Macchine di Turing per calcolare funzioni

18 aprile 2023

Oggi

- Obiettivo: diversi «usi» della MdT
  - MdT riconosce un linguaggio
  - MdT decide un linguaggio
  - MdT calcola una funzione
  - MdT enumera stringhe di un linguaggio

Linguaggio riconosciuto da una MdT

Definizione

Sia $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ *una MdT. Il linguaggio* $L(M)$ *riconosciuto da* M*, è l'insieme delle stringhe che* M *accetta:*

$$L(M) = \{w \in \Sigma^* \mid \exists u, v \in \Gamma^* \; q_0 w \rightarrow^* u q_{accept} v\}.$$

Quindi

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

Parola accettata o rifiutata

Definizione

Una MdT M accetta una parola $w \in \Sigma^*$ se esiste una computazione $C \to^* C'$, dove $C = q_0 w$ è la configurazione iniziale di M con input w e $C' = u q_{accept} v$ è una configurazione di accettazione.

Una MdT M rifiuta una parola $w \in \Sigma^*$ se esiste una computazione $C \to^* C'$, dove $C = q_0 w$ è la configurazione iniziale di M con input w e $C' = u q_{reject} v$ è una configurazione di rifiuto.

Risultati di una computazione

Tre possibili Risultati computazione:

1. M accetta – se si ferma in q_{accept}
2. M rifiuta – se si ferma in q_{reject}
3. M cicla/loop – se non si ferma mai

Mentre M funziona non si può dire se è in loop; si potrebbe fermare in seguito oppure no.

Decidere

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

$$R(M) = \{w \in \Sigma^* \mid M \text{ rifiuta } w\}$$

In generale $L(M) \cup R(M)$ non coincide con Σ^*.

Se $L(M) \cup R(M) = \Sigma^*$, allora M si arresta su ogni input.

In tal caso M è chiamata un decisore (o decider) ed $L(M)$ è il linguaggio deciso da M.

Dal punto di vista delle macchine

Definizione

Una MdT $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ *è un* **decisore** *(o* **decider***) se, per ogni* $w \in \Sigma^*$*, esistono* $u, v \in \Gamma^*$ *e* $q \in \{q_{accept}, q_{reject}\}$ *tali che*

$$q_0 w \rightarrow^* uqv$$

Definizione

Una MdT M **decide** *un linguaggio L se M è un decisore e L = L(M).*

In tal caso L è **deciso** *da M.*

Dal punto di vista dei linguaggi

Definizione

Un linguaggio $L \subseteq \Sigma^*$ è Turing riconoscibile se esiste una macchina di Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ tale che:

1. *M riconosce L*
   (cioè $L = L(M) = \{w \in \Sigma^* \mid \exists u, v \in \Gamma^* \; q_0 w \rightarrow^* u q_{accept} v\}$).

Definizione

Un linguaggio $L \subseteq \Sigma^*$ è decidibile se esiste una macchina di Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ tale che:

1. *M riconosce L*
   (cioè $L = L(M) = \{w \in \Sigma^* \mid \exists u, v \in \Gamma^* \; q_0 w \rightarrow^* u q_{accept} v\}$).
2. *M si arresta su ogni input (cioè per ogni $w \in \Sigma^*$, $q_0 w \rightarrow^* uqv$ con $q \in \{q_{accept}, q_{reject}\}$).*

Se L è decidibile in particolare è riconoscibile; ma non viceversa.

Riconosce o calcola?

Sia $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ *una MdT. Il linguaggio* $L(M)$ *riconosciuto da* M*, è l'insieme delle stringhe che* M *accetta:*

$$L(M) = \{w \in \Sigma^* \mid \exists u, v \in \Gamma^* \; q_0 w \rightarrow^* u q_{accept} v\}.$$

In questa definizione ciò che conta è il raggiungimento dello stato q_{accept} , più che il contenuto finale del nastro, cioè *uv*.

Ribaltando questa visione, ci interesseremo adesso al contenuto finale del nastro, più che allo stato di arrivo, in modo che la MdT calcoli qualcosa.

Funzioni calcolabili

Le MdT possono essere utilizzate per il calcolo di funzioni.

Definizione

Una funzione $f : \Sigma^* \to \Sigma^*$ *è calcolabile se esiste una macchina di Turing* $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ *tale che*

$$\forall w \in \Sigma^* \quad q_0 w \to^* q_{accept} f(w)$$

Funzioni calcolabili

Quindi, una funzione $f : \Sigma^* \to \Sigma^*$ è calcolabile se esiste una macchina di Turing M che, su qualsiasi input w, si ferma avendo solo $f(w)$ sul nastro.

- Nota: in questo caso la MdT deve arrestarsi su ogni input.
- Possibile variante: nessun vincolo sulla posizione della testina nella configurazione di arresto.
- Possibile variante: nessuna distinzione tra q_{accept} e q_{reject}.

Funzioni calcolabili

- Definire una macchina di Turing deterministica M che calcoli la funzione $f(x, y) = x + y$, con x, y interi positivi.
- Si assuma che l'input sia della forma

$$w0z$$

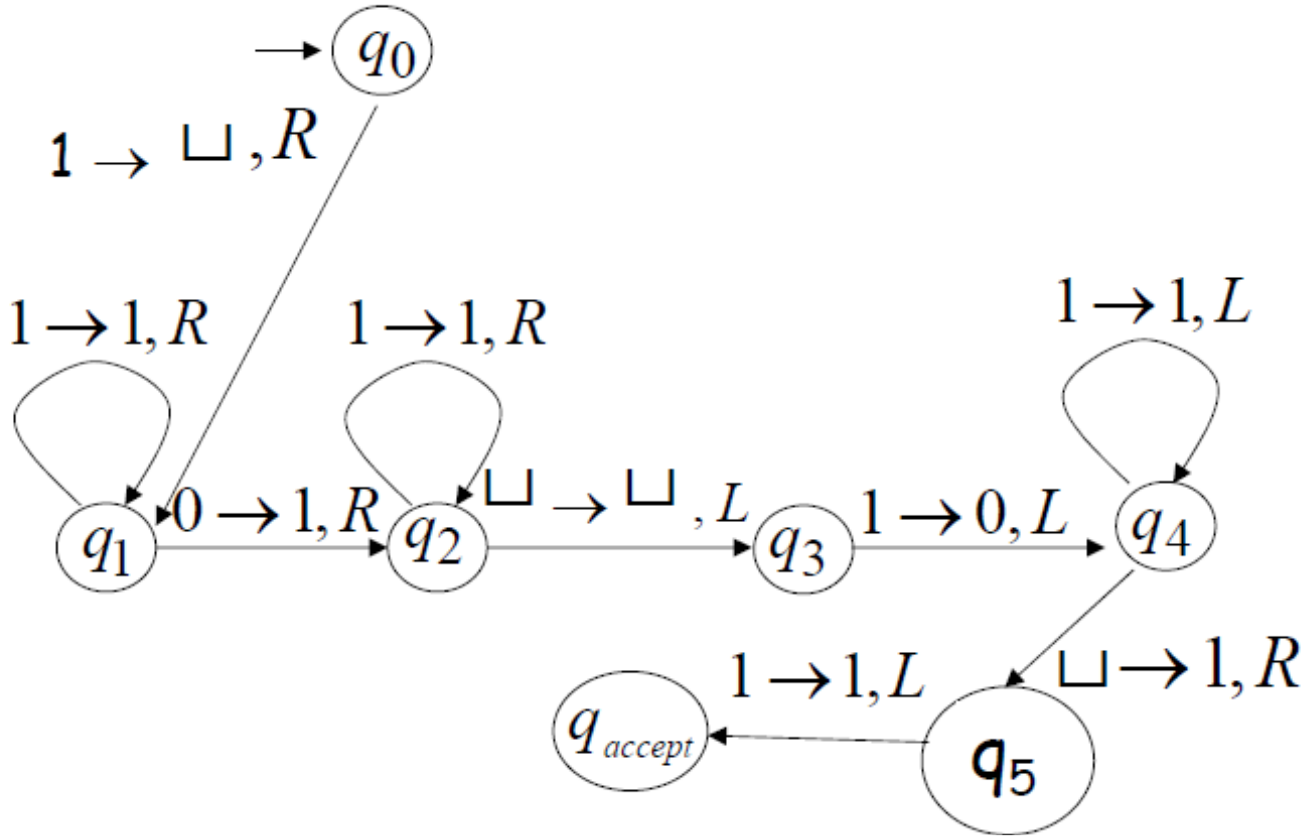
dove w è la rappresentazione unaria di x e z è la rappresentazione unaria di y.

- Si definisca M in modo che l'output sia della forma

$$wz0$$

Per esempio se $x = 3$ e $y = 2$, l'input sarà 111011 e M dovrà fermarsi con 111110 sul nastro.

Macchina di Turing per $f(x,y) = x + y$ con $x \neq 0$



Esercizio

Progettare una MdT che calcoli la funzione f(x,y), differenza intera di due interi positivi x e y

$$f(x, y) = x - y \qquad \text{se } x \geq y$$
$$f(x, y) = 0 \qquad \text{altrimenti.}$$

Si supponga che l'input sia $< x > 0 < y >$, dove $< n > = 1^n$, è la rappresentazione unaria dell'intero positivo n.

Un altro esempio

La funzione $f(x) = 2x$ e' calcolabile

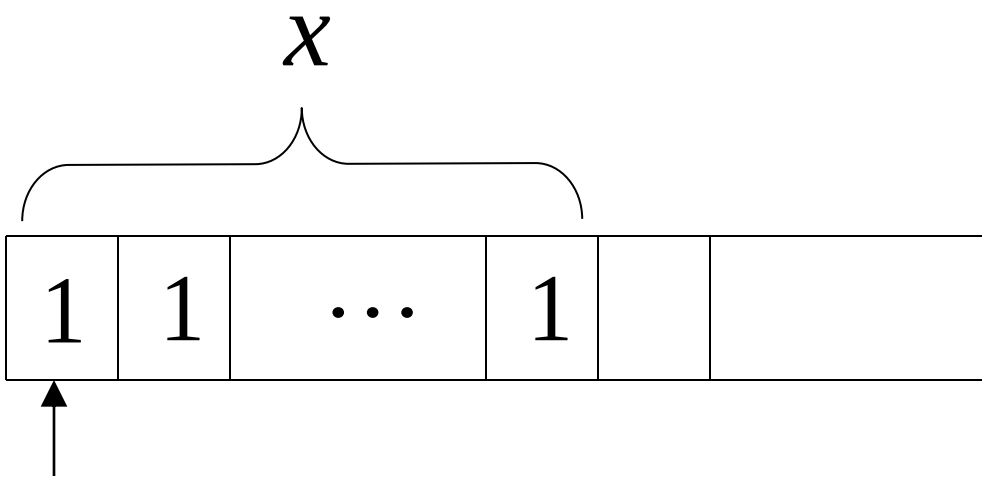
x e' un intero

Macchina di Turing:

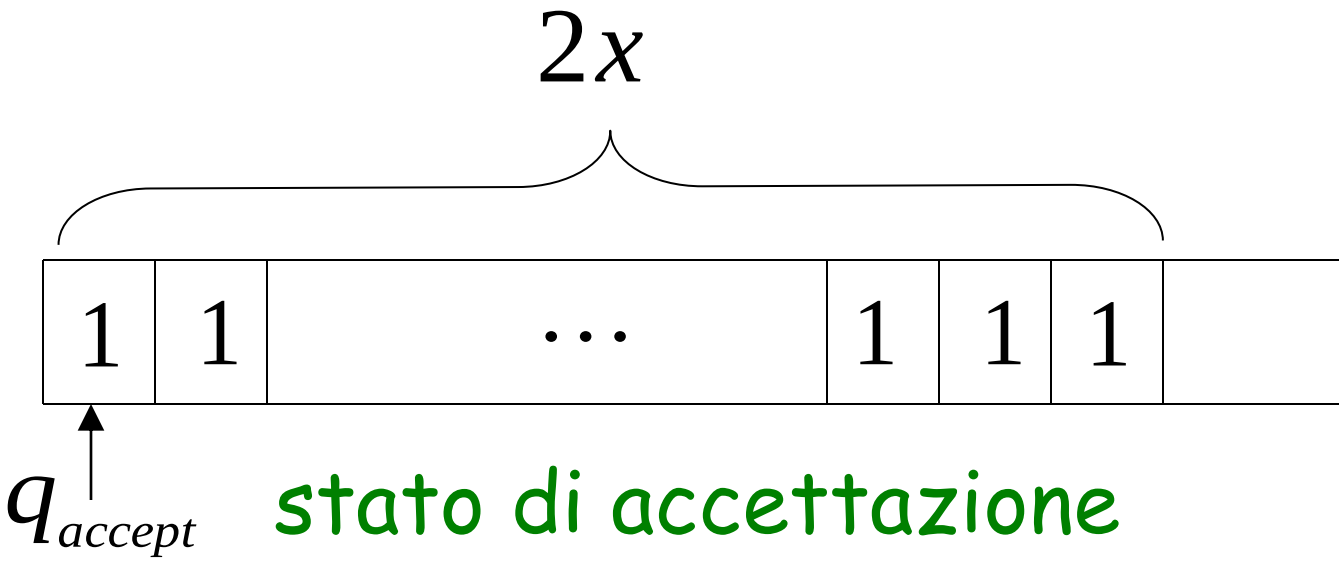
Stringa input: x in notazione unaria

Stringa output: xx in notazione unaria

Inizio


x
1
1
…
1
q_0
stato iniziale

Fine


2x
1
1
…
1
1
1
q_{accept}
stato di accettazione

Un altro esempio

La funzione $f(x) = 2x$ e' calcolabile

x e' un intero

Più in generale Macchina di Turing che copia x:

Stringa input: x Su qualsiasi alfabeto

Stringa output: xx

Un altro esempio

La funzione

e' calcolabile

$$f(x,y) = \begin{cases} 1 & \text{se } x > y \\ 0 & \text{se } x \leq y \end{cases}$$

Quale rappresentazione per lo 0?

1. Progettare una macchina di Turing che sposta l'input a destra di una casella.

2. Progettare una macchina di Turing che calcola il successore in binario.

1. Progettare una macchina di Turing che sposta l'input a destra di una casella.

2. Progettare una macchina di Turing che calcola il successore in binario.

Questi esercizi ci mostrano cosa può fare una MdT:
copiare, spostare, calcolare successore, somma, differenza (in unario o binario), …
contare, ovvero riconoscere occorrenze di ugual numero, lunghezza pari ad una potenza di 2, …..

Livelli di descrizione di MdT

La MdT è nata come modello preciso di «algoritmo», «procedura effettiva di calcolo».

Tre livelli di descrizione:

1. Descrizione ad alto livello
2. Descrizione implementativa (come muove la testina, memorizza i dati, …)
3. Descrizione formale, precisa (come settupla/diagramma/ tabella)

MdT – example

MdT M recognizing $B = \{\mathrm{a}^k\mathrm{b}^k\mathrm{c}^k \mid k \geq 0\}$

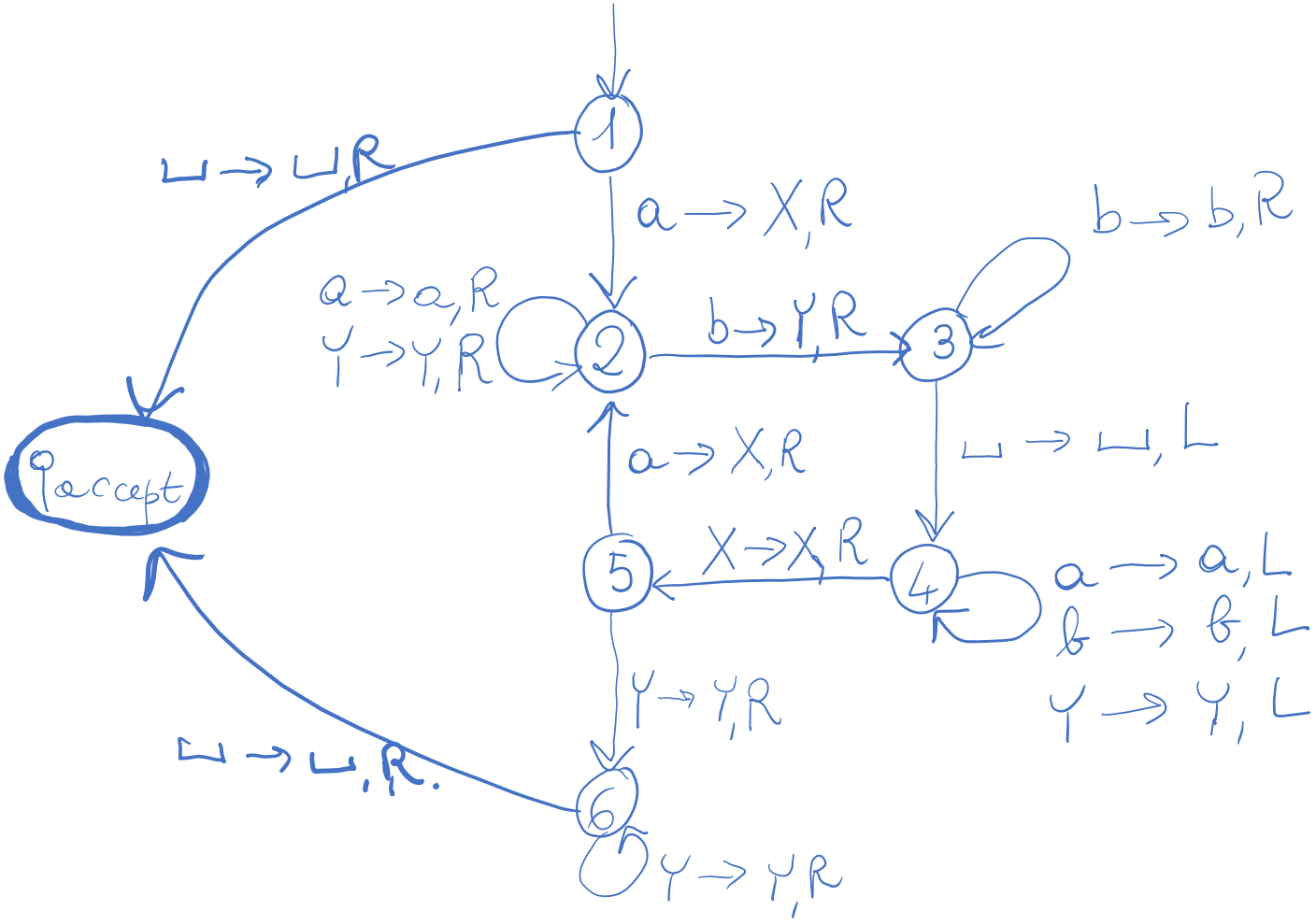
High-level description.

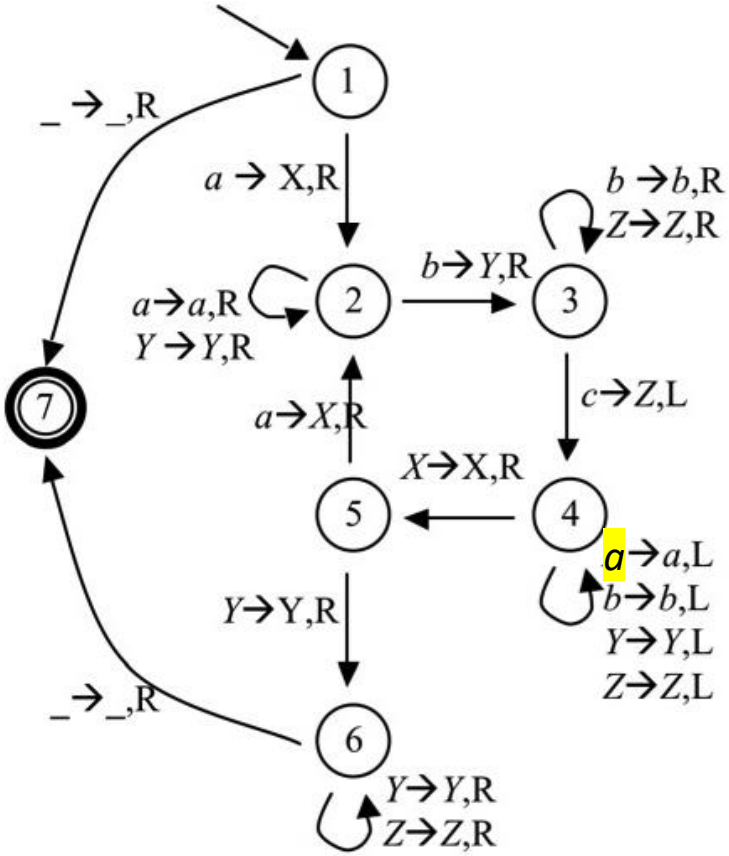
$M =$ "On input w

1. Check if $w \in \mathrm{a}^*\mathrm{b}^*\mathrm{c}^*$, *reject* if not.
2. Count the number of a's, b's, and c's in w.
3. *Accept* if all counts are equal; *reject* if not."

