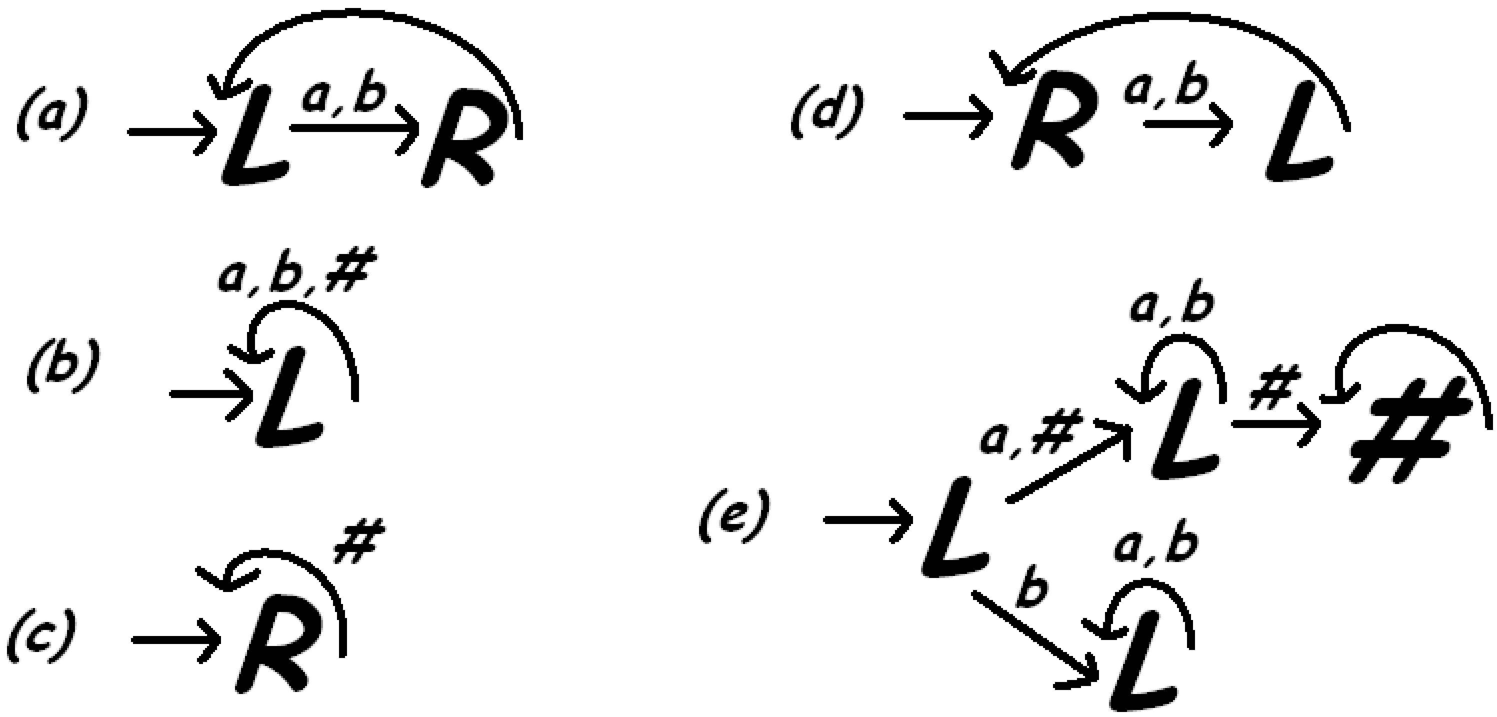
Which languages do the following TM's accept over the alphabet Σ={a, b}?



Recall: A TM M accepts a language L defined over an alphabet Σ if M halts on all w in L and either hangs or computes forever when w is not in L.

Announcements

Assignment #4 has been posted: due Wed. July 18.

Final exam tutorial:

Monday Aug. 6, 10am, ECS 116.

If the building is locked, I will prop open the back door to ECS (the one that opens on to the campus).

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.

start state: middle

```
middle   #   goleft   L
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	#

A TM $M = (K, \Sigma, \delta, s)$ decides a language L defined over an alphabet $\Sigma_1 \subseteq \Sigma$ ($\# \notin \Sigma_1$) if for all strings $w \in \Sigma_1^*$,

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

$(s, \# w [\#]) \vdash^* (h, \# N [\#])$ for $w \notin L$.

Theorem: Turing decidable languages are closed under union.

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

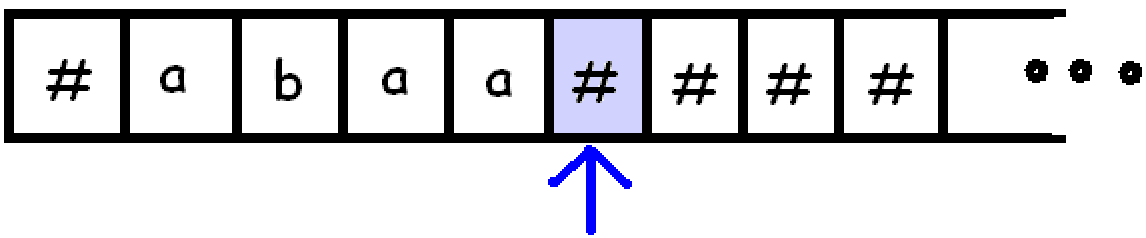
We can easily draw a machine schema for a TM which decides $L = L_1 \cup L_2$.

Pseudo code for algorithm:

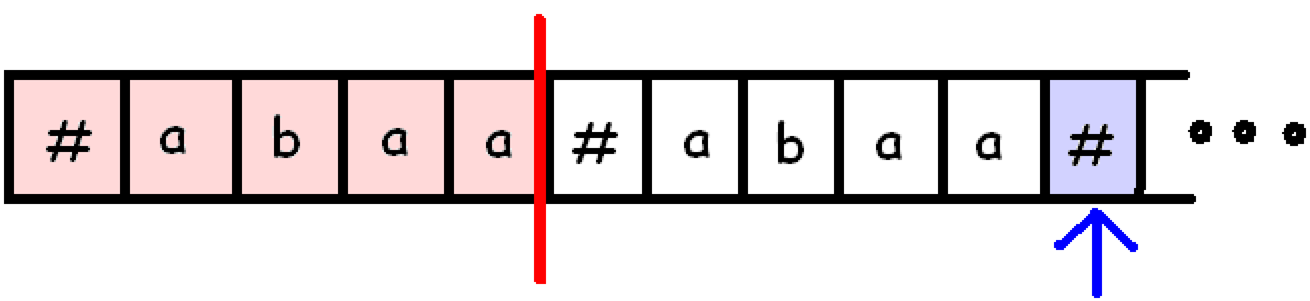
1. Run the copy machine C.
2. Run M_1 on the right hand copy of w.
3. If the answer is Y (yes) clean up the tape by erasing the first copy of w and answer Y.
4. If the answer is N, erase the N and run M_2 on the original copy of w halting with the answer it provides.

Why does this work?

If the TM M_1 does not hang on any inputs:



Then the new machine created does not use the portion of the tape where the original copy of w is stored when running M_1:



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 $L = L_1 \cap L_2$.

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

Output

w2 = d

w1 = abcd

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)