university of groningen

Languages and Machines

L9: Variations of Turing machines

Jorge A. Pérez

Bernoulli Institute for Math, Computer Science, and AI
University of Groningen, Groningen, the Netherlands

May 16, 2023

Languages and Their Machines

Regular	$\leftrightarrow$	Finite State Machines (FSMs)
Context-free	$\leftrightarrow$	Pushdown Machines
Context-sensitive	$\leftrightarrow$	Linearly-bounded Machines
Decidable	$\leftrightarrow$	**Always-terminating Turing Machines**
Semi-decidable	$\leftrightarrow$	**Turing Machines**

Outline

From Last Lecture

Variations of TMs
- Multitrack TMs
- The Example Revisited (I)
- Multitape TMs
- The Example Revisited (II)

Simulating Multitape with Multitrack

Non-Deterministic TMs (NTMs)

Closure Properties

Turing Machines (TMs)

A (simple) **Turing machine** M includes

- A set of **states** Q, with **start state** $q_0 \in Q$
- The **tape alphabet** Γ is such that $\Gamma \cap Q = \emptyset$.
  There is a **blank symbol** $\texttt{B} \in \Gamma \setminus \Sigma$
- The **input alphabet** Σ is such that $\Sigma \subseteq \Gamma \setminus \{\texttt{B}\}$

Graphically:

$\cdots$	$\texttt{B}$	$\texttt{B}$	$\texttt{B}$	$\texttt{B}$	$\texttt{B}$	w_1	w_2	w_3	w_4	w_5	$\texttt{B}$	$\cdots$

$$\begin{array}{c} \uparrow \\ \boxed{q_i} \end{array}$$

Turing Machines (TMs)

A (simple) **Turing machine** M includes

- A set of **states** Q, with **start state** $q_0 \in Q$
- The **tape alphabet** Γ is such that $\Gamma \cap Q = \emptyset$.
  There is a **blank symbol** $\mathtt{B} \in \Gamma \setminus \Sigma$
- The **input alphabet** Σ is such that $\Sigma \subseteq \Gamma \setminus \{\mathtt{B}\}$

Graphically:

$\cdots$	$\mathtt{B}$	$\mathtt{B}$	$\mathtt{B}$	$\mathtt{B}$	$\mathtt{B}$	w_1	w_2	w_3	w_4	w_5	$\mathtt{B}$	$\cdots$

$$\uparrow$$

$$\boxed{q_i}$$

A transition:

- changes the state
- writes a symbol on the square scanned by the head
- moves the head one square to the left or to the right

Turing Machines (TMs)

A (simple) **Turing machine** M is a quintuple $(Q, \Sigma, \Gamma, \delta, q_0)$ where

- Q is a set of **states**
- $q_0 \in Q$ is the **start state**
- Γ is the **tape alphabet**, a set of symbols disjoint from Q. Contains a **blank symbol** $\mathtt{B}$, not in Σ
- $\Sigma \subseteq \Gamma \setminus \{\mathtt{B}\}$ is the **input alphabet**
- The transition function δ is a *partial* function such that

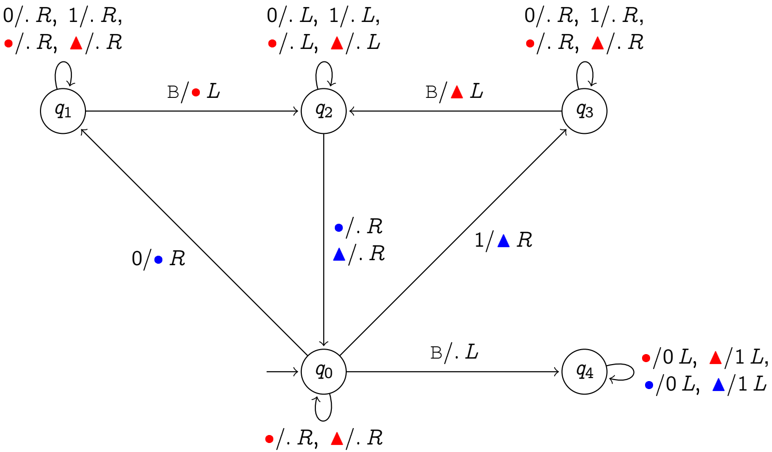
$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

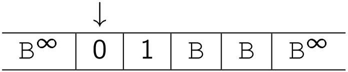
  If $\delta(q, X)$ is undefined then $\delta(q, X) = \perp$.

A set of accepting states $F \subseteq Q$ is convenient for defining acceptance, although it is not indispensable.

From Last Lecture: Example 2

A TM that duplicates the input string $w \in \{0, 1\}^*$.

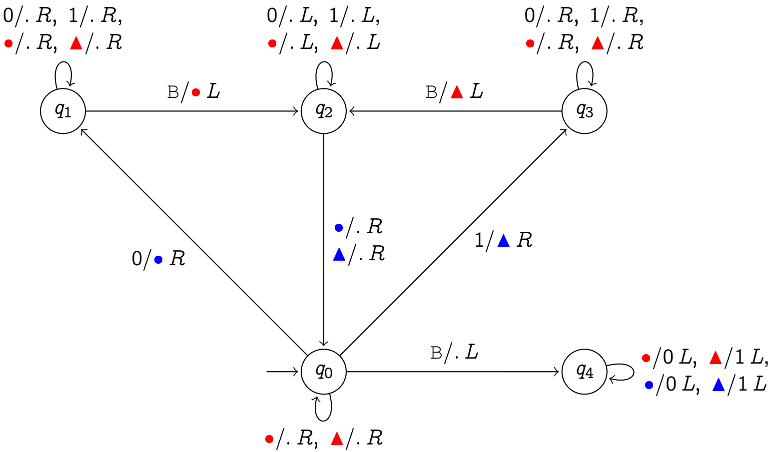




(State = q_0)

From Last Lecture: Example 2

A TM that duplicates the input string $w \in \{0, 1\}^*$.

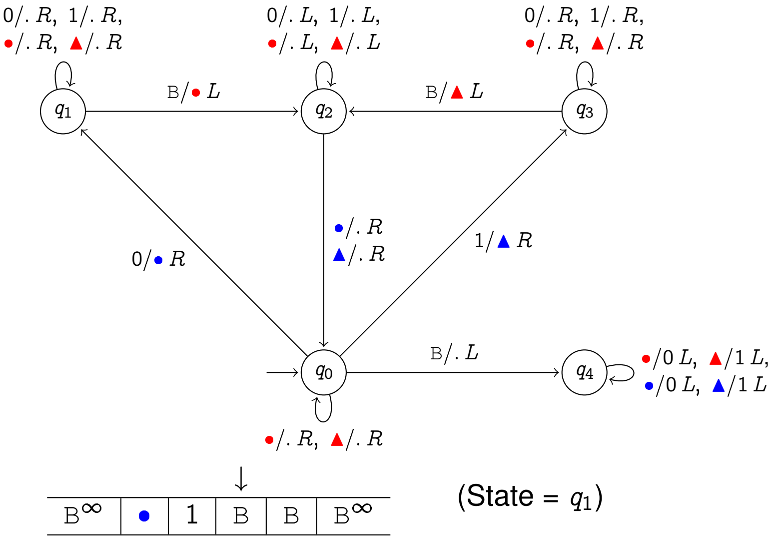


B^∞	●	1	B	B	B^∞

(State = q_1)

From Last Lecture: Example 2

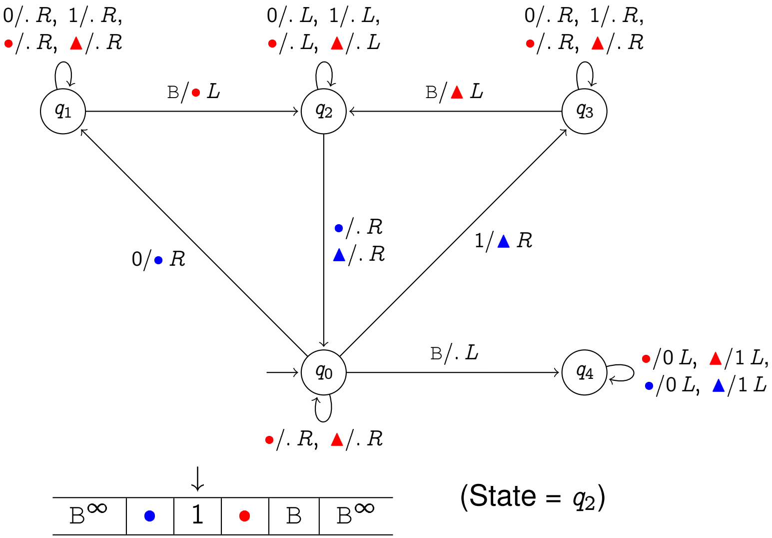
A TM that duplicates the input string $w \in \{0,1\}^*$.