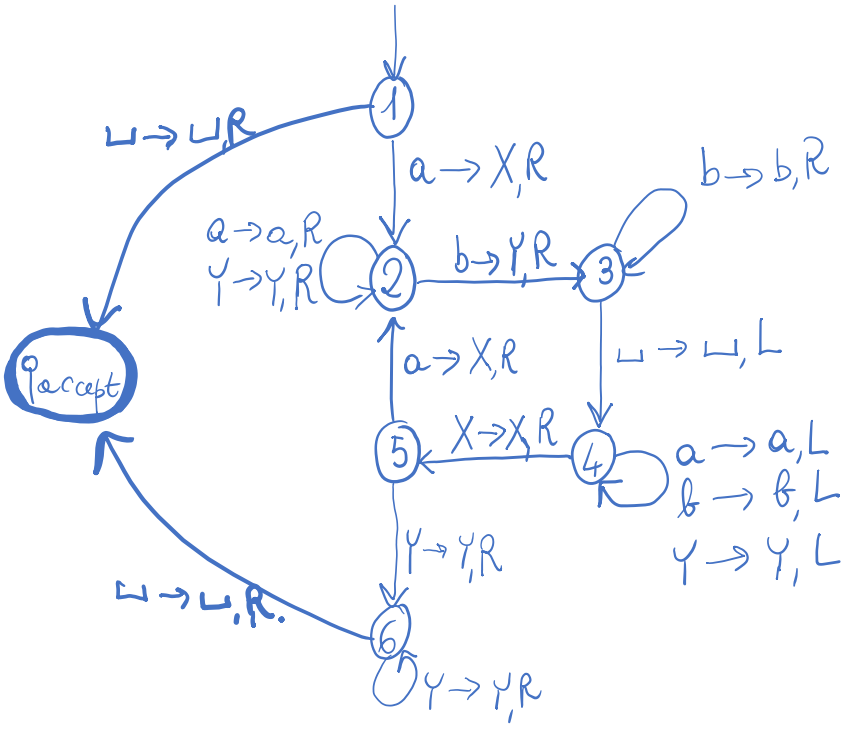
You do not manage tapes, states, etc…

Oppure cominciamo da { $a^n b^n$ }



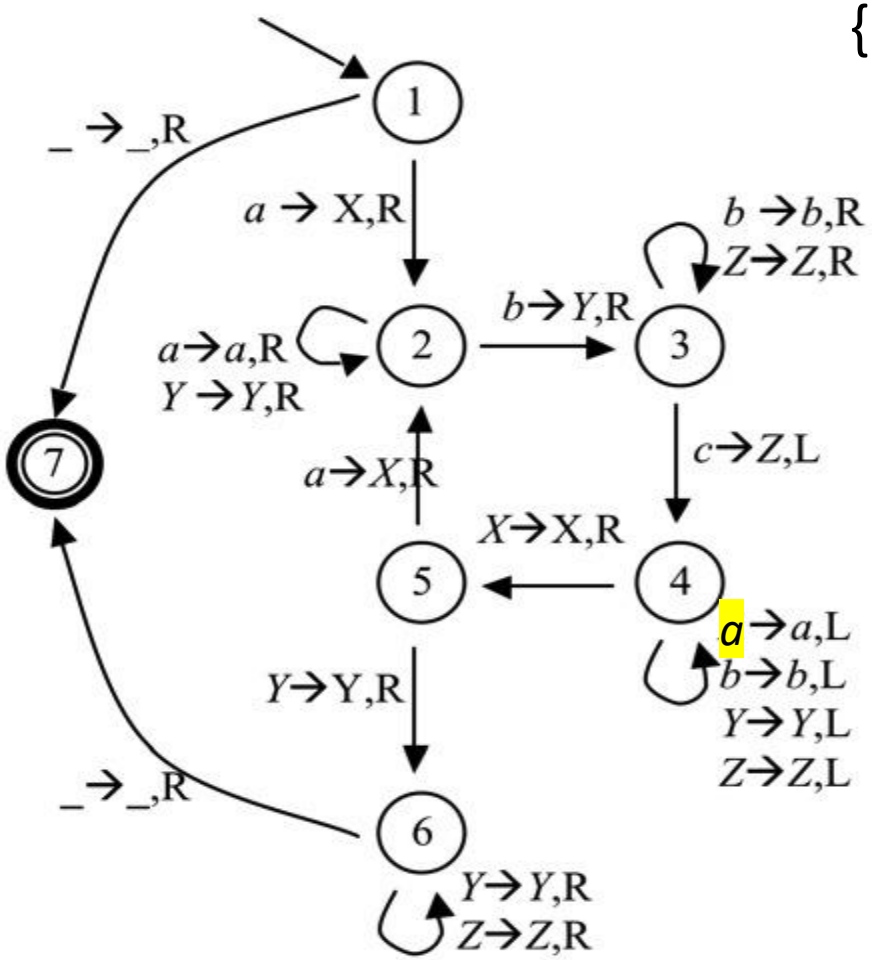

1
_ → _,R
a → X,R
b → b,R
Z→Z,R
b→Y,R
2
3
a→a,R
Y →Y,R
7
a→X,R
c→Z,L
X→X,R
5
4
a→a,L
b→b,L
Y→Y,L
Z→Z,L
Y→Y,R
_→_,R
6
Y→Y,R
Z→Z,R

$\{ a^n b^n c^n \}$


1
␣→␣,R
a→X,R
b→b,R
a→a,R
Y→Y,R
2
b→Y,R
3
accept
a→X,R
␣→␣,L
5
X→X,R
4
a→a,L
b→b,L
Y→Y,L
Y→Y,R
␣→␣,R
6
Y→Y,R

$\{ a^n b^n \}$

$\{ a^k b^k c^k \}$


1
_ → _,R
a → X,R
b → b,R
Z → Z,R
b → Y,R
a → a,R
Y → Y,R
2
3
7
a → X,R
c → Z,L
X → X,R
5
4
a → a,L
b → b,L
Y → Y,L
Z → Z,L
Y → Y,R
_ → _,R
6
Y → Y,R
Z → Z,R

Implementazione della memoria

In alcune occasioni avremo bisogno di memorizzare l'informazione letta sul nastro
Possiamo usare gli stati per memorizzare informazione

Es. Supponiamo MdT M deve leggere i primi 2 bit del nastro e scriverli alla fine dell'input (al posto dei 2 $\perp$ piú a sinistra)

Idea:

- Costruiamo una MdT
  M_{00} che legge i primi 2 bit, cerca la fine dell'input e scrive 00 alla fine dell'input
- Replichiamo per tutte le possibili coppie: M_{01}, M_{10}, M_{11}
- M: dopo aver letto i primi 2 bit input, si sposta sulla replica corrispondente ai 2 bit letti (e quindi li scrive alla fine dell'input).

Implementazione della memoria

Nota: questa è una tecnica utilizzabile in generale

Se vogliamo che M memorizzi una sequenza di k simboli possiamo

- avere una replica per ogni possibile sequenza di k simboli
- M si sposta sulla replica che corrisponde alla sequenza mentre la legge

Necessari circa $|\Sigma|^k$ stati aggiuntivi

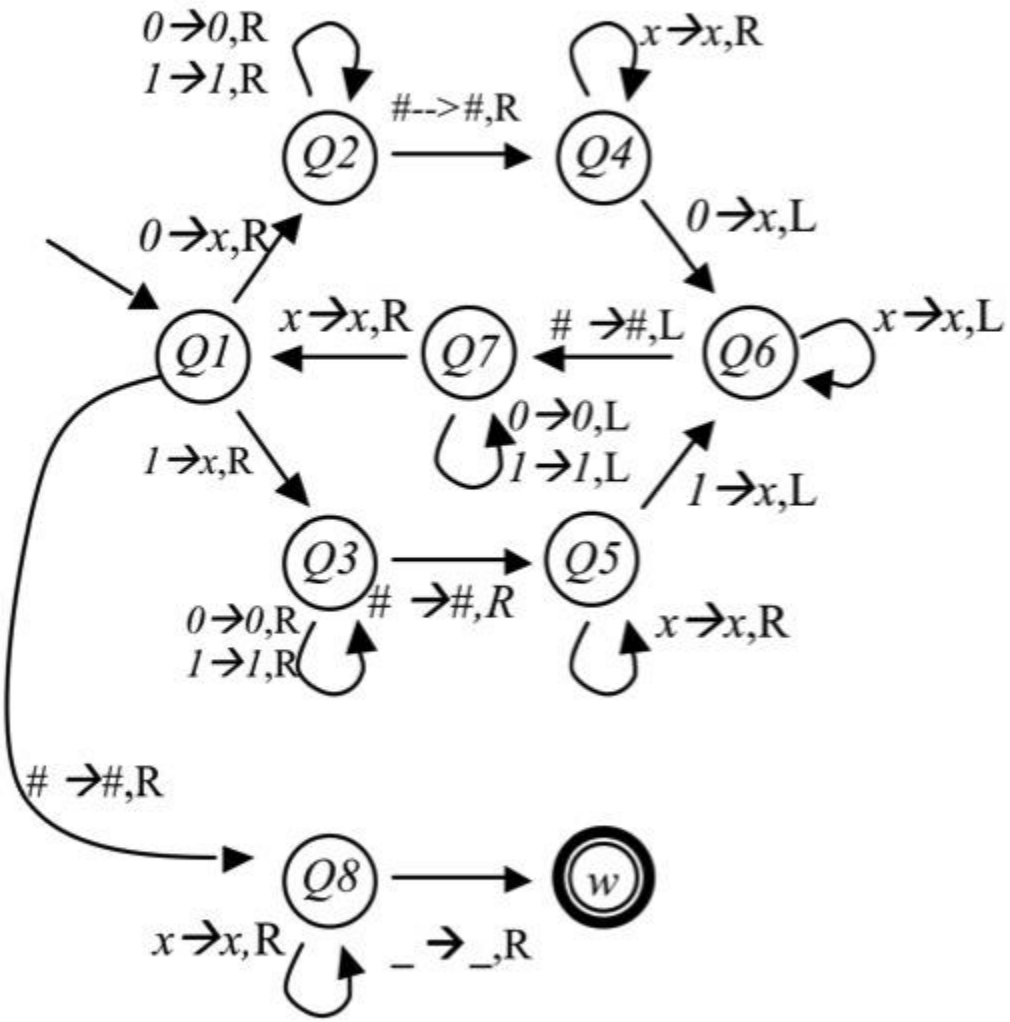
MdT per w#w

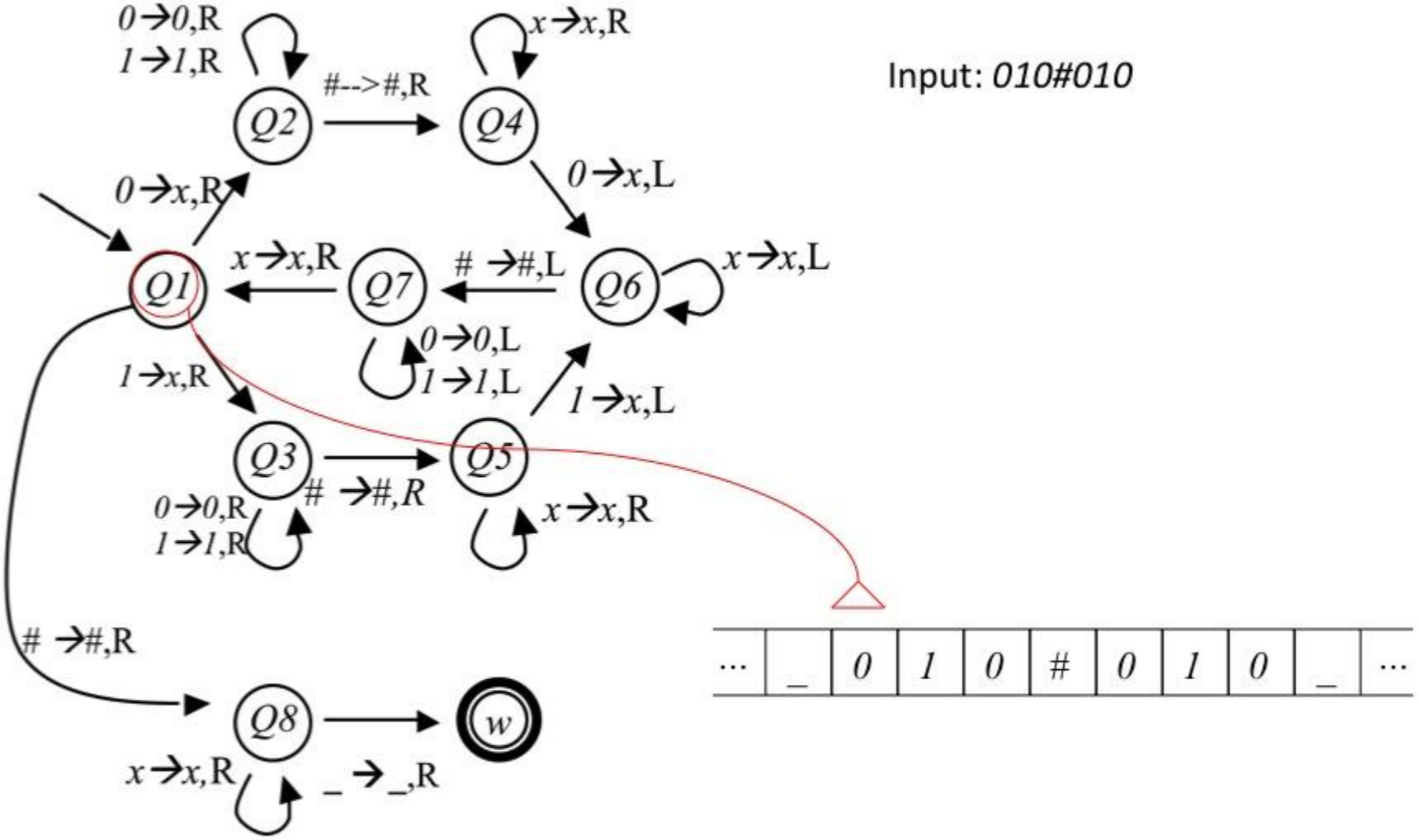
Esercizio

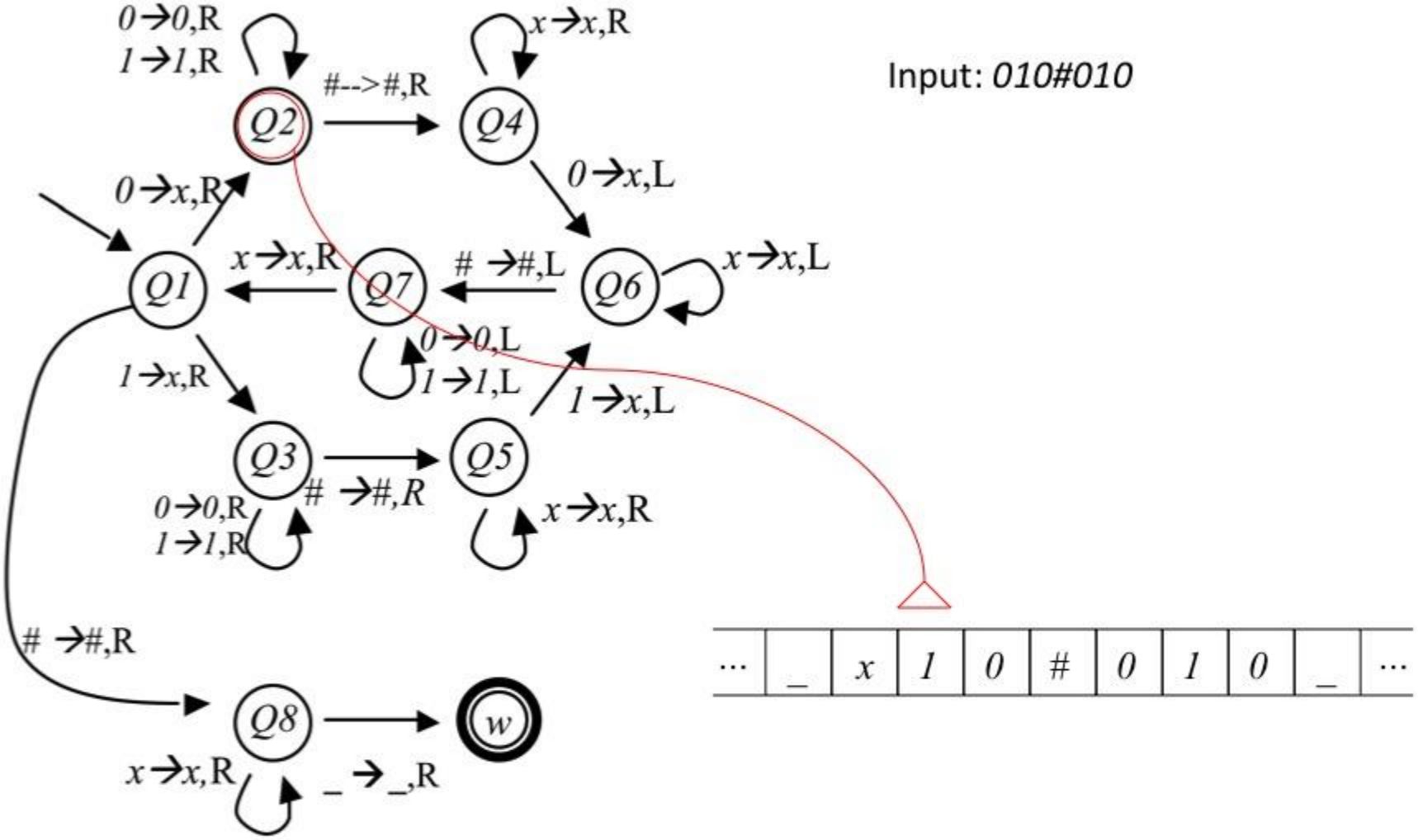
Progettare una MdT che decida il linguaggio

