CSE491/596 Lecture Wed. 9/30: Computations and Computability

An *instantaneous description* (ID), also called a *configuration*, of a Turing machine M specifies:

1. The current internal state q of M.
2. The contents $\vec{w} = w_1, \dots, w_k$ of the k tapes, such that all else on the tapes is blank.
3. The positions $\vec{h} = h_1, \dots, h_k$ of the heads on those tapes.

We can write $I = \langle q, \vec{w}, \vec{h} \rangle$ to denote an ID.

Write $I \vdash_M J$ if there is an instruction in δ that when executed in ID I produces ID J. For $r \geq 2$, write $I \vdash_M^r K$ if there is an ID J such that $I \vdash_M J$ and $J \vdash_M^{r-1} K$. Also write $I \vdash_M^0 I$ for all I and $I \vdash_M^* J$ if $I \vdash_M^r J$ for some r. These notions apply to nondeterministic TMs as well as DTMs.

For a single-tape TM and input x, the initial ID can be written $I_0(x) = \langle s, x, 1 \rangle$ (if we number the cells from 1) or $I_0(x) = \langle s, \wedge x, 1 \rangle$ (if we use the convention of an initial $\wedge$ in cell 0 but still number x from 1 and start up scanning the first bit rather than the $\wedge$). Yet another convention is to start in the ID $\langle s, \wedge x \$, 1 \rangle$ with a right-endmarker $\$$ too. A 1-tape TM is a *linear bounded automaton* (LBA) if δ is syntactically coded so that the only instructions involving the endmarlers have the form $(p, \wedge/\wedge, R, q)$ or $(p, \$/\$, L, q)$, so that the head always stays between $\wedge$ and $\$$.

For k-tape TMs we could use ϵ's to stand for the other tapes being blank and 1's for the other head positions, but we won't go any further into details of IDs until we hit complexity theory. An *accepting ID* has q_{acc} as its state and a *rejecting ID* has q_{rej}. Now we can formally define the language of a TM (NTMs too):

Definition: $L(M) = \left\{ x : I_0(x) \vdash_M^* I_f \textit{ for some accepting ID } I_f \right\}$.

Here is an ad-hoc definition that helps with some technical things including making I_f unique:

Definition: A Turing machine $M = (Q, \Sigma, \Gamma, \delta, \_, s, F)$ does "good housekeeping" if:

1. $F = \{q_{acc}\}$ and q_{rej} is the only other halting state;
2. M never writes the blank _ between two chars that are not blank, on any tape;
3. Whenever M "wants to accept", it first blanks out all of its tapes---it can find the nonblank extremities because there are no internal blanks and then blank them in one right-to-left pass. Then it writes a single 1 on tape 1 and accepts, so $I_f = \langle q_{acc}, 1, 1 \rangle$.
4. Similarly, in a rejecting condition, it blanks all tapes, writes 0, and ends in $I_r = \langle q_{rej}, 0, 1 \rangle$.

Again, we can vary the details but the ideas remain helpful. The main variation is that if we consider the input tape to be read-only and one-way (no L moves on tape 1), then M does not blank the input x but leaves it alone, ends on the blank to its right, and writes the final 1 or 0 on another tape, which could be designated the *output tape*. More generally, we can code such a machine to compute a function $f(x) = y$, with everything except x on the input tape and y on the output tape blanked out, and the output tape head scanning the first bit of y. Such an M, especially when it is deterministic, is called a *transducer*.

It is a useful self-study exercise to show that every TM (using the more-liberal definitions in some other texts or implemented by the "Turing Kit" program) can be converted into an equivalent one that does good housekeeping and is basically no less efficient. Then you may assume a given M does good housekeeping to begin with. For instance, to say that a DTM M on an input x **halts**, written $\boldsymbol{M(x)\downarrow}$, we can specify this means $I_0(x) \vdash_M^* I_f \vee I_0(x) \vdash I_r$. Else, we write $M(x)\uparrow$ and can say the computation of M on x **diverges**. A DTM M is **total** if for all $x \in \Sigma^*$, $M(x)\downarrow$.

Now we can fully appreciate the key definitions:

Definition: For any language A over an alphabet Σ, or function $f: \Sigma^* \to \Sigma^*$:

- A is **computably enumerable** (**c.e.**) if there is a TM M such that $L(M) = A$.
  - Synonyms: **recursively enumerable** (**r.e.**), *Turing-acceptable*.
- A is **decidable** if there is a total DTM M such that $L(M) = A$.
  - Synonym: **recursive**. (Avoid the term "recognizable"---it is used both ways).
- f is **computable** if there is a transducer M that computes $f(x)$ for all $x \in \Sigma^*$.
  - Note that writing $f: \Sigma^* \to \Sigma^*$ standardly means that the domain of f is all of Σ^*, so any M computing f must be total. But we often say f is **total computable** to remind about this and clarify when we are not allowing f to be a *partial function*. Other synonyms: **recursive** function, **total recursive**.

