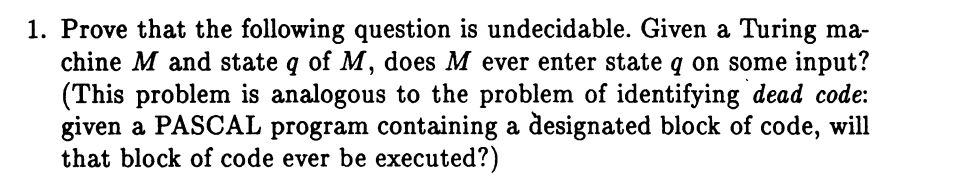


1. Enumeration machines:

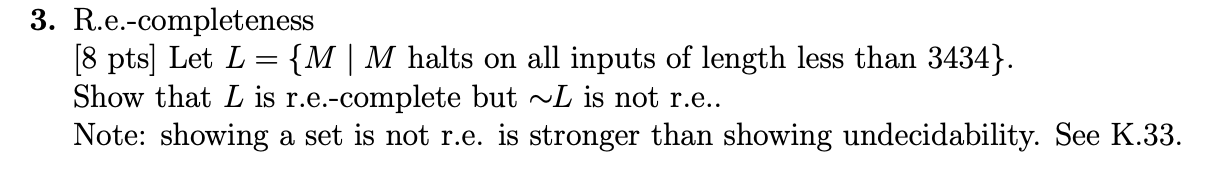
Let L be the given set. If L is finite, it is regular, and thus can be hardwired in a total TM and an 'ordered' enum machine. So assume L is infinite.

Given a total TM M accepting L, we can easily build a enum machine E which enumerates L in increasing order.  
E writes out each string x on its work tape one at a time in the prescribed order, and runs M on x.  
If M accepts, E writes x to its output tape and 'enumerates'.  
If M rejects, E erases its work tape and begins again with the next string.  
As M is total, one of these two events will happen, and E will eventually get to every string.

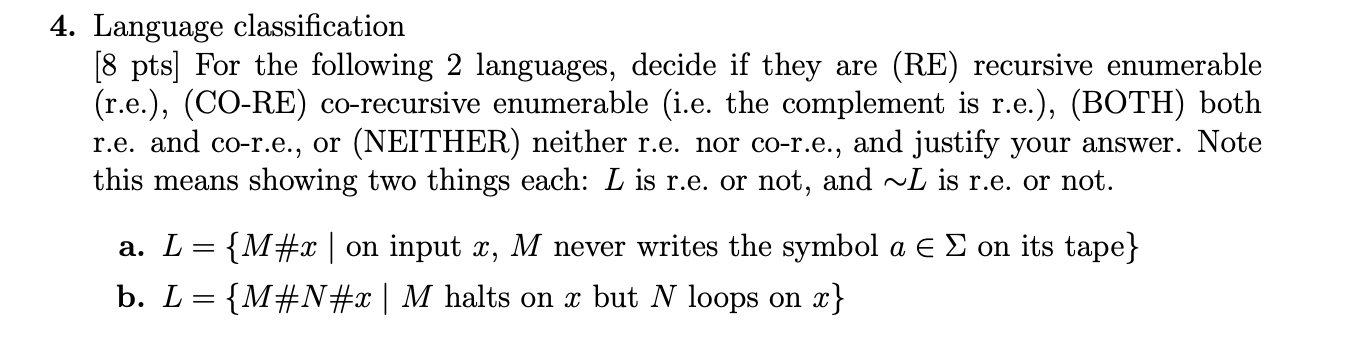
Conversely, given an 'ordered' enum machine E, M can simply run E until a string is enumerated which is either equal to x (accept) or ordered greater than x (reject). The latter must eventually happen as L is infinite.



Suppose N was a total TM accepting this language.  
Given any TM M with accept state t, feed <M>#<t> into N.  
If M accepts any string, then N accepts.  
If M does not accept any string, then N must reject.  
Hence, we can decide {<M> : M accepts any string at all}, which is undecidable by K.32(g)



r.e.: Enumerate all inputs of length <= 481, timeshare M on each, and accept if M halts on any.  
not co-r.e.: Reduction from HP. On M#x, construct the usual M' which on input y runs M on x and accepts y if M halts. M halts on x => M' halts on all inputs => M' halts on all inputs of length <= 481. M loops on x => M' loops on all inputs => M' loops on some input of length <= 481.  
Reduction from HP also shows r.e.-complete, since HP is r.e.-hard, so this language must be too.



a) L = {M#x : on input x, M never writes the symbol a on its tape}

CO-RE.

Complement is r.e.: simulate M on x and accept if M ever writes a.  
Not r.e.: Given M and x for the halting problem, build N which on input y simulates  
M on x, and if M halts, N writes a to its tape and accepts. Then M loops on x if and  
only if N never writes a to its tape on input , i.e. M#x ∈∼ HP ⇐⇒ N#e ∈ L, so  
we have reduced ∼ HP to L, and as ∼ HP is not r.e., neither is L.

b) L = {M#N#x : M halts on x but N loops on x}

NEITHER.

Not co-r.e.: Given M and x for the halting problem, let N be the trivial machine  
that always loops. Then M#N#x ∈ L ⇐⇒ M#x ∈ HP, so we have reduced HP  
to L, ergo L is not co-r.e..  
Not r.e.: Given M and x for the looping problem (∼ HP), let N be the trivial machine that  
always halts. Then N#M#x ∈ L ⇐⇒ M#x ∈∼ HP, so we have reduced ∼ HP to  
L, ergo L is not r.e., and hence neither r.e. nor co-r.e..