Notre Dame Journal of Formal Logic
Volume 35, Number 4, Fall 1994

Elementary Functions and Loop Programs

ZLATAN DAMNJANOVIC

Abstract We study a hierarchy $\{\mathcal{L}_2^k\}$ of Kalmàr elementary functions on integers based on a classification of LOOP programs of limited complexity, namely those in which the depth of nestings of LOOP commands does not exceed two. It is proved that n-place functions in $\mathcal{L}_2^k$ can be enumerated by a single function in $\mathcal{L}_2^{k+2}$, and that the resulting hierarchy of elementary predicates (i.e., functions with 0,1-values) is proper in that there are $\mathcal{L}_2^{k+2}$ predicates that are not in $\mathcal{L}_2^k$. Along the way the rudimentary predicates of Smullyan are classified as $\mathcal{L}_2^2$.

1 We focus here on a relatively small class of computable functions, the class of *elementary* functions, first introduced by Csillag and Kalmàr in the 1940s. (By a function is always meant a possibly partial function of nonnegative integers, mapping n-tuples of members of $\mathbb{N}$ into $\mathbb{N}$.) This is the smallest class of functions that includes the successor $x+1$, the projection functions $U_i^n (1 \leq i \leq n)$, $x \dot{-} y$, $x+y$ and $x \cdot y$, and is closed under composition and bounded sums and products.[1] Essentially, the elementary functions are obtained by composition and bounded iteration of the usual arithmetical operations of $+$ and $\cdot$.

Elementary functions are of particular interest if we are seeking to characterize a "natural" example of a concept of computability narrower than the standard Church-Turing notion. The standard notion allows arbitrarily complex computations and places no bounds on the amount of time or storage space required to complete the computation. For this reason many authors, most notably computer scientists, have taken the view that the Church-Turing model is largely irrelevant from the standpoint of actual computing practice and that the limits on the amount of time and space available must be an essential part of a realistic concept of computation. In attempting to define a theoretically interesting concept that would meet these concerns, it is natural to follow what could best be described as a foundational "predicative" procedure: we may think of the class of functions computable in such a preferred sense as constructed in stages, so that the complexity of computations of the functions at the later stages is in some way bounded by functions obtained at the earlier stages. Ritchie [16] thus inductively defines the class of "predictably computable" functions as follows:

Received May 17, 1993; revised November 30, 1994

the functions computable by finite automata form the initial stage, and at successor stages we introduce the functions computable by Turing machines that use in their computations an amount of tape bounded by some function obtained at the previous stage. (Finite automata are also described in Davis and Weyuker [8], chapter 8.)

Ritchie's "predictably computable" functions turn out to be precisely the elementary functions.[2] In fact, the class of elementary functions has the property that the total functions belonging to the class are exactly those computable by some program, formulated in any one of the standard programming formalisms, in time and space bounded by a function that belongs to that very class. (See, e.g., Theorem 4.1.1 in Cutland [6].) All this suggests that the elementary functions embody an intuitively significant conception of arithmetic that can serve as a basis for a plausibly restricted model of computation. It is widely held among computer scientists that such a model would be comprehensive enough to include all functions computable "in practice."[3]

In §2 we introduce a programming language of Elementary LOOP Programs (ELPs) that allows only computations of elementary functions, and in §3 state some basic properties of such programs and of elementary functions. There we also describe a particular hierarchy, $\{\mathcal{L}_2^k\}$, of elementary functions, originally introduced in Goetze and Nehrlich [9], which is defined in terms of a syntactic classification of ELPs. §4 describes a well known class of predicates—Smullyan's rudimentary predicates—which shall serve as a basis for our own arithmetization of ELPs in §§5–6. (Predicates are functions that take only 0 and 1 as values.) Our main results are in §§7–8: in §7 we classify the rudimentary predicates in the hierarchy $\{\mathcal{L}_2^k\}$ by showing that they all belong to the class $\mathcal{L}_2^2$. Then in §8 we establish enumeration theorems for the hierarchy $\{\mathcal{L}_2^k\}$ and also prove that the hierarchy of predicates is a proper one in a weak sense. (That the hierarchy of *functions* $\{\mathcal{L}_2^k\}$ is proper was established by a different method by Goetze and Nehrlich in [9].) Although the project of carrying out arithmetization by such extremely elementary means is of evident interest, it also makes it possible to set up strongly constructive semantics for arithmetic along the lines of Kleene's recursive realizability that relies only on elementary functions (see Damnjanovic [7]).

2 A *register machine* or abacus consists of a finite number of registers, each of which may contain a finite number (possibly zero) of counters. We consider several programming languages interpreted by reference to such computing devices. A program consists of a finite sequence of instructions, which we may assume are labeled by numerals. The instructions refer to registers by means of variables, distinct registers being associated with distinct variables out of an infinite list of variables. The contents of the registers are then referred to as the *values* of those variables. One such language—which we call RM, for register machines—has instructions of the following kind:

1. "add 1 to X_i";
2. "subtract 1 from X_i if X_i is not empty; otherwise, go to m," where m is a label of some instruction;
3. "GO TO m."

The interpretation of such instructions is obvious, given that $X_1, X_2, \ldots$ is the list of

variables. RM programs are usually represented in the form of directed graphs, or "flow graphs," in which the nodes correspond to instructions of type 1 and 2 and the arrows connecting the nodes express the "GO TO" commands; this is how we shall describe RM programs in §7. The language of RM programs is universal in that every partial recursive function can be shown to be computable, in an appropriate sense, by some RM program.[4]

Our primary focus, however, is on LOOP programs, originally introduced by Meyer and Ritchie in [13]. We define these inductively. The instructions include:

1. $V_i \leftarrow V_i + 1$ meaning "add 1 to V_i";
2. $V_i \leftarrow 0$ meaning "set V_i to 0";
3. $V_i \leftarrow V_j$ meaning "assign the value of V_j to V_i leaving V_j unchanged."

(We list the variables as $V_0, V_1, V_2, \ldots$ to distinguish them from the variables in RM programs. We let $\underline{V}_n$ be the initial segment of this list up to and including V_n.) These instructions we call *arithmetical*, and any finite sequence of such instructions is a LOOP program. In addition, given a LOOP program **P**, the sequence of instructions,

LOOP V_i
  P
END

is a loop segment, and also constitutes a LOOP program, which is interpreted as follows:

> if the value of V_i is k, perform **P** exactly k times and then continue with the instruction following END, if there is any.

The variable V_i may occur in **P** and its value may change in the course of the process. The number of iterations of **P**, however, is determined by the value of V_i before the computation begins. This completes the definition of LOOP programs.[5]

What is noteworthy about LOOP programs is that they provide a direct way of expressing primitive recursion, e.g., $x + y$ is defined by primitive recursion from $x + 1$. Consider the program,

(1)
LOOP X
  $Y \leftarrow Y + 1$
END

(We use $X, Y, Z, U, \ldots$ as "metalinguistic" variables for V_i.) If the initial values of X and Y are x and y, respectively, then at the end of the computation their values will be x and $x + y$, respectively. If we designate the registers X and Y as the input registers, in that order, writing IN X,Y, and Y as the output register, writing OUT Y, we may regard the process as the computation of $x + y$. Then the LOOP program,

(2)
$Y \leftarrow 0$
LOOP Z
  (1)
END

is easily seen to compute $x \cdot z$ if IN X, Z and OUT Y. The same idea is used to show that x^y, which is defined by primitive recursion from $x \cdot y$, is also computable by a

LOOP program. In fact, the functions computable by LOOP programs are precisely the primitive recursive (p.r.) functions (cf. [13], and [8], chapter 13).

We also consider an expanded language of LOOP_μ programs, which differ from LOOP programs in that complex instructions of the form,

WHILE $X \neq 0$ DO
  P
END

are also allowed, **P** being some previously given LOOP_μ program, possibly containing occurrences of the variable X. This is interpreted as follows:

> perform program **P** repeatedly until the value of X is 0; if and when that happens, continue with the instruction following END, if there is any.

If **P** contains X, the repeated applications of **P** may result in changes of the value of X; if X never assumes value 0 in the process, the computation will never terminate. Instructions of this type allow us to directly express the (unbounded) least number operator μ, as in,

$$(\mu y)\, \mathcal{A}(\vec{x}_n, y) \tag{3}$$

as long as the characteristic function of the predicate $\mathcal{A}(\vec{x}_n, y)$ is computable by some LOOP_μ program **Q** in the sense that **Q** terminates with value 0 in the output register Z whenever $\mathcal{A}(\vec{x}_n, y)$ holds, and with value 1 in Z if $\neg\mathcal{A}(\vec{x}_n, y)$, the registers $X_1, \ldots, X_n, Y$ being the input registers with initial values $x_1, \ldots, x_n, y$, respectively, and all other registers in **Q** initially being empty. (The variables for the other registers are called *local*, to be distinguished from the *input* and *output variables*.) Then the LOOP_μ program,

$Y \leftarrow 0$
WHILE $Z \neq 0$ DO
  Q [IN $X_1, \ldots, X_n, Y$; OUT Z]
  $Y \leftarrow Y + 1$
END

computes (3) if IN $X_1, \ldots, X_n$ and OUT Y. The language of LOOP_μ programs is universal and thus equivalent to the language of RM programs. (See [8], pp. 311–312.)

Hence the language of LOOP programs is strictly weaker than that of LOOP_μ programs, since there exist recursive functions that are not p.r. Still weaker languages are the fragments of the LOOP language in which a bound is placed on the number of nested applications of LOOP-END commands. Program (2), for instance, has depth of nesting two, whereas the LOOP program for x^y obtained from (2) in this same fashion would have depth of nesting three. We shall study the LOOP programs in which the maximum depth of nesting is ≤ 2. It turns out that the functions computable by such programs are precisely the elementary functions, and so we call such LOOP programs *elementary*, or ELP for short.[6]

We follow [9] in adopting a classification of LOOP programs according to the number of their subprograms of maximal depth of nesting. In general, given two classes, $\mathcal{A}$ and $\mathcal{B}$, of programs, we let $\mathcal{AB}$ stand for the class of programs of the form $\begin{matrix}\mathbf{P}\\ \mathbf{Q}\end{matrix}$ where $\mathbf{P} \in \mathcal{A}$ and $\mathbf{Q} \in \mathcal{B}$, i.e., of programs that result from composition of programs

from $\mathcal{A}$ and $\mathcal{B}$, respectively. (Program **P** is applied first.) And given a class $\mathcal{A}$, we let $\langle\mathcal{A}\rangle$ be the class that includes the programs in $\mathcal{A}$ as well as all programs of the form,

LOOP X
  P
END

where $\mathbf{P} \in \mathcal{A}$. We let $\mathbf{L}_0$ be the class of LOOP programs with no LOOP commands. Then set

$$\mathbf{L}_1^0 := \mathbf{L}_0 \qquad \text{and} \qquad \mathbf{L}_1^{k+1} := \mathbf{L}_1^k(\langle\mathbf{L}_0\rangle\mathbf{L}_0).$$

Let $\mathbf{L}_1 := \cup_{k\in\mathbb{N}}\mathbf{L}_1^k$ and define

$$\mathbf{L}_2^0 := \mathbf{L}_1 \qquad \text{and} \qquad \mathbf{L}_2^{k+1} := \mathbf{L}_1^k(\langle\mathbf{L}_1\rangle\mathbf{L}_1).$$

Then let $\mathcal{L}_n^k (k \geq 0, n = 1, 2)$ denote the class of functions computable by ELPs in $\mathbf{L}_n^k$.

3 Following [13], we define a series of functions f_k as follows:

$$\begin{aligned} f_0(0) &:= 1 \\ f_0(1) &:= 2 \\ f_0(x) &:= x+2 \text{ for } x \geq 2 \\ f_{n+1}(x) &:= f_n^x(1) \text{ where } f_n^x(y) := \underbrace{f_n(f_n(\ldots(f_n}_{x \text{ times}}(y))\ldots)). \end{aligned}$$

(Thus $f_n^0(y) = y$.) In particular, $f_1(x) = 2x$ and $f_2(x) = 2^x$ if $x \geq 1$. We summarize some useful properties of these functions.[7]

Lemma 3.1

1. *For any* $m, n,\ m < n \Rightarrow f_m \circ f_n(x) \leq f_n \circ f_0(x)$.
2. *For any* $m, n,\ m < n \Rightarrow$ *for any* $j, k,\ f_m^j \circ f_n^k(x) \leq f_n^k \circ f_0^j(x)$.
3. *For any* $n,\ f_n^x(y) \leq f_{n+1}(x+y)$.
4. *For any* $k \geq 1,\ f_2^k(x) \cdot y \leq f_2^k \circ f_1(x+y)$.
5. *For any* $k \geq 1, [f_2^{k+1}(x)]^y \leq f_2^{k+1} \circ f_1(x+y)$.
6. *For any* $k \geq 1,\ f_2^{k+1}(x) \cdot f_2^{k+1}(x) \leq f_2^{k+1} \circ f_0(x)$.

Proof: (1) and (3) are proved in [9], Lemma 4 (a) and (b), and (2) follows from (1). For (4), we have that

$$\begin{aligned} f_2^k(x) \cdot y \leq f_2(f_2^{k-1}(x)) \cdot 2^y &= 2^{f_2^{k-1}(x)} \cdot 2^y = 2^{f_2^{k-1}(x)+y} \\ &= f_2(f_2^{k-1}(x)+y) \leq f_2 \circ f_0^y \circ f_2^{k-1}(x) \\ &\leq f_2 \circ f_2^{k-1} \circ f_0^y(x) \leq f_2^k \circ f_1(x+y). \end{aligned}$$

For (5),

$$\begin{aligned} [f_2^{k+1}(x)]^y &= (2^{f_2^k(x)})^y = 2^{f_2^k(x)\cdot y} \\ &= f_2(f_2^k(x) \cdot y) \leq f_2 \circ f_2^k \circ f_1(x+y) \\ &= f_2^{k+1} \circ f_1(x+y). \end{aligned}$$

Finally, for (6),

$$f_2^{k+1}(x) \cdot f_2^{k+1}(x) = 2^{f_2^k(x)} \cdot 2^{f_2^k(x)} = 2^{2 \cdot f_2^k(x)} = 2^{f_1 \circ f_2^k(x)} \leq 2^{f_2^k \circ f_0(x)} \leq f_2^{k+1} \circ f_0(x).$$

We shall often use these results without explicitly referring to Lemma 3.1.

