Computation, Information, and Intelligence (COMS/ENGRI/INFO/COGST 172), Fall 2005
11/23/05: **Lecture 37 aid — Limits on computability**

Agenda: discover some limits on the powers of computers (and by extension, people, or at least computer scientists) by:

- considering the machine/program duality from the beginning of class more formally, thus seeing that TMs are actually quite powerful, and
- proving (following Kleinberg and Papadimitrious's presentation) that the halting function is not computable, thus showing that there are fundamental limits to our understanding of computational processes.

Announcements: Something we forgot to add to the solutions for Homework Five: in the Earley's algorithm question, an important observation one should draw is that when multiple parses share some substructure (e.g., the sub-tree for the prepositional phrase "on Tuesday"), that substructure is only computed *once in total*, not once for each parse it appears in. This "packing" of shared substructure is the reason that Earley's algorithm is so efficient.
Follow-ups:

- Our restriction of consideration to functions only on the non-negative integers is a simplification. It is not too difficult to see how Turing machines could compute functions whose input and output involve numbers with finite decimal expansions, or complex items such as sentences and parses. What is lethally problematic is input and/or output involving *infinite* sequences, since by definition a TM that computes a function halts its operations in a finite amount of time, whereas the time it would take to just write down a number with an infinite number of digits would blow past the finite-time limit. (There are deeper mathematical problems that arise as well, but discussion of these are beyond the scope of this course.)
- The date ("11/19/05") on last lecture's handout was, of course, wrong. I have no idea what happened.

I. Enumeration of "A-machines"

We denote by $M_1, M_2, \ldots$ an infinite list (with no repeats) of *all* TMs that take only sequences of A's as input.[1] The list is such that *given a (suitably encoded) non-negative integer i, it is possible to recover the program for M_i* (i.e., there exists a Turing machine that does the job).

The sequences of A's are intended to be encodings of non-negative integers.[2] Thus, all computable functions whose domain is the non-negative integers[3] are represented in the list, but there are also many TMs on the list that don't compute such functions.

[1] This is a modification of what was given on the previous handout, which should cause no confusion since we didn't actually discuss that point last time. The technical problem with requiring that only A's be *output* is that it would be best to allow TMs to use different "scratch" symbols, and determining whether a TM prints as part of its output a symbol other than A turns out, for general TMs, to be impossible (this result is in fact a consequence of the non-computability of the halting function).

[2] This encoding is non-traditional; we have chosen it in order to remove a layer of self-reference from our proof.

[3] We technically should put some restrictions on the range, too — essentially, that the elements of the range be representable as finite sequences over a finite symbol set — but we'll ignore this point.

(OVER)

II. The halting function

$$h(M_i, j) = \begin{cases} 1 \text{ (yes)}, & \text{if } M_i \text{ would halt given } j \text{ A's as input} \\ 0 \text{ (no)} & \text{if } M_i \text{ would not halt given } j \text{ A's as input} \end{cases}$$

Note that by "halting", we mean "would halt in a finite number of steps".

III. The evil machine X Suppose a termination detector D exists. Then we could construct a Turing machine X that, when given j A's as input,

(a) recovers the program for M_j;
(b) runs D to determine the value (0 or 1) of $h(M_j, j)$; and
(c1) if that value is "1", enters an artificial infinite loop
(c2) if that value is "0", sets its output to 1 and halts

Common point of confusion: we can show that X, and therefore D, does not exist, meaning that the halting function is not computable. But the function does *exist* (as much as any mathematical function, like $f(x) = x^2$, can be said to exist), and is well-defined, and so on. So for every M_i and j, there is a value, either 0 or 1, to $h(M_i, j)$; it's just that *we have no general way to determine what that value is for arbitrary M_i and j.*