A 1–ary relation is simply a set of numbers. We understand set to mean set of numbers; we will use the word class for other kinds of sets. We use A and B for sets.

If R is a k–ary relation, the representing function of R, designated by χ_R, is the k–ary total function defined by

$$\chi_R(\vec{x}) = 0 \qquad \text{if } R(\vec{x}),$$
$$= 1 \qquad \text{otherwise.}$$

A relation R is computable if the function χ_R is computable. We adopt the convention that whenever we attribute to a relation some property usually attributed to a function, we are actually attributing that property to the representing function of the relation.

3. The Basic Machine

To define our class of functions, we introduce a computing machine called the basic machine. It is an idealized machine in that it has infinitely much memory and never makes a mistake. Except for these features, it is about as simple as a computing machine can be.

For each number i, the computing machine has a register $\mathcal{R}i$. At each moment, $\mathcal{R}i$ contains a number; this number (which has nothing to do with the number i) may change as the computation proceeds.

The machine also has a program holder. During a computation, the program holder contains a program, which is a finite sequence of instructions. If N is the number of instructions in the program, the instructions are numbered 0, 1, ..., N–1 (in the order in which they appear in the program). The machine also has a counter, which at each moment contains a number.

To use the machine, we insert a program in the program holder; put any desired numbers in the registers; and start the machine. This causes 0 to be inserted in the counter. The machine then begins executing instructions. At

each step, the machine executes the instruction in the program whose number is in the counter at the beginning of the step, provided that there is such an instruction. If at any time the number in the counter is larger than any number of an instruction in the program, then the machine halts. If this never happens, the machine goes on executing instructions forever.

The instructions are of three types. The first type has the format INCREASE $\mathcal{R}i$. When the machine executes this instruction, it increases the number in $\mathcal{R}i$ by 1 and increases the number in the counter by 1. The second type has the format DECREASE $\mathcal{R}i,n$, where n is the number of an instruction in the program. If the machine executes this instruction when the number in $\mathcal{R}i$ is not 0, it decreases the number in that register by 1 and changes the number in the counter to n. If it executes this instruction when the number in $\mathcal{R}i$ is 0, it increases the number in the counter by 1. The third type has the format GO TO n, where n is the number of an instruction in the program. When the machine executes this instruction, it changes the number in the counter to n. Note that if $\mathcal{R}i$ is not mentioned in an instruction, then the instruction does not change the number in $\mathcal{R}i$ and the number if $\mathcal{R}i$ does not affect the action of the instruction.

This completes the description of the basic machine. Of course, we have only described the action of the machine, not its physical construction. However, all of the actions of the basic machine can be carried out by a person with pencil and paper and with the program in front of him; he merely keeps track at each step of the number in the counter and the numbers in the registers mentioned in the program.

For each program P for the basic machine and each k, we define an algorithm A_k^P with k inputs. To apply this algorithm to the inputs $x_1,...,x_k$, we start the machine with P in the program holder, $x_1,...,x_k$ in $\mathcal{R}1,...,\mathcal{R}k$ respectively, and 0 in all other registers. If the machine eventually halts, the number in $\mathcal{R}0$ after it halts is the output; otherwise, there is no output. The k–ary function

computed by P is the function computed by A_k^P. A k–ary function F is recursive if it is the k–ary function computed by some program for the basic machine. (In accordance with our convention, a relation is recursive iff its representing function is recursive.)

It is clear that every recursive function is computable. It is not at all evident that every computable function is recursive; but, after some study of the recursive functions, we shall argue that this is also the case.

4. Macros

It is tedious to write programs for the basic machine because of the small number of possible instructions. We shall introduce some new instructions and show that they do not enable us to compute any new functions. The idea is familiar to programmers: the use of subroutines, or, as they are often called nowadays, macros.

For each program P for the basic machine, we introduce a new instruction P^*, called the macro of P. When the machine executes this instruction, it begins executing program P (with whatever numbers happen to be in the registers at the time). If this execution never comes to an end, then the execution of P^* is never completed. If the execution of P is completed, the machine changes the number in the counter to 1 more than the number of the instruction P^* and continues executing instructions. The macro machine is obtained from the basic machine by adding all macros of programs for the basic machine as new instructions. We define the notion of a program computing a function for the macro machine as we did for the basic machine.

We say that the program P and P' are equivalent if the following holds. Suppose that we start two machines with P in the program holder of the first machine, P' in the program holder of the second machine, and the same number in $\mathfrak{N}i$ in both machines for all i. Then either both machines will compute forever;