D) Enumerators

As we saw, a Turing-recognizable language is called in some text-books a recursively enumerable language. The term comes from a variant of a Turing machine called an <u>enumerator</u>. Loosely, an enumerator is a Turing machine with an attached printer.

The enumerator prints out the language L it accepts as a sequence of strings. Note that the enumerator can print out the strings of the language in any order and possibly with repetitions.

<u>Theorem</u> A language L is Turing-recognizable $\Longleftrightarrow$ some enumerator enumerates (outputs) L.

<u>Proof</u> "$\Leftarrow$" Let E be the enumerator. We construct the following Turing machine M:

M = on input w

1. Run E. Every time that E outputs a string, compare it with w.
2. If w ever appears in the output of E, accept w.

Thus, M accepts exactly those strings that appear on E's list and no others, hence exactly L.

"$\Longrightarrow$" Let M be a Turing machine that recognizes L. We would like to construct an enumerator E that outputs L. Let A be the alphabet of L, i.e. $L \subset A^*$. In the unit on countability, we proved A^* is countably infinite (note that the alphabet A is always assumed to be finite), so A^* has an enumeration as a sequence $A^* = \{w_1, w_2, \dots\}$

E = Ignore the input

1. Repeat the following for $i = 1, 2, 3, \dots$
2. Run M for i steps on each input $w_1, w_2, \dots, w_i$
3. If any computations accept, print out the corresponding w_j.

Every string accepted by M will eventually appear on the list of E, and once it does, it will appear infinitely many times because M runs from the beginning on each string for each repetition of step 1. Note that each string accepted by M is accepted in some finite number of steps, say k steps, so this string will be printed on E's list for every $i \geq k$.

(q.e.d.)

Moral of the story

The single-tape Turing machine we first introduced is as powerful as any variants we can think of.

Algorithms

Task Use Hilbert's 10^{th} problem to give an example of something that is Turing-recognizable but not Turing-decidable.

We saw that the Continuum Hypothesis of Cantor was the 1^{st} of Hilbert's 23 problems in 1900 at the International Congress of Mathematicians.

Hilbert's 10^{th} problem

Find a procedure that tests whether a polynomial in several variables with integer coefficients has integer roots.

Example $P(x,y) = 2x^2 - xy - y^2$ is a polynomial in 2 variables (x and y) with integer coefficients $(2,-1,-1)$ that has integer roots

$P(1,1) = 2\cdot 1^2 - 1\cdot 1 - 1^2 = 0$ so $x = 1 = y$, $1 \in \mathbb{Z}$ is a solution.

Hilbert's problem asked how to find integer roots via a set procedure. In 1936 independently Alonzo Church invented λ-calculus to define algorithms, while Alan Turing invented Turing machines. Church's definition was shown to be equivalent to Turing's. This equivalence says

$$\boxed{\text{Intuitive notion of algorithms}} = \boxed{\text{Turing machine algorithms}}$$

and is known as the Church-Turing Thesis. It led to the formal definition of an algorithm and eventually to resolving in the negative Hilbert's 10^{th} problem. Using previous work by Martin Davis, Hilary Putnam, and Julia Robinson, Yuri Matijasevič proved in 1970 that there is no algorithm which can decide whether a polynomial has integer roots. As we shall see now, Hilbert's 10^{th} problem is

An example of a problem that is Turing-recognizable but not Turing-decidable. Let $D = \{p \mid p$ is a polynomial with an integer root$\}$.