$$\begin{aligned} L(y,x,f) &\simeq G(x) && \text{if } y = 0, \\ &\simeq H(\{f\}(y \dot{-} 1, 2x), y \dot{-} 1, x) && \text{otherwise.} \end{aligned}$$

Then we use the Recursion Theorem to obtain a recursive F with an index f such that $F(y,x) \simeq L(y,x,f)$. Clearly F satisfies (2); so the function defined by (2) is recursive.

9. Church's Thesis

We have already remarked that it is clear that every recursive function is computable. The statement that every computable function is recursive is known as Church's Thesis. It was proposed by Church about 1934 and has since come to be accepted by almost all logicians. We shall discuss the reasons for this.

Since the notion of a computable function has not been defined precisely, it may seem that it is impossible to give a proof of Church's Thesis. However, this is not necessarily the case. We understand the notion of a computable function well enough to make some statements about it. In other words, we can write down some axioms about computable functions which most people would agree are evidently true. It might be possible to prove Church's Thesis from such axioms. However, despite strenuous efforts, no one has succeeded in doing this (although some interesting partial results have been obtained).

We are thus reduced to trying to give arguments for Church's Thesis which seem to be convincing. We shall briefly examine these arguments.

The first argument is that all the computable functions which have been produced have been shown to be recursive, using, for the most part, the techniques which we have already described. Moreover, all the known techniques for producing new computable functions from old ones (such as definition by induction or by cases) have been shown to lead from recursive functions to recursive functions.

Another argument comes from various attempts to define computable precisely. We have seen two of these: the definition by means of the basic machine and the definition by means of recursively closed classes (see Proposition 8.3). There are many others, some similar to these two and some quite different. In every case, it has been proved that the class of functions so defined is exactly the class of recursive functions. This at least shows that the class of recursive functions is a very natural class; and it is hard to see why this should be so unless it is indeed the class of computable functions.

Now let us consider how we might generalize the basic machine to produce a new computable function. Since the registers can contain an arbitrarily large finite number of arbitrary numbers, and since so much information can be coded in a single number, it seems pointless to increase the amount of memory. We therefore need to add new instructions. For each new instruction, we must, of course, have an algorithm for executing that instruction. Essentially that means that the functions *Reg* and *Count* of §8, when extended to allow for the new instructions, must be computable. Since these function control just one step in the computation, they should be relatively simple computable functions. We might therefore agree (perhaps on the basis of the above arguments) that *Reg* and *Count* must be recursive. But then we could repeat the remaining definitions of §8 and show that every function computed by the new machine is recursive.

We could elaborate on all of these arguments. However, most people become convinced of Church's Thesis only by a detailed study of recursion theory. The most convincing argument is that all of the results of recursion theory become quite reasonable (or even obvious) when recursive is replaced by computable.

We shall henceforth accept Church's Thesis. It will be used in two ways. First, there are many natural and interesting questions about computable functions. We use Church's Thesis to convert these into precise mathematical 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 re-