consists of a binary function symbol F. Hence by 11.2 and 11.3, it will suffice to construct a model M' of PO such that M is definable in an inessential extension of M'.

Let $M_1 = |M| \cup \{1,2,3\}$, where 1,2,3 are objects not in $|M|$. Let M_2 be the set of ordered pairs (x,i) where $x \in |M|$ and $i \in \{1,2,3\}$. Let M_3 be the set of ordered triples (x,y,z) such that $x,y,z \in |M|$ and $F_M(x,y) = z$. Let $|M'| = M_1 \cup M_2 \cup M_3$. We define $<_{M'}$ as follows. If $x \in M_1$, $w <_{M'} x$ is false for all w. If $<x,i> \in M_2$, then $w <_{M'} <x,i>$ holds for $w = x$ and $w = i$. If $<x,y,z> \in M_3$, then $w <_{M'} <x,y,z>$ if w is one of $<x,1>$, $<y,2>$, $<z,3>$, x, y, z, 1, 2, or 3. Clearly M' is a partially ordered set.

For $x,y,z \in |M'|$ we have

$$x \in |M| \leftrightarrow \forall y \neg(y < x) \ \&\ x \neq 1 \ \&\ x \neq 2 \ \&\ x \neq 3,$$

$$F(x,y) = z \leftrightarrow x,y,z \in |M| \ \&\ \exists u \exists x_1 \exists y_1 \exists z_1 (x < x_1 \ \&\ y < y_1 \ \&\ z < z_1 \ \&$$
$$1 < x_1 \ \&\ 2 < y_1 \ \&\ 3 < z_1 \ \&\ x_1 < u \ \&\ y_1 < u \ \&\ z_1 < u).$$

(In proving the second equivalence from right to left, one should first note that we must have $x_1, y_1, z_1 \in M_2$ and $u \in M_3$.) It follows easily that M is definable in M'', where M'' is an inessential expansion of M' formed by adding three new constants to represent 1, 2, and 3.

It follows that PO is undecidable. It also follows that a theory whose language consists of one binary relation symbol and which has no axioms is undecidable.

Many other strongly undecidable structures can be constructed by these methods. However, the proof that M is definable in M' often requires a very detailed analysis of M and M'.

12. Relative Recursion

Let Φ be a set of *total* functions. We generalize the notion of computable to allow us to use the values of the functions in Φ at any arguments we wish in the course of the computation. Following Turing, we picture the computation as taking place as follows. For each F in Φ, we are given an object called an <u>oracle</u> for F. During the computation, we may ask the oracle for $F(\vec{x})$ for any $\vec{x}$ which we have computed. The oracle supplies the value, which may then be used in the rest of the computation. A function which can be computed using oracles for the functions in Φ is said to be computable <u>in</u> Φ or <u>relative</u> <u>to</u> Φ. (Turing used this terminology because the oracle produces a value of the function without the use of an algorithm. However, one should not consider an oracle as a mystical object. We can think of it as an infinite set of file cards on each of which a value of the function is printed; we get the value at a particular set of arguments by searching through the file.)

We now extend the notion of recursive in a similar way. For each F in Φ, we introduce a new type of instruction, called an <u>F–instruction</u>. This instruction has the format

$$F(\mathfrak{X}i_1,\ldots,\mathfrak{X}i_k) \to \mathfrak{X}j$$

where $i_1,\ldots,i_k,j$ are distinct. If the machine executes this instruction when $x_1,\ldots,x_k$ are in $\mathfrak{X}i_1,\ldots,\mathfrak{X}i_k$ respectively, it replace the number in $\mathfrak{X}j$ by $F(x_1,\ldots,x_k)$ and increases the number in the counter by 1. The <u>Φ–machine</u> is obtained from the basic machine by adding all F–instructions for all F in Φ. We define the notion of a program computing a function for this machine as for the basic machine. A function is <u>recursive</u> <u>in</u> Φ or <u>relative</u> <u>to</u> Φ if it is computed by some program for the Φ–machine. Note that if Φ is empty, recursive in Φ is the same as recursive.

12.1. PROPOSITION. If $\Phi \subseteq \Psi$, then every function recursive in Φ is recursive in Ψ. □

12.2. PROPOSITION. Every function recursive in Φ is recursive in some finite subset of Φ.

Proof. Any program for the Φ–machine can contain F–instructions for only a finite number of F in Φ. □

We will now show that all of the results we have proved remain true if recursive is replaced by recursive in Φ, in some cases under the assumption that Φ is finite. In most cases, no change is required in the proofs. We shall run quickly through these results, except where the presence of Φ makes a significant difference.

First, the results of §4 may be extended from the basic machine to the Φ–machine. This leads to the following result.

12.3. TRANSITIVITY THEOREM. If every function in Ψ is recursive in Φ, then every function which is recursive in Ψ is recursive in Φ.

Proof. Let G be recursive in Ψ and let P be a program for the Ψ–machine which computes it. If P contains a F–instruction, then F is recursive in Φ; so we may replace the instruction by the macro for the Φ–machine with the same format. We thus obtain a program for the macro Φ–machine which computes G; so G is recursive in Φ by 4.2. □

12.4. COROLLARY. If every function in Φ is recursive in Ψ and every function in Ψ is recursive in Φ, then the same functions are recursive in Φ and in Ψ. □

The proof in §5 that the class of recursive functions is recursively closed extends to the relative case. We also have the following result.

