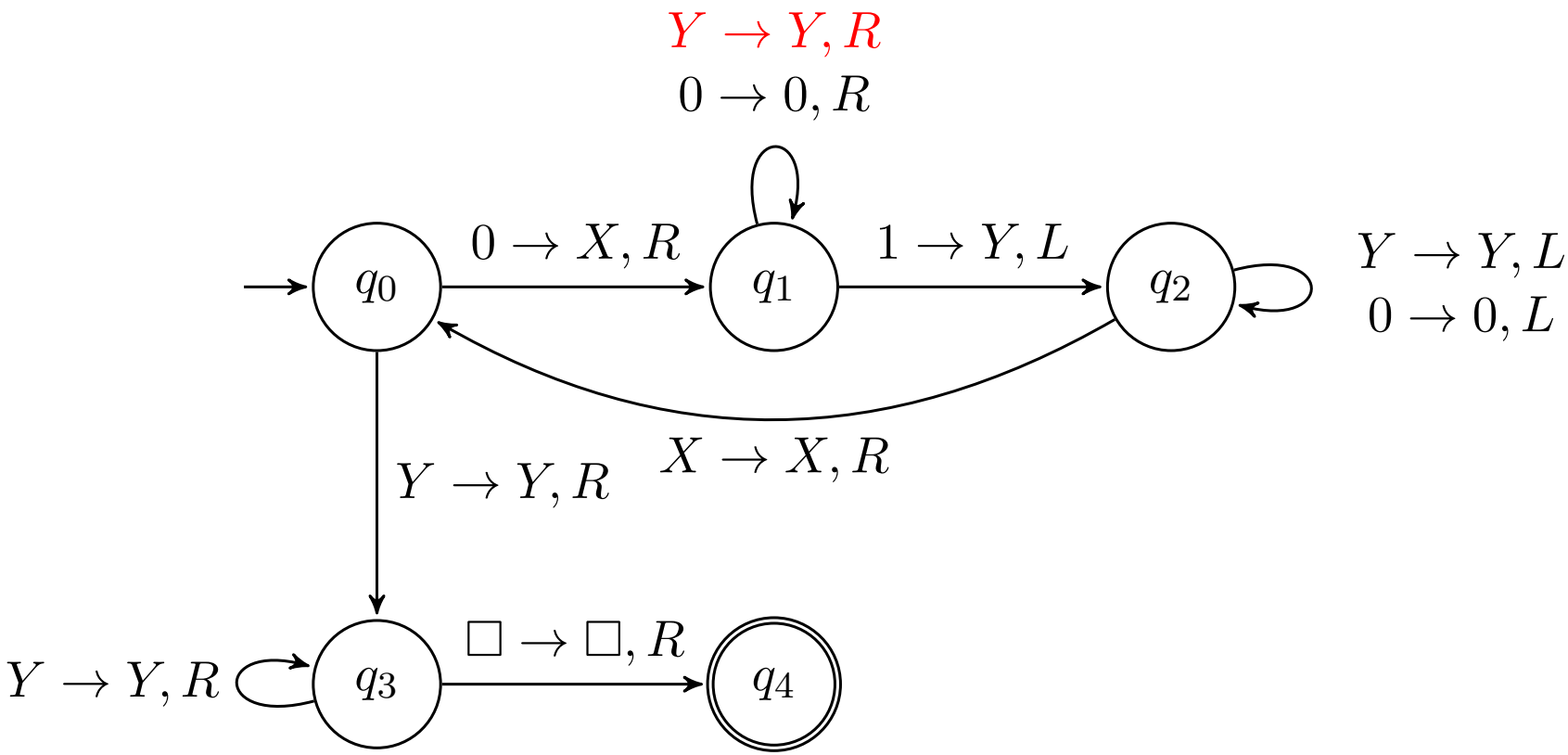




Example for 00001111

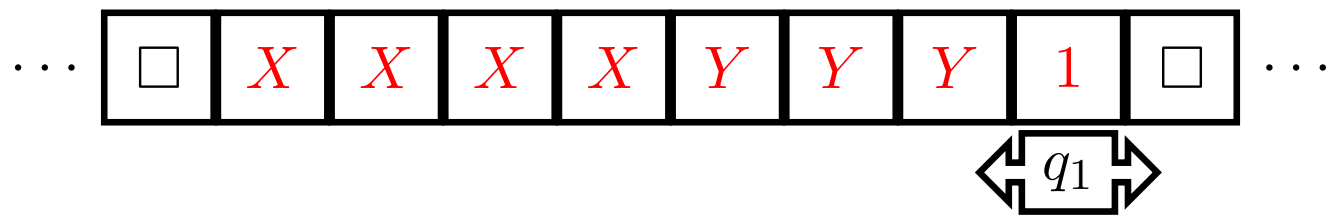
Formalities and examples

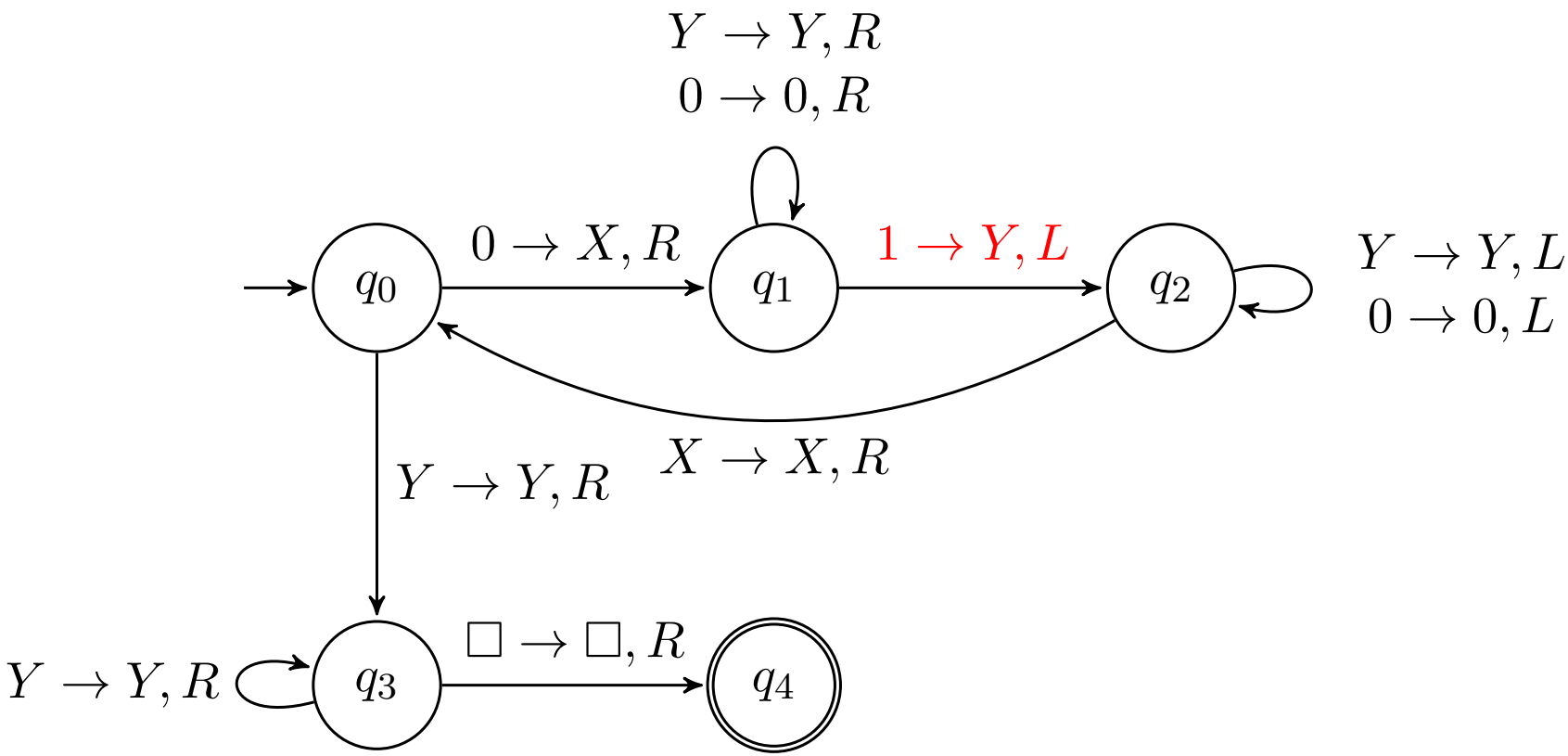




Example for 00001111

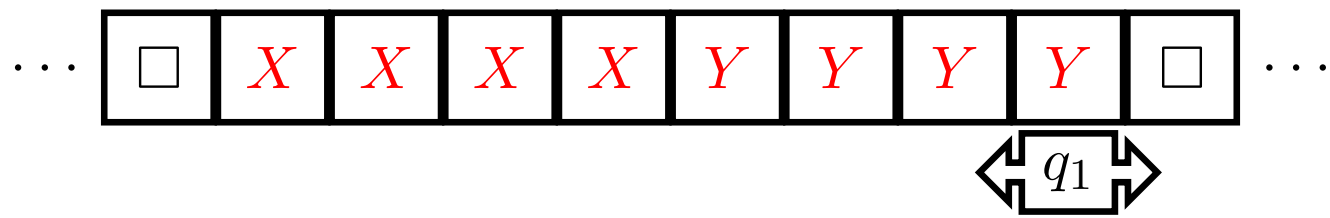
Formalities and examples

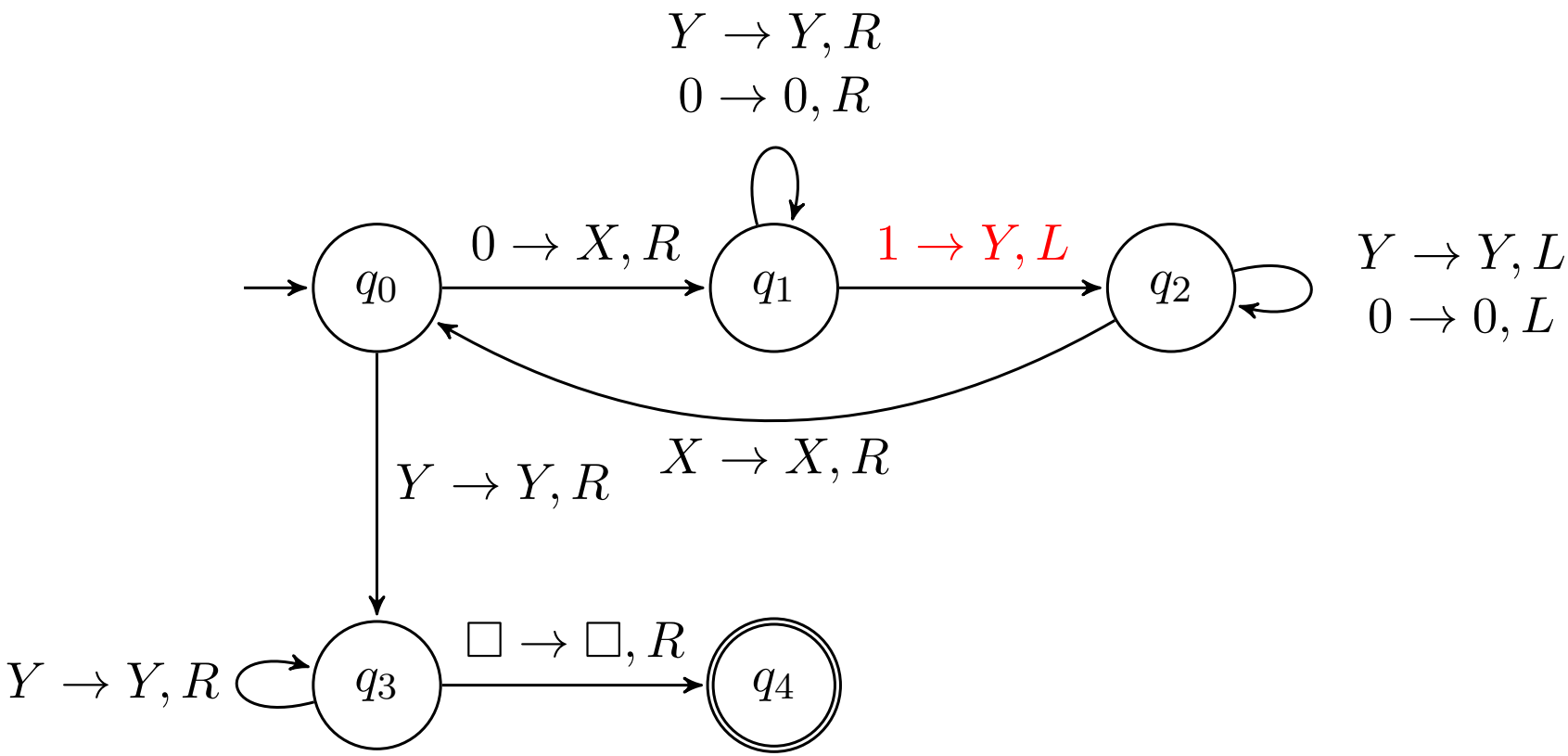




Example for 00001111

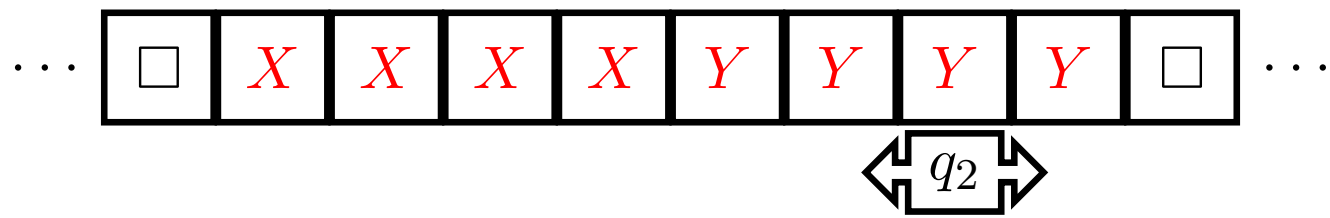
Formalities and examples

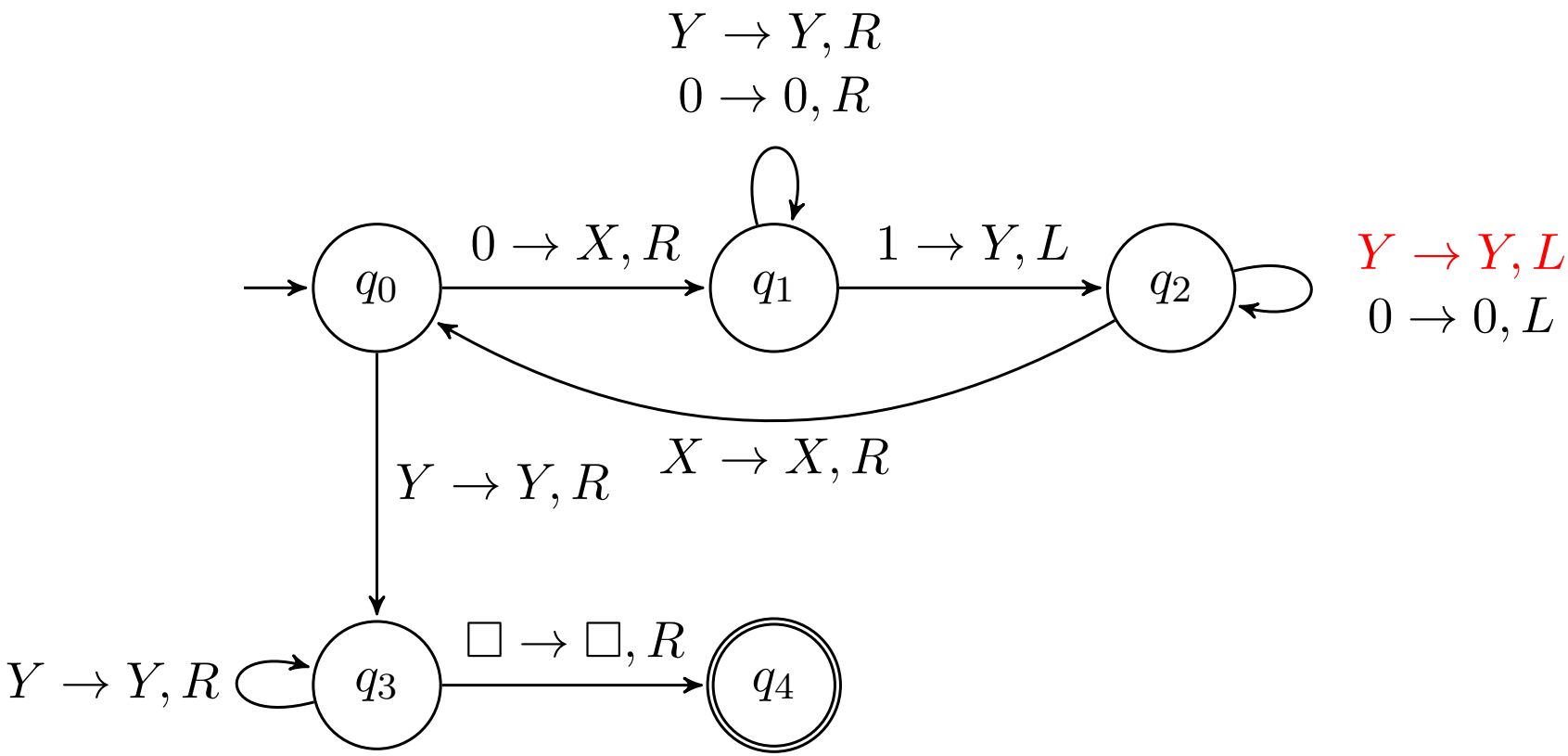




Example for 00001111

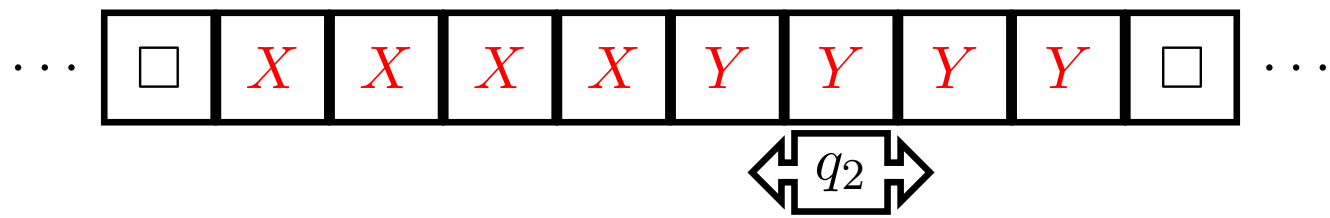
Formalities and examples

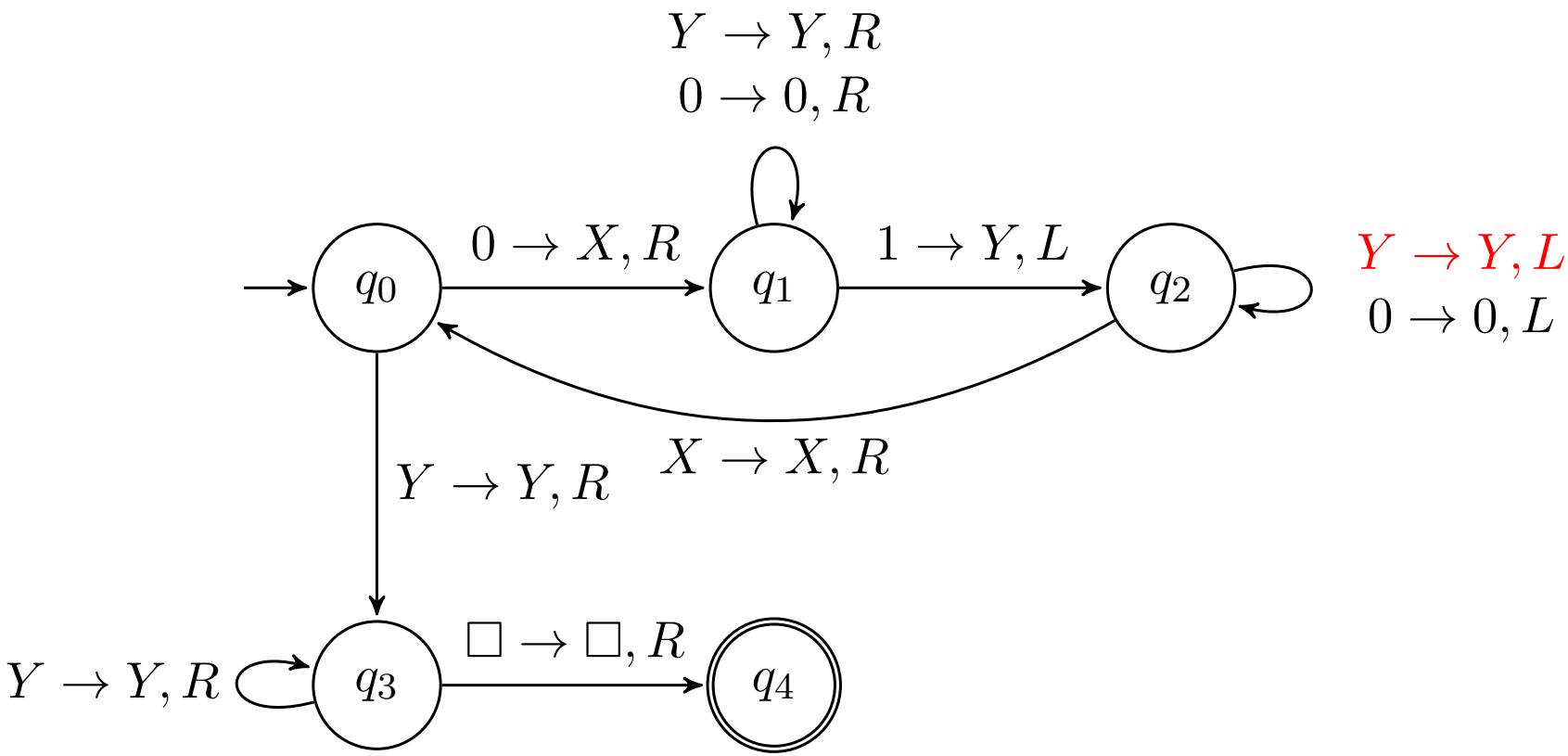




Example for 00001111

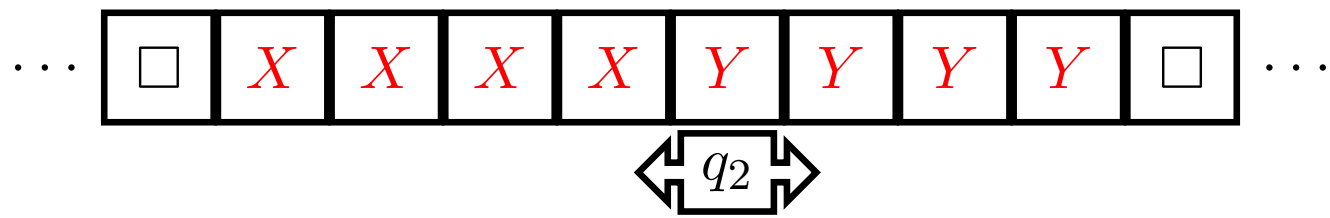
Formalities and examples

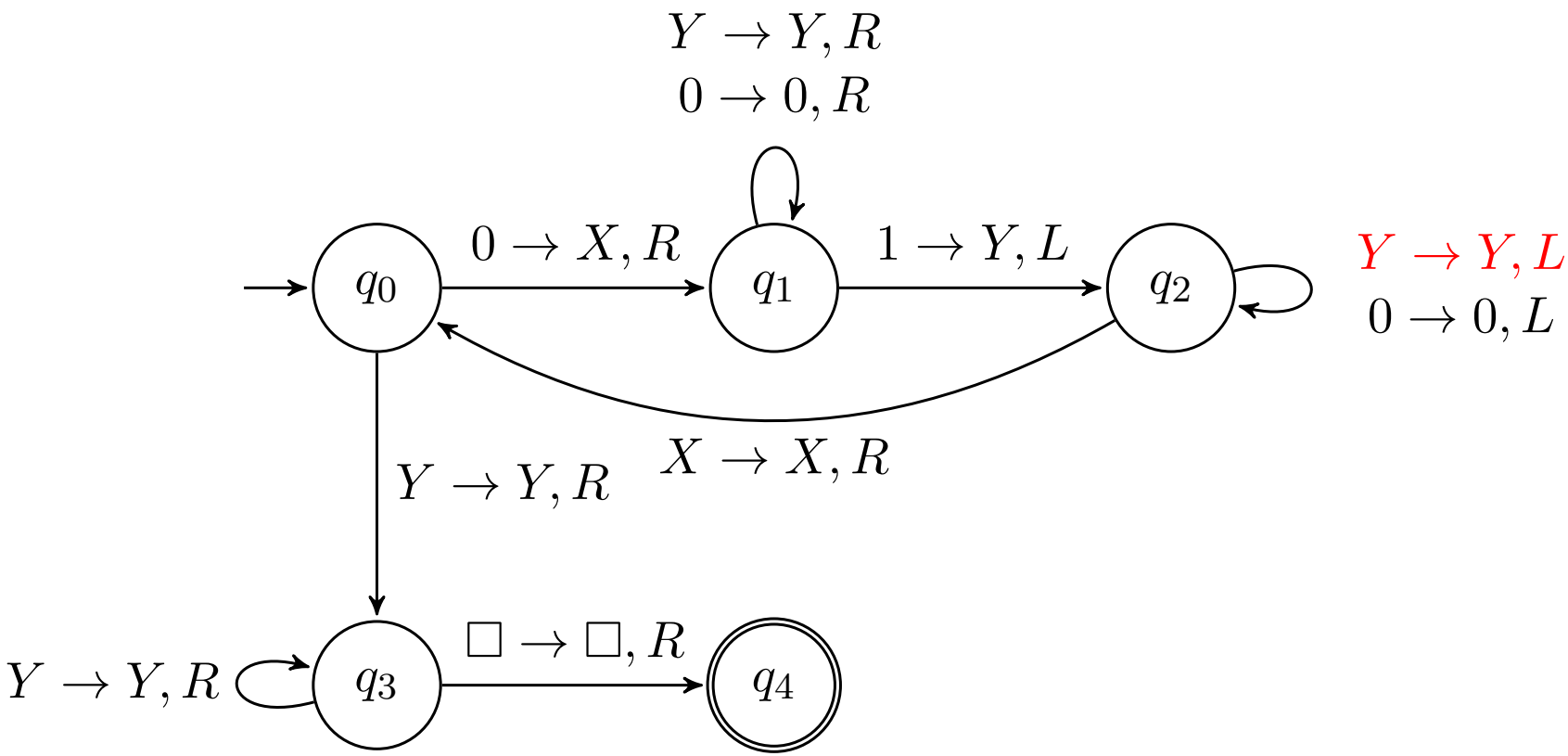




Example for 00001111

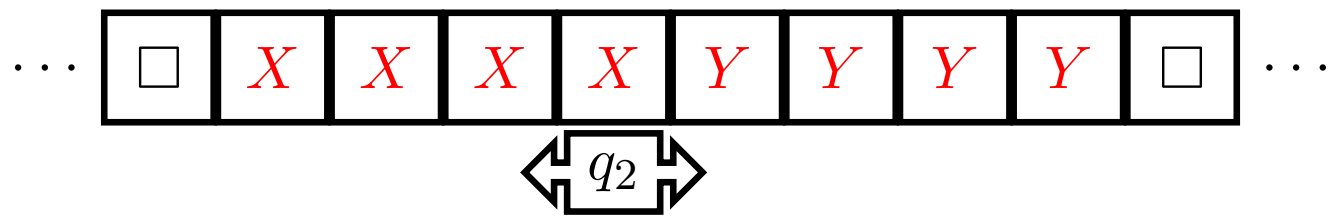
Formalities and examples

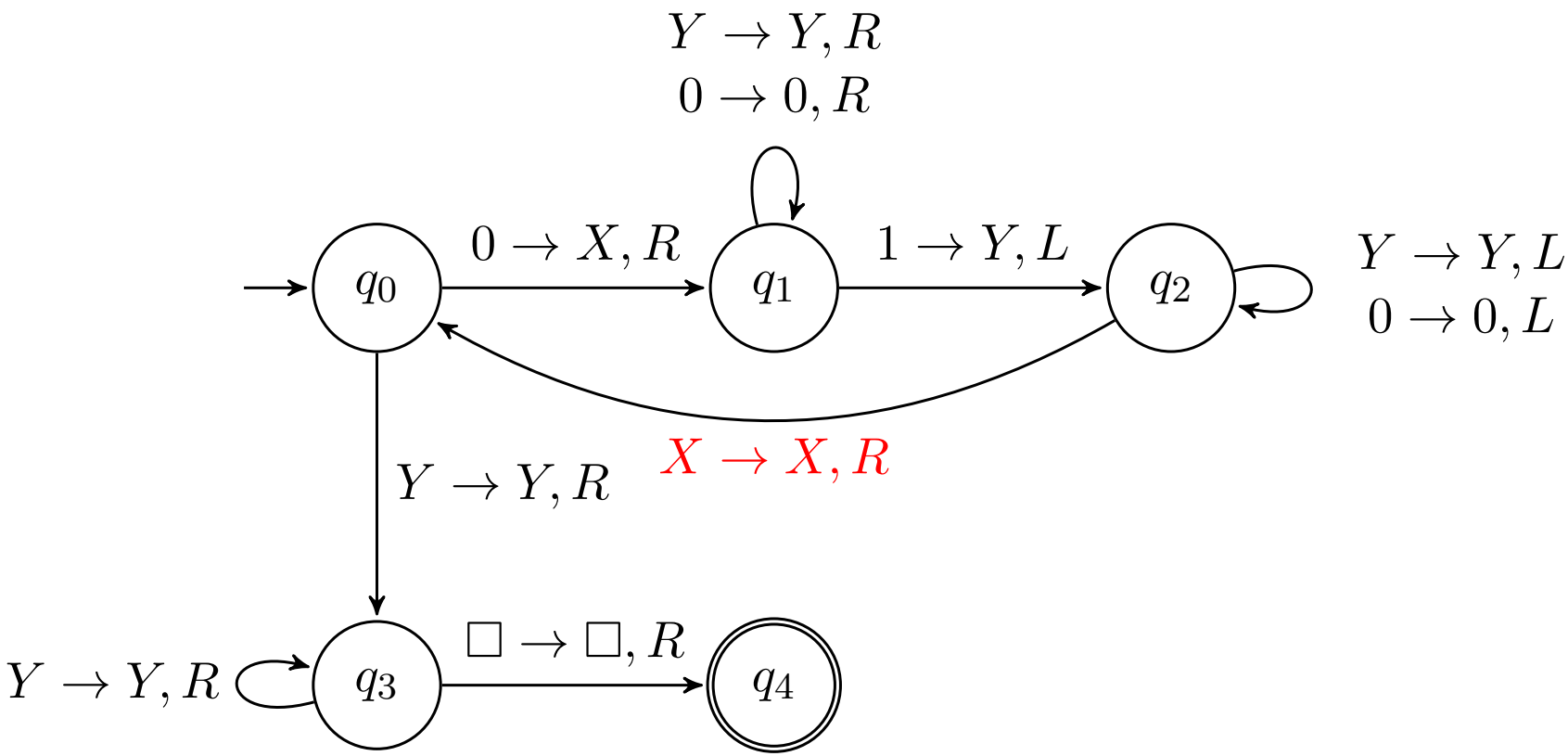




Example for 00001111

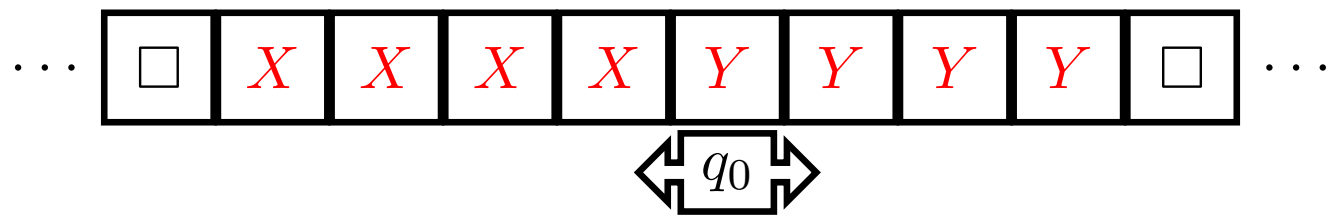
Formalities and examples

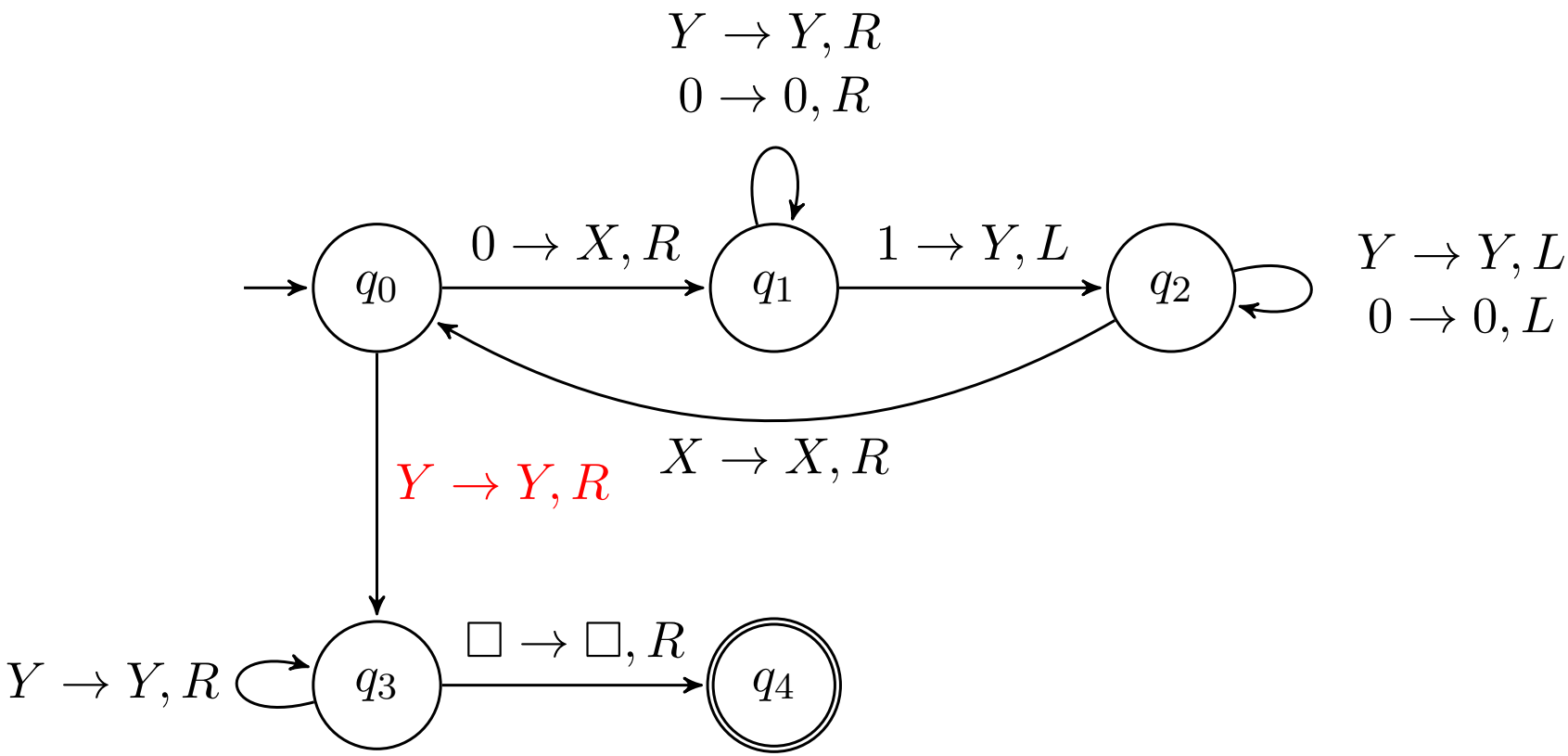




Example for 00001111

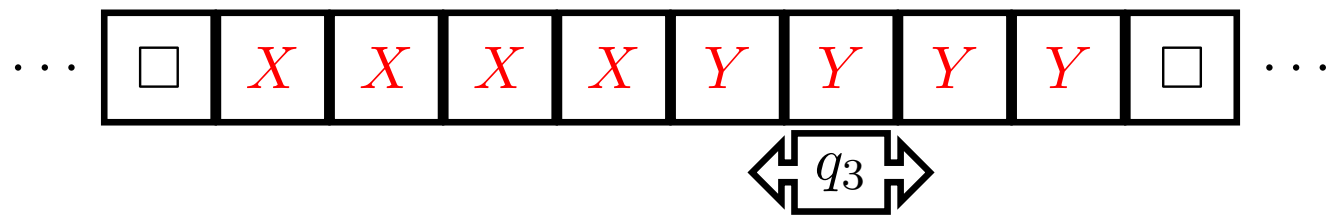
Formalities and examples

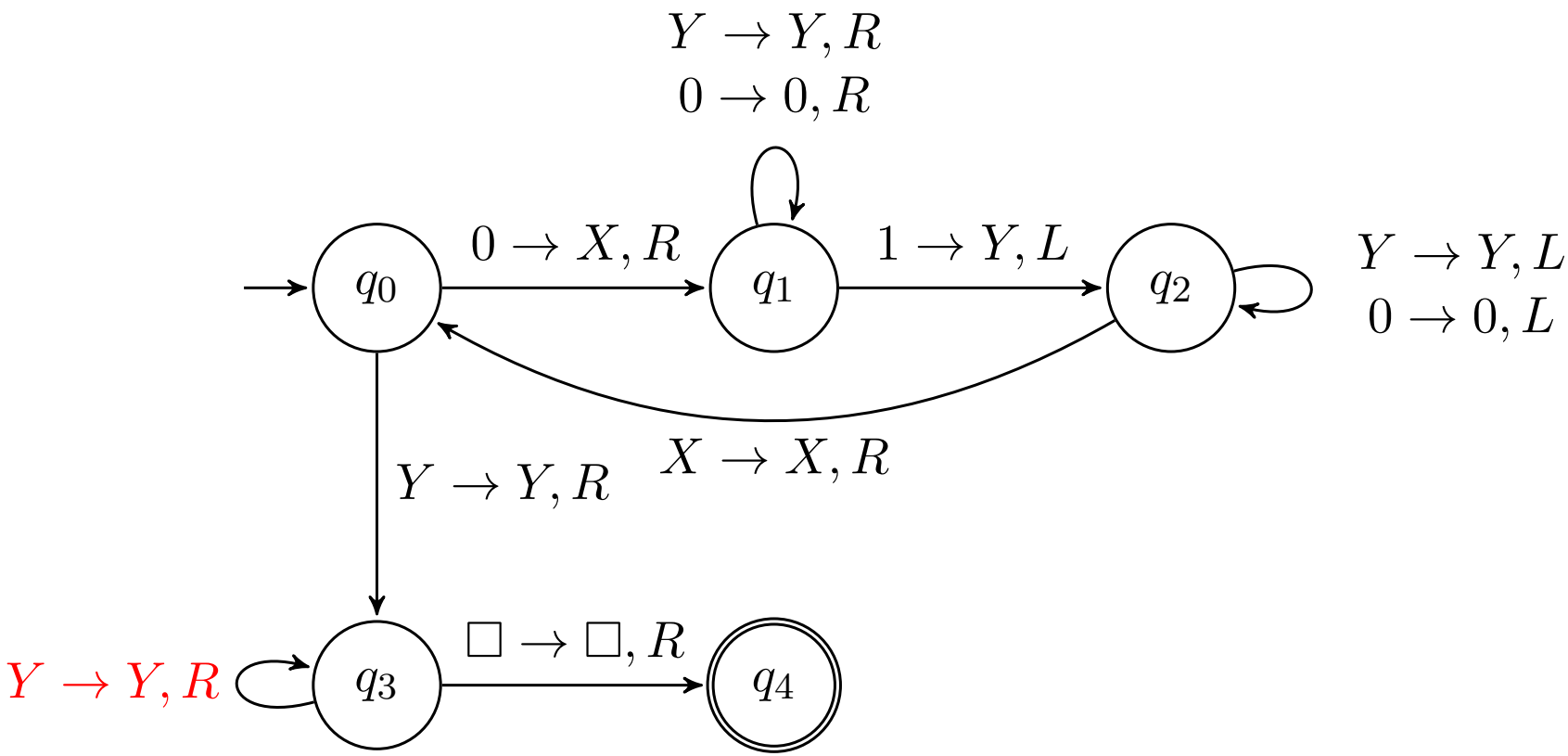




Example for 00001111

