functions that we use will be the fact that it is recursively closed. This will enable us to prove in §9 that the class of recursive functions is the smallest recursively closed class.

6. Definitions of Recursive Functions

We are now going to show that certain kind of definitions of functions and relations always lead to recursive functions and relations. The simplest kind of definition of a function has the form $F(\vec{x}) \simeq$ ____, where ____ is an expression which, if defined, represents a number and which contains only previously defined symbols and variables from the sequence $\vec{x}$. Such a definition is called an <u>explicit</u> definition of F <u>in</u> <u>terms</u> <u>of</u> the symbols which appear in ____ .

6.1. PROPOSITION. If F is defined explicitly in terms of variables and names of recursive functions, then F is recursive.

Proof. We suppose that F is defined by $F(\vec{x}) \simeq$ ____ and use induction on the number of symbols in ____ . If ____ consists of just an x_i, then F is an I_i^k and hence is recursive. Otherwise, ____ is $G(X_1,\ldots,X_n)$ where G is recursive. By the induction hypothesis, we may define a recursive function H_i by $H_i(\vec{x}) \simeq X_i$. Then

$$F(\vec{x}) \simeq G(H_1(\vec{x}),\ldots,H_n(\vec{x}));$$

so F is recursive by 5.2. □

The simplest type of definition of a relation has the form $R(\vec{x}) \leftrightarrow$ ____ where ____ is a statement containing only previously defined symbols and variables from the sequence $\vec{x}$. In order to make sure that this defines a relation, we insist that ____ be defined for all values of $\vec{x}$. We call such a definition an <u>explicit</u> definition of R <u>in</u> <u>terms</u> <u>of</u> whatever symbols appear in ____.

6.2. PROPOSITION. If R is defined explicitly in terms of variables and names of recursive functions and relations, then R is recursive.

Proof. The definition must be $R(\vec{x}) \leftrightarrow Q(X_1,\ldots,X_n)$, where Q is a recursive relation. This may be rewritten as

$$\chi_R(\vec{x}) \simeq \chi_Q(X_1,\ldots,X_n).$$

Then R is recursive by 6.1. □

Our intention is to expand the class of symbols which may be used on the right side of explicit definitions of recursive functions and relations. For simplicity, we call such symbols <u>recursive symbols</u>. Thus we have seen that variables and names of recursive functions and relations are recursive symbols. By 5.4, μ is a recursive symbol.

Now we show that the symbols 0, 1, 2, ... are recursive. If 2, say, appears in an explicit definition, we can think of it as a symbol for a 0–ary function applied to zero arguments. Thus we need only show that 2 is a recursive function. Now this function has the explicit definition $2 \simeq Sc(Sc(0))$; since 0 and Sc are recursive, 2 is recursive by 6.1.

6.3. PROPOSITION. Every constant total function is recursive.

Proof. If F is, say, a total function with constant value 2, then F has the explicit definition $F(\vec{x}) \simeq 2$. □

Let ... and ___z___ be expressions which represent numbers and are defined for all values of their variables. Suppose that ... contains no variable other than $\vec{x}$ and that ___z___ contains no variable other than $\vec{x}$, y, and z. We can define a total function F by induction as follows:

$$F(0,\vec{x}) = \ldots,$$

$$F(y+1,\vec{x}) = \_\_\_F(y,\vec{x})\_\_\_.$$

We call this an <u>inductive definition</u> of F in terms of whatever symbols appear in ... and ___z___.

6.4. PROPOSITION. If F has an inductive definition in terms of recursive symbols, then F is recursive.

Proof. Let F be defined as above. We may define recursive function G and H explicitly by

$$G(\vec{x}) \simeq ...,$$
$$H(z,y,\vec{x}) \simeq \_\_\_z\_\_\_.$$

Then

$$F(0,\vec{x}) = G(\vec{x}),$$
$$F(y+1,\vec{x}) = H(F(y,\vec{x}),y,\vec{x}).$$

Hence F is recursive by 5.2. □

We have required that our inductive definitions be by induction on the first argument; but this is not essential. Suppose that we have a definition of $F(x,y)$ by induction on y. If $F'(y,x) = F(x,y)$, we can convert that definition into a definition of F' by induction on y. If only recursive symbols are involved, then F' is recursive. But F has the explicit definition $F(x,y) \simeq F'(y,x)$; so F is recursive.

