Chapter 6

Recursive sets

A set of string is recursive if it can be 'recognised' by a computer. Recursive sets offer an alternative approach to computability. The concept of recursive enumerablity is more subtle, and links up with the Halting Problem.

6.1 Recursive sets

Definition 6.1. *A set of strings $S \subset \mathbb{S}$ is said to be* recursive *if there exists a Turing machine A such that*

$$A(\langle s \rangle) = \begin{cases} 1 & \text{if } s \in S \\ 0 & \text{if } s \notin S. \end{cases}$$

We say that A is an *acceptor* for S, or that A *recognises* S.

Note that if A is an acceptor then $A(\langle s \rangle)$ must be defined for all s. Since the set $\{\langle s \rangle : s \in \mathbb{S}\}$ is a *maximal* prefix-free set, it follows that $A(p)$ must be undefined for all input strings p *not* of the form $\langle s \rangle$,

Proposition 6.1. *1. The empty set $\emptyset$ and $\mathbb{S}$ are recursive.*

2. If $R, S \subset \mathbb{S}$ are recursive then so are $R \cup S$ and $R \cap S$.

3. If S is recursive then so is its complement $\bar{S} = \mathbb{S} \setminus S$.

Proof ▶. 1. This is trivial.

2. Suppose A, B are acceptors for S, T Then we construct a machine C which first emulates A, and then emulates B.

More precisely, given input $\langle s \rangle$, C first saves the input, and then emulates A, taking $\langle s \rangle$ as input.

We know that A will end by outputting 0 or 1.

If A ends by outputting 0, then C outputs nothing, but instead emulates B, again with input $\langle s \rangle$. If A ends by outputting 1, then C outputs 1 and halts.

Evidently C accepts the union $A \cup B$.

We construct a machine which accepts the intersection $A \cap B$ in exactly the same way, except that now it halts if A outputs 0, and emulates B if A outputs 1.

3. Suppose A accepts S. Let the machine C be identical to A, except that C outputs 1 when A outputs 0, and 0 when A outputs 1. Then C accepts the complementary set $\bar{S}$

6.1.1 Recursive codes

We say that a code

$$\gamma : X \to \mathbb{S}$$

for a set X is *recursive* if the image $\mathrm{im}(\gamma) \subset \mathbb{S}$ is recursive.

For example, the codes $\langle n \rangle$ and $\langle s \rangle$ that we have used for numbers and strings are both recursive and prefix-free (since we want to use them as input to Turing machines): and the same is true of our code $\langle T \rangle$ for Turing machines, and indeed all other codes we have used.

6.2 Recursively enumerable sets

Definition 6.2. *The set $S \subset \mathbb{S}$ is said to be* recursively enumerable *if there exists a Turing machine T such that*

$$s \in S \iff s = T(\langle p \rangle) \text{ for some } p \in \mathbb{S}.$$

We will say in such a case that the machine T *outputs* S.

Proposition 6.2. *1. A recursive set is recursively enumerable.*

2. A set $S \subset \mathbb{S}$ is recursive if and only if S and its complement $\mathbb{S} \setminus S$ are both recursively enumerable.

Proof ►. 1. Suppose S is recursive. Let A be an acceptor for S. Then a slight modification A' of A will output S. Thus given an input string $\langle s\rangle$, A' first saves $\langle s\rangle$ and then emulates A taking $\langle p\rangle$ as input. If A concludes by outputting 1 then A' outputs s; while if A concludes by outputting 0 then A' goes into an infinite loop.

2. If S is recursive then so is its complement $\bar{S} = \mathbb{S} \setminus S$, by Proposition 6.1 So if S is recursive then both S and $\bar{S}$ are recursively enumerable.

   Conversely, suppose S and $\bar{S}$ are recursively enumerable. Let C, D output $S, \bar{S}$, respectively (always with coded input $\langle p\rangle$). Then we construct an acceptor A for S as follows.

   Given an input string $\langle p\rangle$, A starts by saving $\langle p\rangle$. Then A runs through a sequence of steps, which we will call Stage 1, Stage 2, ... At stage n, A runs through all strings p of length $\leq n$, carrying out n steps in the computation of $C(\langle p\rangle)$ and then n steps in the computation of $D(\langle p\rangle)$, saving the output string in coded form in either case. If one or both computations end then the output is compared with $\langle s\rangle$. If $C(\langle p\rangle) = \langle s\rangle$ then A outputs 1 and halts; if $D(\langle p\rangle) = \langle s\rangle$ then A outputs 0 and halts.

   One or other event must happen sooner or later since C and D together output all strings $s \in \mathbb{S}$.

This trick is reminiscent of the proof that $\mathbb{N} \times \mathbb{N}$ is enumerable, where we arrange the pairs (m, n) in a 2-dimensional array, and then run down the diagonals,

$$(0,0), (0,1), (1,0), (0,2), (1,1), (2,0), (0,3), (1,2), \ldots.$$

So we will call it the 'diagonal trick'.

It should not be confused with Cantor's entirely different and much more subtle 'diagonal method', used to show that $\#(X) < \#(2^X)$ and in the proof of the Halting Theorem. Note that Cantor's method is used to prove that something is *not* possible, while the diagonal trick is a way of showing that some procedure *is* possible.

Proposition 6.3. *1. $\emptyset, \mathbb{S}$ are recursively enumerable.*

2. If $R, S \subset \mathbb{S}$ are recursively enumerable then so are $R \cup S$ and $R \cap S$.

Proof ►. 1. This follows at once from the fact that $\emptyset$ and $\mathbb{S}$ are recursive.

2. Suppose C, D output R, S. In each case we use the diagonal trick; at stage n we input $\langle p \rangle$ for all p of length $\leq n$, and run C and D for n steps, and determine for which p (if any) C or D halts.

   For $R \cup S$ we simply output $C(\langle p \rangle)$ or $D(\langle p \rangle)$ in each such case.

   For $R \cap S$, we check to see if $C(\langle p \rangle) = D(\langle p' \rangle) = s$ for any inputs p, p', and if there are any such we output s.

6.3 The main theorem

Theorem 6.1. *There exists a set $S \subset \mathbb{S}$ which is recursively enumerable but not recursive.*

Proof ▶. Suppose U is a universal machine. By the Halting Theorem 5.1,

$$S = \{p : U(\langle p \rangle) \text{ defined}\}$$

is not recursive.

For a halting machine in this case is precisely an acceptor for S; and we saw that such a machine cannot exist.

It is easy to see that S is recursively enumerable, using the diagonal trick. At stage n we run though strings p of length $\leq n$, and follow the computation of $U(< p >)$ for n steps, If $U(< p >)$ completes in this time we output p.

It is clear that we will output all $p \in S$ sooner or later.