Lecture 25

Homework #25: 4.1.8, 4.1.9, 4.1.10

We'd like to be able to create more complex Turing Machines than those we've seen so far.

Let's think about creating a library of simple/useful Turing Machines. We can then use these as subroutines in bigger, more interesting, more complex Turing Machines.

We begin by defining some basic (useful) machines. We can then start talking about how we can combine them.

Most basic: **Symbol-writing** and **head-moving machines**.

1. There are $|\Sigma|$ -1 symbol-writing machines. Each writes a specified symbol in the current tape square and then halts. We do not have a symbol-writing machine for the special left-end-of-tape marker.

   The TM to write the symbol a is $M_a = (K, \Sigma, \delta, s, H)$,
   where $\delta(s,b) = (h,a)$ for all $b \in \Sigma - \{ > \}$.
   $K = \{s, h\}$, $H = \{h\}$

   We'll call M_a "a".

2. There are two basic head moving machines, one that goes left and halts and another that goes right and halts.

   $M_{\leftarrow} = (K, \Sigma, \delta, s, H)$,
   where $\delta(s,b) = (h, \leftarrow)$ for all $b \in \Sigma - \{ > \}$.
   $K = \{s, h\}$, $H = \{h\}$

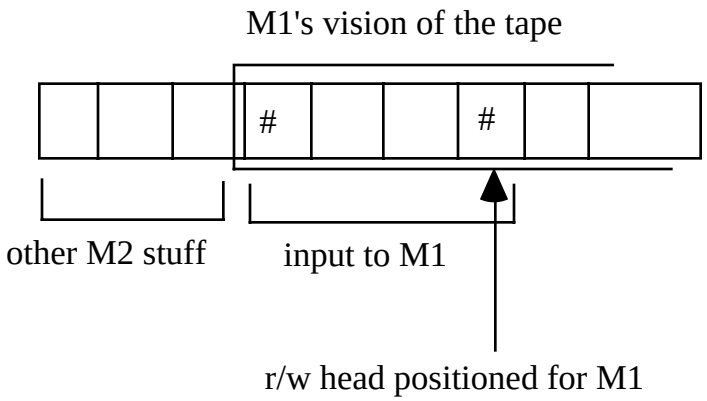
   We'll call $M_{\leftarrow}$ "L" and $M_{\rightarrow}$ "R".

Now that we have some very simple useful machines to work with, we can think about how to combine them to make bigger Turing Machines. That is, we can think about the mechanics of doing the combining.

• Let M_1 be a TM.

• Let M_2 be another TM that "calls" M_1 as a subroutine.

Operation:
• M_2 does some stuff, including making sure that input is ready for M_1 (can think of it as sending the right parameters to M_1).



• Now M_1 does its thing.

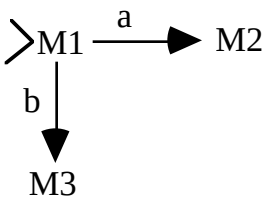
• Control goes back to M_2.

* One more thing we need to assume is that the machines we're combining don't share states. But this is easy because states can be renamed.

Now we can consider the way we'll describe combined machines:

> $M_1 \rightarrow M_2$

means start M_1 and run it till it halts; then run M_2 from the position where you just finished.



means run M1. When M1 finishes, if the symbol being scanned is a, then run M2; if it's b, run M3.

Let M1 = (K1, Σ, δ1, s1, H1)
M2 = (K2, Σ, δ2, s2, H2)
M3 = (K3, Σ, δ3, s3, H3)

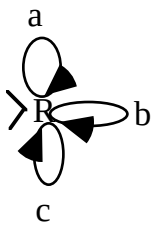
M, the composite TM, is (K, Σ, δ, s, H), where

K = K1 $\cup$ K2 $\cup$ K3
s = s1
H = H2 $\cup$ H3
For each $\sigma \in \Sigma$ and $q \in$ K-H, $\delta(q,\sigma)$ is defined as follows:
(1) $\delta(q,\sigma) = \delta1(q,\sigma)$, if $q \in$ K1-H1.
(2) $\delta(q,\sigma) = \delta2(q,\sigma)$, if $q \in$ K2-H2.
(3) $\delta(q,\sigma) = \delta3(q,\sigma)$, if $q \in$ K3-H3.
(4) $\delta(q,a) = (s2,a)$, if $q \in$ H1.
(5) $\delta(q,b) = (s3,b)$, if $q \in$ H1.
(6) $\delta(q,b) = (h,b)$, otherwise, $h \in$ H.

And now for some **more** examples of **interesting basic machines**:

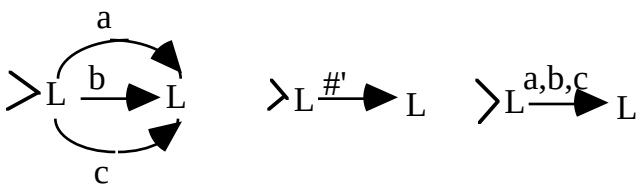
Ex. Move right until blank: $R_{\#}$

if Σ = {a, b, c}, then $R_{\#}$ is



Note that this will always move to the right at least once.

And here are more ways to represent complex machines. The first three are different ways to do the following: move left; if the symbol being scanned is not #, then move left again. Note that I'm using #' to denote "not #"



The last two are different ways to say "move left twice."

And now some more TMs that can be used as building blocks for more complex machines:

Move right/left to x
[We've already seen $R_{\#}$]

finds 1st # to left of current position

finds 1st non-blank to right of current position

finds 1st non-blank to left of current position