questions. Here there seems to be no way to proceed without Church's Thesis. Second, we sometimes use Church's Thesis to prove a function is recursive by observing that it is computable and using Church's Thesis to conclude that it is recursive. This type of use is non–essential; we could always use the methods we have developed to prove that the function is recursive. One of the best ways to convince oneself of Church's Thesis is to examine many such examples and see that in every case the function turns out to be recursive.

10. Word Problems

The initial aim of recursion theory was to show that certain problems of the form "Find an algorithm by which ..." were unsolvable. We shall give a few examples of such problems.

Let us first see how to obtain a non–recursive real F. By 8.1, it is enough to make F different from each $\{e\}$. We shall do this by making it different from $\{e\}$ at the argument e. (This idea, known as the diagonal argument, was used first by Cantor to prove that the set of real numbers is uncountable.) In more detail, we define

$$\begin{aligned} F(e) &\simeq \{e\}(e) + 1 && \text{if } \{e\}(e) \text{ is defined,} \\ &\simeq 0 && \text{otherwise.} \end{aligned}$$

It follows from this construction that the set P defined by

$$P(e) \leftrightarrow \{e\}(e) \text{ is defined}$$

is not recursive; for otherwise, the definition of F would be a definition by cases using only recursive symbols, and hence F would be recursive. Thus, using Church's Thesis, we have our first unsolvable problem: find an algorithm for deciding if $\{e\}(e)$ is defined.

Consider the following problem, called the halting problem: Find an algorithm by which, given a program P and an x, we can decide if the computation of P from x halts. Let P be a program which computes the recursive function F defined by $F(e) \simeq \{e\}(e)$. Then the machine halts with program P and input e iff $\{e\}(e)$ is defined. It follows that the halting program is unsolvable, even for this one program P.

To introduce our next problem, we need a few definitions. An <u>alphabet</u> is a finite sequence of symbols. If Ω is an alphabet, an <u>Ω–word</u> is a finite sequence of Ω–symbols. An <u>Ω–production</u> is an expression $X \to Y$, where X and Y are Ω–words. An <u>Ω–process</u> is a finite set of Ω–productions. We usually suppose Ω is fixed and omit the prefixes Ω.

If X and Y are words and P is a production $Z \to V$, then $X \to_P Y$ means that Y results from X by replacing an occurrence of Z in X by V. If W is a process, $X \to_W Y$ means that $X \to_P Y$ for some production P in W; and $X \Rightarrow_W Y$ means that there are words $Z_1, \ldots, Z_k$ such that Z_1 is X, Z_k is Y, and $Z_i \to_W Z_{i+1}$ for $1 \leq i < k$.

The <u>word problem</u> for an alphabet Ω is the following: Find an algorithm by which, given an Ω–process W and Ω–words X and Y, we can decide if $X \Rightarrow_W Y$. We shall show that this problem is unsolvable, even for a particular choice of W and Y.

Let P be a program for which the halting problem is unsolvable. We shall construct a process W. We use a, b, c, d, and e, possibly with subscripts, for symbols of our alphabet. We use a^r for the expression consisting of r occurrences of a. Let N be the number of instruction in P and let $M > 1$ so that $i < M$ for every i such that $\mathfrak{R}i$ is mentioned in P. If r_i is the number in $\mathfrak{R}i$ and n is the number in the counter, the word

$$\mathrm{b}\,\mathrm{c}_n\,\mathrm{b}_0\,\mathrm{a}^{r_0}\,\mathrm{b}_1\,\mathrm{a}^{r_1}\ldots\mathrm{b}_{M-1}\mathrm{a}^{r_{M-1}}\,\mathrm{b}_M$$

is called the <u>status word</u>. Thus if we do the computation of P from x, the initial status word is $\mathrm{bc}_0\mathrm{b}_0\mathrm{b}_1\mathrm{a}^x\mathrm{b}_2\ldots\mathrm{b}_M$, which we write as Z_x. We construct W so that $Z_x \Rightarrow_W \mathrm{bc}_N$ iff the computation of P from x halts. This will imply that there is no algorithm by which, given x, we can decide if $Z_x \Rightarrow_W \mathrm{bc}_N$.