We now give some inductive definitions of some common functions.

$$0 + x = x,$$
$$(y+1) + x = Sc(y+x),$$
$$0 \cdot x = 0,$$
$$(y+1) \cdot x = (y \cdot x) + x,$$
$$x^0 = 1,$$
$$x^{y+1} = x^y \cdot x.$$

Subtraction is not a total function for us, since we do not allow negative numbers. We therefore introduce a modified subtraction $\dot{-}$ defined by $x \dot{-} y = x - y$ if $x \geq y$, $x \dot{-} y = 0$ otherwise. To show that this is recursive, first define a function Pr inductively by

$$Pr(0) = 0,$$
$$Pr(x+1) = x.$$

Then $\dot{-}$ is defined inductively by

$$x \dot{-} 0 = x,$$
$$x \dot{-} (y+1) = Pr(x \dot{-} y).$$

We recall that if X and Y are statements, then $\neg X$ means not X; $X \vee Y$ means X or Y; X & Y means X and Y; $X \rightarrow Y$ means if X, then Y; and $X \leftrightarrow Y$ means X iff Y. We call $\neg X$ the negation of X; $X \vee Y$ the disjunction of X and Y; and X & Y the conjunction of X and Y. The symbols $\neg$, $\vee$, &, $\rightarrow$, and $\leftrightarrow$ are called propositional connectives. We shall show that all of them are recursive symbols. It is enough to do this for $\neg$ and $\vee$; for we can define X & Y to mean $\neg(\neg X \vee \neg Y)$; $X \rightarrow Y$ to mean $\neg X \vee Y$; and $X \leftrightarrow Y$ to mean $(X \rightarrow Y)$ & $(Y \rightarrow X)$. Thus we must show that if P and P' are recursive, and Q and R are defined by

$$Q(\vec{x}) \leftrightarrow \neg P(\vec{x}),$$
$$R(\vec{x}) \leftrightarrow P(\vec{x}) \vee P'(\vec{x}),$$

then Q and R are recursive. This follows from the explicit definitions

$$\chi_Q(\vec{x}) \simeq 1 \dot{-} \chi_P(\vec{x}),$$
$$\chi_R(\vec{x}) \simeq \chi_P(\vec{x}) \cdot \chi_{P'}(\vec{x}).$$

We shall now show that the relations $\leq$, $\geq$, $<$, $>$, and $=$ are recursive. (By the relation $\leq$, for example, we mean the relation R defined by $R(x,y) \leftrightarrow x \leq y$.) It is easy to see that

$$\chi_>(x,y) \simeq 1 \dot{-} (x \dot{-} y);$$

so $>$ is recursive. The other relations have the explicit definitions:

$$x \leq y \leftrightarrow \neg(y > x)$$
$$x \geq y \leftrightarrow y \leq x,$$
$$x < y \leftrightarrow \neg(y \leq x),$$
$$x = y \leftrightarrow x \leq y \;\&\; y \leq x.$$

6.5. PROPOSITION. Let $R_1,\ldots,R_n$ be recursive relations such that for every $\vec{x}$, exactly one of $R_1(\vec{x}),\ldots,R_n(\vec{x})$ is true. Let $F_1,\ldots,F_n$ be total recursive functions, and define a total function F by

$$\begin{aligned} F(\vec{x}) &= F_1(\vec{x}) \quad \text{if } R_1(\vec{x}), \\ &= \ldots \\ &= F_n(\vec{x}) \quad \text{if } R_n(\vec{x}). \end{aligned}$$

Then F is recursive.