Let $\mathbf{P}$ be an ELP. The *running time* of $\mathbf{P}$, $T_{\mathbf{P}}(\vec{x}_n)$, is the total number of times arithmetical instructions are executed during the computation of $\mathbf{P}$ started on a given input $\vec{x}_n$. Thus, e.g., if $\mathbf{P} \in \mathbf{L}_0$ then $T_{\mathbf{P}}(\vec{x}_n) = \mathrm{lth}(\mathbf{P})$, and all variables in $\mathbf{P}$, including the output variable, have values $\leq \max(\vec{x}_n) + \mathrm{lth}(\mathbf{P}) \leq f_0^{\mathrm{lth}(\mathbf{P})}(\max(\vec{x}_n))$. (Here the *length* of $\mathbf{P}$, $\mathrm{lth}(\mathbf{P})$, is simply the number of instructions in $\mathbf{P}$.) If $\mathbf{P}$ has LOOP instructions, then some arithmetical instructions may be executed more than once: e.g., if $\mathbf{P}$ is of the form

LOOP X
  Q
END

for some ELP $\mathbf{Q} \in \mathbf{L}_0$. Then

$$T_{\mathbf{P}}(\vec{x}_n) \leq \mathrm{lth}(\mathbf{Q}) \cdot x \leq \mathrm{lth}(\mathbf{Q}) \cdot \max(\vec{x}_n) \leq 2^{\mathrm{lth}(\mathbf{Q})} \cdot \max(\vec{x}_n) = f_1^{\mathrm{lth}(\mathbf{P})-2}(\max(\vec{x}_n))$$

where x is the initial value of the variable X. On the other hand, if $V_{\mathbf{P}}(\vec{x}_n)$ is the maximum of the values any variable in $\mathbf{P}$ assumes during the computation started with input $\vec{x}_n$, then $V_{\mathbf{P}}(\vec{x}_n)$ can increase at most by 1 at each step of the computation.

Hence, in general,

$$V_{\mathbf{P}}(\vec{x}_n) \leq \max(\vec{x}_n) + T_{\mathbf{P}}(\vec{x}_n)$$

and in the above example, where $\mathbf{P} \in \langle \mathbf{L}_0 \rangle$, we have that,

$$\begin{aligned} V_{\mathbf{P}}(\vec{x}_n) \leq \max(\vec{x}_n) + f_1^{\mathrm{lth}(\mathbf{P})-2}(\max(\vec{x}_n)) &\leq 2(f_1^{\mathrm{lth}(\mathbf{P})-2}(\max(\vec{x}_n))) \\ &\leq f_1^{\mathrm{lth}(\mathbf{P})}(\max(\vec{x}_n)). \end{aligned}$$

This type of argument can be extended to establish upper bounds on the running time and the maximum value of the variables for any ELP. In particular, we have the following theorem.

Theorem 3.2 *Suppose $k \geq 1$ and $\mathbf{P} \in \mathbf{L}_2^k$. Then both $T_{\mathbf{P}}(\vec{x}_n)$ and $V_{\mathbf{P}}(\vec{x}_n)$ have an upper bound in $f_2^k \circ f_1^{\mathrm{lth}(\mathbf{P})} \circ f_0^{2\,\mathrm{lth}(\mathbf{P})}(\max(\vec{x}_n))$.*

We omit the proof, which may be obtained by filling in the details of the argument given in [9], p. 259, Lemma 5. (See also [8], pp. 301–302, proof of Theorem 2.2.) The bound on $V_{\mathbf{P}}(\vec{x}_n)$ is also an upper bound on the rate of growth of the functions computed by the programs in $\mathbf{L}_2^k$.

Theorem 3.3 *For each $k \geq 1$, for any $p, q \geq 0$, $(f_2^k \circ f_1^p \circ f_0^q) \in \mathcal{L}_2^k$.*

This easily follows from the definition of the classes $\mathbf{L}_2^k$ of ELPs and the fact that $f_2^k \in \mathcal{L}_2^k$ for each k.

4 We now consider the language BA of Bounded Arithmetic. The nonlogical symbols of BA include the constant "0" denoting 0, a single 1-place function symbol for the successor function, and two 3-place predicates which express the graphs of the addition and multiplication functions on $\mathbb{N}$, respectively. The logical apparatus of BA differs from that of the language of First Order Arithmetic only in that the unbounded quantifiers $\exists x$ and $\forall x$ are replaced by the bounded quantifiers $(\exists x < y)$ and $(\forall x < y)$. Aside from that, terms and formulas are built up and interpreted in the usual way. (If $y = 0$, $(\exists x < y)\mathcal{A}(x, \vec{x}_n)$ is false and $(\forall x < y)\mathcal{A}(x, \vec{x}_n)$ is true for any $\vec{x}_n$.) Following Smullyan [18], pp. 30–31, we call the sets and relations of integers definable in the language BA *constructive arithmetical*, or CA for short. The CA predicates are closed under *explicit transformation*, namely the operations of permutation and identification of variables, substitution of a constant for a variable, and introduction of "dummy" variables.

Smullyan found it convenient to represent positive integers in *dyadic* notation for the purposes of developing the fundamentals of recursion theory on $\mathbb{N}$. The dyadic representation is a slight variant of the binary notation: there are two digits, "1" and "2," and a string $a_0, \ldots, a_n$ of these digits is the dyadic numeral for $a_n \cdot 2^n + \cdots + a_0 \cdot 2^0$. This determines a 1-1 correspondence ν between the set D of such strings and the positive integers, the nth string in the lexicographic ordering of the dyadic strings being the numeral for n. We think of the *concatenation* operation on these numerals as determining a relation $C(x, y, z)$ on integers: $C(x, y, z)$ holds iff the dyadic numeral $\overline{z}$ for z is the result of successively writing the digits of the dyadic numeral $\overline{y}$ for y to the right of the last digit of the dyadic numeral $\overline{x}$ for x. We sometimes write $x * y = z$ if this is the case. The dyadic numerals for x and y are then said to be *parts* of the numeral for z. It turns out that $x * y = (x \cdot 2^{|y|}) + y$, where $|y|$, sometimes written $\ell(y)$, is the *length* of the dyadic numeral $\overline{y}$ for y.[8]

Smullyan introduced the class of *rudimentary relations* (RUD) as the smallest class of relations on $\mathbb{N}$ that contains $C(x, y, z)$ and is closed under the logical operations $\neg$, &, $\vee$ and $\rightarrow$, the bounded quantifiers $(\forall x < y)$ and $(\exists x < y)$, and explicit transformation. He showed that $C(x, y, z)$ is CA. (See [18], pp. 77–81.) Formally, the variables range over dyadic strings, but the bounded quantifiers are interpreted with respect to the natural ordering $<$ on $\mathbb{N}$: e.g., $(\forall x < y)\mathcal{A}(x, \vec{y}_n)$ holds just in case $y = 0$ or else $\mathcal{A}(x, \vec{y}_n)$ is true for all strings x such that $\nu(x) < \nu(y)$; analogously for $(\exists x < y)$.

The classes CA and RUD were shown to coincide by Bennett [1].[9] Furthermore, the same class of relations can be equivalently characterized as the smallest class of relations on $\mathbb{N}$ containing the polynomial relations that is closed under bounded quantification. (A relation $R(\vec{x}_n)$ is *polynomial* iff $R(\vec{x}_n) \Leftrightarrow P(\vec{x}_n) = 0$ for some polynomial $P(\vec{x}_n)$ with integral coefficients.) This latter class has also been called the class of *bounded arithmetical relations*. (See Harrow [10] for more details.) It follows from these results that the CA relations are closed under bounded minimization—so that $(\mu y < z)\mathcal{A}(y, \vec{x}_n)$ is CA if $\mathcal{A}(y, \vec{x}_n)$ is—as well as under quantification bounded by a polynomial.

Smullyan showed that all of the formal machinery necessary for establishing the basic results of recursion theory, having to do in particular with Gödel numberings and codings of computations, can be developed using only rudimentary relations,

and thus within CA. This fact was employed by Ritchie [16] in arithmetizing Turing machine computations as part of his analysis of elementary functions as "predictably computable." For our purposes, however, it is convenient to use what appears to be a more elementary formal apparatus than the one presented by Smullyan [18]. We focus on a subclass of RUD, called the *positive rudimentary relations*, PRUD for short. This class was originally introduced by Bennett [1], who proved that the graphs of $x + y$, $x \cdot y$ and x^y are PRUD. The advantage of working with PRUD is that it can be shown (see Theorem 4.2 below) that a wide class of rapidly growing functions defined by primitive recursion from functions with PRUD graphs themselves have PRUD graphs.[10] This will allow us to describe the process of computation of a LOOP program for a given input in a direct way.

To describe PRUD, we introduce quantifiers over subwords: given a predicate $\mathcal{A}(x, \vec{y}_n)$, let $(\forall x \lhd z)\mathcal{A}(x, \vec{y}_n)$, respectively $(\exists x \lhd z)\mathcal{A}(x, \vec{y}_n)$, hold if and only if $\mathcal{A}(u, \vec{y}_n)$ holds for any (respectively, some) string u that is a part (or a substring) of the string z. Thus, e.g., $(\forall x \lhd 22)\mathcal{A}(x, \vec{y}_n)$ holds iff both $\mathcal{A}(2, \vec{y}_n)$ and $\mathcal{A}(22, \vec{y}_n)$ are true. (Note that $22 = \overline{5}$ and $2 = \overline{1}$.) On the other hand, we interpret $(\exists x_{|x|\le|z|})\mathcal{A}(x, \vec{y}_n)$ to hold iff $\mathcal{A}(u, \vec{y}_n)$ is true for some string u such that $|u| < |z|$. In general, this restricts the values of the variable x in $(\exists x_{|x|\le|z|})$ to a set of strings that includes the dyadic numerals for all integers $\le \nu(z)$, and possibly some more.

The smallest class of relations on D (and thus on $\mathbb{N}$) that includes $C(x, y, z)$ and is closed under $\&$, $\vee$, $(\forall x \lhd z)$ and $(\exists x \lhd z)$, are called *strictly rudimentary*. The PRUD relations are in addition closed under $(\exists x_{|x|\le|z|})$. (It is easily shown that $(\exists x \lhd z)$ can be defined in terms of $(\exists x_{|x|\le|z|})$ and the remaining operations.) Those PRUD relations R that have complements $\neg R$ that are also PRUD we call *total positive rudimentary* (TOTAL). (In [1] they are called *strongly rudimentary*.) Then strictly rudimentary relations are PRUD, and PRUD $\subseteq$ RUD. Moreover, the following closure properties obtain:

Lemma 4.1

1. *If R is* TOTAL *and S is* PRUD, *then $R \to S$ is* PRUD.
2. *Let $a, b \in \mathbb{N}$, and let $R(x, \vec{y}_n)$ be a* PRUD *relation. Then*

$$(\exists x_{|x|\le a\cdot|z_1*\cdots*z_m|+b})\, R(x, \vec{y}_n)$$

   is also PRUD.
3. *If R is strictly rudimentary, then so is $\neg R$.*
4. *If R is strictly rudimentary, then R is* TOTAL.

(1) is obvious, (4) follows from (3), and (2) and (3) were proved in [1]. From this point on we use the abbreviated notation $(\exists x \ll z)$ in place of the more cumbersome $(\exists x_{|x|\le|z|})$. Then $(\exists x \ll a \cdot z + b)$ will be short for $(\exists x_{|x|\le a\cdot|z|+b})$.

The important property of PRUD relations mentioned earlier is summarized in the following.

Theorem 4.2 (Proskurin) *Suppose*

$$f(\vec{y}_m, 0) := g(\vec{y}_m) \quad \textit{and} \quad f(\vec{y}_m, z+1) := h(f(\vec{y}_m, z), \vec{y}_m, z)$$

and furthermore that

$$z \leq f(\vec{y}_m, z) \quad \textit{and} \quad (f(\vec{y}_m, z))^2 \leq f(\vec{y}_m, z+1).$$

If the relations $g(\vec{y}_m) = x$ *and* $h(x, \vec{y}_m, z) = y$ *are both* PRUD, *then so is* $f(\vec{y}_m, z) = y$.

The proof is found in [14].

5 We proceed to sketch how ELP computations can be arithmetized using PRUD relations. (The detailed treatment is in Appendix 2.) The first step is to code ELPs as finite sequences of instructions, each of the instructions being one of the five aforementioned types and involving one or two program variables. Thus, e.g., the instructions $V_i \leftarrow 0$ and $V_i \leftarrow V_i + 1$ will be represented by the ordered pairs $(1, i)$ and $(2, i)$, respectively, and $V_j \leftarrow V_i$ by the ordered triple $(3, i, j)$. The description of LOOP instructions includes one extra element: an instruction LOOP V_i that is a part of a loop segment of length m is represented by the ordered triple $(4, m, i)$, and the end instruction END of that segment is represented by the ordered pair $(5, m)$. Then if an $\mathbf{L}_0$ or $\langle \mathbf{L}_0 \rangle \mathbf{L}_0$ program $\mathbf{P}$ of length p consists of the instructions $I_1, \ldots, I_p$, $(p \geq 1)$, we can numerically represent it by means of the indexed series,

$$(*) \qquad (1, \#I_1), (2, \#I_2), \ldots, (p, \#I_p),$$

where $\#I_j$, for $1 \leq j \leq p$, codes the ordered pair or triple representing I_j. Smullyan showed that indexed series of finite length, as well as pairs and triples, can be coded using only rudimentary relations. We let the Gödel number $\#\mathbf{P}$ of the program $\mathbf{P}$ be the code of $(*)$.

