CSE547T Class 12

Jeremy Buhler

February 27, 2017

> A man provided with paper, pencil, and rubber, and subject to strict discipline,
> is in effect a universal machine.
> – Alan Turing, 1948

1 Recursively Enumerable Homesick Blues

Now that we have some experience of Turing machines, let's start to characterize the languages they can accept.

- The interesting thing about TMs is that they have three-valued outcomes.
- Given an input, they can accept, reject, *or* run forever.
- How important is that third option?
- Consider a TM as a black box that takes an input and may eventually accept or reject it.
- You cannot tell what the TM is doing – merely wait for it to halt.
- If a TM fails to halt on your input, you will be waiting forever with no answer.
- Hence, it is useful to make a distinction between TM's that eventually halt with an answer for *every* input, and TM's that sometimes leave you hanging.

This investigation goes to the question of "what is an algorithm?"

- An algorithm is a formal procedure for solving a problem.
- It needs to work for *every* problem instance.
- In particular, a *decision algorithm*, that returns a boolean answer given its input, should be able to answer correctly for every problem instance.
- If you have a procedure that sometimes answers but sometimes runs forever without giving an answer, that is *not* an algorithm.
- If, for a language L, no TM can be built that both accepts L *and* rejects $\Sigma^* - L$, then we are justified saying that *there is no TM-implementable algorithm to recognize* L.
- (Note that this is a stronger criterion than mere acceptance of L.)

- TM's are as powerful as unbounded real computers (see homework), so this is the same as saying that certain decision problems cannot be solved by any algorithm on a computer!
- In fact, the **Church-Turing Thesis** claims more: "Every reasonable manual or automated procedure for algorithmic computation is equivalent to a Turing machine."
- (Not proven, since a "reasonable" procedure is not well-defined, but true for all examples we've ever come up with so far.)

We are going to formally demonstrate that there are natural decision problems for which no algorithm exists on a TM.

2 Formal Defns

- A language $L \subseteq \Sigma^*$ is said to be *recursively enumerable* (RE) if there exists a TM M that accepts $x \in \Sigma^*$ iff $x \in L$.
- (Note that, if $x \notin L$, M might run forever on input x)
- L is said to be *recursive* if there exists a TM M' s.t., for every $x \in \Sigma^*$, M' accepts x if $x \in L$ and rejects x otherwise.
- M' is said to *decide* L. L is also said to be *decidable*.
- For example, language $L = \{ss \mid s \in \Sigma^*\}$ is decidable – we built a TM that accepts $x \in L$ and rejects $x \in \Sigma^* - L$.

What can we say about the relationship between RE and recursive languages?

- **Fact**: every recursive language is RE. (Trivial from definitions)
- **Fact**: if L is recursive, so is $\overline{L} = \Sigma^* - L$.
- **Pf**: Let M be a TM deciding L.
- Construct a new TM $\overline{M}$ by copying M and interchanging all occurrences of h_a and h_r.
- $\overline{M}$ accepts iff M rejects, so $L(\overline{M}) = \overline{L(M)} = \overline{L}$.
- Since M always halts, so too does $\overline{M}$. QED
- **Fact**: if both L and $\overline{L}$ are RE, then L is recursive.
- **Pf**: Let M be a TM accepting L, and let $\overline{M}$ be a TM accepting $\overline{L}$.
- I claim that there exists a TM $\tilde{M}$ that decides L.
- *Construction*: $\tilde{M}$ contains two tapes.
- Given an input x, copy the first tape to the second tape.

- Then, simulate M on x on the first tape, and simultaneously simulate $\overline{M}$ on x on the second tape.
- (You can think of executing one move of M, then one of $\overline{M}$, then one of M, and so forth.)
- Now either $x \in L = L(M)$ or $x \in \overline{L} = L(\overline{M})$.
- Hence, at least one of M and $\overline{M}$ will accept x in finite time.
- If M accepts, $\tilde{M}$ accepts; if $\overline{M}$ accepts, $\tilde{M}$ rejects. QED

3 Why "Enumerable?"

What does enumerability have to do with anything?

