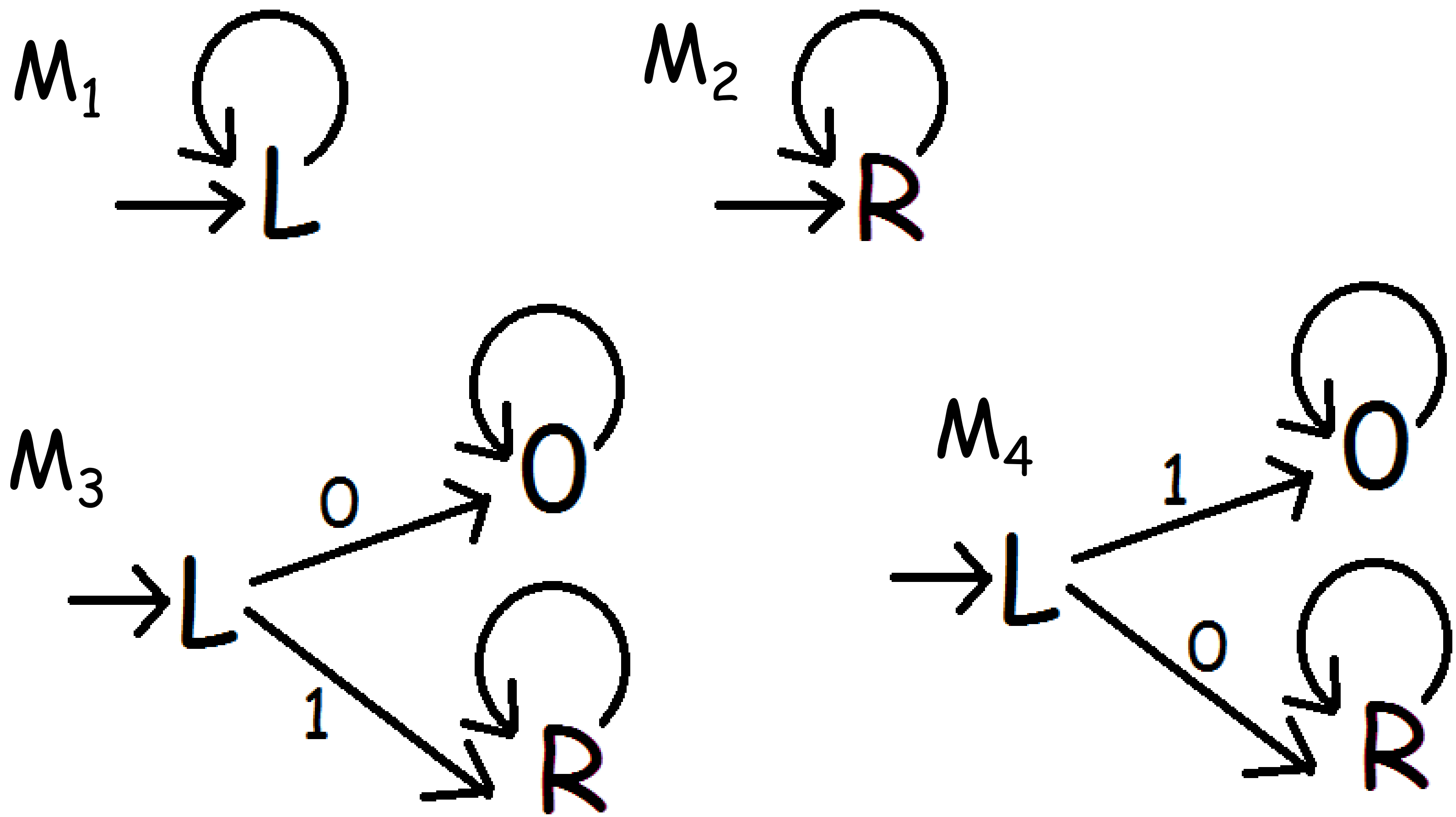
What do these TM's do on input on input 001? Standard input format: (s, #001[#]).