To code $\mathbf{L}_1$ and $\mathbf{L}_2$ programs we employ a somewhat more complex device. An $\mathbf{L}_1^k$, $k \geq 1$, program $\mathbf{P}$ has the form

$$\mathbf{L}_0(\underbrace{\langle \mathbf{L}_0 \rangle \mathbf{L}_0) \ldots (\langle \mathbf{L}_0 \rangle}_{k \text{ times}} \mathbf{L}_0)$$

We represent it by an indexed series

$$(**) \qquad (0, \#\mathbf{P}_0), (1, \#\mathbf{P}_1), \ldots, (k, \#\mathbf{P}_k),$$

where $\#\mathbf{P}_0$ is the Gödel number of an $\mathbf{L}_0$ program and each $\#\mathbf{P}_i$, $1 \leq i \leq k$, is the Gödel number of an $\langle \mathbf{L}_0 \rangle \mathbf{L}_0$ program. (We call the latter the *components* of $\mathbf{P}$.) Assuming that the Gödel numbering of $\mathbf{L}_1$ programs is determined in this way, we may use the same idea to code $\mathbf{L}_2^k$, $k \geq 1$, programs by indexed series such as $(**)$ in which $\#\mathbf{P}_0$ and $\#\mathbf{P}_i$, $1 \leq i \leq k$, are Gödel numbers of the component $\mathbf{L}_1$ and $\langle \mathbf{L}_1 \rangle \mathbf{L}_1$ programs, respectively. Thus, we may think of $\mathbf{L}_2^k$ programs as represented by trees of height 2: the top node is the code of an indexed series of the sort just described, its immediate descendants are the Gödel number of an $\mathbf{L}_1$ program and the Gödel numbers of k many $\langle \mathbf{L}_1 \rangle \mathbf{L}_1$ programs, respectively, and the immediate descendants of the latter and the endpoints of the tree are their $\mathbf{L}_0$ and $\langle \mathbf{L}_0 \rangle \mathbf{L}_0$ components. (This is carried out in Appendix 1.)

The next step is numerically to represent the course of a computation that proceeds according to a given $\mathbf{L}_2^k$ program $\mathbf{P}$. Here we need first to be able to represent a state of the register machine that executes $\mathbf{P}$. Such a *machine state* is characterized by a particular assignment of numerical values to the variables of $\mathbf{P}$, which we represent by the indexed series,

$$(***) \qquad (0, v_0), \ldots, (s, v_s),$$

assuming that all the variables in $\mathbf{P}$ come from $\underline{V_s} : v_i (0 \leq i \leq s)$ is the numerical value of the variable V_i, i.e., the contents of the register i at some given time t. (We shall assume that V_0 is always the output variable of $\mathbf{P}$, that the input variables of $\mathbf{P}$ are always even-numbered, i.e., of the form $V_{2i}, 0 \leq i \leq j$ for some j, and that the local variables of $\mathbf{P}$ are always odd-numbered.) To represent a particular stage of computation according to an $\mathbf{L}_2^k$ program $\mathbf{P}$ we indicate what the state of the machine that executes the program is at that point and which one of the instructions in $\mathbf{P}$ is about to be performed. This we do with an ordered pair (i, σ), where σ codes some machine state characterized by an indexed series such as $(***)$, and i codes an ordered triple (j, q, m) of integers ($j, q \geq 0$ and $m \geq 1$). The triple indicates that the instruction about to be executed is part of the jth $(0 < j \leq k)\langle \mathbf{L}_1 \rangle \mathbf{L}_1$—$\mathbf{L}_1$ if $j = 0$—component $\mathbf{P}_j$ of $\mathbf{P}$, namely that it is the mth instruction in the qth $\langle \mathbf{L}_0 \rangle \mathbf{L}_0$—$\mathbf{L}_0$ if $q = 0$—component $\mathbf{P}_{j,q}$ of $\mathbf{P}_j$.

The pair (i, σ) is an *instantaneous description* (i.d.) of a register machine with at most $s + 1$ registers computing an $\mathbf{L}_2^k$ program. We can then completely describe the sequence of steps in a computation according to $\mathbf{P}$ as a sequence of i.d.s with the following property: in each noninitial term $((j_{p+1}, q_{p+1}, m_{p+1}), \sigma_{p+1})$ of the sequence, σ_{p+1} codes the machine state that results from σ_p after the m_p-th instruction in $\mathbf{P}_{j_p, q_p}$ is executed once, the m_{p+1}-th $(= (m_p + 1)$-st) instruction in $\mathbf{P}_{j_{p+1}, q_{p+1}}$, if it exists, being the next one to be performed, if any. (Then $j_{p+1} = j_p$ and $q_{p+1} = q_p$. In case $\mathbf{P}_{j_p, q_p}$ has only m_p instructions, then $q_{p+1} = q_p + 1$ and $m_{p+1} = 1$; if there are only q_p many $\langle \mathbf{L}_0 \rangle \mathbf{L}_0$ components of $\mathbf{P}_{j_p}$, then $j_{p+1} = j_p + 1$ and $q_{p+1} = 0$ and $m_{p+1} = 1$.) Since LOOP program computations always terminate, the sequence will always be finite and its last term of the form $((k + 1, 0, 1), \sigma^*)$. Then σ^* codes the state the register machine finally assumes; we say that such an i.d. is a *final* i.d. for a program coded by an indexed series of length $k + 1$.

What remains now is to express the value of the function computed by a given program $\mathbf{P}$ for arguments $\vec{x}_n$ as the output of the computation started on $\vec{x}_n$ as input. The output will simply be the value of the output variable V_0 in the resulting final i.d. But to express this value as a function of $\vec{x}_n$, we have to show how the Gödel number for the final i.d. can be determined in terms of $\vec{x}_n$ and the Gödel number of the program $\mathbf{P}$. This we do by precisely describing how, starting with an i.d. that characterizes the initial state of the machine—in which the input variables $V_2, \ldots, V_{2n}$ have values $x_1, \ldots, x_n$, respectively, and all other program variables in $\mathbf{P}$ have value 0: each nonfinal i.d. yields a subsequent one in accordance with the program $\mathbf{P}$.

Such a description is given by the sequence

$$z_0, z_1, \ldots, z_i$$

where z_0 is the i.d. corresponding to the initial state of the machine with input $\vec{x}_n$, each $z_j (0 \leq j < i)$ the (uniquely determined) i.d. at the jth step of the computation of **P** with input $\vec{x}_n$, and z_i the final i.d. (Then i is the *number of steps* in the computation of **P** for input $\vec{x}_n$.) Instead, we consider the sequence,

$$(\dagger) \qquad J^*(1, z_0),\ J^*(z_0, z_1), \ldots,\ J^*(z_{i-1}, z_i),$$

where the function $J^*(x, y) := (x + y)^2 + x$ determines a coding of $\mathbb{N} \times \mathbb{N}$ by a subset of $\mathbb{N}$. We need to be able to express (†) as the sequence of successive values of a function H with a RUD graph, and it is for this that we use Theorem 4.2. In terms of the graph of the function H we define the RUD predicate,

$$T_n(k, e^*, \vec{x}_n, i, u)\ \&\ \mathrm{OUTPUT}(v, y),$$

which holds just in case i is the number of steps in the computation represented in a sequence such as (†) that ends with u, y is the value of the output variable in the resulting final i.d. v, and e^* is the Gödel number of the$\mathbf{L}_2^k$ program **P**. Then the value of the function computed by **P** for input $\vec{x}_n$ can be expressed in the form

$$\mu y[\exists i \exists u(T_n(k, e^*, \vec{x}_n, i, u)\ \&\ \mathrm{OUTPUT}(L^*(u), y))] \tag{4}$$

(K^* and L^* are projection functions associated with J^*, so that $J^*(K^*(x), L^*(x)) = x$ for any integer x in the range of J^*.)

6 For our purposes, it is important to find an upper bound for μy and $\exists i \exists u$ in (4) in terms of e^* and $\vec{x}_n$. If $T_n(k, e^*, \vec{x}_n, i, u)$, then $u = z_i$, where z_i is as in (†). The values of z_m, $0 \leq m \leq i$, are the Gödel numbers of i.d.s of the form

$$((j, q, m), (v_0, \ldots, v_s)^{\#})$$

where $(v_0, \ldots, v_s)^{\#}$ is the *sequence number*, i.e., the code, of the sequence $(v_0, \ldots, v_s)$. (We are referring to the coding scheme for sequences used in [18], p. 82, which is due to Quine.) Furthermore, if $\mathbf{P} \in \mathbf{L}_2^k$, then $v_u \leq f_2^k \circ f_1^{\mathrm{lth}(\mathbf{P})} \circ f_0^{2\,\mathrm{lth}(\mathbf{P})}(\max(\vec{x}_n)) = v$ for $0 \leq u \leq s$, by Theorem 3.2. We first look for an appropriate upper bound for $(v_0, \ldots, v_s)^{\#}$.

Lemma 6.1 *Suppose that $n \geq 2$.*

1. *If $a_i = a$ for each i, $1 \leq i \leq n$, then*

$$a_1 * \ldots * a_n = a(2^{(n-1)l(a)} + 2^{(n-2)l(a)} + \cdots + 2^{l(a)} + 1).$$

2. *If $y > 0$ and $a_i \leq y$ for each i, $1 \leq i \leq n$, then*

$$(a_1, \ldots, a_n)^{\#} < (2^6 \cdot y)^{n(2n+1)+1}.$$

The proof is in Appendix 3.

We have been assuming so far that a Gödel numbering of ELPs has been set up. (This is done in Appendix 1.) It is convenient, however, to introduce an *indexing* of $\mathbf{L}_2^k$ programs that makes explicit the number of instructions, lth(**P**), in a program **P**,

and the number n of variables in $\mathbf{P}$. Let $J^*(e^*, \operatorname{lth}(\mathbf{P}), n)$ be an *index* of $\mathbf{P}$ with Gödel number e^*. (Thus, $n = L^*L^*(e)$, where e is the index of $\mathbf{P}$.) From Lemma 6.1(2) we have that

$$(v_0, \ldots, v_s)^{\#} < (2^6 \cdot v)^{L*L*(e)\cdot(2L*L*(e)+1)+1}$$

Since in general

$$J^*(K^*(x), L^*(x)) = (K^*(x) + L^*(x))^2 + K^*(x),$$

it follows that

$$L^*L^*(e) \cdot (2L^*L^*(e) + 1) + 1 = 2(L^*L^*(e))^2 + L^*L^*(e) + 1 \leq 2L^*(e) + L^*(e) + 1.$$

Hence $(v_0, \ldots, v_s)^{\#} < (2^6 \cdot v)^{3L^*(e)+1}$, and if $k \geq 2$,

$$\begin{aligned}
(v_0, \ldots, v_s)^{\#} &< [2^6(f_2^k \circ f_1^{\operatorname{lth}(\mathbf{P})} \circ f_0^{2\operatorname{lth}(\mathbf{P})}(\max(\vec{x}_n)))]^{3L^*(e)+1} = \\
&= [f_1^6 \circ f_2^k \circ f_1^{\operatorname{lth}(\mathbf{P})} \circ f_0^{2\operatorname{lth}(\mathbf{P})}(\max(\vec{x}_n))]^{3L^*(e)+1} \leq \\
&\leq [f_2^k \circ f_0^6 \circ f_1^{\operatorname{lth}(\mathbf{P})} \circ f_0^{2\operatorname{lth}(\mathbf{P})}(\max(\vec{x}_n))]^{3L^*(e)+1} \leq \\
&\leq [f_2^k \circ f_1^{\operatorname{lth}(\mathbf{P})}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + 12)]^{3L^*(e)+1} \leq \\
&\leq f_2^k \circ f_1(f_1^{\operatorname{lth}(\mathbf{P})}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + 12) + 3L^*(e) + 1) \leq \\
&\leq f_2^k \circ f_1 \circ f_0^{2L^*(e)}(f_1^{\operatorname{lth}(\mathbf{P})}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + 12)) \leq \\
&\leq f_2^k \circ f_1 \circ f_1^{\operatorname{lth}(\mathbf{P})}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + 12 + 4L^*(e)) = \\
&= f_2^k \circ f_1^{\operatorname{lth}(\mathbf{P})+1}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + 4L^*(e) + 12) \leq \\
&\leq f_2^k \circ f_1^{\operatorname{lth}(\mathbf{P})+1}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + e + 12),
\end{aligned}$$

given that $K^*(e) \geq 2$ and $L^*(e) \geq 1$.

