CLASSES OF PREDICTABLY COMPUTABLE FUNCTIONS

BY
ROBERT W. RITCHIE(1)

0. **Introduction.** In this paper we study a sequence of classes of computable functions for which a prediction of the complexity of the calculation may be made in a comparatively simple fashion. The class of recursive functions is accepted as being the class of just those functions for which there is a mechanical procedure for obtaining the values from the arguments. However, this class, since it involves all computations, must involve arbitrarily complex computations. Various more restricted classes of functions have been proposed as being somehow more easily computable, but the greater ease of computation has largely been left on the level of intuition. The notable exception to this is the class F of functions computable(2) by finite automata. However, this class is too severely restricted; it does not even include multiplication. We present a hierarchy of more inclusive classes of functions, each defined as the class of functions whose computational complexity is "predictable" by a function in the preceding class.

It is known that the class F of numerical functions computable by finite automata includes addition and is closed under explicit transformations(3) and composition. In particular then, it contains the linear functions(4). This class, F, is taken as a basic class since these functions are computed "as swiftly as possible;" as soon as the input has been scanned, the output is printed. To get a larger class, we consider the class of functions computable (in binary notation) by Turing machines which use an amount of tape bounded by a simpler function, namely one in F. We call this new class F_1. Continuing in this way, we define F_i to be the class of functions computable by Turing machines which use in their computations an amount of tape bounded by a function in F_{i-1}. That is, we

Received by the editors August 28, 1961.

(1) Parts of this work were carried out while the author held summer positions at IBM Research and Bell Telephone Laboratories. Much of the material appeared in the author's doctoral dissertation at Princeton University where he was a National Science Foundation Predoctoral Fellow.

(2) These functions are conventionally encoded in binary notation, but an encoding in n-ary notation for any n is clearly equally restrictive. For a definition of F see Appendix 2.

(3) In the sense of Smullyan. For a definition see Footnote 7 below.

(4) Containment of the linear functions is implicit in [2, pp. 31 and 41], and this is all we need in what follows. (In fact, if the reader chooses, he may take F to be just the class of linear functions throughout this paper without any loss of generality.) The properties of F which we will use are developed in Appendix 2.

measure the complexity of a computation on a Turing machine in terms of the amount of tape required for the computation. (From this, of course, we can also obtain bounds upon the number of steps in the computation.)

We may interpret the denumerable union $\mathscr{F}$ of the classes F_i in the following way. Any function to be computed by a machine for which the amount of storage required in the computation is a function predictable (on a machine requiring less storage than the one to be used) by methods presently considered practical or by methods which are themselves predictable by present methods, and so on, must be in $\mathscr{F}$. Conversely, every element of $\mathscr{F}$ is in some F_i and so must be computed in this predictible sense with at most i predictions. Thus we refer to $\mathscr{F}$ as the class of predictably computable functions, and view the classes F_i as forming a subdivision in terms of complexity of the class $\mathscr{F}$ of all predictably computable functions.

In investigating $\mathscr{F}$, we first observe that each individual class F_i clearly contains the identity function and the successor function. We then show that it also contains the function f_i where $f_1(x)$ is 2^x and $f_{i+1}(x)$ is defined as $f_1(f_i(x))$. Further, we show that each F_i is closed under explicit transformations and that the union of all the F_i is closed under composition and limited recursion([5]). These facts combine to show that the class $\mathscr{E}$ of elementary functions (Csillag-Kalmar, see [4; 6]) is contained in $\mathscr{F}$. We then proceed to establish the reverse containment, showing that the elementary functions are precisely the predictably computable functions.

The last three sections of this paper are devoted to a more detailed analysis of the classes F_i, and require a new arithmetization of Turing machines which is carried out in an appendix. In these final sections, each of the classes F_i is shown to include the class $\mathscr{E}^2$ defined by Grzegorczyk [3], and each class F_i is characterized without reference to Turing machines.

In the last section, the sequence of classes of relations whose characteristic functions are in the F_i is shown to form a subdivision of the class of all elementary relations. Thus, the concepts developed in this paper yield a classification according to computational complexity not only of functions, but also of relations and decision procedures.

The author wishes to express his special thanks to Professors Alonzo Church and Raymond Smullyan for their guidance during the early stages of this research. He is also greatly indebted to Professors Donald Kreider and Michael Rabin for many stimulating and helpful discussions and to the referee for his extremely worthwhile suggestions about the presentation of this material.

1. **Definition of the classes F_i.** In terms of the usual definitions of Turing machines [1; 4], it is not hard to see that the standard arguments [4, pp. 372, 373] can be mirrored almost exactly to establish Theorem 1 below. The only modi-

([5]) See Lemma 4 and Footnote 10.

fication necessary is merely to avoid wasteful use of tape by erasing all entries which are not to be re-used and closing the resultant gaps. Since this is completely straightforward, we omit the details, and will content ourselves with a mere sketch of the proof of the theorem.

Let us define the *amount of tape used* in a computation $X_1, \ldots, X_q$ as the maximum of the lengths of the tapes t_i appearing in the instantaneous descriptions X_i. Let us further agree to denote the amount of tape used in computing $f(x_1, \ldots, x_n)$ by([6]) $a_f(x_1, \ldots, x_n)$ and the length of the tape bearing the numbers $x_1, \ldots, x_n$ by $l(x_1, \ldots, x_n)$. Adopting the abbreviation $\mathbf{x}$ for $x_1, \ldots, x_n$, we obtain these results:

THEOREM 1. (a) *If h and g are computable, and f is defined by*

$$f(\mathbf{x},\mathbf{y}) = h(\mathbf{x},g(\mathbf{y})),$$

then $a_f(\mathbf{x},\mathbf{y}) = l(\mathbf{x},\mathbf{y}) + \max[l(\mathbf{x}) + a_g(\mathbf{y}),\ a_h(\mathbf{x},g(\mathbf{y}))]$ *where* $l(\mathbf{x}) = \sum_{i=1}^{n} l(x_i) + (n-1)$, *since we adopt the usual convention of writing $x_1, \ldots, x_n$ on tape separated by single blanks.*

(b) *If h,g are computable, and f is defined from them by primitive recursion, i.e. $f(\mathbf{x},0) = g(\mathbf{x})$ and $f(\mathbf{x},s(y)) = h(\mathbf{x},y,f(\mathbf{x},y))$ where $s(y)$ is the successor, $y+1$, of y, then*

$$a_f(\mathbf{x},y) = 2[l(\mathbf{x},y) + \max[\max_{z<y}(a_h(\mathbf{x},z,f(\mathbf{x},z))), a_g(\mathbf{x})]].$$

(c) *For explicit transformations([7]) we have simply that if g is computable and $f(x_1,\ldots,x_n) = g(\xi_1,\ldots,\xi_k)$ then*

$$a_f(x_1,\ldots,x_n) = a_g(\xi_1,\ldots,\xi_k) + l(x_1,\ldots,x_n).$$

Proof. We shall content ourselves with a sketch of part (b). The remaining parts are similar and even less involved. The recursive nature of the evaluation of $f(\mathbf{x},y)$ requires the computation of $h(\mathbf{x},0,g(\mathbf{x}))$, $h(\mathbf{x},1,h(\mathbf{x},0,g(\mathbf{x})))$, and so on. The arguments $\mathbf{x}$ occur in each of these computations of a value of the function h, so it is convenient to assume that the computation of a function by a Turing machine leaves the original arguments unchanged on the tape. That is, we assume that the given machines Z_g and Z_h operate so as to end with both the original inputs and the desired function values on the tape. (This is the convention followed, for example, in [4, p. 361].) With this understanding, we may construct a machine Z_f to begin with $(\mathbf{x},y)$ on its tape and end with $(\mathbf{x},y,f(\mathbf{x},y))$ on its tape.

([6]) This notation is, of course, ambiguous since a_f depends not only upon f but also upon the particular Turing machine chosen to compute f. Thus when we write $a_f(x) = g(x)$ we mean simply that there is a Turing machine which computes f for which the amount of tape used in the computation of $f(x)$ is $g(x)$.

([7]) This concept, introduced by Smullyan [8, p. 21], is used freely throughout the paper. For completeness, we include the definition: The function f is said to be defined from the function g by an explicit transformation if f is $\lambda(x_1, \ldots, x_n)g(\xi_1, \ldots, \xi_n)$ (or, equivalently, $f(x_1, \ldots, x_n) = g(\xi_1, \ldots, \xi_k)$ for all $(x_1, \ldots, x_n)$) where for each $i = 1,2, \ldots, k$, either $\xi_i = x_j$ for some $j = 1,2, \ldots, n$ or ξ_i is a constant.

We include here the sequence of tapes in the machine at various critical stages in this computation and leave the tiring but straightforward details of the construction of such a machine to the reader. (The sequence below is written assuming $y \geqq 2$; if y is 0 or 1 the computation simply stops sooner.)

* | x | y | y | x | g(x) | Copy y, copy x, compute $g(\mathbf{x})$.

* | x | y | y | x | g(x) | 0 | g(x) | Print 0, copy $g(\mathbf{x})$.

| x | y | y | x | 0 | g(x) | Erase the leftmost $g(\mathbf{x})$, close the gap.

| x | y | y | x | 0 | g(x) | h(x,0,g(x)) | Since y is not 0, compute $h(\mathbf{x},0,g(\mathbf{x})) = f(\mathbf{x},1)$.

| x | y | y | x | 1 | f(x,1) | Compute the successor of 0, erase $g(\mathbf{x})$, close the gap.

| x | y | y−1 | x | 1 | f(x,1) | h(x,1,f(x,1)) | Subtract 1 from y, since $y - 1 \neq 0$ compute $h(\mathbf{x},1f(\mathbf{x},1)) = f(\mathbf{x},2)$.

⋮

* | x | y | y−k | x | k | f(x,k) | h(x,k,f(x,k)) | Continue...

⋮

| x | y | 1 | x | y−1 | f(x,y−1) | h(x,y−1,f(x,y−1)) | $\{h(\mathbf{x},y-1,f(\mathbf{x},y-1)) = f(\mathbf{x},y)\}$

| x | y | 1 | x | y | f(x,y) | Compute the successor of $y-1$, erase $f(\mathbf{x},y-1)$, close gap.

| x | y | f(x,y) | Subtract 1 from 1, observe the result is 0, erase and close.

In the computation indicated above, let k be an integer between 0 and y such that $a_h(\mathbf{x},k,f(\mathbf{x},k)) = \max_{0 \leqq z \leqq y-1} a_h(\mathbf{x},z,f(\mathbf{x},z))$. The largest amount of tape used occurs in obtaining one of the three starred tapes above. If it occurs at the first star, it is $l(\mathbf{x},y) + l(y) + a_g(\mathbf{x}) + 2$, where the 2 is the number of blanks separating numbers on the tape which have not already been counted in $l(\mathbf{x},y)$. If it occurs at the second starred tape, it is $2[l(\mathbf{x},y) + l(g(\mathbf{x}))] + 3$, and if it occurs at the last, it is $l(\mathbf{x},y) + l(y-k) + a_h(\mathbf{x},k,f,(\mathbf{x},k)) + 2$. Noting that $a_f(\mathbf{z}) \geqq l(\mathbf{z}) + l[F(\mathbf{z})] + 1$, for any function $F(\mathbf{z})$, we may replace each of the first two cases by the upper bound $2[l(\mathbf{x},y) + a_g(\mathbf{x})]$, and the third case by $2l(\mathbf{x},y) + a_h(\mathbf{x},k,f(\mathbf{x},k))$. Thus in all cases the amount of tape used is less than or equal to the bound given in part (b), as was to be shown.

Now we wish to use these observations on the amounts of tape used in computing various numerical functions to classify these functions. As we have already observed, the functions in F are simple; not only is their computation very rapid, but also it uses only enough storage (tape) to hold the output (a fixed constant number of squares more than that used to hold the input). In Appendix 2 we include a summary of the necessary results about finite automata and the class F. The reader who is not familiar with finite automata may wish to consult this appendix.

Thus we choose the class F of functions computable by finite automata (in binary) as a basic class, and we use these functions to generate larger classes of functions requiring more tape and more time for their computation. We consider the class F_1 of all numerical functions f computable by Turing machines for which the function a_f giving the amount of tape used in the computation of f is bounded by a function in F. That is, F_1 is the class of all functions f such tha a_f is bounded by a function which is computable by a finite automaton.

Two comments are in order at this point. The first is that we shall assume that our functions are encoded in binary notation on the Turing machines, rather than in the unary notation more commonly used in the theory of computability. This agrees more closely with actual computing practice and thus gives more natural classes F_i; it has the further advantage that these classes are much larger than they would have been had we used the unary notation([8]). The second comment is that we only require a_f to be bounded by a function f in F; we do not require a_f itself to be in F. This not only simplifies our subsequent proofs([9]) but also seems to agree more closely with actual practice. In practice one obtains an upper bound on the amount of storage required for a computation, but not necessarily the exact amount required.

With these conventions in mind, the reader can easily convince himself that multiplication and the function taking x to a^x for any fixed constant a are in F_1. However, the function taking x to 2^{2^x} is not in F_1. For the length of 2^{2^x} is $2^x + 1$ (and hence the number of tape squares used is at least this large). But $2^x + 1$ is not a function in F, since it violates the final theorem of Appendix 2. Yet 2^x is in F_1, thus 2^{2^x} has its length (and, as a moment's reflection will show, the

---

([8]) For example, in unary the length of the product of two numbers is this product itself, where in binary the length is at most the sum of the lengths of the numbers. Thus multiplication is in F_1 since we are representing numbers in binary notation. It is easy to see that multiplication would not have been in F_1 had we encoded in unary notation. (Actually, any n-ary notation for $n > 1$ will give exactly the same classes F_i, as the reader may easily check.)

([9]) Actually, this involves no loss of generality. For if we are given a Turing machine Z which computes f for which a_f is bounded by $g \in F$, it is easy to construct a machine Z' computing f for which $a_f = 2 \cdot g$. Merely have Z' first copy the arguments $\mathbf{x}$, compute $g(\mathbf{x})$ and print on $2 \cdot g(\mathbf{x})$ squares, then erase all but the argument, and finally imitate Z. Similar comments hold for all the classes F_i.

amount of tape necessary to compute it from x) in F_1. We now define the class F_2 of all functions f such that a_f is in F_1, and so on.

Precisely, we make the following definition.

DEFINITION 1. A numerical function f is said to be in the class F_1 if and only if

(i) it is computable

and

(ii) there is a function $g \in F$ such that $a_f(x_1,\ldots,x_n) \leqq g(x_1,\ldots,x_n)$ for all n-tuples $(x_1,\ldots,x_n)$ of non-negative integers (in short, a_f is bounded by a function in F).

