CS 410/610, MTH 410/610
Theoretical Foundations of Computing

Fall Quarter 2010

Slides 2

Pascal Hitzler
Kno.e.sis Center
Wright State University, Dayton, OH
http://www.knoesis.org/pascal/

Overview: Turing Machines

Chapter 8 of [Sudkamp 2006].

1. **The Standard Turing Machine**
2. **Turing Machines as Language Acceptors**
3. **Multitrack Machines**
4. **Two-Way Tape Machines**
5. **Multitape Machines**
6. **Nondeterministic Turing Machines**
7. **Language Enumeration by Turing Machines**

The standard Turing Machine

A Turing machine has

- **An infinite tape with cells numbered 0,1,2,3,…**
- **A read-write head, which is always positioned at exactly one cell**
- **A set of internal "states" – depending on the current state the machine behaves differently**
- **A set of instructions which tell the machine at each step**
  - **what to write on the current tape position**
  - **where to move the tape head next (left or right)**
  - **to which internal state to switch before the next step**

In more detail, an instruction is performed by

1. **Reading the current tape cell**
2. **Executing an instruction, depending on the cell read and the current internal state**

Standard Turing Machine

Formally:

Definition 1.1

A *Turing Machine* (TM) is a quintuple (Q, Σ, Γ, δ, q_0) with

Q …………………………….. finite set of *states*
Γ ……………………………... finite set called *tape alphabet*
contains special symbol B (blank)
$\Sigma \sqsubseteq \Gamma \backslash \{B\}$ …………………... the input alphabet
$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L,R\}$ … a partial function
called *transition function*
$q_0 \in Q$ ……………………….. the *start state*

Example 1.2

$Q = \{q_0, q_1, q_2\}$

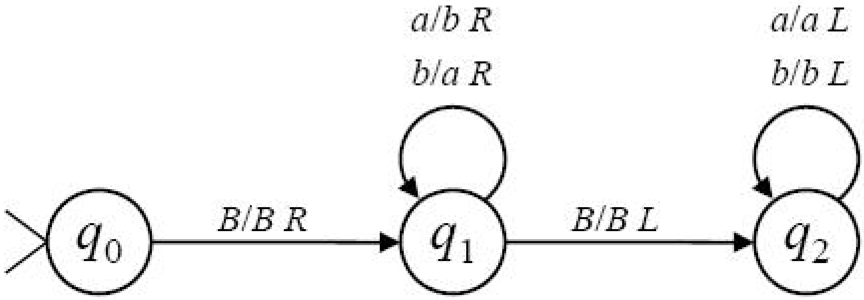
$\Gamma = \{a, b, B\}$

$\Sigma = \{a, b\}$

δ: see table to the right

δ	B	a	b
q_0	q_1,B,R		
q_1	q_2,B,L	q_1,b,R	q_1,a,R
q_2		q_2,a,L	q_2,b,L

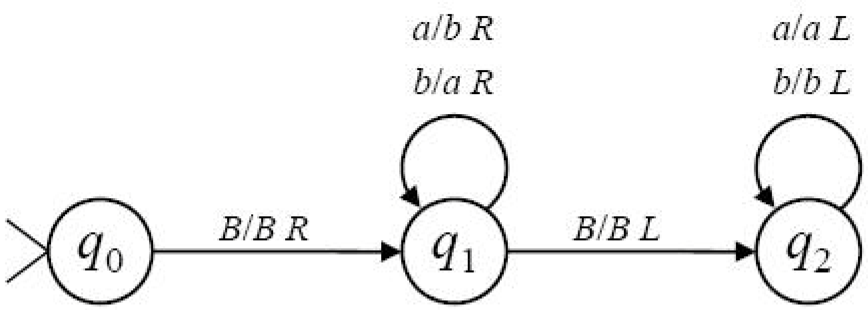
A compact representation of the same Turing Machine:



this is a so-called *state diagram*

Example Run

δ	B	a	b
q_0	q_1,B,R		
q_1	q_2,B,L	q_1,b,R	q_1,a,R
q_2		q_2,a,L	q_2,b,L

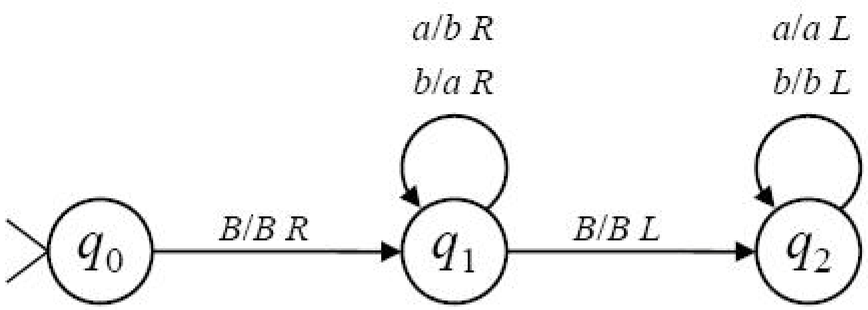


q_0BababB [indicates tape from position 0
everything further to the right is B
the state is written left of the cell with the tape head]

$\vdash$ B$q_1$ababB [$\vdash$ indicates a single transition;
tape head moves one to the right,
state becomes q_1,
B is written]

Example Run

δ	B	a	b
q_0	q_1,B,R		
q_1	q_2,B,L	q_1,b,R	q_1,a,R
q_2		q_2,a,L	q_2,b,L



q_0BababB
⊢ B$q_1$ababB
⊢ Bb$q_1$babB
⊢ Bba$q_1$abB
⊢ Bbabq_1bB
⊢ Bbabaq_1B
⊢ Bbabq_2aB
⊢ Bba$q_2$baB
⊢ Bb$q_2$abaB
⊢ B$q_2$babaB
⊢ q_2BbabaB

The TM changes all a's to b's and all b's to a's and then returns the tape head to the starting position.

We always assume the following

- The input is a single finite string, written in cells numbered 1,2,3,…
- All other cells are initially B.
- The output is also a single finite string, written in cells numbered 1,2,3,…
- The tape head always starts in position 0.
- The start state is always q_0.

- A computation halts if no action is defined for a current symbol/state pair.
- A computation *terminates abnormally* if it moves left of tape position 0.

Exercise 2 [no hand-in]

Define a Turing Machine which never halts.

Example 1.3

A TM for input alphabet {a,b} which copies the input string:

Starting tape: BuB (u is a string of a's and b's)
End tape: BuBuB

How to do this?

Example 1.3

1. Find first symbol

   Move 1 to the right

2. Memorize symbol

   Use 2 different states for 2 symbols

3. Move to where we want to copy the symbol

   Move to right, pass one blank, find next blank

4. Write the symbol

   ☺

5. Find the next symbol. Terminate if no such symbol.

   Tricky. We have to set a marker.
   In fact, we use 2 markers for the 2 input symbols.
   Before termination, we need to convert them back!

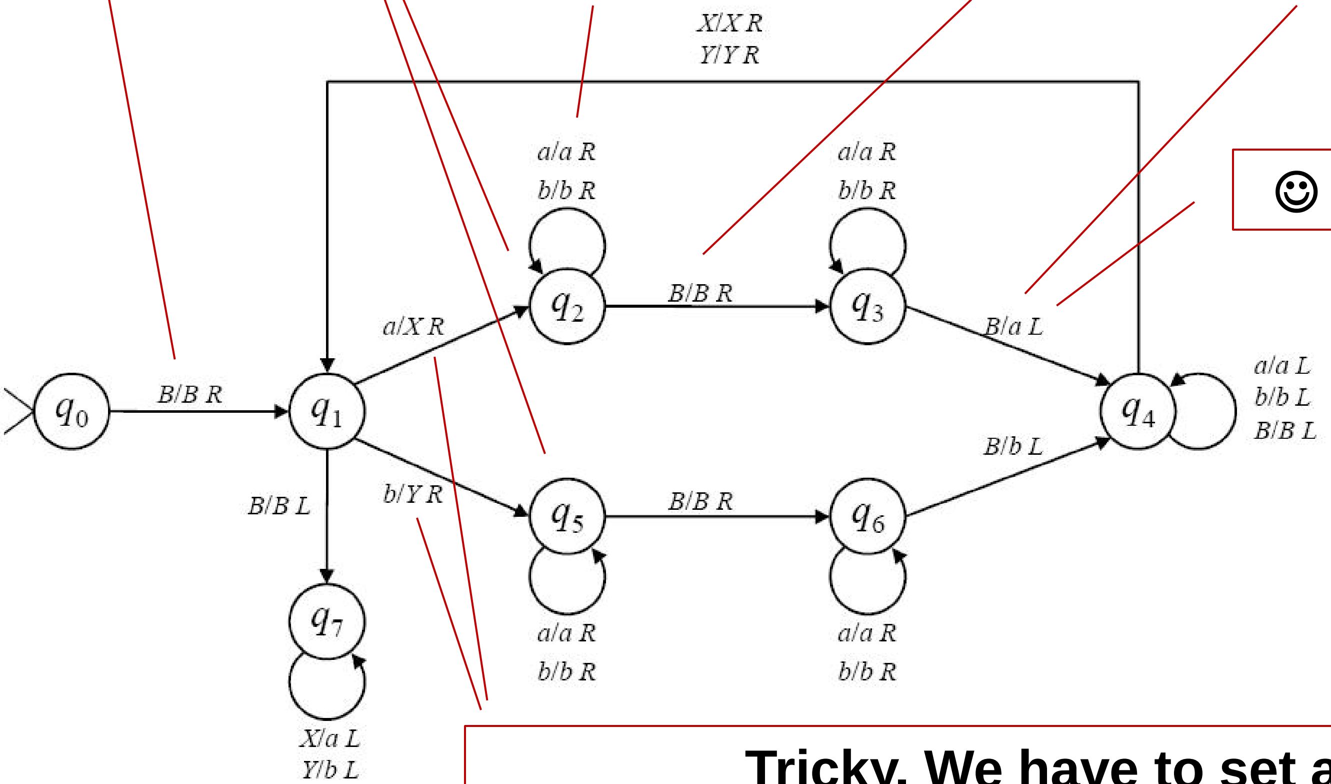
6. Return to step 2.