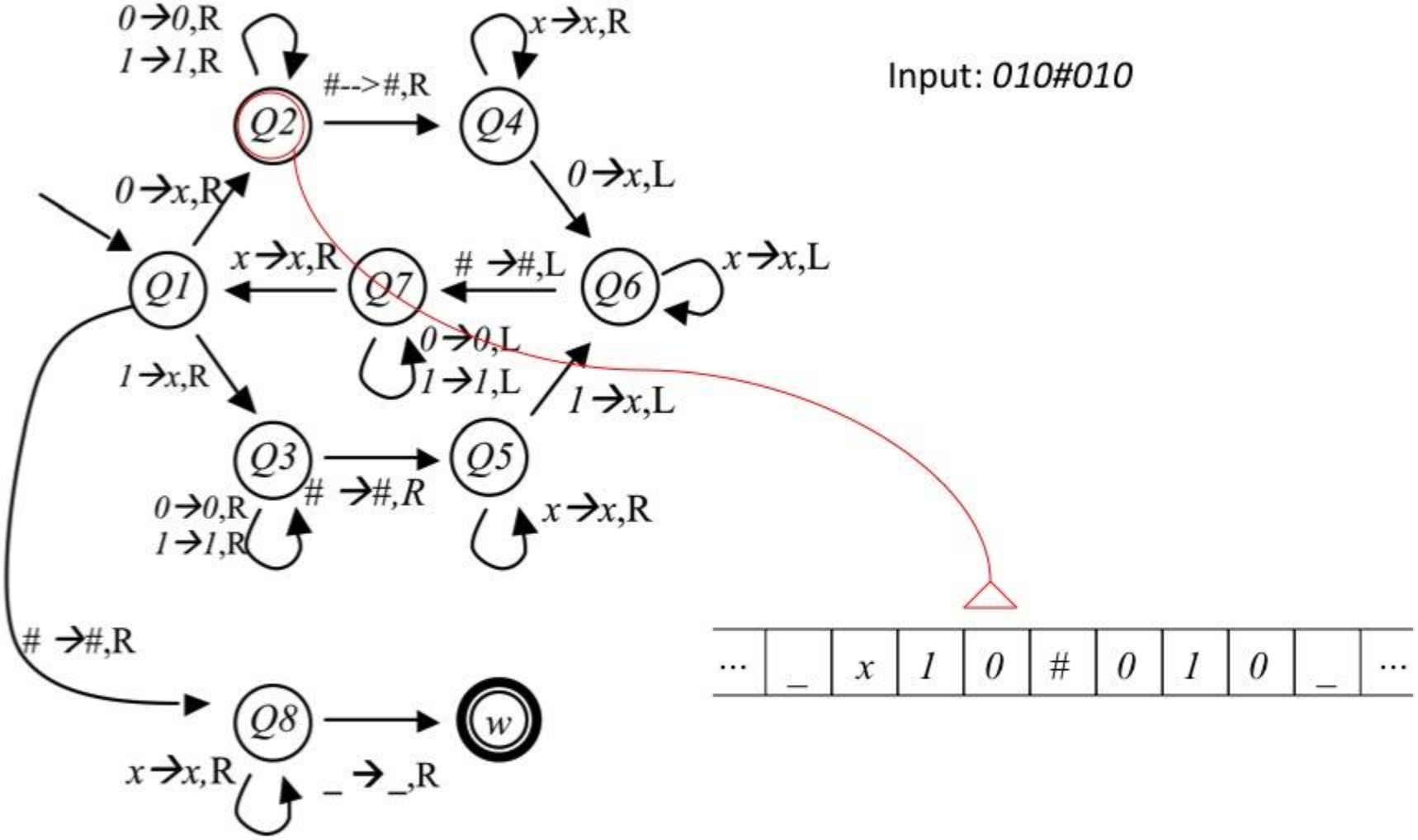
L = { w#w| w è una stringa sull'alfabeto { 0, 1 } }