For each $i > 1$, F_i is the class of all computable functions f such that a_f is bounded by a function in F_{i-1}. We call the union of all classes F_i the class of *predictably computable functions* and denote it by $\mathscr{F}$.

Trivially, we may see that F_i is properly contained in F_{i+1} for all $i \geqq 1$ since $2^x \in F_1$, 2^{2^x} is in F_2 but not F_1,

$$2^{2^{2^x}} \text{ is in } F_3 \text{ but not } F_2,$$

and so on.

In the next two sections we investigate the classes F_i of increasing complexity and increasing tape requirements, and show that their union $\mathscr{F}$ is precisely the class of elementary functions.

2. **Properties of the classes F_i.** In this section we shall establish closure properties of the classes F_i with respect to explicit transformations, composition and recursive definitions. With respect to the first of these, we have the following simple result.

LEMMA 1. *Each class F_i is closed under explicit transformations.*

Proof. If $f(x_1,\ldots,x_n) = g(\xi_1,\ldots,\xi_m)$ for $g \in F_i$, then $a_f(x_1,\ldots,x_n) = a_g(\xi_1,\ldots,\xi_m) + l(x_1,\ldots,x_n)$ where $a_g \in F_{i-1}$. However, the function a'_g defined by

$$a'_g(x_1,\ldots,x_n) = a_g(\xi_1,\ldots,\xi_m)$$

is an explicit transformation of a_g and $a_g \in F_{i-1}$ if $i > 1$, or $a_g \in F$ if $i = 1$. Thus, by induction, since F is closed under explicit transformations, so is each F_i.

With respect to composition, Lemma 2 below says that we may compose functions at will, as long as we remember that the result will be in a high enough class in the hierarchy. For example,

$$2^x \in F_1, \quad 2^{2^x} \in F_2, \quad 2^{2^{2^x}} \in F_3,$$

and so on.

LEMMA 2. *For any positive integers i and j, if $h \in F_i$ and $g \in F_j$, then $f \in F_{i+j}$ where $f(\mathbf{x},\mathbf{y}) = (h(\mathbf{x},g(\mathbf{y}))$.*

Proof. Let us first establish the fact that if g and g' are in F_j then so is $f'(\mathbf{x},\mathbf{y}) = g(\mathbf{x}) + g'(\mathbf{y})$. Observe that the amount of tape needed to add two

numbers is merely the amount needed to store them; we may write the sum directly over the arguments digit by digit. Thus $a_{f'}(\mathbf{x},\mathbf{y}) = a_g(\mathbf{x}) + a_{g'}(\mathbf{y})$ since to compute $f'(\mathbf{x},\mathbf{y})$ we may first compute $g(\mathbf{x})$ and $g'(\mathbf{y})$ and then add the two results. Thus, if F_{j-1} is closed under addition, so is F_j. Since F is closed under addition (by Appendix 2), we have the result by induction. Further, if g is in F_j and h is in F, then $f'(\mathbf{x},\mathbf{y}) = h(\mathbf{x},g(\mathbf{y}))$ is in F_j. For, by Appendix 2, $a_h(\mathbf{x},g(\mathbf{y})) \leqq l(\mathbf{x}) + l(g(\mathbf{y})) + K$. But this is at most $l(\mathbf{x}) + a_g(\mathbf{y}) + K$ which is in F_{j-1}, thus $f' \in F_j$ since to compute $f'(\mathbf{x},\mathbf{y})$ we may first compute $g(\mathbf{y})$ and then compute h applied to $\mathbf{x}$ and $g(\mathbf{y})$.

Now let us turn to the proof of the lemma itself. First consider the case in which $i = 1$ and j is arbitrary. We have $a_f(\mathbf{x},\mathbf{y}) \leqq a_h(\mathbf{x},g(\mathbf{y})) + a_g(\mathbf{y}) + l(\mathbf{x})$ where $a_h \in F$ and $a_g \in F_{j-1}$ by Theorem 1b. By the preceding remarks we know the right member of this inequality is in F_j thus $f \in F_{j+1}$ as desired. Now by induction we assume that the lemma holds for $i = k$, j arbitrary, and that $h \in F_{k+1}$. Then we have $a_h \in F_k$ so that by the inductive assumption $a_h(\mathbf{x},g(\mathbf{y})) \in F_{k+j}$ and thus $f(\mathbf{x},\mathbf{y}) \in F_{(k+1)+j}$, completing the proof.

To have a notation for the functions 2^x, 2^{2^x}, etc., let us make the following definition.

DEFINITION 2. Let $f_0(x) = x$ and $f_{i+1}(x) = 2^{f_i(x)}$ for all $i \geqq 1$.

Now let us show that not only is f_i in F_i, but, further, each function in F_i is bounded by f_i in the following sense. If f is in F_i then for some constant K $f(\mathbf{x}) < f_i(K \max(\mathbf{x},1))$, where $\max(\mathbf{x},1)$ is the largest of the $n+1$ numbers $x_1, x_2, \ldots, x_n, 1$ (with $\mathbf{x} = x_1, \ldots, x_n$).

LEMMA 3. *For each $i > 1$, and each $f \in F_i$ there is an integer K such that $f(\mathbf{x}) < f_i(K \max(\mathbf{x},1))$ and further, $a_f(\mathbf{x}) < f_{i-1}(K \max(\mathbf{x},1))$.*

Proof. For $f \in F_1$, $a_f(\mathbf{x}) < K \max(\mathbf{x},1)$ by the final theorem of Appendix 2. Thus

$$f(\mathbf{x}) < 2^{l[f(\mathbf{x})]} < 2^{a_f(\mathbf{x})} < 2^{k \max(\mathbf{x},1)} = f_1(K \max(\mathbf{x},1)).$$

Now we assume the result for F_i and prove it holds for F_{i+1}. Thus, for $f \in F_{i+1}$, $a_f(\mathbf{x}) < f_i(K \max(\mathbf{x},1))$. Then we have also that

$$f(\mathbf{x}) < f_1(l[f(\mathbf{x})]) < f_1(a_f(\mathbf{x})) < f_1(f_i(K \max(\mathbf{x},1))) = f_{i+1}(K \max(\mathbf{x},1)),$$

which completes the proof of this lemma.

We may now establish Lemma 4. This says that *limited* recursion([10]) does not lead outside of $\mathscr{F}$. In fact, it specifies the particular class F_i in which a function introduced by limited recursion will lie.

([10]) We clearly cannot hope for the closure of $\bigcup_{i=1}^{\infty} F_i$ under under primitive recursion; our scheme for computing a function f defined by primitive recursion requires an amount of tape dependent upon f itself, since the term $a_h(\mathbf{x},z,f(\mathbf{x},z))$ appears. However, since we have a bound on f, we are able to keep a_f bounded as well. Such a bound for a function defined by primitive recursion has already been introduced by Grzegorczyk [3] in this notion of limited recursion.

LEMMA 4. *For any positive integers i and k, if* $h \in F_i$, $j \in F_k$ *and* $g \in F_{i+k}$, *then* $f \in F_{i+k}$ *where f is defined from g, h and j by limited recursion. (We follow Grzegorczyk to say that f is defined from g, h and j by limited recursion if* $f(\mathbf{x},0) = g(\mathbf{x})$, $f(\mathbf{x},s(y)) = h(\mathbf{x},y,f(\mathbf{x},y))$ *and* $f(\mathbf{x},y) \leqq j(\mathbf{x},y)$; *that is f is defined by primitive recursion from g and h and further f is pointwise bounded by* $j(^{10})$.)

Proof. By Theorem 1b,

$$a_f(\mathbf{x},y) \leqq 2[\max(\max_{z<y} a_h(\mathbf{x},z,f(\mathbf{x},z)),\ a_g(\mathbf{x})) + l(\mathbf{x},y)].$$

By Lemma 3 and the hypothesis that $f(\mathbf{x},z) \leqq j(\mathbf{x},z)$, we have then

$$a_f(\mathbf{x},y) \leqq 2[\max[\max_{z<y} f_{i-1}(K \max(\mathbf{x},z,f_k(K' \max(\mathbf{x},z)))),\ a_g(\mathbf{x})] + l(\mathbf{x},y)].$$

Thus we obtain

$$\begin{aligned} a_f(\mathbf{x},y) &\leqq 2[\max[f_{i-1}(K \max(\mathbf{x},y,f_k(K' \max(\mathbf{x},y)))),\ a_g(\mathbf{x})] + l(\mathbf{x},y)] \\ &\leqq 2[f_{i-1}(Kf_k(K'(\max(\mathbf{x},y)))) + a_g(\mathbf{x}) + l(\mathbf{x},y)]. \end{aligned}$$

We now show that this bounds a_f by a function in $F_{(i+k)-1}$ and thus complete the proof. First, since $f_1(K' \max(\mathbf{x},y)$ is clearly in F_1, Lemma 2 yields the containment of $f_k(K' \max(\mathbf{x},y))$ in F_k. Since F_k is closed under addition, $K \cdot f_k(K' \max(\mathbf{x},y))$ is in F_k, and Lemma 2 shows that the composition of f_{i-1} with this function is in $F_{(i-1)+k}$. Closure of $F_{(i-1)+k} = F_{(i+k)-1}$ under addition then establishes the desired result.

Combining our results we have the following theorem.

THEOREM 2. *The class* $\mathscr{E}$ *of elementary functions is contained in the union* $\mathscr{F}$ *of the sequence of classes* F_i.

Proof. Grzegorczyk has shown [3, Theorem 2.5] that $\mathscr{E}$ is the smallest class of functions containing the successor function and the exponential function x^y which is closed under explicit transformations, composition and limited recursion(^{11}). We have observed that each F_i contains the successor function, and Lemmas 1, 2 and 4 give us closure of $\bigcup_{i=1}^{\infty} F_i$ under the three necessary operations. Thus we need only show that the exponential function is in $\bigcup_{i=1}^{\infty} F_i$ to establish the theorem. In fact, x^y is in F_2 (use one limited recursion on $x \cdot y$ with $2^{2^{x+y}}$ as a bound—we omit the straightforward details).

3. **The equality of $\mathscr{F}$ and $\mathscr{E}$.** We have just shown that $\mathscr{E} \subseteq \bigcup_{i=1}^{\infty} F_i = \mathscr{F}$. We now turn to establishing the reverse inclusion. To do this, we shall arithmetize the classes F_i within $\mathscr{E}$.

(^{11}) Grzegorczyk did not use the notion of explicit transformations, but it is easily seen that this is merely the appropriate rewording of his Theorem 2.5.

In [1, pp. 56–62], an arithmetization of Turing machines is carried out which results in an elementary relation([12]) T_n for each positive integer n and in an elementary function([12]) U. We must modify this arithmetization since in [1] numerical functions are encoded as strings of 1's and any output is considered as representing a number (n if it contains n 1's distributed in any way along it). Our convention, on the other hand, is to encode numbers as strings of 0's and 1's (in binary notation) and requires that an output tape consist exactly of the binary representation of a number before it is interpreted as signifying a numerical output.

Fortunately, the modification of this arithmetization to suit our needs is very simple. We need only change the functions MR and Corn and the relation Fin. The resulting arithmetization yields elementary U and T_n such that an n-argument function f is computable (in binary) if and only if there is a number z_0 such that for all $x_1, \ldots, x_n$

$$f(\mathbf{x}) = U(\min y[T_n(z_0,\mathbf{x},y)]).$$

To accomplish this, we redefine MR so that MR(n) is the Gödel number of the tape which consists exactly of the binary representation of n; we redefine Corn so that Corn(x) = n if and only if MR(n) = x; and we redefine Fin so that Fin(x,z) is true if and only if both Fin(x,z) holds in Davis' sense and also the tape occurring in description number x is a binary representation of a number. The elementary definitions of MR, Corn, and Fin are given in Appendix 1.

Now assume the n-argument numerical function f is in F_i. Then in particular it is computable, so we have a z_0 for which

$$f(\mathbf{x}) = U(\min y[T_n(z_0,\mathbf{x},y)]).$$

In order to show that f is elementary, it will suffice to show that y, viewed as a function of $\mathbf{x}$, is bounded by an elementary function g. For then

$$f(\mathbf{x}) = U(\min y \leqq g(\mathbf{x})[T_n(z_0,\mathbf{x},y)])$$

where the right member is elementary([12]).

To show that there is an elementary function g which bounds y, we proceed as follows. The maximum number of tape squares which can occur in any step of the computation on the given machine Z is $a_f(\mathbf{x})$. By Lemma 3, this is less than $h(\mathbf{x})$ where

$$h(\mathbf{x}) = f_{i-1}(K(\max(\mathbf{x}))),$$

([12]) Davis only notes in [1] that T_n and U are primitive recursive. However, they are elementary since all minimalizations and quantifications are bounded. See [3, pp. 8, 9, and 11] for these closure properties of elementary functions and relations. (We may define the elementary relations as those relations whose characteristic functions are elementary. This is easily equivalent to the definition in [3].)

and by [3, pp. 8 and 12] h is elementary. On the machine Z there is a finite number, m, of symbols which are allowed to appear on tape. Thus the number of different tapes which can occur in the computation is less than $m^{h(\mathbf{x})}$. There is also a fixed finite number of internal configurations of Z, say k. Hence the total number of instantaneous descriptions of Z (and thus the number of steps([13])) which can occur in the given computation is less than $j(\mathbf{x}) = k \cdot h(\mathbf{x}) \cdot m^{h(\mathbf{x})}$, an elementary function of $\mathbf{x}$. Now suppose that $q(\mathbf{x})$ is the maximum Gödel number of any instantaneous description which occurs in this computation. Then our bound $g(\mathbf{x})$ is given by([14]) $[\mathrm{Pr}(j(\mathbf{x}))^{q(\mathbf{x})}]^{j(\mathbf{x})}$ since it exceeds the number of any sequence of length $j(\mathbf{x})$ on which Gödel number of each term is less than $q(\mathbf{x})$. Hence g is elementary if q is. To see that q is elementary, we first let M be the maximum Gödel number of any internal configuration of Z and of any symbol which occurs on tapes of Z. We may take

$$q(\mathbf{x}) = [\mathrm{Pr}(h(\mathbf{x}) + 1)^M]^{h(\mathbf{x})+1} .$$

Thus g is elementary. As we have stated, this suffices to show that the given arbitrary function f in F_i is elementary.

We have thus shown that $\mathscr{F} = \bigcup_{i=1}^{\infty} F_i \subseteq \mathscr{E}$. Together with Theorem 2 this establishes our main result.

THEOREM 3. *The class $\mathscr{F}$ of predictably computable functions is precisely the class of elementary functions: that is $\mathscr{F} = \mathscr{E}$.*

4. **The classes F' and $\mathscr{E}^2$.** The class $\mathscr{F}$ of predictably computable functions is defined in terms of the complexity of computations by Turing machines. However, by proving this class equal to the class $\mathscr{E}$ of elementary functions, we have characterized $\mathscr{F}$ in purely number theoretic terms without any reference to Turing machines. In this section we lay the groundwork for a similar purely number theoretic characterization of each individual class F_i.