Example 1.3

Move 1 to the right

Use 2 different states for 2 symbols

Move to right, pass one blank, find next blank



☺

Tricky. We have to set a marker.
In fact, we use 2 markers for the 2 input symbols.
Before termination, we need to convert them back!

Exercise 3 (no hand-in)

Define a standard Turing Machine which moves an input string consisting of a's and b's one position to the right.

Exercise 4 (hand-in)

Give, as a state diagram, a standard Turing Machine M which increments a number in binary representation by one (i.e., add 1 to the number).

Assume that the input is given with lowest bit first, e.g., the binary representation for ”6", which is “110", is represented on the tape as “B011B".

Also, sketch in words the strategy of your TM.

Exercise 5 (hand-in)

Define a standard Turing Machine which, on any input consisting of a string of 2n a's, deletes the first n of the a's (n is any non-negative integer).

For example, the input BaaaaaaB shall become BBBBaaaB.

Also, sketch in words the strategy of your TM.

Overview: Turing Machines

Chapter 8 of [Sudkamp 2006].

1. **The Standard Turing Machine**
2. **Turing Machines as Language Acceptors**
3. **Multitrack Machines**
4. **Two-Way Tape Machines**
5. **Multitape Machines**
6. **Nondeterministic Turing Machines**
7. **Language Enumeration by Turing Machines**

Strings and Languages

- **An *alphabet* Σ is a finite set of symbols.**

- **The *set of strings* Σ^* over an alphabet Σ is recursively defined as follows.**
  - $\lambda \in \Sigma^*$
  - **If $w \in \Sigma^*$ and $a \in \Sigma$, then $wa \in \Sigma^*$**
  - **Nothing else is in Σ^***

  λ is called the *empty string*.

- **For $w \in \Sigma^*$, length(w) is the number of symbols in w (called the *length* of w).**

- **A *language* over an alphabet Σ is a subset of Σ^*.**

Exercise 6 [no hand-in]

Give a language over the alphabet {a,b} containing exactly 6 strings.

Regular expressions

Σ an alphabet. The regular expressions over Σ are defined recursively as follows.

- $\emptyset$,λ, and a (for each a$\in\Sigma$) are regular expressions over Σ.
- If u and v are regular expressions over Σ, then

$(u \cup v)$	(choice)
(uv)	(concatenation)
(u^*)	(finite repetition [Kleene star])

  are regular expressions over Σ.
- Nothing else is a regular expression over Σ.

We write u^+ for uu^*.

Example 1.4

- **$a(b)^*a$ – examples for strings in this language:**
  aa, aba, abba, abbba, …

Kleene star: allow finite number of repetitions (including zero)

- **$a^*(b \cup c)d^*$ – examples for strings in this language:**
  b, c, ab, aac, abd, aaabdddd, …

$\cup$: use either of the two expressions to the left of right

- **$(a \cup b)^*$ – examples for strings in this language:**
  λ, a, aa, ab, bb, ababba, …
  The language is in fact $\{a,b\}^*$.
- **$ba(a \cup b)^*ab$ – examples for strings in this language:**
  baab, abaabbabaababbaab, …

Background reading

In this lecture, we assume that you are familiar with languages and regular expressions as just defined.

If you had no previous exposure to these, or if you feel that you need to further refresh your memory, it is required that you thoroughly read pages 41 to 54 of [Sudkamp 2006].

Exercise 7 [no hand-in]

Give 5 distinct strings in the language

$$(a^* \cup b)^*(c \cup (d \cup e)^*)$$

Language accepting TMs

For this, we augment Turing Machines with final states. Such TMs are sextuples $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where $F \subseteq Q$ is the *set of final states*.

– In diagrams, final states are indicated with double or boldface circles

- Given a TM M, a string $u \in \Sigma^*$ is *accepted by final state* if the computation of M with input u halts in a final state.

  – Note that u isn't accepted if M terminates abnormally (even if it does so in a final state).

- The *language of M*, denoted L(M), is the set of all strings accepted by M.

Language accepting TMs

- **A language A is called *recursively enumerable* if there is a TM M with L(M)=A.**

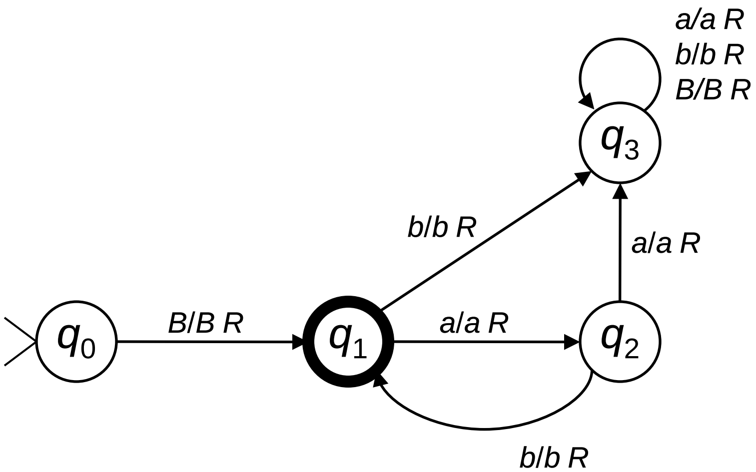
- **A language A over an alphabet Σ is called *recursive* if there is a TM M with L(M)=A and if, furthermore, M halts on every input $w \in \Sigma^*$.**

Example 1.5

- Consider (ab)* as language over {a,b}.
- Is this language recursive? Recursively enumerable?
- Idea for constructing TMs for this?

Example 1.5

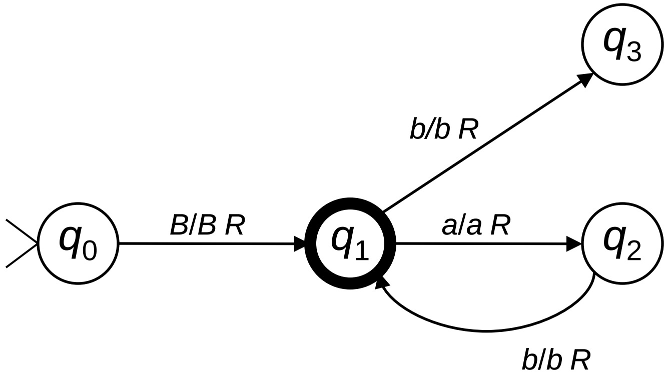
- (ab)* as language over {a,b}.



- Hence, language is recursively enumerable.
  Is it also recursive?

Example 1.5

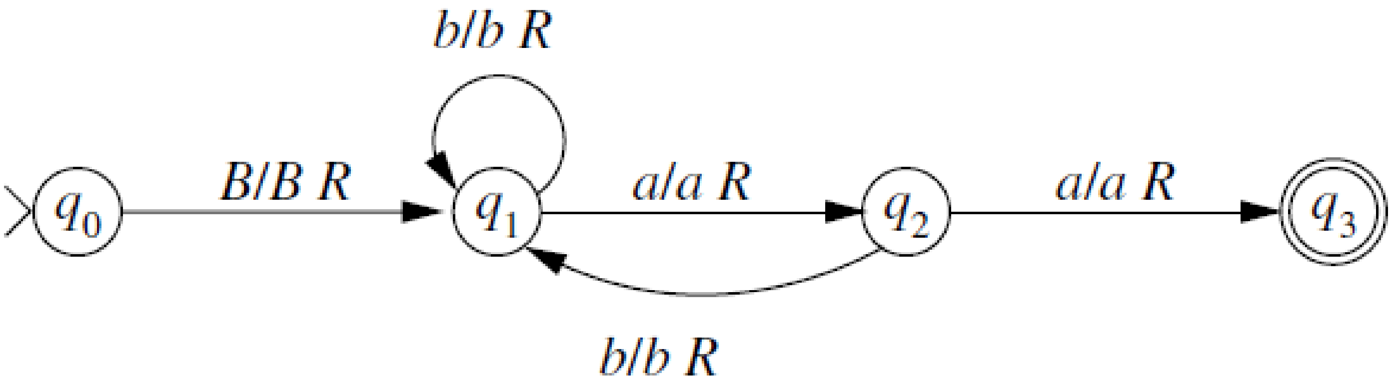
- (ab)* as language over {a,b}.



- Yes, the language is recursive as well!

Example 1.6

- **TM accepting which language?**



Language: (b*(ab)*)*aa

Example 1.7

- **TM accepting the language $\{a^i b^i c^i \mid i \geq 0\}$.**

  Note: a^5 = aaaaa etc.