To obtain an upper bound for the values of z_m, note that assuming $k \geq 2$ we have that, for some integer $p \leq e$ that codes an ordered triple (j, q, m),

$$\begin{aligned}
J(p, (v_0, \ldots, v_s)^{\#}) &\leq J(e, f_2^k \circ f_1^{\operatorname{lth}(\mathbf{P})+1}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + e + 12)) \leq \\
&\leq [2^6(f_2^k \circ f_1^{\operatorname{lth}(\mathbf{P})+1}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + e + 12))]^{11} = \\
&= [f_1^6 \circ f_2^k \circ f_1^{\operatorname{lth}(\mathbf{P})+1}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + e + 12)]^{11} \leq \\
&\leq [f_2^k \circ f_0^6 \circ f_1^{\operatorname{lth}(\mathbf{P})+1}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + e + 12)]^{11} \leq \\
&\leq [f_2^k \circ f_1^{\operatorname{lth}(\mathbf{P})+1}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + e + 24)]^{11} \leq \\
&\leq f_2^k \circ f_1(f_1^{\operatorname{lth}(\mathbf{P})+1}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + e + 24) + 11) \leq \\
&\leq f_2^k \circ f_1 \circ f_0^6 \circ f_1^{\operatorname{lth}(\mathbf{P})+1}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + e + 24) \leq \\
&\leq f_2^k \circ f_1^{\operatorname{lth}(\mathbf{P})+2}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + e + 36).
\end{aligned}$$

Thus each term of the sequence (†) has an upper bound:

$$\begin{aligned}
J^*(f_2^k \circ f_1^{\operatorname{lth}(\mathbf{P})+2}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + e + 36), & \\
f_2^k \circ f_1^{\operatorname{lth}(\mathbf{P})+2}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + e + 36)) &\leq \\
\leq 5[f_2^k \circ f_1^{\operatorname{lth}(\mathbf{P})+2}(\max(\vec{x}_n) + 4\operatorname{lth}(\mathbf{P}) + e + 36)]^2 &\leq
\end{aligned}$$

$$\begin{aligned}
&\leq 5[f_2^k \circ f_0 \circ f_1^{\mathrm{lth}(\mathbf{P})+2}(\max(\vec{x}_n) + 4\,\mathrm{lth}(\mathbf{P}) + e + 36)] \leq \\
&\leq 5[f_2^k \circ f_1^{\mathrm{lth}(\mathbf{P})+2}(\max(\vec{x}_n) + 4\,\mathrm{lth}(\mathbf{P}) + e + 38)] \leq \\
&\leq f_1^3 \circ f_2^k \circ f_1^{\mathrm{lth}(\mathbf{P})+2}(\max(\vec{x}_n) + 4\,\mathrm{lth}(\mathbf{P}) + e + 38) \leq \\
&\leq f_2^k \circ f_0^3 \circ f_1^{\mathrm{lth}(\mathbf{P})+2}(\max(\vec{x}_n) + 4\,\mathrm{lth}(\mathbf{P}) + e + 38) \leq \\
&\leq f_2^k \circ f_1^{\mathrm{lth}(\mathbf{P})+2}(\max(\vec{x}_n) + 4\,\mathrm{lth}(\mathbf{P}) + e + 44) \leq \\
&\leq f_2^k \circ f_1^{\mathrm{lth}(\mathbf{P})+2} \circ f_0^{2\,\mathrm{lth}(\mathbf{P})+22}(\max(\vec{x}_n) + e) \leq \\
&\leq f_2^k \circ f_1^{\mathrm{lth}(\mathbf{P})+2} \circ f_0^{2\,\mathrm{lth}(\mathbf{P})+22}(\Sigma_{i=1}^n x_i + e).
\end{aligned}$$

(Here we note that $J^*(x, x) = (2x)^2 + x \leq 5x^2$.) An upper bound for the values of H suffices as an upper bound for i as well, since

$$i \leq f_2^k \circ f_1^{\mathrm{lth}(\mathbf{P})} \circ f_0^{2\,\mathrm{lth}(\mathbf{P})}(\max(\vec{x}_n)) \leq f_2^k \circ f_1^{\mathrm{lth}(\mathbf{P})}(\max(\vec{x}_n) + 4\,\mathrm{lth}(\mathbf{P}))$$

by Theorem 3.2.

The value of the output variable of $\mathbf{P}$ at the end of the computation that started with input $\vec{x}_n$ is $\leq f_2^k \circ f_1^{\mathrm{lth}(\mathbf{P})} \circ f_0^{2\,\mathrm{lth}(\mathbf{P})}(\max(\vec{x}_n))$, as noted earlier. Hence the upper bound for i and u in (4) just obtained also serves as an upper bound for μy. Let

$$\psi_n^{k,j}(u, \vec{x}_n) := f_2^k \circ f_1^{j+2} \circ f_0^{2j+22}(\Sigma_{i=1}^n x_i + u)$$

For each $k, j, n \geq 1$, $\psi_n^{k,j} \in \mathcal{L}_2^k$. (Cf. Theorem 3.3 and [8], pp. 307–308.) And we have the following result.

Theorem 6.2 *Suppose $n \geq 1$ and $k \geq 2$. For any n-place function $f \in \mathcal{L}_2^k$,*

$f(\vec{x}_n) =$

$(\mu y < \psi_n^{k,j}(e, \vec{x}_n))[\exists i, u \leq \psi_n^{k,j}(e, \vec{x}_n)(T_n(k, e, \vec{x}_n, i, u)\ \&\ \mathrm{OUTPUT}(L^*(u), y))]$

for some integers e and j.

7 For a given register machine (RM) program $\mathbf{P}$ with registers $X_1, \ldots, X_n, Z_1, \ldots, Z_m$, we let $\mathrm{In}(X_i)$ and $\mathrm{In}(Z_j)$, for $1 \leq i \leq n$ and $1 \leq j \leq m$, be the initial contents of X_i and Z_j, i.e., before the computation begins. We let $\mathrm{Out}(X_i)$ and $\mathrm{Out}(Z_j)$ be the contents of the registers X_i and Z_j at the end of the computation if the computation terminates. We say that $\mathbf{P}$ *computes* a predicate $\mathcal{A}(\vec{x}_n)$ *in the standard format* just in case for any integers $m_1, \ldots, m_n$, if $\mathrm{In}(X_i) = m_i$ and $\mathrm{In}(Z_j) = 0$ for all i, $j (1 \leq i \leq n$ and $1 \leq j \leq m)$ and the computation eventually terminates, then $\mathrm{Out}(X_i) = \mathrm{In}(X_i)$,

$$\begin{aligned}
\mathrm{Out}(Z_1) = 0 &\quad \Leftrightarrow \quad \mathcal{A}(\underline{\vec{m}}_n) \text{ is true} \\
\mathrm{Out}(Z_1) = 1 &\quad \Leftrightarrow \quad \mathcal{A}(\underline{\vec{m}}_n) \text{ is false}
\end{aligned}$$

and $\mathrm{Out}(Z_j) = 0$ for all $j, 1 < j \leq m$.

Let $T_{\mathbf{P}}(\vec{x}_n)$ be the number of steps in the computation of $\mathbf{P}$ that begins with $\mathrm{In}(X_i) = x_i$ and $\mathrm{In}(Z_j) = 0$ for all i, $j (1 \leq i \leq n$ and $1 \leq j \leq m)$. The function $T_{\mathbf{P}}$ is possibly partial. Let $\vec{m}_n^*$ be the sequence of integers that results when the ith term, m_i, of $\vec{m}_n$ is replaced by m_i^*. Suppose that in general if $m_i \leq m_i^*$, then $T_{\mathbf{P}}(\vec{m}_n^*)\downarrow$ if $T_{\mathbf{P}}(\vec{m}_n)\downarrow$ and furthermore $T_{\mathbf{P}}(\vec{m}_n) \leq T_{\mathbf{P}}(\vec{m}_n^*)$. In that case we say that the program $\mathbf{P}$ is *regular*.

Lemma 7.1 *There are regular* RM *programs* $\mathrm{ID}(X_i, X_j)$ *and* $\mathrm{LE}(X_i, X_j)$ *that compute* $x_i = x_j$ *and* $x_i \leq x_j$*, respectively, in the standard format, and for any* x_1, x_2*,* $T_{\mathrm{ID}}(x_1, x_2)\downarrow$ *and* $T_{\mathrm{LE}}(x_1, x_2)\downarrow$ *and*

$$T_{\mathrm{ID}}(x_1, x_2) \leq 6x_1 + 8x_2 + 9 \quad \textit{and} \quad T_{\mathrm{LE}}(x_1, x_2) \leq 7x_1 + 6x_2 + 6.$$

Such programs are easily constructed.

Theorem 7.2 *Let* $\mathcal{A}(\vec{x}_n)$ *be a* CA *predicate. Then there is a regular* RM *program* **P** *that computes* $\mathcal{A}(\vec{x}_n)$ *in the standard format and there are integers* p, q*, depending on the logical complexity of* $\mathcal{A}(\vec{x}_n)$*, such that for any* $\vec{x}_n$*,* $T_{\mathbf{P}}(\vec{x}_n)\downarrow$ *and*

$$T_{\mathbf{P}}(\vec{x}_n) \leq (f_2 \circ f_1^p \circ f_0^q)(\max(\vec{x}_n)).$$

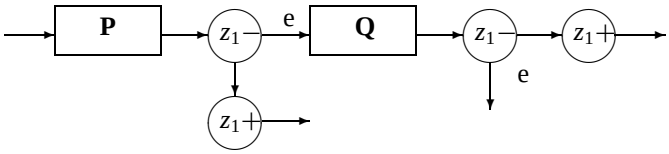
Proof: The argument is by induction on the complexity of the predicate $\mathcal{A}(\vec{x}_n)$.

If $\mathcal{A}(\vec{x}_n)$ is atomic, it is of the form $x_j + x_k = x_i$ or $x_j \cdot x_k = x_i$, or else is obtained by explicit transformation from such a formula. Now, there are regular RM programs ID^+ and $\mathrm{ID}^\times$ that compute $x_j + x_k = x_i$ and $x_j \cdot x_k = x_i$ in the standard format, respectively, and for any x_1, x_2, x_3, $T_{\mathrm{ID}^+}(x_1, x_2, x_3)\downarrow$ and $T_{\mathrm{ID}^\times}(x_1, x_2, x_3)\downarrow$ and $T_{\mathrm{ID}^+}(x_1, x_2, x_3) \leq f_1^6 \circ f_0^8(\max(\vec{x}_n))$ and

$$T_{\mathrm{ID}^\times}(x_1, x_2, x_3) \leq (f_2 \circ f_1^1 \circ f_0^3)(\max(x_1, x_2, x_3)).$$

(We leave the construction of such programs to the reader.)

Assume now, as the induction hypothesis, that the Theorem holds for CA predicates $\mathcal{B}(\vec{x}_n)$ and $C(\vec{x}_m)$ and that regular RM programs **P** and **Q** compute $\mathcal{B}(\vec{x}_n)$ and $C(\vec{x}_m)$, respectively, as stated in the Theorem. Let $\vec{y}_k (k \leq m)$ be the free variables of $C(\vec{x}_m)$ that are not among $\vec{x}_n$. Then the following is a regular RM program **P**&**Q** that computes $(\mathcal{B}\&C)(\vec{x}_n, \vec{y}_k)$ in the standard format such that $T_{\mathbf{P\&Q}}(\vec{x}_n, \vec{y}_k)\downarrow$ for any $\vec{x}_n, \vec{y}_k$, and $T_{\mathbf{P\&Q}}(\vec{x}_n, \vec{y}_k) \leq T_{\mathbf{P}}(\vec{x}_n) + T_{\mathbf{Q}}(\vec{x}_n, \vec{y}_k) + 3$.



By the induction hypothesis, $T_{\mathbf{P}}(\vec{x}_n)\downarrow$ and $T_{\mathbf{Q}}(\vec{x}_n, \vec{y}_k)\downarrow$ for any $\vec{x}_n, \vec{y}_k$. In addition

$$T_{\mathbf{P}}(\vec{x}_n) \leq (f_2 \circ f_1^{p_1} \circ f_0^{q_1})(\max(\vec{x}_n))$$

and

$$T_{\mathbf{Q}}(\vec{x}_n, \vec{y}_k) \leq (f_2 \circ f_1^{p_2} \circ f_0^{q_2})(\max(\vec{x}_n, \vec{y}_k))$$

for some p_1, q_1, p_2, q_2. Then,

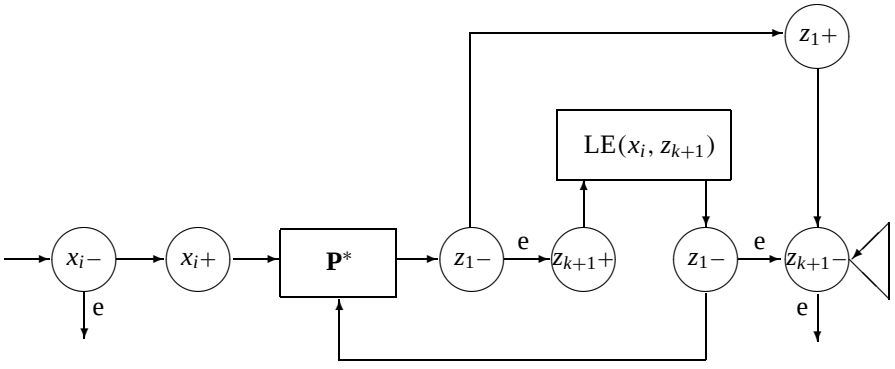
$$\begin{aligned} T_{\mathbf{P\&Q}}(\vec{x}_n, \vec{y}_k) &\leq (f_2 \circ f_1^{p_1} \circ f_0^{q_1})(\max(\vec{x}_n)) + \\ &\quad + (f_2 \circ f_1^{p_2} \circ f_0^{q_2})(\max(\vec{x}_n, \vec{y}_k)) + 3 \leq \\ &\leq \; 2(f_2 \circ f_1^{\max(p_1, p_2)} \circ f_0^{\max(q_1, q_2)})(\max(\vec{x}_n, \vec{y}_k)) + 3 \leq \end{aligned}$$

$$\begin{aligned}
&\leq (f_0^2 \circ f_1^1 \circ f_2^1 \circ f_1^{\max(p_1,p_2)} \circ f_0^{\max(q_1,q_2)})(\max(\vec{x}_n, \vec{y}_k)) \leq \\
&\leq (f_0^2 \circ f_2^1 \circ f_0^1 \circ f_1^{\max(p_1,p_2)} \circ f_0^{\max(q_1,q_2)})(\max(\vec{x}_n, \vec{y}_k)) \leq \\
&\leq (f_2^1 \circ f_0^3 \circ f_1^{\max(p_1,p_2)} \circ f_0^{\max(q_1,q_2)})(\max(\vec{x}_n, \vec{y}_k)) \leq \\
&\leq (f_2 \circ f_1^{\max(p_1,p_2)} \circ f_0^{\max(q_1,q_2)+3})(\max(\vec{x}_n, \vec{y}_k)).
\end{aligned}$$

A similar but simpler argument shows that if a regular RM program $\mathbf{P}$ computes a CA predicate $\mathcal{B}(\vec{x}_n)$ as described in the Theorem, then a regular RM program $\neg\mathbf{P}$ can be found that computes $\neg\mathcal{B}(\vec{x}_n)$ in the standard format such that, for all $\vec{x}_n$, $T_{\neg\mathbf{P}}(\vec{x}_n)\downarrow$ if $T_{\mathbf{P}}(\vec{x}_n)\downarrow$, and $T_{\neg\mathbf{P}}(\vec{x}_n) \leq T_{\mathbf{P}}(\vec{x}_n) + 2$. Then there are integers p, q such that

$$T_{\mathbf{P}}(\vec{x}_n) \leq (f_2 \circ f_1^p \circ f_0^q)(\max(\vec{x}_n)) \Rightarrow T_{\neg\mathbf{P}}(\vec{x}_n) \leq (f_2 \circ f_1^p \circ f_0^{q+1})(\max(\vec{x}_n)).$$