In the previous section we obtained some idea of the rates of growth and closure properties of the classes F_i. However, the results obtained so far are clearly not sufficient to characterize these individual classes. For example, consider the function h in F_i defined by $h(x,y) = 2^x y$. Lemma 2 guarantees that the result of substituting f_1 into either argument of h yields a function in F_2, but in fact we can easily show that $f \in F_1$ where $f(x,y) = 2^x 2^y = h(x,2^y)$. Of course this holds because h depends only "multiplicatively" rather than "exponentially" on its second argument; more precisely, because a_h is of the form $a_h(x,y) = q(x) + Kl(y)$ for some $q \in F$, $K \geqq 0$. In general, we have the following result.

([13]) If there were more than $j(\mathbf{x})$ steps in the computation, Z would have reached the same instantaneous description twice and thus would be repeating a cycle of instantaneous descriptions and could never stop.

([14]) Following [1] we denote the nth prime by $\mathrm{Pr}(n)$, setting $\mathrm{Pr}(0) = 0$.

LEMMA 5. *Assume that the function $h \in F_i$ has the property that there is a $q \in F_{i-1}$ such that $a_h(\mathbf{x},y) \leqq q(\mathbf{x}) + Kl(y)$. Then the function f defined by $f(\mathbf{x},\mathbf{z}) = h(\mathbf{x},g(\mathbf{z}))$ is in F_i if g is in F_i. Further, if g, h and j are in F_i and $a_h(\mathbf{x},y,z) \leqq q(\mathbf{x},y) + Kl(z)$ with $q \in F_{i-1}$, then the function f defined from them by limited recursion is in F_i.*

Proof.

$$\text{(a)} \qquad \begin{aligned} a_f(\mathbf{x},\mathbf{z}) &\leqq a_h(\mathbf{x},g(\mathbf{z})) + a_g(\mathbf{z}) + l(\mathbf{x}) \\ &\leqq q(\mathbf{x}) + Klg(\mathbf{z}) + a_g(\mathbf{z}) + l(\mathbf{x}) \\ &\leqq q(\mathbf{x}) + (K+1)a_g(\mathbf{z}) + l(\mathbf{x}), \end{aligned}$$

where $q, a_g \in F_{i-1}$ and thus $f \in F_i$.

$$\text{(b)} \qquad \begin{aligned} a_f(\mathbf{x},y) &\leqq 2[\max[\max_{z<y}(a_h(\mathbf{x},z,f(\mathbf{x},z))), a_g(\mathbf{x})] + l(\mathbf{x},y)] \\ &\leqq 2[\max[\max_{z<y}(q(\mathbf{x},z) + Klf(\mathbf{x},z)), a_g(\mathbf{x})] + l(\mathbf{x},y)] \\ &\leqq 2[\max[\max_{z<y}(q(\mathbf{x},z) + Klj(\mathbf{x},z)), a_g(\mathbf{x})] + l(\mathbf{x},y)] \\ &\leqq 2[\max[\max_{z<y}(q(\mathbf{x},z) + Ka_j(\mathbf{x},z)), a_g(\mathbf{x})] + l(\mathbf{x},y)]. \end{aligned}$$

At this point we use Lemma 3 to obtain monotone functions and continue, obtaining

$$\begin{aligned} a_f(\mathbf{x},y) &\leqq 2[\max[\max_{z<y}(f_{i-1}(K_1\max(\mathbf{x},z) + Kf_{i-1}(K_2\max(\mathbf{x},z)))), a_g(\mathbf{x})] + l(\mathbf{x},y)] \\ &\leqq 2[\max[f_{i-1}(K_1\max(\mathbf{x},y)) + Kf_{i-1}(K_2\max(\mathbf{x},y)), a_g(\mathbf{x})] + l(\mathbf{x},y)]. \end{aligned}$$

Since $\max(F,G) \leqq F + G$, we have $f \in F_i$ as was to be shown.

Our characterization of each F_i will make heavy use of functions with the "multiplicative" property occurring in the hypotheses of Lemma 5. We now concentrate our attention on a particularly useful subclass F' of F_1 (which we will show is precisely the class $\mathscr{E}^2$([15])) all of whose elements have this "multiplicative" property.

DEFINITION 3. F' is the class of all computable functions f for which the amount of tape used in computing $f(x_1,\ldots,x_n)$ is less than or equal to a constant K_f times $l(x_1,\ldots,x_n)$. That is, for all $x_1,\ldots,x_n$

$$\begin{aligned} a_f(x_1,\ldots,x_n) &\leqq l(x_1\beta\ldots\beta x_n x_1\beta\ldots\beta x_n\ldots x_1\beta\ldots\beta x_n) \\ &= K_f\cdot(l(x_1) + l(x_2) + \ldots + l(x_n) + (n-1)). \end{aligned}$$

Let us begin by obtaining various closure properties of F'.

LEMMA 6. *The class F' of functions is closed under explicit transformations, composition, and limited recursion.*

([15]) The class $\mathscr{E}^2$ was originally introduced in [3]. We repeat the definition in Theorem 4, using the terminology of explicit transformations.

Proof. (a) Suppose $g \in F'$, $a_g(x_1, \ldots, x_m) \leqq k_g l(x_1, \ldots, x_m)$, and $f(x_1, \ldots, x_n) = g(\xi_1, \ldots, \xi_m)$. Then by Theorem 1,

$$a_f(x_1, \ldots, x_n) \leqq k_g l(\xi_1, \ldots, \xi_m) + l(x_1, \ldots, x_n)$$

and since $l(\xi_1, \ldots, \xi_m) \leqq c l(x_1, \ldots, x_n)$ (for some constant c depending upon which ξ_i are constants and which are variables), we have that

$$a_f(x_1, \ldots, x_n) \leqq (k_g c + 1)\, l(x_1, \ldots, x_n).$$

Thus k_f may be taken as $k_g c + 1$.

(b) Suppose $g, h \in F'$, $a_g, a_h \leqq k_g l, k_g l$ respectively, and f is defined by $f(\mathbf{x},\mathbf{y}) = h(\mathbf{x}, g(\mathbf{y}))$. Now, by Theorem 1 we have (using $l(\mathbf{x},\mathbf{y}) = l(\mathbf{x}) + l(\mathbf{y}) + 1$ and $lg(\mathbf{y}) \leqq a_g(\mathbf{y}) \leqq k_g l(\mathbf{y})$)

$$\begin{aligned} a_f(\mathbf{x},\mathbf{y}) &\leqq [l(\mathbf{x}) + l(\mathbf{y}) + 1] + \max[l(\mathbf{x}) + k_g l(\mathbf{y}),\ k_h(l(\mathbf{x}) + lg(\mathbf{y}) + 1)] \\ &\leqq [l(\mathbf{x}) + l(\mathbf{y}) + 1] + k_h[l(\mathbf{x}) + k_g l(\mathbf{y}) + 1] \\ &\leqq [(k_h + 1)k_g] l(\mathbf{x},\mathbf{y}) \end{aligned}$$

so that k_f may be taken as $(k_h + 1)k_g$.

(c) Let g, h, j be in F' with k_g, k_h, k_j their associated constants. Let f be defined from them by limited recursion. Then by Theorem 1,

$$\begin{aligned} a_f(\mathbf{x},y) &\leqq 2[\max[\max_{z<y}(k_h l(\mathbf{x},z,f(\mathbf{x},z))), k_g l(\mathbf{x})] + l(\mathbf{x},y)] \\ &\leqq 2[\max[\max_{z<y}(k_h l(\mathbf{x},z) + k_h k_j l(\mathbf{x},z)) + k_h,\ k_g l(\mathbf{x})] + l(\mathbf{x},y)] \\ &\leqq 2[\max[k_h l(\mathbf{x},y) + k_h k_j l(\mathbf{x},y) + k_h, k_g l(\mathbf{x})] + l(\mathbf{x},y)] \\ &\leqq 2[\max(k_h(k_j + 2), k_g) + 1] \cdot l(\mathbf{x},y). \end{aligned}$$

Thus we may take k_f as $2\max[(k_h(k_j + 2), k_g) + 1]$.

As an example of the use of this lemma, we may show that addition and multiplication are functions in F'. The successor function is clearly computable by a Turing machine using precisely one more square than the number needed to hold the input. Furthermore, concatenation([16]) is obviously in F', and addition is definable from the successor function by recursion limited by concatenation. Also multiplication is definable from addition by recursion limited by concatenation, so that addition and multiplication are in F'.

To summarize our information about the class F', we state the following theorem; it is merely a condensation of Lemma 6 and the preceding paragraph.

([16]) (Binary) concatenation is the function which, applied to the numbers represented in binary as $a_n a_{n-1} \ldots a_1 a_0$ and $b_m b_{m-1} \ldots b_1 b_0$, yields the number whose binary representation is $a_n a_{n-1} \ldots a_1 a_0 b_m b_{m-1} \ldots b_1 b_0$. For example, the concatenation of 10 and 11 is 1011, while that of 0 and 11 is 011; the number three.

THEOREM 4. *F' contains the class $\mathscr{E}^2$ (where $\mathscr{E}^2$ is the smallest class containing the successor function and multiplication which is closed under explicit transformations, composition and limited recursion*([15])).

Now let us characterize the class F' without reference to Turing machines or bounded computations. We have just shown that $\mathscr{E}^2$ is a subclass of F'. The remainder of this section is devoted to arithmetizing F' within $\mathscr{E}^2$ and hence establishing the equality of these two classes. Since the definition of $\mathscr{E}^2$ is purely number theoretic, this gives the desired characterization of F'.

We begin by considering the bound on the computation of a function f imposed by the requirement that the function be in F'. To fix the notations in this discussion, let us assume that the machine which computes f has Gödel number z, has k internal states (configurations), and employs three tape symbols([17]). If the arguments are $x_1, \ldots, x_n$, then the length of the longest tape which can occur is $K \cdot l(x_1, \ldots, x_n)$ (by the definition of F'). Since each square of tape can contain three different symbols, the number of possible tapes which can occur in the computation of $f(x_1, \ldots, x_n)$ is at most $3^{K \cdot l(x_1,\ldots,x_n)}$. Thus, the number of instantaneous descriptions possible is less than $3^{K \cdot l(x_1,\ldots,x_n)} \cdot k \cdot l(x_1, \ldots, x_n)$. However, we may replace this bound by

$$[(x_1 + 1) \ldots (x_n + 1)]^{2 \cdot K} \cdot 2^{2 \cdot K \cdot (2n-1)} \cdot k \cdot [x_1 + \ldots + x_n + (n-1)]$$

which we will denote $b_{n,z}(x_1, \ldots, x_n)$, since

$$\begin{aligned} 3^{K \cdot l(x_1,\ldots,x_n)} &< [2^{l(x_1,\ldots,x_n)}]^{2 \cdot K} = [2^{lx_1} \ldots 2^{lx_n}]^{2 \cdot K} \cdot [2^{n-1}]^{2 \cdot K} \\ &< [2n(x_1 + 1) \ldots (x_n + 1)]^{2 \cdot K} \cdot 2^{2 \cdot K \cdot (n-1)}. \end{aligned}$$

Since $b_{n,z}$ is formed using only addition and multiplication, the function $b_{n,z}(x_1, \ldots x_n)$ is clearly in $\mathscr{E}^2$.

We arrange our arithmetization to make use of the containment of $b_{n,z}$ in $\mathscr{E}^2$ as follows. We replace the usual representation of $f(x_1, \ldots, x_n)$ as

$$U(\min y T_n(z, x_1, \ldots, x_n, y))$$

for this fixed value of z ([18]) by

$$U'_{n,z}(x_1, \ldots, x_n, \min y T'_{n,z}(x, \ldots, x_n, y)).$$

Here the relation $T'_{n,z}$ holds of $x_1, \ldots, x_n, y$ if and only if y is the *number of* instantaneous descriptions of the computation on machine number z beginning with $(x_1, \ldots, x_n)$. Further, $U'_{n,z}(x_1, \ldots, x_n, y)$ is the number on the tape in the yth

([17]) In Appendix 2 we formulate definitions of Turing machines operating over an alphabet of three symbols.

([18]) Thus we lose the "normal form" property of the usual approach. However, this is necessarily the case, as a standard diagonalization argument will show (see §6).

instantaneous description of this computation (if the tape represents a number and 0 otherwise). Since, by the above, $T'_{n,z}(\mathbf{x},y)$ implies $y < b_{n,z}(\mathbf{x})$, we have

$$f(\mathbf{x}) = U'_{n,z}(\mathbf{x}, \min y \leqq b_{n,z}(\mathbf{x})\,[T'_{n,z}(\mathbf{x},y)]).$$

Thus, as soon as we can show $U'_{n,z}$ and $T'_{n,z}$ are in $\mathscr{E}^2$ and the associated class $\mathscr{E}^2_*$ of relations([19]), respectively, we will have shown that f is in $\mathscr{E}^2$ and in general that $F' \subseteq \mathscr{E}^2$.

To give precise descriptions of $T'_{n,z}$ and $U'_{n,z}$ we proceed as follows. Let us assume that we have carried out an arithmetization of Turing machines within the class $\mathscr{E}^2$ of functions and the class $\mathscr{E}^2_*$ of relations. In particular, suppose we have defined functions Init_n, Yield, and Decode in $\mathscr{E}^2$ and a relation Fin in $\mathscr{E}^2_*$ with the following interpretations. $\text{Init}_n(x_1, \ldots, x_n)$ is the Gödel number of the instantaneous description of (any) Turing machine in its initial state which is scanning in standard position the input $(x_1, \ldots, x_n)$ on the tape; Yield(x, z) is the (Gödel) number of the instantaneous description which is the immediate result of instantaneous description number x on machine number z; Decode(x) is the number represented by the tape in the final instantaneous description whose (Gödel) number is x (if this tape is the binary encoding of a number, otherwise Decode(x) is 0); and Fin(x) is true if and only if x is the (Gödel) number of a final instantaneous description (one in which the machine is in its final, or passive, state).

The reader familiar with the work of Grzegorczyk [3] or Smullyan [8] with small bases for the recursively enumerable sets should be able to convince himself that such an arithmetization can be carried out even though the usual prime-power arithmetization takes us([20]) outside $\mathscr{E}^2$. The details of such an arithmetization are carried out in Appendix 3.

Given these functions Init_n, Yield, and Decode and the relation Fin, we can construct the function $U'_{n,z}$ and the relation $T'_{n,z}$ as follows. For each fixed value of z, we define the function Yield_z by setting $\text{Yield}_z(x) = \text{Yield}(x,z)$. Since $\mathscr{E}^2$ is closed under explicit transformations, this function is in $\mathscr{E}^2$. Now a limited recursion with respect to Yield_z results in a function $H_{n,z}$ whose value for the argument $(x_1, \ldots, x_n, y)$ is the instantaneous description of the yth step of the computation on machine number z beginning with the tape representing the n-tuple $(x_1, \ldots, x_n)$.