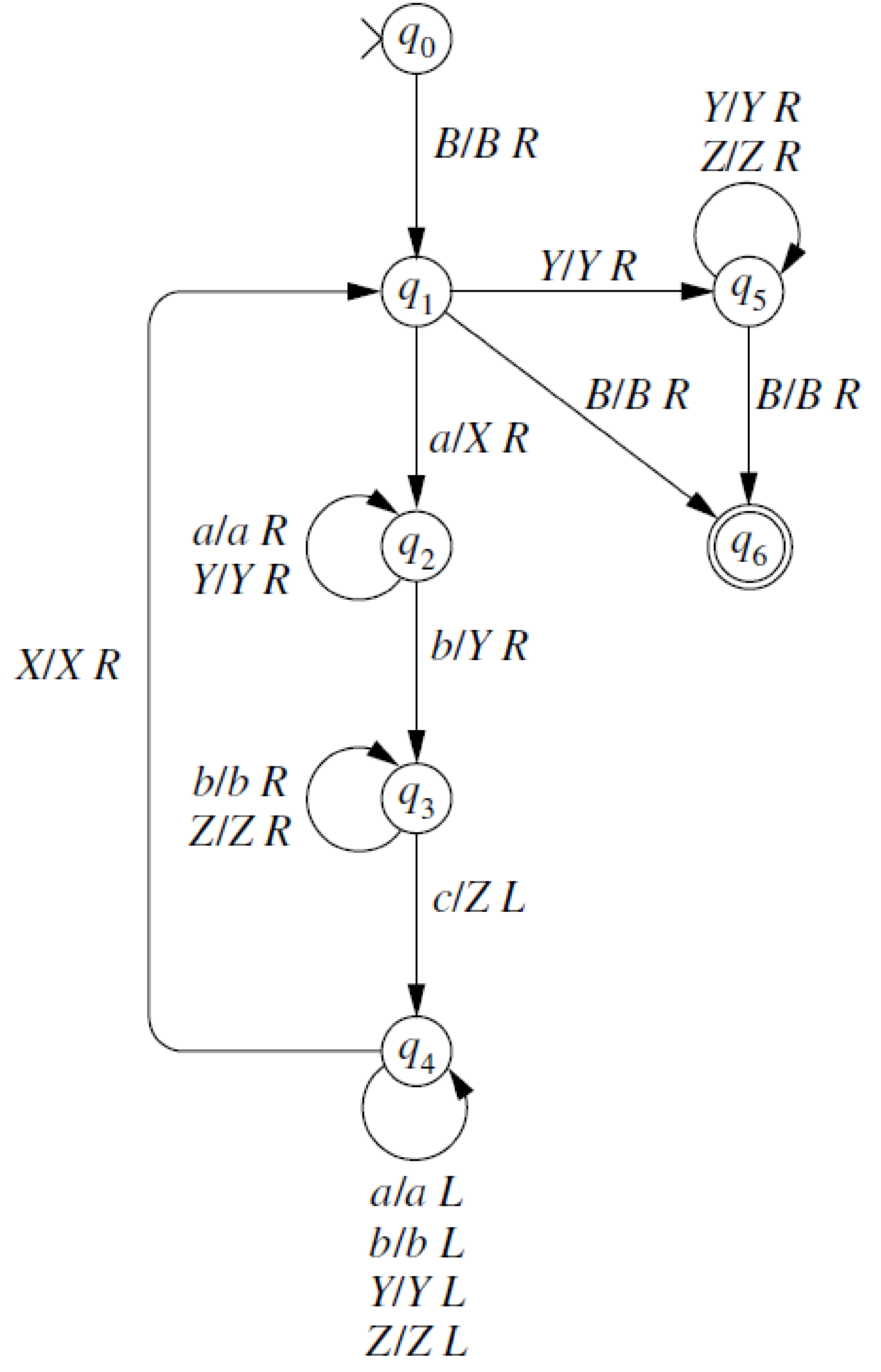
- **Idea how to do this?**

- **Use markers.**
  - **Make repeated passes through the whole string.**
  - **At each pass, mark one of each a, b, c as read. If you don't find a b and a c after finding an a, don't accept.**
  - **When all a's are gone, check if there are any b's or c's left.**

Example 1.7

- **TM accepting the language $\{a^ib^ic^i \mid i \geq 0\}$.**

- **The language is recursive.**

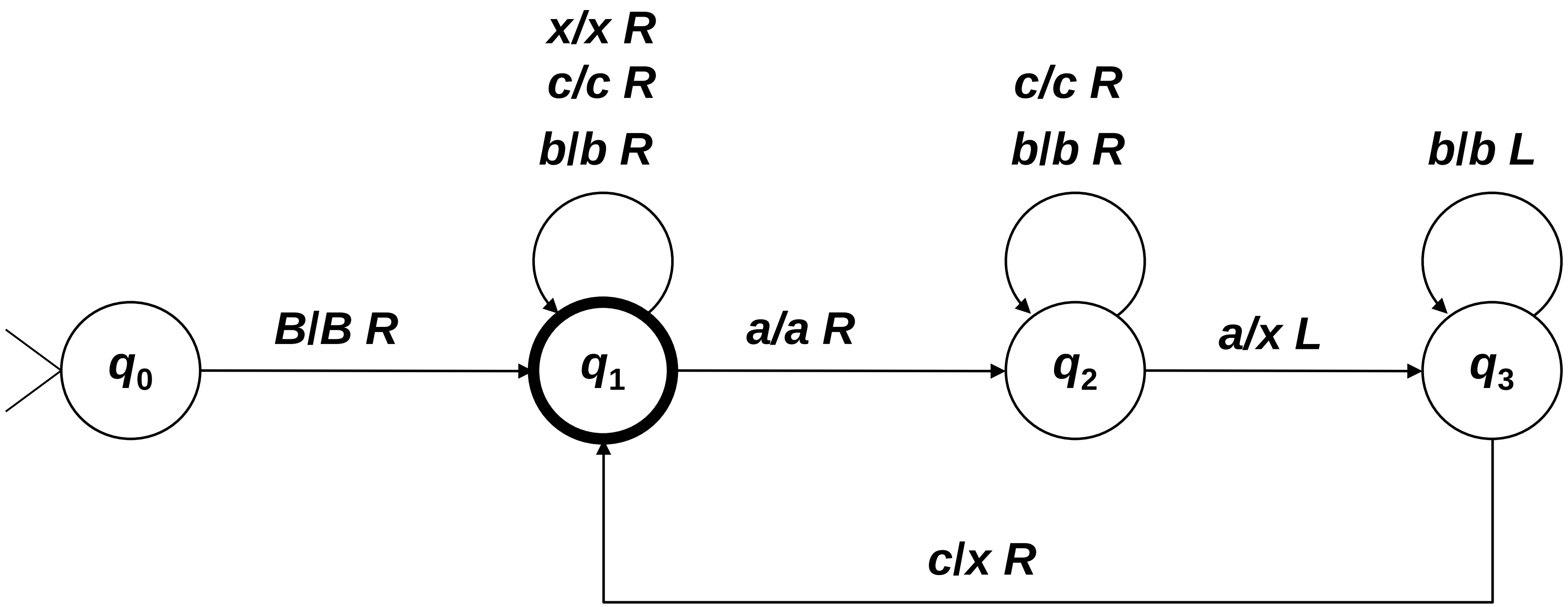


Exercise 8 [hand-in]

Show that the language $(a \cup b)^*$ is recursive.

Exercise 9 [no hand-in]

- **What is the language accepted by the following TM?**



Exercise 10 [hand-in]

Show that there are languages which are not recursively enumerable.

Hint: Use diagonalization. It is possible to adjust the proof given in the introductory session, that not all sets of non-negative integers can be computed. You do not need to spell out all details, but the argument must be convincing.

Acceptance by halting

- **Let $M=(Q,\Sigma,\Gamma,\delta,q_0)$ be a TM. A string $u \in \Sigma^*$ is *accepted by halting* if the computation of M with input u halts (normally).**

Does this make a difference to acceptance by final state?

Actually, no.

Acceptance by halting

Theorem 1.8

The following statements are equivalent.

1. **The language L is accepted by a Turing machine that accepts by final state.**
2. **The language L is accepted by a Turing machine that accepts by halting.**

How to prove this?

Acceptance by halting

Acceptance by halting = acceptance by final state.

How to prove this?

Two steps:

1. ***Assume you have a TM which accepts L by halting. Then construct a TM which accepts L by final state.***

2. ***Vice-versa.***

Acceptance by halting

1. ***Assume you have a TM M which accepts L by halting. Then construct a TM M' which accepts L by final state.***

Easy:

If $M = (Q, \Sigma, \Gamma, \delta, q_0)$

then use $M' = (Q, \Sigma, \Gamma, \delta, q_0, Q)$.

I.e. every state is final.

[Equivalence proofs often have an easy and a difficult direction. This was the easy direction.]

Acceptance by halting

2. ***Assume you have a TM M which accepts L by final state. Then construct a TM M' which accepts L by halting.***

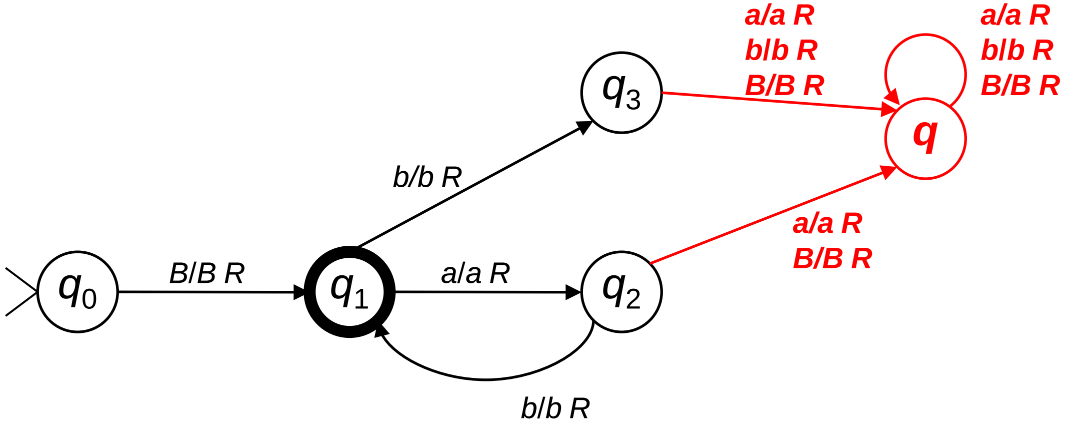
Idea:

- **Add one additional state q.**
- **Add transitions such that, when a computation is in q, then it will stay in q, and loop there indefinitely.**
- **Make sure that M' can never halt in any of the non-final states of M, by adding additional transitions from these states to q.**

Acceptance by halting

2. *Assume you have a TM M which accepts L by final state. Then construct a TM M' which accepts L by halting.*

- **Add one additional state q.**
- **Add transitions such that, when a computation is in q, then it will stay in q, and loop there indefinitely.**
- **Make sure that M' can never halt in any of the non-finite states of M, by adding additional transitions from these states to q.**



Exercise 11 [no hand-in]

Use the construction from the proof of Theorem 1.8 to convert your TM from Exercise 8 into one that accepts by halting.

Acceptance by halting – again

2. ***Assume you have a TM M which accepts L by final state. Then construct a TM M' which accepts L by halting.***

Formal proof:

If $M=(Q,\Sigma,\Gamma,\delta,q_0,F)$, then $M'=(Q\cup\{q\},\Sigma,\Gamma,\delta',q_0)$.

For each $x\in\Gamma$, set $\delta'(q,x)=[q,x,R]$.

Set $\delta'(q_i,x)=\delta(q_i,x)$ if the latter is defined.

For each $q_i\in Q\setminus F$, if $\delta(q_i,x)$ is undefined, set $\delta'(q_i,x)=[q,x,R]$.

Now: If M accepts w, then M' accepts w (computation is identical).

If M does not accept w, then one of the following holds:

(1) M terminated abnormally: then so will M'
(2) M did not terminate: then so will M'
(3) M terminated in a non-final state: then M' will loop in q.

Hence M' does not accept w. This completes the proof.

