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

numerical output. If this output is the code of a member of I, that member is the final output of the computation.

We want the first and last steps of the above procedure to be performed according to an algorithm. This means that we should do our coding so that there is an algorithm by which we can find the code of a given member of I, and an algorithm by which, given a number, we can decide if it is the code of a member of I and, if it is, find that member. If this is the case, we say that the coding is effective.

We shall now assign a code to every finite sequence of numbers. Let p_0, p_1, ... be the primes in increasing order. To the finite sequence x_0, ..., x_{k-1} we assign the code

$$p_0^{x_0+1} \cdot \ldots \cdot p_{k-1}^{x_{k-1}+1}.$$

(The empty sequence has the code 1.) The theorem on unique decomposition into prime factors shows that different sequences have different codes. (Note that this would not be true if we omitted the +1's in the exponents.) Since there is an algorithm for decomposing a number into prime factors, this coding is effective.

We shall show that some functions and relations associated with this coding are recursive. To do this, we give definitions of these functions and relations which show, by the results of the last section, that they are recursive. We always take our functions to be total, even when we are only interested in them for certain arguments.

We want to define $Div(x,y)$ to mean that x is divisible by y. The obvious definition is $Div(x,y) \leftrightarrow \exists z(x = y \cdot z)$. This does not show that Div is recursive, since unbounded quantifiers are not recursive. We therefore seek a bound for z. This leads to the definition:

$$Div(x,y) \leftrightarrow (\exists z \leq x)(x = y \cdot z).$$

Next we define the set Pr of primes:

$$Pr(x) \leftrightarrow x > 1 \;\&\; (\forall y < x)(y > 1 \to \neg Div(x,y)).$$

We then define by induction

$$p_0 = 2,$$
$$p_{i+1} = \mu x(Pr(x) \;\&\; x > p_i).$$

We define

$$exp(x,i) = \mu y(\neg Div(x,p_i^{y+1}) \vee x = 0),$$

so that $exp(x,i)$ is the exponent of p_i in the decomposition of x. (The clause $\vee\, x = 0$ is there to make the function total; it makes $exp(0,i) = 0$.)

For each k we define

$$<x_0,\ldots,x_{k-1}> = p_0^{x_0+1} \cdot \ldots \cdot p_{k-1}^{x_{k-1}+1}$$

Then $<x_0,\ldots,x_{k-1}>$ is the code of the sequence $x_0,\ldots,x_{k-1}$.

Next we define

$$lh(x) = \mu i(exp(x,i) = 0),$$
$$(x)_i = exp(x,i) \dot{-} 1.$$

If $x = <x_0,\ldots,x_{k-1}>$, then $lh(x) = k$; $(x)_i = x_i$ for $i < k$; and $(x)_i = 0$ for $i \geq k$. Since $z \leq p^z$ for $p > 1$, $exp(x,i) \leq x$. It follows that

$$(1) \qquad x \neq 0 \to (x)_i < x,$$

and hence

$$(2) \qquad x_i < <x_1,\ldots,x_k>.$$

Since $Div(x,p_i)$ implies $i < p_i \leq x$ for $x > 0$, the set Seq of codes of finite sequences is defined by

$$Seq(x) \leftrightarrow x \neq 0 \;\&\; (\forall i < x)(Div(x,p_i) \to i < lh(x)).$$

We define $x * y$ so that $<x_1,\ldots,x_k> * <y_1,\ldots,y_l>$ is $<x_1,\ldots,x_k,y_1,\ldots,y_l>$:

$$x * y \simeq \mu z(Seq(z) \;\&\; lh(z) = lh(x) + lh(y) \;\&$$
$$(\forall i < lh(x))((z)_i = (x)_i) \;\&\; (\forall i < lh(y))((z)_{lh(x)+i} = (y)_i)).$$

As a first application of these codes, we show how to replace k–ary

functions and relations by 1–ary functions and relations. If F is a k–ary function, we define a 1–ary function $<F>$, called the contraction of F, by

$$<F>(x) \simeq F((x)_0,\ldots,(x)_{k-1}).$$

We can recover F from $<F>$ by the equation

$$F(x_1,\ldots,x_k) \simeq <F>(<x_1,\ldots,x_k>).$$

These two equations are called the contraction equations. Considered as explicit definitions, they show that F is recursive iff $<F>$ is recursive. A similar procedure holds for relations; we leave the details to the reader.