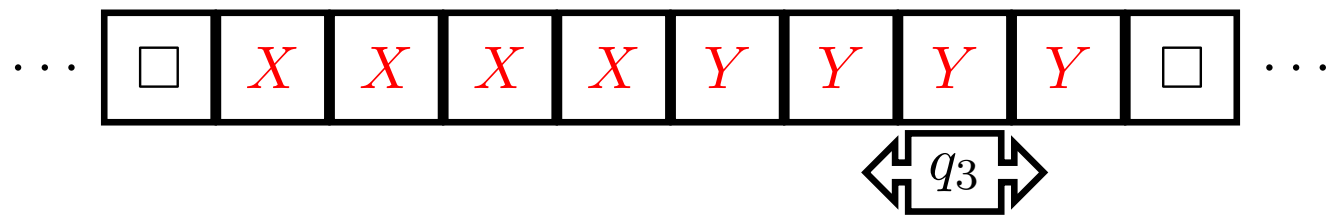
Formalities and examples

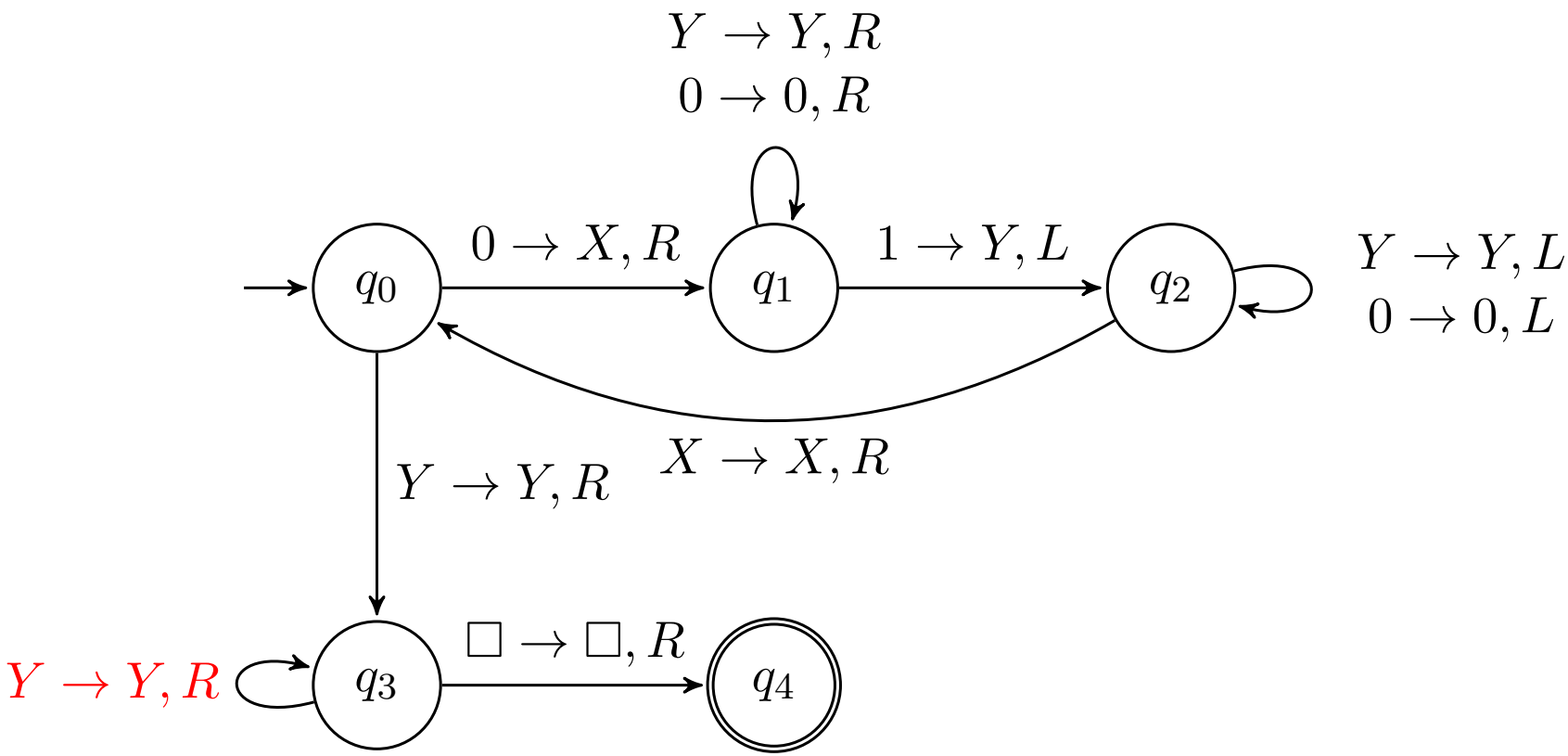




Example for 00001111

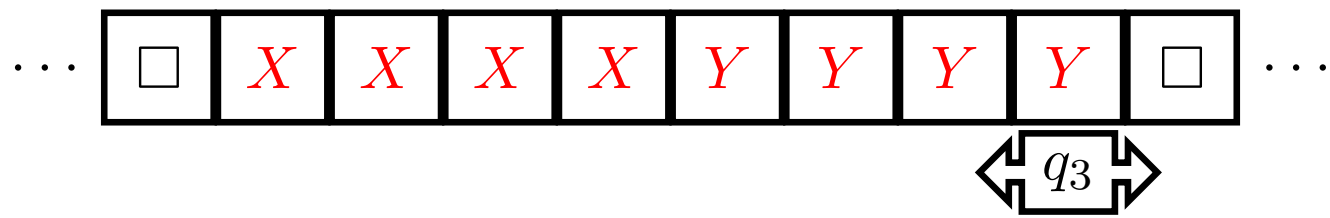
Formalities and examples

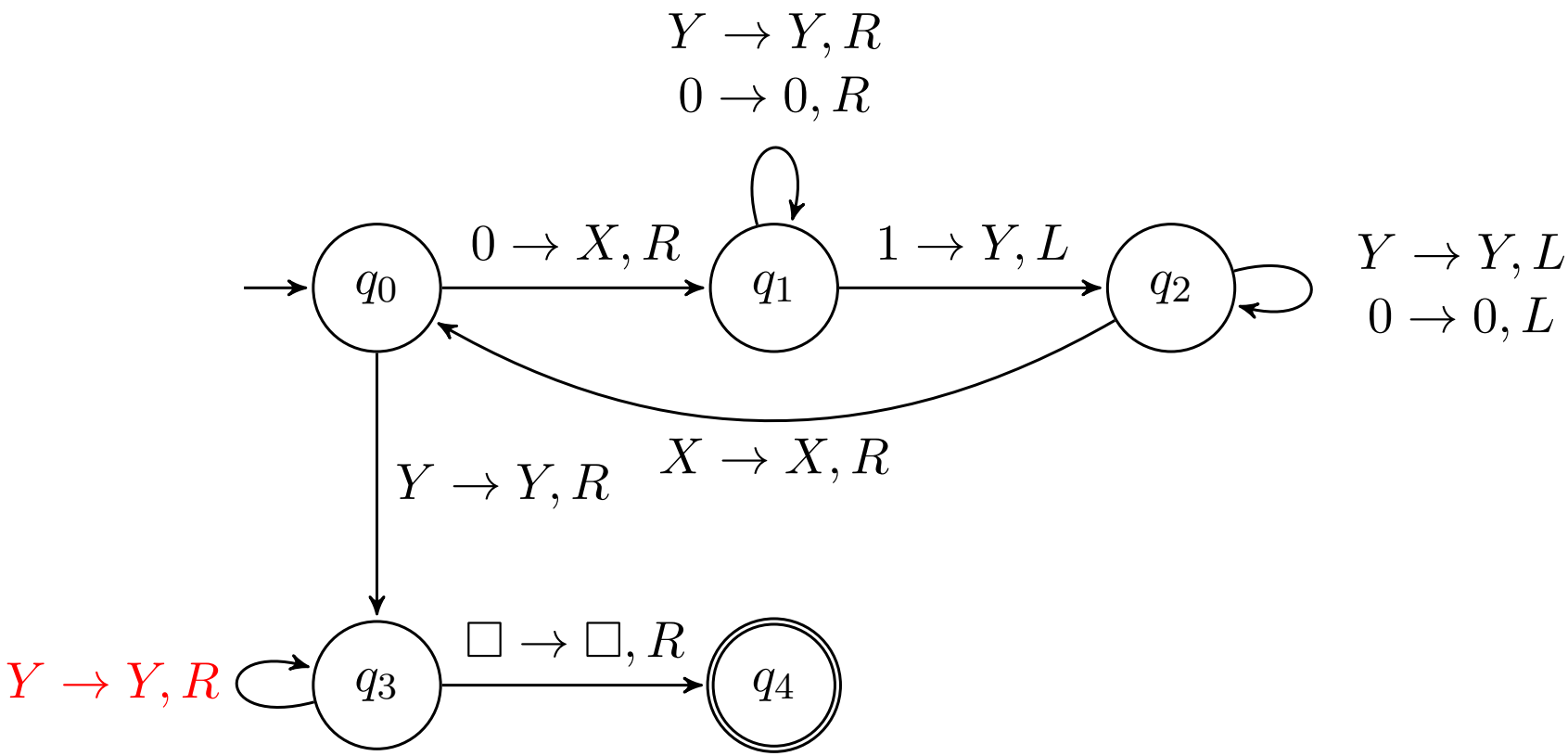




Example for 00001111

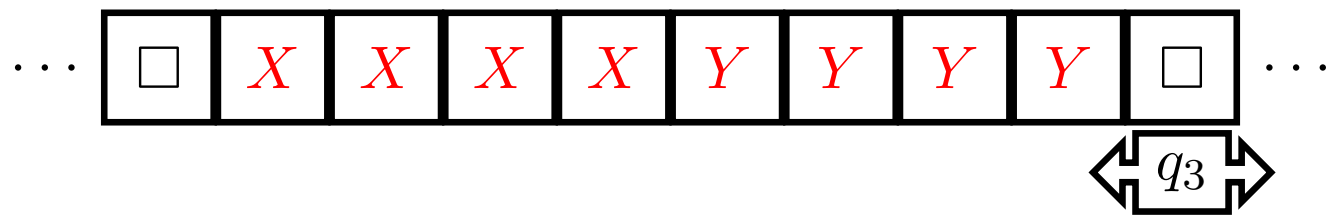
Formalities and examples

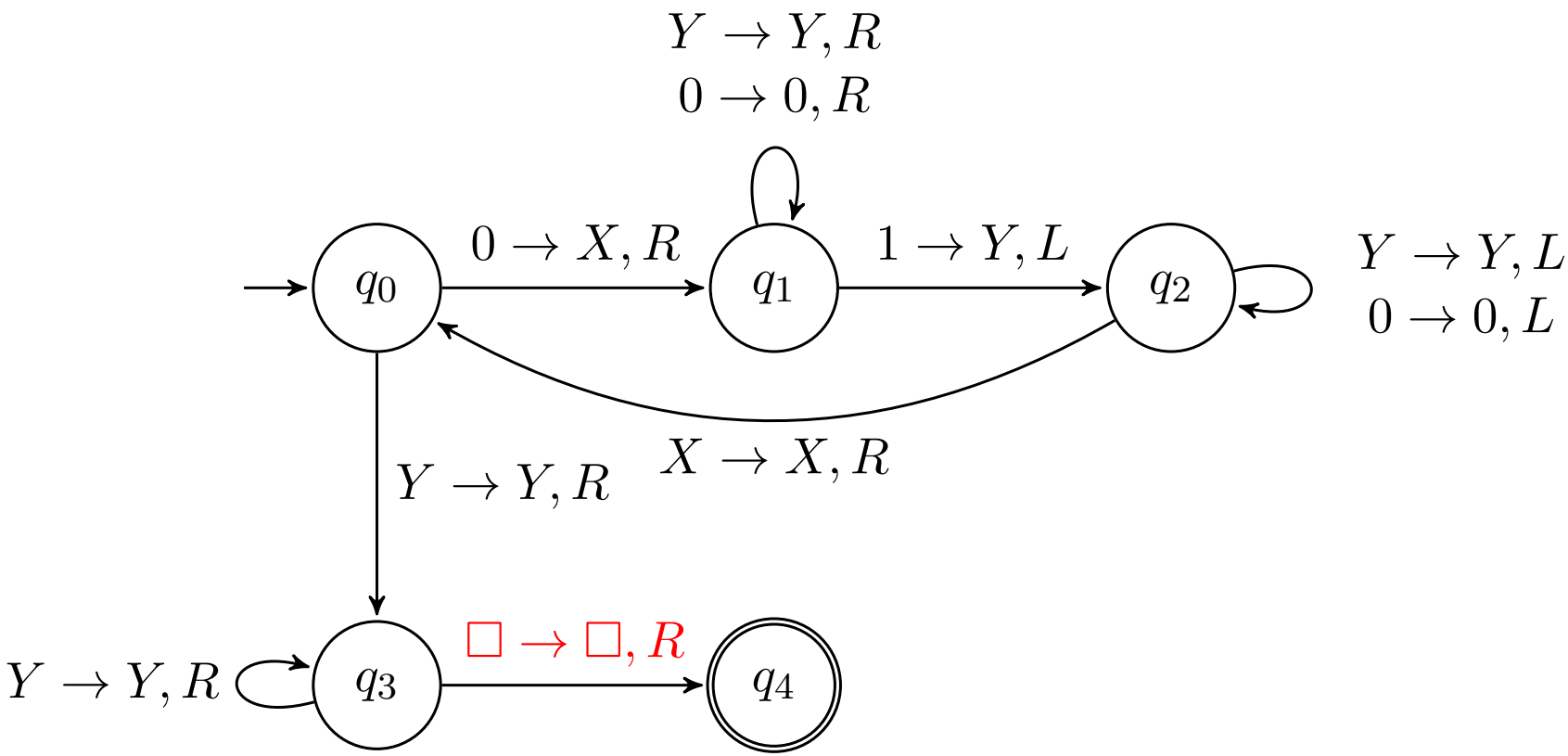




Example for 00001111

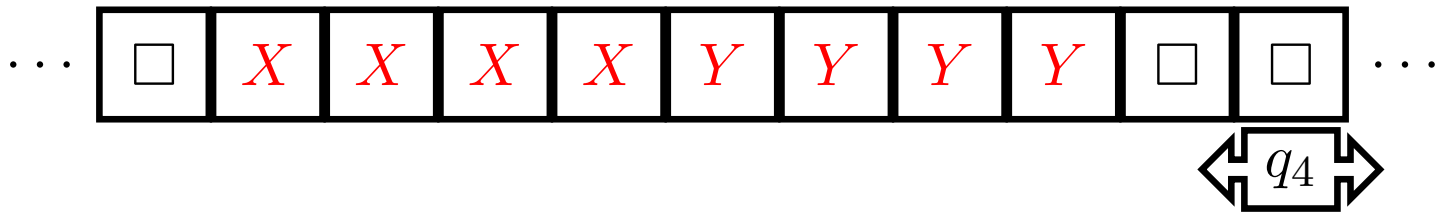
Formalities and examples

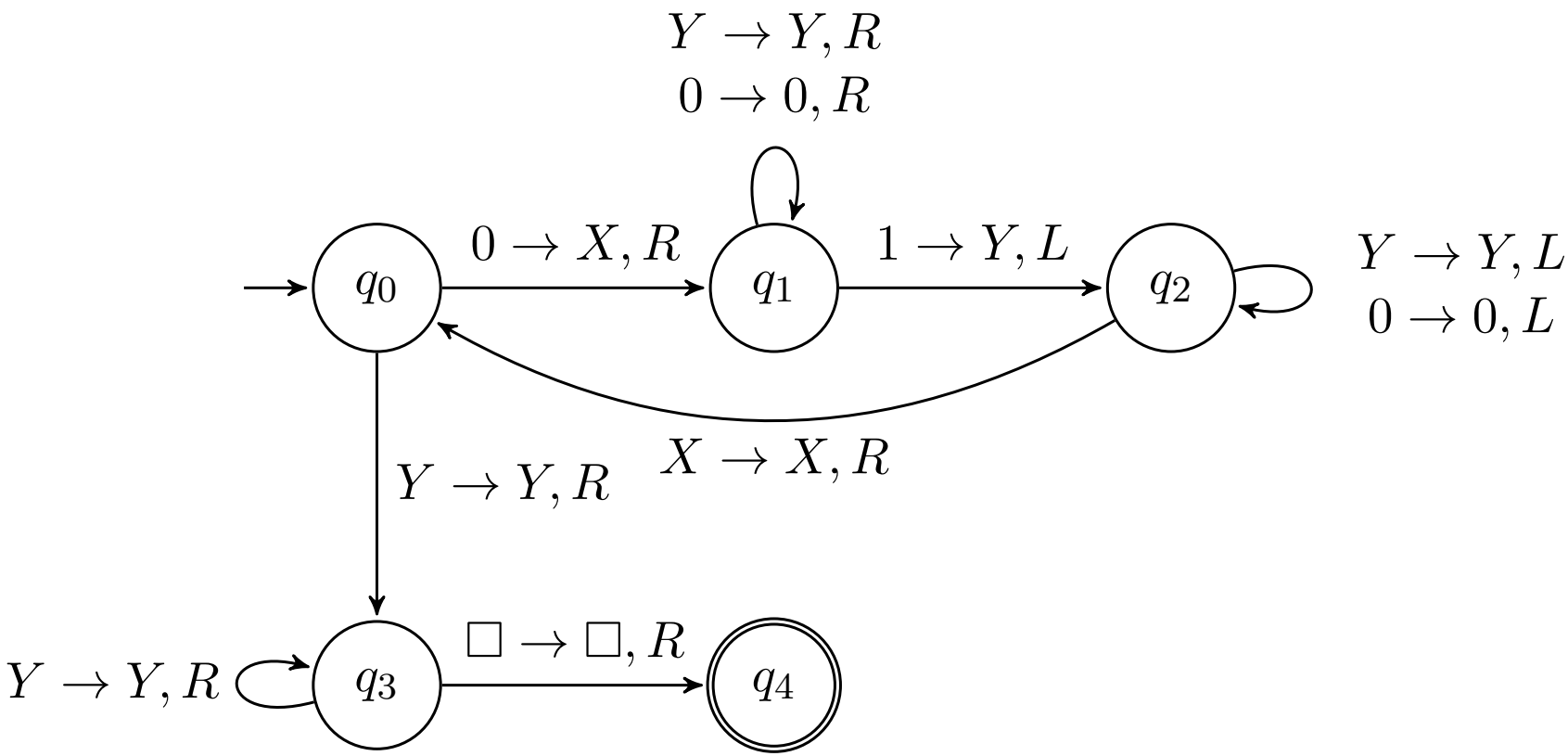




Example for 00001111

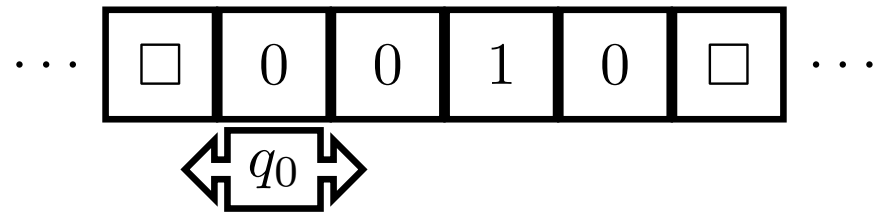
Formalities and examples

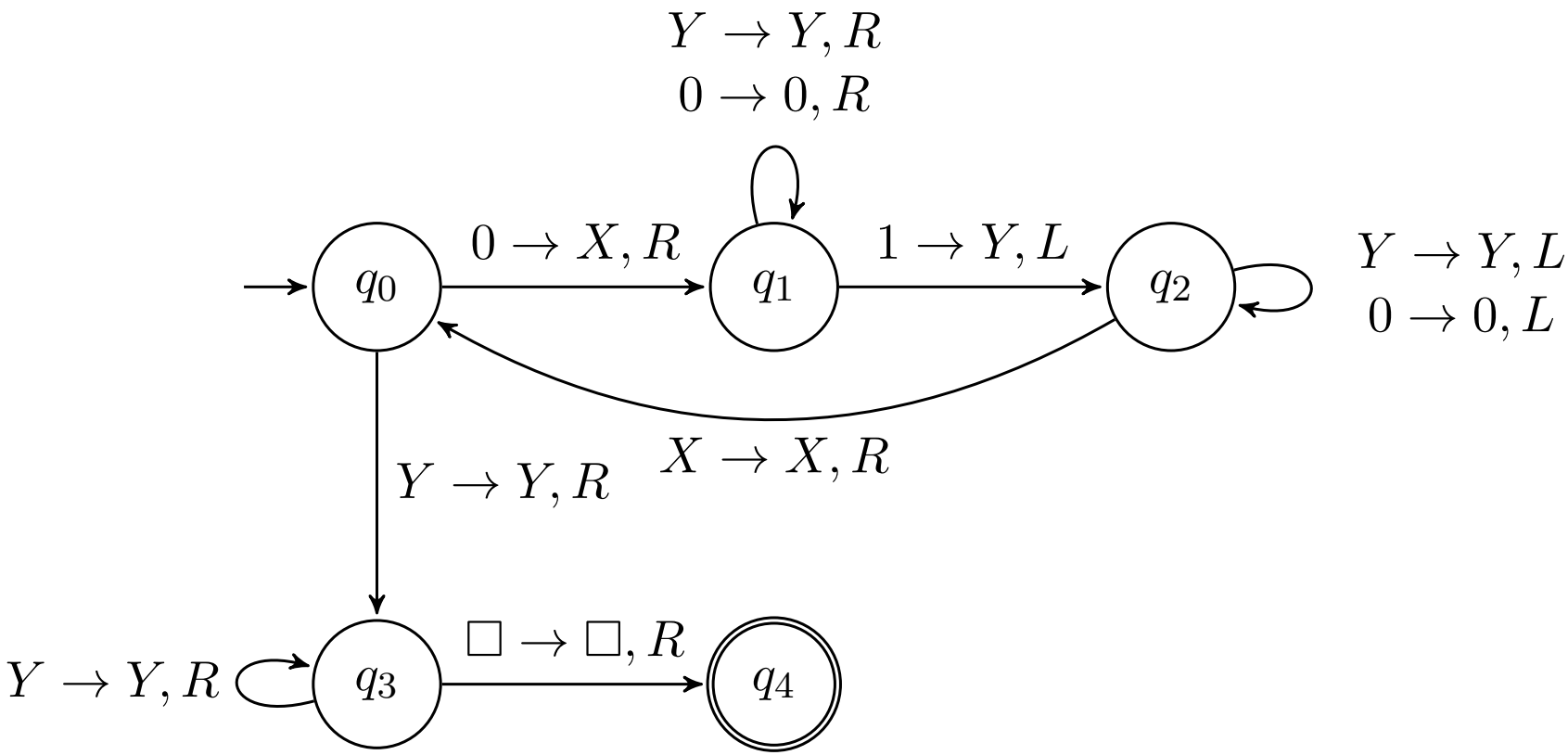




Example for 0010

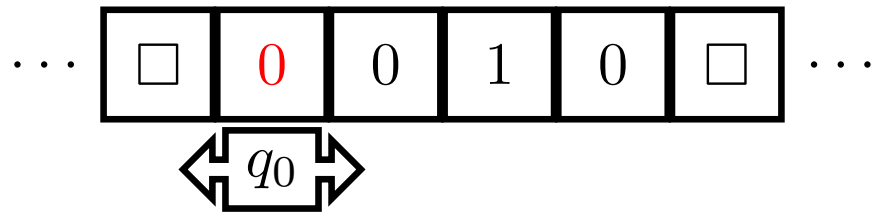
Formalities and examples

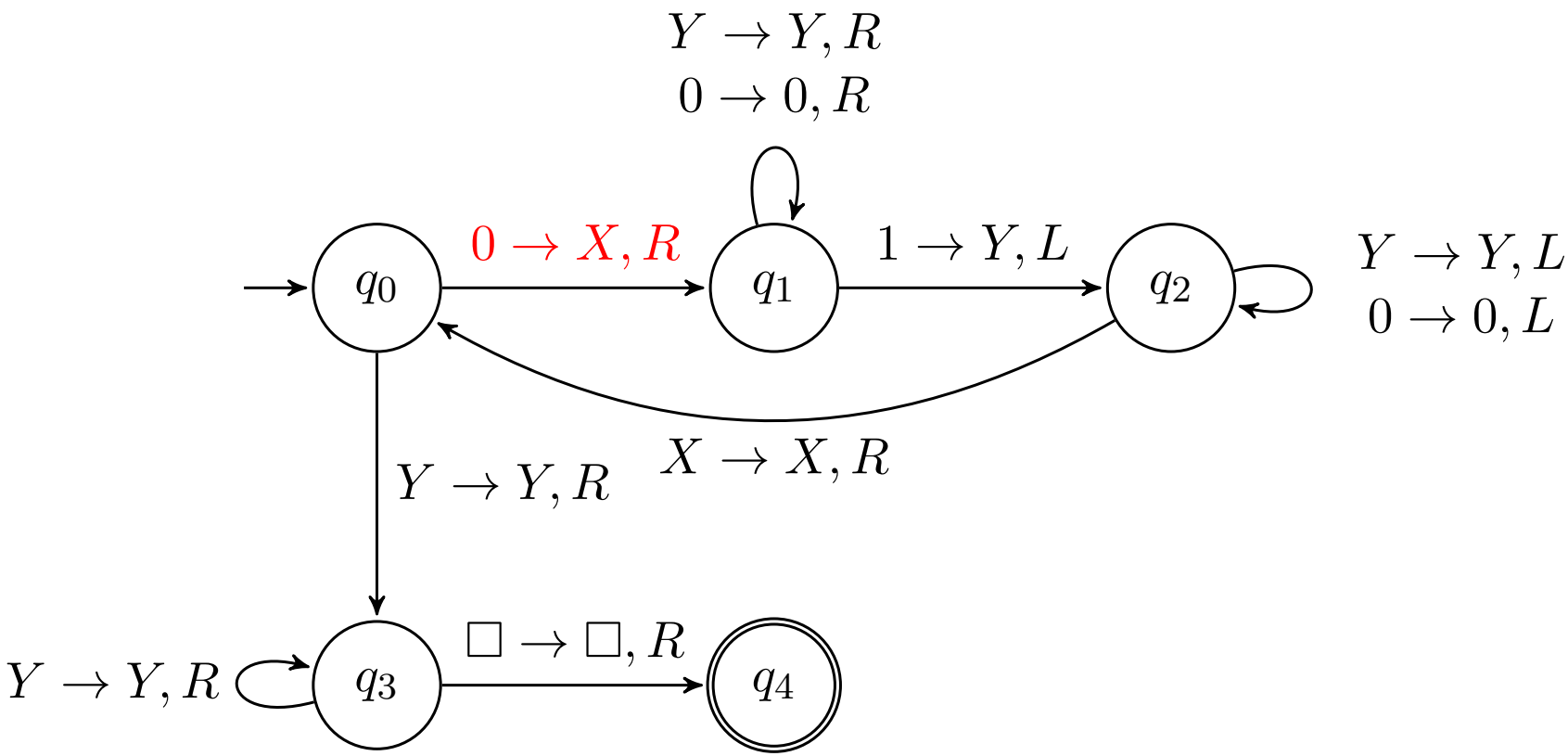




Example for 0010

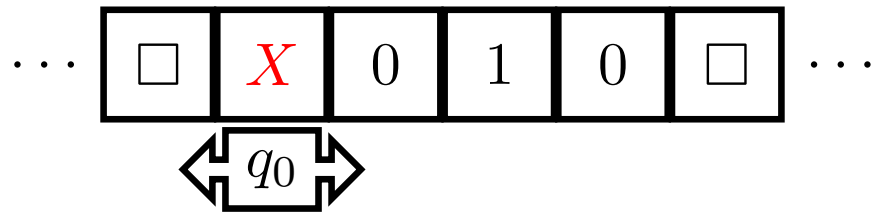
Formalities and examples

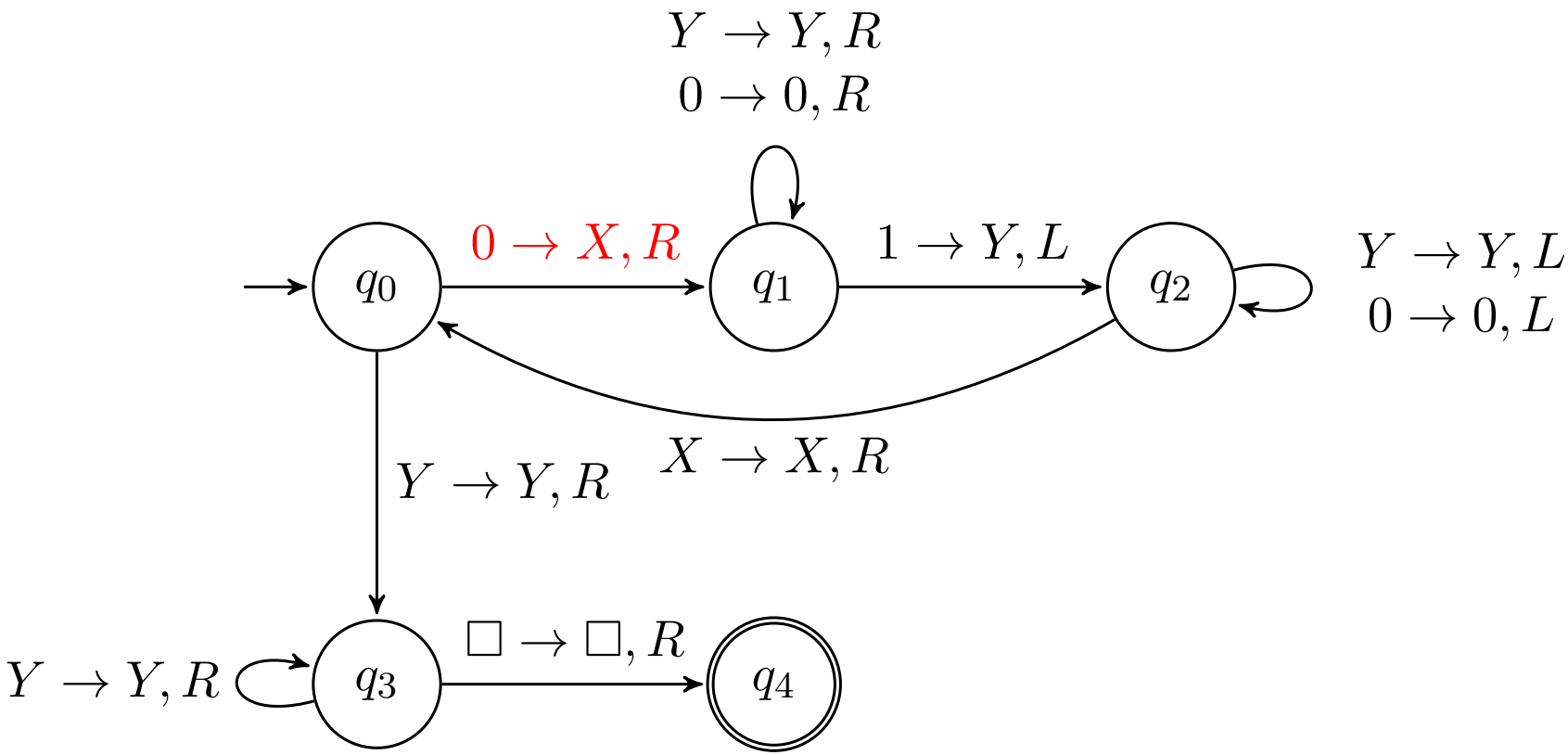




Example for 0010

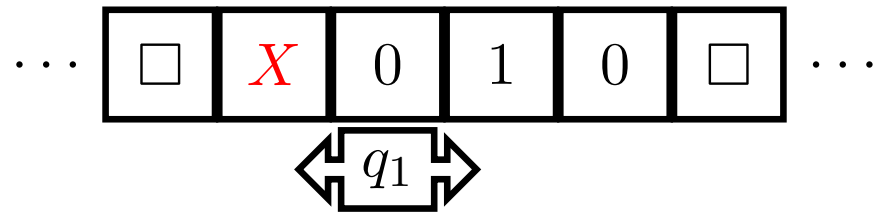
Formalities and examples

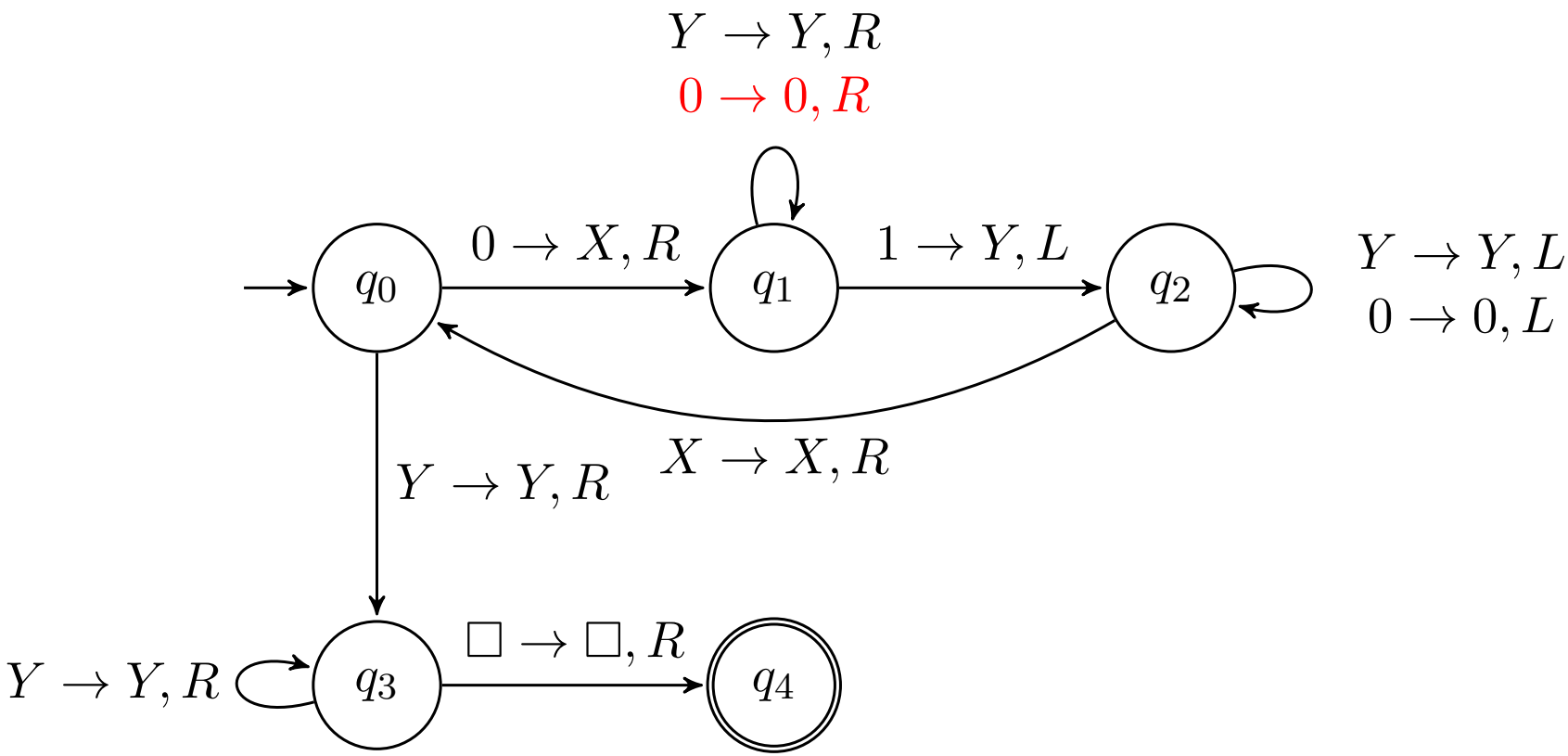




Example for 0010

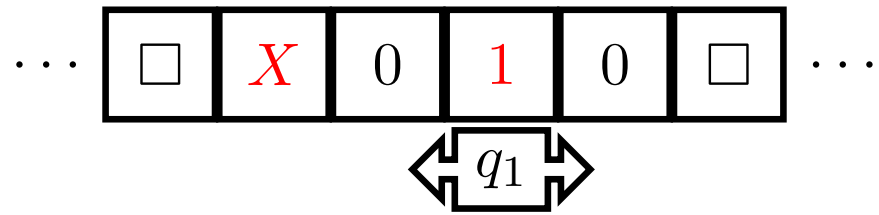
Formalities and examples