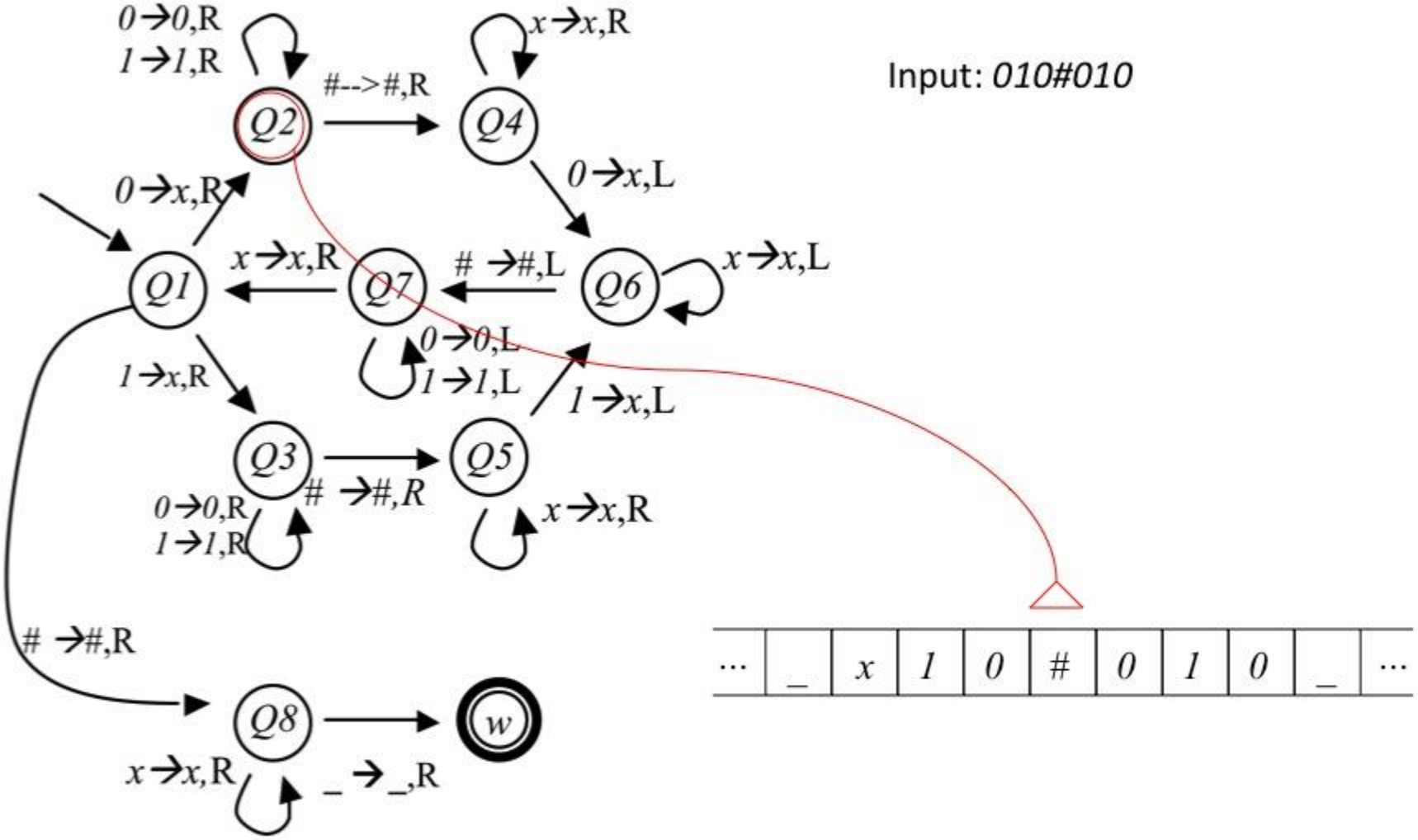
MdT per w#w

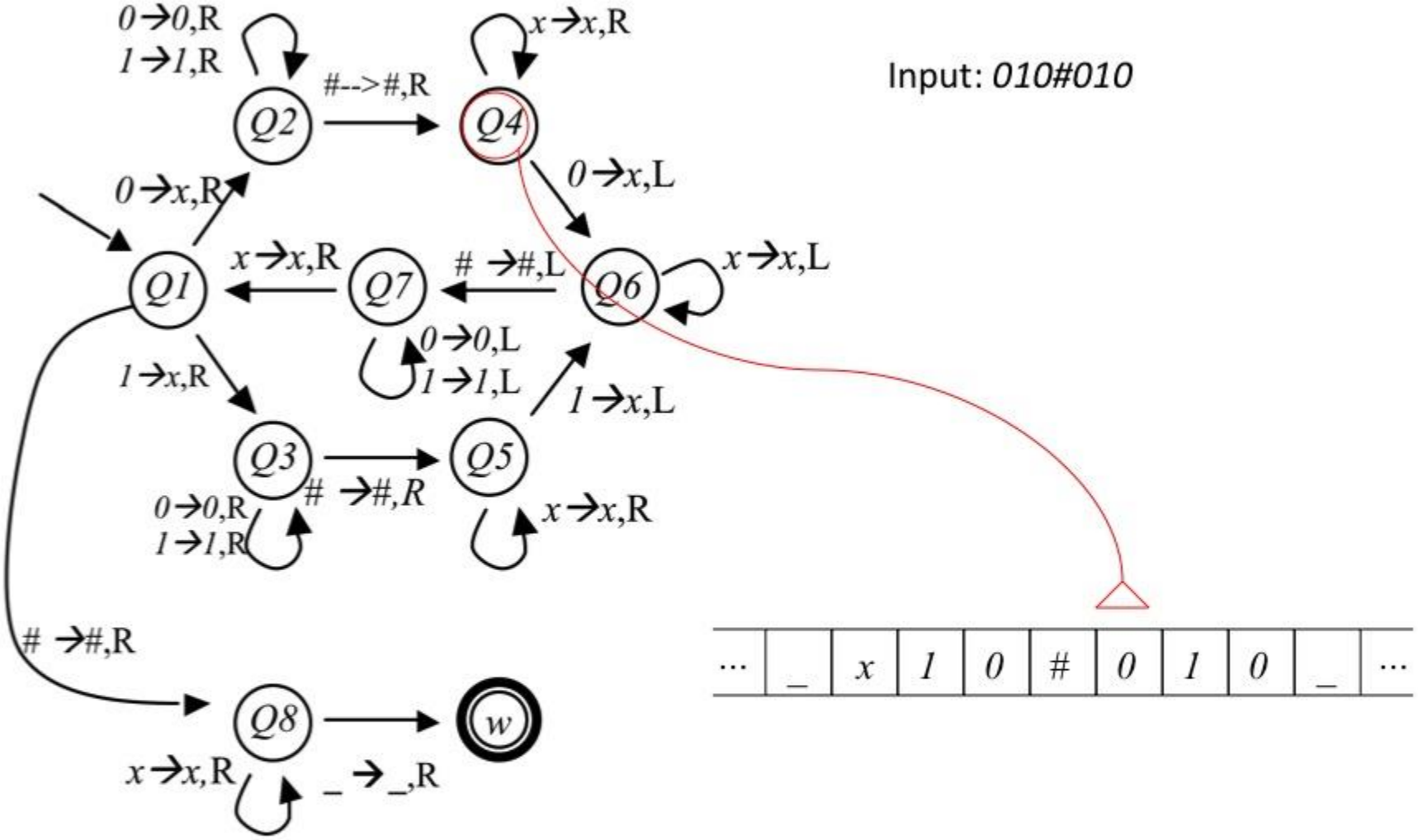

0→0,R
1→1,R
#-->#,R
x→x,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R

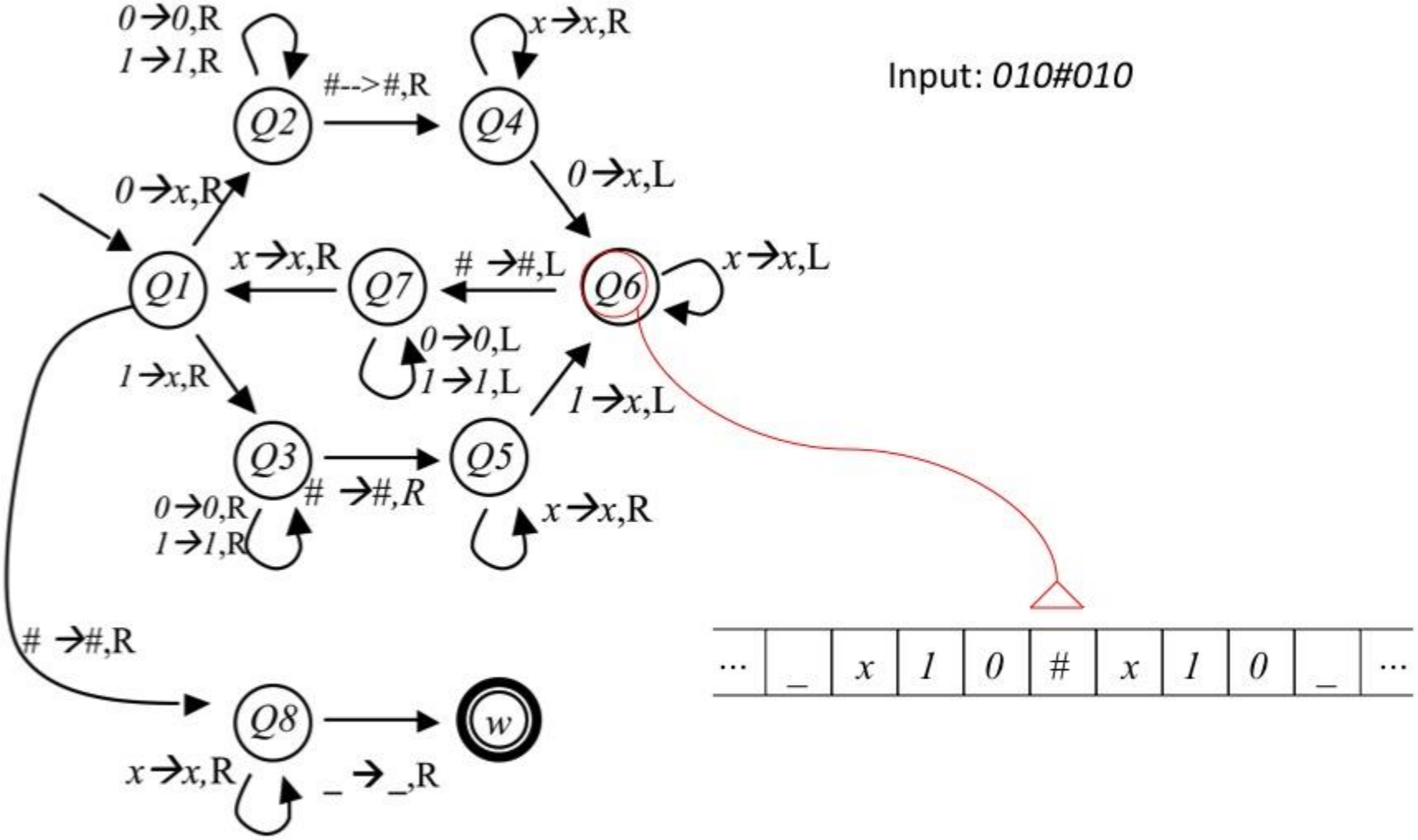

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ 0 1 0 # 0 1 0 _ …

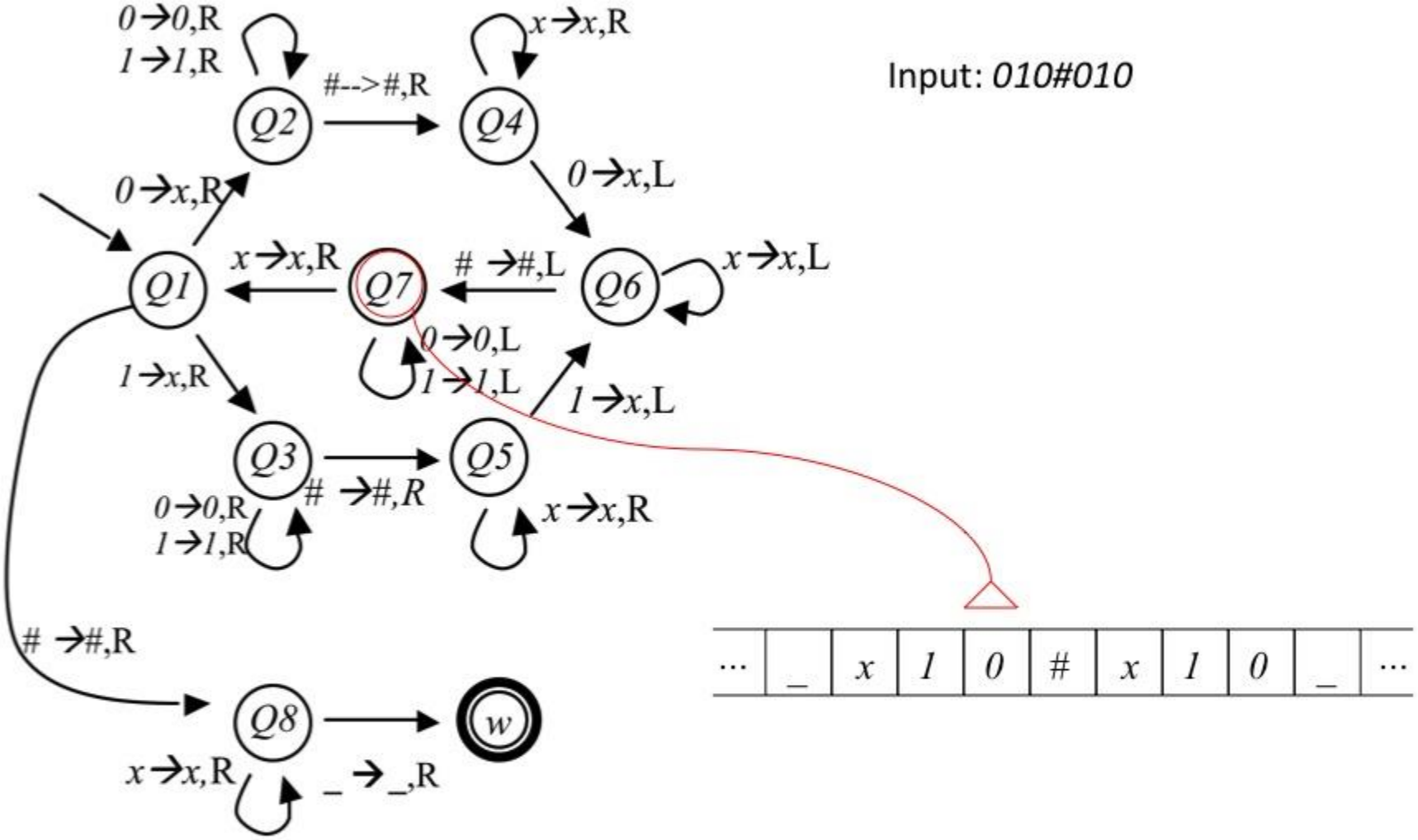

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x 1 0 # 0 1 0 _ …

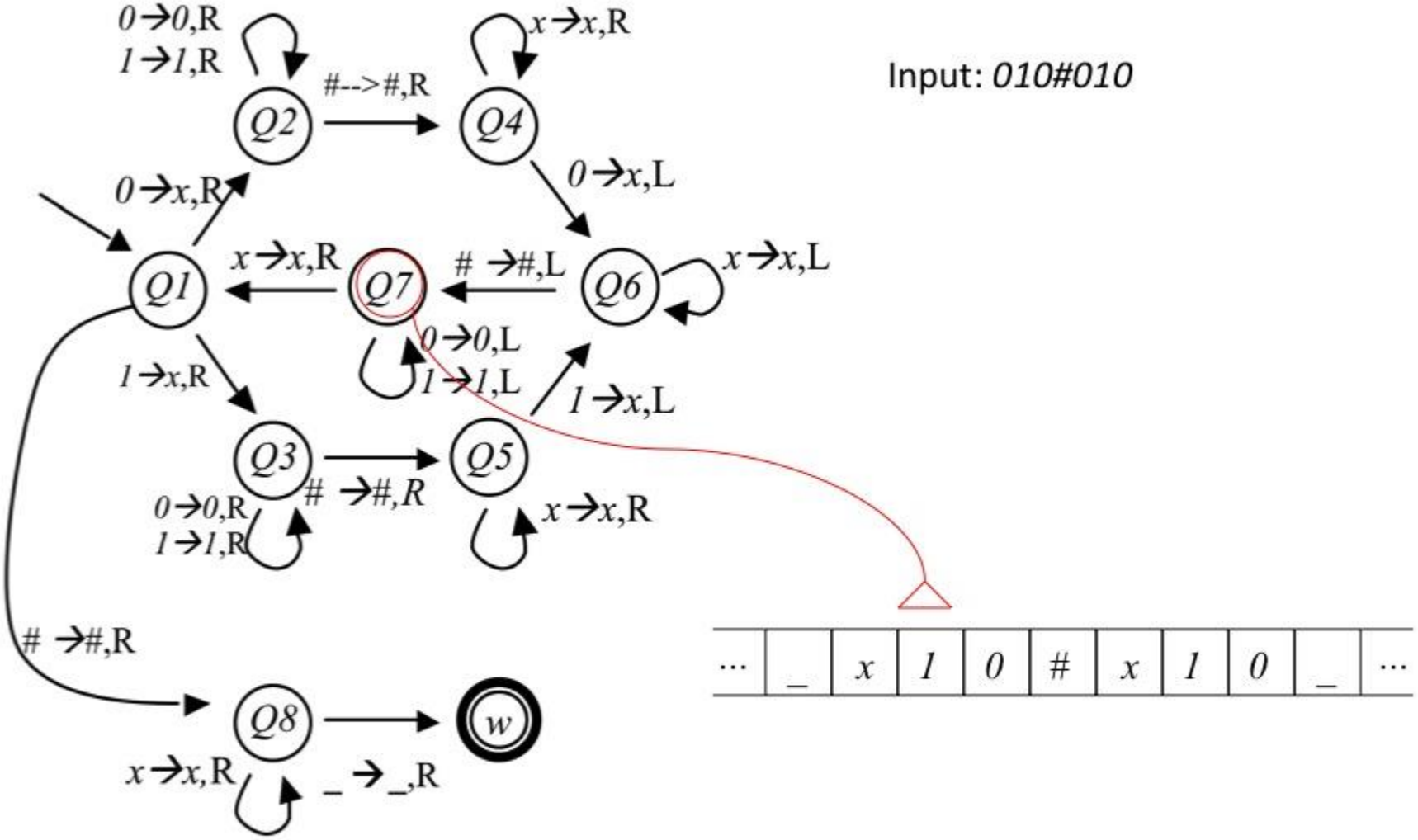

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x 1 0 # 0 1 0 _ …

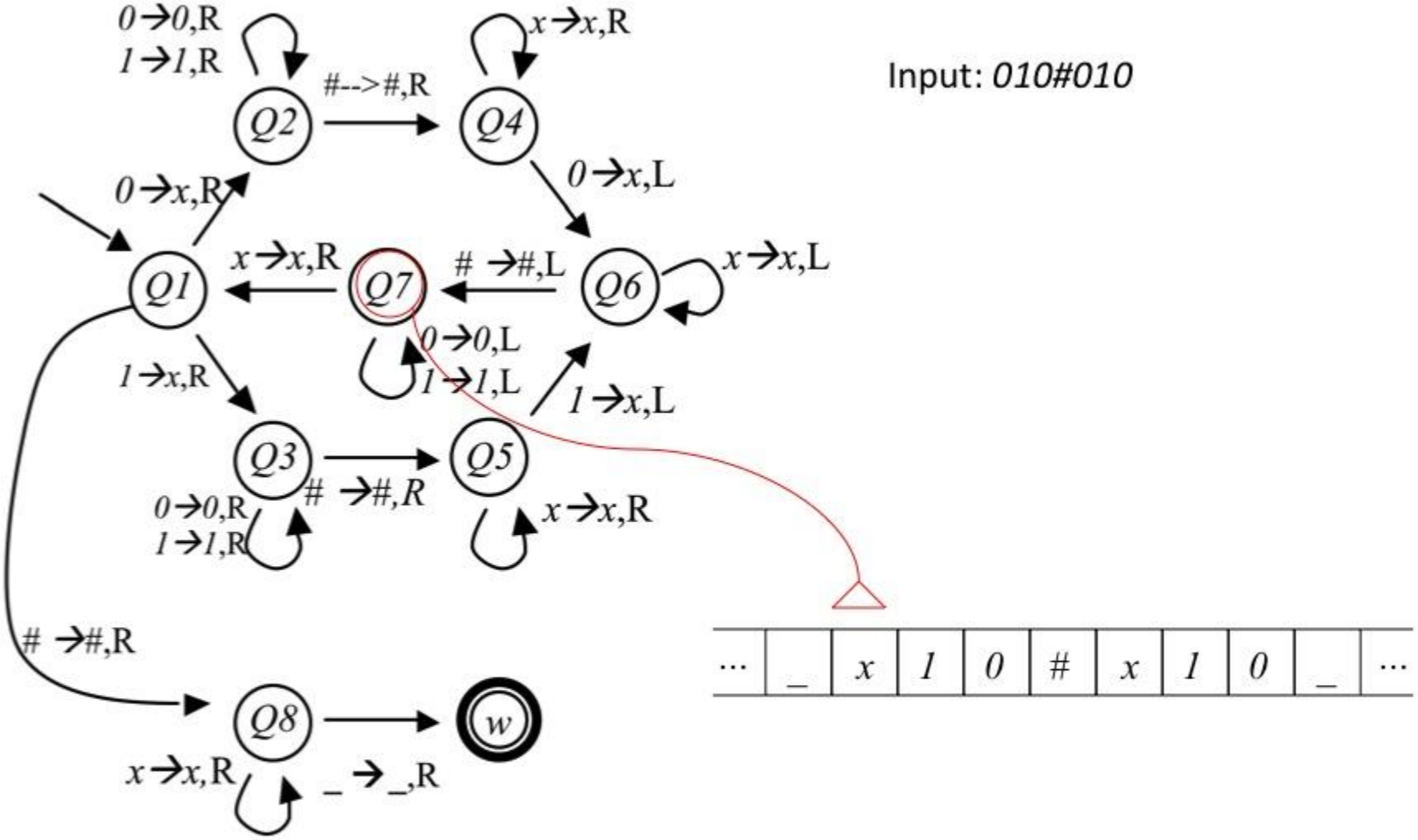

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x 1 0 # 0 1 0 _ …

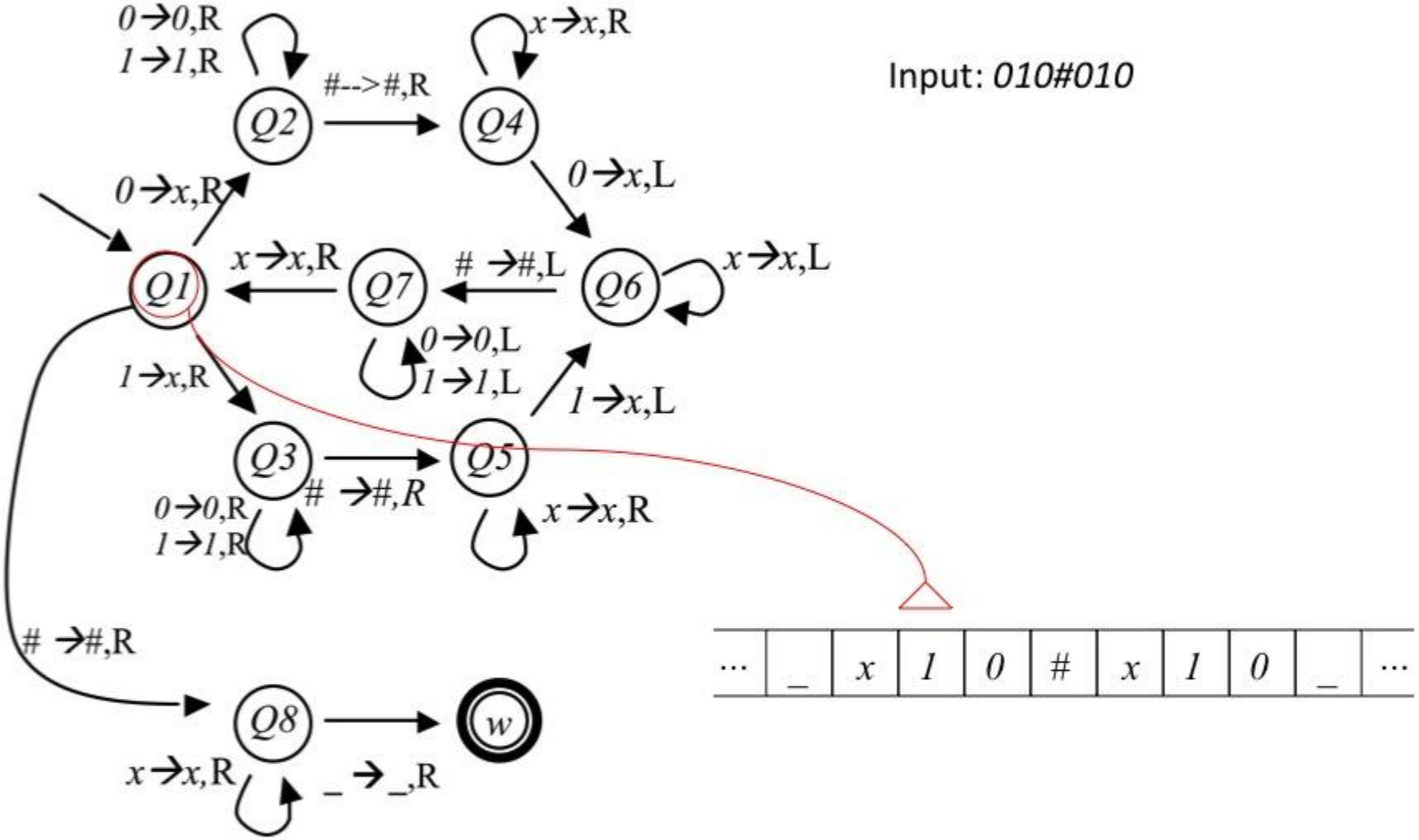

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x 1 0 # 0 1 0 _ …

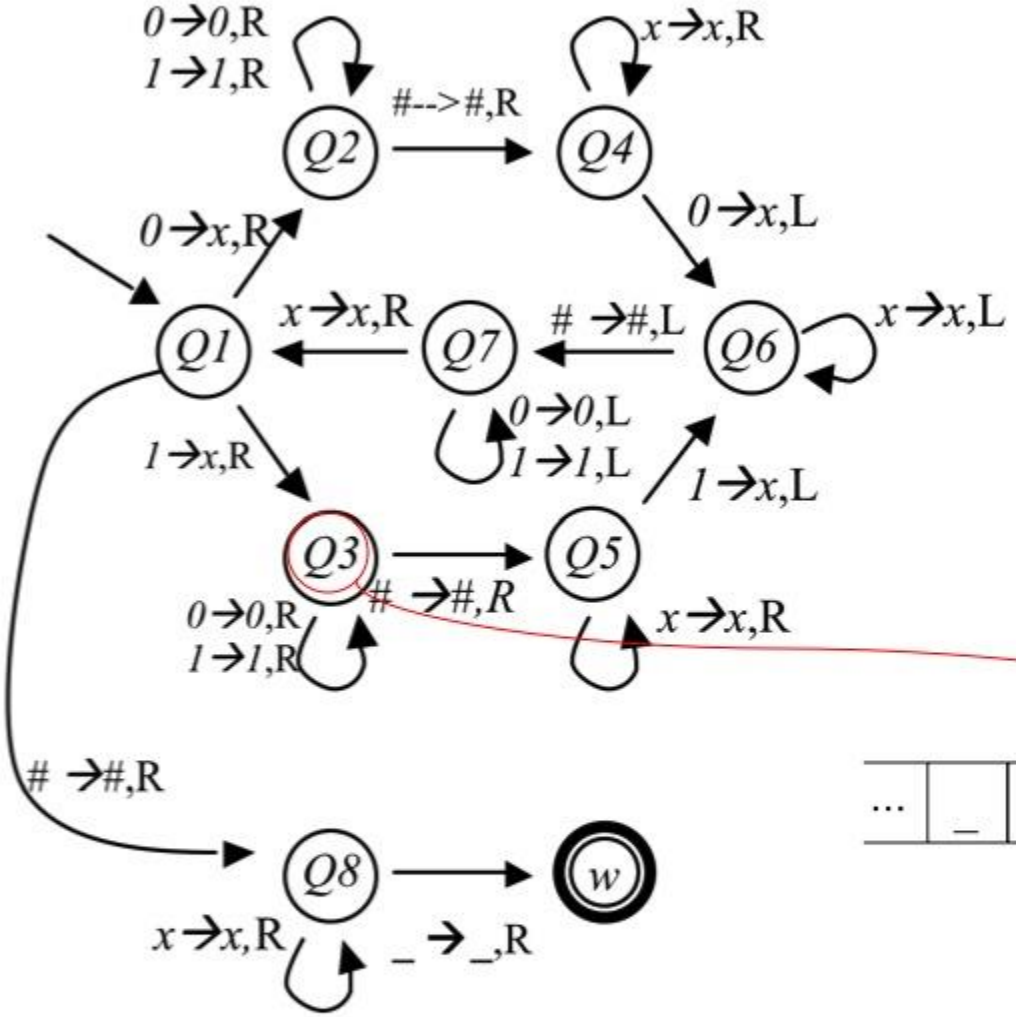

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x 1 0 # x 1 0 _ …


0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x 1 0 # x 1 0 _ …


0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x 1 0 # x 1 0 _ …

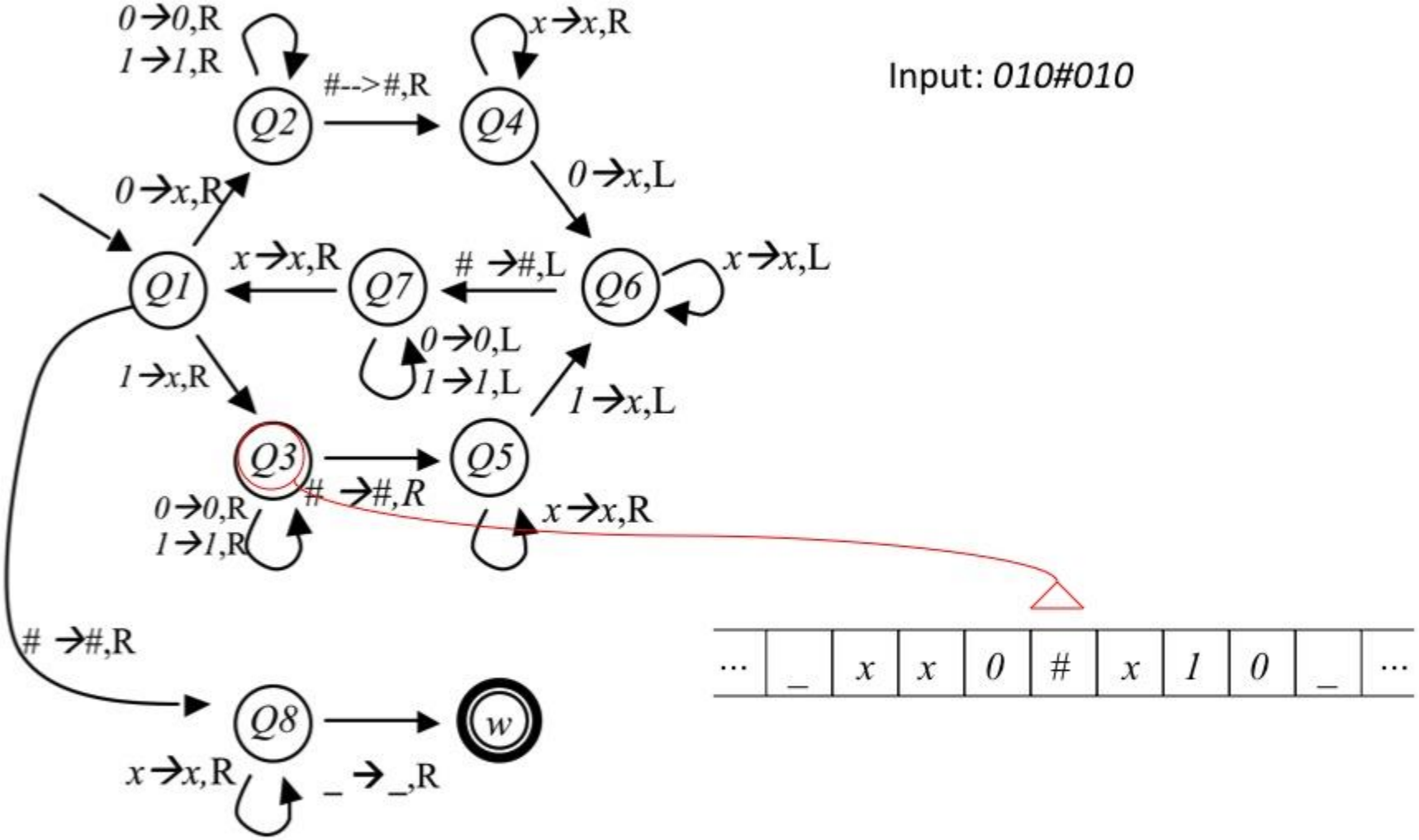

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x 1 0 # x 1 0 _ …

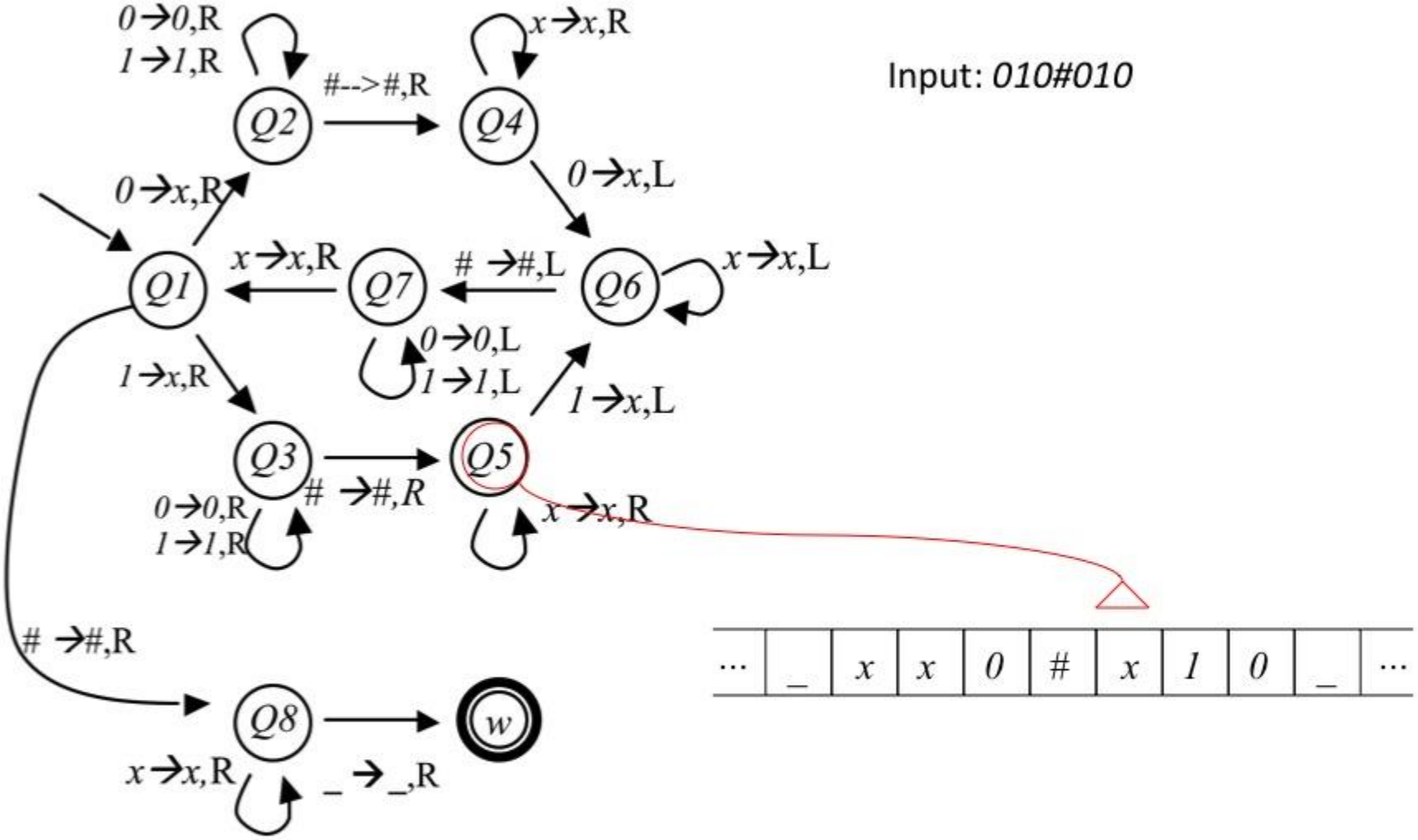

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x 1 0 # x 1 0 _ …