As a second application, we consider simultaneous definitions of functions by induction. Suppose we define

$$\begin{aligned}
F_1(0,\vec{x}) &= G_1(\vec{x}),\\
F_2(0,\vec{x}) &= G_2(\vec{x}),\\
F_1(y+1,\vec{x}) &= H_1(y,\vec{x},F_1(y,\vec{x}),F_2(y,\vec{x})),\\
F_2(y+1,\vec{x}) &= H_2(y,\vec{x},F_1(y,\vec{x}),F_2(y,\vec{x})),
\end{aligned}$$

where G_1, G_2, H_1, and H_2 are total and recursive. This defines total functions F_1 and F_2 by induction on y. We shall show that F_1 and F_2 are recursive. It suffices to show that the function F defined by

$$F(y,\vec{x}) = <F_1(y,\vec{x}),F_2(y,\vec{x})>$$

is recursive; for $F_1(y,\vec{x}) = (F(y,\vec{x}))_0$ and $F_2(y,\vec{x}) = (F(y,\vec{x}))_1$. But F has the recursive definition

$$\begin{aligned}
&F(0,\vec{x}) = <G_1(\vec{x}),G_2(\vec{x})>,\\
&F(y+1,\vec{x}) = <H_1(y,\vec{x},(F(y,\vec{x}))_0,(F(y,\vec{x}))_1),\ H_2(y,\vec{x},(F(y,\vec{x}))_0,(F(y,\vec{x}))_1)>.
\end{aligned}$$

As a third application, we introduce a more general form of definition by induction in which the value of the function at y depends on all the previous values. If F is a total $(k+1)$–ary function, we define another total $(k+1)$–ary function $\overline{F}$ by

$$\overline{F}(y,\vec{x}) \simeq <F(0,\vec{x}),\ldots,F(y-1,\vec{x})>.$$

Thus $\overline{F}(y,\vec{x})$ codes a sequence which gives the values of $F(i,\vec{x})$ for $i < y$. We

show that $\overline{F}$ is recursive iff F is recursive. We cannot use the preceding equation as an explicit definition; for we cannot fill in ... until we know the value of the argument y. However, we have the explicit definitions

$$\overline{F}(y,\vec{x}) \simeq \mu z(Seq(z) \;\&\; lh(z) = y \;\&\; (\forall i < y)((z)_i = F(i,\vec{x}))),$$

$$F(y,\vec{x}) \simeq (\overline{F}(y+1,x))_y.$$

Given a total function G, we may define a total function F by induction on y as follows:

$$F(y,\vec{x}) \simeq G(\overline{F}(y,\vec{x}),\, y,\, \vec{x}).$$

We shall show that if G is recursive, then F is recursive. By the above, it is enough to show that $\overline{F}$ is recursive. But $\overline{F}$ has the inductive definition

$$\overline{F}(0,\vec{x}) \simeq <>,$$

$$\overline{F}(y+1,\vec{x}) = \overline{F}(y,\vec{x}) * <G(\overline{F}(y,\vec{x}),\, y\,,\vec{x})>.$$

An inductive definition of this sort is called a course–of–values inductive definition.

8. Indices

We are now going to assign codes to some of the elements in the operation of the basic machine. This will lead to some of the most important theorems of recursion theory.

First, a general remark on coding. Suppose that we want to code the members of I. We may be able to identify each member b of I with a finite sequence $a_1,...,a_k$ of objects which have already been coded. We can then assign to b the code $<x_1,...,x_k>$, where x_i is the code of a_i.

We begin by assigning codes to the instructions for the basic machine. We assign the code $<0,i>$ to the instruction INCREASE $\mathcal{R}i$; the code $<1,i,n>$ to the instruction DECREASE $\mathcal{R}i,n$; and the code $<2,n>$ to the instruction GO TO n. If P is a program consisting of N instructions with codes $x_1,...,x_N$, we assign the code $<x_1,...,x_N>$ to P.

We define