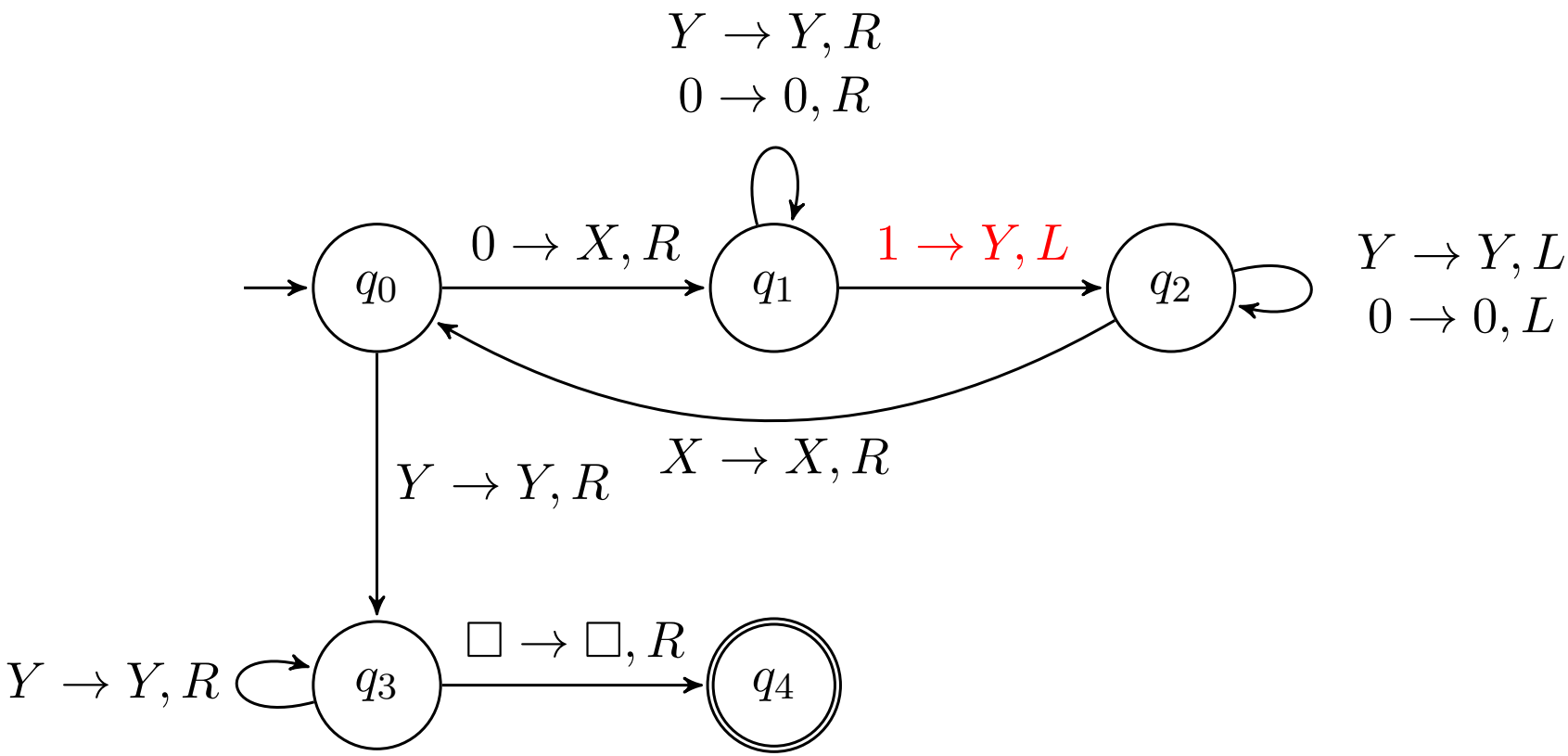




Example for 0010

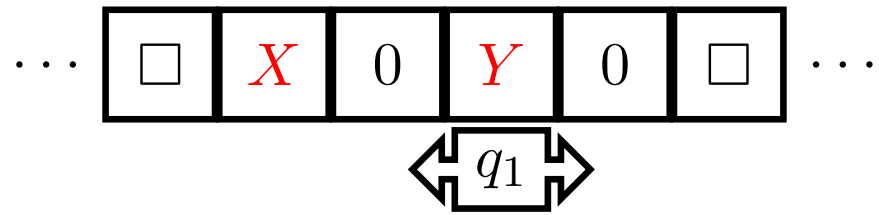
Formalities and examples

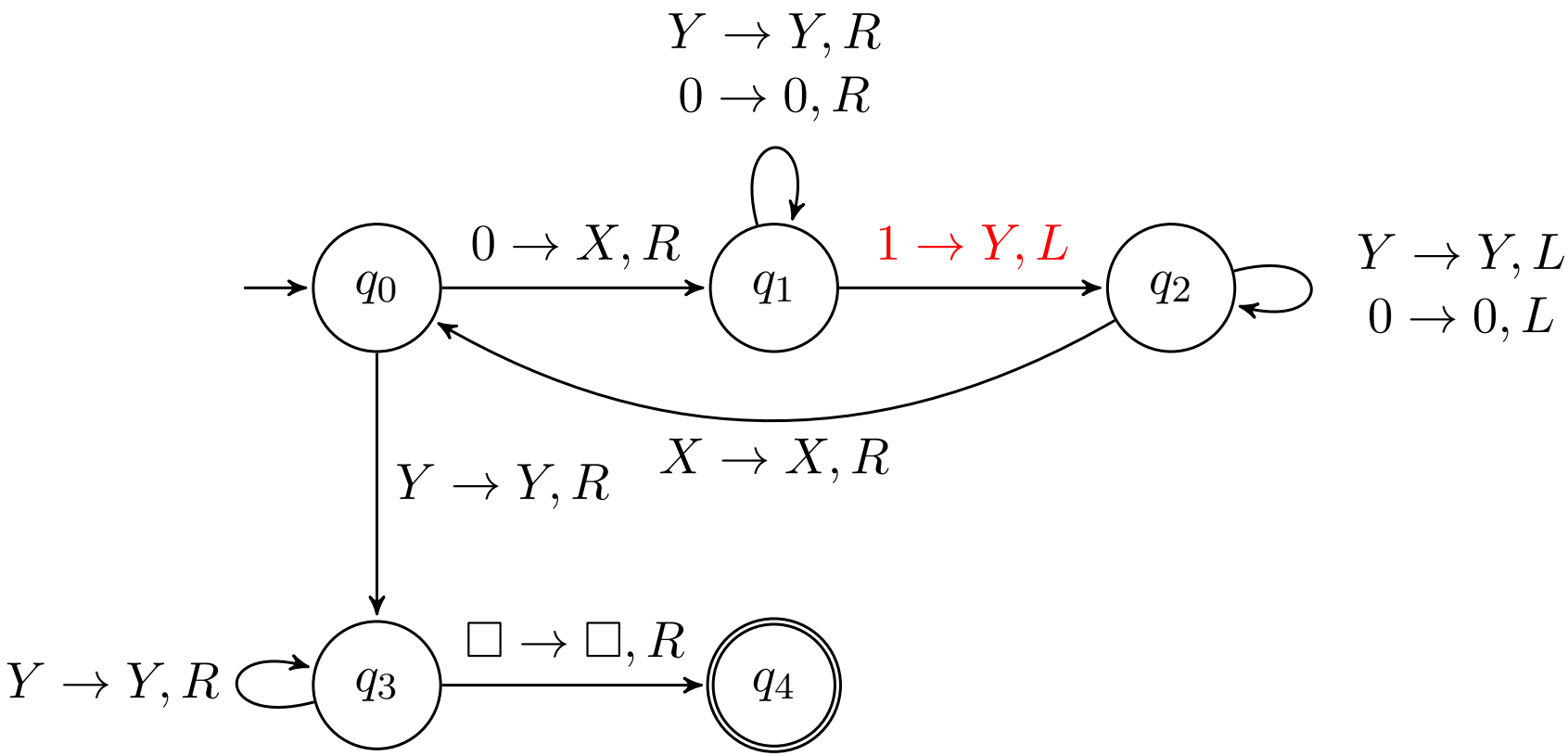




Example for 0010

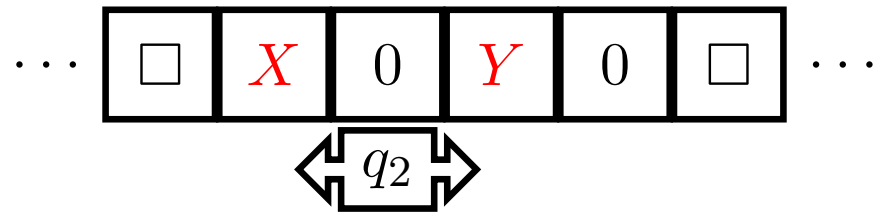
Formalities and examples

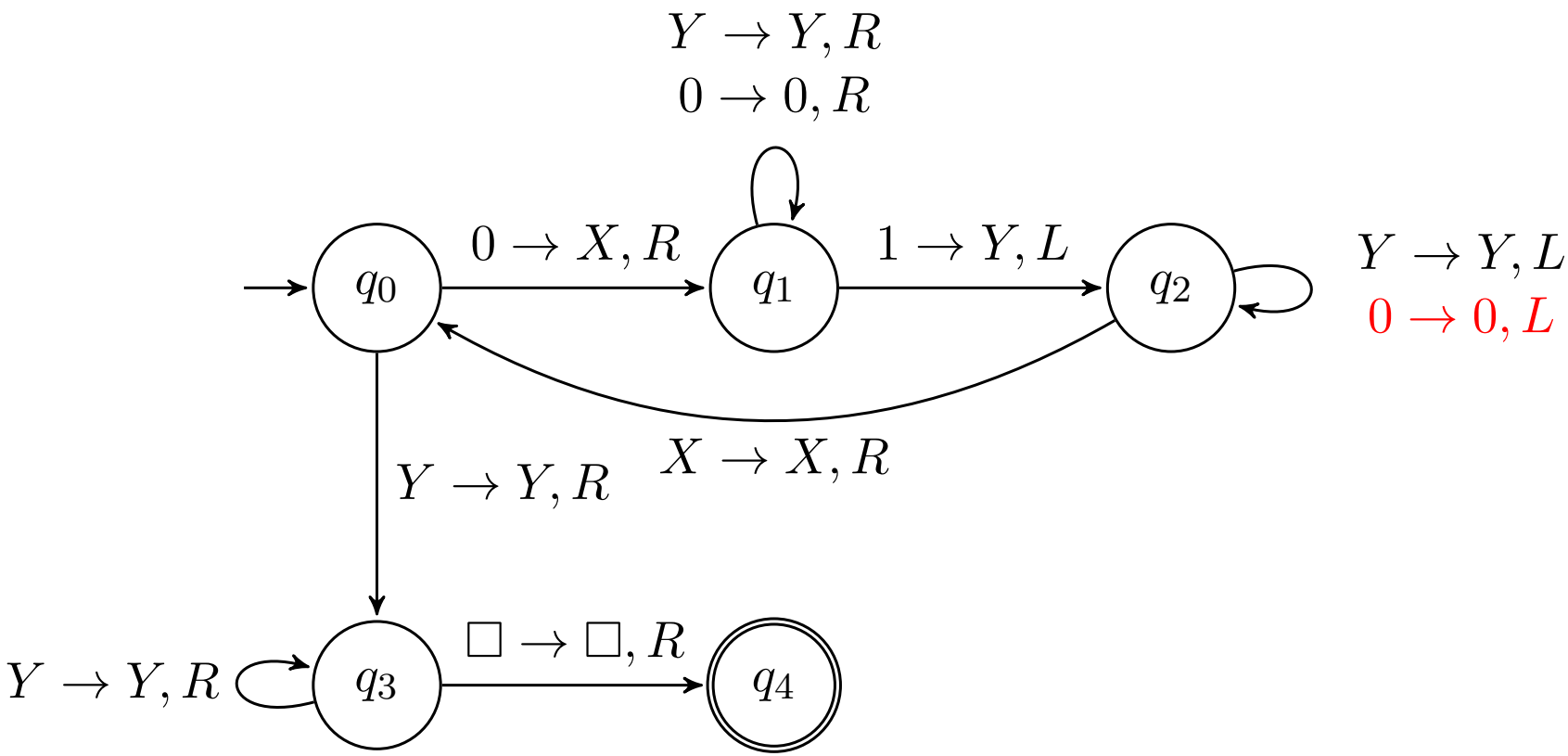




Example for 0010

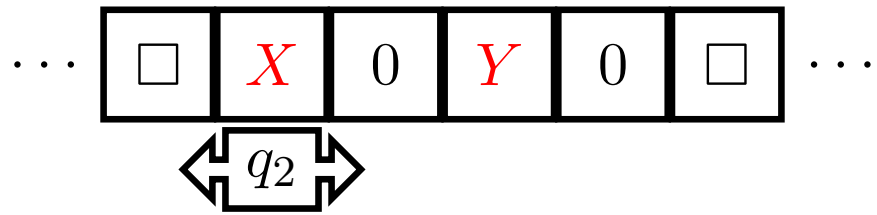
Formalities and examples

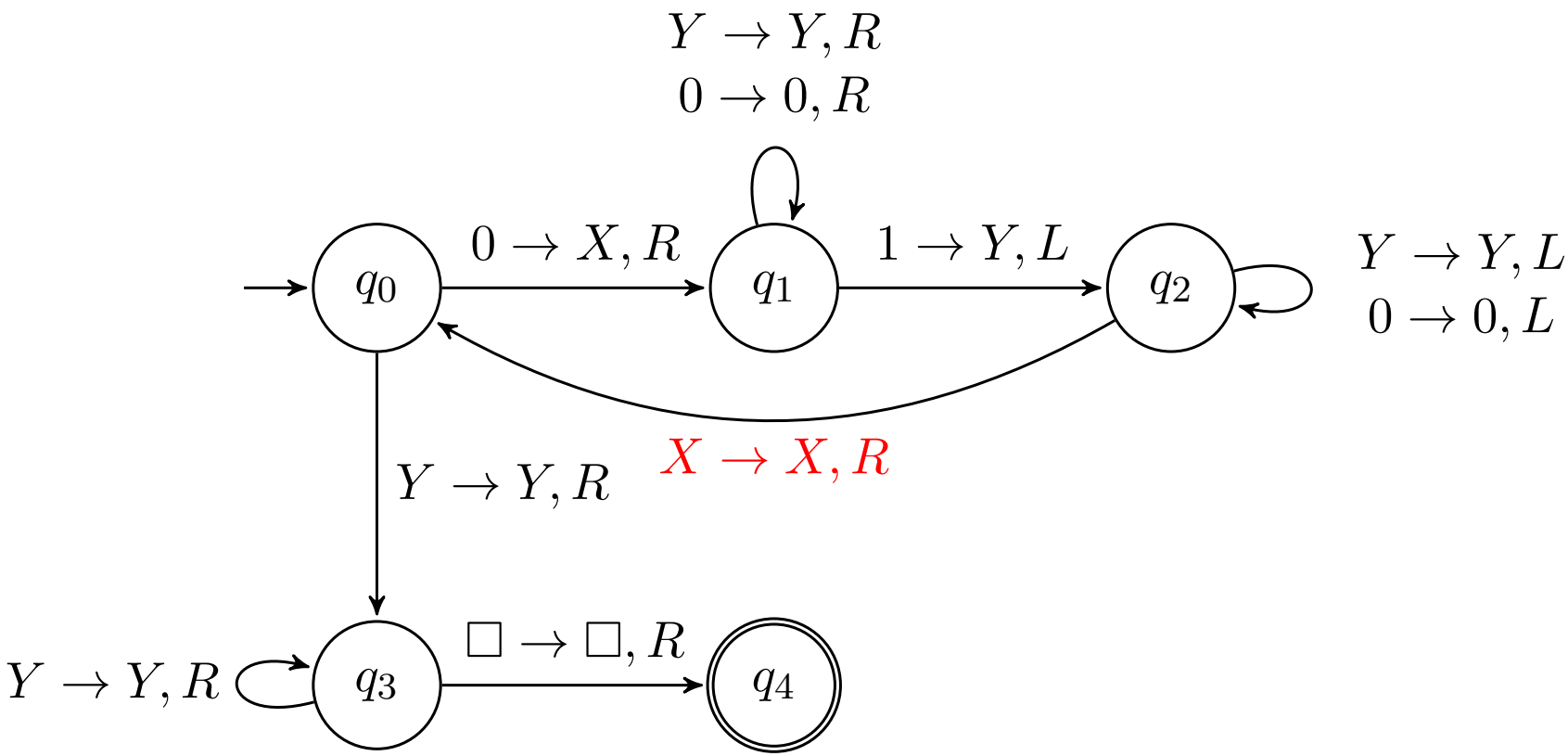




Example for 0010

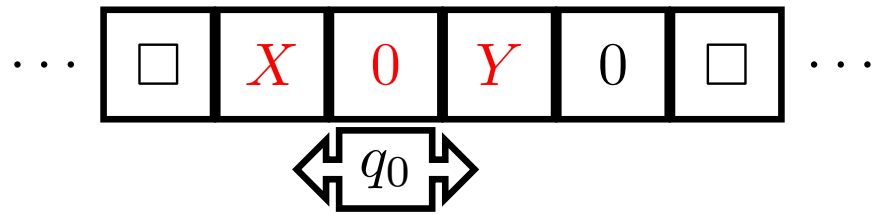
Formalities and examples

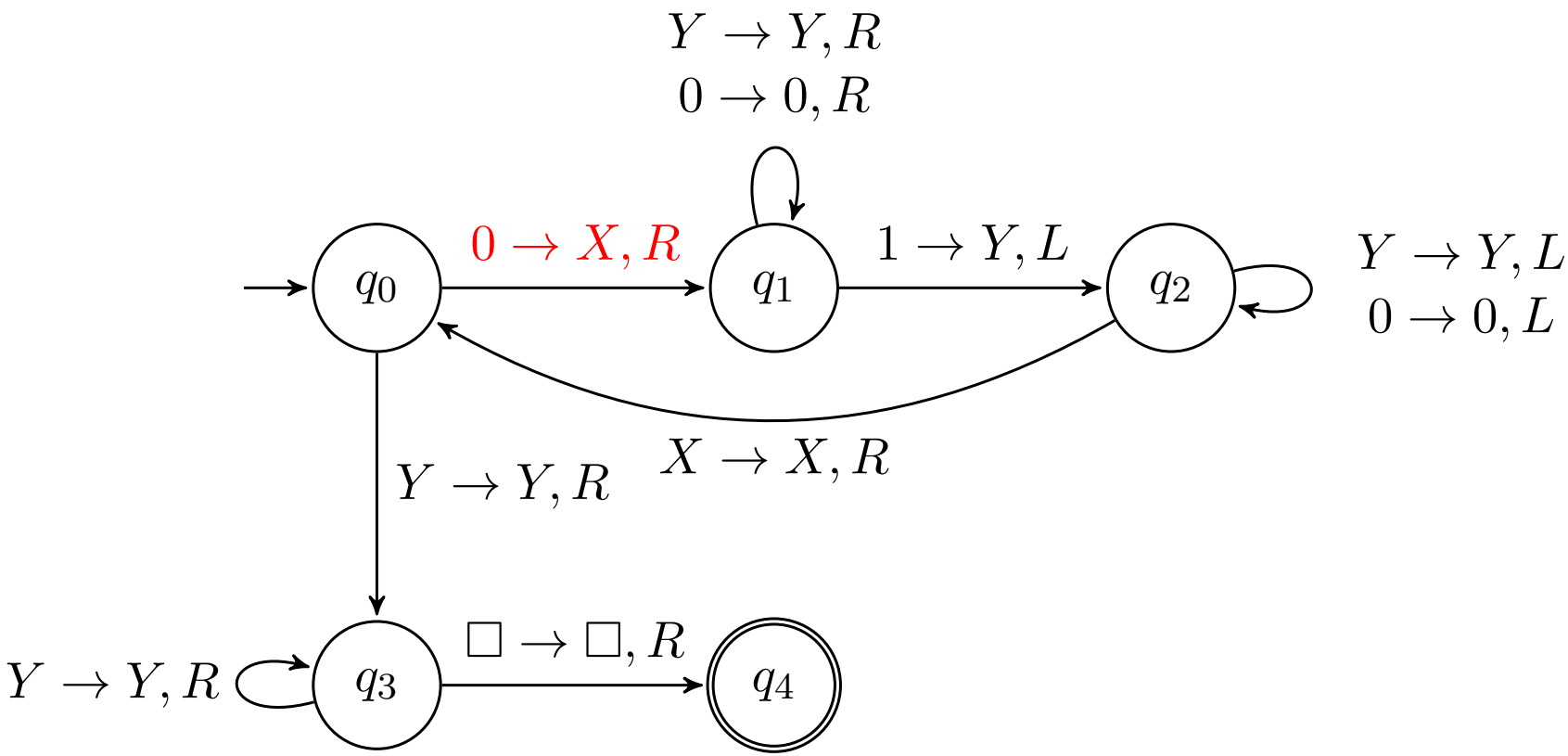




Example for 0010

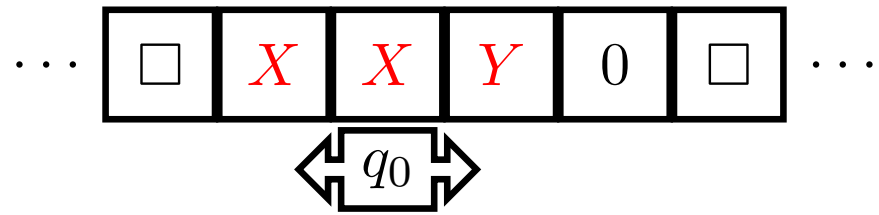
Formalities and examples

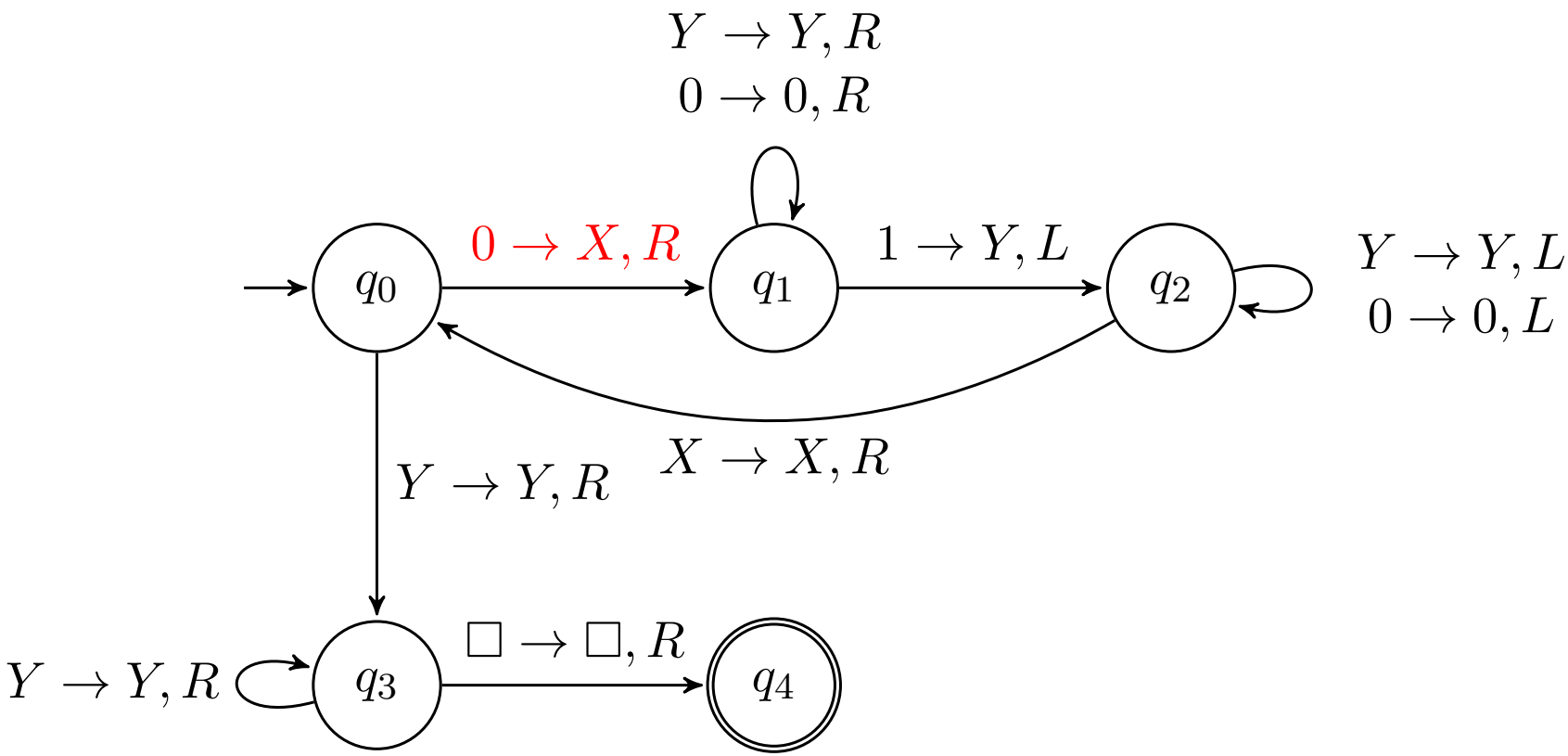




Example for 0010

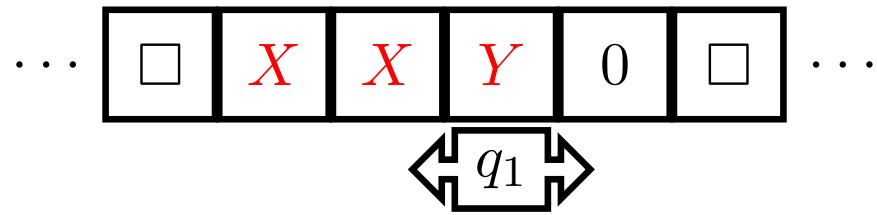
Formalities and examples

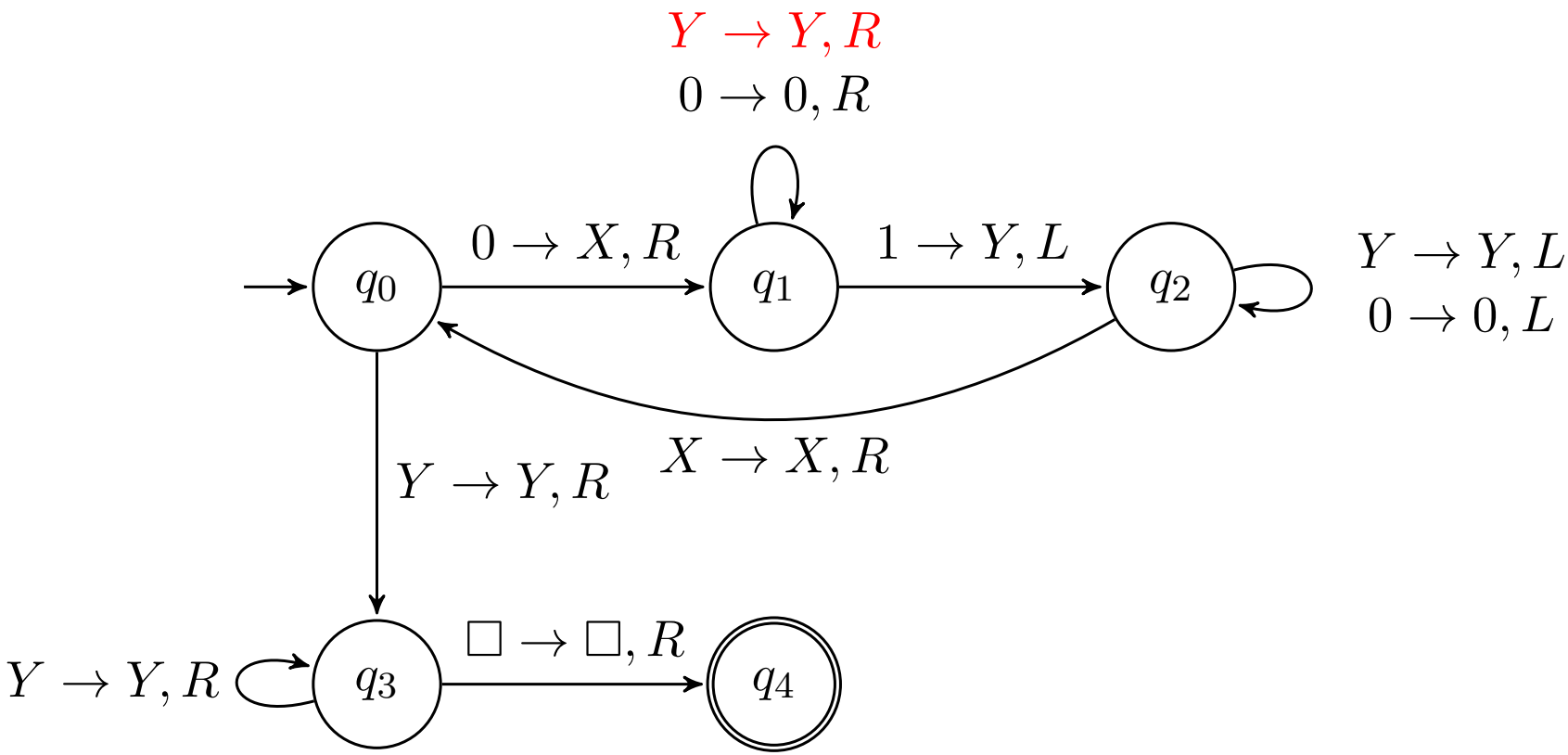




Example for 0010

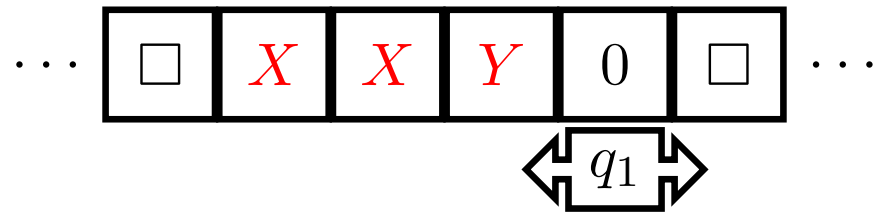
Formalities and examples

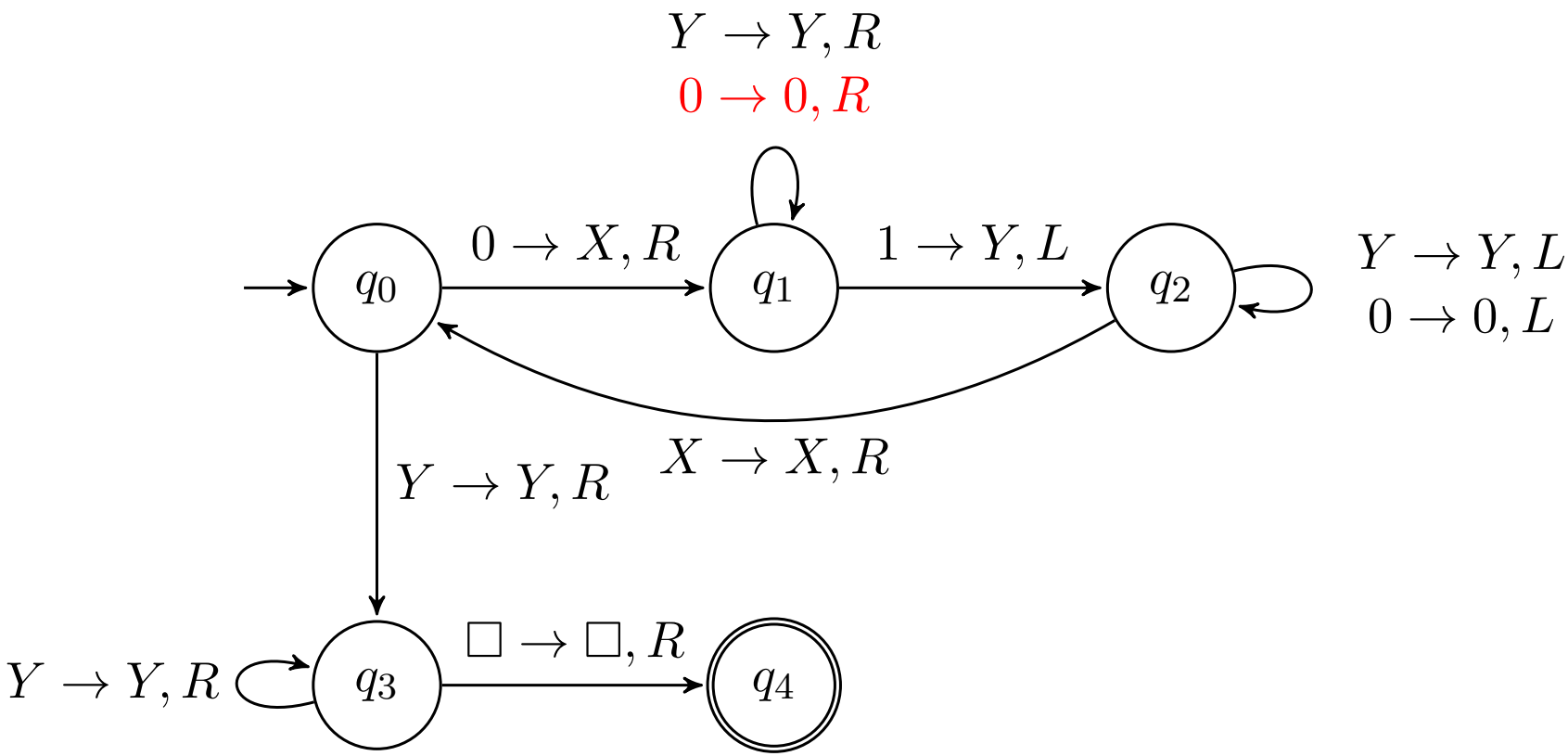




Example for 0010

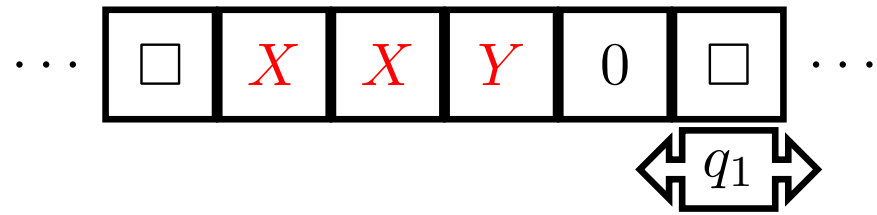
Formalities and examples

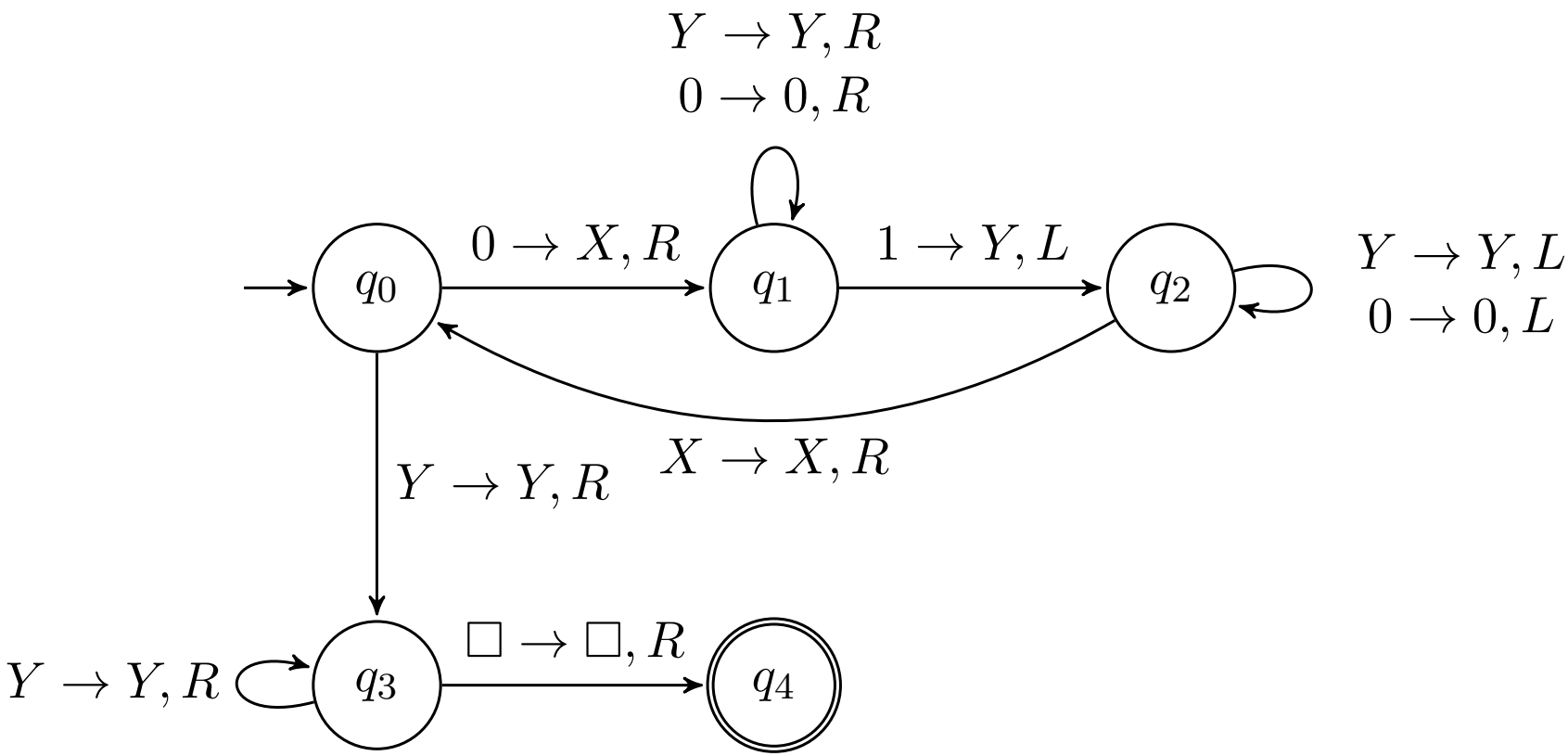




Example for 0010

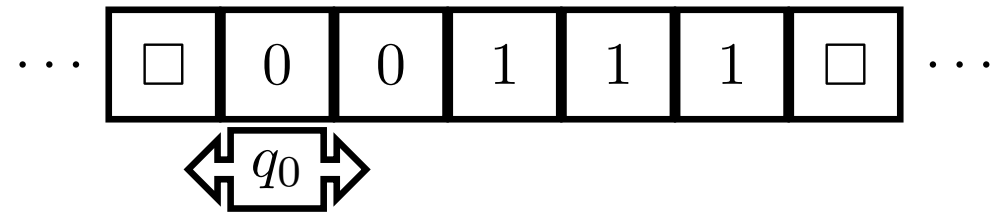