([19]) In [3], the class $\mathscr{E}^2_*$ is defined and studied. It may simply be taken to be the class of all relations whose characteristic functions lie in $\mathscr{E}^2$. (We employ the convention of denoting by X_* the class of relations whose characteristic functions lie in the class X, for any class X of functions.)

([20]) For example, consider Init_1. Let the Gödel number of the symbol a be denoted $gn(a)$, let $x_1 = a_k a_{k-1} \ldots a_1 a_0$ where each a_i is either 0 or 1 and let m be the minimum of $gn(0)$ and $gn(1)$. Then $\text{Init}_1(x_1) = 2^{gn(a_k)} \cdot 3^{gn(a_{k-1})} \cdot \ldots \cdot \text{Pr}(k)^{gn(a_1)} \cdot \text{Pr}(k+1)^{gn(a_0)} \geqq [(k+1)!]^m$. Thus $\text{Init}_1(x_1) \geqq [(l(x_1) + 1)!]^m$ and cannot lie in $\mathscr{E}^2$; its length as a function of x_1 is not bounded by a linear function so it is not even in F_1.

$$H_{n,z}(\mathbf{x},0) = \mathrm{Init}_n(\mathbf{x}),$$
$$H_{n,z}(\mathbf{x},s(y)) = \mathrm{Yield}_z(H_{n,z}(\mathbf{x},y)),$$
$$H_{n,z}(\mathbf{x},y) \leqq b_{n,z}(\mathbf{x}).$$

Here the bound on $H_{n,z}$ is simply the requirement that the amount of tape used is less than $K \cdot l(\mathbf{x})$. This implies that the largest number which can occur on the tape is less than $2^{K \cdot l(\mathbf{x})}$ which in turn is less than $b_{n,z}(\mathbf{x})$.

We now define the relation $T'_{n,z}$ by setting

$$T'_{n,z}(\mathbf{x},y) = \mathrm{Fin}[H_{n,z}(\mathbf{x},y)].$$

Since $H_{n,z}$ is in $\mathscr{E}^2$ we have $T'_{n,z} \in \mathscr{E}^2_*$. Finally, $U'_{n,z}(\mathbf{x},y)$ is to be that number, if any, which instantaneous description number $H'_{n,z}(\mathbf{x},y)$ represents. Thus we simply set

$$U'_{n,z}(\mathbf{x},y) = \mathrm{Decode}[H_{n,z}(\mathbf{x},y)]$$

and we have $U'_{n,z} \in \mathscr{E}^2_*$, as desired. We summarize this result in Theorem 5.

THEOREM 5. *A function f is in F' if and only if there is a number z such that for all $x_1, \ldots, x_n$,*

$$f(\mathbf{x}) = U'_{n,z}(\mathbf{x}, \min y \leqq b_{n,z}(\mathbf{x})[T'_{n,z}(\mathbf{x},y)]).$$

Proof. If there is such a z, it is the number of a Turing machine which computes a function in F'. But, by the definition of $U'_{n,z}$, this machine computes the function f. Conversely, if $f \in F'$ there is a machine, say with number z, which computes it and again by the definition of $U'_{n,z}$ and $T'_{n,z}$ the theorem holds.

COROLLARY. $\mathscr{E}^2 = F'$.

Proof. By Theorem 3 we have $\mathscr{E}^2 \subset F'$. Conversely, the above theorem shows $F' \subset \mathscr{E}^2$, since $U'_{n,z}$ is in $\mathscr{E}^2$ and $T'_{n,z}$ is in $\mathscr{E}^2_*$.

5. **The characterization of the classes F_i.** In the last section we stated that before characterizing the classes F_i we would obtain subclasses of F_i which satisfy the hypotheses of Lemma 5 and also contain such functions as $2^x y$. Now that we have studied the class F' we are in a position to do just that. Namely, we now observe that F' itself satisfies the hypotheses of the lemma; every function in F' depends "multiplicatively" on each argument. Then we fix one argument place, say the kth, and extend F' as far as possible by means of those subcases of explicit transformations, composition and limited recursion which preserve this "multiplicative" dependence on the kth argument. In this way, to any class X containing F' we associate a subclass X^k (for each $k \geqq 1$) which consists of functions depending "multiplicatively" on their kth arguments.

DEFINITION 4. Given a class X of functions containing the class F', we define the subclass X^k of X by requiring that a function $f \in X$ be in X^k if and only if one of the following holds:

(a) $f \in F'$;

(b) $f(x_1, \ldots, x_n) = g(\xi_1, \ldots, \xi_m)$ for $k \leqq n$, $g \in X$, and further $g \in X^l$ for every l such that $\xi_l = x_k$;

(c) $f(x_1, \ldots, x_l, y_1, \ldots, y_m) = h(x_1, \ldots, x_l, g(y_1, \ldots, y_m))$ where either (i) $k \leqq l$ and $h \in X^k$ or (ii) $k = l + m$, $h \in X^{l+1}$ and $g \in X^m$;

(d) $$\begin{aligned} f(x_1, \ldots, x_{k-1}, 0) &= g(x_1, \ldots, x_{k-1}), \\ f(x_1, \ldots, x_{k-1}, s(y)) &= h(x_1, \ldots, x_{k-1}, y, f(x_1, \ldots, x_{k-1}, y)), \\ f(x_1, \ldots, x_{k-1}, y) &\leqq j(x_1, \ldots, x_{k-1}, y) \end{aligned}$$

where $g \in X$, $h \in X^k \cap X^{k+1}$ and $j \in X^k$.

LEMMA 7. *If $f \in F_i^k$ then $a_f(x_1, \ldots, x_n) \leqq q(x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n) + Kl(x_k)$ for some integer $K \geqq 0$ and some $q \in F_{i-1}$.*

Proof. $f \in F_i^k$ means either $f \in F'$, or it is formed from functions in F_i^l by rules (b), (c) or (d). If $f \in F'$, the lemma clearly holds, since

$$a_f(\mathbf{x}, y, z) \leqq K[l(\max(\mathbf{x}, y, z))] \leqq K[l(\mathbf{x}, y, z)] = K[l(\mathbf{x}, y) + .1] + Kl(z).$$

Then the proof is completed by a straightforward induction in three cases on the number of applications of (b), (c), and (d) in the definition of f.

Lemmas 5 and 7 tell us that if the n-argument function h is in F_i^n and g is in F_i, then the function f is in F_i where $f(x_1, \ldots, x_{n-1}, y_1, \ldots, y_m)$ is equal to $h(x_1, \ldots, x_{n-1}, g(y_1, \ldots, y_m))$. Further, if the $(n+1)$-argument function h is in F^{n+1} and g and j are in F_i, then the function f is also in F_i where

$$\begin{aligned} f(x_1, \ldots, x_{n-1}, 0) &= g(x_1, \ldots, x_{n-1}), \\ f(x_1, \ldots, x_{n-1}, s(y)) &= h(x_1, \ldots, x_{n-1}, y, f(x_1, \ldots, x_{n-1}, y)) \end{aligned}$$

and

$$f(x_1, \ldots, x_{n-1}, y) \leqq j(x_1, \ldots, x_{n-1}, y).$$

To summarize these results we introduce the following two definitions and then state Theorem 6.

DEFINITION 5. The class X (of functions) will be said to be closed under *substitutions into functions in the class Y* if whenever $h \in Y$ and $g \in X$, then $f \in X$ where $f(\mathbf{x}, \mathbf{y}) = h(\mathbf{x}, g(\mathbf{y}))$.

DEFINITION 6. The class X will be said to be closed under *limited recursion with respect to functions in the class Y* if whenever $h \in Y$, $g, j \in X$, then $f \in X$ where

$$\begin{aligned} f(\mathbf{x}, 0) &= g(\mathbf{x}), \\ f(\mathbf{x}, s(y)) &= h(\mathbf{x}, y, f(\mathbf{x}, y)), \\ f(\mathbf{x}, y) &\leqq j(\mathbf{x}, y). \end{aligned}$$

THEOREM 6. *Each class F_i contains the class $\mathscr{E}^2$ and the function $f_1 \circ f$ for each $f \in F_{i-1}$. Further, it is closed under explicit transformations, substitutions*

into k-argument functions in F_i^k, and limited recursion with respect to $(k+1)$-argument functions in F_i^{k+1} for all $k \geqq 1$.

Proof. Containment of $f_1 \circ f$ follows from Lemma 2. Closure under explicit transformations was a result of Lemma 1. Lemmas 5 and 7 combine to prove the remaining assertions.

COROLLARY. *F_i is closed under substitutions into functions in $\mathscr{E}^2$ and limited recursion with respect to functions $\mathscr{E}^2$.*

The properties of the classes F_i expressed in Theorem 6 are sufficient to characterize them. We do so now, first defining classes $\bar{F}_1$, which we will prove are precisely the F_i.

DEFINITION 7. For each positive integer i the class $\bar{F}_i$ is to be the smallest class X of functions including $\mathscr{E}^2$ and containing the functions $f_1 \circ f$ for each $f \in \bar{F}_{i-1}$, if $i > 1$, or $f \in F$ for $i = 1$, which is closed under explicit transformations, substitutions into k-argument functions in X^k and limited recursion with respect to functions of $(k+1)$-arguments which lie in X^{k+1} for each $k \geqq 1$. ($\bar{F}_i$ is well defined: there are clearly classes X satisfying these conditions and, since $X^n \cap Y^n = (X \cap Y)^n$, the intersection of such classes again satisfies the conditions.)

Definition 7 is merely designed to reflect those properties of the classes F_i listed in Theorem 6, so that we have immediately $\bar{F}_i \subset F_i$. We summarize the immediate consequences of our earlier results on this definition in the following theorem.

THEOREM 7. *For each $i \geqq 1$, $\bar{F}_i \subseteq F_i$, $\bar{F}_i \subset \bar{F}_{i+1}$, $\bar{F}_i \supset \mathscr{E}^2$, and $\bar{F}_i \subset \mathscr{E}$.*

Proof. $\bar{F}_i \subseteq F_i$ by Theorem 6, $\bar{F}_i \subset \bar{F}_{i+1}$ by definition, and $\mathscr{E}^2 \subset \bar{F}_i$ again by definition. That $\bar{F}_i \subset \mathscr{E}$ is again trivial, since multiplication and 2^x are easily shown to be in $\mathscr{E}$. The remainder of this section is devoted to showing that $\bar{F}_i = F_i$. Thus every elementary function is in some $\bar{F}_i$. Let us note in passing that this result gives an alternate characterization of $\mathscr{E}$ in terms of the special types of substitutions and recursions occurring in Definition 7 above.

To complete the characterization, we wish to prove $F_i \subseteq \bar{F}_i$. To do so we follow the pattern used to prove $F' \subseteq \mathscr{E}^2$. That is, we define([19]) $T^i_{n,z} \in (\bar{F}_i)^*$ and $U^i_{n,z}$, $b^i_{n,z} \in \bar{F}_i$ and show that if $f \in F_i$ then

$$f(\mathbf{x}) = U^i_{n,z}(\mathbf{x}, \min y \leqq b^i_{n,z}(\mathbf{x})[T^i_{n,z}(\mathbf{x},y)]).$$

We will show in fact that

$$\min y \leqq b^i_{n,z}(\mathbf{x})[T^i_{n,z}(\mathbf{x},y)]$$

defines a function in $\bar{F}_i$ and also that $U^i_{n,z} \in \bar{F}_i^{n+1}$. Thus $f(\mathbf{x})$ will be shown to be the result of substituting a function in $\bar{F}_i$ into an $(n+1)$-argument function in $\bar{F}_i^{n+1}$ and hence (by Definition 7) $f(\mathbf{x})$ must be in $\bar{F}_i$. We first prove the theorem needed to establish these inclusions([21]).

([21]) We follow the method used in [3] to show that limited minimalization is permissible in a wide variety of classes.

LEMMA 8. *Whenever* $f \in \bar{F}_i^{n+1}$, *the function* F *whose value on* $(x_1, \ldots, x_n, y)$ *is equal to the number of* z*'s less than or equal to* y *for which* $f(x_1, \ldots, x_n, z) = 0$ *is also in* $\bar{F}_i^{n+1}$.

Proof. We begin by defining the functions p, $\dot{-}$, and τ in $\mathscr{E}^2$.

$$\left.\begin{aligned} p(0) &= 0 \\ p(s(x)) &= x \\ p(x) &\leqq x \end{aligned}\right\} p(x) \text{ is the predecessor of } x.$$

$$\left.\begin{aligned} x \dot{-} 0 &= x \\ x \dot{-} s(y) &= p(x \dot{-} y) \\ x \dot{-} y &\leqq x \end{aligned}\right\} x \dot{-} y \text{ is 0 if } x \leqq y, \text{ and } x - y \text{ otherwise.}$$

$$\left.\begin{aligned} \tau(x,0) &= s(x) \\ \tau(x,s(y)) &= x \\ \tau(x,y) &\leqq s(x) \end{aligned}\right\} \tau(x,y) \text{ is } s(x) \text{ if } y = 0 \text{ and is } x \text{ otherwise.}$$

Since $f \in \bar{F}_i^{n+1}$, so is the function t, where

$$t(x_1, \ldots, x_n, y, z) = \tau(z, f(x_1, \ldots, x_n, s(y))).$$

Now we define F by a limited recursion with respect to t, and thus by Definition 7 $F \in \bar{F}_i^{n+1}$.

$$\begin{aligned} F(x_1, \ldots, x_n, 0) &= 1 \dot{-} f(x_1, \ldots, x_n, 0), \\ F(x_1, \ldots, x_n, s(y)) &= t(x_1, \ldots, x_n, y, F(x_1, \ldots, x_n, y)), \\ F(x_1, \ldots, x_n, y) &\leqq s(y). \end{aligned}$$

That F is the required function follows immediately upon noting that $1 \dot{-} z$ is 1 if $z = 0$ and 0 otherwise, and that τ adds 1 if and only if its second argument is 0.

THEOREM 8. *If* $f \in \bar{F}_i^{n+1}$ *so is the function* g *defined by* $g(x_1, \ldots, x_n, y) = \min z \leqq y[f(x_1, \ldots, x_n, z) = 0]$.

