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;

or both machines will halt, and, when they do, the same number will be in $\mathcal{R}i$ in both machines for every i. Clearly equivalent programs compute the same k–ary function.

4.1. PROPOSITION. Every program for the macro machine is equivalent to a program for the basic machine.

Proof. Let P be a program for the macro machine. For each macro Q^* in P, we replace the instruction Q^* by the sequence of instruction Q. We then number the instructions in the resulting program. Finally, we change each instruction number within an instruction (i.e., each number n in an instruction DECREASE $\mathcal{R}i,n$ or GO TO n) so that it refers to the same instruction (in P or in one of the Q's) that it did originally. The result is a program P' for the basic machine.

Suppose that we start with two machines as in the definition of equivalent. The machines will perform the same operations until the first executes a macro Q^*. Then both machines begin executing Q. If neither finishes executing Q, we are done. Otherwise, both finish Q with the same number in $\mathcal{R}i$ in both machines for all i. The number in the counter of the first will be 1 more than the number of Q^*; and the number in the counter of the second will be 1 more than the number of the last instruction in Q. (This is because the execution of Q can only stop by executing the last instruction and having the counter increase by 1.) Thus either both machines will stop, or they will continue performing the same operations. □

4.2. COROLLARY. Every function computed by a program for the macro machine is recursive. □

We now introduce some useful macros. The program

0) DECREASE $\mathcal{R}i$,0

causes the number in $\mathcal{R}i$ to be changed to 0. We write the macro of this program as ZERO $\mathcal{R}i$.

We now want a program to move the number in $\mathcal{R}i$ into $\mathcal{R}j$. We could do this by repeatedly decreasing $\mathcal{R}i$ and increasing $\mathcal{R}j$, but this would change the number in $\mathcal{R}i$ to 0. If we want to avoid this, we need another register $\mathcal{R}k$. We then move the number in $\mathcal{R}i$ into $\mathcal{R}j$ and $\mathcal{R}k$, and then move it back from $\mathcal{R}k$ to $\mathcal{R}i$.

In detail, suppose that i, j, and k are distinct. Then the program

0) ZERO $\mathcal{R}j$,

1) ZERO $\mathcal{R}k$,

2) GO TO 5,

3) INCREASE $\mathcal{R}j$,

4) INCREASE $\mathcal{R}k$,

5) DECREASE $\mathcal{R}i$,3,

6) GO TO 8,

7) INCREASE $\mathcal{R}i$,

8) DECREASE $\mathcal{R}k$,7

causes the number in $\mathcal{R}i$ to be moved into $\mathcal{R}j$ without changing the number in $\mathcal{R}i$. We write the macro of this program as MOVE $\mathcal{R}i$ TO $\mathcal{R}j$ USING $\mathcal{R}k$. (More precisely, this is the macro of the program for the basic machine which is, by 4.1, equivalent to the above program for the macro machine.) Usually we are not interested in $\mathcal{R}k$; we then write simply MOVE $\mathcal{R}i$ TO $\mathcal{R}j$, and understand that $\mathcal{R}k$ is to be chosen different from all registers mentioned in the program.

Let F be a k–ary recursive function, and let P be a program which computes F. Choose $m > k$ so that P does not mention $\mathcal{R}i$ for $i \geq m$. Let Q be the program for the macro machine consisting of the following instructions: MOVE $\mathcal{R}i$ TO $\mathcal{R}(m+i)$ USING $\mathcal{R}m$ for $1 \leq i < m$; ZERO $\mathcal{R}0$; ZERO $\mathcal{R}i$ for $k < i < m$; P^*; and MOVE $\mathcal{R}(m+i)$ TO $\mathcal{R}i$ USING $\mathcal{R}m$ for $1 \leq i < m$. We leave the reader to check that if Q is executed with $x_1,\ldots,x_k$ in $\mathcal{R}1,\ldots,\mathcal{R}k$, then the machine eventually halts iff $F(x_1,\ldots,x_k)$ is defined; and in this case, $F(x_1,\ldots,x_k)$ is in $\mathcal{R}0$, and the number in $\mathcal{R}i$ is unchanged unless $i = 0$ or $m \leq i < 2m$.

Now let $i_1,...,i_k,j,n_1,...,n_m$ be distinct. By changing register numbers in Q, we produce a program Q' with the following property. If Q' is executed with $x_1,...,x_k$ in $\mathcal{R}i_1,...,\mathcal{R}i_k$, then the machine eventually halts iff $F(x_1,...,x_k)$ is defined; and in this case, $F(x_1,...,x_k)$ is in $\mathcal{R}j$, and the number in $\mathcal{R}i$ is unchanged unless $i = j$ or i is one of $n_1,...n_m$. We write the macro of Q' as

$$F(\mathcal{R}i_1,...,\mathcal{R}i_k) \to \mathcal{R}j \text{ USING } \mathcal{R}n_1,...,\mathcal{R}n_m.$$

As above, we generally omit USING $\mathcal{R}n_1,...,\mathcal{R}n_m$.

5. Closure Properties

We are going to show that the class of recursive functions has certain closure properties; i.e., that certain operations performed on members of the class lead to other members of the class. In later sections, we shall use these results to see that various functions are recursive.

If $1 \leq i \leq k$, we define the total k–ary function I_i^k by $I_i^k(x_1,...,x_k) = x_i$. Recall that every number is a 0–ary total function. The successor function Sc is defined by $Sc(x) = x + 1$. The function I_i^k, 0, and Sc are called the initial functions.

5.1. PROPOSITION. The initial functions are recursive.

Proof. The function I_i^k is computed by the program

0) MOVE $\mathcal{R}i$ TO $\mathcal{R}0$.

The function 0 is computed by the program

0) ZERO $\mathcal{R}0$.

The function Sc is computed by the program

0) MOVE $\mathcal{R}1$ TO $\mathcal{R}0$,

1) INCREASE $\mathcal{R}0$. □

Because our functions need not be total, we often meet expressions which may be undefined. Thus if F and G are unary, $F(G(x))$ is defined iff x is in the domain of G and $G(x)$ is in the domain of F. Suppose that X and Y are