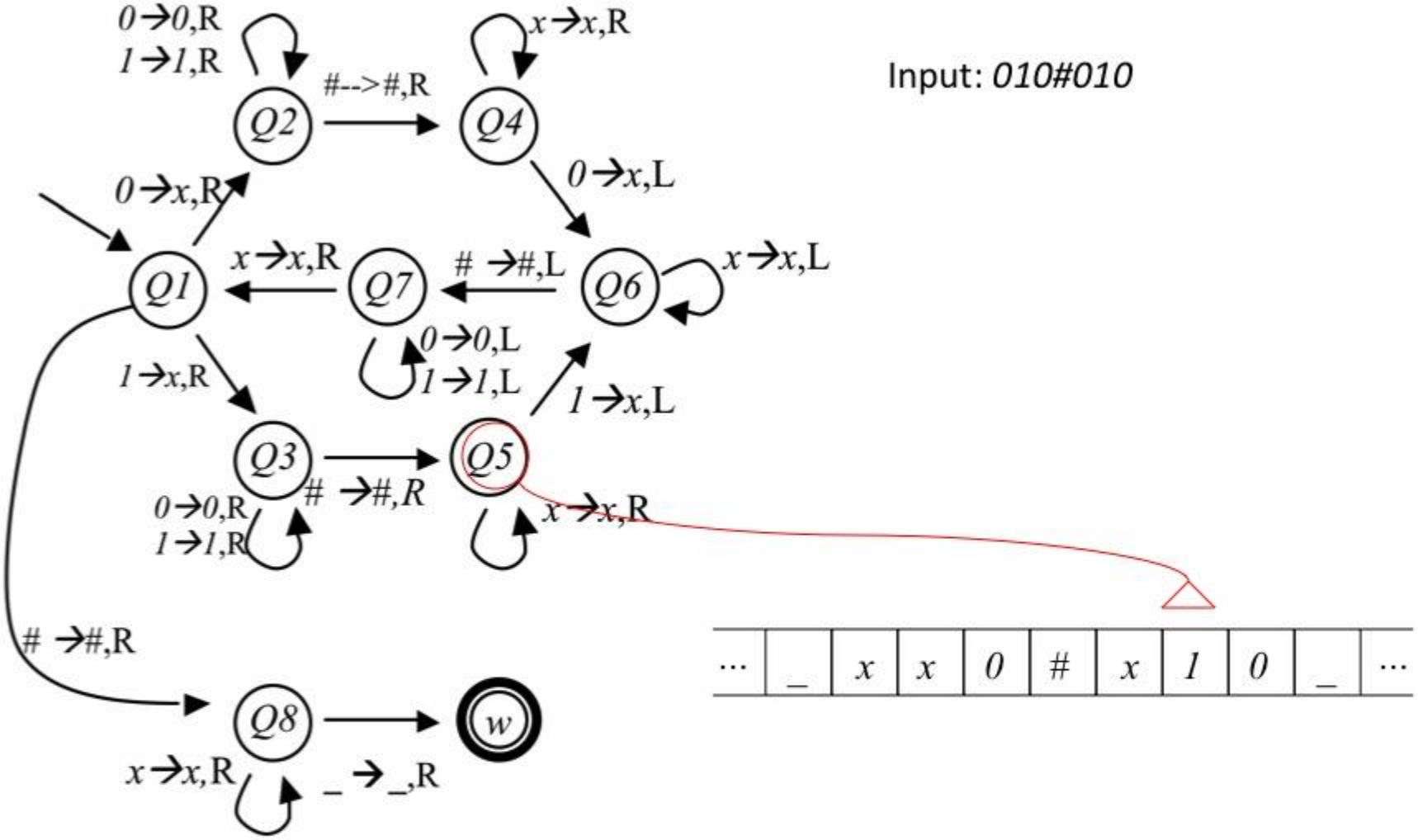


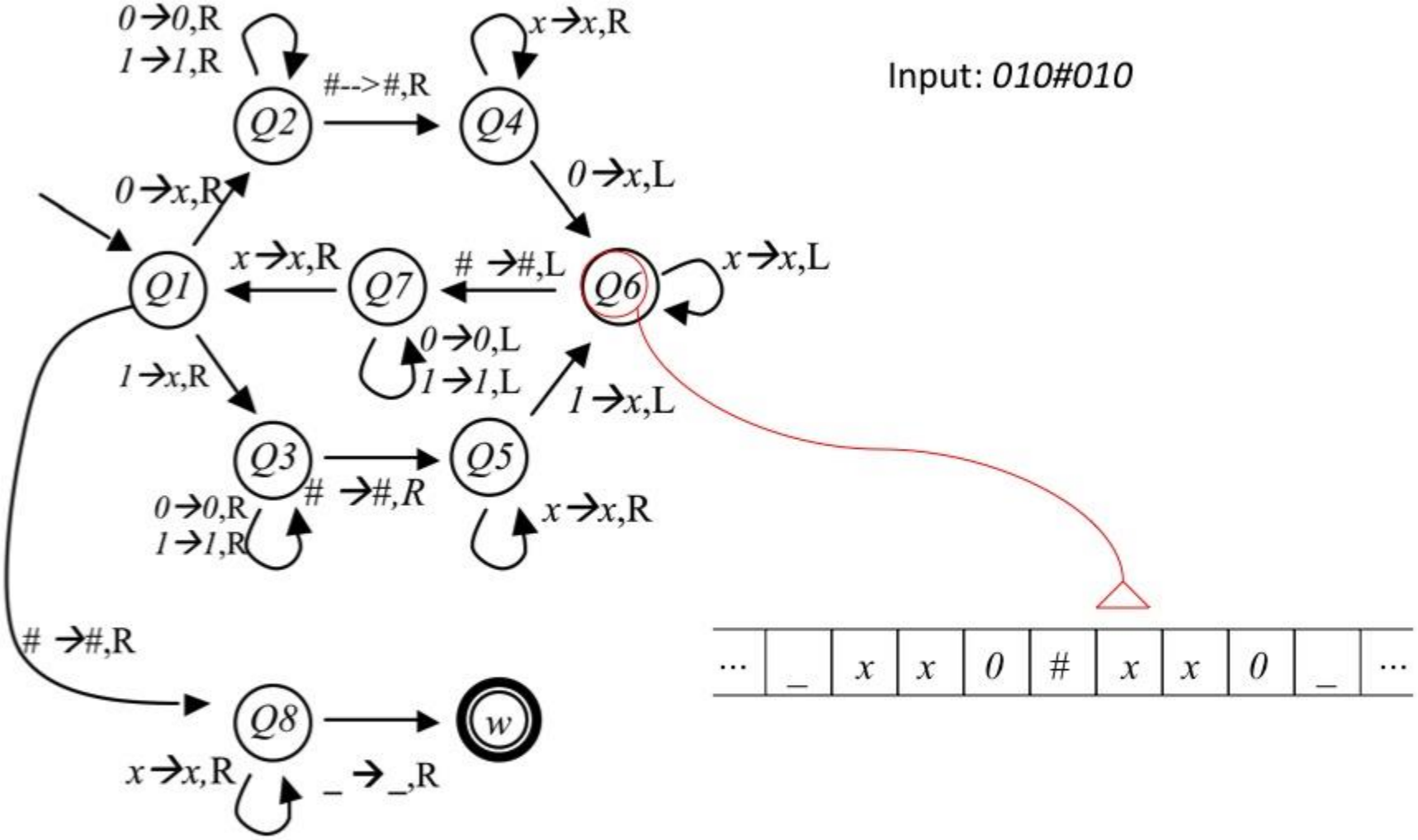
Input: *010#010*

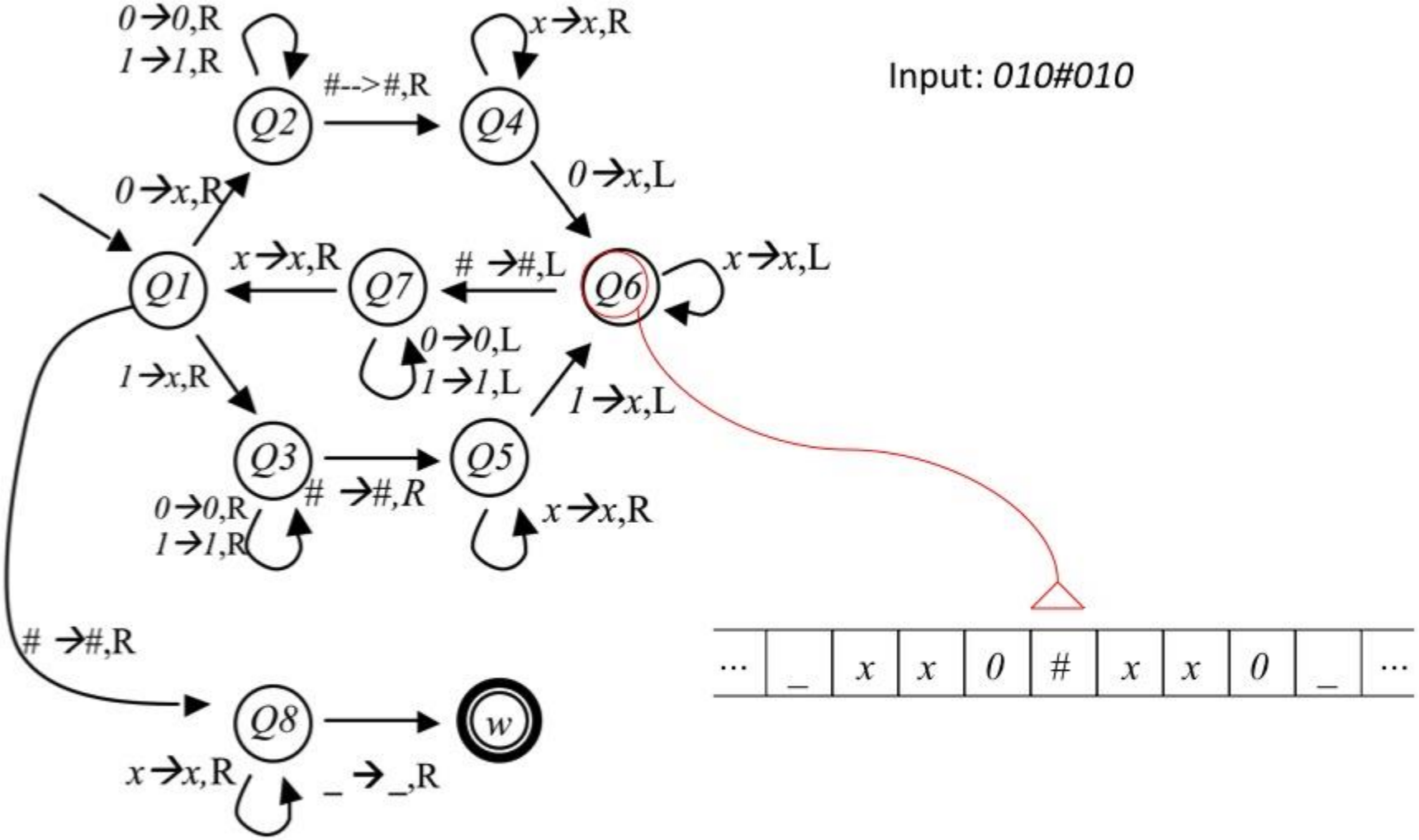
…	_	x	x	0	#	x	1	0	_	…

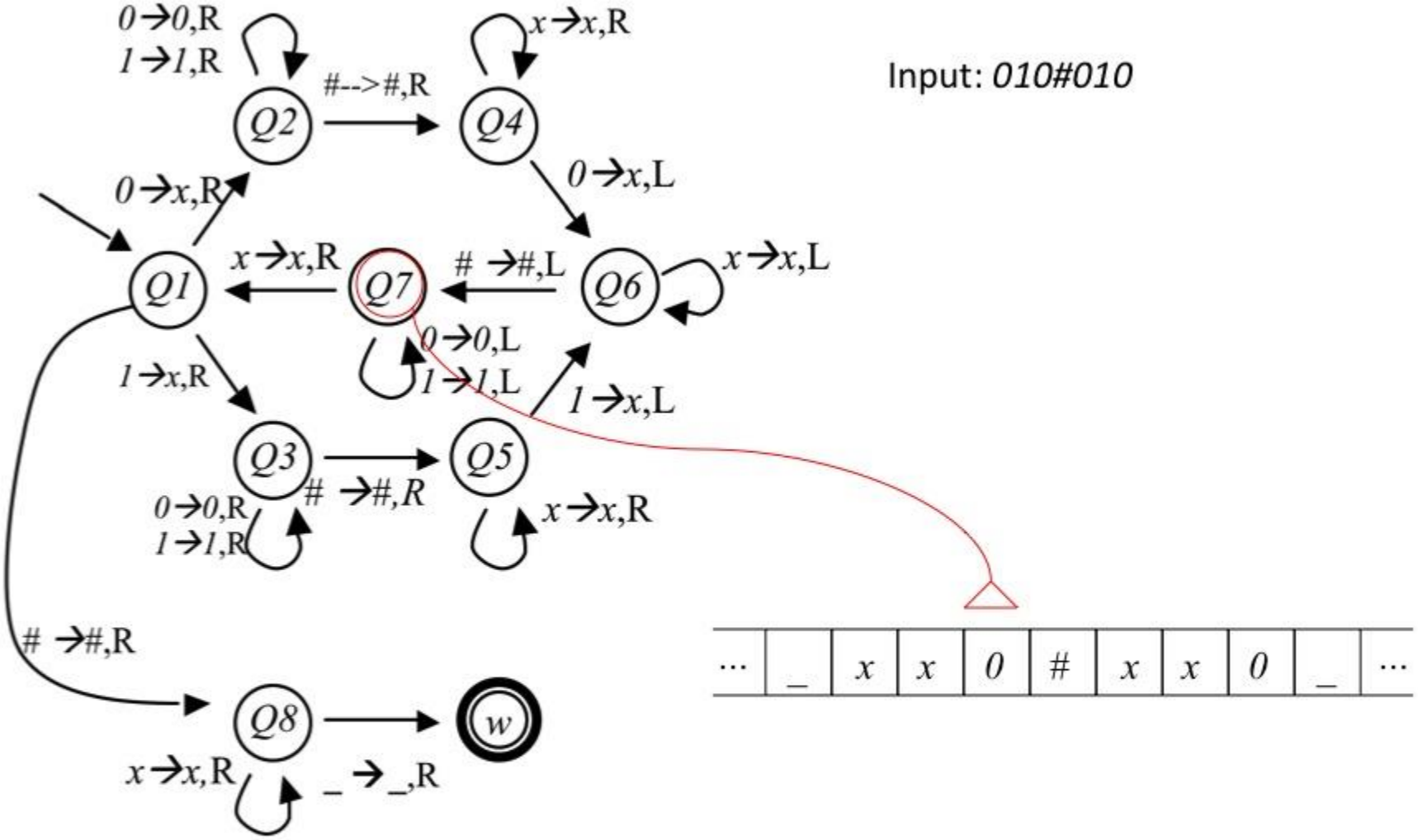

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x 0 # x 1 0 _ …

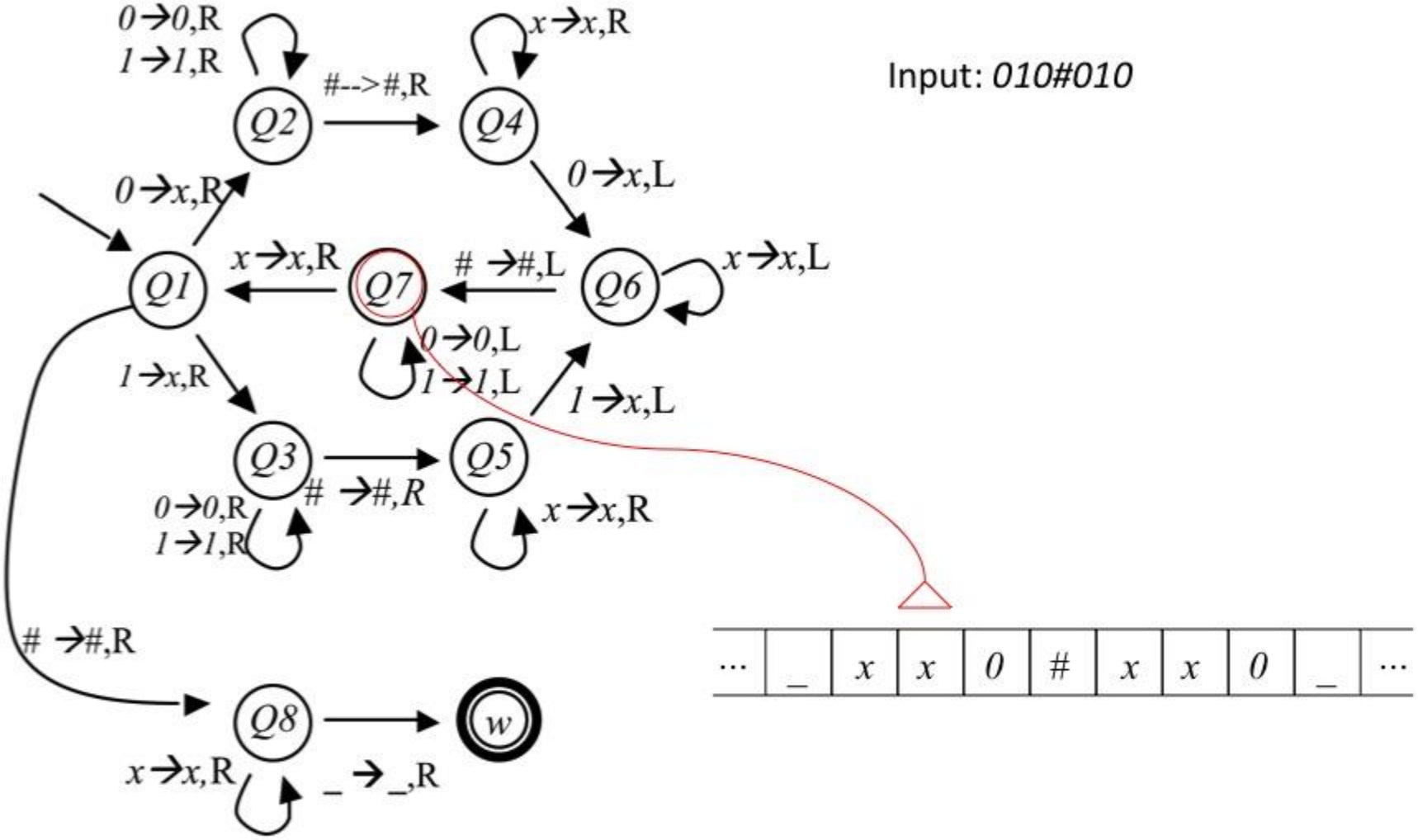

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x 0 # x 1 0 _ …

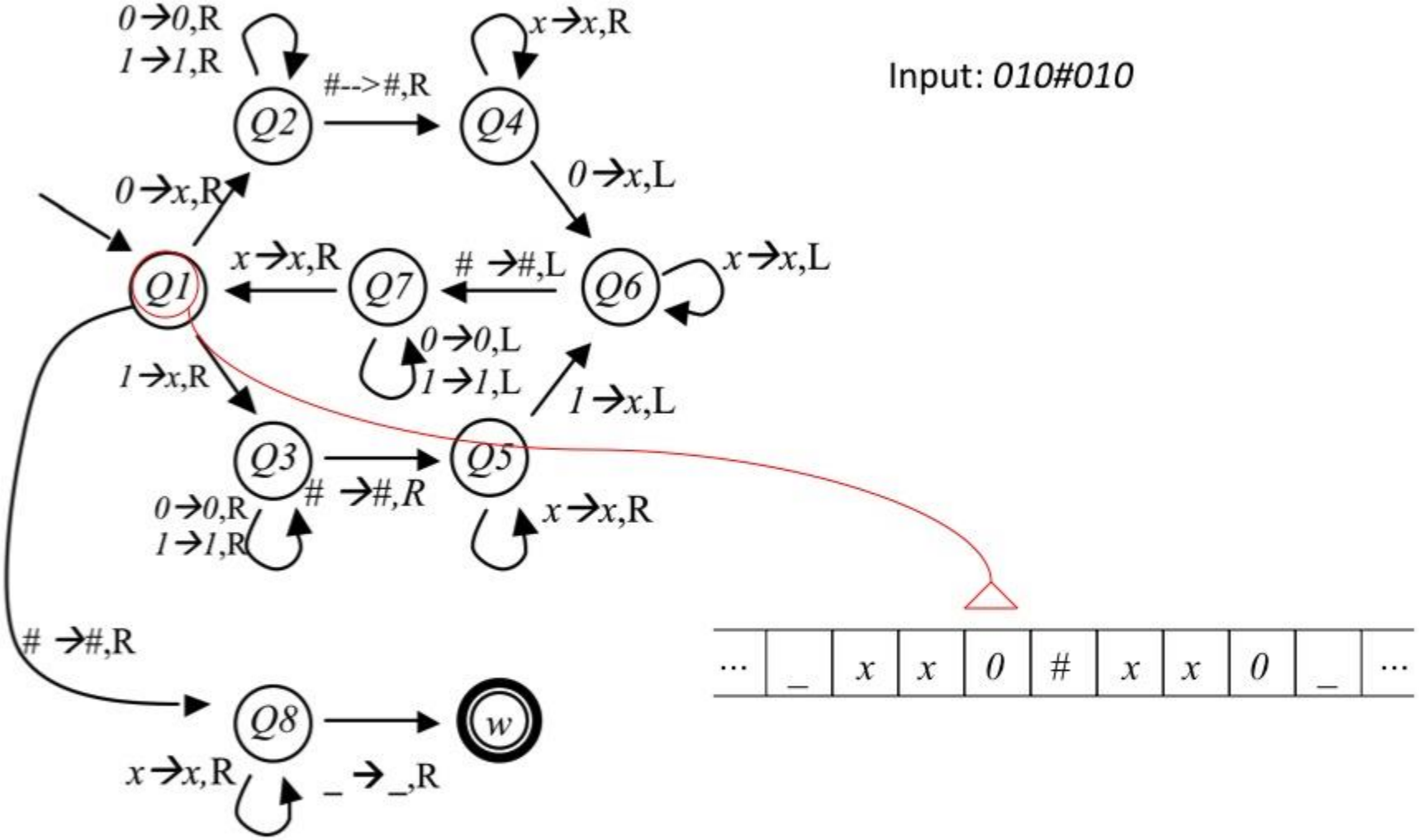

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x 0 # x 1 0 _ …

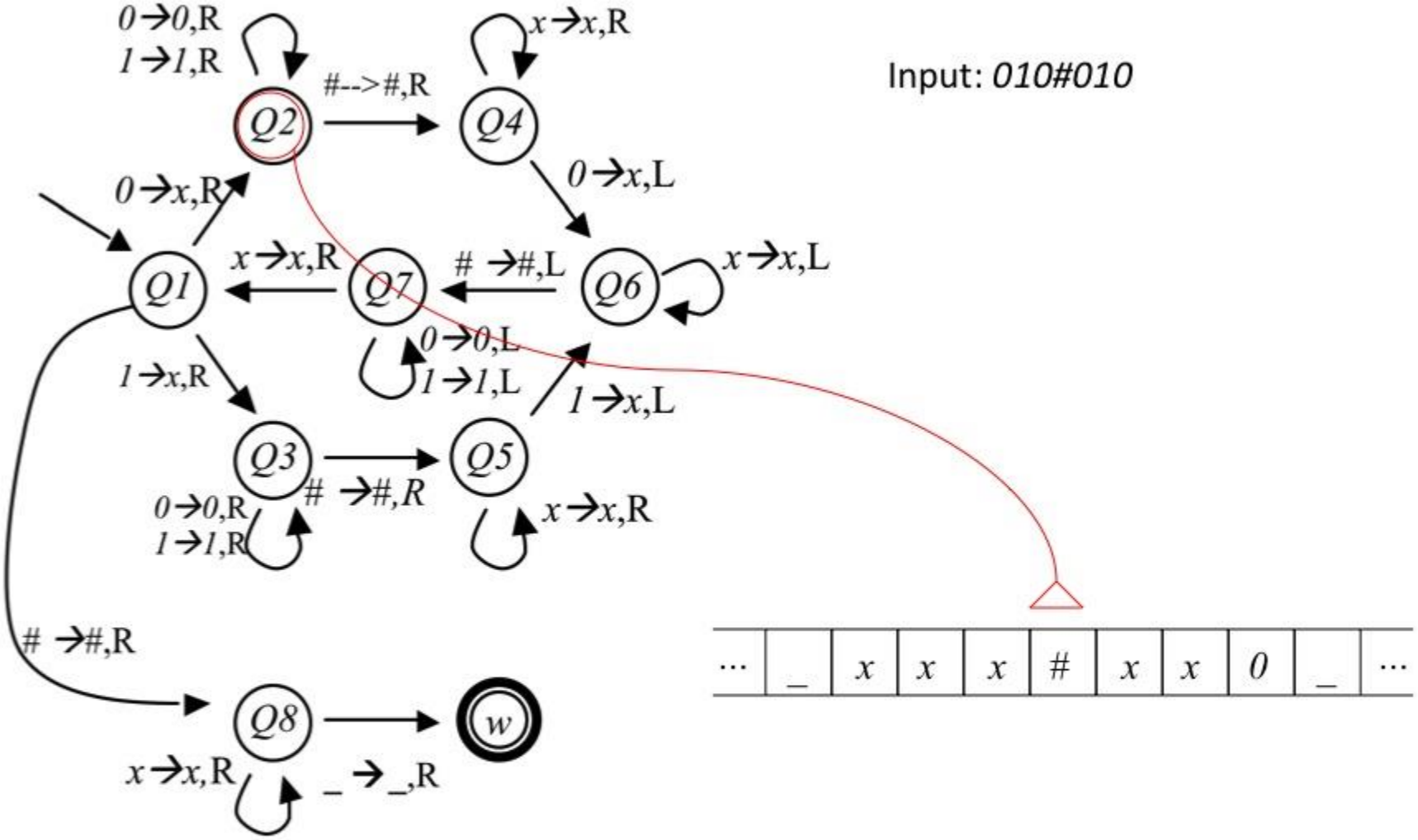

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x 0 # x x 0 _ …

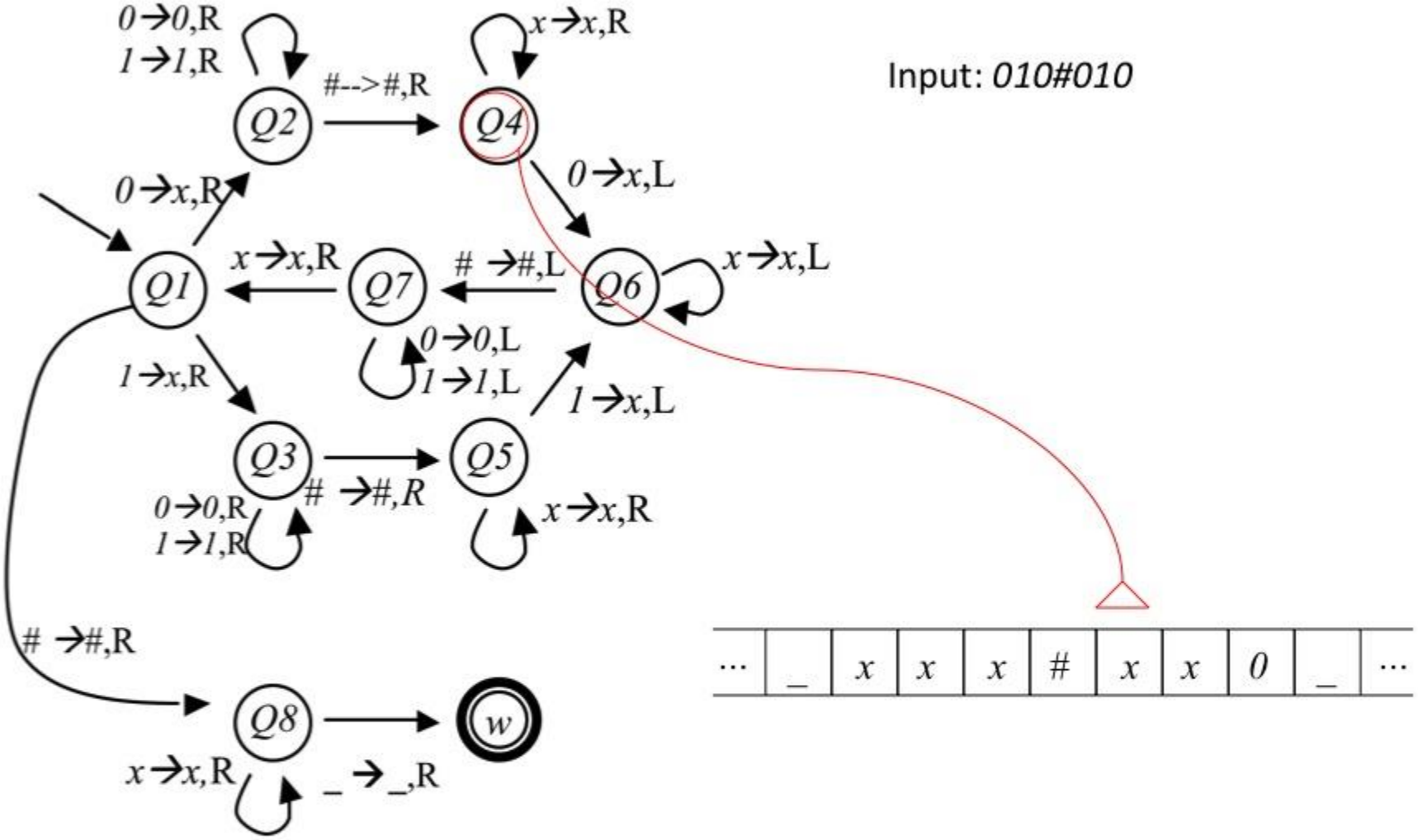

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x 0 # x x 0 _ …

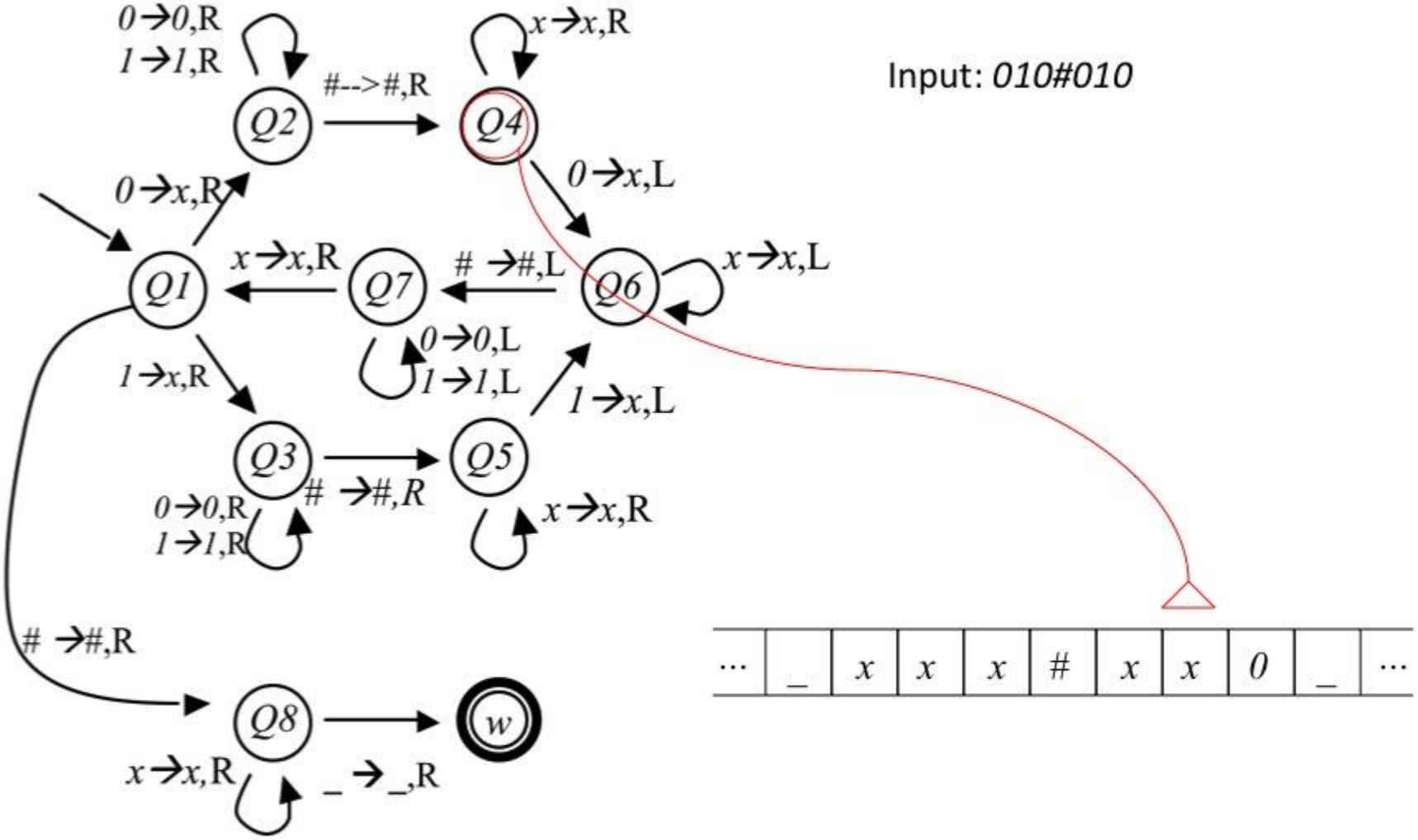

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x 0 # x x 0 _ …

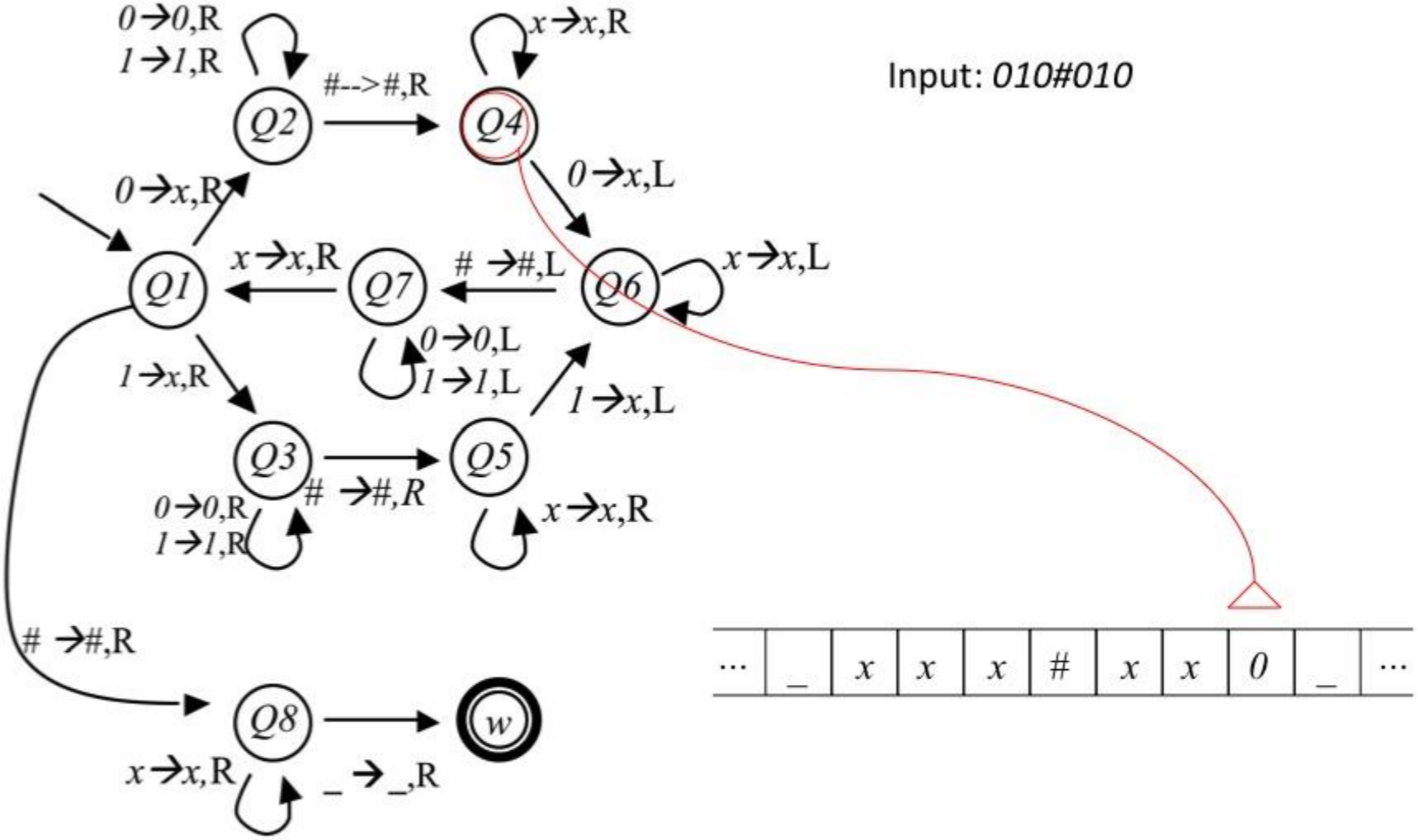

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x 0 # x x 0 _ …

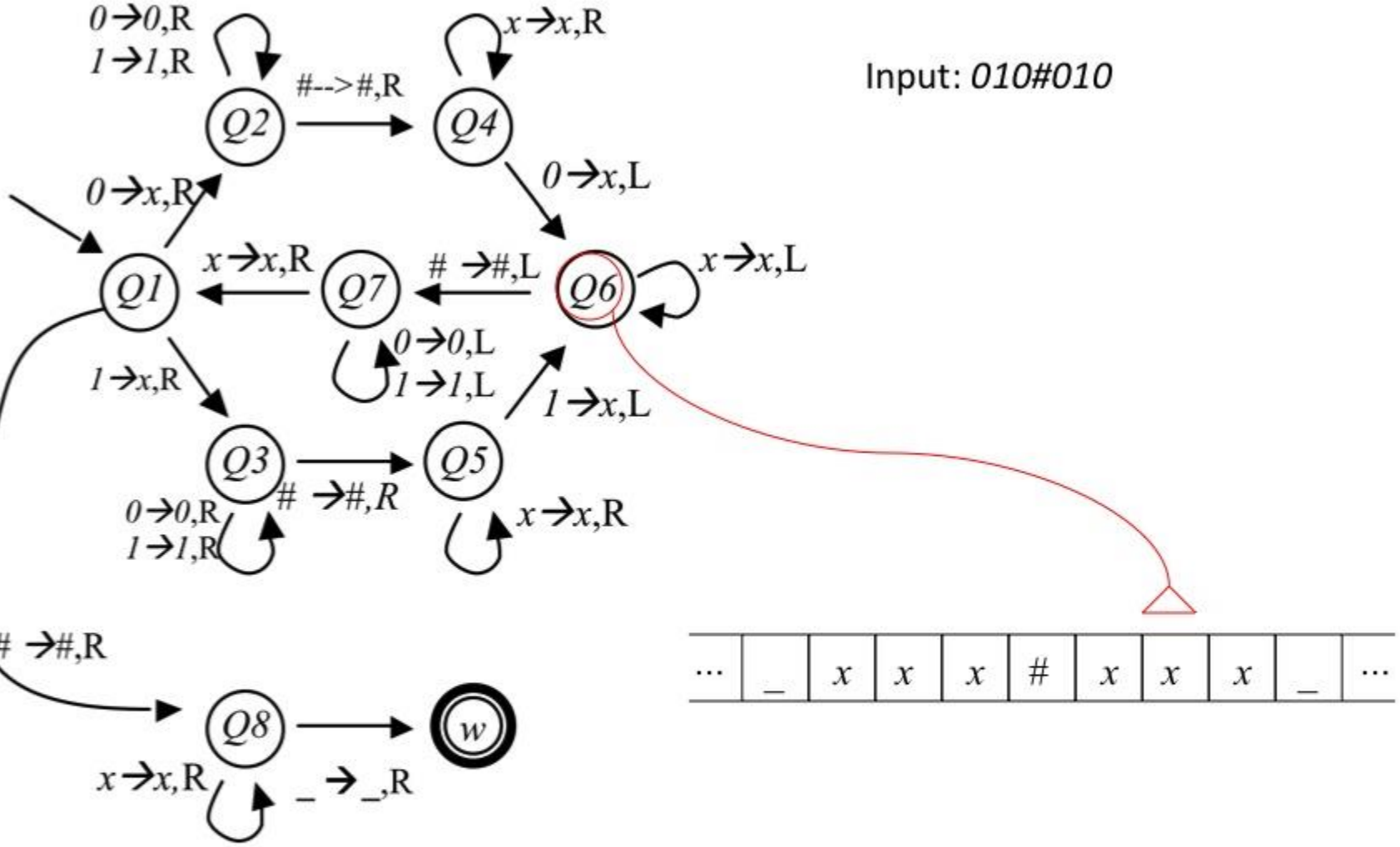

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x 0 # x x 0 _ …

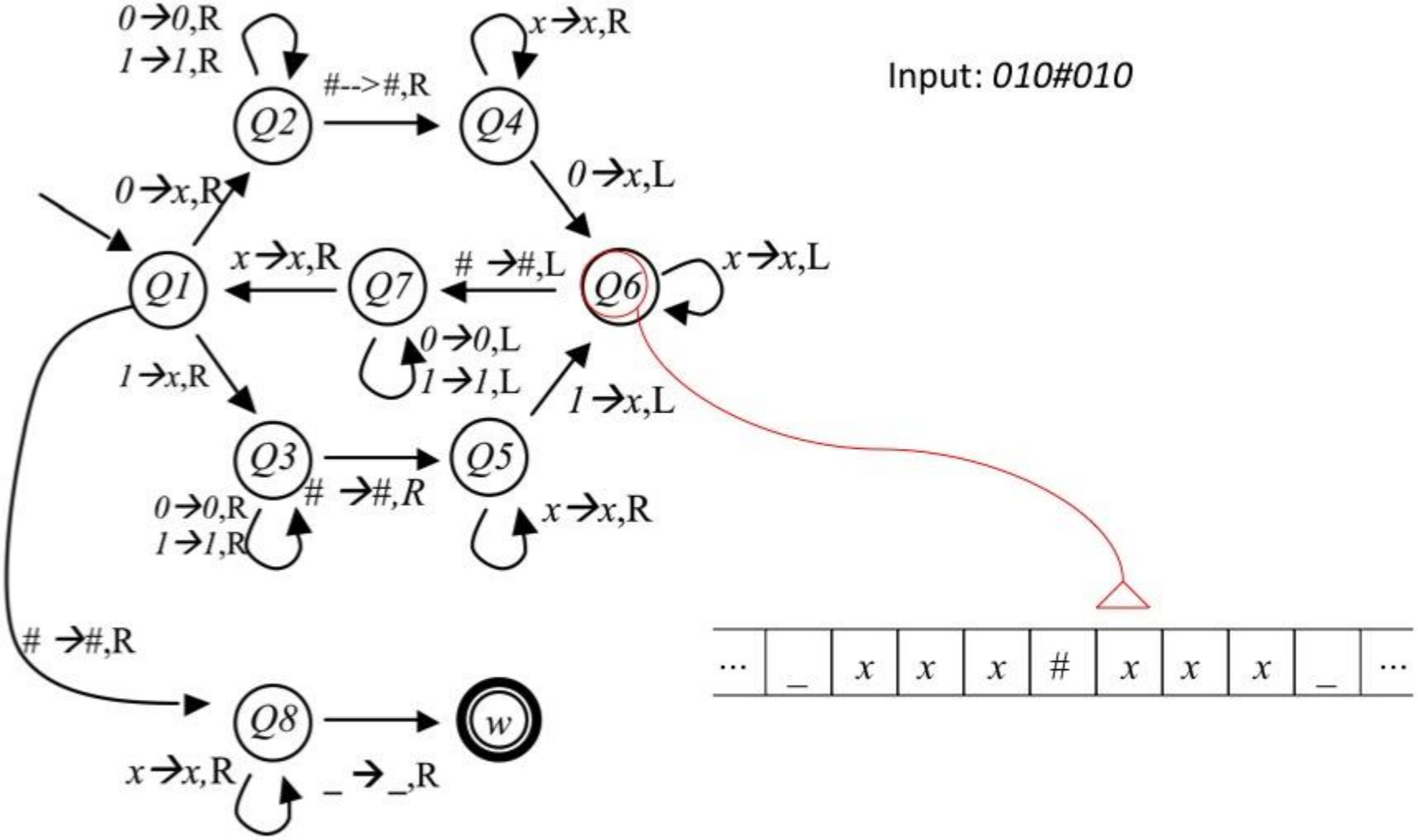

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x x # x x 0 _ …

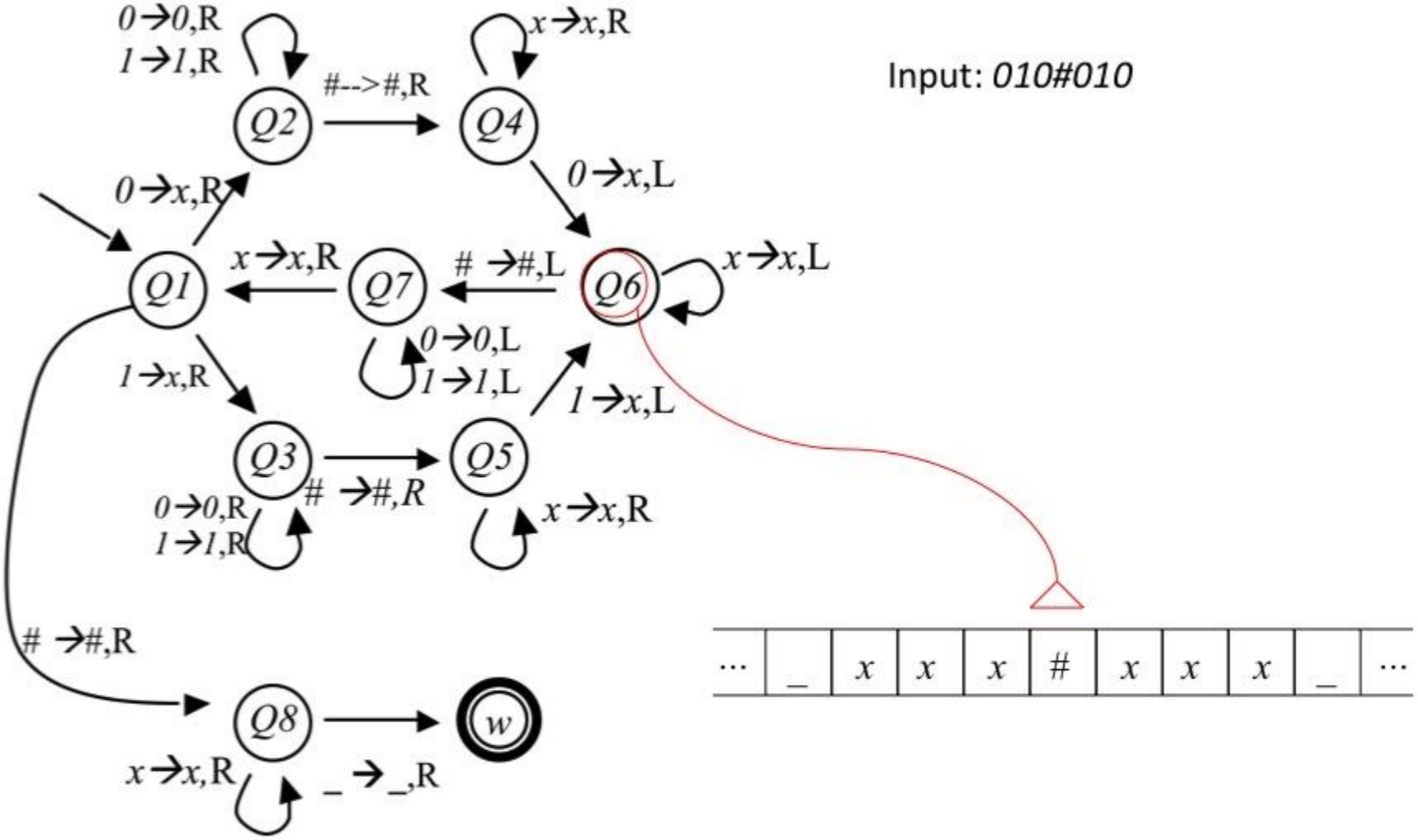