(State = q_1)

From Last Lecture: Example 2

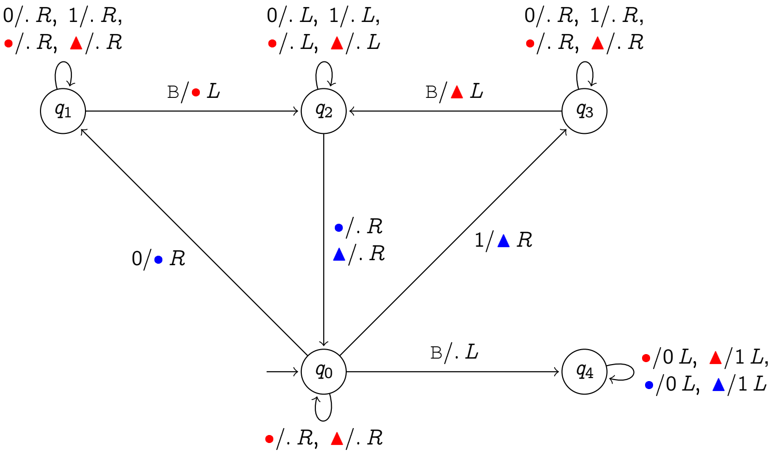
A TM that duplicates the input string $w \in \{0,1\}^*$.

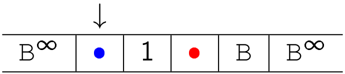


(State = q_2)

From Last Lecture: Example 2

A TM that duplicates the input string $w \in \{0, 1\}^*$.

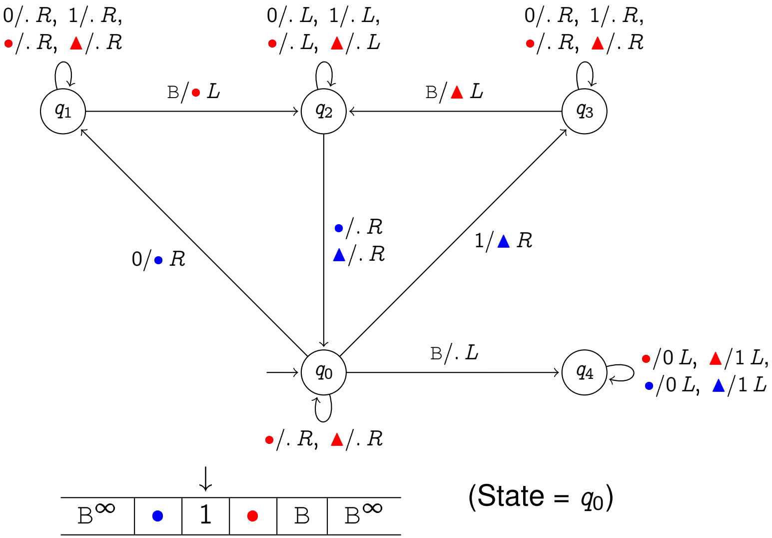




(State = q_2)

From Last Lecture: Example 2

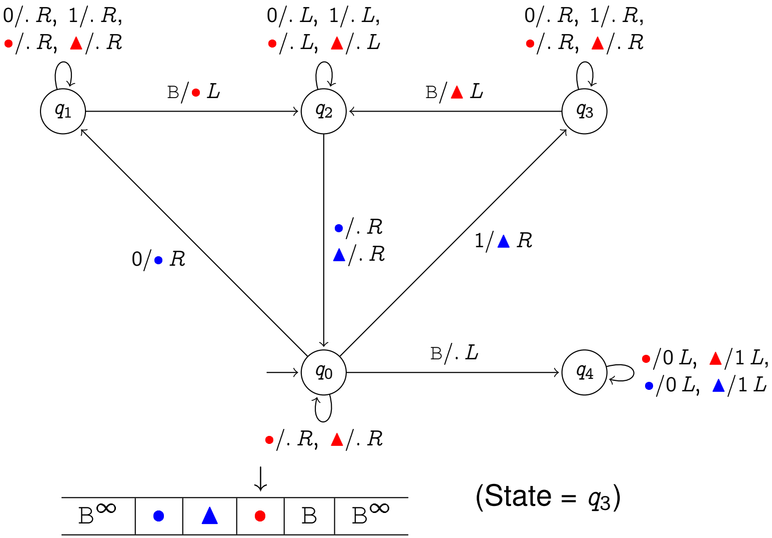
A TM that duplicates the input string $w \in \{0, 1\}^*$.



(State = q_0)

From Last Lecture: Example 2

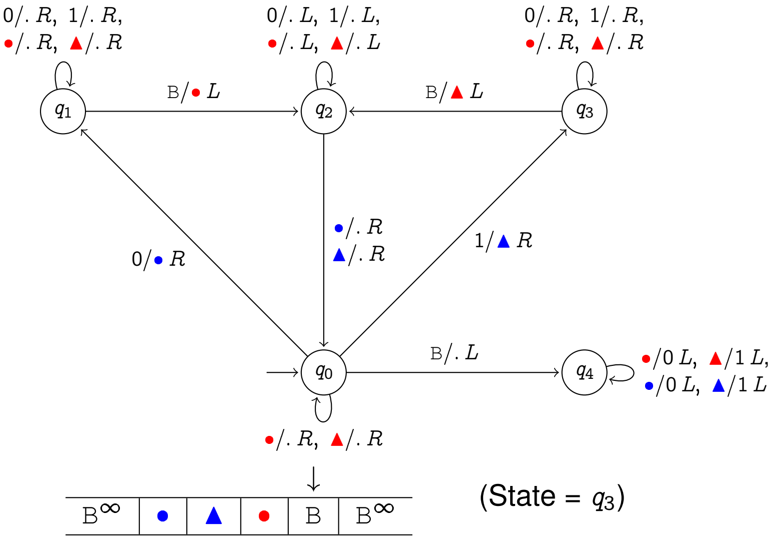
A TM that duplicates the input string $w \in \{0, 1\}^*$.



(State = q_3)

From Last Lecture: Example 2

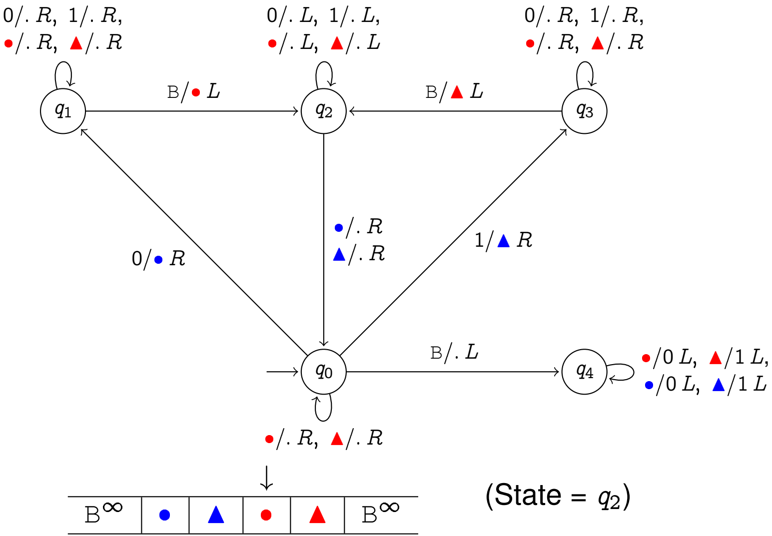
A TM that duplicates the input string $w \in \{0,1\}^*$.



(State = q_3)

From Last Lecture: Example 2

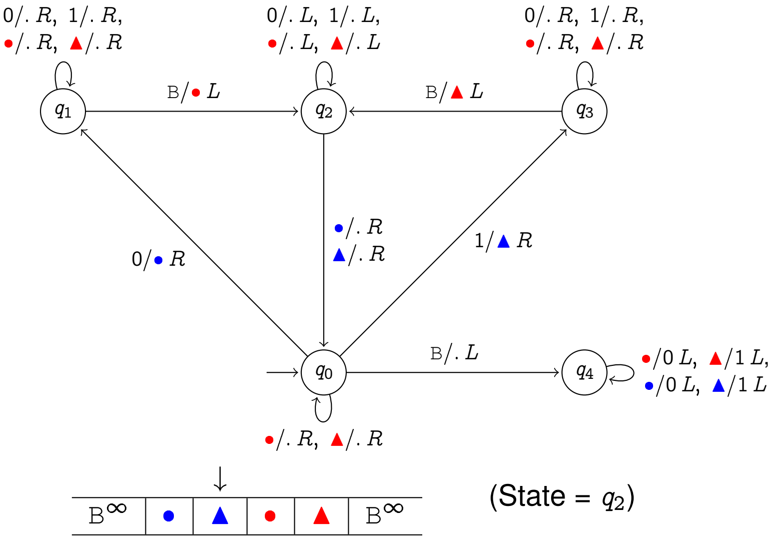
A TM that duplicates the input string $w \in \{0, 1\}^*$.