Overview: Turing Machines

Chapter 8 of [Sudkamp 2006].

1. **The Standard Turing Machine**
2. **Turing Machines as Language Acceptors**
3. **Multitrack Machines**
4. **Two-Way Tape Machines**
5. **Multitape Machines**
6. **Nondeterministic Turing Machines**
7. **Language Enumeration by Turing Machines**

Driving question

- **Are TMs *maximal* computational models?**

- **Can we add further features to TMs such that**
  - **these features are feasible for a model of computation and**
  - **the resulting enhanced TMs can compute things which the standard TMs cannot compute?**

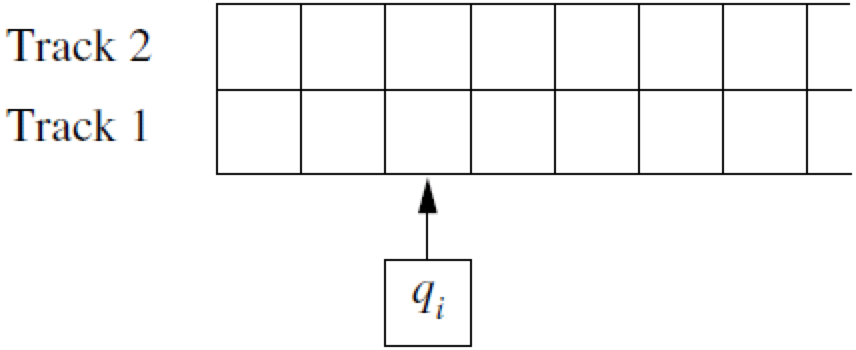
- **It turns out the answer that TMs seem to be maximal, i.e. no known enhancement adds real computational power.**

- **This is an *observation*, there is no formal proof for this. However, a century of research gives rather powerful evidence that TMs are really maximal computational models.**

Multi-track

- **Enhancement:**
  The tape has n tracks.
  Reading is done from all n tracks at the same time.
  Writing is done to all n tracks at the same time.
- **Tape position represented by tuple $[x_1,x_2,\ldots,x_n]$.**
- **A transition is written $\delta(q_i,[x_1,\ldots,x_n]) = [q_j,[y_1,\ldots,y_n],d]$, where $d\in\{L,R\}$.**
- **Input is placed in track 1 in the standard position, rest is blank.**
- **Acceptance is by final state.**

2-track tape



Multi-tracks don't add anything new

Theorem 1.9

A language L is accepted by a 2-track TM if, and only if, it is accepted by a standard TM.

Proof idea?

Proof: Easy direction

If L is accepted by a standard TM, then a 2-track TM accepting L is obtained by adding a second track, which is ignored.

Proof: “Difficult” direction

Let M = $(Q,\Sigma,\Gamma,\delta,q_0,F)$ be a 2-track TM.

The one-track TM M’ has

- Tape alphabet: ordered pairs of tape elements of M.
- Input: ordered pairs with second component blank.
  An input symbol a becomes [a,B] for M’.

- Formally, M’ = $(Q,\Sigma\times\{B\},\Gamma\times\Gamma,\delta',q_0,F)$
  where
  $$\delta'(q_i,[x,y]) = \delta(q_i,[x,y]).$$

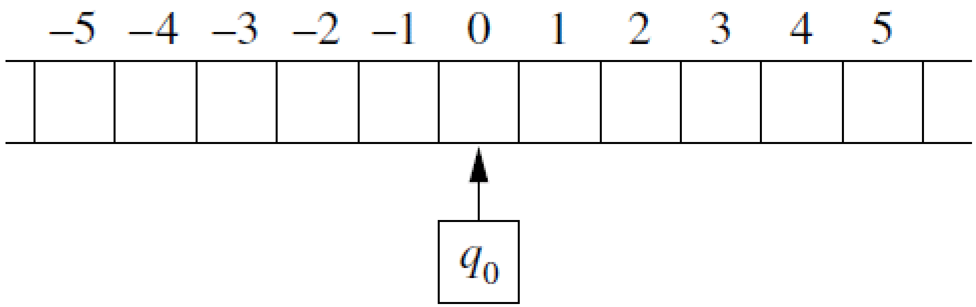
Overview: Turing Machines

Chapter 8 of [Sudkamp 2006].

1. **The Standard Turing Machine**
2. **Turing Machines as Language Acceptors**
3. **Multitrack Machines**
4. **Two-Way Tape Machines**
5. **Multitape Machines**
6. **Nondeterministic Turing Machines**
7. **Language Enumeration by Turing Machines**

2-way tape TMs

- **Enhancement:**
  Tape extending infinitely in *both* directions.



- **Input is anywhere on the tape, rest blank**
- **Initially, tape head on blank to the immediate left of input.**

2-way tape TMs

Theorem 1.10

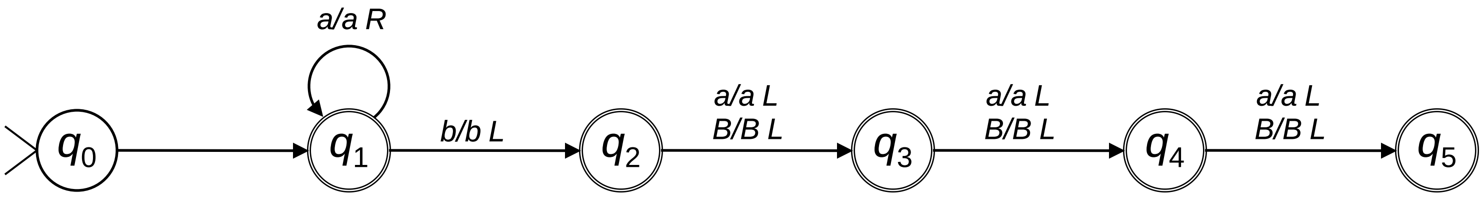
A language L is accepted by a TM with a 2-way tape if, and only if, it is accepted by a standard TM.

Proof idea – easy direction

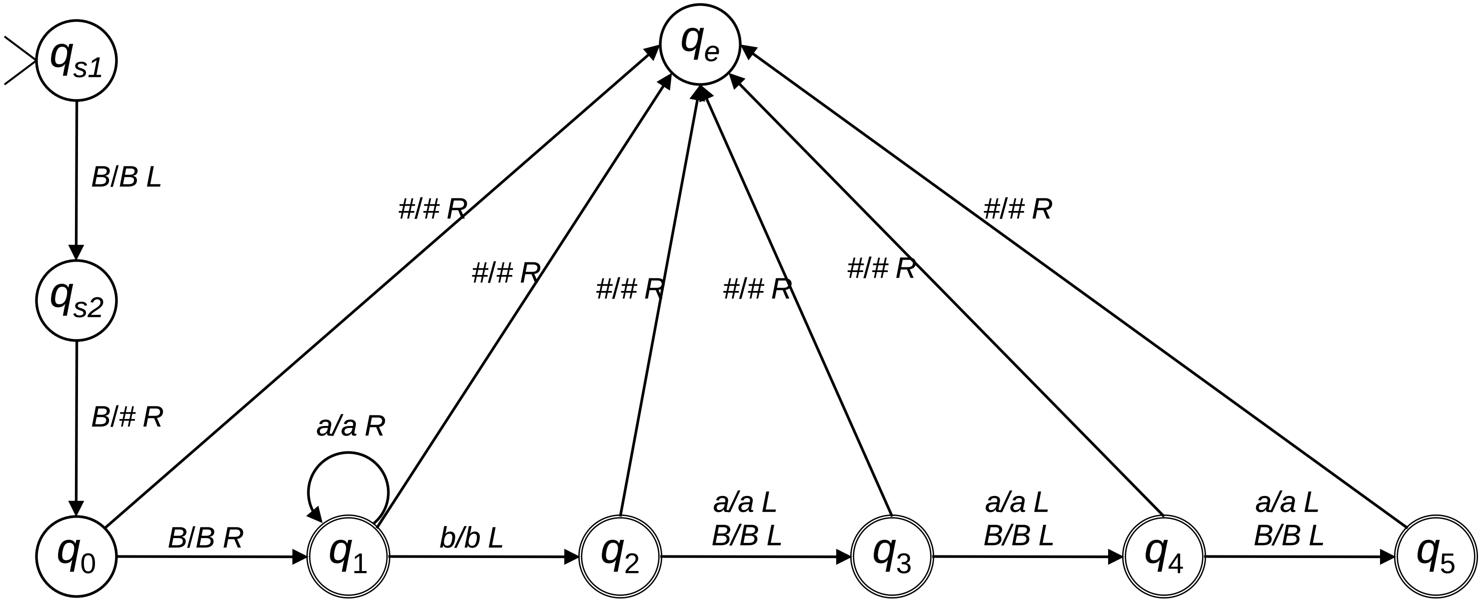
Simulate standard TM M by 2-way TM M'.

- **Add a *new* tape symbol #, representing the left boundary.**
- **M' starts by writing # to the immediate left of the initial tape position.**
- **Rest of computation is identical except for abnormal termination:**
  When M attempts to move left of the tape boundary,
  M' reads # and enters a non-accepting state that terminates the computation.

Easy direction – example



First *b* preceded by at least three *a*'s.



Exercise 12 [no hand-in]

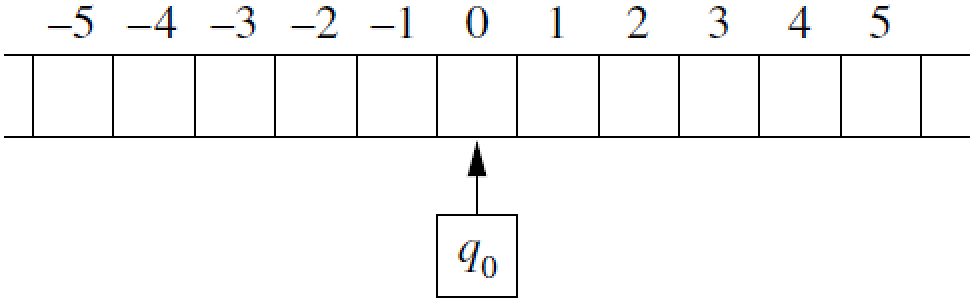
Convert the TM from Exercise 9 into a 2-way TM in the way just shown.