- TMs can do more than accept or reject a language; they can *produce output* by writing strings onto their tapes.
- We have seen an example of this with our TM that could count in base k.
- We will consider a special kind of output: enumerating a language L.
- **Defn**: Let M be a k-tape TM, with $k \geq 1$.
- M *enumerates* the language L if, starting with a blank tape 1, it behaves as follows:
- The head on the first tape of M (the "output tape") moves only to the right.
- The output tape is *write-once*: no non-blank symbol, once written, is ever overwritten or erased.
- for every $x \in L$, after some number of moves of M, output tape contains the string

  $$x_1 \# x_2 \# \ldots \# x_n \# x \#$$

  where

  1. # is a fixed separator symbol not in Σ^*;
  2. $x_1 \ldots x_n$ are elements of L, all distinct from x;
  3. no element of L is repeated on the tape.
- (This is a fancy way to say that M enumerates the elements of L on tape 1 in some order.)

I repeat, what does enumerability have to do with anything?

- **Lemma**: A language L is RE iff there exists a TM that enumerates L.
- **Pf**: We have two directions to prove.
- ($\leftarrow$): Suppose M is a TM enumerating L.
- We can construct M' accepting L as follows.

- On input x, M' simulates operation of M.
- Each time M writes # to output tape, M' checks whether the string between the last two #'s is x.
- If so, M' accepts x.
- By defn, if $x \in L$, M will eventually write x to output tape, so M' indeed accepts x.
- Conversely, if $x \notin L$, M will never write x to output tape (in fact, will run for ever), and so M' will never accept x.
- ($\rightarrow$): Suppose M is a TM accepting L.
- We'd like to simply feed each $x \in \Sigma^*$ successively to M and add it to the output tape if M accepts.
- However, we can't do this, since M may run forever on some x's.
- Instead, we will *interleave* checking of all x, using similar trick to NTM construction.
- Fix a total ordering on input alphabet Σ.
- The *canonical ordering* of strings in Σ^* orders smaller strings before larger strings, and orders strings of the same size lexicographically.
- **Example**: $\Sigma = \{0, 1\}$, $0 < 1$
- Order is: ε, 0, 1, 00, 01, 10, 11, 000, 001, 010, ...
- Let $z_1, z_2, z_3, \ldots$ be the strings of Σ^* in canonical order.
- M' runs in a succession of phases.
- In its ith phase, it enumerates all strings $z_1 \ldots z_i$ in succession using our counting algorithm.
- For each string z_k, M' simulates M for $i - k + 1$ steps on input z_k.
- If during phase i, M accepts some z_k in *precisely* $i - k + 1$ steps, M' appends z_k to its output tape.
- (M' needs only four tapes: one to track i, one to enumerate $z_1 \ldots z_i$ during phase i, one to simulate M, and its output tape.)
- Observe that every $x \in L$ is eventually reached in canonical order.
- Moreover, for each $x \in L$, there is a first phase i_x in which M accepts x.
- Hence, x is written to output tape exactly once (in phase i_x).
- Moreover, no string not in L is ever written to output.
- Conclude that M' enumerates L.

Is there an equivalent idea of enumeration for recursive languages?

- **Lemma**: a language L is recursive iff there exists a TM that enumerates L *in canonical order*.
- **Pf**: again, two directions to prove.
- ($\rightarrow$) Since L is recursive, let M be a TM deciding L.
- Construct an enumerator M' as follows.
- M' internally enumerates Σ^* in canonical order.
- For each enumerated string z, M' simulates M on z.
- If M accepts z, M' appends z to output tape.
- Otherwise, M rejects z in finitely many moves, and M' continues without writing z to output.
- Any $x \in L$ eventually appears in output, in the order that it was tested (i.e. canonical order).
- Moreover, no string not in L is ever written to output.
- Hence, M' enumerates L.
- ($\leftarrow$) let M be a TM enumerating L in canonical order.
- Construct a TM M' to decide L as follows.
- On input x, M' simulates the enumerator M.
- Each time M writes a new string z to its output tape, M' compares x to z.
- If $x = z$, then $x \in L$, and so M' accepts x.
- If $x < z$ in canonical order, then $x \notin L$ (else it would have been emitted before z), so M' rejects x.
- For any x, M' will eventually discover that $x \in L$ or $x \notin L$, so M' decides L.

How is enumeration related to algorithmic computation?

- An enumerator M for an arbitrary RE language L will print any given $x \in L$ in finite time.
- However, no way to tell that M will *never* print a given string z.
- OTOH, an enumerator M for a recursive L prints output in a known order.
- Hence, we will know in finite time for any z whether $z \in L$ or not.