Lecture 31

Homework #31: 5.3.2, 5.3.3
Note that you already did the exercise described in 5.3.3 in the context of learning about nondeterministic Turing Machines. Now do it again, but make your constructions deterministic. You needn't do Kleene star, as we will do it as an example in class.

Church's Thesis (Church-Turing Thesis)

No computational procedure will be considered an algorithm unless it can be presented as a Turing Machine (that halts on all inputs).

Notes:

- a thesis, not a theorem.
- possibility that someone could find a machine that performs "finite labor at each step" and is provably capable of doing computations that a TM can't do
- historical note -- in 1936 Alonzo Church formulated the thesis about what it means to compute. He believed that he had captured the notion precisely by means of a formal system called the **λ-calculus** (on which Lisp is based).

another paper around the same time made a similar claim about **general recursive functions** (a special class of functions).

third claimant was Turing with his **Turing Machines**.

⇒ all three turned out to be computationally the same.

other models are also equivalent.

Other models:

• general class of grammars that are as powerful as Turing Machines • μ-recursive functions - idea is that certain very simple functions should be regarded as computable; and certain ways of combining computable functions yield more computable functions.

And now we begin to talk about what can and what can't be performed by an algorithm.

We'll do this by considering what things can and cannot be done by a Turing Machine.

Important: what will be most important to us is the notion of decidability – an algorithm will not be considered to be an algorithm unless it actually halts on its input.

What we'll discover is that acceptance (i.e., semi-decidability) is easy to achieve, but that decidability in general is not – so let's begin by reminding ourselves of the relationship between decidability and semi-decidability, and about some closure properties for these classes of languages.

Thm. If a language is recursive, then it is recursively enumerable.

Proof done earlier. (But let's consider a diagrammatic way of expressing this – will be useful for many of our proofs in the next sections.)

Thm. If a language is recursive, then its complement is recursive.

Thm. The class of recursively enumerable languages is closed under union, concatenation, Kleene *, and intersection.

[See homework for proof.]

Thm. The class of recursive languages is closed under union, complementation, intersection, concatenation, and Kleene star.

Recall that this was a homework problem for the section on nondeterministic Turing Machines. Here we consider how to prove this without the power of nondeterminism.

Proof of Kleene Star.

Suppose M decides L. Build M' as follows:

Given input w, M' lexicographically generates all words of length $\leq |w|$.

For each such word w_i, w_i is run through M to see if $w_i \in L$.

If $w_i \in L$, then w_i is added to the end of a tape which contains all words generated so far that are in L.

So now we have a spare tape containing $\$w_1\#w_2\#...w_k$. i.e., all words in L of length $\leq |w|$

We can now check all combinations of those words that are of length ≤ |w| to determine whether any of the combinations is w. There are a finite number of such combinations (assuming we don't include the empty string).

What about the flip-side of the first theorem? Does acceptability imply decidability? That is, if a language is recursively enumerable, does that imply that it is recursive?

⇒ No, because you need to determine halting. But this is just the Halting Problem! (Recall proof from 1st day of class.)

And what about the following: Does acceptability imply that the complement of a language is acceptable? No, because if it were, then it would be decidable!

Let's look at another theorem.

Thm. A language is recursive iff both it and its complement are recursively enumerable.

Pf.

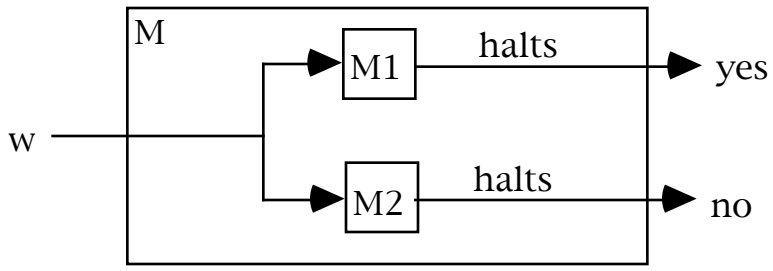
(⇒) Let L be a recursive language. Then L is r.e. by the earlier theorem.

$\bar{L}$ is also recursive (by thm), and so is also r.e.

(⇐) Suppose that L is accepted by M1.

Suppose that $\bar{L}$ is accepted by M2.

Construct a new Turing Machine M that decides L as follows:



What might M look like at the next level of detail?

separate tapes for M1 and M2.
jump back and forth between the two simulations.

Given the theorems we've seen so far, what can we say about L and L(bar)? Either:

1) L and $\bar{L}$ are decidable (recursive).

2) neither L nor $\bar{L}$ is acceptable (r.e.).

3) one of L and $\bar{L}$ is acceptable, but not decidable; the other is not acceptable.
The Halting Problem is an example of this third group.

Some terminology:
Decidable = Recursive
not decidable = unDecidable = unsolvable by an algorithm.