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x x # x x 0 _ …

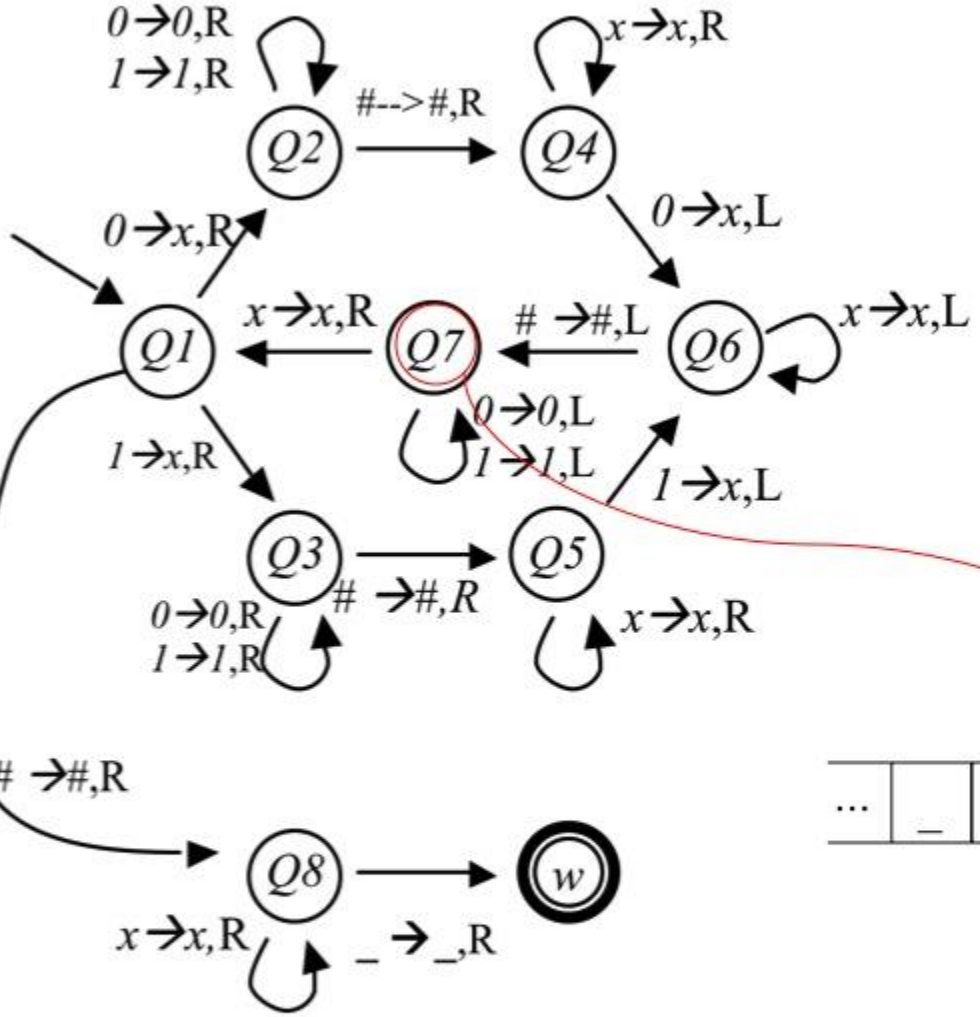

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x x # x x 0 _ …


0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x x # x x 0 _ …

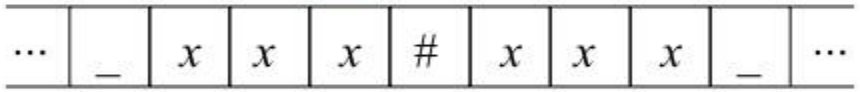

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x x # x x x _ …

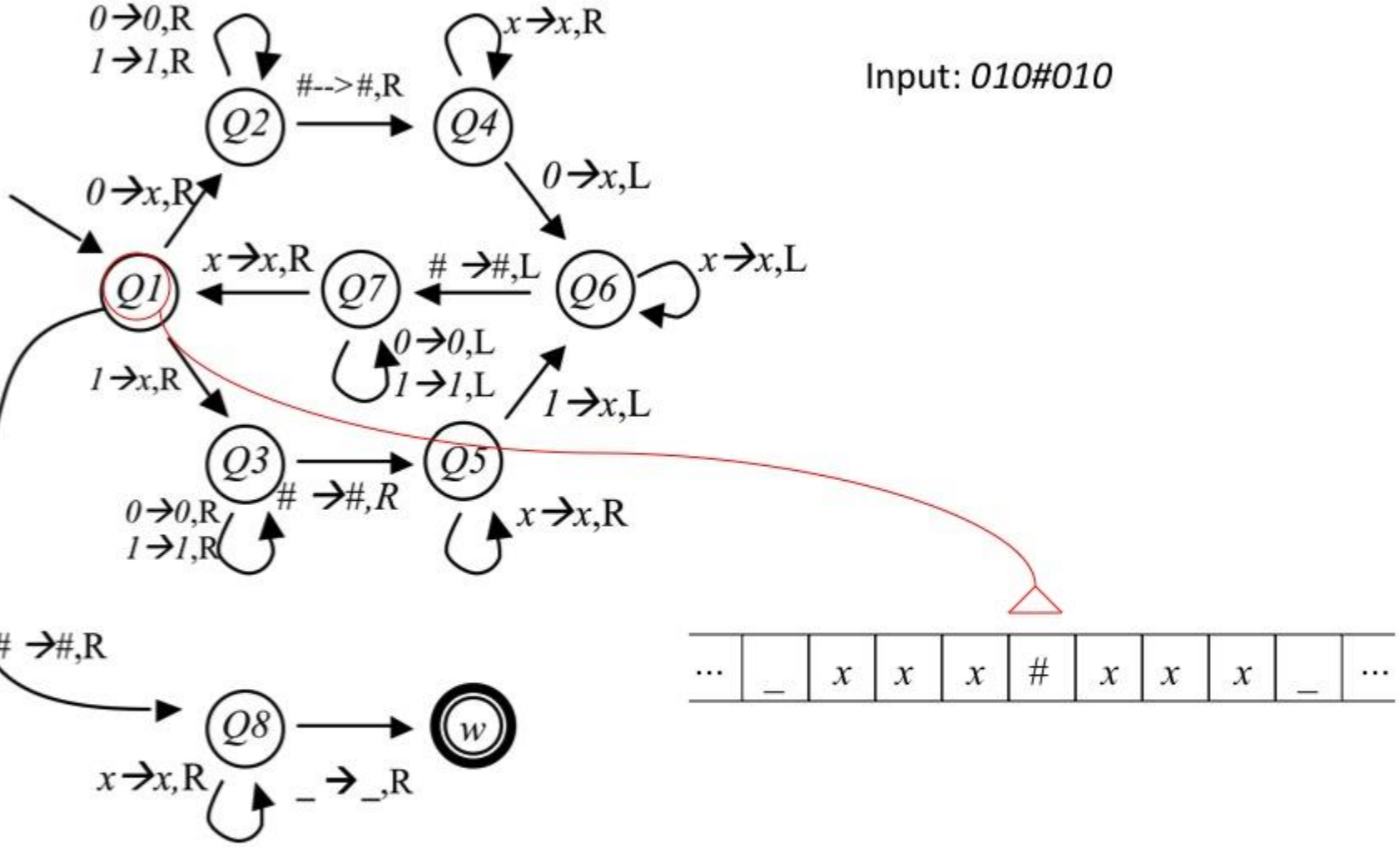

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x x # x x x _ …

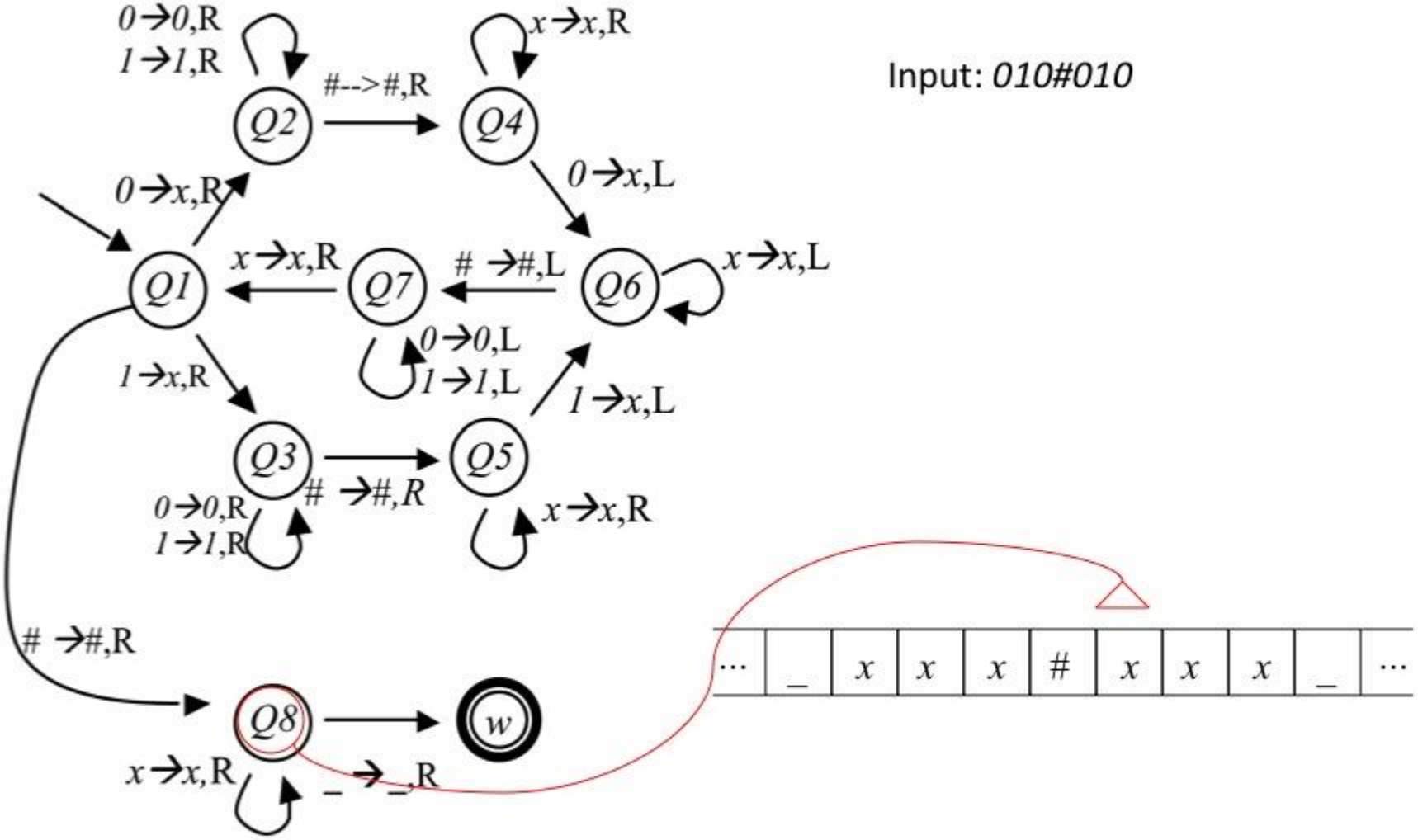

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x x # x x x _ …

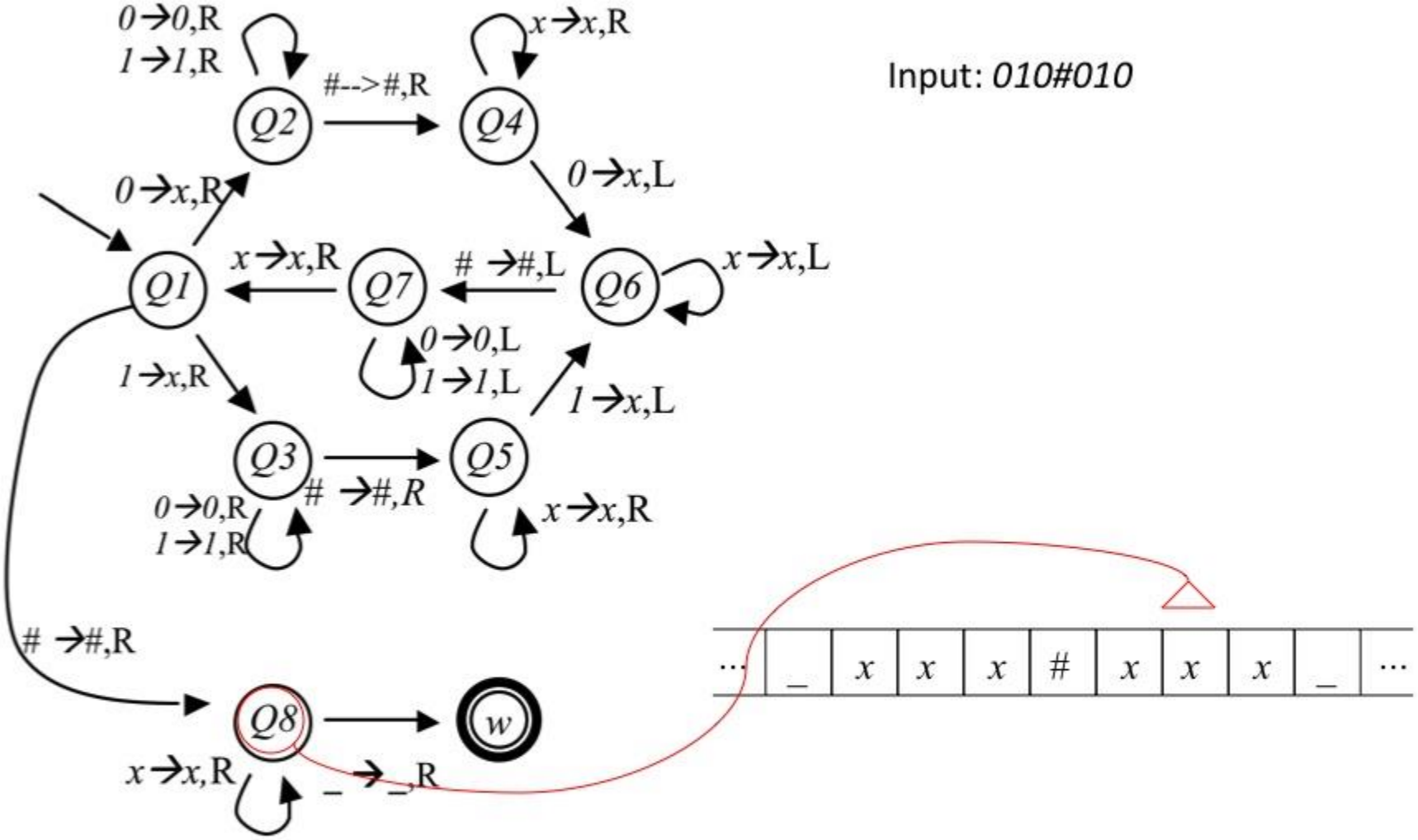


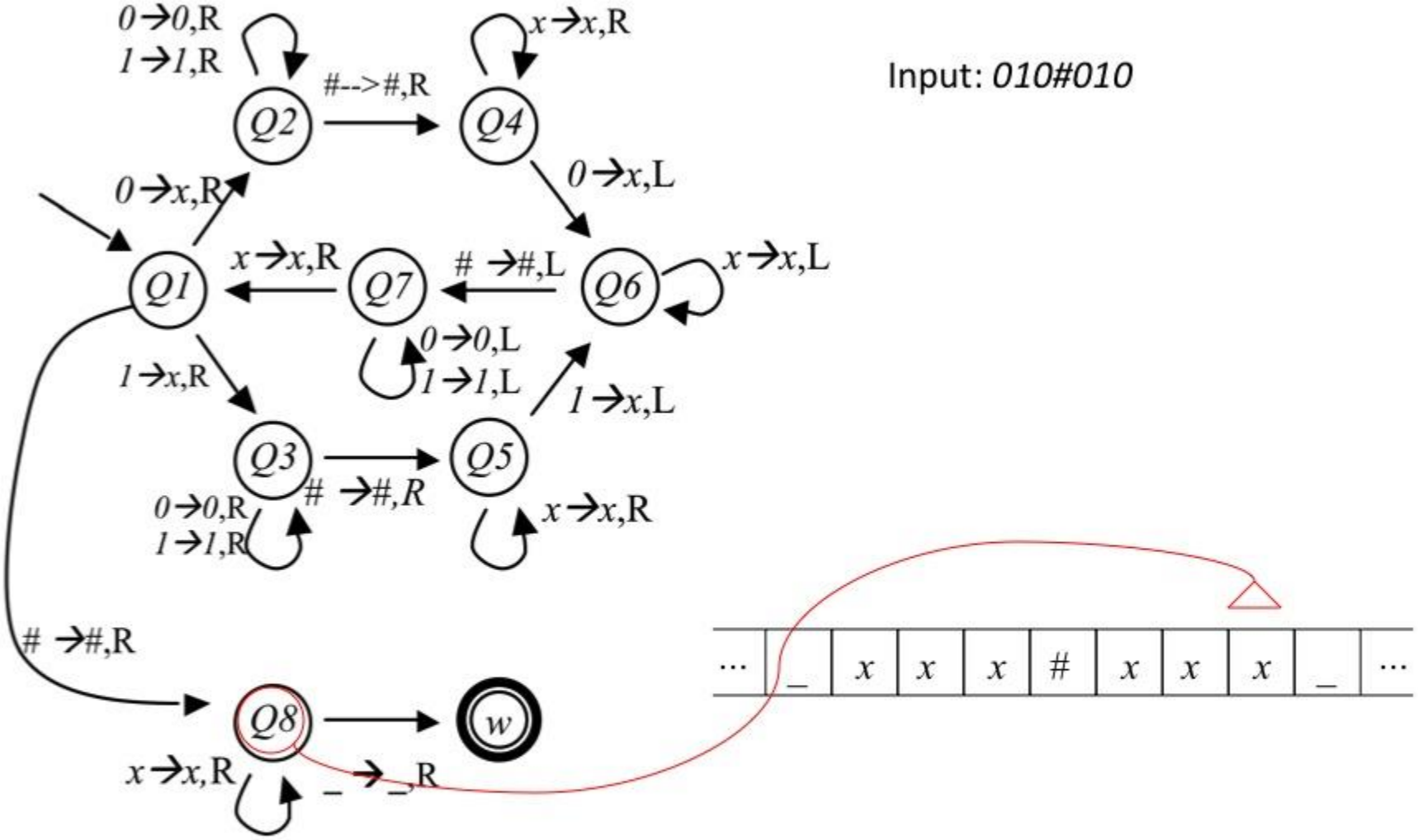
Input: *010#010*

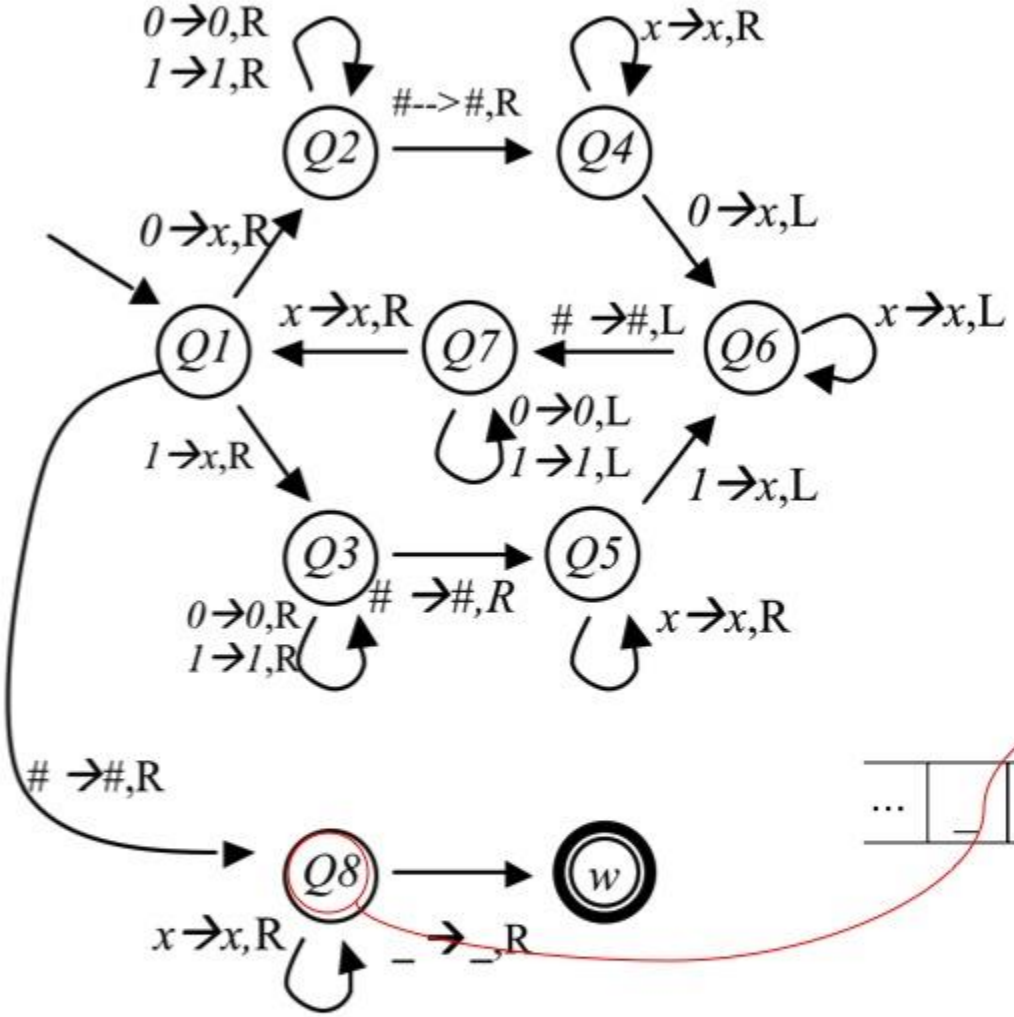



0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x x # x x x _ …

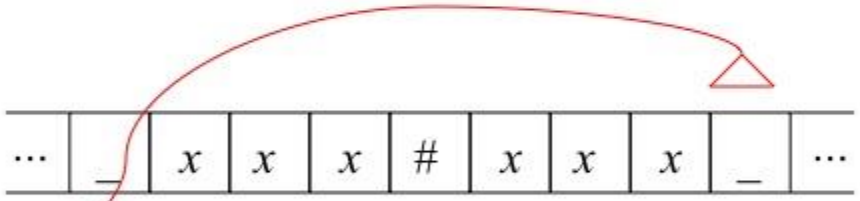

0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x x # x x x _ …


0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x x # x x x _ …


0→0,R
1→1,R
x→x,R
#-->#,R
Q2
Q4
0→x,R
0→x,L
Q1
x→x,R
Q7
→#,L
Q6
x→x,L
0→0,L
1→1,L
1→x,R
1→x,L
Q3
Q5
→#,R
0→0,R
1→1,R
x→x,R
→#,R
Q8
w
x→x,R
_ →_,R
Input: 010#010
… _ x x x # x x x _ …



Input: *010#010*



Implementazione della memoria