Suppose now that a regular RM program $\mathbf{P}$ computes $\mathcal{B}(\vec{x}_n)$ in the standard format, and assume that the local variables in $\mathbf{P}$ are all among $Z_1, \ldots, Z_k$. Let $\mathbf{P}^*$ be the RM program that results when the input variable X_1 in $\mathbf{P}$ is replaced by Z_{k+1}. Then the following RM program, $\forall z < x\mathbf{P}$, computes $(\forall z < x_1)\mathcal{B}(\vec{x}_n)$ in the standard format:



Then, for all $\vec{x}_n$, $T_{\forall z<x\mathbf{P}}(\vec{x}_n)\downarrow$ if $T_{\mathbf{P}}(\vec{x}_n)\downarrow$. Furthermore, we have that,

$$\begin{aligned}
T_{\forall z<x\mathbf{P}}(\vec{x}_n) &\leq 2 + \Sigma_{z=0}^{x_1}(T_{\mathbf{P}}(z, x_2, \ldots, x_n) + 2 + T_{\mathrm{LE}}(x_1, z) + 1) + (x_1 + 1) = \\
&= x_1 + 3 + \Sigma_{z=0}^{x_1}(T_{\mathbf{P}}(z, x_2, \ldots, x_n) + (7x_1 + 6z + 6) + 3) = \\
&= (x_1 + 3) + (x_1 + 1)(7x_1 + 9) + \Sigma_{z=0}^{x_1}(T_{\mathbf{P}}(z, x_2, \ldots, x_n) + 6z) = \\
&= 2 + (x_1 + 1)(7x_1 + 10) + 6 \cdot \Sigma_{z=0}^{x_1} z + \Sigma_{z=0}^{x_1} T_{\mathbf{P}}(z, x_2, \ldots, x_n) = \\
&= 2 + (x_1 + 1)(7x_1 + 10) + 3x_1(x_1 + 1) + \Sigma_{z=0}^{x_1} T_{\mathbf{P}}(z, x_2, \ldots, x_n) = \\
&= 2 + 10(x_1 + 1)^2 + \Sigma_{z=0}^{x_1} T_{\mathbf{P}}(z, x_2, \ldots, x_n) \leq \\
&\leq 2 + 10(2^{x_1+1} + 1) + \Sigma_{z=0}^{x_1} T_{\mathbf{P}}(z, x_2, \ldots, x_n) \leq \\
&\leq 2^4 + 2^{x_1+5} + \Sigma_{z=0}^{x_1} T_{\mathbf{P}}(z, x_2, \ldots, x_n) \leq \\
&\leq 2^{x_1+6} + \Sigma_{z=0}^{x_1} T_{\mathbf{P}}(z, x_2, \ldots, x_n).
\end{aligned}$$

By the induction hypothesis, $T_{\mathbf{P}}(\vec{x}_n) \leq (f_2 \circ f_1^p \circ f_0^q)(\max(\vec{x}_n))$ for some p, q, and the program $\mathbf{P}$ is regular. Hence for each $z \leq x_1$, $T_{\mathbf{P}}(z, x_2, \ldots, x_n) \leq T_{\mathbf{P}}(\vec{x}_n)$, and so

$$\Sigma_{z=0}^{x_1} T_{\mathbf{P}}(z, x_2, \ldots, x_n) \leq x_1 \cdot T_{\mathbf{P}}(\vec{x}_n) \leq 2^{x_1} \cdot T_{\mathbf{P}}(\vec{x}_n) = 2^{x_1 + g(\vec{x}_n)},$$

where $g(\vec{x}_n) = f_1^p \circ f_0^q(\max(\vec{x}_n))$.

Then $T_{\forall z<x\mathbf{P}}(\vec{x}_n) \leq 2^{x_1+6} + 2^{x_1+g(\vec{x}_n)} \leq 2^{g(\vec{x}_n)+2x_1+6}$. Now we have that,

$$\begin{aligned} g(\vec{x}_n) + 2x_1 + 6 &= f_1^p \circ f_0^q(\max(\vec{x}_n)) + 2x_1 + 6 = \\ &= 2(f_1^{p-1} \circ f_0^q(\max(\vec{x}_n)) + x_1 + 3) \leq \\ &\leq 2(f_1^{p-1} \circ f_0^q(\max(\vec{x}_n)) + f_0^2(x_1)) \leq \\ &\leq 2(f_1^p \circ f_0^{max(q,2)}(\max(\vec{x}_n)) = \\ &= f_1^{p+1} \circ f_0^{max(q,2)}(\max(\vec{x}_n)). \end{aligned}$$

But then $T_{\forall z<x\mathbf{P}}(\vec{x}_n) \leq f_2 \circ f_1^{p+1} \circ f_0^{\max(q,2)}(\max(\vec{x}_n))$. The program $\forall z < x\mathbf{P}$ is easily seen to be regular.

A slightly different construction yields a regular program $\exists z < x\mathbf{P}$ that computes $(\exists z < x_1)\mathcal{B}(z, x_2, \ldots, x_n)$ in the standard format with the same estimate of running time. This completes the induction on the complexity of $\mathcal{A}(\vec{x}_n)$. Then the Theorem is easily seen to extend to the predicates obtained by explicit transformation from CA predicates considered in the induction. This completes the proof of Theorem 7.2.

The estimate of the running time given in Theorem 7.2 will help us place the CA predicates in the hierarchy $\mathcal{L}_2^k$. It turns out that computations of RM programs can be "simulated" using a sufficient number of computations of appropriate loop programs of very low complexity, with depth of nesting not exceeding 1. This is a corollary of the following result of Meyer and Ritchie [13].

Theorem 7.3 *For any* RM*-program* $\mathbf{P}$ *there is an* $\mathbf{L}_1$ *program* $\mathbf{P}^*$ *such that the* LOOP$_\mu$ *program,*

$H \leftarrow 1$
WHILE $H \neq 0$ DO
  $\mathbf{P}^*$
END

is equivalent to $\mathbf{P}$.

(Here H is a local variable occurring in $\mathbf{P}^*$.)

The corollary that interests us is the following.

Theorem 7.4 *Let* $\mathbf{P}$ *be an* RM *program such that* $T_{\mathbf{P}}(\vec{x}_n) \leq f(\vec{x}_n)$. *Then there is an* $\mathbf{L}_1$ *program* $\mathbf{P}^*$ *such that, for any* $k \geq 2$, *given any* $\mathbf{L}_2^k$ *program* $\mathbf{B}_f$ *computing* f *(if there is one) an* $\mathbf{L}_2^{k+1}$ *program* $L(\mathbf{P})$ *of the form*

$\mathbf{B}_f$
$S \leftarrow F$
$H \leftarrow 1$
LOOP S
  $\mathbf{P}^*$
END

can be obtained that is equivalent to $\mathbf{P}$.

(Here, F is the output variable of $\mathbf{B}_f$ and H is a local variable appearing in $\mathbf{P}^*$.) For a detailed proof of both results, see [5], §4.2. Note that if $\mathbf{B}_f$ is an $\mathbf{L}_2^k$ program ($k \geq 1$), then $L(\mathbf{P})$ belongs to

$$\mathbf{L}_2^k\mathbf{L}_0\langle\mathbf{L}_1\rangle = \mathbf{L}_2^{k-1}(\langle\mathbf{L}_1\rangle\mathbf{L}_1)\mathbf{L}_0\langle\mathbf{L}_1\rangle \subseteq \mathbf{L}_2^{k-1}\langle\mathbf{L}_1\rangle\mathbf{L}_1\langle\mathbf{L}_1\rangle \subseteq \mathbf{L}_2^k\langle\mathbf{L}_1\rangle \subseteq \mathbf{L}_2^{k+1}.$$

The program $L(\mathbf{P})$ repeats $\mathbf{P}^*$ sufficiently many times ensuring at the same time that a larger-than-necessary number of repetitions does not damage the simulation. (Cf. [5], pp. 546-547.) From Theorems 7.2, 7.4 and 3.3 we therefore have the following.

Theorem 7.5 *For any* CA *predicate* f, $f \in \mathcal{L}_2^2$.

8 We now use the ideas behind Theorem 7.4 together with other results established so far to prove the following enumeration theorem for the functions in the hierarchy $\{\mathcal{L}_2^k \mid k \geq 2\}$. Let $\mathcal{L}_2^{k,j}$ be the class of functions in $\mathcal{L}_2^k$ computable by some LOOP program of length $\leq j$.

Theorem 8.1 *Let $k \geq 2$ and $n \geq 1$. For any $j \geq 1$ there is a function $\Phi_n^{k,j} \in \mathcal{L}_2^{k+1}$, uniformly depending on k and j, such that for any n-place function $g \in \mathcal{L}_2^{k,j}$,*

$$g(\vec{x}_n) = \Phi_n^{k,j}(e, \vec{x}_n)$$

for some integer e, an index of g.

Proof: Let

$$\Psi_n(k, b, \vec{x}_n, v) := (\mu y < v)[\exists i, u \leq v(T_n(k, b, \vec{x}_n, i, u) \,\&\, \mathrm{OUTPUT}\,(L^*(u), y))]$$

for any $n \geq 1$. Then, by Theorem 6.2, for any $k \geq 2$, $n \geq 1$,

$$g(\vec{x}_n) = \Psi_n(k, e, \vec{x}_n, \psi_n^{k,j}(e, \vec{x}_n))$$

for some e and j, where $\psi_n^{k,j} \in \mathcal{L}_2^k$ is as described in §6. (Recall from the proof of Theorem 6.2 that the constant j is given as $\geq \mathrm{lth}(\mathbf{P})$ for some $\mathbf{L}_2^k$ program $\mathbf{P}$ with index e that computes g.) The function Ψ_n has a RUD graph; moreover,

$$g(\vec{x}_n) = y \Leftrightarrow \underline{\Psi}_n(k, e, \vec{x}_n, \psi_n^{k,j}(e, \vec{x}_n), y) = 0$$

where $\underline{\Psi}_n$ is the RUD predicate expressing the characteristic function of the RUD graph of Ψ_n. By the results of §7 and Theorem 3.3, $\underline{\Psi}_n$ is computable by an $\mathbf{L}_2^1\langle\mathbf{L}_1\rangle$ program $L(\mathbf{P})$ of the form,

$\mathbf{B}_f$
$S \leftarrow F$
$H \leftarrow 1$
LOOP S
  $\mathbf{P}^*$
END

where $\mathbf{P}^* \in \mathbf{L}_1$, and $\mathbf{B}_f$ computes $f_2 \circ f_1^{p_1} \circ f_0^{q_1}(\max(\vec{x}_n))$ for some p_1, q_1, and OUT F for $\mathbf{B}_f$. From the construction of $L(\mathbf{P})$ it follows that replacing $\mathbf{B}_f$ by an $\mathbf{L}_2^k$ program $\mathbf{B}_f^{k,j,n}$, also with OUT F, that computes

$$f_2^k \circ f_1^{j+p_1+2} \circ f_0^{2j+q_1+22}(\Sigma_{i=1}^n x_i + e),$$

which in turn $\geq \psi_n^{k,j}(e, \vec{x}_n)$, results in an $\mathbf{L}_2^k\langle\mathbf{L}_1\rangle$ program equivalent to $L(\mathbf{P})$. Then $\Psi_n(k, e, \vec{x}_n, \psi_n^{k,j}(e, \vec{x}_n))$ is computed by an $\mathbf{L}_2^k\langle\mathbf{L}_1\rangle \subseteq \mathbf{L}_2^{k+1}$ program of the form,

$\mathbf{B}_f^{k,j,n}$
$S \leftarrow F$
$T \leftarrow F$
$H \leftarrow 1$
$V \leftarrow 0$
LOOP S
  $\mathbf{P}^*$ [IN: $K, U, X_1, \ldots, X_n, T, V$]
  $Y \leftarrow V$
  $V \leftarrow V + 1$
END

where OUT Y, and assuming that the value of the input variable V in $\mathbf{P}^*$—which corresponds to the argument y in $\underline{\Psi}_n(k, e, \vec{x}_n, z, y)$—is restored at the end of the computation of $\mathbf{P}^*$ for a given set of inputs. Hence we may let

$$\Phi_n^{k,j}(b, \vec{x}_n) = \Psi_n(k, b, \vec{x}_n, \psi_n^{k,j}(b, \vec{x}_n)).$$

This completes the proof of Theorem 8.1.

Remark 8.2 One way to further "uniformize" Theorem 8.1 is to eliminate the dependence of the enumerating functions $\Phi_n^{k,j}$ on j. This comes at the price of "pushing" the enumerating functions for $\mathcal{L}_2^k$ "up" to $\mathcal{L}_2^{k+2}$. Note that, for $j = \text{lth}(\mathbf{P})$,

$$\begin{aligned}
\psi_n^{k,j}(e, \vec{x}_n) &\leq f_2^k \circ f_1^{j+p_1+2} \circ f_0^{2j+q_1+22}(\Sigma_{i=1}^n x_i + e) \leq \\
&\leq f_2^k \circ f_2((j + p_1 + 2) + f_0^{2j+q_1+22}(\Sigma_{i=1}^n x_i + e)) \leq \\
&\leq f_2^{k+1} \circ (f_0(j + p_1) + f_1(2j + q_1 + 22 + (\Sigma_{i=1}^n x_i + e)) \leq \\
&\leq f_2^{k+1}(2 f_1(\Sigma_{i=1}^n x_i + (p_1 + q_1) + 22 + e + 2j)) \leq \\
&\leq f_2^{k+1} \circ f_1^2(\Sigma_{i=1}^n x_i + (p_1 + q_1) + 22 + 2e) \leq \\
&\leq f_2^{k+1} \circ f_1^2 \circ f_0^{p_1+q_1+11}(\Sigma_{i=1}^n x_i + 2e) := \xi_n^k(e, \vec{x}_n)
\end{aligned}$$

since $\text{lth}(\mathbf{P}) \leq e$ if e is the index of $\mathbf{P}$. Then $g(\vec{x}_n) = \Psi_n(k, e, \vec{x}_n, \xi_n^k(e, \vec{x}_n))$ where $\xi_n^k \in \mathcal{L}_2^{k+1}$ and $\Xi_n^k(b, \vec{x}_n) := \Psi_n(k, b, \vec{x}_n, \xi_n^k(b, \vec{x}_n))$ is computed by an $\mathbf{L}_2^{k+1}\langle\mathbf{L}_1\rangle \subseteq \mathbf{L}_2^{k+2}$ program of the same form as above. Hence Ξ_n^k is the desired $\mathcal{L}_2^{k+2}$ enumeration of $\mathcal{L}_2^k$.