Here is a helpful little proposition that helps in understanding these concepts. Recall that with the Myhill-Nerode theorem, we have been writing $L(x)$ as if the langauge L is a Boolean-valued function. We can distinguish the function from L by calling it $\chi_L(x)$, where χ is the Greek letter chi to stand for characteristic function.

Proposition: A language L is decidable if and only if χ_L is a total computable function.

The proof is "by good housekeeping." The important contrast is that when L is only known to be c.e., then χ_L need not be computable: on some $x \notin L$, the machine might never halt. For a pivotal example, consider the language of the "3n+1 Problem" shown in the opening week:

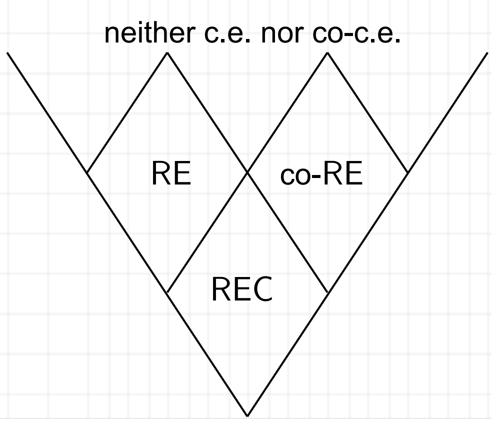
$L = \left\{x \in \mathbb{N}^+ : (\exists r)\, f^r(x) = 1\right\}$, where $f(x) = \mathit{if}\ x\ \mathit{is\ even\ then}\ x/2\ \mathit{else}\ 3x+1$.

We can regard binary numbers and binary strings as interchangeable, in various ways. One way specific to $\mathbb{N}^+$, meaning the positive natural numbers, is to delete the leading 1 in standard binary notation, which gives a 1-to-1 correspondence to a language L' over $\{0,1\}^*$. The demo showed a particular TM M that ends on a single 1 whenever $x \in L$ but does not halt otherwise.

- The *Collatz conjecture* says that L equals all of $\mathbb{N}^+$, likewise $L' = \Sigma^*$. Then M is actually total and that makes L "trivially" decidable.
- But all we *know* at this point is that L is computably enumerable. The M shown in the demo is, I believe, the *tiniest* program that no one has been able to prove is total.

If a language is not decidable, it is called **undecidable**. This includes c.e. languages that are not decidable. Starting next week we will cover techniques for showing that languages are undecidable. It helps to have notation to map out *classes* of languages:

- The class of c.e. languages is denoted (only) by RE.
- The class of decidable languages is denoted by REC (occasionally, DEC).
- The class of regular languages is denoted by REG. The facts that every regular language is decidable, and some decidable languages are not regular (such as $\{a^n\, b^n\}$) can be neatly captured by writing $\text{REG} \subset \text{REC}$.
- The classes of langauges recognized by deterministic and nondeterministic PDAs are denoted by DCFL and CFL, respectively, as we have seen.
- The classes of languages recognized by deterministic and nondeterministic LBAs are denoted by DLBA and NLBA, respectively.
- The progression $\text{REG} \subset \text{CFL} \subset \text{NLBA} \subset \text{RE}$ is called the *Chomsky Hierarchy* after Noam Chomsky, who characterized these classes via notions of *grammars*. One can insert DCFL and REC and keep a proper progression, but the corresponding grammar notions are "wonky" in the former case and *nonexistent* in the latter.
- However, although $\text{CFL} \subset \text{DLBA}$, whether DLBA is properly contained in NLBA is unknown. It is rather like the P versus NP question. We will not address grammars but we will later see that DLBA and NLBA equal deterministic and nondeterministic space, respectively.
- For any class **C**, the complements of languages in C form the class co-C. Note that since the complement of a regular language is always regular, we have $co\text{-REG} = \text{REG}$; the co- does not mean "not regular" here.
- We will concentrate on REC, RE, co-RE, and "neither c.e. nor co-c.e." for the two coming weeks. Here is a little roadmap:



This diagram conveys some extra information:
◎ REC is closed under complements,
◎ $\text{RE} \cap \text{co}-\text{RE} = \text{REC}$, and
◎ All three classes are closed downward under *computable many-one/mapping reductions*.
We will prove these after we establish the equivalence between Turing machines and high-level programming languages.