Proof. By 17.5. □

A <u>maximal</u> <u>set</u> is a coinfinite RE set A such that for every coinfinite RE set B including A, $B - A$ is finite. Thus a maximal set is a coinfinite RE set with as few RE sets as possible including it.

It is fairly easy to show that a maximal set is hypersimple. However, it is not a simple matter to show that maximal sets exist; this was done by Friedberg. The final result of a series of investigations of this question is the following theorem of Martin: an RE degree $\mathbf{a}$ contains a maximal set iff $\mathbf{a}' = \mathbf{0}''$. Thus this notion of largeness does tell us more about the degree than our previous notions, but does not tell us that the degree cannot be $\mathbf{0}'$.

18. Function of Reals

We now extend our notion of a function to allow reals as arguments. (We could allow all total functions as arguments; but this would complicate matters without really adding anything, since a function can be replaced by its contraction.) We use lower case Greek letters, usually α, β, and γ, for reals. When the value of m is not important, we write $\vec{\alpha}$ for $\alpha_1,\dots,\alpha_m$. We use $\mathbb{R}$ for the class of reals and $\mathbb{R}^{m,k}$ for the class of all $(m+k)$–tuples $(\alpha_1,\dots,\alpha_m,x_1,\dots,x_k)$. An <u>$(m,k)$–ary</u> <u>function</u> is a mapping of a subset of $\mathbb{R}^{m,k}$ into ω. (Thus a $(0,k)$–ary function is just a k–ary function.) From now on, a function is always an (m,k)–ary function for some m and k. Such a function is <u>total</u> if its domain is all of $\mathbb{R}^{m,k}$. An <u>(m,k)–ary</u> <u>relation</u> is a subset of $\mathbb{R}^{m,k}$. We define the representing function of such a relation as before.

Note that the real arguments to a function or relation must precede the number arguments. It may sometimes be convenient to write them in a different order. It is then understood that we are to move all real arguments to the left of all number arguments without otherwise changing the order of the arguments.

Now we consider how to extend the idea of computability. The new question is: how are we to be given the real inputs? The obvious answer is that we are given an oracle for each real input.

We now modify the basic machine accordingly. For each m, we have a machine called the m–real machine. (The 0–real machine will be identical with the basic machine.) In addition to the parts of the basic machine, the m–real machine has m real registers $\mathcal{F}1,\ldots\ \mathcal{F}m$. At any moment, each of these registers contains a real α.

We have one new type of instruction. It has the format

$$\mathcal{F}k(\mathcal{R}i) \to \mathcal{R}j$$

where $1 \leq k \leq m$ and $i \neq j$. If the machine executes this instruction when α is in $\mathcal{F}k$ and x is in $\mathcal{R}i$, it changes the number in $\mathcal{R}j$ to $\alpha(x)$ and increases the number in the counter by 1. Note that no instruction changes the contents of a real register.

With each program P and each m and k, we associate an algorithm $\mathrm{A}_P^{m,k}$ with m real inputs and k number inputs. To perform this algorithm with the inputs $\alpha_1,\ldots,\alpha_m,x_1,\ldots,x_k$, we put P in the program holder; $\alpha_1,\ldots\alpha_m$ in $\mathcal{F}1,\ldots,\mathcal{F}m$ respectively; $x_1,\ldots,x_k$ in $\mathcal{R}1,\ldots,\mathcal{R}k$ respectively; and 0 in all other registers. We then start the machine. If the machine ever halts, the number in $\mathcal{R}0$ after it halts is the output; otherwise, there is no output. The function computed by $\mathrm{A}_P^{m,k}$ is called the (m,k)–ary function computed by P. An (m,k)–ary function F is recursive if there is a program P such that F is the (m,k)–ary function computed by P.

We now propose to extend our previous results to these new functions. We shall have something to say only when the extensions present some problems.

We extend §4 without difficulty. The last macro now reads

$$F(\mathcal{F}p_1,\ldots,\mathcal{F}p_m,\mathcal{R}i_1,\ldots,\mathcal{R}i_k) \to \mathcal{R}j.$$

(As before, $i_1,\ldots,i_k$ should be distinct; but $p_1,\ldots,p_m$ need not be. The reader should check that this is all right.)

In §5, we need some changes to allow for the real arguments. The initial functions are now the $I_i^{m,k}$, 0, Sc, and Ap, where $I_i^{m,k}(\alpha_1,\ldots,\alpha_m,x_1,\ldots,x_k) = x_i$ and $Ap(\alpha,x) = \alpha(x)$. The program for Ap consists of the one instruction $\mathcal{F}1(\mathfrak{R}1) \to \mathfrak{R}0$. In the definition of closed under composition, F is now defined by

$$F(\alpha_1,\ldots,\alpha_m,\vec{x}) \simeq G(\alpha_{i_1},\ldots,\alpha_{i_r},H_1(\alpha_1,\ldots,\alpha_m,\vec{x}),\ldots,H_k(\alpha_1,\ldots,\alpha_m,\vec{x})).$$

In the case of inductively closed and μ–closed, all the functions have the same sequence $\vec{\alpha}$ of real arguments.

In §6, there are a couple of new details in the proof of 6.1. First, there is a new context $\alpha(\_\_\_\_)$. We take care of this by replacing it by $Ap(\alpha,\_\_\_\_)$. We leave it to the reader to check that our modification in the definition of closed under composition is just what is needed here.

We can extend the result in §7 that F is recursive iff $\overline{F}$ is recursive. In particular, $\overline{Ap}$ is recursive. But $\overline{Ap}(\alpha,x) = \overline{\alpha}(x)$. Thus show that $\overline{\alpha}$ (where α is a variable) is a recursive symbol.