Proof idea – difficult direction

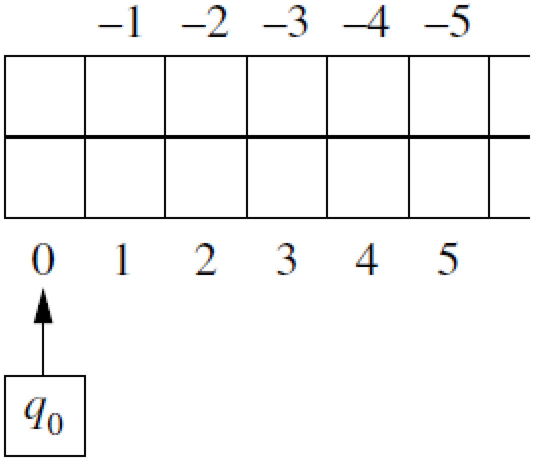
Simulate 2-way TM M by standard TM M''.

In fact, we construct a 2-track TM M' (which suffices).

The 2-way TM



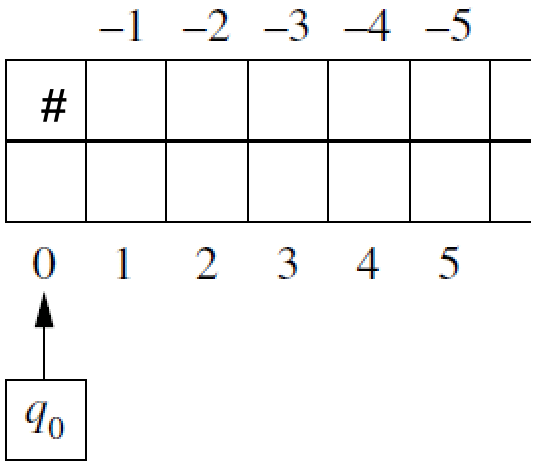
becomes



Proof – difficult direction

Standard TM M = (Q,Σ,Γ,δ,q_0,F)

What is M' = (Q',Σ',Γ ',δ',q_0',F')?

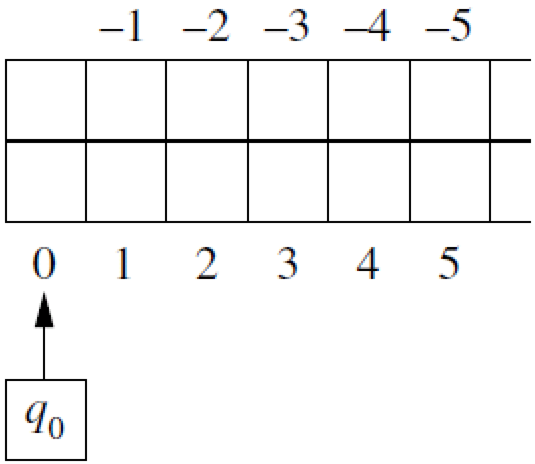


- **Double each state, one for track 1, one for track 2.**
- **Add start state which adds a "left tape end" marker # in position 0, track 2.**
- **Add an additional state after start state which returns tape head to original position to start simulation.**
- **Use # marker to detect when to switch between tracks.**

Proof – difficult direction

Standard TM M = $(Q,\Sigma,\Gamma,\delta,q_0,F)$

What is M' = $(Q',\Sigma',\Gamma',\delta',q_0',F')$?



- $Q' = (Q \cup \{q_s,q_t\}) \times \{U,D\}$
- $\Sigma' = \Sigma$
- $\Gamma' = \Gamma \cup \{\#\}$
- $q_0' = [q_s,D]$
- $F' = \{[q_i,U],[q_i,D] \mid q_i \in F\}$

Transitions defined on next slide

2-way → 2-track tape simulation

Write # and go to q_0

1. $\delta'([q_s, D], [B, B]) = [[q_t, D], [B, \#], R]$.
2. For every $x \in \Gamma$, $\delta'([q_t, D], [x, B]) = [[q_0, D], [x, B], L]$.

Read only lower symbol (many transitions)

3. For every $z \in \Gamma - \{\#\}$ and $d \in \{L, R\}$, $\delta'([q_i, D], [x, z]) = [[q_j, D], [y, z], d]$ whenever $\delta(q_i, x) = [q_j, y, d]$ is a transition of M.

Read only upper symbol

4. For every $x \in \Gamma - \{\#\}$ and $d \in \{L, R\}$, $\delta'([q_i, U], [z, x]) = [[q_j, U], [z, y], d']$ whenever $\delta(q_i, x) = [q_j, y, d]$ is a transition of M, where d' is the opposite direction of d.
5. $\delta'([q_i, D], [x, \#]) = [[q_j, U], [y, \#], R]$ whenever $\delta(q_i, x) = [q_j, y, L]$ is a transition of M.

Change directions

6. $\delta'([q_i, D], [x, \#]) = [[q_j, D], [y, \#], R]$ whenever $\delta(q_i, x) = [q_j, y, R]$ is a transition of M.
7. $\delta'([q_i, U], [x, \#]) = [[q_j, D], [y, \#], R]$ whenever $\delta(q_i, x) = [q_j, y, R]$ is a transition of M.
8. $\delta'([q_i, U], [x, \#]) = [[q_j, U], [y, \#], R]$ whenever $\delta(q_i, x) = [q_j, y, L]$ is a transition of M.

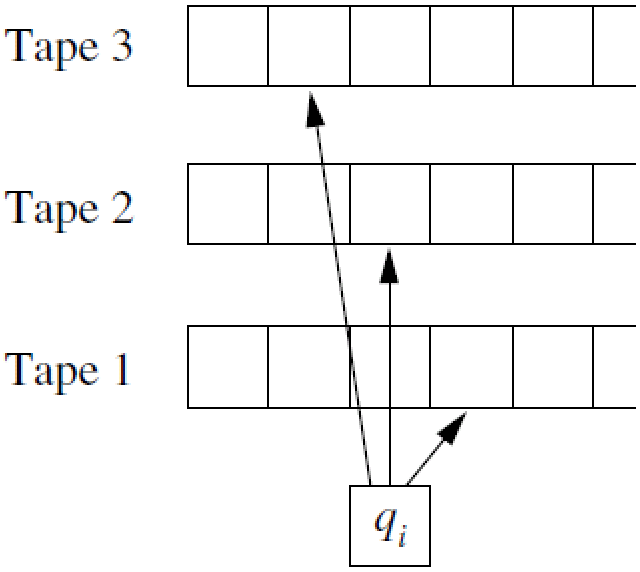
Overview: Turing Machines

Chapter 8 of [Sudkamp 2006].

1. **The Standard Turing Machine**
2. **Turing Machines as Language Acceptors**
3. **Multitrack Machines**
4. **Two-Way Tape Machines**
5. **Multitape Machines**
6. **Nondeterministic Turing Machines**
7. **Language Enumeration by Turing Machines**

Multitape-TMs

- **Enhancement:**
  TM has k tapes (k>0) – called a *k-tape TM*.

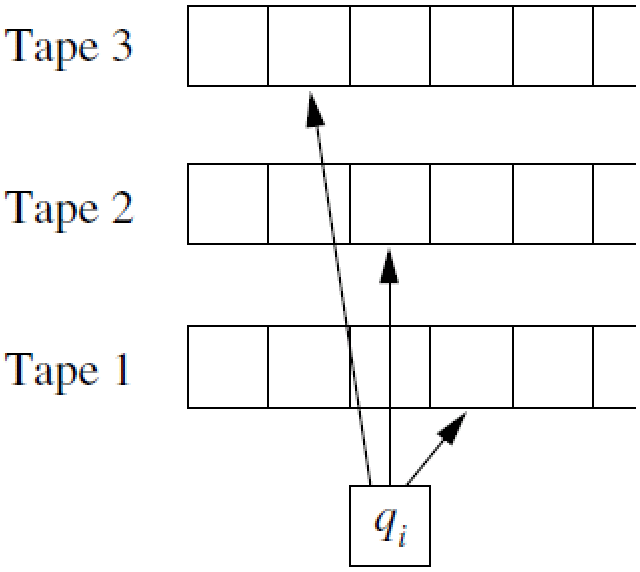


- **A transition may**
  - **change the state**
  - **write a symbol on each tape (symbols can differ for each tape)**
  - **repositions each tape head (independent of each other, stop also possible)**

- **Input is in standard position on tape 1, rest is blank**
- **Tape heads on leftmost positions**

Multitape-TMs

Formally (2-tape example):



Transitions written as

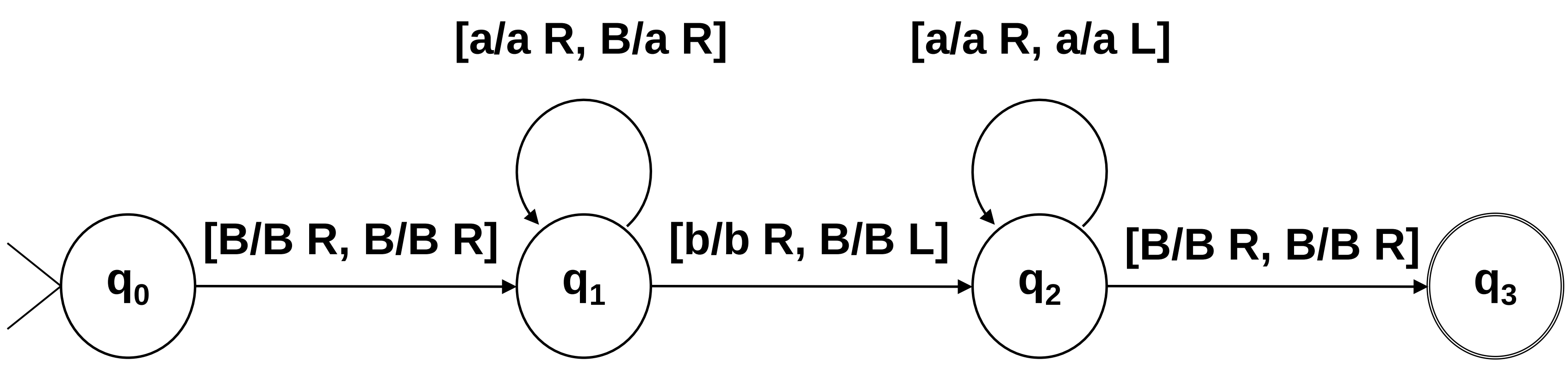
$\delta(q_i,x_1,x_2) = [q_j; y_1,d_1; y_2,d_2]$

