7 A Universal Turing Machine

7.1 A universal Turing machine

On the module web page there is a C program which simulates Turing machines presented as certain sequences of ASCII characters. This program is an interpreter for a very simple programming language (Turing machines presented as sets of quintuples).

We also have a way to encode Turing machines as certain sequences of binary digits. One could write a program in C for simulating Turing machines presented in this way: more importantly, one could construct ***a Turing Machine U which is an 'interpreter' for the bitstring encoding of Turing machines.*** That is, there exists a Turing machine U such that

$$\begin{cases} U(x)\uparrow & \text{if } x \text{ does not factorise as } yz \\ & \text{where } y \in \text{ TM} \\ U(x)\uparrow & \text{if } x = yz \text{ where } y \in \text{ TM and } T_y(z)\uparrow \\ U(x)\downarrow & \text{if } x = yz \text{ where } y \in \text{ TM and } T_y(z)\downarrow. \end{cases}$$

The output of Turing machines can also be produced. For our purposes, it can be assumed that the output alphabet, and the input alphabet, is $\{0,1\}$. Recall that a Turing machine M computes a partial function $f : \{0,1\}^* \nrightarrow \{0,1\}$ if, when $f(z)$ is defined and equal to w, $M(z)\downarrow w$, i.e., M, when started on input z, eventually halts in a halting configuration $q_i w$, and if $f(z)$ is undefined, $M(z)\uparrow$.

A Turing machine U can be constructed with the following sharper property:

$$\begin{cases} U(x)\uparrow & \text{if } x \text{ does not factorise as } yz \\ & \text{where } y \in \text{ TM} \\ U(x)\uparrow & \text{if } x = yz \text{ where } y \in \text{ TM and } T_y(z)\uparrow \\ U(x)\downarrow w & \text{if } x = yz \text{ where } y \in \text{ TM and } T_y(z)\downarrow w. \end{cases}$$

Such a Turing machine is called a *Universal Turing machine.* Note

$$\text{HALTING} = \{x \in \{0,1\}^* : U(x)\downarrow\}$$

We shan't construct U explicitly. The nearest thing to a universal turing machine is the program `turinginC.c` on the module web page.

7.2 Recursively enumerable and recursive sets and functions

(7.1) Definition (i) *A set X of bitstrings is* recursively enumerable *if there exists a Turing machine M such that*

$$X = \{x \in \{0,1\}^* : M(x)\downarrow\}.$$

(ii) *A partial function* $f : \{0,1\}^* \nrightarrow \{0,1\}^*$ *is* partial recursive *if there exists a Turing machine* M *such that* $M(y) \downarrow w$ *if* $f(y)$ *is defined and equal to* w*, and* $M(y) \uparrow$ *if* $f(y)$ *is undefined.*

(iii) *A partial function* $f : \{0,1\}^* \nrightarrow \{0,1\}^*$ *is* recursive *if it is (a) partial recursive and (b) defined everywhere, so its domain is* $\{0,1\}^*$*.*

(iv) *The* characteristic function χ_X *of a set* X *(of bitstrings) is*

$$\chi_X(z) = \begin{cases} 1 & \text{if } z \in X \\ 0 & \text{if } z \notin X \end{cases}$$

A set X *of strings is* recursive *if its characteristic function is recursive.*

(7.2) Corollary *The set* HALTING *is* recursively enumerable *but* not recursive. ∎

This is a widespread phenomenon. Generally, the set of theorems in logic is recursively enumerable but not recursive; ditto, the solvable Diophantine equations, the set of polynomials in triganometric functions which evaluate to zero everywhere; words evaluating to the identity in a group; and more.

7.3 Complements of sets and recursiveness

Let $X \subseteq \{0,1\}^*$ be given. We write $\overline{X}$ to mean the complement of X, possibly for the purposes of this section only. That is, $\overline{X} = \{0,1\}^* \backslash X$.

(7.3) Lemma *If* X *is recursive then* $\overline{X}$ *is recursive.*

Proof. $\chi_{\overline{X}} = 1 - \chi_X$. It is easy to argue that if one function is recursive then so is the other. ∎

(7.4) Lemma *If* X *is recursively enumerable, and* $\overline{X}$ *is recursively enumerable, then* X *and* $\overline{X}$ *are recursive.*

Informal sketch of proof. Let M_1 be a Turing machine which on input x halts if, and only if, $x \in X$. Let M_2 halt if and only if $x \in \overline{X}$. It is possible to construct a Turing machine M which simulates steps of M_1 and M_2 alternately.

On input x, either M_1 will halt and M_2 loop, or vice-versa. So eventually M will discover one or the other case. If M_1 halts, M will halt with output 1, and if M_2 halts, M will halt with output 0. The machine M halts on all inputs and computes χ_X, so X is recursive. ∎

(7.5) Corollary HALTING *is recursively enumerable, but its complement is not.* ∎