

1. Design a TM which on a input $w \in \{0, 1\}^*$, shifts w over one position to the right.

That is: $(s, \# w [\#]) \vdash^* (h, \# \# w [\#])$.

2. Show the computation of your TM on the input 010:

$(s, \# 0 1 0 [\#]) \vdash \dots$

3. Show what your TM does on input ε :

$(s, \# [\#]) \vdash \dots$

Announcements:

My TM definitions are slightly different from the book- use my definitions on assignments and with the TM simulator.

Assignment #4: Due at the beginning of class, Wed. July 18.

Turing Machines- Definition

This lecture formally defines the Turing machine and gives more examples of how they work.

With them, we will be able to prove statements regarding the limitations of modern day computers.

$L = \{ w \in \{a, b\}^* : w \text{ has an even number of } a\text{'s} \}$

A TM M **decides** a language L if

$(s, \# w \#) \vdash^* (h, \# Y \#)$ for $w \in L$ and

$(s, \# w \#) \not\vdash^* (h, \# N \#)$ for $w \notin L$.

To decide L :

Move left erasing symbols as we go and keeping track of the number of a 's modulo 2 until reaching the blank at the end and then write the answer on the tape.

State	Symbol	Next state	Head Instr.
start	#	evena	L
evena	a	rem_o	#
evena	b	rem_e	#
odda	a	rem_e	#
odda	b	rem_o	#
rem_e	#	evena	L
rem_o	#	odda	L
evena	#	writey	R
odda	#	writen	R
writey	#	writey	Y
writen	#	writen	N
writey	Y	h	R
writen	N	h	R

A **Turing Machine** M consists of a quadruple (K, Σ, δ, s) where

K is a finite set of states, Σ is an alphabet,

δ , the transition function is a function from $K \times \Sigma$ to $(K \cup h) \times (\Sigma \cup \{L, R\})$ and

$s \in K$ is the start state.

Technical notes: in practice we allow leaving part of the function undefined and just say the TM hangs when an undefined transition is encountered. Also, $\#$ is always in Σ and h (the halt state) $\notin K$.

Example TM $M = (K, \Sigma, \delta, s)$

where

$K = \{\text{start, evena, odda, rem_e, rem_o, writey, writen}\},$

$\Sigma = \{\#, a, b, Y, N\},$

δ was given previously (but not all transitions were defined), and $s = \text{start}$.

Side note: these definitions differ slightly from second edition of text for ease in programming.

The format of the input to the TM simulator is as follows:

<Name of start state>

<current state> <current symbol> <next state> <head instruction>

...

<current state> <current symbol> <next state> <head instruction>

\$

<input string w1>

<input string w2>

...

Rules for TM Descriptions

1. The state name `h` is used to denote the halting state.
2. Use the symbol `#` to represent a blank.
3. If a line starts with `//` it is a comment. Comments can also be added on the same line as an instruction at the end of the line (start with `//`).
4. Each of the state names is an arbitrary string. The current symbol and new symbol each must be a single symbol.
6. The head instruction is either `L` (move the head one square left) or `R` (move the head one square right) or a symbol to replace the current tape square contents.
7. The `$` indicates the end of the TM description.

start

Input to TM
Simulator

start # evena L

// Erase current symbol using state to

// remember even/odd

evena a rem_o #

evena b rem_e #

odda a rem_e #

odda b rem_o #

// Move left off square just blanked out

// to correct state

rem_o # odda L

rem_e # evena L

// Found # at LH end of tape

evena # writey R

odda # writen R

// Write the answer

writey # writey Y

writen # writen N

// Position head to right of answer and halt

writey Y h R

writen N h R

\$

abaa

baab

Step 0 : (start, #baab[#])
Step 1 : (evena, #baa[b])
Step 2 : (rem_e, #baa[#])
Step 3 : (evena, #ba[a])
Step 4 : (rem_o, #ba[#])
Step 5 : (odda, #b[a])
Step 6 : (rem_e, #b[#])
Step 7 : (evena, #[b])
Step 8 : (rem_e, #[#])
Step 9 : (evena, [#])
Step 10 : (writey, #[#])
Step 11 : (writey, #[Y])
Step 12 : (h, #Y[#])

Computation
on input baab
which is in L.

Step 0 : (start, #abaa[#])
Step 1 : (evena, #aba[a])
Step 2 : (rem_o, #aba[#])
Step 3 : (odda, #ab[a])
Step 4 : (rem_e, #ab[#])
Step 5 : (evena, #a[b])
Step 6 : (rem_e, #a[#])
Step 7 : (evena, #[a])
Step 8 : (rem_o, #[#])
Step 9 : (odda, [#])
Step 10 : (writen, #[#])
Step 11 : (writen, #[N])
Step 12 : (h, #N[#])

Computation
on input abaa
which is not
in L.

A COPY TM.

On input $w \in \{a, b\}^*$, this TM halts with w followed by $\#$ followed by a copy of w .

That is:

$$(s, \# w [\#]) \vdash^* (h, \# w \# w [\#]).$$

The program for this TM is available from the page which gives the TM simulator.

The algorithm changes each a to A and each b to B in the first copy of w to mark that it has been copied over already.

// Find leftmost symbol of w not copied yet.

middle # goleft L

start state: middle

goleft a goleft L

goleft b goleft L

// Found either #, A, B from part being copied.

goleft A next_s R

goleft B next_s R

goleft # next_s R

// Go to # between w and copy of w

//remembering symbol to copy.

next_s a next_s A

next_s b next_s B

next_s A RtoM_a R

next_s B RtoM_b R

next_s # clean L

// Done- clean up.

// Go right to the middle

RtoM_a a RtoM_a R
RtoM_a b RtoM_a R
RtoM_a # RtoR_a R

RtoM_b a RtoM_b R
RtoM_b b RtoM_b R
RtoM_b # RtoR_b R

// Go right to the right hand end

RtoR_a a RtoR_a R
RtoR_a b RtoR_a R
RtoR_a # left1 a

RtoR_b a RtoR_b R
RtoR_b b RtoR_b R
RtoR_b # left1 b

// Go left to blank in middle.

left1 a left1 L

left1 b left1 L

left1 # middle #

// Clean up the tape-

//change A back to a and B back to b.

clean	A	clean	a
clean	B	clean	b
clean	a	clean	L
clean	b	clean	L
clean	#	right1	R

// Position head to right of copy of w.

right1	a	right1	R
right1	b	right1	R
right1	#	right2	R
right2	a	right2	R
right2	b	right2	R
right2	#	h	#