Theorem: Turing decidable languages are closed under difference.

Proof:

Let M_1 be a TM which decides L_1, and

let M_2 be a TM which decides L_2.

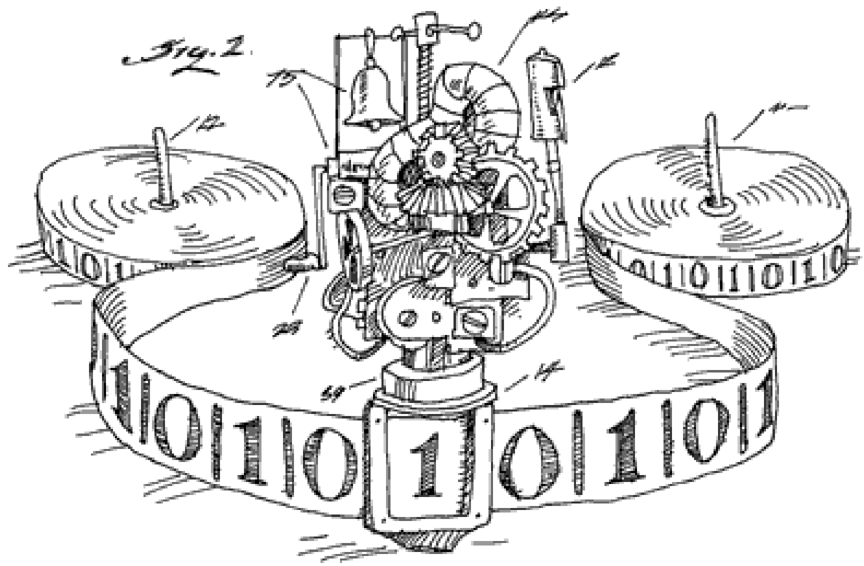
Let C be a TM which makes a copy of the

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

Finish the proof by drawing a machine schema for a TM which decides $L = L_1 - L_2$.

Extensions of TM's/UTM's

It can be proven that adding extra power to a TM by adding multiple tracks, tapes, or tape heads does not change what it is able to compute. These more powerful models can be simulated on our single tape/one head machine.



Turing Machine by Tom Dunne
American Scientist, March-April 2002

Two-way infinite tape:

…	#	a	b	a	a	#	b	b	#	…

Multiple tapes:

#	a	a	a	#	b	b	b	#	…

#	a	b	a	b	#	a	b	#	…

Multiple tracks:

#	a	b	a	a	b	#	#	#	…
#	b	a	b	b	a	#	#	#	

Two tracks- Each tape square has one symbol on upper track and one symbol on lower track:

#	a	b	b	#	#	#	#	#	...
#	#	#	#	#	#	#	#	#	

To simulate this on a standard TM:

On standard TM initially:

#	a	b	b	#	#	#	#	#	...

Initial alphabet is s1, s2, … sk.

New symbols= { $\binom{si}{sj}$: i=1, 2, …k, j=1, 2, …k}

#	a	b	b	#	#	#	#	#	...

Reformat initial tape:

$\binom{\#}{\#}$	$\binom{a}{\#}$	$\binom{b}{\#}$	$\binom{b}{\#}$	$\binom{\#}{\#}$	#	#	#	#

Each time TM moves onto a # square, reformat it to be: $\binom{\#}{\#}$

For each state q add a transition:

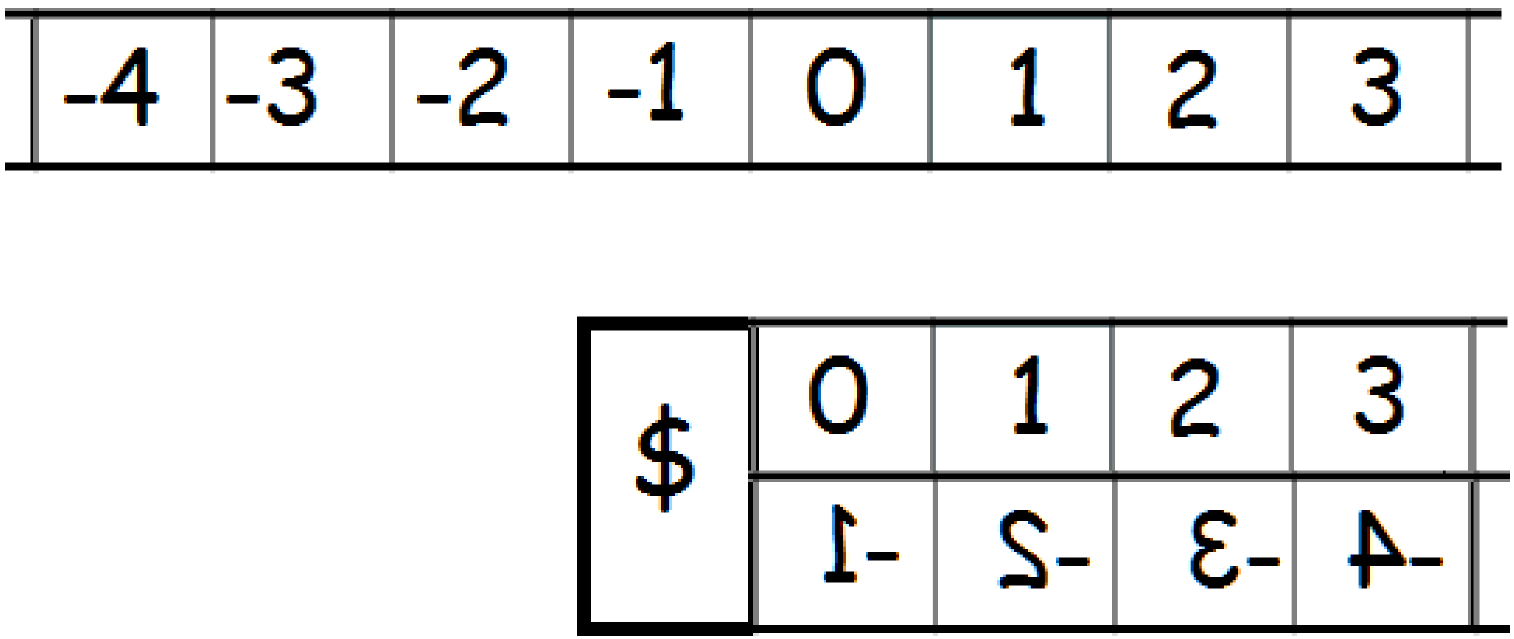
$\delta(q, \#) \rightarrow (q, \binom{\#}{\#})$

Two-way infinite tape:

…	#	a	b	a	a	#	b	b	#	…

How can this be simulated with a Turing machine that has 2 tracks?

Conceptually, the infinite tape is "bent" and wrapped around at the as follows, with $ to mark the bend:



To initialize the tape:

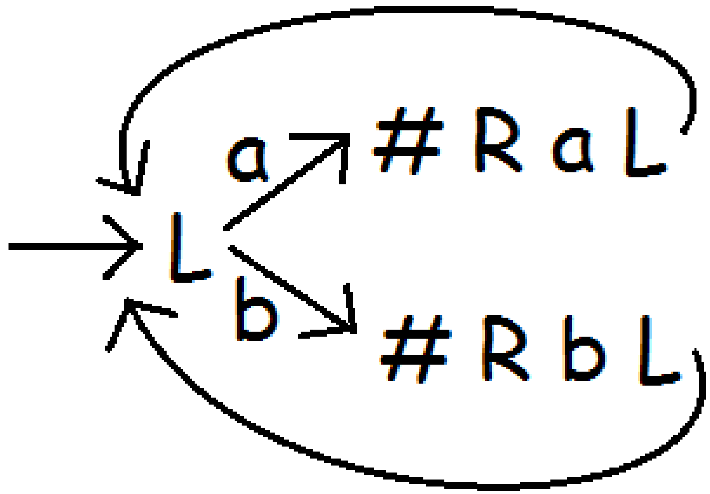
...