Formalities and examples

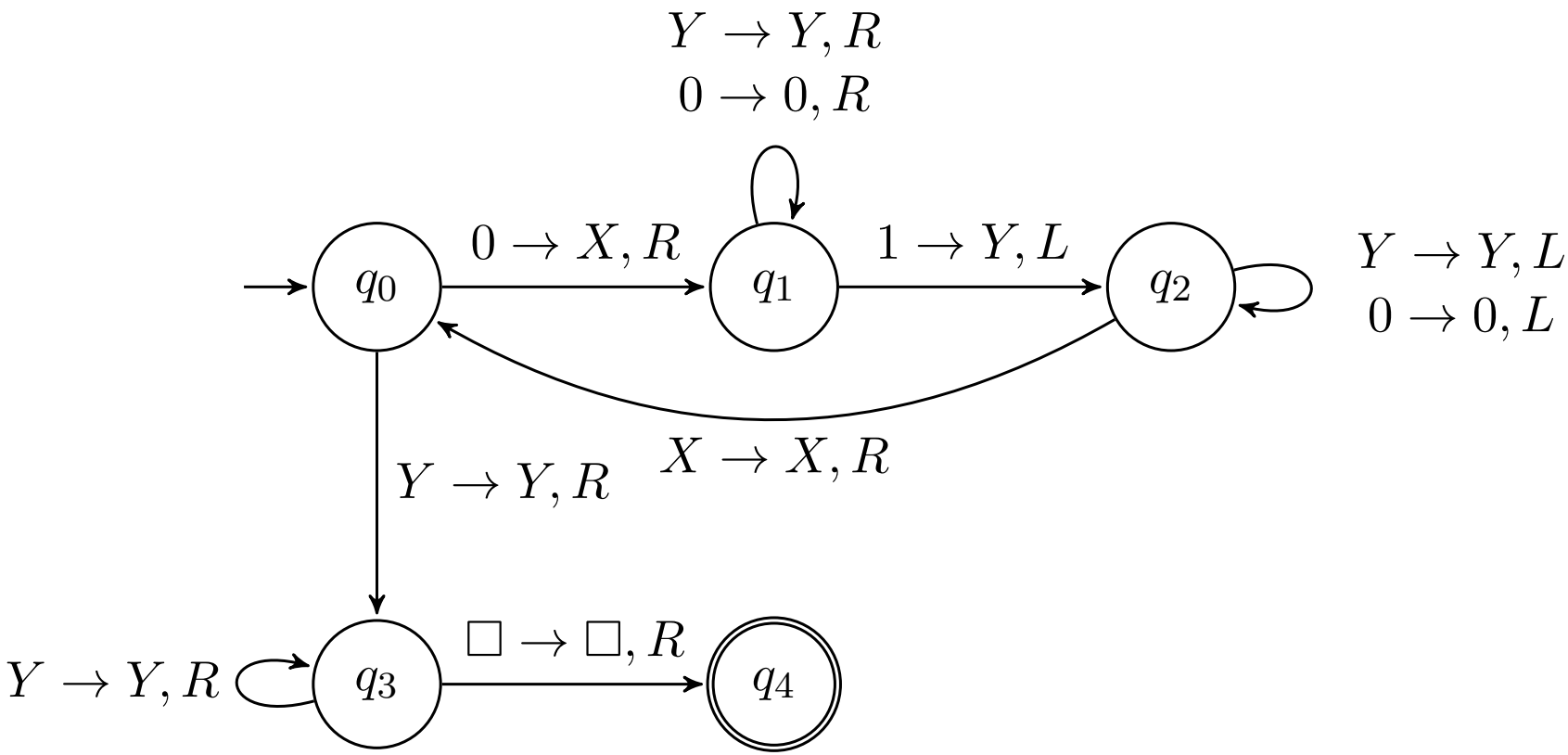




Example for 00111

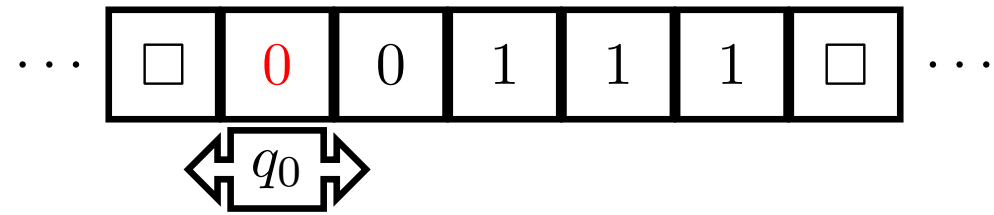
Formalities and examples

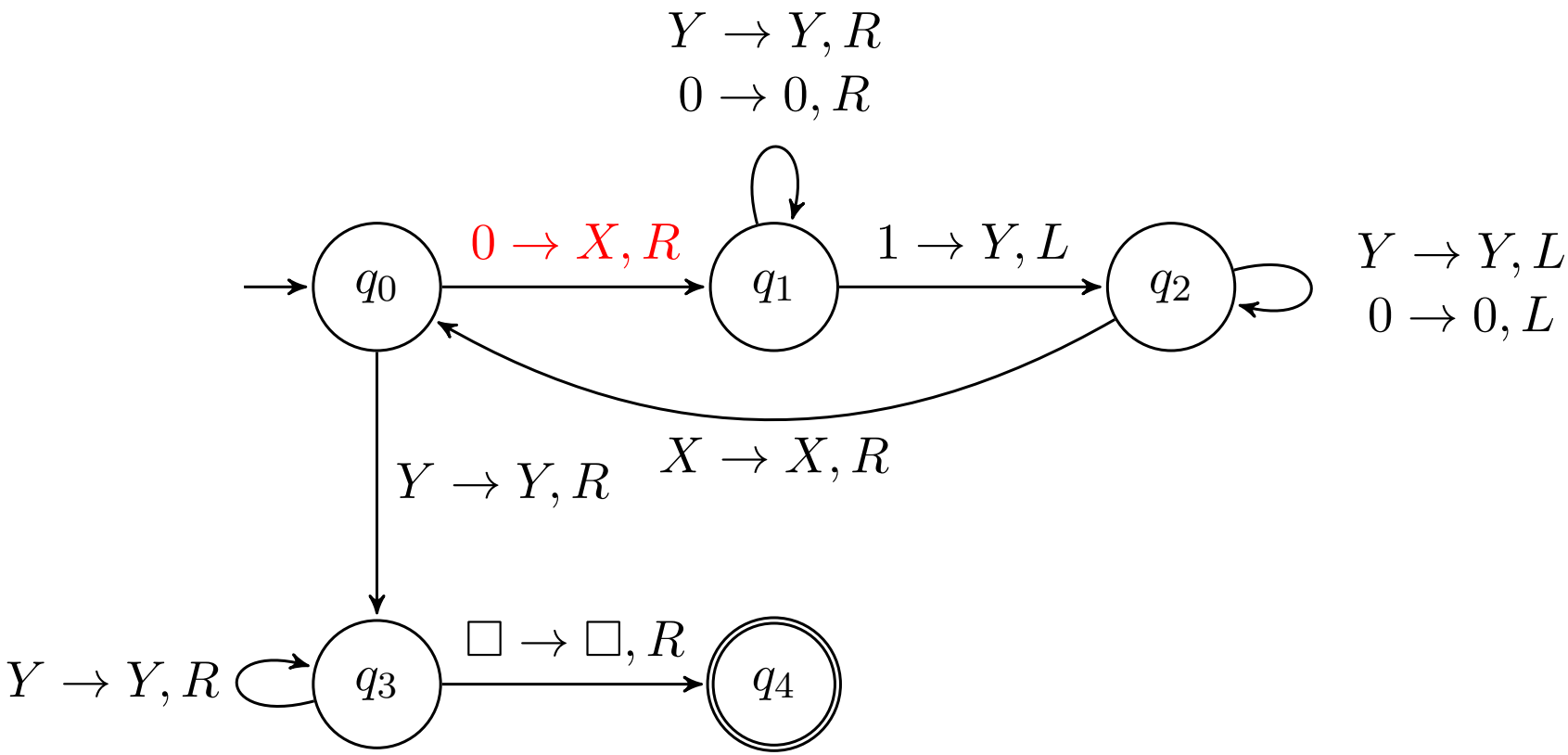




Example for 00111

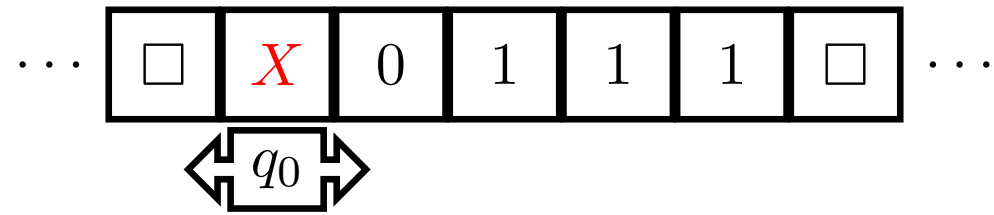
Formalities and examples

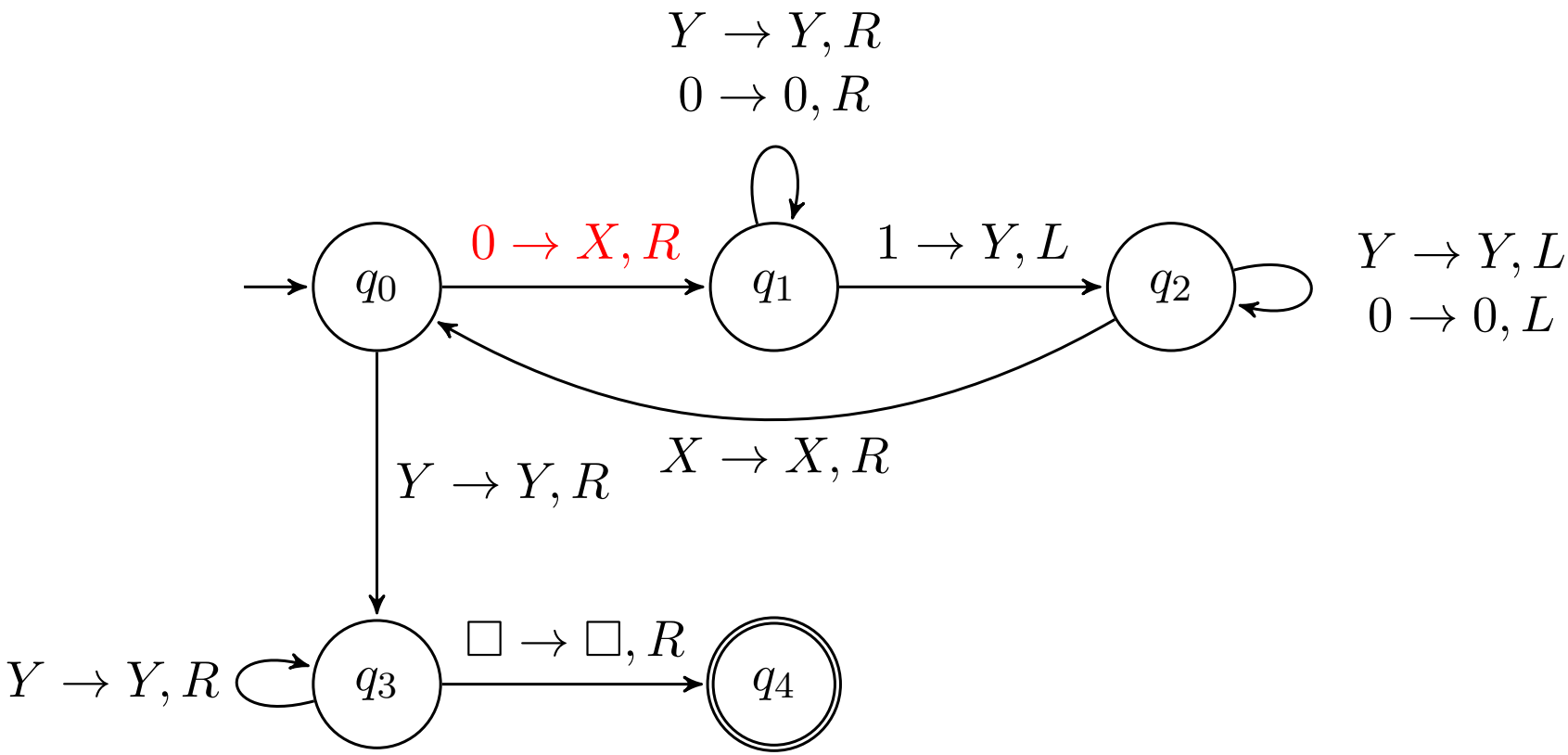




Example for 00111

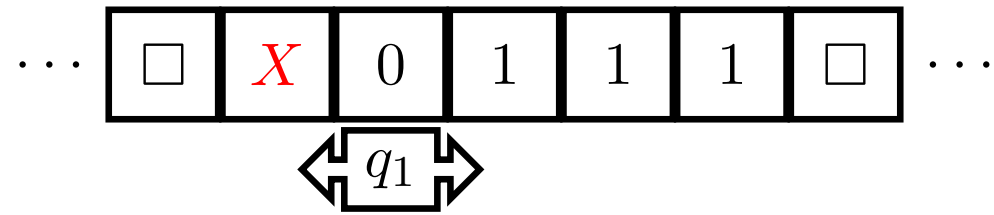
Formalities and examples

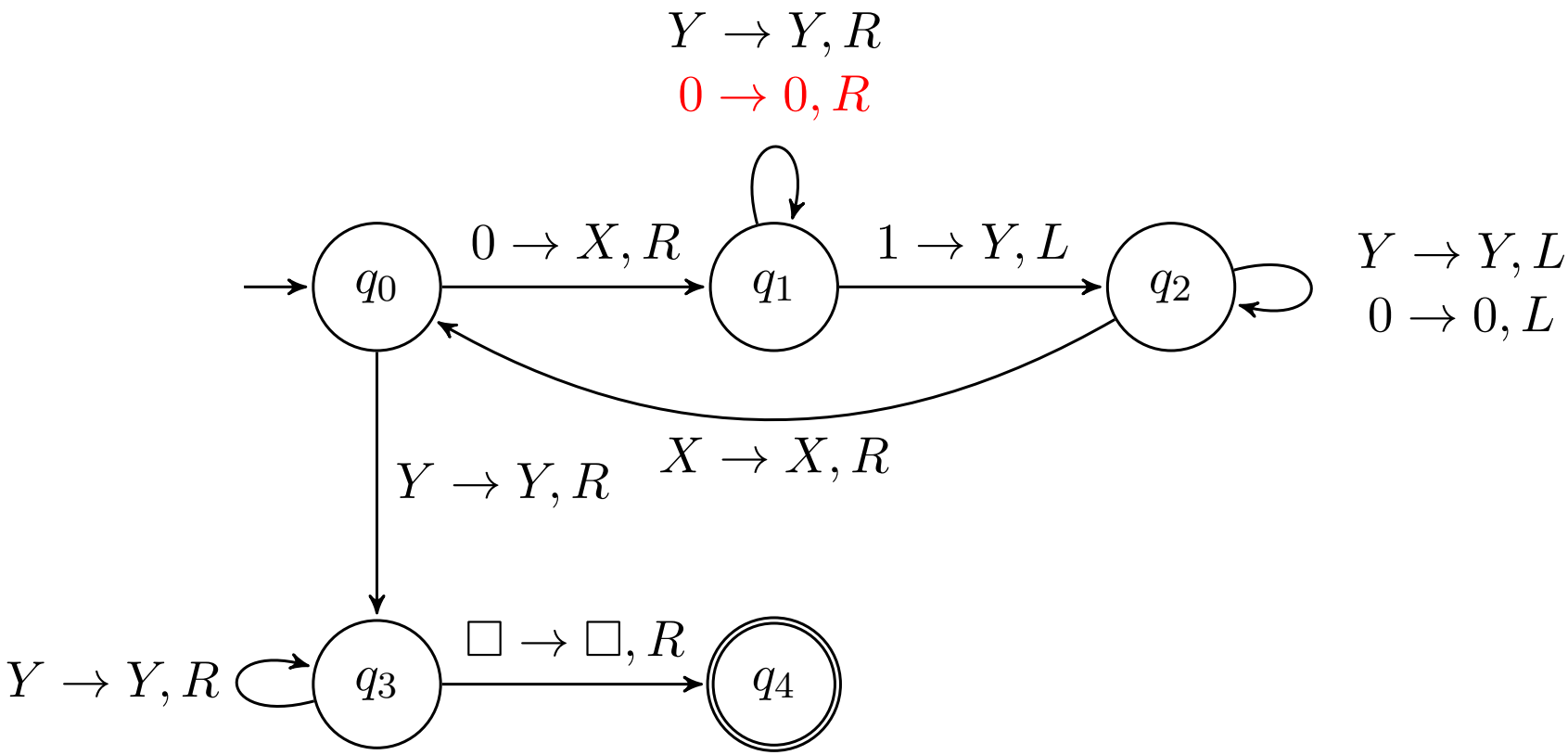




Example for 00111

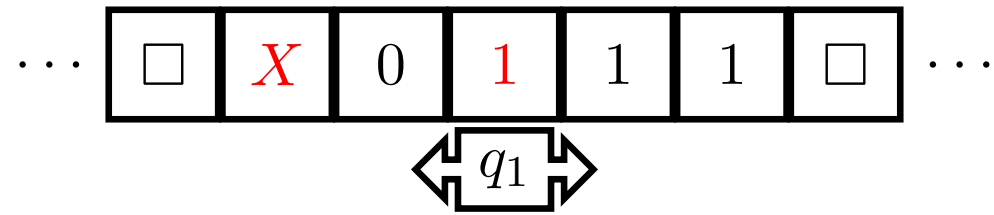
Formalities and examples

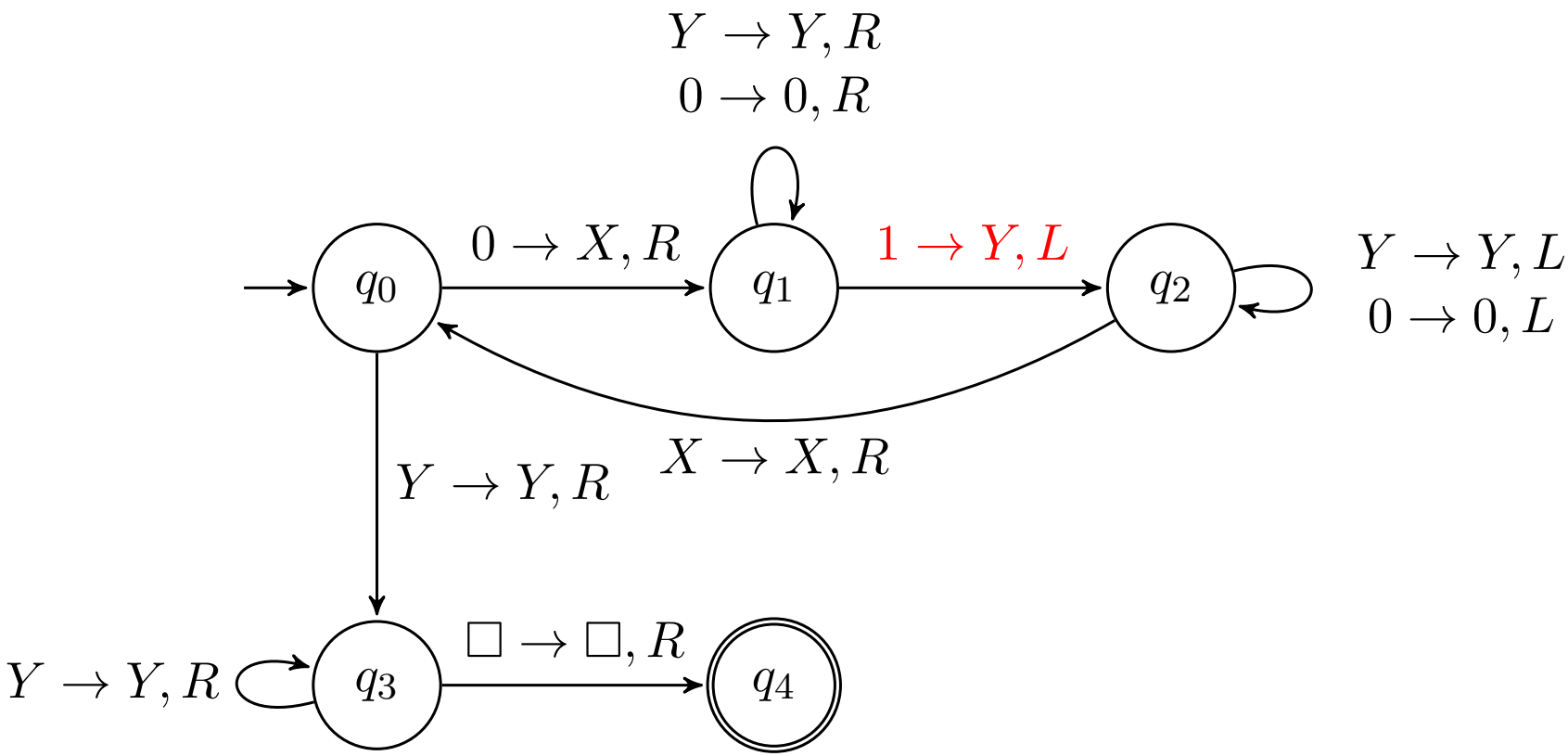




Example for 00111

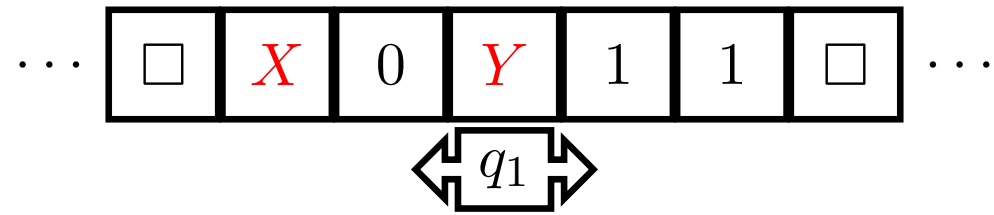
Formalities and examples

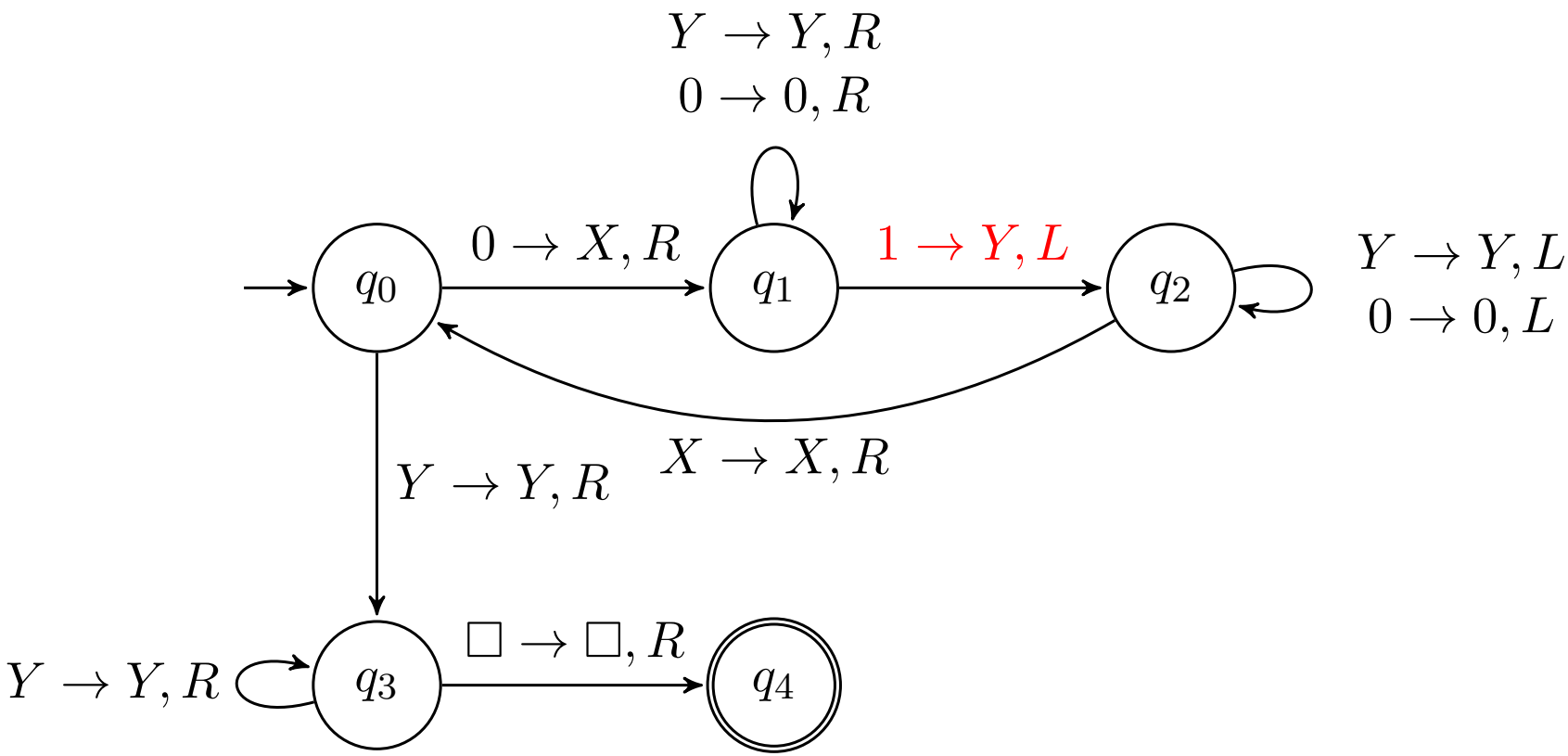




Example for 00111

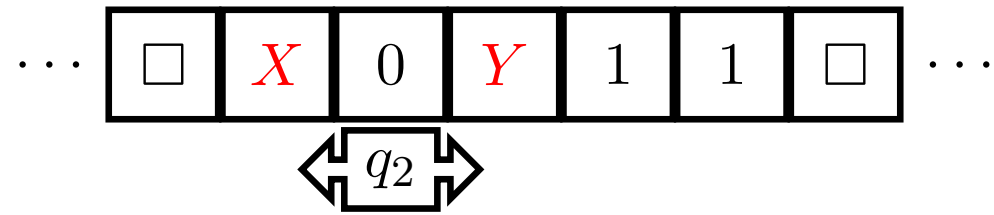
Formalities and examples

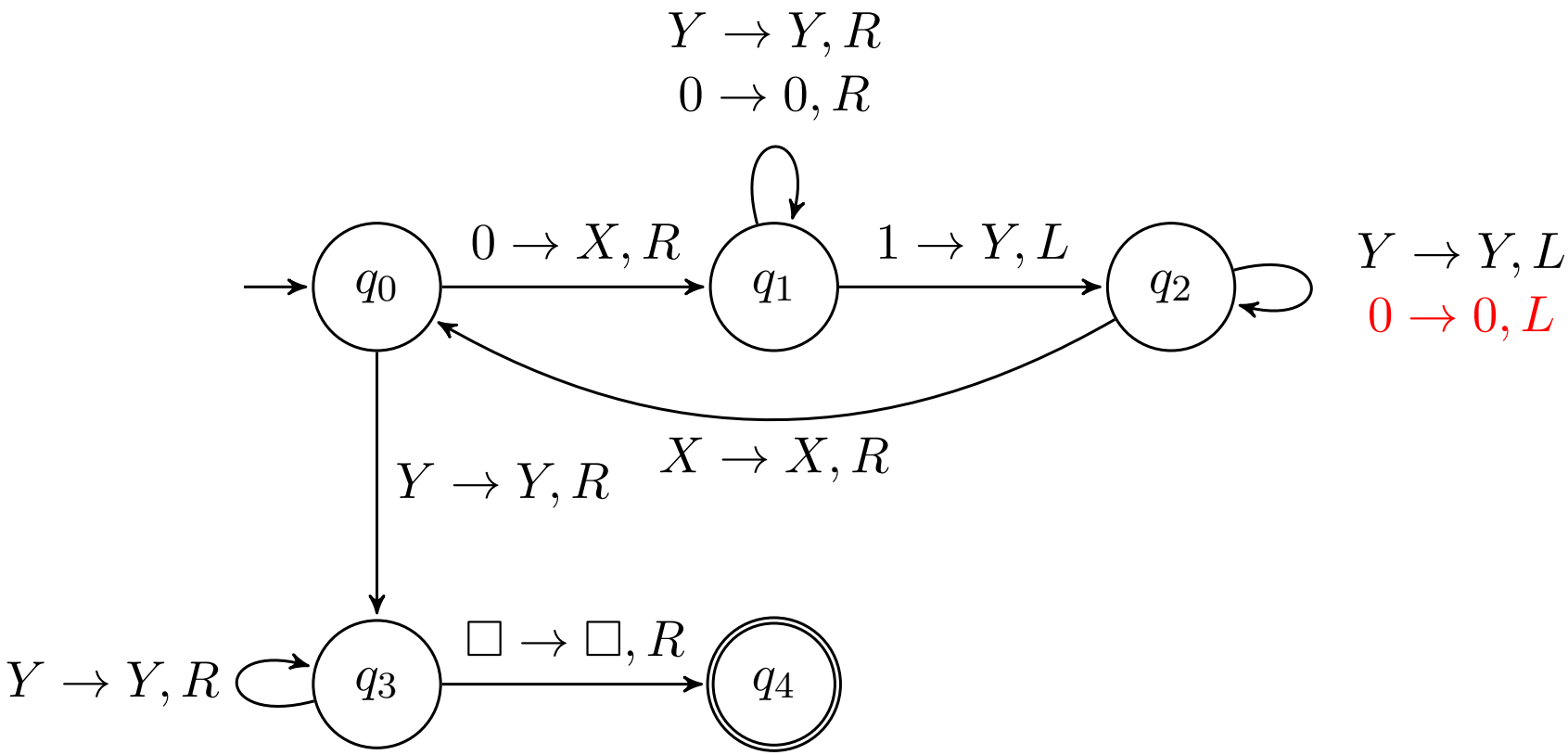




Example for 00111

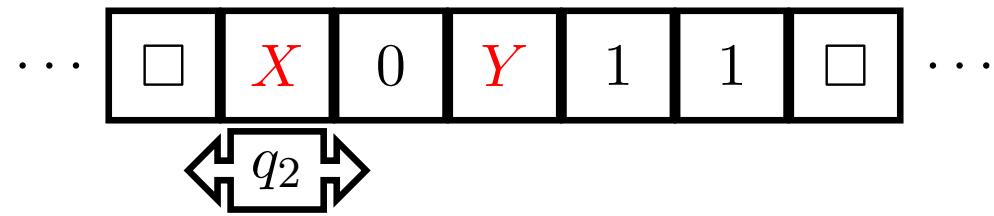
Formalities and examples

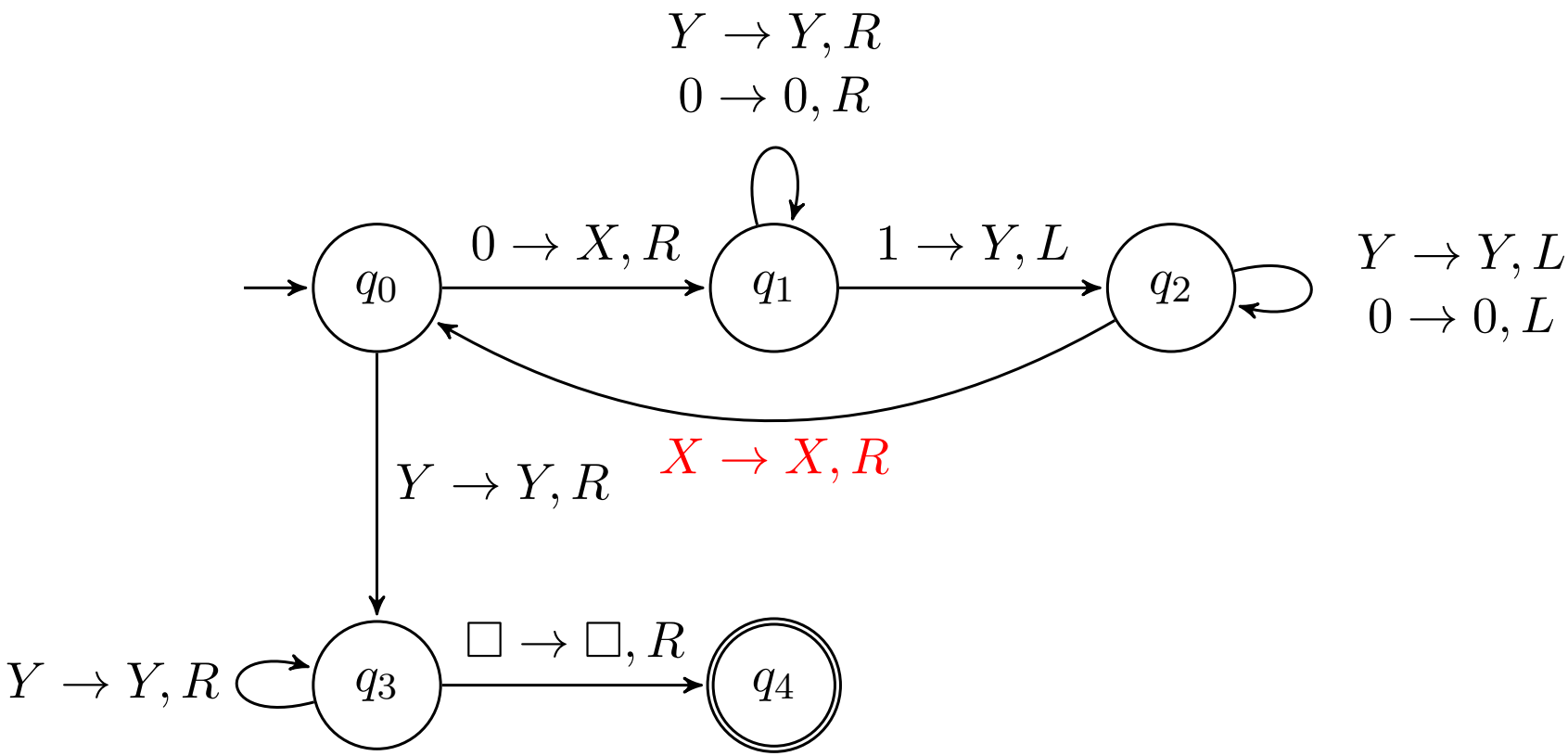




Example for 00111

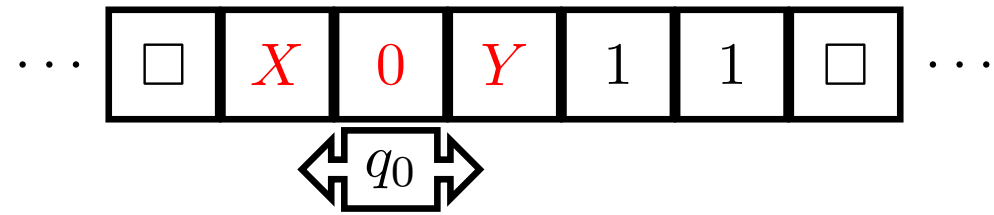
Formalities and examples

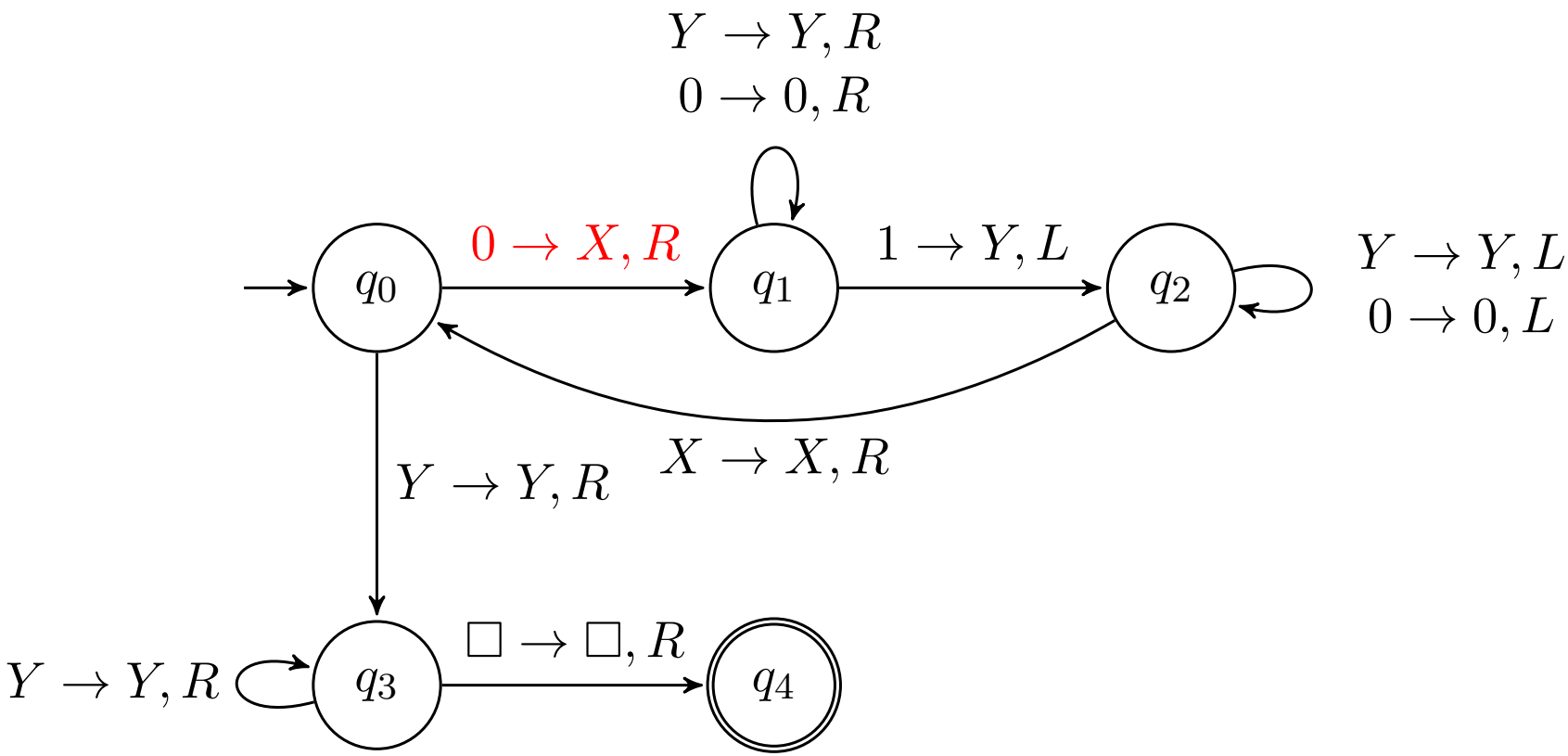




Example for 00111