Nota: questa è una tecnica utilizzabile in generale

Se vogliamo che M memorizzi una sequenza di k simboli possiamo

- avere una replica per ogni possibile sequenza di k simboli
- M si sposta sulla replica che corrisponde alla sequenza mentre la legge

3.8 Give implementation-level descriptions of Turing machines that decide the following languages over the alphabet $\{0,1\}$.

[A]a. $\{w \mid w$ contains an equal number of 0s and 1s$\}$

b. $\{w \mid w$ contains twice as many 0s as 1s$\}$

c. $\{w \mid w$ does not contain twice as many 0s as 1s$\}$

3.8 (a) "On input string w:

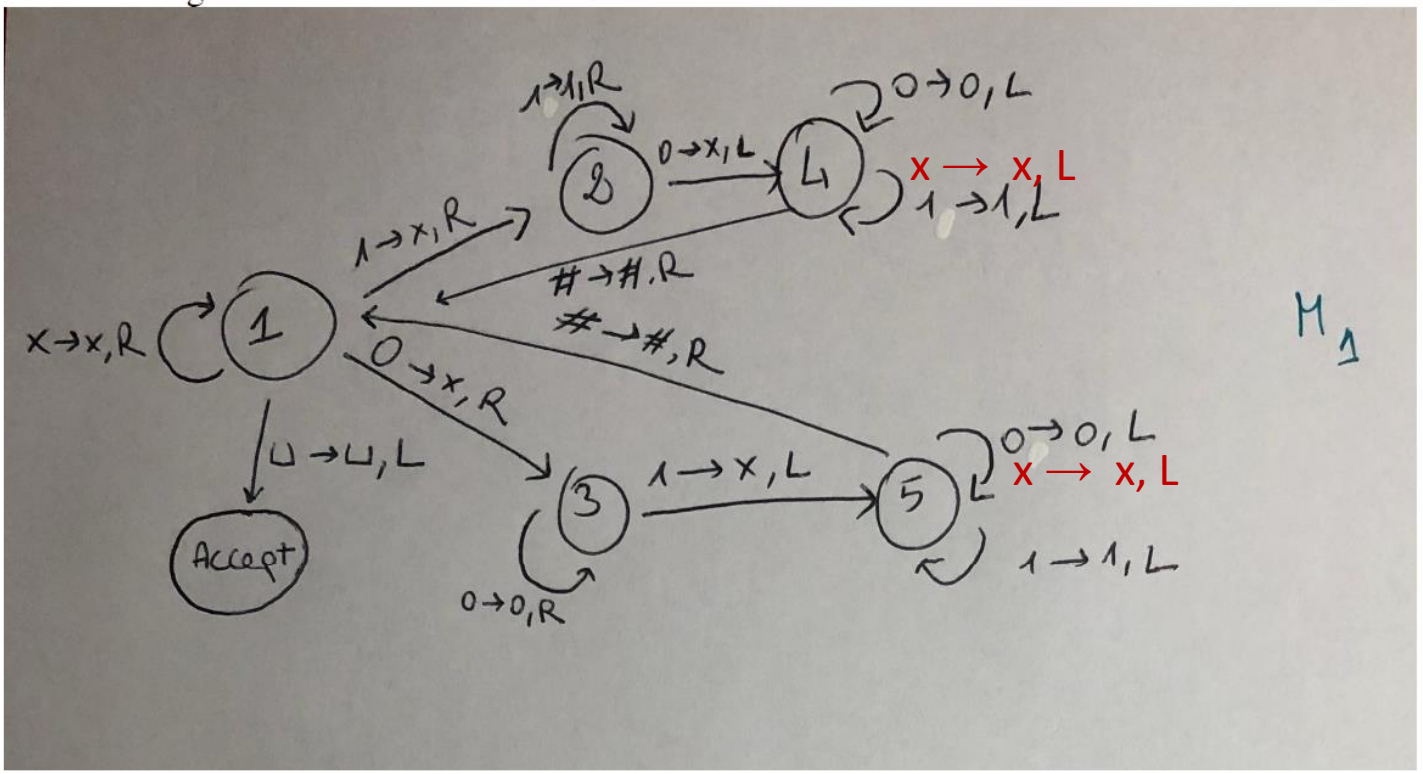
1. Scan the tape and mark the first 0 that has not been marked. If no unmarked 0 is found, go to stage 4. Otherwise, move the head back to the front of the tape.
2. Scan the tape and mark the first 1 that has not been marked. If no unmarked 1 is found, *reject*.
3. Move the head back to the front of the tape and go to stage 1.
4. Move the head back to the front of the tape. Scan the tape to see if any unmarked 1s remain. If none are found, *accept*; otherwise, *reject*."

Sipser, es. 3.8 (a)

Descrizione verbale:

Segno il primo simbolo 0 o 1 che incontro. Scorro a destra cercando la prima istanza di simbolo diverso da quello segnato. Se lo trovo, lo segno e ripeto. Altrimenti (ovvero incontro un blank) rifiuto.