#	b	a	b	#	#	#	#	#

Shift right and convert to two track mode with end of tape marker:

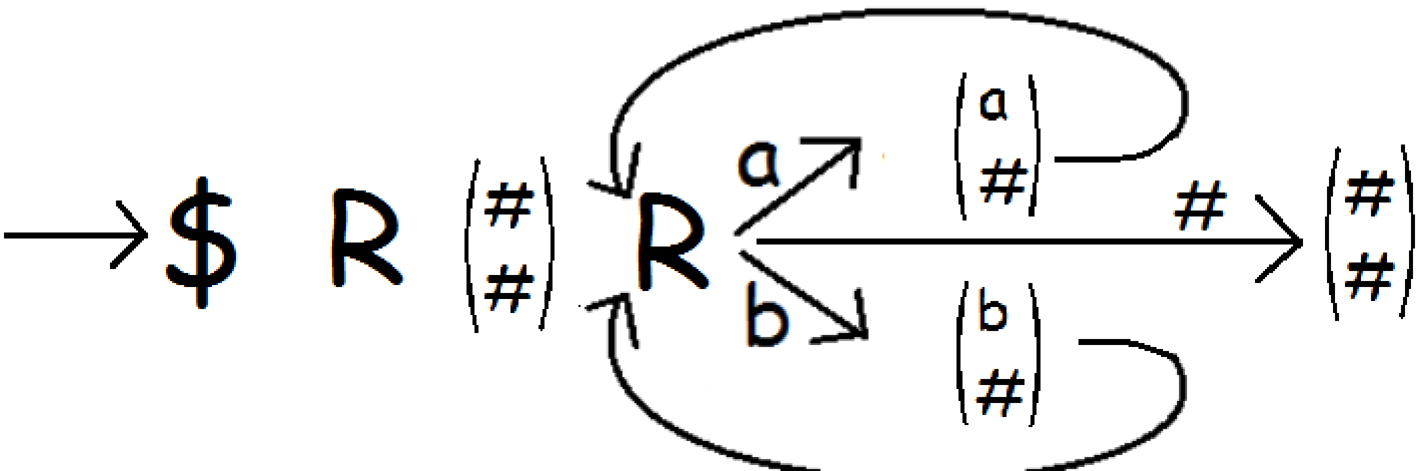
$	#	b	a	b	#	#	#	#	#
	#	#	#	#	#	#	#	#	#

M_1:



w [#]
changes to
[#] # w #

M_2: 2-track formatting.