12.5. PROPOSITION. Every function in Φ is recursive in Φ.

Proof. A function F in Φ is computed by the program consisting of the one instruction $F(\mathfrak{X}1, \ldots, \mathfrak{X}k) \to \mathfrak{X}0$. □

The extensions of the results of the next few sections depend only on the fact that the class of functions recursive in Φ is recursively closed and includes Φ. All of the results of §6 extend; i.e., all of the symbols proved recursive are recursive in Φ. In addition, names of functions and relations in Φ are recursive in Φ by 12.5. The results of §7 extend.

12.6. PROPOSITION. If Ψ consists of the contractions of the functions in Φ, then the same functions are recursive in Φ and in Ψ.

Proof. By 12.4 and the contraction formulas. □

In §8 we run into a problem; if Φ is too large, we cannot assign codes to all of the F–instructions for the Φ–machine. We shall therefore now suppose that Φ is finite. By 12.6, we may suppose that Φ is a finite sequence $H_1, \ldots, H_m$ of reals. To the H_r–instruction $H_r(\mathfrak{X}i) \to \mathfrak{X}j$ we assign the code $<3,i,j,r>$. Except for this, codes are defined as before.

We can now extend §8 straightforwardly. Since the functions and relations defined there now depend on Φ, we add a superscript Φ to their names. Thus $T_k^{\Phi}(e,\vec{x},y)$ means that e is the code of a program P for the Φ–machine, and y is the code of the P–computation from $\vec{x}$. Then T_k^{Φ} is recursive in Φ. Note, however, that we do not need a U^{Φ}, since the old U still serves.

The Normal Form Theorem now becomes $\{e\}^{\Phi}(\vec{x}) \simeq U(\mu y T_k^{\Phi}(e,\vec{x},y))$. We say that e is a <u>Φ–index</u> of F if $F(\vec{x}) \simeq \{e\}^{\Phi}(\vec{x})$ for all $\vec{x}$. Then a function has a Φ–index iff it is recursive in Φ.

Results which mention indices can be relativized to Φ only for finite Φ, since it is only for finite Φ that Φ–index is defined. Sometimes a result may not mention indices, but its proof may use indices. In this case, all we can immediately say is that the result holds when relativized to a finite Φ. In some cases we can then extend the result to all Φ by using 12.2.

An example where such an extension is possible is 8.3. Our extension says that the class of functions recursive in Φ is the smallest recursively closed class which includes Φ. We can immediately say this is true for finite Φ. Now let Φ be arbitrary and let Ψ be a be a recursively closed class including Φ. We want to show that if F is recursive in Φ, then F is in Ψ. Now F is recursive in a finite subset of Φ. Since Ψ includes this finite subset, F is in Ψ.

We now consider how T_k^{Φ} depends on Φ. When the definitions of §8 are relativized to Φ, the H_r appear explicitly only in the definition of *Reg*. The definition of this function has a new clause for each r:

$$Reg(j,e,x,n+1) = H_r(Reg((i)_1,e,x,n)) \text{ if } (i)_0 = 3 \ \& \ (i)_3 = r \ \& \ (i)_2 = j.$$

This means that in the definition of $T_k^{\Phi}(e,\vec{x},y)$, H_r appears only in contexts $H_r(X)$ where X designates a number appearing in a register during the P–computation from $\vec{x}$ and hence $< y$. Thus we may replace $H_r(X)$ by $(H_r(y))_X$.

If Φ is $H_1,...,H_m$, we write $\overline{\Phi}(z)$ for $\overline{H_1}(z),...,\overline{H_m}(z)$. The above can be summarized as follows: there is a recursive relation $T_{k,m}$ such that

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

Thus if $\{e\}^{\Phi}(\vec{x}) \simeq z$ with computation number y, and $\overline{\Phi}(y) = \overline{\Phi'}(y)$, then $\{e\}^{\Phi'}(\vec{x}) \simeq z$.

13. The Arithmetical Hierarchy

We are now going to study the effect of using unbounded quantifiers in definitions of relations. From now on, we agree that n designates a non–zero number. The results of this section are due to Kleene.

A relation R is <u>arithmetical</u> if it has an explicit definition

$$(1) \qquad R(\vec{x}) \leftrightarrow Qy_1...Qy_nP(\vec{x},y_1,...,y_n)$$

where each Qy_i is either $\exists y_i$ or $\forall y_i$ and P is recursive. We call $Qy_1...Qy_n$ the <u>prefix</u> and $P(\vec{x},y_1,...,y_n)$ the <u>matrix</u> of the definition. We are chiefly interested in the prefix, since it measures how far the definition is from being recursive.

We shall first see how prefixes can be simplified. As z runs through all number, $(z)_0,(z)_1$ runs through all pairs of numbers. It follows that

$$\forall x \forall y R(x,y) \leftrightarrow \forall z R((z)_0,(z)_1)$$

and

$$\exists x \exists y R(x,y) \leftrightarrow \exists z R((z)_0,(z)_1).$$

Using these equivalences, we can replace two adjacent universal quantifiers in a prefix by a single such quantifier, and similarly for existential quantifiers. For example, a definition