- **Scans x_1 from tape 1, x_2 from tape 2**
- **Writes y_i on tape 1, y_2 on tape 2**
- **Changes from q_i to q_j**
- **$d_i \in \{L,R,S\}$, where S means head remains stationary**

- **If one tape head moves off the tape, TM terminates abnormally.**

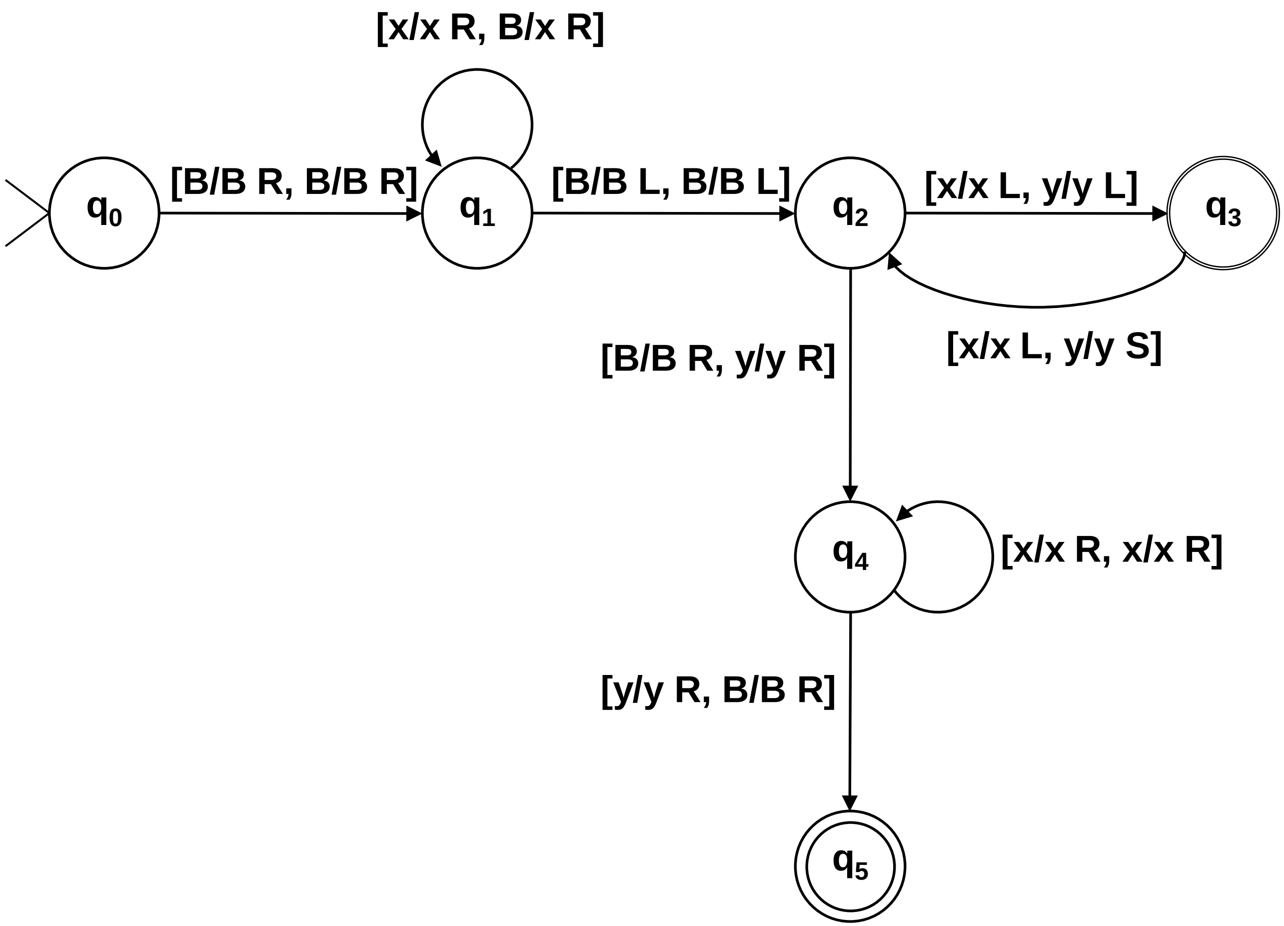
Example 1.11

Two-tape machine accepting $\{a^iba^i \mid i \geq 0\}$



Example 1.12

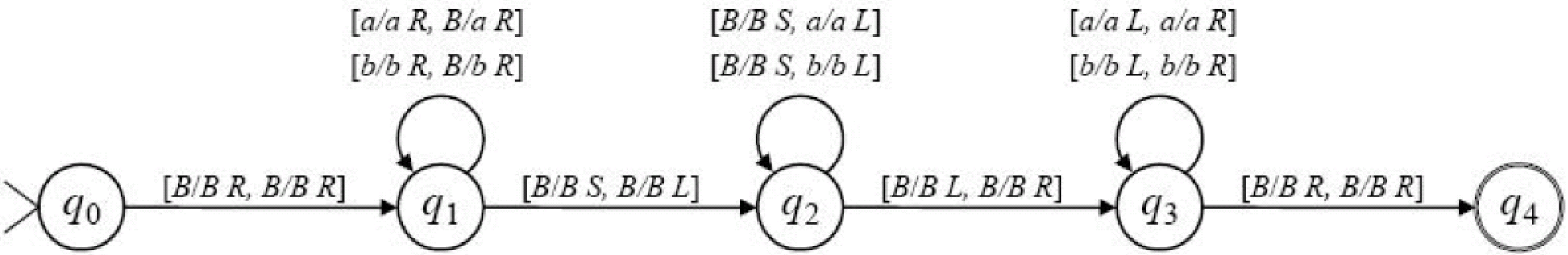
Two-tape machine accepting $\{uu \mid u \in \{a, b\}^*\}$



$x \in \{a,b\}$
$y \in \{a,b,B\}$

Example 1.13

- **2-tape TM accepting palindromes over {a,b}**



Exercise 13 [hand-in]

Make a 2-tape TM which inverts a string of a’s and b’s.

E.g., input aababba should give the output abbabaa.

Recall that the output string has to be at the same position as the input string.

Also describe, in words, how your TM performs this.

Exercise 14 [hand-in]

Prove, that every standard TM can be simulated by a 2-tape TM.

Simulation of k-tape TMs.

Theorem 1.14

If L is accepted by a k-tape TM M, then there is a standard TM N with L(N)=L(M).

Proof:

We prove this for k=2, but the argument generalizes.

Proof idea?

We construct a 2k+1-track TM.

For k=2:

Tracks 1 and 3 maintain info on tapes 1 and 2

Tracks 2 and 4 have a single non-blank position indicating the position of the tape heads of M.

Initially: write # in track 5, position 1 and X in tracks 2 and 4, position 1.
States: 8-tuples of the form $[s, q_i, x_1, x_2, y_1, y_2, d_1, d_2]$, where $q_i \in Q$, $x_i, y_i \in \Gamma \cup \{U\}$, $d_i \in \{L, R, S, U\}$. s represents the status of the simulation. U indicates an unknown item.

Let $\delta : (q_i, x_1, x_2) \mapsto [q_j; y_1, d_1; y_2, d_2]$ be the applicable transition of M.
M' start state: $[f1, q_i, U, U, U, U, U, U]$. The following actions simulate the transition of M:

1. $f1$ (find first symbol): M' moves to the right until X on track 2.
   Enter state $[f1, q_i, x_1, U, U, U, U, U]$, where x_1 is symbol on track 1 under x.
   M' returns to the position with # in track 5.
2. $f2$ (find second symbol): Same as above for recording symbol x_2 in track 3 under X in track 4.
   Enter state $[f2, q_i, x_1, x_2, U, U, U, U]$.
   Tape head returns to #.
3. Enter state $[p1, q_j, x_1, x_2, y_1, y_2, d_1, d_2]$, with q_j, y_1, y_2, d_1, d_2 obtained from $\delta(q_i, x_1, x_2)$.
4. $p1$ (print first symbol): move to X in track 2.
   Write symbol y_1 on track 1. Move X on track 2 in direction indicated by d_1.
   Tape head returns to #.
5. $p2$ (print second symbol): move to X in track 4.
   Write symbol y_2 on track 3. Move X on track 4 in direction indicated by d_2.
   Tape head returns to #.

If $\delta(q_i, x_1, x_2)$ is undefined, then simulation halts after step 2. $[f2, q_i, x_1, y_1, U, U, U, U]$ is accepting whenever q_i is accepting.

For each additional tape, add two trackes, and states obtain 3 more parameters. The simulation has 2 more actions (a find and a print for the new tape).

Overview: Turing Machines

Chapter 8 of [Sudkamp 2006].