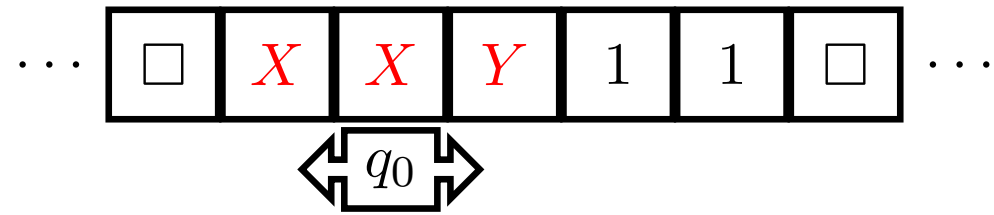
Formalities and examples

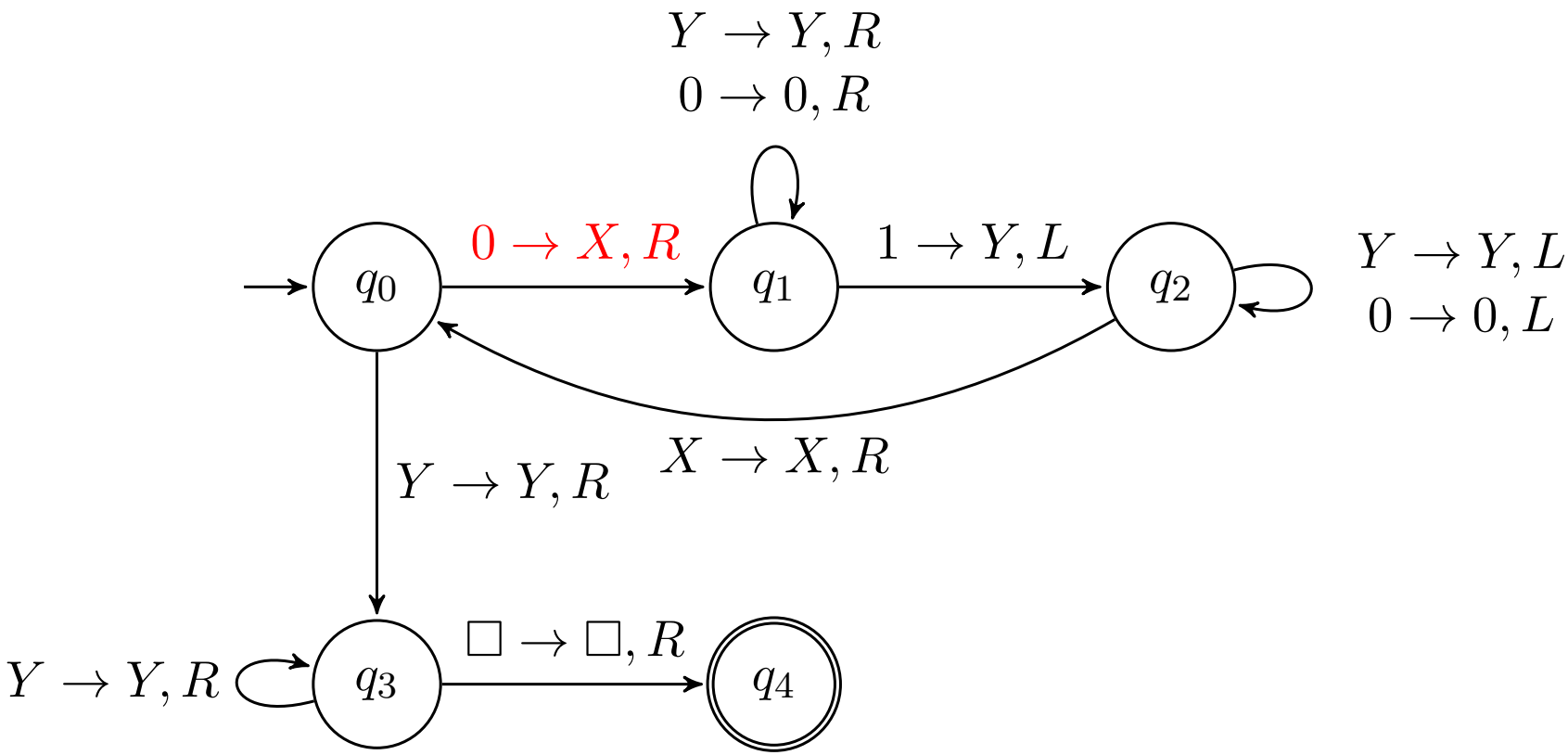




Example for 00111

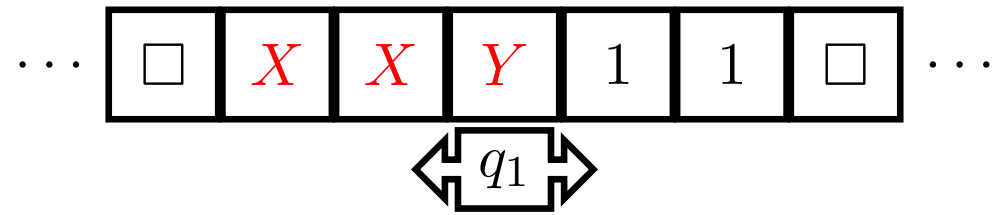
Formalities and examples

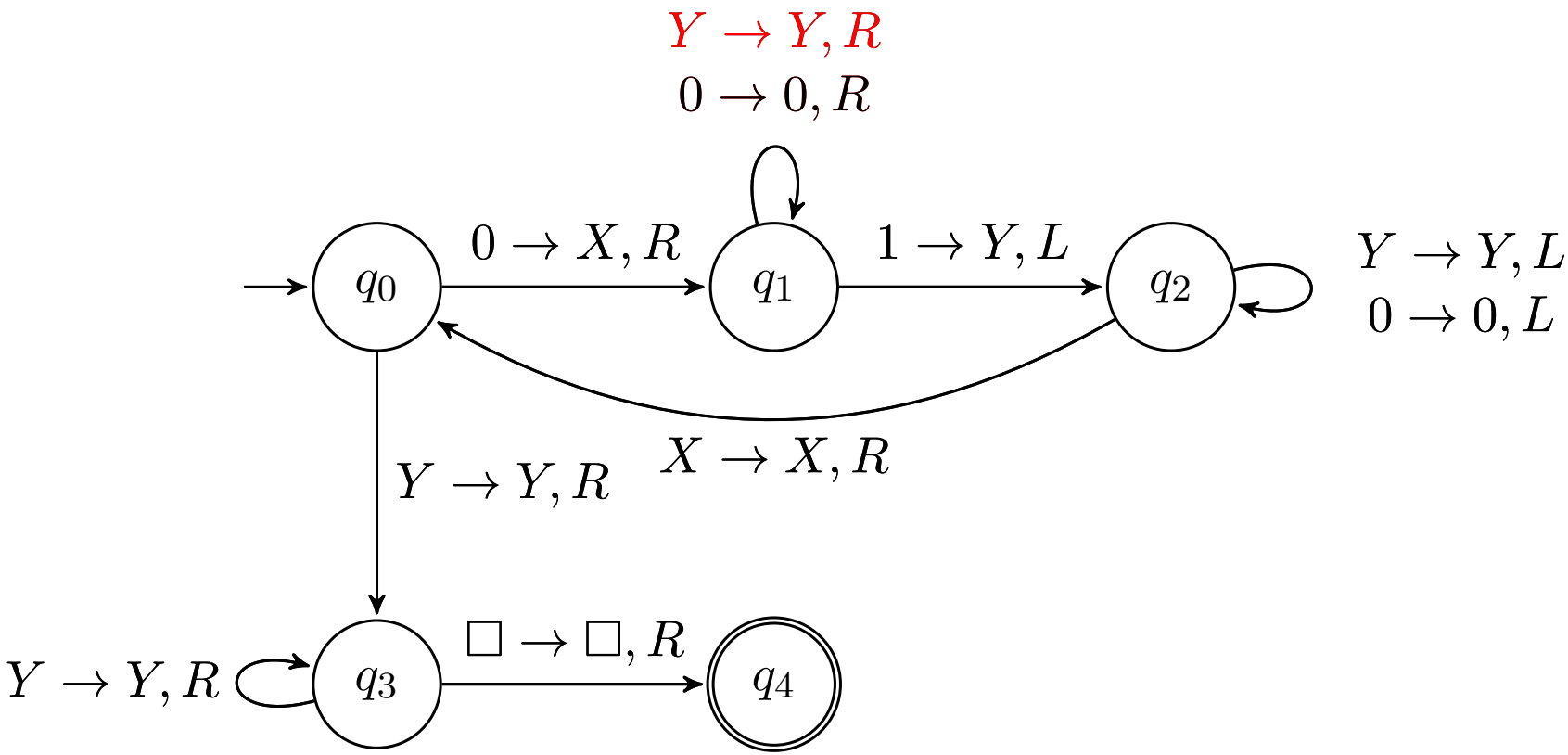




Example for 00111

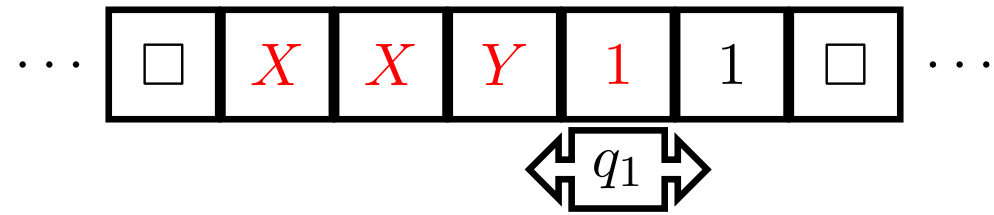
Formalities and examples

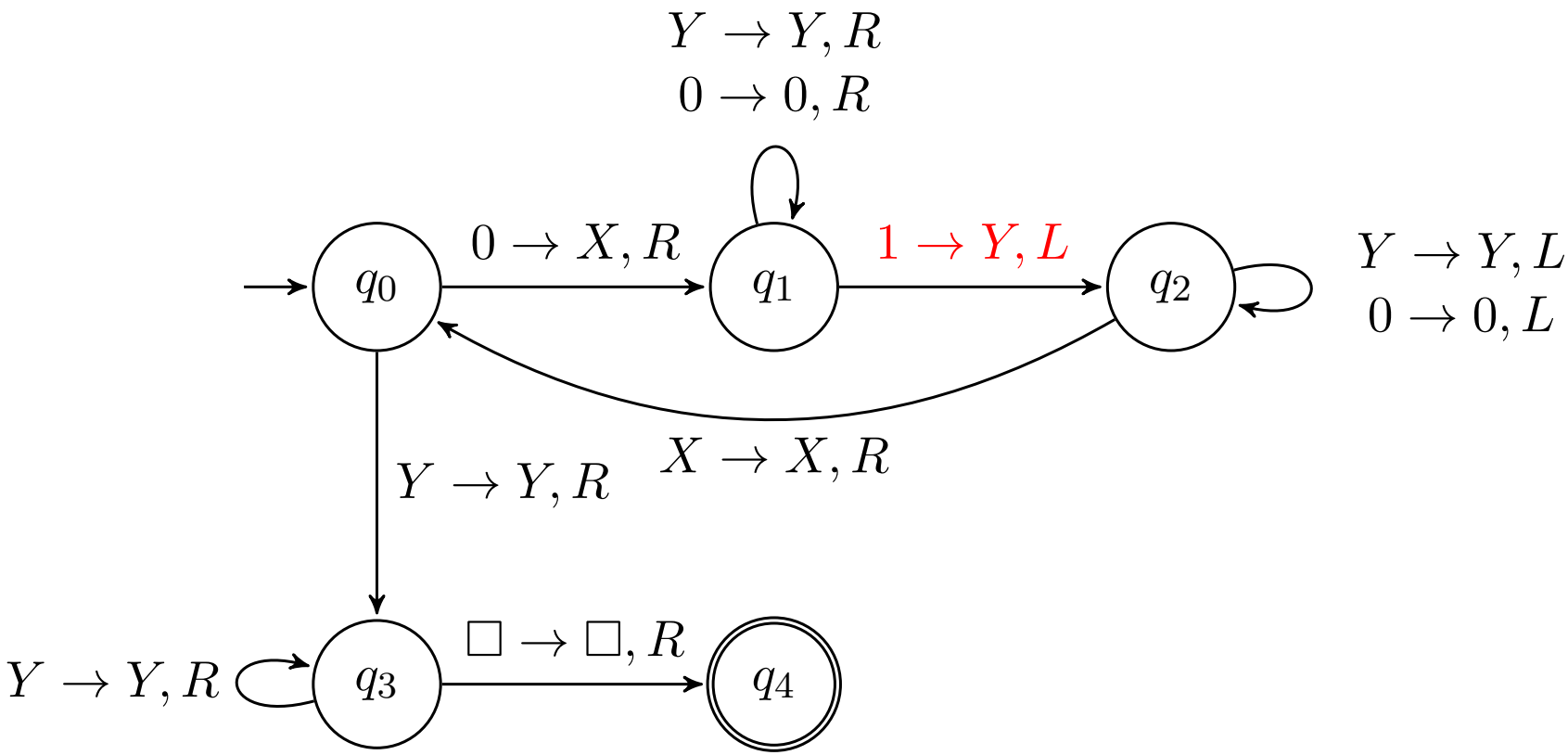




Example for 00111

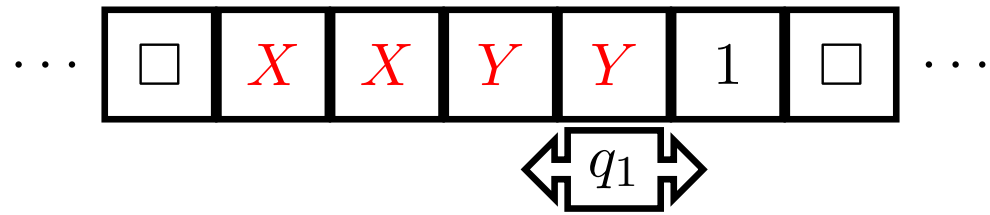
Formalities and examples

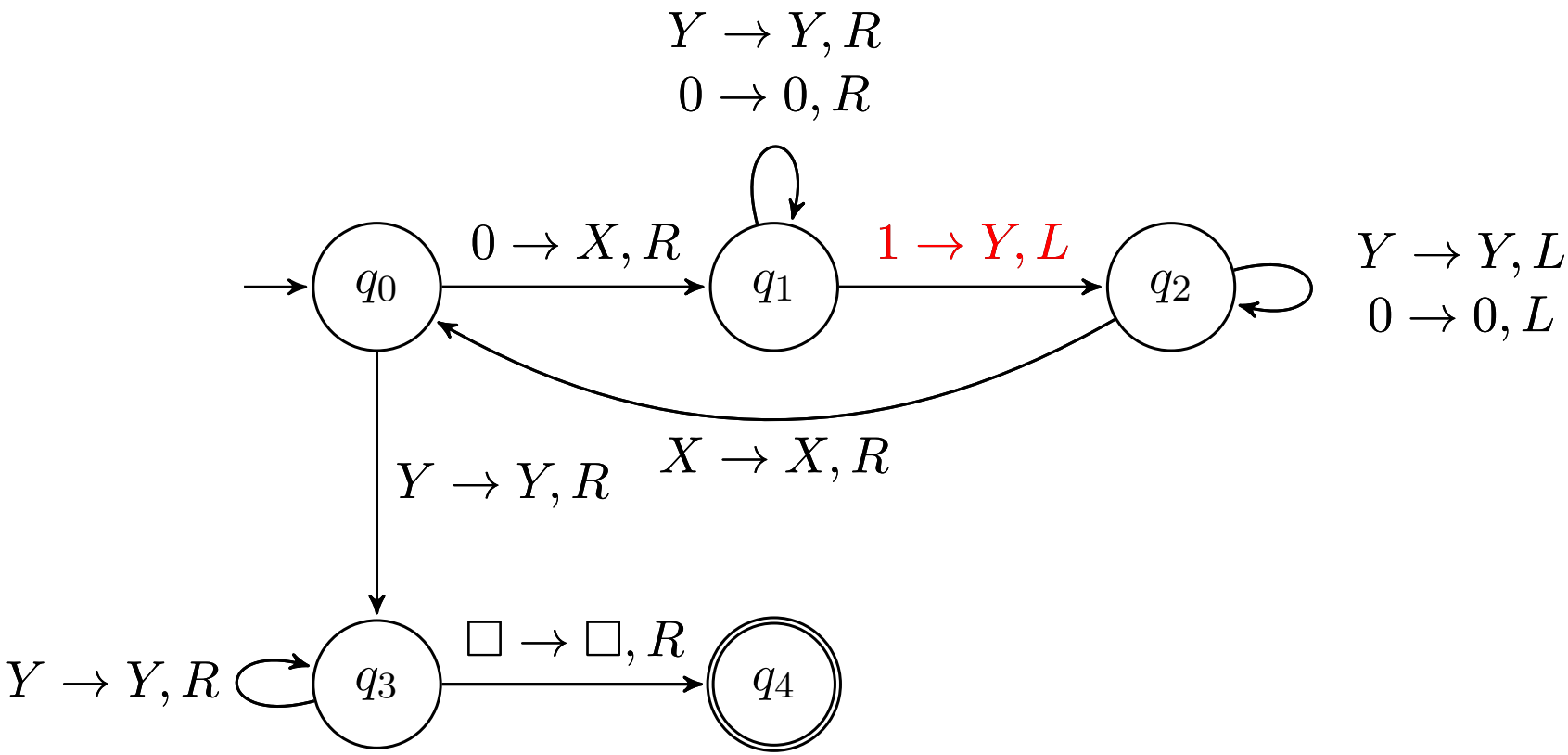




Example for 00111

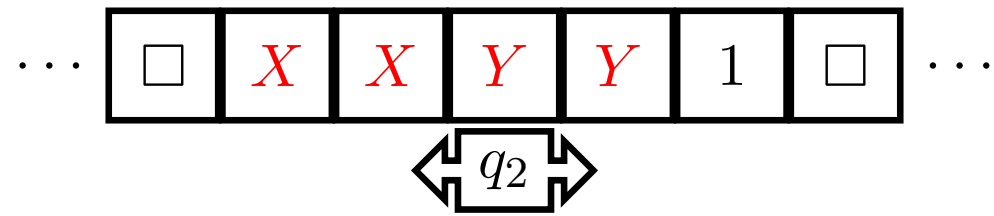
Formalities and examples

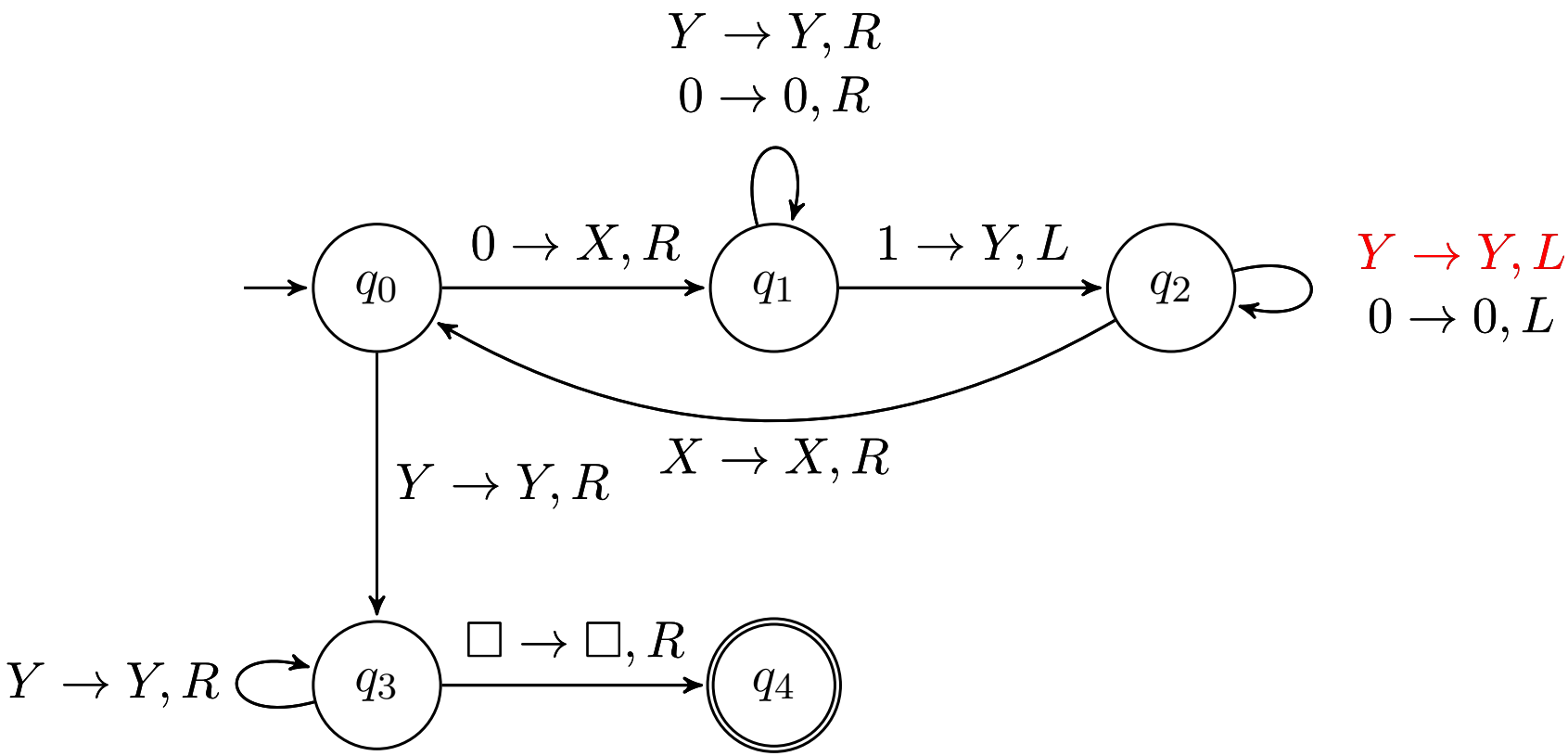




Example for 00111

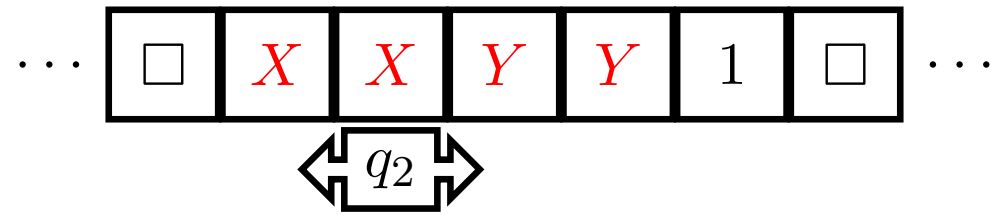
Formalities and examples

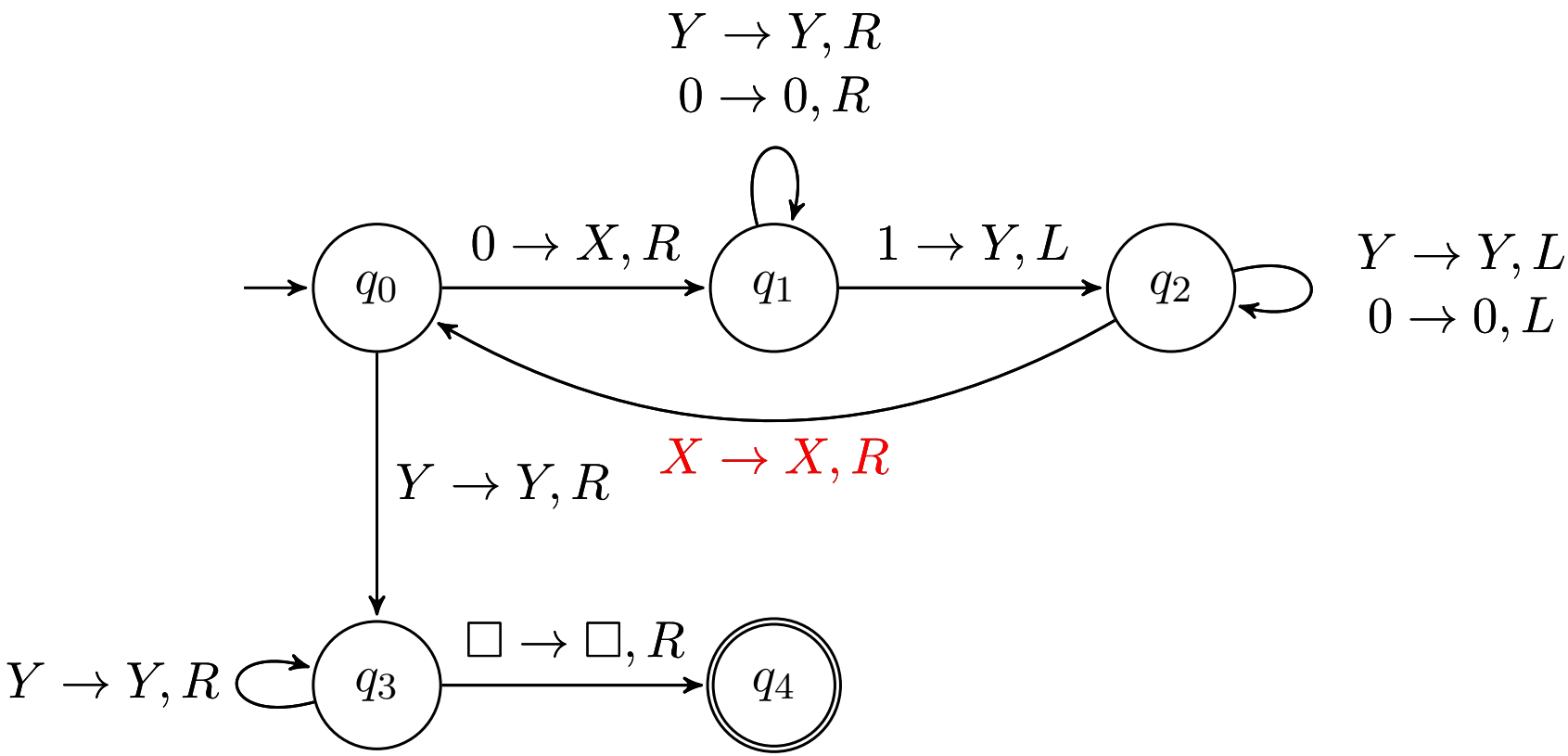




Example for 00111

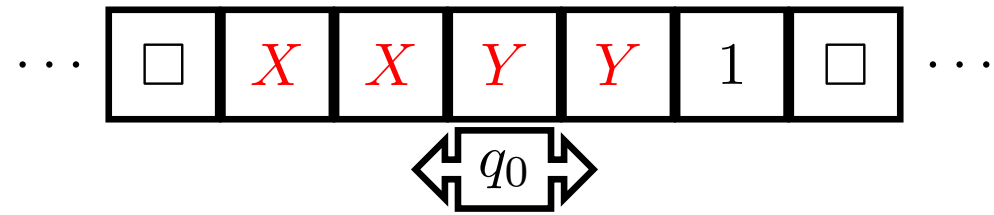
Formalities and examples

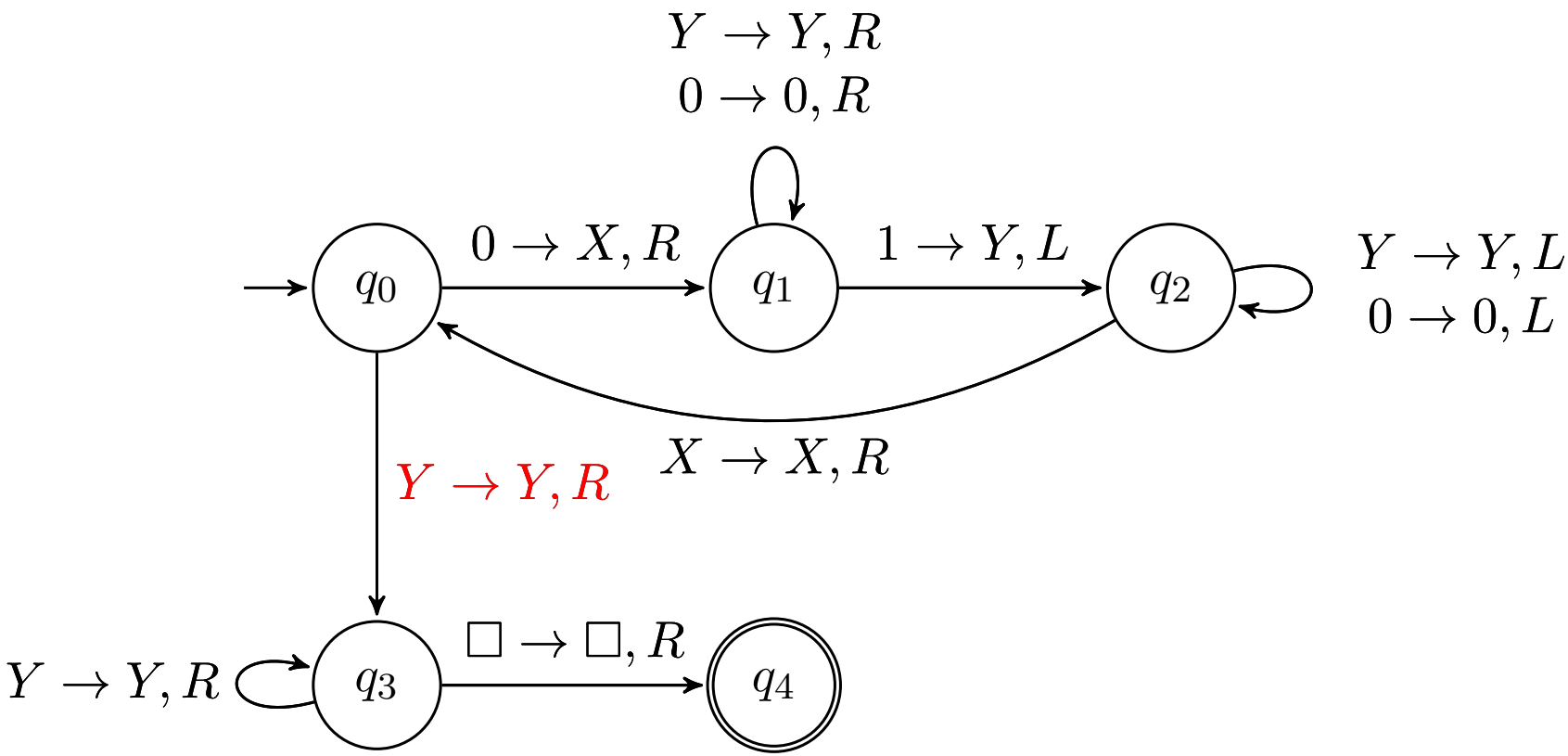




Example for 00111

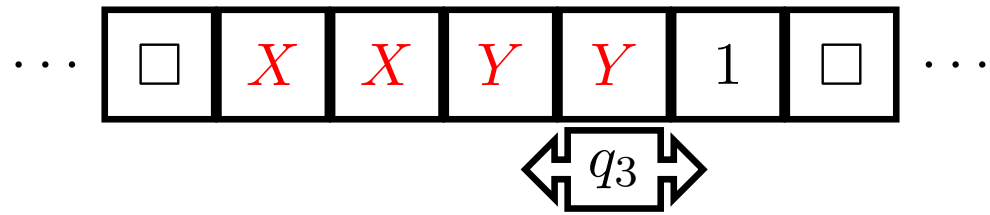
Formalities and examples

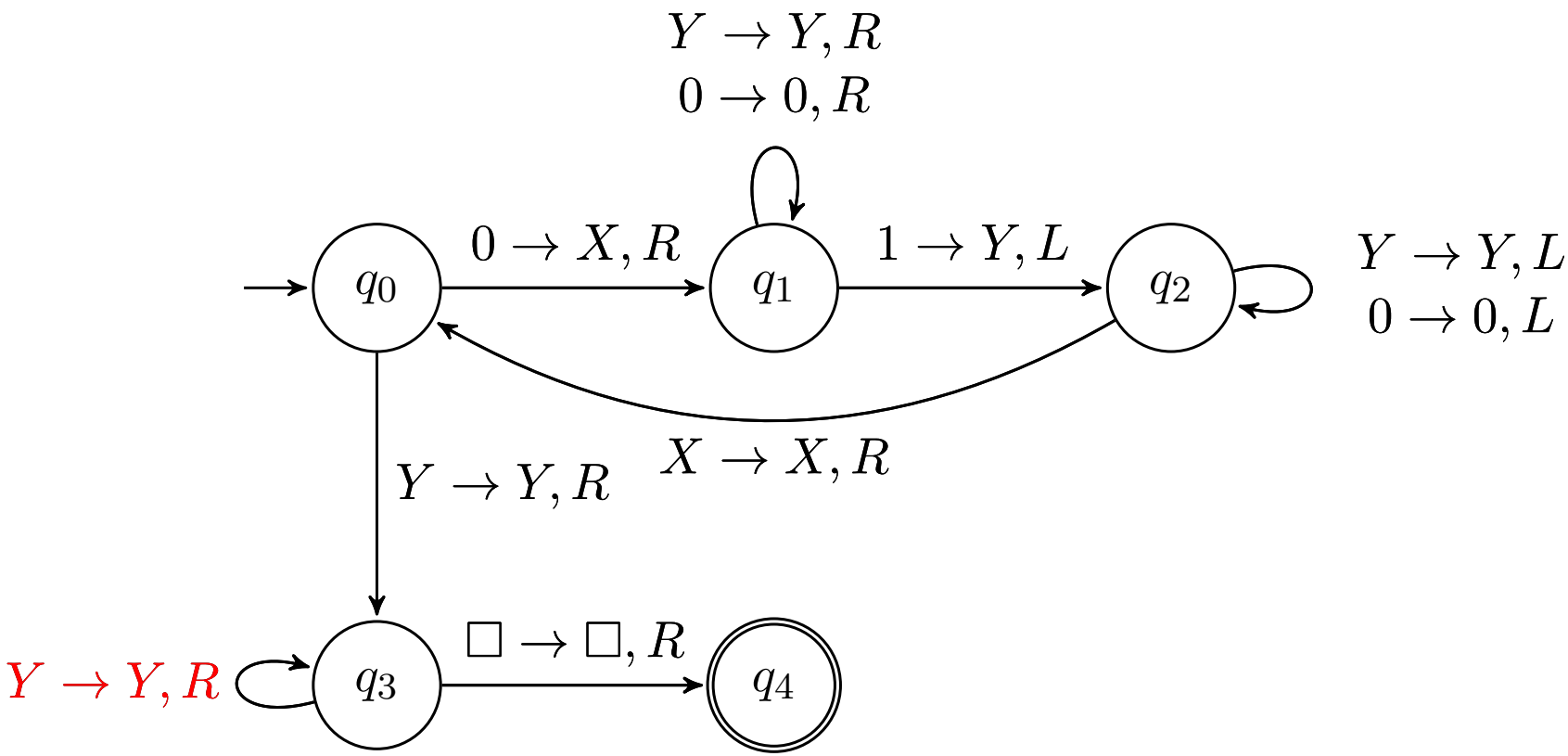




Example for 00111

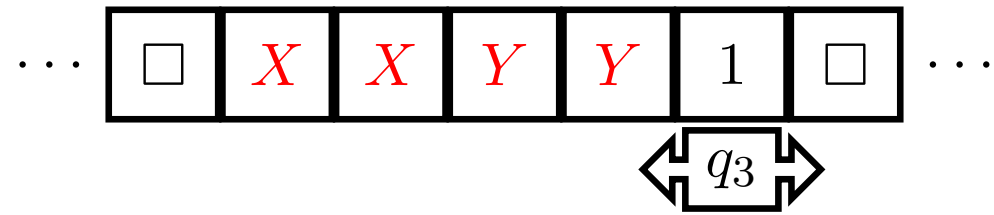
Formalities and examples

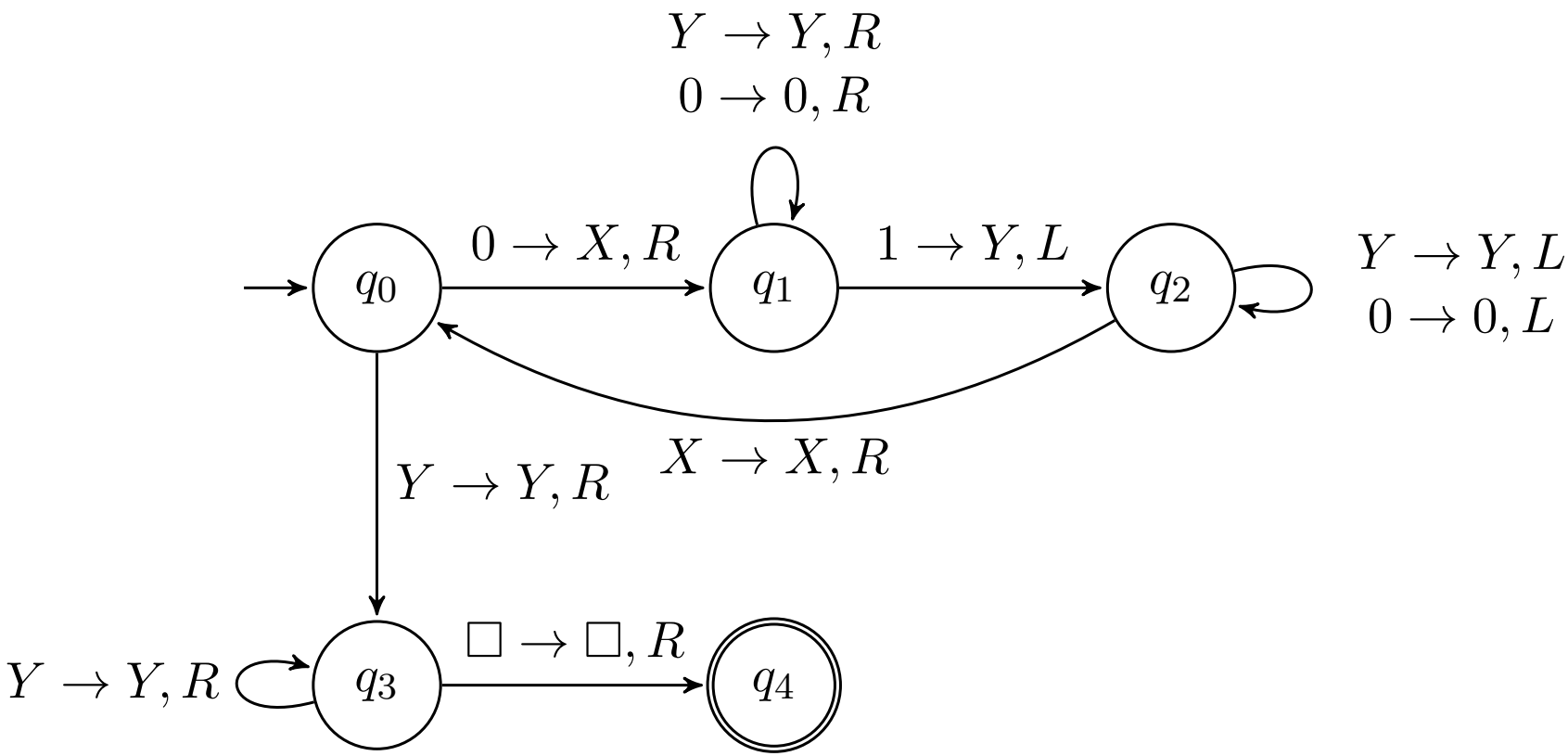