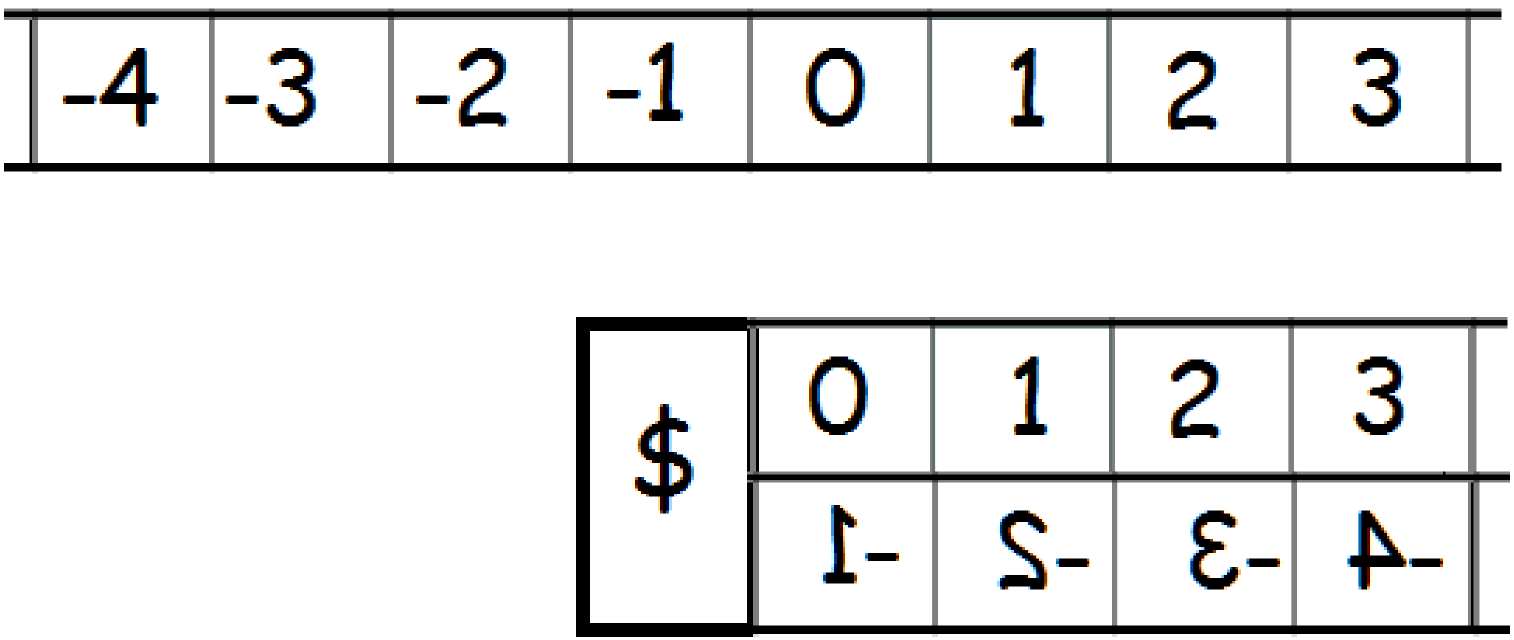
Old Transitions:

s	#	s	L
s	a	h	#
s	b	s	R

Add:

1. Upper and lower track states and transitions.
2. Reformat of # squares to 2-track.
3. If we hit $ change tracks.

Careful: if we are going left on the original tape, this corresponds to left for squares 0, 1, 2, …
but right for squares -1, -2, -3, …
on the 2-track simulation.



Multiple tape heads:

#	a	a	b	b	a	a	#

Keep track of tape head positions on extra tracks:

<table>
<tr><td>#</td><td>a</td><td>b</td><td>b</td><td>a</td><td>a</td><td>#</td><td rowspan="3">#</td><td rowspan="3">#</td><td rowspan="3">#</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

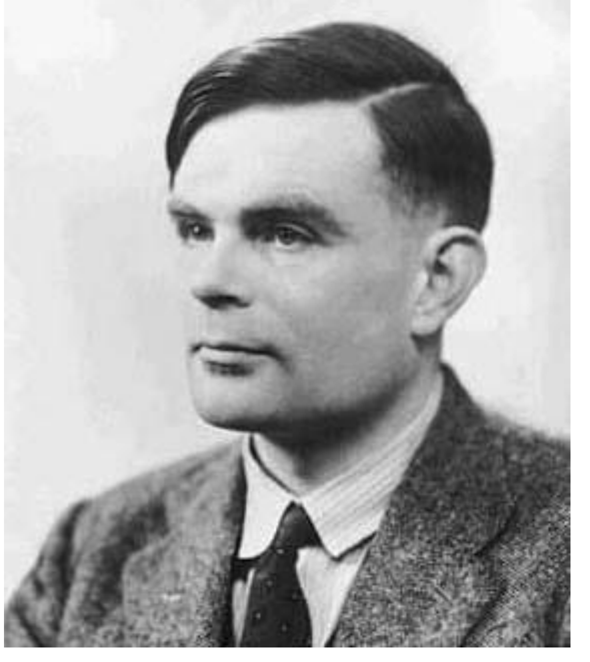
Multiple tapes:

<table>
<tr><td>$</td><td>#</td><td>b</td><td>b</td><td>a</td><td>b</td><td>b</td><td>#</td><td rowspan="4">#</td><td rowspan="4">#</td><td rowspan="4">#</td></tr>
<tr><td>$</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>$</td><td>#</td><td>b</td><td>b</td><td>a</td><td>b</td><td>b</td><td>#</td></tr>
<tr><td>$</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
</table>

Universal Turing Machines

A Universal TM (UTM) is like my java TM simulator but written in TM. We can argue that a 3-tape TM can be used to create a UTM which can execute instructions from an arbitrary TM program.

The existence of a UTM is used to acquire some problems which can be proven to not be Turing-decidable.



Alan Turing (1912-1954)

Universal TM's:

the birth of the idea of having programmable computers.

Software can be used instead of designing new hardware.

"Universal Turing Machine" Jin Wicked.

Problem: A UTM is a Turing machine and hence it must have a fixed finite alphabet.

But it must be able to simulate any TM with an arbitrary alphabet.

How can we do this?

Hint: Our computers which we use to run Java and C programs have an underlying alphabet of 0/1. But they still can represent lots of symbols!

First number the states and symbols:

State	Sym	Next state	Head
s	#	s	L
s	a	t	a
s	b	t	L
t	#	h	#
t	a	t	R
t	b	t	b

Num	State	Head
0	h	L
1	s	R
2	t	#
3		a
4		b

Num	State	"State"	Head	"Head"
0	h	q00	L	a000
1	s	q01	R	a001
2	t	q10	#	a010
3			a	a011
4			b	a100

Assumptions: h is always state 0.

The start state is state 1.

The symbols always have L= 0, R = 1, # = 2.

State	Sym	Next state	Head
s	#	s	L
s	a	t	a
s	b	t	L
t	#	h	#
t	a	t	R
t	b	t	b

0	h	q00	L	a000
1	s	q01	R	a001
2	t	q10	#	a010
3			a	a011
4			b	a100

"M" is a string representing a TM M which uses the alphabet { (,), q, a, 0, 1, ,}

"w" is a string representing w which uses the alphabet {a, 0, 1}.

State	Sym	Next state	Head
s	#	s	L
s	a	t	a
s	b	t	L
t	#	h	#
t	a	t	R
t	b	t	b

0	h	q00	L	a000
1	s	q01	R	a001
2	t	q10	#	a010
3			a	a011
4			b	a100

"M"=

(q01, a010, q01, a000), (q01, a011, q10, a011), (q01, a100, q10, a000),

(q10, a010, q00, a010), (q10, a011, q10, a001), (q10, a100, q10, a100)

"abaa"= a011a100a011a011

"#ab#"= a010a011a100a010

Initially:

Tape 1: # "M" [#]

Tape 2: #q00… 01[#]

Tape 3: # "Tape contents for M" #

#ab[#] → #a010a011a100[a]010#

Original TM M:

Head moves left/right: move head on third tape left/right until reaching "a" (or #).

Blanks to right of input: reformat to "#"= a0…10.

Hit blank to left of input: original TM hangs.

To do one move:

Search for the current symbol (head on tape 3 is on "a" of its encoding) and the current state (from tape 2) in "M" on tape 1.

When the applicable transition is found:

1. update the current state name on tape 2,

2. move the head on tape 3 (head instruction is a0…0 = L or a00..01 = R) or replace the current symbol encoding on tape 3.

Sample final exam question:

For each of the following languages, indicate the most restrictive of the classes below into which it falls

(a) finite
(b) regular
(c) context-free
(d) Turing-decidable
(e) Turing-acceptable
(f) None of the above.

1. Φ
2. $(a \cup b)^*$
3. $\{a^n b^n : n \geq 0\}$
4. $\{w \in \{\#,(,),0,1,a,q,,\}^*$: w="M" for some TM M}
5. H = { "M" "w" : M halts when run on input w}
6. { "M" "w" : M does not halt on input w}