(State = q_2)

From Last Lecture: Example 2

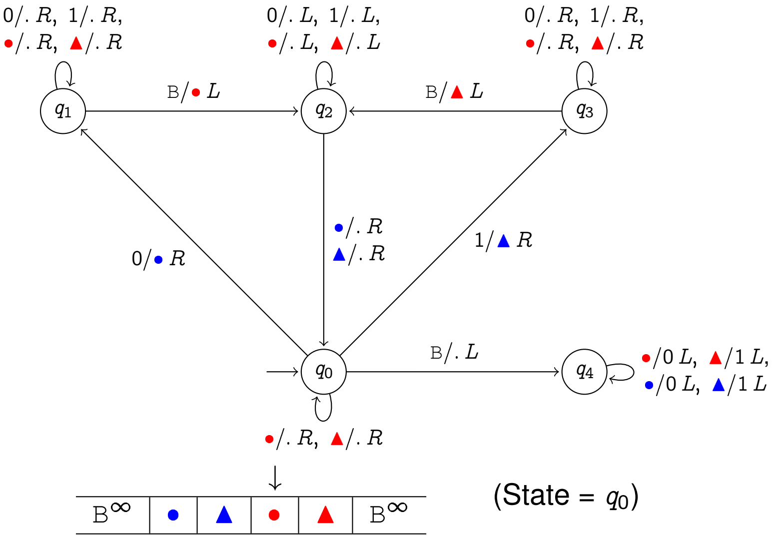
A TM that duplicates the input string $w \in \{0, 1\}^*$.



(State = q_2)

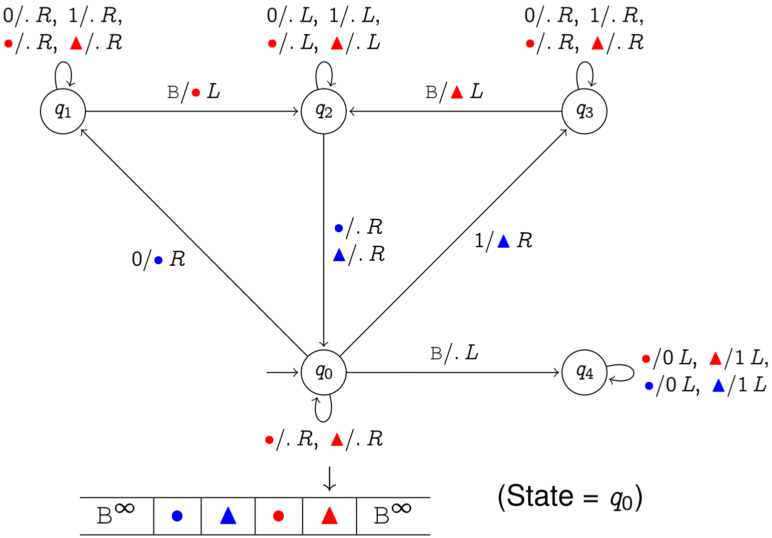
From Last Lecture: Example 2

A TM that duplicates the input string $w \in \{0,1\}^*$.



From Last Lecture: Example 2

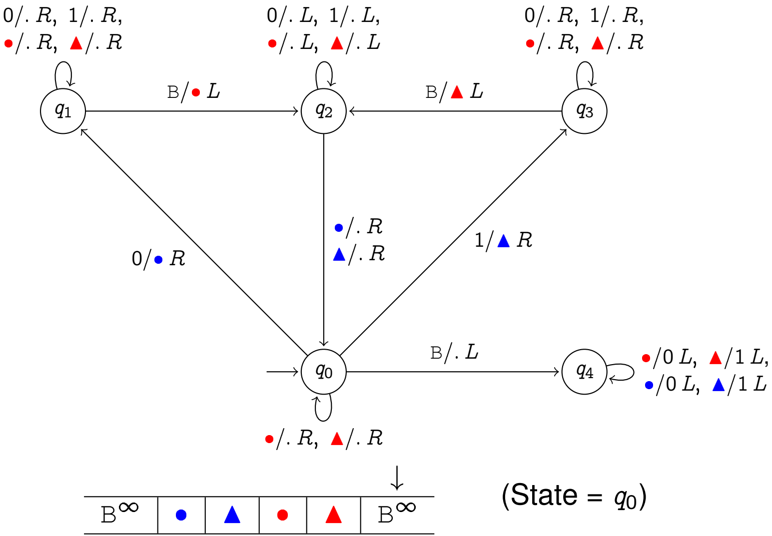
A TM that duplicates the input string $w \in \{0,1\}^*$.



(State = q_0)

From Last Lecture: Example 2

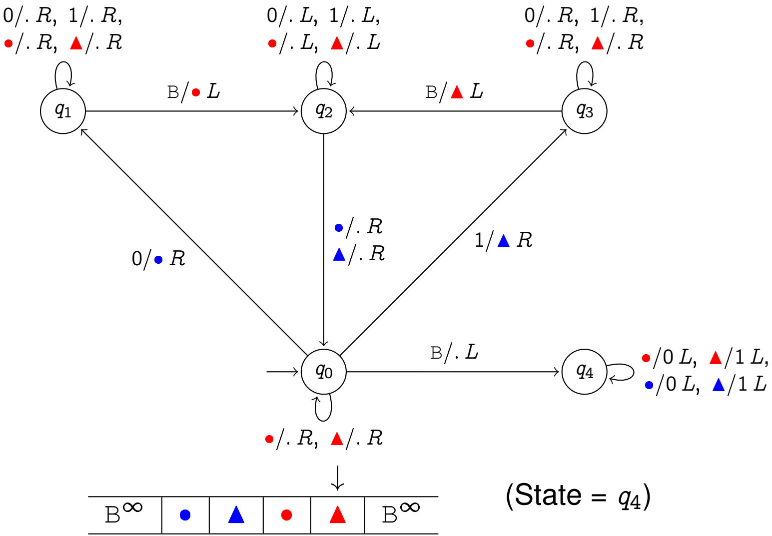
A TM that duplicates the input string $w \in \{0,1\}^*$.



(State = q_0)

From Last Lecture: Example 2

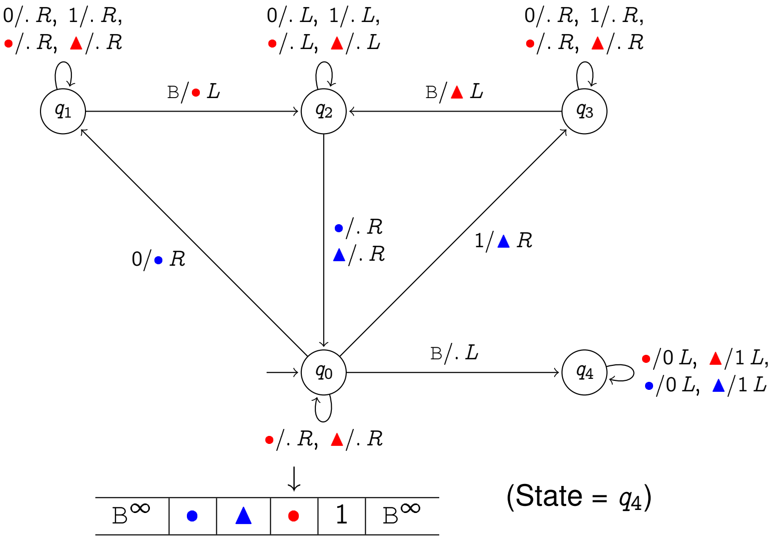
A TM that duplicates the input string $w \in \{0, 1\}^*$.



(State = q_4)

From Last Lecture: Example 2

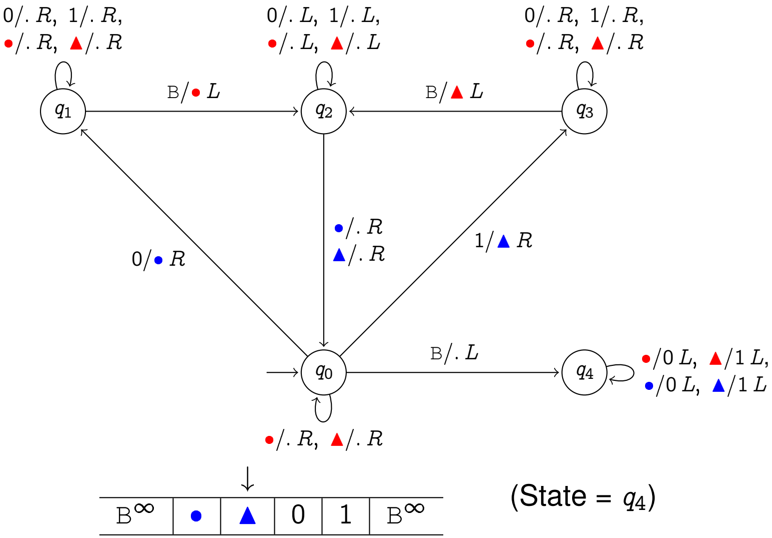
A TM that duplicates the input string $w \in \{0,1\}^*$.