Proof. By Lemma 8, $F \in \bar{F}_i^{n+1}$. Applying Lemma 8 to F itself, we obtain the function $G \in \bar{F}_i^{n+1}$ where $G(\mathbf{x}, y)$ is equal to the number of z's less than $y + 1$ for which $F(\mathbf{x}, z) = 0$. But $F(\mathbf{x}, z) = 0$ means that for no $w \leqq z$ is $f(\mathbf{x}, w) = 0$. Thus $G(\mathbf{x}, y)$ is precisely the number of z's less than $y + 1$ such that, for every $w \leqq z$, $f(\mathbf{x}, w) = 0$. That is, $G(\mathbf{x}, y)$ is the least $z \leqq y$ such that $f(\mathbf{x}, z) = 0$ if such a z exists, but is $y + 1$ if no such z exists. To modify G so that $G(\mathbf{x}, y) = 0$ if no such z exists, we first define a function $\sigma \in \mathscr{E}^2$.

$$\left.\begin{aligned} \sigma(x,0) &= x \\ \sigma(x,s(y)) &= 0 \\ \sigma(x,y) &\leqq x \end{aligned}\right\} \sigma(x,y) \text{ is } x \text{ if } y = 0, \text{ 0 otherwise.}$$

Now we note that $G(\mathbf{x},y) \dot{-} y$ is 0 if and only if $G(\mathbf{x},y) \leqq y$. Hence $\sigma(G(\mathbf{x},y), G(\mathbf{x},y) \dot{-} y)$ is precisely the desired function g. Thus g lies in $\bar{F}_i^{n+1}$ by definition.

COROLLARY. *If* $R \in (\bar{F}_i^{n+1})_*$ *and* $g \in \bar{F}_i$, *then* $h \in \bar{F}_i$ *where*

$$h(x_1, \ldots, x_n, y_1, \ldots, y_m) = \min z \leqq g(y_1, \ldots, y_m)\,[R(x_1, \ldots, x_n, z)].$$

Proof. By the definition of $(\bar{F}_i^{n+1})_*$, R has a characteristic function $f \in \bar{F}_i^{+1}$. Thus, the function j is in $\bar{F}_i^{n+1}$ where $j(x_1, \ldots, x_n, y) = \min z \leqq y[R(x_1, \ldots, x_n, z)]$. But h is the result of substituting g into the $(n+1)$st argument of j, so by Definition 7, $h \in \bar{F}_i$.

This corollary will allow us to conclude that the n-argument function f is in $\bar{F}_i$ as soon as we have shown that

$$f(\mathbf{x}) = U_{n,z}^i(\mathbf{x}, \min y \leqq b_{n,z}^i(\mathbf{x})\,[T_{n,z}^i(\mathbf{x},y)])$$

where $U_{n,z}^i$, $b_{n,z}^i$, and $T_{n,z}^i$ lie in $\bar{F}_i^{n+1}$, $\bar{F}_i$, and $(\bar{F}_i^{n+1})_*$, respectively.

We now define $b_{n,z}^i$, $U_{n,z}^i$ and $T_{n,z}^i$ by an exact parallel of the definitions of $b_{n,z}$, $U_{n,z}$ and $T_{n,z}$ given at the end of the previous section, and show that they do lie in the desired classes. Assume we are considering a computation of a function in F_i by Turing machine number z with k (internal) states and three symbols appearing on its tape([17]). Given the argument $(x_1, \ldots, x_n)$, the amount of tape used is less than $f_{i-1}(K \max(x_1, \ldots, x_n, 1))$ by Lemma 3. Thus the number of instantaneous descriptions which occur is less than

$$3^{f_{i-1}(K \max(x),1))} \cdot k \cdot l(\mathbf{x}).$$

This is in turn less than

$$[f_i(K(x_1 + \ldots + x_n + 1))]^2 \cdot k \cdot ((x_1 + 1) + \ldots + (x_n + 1))$$

which we call $b_{n,z}^i(\mathbf{x})$. Since multiplication is a function in $\mathscr{E}^2$ and each $\bar{F}_i$ is closed under substitutions into functions in $\mathscr{E}^2$, each class $\bar{F}_i$ must be closed under multiplication. Thus, since $f_1(K(x_1 + \ldots + x_n + 1)) = [f_1(x_1) \ldots f_1(x_n) \cdot 2]^K$, $f_1(K(x_1 + \ldots + x_n + 1))$ is in $\bar{F}_1$. Hence $f_i(K(x_1 + \ldots + x_n + 1))$ is in $\bar{F}_i$ by the clause in Definition 7 assuring containment of $f_1 \circ f$ in $\bar{F}_i$ whenever f was in $\bar{F}_{i-1}$. Using the closure of $\bar{F}_i$ under multiplication we have the containment of $b_{n,z}^i$ in $\bar{F}_i$ as desired.

Now we may define the function $H_{n,z}^i$ where $H_{n,z}^i(\mathbf{x},y)$ is the number of the instantaneous description at the yth step of machine number z given input $(x_1, \ldots, x_n)$. This definition is precisely the result of replacing the bound $b_{n,z}$ in $H_{n,z}$ by $b_{n,z}^i$.

$$H_{n,z}^i(\mathbf{x},0) = \mathrm{Init}_n(\mathbf{x}),$$

$$H_{n,z}^i(\mathbf{x},s(y)) = \mathrm{Yield}_z(H_{n,z}^i(\mathbf{x},y)),$$

$$H_{n,z}^i(\mathbf{x},y) \leqq b_{n,z}^i(\mathbf{x}).$$

Since $\text{Yield}_z \in \mathscr{E}^2$ and $b^i_{n,z}$ is trivially in $\bar{F}^{n+1}_i$, this recursion defines $H^i_{n,z}$ as a function in $\bar{F}^{n+1}_i$ (by Definition 7). Now we define $U^i_{n,z}$ and $T^i_{n,z}$ where $U^i_{n,z}(\mathbf{x},y)$ is the number on the tape in the yth instantaneous description of this computation and $T^i_{n,z}(\mathbf{x},y)$ holds precisely when y is the number of instantaneous descriptions in the computation.

$$U^i_{n,z}(\mathbf{x},y) = \text{Decode}(H^i_{n,z}(\mathbf{x},y)),$$

$$T^i_{n,z}(\mathbf{x},y) = \text{Fin}(H^i_{n,z}(\mathbf{x},y).$$

Since the function Decode is in $\mathscr{E}^2$ and the relation Fin is in $\mathscr{E}^2_*$ we have $U^i_{n,z} \in \bar{F}^{n+1}_i$ and $T^i_{n,z} \in (\bar{F}^{n+1}_i)_*$. Thus

$$\min y \leqq b^{i\,\prime\prime}_{n,z}(\mathbf{x})[T^{i\,\prime\prime}_{n,z}(\mathbf{x},y)]$$

defines a function in $\bar{F}_i$ by the corollary of Theorem 8. This establishes Theorem 9 below as desired.

THEOREM 9. *A function f is in F_i if and only if there is a number z such that*

$$f(\mathbf{x}) = U^i_{n,z}(\mathbf{x}, \min y \leqq b^i_{n,z}(\mathbf{x})[T^i_{n,z}(\mathbf{x},y)]).$$

COROLLARY. *For all positive integers*([22]) *i, $F_i = \bar{F}_i$.*

6. **The classes $(F_i)_*$ of relations.** We know that F_i is properly contained in F_{i+1} and that $\bigcup_{i=1}^{\infty} F_i = \mathscr{E}$. Thus we also know that the corresponding classes of relations satisfy $(F_i)_* \subseteq (F_{i+1})_*$ and $\bigcup_{i=1}^{\infty}(F_i)_* = \mathscr{E}_*$. We now show that in fact the containment of $(F_i)_*$ in $(F_{i+1})_*$ is proper. Thus these classes subdivide the class of elementary sets (and relations) into classes according to the difficulty of the decision procedure for the classes.

For instance, it is often said that Tarski's decision procedure for the first-order theory of real numbers [9] is inherently more complicated than the truth table procedure for propositional calculus. Once we have shown that the classes $(F_i)_*$ form a subdivision of the elementary sets, we may make this statement precise. We need only observe that the truth table method is in class F_1 and that Tarski's procedure is (according to the author's estimates) in F_8 ([23]). This seems

([22]) This corollary gives a proof that $\mathscr{E} = \bigcup_{i=1}^{\infty} F_i$ without any reference to our considerations in §3. We showed in Theorem 2 that $\mathscr{E} \subseteq \bigcup_{i=1}^{\infty} F_i$ and in Theorem 6 that $\bigcup_{i=1}^{\infty} \bar{F}_i \subseteq \mathscr{E}$; thus we obtain the desired equality.

([23]) This, of course, assumes an appropriate translation of formulas into numbers, since the classes F_1 consist of numerical functions. A useful translation is the following: Assuming the language has basic symbols $a_1, \ldots, a_n$, interpret the symbol a_i as the integer i and interpret any formula as the positive integer whose n-adic encoding it is. This establishes a one-one correspondence between finite strings on the alphabet $a_1, \ldots, a_n$ and the positive integers, so that these sets may be identified.

ample justification for the current hesitation about programming the latter decision method and hoping for immediate results([24]).

Before showing that there is, in fact, a relation in $(F_{i+1})_*$ which is not in $(F_i)_*$, let us note that although the function f_{i+1} is not in F_i the relation $\lambda(x,y)[f_{i+1}(x)=y]$ is in $(F_i)_*$. In fact, it is in $\mathscr{E}^2$, as the reader may easily verify. In this sense the complexity of f_i lies almost entirely in its rapid growth. Given enough tape to express the answer, the computation lies in $\mathscr{E}^2$. However, a characteristic function in F_{i+1} but not in F_i is in a stronger sense more complex than one in F_i. The added tape is actually used for computational purposes and is not merely needed to express an enormous answer.

To show that $(F_i)_* \neq (F_{i+1})_*$ we construct in F_{i+1} a characteristic function Φ_i with the following property. The value of $\Phi_i(z,x)$ is 0 if the value of $f(x)$ is 0 where f is the function computed by Turing machine number z and the value of $\Phi_i(z,x)$ is 1 otherwise. A standard diagonalization argument will then show that $\Phi_i \notin F_i$ so the relation whose characteristic function is Φ_i lies in $(F_{i+1})_*$ but not in $(F_i)_*$.

To define Φ_i we use the arithmetization in Appendix 3 to obtain a new T-predicate and function U which are functions of z. We begin by defining a bounding function b_r^i in F_{n+1}^i for the number of steps in a computation of an n-argument function in F_i. We would like to do this by merely setting

$$b_n^i(z,\mathbf{x}) = b_{n,z}^i(\mathbf{x}) = [f_i(K\cdot(x_1 + \ldots + x_n + 1))]^2 \cdot k \cdot ((x_1 + 1) + \ldots + (x_n + 1))$$

where K and k are numbers depending only upon z. Since any given Turing machine is assigned infinitely many Gödel numbers([25]), we can in fact choose a Gödel number z such that K and k are both less than z. Hence we may replace each of them by z in this bound. Thus we may use the definition

$$b_n^i(z,\mathbf{x}) = [f_i(z\cdot(x_1 + \ldots + x_n + 1))]^2 \cdot z \cdot ((x_1 + 1) + \ldots + (x_n + 1))$$

for the bounding function on computations. Since F_1 is closed under multiplication, $z\cdot(x_1 + \ldots + x_n + 1)$ is in F_1. Thus $f_i(z\cdot(x_1 + \ldots + x_n + 1))$ is in F_{i+1} by Lemma 2, and so is b_n^i.

Now we may simply define H_n^i, T_n^i, U_n^i and a function ψ_n^i as follows.

$$\begin{aligned} H_n^i(z,\mathbf{x},0) &= \mathrm{Init}_n(\mathbf{x}), \\ H_n^i(z,\mathbf{x},s(y)) &= \mathrm{Yield}(H_n^i(z,\mathbf{x},y),z), \\ H_n^i(z,\mathbf{x},y) &\leqq b_n^i(z,\mathbf{x}) \\ U_n^i(z,\mathbf{x},y) &= \mathrm{Decode}(H_n^i(z,\mathbf{x},y)), \end{aligned}$$

([24]) This, of course, does not say that *no* decision procedure for this is in F_1, but only that the one offered in [9] is not. That is, it shows that the set of valid formulas is a set in F_{8*}, but does not exclude the possibility that it is F_{i*} for some i less than 8.

([25]) Our arithmetization allows the list of states of a Turing machine to contain repetitions.

$$T_n^i(z,\mathbf{x},y) \equiv \mathrm{Fin}(H_n^i(z,\mathbf{x},y)),$$

$$\Psi_n^i(z,\mathbf{x}) = U_n^i(z,\mathbf{x},\min y \leqq b_n^i(z,\mathbf{x})[T_n^i(z,\mathbf{x},y)]).$$

If f is in F_i, by Lemma 3 there is a K for which $a_f(\mathbf{x}) < f_{i-1}(K \cdot \max(\mathbf{x},1))$. Thus, by the above, f is computable by a Turing machine whose Gödel number z is greater than K. The number of steps in the computation of $f(\mathbf{x})$ is less than $b_{n,z}^i(\mathbf{x})$, so for any function f in F_i, there is a number z such that

$$f(\mathbf{x}) = \Psi_n^i(z,\mathbf{x}).$$

Now we define the characteristic function Φ_i (of two arguments) by setting

$$\Phi_i(z,x) = \sigma(1,1 \dot{-} \Psi_1^i(z,x))$$

so we have $\Phi_i(z,x) = \min(1,f(x))$ if machine number z computes the one-argument function f. Further, Φ_i is in F_{i+1} since Ψ_n^i was constructed within F_{i+1}.

To show that $\Phi_i \notin F_i$ we assume that in fact Φ_i does lie in F_i. Then the function Θ is also in F_i where $\Theta(x) = 1 \dot{-} \Phi_i(x,x)$ and since Θ is a one-argument function whose only values are 0 and 1 we have a z_0 for which $\Theta(x) = \Psi_1^i(z_0,x) = \Phi_i(z_0,x)$.

In particular, $\Theta(z_0) = \Phi_i(z_0,z_0)$, but by definition $\Theta(z_0) = 1 \dot{-} \Phi_i(z_0,z_0) \neq \Phi_i(z_0,z_0)$ which is a contradiction. This proves Theorem 10 below, since the relation whose characteristic function is Φ_i lies in $(F_{i+1})_*$ but not in $(F_i)_*$.

THEOREM 10. *For each positive integer i, $(F_i)_*$ is properly contained in $(F_{i+1})_*$.*

APPENDIX 1

Elementary definitions of MR, Corn, and Fin

In [1], a convention is introduced by which the blank, β, is written for S_0 and 1 is written for S_1. Let us extend this convention by writing 0 for S_2 throughout. Since the symbol S_i is assigned Gödel number 2^{4i+7} in [1], this means that β, 1 and 0 receive Gödel numbers 2^7, 2^{11} and 2^{15} respectively. Following the system of Gödel numbering used in [1], we then have unique numbers assigned to every tape consisting of β, 1 and 0. For example, the tape $10\beta1$ has Gödel number $2^{11}3^{15}5^77^{11}$. Using the concatenation function $*$ introduced in [1], we may write this Gödel number as $2^{11}*2^{15}*2^7*2^{11}$. (We will make free use of these conventions in this appendix.)