Possibile diagramma di stato:



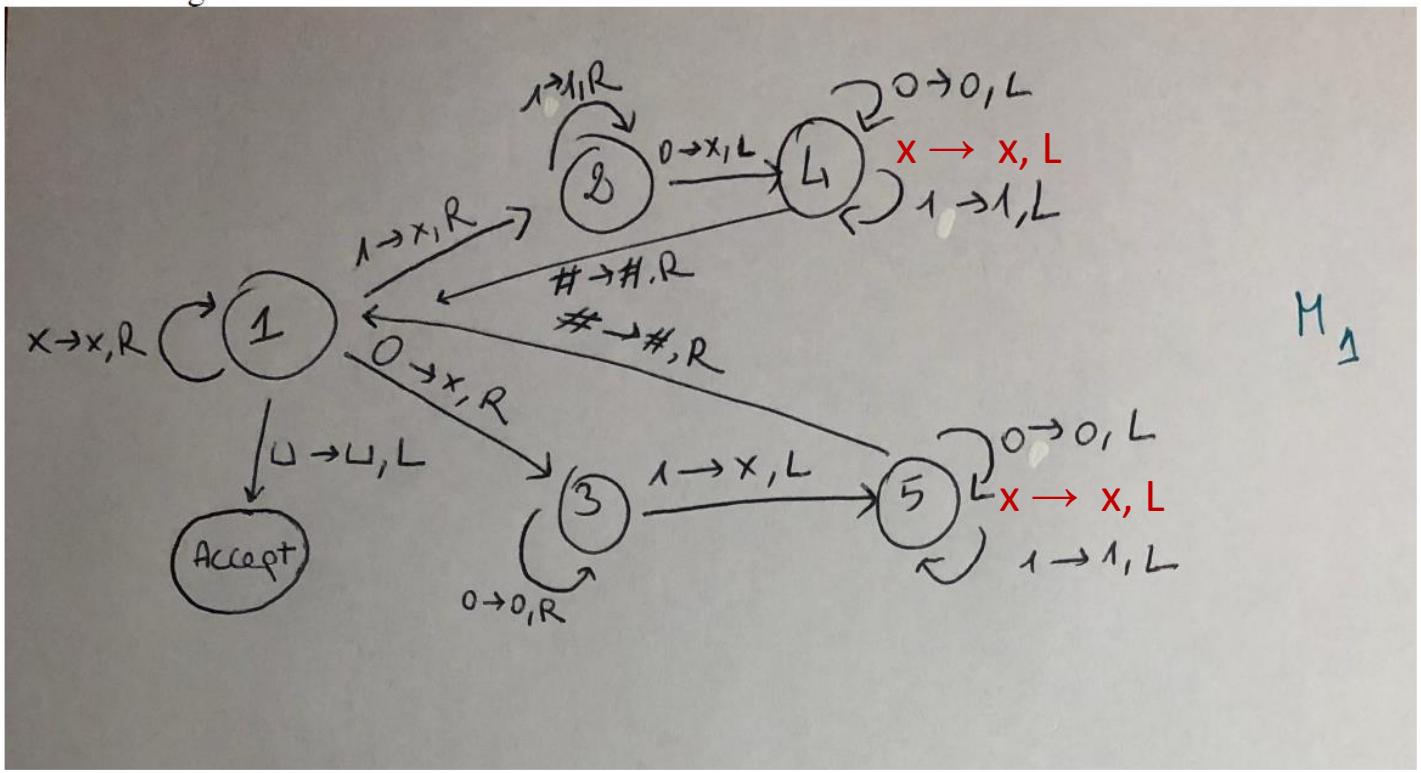
Presuppone che il simbolo nella prima cella sia marcato con #.
Come inserire un meccanismo che consenta ciò?

Sipser, es. 3.8 (a)

Descrizione verbale:

Segno il primo simbolo 0 o 1 che incontro. Scorro a destra cercando la prima istanza di simbolo diverso da quello segnato. Se lo trovo, lo segno e ripeto. Altrimenti (ovvero incontro un blank) rifiuto.

Possibile diagramma di stato:



Nota: il ciclo 1, 2, 4, 1 è una «replica» del ciclo 1, 3, 5, 1, ma riferito ad un primo simbolo 1, anziché 0.

1. Progettare una macchina di Turing che sposta l'input a destra di una casella.

2. Progettare una macchina di Turing che calcola il successore in binario.

Questi esercizi ci mostrano cosa può fare una MdT:
copiare, spostare, calcolare successore, somma, differenza (in unario o binario), …
contare, ovvero riconoscere occorrenze di ugual numero, lunghezza pari ad una potenza di 2, …..

Esercizio

Progettare una MdT che decida il linguaggio

$$L = \{ 0^{2^n} \mid n \geq 0 \}$$

e che sia **concettualmente diversa** da quella vista in una precedente lezione.