We may now invoke standard diagonalization arguments to derive some more information about how the classification $\{\mathbf{L}_2^k\}$ of ELPs determines the hierarchy of functions $\{\mathcal{L}_2^k\}$. Theorem 8.1 states that $\Phi_n^{k,j}$ enumerates n-place functions in $\mathcal{L}_2^{k,j}$

for $k \geq 2$. We claim that $\Phi_1^{k,j} \notin \mathcal{L}_2^{k,j}$ and so $\mathcal{L}_2^{k,j} \subset \mathcal{L}_2^{k+1}$ for $k \geq 2$. Note that since $\Phi_1^{k,j} \in \mathcal{L}_2^{k+1}$, then $\Theta^{k,j} \in \mathcal{L}_2^{k+1}$ where $\Theta^{k,j} := 1 \dot{-} \Phi_1^{k,j}(x, x)$, as $1 \dot{-} y$ is easily seen to be in $\mathcal{L}_1$. If $\Theta^{k,j} \in \mathcal{L}_2^{k,j}$ then $\Theta^{k,j}(x) = \Phi_1^{k,j}(e, x)$ for some e, and $\Phi_1^{k,j}(e, e) = \Theta^{k,j}(e) = 1 \dot{-} \Phi_1^{k,j}(e, e)$, a contradiction. On the other hand, a similar argument involving Ξ_1^k instead of $\Phi_1^{k,j}$ shows that for any $k \geq 2$, there are predicates, i.e., 0,1-functions, in $\mathcal{L}_2^{k+2}$ that are not contained in $\mathcal{L}_2^k$. (Note that the function,

$$sg(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$$

is in $\mathcal{L}_1$.)

Appendix Appendix 1 We make use of the work of Smullyan [18], pp. 77–87. It is easily seen from the proofs given there that the following predicates are strictly rudimentary and hence TOTAL:

1. $J(x, y, z)$, the graph of the pairing operation $(x, y)^{\#}$, and $K(z, x)$ and $L(z, y)$, the graphs of the associated projection operations; we sometimes refer to these functions directly by $J(x, y)$, $K(x)$ and $L(x)$, respectively;
2. $<, \leq, =$, the ordering relations between integers;
3. $\mathrm{Seq}(x)$, the set of sequence numbers; $x \in y$, which holds iff y is a sequence number and x is a term of the sequence it codes; and $x <_w y$, which holds just in case w is a sequence number, x and y are terms of the sequence coded, and x precedes y in that sequence.

We first define several predicates that code the Gödel numbers of arithmetical instructions that may occur in LOOP programs, at the same time showing that the predicates are strictly rudimentary and hence TOTAL:

$$\begin{aligned} Z(x) &:\Leftrightarrow (\exists y \lhd x)\, J(\overline{1}, y, x) \\ SC(x) &:\Leftrightarrow (\exists y \lhd x)\, J(\overline{2}, y, x) \\ A(x) &:\Leftrightarrow (\exists y_1, y_2, y_3 \lhd x)(J(y_1, y_2, y_3)\ \&\ J(\overline{3}, y_3, x)). \end{aligned}$$

Next, we state the condition under which an integer x codes an indexed series of length y:

$$\begin{aligned} \mathrm{SRS}(x, y) :\Leftrightarrow\ & \mathrm{Seq}(x)\ \&\ (\exists v_2, v \lhd x)(J(\overline{1}, v_2, v)\ \&\ v \in x)\ \& \\ & \&\ (\forall v_1, v_2, v \lhd x)(J(v_1, v_2, v)\ \&\ v \in x\ \&\ v_1 < y \rightarrow \\ & \qquad (\exists u_1, u_2, u \lhd x)(\underline{\mathrm{Plus}}(v_1, \overline{1}, u_1)\ \&\ J(u_1, u_2, u)\ \&\ u \in x))\ \& \\ & \&\ (\forall z \lhd x)(z \in x \rightarrow (\exists v_1 \lhd z)(\exists v_2 \lhd z)(J(v_1, v_2, z)\ \&\ y \geq v_1 \geq \overline{1}))\ \& \\ & \&\ (\forall v_1, v_2, v \lhd x)(\forall u_2, u \lhd x)(J(v_1, v_2, v)\ \&\ J(v_1, u_2, u)\ \& \\ & \qquad v \in x\ \&\ u \in x \rightarrow v_2 = u_2)\ \& \\ & \&\ (\forall v_1, v_2, v \lhd x)(\forall u_1, u_2, u \lhd x)(J(v_1, v_2, v)\ \&\ J(u_1, u_2, u)\ \& \\ & \qquad v \in x\ \&\ u \in x\ \&\ v_1 < u_1 \rightarrow v <_x u). \end{aligned}$$

(Throughout Appendices 1 and 2, $\underline{\mathrm{Plus}}(x_1, x_2, x_3)$ and $\underline{\mathrm{Times}}(x_1, x_2, x_3)$ are strictly rudimentary predicates that define the graphs of the addition and the multiplication

function, respectively. That such predicates exist was proved in [1].) We define the predicate "$\mathrm{SRS}_0(x, y)$" the same way except that in the second conjunct "$J(\overline{1}, v_2, v)$" is replaced by "$J(\overline{0}, v_2, v)$," and in the fourth conjunct "$v_1 \geq \overline{1}$" is replaced by "$v_1 \geq \overline{0}$". Again, these relations are strictly rudimentary and hence TOTAL.

We are now ready to define the predicate "x is the Gödel number of an $\mathbf{L}_0$ program $\mathbf{P}$ of length m and all variables in $\mathbf{P}$ are from $\underline{V}_n$":

$$\begin{aligned}\mathrm{LP}_0(x, m, n) :\Leftrightarrow\ & \mathrm{SRS}(x, m)\ \&\\ & \&\ (\forall y, v \lhd x)[y \in x\ \&\ L(y, v) \rightarrow (Z(v) \vee SC(v) \vee A(v))\ \&\\ & \&\ ((Z(v) \vee SC(v))\ \&\ (\exists v_2 \lhd v)(L(v, v_2)\ \&\ v_2 \leq n)) \vee\\ & \qquad \vee (A(v)\ \&\ (\exists v_1 \lhd v)(\exists v_2, v_3 \lhd v_1)(L(v, v_1)\ \&\\ & \qquad \&\ J(v_2, v_3, v_1)\ \&\ v_2 \leq n\ \&\ v_3 \leq n))].\end{aligned}$$

To describe loops, we define the predicates

$$\begin{aligned}\mathrm{LOOP}(x, m) &:\Leftrightarrow (\exists y_1, y_2 \lhd x)(J(m, y_1, y_2)\ \&\ J(\overline{4}, y_2, x))\\ \mathrm{END}(x, m) &:\Leftrightarrow J(\overline{5}, m, x)\end{aligned}$$

meaning "x is the Gödel number of a LOOP instruction of length m" and "x is the Gödel number of the end of a LOOP instruction of length m." (All of these predicates are TOTAL, for the same reason as above.) Then the Gödel numbers of $\langle \mathbf{L}_0 \rangle$ programs of length m with variables from $\underline{V}_n$ are defined by,[11]

$$\begin{aligned}\mathrm{LP}_{\langle 0\rangle}(x, m, n) :\Leftrightarrow\ & \mathrm{LP}_0(x, m, n) \vee (\exists y \ll x)(\exists m_1 \ll m)(\exists k \ll n)[\mathrm{SRS}(x, m)\ \&\\ & \&\ \mathrm{SRS}(y, m_1)\ \&\ \mathrm{LP}_{\langle 0\rangle}(y, m_1, k)\ \&\ k \leq n\ \&\ (\exists z \lhd x)(\mathrm{LOOP}(z, m)\ \&\\ & \&\ LL(z) \leq n\ \&\ J(\overline{1}, z) \in x)\ \&\ (\forall z \lhd y)(z \in y \rightarrow (\exists u \lhd z)(\underline{\mathrm{Plus}}(K(z), \overline{1}, u)\ \&\\ & \&\ J(u, L(z)) \in x))\ \&\ J(m, J(\overline{5}, m)) \in x].\end{aligned}$$

The complement relation $\neg\mathrm{LP}_{\langle 0\rangle}(x, m, n)$ is also PRUD:

$$\begin{aligned}\neg\mathrm{LP}_{\langle 0\rangle}(x, m, n) :\Leftrightarrow\ & \neg\mathrm{LP}_0(x, m, n)\ \&\ (\neg\mathrm{SRS}(x, m) \vee (\mathrm{SRS}(x, m)\ \&\\ & \&\ ((\forall z \lhd x)(J(\overline{1}, z) \in x \rightarrow \neg\mathrm{LOOP}(z, m) \vee LL(z) \geq n) \vee\\ & \vee J(m, J(\overline{5}, m)) \notin x \vee (\exists z \ll x)(\exists k \ll m)(\underline{\mathrm{Plus}}(k, \overline{2}, m)\ \&\ \mathrm{SRS}(z, k)\ \&\\ & \&\ (\forall v_1, v_2 \lhd x)(\forall u \lhd x)(J(v_1, v_2) \in x\ \&\ \overline{2} \leq v_1 < m\ \&\ \underline{\mathrm{Plus}}(u, \overline{1}, v_1) \rightarrow\\ & \rightarrow J(u, v_2) \in z)\ \&\ \neg\mathrm{LP}_0(z, k, n))))).\end{aligned}$$

We proceed to define the Gödel numbers of composite LOOP programs. First we need the predicate $\mathrm{CONC}(x, x_1, x_2, m)$ meaning "indexed series with the Gödel number x of length m is the concatenation of the two indexed series with the Gödel numbers x_1 and x_2, respectively, in that order":

$$\begin{aligned}\mathrm{CONC}(x, x_1, x_2, m) :\Leftrightarrow\ & (\exists u_1 \lhd x_1)(\exists u_2 \lhd x_2)[\mathrm{SRS}(x_1, u_1)\ \&\ \mathrm{SRS}(x_2, u_2)\ \&\\ & \&\ \underline{\mathrm{Plus}}(u_1, u_2, m)\ \&\ \mathrm{SRS}(x, m)\ \&\ (\forall y \lhd x_1)(y \in x_1 \rightarrow y \in x)\ \&\\ & \&\ (\forall y \lhd x_2)(y \in x_2 \rightarrow (\exists y_1, y_2 \lhd y)(\exists v_1 \lhd x)(J(y_1, y_2, y)\ \&\\ & \&\ \underline{\mathrm{Plus}}(u_1, y_1, v_1)\ \&\ u_1 \leq n\ \&\ u_2 \leq n\ \&\ J(v_1, y_2) \in x)].\end{aligned}$$

This is a strictly rudimentary predicate and hence TOTAL. We then set

$$\begin{aligned}\mathrm{LP}_{\langle 0\rangle 0}(x,m,n) :\Leftrightarrow\ & (\exists x_1, x_2 \ll x)(\exists m_1, m_2 \ll m)(\exists u_1, u_2 \ll n)\\ & (\mathrm{SRS}(x_1,m_1)\ \&\ \mathrm{SRS}(x_2,m_2)\ \&\ \mathrm{LP}_{\langle 0\rangle}(x_1,m_1,u_1)\ \&\\ & \&\ \mathrm{LP}_0(x_2,m_2,u_2)\ \&\ \mathrm{CONC}(x,x_1,x_2,m)\ \&\ \underline{\mathrm{Plus}}(m_1,m_2,m)),\end{aligned}$$

which defines the set of Gödel numbers of $\langle \mathbf{L}_0\rangle \mathbf{L}_0$ programs of length m with variables from $\underline{V}_n$. To define the complement of this set, it is convenient to introduce the following abbreviation. Given some 3-place predicate $\varphi(x,y,z)$, let $U(x,m,k,n,\varphi)$ stand for

$$\begin{aligned}& (\exists u \ll x)(\exists p \ll m)(\underline{\mathrm{Plus}}(k,p,m)\ \&\ \mathrm{SRS}(u,p)\ \&\\ & \&\ (\forall v_1, v_2 \lhd x)(\exists q \ll m)(J(v_1,v_2)\in x\ \&\ k < v_1 \le m \rightarrow\\ & \rightarrow \underline{\mathrm{Plus}}(q,k,v_1)\ \&\ J(q,v_2)\in u\ \&\ \neg\varphi(u,p,n)].\end{aligned}$$

Then

$$\begin{aligned}\neg\mathrm{LP}_{\langle 0\rangle 0}(x,m,n) :\Leftrightarrow\ & \neg\mathrm{SRS}(x,m)\vee(\mathrm{SRS}(x,m)\ \&\\ & \&\ (\forall z,k\lhd x)(\mathrm{SRS}(z,k)\ \&\ k<m\rightarrow(\neg\mathrm{LP}_{\langle 0\rangle}(z,k,n)\vee(\mathrm{LP}_{\langle 0\rangle}(z,k,n)\ \&\\ & \&\ U(x,m,k,n,\mathrm{LP}_0))))).\end{aligned}$$