In order to define MR, let us first introduce the elementary sets (1-place relations) Zeros and Ones and the elementary function S. The first two are self-explanatory; the last, when applied to the Gödel number of a tape representing n, will yield the Gödel number of the tape representing $n+1$.

$$\mathrm{Zeros}(x) \leftrightarrow (\forall y)_{\leqq x}[\mathscr{L}(y) = 1 \wedge (\exists z,w)_{\leqq x}[z*y*w=x] \rightarrow y=2^{15}],$$

$$\mathrm{Ones}(x) \leftrightarrow (\forall y)_{\leqq x}[\mathscr{L}(y) = 1 \wedge (\exists z,w)_{\leqq x}[z*y*w=x] \rightarrow y=2^{11}].$$

$$S(x) = \min y \leqq (x * x * x)\{(\exists z)_{\leqq x}[z * 2^{15} = x \wedge z * 2^{11} = y]$$
$$\vee (\exists z,w,v)_{\leqq y}[z * 2^{15} * w = x \wedge \text{Ones}(w) \wedge \text{Zeros}(v)$$
$$\wedge \mathscr{L}(w) = \mathscr{L}(v) \wedge z * 2^{11} * v = y]$$
$$\vee (\exists v)_{\leqq y}[\text{Ones}(x) \wedge \text{Zeros}(v) \wedge \mathscr{L}(x) = \mathscr{L}(v) \wedge 2^{11} * v = y]\}.$$

Here $\mathscr{L}(x)$ is the number of symbols occurring in the sequence whose Gödel number is x (as defined on p. 58 of [1]).

Now we define MR by limited recursion as follows:

$$\begin{aligned} \text{MR}(0) &= 2^{11}, \\ \text{MR}(n+1) &= S(\text{MR}(n)), \\ \text{MR}(n) &\leqq (\text{Pr}(n+1)^{15})^{n+1}. \end{aligned}$$

Then Corn is simply defined as MR^{-1}, that is

$$\text{Corn}(x) = \min y \leqq x[\text{MR}(y) = x].$$

The one remaining change to be made is in the definition of Fin, and the required definition is:

$$\text{Fin}(x,z) \leftrightarrow \text{ID}(x) \wedge \text{TM}(z) \wedge (\exists F,G,r,s)_{\leqq x}\{(x = F * 2^r * 2^s * G) \wedge \text{IC}(r)$$
$$\wedge Al(s) \wedge (\exists y)_{\leqq x}[\text{MR}(y) = F * 2^s * G]$$
$$\wedge (\forall n)_{\leqq \mathscr{L}(z)}[(1Gl(nGlz) \neq r) \vee (2Gl(nGlz) \neq s)]\}.$$

APPENDIX 2

Turing machines and finite automata

In this appendix, we present formal definitions of Turing machine and of computation by such a machine. We then define "computation of a Turing machine when viewed as a finite automaton," and we define the class F of numerical functions computable in this way. We conclude the appendix by establishing the properties of F which are needed in the body of this paper.

DEFINITION 1. An *alphabet* is a finite set whose elements are called symbols which always includes the particular symbol β (called the blank symbol) as an element.

DEFINITION 2. A *Turing machine* Z over the alphabet A is a quadruple (S,m,s_0,f) where S is a finite set, s_0 and f are elements of S, and m: $A \times (S - \{f\}) \rightarrow A \times S \times \{1,-1,0\}$. We call S the set of states of Z, s_0 the initial state, f the final state, and m the transition function.

DEFINITION 3. Given a Turing machine $Z = (S,m,s_0,f)$ over the alphabet A we define an *instantaneous description* of Z to be a triple (t,s,p) where t is a finite sequence of elements of A, p is a positive integer no greater than the length

of t, and s is an element of S. We call t the tape in Z, p the number of the scanned square, and s the state of Z.

The tape in a machine Z is extendable to the right, but has a fixed leftmost square, denoted by the positive integer 1. The extendability on the right edge is the cause for the final three clauses in the definition of the yield operation below. These clauses allow for the addition of an extra square with a blank on it whenever the machine is about to run off the right edge of the tape and also provide for dropping a rightmost blank from consideration whenever it is no longer being scanned.

DEFINITION 4. Given a Turing machine $Z = (S,m,s_0,f)$ over the alphabet A we define the *yield operation*, $\rightarrow$, between instantaneous descriptions of Z as follows. $X \rightarrow Y(Z)$ is to mean that one of the following conditions obtains, where X and Y are instantaneous descriptions of Z and a_i and b_i are in A for all positive integers i.

(1) $X = (a_1 \dots a_n, s, p)$ and $Y = (b_1 \dots b_n, s', p')$ with $a_j = b_j$ for all $j \neq p$, $m(a_p, s) = (b_p, s', p'-p)$ and either $p < n$ or $p = p' = n$.

(2) $X = (a_1 \dots a_{n-1}a_n^p, s, n)$ and $Y = (a_1 \dots a_{n-1}b_n\beta, s', n+1)$ where $m(a_n, s) = (b_n, s', 1)$.

(3) $X = (a_1 \dots a_{n-1}a_n, s, n)$ and $Y = (a_1 \dots a_{n-1}b_n, s', n-1)$ where $m(a_n, s) = (b_n, s', -1)$ and $b_n \neq \beta$.

(4) $X = (a_1 \dots a_{n-1}a_n, s, n)$ and $Y = (a_1 \dots a_{n-1}, s', n-1)$ where $m(a_n, s) = (\beta, s', -1)$.

DEFINITION 5. A *computation* by a Turing machine Z over the alphabet A is to be a finite sequence $X_1, \dots, X_q$ of instantaneous descriptions of Z such that, for all $i = 1, \dots, q-1$, $X_i \rightarrow X_{i+1}(Z)$ and $X_q = (t,f,p)$ for some finite sequence t of elements of A and some integer p. We say that X_1 *begins* this computation and that X_q is the *resultant* of X_1.

DEFINITION 6. Given a subset D of the set T_A of all finite sequences of elements of an alphabet A, the function $\phi: D \rightarrow T_A$ is said to be *computed* by the Turing machine Z over A if the following conditions hold. For each $t \in D$ there is a computation by Z beginning with $(t,s_0,1)$ and the resultant of $(t,s_0,1)$ is $(\phi(t),f,p)$ for some p.

Since we have agreed to encode our numerical functions in binary notation, we will establish the convention that the particular alphabet $\{0,1,\beta\}$ is the one fixed set taken as the alphabet over which all Turing machines computing numerical functions operate. We will often omit the phrase "over the alphabet $\{0,1,\beta\}$" when referring to Turing machines computing numerical functions. With this convention in mind, let us define a computation of a numerical function as follows.

DEFINITION 7. The Turing machine Z over the alphabet $\{0,1,\beta\}$ will be said to *compute* the function f from n-tuples of non-negative integers to non-negative integers if it computes the function $f' : D \rightarrow T^{\{0,1\}}$ where

(i) D is the set of all strings $\bar{a}_1\beta\bar{a}_2\beta \dots \beta\bar{a}_n$ for $(a_1, a_2, \dots, a_n)$ an n-tuple in the domain of f and

(ii) $f'(\bar{a}_1\beta \dots \beta\bar{a}_n)$ is defined to be $\overline{f(a_1, \dots, a_n)}$

where $\bar{x}$ denotes the binary encoding of the non-negative integer x.

This completes our formal definitions of Turing machines and of computations by such machines. In Appendix 3 we arithmetize these machines and establish some results needed in §§4, 5 and 6 of this paper.

Now we turn our attention to finite automata. In the present context these are most easily viewed as Turing machines which compute in a different (and very restrictive) way. A computation by such a machine always begins on the rightmost square of the input tape and proceeds by moving one square to the left at each step of the computation. In these computations the machine is allowed to extend the tape *on the left* as much as necessary. Because of this it is convenient to think of the squares as being numbered from right to left (reversing the convention used in Definition 4 above). We make these remarks precise in the following definitions.

DEFINITION 8. Given a Turing machine $Z = (S, m, s_0, f)$ over the alphabet A, we define the *yield$_F$ operation*, $\rightarrow_F$, between instantaneous descriptions of Z as follows. $X \rightarrow_F Y(Z)$ is to mean that one of the following conditions obtains, where X and Y are instantaneous descriptions of Z and a_i and b_i are elements of A.

(1) $X = (a_n a_{n-1} \dots a_1 a_0, s, p)$ and $Y = (b_n b_{n-1} \dots b_1 b_0, s', p+1)$ with $p < n$, $b_j = a_j$ for all $j \neq p$, and also $m(a_p, s) = (b_p, s', -1)$.

(2) $X = (a_n a_{n-1} \dots a_1 a_0, s, n)$ and $Y = (b_n a_{n-1} \dots a_1 a_0, f, n+1)$ with $m(a_n, s) = (b_n, f, -1)$.

(3) $X = (a_n a_{n-1} \dots a_1 a_0, s, n)$ and $Y = (\beta b_n a_{n-1} \dots a_1 a_0, s', n+1)$ with $m(a_n, s) = (b_n, s', -1)$ and $s' \neq f$.

DEFINITION 9. An *F-computation* by a Turing machine Z over the alphabet A is a finite sequence $X_1, \dots, X_q$ of instantaneous descriptions of Z such that

(i) $X_i \rightarrow_F Y(Z)$ for all $i = 1, \dots, q-1$, and

(ii) $X_1 = (t, s_0, 0)$ and $X_q = (t', f, l(t'))$ where t and t' are finite sequences of elements of A and $l(t')$ denotes the length of the sequence t'.

We say that X_1 *begins* the computation and that X_q is the *resultant* of X_1.

DEFINITION 10. Given a subset D of the set T_A of all finite sequences of elements of the alphabet A, the function $\phi: D \rightarrow T_A$ is said to be *computed by the Turing machine Z viewed as a finite automaton* over A if the following conditions hold. For each $t \in D$ there is an F-computation by Z beginning with $(t, s_0, 0)$ and the resultant of $(t, s_0, 0)$ is $(\phi(t), f, l(\phi(t)))$.

When a function ϕ is computed by a Turing machine Z viewed as a finite automaton we speak of Z as a finite automaton and of ϕ as being computed by this finite automaton. Thus, whenever we refer to a finite automaton we mean simply a Turing machine which is viewed as carrying out only F-computations.

Whenever we refer to a computation by a finite automaton Z we mean simply an F-computation by the Turing machine Z. We shall use this abbreviated terminology freely throughout this paper.

THEOREM 1. *If ϕ is computed by a finite automaton Z with k nonfinal states then, for every argument t in the domain of ϕ, the length of $\phi(t)$ is at most $k + l(t)$ where $l(t)$ is the length of the tape t.*

Proof. By Definition 1 above, after Z has read all the symbols of t it continues moving left reading a blank square at each step (unless it enters the final state and stops). However, if it enters the same state twice while reading these blanks it must be in a cycle and will continue for ever. Since k is the number of distinct nonfinal states of Z and t is in the domain of ϕ, Z must enter f within k steps after reading the last symbol of t. Thus the length of $\phi(t)$ is at most k plus the length of t.

Because of this and other severe restrictions on functions computable by finite automata, the accepted methods for encoding numerical inputs differ from the encoding indicated in our Definition 7 for Turing machines. (See, for example [2, p.31].)

To input n numerical arguments to a Turing machine, each number is written in binary and these n binary strings are written in order on the tape, separated by blanks. One may think of this as feeding the numbers into the machine in series, one after the other. In contrast to this, numbers are fed into finite automata in parallel; the least significant digit of each of the n numbers is read on the first step of the computation, the next significant digit of each number at the second step, and so on. To simplify the notation, we will restrict our discussion here to functions of one or two arguments. (The only additional difficulties encountered in treating the general case are notational.)

A finite automaton which is to compute a numerical function of one argument will always be assumed to operate over the alphabet $A_1 = \{\beta,0,1\}$. Let us call a string on this alphabet *an encoding* of the nonnegative integer n if it is of the form $\beta \ldots \beta\bar{n}$ where $\bar{n}$ is the (unique) binary encoding of n. Thus, for example, $\beta\beta101$ and 101 are both encodings of the number five, but neither 0101 nor 101β is. With these conventions established, we now define computation of numerical functions of a single variable.

DEFINITION 11. The finite automaton Z over the alphabet A_1 will be said to *compute the function f from non-negative integers to non-negative integers* if it computes some function $f': D \to T_{\{0,1\}}$ such that

(i) D is the set of all binary encodings $\bar{n}$ of non-negative integers n, and

(ii) $f'(\bar{n})$ is an encoding of $f(n)$.

Before defining a computation of a numerical function of two arguments we establish two further conventions. First, a finite automaton computing such a function will always be assumed to operate over the particular alphabet

$$A_2 = \begin{Bmatrix} \beta & \beta & \beta & 0 & 1 & 0 & 0 & 1 & 1 \\ \beta' & 0' & 1' & \beta' & \beta' & 0' & 1' & 0' & 1' \end{Bmatrix}.$$

We will sometimes wish to write

$$\beta \text{ for the blank symbol } \begin{matrix} \beta \\ \beta \end{matrix},$$

$$0 \text{ for the symbol } \begin{matrix} \beta \\ 0 \end{matrix},$$

and

$$1 \text{ for } \begin{matrix} \beta \\ 1 \end{matrix}.$$

Thus we will be able to speak as if A_1 were a subset of A_2. Second, for each pair (m,n) of non-negative integers, we define the strings m_n and n_m on the alphabet A_1 as follows. Assume (without loss of generality) that the binary encoding of n is k digits longer than that of $m, (k \geqq 0)$. Then n_m is simply $\bar{n}$, the binary encoding of n, and m_n is $\beta \ldots \beta \bar{m}$ with k β's. Thus m_n and n_m are encodings of m and n respectively, they have the same length, and one is an encoding in binary. Now given the pair (m,n), let $m_n = a_1 a_2 \ldots a_p$ and $n_m = b_1 b_2 \ldots b_p$ with a_i and b_i in A_1. We denote by $\overline{(m,n)}$ the tape

$$\begin{Bmatrix} a_1 a_2 \ldots a_p \\ b_1 b_2 \ldots b_p \end{Bmatrix}$$

no the alphabet A_2.

DEFINITION 12. The finite automaton Z over the alphabet A_2 will be said to *compute the function f from pairs of non-negative integers to non-negative integers* if it computes some function $f' : D \to T_{A_1}$ such that

(i) D is the set of all tapes $\overline{(m,n)}$ for all pairs (m,n) of non-negative integers, and

(ii) $f'(\overline{m,n})$ is an encoding of $f(m,n)$. (In this definition we have identified A_1 with a subset of A_2.)