(State = q_4)

From Last Lecture: Example 2

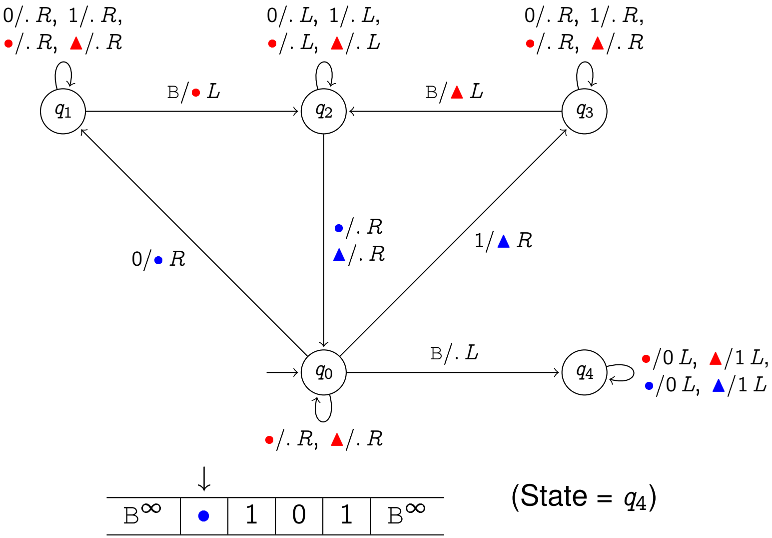
A TM that duplicates the input string $w \in \{0, 1\}^*$.



(State = q_4)

From Last Lecture: Example 2

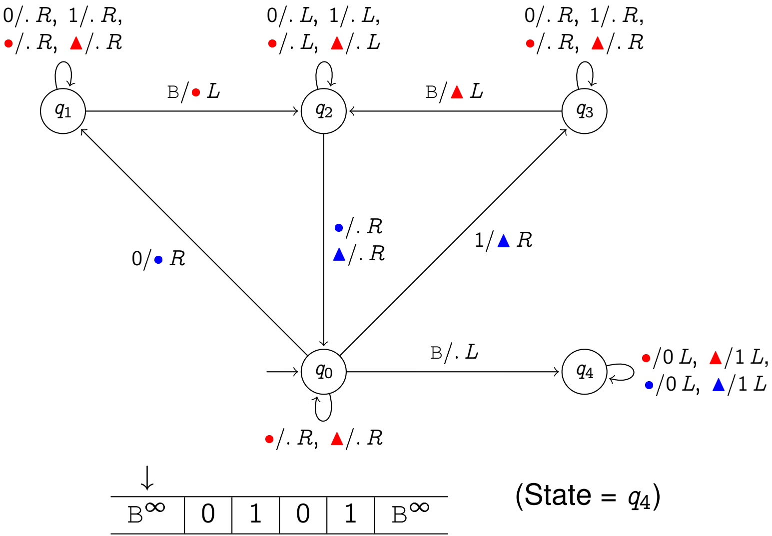
A TM that duplicates the input string $w \in \{0, 1\}^*$.



(State = q_4)

From Last Lecture: Example 2

A TM that duplicates the input string $w \in \{0,1\}^*$.



(State = q_4)

Terminology

A TM is **always terminating** if it terminates for every input.

Let L be a language.

- L is **semi-decidable** (or **recursively enumerable, RE**) if there exists a TM M such that $L = L(M)$.
- L is **decidable** (or **recursive**) if there is an always terminating TM that accepts L by termination in an accepting state.
- If L is decidable, then it is also semi-decidable. The converse doesn't hold!

Outline

Variations of TMs

- Extensions of TMs: multitrack, multitape, non-deterministic
- These generalized machines are convenient...

Variations of TMs

- Extensions of TMs: multitrack, multitape, non-deterministic
- These generalized machines are convenient...
- ...but don't add expressive power: the languages accepted by them are precisely those accepted by standard TMs

Variations of TMs

- Extensions of TMs: multitrack, multitape, non-deterministic
- These generalized machines are convenient...
- ...but don't add expressive power: the languages accepted by them are precisely those accepted by standard TMs

The extensions will be useful next lecture, when discussing Universal Turing machines.

Disclaimer

- The TMs discussed in the previous lecture are **two-way**: the tape extends indefinitely in both directions:

···	-5	-4	-3	-2	-1	0	1	2	3	4	5	···
···												···

$$\uparrow$$
$$\boxed{q_0}$$

Disclaimer

- The TMs discussed in the previous lecture are **two-way**: the tape extends indefinitely in both directions:

...	-5	-4	-3	-2	-1	0	1	2	3	4	5	...
...												...
						$\uparrow$						
						q_0						

- But this is actually an extension of a **simple** TM in which there is a left boundary: the tape extends indefinitely only in one direction:

0	1	2	3	4	5	...
						...
$\uparrow$						
q_0						

- A simple TM can simulate the actions of a two-way TM. This can be proved by using a TM with **two tracks**.

Multitrack Turing Machines (TMs)

- A TM in which the tape is divided into tracks
- A tape position in an n-track tape contains n symbols from the tape alphabet. The TM reads an entire tape position.

Multitrack Turing Machines (TMs)

- A TM in which the tape is divided into tracks
- A tape position in an n-track tape contains n symbols from the tape alphabet. The TM reads an entire tape position.
- In the case of a two-track TM, we would have:

Track 2	$\cdots$	1	2	3	4	5	6	7	$\cdots$
Track 1	$\cdots$	a	b	c	d	e	f	g	$\cdots$

$$\uparrow$$
$$\boxed{q_i}$$

The machine simultaneously reads b and 2.

Multitrack Turing Machines (TMs)

- A TM in which the tape is divided into tracks
- A tape position in an n-track tape contains n symbols from the tape alphabet. The TM reads an entire tape position.
- In the case of a two-track TM, we would have:

Track 2	$\cdots$	1	2	3	4	5	6	7	$\cdots$
Track 1	$\cdots$	a	b	c	d	e	f	g	$\cdots$

$$\uparrow$$
$$\boxed{q_i}$$

The machine simultaneously reads b and 2.

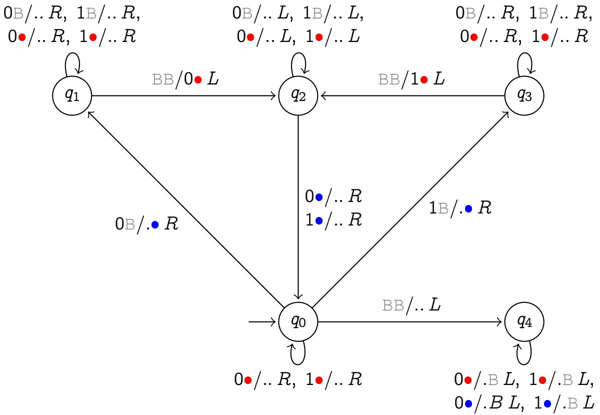
- A multitrack TM can be represented as a one-track TM using tuples. In the case of two-track TMs, ordered pairs suffice:

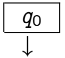
$\cdots$	$(a,1)$	$(b,2)$	$(c,3)$	$(d,4)$	$(e,5)$	$(f,6)$	$(g,7)$	$\cdots$

$$\uparrow$$
$$\boxed{q_i}$$

Example 2

A multitrack TM that duplicates the input string $w \in \{0,1\}^*$.

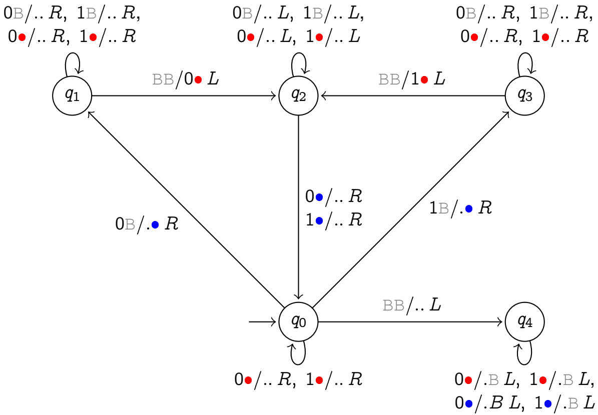




B^∞	B	0	1	B	B	B^∞
B^∞	B	B	B	B	B	B^∞

Example 2

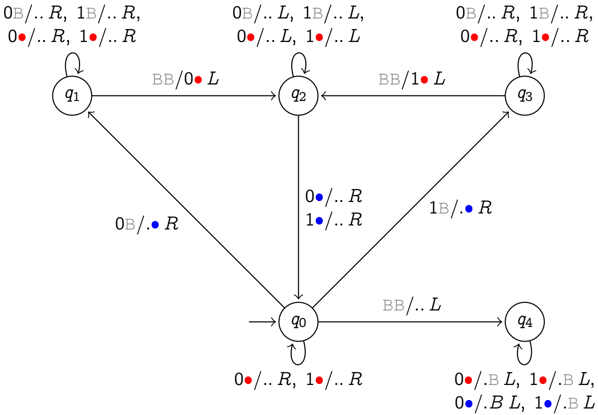
A multitrack TM that duplicates the input string $w \in \{0,1\}^*$.