1. **The Standard Turing Machine**
2. **Turing Machines as Language Acceptors**
3. **Multitrack Machines**
4. **Two-Way Tape Machines**
5. **Multitape Machines**
6. **Nondeterministic Turing Machines**
7. **Language Enumeration by Turing Machines**

Nondeterministic TMs

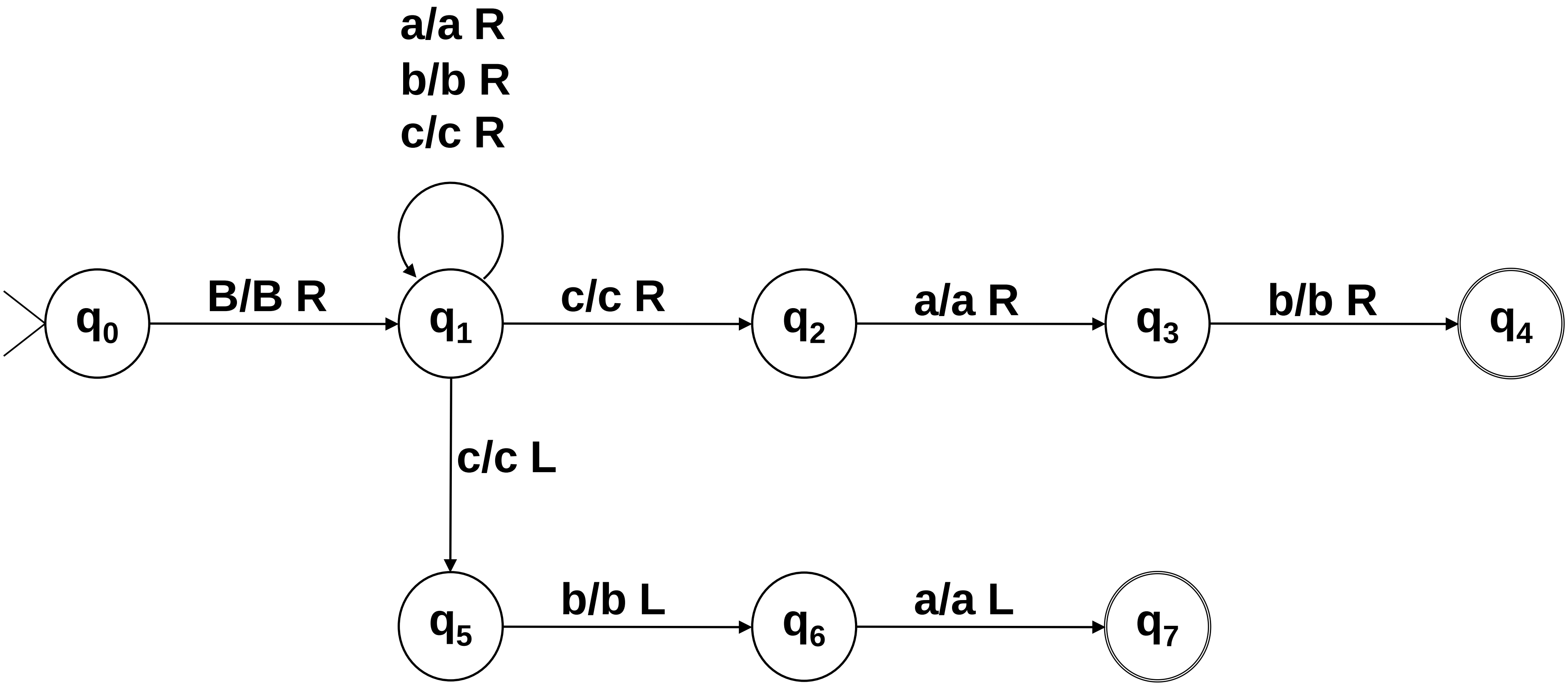
- A nondeterministic (ND) TM may specify any finite number of transitions for a given configuration.
  Formally, transitions are defined by a function from $Q\times\Gamma$ to subsets of $Q\times\Gamma\times\{L,R\}$.
- A computation arbitrarily chooses one of the possible transitions.
  Input is accepted if there is at least one computation terminating in an accepting state.
- The remaining definition is as for standard TMs.

Note:

- Other types of TMs (multi-track, 2-way, k-tape) have ND versions, too, defined similarly as above.

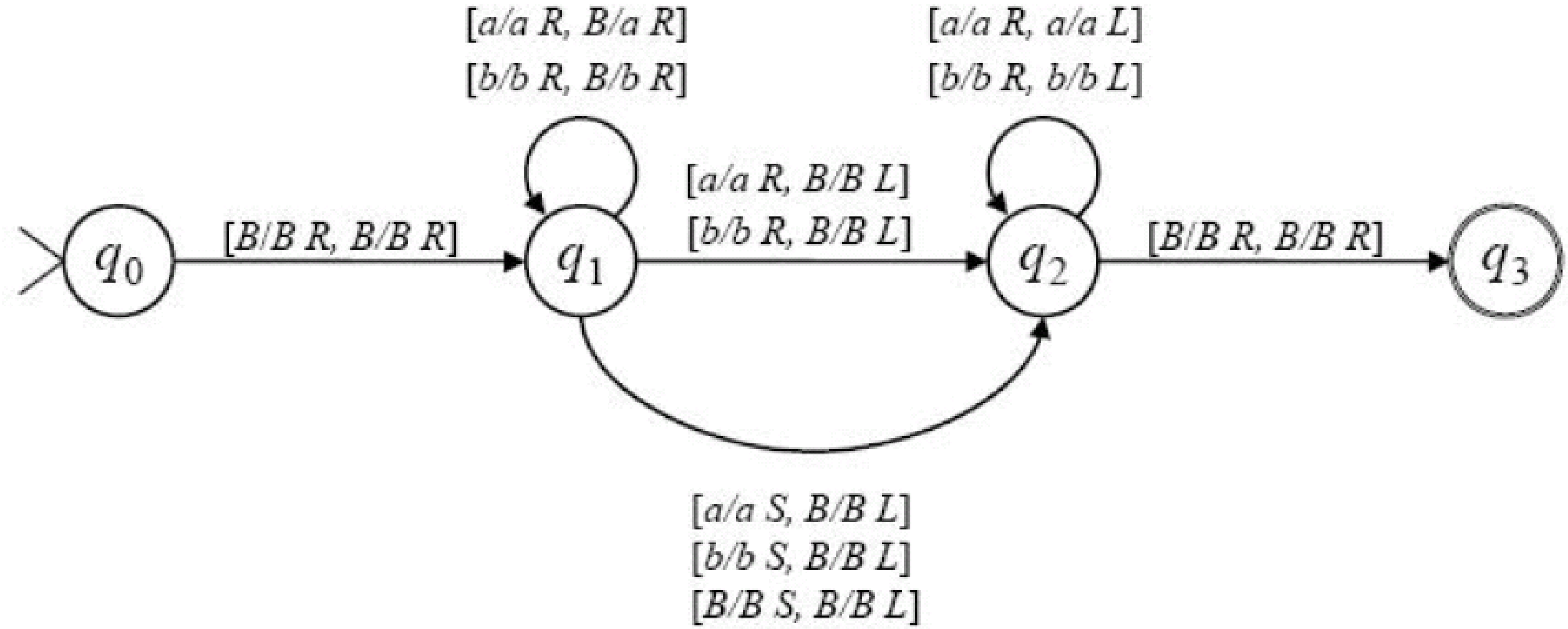
Example 1.15

ND TM accepting strings with a c preceeded <u>or</u> followed by ab



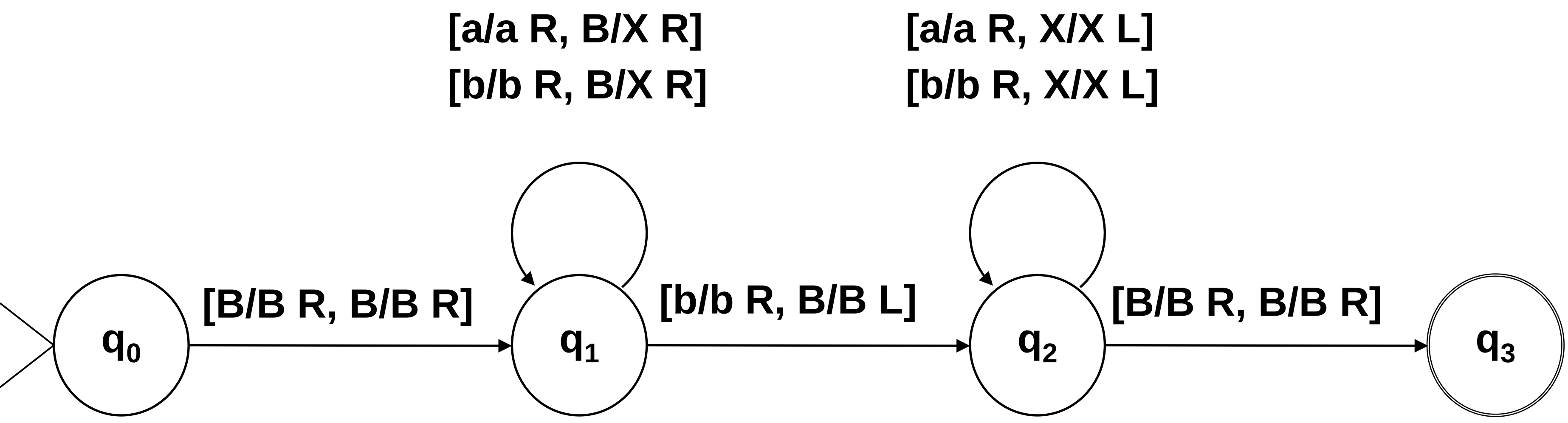
Example 1.16

2-tape ND palindrome finder



Example 1.17

ND TM accepting strings over {a,b} with b in the middle position



Exercise 15 [hand-in]

Give a ND 2-tape TM for $\{uu|u\in\{a,b\}^*\}$ which is quicker than the TM from Example 1.12.

Also, describe in words the strategy of your TM.

Exercise 16 [no hand-in]

Make a (deterministic) pseudo-code algorithm for an exhaustive search on a tree (i.e., if the sought element is not found, the whole tree should be traversed).

Exercise 17 [hand-in]

Make a non-deterministic pseudo-code algorithm for an exhaustive search on a tree.

ND TM simulation

Theorem 1.18

Let L be accepted by a ND TM M. Then there is a (deterministic) standard TM M' with L(M')=L(M).

Proof idea?

ND TM simulation – proof sketch

Let c be the maximum number of transitions for any state, symbol pair of M.

Simulation idea: use 3-tape TM M'.

- Tape 1 holds input
- Tape 2 is used for simulating the tape of M
- Tape 3 holds sequences $(m_1,\ldots,m_n)$ $(1 \leq m_i \leq c)$, which encode computations of M:
  m_i indicates that, from the (maximally) c choices M has in performing the i-th transition, the m_i-th choice is selected.
  It is easy to generate all $(m_1,\ldots,m_n)$ in sequence.

M is simulated as follows:

1. Generate the first tuple $(m_1,\ldots,m_n) = (1,\ldots,1)$
2. Simulate M according to $(m_1,\ldots,m_n)$
3. If input is not accepted, generate next $(m_1,\ldots,m_n)$ and continue with step 2

Overview: Turing Machines

Chapter 8 of [Sudkamp 2006].