Example for 00111

Formalities and examples





Variants of Turing Machines

There are many alternative definitions of Turing machines. These are called **variants**, and they have the same language recognition functionality as the basic Turing machine (previously defined).

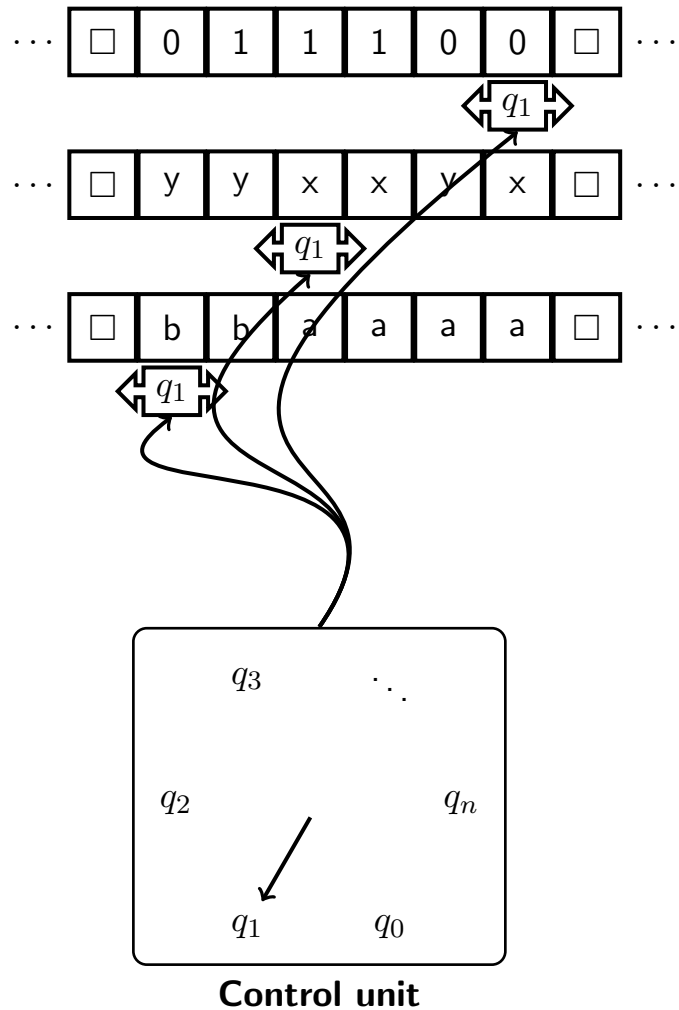
- Multitape Turing Machines.
- Nondeterministic Turing Machines.
- Enumerators.
- Restricted Turing Machines.
- Multistack Machines.
- Counting Machines.
- Computers.

Multitape Turing Machines

Variants of Turing Machines

With a finite control unit and a finite number of tapes.

Each tape is divided in cells (like before), and each cell can contain any symbol of the finite tape alphabet symbol.

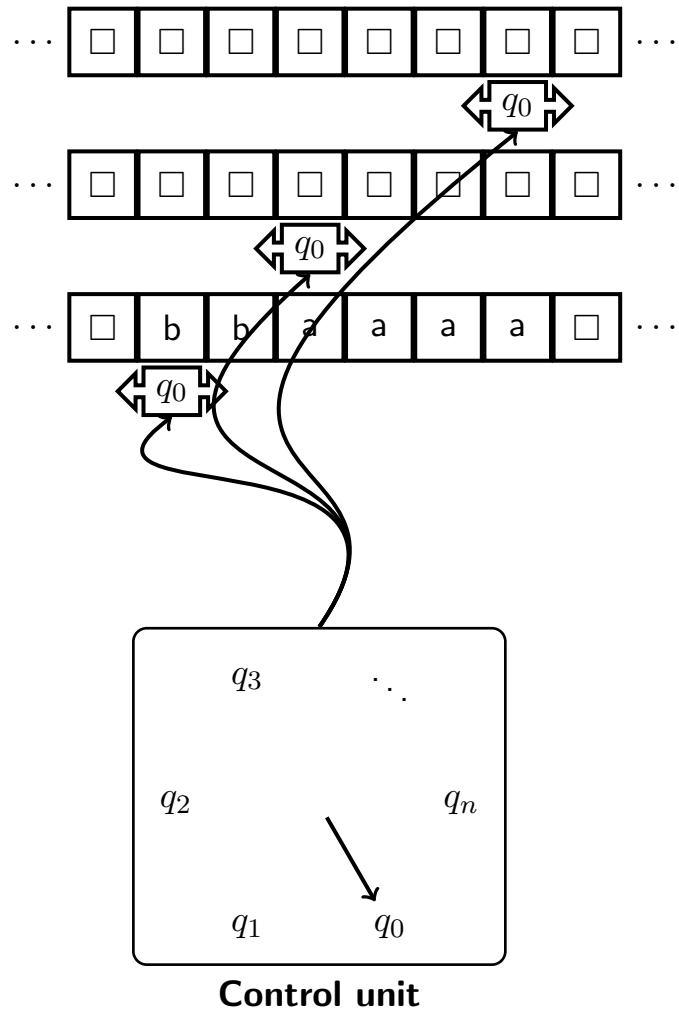


Multitape Turing Machines

Variants of Turing Machines

The set state include an initial state and several acceptance states. Initially:

1. The input is copied to the first tape (like before).
2. All cells, from all tapes contain □.
3. The control unit is in the initial state.
4. The head tape of the other tapes points to any cell.

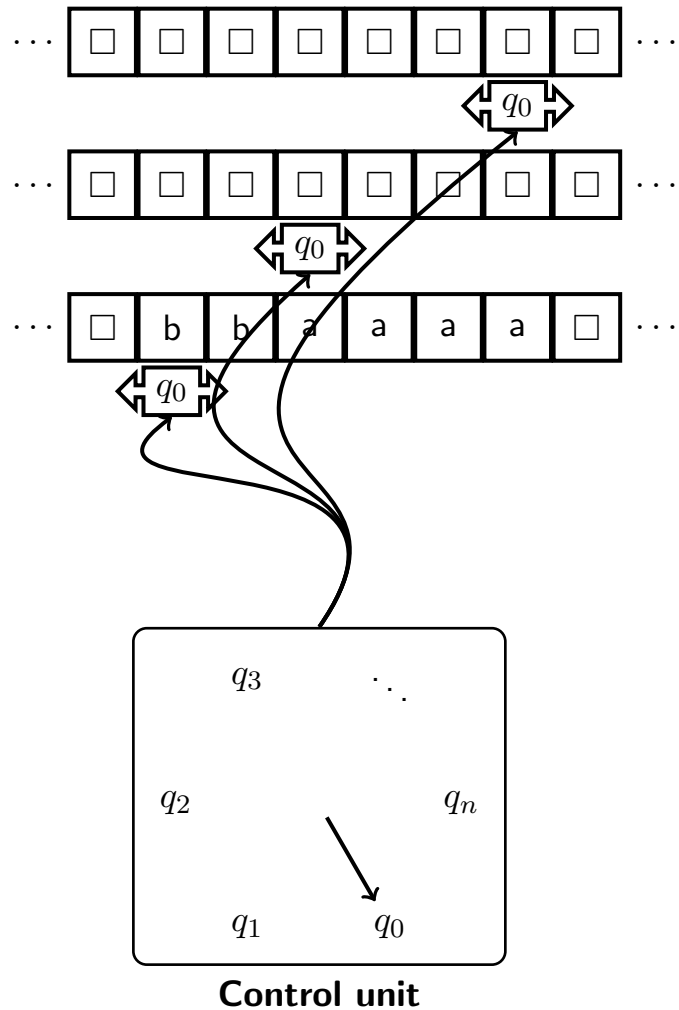