B^∞	B	0	1	B	B	B^∞
B^∞	B	B	B	B	B	B^∞

Example 2

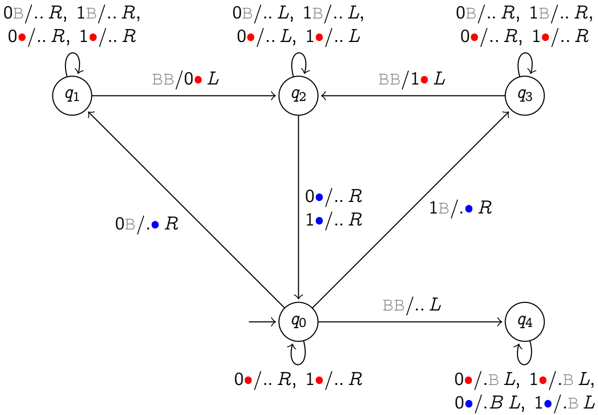
A multitrack TM that duplicates the input string $w \in \{0,1\}^*$.

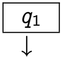


B^∞	B	0	1	B	B	B^∞
B^∞	B	●	B	B	B	B^∞

Example 2

A multitrack TM that duplicates the input string $w \in \{0, 1\}^*$.

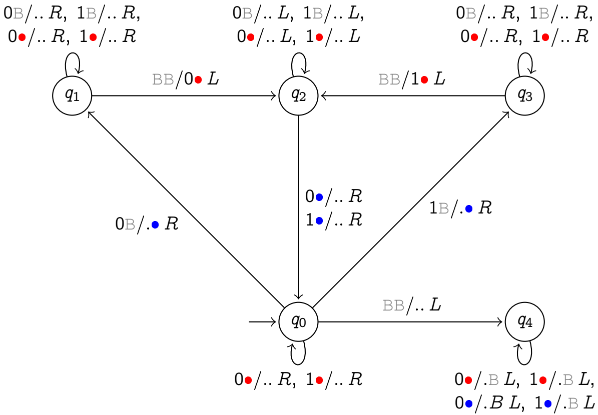


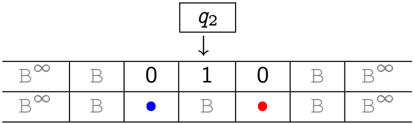


B^∞	B	0	1	B	B	B^∞
B^∞	B	●	B	B	B	B^∞

Example 2

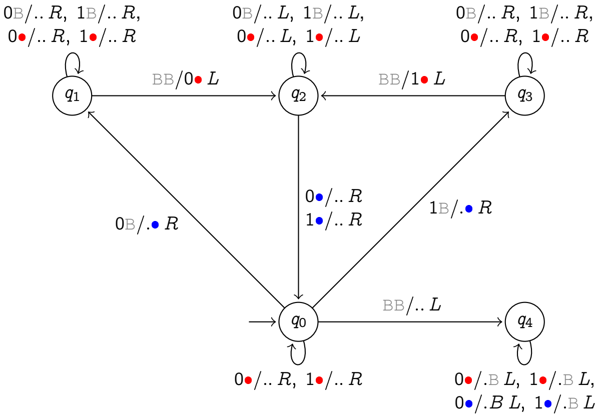
A multitrack TM that duplicates the input string $w \in \{0,1\}^*$.





Example 2

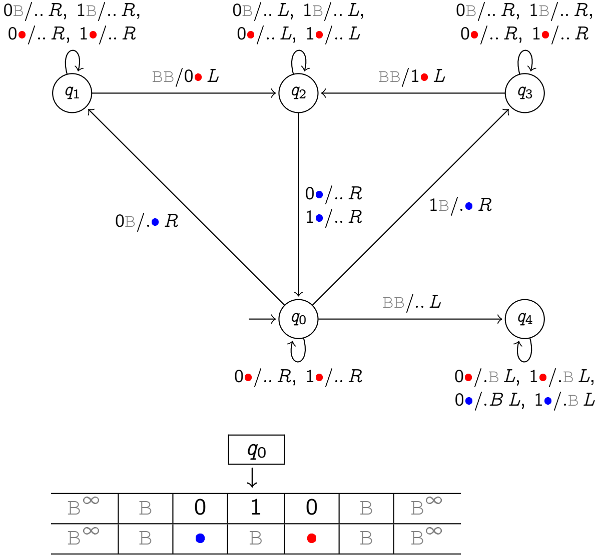
A multitrack TM that duplicates the input string $w \in \{0,1\}^*$.



B^∞	B	0	1	0	B	B^∞
B^∞	B	•	B	•	B	B^∞

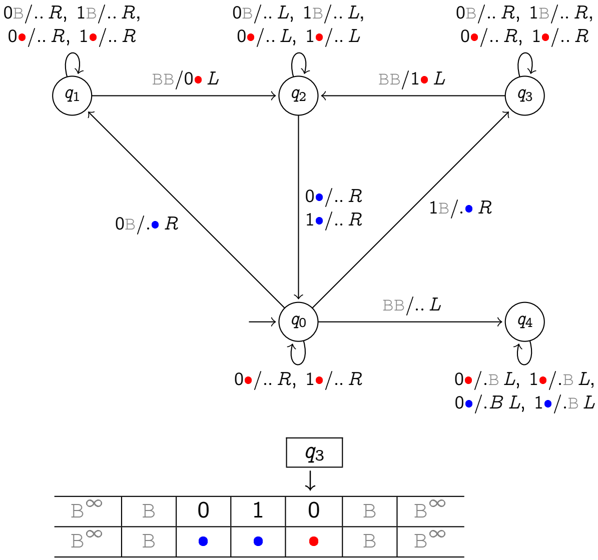
Example 2

A multitrack TM that duplicates the input string $w \in \{0,1\}^*$.



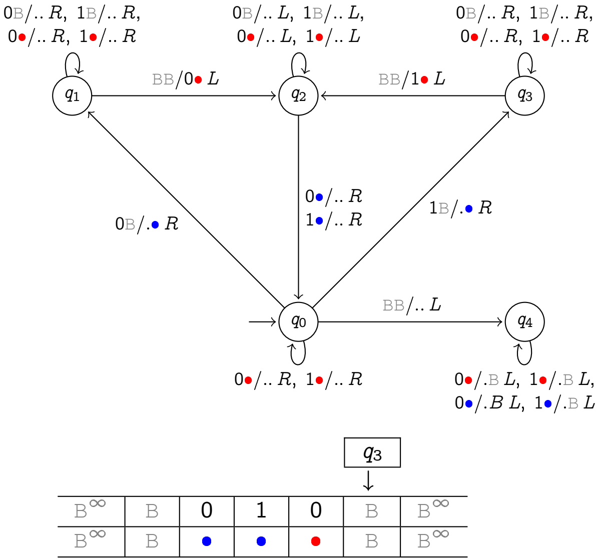
Example 2

A multitrack TM that duplicates the input string $w \in \{0, 1\}^*$.



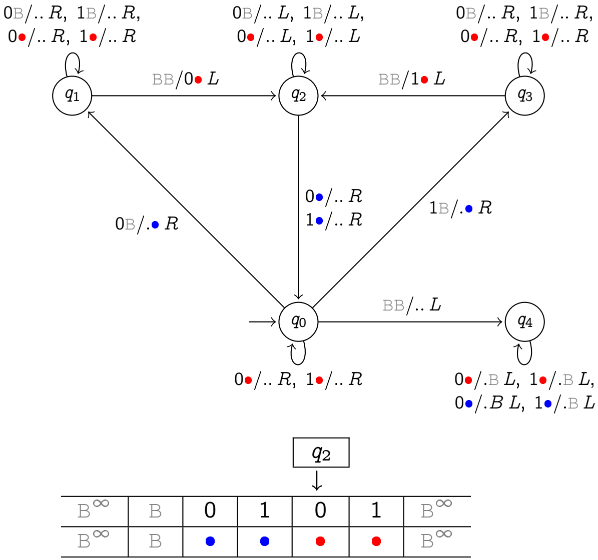
Example 2

A multitrack TM that duplicates the input string $w \in \{0,1\}^*$.