Similar definitions can clearly be made for computations by finite automata of numerical functions of any number of arguments. We assume these definitions have been made and denote by F the class of all numerical functions of any number of arguments which are computable by finite automata.

As examples of these definitions, consider the successor function and addition. The successor function is computed by a finite automaton Z which acts as follows. Upon input 0, Z outputs $\beta 1$, upon input 1 Z outputs 10, upon input 10 Z outputs $\beta 11$, and so on. (The blanks in the outputs arise since the computation cannot be terminated until a blank to the left of the input is reached.) The construction of such a machine is left to the reader. Addition is computed by an automaton whose input and output pairs include, for example,

$$\begin{Bmatrix} 1 \\ 0 \end{Bmatrix}, \beta 1\Big\}, \quad \begin{Bmatrix} 1 \\ 1 \end{Bmatrix}, 10\Big\}, \quad \begin{Bmatrix} \beta 1 \\ 10 \end{Bmatrix}, \beta 11\Big\}, \quad \begin{Bmatrix} \beta 1 \\ 11 \end{Bmatrix}, 100\Big\} \text{ and } \begin{Bmatrix} \beta\beta 101 \\ 10011 \end{Bmatrix}, \beta 11000\Big\}.$$

One can also easily construct a finite automaton to compute each constant function C_q^n of n arguments and each identity function U_i^n of n arguments, as defined, for example, in [4, p. 220].

We now wish to establish some properties of the class F of numerical functions computable by finite automata. The desired properties are closure under composition and explicit transformations, and linear bounds in an appropriate sense. We begin by indicating a direct proof of the closure of F under composition. We will construct a finite automaton to compute $g \circ h$, where $(g \circ h)(y) = g(h(y))$, given automata computing g and h. We will omit the details of the proof that this finite automaton actually works, as well as the generalization to composition of functions of arbitrarily many arguments.

Assume the numerical functions g and h of one argument are computed by the finite automata $Z_g = (S_g, m_g, s_{g0}, f_g)$ and $Z_h = (S_h, m_h, s_{h0}, f_h)$ respectively. To shorten the definition of the automaton for computing $g \circ h$, we introduce the following abbreviations. For $a \in A$, $s_h \in S_h$ and $s_g \in S_g$,

(i) let $a_h = a$ and $s_h' = f_h$ if $s_h = f_h$, otherwise let $a_h = x$ and $s_h' = y$ where $m_h(a, s_h) = (x, y, -1)$; and

(ii) let $a' = a_h$ and $s_g' = f_g$ if $s_g = f_g$, otherwise let $a' = u$ and $s_g' = v$ where $m_g(a_h, s_g) = (u, v, -1)$.

Now define the function

$$m' : A_1 \times [(S_g \times S_h) - \{(f_g, f_h)\}] \to A_1 \times (S_g \times S_h) \times \{-1\}$$

by setting $m'(a, (s_g, s_h)) = (a', (s_g', s_h'), -1)$, and set

$$Z_{g \circ h} = (S_g \times S_h,\ m',\ (s_{g0}, s_{h0}),\ (f_g, f_h)).$$

To see that $Z_{g \circ h}$ computes $g \circ h$, assume that $Z_{g'' h}$ is in state (s_g, s_h) at the nth step in its computation beginning with tape t, and that Z_h applied to tape t yields the tape t'. Then s_h is the state Z_h reaches at the nth step of its computation beginning with tape t, and s_g is the state Z_g reaches at the nth step of its computation begun on the tape t'. The inductive proof that $Z_{g \circ h}$ computes $g \circ h$ based on this observation is straightforward and left to the reader.

The closure of F under explicit transformations is a corollary of the preceding results. Let $f(x_1, \ldots, x_n) = g(\xi_1, \ldots, \xi_k)$. If $\xi_i = x_j$, define the function f_i by setting $f_i(x_1, \ldots, x_n) = U_j^n(x_1, \ldots, x_n)$ and if ξ_i is a constant, set $f_i(x_1, \ldots, x_n) = C_{\xi_i}^n(x_1, \ldots, x_n)$. Each f_i is in F by the preceding discussion. Since

$$f(x_1, \ldots, x_n) = g(f_1(x_1, \ldots, x_n), \ldots, f_k(x_1, \ldots, x_n))$$

and F is closed under composition, f is in F as desired.

Since the linear functions may be defined as the closure of addition and the identity and constant functions under composition and explicit transformations, we now know that every linear function is in F. Although not every function in F

is linear, it is true that every function in F is bounded by a linear function in the sense of the following theorem.

THEOREM 2. *If $f(x_1, \ldots, x_n)$ is in F, then there is a non-negative integer K such that $f(x_1, \ldots, x_n) < K \max(x_1, \ldots, x_n, 1)$ for all arguments $x_1, \ldots, x_n$.*

Proof. For any integer y, let us denote the length of the binary encoding of y by $l(y)$. It is immediate that $y < 2^{l(y)} \leqq 2 \cdot \max(y,1)$. By Definitions 11 and 12 of this appendix (and their implicit generalization), the encoding of the n-tuple $(x_1, \ldots, x_n)$ for a finite automaton is a tape of length $l[\max(x_1, \ldots, x_n, 1)]$. Let us introduce the abbreviation $\mathbf{x}$ for $x_1, \ldots, x_n$. Now by Theorem 1 of this appendix, there is a k such that $l[f(\mathbf{x})] < l[\max(\mathbf{x},1)] + k$. The theorem follows from this inequality since

$$f(\mathbf{x}) < 2^{l[f(\mathbf{x})]} < 2^k 2^{l[\max(\mathbf{x},1)]} \leqq 2^{k+1} \max(\mathbf{x},1)$$

APPENDIX 3

An arithmetization

In this appendix, we wish to arithmetize Turing machines which compute numerical functions, and to do so within a very restricted class of functions. We must abandon the prime-power representation of sequences and turn to a more economical one, as mentioned in the main body of this paper. We have chosen to work with concatenation of strings of symbols since this procedure seems to be suggested by the very operation of the Turing machines themselves. In so doing, we will follow Smullyan [8] and use the dyadic encoding of positive integers as strings of 1's and 2's. This encoding has the technical advantage, not shared by binary, of setting up a one-to-one correspondence between finite strings and numbers. We simply observe that any positive integer has a *unique* representation as a sum of terms of the form $a_i 2^i$ where a_i is either 1 or 2, and encode the number $a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_0 2^0$ as the string $a_n a_{n-1} \ldots a_0$.

In [8], Smullyan investigated a class of relations over the positive integers which he called the rudimentary relations. This is the smallest class of relations which includes (dyadic) concatenation and which is closed under explicit transformations, Boolean operations and bounded quantification. Here concatenation is the relation defined as follows: z is called the concatenation of x and y (written $z = xy$)([26]) if the dyadic encoding of z is $x_n x_{n-1} \ldots x_0 y_m y_{m-1} \ldots y_0$ where $x_n \ldots x_0$ and $y_m \ldots y_0$ are the dyadic encodings of x and y. We wish to employ rudimentary relations in our arithmetization, and to do so we now show that every rudimentary relation is in the class $\mathscr{E}^2_*$.

Grzegorczyk [3] showed that the class $\mathscr{E}^2_*$ is closed under explicit transformations, Boolean operations, and bounded quantification, and Smullyan [8, pp.

([26]) Throughout this appendix, if we need to refer to the product of x and y, we will denote it by $x \cdot y$; we reserve the notation xy for concatenation.

30–31, 77–80] showed that any class closed under these operations which contains the addition and multiplication relations will contain concatenation. In [3], it is shown also that $\mathscr{E}^2$ contains the addition and multiplication functions. To establish our result, we prove the following theorem.

THEOREM 1. *If the function f lies in $\mathscr{E}^2$, the relation F is in $\mathscr{E}^2_*$ where $F(x_1, \ldots, x_n) \equiv f(x_1, \ldots, x_{n-1}) = x_n$.*

Proof. Assuming $f \in \mathscr{E}^2$, let R and S be relations defined by setting $R(\mathbf{x},y) \equiv f(\mathbf{x}) \dot{-} y = 0$ and $S(\mathbf{x},y) \equiv y \dot{-} f(\mathbf{x}) = 0$. R and S are in $\mathscr{E}^2_*$ since their characteristic functions g and h may be defined by setting $g(\mathbf{x},y) = 1 \dot{-} (1 \dot{-} (f(\mathbf{x}) \dot{-} y))$ and $h(\mathbf{x},y) = 1 \dot{-} (1 \dot{-} (y \dot{-} f(\mathbf{x})))$. But $R \cap S$ is then a relation in $\mathscr{E}^2_*$ and it is precisely the desired relation since

$$(R \cap S)(\mathbf{x},y) \equiv R(\mathbf{x},y) \wedge S(\mathbf{x},y) \equiv f(\mathbf{x}) \leqq y \wedge y \leqq f(\mathbf{x}) \equiv f(\mathbf{x}) = y.$$

COROLLARY([27]). *The class of rudimentary relations is contained in the class $\mathscr{E}^2_*$.*

Now we can proceed with our arithmetization using rudimentary techniques; that is, by defining relations from concatenation by explicit transformations, Boolean operations and bounded quantification. Before we actually begin, we define relations which will be used in the later development.

$$x_1x_2\ldots x_n = y \equiv (\exists z_2,z_3,\ldots,z_{n-1})_{\leqq y} [z_2 = x_1x_2 \wedge z_3 = z_2x_3 \wedge \ldots \wedge z_{n-1} = z_{n-2}x_{n-1} \wedge y = z_{n-1}x_n],$$

$$xBy \equiv x = y \vee (\exists z)_{\leqq y}[xz = y] \qquad \text{(read ``x begins y''),}$$

$$xEy \equiv x = y \vee (\exists z)_{\leqq y}[zx = y] \qquad \text{(read ``x ends y''),}$$

$$xPy \equiv xBy \vee xEy \vee (\exists z_1,z_2)_{\leqq y}[z_1xz_2 = y] \qquad \text{(read ``x part of y''),}$$

$$\text{tally}(x) \equiv \sim 2Px \qquad \text{(read ``x is a tally,'' i.e. a string of 1's),}$$

$$l(x) = l(y) \equiv (\exists z)_{\leqq x}[\text{tally}(z) \wedge z \leqq x \leqq 2z \wedge z \leqq y \leqq 2z]$$
(read ``length of x equals length of y''),

$$l(x) = 5 \cdot l(y) \equiv (\exists z)_{\leqq x}[\text{tally}(z) \wedge (\exists w)_{\leqq y}[\text{tally}(w) \wedge l(y) = l(w) \; l(z) = l(x) \wedge wwwww = z]].$$

With these preliminaries behind us, we are prepared to arithmetize Turing machines (as defined in Appendix 2). That is, we can now encode descriptions of Turing machines and of computations by Turing machines as numbers (written in dyadic notation) and numerical functions, respectively. Then we will be able to express properties of Turing machines as properties of numerical functions as needed in §§4, 5 and 6 above. To do this we will assign to each state of a Turing

([27]) It is easily seen that a relation is accepted by a linear bounded automaton in the sense of [5] if and only if its characteristic function is in F'. Thus, this corollary with Theorem 4 yields the result announced in [5] that each rudimentary relation is accepted by some linear bounded automaton.

machine $Z = (S,m,s_0,f)$ a number of the form $11\ldots11$. In particular we agree to assign the number 1 to the initial state s_0 and the number 11 to the final state f. We denote the three symbols 0, 1 and β of the alphabet which may occur on tape by the numbers 1, 2 and 11 respectively, and denote the elements 1, -1 and 0 which occur in the range of the function m by 1, 2 and 11 respectively (i.e. 1 denotes a movement of the tape to the left, 2 a movement to the right, and 11 no movement). For convenience in remembering what states, tape symbols and shift instructions various numbers denote, we introduce the following abbreviations.

$St(x) \equiv \text{tally}(x)$ (x is the number of a state),

$s_0(x) \equiv x = 1$ (x is the number of state s_0),

$f(x) \equiv x = 11$ (x is the number of state f),

$A(x) \equiv x = 1 \vee x = 2 \vee x = 11$ (x is the number of a symbol in the alphabet),

$L(x) \equiv x = 1$ (x is the number denoting a left shift)

$R(x) \equiv x = 2$ (x denotes a right shift),

$N(x) \equiv x = 11$ (x denotes no shift of the tape).

A complete description of a Turing machine $Z = (S,m,s_0,f)$ (over the alphabet $(0,1,\beta)$) is obtained by listing s_0, f and m since knowing m we must know S for $m: A \times (S-\{f\}) \to A \times S \times \{1,-1,0\}$. Since we have agreed on notations for s_0 and f, we need only list m to specify Z. We give m as the table of quintuples (a,s,a',s',p) such that $m(a,s) = (a',s',p)$. Thus to define a number of a Turing machine Z we first define the set Quint of numbers of quintuples (a,s,a',s',p), the set Table of numbers which are numbers of tables (lists) of quintuples, the relation Quint which holds of x and y if x is a number of a quintuple which occurs in table number y, the set Fn of numbers of tables which are functions (that is tables of quintuples in which there are no two different quintuples with the same two first elements), and then finally the set TM of numbers which are numbers of Turing machines. TM is the set of all numbers Fn such that to each state s in $S-\{f\}$ there is a quintuple in the list beginning with the pair $(0,s)$, one beginning $(1, s)$, and one beginning (β,s) (so that the domain of the function is the entire set $A \times (S - \{f\})$).