Proof. We have

$$F(\vec{x}) = F_1(\vec{x})\cdot\chi_{\neg R_1}(\vec{x}) + \ldots + F_n(\vec{x})\cdot\chi_{\neg R_n}(\vec{x}). \square$$

We use 6.5 in connection with definitions by cases of functions and relations. For example, suppose that we define F by

$$\begin{aligned} F(x,y) &= x && \text{if } x < y,\\ &= y + 2 && \text{if } y \le x \;\&\; x = 4,\\ &= 3 && \text{otherwise.}\end{aligned}$$

This comes under 6.5 if we define

$$\begin{aligned} F_1(x,y) &\simeq x, && R_1(x,y) \leftrightarrow x < y,\\ F_2(x,y) &\simeq y + 2, && R_2(x,y) \leftrightarrow y \le x \;\&\; x = 4,\\ F_3(x,y) &\simeq 3, && R_3(x,y) \leftrightarrow \neg R_1(x,y) \;\&\; \neg R_2(x,y).\end{aligned}$$

A definition by cases of a relation is easily converted into a definition by cases of its representing function. The conclusion is that if we define a total function or relation by cases in terms of recursive symbols, then the function or relation is recursive. We shall consider definition by cases of partial functions in §8.

6.6. **Proposition.** Let F be a total recursive function, and let G be a total function such that $G(\vec{x}) = F(\vec{x})$ for all but a finite number of $\vec{x}$. Then G is recursive.

Proof. Suppose, for example that F is unary and $G(x) = F(x)$ except that $G(3) = 5$ and $G(7) = 2$. Then we can define G by cases as follows:

$$\begin{aligned} G(x) &= 5 && \text{if } x = 3,\\ &= 2 && \text{if } x = 7,\\ &= F(x) && \text{otherwise. } \square\end{aligned}$$

6.7. **Corollary.** Every finite relation is recursive.

Proof. If R is finite, $\chi_R(\vec{x}) = 1$ for all but a finite number of $\vec{x}$. $\square$

Recall that a unary relation is simply a set. Then 6.7 shows that every finite set is recursive. The complement of a recursive set is recursive; for the complement of A is $\neg A$. The union and intersection of two recursive sets is recursive; for the union of A and B is $A \vee B$, and the intersection of A and B is $A \& B$.

Recall that $\forall x$ means for all x and $\exists x$ means for some x. We call $\forall x$ a universal quantifier and $\exists x$ an existential quantifier. As we shall see in §13, these are not recursive symbols. We introduce some modified quantifiers, called bounded quantifiers, which are recursive. We let $(\forall x < y)X(x)$ mean that $X(x)$ holds for all x less than y, and let $(\exists x < y)X(x)$ mean that $X(x)$ holds for some x less than y. To see that these are recursive, note that

$$(\forall x < y)X(x) \leftrightarrow \mu x(\neg X(x) \vee x = y) = y,$$

$$(\exists x < y)X(x) \leftrightarrow \mu x(X(x) \vee x = y) < y.$$

To allow us to use bounded quantifiers with $\leq$ instead of $<$, we agree that $(\forall x \leq y)$ means $(\forall x < y+1)$ and similarly for $\exists$.

We summarize the results of this section. If a function or a relation has an explicit definition or an inductive definition or a definition by cases in terms of recursive symbols, then it is recursive. Recursive symbols include variables, names of recursive functions and relations, μ, propositional connectives, and bounded quantifiers. The recursive functions include the initial functions, $+$, $\cdot$, x^y, $\dot{-}$, and all constant total functions. The recursive relations include all finite relations, $<$, $>$, $\leq$, $\geq$, and $=$.

7. Codes

Suppose that we wish to do computations with a class I other than ω as our set of inputs and outputs. One approach is to assign to each member of I a number, called the code of that member, so that different codes are assigned to different members. Given some inputs in I, we first replace each input by its code. We then do a computation with these numerical inputs to obtain a