We let

$$\begin{aligned}\mathrm{LP}_1(k,x,n) :\Leftrightarrow\ & \mathrm{SRS}_0(x,k)\ \&\ (\exists z\lhd x)(\exists m\lhd z)(J(\overline{0},z)\in x\ \&\ \mathrm{LP}_0(z,m,n))\ \&\\ & \&\ (\forall i\lhd x)(\overline{0}<i\le k\rightarrow(\exists u\lhd x)(\exists m\lhd u)(J(i,u)\in x\ \&\ \mathrm{LP}_{\langle 0\rangle 0}(u,m,n))),\end{aligned}$$

and

$$\begin{aligned}\neg\mathrm{LP}_1(k,x,n) :\Leftrightarrow\ & \neg\mathrm{SRS}_0(x,k)\vee(\mathrm{SRS}_0(x,k)\ \&\ (\exists z\lhd x)((J(\overline{0},z)\in x\ \&\\ & \&\ (\forall m\lhd z)\neg\mathrm{LP}_0(z,m,n))\vee(\exists i\lhd z)(\overline{0}<i\le k\ \&\ J(i,z)\in x\ \&\\ & \&\ (\forall m\lhd z)\neg\mathrm{LP}_{\langle 0\rangle 0}(z,m,n)))).\end{aligned}$$

Then $\mathrm{LP}_1(k,x,n)$ holds just in case x is the Gödel number of an $\mathbf{L}_2^k$ program all of whose variables are in $\underline{V}_n$. To obtain a PRUD definition of $\mathrm{LP}_{\langle 1\rangle}(x,m,n)$ and $\neg\mathrm{LP}_{\langle 1\rangle}(x,m,n)$, we replace the predicate LP_0 in the definitions of $\mathrm{LP}_{\langle 0\rangle}$, and $\neg\mathrm{LP}_{\langle 0\rangle}$, by $(\exists k\lhd\ )\mathrm{LP}_1$. Analogous remarks apply to the definitions of $\mathrm{LP}_{\langle 1\rangle 1}(x,m,n)$ and $\neg\mathrm{LP}_{\langle 1\rangle 1}(x,m,n)$. Thus $\mathrm{LP}_{\langle 1\rangle 1}$ defines the set of Gödel numbers of $\langle \mathbf{L}_1\rangle\mathbf{L}_1$ programs. PRUD definitions of $\mathrm{LP}_2(k,x,n)$ and $\neg\mathrm{LP}_2(k,x,n)$ are obtained in the same way as the definitions of $\mathrm{LP}_1(k,x,n)$ and $\neg\mathrm{LP}_1(k,x,n)$, the sole difference being that LP_0 and $\mathrm{LP}_{\langle 0\rangle 0}$ are replaced by LP_1 and $\mathrm{LP}_{\langle 1\rangle 1}$.

Appendix Appendix 2 We now proceed to arithmetize computations on LOOP programs. We first set

$$\begin{aligned}\mathrm{ID}(x,m,n) :\Leftrightarrow\ & (\exists i\ll m)(\exists j,k,q\lhd x)(\exists v_1,v_2,v_3\lhd x)(\underline{\mathrm{Plus}}(i,\overline{1},q)\ \&\\ & \&\ J(j,k,v_1)\ \&\ J(q,v_1,v_2)\ \&\ J(v_2,v_3,x)\ \&\ \mathrm{SRS}_0(v_3,n)),\end{aligned}$$

meaning "x is the Gödel number of an instantaneous description (i.d.) for a program coded by an indexed series of length m and with variables from $\underline{V}_n$." Then ID is strictly rudimentary and so TOTAL. Also, let

$$\mathrm{Fin}(x, m) :\Leftrightarrow (\exists n, u, v \ll x)(\mathrm{ID}(x, m, n) \,\&\, KK(x) = u \,\&\, \underline{\mathrm{Plus}}(m, \overline{1}, u))$$

define "x is a final i.d. for a program coded by an indexed series of length m."

We define several auxiliary predicates that will help us describe the process of computation step-by-step. Let

$$\begin{aligned}
&Y_1(x_1, x_2, n, y) :\Leftrightarrow (\forall j, v \lhd x_1)[j \le n \,\&\, J(j, v) \in L(x_1) \to \\
&\quad \to ((j \ne L(y) \to J(j, v) \in L(x_2)) \,\&\, (j = L(y) \to J(j, \overline{0}) \in L(x_2)))]; \\
&Y_2(x_1, x_2, n, y) :\Leftrightarrow (\forall j, v \lhd x_1)[j \le n \,\&\, J(j, v) \in L(x_1) \to \\
&\quad \to ((j \ne L(y) \to J(j, v) \in L(x_2)) \,\&\, (j = L(y) \to \\
&\quad \to (\exists u \lhd x_2)(\underline{\mathrm{Plus}}(v, \overline{1}, u) \,\&\, J(j, u) \in L(x_2))))]; \\
&Y_3(x_1, x_2, n, y) :\Leftrightarrow (\forall j, v \lhd x_1)[j \le n \,\&\, J(j, v) \in L(x_1) \to ((j \ne KL(y) \to \\
&\quad \to J(j, v) \in L(x_2)) \,\&\, (j = KL(y) \to (\forall w \lhd x_1)(J(KK(y), w) \in L(x_1) \to \\
&\quad \to J(j, w) \in L(x_2)))))]; \\
&Y_4(x_1, x_2, n, i, k, y) :\Leftrightarrow (\forall v \lhd x_1)[J(LL(y), v) \in x_1 \to \\
&\quad \to ((v = 0 \to (\exists v_1, v_2 \lhd x_2)(\underline{\mathrm{Plus}}(i, k, v_1) \,\&\, \underline{\mathrm{Plus}}(v, \overline{1}, v_2) \,\& \\
&\quad \& KK(x_2) = v_2)) \,\&\, (v \ne 0 \to (\exists v_1 \lhd x_2)(\underline{\mathrm{Plus}}(i, \overline{1}, v_1) \,\&\, KK(x_2) = v_1))] \,\& \\
&\quad \& (\forall j, v \lhd x_1)[j \le n \,\&\, J(j, v) \in L(x_1) \to J(j, v) \in L(x_2)]; \\
&Y_5(x_1, x_2, z, k) :\Leftrightarrow (\forall w \lhd z)(\forall q, v \lhd x_1)(J(k, w) \in z \,\&\, J(q, v) \in L(x_1) \to \\
&\quad \to (q \ne LL(w) \to J(q, v) \in L(x_2)) \,\&\, (q = LL(w) \to \\
&\quad \to (\exists v_1 \ll v)[\underline{\mathrm{Plus}}(v_1, \overline{1}, v) \,\&\, J(LL(w), v_1) \in L(x_2))))].
\end{aligned}$$

This enables us to define the PRUD relation,

$$\begin{aligned}
Y^*(z, x_1, x_2, n, y, k_1, k_2) :\Leftrightarrow\ & ((Z(y) \,\&\, \underline{\mathrm{Plus}}(k_1, \overline{1}, k_2) \,\&\, Y_1(x_1, x_2, n, y)) \vee \\
& \vee (SC(y) \,\&\, \underline{\mathrm{Plus}}(k_1, \overline{1}, k_2) \,\&\, Y_2(x_1, x_2, n, y)) \vee \\
& \vee (A(y) \,\&\, \underline{\mathrm{Plus}}(k_1, \overline{1}, k_2) \,\&\, Y_3(x_1, x_2, n, y)) \vee \\
& \vee (\forall p \lhd z)[(\mathrm{LOOP}(y, p) \to Y_4(x_1, x_2, n, i, p, y)) \,\& \\
& \& (\mathrm{END}(y, p) \to (\exists k_3 \ll x_2)(\underline{\mathrm{Plus}}(k_3, p, k_1) \,\& \\
& \& \underline{\mathrm{Plus}}(k_3, \overline{1}, k_2) \to Y_5(x_1, x_2, z, k_2)))).
\end{aligned}$$

We let $\mathrm{Yield}(z, x_1, x_2)$ abbreviate:

$$\begin{aligned}
&(\exists m \lhd z)(\exists n \lhd x_1)(\exists i_1, j_1, k_1 \lhd x_1)(\exists i_2, j_2, k_2 \lhd x_2)(\exists m' \ll m + 1)[\mathrm{SRS}_0(z, m) \,\& \\
&\quad \& \mathrm{ID}(x_1, m, n) \,\& \mathrm{ID}(x_2, m, n) \,\& K(x_1) = J(i_1, J(j_1, k_1)) \,\& \\
&\quad \& K(x_2) = J(i_2, J(j_2, k_2)) \,\& \underline{\mathrm{Plus}}(m, \overline{1}, m') \,\& \overline{0} \le i_1 \le m \,\& \overline{0} \le i_2 \le m' \,\& \\
&\quad \& (\exists y \lhd z)(\exists r \lhd y)(\exists u \lhd y)(\exists q \lhd u)((i_1 = \overline{0} \to \mathrm{LP}_1(r, y, n)) \,\& \\
&\quad \& A(y, i_1, j_1, u, r, q, m, n) \,\& (\overline{1} \le k_1 \le q \to Y^*(y, x_1, x_2, n, u, k_1, k_2) \,\& \\
&\quad \& j_1 = j_2 \,\& i_1 = i_2) \,\& (j_1 < r \,\& k_1 = q \to (\exists v \lhd y)(\exists q_1 \lhd v)(J(j_2, v) \in y \,\&
\end{aligned}$$

$$\begin{aligned}
&\& \mathrm{LP}_{\langle 0\rangle 0}(v, q_1, n) \,\&\, k_2 = \overline{1} \,\&\, \underline{\mathrm{Plus}}(j_1, \overline{1}, j_2) \,\&\, i_1 = i_2 \,\& \\
&\& L(x_1) = L(x_2))) \,\&\, (i_1 < m \,\&\, j_1 = r \,\& \\
&\& k_1 = q \to (\exists w \lhd z)(\exists s \lhd w)(\exists t \lhd s)(\exists q_2 \lhd t)(J(i_2, w) \in z \,\& \\
&\& \underline{\mathrm{Plus}}(i_1, \overline{1}, i_2) \,\&\, j_2 = \overline{0} \,\&\, k_2 = \overline{1} \,\&\, L(x_1) = L(x_2) \,\& \\
&\& A(w, i_2, j_2, t, s, q_2, m, n))) \,\&\, (i_1 = m \,\&\, j_1 = r \,\& \\
&\& k_1 = q \to i_2 = m' \,\&\, j_2 = \overline{0} \,\&\, k_2 = \overline{1} \,\&\, L(x_1) = L(x_2)))]
\end{aligned}$$

where

$$\begin{aligned}
A(y, i, j, u, r, q, m, n) \quad :\Leftrightarrow \quad & (\overline{0} < i \leq m \to \mathrm{LP}_{\langle 1\rangle 1}(r, y, n)) \,\&\, (j = \overline{0} \to \\
\to \quad & \mathrm{LP}_0(u, q, n)) \,\&\, (j > \overline{0} \to \mathrm{LP}_{\langle 0\rangle 0}(u, q, n)).
\end{aligned}$$

Then $\mathrm{Yield}(z, x_1, x_2)$ holds iff the i.d. with Gödel number x_1 yields the i.d. with Gödel number x_2 according to the $\mathbf{L}_2^k$ program with Gödel number z. Note that whenever z is the Gödel number of some $\mathbf{L}_2^k$ program and x_1 is the Gödel number of an appropriate i.d., then an x_2 such that $\mathrm{Yield}(z, x_1, x_2)$ is uniquely determined.

For each $n \geq 1$, we let $\mathrm{Init}_n(\vec{x}_n, z, y)$ abbreviate:

$$\begin{aligned}
&(\exists y_2 \lhd y)(K(y) = J(\overline{0}, J(\overline{0}, \overline{1})) \,\&\, L(y) = y_2 \,\&\, \mathrm{SRS}_0(y_2, z) \,\& \\
&\& (\exists z_1 \ll z)(\overline{2 \cdot n} \leq z_1 \leq z \,\&\, \overline{n} \geq \overline{1} \,\&\, (\forall j, v \lhd y_2)(\forall i \lhd z)(J(j, v) \in y_2 \to \\
&\to i < \overline{n} \to (\exists i_1, i_2 \ll z)(\underline{\mathrm{Plus}}(i, \overline{1}, i_1) \,\&\, \underline{\mathrm{Times}}(\overline{2}, i_1, i_2) \,\& \\
&\& (j \neq i_2 \to v = 0)) \,\&\, J(\overline{2}, x_1) \in y_2 \,\&\, J(\overline{4}, x_2) \in y_2 \,\& \ldots \\
&\& J(\overline{2 \cdot n}, x_n) \in y_2))).
\end{aligned}$$

Then $\mathrm{Init}_n(\vec{x}_n, z, y)$ holds just in case y is the Gödel number of the i.d. that corresponds to the initial state of the register machine in which the variables $V_2, V_4, \ldots, V_{2n}$ are assigned the values $x_1, \ldots, x_n$, respectively, and all the other variables V_j $(0 \leq j < z)$ are assigned the value 0. Clearly, $\mathrm{Init}_n(\vec{x}_n, z, y)$ is a PRUD graph of a function $\mathrm{Init}_n^*(\vec{x}_n, z)$.

We let

$$\begin{aligned}
R(k, z, y, u) :\Leftrightarrow\ & (\exists m, n \lhd z)(\mathrm{LP}_2(k, z, n) \,\&\, k > \overline{0} \,\&\, \mathrm{SRS}_0(z, m) \,\& \\
& \& \mathrm{ID}(y, m, n) \,\&\, \mathrm{Yield}(z, y, u)) \vee ((k = \overline{0} \vee (\forall m, n \lhd z)\neg \mathrm{ID}(y, m, n) \vee \\
& \vee (\forall n \lhd z)\neg \mathrm{LP}_2(k, z, n)) \,\&\, u = \overline{0}).
\end{aligned}$$

Again, we note that for any k, z, and y, an integer u such that $R(k, z, y, u)$ is uniquely determined and $= \mu u\, R(k, z, y, u)$. Hence the latter function has a PRUD graph. We introduce the function H by the following recursion:

$$\begin{aligned}
H(y, \vec{x}_n, k, m, 0) \quad &:= \quad J^*(k + 1, \mathrm{Init}_n^*(\vec{x}_n, m)) \\
H(y, \vec{x}_n, k, m, i + 1) \quad &:= \quad J^*(H(y, \vec{x}_n, k, m, i), \mu u\, R(k, y, L^*(H(y, \vec{x}_n, k, m, i)), u)
\end{aligned}$$

If we let,

$$G(y, \vec{x}_n, k, m) := J^*(U_{n+2}^{n+3}(y, \vec{x}_n, k, m) + 1, \mathrm{Init}_n^*(\vec{x}_n, m))$$

and

$$F(y, \vec{x}_n, k, m, i, z) := J^*(U^{n+5}_{n+5}(y, \vec{x}_n, k, m, i, z), \mu u\, R(k, y, L^*(z), u)$$

then

$$H(y, \vec{x}_n, k, m, 0) = G(y, \vec{x}_n, k, m)$$

and

$$H(y, \vec{x}_n, k, m, i+1) = F(y, \vec{x}_n, k, m, i, H(y, \vec{x}_n, k, m, i)).$$

It is easily seen that the function J^* has a PRUD graph given that $+$ and $\times$ do, and the same applies to the associated projection functions K^* and L^*. Since J^* is increasing, it is clear that the compositions G and F have PRUD graphs. To conclude that H has a PRUD graph, it remains only to verify that H satisfies the conditions of Theorem 4.2.