Multitape Turing Machines

Variants of Turing Machines

The set state include an initial state and several acceptance states. Initially:

1. The input is copied to the first tape (like before).
2. All cells, from all tapes contain $\square$.
3. The control unit is in the initial state.
4. The head tape of the other tapes points to any cell.

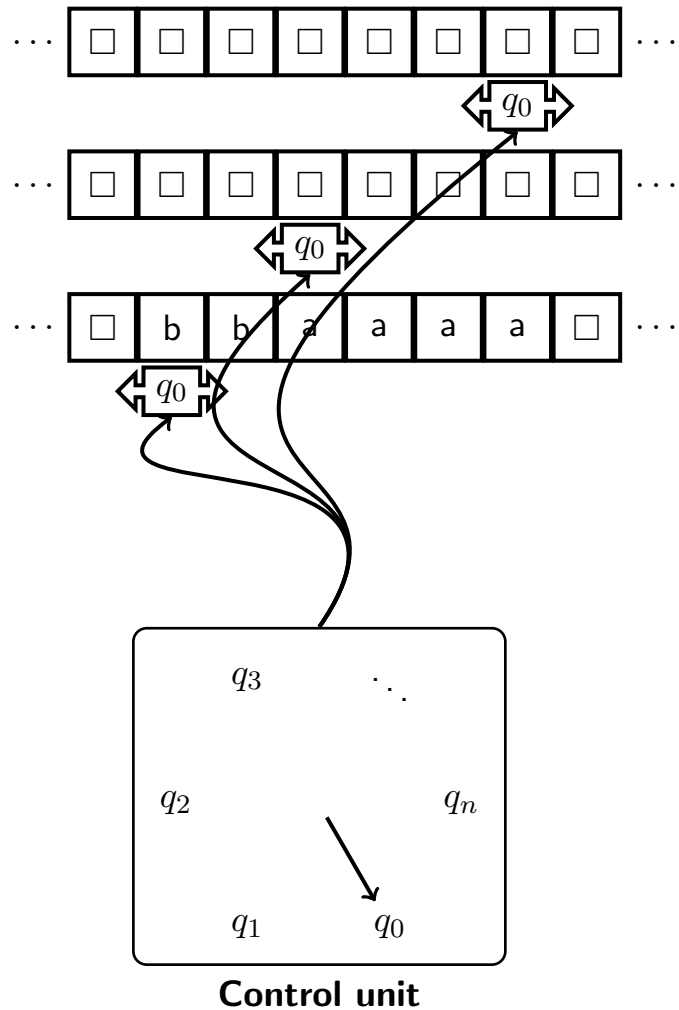


Multitape Turing Machines

Variants of Turing Machines

The set state include an initial state and several acceptance states. Initially:

1. The input is copied to the first tape (like before).
2. All cells, from all tapes contain $\square$.
3. The control unit is in the initial state.
4. The head tape of the other tapes points to any cell.

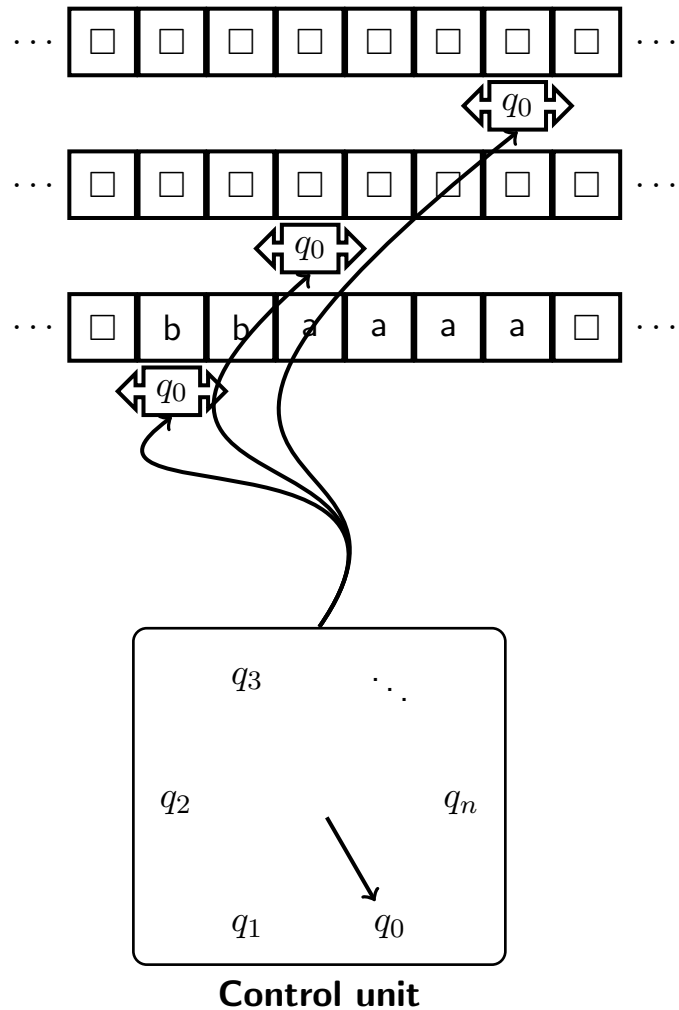


Multitape Turing Machines

Variants of Turing Machines

The set state include an initial state and several acceptance states. Initially:

1. The input is copied to the first tape (like before).
2. All cells, from all tapes contain $\square$.
3. The control unit is in the initial state.
4. The head tape of the other tapes points to any cell.

