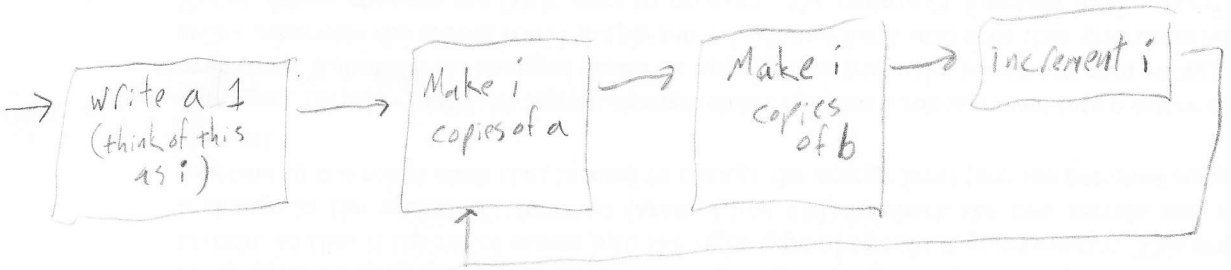
Page 272

(5) $L = \{a^n b^n : n \geq 1\}$



(6) Let S_1 & S_2 be countable sets. Then there are algorithms and T.M.s to enumerate the elements of S_1 and S_2 (call the TM's T_1 & T_2).

For $S_1 \cup S_2$,

Create a new TM T_U that alternates the enumeration done by T_1 and T_2, I.e. it prints an element of S_1, then S_2, then the next element of S_1, next element of T_2, etc.

For $S_1 \times S_2$

Use the same construct that the text used for enumerating the rationals (figure 10.17) but using the elements of $S_1 \& S_2$ in the place of the integers.

Page 284

(3) Let L be finite

L^+ is the set of all strings consisting of 1 or more elements of L concatenated together.

Let the elements of L be $l_1, l_2, l_3, \ldots l_n$. Where $|L| = n$, i.e. L has n elements

Enumerate (list) L^+ as follow.

For $i = 1$ to ∞

list all possible permutations of i elements of L (in some order)

Because L is finite all permutations of i elements is finite, so that step will terminate for all values of i and go onto the next value of i.

(Note if L were infinite this process would "get stuck" at $i = 2$ and never do combinations of 3 elements.)

(6) Answer in text.

(1) Some examples for the languages of interest:

Finite Languages: any set containing a finite # of strings;
$\{a\}$, $\{aa\}$, $\{aaa, bb\}$, etc.

Regular Languages; any language you can describe w/ a regular expression;
a^*, $(a+b)^*$, $(aa+bb)a^*$, etc.

Deterministic Context Free Languages: wcw^R where $w \in \{a,b\}^*$
note that the 'c' identifies the middle of the palandrome, making it determistic.

Non-deterministic Context free Languages; $a^n b^n$ ww^R

Recursive Languages: ww, $a^n b^n c^n$...

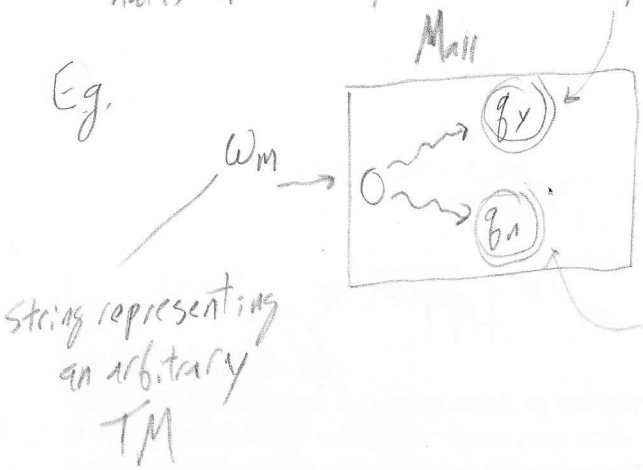
Recursively Enumerable Languages: Languages like the set of strings defining TM's that halt on the string 000 (or any other string and most other TM defined sets).

Remember that these Languages are "nested" so all the finite languages are also regular, all the regular languages are Deterministic context free, etc.

Page 307

(5) Proof by Contradiction. If there is an algorithm, then by the Turing Thesis there is a T.M. to execute the algorithm

Assume there is a Turing Machine, M_{all} that decides whether or not an arbitrary T.M halts on all input



For any TM M' we can create a new TM $\hat{M}$ which erases the input (clearly doable) and writes the string w (also doable) then runs like M', but on w.

I.e. $\hat{M}$ is M', but always applied to w.

Now give $\hat{M}$ to Mall.

If Mall says $\hat{M}$ always halts then we know M' halts on w

If Mall says $\hat{M}$ doesn't halt then we know M' doesn't halt on w.

So, this solves the halting problem, which we know is impossible, therefore ~~our~~ the assumption that Mall exists must be false.

Page 307

(7) Show that there is no algorithm for deciding if any 2 TMs M_1 and M_2 accept the same language.

Consider the case where M_1 accepts some string w.

To determine that M_2 accepts the same language requires knowing that M_2 will halt and accept w as well. But that requires knowing that M_2 will halt, which is not possible.