Theorem: Turing decidable languages are closed under intersection.

Proof:

Let M_1 be a TM which decides L_1, and

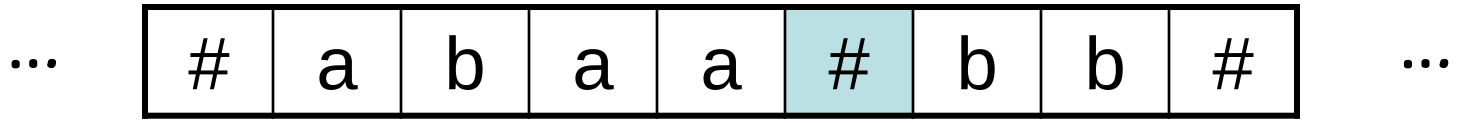
let M_2 be a TM which decides L_2.

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 $L = L_1 \cap L_2$.

Two-way infinite tape:



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:

-4	-3	-2	-1	0	1	2	3

$	0	1	2	3
	-1	-2	-3	-4

To initialize the tape:

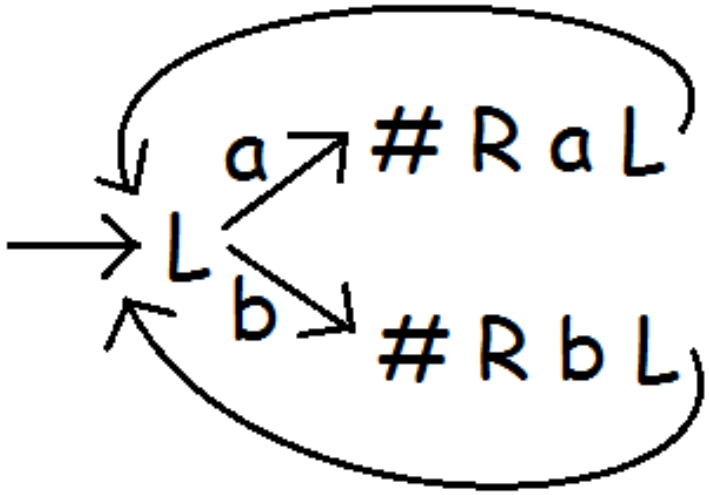
...

#	b	a	b	#	#	#	#	#

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

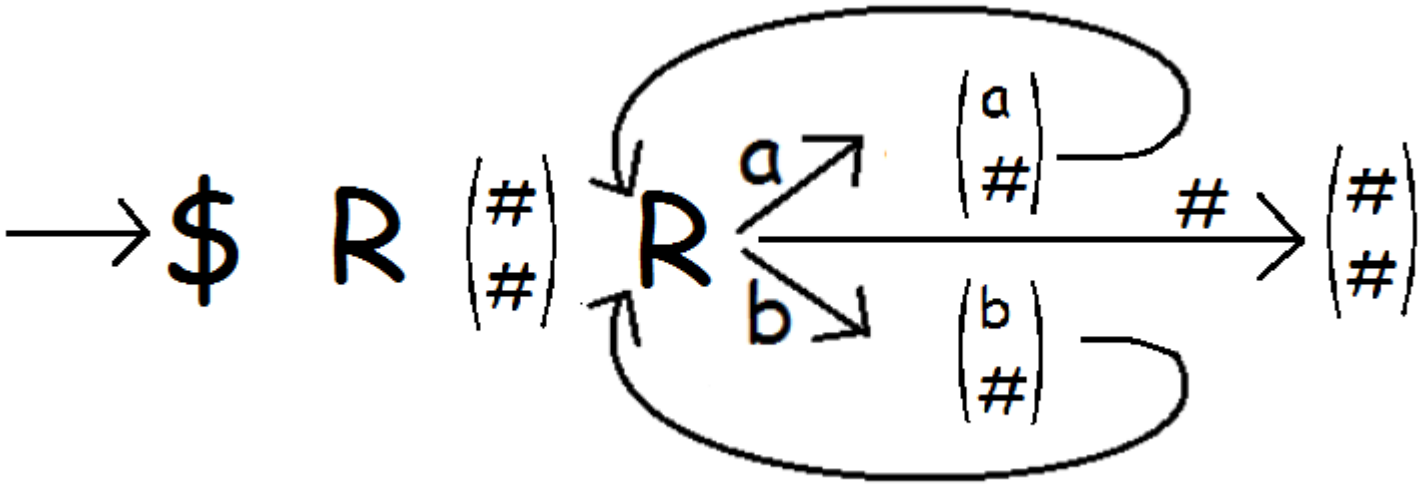
$	#	b	a	b	#	#	#	#	#
	#	#	#	#	#	#	#	#	#

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.

-4	-3	-2	-1	0	1	2	3

$	0	1	2	3
	-1	-2	-3	-4

Theorem: Turing decidable languages are closed under complement.

Proof:

Let M be a TM which decides L.

It is easy to construct the machine schema for a TM which decides the complement of L.

Algorithm: Run M. Change Y to N and N to Y at end then position head appropriately.

Theorem: All Turing-decidable languages are Turing-acceptable.

Recall:

Decide means to halt with (h, #Y[#]) when w is in L and (h, #N[#]) when w is not in L.

Accept means that the TM halts on w when w is in L and hangs (head falls off left hand end of tape or there is an undefined transition) or computes forever when w is not in L.

Proof: Given a TM M_1 that decides L we can easily design a machine M_2 which accepts L.

Theorem: Turing-decidable languages are closed under Kleene star.

Example: w= abcd

Which factorizations of w must be considered?

w_1	w_2	w_3	w_4
a	b	c	d
a	b	cd	
a	bc	d	
a	bcd		
ab	c	d	
ab	cd		Kleene
abc	d		Star
abcd			Cases

```
public static void testW(int level, String w)
{
    int i, j, len;  String u, v;

    len= w.length( ); if (len == 0) return;

    for (i=1; i <= len; i++)
    {
        u= w.substring(0,i);
        for (j=0; j < level-1; j++)
             System.out.print("            ");
        System.out.println(
            "W" + level + " = " + u);
        v= w.substring(i, len);
        testW(level+1, v);
    }
}
```

```
w1 = a
                w2 = b
                                  w3 = c
                                                    w4 = d
                                  w3 = cd
                w2 = bc
                                  w3 = d
                w2 = bcd
w1 = ab
                w2 = c
                                  w3 = d
                w2 = cd
w1 = abc
                w2 = d
w1 = abcd
```

Output

Thought question: what would you do to determine if a string w is in $L_1 \cdot L_2$ if you have TM's which decide L_1 and L_2?

High level pseudo code is fine. It would not be fun to program this on a TM.

Summary: Closure

Operation	Turing-decidable	Turing-acceptable
Union	yes	yes
Concatenation	yes	yes
Kleene star	yes	yes
Complement	yes	no *
Intersection	yes	yes

* - need proof (coming soon)

Argue that the following languages are Turing-decidable:

(a) {"M" "w" : M halts on w after at most 700 steps}

(b) {"M" "w": M halts on w without using more than the first 100 tape squares}

For part (b): we only have to simulate M for a finite number of steps and in that time frame it will either halt, hang, use more than the first 100 tape squares, or it will never halt. How many steps must we run M for?

Tutorials: What happens in them? Would you prefer more structured ones (posted problems to solve) or do you prefer to just ask questions in your trouble areas? How could they be structured to better help you to learn the class material?

Do you like our unusual grading scheme?