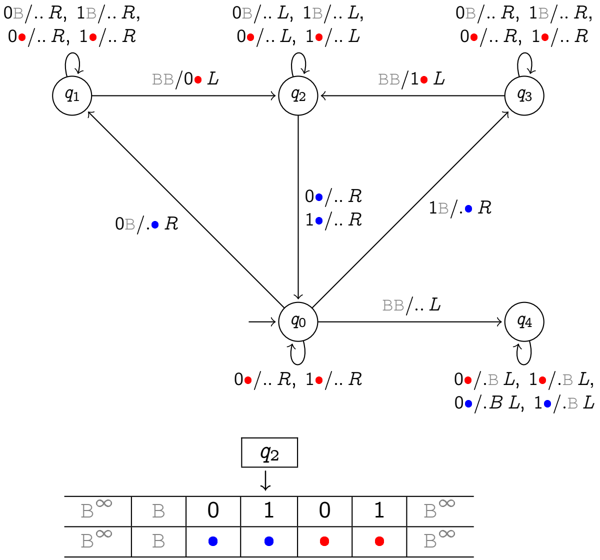
Example 2

A multitrack TM that duplicates the input string $w \in \{0, 1\}^*$.



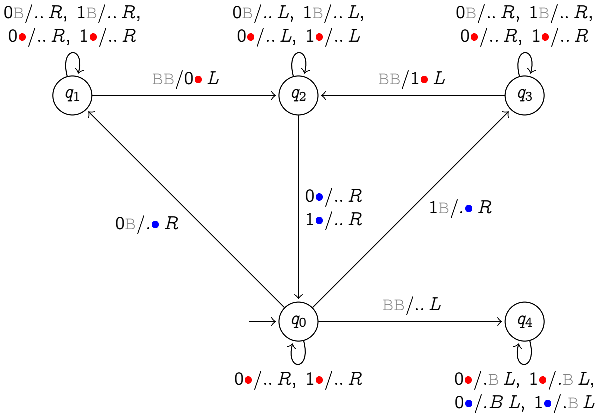
Example 2

A multitrack TM that duplicates the input string $w \in \{0, 1\}^*$.



Example 2

A multitrack TM that duplicates the input string $w \in \{0, 1\}^*$.



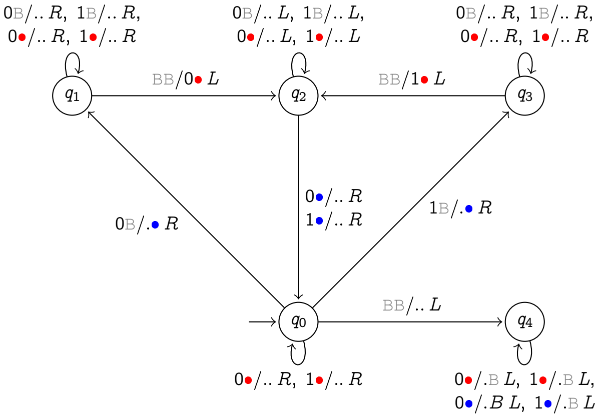
q_0

↓

B^∞	B	0	1	0	1	B^∞
B^∞	B	●	●	●	●	B^∞

Example 2

A multitrack TM that duplicates the input string $w \in \{0, 1\}^*$.

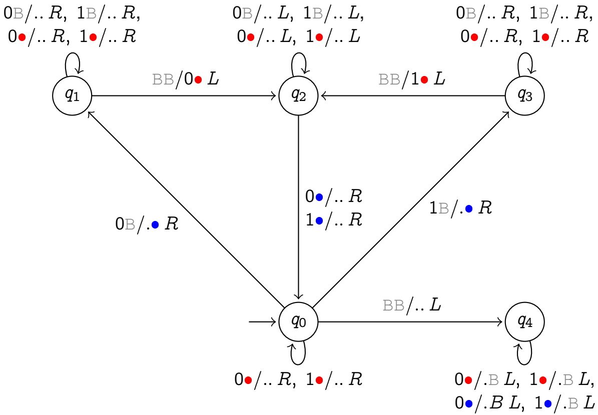


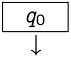
q_0

B^∞	B	0	1	0	1	B^∞
B^∞	B	•	•	•	•	B^∞

Example 2

A multitrack TM that duplicates the input string $w \in \{0, 1\}^*$.

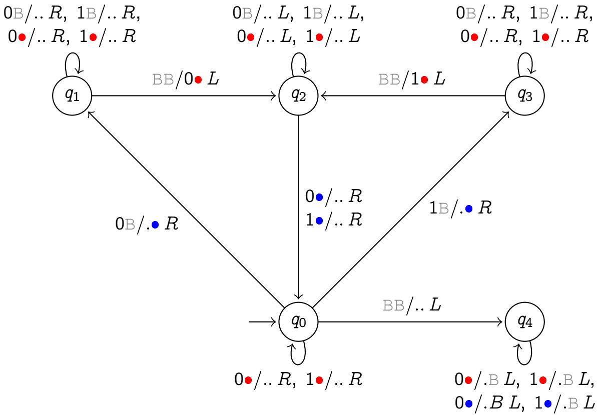


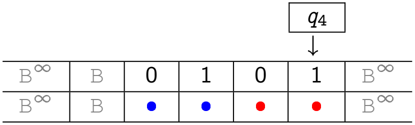


B^∞	B	0	1	0	1	B^∞
B^∞	B	●	●	●	●	B^∞

Example 2

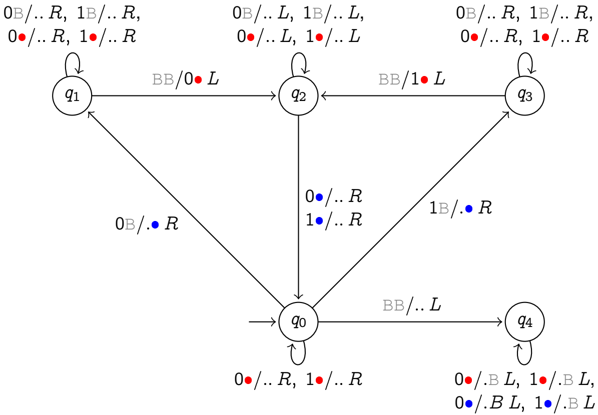
A multitrack TM that duplicates the input string $w \in \{0,1\}^*$.





Example 2

A multitrack TM that duplicates the input string $w \in \{0,1\}^*$.

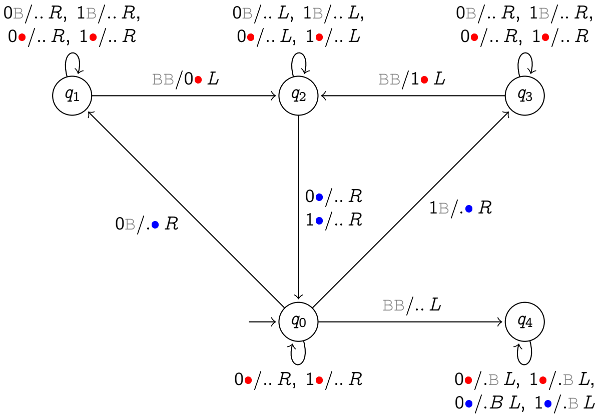


q_4

B^∞	B	0	1	0	1	B^∞
B^∞	B	●	●	●	B	B^∞

Example 2

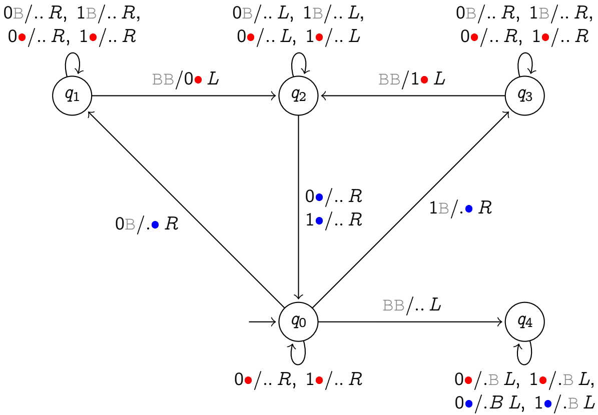
A multitrack TM that duplicates the input string $w \in \{0,1\}^*$.



B^∞	B	0	1	0	1	B^∞
B^∞	B	●	●	B	B	B^∞

Example 2

A multitrack TM that duplicates the input string $w \in \{0,1\}^*$.