Suppose the machine is executing P. If X is a status word beginning with bc_n, where $n < N$, then there is a next status word Y. We shall put productions in W to insure that $X \Rightarrow_W Y$.

Suppose first that instruction n in P is INCREASE $\mathfrak{R}i$. We put into W the productions $\mathrm{c}_n\mathrm{b}_j \rightarrow \mathrm{b}_j\mathrm{c}_n$ for $j < i$ and the production $\mathrm{c}_n\mathrm{a} \rightarrow \mathrm{ac}_n$. Applying these productions to X enables us to move the c_n until it stands just before b_i. We also put $\mathrm{c}_n\mathrm{b}_i \rightarrow \mathrm{d}_n\mathrm{b}_i\mathrm{a}$ into W; this enables us to increase the number in $\mathfrak{R}i$ by 1 while changing c_n to d_n. The productions $\mathrm{ad}_n \rightarrow \mathrm{d}_n\mathrm{a}$ and $\mathrm{b}_j\mathrm{d}_n \rightarrow \mathrm{d}_n\mathrm{b}_j$ for $j < i$ enable us to move the d_n until it stands just after b. The production $\mathrm{bd}_n \rightarrow \mathrm{bc}_{n+1}$ then gives Y.

Now suppose that instruction n in P is DECREASE $\mathfrak{R}i,m$. Just as above, the productions $\mathrm{c}_n\mathrm{b}_j \rightarrow \mathrm{b}_j\mathrm{c}_n$ for $j < i$, $\mathrm{c}_n\mathrm{a} \rightarrow \mathrm{ac}_n$, $\mathrm{c}_n\mathrm{b}_i\mathrm{a} \rightarrow \mathrm{d}_n\mathrm{b}_i$, $\mathrm{ad}_n \rightarrow \mathrm{d}_n\mathrm{a}$, $\mathrm{b}_j\mathrm{d}_n \rightarrow \mathrm{d}_n\mathrm{b}_j$ for $j < i$, and $\mathrm{bd}_n \rightarrow \mathrm{bc}_m$ take care of the case in which the number in $\mathfrak{R}i$ is not 0. If it is 0, the above productions again move c_n until it is just before b_i. Then $\mathrm{c}_n\mathrm{b}_i\mathrm{b}_{i+1} \rightarrow \mathrm{e}_n\mathrm{b}_i\mathrm{b}_{i+1}$ changes c_n to e_n; $\mathrm{ae}_n \rightarrow \mathrm{e}_n\mathrm{a}$ and $\mathrm{b}_j\mathrm{e}_n \rightarrow \mathrm{e}_n\mathrm{b}_j$ for $j < i$ bring e_n to just after b; and $\mathrm{be}_n \rightarrow \mathrm{bc}_{n+1}$ gives Y.

If instruction n is GO TO m, then $\mathrm{bc}_n \rightarrow \mathrm{b}c_m$ changes X to Y.

We also add the productions $\mathrm{c}_N\mathrm{b}_i \rightarrow \mathrm{c}_N$ for all i and the production $\mathrm{c}_N\mathrm{a} \rightarrow \mathrm{c}_N$; these enable us to convert any status word beginning with bc_N to bc_N. Hence if the computation of P from x halts, then $Z_x \Rightarrow_W \mathrm{bc}_N$.

A word is <u>special</u> if it contains exactly one occurrence of the c_i, d_i, and e_i symbols. Every status word is special. Moreover, if X is special and $X \rightarrow_W Y$, then Y is special. It is easily checked that if X is special, there is at most one Y such that $X \rightarrow_W Y$.

Now suppose that the computation of P from x never halts. Then there is an infinite sequence X_0, X_1, ... beginning with Z_x such that $X_i \rightarrow_W X_{i+1}$ for all i. The remarks of the previous paragraph then show that the X_i are the only words

X such that $Z_x \Rightarrow_W X$. Hence we cannot have $Z_x \Rightarrow_W \mathrm{bc}_N$; for there is no word V such that $\mathrm{bc}_N \rightarrow_W V$. This completes our proof that the word problem is unsolvable.

A process W is symmetric if whenever it contains $X \rightarrow Y$ it also contains $Y \rightarrow X$. We will show that the word problem is unsolvable even for symmetric processes.