Note that $0 < G(y, \vec{x}_n, k, m) = H(y, \vec{x}_n, k, m, 0)$. Furthermore, assuming that $i < H(y, \vec{x}_n, k, m, i)$, we have that,

$$0 \leq i < H(y, \vec{x}_n, k, m, i) = K^*(H(y, \vec{x}_n, k, m, i+1)) < H(y, \vec{x}_n, k, m, i+1),$$

and so $i+1 < H(y, \vec{x}_n, k, m, i+1)$. It follows that,

$$i < H(y, \vec{x}_n, k, m, i) \tag{1}$$

for all $i, y, \vec{x}_n, k, m$. On the other hand, from the definitions of F and J^*, we immediately have that,

$$\begin{aligned}(H(y, \vec{x}_n, k, m, i))^2 \leq F(y, \vec{x}_n, k, m, i, H(y, \vec{x}_n, k, m, i)) = \\ = H(y, \vec{x}_n, k, m, i+1).\end{aligned} \tag{2}$$

Let H^+ be a PRUD graph of the function H, and let

$$\begin{aligned}T_n(k, y, \vec{x}_n, i, u) \Leftrightarrow (\exists y_1, y_2, y_3 \leq y)[J^*(y_1, J^*(y_2, y_3)) = y \,\&\, \mathrm{LP}_2(k, y_1, y_3) \,\&\\ \&\, y_3 \geq \overline{2 \cdot n} \,\&\, H^+(y, \vec{x}_n, k, y_3, i, u) \,\&\, \mathrm{Fin}(L^*(u), y_2)].\end{aligned}$$

Then we let,

$$\mathrm{OUTPUT}(z, y) \Leftrightarrow (\exists z_1 \leq z)(L(z, z_1) \,\&\, J(\overline{0}, y) \in z_1).$$

Appendix Appendix 3

Proof of Lemma 6.1: (a) is proved by induction on n. The case $n = 2$ is immediate by the definition of dyadic concatenation. From that definition and the induction hypothesis we have that,

$$\begin{aligned}a_1 * \cdots * a_n * a_{n+1} &= a(2^{(n-1)l(a)} + \cdots + 2^{l(a)} + 1) \cdot 2^{l(a)} + a = \\ &= a(2^{n \cdot l(a)} + 2^{(n-1)l(a)} + \cdots + 2^{l(a)}) + a \\ &= a(2^{n \cdot l(a)} + 2^{(n-1)l(a)} + \cdots + 2^{l(a)}) + 1),\end{aligned}$$

provided $a_{n+1} = a$.

For (b), let us first abbreviate a concatenation $a_0 * a_1 * \cdots * a_n$ by "$a_0 a_1 \ldots a_n$." A *tally* is a string of 1s. The sequence number $(a_1, \ldots, a_n)^{\#}$ is of the form $\mathring{\mathrm{v}} a_1 \mathring{\mathrm{v}} a_2 \mathring{\mathrm{v}} \ldots \mathring{\mathrm{v}} a_n \mathring{\mathrm{v}}$ where $\mathring{\mathrm{v}} = 2v2$ and v is the smallest tally that is not a part of any a_i. Then $(a_1, \ldots, a_n)^{\#} \leq (y_1, \ldots, y_n)^{\#}$, where $y_i = y$ for each i, $1 \leq i \leq n$, and the sequence number $(y_1, \ldots, y_n)^{\#}$ is of the form $\mathring{\mathrm{w}} y_1 \mathring{\mathrm{w}} y_2 \mathring{\mathrm{w}} \ldots \mathring{\mathrm{w}} y_n \mathring{\mathrm{w}}$ where w is the smallest tally that is not a part of y. Then $w < 2(2^{l(y)+1} - 1)$ and $\mathring{\mathrm{w}} < 2(2^{l(y)+3} - 1)$. Since $2^{l(y)} - 1 \leq y \leq 2(2^{l(y)} - 1)$, it follows that $\mathring{\mathrm{w}} < 2^5 \cdot y$, and so,

$$(a_1, \ldots, a_n)^{\#} < z_1 * \cdots * z_{2n+1},$$

where $z_i = 2^5 \cdot y$ for each i, $1 \leq i \leq 2n+1$. But then, by (a), if we let $\alpha(y) = 2^5 \cdot y$, we have that,

$$\begin{aligned}(a_1, \ldots, a_n)^{\#} &< \alpha(y) \cdot 2^{2n \cdot l(\alpha(y))} + 2^{(2n-1) \cdot l(\alpha(y))} + \cdots + 2^{l(\alpha(y))} + 1) < \\ &< \alpha(y) \cdot (2^{n \cdot (2n+1) \cdot l(\alpha(y))} + 1) = (2^5 \cdot y)(2^{r \cdot l(2^5 \cdot y)} + 1) < (2^6 \cdot y) \cdot 2^{r \cdot l(2^5 \cdot y)},\end{aligned}$$

where $r := n(2n+1)$. Since $2^{l(2^5 \cdot y)} \leq 2^5 \cdot y + 1$, we then derive

$$(a_1, \ldots, a_n)^{\#} < (2^6 \cdot y)(2^5 \cdot y + 1)^r < (2^6 \cdot y)(2^6 \cdot y)^r = (2^6 \cdot y)^{r+1}$$

as required.

NOTES

1. Here
$$x \dot{-} y := \begin{cases} x - y & \text{if} \quad y \leq x \\ 0 & \text{if} \quad y > x \end{cases};$$
the last two operations are usually defined by primitive recursion:
$$\Sigma_{y<0} f(\vec{x}_n, y) := 0 \qquad \Sigma_{y<z+1} f(\vec{x}_n, y) := \Sigma_{y<z} f(\vec{x}_n, y) + f(\vec{x}_n, z)$$
and
$$\Pi_{y<0} f(\vec{x}_n, y) := 1 \qquad \Pi_{y<z+1} f(\vec{x}_n, y) := \Pi_{y<z} f(\vec{x}_n, y) \cdot f(\vec{x}_n, z).$$
(The sign ":=" means that the identity in question holds by definition.) In general, the elementary functions are not closed under primitive recursion, and thus they form a proper subclass of the primitive recursive (p.r.) functions: e.g., superexponentiation (i.e., iterated exponential) is not elementary.

2. A different hierarchy of elementary functions based on an idea similar to Ritchie's but using register machine programs (see below) instead of Turing machines was given by Cleave [4]. These and other hierarchies of elementary functions are compared in Herman [11]. For textbook treatments of elementary functions see Rose [17], Brainerd and Landweber [3], or Cutland [6].

3. See, e.g., [3], pp. 269–270. Whether or not this is the case, it would be a mistake to conclude that the Church-Turing model of computability should therefore be abandoned as obsolete. An unacceptable price would have to be paid in terms of efficiency. A fundamental theorem due to Blum implies that for any programming formalism restricted to, say, elementary functions, there are functions for which the shortest program in such a restricted formalism is simply too lengthy and runs far too long in comparison to programs for computing the same function formulated in a formalism designed to express arbitrarily complex computation procedures. For a comprehensive survey of various other approaches to bounded computability see [17].

4. In Jeffrey [12], chapters 7,8, there is a detailed informal treatment of elements of the theory of computability cast in terms of RM programs. A formally precise description of essentially the same language, G_3, is given, e.g., in Constable and Borodin [5], §2. In Boolos and Jeffrey [2], chapters 6–8, it is proved that the functions computable by RM programs are precisely the Turing computable functions.

5. It is easily seen that every LOOP program is equivalent to one in which the variables V_i in the LOOP commands occur neither in the programs **P** to which the LOOP commands apply nor in any of the subsequent instructions in **P**. (Two programs are *equivalent* if they compute the same function.)

6. This was first established by Meyer and Ritchie [13], who extended the result to show that the functions computable by LOOP programs with maximum depth of nesting $\leq n$ are precisely the functions in the $n+1$st class $\mathcal{E}^{n+1}$ of the Grzegorczyk hierarchy of p.r. functions, for $n \geq 2$. (See[17] for more information about the latter hierarchy.) The elementary functions form the third class, $\mathcal{E}^3$, of the Grzegorczyk hierarchy.

7. "$f \circ g$" stands for the composition $f(g(x))$ of f and g, and similarly "$f_1 \circ \cdots \circ f_n$" stands for $f_1(f_2(\ldots(f_n(x))\ldots))$.

8. Since we are interested primarily in sets and relations of nonnegative integers, we shall systematically interpret the numeral $\overline{y}$ as the numeral $\overline{y-1}$ for $y-1$. Thus, e.g., "1" is $\overline{0}$, the numeral for 0, and "2" is $\overline{1}$.

9. Bennett's view of the significance of this result is that it establishes a strong "isomorphism" between the theory of concatenation of strings of symbols on the one hand, and the theory of integers on the other, at the level of "finite theories." Quine [15] originally showed that such a relation obtains at the level of "infinite theories" with unbounded quantifiers allowed.

10. Proskurin [14] proved that the majorizing functions f_n, $n \geq 0$, that determine the Grzegorczyk Hierarchy of p.r. functions all have PRUD graphs, and that the same is true of even the Ackermann function, which is known to be non-p.r.

11. Here we use some obvious abbreviations when dealing with the pairing operations J, K and L: e.g.,
$$KL(x) = y :\Leftrightarrow (\exists z \lhd x)(L(x,z) \,\&\, K(z,y)).$$

REFERENCES

[1] Bennett, J. H., "On spectra," Ph.D. thesis, Princeton University, 1962. 4, 4, 4, 4, 8

[2] Boolos, G. S. and R. C. Jeffrey, *Computability and Logic*, Third Edition, Cambridge University Press, Cambridge, 1989. Zbl 0708.03001 MR 90h:03001 8

[3] Brainerd, W. S. and L. H. Landweber, *Theory of Computation*, Wiley, New York, 1974. MR 53:4590 8, 8

[4] Cleave, J. P., "A hierarchy of primitive recursive functions," *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, vol. 9 (1963), pp. 331–345. Zbl 0224.02031 MR 28:2970 8

[5] Constable, R. L. and A. B. Borodin, "Subrecursive programming languages, part I: Efficiency and program structure," *Journal of the Association for Computing Machinery*, vol. 19 (1972), pp. 526–568. Zbl 0259.68036 7, 7, 8

[6] Cutland, N. J., *Computability: An Introduction to Recursive Function Theory*, Cambridge University Press, Cambridge, 1980. Zbl 0448.03029 MR 81i:03001 1, 8

[7] Damnjanovic, Z., "Elementary realizability," forthcoming in *Journal of Philosophical Logic*. Zbl 0874.03067 MR 98j:03088 1

[8] Davis, M. D. and E. J. Weyuker, *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*, Academic Press, New York, 1983. Zbl 0569.68042 MR 86b:03001 1, 2, 2, 3, 6

[9] Goetze, B. and W. Nehrlich, "The structure of loop programs and subrecursive hierarchies," *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, vol. 26 (1980), pp. 255–278. Zbl 0439.03018 MR 81j:03066 1, 1, 3, 3

[10] Harrow, K., "The bounded arithmetic hierarchy," *Information and Control*, vol. 36 (1978), pp. 102–117. Zbl 0374.02019 MR 57:16010 4

[11] Herman, G. T., "The equivalence of different hierarchies of elementary functions," *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, vol. 17 (1971), pp. 219–224. Zbl 0222.02043 MR 44:6494 8

[12] Jeffrey, R. C., *Formal Logic: Its Scope and Limits*, Third Edition, McGraw Hill, New York, 1991. Zbl 0925.03002 8

[13] Meyer, A. R. and D. M. Ritchie, "The complexity of loop programs," pp. 465–469 in *Proceedings of the 22nd National Conference of the Association for Computing Machinery*, Thompson, Washington, D.C., 1967. 2, 2, 3, 7, 8

[14] Proskurin, A. V., "Positive rudimentarity of the graphs of Ackermann and Grzegorczyk," (in Russian), *Journal of Soviet Mathematics*, vol. 20 (1982), pp. 2363–2366. Zbl 0493.03017 MR 81b:03044 4, 8

[15] Quine, W. V. O., "Concatenation as a basis for arithmetic," *Journal of Symbolic Logic*, vol. 11 (1946), pp. 105-114. Zbl 0063.06362 MR 8,307b 8

[16] Ritchie, R. W., "Classes of predictably computable functions," *Transactions of the American Mathematical Society*, vol. 106 (1963), pp. l39-173. Zbl 0107.01001 MR 28:2045 1, 4

[17] Rose, H. E., *Subrecursion: Functions and Hierarchies*, Clarendon Press, Oxford, 1984. Zbl 0539.03018 MR 86g:03004 8, 8, 8

[18] Smullyan, R. M., *Theory of Formal Systems*, Annals of Mathematics Studies, vol. 47, Princeton University Press, Princeton, 1961. Zbl 0097.24503 MR 22:12042 4, 4, 4, 6, 8

School of Philosophy
University of Southern California
3709 Trousdale Parkway
Los Angeles, CA 90089-0451