To define Quint we must represent a quintuple of numbers by a single number. To do this we follow Quine [7] and Smullyan [8] to represent a sequence of numbers by their "separated concatenation," that is, by stringing them together one after the other with spacers between. These spacers are to have the property that they allow unique decomposition of the "separated concatenation" into the sequence it represents. To define Quint(x) we observe that each element y of the quintuple satisfies either $A(y)$, $R(y)$, $L(y)$, $N(y)$, or $St(y)$. Since the string 22 does not occur in any of these elements of a quintuple, we utilize 1221 as a spacer. Then a string which neither begins nor ends with 1221 but which contains four

occurrences of 1221 and no occurrences of 222 is uniquely interpretable as a quintuple (an ordered quintuple). For example, 21221111221112211112212 represents the quintuple (2,111,1,11,2). In general we define the set Quint as follows:

$$\begin{aligned}\text{Quint}(x) \equiv (\exists x_1,x_2,x_3,x_4,x_5)_{\leqq x}[&x_1 1221 x_2 1221 x_3 1221 x_4 1221 x_5 = x\\ &\wedge A(x_1) \wedge St(x_2) \wedge \sim f(x_2)\\ &\wedge A(x_3) \wedge St(x_4) \wedge (R(x_5) \vee L(x_5) \vee N(x_5))].\end{aligned}$$

Since a table is to be merely a sequence of quintuples, we use the corresponding device to represent a sequence of quintuples by a single number. Now the spacer is 12221 since 22 is the longest string of 2's occurring in any quintuple. The remaining definitions leading up to TM(x) should now be self-explanatory.

$$\begin{aligned}\text{Table}(x) \equiv (12221)Bx \wedge (12221)Ex \wedge \sim (2222)Px\\ \wedge (y)_{\leqq x}[(12221y12221)Px \wedge \sim (222)Py \rightarrow \text{Quint}(y)],\end{aligned}$$

$$\text{Quint}(x,y) \equiv \text{Quint}(x) \wedge \text{Table}(y) \wedge (12221x12221)Py,$$

$$\begin{aligned}\text{Quint}((x_1,x_2,x_3,x_4,x_5),y) \equiv (\exists x)_{\leqq y}[&\text{Quint}(x,y) \wedge x\\ &= x_1 1221 x_2 1221 x_3 1221 x_4 1221 x_5],\end{aligned}$$

$$\begin{aligned}\text{Agree}(x,y) \equiv (x_1,x_2)_{\leqq x}[&(x_1 1221 x_2 1221)Bx \wedge (x_1 1221 x_2 1221)By\\ &\rightarrow x = y],\end{aligned}$$

$$\text{Fn}(x) \equiv \text{Table}(x) \wedge (y,z)_{\leqq x}[\text{Quint}(y,x) \wedge \text{Quint}(z,x) \rightarrow \text{Agree}(y,z)],$$

$$\begin{aligned}\text{TM}(x) \equiv \text{Fn}(x) \wedge (y)_{\leqq z}\{&(\exists y_1,y_2,y_3,y_4)_{\leqq x}\\ &[\text{Quint}(y_1 1221 y1221 y_2 1221 y_3 1221 y_4)\\ &\vee (\text{Quint}(y_1 1221 y_2 1221 y_3 1221 y1221 y_4)\\ &\wedge y \neq 11)] \rightarrow (\exists z_1,z_2,z_3,w_1,w_2,w_3,v_1,v_2,v_3)_{\leqq x}\\ &[\text{Quint}((1,y,z_1,z_2,z_3),x) \wedge \text{Quint}((2,y,w_1,w_2,w_3),x)\\ &\wedge \text{Quint}((11,y,v_1,v_2,v_3),x)]\}.\end{aligned}$$

In addition to numbers for Turing machines we also need numbers for tapes and instantaneous descriptions, so that eventually we can analyze the computation of a function by a Turing machine numerically. We will assign the number $1221\bar{a}_1 1221\bar{a}_2 \ldots \bar{a}_{n-1} 1221\bar{a}_n 1221$ to the tape $a_1 a_2 \ldots a_{n-1} a_n$ where $a_i \in \{0,1,\beta\}$ and $\bar{0} = 1$, $\bar{1} = 2$, and $\bar{\beta} = 11$. Then to assign a number to an instantaneous description (t,s,p) we first replace the positive integer p by the number P of the first p symbols on the tape t. For example, in $(a_1 a_2 a_3 a_4,s,3)$ we replace 3 by the number of $a_1 a_2 a_3$, that is, by $1221\bar{a}_1 1221\bar{a}_2 1221\bar{a}_3 1221$. Then we assign to the description (t,s,p) the number $T12221S12221P$ where T is the number of the tape t and S is the number of the state s. Formally we obtain

$$\text{Tape}(x) \equiv (1221)Bx \wedge (1221)Ex \wedge \sim(222)Px \wedge (y)_{\leqq x}[(1221y1221)Px \wedge \sim(22)Py \to A(y)].$$

$$ID(x,y,z,w) \equiv [x = y12221z12221w \wedge \text{Tape}(y) \wedge \text{Tape}(w) \wedge wBy \wedge St(z)].$$

$$ID(x) \equiv (\exists y,z,w)_{\leqq x}[ID(x,y,z,w)].$$

In our definition of a computation by a Turing machine one instantaneous description led to another by means of the operation $\to$ defined in four steps. We now mirror each step in defining the yield function (where the first three are subcases of case one of the definition of $\to$).

$$\text{Yield}_{1a}(x,y,z) \equiv \text{TM}(z) \wedge (\exists x_1,x_2,a_1,z_1)_{\leqq x}(\exists a_2,z_2)_{\leqq y} [ID(x,x_1a_1x_2,z_1,x_1a_11221) \wedge ID(y,x_1a_2x_2,z_2,x_1a_21221) \wedge \text{Quint}((a_1,z_1,a_2,z_2,11),z)],$$

$$\text{Yield}_{1b}(x,y,z) \equiv \text{TM}(z) \wedge (\exists x_1,x_2,a_1,z_1)_{\leqq x}(\exists a_2,a_3,z_2)_{\leqq y} [ID(x,x_1a_11221a_3x_2,z_1,x_1a_11221) \wedge ID(y,x_1a_21221a_3x_2,z_2,x_1a_21221a_31221) \wedge \text{Quint}((a_1,z_1,a_2,z_2,1),z)],$$

$$\text{Yield}_{1c}(x,y,z) \equiv \text{TM}(z) \wedge (\exists x_1,x_2,a_1,a_3,z_1)_{\leqq x} (\exists a_2,z_2)_{\leqq y} {}_f\text{Quint}((a_1,z_1,a_2,z_2,2),z) \wedge ID(x,x_1a_31221a_11221x_2,z_1,x_1a_31221a_11221) \wedge ID(y,x_1a_31221a_21221x_2,z_2,x_1a_31221)],$$

$$\text{Yield}_2(x,y,z) \equiv \text{TM}(z) \wedge (\exists x_1,a_1,z_1)_{\leqq x}(\exists a_2,z_2)_{\leqq y} [\text{Quint}((a_1,z_1,a_2,z_2,1),z) \wedge ID(x,x_1a_11221,z_1,x_1a_11221) \wedge ID(y,x_1a_21221111221,z_2,x_1a_21221111221)],$$

$$\text{Yield}_3(x,y,z) \equiv \text{TM}(z) \wedge (\exists x_1,a_1,a_3,z_1)_{\leqq x}(\exists a_2,z_2)_{\leqq y} [\text{Quint}((a_1,z_1,a_2,z_2,2),z) \wedge ID(x,x_1a_31221a_11221,z_1,x_1a_31221a_11221) \wedge ID(y,x_1a_31221a_21221,z_2,x_1a_31221)],$$

$$\text{Yield}_4(x,y,z) \equiv \text{TM}(z) \wedge (\exists x_1,a_1,a_3,z_1)_{\leqq x}(\exists z_2)_{\leqq y} [\text{Quint}((a_1,z_1,11,z_2,2),z) \wedge ID(x,x_1a_31221a_11221,z_1,x_1a_31221a_11221) \wedge ID(y,x_1a_31221,z_2,x_1,a_31221)],$$

$$\text{Yield}(x,y,z) \equiv [\text{Yield}_{1a}(x,y,z) \vee \text{Yield}_{1b}(x,y,z) \vee \text{Yield}_{1c}(x,y,z) \vee \text{Yield}_2(x,y,z) \vee \text{Yield}_3(x,y,z) \vee \text{Yield}_4(x,y,z)].$$

Since the relation Yield is in $\mathscr{E}^2_*$, the following definition shows that the function Yield of two arguments is in $\mathscr{E}^2$. The meaning of this function should be clear enough: Yield(x,z) is the Gödel number of the instantaneous description

which immediately follows instantaneous description number x on Turing machine number z.

$$\text{Yield}(x,z) = \min y \leqq x \cdot z\ [\text{Yield}(x,y,z)].$$

A computation on a Turing machine ends when an instantaneous description is reached in which the state is the final state f. Thus we define $\text{Fin}(x)$ as follows to be the set of numbers x which are numbers of instantaneous descriptions involving the state f.

$$\text{Fin}(x) \equiv (\exists y,z)_{\leqq x}[ID(x,y,11,z)].$$

Now we wish to arithmetize computations of numerical functions, but before we can do this, we must find a method for translating non-negative integers encoded in binary into positive integers encoded in dyadic. Specifically, we need a function e in $\mathscr{E}^2$ mapping the non-negative integers one-to-one into the positive integers. We have, in fact, been using such a function implicitly in arithmetizing tapes when we assigned the Gödel number 1 to the integer 0 and the Gödel number 2 to the integer 1. Extending this assignment to the entire binary representation of any non-negative integer results in a corresponding dyadic representation of a positive integer. In fact, this replacement of 0 by 1 and 1 by 2 throughout the binary representation of the integer n results in the dyadic representation of $n + 2^{l(n)-1} + \ldots + 2^1 + 2^0 = n + 2^{l(n)} - 1$. Thus, defining the function e by setting $e(n) = n + 2^{l(n)} - 1$, we have the desired translation of non-negative integers into positive integers. We now must show that the relation ε of two arguments is in $\mathscr{E}^2_*$, where $\varepsilon(x,y) \equiv e(x) = y$.

To accomplish this, we first give definitions in $\mathscr{E}^2$ of the remainder function r (as was originally done in [3]), of the function t whose value is twice the first argument if the second argument is 0 and is exactly the first argument otherwise, and finally of $2^{l(x)}$. Then, since $\mathscr{E}^2$ is closed under addition and proper subtraction (see [3]), we will have shown that the function e is in $\mathscr{E}^2$, and by Theorem 1, ε is in $\mathscr{E}^2_*$ as desired. The definitions of r, t, and $2^{l(x)}$ are:

$$\left.\begin{aligned} r(0,y) &= 0 \\ r(s(x),y) &= \sigma[s(r(x,y)),1 \div (y \div s(r(x,y)))] \\ r(x,y) &\leqq y \end{aligned}\right\} \quad r(x,y) \text{ is the remainder upon dividing } x \text{ by } y.$$

$$\left.\begin{aligned} t(x,0) &= 2 \cdot x \\ t(x,s(y)) &= x \\ t(x,y) &\leqq 2 \cdot x \end{aligned}\right\} \quad t(x,y) = 2 \cdot x \text{ if } y = 0,\ x \text{ otherwise.}$$

$$\begin{aligned} 2^{l(0)} &= 2, \\ 2^{l(s(x))} &= t[2^{l(x)}, r(s(x), 2^{l(x)})], \\ 2^{l(x)} &\leqq 2 \cdot (x+1). \end{aligned}$$

Now that we have shown that the relation which holds of x and y if and only if $e(x) = y$ is in $\mathscr{E}^2_*$, we are prepared to analyze computations of numerical functions. Such a computation was defined to be a sequence of instantaneous descriptions in which the initial tape has the form $\bar{x}_1\beta\bar{x}_2\beta \ldots \bar{x}_{n-1}\beta\bar{x}_n$ and the final tape

has the form $\bar{x}_0$, where $\bar{x}_i$ is the binary encoding of the number x_i. We begin with a definition of the relation Code where Code(y,x) holds if and only if y is the Gödel number of the tape $\bar{x}$. From this relation, we can then define the function Decode which gives the number (if any) represented by the tape in a final instantaneous description, and the relation Encode$_n$ for each n where Encode$_n(x_1,\ldots,x_n,y)$ holds if and only if y is the Gödel number of the tape $\bar{x}_1\beta\ldots\bar{x}_{n-1}\beta\bar{x}_n$. Finally, from Encode$_n$ we then define the function Init$_n$ which gives the Gödel number of the initial instantaneous description whose tape bears the binary encoding of its n arguments in order. We conclude with these definitions.

$$\begin{aligned}
\text{Code}(y,x) &\equiv \text{Tape}(y)\wedge(\exists u)_{\leqq y}\{e(x)=u\wedge(z,w)_{\leqq y}\{[zBu\wedge wEz\\
&\quad\wedge(w=1\vee w=2)]\rightarrow(\exists v)_{\leqq y}[vBy\wedge l(v)=5\cdot l(z)\wedge(1221w)Ev]\}\}\\
\text{Decode}(x) &= \min y\leqq x\{ID(x)\wedge(\exists z)_{\leqq x}[(z12221)Bx\wedge\text{Code}(z,y)]\},\\
\text{Encode}_n(x_1,\ldots,x_n,x) &\equiv (\exists y_1,\ldots,y_n)_{\leqq x}[\text{Code}(y_1,x_1)\wedge\ldots\\
&\quad\wedge\text{Code}(y_n,x_n)\wedge x=y_1 11 y_2\ldots y_{n-1}11y_n],\\
\text{Init}_n(x_1,\ldots,x_n) &= \min w\leqq 2^{6\cdot n+19}\cdot[(x_1+1)\cdot\ldots\cdot(x_n+1)]^5\\
&\quad\{(\exists x)_{\leqq w}[\text{Encode}_n(x_1,\ldots,x_n,x)\\
&\quad\wedge(\exists y)_{\leqq x}\cdot[l(y)=9\wedge yBx\wedge x12221112221y]]\}.
\end{aligned}$$

The bound on W is determined by the constraints that $l(w)=l(x)+20$ and $l(x)=5\cdot(l(x_1)+\ldots+l(x_n))+4+6\cdot(n-1)$ so that

$$\begin{aligned}
w<2\cdot 2^{l(w)} &= 2\cdot 2^{5\cdot(l(x_1)+\ldots+l(x_n))+18+6\cdot n}\\
&= 2^{6\cdot n+19}\cdot(2^{l(x_1)}\cdot\ldots\cdot 2^{l(x_n)})^5\\
&\leqq 2^{6\cdot n+19}\cdot[(x_1+1)\cdot\ldots\cdot(x_n+1)]^5.
\end{aligned}$$

BIBLIOGRAPHY

1. Martin Davis, *Computability and unsolvability*, McGraw-Hill, New York, 1958.

2. C. C. Elgot, *Decision problems of finite automata design and related arithmetics*, Trans. Amer. Math. Soc. **98** (1961), 21–51.

3. Andrej Grzegorczyk, *Some classes of recursive functions*, Rozprawy Matematyczne, Warsaw, 1953.

4. S. C. Kleene, *Introduction to metamathematics*, Van Nostrand, Princeton, N. J., 1952.

5. John Myhill, *Linear bounded automata*, WADD Technical Note, 60–165, Univ. Pennsylvania Report Nr. 60–22, June 1960.

6. Rózsa Péter, *Rekursive funktionen*, Akadémiai Kiadó, Budapest, 1957, pp. 76–86.

7. W. V. Quine, *Concatenation as a basis for arithmetic*, J. Symb. Logic **11** (1946), 105–114.

8. R. M. Smullyan, *Theory of formal systems*, Annals of Mathematics Studies, No. 47, Princeton Univ. Press, Princeton, N. J., 1961.

9. Alfred Tarski, *A decision method for elementary algebra and geometry*, Project RAND Report R-109, The RAND Corporation, Santa Monica, Calif., 1957.

PRINCETON UNIVERSITY,
PRINCETON, NEW JERSEY

DARTMOUTH COLLEGE,
HANOVER, NEW HAMPSHIRE