Let W be the process constructed above. Let W' be the symmetric process obtained from W by adding the production $Y \rightarrow X$ for every production $X \rightarrow Y$ in W. We shall show that $Z_x \Rightarrow_{W'} \mathrm{bc}_N$ iff $Z_x \Rightarrow_W \mathrm{bc}_N$; it will then follow that the word problem for W' is unsolvable.

It is enough to show that $Z_x \Rightarrow_{W'} \mathrm{bc}_N$ implies $Z_x \Rightarrow_W \mathrm{bc}_N$. Let $X_1, \ldots, X_k$ be a sequence of the minimum length such that X_1 is Z_x, X_k is bc_N, and $X_i \rightarrow_{W'} X_{i+1}$ for $1 \leq i < k$. Since X_1 is special, it follows by induction on i that X_i is special. If $X_i \rightarrow_W X_{i+1}$ holds for all $i < k$, we are done. If not, pick the largest i such that this is false. Then $X_{i+1} \rightarrow_W X_i$. It follows that X_{i+1} is not bc_N; so $i+1 < k$. By choice of i, $X_{i+1} \rightarrow_W X_{i+2}$. Since X_{i+1} is special, it follows that $X_i = X_{i+2}$. But this means that we could omit X_i and X_{i+1} from our sequence, contradicting the choice of that sequence.

Symmetric processes are interesting because of their relation to semigroups. A semigroup is a class S with a binary operation $\cdot$ such that the associative law holds (i.e., $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ for all $x,y,z \in S$) and there is a unit element (i.e., an $e \in S$ such that $e \cdot x = x \cdot e = x$ for all $x \in S$).

Let Ω be an alphabet. An Ω-semigroup consists of a semigroup S and an element x_{a} of S for every symbol a in Ω. We think of the symbol a as designating the element x_{a}. More generally, the word $\mathrm{a}_1 \ldots \mathrm{a}_k$ designates the element $x_{\mathrm{a}_1} \cdot \ldots \cdot x_{\mathrm{a}_k}$. (Note that no parentheses are needed because of the associative law.)

An <u>Ω–relation</u> is an expression $X = Y$ where X and Y are Ω–words. Then if S is an Ω–semigroup, $X = Y$ is either true or false in S. Now let R be a finite set of Ω–relations and let K be an Ω–relation. Then $R \Rightarrow K$ means that K is true in every Ω–semigroup in which all of the relations in R are true. The <u>word problem for Ω–semigroups</u> is to find an algorithm by which, given R and K, we can decide if $R \Rightarrow K$.

We shall show that the word problem for Ω–semigroups is unsolvable. (This was proved independently by Post and Markov.) Let W' be the symmetric process constructed above. Let R consist of the relations $X = Y$ such that $X \to Y$ is in W' (and hence $Y \to X$ is in W'). We shall show that $X \Rightarrow_{W'} Y$ iff $R \Rightarrow X = Y$. Hence the word problem for Ω–semigroups is unsolvable even for this particular R.

Clearly $X \Rightarrow_{W'} Y$ implies $R \Rightarrow X = Y$. To prove the implication in the other direction, we construct an Ω–semigroup. First note that the relation $X \Rightarrow_{W'} Y$ between X and Y is an equivalence relation on the class of Ω–words; this follows from the fact that W' is symmetric. Let X^* be the equivalence class of X. Let S be the set of all these equivalence classes; and define a binary operation $\cdot$ on S by $X^* \cdot Y^* = (XY)^*$ (where XY is X followed by Y). A little thought shows that $(XY)^*$ depends only on the equivalence classes X^* and Y^*; so our definition makes sense. It is easy to see that S is then a semigroup; the unit element is the equivalence class of the empty word.

We make S into an Ω–semigroup by letting the symbol a represent a^*; the word X then represents X^*. If $X = Y$ is in R, then X and Y are equivalent; so $X^* = Y^*$; so $X = Y$ is true in S. It follows that if $R \Rightarrow X = Y$, then $X = Y$ is true in S and hence $X \Rightarrow_{W'} Y$. This completes our proof.

11. Undecidable Theories

We shall see how some problems of the following type can be shown to be