In extending §8, we assign the code $<3,i,j,k>$ to $\mathcal{F}k(\mathfrak{R}i) \to \mathfrak{R}j$. Other codes are as before. (In particular, these codes do not depend in any way on what is in the real registers.) We now take $T_{k,m}(e,\alpha_1,\ldots,\alpha_m,x_1,\ldots,x_k,y)$ to mean that e is the code of a program for the real m–machine and y is the code of the P–computation from $\alpha_1,\ldots,\alpha_m,x_1,\ldots,x_k$, and leave U as before. The extension of §8 is then straightforward.

Note that if Φ is $\alpha_1,\ldots,\alpha_m$, the code of the instruction $\mathcal{F}k(\mathfrak{R}i) \to \mathfrak{R}j$ is the same as the code assigned to the instruction $\alpha_k(\mathfrak{R}i) \to \mathfrak{R}k$ for the Φ–machine; and if $\alpha_1,\ldots,\alpha_k$ are in $\mathcal{F}1,\ldots,\mathcal{F}k$, performing these two instructions has the same effect on the two machines. It follows that

$$T_{k,m}(e,\vec{\alpha},\vec{x},y) \leftrightarrow T_k^{\vec{\alpha}}(e,\vec{x},y).$$

If we use (1) of §12, we can rewrite this as

$$(1) \qquad T_{k,m}(e,\vec{\alpha},\vec{x},y) \leftrightarrow T_{k,m}(e,\vec{x},y,\overline{\vec{\alpha}}(y)).$$

(The two $T_{k,m}$'s are different, but no confusion will result.) It follows by the Normal Form Theorem that

$$(2) \qquad \{e\}(\vec{\alpha},\vec{x}) \simeq \{e\}^{\vec{\alpha}}(\vec{x}).$$

The definitions and results of §12 can be extended without difficulty; but this does not give us what we really want if the functions in Φ have real arguments. The problem is that the F–instructions are not general enough. They evaluate F only at real arguments which are in a real register. Since the contents of a real register do not change during a computation, we cannot evaluate F at real arguments computed during the computation.

We shall therefore restrict the Φ in relative recursion to consists of total function of number arguments only. The results of §12 then extend without difficulty. We call the machine obtained from the real m–machine by adding the F–instructions for F in Φ the <u>Φ–m–machine</u>. If Φ is a finite sequence $H_1,...,H_p$ of reals, we assign to the H_k–instruction $H_k(\mathfrak{X}i) \to \mathfrak{X}j$ the code $<3,i,j,m+k>$. In place of (2) we now have $\{e\}^{\Phi}(\vec{\alpha},\vec{x}) \simeq \{e\}^{\vec{\alpha},\Phi}(\vec{x})$. From this and (2),

$$(3) \qquad \{e\}^{\Phi}(\vec{\alpha},\vec{x}) \simeq \{e\}(\vec{\alpha},\Phi,\vec{x}).$$

This gives the following alternative definition of relative recursion.

18.1. Proposition. A function F is recursive in Φ iff it has a definition $F(\vec{\alpha},\vec{x}) \simeq G(\vec{\beta},\vec{\alpha},\vec{x})$ where G is recursive and $\vec{\beta}$ is a sequence of contractions of functions in Φ. $\square$

Remark. Even if the F in 18.1 is total, we cannot always take the G to be total.

We now consider substitution for real variables. Of course, we cannot substitute something like $F(\_\_\_\_)$ for α, since $F(\_\_\_\_)$ is a number. We therefore need some notation. We let $\lambda x(..x..)$ be the unary function F defined by $F(x) \simeq ..x..$. Then $\lambda x(..x..)$ is a real iff $..x..$ is defined for all x.

18.2. Substitution Theorem. If G and H are recursive, there is a recursive F such that

$$(4) \qquad F(\vec{\alpha},\vec{x}) \simeq G(\lambda z H(z,\vec{\alpha},\vec{x}), \vec{\alpha}, \vec{x})$$

for all $\vec{\alpha},\vec{x}$ such that $\lambda z H(z,\vec{\alpha},\vec{x})$ is a real. In particular, if H is total, then the F defined by (4) is recursive.

Proof. Let g be an index of G. If $\lambda z H(z,\vec{\alpha},\vec{x})$ is a real, then the right side of (4) is, by (1),

$$U(\mu y T_{m,k}(g,\vec{x},y,\overline{H}(y,\vec{\alpha},\vec{x}),\vec{\overline{\alpha}}(y))).$$

We can use this as our definition of $F(\vec{\alpha},\vec{x})$. □

In particular, it follows that λy is a recursive expression when it is used in front of an expression defined for all values of y.

Remark. If H is not total, there may be $\vec{\alpha},\vec{x}$ such that $F(\vec{\alpha},\vec{x})$ is defined, but such that $\lambda z H(y,\vec{\alpha},\vec{x})$ is not a real and hence such that the right side of (4) is not defined.

The results of §13 and §14 extend without difficulty. However, in §13 it is natural to consider a further extension in which we allow quantifiers on real variables. We investigate this in the next section.

19. The Analytical Hierarchy

A relation is <u>analytical</u> if it has an explicit definition with a prefix consisting of quantifiers, which may be either universal or existential and may be on either number variables or real variables, and a recursive matrix. The basic theory of analytical relations is due to Kleene.

We begin with some rules for simplifying prefixes. As before, these may change the matrix, but they leave it recursive.

Two quantifiers are of the <u>same kind</u> if they are both universal or both existential; they are of the <u>same type</u> if they are both on real variables or both on