1. **The Standard Turing Machine**
2. **Turing Machines as Language Acceptors**
3. **Multitrack Machines**
4. **Two-Way Tape Machines**
5. **Multitape Machines**
6. **Nondeterministic Turing Machines**
7. **Language Enumeration by Turing Machines**

Language enumeration

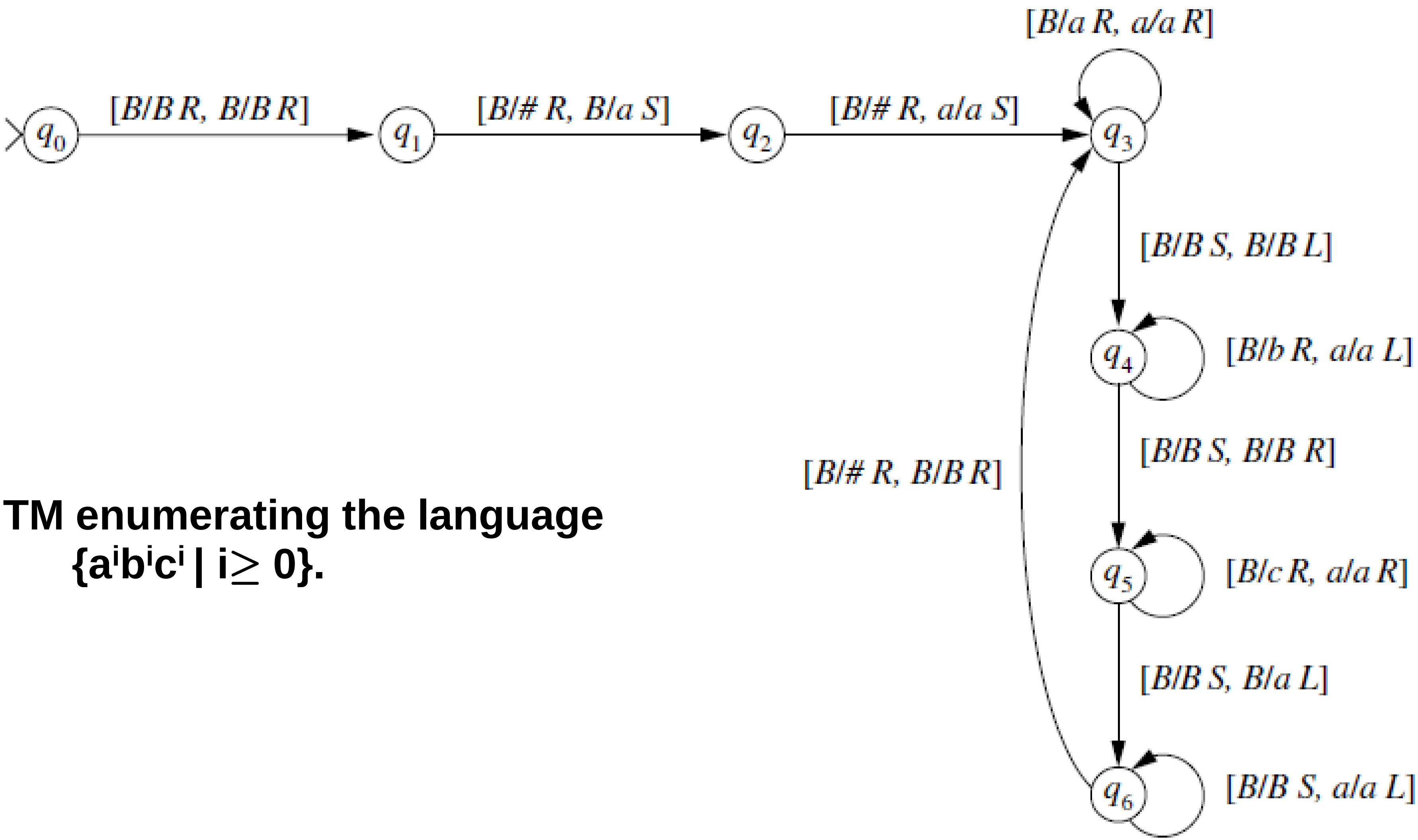
A deterministic k-tape ($k \geq 2$) Turing Machine *enumerates* a language L if all of the following hold.

- **The computation begins with all tapes blank.**
- **With each transition, the tape head on tape 1 (the output tape) remains stationary or moves to the right.**
- **At any point in the computation, the nonblank portion of tape 1 has the form**
  B#u1#u2#...#um# or B#u1#u2…#um#v
  where u1,u2,… are in L and v is a string over the tape alphabet.
- **A string u will be written on tape 1 preceded and followed by # if, and only if, u is in L.**

Note: computation does not need to halt.

Does this give rise to a different notion, which languages are computable?

Example 1.17



TM enumerating the language $\{a^i b^i c^i \mid i \geq 0\}$.

Exercise 18 [hand-in]

Make a (deterministic) Turing Machine which enumerates all strings in the language a*b*.

Give, in words, a description of the strategy of your Turing Machine.

Enumeration

Theorem 1.19

Let L be a language enumerated by a Turing Machine E. Then there is a Turing Machine E' that enumerates L and each string in L appears only once on the output tape of E'.

Proof idea?

Proof

Let E be a k-tape TM enumerating L.

Define (k+1)-tape TM E’ using E as a submachine that produces strings to be considered for output by E’.

The additional tape is the output tape of E’ (called tape 1).

Output tape of E becomes a working tape for E’.

I.e. tapes 2,…,k+1 simulate E (simulation output goes on tape 2).

Actions of E’: the following sequence of steps.

1. Simulate E on tapes 2,…,k+1
2. After #u# appears on tape 2, E’ checks if u is already on tape 2.
3. If u is not on tape 2, it is added to tape 1 as output.
4. Restart simulation of E to produce the next string.

Equivalence of both approaches

Theorem 1.20

A language is recursively enumerable if, and only if, it can be enumerated by a Turing Machine.

Proof idea?

Which is the easy direction?

Proof Theorem 1.20 part 1

Lemma 1.21: If L is enumerated by a TM, then it is rec. enumerable.

Assume L is enumerated by a k-tape TM E.

We construct a (k+1)-tape TM M accepting L.

Additional tape of M is the input tape, remaining tapes simulate E.

M starts with a string u on its input tape.

Then M simulates E;
when the simulation writes #, a string $w \in L$ has been generated.

M then compares u with w and accepts u if u=w.

Otherwise, the simulation of E is used to produce another string from L and the comparison cycle is repeated.

If $u \in L$, it will eventually be produced by E and thus accepted by M.

Proof Theorem 1.20 part 2

Lemma 1.22

If L is recursively enumerable, then there is a TM E which enumerates it.

Proof idea?

Not a proof Theorem 1.20 part 2

Why does the following not work?

Let M be such that it accepts L.

Actions of E:

1. **Generate a string $u \in \Sigma^*$.**
2. **Simulate the computation of M with input u.**
3. **If M accepts, write u on the output tape.**
4. **Continue at step 1 until all strings in $u \in \Sigma^*$ have been tested.**

Proof Theorem 1.20 part 2 (cont)

Definition 1.23

Let $\Sigma=\{a_1,\ldots,a_n\}$. The lexicographic ordering lo of Σ^* is defined recursively as follows.

1. **Basis: $lo(\lambda)=0$, $lo(a_i)=i$ for $i=1,2,\ldots,n$.**
2. **Recursive step: $lo(a_i u)= lo(u) + I \cdot n^{length(u)}$.**

We write

$u<v$ if $lo(u)<lo(v)$

$u=v$ if $lo(u)=lo(v)$

$u>v$ if $lo(u)>lo(v)$

Exercise 19 [no hand-in]

Let $\Sigma = \{0,1\}$ with lo(0) < lo(1).

Give the first ten strings of Σ^* in the lexicographic ordering.

Proof Theorem 1.20 part 2 (cont)

Lemma 1.24

For any alphabet Σ, there is a TM E_Σ, that enumerates Σ^* in lexicographic order.

Proof.

Skipped.

Exercise 20 [hand-in]

Give a TM which enumerates all strings over {0,1}.

Give, in words, a description of the strategy of your Turing Machine.

Proof Theorem 1.20 part 2 (cont)

Let M be a TM that accepts L.

The lexicographic ordering produces a listing $u_0=\lambda$, u_1, u_2, … of the strings in Σ^*.

Consider all pairs [u_i, j], where j is a non-negative integer.

[u_i, j] means: run M on u_i for j steps.

Idea: Subsequently do this for all [u_i, j]
with i+j=0 (1 pair),
then i+j=1 (2 pairs),
then i+j=2, (3 pairs),
…
and each of these can be guaranteed to terminate.

Proof Theorem 1.20 part 2 (cont)

More formally:

1. **Generated an ordered pair [i, j].**
2. **Run a simulation of M with input u_i for j transitions or until the simulation halts.**
3. **If M accepts, write u_i on the output tape.**
4. **Generate the next ordered pair.**
5. **Continue with step 2.**

If $u_i \in L$, then the computation of M with input u_i halts and accepts after m transitions, for some number m. Thus, u_i will be written to the output tape of E when the ordered pair [i, m] is processed.

The second element in a pair [i, j] ensures that the simulation of M terminates (after j steps).

Recursivity

- **We just characterized recursively enumerable languages.**
- **How to characterize *recursive* languages?**

Recursivity

Theorem 1.25

L is recursive if, and only if, L can be enumerated in lexicographical order.

Proof idea?

Proof part 1

Let L be recursive over Σ, accepted by M which always halts.

Let F be a TM which enumerates all strings in Σ^*.

Let E be the TM which does the following.

1. Run F, producing some $u \in \Sigma^*$.
2. Run M with input u.
3. If M accepts u, u is written on the output tape of E.
4. The generate-and-test loop continues with step 1.

Since M always terminates, each string $u \in \Sigma^*$ will be generated and tested for membership in L.

Proof part 2

Let L be a language enumerated by a TM E in lexicographic order.

Case 1: L is finite. Then L is recursive since every finite language is recursive (Exercise 21).

Case 2: L is infinite.

We construct a (k+1)-tape TM M enumerating L in lexicogr. order.

Additional tape is input tape, other tapes are for simulating E.

M starts with a string u on its input tape.

Next, M simulates E.

If the simulation produces a string w, M compares u with w.

If u=w, then M halts and accepts.

If w is greater than u in the ordering, M halts and rejects.

If lo(w)<lo(u), simulation of E is restarted to produce another element of L, and the comparison cycle is repeated.

Exercise 21 [hand-in]

Show, that every finite language is recursive.