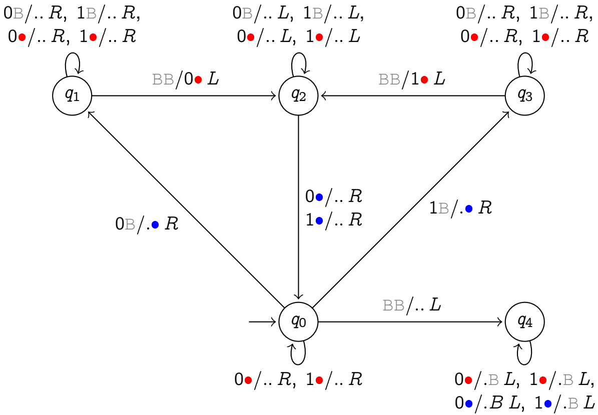


q_4

B^∞	B	0	1	0	1	B^∞
B^∞	B	•	B	B	B	B^∞

Example 2

A multitrack TM that duplicates the input string $w \in \{0,1\}^*$.

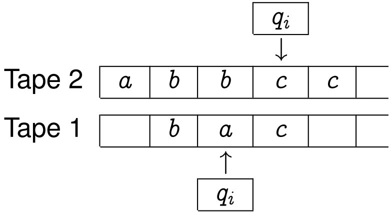


q_4

B^∞	B	0	1	0	1	B^∞
B^∞	B	B	B	B	B	B^∞

Multitape TMs

- A k-tape TM consists of k tapes and k independent tape heads
- The TM reads the tapes simultaneously, but has only one state
- A two tape machine:



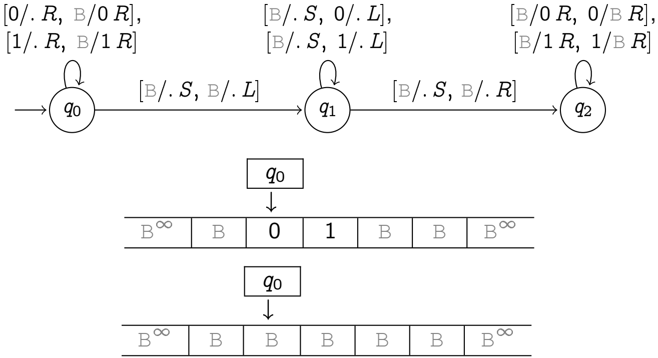
- A transition of a two-tape machine:

$$\delta(q_i, x_1, x_2) = [q_j;\ y_1, d_1;\ y_2, d_2]$$

- x_i and y_i are the old and new symbols on tape i;
- q_i and q_j are the old and new states;
- $d_i \in \{L, R, S\}$ is the direction of movement for tape head i, where S stands for “stationary” / “stand still”

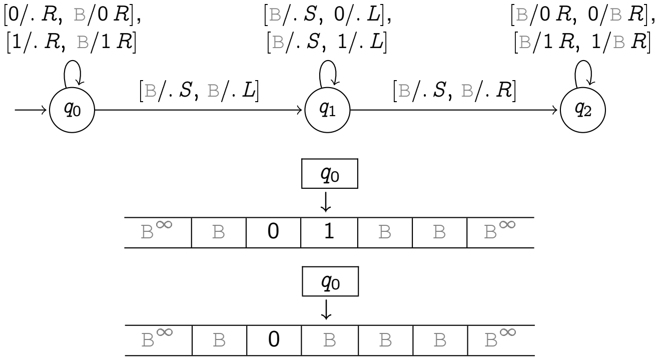
Example 2

A multitape TM that duplicates the input string $w \in \{0, 1\}^*$.



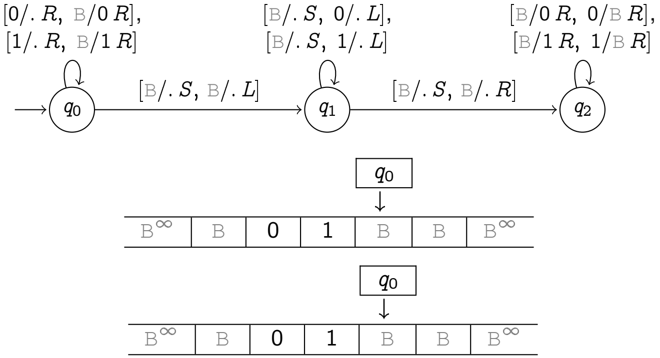
Example 2

A multitape TM that duplicates the input string $w \in \{0,1\}^*$.



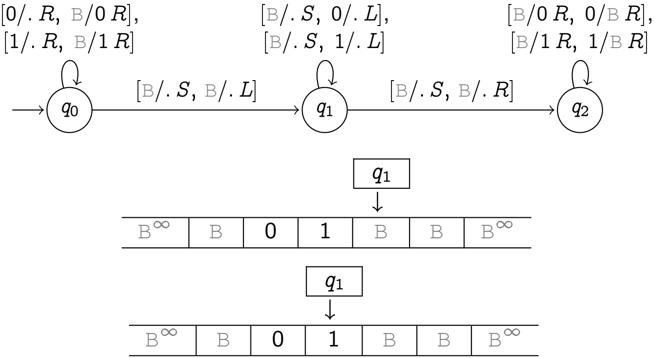
Example 2

A multitape TM that duplicates the input string $w \in \{0, 1\}^*$.



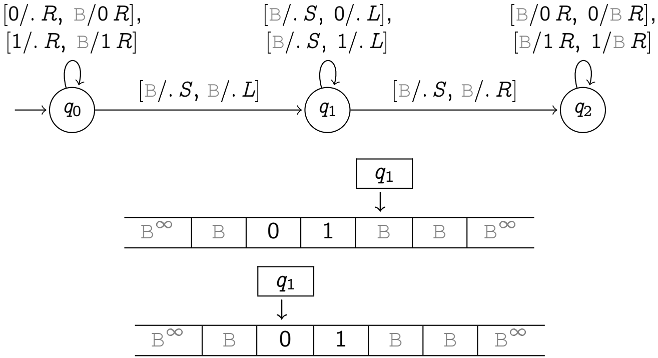
Example 2

A multitape TM that duplicates the input string $w \in \{0,1\}^*$.



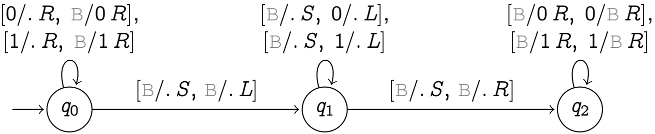
Example 2

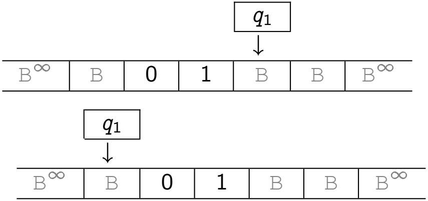
A multitape TM that duplicates the input string $w \in \{0,1\}^*$.



Example 2

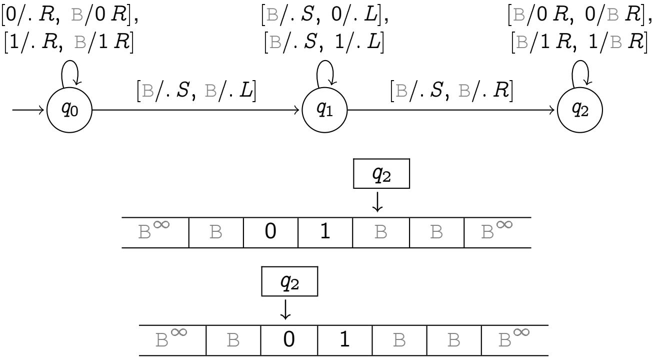
A multitape TM that duplicates the input string $w \in \{0, 1\}^*$.





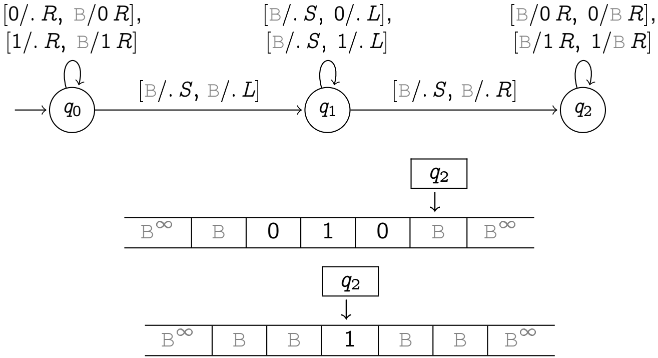
Example 2

A multitape TM that duplicates the input string $w \in \{0,1\}^*$.



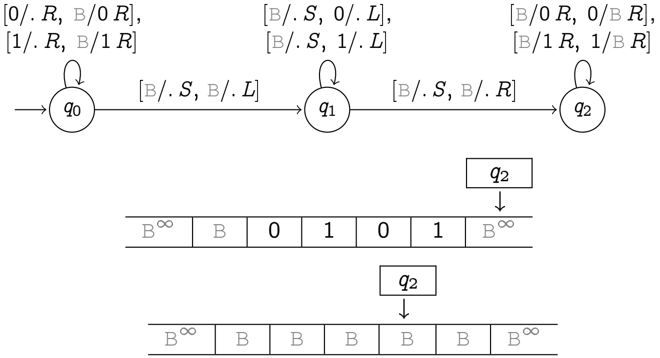
Example 2

A multitape TM that duplicates the input string $w \in \{0,1\}^*$.



