THEORY OF COMPUTATION
More about Turing machines - 20

Prof. Dan A. Simovici

UMB

Recall the universal partially computable function $\Phi(x, z)$. For a fixed z, $\Phi(x, z)$ is the unary partial function Φ_z computed by the program $\mathcal{P}$ with $\#(\mathcal{P}) = z$.

Definition

The TM (in either quadruples of quintuple form) that computed Φ is the universal TM.

Let $g(x)$ be a partially computable function of one variable and let $z_0 = \#(\mathcal{P})$ be the number of a program that computes g. Then, if $\mathcal{M}$ begins with a configuration

$$\underset{\substack{\uparrow \\ q_1}}{B}\, xBz_0$$

where x and z_0 are written as blocks of 1s, then $\mathcal{M}$ will compute $\Phi(x, z_0)$. Thus, $\mathcal{M}$ can be used to compute any partially computable function of one variable.

$\mathcal{M}$ provides a model of an all-purpose computer where the data and program are stored together in a single memory.

z_0 is the coded version of a program to compute g and x is the input to that program.

Turing anticipated this idea in 1936!

Definition

Given a TM with alphabet $A = \{s_1, \ldots, s_n\}$, a word $u \in A^*$ is accepted by $\mathcal{M}$ if when $\mathcal{M}$ begins with the configuration

$$\begin{array}{l} s_0\, u \\ \uparrow \\ q_1 \end{array}$$

it will eventually halt. The set of all words $u \in A^*$ that $\mathcal{M}$ accepts is called the language accepted by $\mathcal{M}$.

Theorem

A language L is accepted by some TM $\mathcal{M}$ if and only if L is recursively enumerable.

Proof.

Let L be the language accepted by a TM $\mathcal{M}$ with alphabet A, and let $g(x)$ by the unary function on A^* that $\mathcal{M}$ computes. Then, g is a partially computable function and

$$L = \{x \in A^* \mid g(x)\downarrow\},$$

which shows that L is r.e.
Conversely, if L is r.e., there is a partially computable function g such that $L = \{x \in A^* \mid g(x)\downarrow\}$. If $\mathcal{M}$ is a TM that computes g strictly, then $\mathcal{M}$ accepts L. $\square$

Theorem

Let A and $\tilde{A}$ be two alphabets such that $A \subseteq \tilde{A}$. The set $\tilde{A}^* - A^*$ is recursively enumerable.

Proof.

Define the TM $\mathcal{M}$ that halts on all words in $\tilde{A}^* - A^*$. In other words, the machine halts if and only if its tape does not contain any symbol of A; otherwise, that is, the machine encounters a symbol of A, the machine cycles indefinitely. Such a machine is defined by the quadruples

$$qs'Rq$$

for every symbol $s' \in \tilde{A} - A$ and

$$qssq$$

for every symbol $s \in A$. Thus, when $\mathcal{M}$ encounters a symbol of A it enters an infinite cycle. □

Theorem

Let A and $\tilde{A}$ be two alphabets such that $A \subseteq \tilde{A}$, and let L be such that $L \subseteq A^*$. Then, L is a r.e. set on the alphabet A if and only if L is an r.e. on $\tilde{A}$.

Proof.

Let L be a r.e. set on alphabet A and let $\mathcal{M}$ be a TM on alphabet A that accepts L.
We may assume that $\mathcal{M}$ begins by moving right until if finds a blank and then returns to its initial position.
Let $\tilde{\mathcal{M}}$ be the TM obtained from $\mathcal{M}$ by adding the quadruples $qssq$ for each symbol $s \in \tilde{A} - A$ and each state q of $\mathcal{M}$. Thus, $\tilde{\mathcal{M}}$ enters an infinite loop if it encounters a symbol in $\tilde{A} - A$. Since $\tilde{\mathcal{M}}$ has the alphabet $\tilde{A}$ and accepts the language L, it follows that L is a r.e. language on $\tilde{A}$. $\square$

Proof cont'd

Conversely, let L be language over alphabet A that is a r.e. as a language over $\tilde{A}$, and let $\mathcal{M}$ be a TM with alphabet $\tilde{A}$ that accepts L.
Let $g(x)$ be the function on A^* that $\mathcal{M}$ computes. The symbols in $\tilde{A} - A$ serve as markers.
Since $L \subseteq A^*$ we have:

$$L = \{x \in A^* \mid g(x) \downarrow\}.$$

Since $g(x)$ is partially computable, it follows that L is a r.e. language over A.

Corollary

Let A and $\tilde{A}$ be two alphabets such that $A \subseteq \tilde{A}$, and let L be such that $L \subseteq A^*$. Then, L is a recursive language on A if and only if L is a recursive language on $\tilde{A}$.

Proof.

Suppose that L is a recursive language on A. Then, both L and $A^* - L$ are r.e. languages over A and, therefore, they are r.e. languages over $\tilde{A}$.
Since $\tilde{A}^* - L = (\tilde{A}^* - A^*) \cup (A^* - L)$, and $\tilde{A}^* - A^*$ is r.e. by the Theorem on Slide 9, it follows that $\tilde{A}^*$ is r.e. Therefore, L is a recursive language on $\tilde{A}$. □

Proof cont'd

Conversely, let L be a recursive language on $\tilde{A}$. Then both L and $\tilde{A}^* - L$ are r.e. languages on $\tilde{A}$, and therefore, L is a r.e. language over A.
Since $A^* - L = (\tilde{A}^* - L) \cap A^*$, and A^* is obviously r.e. (as a language on A and therefore on $\tilde{A}$) it follows that $A^* - L$ is a r.e. language on $\tilde{A}$ and hence on A. Thus, L is a recursive language on A.

Theorem

A set U of numbers is r.e. if and only if there is a TM $\mathcal{M}$ with alphabet $\{1\}$ that accepts 1^x if and only if $x \in U$.

Proof.

This follows immediately from the previous theorem. $\square$

Example

Let F be a set of program codes

$$F = \{x \mid \Phi_x(y) = 0 \text{ for finitely many } y\}.$$

The set of ys such that $\Phi_x(y) = 0$ is called here the support of Φ_x. The set F is non trivial since some but not all functions have finite support. It is also an index set since if $\Phi_a = \Phi_b$, then $a \in F$ if and only $b \in F$.
Let d be such that $\Phi_d(y) \uparrow$ for all y. Clearly, $d \in F$ because the support of Φ_d is empty, and therefore, finite. There are computable extensions of Φ_d such that these extensions are 0 for infinitely many y. Thus, F is not r.e. by the second Rice theorem.
So what?

The observation of the previous slide is essential for machine learning (my research field).
It implies that there are no algorithms to learn functions in F from the values of the inputs in the finite support. One cannot get around this learning limitation by enumerating functions having finite supports until you find a function that matches the given function. Thus, enumeration fails in machine learning! This makes ML interesting and challenging!