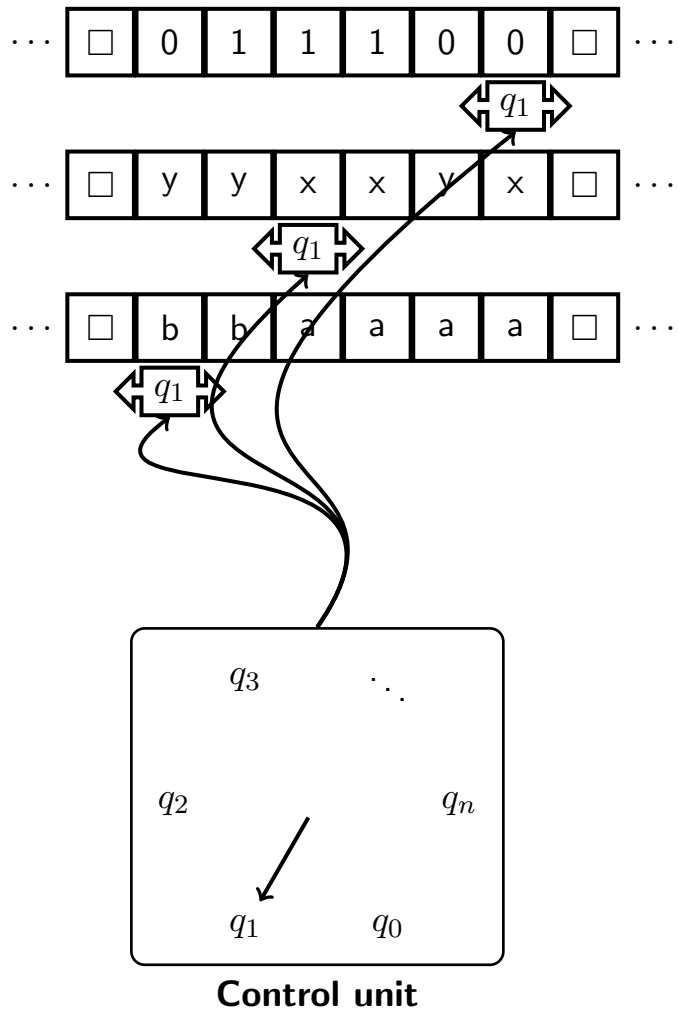


Multitape Turing Machines

Variants of Turing Machines

The movement on the TM depends on the state and the designated symbol by the tape heads of each symbol. Then in a single movement the TM performs:

1. The control unit goes to a new state (it can be the same).
2. A new symbol is written in each cell from all tapes where the tape head is pointing.
3. Each head tape can move to the left (L), to the right (R) or stay (S) in the same cell (each head moves independently, they can move in different directions or not move at all).

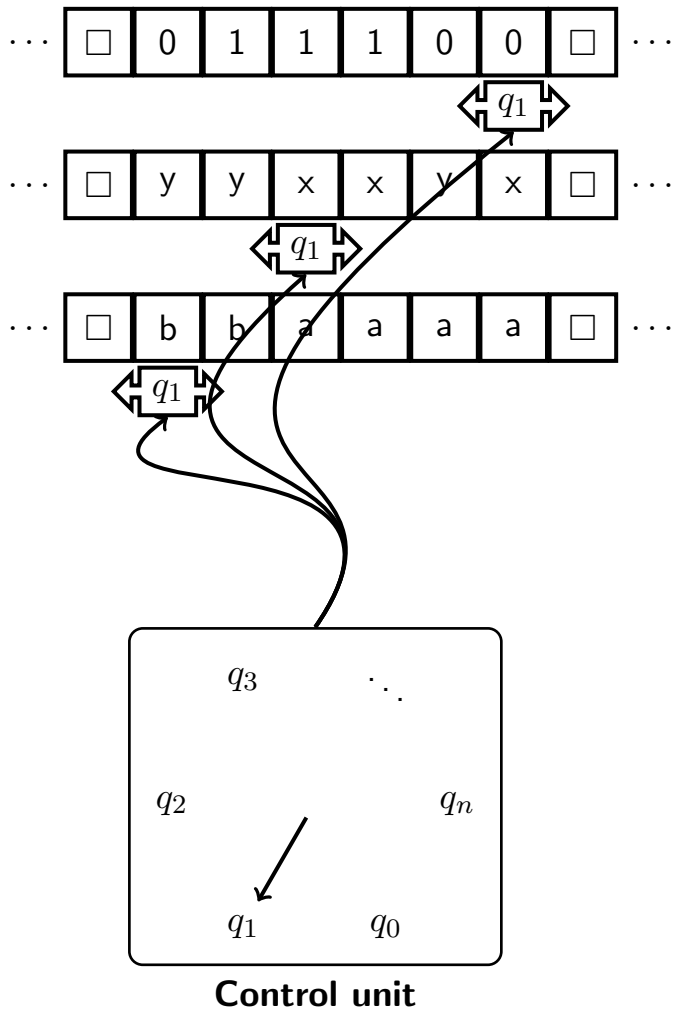


Multitape Turing Machines

Variants of Turing Machines

The movement on the TM depends on the state and the designated symbol by the tape heads of each symbol. Then in a single movement the TM performs:

1. The control unit goes to a new state (it can be the same).
2. A new symbol is written in each cell from all tapes where the tape head is pointing.
3. Each head tape can move to the left (L), to the right (R) or stay (S) in the same cell (each head moves independently, they can move in different directions or not move at all).

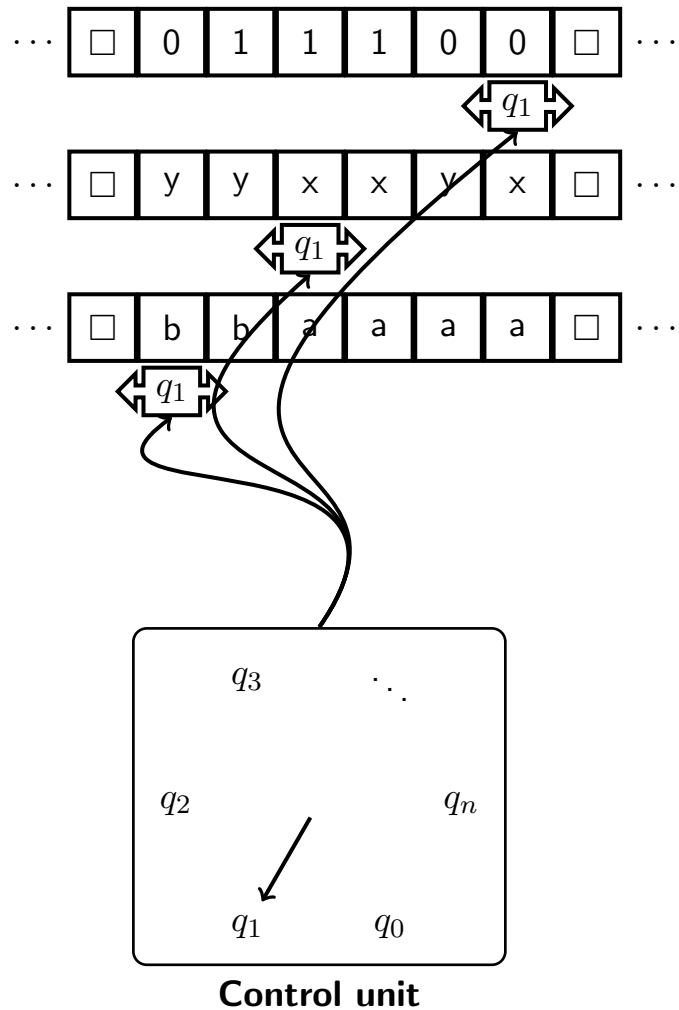


Multitape Turing Machines

Variants of Turing Machines

The movement on the TM depends on the state and the designated symbol by the tape heads of each symbol. Then in a single movement the TM performs:

1. The control unit goes to a new state (it can be the same).
2. A new symbol is written in each cell from all tapes where the tape head is pointing.
3. Each head tape can move to the left (L), to the right (R) or stay (S) in the same cell (each head moves independently, they can move in different directions or not move at all).



Multitape Turing Machines

Variants of Turing Machines

The general idea for a δ function is

- from a given state, and depending on the symbols of each tape, then you perform a replacement in each cell with a symbol, then you perform a movement in each cell. For example, with 3 tapes:

$$\delta(q_0, b, x, 0) \rightarrow (q_1, a, y, 0, L, R, S).$$

So, just because it has more tapes this variant is more powerful than the basic Turing machine? The answer is **no**.

Why or how? Any thoughts?

Multitape Turing Machines

Variants of Turing Machines

The general idea for a δ function is

- from a given state, and depending on the symbols of each tape, then you perform a replacement in each cell with a symbol, then you perform a movement in each cell. For example, with 3 tapes:

$$\delta(q_0, b, x, 0) \to (q_1, a, y, 0, L, R, S).$$

So, just because it has more tapes this variant is more powerful than the basic Turing machine? The answer is **no**.

Why or how? Any thoughts?

Multitape Turing Machines

Variants of Turing Machines

The general idea for a δ function is

- from a given state, and depending on the symbols of each tape, then you perform a replacement in each cell with a symbol, then you perform a movement in each cell. For example, with 3 tapes:

$$\delta(q_0, b, x, 0) \to (q_1, a, y, 0, L, R, S).$$

So, just because it has more tapes this variant is more powerful than the basic Turing machine? The answer is **no**.

Why or how? Any thoughts?

Multitape Turing Machines

Variants of Turing Machines

The general idea for a δ function is

- from a given state, and depending on the symbols of each tape, then you perform a replacement in each cell with a symbol, then you perform a movement in each cell. For example, with 3 tapes:

$$\delta(q_0, b, x, 0) \to (q_1, a, y, 0, L, R, S).$$

So, just because it has more tapes this variant is more powerful than the basic Turing machine? The answer is **no**.

Why or how? Any thoughts?

Multitape Turing Machines

Variants of Turing Machines

Theorem

Every multitape Turing machine has an equivalent single tape Turing machine.

Proof idea. Let us say that we have a multitape TM with k tapes and we have a single tape TM S.

Now, introduce the a new tape symbol into S as a delimiter, e.g. $\#$, we use this symbol to indicate the beginning and/or the end of a tape.

S must keep track of the locations of the heads. Simulate the tape head as a new symbol like $\dot{a}, \dot{x}, \dot{1}$.

Starting from the symbol with the $\dot{a}$ inside the leftmost $\#$ and the first $\#$ found to the right, move to the next symbol inside the 2 $\#$, continue this for k times, then go back to the first symbol with $\dot{a}$. □

Nondeterministic Turing Machines

Variants of Turing Machines

In a **Nondeterministic Turing Machine (NTM)**, at any point in a computation the machine may proceed according to several possibilities.

- A simple Turing machine:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}.$$

- A nondeterministic Turing Machine:

$$\delta : Q \times \Gamma \to \mathscr{P}(Q \times \Gamma \times \{L, R\}).$$

The computation of the NTM is a *tree* whose branches correspond to different possibilities for the machine (like a NFA, remember?).

The NTM accepts the input if one of the branches leads to the accepting state.

Nondeterministic Turing Machines

Variants of Turing Machines

The formal definition of a NTM is similar to its deterministic variant in the same fashion as the NFA is similar to a DFA.

The main difference is that instead of going to one state (with its corresponding operations) is a set of tuples with the states and its operations. For example:

- A simple Turing machine:

$$\delta(q_0, 0) \to (q_1, X, R).$$

- A nondeterministic Turing Machine:

$$\delta(q_0, 0) \to \{(q_1, X, R), (q_2, 1, L), (q_3, Y, R)\}.$$

Nondeterministic Turing Machines I

Variants of Turing Machines

Theorem

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof idea. Any computation of a NTM N is a branch where at least 1 of its branches reaches an accepting state (the path from the starting state to the accepting state is called *accepting configuration*).

Using a breath first search (BFS), a deterministic TM D can explore all branches with the same depth before going on to explore any branch to the next depth. Then D will visit every node in the tree until it encounters an *accepting configuration*.

In order to simulate BFS, D uses 3 tapes. Tape 1 always contains the input string and it is never altered. Tape 2 maintains a copy of N's tape on some

Nondeterministic Turing Machines II

Variants of Turing Machines

branch of its nondeterministic run. Tape 3 keeps track D's location in N's nondeterministic computation tree.

In tape 3 some string ID is used to control the depth and the children for each node. Starting from the the root node, a node 21 could mean, from the 2nd child, going to that node's 1st child, then explore 22 (2nd child, going to that node's 2nd child), and so on to $2b$, where b is the number of possible offspring of node 2, if no accepting configuration is found, continue to 31 and so on.

If no accepting configuration is found continue to expanding 221, continue until an *accepting configuration* is found.

Nondeterministic Turing Machines III

Variants of Turing Machines

For your information, the main algorithm would look something like this:

1. Initialise tape 1 with w, tape 2 and 3 are empty.
2. Copy tape 1 to tape 2.
3. Use tape 2 as N with input w. Before each step of N consult the next symbol on tape 3 to determine which choice to make among those allowed by N's transition function.
   - If no symbols remain on tape 3 or if this choice is invalid, abort the branch and go to step 4.
   - If a rejecting configuration is encountered go to step 4.
   - If an accepting configuration is found, *accept* the input.
4. Replace the string on tape 3 with the lexicographically next string. Simulate the next branch of N's computation by going to step 2.

Finally, since we know that every multitape TM has an equivalent single tape Turing machine, we can conclude that any NTM represented by a multitape TM has an equivalent single tape Turing machine. □

Enumerators

Variants of Turing Machines

A Turing machine with a *printer* attached. The TM uses this printer as an output device to print strings.

Every time the TM wants to add a string to the tape, it sends the string to the printer. Some properties of an enumerator E are:

- An E starts with a blank tape.
- If E does not halt, it may print an infinite list of strings.
- The language enumerated by E is the collection of all the strings that it eventually prints.
- E may generate the strings of the language in any order, possibly with repetitions.

Enumerators I

Variants of Turing Machines

Theorem

A language is Turing-recognisable if and only if some enumerator enumerates it.

Proof idea. Let us say that an enumerator E enumerates a language L, and L is recognisable by a TM M.

$M =$ "On input $w \in L$:

1. Run E. Every time that E outputs a string, compare it with w.
2. If w ever appears in the output of E, *accept*".

Clearly, M accepts those strings that appear on E's list. Now, if M recognises L, an E can be defined for L. Say that $s_1, s_2, s_3, \ldots$ is a list of all possible strings in Σ^*.

$E =$ "Ignore the input:

1. Repeat the following forever.

Enumerators II

Variants of Turing Machines

2. Run M for i steps on each input $s_1, s_2, s_3, \ldots$.
3. If any computations *accept*, print out the corresponding s_i".

If M accepts a particular string s_i, eventually it will appear on the list generated by E. Another way to look at it is that M is running in parallel on all possible input strings. □

Restricted Turing Machines

Variants of Turing Machines

There are TM with some kind of restriction on one of its components. For example:

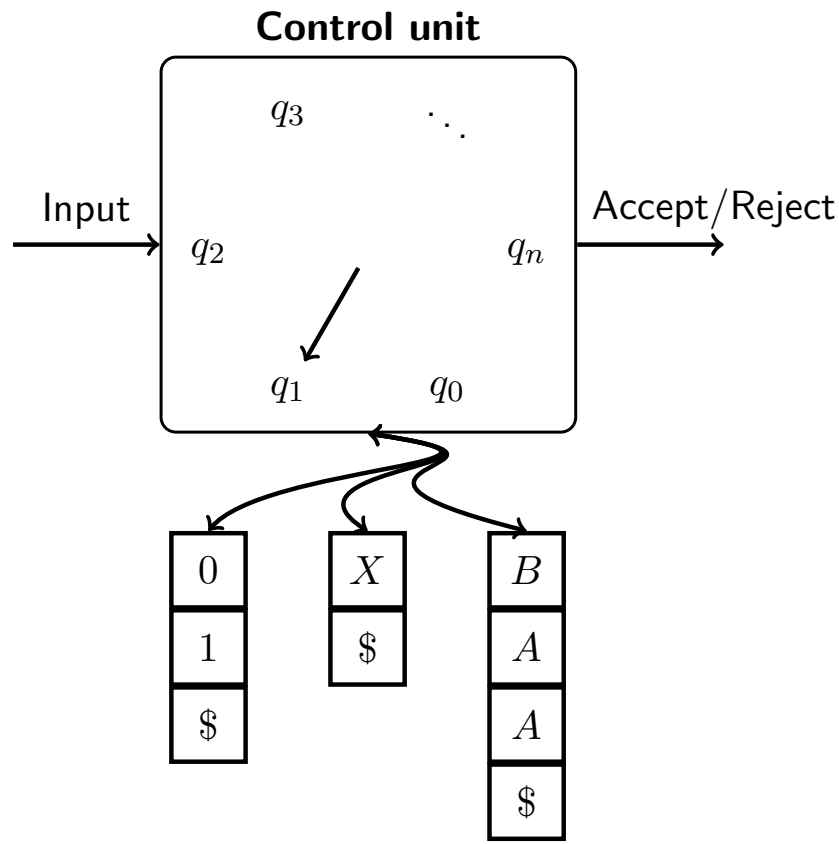
- The movements it can perform (no movements to the left or right).
- The symbols used.
- Simulate the tape with stacks.
- Make the tape only process integer numbers.
- Make the machine only recognise 0s or different.

Multistack Machines

Variants of Turing Machines

A TM can accept languages that are not accepted by a pushdown automata.

But if we add another stack to the pushdown automaton, then the pushdown automaton can accept any language accepted by a TM.

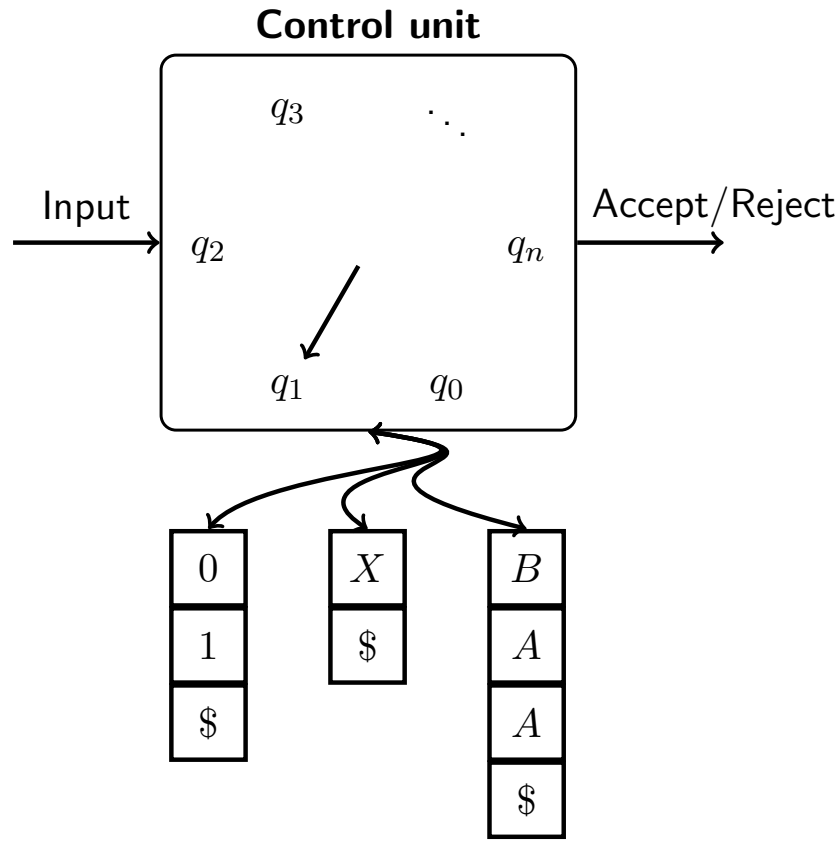


Multistack Machines

Variants of Turing Machines

A multistack machine with k stacks is a deterministic pushdown automata with k stacks with the following properties:

- Process the input in the same way as PA.
- It has a control unit.
- It has a stack alphabet (used by all stacks).



Multistack Machines I

Variants of Turing Machines

A transition in this multistack machine is based on the following:

1. A state from the control unit.
2. A symbol from the input is read, ε can be used as an input but no other transition ε can be performed at the same time.
3. The top symbol of each stack.

In a single movement, the multistack can:

1. Change of state.
2. Replace the top symbol of each stack and/or push symbols on each stack.

Multistack Machines II

Variants of Turing Machines

For example, a transition for a multistack machine of k stacks could be:

$$\delta(q_0, a, X_1, X_2, \ldots, X_k) = (q_1, \gamma_1, \gamma_2, \ldots, \gamma_k).$$

It can be read as "*from state q_0, reads a from the input, pops the top stack symbols X_i from stack i, for all stacks $i = 1, 2, \ldots, k$, then go to state q_1 and push the stack symbol γ_i on stack i, for all stacks $i = 1, 2, \ldots, k$*".

The multistack machine accepts the input if it reaches the accepting state.

Multistack Machines I

Variants of Turing Machines

Theorem

If a language is Turing-recognisable, then the language is accepted by a two stacks machine.

Proof idea. The main idea is to simulate the TM tape with two stacks. Store in 1 stack what it is stored at the left side of the tape head and on the other what it is stored at the right side of the tape head (the $\square$ at the left of the first symbol and the $\square$ at the right of the last symbol are not stored).

1. Stack 1 and 2 starts with $\$$ (the $\square$ at the left of the first symbol and the $\square$ at the right of the last symbol, respectively).
2. Push the input into Stack 1 $w\$$ (the input is in the reverse order).
3. Pop w from Stack 1, and push it into Stack 2 (the input is in the correct order).

Multistack Machines II

Variants of Turing Machines

4. The element on the top of the Stack 1, represents the element read by the tape header of the TM.
   - If you move to the left, pop what it is on the top of Stack 1 and push it to Stack 2, the element on the top of Stack 1 now represents the element on the left of the pop element.
   - If you move to the right, pop what it is on the top of Stack 2 and push it to Stack 1, the element on the top of Stack 1 now represents the element on the right of the previous element that was on the top of the Stack 1.
5. A change on the tape in the TM is represented by a simultaneous pop and push on the top of Stack 1.

As you can see a multistack with 2 stacks simulated perfectly the definition of a TM. □

Counting Machines

Variants of Turing Machines

A counting machine can be defined in two ways:

1. A multistack machine, each stack works as a counter. The machine can only store non-negative integer and it can only recognise 0 or not. The movement in this machine depends on its state, input symbol and the counters (if they exist). In a movement the machine can:
   1. Change of state.
   2. add or subtract 1 from any counter independently (it cannot subtract 1 from 0, no negative allowed).
2. A restricted multistack machine with restriction:
   1. Only two stack symbols exist, $\$$ (empty stack symbol) and some other, e.g. X.
   2. Initially, $\$$ is each the stack.
   3. $\$$ can only be replaced with a string with the form $X^i\$$, for $i \geq 0$ are allowed.
   4. X can only be replaced with X^i, for $i \geq 0$. $\$$ is at the bottom of the stack and the remaining ones (if they exist) are X.

Counting Machines I

Variants of Turing Machines

Theorem

All language accepted by a TM is accepted by a machine with three counters.

Proof idea. Imagine for a second how do you use a counter in any programming language, define an integer variable $X = 0$, then you add or subtract by one depending on certain conditions.

The current value for X is 0, if $X = X + 1$, then $X = 1$, if $X = X + 1$, then $X = 2$. This kind of operations simulate the operations pop and push of a stack. If 0 is at the top of the stack, you replace it with 1, if 1 is at the top of the stack, then you replace it with 2.

Counting Machines II

Variants of Turing Machines

Two counters (stacks) are used to store the integers represented as symbols, and the third one is used to adjust the other two counters. In particular, the third counter is used for multiplications and divisions by X.

This machine can be transformed to a machine with two counters by simulating 3 counters with two, and since we know that a language that is recognisable by a two stacks machine can be recognisable by a TM, then the language recognisable by a machine with two counters can be recognisable by a TM. □

Computers

Variants of Turing Machines

The main idea from this section is to show that

1. A computer (the one that normally you use) can simulate a Turing machine.
2. A TM can simulate a computer and can do it in polynomial time (to be seen in **complexity theory**, $T(n) = O(n^k)$ for some positive constant k), with respect to the steps used for the computer.

Lemma

A computer (the one that normally you use) can simulate a Turing machine.

Proof idea. It is necessary to design an algorithm that acts or do what the TM does. *That's it*.

Given that there is a finite set of states (can be coded as strings) and a finite number of transitions, the algorithm can use the transition to perform the required tasks (**any idea how?**). The tape can be coded as a string with finite length.

The only problem with this approach is that the computer does not have an infinite memory like the tape in the TM (**any idea how can we avoid this problem?**). □

Computers I

Variants of Turing Machines

Lemma

A TM can simulate a computer and can do it in polynomial time, with respect to the steps used for the computer.

Proof idea. What do we need to simulate a computer?

- Storage, a place to store an unlimited number of strings (normally with size 32 or 64 bits), with some *address* associated to them so we can access to them.
- Memory, since we need some address to access the words, we need a way to store some machine or assembler language, this way the computer can move, add, remove, replace data.
- Each instruction implies a finite number of strings and each instruction modifies the value of one words (at most).

Computers II

Variants of Turing Machines

- Registers, words stored in memory that can be accessed faster (like addition).

Using a multi-tape TM one can simulate:

1. Represent the storage of the computer. You can include any format to represent the elements saved there like binary, add some delimiters to represent the beginning and the end of the elements stored there like the addresses and content, different delimiters to distinguish between a content and an address.
2. Instructions counter, use it to store the positions of the memory in tape 1. You can use it to know which instruction or operation is next.
3. Memory address, stored the address of the element wanted from tape 1, once the instruction is executed and if the elements change, then the content is modified using this address.

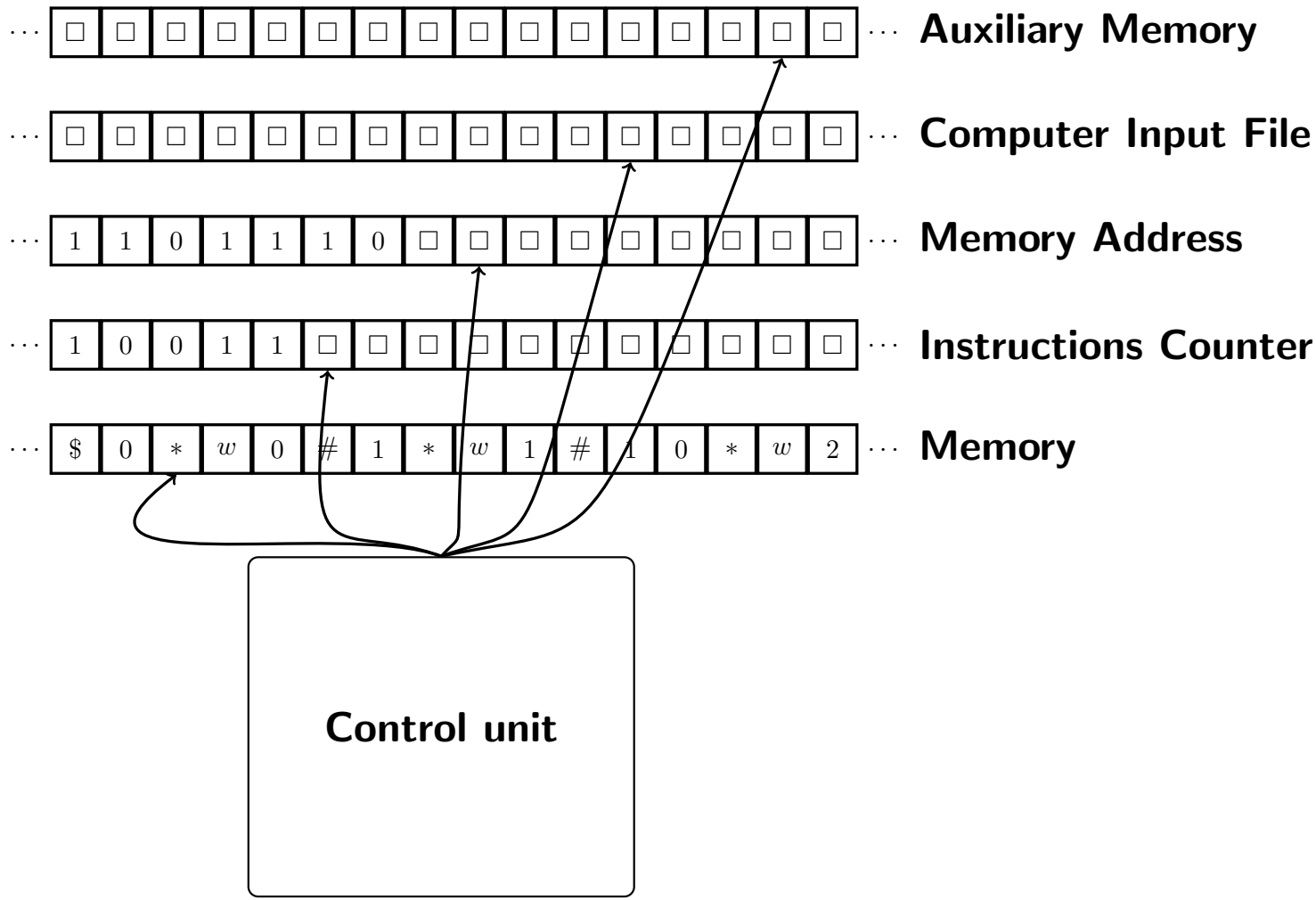
Computers III

Variants of Turing Machines

4. Store the input of the computer, initial state, previous state, etc.
5. Some auxiliary memory to store and/or use some instructions for the computer like arithmetic operations like multiplication. □

Computers

Variants of Turing Machines



Computers

Variants of Turing Machines

Theorem

If a computer:

1. *Has only instructions that increases the maximum length of a word in, at most 1, and*
2. *Has only instructions that only a multi-tape TM can execute on strings with length k in $O(k^2)$ steps or less,*

then the TM described previously can simulate the n steps of a computer in $O(n^3)$ steps.

Theorem

The computer described in the previous theorem can be simulated in n steps by a TM with one tape, using as maximum $O(n^6)$ steps.

Any ideas? Think about it.