Example 2

A multitape TM that duplicates the input string $w \in \{0,1\}^*$.



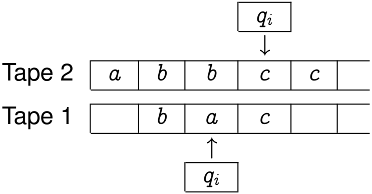
Outline

Simulating Multitape with Multitrack

- It is possible to simulate a **two-tape** machine using a **five-track** machine.

Simulating Multitape with Multitrack

- It is possible to simulate a **two-tape** machine using a **five-track** machine. Key idea:

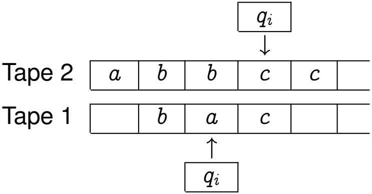


$\Rightarrow$

Track 5	#					
Track 4				X		
Track 3	a	b	b	c	c	
Track 2			X			
Track 1		b	a	c		

Simulating Multitape with Multitrack

- It is possible to simulate a **two-tape** machine using a **five-track** machine. Key idea:

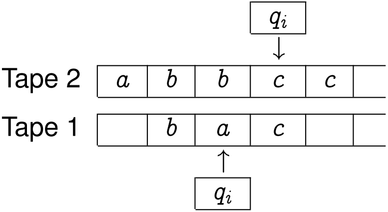


$\Rightarrow$

Track 5	#					
Track 4				X		
Track 3	a	b	b	c	c	
Track 2			X			
Track 1		b	a	c		

- In general, a language accepted by a k-tape machine is accepted by a $2k+1$-track machine

Simulating Multitape with Multitrack



$\Rightarrow$

Track 5	#					
Track 4				X		
Track 3	a	b	b	c	c	
Track 2			X			
Track 1		b	a	c		

Consider a transition $\delta(q_i, x_1, x_2) = [q_j; y_1, d_1; y_2, d_2]$.
Its simulation in the multitrack machine involves:

1. Finding the x_1 and x_2 in T1 and T3, using the Xs in T2 and T4.
2. With x_1 and x_2, the y_1 and y_2 to be printed and the directions d_1 and d_2 can be determined.
3. Printing y_1 and y_2 in T1 and T3, and moving the Xs in T2 and T4, according to d_1 and d_2.

Outline

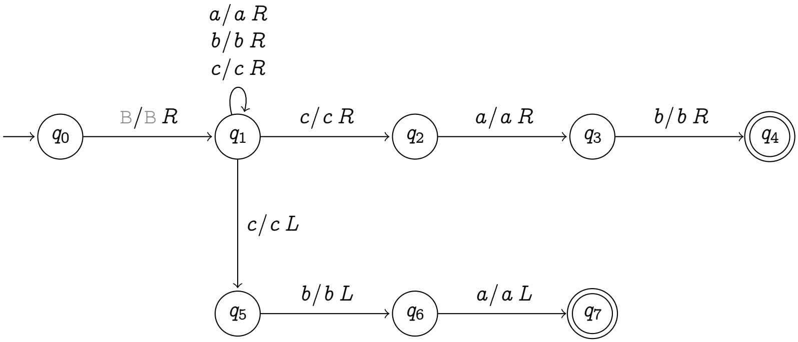
Non-Deterministic TMs (NTMs)

- Just as the other machines, TMs can be non-deterministic
- This means that the transition function is defined as

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

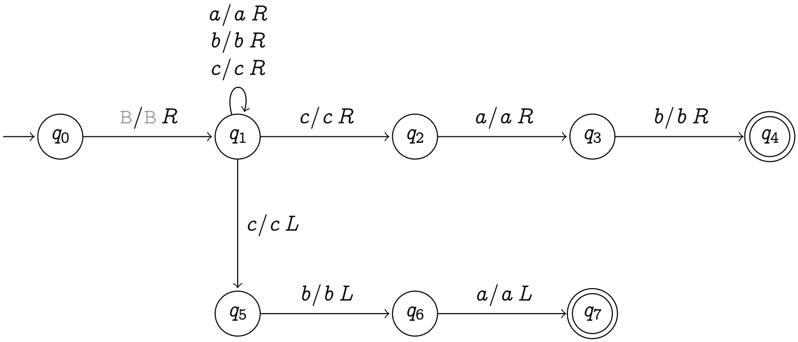
- When more than one transition is possible, the computation chooses arbitrarily one of them
- Given an input string, an NTM may produce several computations

Example 3: An NTM



Example 3: An NTM

A TM that accepts strings whose last occurrence of c is preceded or followed by ab:



Non-Deterministic TMs (NTMs)

- Just as other machines we have seen, TMs can be non-deterministic
- This means that the transition function is defined as

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

- When more than one transition is possible, the computation chooses arbitrarily one of them
- Given an input string, an NTM may produce several computations
- The reader describes a breadth-first procedure to represent NTM computations using a (deterministic) two-tape TM
- Non-determinism + multitracks + multitape?
  Combinations are possible and handled as expected

Turing machines

The following are equivalent:

- Simple TMs
- Two-way TMs
- Multitrack TMs
- Multitape TMs
- Non-deterministic TMs (NTMs)
- Non-deterministic, multitrack TMs
- Non-deterministic, multitape TMs

Always Terminating NTMs

Given an NTM with a set of accepting states, there are three kinds of computations:

1. Terminating and accepting
2. Terminating and non-accepting
3. Non terminating (infinite!)

An input is accepted iff it has at least one accepting computation (it may also have non-accepting and non-terminating computations)

A TM is **always terminating** if for every input string every computation terminates

From Last Lecture

A TM is **always terminating** if it terminates for every input.

Let L be a language.

- L is **semi-decidable** (or **recursively enumerable, RE**) if there exists a TM M such that $L = L(M)$.
- L is **decidable** (or **recursive**) if there is an always terminating TM that accepts L by termination in an accepting state.
- If L is decidable, then it is also semi-decidable. The converse doesn't hold!

Complexity classes

A (non)deterministic TM M has **time complexity** $T(n)$ if M is guaranteed to terminate in at most $T(n)$ steps for every input string w of length n (regardless of whether w is accepted).

Let L be a language and let $T(n)$ be a **polynomial function**:

- L belongs to the class $\mathcal{P}$ if there is a deterministic TM M with $L = L(M)$ and with time complexity $T(n)$.
- L belongs to the class $\mathcal{NP}$ if there is an NTM M with $L = L(M)$ and with time complexity $T(n)$.
- Because every deterministic TM can be regarded as an NTM with the same time complexity, we have $\mathcal{P} \subseteq \mathcal{NP}$.
- Conjecture: $\mathcal{P} \neq \mathcal{NP}$.

Computers, DTMs, and NTMs

- Everything that can be computed with a DTM, can be computed with an ordinary computer at least with the same efficiency, up-to memory extensions.
- Everything that can be computed with such an extendable computer, say in n steps, can be computed on a deterministic Turing machine in $T(n)$ steps for some polynomial $T(n)$.
- Ordinary computers are closer to the DTM than to the NTM.

Outline

Closure Properties

We know:

$$L \text{ is decidable} \Rightarrow L \text{ is semi-decidable} \qquad (*)$$

Furthermore:

$$
\begin{array}{lrcl}
1. & L \text{ is decidable} & \Rightarrow & \overline{L} \text{ is decidable} \\
2. & L \text{ and } \overline{L} \text{ are semi-decidable} & \Leftrightarrow & L \text{ is decidable} \\
3. & L \text{ is semi-decidable} & \Leftrightarrow & L^* \text{ is semi-decidable} \\
4. & L_1 \text{ and } L_2 \text{ are semi-decidable} & \Rightarrow & L_1L_2,\ L_1 \cup L_2, \text{ and} \\
 & & & L_1 \cap L_2 \text{ are semi-decidable}
\end{array}
$$

Key ideas:

1. Use the complement of the set of accepting states.
2. $\Rightarrow$) Given M_1 and M_2 for L and $\overline{L}$, devise a two-tape TM that runs M_1 and M_2 in lockstep. $\Leftarrow$) Immediate from (1) and $(*)$
3. Exercise 5.13
4. These properties proven by building appropriate TMs.

Taking Stock

This lecture:

- Variants of Turing machines
- DTMs, NTMs, and their complexity classes
- Closure properties

Next Lecture: Thursday, May 25th

- Decision problems, in particular the halting problem
- Problems, languages, and (semi-)decidability
- Universal Turing machines